# Efficient Parsing for Transducer Grammars

University of California

CAL NLP

Berkeley

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**

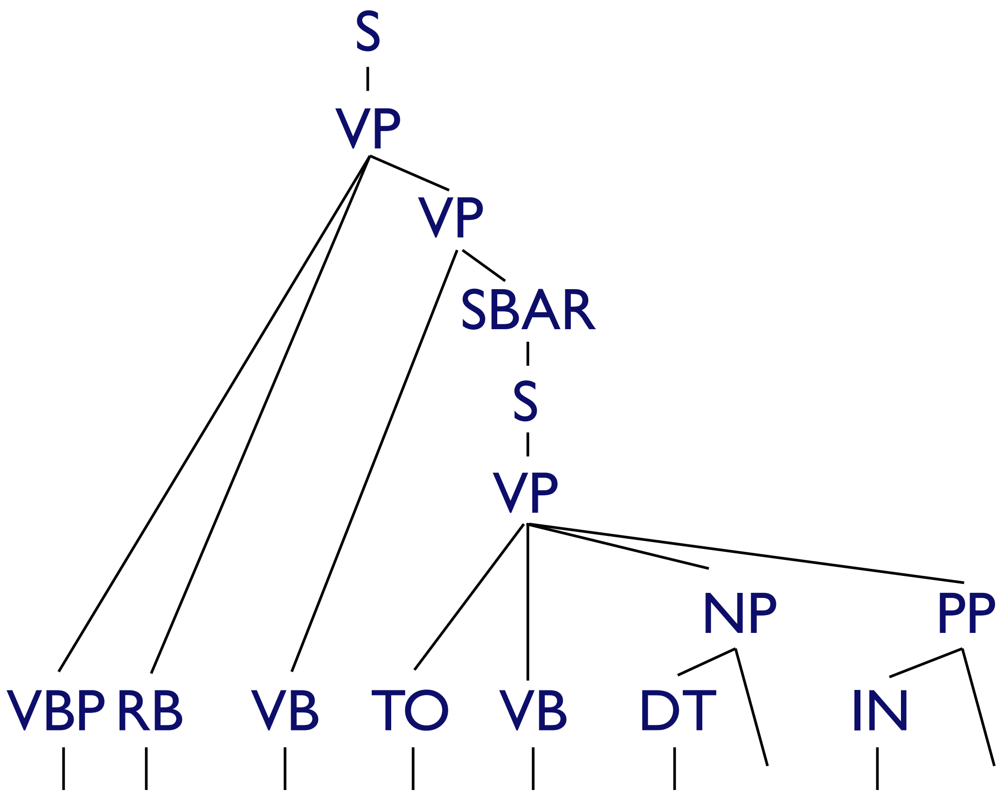
# Tree Transducer Grammars

No se olvide de subir un 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 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

NN

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**

NN

boulder

# Tree Transducer Grammars

NN NNP

No se olvide de subir un canto rodado en Colorado

### Synchronous Grammar

***NNP*** → Colorado ; *Colorado*

***NN*** → canto rodado ; *boulder*

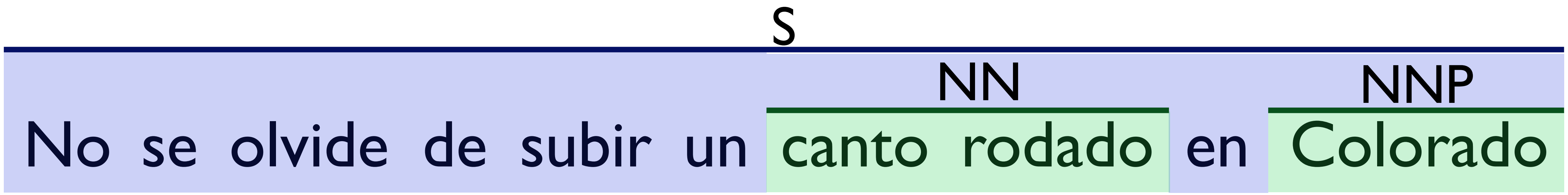
***S*** → No se olvide de subir un ***NN*** en ***NNP*** ; *Don't forget to climb a **NN** in **NNP***

### Output

NN NNP

boulder Colorado

# Tree Transducer Grammars



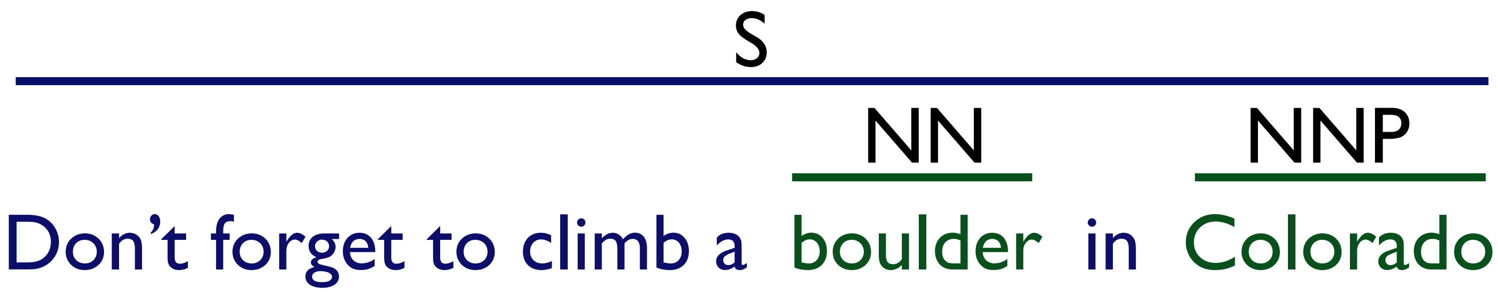
**Synchronous Grammar**

***NNP*** → Colorado ; *Colorado*

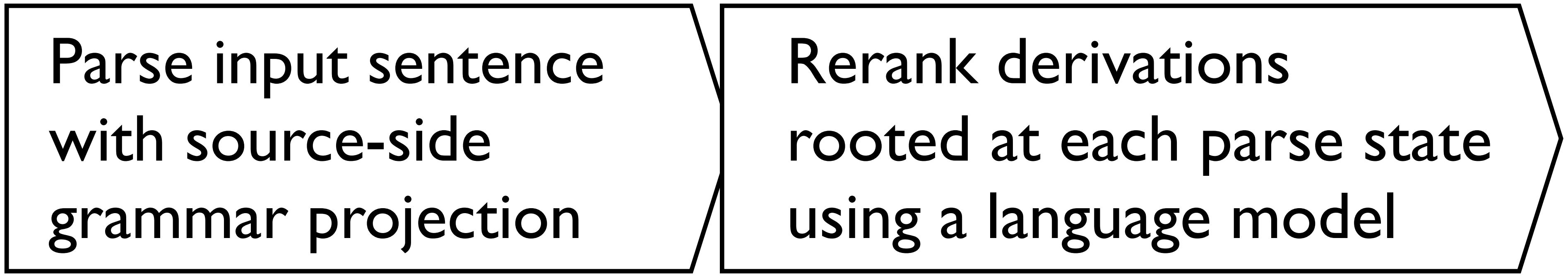
***NN*** → canto rodado ; *boulder*

***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

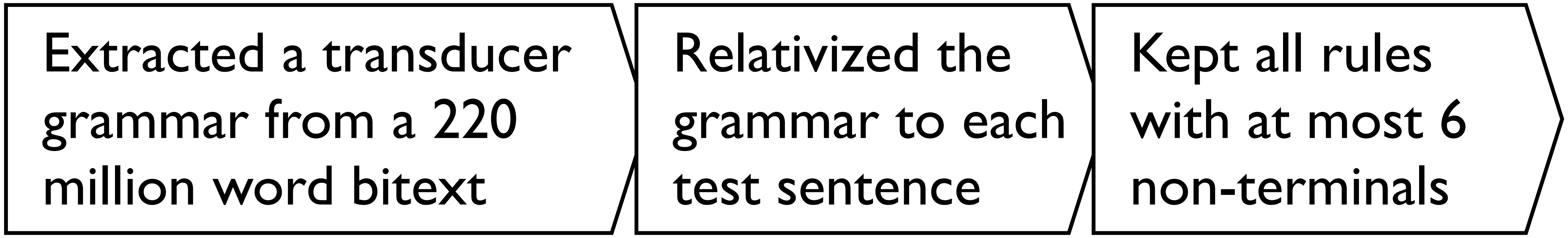
Rerank derivations rooted at each parse state using a language model

**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

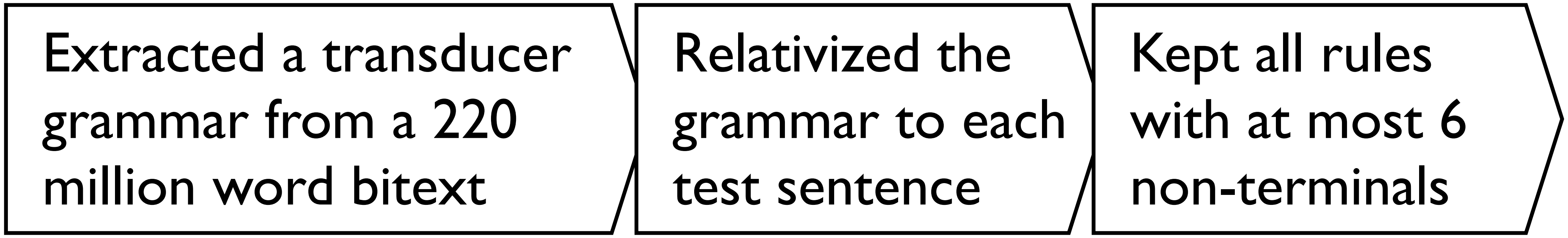


Extracted a transducer grammar from a 220 million word bitext

Relativized the grammar to each test sentence

Kept all rules with at most 6 non-terminals

# The Size of Tree Transducer Grammars

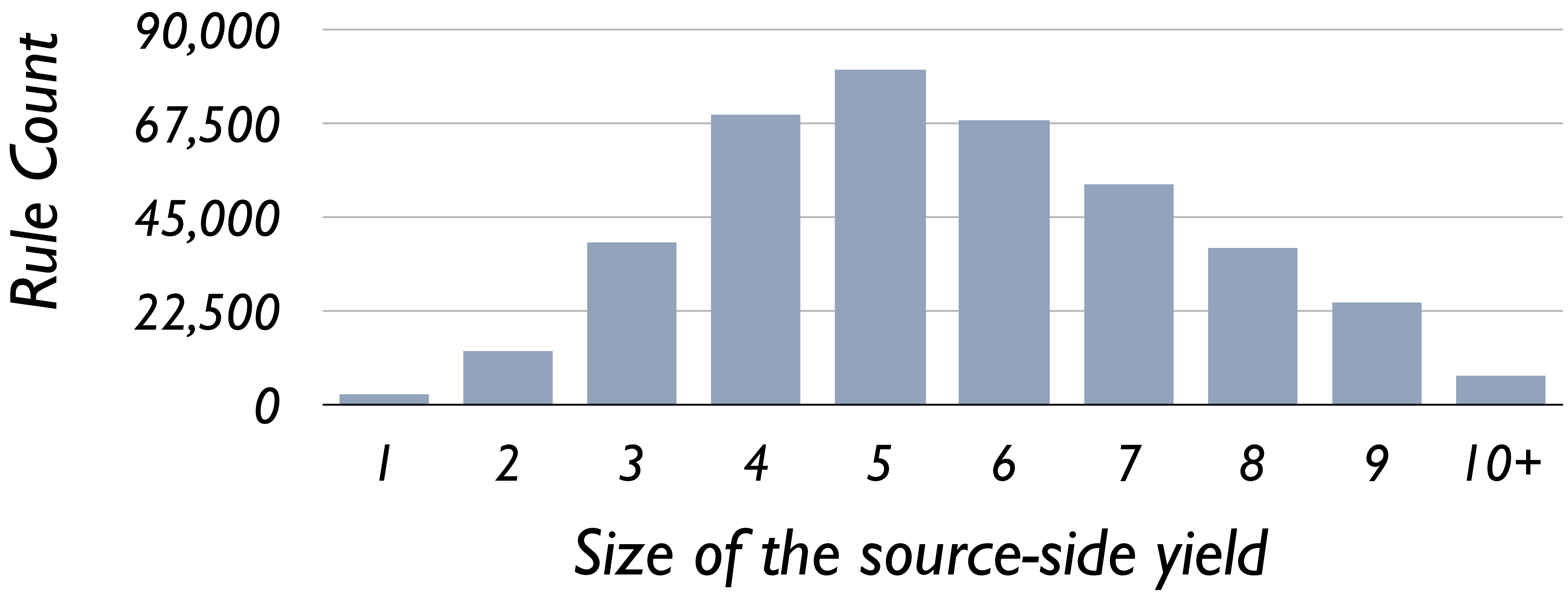


Extracted a transducer grammar from a 220 million word bitext

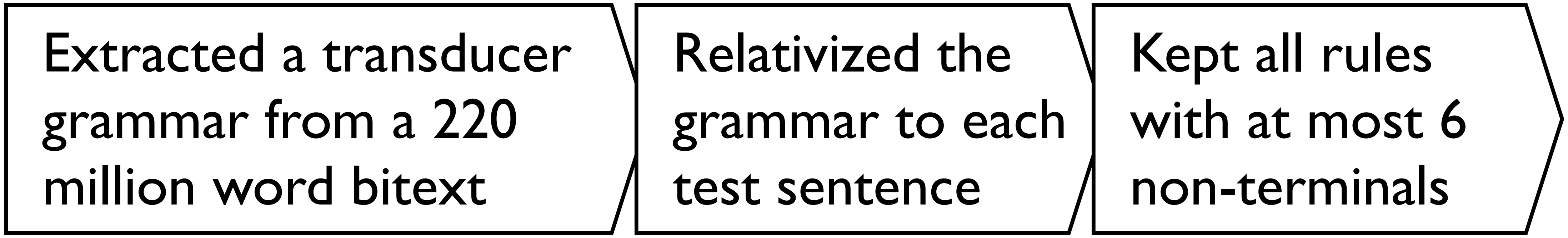
Relativized the grammar to each test sentence

Kept all rules with at most 6 non-terminals

## Rules matching an example 40-word sentence



# The Size of Tree Transducer Grammars

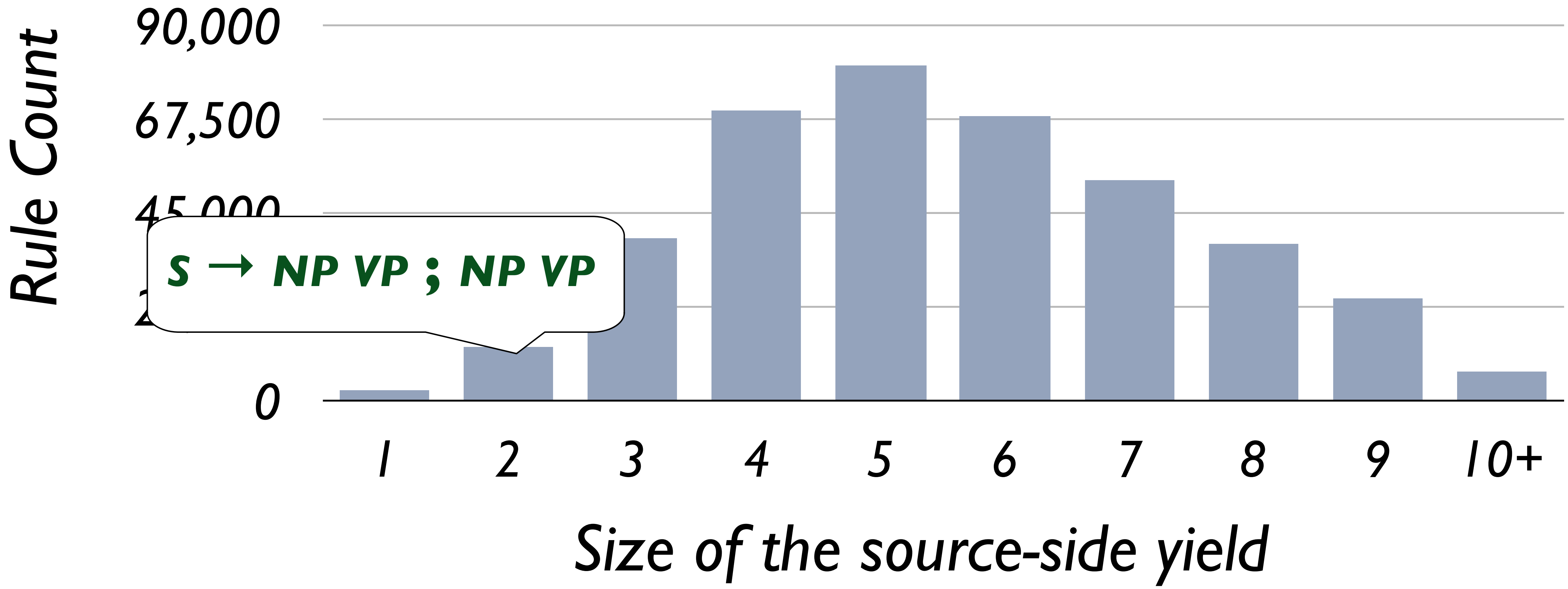


Extracted a transducer grammar from a 220 million word bitext

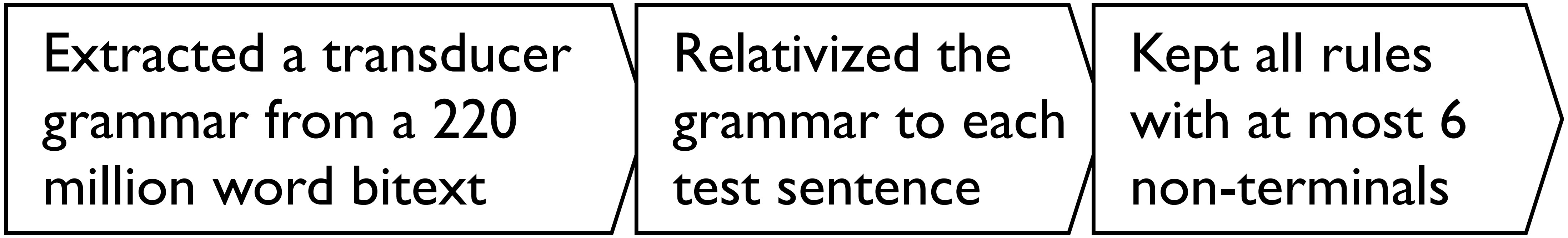
Relativized the grammar to each test sentence

