Optimal Shift-Reduce Constituent Parsing with Structured Perceptron

Le Quang Thang

Hanoi University of Science and Technology

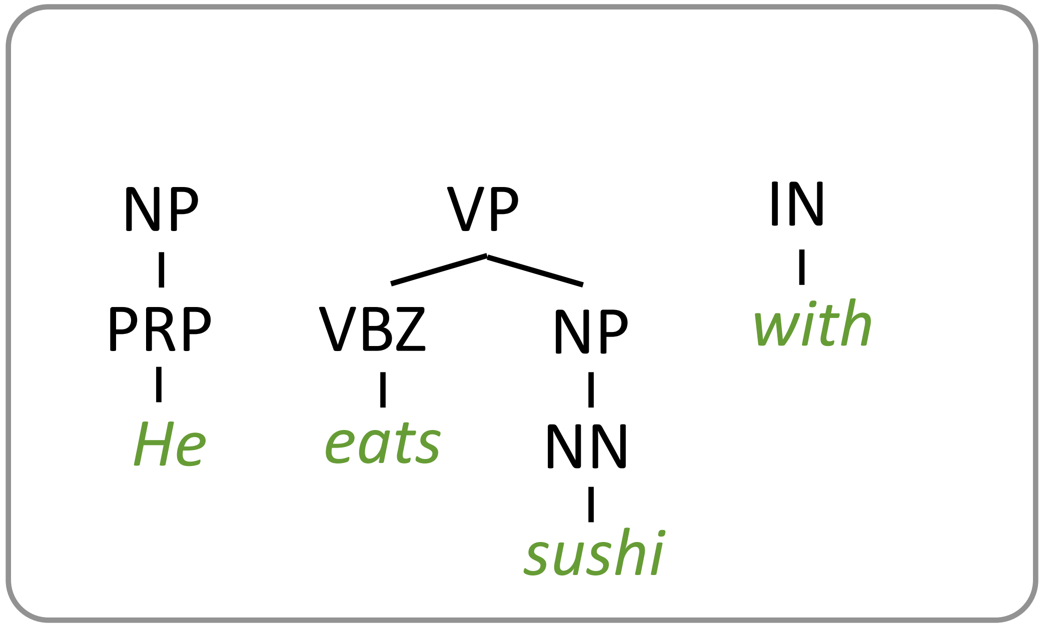
Vietnam

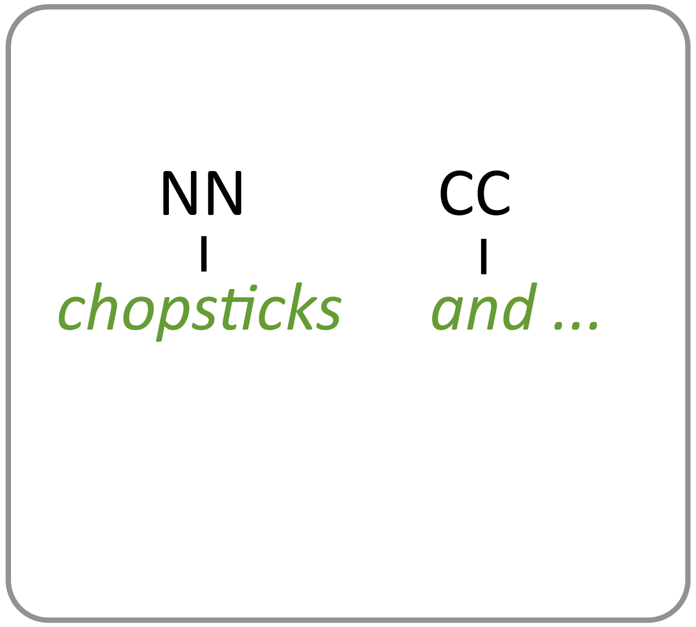
Hiroshi Noji Yusuke Miyao

National Institute of Informatics

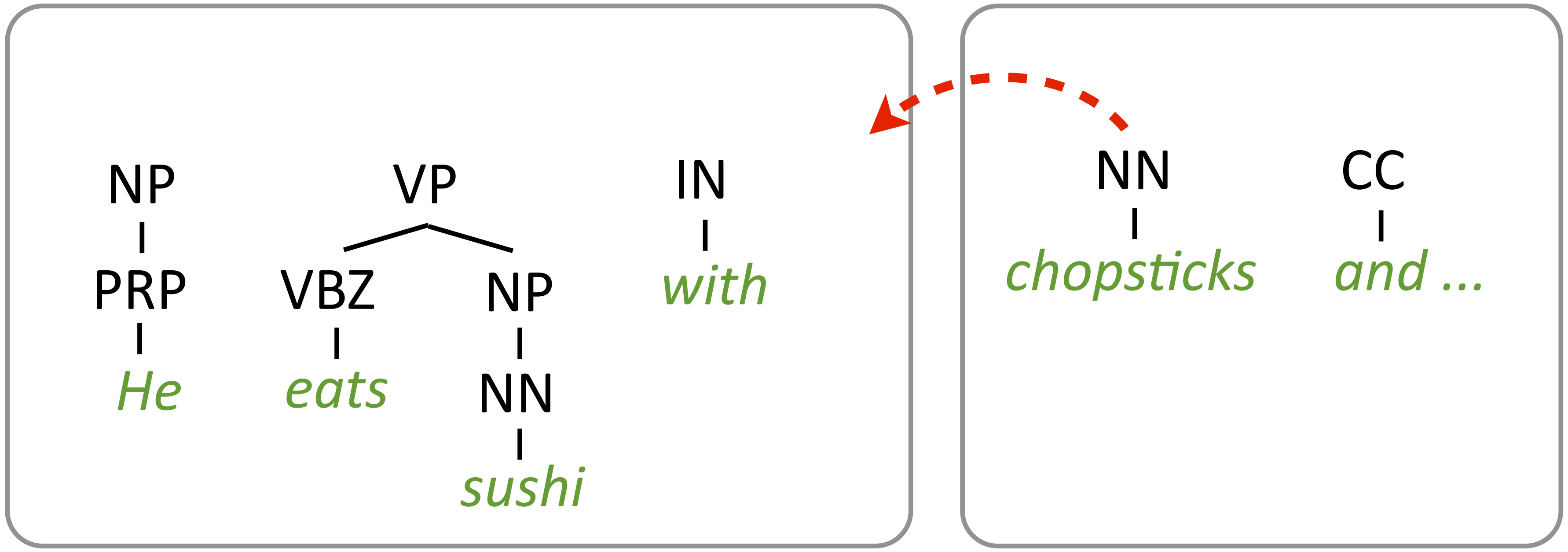
Japan

Advantages of Shift-reduce Parsing ?

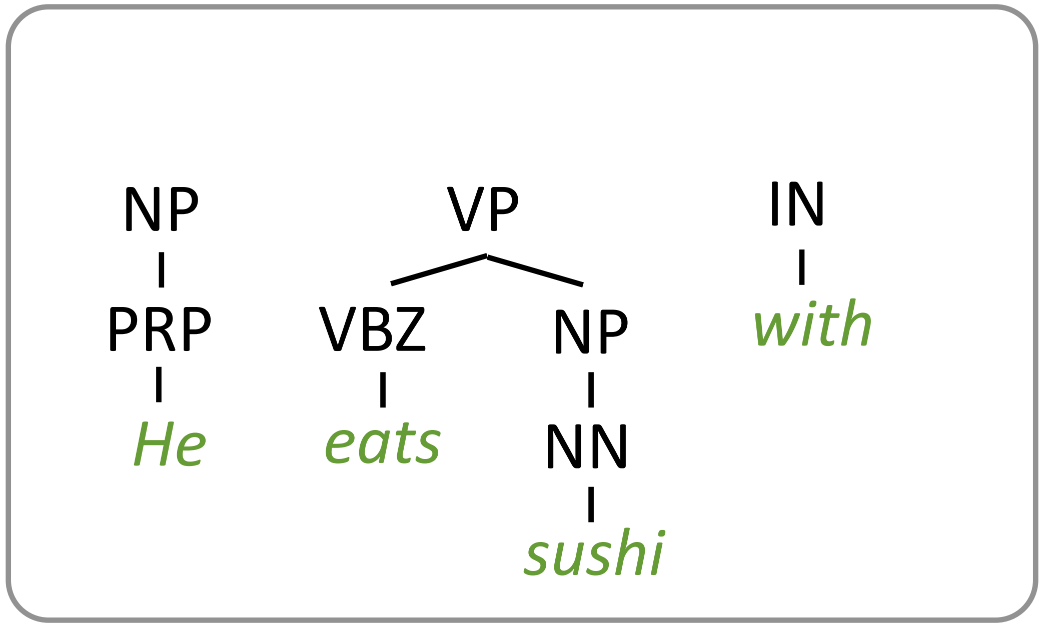


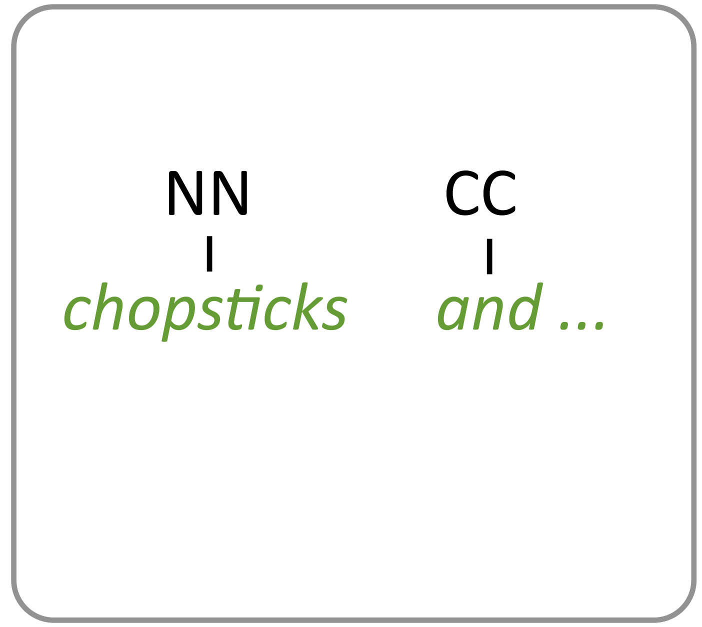


Advantages of Shift-reduce Parsing ?

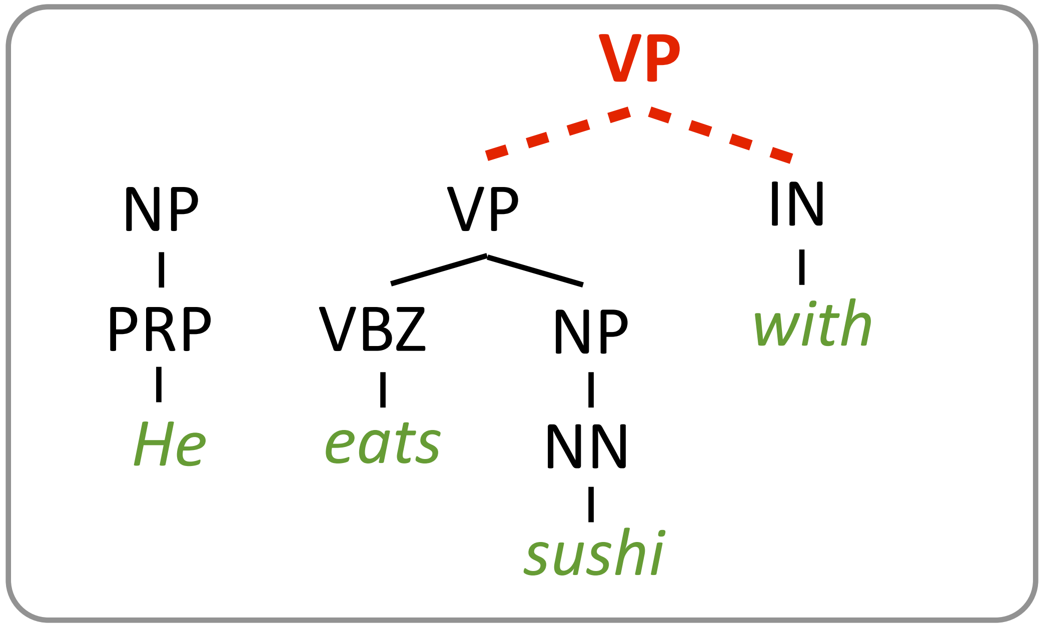


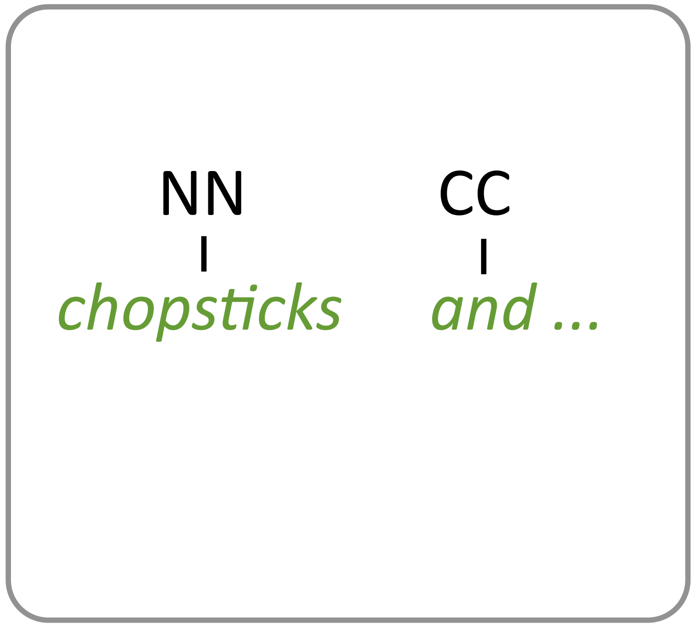
Advantages of Shift-reduce Parsing ?



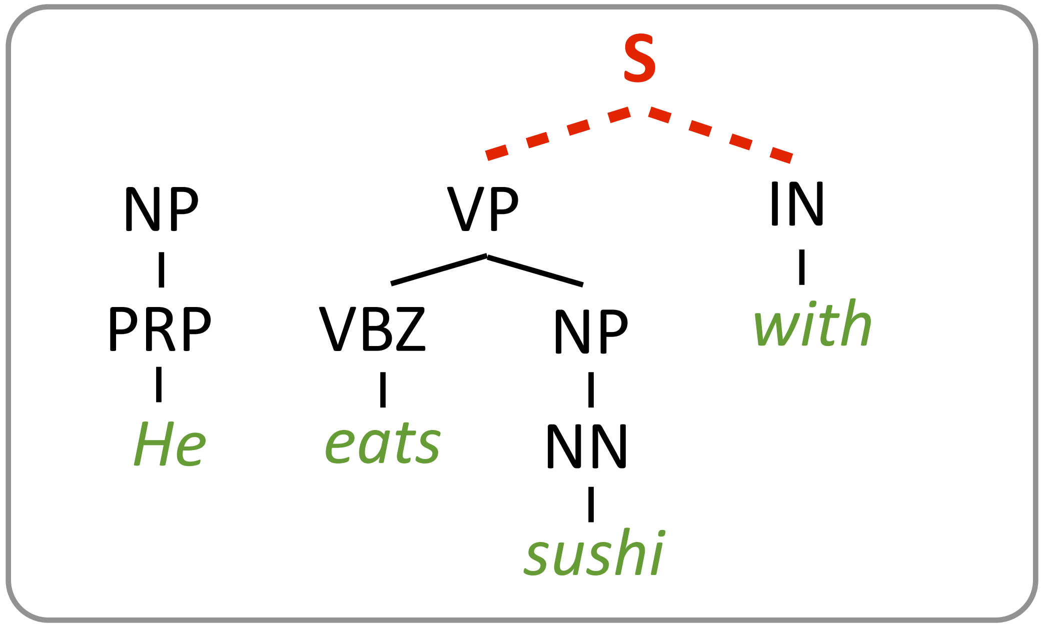


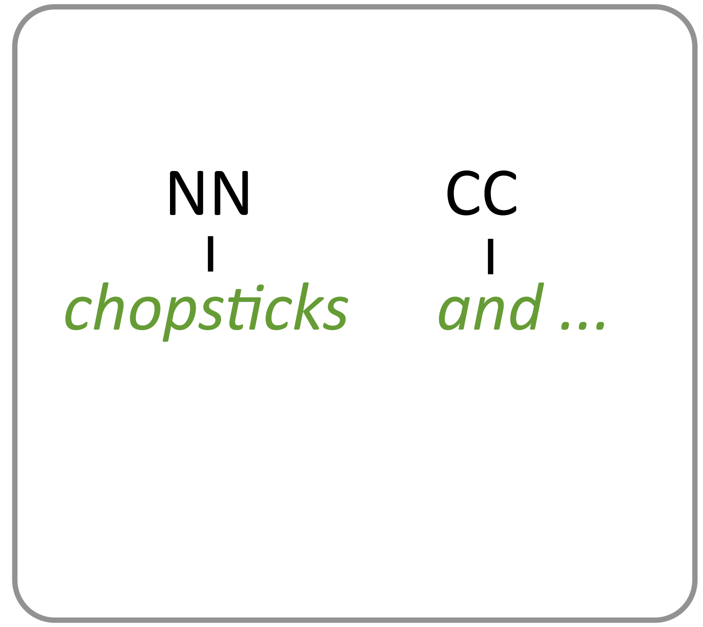
Advantages of Shift-reduce Parsing ?



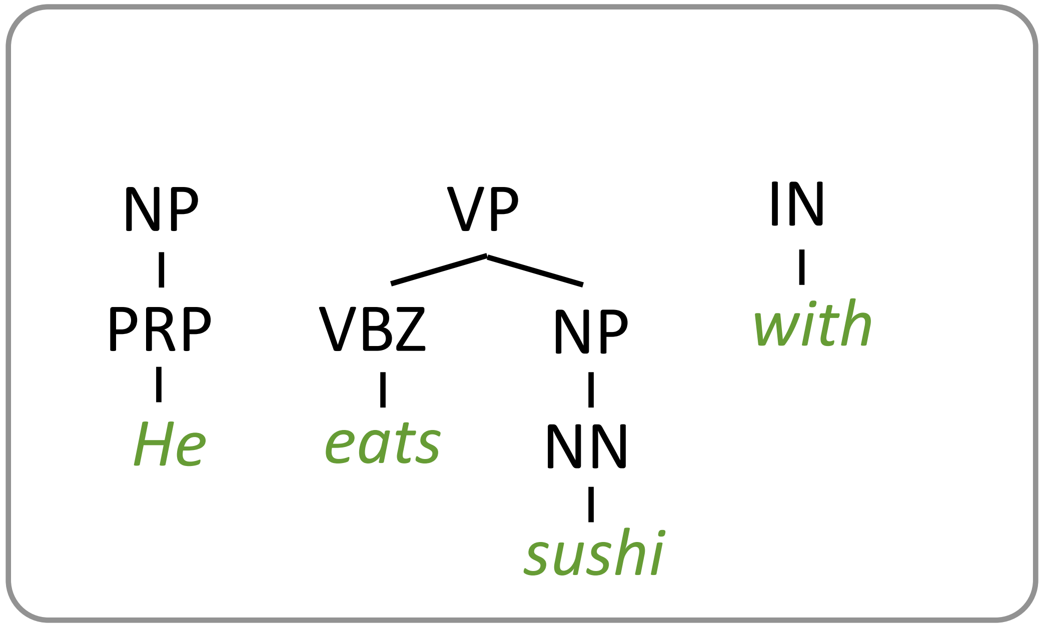


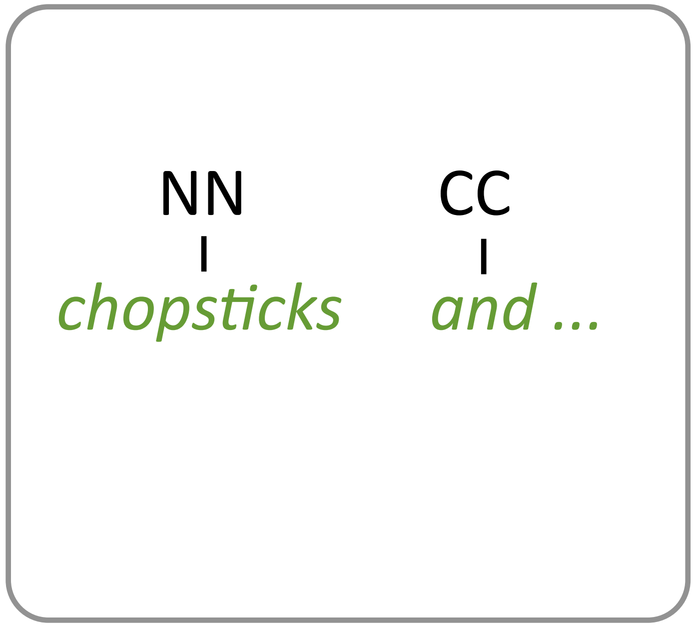
Advantages of Shift-reduce Parsing ?



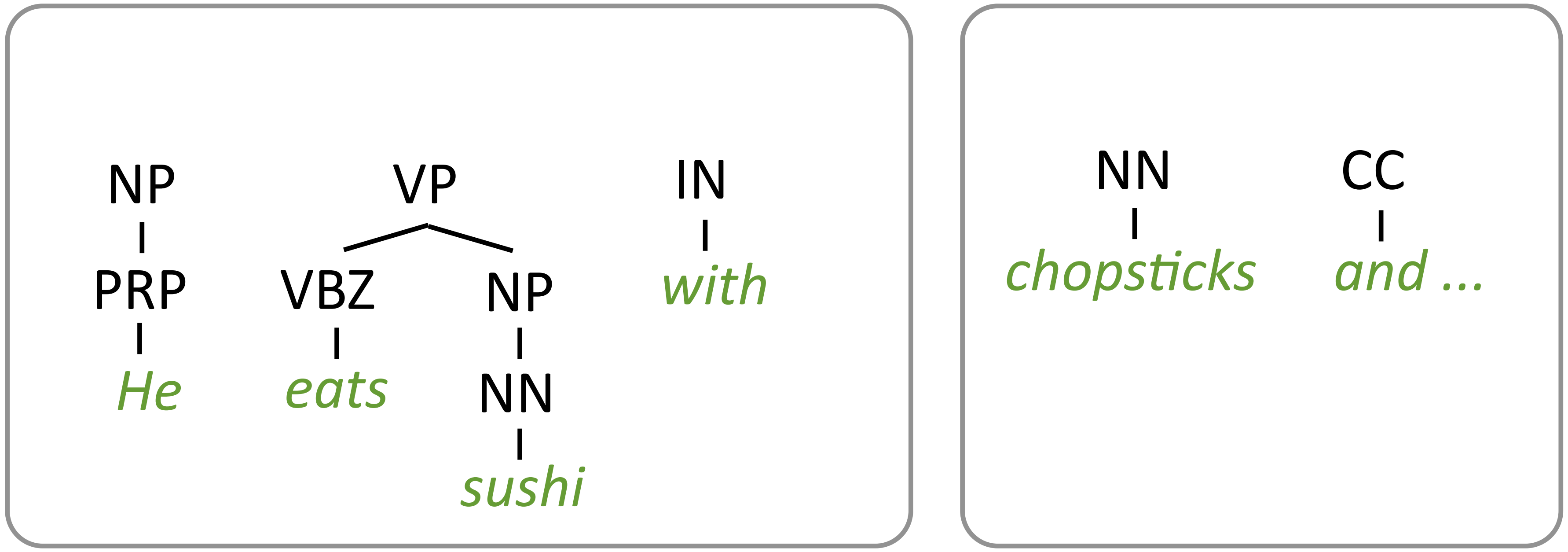


Advantages of Shift-reduce Parsing ?



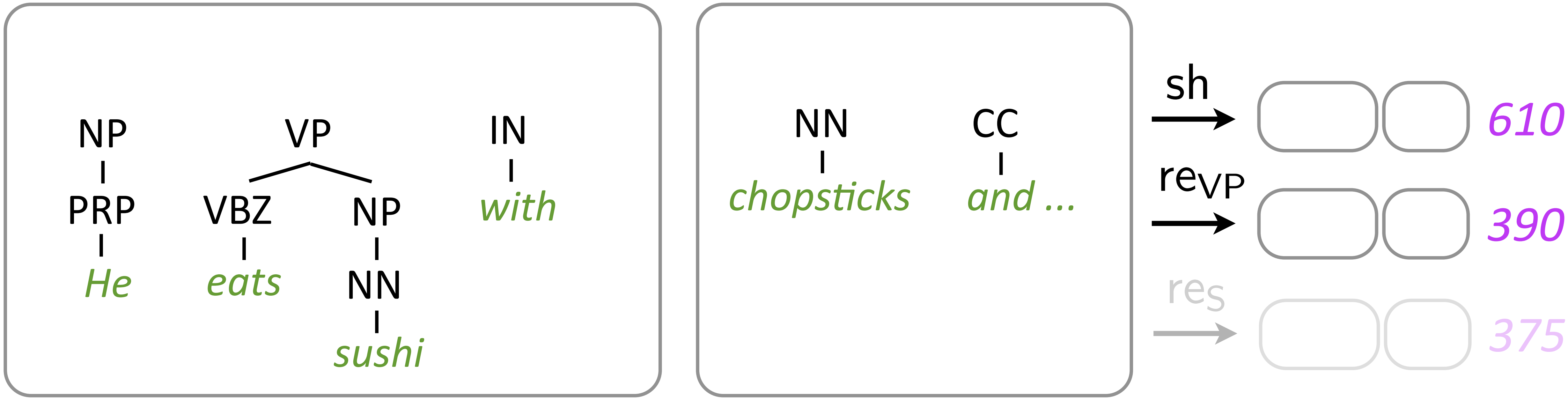


Advantages of Shift-reduce Parsing ?



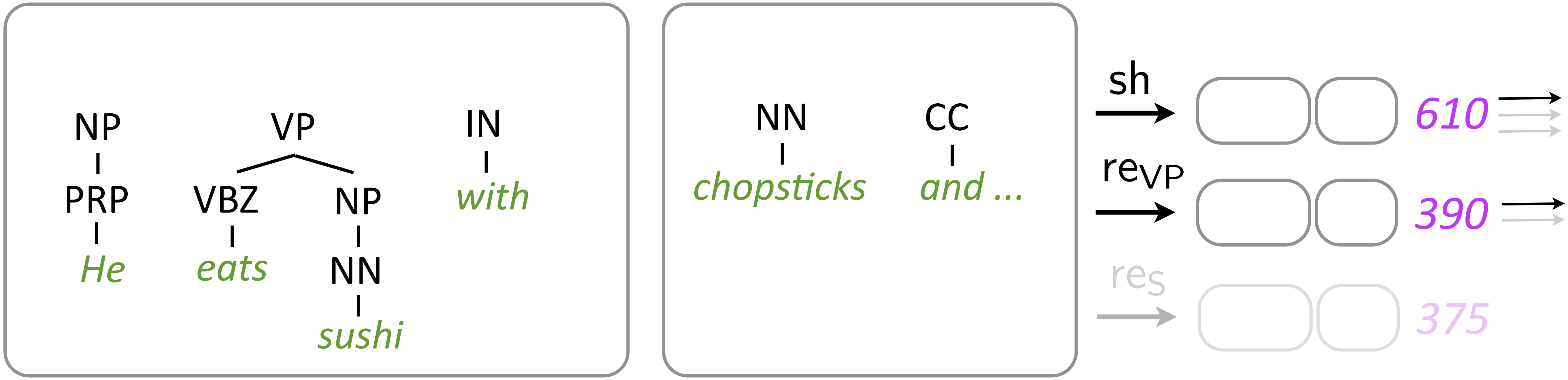
- Runtime is linear with beam search

Advantages of Shift-reduce Parsing ?



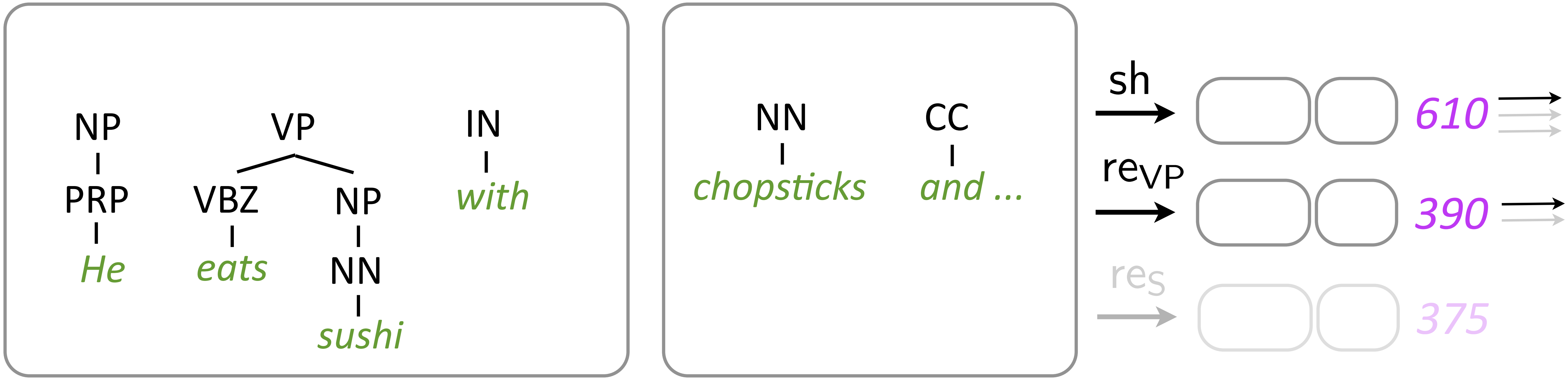
- Runtime is linear with beam search

Advantages of Shift-reduce Parsing ?



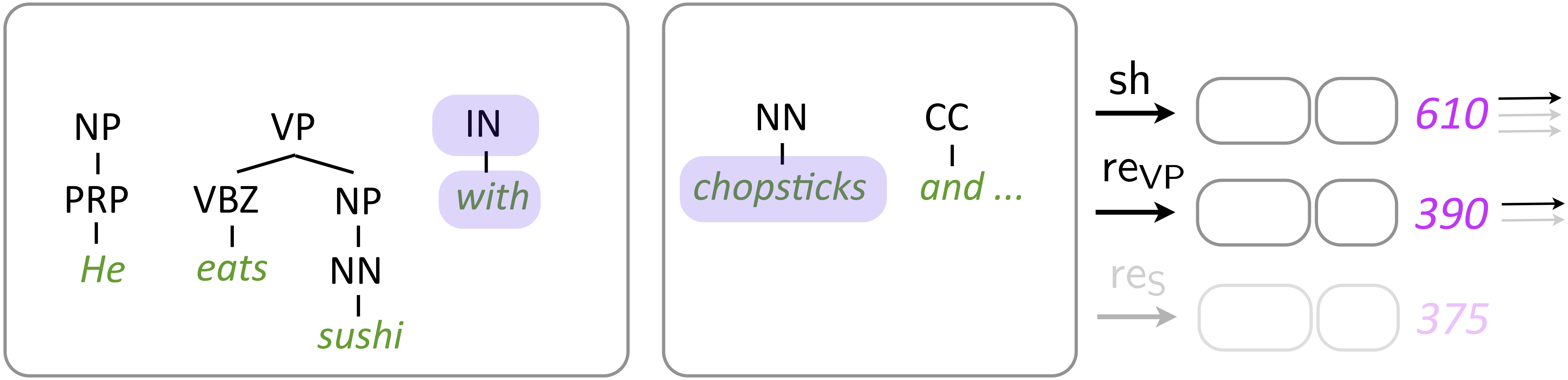
- Runtime is linear with beam search

Advantages of Shift-reduce Parsing ?



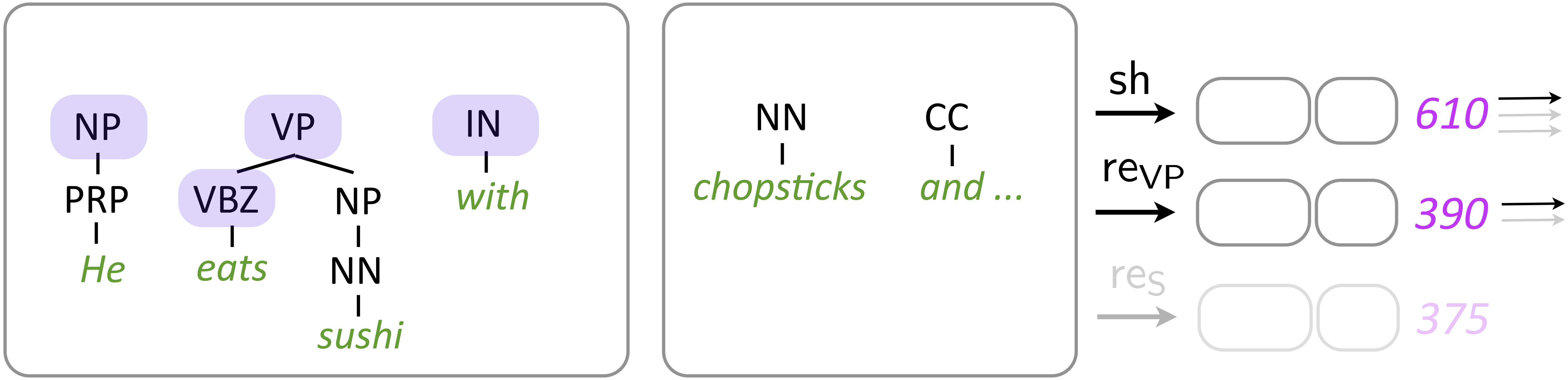
- Runtime is linear with beam search
- Arbitrary features can be exploited

Advantages of Shift-reduce Parsing ?



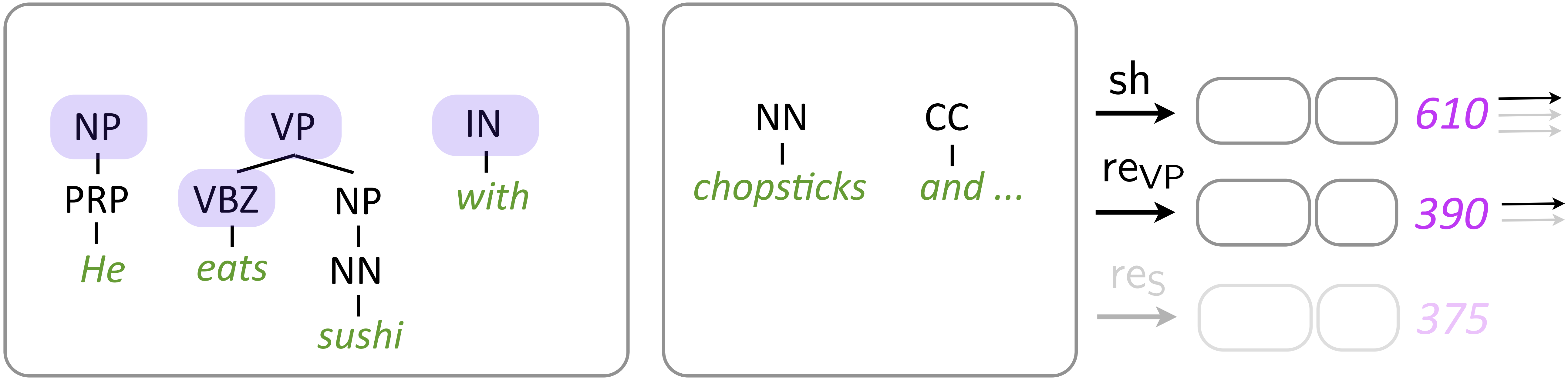
- Runtime is linear with beam search
- Arbitrary features can be exploited

Advantages of Shift-reduce Parsing ?



- Runtime is linear with beam search
- Arbitrary features can be exploited

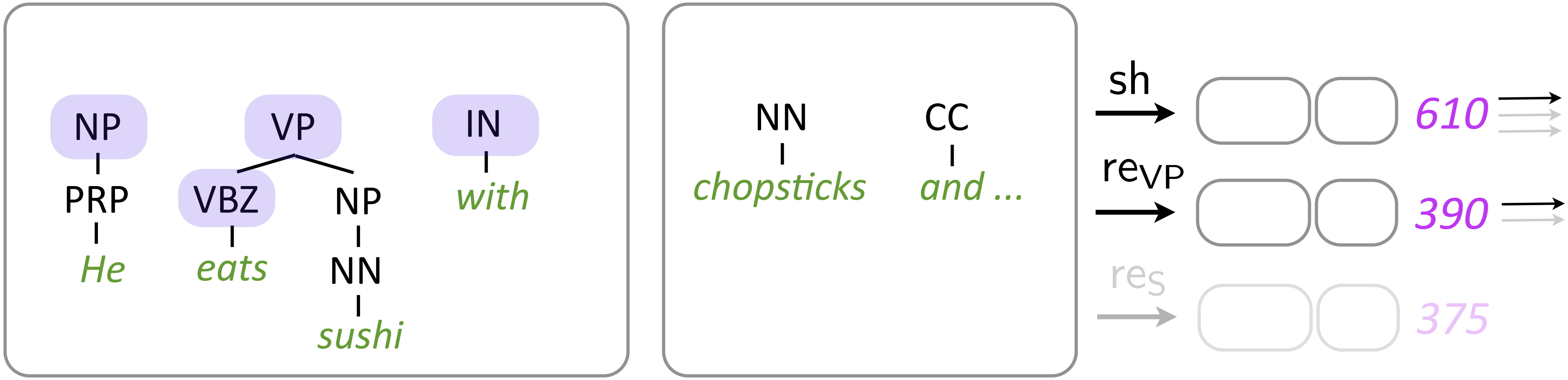
Advantages of Shift-reduce Parsing ?



- Runtime is linear with beam search
- Arbitrary features can be exploited

⇒ search errors exist, but practically rich feature helps more

Advantages of Shift-reduce Parsing ?



- Runtime is linear with beam search
- Arbitrary features can be exploited

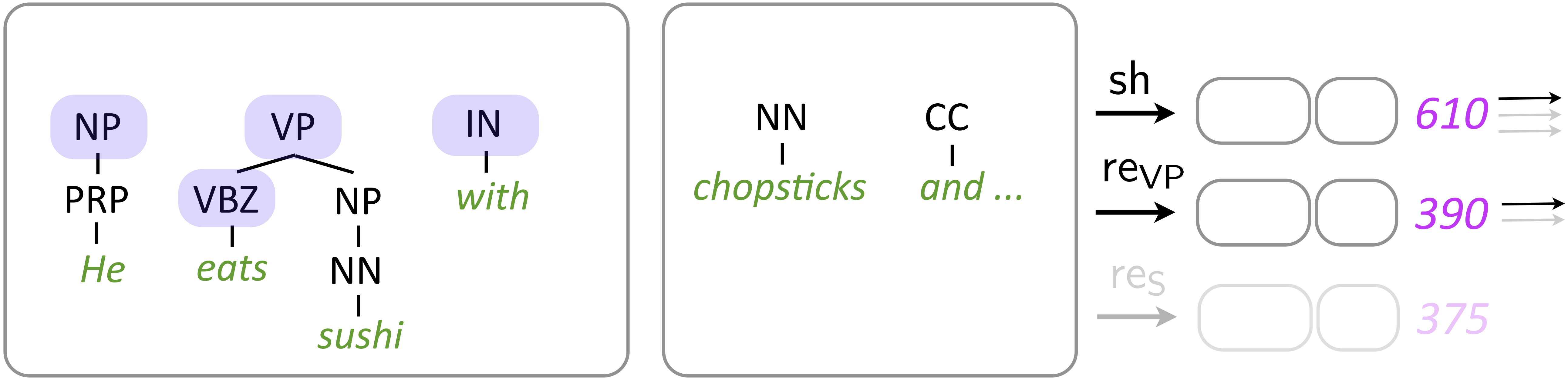
$\Rightarrow$ search errors exist, but practically rich feature helps more

Constituency: [Zhang & Clark, 2009; Zhu et al., 2013; Wang & Xue, 2014, Mi et al., 2015]

Dependency: [Huang & Sagae 2010; Zhang & Nivre, 2011; Bohnet et al., 2013]

CCG, etc: [Zhang & Clark, 2011; Xu et al., 2013; Ballesteros & Carreras, 2015]

Advantages of Shift-reduce Parsing ?



