Linear Time Constituency Parsing with RNNs and Dynamic Programming

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

[1] Oregon State University

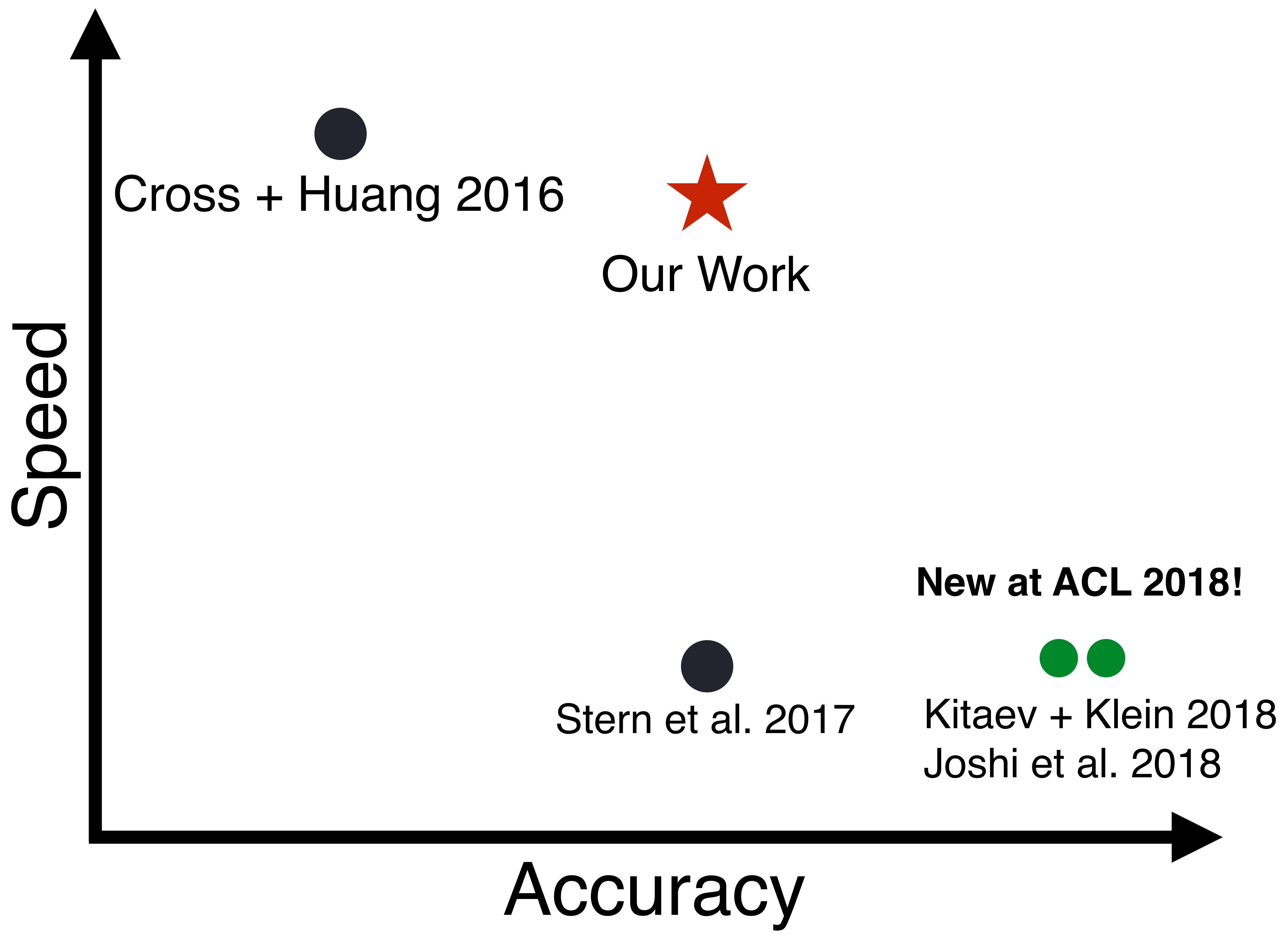
[2] Baidu Research Silicon Valley AI Lab

Oregon State University

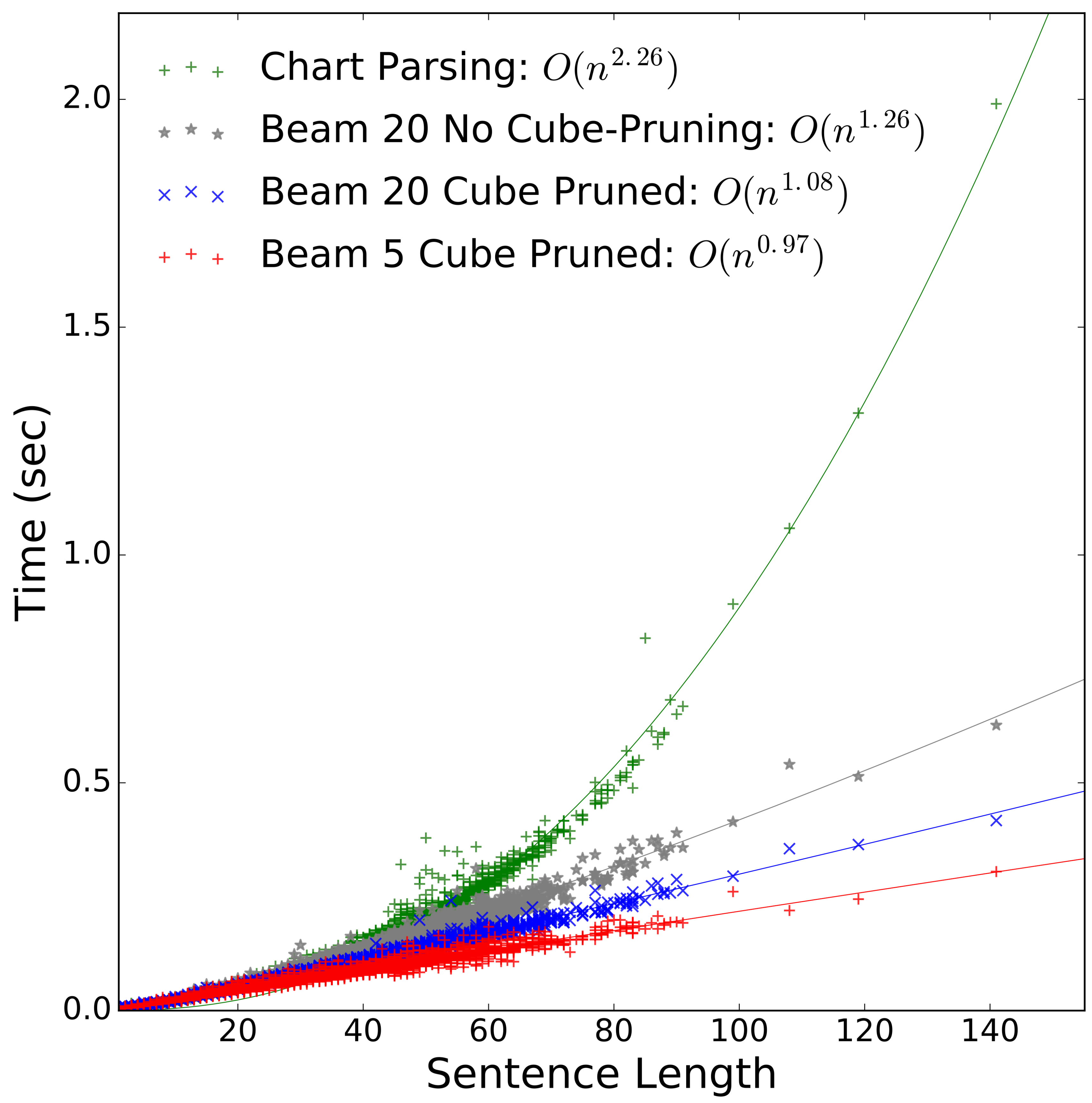
Baidu Research

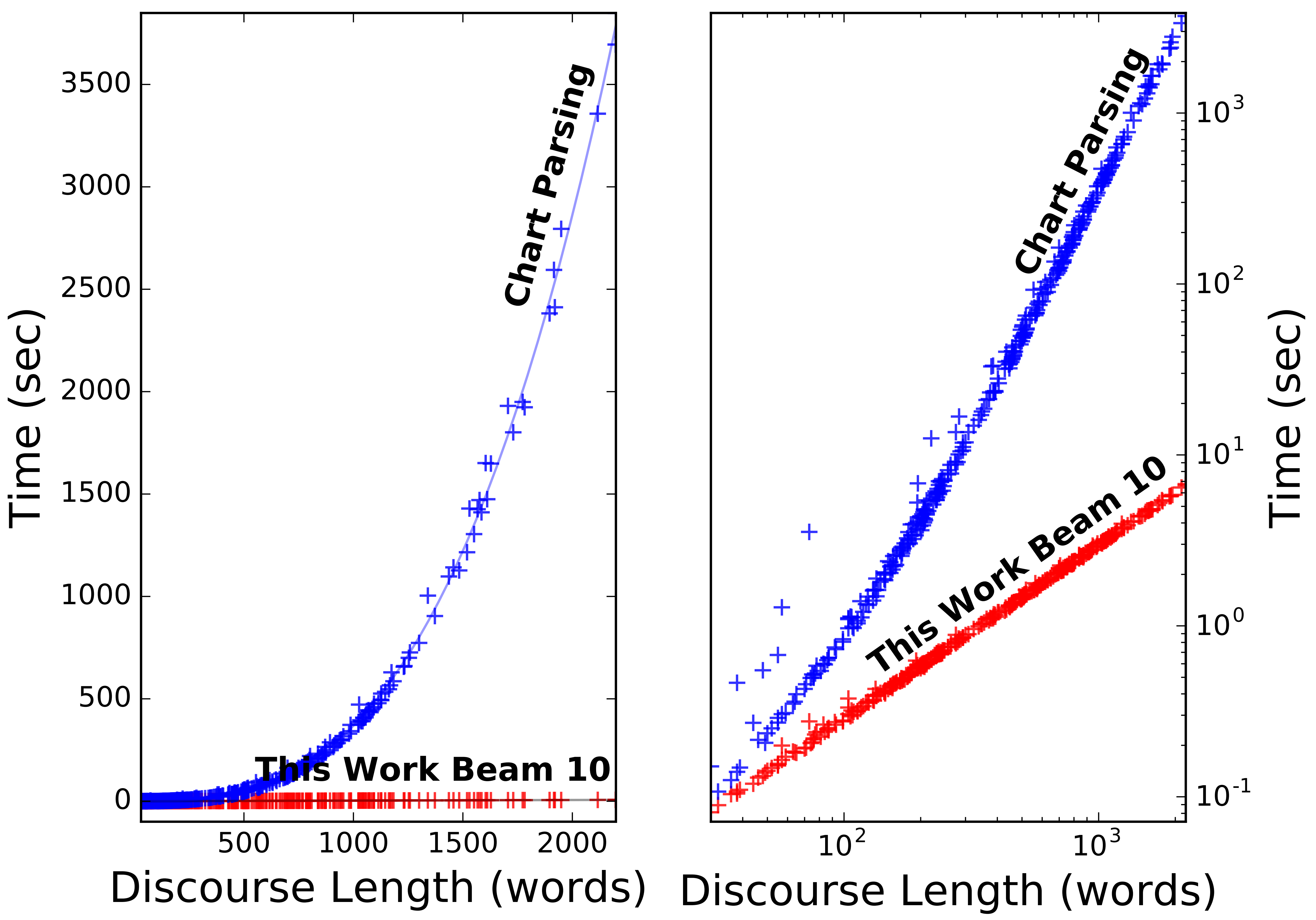
A Brief History of Span Parsing

- Cross+Huang 2016 Introduced Span Parsing
  - But with greedy decoding.
- Stern et al. 2017 had Span Parsing with Global Search
  - But was too slow: $O(n^3)$
- Can we get something in between?
  - Something that is both fast and accurate?



Both Fast and Accurate!





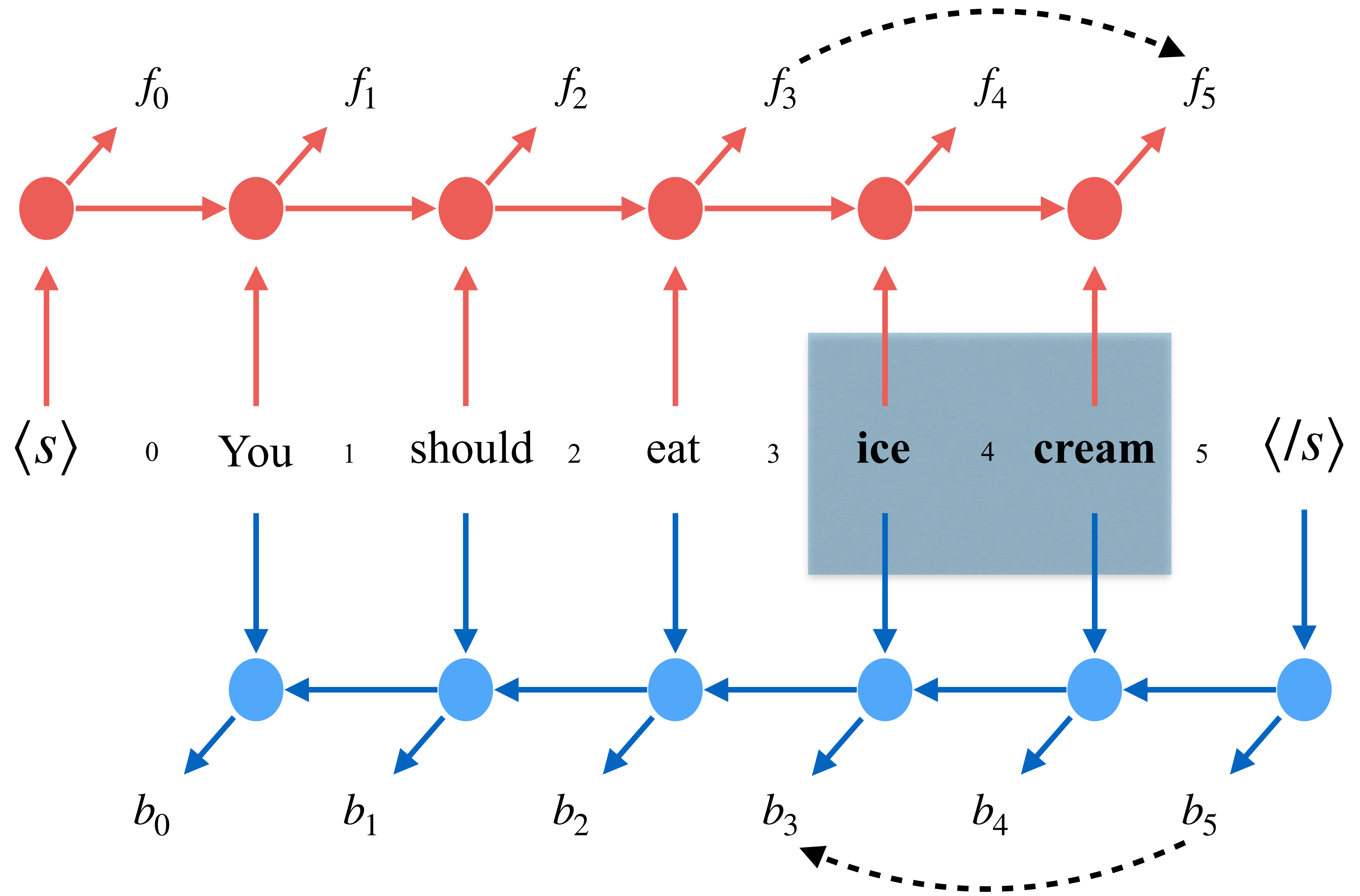
Baseline Chart Parser (Stern et al. 2017a)	91.79
Our Linear Time Parser	**91.97**

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.
- An improved loss function for Loss-Augmented Decoding.

Span Parsing

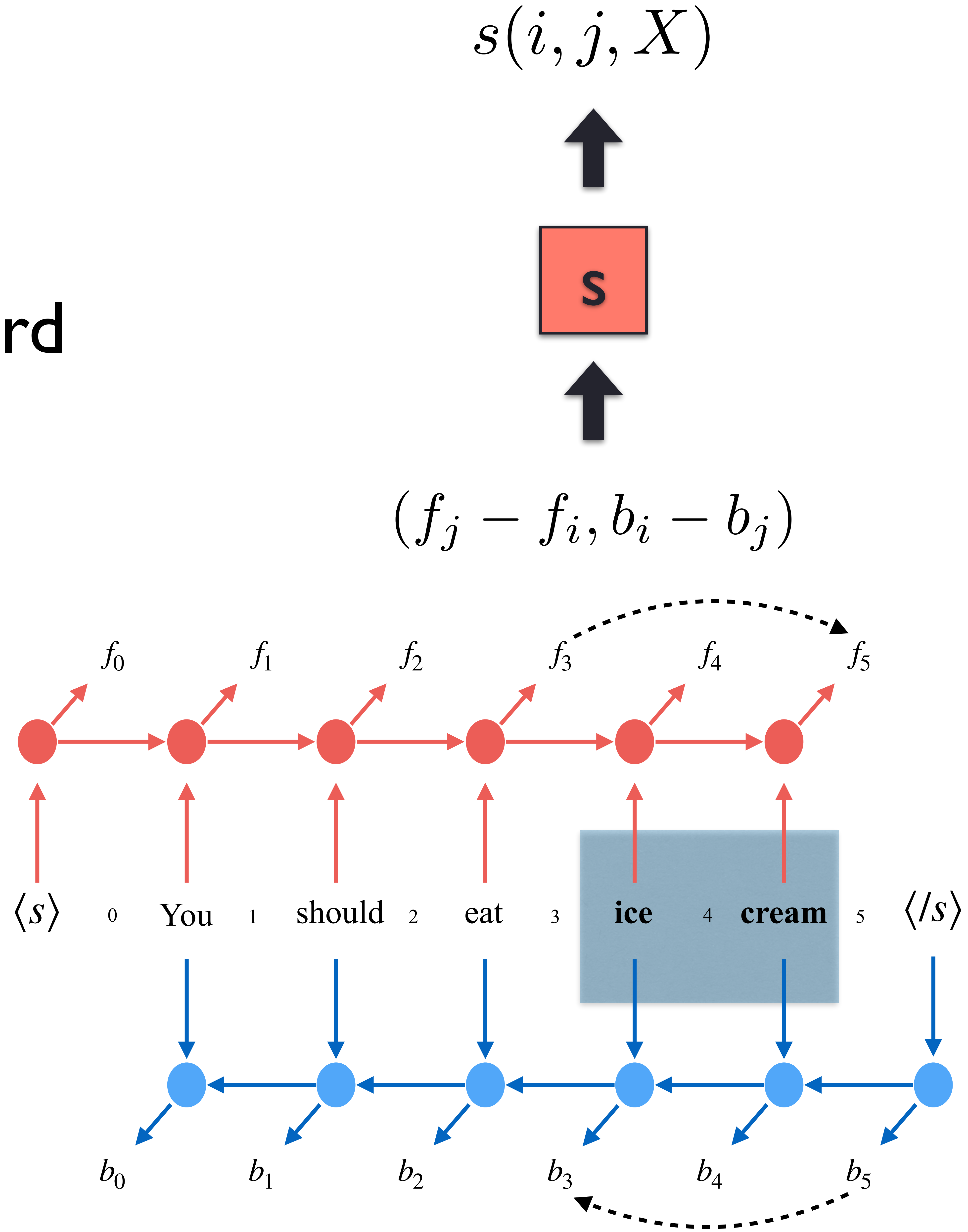
- Span differences are taken from an encoder (in our case: a bi-LSTM)



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

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.