Kept all rules with at most 6 non-terminals

## Rules matching an example 40-word sentence



# The Size of Tree Transducer Grammars

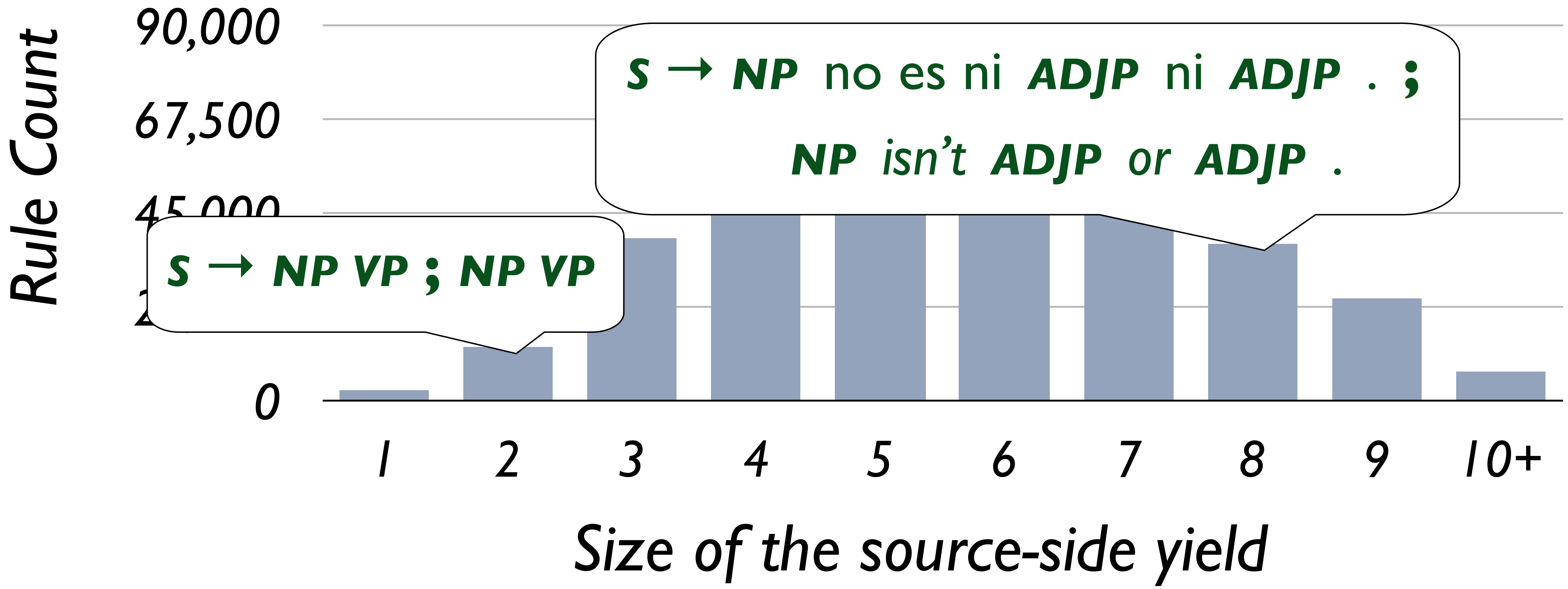


Extracted a transducer grammar from a 220 million word bitext

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

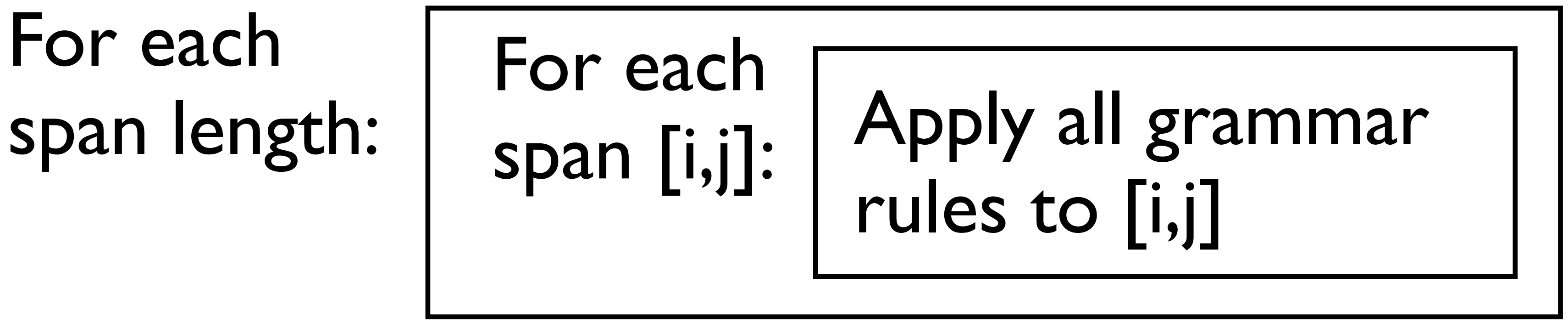
For each
span length:

# CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

# CKY-style Bottom-up Parsing

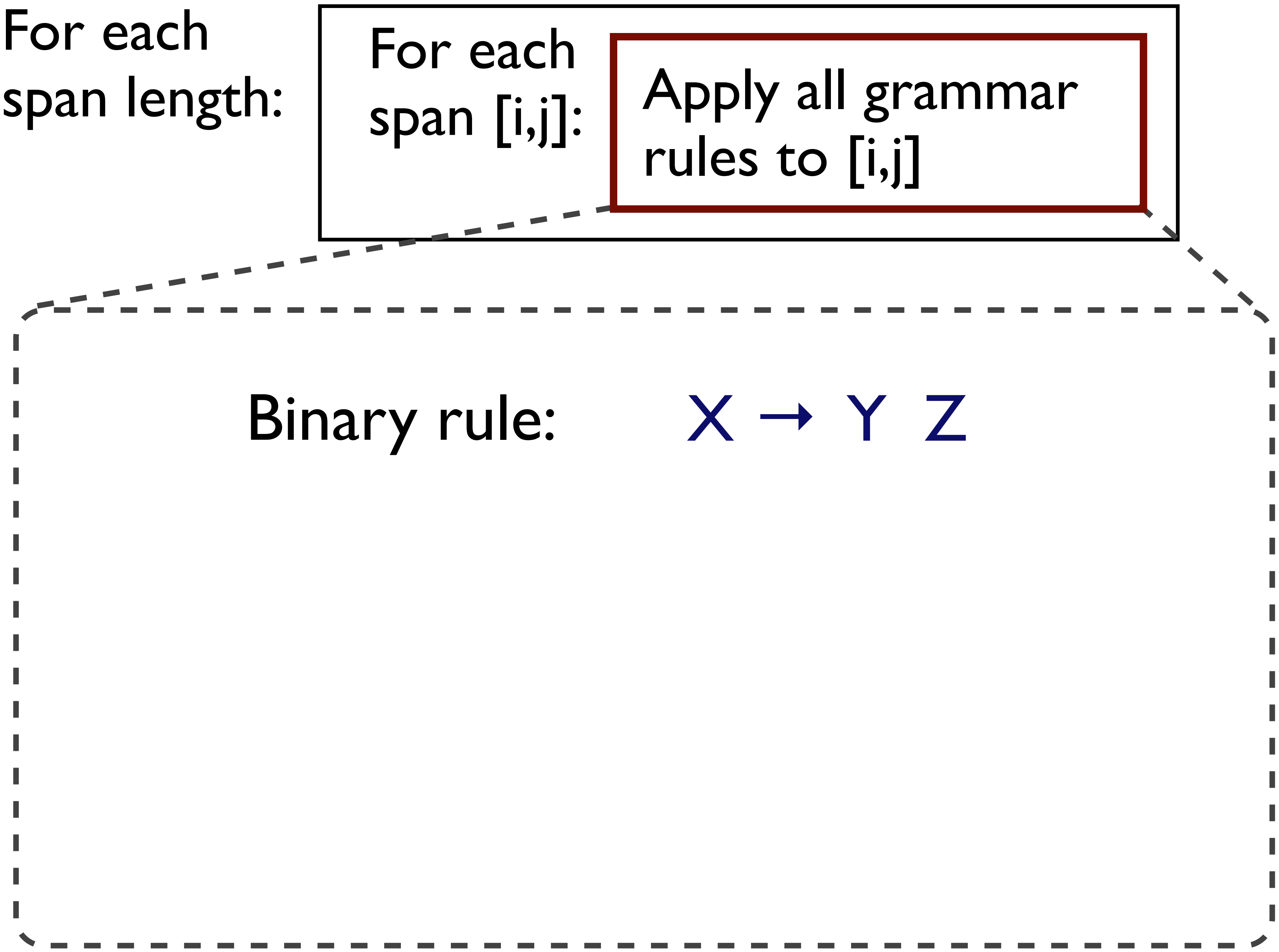


For each span length:

For each span [i,j]:

Apply all grammar rules to [i,j]

# CKY-style Bottom-up Parsing



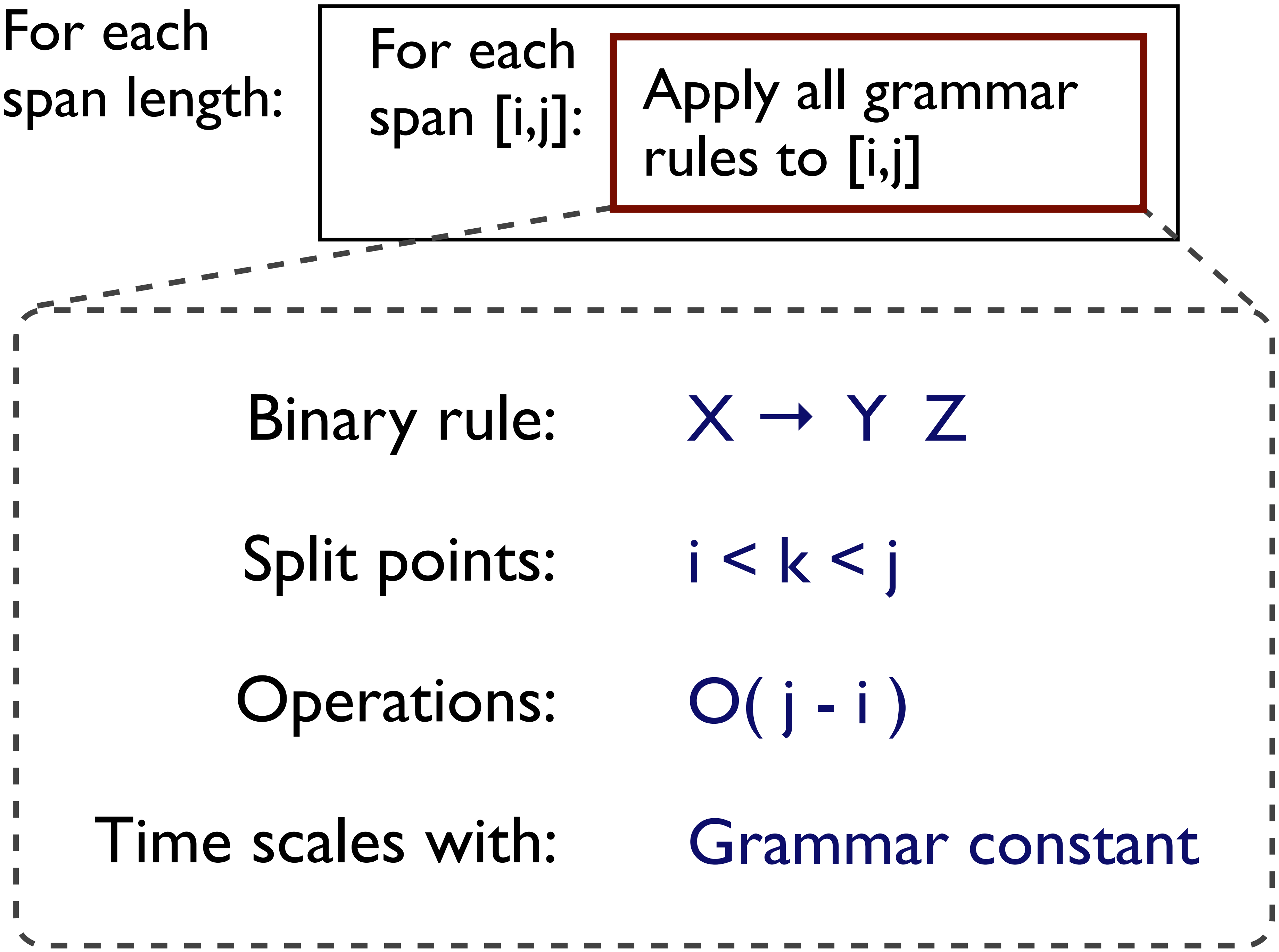
For each span length:

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 → 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

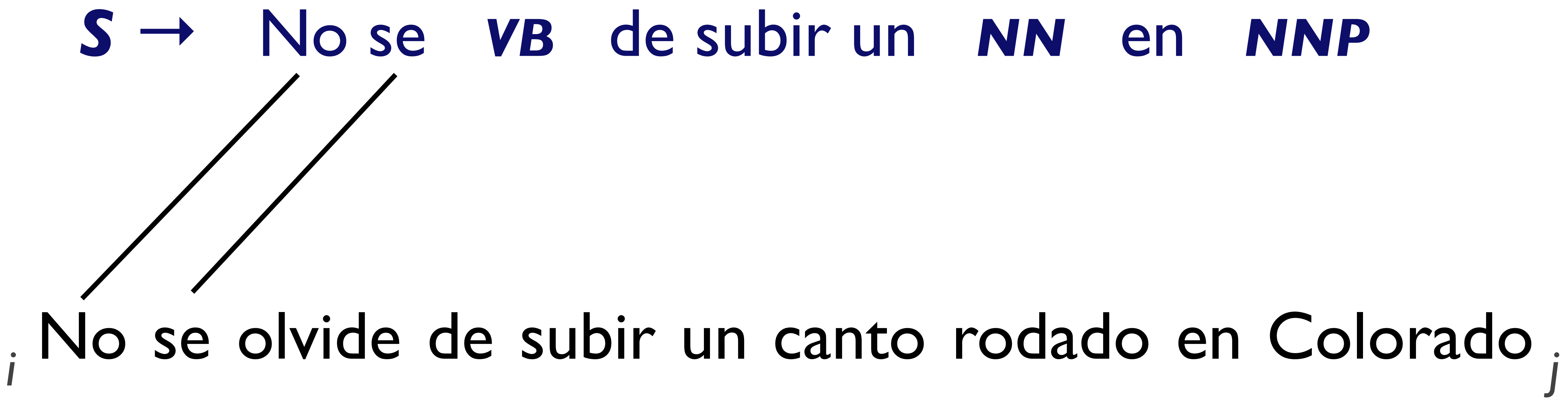
```
For each span length:
    For each span [i,j]:
        Apply all grammar rules to [i,j]
```

***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]
```

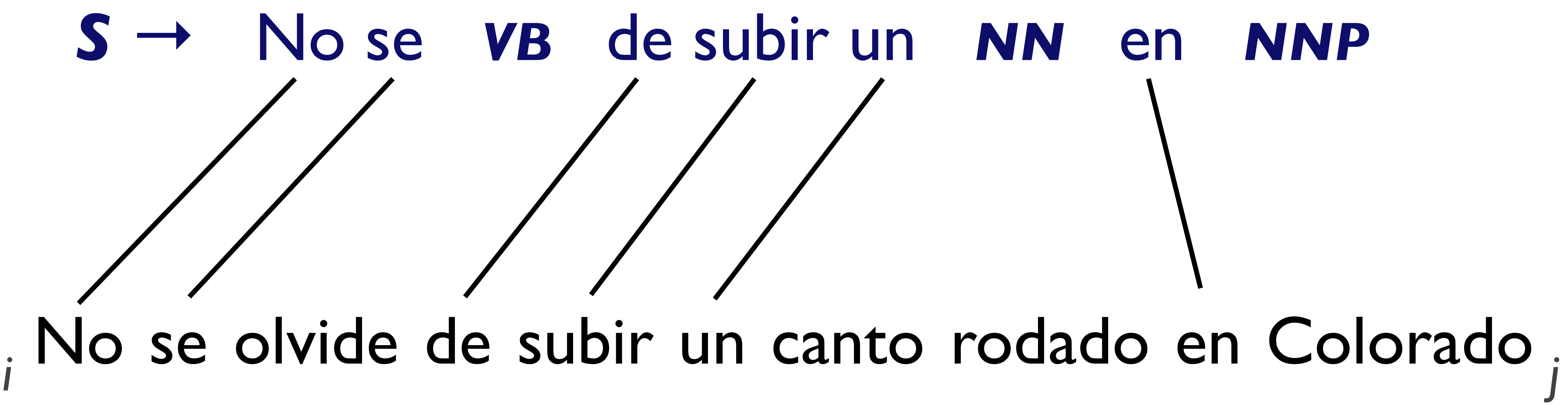


***S*** → No se ***VB*** de subir un ***NN*** en ***NNP***

*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]
```

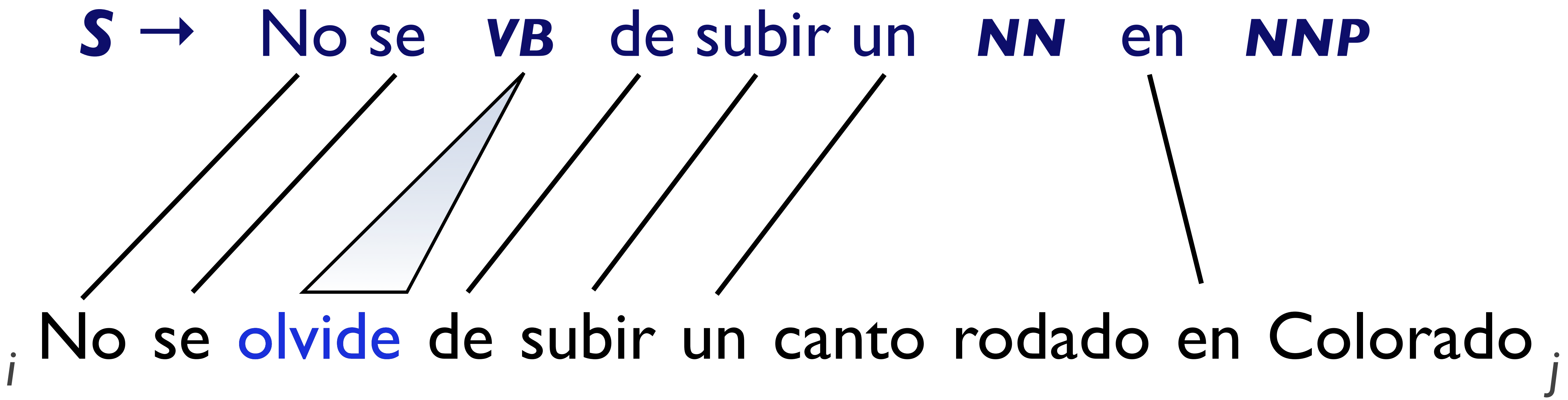


***S*** → No se ***VB*** de subir un ***NN*** en ***NNP***

*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]
```

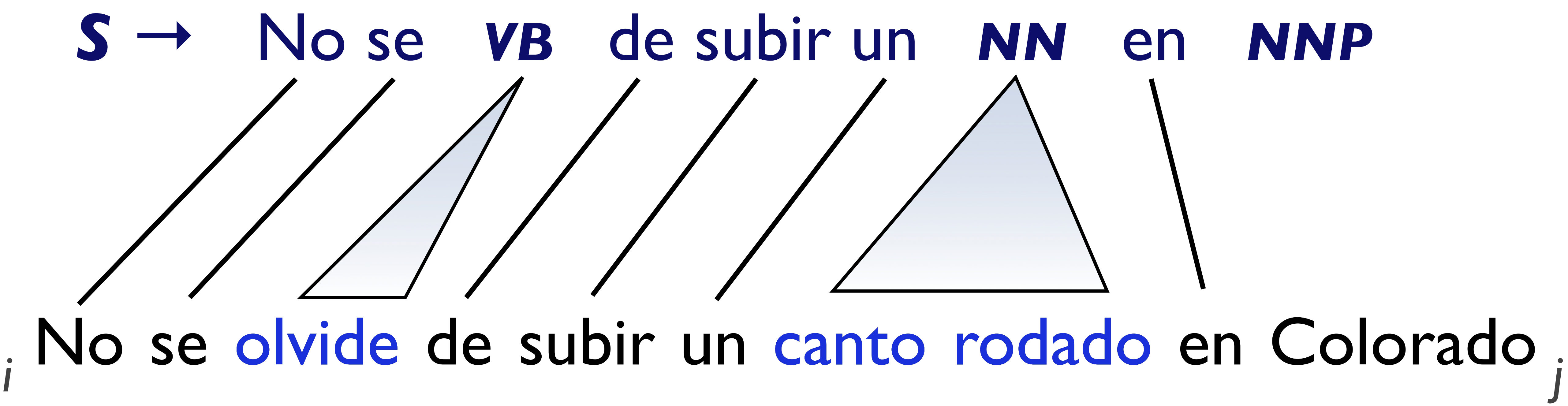


***S*** → No se ***VB*** de subir un ***NN*** en ***NNP***

$_i$ 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]
```