- Runtime is linear with beam search
- Arbitrary features can be exploited

⇒ search errors exist, but practically rich feature helps more

Constituency: [Zhang & Clark, 2009; Zhu et al., 2013; Wang & Xue, 2014, Mi et al., 2015]

Dependency: [Huang & Sagae 2010; Zhang & Nivre, 2011; Bohnet et al., 2013]

CCG, etc: [Zhang & Clark, 2011; Xu et al., 2013; Ballesteros & Carreras, 2015]

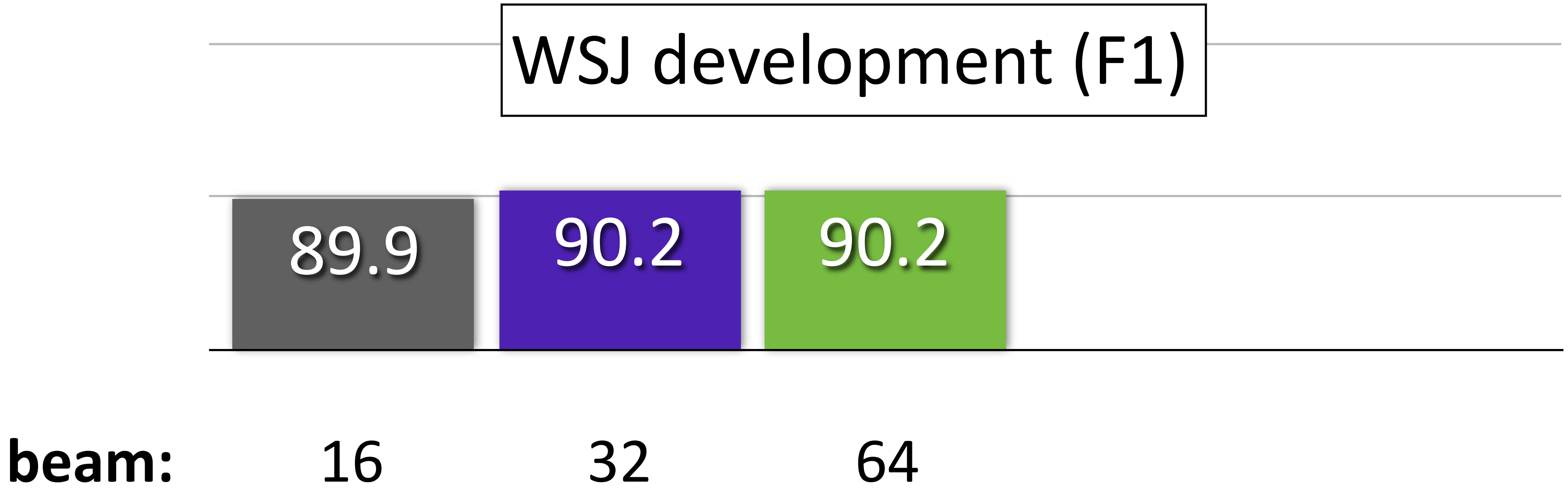
Question: Is this current design ... really the best?

What we have done

Shift-reduce constituent parser **with exact search**

What we have done

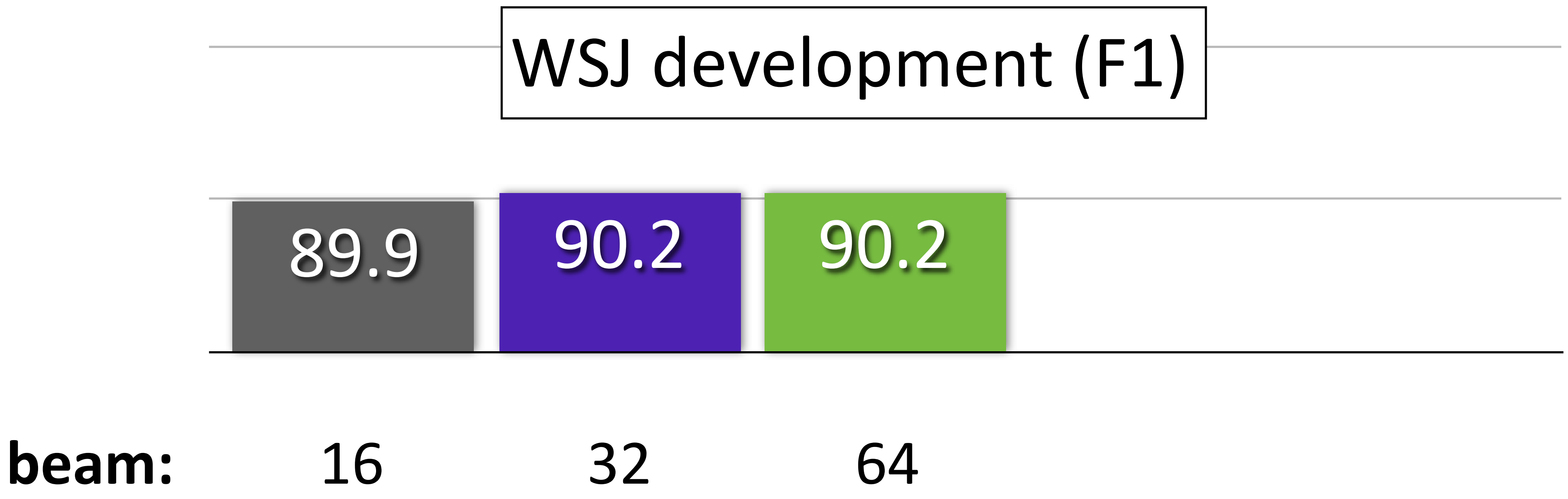
Shift-reduce constituent parser **with exact search**



What we have done

Shift-reduce constituent parser **with exact search**

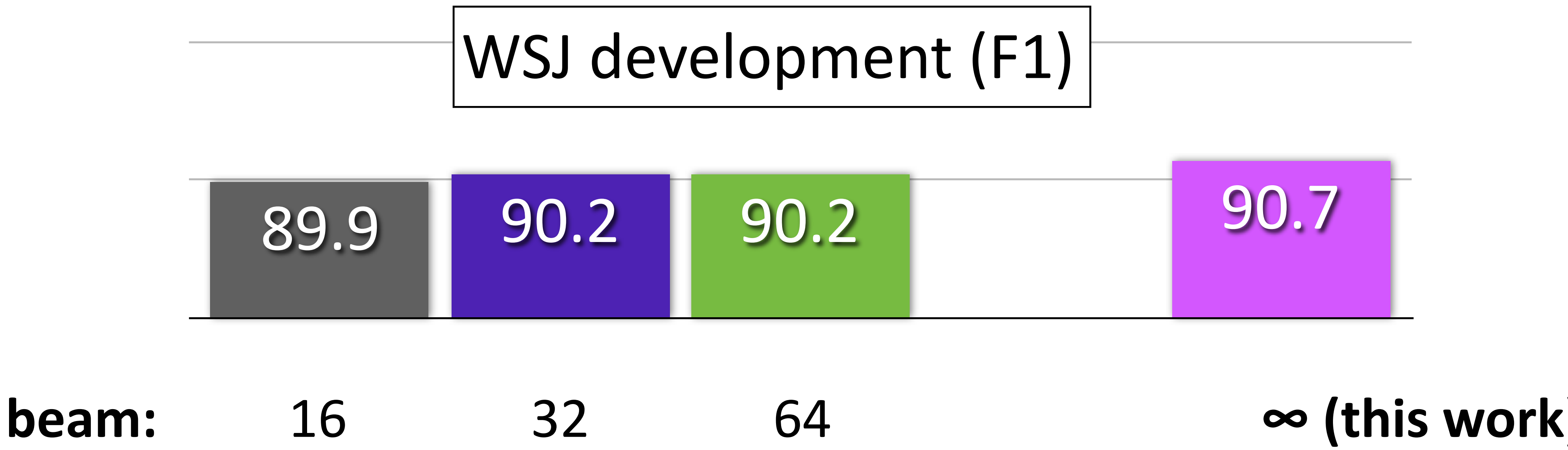
- state-of-the-art accuracy



What we have done

Shift-reduce constituent parser **with exact search**

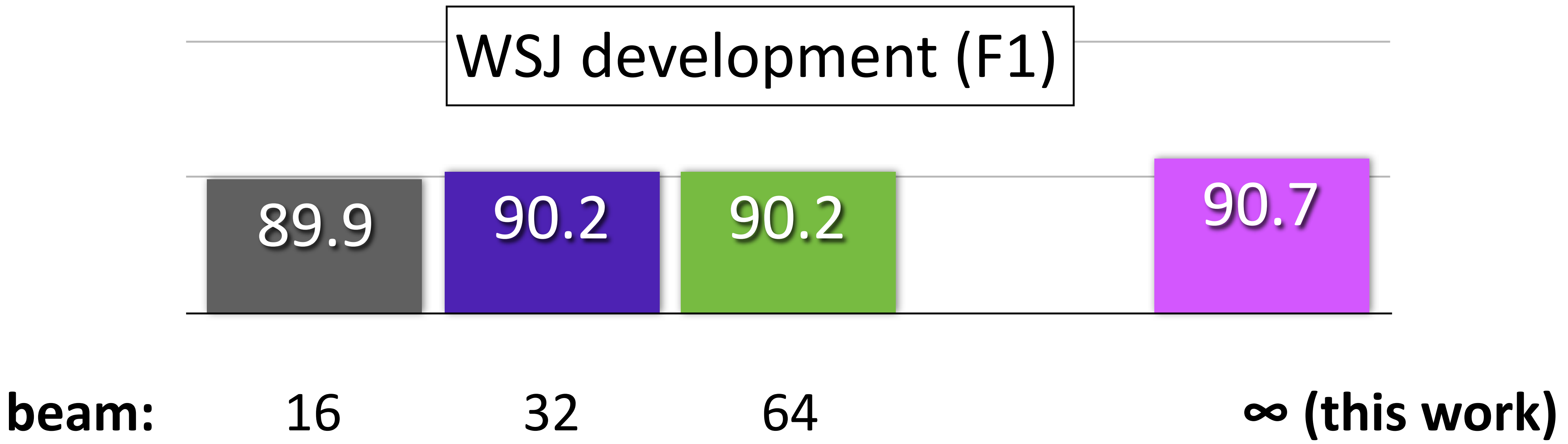
- state-of-the-art accuracy



What we have done

Shift-reduce constituent parser **with exact search**

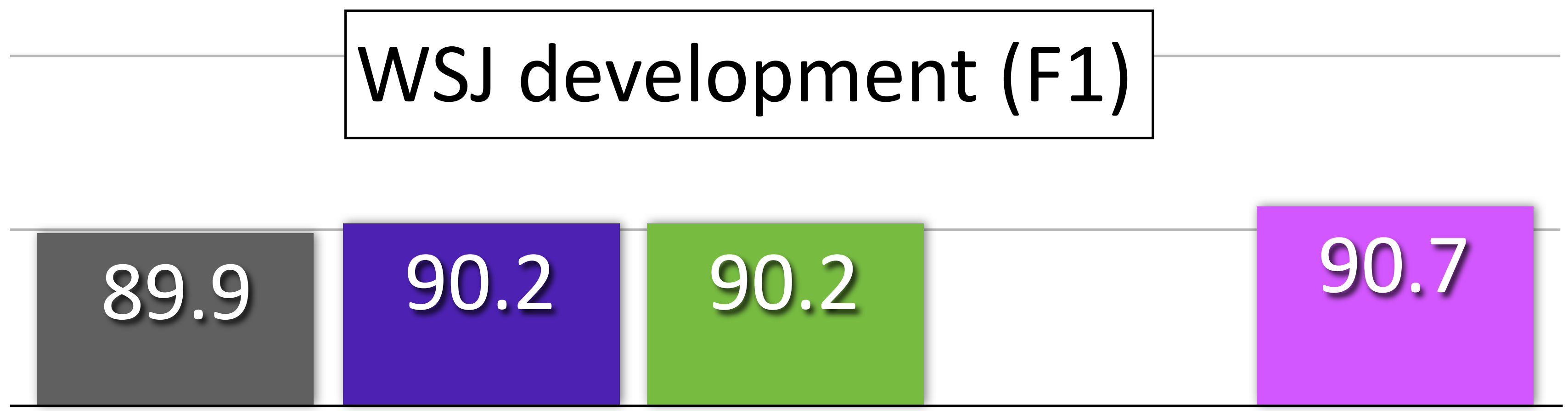
- state-of-the-art accuracy
- in a practical runtime



What we have done

Shift-reduce constituent parser **with exact search**

- state-of-the-art accuracy
- in a practical runtime

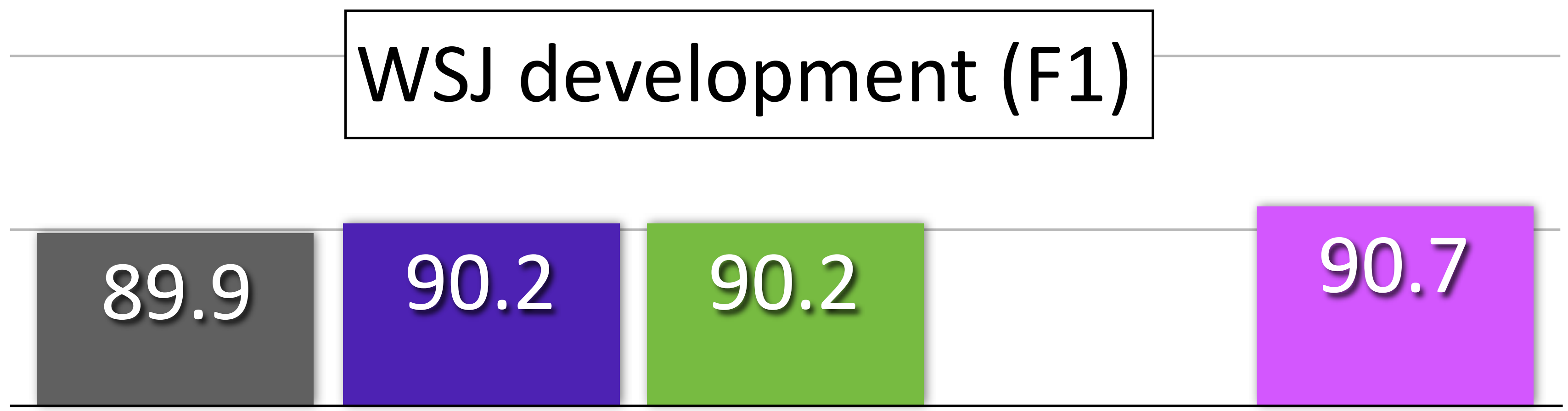


beam:	16	32	64	**∞ (this work)**
speed (Sent./s.):	31.9	17.0	9.1	

What we have done

Shift-reduce constituent parser **with exact search**

- state-of-the-art accuracy
- in a practical runtime

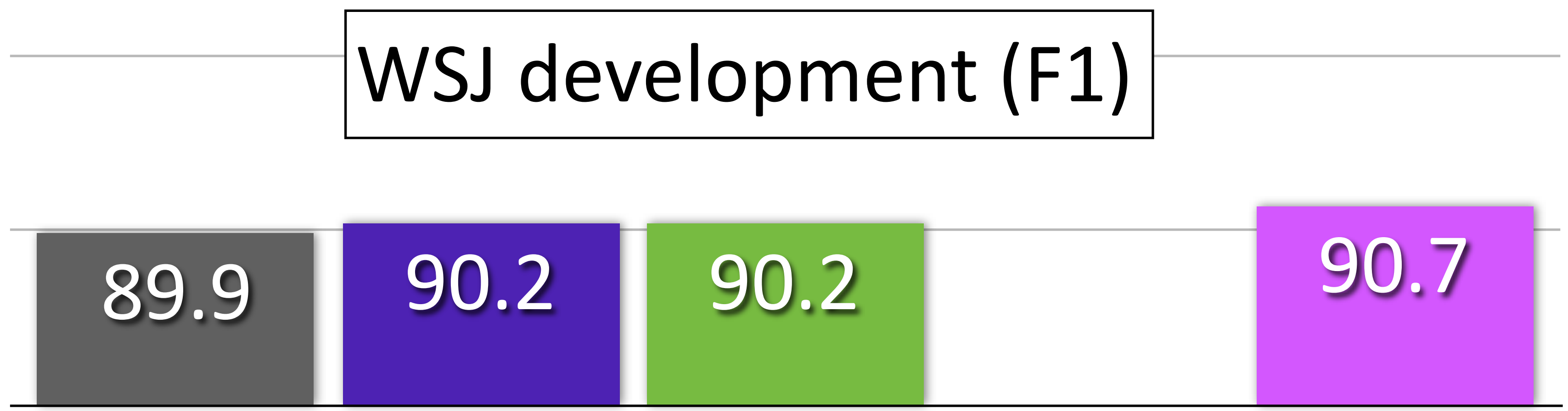


beam:	16	32	64	**∞ (this work)**
speed (Sent./s.):	31.9	17.0	9.1	13.6

What we have done

Shift-reduce constituent parser **with exact search**

- state-of-the-art accuracy
- in a practical runtime

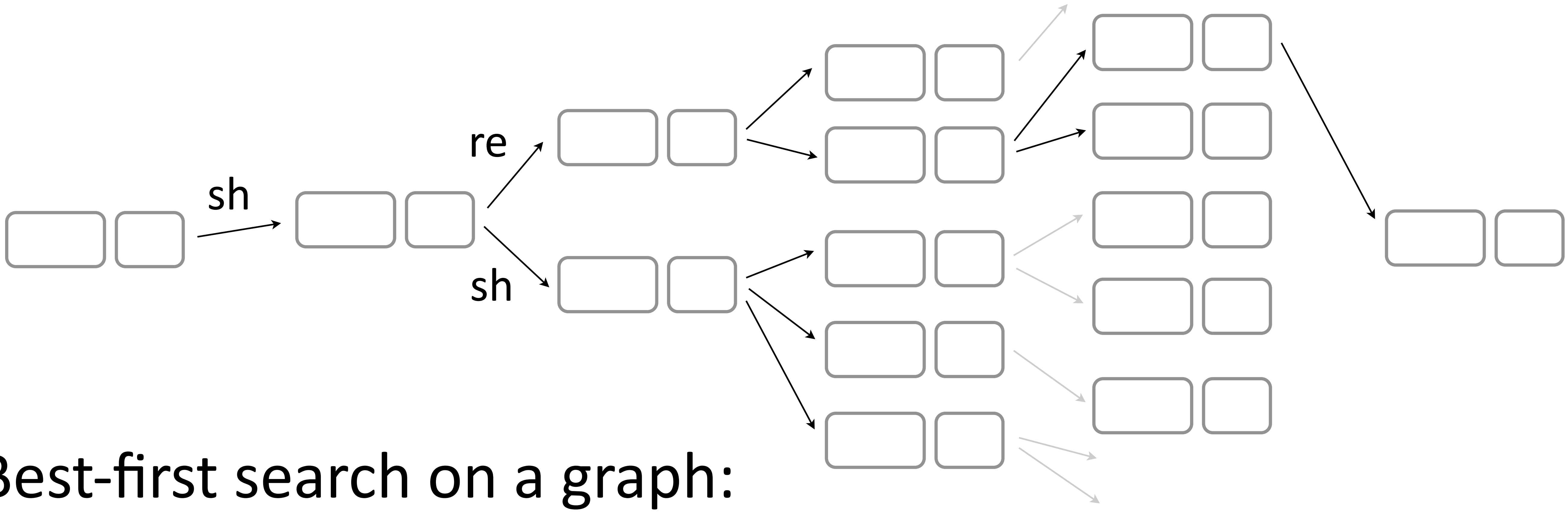


beam:	16	32	64	**∞ (this work)**
speed (Sent./s.):	31.9	17.0	9.1	13.6

Technical contributions:

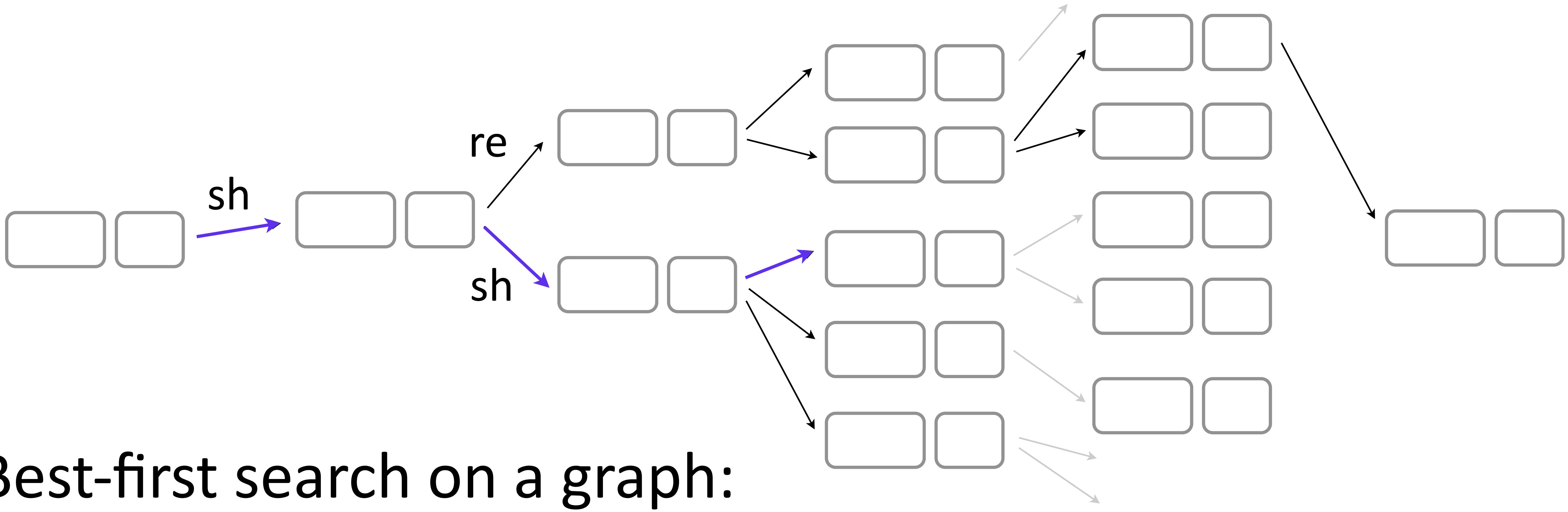
- New feature templates that facilitate search
- New A* heuristics to further speed up search

Framework



Best-first search on a graph:

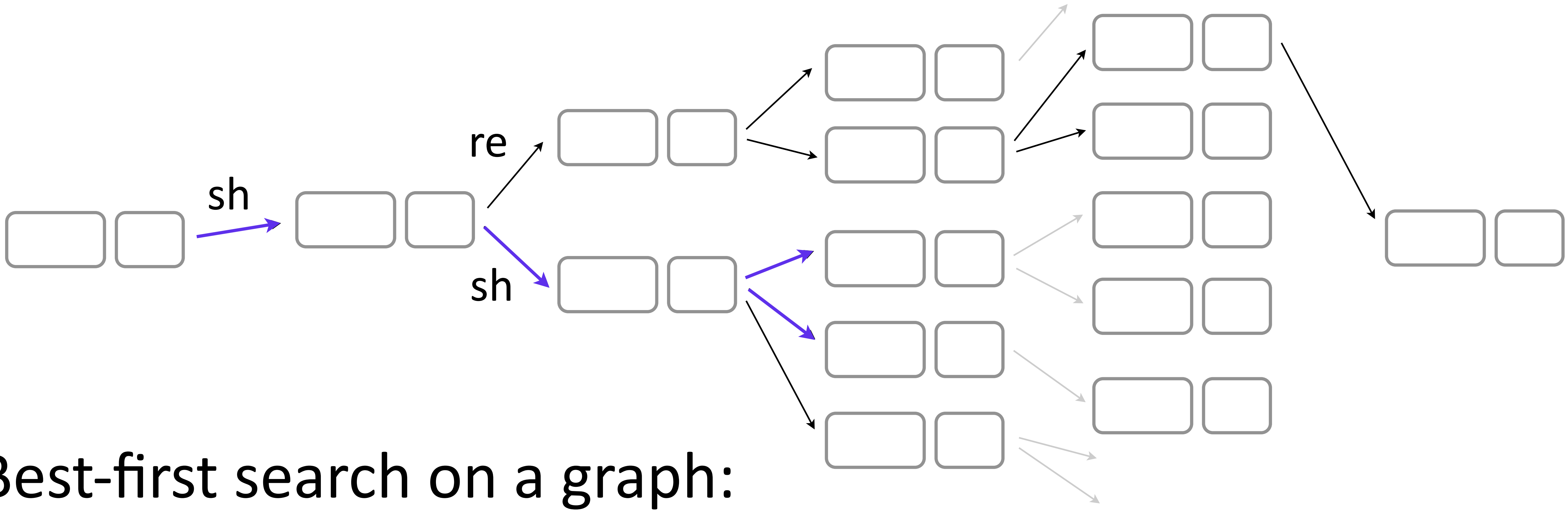
Framework



Best-first search on a graph:

Goal: Finding the lowest cost path

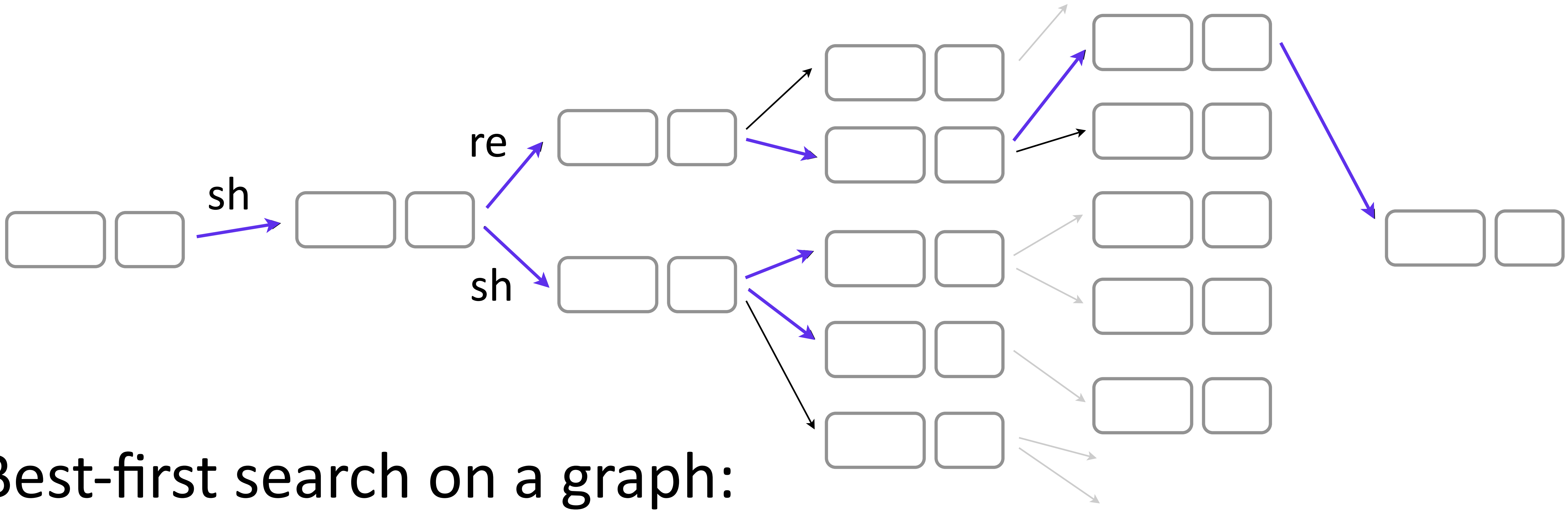
Framework



Best-first search on a graph:

Goal: Finding the lowest cost path

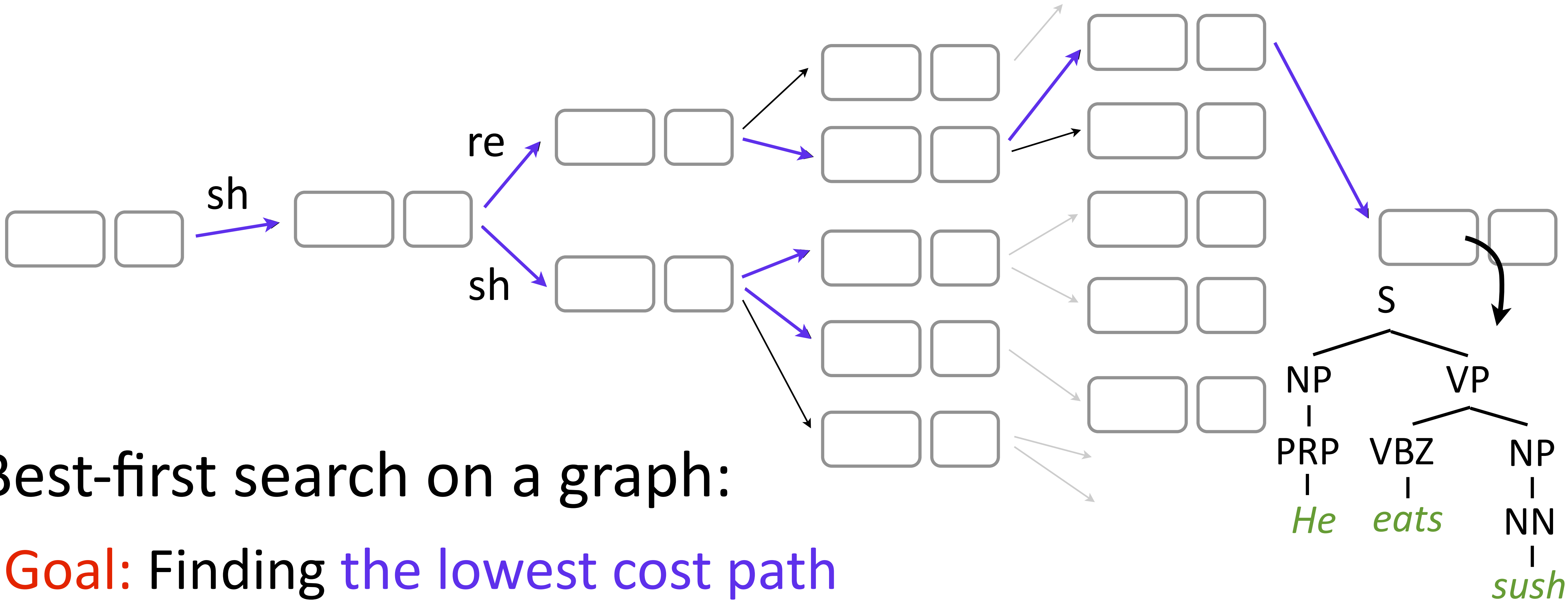
Framework



Best-first search on a graph:

Goal: Finding the lowest cost path

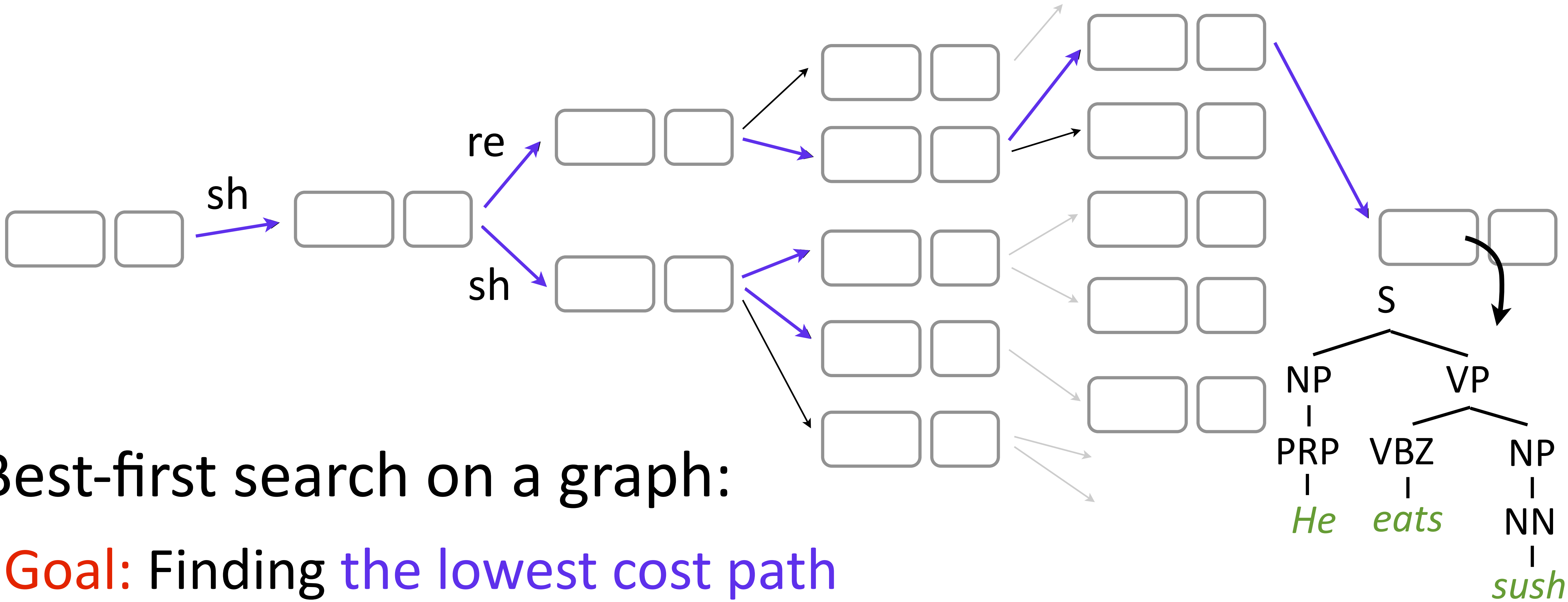
Framework



Best-first search on a graph:

Goal: Finding the lowest cost path

Framework

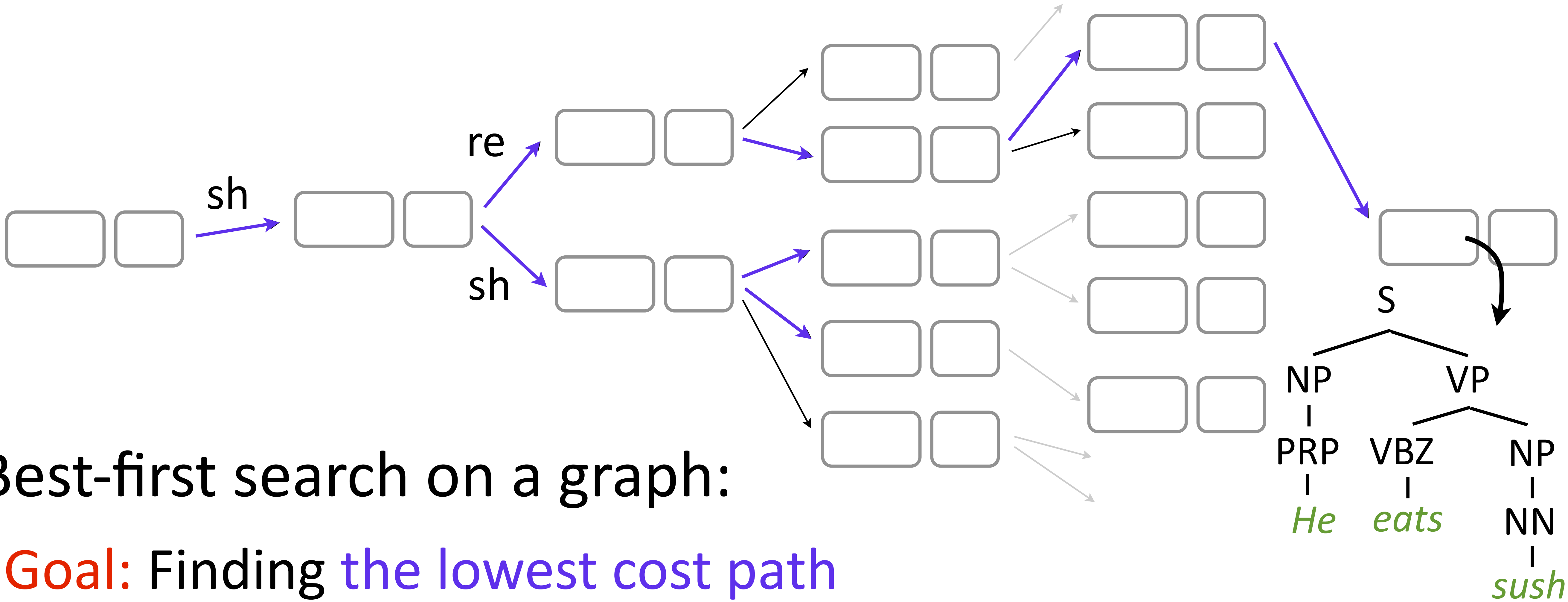


Best-first search on a graph:

Goal: Finding the lowest cost path

- The first found derivation is optimal

Framework



Best-first search on a graph:

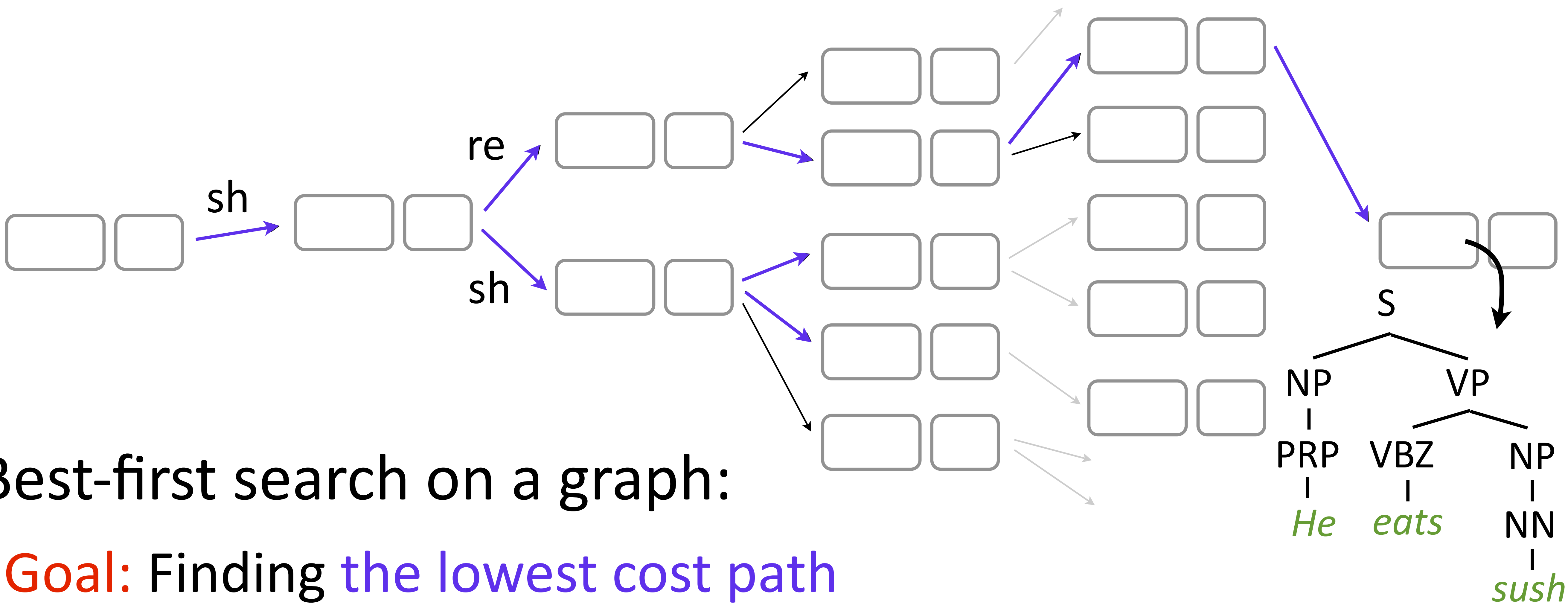
Goal: Finding the lowest cost path

- The first found derivation is optimal

Model: Structured perceptron

- State-of-the-art model for shift-reduce parsing [Zhu et al., 2013; Wang & Xue, 2014; Mi et al., 2015]

