Linear Time Constituency Parsing with RNNs and Dynamic Programming

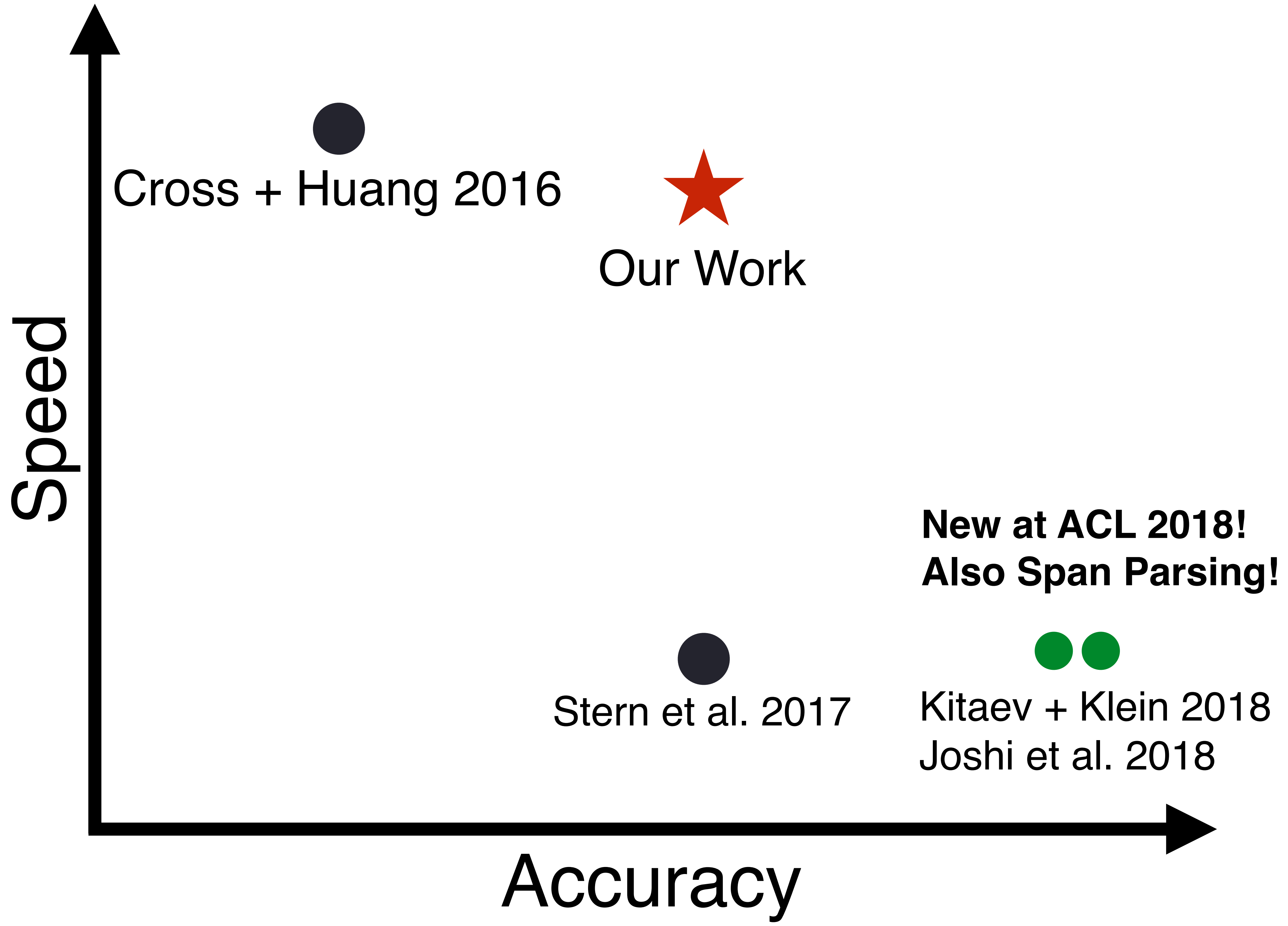
Juneki Hong [1] **Liang Huang** [1,2]

[1] Oregon State University

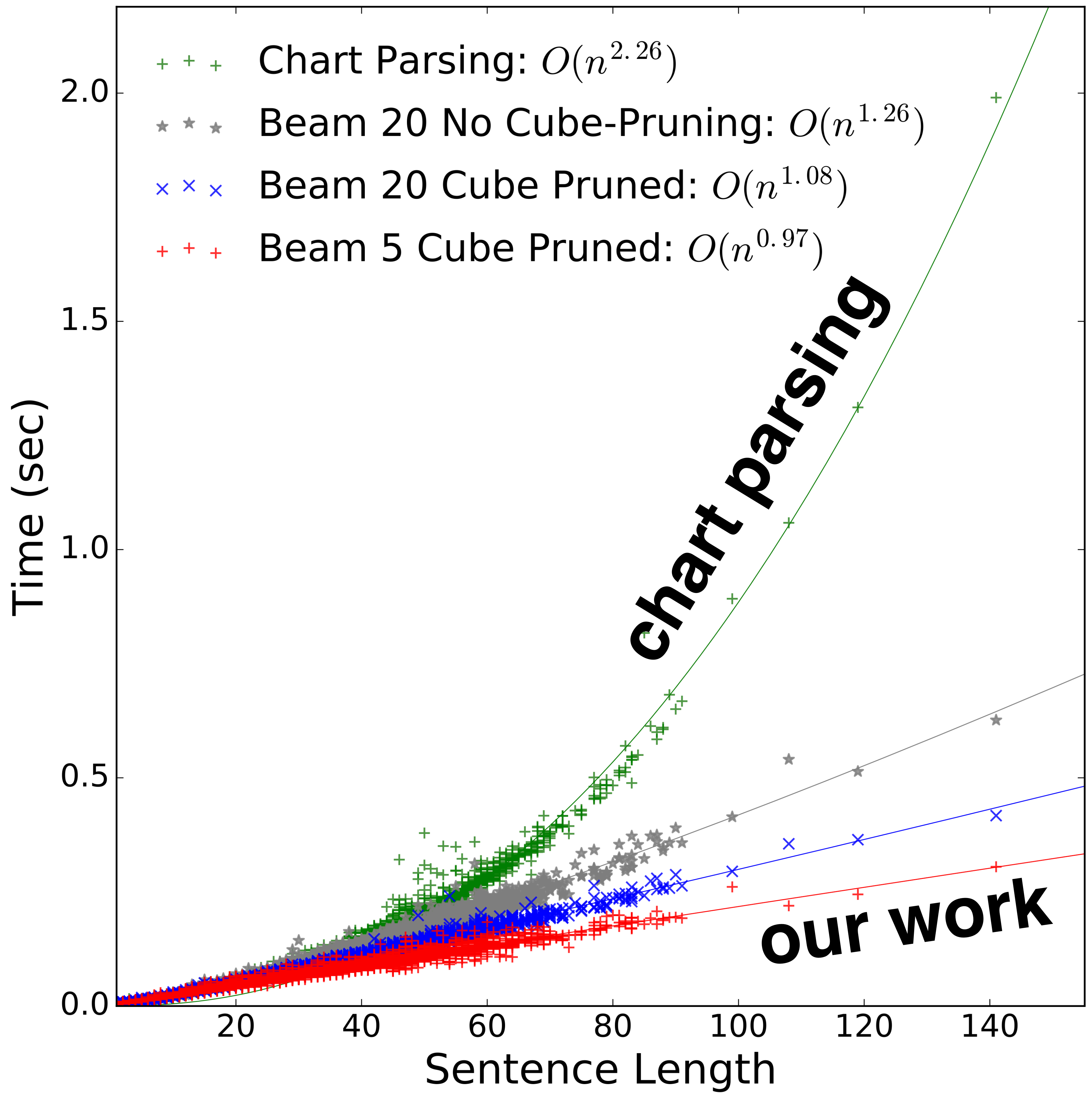
[2] Baidu Research Silicon Valley AI Lab

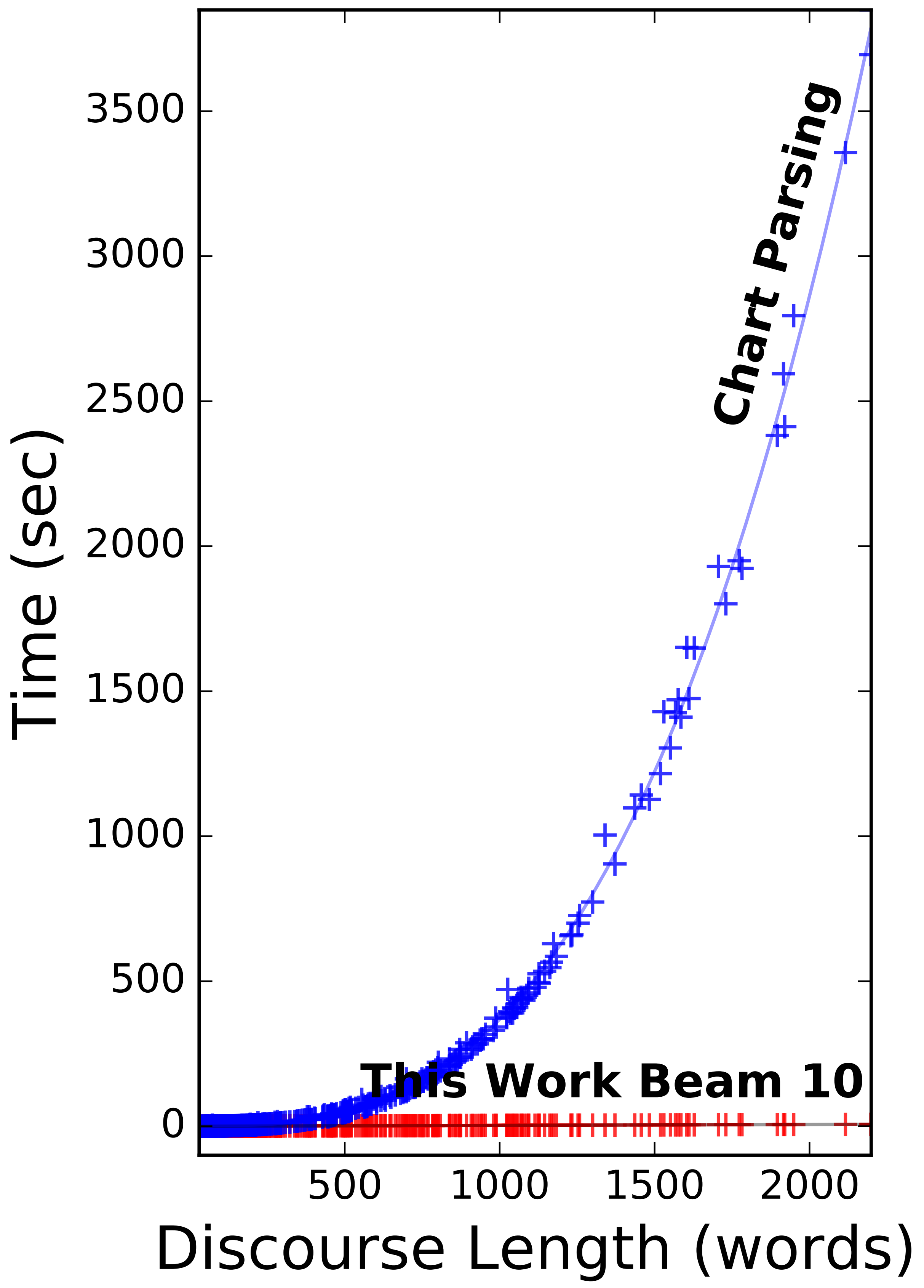
Span Parsing is SOTA in Constituency Parsing

- Cross+Huang 2016 introduced Span Parsing
  - But with greedy decoding.
- Stern et al. 2017 had Span Parsing with Exact Search and Global Training
  - But was too slow: $O(n^3)$
- Can we get the best of both worlds?
  - Something that is both fast and accurate?



Both Fast and Accurate!





Baseline Chart Parser (Stern et al. 2017a)	91.79
Our Linear Time Parser	**91.97**

In this talk, we will discuss:

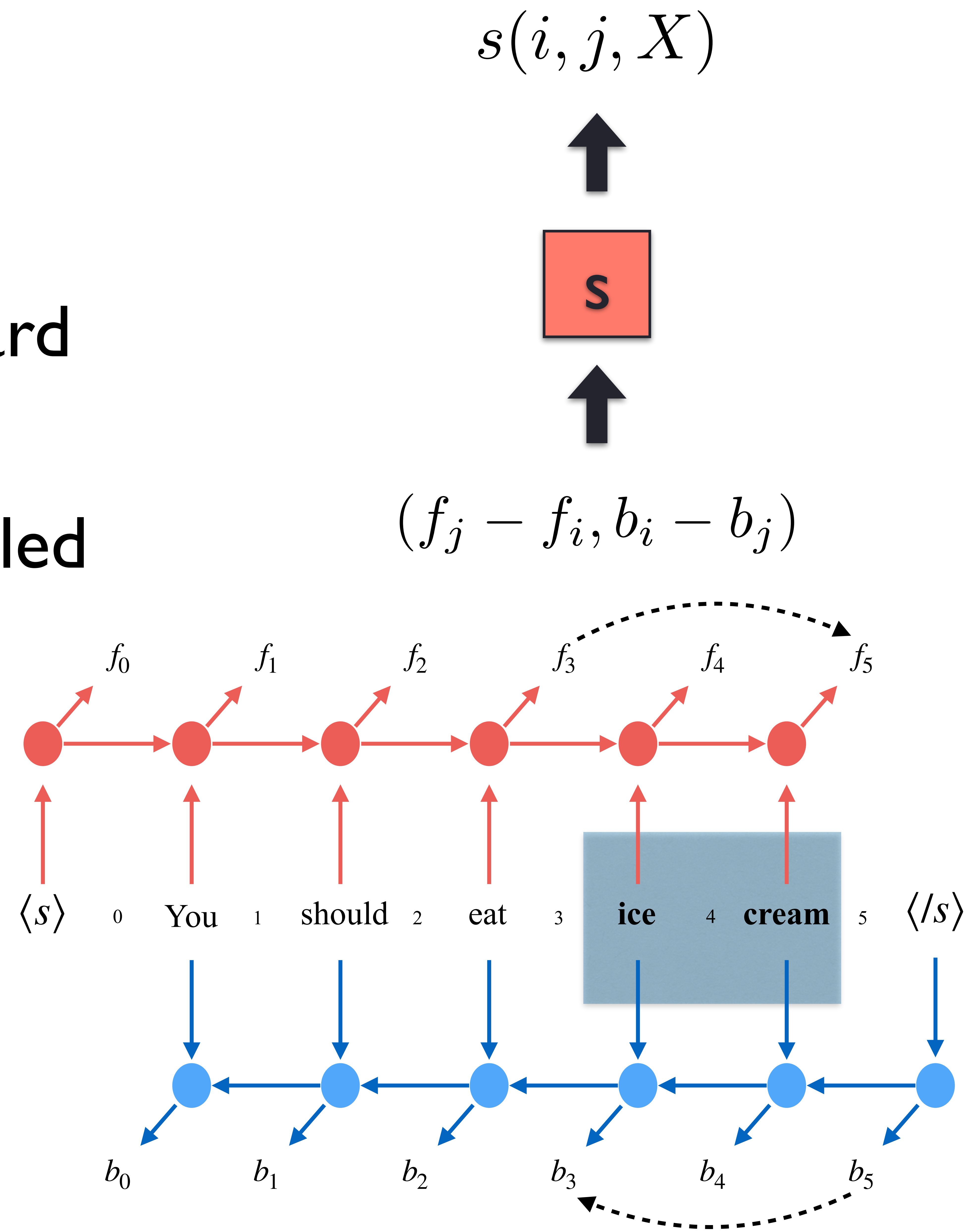
- Linear Time Constituency Parsing using dynamic programming
  - Going slower in order to go faster: $O(n^3) \rightarrow O(n^4) \rightarrow O(n)$
- Cube Pruning to speed up Incremental Parsing with Dynamic Programming
  - From $O(n\ b^2)$ to $O(n\ b \log b)$
- An improved loss function for Loss-Augmented Decoding
  - 2nd highest accuracy among single systems trained on PTB only

$$O(2^n) \rightarrow O(n^3) \rightarrow O(n^4) \rightsquigarrow O(nb^2) \rightsquigarrow O(nb \log b)$$

Span Parsing

- Span differences are taken from an encoder (in our case: a bi-LSTM)
- A span is scored and labeled by a feed-forward network.
- The score of a tree is the sum of all the labeled span scores

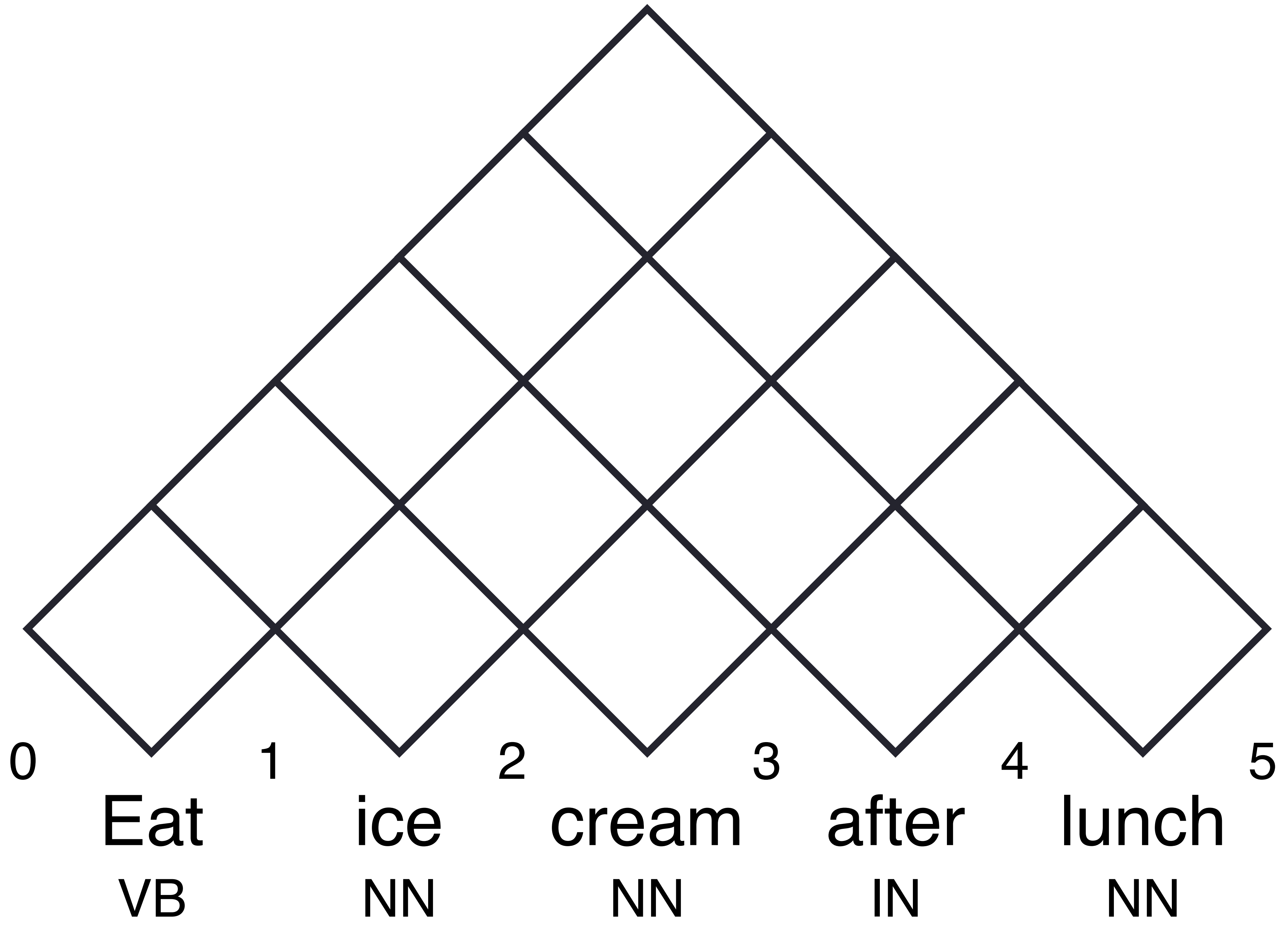
$$s_{tree}(t) = \sum_{(i,j,X)\in t} s(i,j,X)$$