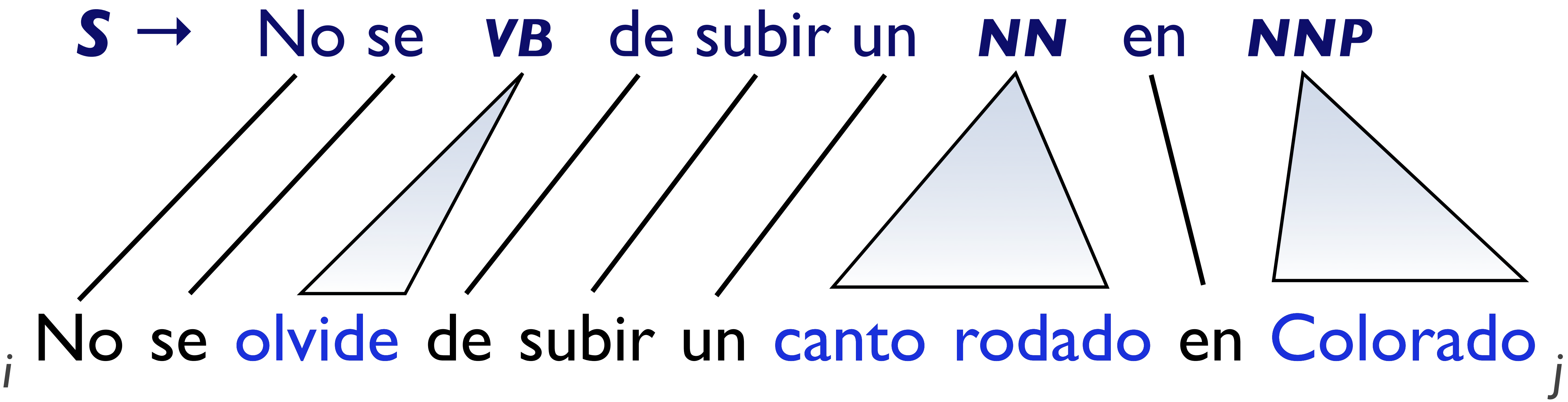


***S*** → No se ***VB*** de subir un ***NN*** en ***NNP***

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

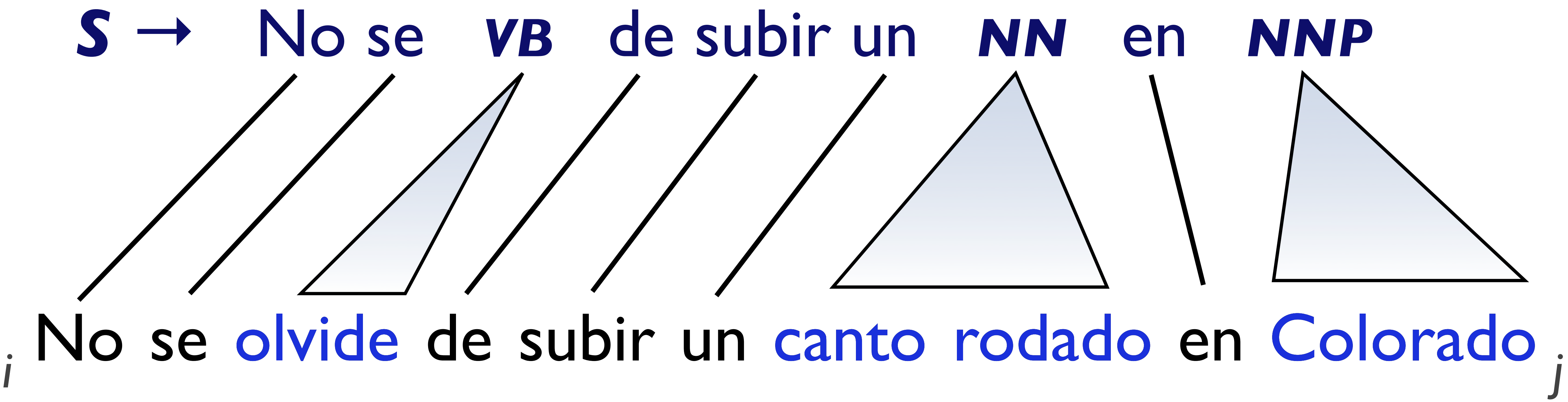
# CKY-style Bottom-up Parsing

```
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]
```



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

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

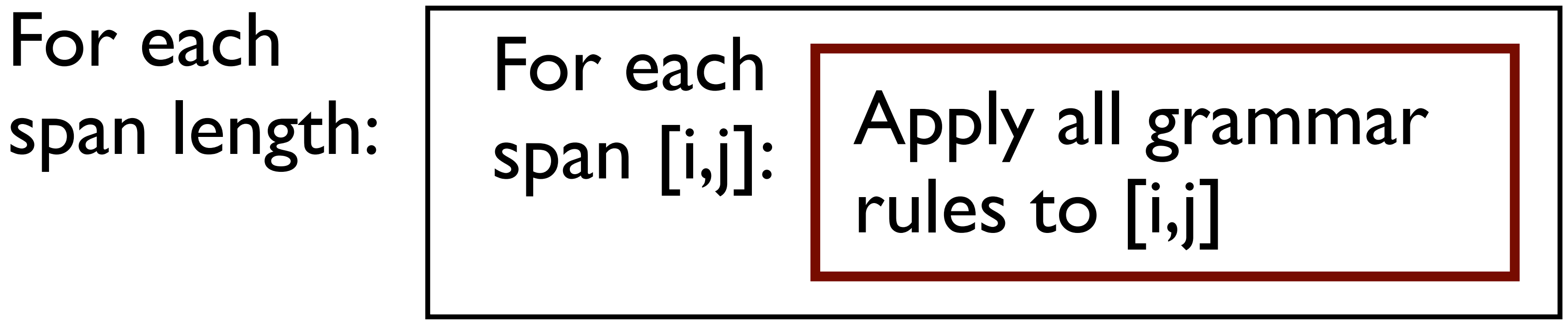
*Many untransformed lexical rules can be applied in linear time*

# 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]:

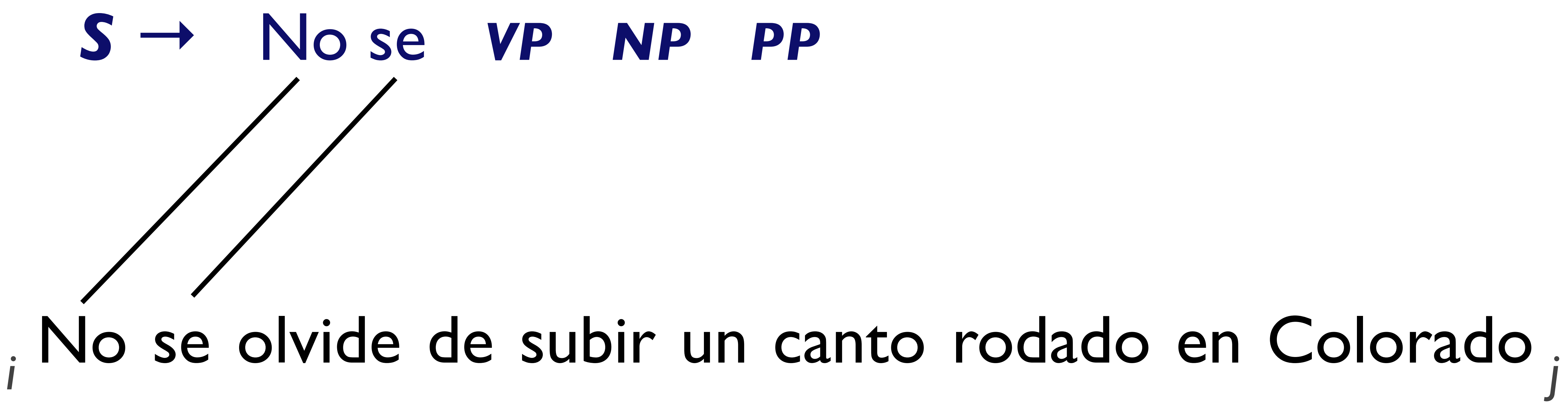
Apply all grammar rules to [i,j]

***S*** → No se ***VP*** ***NP*** ***PP***

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

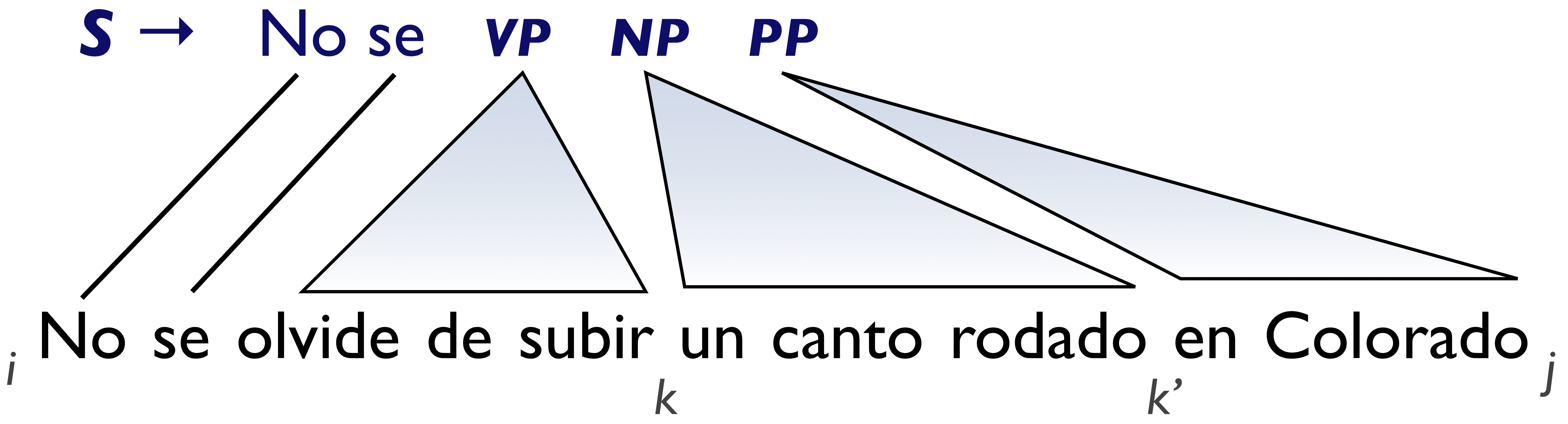
# CKY-style Bottom-up Parsing

```
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]
```

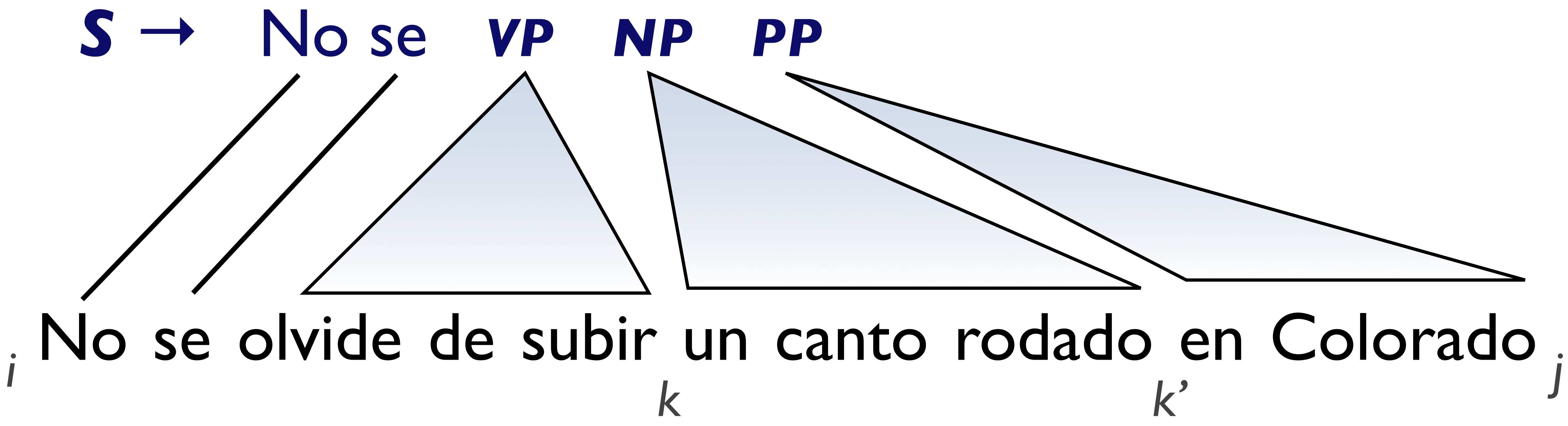


**S** → No se ***VP*** ***NP*** ***PP***

*i* No se olvide de subir *k* un canto rodado *k'* 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***

No se olvide de subir un canto rodado en Colorado

***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*** → No se ***VB*** ***VB*** un ***NN*** ***PP***

Transformed rules: ***S*** → No se ***VB~VB*** un ***NN~PP***

***VB~VB*** → ***VB*** ***VB***

***NN~PP*** → ***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*** → No se ***VB*** ***VB*** un ***NN*** ***PP***

Transformed rules: ***S*** → No se ***VB~VB*** un ***NN~PP***

***VB~VB*** → ***VB*** ***VB***

***NN~PP*** → ***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-*

*(b) all unlexicalized rules are binary.*

SPEED
LIMIT
**68**
sentences
per hour

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

Transformed rules: ***S*** → No se ***VB~VB*** un ***NN~PP***

***VB~VB*** → ***VB*** ***VB***

***NN~PP*** → ***NN*** ***PP***

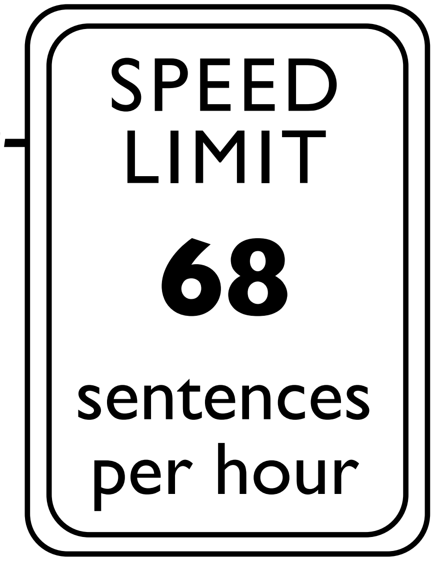
Parsing stages:

- 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-*
*(b) all unlexicalized rules are binary.*



