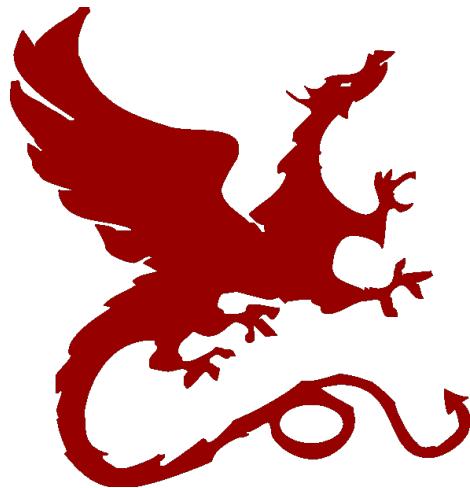


Algorithms for NLP



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

ADV
lo haré de muy buen grado .

Output

ADV
|
gladly

Grammar

ADV → < de muy buen grado ; gladly >

Translating with Tree Transducers

Input

ADV
lo haré de muy buen grado .

Output

ADV
|
gladly

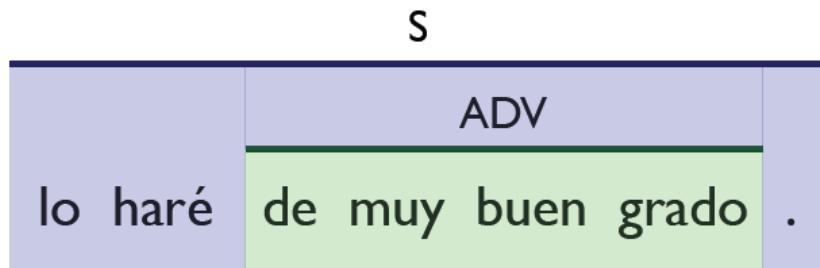
Grammar

$s \rightarrow \langle \text{lo haré ADV . ; I will do it ADV .} \rangle$

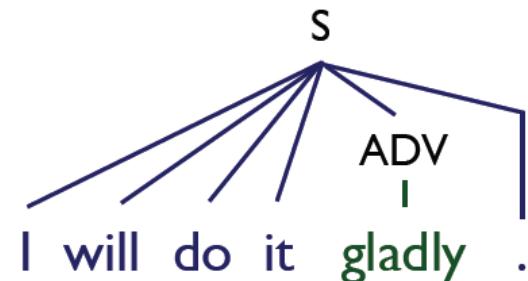
$\text{ADV} \rightarrow \langle \text{de muy buen grado ; gladly} \rangle$

Translating with Tree Transducers

Input



Output



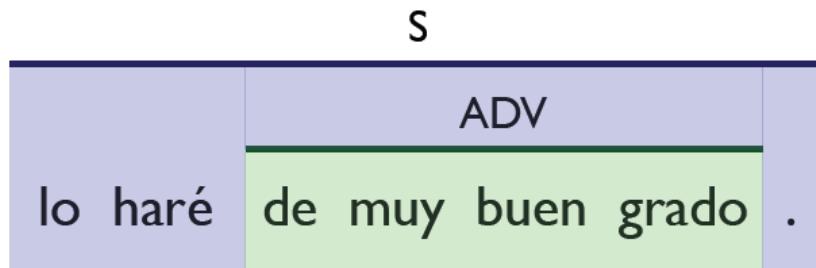
Grammar

$s \rightarrow \langle \text{lo haré ADV .} ; \text{I will do it ADV .} \rangle$

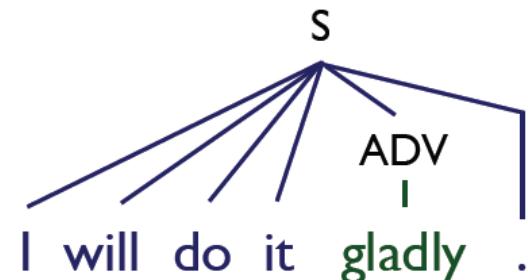
$\text{ADV} \rightarrow \langle \text{de muy buen grado} ; \text{gladly} \rangle$

Translating with Tree Transducers

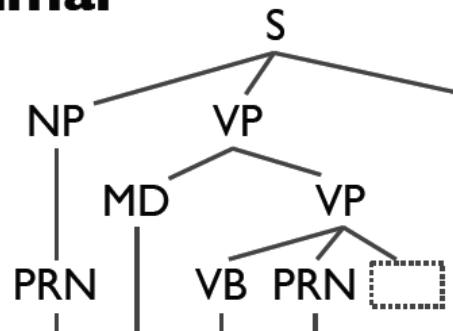
Input



Output



Grammar



$s \rightarrow \langle \text{lo haré} \text{ ADV} . ; \text{ I will do it ADV} . \rangle$

$\text{ADV} \rightarrow \langle \text{de muy buen grado} ; \text{ gladly} \rangle$

Translating with Tree Transducers

Input

ADV
lo haré de muy buen grado .

Output

ADV
|
gladly

Grammar

$s \rightarrow \langle \text{lo haré ADV . ; I will do it ADV .} \rangle$

$\text{ADV} \rightarrow \langle \text{de muy buen grado ; gladly} \rangle$

Translating with Tree Transducers

Input

ADV
lo haré de muy buen grado .

Output

ADV
|
gladly

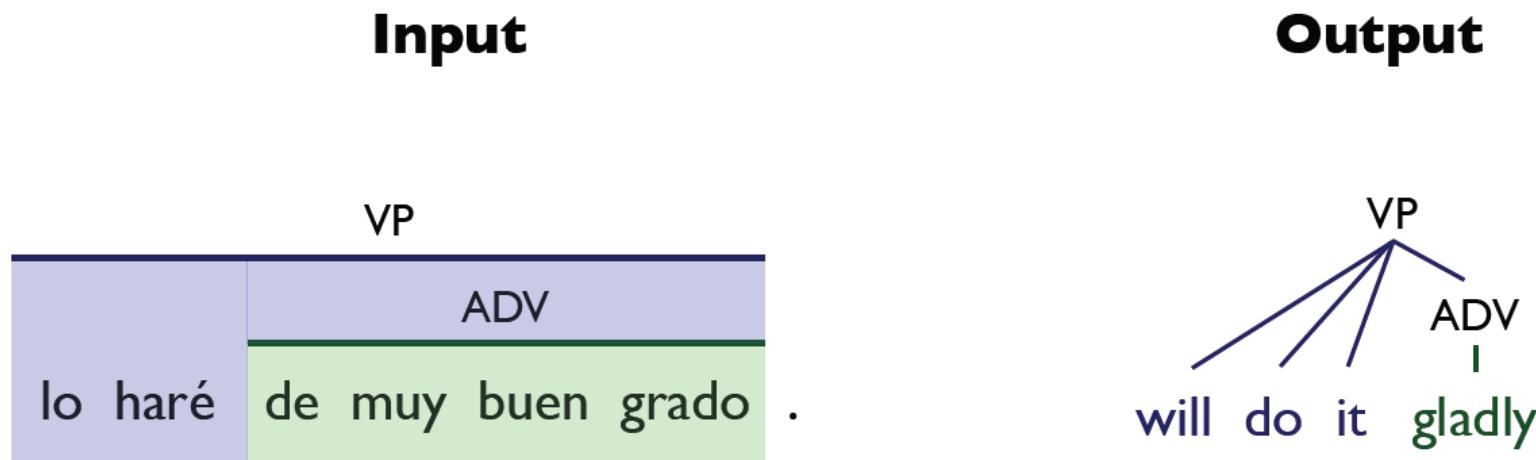
Grammar

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



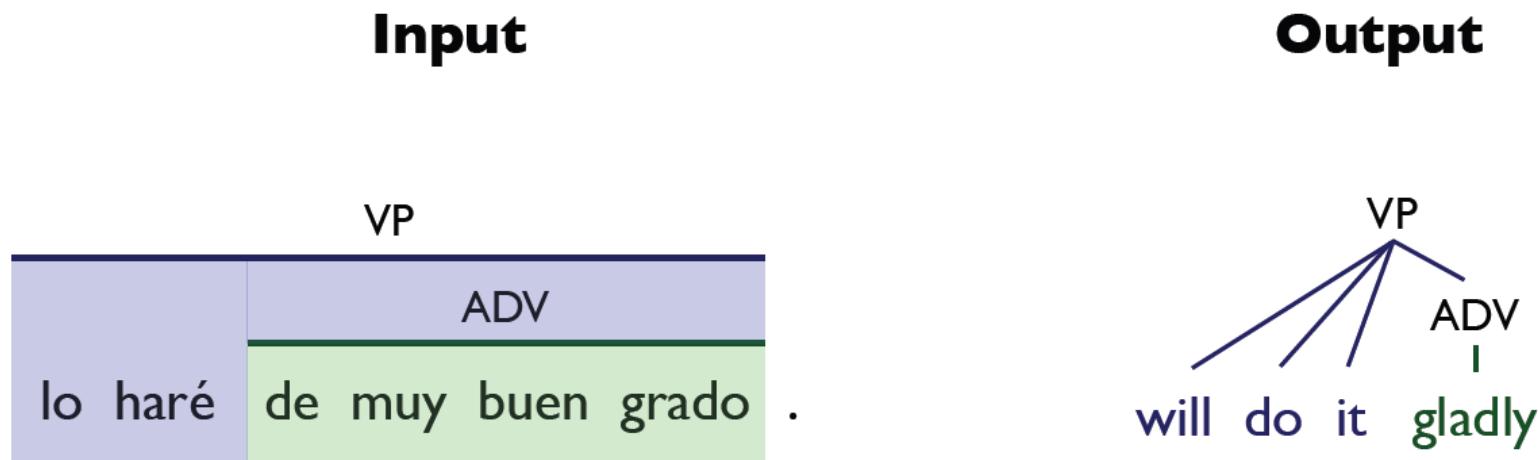
Grammar

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



Grammar

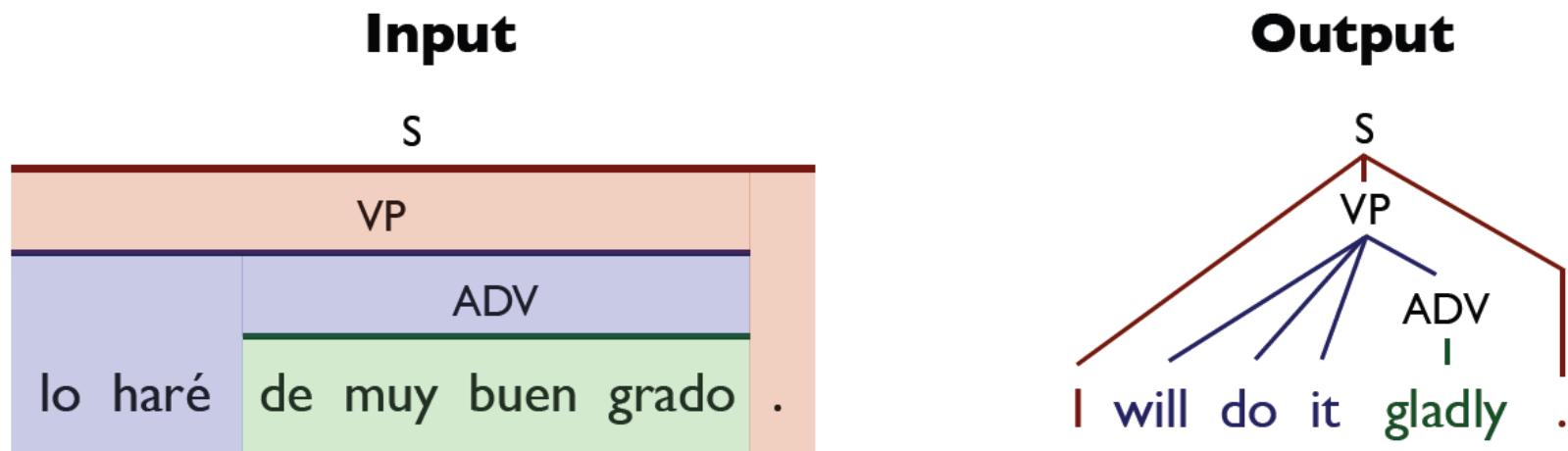
$$S \rightarrow \langle VP . ; | VP . \rangle$$

$$VP \rightarrow \langle lo\;haré\;ADV ; will\;do\;it\;ADV \rangle$$

$$S \rightarrow \langle lo\;haré\;ADV . ; | will\;do\;it\;ADV . \rangle$$

$$ADV \rightarrow \langle de\;muy\;buen\;grado ; gladly \rangle$$

Translating with Tree Transducers



Grammar

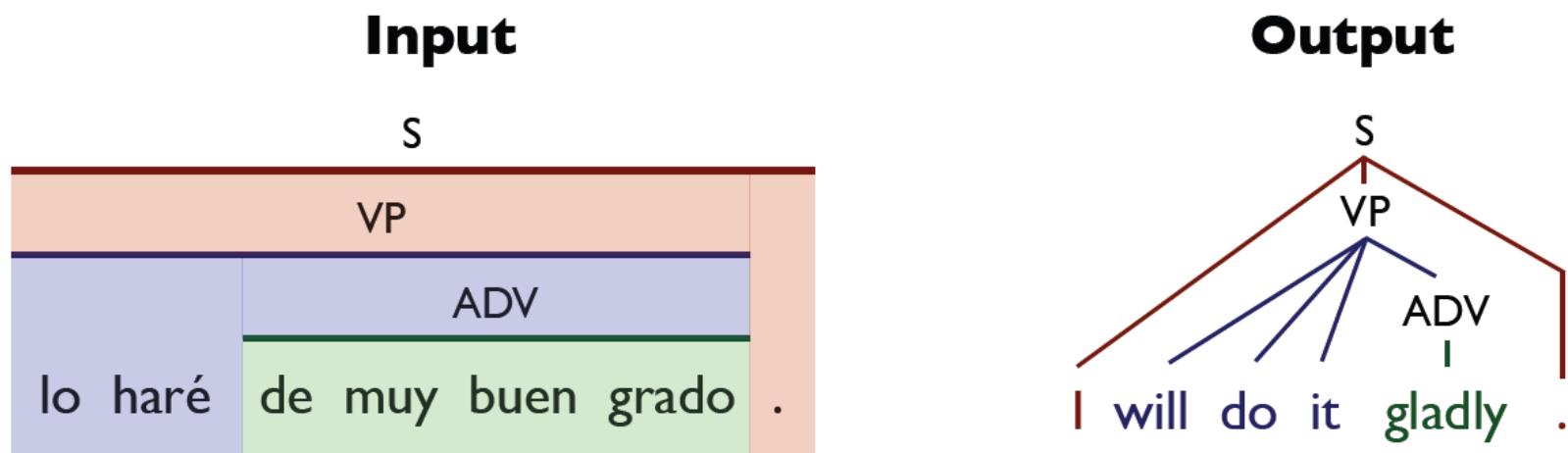
$$S \rightarrow \langle VP . ; I VP . \rangle$$

$$VP \rightarrow \langle lo\;haré\;ADV ; will\;do\;it\;ADV \rangle$$

$$S \rightarrow \langle lo\;haré\;ADV . ; I\;will\;do\;it\;ADV . \rangle$$

$$ADV \rightarrow \langle de\;muy\;buen\;grado ; gladly \rangle$$

Translating with Tree Transducers



Grammar

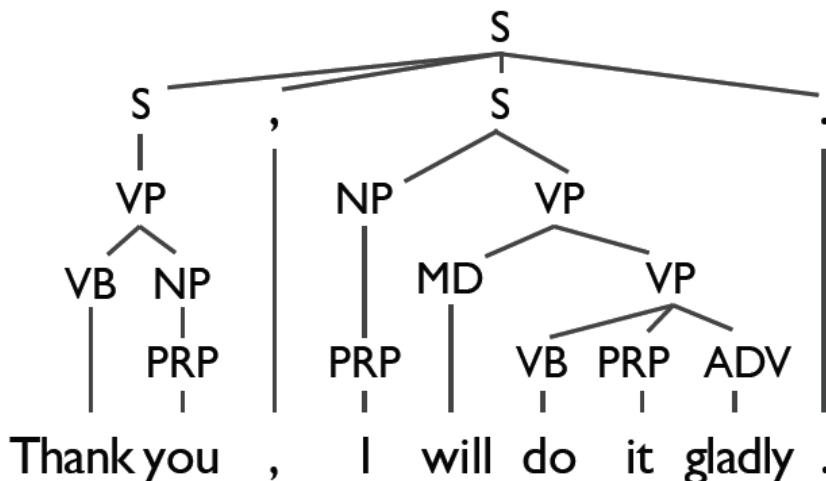
$s \rightarrow \langle VP . ; I VP . \rangle$ OR $s \rightarrow \langle VP . ; you VP . \rangle$

$VP \rightarrow \langle lo\ haré\ ADV ; will\ do\ it\ ADV \rangle$

$s \rightarrow \langle lo\ haré\ ADV . ; I\ will\ do\ it\ ADV . \rangle$

$ADV \rightarrow \langle de\ muy\ buen\ grado ; gladly \rangle$

Learning Grammars for Translation



Gracias
,

lo

haré

de

muy

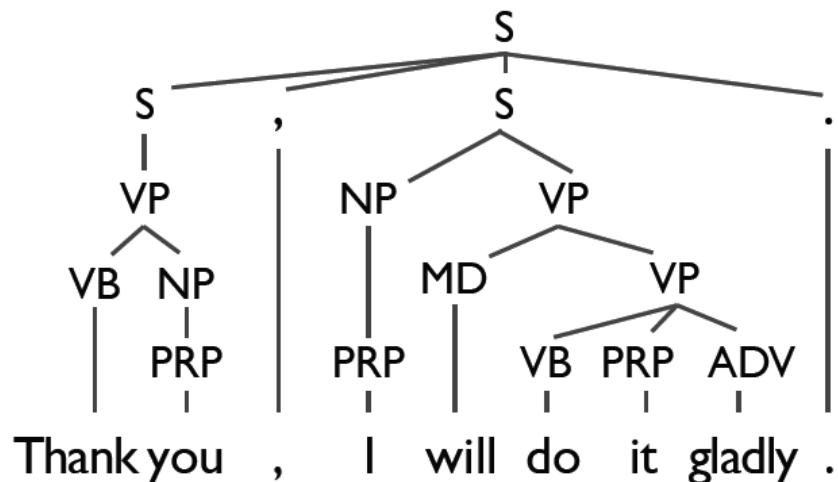
buen

grado

.

