

# Lecture 11: Seq2Seq + Attention

Alan Ritter

(many slides from Greg Durrett)

# Recall: CNNs vs. LSTMs

---



the movie was good

# Recall: CNNs vs. LSTMs

---



c filters,  
m x k each



n x k

the movie was good

# Recall: CNNs vs. LSTMs

---



$O(n) \times c$



$c$  filters,  
 $m \times k$  each



$n \times k$

the movie was good

# Recall: CNNs vs. LSTMs

---



$O(n) \times c$

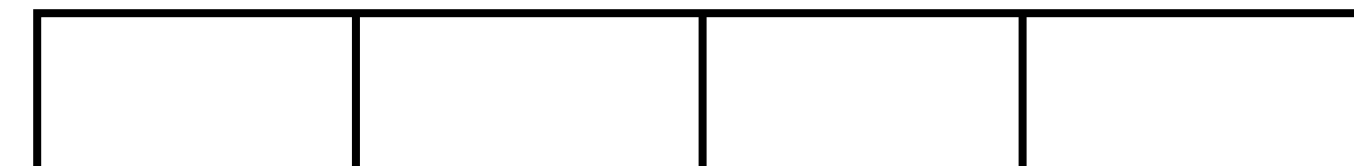


$c$  filters,  
 $m \times k$  each



$n \times k$

the movie was good



$n \times k$

the movie was good

# Recall: CNNs vs. LSTMs

---



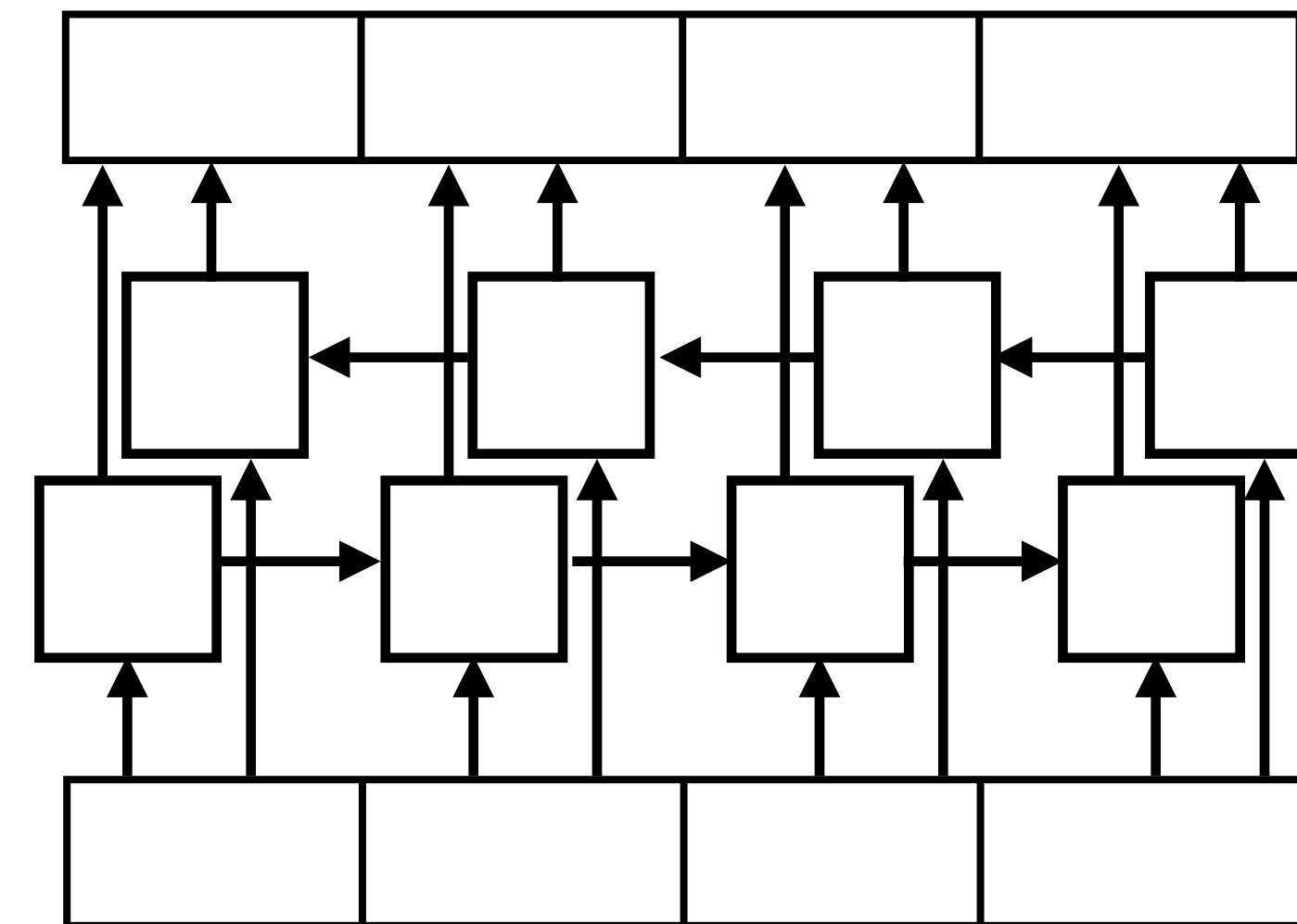
$O(n) \times c$

$c$  filters,  
 $m \times k$  each



$n \times k$

the movie was good



$n \times 2c$

BiLSTM with  
hidden size  $c$

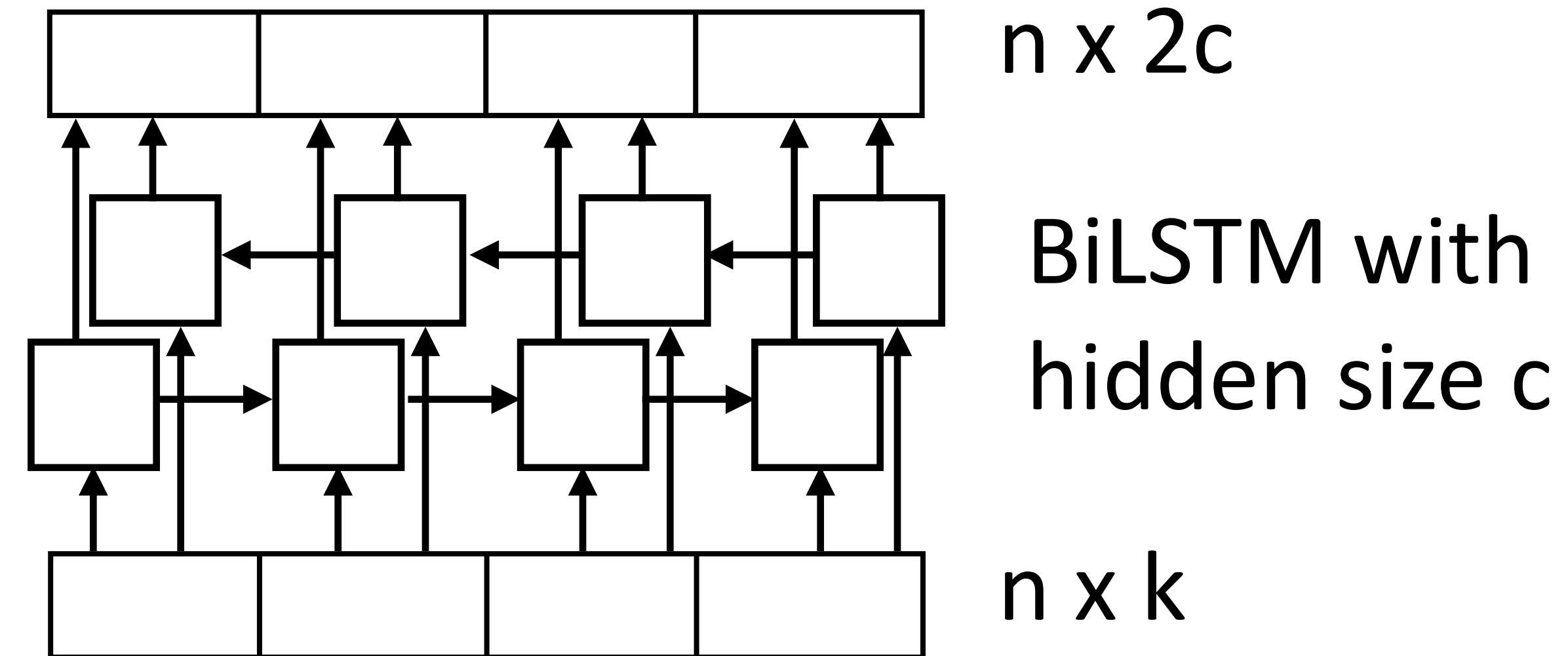
$n \times k$

the movie was good

# Recall: CNNs vs. LSTMs



the movie was good



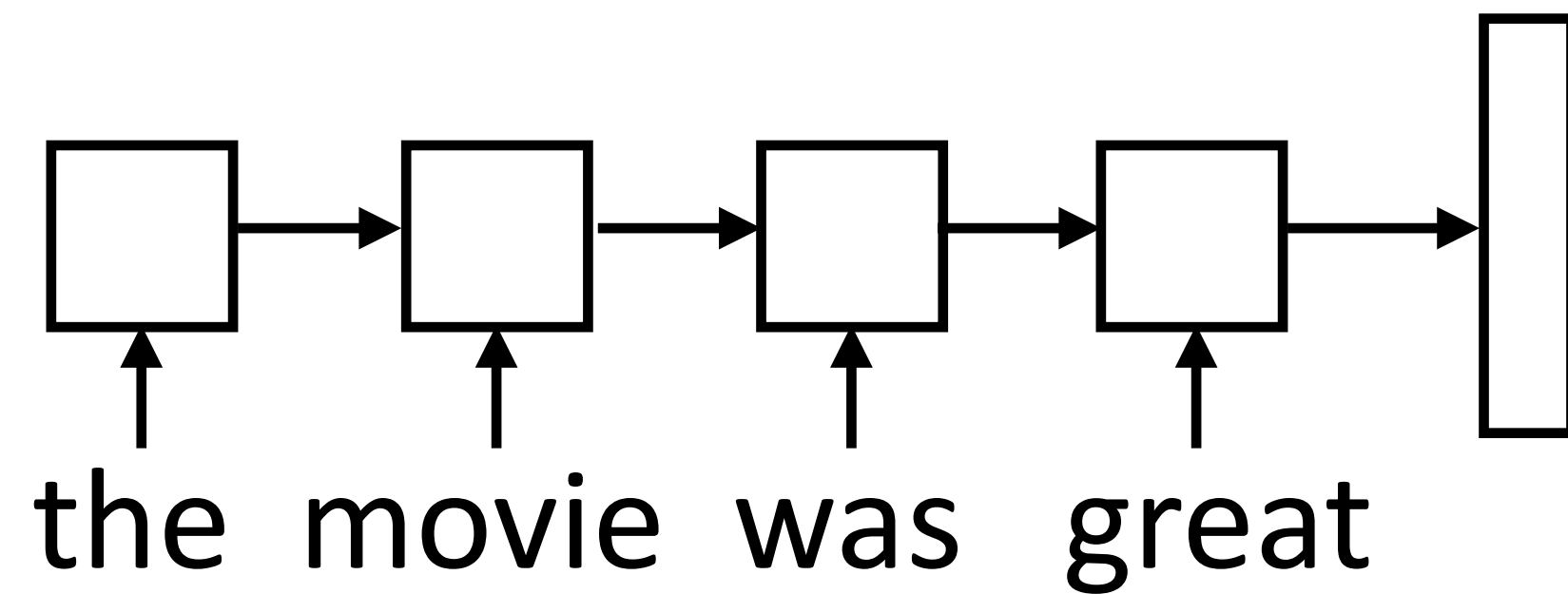
the movie was good

- ▶ Both LSTMs and convolutional layers transform the input using context
- ▶ LSTM: “globally” looks at the entire sentence (but local for many problems)
- ▶ CNN: local depending on filter width + number of layers

# Encoder-Decoder

---

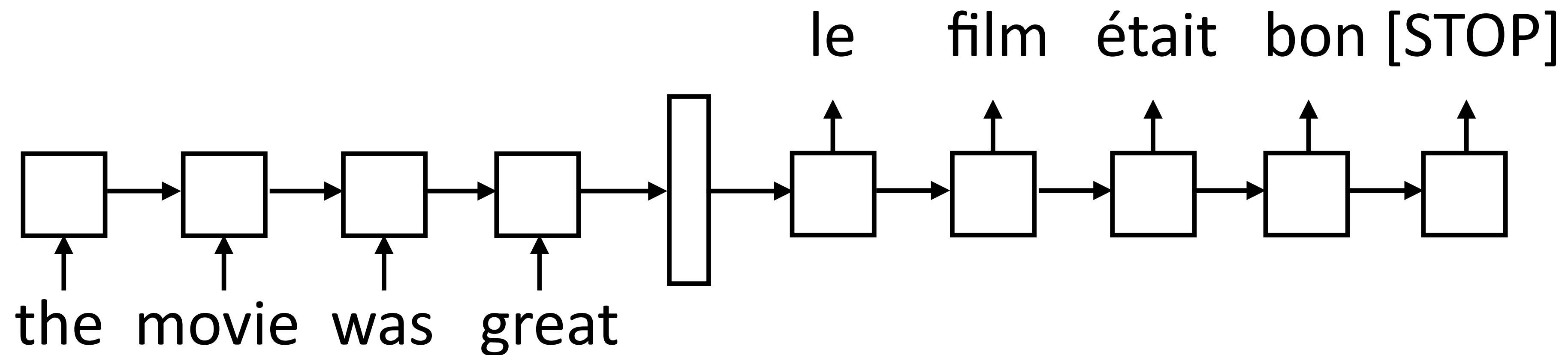
- ▶ Encode a sequence into a fixed-sized vector





# Encoder-Decoder

- ▶ Encode a sequence into a fixed-sized vector



- ▶ Now use that vector to produce a series of tokens as output from a separate LSTM *decoder*

# Encoder-Decoder

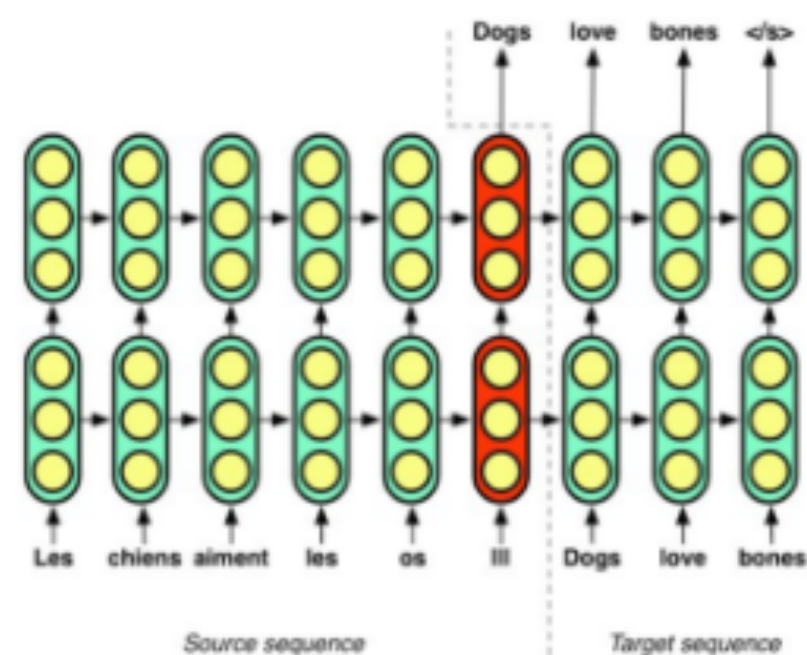


Edward Grefenstette  
@egrefen

Follow

It's not an ACL tutorial on vector representations of meaning if there's at least one Ray Mooney quote.

## A Transduction Bottleneck



Single vector representations of sentences cause a transduction bottleneck.

- Training focusses on learning marginal language model of target language first.
- Longer input sequences cause compressive loss.
- Encoder gets significantly diminished gradient.

In the words of Ray Mooney...

"You can't cram the meaning of a whole %&!\$ing sentence into a single \$&!\*ing vector!"

Yes, the censored-out swearing is copied verbatim.

In the words of Ray Mooney...

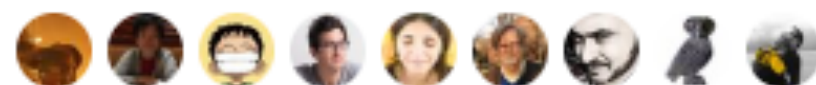
"You can't cram the meaning of a whole %&!\$ing sentence into a single \$&!\*ing vector!"

Yes, the censored-out swearing is copied verbatim.

- Is this true? Sort of...we'll come back to this later

12:27 AM - 11 Jul 2017

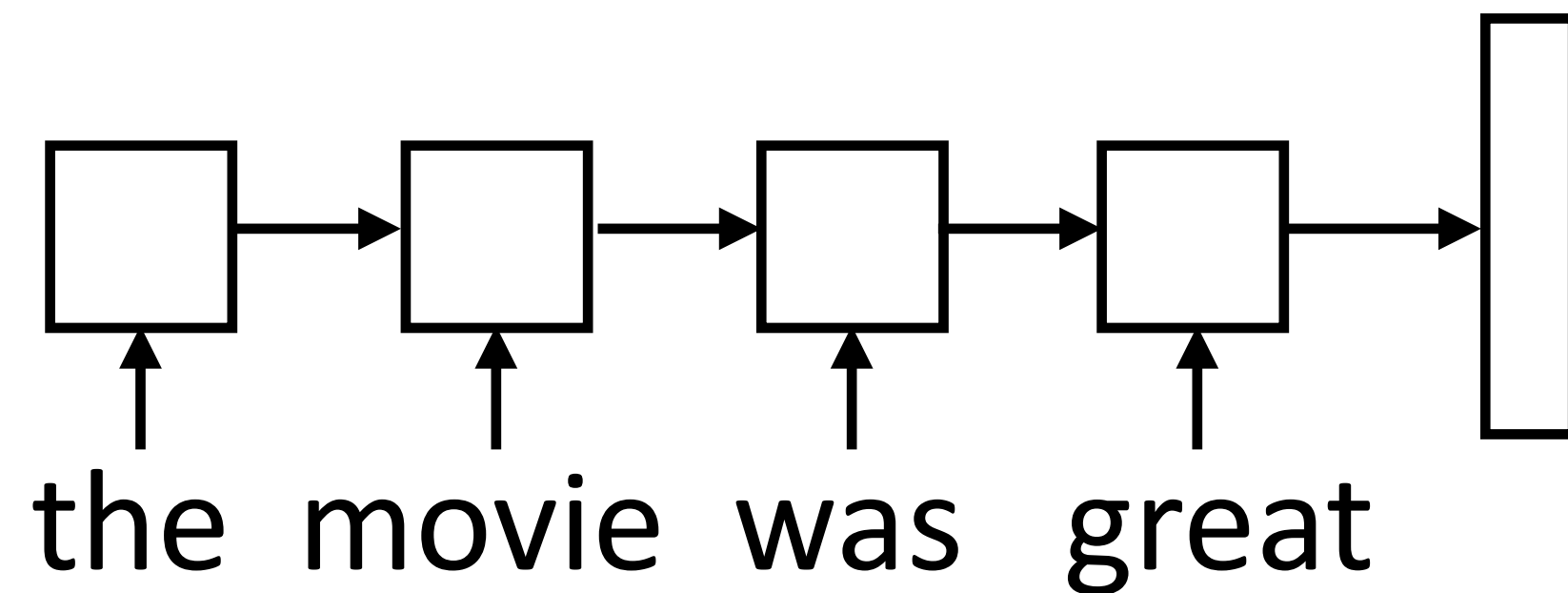
20 Retweets 127 Likes



# Model

---

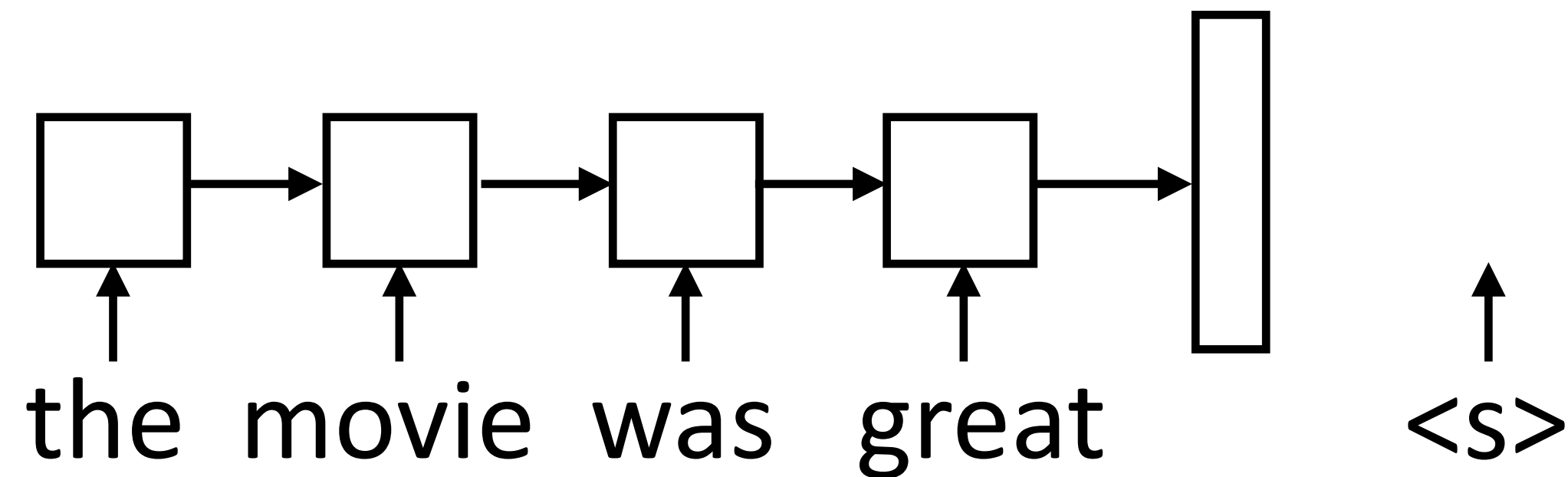
- ▶ Generate next word conditioned on previous word as well as hidden state



# Model

---

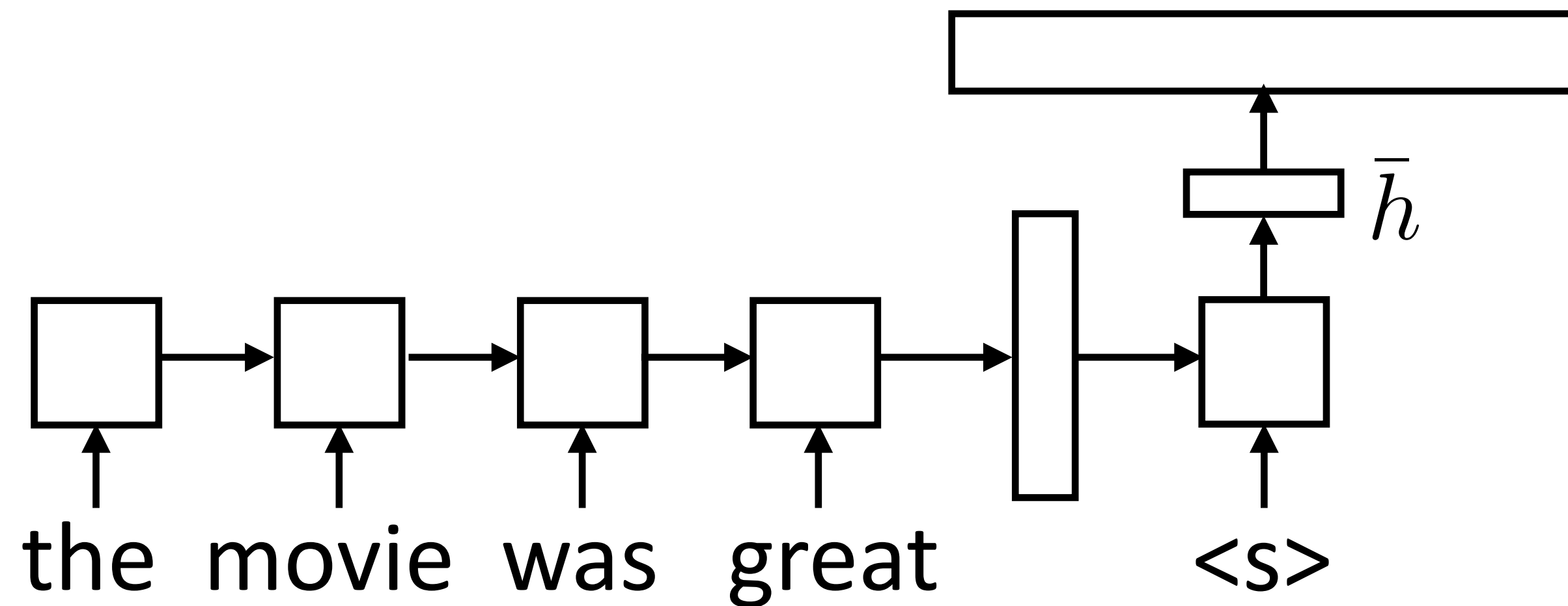
- ▶ Generate next word conditioned on previous word as well as hidden state