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

Transformed rules: ***S*** → No se ***VB~VB*** un ***NN***

***VB~VB*** → ***VB*** ***VB***

***NN~PP*** → ***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*

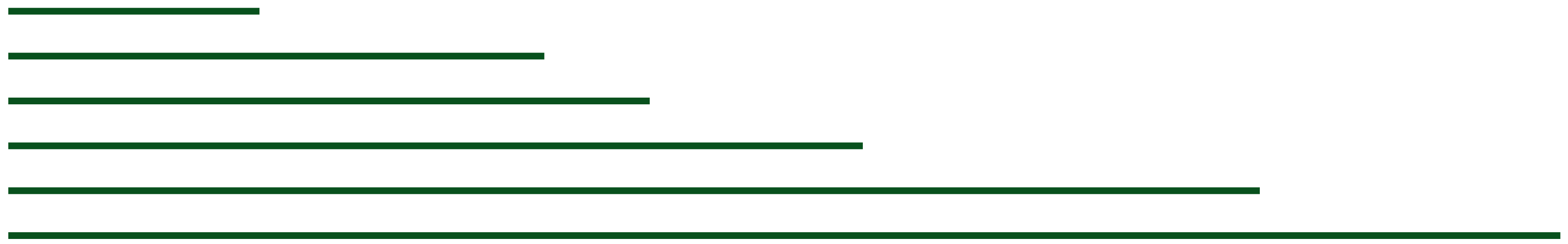
***VP~VP*** → ***VP*** ***VP***

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

# Bounding Split Points

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

***VP∼VP → VP VP***

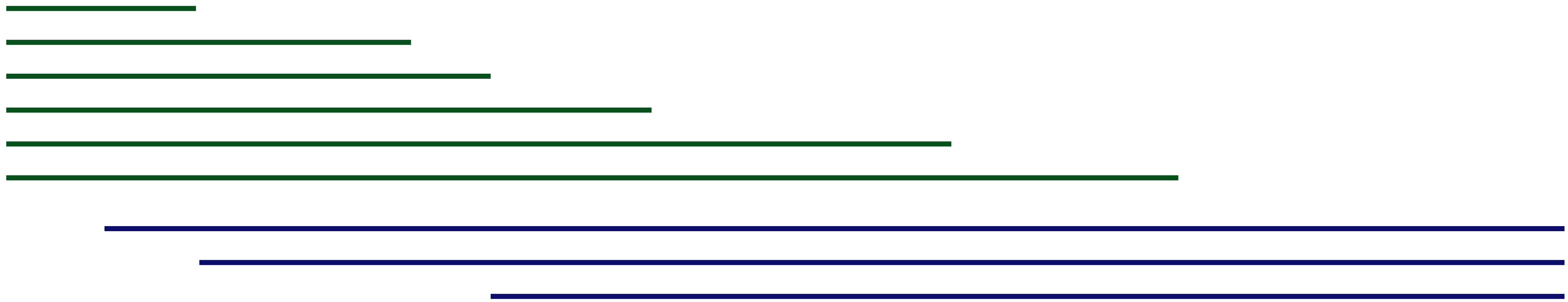


$_i$ No se olvide de subir un canto rodado en Colorado $_j$

# Bounding Split Points

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

***VP∼VP* → *VP VP***

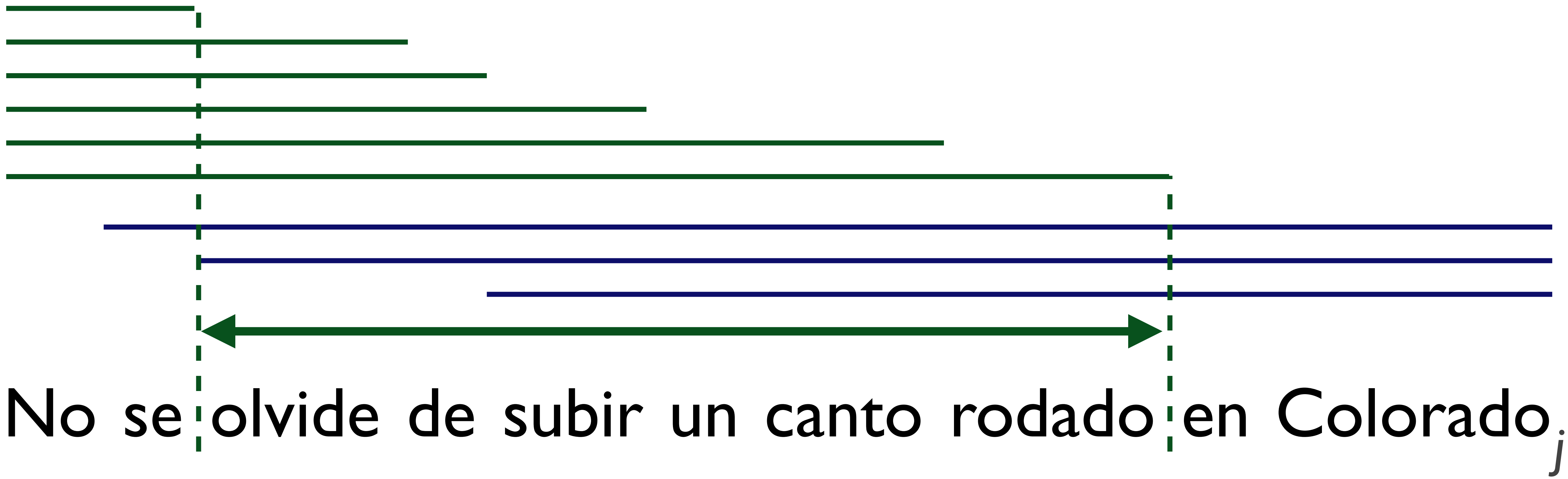


$_i$ No se olvide de subir un canto rodado en Colorado $_j$

# Bounding Split Points

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

***VP∼VP → VP VP***

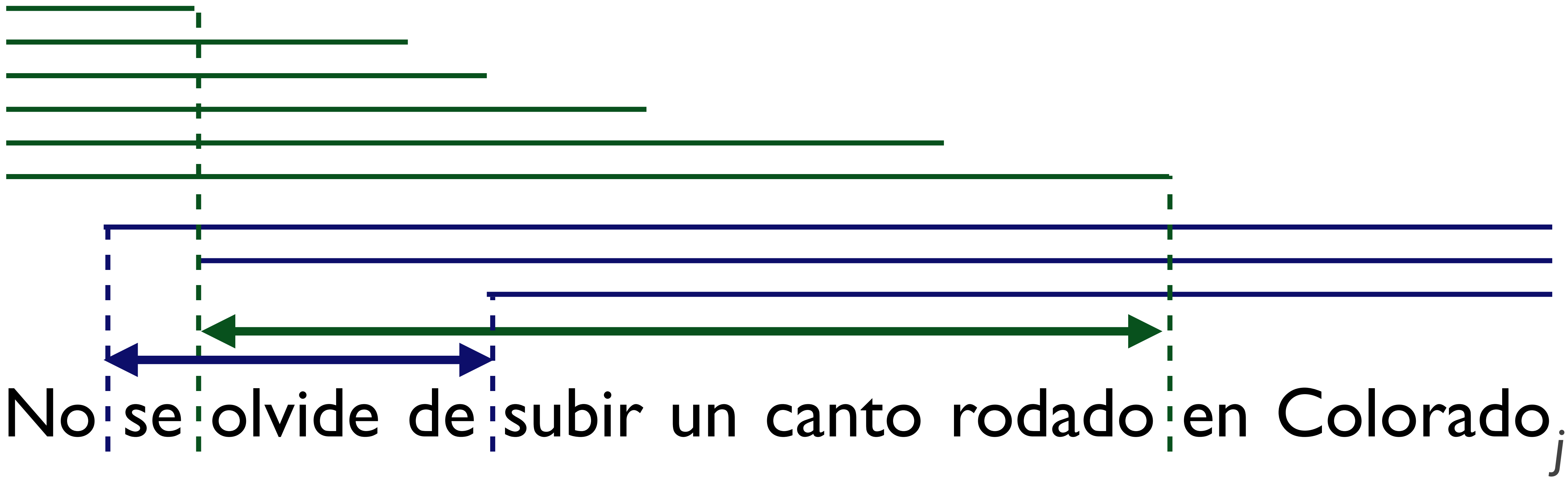


$_i$ No se olvide de subir un canto rodado en Colorado $_j$

# Bounding Split Points

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

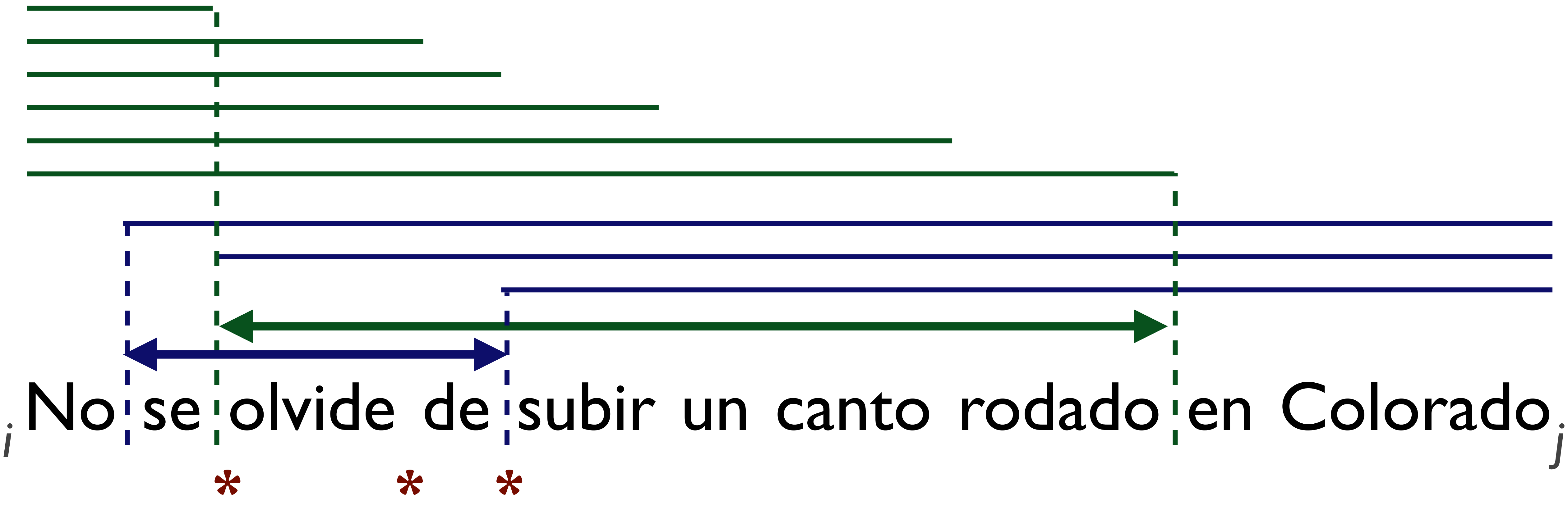
***VP~VP*** → ***VP*** ***VP***



# Bounding Split Points

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

***VP~VP → VP VP***

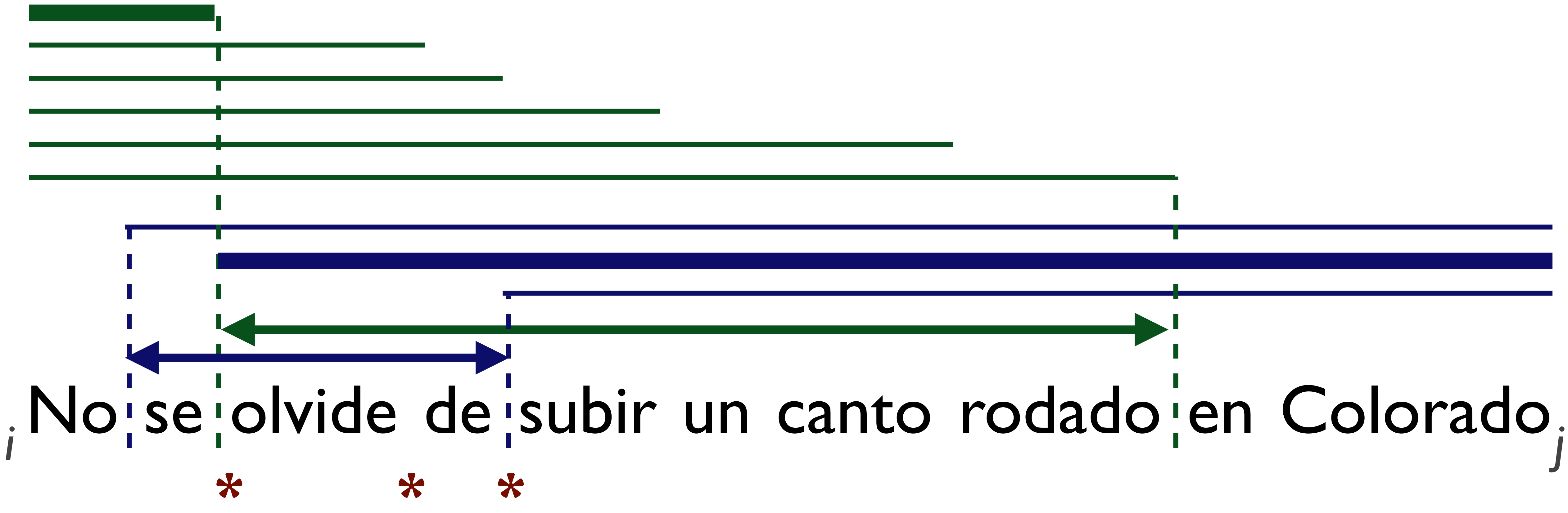


*Only consider split points k that might result in a valid parse*

# Bounding Split Points

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

***VP~VP → VP VP***

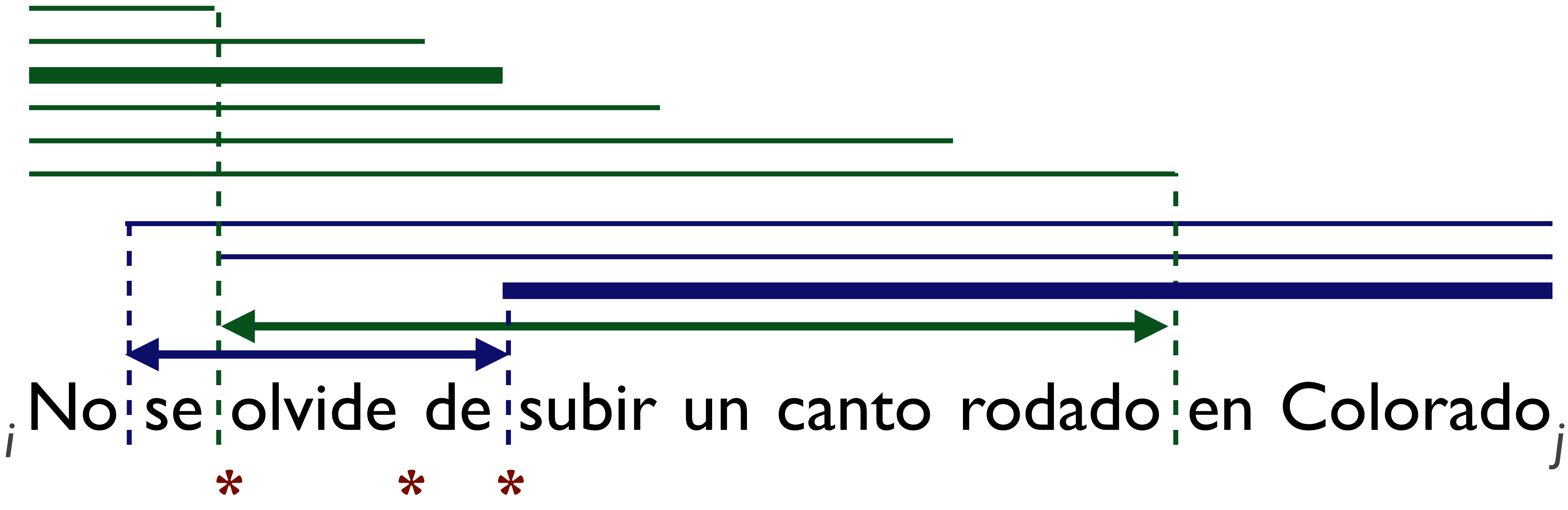


*Only consider split points k that might result in a valid parse*

# Bounding Split Points

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

***VP~VP → VP VP***



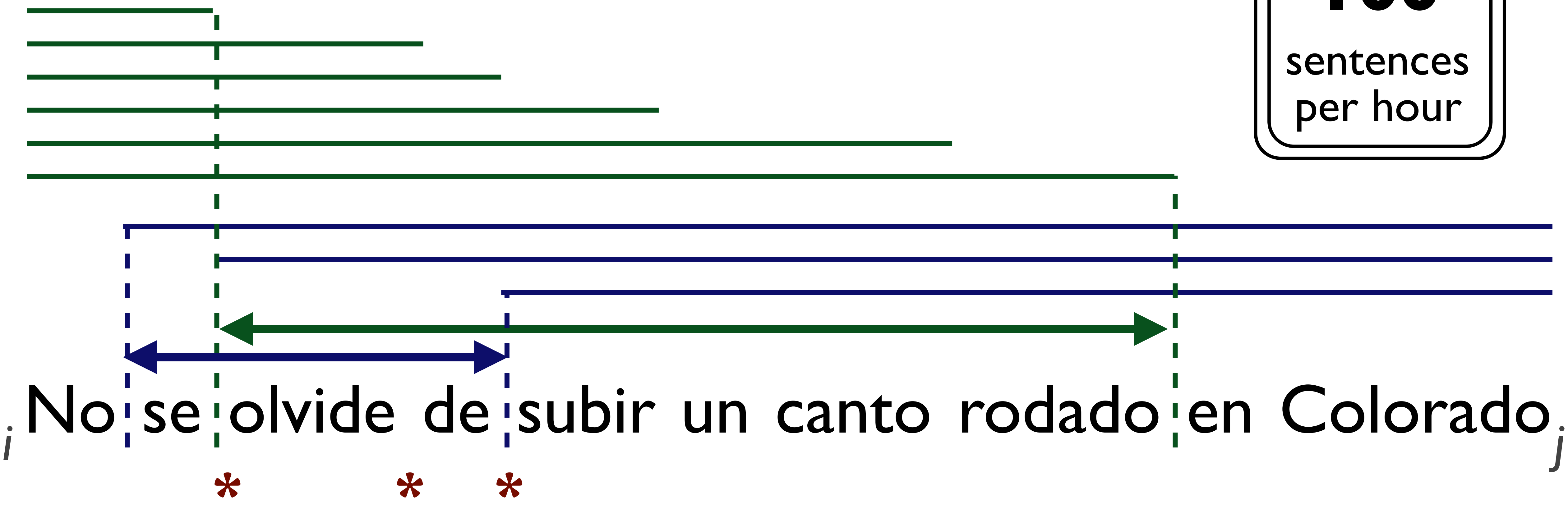
*Only consider split points k that might result in a valid parse*

# Bounding Split Points

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

***VP~VP → VP VP***

SPEED
LIMIT
**100**
sentences
per hour



*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

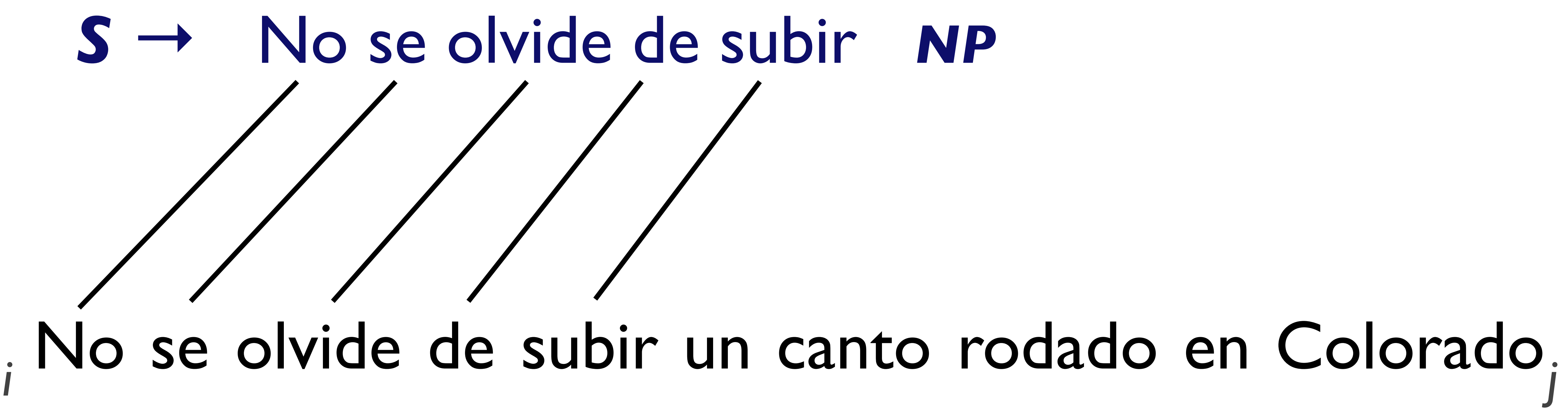
***Problem:*** *Lexical rules can apply to many spans*

***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*

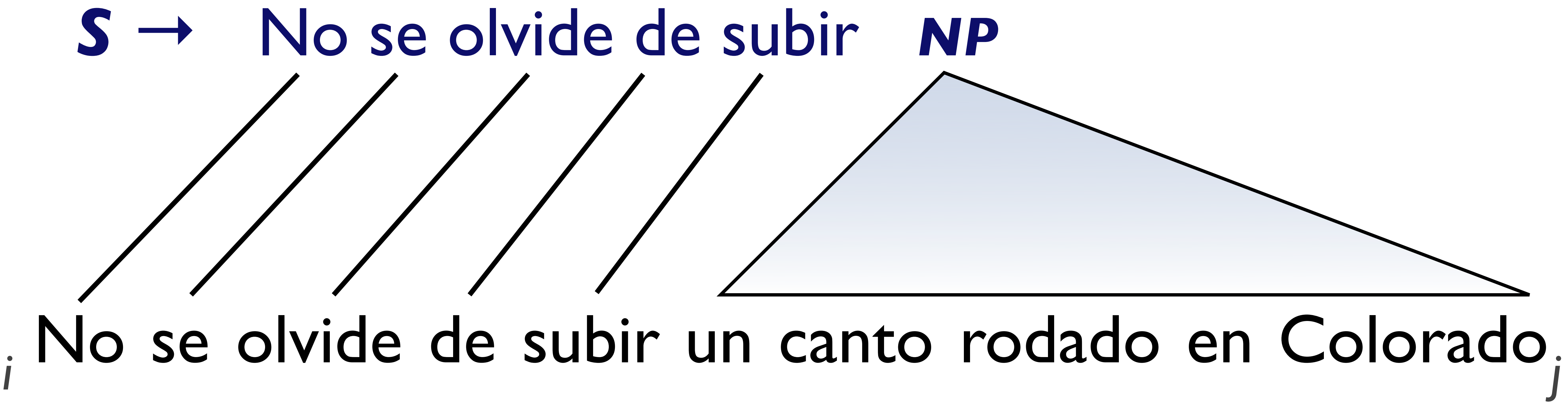


***S*** → No se olvide de subir ***NP***

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

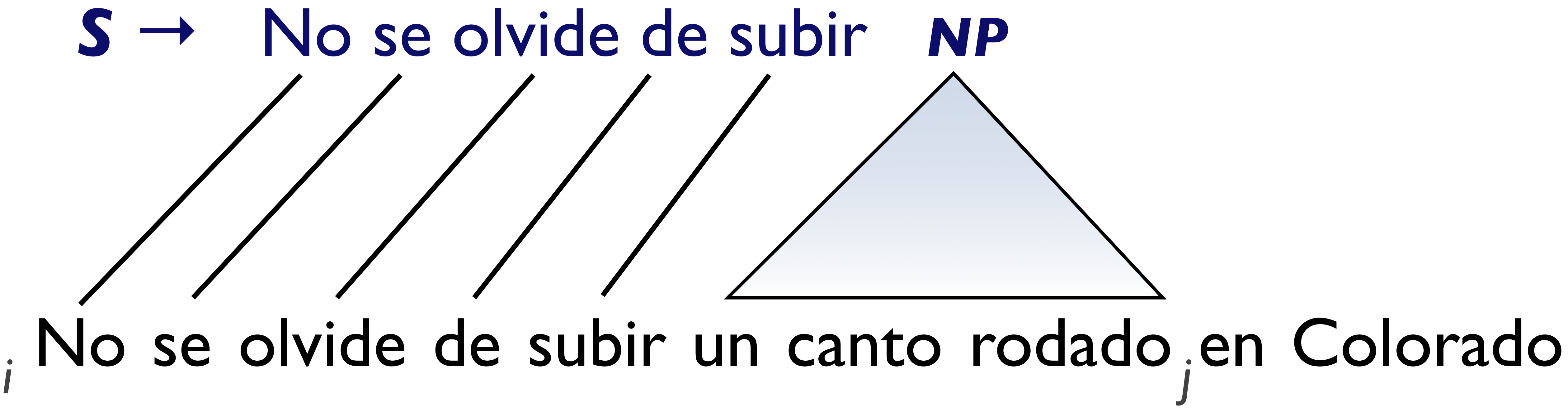
# Speeding up Lexical Rule Application

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


S → No se olvide de subir NP
No se olvide de subir un canto rodado en Colorado
i
j

