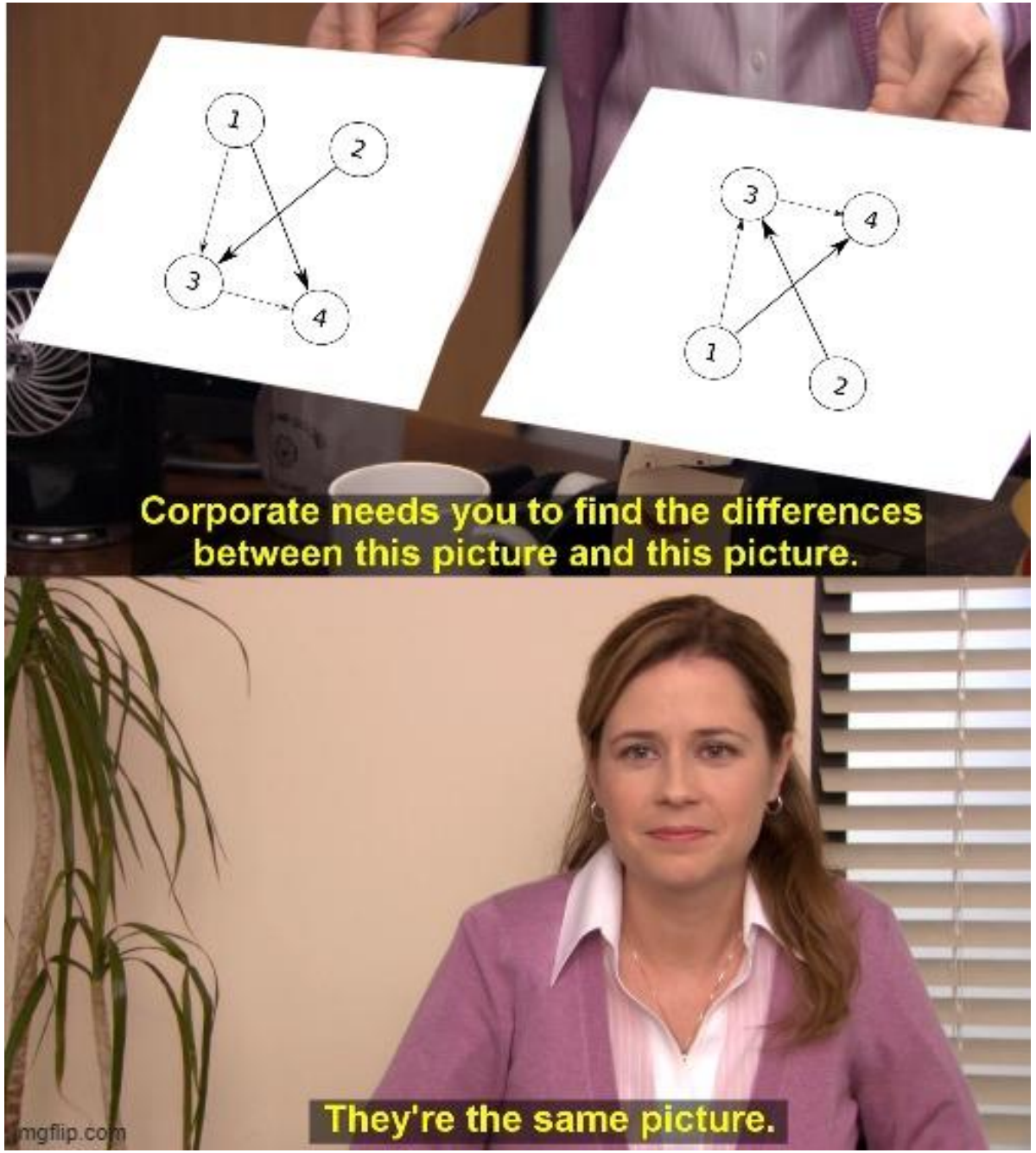


Dependency Parsing II

CSC485/2501

Lecture 5

Based on slides by Roger Levy, Yuji Matsumoto, Dragomir Radev, Dan Roth, David Smith and Jason Eisner

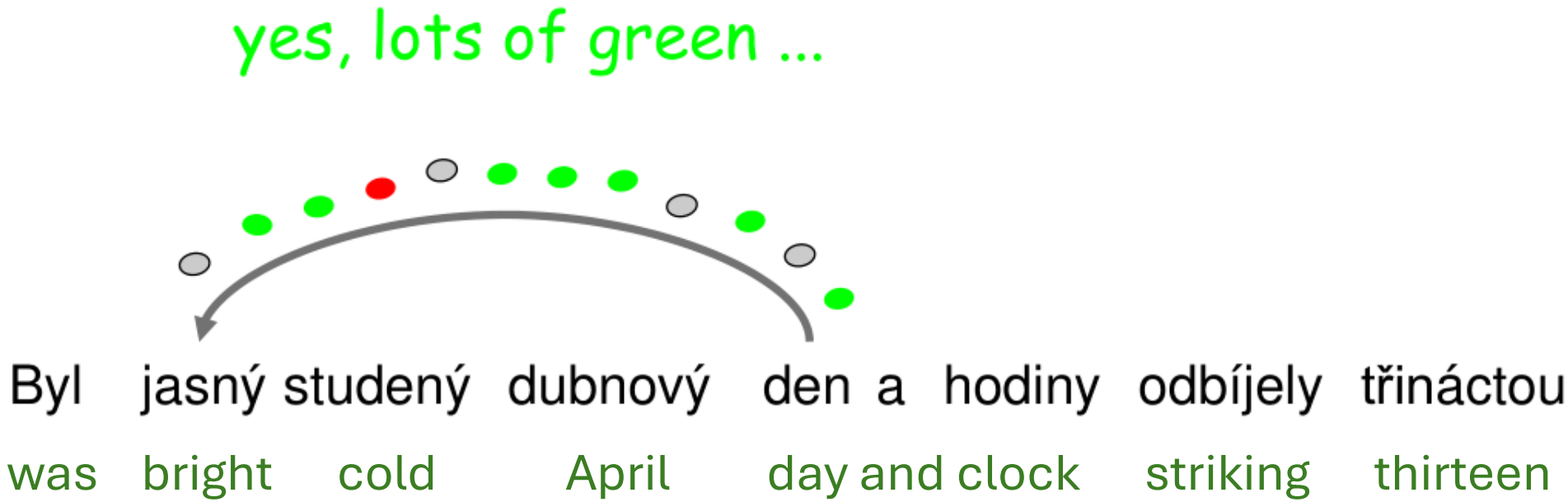
Edge-Factored Parsers

- Log-linear models great for n-way classification.
- Also good for predicting sequences.

- Also good for dependency parsing

Edge-Factored Parsers (McDonald et al. 2005)

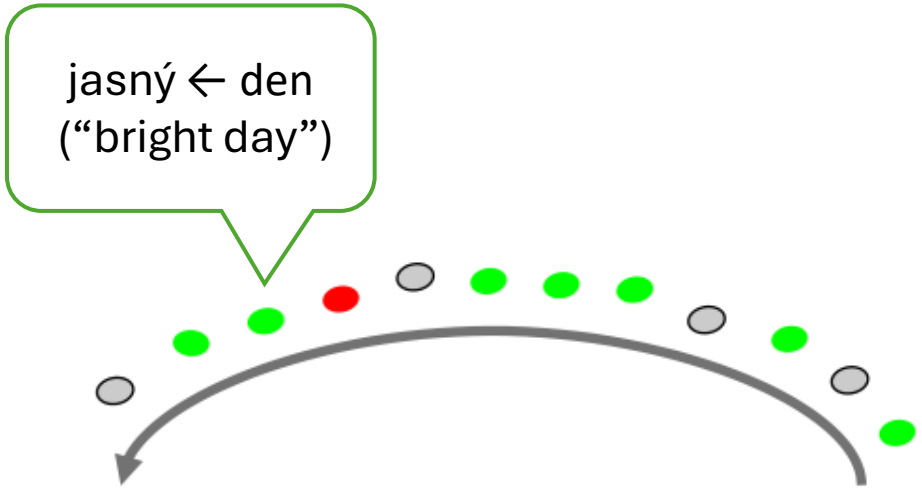
- Is this a good edge?



"It was a bright cold day in April and the clocks were striking thirteen."

Edge-Factored Parsers (McDonald et al. 2005)

- Is this a good edge?



Byl jasný studený dubnový den a hodiny odbíjely třináctou

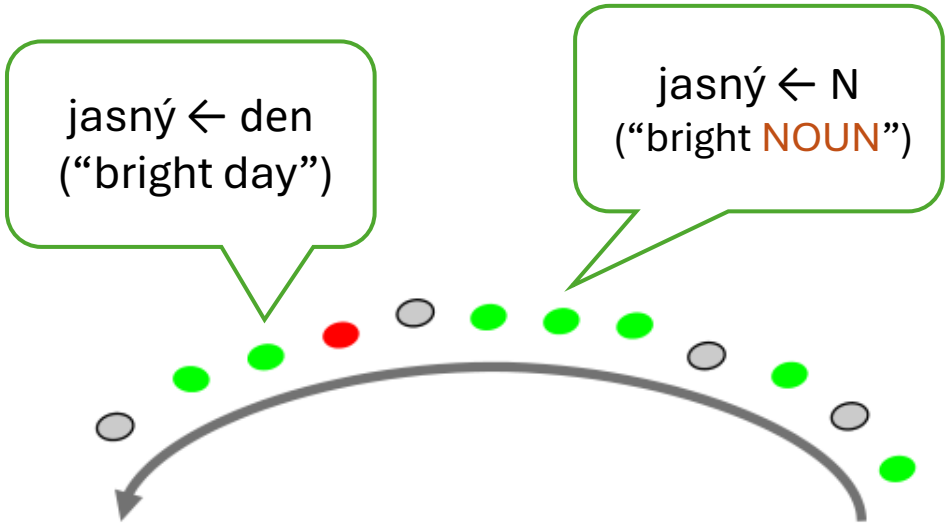
was bright cold April day and clock striking thirteen

V A A A N J N V C

“It was a bright cold day in April and the clocks were striking thirteen.”

Edge-Factored Parsers (McDonald et al. 2005)

- Is this a good edge?



