Language Technologies Institute

Carnegie Mellon University

Algorithms for NLP

CS 11-711 · Fall 2020

Lecture 14: Graph-based dependency parsing

Emma Strubell

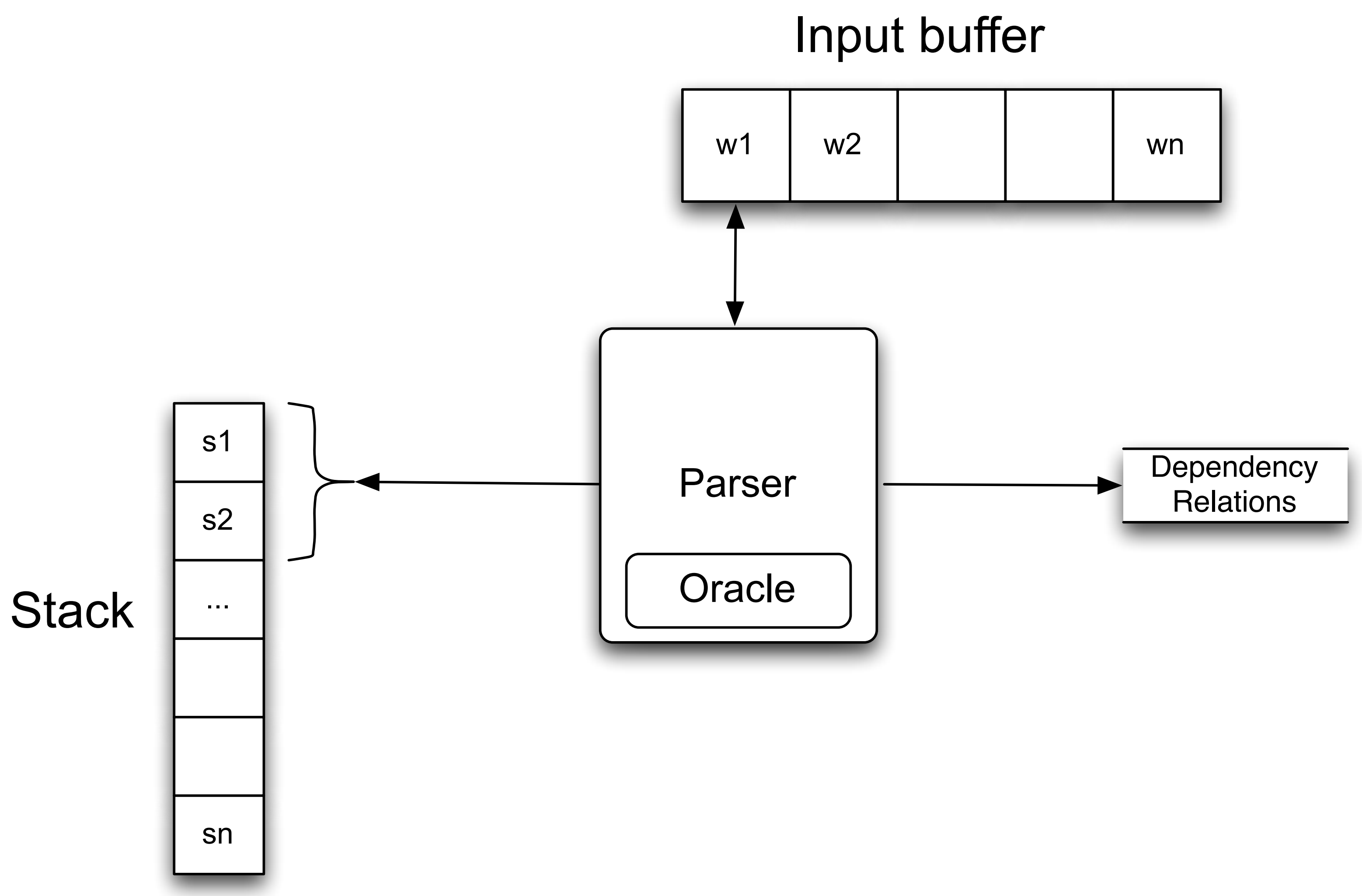
Announcements

- No recitation on Friday (Tartan Community Day).

Dependency parsing

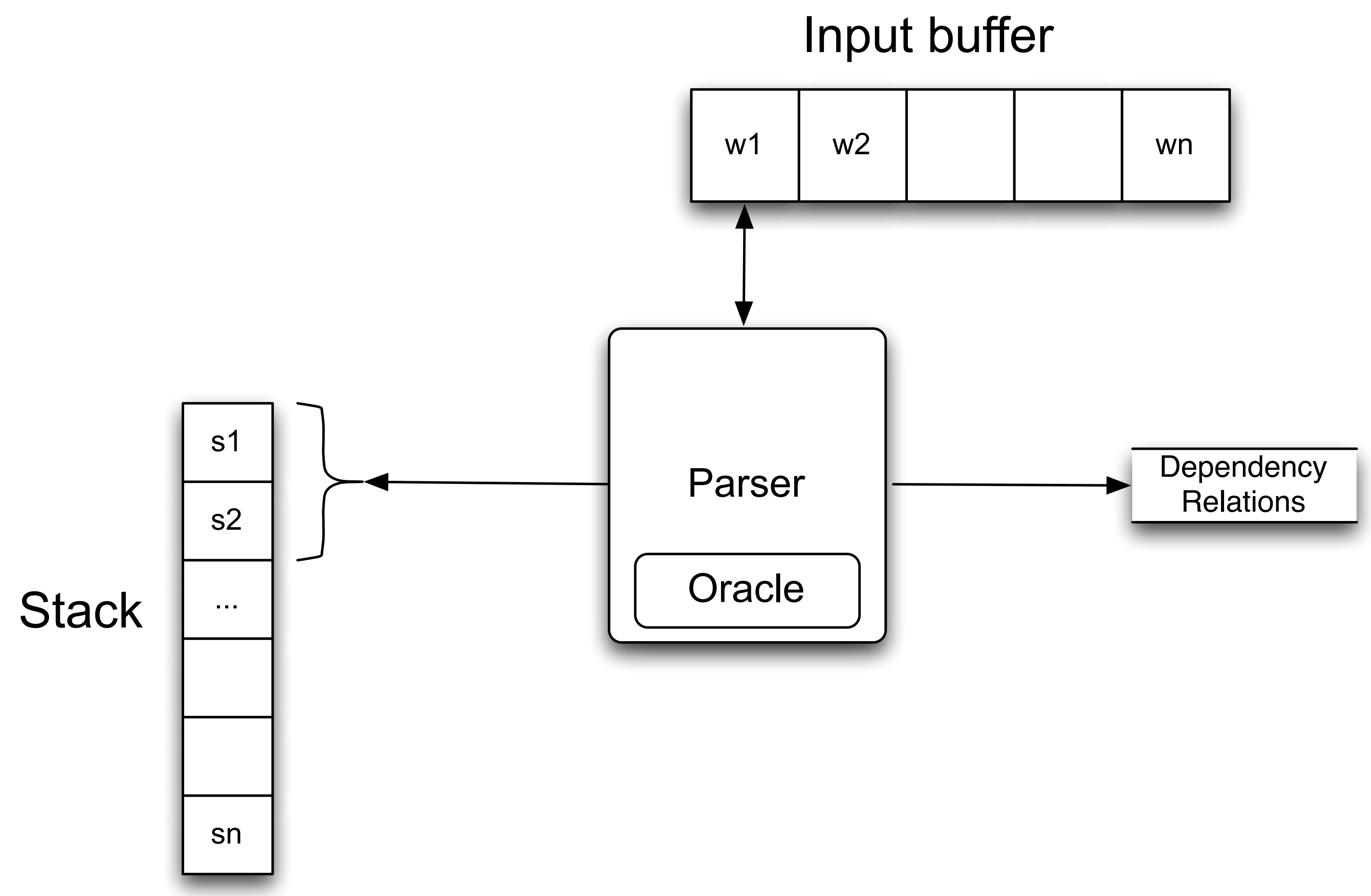
Dependency parsing

- Transition-based (shift-reduce) parsing:



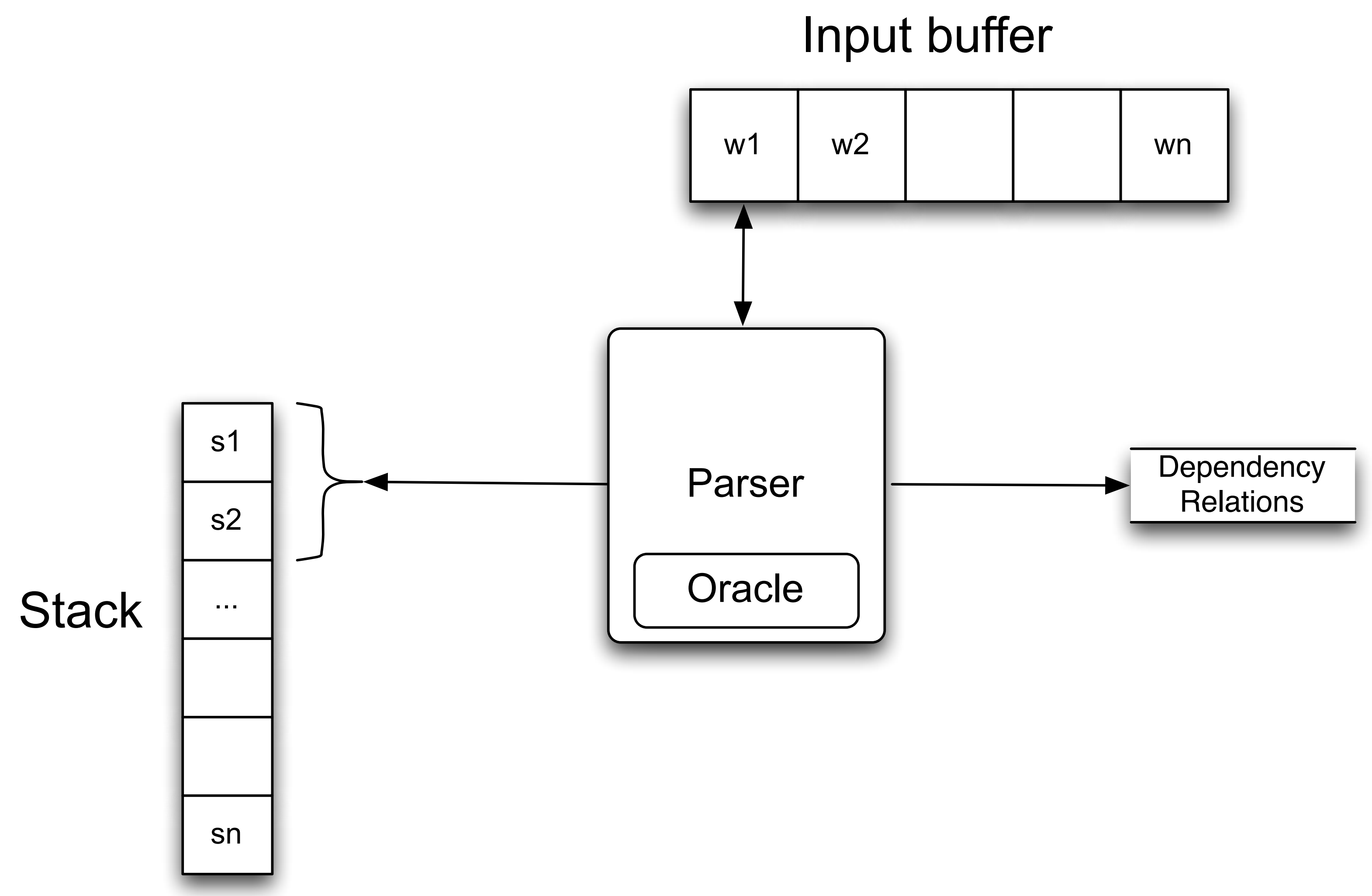
Dependency parsing

- Transition-based (shift-reduce) parsing:
  - Greedy choice of local transitions guided by a good classifier.



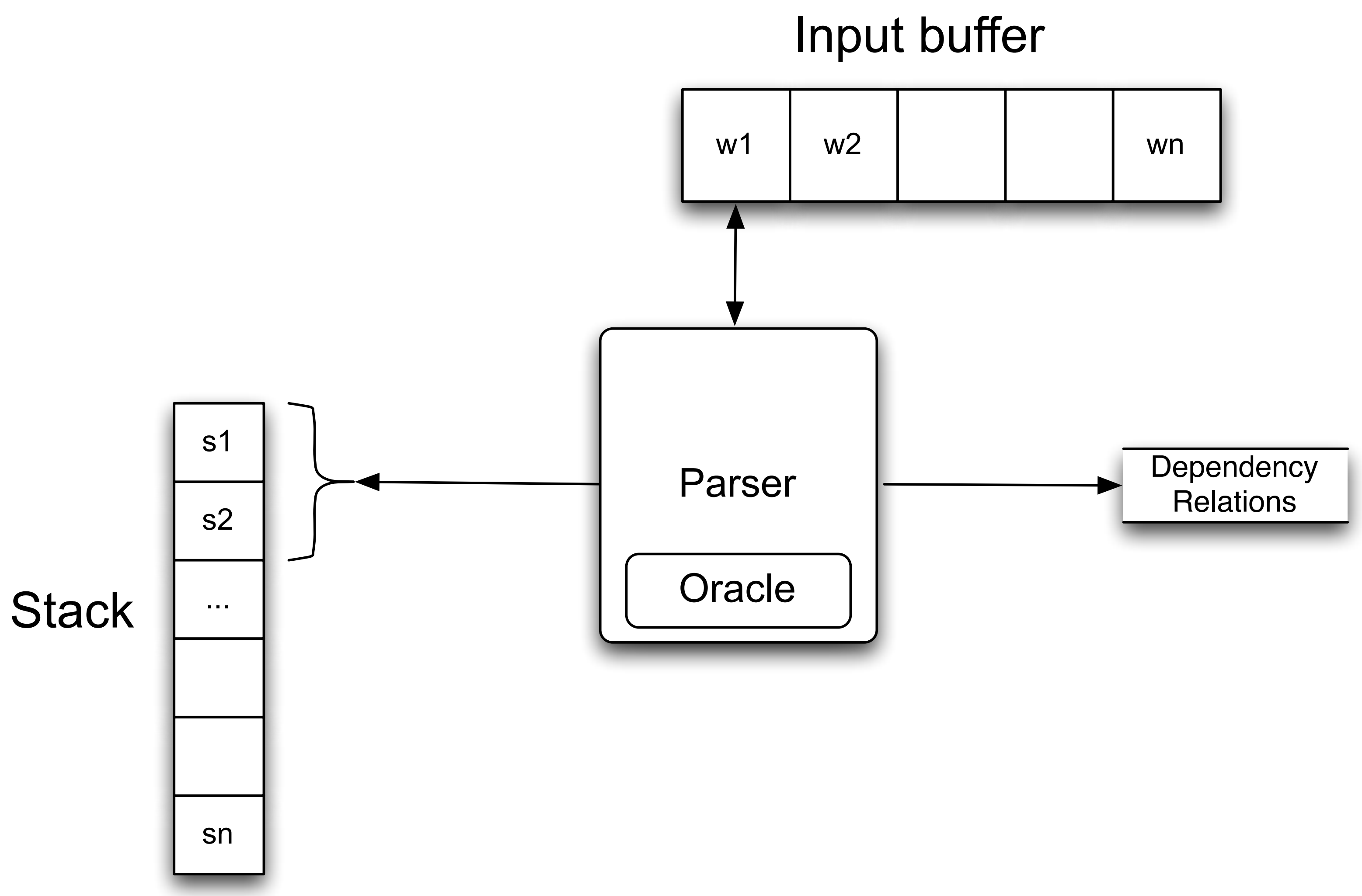
Dependency parsing

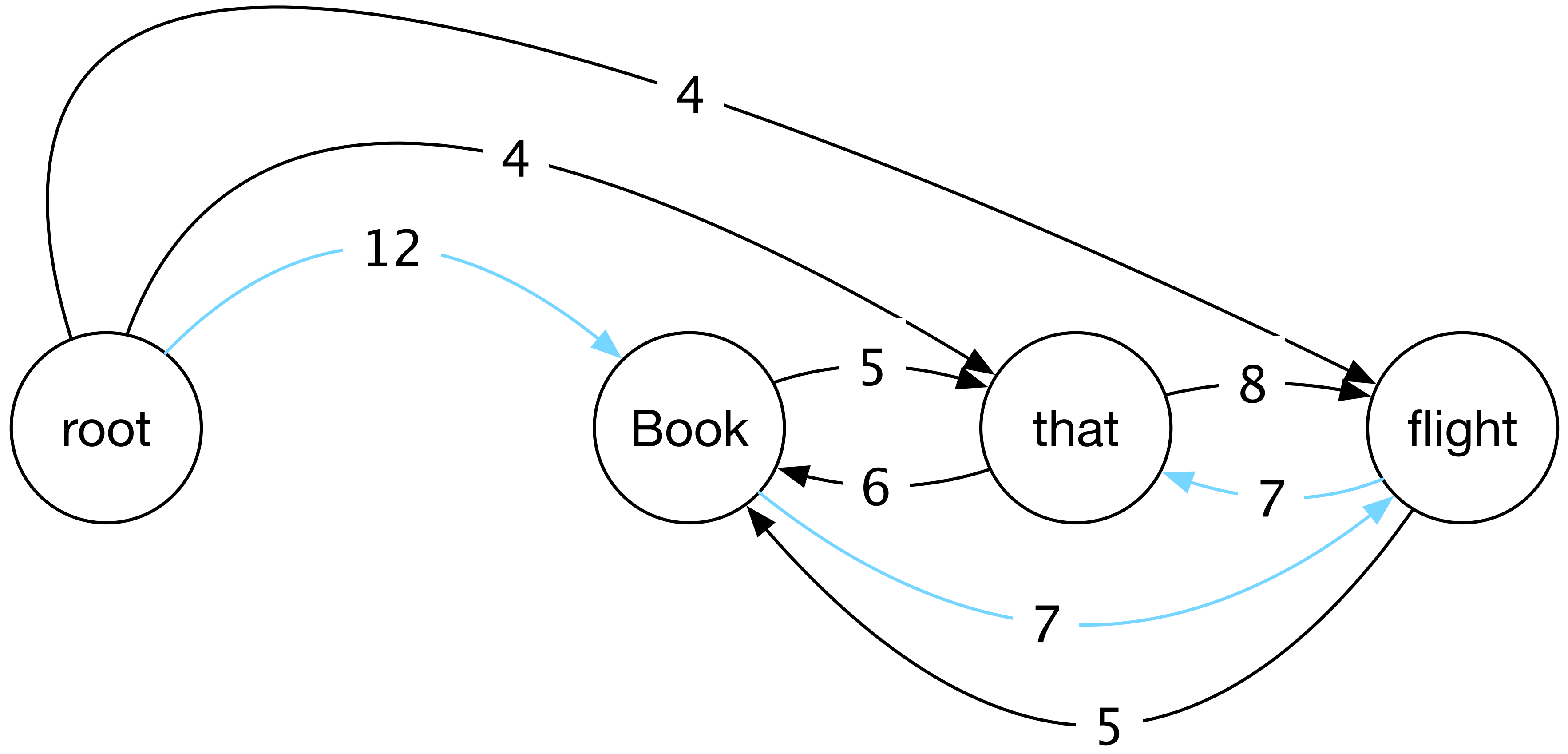
- Transition-based (shift-reduce) parsing:
    - Greedy choice of local transitions guided by a good classifier.
    - Examples: MaltParser [Nivre et al. 2008], Stack LSTM [Dyer et al. 2015]



Dependency parsing

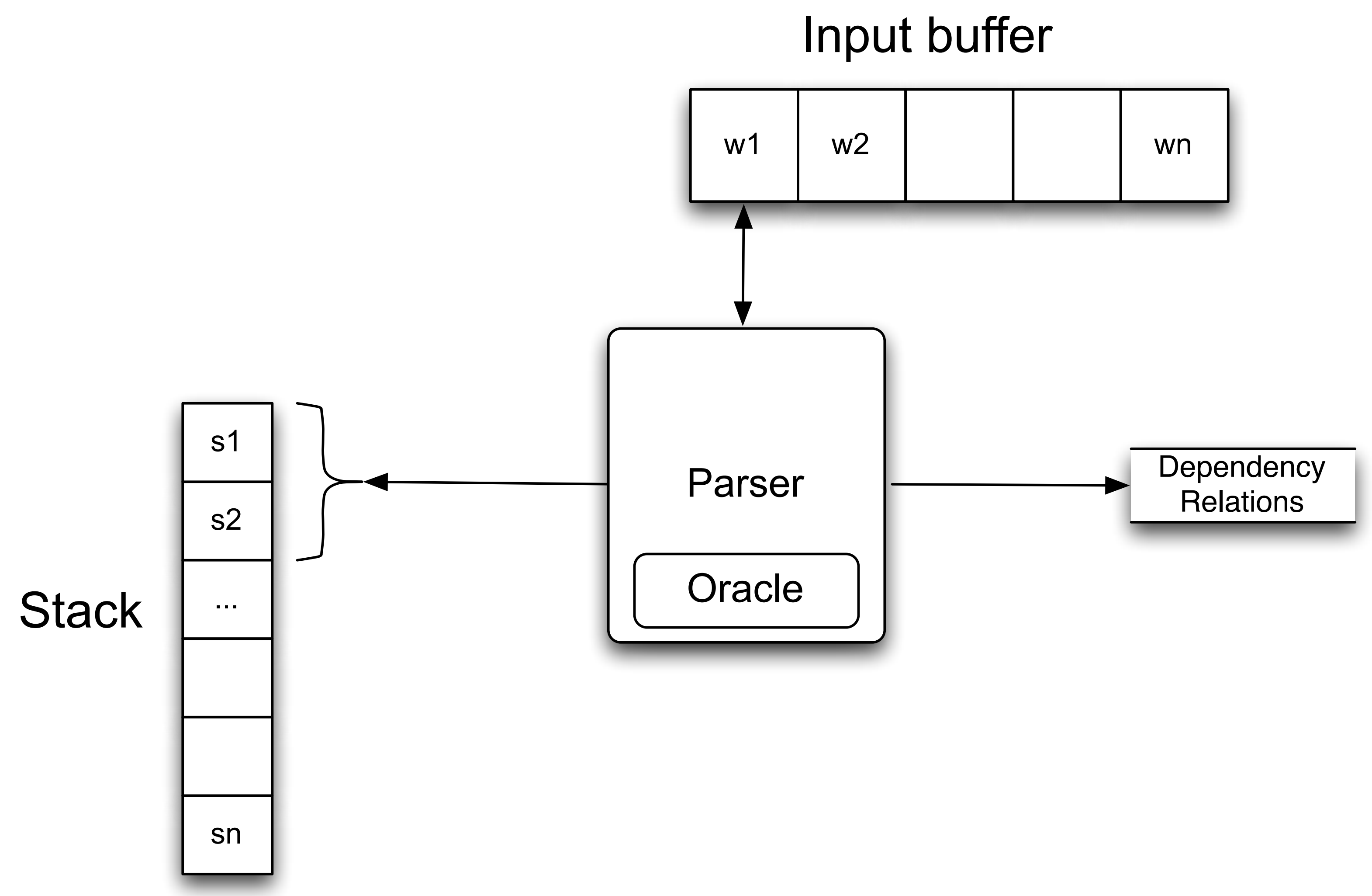
- Transition-based (shift-reduce) parsing:
  - Greedy choice of local transitions guided by a good classifier.
  - Examples: MaltParser [Nivre et al. 2008], Stack LSTM [Dyer et al. 2015]
- Graph-based dependency parsing:



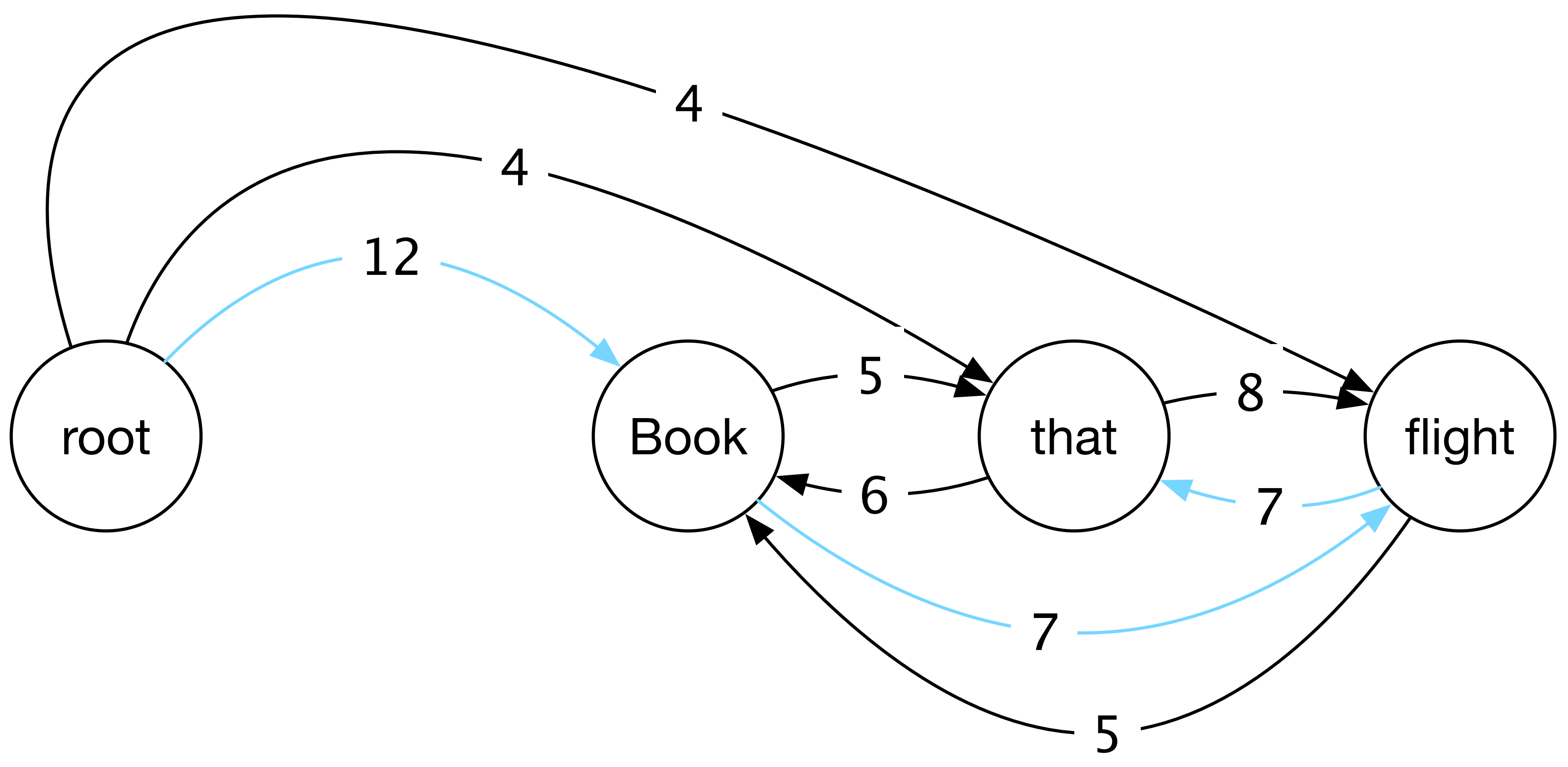


Dependency parsing

- Transition-based (shift-reduce) parsing:
    - Greedy choice of local transitions guided by a good classifier.
    - Examples: MaltParser [Nivre et al. 2008], Stack LSTM [Dyer et al. 2015]

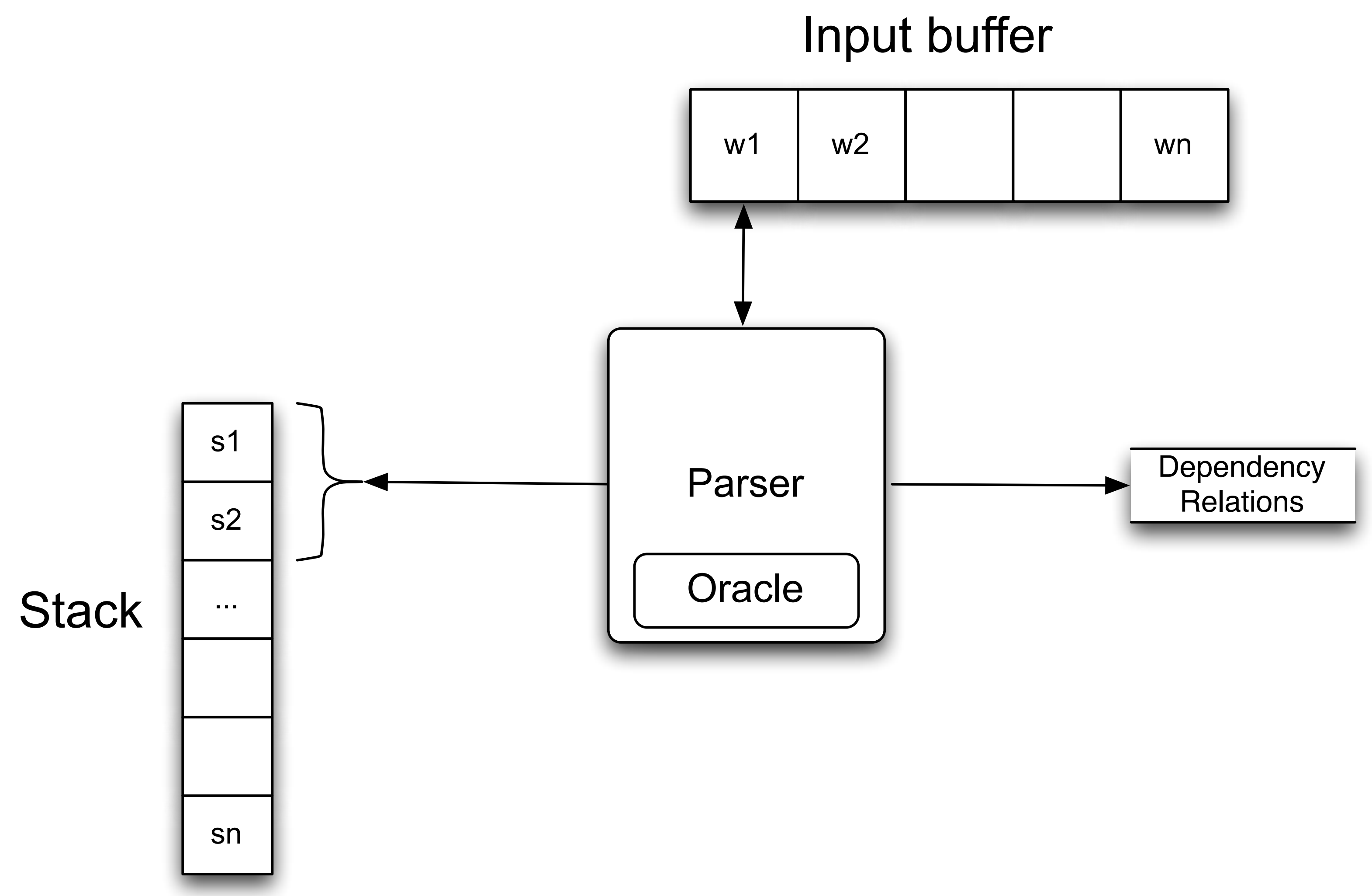


- Graph-based dependency parsing:
    - Given scores for every pair of words, find the (globally) highest scoring set of edges.

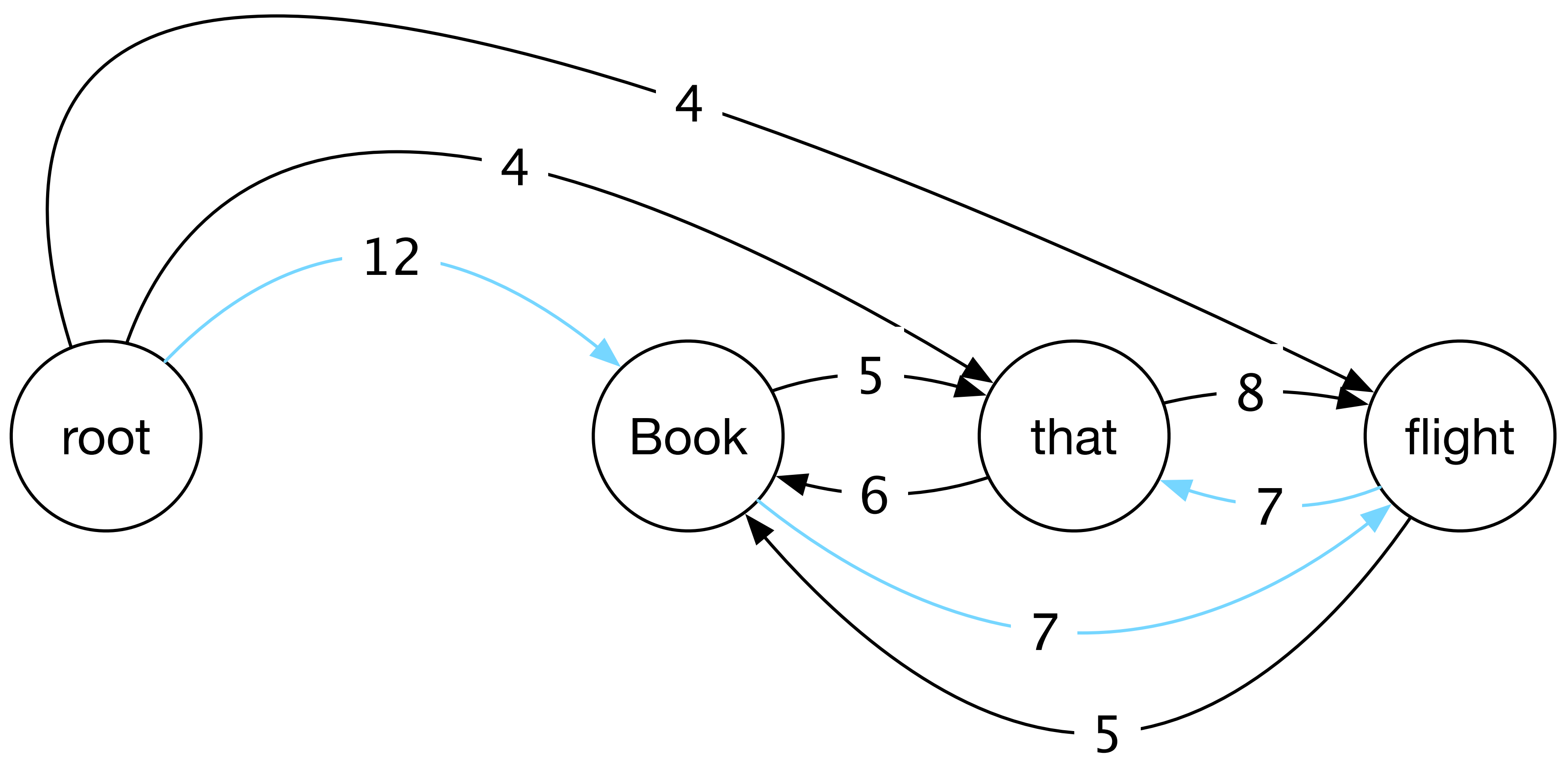


Dependency parsing

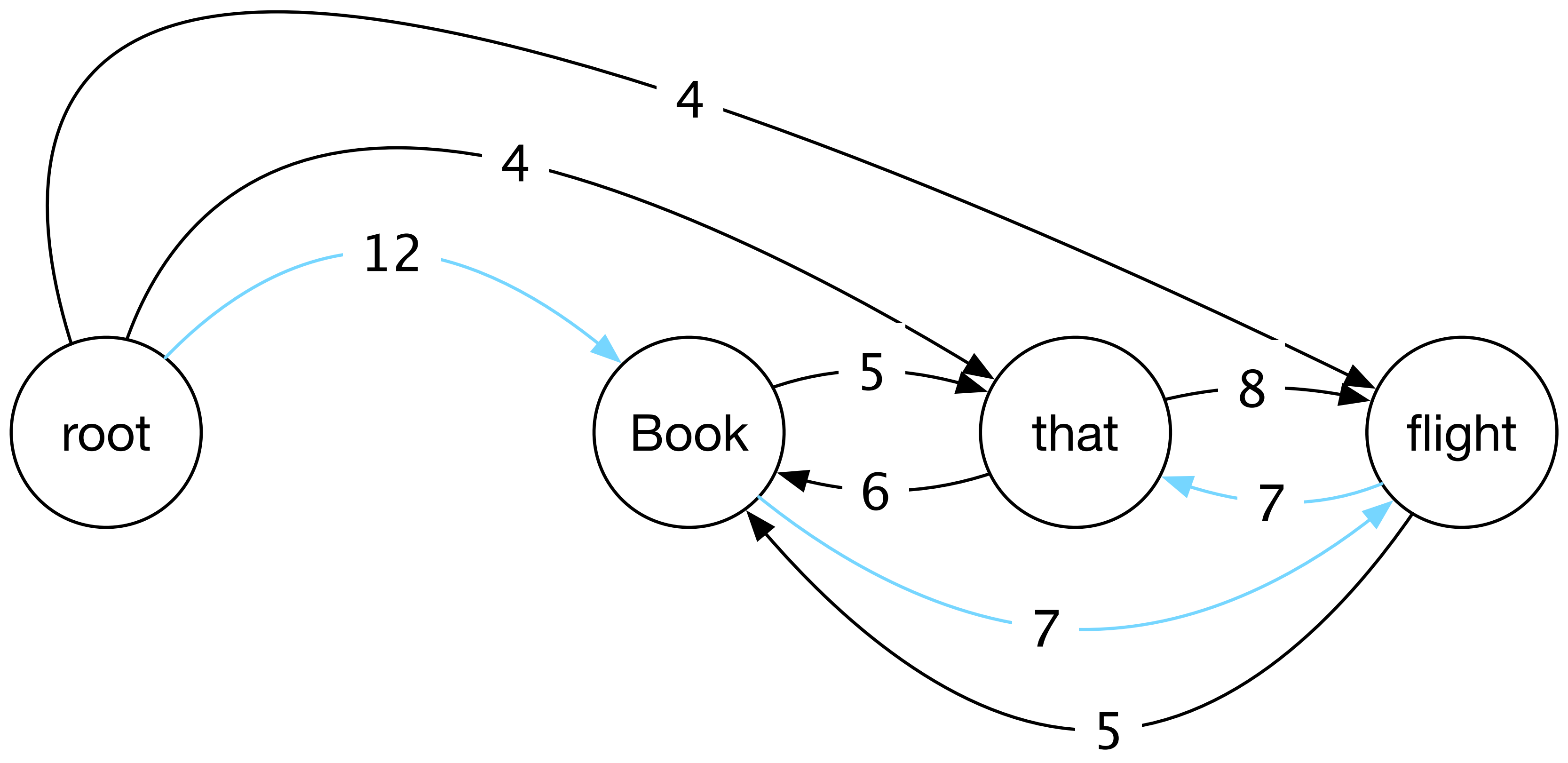
- Transition-based (shift-reduce) parsing:
  - Greedy choice of local transitions guided by a good classifier.
  - Examples: MaltParser [Nivre et al. 2008], Stack LSTM [Dyer et al. 2015]



- Graph-based dependency parsing:
  - Given scores for every pair of words, find the (globally) highest scoring set of edges.
  - Examples: MSTParser [McDonald et al. 2005], TurboParser [Martins et al. 2009], Deep Biaffine [Dozat et al. 2017]

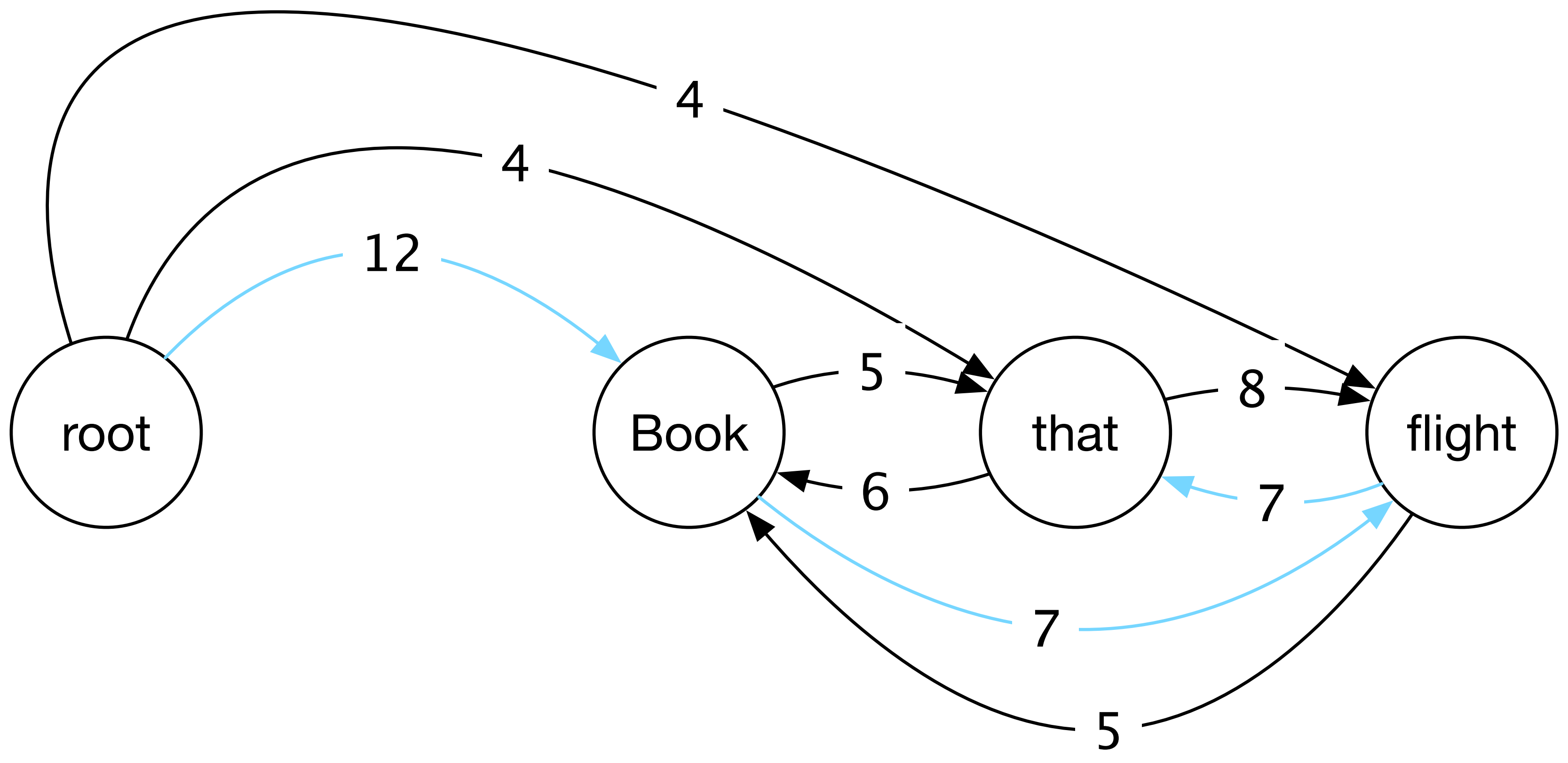


Graph-based dependency parsing



Graph-based dependency parsing

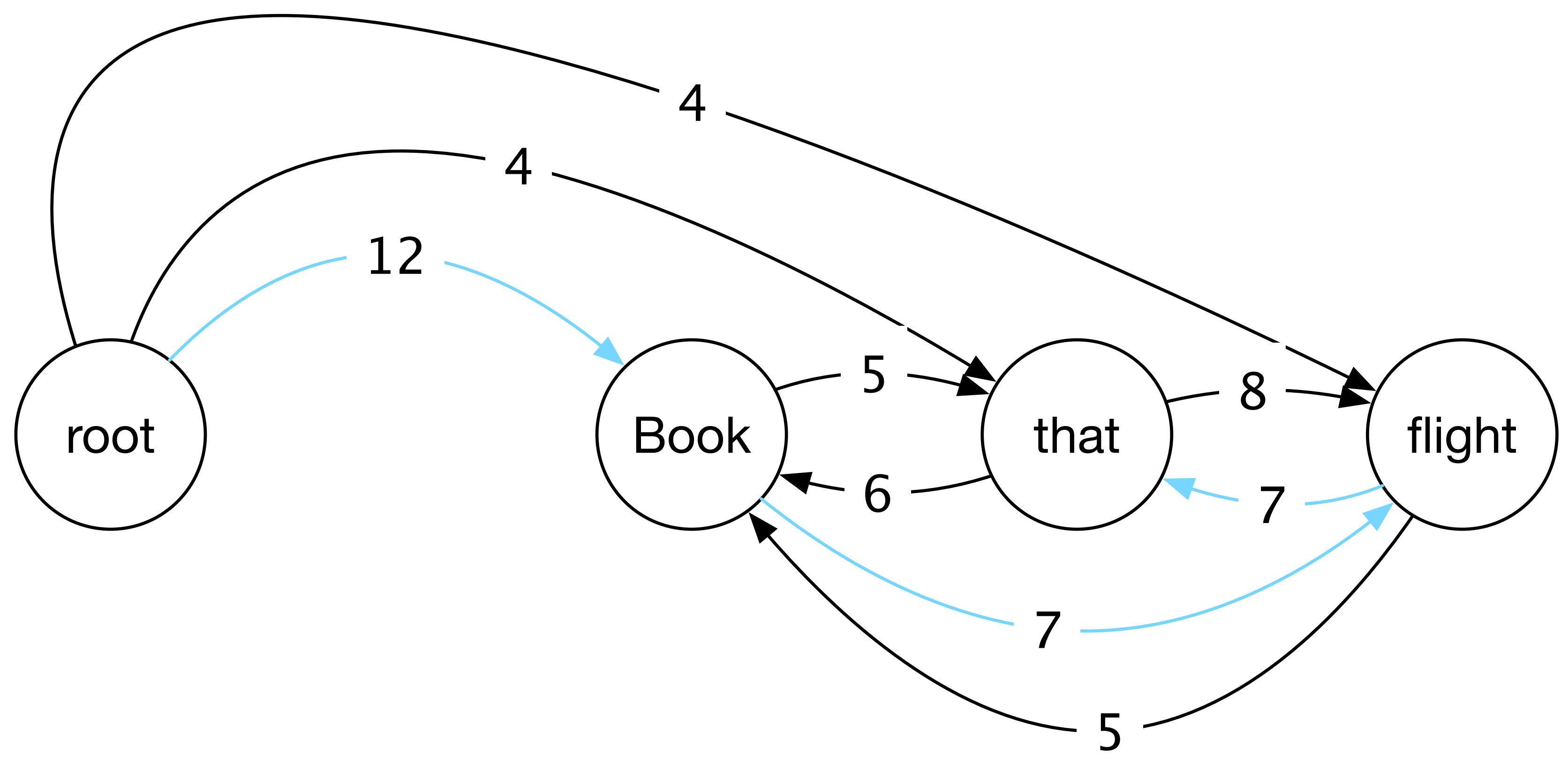
- **Edge-factored** (or **arc-factored**) approaches:



Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches:
  - Score of a tree decomposes as sum of edge scores:

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

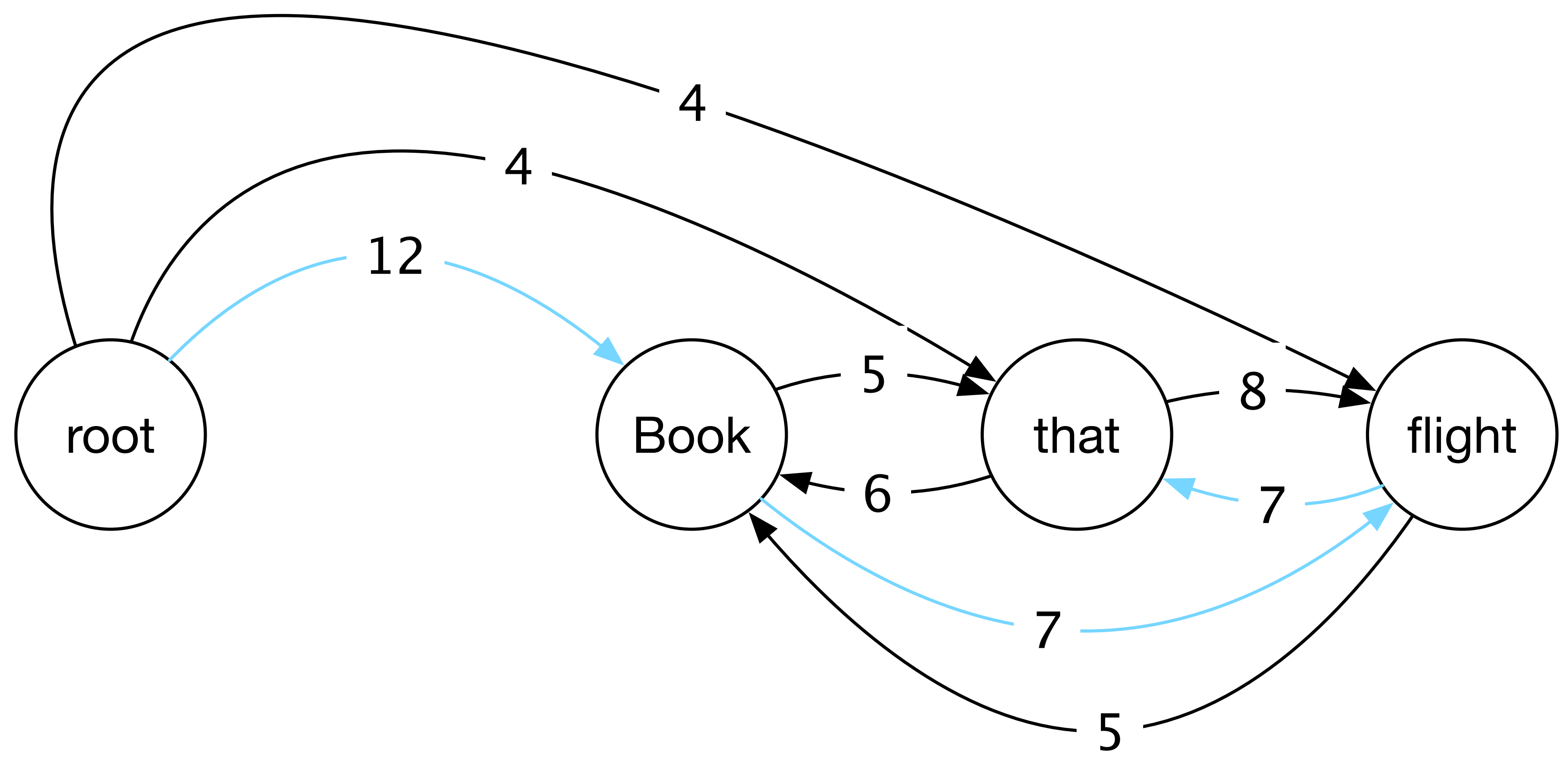


Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches:
  - Score of a tree decomposes as sum of edge scores:

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

  - Start with a fully-connected directed graph

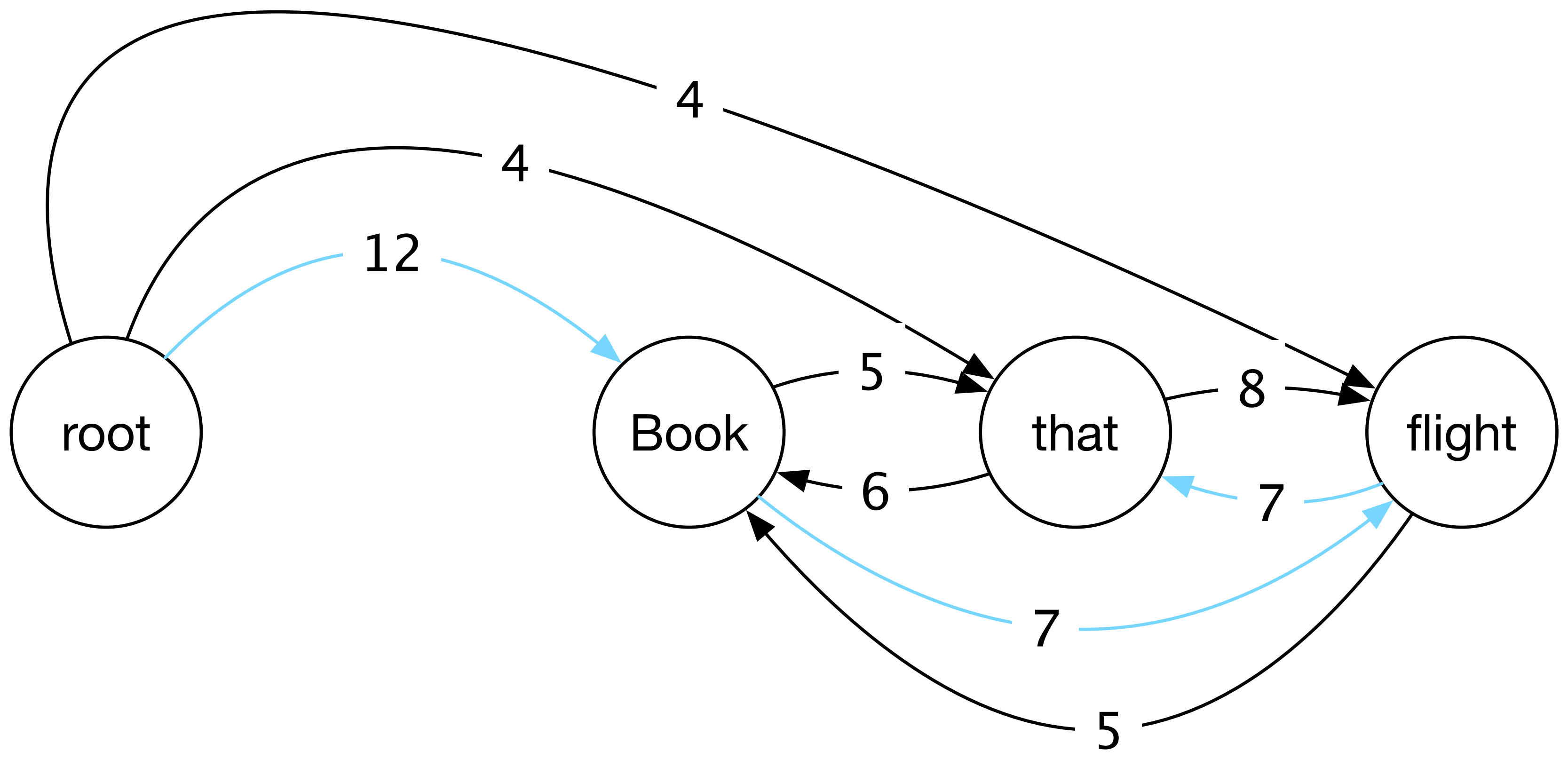


Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches:
  - Score of a tree decomposes as sum of edge scores:

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

  - Start with a fully-connected directed graph
  - How to infer the highest scoring tree?

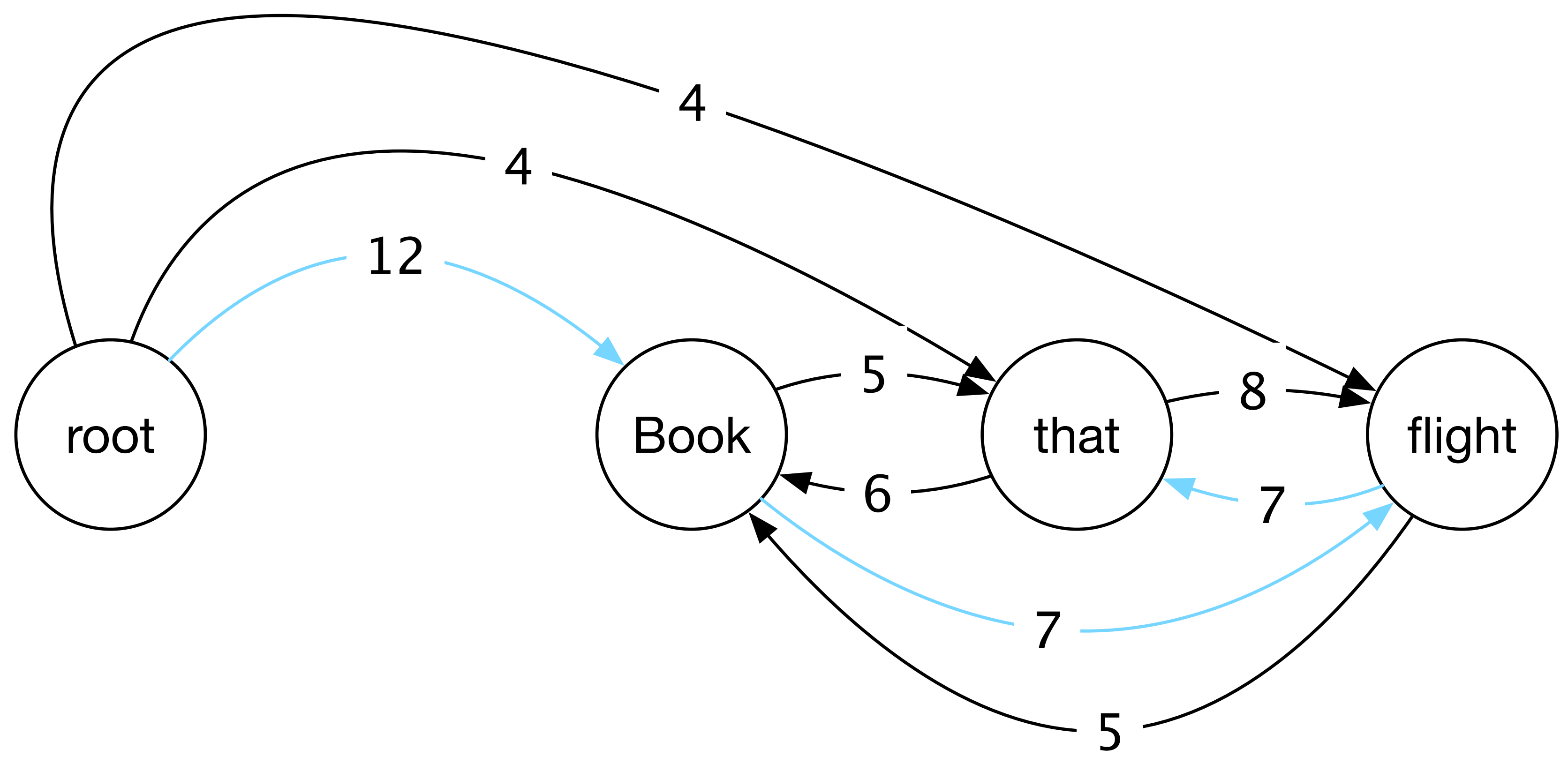


Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches:
  - Score of a tree decomposes as sum of edge scores:

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

  - Start with a fully-connected directed graph
  - How to infer the highest scoring tree?
  - Find a **maximum directed spanning tree**:
    Chu and Liu (1965) and Edmonds (1967) algorithm



Chu-Liu-Edmonds 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*

Chu-Liu-Edmonds 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*

select best incoming edge for each node

Chu-Liu-Edmonds 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*

select best incoming edge for each node

subtract its score from all incoming edges

Chu-Liu-Edmonds algorithm

function MAXSPANNINGTREE(G=(V,E), *root*, *score*) **returns** *spanning tree*

$F \leftarrow []$
$T' \leftarrow []$
$score' \leftarrow []$
for each $v \in V$ **do**
  $bestInEdge \leftarrow \text{argmax}_{e=(u,v)\in E}\ score[e]$ — select best incoming edge for each node
  $F \leftarrow F \cup bestInEdge$
  for each $e=(u,v) \in E$ **do** — subtract its score from all incoming edges
    $score'[e] \leftarrow score[e] - score[bestInEdge]$

  if $T=(V,F)$ is a spanning tree **then return** it — stopping condition
  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*

Chu-Liu-Edmonds algorithm

function MAXSPANNINGTREE(*G=(V,E)*, *root*, *score*) **returns** *spanning tree*

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

select best incoming edge for each node

subtract its score from all incoming edges

stopping condition

contract nodes if there are cycles

Chu-Liu-Edmonds algorithm

function MAXSPANNINGTREE(G=(V,E), $root$, $score$) **returns** *spanning tree*

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

select best incoming edge for each node

subtract its score from all incoming edges

stopping condition

contract nodes if there are cycles

recursively compute MST

Chu-Liu-Edmonds algorithm

function MAXSPANNINGTREE($G=(V,E)$, *root*, *score*) **returns** *spanning tree*

$F \leftarrow []$
$T' \leftarrow []$
$score' \leftarrow []$
for each $v \in V$ **do**
  $bestInEdge \leftarrow \text{argmax}_{e=(u,v) \in E}\ score[e]$ — select best incoming edge for each node
  $F \leftarrow F \cup bestInEdge$
  for each $e=(u,v) \in E$ **do** — subtract its score from all incoming edges
    $score'[e] \leftarrow score[e] - score[bestInEdge]$

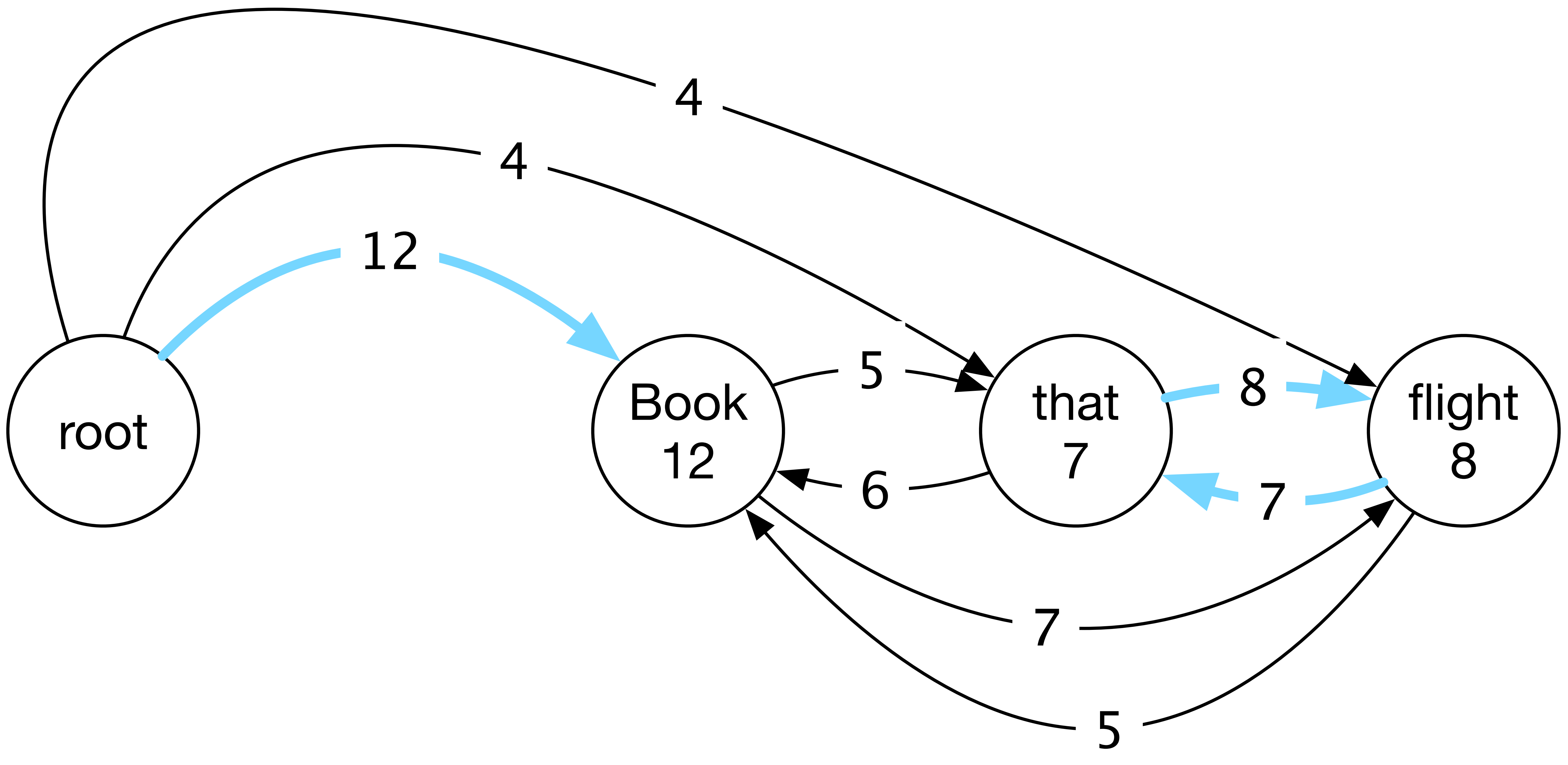
  if $T=(V,F)$ is a spanning tree **then return** it — stopping condition
  else
    $C \leftarrow$ a cycle in F — contract nodes if there are cycles
    $G' \leftarrow$ CONTRACT(G, C)
    $T' \leftarrow$ MAXSPANNINGTREE(G', *root*, *score'*) — recursively compute MST
    $T \leftarrow$ EXPAND(T', C) — expand contracted nodes
    return T