# Model

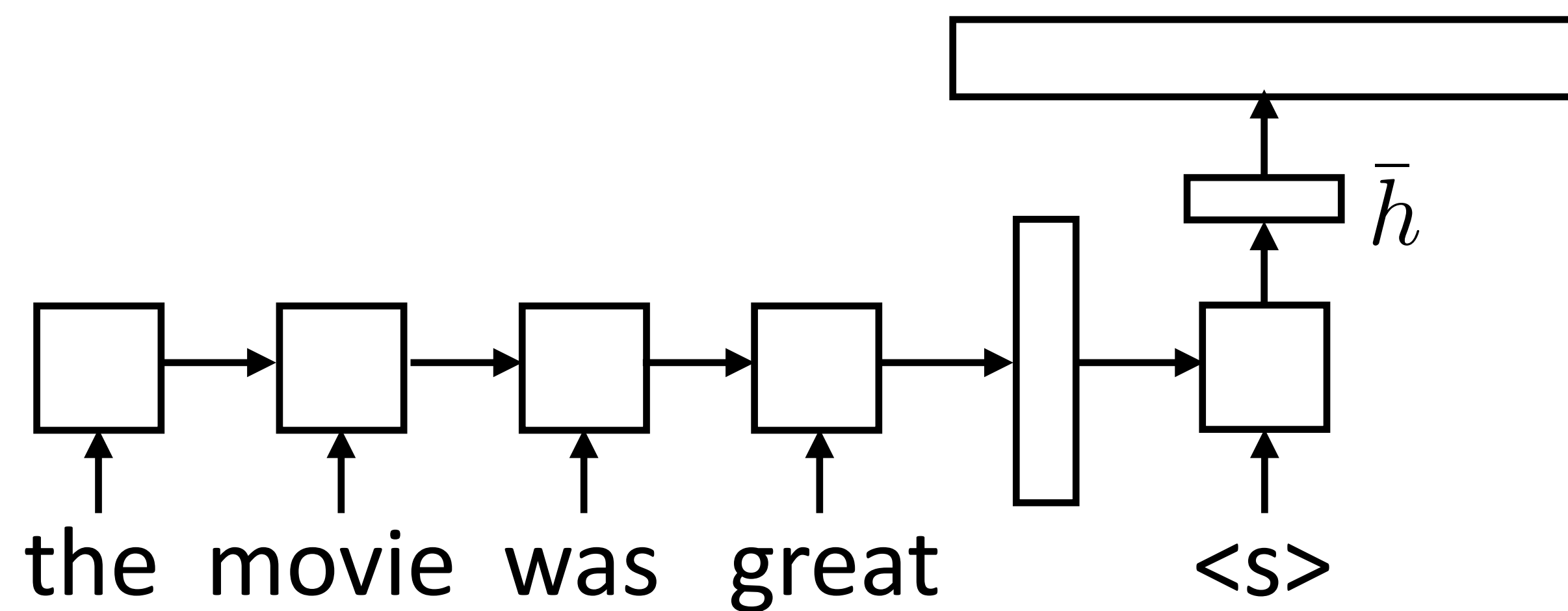
- ▶ Generate next word conditioned on previous word as well as hidden state
- ▶  $W$  size is  $|\text{vocab}| \times |\text{hidden state}|$ , softmax over entire vocabulary

$$P(y_i | \mathbf{x}, y_1, \dots, y_{i-1}) = \text{softmax}(W \bar{h})$$



# Model

- ▶ Generate next word conditioned on previous word as well as hidden state
- ▶  $W$  size is  $|\text{vocab}| \times |\text{hidden state}|$ , softmax over entire vocabulary

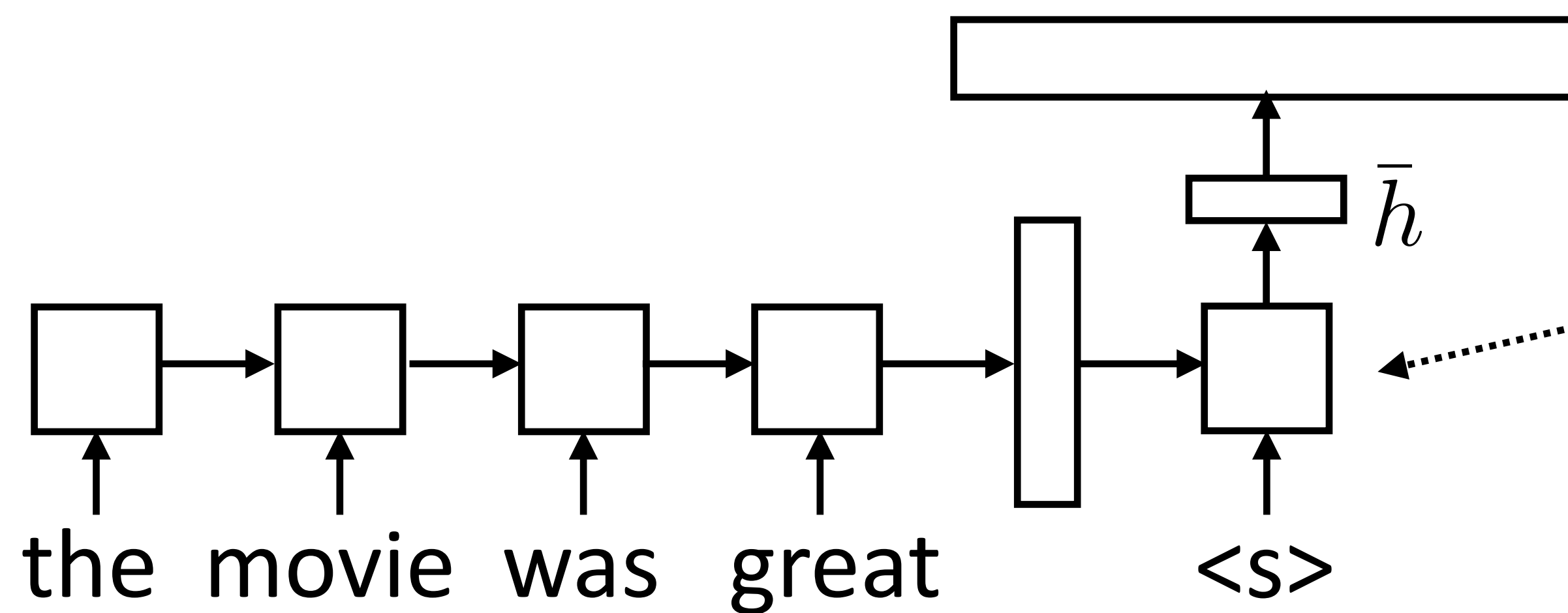


$$P(y_i | \mathbf{x}, y_1, \dots, y_{i-1}) = \text{softmax}(W \bar{h})$$

$$P(\mathbf{y} | \mathbf{x}) = \prod_{i=1}^n P(y_i | \mathbf{x}, y_1, \dots, y_{i-1})$$

# Model

- ▶ Generate next word conditioned on previous word as well as hidden state
- ▶  $W$  size is  $|\text{vocab}| \times |\text{hidden state}|$ , softmax over entire vocabulary



$$P(y_i | \mathbf{x}, y_1, \dots, y_{i-1}) = \text{softmax}(W \bar{h})$$

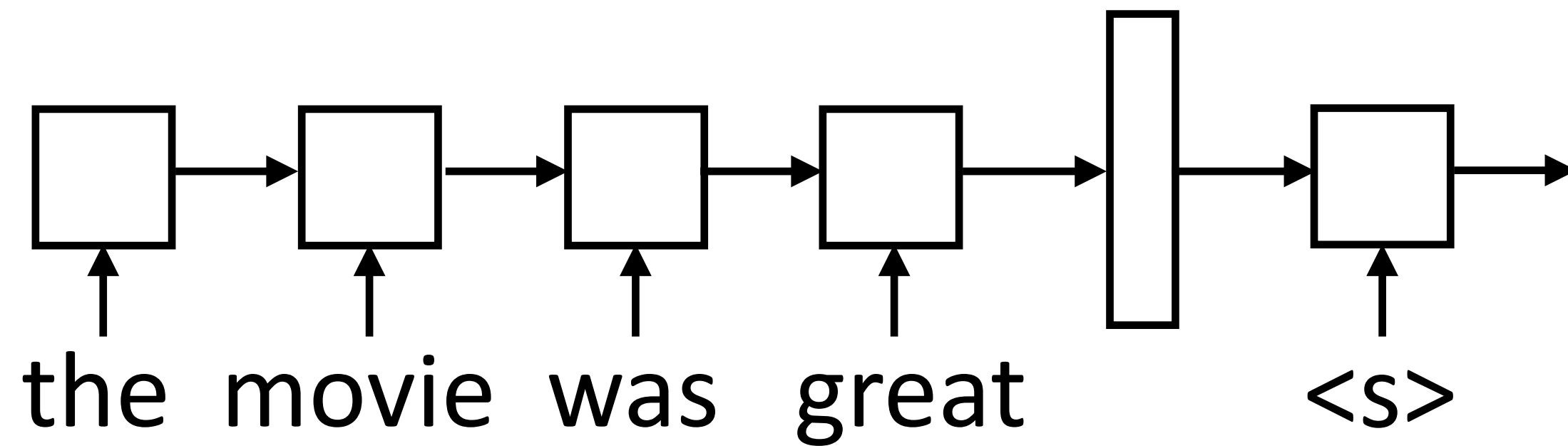
$$P(\mathbf{y} | \mathbf{x}) = \prod_{i=1}^n P(y_i | \mathbf{x}, y_1, \dots, y_{i-1})$$

Decoder has separate parameters from encoder, so this can learn to be a language model (produce a plausible next word given current one)

# Inference

---

- ▶ Generate next word conditioned on previous word as well as hidden state

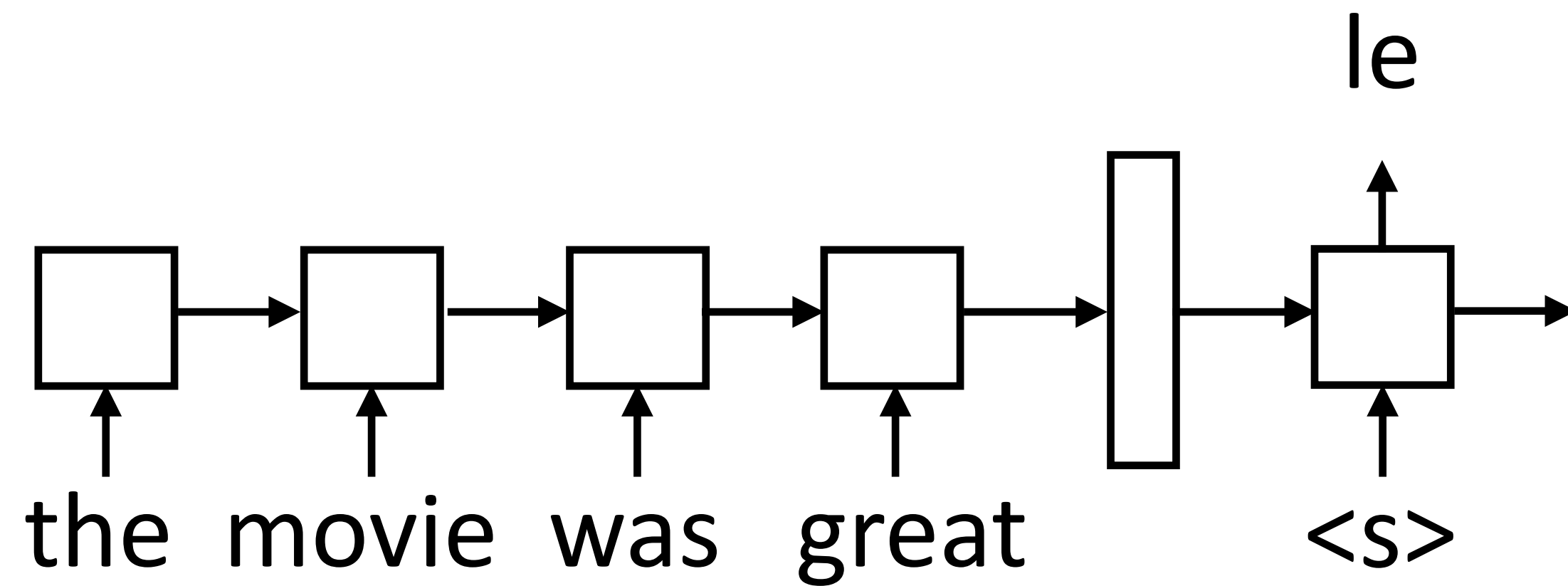




# Inference

---

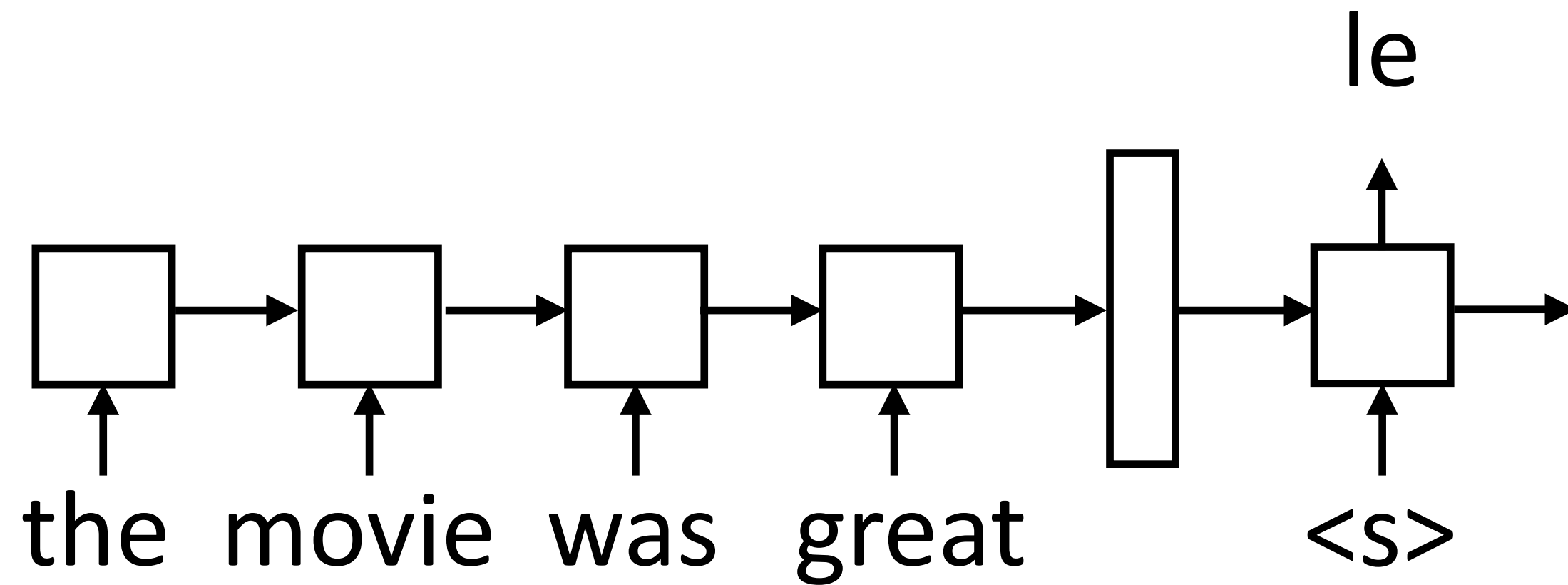
- ▶ Generate next word conditioned on previous word as well as hidden state



# Inference

---

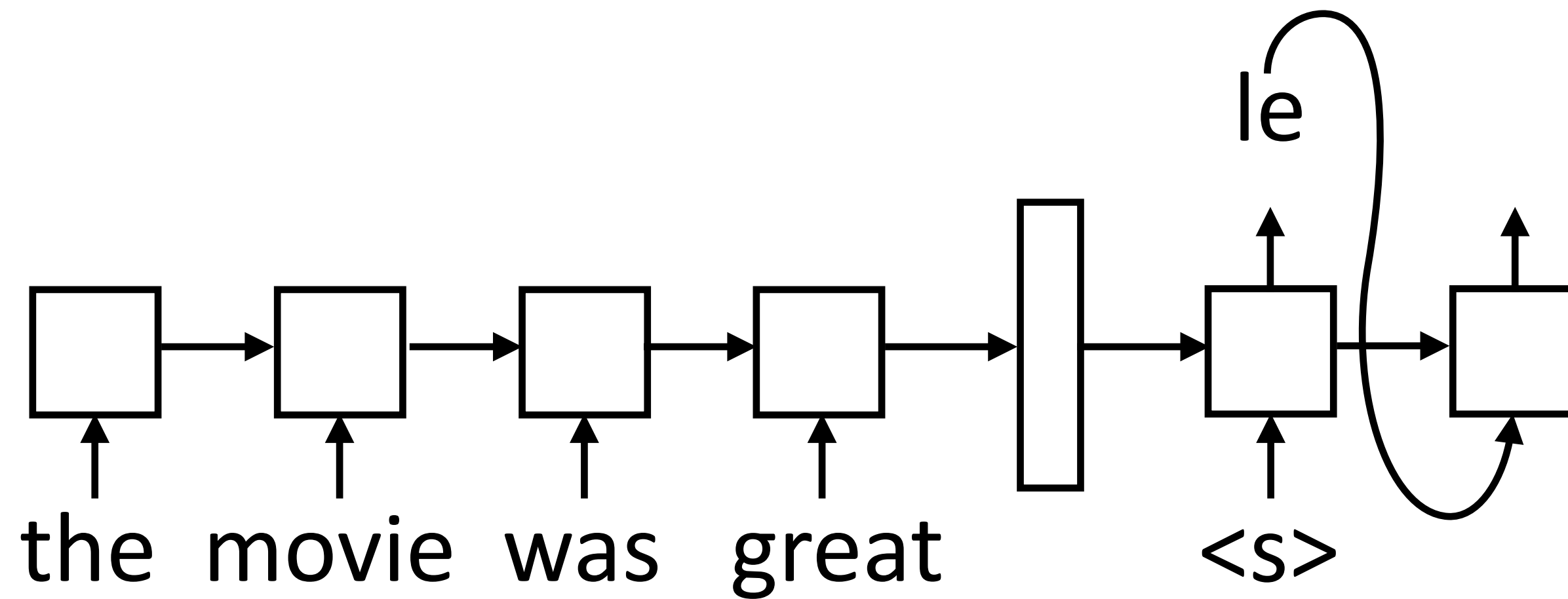
- ▶ Generate next word conditioned on previous word as well as hidden state



- ▶ During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state

# Inference

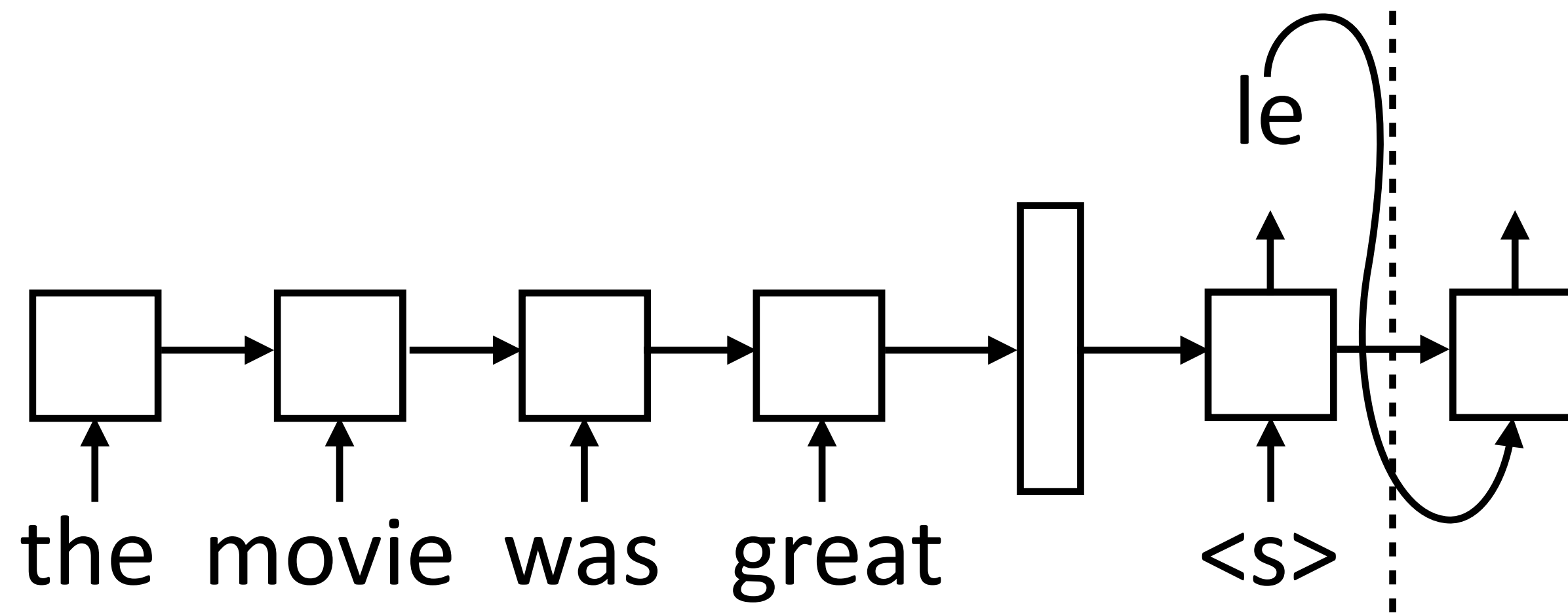
- ▶ Generate next word conditioned on previous word as well as hidden state



- ▶ During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state

# Inference

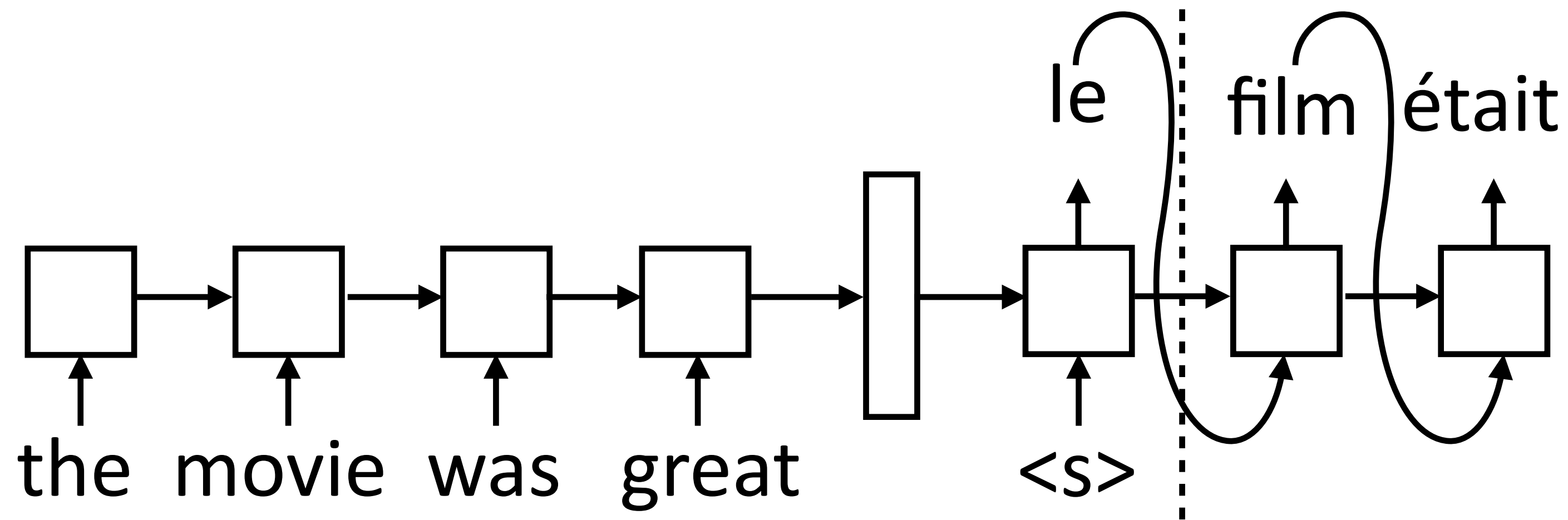
- ▶ Generate next word conditioned on previous word as well as hidden state



- ▶ During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state
- ▶ Need to actually evaluate computation graph up to this point to form input for the next state