Byl jasný studený dubnový den a hodiny odbíjely třináctou

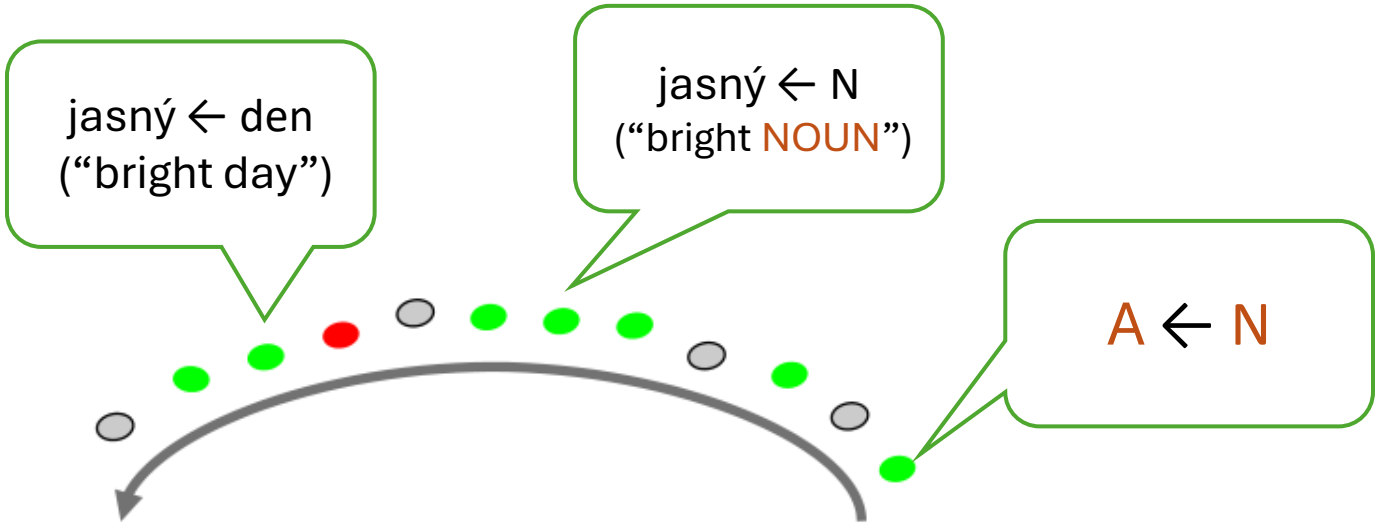
was bright cold April day and clock striking thirteen

V A A A N J N V C

“It was a bright cold day in April and the clocks were striking thirteen.”

Edge-Factored Parsers (McDonald et al. 2005)

- Is this a good edge?



Byl jasný studený dubnový den a hodiny odbíjely třináctou

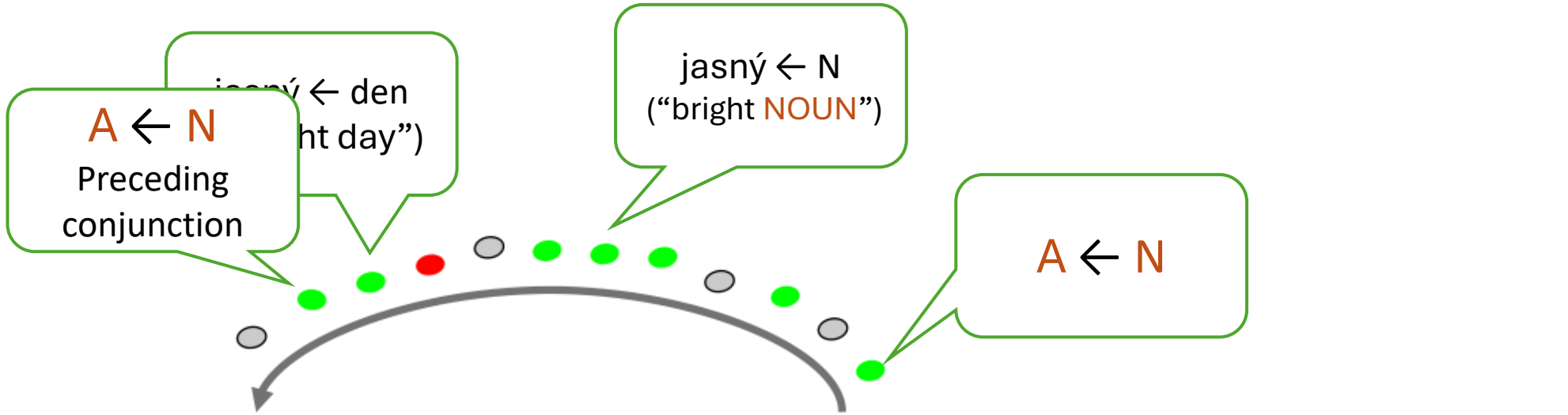
was bright cold April day and clock striking thirteen

V A A A N J N V C

“It was a bright cold day in April and the clocks were striking thirteen.”

Edge-Factored Parsers (McDonald et al. 2005)

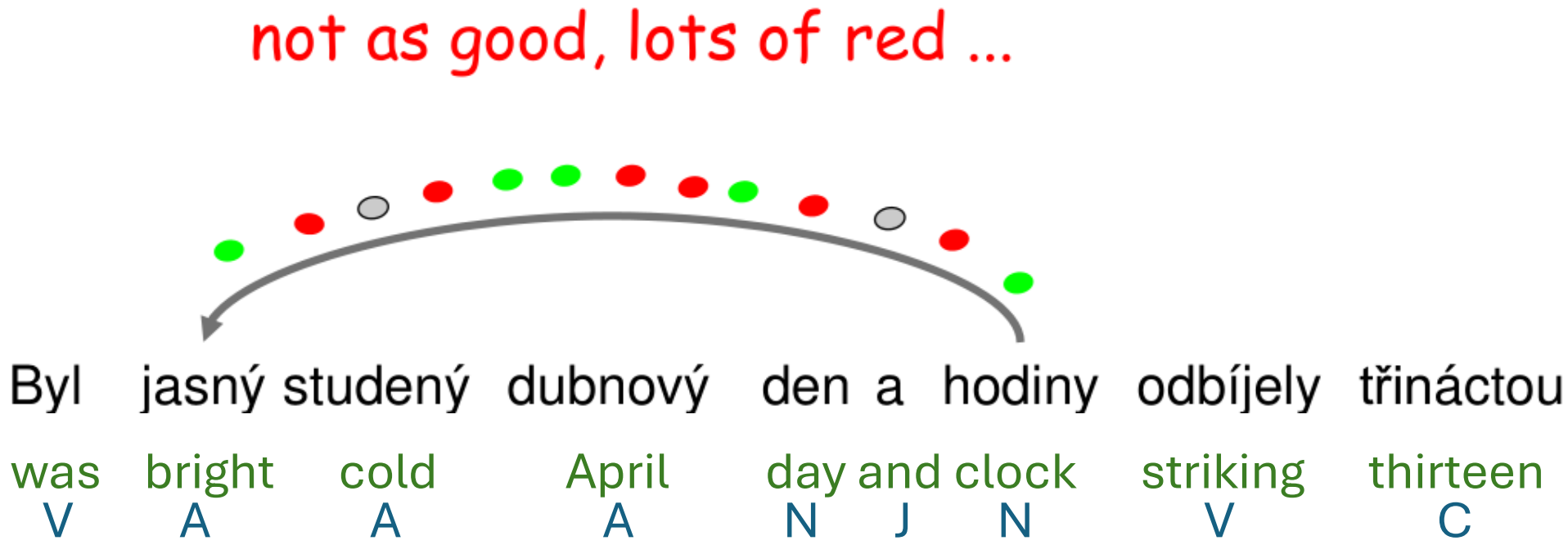
- Is this a good edge?



“It was a bright cold day in April and the clocks were striking thirteen.”

Edge-Factored Parsers (McDonald et al. 2005)

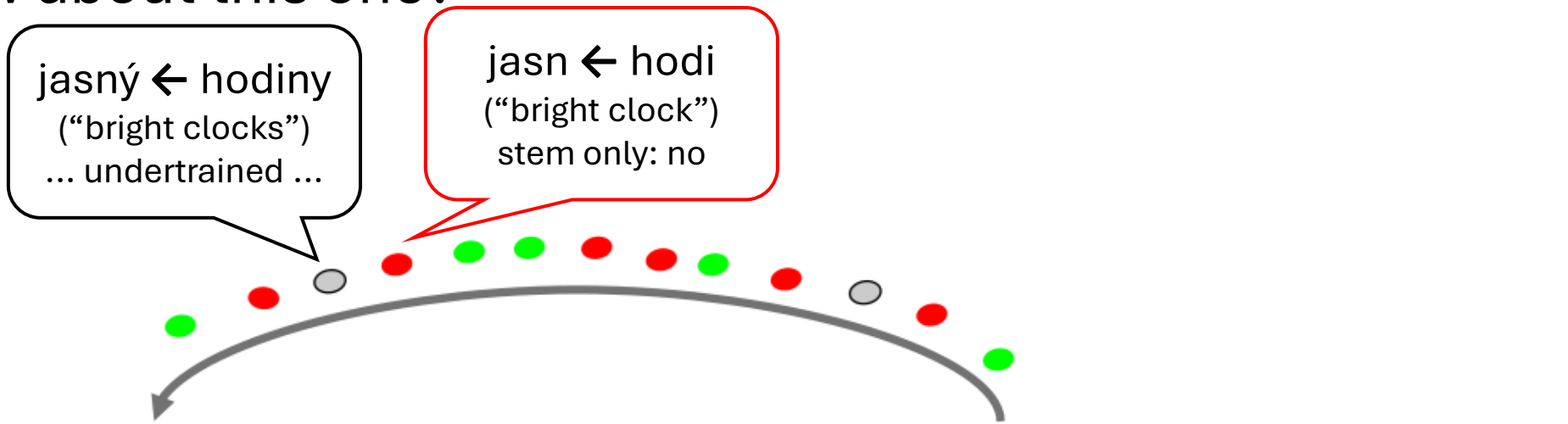
- How about this one?



"It was a bright cold day in April and the clocks were striking thirteen."

Edge-Factored Parsers (McDonald et al. 2005)

- How about this one?