function CONTRACT(G, C) **returns** *contracted graph*

function EXPAND(T, C) **returns** *expanded graph*

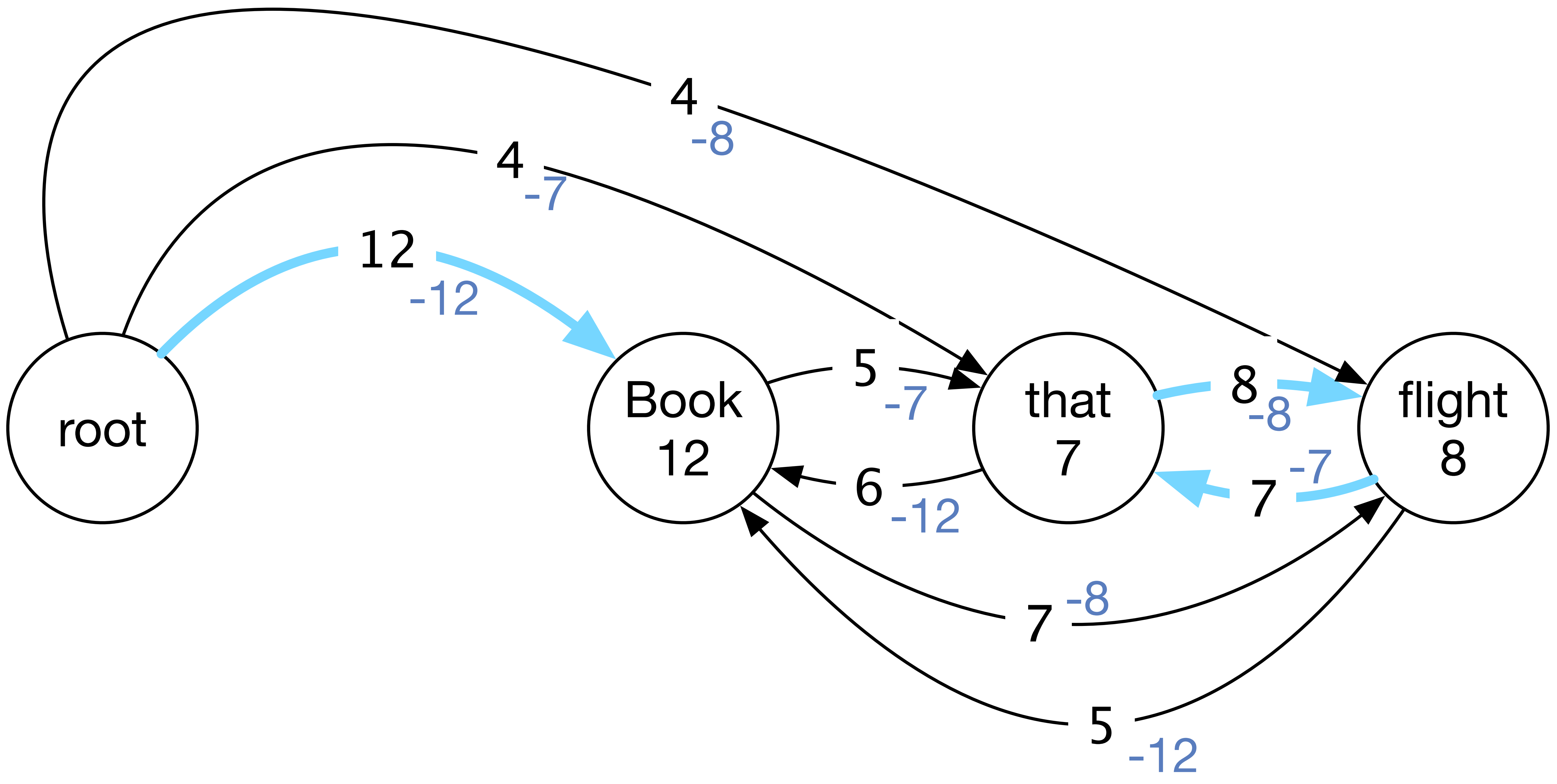
Chu-Liu-Edmonds algorithm

- Select best incoming edge for each node



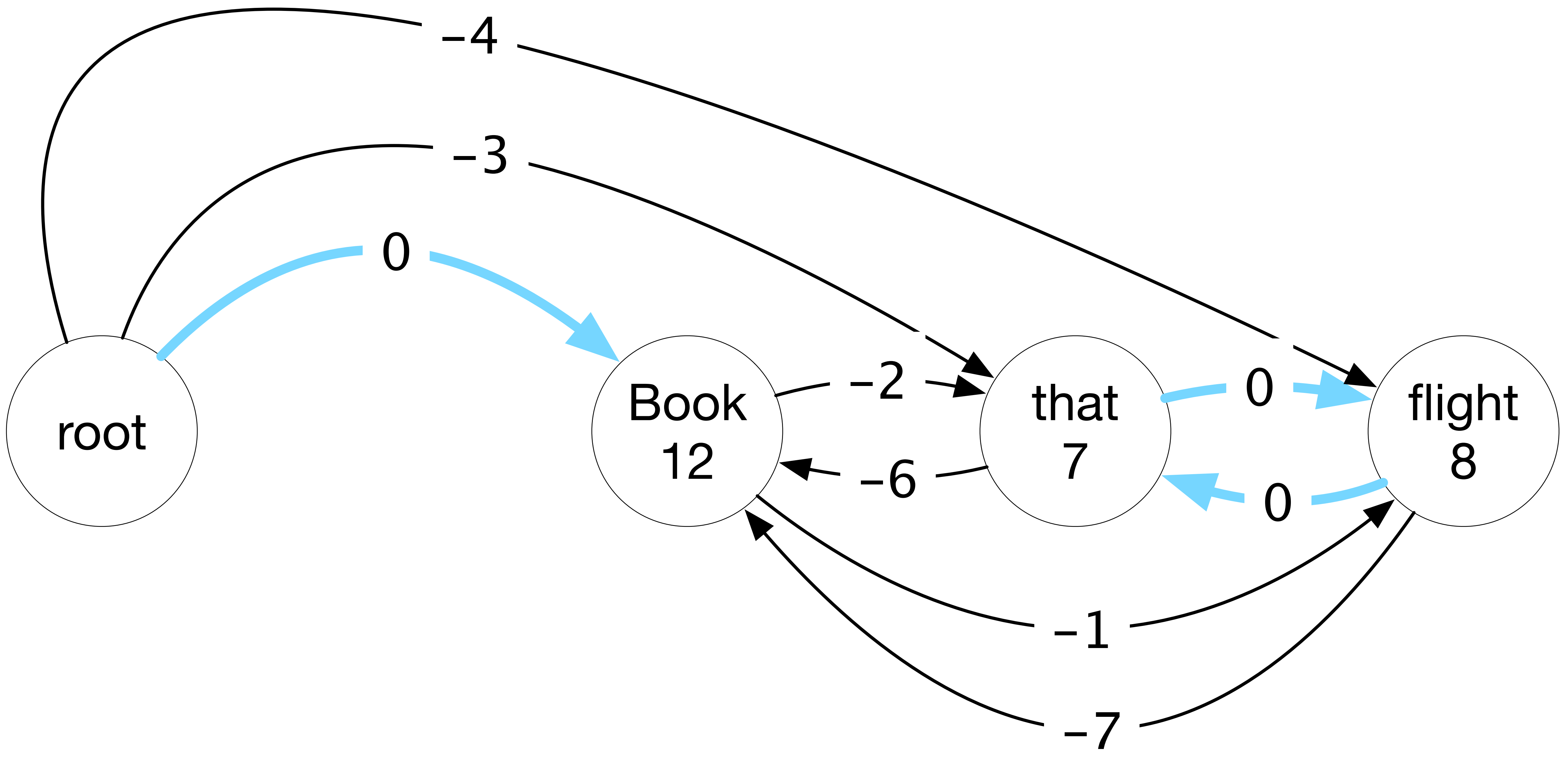
Chu-Liu-Edmonds algorithm

- Subtract its score from all incoming edges



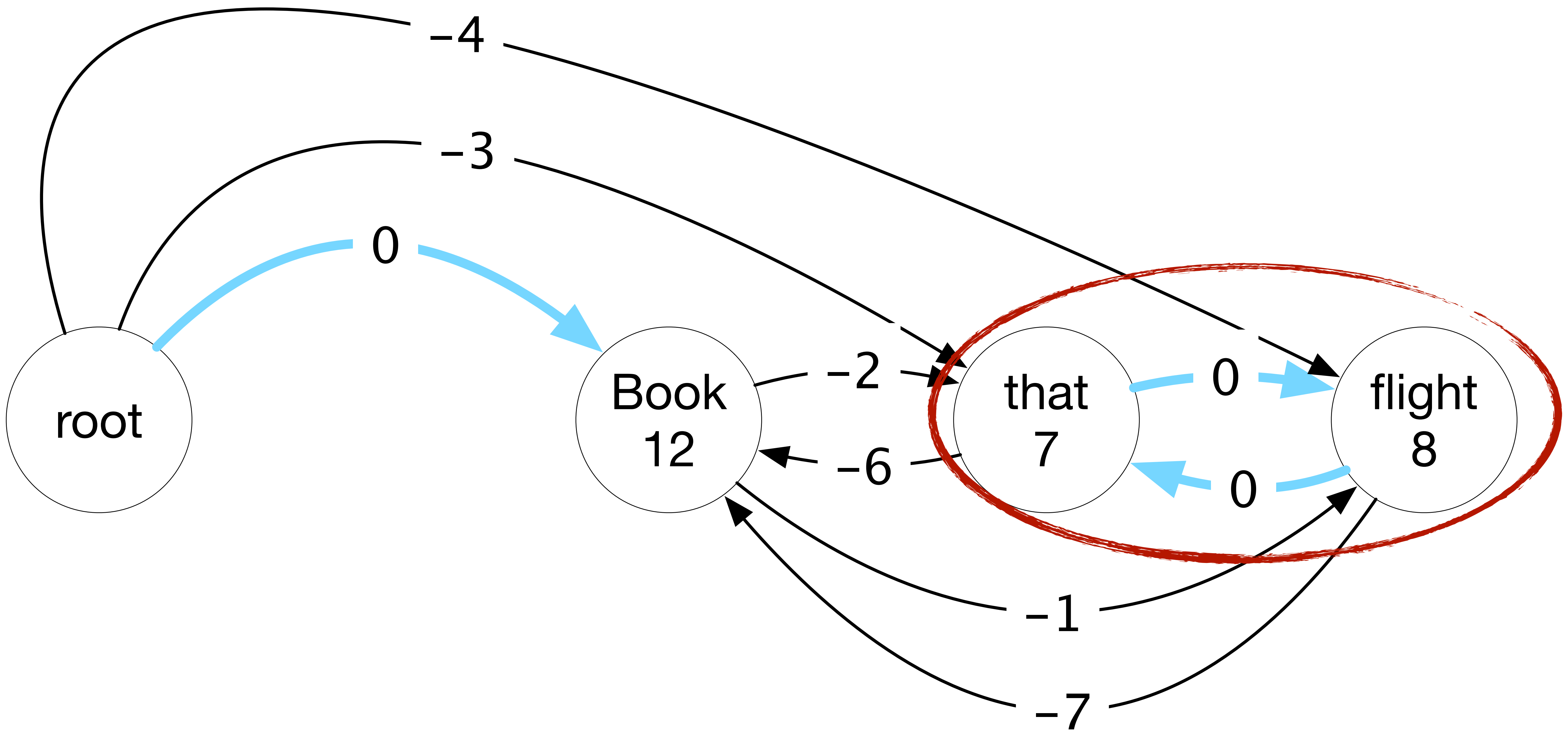
Chu-Liu-Edmonds algorithm

- Subtract its score from all incoming edges



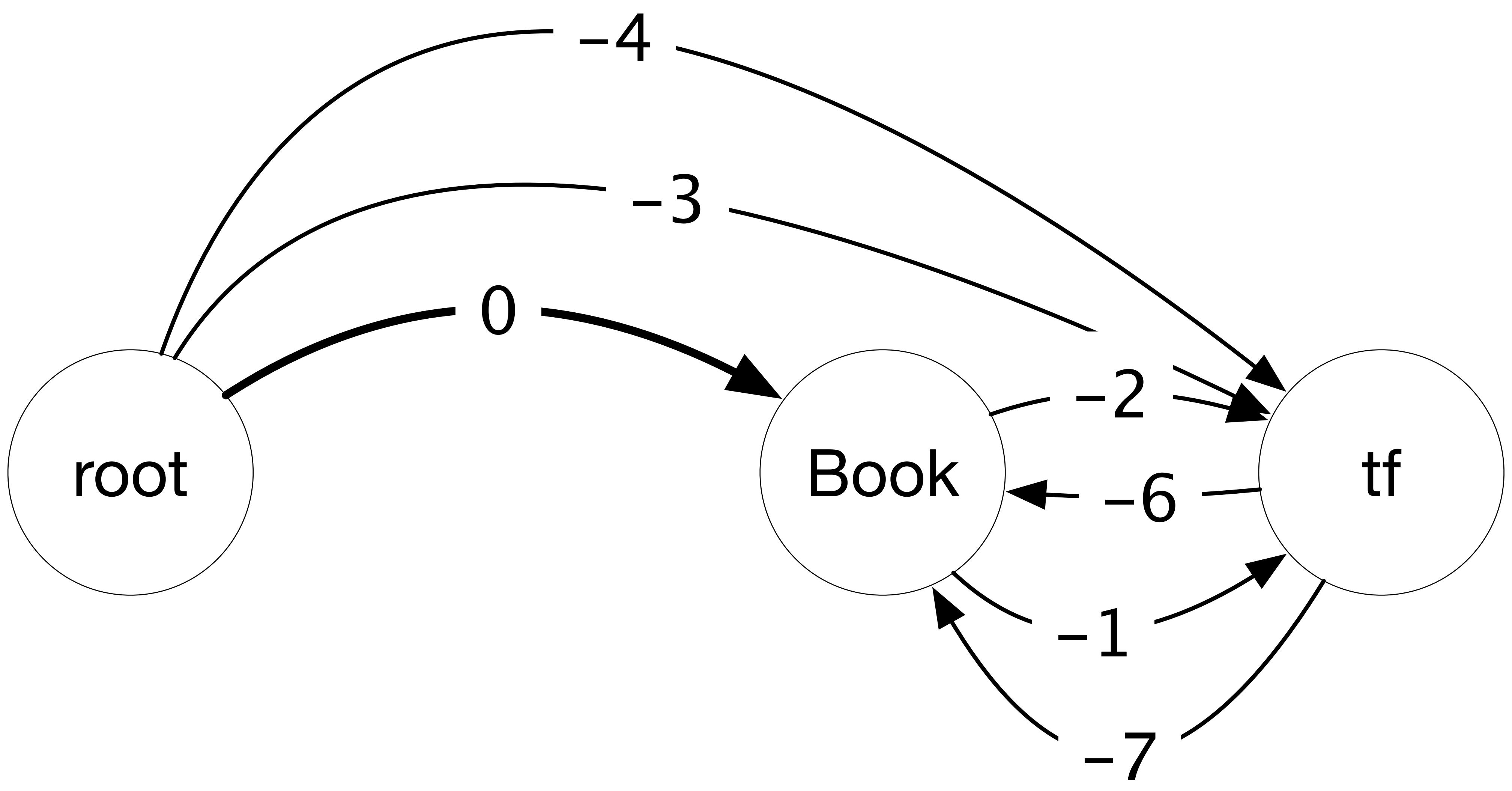
Chu-Liu-Edmonds algorithm

- Contract nodes if there are cycles



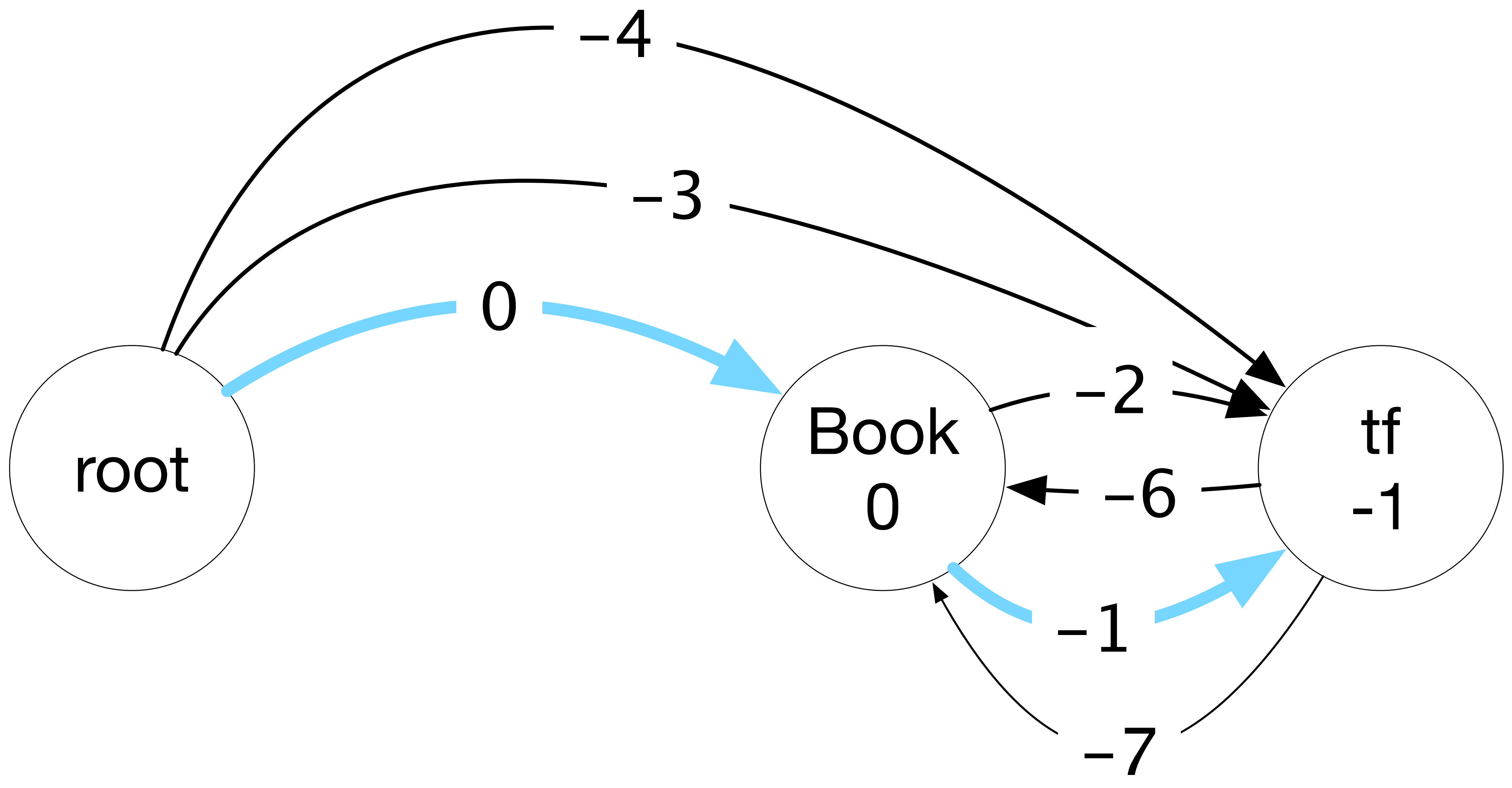
Chu-Liu-Edmonds algorithm

- Contract nodes if there are cycles



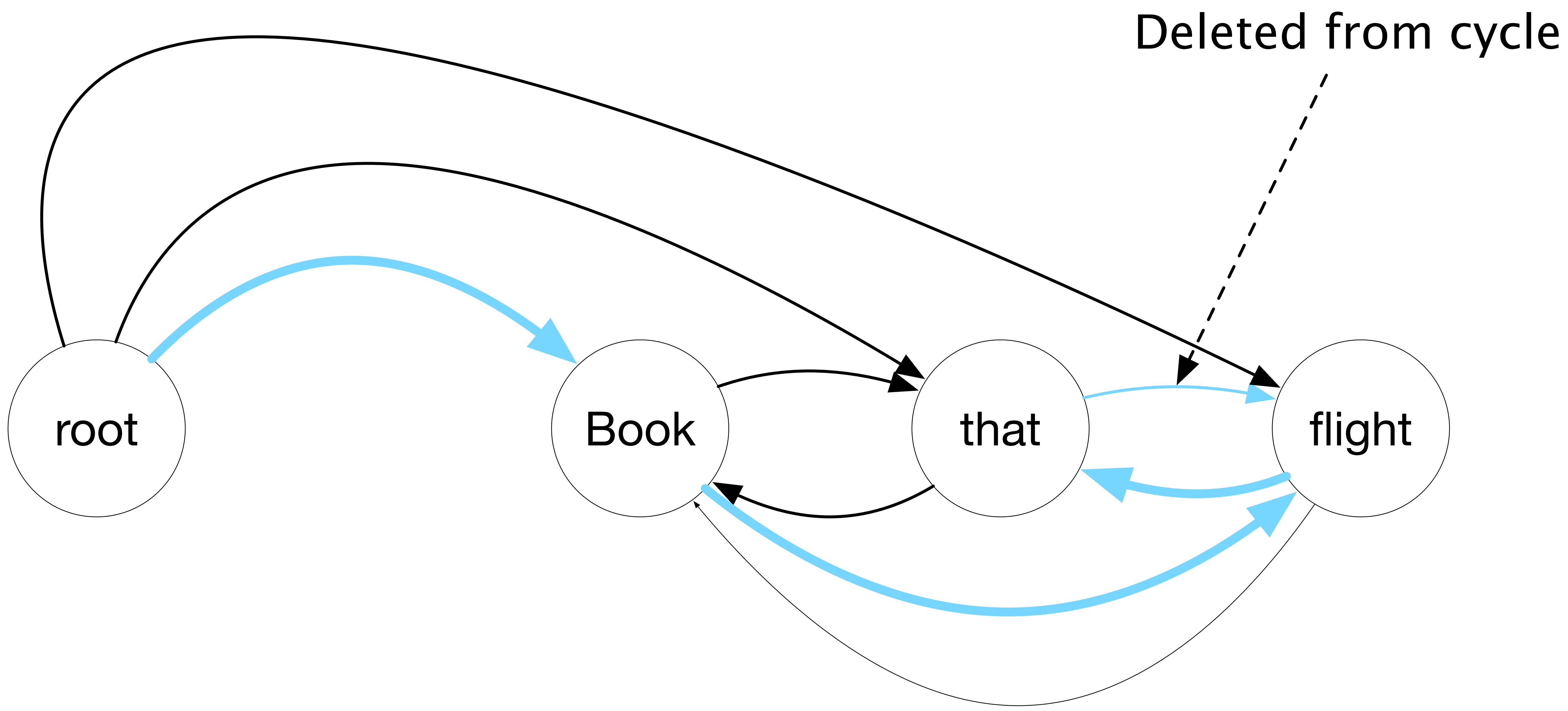
Chu-Liu-Edmonds algorithm

- Recursively compute MST



Chu-Liu-Edmonds algorithm

- Expand contracted nodes



Chu-Liu-Edmonds 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*

runtime?

Chu-Liu-Edmonds algorithm

function MAXSPANNINGTREE(G=(V,E), *root*, *score*) **returns** *spanning tree*

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

runtime?   naive: $O(n^3)$

Chu-Liu-Edmonds algorithm

function MAXSPANNINGTREE(*G=(V,E)*, *root*, *score*) **returns** *spanning tree*

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

runtime? naive: $O(n^3)$

fancy: $O(n^2 + n\log n)$

Chu-Liu-Edmonds 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*

runtime? naive: $O(n^3)$

fancy: $O(n^2 + n\log n)$

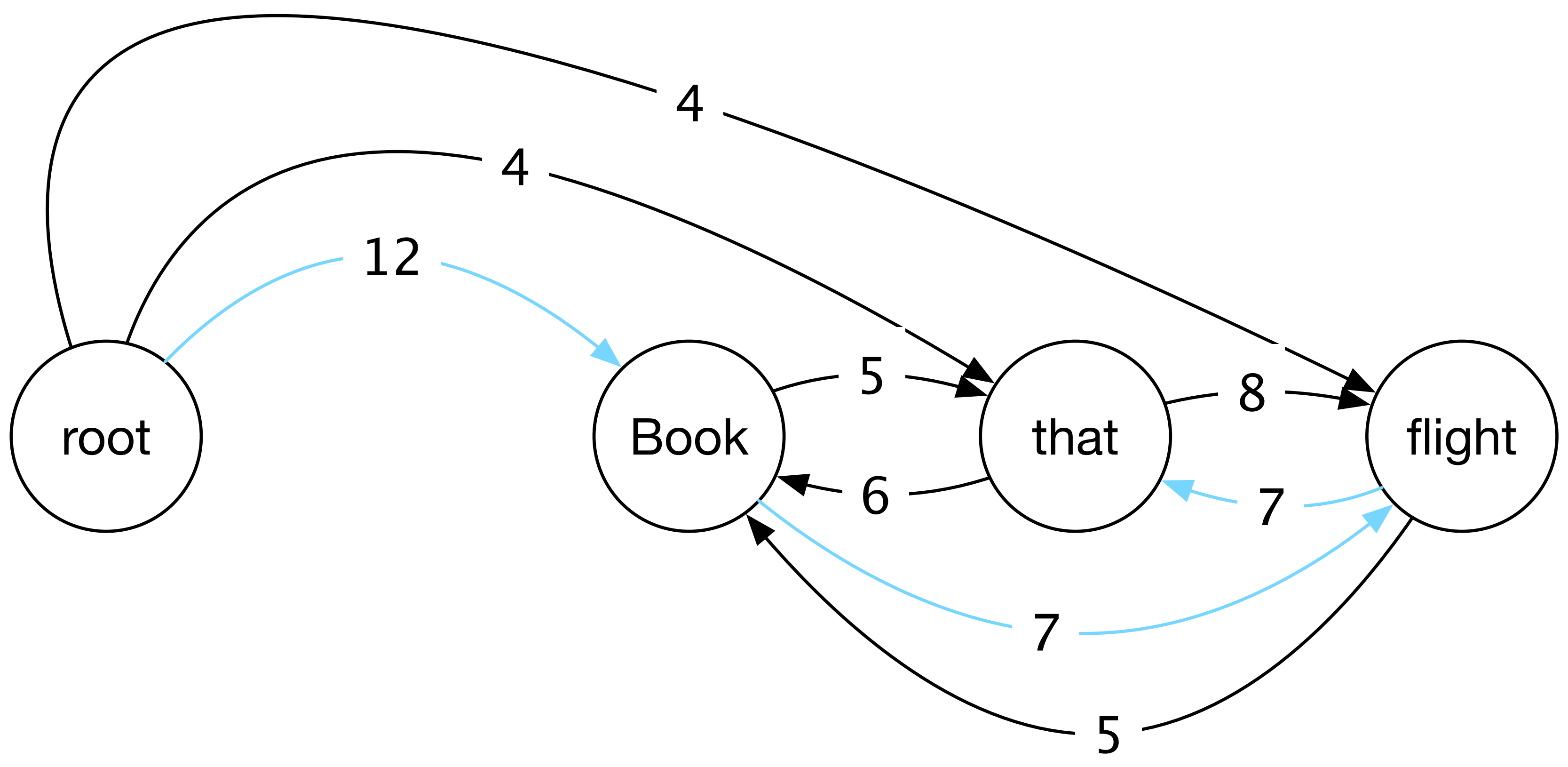
what about labeled parsing?

Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches:
  - Score of a tree decomposes as sum of edge scores:

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

  - Start with a fully-connected directed graph
  - How to infer the highest scoring tree?
  - Find a **maximum directed spanning tree**:
    Chu and Liu (1965) and Edmonds (1967) algorithm

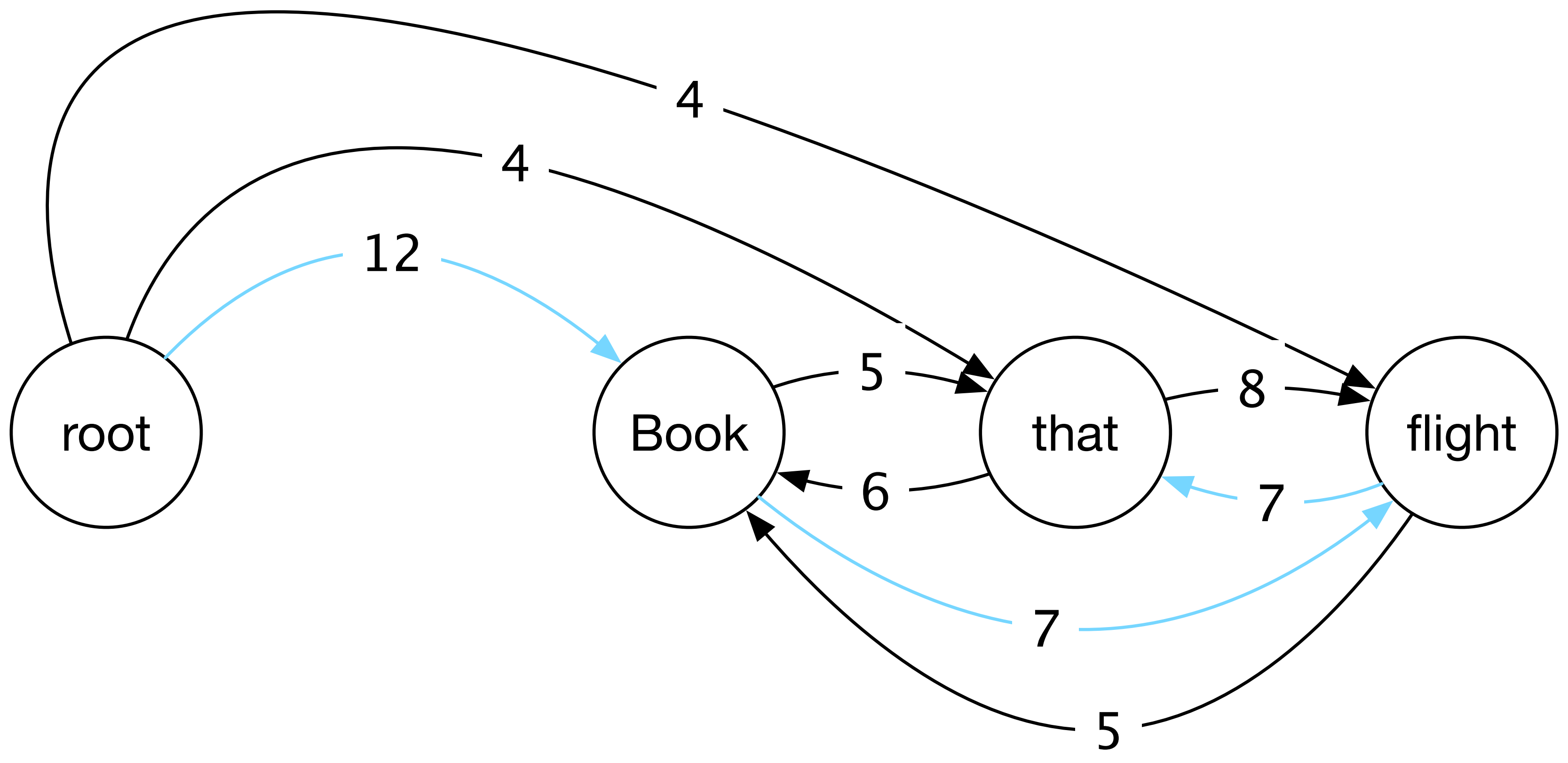


Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches:
  - Score of a tree decomposes as sum of edge scores:

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

  - Start with a fully-connected directed graph
  - How to infer the highest scoring tree?
  - Find a **maximum directed spanning tree**:
    Chu and Liu (1965) and Edmonds (1967) algorithm
  - For projective trees: **Eisner's algorithm** [Eisner 1996]



Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

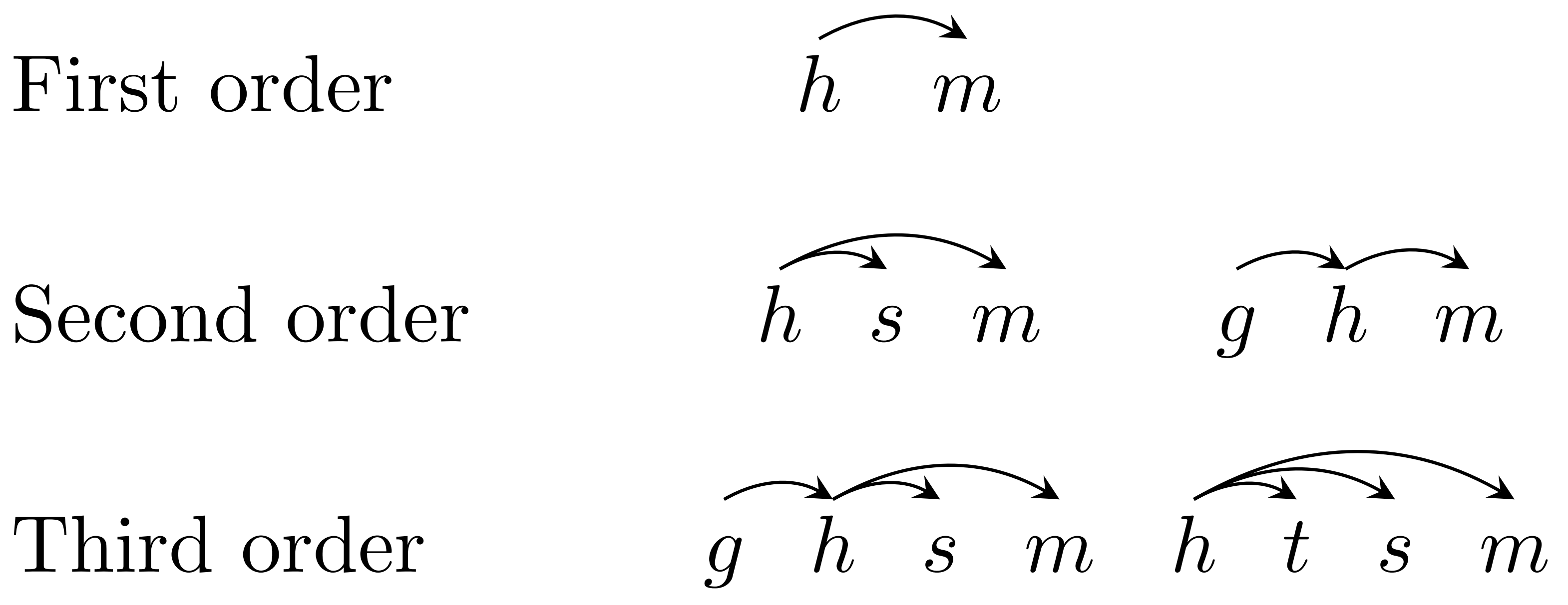
$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

- Can also define **higher-order** models: score decomposes as a sum of scores of local subgraphs.