# Speeding up Lexical Rule Application

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

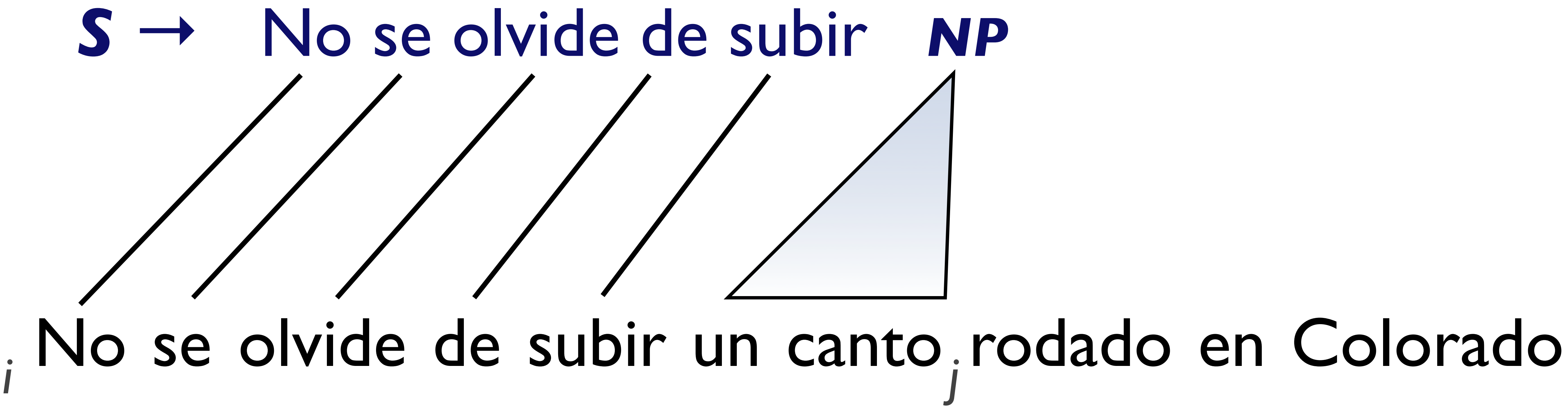


***S*** → No se olvide de subir ***NP***

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

# 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 $_j$ rodado en Colorado

# 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*** → No se ***VB*** ***VB*** un ***NN*** ***PP***

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

Transformed rules: ***S*** → ***S/NN~PP*** ***NN~PP***

***S/NN~PP*** → No se ***VB~VB*** 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*** → No se ***VB*** ***VB*** un ***NN*** ***PP***

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

Transformed rules: ***S*** → ***S/NN~PP*** ***NN~PP***

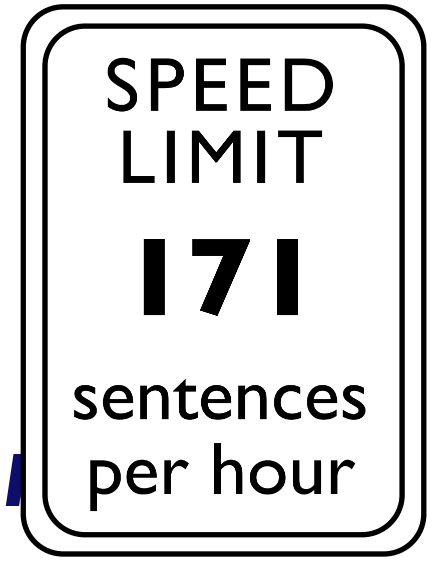
***S/NN~PP*** → No se ***VB~VB*** un

*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** → No se **VB** **VB** un