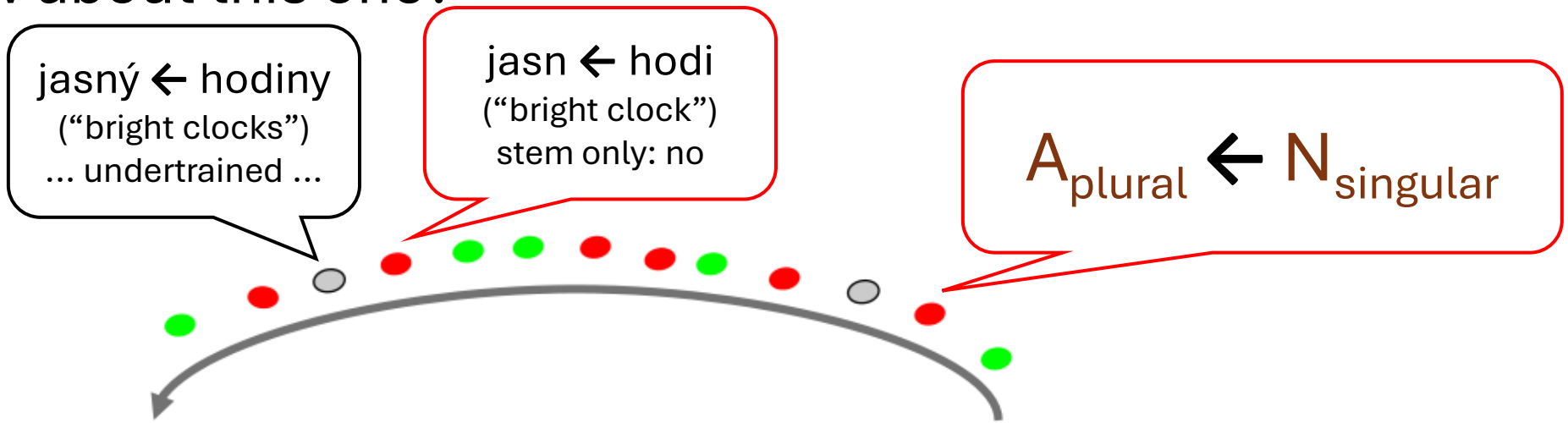


	Byl	jasný	studený	dubnový	den	a	hodiny	odbíjely	třináctou
	was	bright	cold	April	day	and	clock	striking	thirteen
POS:	V	A	A	A	N	J	N	V	C
Stem:	byl	jasn	stud	dubn	den	a	hodi	odbí	třin

“It was a bright cold day in April and the clocks were striking thirteen.”

Edge-Factored Parsers (McDonald et al. 2005)

- How about this one?

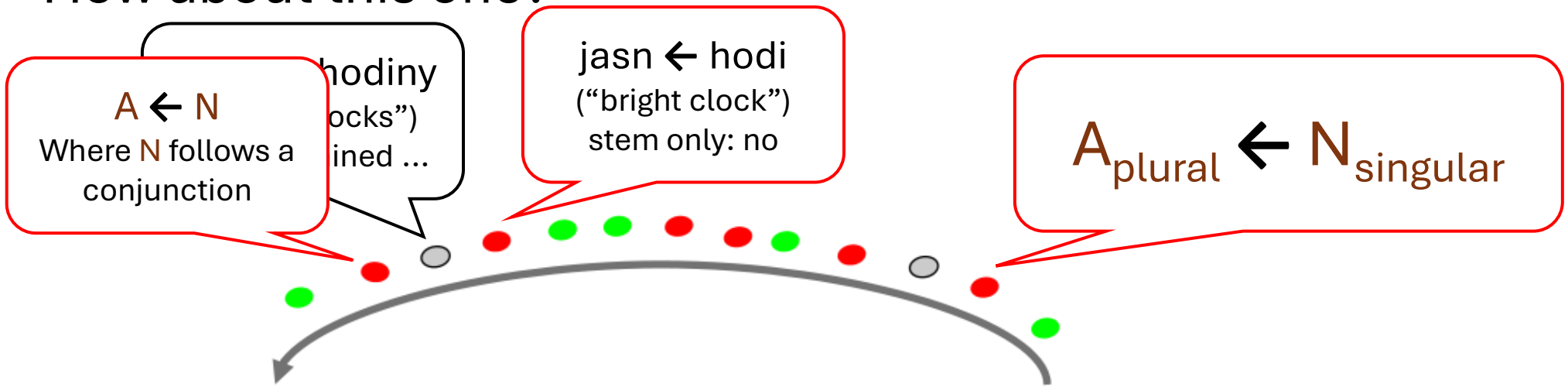


	Byl	jasný	studený	dubnový	den	a	hodiny	odbíjely	třináctou	
	was	bright	cold	April	day	and	clock	striking	thirteen	
POS:	V	A	A	A	N	J	N	V	C	
Stem:	byl	jasn	stud	dubn	den	a	hodi	odbí	třin	

“It was a bright cold day in April and the clocks were striking thirteen.”

Edge-Factored Parsers (McDonald et al. 2005)

- How about this one?

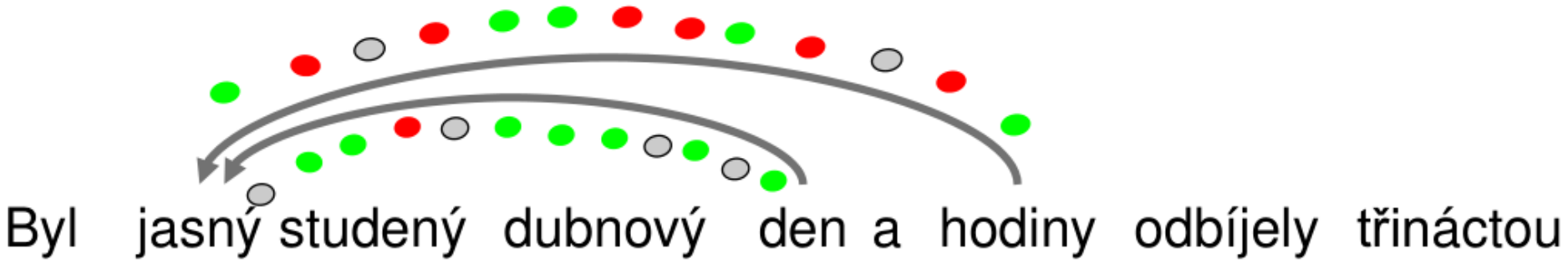


	Byl	jasný	studený	dubnový	den	a	hodiny	odbíjely	třináctou	
	was	bright	cold	April	day	and	clock	striking	thirteen	
POS:	V	A	A	A	N	J	N	V	C	
Stem:	byl	jasn	stud	dubn	den	a	hodi	odbí	třin	

"It was a bright cold day in April and the clocks were striking thirteen."

Edge-Factored Parsers (McDonald et al. 2005)

- Which edge is better?
  - “bright day” or “bright clocks”?

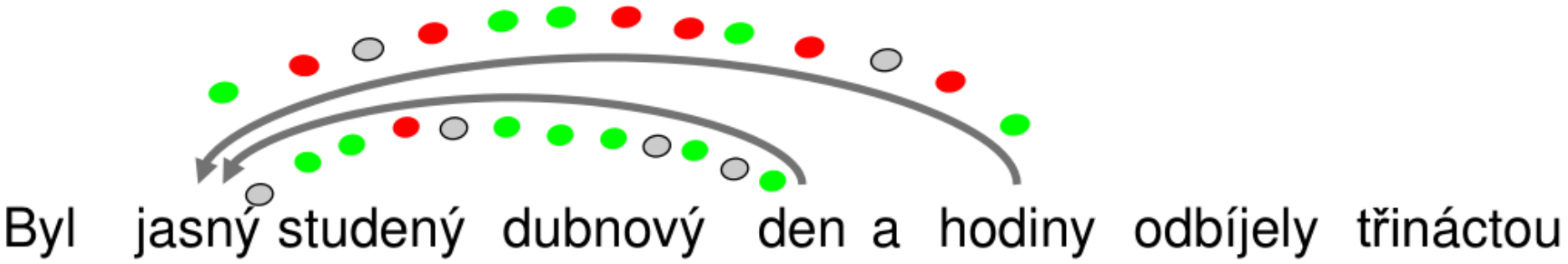


	was	bright	cold	April	day	and	clock	striking	thirteen
POS:	V	A	A	A	N	J	N	V	C
Stem:	byl	jasn	stud	dubn	den	a	hodi	odbí	třin

“It was a bright cold day in April and the clocks were striking thirteen.”

Edge-Factored Parsers (McDonald et al. 2005)

- Which edge is better?
- Score of an edge: $\text{score}(e) = \theta \cdot \mathbf{features}(e)$ — θ: our current weight vector
- Standard algorithms: valid parse with **max** total score.



	was	bright	cold	April	day	and	clock	striking	thirteen
POS:	V	A	A	A	N	J	N	V	C
Stem:	byl	jasn	stud	dubn	den	a	hodi	odbí	třin

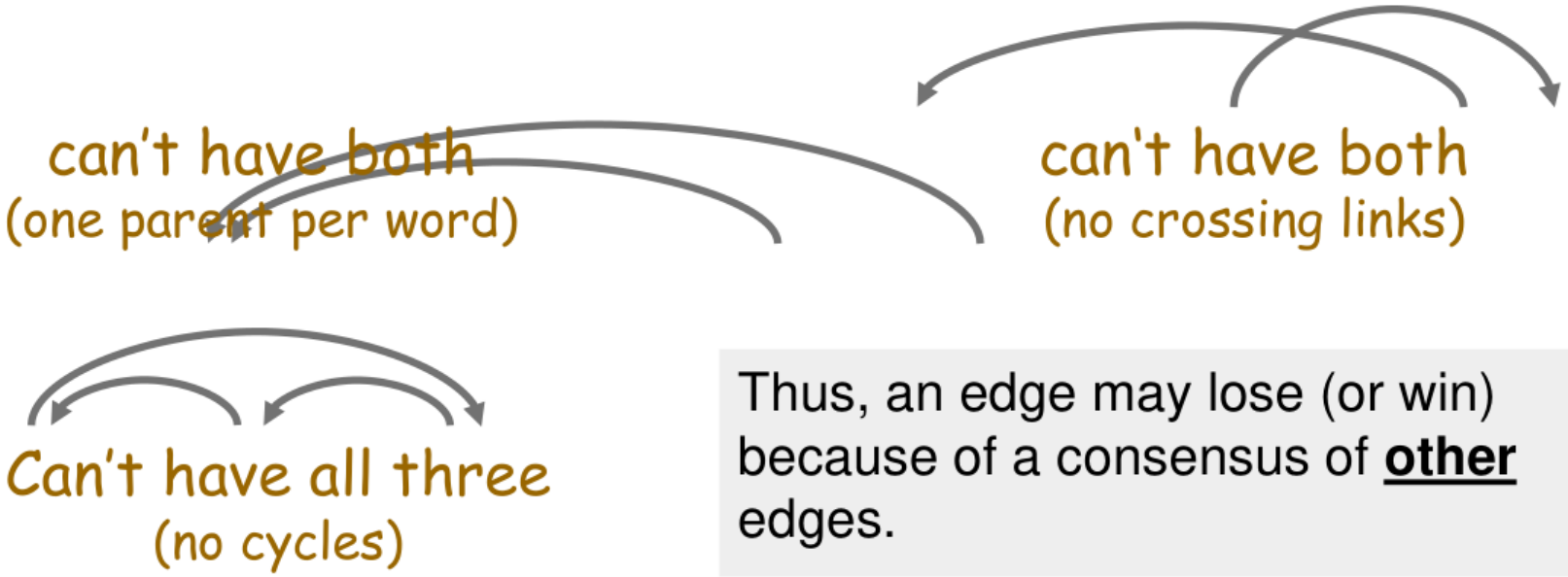
"It was a bright cold day in April and the clocks were striking thirteen."