Grammar Rules

Learning Grammars for Translation



Gracias
,

lo

haré

de

muy

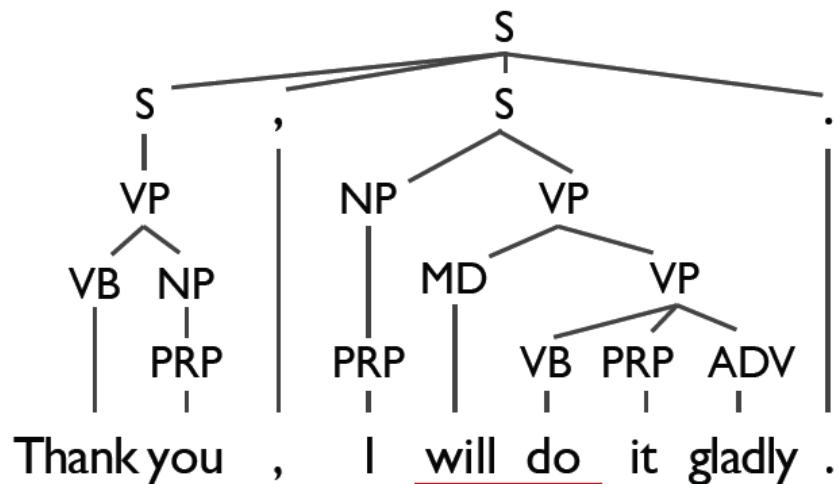
buen

grado

.

Grammar Rules

Learning Grammars for Translation



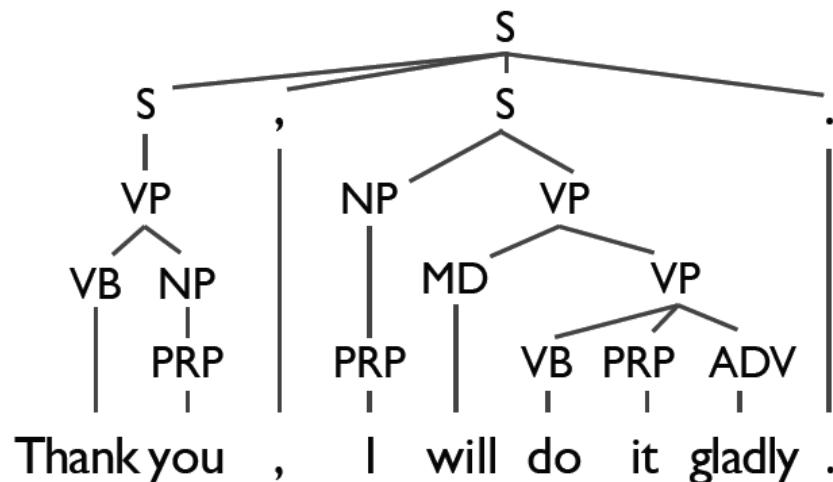
The diagram illustrates a convolutional operation on a 10x10 input grid. The input grid consists of 100 white squares arranged in a 10x10 pattern. A 3x3 kernel, highlighted by a blue shaded area, is applied to the input. The output layer, shown below the input, has 8 columns and 8 rows. The first row of the output layer contains three brown squares, representing the result of the convolution step. The remaining squares in the output layer are white, indicating they have not yet been processed by the kernel.

Gracias,
lo
haré
de
muy
buen
grado

Grammar Rules

⟨haré ; will do⟩

Learning Grammars for Translation



Gracias
,

lo

haré

de

muy

buen

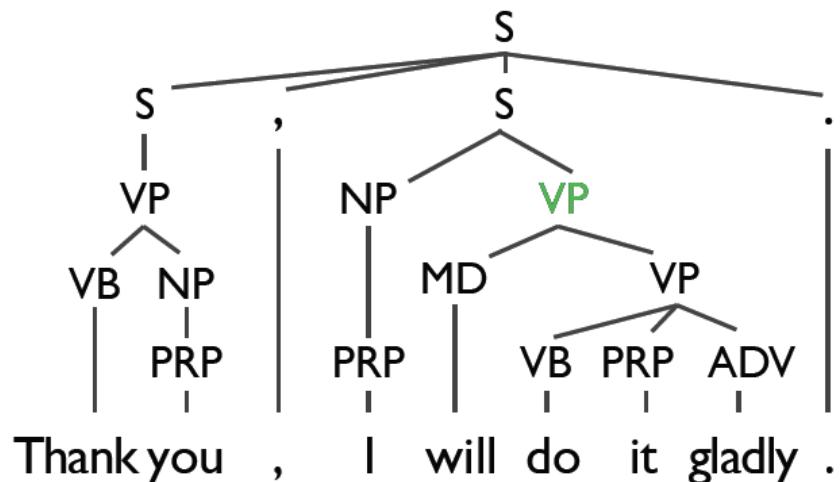
grado

.

Grammar Rules

~~⟨haré ; will do⟩~~

Learning Grammars for Translation



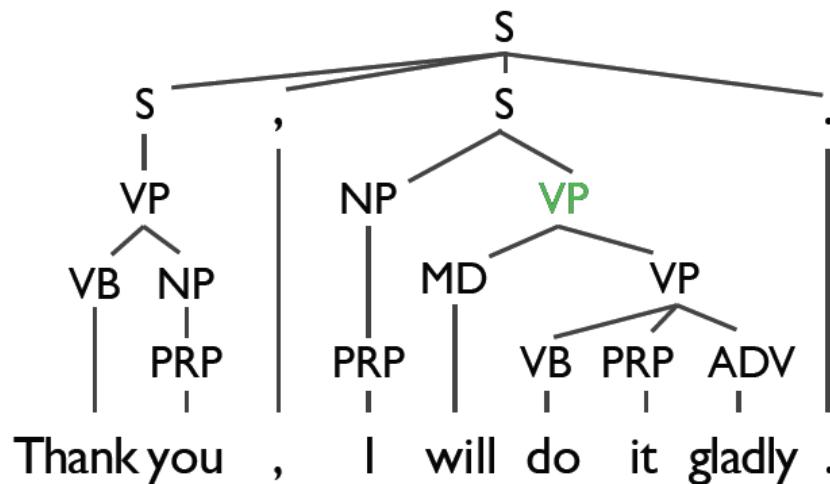
The diagram illustrates a convolutional operation mapping a larger input layer to a smaller output layer. The input layer consists of 10 columns and 10 rows, with colors alternating in a repeating pattern of white, light blue, and brown. A 5x5 kernel is applied to the input, resulting in a 5x5 output layer in the center, colored entirely in light blue. The output layer is highlighted with a thick blue border.

Gracias,
lo
haré
de
muy
buen
grado

Grammar Rules

~~⟨haré ; will do⟩~~

Learning Grammars for Translation



The diagram illustrates a convolutional operation mapping a larger input layer to a smaller output layer. The input layer has 10 columns of brown squares. The output layer has 5 columns of blue squares. A 5x5 kernel, highlighted with a blue border, is applied to the first column of the input to produce the first column of the output. The result is shown as a 5x5 grid of blue squares.

Gracias,
lo
haré
de
muy
buen
grado

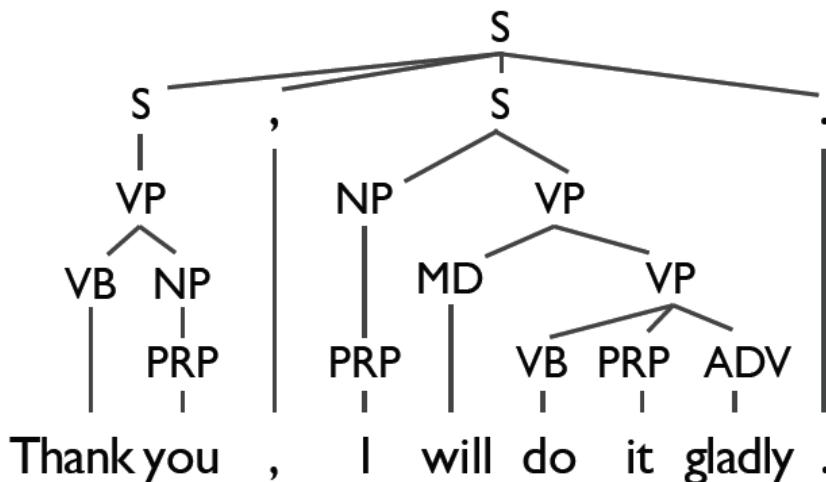
Grammar Rules

~~haré ; will do~~

VP →

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

Learning Grammars for Translation



A 10x10 grid diagram illustrating a convolution operation. The input layer (left) has two blue cells at positions (1,1) and (2,2). A 5x3 kernel (right) with a red border is applied. The output layer (bottom) shows the result of the convolution. The output cells are blue, and the output layer has dimensions 5x2.

Gracias,
lo
haré
de
muy
buen
grado

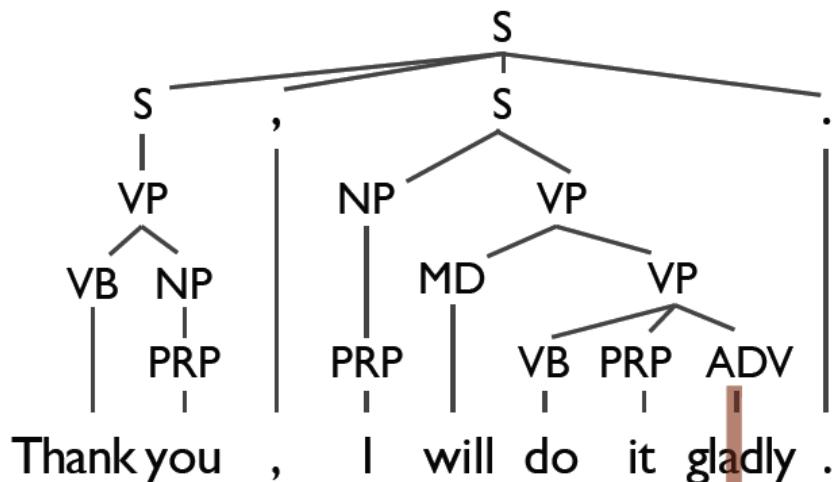
Grammar Rules

~~haré ; will do~~

VP →

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

Learning Grammars for Translation



The diagram illustrates a path through a 10x10 grid. The grid consists of 100 squares arranged in 10 rows and 10 columns. The path starts at the bottom-right square (row 10, column 10) and ends at the top-left square (row 1, column 1). The path is composed of several segments: a vertical segment down the rightmost column, a horizontal segment across the bottom row, a diagonal segment from (9, 9) to (7, 7), a vertical segment down column 7, a horizontal segment across row 7, and finally a diagonal segment from (6, 6) to (1, 1).

Gracias
,
lo
haré
de
muy
buen
grado

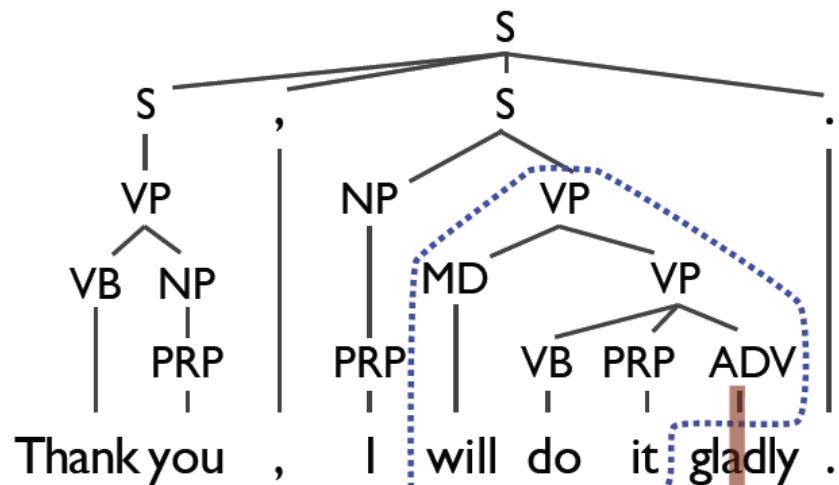
Grammar Rules

~~haré ; will do~~

VP →

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

Learning Grammars for Translation



The diagram illustrates a path through a 10x10 grid. The path starts at the top-left corner (blue square) and ends at the bottom-right corner (blue square). The path consists of several segments: a horizontal segment of three squares to the right, a vertical segment of four squares down, another horizontal segment of three squares to the right, and finally a vertical segment of four squares down to the target. The path is highlighted with a thick red outline. A brown arrow points from the start to the end of the path.