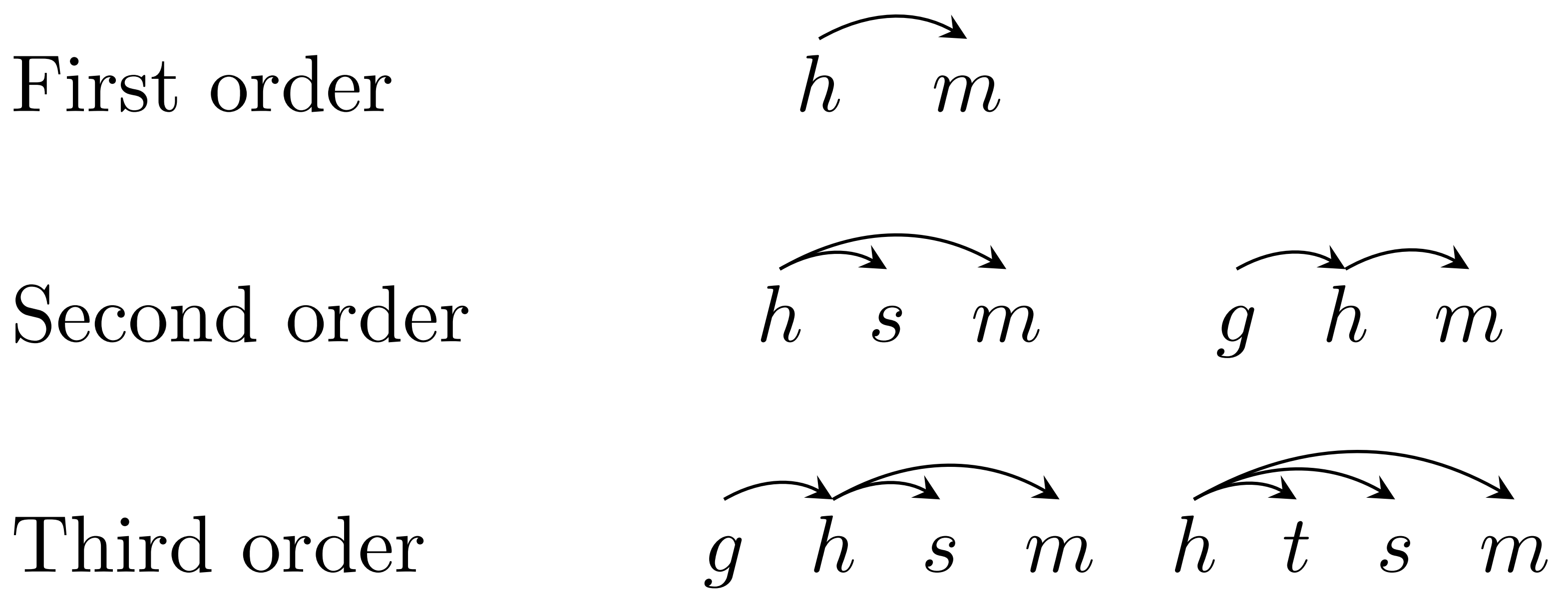


Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

- Can also define **higher-order** models: score decomposes as a sum of scores of local subgraphs.



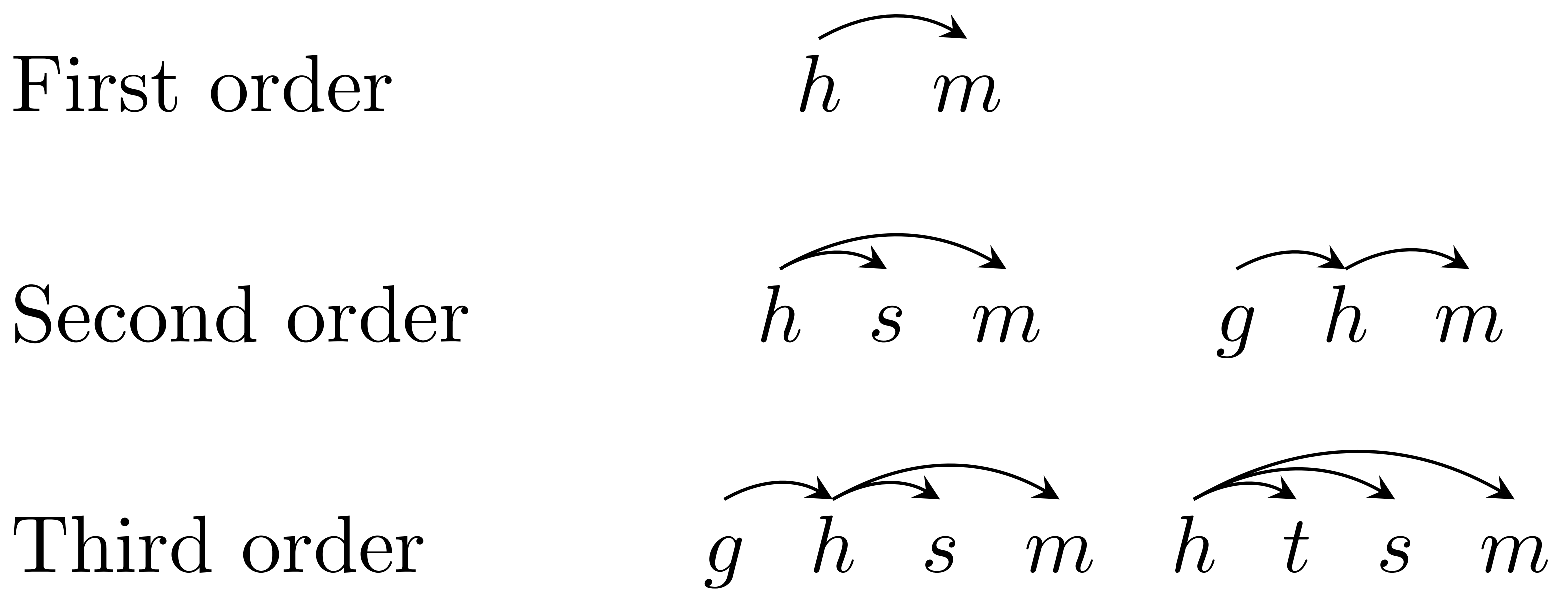
  - Have efficient (polynomial time) algorithms for second [Eisner 1996] and third order [Koo and Collins, 2010] *projective* dependency parsing.

Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

- Can also define **higher-order** models: score decomposes as a sum of scores of local subgraphs.



  - Have efficient (polynomial time) algorithms for second [Eisner 1996] and third order [Koo and Collins, 2010] *projective* dependency parsing.
  - Second order parsing is NP-hard for *non-projective* dependency graphs [McDonald and Pereira, 2006]!

Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

First order $\overset{\curvearrowright}{h \quad m}$

Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

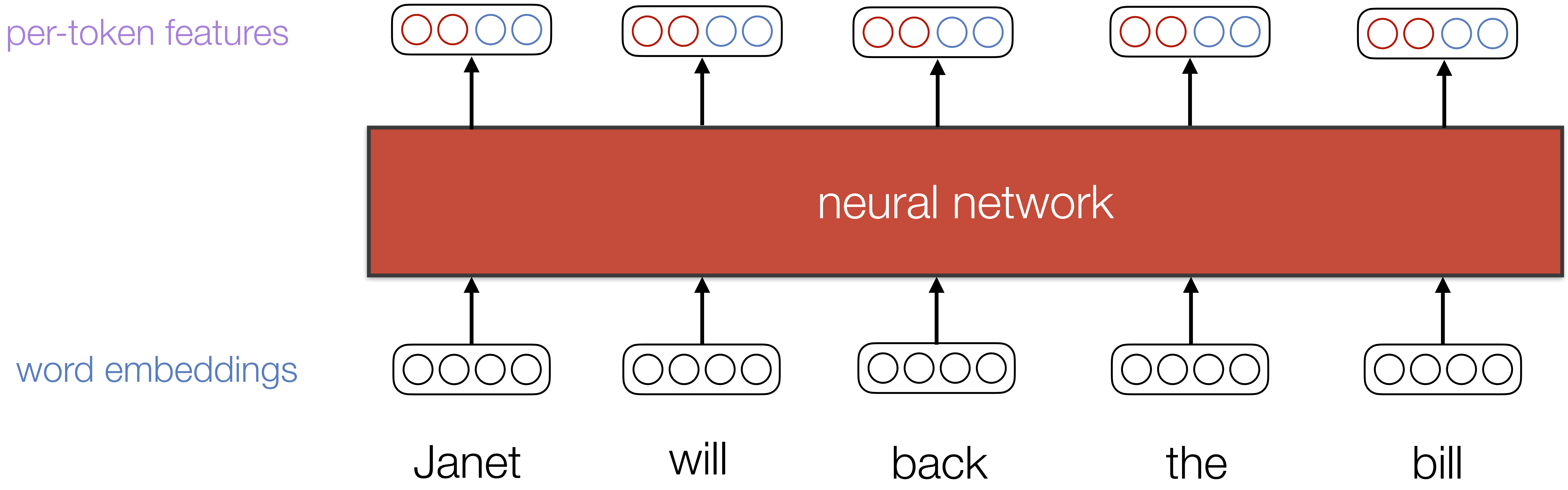
First order $h \qquad m$

- How to parameterize ψ ?

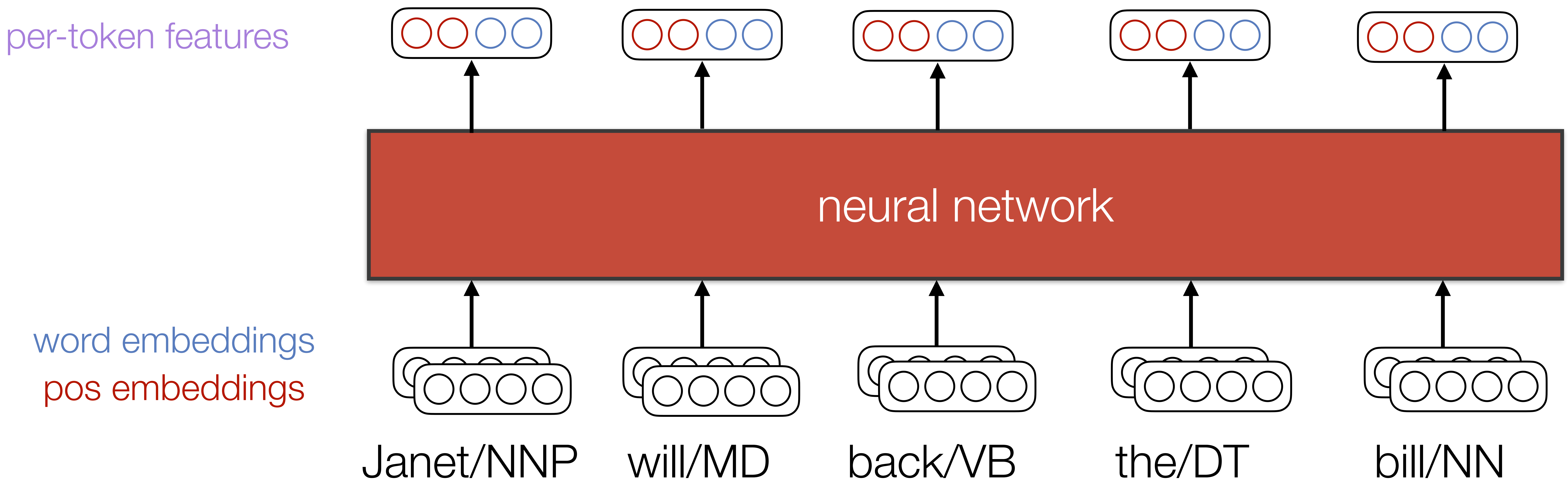
Classic graph-based parsing features

- Word forms, lemmas, parts of speech of the head word and its dependent
- Corresponding features derived from the contexts before, after, between words
- Word embeddings
- Dependency relation
- Direction of the relation (right or left)
- Distance from the head to the dependent
- Combinations of all of the above

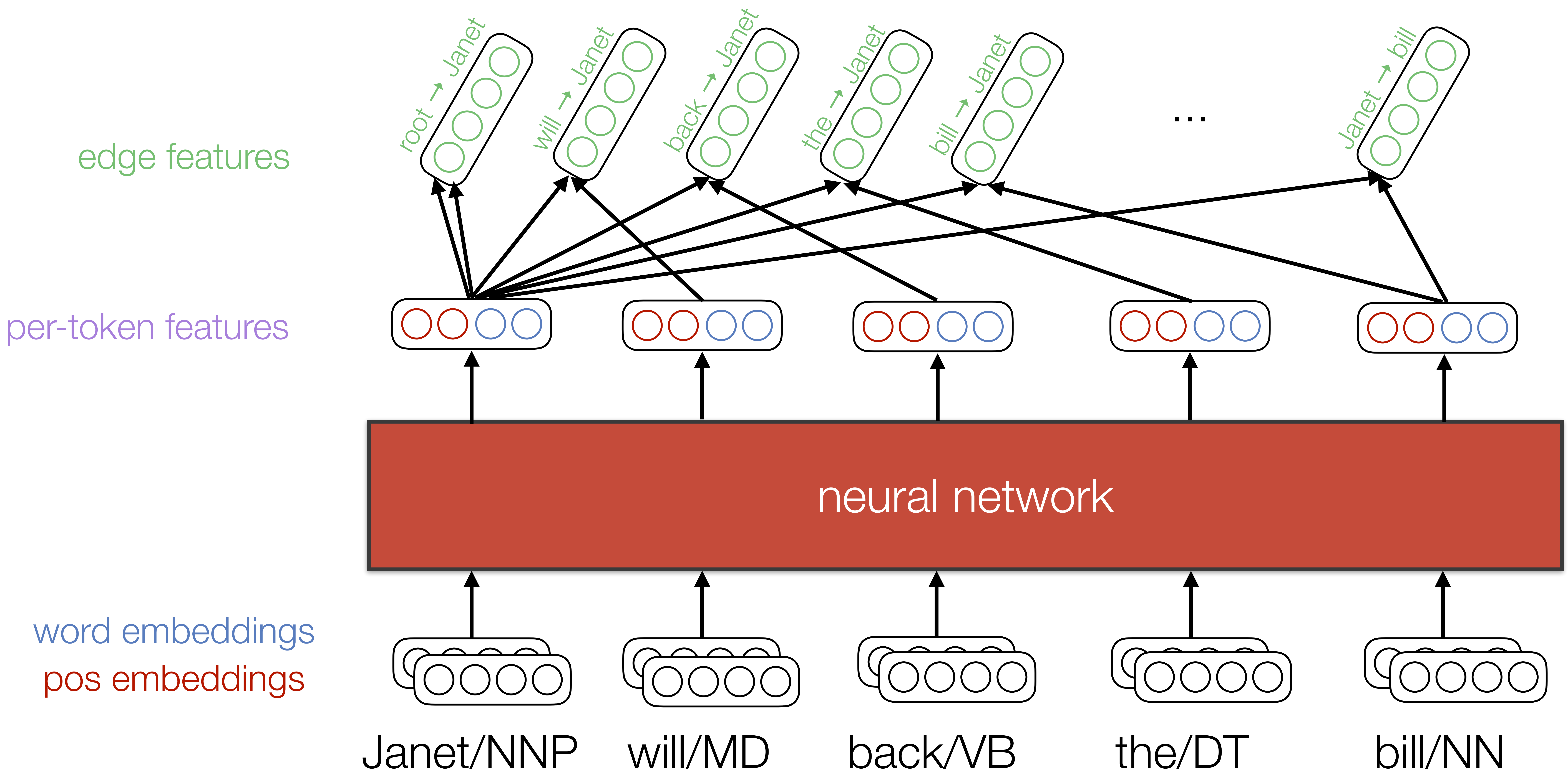
Graph-based neural network parsers



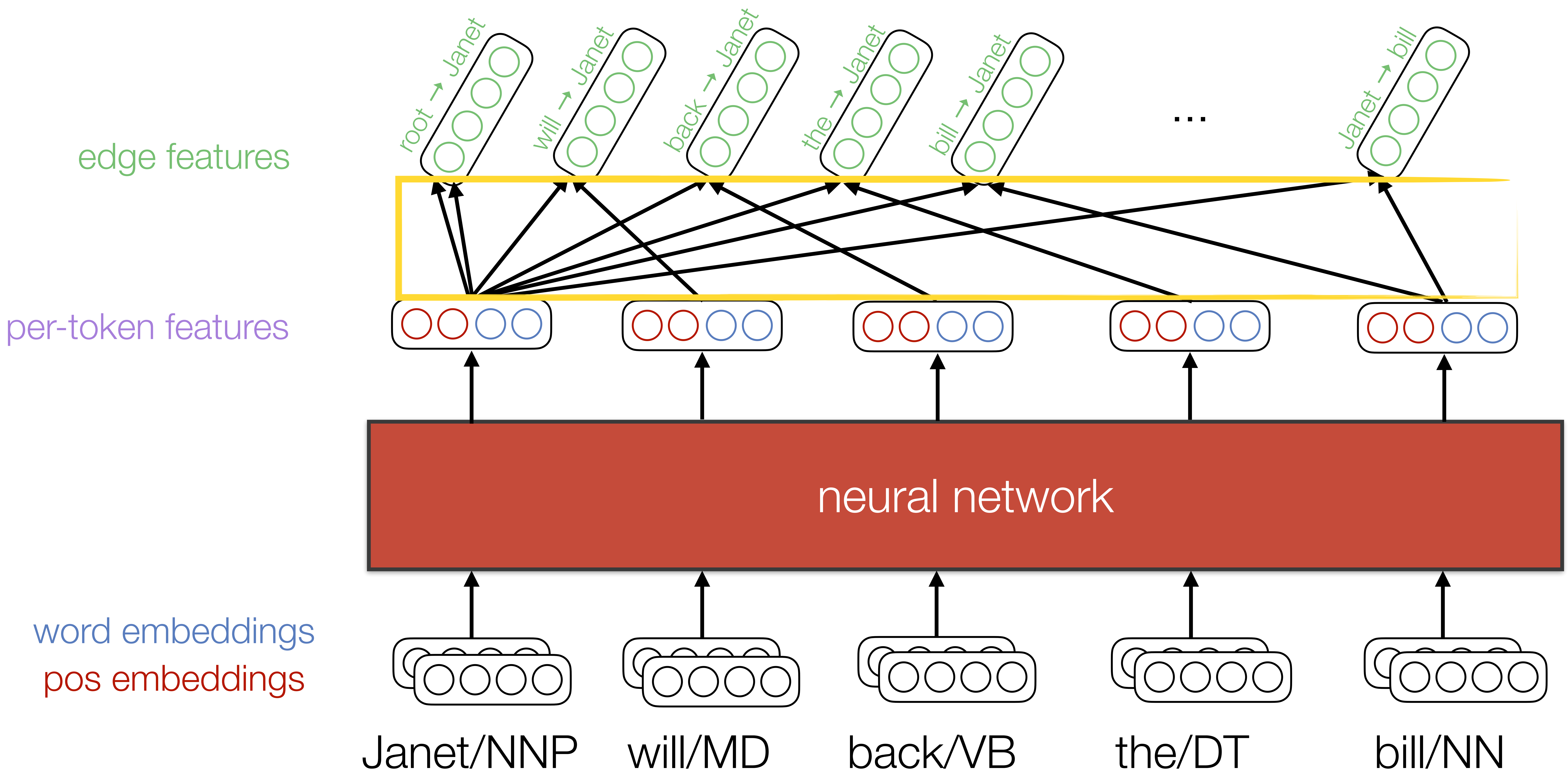
Graph-based neural network parsers



Graph-based neural network parsers



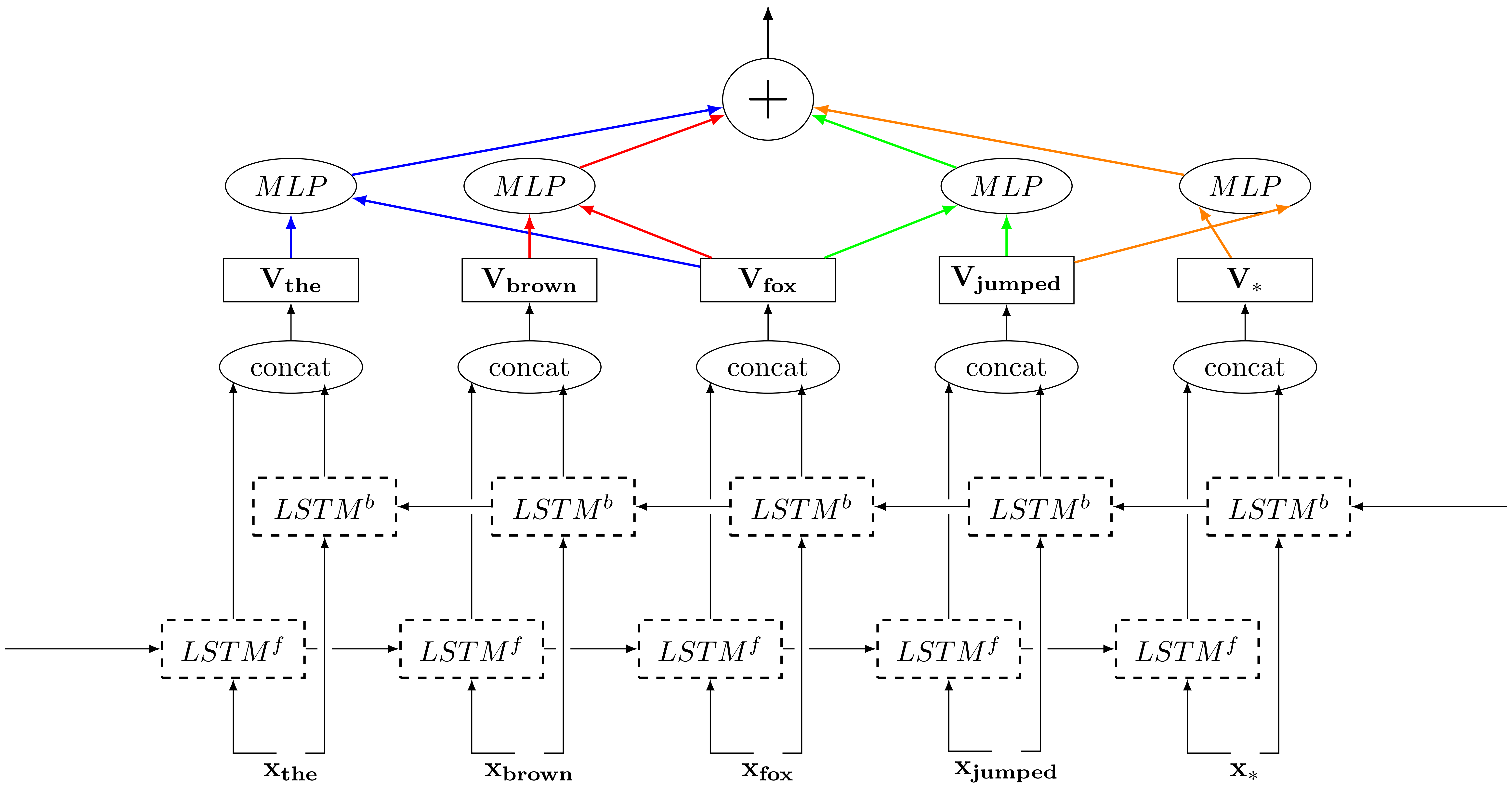
Graph-based neural network parsers



Graph-based neural network parsers

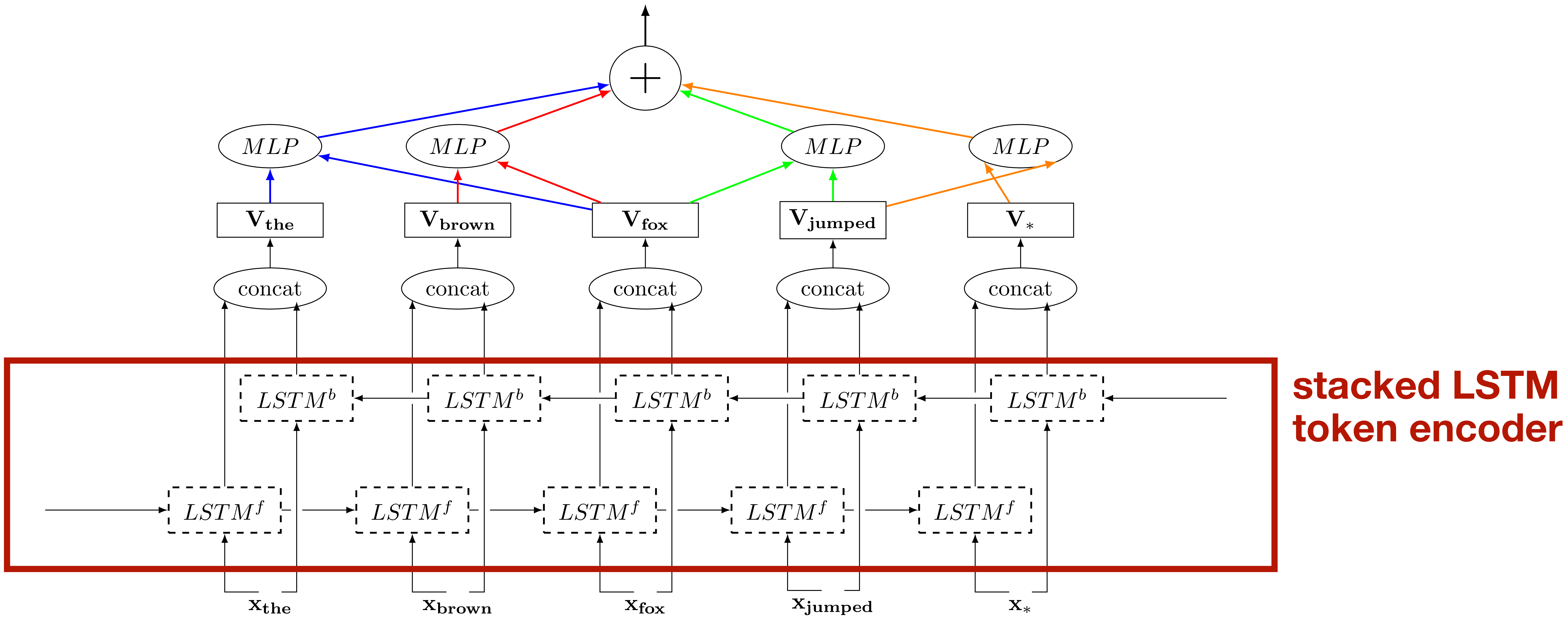
Graph-based neural network parsers

- Concat + feed-forward [Kiperwasser and Goldberg, 2016]