# Inference

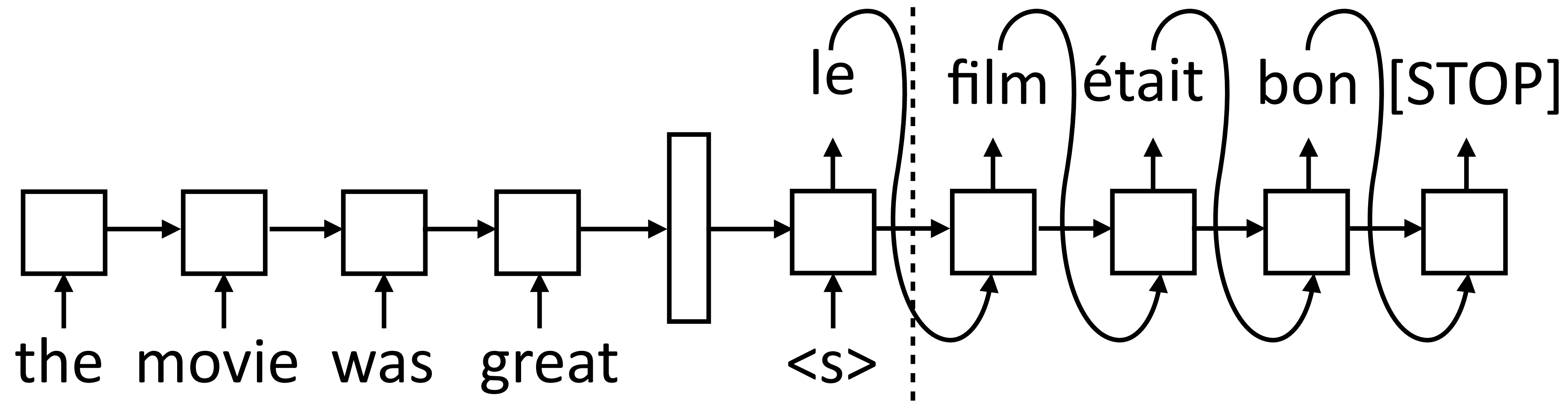
- ▶ Generate next word conditioned on previous word as well as hidden state



- ▶ During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state
- ▶ Need to actually evaluate computation graph up to this point to form input for the next state

# Inference

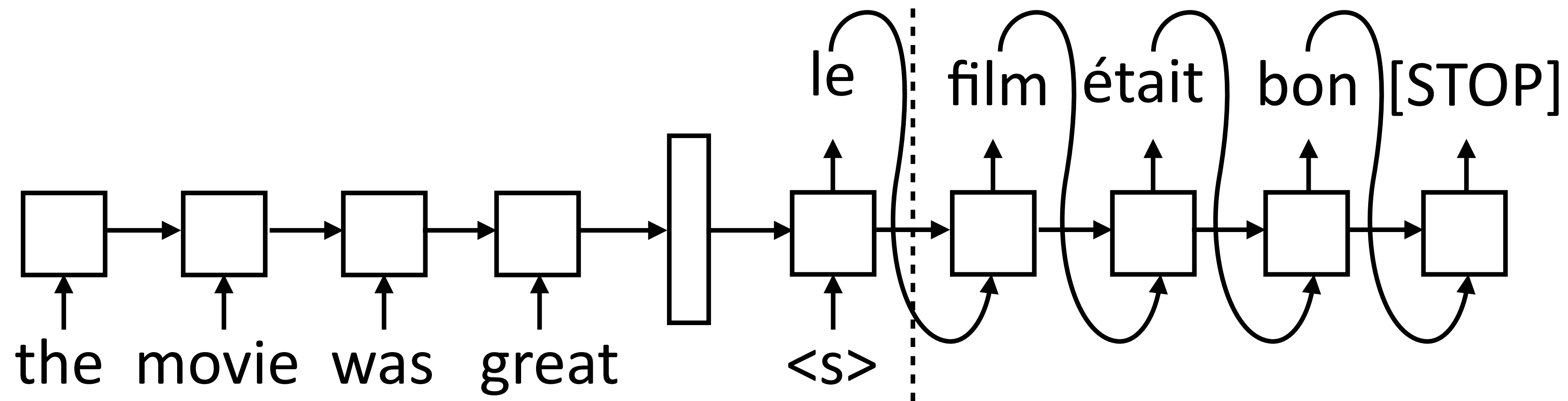
- ▶ Generate next word conditioned on previous word as well as hidden state



- ▶ During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state
- ▶ Need to actually evaluate computation graph up to this point to form input for the next state

# Inference

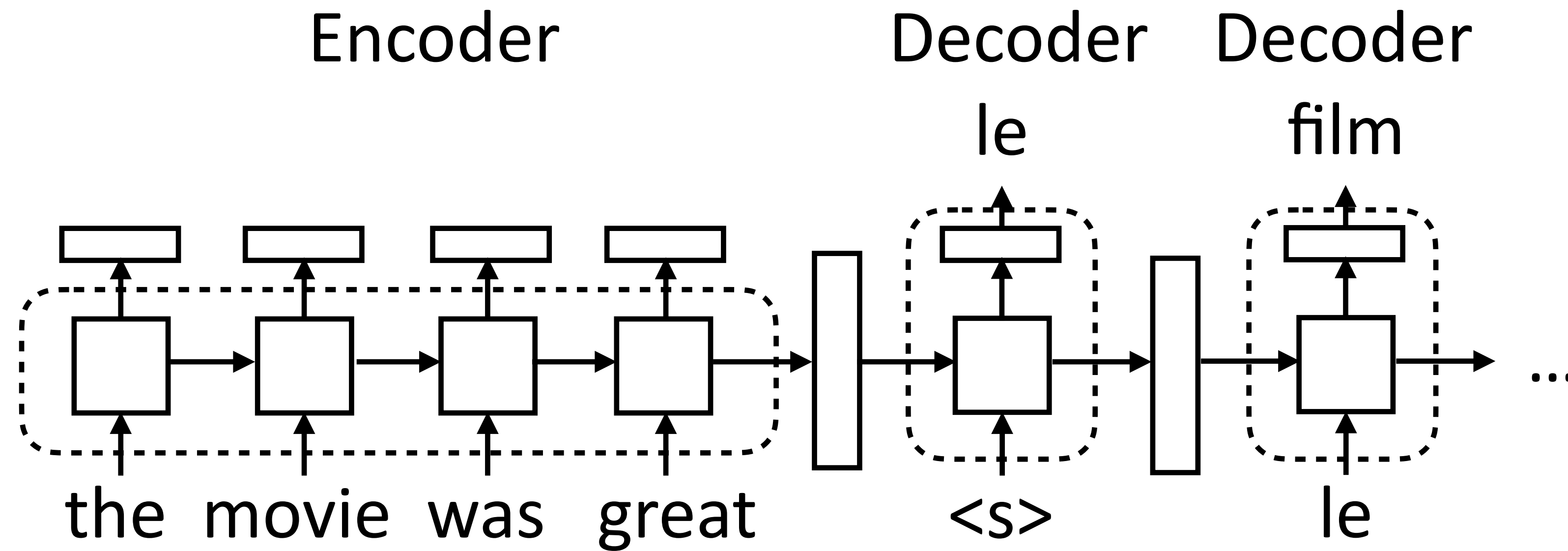
- ▶ Generate next word conditioned on previous word as well as hidden state



- ▶ During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state
- ▶ Need to actually evaluate computation graph up to this point to form input for the next state
- ▶ Decoder is advanced one state at a time until [STOP] is reached

# Implementing seq2seq Models

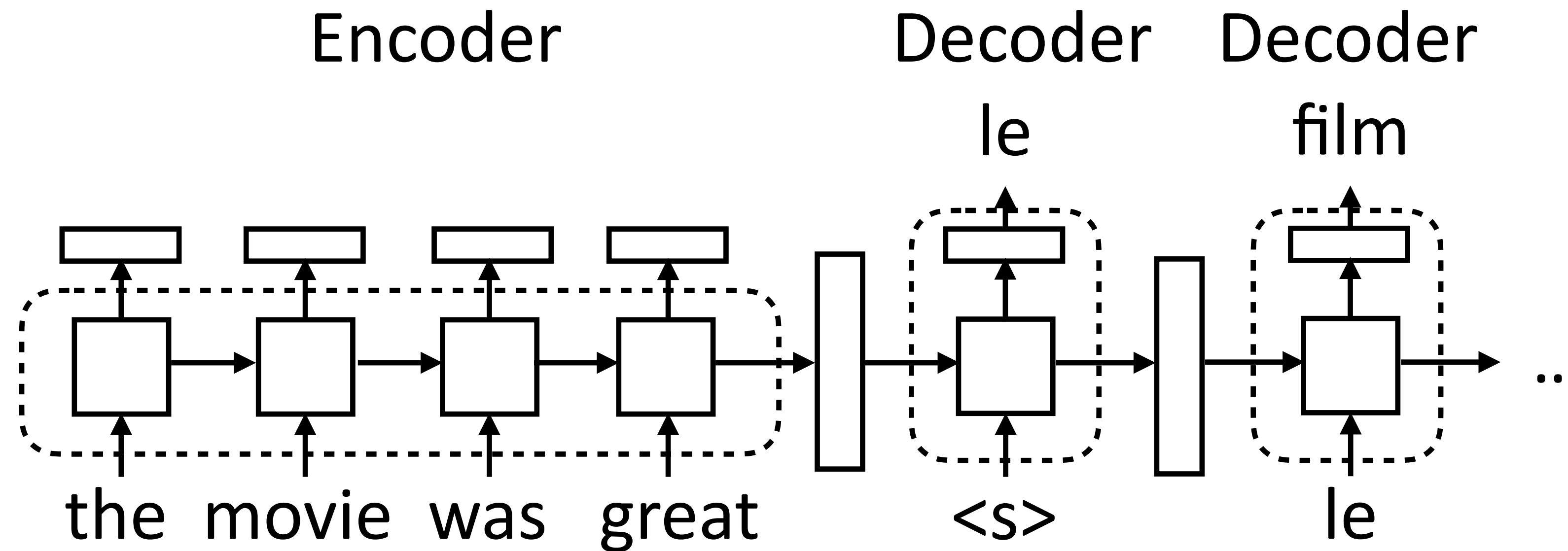
---





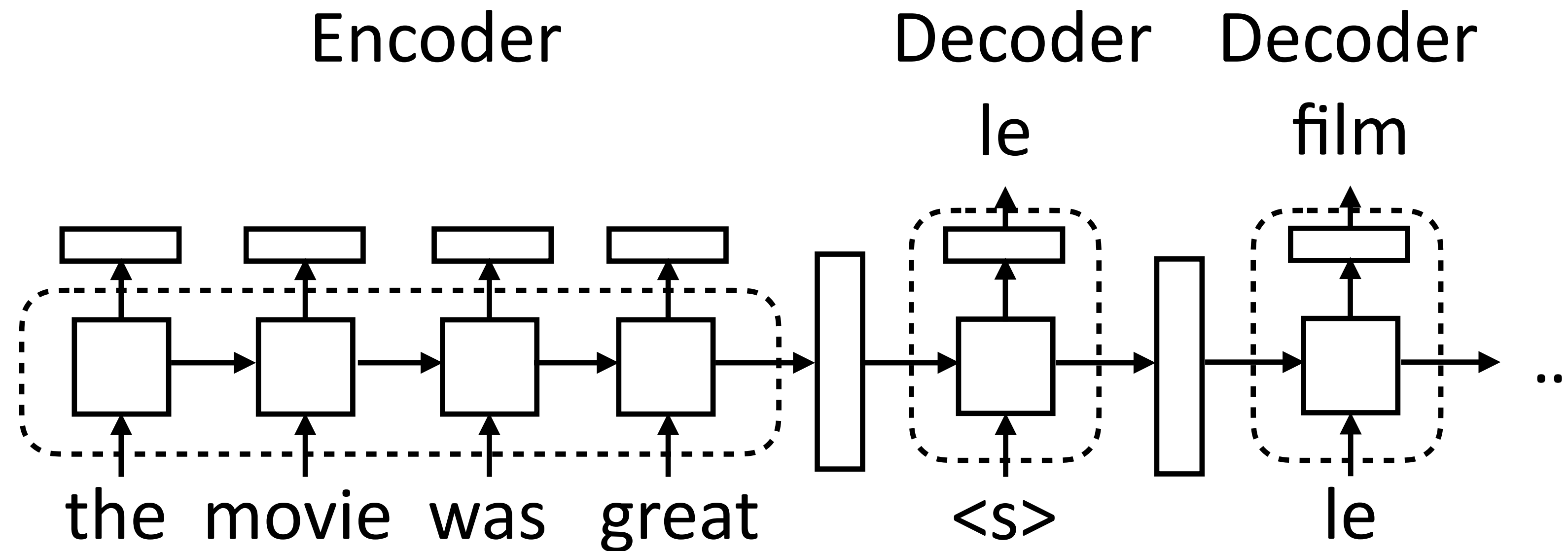
# Implementing seq2seq Models

---



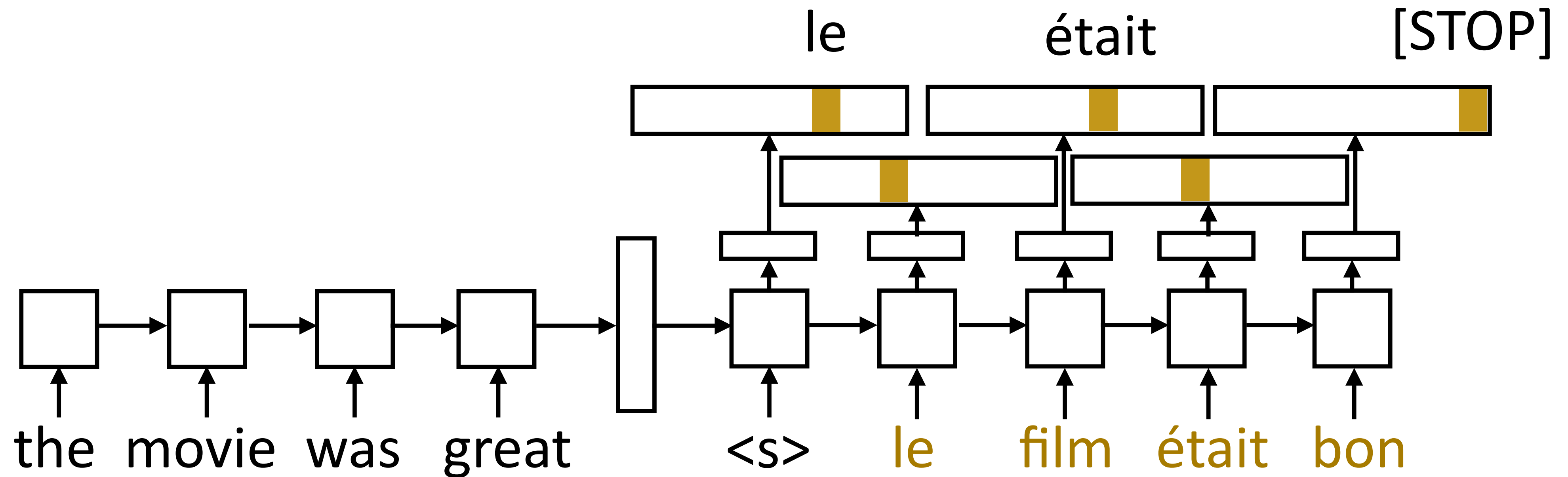
- ▶ Encoder: consumes sequence of tokens, produces a vector. Analogous to encoders for classification/tagging tasks

# Implementing seq2seq Models



- ▶ Encoder: consumes sequence of tokens, produces a vector. Analogous to encoders for classification/tagging tasks
- ▶ Decoder: separate module, single cell. Takes two inputs: hidden state (vector  $h$  or tuple  $(h, c)$ ) and previous token. Outputs token + new state

# Training

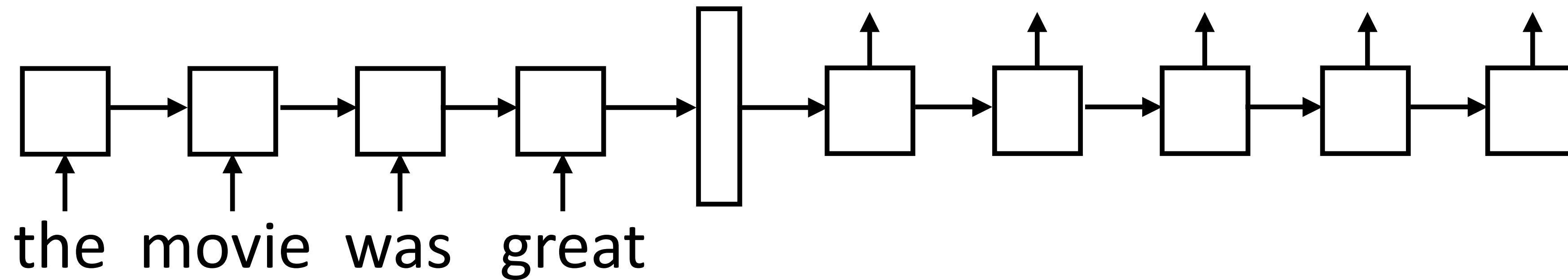


- ▶ Objective: maximize  $\sum_{(\mathbf{x}, \mathbf{y})} \sum_{i=1}^n \log P(y_i^* | \mathbf{x}, y_1^*, \dots, y_{i-1}^*)$
- ▶ One loss term for each target-sentence word, feed the correct word regardless of model's prediction

# Training: Scheduled Sampling

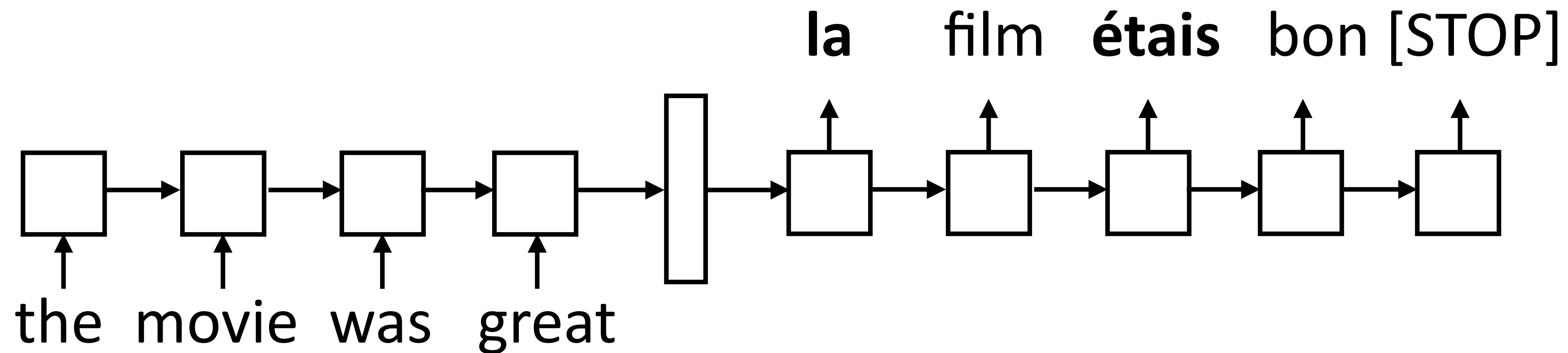
---

- ▶ Model needs to do the right thing even with its own predictions



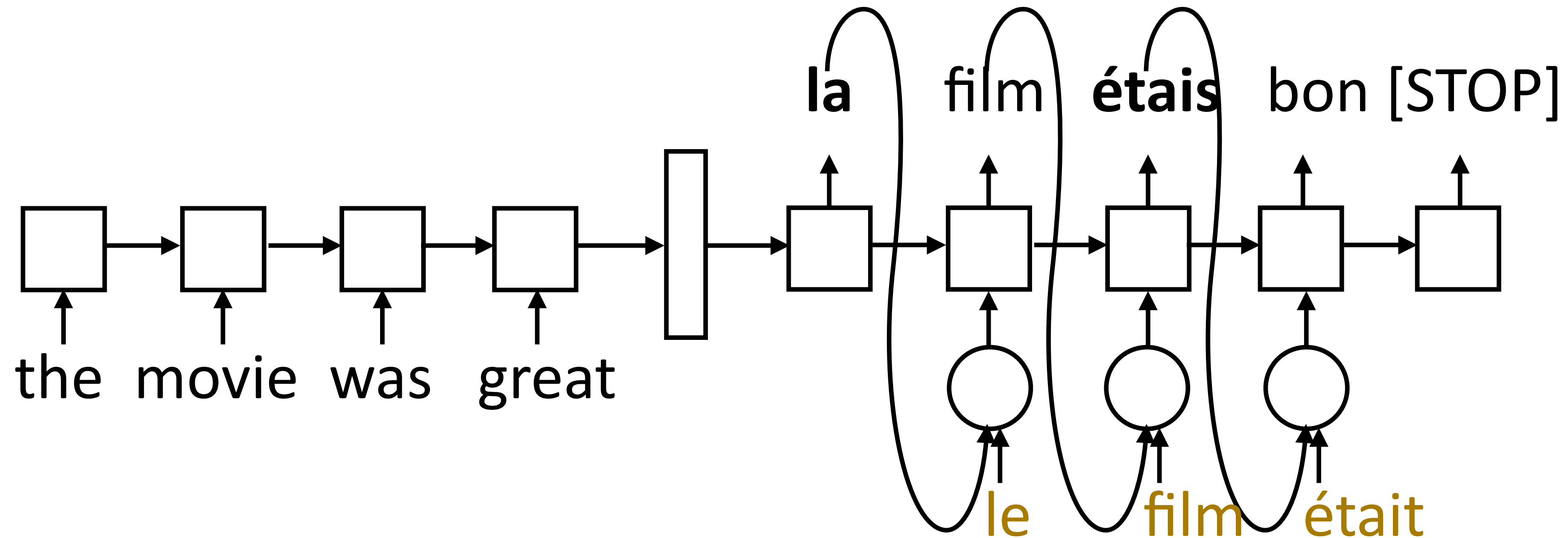
# Training: Scheduled Sampling

- ▶ Model needs to do the right thing even with its own predictions



# Training: Scheduled Sampling

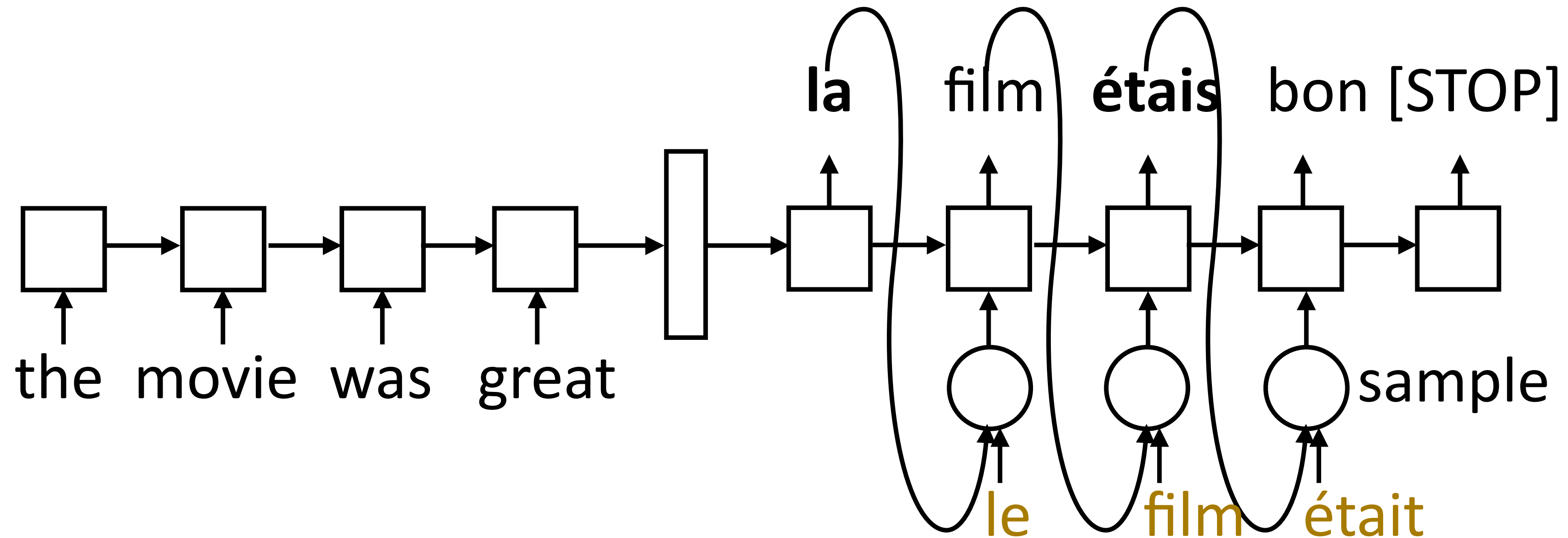
- ▶ Model needs to do the right thing even with its own predictions



- ▶ Scheduled sampling: with probability  $p$ , take the gold as input, else take the model's prediction

# Training: Scheduled Sampling

- ▶ Model needs to do the right thing even with its own predictions



- ▶ Scheduled sampling: with probability  $p$ , take the gold as input, else take the model's prediction
- ▶ Starting with  $p = 1$  and decaying it works best

# Implementation Details

---



# Implementation Details

---

- ▶ Sentence lengths vary for both encoder and decoder:

# Implementation Details

---

- ▶ Sentence lengths vary for both encoder and decoder:
  - ▶ Typically pad everything to the right length

# Implementation Details

---

- ▶ Sentence lengths vary for both encoder and decoder:
  - ▶ Typically pad everything to the right length
- ▶ Encoder: Can be a CNN/LSTM/...