Gracias,
lo
haré
de
muy
buen
grado

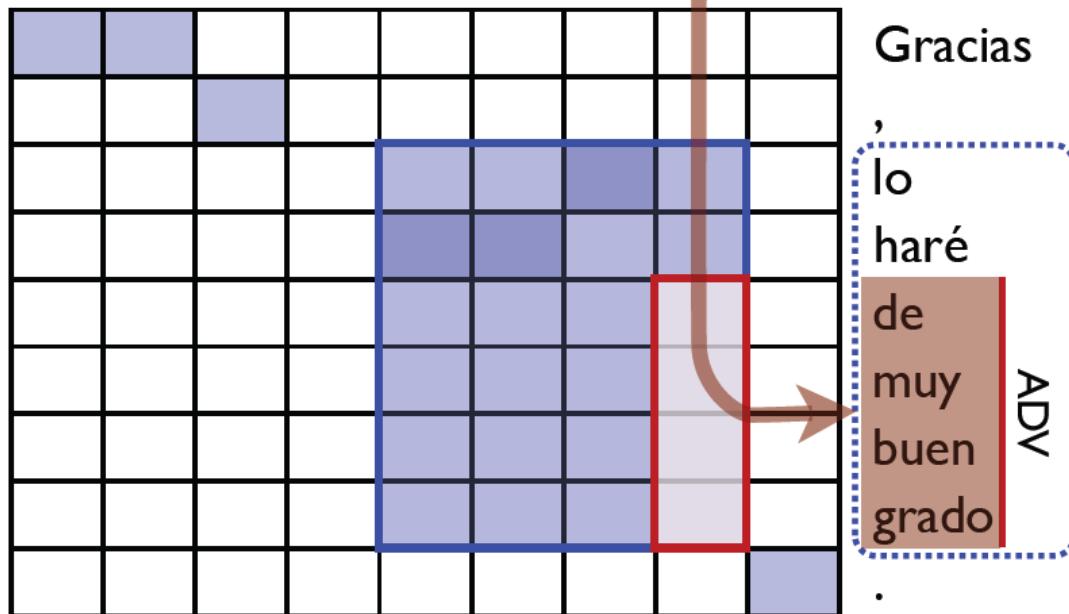
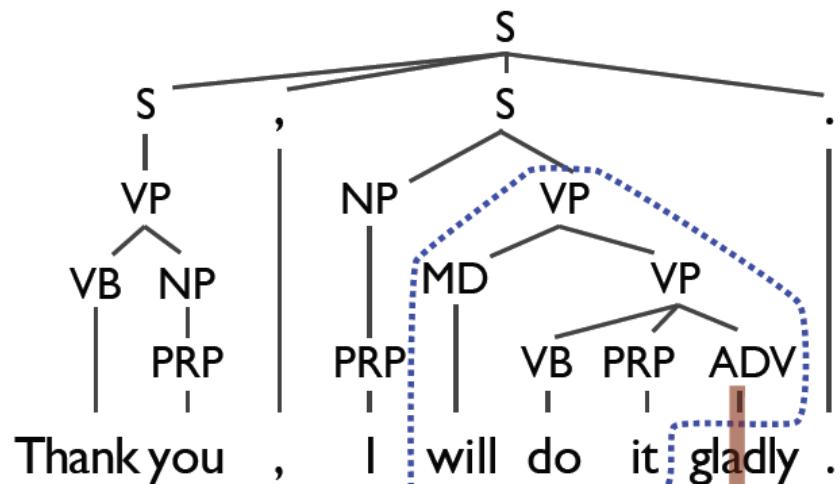
Grammar Rules

~~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⟩

VP →

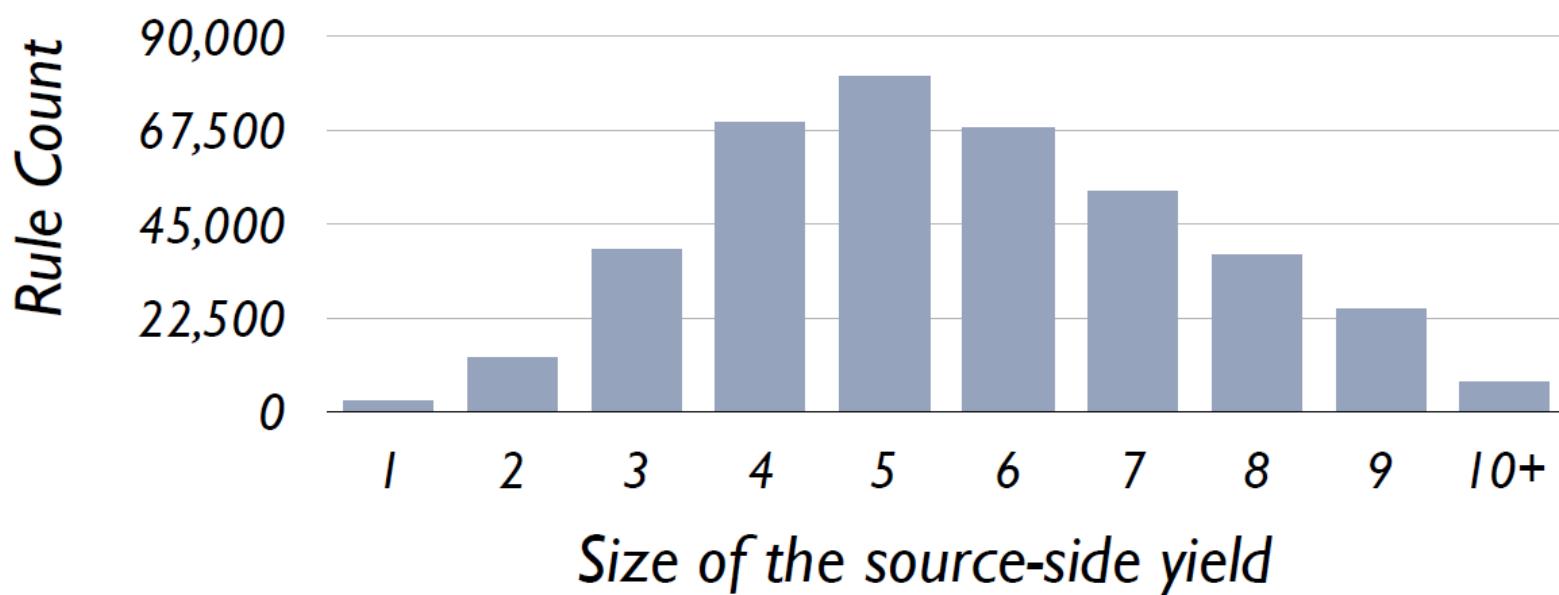
⟨lo haré ADV ;
will do it ADV⟩

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

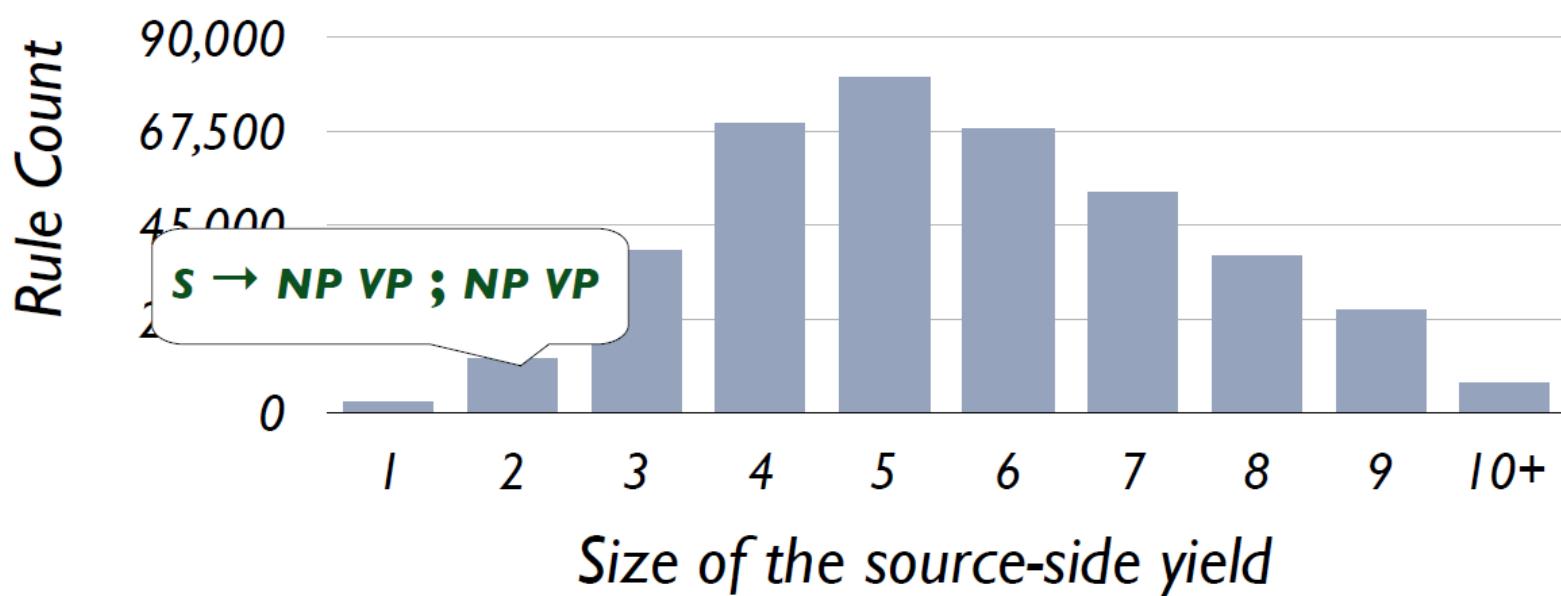


The Size of Tree Transducer Grammars

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

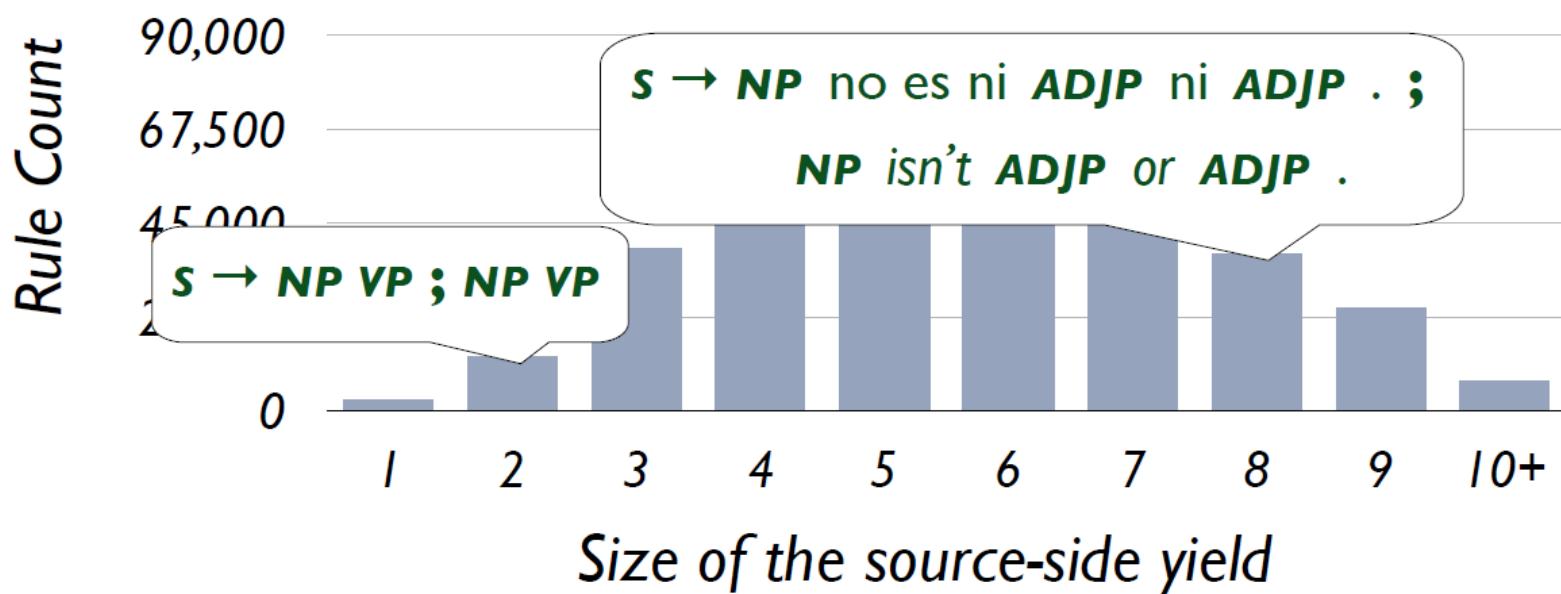


The Size of Tree Transducer Grammars

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

S		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

S	NN	NNP
Don't forget to climb a boulder in Colorado		

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

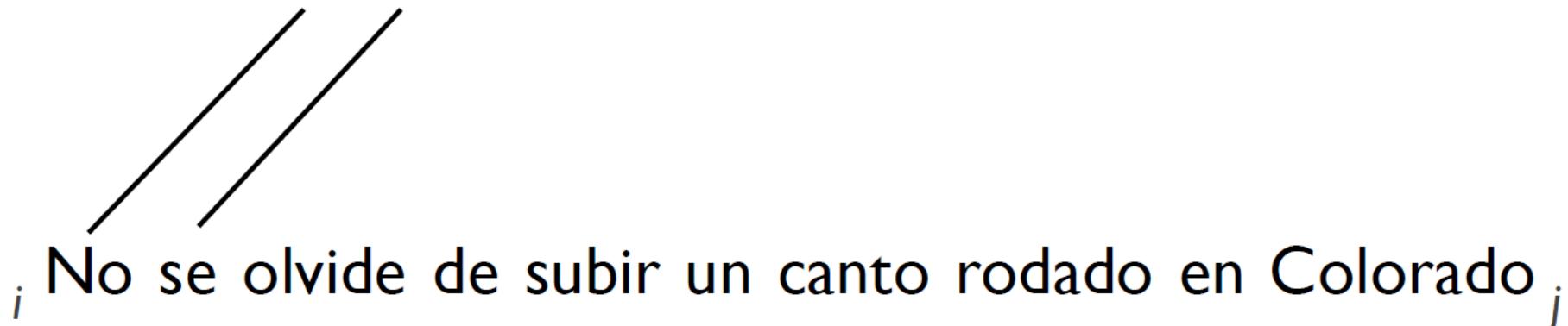
CKY-style Bottom-up Parsing

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**



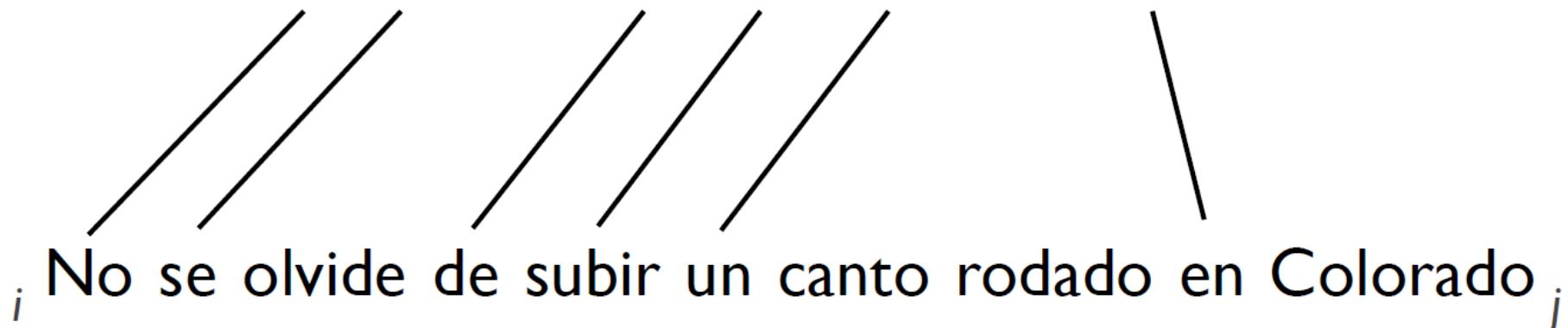
CKY-style Bottom-up Parsing

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**



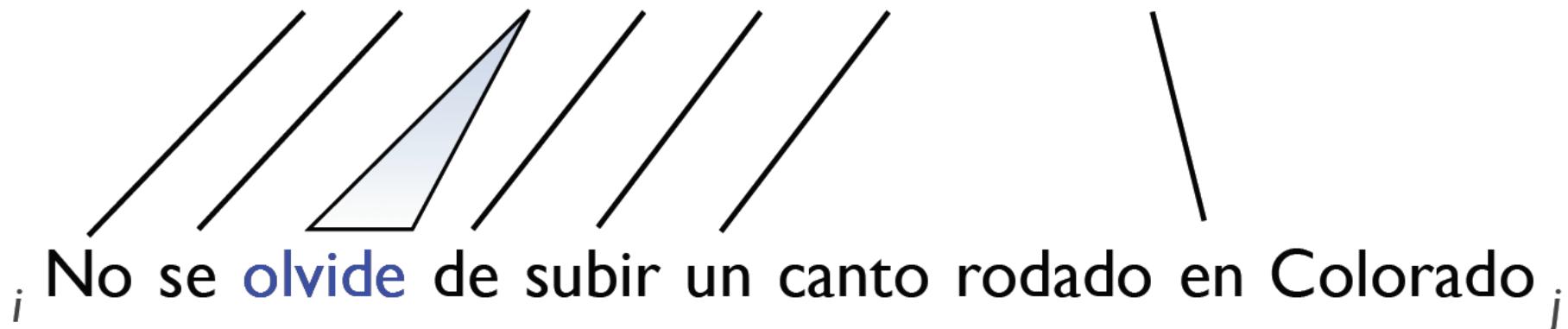
CKY-style Bottom-up Parsing

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**



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**

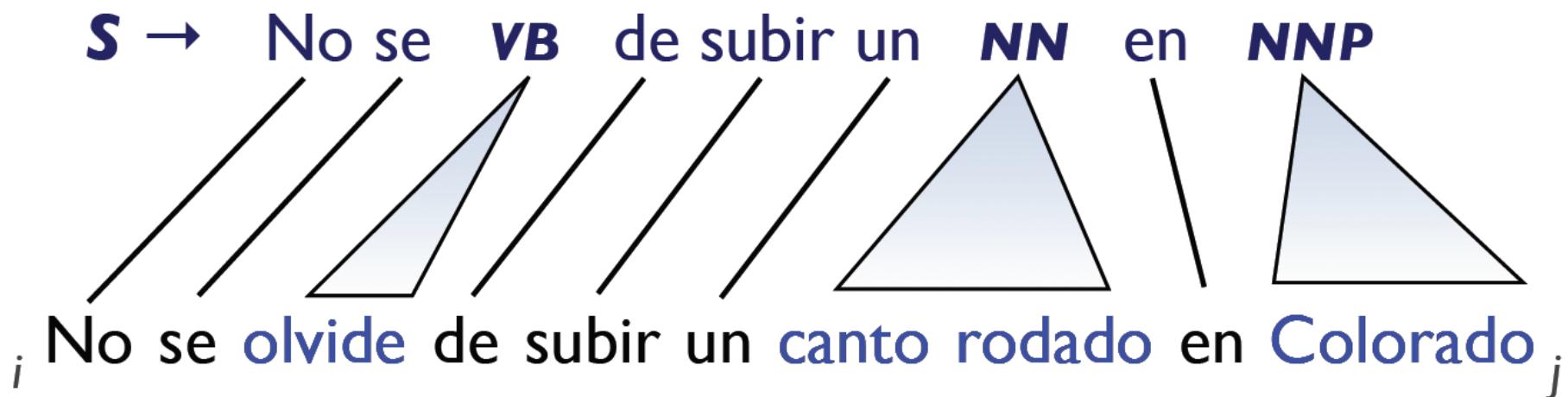


CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

Apply all grammar rules to [i,j]