Framework



Best-first search on a graph:

Goal: Finding the lowest cost path

- The first found derivation is optimal

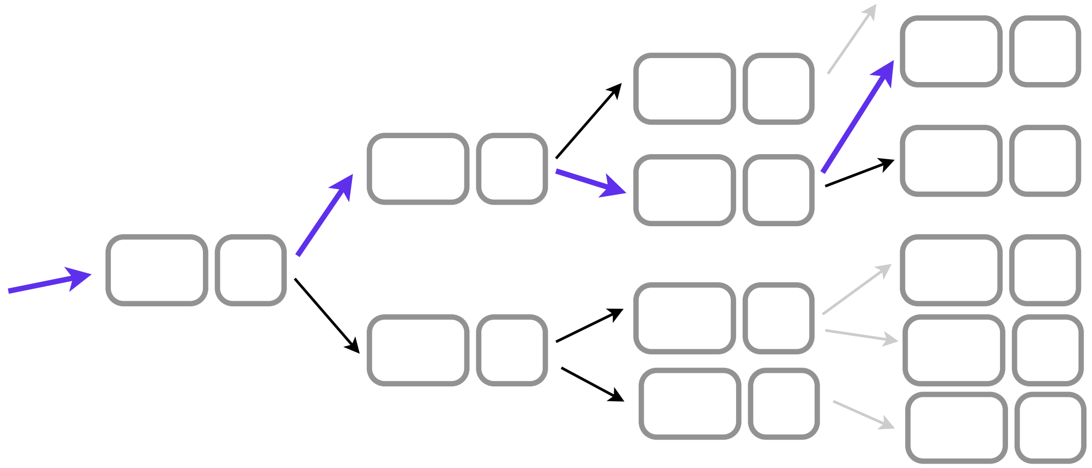
Model: Structured perceptron

- State-of-the-art model for shift-reduce parsing
  [Zhu et al., 2013; Wang & Xue, 2014; Mi et al., 2015]
- but all previous parsers rely on beam-search

Previous best-first shift-reduce parsing

- Initial work [Sagae & Lavie, 2006]

  Search on exponentially large graph



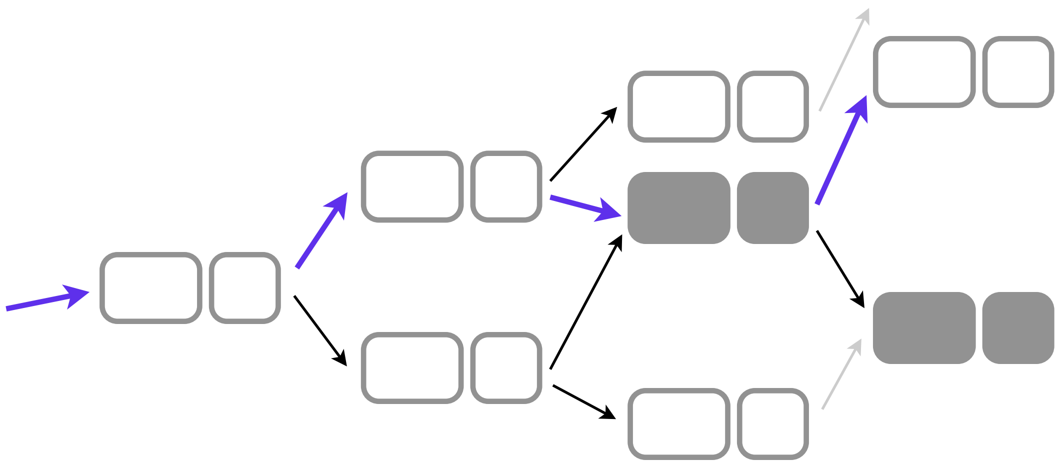
Previous best-first shift-reduce parsing

- Initial work [Sagae & Lavie, 2006]

  Search on exponentially large graph

- Improving search with dynamic programming [Zhao et al., 2013]

  Graph size is polynomial by state merging [Huang & Sagae, 2010]



Previous best-first shift-reduce parsing

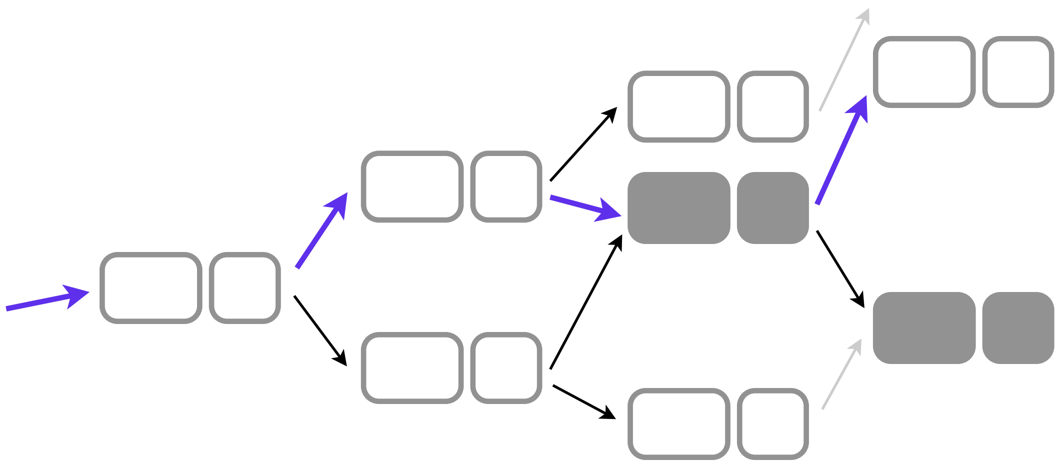
- Initial work [Sagae & Lavie, 2006]

  Search on exponentially large graph

- Improving search with dynamic programming [Zhao et al., 2013]

  Graph size is polynomial by state merging [Huang & Sagae, 2010]

Limitation of previous works:



- Accuracy is **not state-of-the-art**
  - because the model is MaxEnt
  - MaxEnt does not judge the quality of overall action path

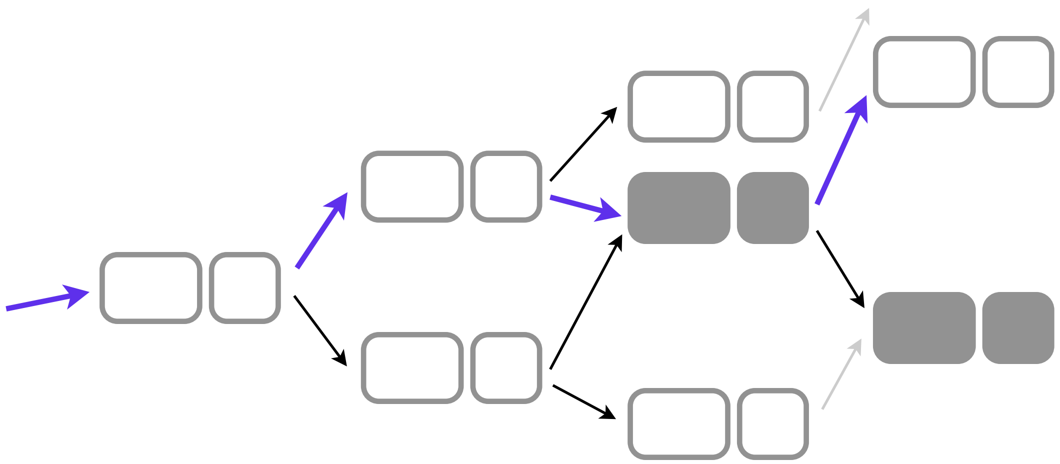
Previous best-first shift-reduce parsing

- Initial work [Sagae & Lavie, 2006]

  Search on exponentially large graph

- Improving search with dynamic programming [Zhao et al., 2013]

  Graph size is polynomial by state merging [Huang & Sagae, 2010]



Limitation of previous works:

- Accuracy is **not state-of-the-art**
  - because the model is MaxEnt
  - MaxEnt does not judge the quality of overall action path
- Dynamic programming has only been applied to **dependency**

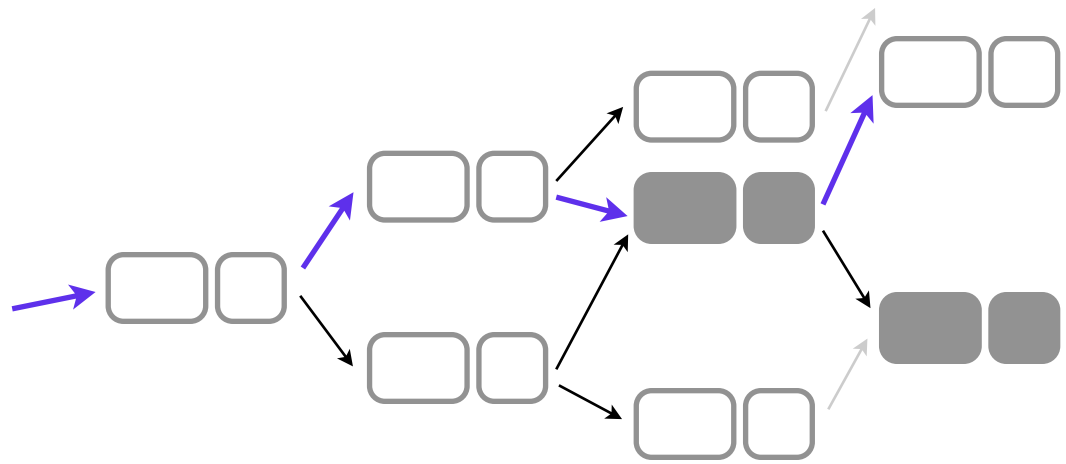
Previous best-first shift-reduce parsing

- Initial work [Sagae & Lavie, 2006]

  Search on exponentially large graph

- Improving search with dynamic programming [Zhao et al., 2013]

  Graph size is polynomial by state merging [Huang & Sagae, 2010]



Limitation of previous works:

- Accuracy is **not state-of-the-art**
  - because the model is MaxEnt
  - MaxEnt does not judge the quality of overall action path
- Dynamic programming has only been applied to **dependency**

We explore DP best-first shift-reduce parsing for **constituency** with structured perceptron

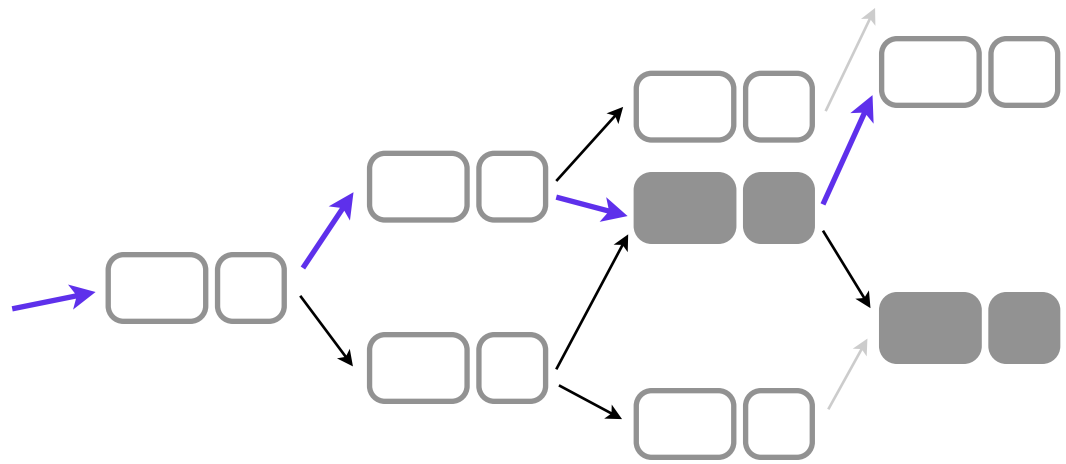
Previous best-first shift-reduce parsing

- Initial work [Sagae & Lavie, 2006]

  Search on exponentially large graph

- Improving search with dynamic programming [Zhao et al., 2013]

  Graph size is polynomial by state merging [Huang & Sagae, 2010]



Limitation of previous works:

- Accuracy is **not state-of-the-art**
  - because the model is MaxEnt
  - MaxEnt does not judge the quality of overall action path
- Dynamic programming has only been applied to **dependency**

We explore DP best-first shift-reduce parsing for **constituency** with structured perceptron

Challenge: Search gets much harder with structured perceptron

Outline

DP best-first shift-reduce parsing

- for **constituency**
- with **structured perceptron**

MaxEnt vs. Structured perceptron

- Structured perceptron is strong
- but its search is much harder

Improving search efficiency of structured perceptron

- New feature templates
- A* search

Final experiment

Outline

DP best-first shift-reduce parsing

- for **constituency**
- with **structured perceptron**

MaxEnt vs. Structured perceptron

- Structured perceptron is strong
- but its search is much harder

Improving search efficiency of structured perceptron

- New feature templates
- A* search

Final experiment

DP best-first parsing for constituency

Basic algorithm: Zhao et al (2013)'s DP best-first search

DP best-first parsing for constituency

Basic algorithm: Zhao et al (2013)'s DP best-first search

How to apply for constituent parsing?

DP best-first parsing for constituency

Basic algorithm: Zhao et al (2013)'s DP best-first search

How to apply for constituent parsing?

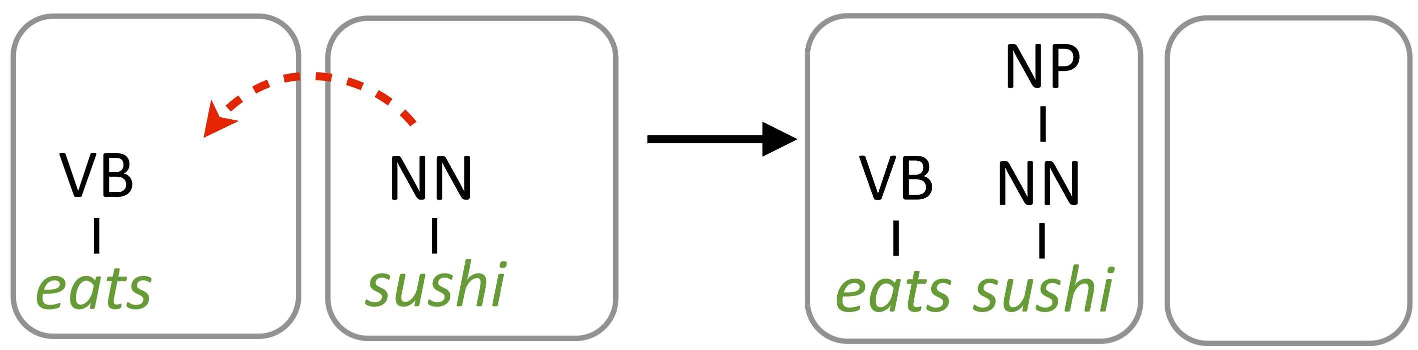
- The main difficulty with constituent parsing is unary rule
- We develop a transition system without unary rules

DP best-first parsing for constituency

Basic algorithm: Zhao et al (2013)'s DP best-first search

How to apply for constituent parsing?

- The main difficulty with constituent parsing is unary rule
- We develop a transition system without unary rules

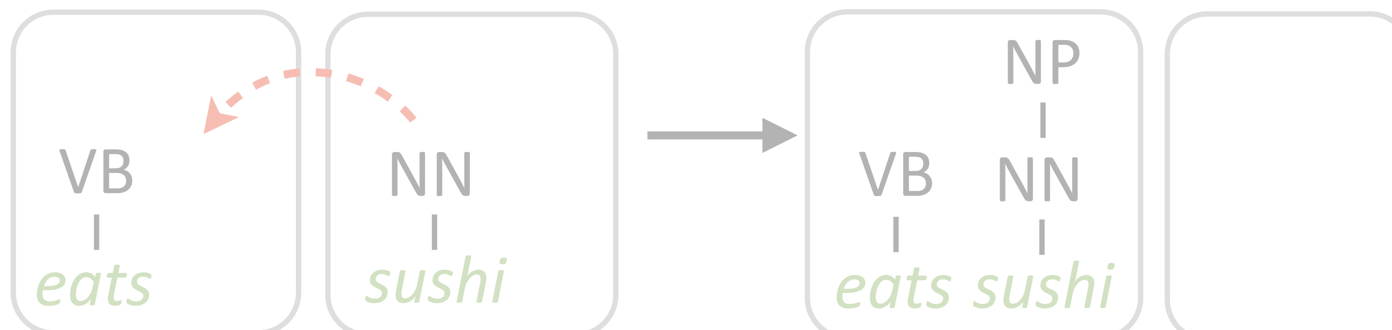


DP best-first parsing for constituency

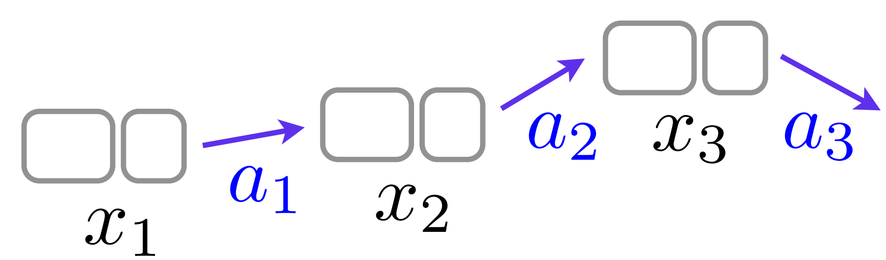
Basic algorithm: Zhao et al (2013)'s DP best-first search

How to apply for constituent parsing?

- The main difficulty with constituent parsing is unary rule
- We develop a transition system without unary rules



What is edge cost in structured perceptron?

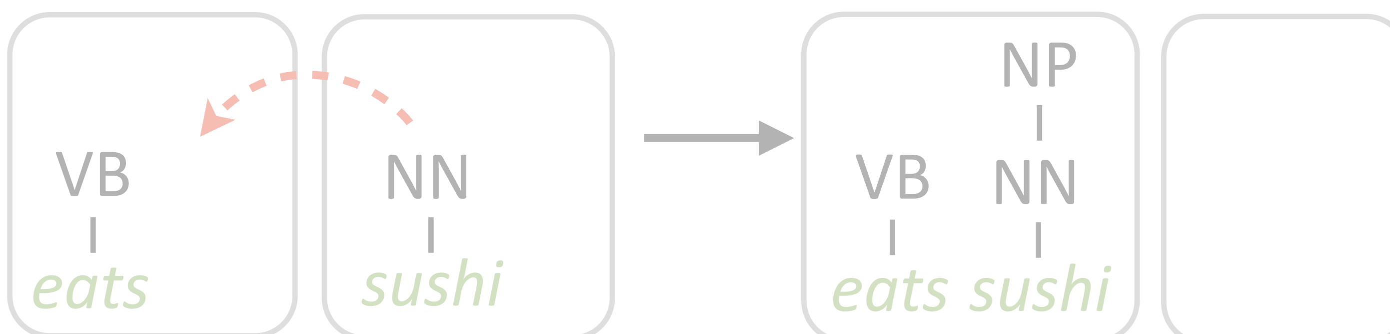


DP best-first parsing for constituency

Basic algorithm: Zhao et al (2013)'s DP best-first search

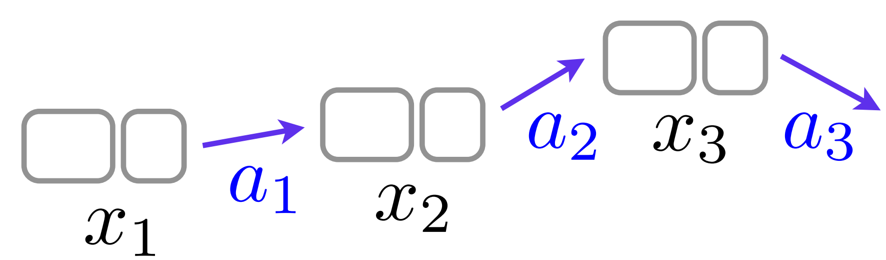
How to apply for constituent parsing?

- The main difficulty with constituent parsing is unary rule
- We develop a transition system without unary rules



What is edge cost in structured perceptron?

Constraint for optimality: Every cost must be positive

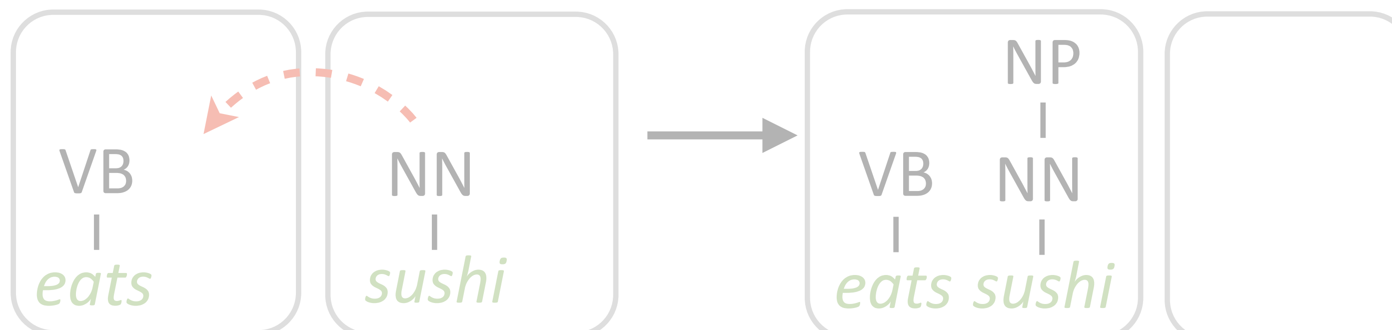


DP best-first parsing for constituency

Basic algorithm: Zhao et al (2013)'s DP best-first search

How to apply for constituent parsing?

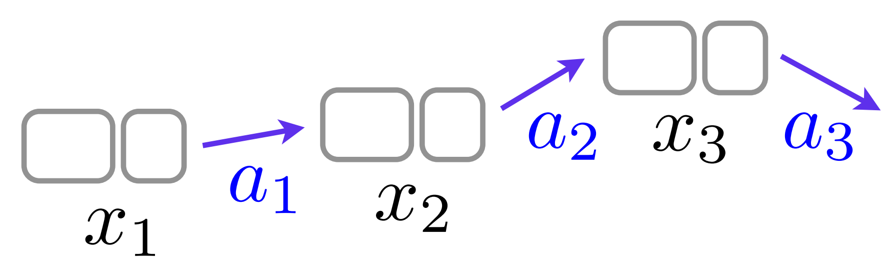
- The main difficulty with constituent parsing is unary rule
- We develop a transition system without unary rules



What is edge cost in structured perceptron?

Constraint for optimality: Every cost must be positive

Score to a derivation: $\sum_{i=1}^{2n} \theta^{\intercal} f(a_i, x_i)$

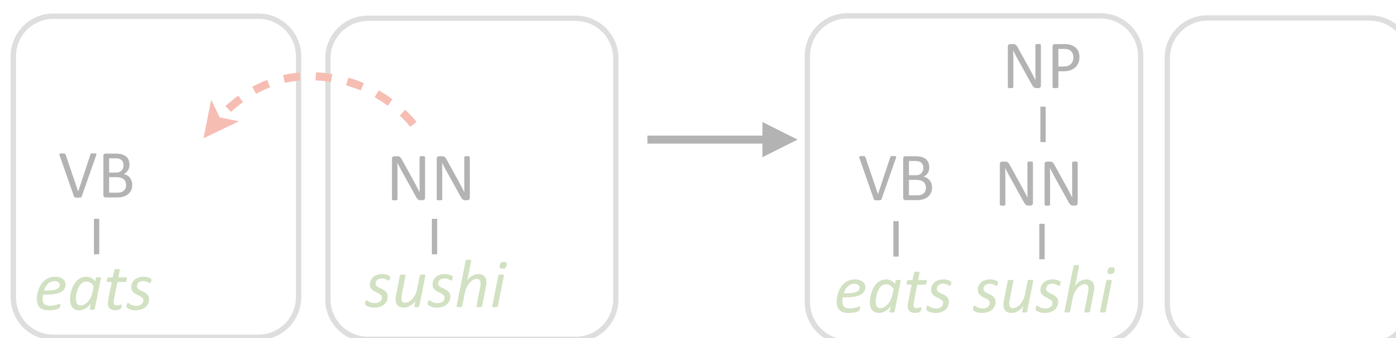


DP best-first parsing for constituency

Basic algorithm: Zhao et al (2013)'s DP best-first search

How to apply for constituent parsing?

- The main difficulty with constituent parsing is unary rule
- We develop a transition system without unary rules

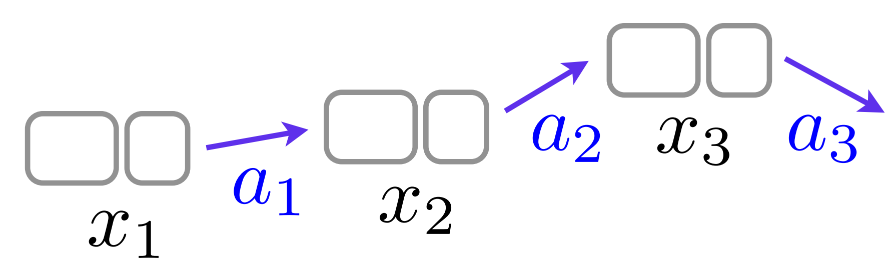


What is edge cost in structured perceptron?

Constraint for optimality: Every cost must be positive

Score to a derivation: $\sum_{i=1}^{2n} \theta^{\mathsf{T}} f(a_i, x_i)$

Our edge cost: $\theta^{\mathsf{T}} f(a, x) + \delta$

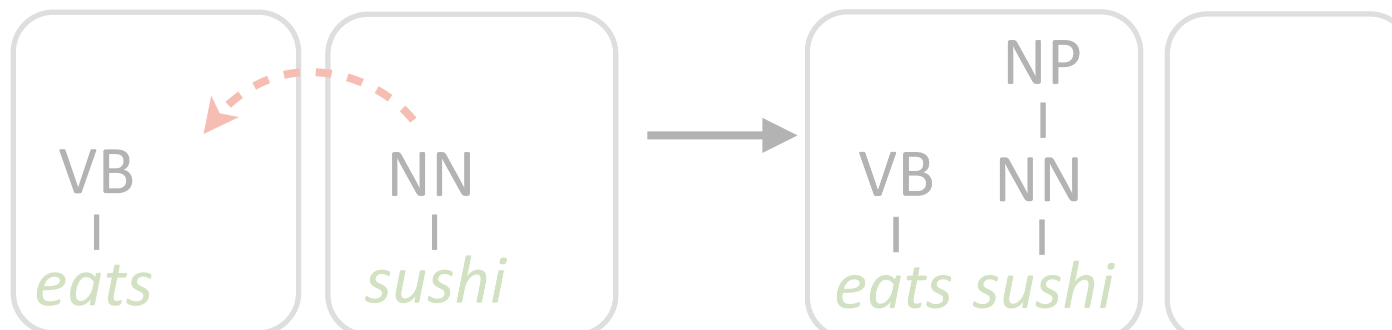


DP best-first parsing for constituency

Basic algorithm: Zhao et al (2013)'s DP best-first search

How to apply for constituent parsing?

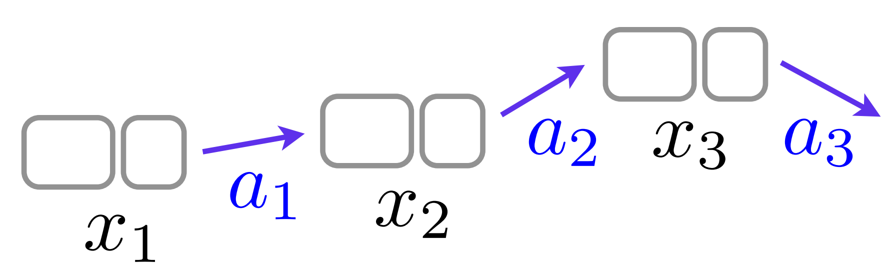
- The main difficulty with constituent parsing is unary rule
- We develop a transition system without unary rules



What is edge cost in structured perceptron?

Constraint for optimality: Every cost must be positive

Score to a derivation: $\sum_{i=1}^{2n} \theta^{\intercal} f(a_i, x_i)$



Our edge cost: $\theta^{\intercal} f(a, x) + \delta$

could be negative

DP best-first parsing for constituency

Basic algorithm: Zhao et al (2013)'s DP best-first search

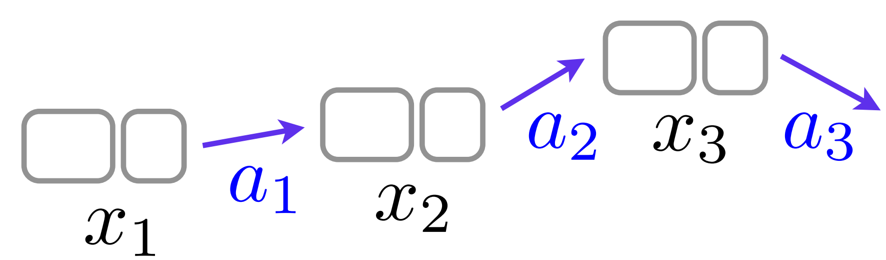
How to apply for constituent parsing?

- The main difficulty with constituent parsing is unary rule
- We develop a transition system without unary rules

What is edge cost in structured perceptron?

Constraint for optimality: Every cost must be positive

Score to a derivation: $\sum_{i=1}^{2n} \theta^\intercal f(a_i, x_i)$



Our edge cost: $\theta^\intercal f(a, x) + \delta$

could be negative

Constant offset to prevent negative

Outline

DP best-first shift-reduce parsing

- for **constituency**
- with **structured perceptron**

MaxEnt vs. Structured perceptron

- Structured perceptron is strong
- but its search is much harder

Improving search efficiency of structured perceptron

- New feature templates
- A* search

Final experiment

MaxEnt. vs. Structured Perceptron

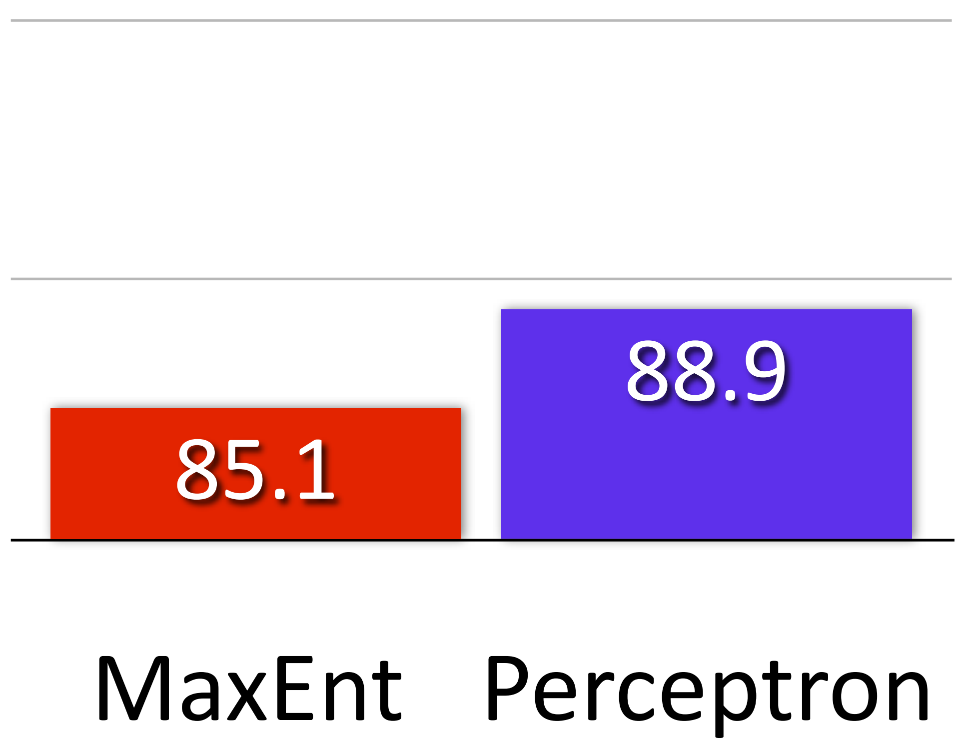
- Data: WSJ Section 22
- Relatively simple features

F1 score **Search efficiency**

MaxEnt. vs. Structured Perceptron

- Data: WSJ Section 22
- Relatively simple features

F1 score



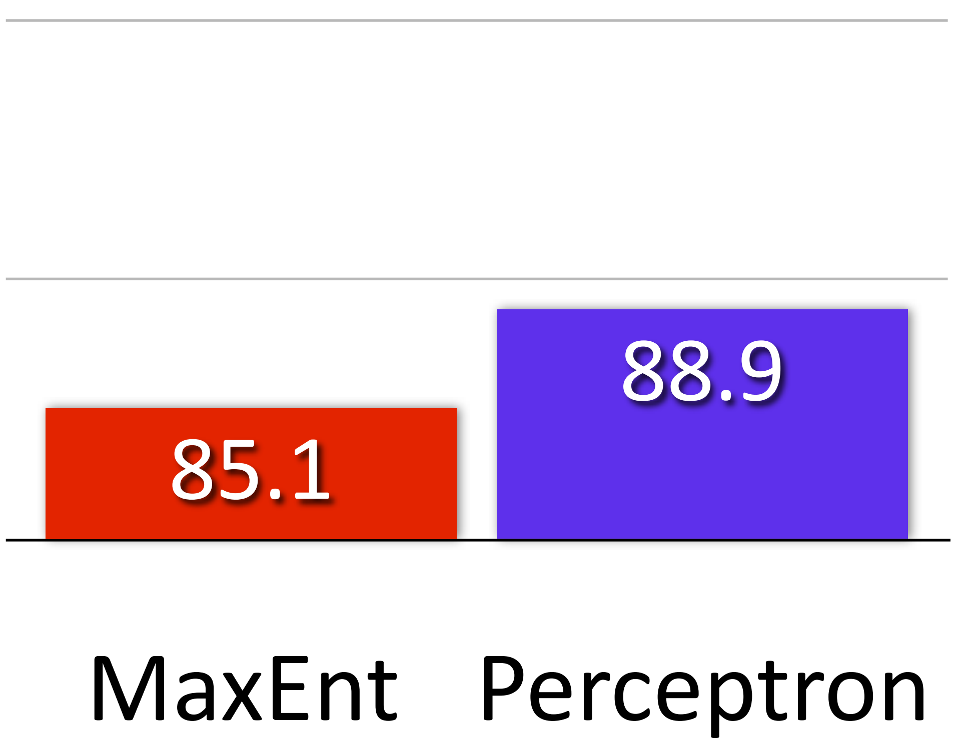
Search efficiency

- Structured perceptron is strong

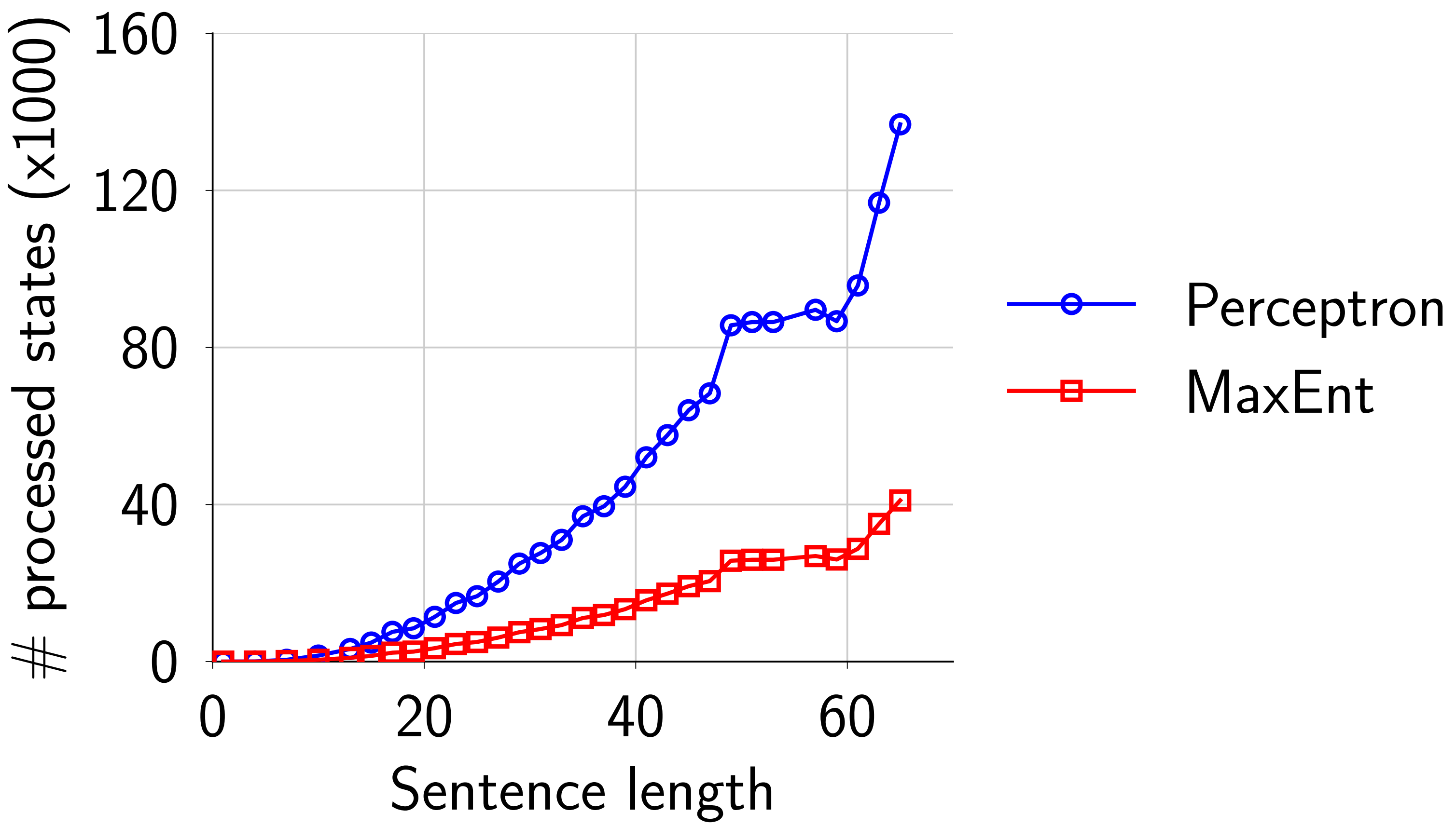
MaxEnt. vs. Structured Perceptron

- Data: WSJ Section 22
- Relatively simple features

F1 score

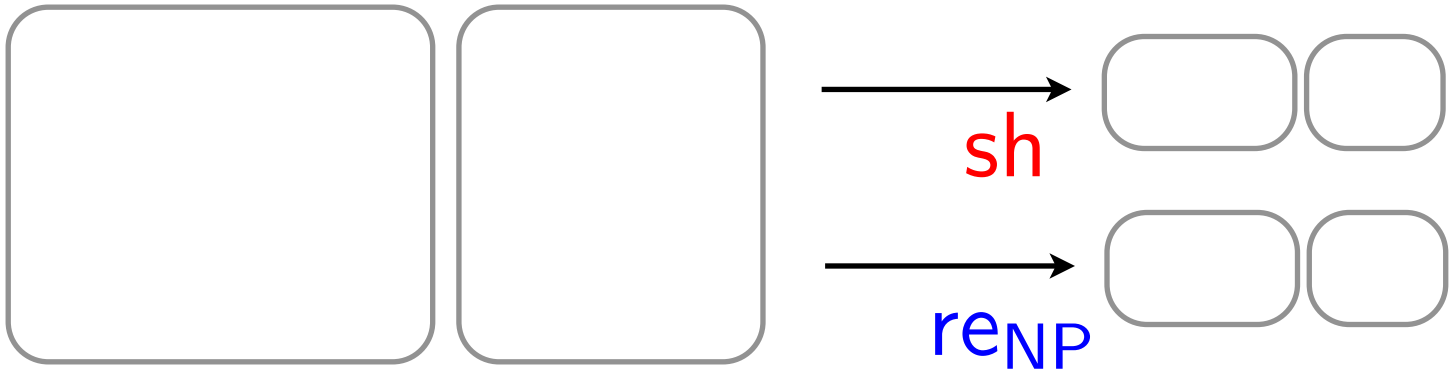


Search efficiency



- Structured perceptron is strong
- but its search is much harder ⇒ Why?

