Algorithms for NLP

Machine Translation IV

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein, David Hall – UC Berkeley

Translating with Tree Transducers

Input **Output**

lo haré de muy buen grado .

Grammar

Translating with Tree Transducers

Input **Output**

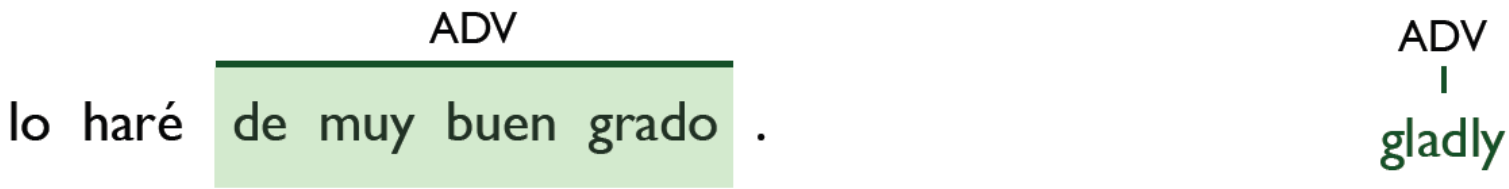
lo haré de muy buen grado .

Grammar

ADV → ⟨ de muy buen grado ; gladly ⟩

Translating with Tree Transducers

Input **Output**

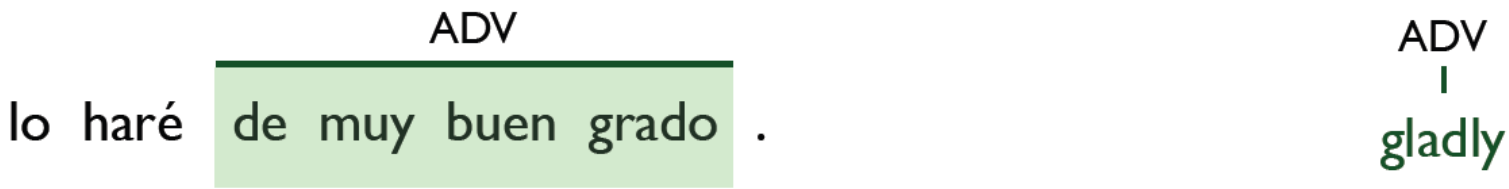


Grammar

ADV → ⟨ de muy buen grado ; gladly ⟩

Translating with Tree Transducers

Input **Output**



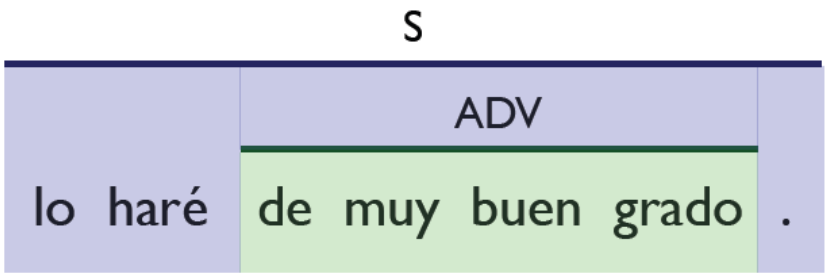
Grammar

S → ⟨ lo haré ADV . ; I will do it ADV . ⟩

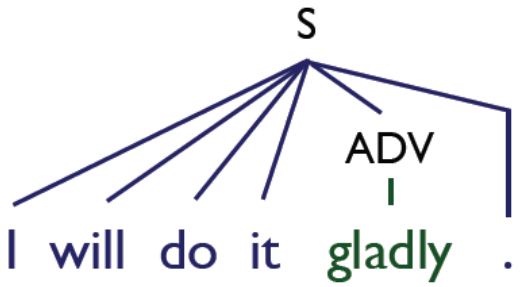
ADV → ⟨ de muy buen grado ; gladly ⟩

Translating with Tree Transducers

Input



Output



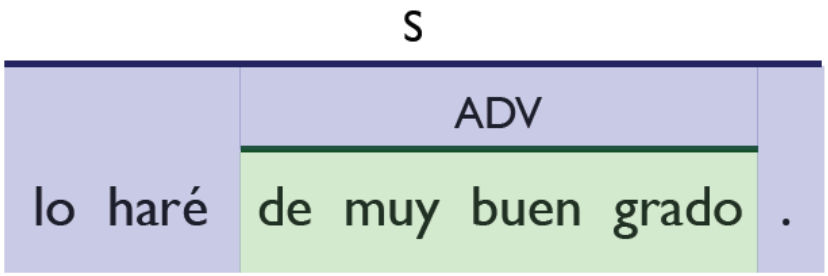
Grammar

S → ⟨ lo haré ADV . ; I will do it ADV . ⟩

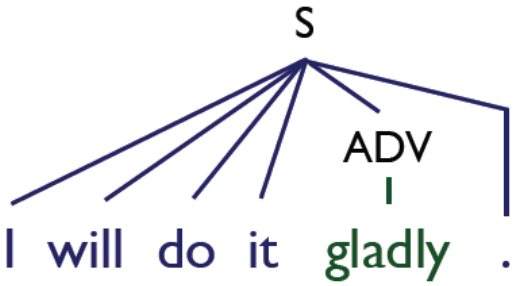
ADV → ⟨ de muy buen grado ; gladly ⟩

Translating with Tree Transducers

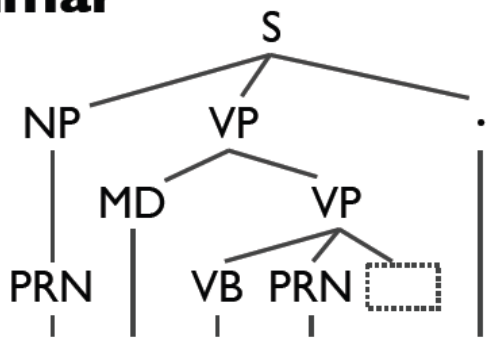
Input



Output



Grammar

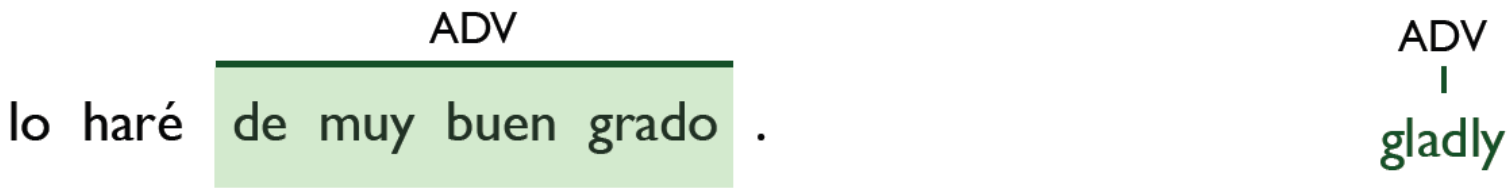


S → ⟨ lo haré ADV . ; I will do it ADV . ⟩

ADV → ⟨ de muy buen grado ; gladly ⟩

Translating with Tree Transducers

Input **Output**



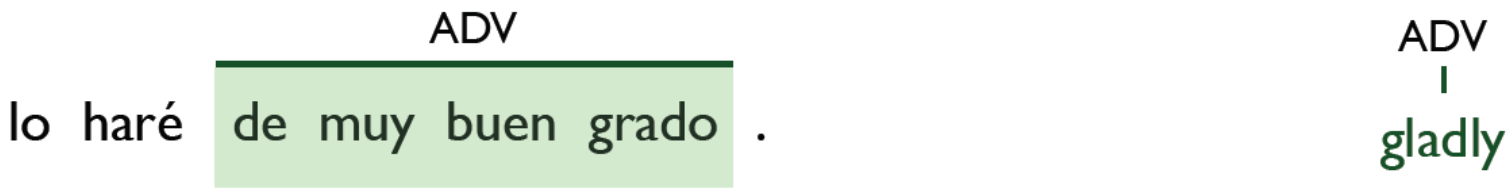
Grammar

S → ⟨ lo haré ADV . ; I will do it ADV . ⟩

ADV → ⟨ de muy buen grado ; gladly ⟩

Translating with Tree Transducers

Input **Output**



Grammar

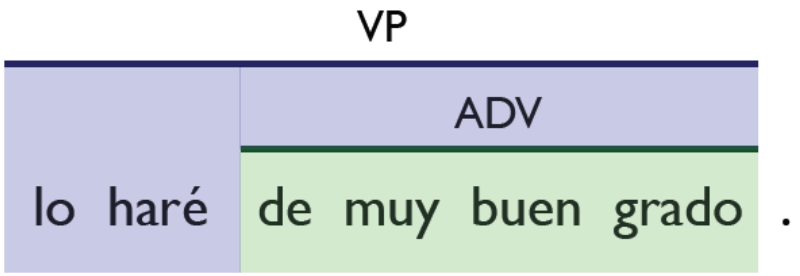
VP → ⟨ lo haré ADV ; will do it ADV ⟩

S → ⟨ lo haré ADV . ; I will do it ADV . ⟩

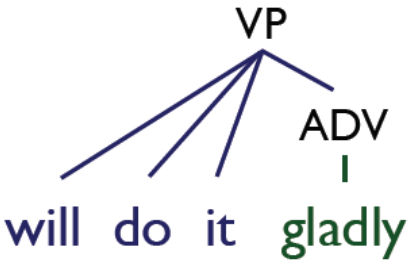
ADV → ⟨ de muy buen grado ; gladly ⟩

Translating with Tree Transducers

Input



Output



Grammar

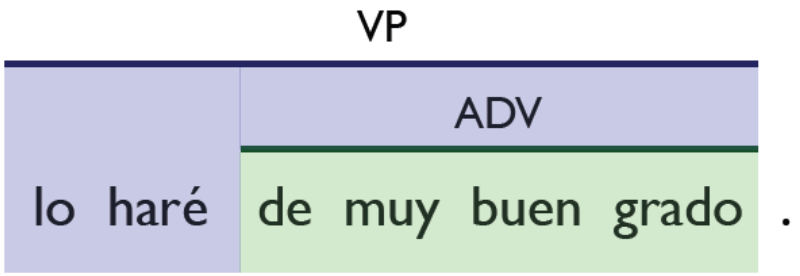
VP → ⟨ lo haré ADV ; will do it ADV ⟩

S → ⟨ lo haré ADV . ; I will do it ADV . ⟩

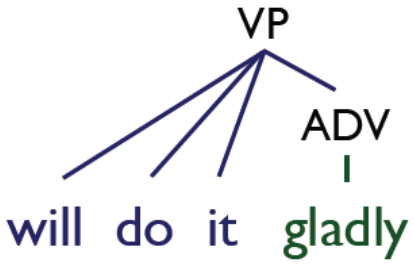
ADV → ⟨ de muy buen grado ; gladly ⟩

Translating with Tree Transducers

Input



Output



Grammar

S → ⟨ VP . ; I VP . ⟩

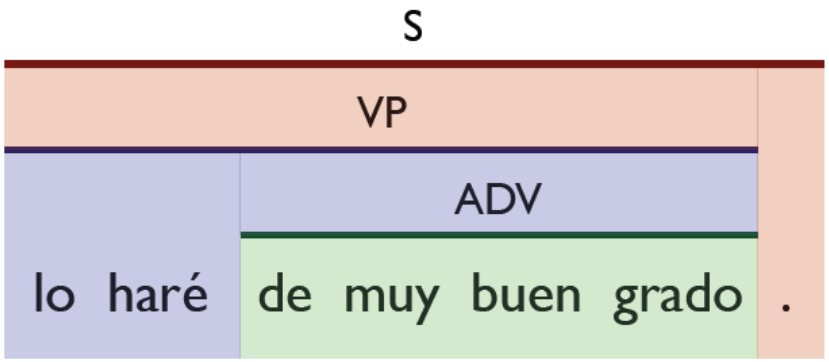
VP → ⟨ lo haré ADV ; will do it ADV ⟩

S → ⟨ lo haré ADV . ; I will do it ADV . ⟩

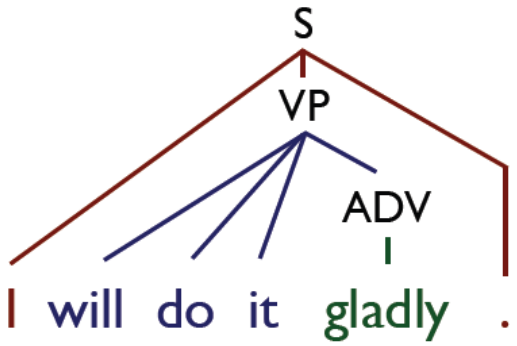
ADV → ⟨ de muy buen grado ; gladly ⟩

Translating with Tree Transducers

Input



Output



Grammar

S → ⟨ VP . ; I VP . ⟩

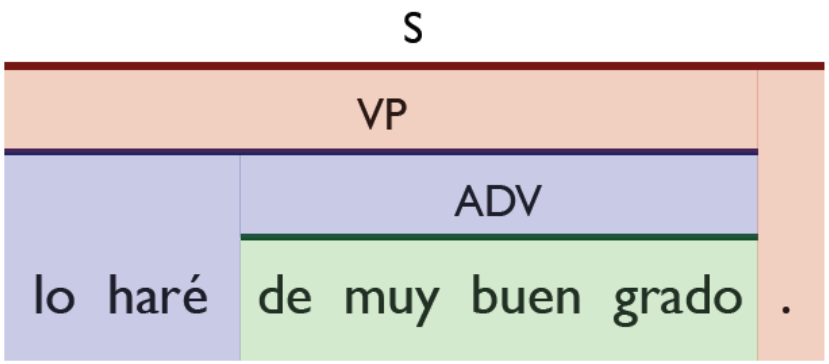
VP → ⟨ lo haré ADV ; will do it ADV ⟩

S → ⟨ lo haré ADV . ; I will do it ADV . ⟩

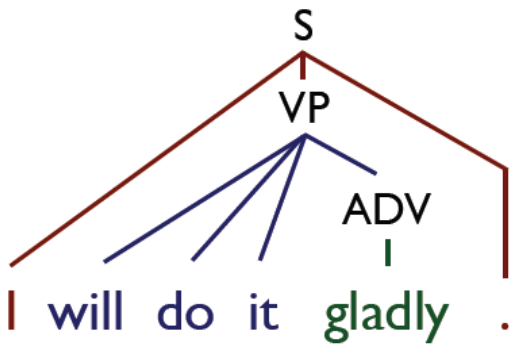
ADV → ⟨ de muy buen grado ; gladly ⟩

Translating with Tree Transducers

Input



Output



Grammar

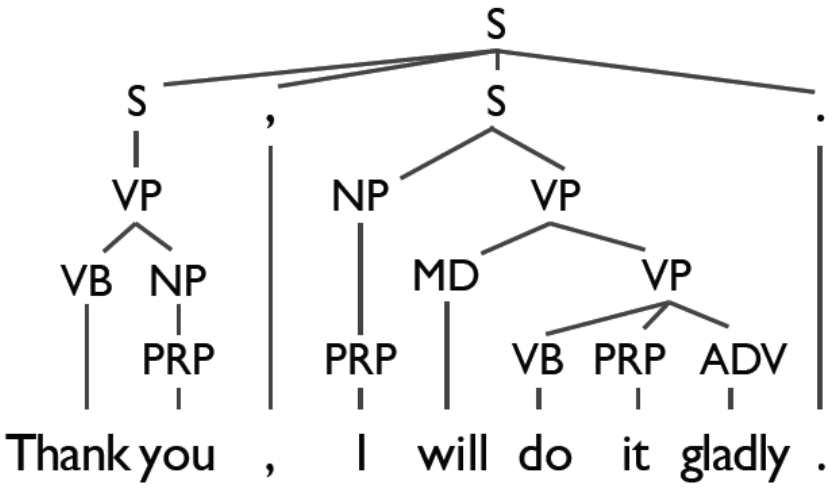
S → ⟨ VP . ; I VP . ⟩ **OR** S → ⟨ VP . ; you VP . ⟩

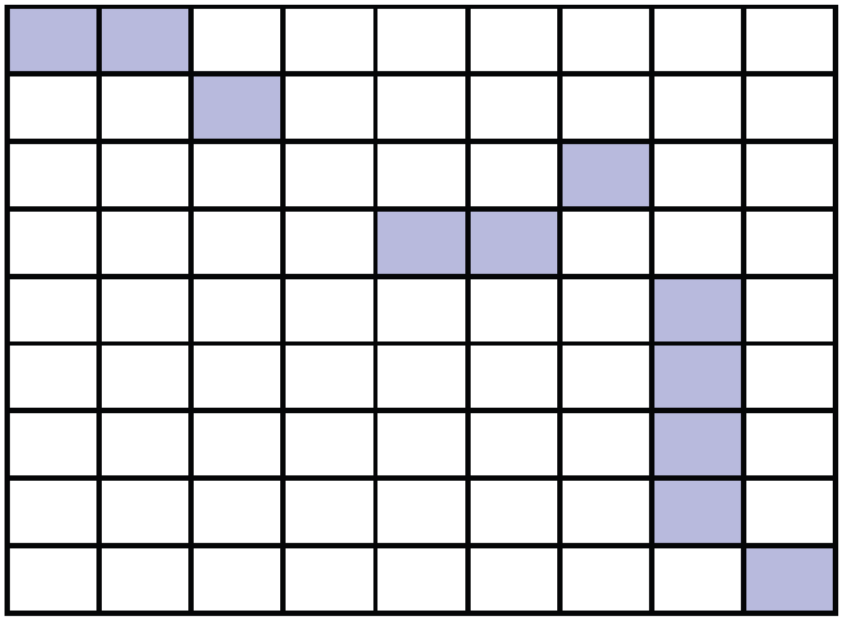
VP → ⟨ lo haré ADV ; will do it ADV ⟩

S → ⟨ lo haré ADV . ; I will do it ADV . ⟩

ADV → ⟨ de muy buen grado ; gladly ⟩

Learning Grammars for Translation

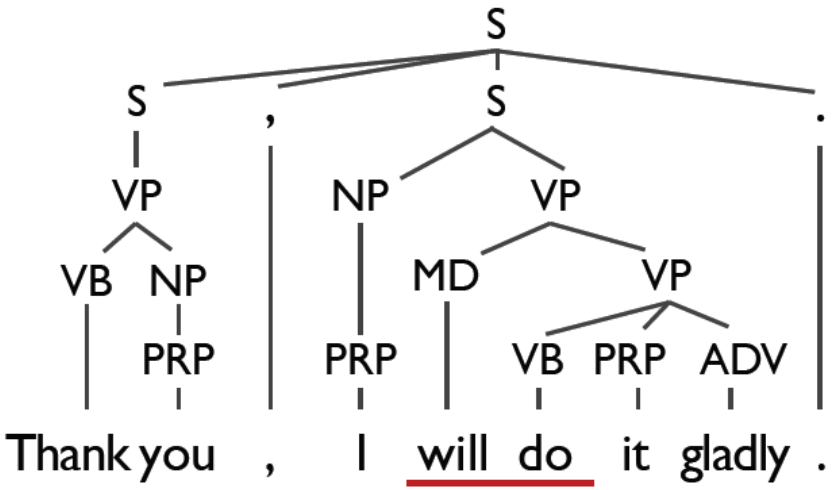


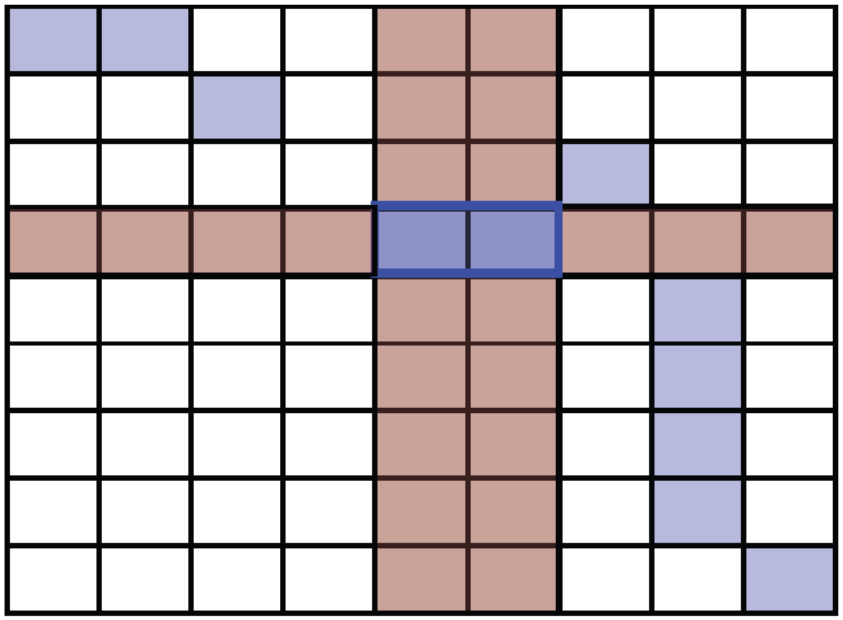


Gracias
,
lo
haré
de
muy
buen
grado
.