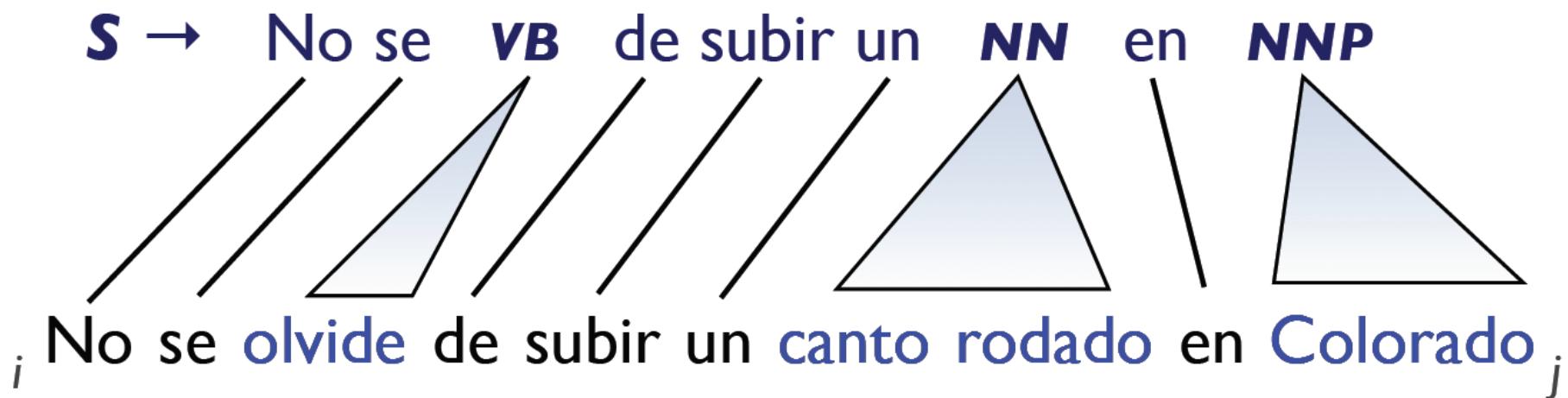


CKY-style Bottom-up Parsing

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

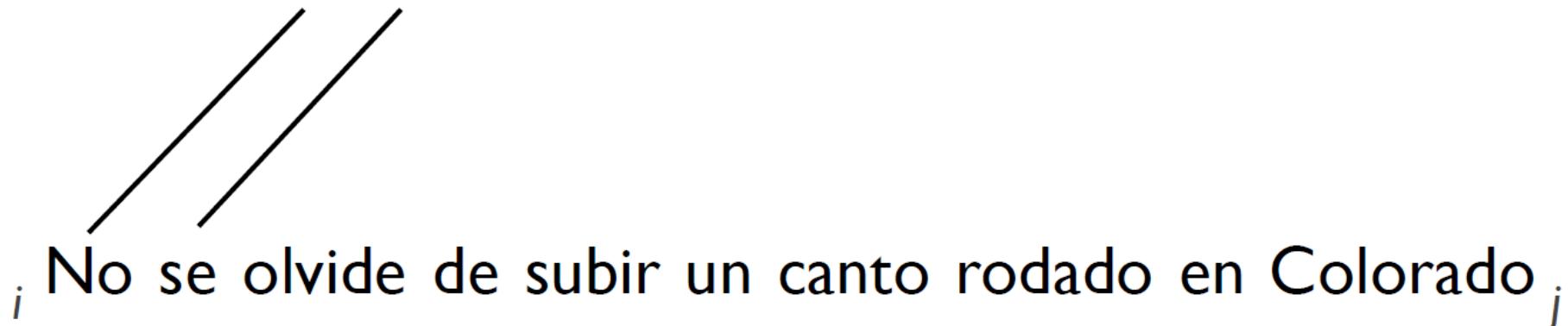
CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

Apply all grammar rules to [i,j]

S → No se VP NP PP



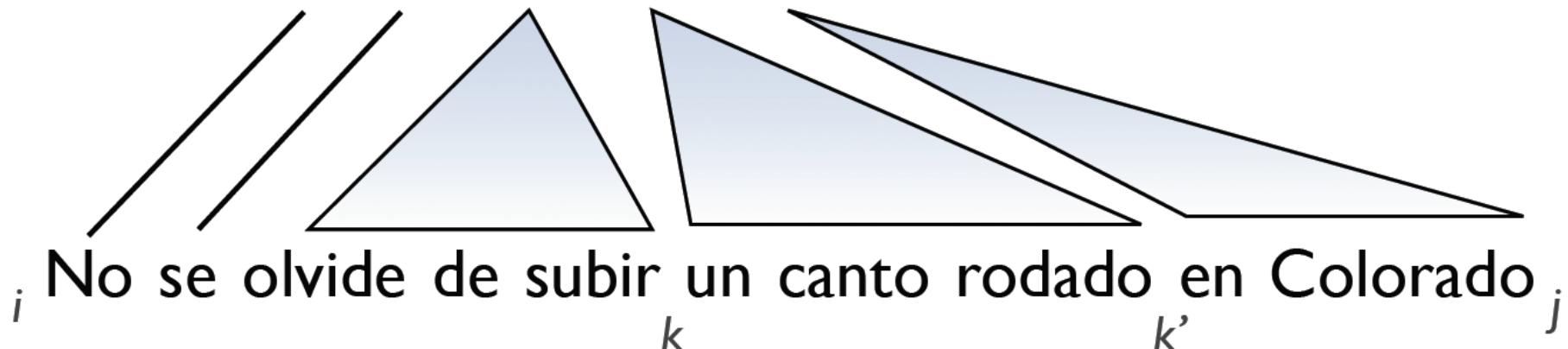
CKY-style Bottom-up Parsing

For each span length:

For each span [i,j]:

Apply all grammar rules to [i,j]

S → No se **VP** **NP** **PP**

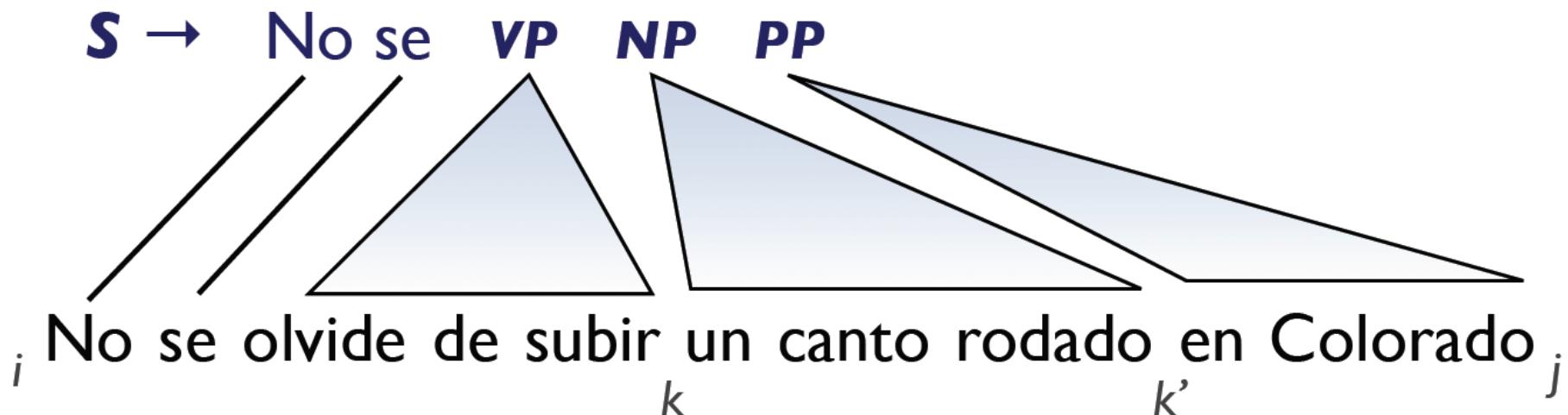


CKY-style Bottom-up Parsing

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



WIKIPEDIA
The Free Encyclopedia

≈2.6 billion words



Lots to Parse



WIKIPEDIA
The Free Encyclopedia

≈6 months (CPU)



Lots to Parse



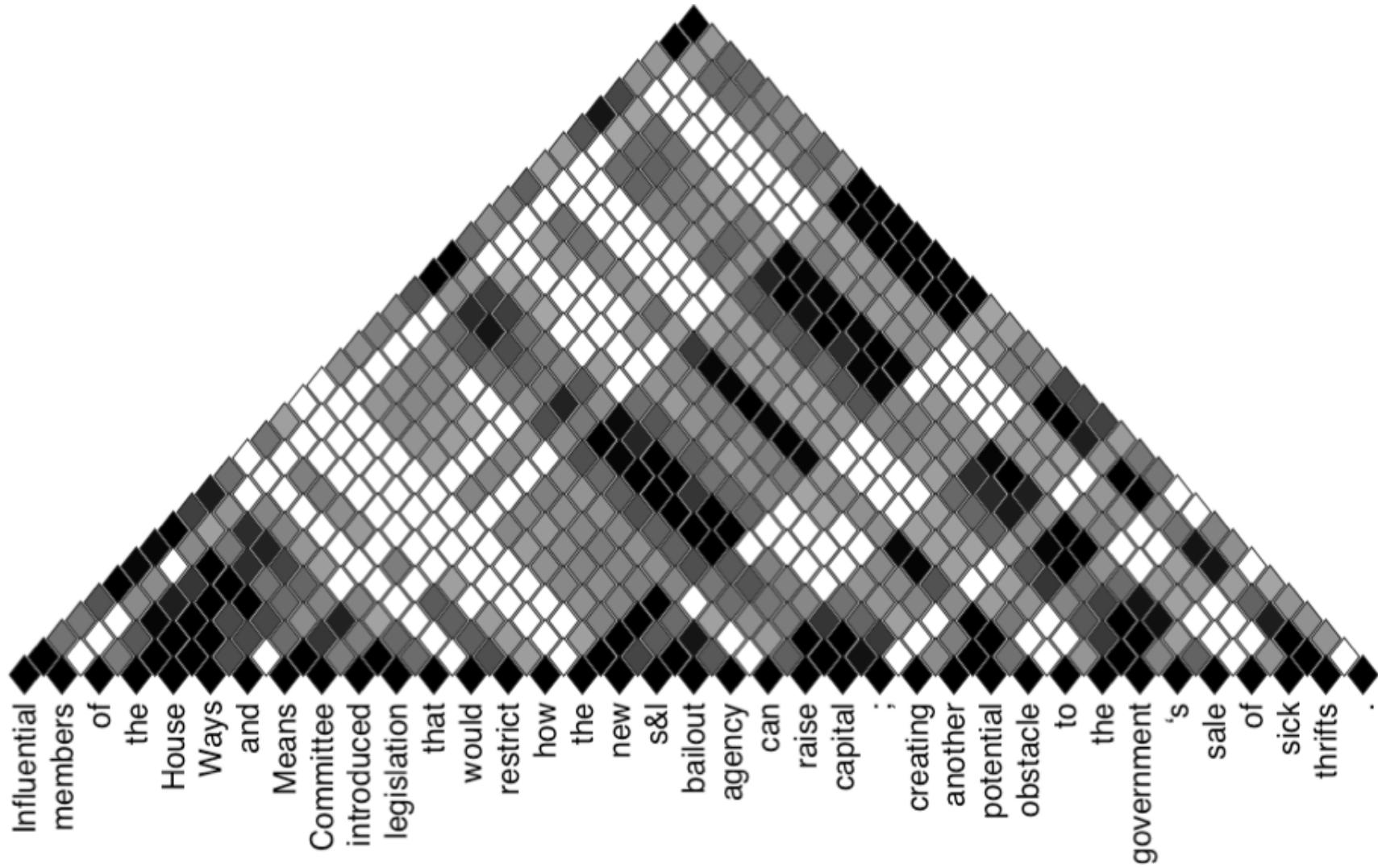
WIKIPEDIA
The Free Encyclopedia

≈3.6 days (GPU)



CPU Parsing

[Petrov & Klein, 2007]

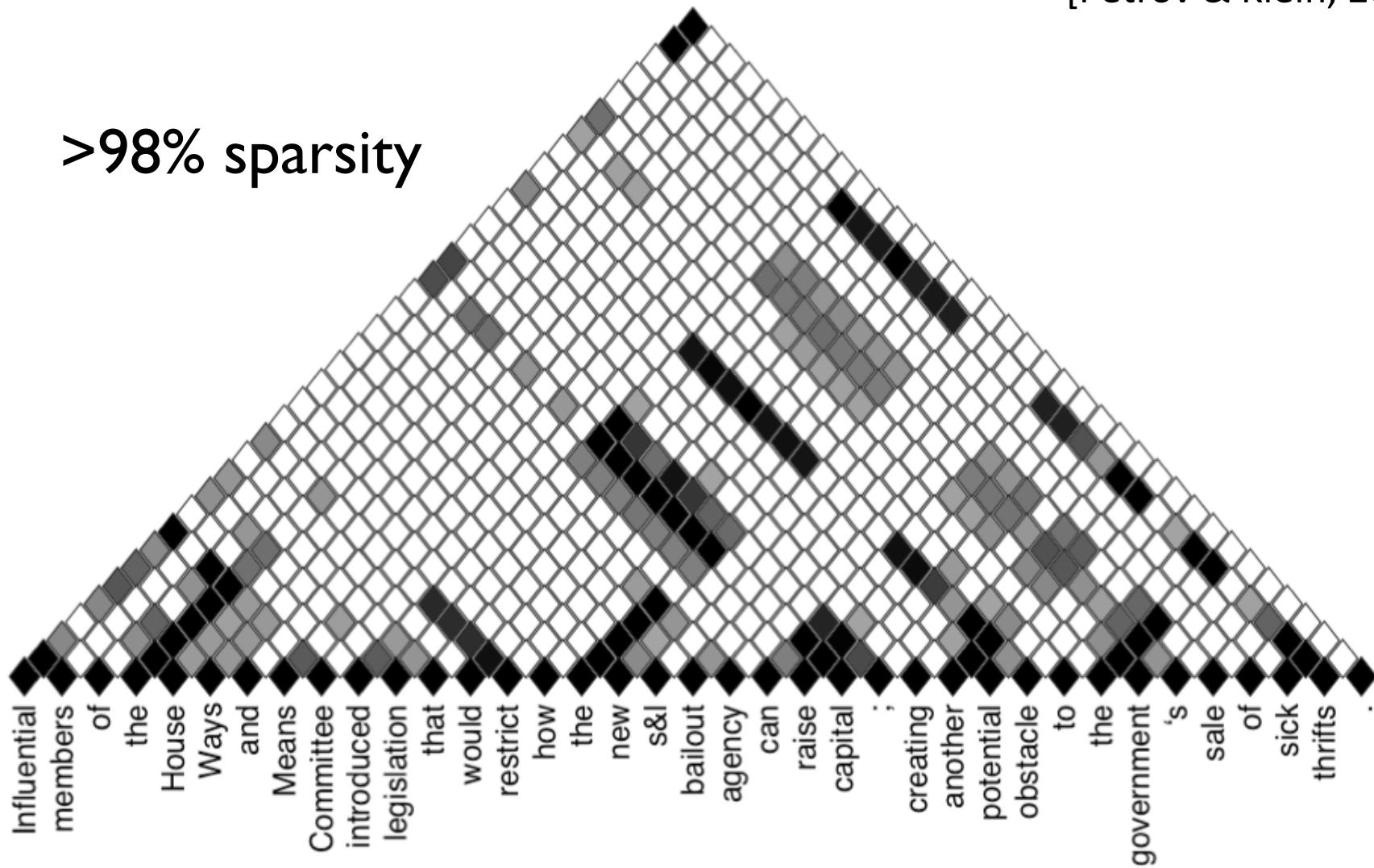




CPU Parsing

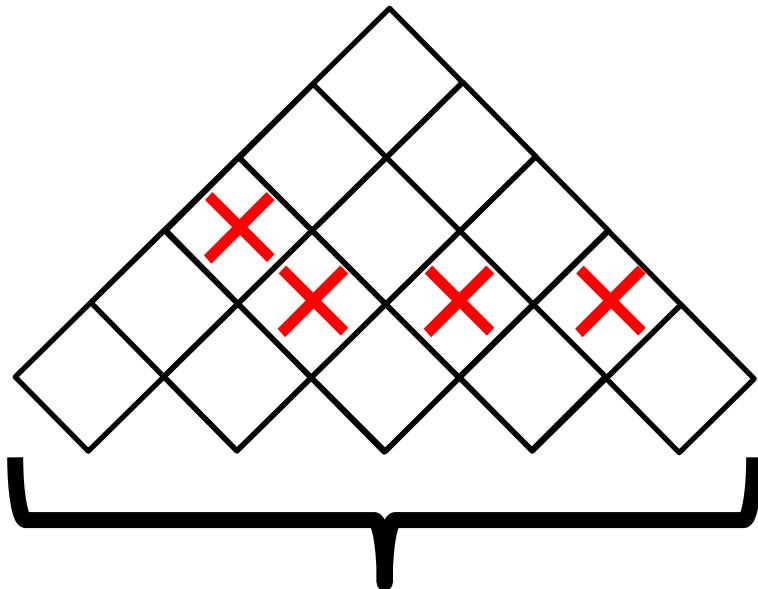
[Petrov & Klein, 2007]