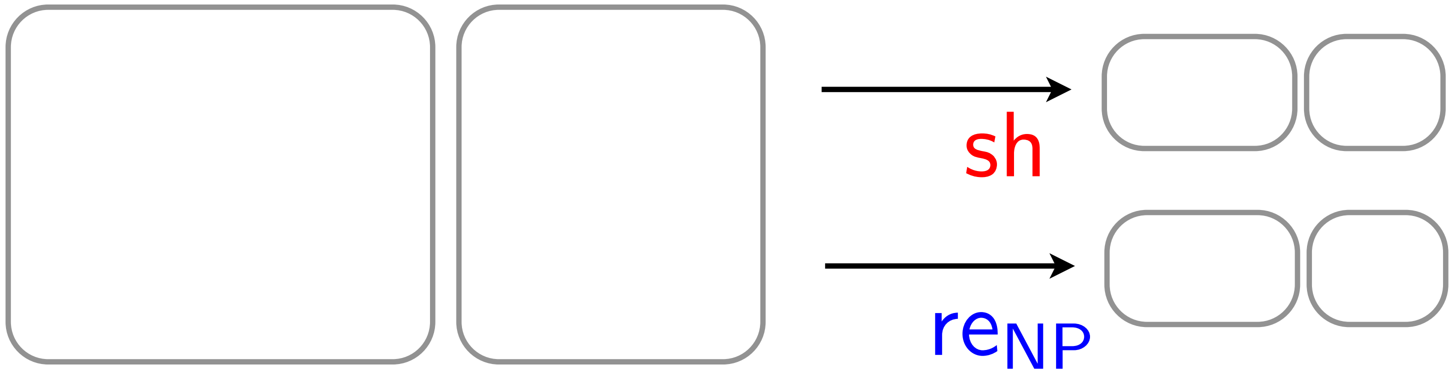
Why is search of structured perceptron so hard?



Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\top f(\mathsf{sh}, x)$	$\theta^\top f(\mathsf{re}_{\mathsf{NP}}, x)$
Structured perceptron	$\theta^\top f(\mathsf{sh}, x) + \delta$	$\theta^\top f(\mathsf{re}_{\mathsf{NP}}, x) + \delta$
MaxEnt	$-\log\left(\frac{e^{\theta^\top f(\mathsf{sh},x)}}{e^{\theta^\top f(\mathsf{sh},x)} + e^{\theta^\top f(\mathsf{re}_{\mathsf{NP}},x)}}\right)$	$-\log\left(\frac{e^{\theta^\top f(\mathsf{re}_{\mathsf{NP}},x)}}{e^{\theta^\top f(\mathsf{sh},x)} + e^{\theta^\top f(\mathsf{re}_{\mathsf{NP}},x)}}\right)$

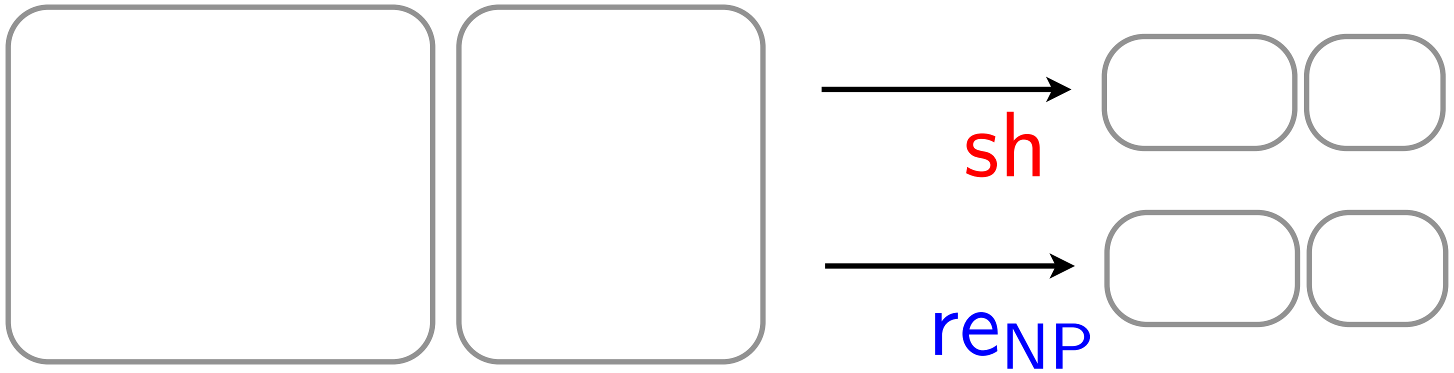
Why is search of structured perceptron so hard?



Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\top f(\text{sh}, x)$		$\theta^\top f(\text{re}_{\text{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\top f(\text{sh}, x) + \delta$		$\theta^\top f(\text{re}_{\text{NP}}, x) + \delta$
MaxEnt	$-\log\left(\frac{e^{\theta^\top f(\text{sh},x)}}{e^{\theta^\top f(\text{sh},x)} + e^{\theta^\top f(\text{re}_{\text{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\top f(\text{re}_{\text{NP}},x)}}{e^{\theta^\top f(\text{sh},x)} + e^{\theta^\top f(\text{re}_{\text{NP}},x)}}\right)$

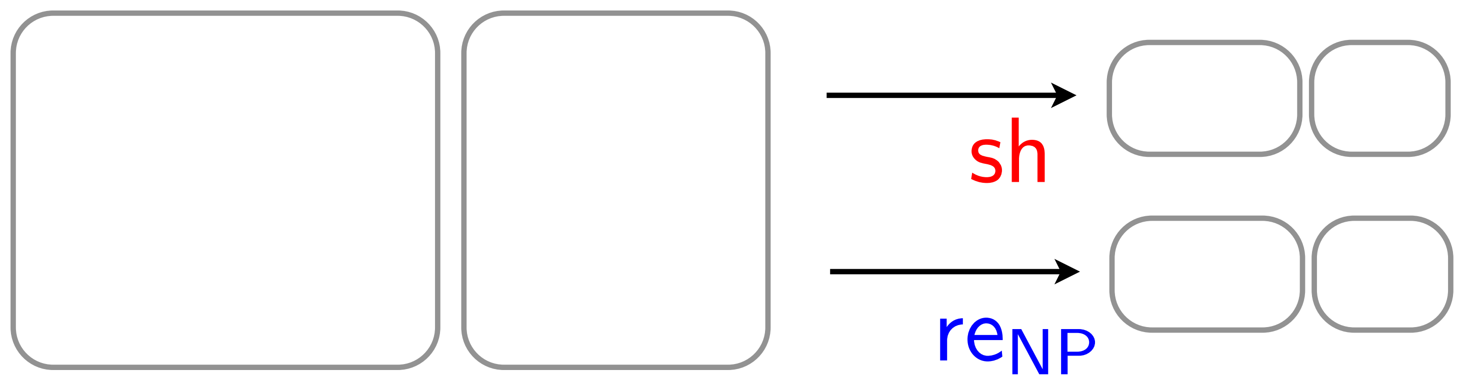
Why is search of structured perceptron so hard?



Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\top f(\text{sh}, x)$		$\theta^\top f(\text{re}_{\text{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\top f(\text{sh}, x) + \delta$		$\theta^\top f(\text{re}_{\text{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\top f(\text{sh},x)}}{e^{\theta^\top f(\text{sh},x)} + e^{\theta^\top f(\text{re}_{\text{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\top f(\text{re}_{\text{NP}},x)}}{e^{\theta^\top f(\text{sh},x)} + e^{\theta^\top f(\text{re}_{\text{NP}},x)}}\right)$

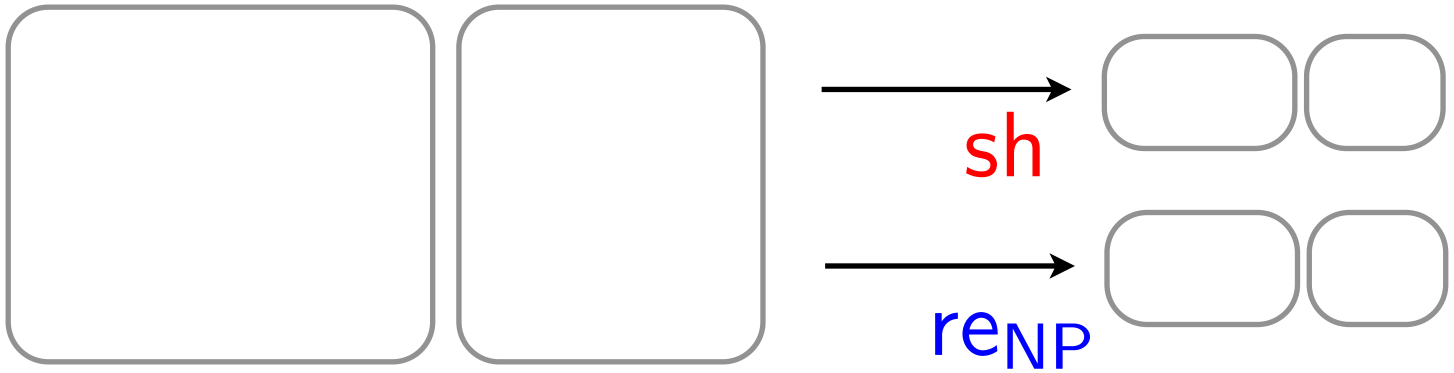
Why is search of structured perceptron so hard?



Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\intercal f(\text{sh}, x)$		$\theta^\intercal f(\text{re}_{\text{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\intercal f(\text{sh}, x) + \delta$		$\theta^\intercal f(\text{re}_{\text{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\intercal f(\text{sh},x)}}{e^{\theta^\intercal f(\text{sh},x)} + e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}{e^{\theta^\intercal f(\text{sh},x)} + e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}\right)$
	4.34	**vs.**	**0.0000197**

Why is search of structured perceptron so hard?

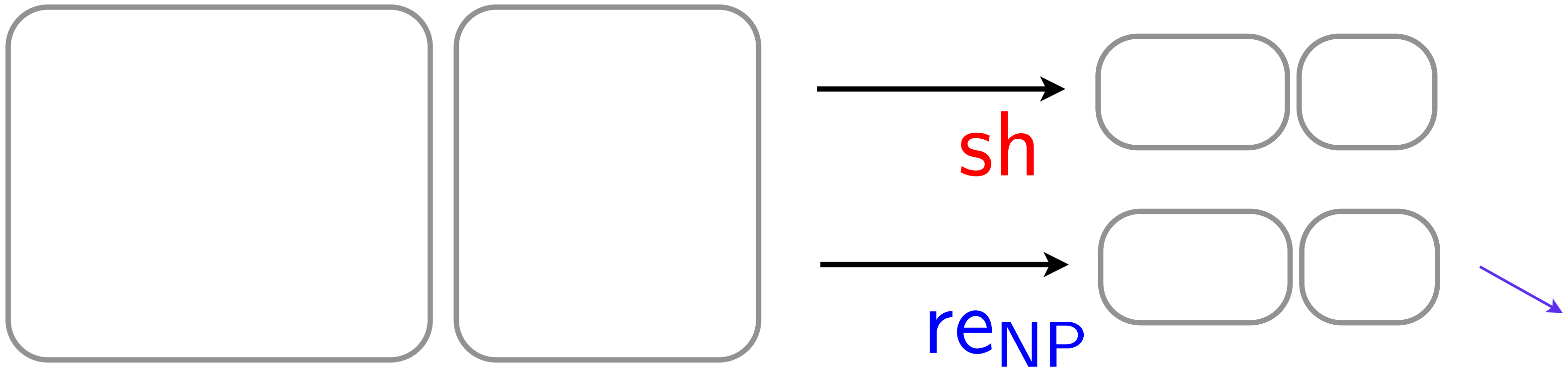


Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\top f(\text{sh}, x)$		$\theta^\top f(\text{re}_{\text{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\top f(\text{sh}, x) + \delta$		$\theta^\top f(\text{re}_{\text{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\top f(\text{sh},x)}}{e^{\theta^\top f(\text{sh},x)} + e^{\theta^\top f(\text{re}_{\text{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\top f(\text{re}_{\text{NP}},x)}}{e^{\theta^\top f(\text{sh},x)} + e^{\theta^\top f(\text{re}_{\text{NP}},x)}}\right)$
	4.34	**vs.**	**0.0000197**

- Exponential operation of MaxEnt facilitates search

Why is search of structured perceptron so hard?

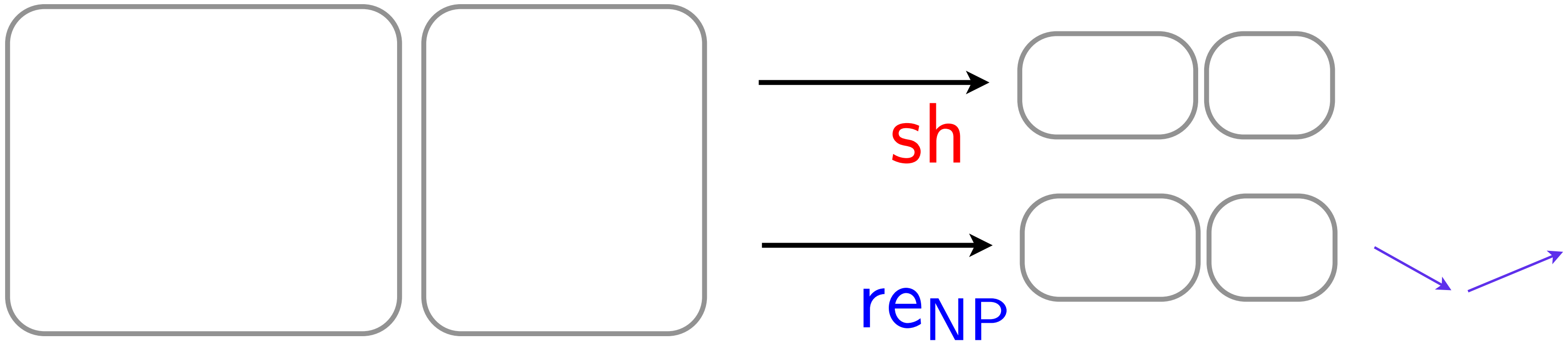


Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\top f(\mathsf{sh}, x)$		$\theta^\top f(\mathsf{re}_{\mathsf{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\top f(\mathsf{sh}, x) + \delta$		$\theta^\top f(\mathsf{re}_{\mathsf{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\top f(\mathsf{sh},x)}}{e^{\theta^\top f(\mathsf{sh},x)} + e^{\theta^\top f(\mathsf{re}_{\mathsf{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\top f(\mathsf{re}_{\mathsf{NP}},x)}}{e^{\theta^\top f(\mathsf{sh},x)} + e^{\theta^\top f(\mathsf{re}_{\mathsf{NP}},x)}}\right)$
	4.34	**vs.**	**0.0000197**

- Exponential operation of MaxEnt facilitates search

Why is search of structured perceptron so hard?

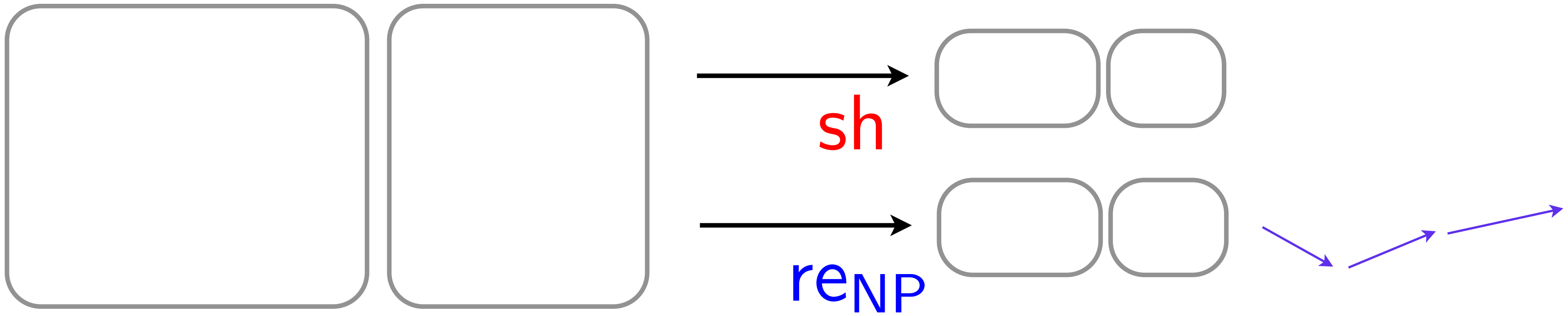


Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\intercal f(\mathsf{sh}, x)$		$\theta^\intercal f(\mathsf{re}_{\mathsf{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\intercal f(\mathsf{sh}, x) + \delta$		$\theta^\intercal f(\mathsf{re}_{\mathsf{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\intercal f(\mathsf{sh},x)}}{e^{\theta^\intercal f(\mathsf{sh},x)} + e^{\theta^\intercal f(\mathsf{re}_{\mathsf{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\intercal f(\mathsf{re}_{\mathsf{NP}},x)}}{e^{\theta^\intercal f(\mathsf{sh},x)} + e^{\theta^\intercal f(\mathsf{re}_{\mathsf{NP}},x)}}\right)$
	4.34	**vs.**	**0.0000197**

- Exponential operation of MaxEnt facilitates search

Why is search of structured perceptron so hard?

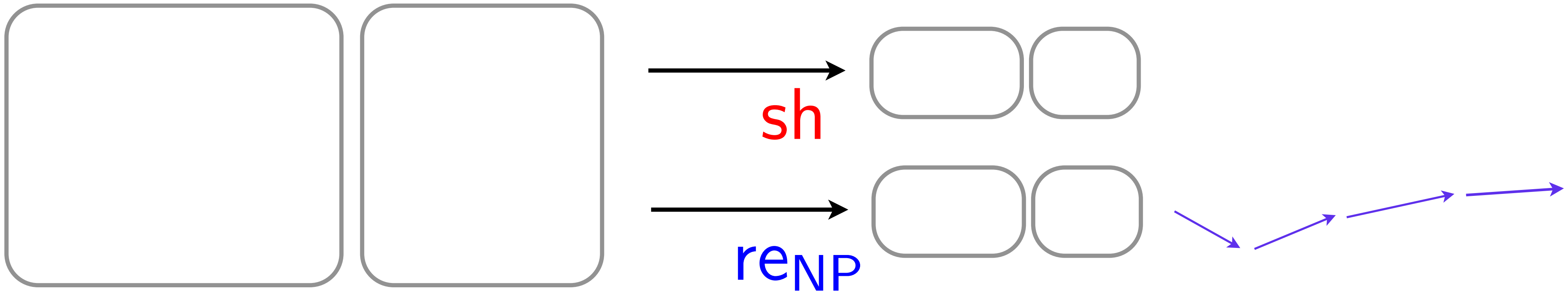


Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\top f(\mathsf{sh}, x)$		$\theta^\top f(\mathsf{re}_{\mathsf{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\top f(\mathsf{sh}, x) + \delta$		$\theta^\top f(\mathsf{re}_{\mathsf{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\top f(\mathsf{sh},x)}}{e^{\theta^\top f(\mathsf{sh},x)} + e^{\theta^\top f(\mathsf{re}_{\mathsf{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\top f(\mathsf{re}_{\mathsf{NP}},x)}}{e^{\theta^\top f(\mathsf{sh},x)} + e^{\theta^\top f(\mathsf{re}_{\mathsf{NP}},x)}}\right)$
	4.34	**vs.**	**0.0000197**

- Exponential operation of MaxEnt facilitates search

Why is search of structured perceptron so hard?

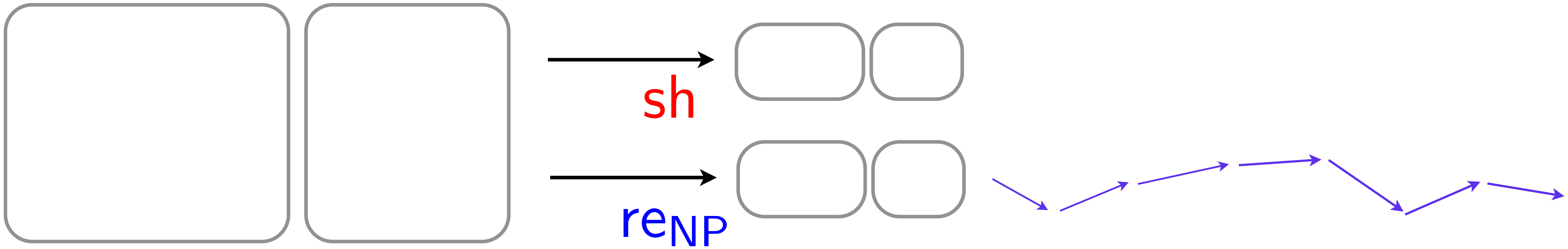


Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\intercal f(\text{sh}, x)$		$\theta^\intercal f(\text{re}_{\text{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\intercal f(\text{sh}, x) + \delta$		$\theta^\intercal f(\text{re}_{\text{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\intercal f(\text{sh},x)}}{e^{\theta^\intercal f(\text{sh},x)} + e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}{e^{\theta^\intercal f(\text{sh},x)} + e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}\right)$
	4.34	**vs.**	**0.0000197**

- Exponential operation of MaxEnt facilitates search

Why is search of structured perceptron so hard?

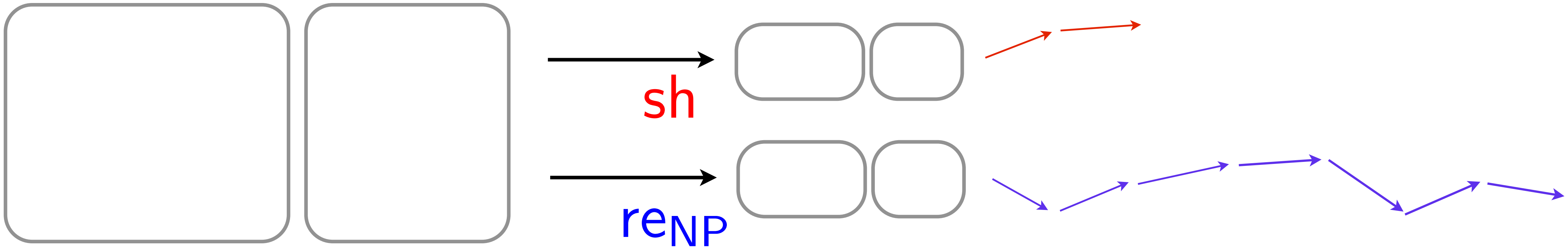


Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\intercal f(\mathsf{sh}, x)$		$\theta^\intercal f(\mathsf{re}_{\mathsf{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\intercal f(\mathsf{sh}, x) + \delta$		$\theta^\intercal f(\mathsf{re}_{\mathsf{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\intercal f(\mathsf{sh},x)}}{e^{\theta^\intercal f(\mathsf{sh},x)} + e^{\theta^\intercal f(\mathsf{re}_{\mathsf{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\intercal f(\mathsf{re}_{\mathsf{NP}},x)}}{e^{\theta^\intercal f(\mathsf{sh},x)} + e^{\theta^\intercal f(\mathsf{re}_{\mathsf{NP}},x)}}\right)$
	4.34	**vs.**	**0.0000197**

- Exponential operation of MaxEnt facilitates search

Why is search of structured perceptron so hard?

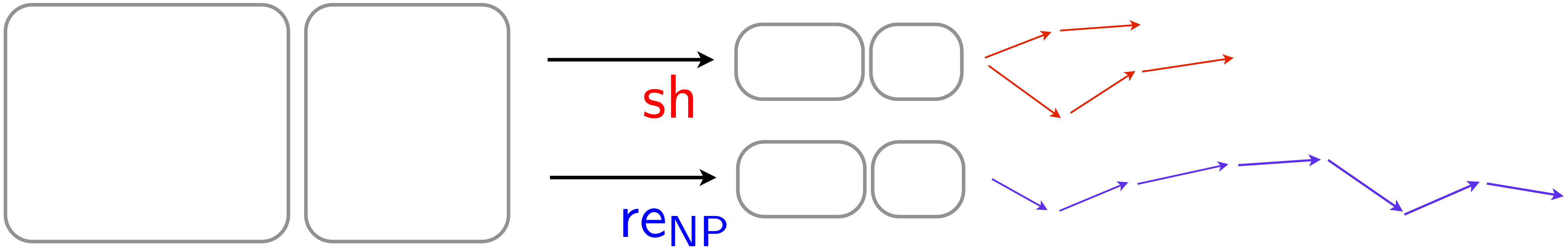


Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\intercal f(\text{sh}, x)$		$\theta^\intercal f(\text{re}_{\text{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\intercal f(\text{sh}, x) + \delta$		$\theta^\intercal f(\text{re}_{\text{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\intercal f(\text{sh},x)}}{e^{\theta^\intercal f(\text{sh},x)} + e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}{e^{\theta^\intercal f(\text{sh},x)} + e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}\right)$
	4.34	**vs.**	**0.0000197**

- Exponential operation of MaxEnt facilitates search

Why is search of structured perceptron so hard?

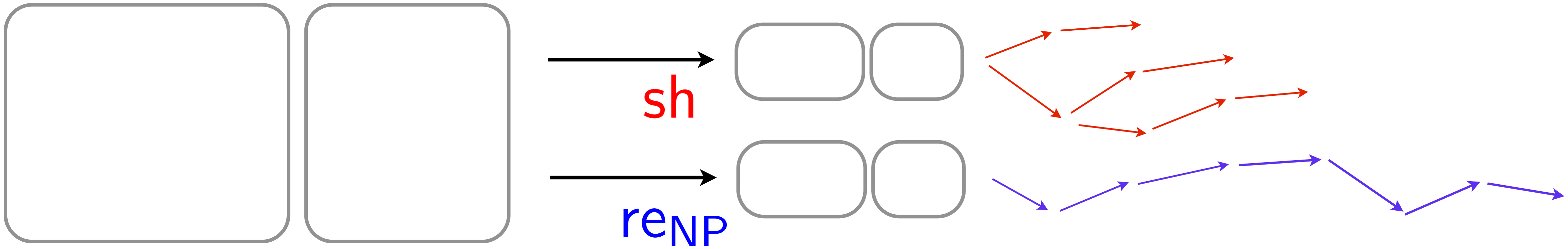


Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\top f(\text{sh}, x)$		$\theta^\top f(\text{re}_{\text{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\top f(\text{sh}, x) + \delta$		$\theta^\top f(\text{re}_{\text{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\top f(\text{sh},x)}}{e^{\theta^\top f(\text{sh},x)} + e^{\theta^\top f(\text{re}_{\text{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\top f(\text{re}_{\text{NP}},x)}}{e^{\theta^\top f(\text{sh},x)} + e^{\theta^\top f(\text{re}_{\text{NP}},x)}}\right)$
	4.34	**vs.**	**0.0000197**

- Exponential operation of MaxEnt facilitates search

Why is search of structured perceptron so hard?

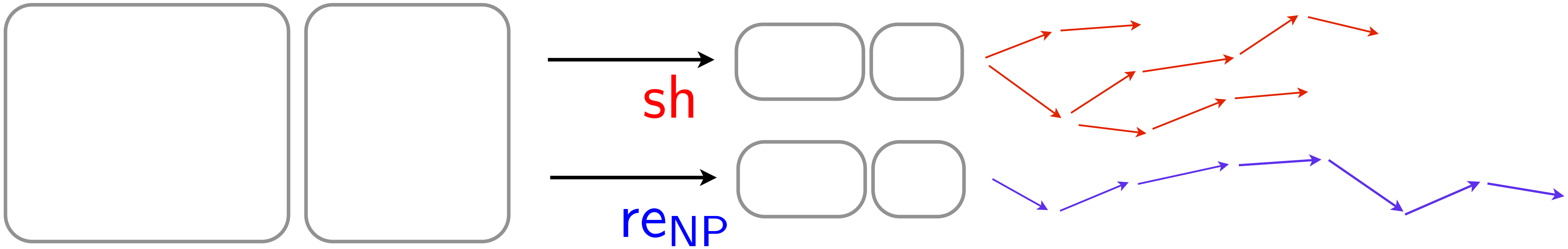


Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^\intercal f(\text{sh}, x)$		$\theta^\intercal f(\text{re}_{\text{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^\intercal f(\text{sh}, x) + \delta$		$\theta^\intercal f(\text{re}_{\text{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^\intercal f(\text{sh},x)}}{e^{\theta^\intercal f(\text{sh},x)} + e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}\right)$		$-\log\left(\frac{e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}{e^{\theta^\intercal f(\text{sh},x)} + e^{\theta^\intercal f(\text{re}_{\text{NP}},x)}}\right)$
	4.34	**vs.**	**0.0000197**

- Exponential operation of MaxEnt facilitates search

Why is search of structured perceptron so hard?



Main reason: Sparsity of edge (action) costs

Feature weight	$\theta^{\mathsf{T}} f(\mathsf{sh}, x)$		$\theta^{\mathsf{T}} f(\mathsf{re}_{\mathsf{NP}}, x)$
	40	vs.	50
Structured perceptron	$\theta^{\mathsf{T}} f(\mathsf{sh}, x) + \delta$		$\theta^{\mathsf{T}} f(\mathsf{re}_{\mathsf{NP}}, x) + \delta$
	1040	vs.	1050
MaxEnt	$-\log\left(\frac{e^{\theta^{\mathsf{T}} f(\mathsf{sh}, x)}}{e^{\theta^{\mathsf{T}} f(\mathsf{sh}, x)} + e^{\theta^{\mathsf{T}} f(\mathsf{re}_{\mathsf{NP}}, x)}}\right)$		$-\log\left(\frac{e^{\theta^{\mathsf{T}} f(\mathsf{re}_{\mathsf{NP}}, x)}}{e^{\theta^{\mathsf{T}} f(\mathsf{sh}, x)} + e^{\theta^{\mathsf{T}} f(\mathsf{re}_{\mathsf{NP}}, x)}}\right)$
	4.34	**vs.**	**0.0000197**

- Exponential operation of MaxEnt facilitates search

Outline

DP best-first shift-reduce parsing

- for **constituency**
- with **structured perceptron**

MaxEnt vs. Structured perceptron

- Structured perceptron is strong
- but its search is much harder

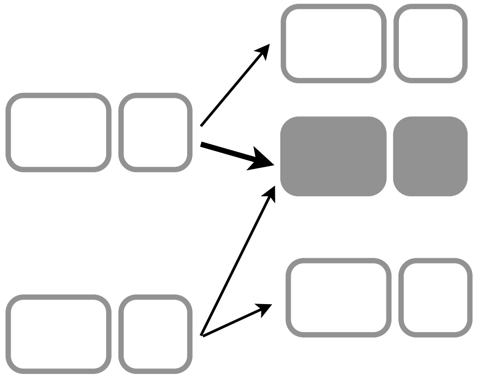
Improving search efficiency of structured perceptron

- New feature templates
- A* search

Final experiment

Why feature matters?

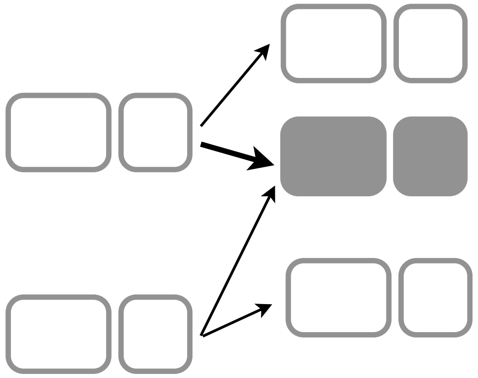
Feature design determines the worst time complexity of DP



Why feature matters?

Feature design determines the worst time complexity of DP

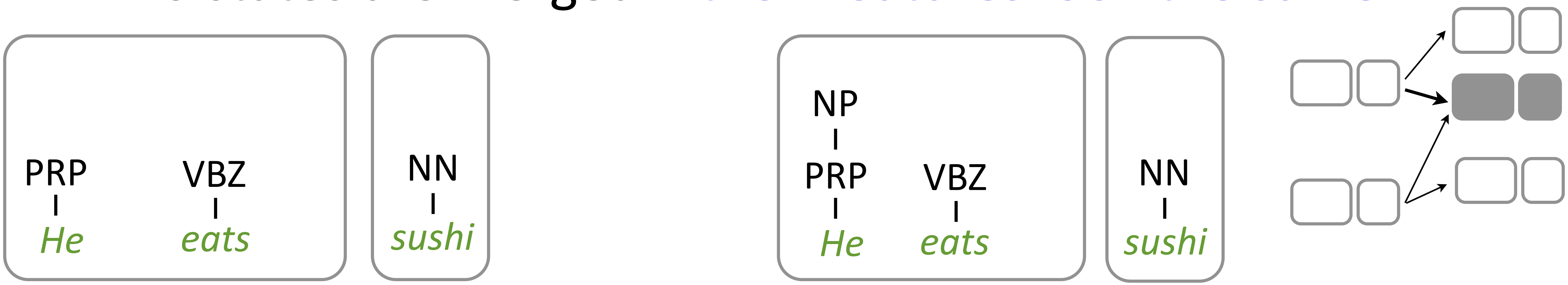
⇒ Two states are merged if their features look the same



Why feature matters?

Feature design determines the worst time complexity of DP

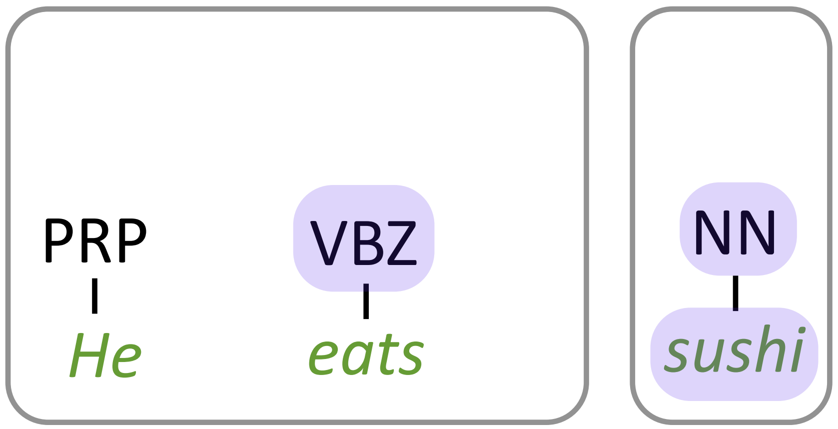
⇒ Two states are merged if their features look the same

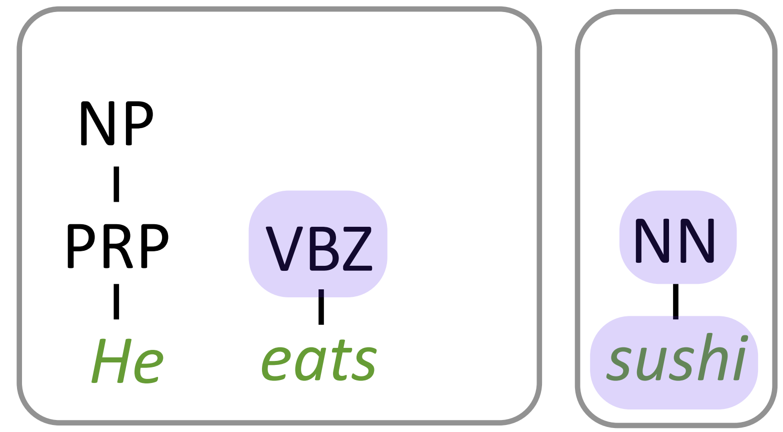


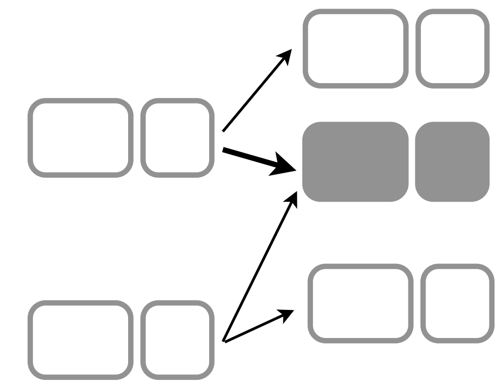
Why feature matters?