Cross + Huang 2016 Stern et al. 2017 Wang + Chang 2016

Incremental Span Parsing Example

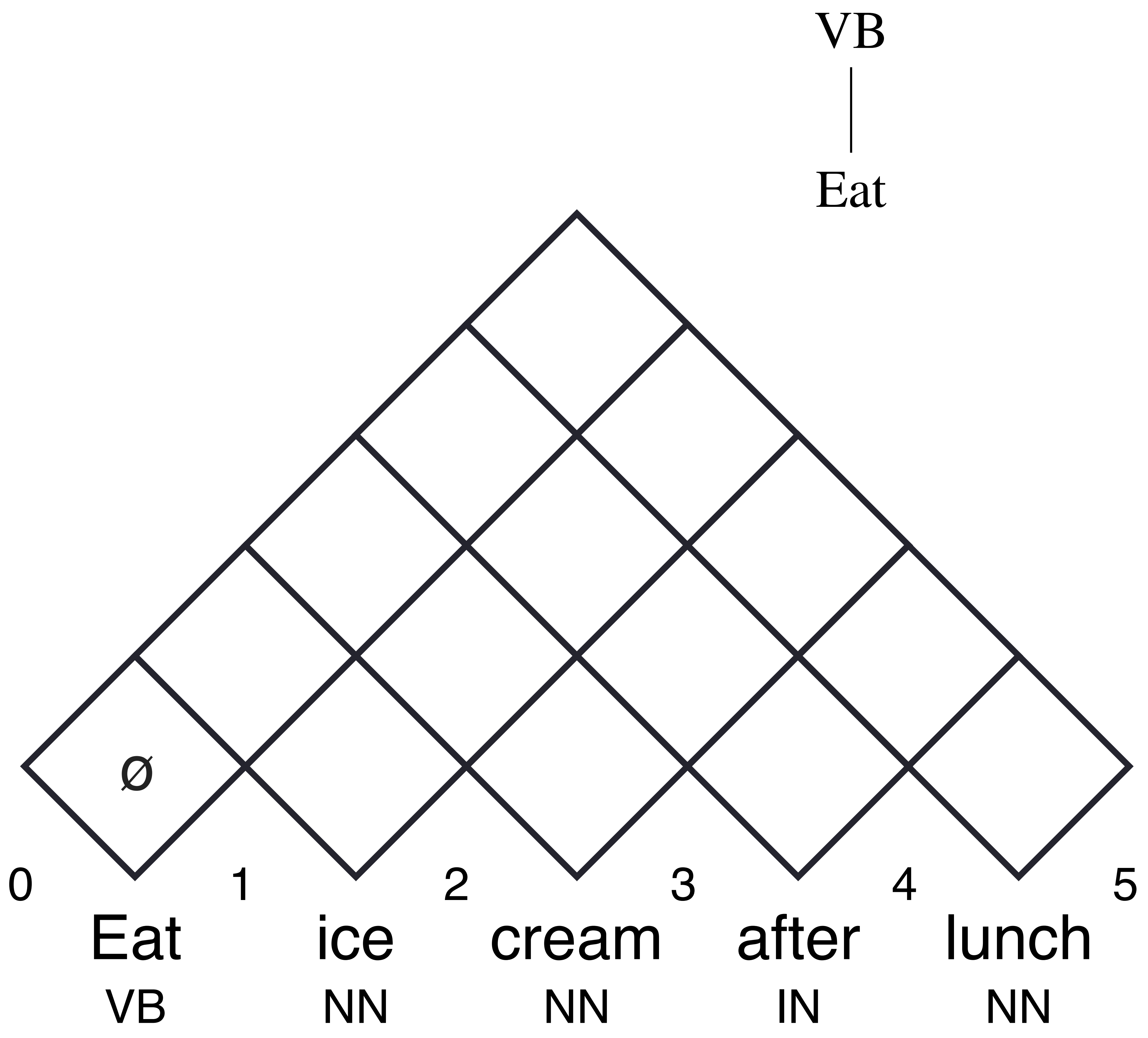
	Action	Label	Stack



Cross + Huang 2016

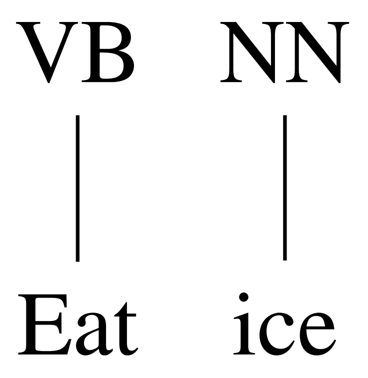
Incremental Span Parsing Example

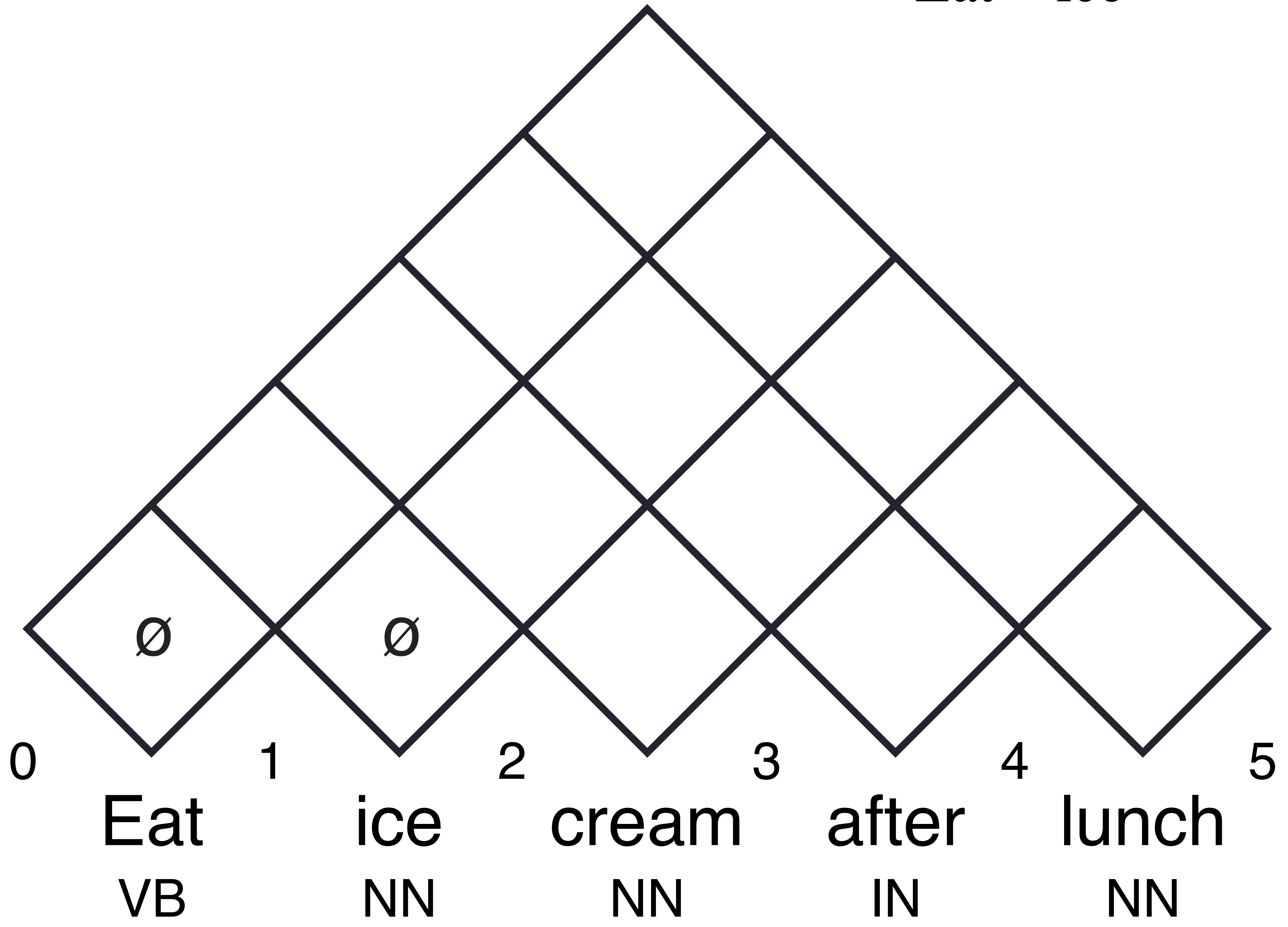
	Action	Label	Stack
1	Shift	ø	(0, 1)



Cross + Huang 2016

Incremental Span Parsing Example

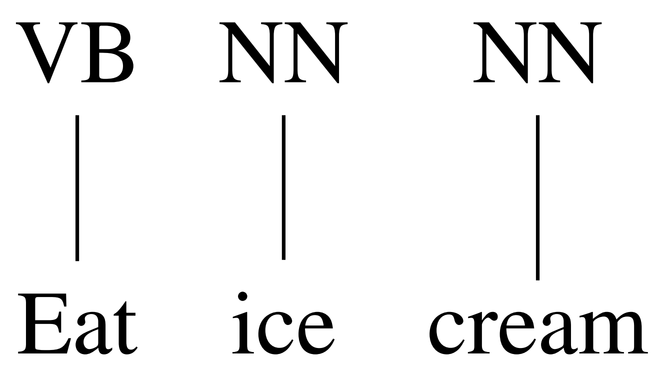


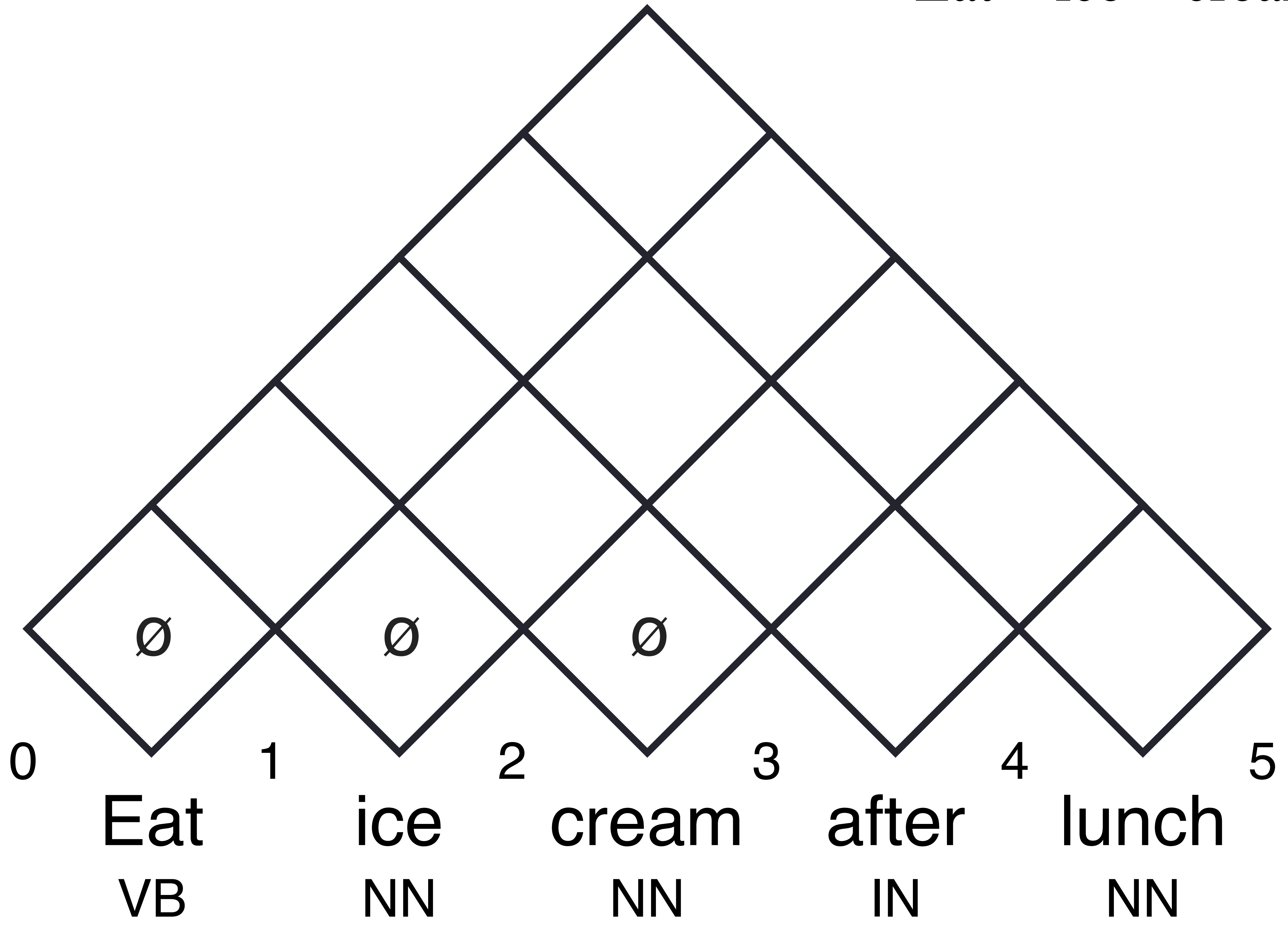


	Action	Label	Stack
1	Shift	ø	(0, 1)
2	Shift	ø	(0, 1) (1, 2)

Cross + Huang 2016

Incremental Span Parsing Example

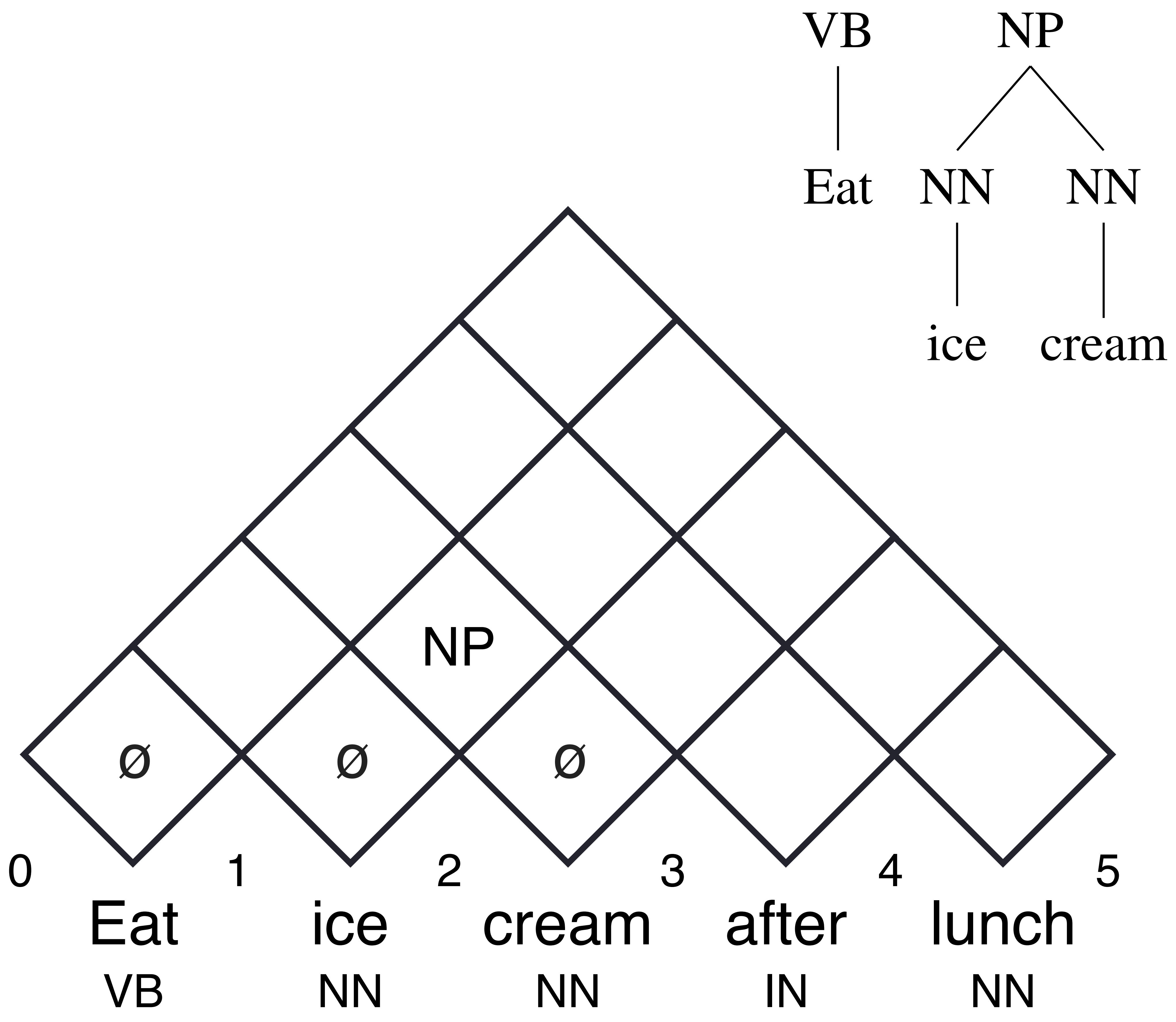




	Action	Label	Stack
1	Shift	ø	(0, 1)
2	Shift	ø	(0, 1) (1, 2)
3	Shift	ø	(0, 1) (1, 2) (2, 3)