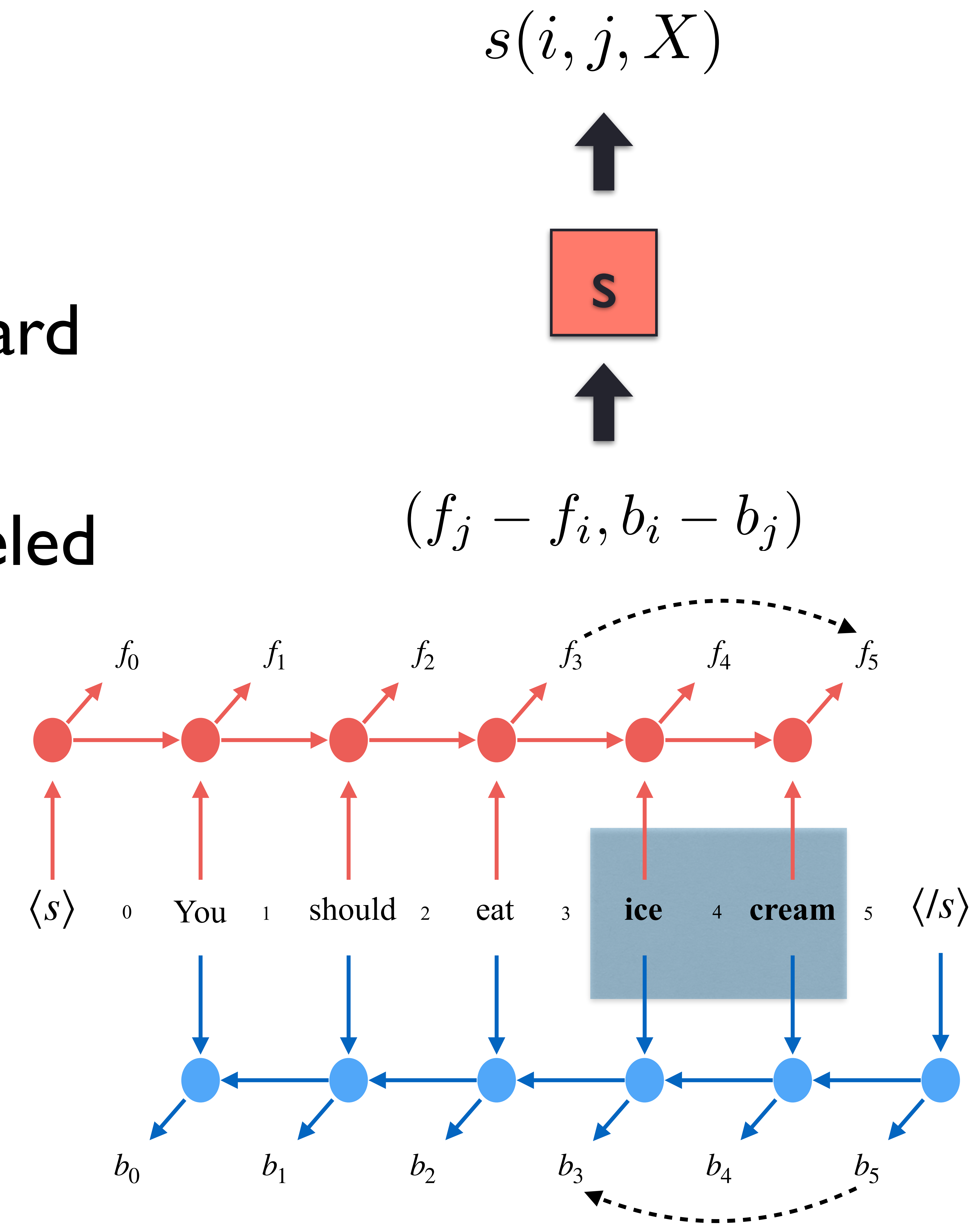


Cross + Huang 2016 Stern et al. 2017 Wang + Chang 2016

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

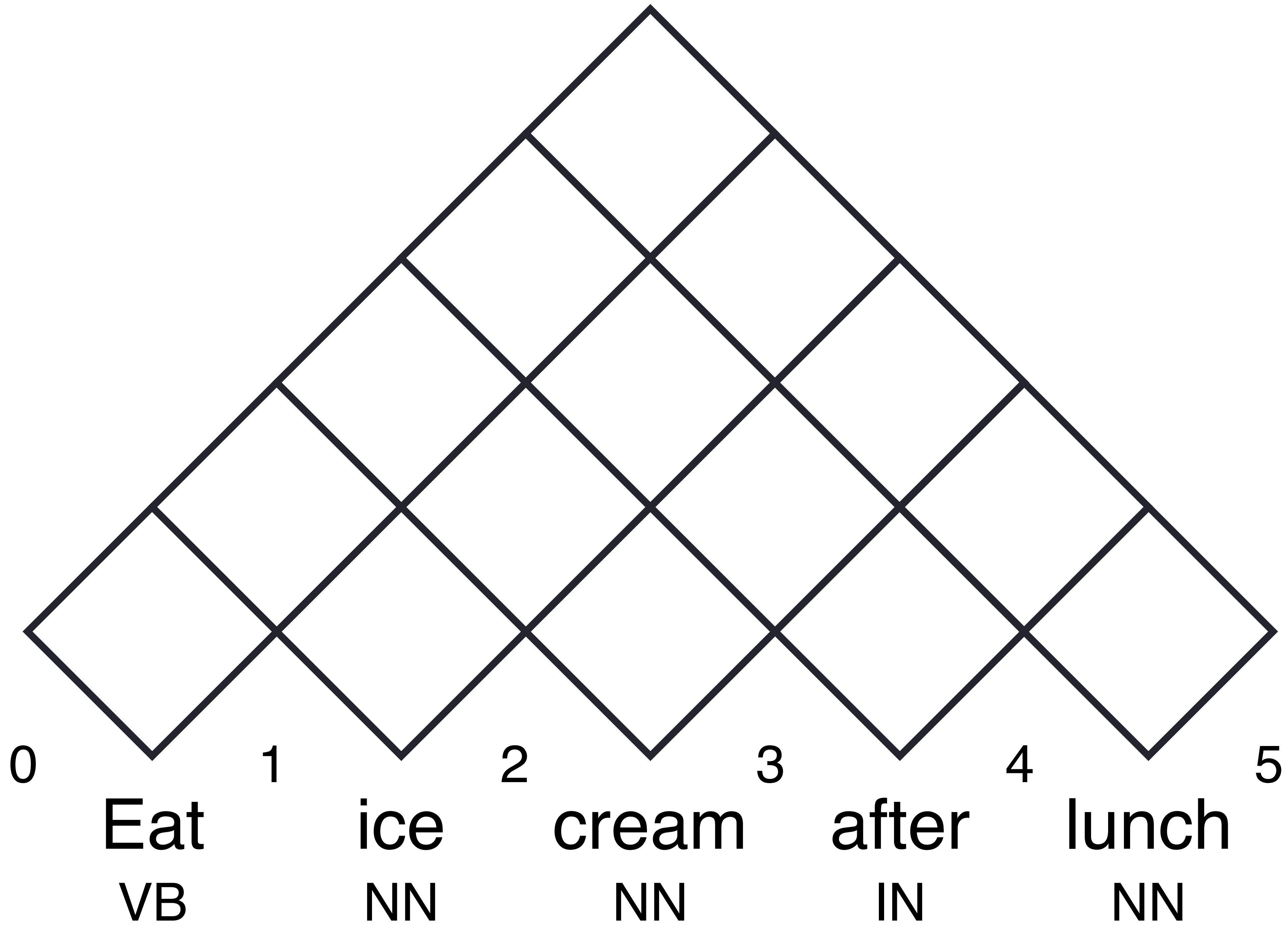
$$s_{tree}(t) = \sum_{(i,j,X)\in t} s(i,j,X)$$



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

Incremental Span Parsing Example

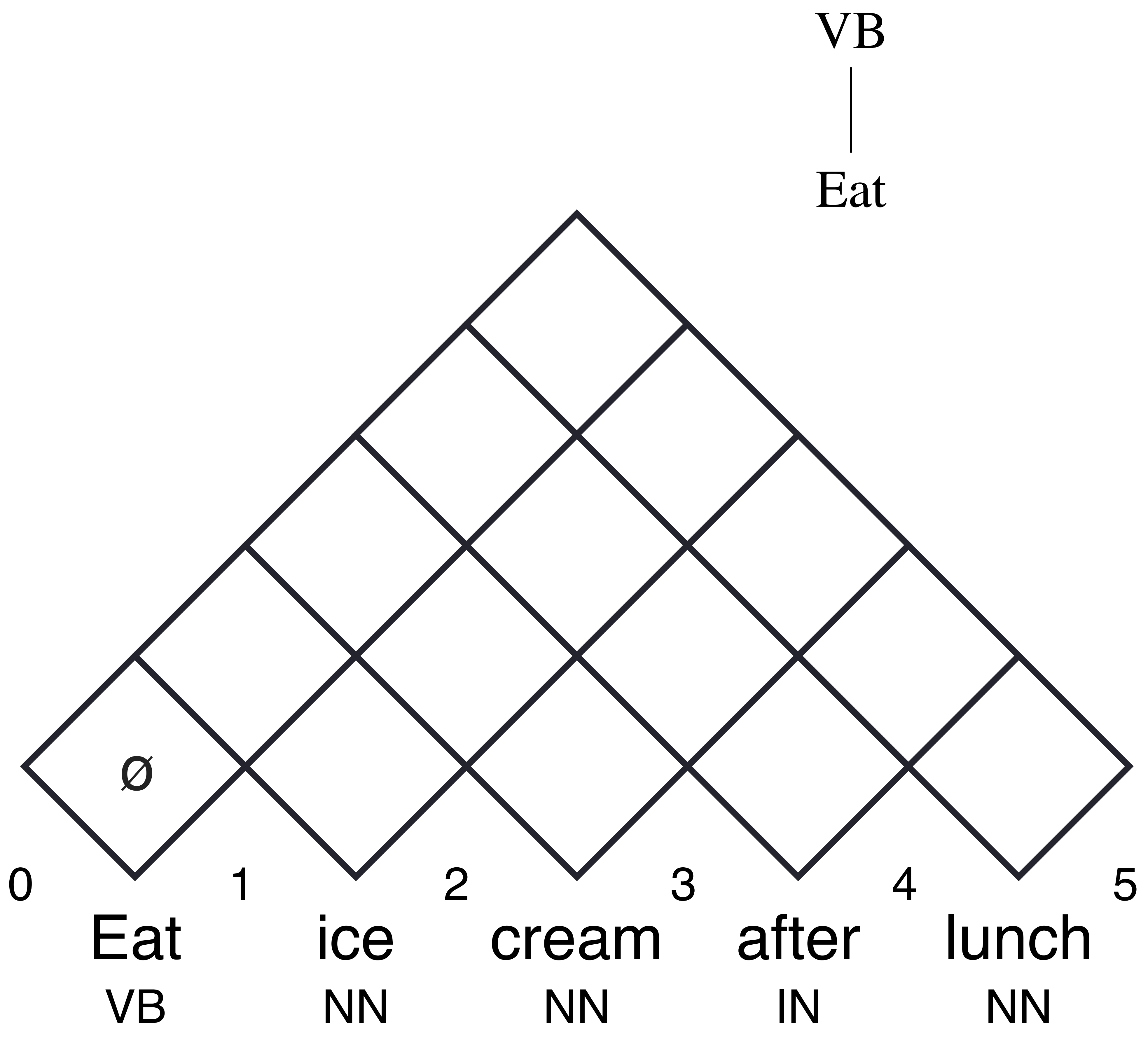
	Action	Label	Stack



Cross + Huang 2016

Incremental Span Parsing Example

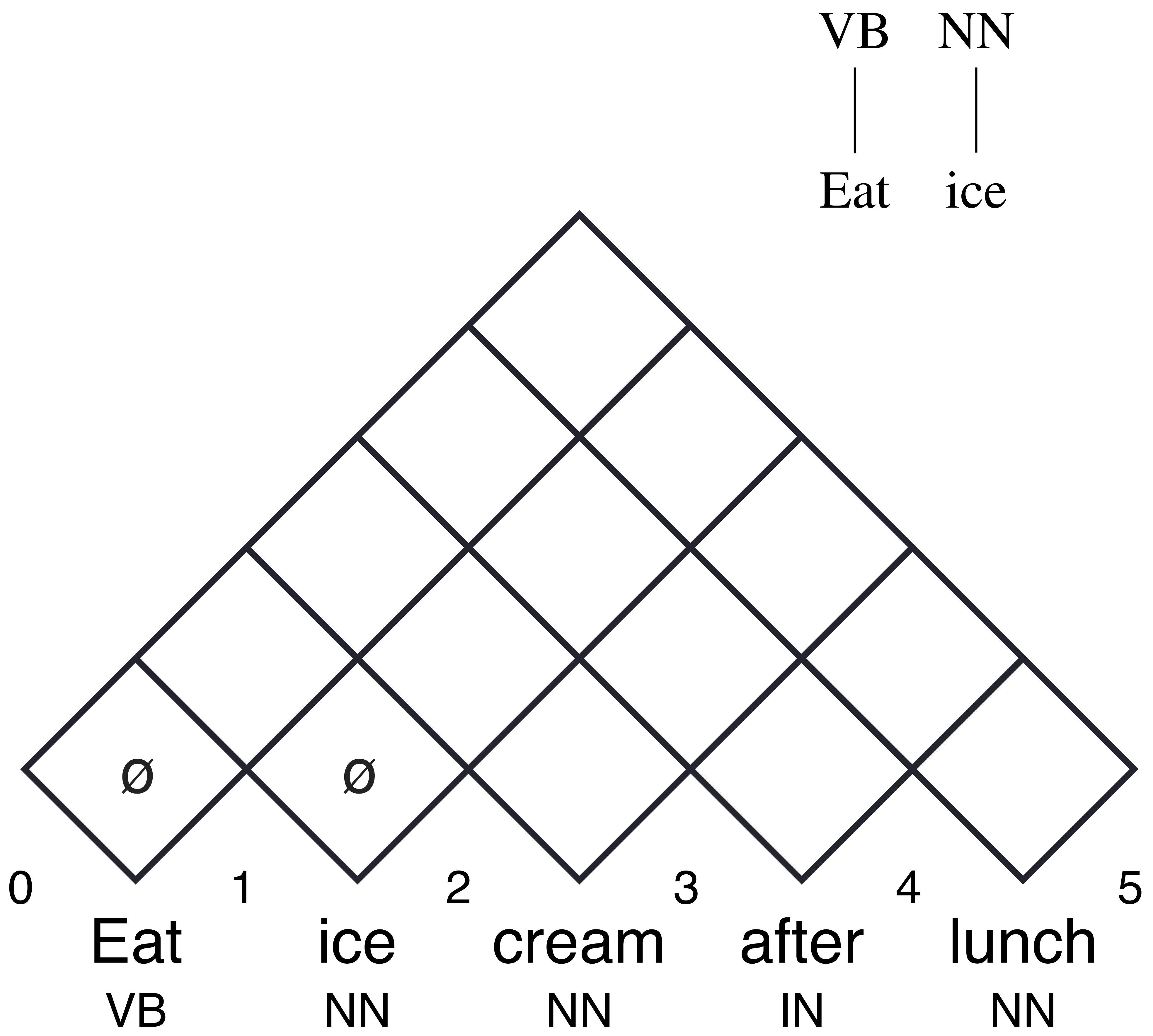
	Action	Label	Stack
1	Shift	ø	(0, 1, ø)



Cross + Huang 2016

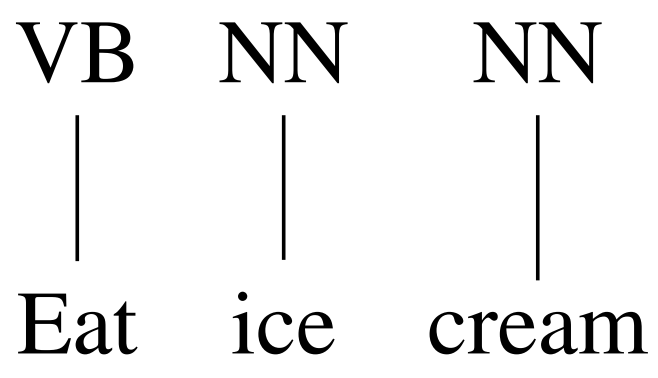
Incremental Span Parsing Example

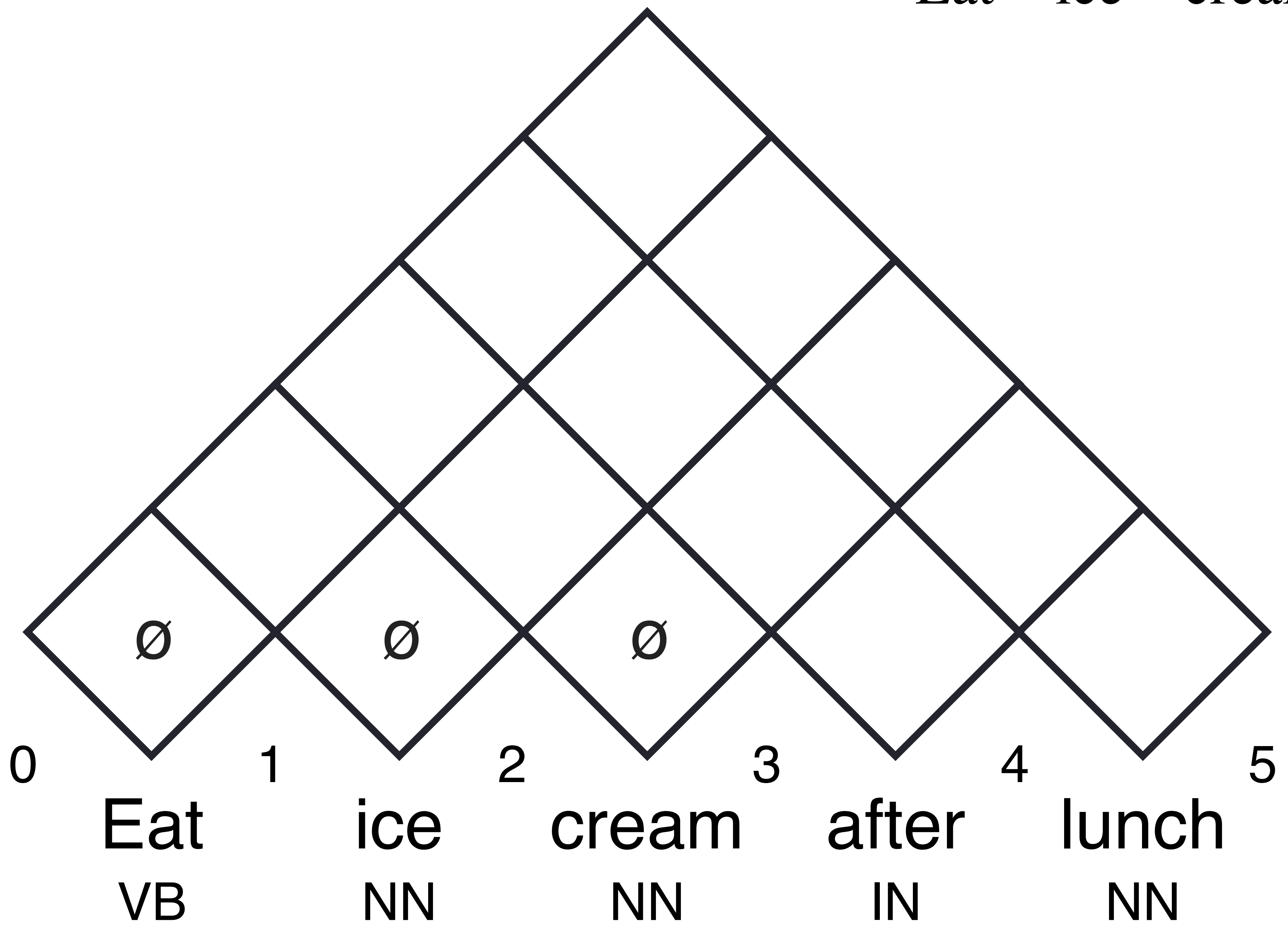
	Action	Label	Stack
1	Shift	ø	(0, 1, ø)
2	Shift	ø	(0, 1, ø) (1, 2, ø)



Cross + Huang 2016

Incremental Span Parsing Example

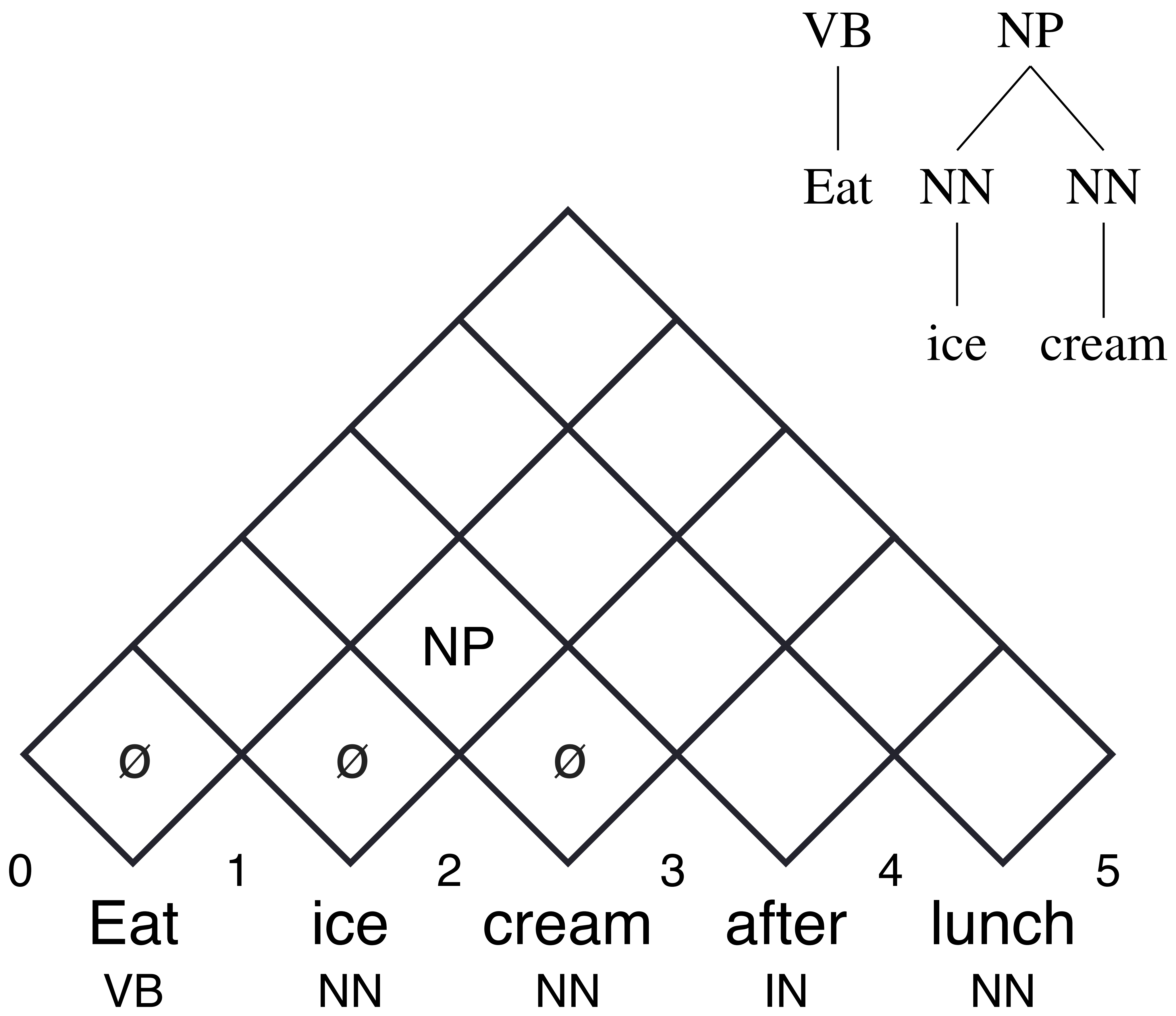




	Action	Label	Stack
1	Shift	ø	(0, 1, ø)
2	Shift	ø	(0, 1, ø) (1, 2, ø)
3	Shift	ø	(0, 1, ø) (1, 2, ø) (2, 3, ø)