Grammar Rules

Learning Grammars for Translation

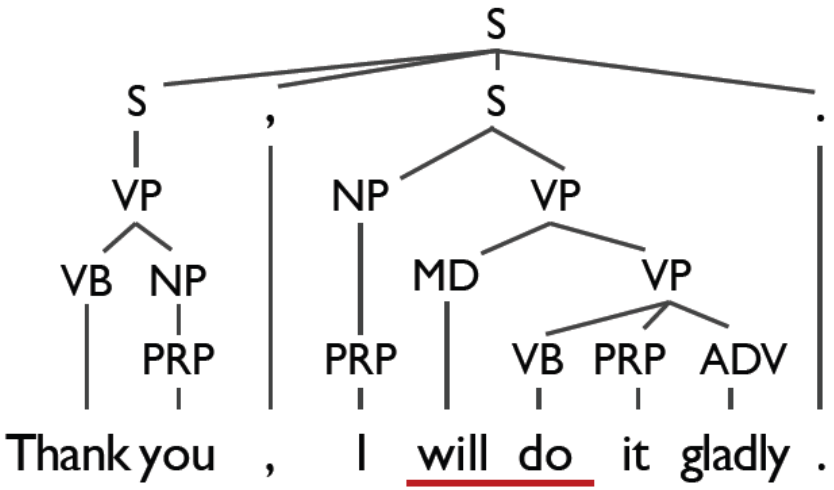


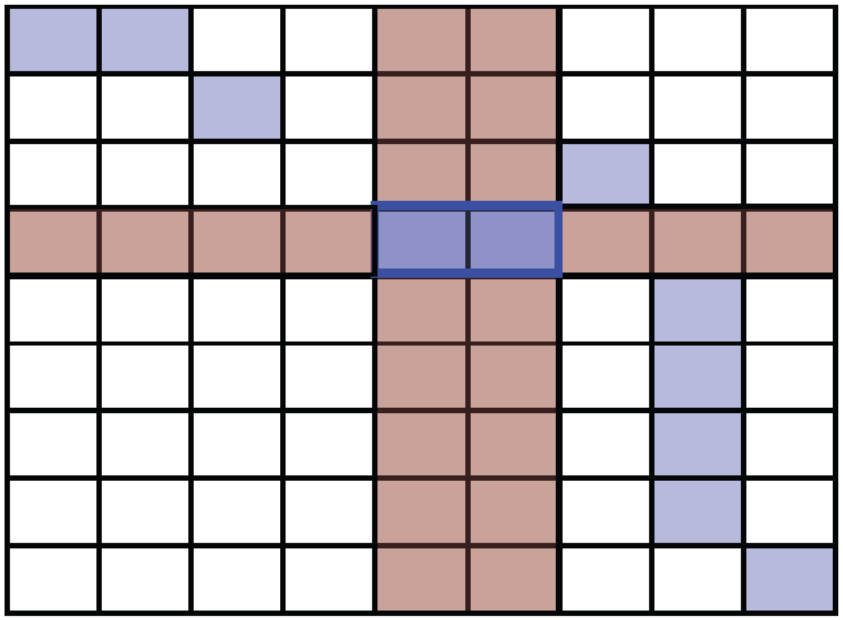


Gracias
,
lo
haré
de
muy
buen
grado
.

Grammar Rules

Learning Grammars for Translation



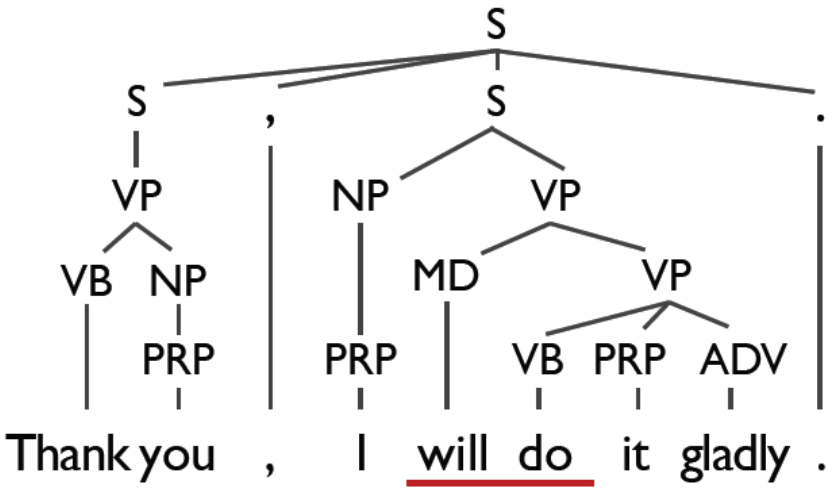


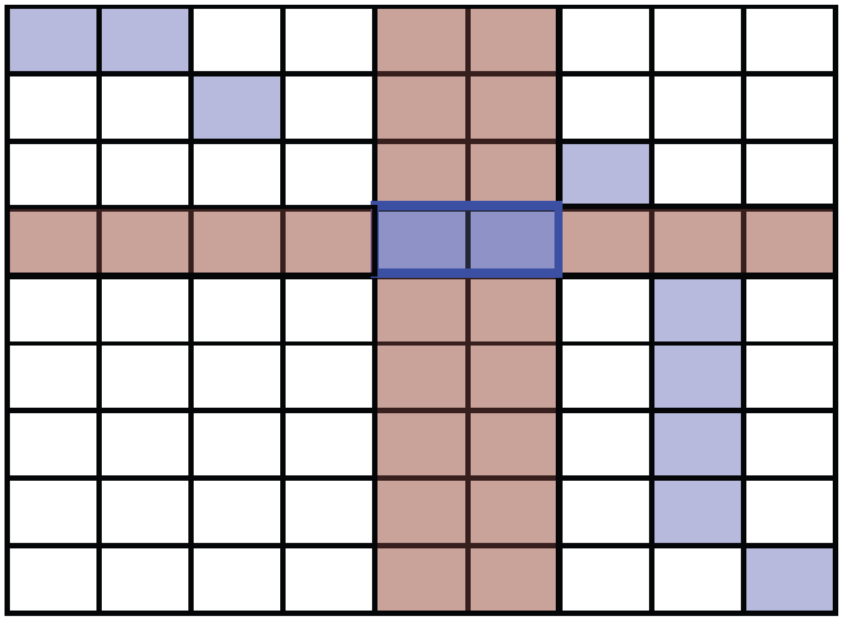
Gracias
,
lo
haré
de
muy
buen
grado
.

Grammar Rules

⟨haré ; will do⟩

Learning Grammars for Translation



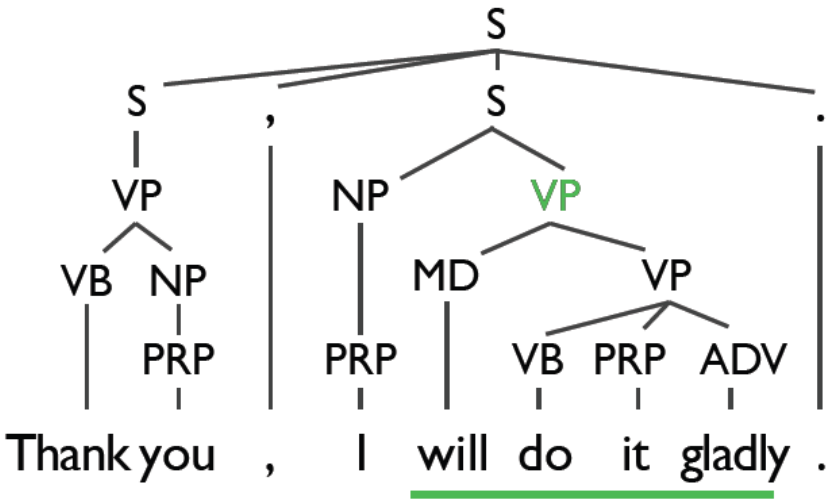


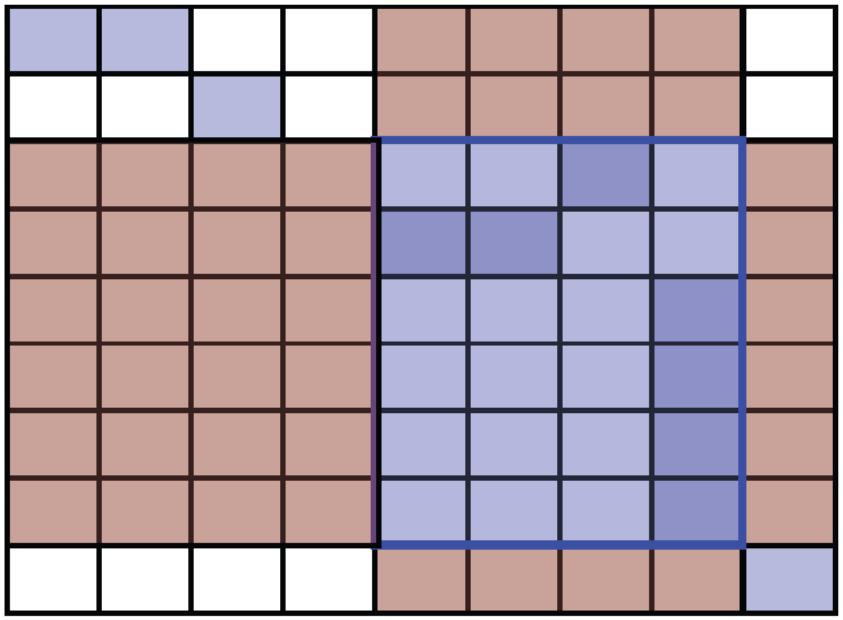
Gracias
,
lo
haré
de
muy
buen
grado
.

Grammar Rules

~~⟨haré ; will do⟩~~

Learning Grammars for Translation



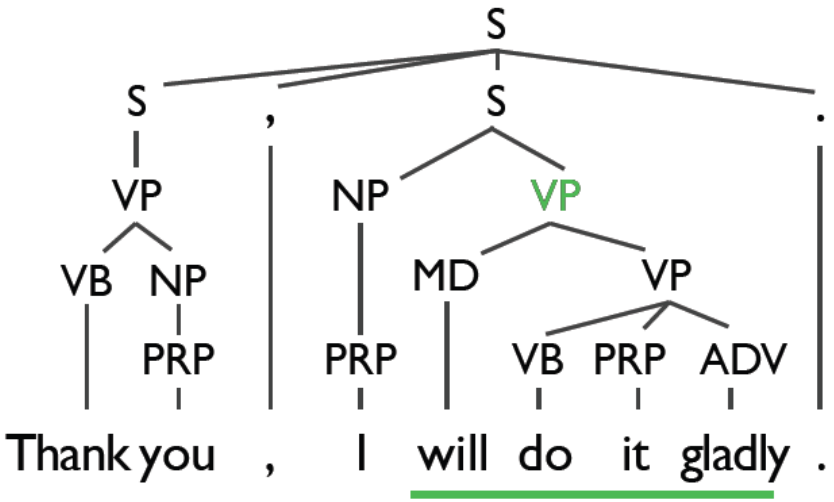


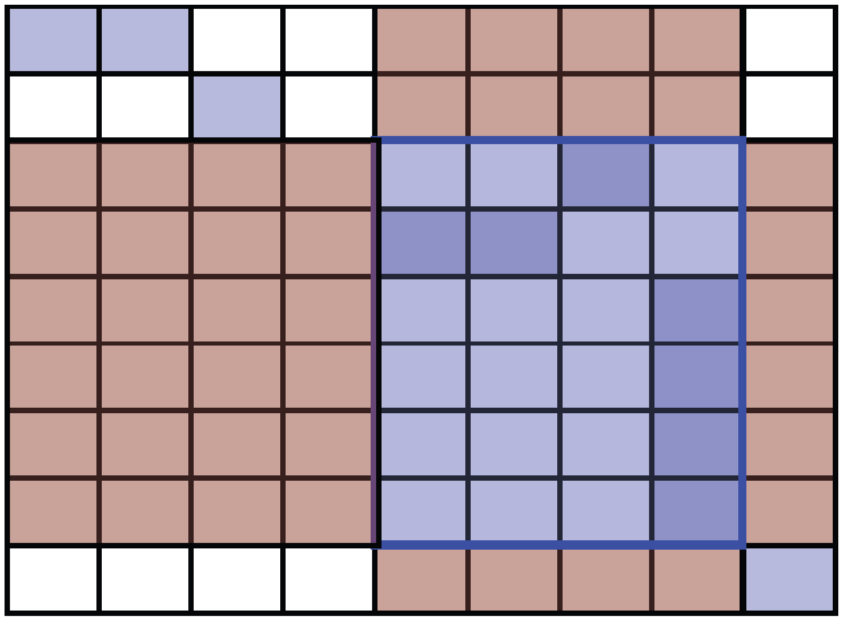
Gracias
,
lo
haré
de
muy
buen
grado
.

Grammar Rules

~~⟨haré ; will do⟩~~

Learning Grammars for Translation





Gracias , lo haré de muy buen grado .

Grammar Rules

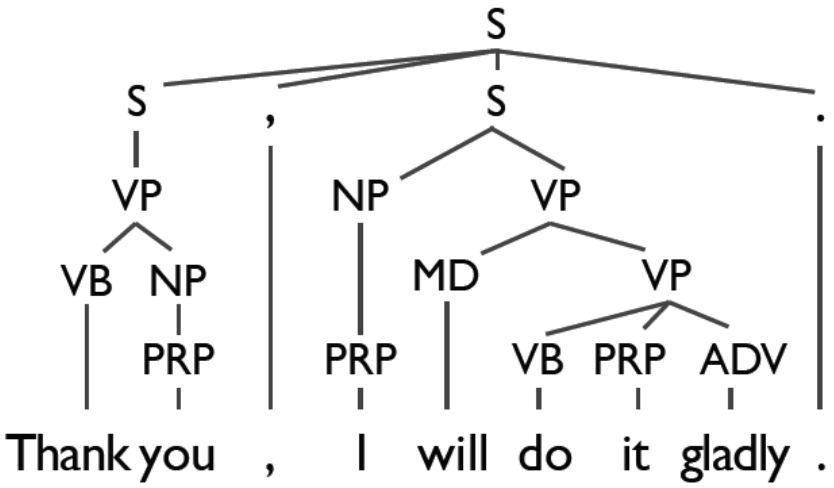
~~⟨haré ; will do⟩~~

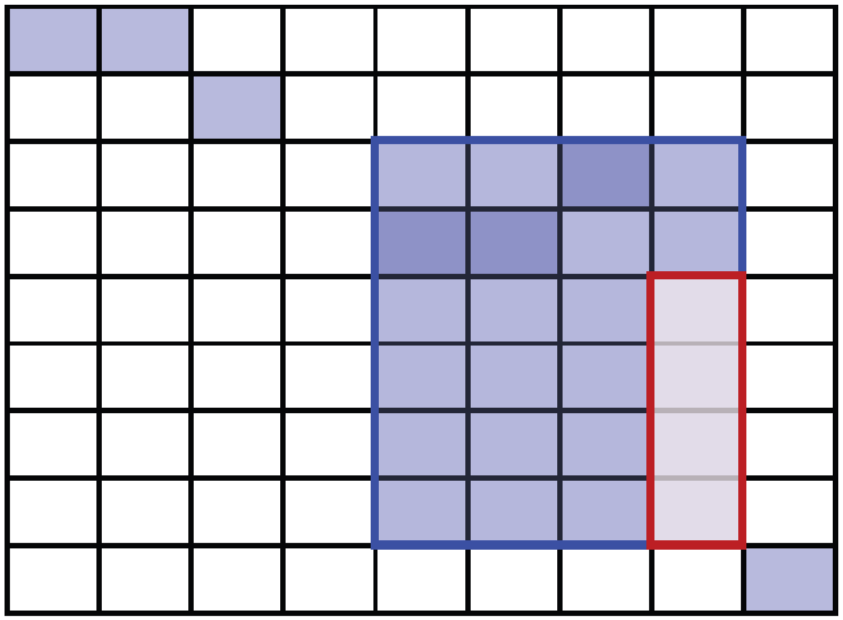
VP →

⟨lo haré de ... grado ;

will do it gladly⟩

Learning Grammars for Translation





Gracias
,
lo
haré
de
muy
buen
grado
.