Cross + Huang 2016

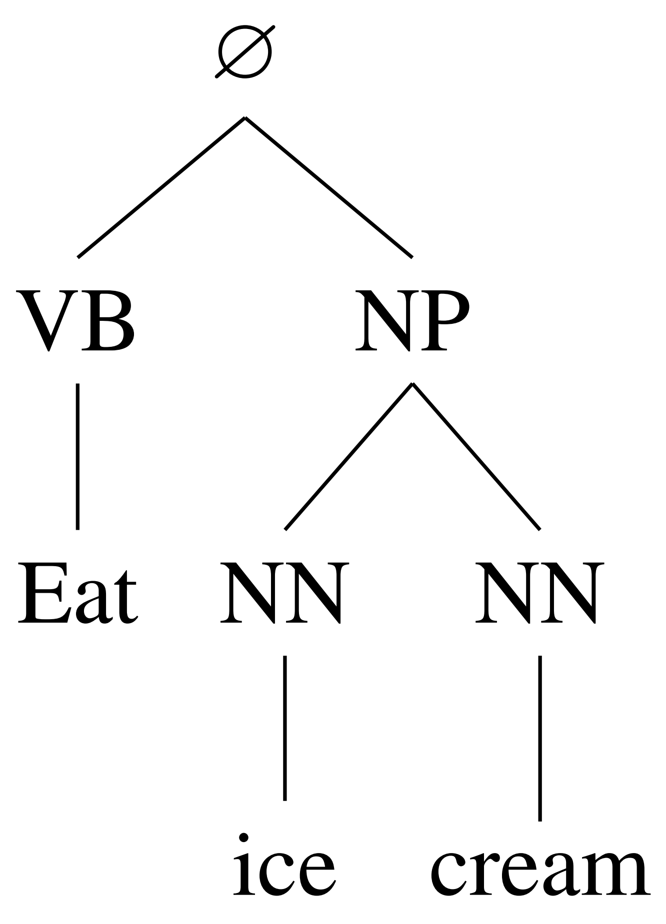
Incremental Span Parsing Example

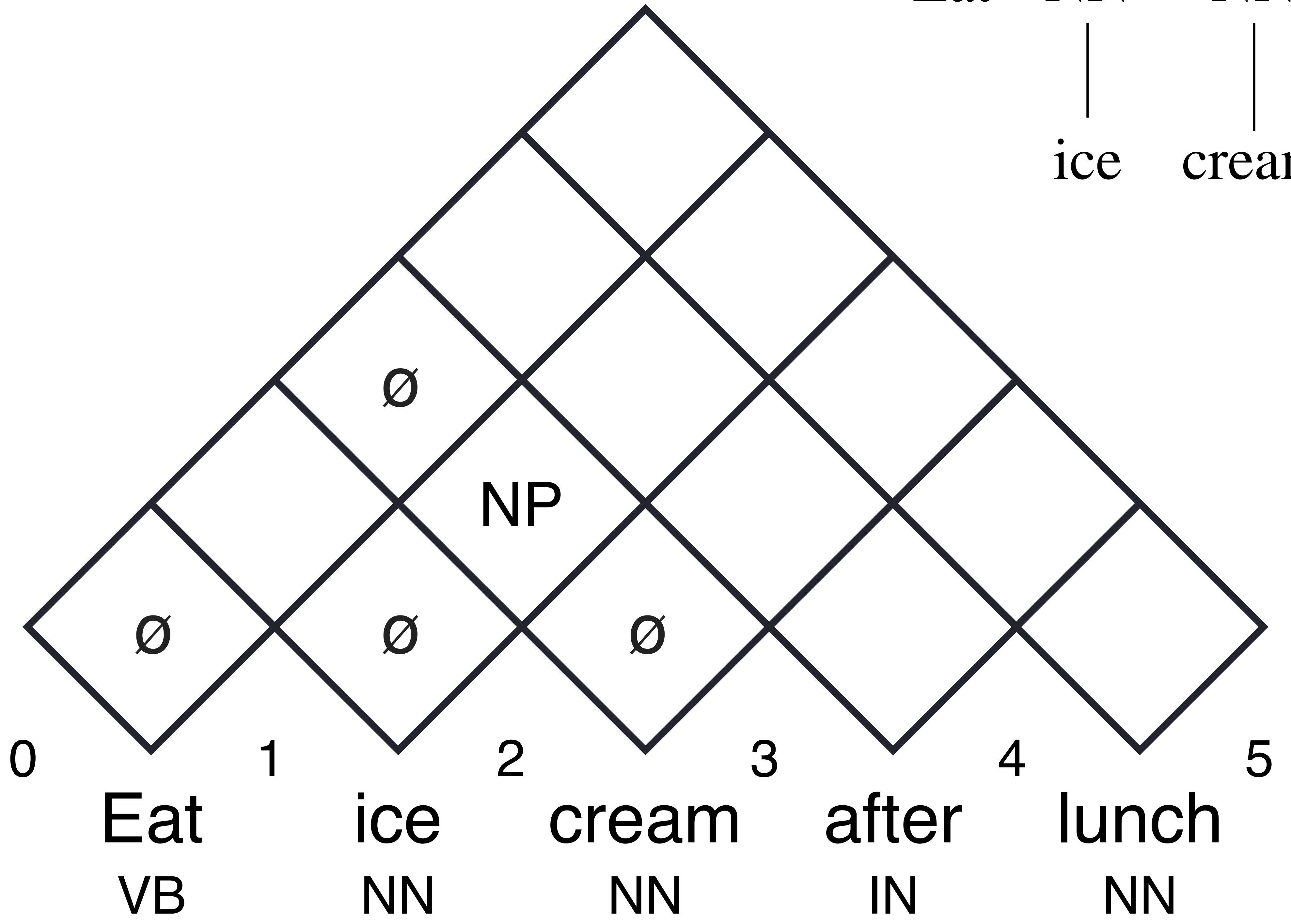


	Action	Label	Stack
1	Shift	ø	(0, 1)
2	Shift	ø	(0, 1) (1, 2)
3	Shift	ø	(0, 1) (1, 2) (2, 3)
4	Reduce	NP	(0, 1) (1, 3)

Cross + Huang 2016

Incremental Span Parsing Example

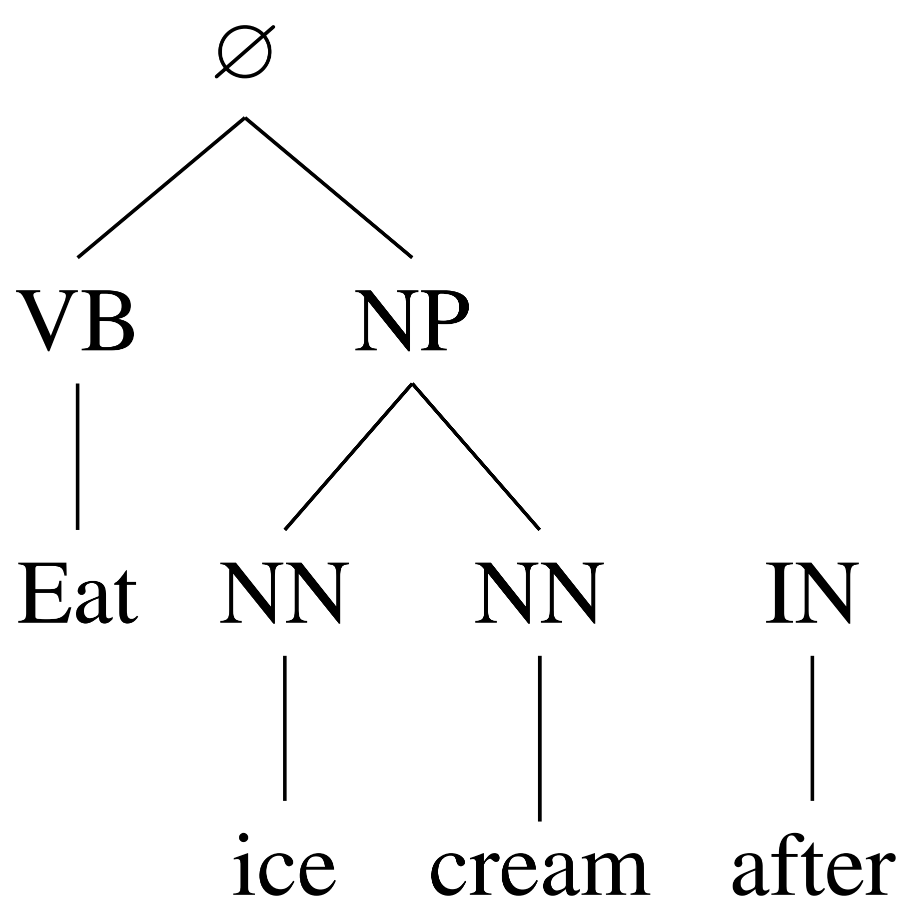


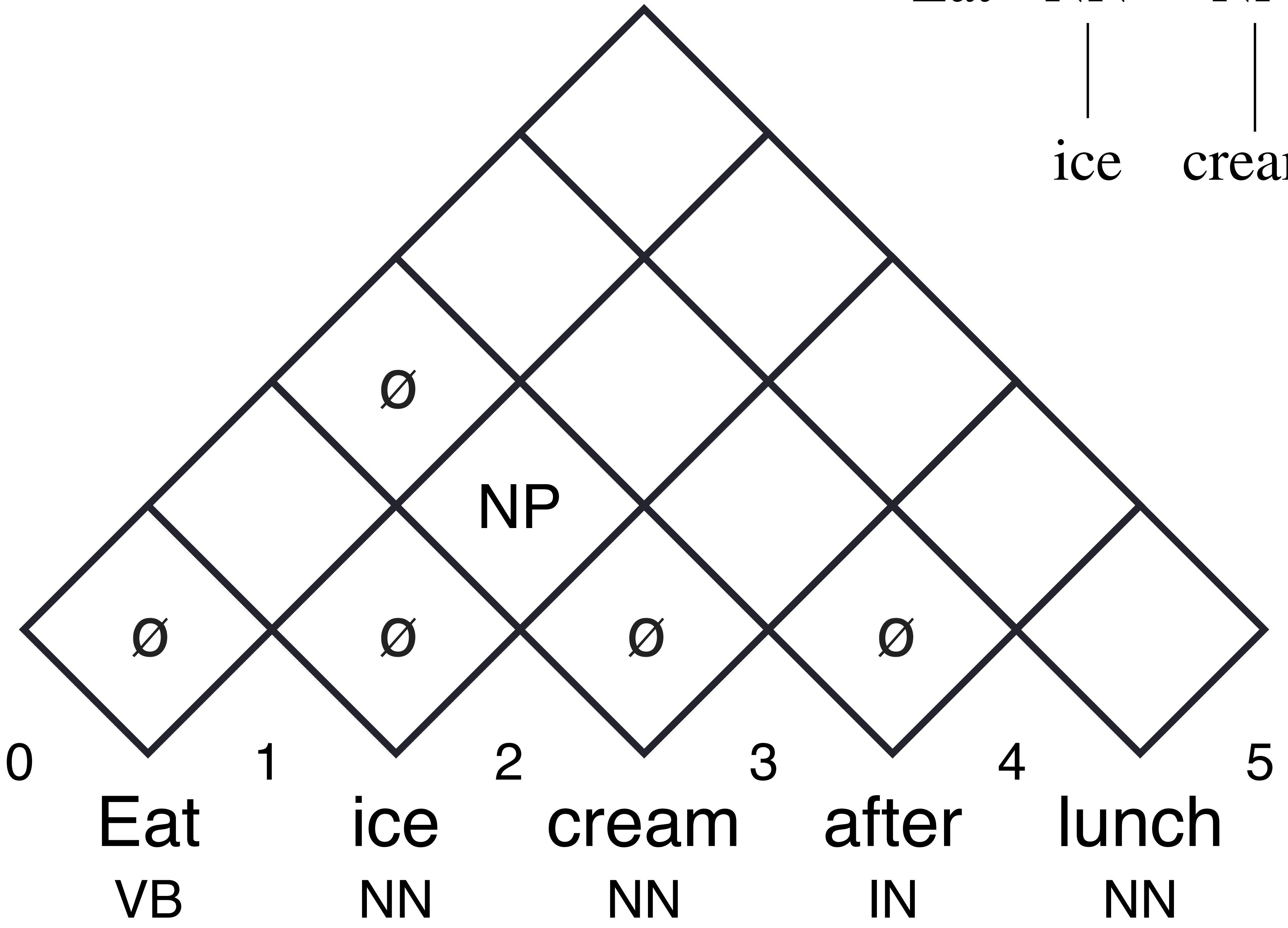


	Action	Label	Stack
1	Shift	ø	(0, 1)
2	Shift	ø	(0, 1) (1, 2)
3	Shift	ø	(0, 1) (1, 2) (2, 3)
4	Reduce	NP	(0, 1) (1, 3)
5	Reduce	ø	(0, 3)

Cross + Huang 2016

Incremental Span Parsing Example

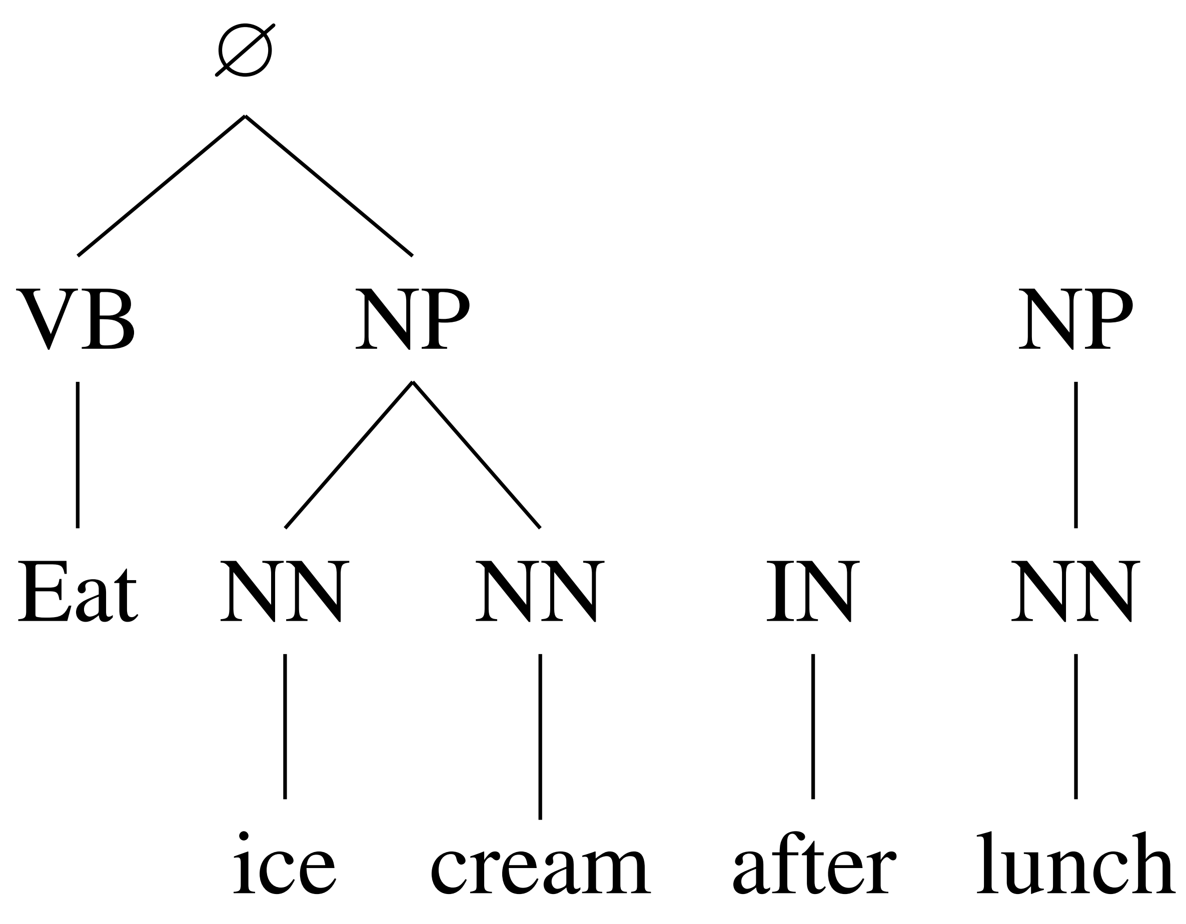


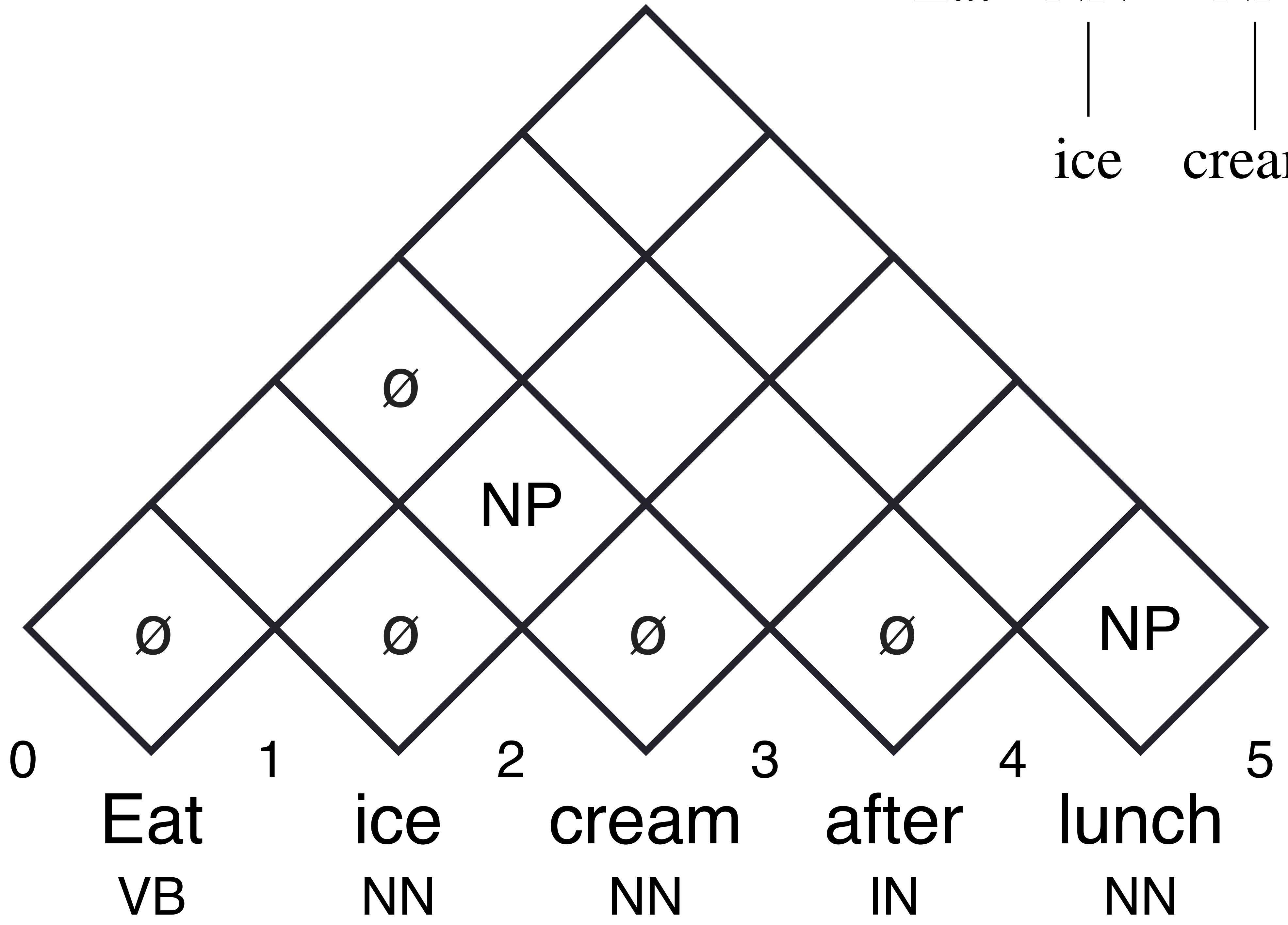


	Action	Label	Stack
1	Shift	ø	(0, 1)
2	Shift	ø	(0, 1) (1, 2)
3	Shift	ø	(0, 1) (1, 2) (2, 3)
4	Reduce	NP	(0, 1) (1, 3)
5	Reduce	ø	(0, 3)
6	Shift	ø	(0, 3) (3, 4)

Cross + Huang 2016

Incremental Span Parsing Example

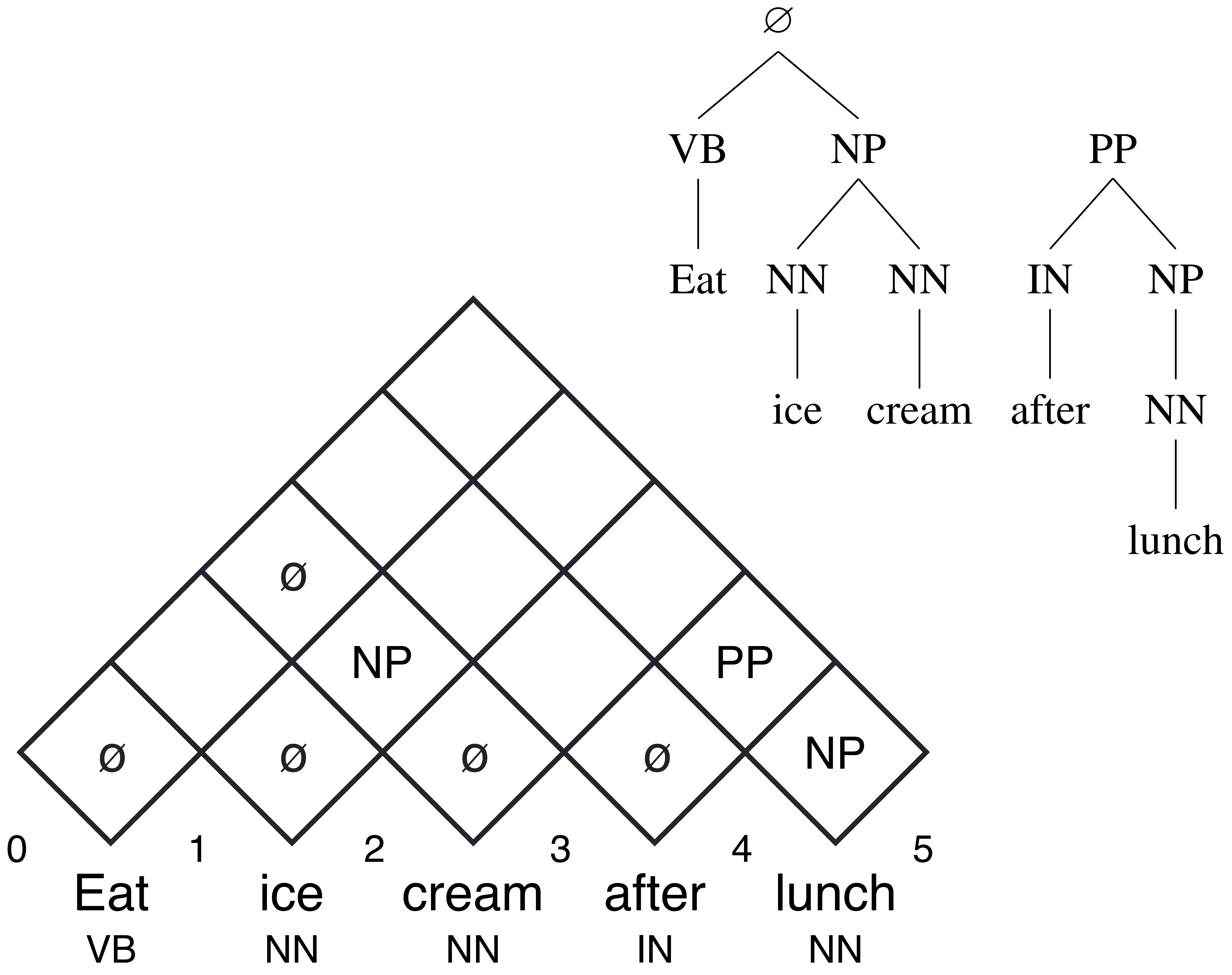




	Action	Label	Stack
1	Shift	ø	(0, 1)
2	Shift	ø	(0, 1) (1, 2)
3	Shift	ø	(0, 1) (1, 2) (2, 3)
4	Reduce	NP	(0, 1) (1, 3)
5	Reduce	ø	(0, 3)
6	Shift	ø	(0, 3) (3, 4)
7	Shift	NP	(0, 3) (3, 4) (4, 5)

