Comparing Convolution Kernels and RNNs on a wide-coverage computational analysis of natural language

Fabrizio Costa, Paolo Frasconi, Sauro Menchetti
Dept. Systems and Computer Science
Università di Firenze

Massimiliano Pontil
Dept. Information Engineering
Università di Siena

Related papers available from
http://www.dsi.unifi.it/~paolo
http://www.dsi.unifi.it/~costa

Overview

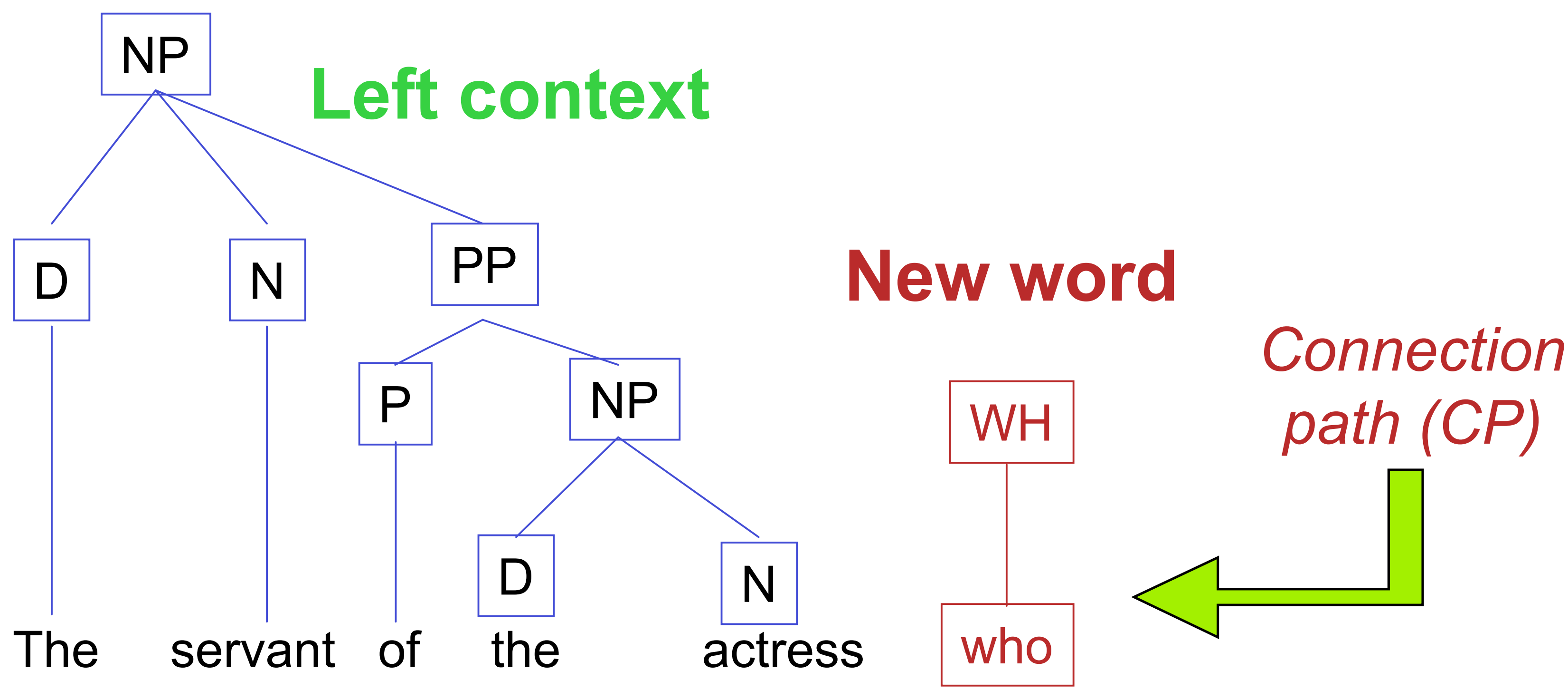
- Incremental parsing of natural language
  - A ranking problem on labeled forests
- Supervised learning of discrete structures
  - Recursive neural networks (RNNs)
  - Kernel-based approaches
- New results with RNNs
- Experimental comparison

Human vs computer parsing

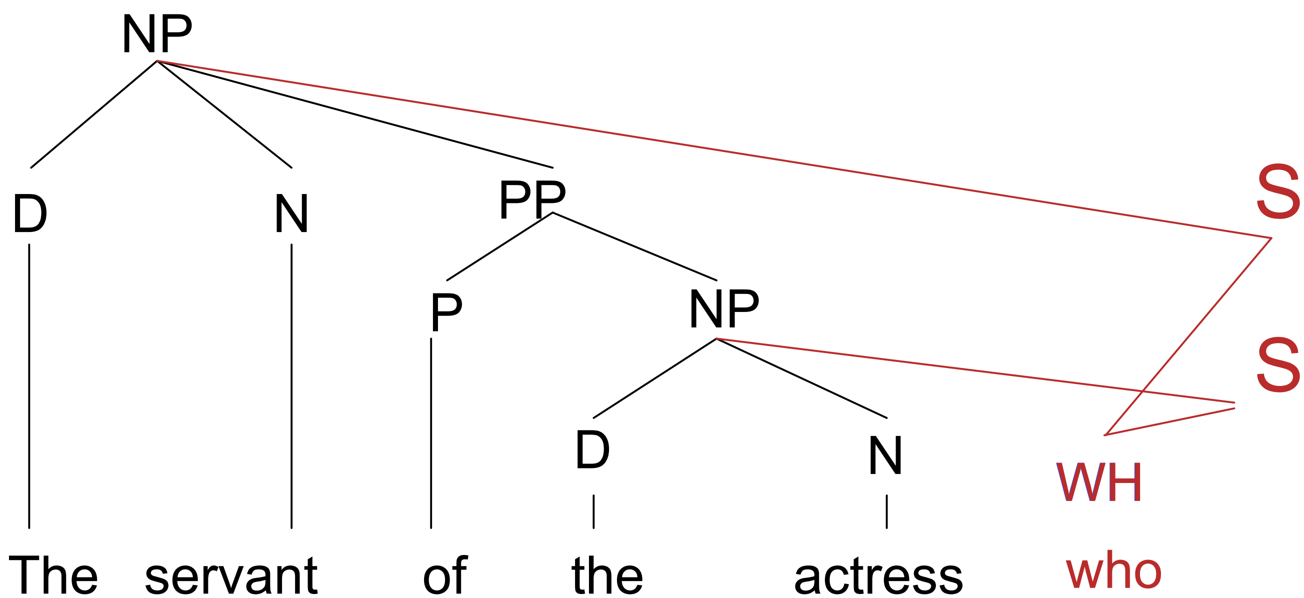
- Computer parsing: typically bottom up
  - `islands' are built at the beginning that are subsequently joined together
- Human parsing: known to be left-to-right
  - E.g., perception of speech is sequential, reading is sequential, etc.

Strong incrementality hypothesis

- The human parser maintains a connected structure that explains the first n-1 words
- When n-th word arrives it is attached to the existing structure

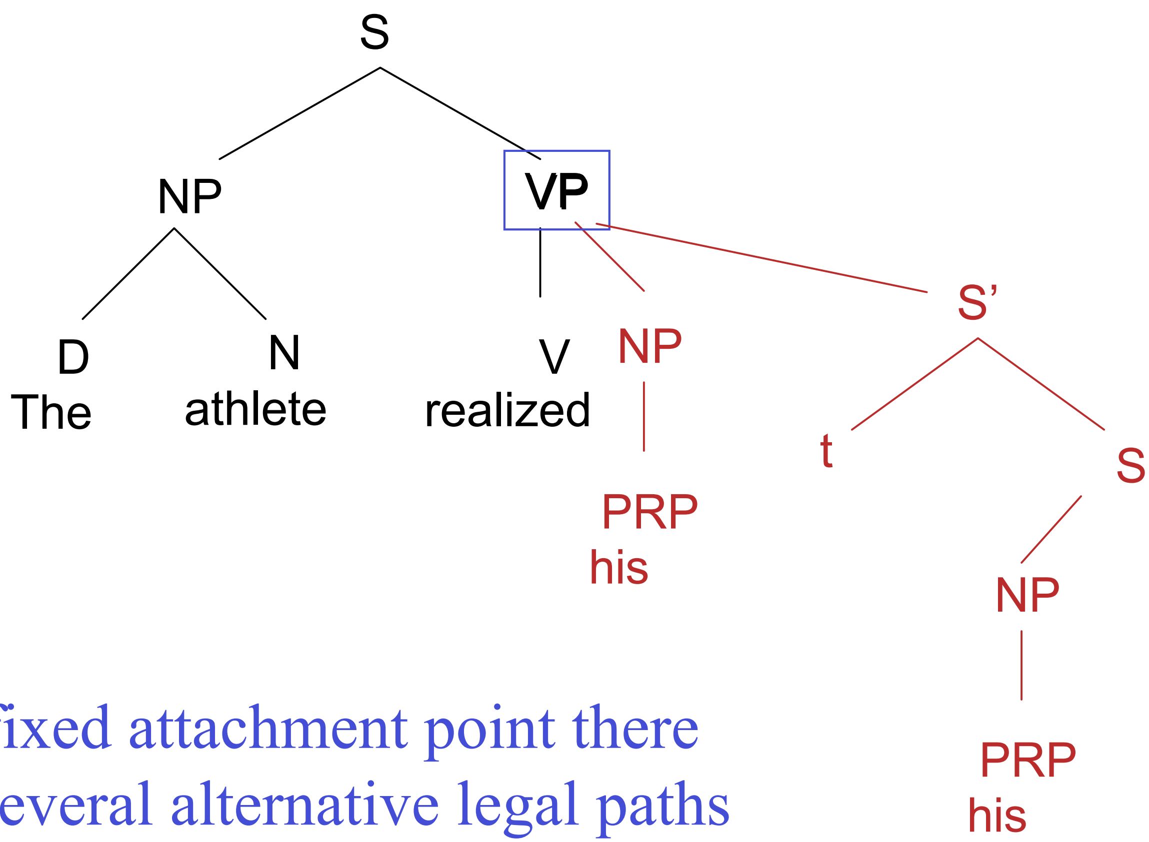


Attachment ambiguity



E.g. low vs. high attachment

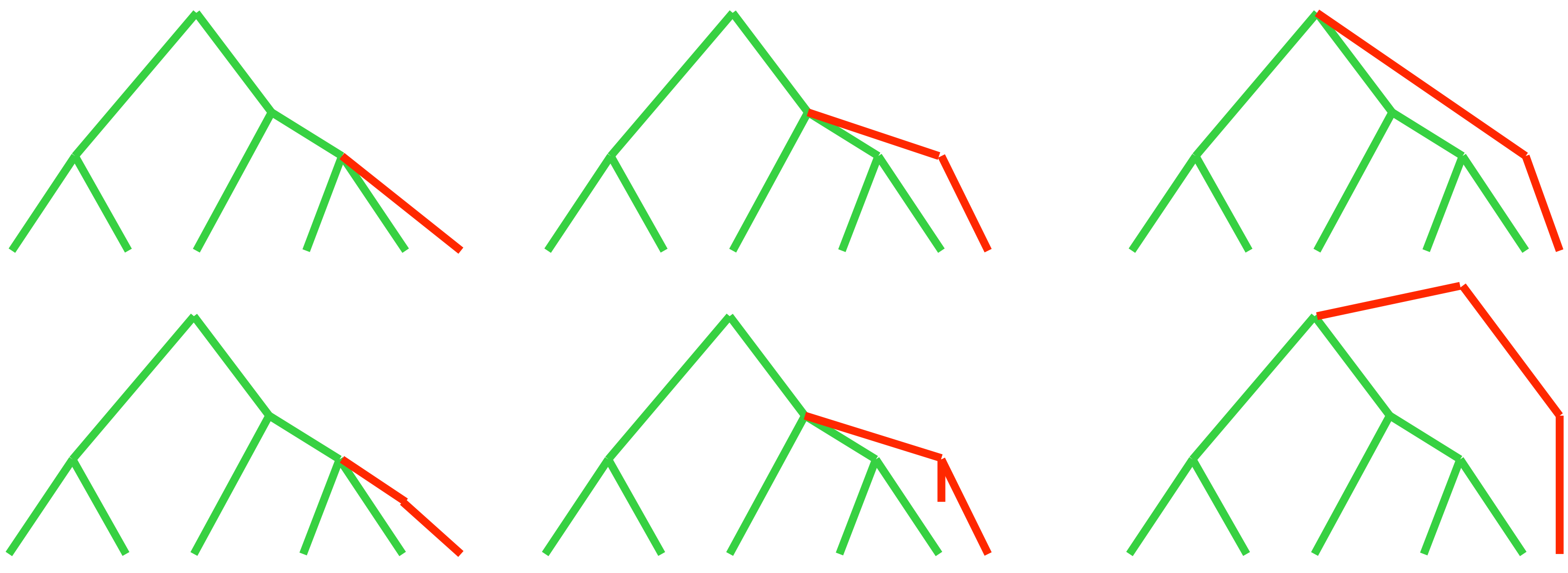
Connection path ambiguity



Even for a fixed attachment point there may be several alternative legal paths (those matching the POS tag of the new word)

A forest of alternatives

- Given a dynamic grammar, a left context and a next word
- Many legal trees can be formed attacching a CP
- One is correct — we want to predict it



Supervised Learning of Discrete Structures

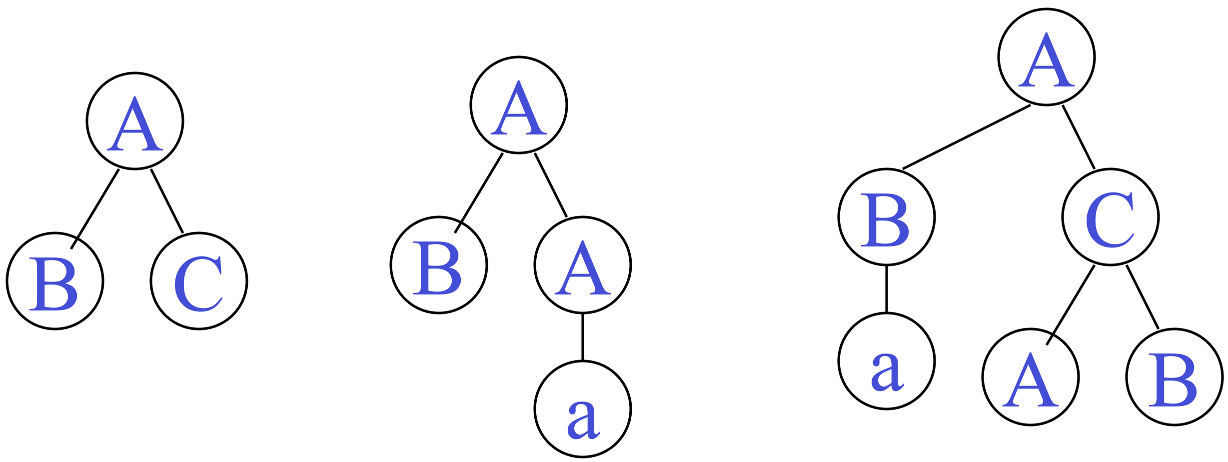
- Lack of methods that handle “directly” recursive or relational structures such as trees and graphs
- General approach:
  1. Convert structures to real vectors
  2. Apply known learning methods on vectors
- These steps can be elegantly merged within a more general theoretical framework:
  1. Recursive neural networks (Göller & Küchler *IJCNN* 96, Frasconi etal *TNN* 98)
  2. Kernel machines (Haussler 99, Collins & Duffy *NIPS* 01, *ACL* 02)

Differences

- Kernel-based methods map a tree into a vector □($\mathbf{x}$) in a very high-dimensional space, perhaps infinite
- Bag-of-something kind of representation
- Kernel choice difficult (prior knowledge?)
- RNN map a tree into a low dimensional vector e.g. □($\mathbf{x}$) □ ^{30}
- Distributed representation
- Task-driven: □($\mathbf{x}$) in this case depend on the specific learning problem

Kernels

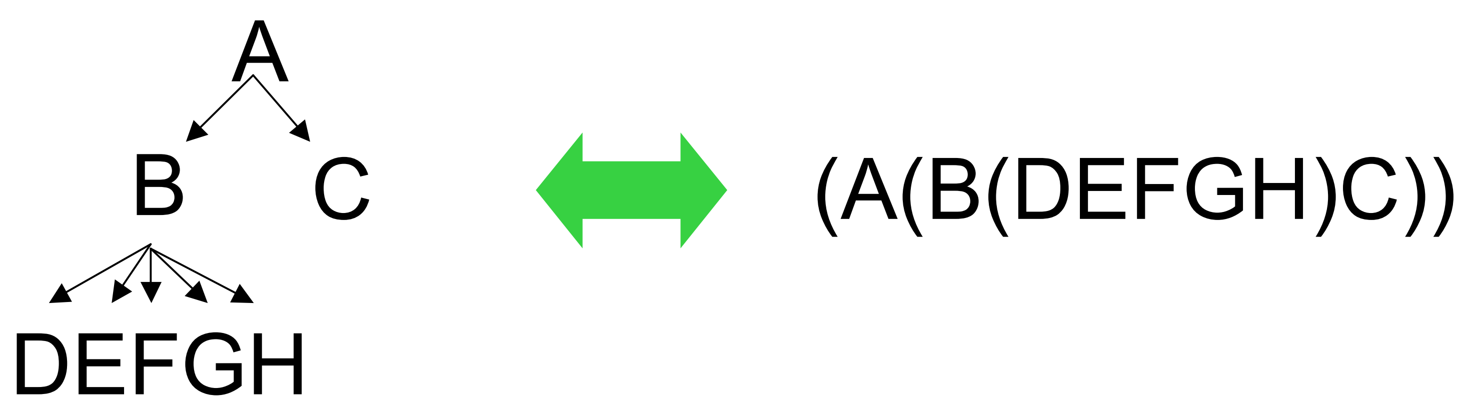
- Given sets of nonterminals $\{A,B,\ldots\}$ and terminals $\{a,b,\ldots\}$ there are infinite possible subtrees:



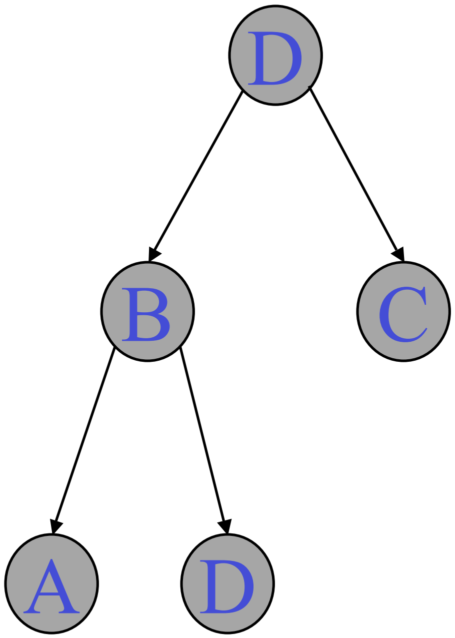
- □$_i(t)$: count occurrences of subtree i in tree t
- □(t)=[□$_1(t)$, □$_2(t)$, □$_3(t)$,…] has infinite dimensionality but
- □$(t)^T$ □(s) can be computed without actually enumerating all subtrees by dynamic programming (Collins & Duffy NIPS 2001)

Recursive neural networks

- Recurrent networks can in principle realize arbitrarily complex dynamical systems
- Skepticism: Long-term dependencies cannot be easily learned
- But trees are different!
  - Path lengths are O(log n)
  - Vanishing gradient problems not as serious for RNNs on trees



Recursive Neural Networks



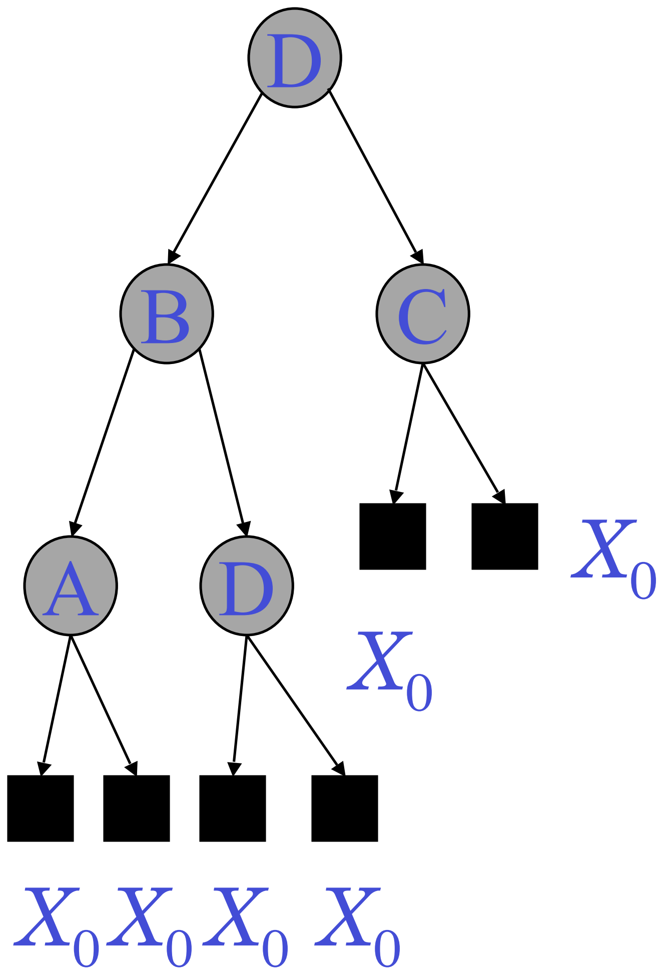
- Let's introduce a representation vector

$$X(v) \in \mathbb{R}^n$$

  for each vertex v in tree t

- $X(v)$ computed bottom-up

Recursive Neural Networks

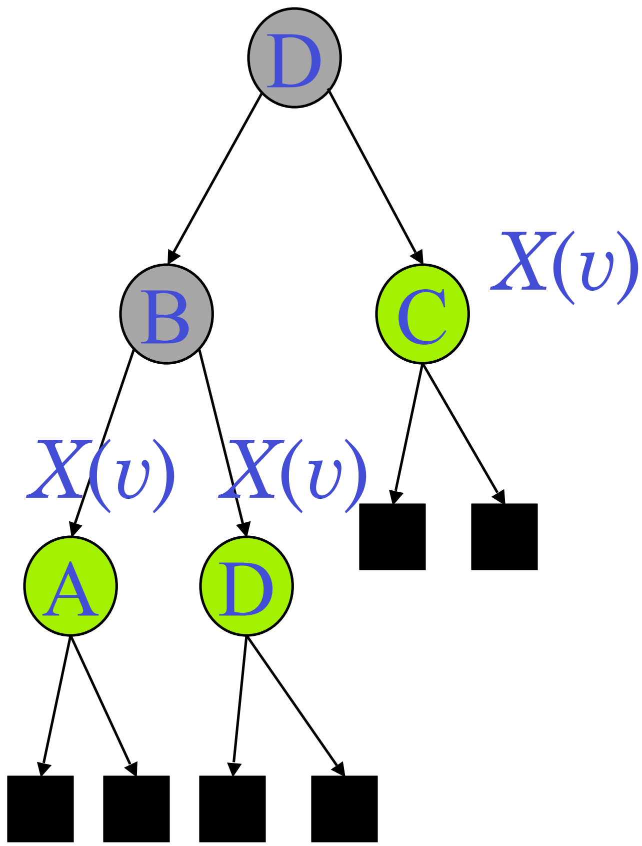


- Base step:

  The representation of external nodes ("nil children") is a constant

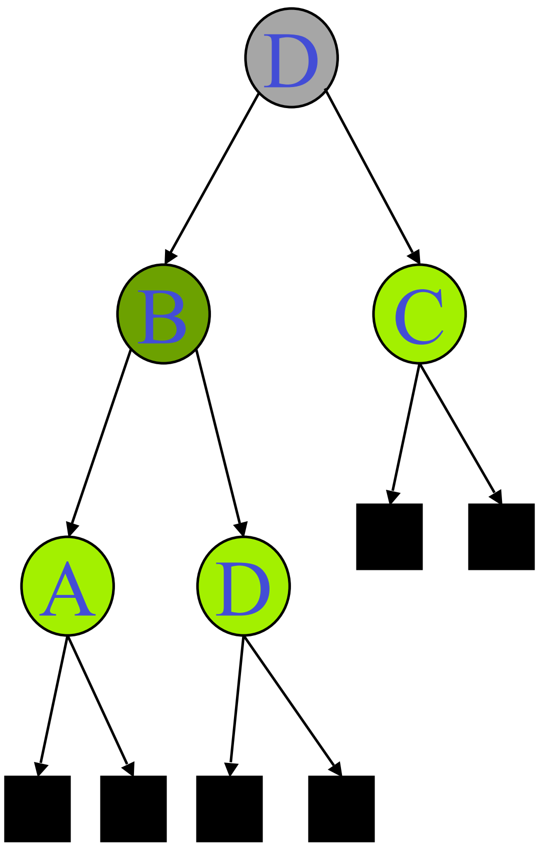
  $X(v) = X_0$

Recursive Neural Networks

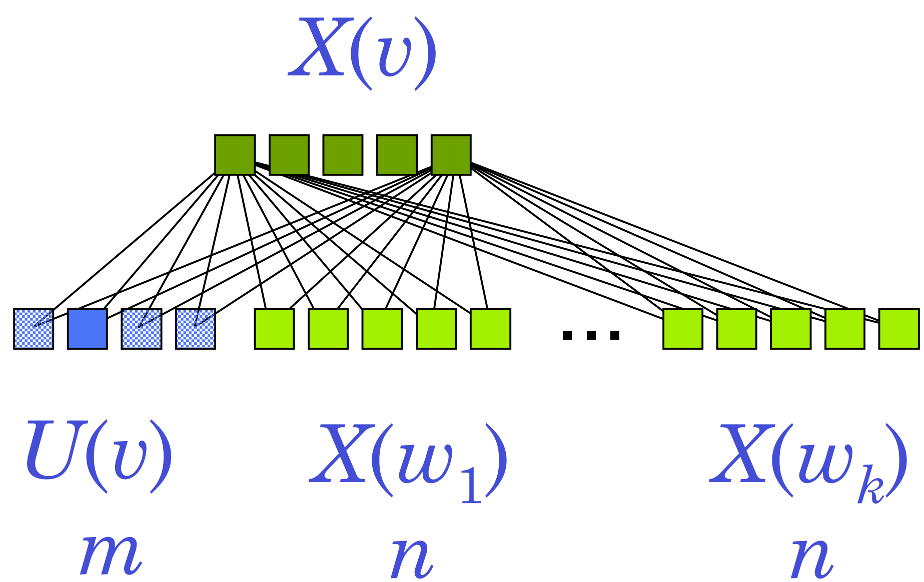


- Induction:
  the representation of the subtree rooted at v is a function of
  1. The representations at the children of v
  2. the symbol $U(v)$
- $X(v) = f(X(w_1),\ldots,X(w_k),U(v))$
- $w_1,\ldots,w_k$ are v's children
  (k assigned)

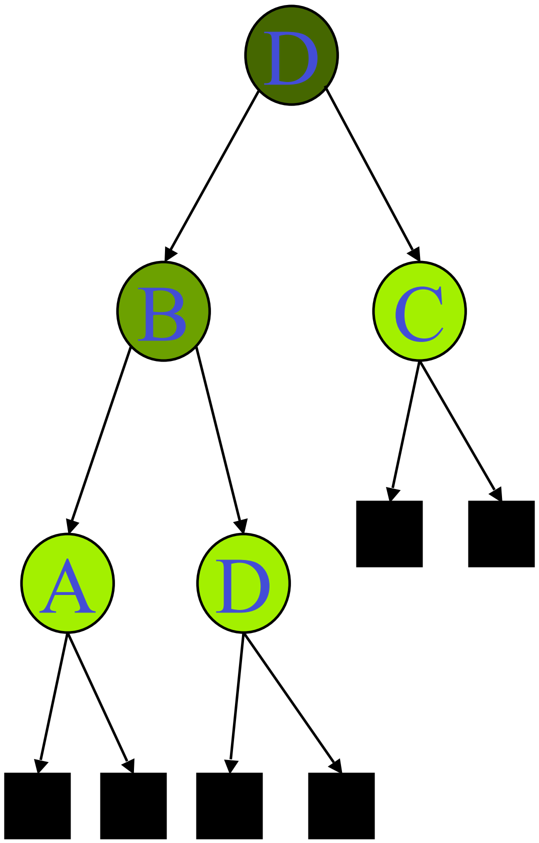
Recursive Neural Networks



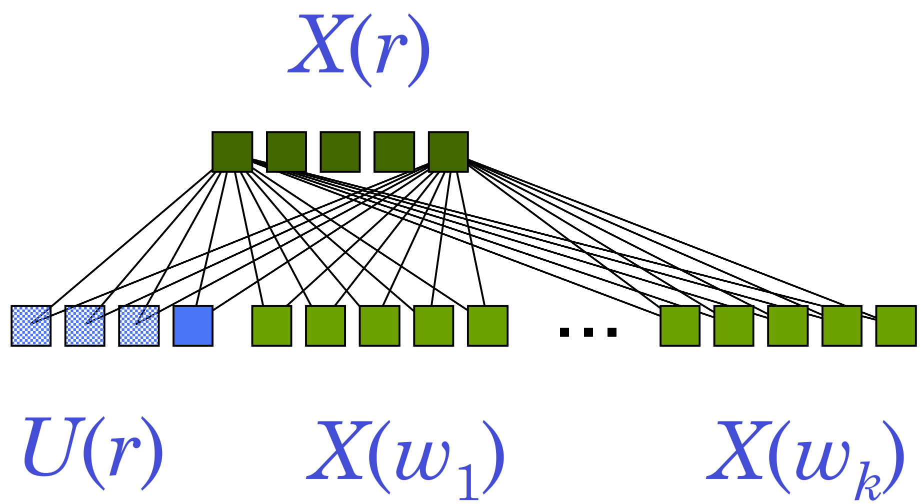
- What is more precisely f ?
  $X(v) = f(X(w_1),\ldots,X(w_k),U(v))$
- f is realized by an MLP:
  - n outputs, $nk+m$ inputs



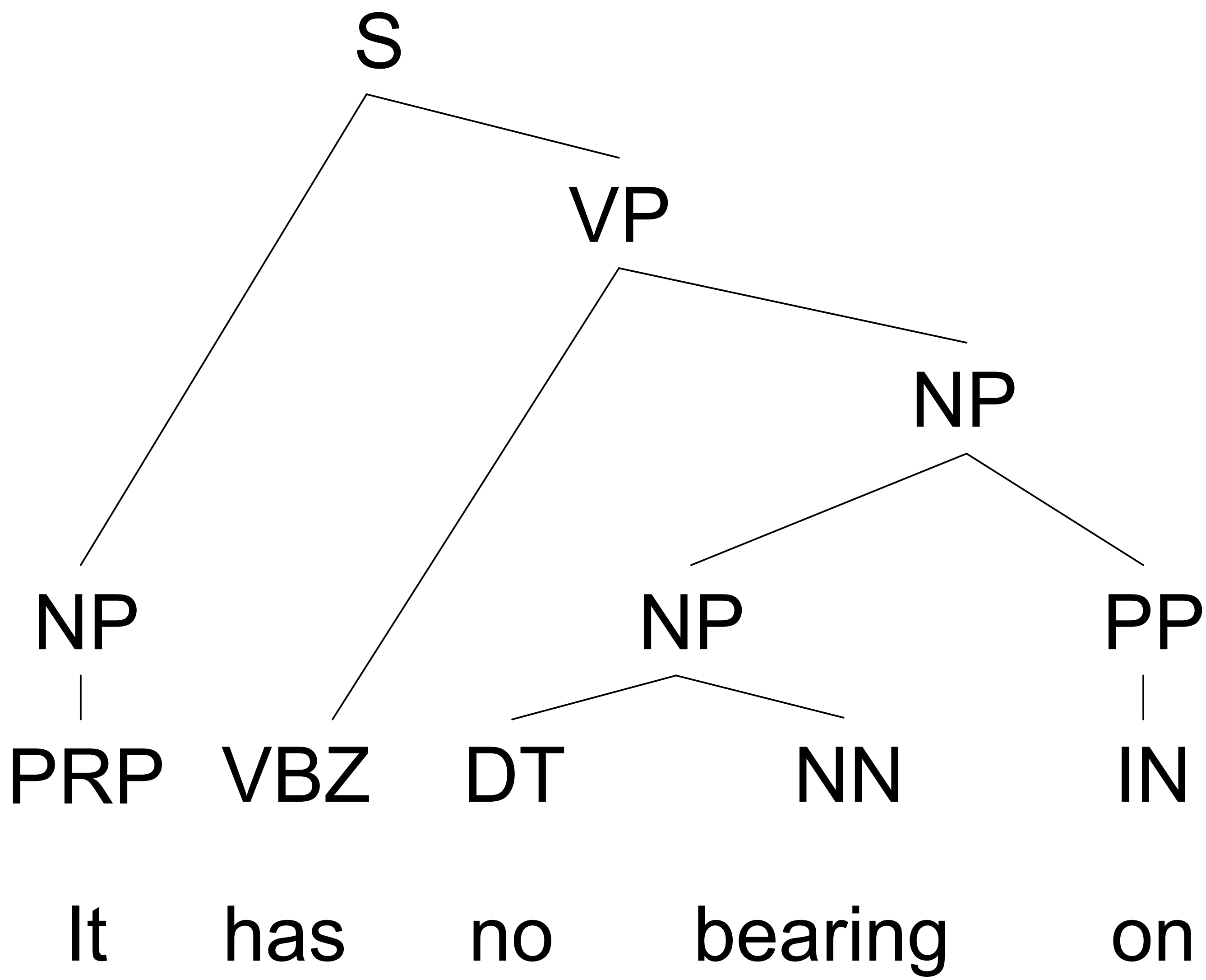
Recursive Neural Networks



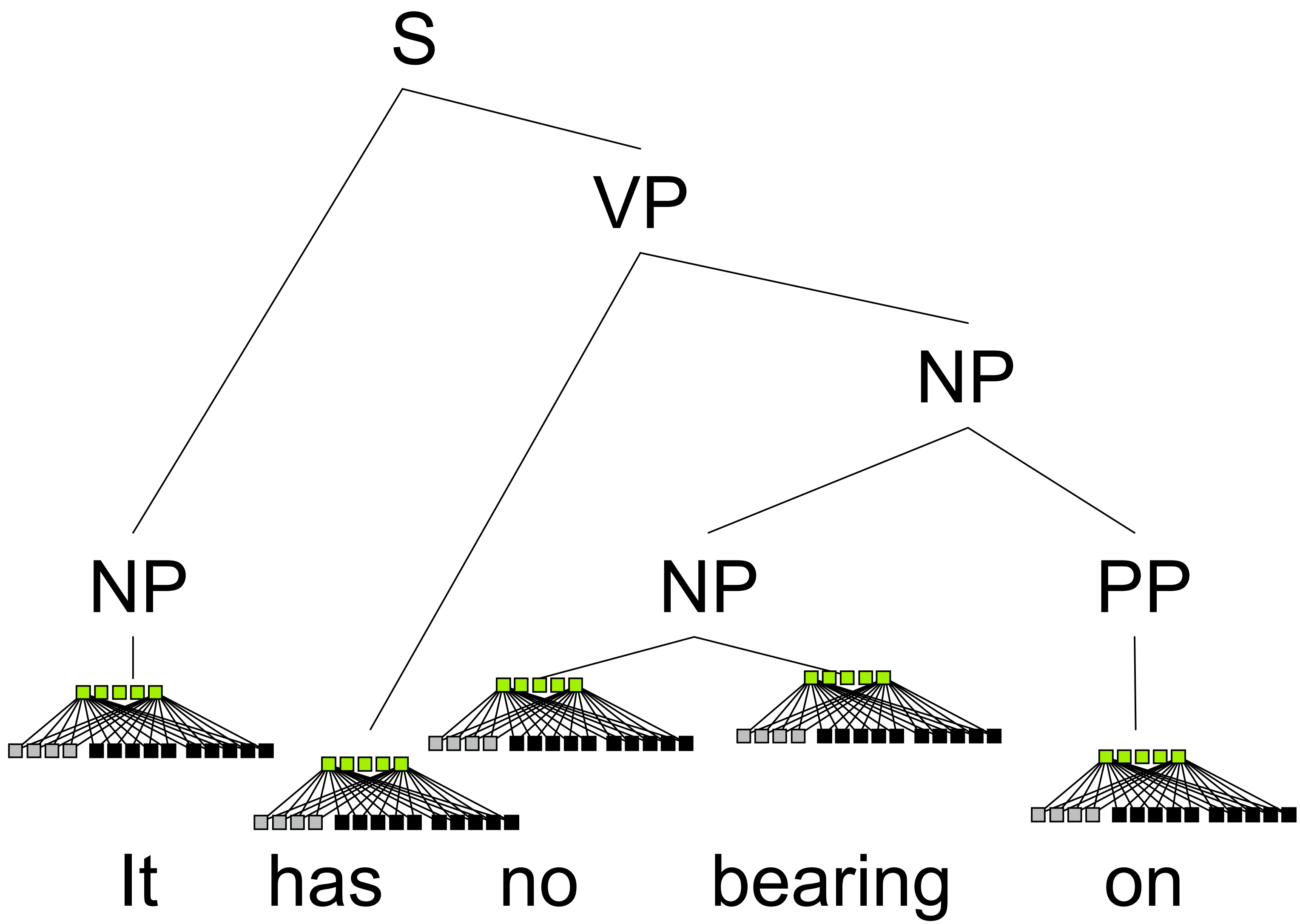
- The computation continues bottom-up until the root r is reached
- $X(r)$ encodes the whole tree in a real vector — same role as □(t)