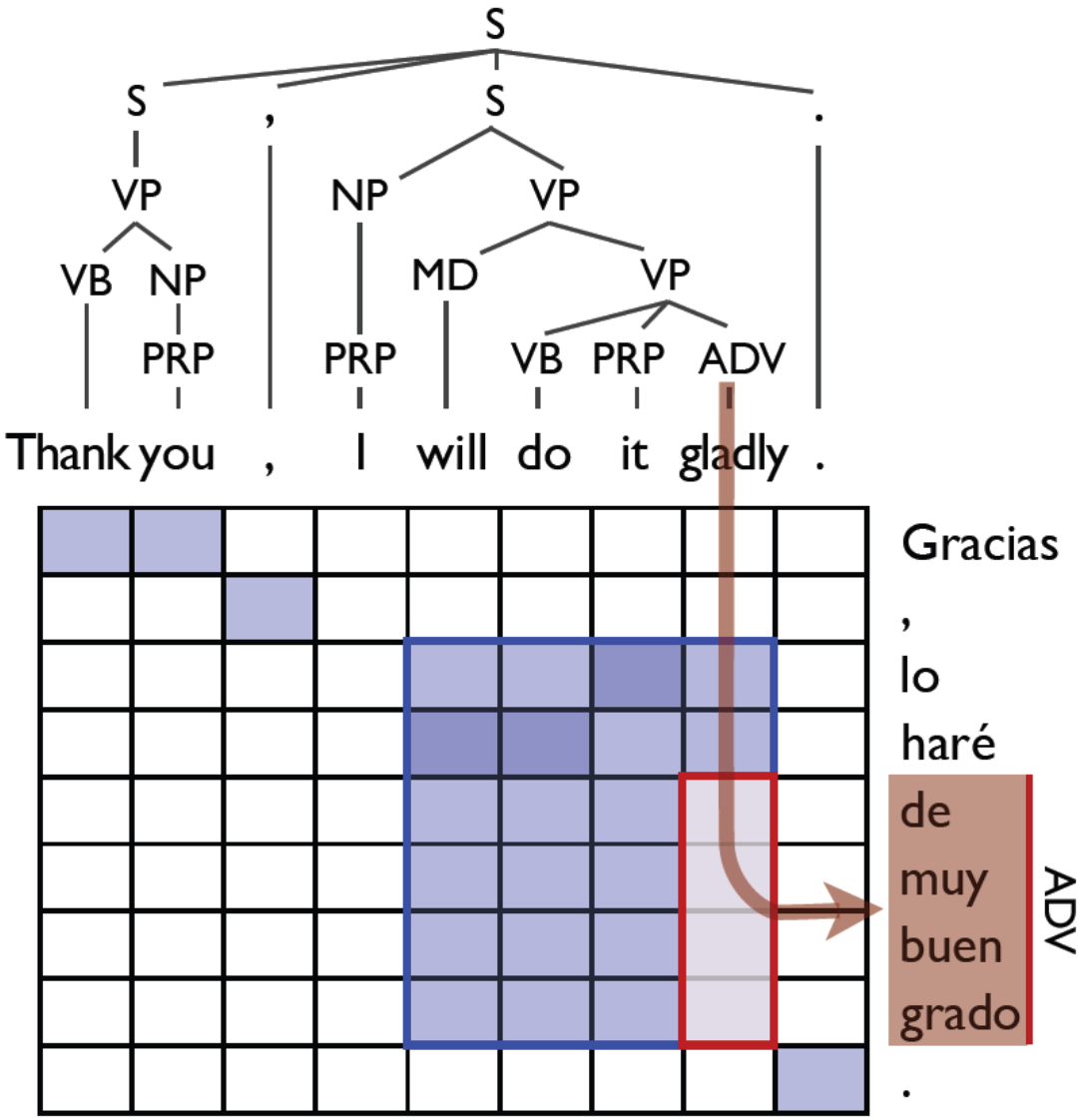
Grammar Rules

~~⟨haré ; will do⟩~~

VP →

⟨lo haré de ... grado ;
will do it gladly⟩

Learning Grammars for Translation



Grammar Rules

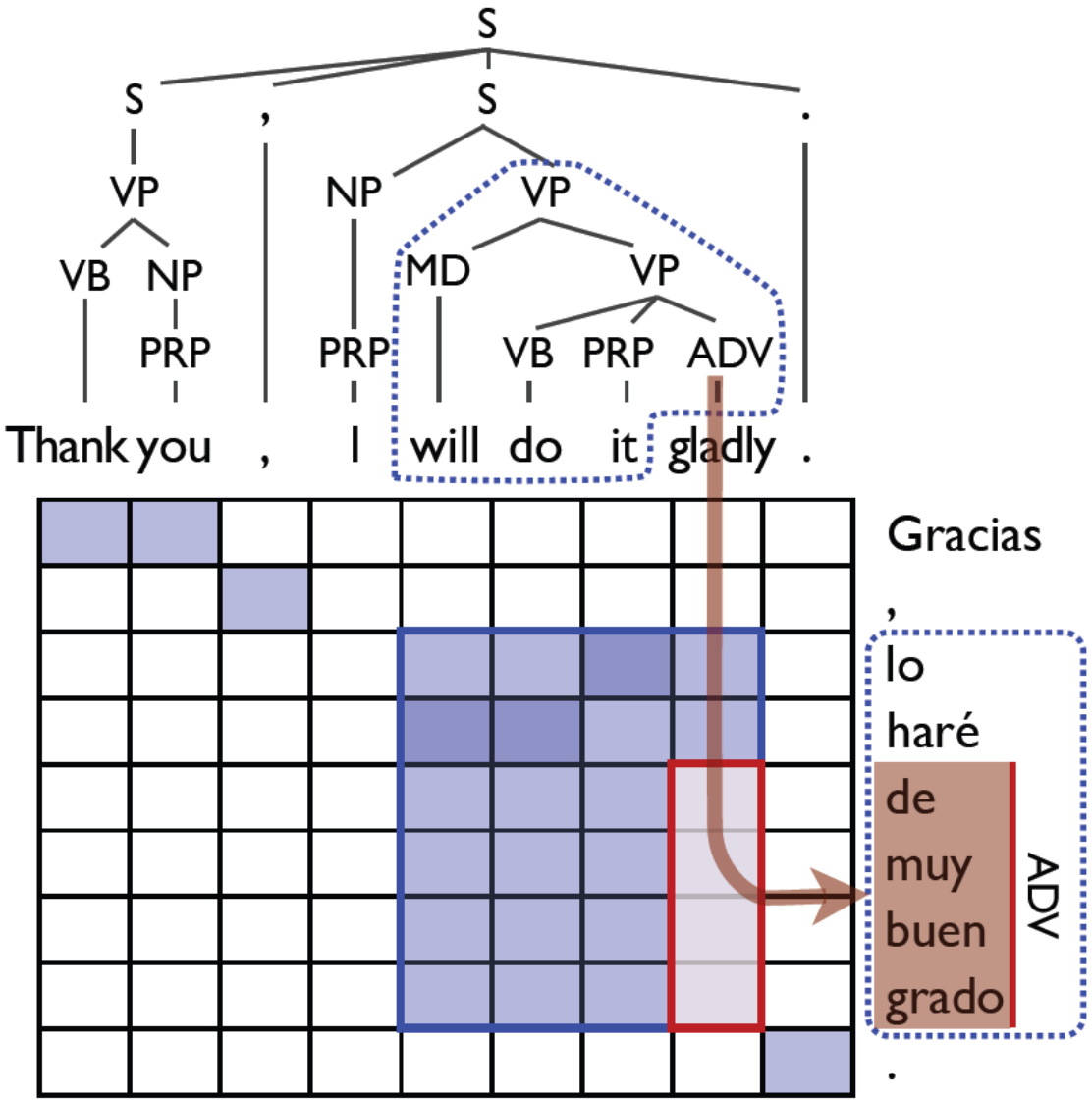
~~⟨haré ; will do⟩~~

VP →

⟨lo haré de ... grado ;

will do it gladly⟩

Learning Grammars for Translation



Grammar Rules

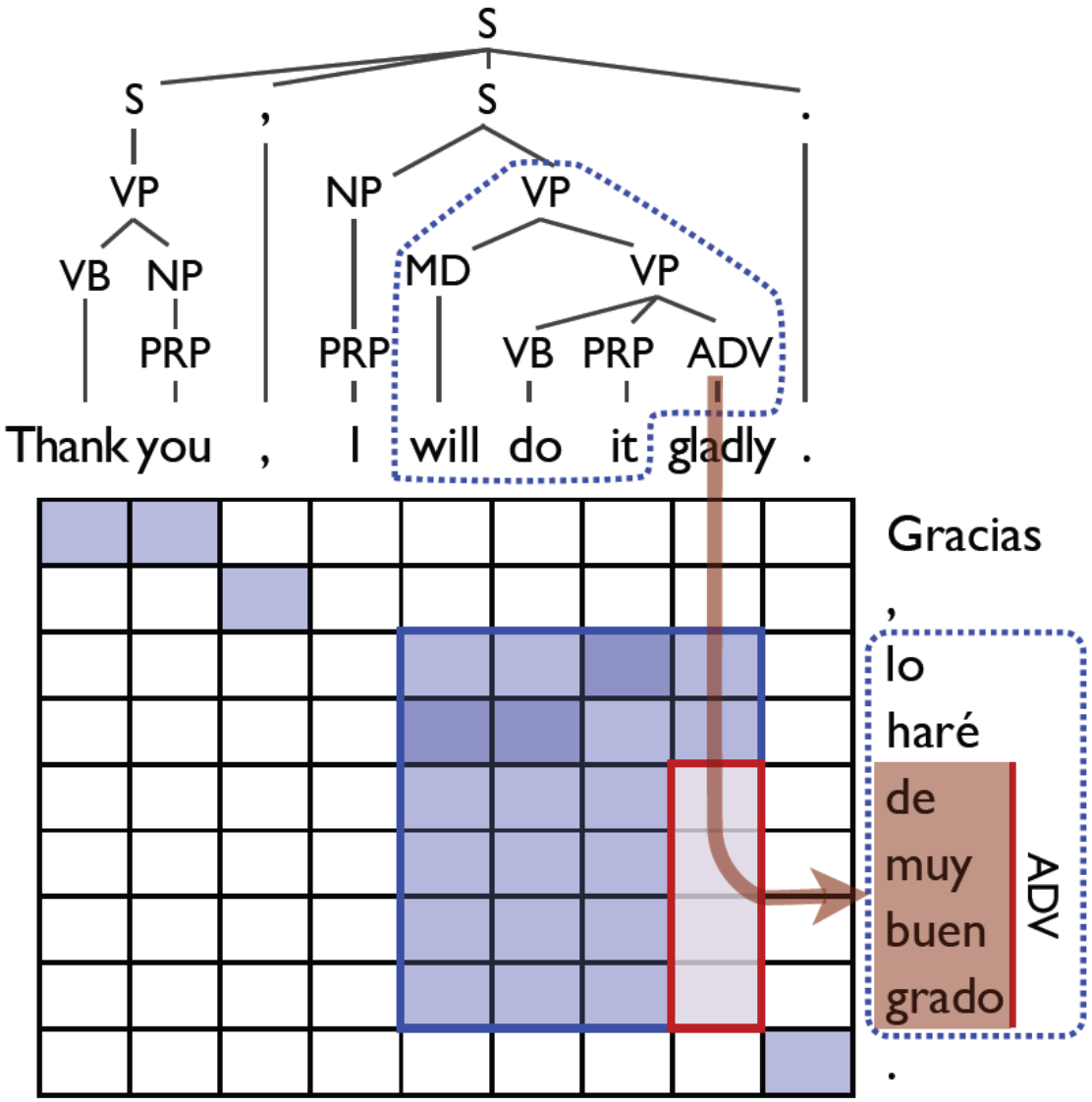
~~⟨haré ; will do⟩~~

VP →

⟨lo haré de ... grado ;

