

# Efficient Parsing for Transducer Grammars



John DeNero, Mohit Bansal, Adam Pauls, and Dan Klein

# Overview

---

- Syntactic decoding can decompose into two phases: parsing and language model integration
- The parsing phase alone is time consuming for large tree transducer grammars
- Grammar transformations, optimizations, and coarse-to-fine techniques increase parsing speed
- The accelerated parsing pass improves translation

# Tree Transducer Grammars

---

No se olvide de subir un canto rodado en Colorado

**Synchronous Grammar**

**Output**

# Tree Transducer Grammars

---

No se olvide de subir un canto rodado en Colorado

## Synchronous Grammar

*NNP* → Colorado ; *Colorado*

**Output**

# Tree Transducer Grammars

---

No se olvide de subir un canto rodado en Colorado

## Synchronous Grammar

***NNP*** → Colorado ; *Colorado*

***NN*** → canto rodado ; *boulder*

**Output**

# Tree Transducer Grammars

---

No se olvide de subir un canto rodado en Colorado

## Synchronous Grammar

***NNP*** → Colorado ; *Colorado*

***NN*** → canto rodado ; *boulder*

***S*** → No se olvide de subir un ***NN*** en ***NNP*** ; *Don't forget to climb a ***NN*** in ***NNP****

## Output

# Tree Transducer Grammars

No se olvide de subir un can

## Synchronous G

*NNP* → Colorado ; *Colorado*

*NN* → canto rodado ; *boulder*

*S* → No se olvide de subir un *NN* en *NNP* ; *Don't forget to climb a **NN** in **NNP***

**Output**

# Tree Transducer Grammars

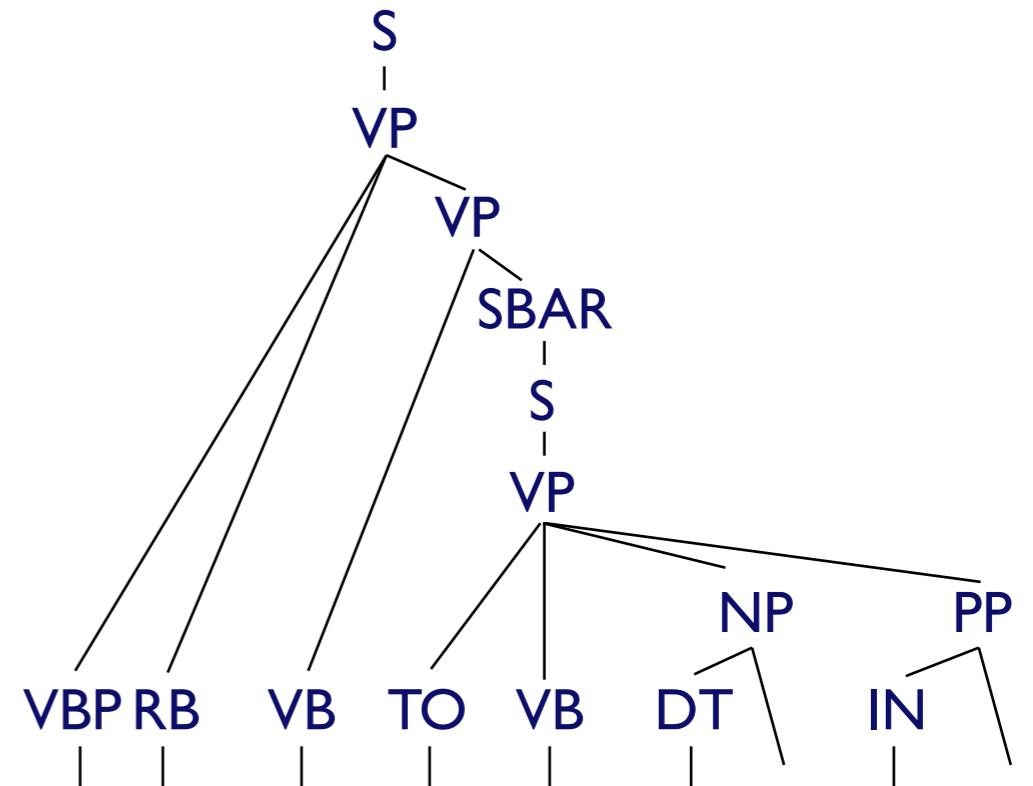
No se olvide de subir un cano

**Synchronous G**

**NNP** → Colorado ; *Colorado*

**NN** → canto rodado ; *boulder*

**S** → No se olvide de subir un **NN** en **NNP** ; *Don't forget to climb a **NN** in **NNP***



**Output**

# Tree Transducer Grammars

---

No se olvide de subir un canto rodado en Colorado

## Synchronous Grammar

***NNP*** → Colorado ; *Colorado*

***NN*** → canto rodado ; *boulder*

***S*** → No se olvide de subir un ***NN*** en ***NNP*** ; *Don't forget to climb a ***NN*** in ***NNP****

## Output

# Tree Transducer Grammars

No se olvide de subir un  $\overline{\text{canto rodado}}$  en Colorado

## Synchronous Grammar

$\mathbf{NNP} \rightarrow \text{Colorado} \quad ; \quad \text{Colorado}$

$\mathbf{NN} \rightarrow \text{canto rodado} \quad ; \quad \text{boulder}$

$\mathbf{S} \rightarrow \text{No se olvide de subir un } \mathbf{NN} \text{ en } \mathbf{NNP} \quad ; \quad \text{Don't forget to climb a } \mathbf{NN} \text{ in } \mathbf{NNP}$

## Output

$\overline{\text{NN}}$   
boulder

# Tree Transducer Grammars

No se olvide de subir un  $\overline{\text{canto rodado}}$  en  $\overline{\text{Colorado}}$

## Synchronous Grammar

$\mathbf{NNP} \rightarrow \text{Colorado} \quad ; \quad \text{Colorado}$

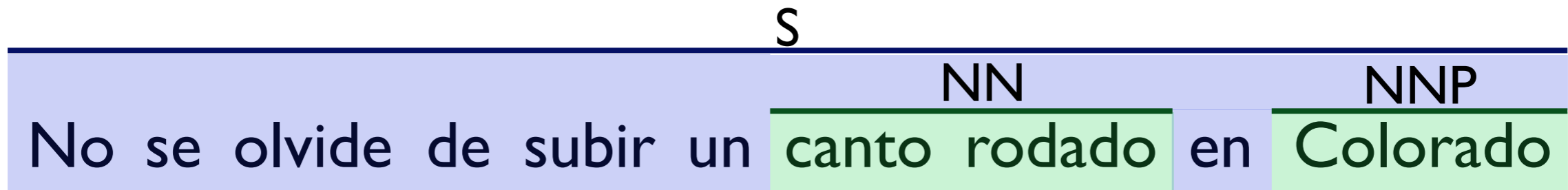
$\mathbf{NN} \rightarrow \text{canto rodado} \quad ; \quad \text{boulder}$

$\mathbf{S} \rightarrow \text{No se olvide de subir un } \mathbf{NN} \text{ en } \mathbf{NNP} \quad ; \quad \text{Don't forget to climb a } \mathbf{NN} \text{ in } \mathbf{NNP}$

## Output

$\overline{\text{boulder}} \quad \overline{\text{Colorado}}$

# Tree Transducer Grammars



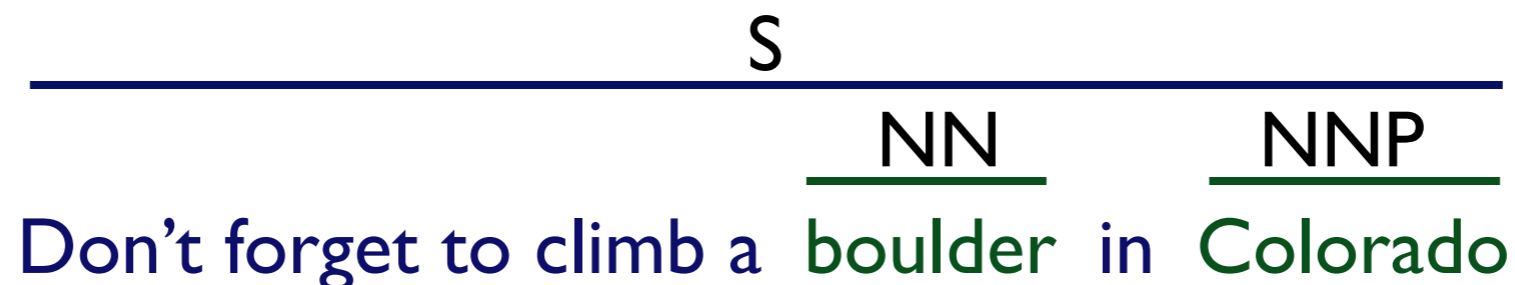
## Synchronous Grammar

**NNP** → Colorado ; *Colorado*

**NN** → canto rodado ; *boulder*

**S** → No se olvide de subir un **NN** en **NNP** ; *Don't forget to climb a NN in NNP*

## Output



# Multi-Pass Syntactic Decoding

Parse input sentence  
with source-side  
grammar projection

Rerank derivations  
rooted at each parse state  
using a language model

**Cube growing** [Huang and Chiang '07]:

*Lazy forest reranking algorithm*

**Two-pass SCFG decoding** [Venugopal et al '07]:

*Local search over derivations in a parse forest*

**Coarse-to-fine LMs** [Zhang et al '08, Petrov et al '08]:

*Multi-pass bottom-up LM integration over forests*

# The Size of Tree Transducer Grammars

---

Extracted a transducer grammar from a 220 million word bitext

Relativized the grammar to each test sentence

Kept all rules with at most 6 non-terminals

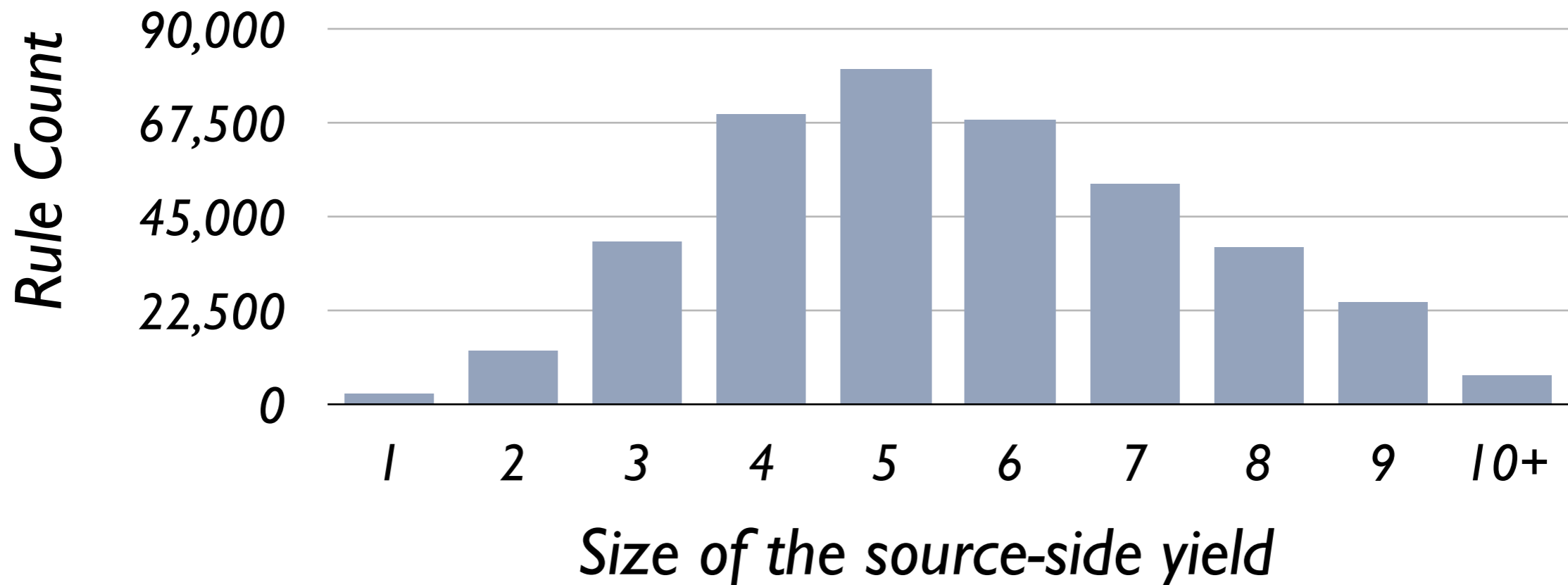
# The Size of Tree Transducer Grammars

Extracted a transducer grammar from a 220 million word bitext

Relativized the grammar to each test sentence

Kept all rules with at most 6 non-terminals

## Rules matching an example 40-word sentence



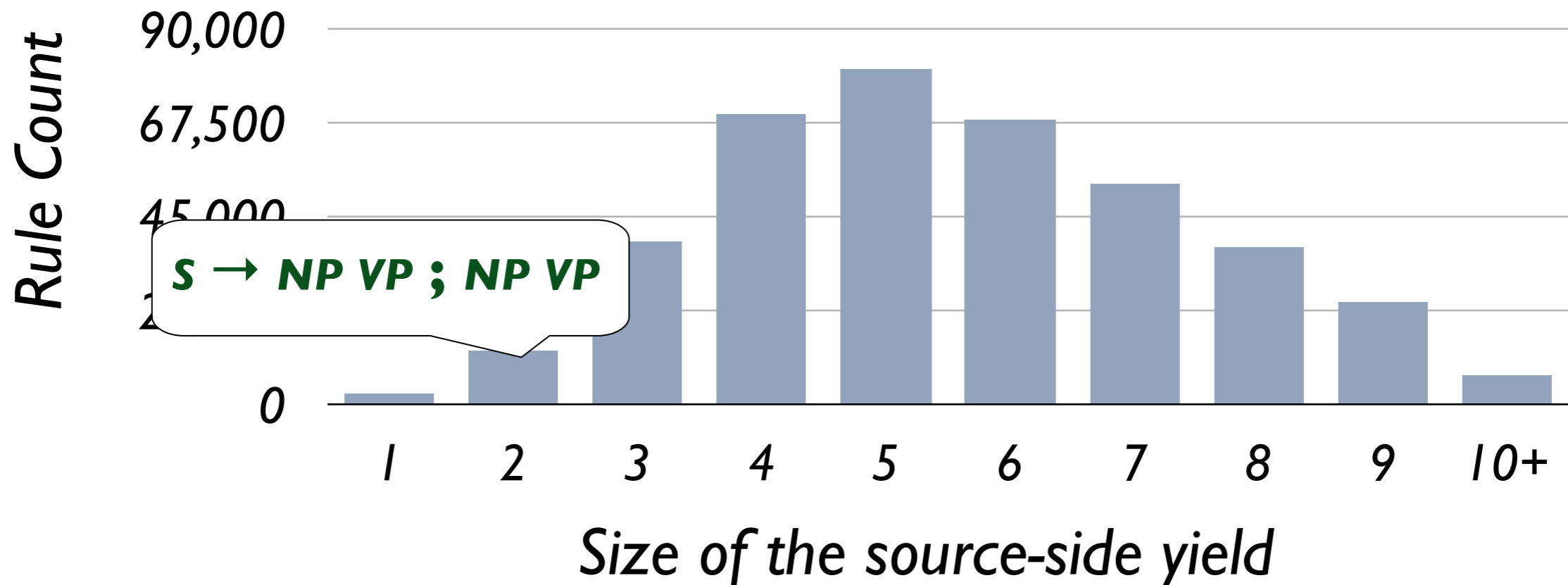
# The Size of Tree Transducer Grammars

Extracted a transducer grammar from a 220 million word bitext

Relativized the grammar to each test sentence

Kept all rules with at most 6 non-terminals

## Rules matching an example 40-word sentence



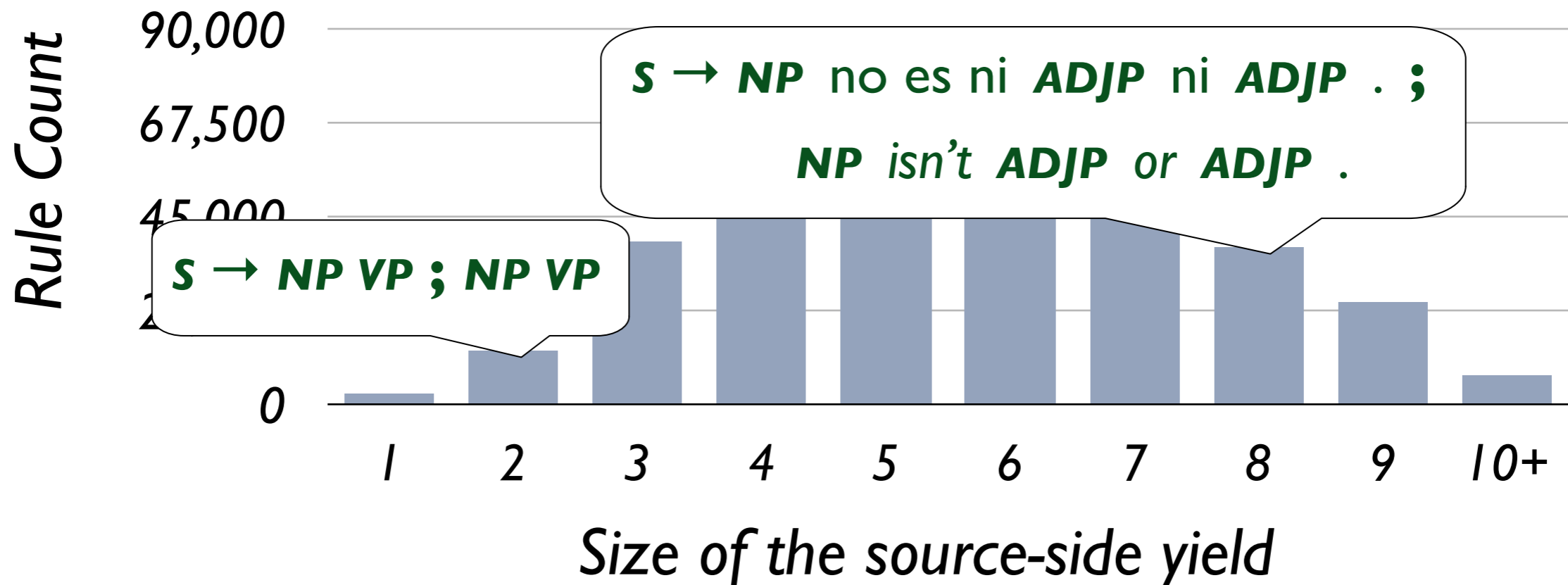
# The Size of Tree Transducer Grammars

Extracted a transducer grammar from a 220 million word bitext

Relativized the grammar to each test sentence

Kept all rules with at most 6 non-terminals

## Rules matching an example 40-word sentence





# CKY-style Bottom-up Parsing

---

For each  
span length:

# CKY-style Bottom-up Parsing

---

For each  
span length:

For each  
span  $[i,j]$ :

# CKY-style Bottom-up Parsing

---

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

Binary rule:  $X \rightarrow Y Z$

# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

Binary rule:  $X \rightarrow Y Z$

Split points:  $i < k < j$

Operations:  $O(j - i)$

Time scales with: Grammar constant

# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

$i$  No se olvide de subir un canto rodado en Colorado  $j$

# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

**S** → No se **VB** de subir un **NN** en **NNP**

$_i$  No se olvide de subir un canto rodado en Colorado  $_j$

# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

**S** → **No se VB de subir un NN en NNP**

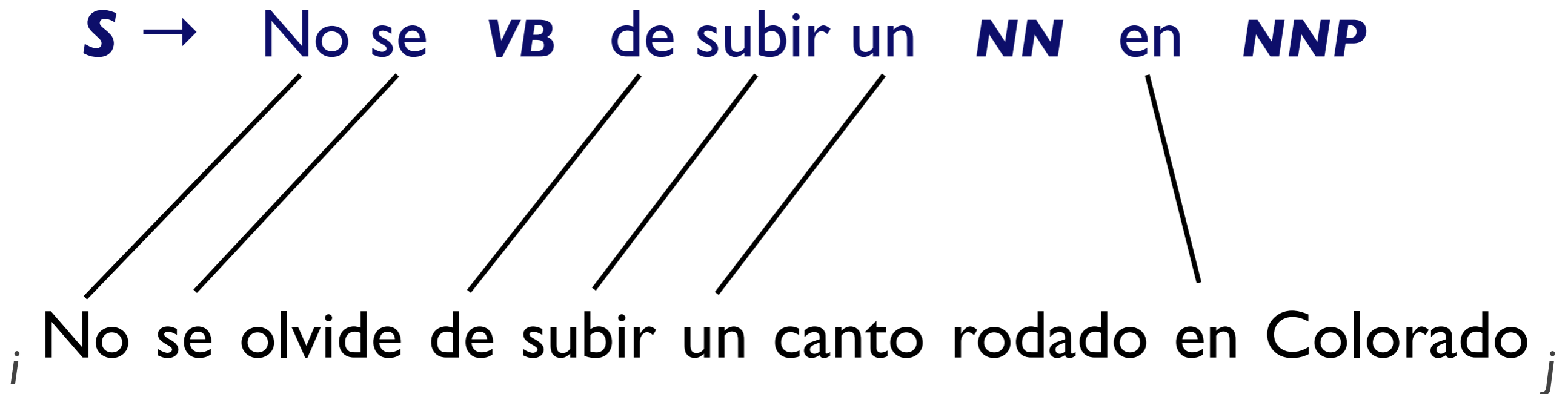
$i$  No se olvide de subir un canto rodado en Colorado  $j$

# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

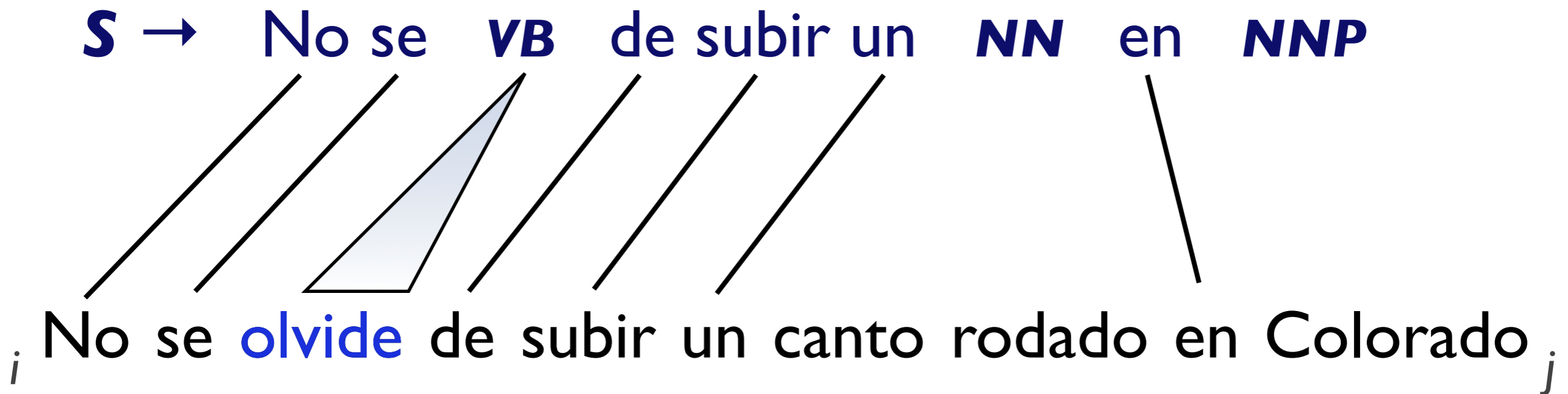


# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

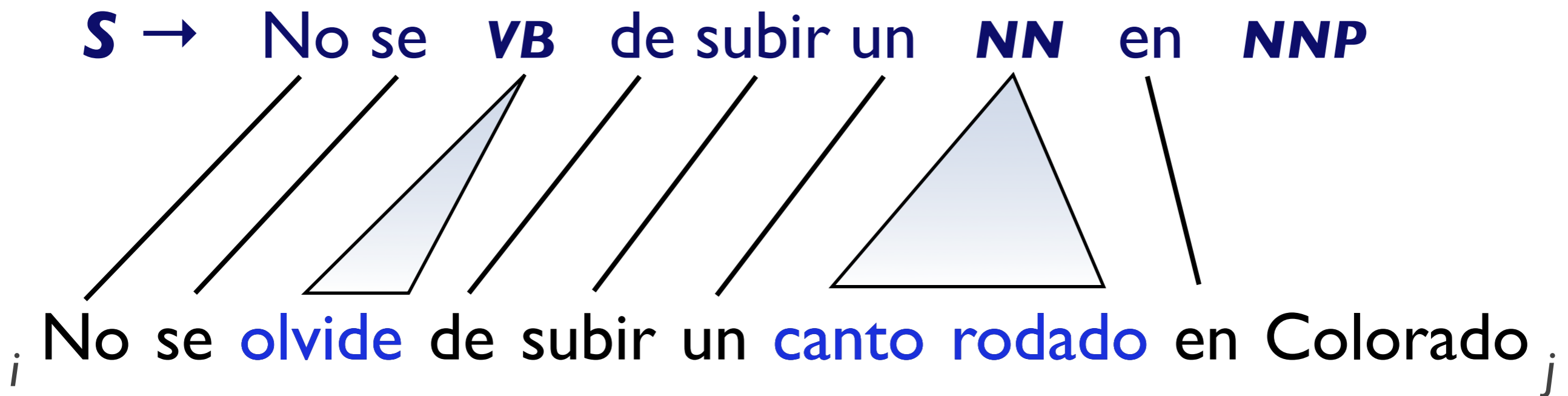


# CKY-style Bottom-up Parsing

For each span length:

For each span  $[i,j]$ :

Apply all grammar rules to  $[i,j]$

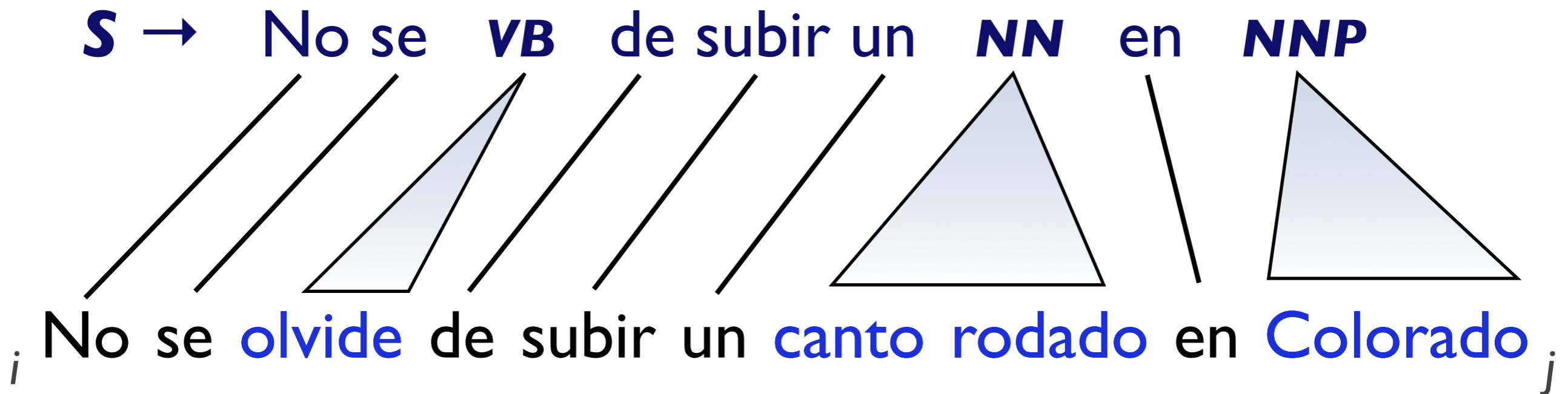


# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

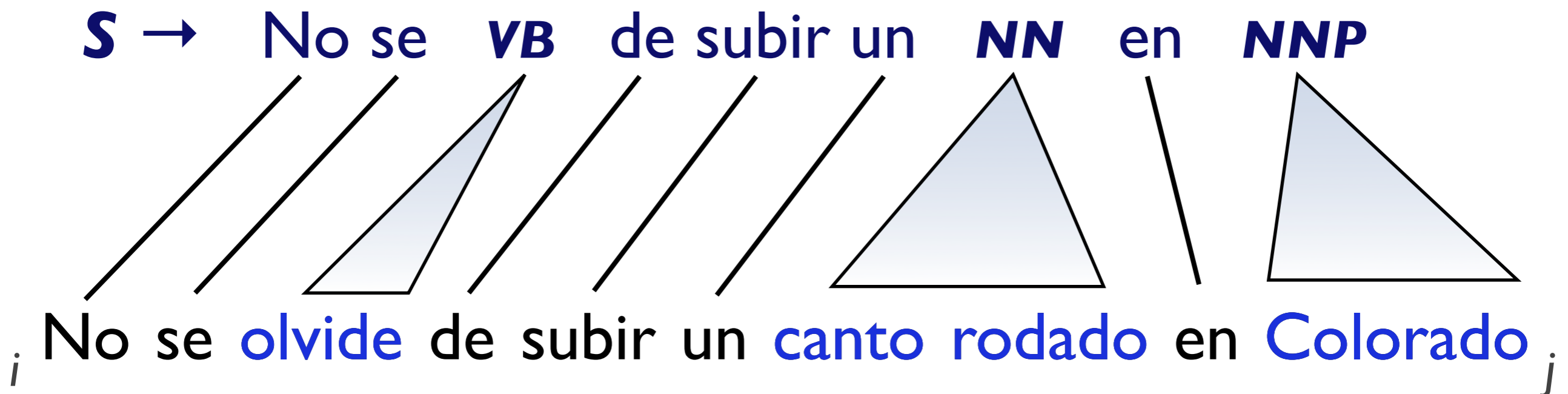


# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$



*Many untransformed lexical rules can be applied in linear time*

# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

$i$  No se olvide de subir un canto rodado en Colorado  $j$

# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

**$S \rightarrow$  No se *VP NP PP***

$_i$  No se olvide de subir un canto rodado en Colorado  $_j$

# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

**S** → **No se VP NP PP**

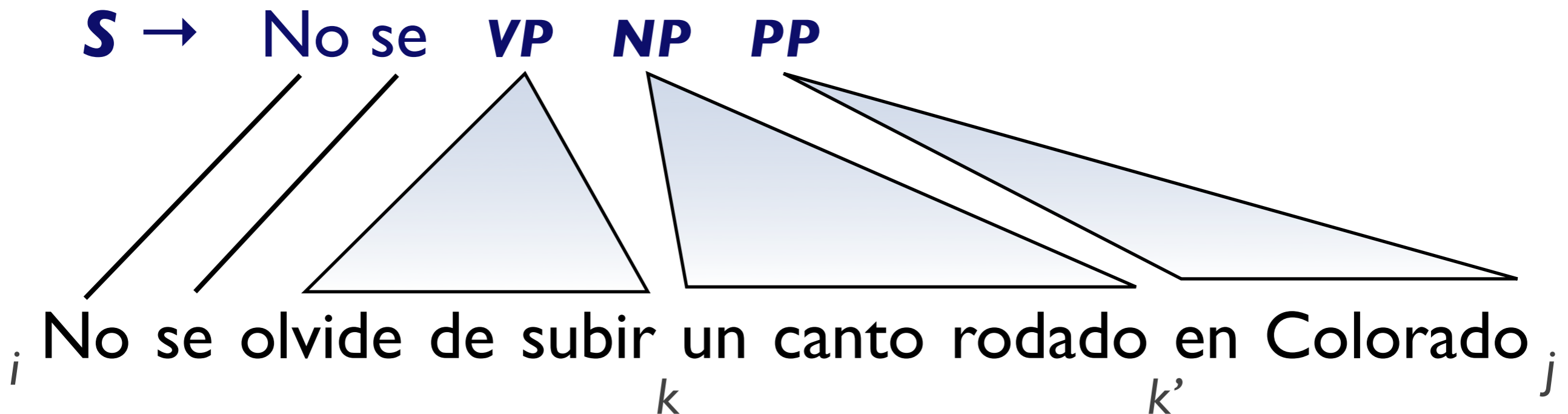
$_i$  No se olvide de subir un canto rodado en Colorado  $_j$

# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$

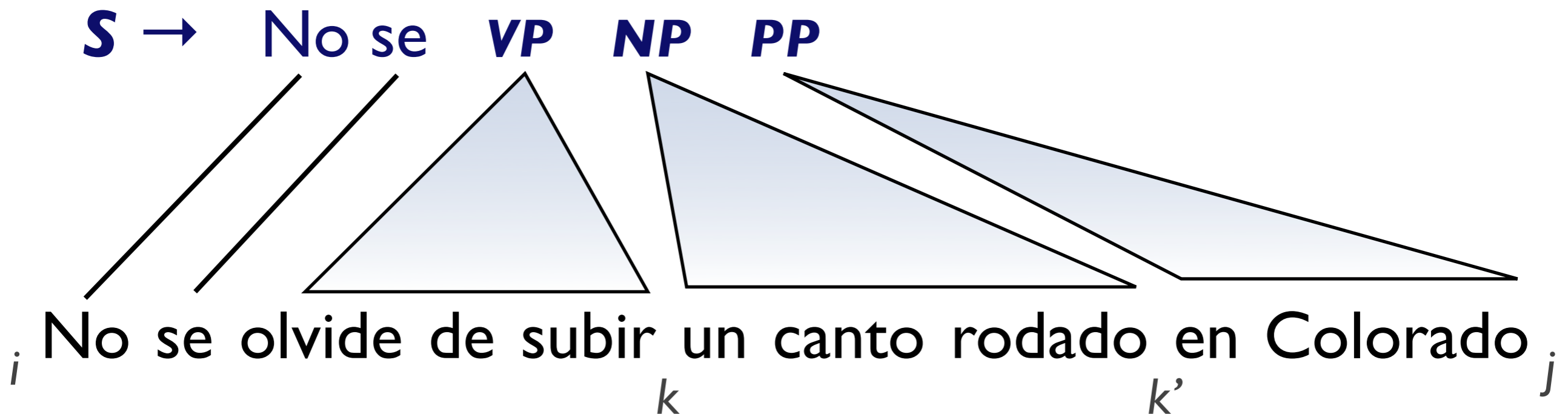


# CKY-style Bottom-up Parsing

For each  
span length:

For each  
span  $[i,j]$ :

Apply all grammar  
rules to  $[i,j]$



**Problem:** Applying adjacent non-terminals is slow

# Eliminating Non-terminal Sequences

---

## Lexical Normal Form (LNF)

- (a) lexical rules have at most one adjacent non-terminal*
- (b) all unlexicalized rules are binary.*

# Eliminating Non-terminal Sequences

## Lexical Normal Form (LNF)

- (a) lexical rules have at most one adjacent non-terminal*
- (b) all unlexicalized rules are binary.*

Original rule:                    **S** → No se **VB** **VB** un **NN** **PP**

# Eliminating Non-terminal Sequences

## Lexical Normal Form (LNF)

- (a) *lexical rules have at most one adjacent non-terminal*
- (b) *all unlexicalized rules are binary.*

Original rule:  $S \rightarrow \text{No se } VB \ VB \ \text{un } NN \ PP$

Transformed rules:  $S \rightarrow \text{No se } VB \sim VB \ \text{un } NN \sim PP$

$VB \sim VB \rightarrow VB \ VB$

$NN \sim PP \rightarrow NN \ PP$

# Eliminating Non-terminal Sequences

## Lexical Normal Form (LNF)

- (a) *lexical rules have at most one adjacent non-terminal*
- (b) *all unlexicalized rules are binary.*

Original rule:  $S \rightarrow \text{No se } VB \ VB \ \text{un } NN \ PP$

Transformed rules:  $S \rightarrow \text{No se } VB \sim VB \ \text{un } NN \sim PP$

$VB \sim VB \rightarrow VB \ VB$

$NN \sim PP \rightarrow NN \ PP$

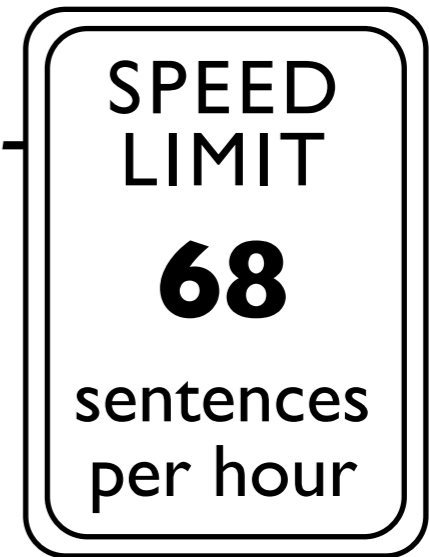
Parsing stages:

- Lexical rules are applied by matching
- Unlexicalized rules are applied by iterating over split points

# Eliminating Non-terminal Sequences

## Lexical Normal Form (LNF)

- (a) lexical rules have at most one adjacent non-terminal  
 (b) all unlexicalized rules are binary.



Original rule:  $S \rightarrow \text{No se VB VB un}$

Transformed rules:  $S \rightarrow \text{No se VB} \sim \text{VB un NN} \sim \text{PP}$

$\text{VB} \sim \text{VB} \rightarrow \text{VB VB}$

$\text{NN} \sim \text{PP} \rightarrow \text{NN PP}$

Parsing stages:

- Lexical rules are applied by matching
- Unlexicalized rules are applied by iterating over split points

# Eliminating Non-terminal Sequences

## Lexical Normal Form (LNF)

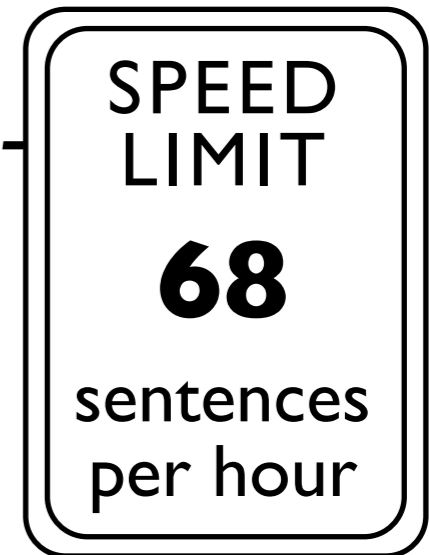
- (a) lexical rules have at most one adjacent non-terminal  
 (b) all unlexicalized rules are binary.

Original rule:  $S \rightarrow \text{No se VB VB un}$

Transformed rules:  $S \rightarrow \text{No se VB} \sim \text{VB un NN}$

$\text{VB} \sim \text{VB} \rightarrow \text{VB VB}$

$\text{NN} \sim \text{PP} \rightarrow \text{NN PP}$



Parsing stages:

- Lexical rules are applied by matching
- Unlexicalized rules are applied by iterating over split points