Edge-Factored Parsers (McDonald et al. 2005)

- Which edge is better?
- Score of an edge: $\text{score}(e) = \theta \cdot \mathbf{features}(e)$

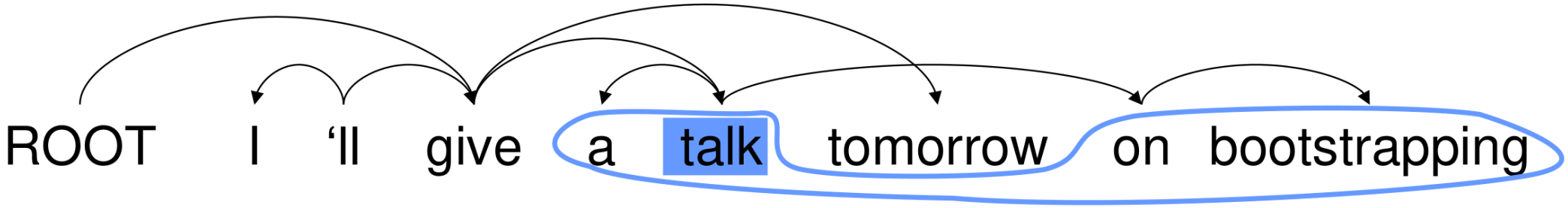
our current weight vector

- Standard algorithms: **valid** parse with **max** total score.



Thus, an edge may lose (or win) because of a consensus of **other** edges.

Non-Projective Parses

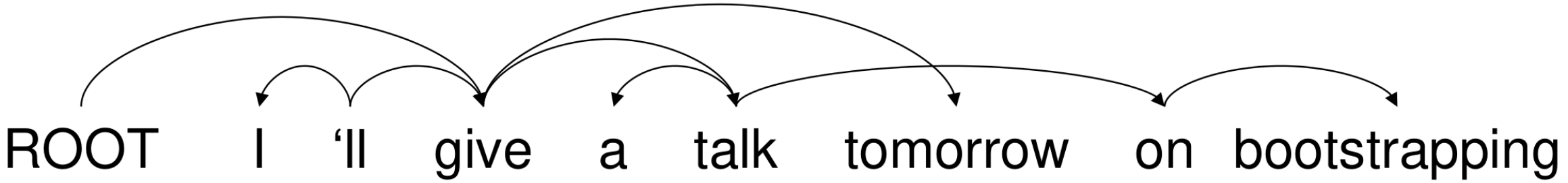


subtree rooted at "talk"
is a **discontiguous** noun phrase

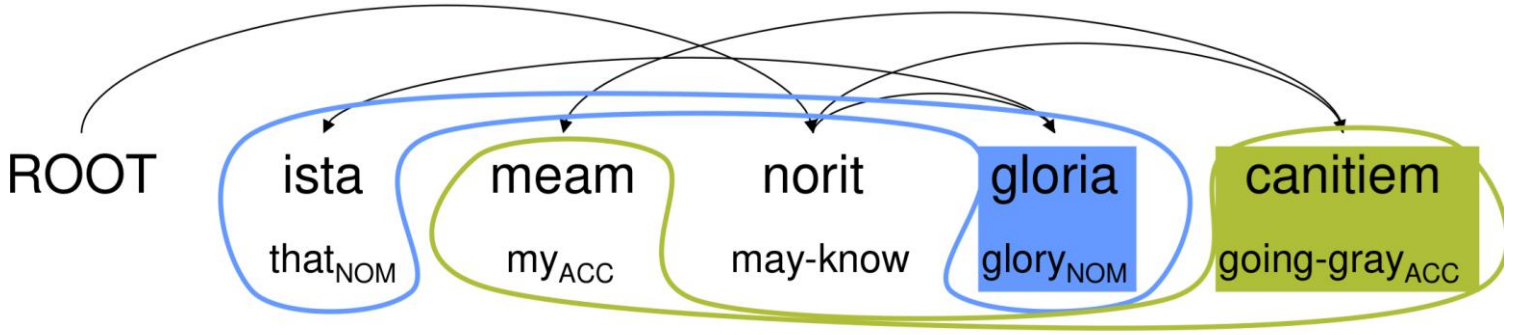
can't have both
(no crossing links)

The "projectivity" restriction.
Do we really want it?

Non-Projective Parses



English: non-projectivity occasionally



That glory may-know my going-gray
(i.e., it shall last till I go gray)

Frequent non-projectivity in Latin, etc.

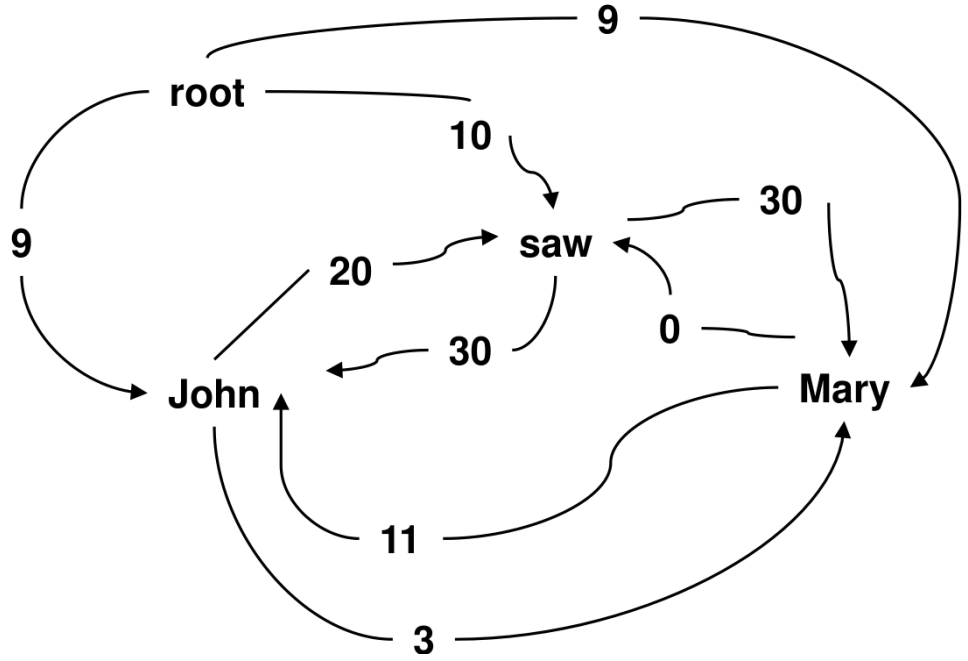
Non-Projective Parsing Algorithms

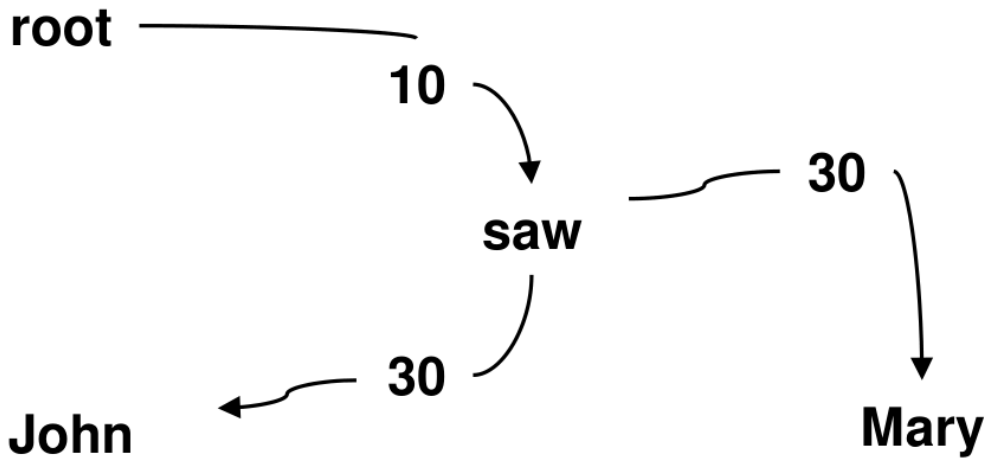
- Complexity considerations:
  - Projective (Proj)
  - Non-projective (NonP)

Problem/Algorithm	Proj	NonP
Complete grammar parsing [Gaifman 1965, Neuhaus and Bröker 1997]	P	NP hard
Deterministic parsing [Nivre 2003, Covington 2001]	$O(n)$	$O(n^2)$
First order spanning tree [McDonald et al. 2005b]	$O(n^3)$	$O(n^2)$
Nth order spanning tree ($N > 1$) [McDonald and Pereira 2006]	P	NP hard

McDonald's Approach (non-projective)

- Example: " Jonh saw Mary "
- We can use the **Chu-Liu-Edmonds** (CLE) algorithm to find the maximum-weight spanning tree.
  - Can be non-projective!
- Can be found in time $O(n^2)$.