# Implementation Details

---

- ▶ Sentence lengths vary for both encoder and decoder:
  - ▶ Typically pad everything to the right length
- ▶ Encoder: Can be a CNN/LSTM/...
- ▶ Decoder: also flexible in terms of architecture (more later). Execute one step of computation at a time, so computation graph is formulated as taking one input + hidden state

# Implementation Details

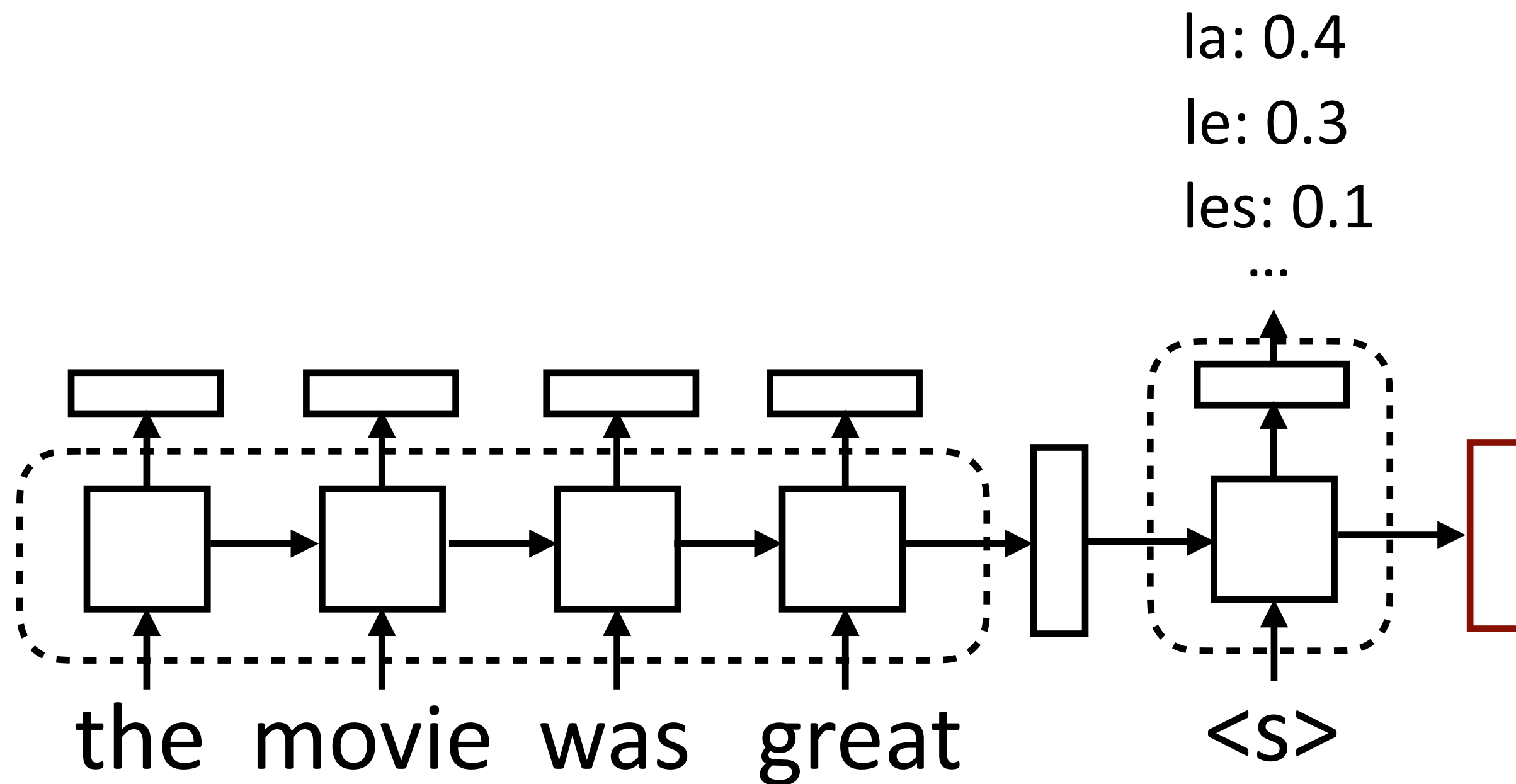
---

- ▶ Sentence lengths vary for both encoder and decoder:
  - ▶ Typically pad everything to the right length
- ▶ Encoder: Can be a CNN/LSTM/...
- ▶ Decoder: also flexible in terms of architecture (more later). Execute one step of computation at a time, so computation graph is formulated as taking one input + hidden state
- ▶ Beam search: can help with lookahead. Finds the (approximate) highest scoring sequence:

$$\operatorname{argmax}_{\mathbf{y}} \prod_{i=1}^n P(y_i | \mathbf{x}, y_1, \dots, y_{i-1})$$

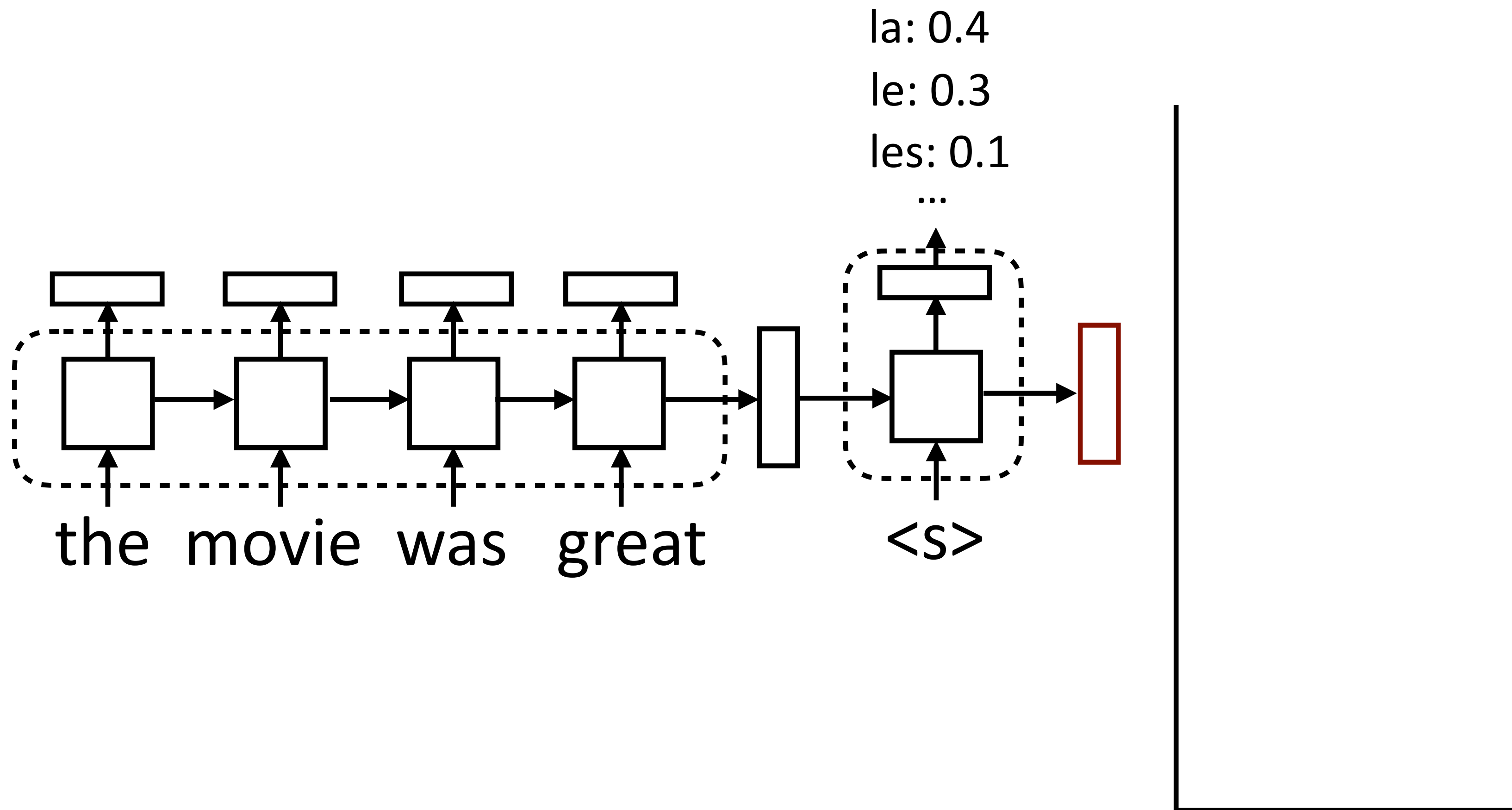
# Beam Search

- ▶ Maintain decoder state, token history in beam



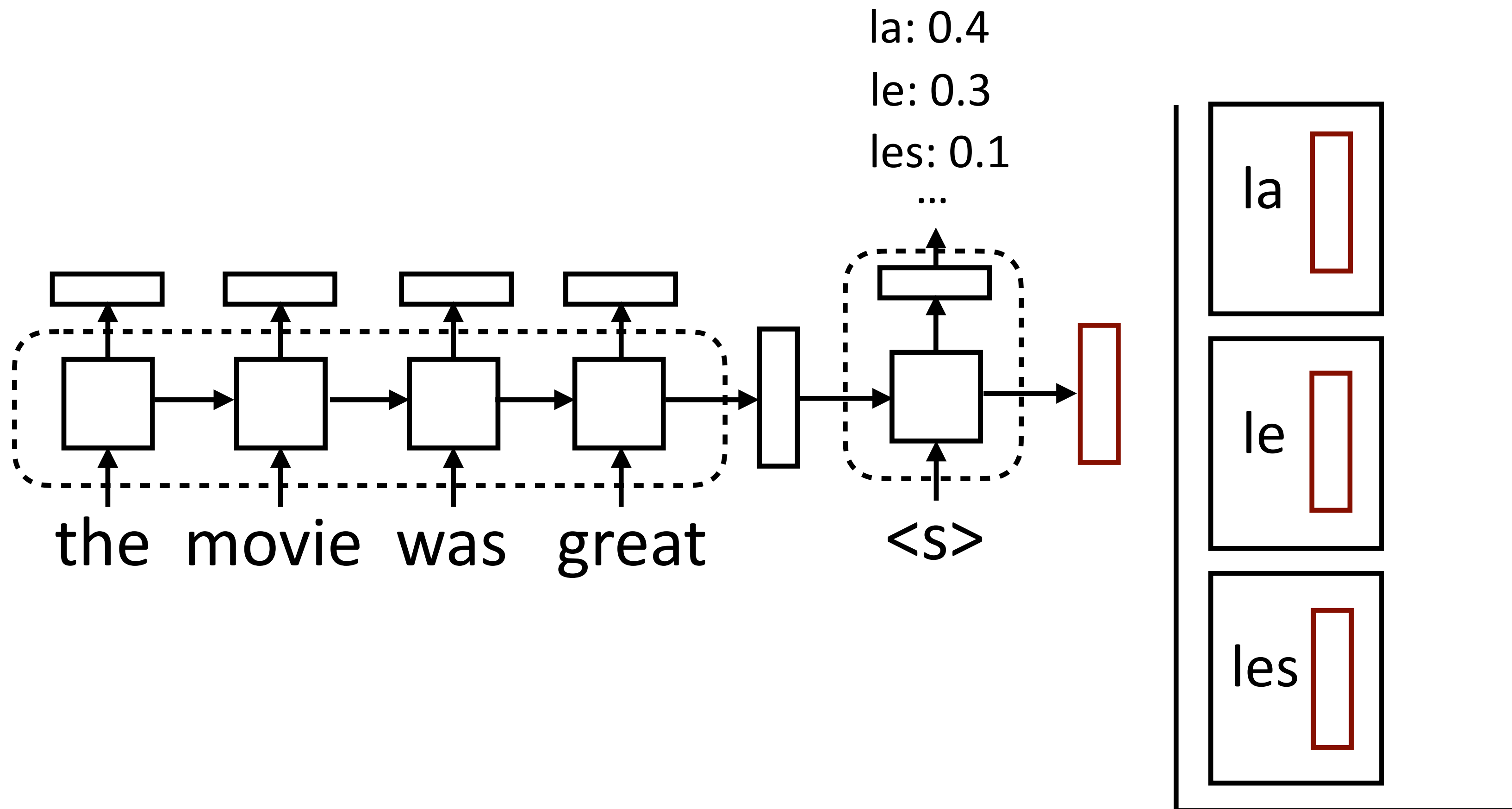
# Beam Search

- ▶ Maintain decoder state, token history in beam



# Beam Search

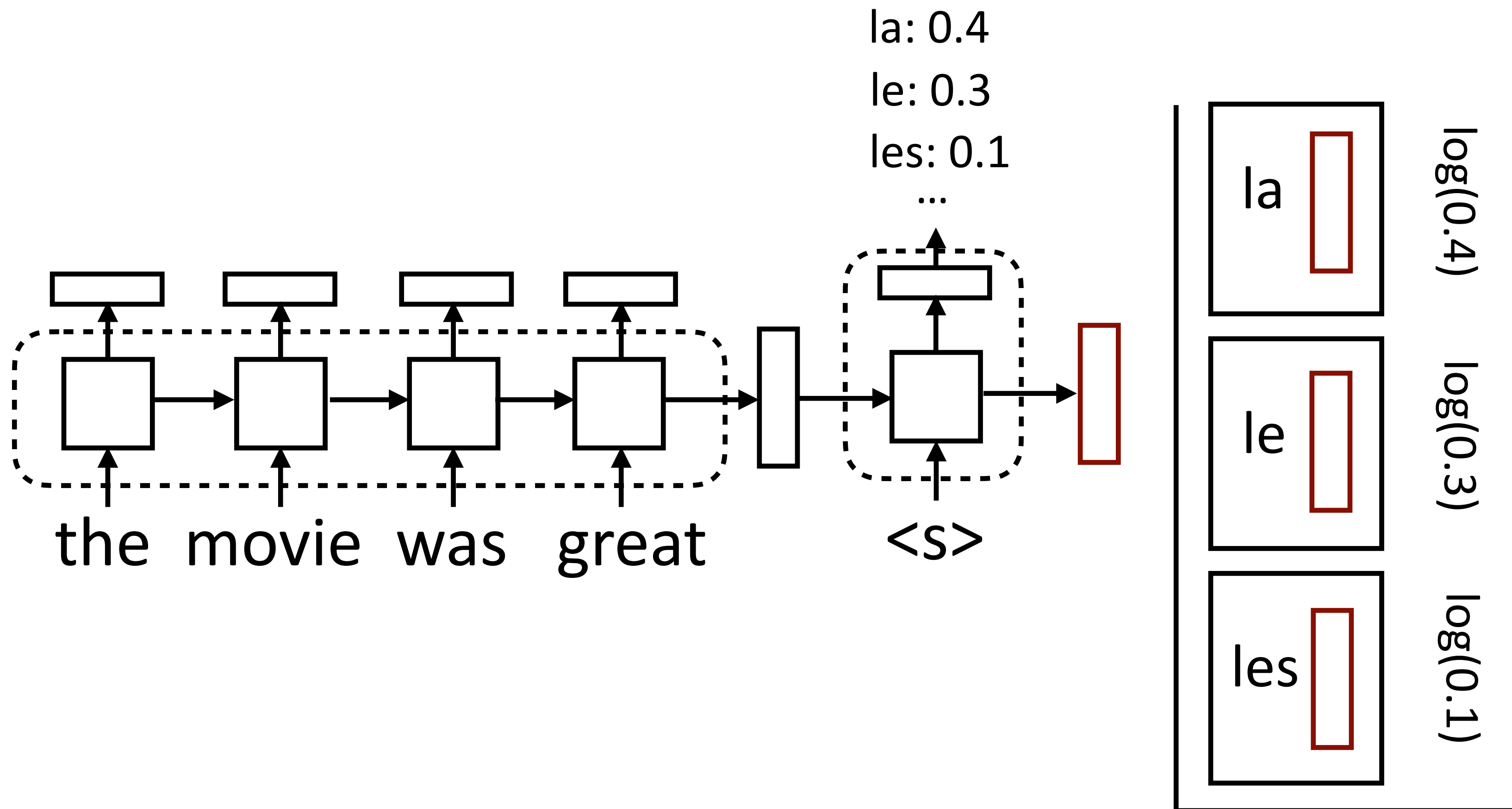
- ▶ Maintain decoder state, token history in beam