Cross + Huang 2016

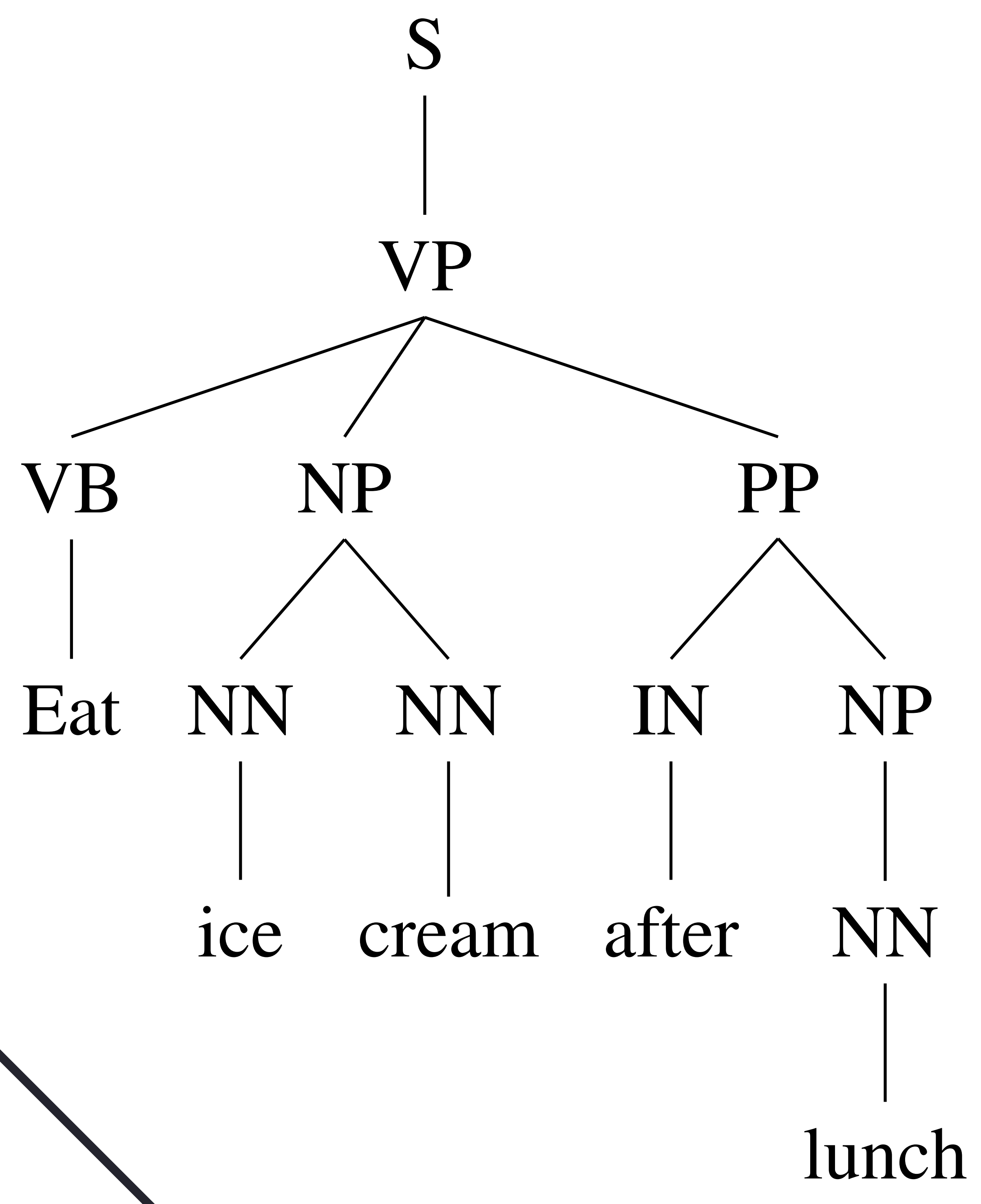
Incremental Span Parsing Example

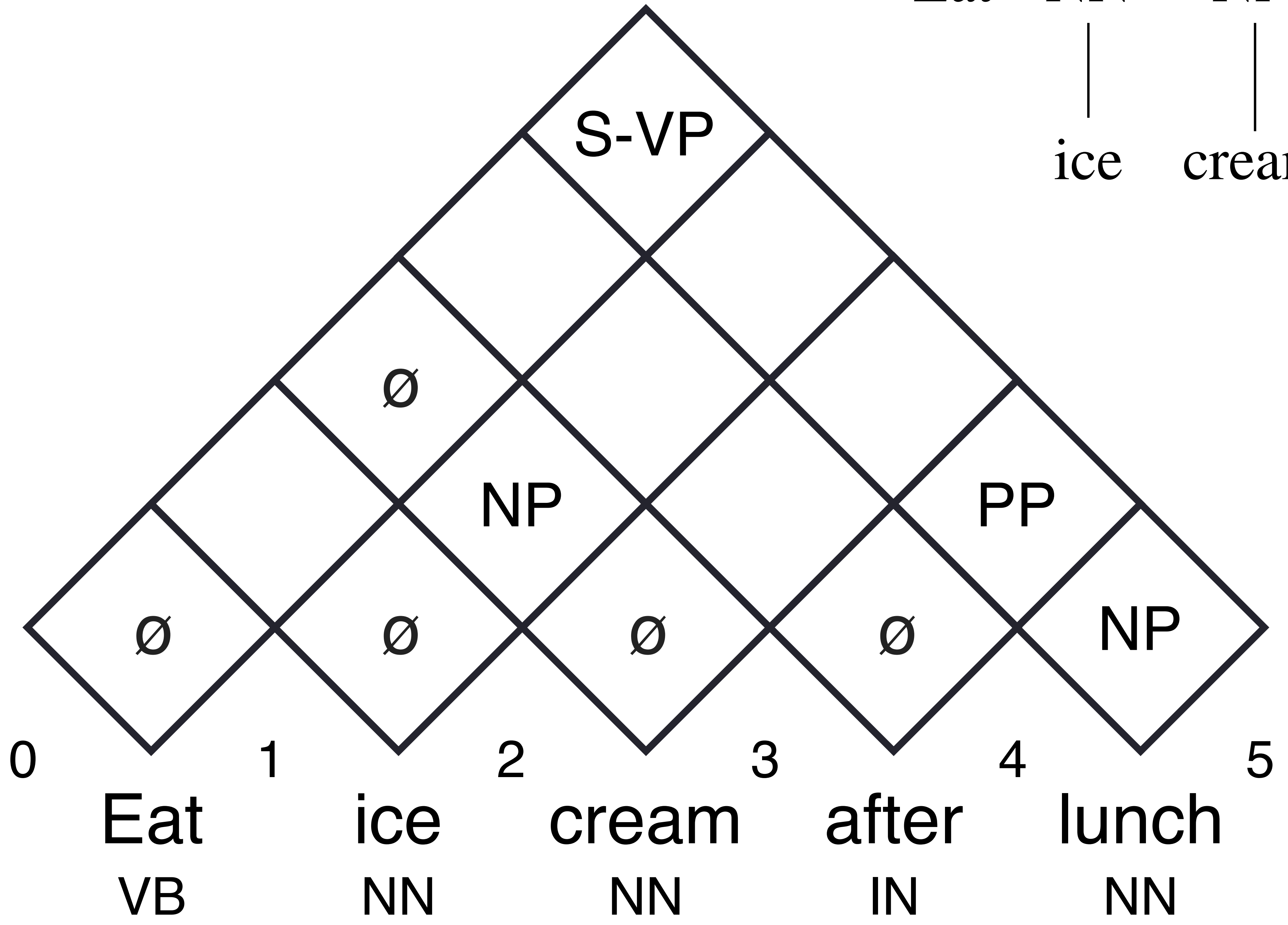


	Action	Label	Stack
1	Shift	ø	(0, 1)
2	Shift	ø	(0, 1) (1, 2)
3	Shift	ø	(0, 1) (1, 2) (2, 3)
4	Reduce	NP	(0, 1) (1, 3)
5	Reduce	ø	(0, 3)
6	Shift	ø	(0, 3) (3, 4)
7	Shift	NP	(0, 3) (3, 4) (4, 5)
8	Reduce	PP	(0, 3) (3, 5)

Cross + Huang 2016

Incremental Span Parsing Example



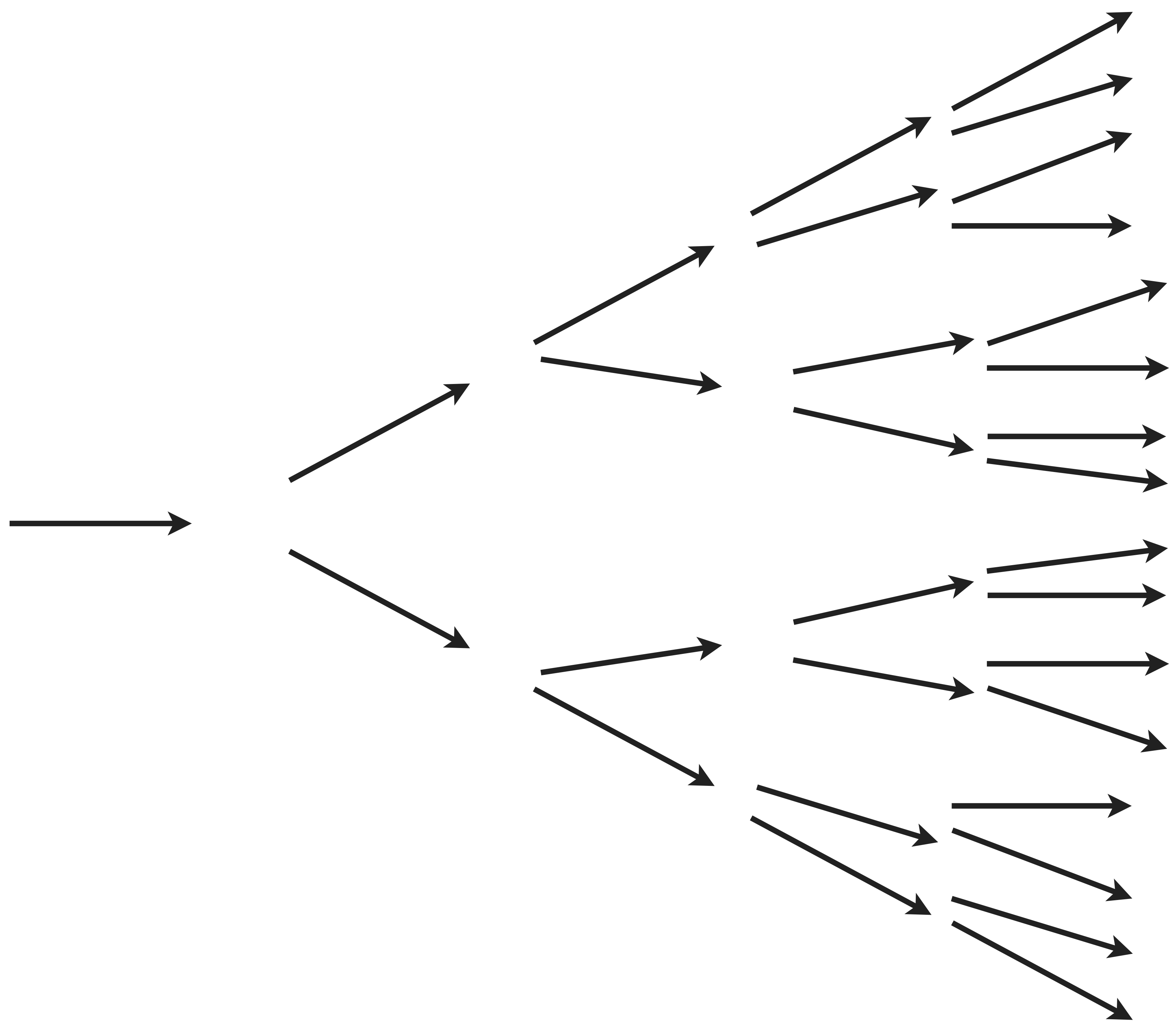


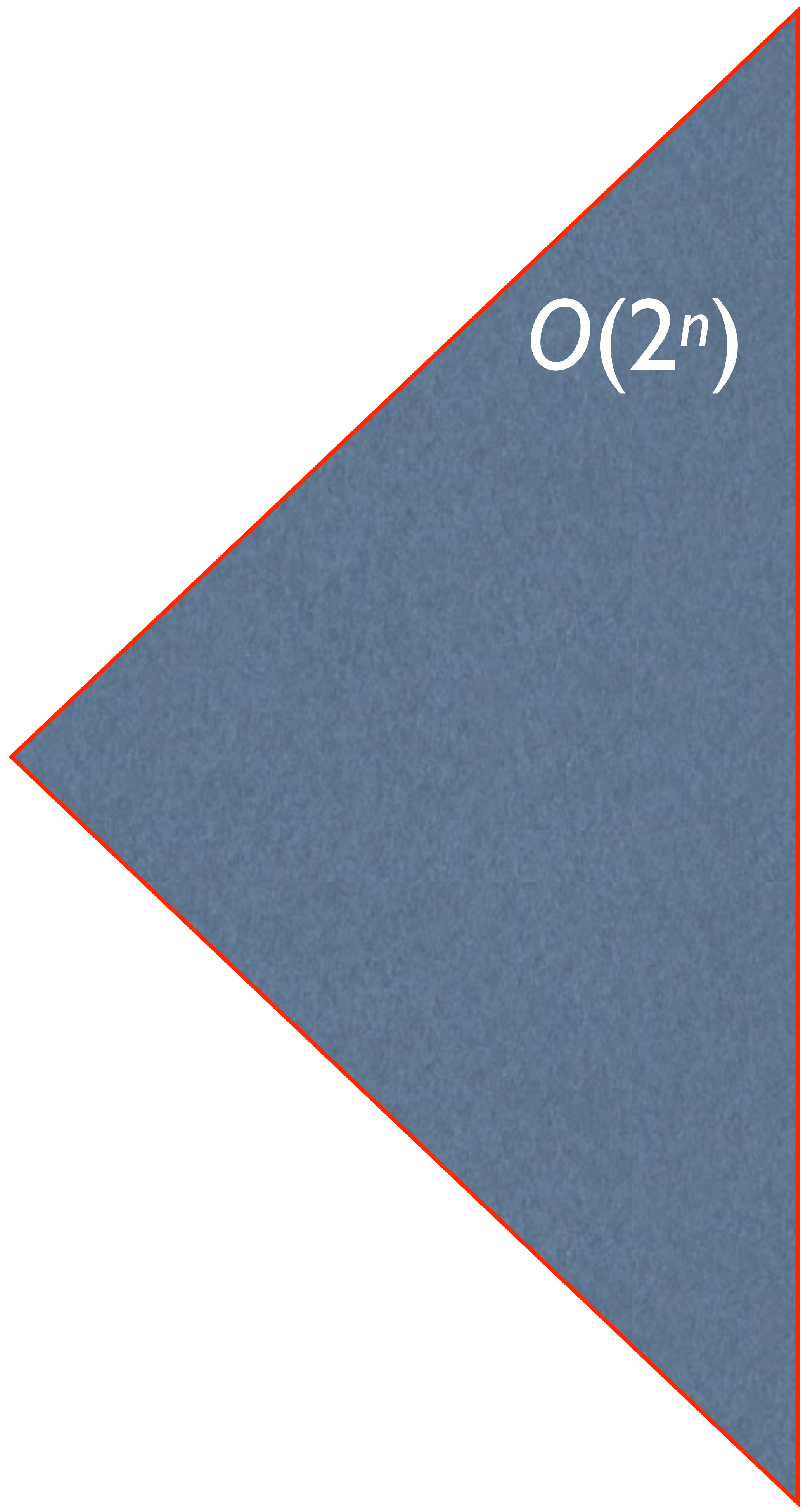
	Action	Label	Stack
1	Shift	ø	(0, 1)
2	Shift	ø	(0, 1) (1, 2)
3	Shift	ø	(0, 1) (1, 2) (2, 3)
4	Reduce	NP	(0, 1) (1, 3)
5	Reduce	ø	(0, 3)
6	Shift	ø	(0, 3) (3, 4)
7	Shift	NP	(0, 3) (3, 4) (4, 5)
8	Reduce	PP	(0, 3) (3, 5)
9	Reduce	S-VP	(0, 5)

Cross + Huang 2016

How Many Possible Parsing Paths?

- 2 actions per state.
- $O(2^n)$





Equivalent Stacks?

- Observe that all stacks that end with (i, j) will be treated the same!
  - …Until (i, j) is popped off.

$[(0, 2), (2, 7), (7, 9)]$

$[(0, 3), (3, 7), (7, 9)]$

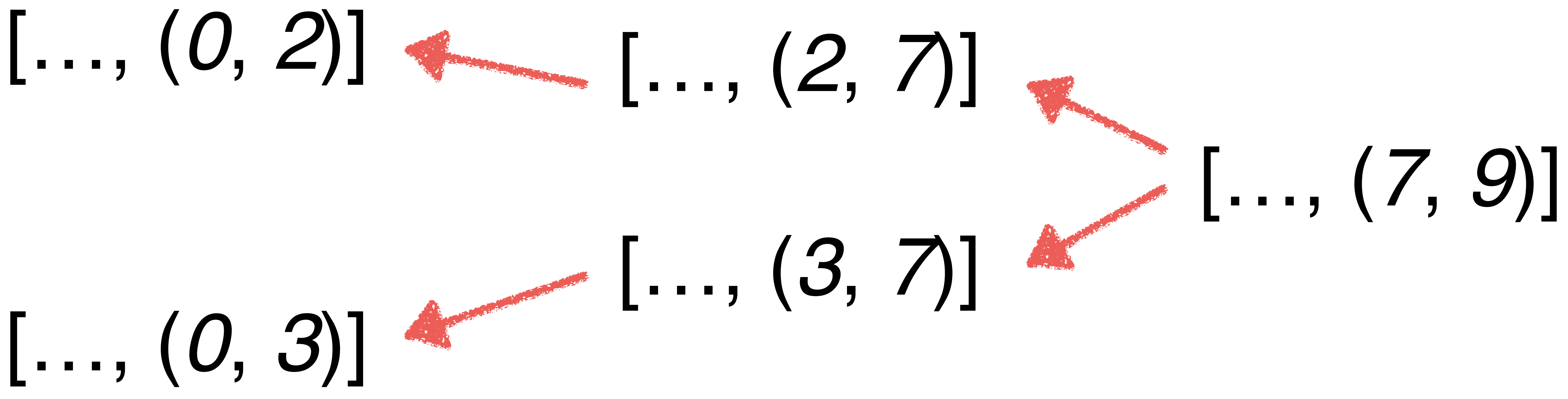
becomes $[\ldots, (7, 9)]$

- So we can treat these as "temporarily equivalent", and merge.