Cross + Huang 2016

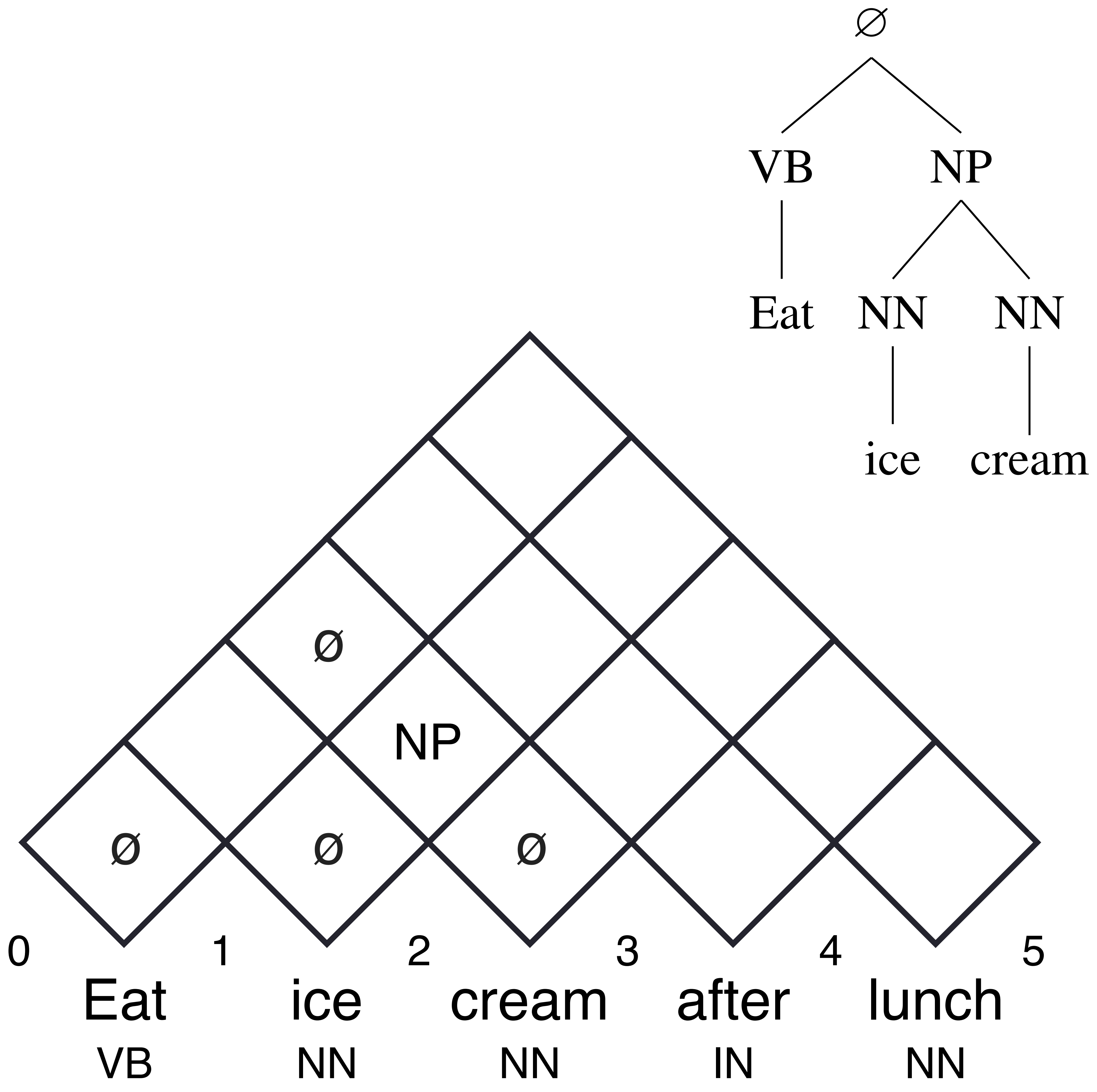
Incremental Span Parsing Example



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

Cross + Huang 2016

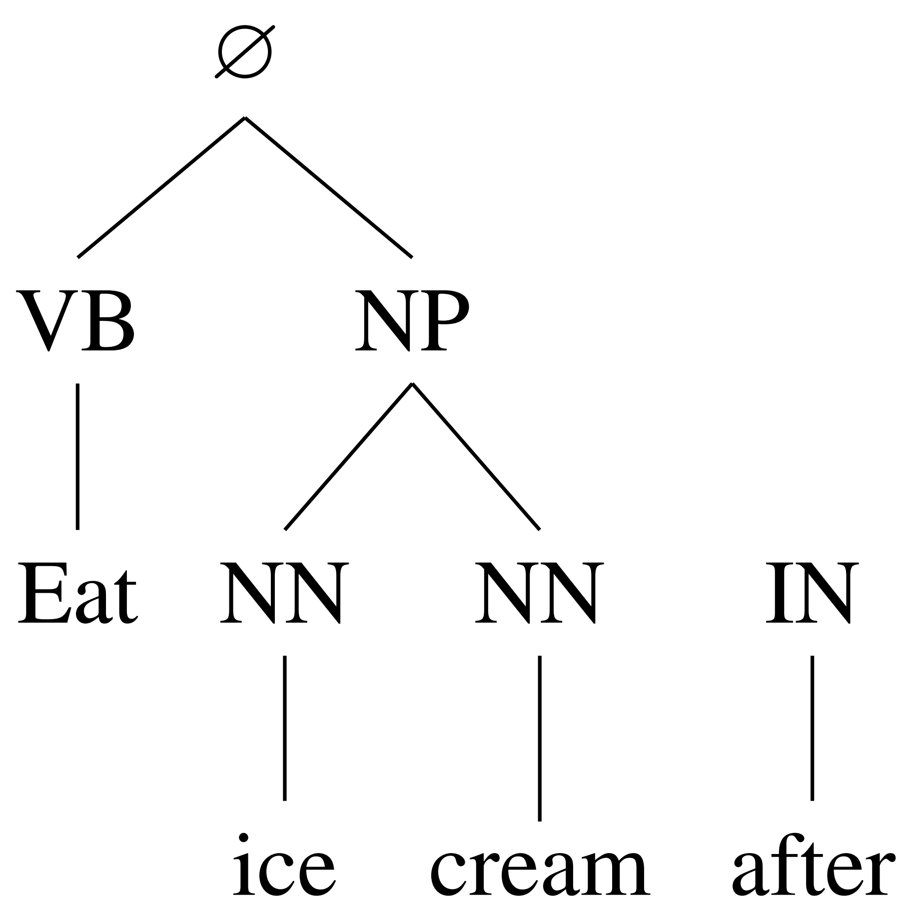
Incremental Span Parsing Example

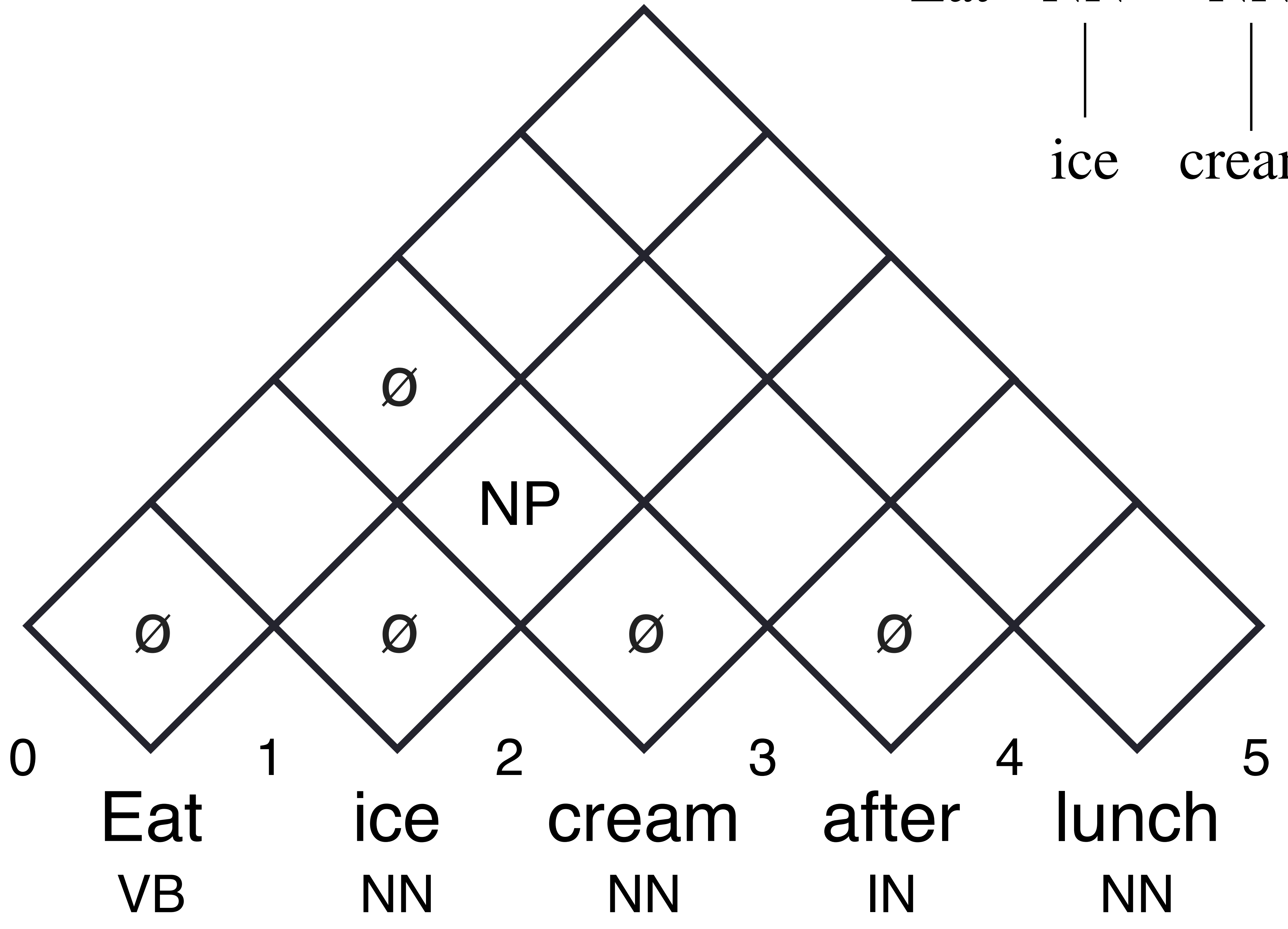


	Action	Label	Stack
1	Shift	ø	(0, 1, ø)
2	Shift	ø	(0, 1, ø) (1, 2, ø)
3	Shift	ø	(0, 1, ø) (1, 2, ø) (2, 3, ø)
4	Reduce	NP	(0, 1, ø) (1, 3, NP)
5	Reduce	ø	(0, 3, ø)

Cross + Huang 2016

Incremental Span Parsing Example

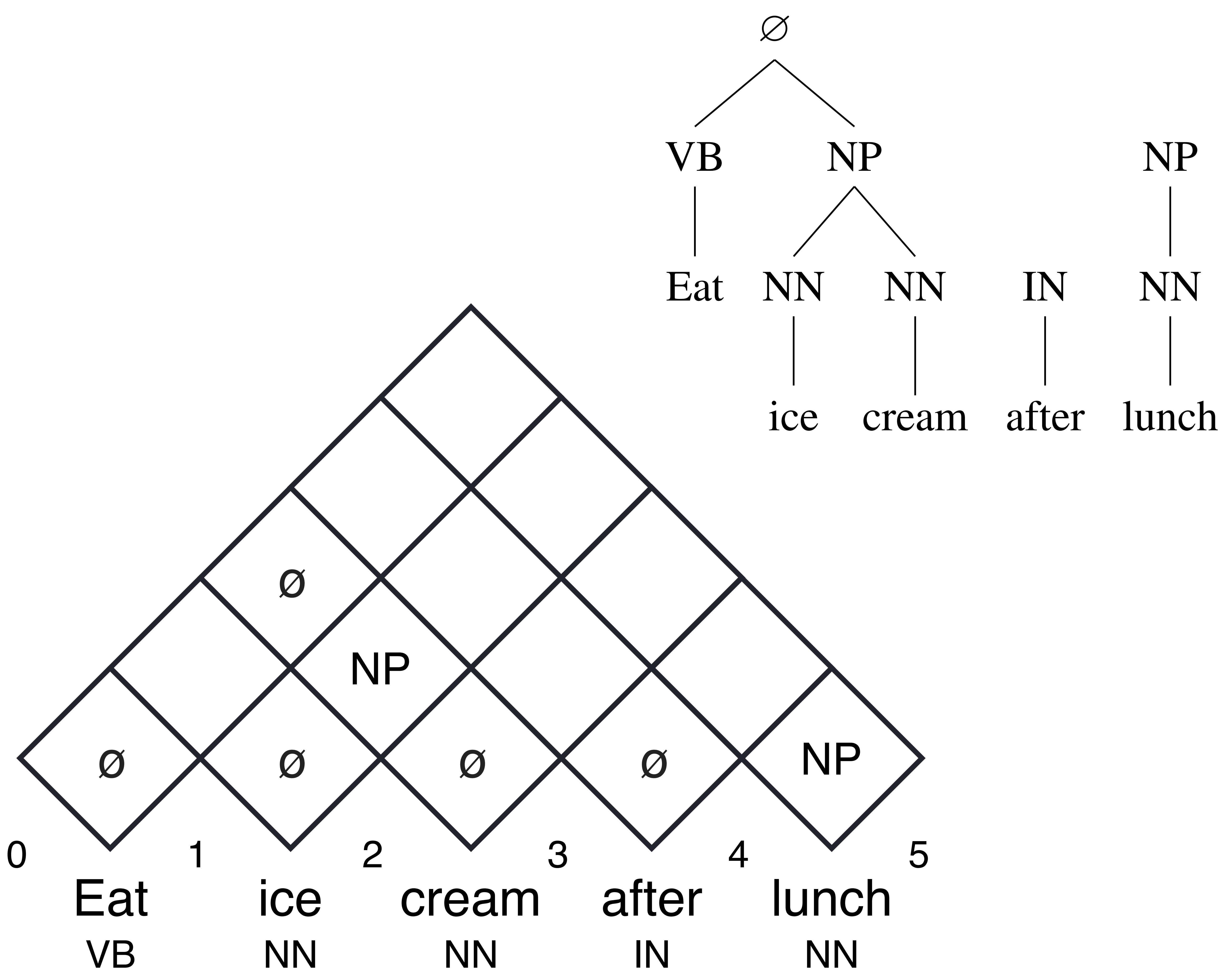




	Action	Label	Stack
1	Shift	ø	(0, 1, ø)
2	Shift	ø	(0, 1, ø) (1, 2, ø)
3	Shift	ø	(0, 1, ø) (1, 2, ø) (2, 3, ø)
4	Reduce	NP	(0, 1, ø) (1, 3, NP)
5	Reduce	ø	(0, 3, ø)
6	Shift	ø	(0, 3, ø) (3, 4, ø)