will do it gladly⟩

Learning Grammars for Translation



Grammar Rules

~~⟨haré ; will do⟩~~

VP →
⟨lo haré de ... grado ;
will do it gladly⟩

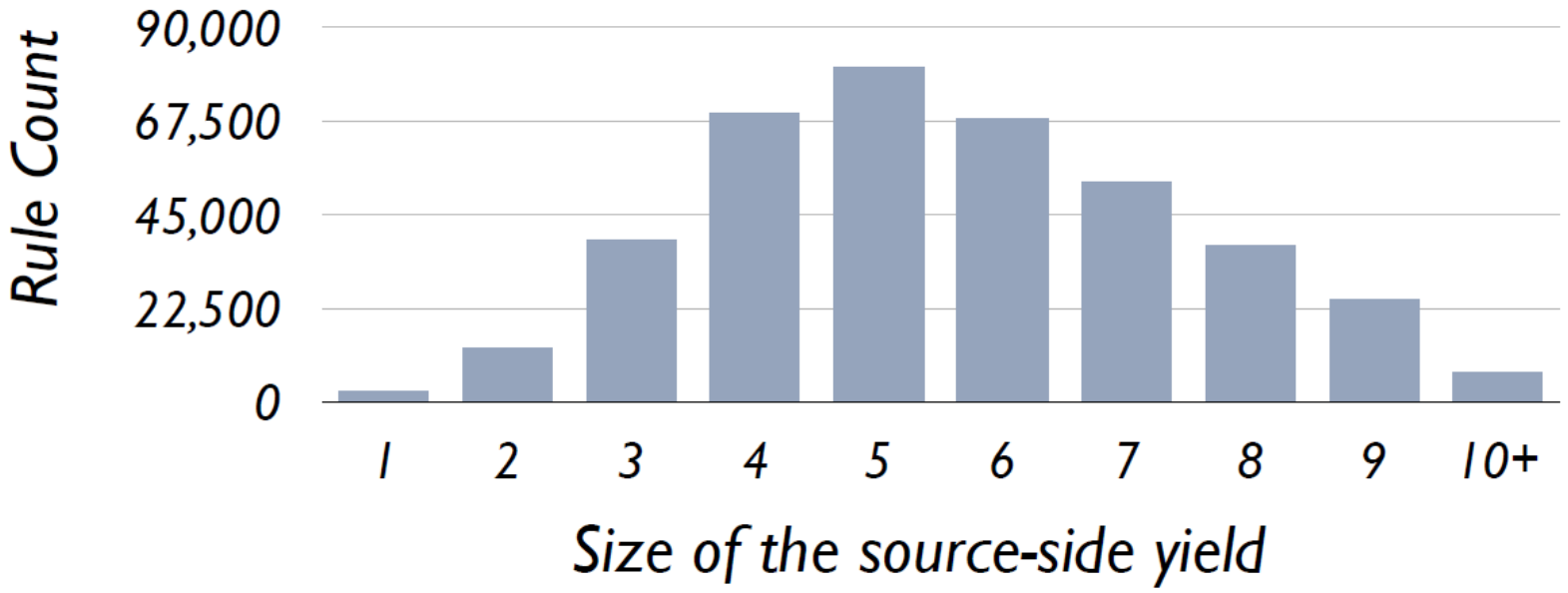
VP →
⟨lo haré ADV ;
will do it ADV⟩

The Size of Tree Transducer Grammars

Extracted a transducer grammar from a 220 million word bitext

Kept all rules with at most 6 non-terminals

Rules matching an example 40-word sentence

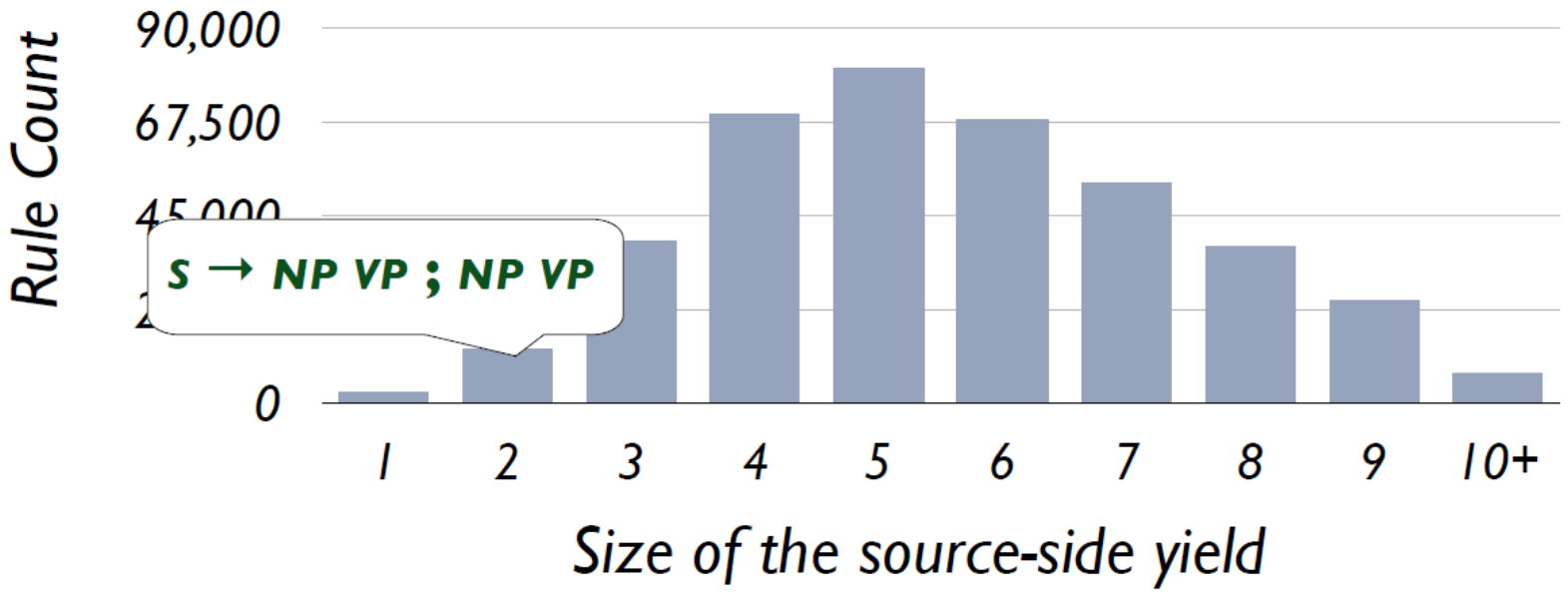


The Size of Tree Transducer Grammars

Extracted a transducer grammar from a 220 million word bitext

Kept all rules with at most 6 non-terminals

Rules matching an example 40-word sentence

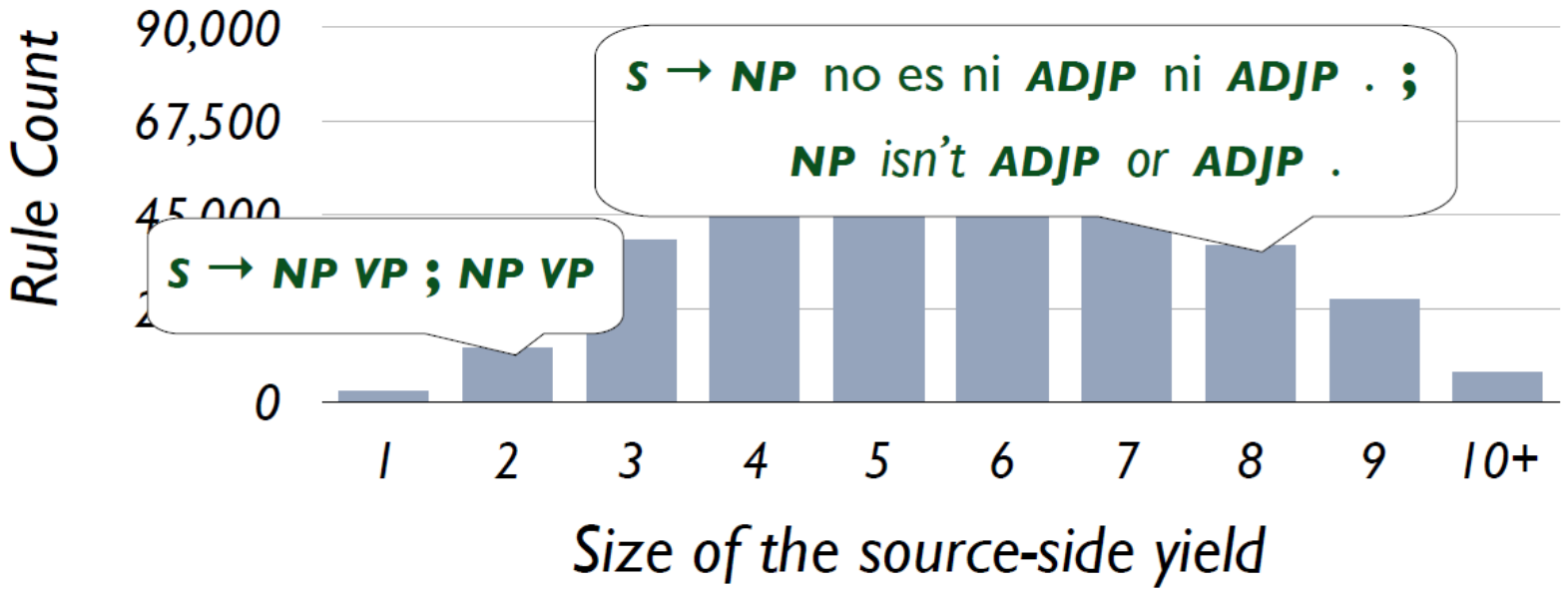


The Size of Tree Transducer Grammars

Extracted a transducer grammar from a 220 million word bitext

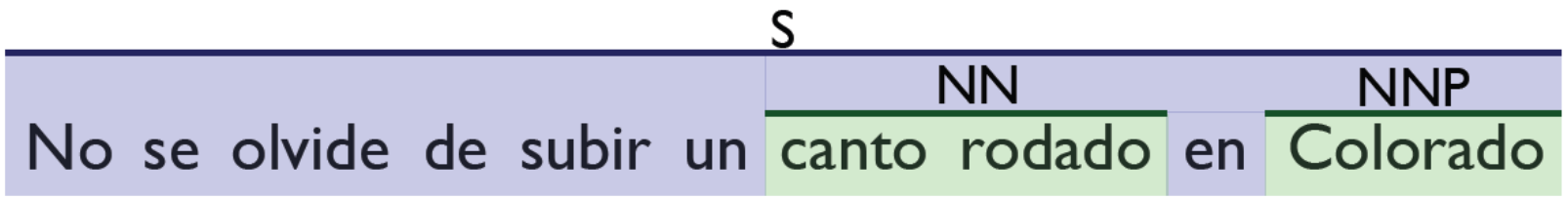
Kept all rules with at most 6 non-terminals

Rules matching an example 40-word sentence



Syntactic Decoding

Tree Transducer Grammars



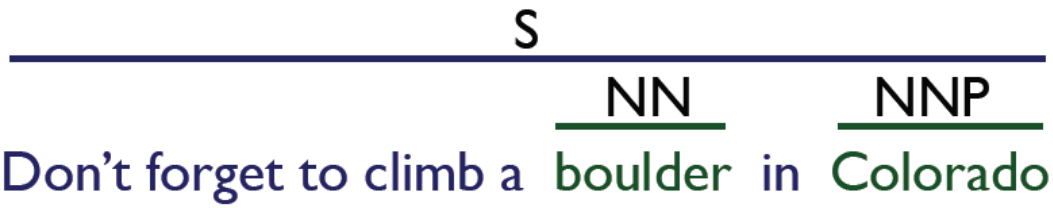
Synchronous Grammar

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



University of California Berkeley

CKY-style Bottom-up Parsing

For each
span length:

CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

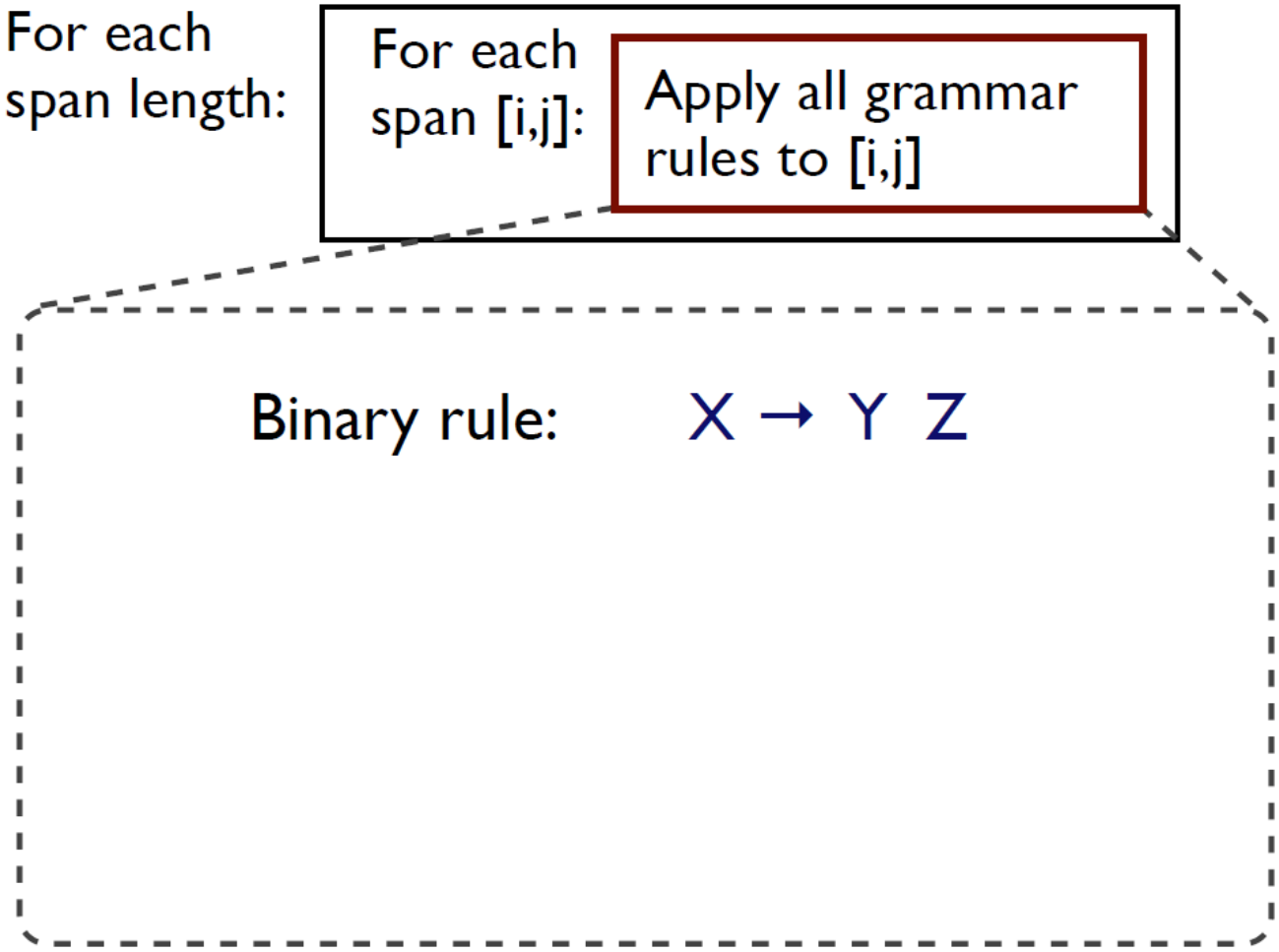
CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

Apply all grammar rules to [i,j]

CKY-style Bottom-up Parsing



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**

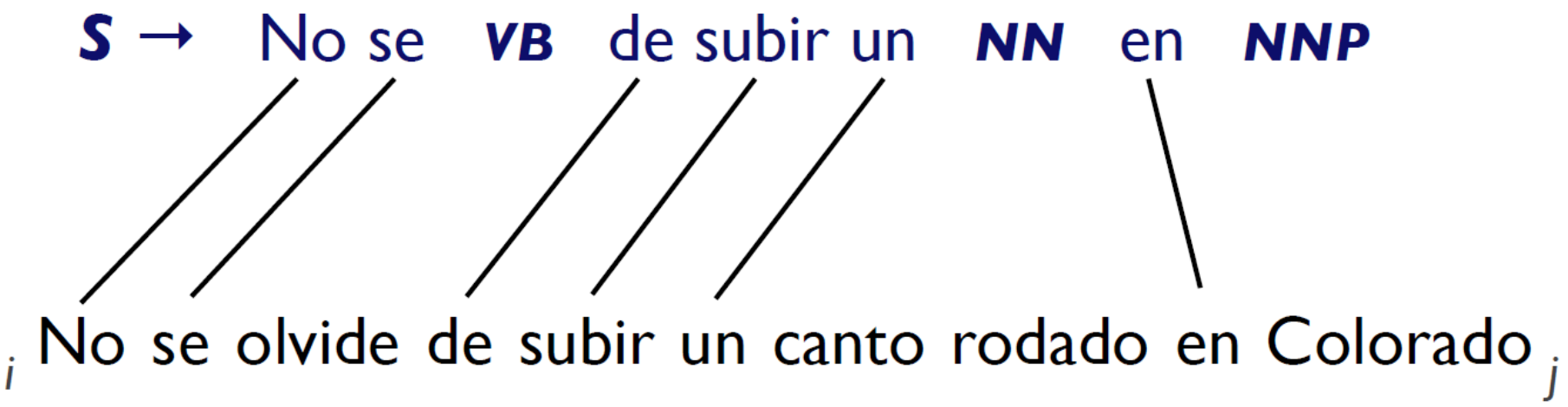
$_i$ No se olvide de subir un canto rodado en Colorado $_j$

CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

Apply all grammar rules to [i,j]

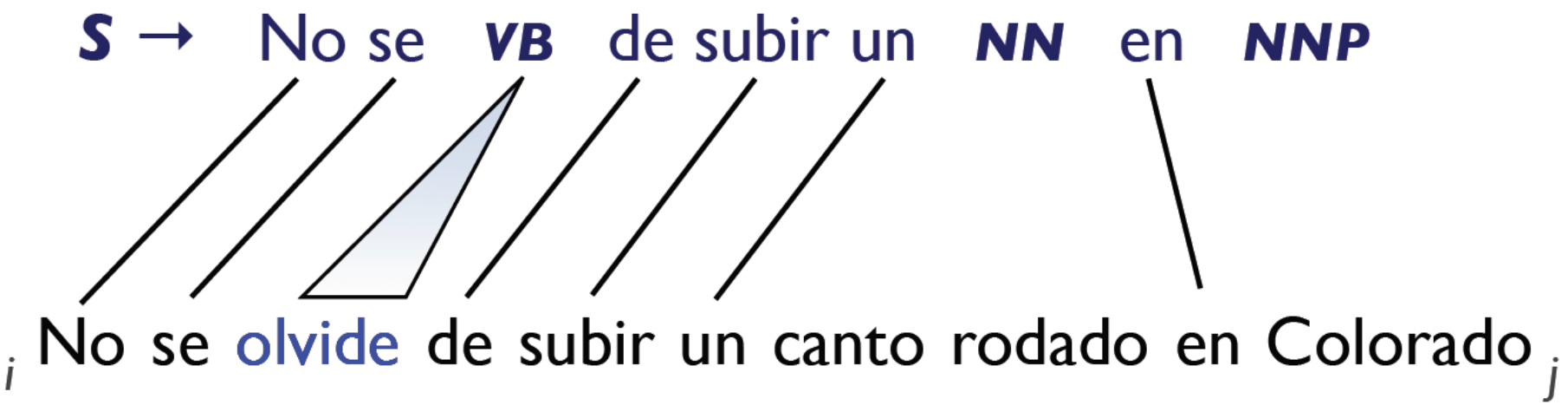


CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

Apply all grammar rules to [i,j]

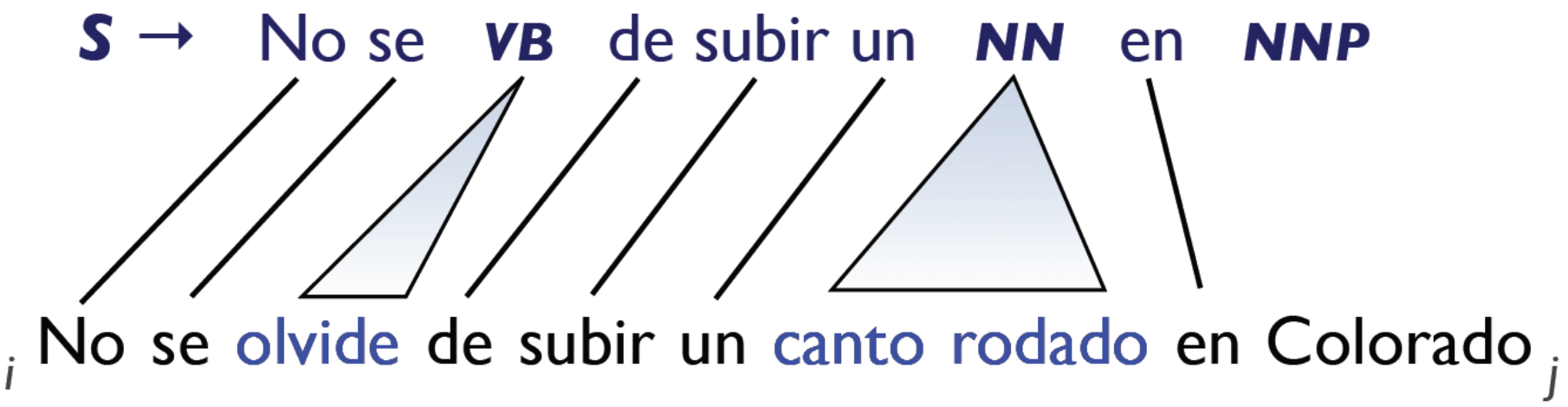


CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

Apply all grammar rules to [i,j]

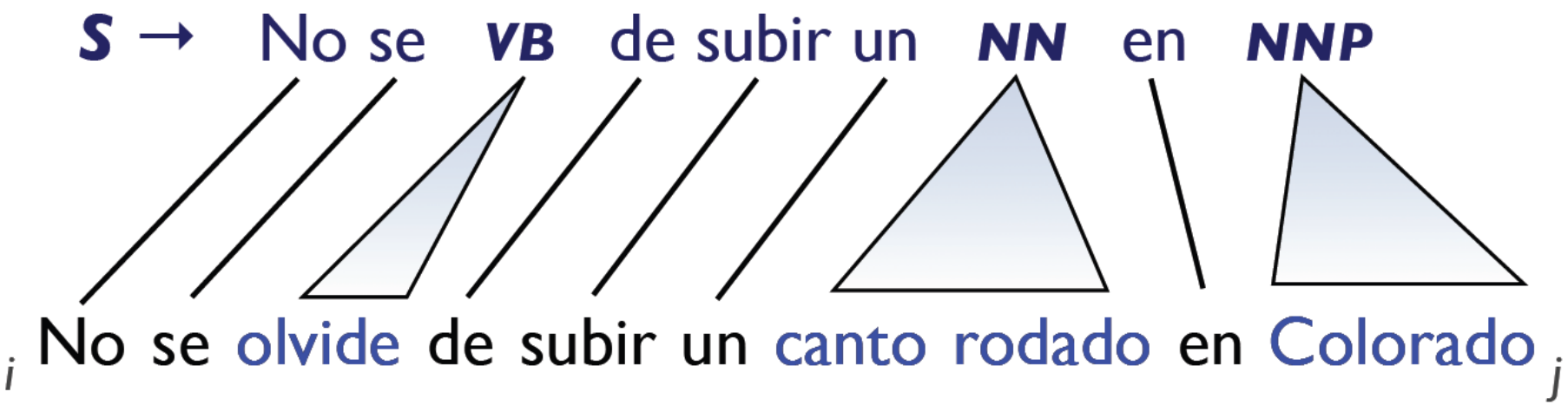


CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

Apply all grammar rules to [i,j]

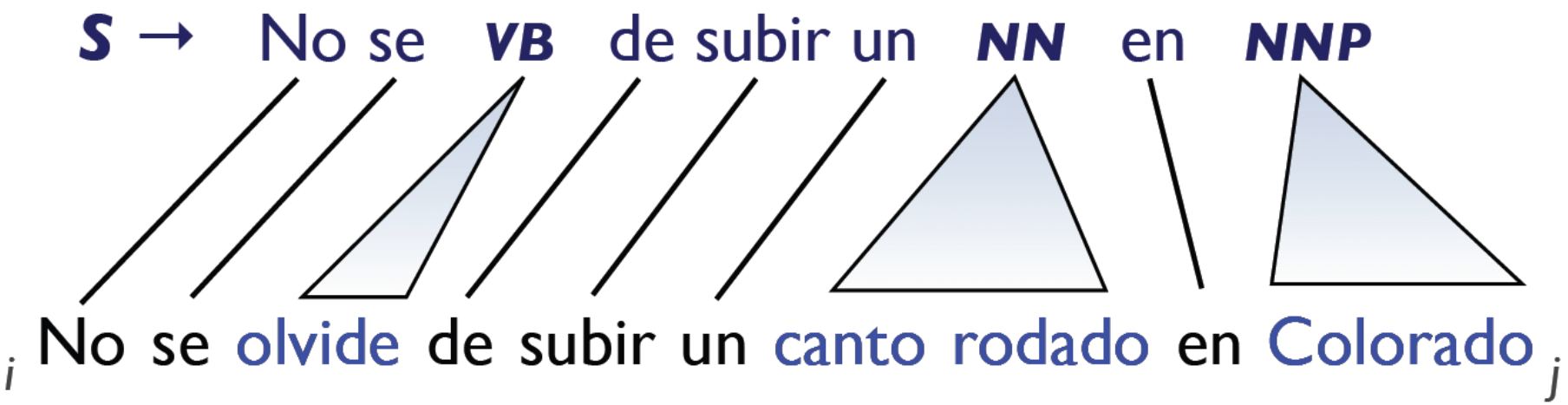


CKY-style Bottom-up Parsing

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]