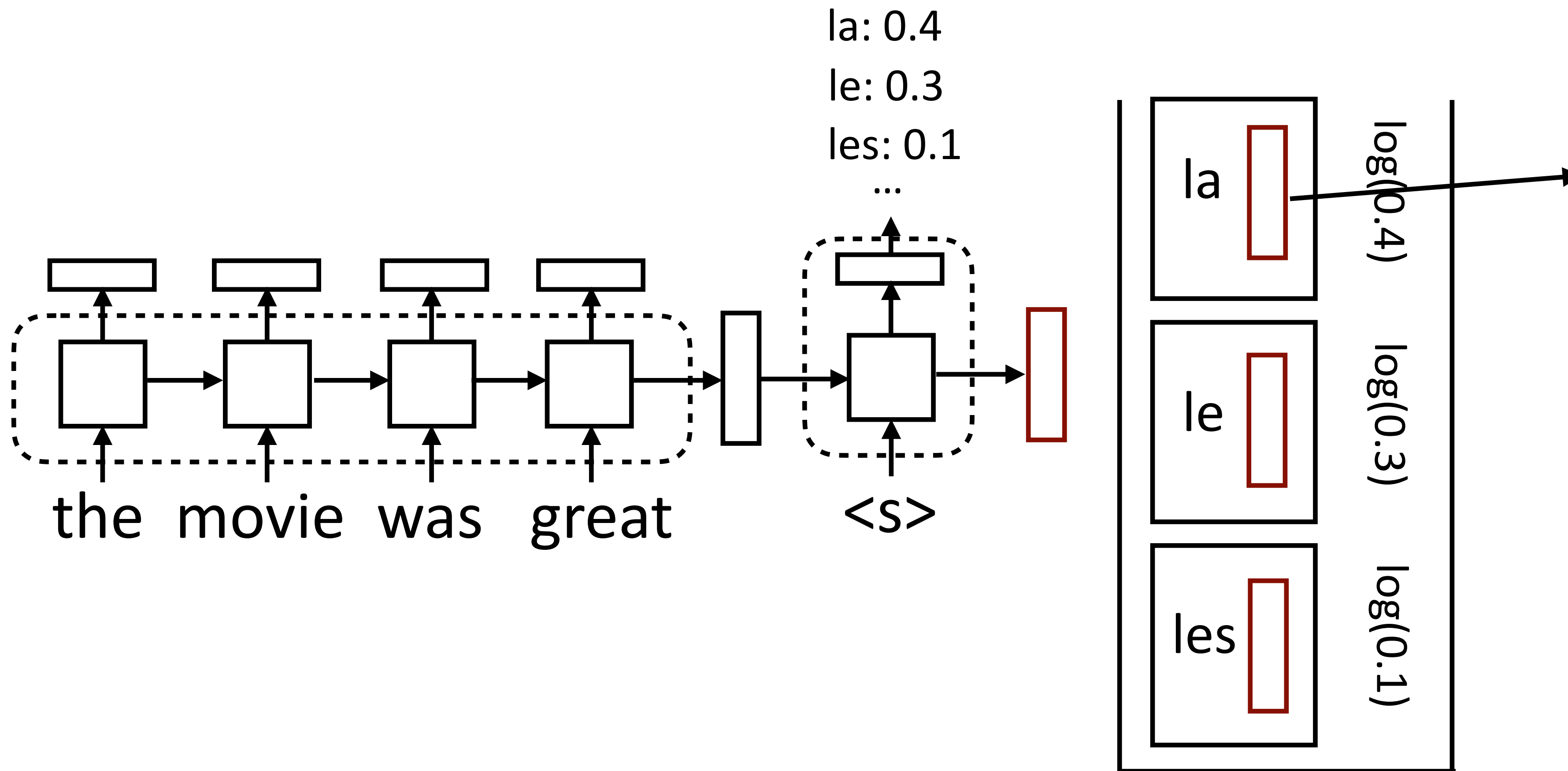
# Beam Search

- ▶ Maintain decoder state, token history in beam



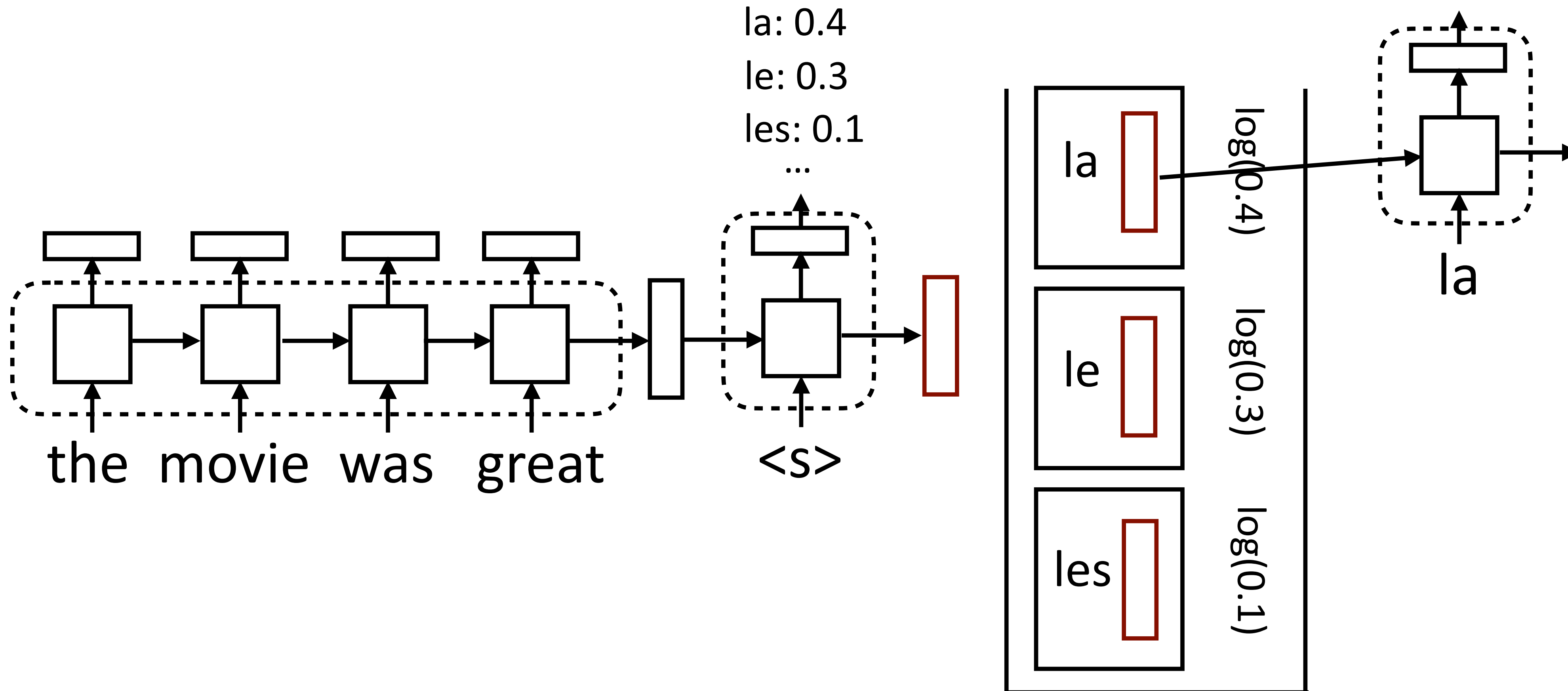
# Beam Search

- ▶ Maintain decoder state, token history in beam



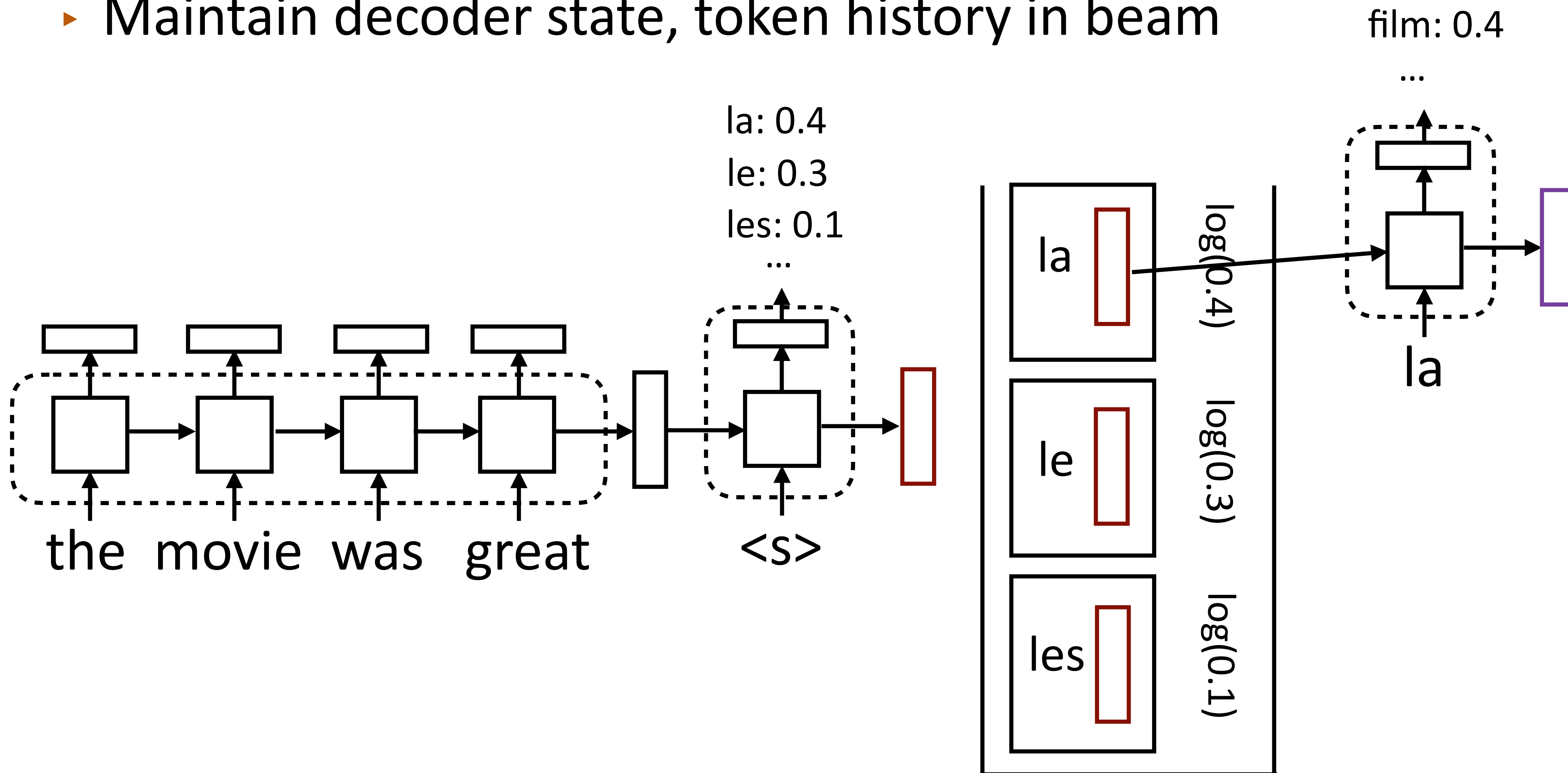
# Beam Search

- ▶ Maintain decoder state, token history in beam



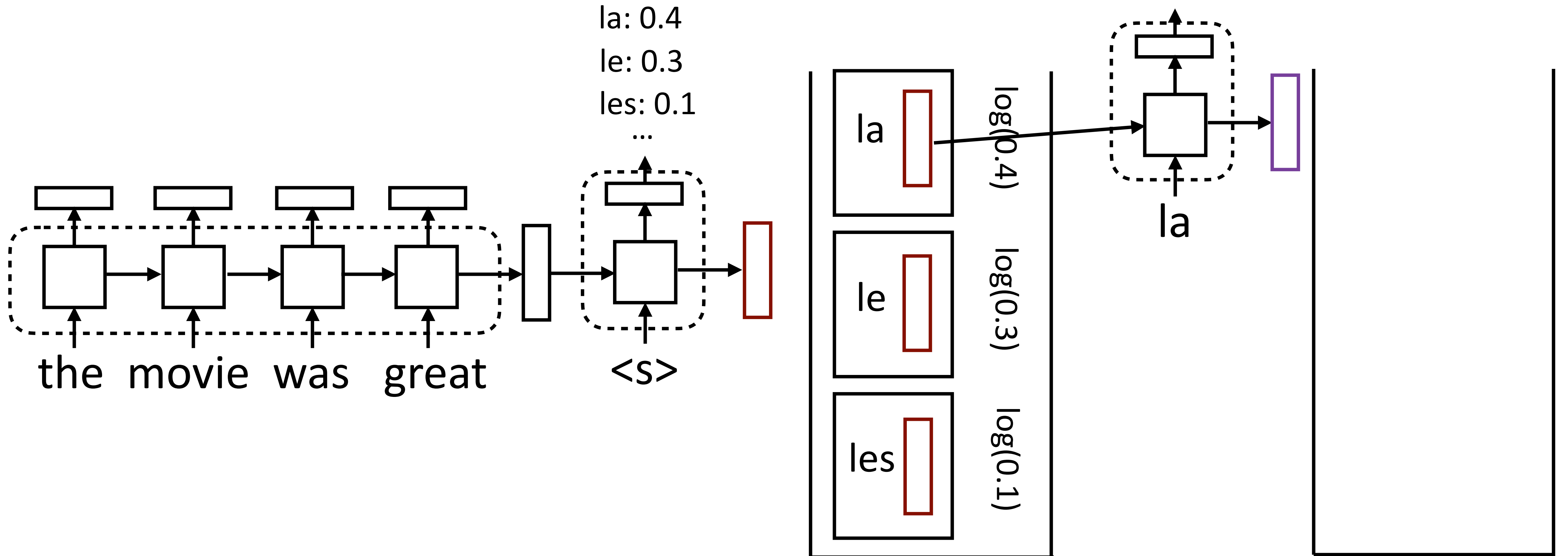
# Beam Search

- Maintain decoder state, token history in beam



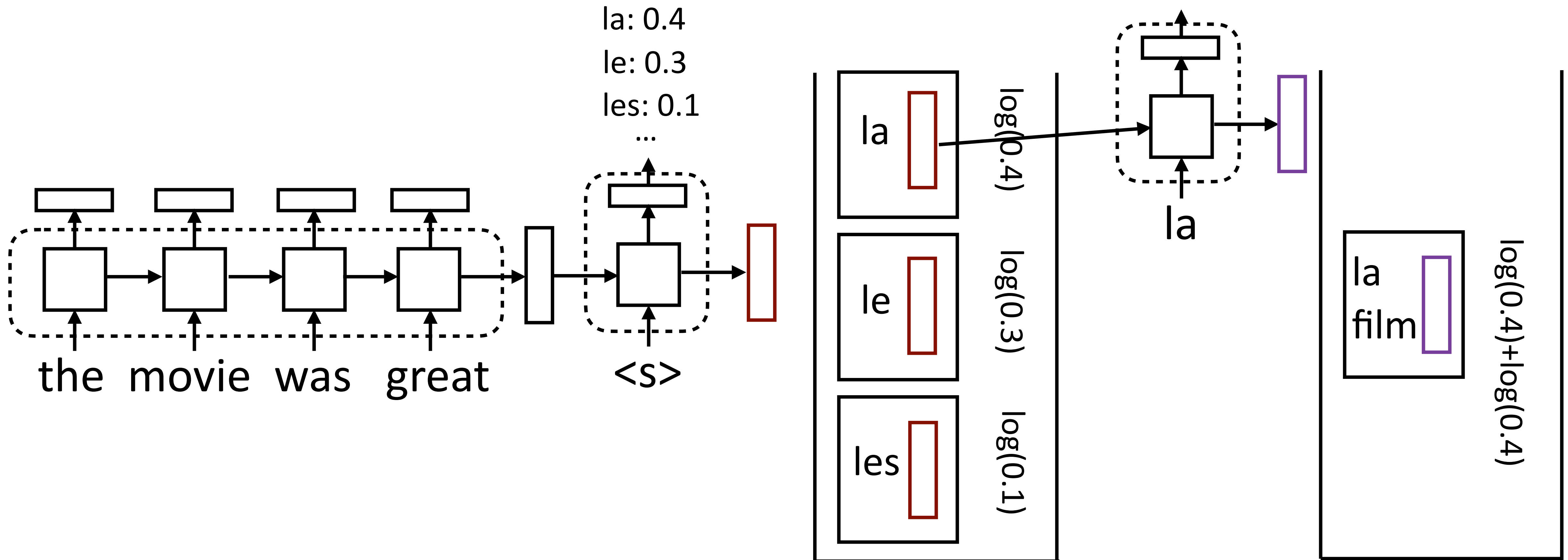
# Beam Search

- Maintain decoder state, token history in beam



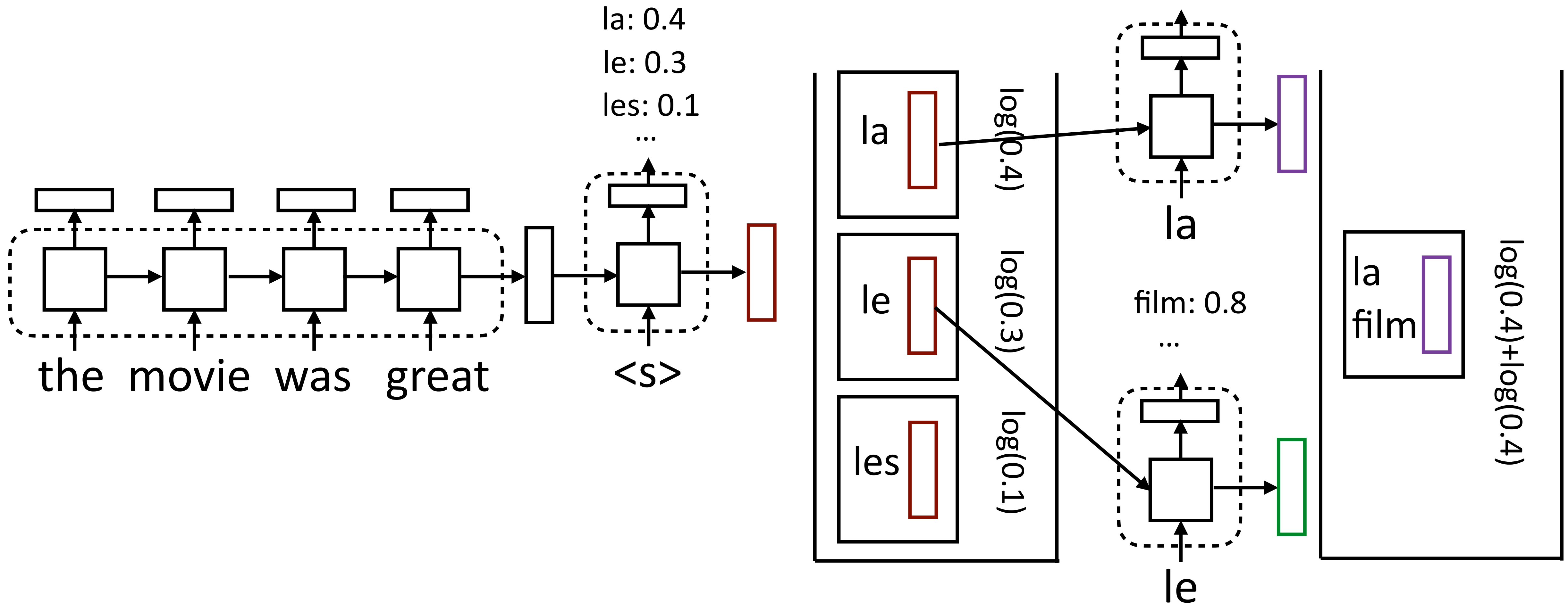
# Beam Search

- Maintain decoder state, token history in beam



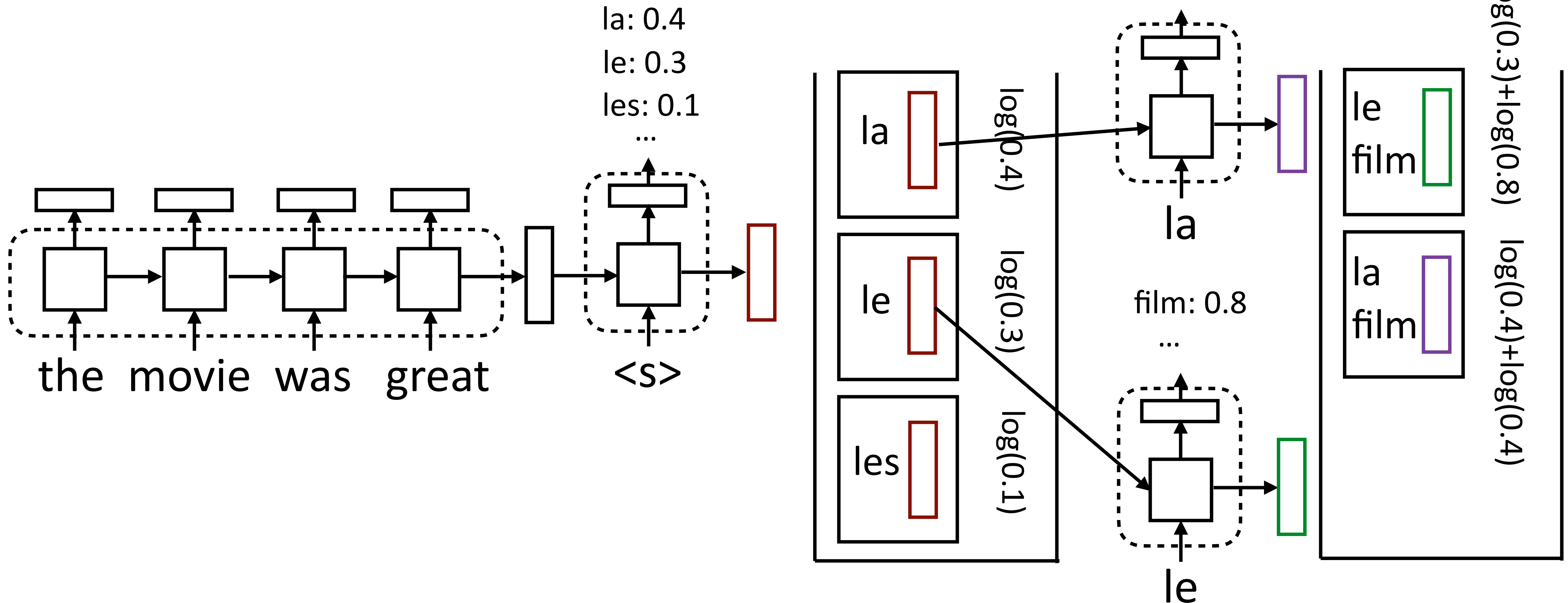
# Beam Search

- Maintain decoder state, token history in beam



# Beam Search

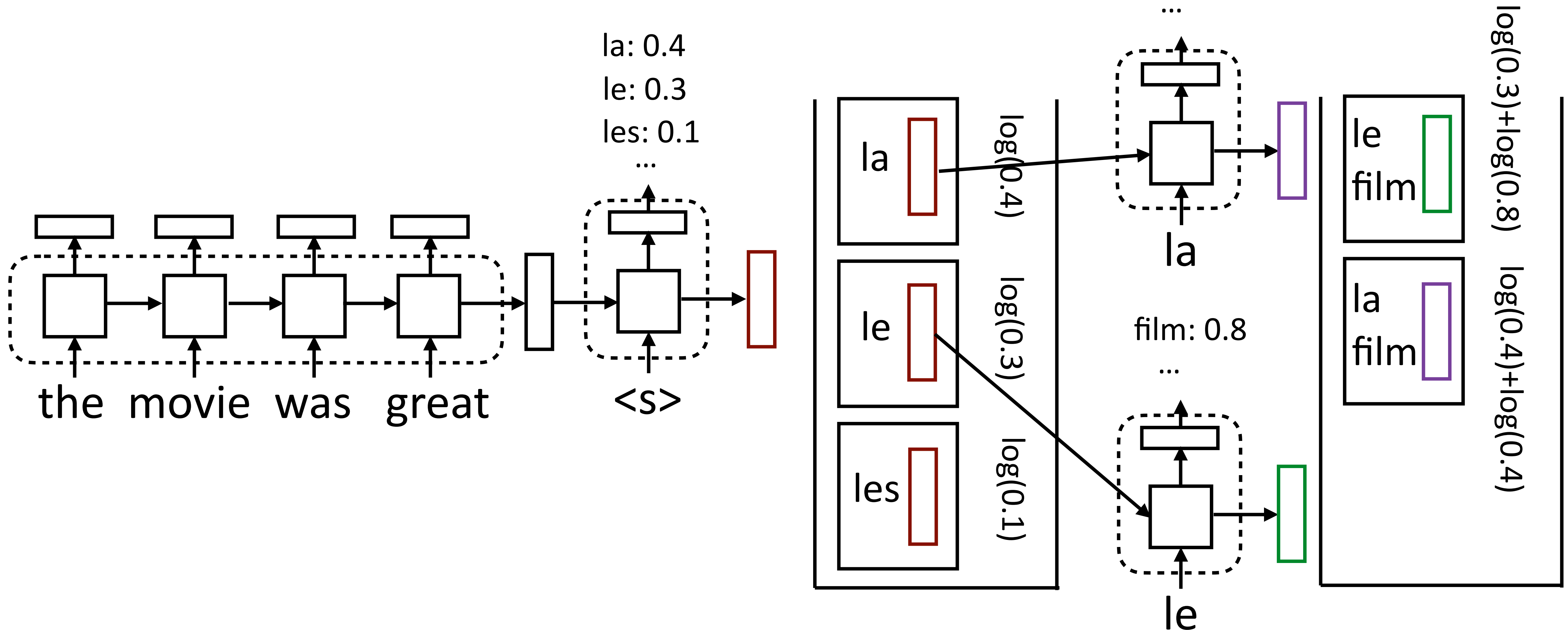
- Maintain decoder state, token history in beam





# Beam Search

- ▶ Maintain decoder state, token history in beam



- ▶ Do **not** max over the two *film* states! Hidden state vectors are different

# Semantic Parsing as Translation

---

*“what states border Texas”*



```
lambda x ( state ( x ) and border ( x , e89 ) ) )
```

# Semantic Parsing as Translation

---

*“what states border Texas”*



`lambda x ( state ( x ) and border ( x , e89 ) ) )`

- ▶ Write down a linearized form of the semantic parse, train seq2seq models to directly translate into this representation

# Semantic Parsing as Translation

---

*“what states border Texas”*



`lambda x ( state ( x ) and border ( x , e89 ) ) )`

- ▶ Write down a linearized form of the semantic parse, train seq2seq models to directly translate into this representation
- ▶ No need to have an explicit grammar, simplifies algorithms

# Semantic Parsing as Translation

---

*“what states border Texas”*



`lambda x ( state ( x ) and border ( x , e89 ) ) )`

- ▶ Write down a linearized form of the semantic parse, train seq2seq models to directly translate into this representation
- ▶ No need to have an explicit grammar, simplifies algorithms
- ▶ Might not produce well-formed logical forms, might require lots of data

# Regex Prediction

---

# Regex Prediction

---

- ▶ Can use for other semantic parsing-like tasks

# Regex Prediction

---

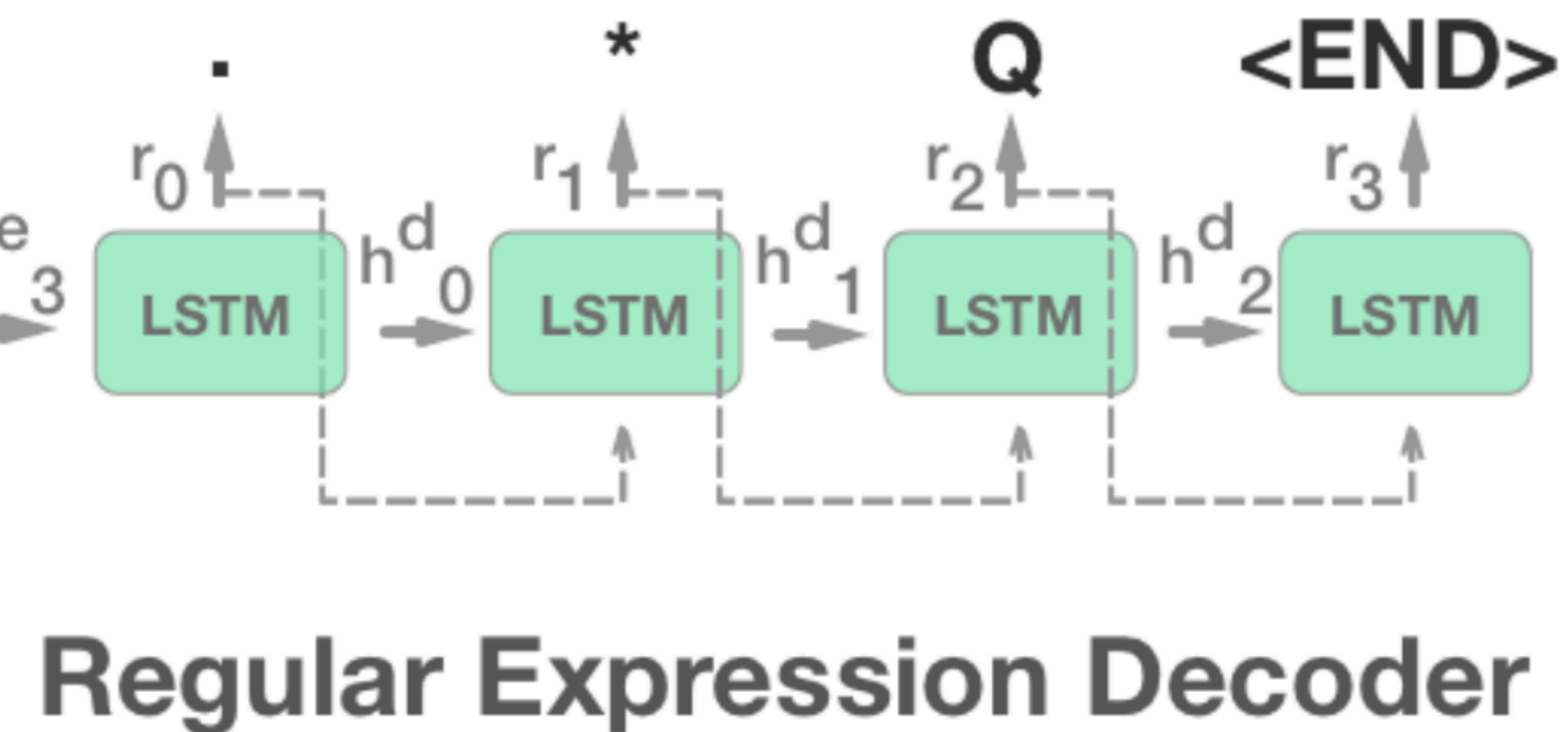
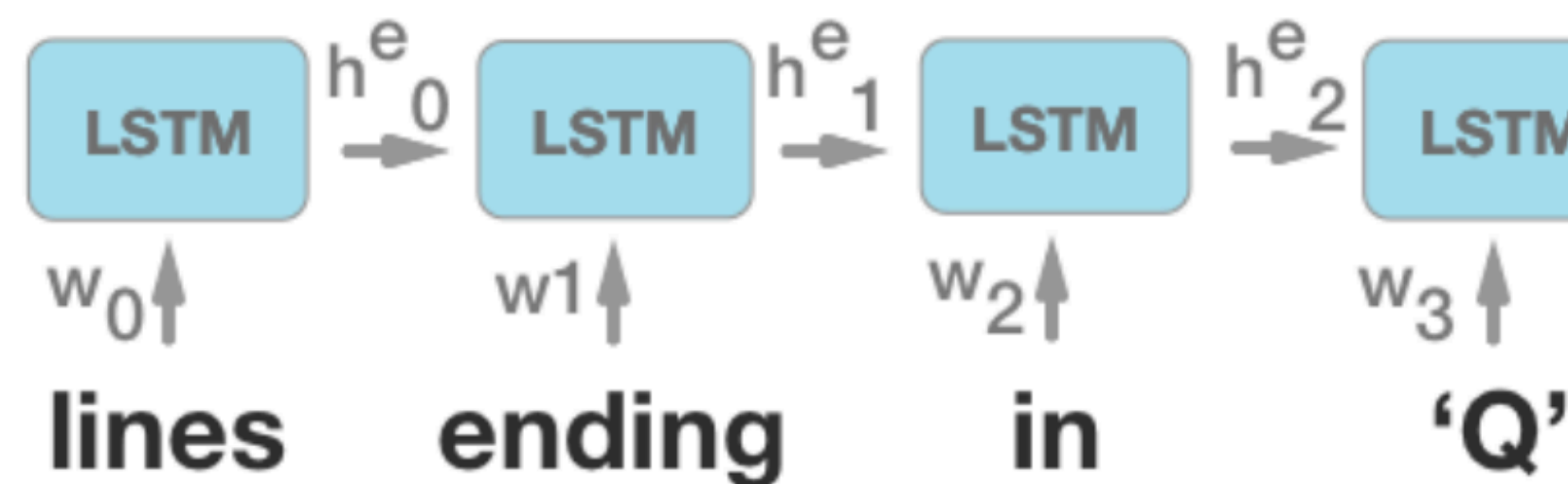
- ▶ Can use for other semantic parsing-like tasks
- ▶ Predict regex from text



# Regex Prediction

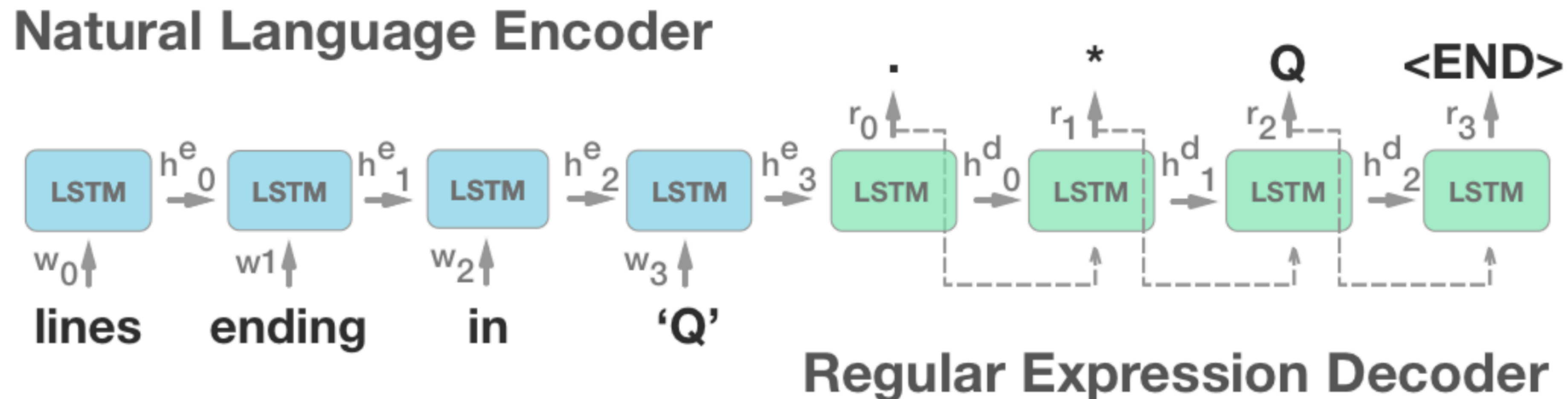
- ▶ Can use for other semantic parsing-like tasks
- ▶ Predict regex from text

## Natural Language Encoder



# Regex Prediction

- ▶ Can use for other semantic parsing-like tasks
- ▶ Predict regex from text



- ▶ Problem: requires a lot of data: 10,000 examples needed to get ~60% accuracy on pretty simple regexes

# SQL Generation

---

- ▶ Convert natural language description into a SQL query against some DB

Question:

How many CFL teams are from York College?