Cross + Huang 2016

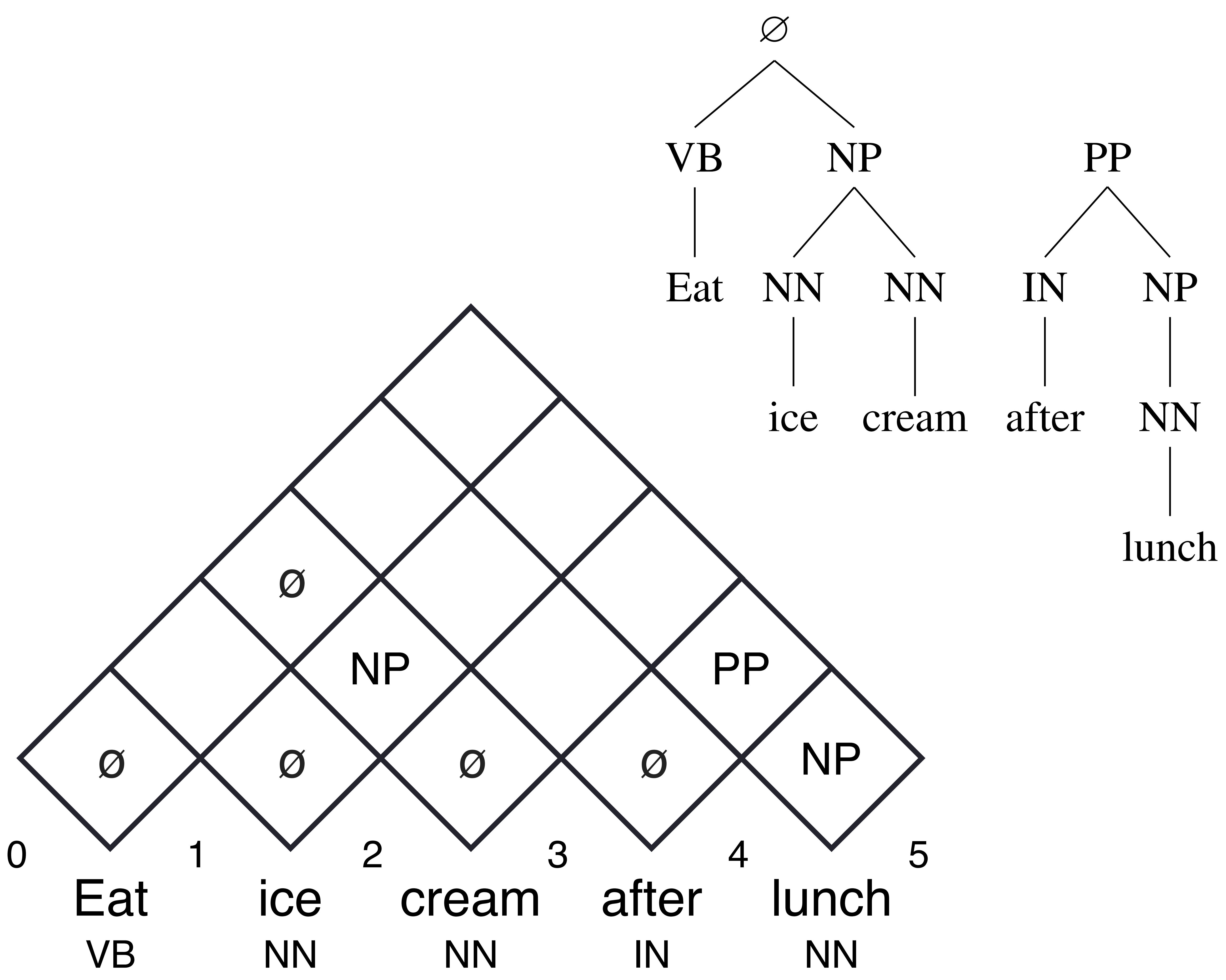
Incremental Span Parsing Example



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

Cross + Huang 2016

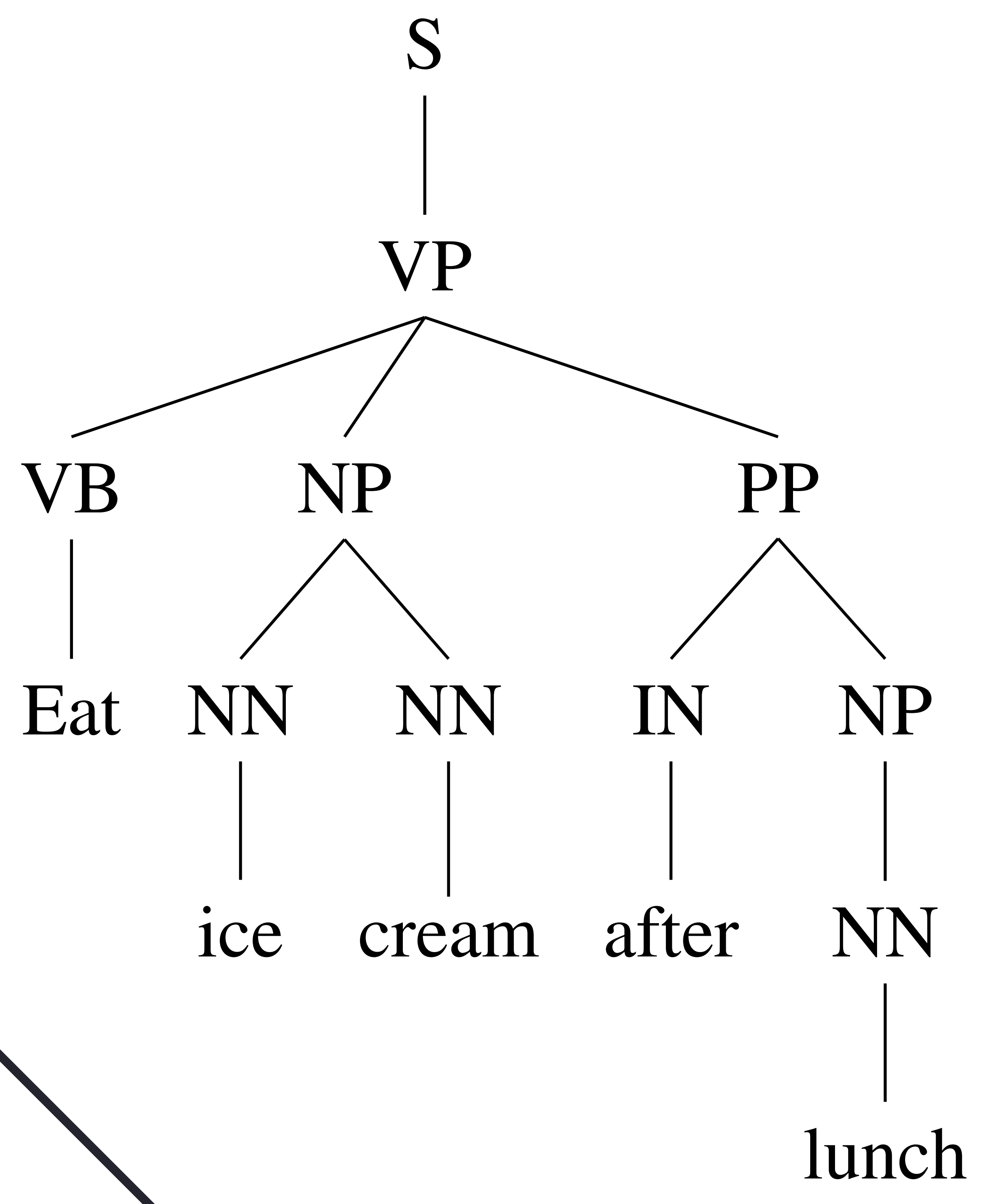
Incremental Span Parsing Example

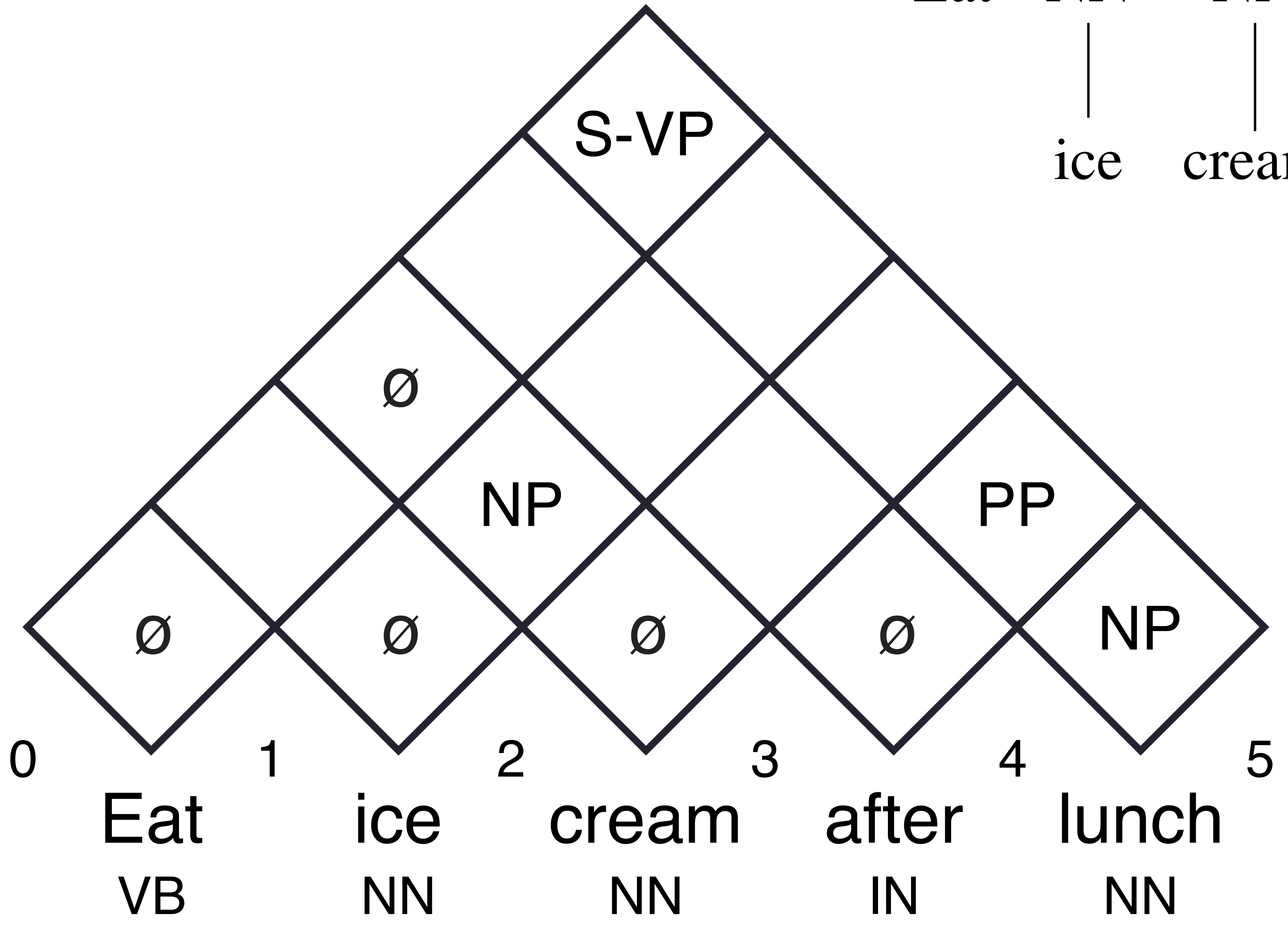


	Action	Label	Stack
1	Shift	ø	(0, 1, ø)
2	Shift	ø	(0, 1, ø) (1, 2, ø)
3	Shift	ø	(0, 1, ø) (1, 2, ø) (2, 3, ø)
4	Reduce	NP	(0, 1, ø) (1, 3, NP)
5	Reduce	ø	(0, 3, ø)
6	Shift	ø	(0, 3, ø) (3, 4, ø)
7	Shift	NP	(0, 3, ø) (3, 4, ø) (4, 5, NP)
8	Reduce	PP	(0, 3, ø) (3, 5, PP)

Cross + Huang 2016

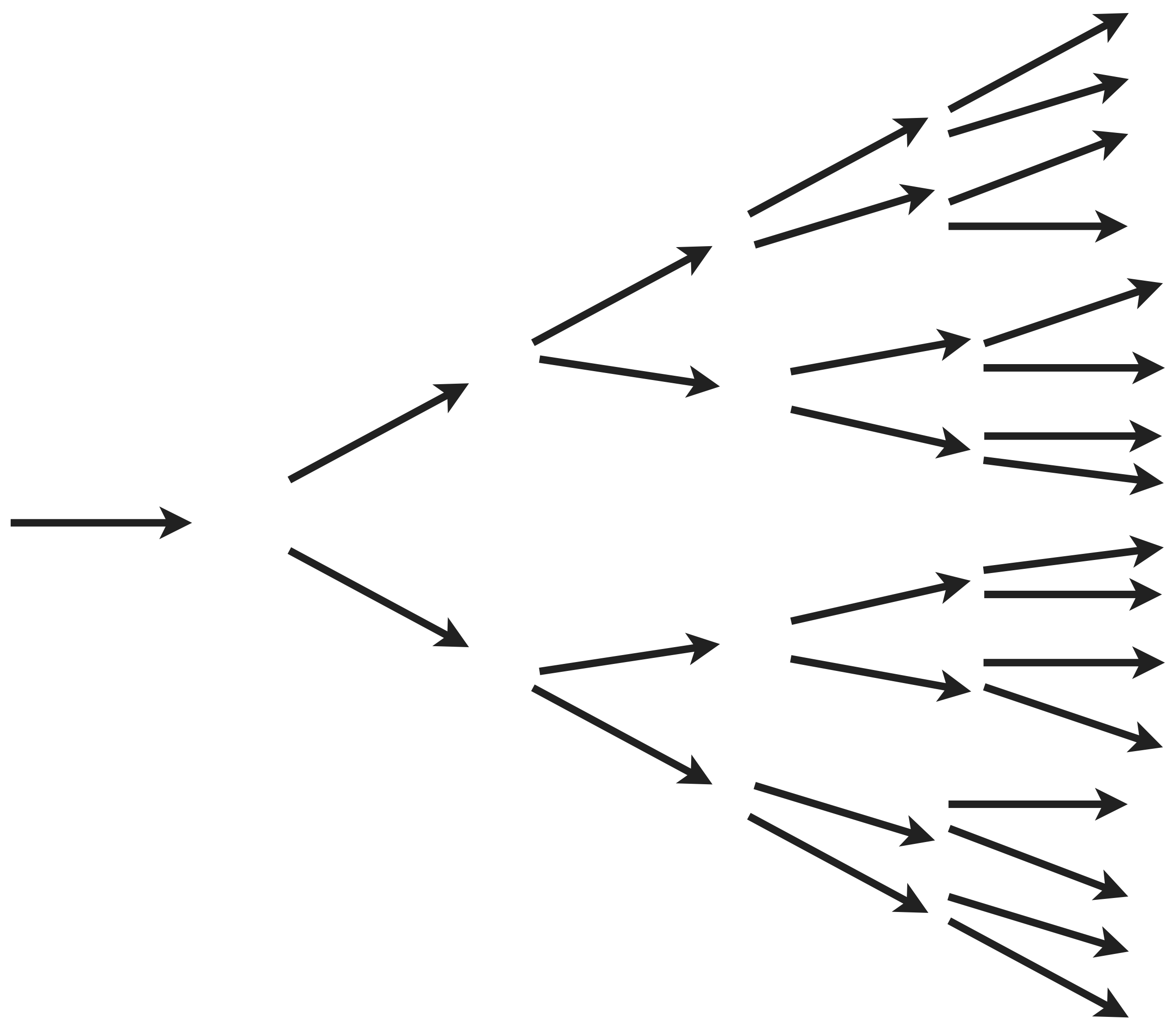
Incremental Span Parsing Example

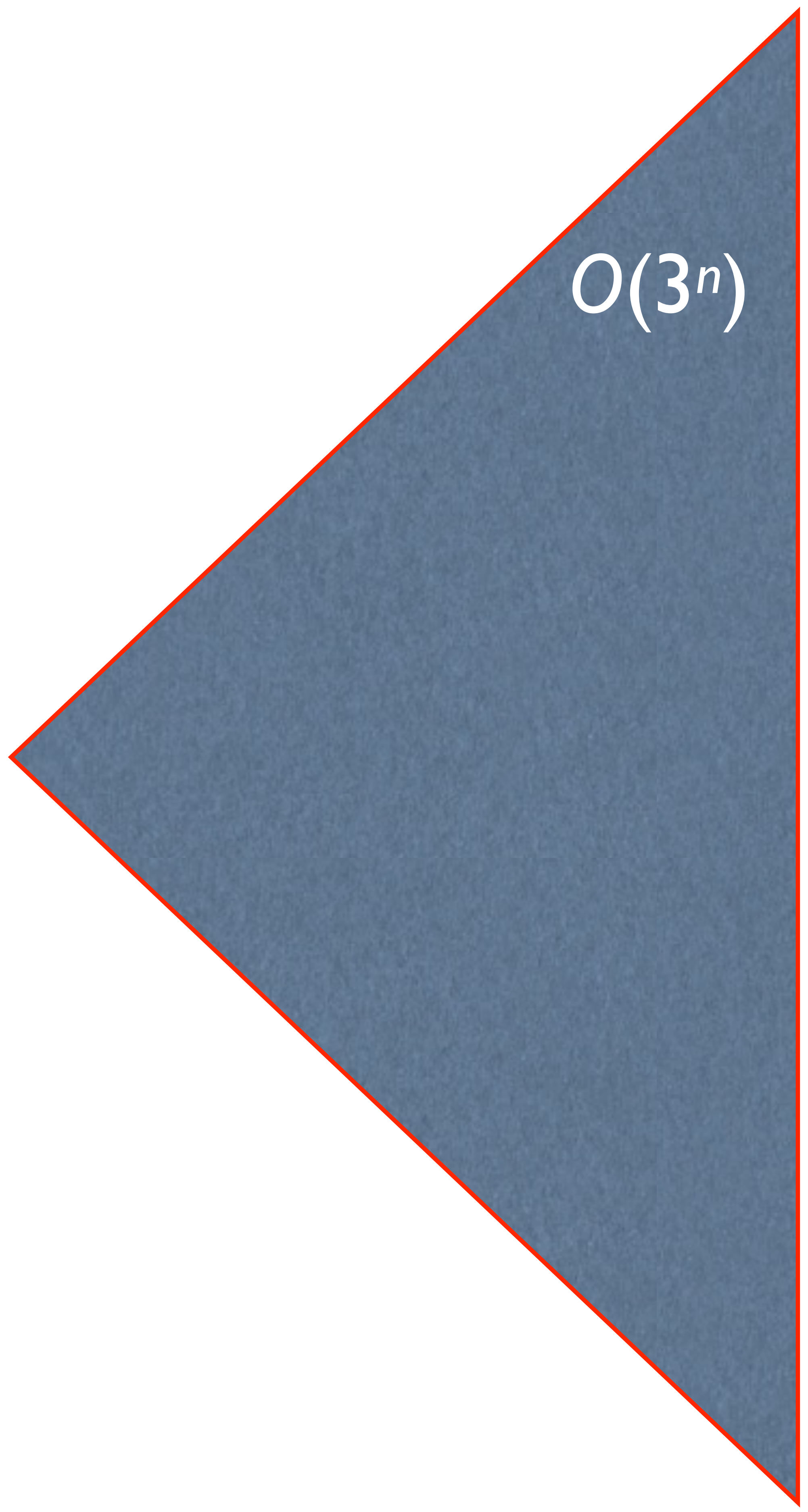




	Action	Label	Stack
1	Shift	ø	(0, 1, ø)
2	Shift	ø	(0, 1, ø) (1, 2, ø)
3	Shift	ø	(0, 1, ø) (1, 2, ø) (2, 3, ø)
4	Reduce	NP	(0, 1, ø) (1, 3, NP)
5	Reduce	ø	(0, 3, ø)
6	Shift	ø	(0, 3, ø) (3, 4, ø)
7	Shift	NP	(0, 3, ø) (3, 4, ø) (4, 5, NP)
8	Reduce	PP	(0, 3, ø) (3, 5, PP)
9	Reduce	S-VP	(0, 5, S-VP)

Cross + Huang 2016




$O(3^n)$

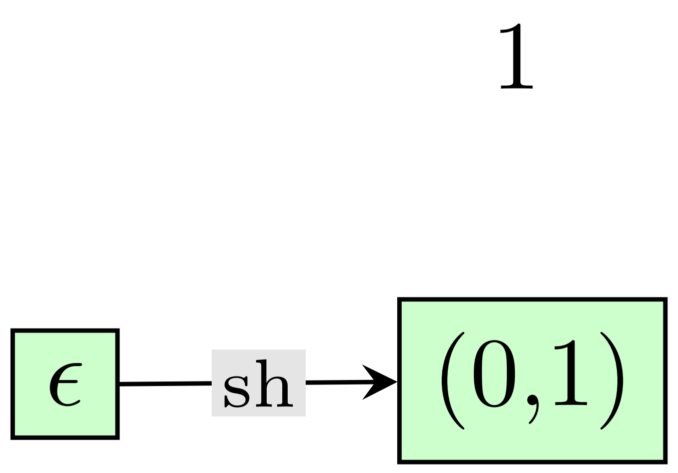
Using a Graph Structured Stack

- This parsing procedure requires a stack of spans.
- We can use a Graph Structured Stack
  - To keep track of the next span on the stack
- And only use the top span (i, j) as our parsing state.

Graph Structured Stack Example

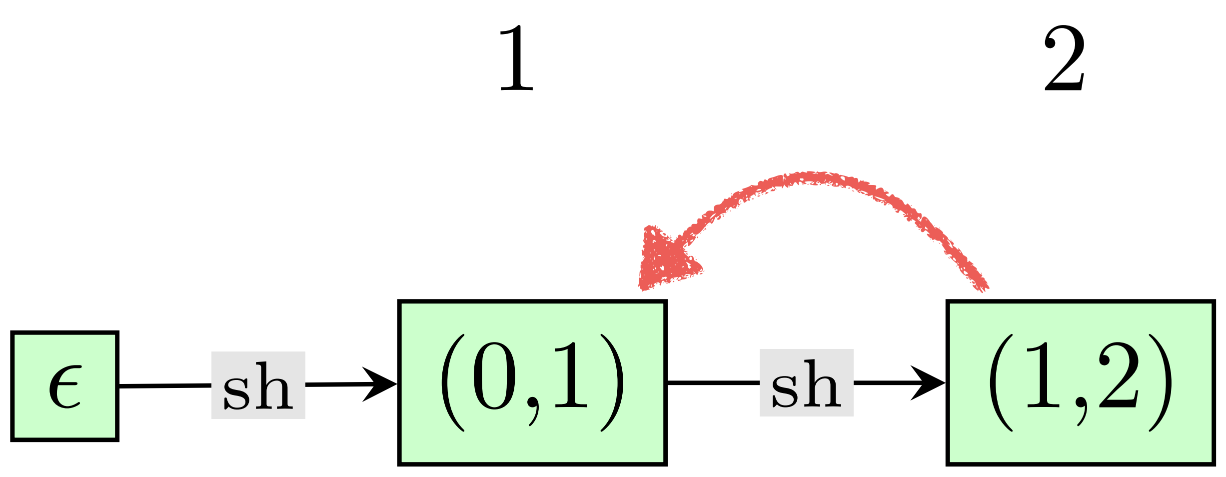
(Note: Spans are independently labeled)

(So we can worry about just the spans themselves)



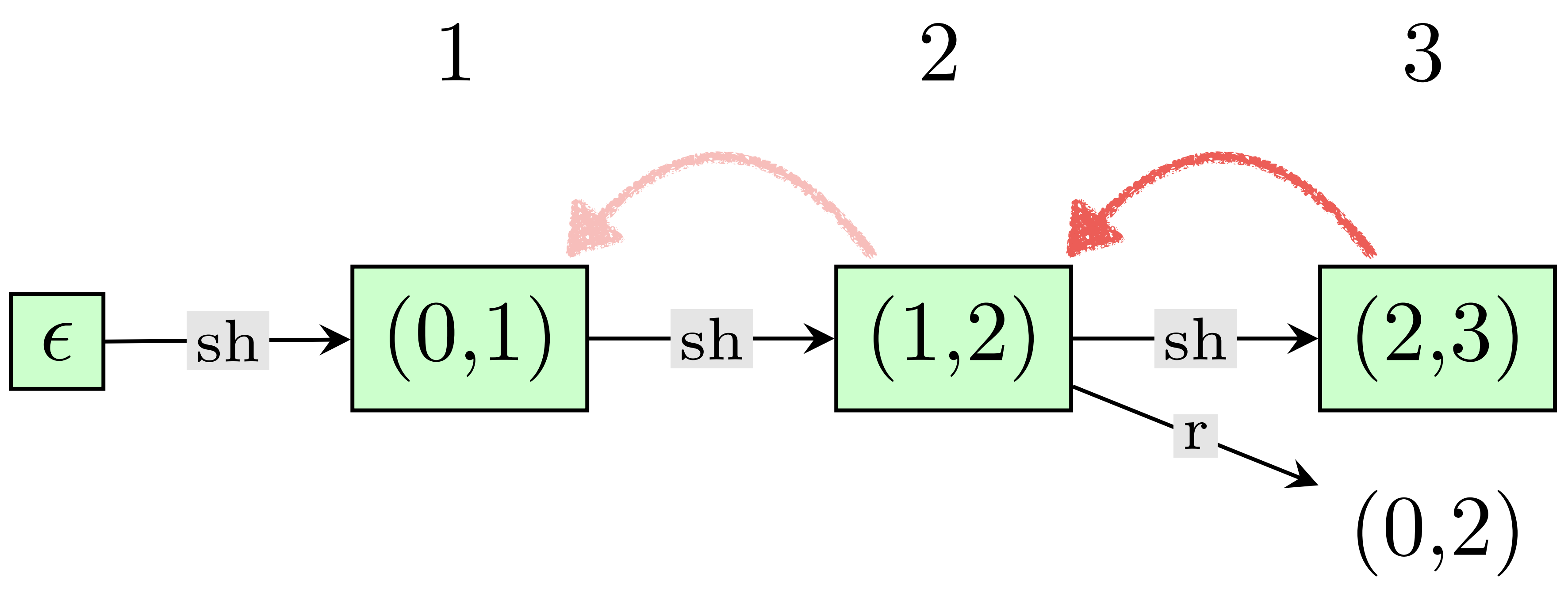
Gold:	Shift (0,1)