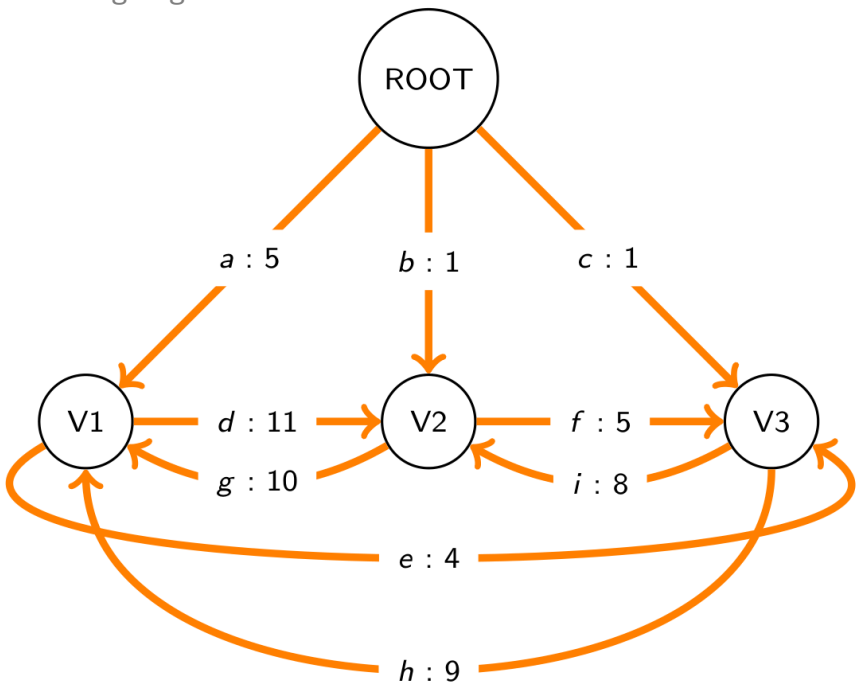


The **maximum-weight spanning tree**.
The best parse with the highest score!

Chu-Liu-Edmonds - Contracting Stage

- For each non-ROOT node v , set `bestInEdge`[v] to be its highest scoring incoming edge.
- If a cycle C is formed:
  - contract the nodes in C into a new node v_C
  - edges outgoing from any node in C now get source v_C
  - edges incoming to any node in C now get destination v_C
  - For each node u in C , and for each edge e incoming to u from outside of C:
    - add to e.`kicksOut` the edge bestInEdge[u], and
    - set e.`score` to be e.`score` – e.`kicksOut`.`score`.
- Repeat until every non-ROOT node has an incoming edge and no cycles are formed.

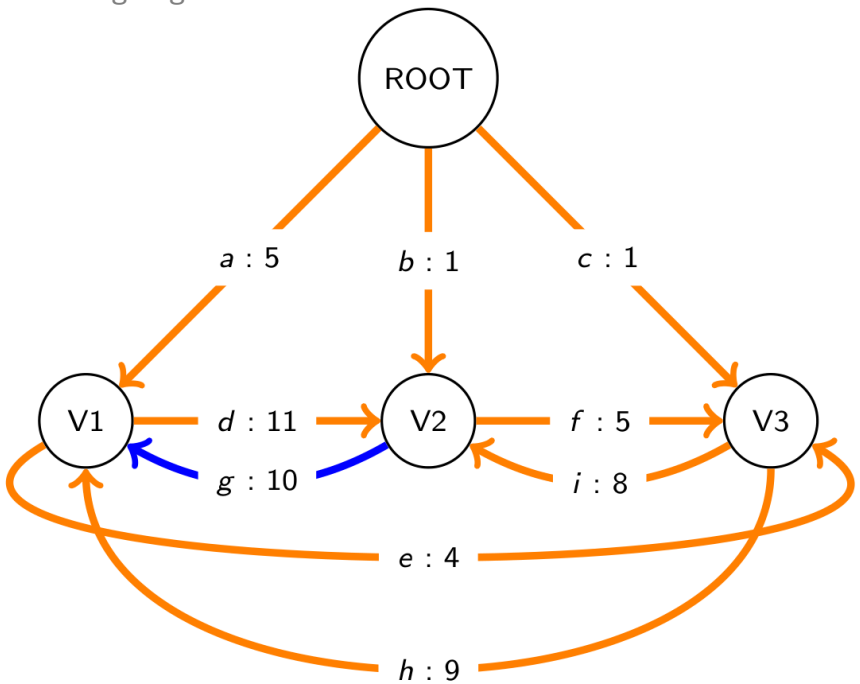
- **For each non-ROOT node `v` , set `bestInEdge[v]` to be its highest scoring incoming edge.**
- If a cycle *C* is formed:
  - contract the nodes in *C* into a new node v_C
  - edges outgoing from any node in *C* now get source v_C
  - edges incoming to any node in *C* now get destination v_C
  - For each node *u* in *C* , and for each edge *e* incoming to *u* from outside of *C*:
    - add to *e*.`kicksOut` the edge bestInEdge[*u*], and
    - set *e*.`score` to be *e*.`score` – *e*.`kicksOut.score`.
- Repeat until every non-ROOT node has an incoming edge and no cycles are formed.



	bestInEdge
V1	
V2	
V3	

	kicksOut
a	
b	
c	
d	
e	
f	
g	
h	
i	

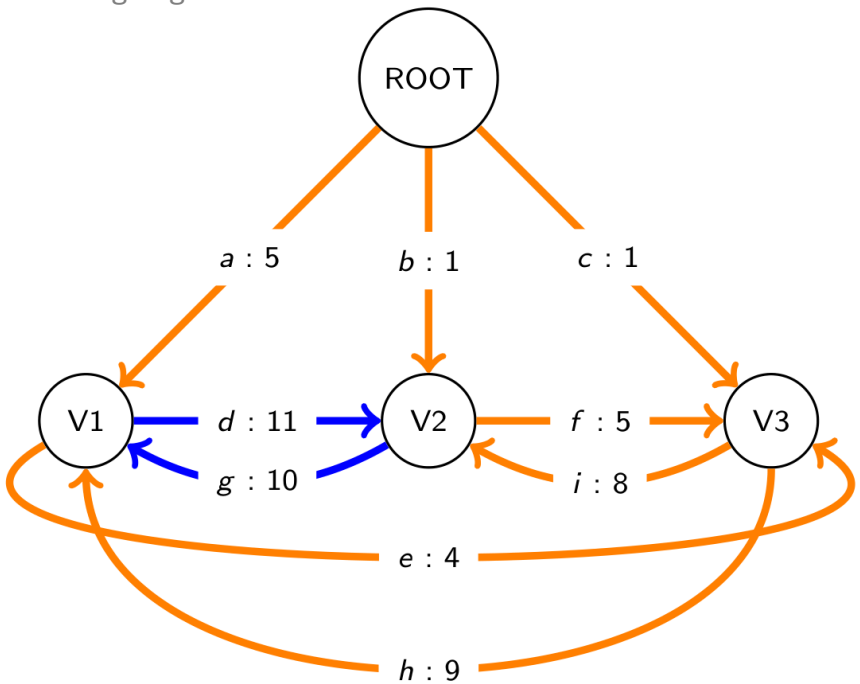
- **For each non-ROOT node *v* , set `bestInEdge[v]` to be its highest scoring incoming edge.**
- If a cycle *C* is formed:
  - contract the nodes in *C* into a new node v_C
  - edges outgoing from any node in *C* now get source v_C
  - edges incoming to any node in *C* now get destination v_C
  - For each node *u* in *C* , and for each edge *e* incoming to *u* from outside of *C*:
    - add to *e*.`kicksOut` the edge bestInEdge[*u*], and
    - set *e*.`score` to be *e*.`score` – *e*.`kicksOut.score`.
- Repeat until every non-ROOT node has an incoming edge and no cycles are formed.



	bestInEdge
V1	g
V2	
V3	

	kicksOut
a	
b	
c	
d	
e	
f	
g	
h	
i	

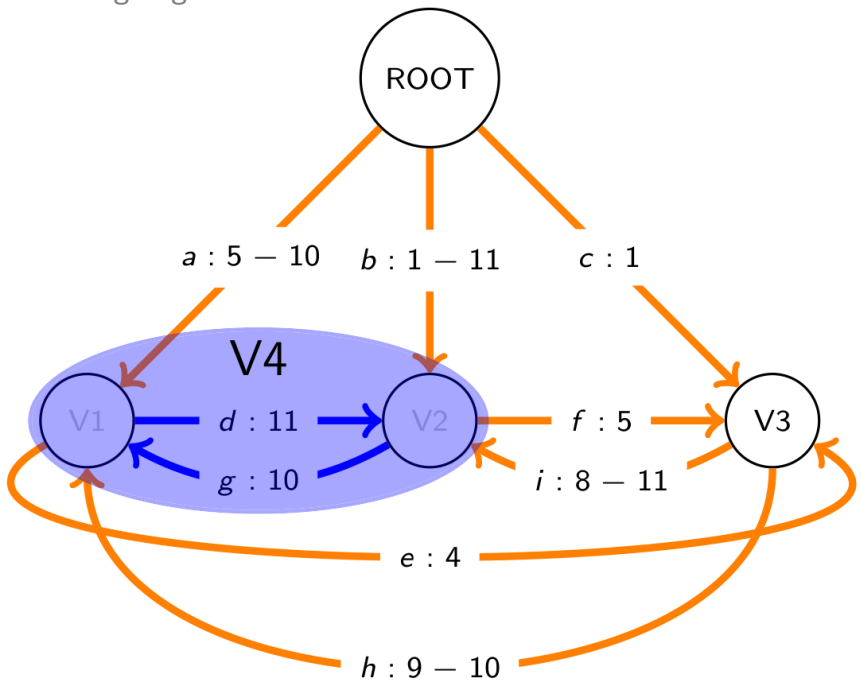
- **For each non-ROOT node *v* , set `bestInEdge[v]` to be its highest scoring incoming edge.**
- If a cycle *C* is formed:
  - contract the nodes in *C* into a new node v_C
  - edges outgoing from any node in *C* now get source v_C
  - edges incoming to any node in *C* now get destination v_C
  - For each node *u* in *C* , and for each edge *e* incoming to *u* from outside of *C*:
    - add to *e*.`kicksOut` the edge bestInEdge[*u*], and
    - set *e*.`score` to be *e*.`score` – *e*.`kicksOut.score`.
- Repeat until every non-ROOT node has an incoming edge and no cycles are formed.