# Bounding Split Points

---

**Problem:** *Applying binary rules performs wasted work*

$VP \sim VP \rightarrow VP \quad VP$

$_i$  No se olvide de subir un canto rodado en Colorado  $_j$

# Bounding Split Points

**Problem:** *Applying binary rules performs wasted work*

$VP \sim VP \rightarrow VP \quad VP$

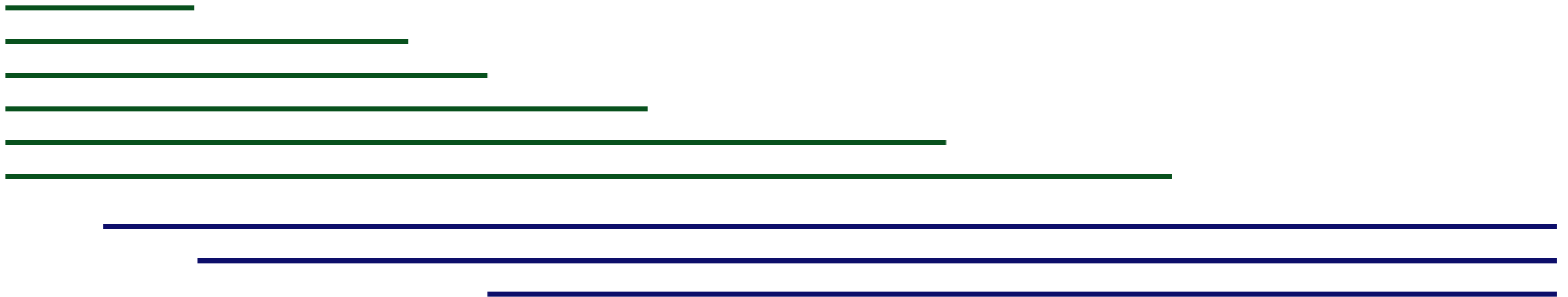


$i$  No se olvide de subir un canto rodado en Colorado  $j$

# Bounding Split Points

**Problem:** *Applying binary rules performs wasted work*

$VP \sim VP \rightarrow VP \quad VP$



$i$  No se olvide de subir un canto rodado en Colorado  $j$

# Bounding Split Points

**Problem:** Applying binary rules performs wasted work

$VP \sim VP \rightarrow VP \quad VP$

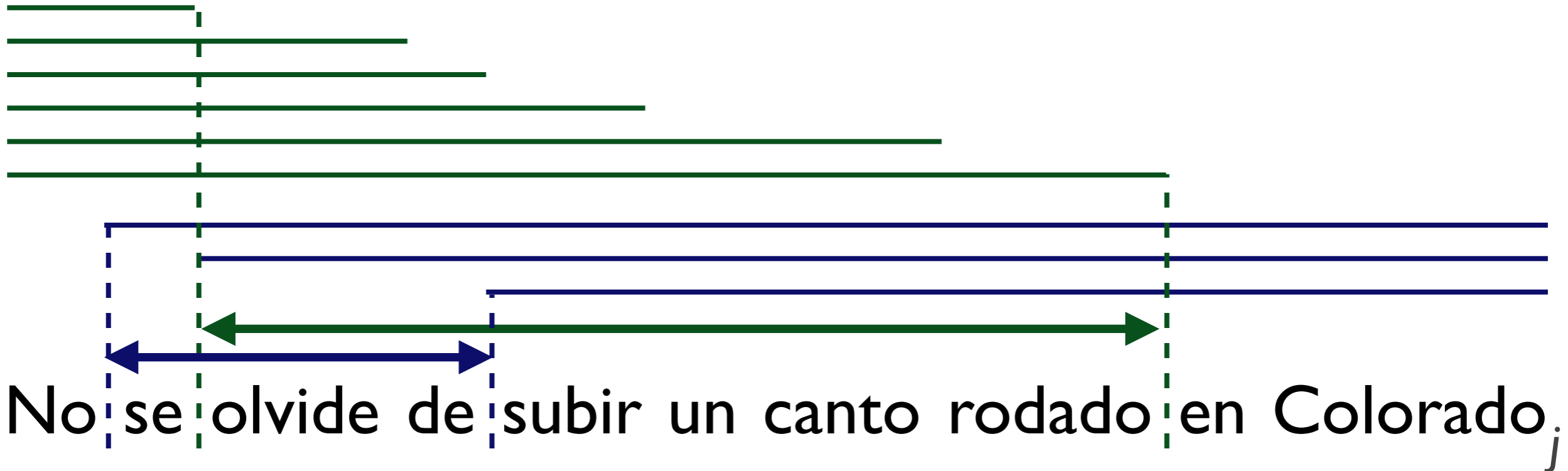


$_i$  No se olvide de subir un canto rodado en Colorado  $_j$

# Bounding Split Points

**Problem:** Applying binary rules performs wasted work

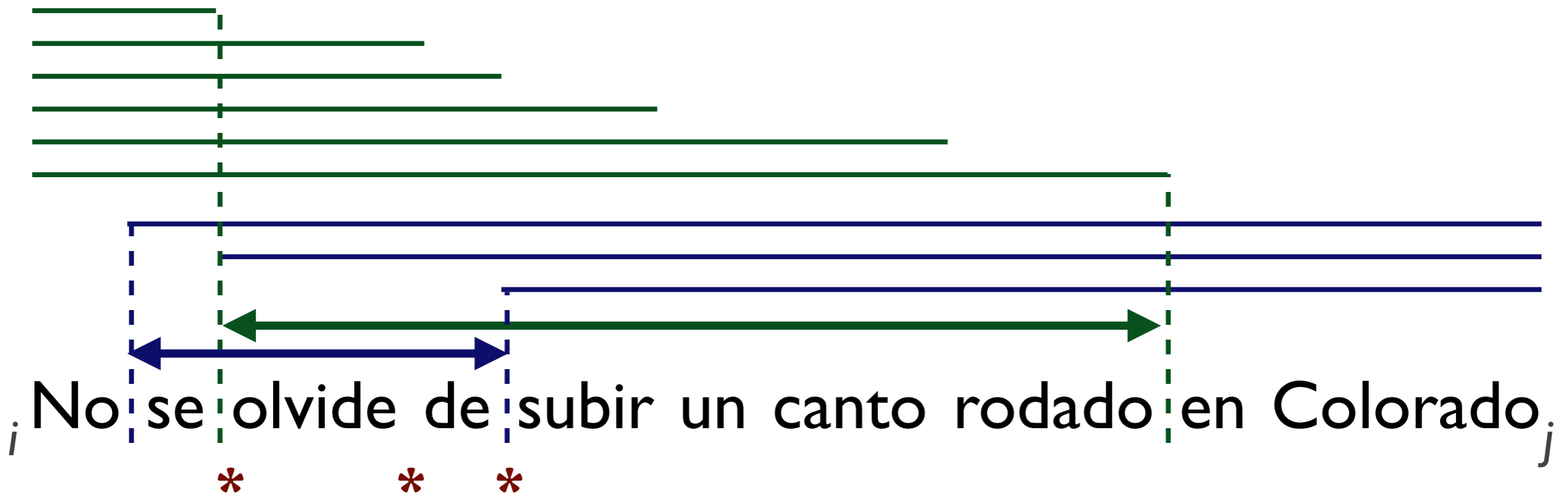
$VP \sim VP \rightarrow VP \quad VP$



# Bounding Split Points

**Problem:** Applying binary rules performs wasted work

$VP \sim VP \rightarrow VP \quad VP$

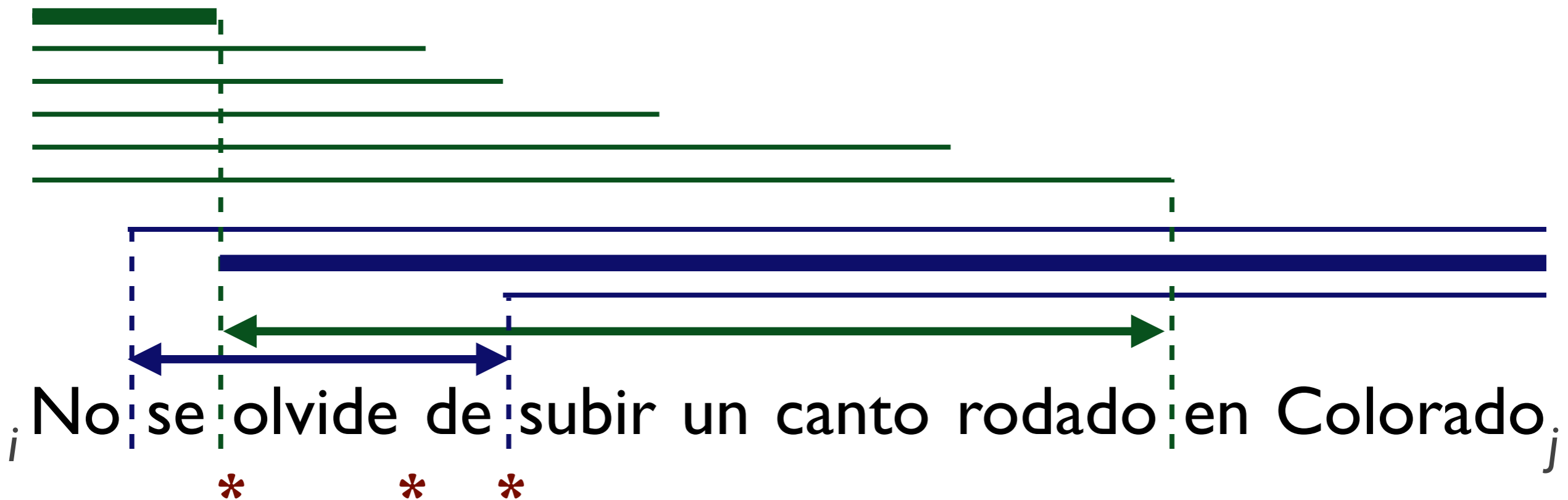


*Only consider split points  $k$  that might result in a valid parse*

# Bounding Split Points

**Problem:** Applying binary rules performs wasted work

$$VP \sim VP \rightarrow VP \quad VP$$

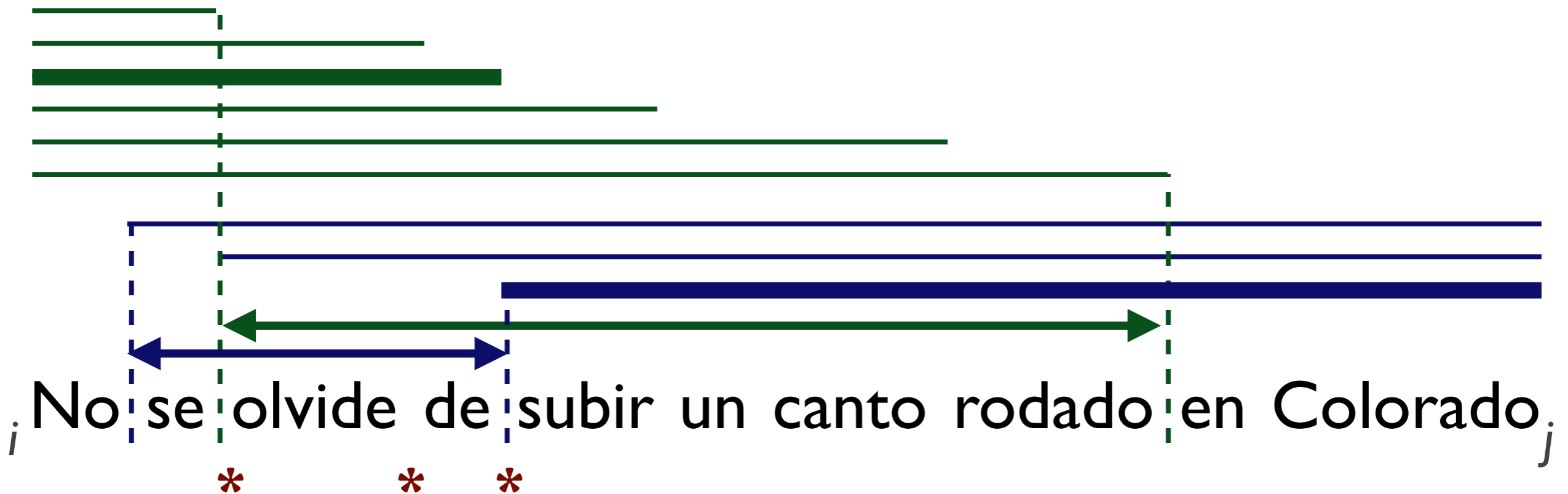


*Only consider split points  $k$  that might result in a valid parse*

# Bounding Split Points

**Problem:** Applying binary rules performs wasted work

$VP \sim VP \rightarrow VP \quad VP$

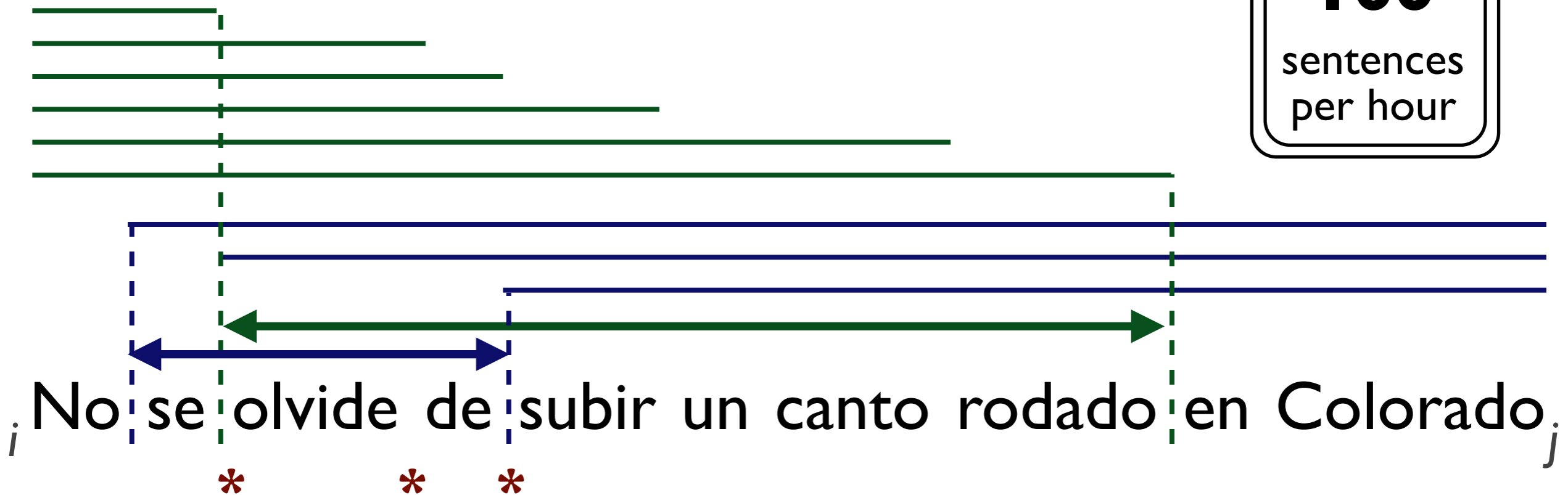
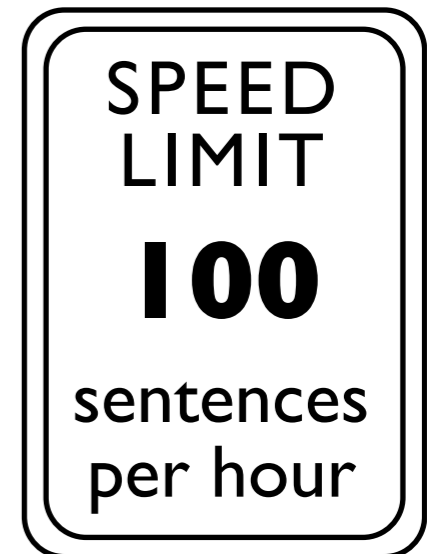


*Only consider split points  $k$  that might result in a valid parse*

# Bounding Split Points

**Problem:** Applying binary rules performs wasted work

$VP \sim VP \rightarrow VP \quad VP$



*Only consider split points  $k$  that might result in a valid parse*

# Speeding up Lexical Rule Application

---

**Problem:** *Lexical rules can apply to many spans*

$_i$  No se olvide de subir un canto rodado en Colorado  $_j$

# Speeding up Lexical Rule Application

---

**Problem:** *Lexical rules can apply to many spans*

**S** → No se olvide de subir **NP**

$_i$  No se olvide de subir un canto rodado en Colorado  $_j$

# Speeding up Lexical Rule Application

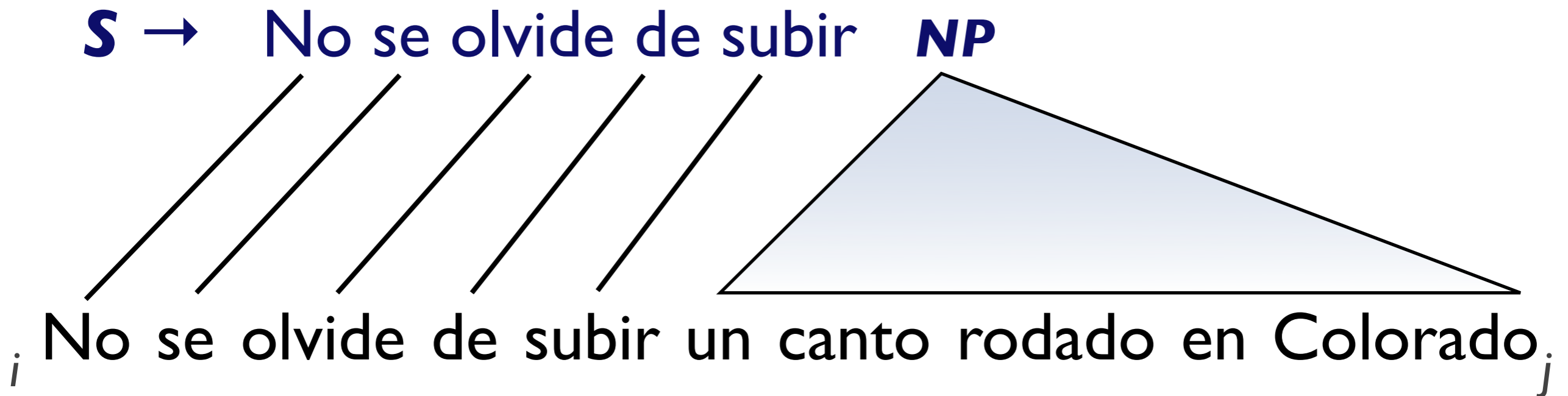
**Problem:** *Lexical rules can apply to many spans*

**S** → No se olvide de subir **NP**

$_i$  No se olvide de subir un canto rodado en Colorado  $_j$

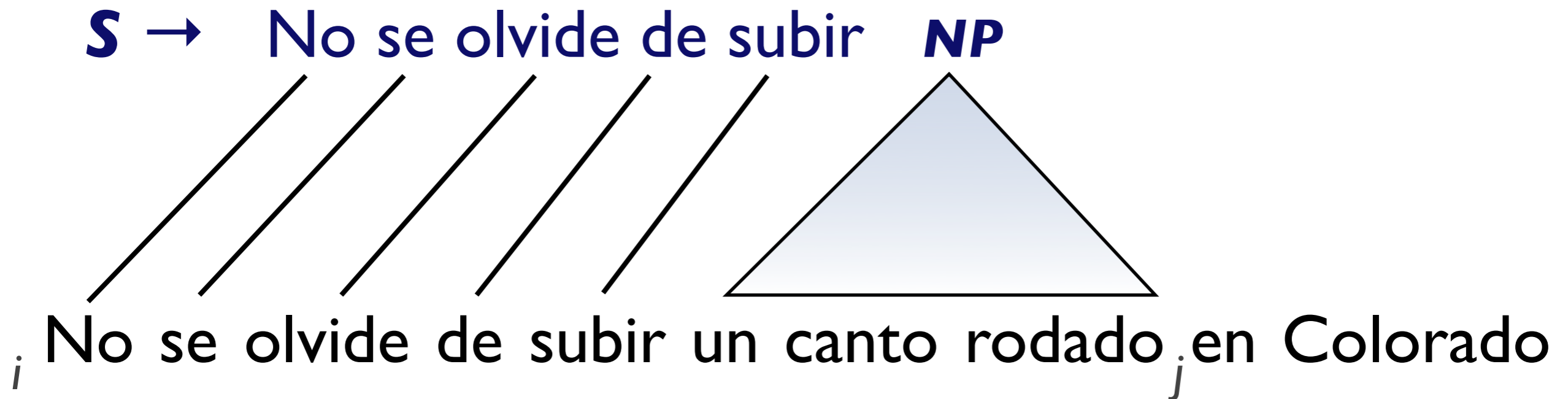
# Speeding up Lexical Rule Application

**Problem:** *Lexical rules can apply to many spans*



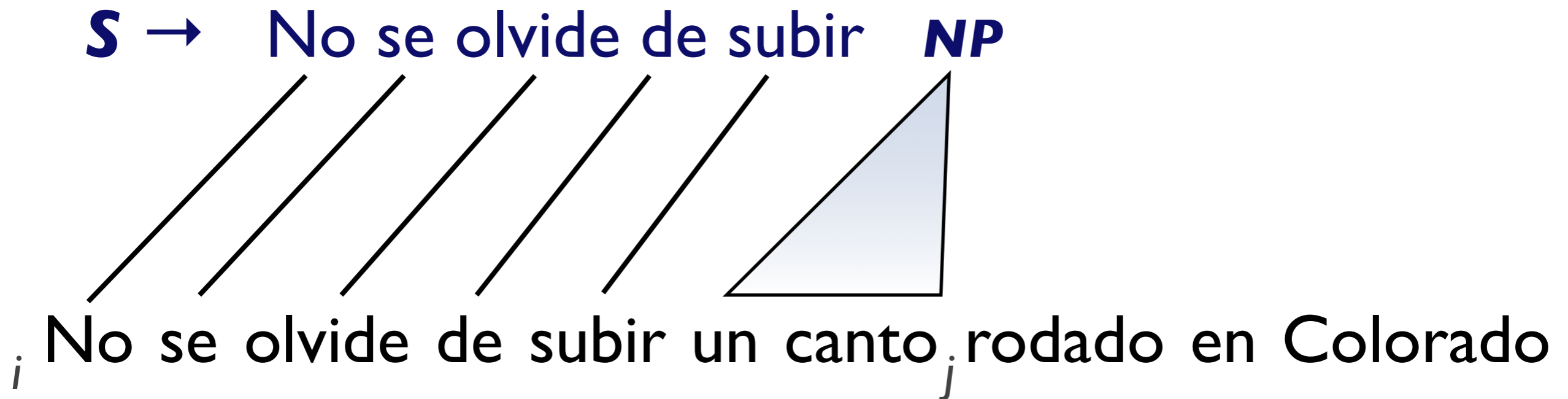
# Speeding up Lexical Rule Application

**Problem:** *Lexical rules can apply to many spans*



# Speeding up Lexical Rule Application

**Problem:** *Lexical rules can apply to many spans*



# Speeding up Lexical Rule Application

---

## **Anchored Lexical Normal Form (ALNF)**

- (a) lexical rules match a constant number of spans*
- (b) all unlexicalized rules are binary.*

# Speeding up Lexical Rule Application

## Anchored Lexical Normal Form (ALNF)

- (a) *lexical rules match a constant number of spans*
- (b) *all unlexicalized rules are binary.*

Original rule:            **S** → No se **VB** **VB** un **NN** **PP**

LNf rule:                **S** → No se **VB~VB** un **NN~PP**

# Speeding up Lexical Rule Application

## Anchored Lexical Normal Form (ALNF)

- (a) *lexical rules match a constant number of spans*
- (b) *all unlexicalized rules are binary.*

Original rule:  $S \rightarrow \text{No se } VB \ VB \ \text{un } NN \ PP$

LNf rule:  $S \rightarrow \text{No se } VB \sim VB \ \text{un } NN \sim PP$

Transformed rules:  $S \rightarrow S/NN \sim PP \ NN \sim PP$

$S/NN \sim PP \rightarrow \text{No se } VB \sim VB \ \text{un}$

# Speeding up Lexical Rule Application

## Anchored Lexical Normal Form (ALNF)

- (a) *lexical rules match a constant number of spans*
- (b) *all unlexicalized rules are binary.*

Original rule:  $S \rightarrow \text{No se VB VB un NN PP}$

LNf rule:  $S \rightarrow \text{No se VB~VB un NN~PP}$

Transformed rules:  $S \rightarrow S/\text{NN~PP NN~PP}$

$S/\text{NN~PP} \rightarrow \text{No se VB~VB un}$

*All lexical rule yields begin and end with a lexical item*

# Speeding up Lexical Rule Application

## Anchored Lexical Normal Form (ALNF)

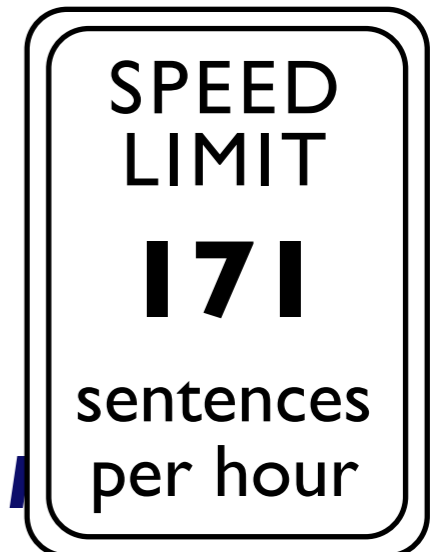
- (a) lexical rules match a constant number of spans
- (b) all unlexicalized rules are binary.