Feature design determines the worst time complexity of DP

⇒ Two states are merged if their features look the same



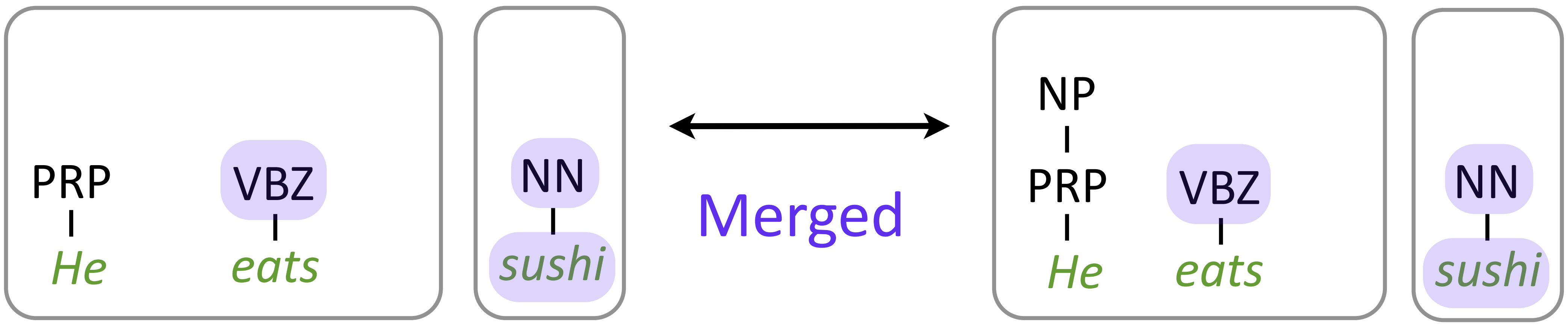


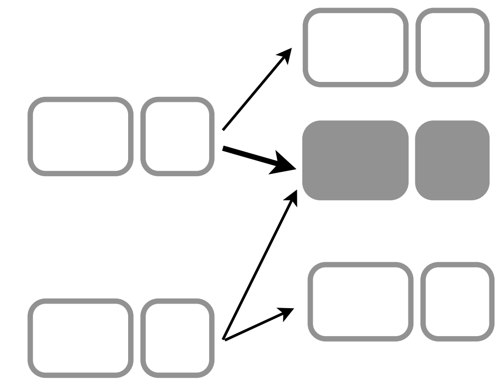


Why feature matters?

Feature design determines the worst time complexity of DP

⇒ Two states are merged if their features look the same

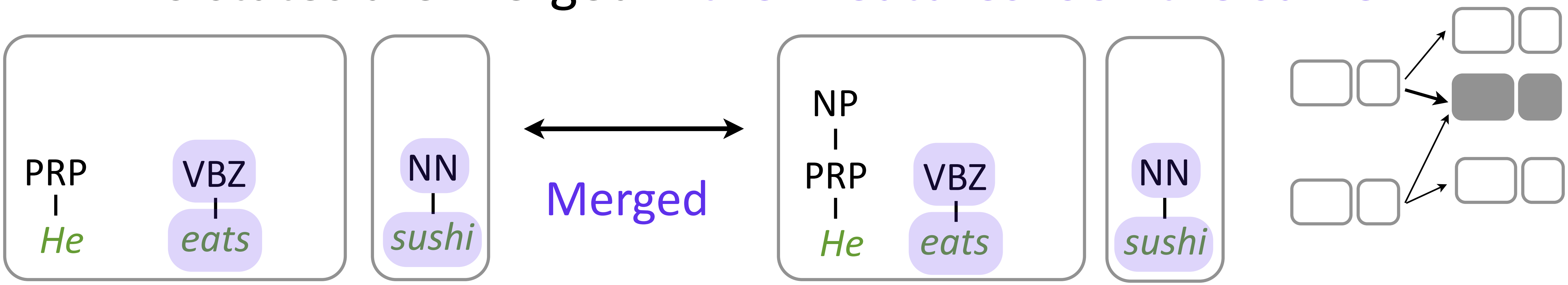




Why feature matters?

Feature design determines the worst time complexity of DP

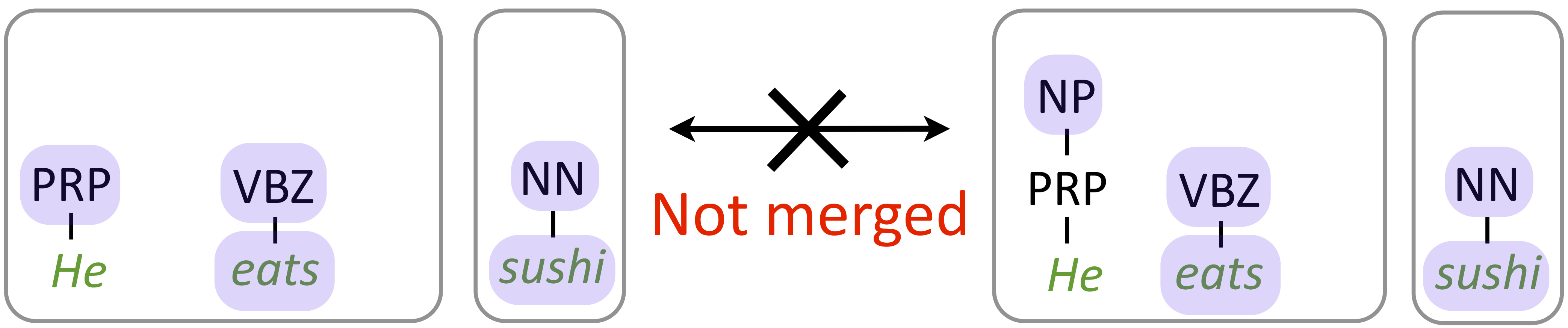
⇒ Two states are merged if their features look the same

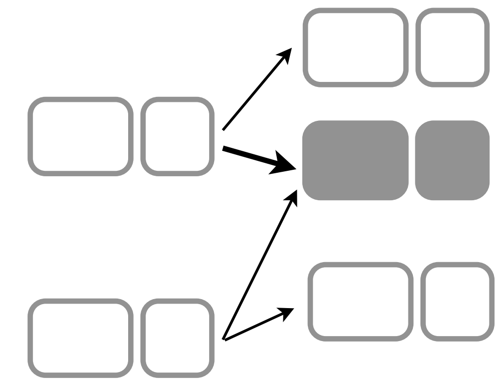


Why feature matters?

Feature design determines the worst time complexity of DP

⇒ Two states are merged if their features look the same

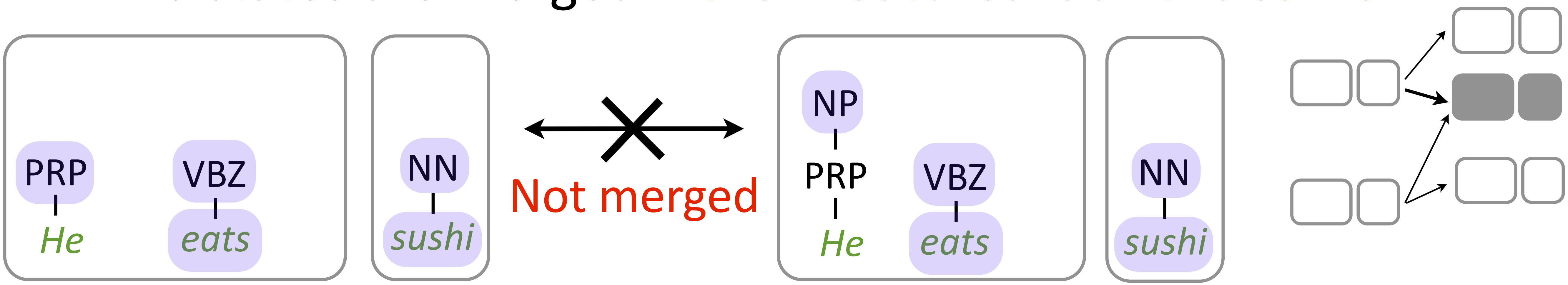




Why feature matters?

Feature design determines the worst time complexity of DP

⇒ Two states are merged if their features look the same

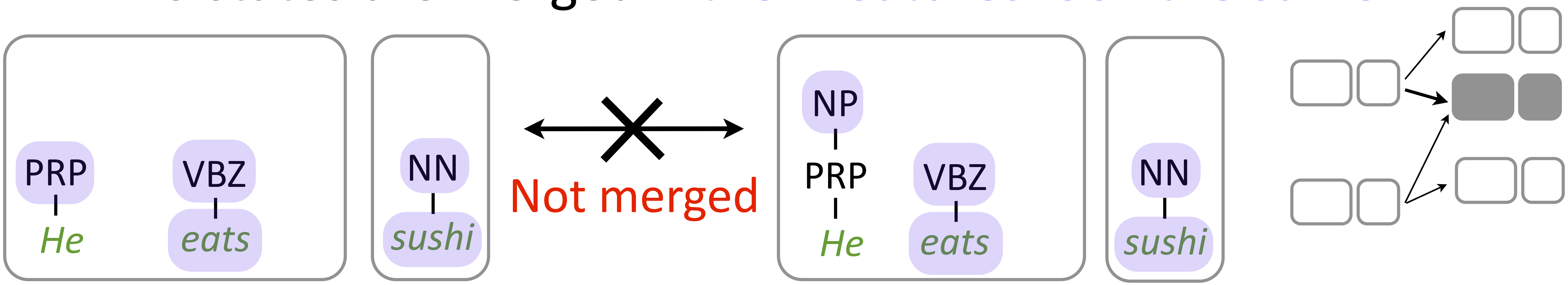


- Chance of merging decreases by adding more features

Why feature matters?

Feature design determines the worst time complexity of DP

⇒ Two states are merged if their features look the same

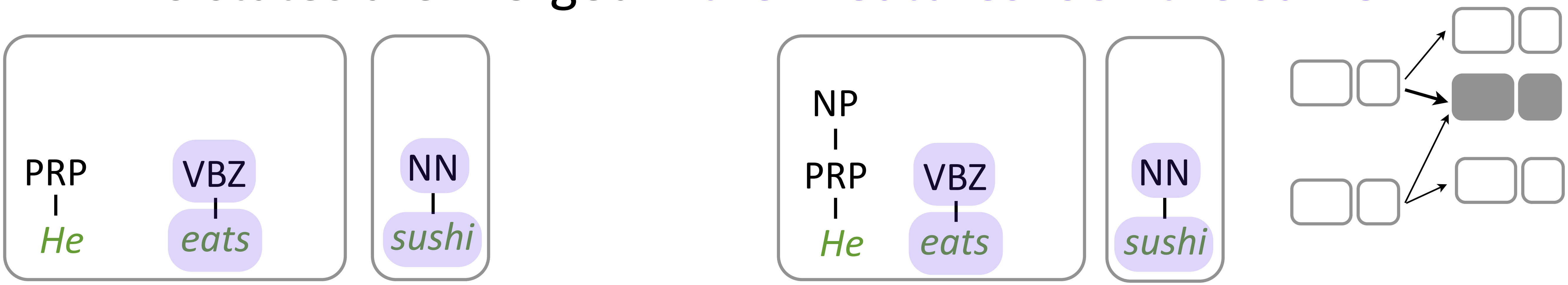


- Chance of merging decreases by adding more features
- Time complexity:

Why feature matters?

Feature design determines the worst time complexity of DP

⇒ Two states are merged if their features look the same

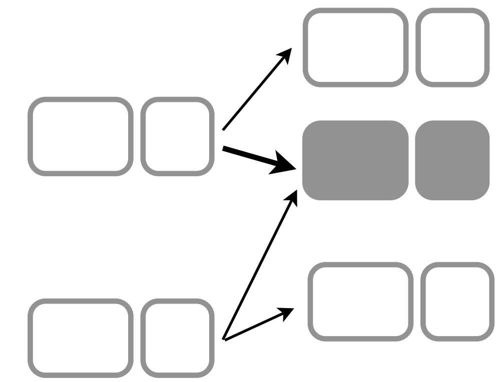


- Chance of merging decreases by adding more features
- Time complexity: $O(n^3|G|)$

Why feature matters?

Feature design determines the worst time complexity of DP

⇒ Two states are merged if their features look the same

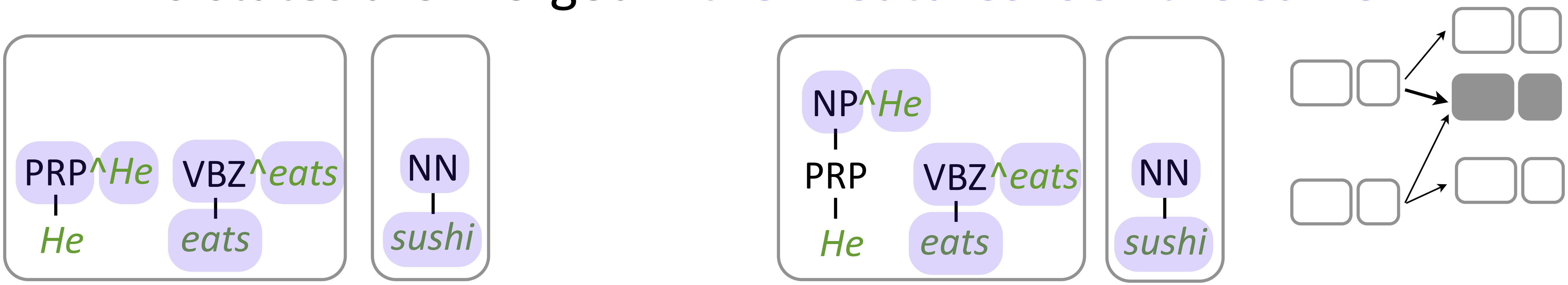


- Chance of merging decreases by adding more features
- Time complexity: $O(n^3|G|) \longrightarrow O(n^3|G||N|)$

Why feature matters?

Feature design determines the worst time complexity of DP

⇒ Two states are merged if their features look the same

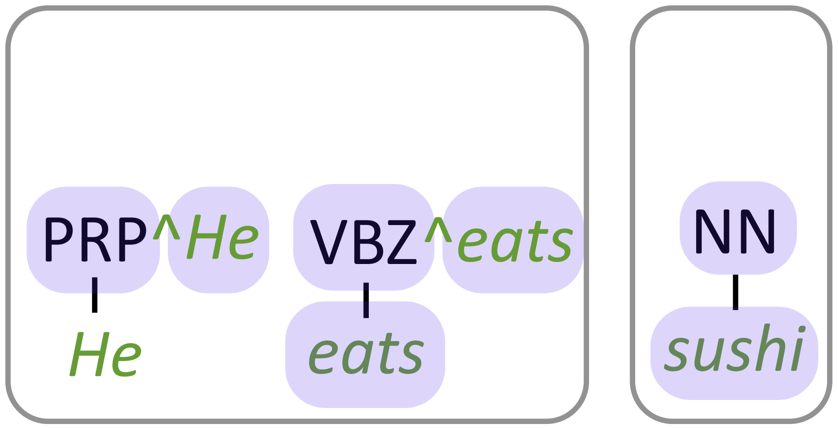


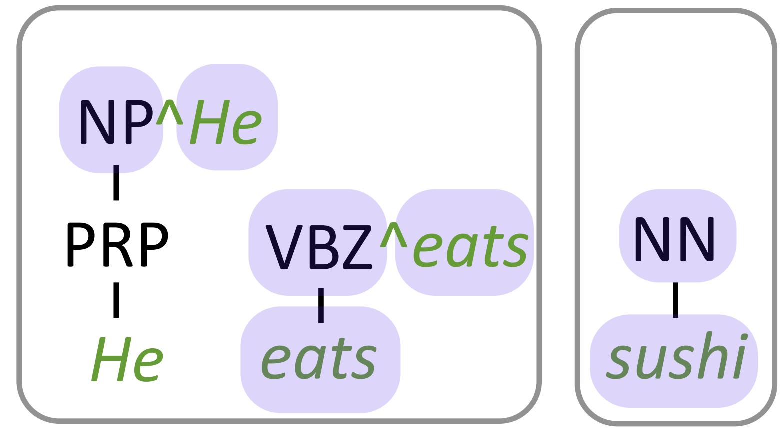
- Chance of merging decreases by adding more features
- Time complexity: $O(n^3|G|) \longrightarrow O(n^3|G||N|) \longrightarrow O(n^6|G||N|)$

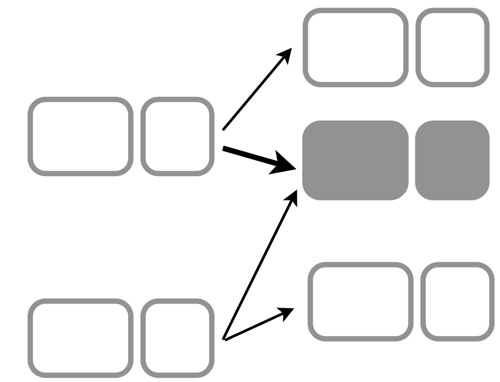
Why feature matters?

Feature design determines the worst time complexity of DP

⇒ Two states are merged if their features look the same







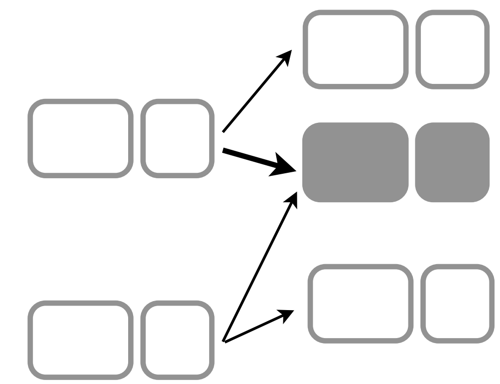
- Chance of merging decreases by adding more features
- Time complexity: $O(n^3|G|) \longrightarrow O(n^3|G||N|) \longrightarrow O(n^6|G||N|)$

State-of-the-art features [Zhu et al., 2013] (with beam-search)

Why feature matters?

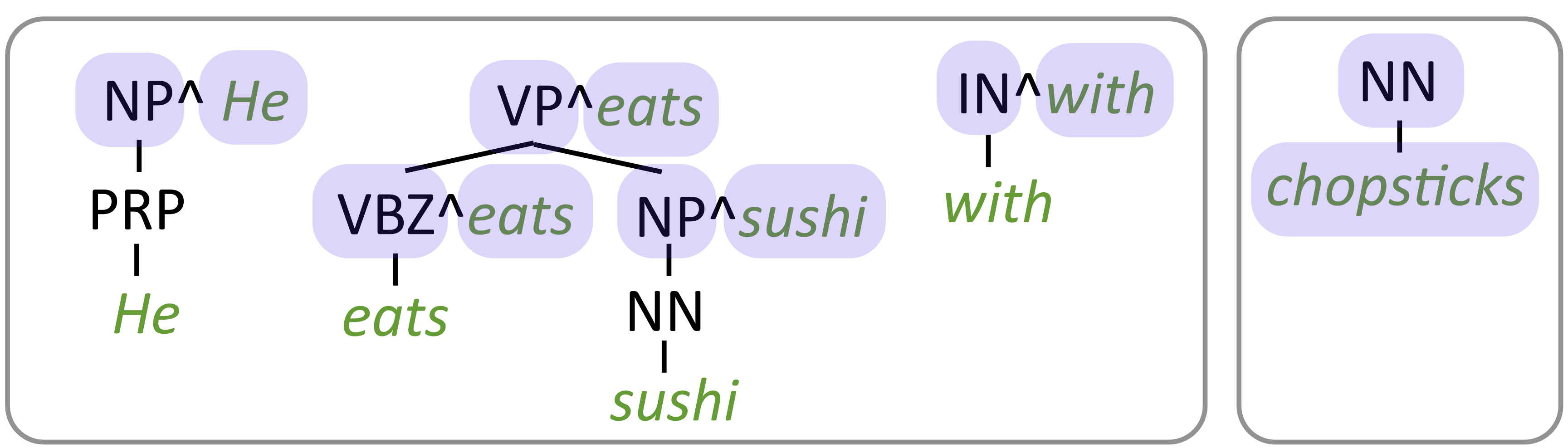
Feature design determines the worst time complexity of DP

$\Rightarrow$ Two states are merged if their features look the same



- Chance of merging decreases by adding more features
- Time complexity: $O(n^3|G|) \longrightarrow O(n^3|G||N|) \longrightarrow O(n^6|G||N|)$

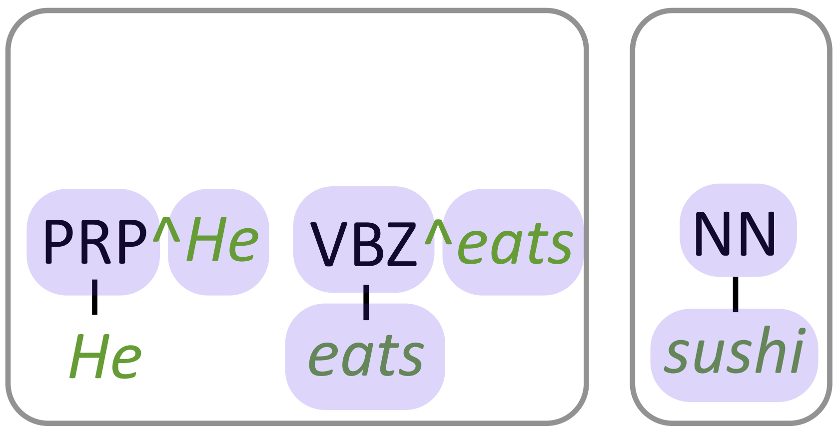
State-of-the-art features [Zhu et al., 2013] (with beam-search)

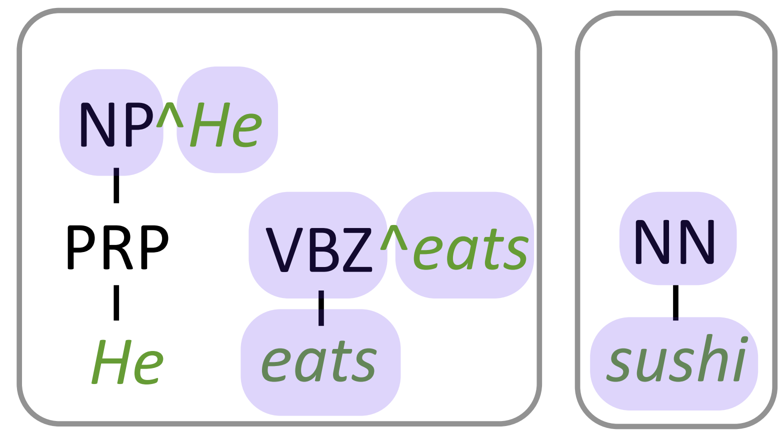


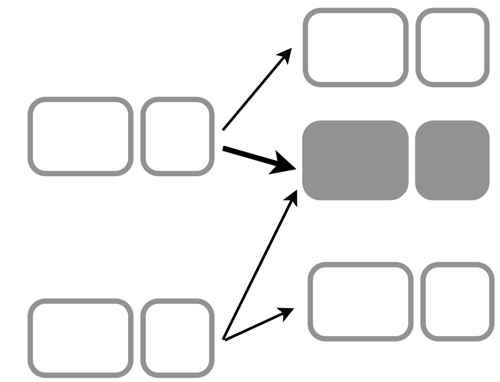
Why feature matters?

Feature design determines the worst time complexity of DP

⇒ Two states are merged if their features look the same

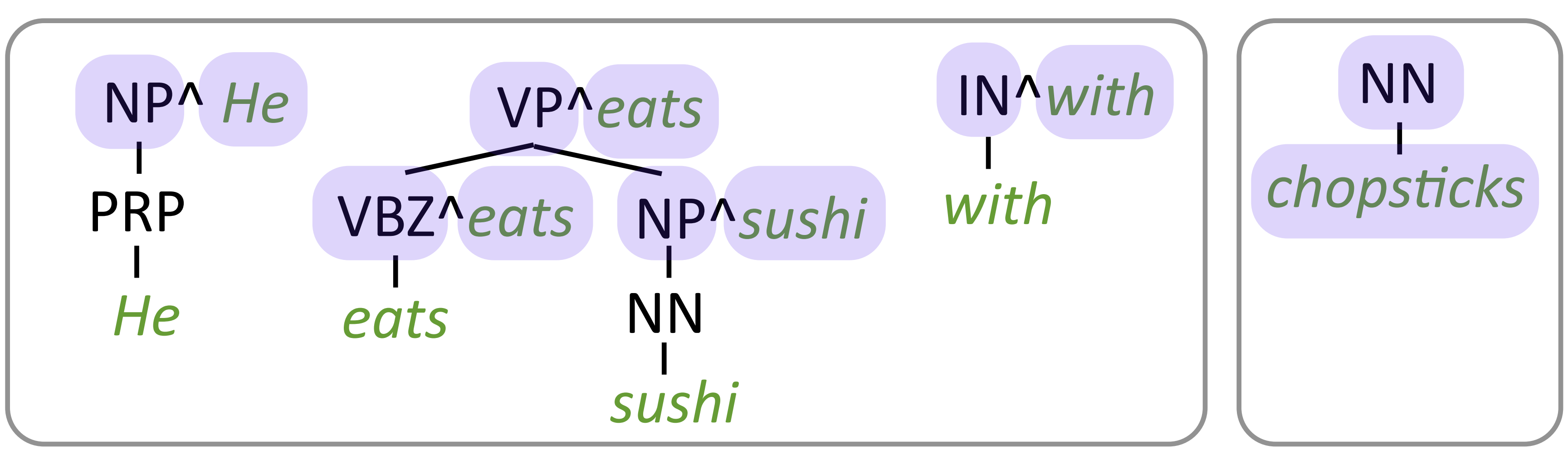






- Chance of merging decreases by adding more features
- Time complexity: $O(n^3|G|) \longrightarrow O(n^3|G||N|) \longrightarrow O(n^6|G||N|)$

State-of-the-art features [Zhu et al., 2013] (with beam-search)

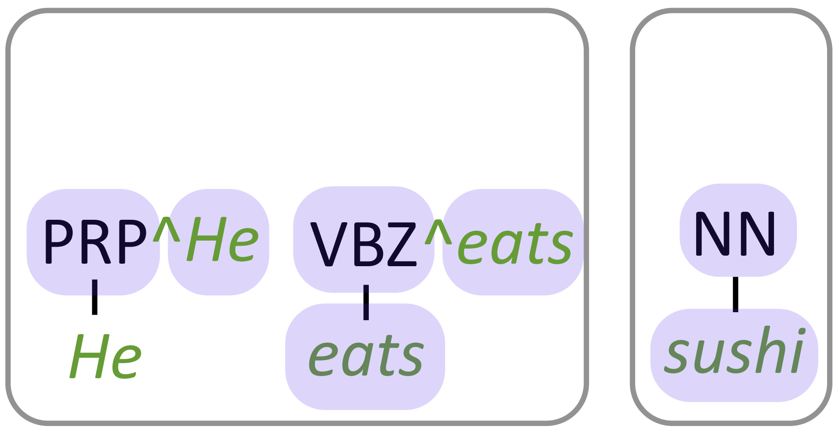


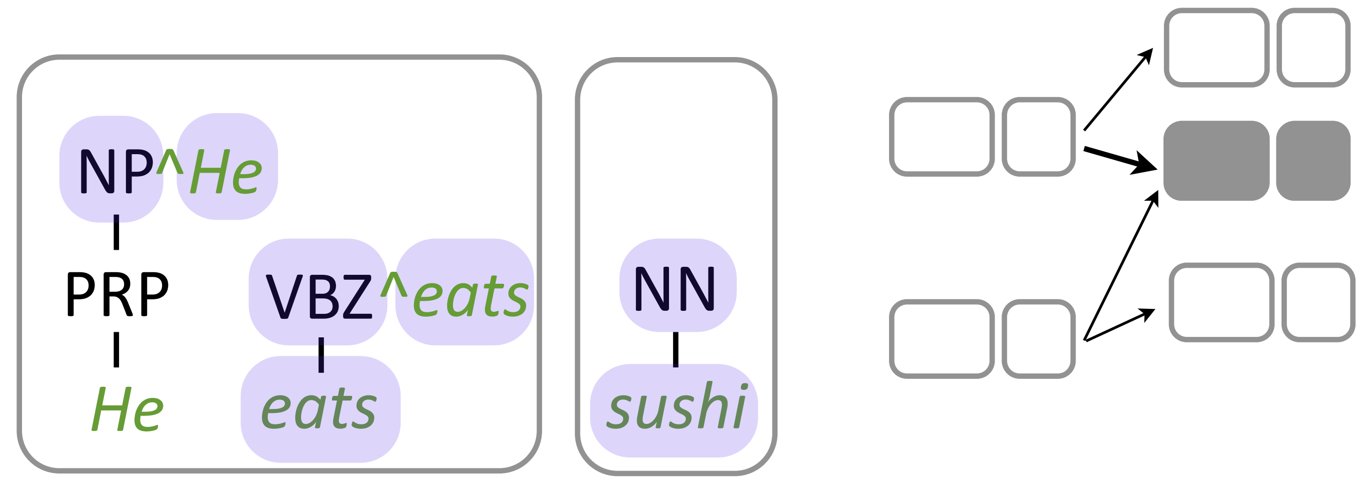
- Intractable!
- at least $O(n^{10})$?

Why feature matters?

Feature design determines the worst time complexity of DP

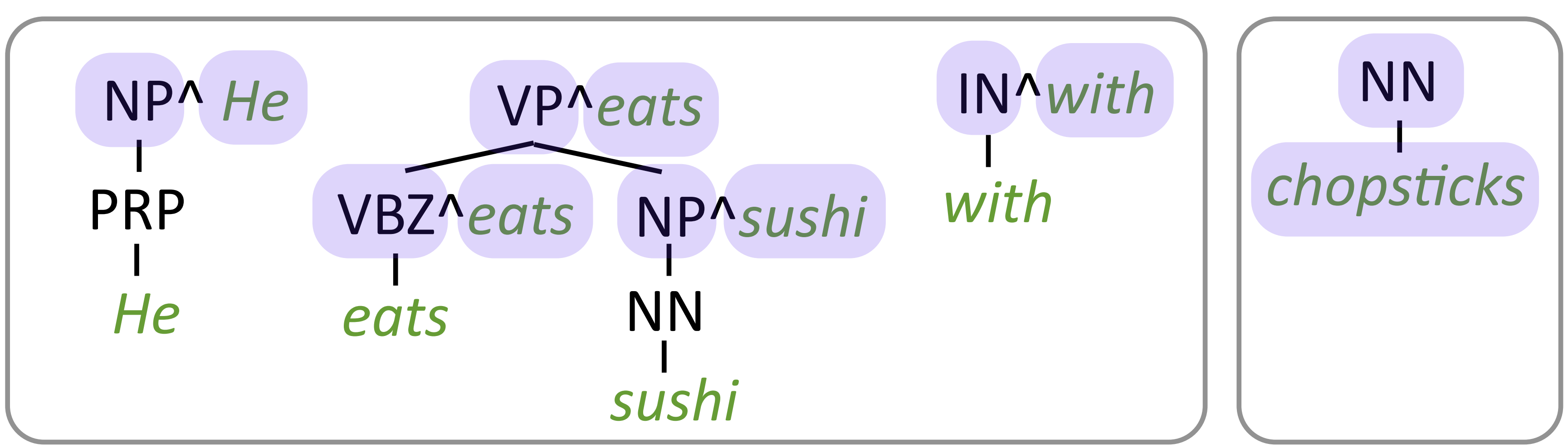
⇒ Two states are merged if their features look the same





- Chance of merging decreases by adding more features
- Time complexity: $O(n^3|G|) \longrightarrow O(n^3|G||N|) \longrightarrow O(n^{6}|G||N|)$

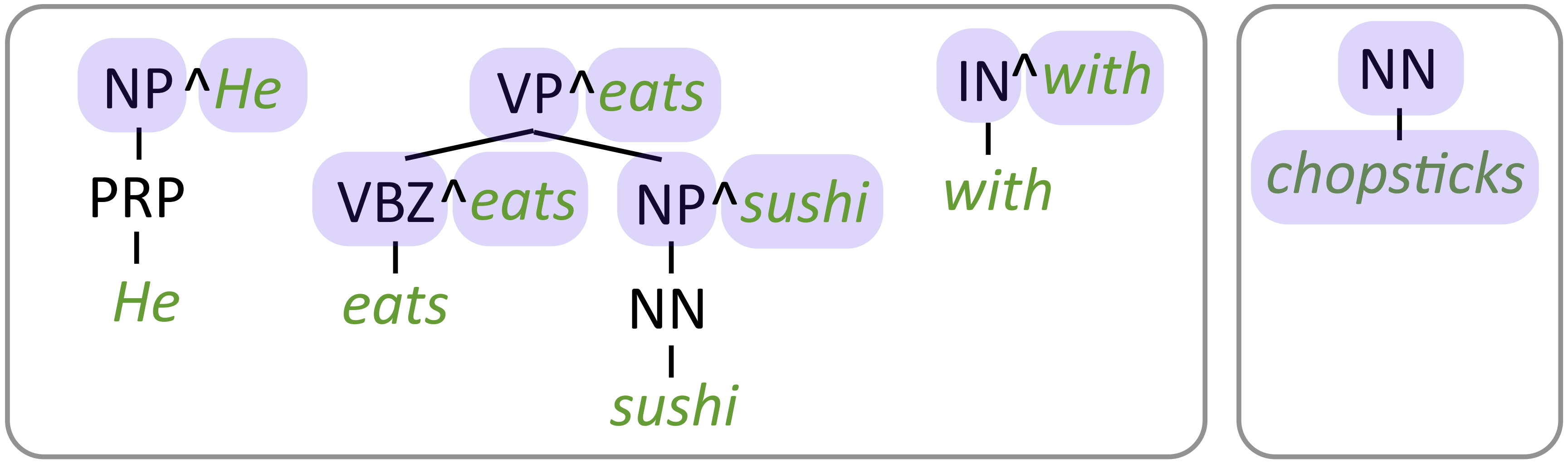
State-of-the-art features [Zhu et al., 2013] (with beam-search)



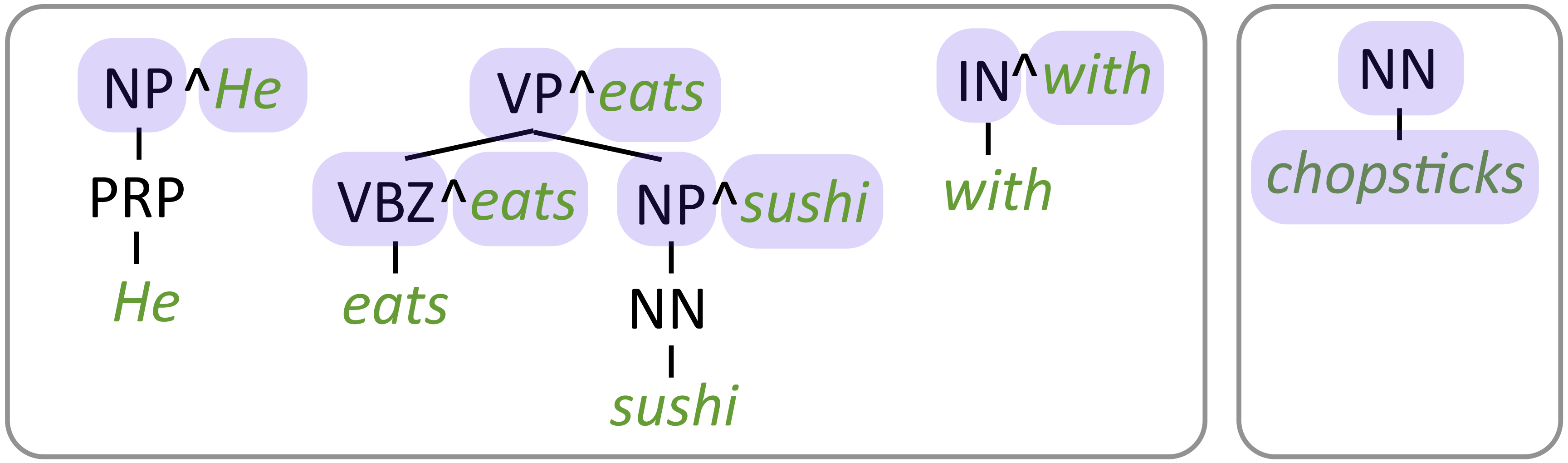
- Intractable!
- at least $O(n^{10})$?

How to solve this?