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]

S → No se **VP** **NP** **PP**

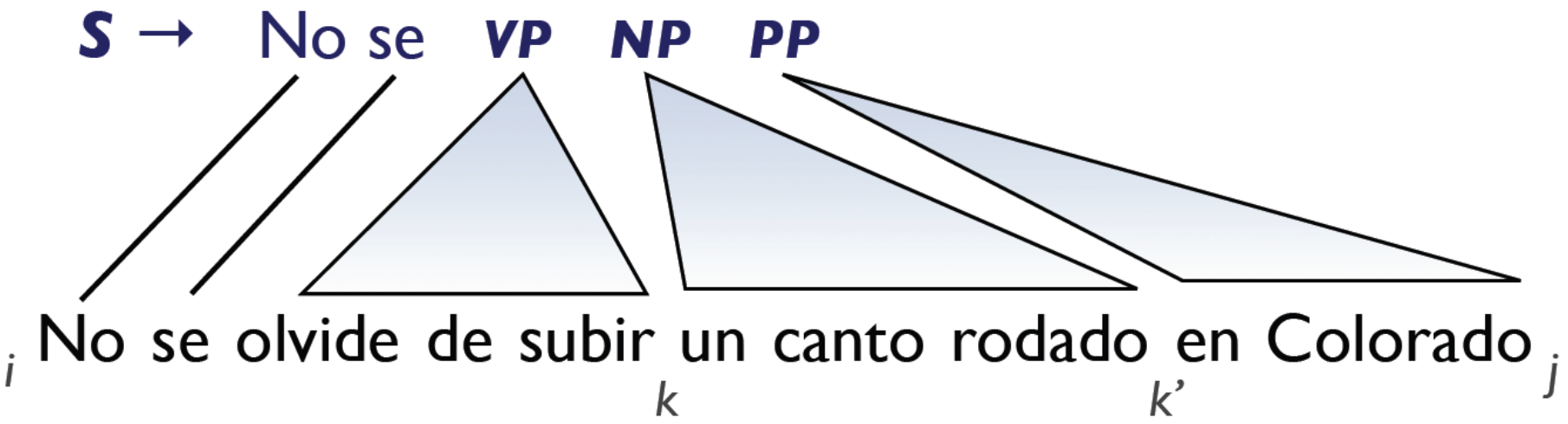
$_i$ No se olvide de subir un canto rodado en Colorado $_j$

CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

Apply all grammar rules to [i,j]

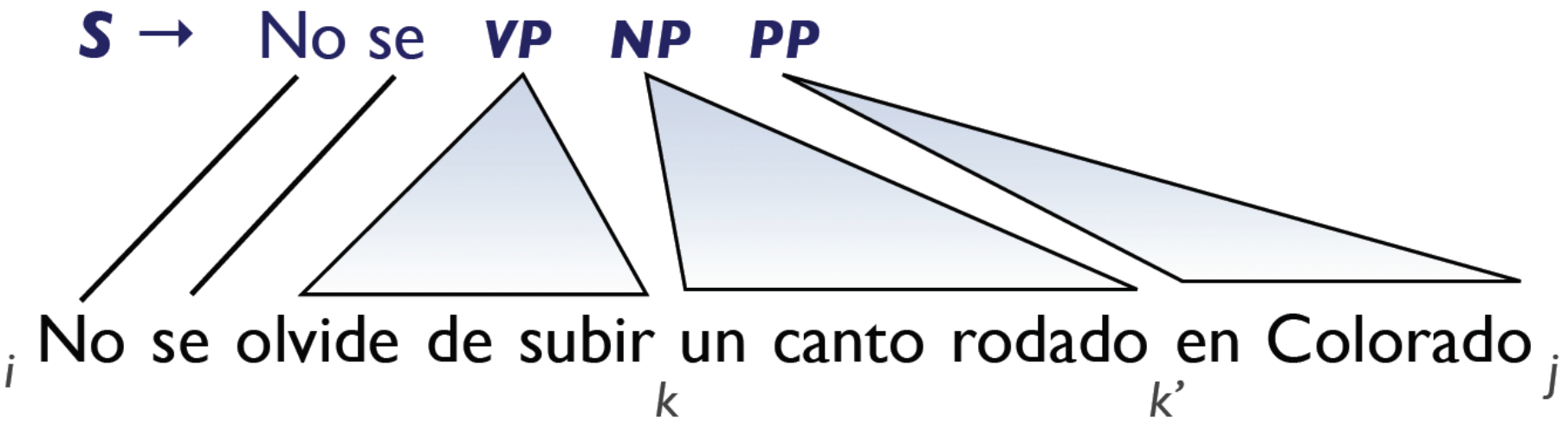


CKY-style Bottom-up Parsing

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.

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

Exploiting GPUs

Lots to Parse

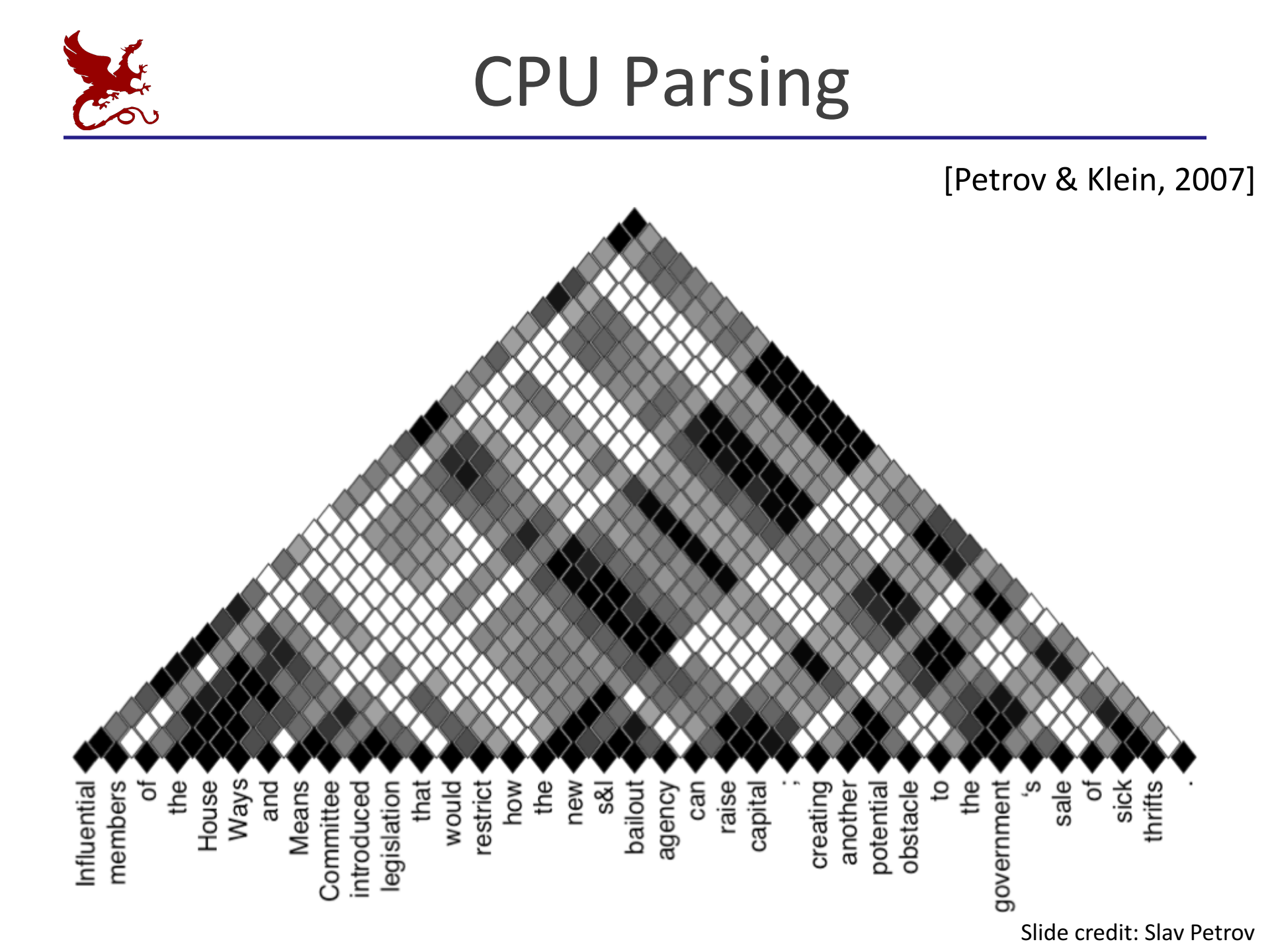
≈2.6 billion words

Lots to Parse

≈6 months (CPU)

Lots to Parse

≈3.6 days (GPU)



CPU Parsing

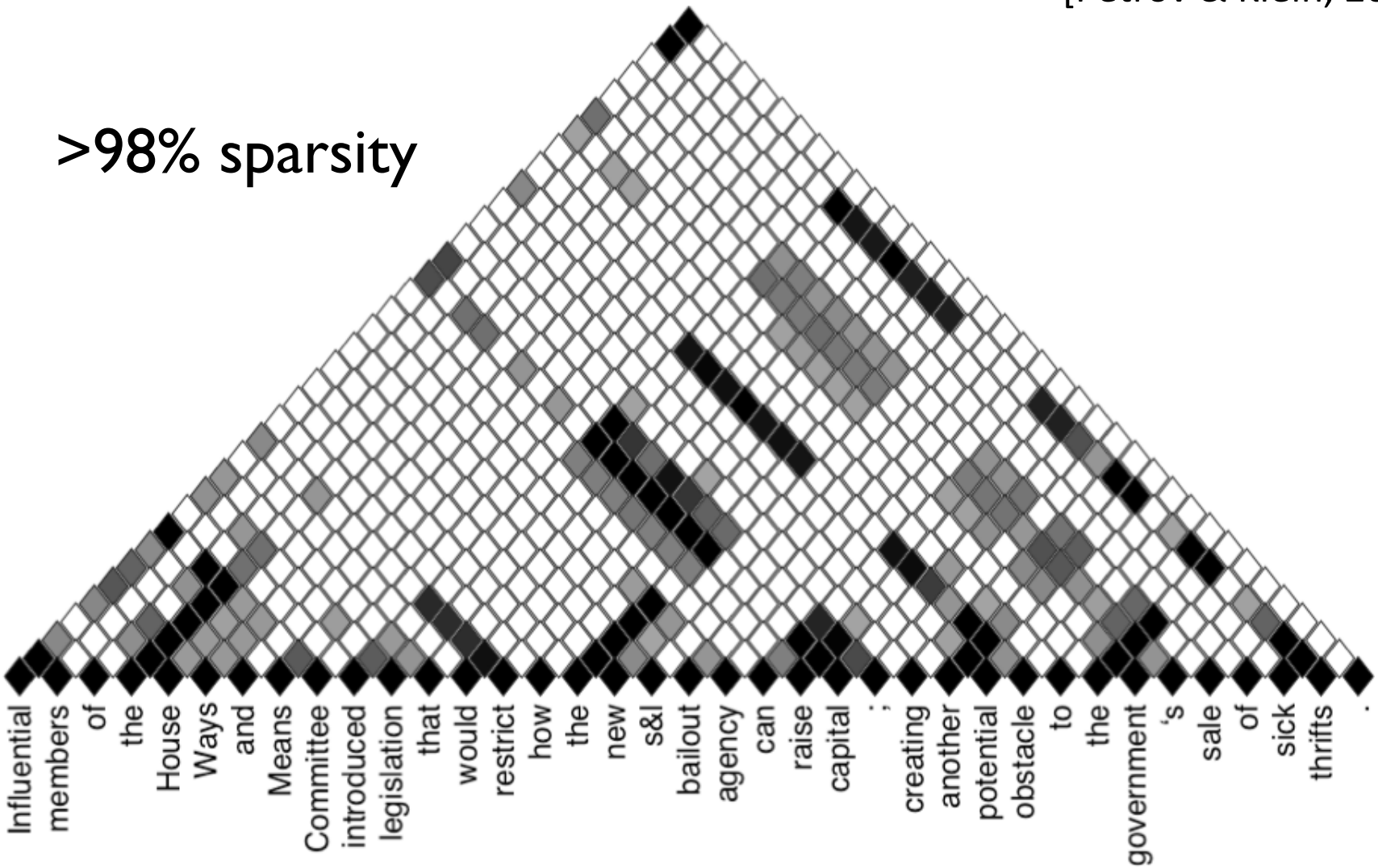
[Petrov & Klein, 2007]

Slide credit: Slav Petrov

CPU Parsing

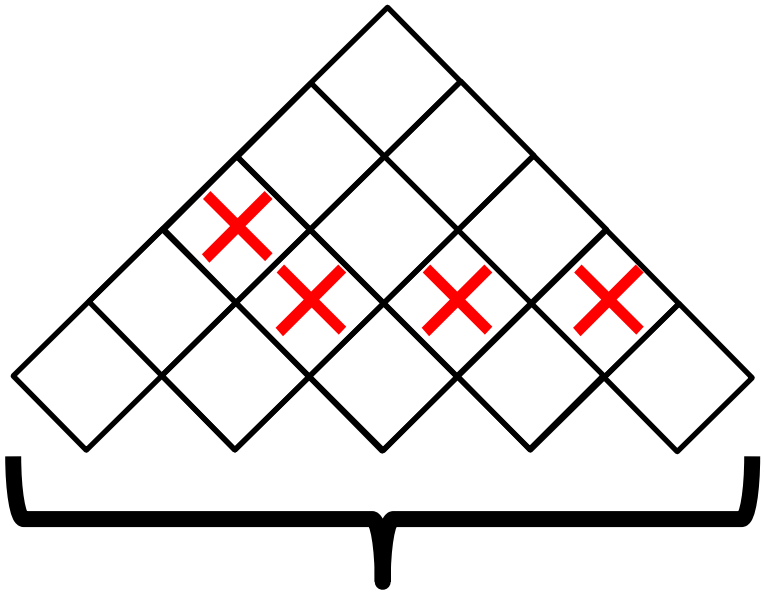
[Petrov & Klein, 2007]