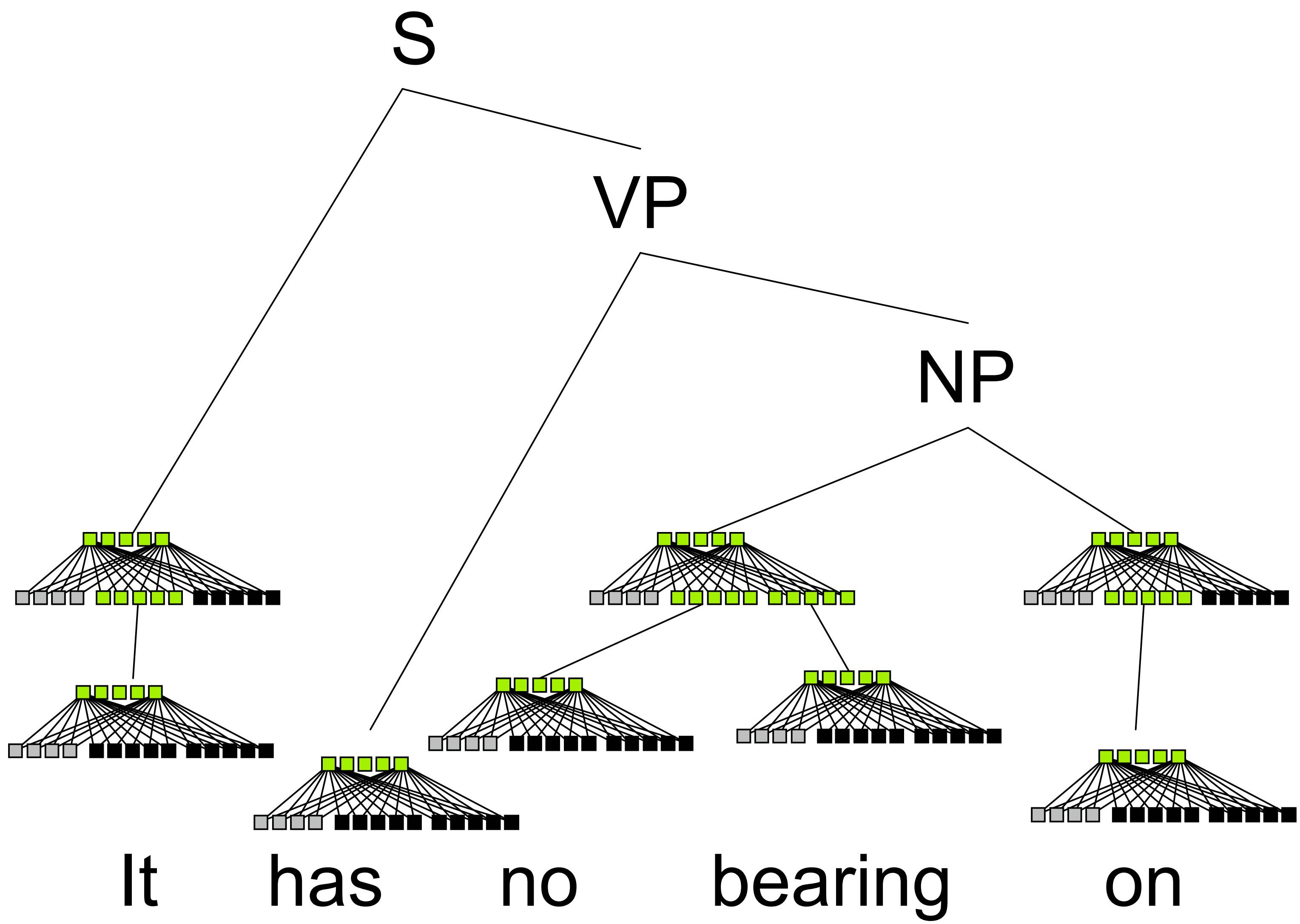
Structure unfolding



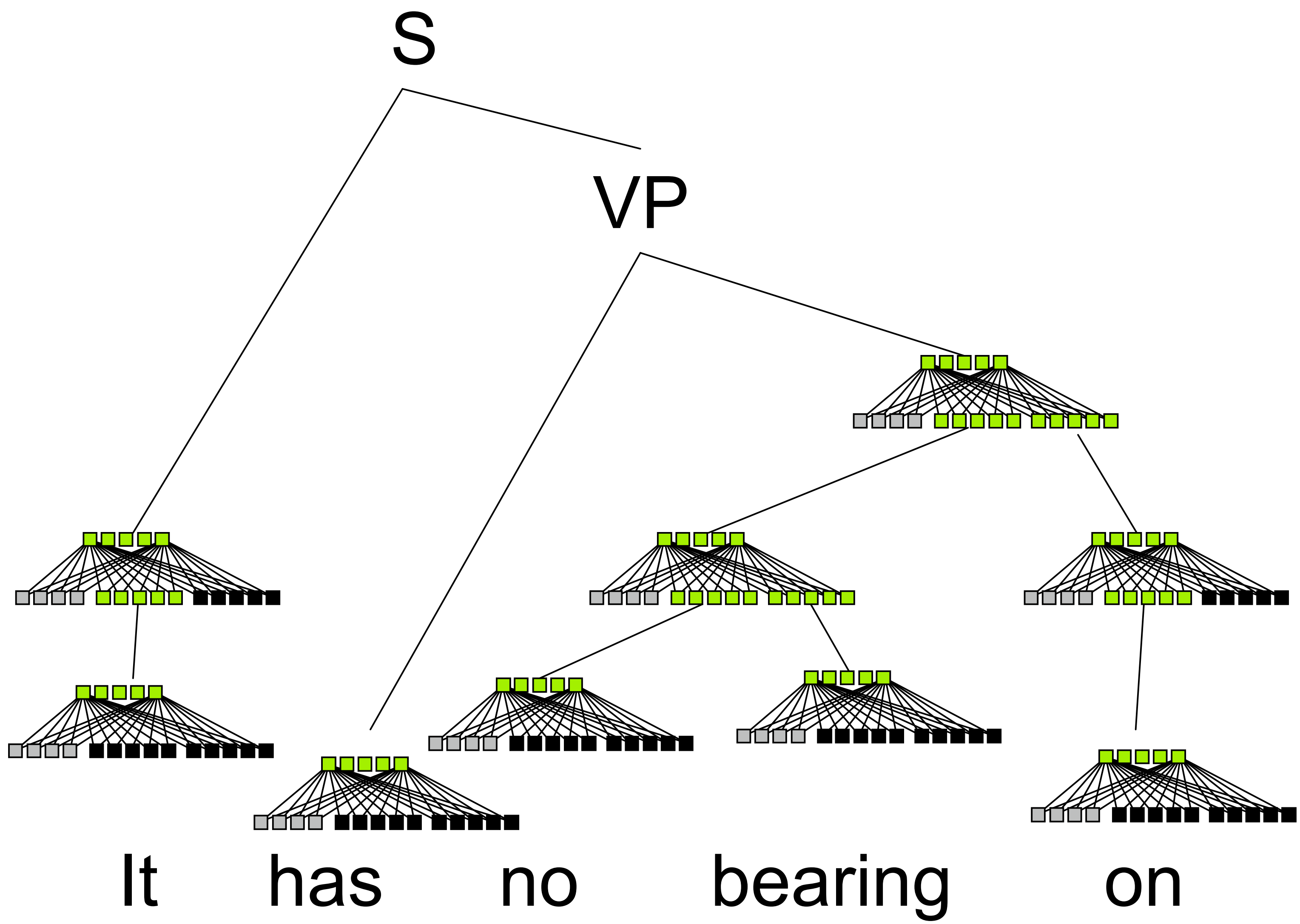
Structure unfolding



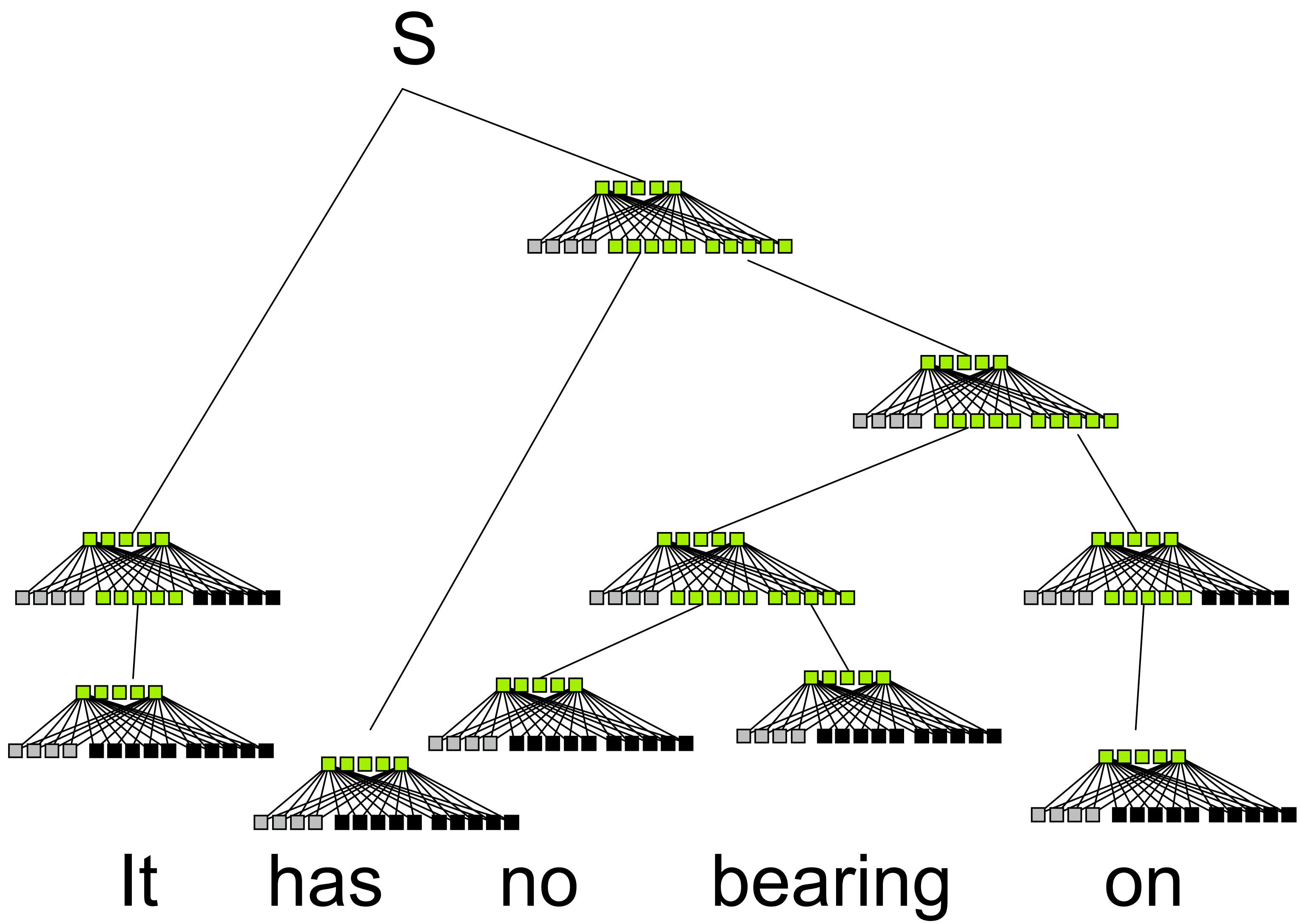
Structure unfolding



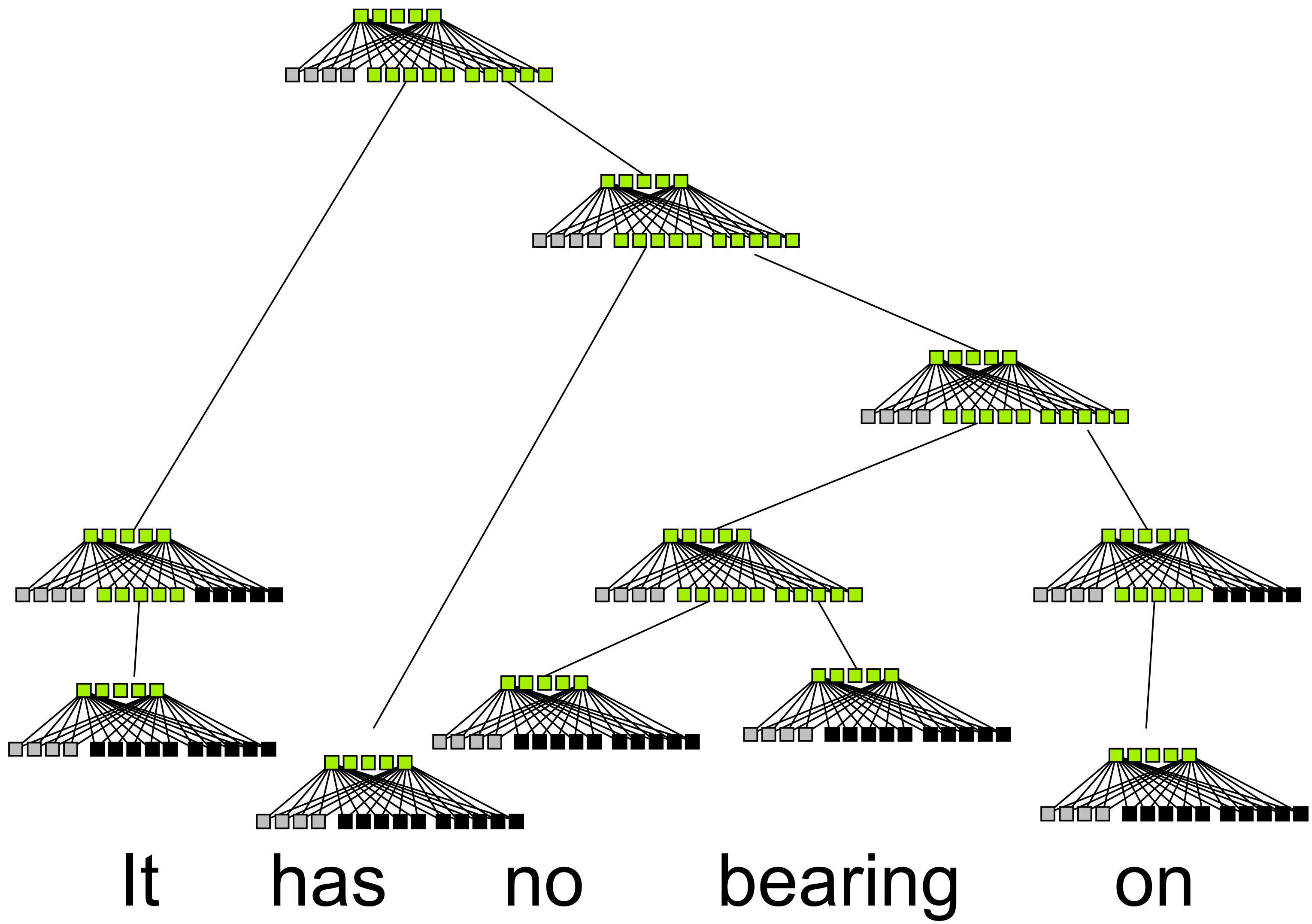
Structure unfolding



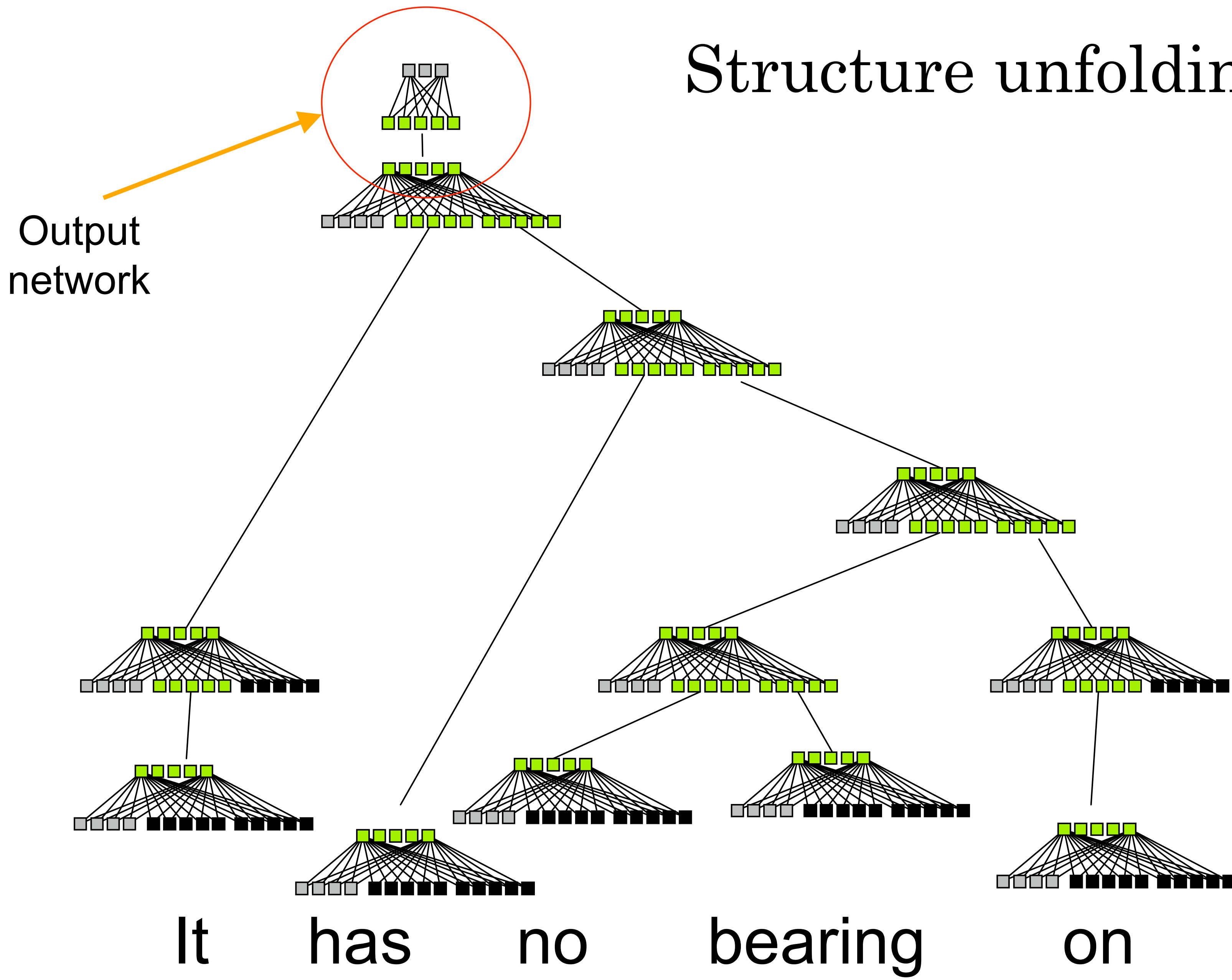
Structure unfolding



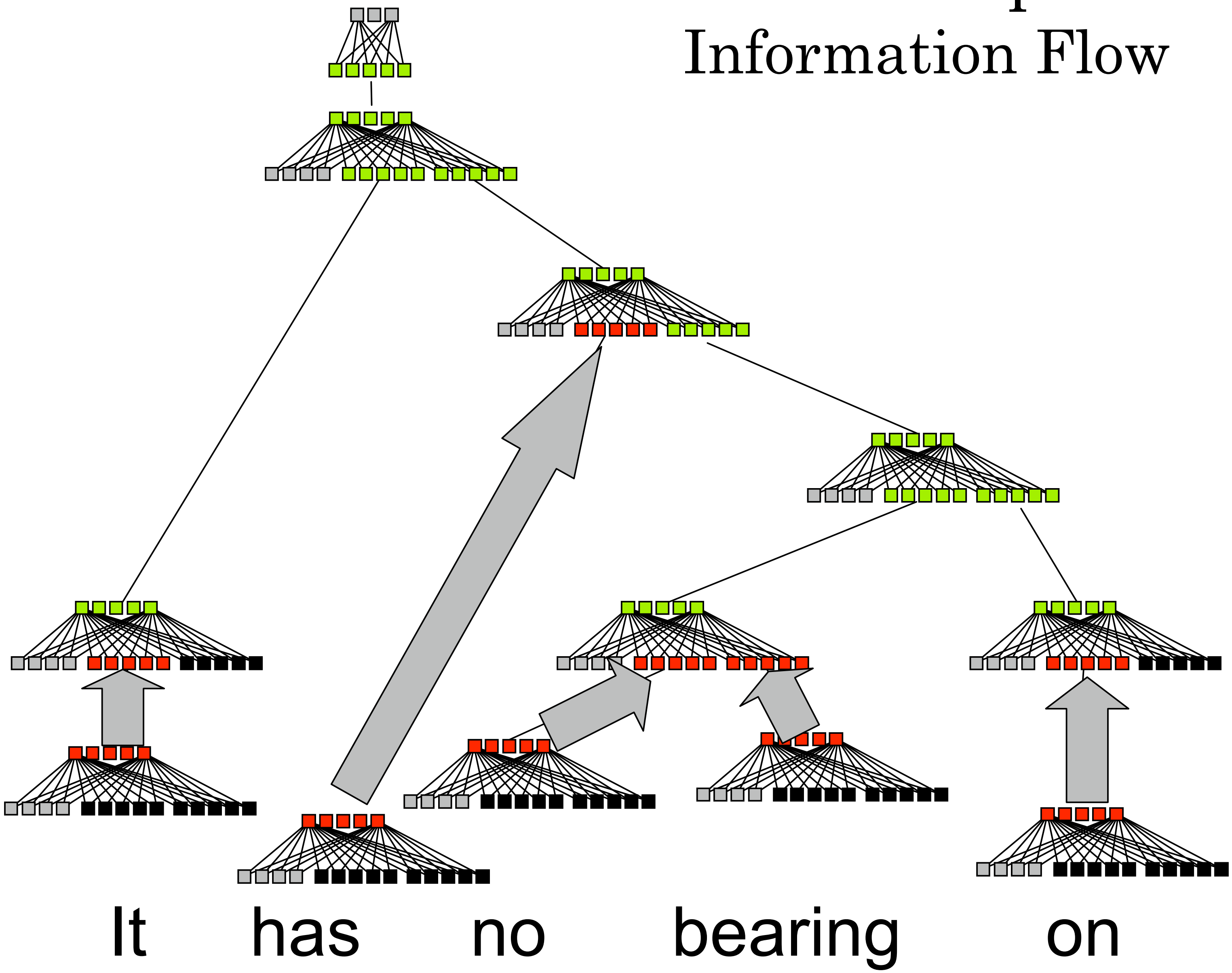
Structure unfolding

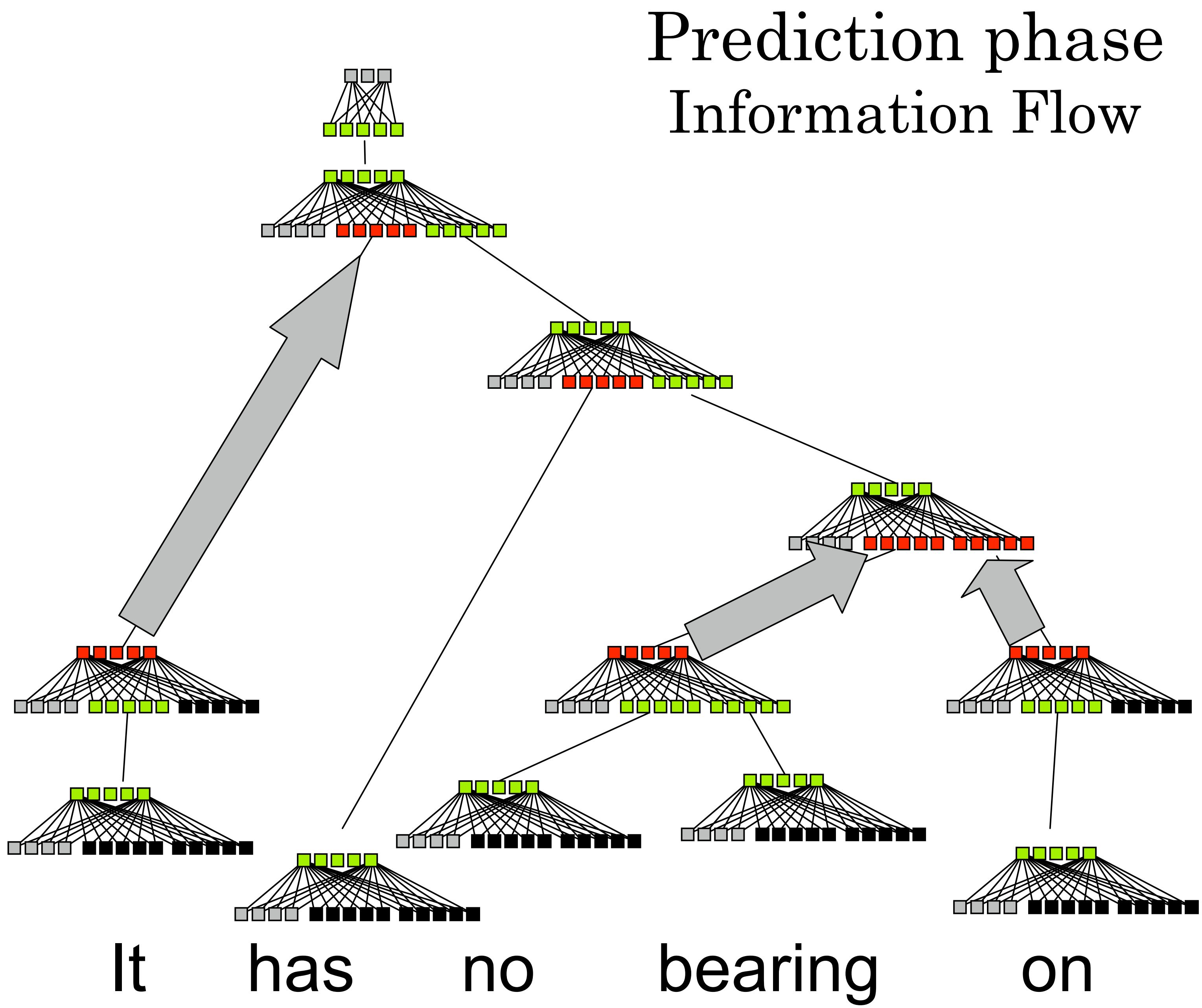


Structure unfolding

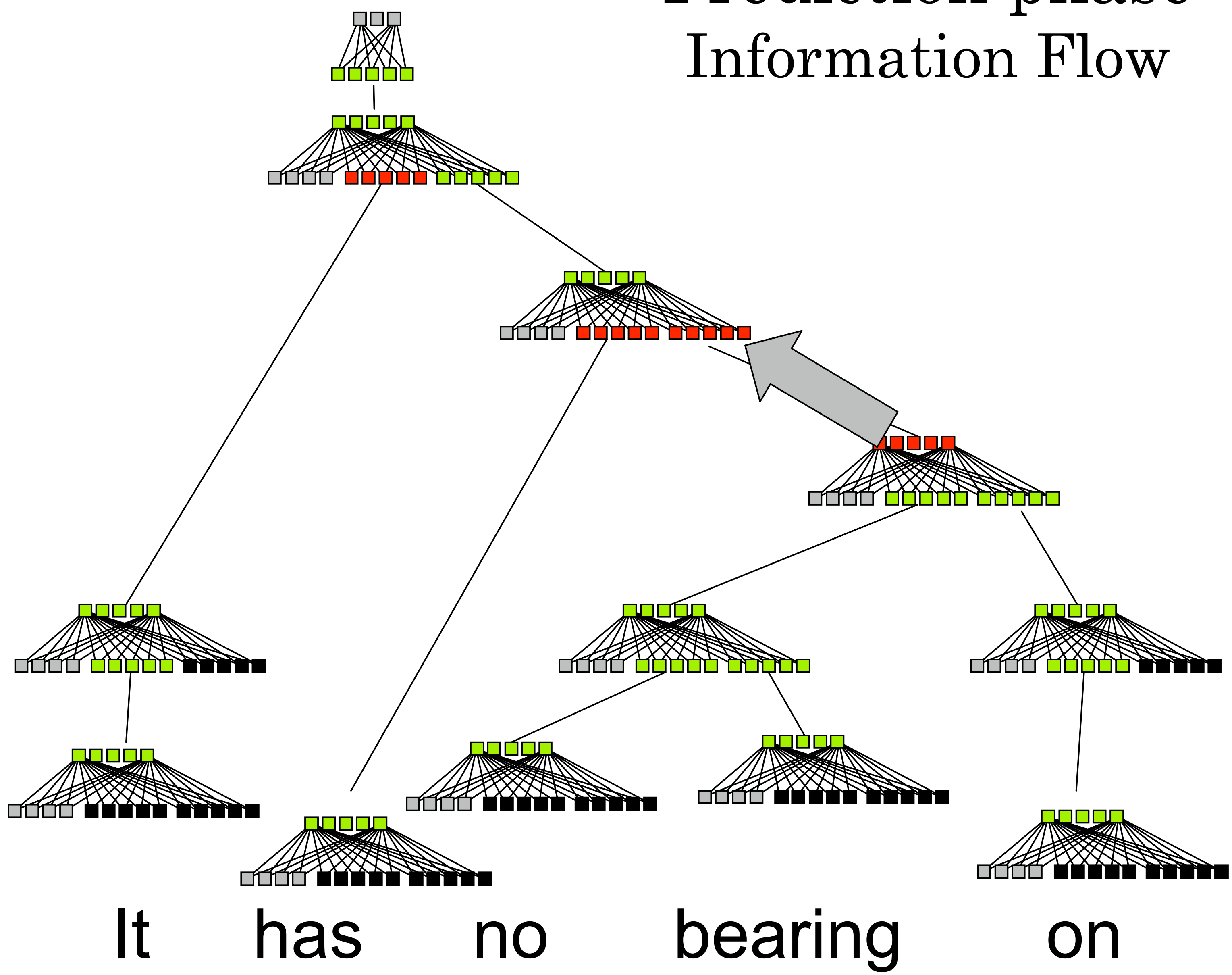


Prediction phase
Information Flow

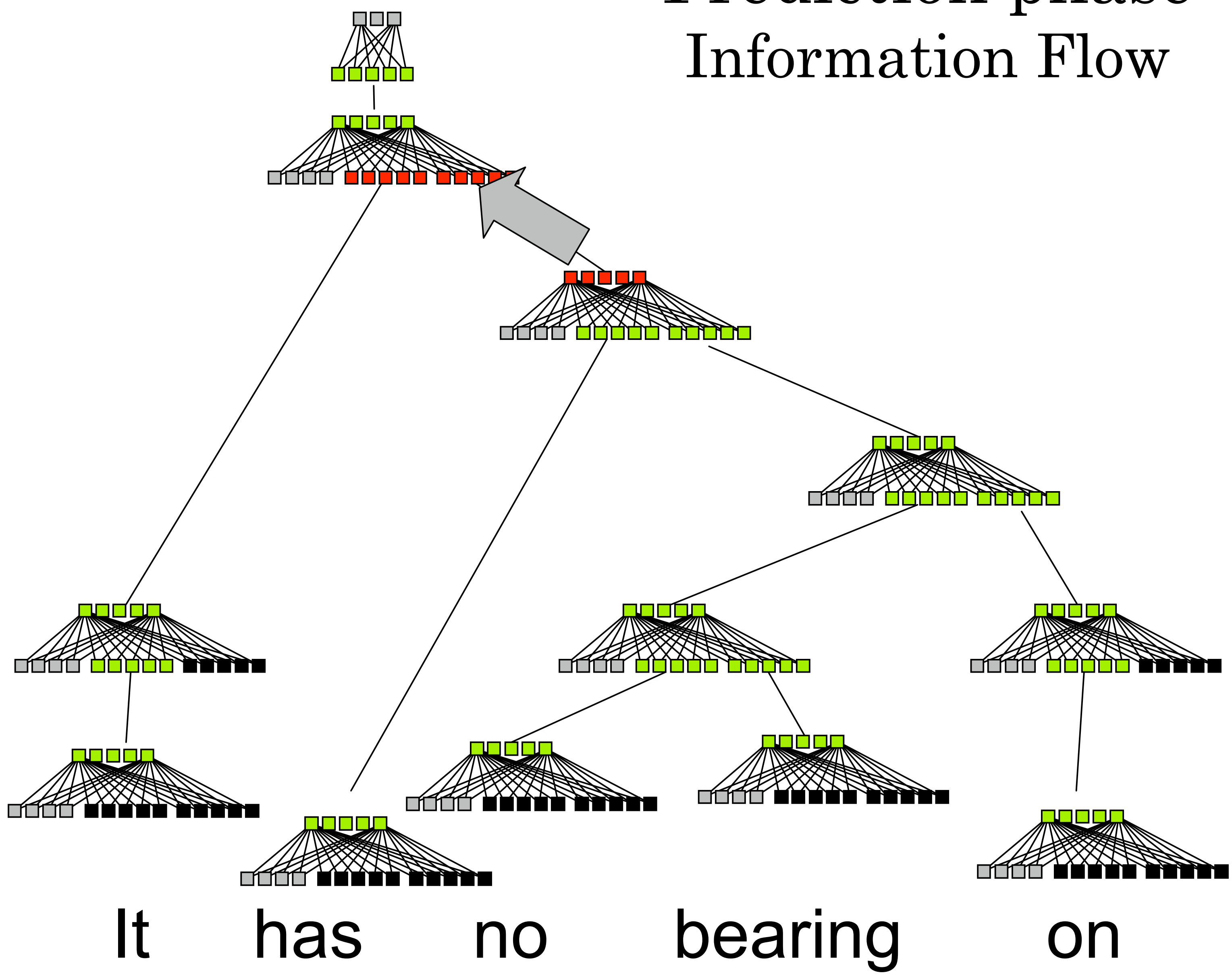



Prediction phase
Information Flow
It
has
no
bearing
on

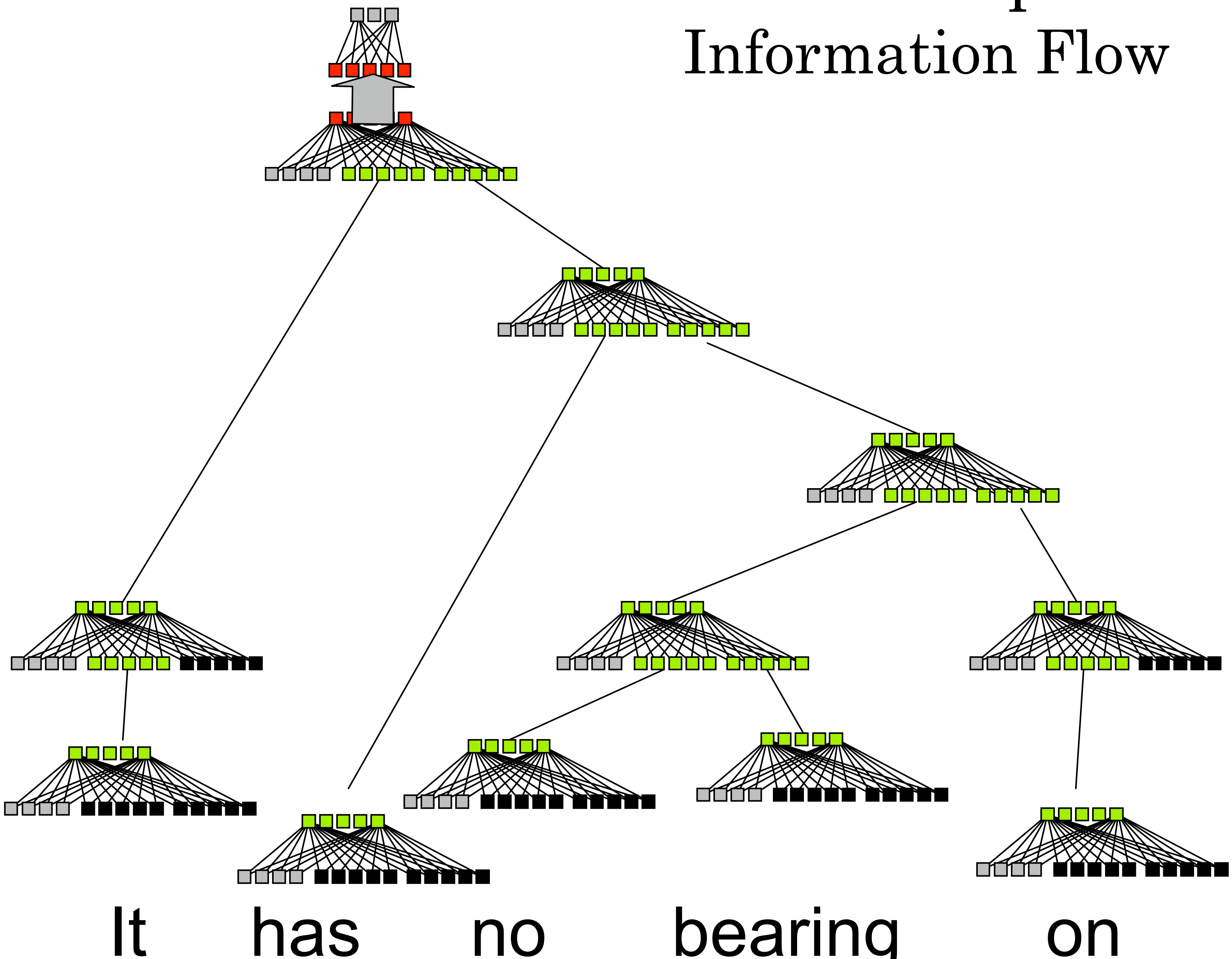
Prediction phase
Information Flow



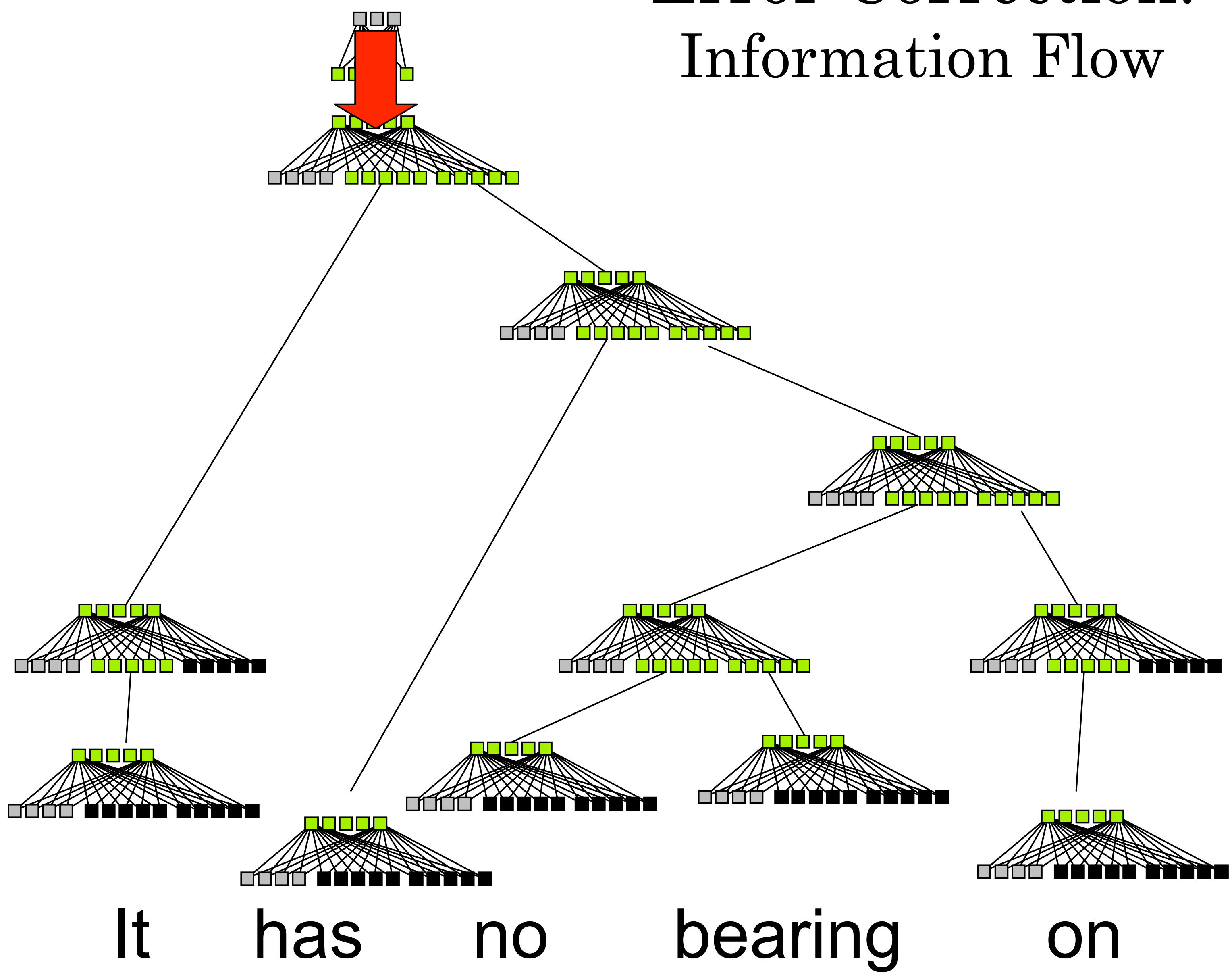
Prediction phase
Information Flow