Recipe 1: Utilizing cheap features

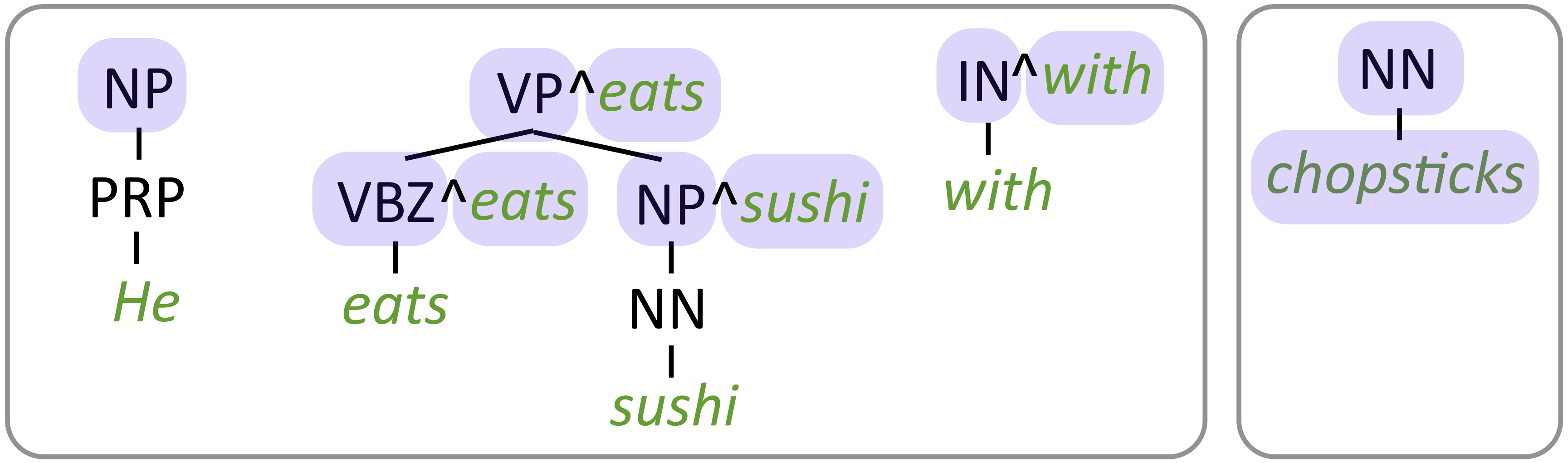


Recipe 1: Utilizing cheap features



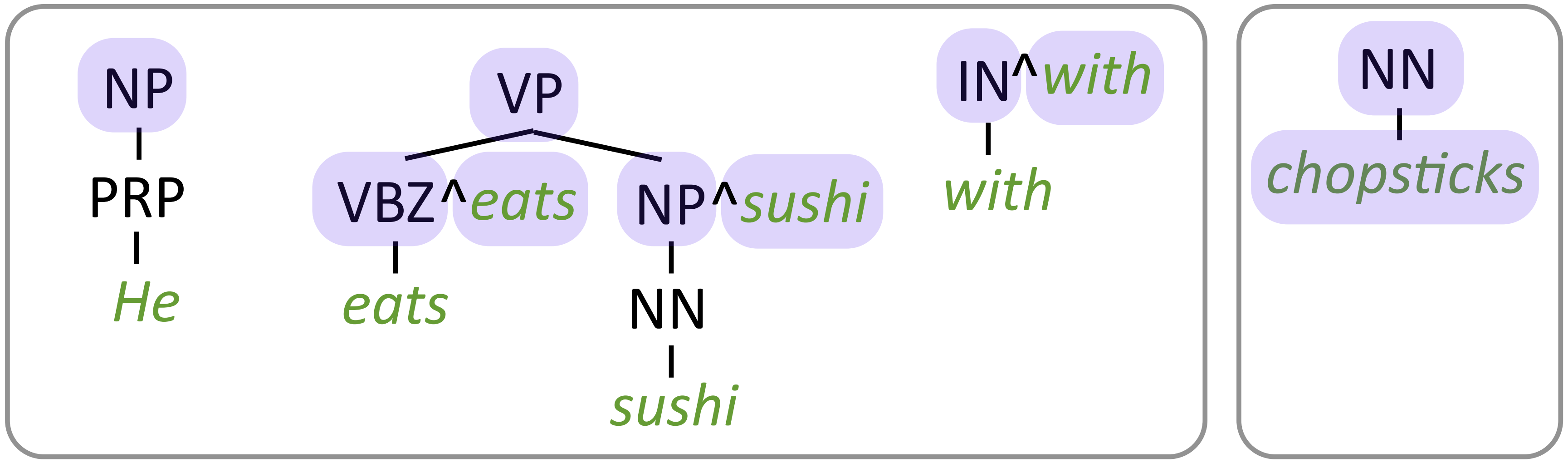
- Remembering head is expensive

Recipe 1: Utilizing cheap features



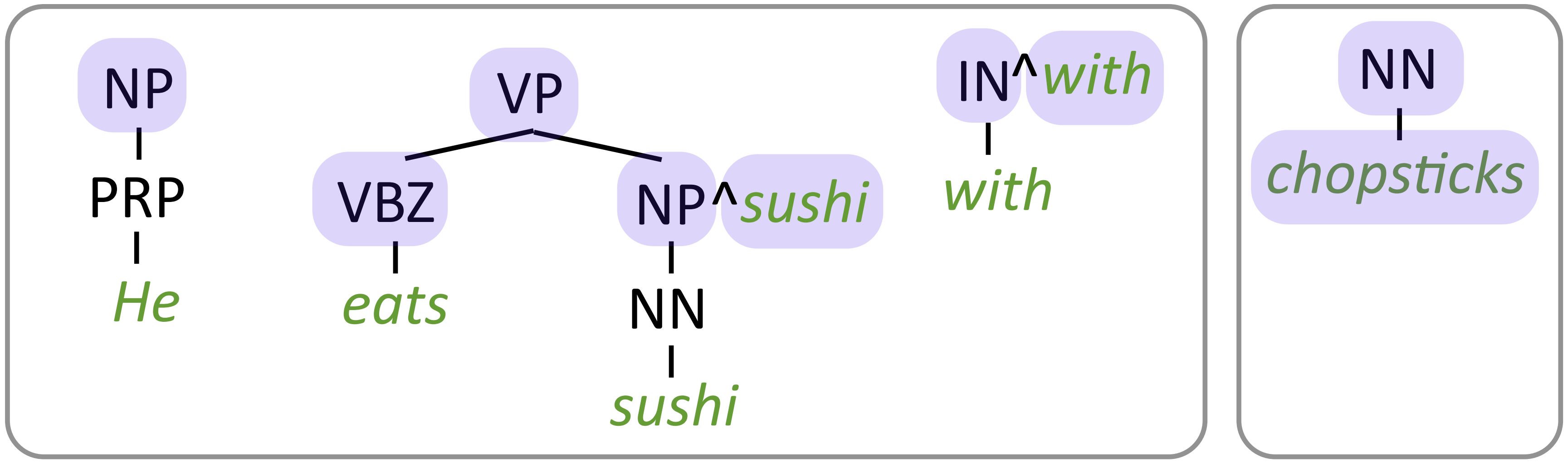
- Remembering head is expensive

Recipe 1: Utilizing cheap features



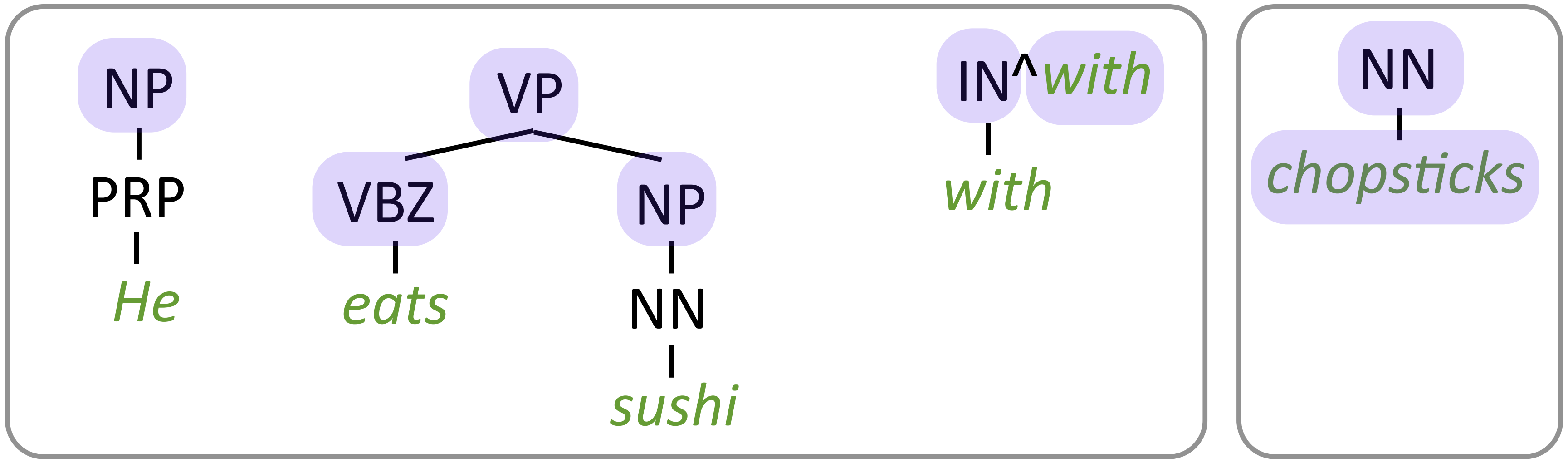
- Remembering head is expensive

Recipe 1: Utilizing cheap features



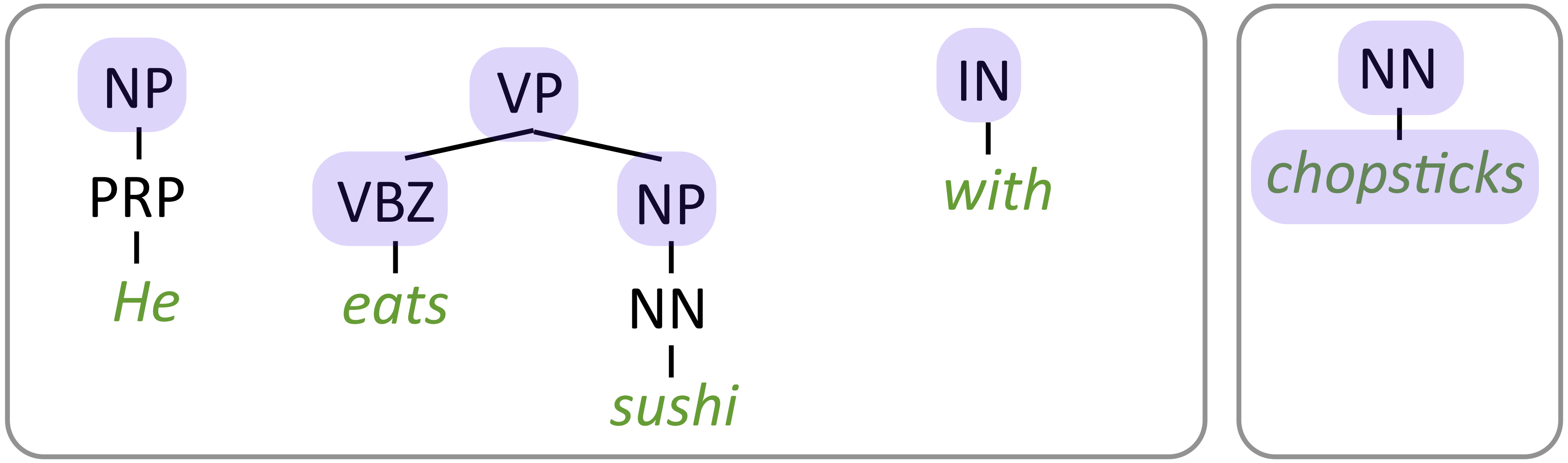
- Remembering head is expensive

Recipe 1: Utilizing cheap features



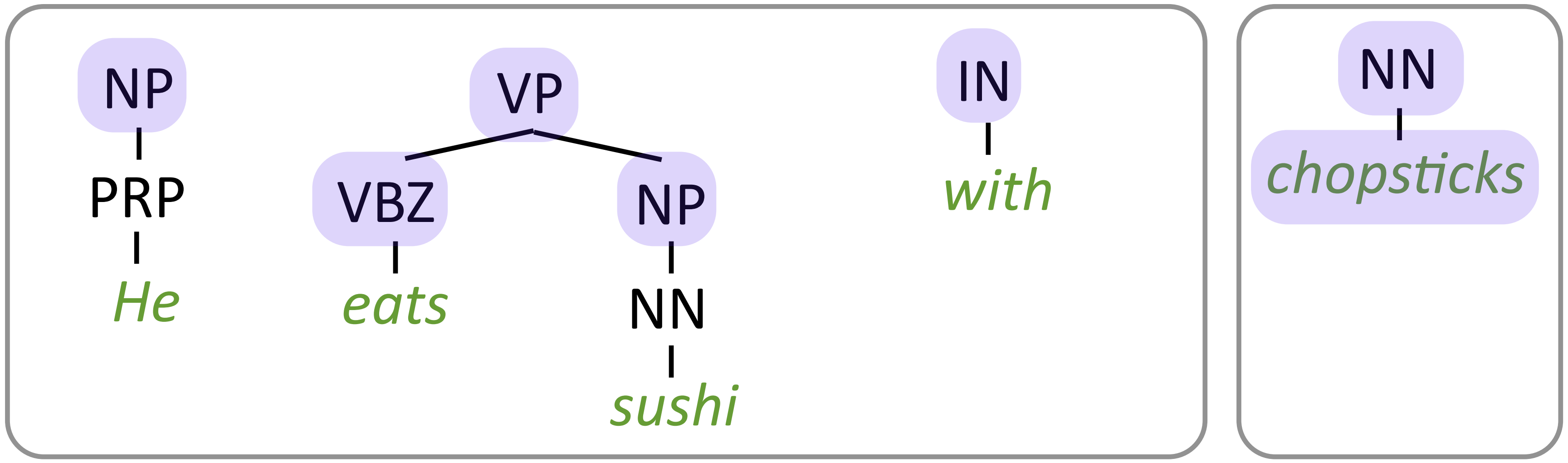
- Remembering head is expensive

Recipe 1: Utilizing cheap features



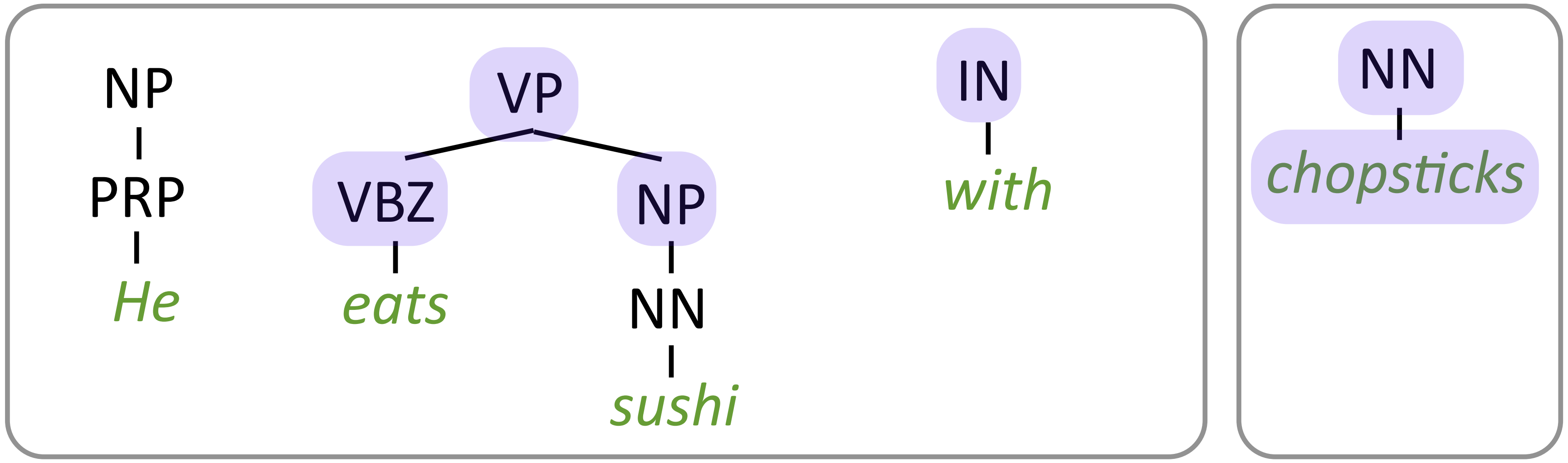
- Remembering head is expensive

Recipe 1: Utilizing cheap features



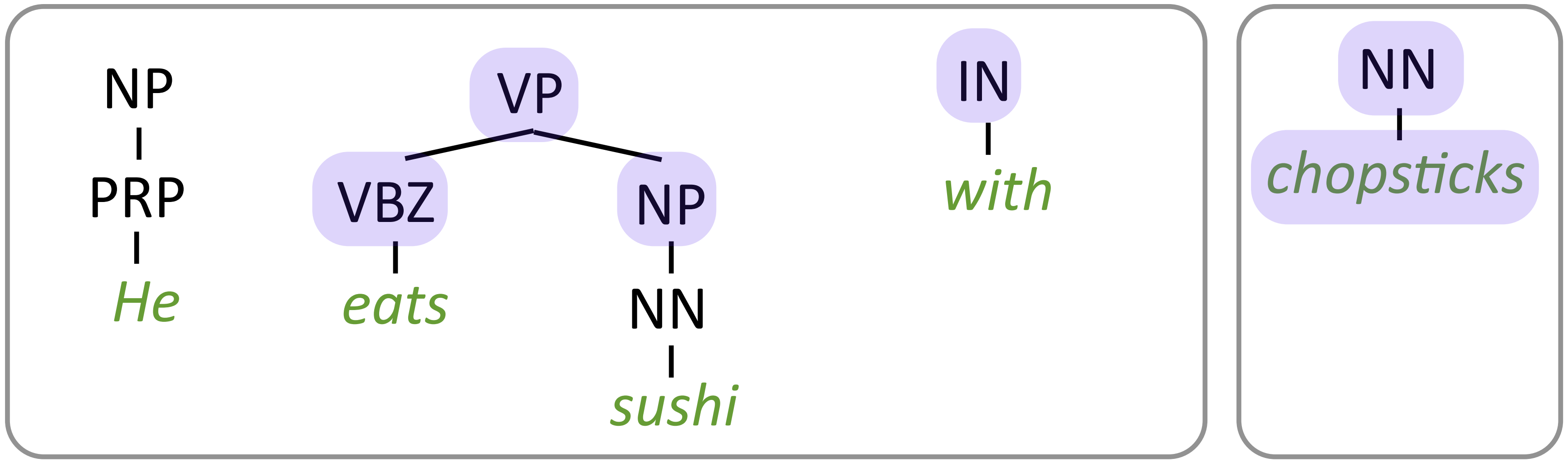
- Remembering head is expensive
- Deeper stack element is expensive

Recipe 1: Utilizing cheap features

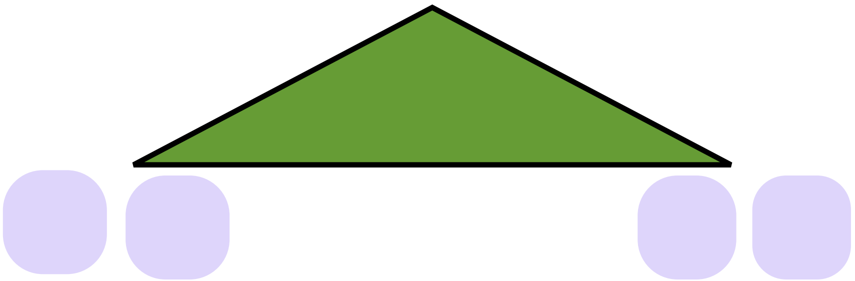


- Remembering head is expensive
- Deeper stack element is expensive

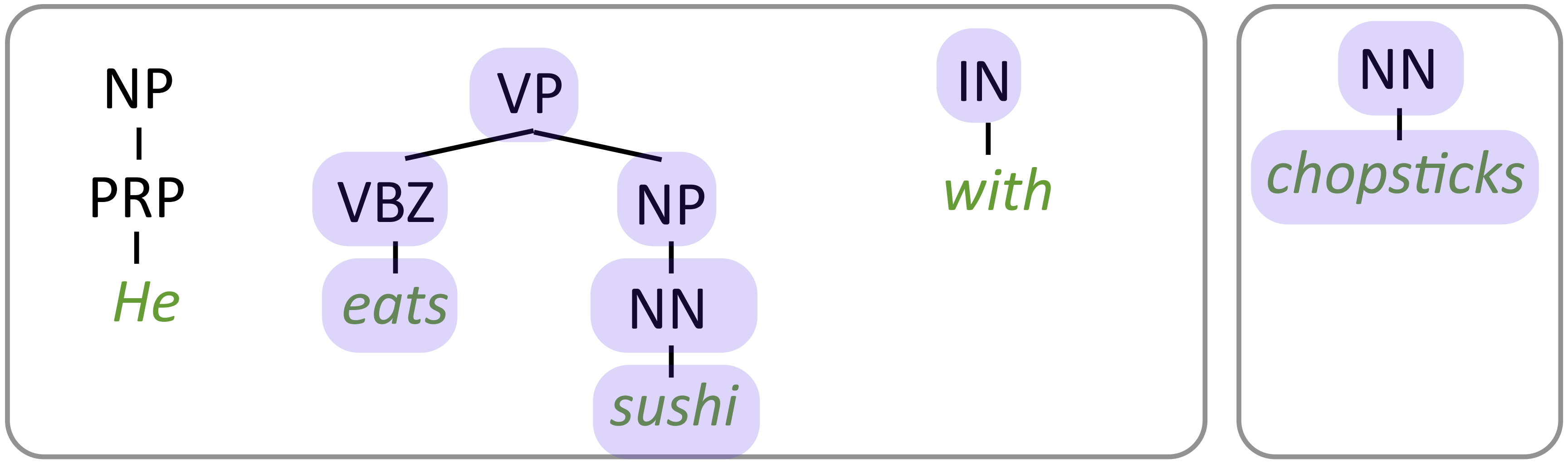
Recipe 1: Utilizing cheap features



- Remembering head is expensive
- Deeper stack element is expensive
- The span around a subtree is cheap

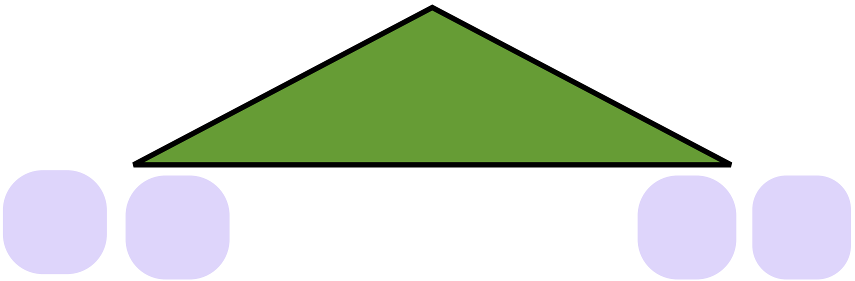


Recipe 1: Utilizing cheap features

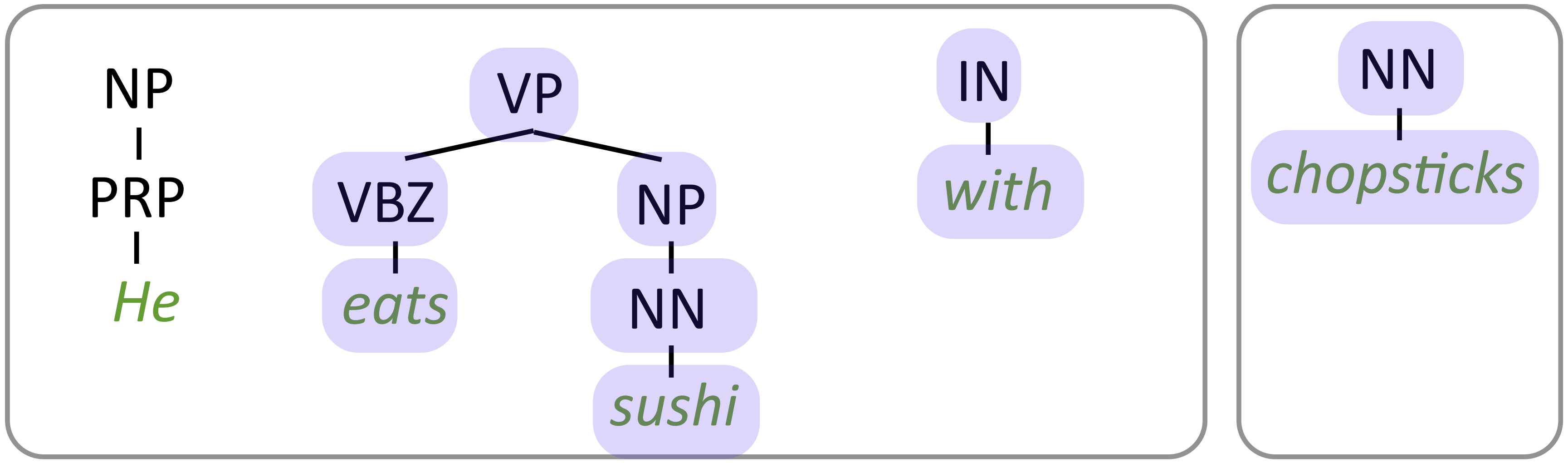


- Remembering head is expensive
- Deeper stack element is expensive

- The span around a subtree is cheap

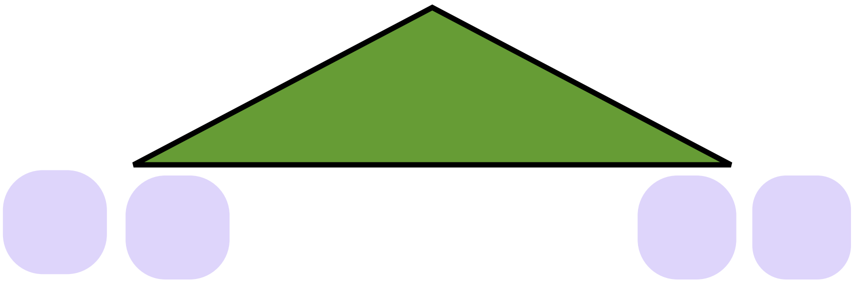


Recipe 1: Utilizing cheap features

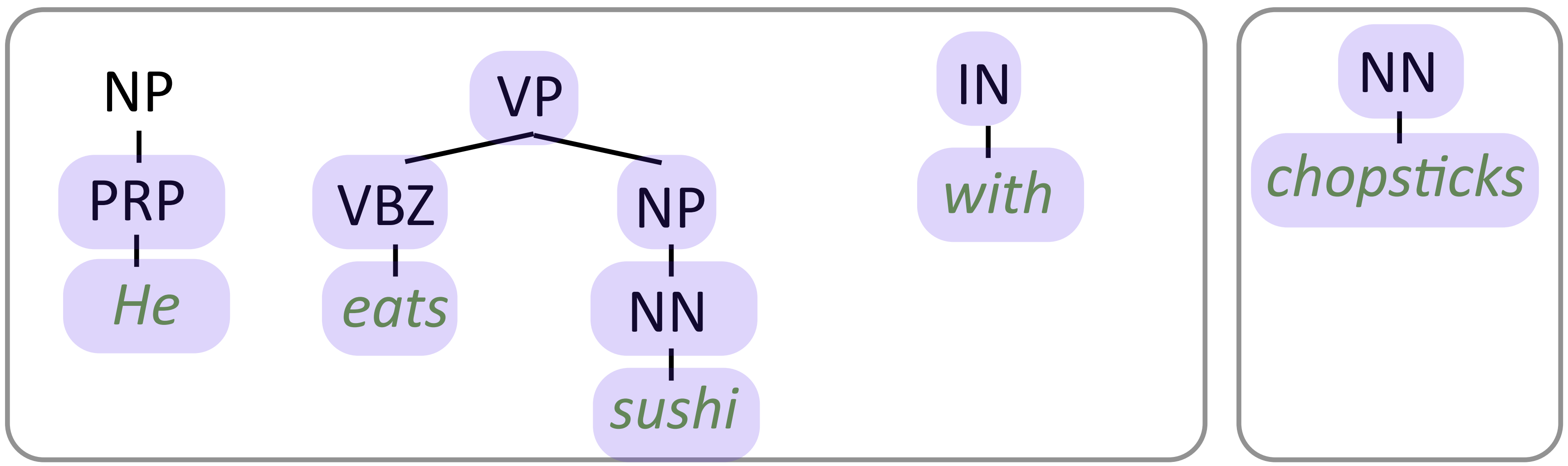


- Remembering head is expensive
- Deeper stack element is expensive

- The span around a subtree is cheap

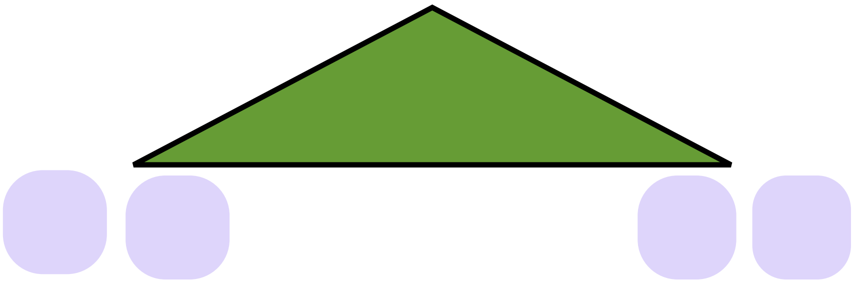


Recipe 1: Utilizing cheap features

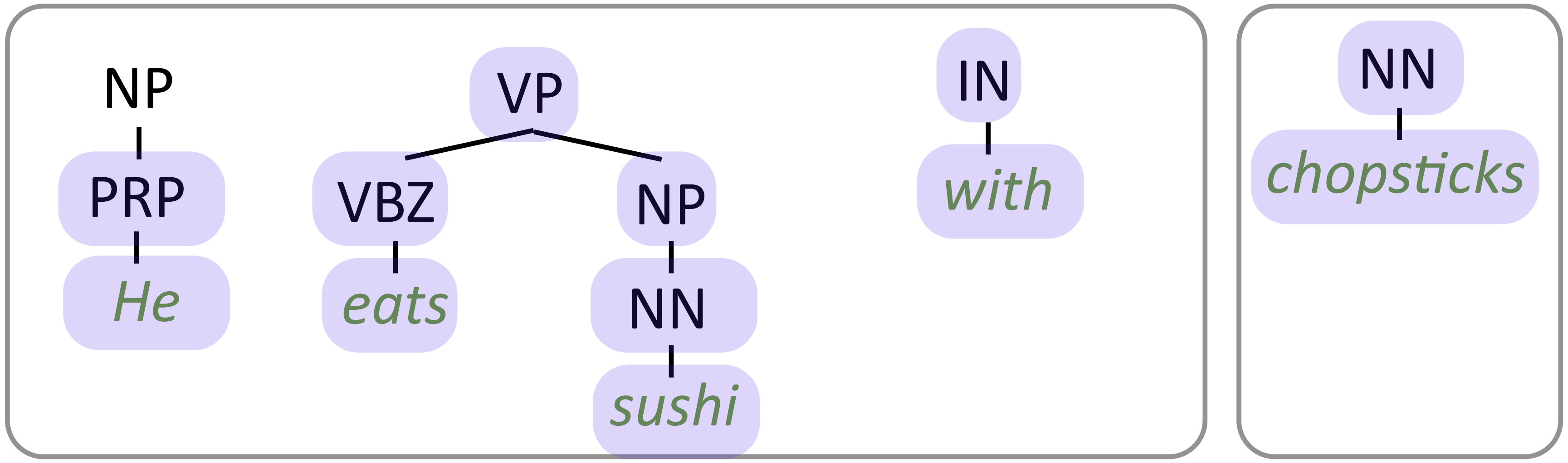


- Remembering head is expensive
- Deeper stack element is expensive

- The span around a subtree is cheap

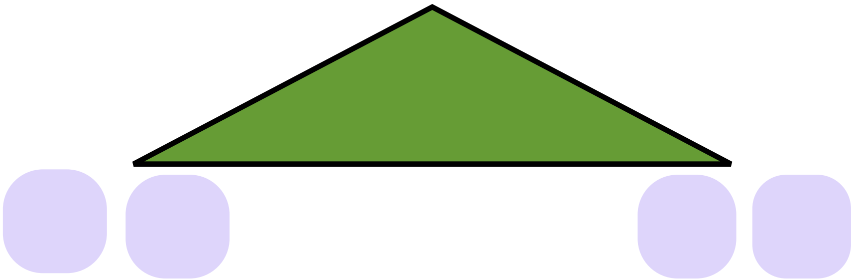


Recipe 1: Utilizing cheap features



- Remembering head is expensive
- Deeper stack element is expensive

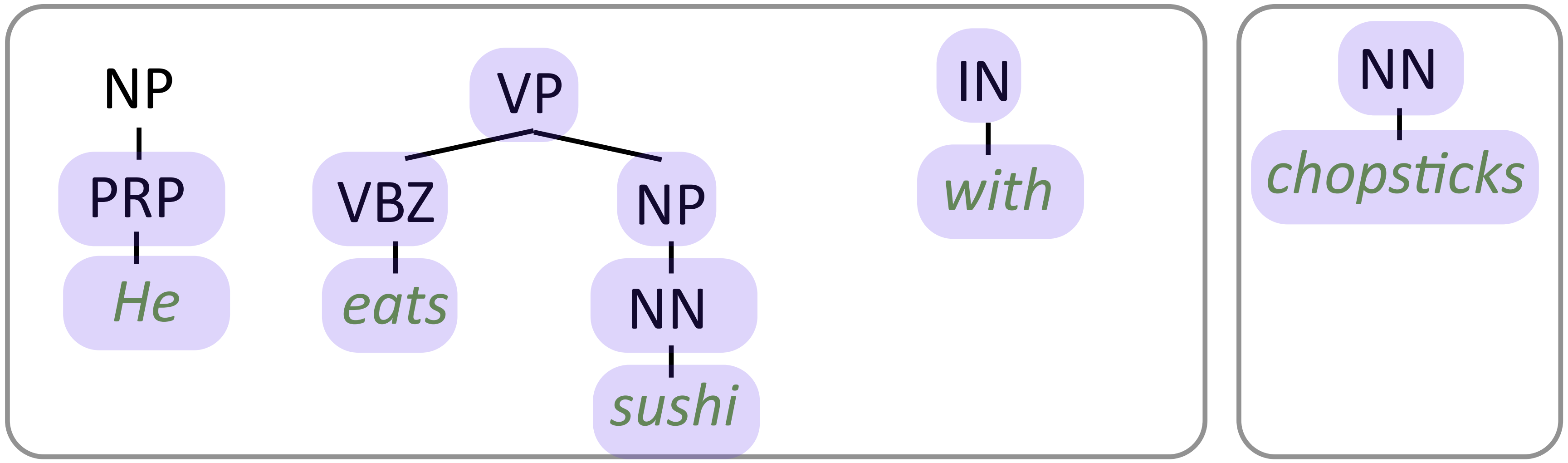
- The span around a subtree is cheap



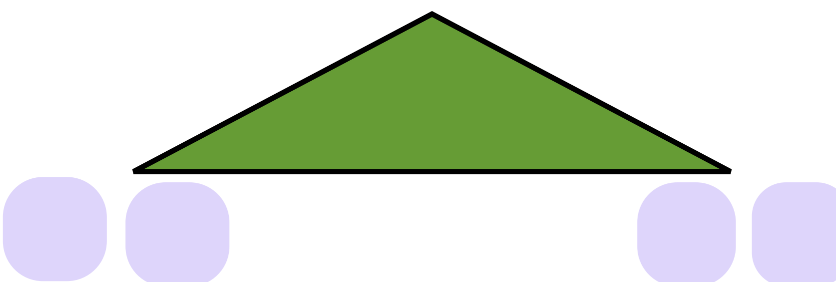
⇒ Inspired by the recent CRF parser with span features

[Hall et al., 2014]

Recipe 1: Utilizing cheap features



- Remembering head is expensive
- Deeper stack element is expensive



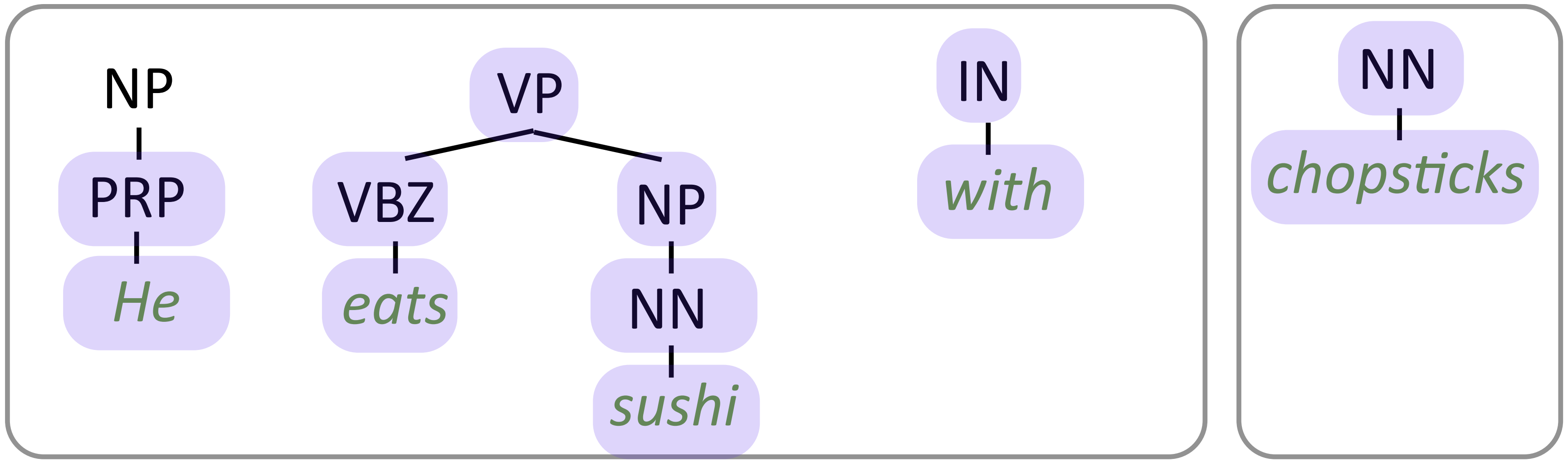
- The span around a subtree is cheap

  ⇒ Inspired by the recent CRF parser with span features

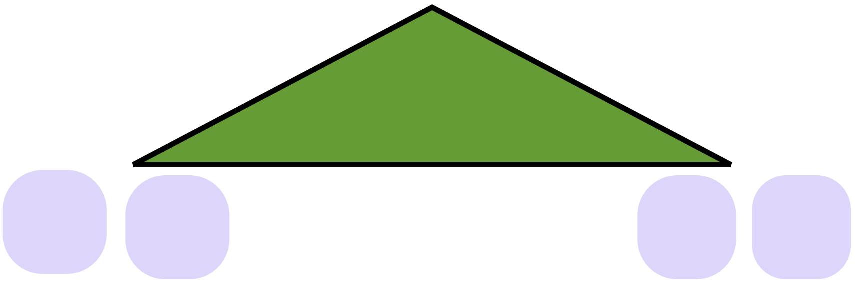
  [Hall et al., 2014]

- Worst time complexity: $O(n^4|G|^3|N|)$

Recipe 1: Utilizing cheap features



- Remembering head is expensive
- Deeper stack element is expensive



- The span around a subtree is cheap

  ⇒ Inspired by the recent CRF parser with span features

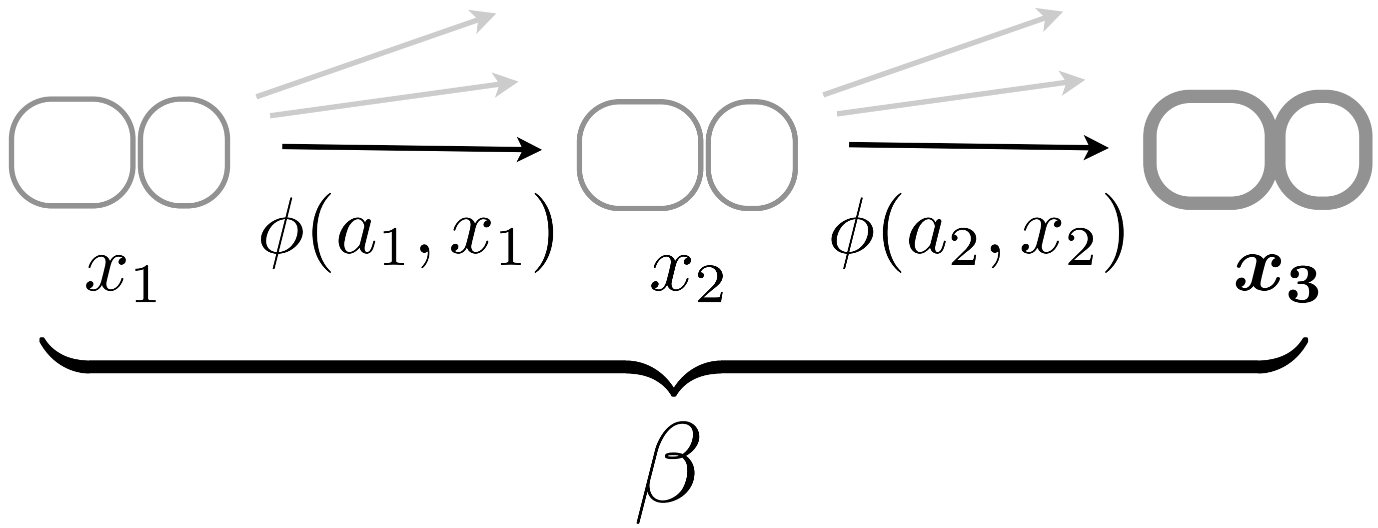
  [Hall et al., 2014]

- Worst time complexity: $O(n^4|G|^3|N|)$

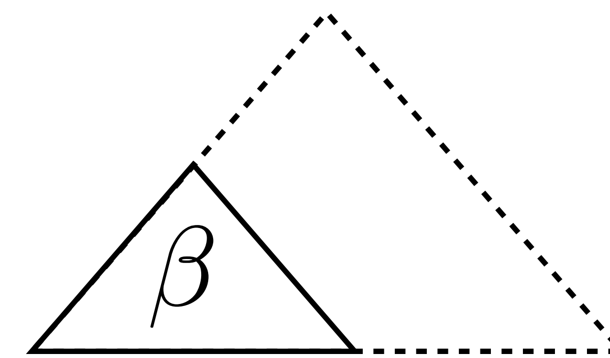
  ⇒ Improved but is still expensive...