	bestInEdge
V1	g
V2	d
V3	

	kicksOut
a	
b	
c	
d	
e	
f	
g	
h	
i	

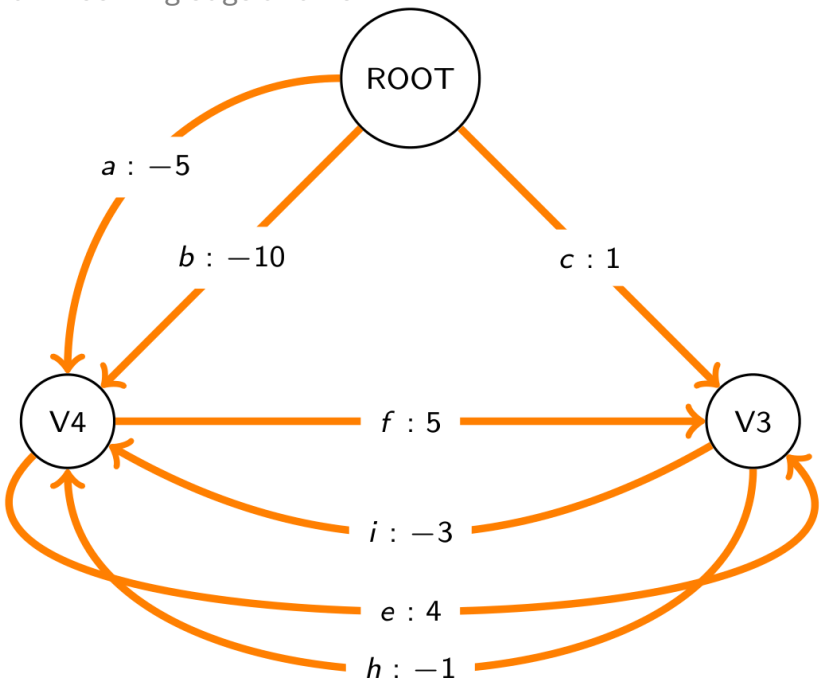
- For each non-ROOT node v , set `bestInEdge[v]` to be its highest scoring incoming edge.
- **If a cycle C is formed:**
  - **contract the nodes in C into a new node v_C**
  - **edges outgoing from any node in C now get source v_C**
  - **edges incoming to any node in C now get destination v_C**
  - **For each node u in C , and for each edge e incoming to u from outside of C:**
    - **add to e.`kicksOut` the edge bestInEdge[u], and**
    - **set e.`score` to be e.`score` – e.`kicksOut`.`score`.**
- Repeat until every non-ROOT node has an incoming edge and no cycles are formed.



	bestInEdge
V1	g
V2	d
V3	

	kicksOut
a	g
b	d
c	
d	
e	
f	
g	
h	g
i	d

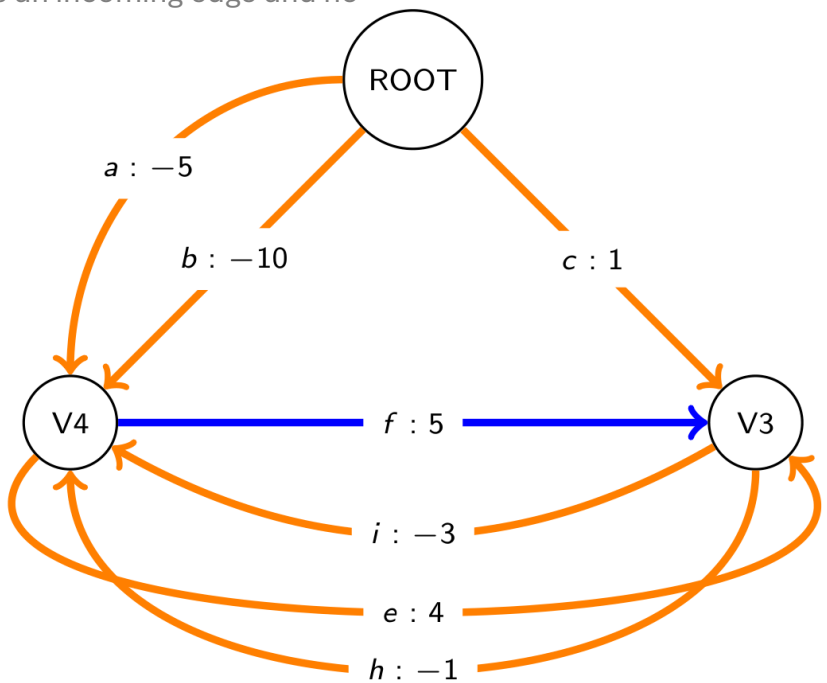
- For each non-ROOT node v , set `bestInEdge[v]` to be its highest scoring incoming edge.
- **If a cycle C is formed:**
  - **contract the nodes in C into a new node v_C**
  - **edges outgoing from any node in C now get source v_C**
  - **edges incoming to any node in C now get destination v_C**
  - **For each node u in C , and for each edge e incoming to u from outside of C:**
    - **add to e.`kicksOut` the edge bestInEdge[u], and**
    - **set e.`score` to be e.`score` – e.`kicksOut`.`score`.**
- Repeat until every non-ROOT node has an incoming edge and no cycles are formed.



	bestInEdge
V1	g
V2	d
V3	
V4	

	kicksOut
a	g
b	d
c	
d	
e	
f	
g	
h	g
i	d

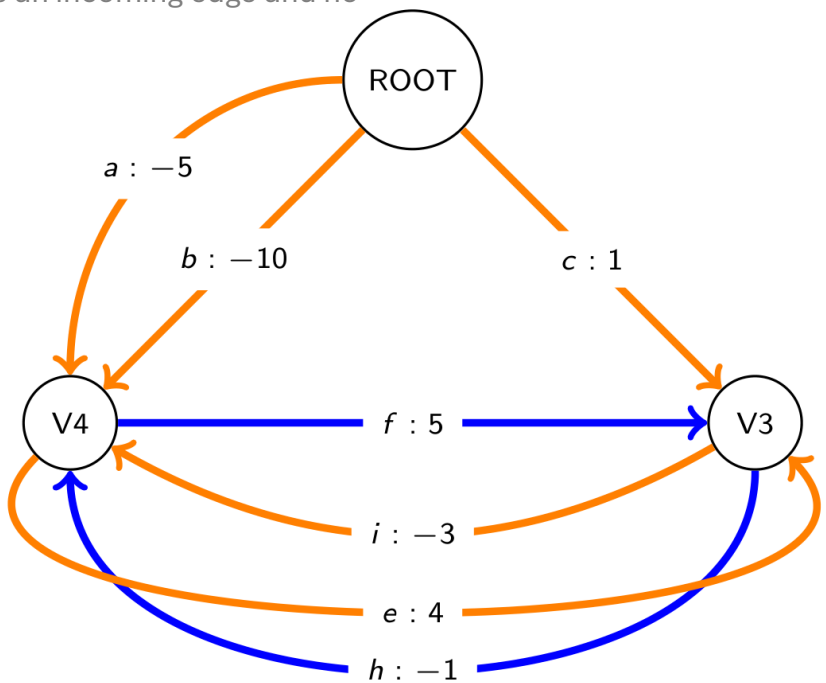
- **For each non-ROOT node `v` , set `bestInEdge[v]` to be its highest scoring incoming edge.**
- If a cycle *C* is formed:
  - contract the nodes in *C* into a new node v_C
  - edges outgoing from any node in *C* now get source v_C
  - edges incoming to any node in *C* now get destination v_C
  - For each node *u* in *C* , and for each edge *e* incoming to *u* from outside of *C*:
    - add to *e*.`kicksOut` the edge bestInEdge[*u*], and
    - set *e*.`score` to be *e*.`score` – *e*.`kicksOut.score`.
- Repeat until every non-ROOT node has an incoming edge and no cycles are formed.



	bestInEdge
V1	g
V2	d
V3	f
V4	

	kicksOut
a	g
b	d
c	
d	
e	
f	
g	
h	g
i	d

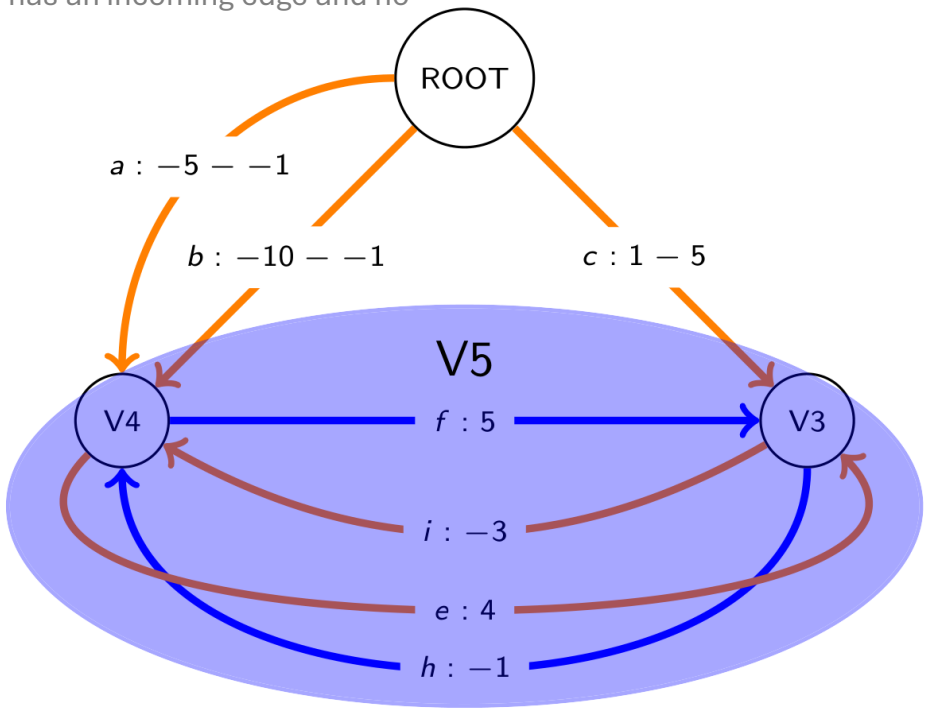
- **For each non-ROOT node `v` , set `bestInEdge[v]` to be its highest scoring incoming edge.**
- If a cycle *C* is formed:
  - contract the nodes in *C* into a new node v_C
  - edges outgoing from any node in *C* now get source v_C
  - edges incoming to any node in *C* now get destination v_C
  - For each node *u* in *C* , and for each edge *e* incoming to *u* from outside of *C*:
    - add to *e*.kicksOut the edge bestInEdge[*u*], and
    - set *e*.score to be *e*.score – *e*.kicksOut.score.
- Repeat until every non-ROOT node has an incoming edge and no cycles are formed.



	bestInEdge
V1	g
V2	d
V3	f
V4	h

	kicksOut
a	g
b	d
c	
d	
e	
f	
g	
h	g
i	d

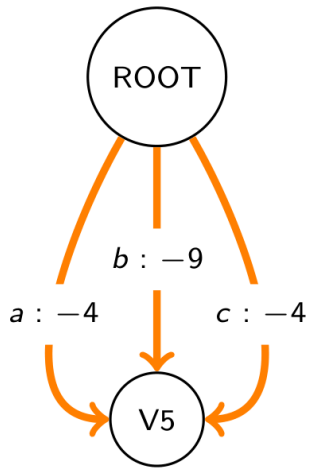
- For each non-ROOT node v, set `bestInEdge[v]` to be its highest scoring incoming edge.
- **If a cycle C is formed:**
  - **contract the nodes in C into a new node v_C**
  - **edges outgoing from any node in C now get source v_C**
  - **edges incoming to any node in C now get destination v_C**
  - **For each node u in C, and for each edge e incoming to u from outside of C:**
    - **add to e.`kicksOut` the edge bestInEdge[u], and**
    - **set e.`score` to be e.`score` – e.`kicksOut`.`score`.**
- Repeat until every non-ROOT node has an incoming edge and no cycles are formed.