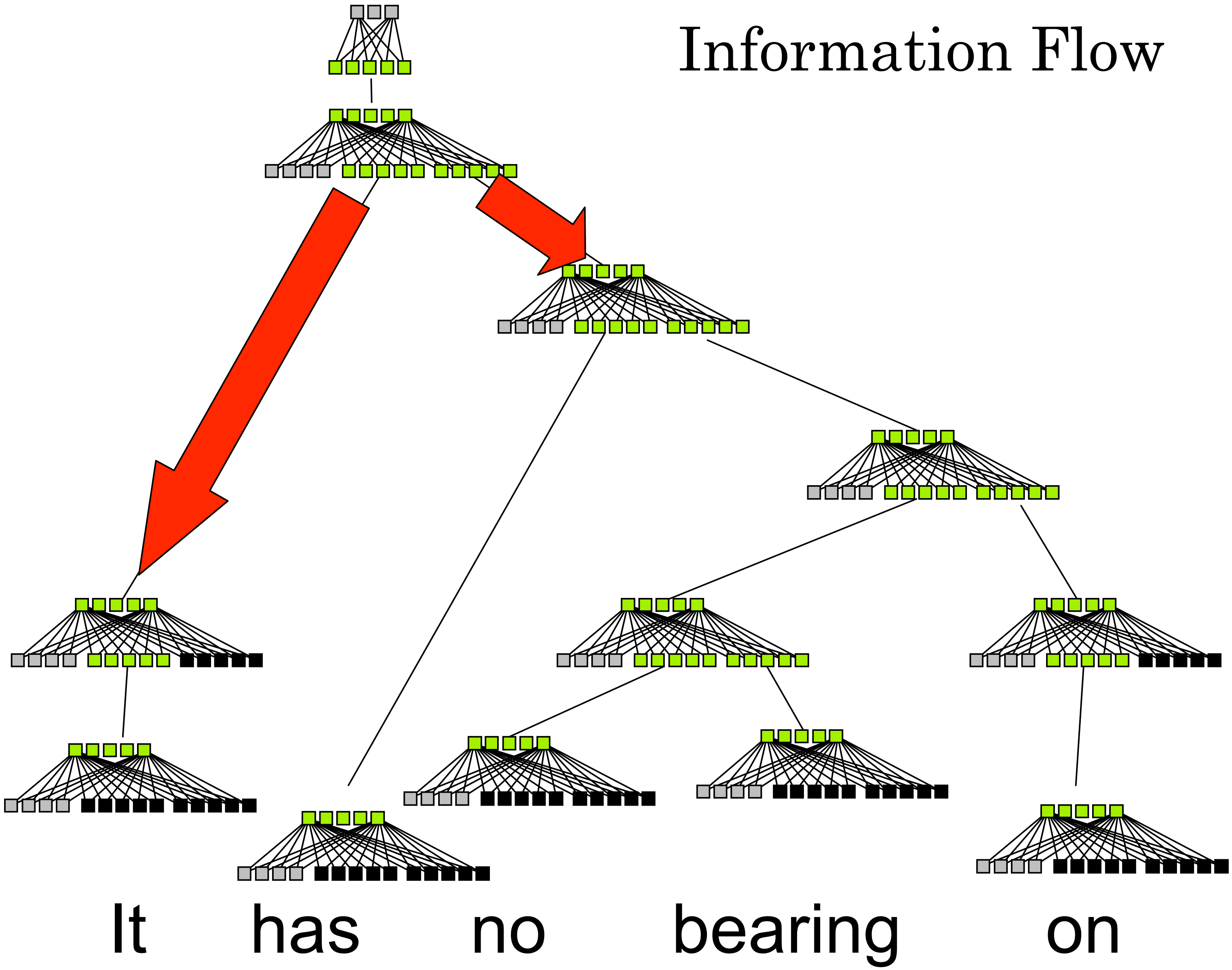
Prediction phase
Information Flow



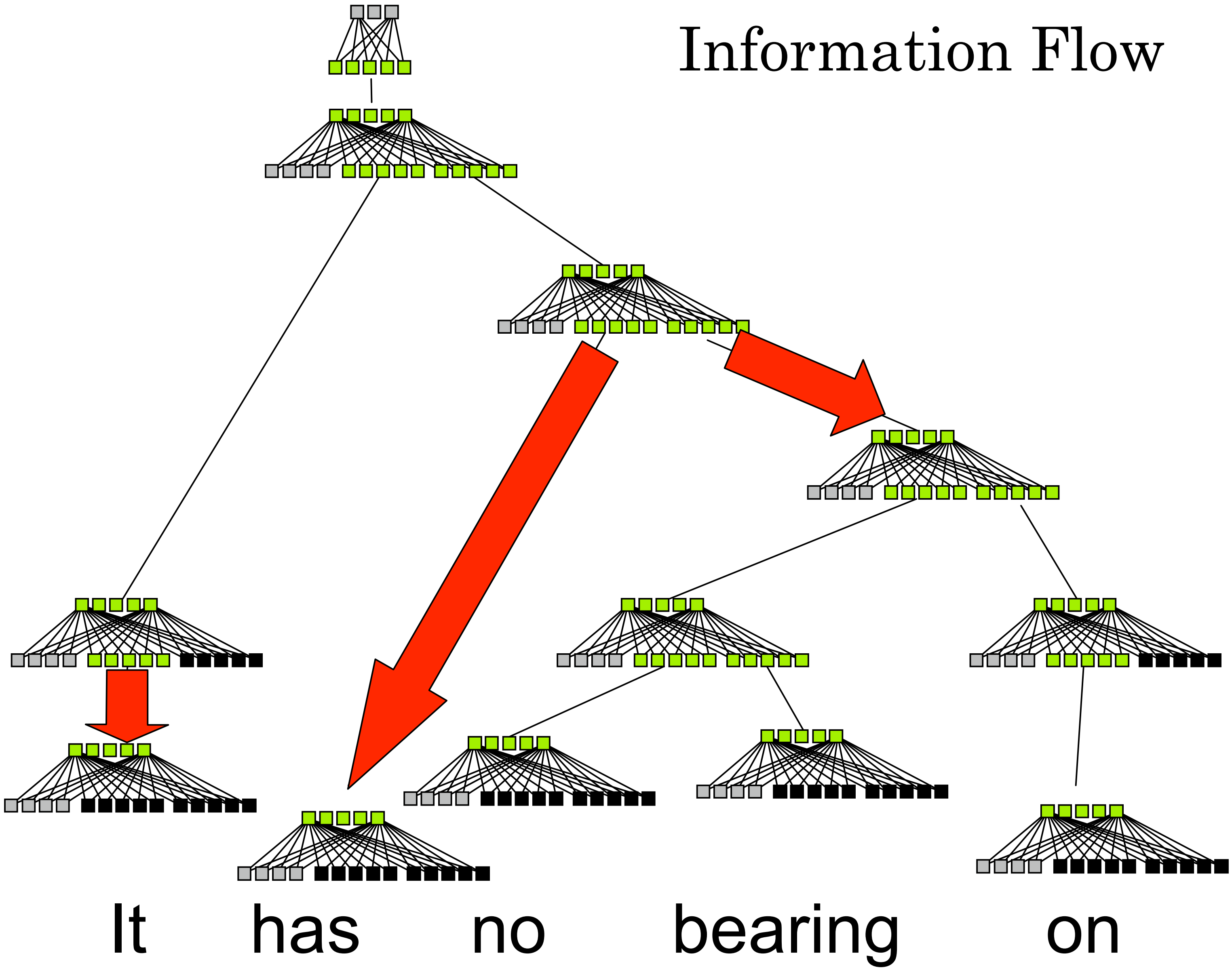
Error Correction: Information Flow



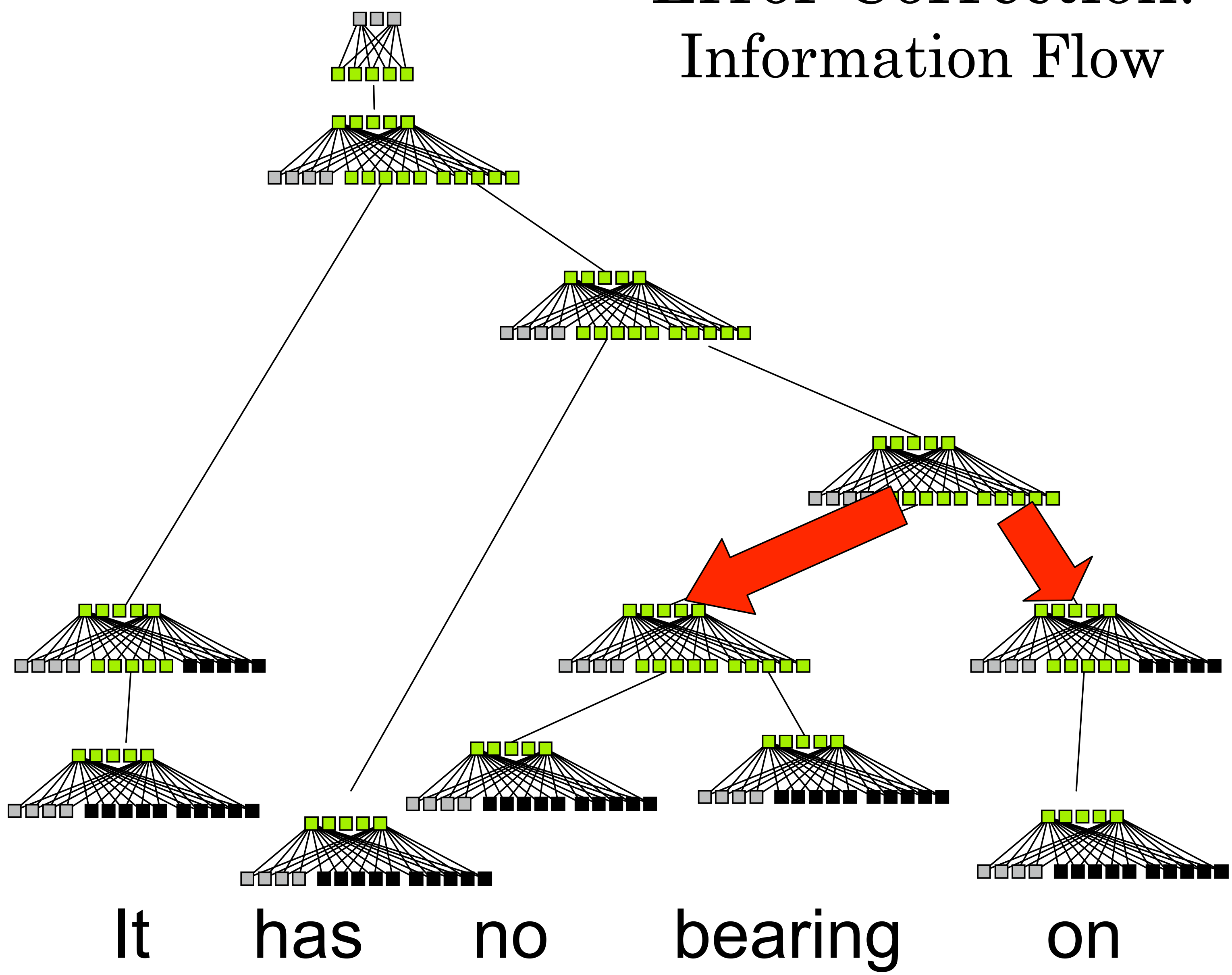
Error Correction: Information Flow



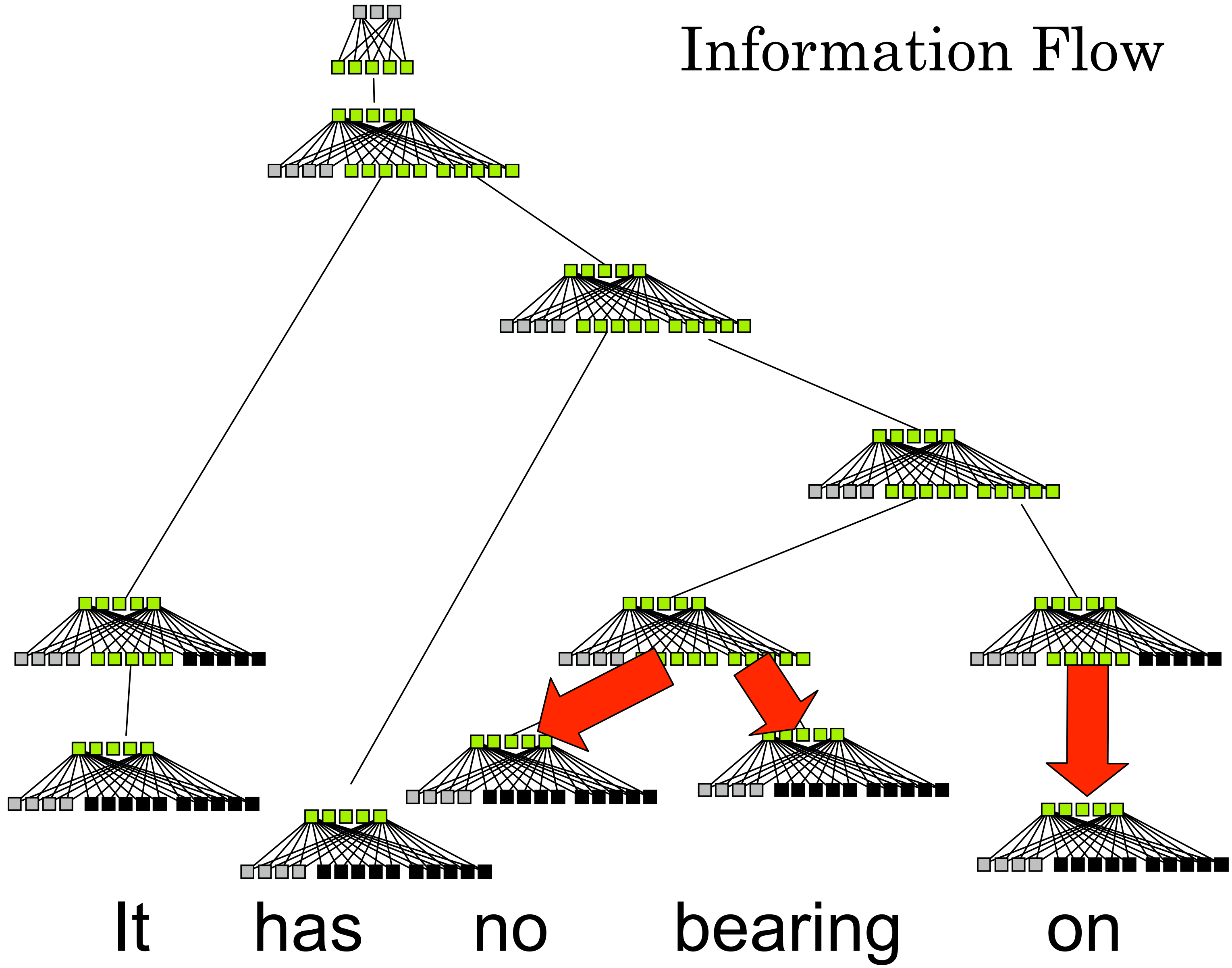
Error Correction:
Information Flow



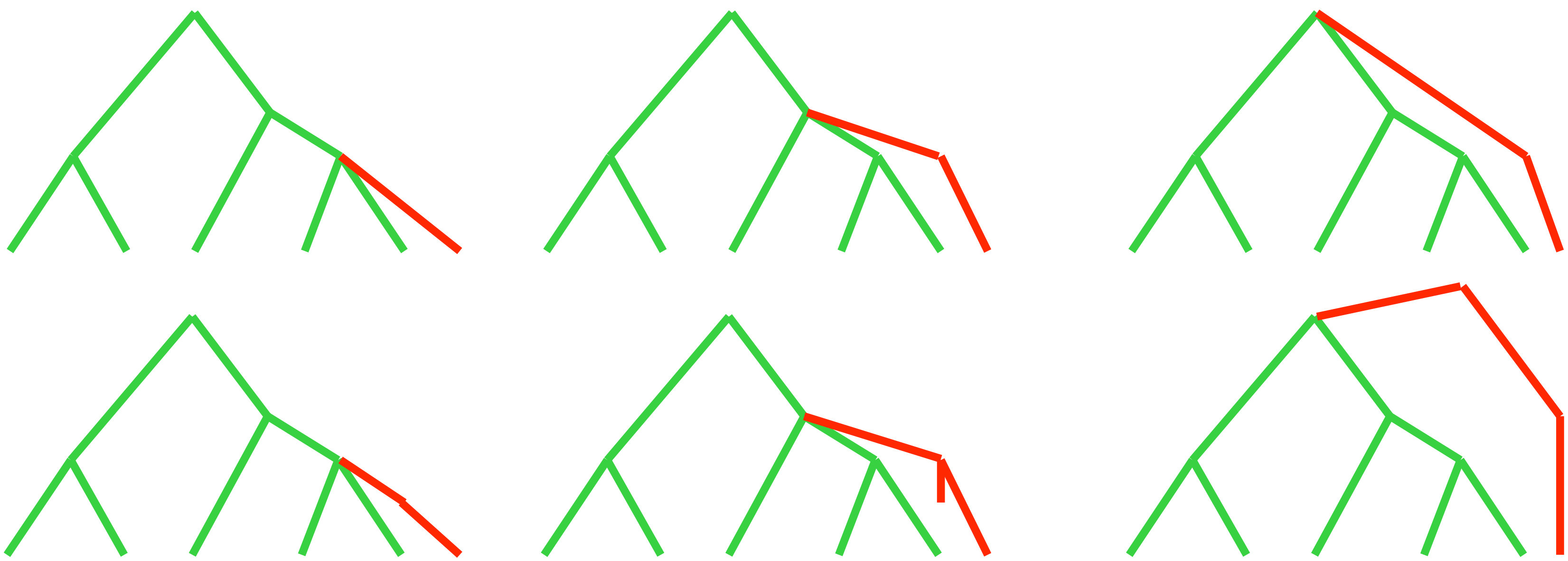
Error Correction: Information Flow



Error Correction: Information Flow



Disambiguation is a preference task



Learning preferences

- Ranking: given an list of entities $(\mathbf{x}_1,\ldots,\mathbf{x}_r)$ find a corresponding list of integers $(y_1,\ldots,y_r)$, with y_i in $[1,r]$ such that y_i is the rank of $\mathbf{x}_i$
- In total ranking: $y_i \neq y_j$