Graph-Structured Stack (Tomita 1988; Huang + Sagae 2010)

Equivalent Stacks?

- Observe that all stacks that end with (i, j) will be treated the same!
  - …Until (i, j) is popped off.



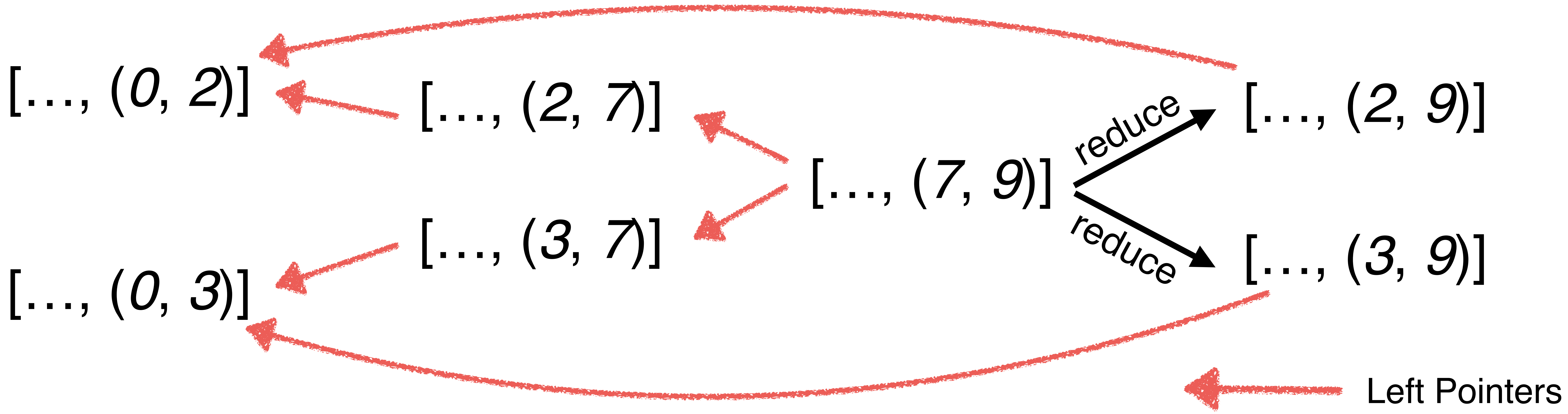
- This is our new stack representation.

Left Pointers

Graph-Structured Stack (Tomita 1988; Huang + Sagae 2010)

Equivalent Stacks?

- Observe that all stacks that end with (i, j) will be treated the same!
  - …Until (i, j) is popped off.

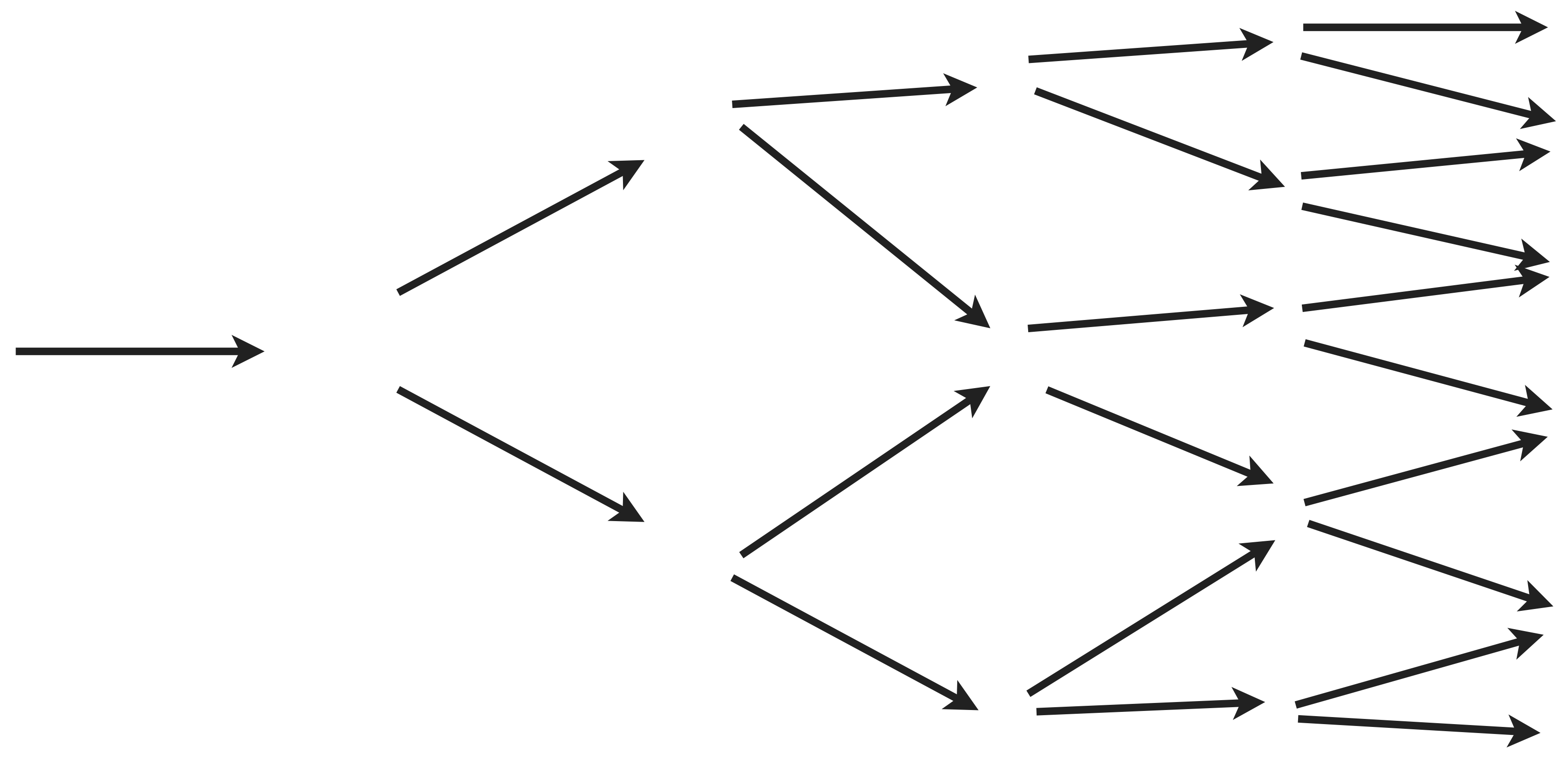


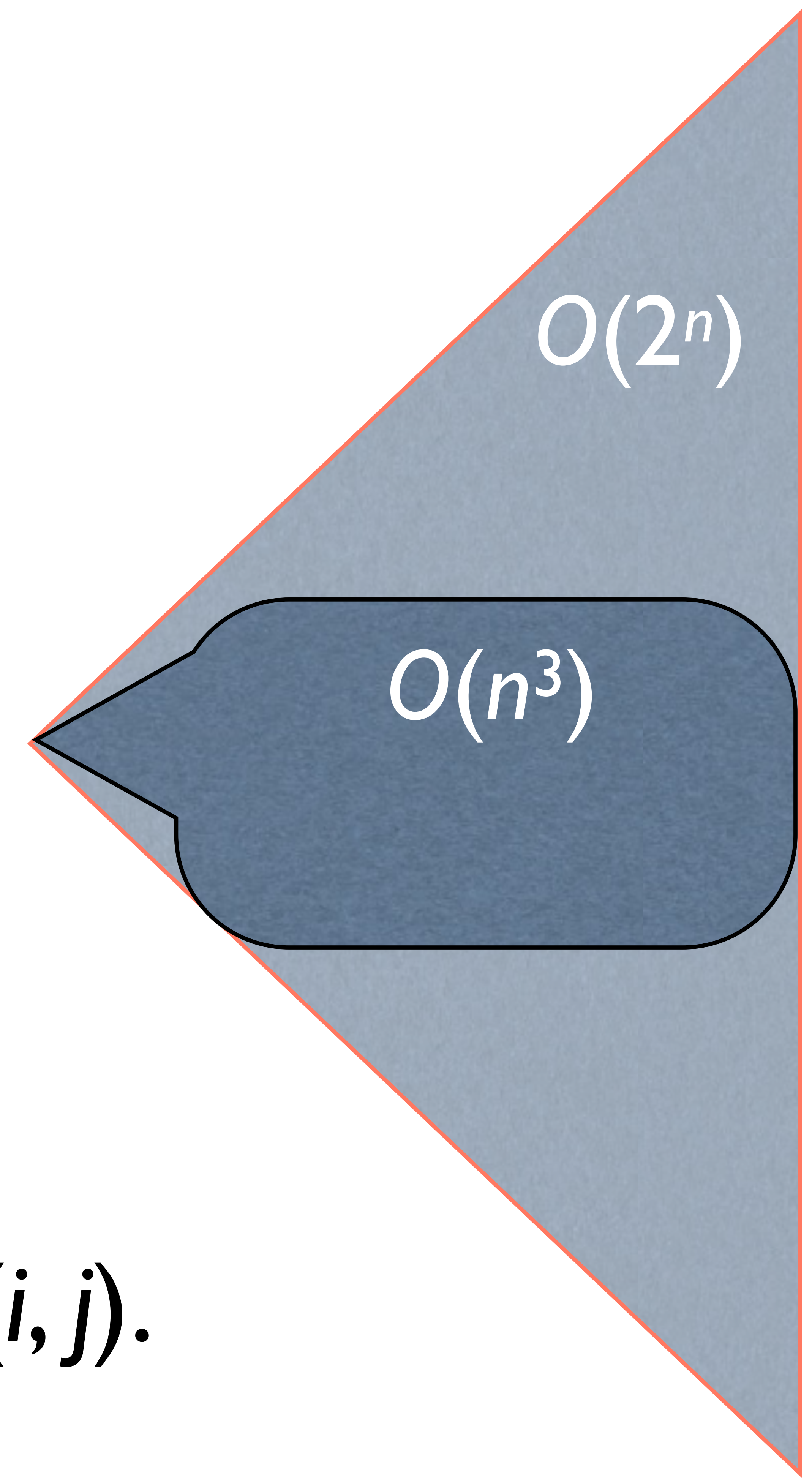
Reduce Actions: $\dfrac{[\ldots, (k, i)] \quad [\ldots, (i, j)]}{[\ldots, (k, j)]}$ $O(n^3)$

Graph-Structured Stack (Tomita 1988; Huang + Sagae 2010)

Dynamic Programming: Merging Stacks

- Temporarily merging stacks will make our state space polynomial.

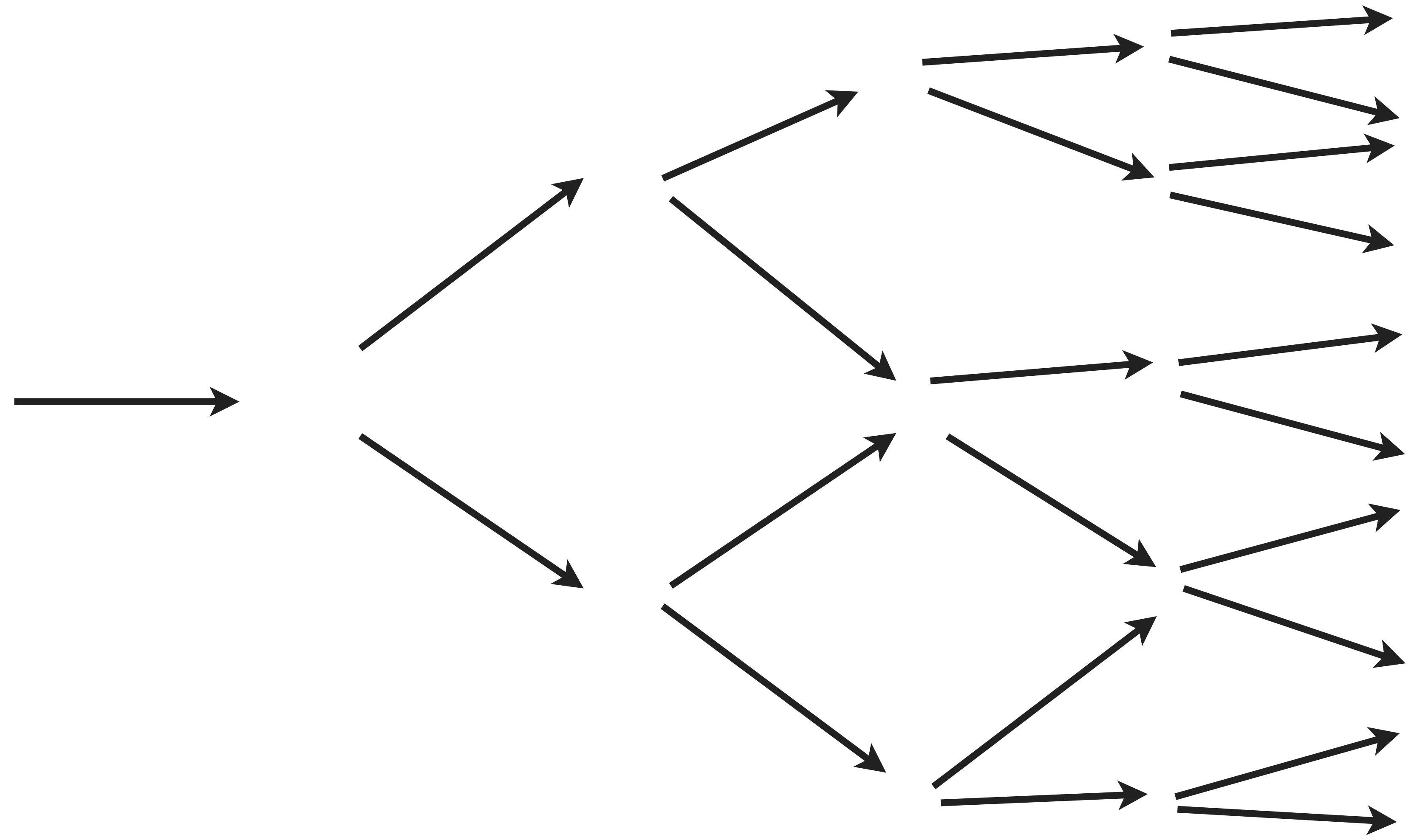


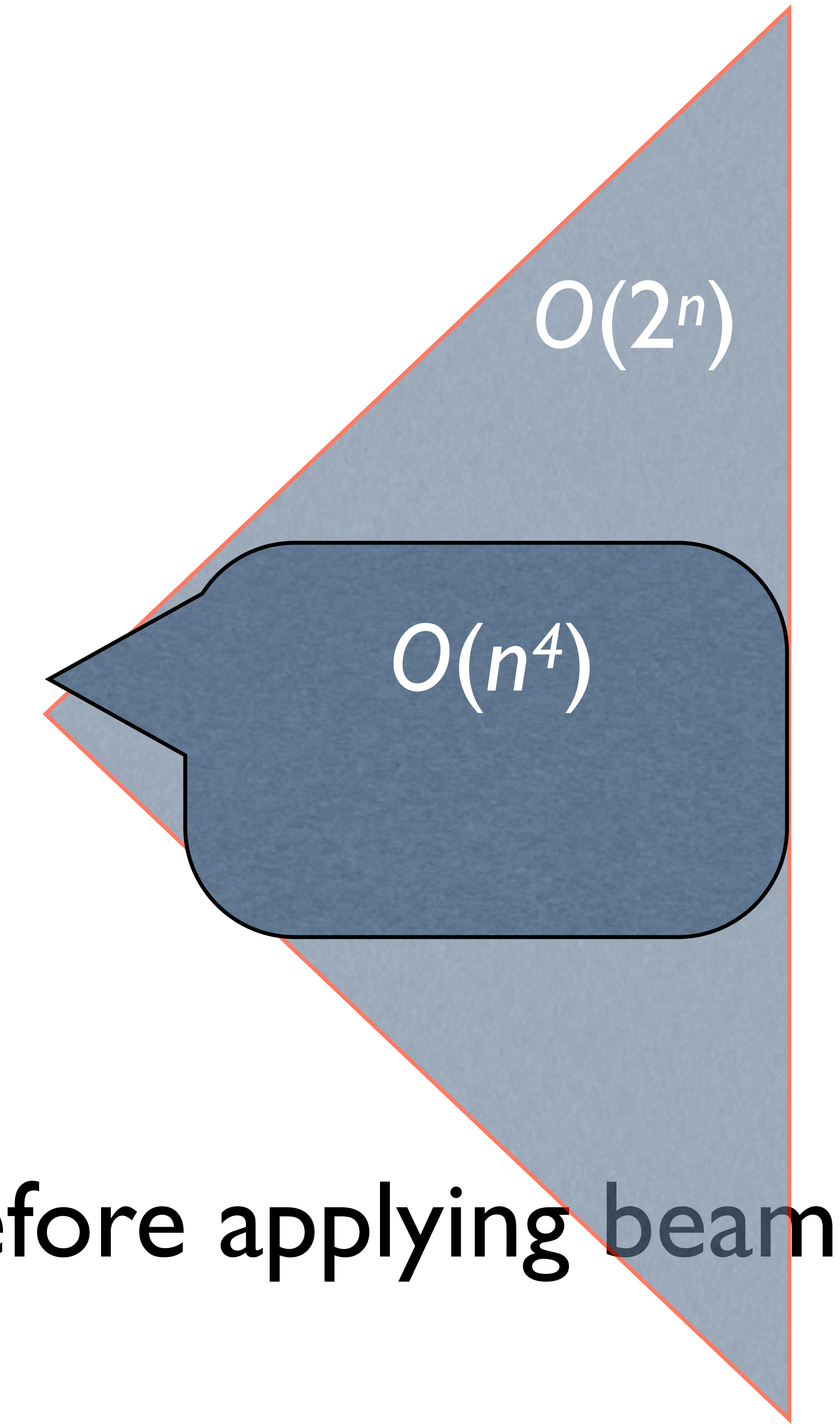


- And our parsing state is represented by top span (i, j).