LNF rule: **S** → No se **VB~VB** un

Transformed rules: **S** → **S/NN~PP** **NN~PP**

**S/NN~PP** → No se **VB~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*** → ***VB NP NP PP***

**Binarization options:**

***S*** → ***VB~NP~NP PP***

***S*** → ***VB NP~NP~PP***

***S*** → ***VB~NP NP~PP***

# Binarizing Sequences of Non-Terminals

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

**Original:**

***S → VB NP NP PP***

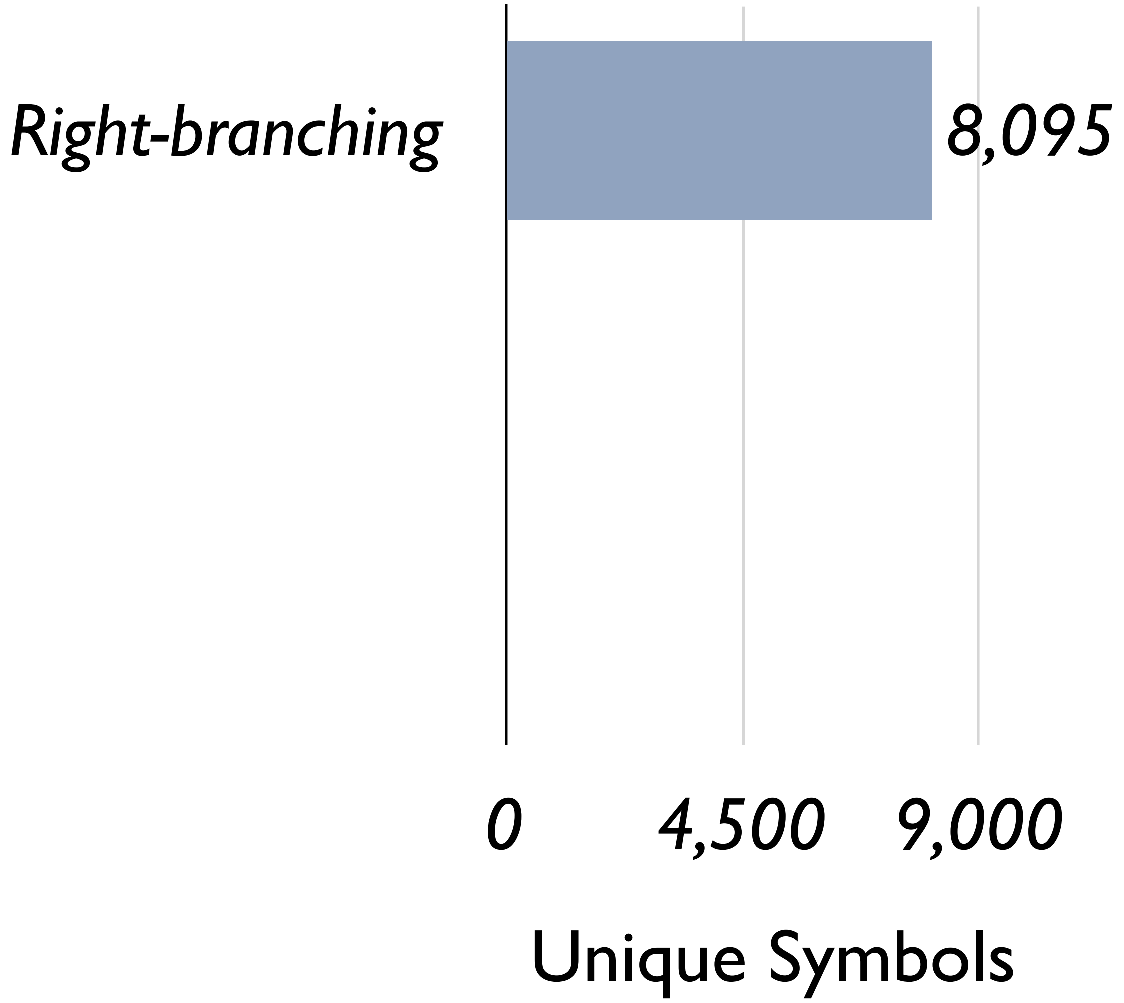
**Binarization options:**

***S → VB~NP~NP PP***

***S → VB NP~NP~PP***

***S → VB~NP NP~PP***

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 → VB NP NP PP***

**Binarization options:**

***S → VB~NP~NP PP***

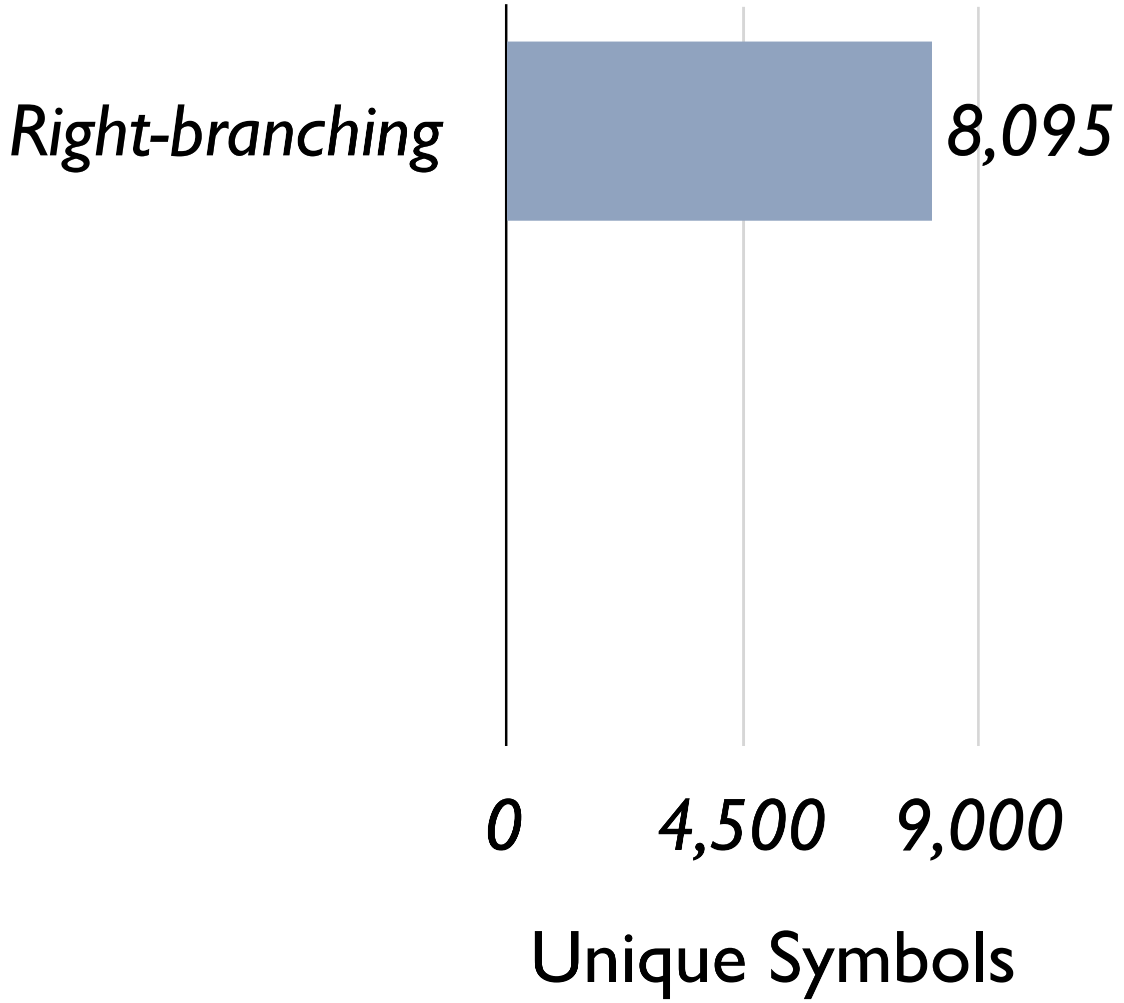
***S → VB NP~NP~PP***

***S → VB~NP NP~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 → VB NP NP PP**

**Binarization options:**

**S → VB~NP~NP PP**

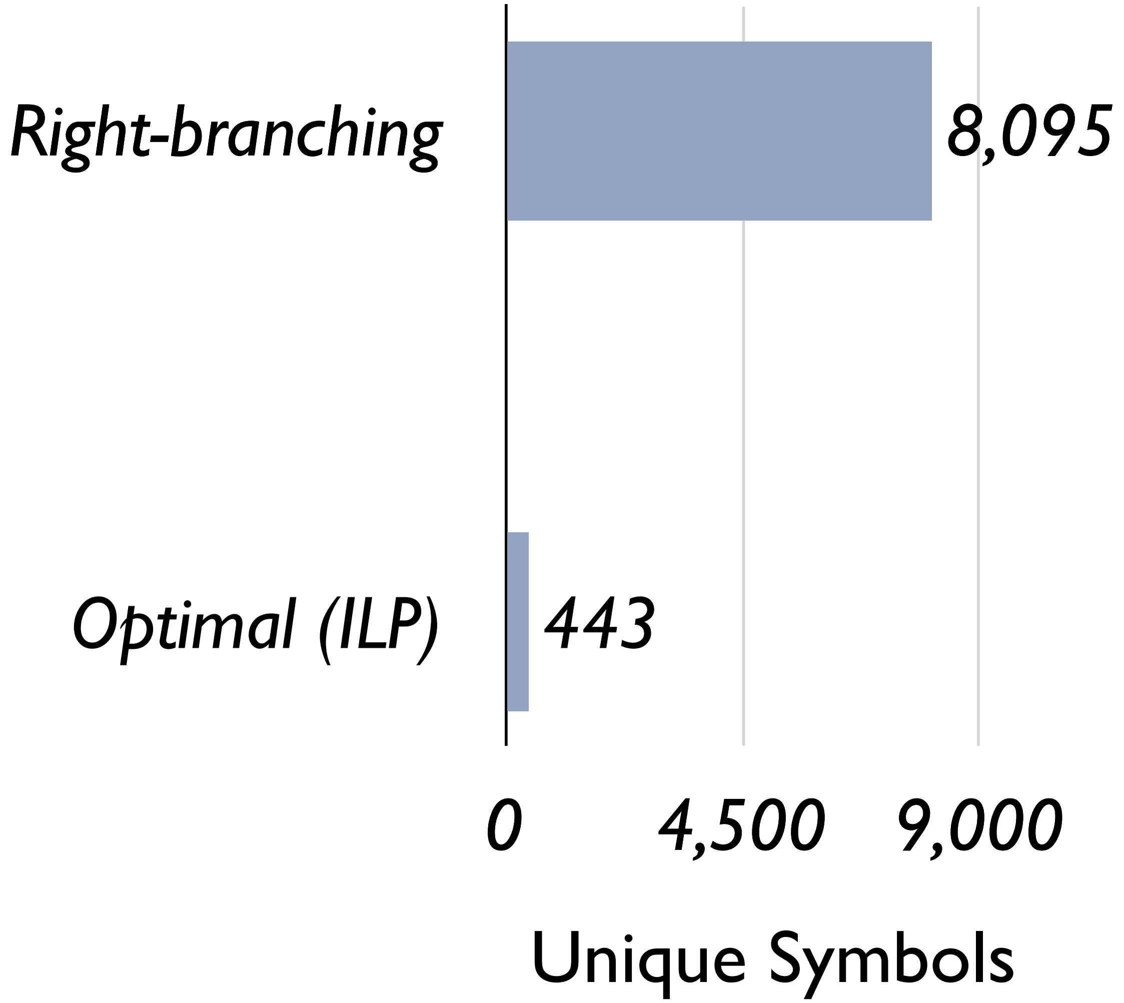
**S → VB NP~NP~PP**

**S → VB~NP NP~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 → VB NP NP PP***

**Binarization options:**

***S → VB~NP~NP PP***

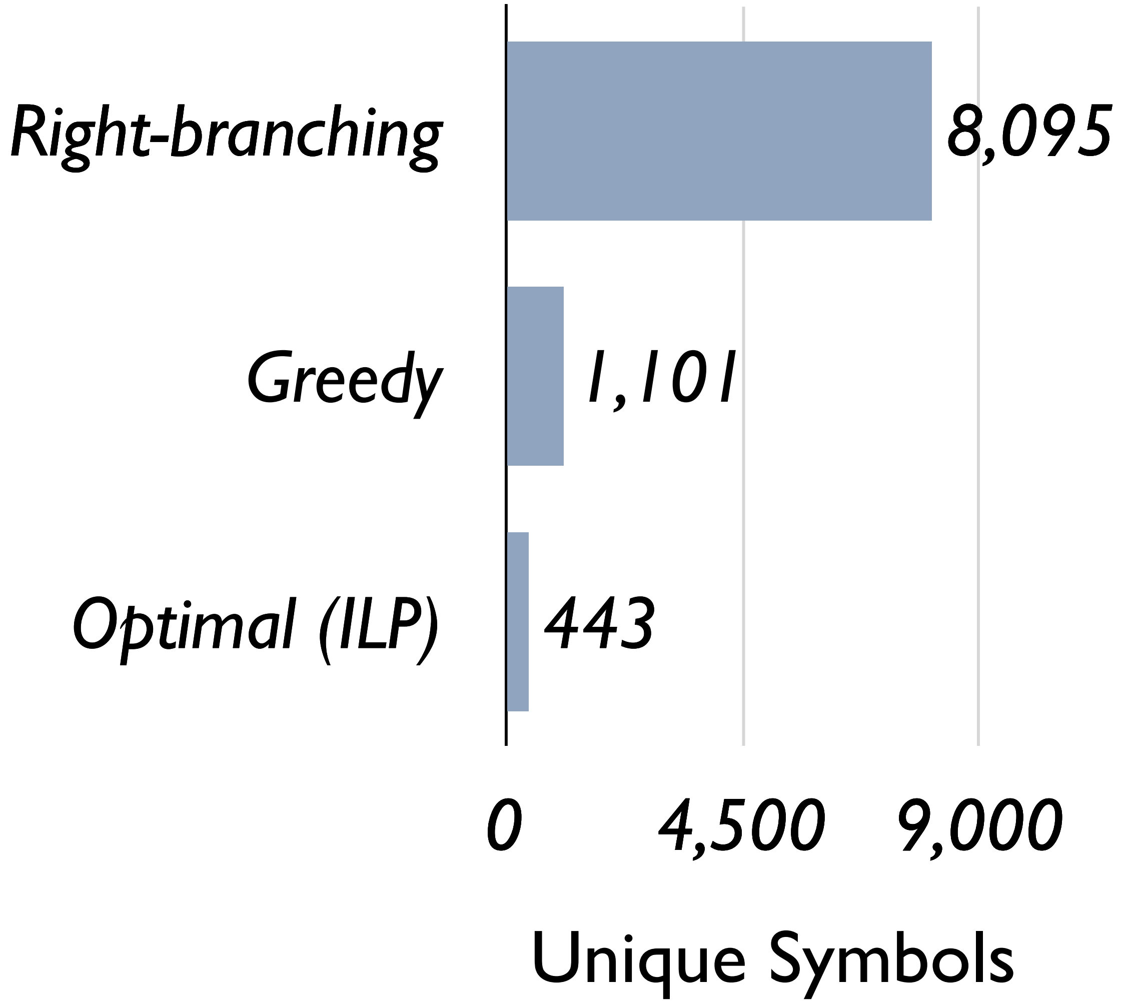
***S → VB NP~NP~PP***

***S → VB~NP NP~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



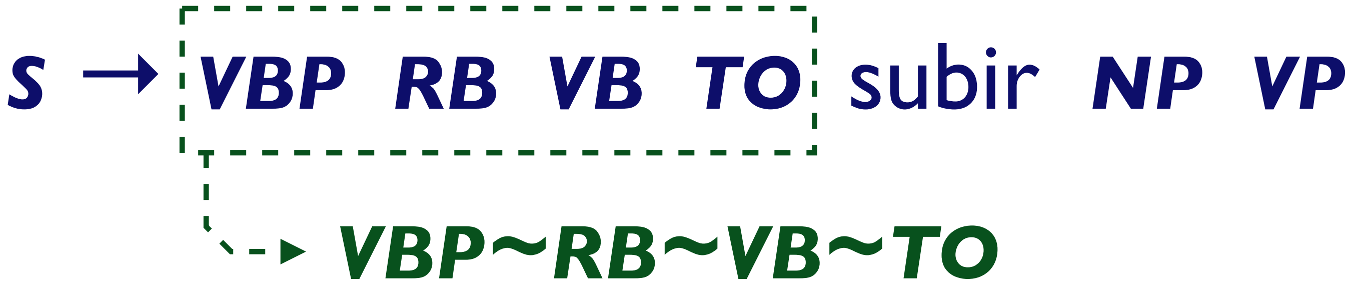
# Subsets of Grammar Symbols

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

***S*** → ***VBP RB VB TO*** subir ***NP VP***

# Subsets of Grammar Symbols

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

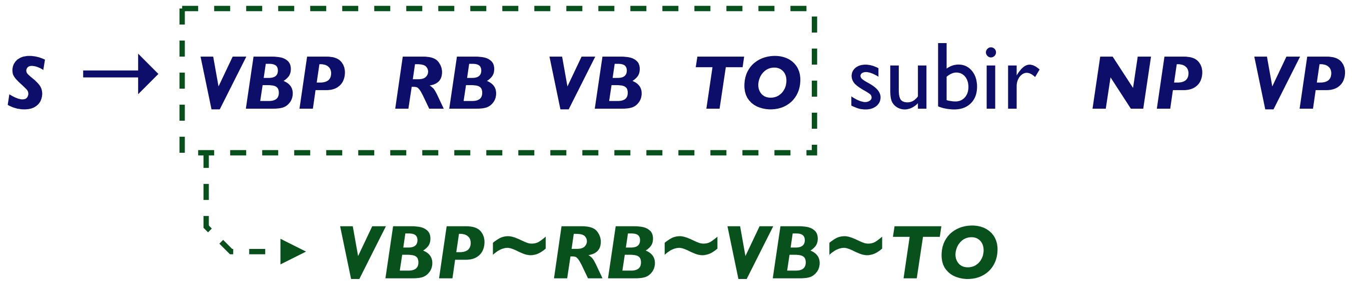


***S*** → ***VBP RB VB TO*** subir ***NP VP***

***VBP~RB~VB~TO***

# Subsets of Grammar Symbols

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



***S*** → ***VBP RB VB TO*** subir ***NP VP***

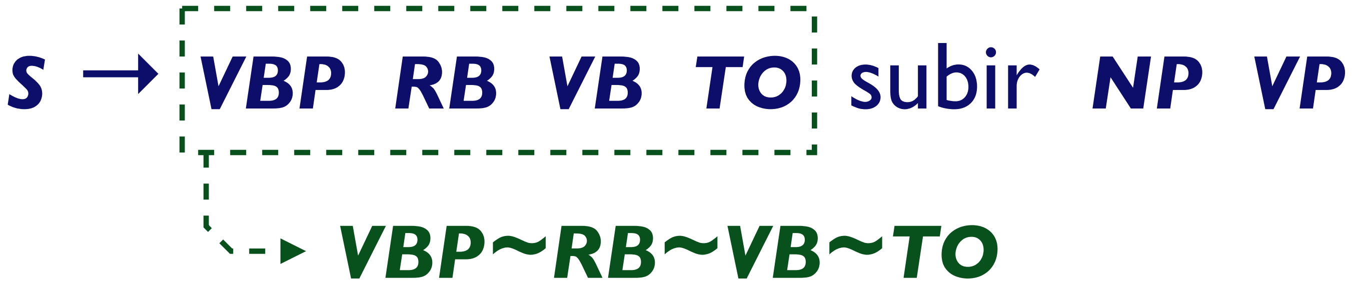
***VBP~RB~VB~TO***

***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

# Subsets of Grammar Symbols

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



***S*** → ***VBP RB VB TO*** subir ***NP VP***

***VBP~RB~VB~TO***

SPEED
LIMIT

**327**

sentences
per hour

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

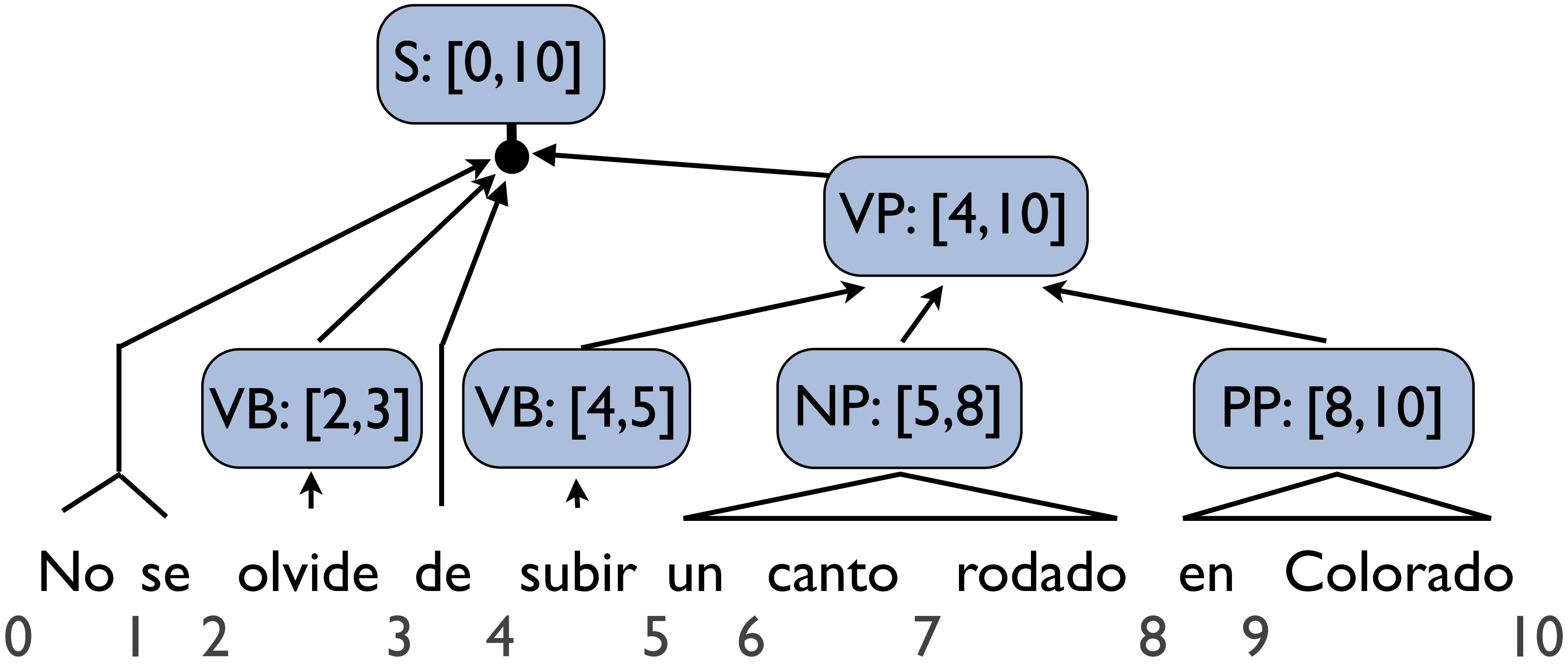
1. Choose a small subset of high-use symbols
2. Parse with the coarse grammar and prune unlikely states
3. Parse with the fine grammar in the pruned search space

# Integrating a Language Model

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

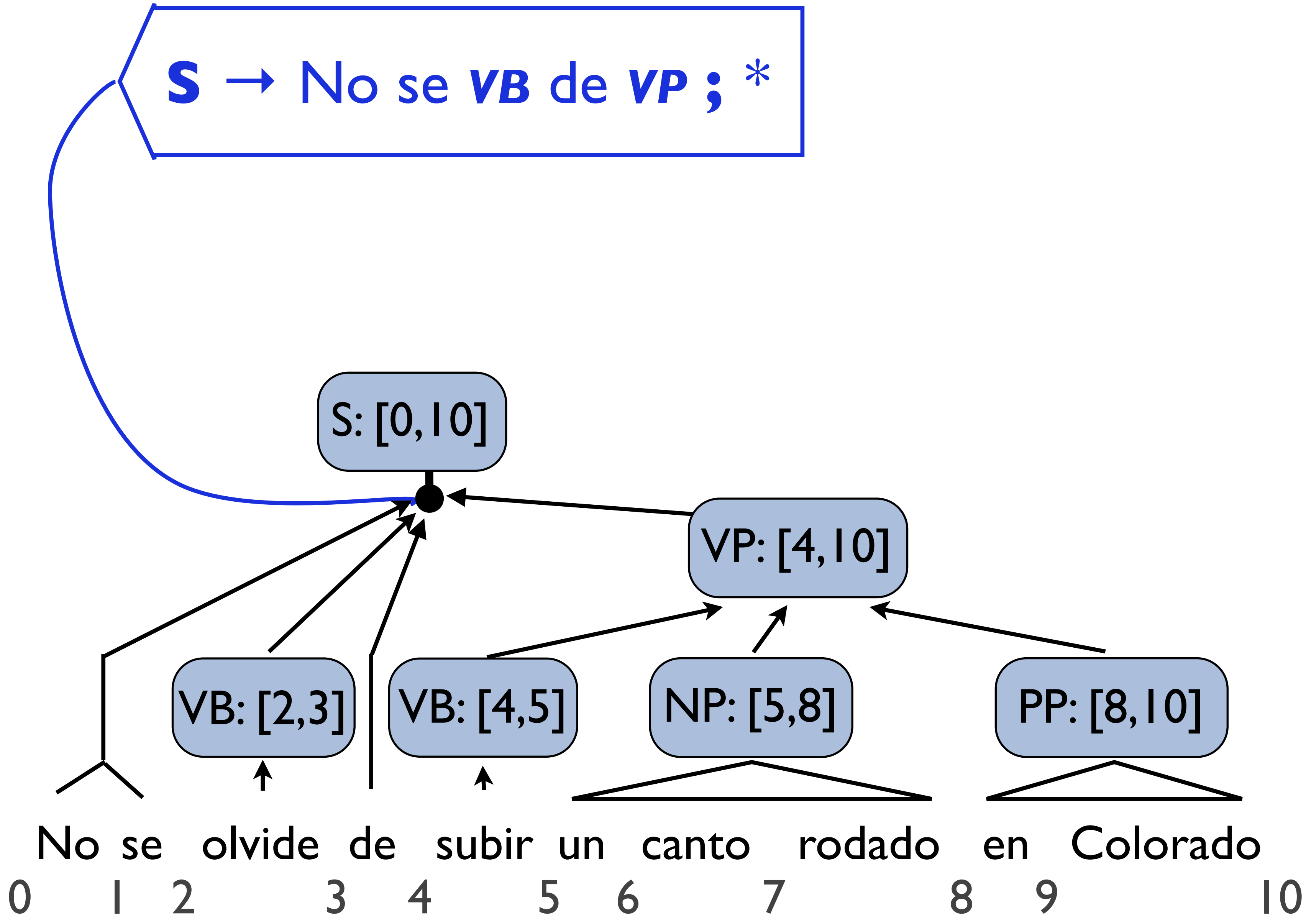
# Integrating a Language Model

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



# Integrating a Language Model

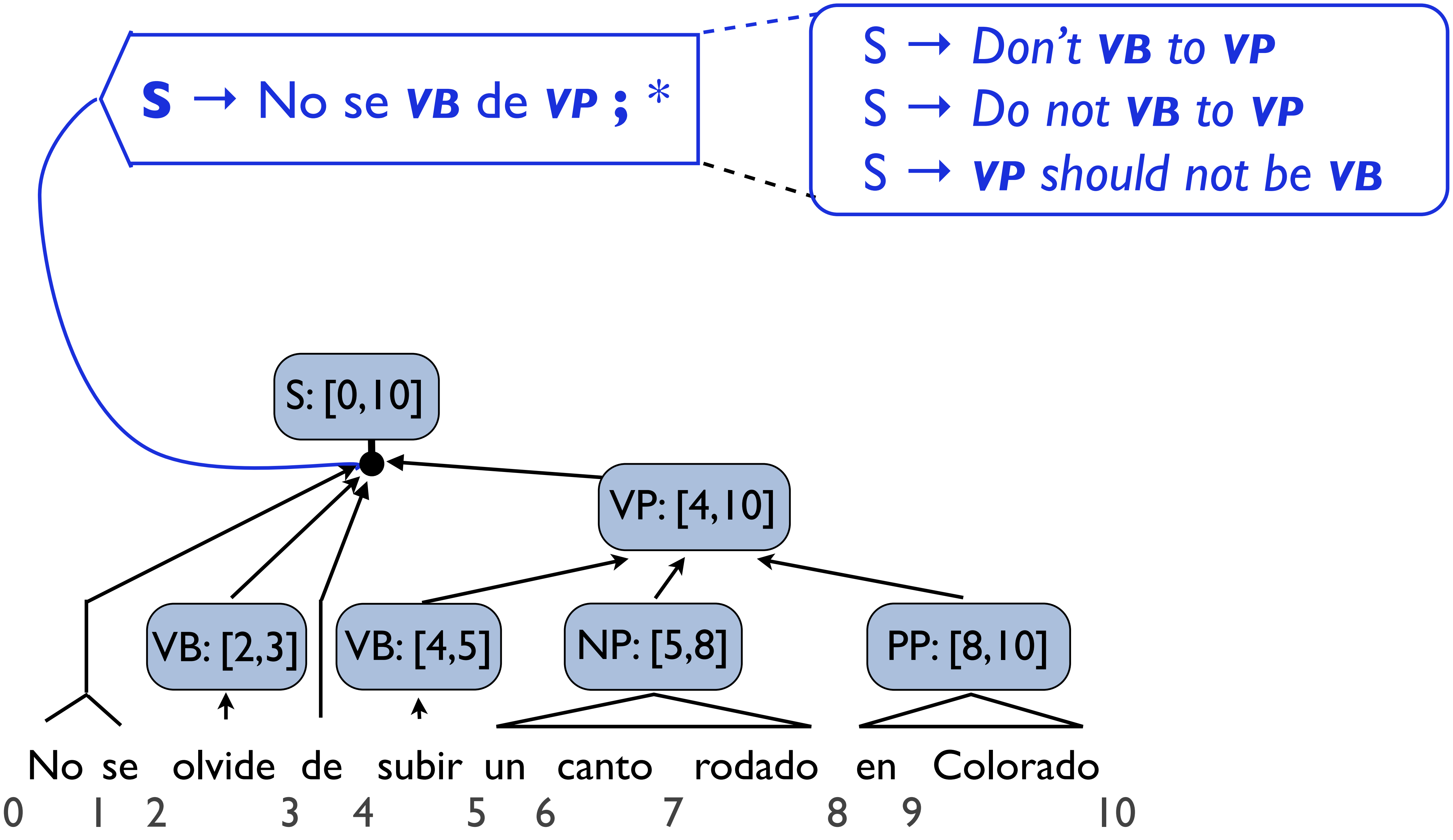
***Approach:*** *Top-down lazy forest reranking with priority queues (a.k.a., cube growing)*



**S** → No se ***VB*** de ***VP*** **;** *
S: [0,10]
VP: [4,10]
VB: [2,3]
VB: [4,5]
NP: [5,8]
PP: [8,10]
No se olvide de subir un canto rodado en Colorado
0 1 2 3 4 5 6 7 8 9 10

# Integrating a Language Model

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



**S** → No se ***VB*** de ***VP*** **;** *

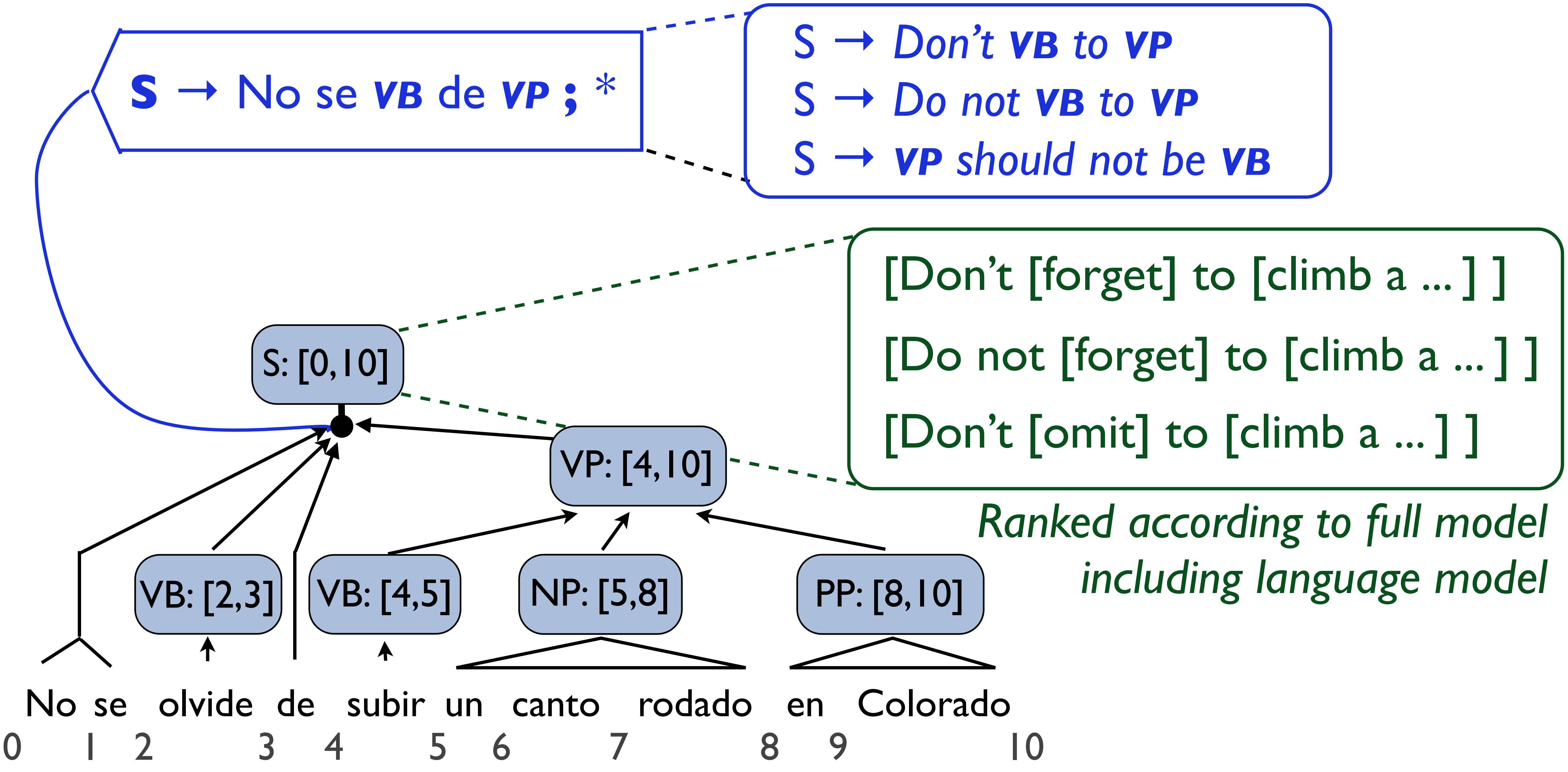
S → *Don’t **VB** to **VP***

S → *Do not **VB** to **VP***

S → ***VP** should not be **VB***

# Integrating a Language Model

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



**S** → No se ***VB*** de ***VP*** **;** *

S → *Don't* ***VB*** *to* ***VP***

S → *Do not* ***VB*** *to* ***VP***

S → ***VP*** *should not be* ***VB***

[Don't [forget] to [climb a ... ] ]