Recipe 2: A* search

$\phi(a, x)$: edge cost

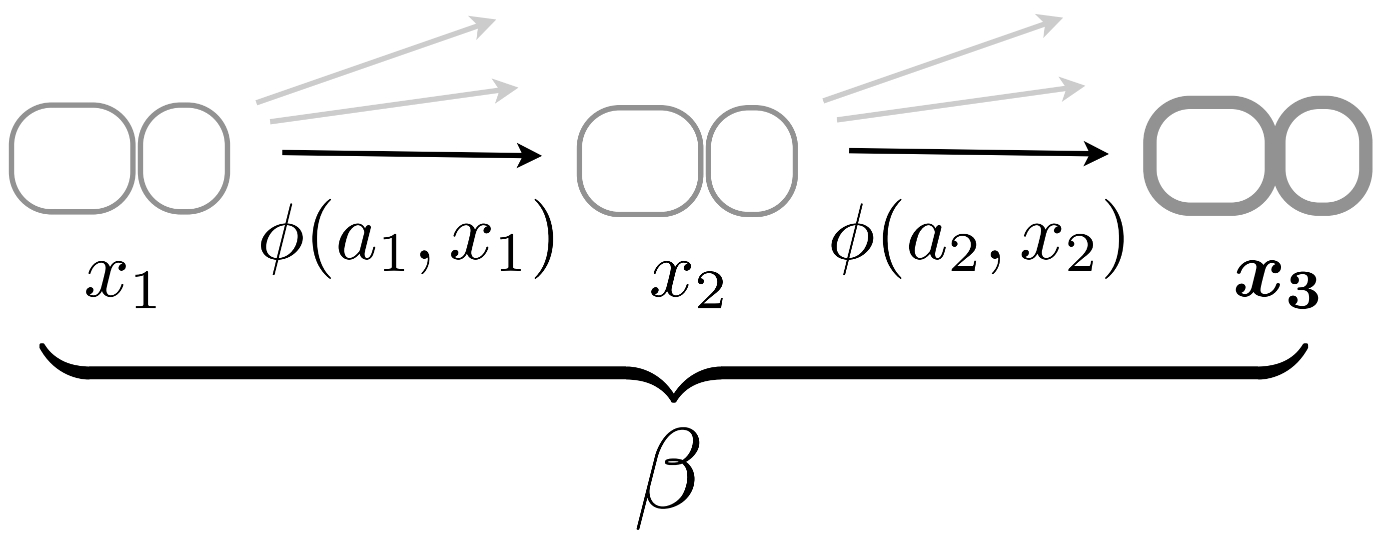


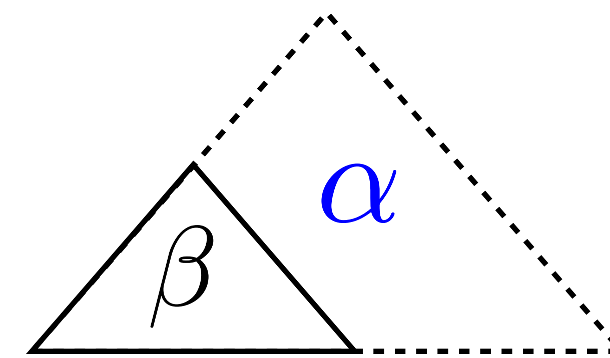
- Modify the state cost: β



Recipe 2: A* search

$\phi(a, x)$: edge cost

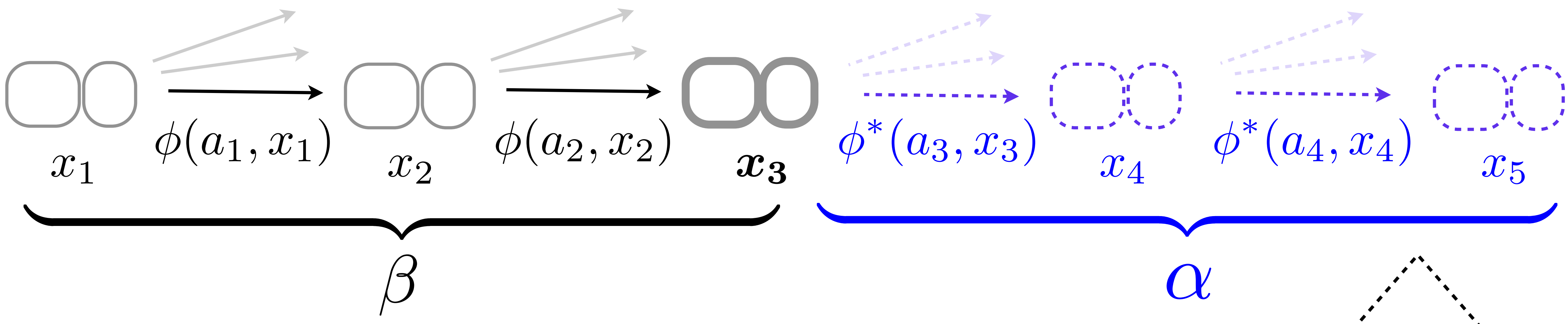


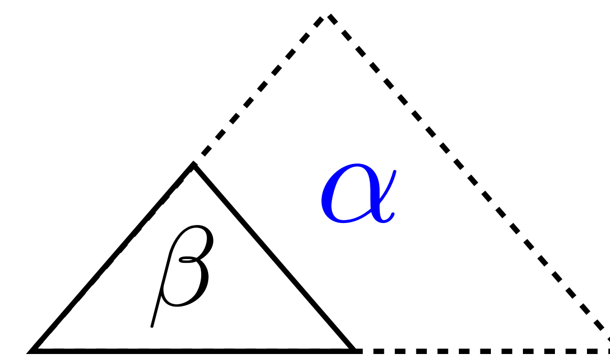


- Modify the state cost: $\beta + \alpha$
- α: approximation of the cost outside the current parse

Recipe 2: A* search

$\phi(a, x)$: edge cost

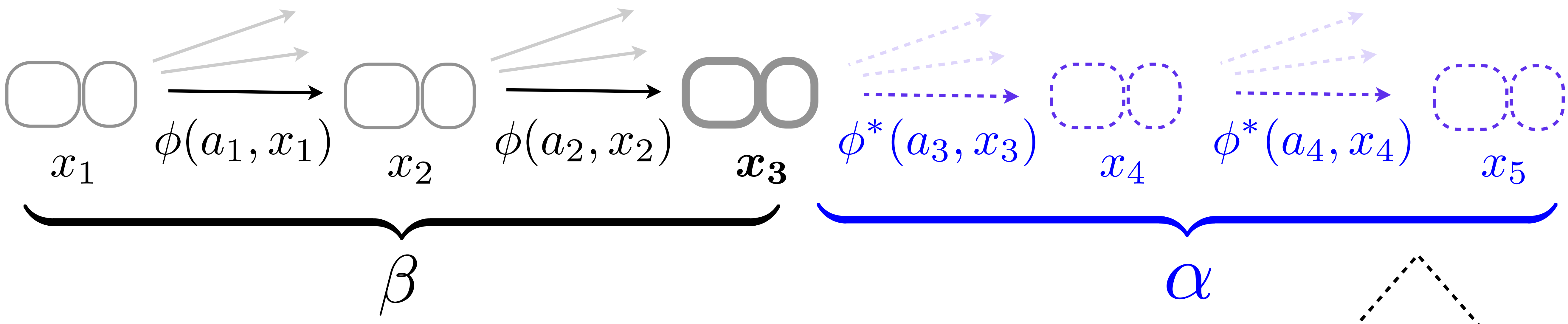


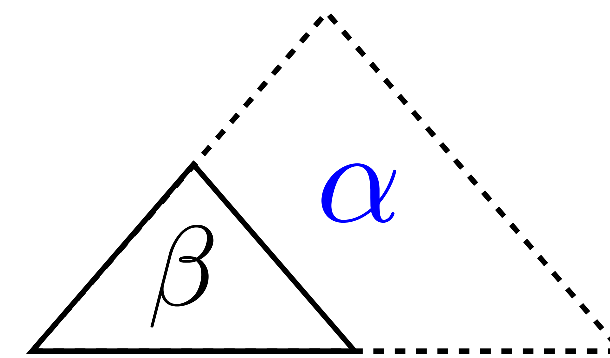


- Modify the state cost: $\beta + \alpha$
- α: approximation of the cost outside the current parse

Recipe 2: A* search

$\phi(a, x)$: edge cost

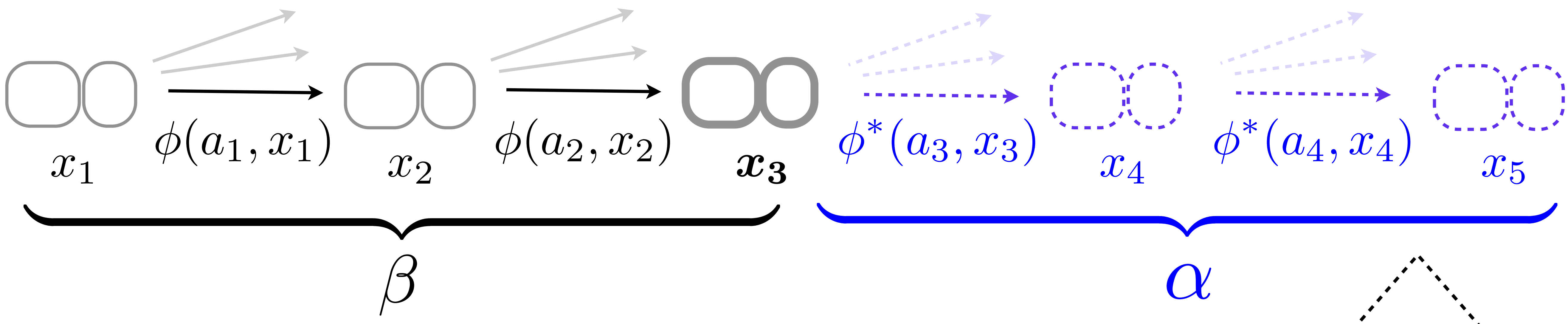


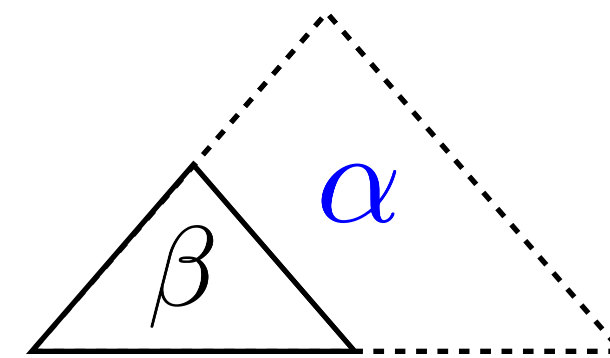


- Modify the state cost: $\beta + \alpha$
- α: approximation of the cost outside the current parse
  - $\Rightarrow$ If approximation is good, search is accelerated

Recipe 2: A* search

$\phi(a, x)$: edge cost





- Modify the state cost: $\beta + \alpha$
- α: approximation of the cost outside the current parse

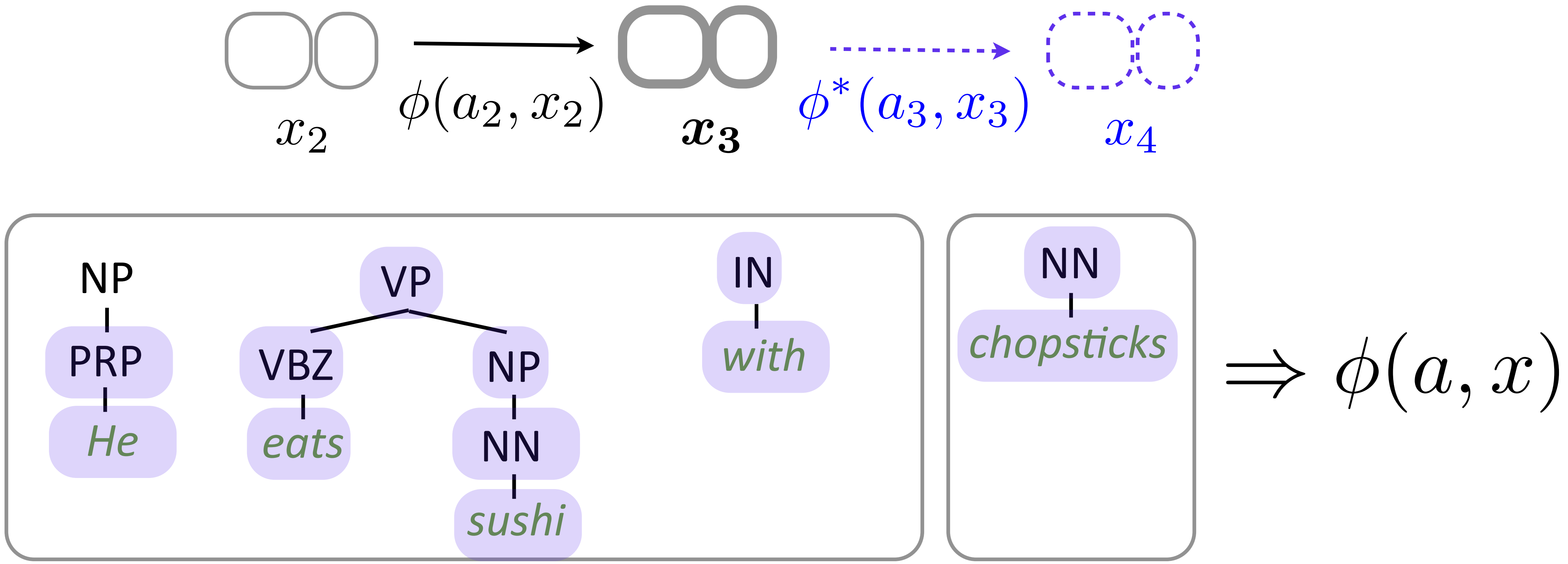
  ⇒ If approximation is good, search is accelerated

Constraint for optimality:

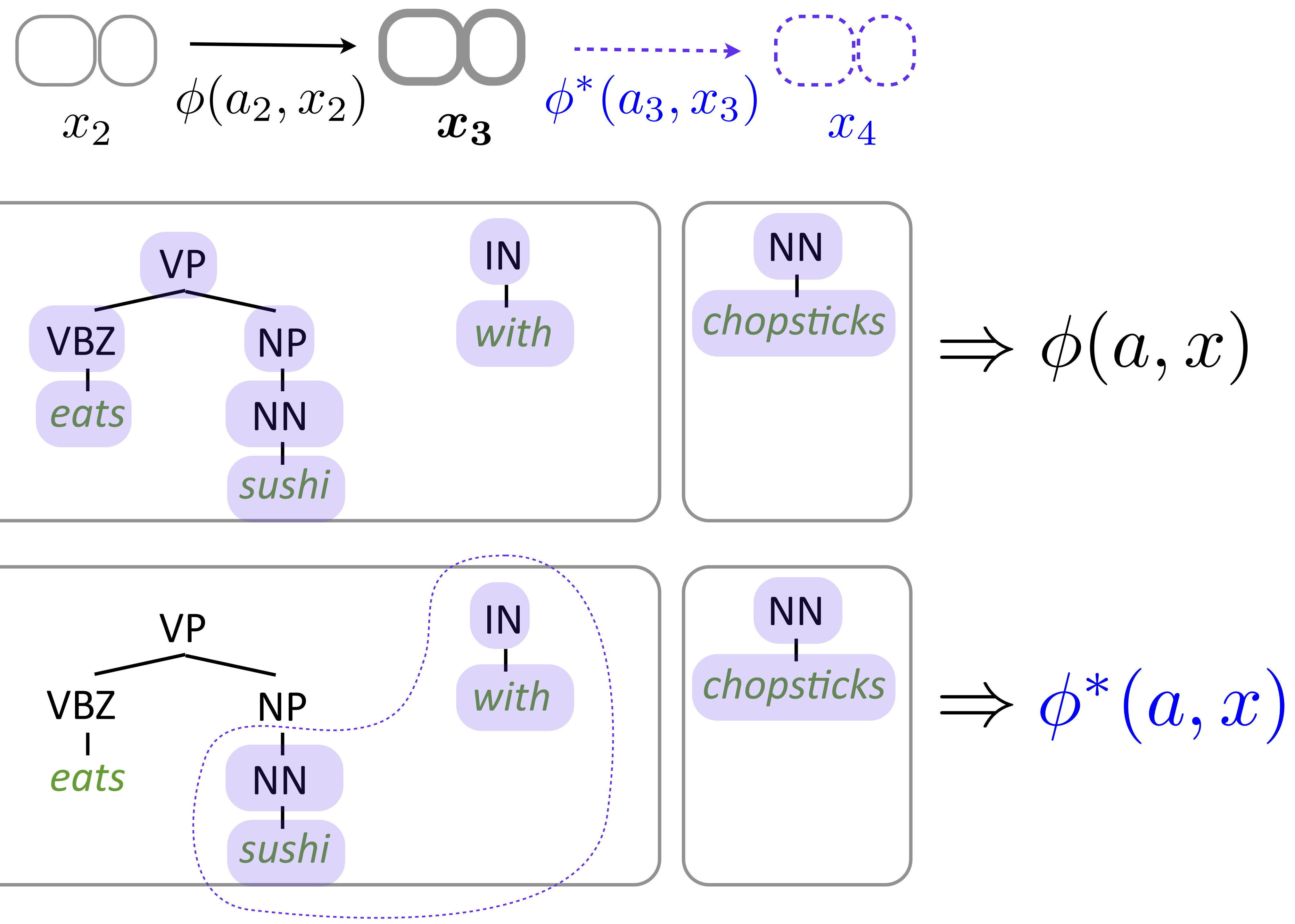
α must be a lower bound of actual outside cost

$$\phi(a, x) \geq \phi^*(a, x)$$

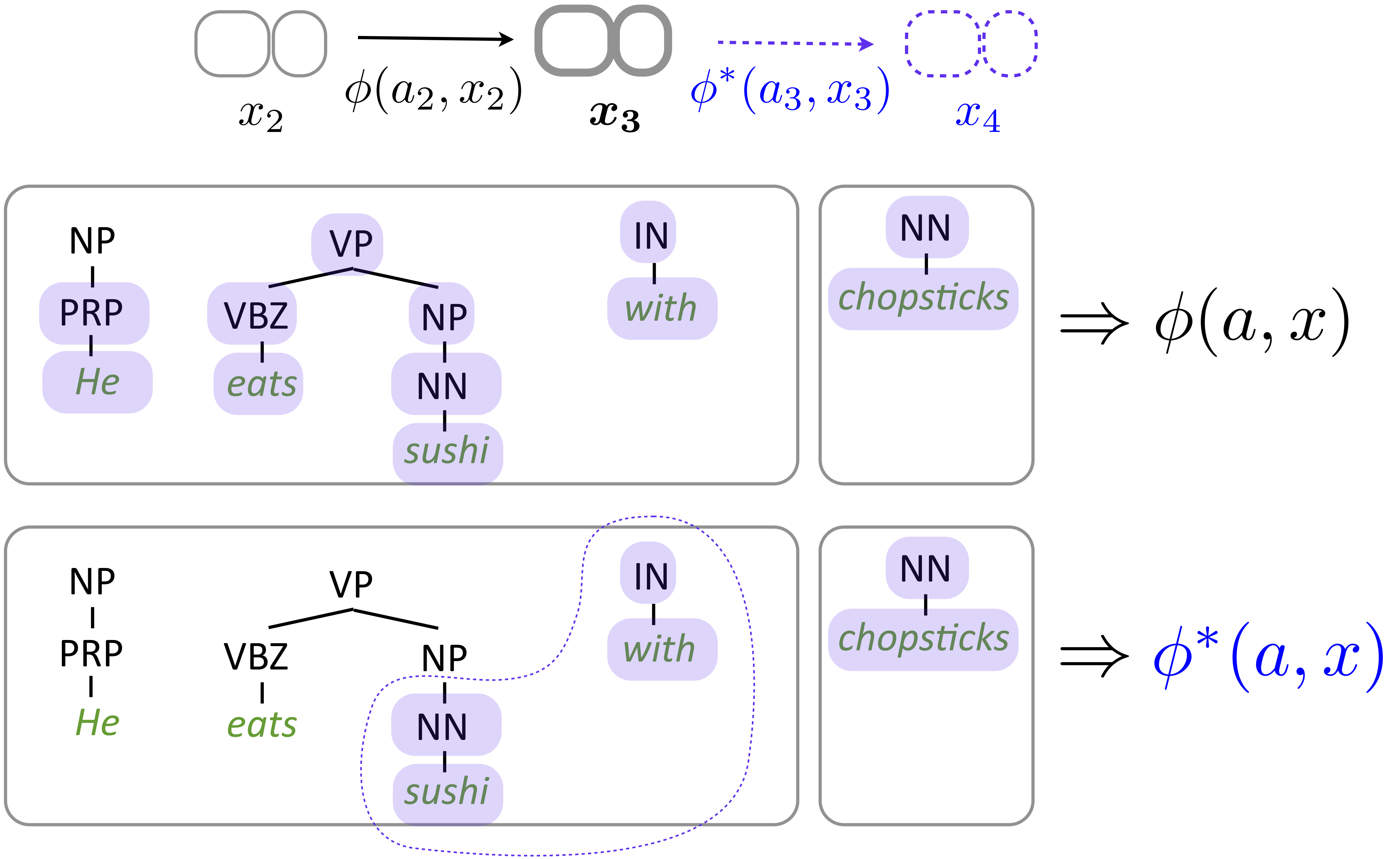
Idea: Parsing with simpler features



Idea: Parsing with simpler features

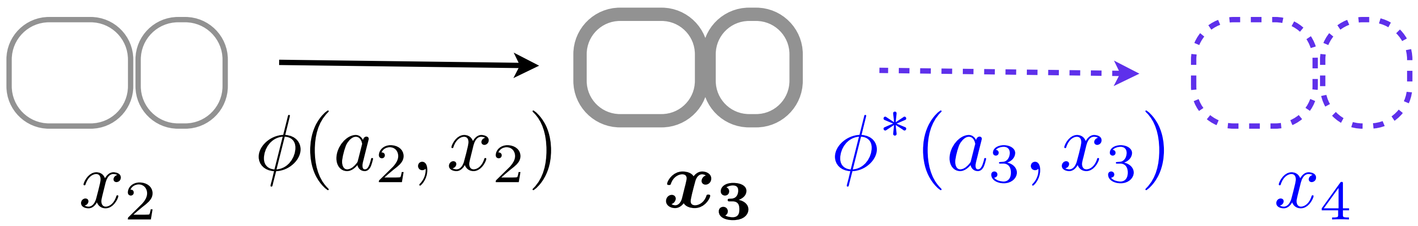


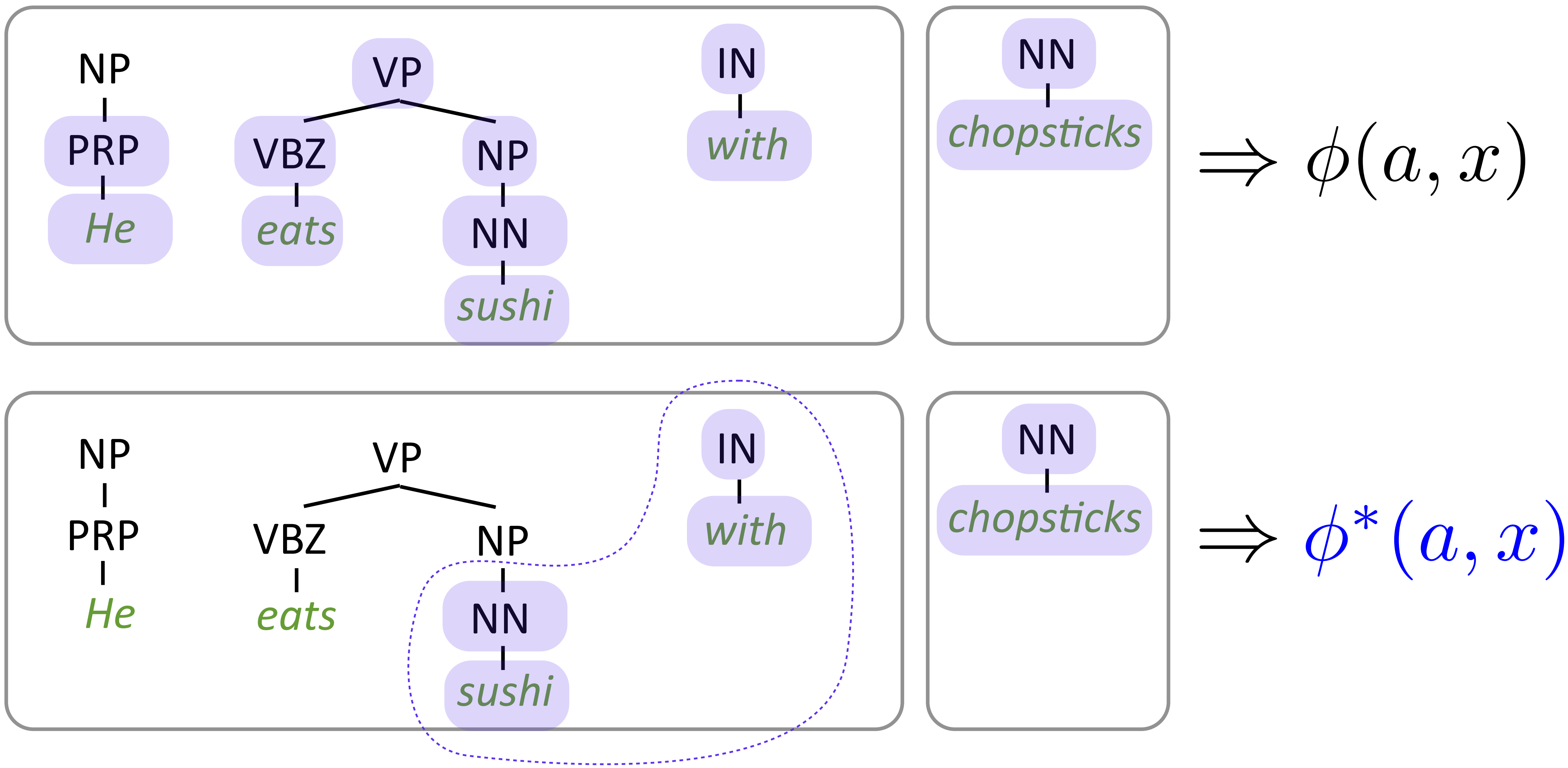
Idea: Parsing with simpler features



- Time complexity to calculate heuristic cost: $O(n^3|G|)$

Idea: Parsing with simpler features

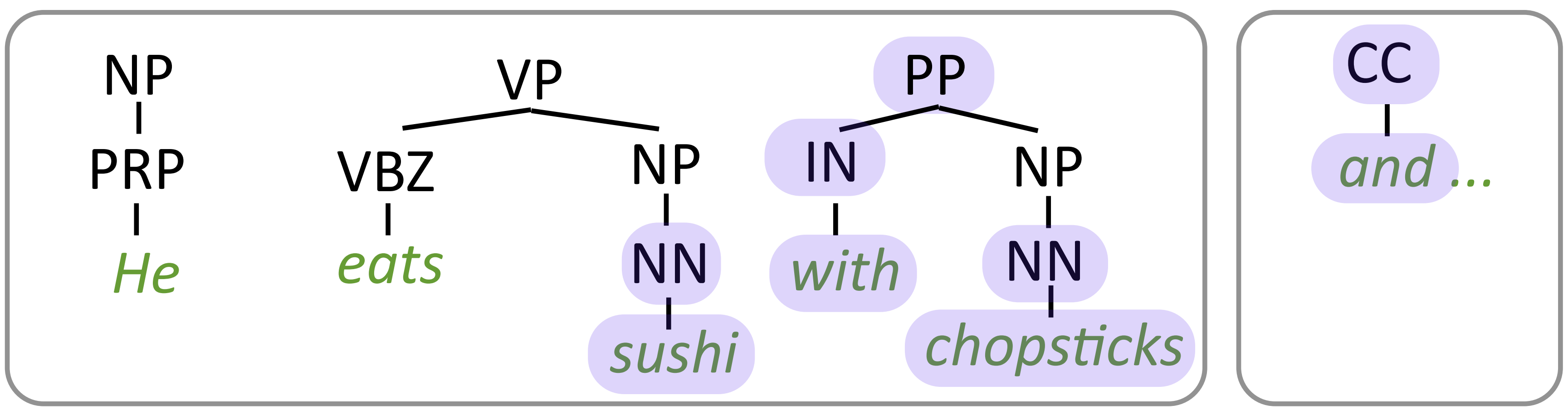




- Time complexity to calculate heuristic cost: $O(n^3|G|)$
- We can calculate $\phi^*(a|x)$ to always underestimate the true cost

Applying hierarchical A*

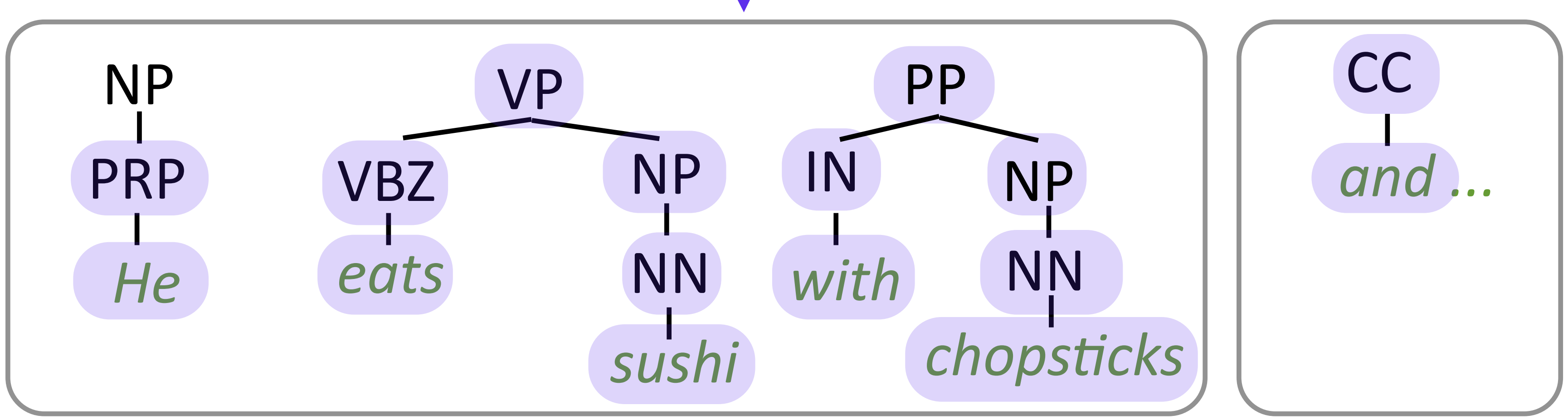
[Pauls & Klein, 2009]



$O(n^3|G|)$

still a bit expensive

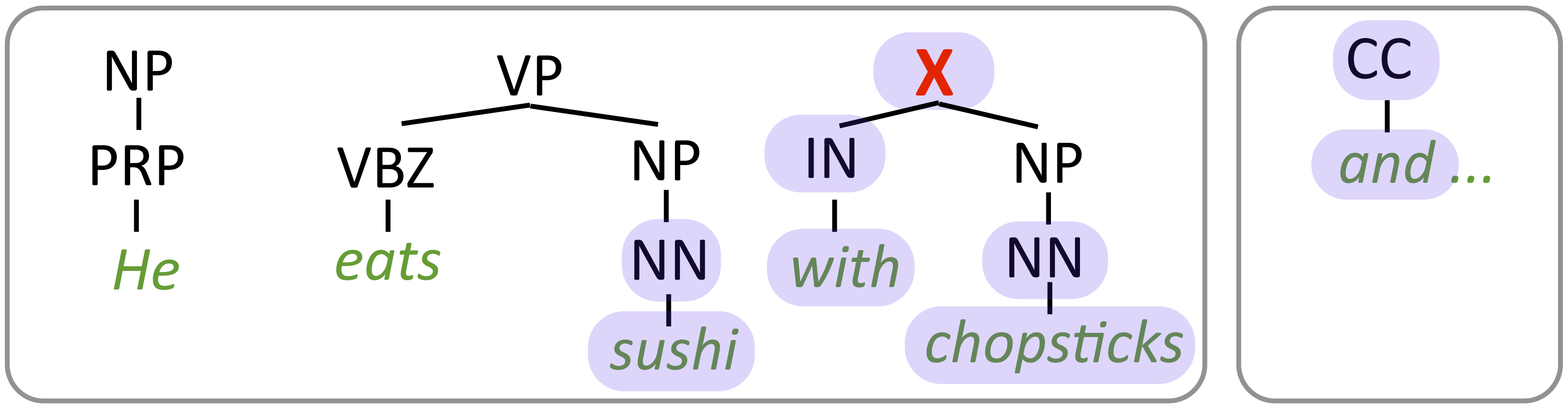
Simplify features



$O(n^4|G|^3|N|)$

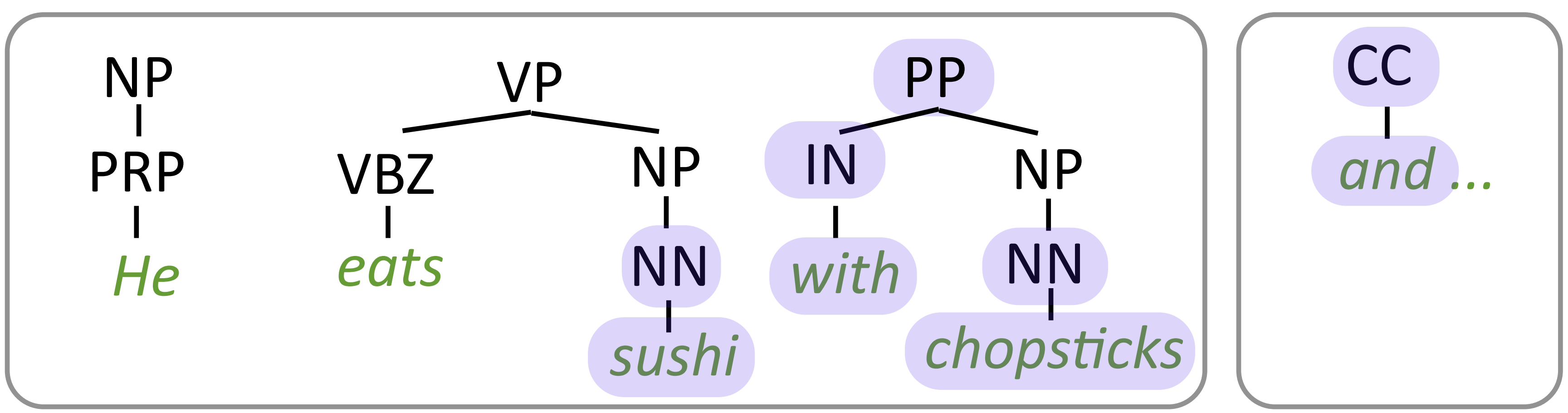
Applying hierarchical A*

[Pauls & Klein, 2009]



$O(n^3)$

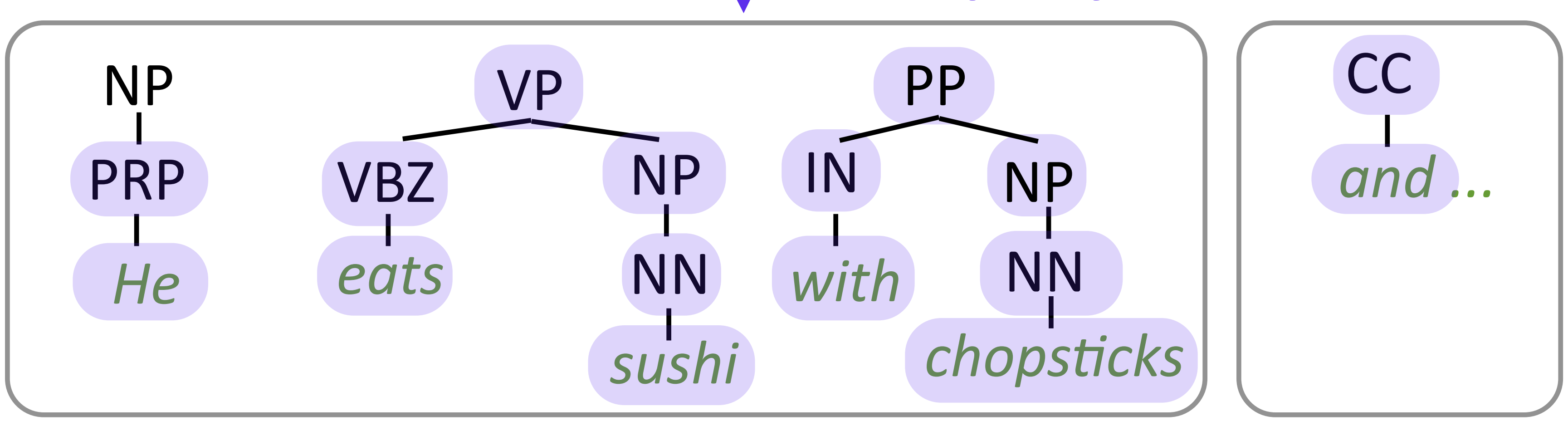
Simplify grammar [Klein & Manning, 2003]



$O(n^3|G|)$

still a bit expensive

Simplify features



$O(n^4|G|^3|N|)$

Outline

DP best-first shift-reduce parsing

- for **constituency**
- with **structured perceptron**

MaxEnt vs. Structured perceptron

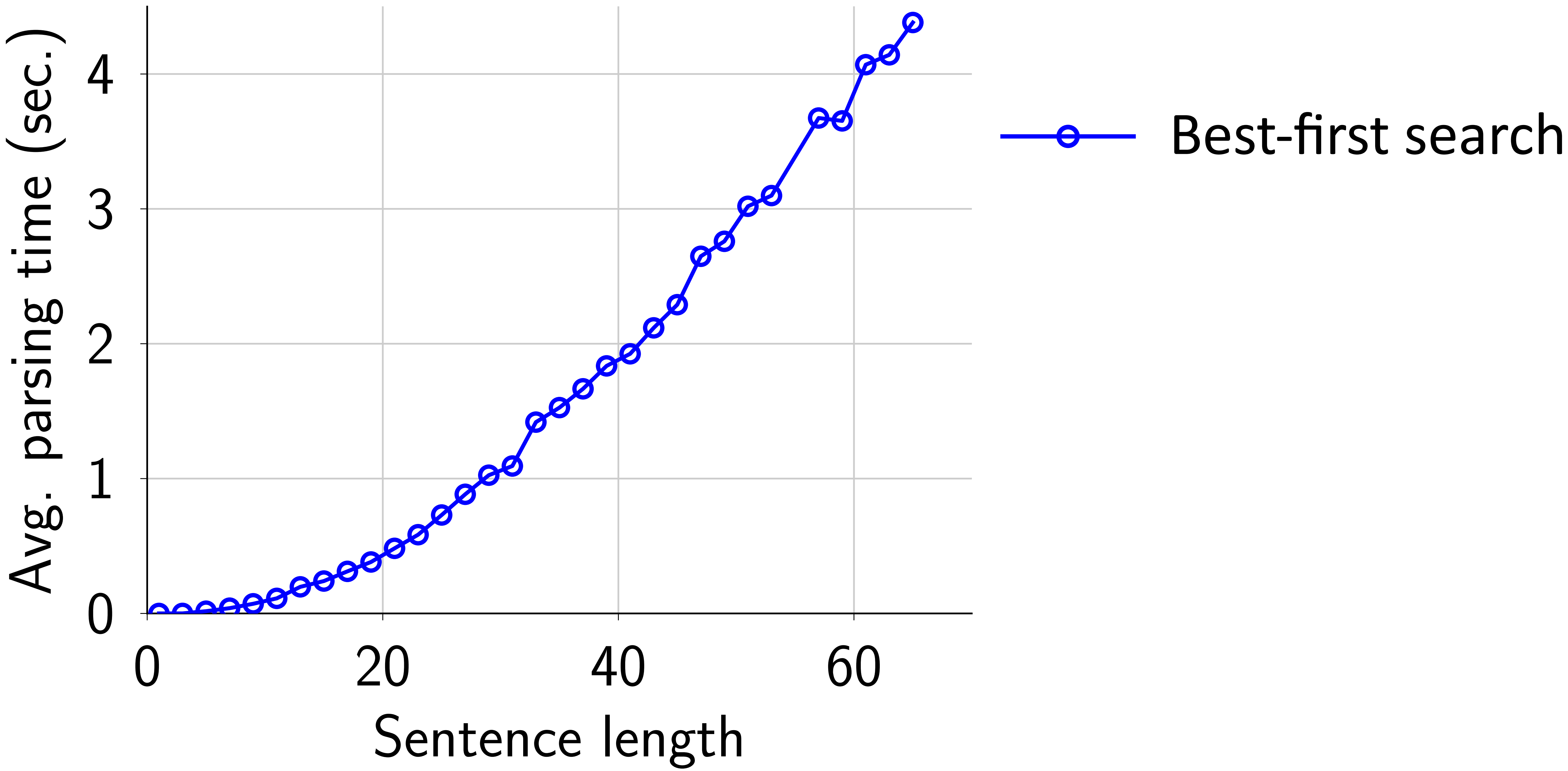
- Structured perceptron is strong
- but its search is much harder