- In our case the favorite element $\mathbf{x}_1$ gets $y_1=1$ and other $\mathbf{x}_j$ get $y_j=0$
  - typically $r=120$ (but goes up to 2000)
- Linear utility function:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_1 - \mathbf{w}^{\mathrm{T}}\mathbf{x}_j > 0 \quad \text{for } j=2,\ldots,r$$

- Set of constraints — similar to binary classification but we have differences between vectors
- Can be used with SVM and Voting Perceptron:

$$\mathbf{w}^{\mathrm{T}}[\square(\mathbf{x}_1) - \square(\mathbf{x}_j)] = \square_{\underline{\mathrm{sv}}}\, y[\square(\underline{\mathbf{x}}_1) - \square(\underline{\mathbf{x}}_j)]^{\mathrm{T}}[\square(\mathbf{x}_1) - \square(\mathbf{x}_j)]$$

Learning preferences

- To get a differentiable version we use the softmax function

$$y_j = \frac{e^{\mathbf{w}^T \mathbf{x}_j}}{\sum_k e^{\mathbf{w}^T \mathbf{x}_k}}$$

- Find $\mathbf{w}$ and $\mathbf{x}_j$ by maximizing

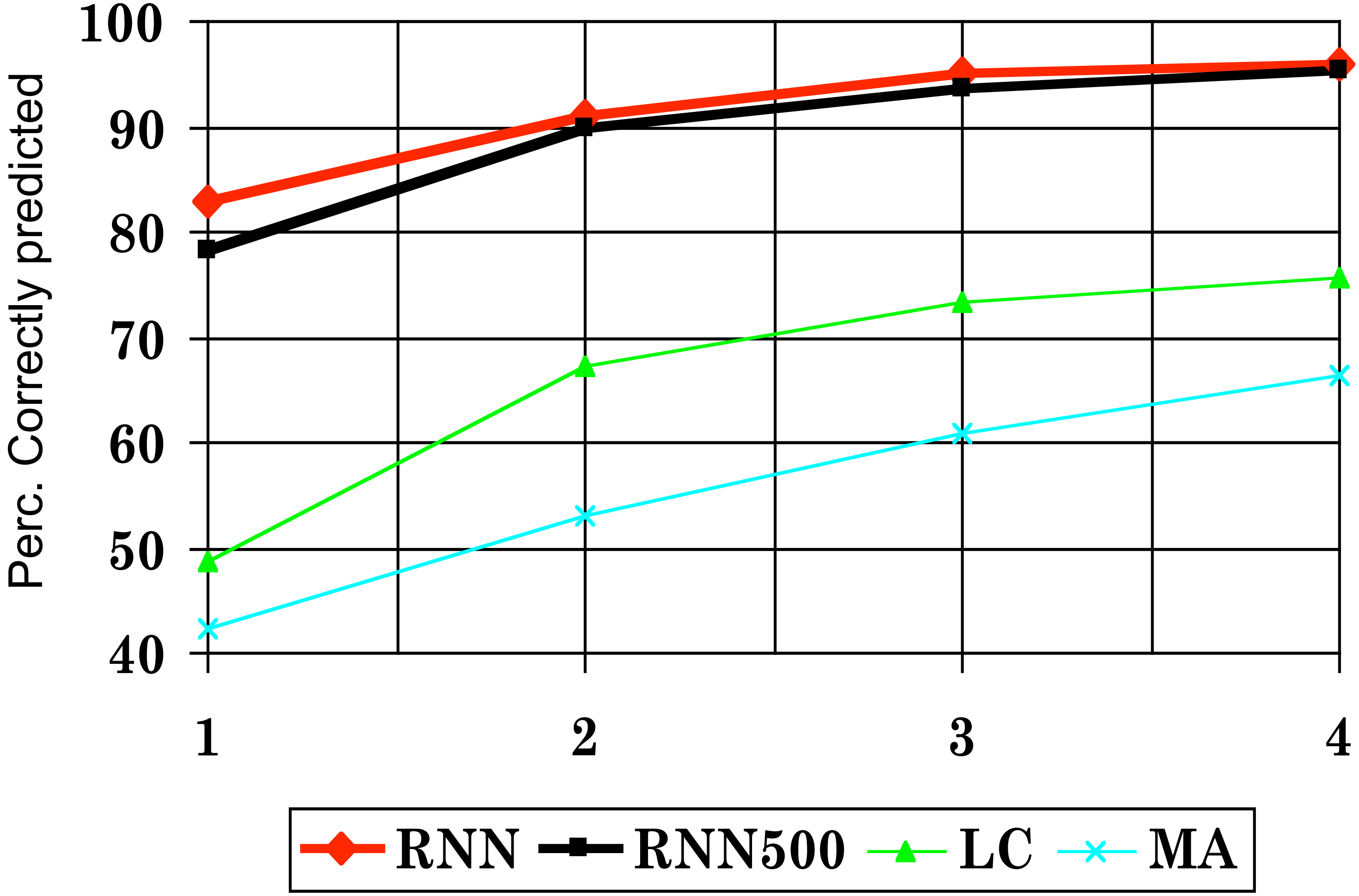
$$\sum_i \sum_j z_j \log y_j + (1 - z_j) \log(1 - y_j)$$

- Where z_1=1 and z_j=0 for j>1
- Gradients wrt $\mathbf{x}_j$ are passed to the RNN so in this sense $\mathbf{x}_j$ is an adaptive encoding

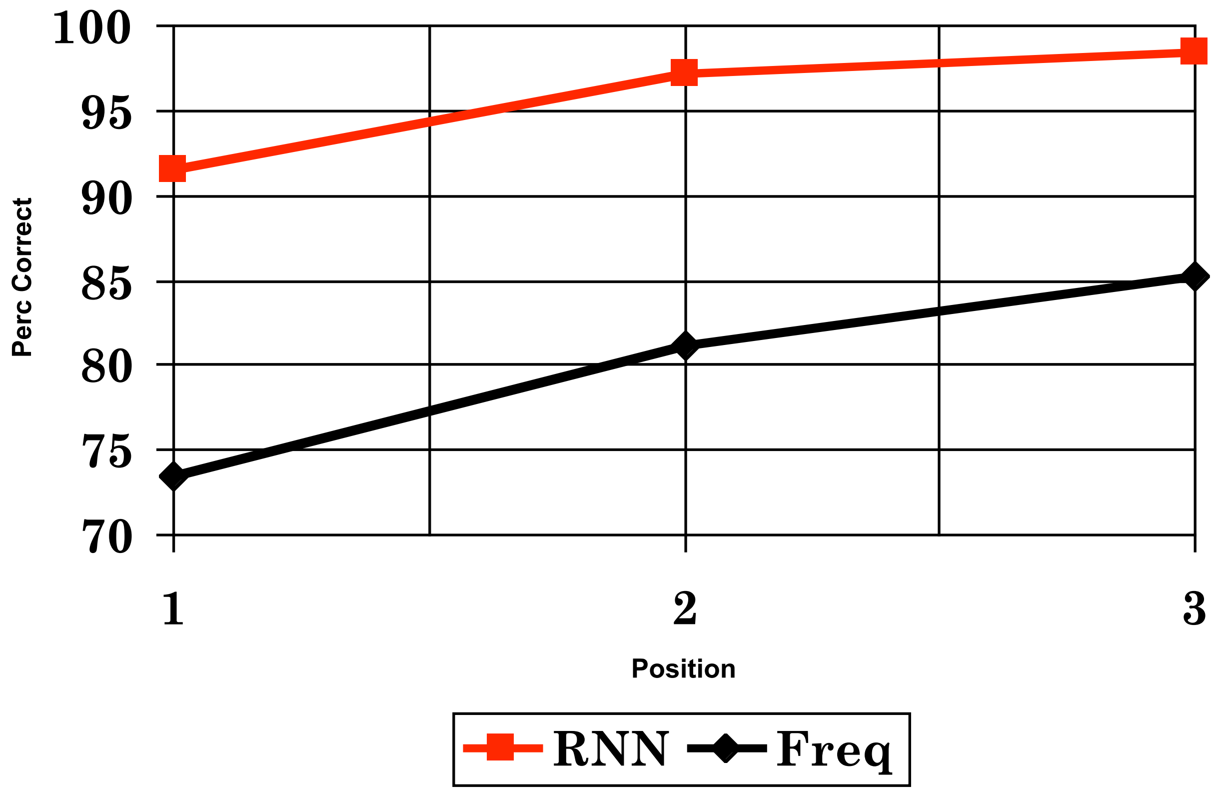
Experimental setup

- Training on WSJ section of Penn treebank
  - realistic corpus representative of natural language
  - large size (40.000 sentence, 1 million words)
  - uniform language (articles on economic subject)
  - Train on sections 2-21, test on section 23
- Note: we are not (yet) into building a parser
- Extending earlier results (Costa et al 2000, Sturt et al, *Cognition*, in press)