Becoming Action Synchronous

- Shift-Reduce Parsers are traditionally action synchronous.
  - This makes beam-search straight forward.
  - We will also do the same

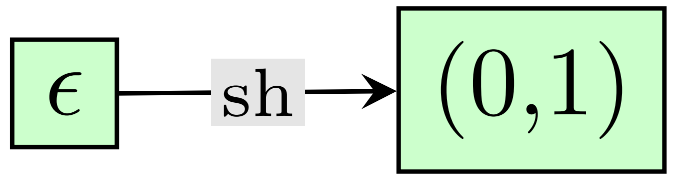




- But will show that this will slow down our DP (before applying beam-search)

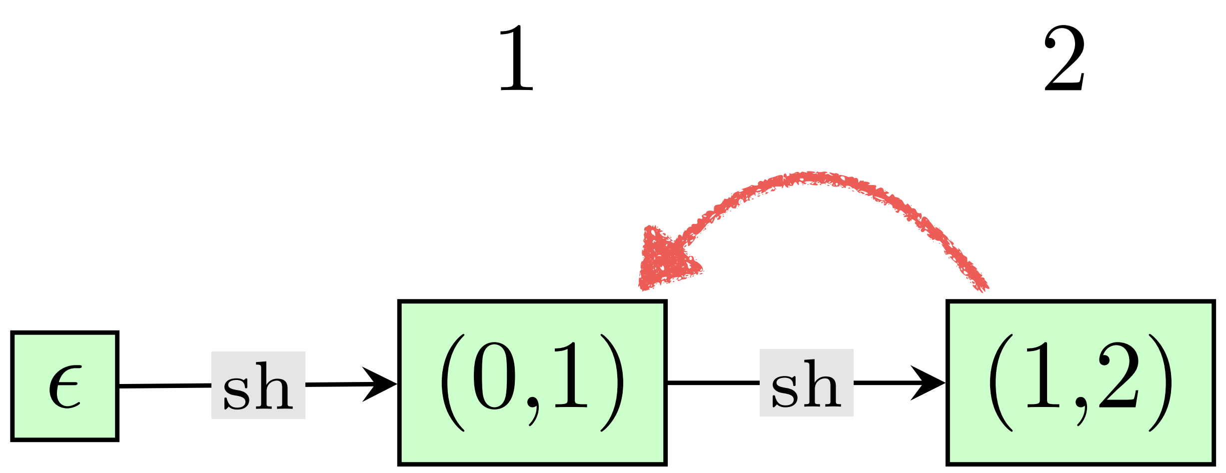
Action Synchronous Parsing Example

1



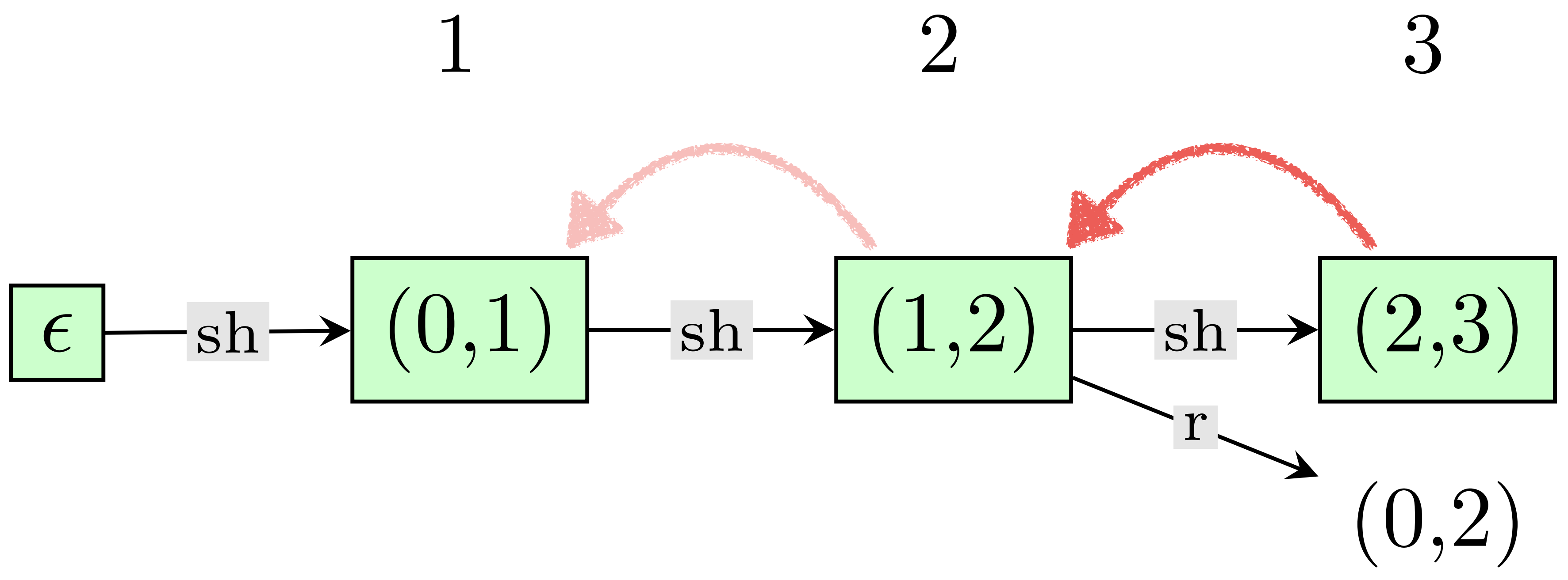
Gold:	Shift (0,1)

Action Synchronous Parsing Example



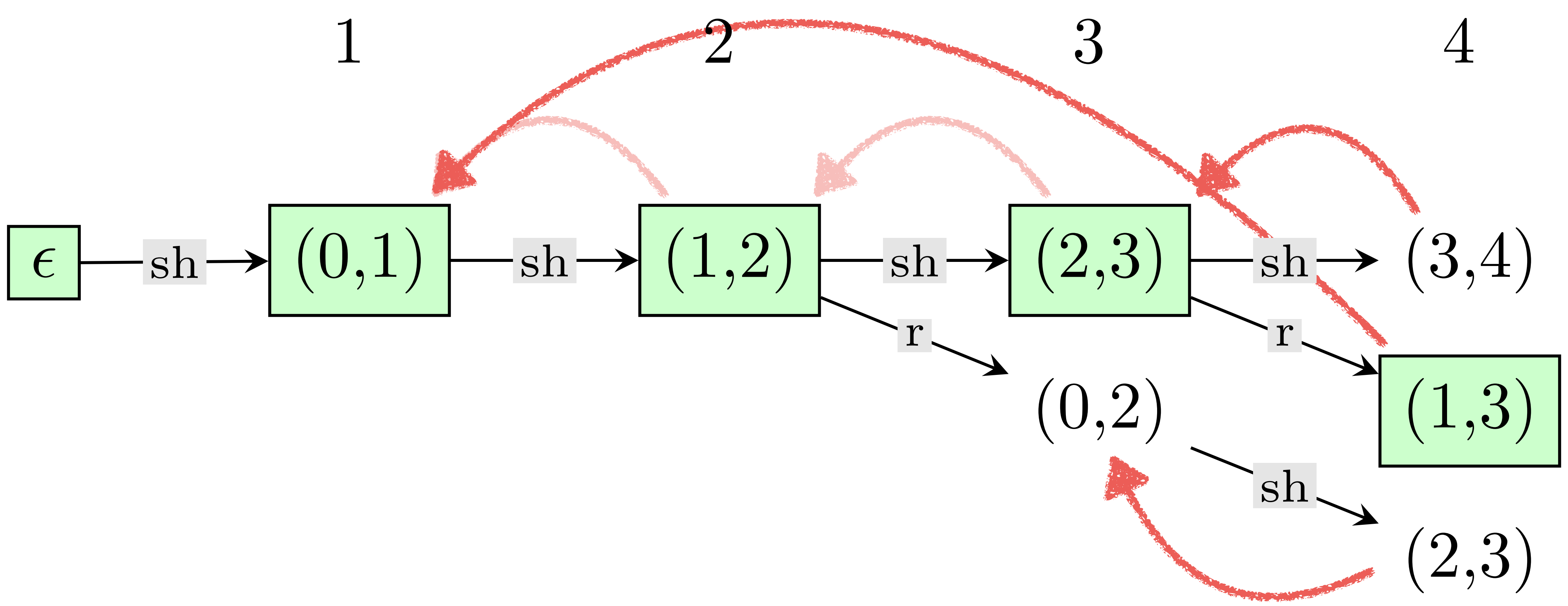
Gold:	Shift (0,1)	Shift (1,2)

Action Synchronous Parsing Example



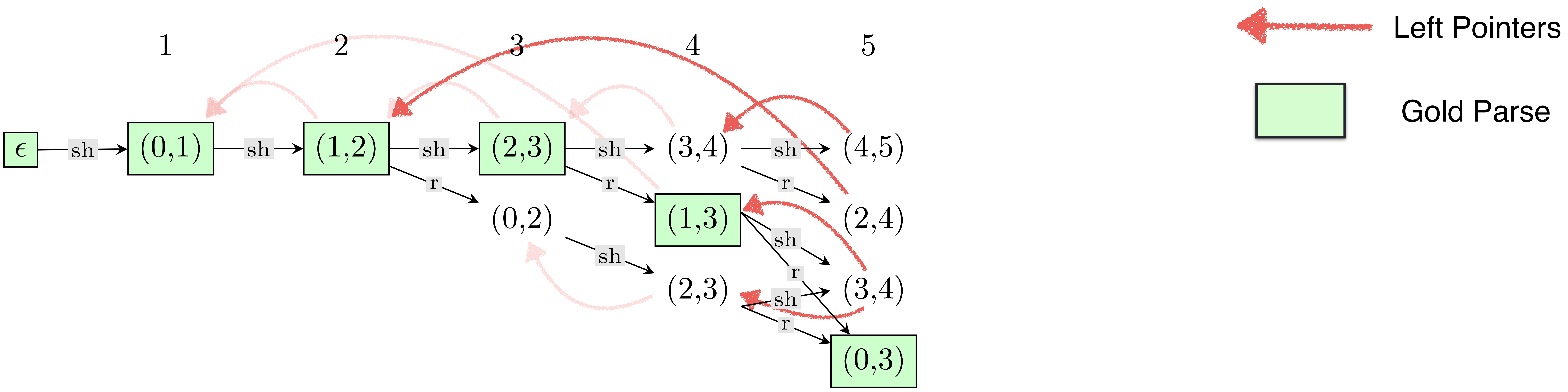
Gold:	Shift (0,1)	Shift (1,2)	Shift (2, 3)

Action Synchronous Parsing Example



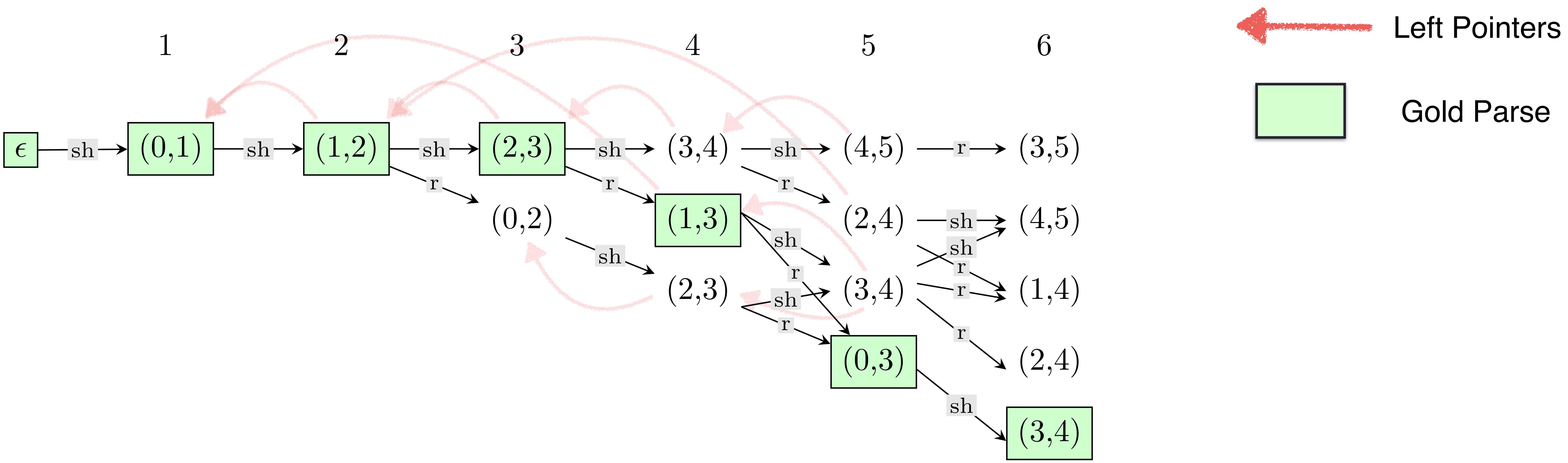
Gold:	Shift (0,1)	Shift (1,2)	Shift (2, 3)	Reduce (1, 3)

Action Synchronous Parsing Example



Gold:	Shift (0,1)	Shift (1,2)	Shift (2, 3)	Reduce (1, 3)	Reduce (0, 3)

Action Synchronous Parsing Example

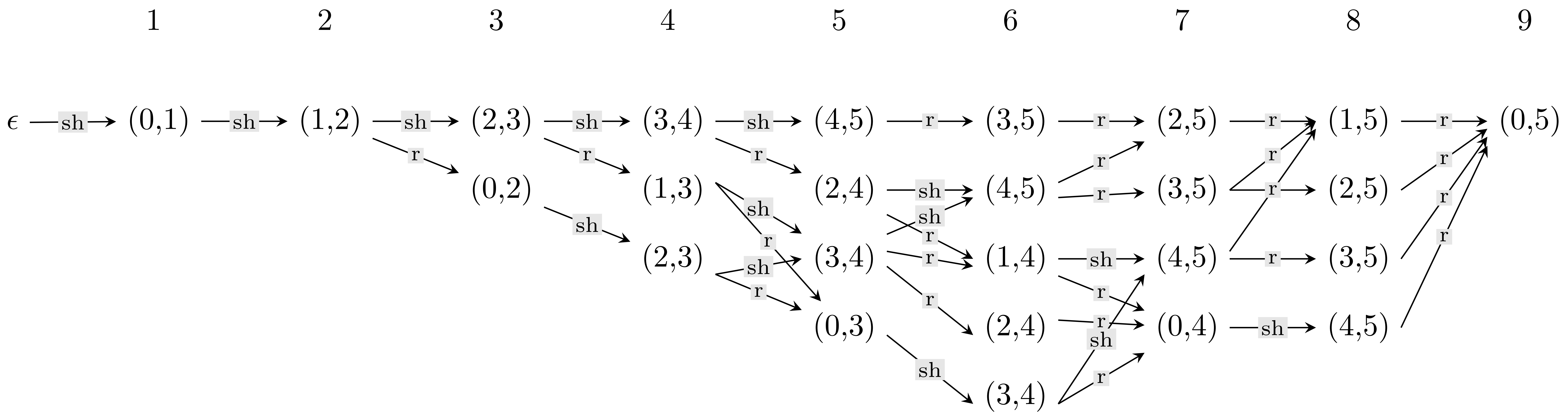


Gold:	Shift (0,1)	Shift (1,2)	Shift (2, 3)	Reduce (1, 3)	Reduce (0, 3)	Shift (3, 4)

Action Synchronous Parsing Example

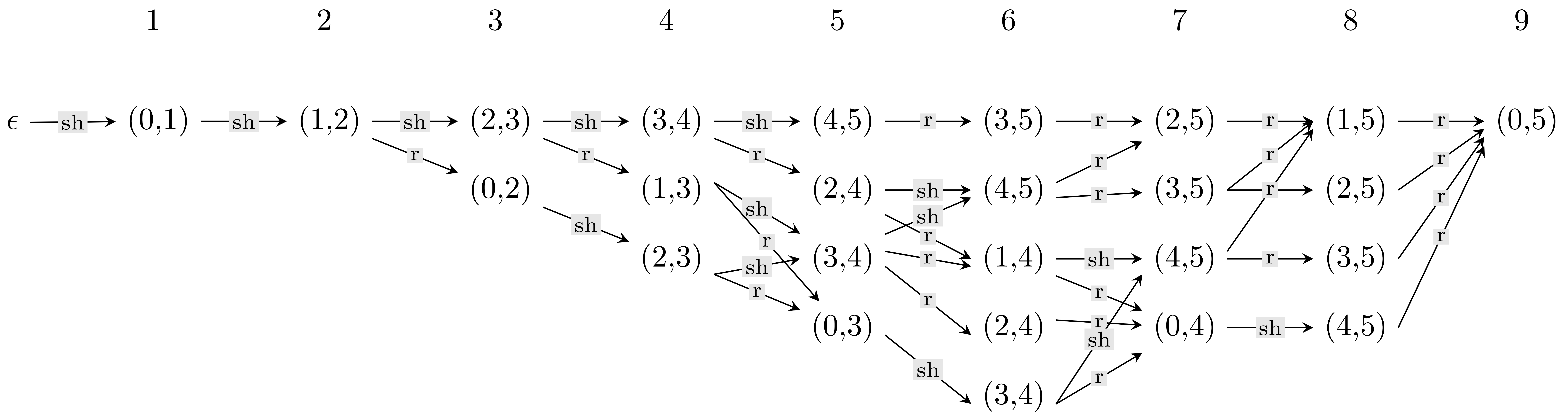
Gold:	Shift (0,1)	Shift (1,2)	Shift (2, 3)	Reduce (1, 3)	Reduce (0, 3)	Shift (3, 4)	Shift (4, 5)	Reduce (3, 5)	Reduce (0, 5)

Runtime Analysis: $O(n^4)$



Huang+Sagae 2010

Runtime Analysis: $O(n^4)$



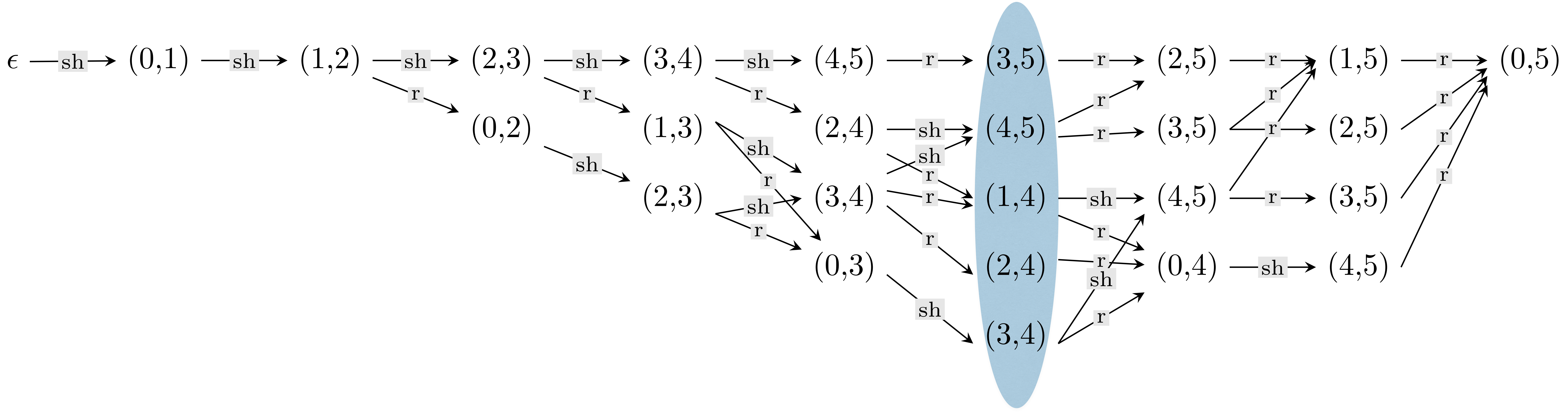
#steps: $2n - 1 = O(n)$

Huang+Sagae 2010

Runtime Analysis: $O(n^4)$

(i, j)

#states per step: $O(n^2)$



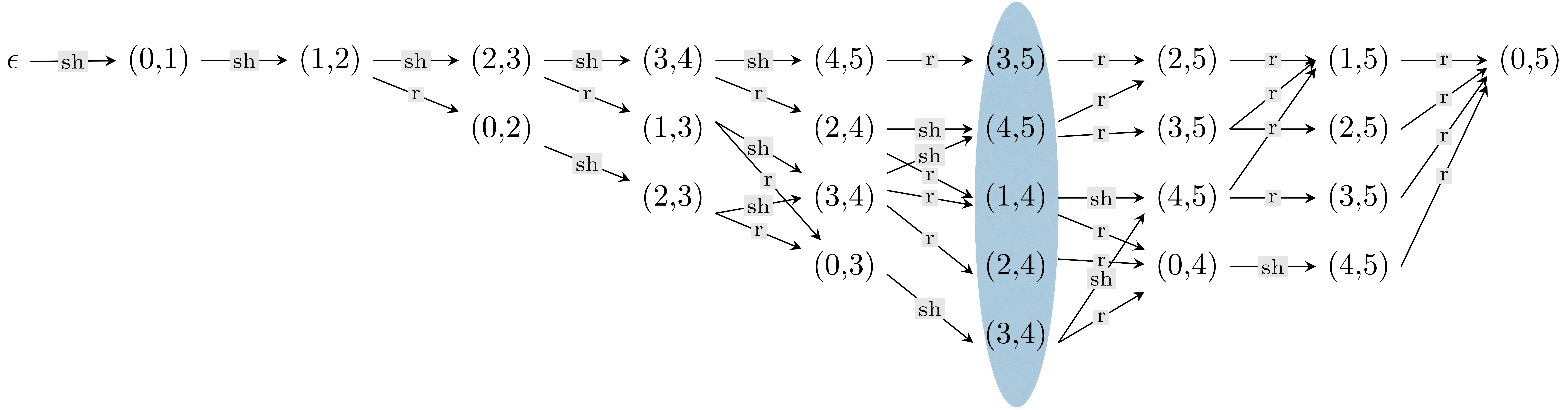
#steps: $2n - 1 = O(n)$

Huang+Sagae 2010

Runtime Analysis: $O(n^4)$

(i, j)

#states per step: $O(n^2)$



#steps: $2n - 1 = O(n)$

$O(n^3)$ states

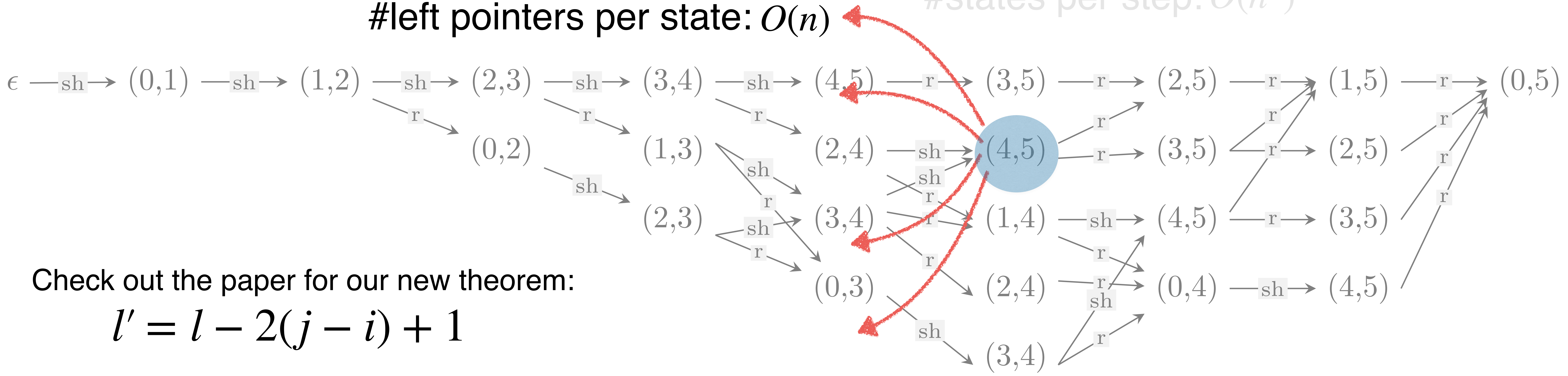
Huang+Sagae 2010

Runtime Analysis: $O(n^4)$

(i, j)

#states per step: $O(n^2)$

#left pointers per state: $O(n)$



Check out the paper for our new theorem:

$$l' = l - 2(j - i) + 1$$

Thanks to Dezhong Deng!