>98% sparsity





CPU Parsing



Skip Spans



Skip Rules

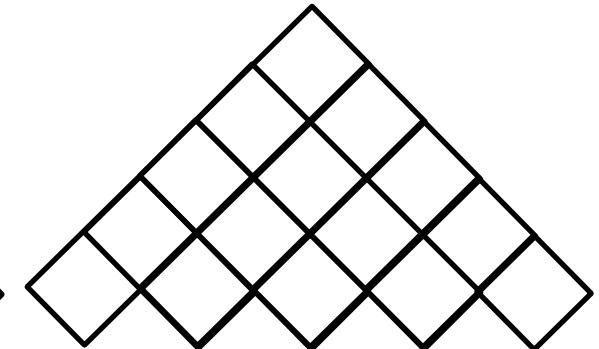
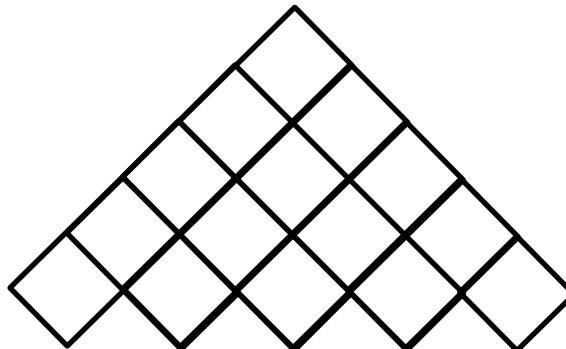
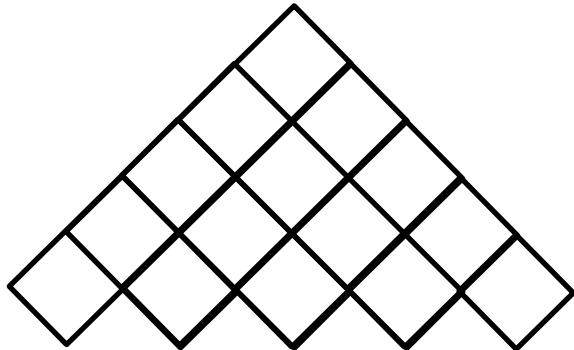


CPU Parsing

CPU



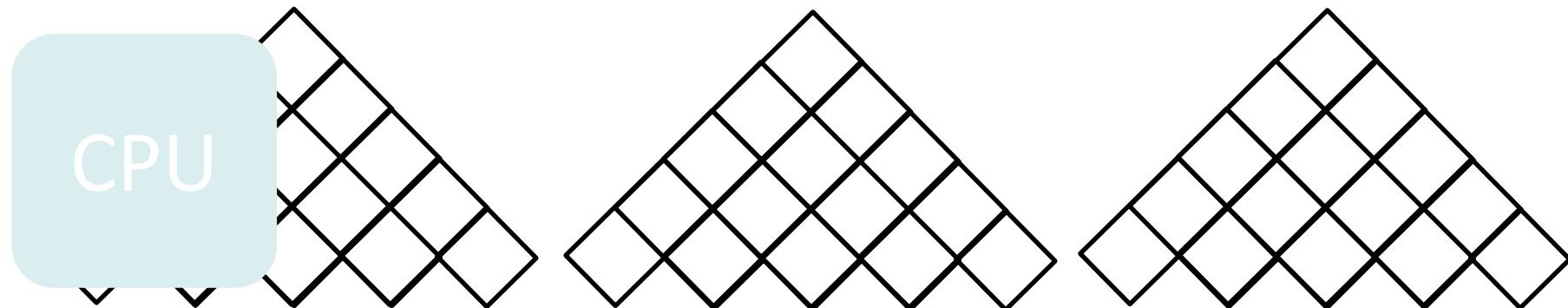
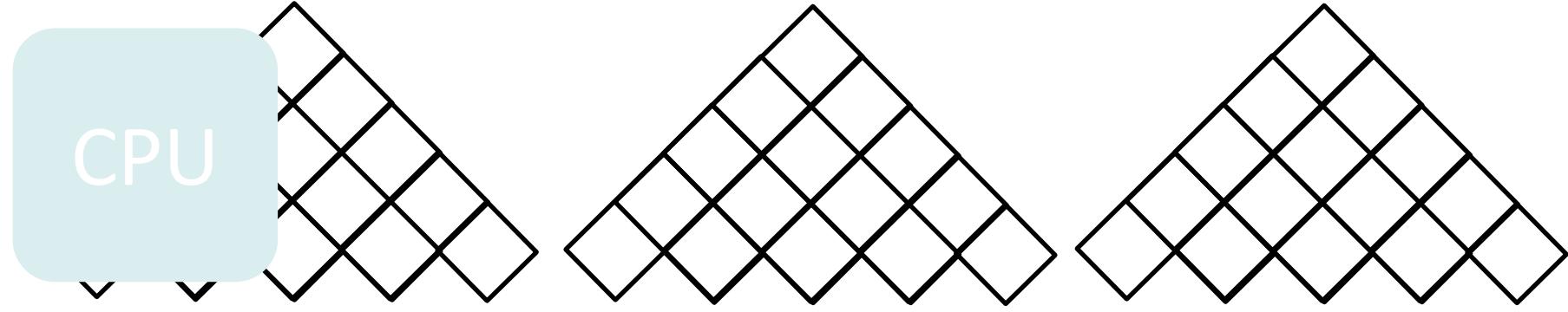
CPU Parsing