GSS Graph Structured Stack Example



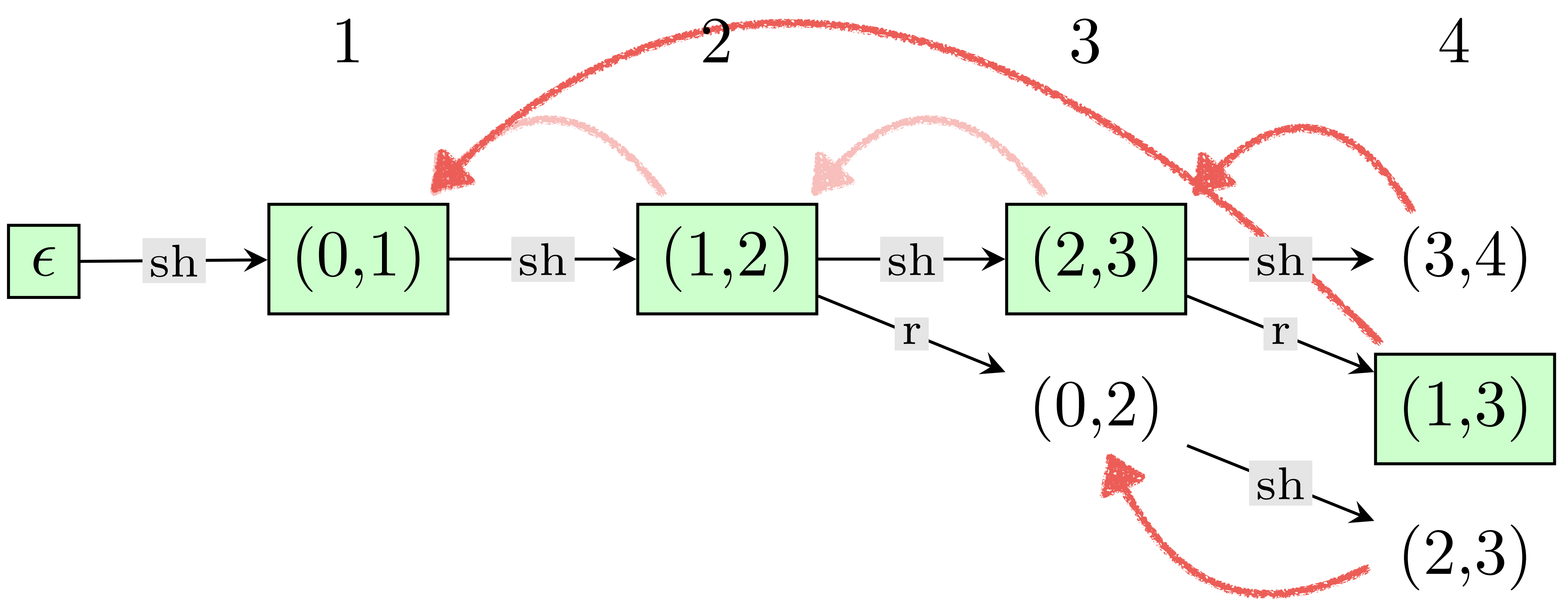
Gold:	Shift (0,1)	Shift (1,2)

Graph Structured Stack Example



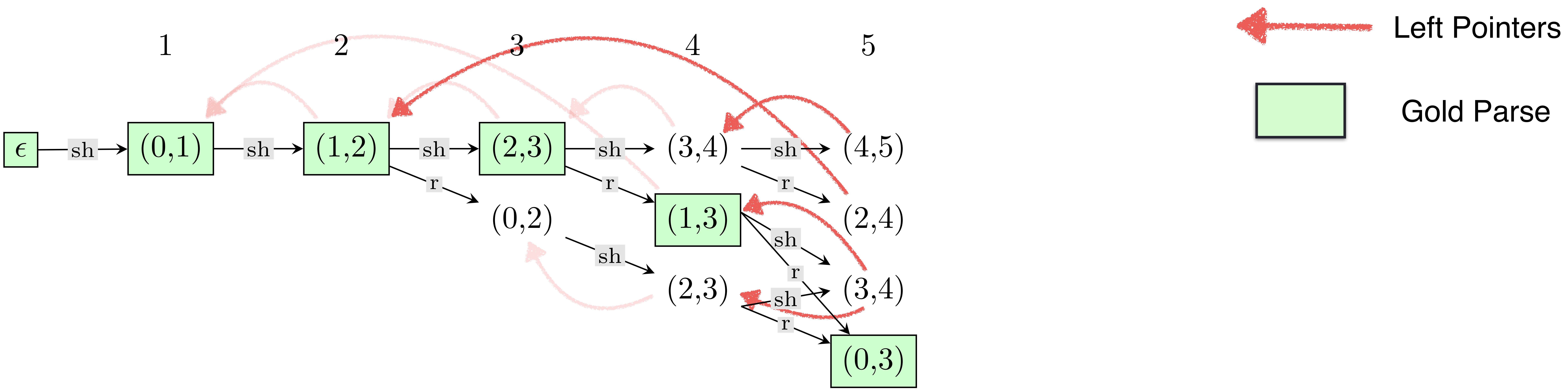
Gold:	Shift (0,1)	Shift (1,2)	Shift (2, 3)

Graph Structured Stack Example



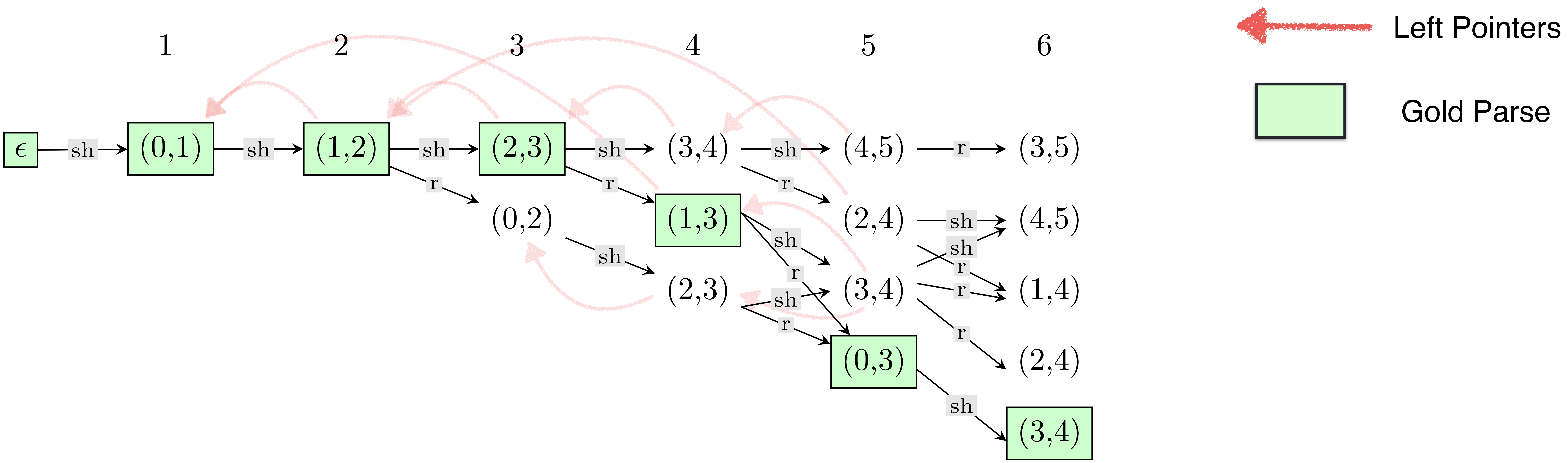
Gold:	Shift (0,1)	Shift (1,2)	Shift (2, 3)	Reduce (1, 3)

Graph Structured Stack Example



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

Graph Structured Stack Example

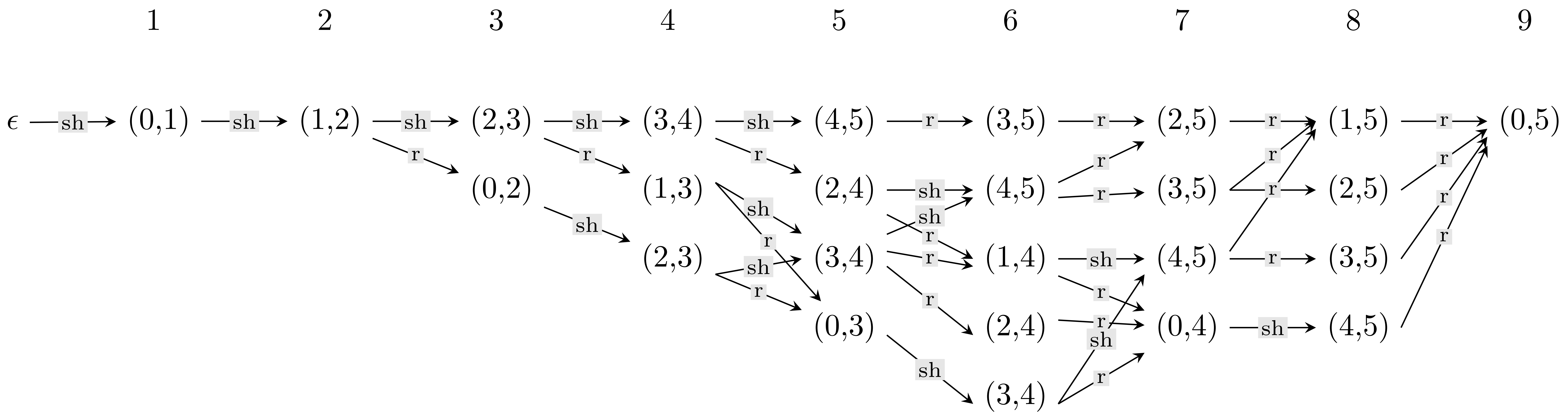


Gold:	Shift (0,1)	Shift (1,2)	Shift (2, 3)	Reduce (1, 3)	Reduce (0, 3)	Shift (3, 4)

Graph Structured Stack Example

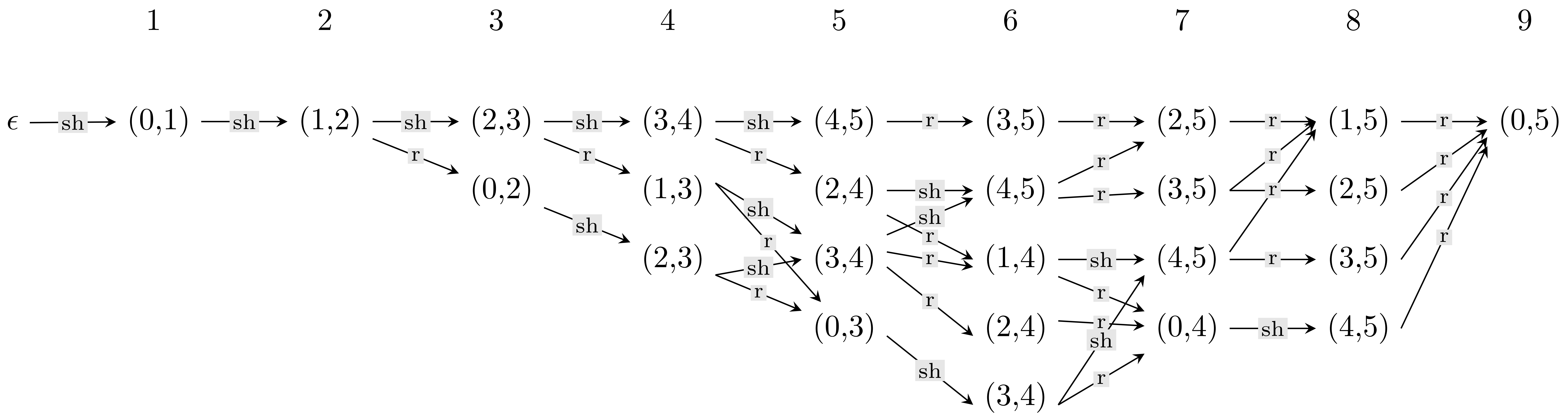
Gold:	Shift (0,1)	Shift (1,2)	Shift (2, 3)	Reduce (1, 3)	Reduce (0, 3)	Shift (3, 4)	Shift (4, 5)	Reduce (3, 5)	Reduce (0, 5)

Runtime Analysis: $O(n^4)$



Huang+Sagae 2010

Runtime Analysis: $O(n^4)$



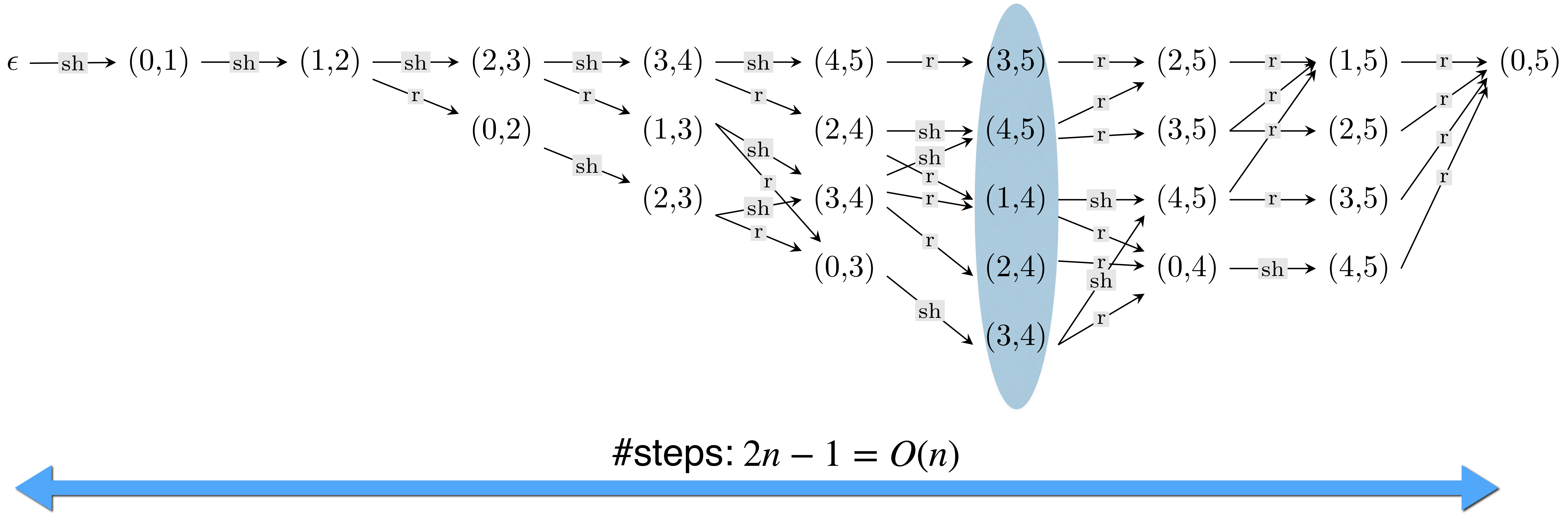
#steps: $2n - 1 = O(n)$

Huang+Sagae 2010

Runtime Analysis: $O(n^4)$

(i, j)

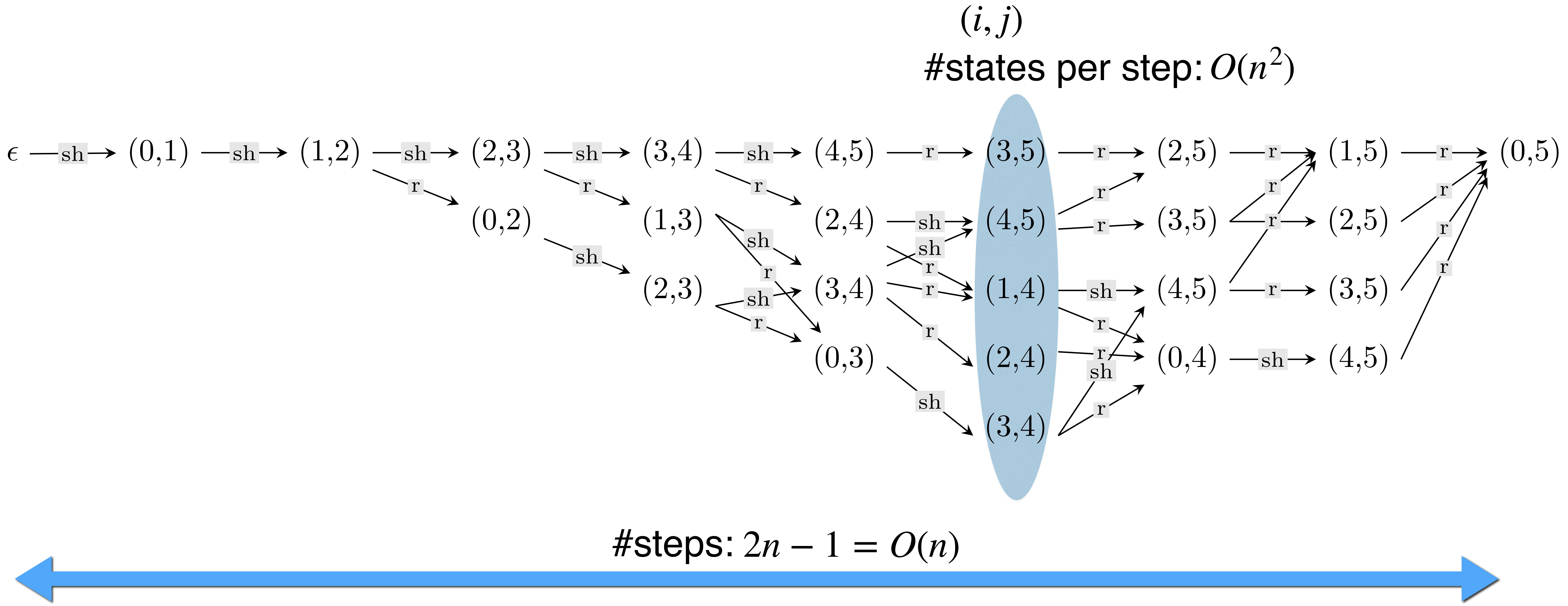
#states per step: $O(n^2)$



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

Huang+Sagae 2010

Runtime Analysis: $O(n^4)$



$O(n^3)$ states

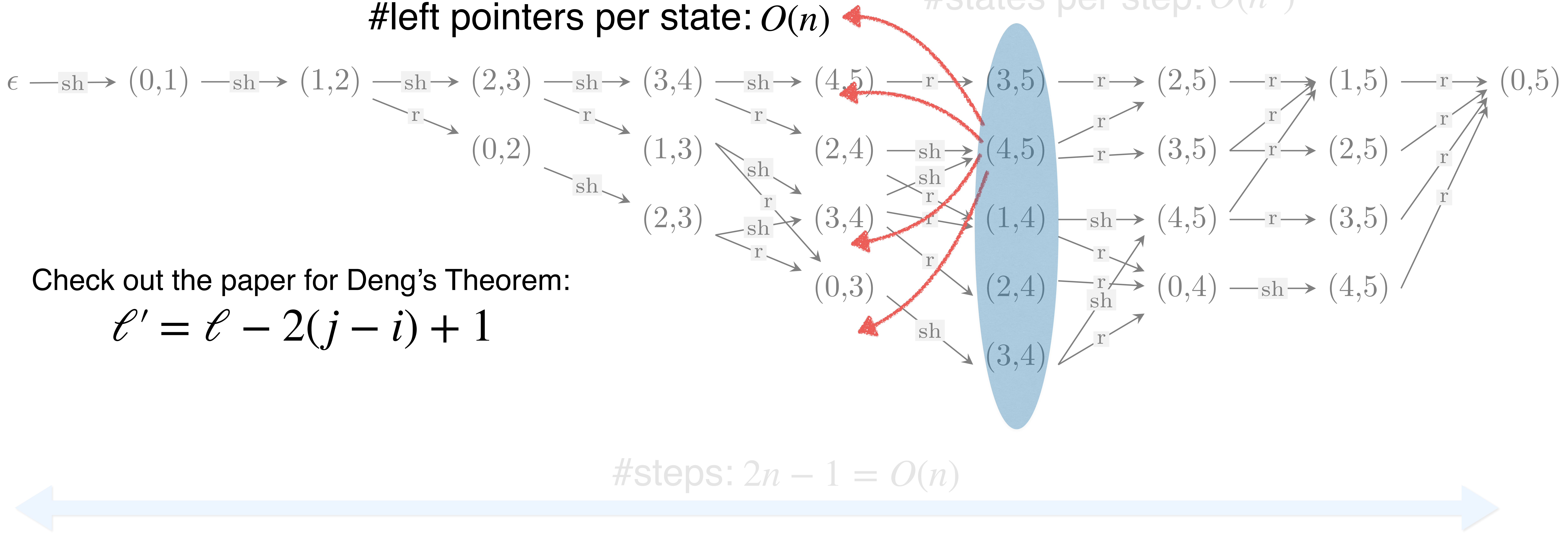
Huang+Sagae 2010

Runtime Analysis: $O(n^4)$

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

(i, j)