Original rule:  $S \rightarrow \text{No se VB VB un}$

LNf rule:  $S \rightarrow \text{No se VB} \sim \text{VB un}$

Transformed rules:  $S \rightarrow S/NN \sim PP \quad NN \sim PP$

$S/NN \sim PP \rightarrow \text{No se VB} \sim \text{VB un}$



*All lexical rule yields begin and end with a lexical item*

# Binarizing Sequences of Non-Terminals

---

*We must select a binary derivation for each non-terminal sequence*

**Original:**

**$S \rightarrow VB \ NP \ NP \ PP$**

**Binarization options:**

**$S \rightarrow VB \sim NP \sim NP \ PP$**

**$S \rightarrow VB \ NP \sim NP \sim PP$**

**$S \rightarrow VB \sim NP \ NP \sim PP$**

# Binarizing Sequences of Non-Terminals

*We must select a binary derivation for each non-terminal sequence*

## Original:

**$S \rightarrow VB \ NP \ NP \ PP$**

## Binarization options:

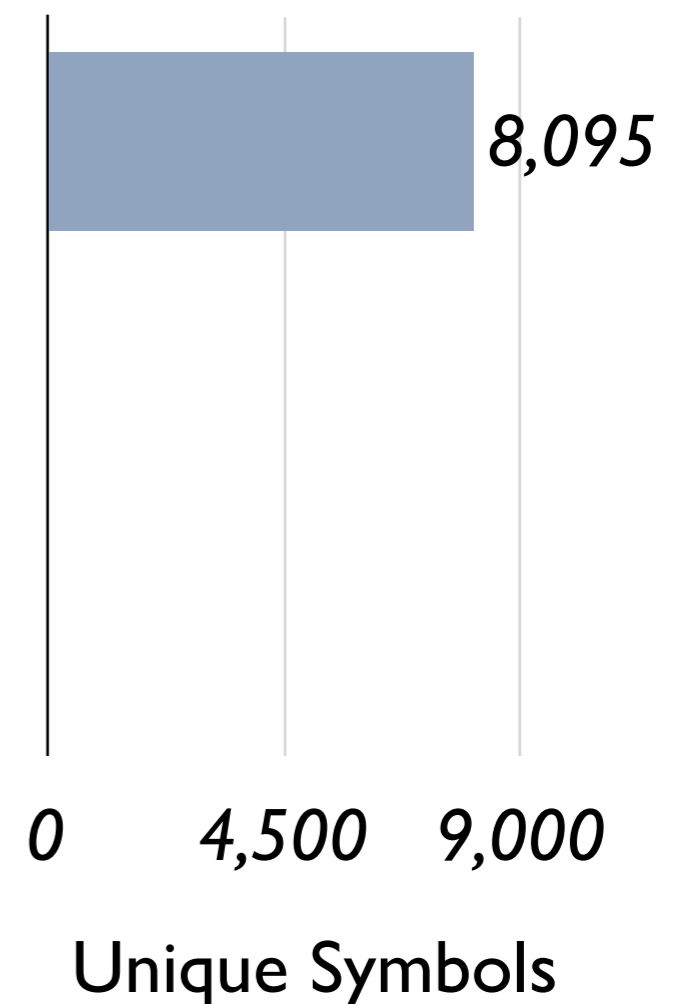
**$S \rightarrow VB \sim NP \sim NP \ PP$**

**$S \rightarrow VB \ NP \sim NP \sim PP$**

**$S \rightarrow VB \sim NP \ NP \sim PP$**

Binarization of an example sentence-specific grammar

*Right-branching*



# Binarizing Sequences of Non-Terminals

*We must select a binary derivation for each non-terminal sequence*

## Original:

**$S \rightarrow VB \ NP \ NP \ PP$**

## Binarization options:

**$S \rightarrow VB \sim NP \sim NP \ PP$**

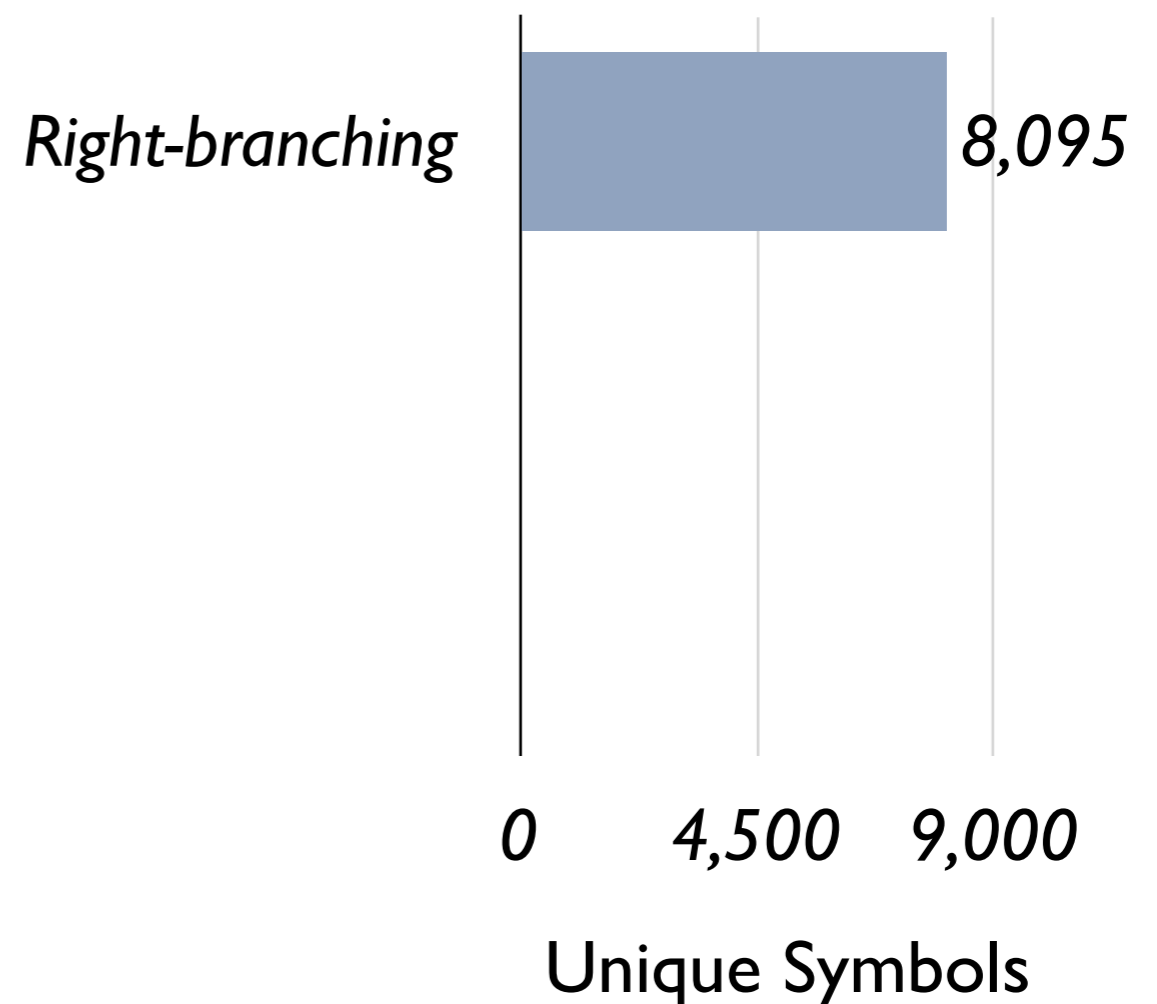
**$S \rightarrow VB \ NP \sim NP \sim PP$**

**$S \rightarrow VB \sim NP \ NP \sim PP$**

## Objective function:

*The minimum number of grammar symbols, such that all non-terminal sequences have binary derivations*

Binarization of an example sentence-specific grammar



# Binarizing Sequences of Non-Terminals

*We must select a binary derivation for each non-terminal sequence*

## Original:

**$S \rightarrow VB \ NP \ NP \ PP$**

## Binarization options:

**$S \rightarrow VB \sim NP \sim NP \ PP$**

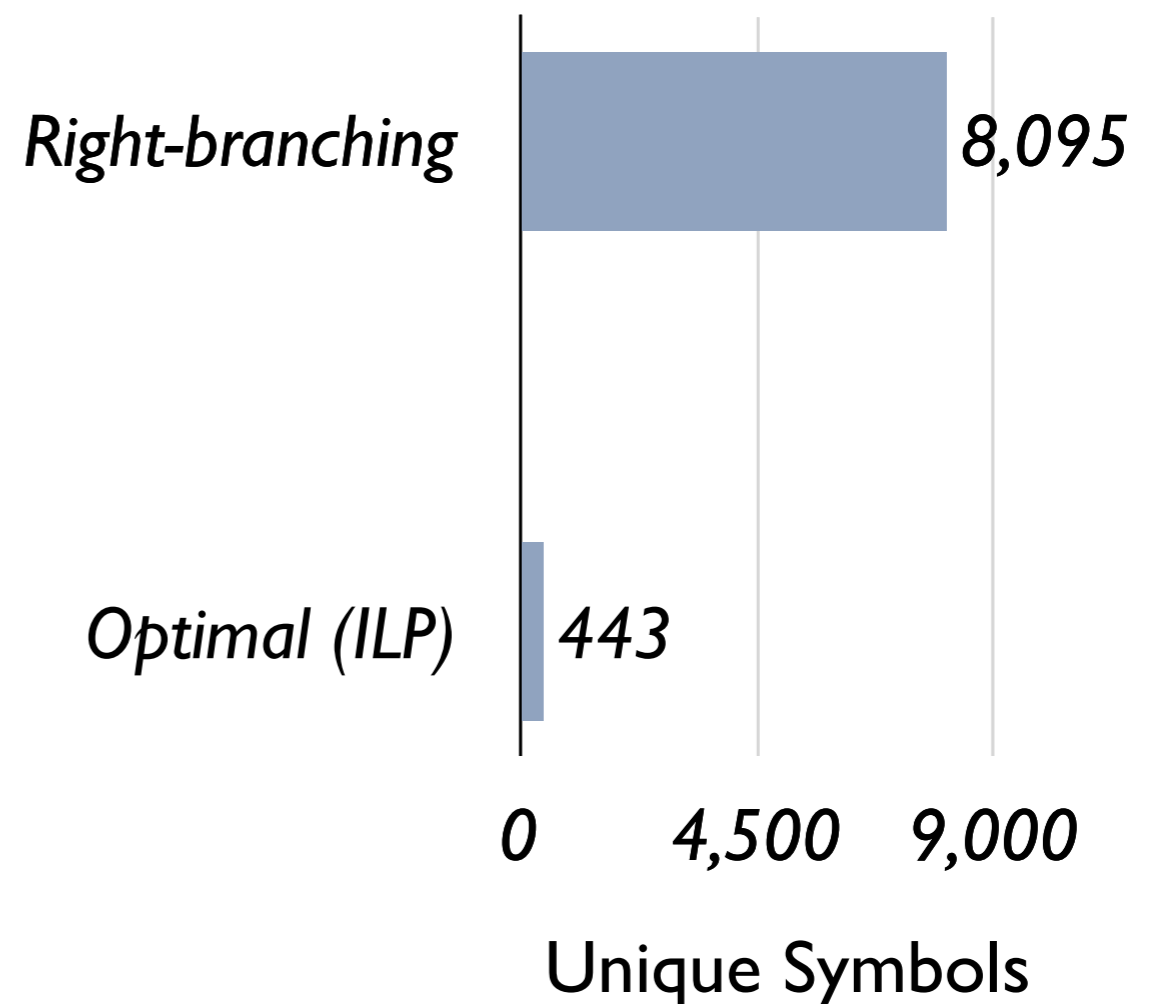
**$S \rightarrow VB \ NP \sim NP \sim PP$**

**$S \rightarrow VB \sim NP \ NP \sim PP$**

## Objective function:

*The minimum number of grammar symbols, such that all non-terminal sequences have binary derivations*

Binarization of an example sentence-specific grammar



# Binarizing Sequences of Non-Terminals

*We must select a binary derivation for each non-terminal sequence*

## Original:

**$S \rightarrow VB \ NP \ NP \ PP$**

## Binarization options:

**$S \rightarrow VB \sim NP \sim NP \ PP$**

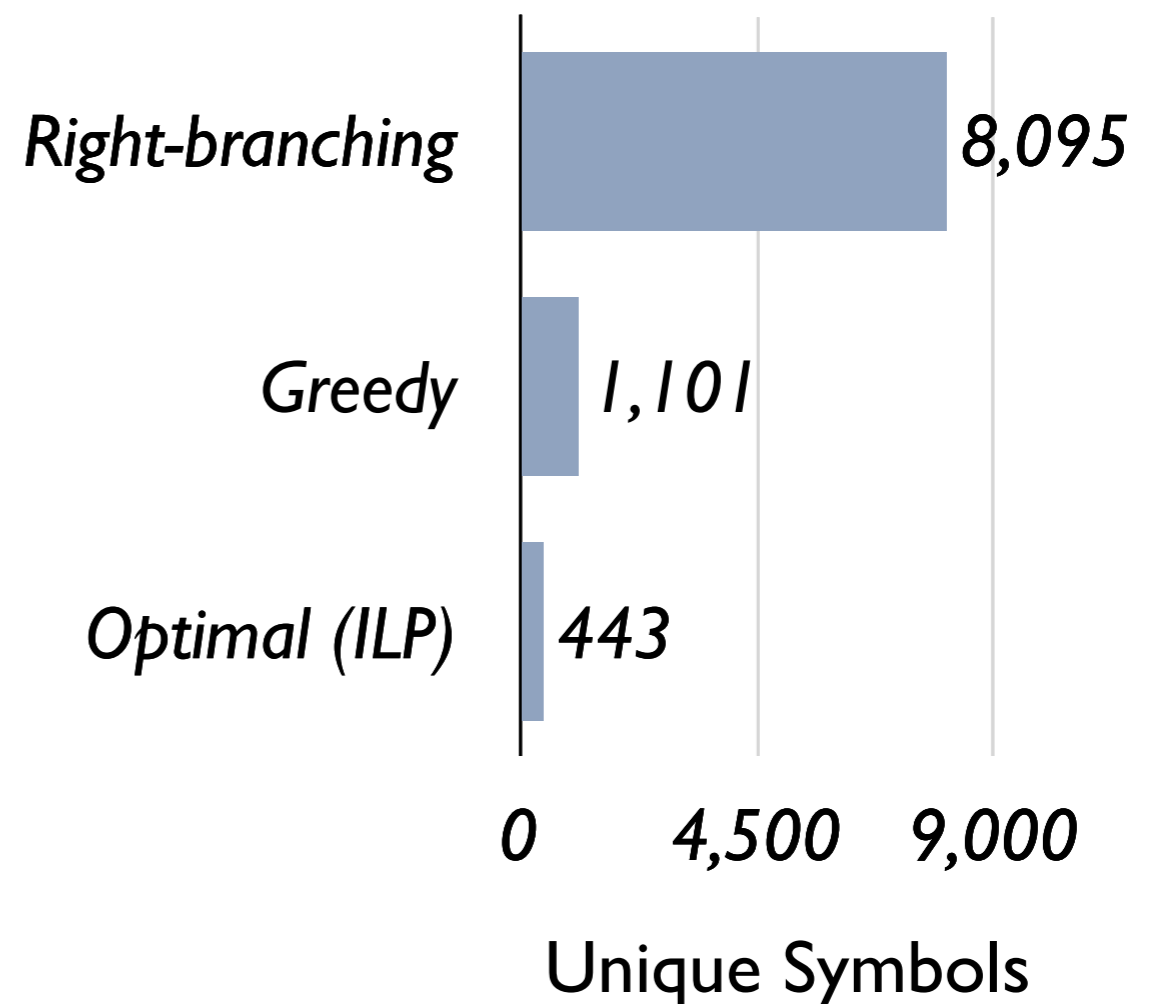
**$S \rightarrow VB \ NP \sim NP \sim PP$**

**$S \rightarrow VB \sim NP \ NP \sim PP$**

## Objective function:

*The minimum number of grammar symbols, such that all non-terminal sequences have binary derivations*

Binarization of an example sentence-specific grammar



# Subsets of Grammar Symbols

---

**Problem:** *Certain large rules always introduce new symbols*

**$S \rightarrow VBP \ RB \ VB \ TO \ \text{subir} \ NP \ VP$**

# Subsets of Grammar Symbols

**Problem:** *Certain large rules always introduce new symbols*

$S \rightarrow$   **$VBP$   $RB$   $VB$   $TO$**  *subir*  **$NP$   $VP$**   
     $\dashrightarrow$   **$VBP \sim RB \sim VB \sim TO$**

# Subsets of Grammar Symbols

**Problem:** *Certain large rules always introduce new symbols*

$S \rightarrow$  
***VBP RB VB TO***
*subir NP VP*  

 $\dashrightarrow$  ***VBP~RB~VB~TO***

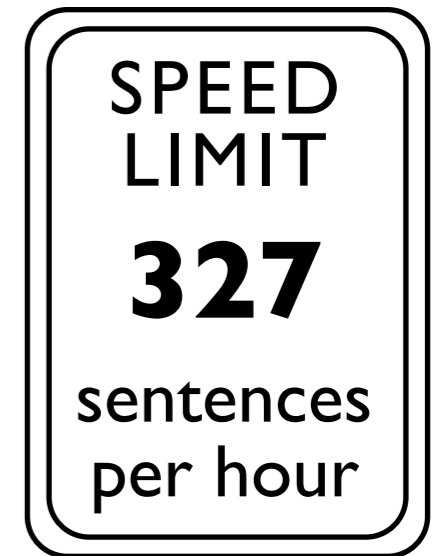
## ***Coarse-to-fine parsing:***

1. Choose a small subset of high-use symbols
2. Parse with the coarse grammar and prune unlikely states
3. Parse with the fine grammar in the pruned search space