Results

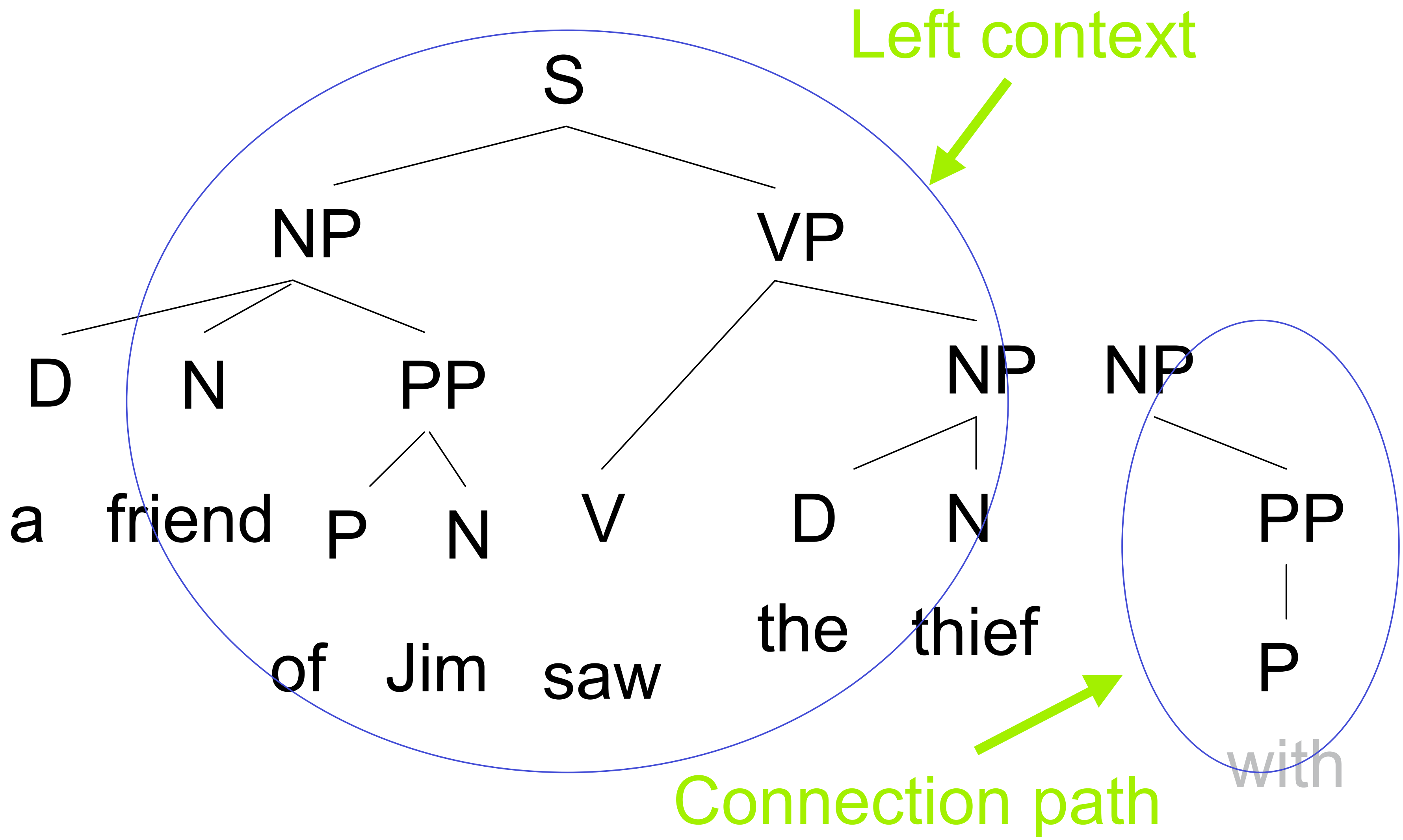

Perc. Correctly predicted
100
90
80
70
60
50
40
1
2
3
4
RNN
RNN500
LC
MA

Selecting the right attachment



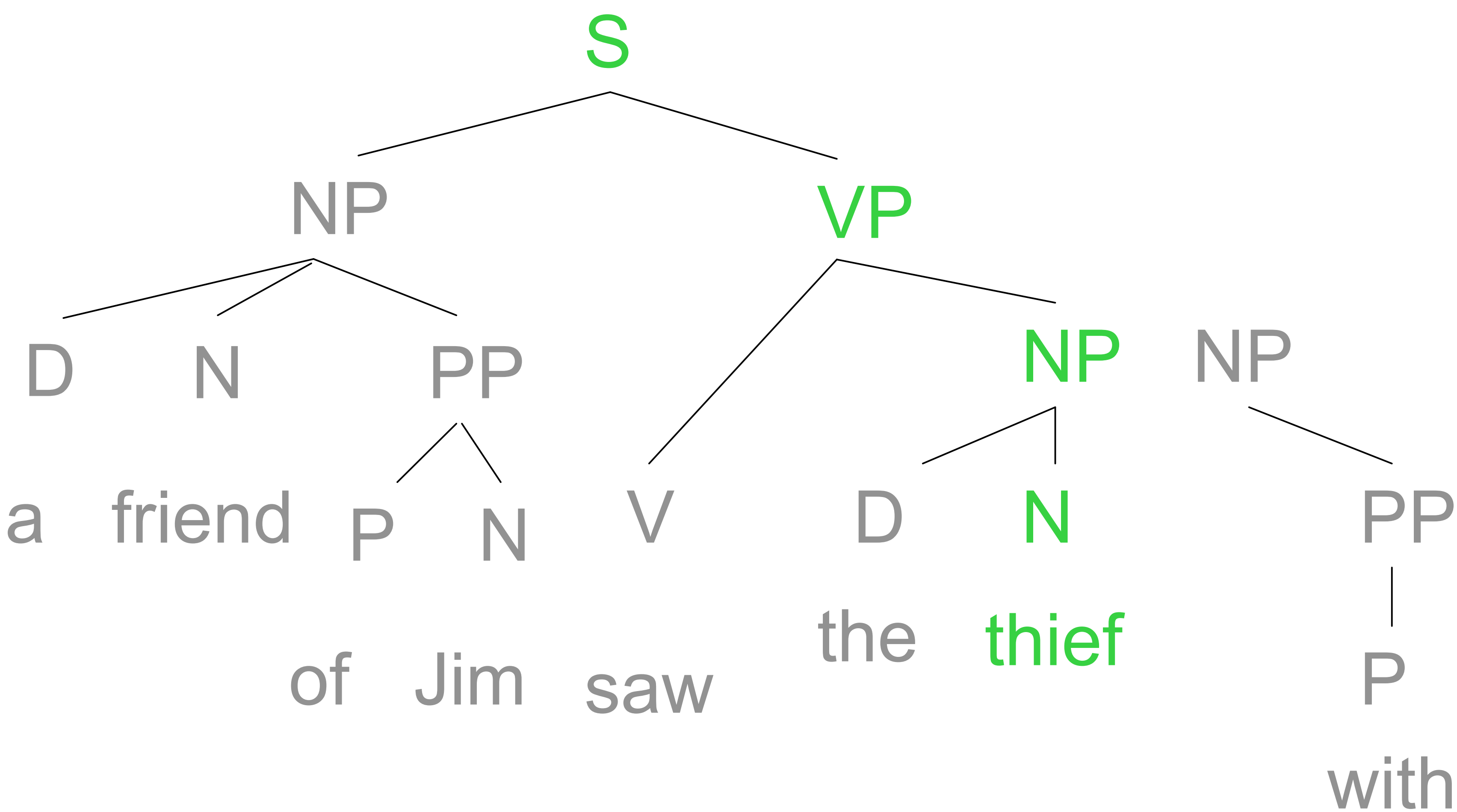
- Given attachment-site, correct connection-path is chosen 89% of the time