SQL:

```
SELECT COUNT CFL Team FROM  
CFLDraft WHERE College = "York"
```

# SQL Generation

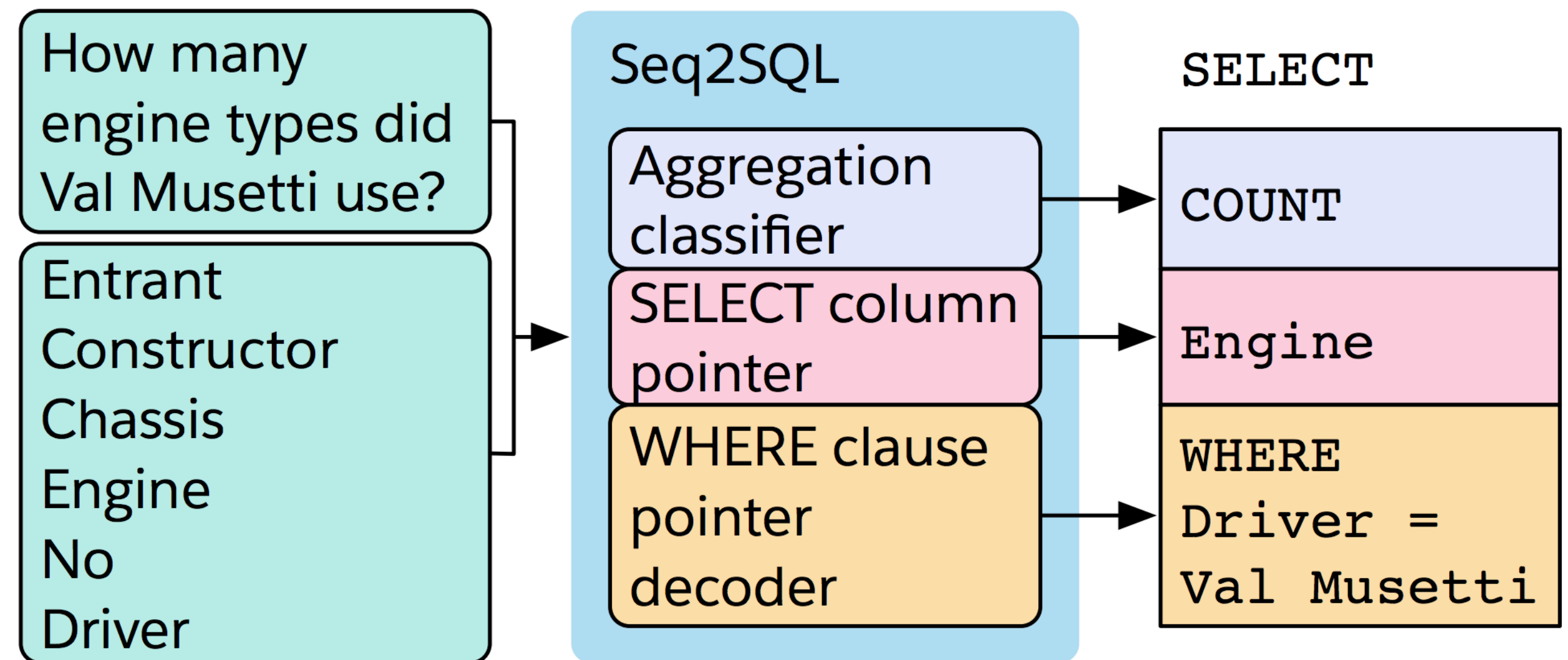
- Convert natural language description into a SQL query against some DB

Question:

How many CFL teams are from York College?

SQL:

```
SELECT COUNT CFL Team FROM  
CFLDraft WHERE College = "York"
```



Zhong et al. (2017)

# SQL Generation

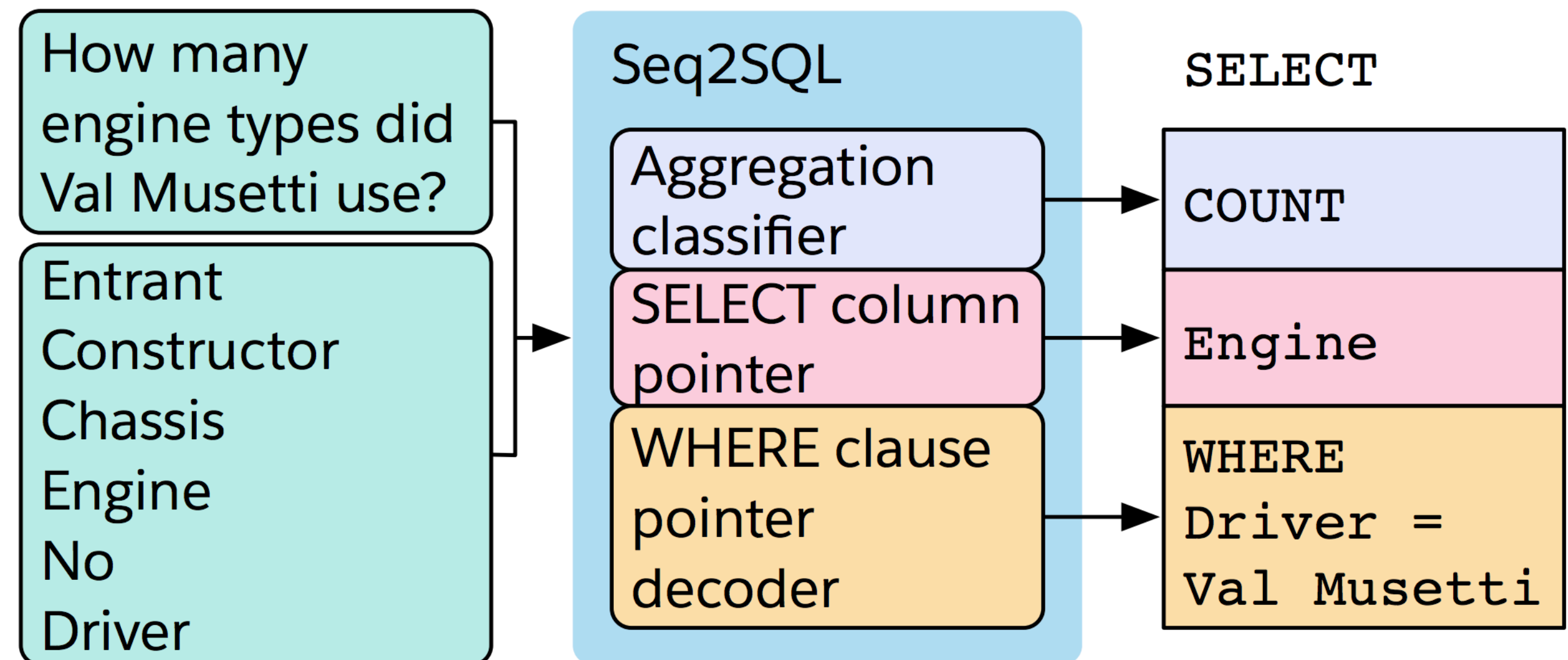
- ▶ Convert natural language description into a SQL query against some DB
- ▶ How to ensure that well-formed SQL is generated?

Question:

How many CFL teams are from York College?

SQL:

```
SELECT COUNT CFL Team FROM  
CFLDraft WHERE College = "York"
```



Zhong et al. (2017)



# SQL Generation

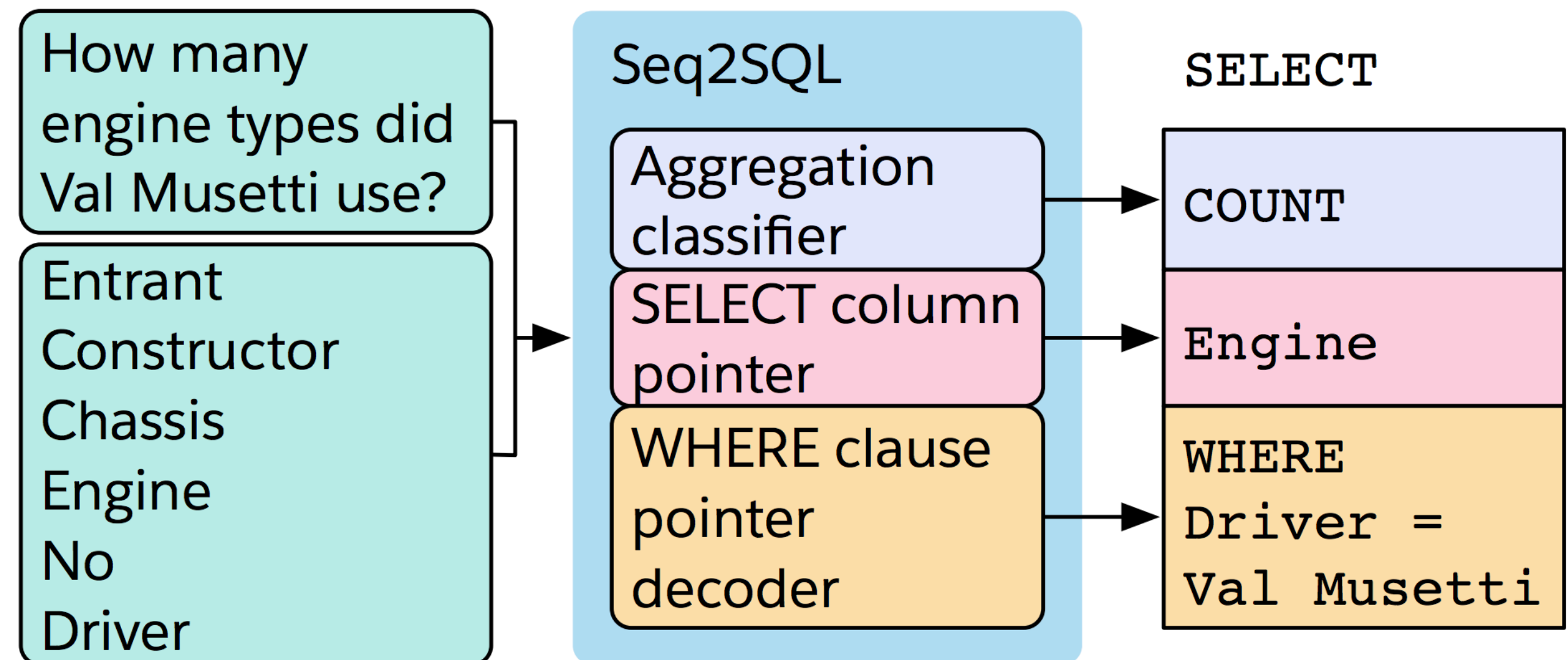
- ▶ Convert natural language description into a SQL query against some DB
- ▶ How to ensure that well-formed SQL is generated?
  - ▶ Three seq2seq models

Question:

How many CFL teams are from York College?

SQL:

```
SELECT COUNT CFL Team FROM  
CFLDraft WHERE College = "York"
```



Zhong et al. (2017)

# SQL Generation

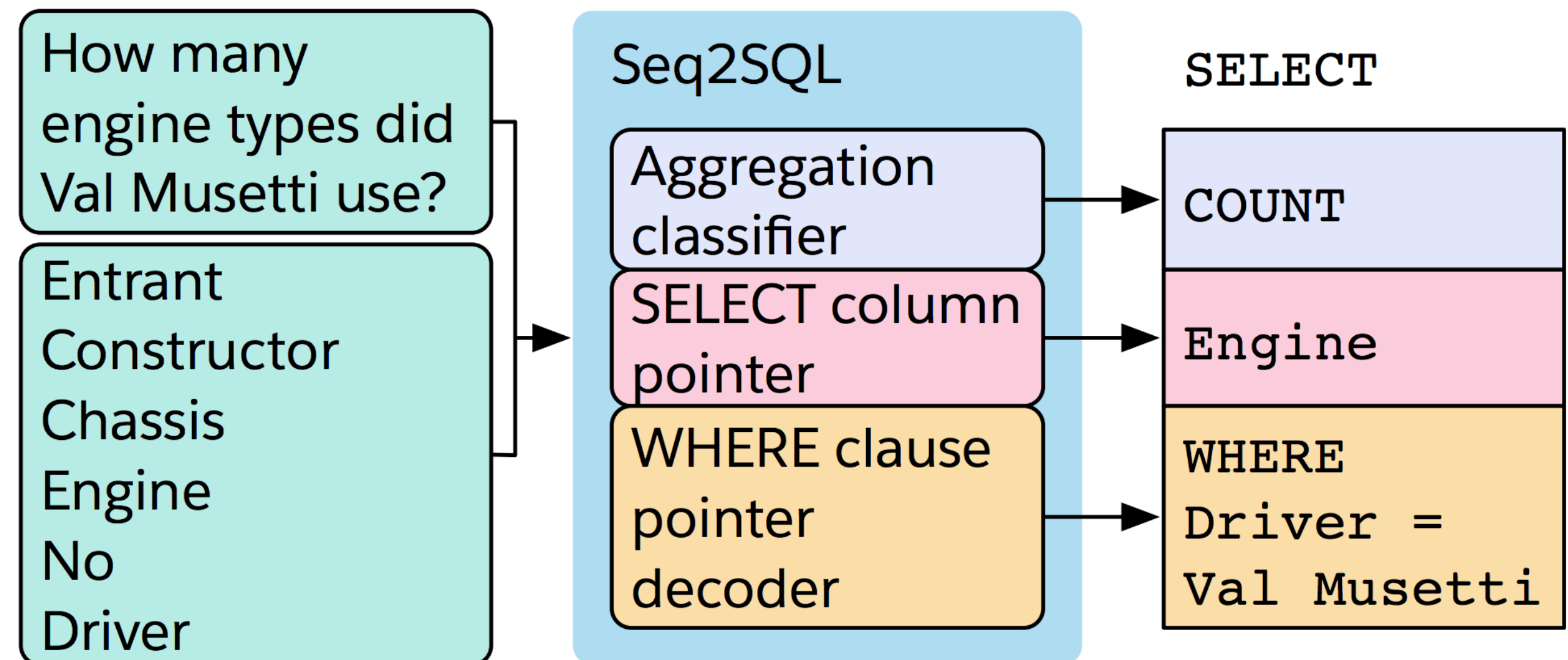
- ▶ Convert natural language description into a SQL query against some DB
- ▶ How to ensure that well-formed SQL is generated?
  - ▶ Three seq2seq models
- ▶ How to capture column names + constants?

Question:

How many CFL teams are from York College?

SQL:

```
SELECT COUNT CFL Team FROM  
CFLDraft WHERE College = "York"
```



Zhong et al. (2017)

# SQL Generation

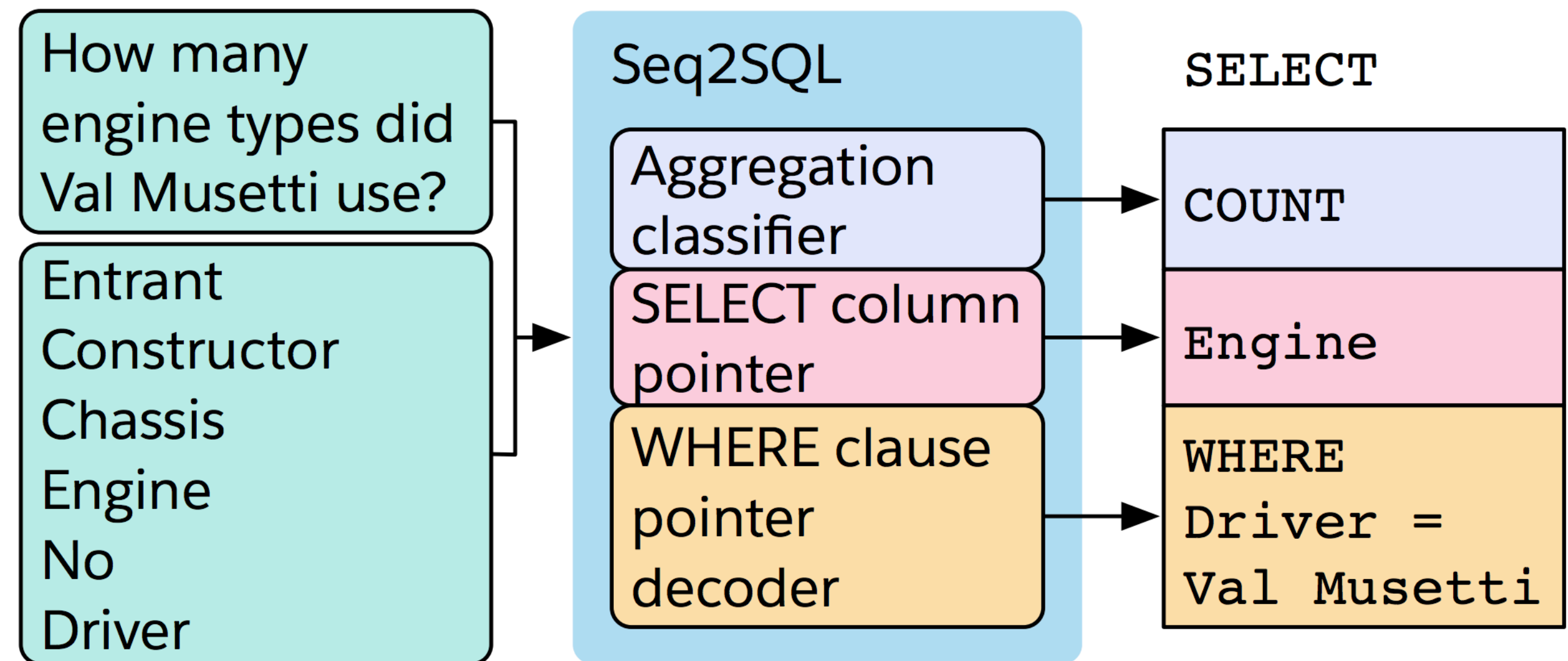
- ▶ Convert natural language description into a SQL query against some DB
- ▶ How to ensure that well-formed SQL is generated?
  - ▶ Three seq2seq models
- ▶ How to capture column names + constants?
  - ▶ Pointer mechanisms

Question:

How many CFL teams are from York College?

SQL:

```
SELECT COUNT CFL Team FROM  
CFLDraft WHERE College = "York"
```



Zhong et al. (2017)



Attention

# Problems with Seq2seq Models

---

- ▶ Encoder-decoder models like to repeat themselves:

# Problems with Seq2seq Models

---

- ▶ Encoder-decoder models like to repeat themselves:

Un garçon joue dans la neige → A boy plays in the snow **boy plays boy plays**

# Problems with Seq2seq Models

---

- ▶ Encoder-decoder models like to repeat themselves:

Un garçon joue dans la neige → A boy plays in the snow **boy plays boy plays**

- ▶ Often a byproduct of training these models poorly

# Problems with Seq2seq Models

---

- ▶ Encoder-decoder models like to repeat themselves:

Un garçon joue dans la neige → A boy plays in the snow **boy plays boy plays**

- ▶ Often a byproduct of training these models poorly
- ▶ Need some notion of input coverage or what input words we've translated

# Problems with Seq2seq Models

---

- ▶ Unknown words:

*en:* The ecotax portico in Pont-de-Buis , ... [truncated] ... , was taken down on Thursday morning

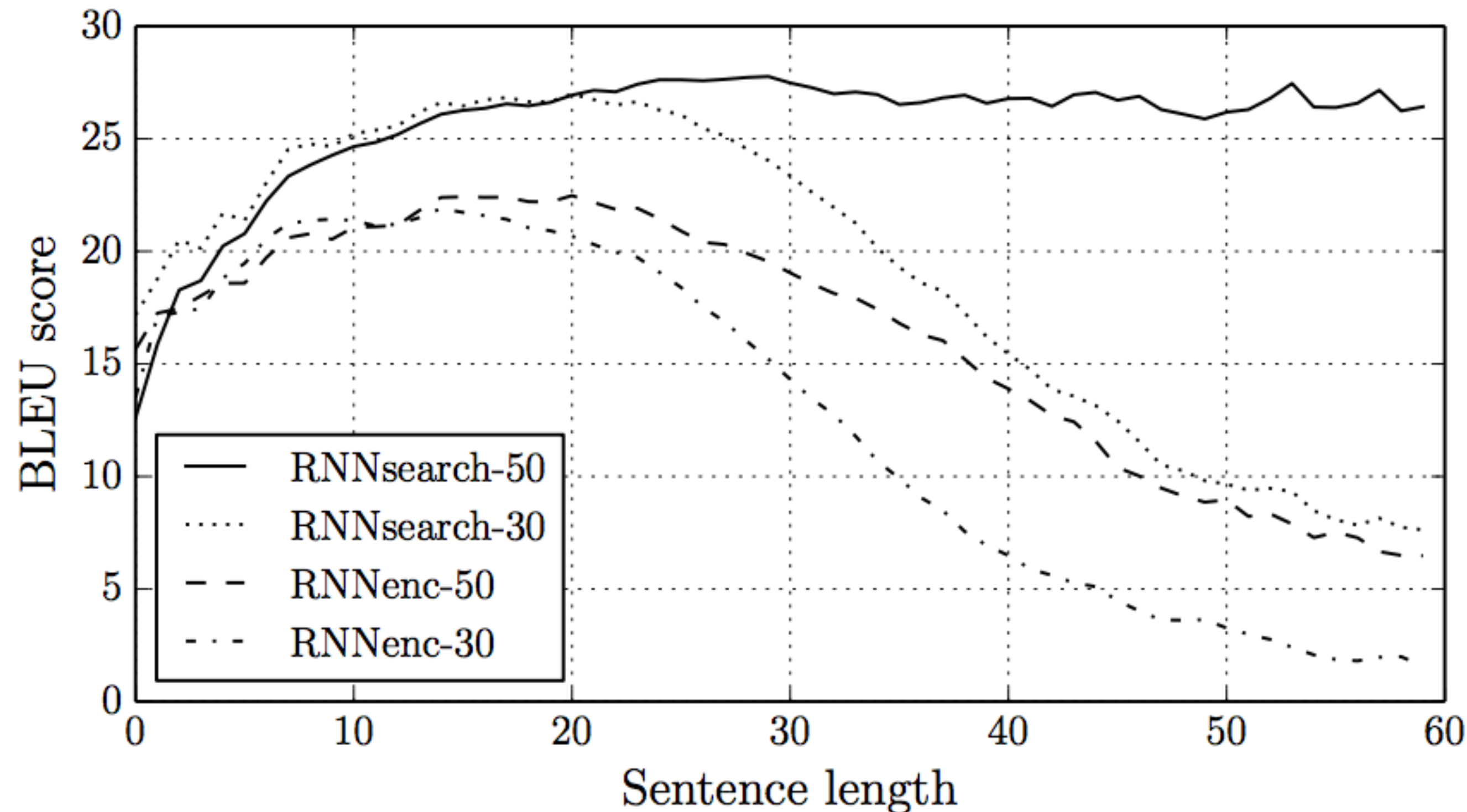
*fr:* Le portique écotaxe de Pont-de-Buis , ... [truncated] ... , a été démonté jeudi matin

*nn:* Le unk de unk à unk , ... [truncated] ... , a été pris le jeudi matin

- ▶ No matter how much data you have, you'll need some mechanism to copy a word like Pont-de-Buis from the source to target

# Problems with Seq2seq Models

- ▶ Bad at long sentences: 1) a fixed-size representation doesn't scale; 2) LSTMs still have a hard time remembering for really long periods of time



RNNsearch: introduces attention mechanism to give “variable-sized” representation

# Aligned Inputs

---

- ▶ Suppose we knew the source and target would be word-by-word translated



# Aligned Inputs

---

- ▶ Suppose we knew the source and target would be word-by-word translated

the movie was great  
/ / / /  
le film était bon

# Aligned Inputs

---

- ▶ Suppose we knew the source and target would be word-by-word translated
- ▶ Can look at the corresponding input word when translating — this could scale!