# Subsets of Grammar Symbols

**Problem:** *Certain large rules always introduce new symbols*

$S \rightarrow$  **VBP RB VB TO** *subir* **NP VP**  
→ **VBP~RB~VB~TO**



## **Coarse-to-fine parsing:**

1. Choose a small subset of high-use symbols
2. Parse with the coarse grammar and prune unlikely states
3. Parse with the fine grammar in the pruned search space



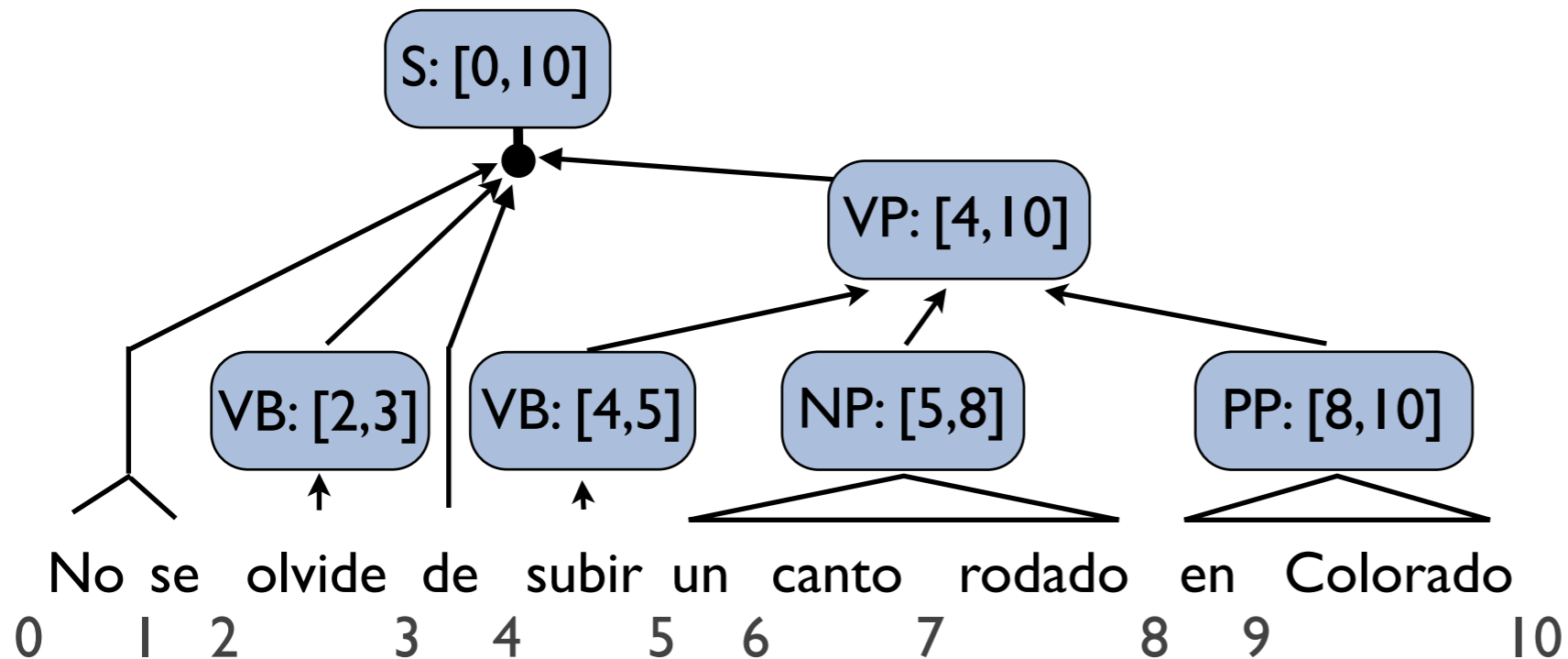
# Integrating a Language Model

---

**Approach:** *Top-down lazy forest reranking with priority queues (a.k.a., cube growing)*

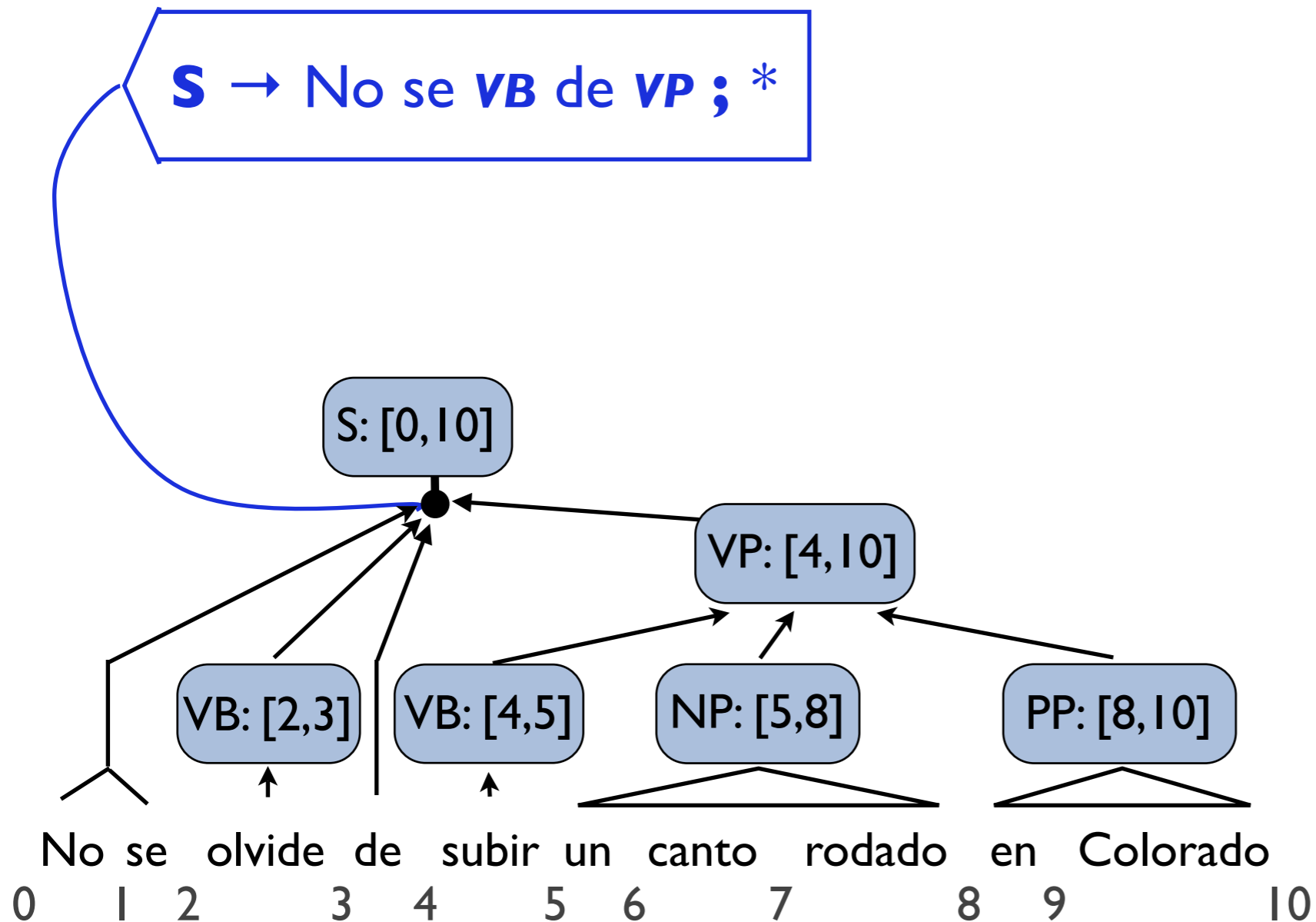
# Integrating a Language Model

**Approach:** *Top-down lazy forest reranking with priority queues (a.k.a., cube growing)*



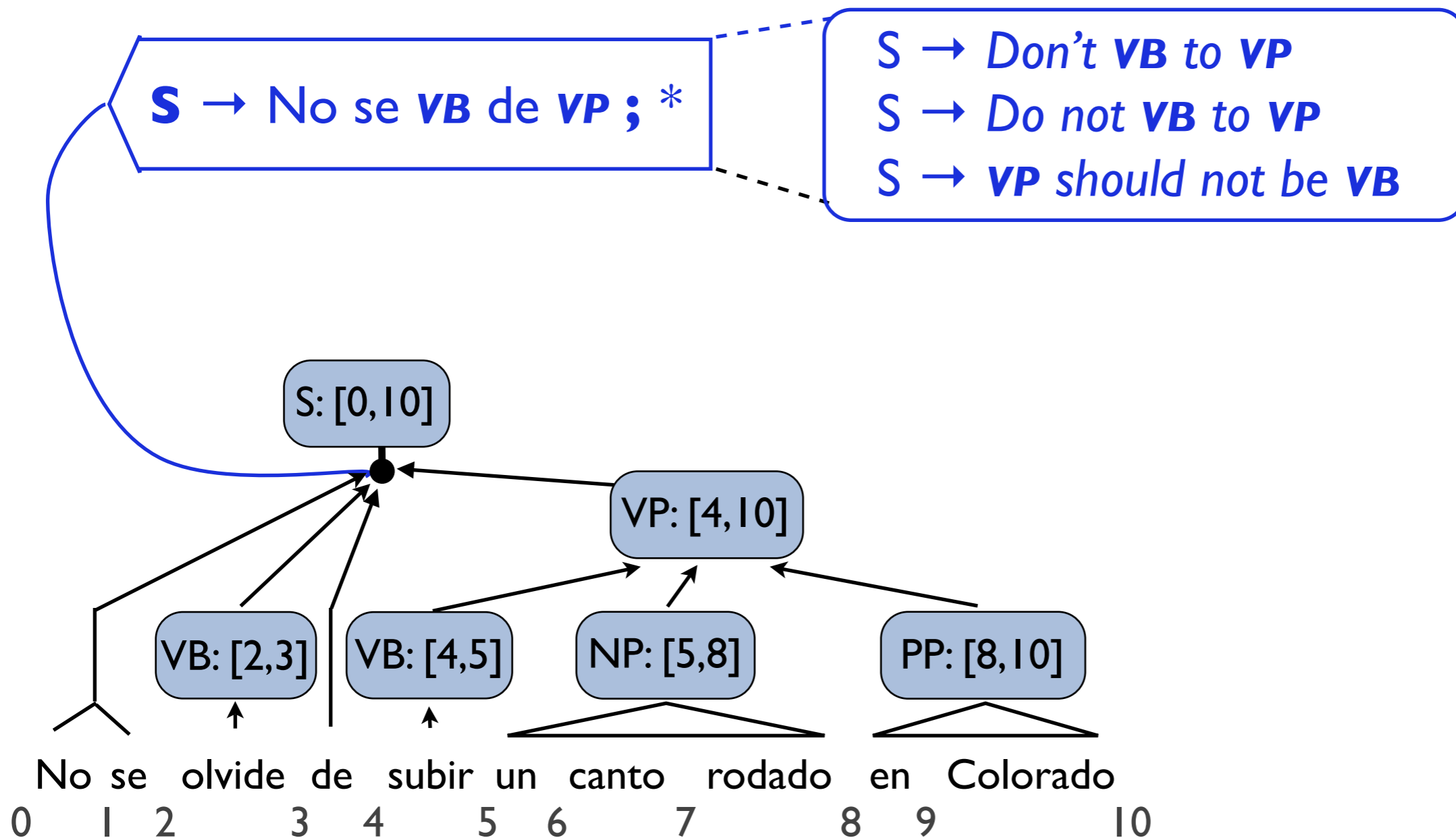
# Integrating a Language Model

**Approach:** *Top-down lazy forest reranking with priority queues (a.k.a., cube growing)*



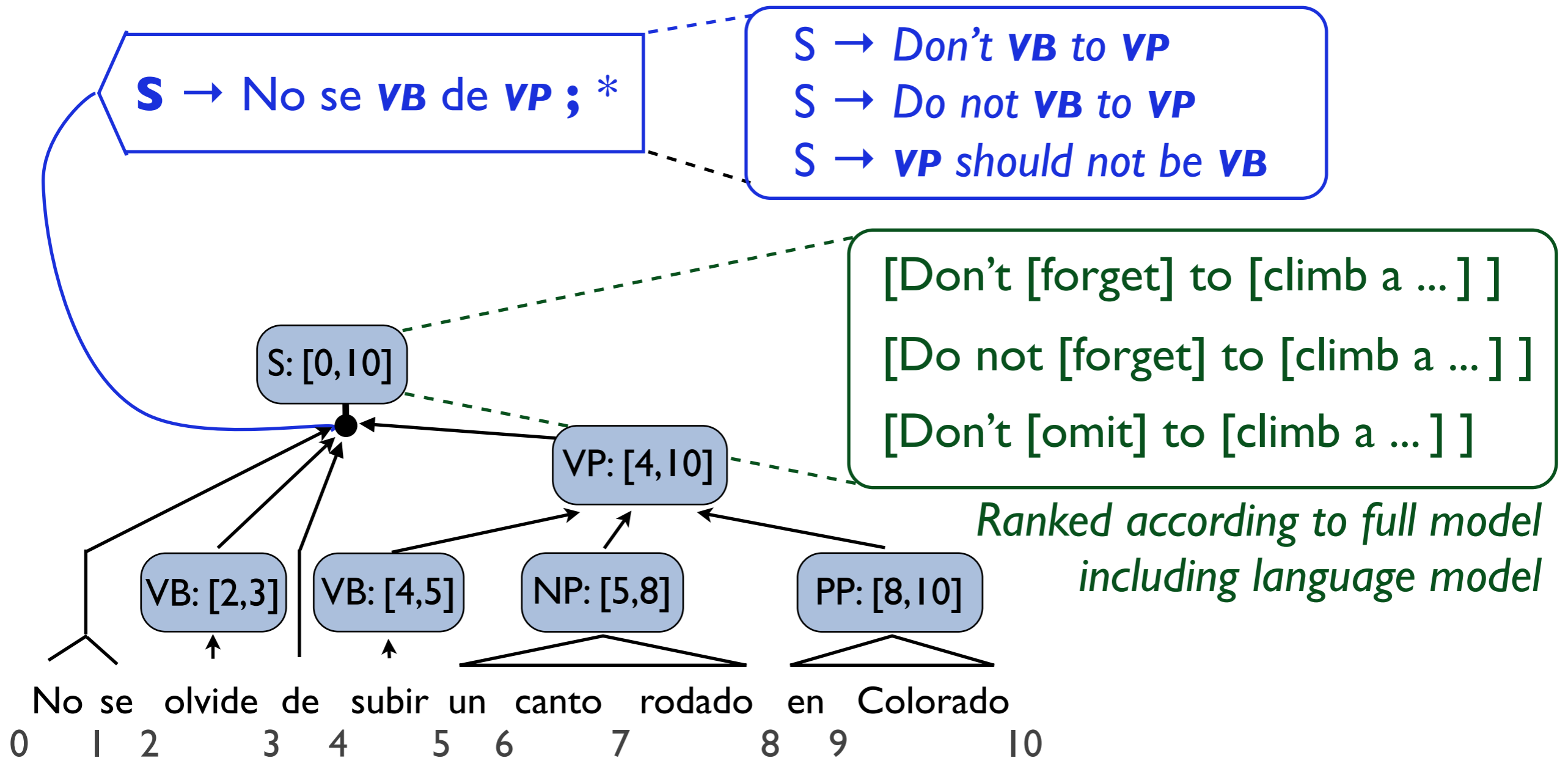
# Integrating a Language Model

**Approach:** *Top-down lazy forest reranking with priority queues (a.k.a., cube growing)*



# Integrating a Language Model

**Approach:** *Top-down lazy forest reranking with priority queues (a.k.a., cube growing)*



# Coarse-to-Fine LM Integration

---

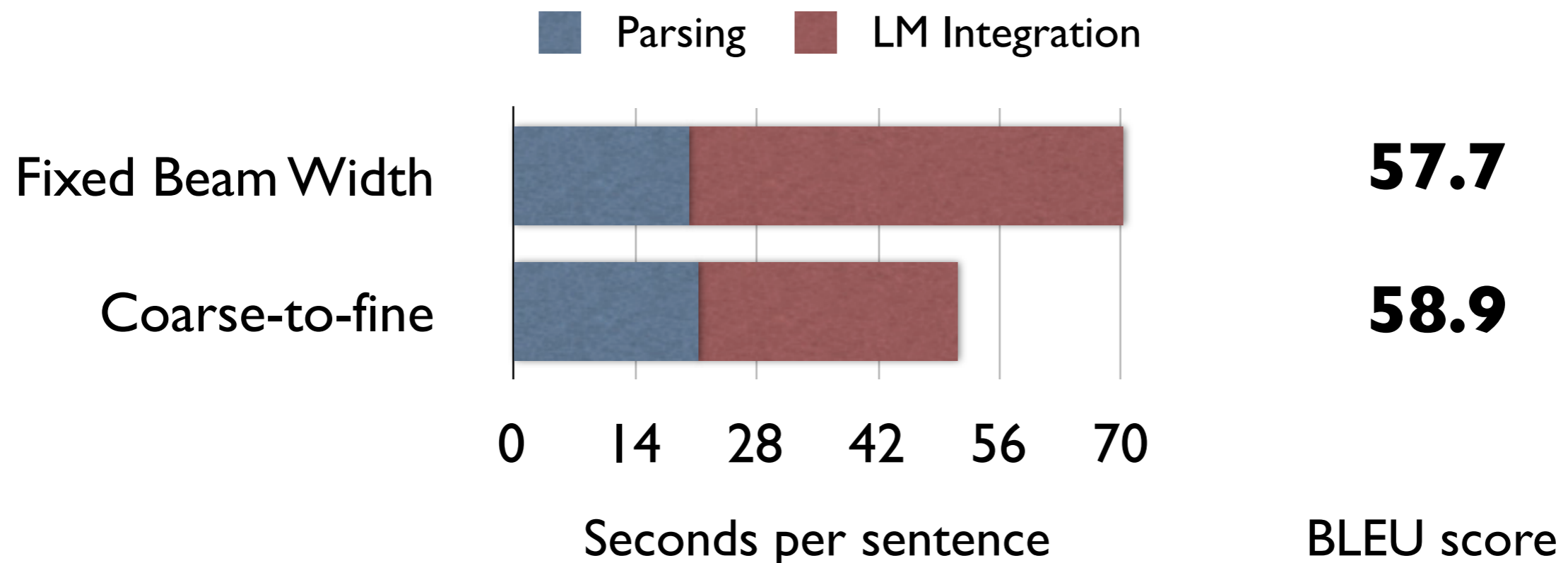
**Observation:** *The best translations almost always have a translation model score close to the Viterbi parse score*