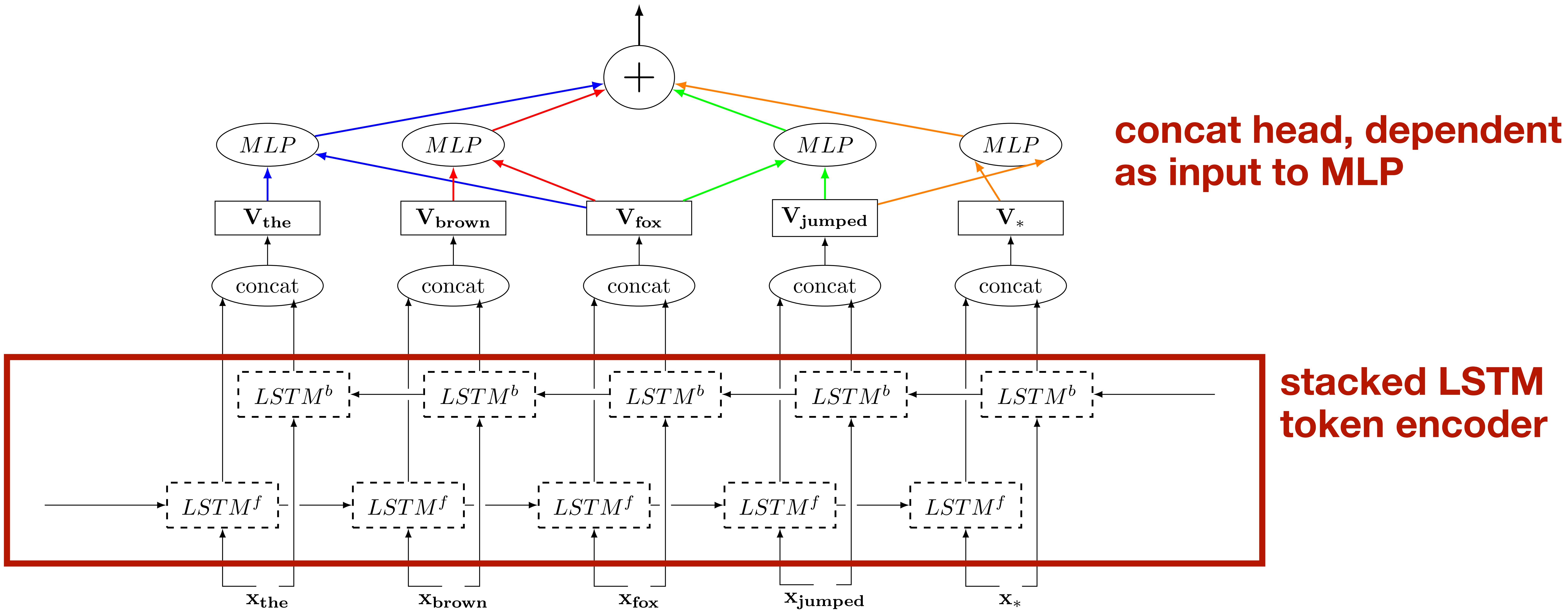
Graph-based neural network parsers

- Concat + feed-forward [Kiperwasser and Goldberg, 2016]



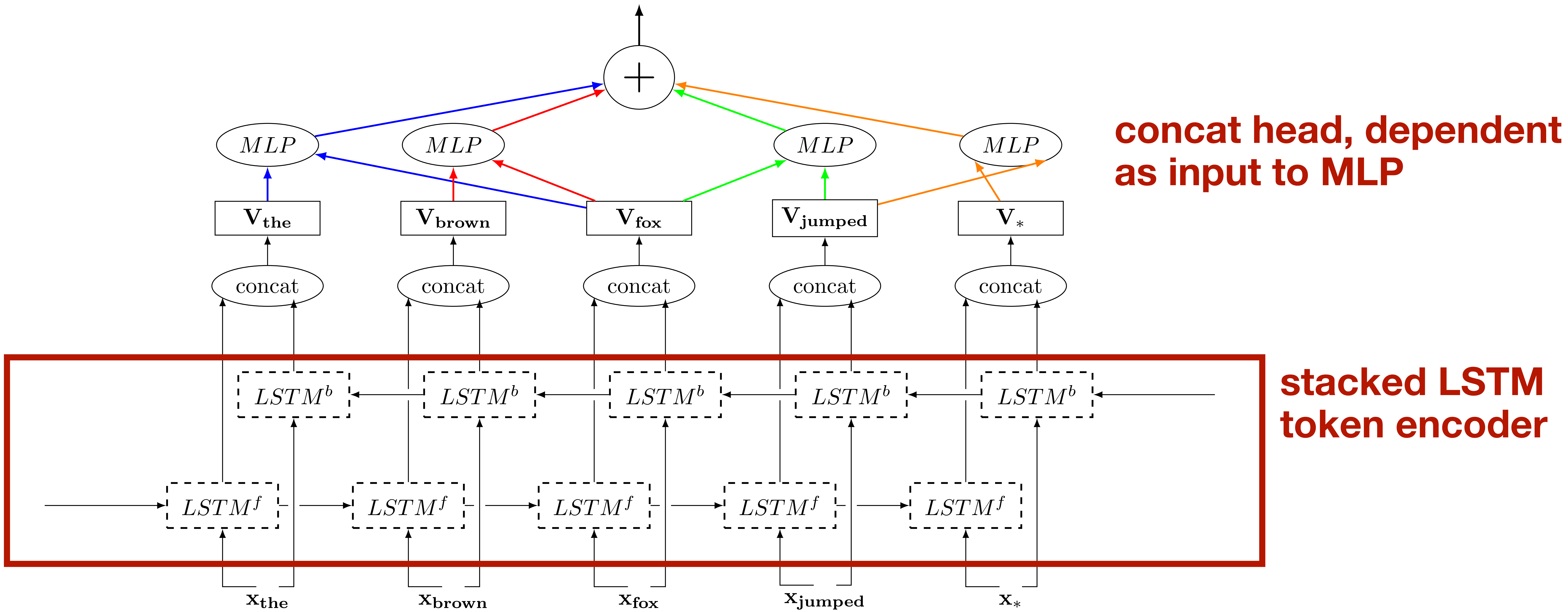
Graph-based neural network parsers

- Concat + feed-forward [Kiperwasser and Goldberg, 2016]



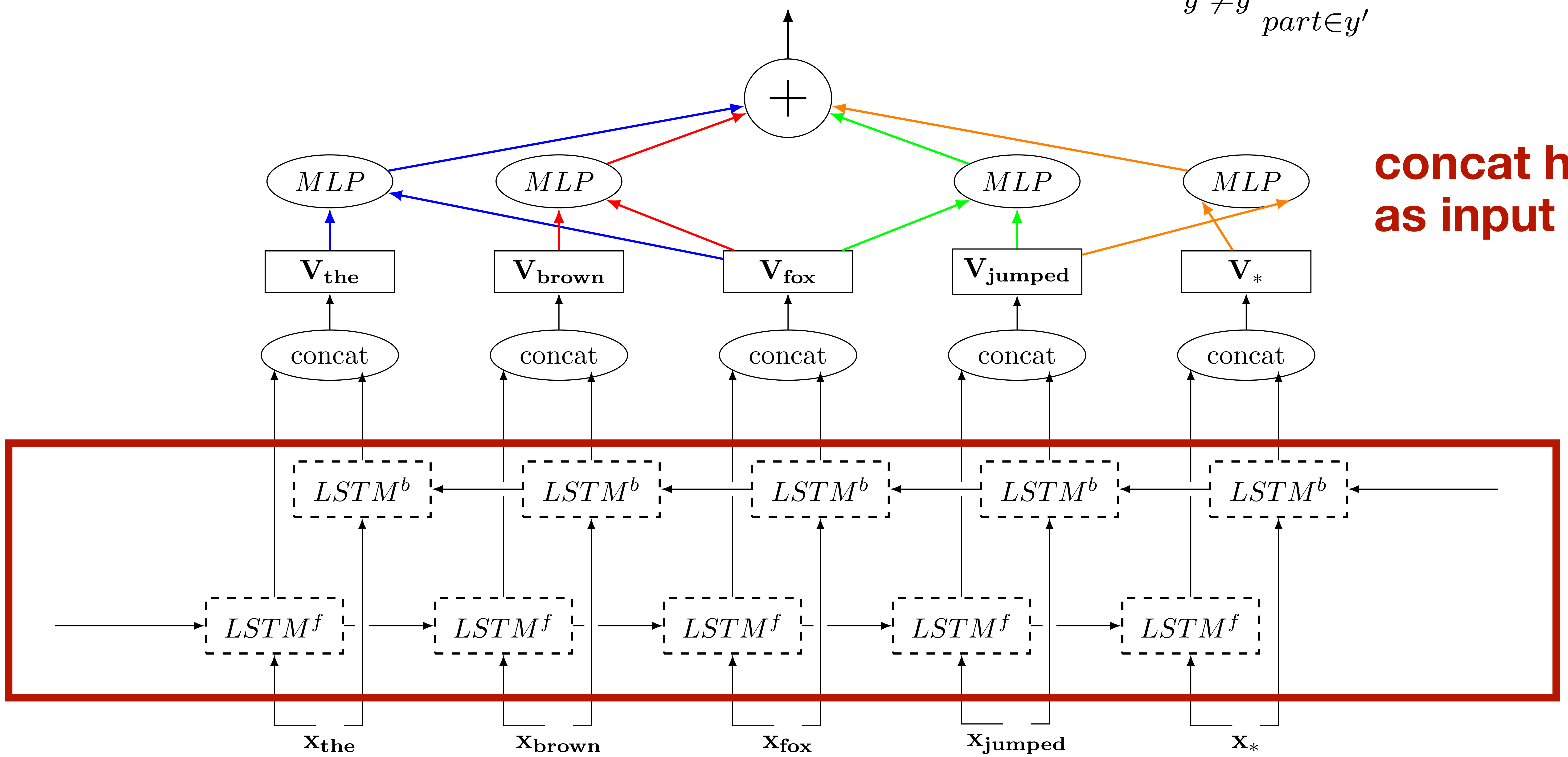
Graph-based neural network parsers

- Concat + feed-forward [Kiperwasser and Goldberg, 2016]
- Hinge loss: $max\Big(0, 1 - \max_{y' \neq y} \sum_{(h,m) \in y'} MLP(v_h \circ v_m) + \sum_{(h,m) \in y} MLP(v_h \circ v_m)\Big)$



Graph-based neural network parsers

- Concat + feed-forward [Kiperwasser and Goldberg, 2016]
- Hinge loss: $max\Big(0, 1 - \max_{y' \neq y} \sum_{(h,m) \in y'} MLP(v_h \circ v_m) + \sum_{(h,m) \in y} MLP(v_h \circ v_m)\Big)$
  - w/ loss augmented inference: $max(0, 1 + score(x,y) - \max_{y' \neq y} \sum_{part \in y'} (score_{local}(x, part) + \mathbb{1}_{part \notin y}))$



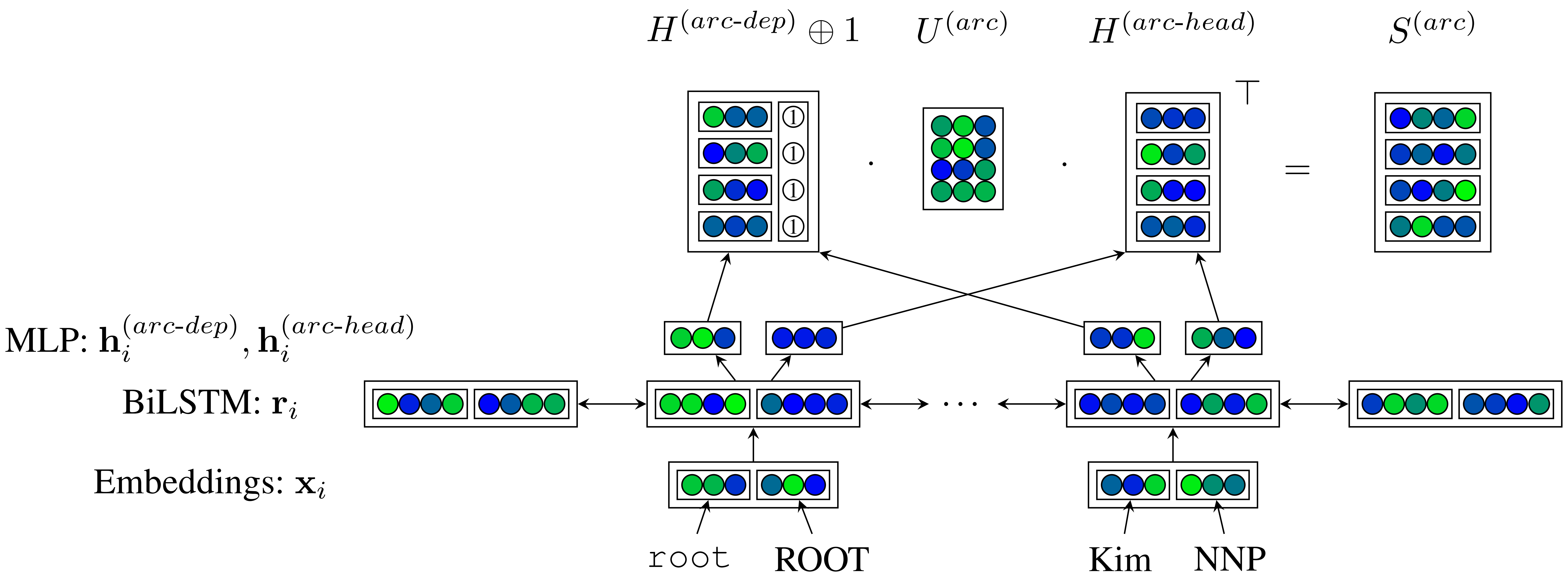
concat head, dependent as input to MLP

stacked LSTM token encoder

Graph-based neural network parsers

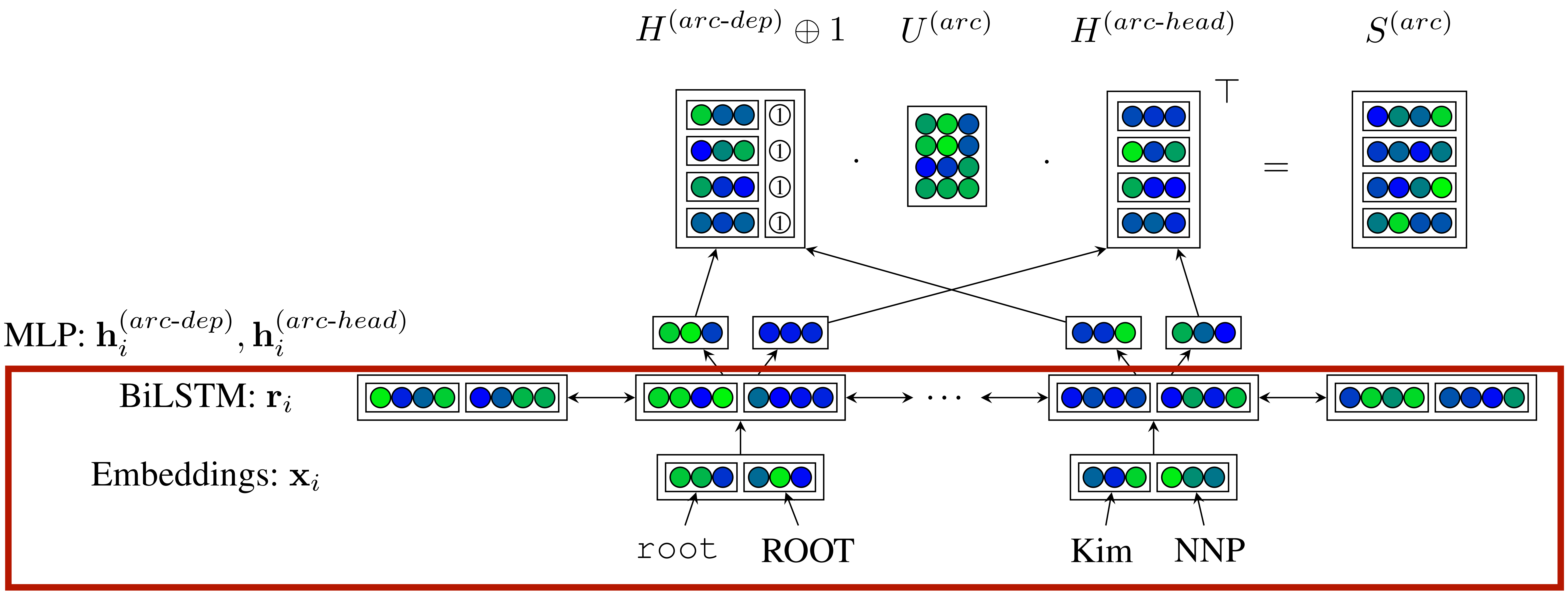
Graph-based neural network parsers

- Biaffine classifier [Dozat and Manning, 2017]



Graph-based neural network parsers

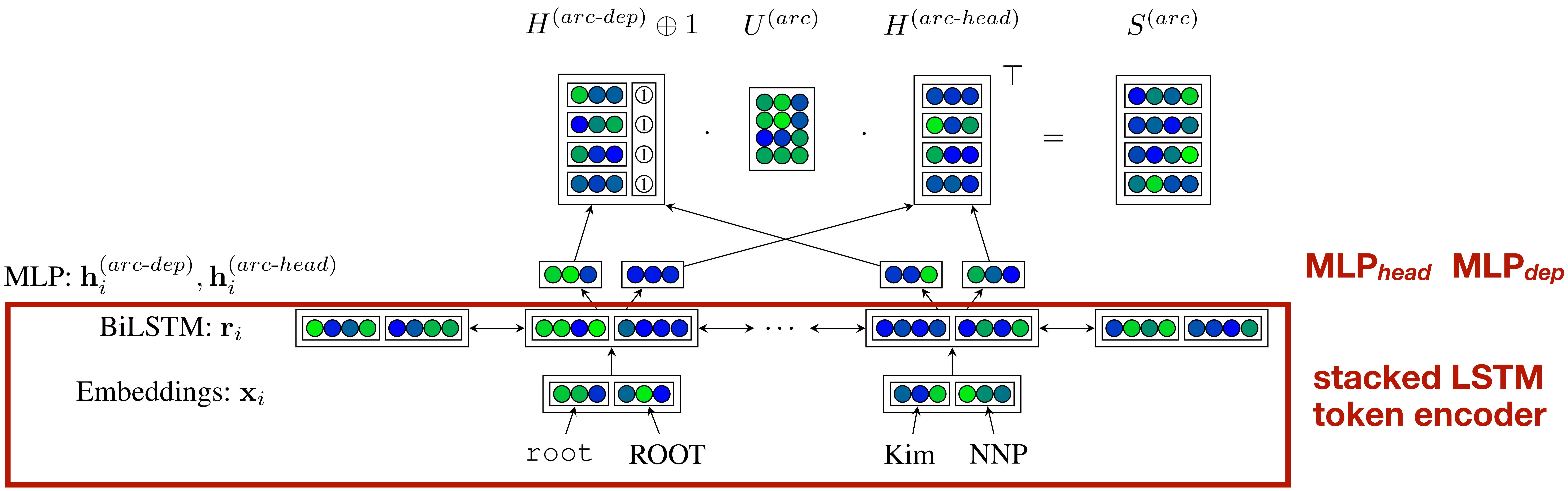
- Biaffine classifier [Dozat and Manning, 2017]



stacked LSTM token encoder

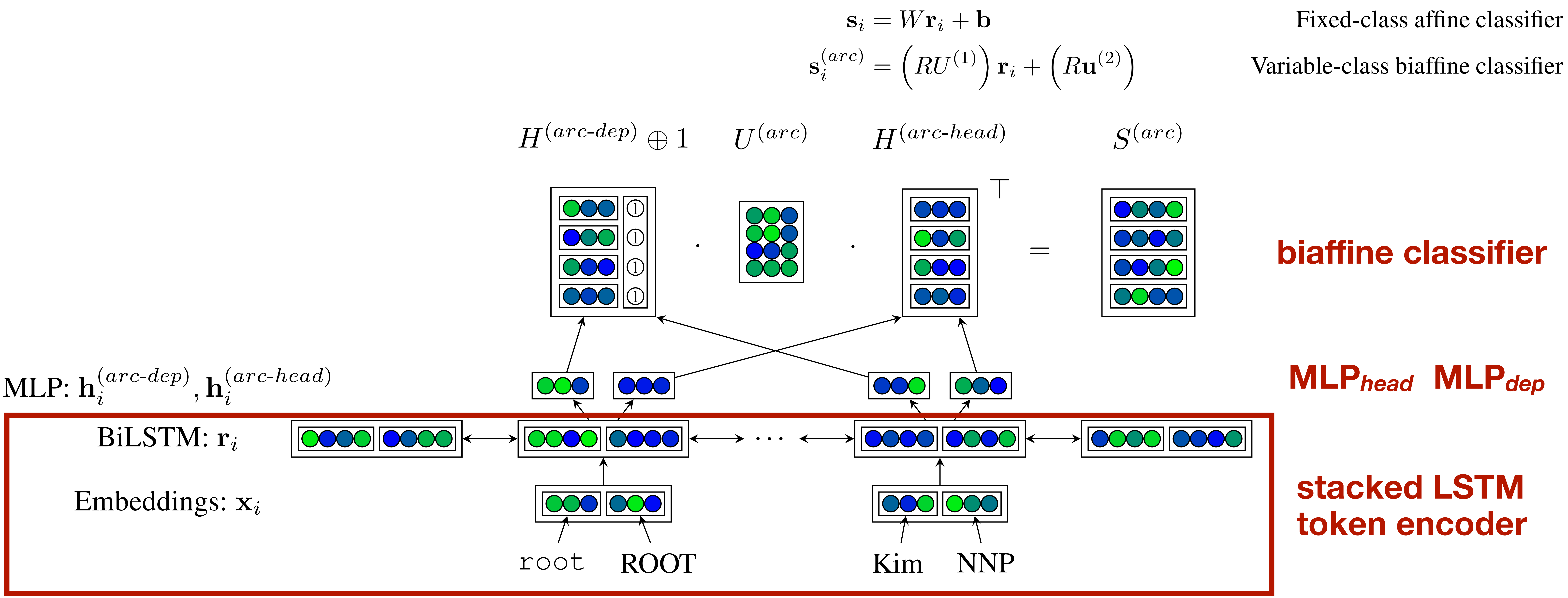
Graph-based neural network parsers

- Biaffine classifier [Dozat and Manning, 2017]



Graph-based neural network parsers

- Biaffine classifier [Dozat and Manning, 2017]

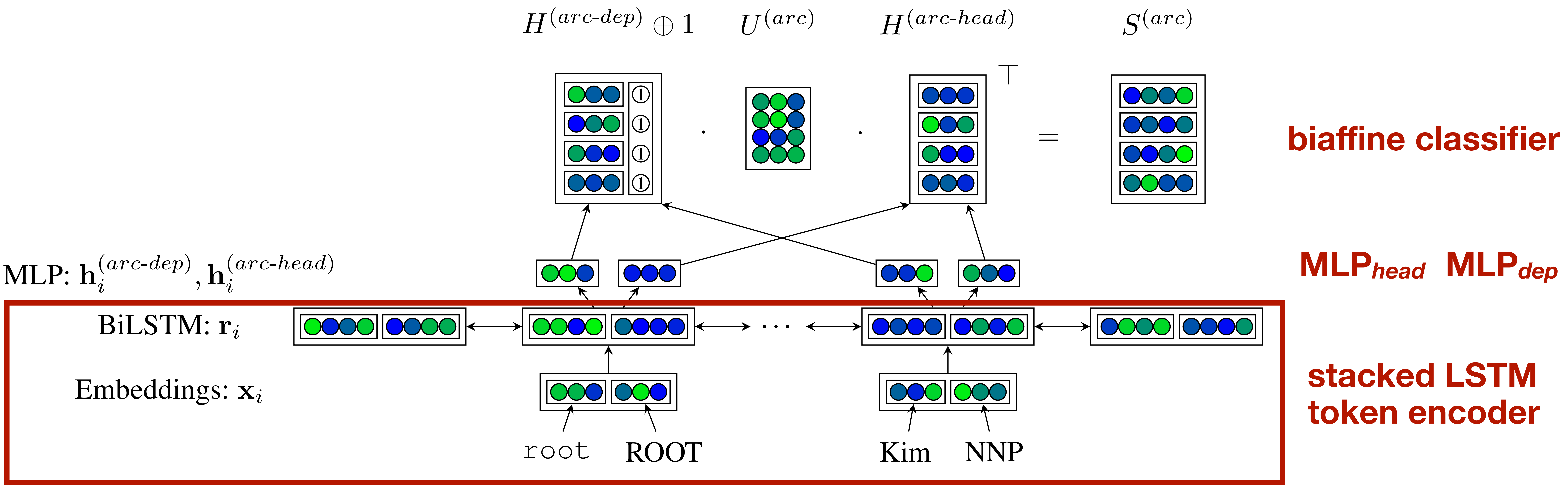


Graph-based neural network parsers

- Biaffine classifier [Dozat and Manning, 2017]
- Locally normalized log loss.

$$\mathbf{s}_i = W\mathbf{r}_i + \mathbf{b} \qquad \text{Fixed-class affine classifier}$$

$$\mathbf{s}_i^{(arc)} = \left(RU^{(1)}\right)\mathbf{r}_i + \left(R\mathbf{u}^{(2)}\right) \qquad \text{Variable-class biaffine classifier}$$



Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

First order $\overset{\frown}{h \quad m}$

- How to parameterize ψ ?

Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

First order $\overset{\frown}{h \quad m}$

- How to parameterize ψ ?
- How to learn θ ?

Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

First order $\overset{\frown}{h \quad m}$

- How to parameterize ψ ?
- How to learn θ ?
    - Locally normalized: predict each head and its label, softmax, log loss.

Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

First order $\overset{\curvearrowright}{h \quad m}$

- How to parameterize ψ ?
- How to learn θ ?
  - Locally normalized: predict each head and its label, softmax, log loss.
  - Hinge loss: Find the best scoring tree, and penalize edges not in the gold tree (with a margin).

Graph-based dependency parsing

- **Edge-factored** (or **arc-factored**) approaches: score of a tree decomposes as sum of edge scores.

$$\Psi(\mathbf{y}, \mathbf{w}; \theta) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}, \theta)$$

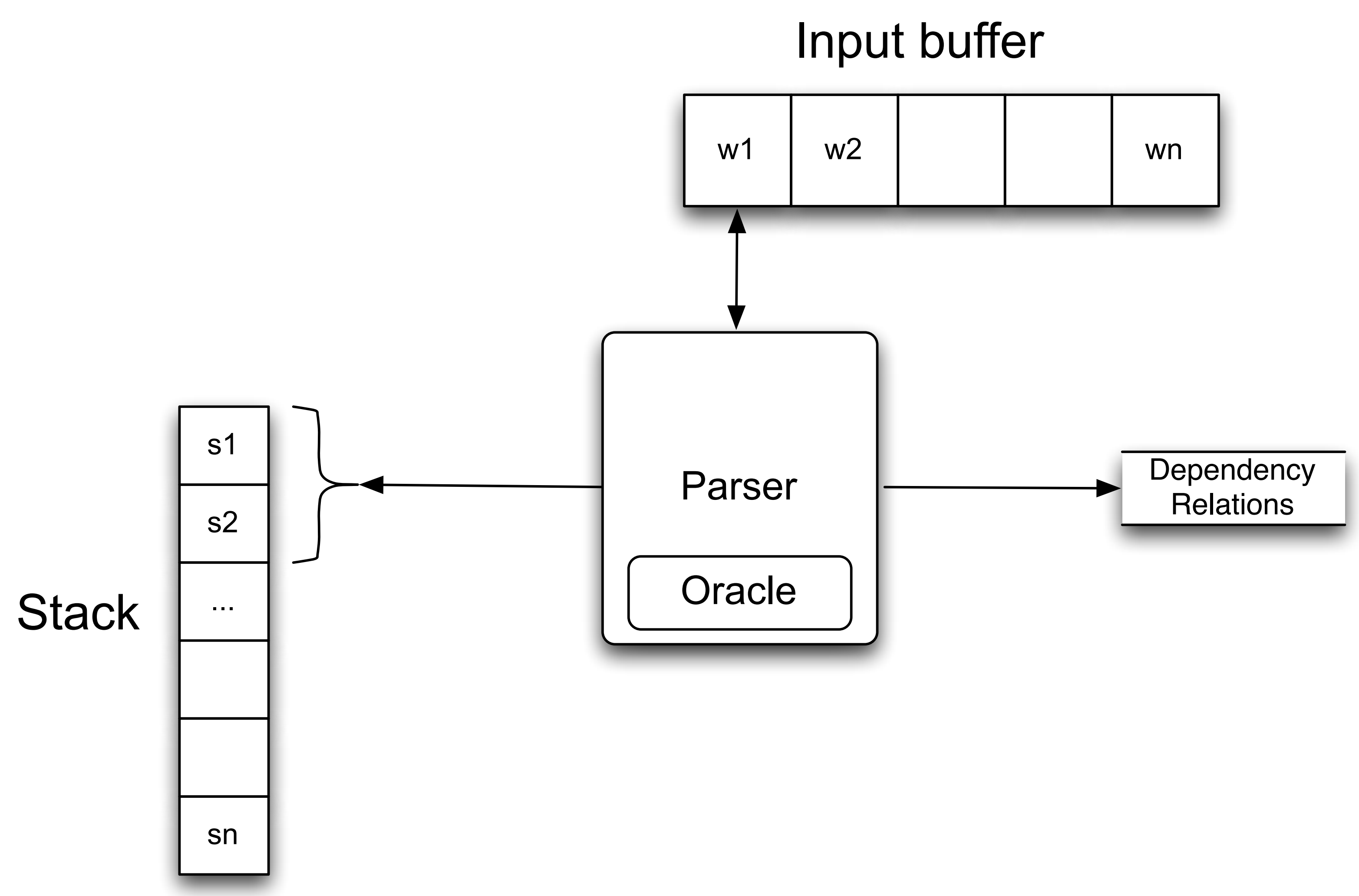
First order $\overset{\frown}{h \quad m}$

- How to parameterize ψ ?
- How to learn θ ?
  - Locally normalized: predict each head and its label, softmax, log loss.
  - Hinge loss: Find the best scoring tree, and penalize edges not in the gold tree (with a margin).
  - Globally normalized CRF: can compute marginals/partition function using a variant of Kirchhoff's Matrix-Tree Theorem [Tutte, 1984; Koo et al. 2007].

Graph-based vs. transition-based parsing?

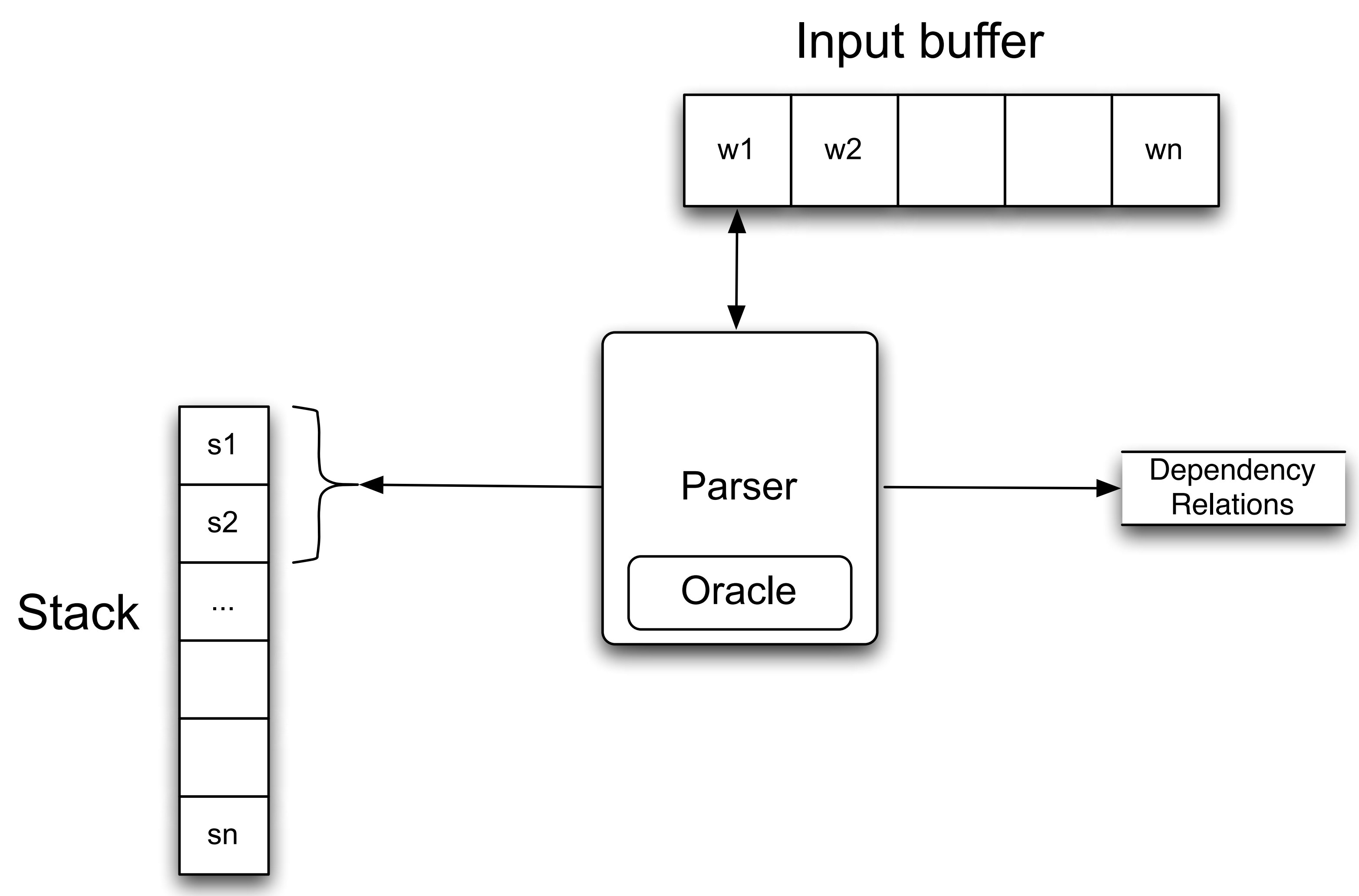
Graph-based vs. transition-based parsing?

- **Transition-based**
  - Fast
  - Greedy / local inference
  - Maybe closer to humans?

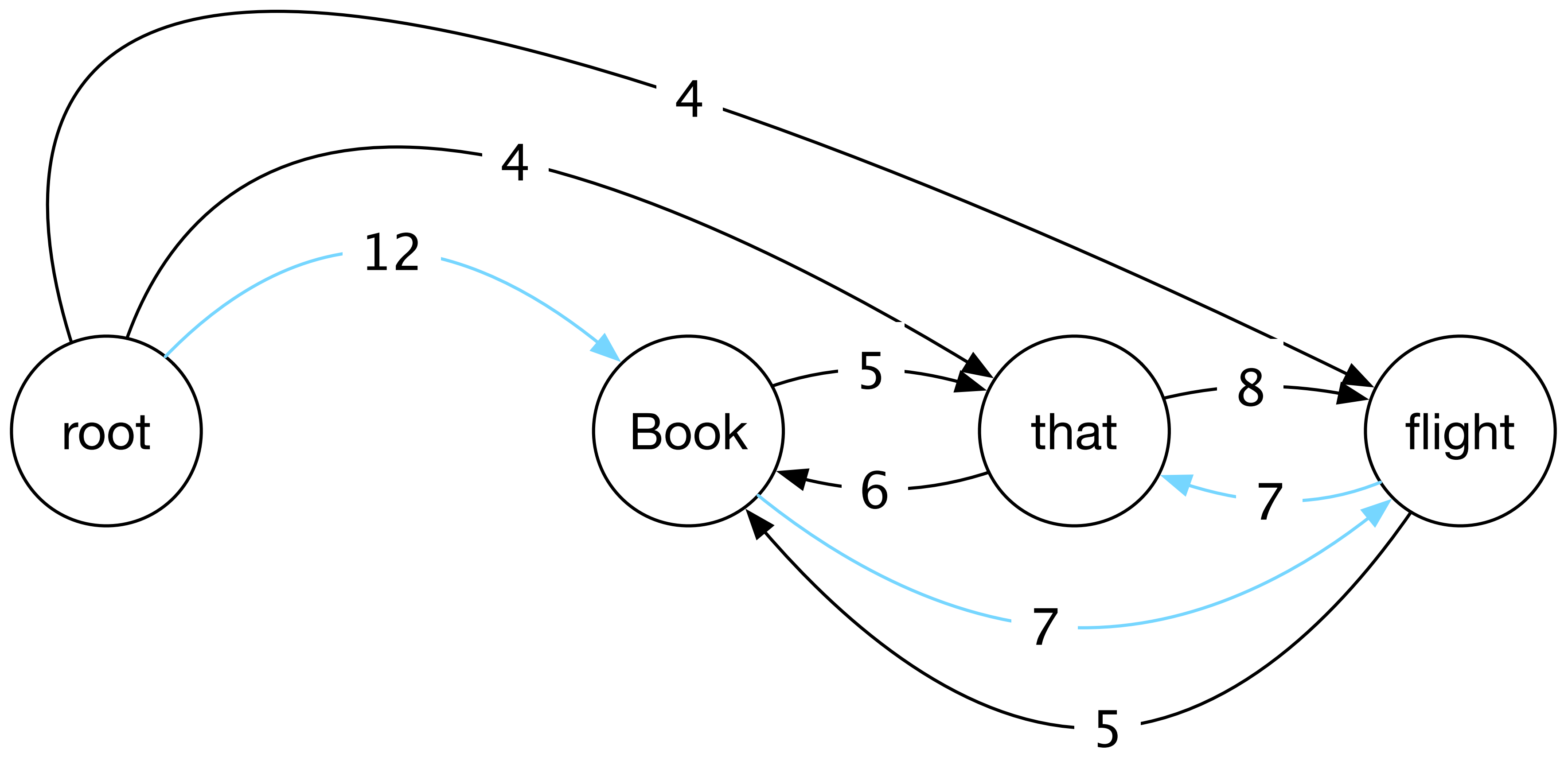


Graph-based vs. transition-based parsing?

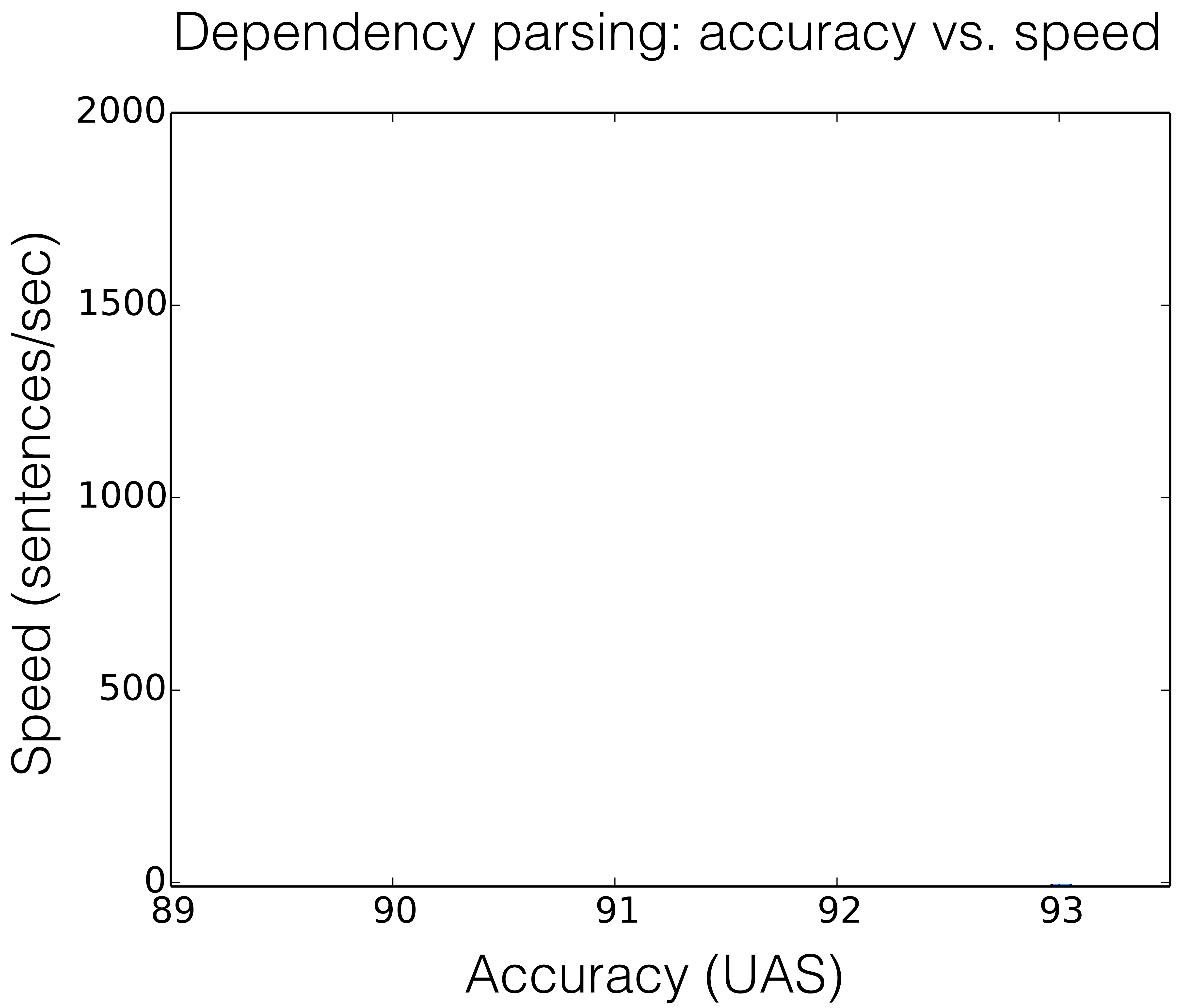
- **Transition-based**
  - Fast
  - Greedy / local inference
  - Maybe closer to humans?



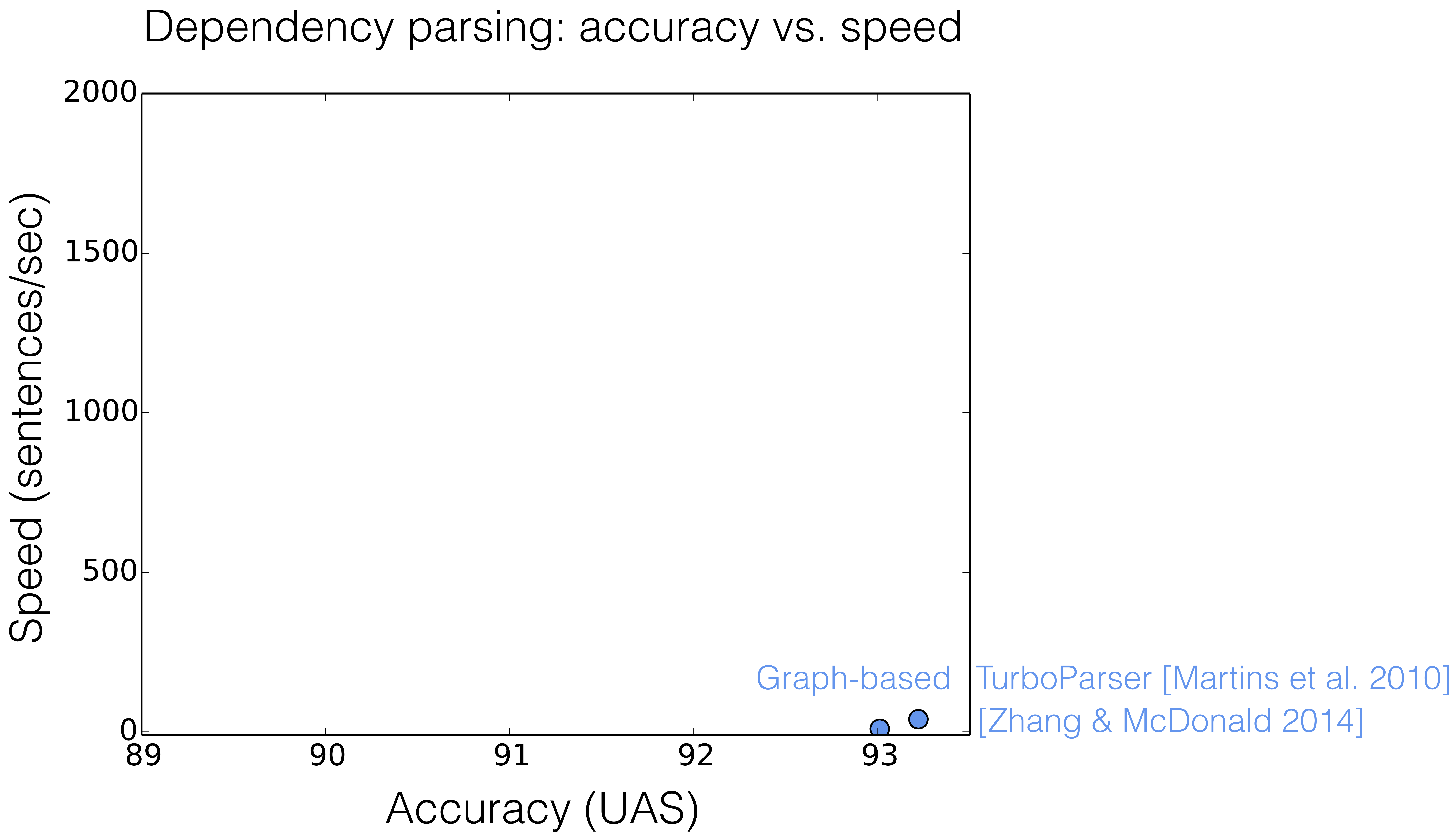
- **Graph-based**
  - Slow
  - Exact inference
  - More accurate



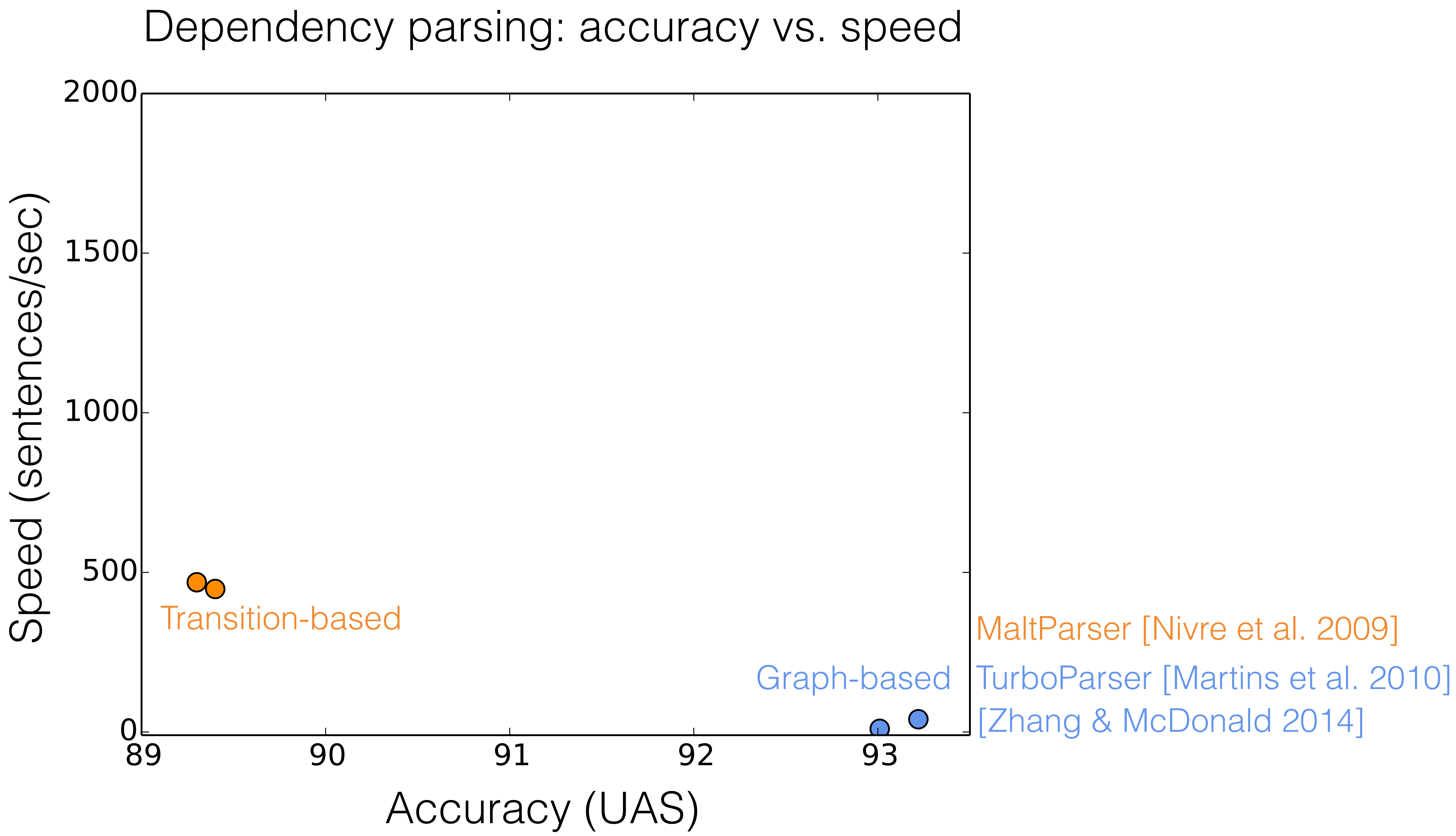
Graph-based vs. transition-based parsing?