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



Check out the paper for Deng's Theorem:

$$\ell' = \ell - 2(j - i) + 1$$

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

$O(n^3)$ states

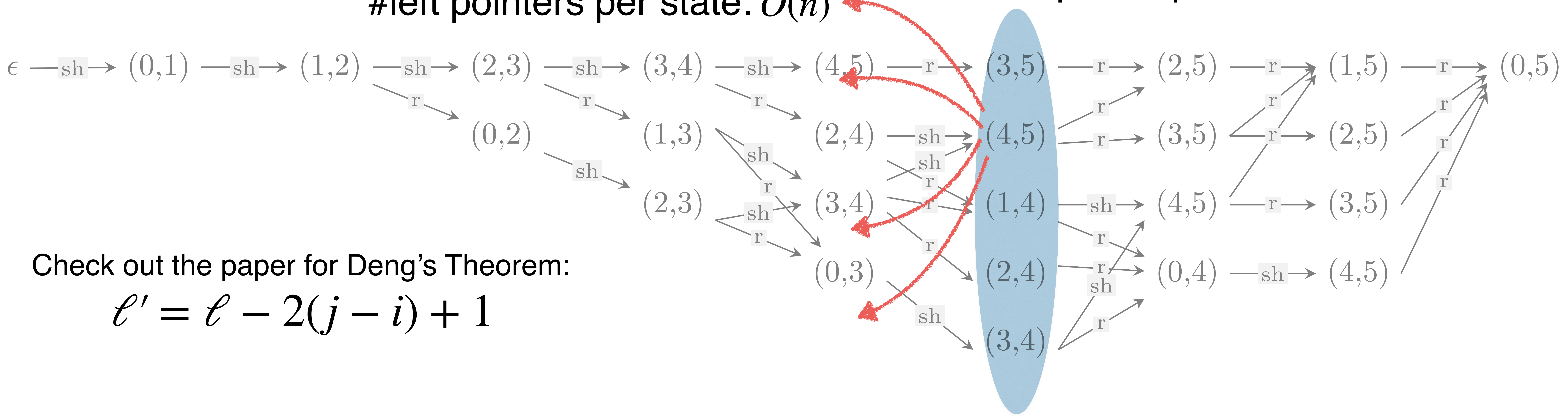
Huang+Sagae 2010

Runtime Analysis: $O(n^4)$

(i, j)

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

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



Check out the paper for Deng's Theorem:

$$\ell' = \ell - 2(j - i) + 1$$

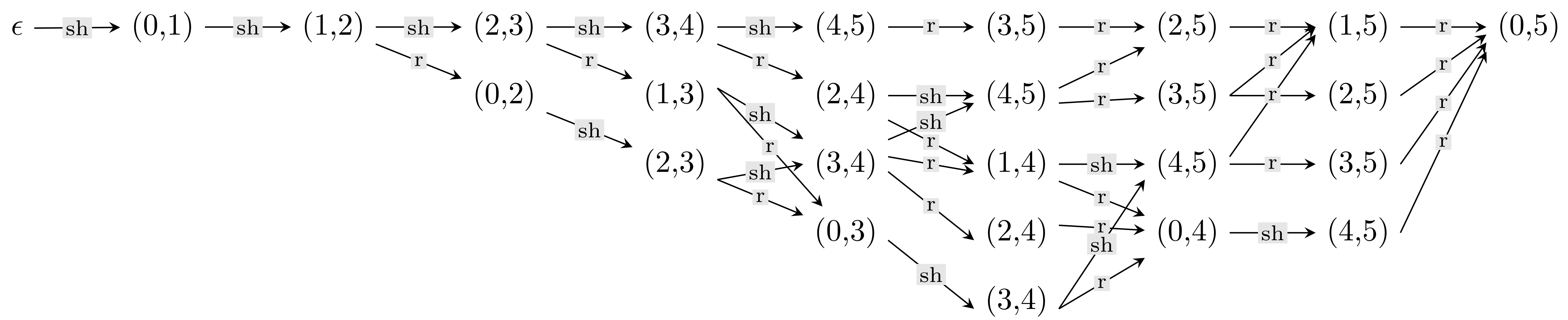
#steps: $2n - 1 = O(n)$

$O(n^3)$ states with $O(n)$ reduce actions: $O(n^4)$ runtime

Huang+Sagae 2010

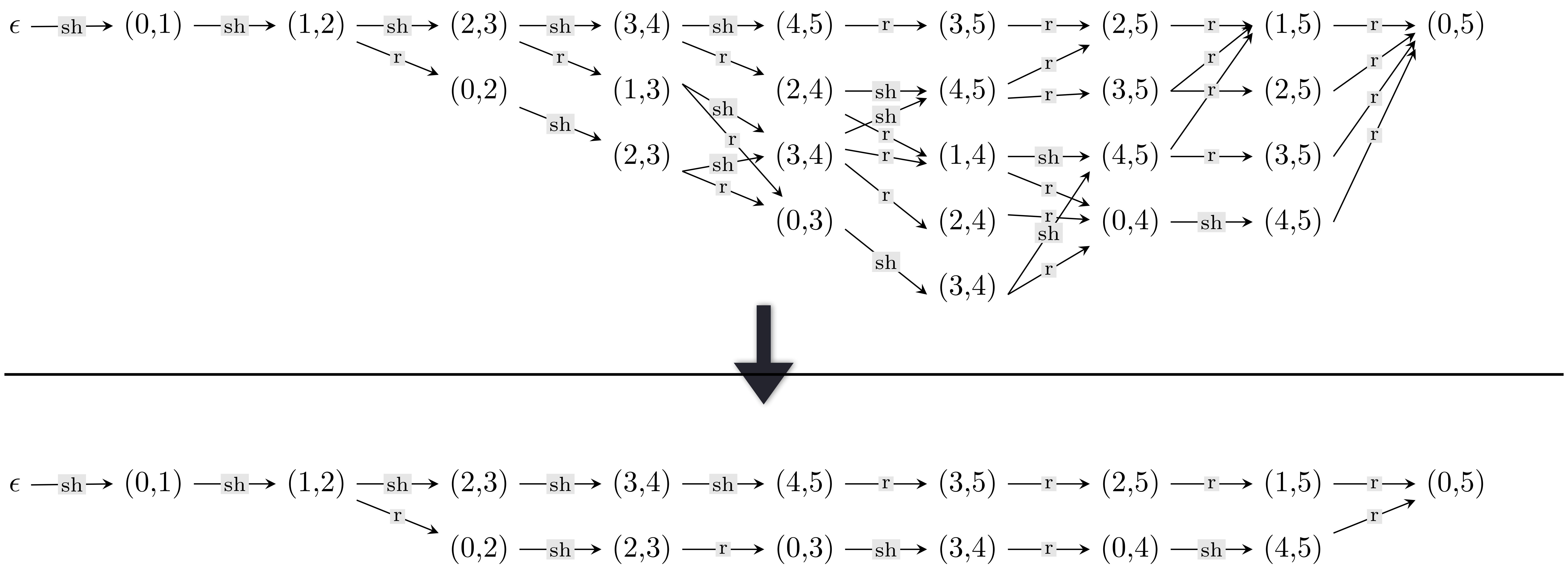
Going slower to go faster

- Our Action-Synchronous algorithm has a slower runtime than CKY!



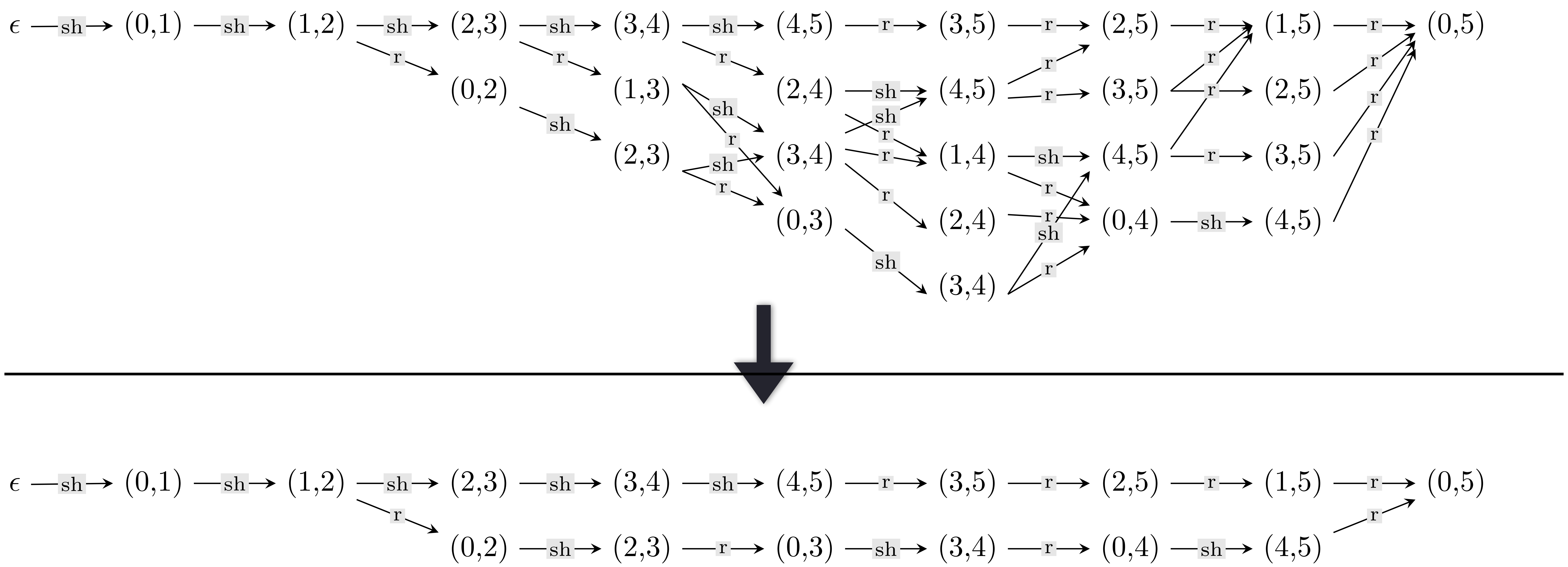
Going slower to go faster

- Our Action-Synchronous algorithm has a slower runtime than CKY!
- However, it also becomes straightforward to prune using beam search.



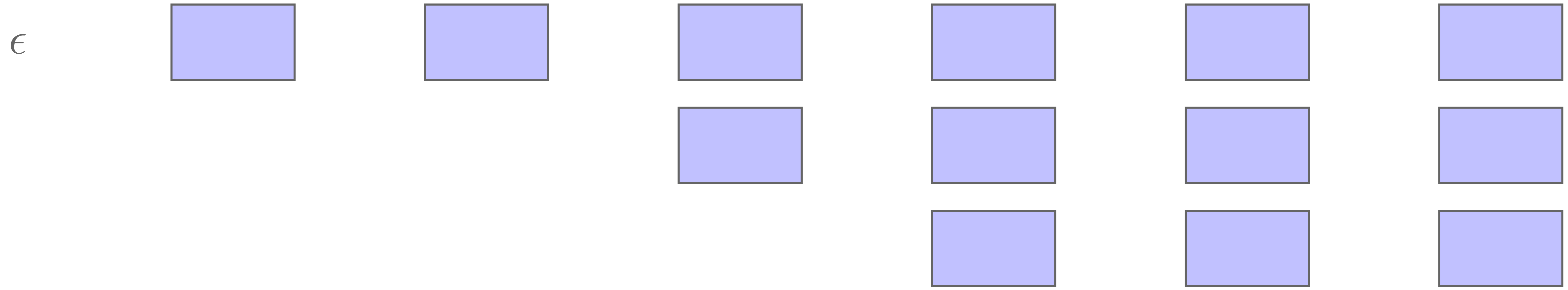
Going slower to go faster

- 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 the O(n) is hiding a constant.



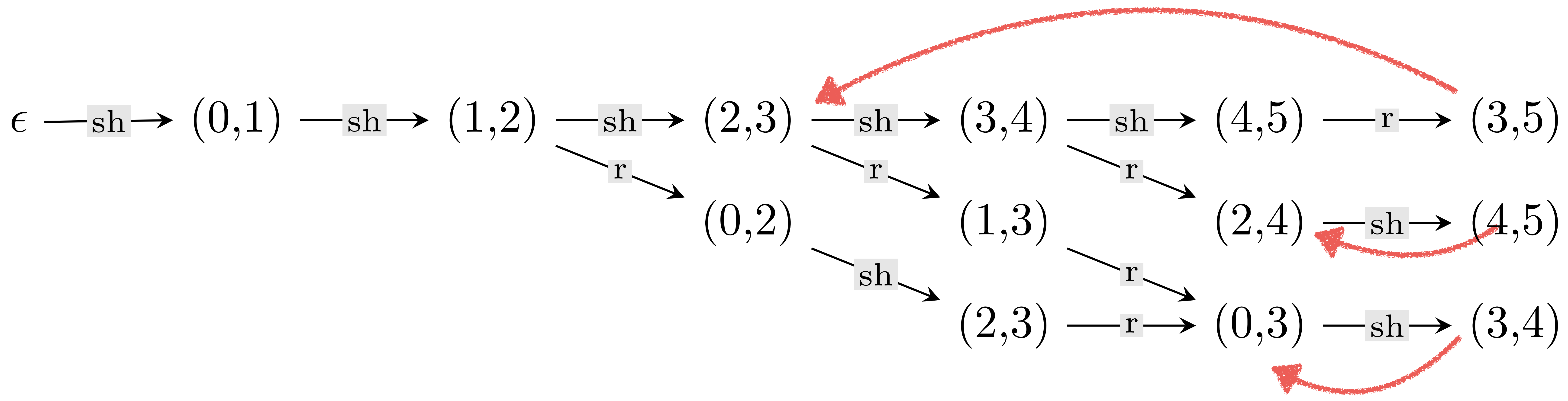
But the O(n) is hiding a constant.

$O(b)$ left pointers per state

$O(nb^2)$ runtime

Cube-Pruning

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

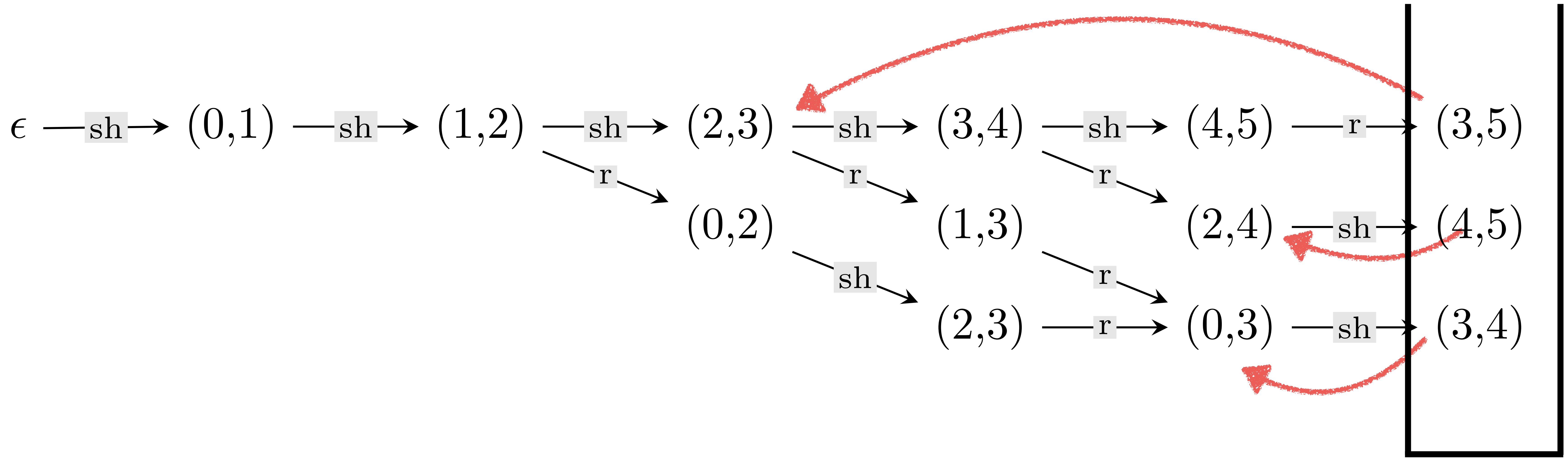


Chiang 2007

Huang+Chiang 2007

Cube-Pruning

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



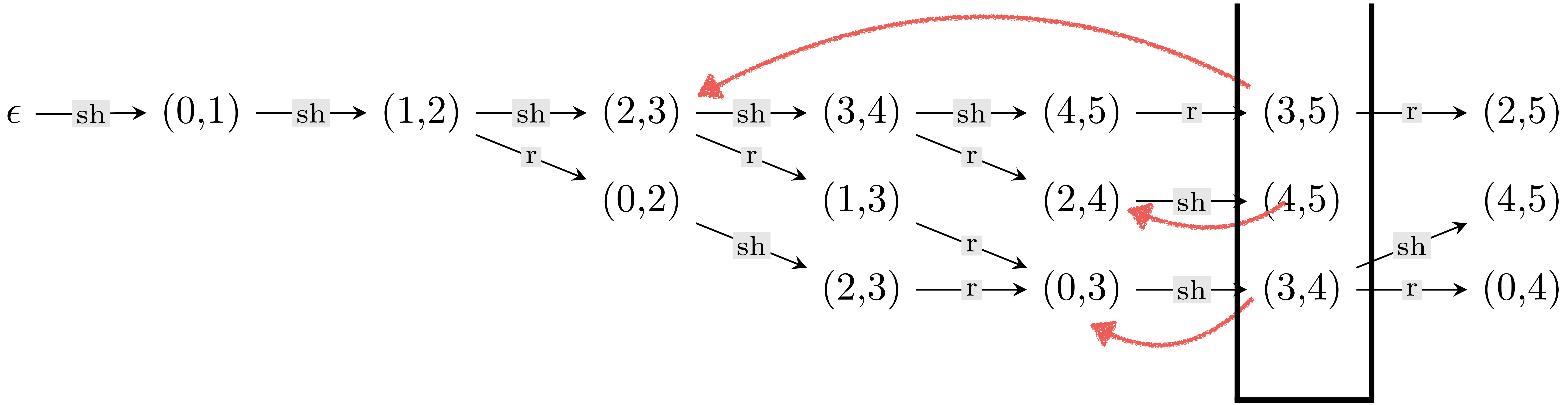
- By pushing all states and their left pointers into a heap

Chiang 2007

Huang+Chiang 2007

Cube-Pruning

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



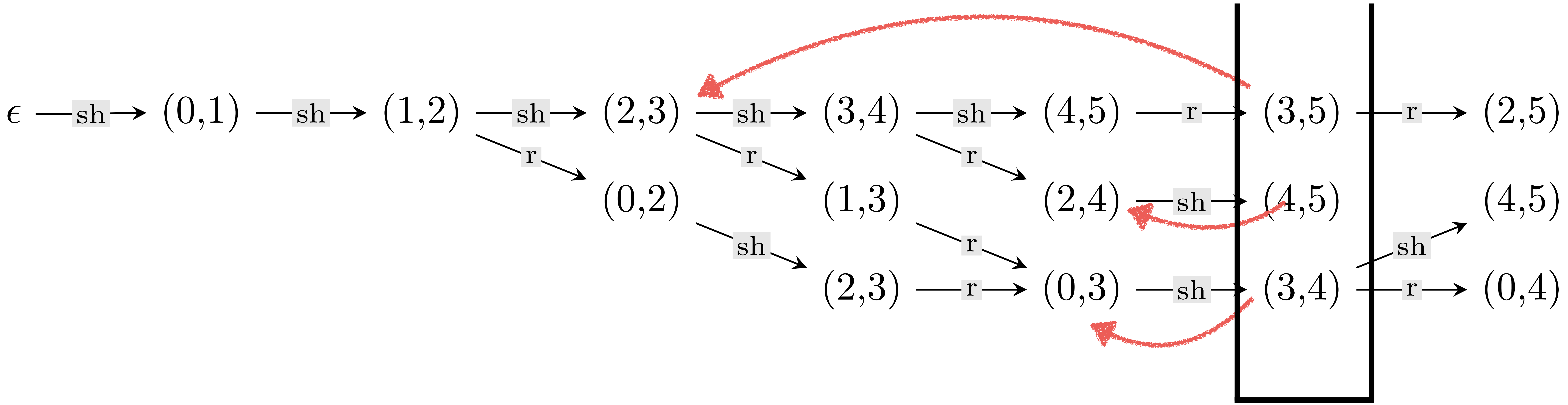
- By pushing all states and their left pointers into a heap
- And popping the top b unique subsequent states