**Coarse-to-fine beaming:** *A forest node's beam size is proportional to its posterior under the translation model*

# Coarse-to-Fine LM Integration

**Observation:** *The best translations almost always have a translation model score close to the Viterbi parse score*

**Coarse-to-fine beaming:** *A forest node's beam size is proportional to its posterior under the translation model*



# Summary

---

- Parsing with the projection of a tree transducer grammar is a non-trivial search problem
- Grammar transformations and algorithmic optimizations decrease parsing time
- Coarse-to-fine search speeds up parsing and makes language model integration more accurate

# Thanks!

---

## Questions?





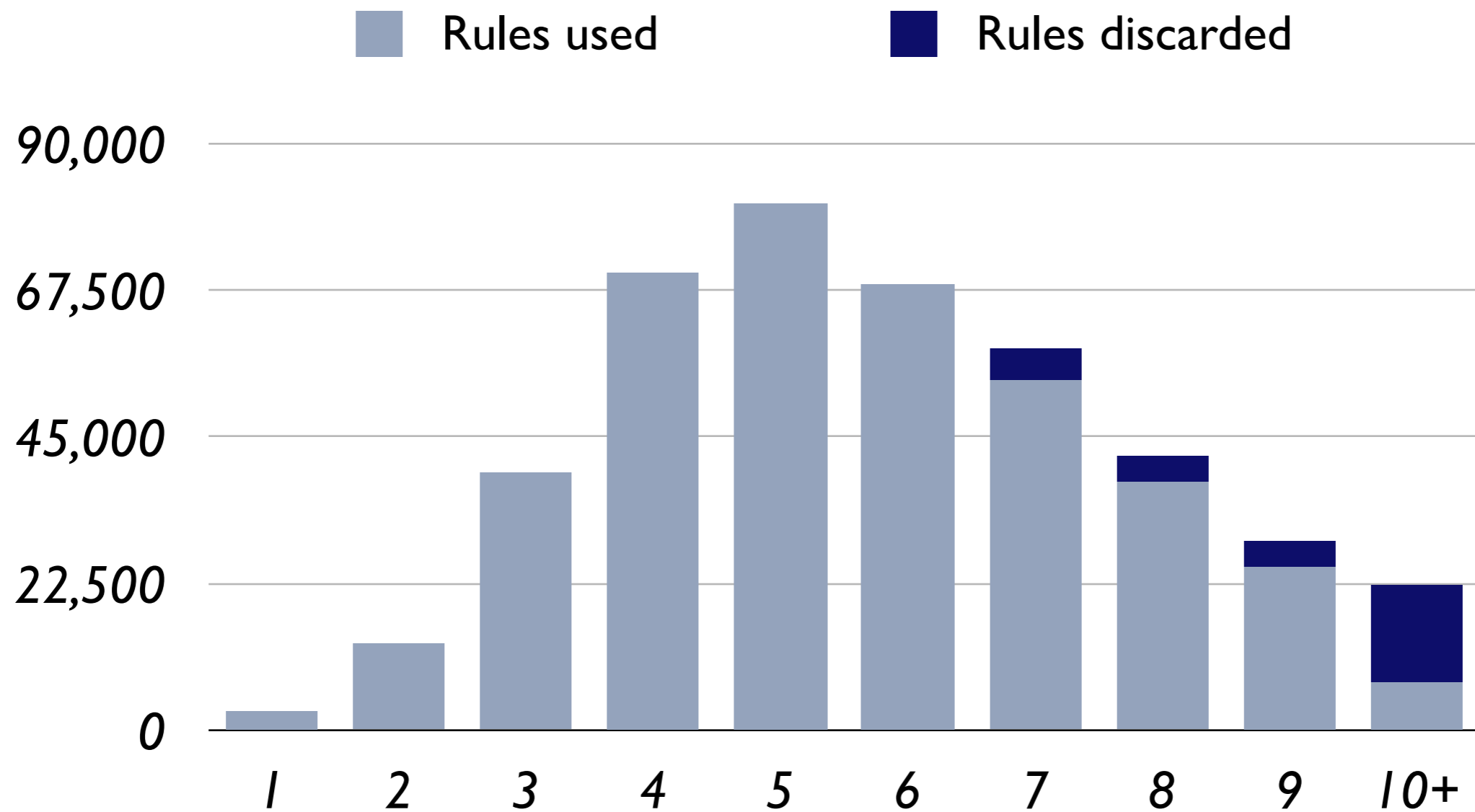
# The Size of Tree Transducer Grammars

---

■ Rules used

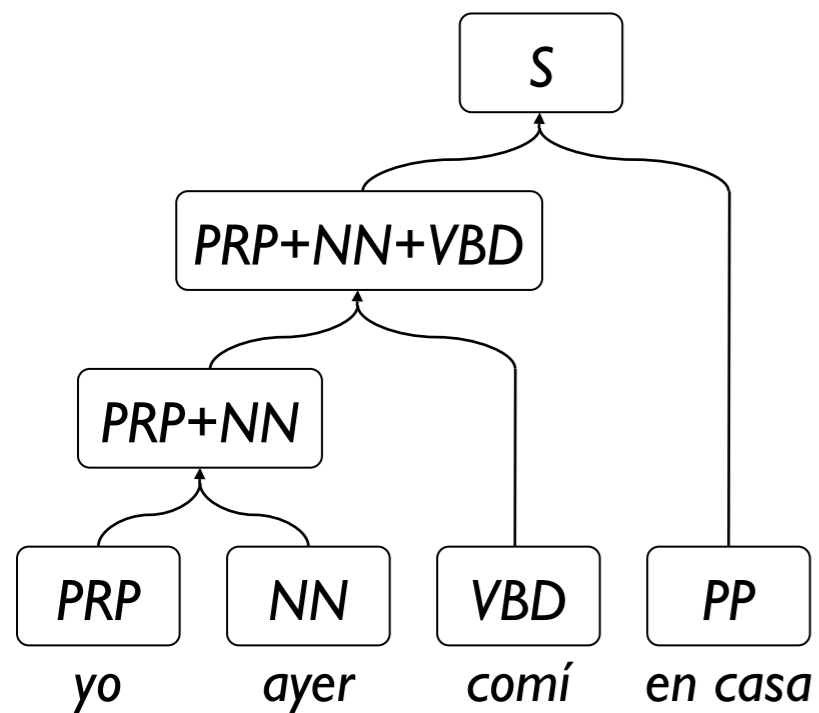
■ Rules discarded

# The Size of Tree Transducer Grammars



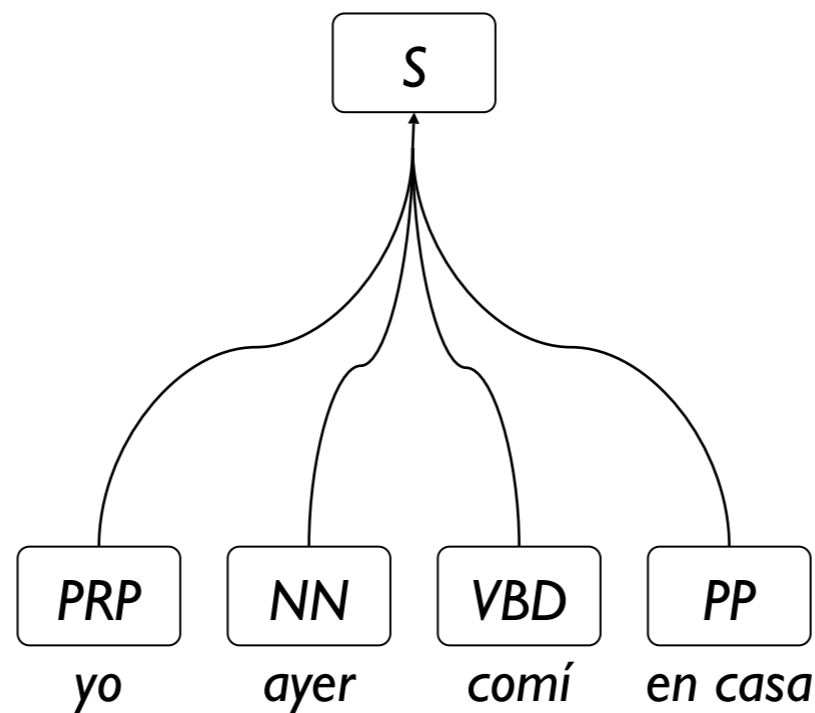
# Rebinarizing for LM Integration (ACL '09)

## Parse with ALNF grammar



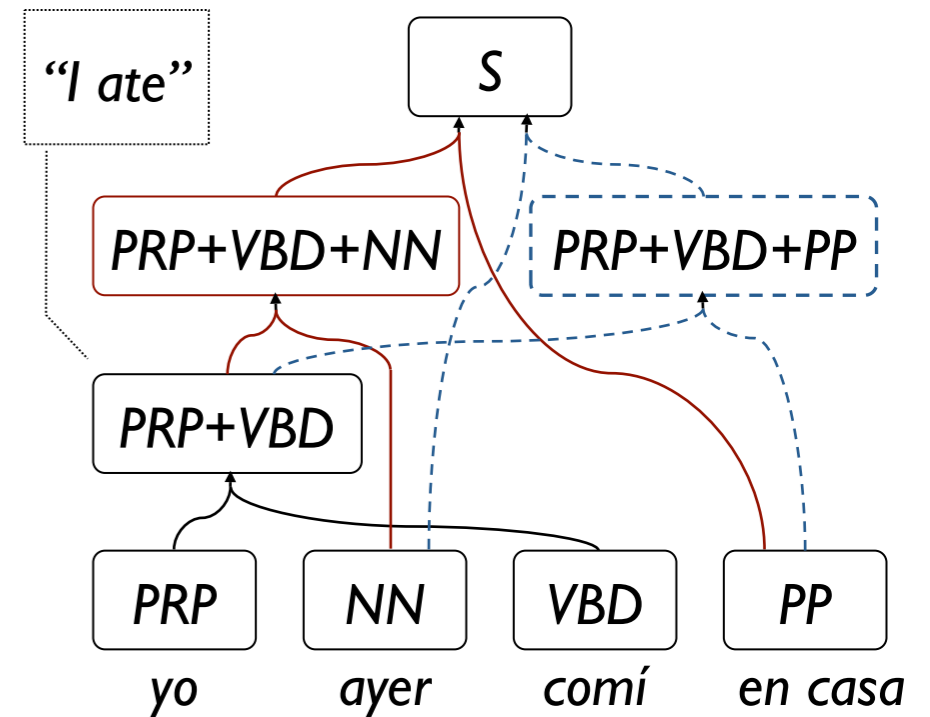
$S \rightarrow PRP_1 \ VBD_3 \ NN_2 \ PP_4$   
 $S \rightarrow [[PRP_1 \ NN_2] \ VBD_3] \ PP_4$   
 $S \rightarrow PRP_1 \ VBD_3 \ PP_4 \ NN_2$   
 $S \rightarrow [[PRP_1 \ NN_2] \ VBD_3] \ PP_4$

## Collapse out binarization



$S \rightarrow PRP_1 \ VBD_3 \ NN_2 \ PP_4$   
 $S \rightarrow PRP_1 \ NN_2 \ VBD_3 \ PP_4$   
 $S \rightarrow PRP_1 \ VBD_3 \ PP_4 \ NN_2$   
 $S \rightarrow PRP_1 \ NN_2 \ VBD_3 \ PP_4$

## Rebinarize for LM integration

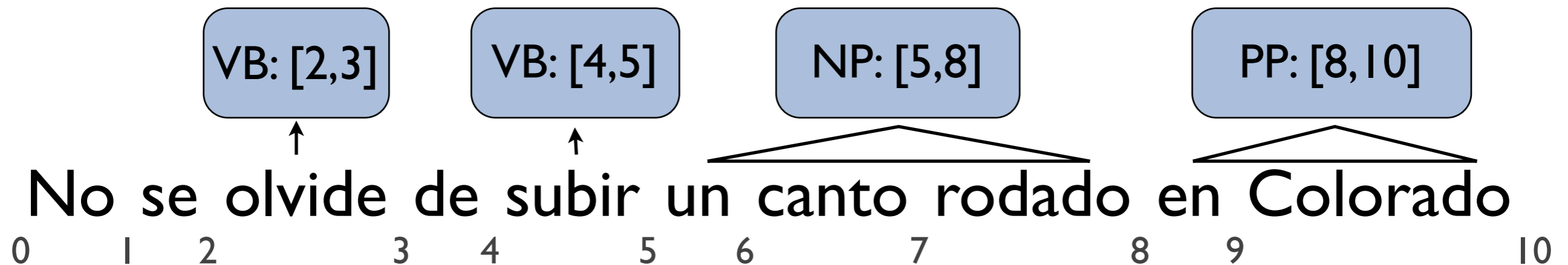


$S \rightarrow [[PRP_1 \ VBD_3] \ NN_2] \ PP_4$   
 $S \rightarrow PRP_1 \ NN_2 \ VBD_3 \ PP_4$   
 $S \rightarrow [[PRP_1 \ VBD_3] \ PP_4] \ NN_2$   
 $S \rightarrow PRP_1 \ NN_2 \ VBD_3 \ PP_4$

# Multi-Pass Syntactic Decoding

Parse input sentence  
with source-side  
grammar projection

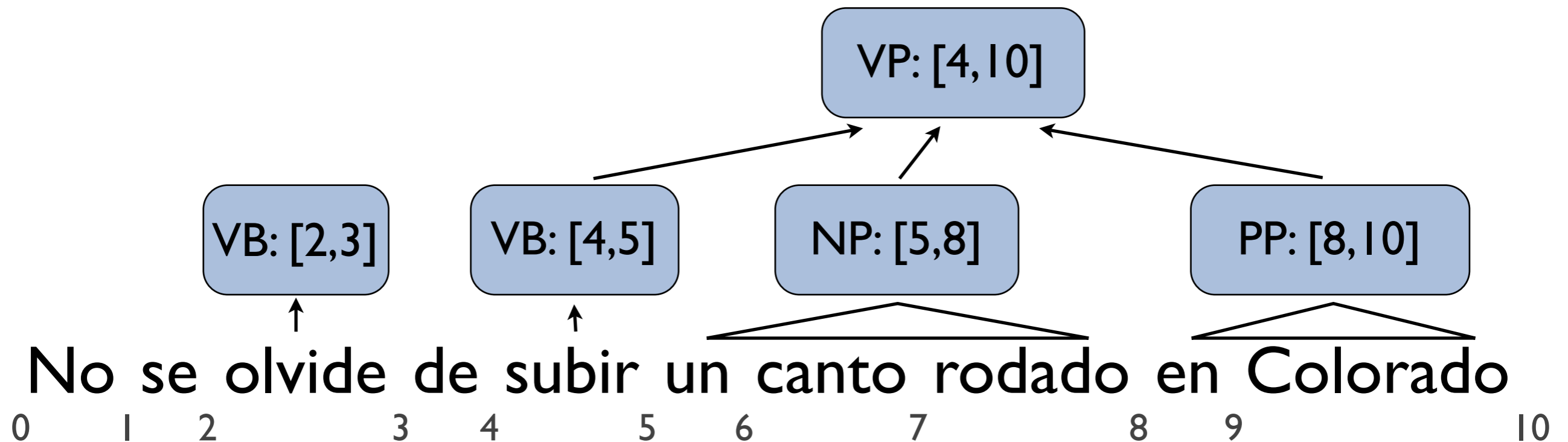
Rerank derivations  
rooted at each state with  
a language model