CPU



CPU Parsing



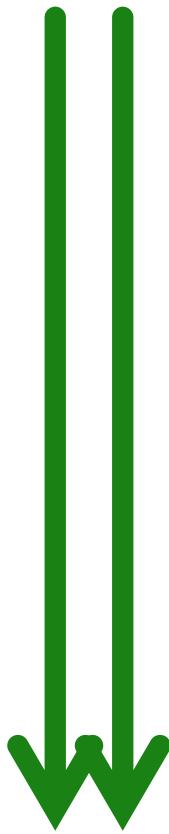


The Future of Hardware



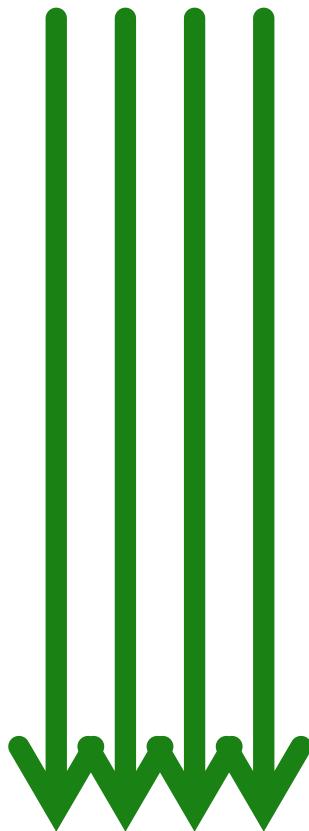


The Future of Hardware



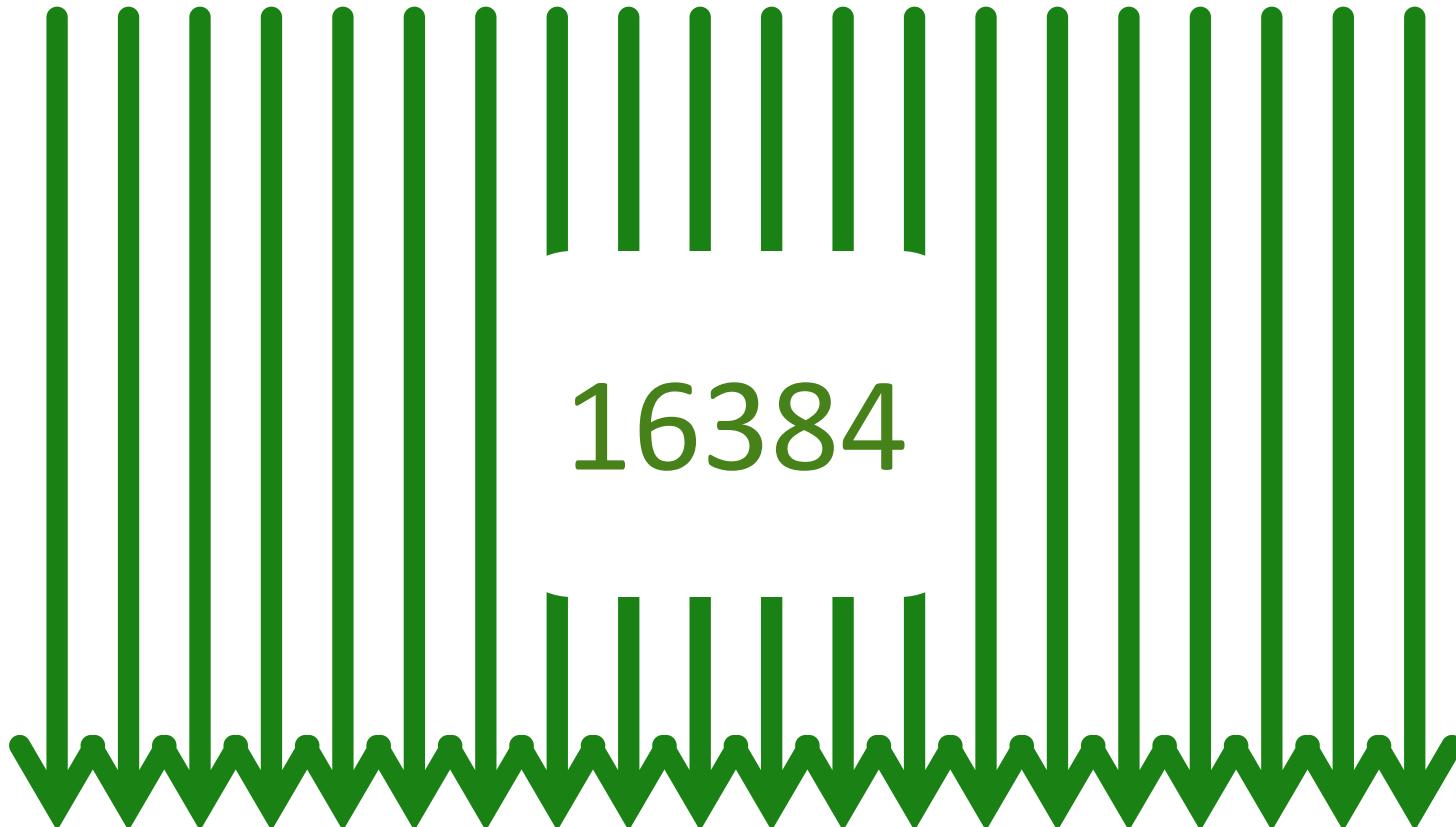


The Future of Hardware



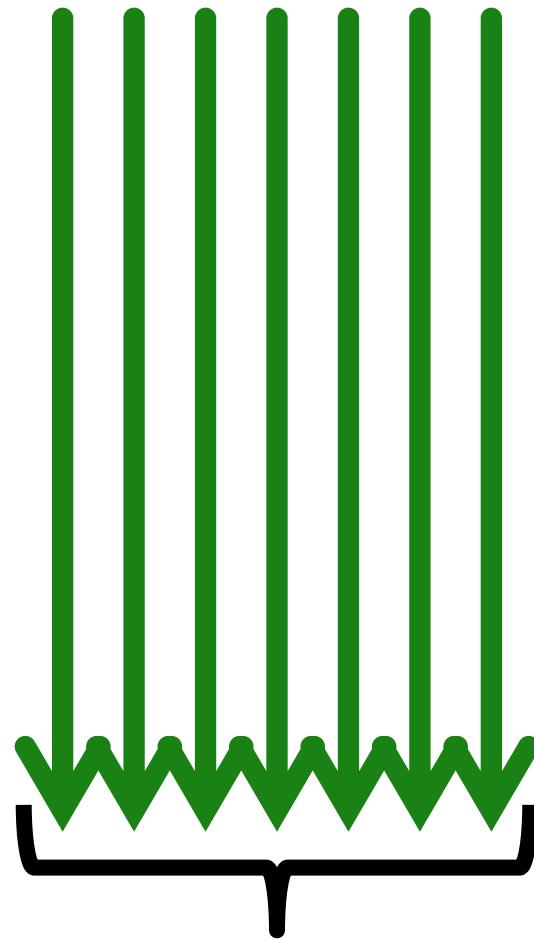


The Future of Hardware





The Future of Hardware



32 Threads



The Future of Hardware

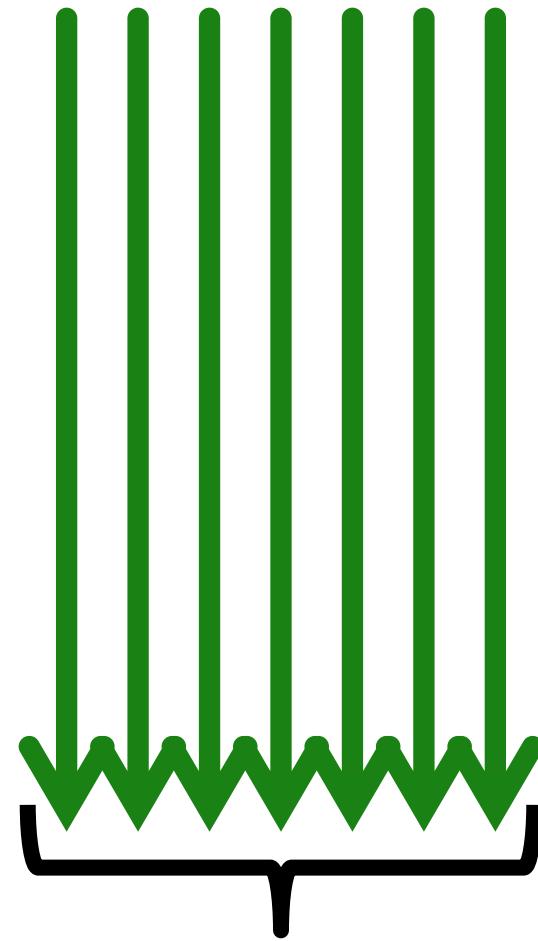
```
add.s32    %r1, %r631, %r0;  
ld.global.f32    %f81, [%r1];  
ld.global.f32    %f82, [%r34];  
mul.ftz.f32    %f94, %f82, %f81;  
mov.f32    %f95, 0F3E002E23;  
mov.f32    %f96, 0F00000000;  
mad.f32    %f93, %f94, %f95, %f96;  
shl.b32    %r2, %r646, 8;  
add.s32    %r3, %r658, %r2;  
shl.b32    %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



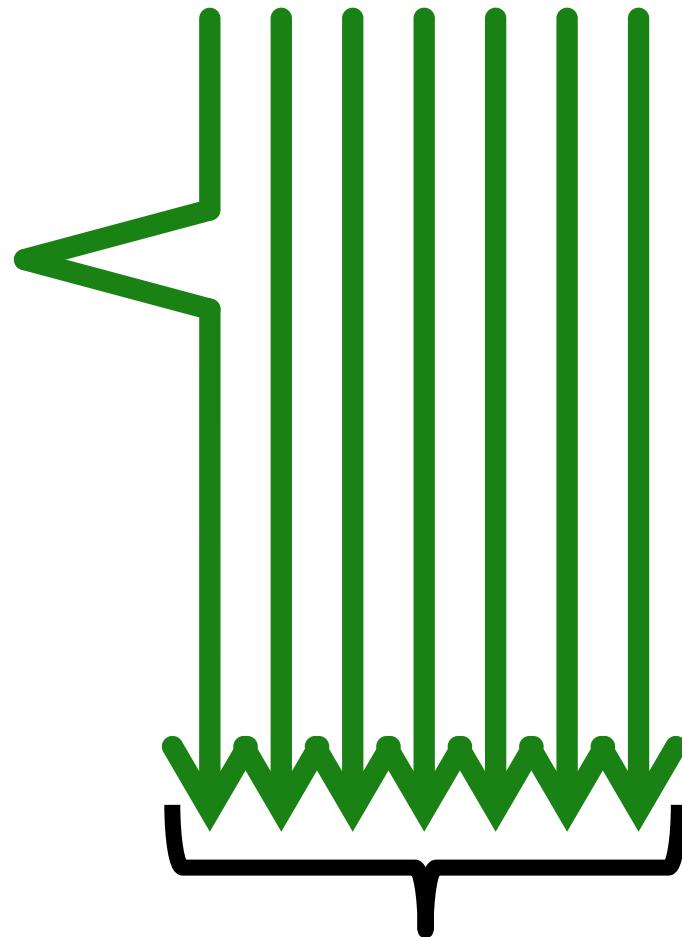
Warps



Warp



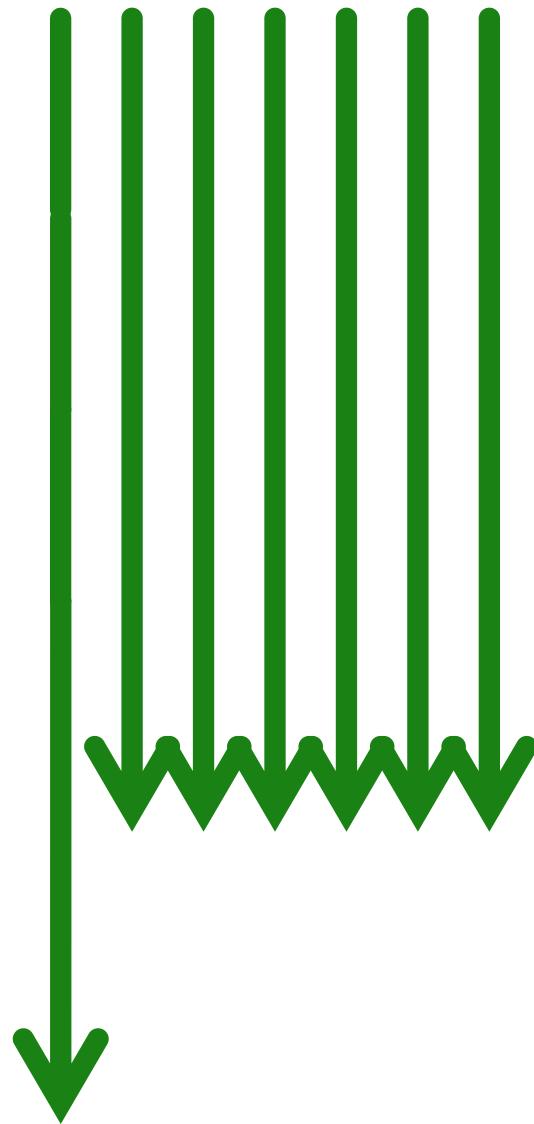
Warps



Warp Divergence

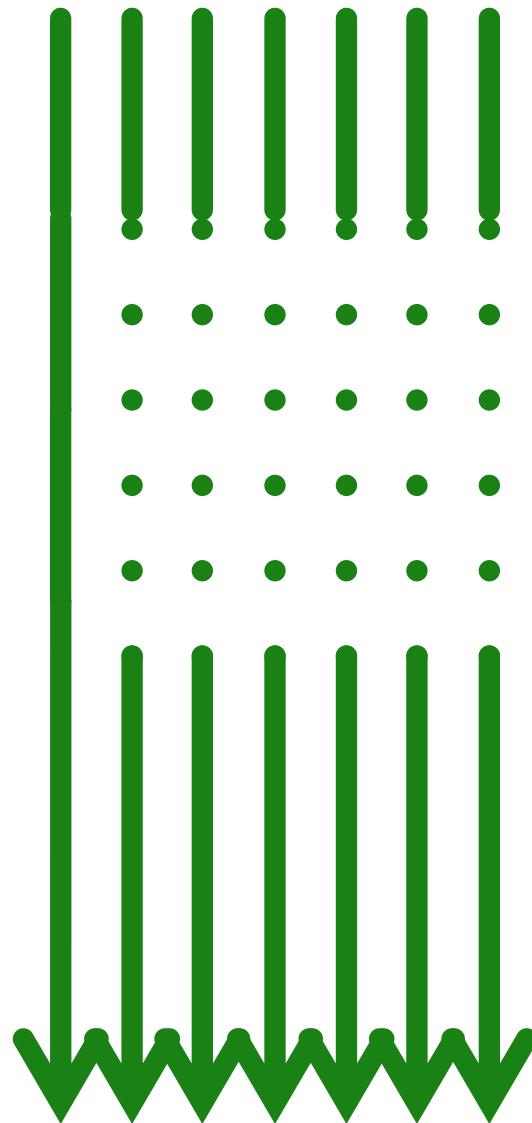


Warps



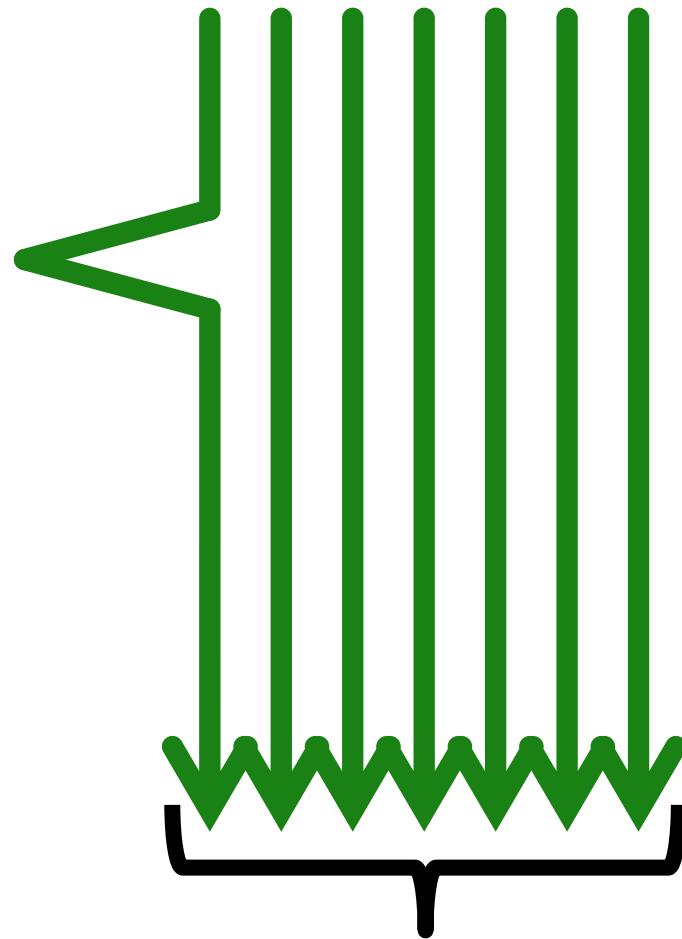


Warps