[Do not [forget] to [climb a ... ] ]

[Don't [omit] to [climb a ... ] ]

*Ranked according to full model including language model*

# Coarse-to-Fine LM Integration

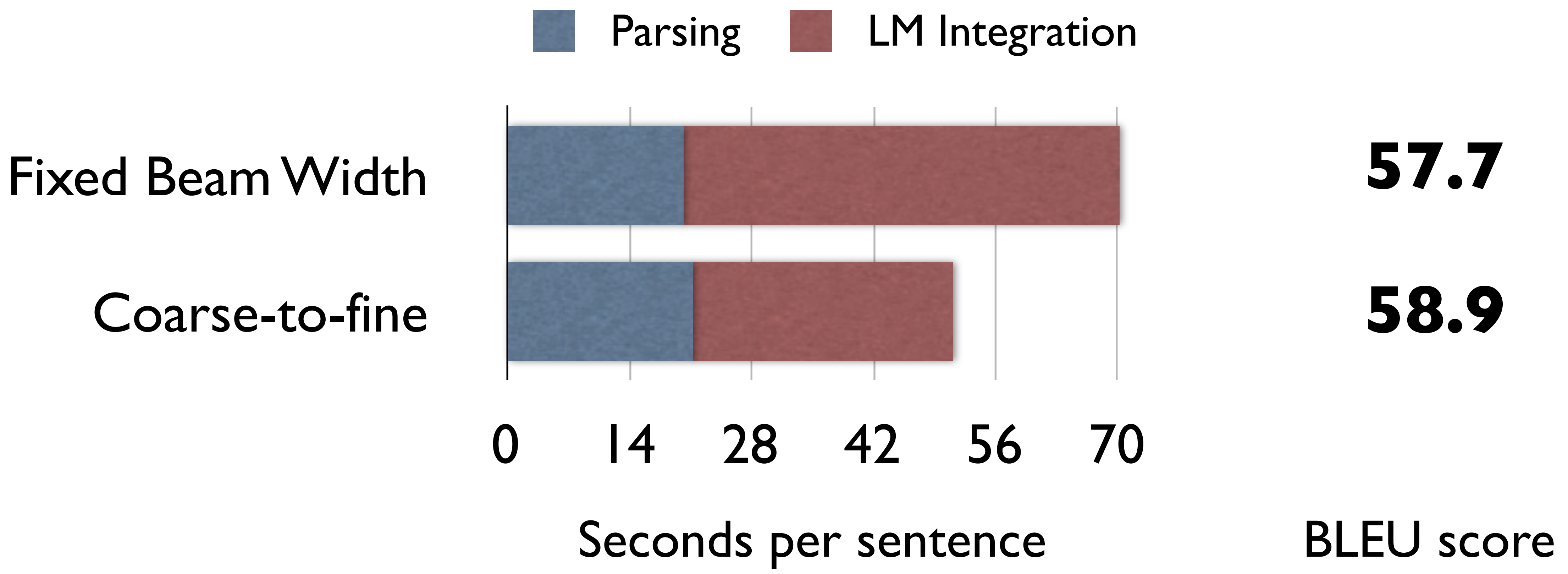
**Observation:** *The best translations almost always have a translation model score close to the Viterbi parse score*

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

# 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

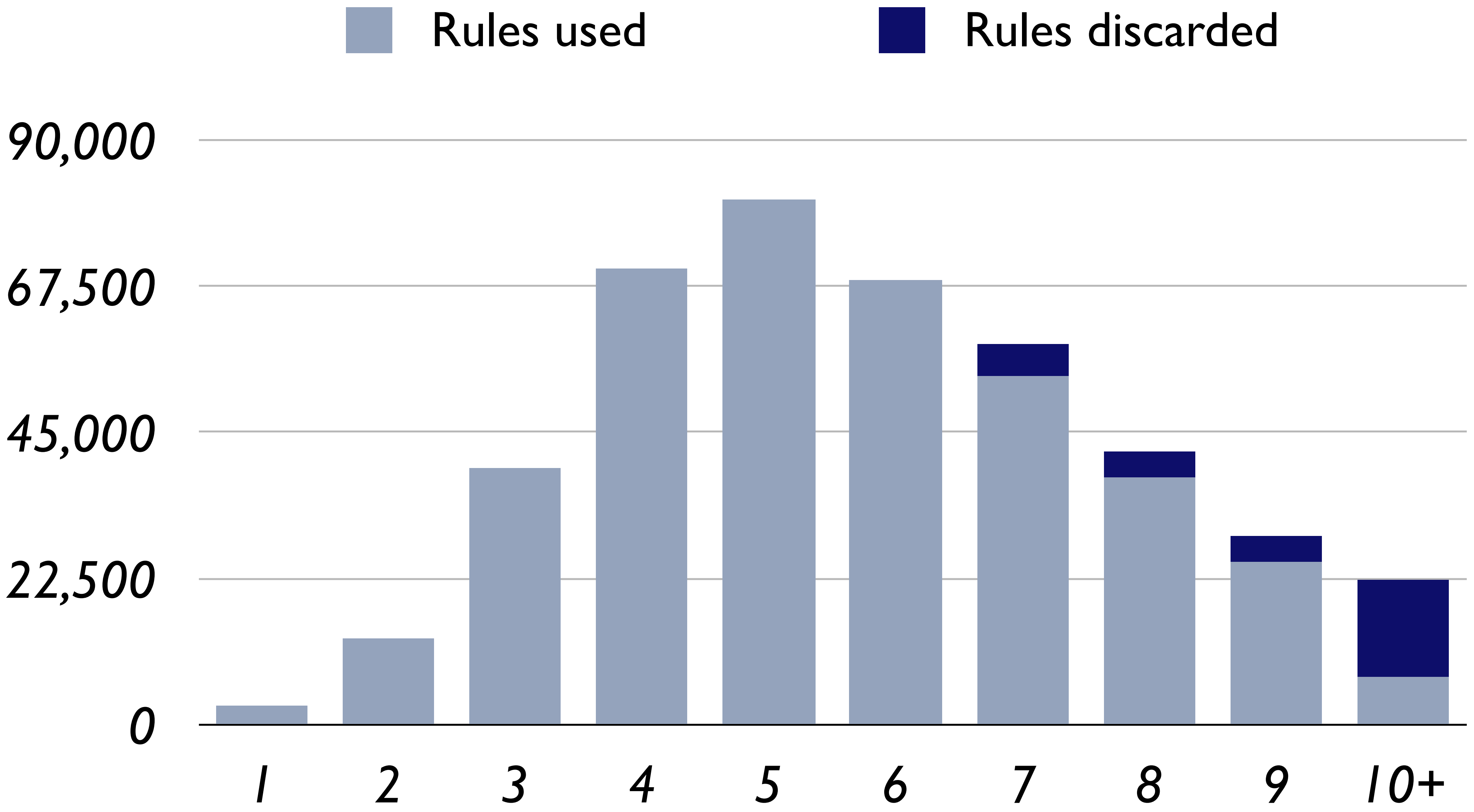
# Thanks!

Questions?

# The Size of Tree Transducer Grammars

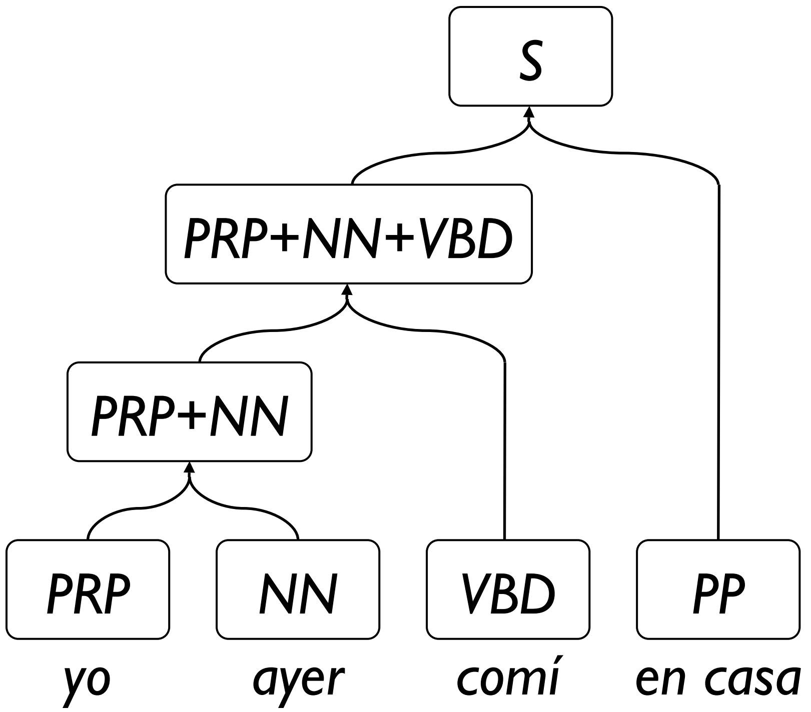
🟦 Rules used     ■ Rules discarded

# The Size of Tree Transducer Grammars



# Rebinarizing for LM Integration (ACL ’09)

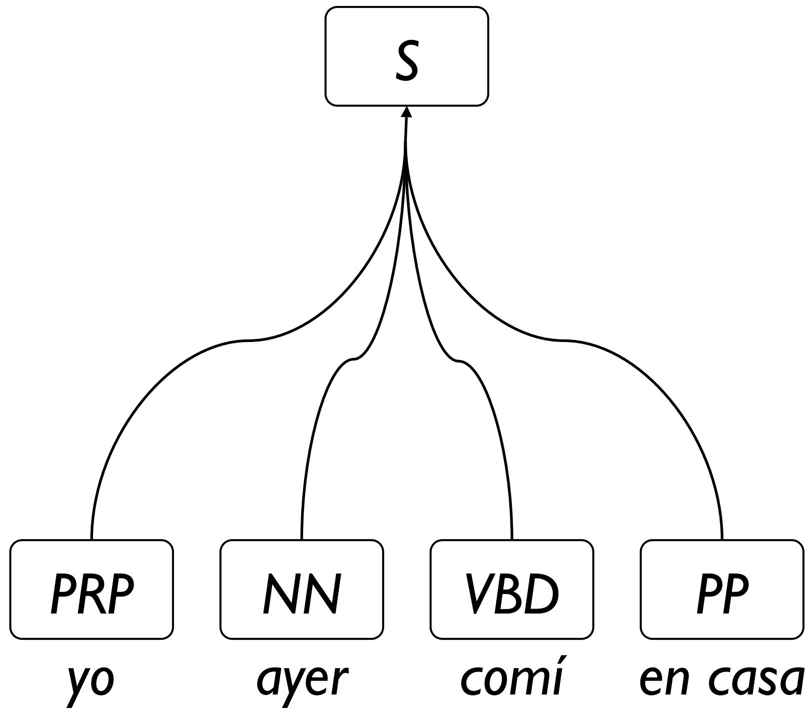
## *Parse with ALNF grammar*



S → PRP$_1$ VBD$_3$ NN$_2$ PP$_4$
[[PRP$_1$ NN$_2$] VBD$_3$] PP$_4$

S → PRP$_1$ VBD$_3$ PP$_4$ NN$_2$
[[PRP$_1$ NN$_2$] VBD$_3$] PP$_4$

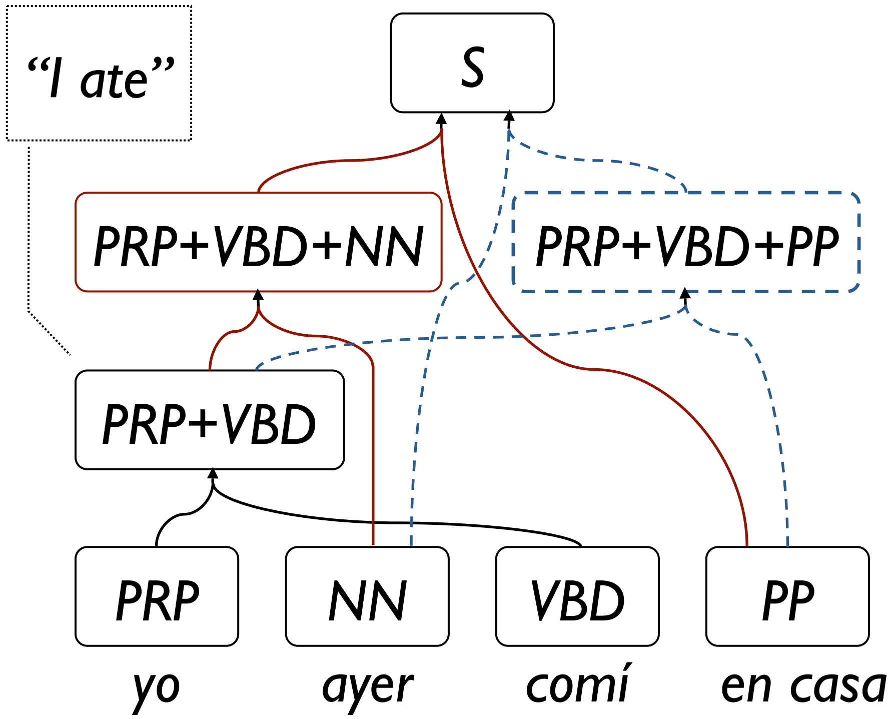
## *Collapse out binarization*



S → PRP$_1$ VBD$_3$ NN$_2$ PP$_4$
PRP$_1$ NN$_2$ VBD$_3$ PP$_4$

S → PRP$_1$ VBD$_3$ PP$_4$ NN$_2$
PRP$_1$ NN$_2$ VBD$_3$ PP$_4$

## *Rebinarize for LM integration*



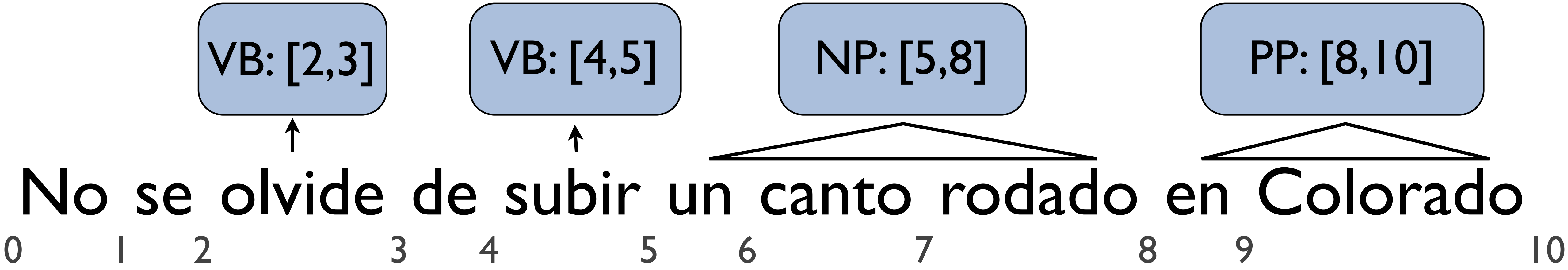
S → [[PRP$_1$ VBD$_3$] NN$_2$] PP$_4$
PRP$_1$ NN$_2$ VBD$_3$ PP$_4$

S → [[PRP$_1$ VBD$_3$] PP$_4$] NN$_2$
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



VB: [2,3]

VB: [4,5]

NP: [5,8]

PP: [8,10]

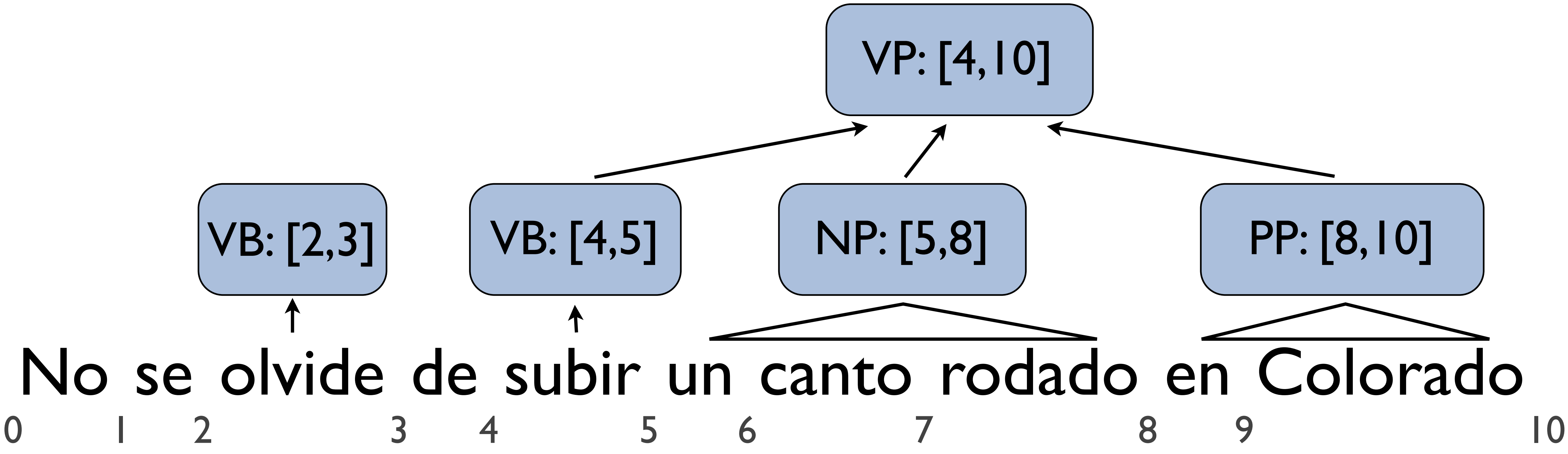
No se olvide de subir un canto rodado en Colorado

0 1 2 3 4 5 6 7 8 9 10

# Multi-Pass Syntactic Decoding

Parse input sentence with source-side grammar projection

Rerank derivations rooted at each state with a language model



VP: [4,10]

VB: [2,3]

VB: [4,5]

NP: [5,8]

PP: [8,10]