>98% sparsity



Slide credit: Slav Petrov

CPU Parsing



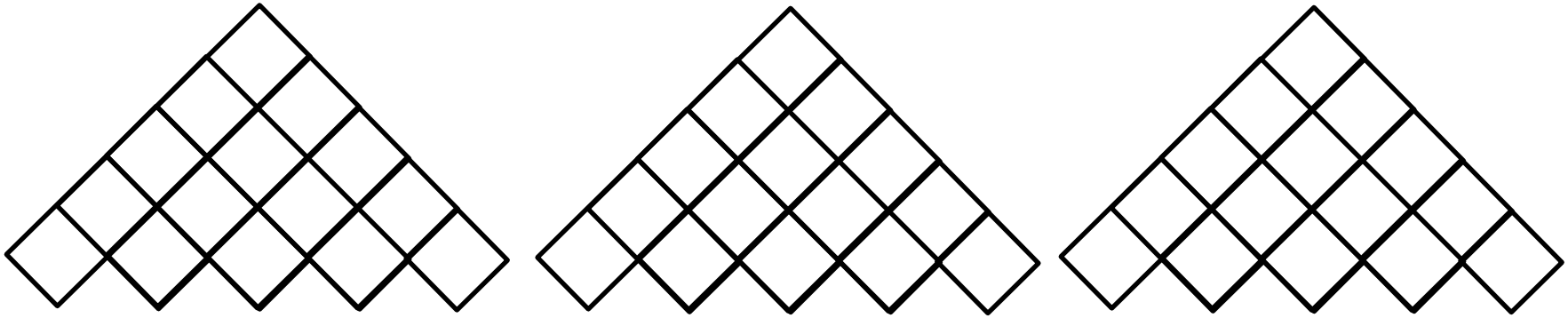
Skip Spans

Skip Rules

CPU Parsing

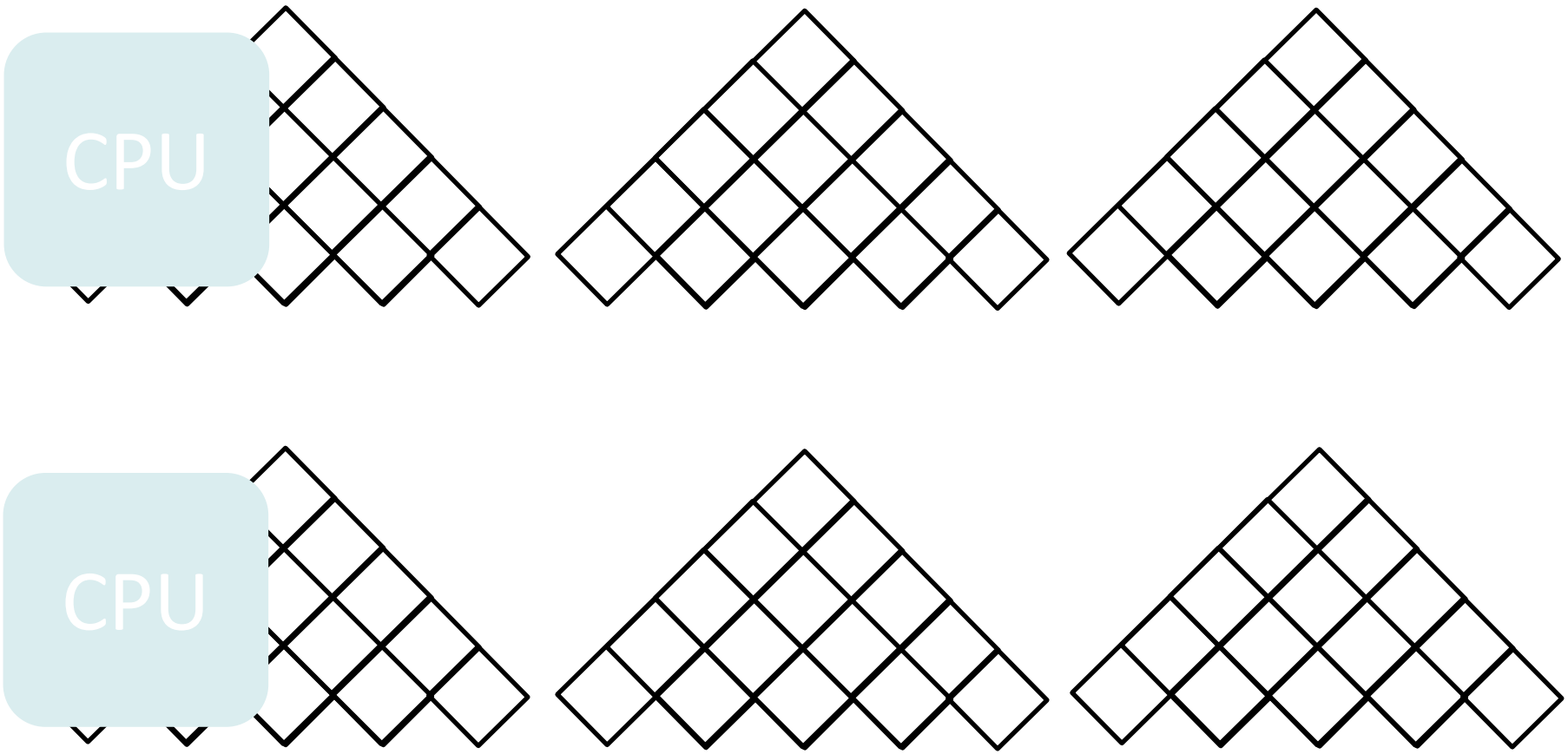
CPU

CPU Parsing



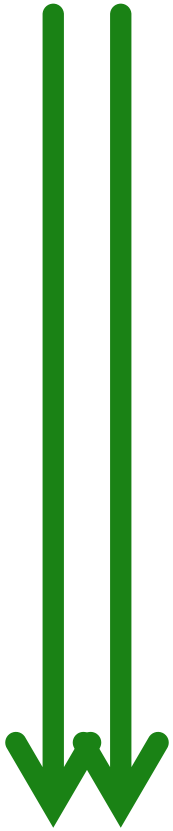
CPU

CPU Parsing

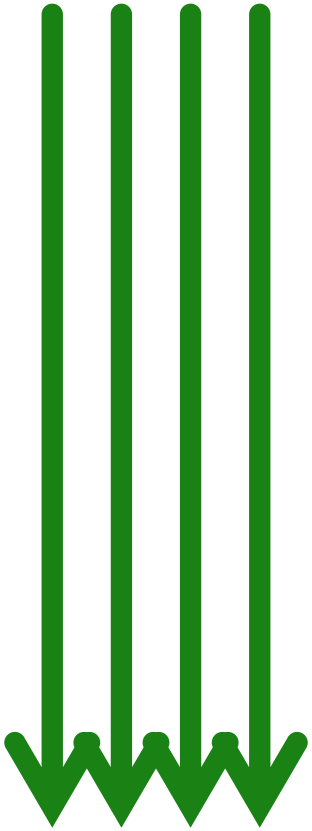


The Future of Hardware

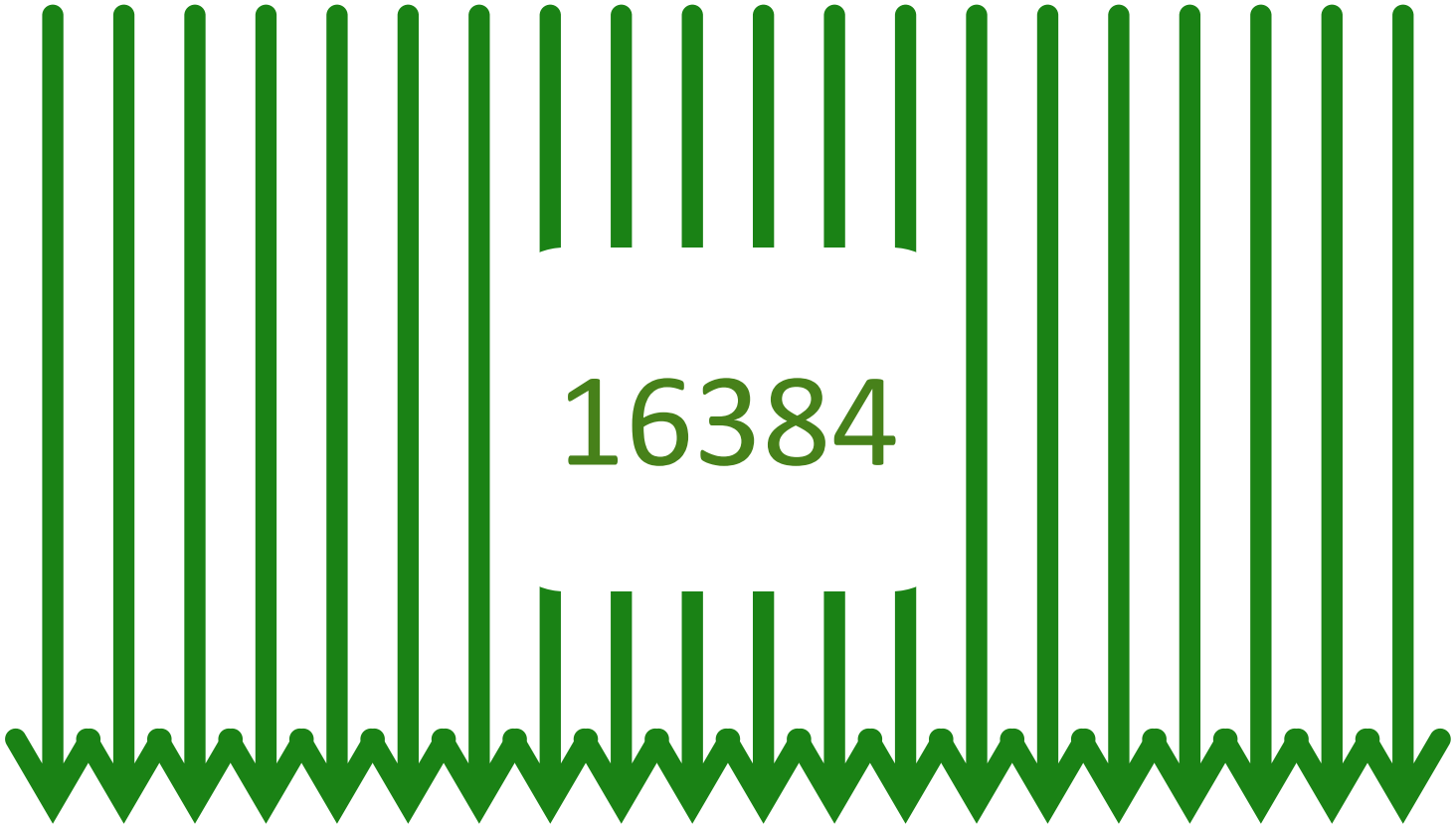
The Future of Hardware



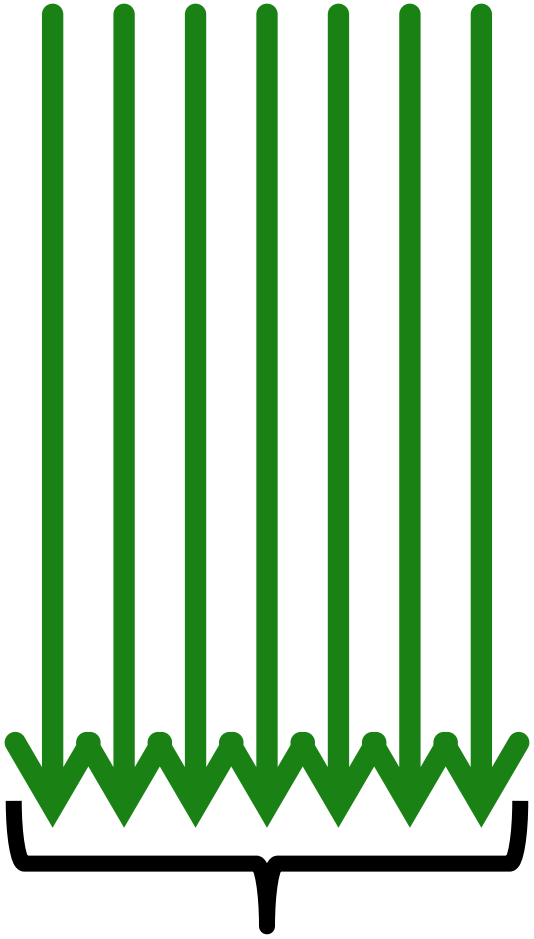
The Future of Hardware



The Future of Hardware



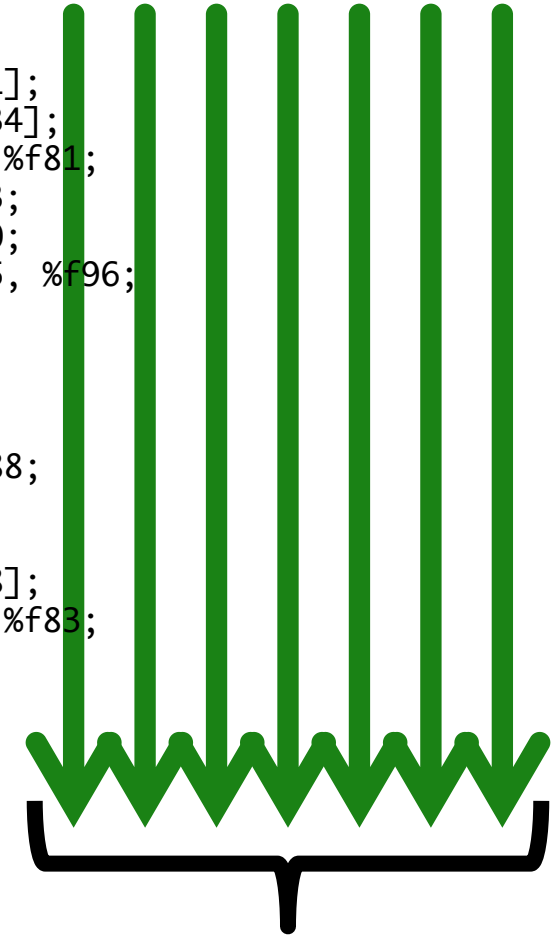
The Future of Hardware



32 Threads

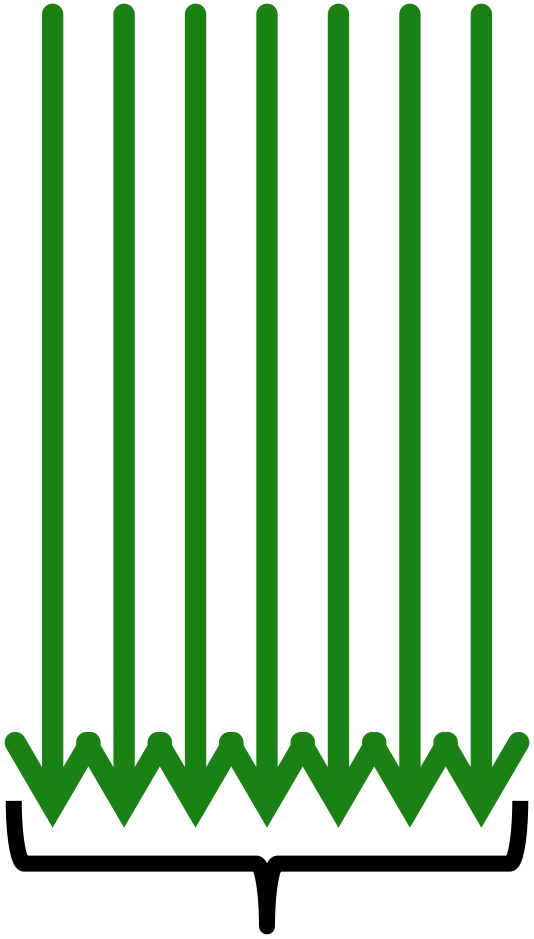
The Future of Hardware

```
add.s32   %r1, %r631, %r0;
ld.global.f32   %f81, [%r1];
ld.global.f32   %f82, [%r34];
mul.ftz.f32   %f94, %f82, %f81;
mov.f32   %f95, 0f3E002E23;
mov.f32   %f96, 0f00000000;
mad.f32   %f93, %f94, %f95, %f96;
shl.b32   %r2, %r646, 8;
add.s32   %r3, %r658, %r2;
shl.b32   %r4, %r3, 2;
add.s32   %r5, %r631, %r4;
mul.lo.s32  %r6, %r646, 588;
shl.b32   %r7, %r6, 1;
add.s32   %r8, %r5, %r7;
ld.global.f32   %f83, [%r8];
mul.ftz.f32   %f98, %f82, %f83;
```