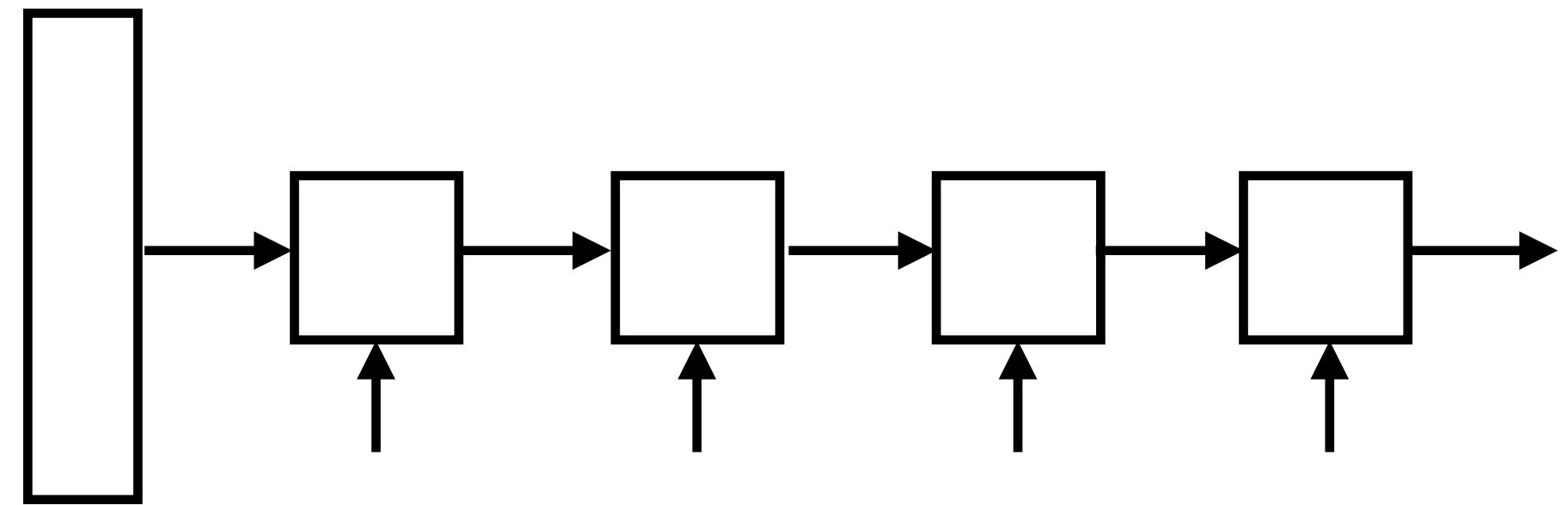
the movie was great  
/ / / /  
le film était bon

# Aligned Inputs

---

- ▶ Suppose we knew the source and target would be word-by-word translated
- ▶ Can look at the corresponding input word when translating — this could scale!

the movie was great  
/ / / /  
le film était bon

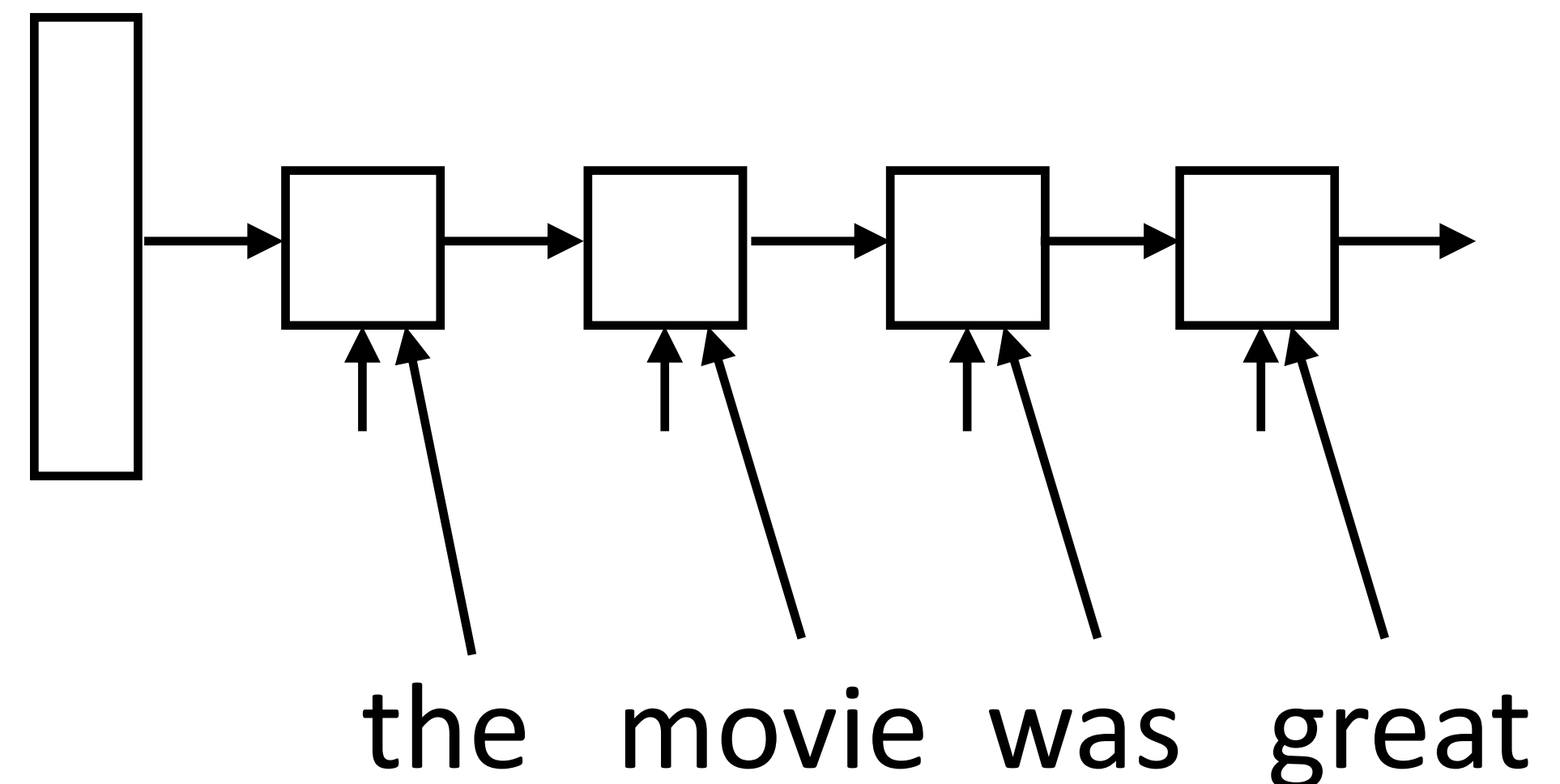


# Aligned Inputs

---

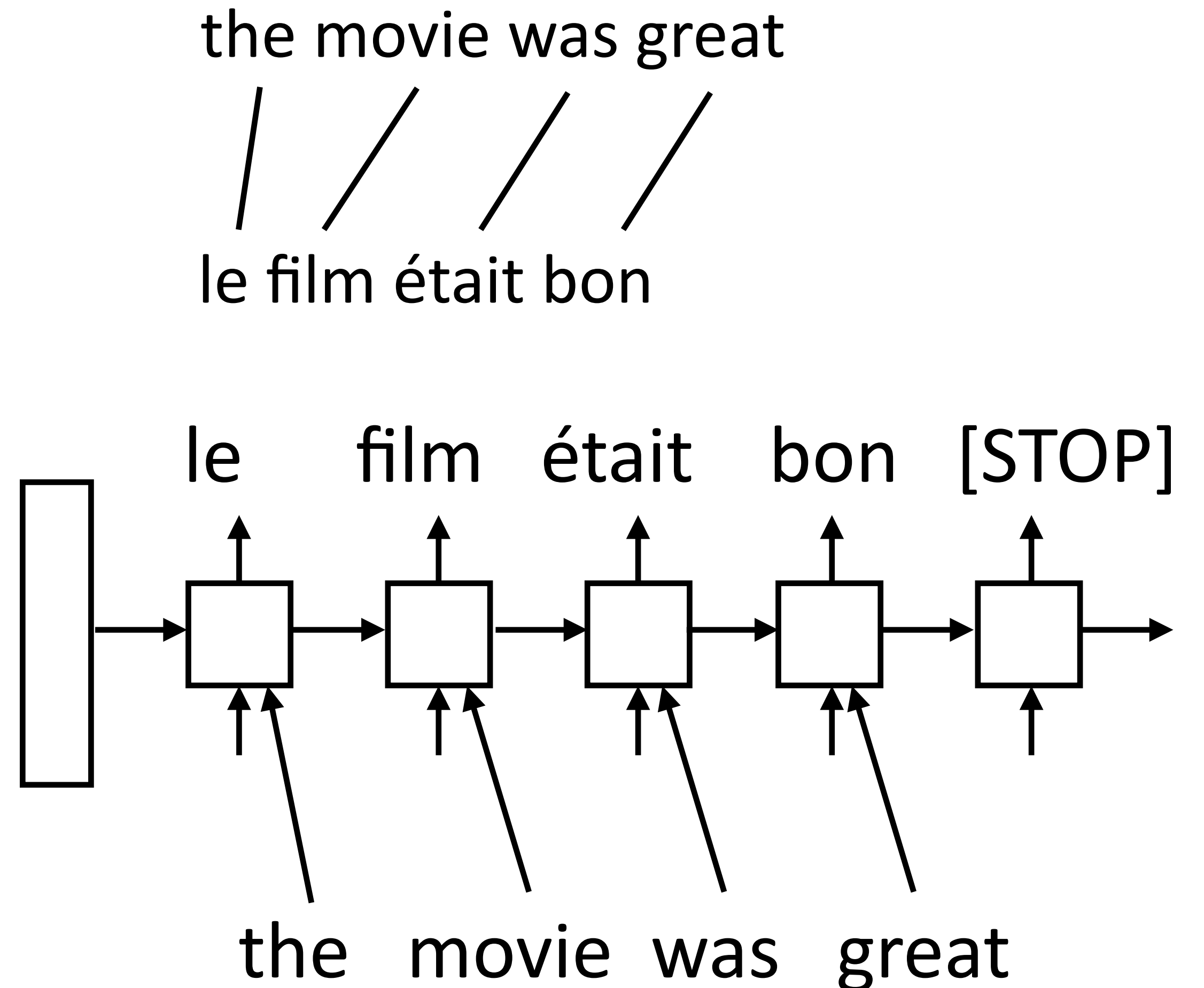
- ▶ Suppose we knew the source and target would be word-by-word translated
- ▶ Can look at the corresponding input word when translating — this could scale!

the movie was great  
/ / / /  
le film était bon



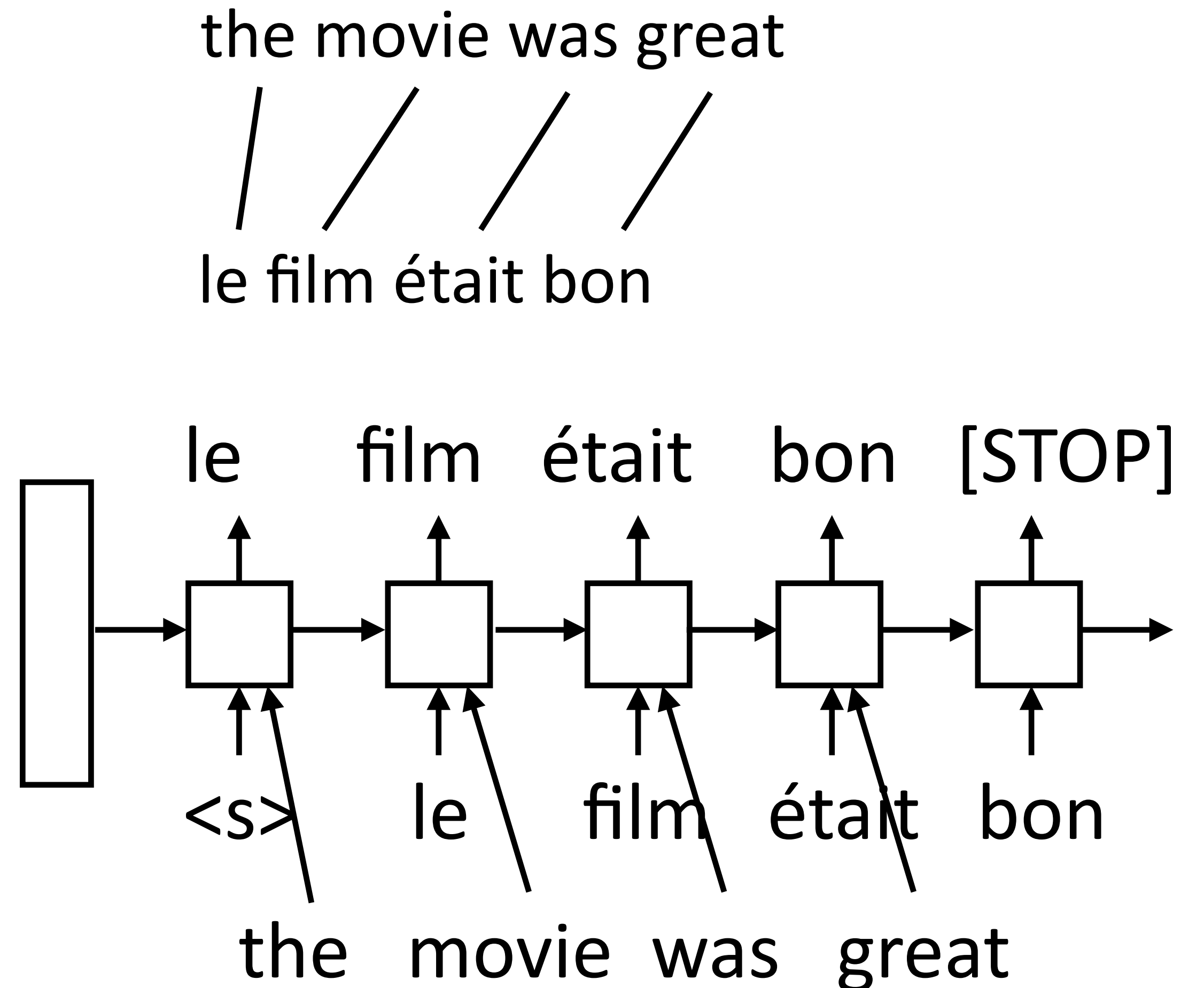
# Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated
- Can look at the corresponding input word when translating — this could scale!



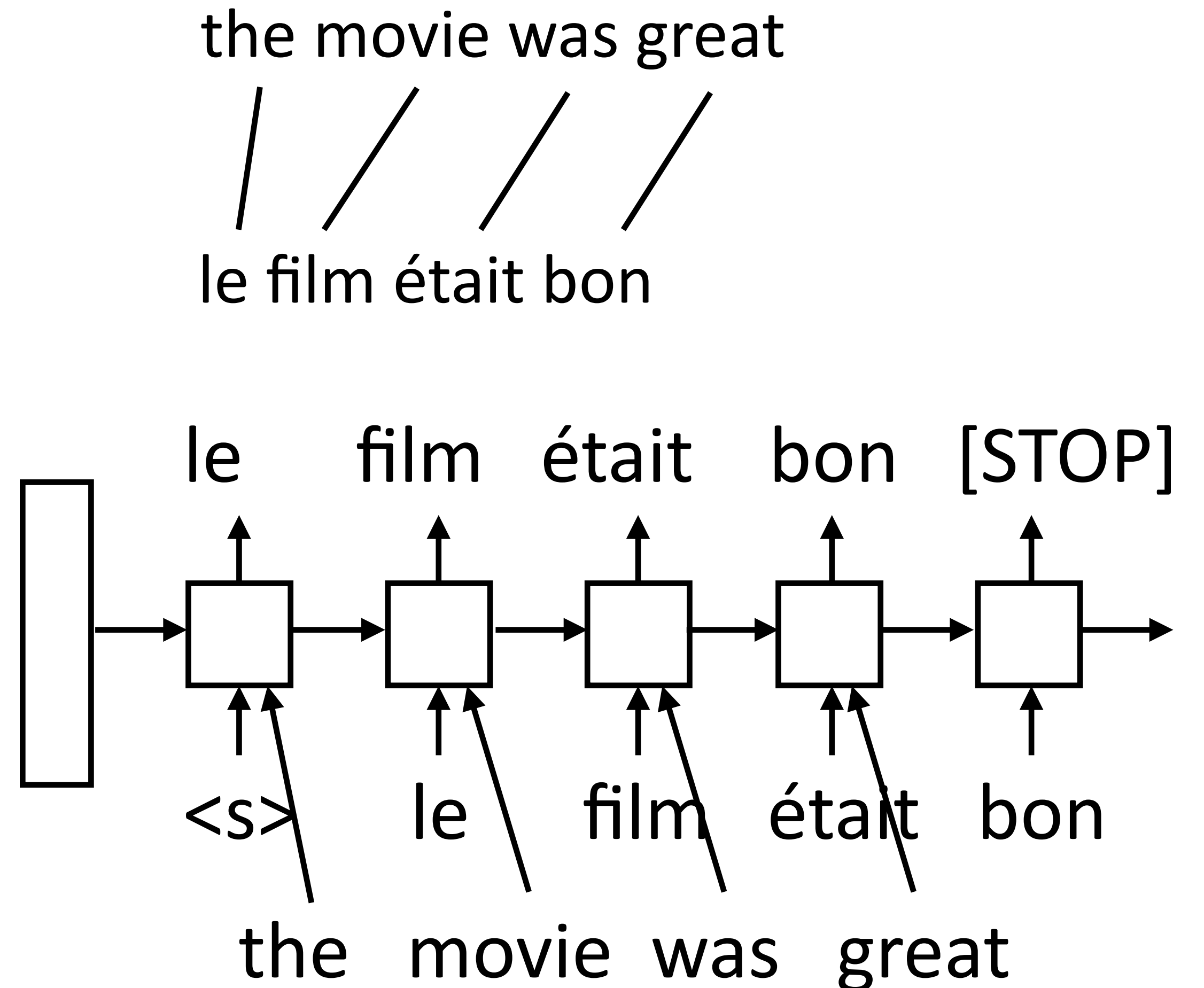
# Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated
- Can look at the corresponding input word when translating — this could scale!



# Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated
- Can look at the corresponding input word when translating — this could scale!
- Much less burden on the hidden state



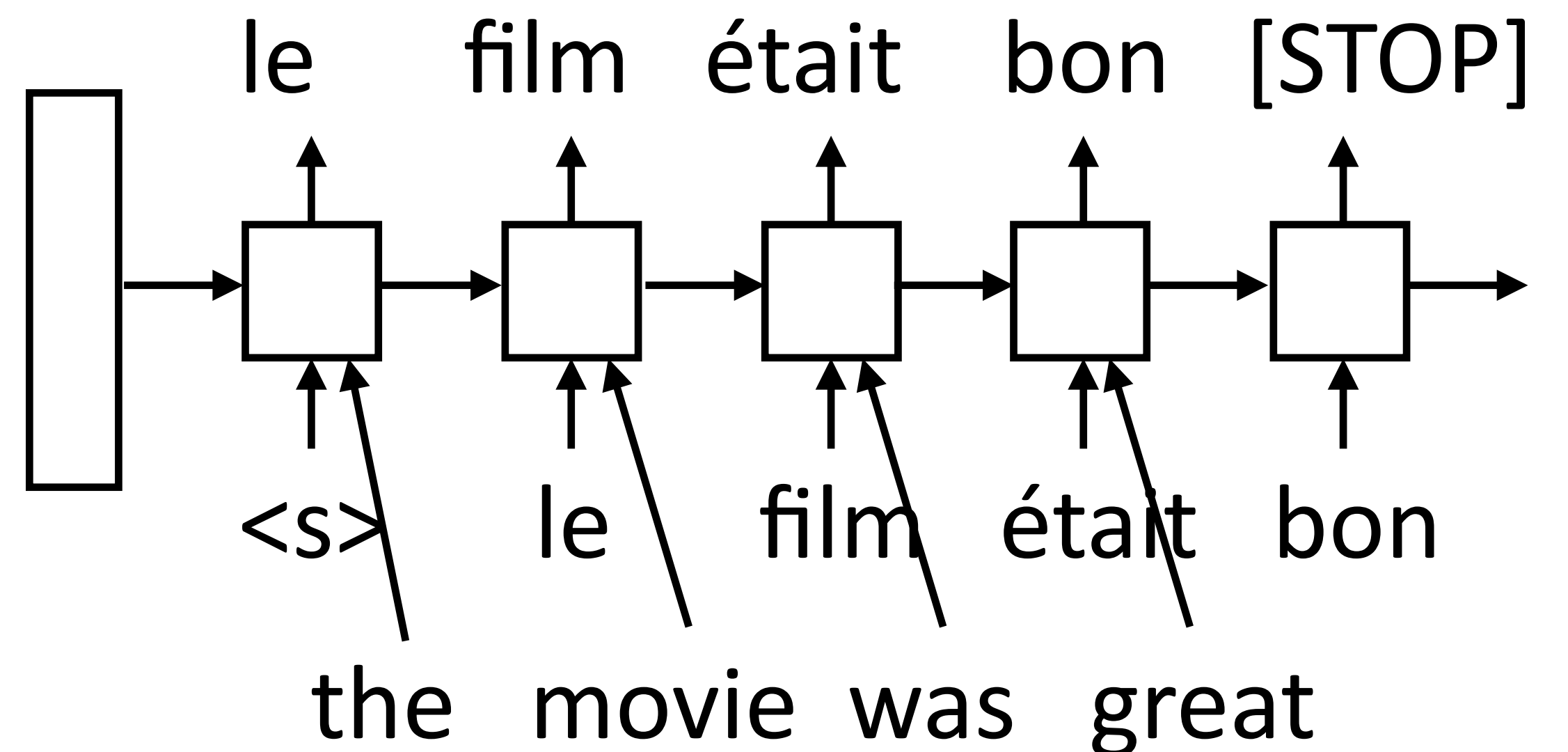
# Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated

the movie was great  
/ / / /  
le film était bon

- Can look at the corresponding input word when translating — this could scale!