	bestInEdge
V1	g
V2	d
V3	f
V4	h
V5	

	kicksOut
a	g, h
b	d, h
c	f
d	
e	
f	
g	
h	g
i	d

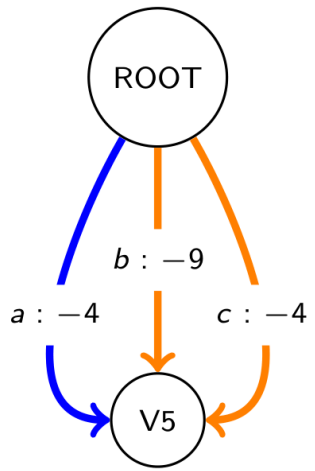
- **For each non-ROOT node *v* , set `bestInEdge[v]` to be its highest scoring incoming edge.**
- If a cycle *C* is formed:
  - contract the nodes in *C* into a new node v_C
  - edges outgoing from any node in *C* now get source v_C
  - edges incoming to any node in *C* now get destination v_C
  - For each node *u* in *C* , and for each edge *e* incoming to *u* from outside of *C*:
    - add to *e*.`kicksOut` the edge bestInEdge[*u*], and
    - set *e*.`score` to be *e*.`score` – *e*.`kicksOut.score`.
- Repeat until every non-ROOT node has an incoming edge and no cycles are formed.



	bestInEdge
V1	g
V2	d
V3	f
V4	h
V5	

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

- For each non-ROOT node v , set `bestInEdge[v]` to be its highest scoring incoming edge.
- If a cycle C is formed:
  - contract the nodes in C into a new node v_C
  - edges outgoing from any node in C now get source v_C
  - edges incoming to any node in C now get destination v_C
  - For each node u in C , and for each edge e incoming to u from outside of C:
    - add to e.`kicksOut` the edge bestInEdge[u], and
    - set e.`score` to be e.`score` – e.`kicksOut.score`.
- **Repeat until every non-ROOT node has an incoming edge and no cycles are formed.**



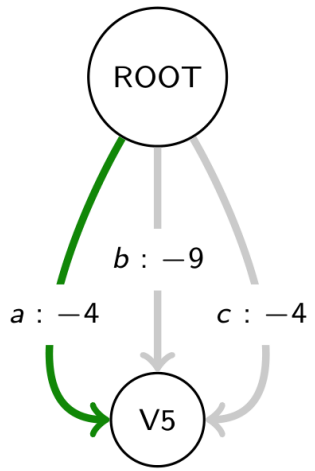
	bestInEdge
V1	g
V2	d
V3	f
V4	h
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

Chu-Liu-Edmonds - Expanding Stage

- After the contracting stage, every contracted node will have exactly one `bestInEdge`. This edge will kick out one edge inside the contracted node, breaking the cycle.
  - Go through each `bestInEdge` of *e* in the reverse order that we added them
  - **lock down** *e* and **remove** every edge in `kicksOut(`*e*`)` from `bestInEdge`.

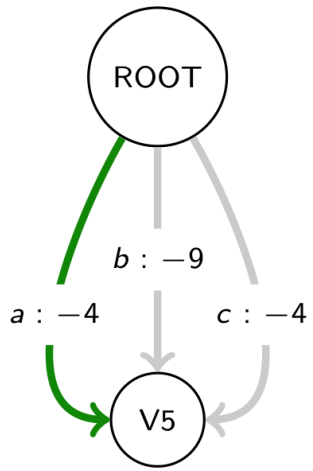
- Go through each `bestInEdge` of *e* in the reverse order that we added them
- **lock down** *e* and **remove** every edge in `kicksOut(e)` from `bestInEdge`.



	bestInEdge
V1	g
V2	d
V3	f
V4	h
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

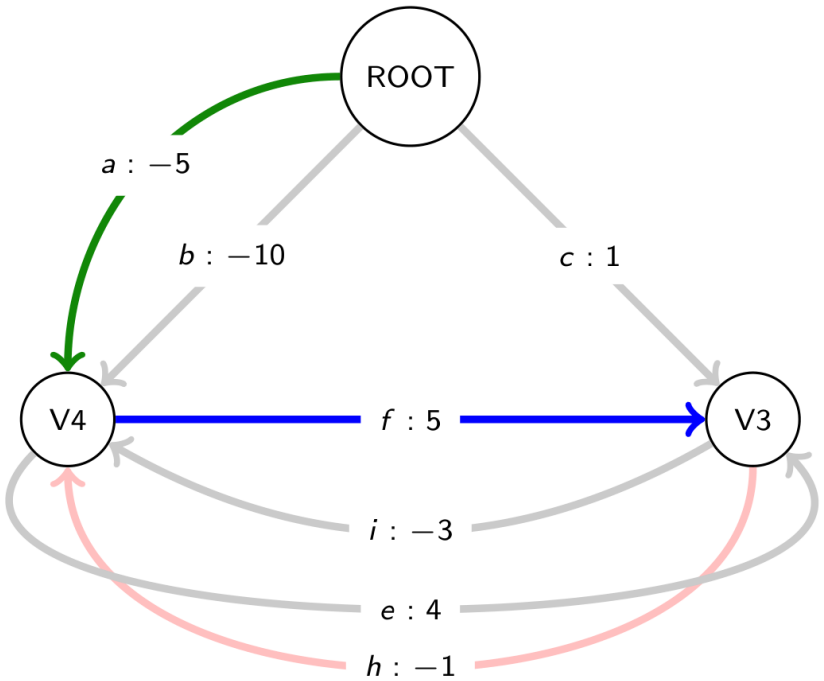
- Go through each `bestInEdge` of *e* in the reverse order that we added them
- **lock down** *e* and **remove** every edge in `kicksOut(e)` from `bestInEdge`.



	bestInEdge
V1	a ~~g~~
V2	d
V3	f
V4	a ~~h~~
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

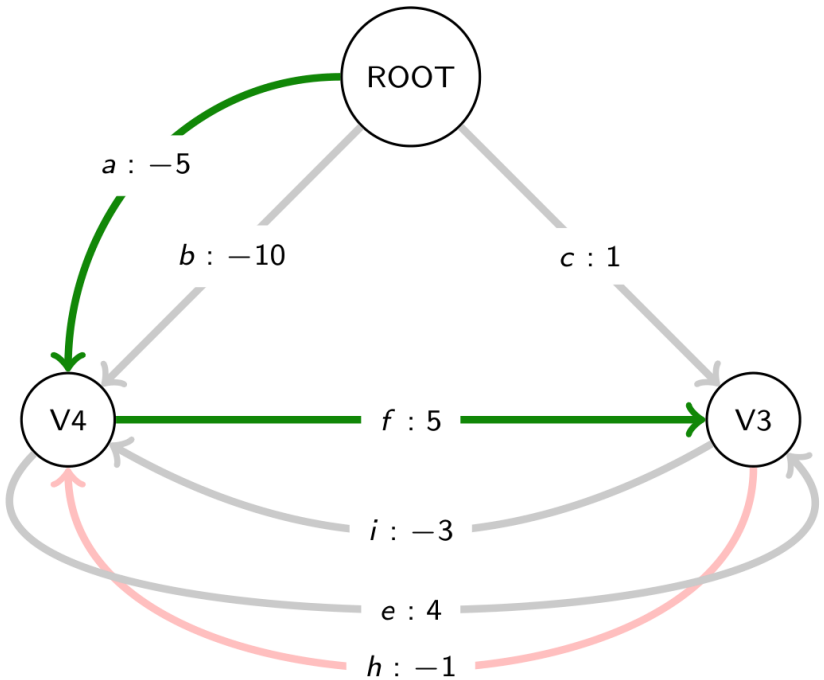
- Go through each `bestInEdge` of e in the reverse order that we added them
- **lock down** e and **remove** every edge in `kicksOut(`e`)` from `bestInEdge`.



	bestInEdge
V1	a ~~g~~
V2	d
V3	f
V4	a ~~h~~
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

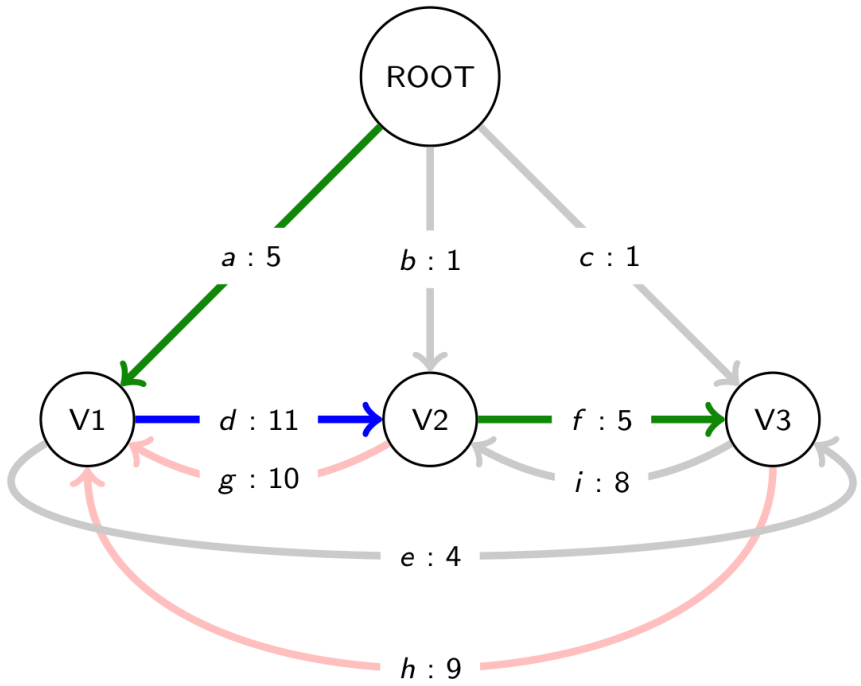
- Go through each `bestInEdge` of e in the reverse order that we added them
- **lock down** e and **remove** every edge in `kicksOut(`e`)` from `bestInEdge`.