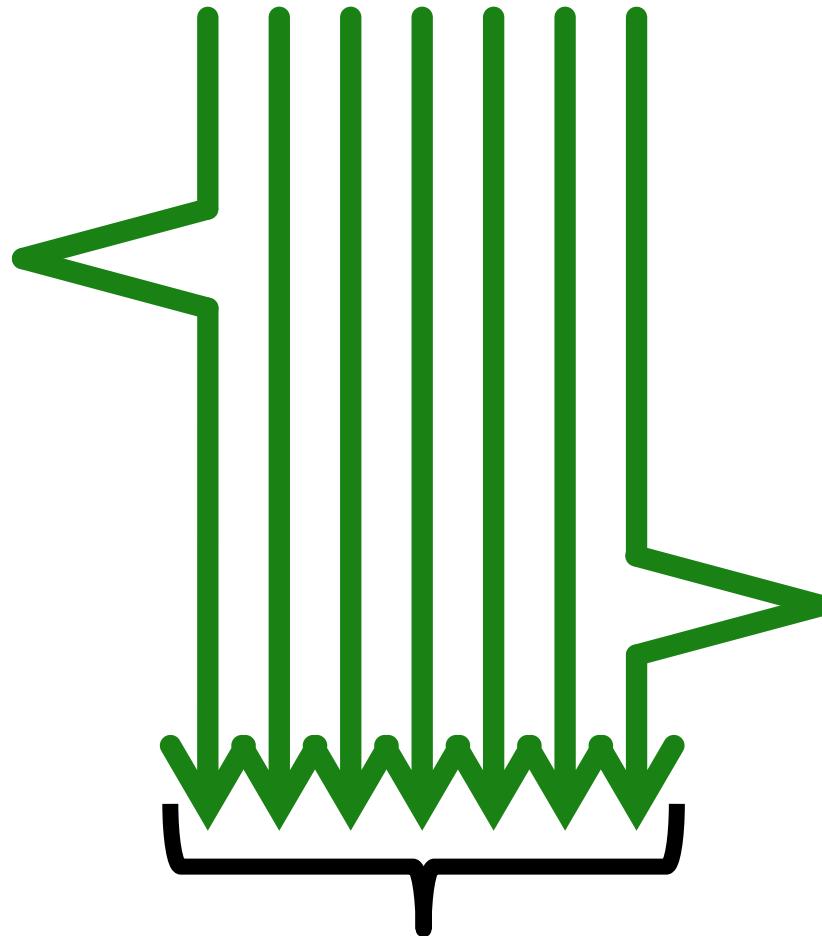
Warps



Warp Divergence



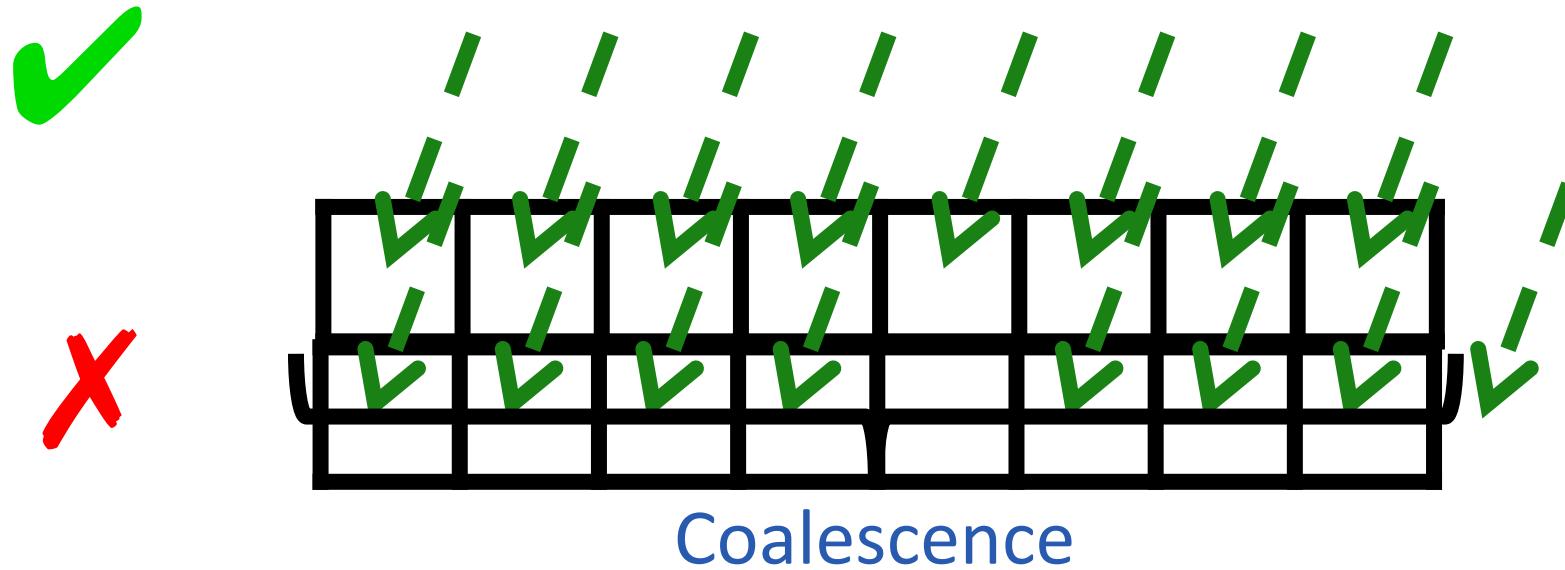
Warps



Warp Divergence

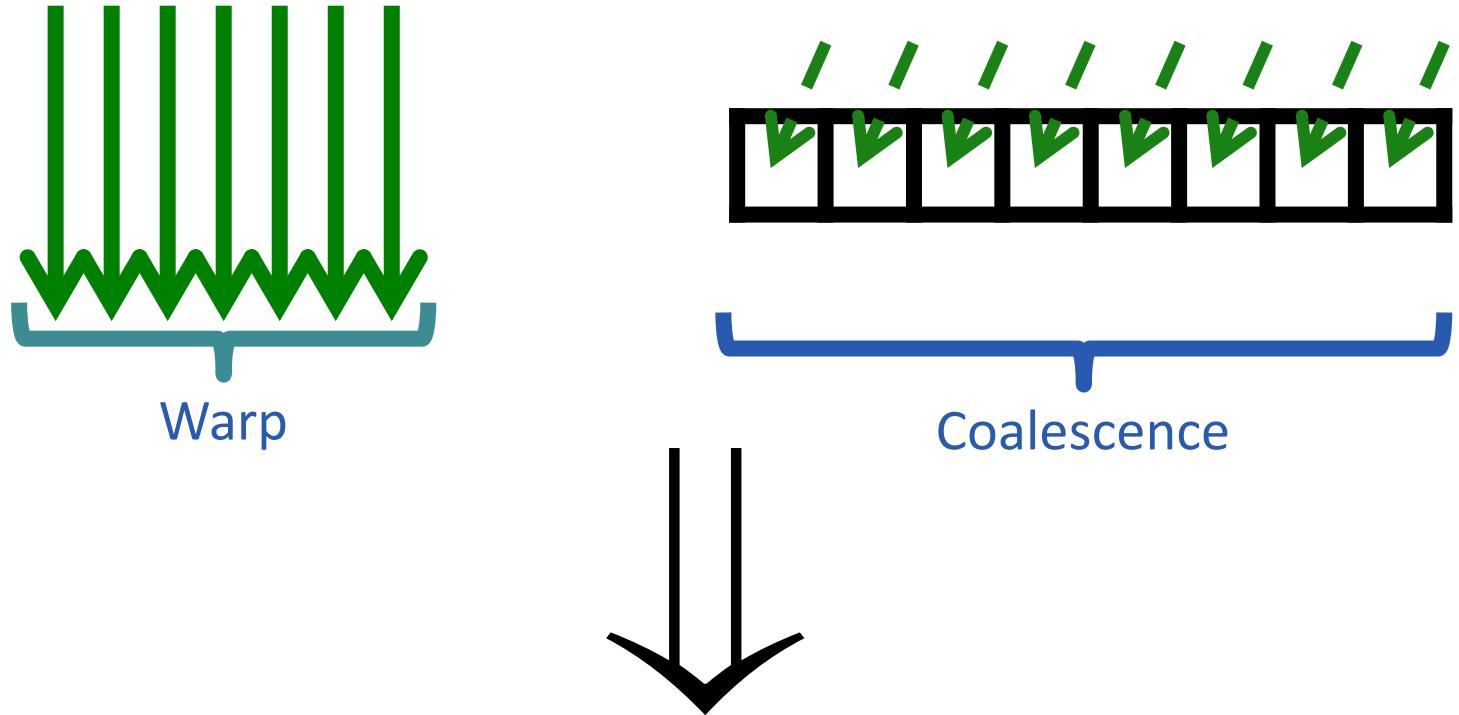


Warps





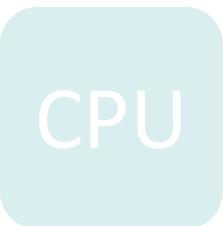
Designing GPU Algorithms



Dense, Uniform Computation

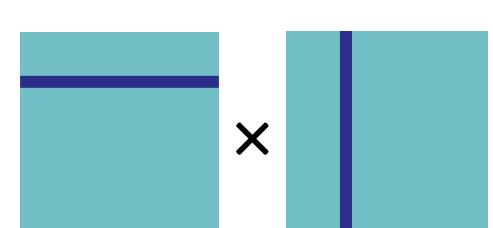
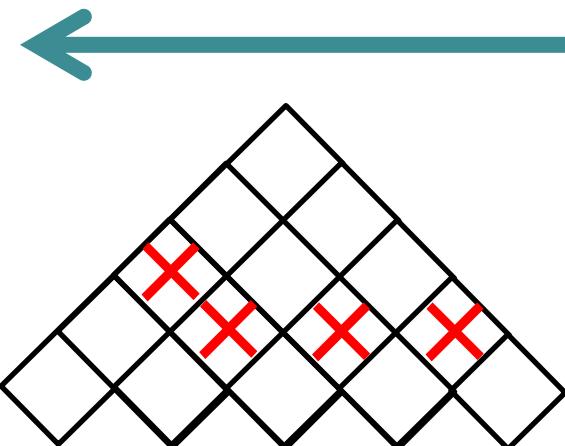


Designing GPU Algorithms



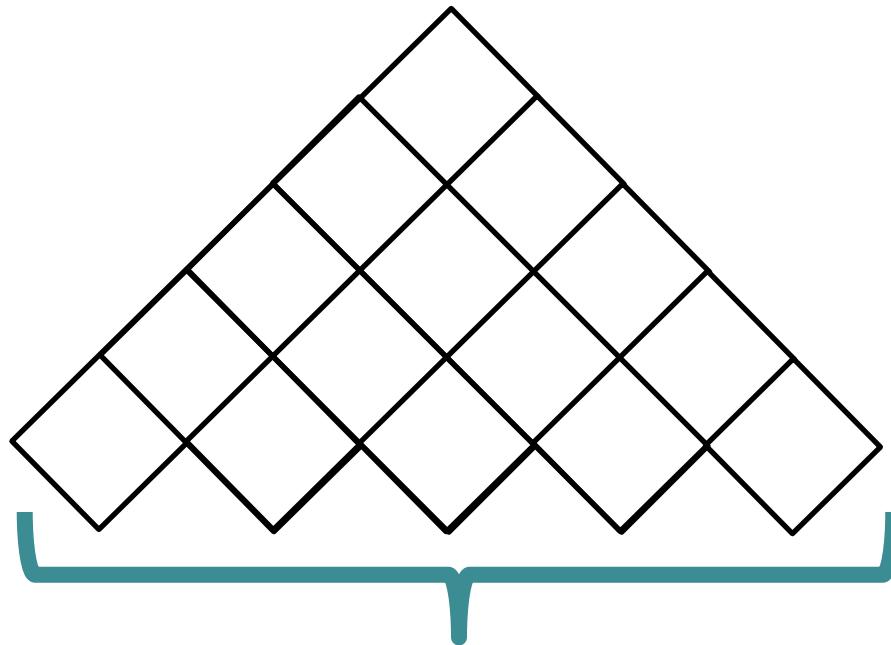
Irregular,
Sparse

Regular,
Dense





Designing GPU Algorithms



CKY Algorithm



CKY Parsing

for each sentence:

 for each span (begin, end):

 for each split:

 for each rule ($P \rightarrow L R$):

$score[begin, end, P]$

$\leftarrow ruleScore[P \rightarrow L R]$

 * $score[begin, split, L]$

 * $score[split, end, R]$

}

}

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:

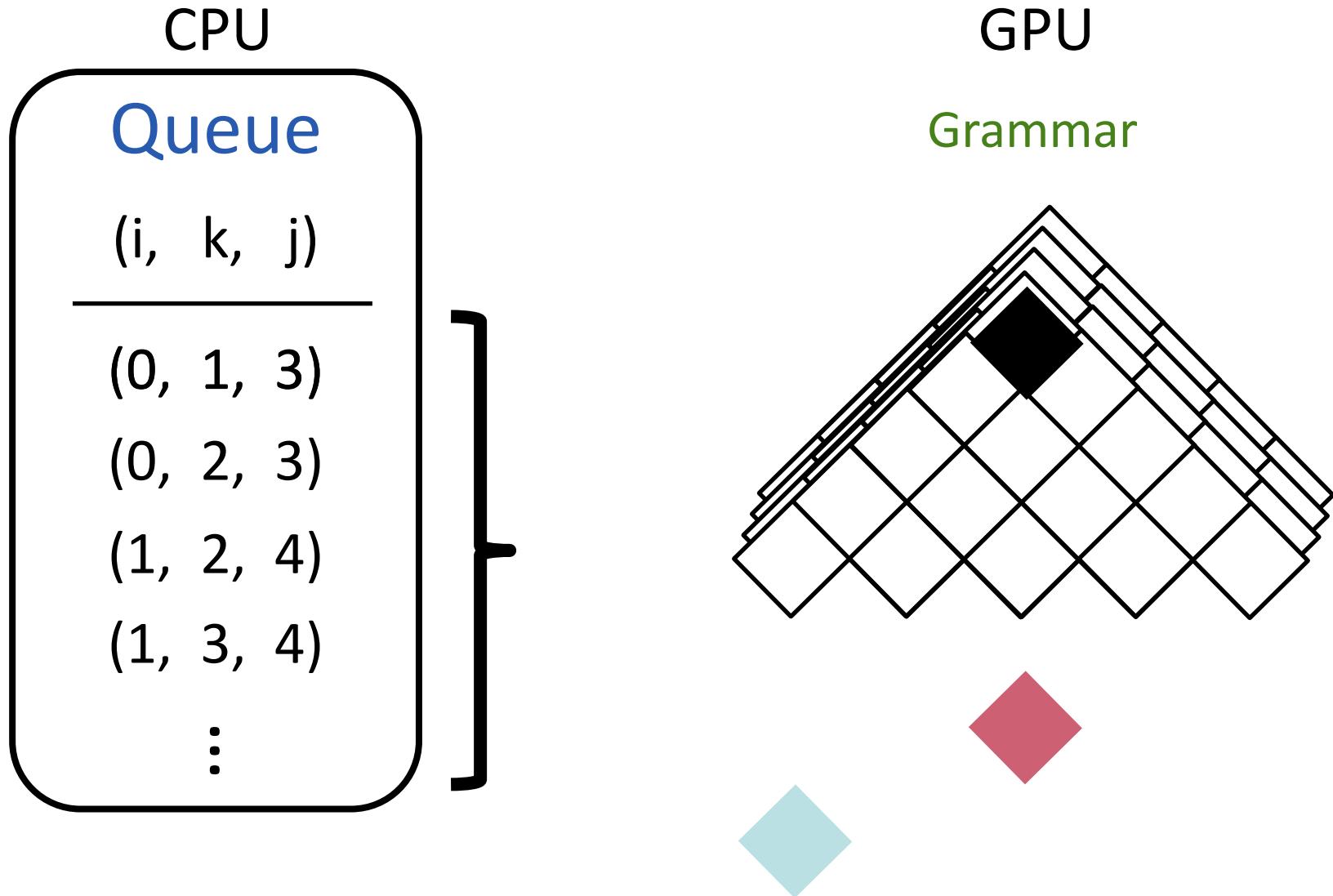
} CPU

applyGrammar(item)

GPU

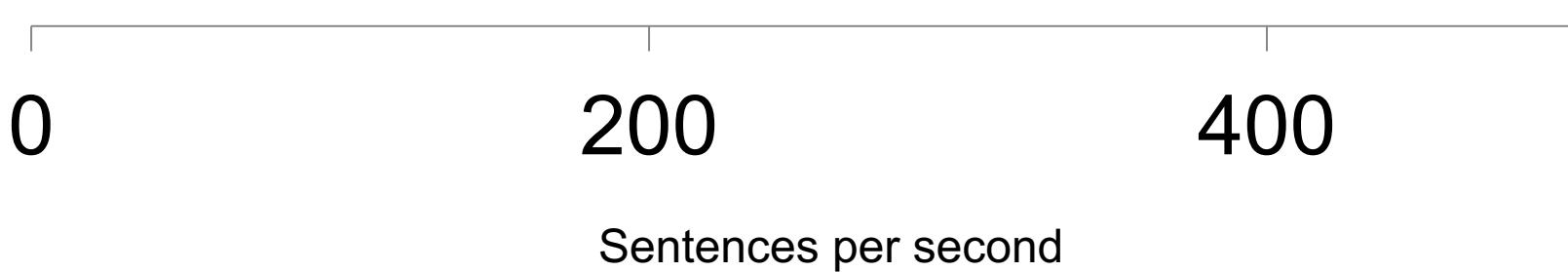


GPU Parsing Pipeline



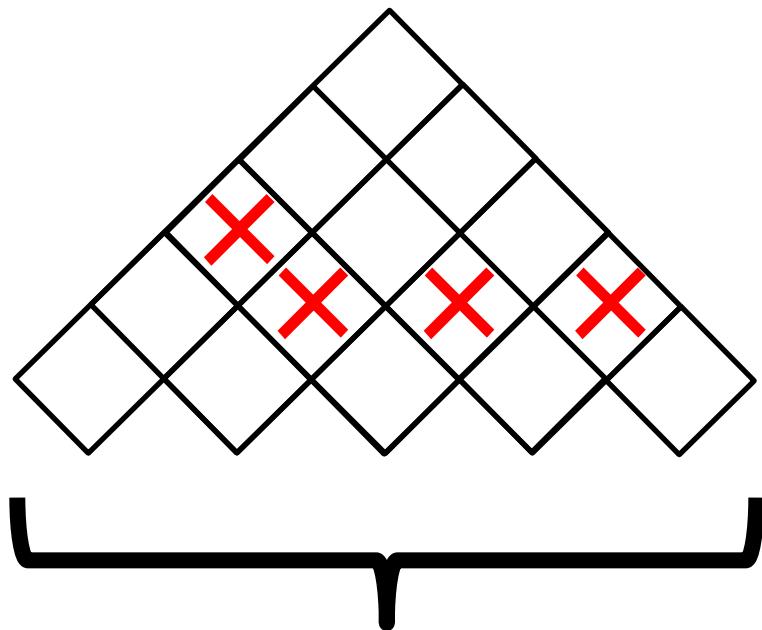