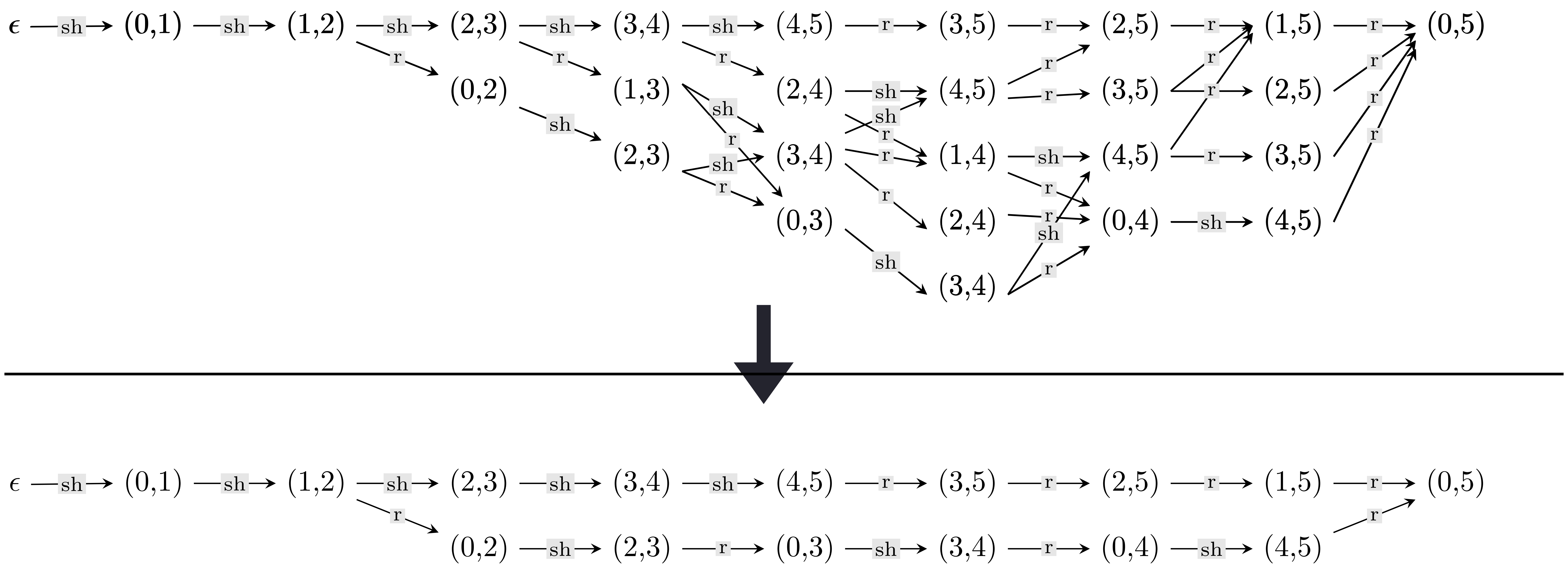
#steps: $2n - 1 = O(n)$

$O(n^3)$ states

$$\frac{l': [\ldots, (k, i)] \quad l: [\ldots, (i, j)]}{l{+}1: [\ldots, (k, j)]} \quad O(n^4)$$

Going slower to go faster

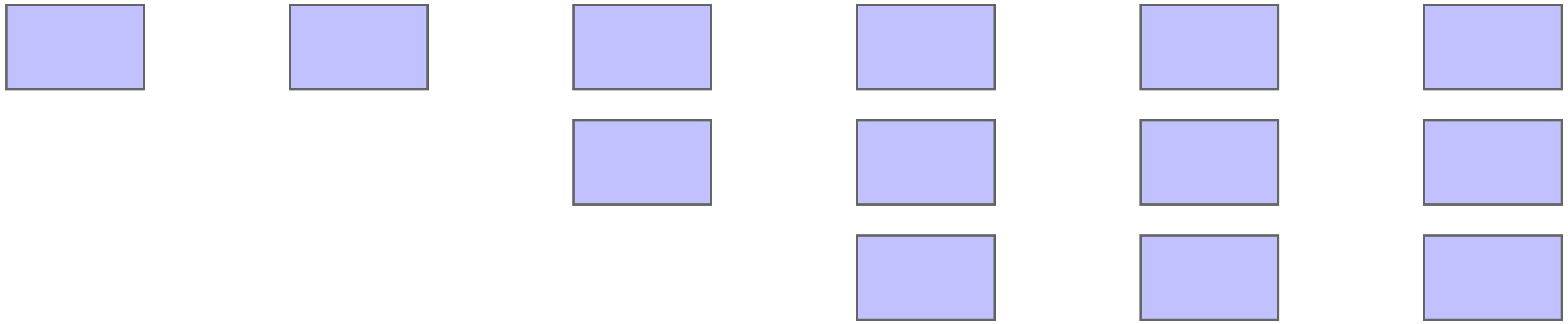
- Our Action-Synchronous algorithm has a slower runtime than CKY!
- However, it also becomes straightforward to prune using beam search.
- So we can achieve a linear runtime in the end.



Now our runtime is $O(n)$.

But this $O(n)$ is hiding a constant.

ϵ



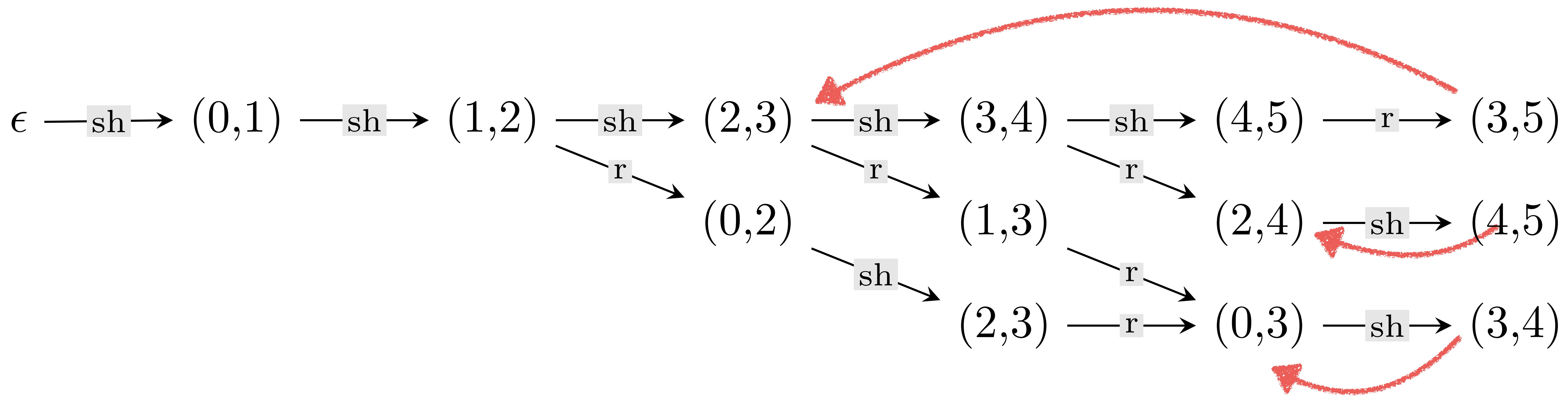
But this $O(n)$ is hiding a constant.

$O(b)$ left pointers per state

$O(nb^2)$ runtime

Cube Pruning

- We can apply cube pruning to make $O(nb \log b)$

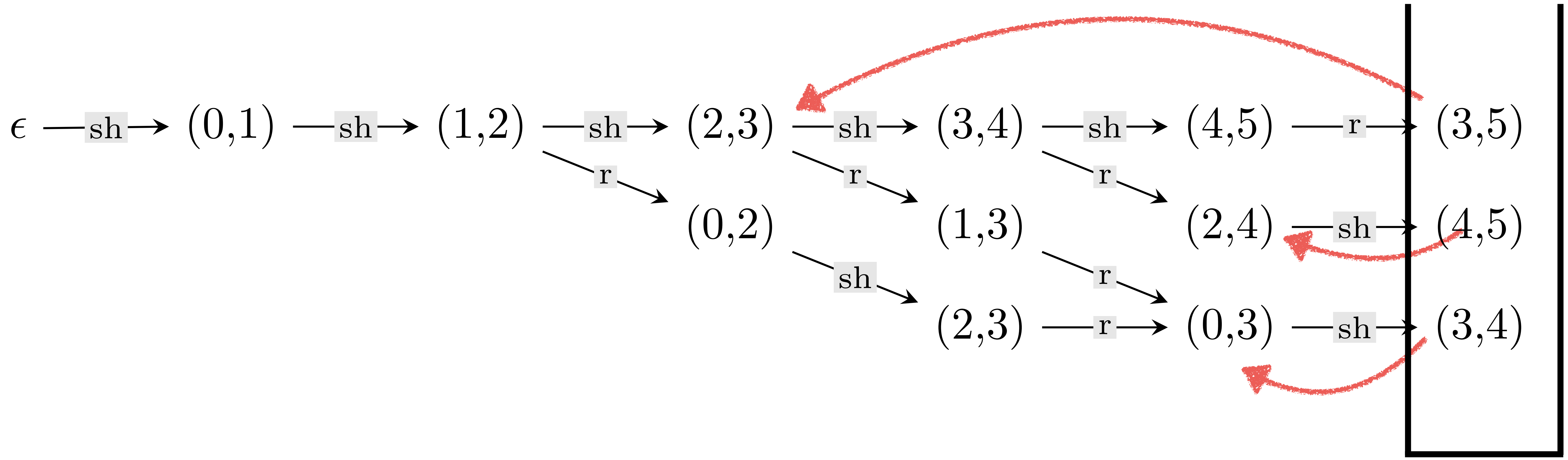


Chiang 2007

Huang+Chiang 2007

Cube Pruning

- We can apply cube pruning to make $O(nb \log b)$



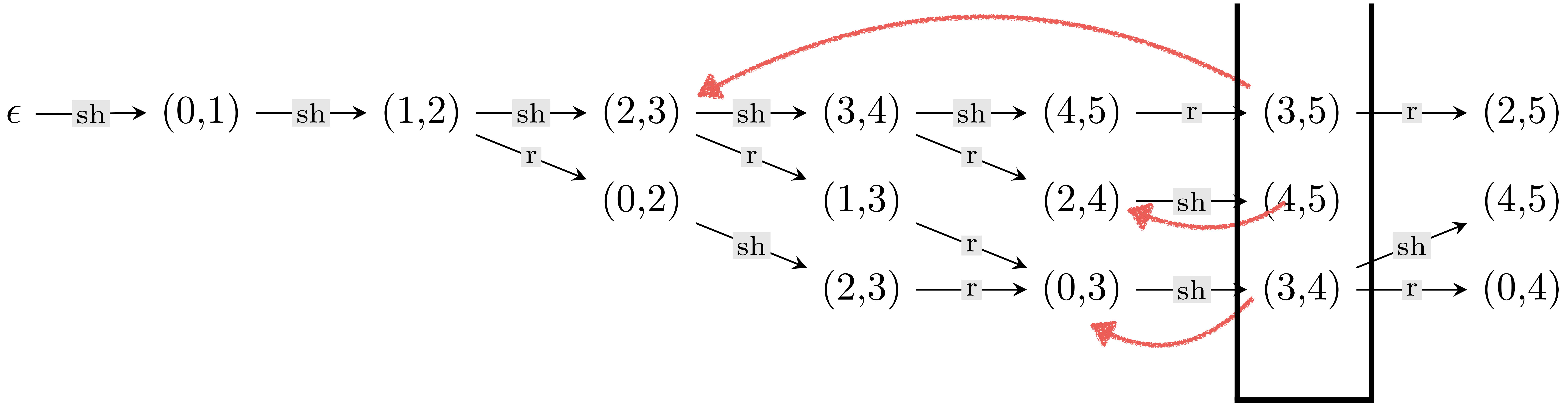
- By pushing all states and their left pointers into a heap

Chiang 2007

Huang+Chiang 2007

Cube Pruning

- We can apply cube pruning to make $O(nb \log b)$



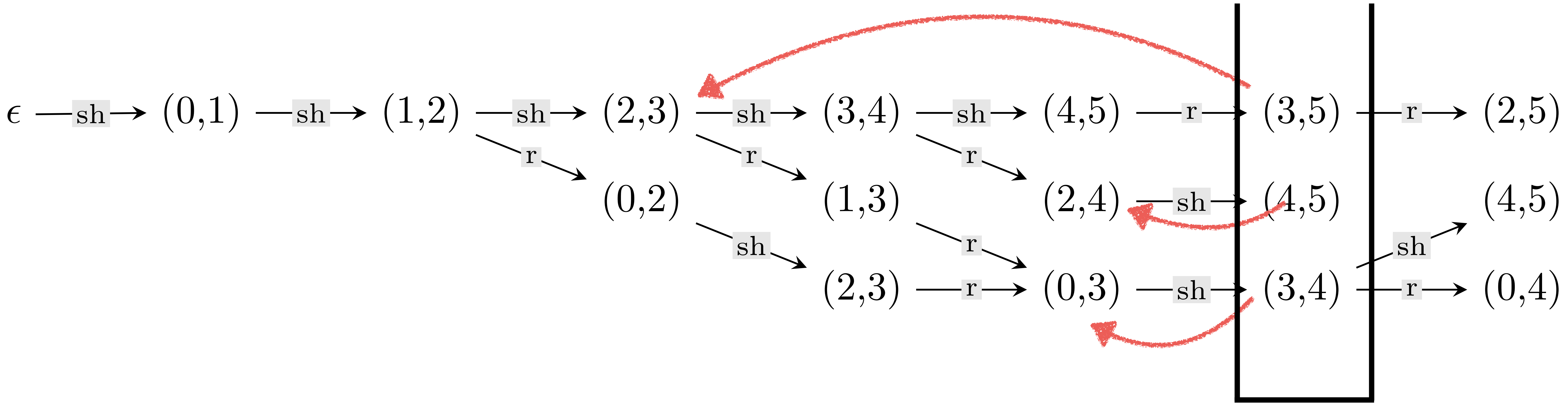
- By pushing all states and their left pointers into a heap
- And popping the top b unique subsequent states