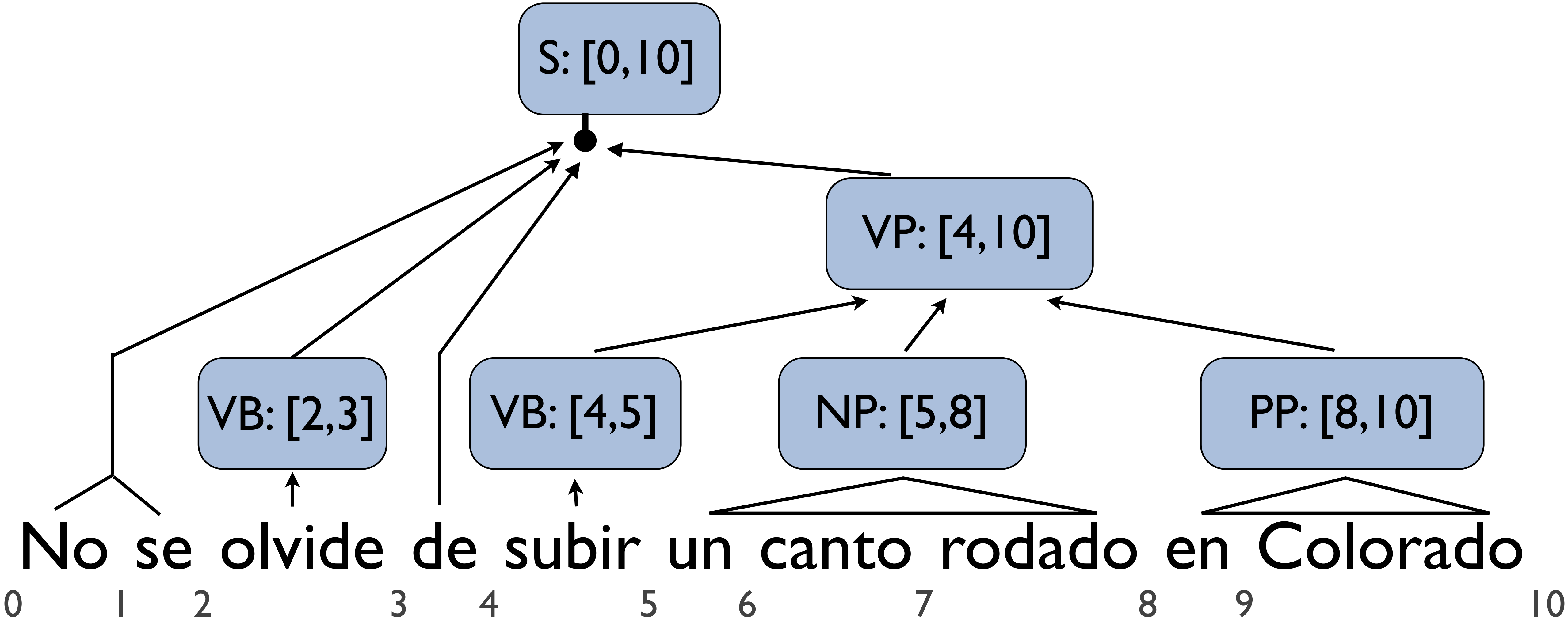
No se olvide de subir un canto rodado en Colorado

0 1 2 3 4 5 6 7 8 9 10

# Multi-Pass Syntactic Decoding

Parse input sentence with source-side grammar projection

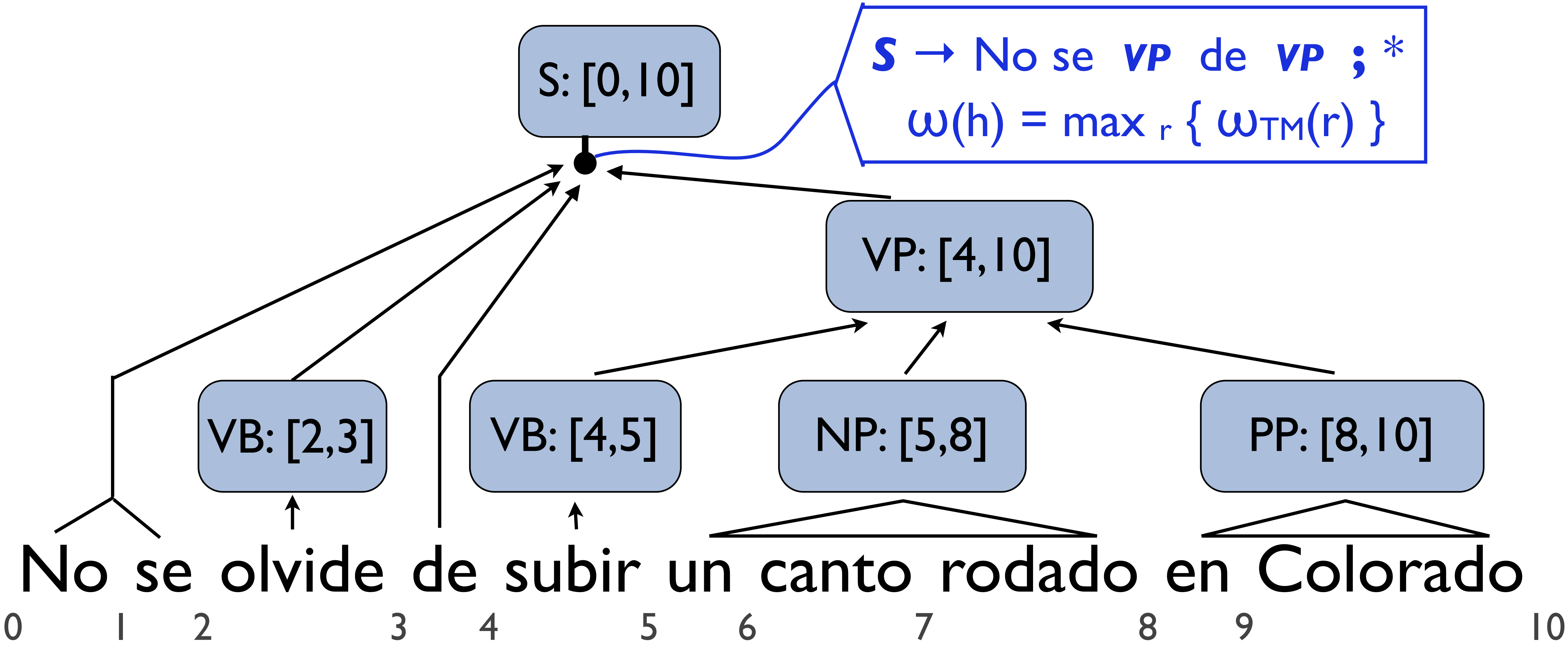
Rerank derivations rooted at each state with a language model



# Multi-Pass Syntactic Decoding

Parse input sentence with source-side grammar projection

Rerank derivations rooted at each state with a language model



***S*** → No se ***VP*** de ***VP*** **;** *

$\omega(h) = \max_r \{ \omega_{TM}(r) \}$

S: [0,10]

VP: [4,10]

VB: [2,3]

VB: [4,5]

NP: [5,8]

PP: [8,10]

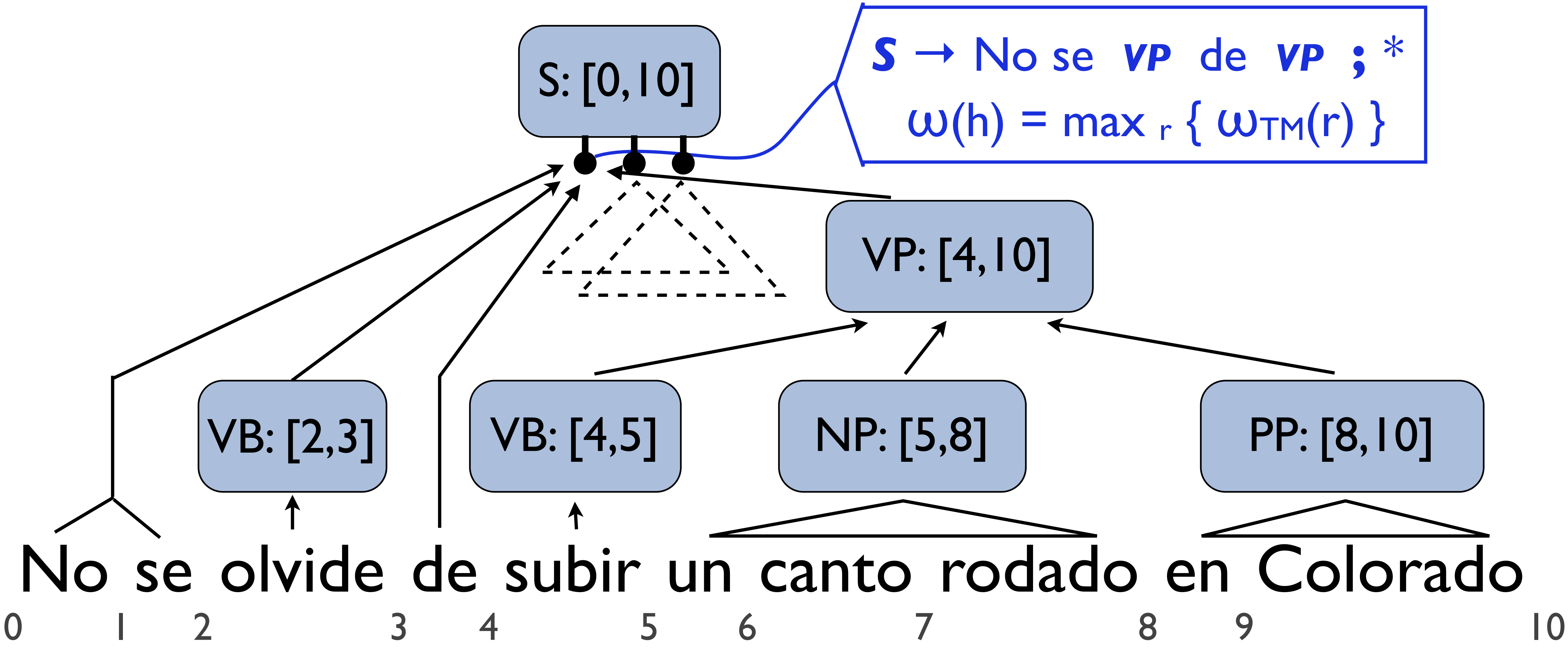
No se olvide de subir un canto rodado en Colorado

0 1 2 3 4 5 6 7 8 9 10

# Multi-Pass Syntactic Decoding

Parse input sentence with source-side grammar projection

Rerank derivations rooted at each state with a language model



$\boldsymbol{S} \rightarrow$ No se $\boldsymbol{VP}$ de $\boldsymbol{VP}$ ; *

$\omega(h) = \max_r \{ \omega_{TM}(r) \}$

S: [0,10]

VP: [4,10]

VB: [2,3]

VB: [4,5]

NP: [5,8]

PP: [8,10]

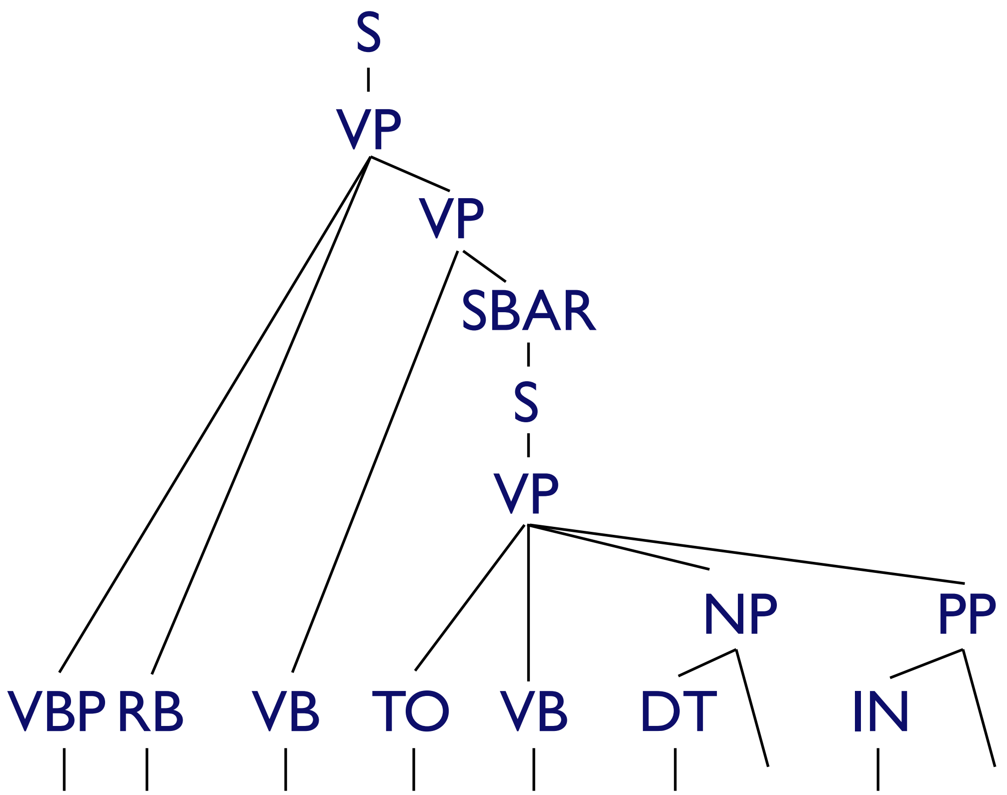
No se olvide de subir un canto rodado en Colorado

0 1 2 3 4 5 6 7 8 9 10

# 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

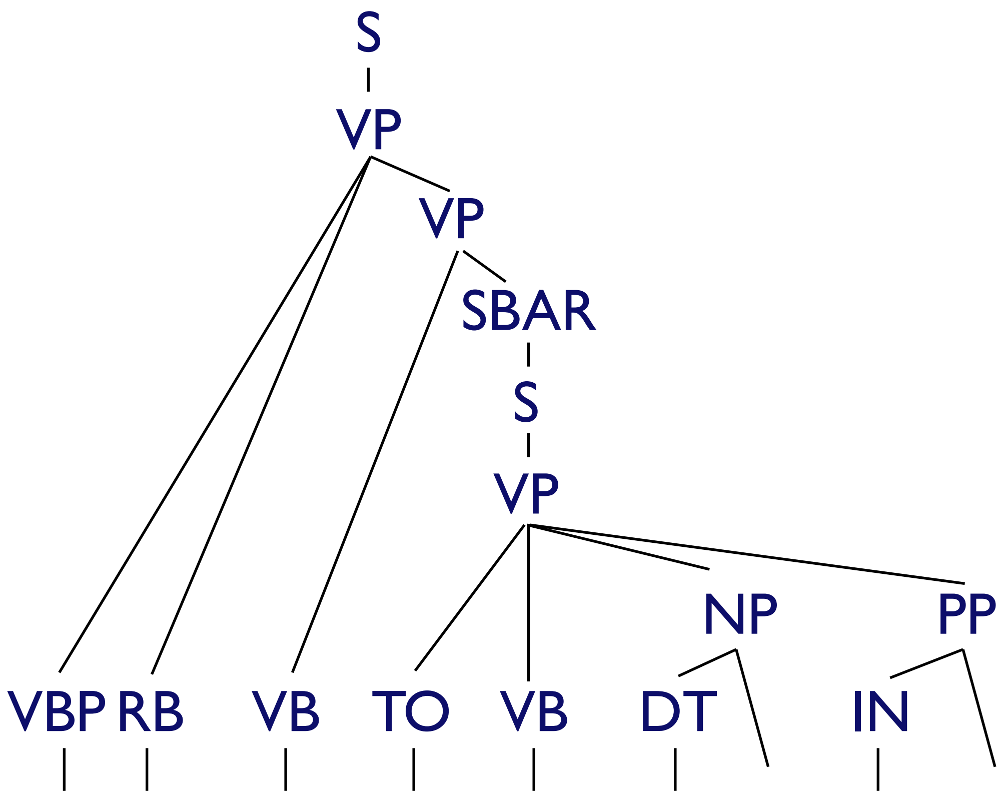
**Rules**

NN

canto rodado ; *boulder*

NNP

Colorado ; *Colorado*

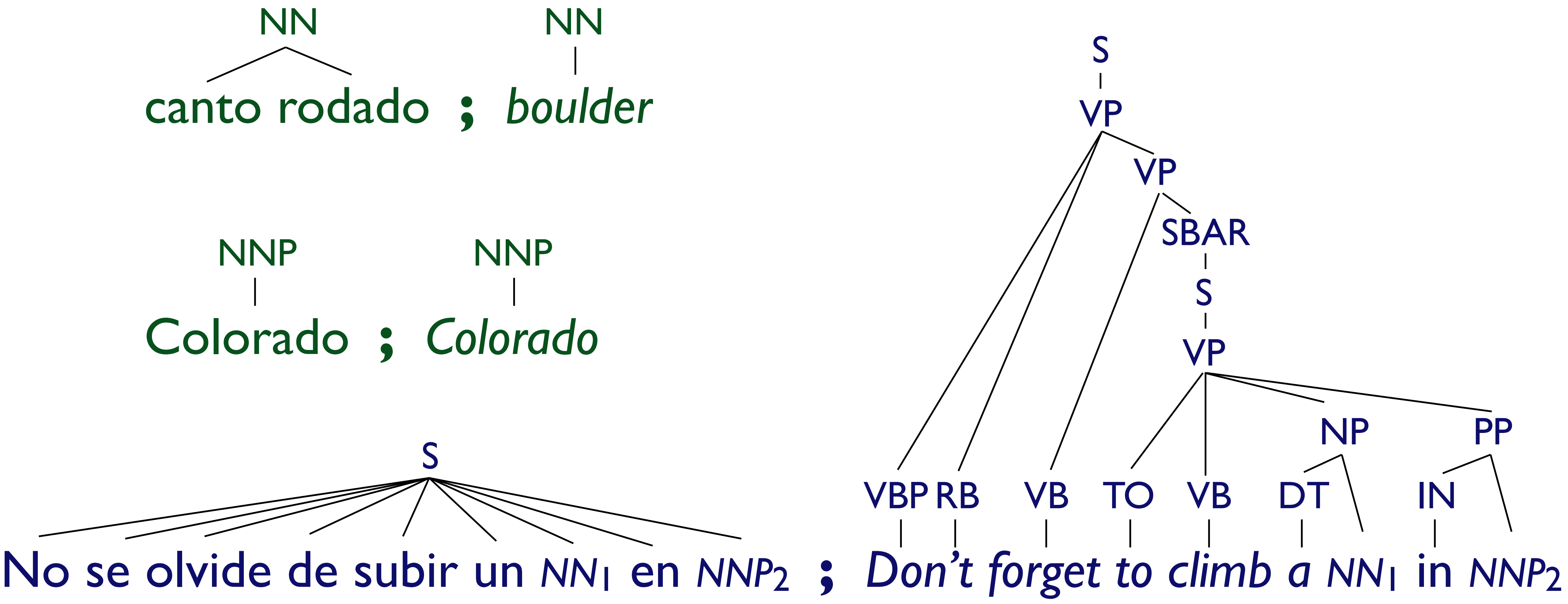


No se olvide de subir un $NN_1$ en $NNP_2$ ; *Don’t forget to climb a* $NN_1$ in $NNP_2$

# Tree Transducer Grammars

No se olvide de subir un canto rodado en Colorado

**Rules**



NN → canto rodado ; NN → *boulder*

NNP → Colorado ; NNP → *Colorado*

S → No se olvide de subir un *$NN_1$* en *$NNP_2$* ; S(VP(VBP RB VP(VB SBAR(S(VP(TO VB NP(DT *$NN_1$*) PP(IN *$NNP_2$*))))))) → *Don't forget to climb a $NN_1$* in *$NNP_2$*