Parsing Speed

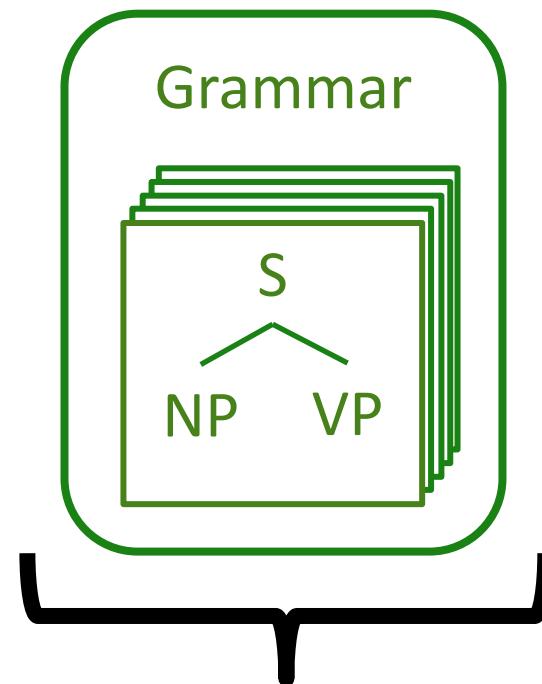




Exploiting Sparsity



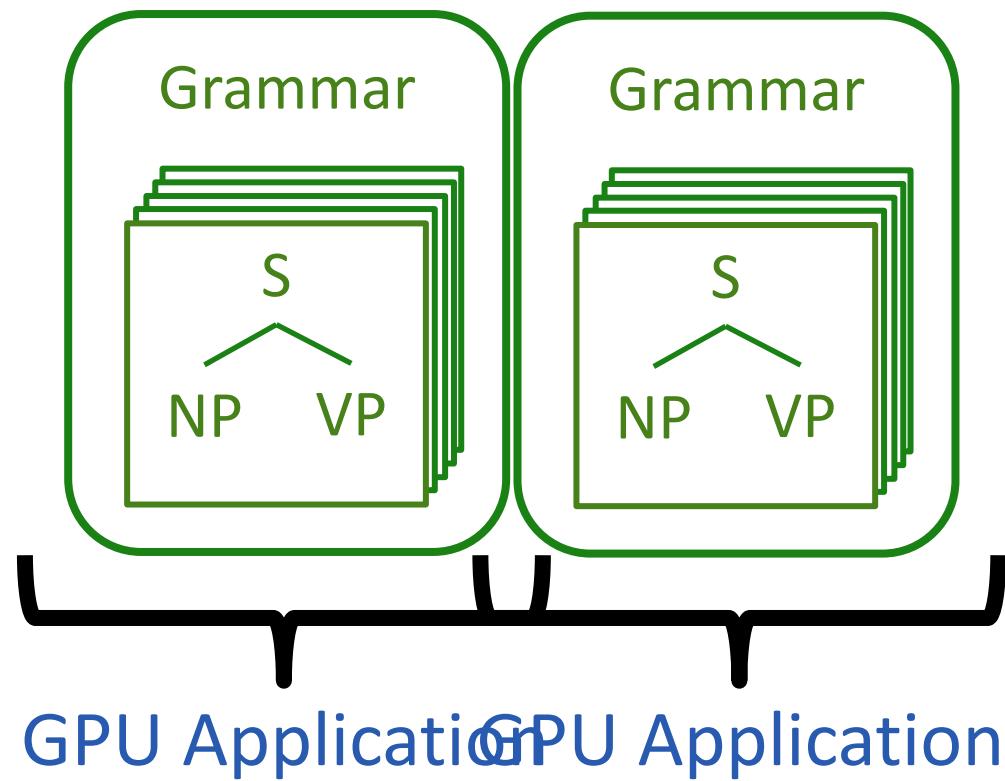
CPU Queuing



GPU Application

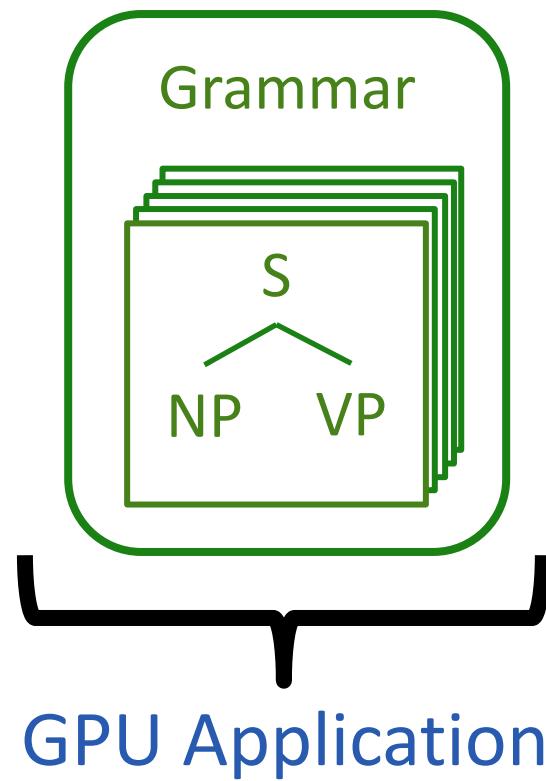


Exploiting Sparsity



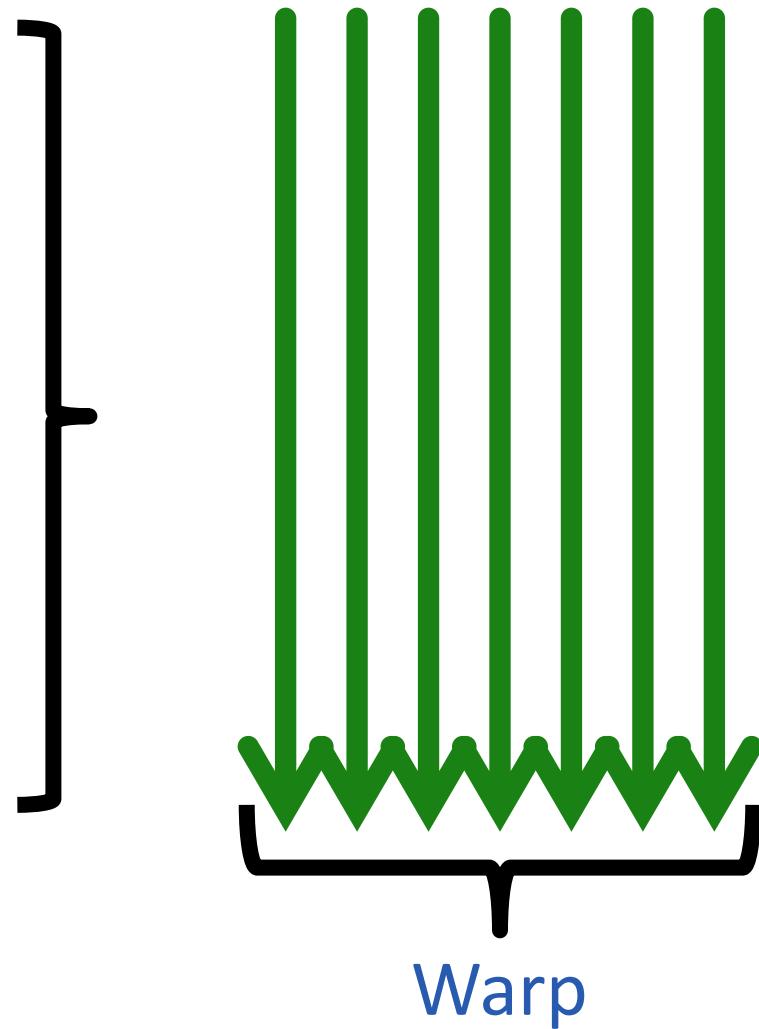


Exploiting Sparsity



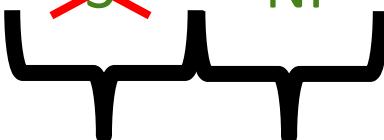


Exploiting Sparsity



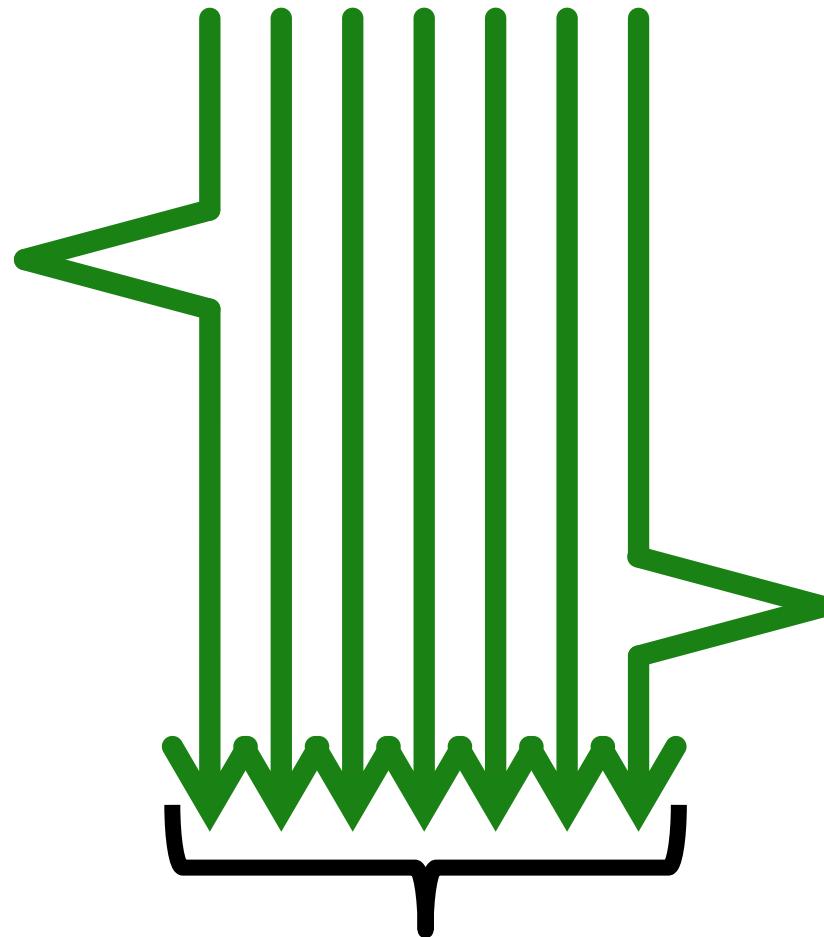


Exploiting Sparsity

(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	...
:					



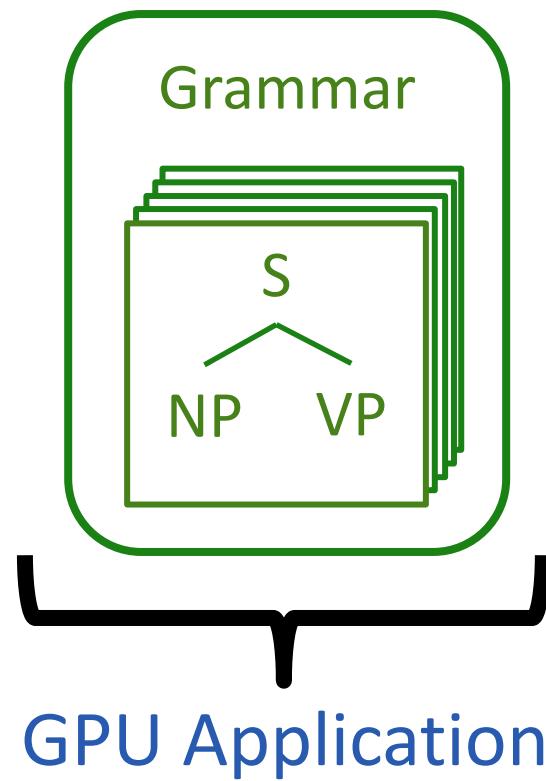
Exploiting Sparsity



Warp Divergence

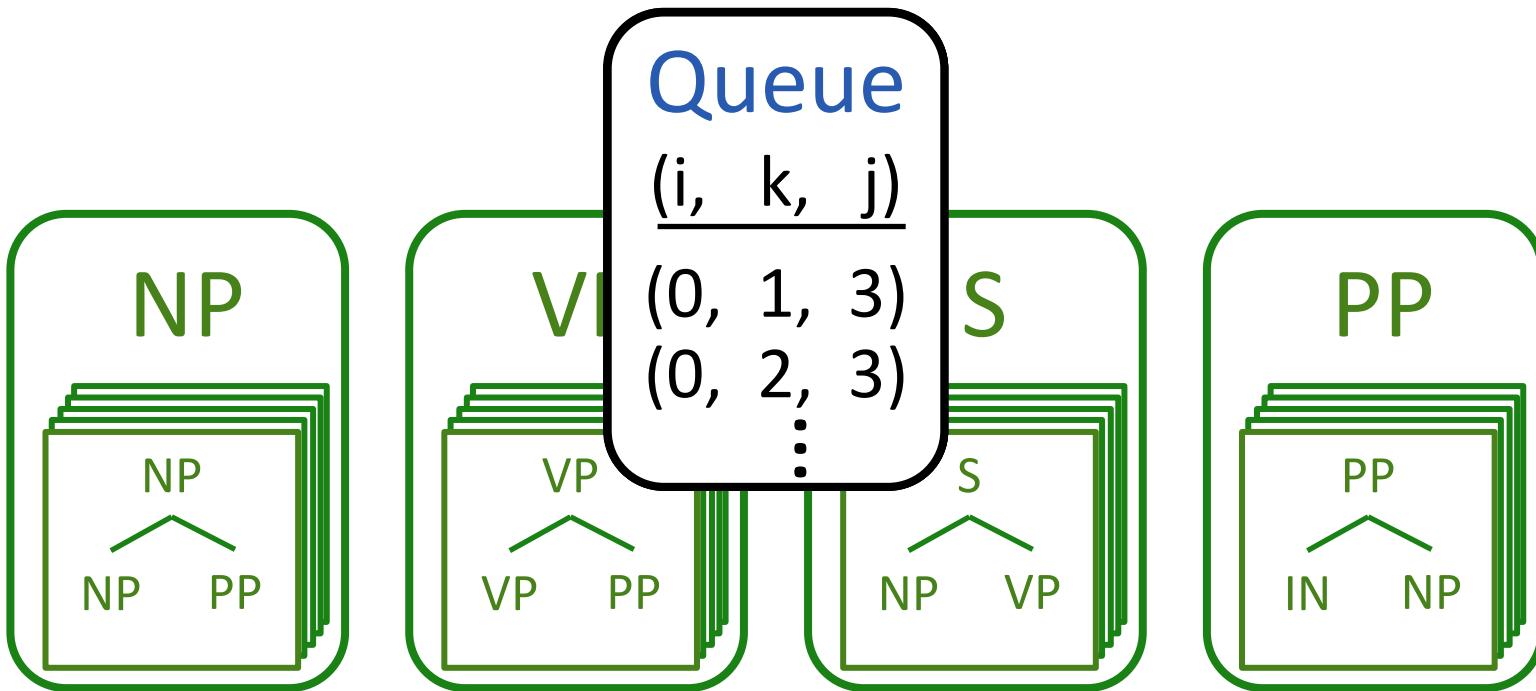


Exploiting Sparsity





Exploiting Sparsity





Exploiting Sparsity

CPU

NP Queue

(i, k, j)

(0, 1, 3)

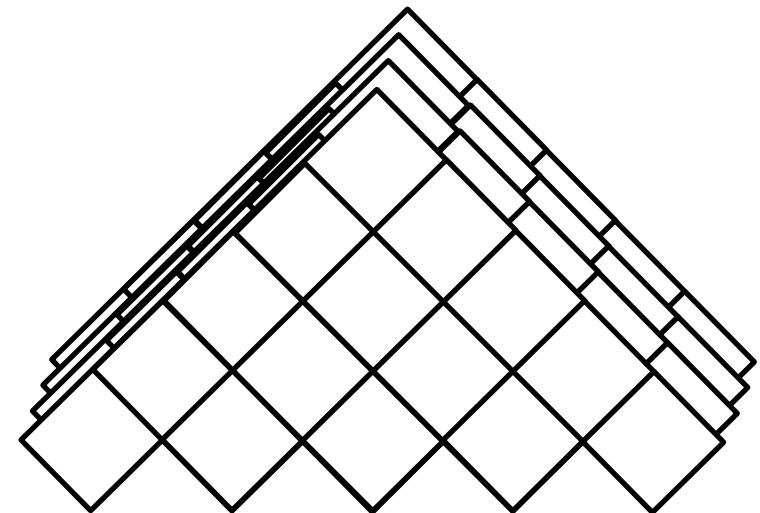
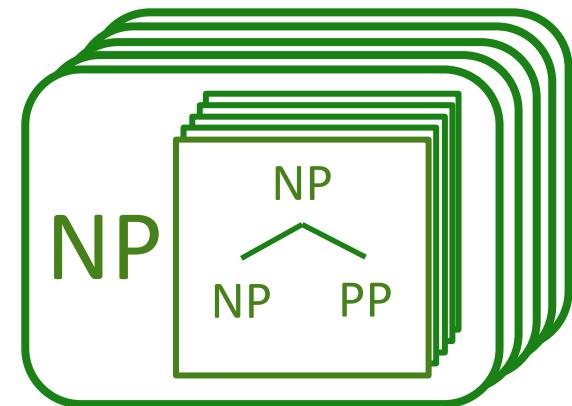
(0, 2, 3)

~~(1, 2, 4)~~

~~(1, 3, 4)~~

⋮

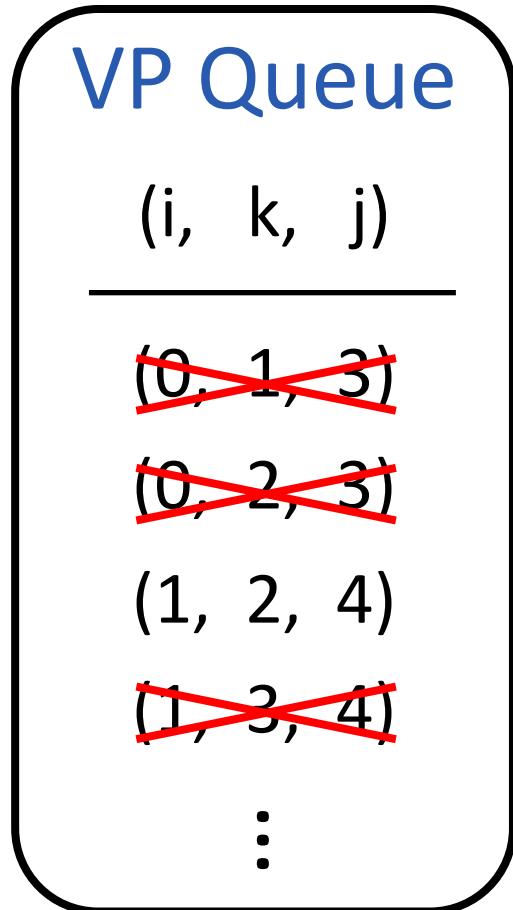
GPU



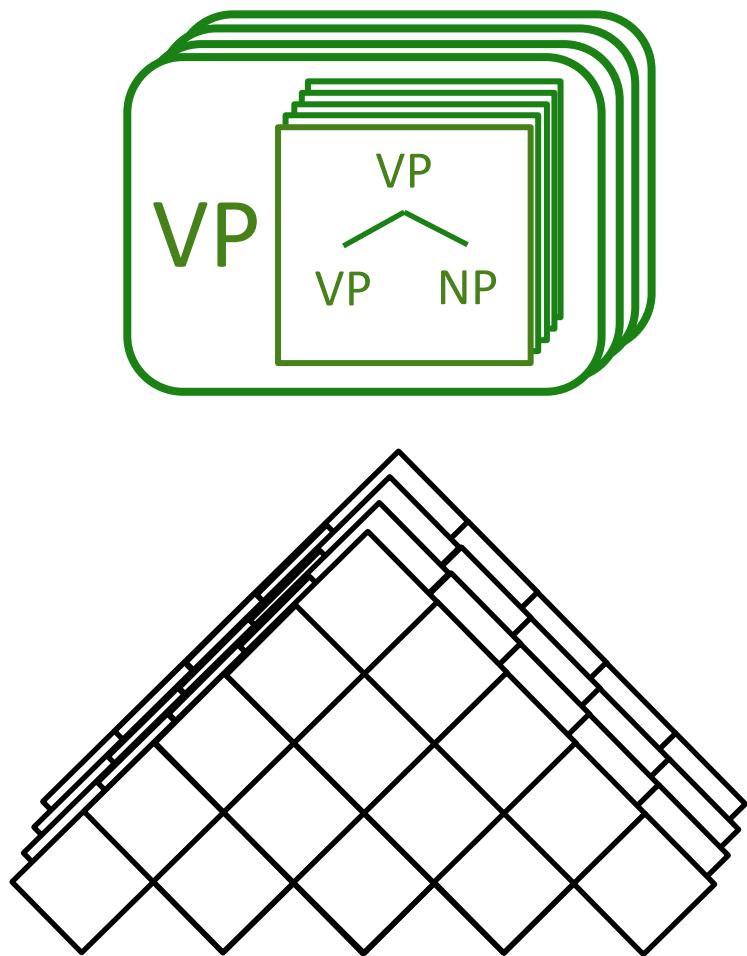


Exploiting Sparsity

CPU



GPU





Parsing Speed