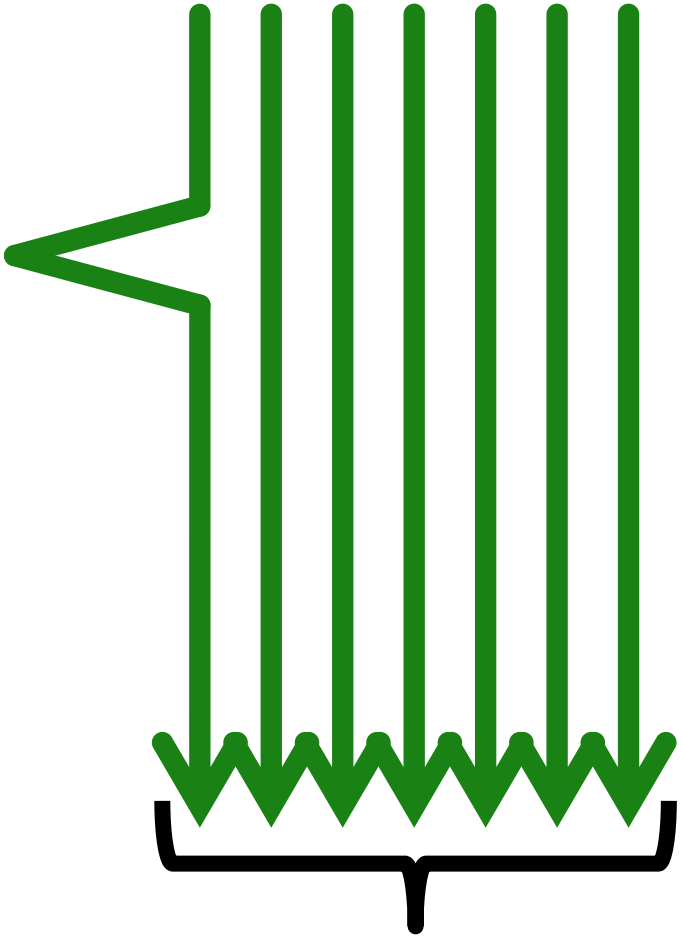


Warp

Warps

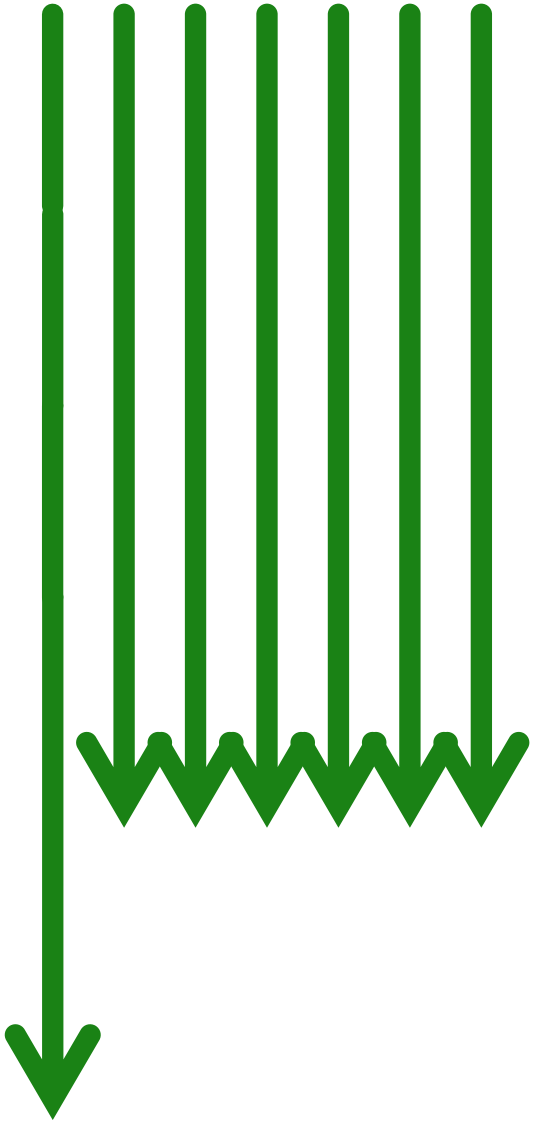


Warps

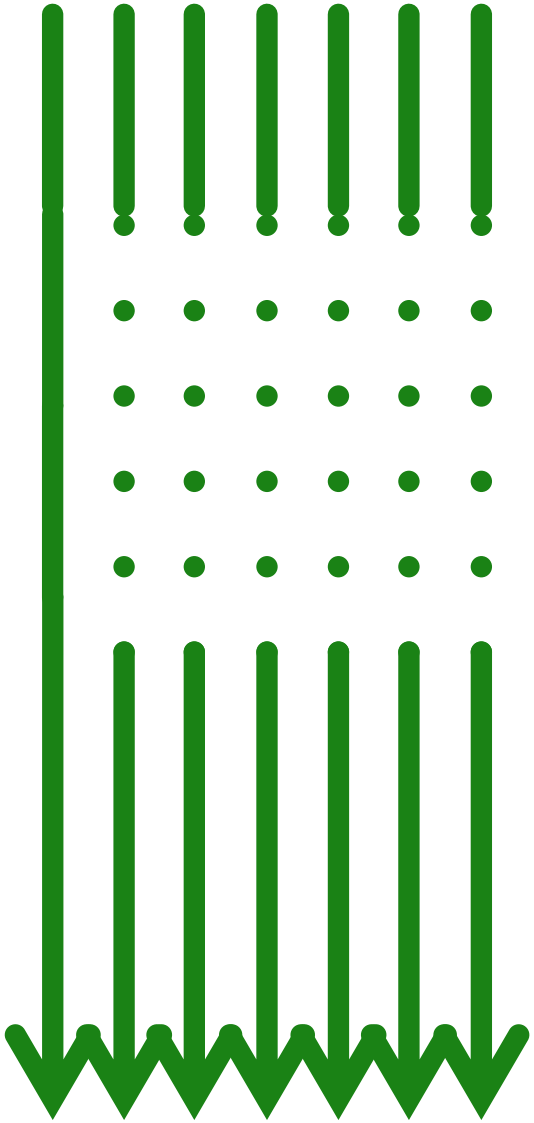


Warp Divergence

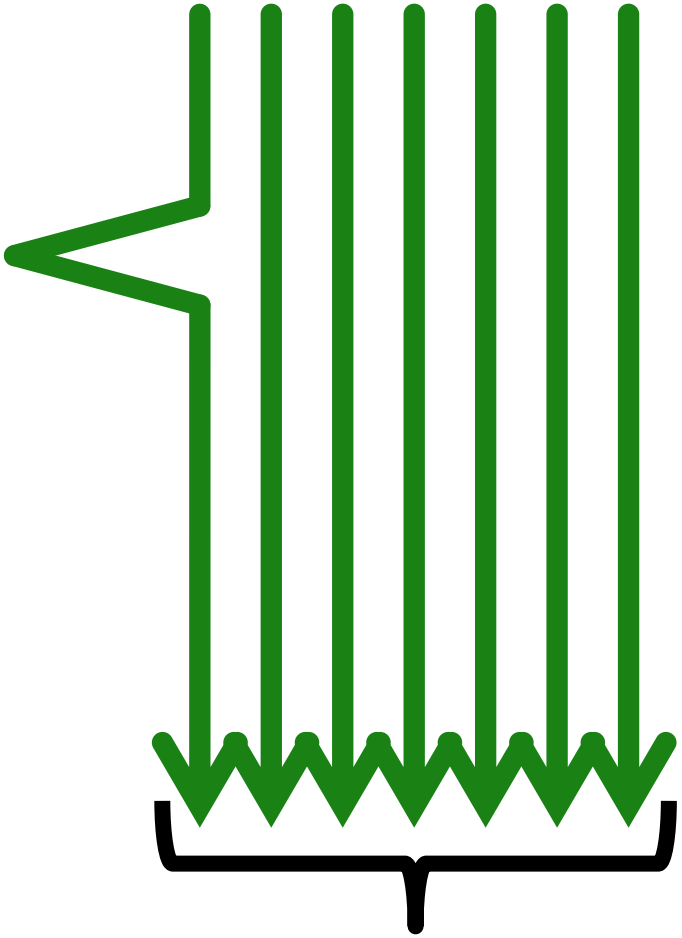
Warps



Warps

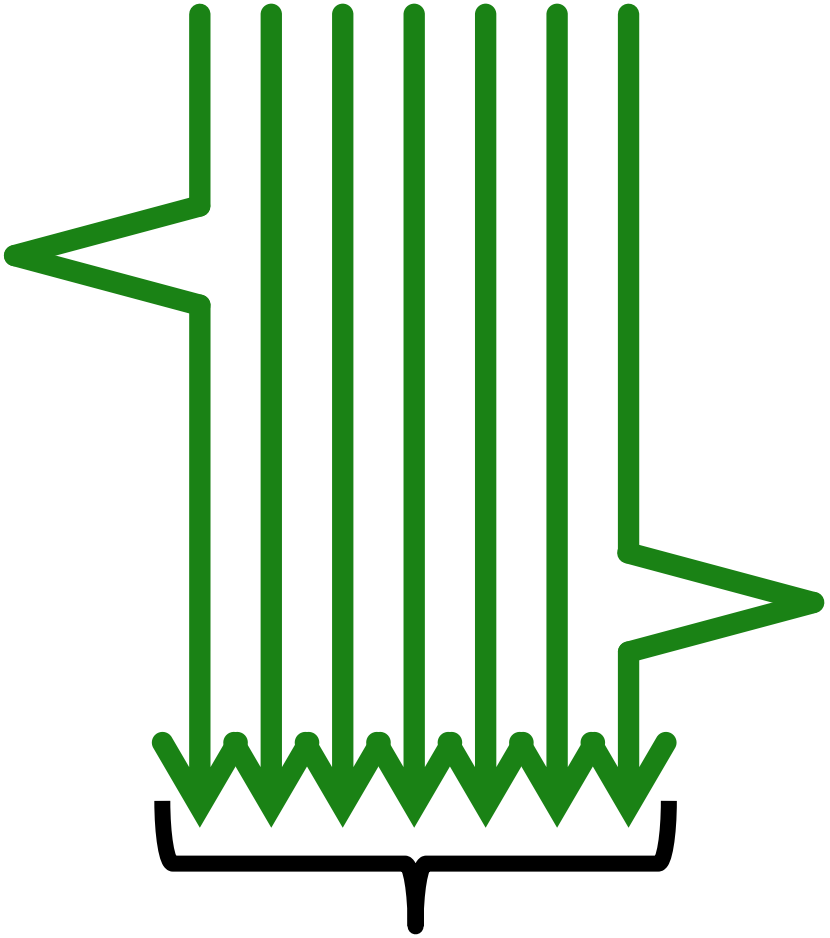


Warps



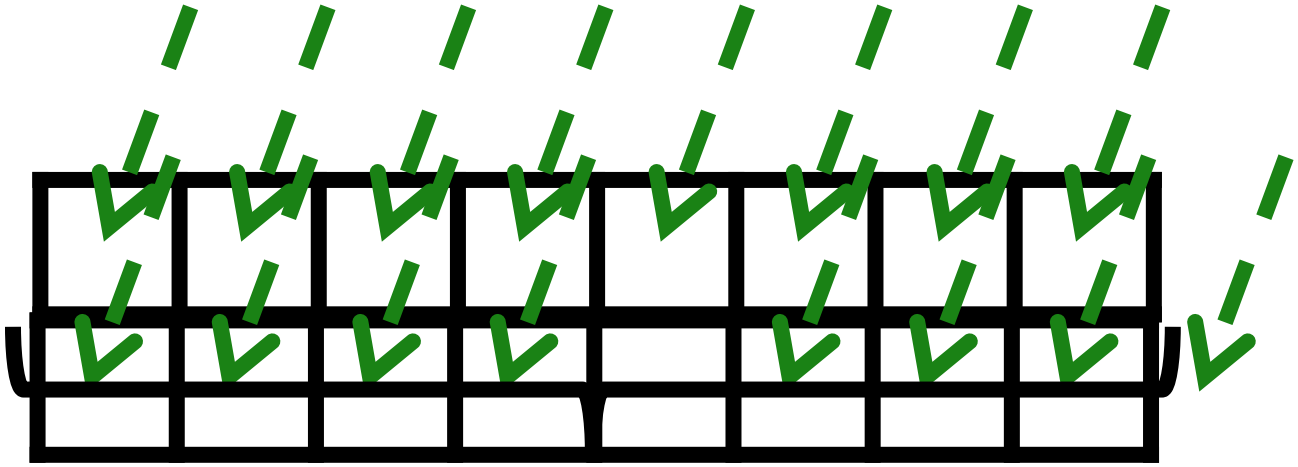
Warp Divergence

Warps



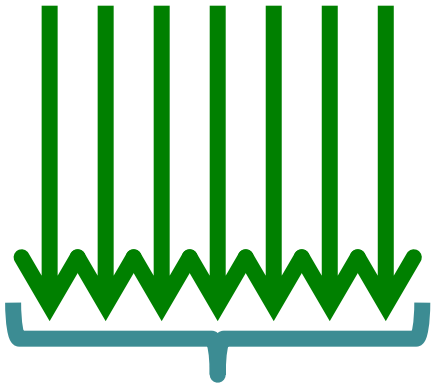
Warp Divergence

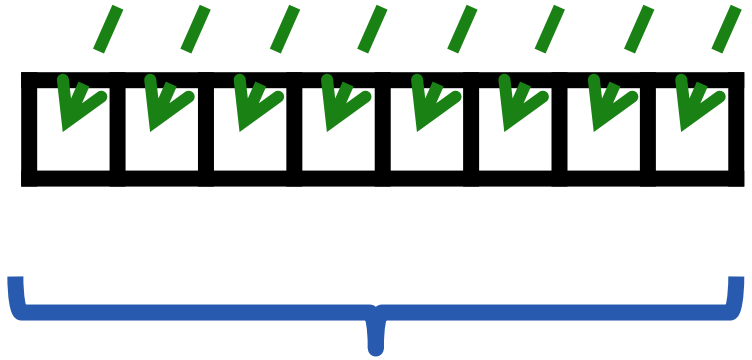
Warps

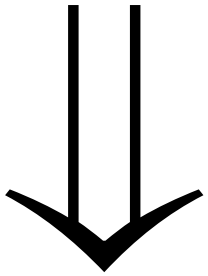


Coalescence

Designing GPU Algorithms







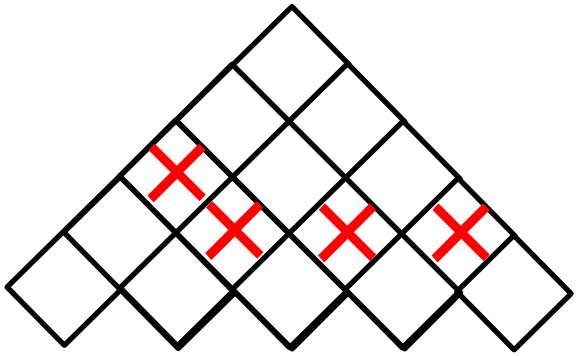
Dense, Uniform Computation

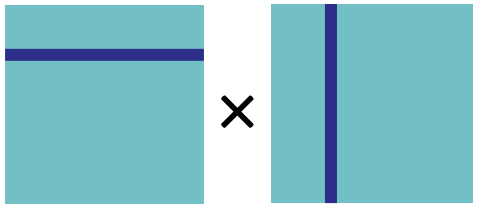
Designing GPU Algorithms

CPU

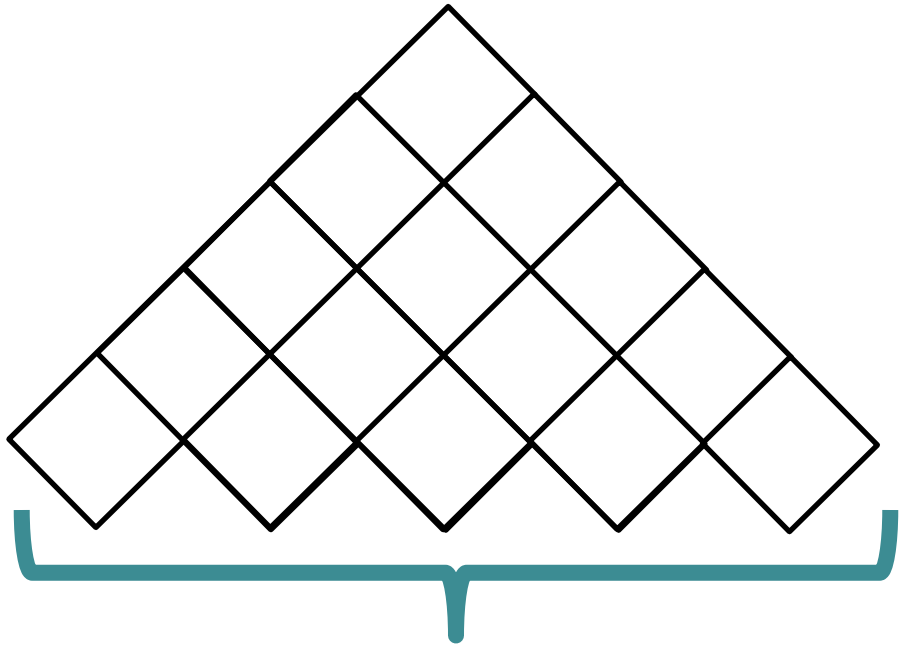
Irregular,
Sparse

Regular,
Dense





Designing GPU Algorithms



CKY Parsing

```
for each sentence:
  for each span (begin, end):
    for each split:
      for each rule (P -> L R):
        score[begin, end, P]
         += ruleScore[P -> L R]
           * score[begin, split, L]
           * score[split, end, R]
```

Item Queue

Grammar Application

CKY Parsing

```
for each sentence:
  for each span (begin, end):
    for each split:

      applyGrammar(begin, split, end)
```

Item Queue

Grammar Application

CKY Parsing

```
for each parse item in sentence:
    applyGrammar(item)
```

Item Queue

Grammar Application

CKY Parsing

```
for each parse item in sentence:
    applyGrammar(item)
```

CPU

GPU

GPU Parsing Pipeline

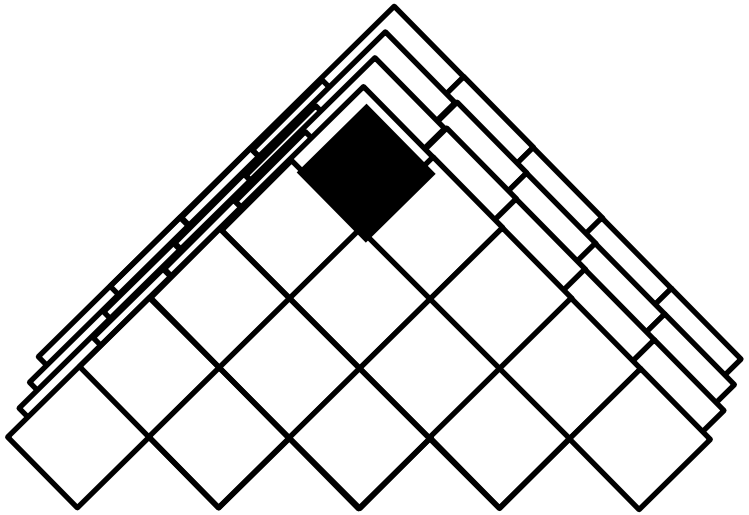
CPU

Queue

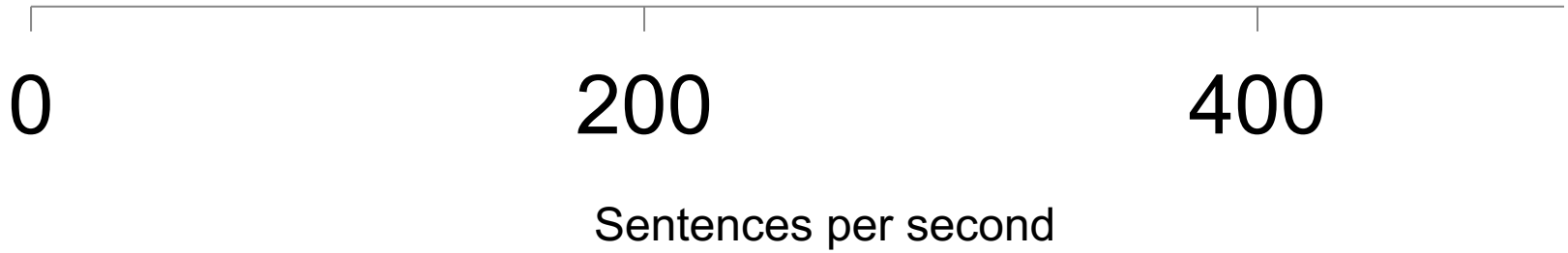
(i,	k,	j)
(0,	1,	3)
(0,	2,	3)
(1,	2,	4)
(1,	3,	4)
⋮		

GPU

Grammar

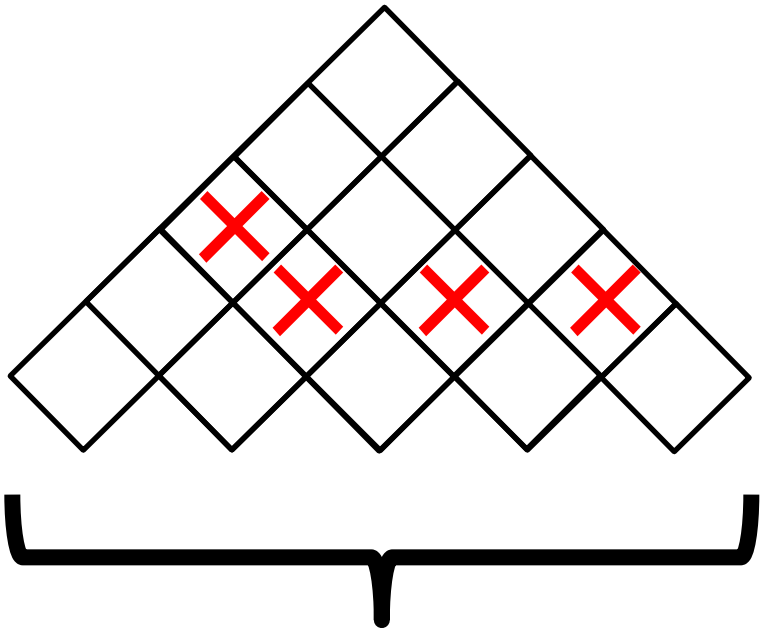


Parsing Speed

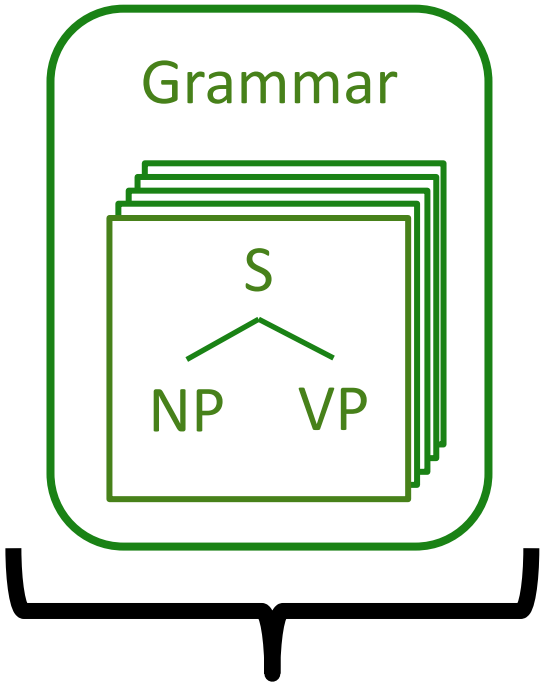


[Canny, Hall, and Klein, 2013]