Chiang 2007

Huang+Chiang 2007

Cube-Pruning

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

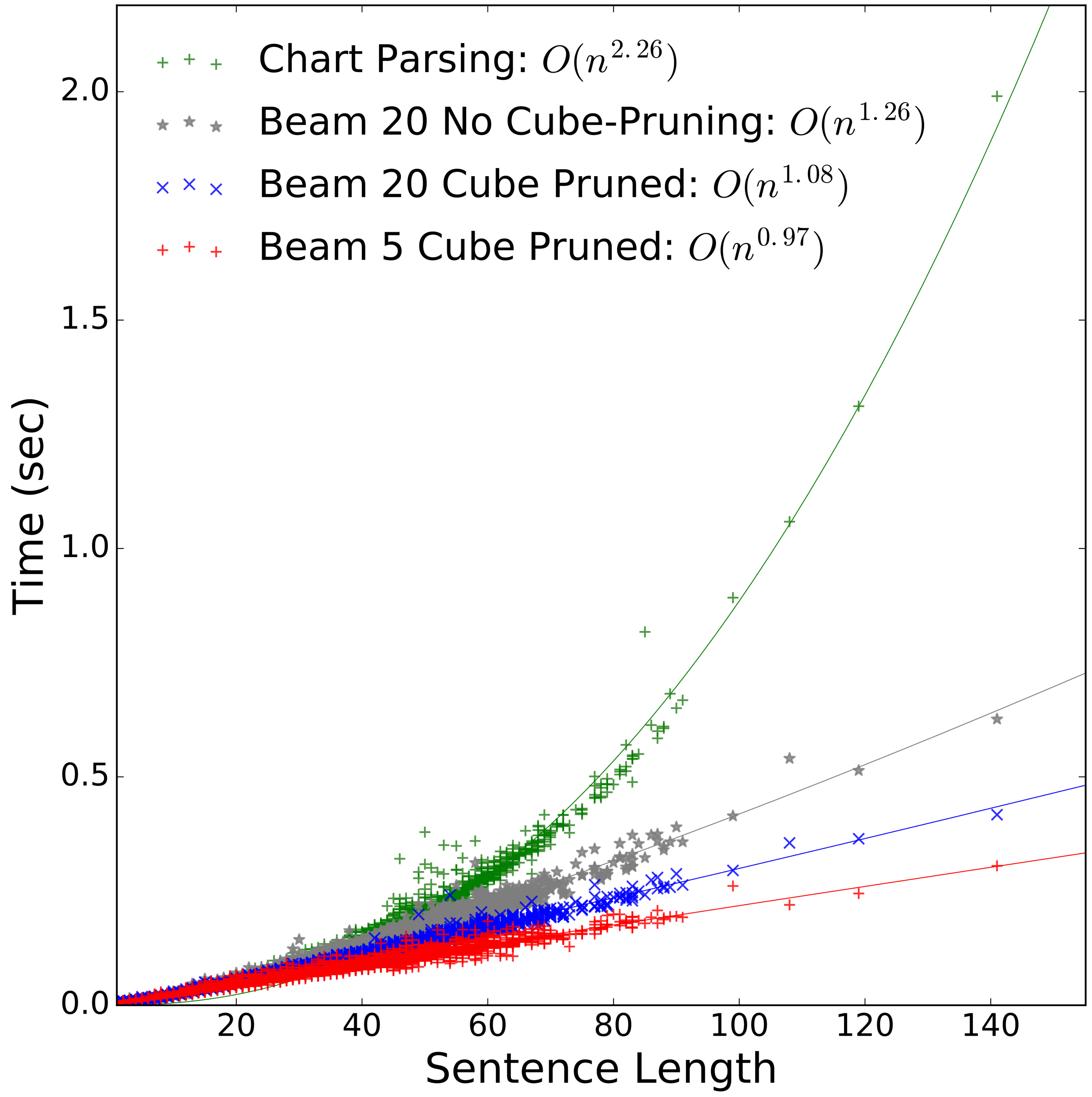


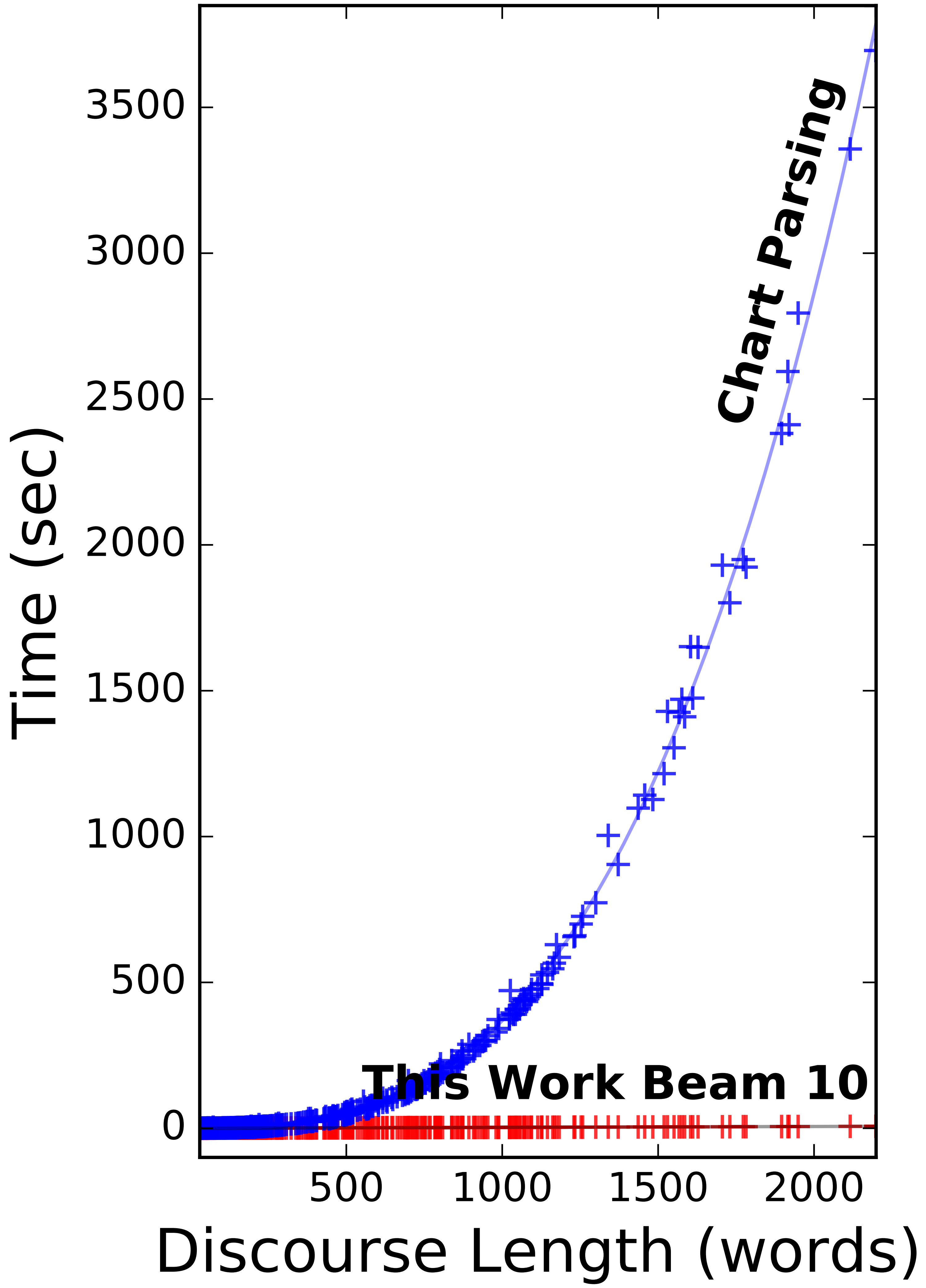
- By pushing all states and their left pointers into a heap
- And popping the top b unique subsequent states
- First time Cube-Pruning has been applied to Incremental Parsing

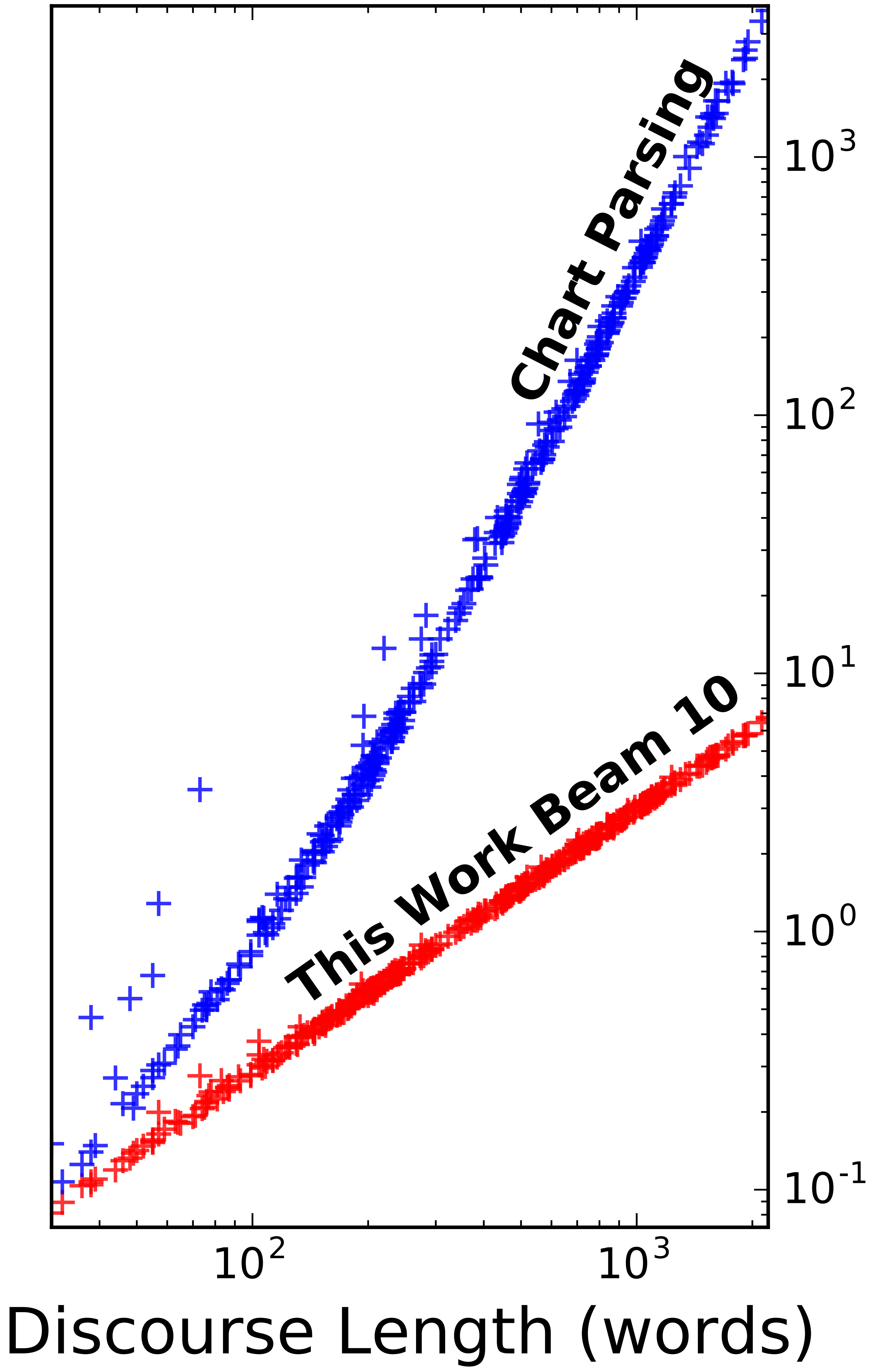
Chiang 2007

Huang+Chiang 2007

Runtime on PTB and Discourse







Training

Training

- Structured SVM approach:
  - Goal: Score the gold tree higher than all others by a margin:

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

Training

- Structured SVM approach:
  - 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 tree with highest augmented score:

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

Training

- Structured SVM approach:
  - Goal: Score the gold tree higher than all others by a margin:

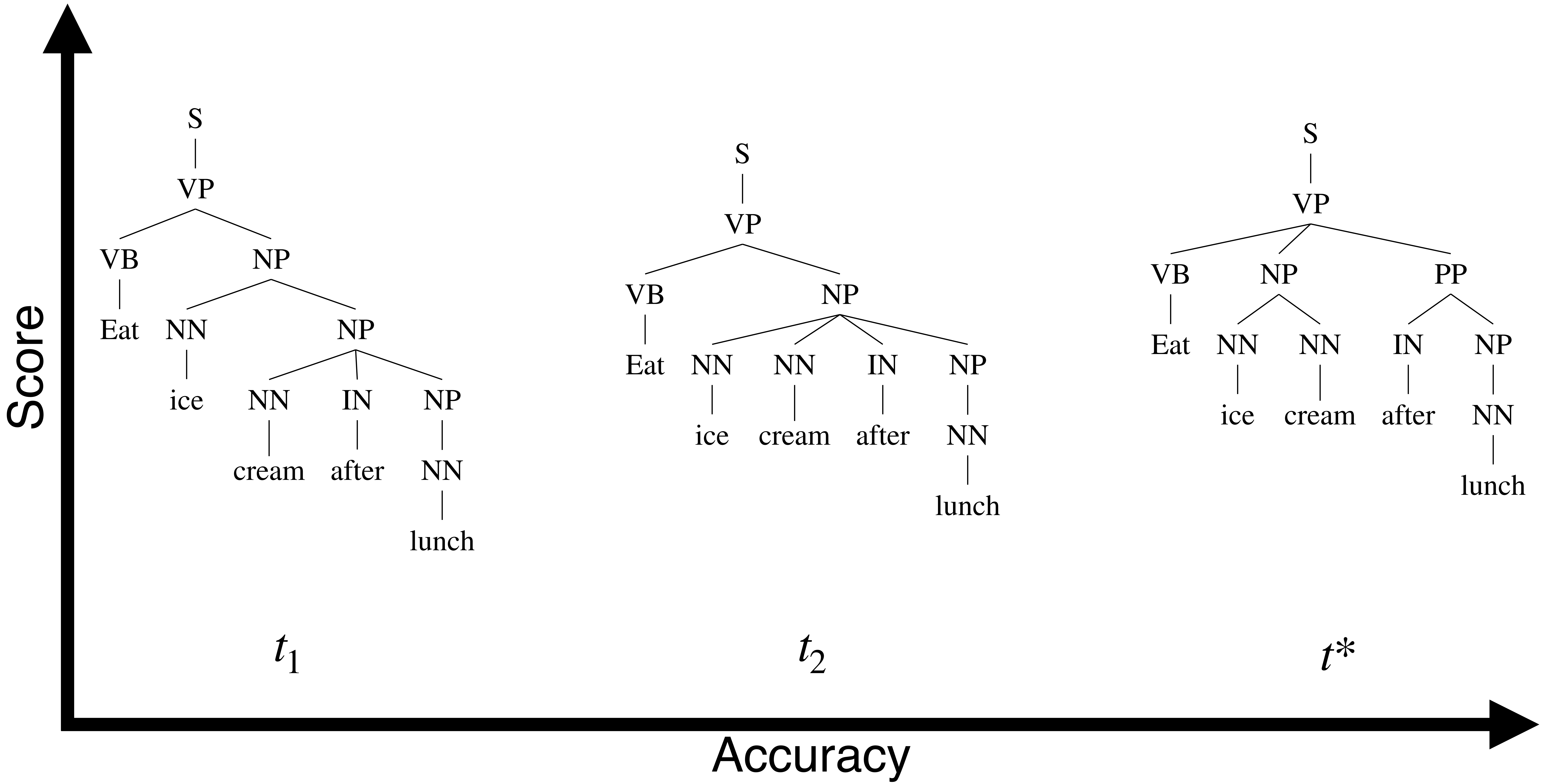
$$\forall t, s(t^*) - s(t) \geq \Delta(t, t^*)$$

- Loss Augmented Decoding:
  - During Training: Return tree with highest augmented score:

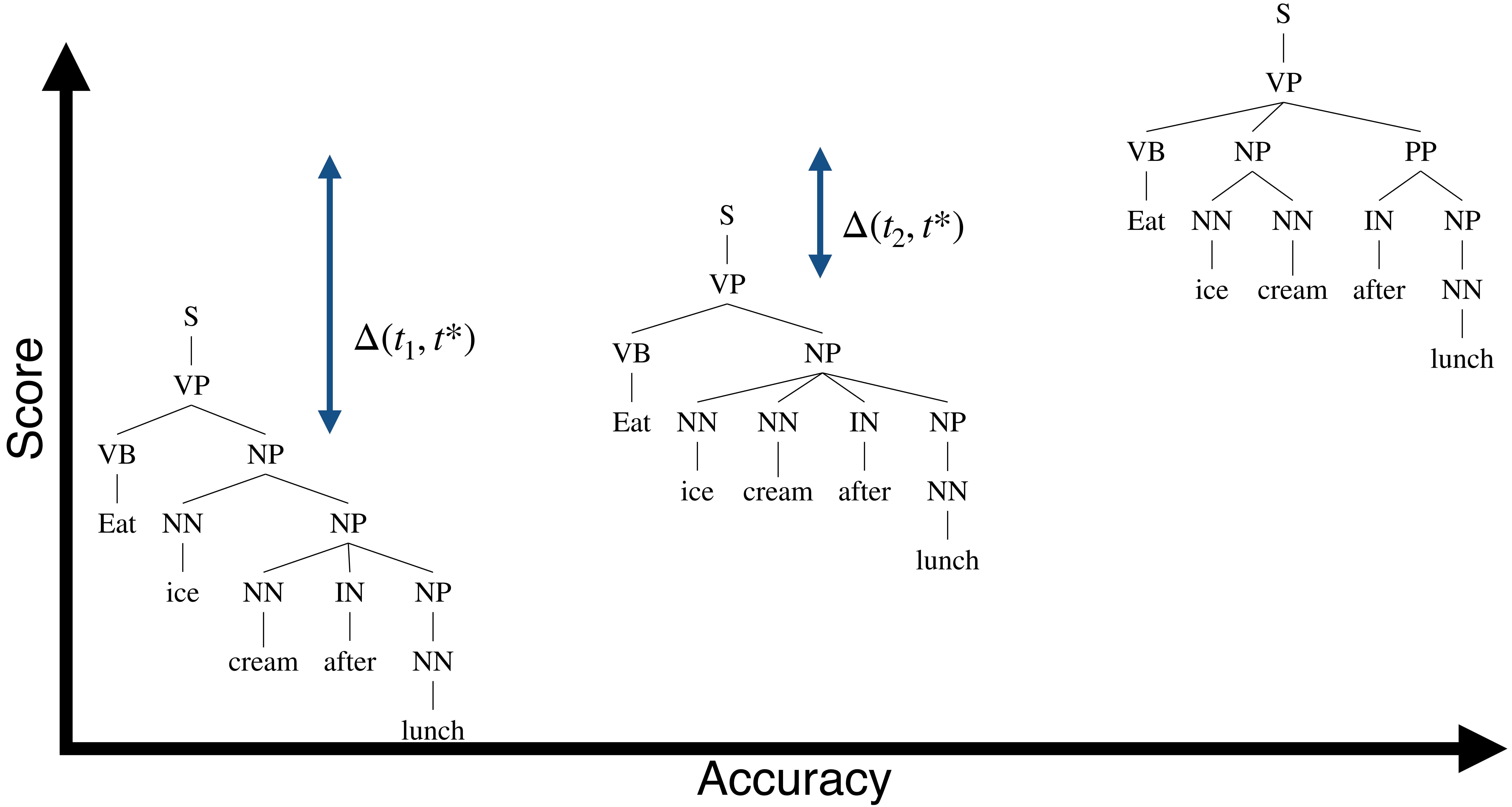
$$\hat{t} = \arg\max_t \left(s(t) + \Delta(t, t^*) \right)$$

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

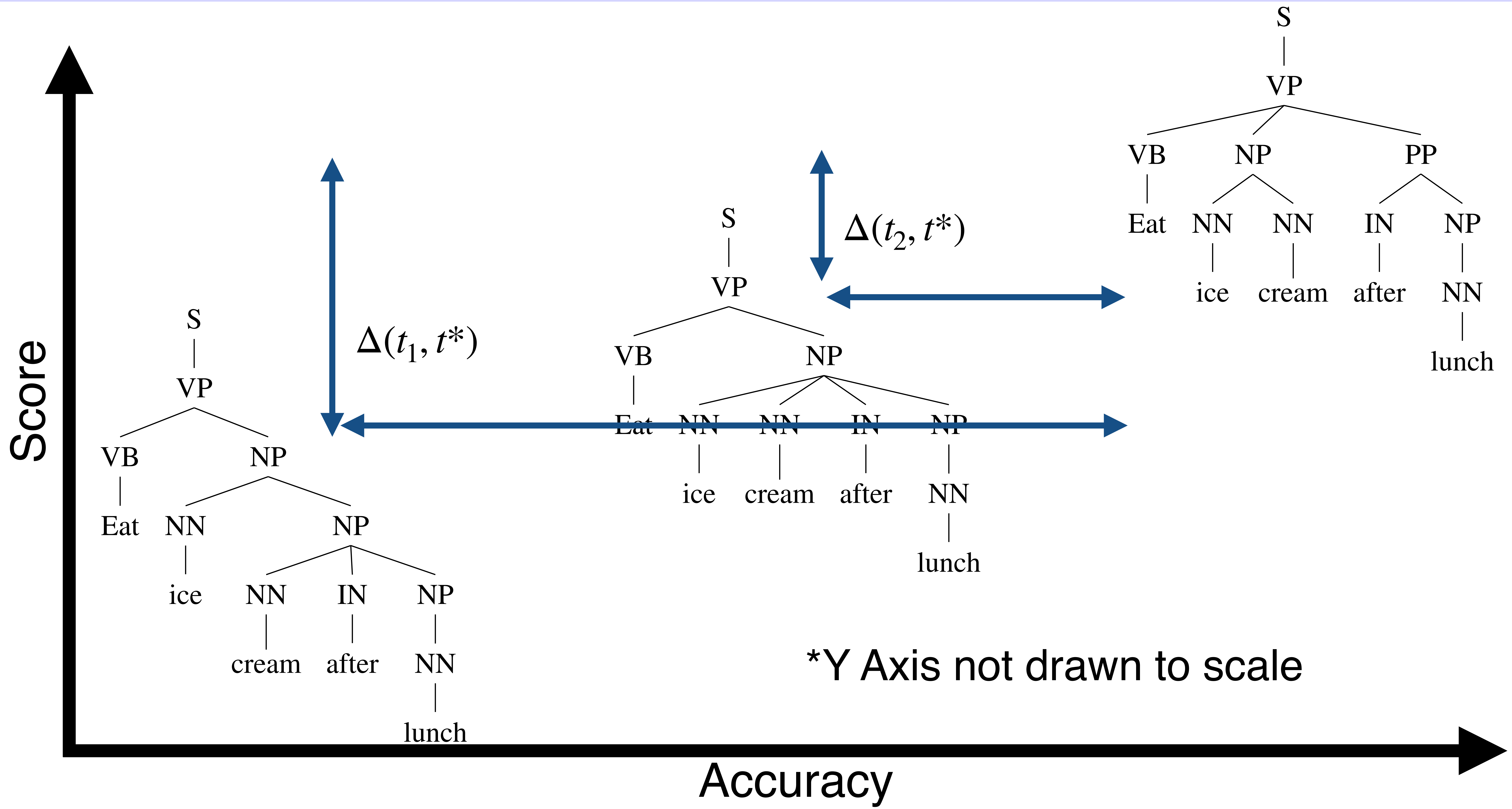
Before training:



Ideally after training:



Ideally after training:



Delta Margins

- Counts the incorrectly labeled spans in the tree.
  - Happens to be decomposable, so can even be used to compare partial trees.