Improving search efficiency of structured perceptron

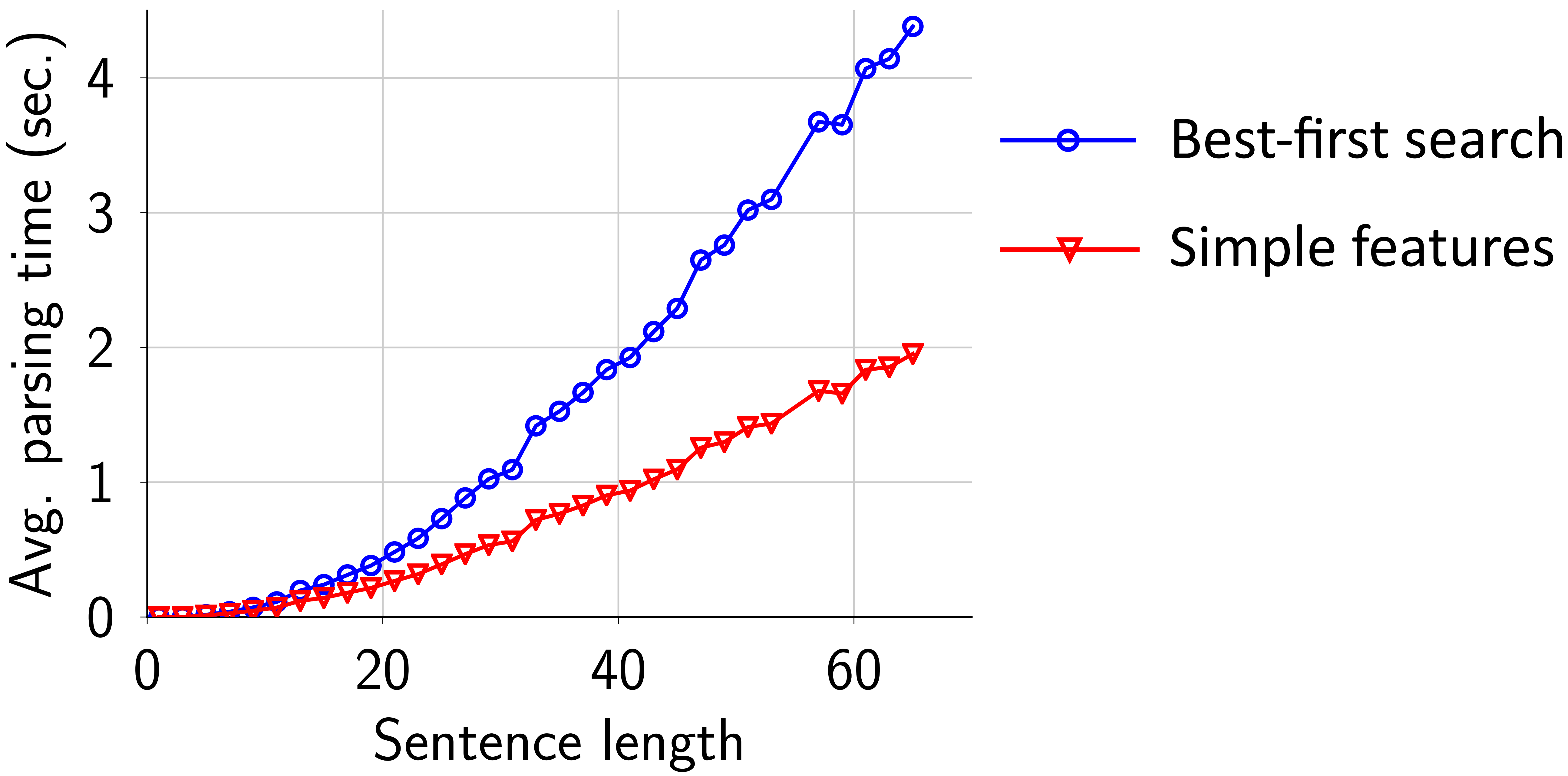
- New feature templates
- A* search

Final experiment

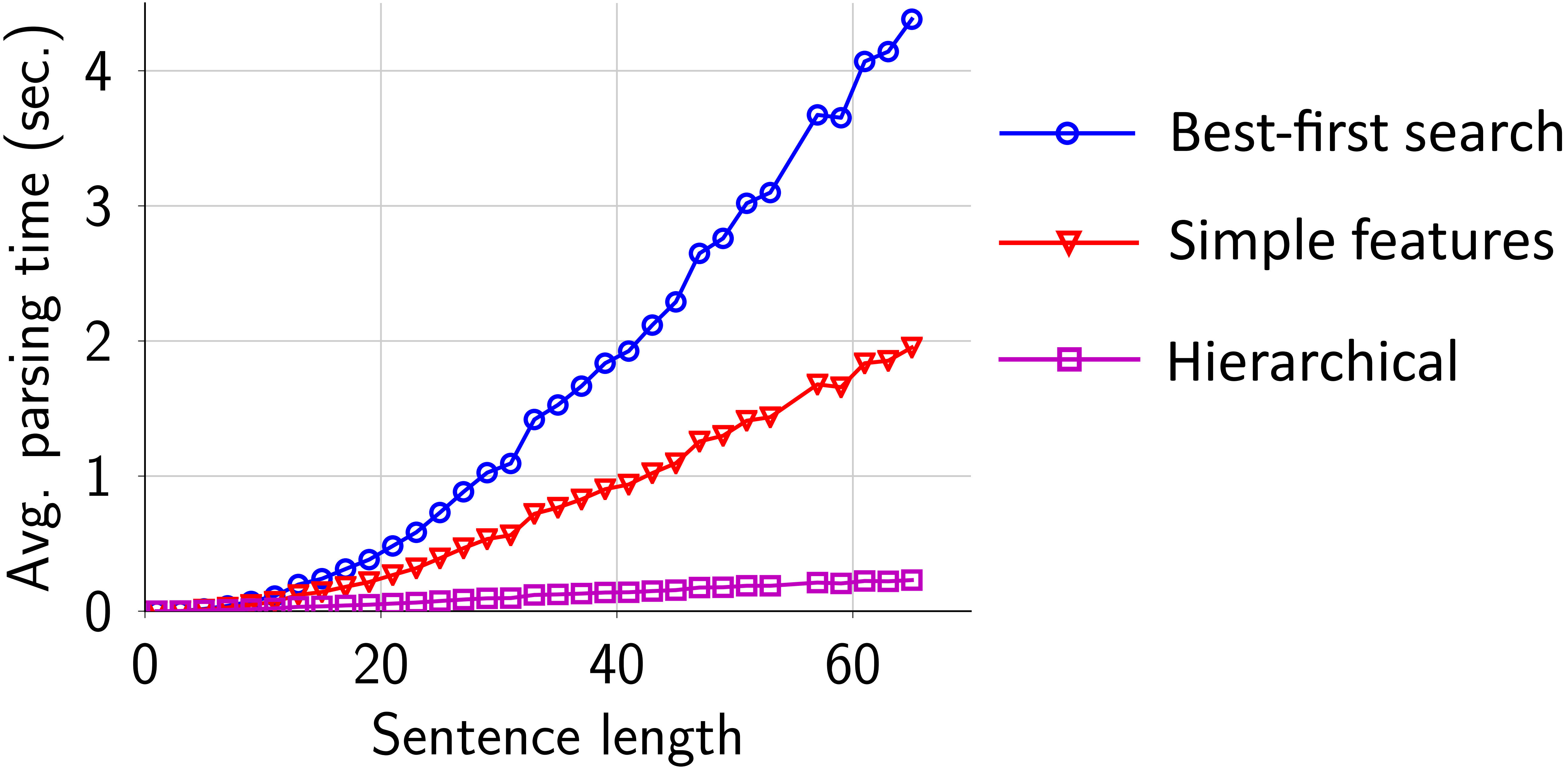
Hierarchical A* works quite well



Hierarchical A* works quite well



Hierarchical A* works quite well



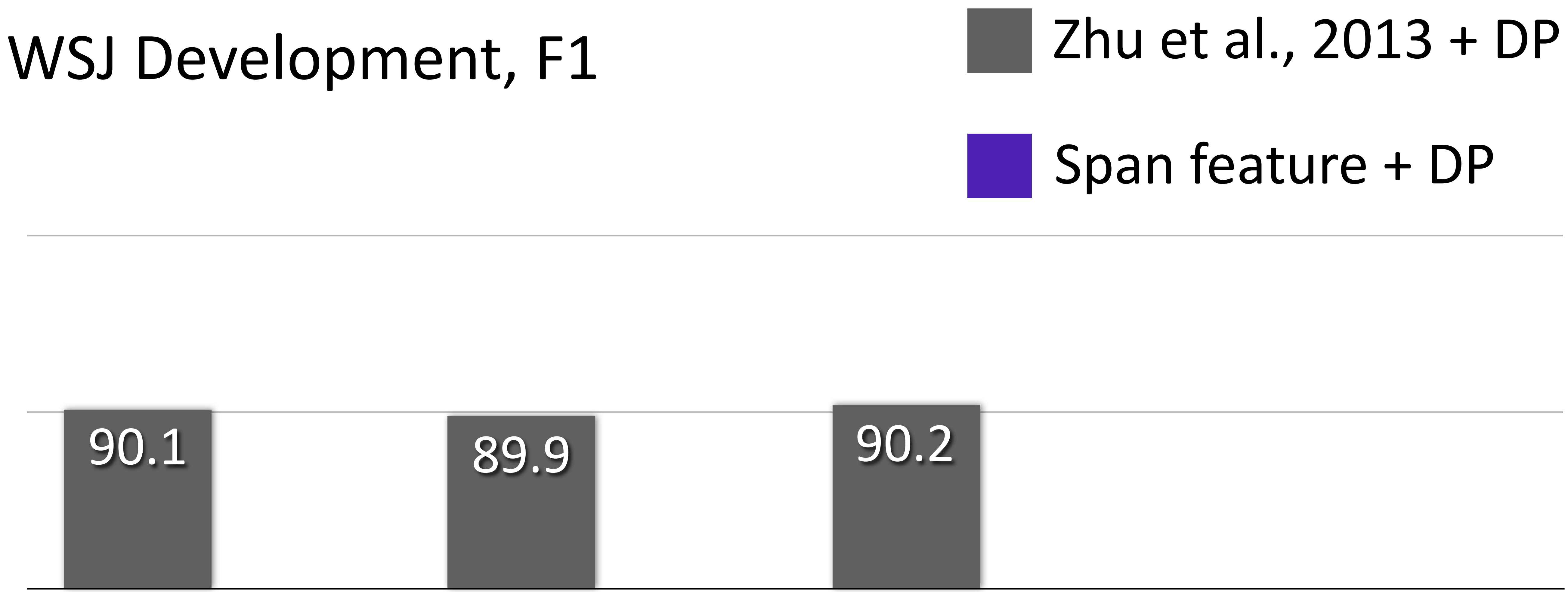
Effect of span features

WSJ Development, F1

Zhu et al., 2013 + DP

Span feature + DP

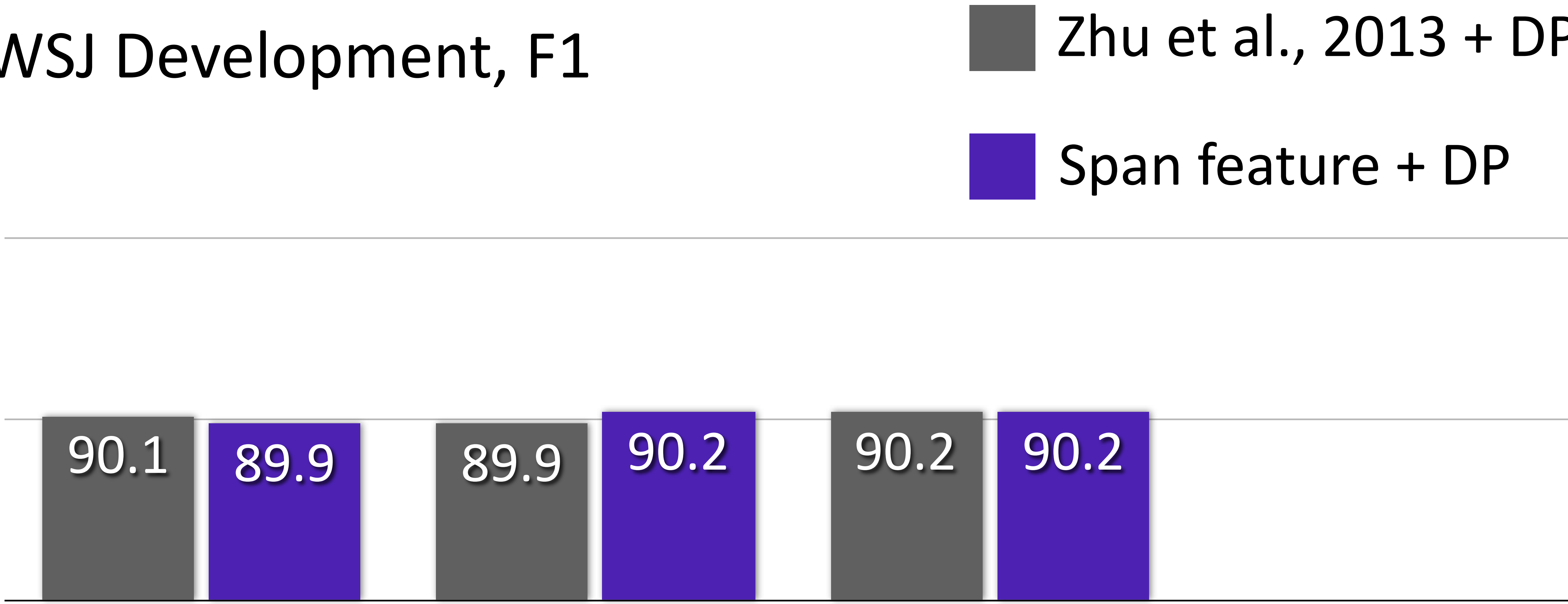
Effect of span features



beam:	16	32	64
Sent./s.:	31.9	17.0	9.1

Effect of span features

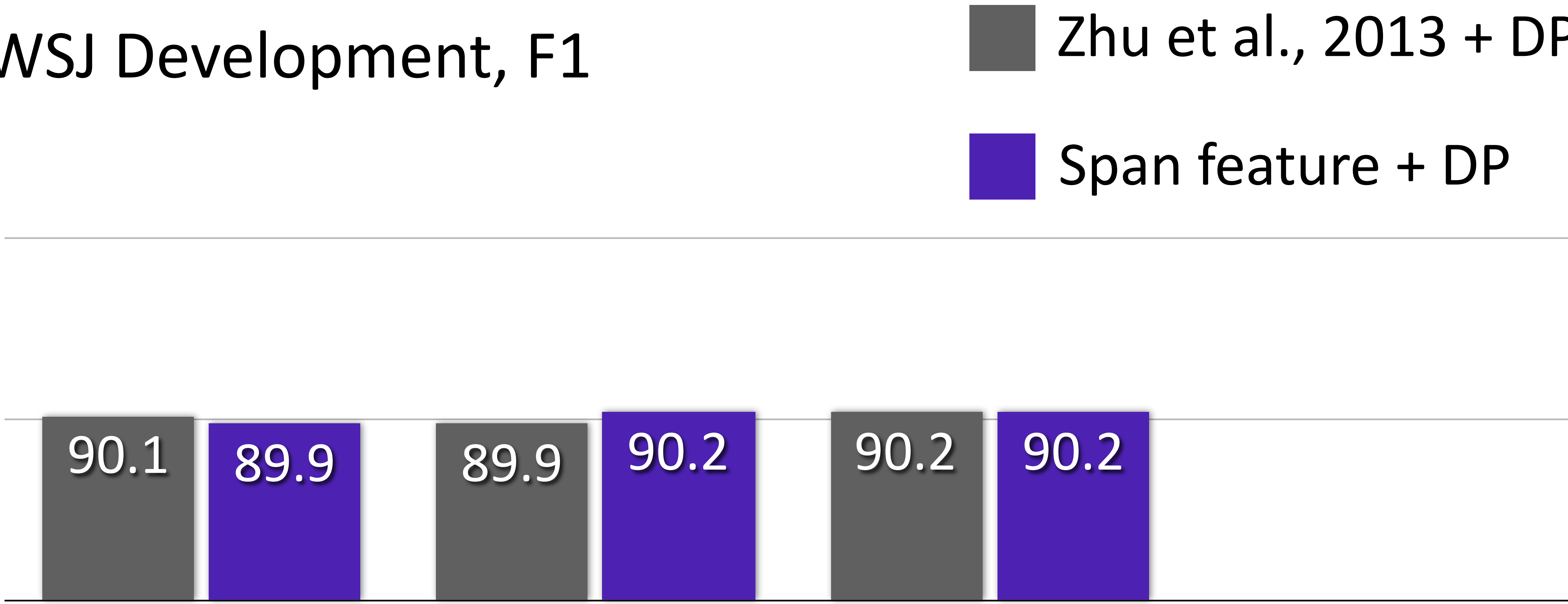
WSJ Development, F1



beam:	16	32	64
Sent./s.:	31.9	17.0	9.1

Effect of span features

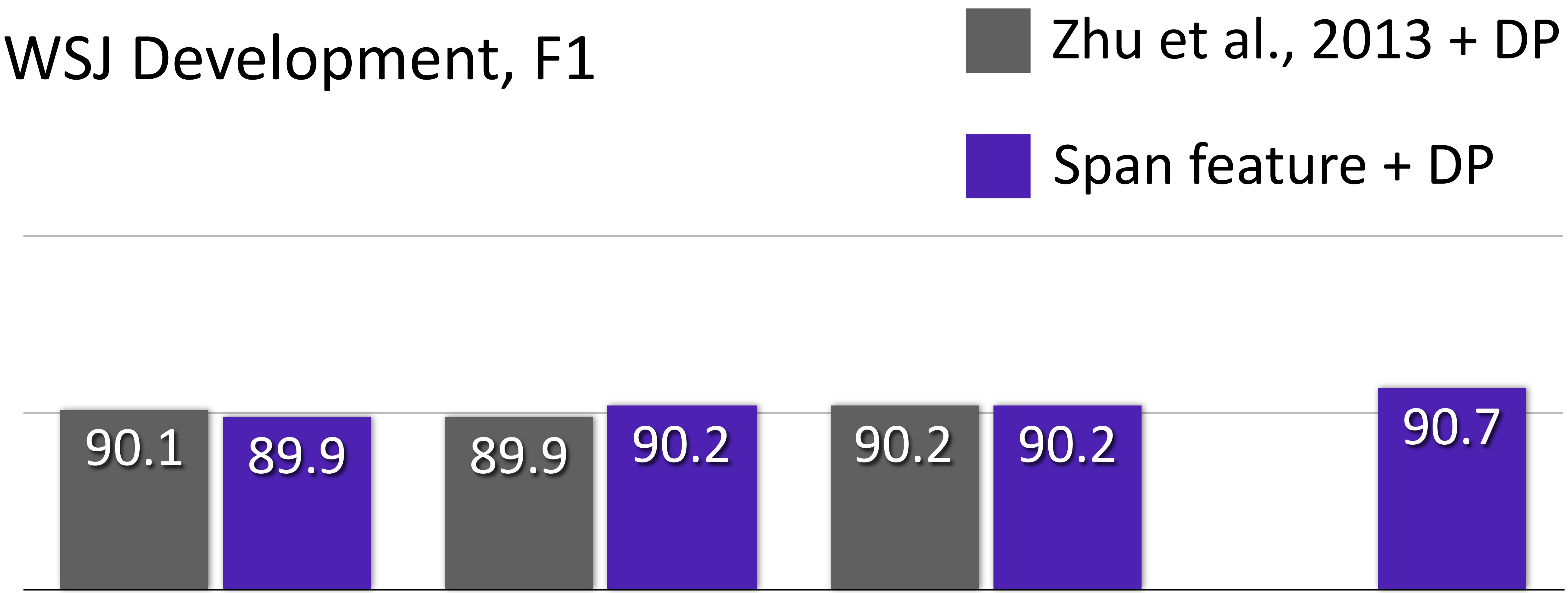
WSJ Development, F1



beam:	16	32	64
Sent./s.:	31.9	17.0	9.1

- Span feature is competitive to the current state-of-the-art

Effect of span features



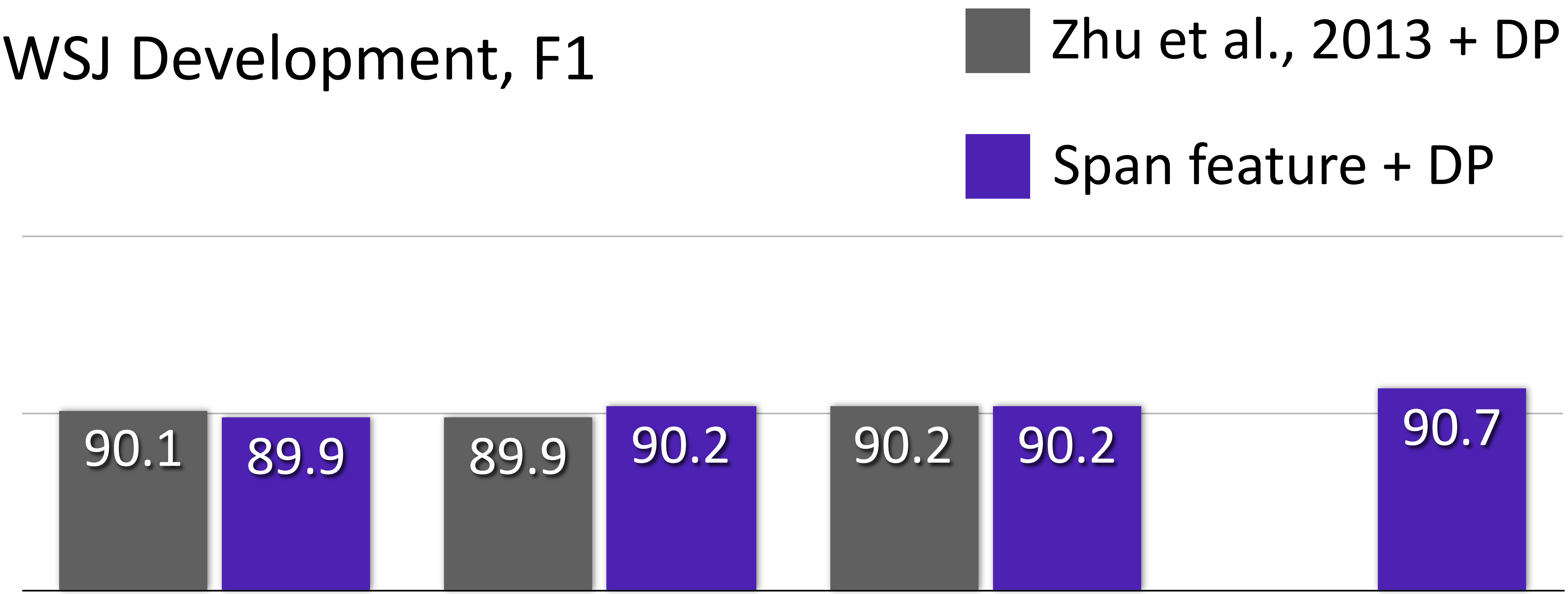
beam: 16 32 64 ∞

Sent./s.: 31.9 17.0 9.1 13.6

- Span feature is competitive to the current state-of-the-art

Effect of span features

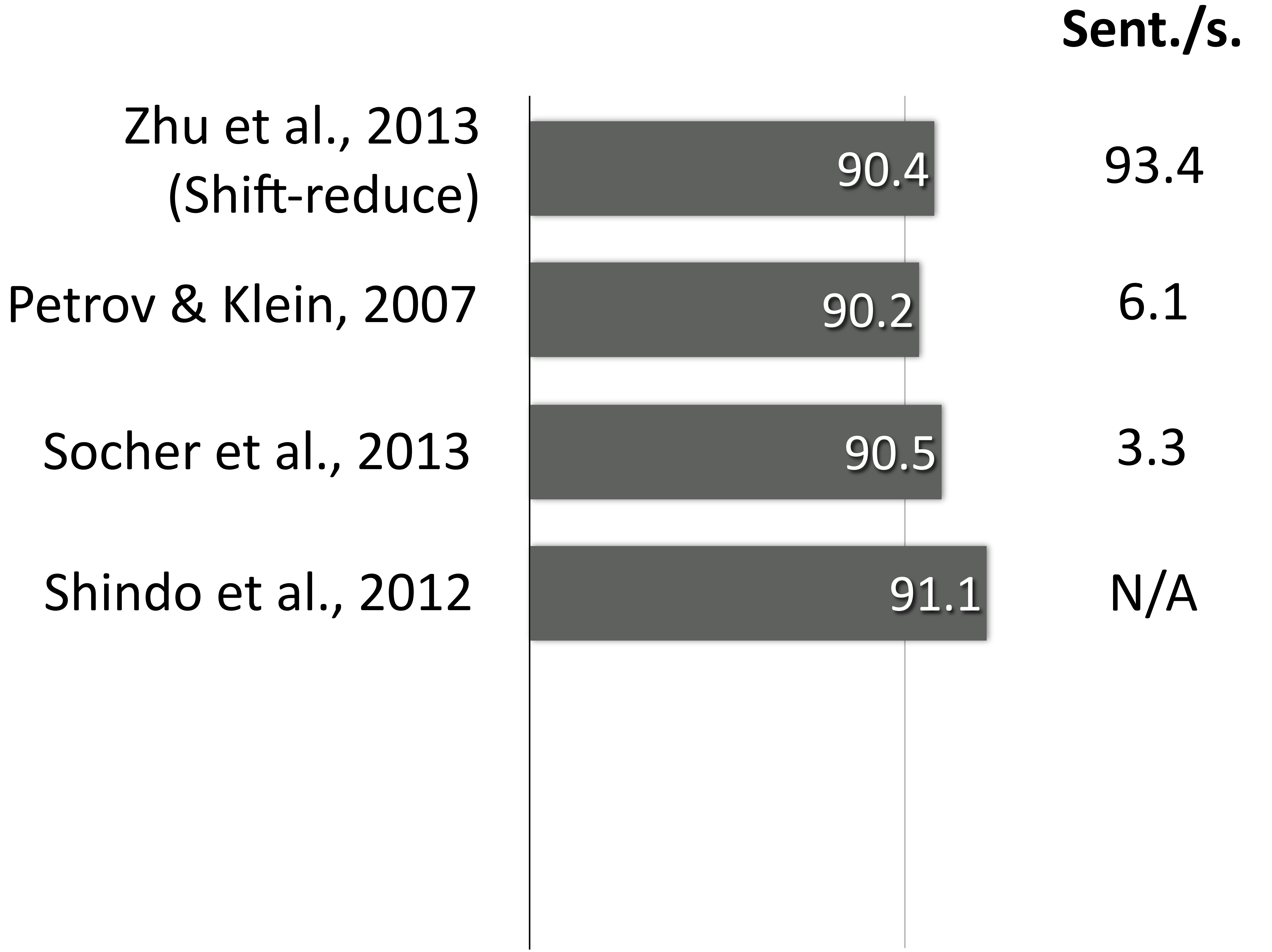
WSJ Development, F1



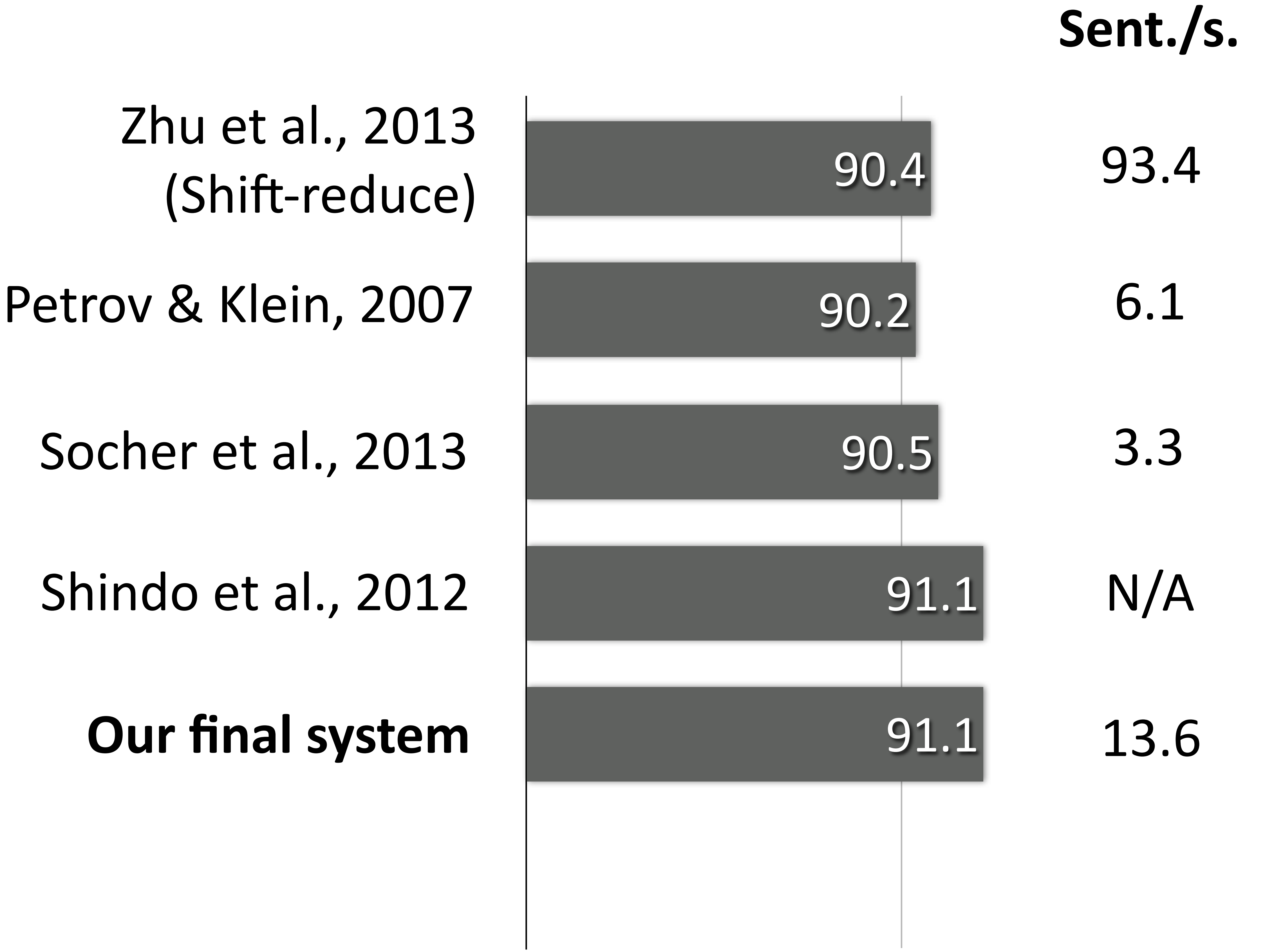
beam:	16	32	64	∞
Sent./s.:	31.9	17.0	9.1	13.6

- Span feature is competitive to the current state-of-the-art
- A* search gives the best score (faster than beam = 64)

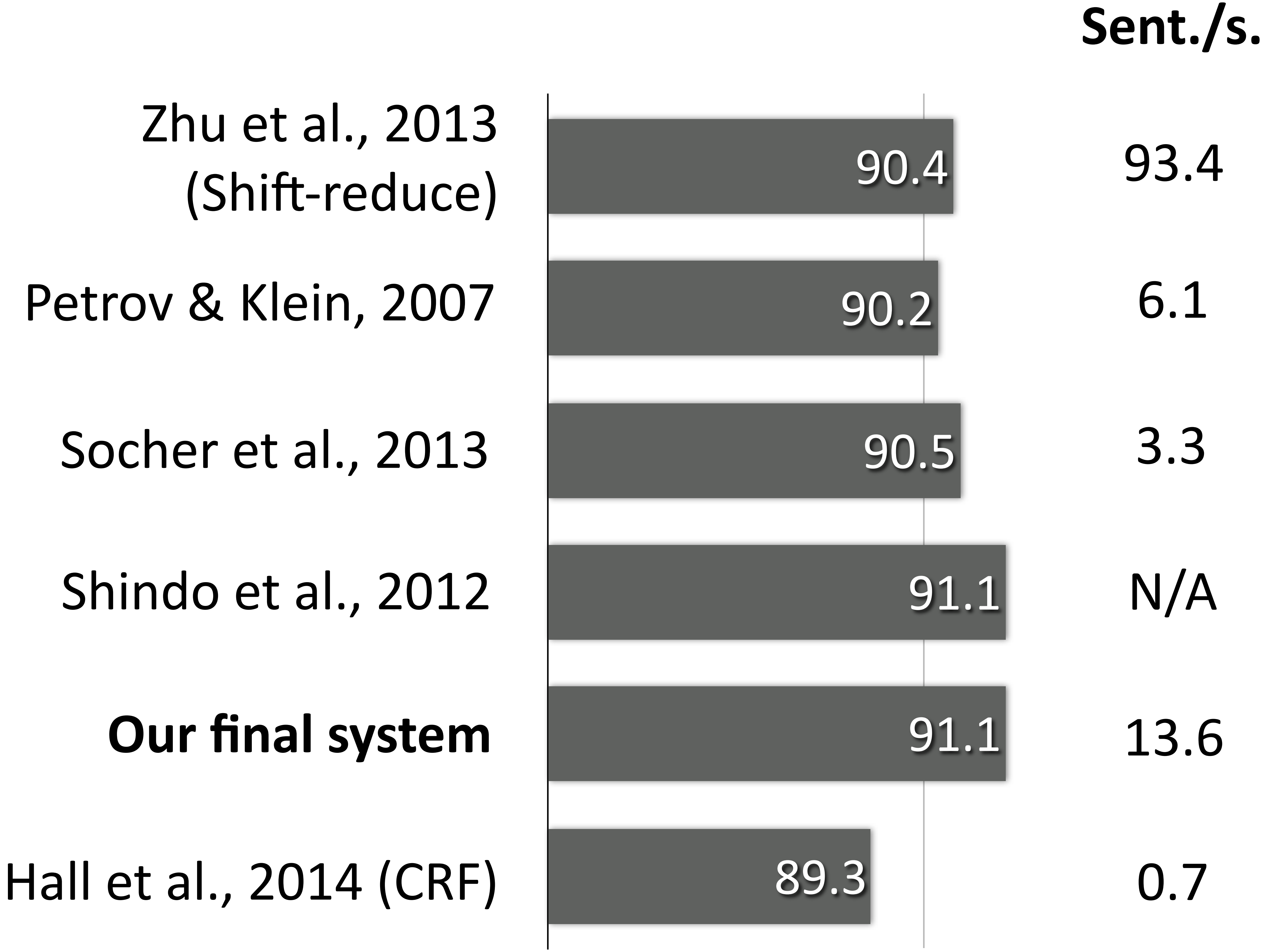
Comparison on WSJ test set



Comparison on WSJ test set

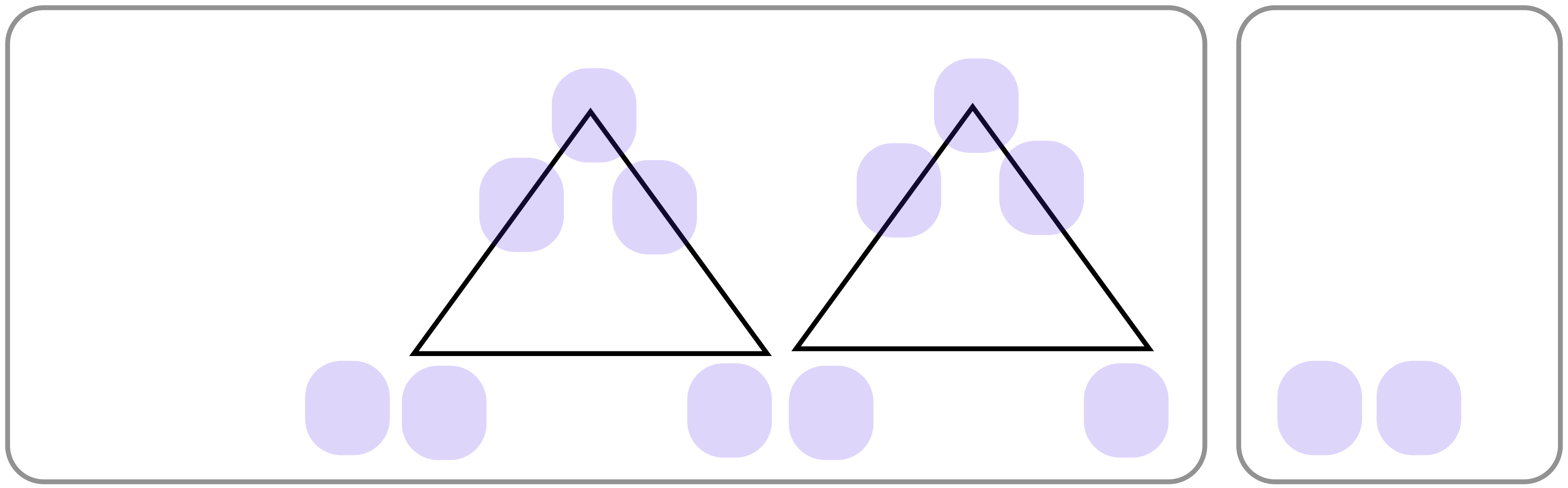


Comparison on WSJ test set



[Hall et al., 2014]

What's the difference with CRF parsing?



Our parser looks similar to the discriminative CKY (CRF):

- Features come from the top two subtrees
- but our parser outperforms the CRF parser (91.1 vs. 89.3)

What's the source of this difference?

- Scoring for shift actions?
- Or just because our parser utilizes more features?

Conclusion

Question:

Are beam-search & complex features the best strategy for shift-reduce parsing?

Our findings:

- It is not always the best in constituent parsing
- Practical shift-reduce parsing with exact search is possible

Our system is available at:

https://github.com/mynlp/optsr

The trained model is still unavailable (coming soon)