# Multi-Pass Syntactic Decoding

Parse input sentence  
with source-side  
grammar projection

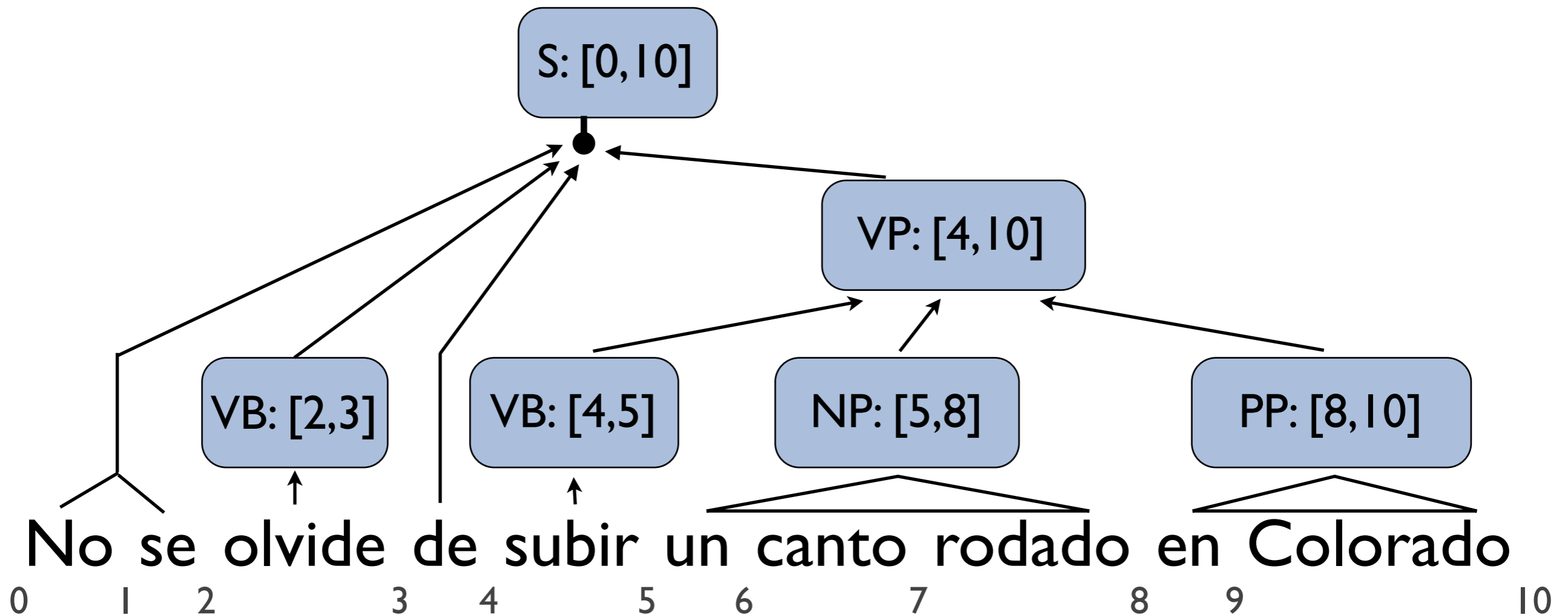
Rerank derivations  
rooted at each state with  
a language model



# Multi-Pass Syntactic Decoding

Parse input sentence  
with source-side  
grammar projection

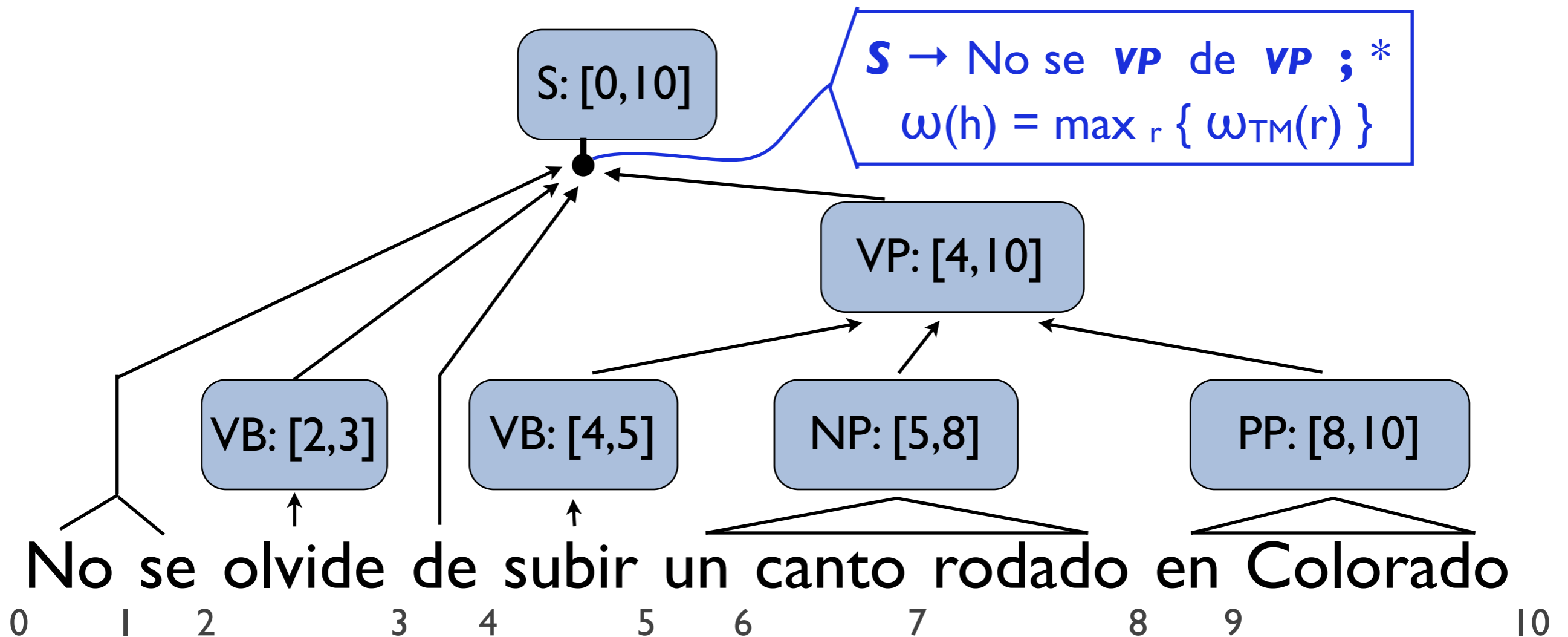
Rerank derivations  
rooted at each state with  
a language model



# Multi-Pass Syntactic Decoding

Parse input sentence with source-side grammar projection

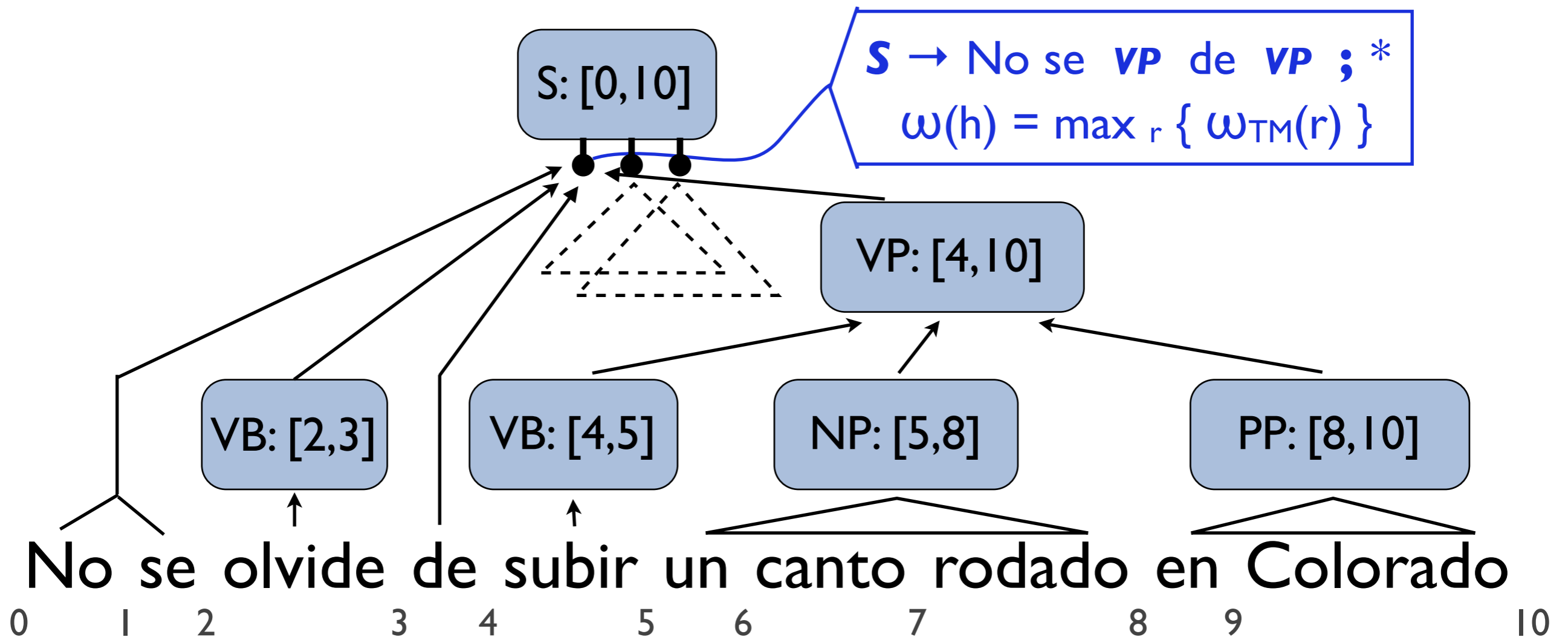
Rerank derivations rooted at each state with a language model



# Multi-Pass Syntactic Decoding

Parse input sentence with source-side grammar projection

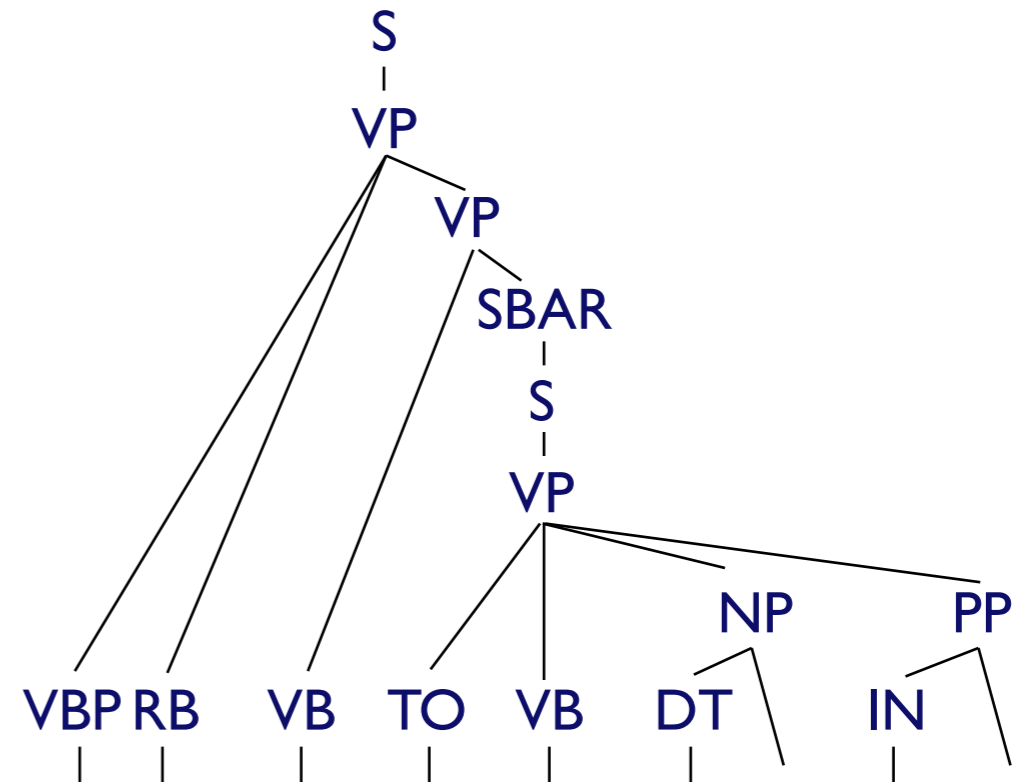
Rerank derivations rooted at each state with a language model



# Tree Transducer Grammars

No se olvide de subir un canto rodado en Colorado

## Rules



No se olvide de subir un  $NN_1$  en  $NNP_2$  ; *Don't forget to climb a  $NN_1$  in  $NNP_2$*

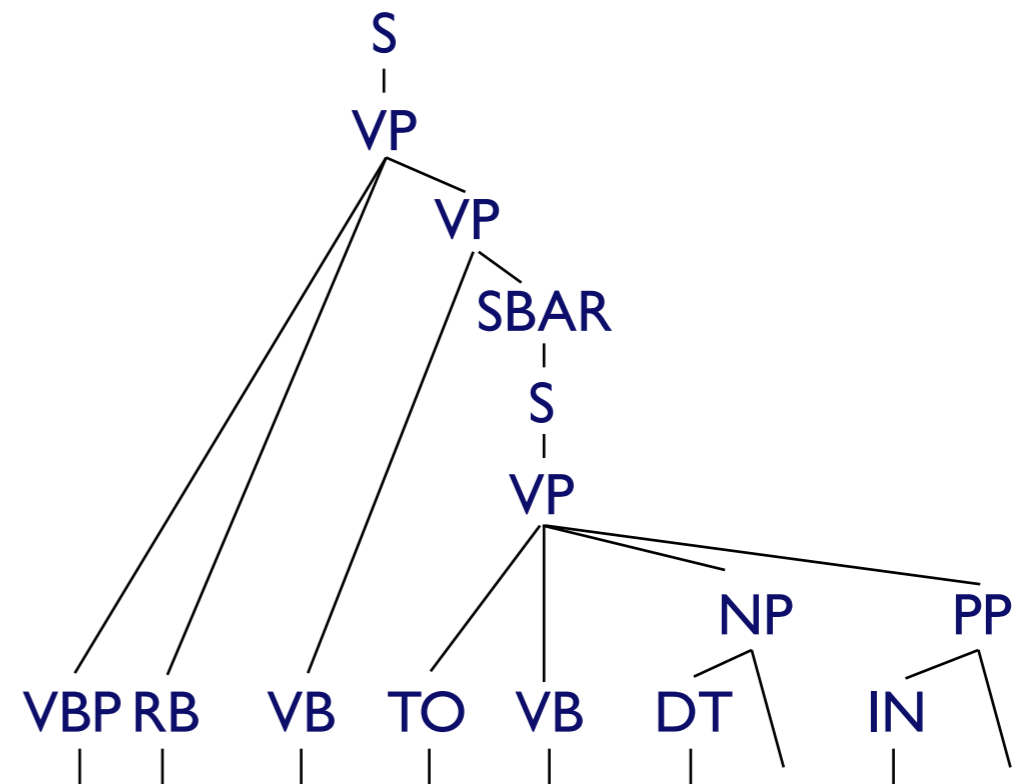
# Tree Transducer Grammars

No se olvide de subir un canto rodado en Colorado

## Rules

$\begin{array}{c}
 \text{NN} \\
 | \\
 \text{canto rodado ; } \textit{boulder}
 \end{array}$

$\begin{array}{c}
 \text{NNP} \\
 | \\
 \text{Colorado ; } \textit{Colorado}
 \end{array}$



No se olvide de subir un  $NN_1$  en  $NNP_2$  ; *Don't forget to climb a  $NN_1$  in  $NNP_2$*

# Tree Transducer Grammars

No se olvide de subir un canto rodado en Colorado

## Rules