Exploiting Sparsity

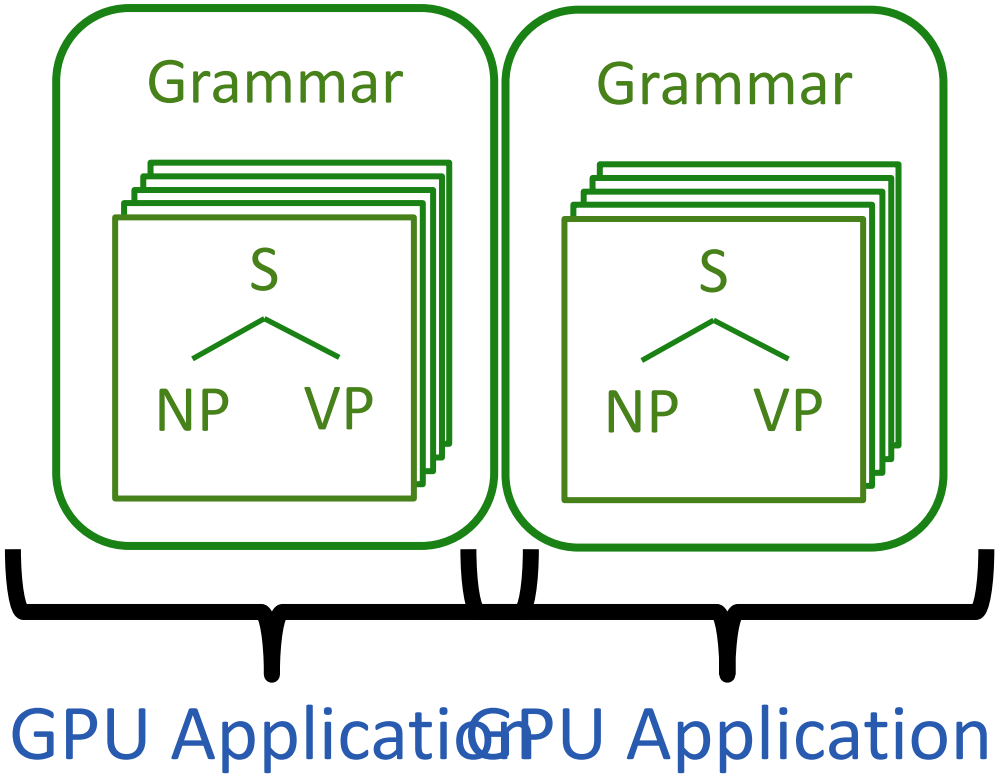


CPU Queuing

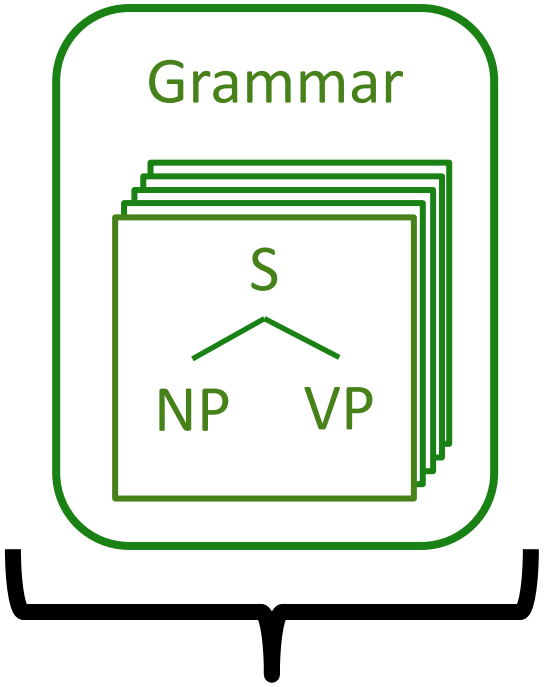


GPU Application

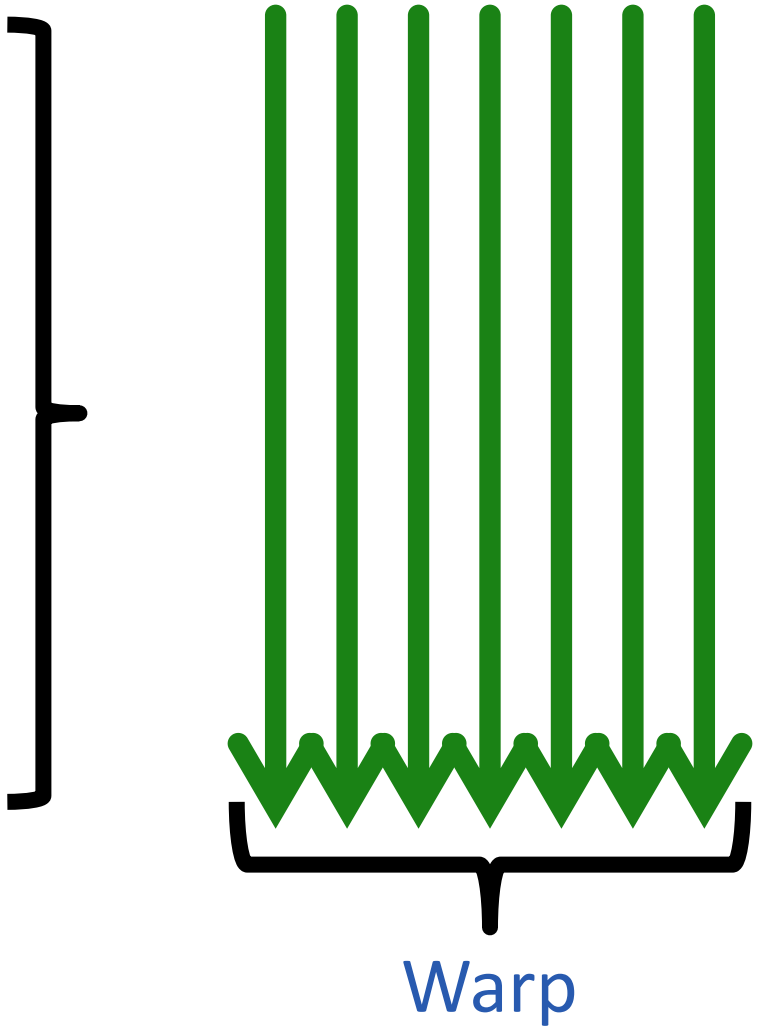
Exploiting Sparsity



Exploiting Sparsity

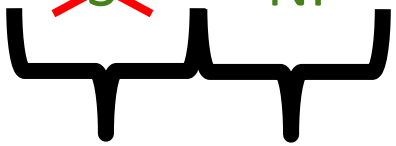


Exploiting Sparsity

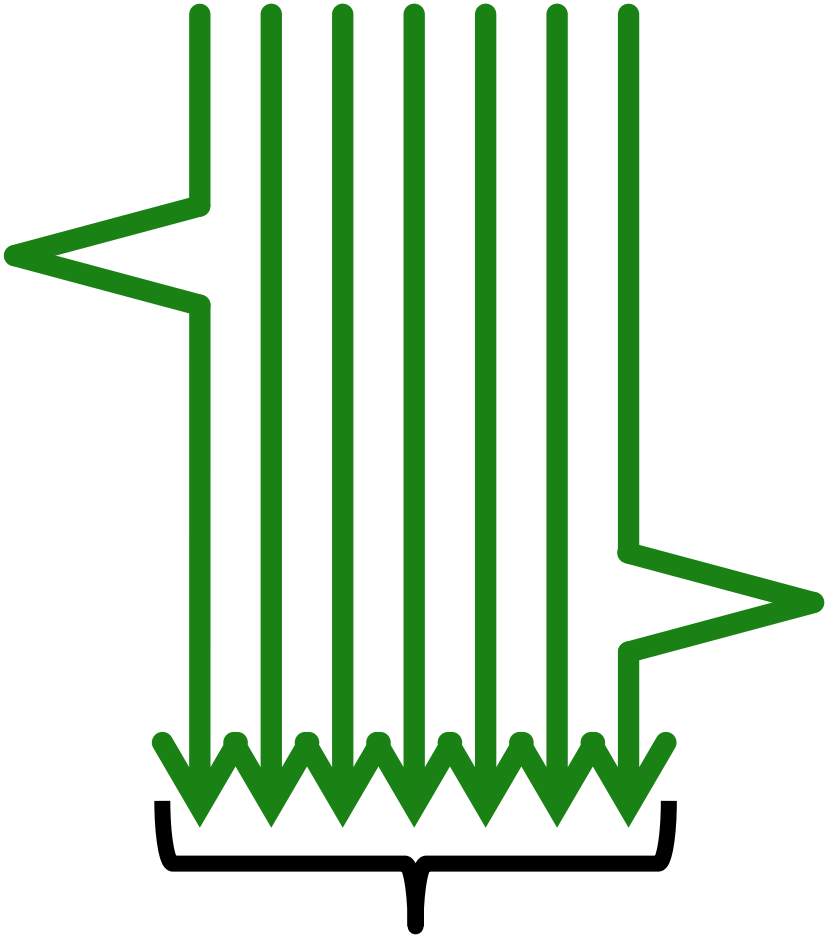


Exploiting Sparsity

(0, 1, 3)	~~S~~	NP	~~VP~~	~~PP~~	…
(0, 2, 3)	~~S~~	~~NP~~	VP	~~PP~~	…
(1, 2, 4)	~~S~~	NP	~~VP~~	PP	…
(1, 3, 4)	~~S~~	NP	VP	~~PP~~	…
(2, 3, 5)	~~S~~	NP	VP	~~PP~~	…
(2, 4, 5)	~~S~~	NP	~~VP~~	PP	…
(3, 4, 6)	~~S~~	NP	~~VP~~	PP	…
⋮					

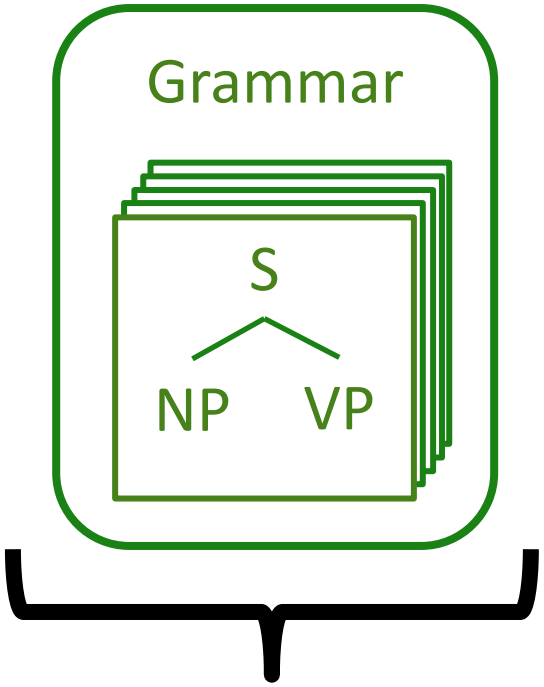


Exploiting Sparsity

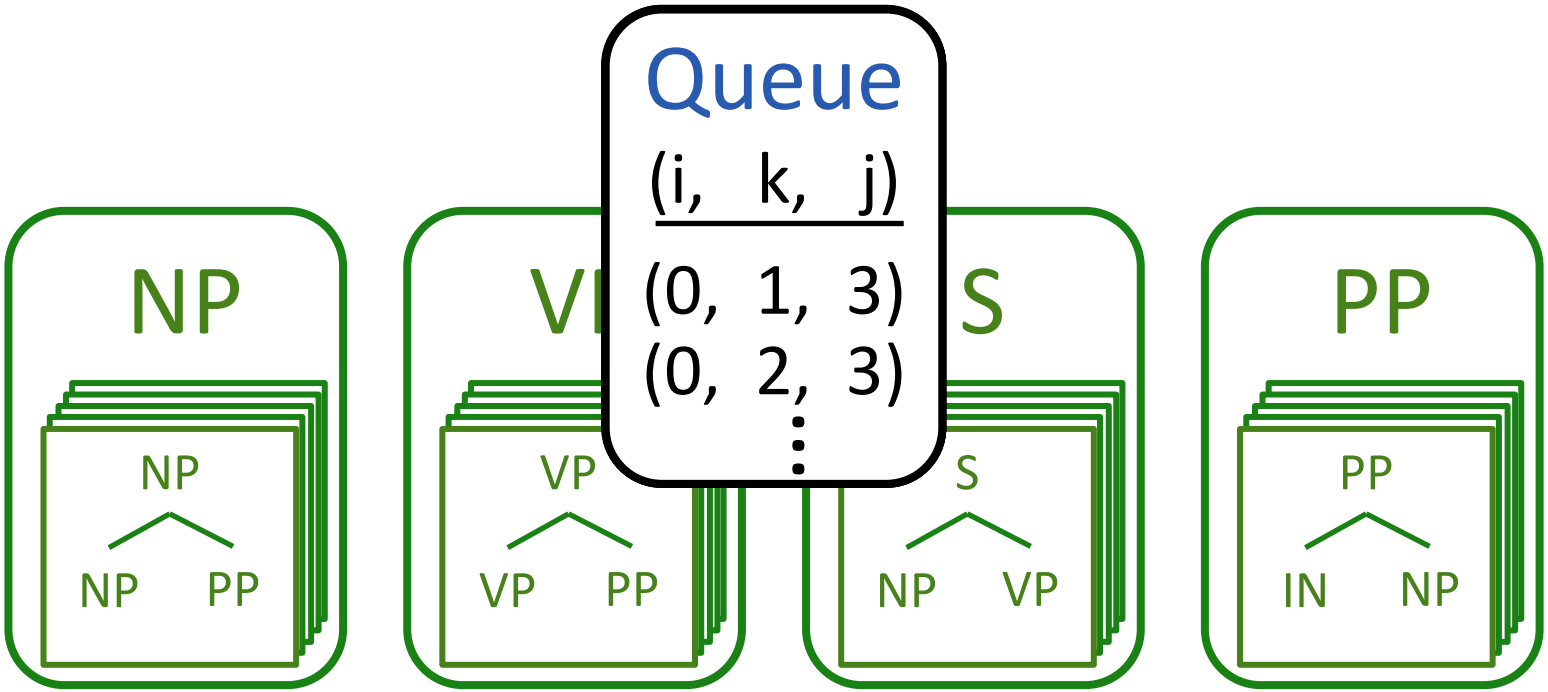


Warp Divergence

Exploiting Sparsity



Exploiting Sparsity



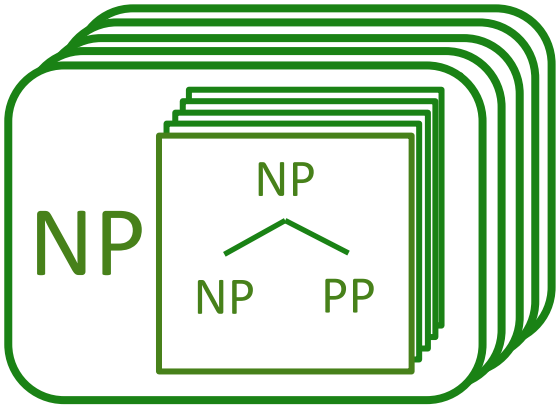
Exploiting Sparsity

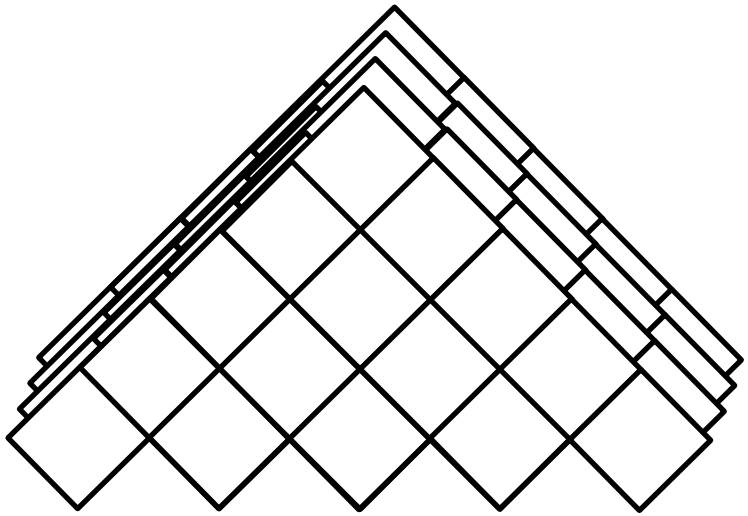
CPU

NP Queue

(i, k, j)
(0, 1, 3)
(0, 2, 3)
~~(1, 2, 4)~~
~~(1, 3, 4)~~
⋮

GPU





Exploiting Sparsity

CPU

VP Queue

(i, k, j)

~~(0, 1, 3)~~

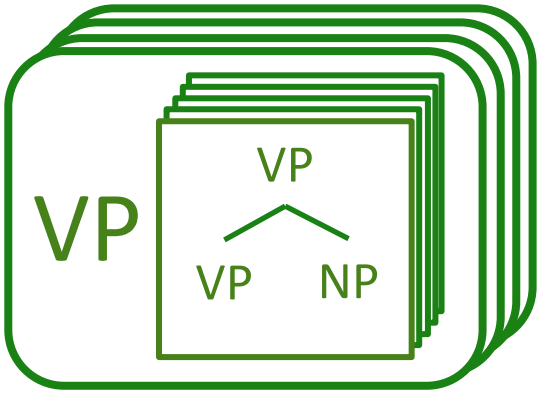
~~(0, 2, 3)~~

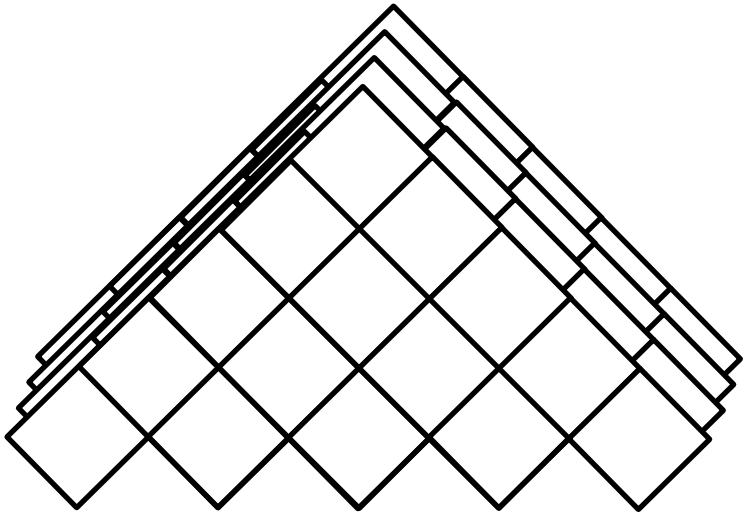
(1, 2, 4)

~~(1, 3, 4)~~

⋮

GPU





Parsing Speed