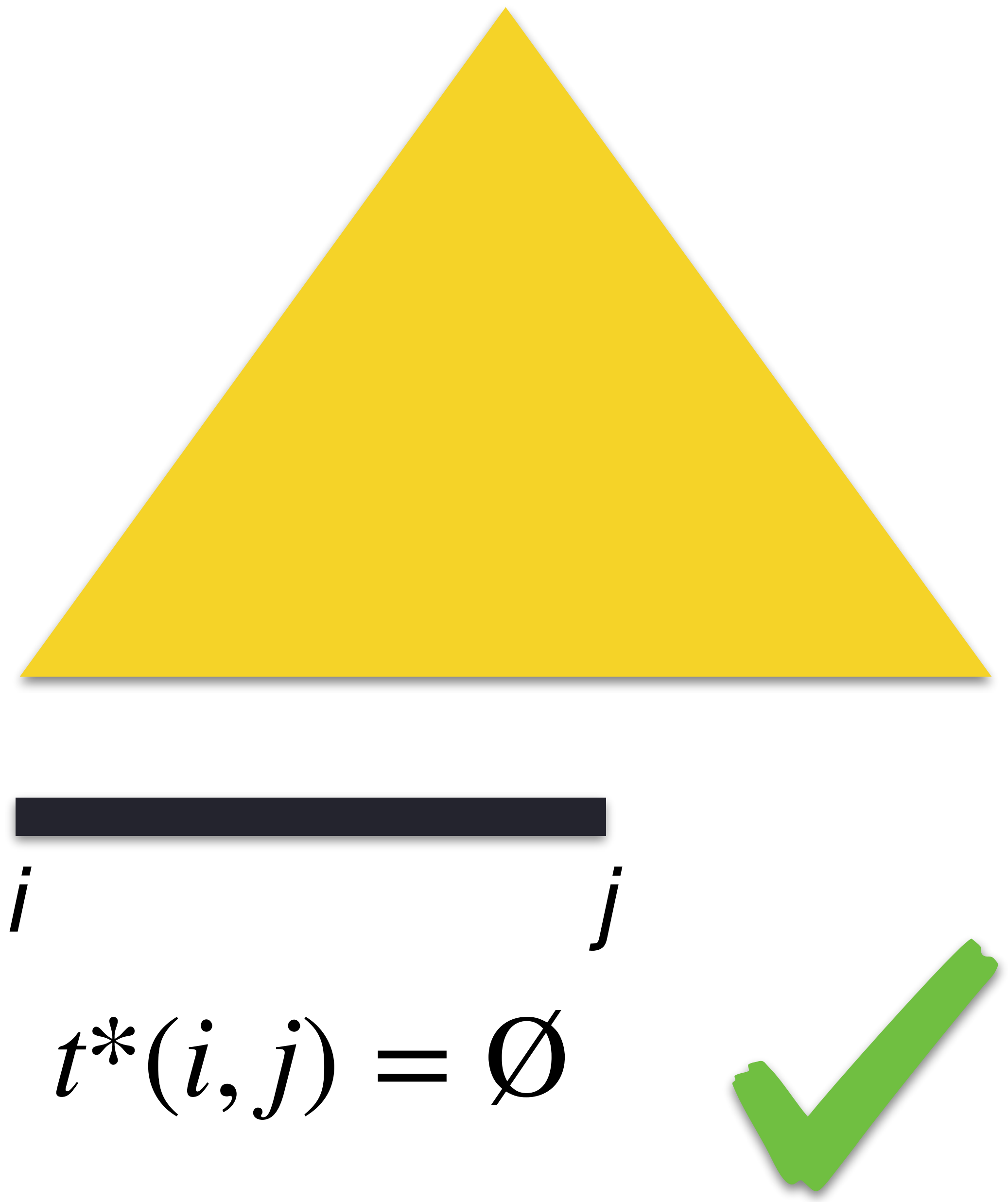
$$\Delta(t, t^*) = \sum_{(i,j,X) \in t} \mathbb{1}\left(X \neq t^*_{(i,j)}\right)$$

Cross-Span Loss

- We observe that the null label ø is used in two different ways:

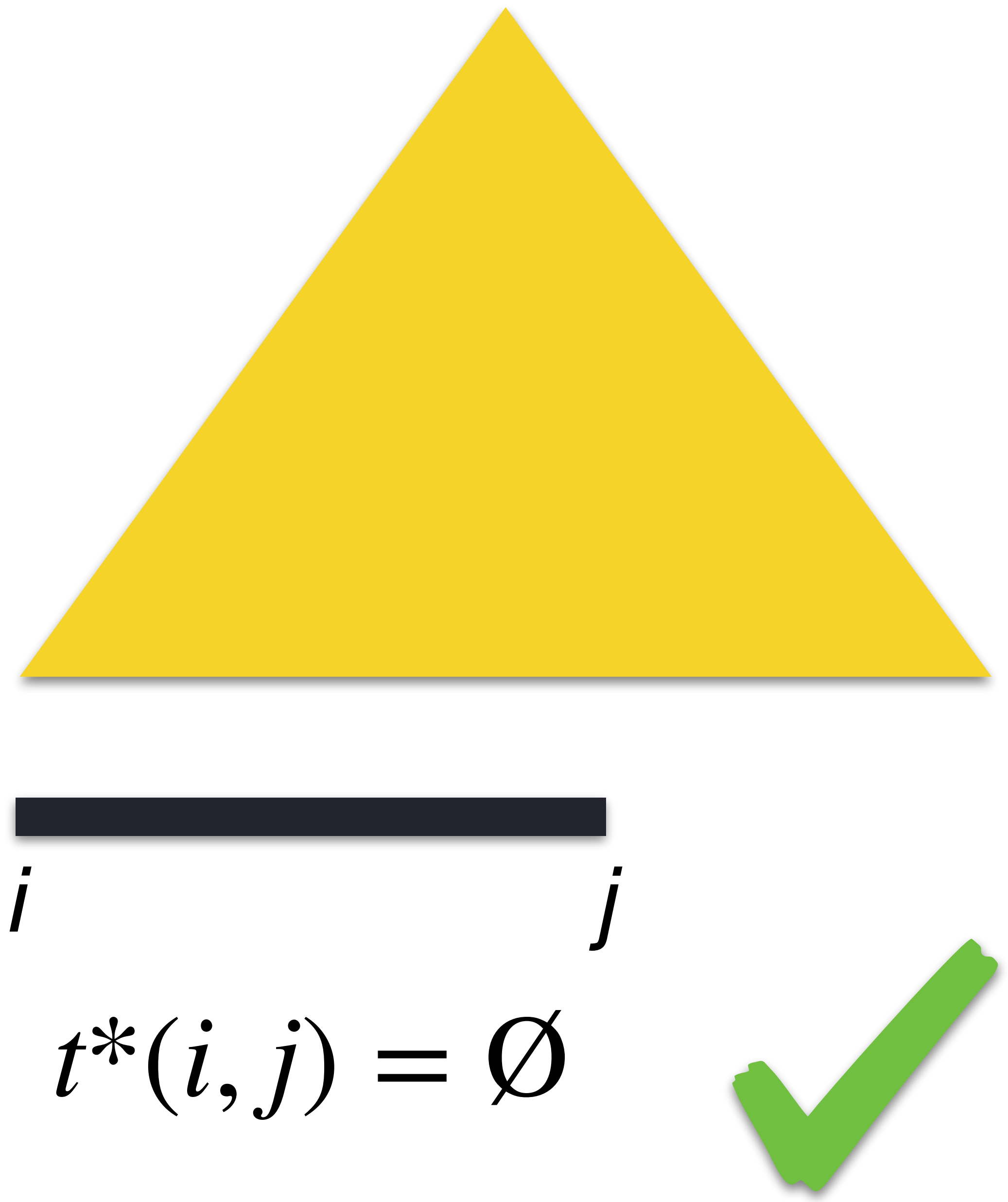
Cross-Span Loss

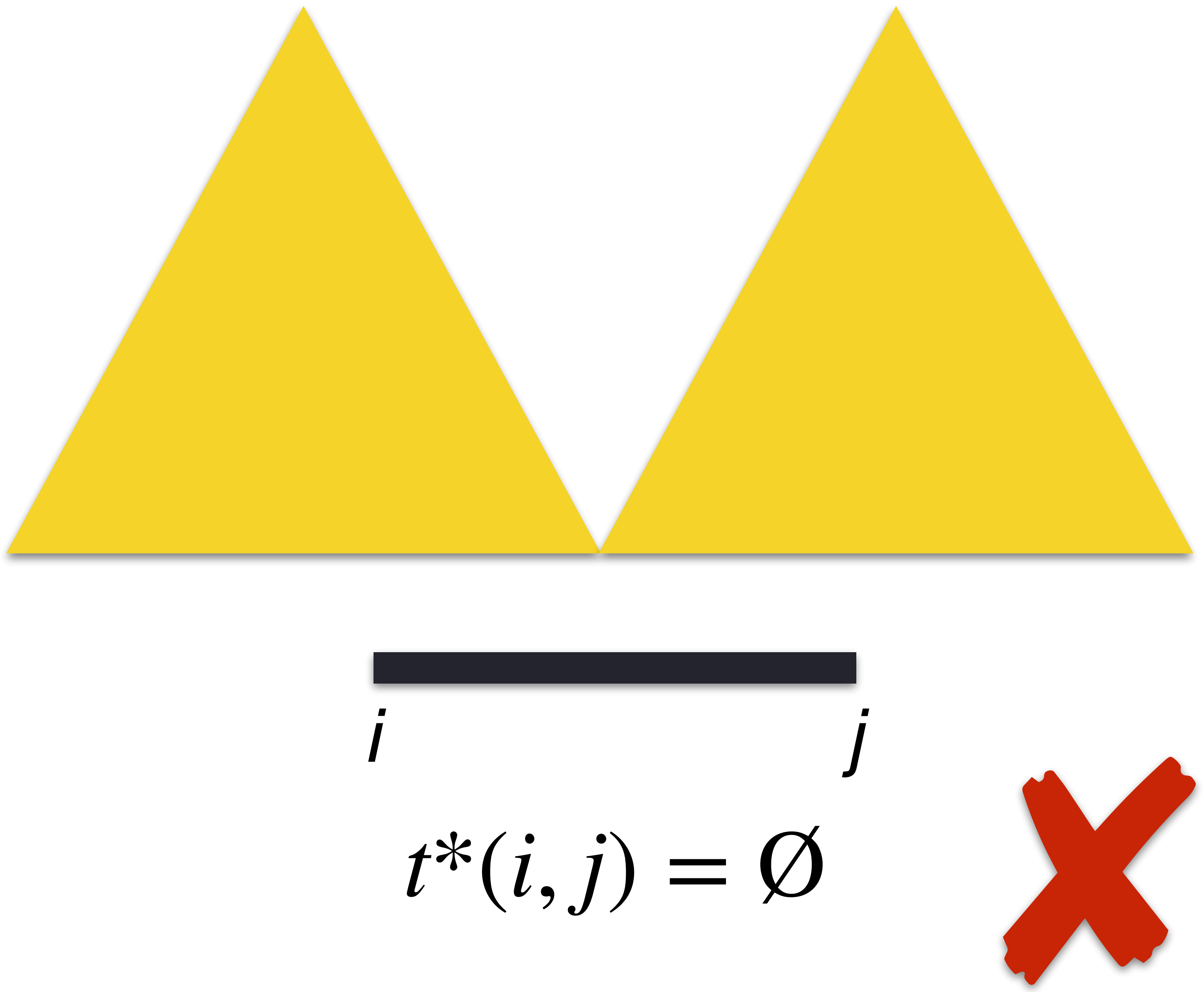
- We observe that the null label ø is used in two different ways:
  - To facilitate ternary and n-ary branching trees.



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.





Cross-Span Loss

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

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

Cross-Span Loss

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

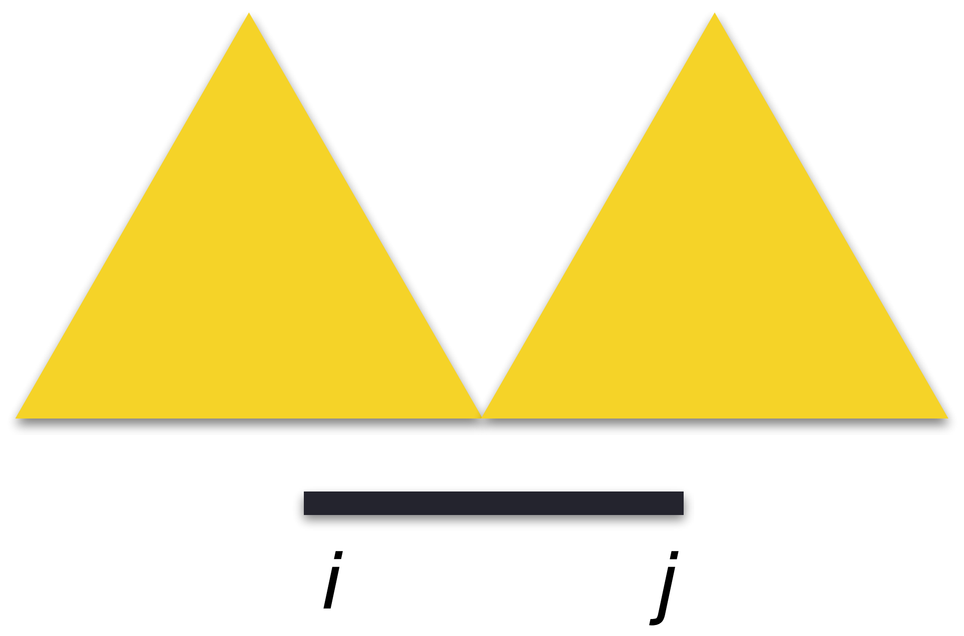
$$\Delta(t, t^*) = \sum_{(i,j,X) \in t} \mathbb{1}\left(X \neq t^*_{(i,j)} \vee \mathrm{cross}(i, j, t^*)\right)$$

Cross-Span Loss

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

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

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



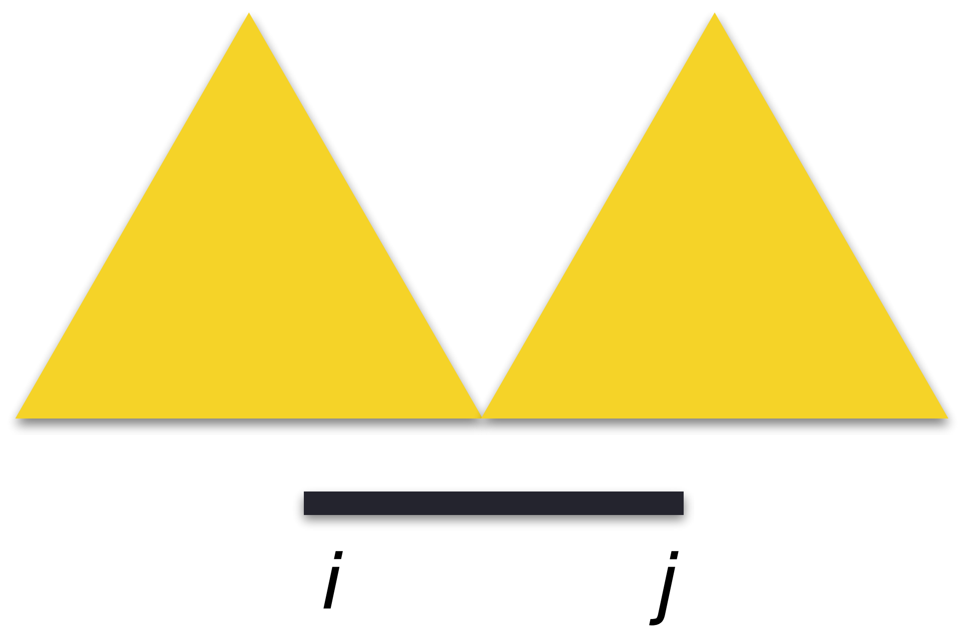
$$\Delta(t, t^*) = \sum_{(i,j,X)\in t} \mathbb{1}\Big(X \neq t^*_{(i,j)} \vee \text{cross}(i, j, t^*)\Big)$$

Cross-Span Loss

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

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

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

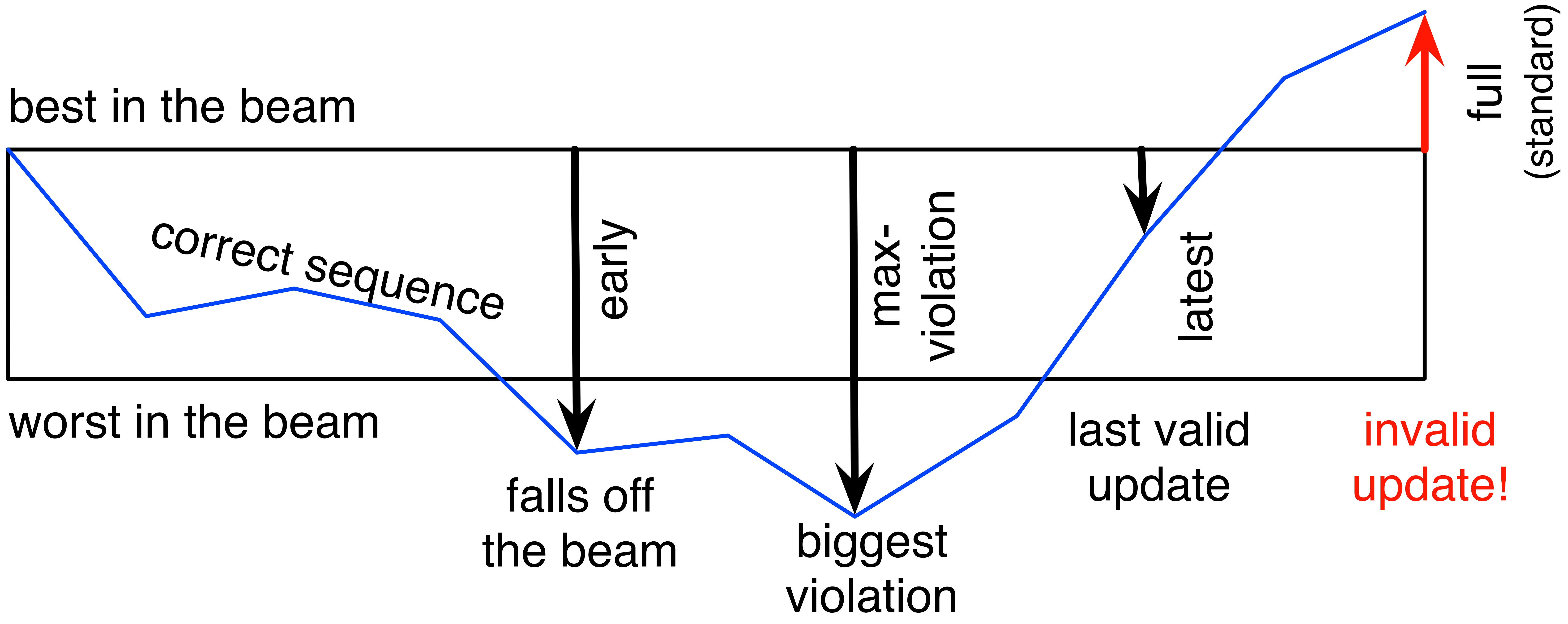


$$\Delta(t, t^*) = \sum_{(i,j,X)\in t} \mathbb{1}\Big(X \neq t^*_{(i,j)} \vee \text{cross}(i, j, t^*)\Big)$$

- 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

Experiments: PTB Test

Model	Note	F1 Score
Stern et al. (2017a)	**Baseline Method (Chart Parser)**	91.79
Stern et al. (2017a) + cross-span	**+our improved loss**	91.81
Stern et al. (2017b)	**Github Code**	91.86
GSS Beam 15	**Our Work**	91.84
GSS Beam 20	**Our Work**	**91.97**

Comparison to other parsers

PTB only, Single Model, End-to-End

Model	Note	F1 Score
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 2017a	Baseline Chart Parser	91.79
Stern 2017c	Separate Decoding	92.56
Our Work	Beam 20	91.97

Reranking, Ensemble, Extra Data

Model	Note	F1 Score
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.

Caveats:

- 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!

Questions?

Thank You