	bestInEdge
V1	a ~~g~~
V2	d
V3	f
V4	a ~~h~~
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

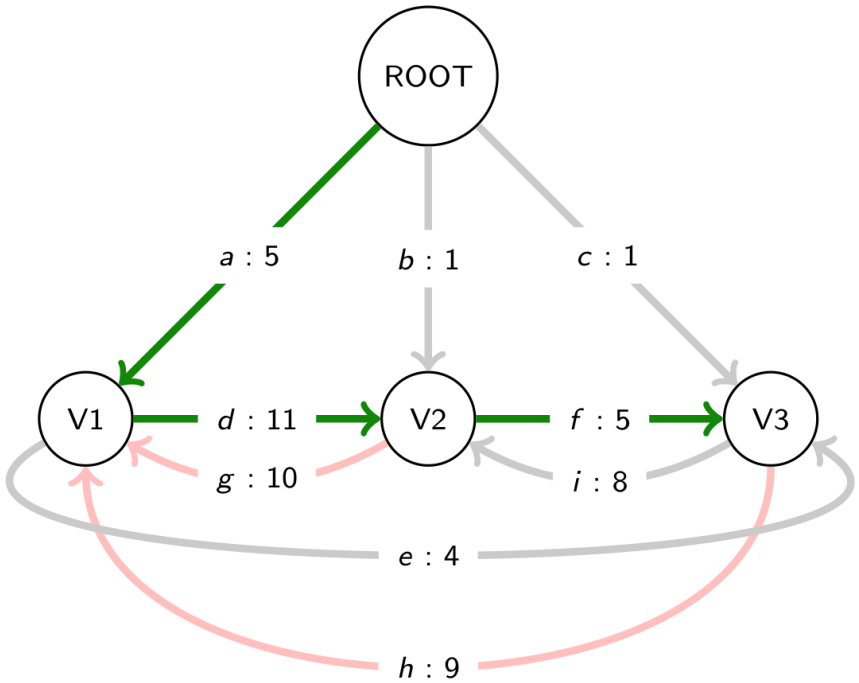
- Go through each `bestInEdge` of *e* in the reverse order that we added them
- **lock down** *e* and **remove** every edge in `kicksOut(e)` from `bestInEdge`.



	bestInEdge
V1	a ~~g~~
V2	d
V3	f
V4	a ~~h~~
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

- Go through each `bestInEdge` of *e* in the reverse order that we added them
- **lock down** *e* and **remove** every edge in `kicksOut(e)` from `bestInEdge`.



	bestInEdge
V1	a ~~g~~
V2	d
V3	f
V4	a ~~h~~
V5	a

	kicksOut
a	g, h
b	d, h
c	f
d	
e	f
f	
g	
h	g
i	d

Evaluation

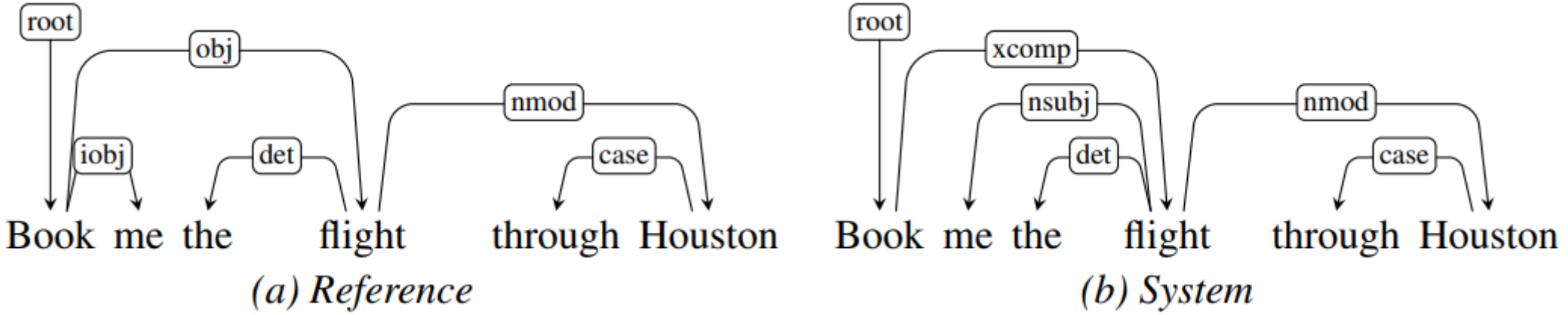
- For a sequence labelling task (e.g., PoS tagging), evaluation is straightforward.
  - Exact match (EM) accuracy.

		Correct	Model
1.	Unionized	VBN	JJ
2.	workers	NNS	NNS
3.	are	VBP	VBP
4.	usually	RB	RB
5.	better	RBR	RBR
6.	paid	VBN	VBN
7.	than	IN	IN
8.	their	PRP$	PRP$
9.	non-union	JJ	JJ
10.	counterparts	NNS	NNS

EM Acc
= Correct / Total
= 90%

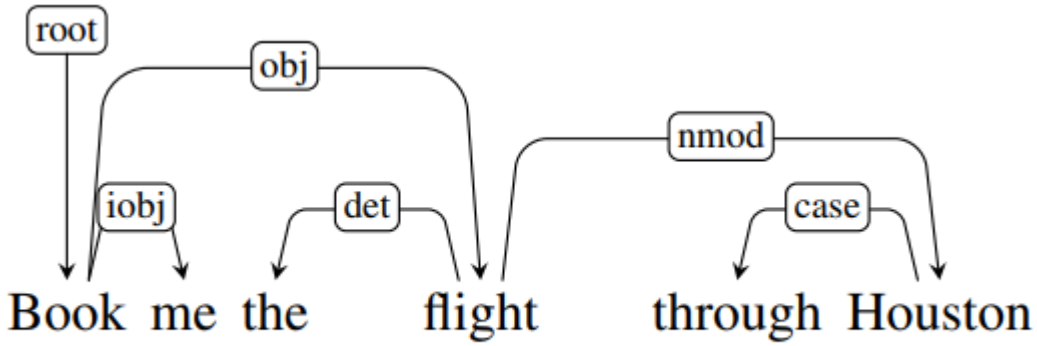
Evaluation

- What about dependency parses?

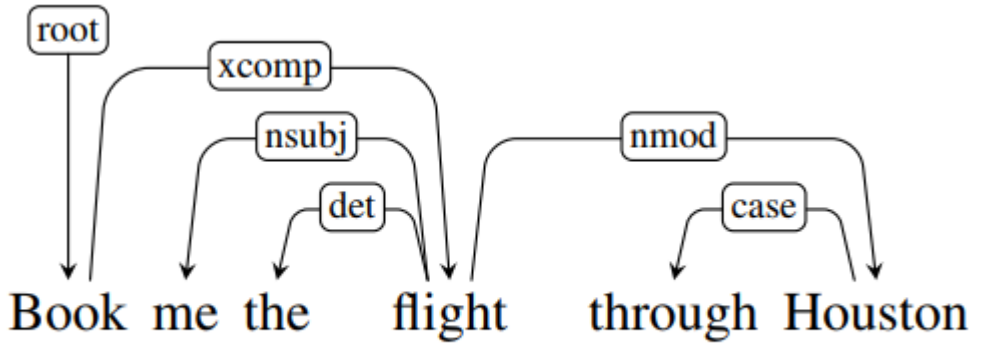


- Solution: count # words are correctly connected to their heads.

Evaluation: Unlabeled dependency accuracy (UAS)



(a) Reference

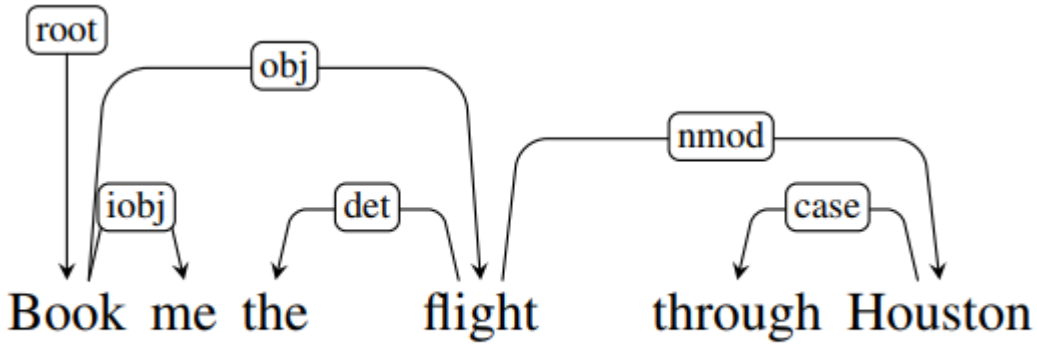


(b) System

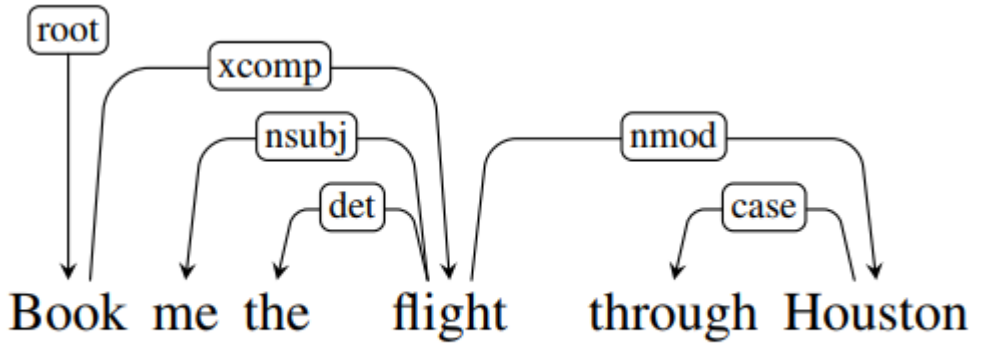
		(a) Reference	(b) System
1	book	0	0
2	me	1	4
3	the	4	4
4	flights	1	1
5	through	6	6
6	Houston	4	4

Accuracy
= 5/6 = 83.3%

Evaluation: Labeled dependency accuracy (LAS)



(a) Reference



(b) System

		(a) Reference		(b) System	
		head	label	head	label
1	book	0	root	0	root
2	me	1	iobj	**4**	**nsubj**
3	the	4	det	4	det
4	flights	1	obj	1	**xcomp**
5	through	6	case	6	case
6	Houston	4	nmod	4	nmod

Accuracy
= 4/6 = 66.7%