Reduced incremental trees
Example tree



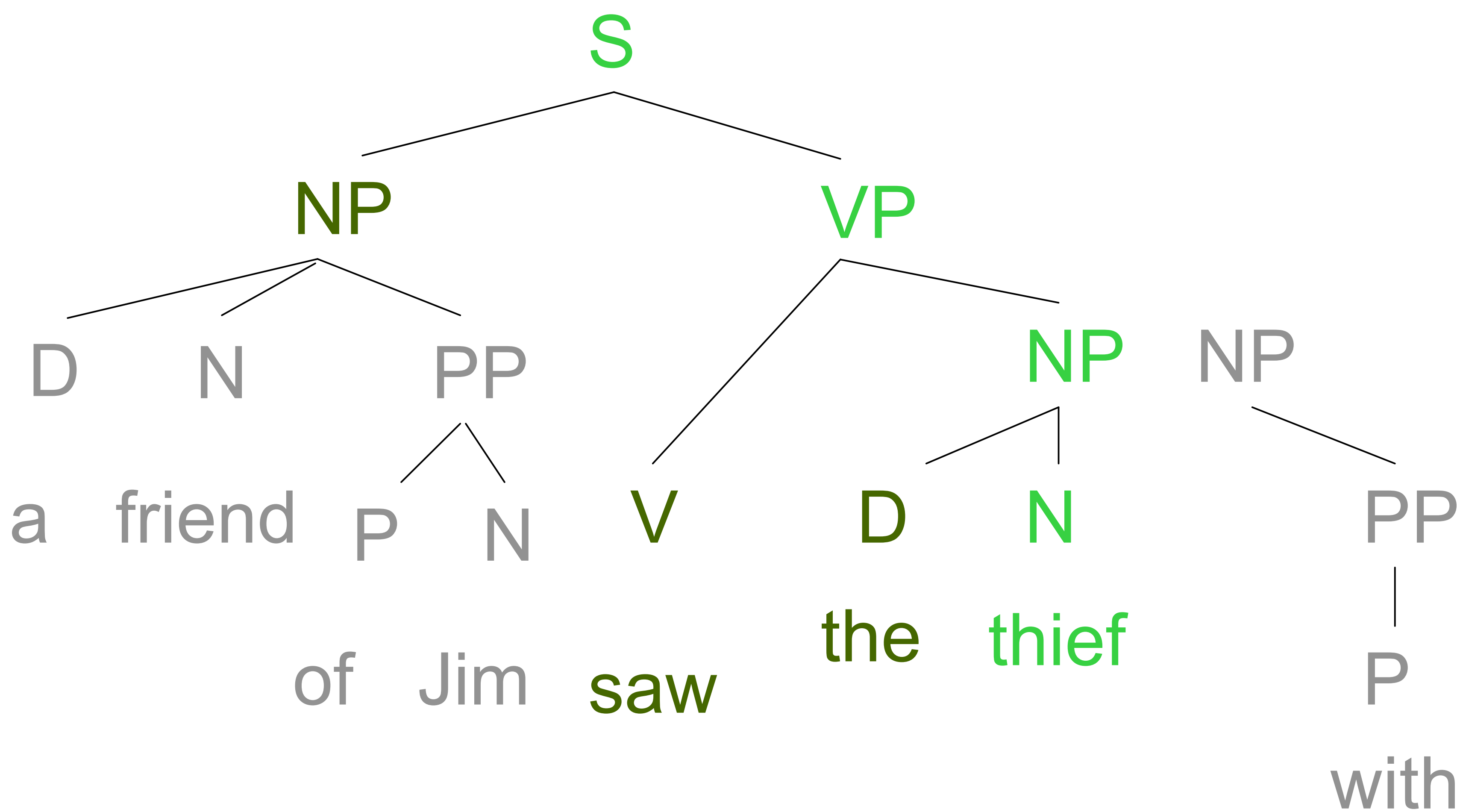
Reduced incremental trees

Right frontier



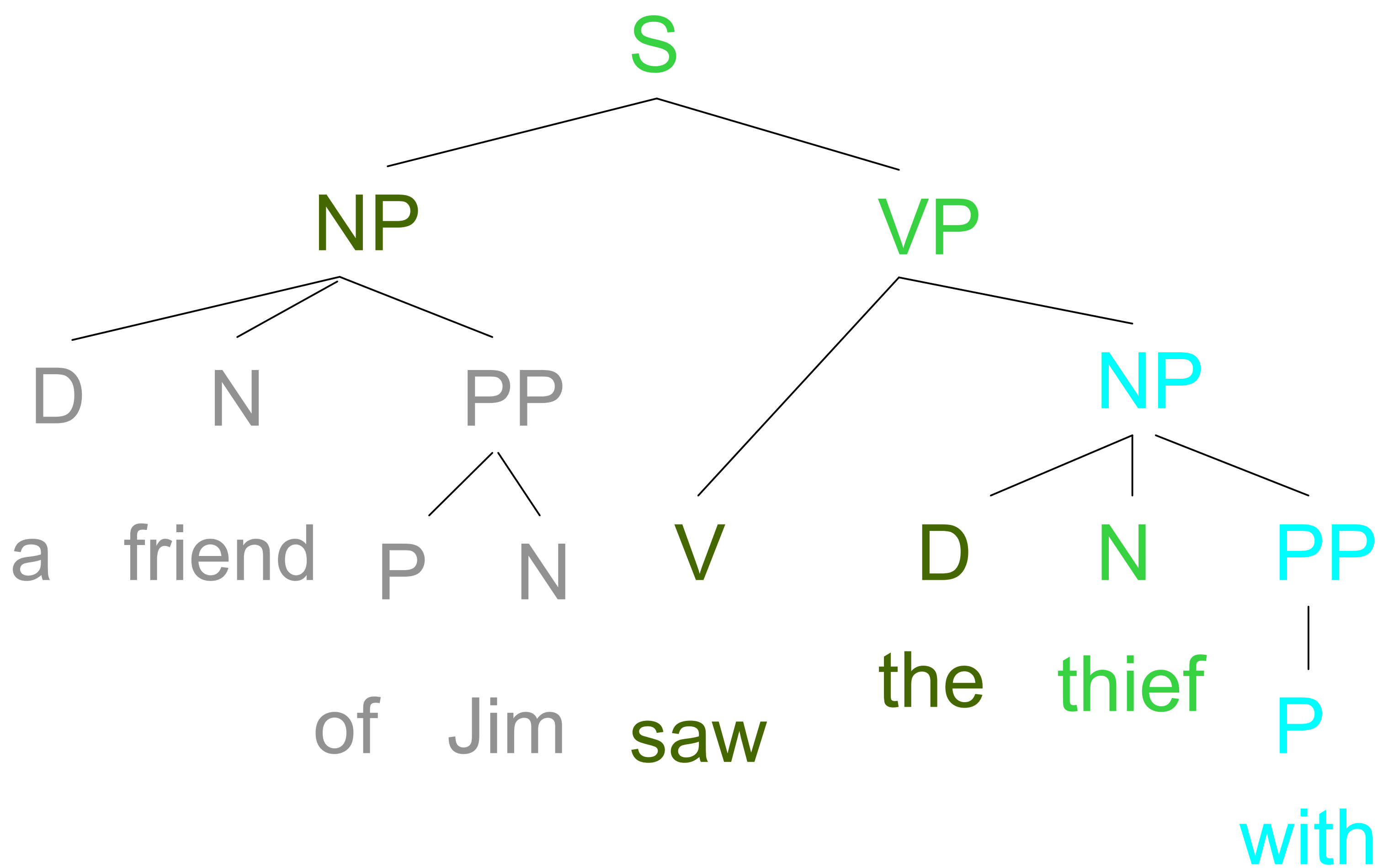
Reduced incremental trees

Right frontier + c-commanding nodes

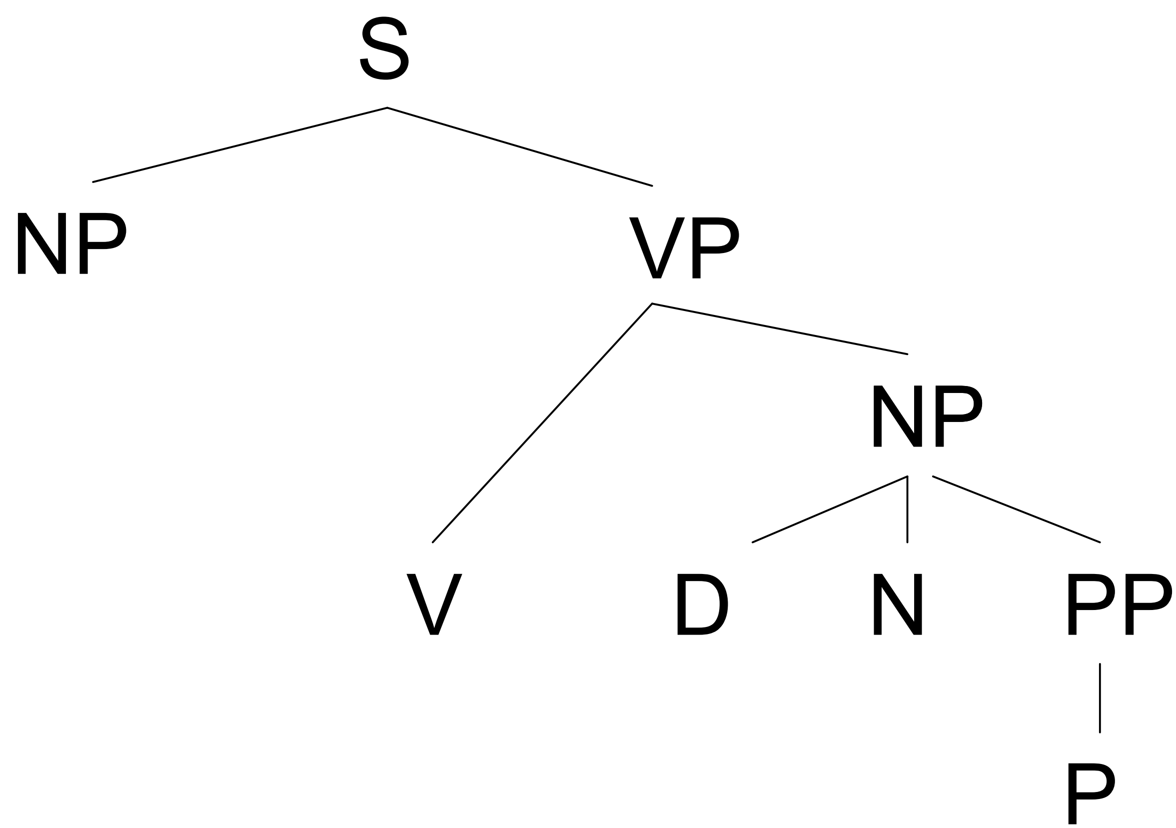


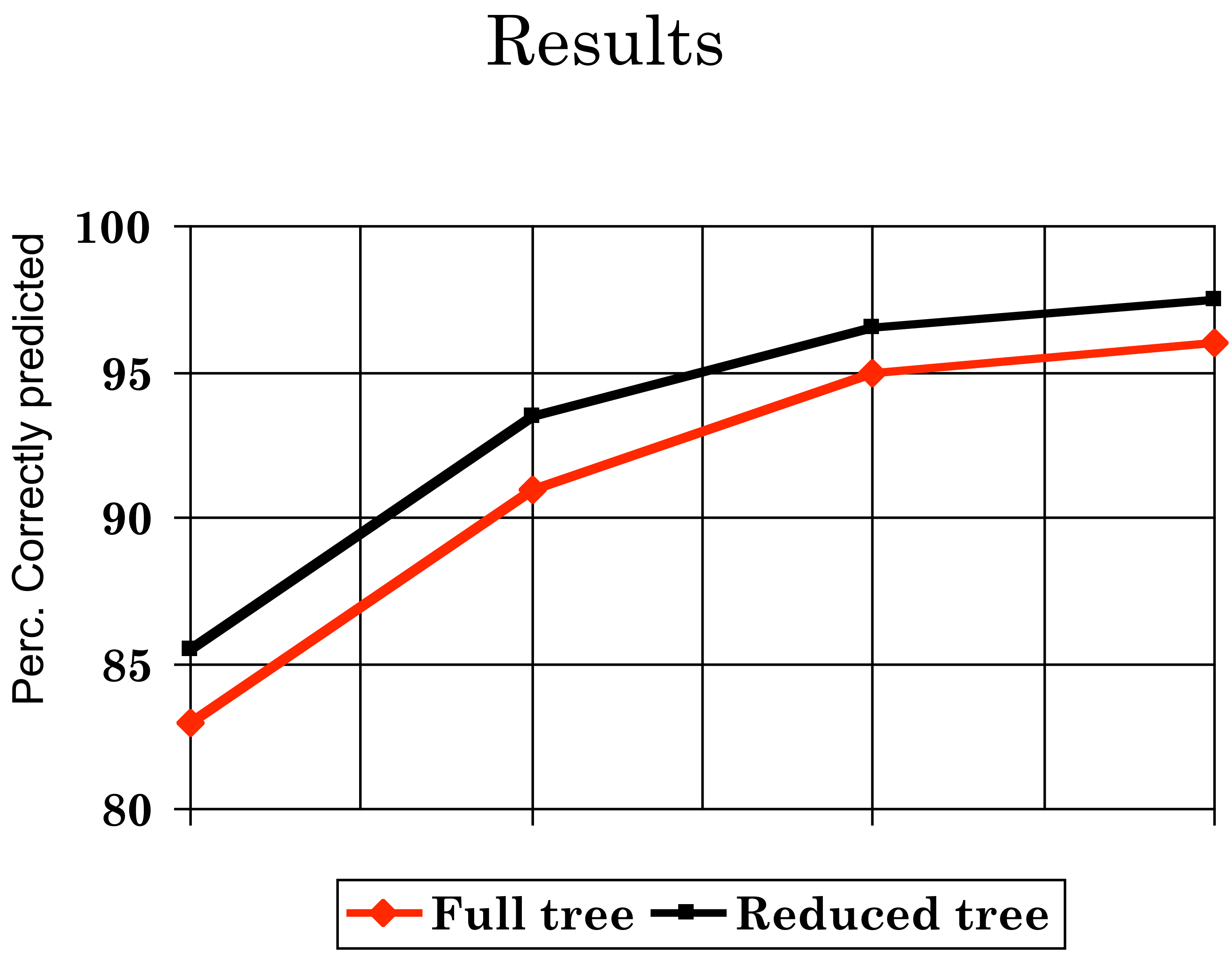
Reduced incremental trees

Right frontier + c-commanding nodes + connection path



Reduced incremental trees





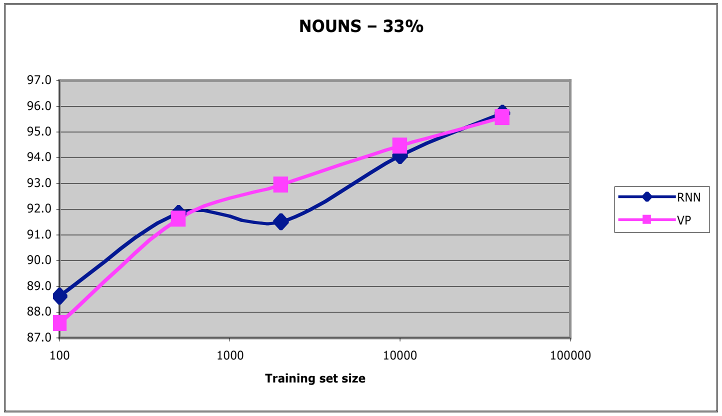
Results
100
95
90
85
80
Perc. Correctly predicted
Full tree
Reduced tree

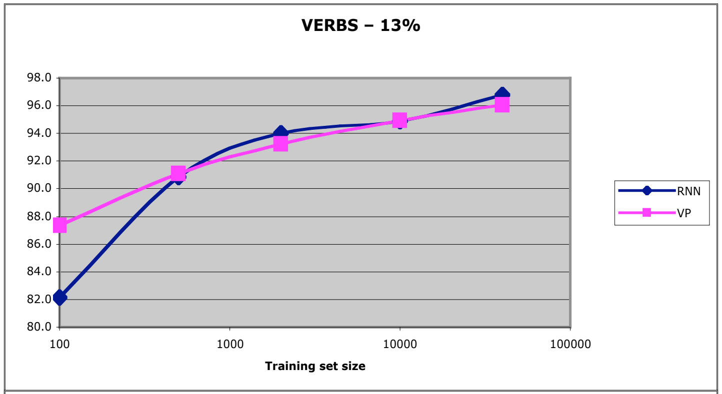
Data set partitioning (POS-tag based)

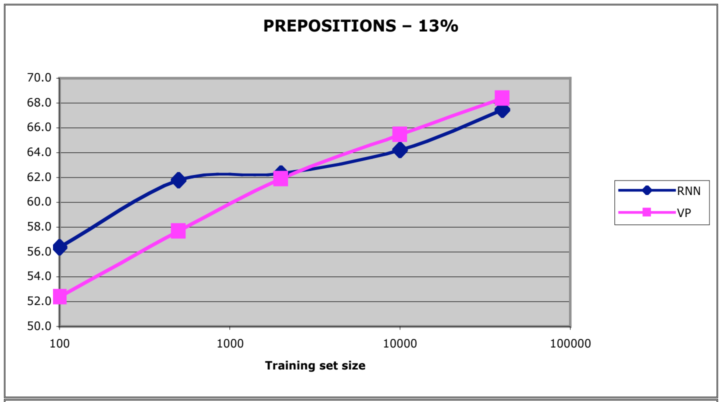
Comparing RNN and VP

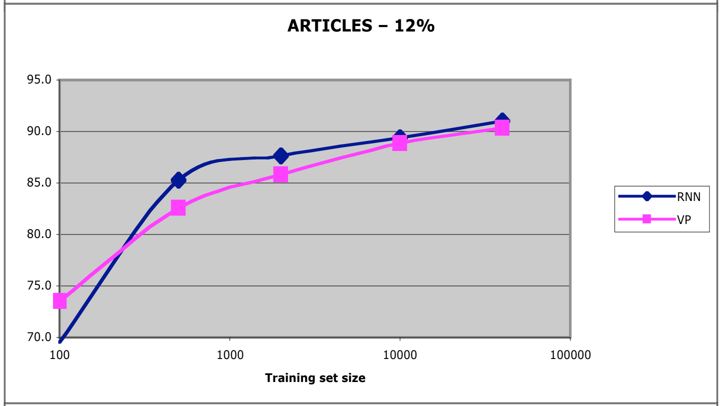
- Regularization parameter □=0.5 (best value based on preliminary trials using validation set)
- Modularization in 10 POS-tag categories
- Performance assessment at 100, 500, 2000, 10000, and 40000 training sentences
- Small datasets: CPU(VP) ~ k CPU(RNN)
- Larger datasets:
  - RNN learns in 1-2 epochs (~ 3 days 2GHz)
  - VP took over 2 months to complete 1 epoch

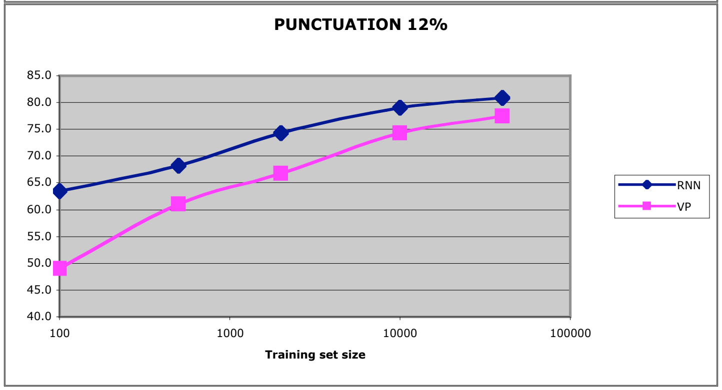
VP vs. RNN

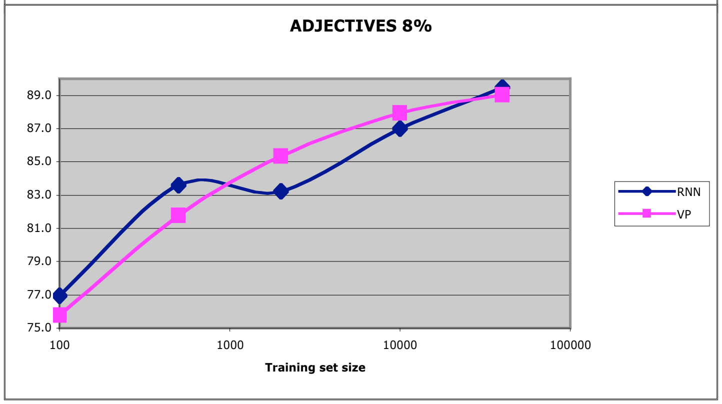

NOUNS – 33%
Training set size
RNN
VP


VERBS – 13%
Training set size
RNN
VP

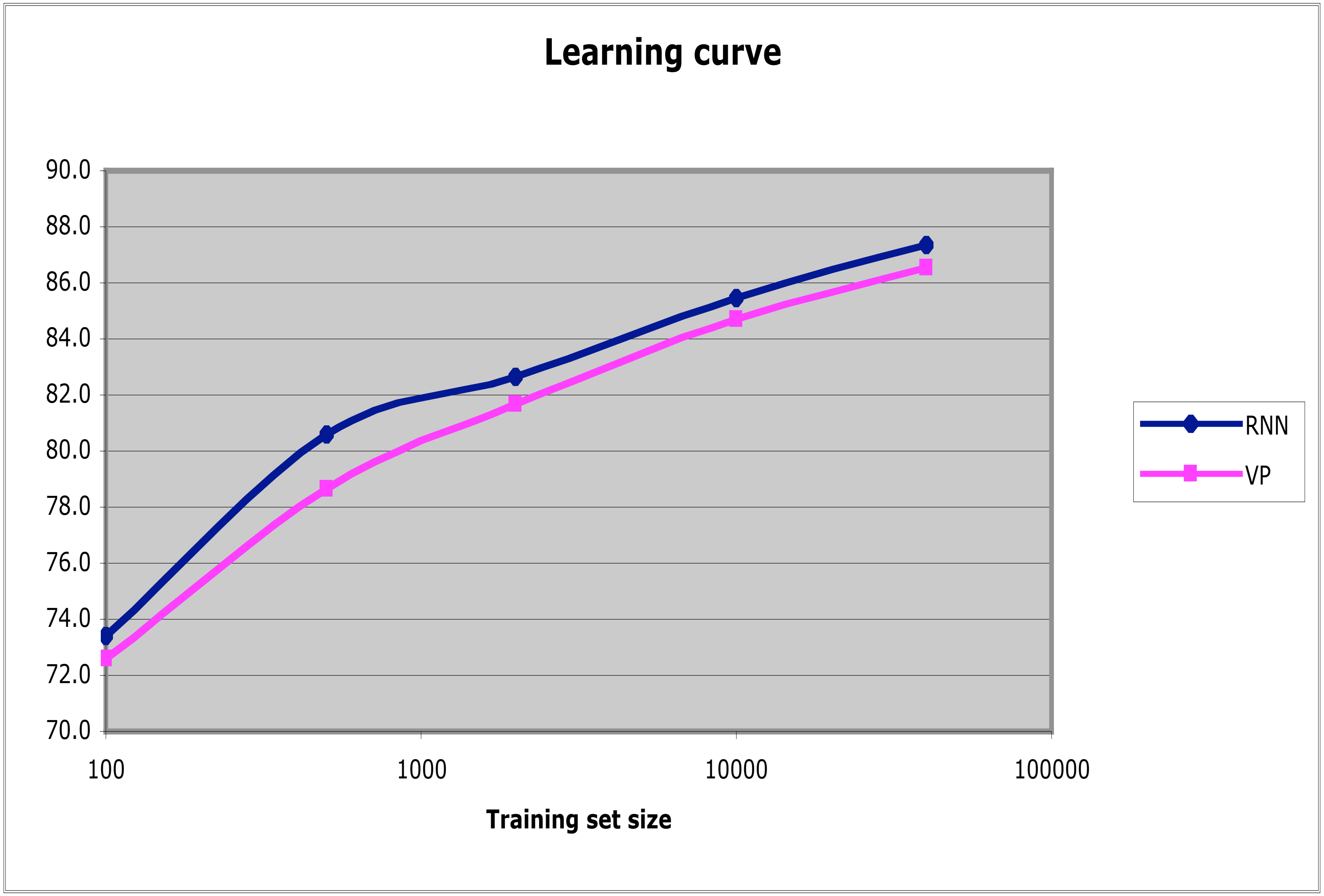

PREPOSITIONS – 13%
Training set size
RNN
VP


ARTICLES – 12%
Training set size
RNN
VP

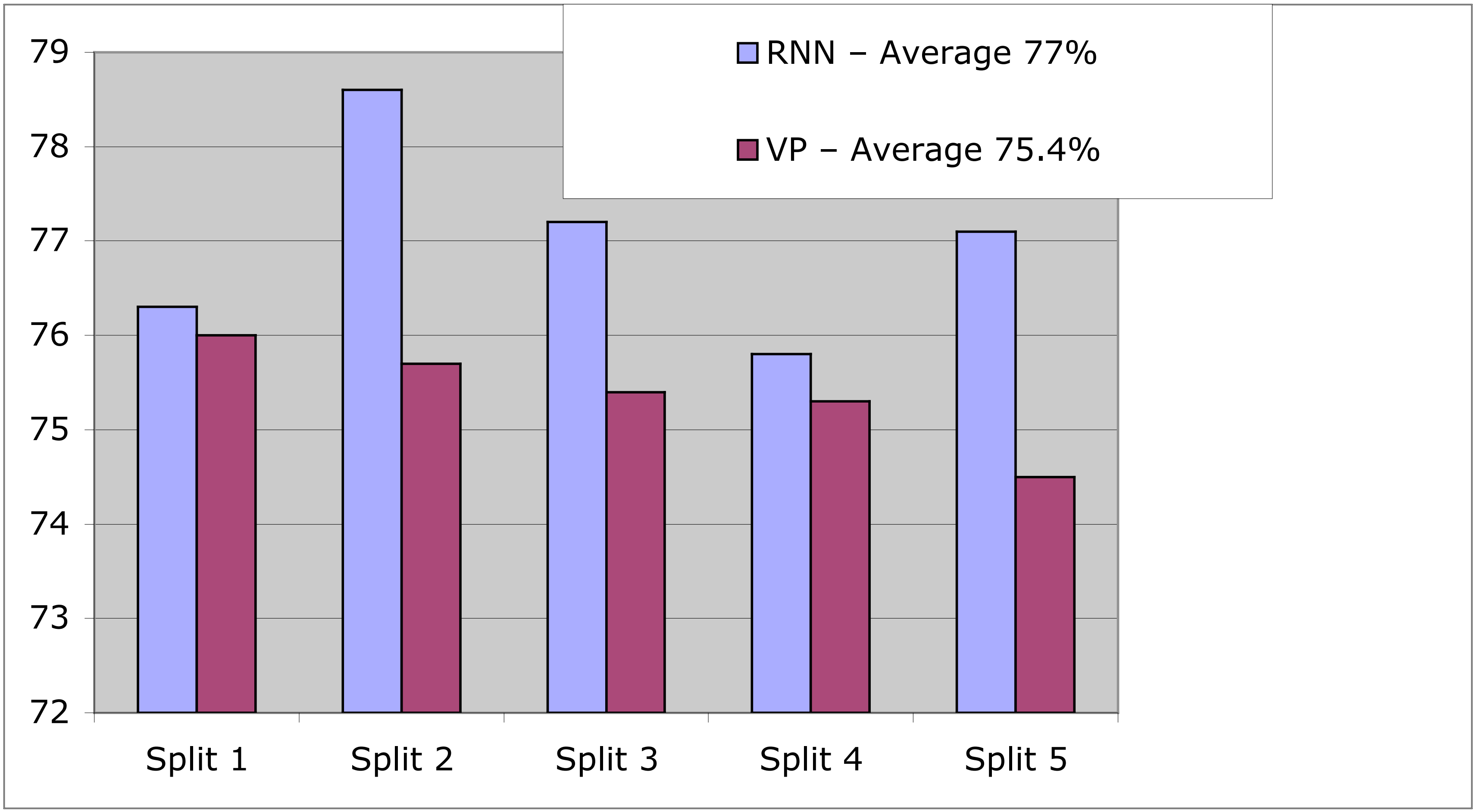

PUNCTUATION 12%
Training set size
RNN
VP


ADJECTIVES 8%
Training set size
RNN
VP

VP vs. RNN Modularization



5 independent splits
No modularization



Summary

- VP results perhaps to be strengthened but
  - 5 x 100 sentences takes ~ a week on a 2GHz CPU
  - VP does not scale up linearly with # examples
- However it appears that
  - RNN to be preferred, unless one has good knowledge to put into the design of the right kernel
- Ongoing work: Collins' relabeling task
  - Same problem setting (ranking on forests)
  - Less computation involved (1 forest for each sentence vs. 1 forest for each word)

Thanks:

Vincenzo Lombardo
Università di Torino

Patrick Sturt
University of Glasgow

Giovanni Soda
Università di Firenze