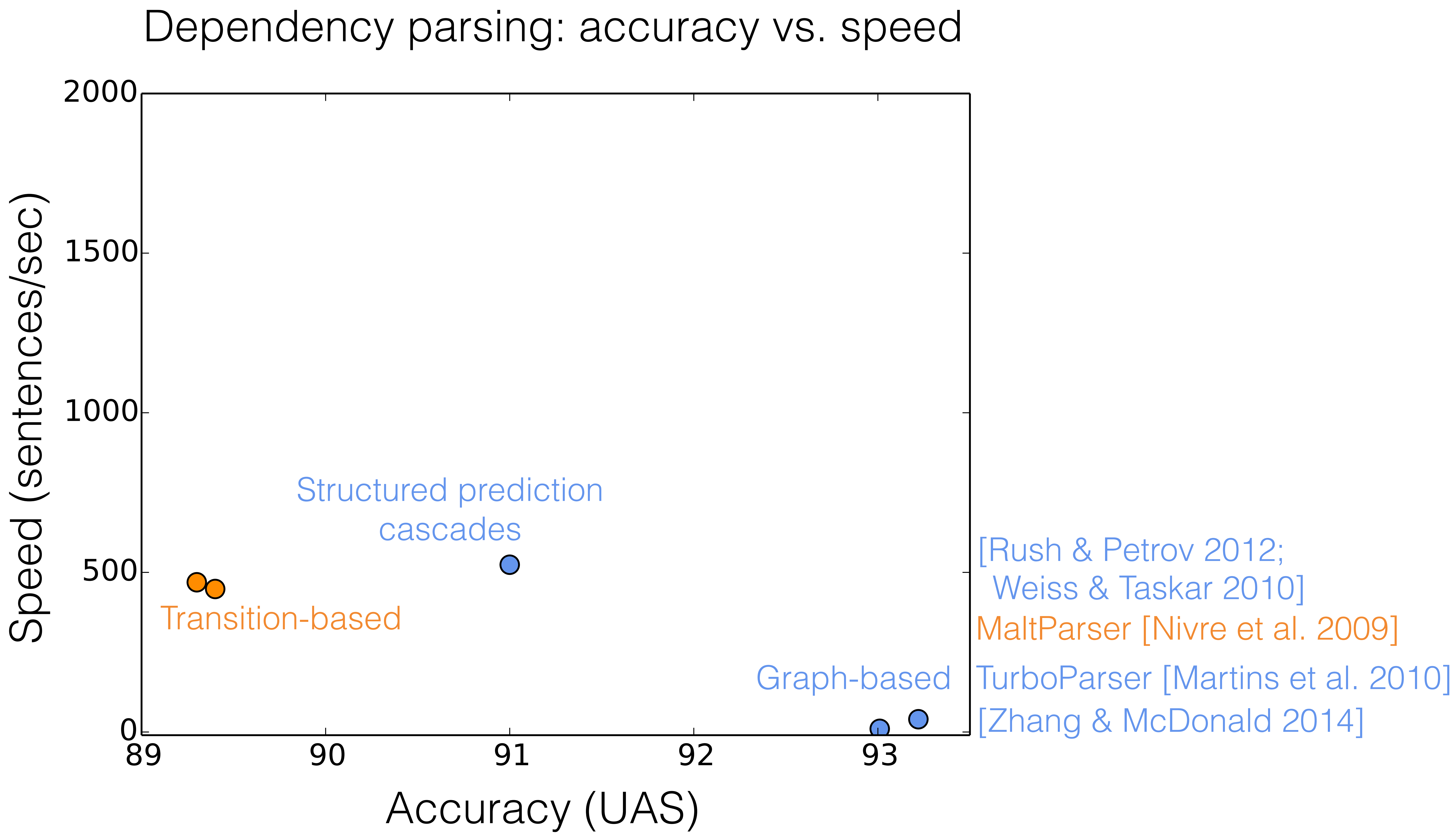
Graph-based vs. transition-based parsing?



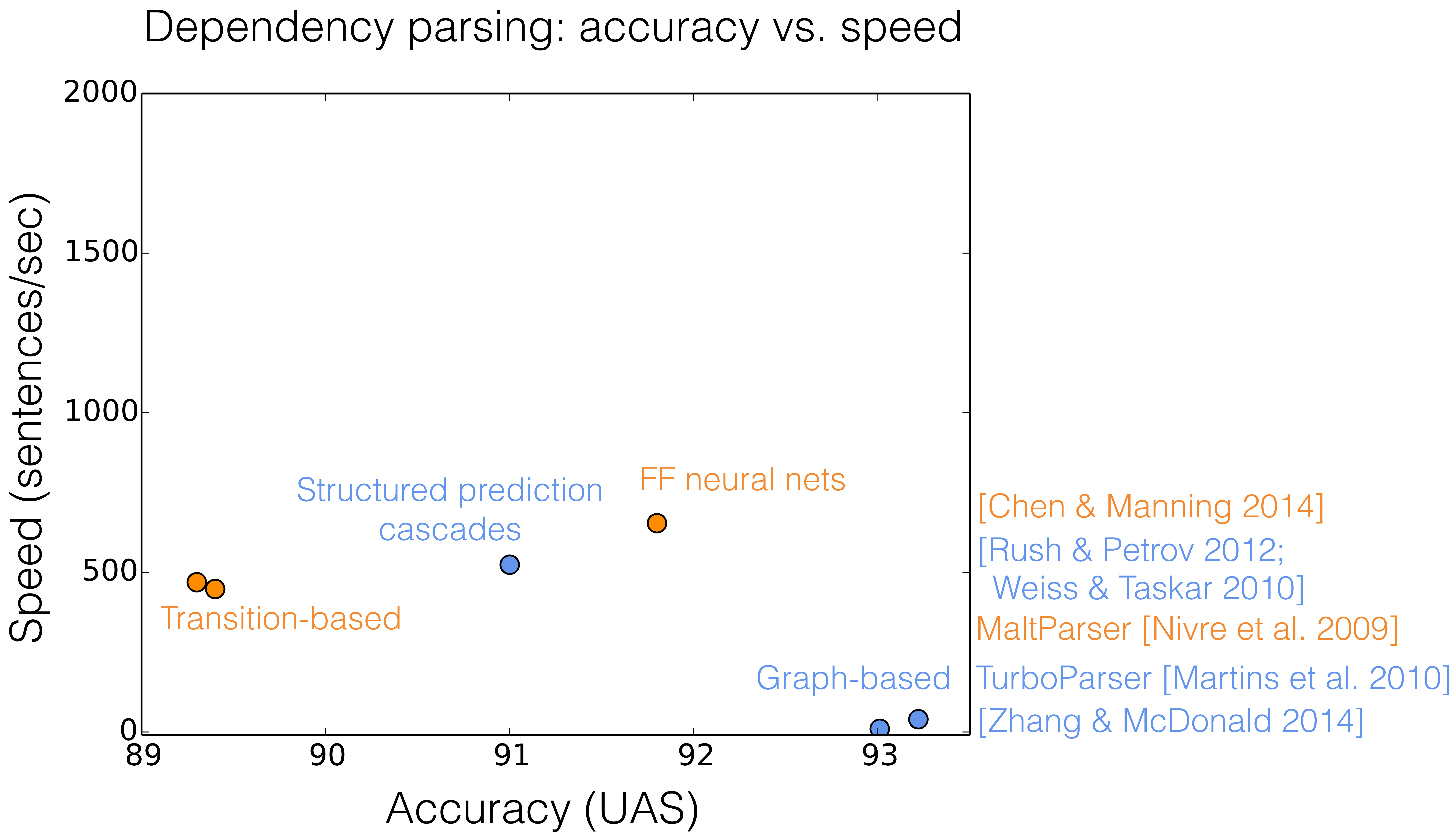
Graph-based vs. transition-based parsing?



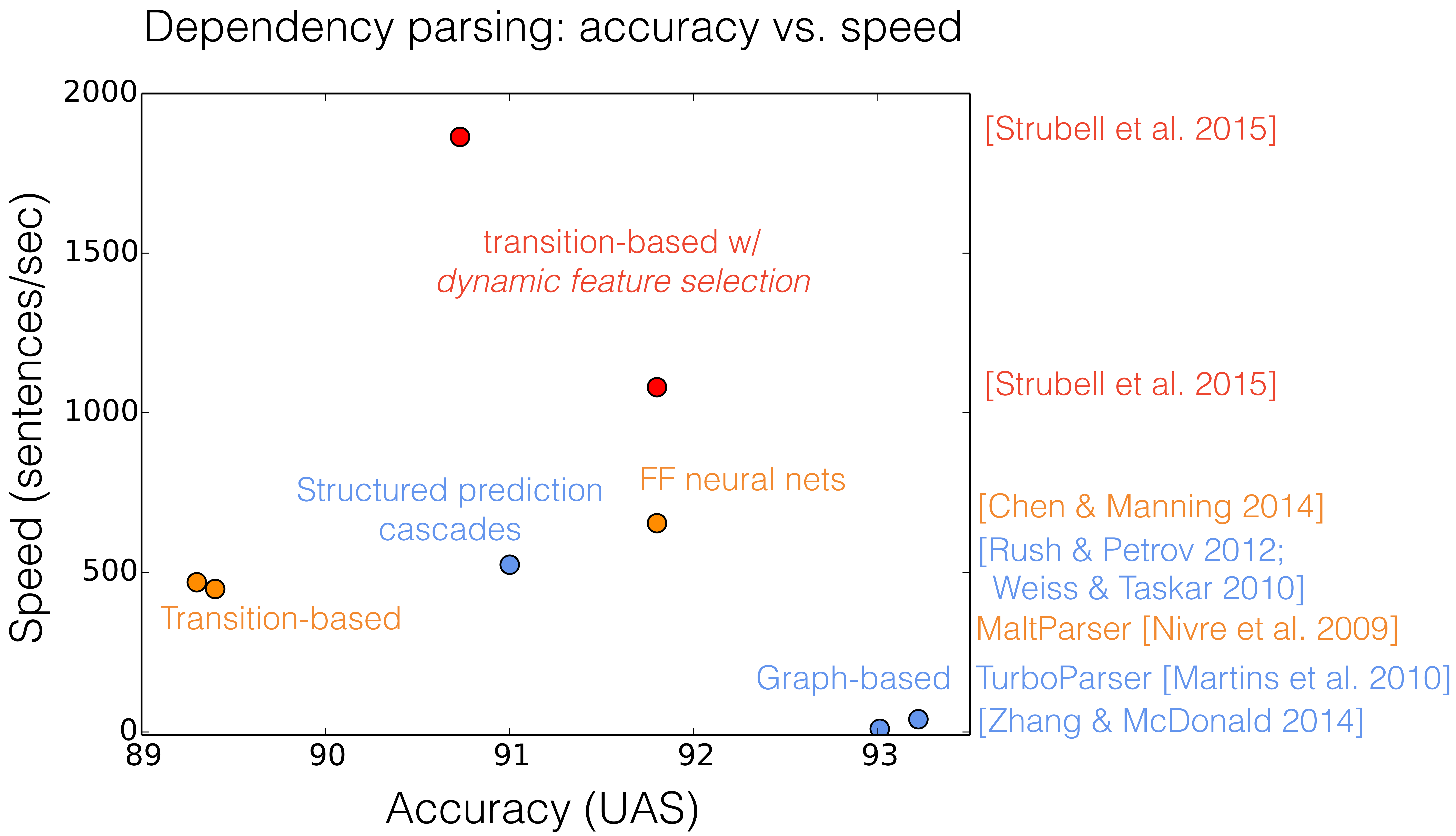
Graph-based vs. transition-based parsing?



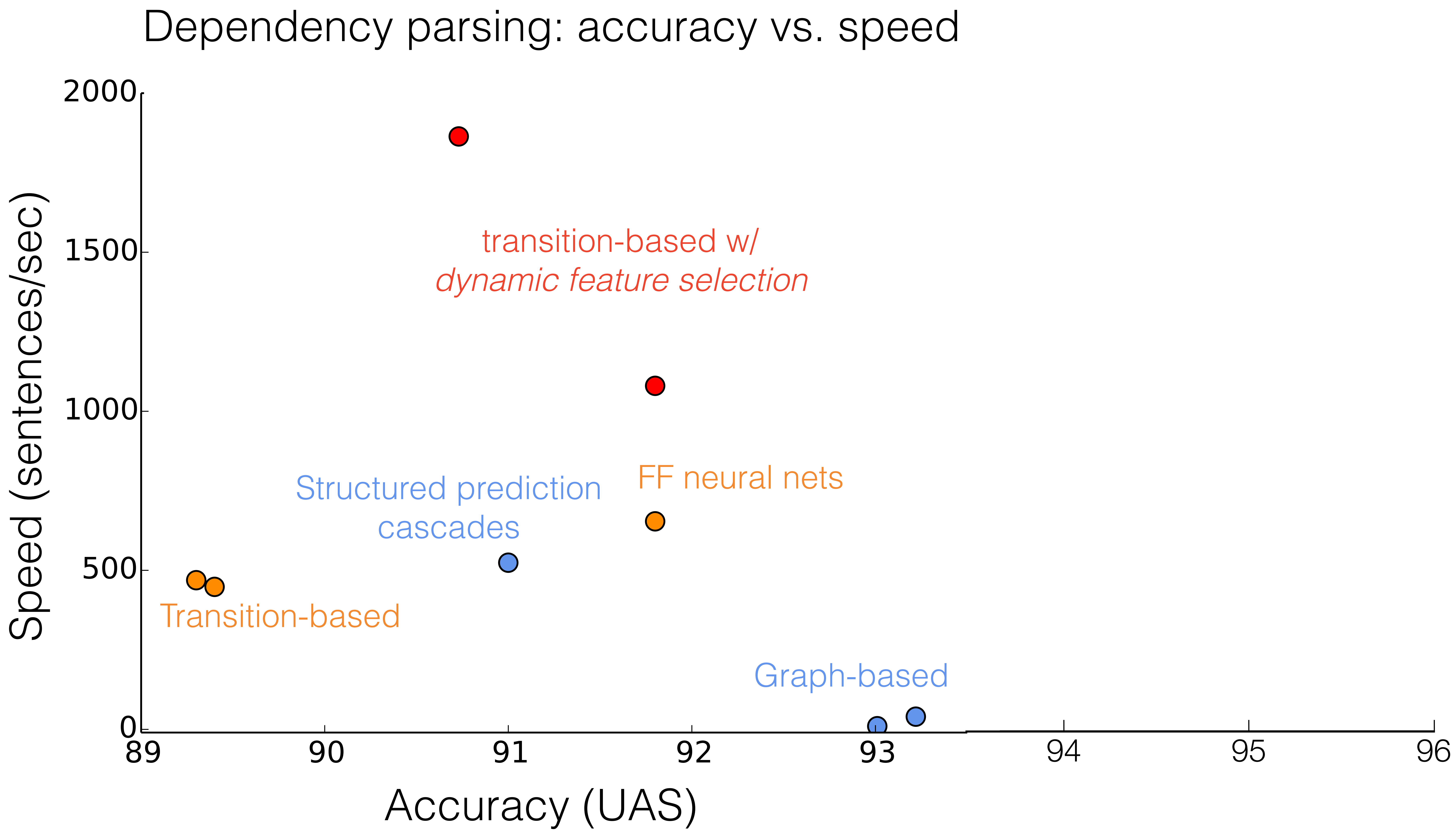
Graph-based vs. transition-based parsing?



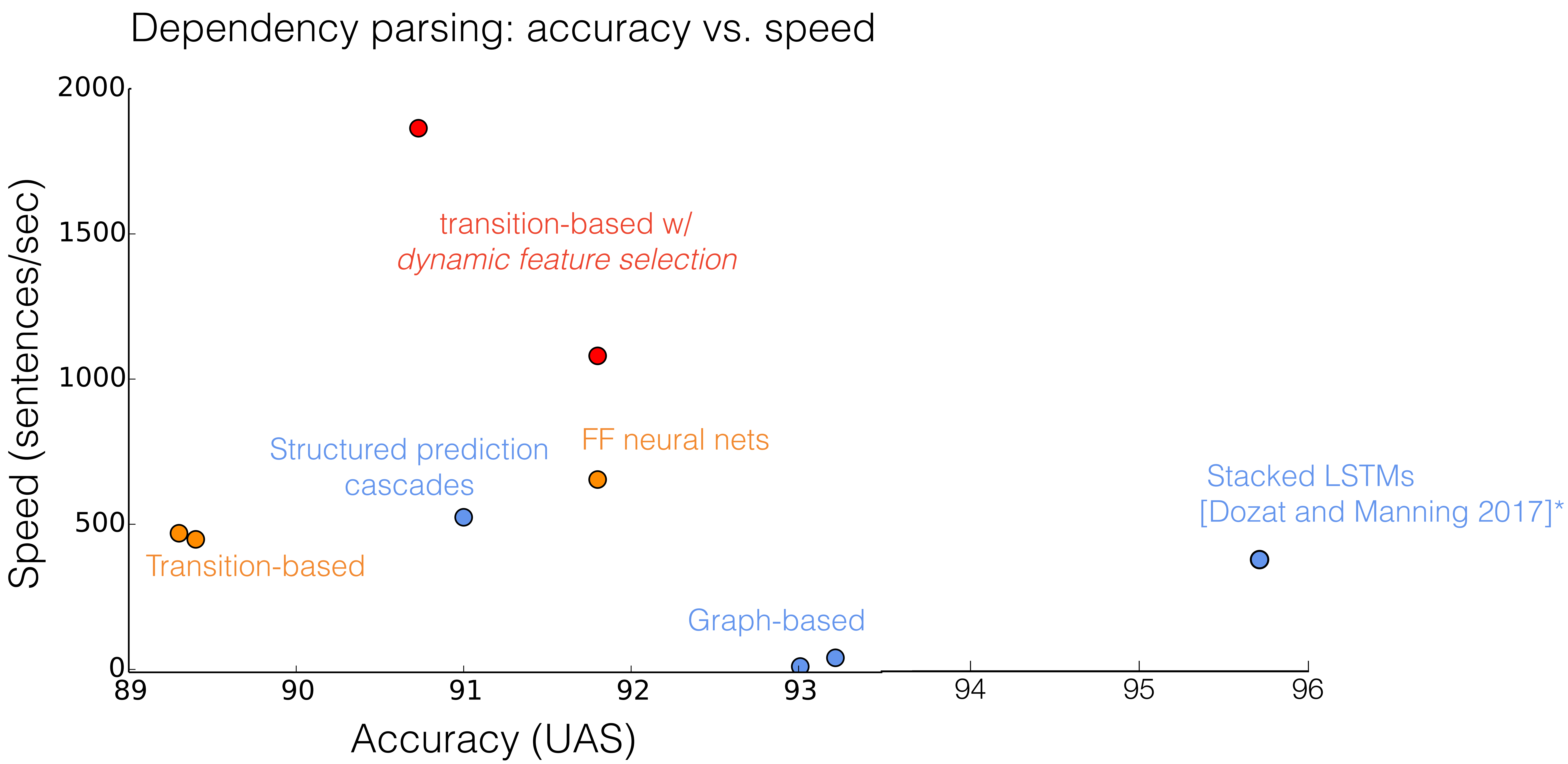
Graph-based vs. transition-based parsing?



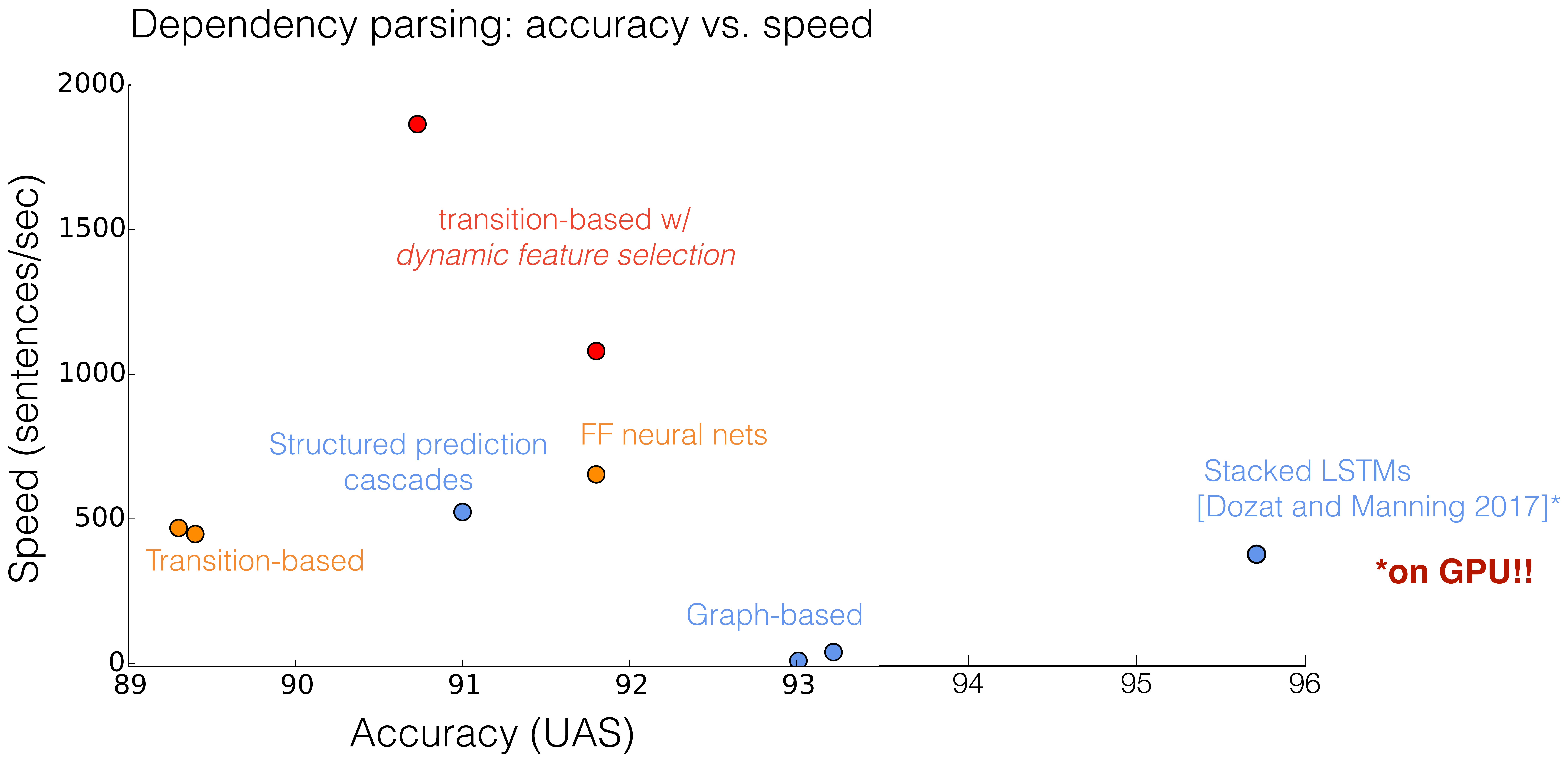
Graph-based vs. transition-based parsing?



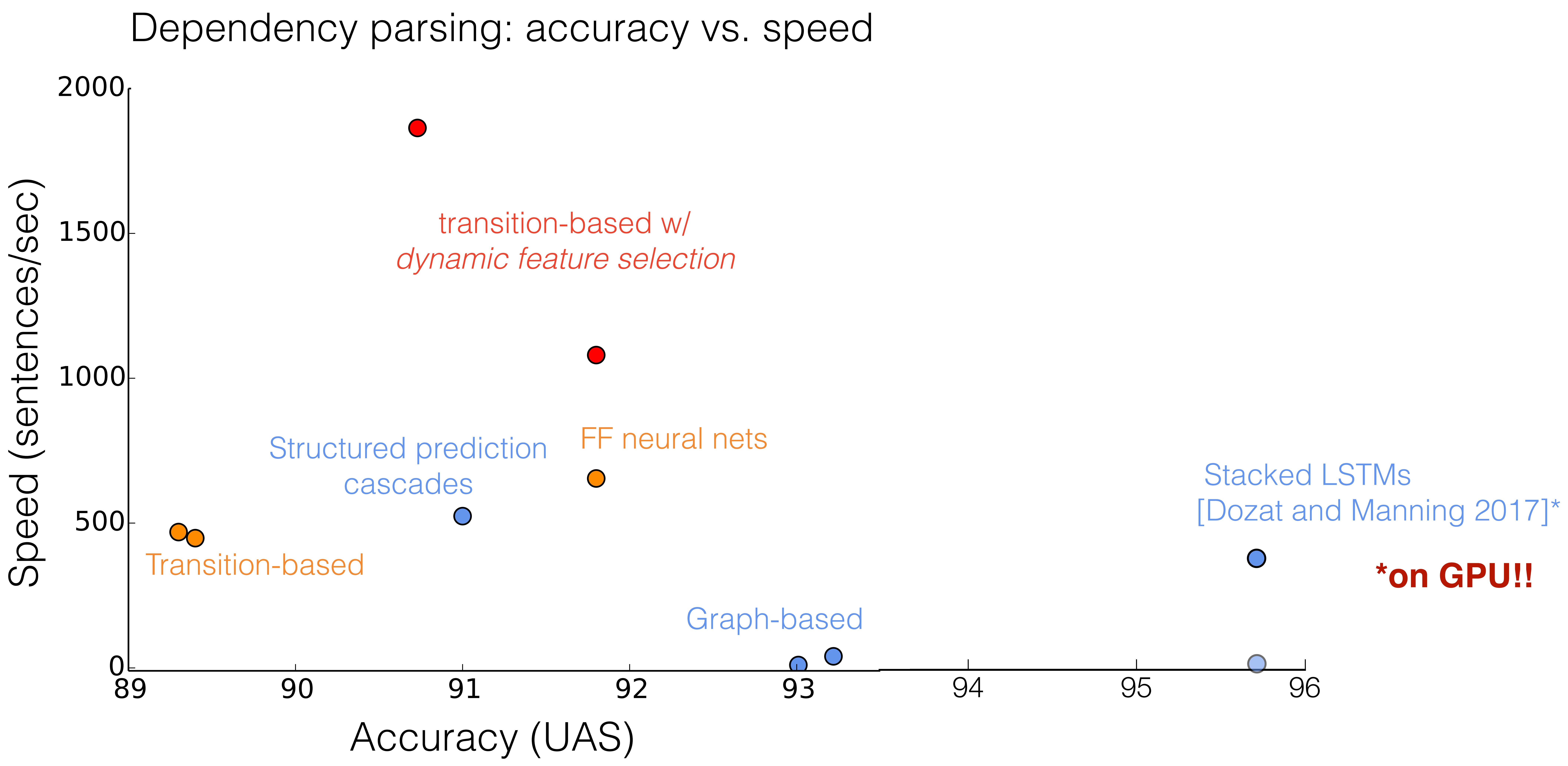
Graph-based vs. transition-based parsing?



Graph-based vs. transition-based parsing?



Graph-based vs. transition-based parsing?



Announcements

- No recitation on Friday (Tartan Community Day).