Chiang 2007

Huang+Chiang 2007

Cube Pruning

- We can apply cube pruning to make $O(nb \log b)$

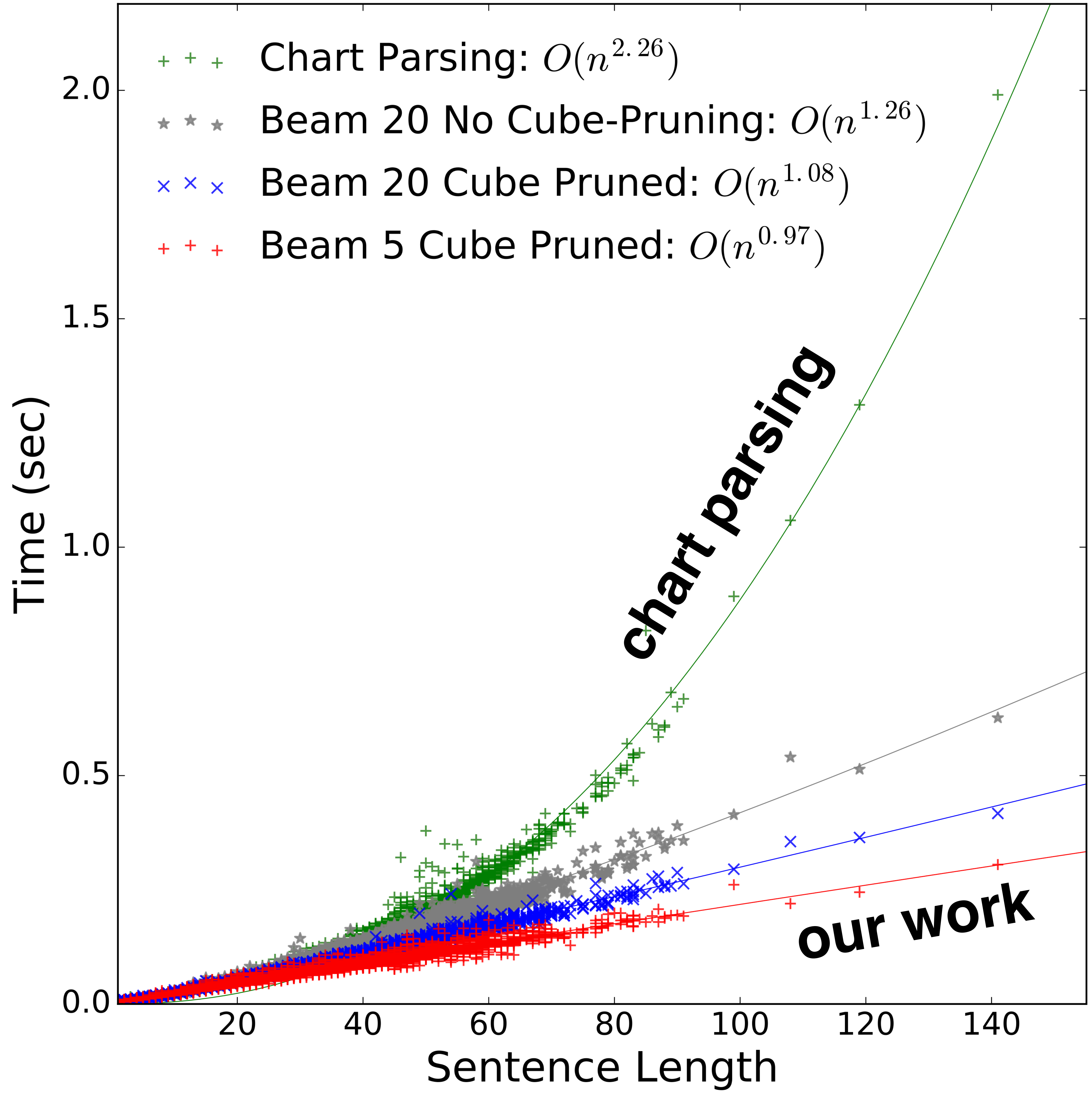


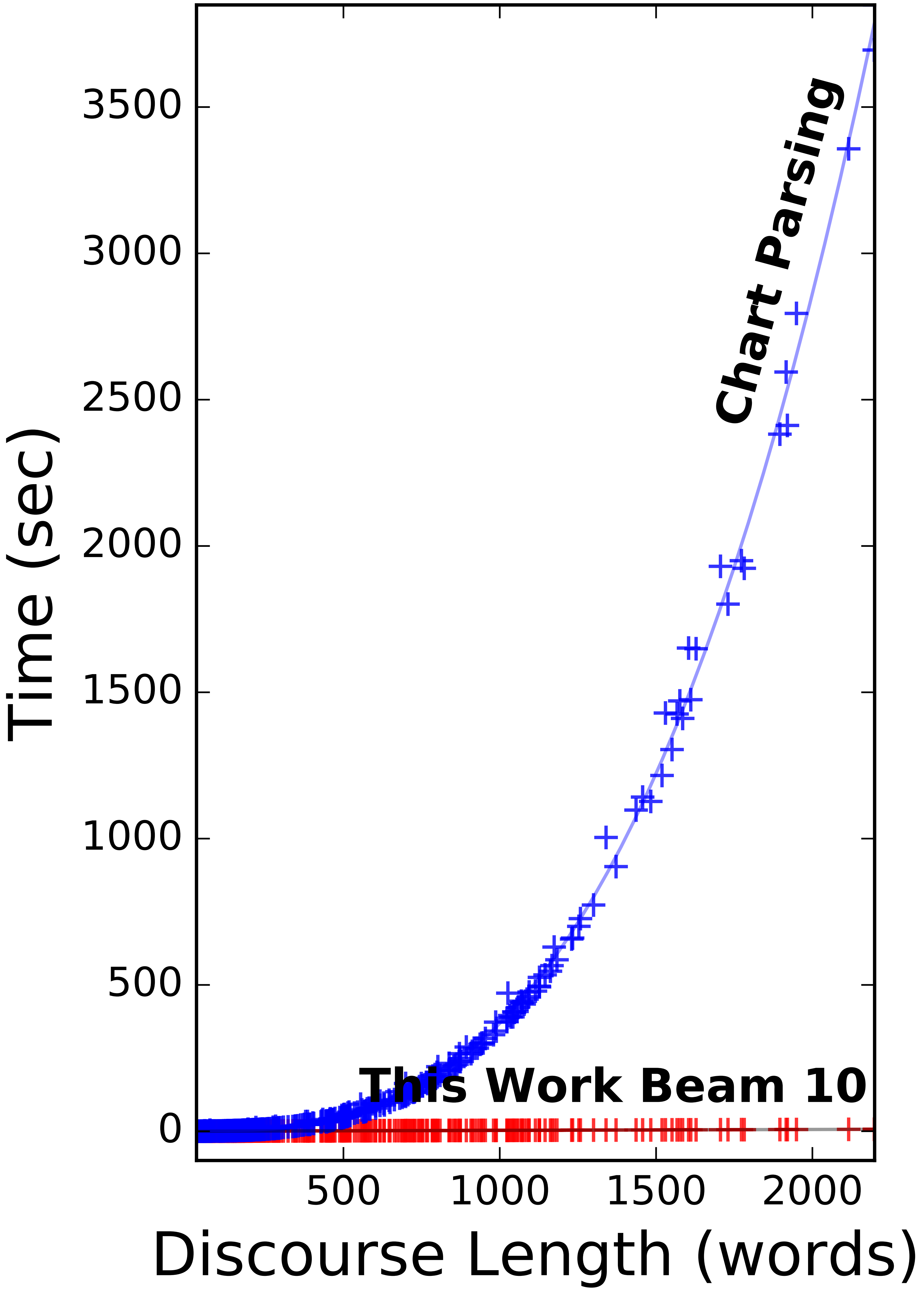
- By pushing all states and their left pointers into a heap
- And popping the top b unique subsequent states
- First time Cube-Pruning has been applied to Incremental Parsing

Chiang 2007

Huang+Chiang 2007

Runtime on PTB and Discourse Treebank





Training

- Structured SVM approach (Taskar et al. 2003; Stern et al. 2017):
  - Goal: Score the gold tree higher than all others by a margin:

$$\forall t, s(t^*) - s(t) \geq \Delta(t, t^*)$$

- Loss Augmented Decoding:
  - During Training: Return the most violated tree (i.e., highest augmented score):

$$\hat{t} = \arg\max_{t} \left(s(t) + \Delta(t, t^*) \right)$$

- Minimize: $\left(s(\hat{t}) + \Delta(\hat{t}, t^*) \right) - s(t^*)$

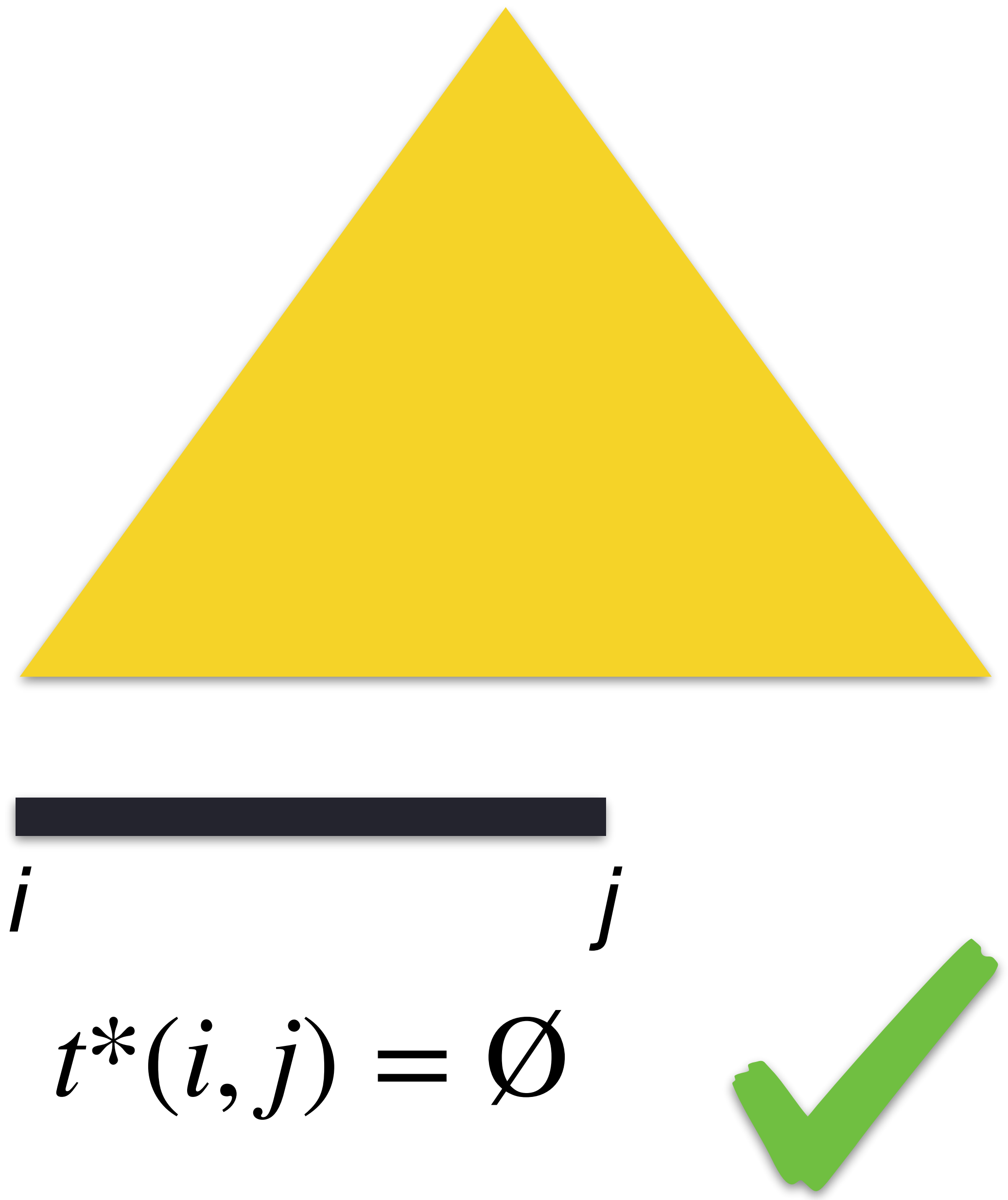
Loss Function

- Counts the incorrectly labeled spans in the tree (Stern et al. 2017)
  - Happens to be decomposable, so can even be used to compare partial trees.

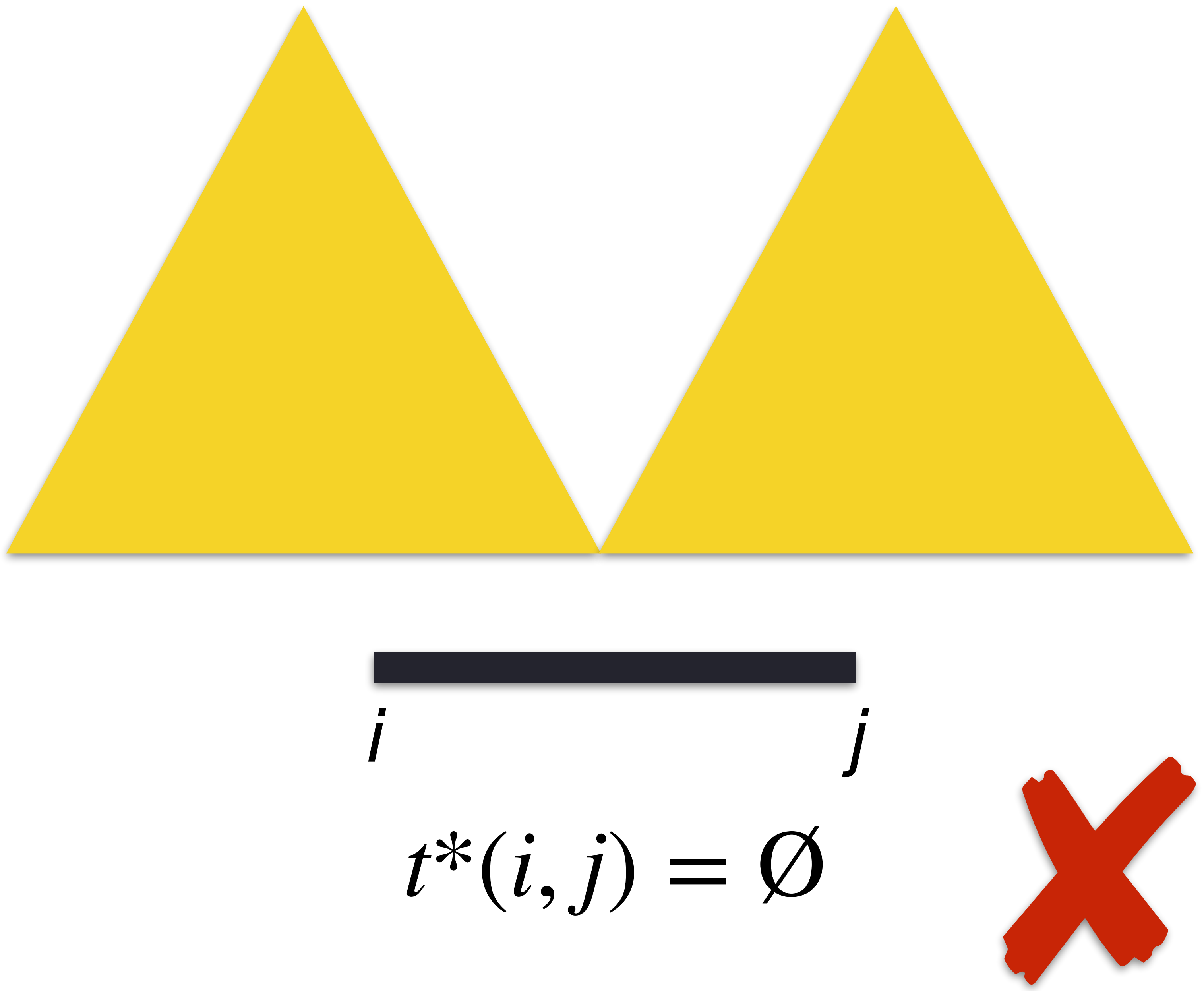
$$\Delta(t, t^*) = \sum_{(i,j,X) \in t} \mathbb{1}\left(X \neq t^*_{(i,j)}\right)$$

Novel Cross-Span Loss

- We observe that the null label ø is used in two different ways:
  - To facilitate ternary and n-ary branching trees.
  - As a default label for incorrect spans that violate other gold spans.



$t^*(i, j) = \varnothing$



$t^*(i, j) = \varnothing$

Novel Cross-Span Loss

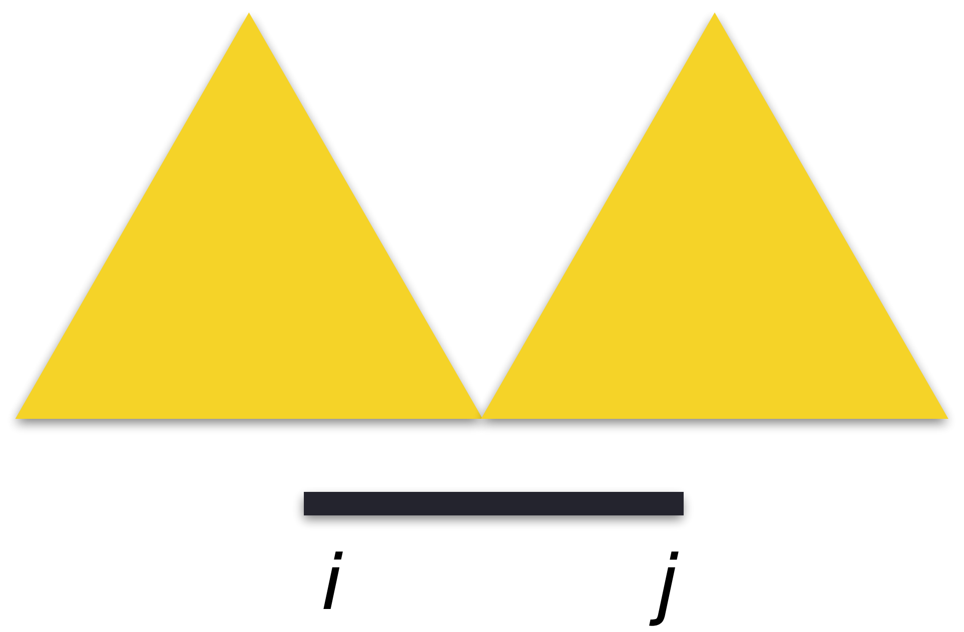
- We modify the loss to account for incorrect spans in the tree.

$$\Delta(t, t^*) = \sum_{(i,j,X) \in t} \mathbb{1}\left(X \neq t^*_{(i,j)}\right)$$

Novel Cross-Span Loss

- We modify the loss to account for incorrect spans in the tree.

$$\mathrm{cross}(i, j, t^*)$$



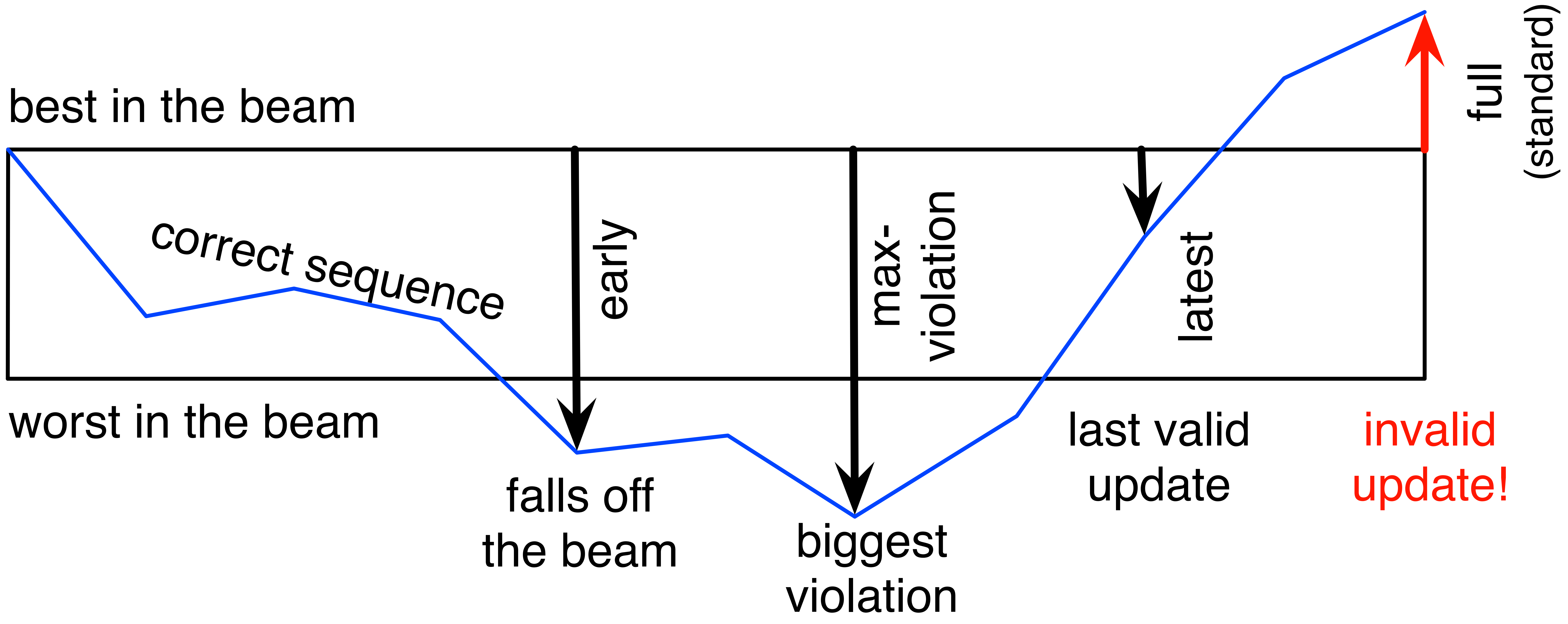
  - Indicates whether (i, j) is crossing a span in the gold tree

$$\Delta(t, t^*) = \sum_{(i,j,X) \in t} \mathbb{1}\Big(X \neq t^*_{(i,j)} \vee \mathrm{cross}(i, j, t^*)\Big)$$

- Still decomposable over spans, so can be used to compare partial trees.

Max-Violation Updates

- Take the largest augmented loss value across all time steps.
- This is the Max-Violation, that we use to train.



Huang et. al. 2012

Comparison with Baseline Chart Parser

Model	Note	F1 (PTB test)
Stern et al. (2017a)	Baseline Chart Parser	91.79
	+our cross-span loss	91.81
Our Work	Beam 15	91.84
	Beam 20	**91.97**

Comparison to Other Parsers

PTB only, Single Model, End-to-End

Model	Note	F1
Durett + Klein 2015		91.1
Cross + Huang 2016	Original Span Parser	91.3
Liu + Zhang 2016		91.7
Dyer et al. 2016	Discriminative	91.7
Stern et al. 2017a	Baseline Chart Parser	91.79
Stern et al. 2017c	Separate Decoding	92.56
Our Work	Beam 20	91.97

Reranking, Ensemble, Extra Data

Model	Note	F1
Vinyals et al. 2015	Ensemble	90.5
Dyer et al. 2016	Generative Reranking	93.3
Choe + Charniak 2016	Reranking	93.8
Fried et al. 2017	Ensemble Reranking	94.25

Conclusions

- Linear-Time, Span-Based Constituency Parsing with Dynamic Programming
- Cube-Pruning to speedup Incremental Parsing with Dynamic Programming
- Cross-Span Loss extension for improving Loss-Augmented Decoding
- Result: Faster and more accurate than cubic-time Chart Parsing
  - 2nd highest accuracy for single-model end-to-end systems trained on PTB only
    - Stern et al. 2017c is more accurate, but with separate decoding, and is much slower
  - After this ACL, definitely no longer true. (e.g. Joshi et al. 2018, Kitaev+Klein 2018)
    - But both are Span-Based Parsers and can be linearized in the same way!

$$O(2^n) \rightarrow O(n^3) \rightarrow O(n^4) \rightsquigarrow O(nb^2) \rightsquigarrow O(nb\log b)$$

Acknowledgements

- Dezhong Deng for his theorem for predecessor states.
  - And his mathematical proofreading of the training sections.
- Mitchell Stern for releasing his code and his suggestions.

Thank you! Questions?