- Much less burden on the hidden state

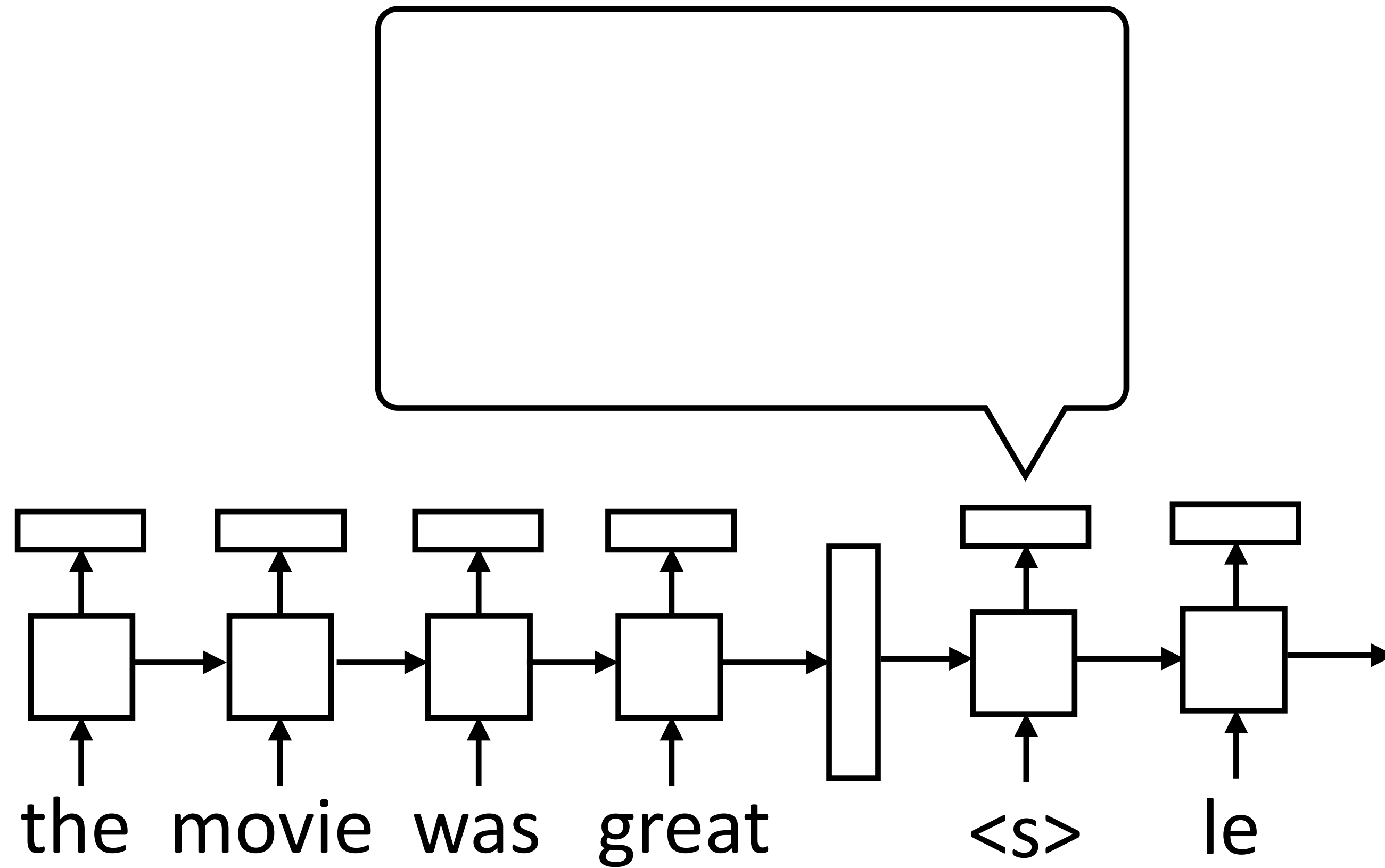


- How can we achieve this without hardcoding it?

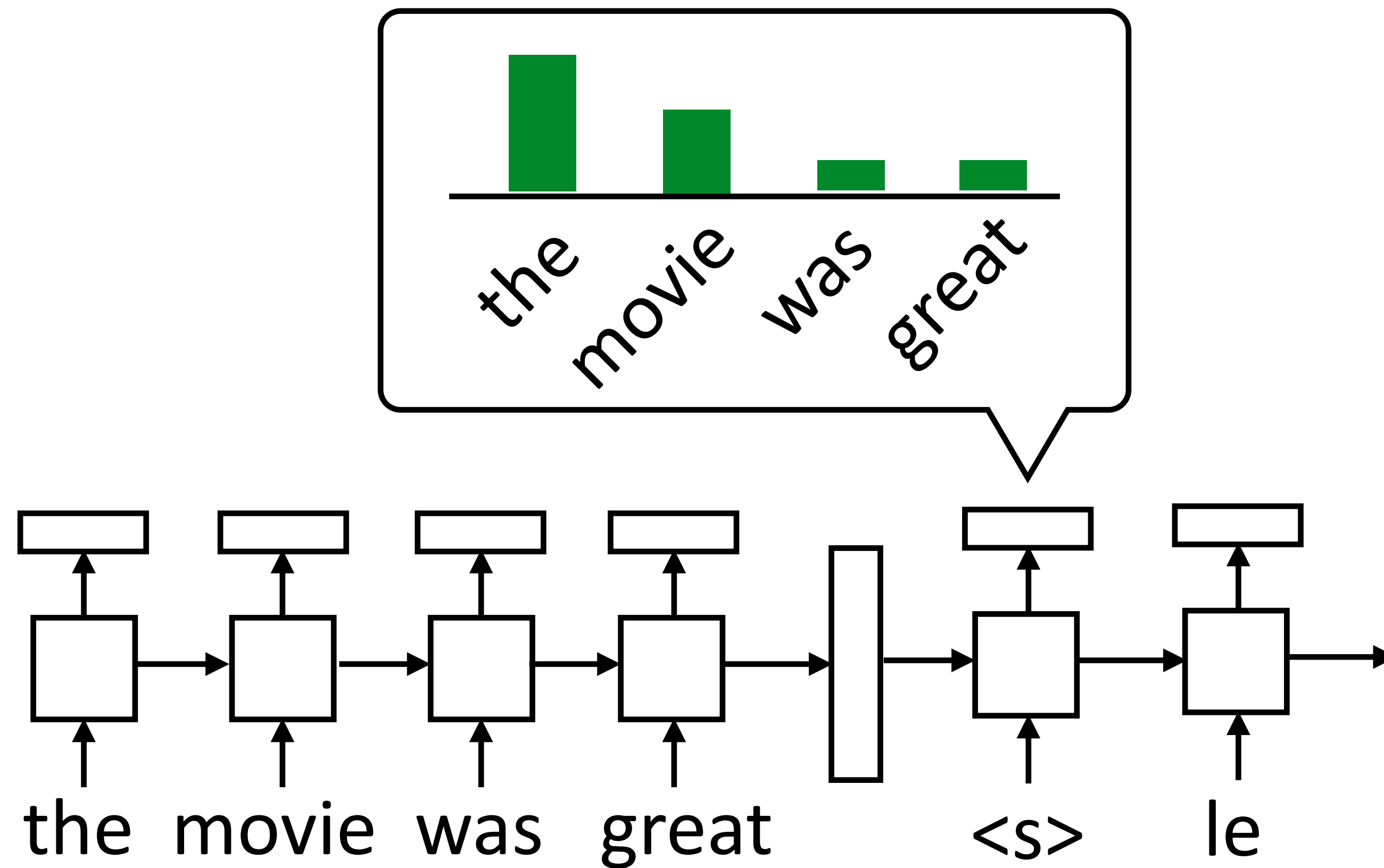


# Attention

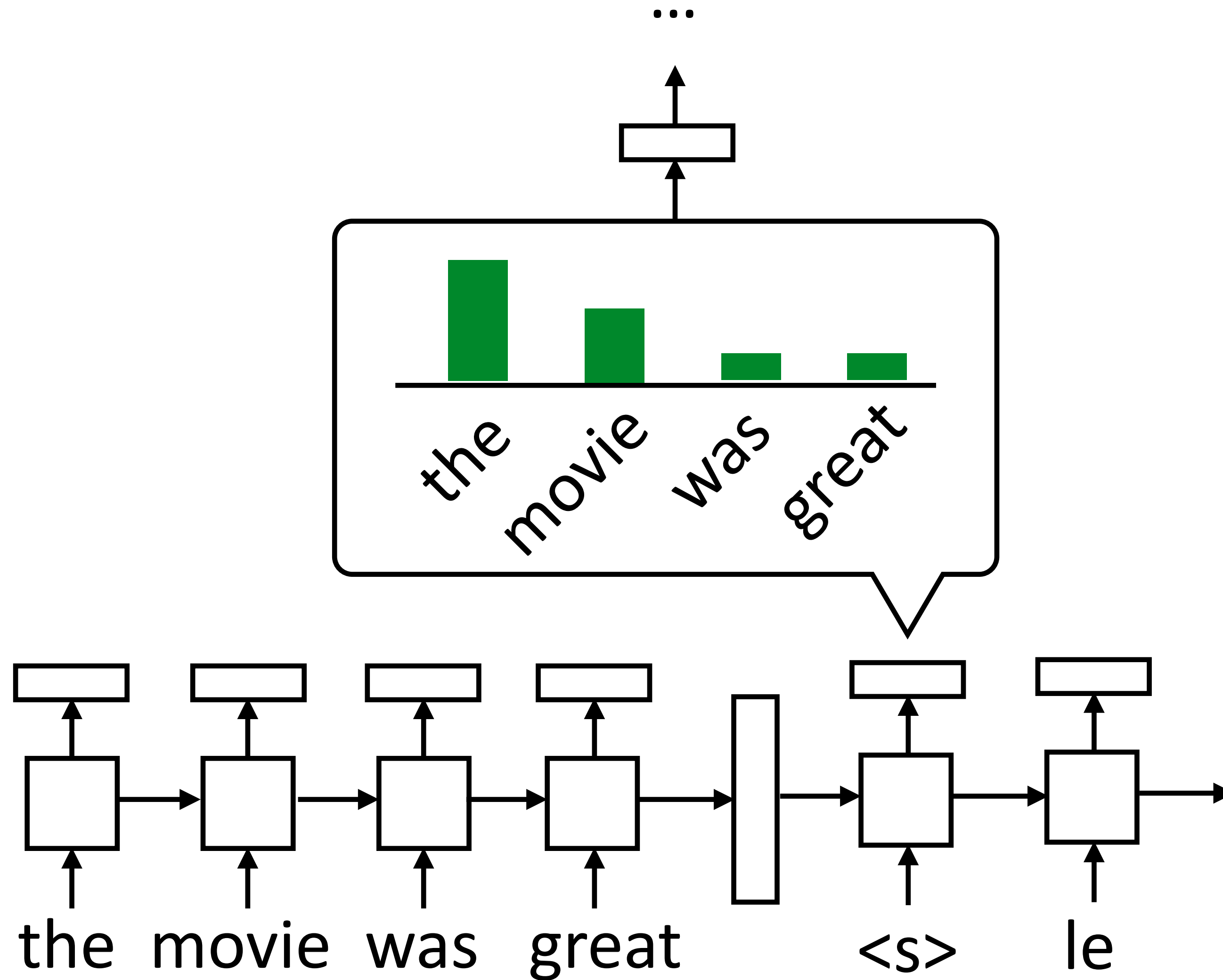
---



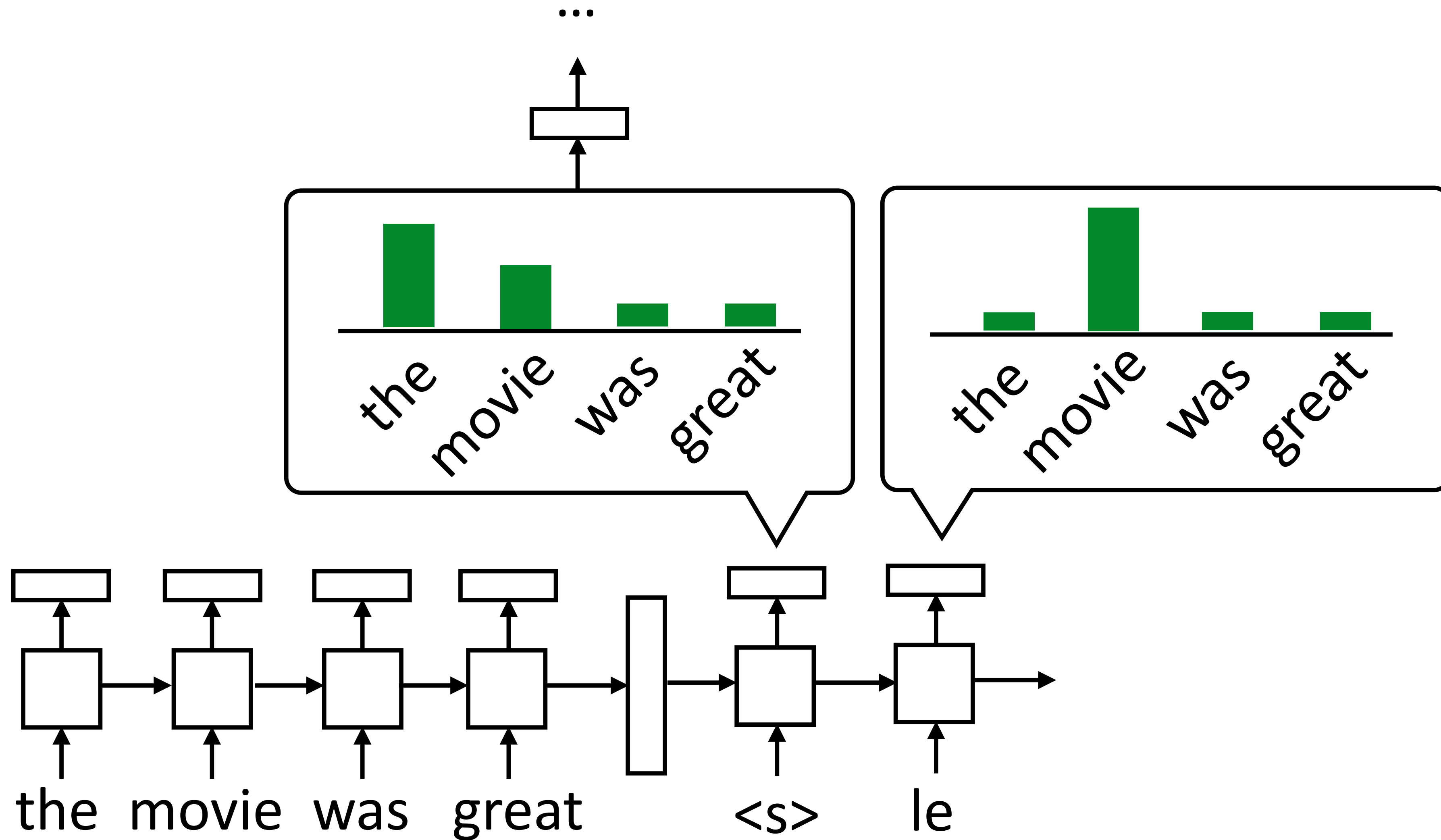
# Attention



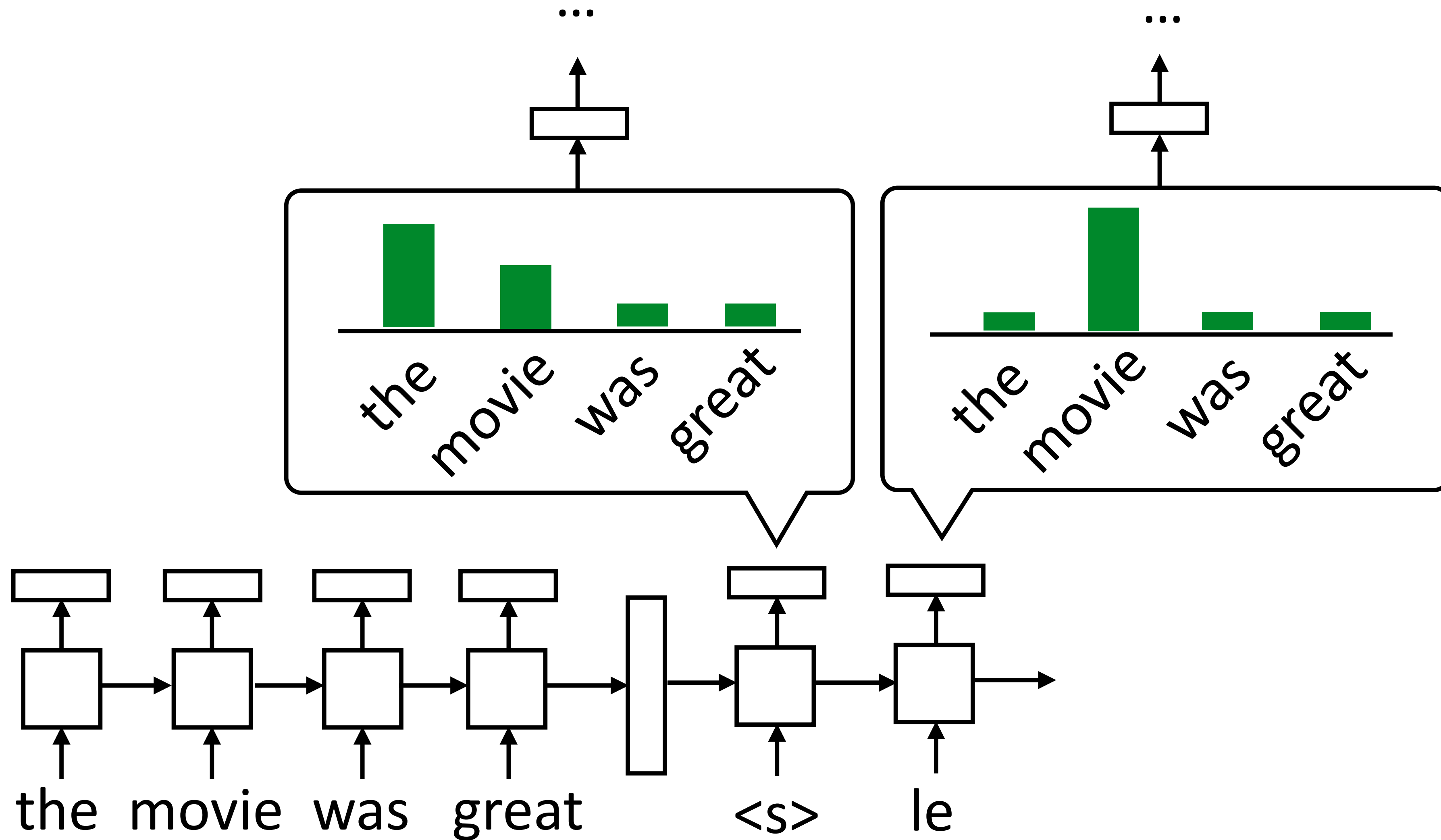
# Attention



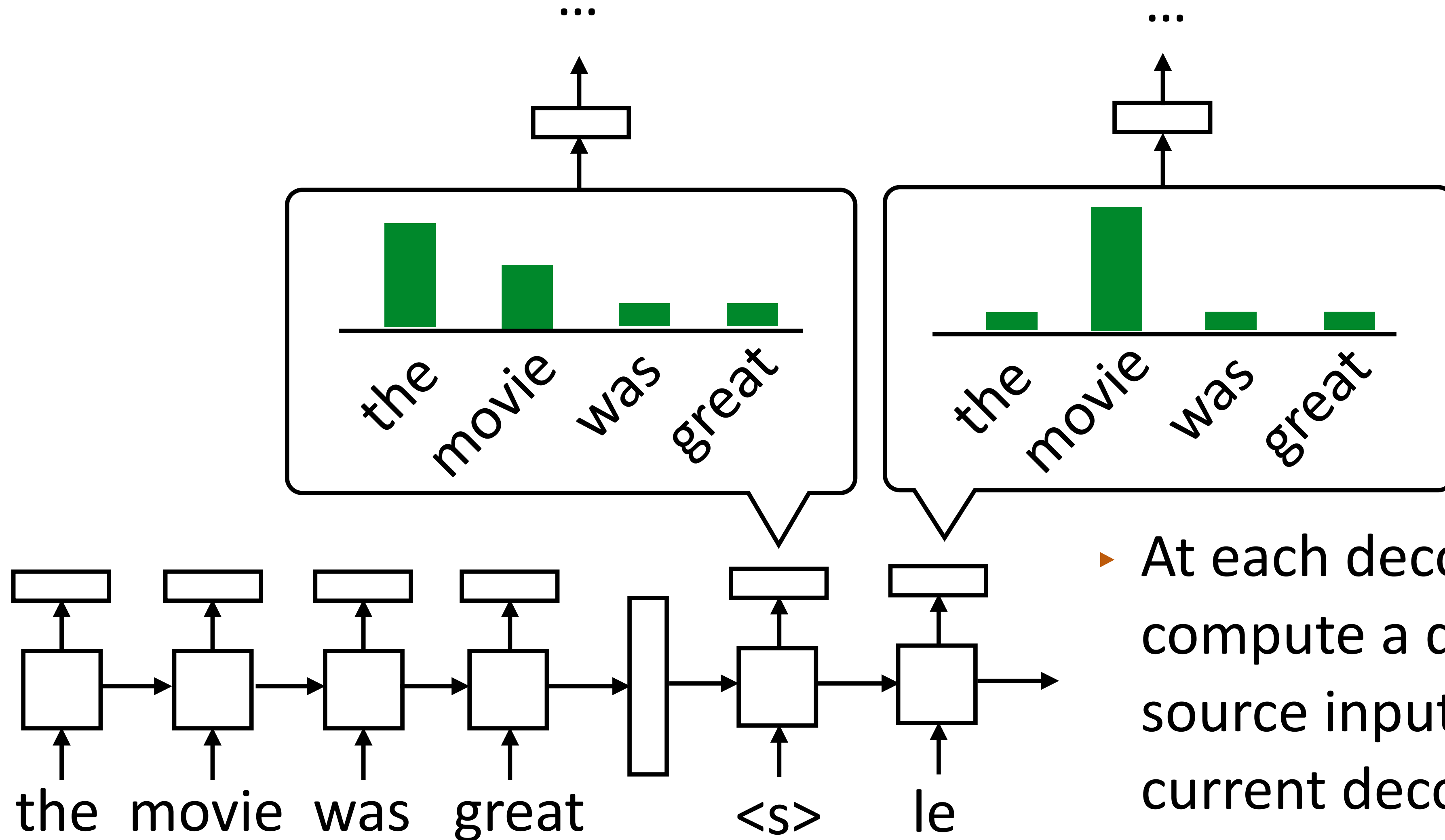
# Attention



# Attention

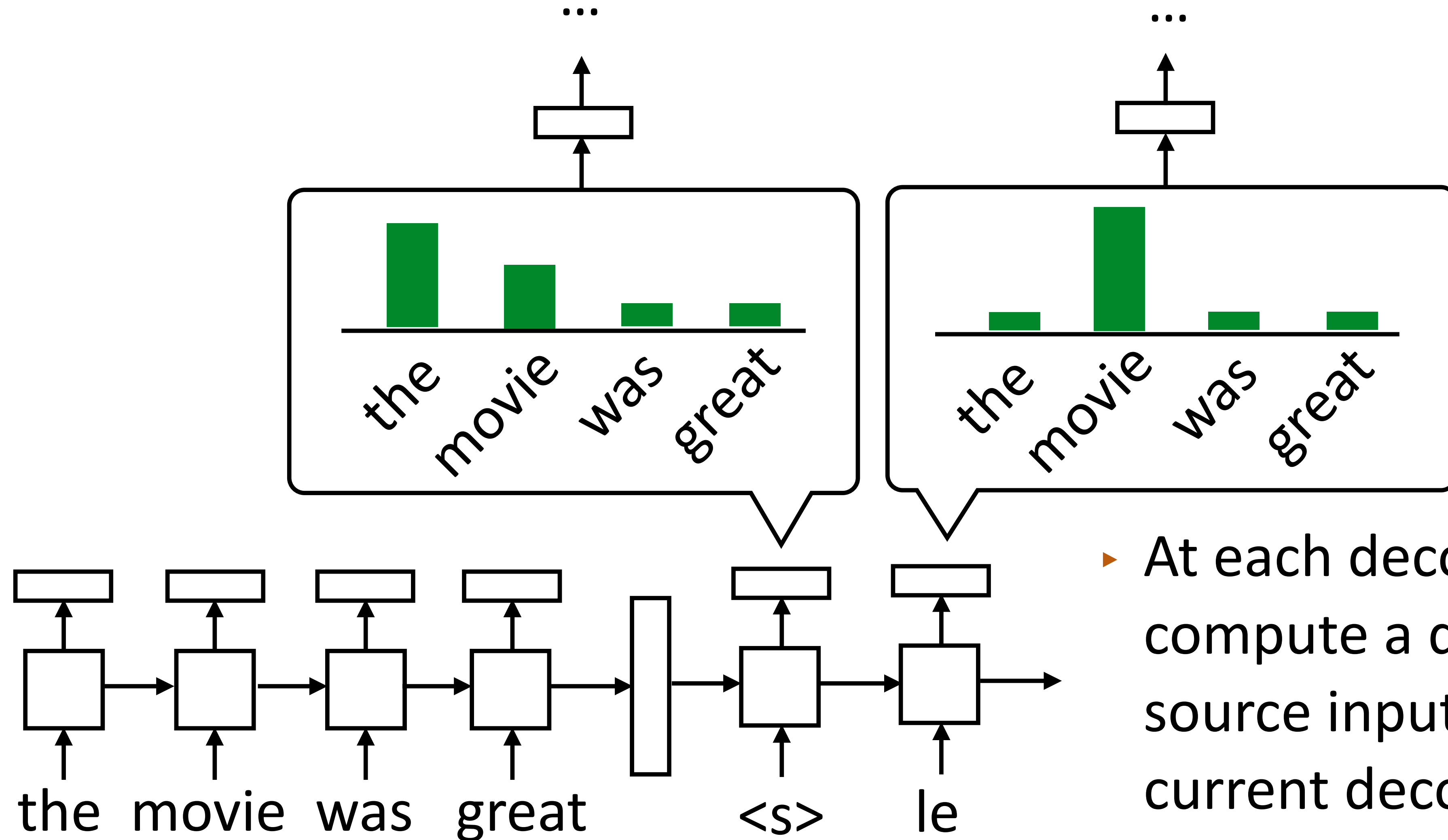


# Attention



- At each decoder state, compute a distribution over source inputs based on current decoder state

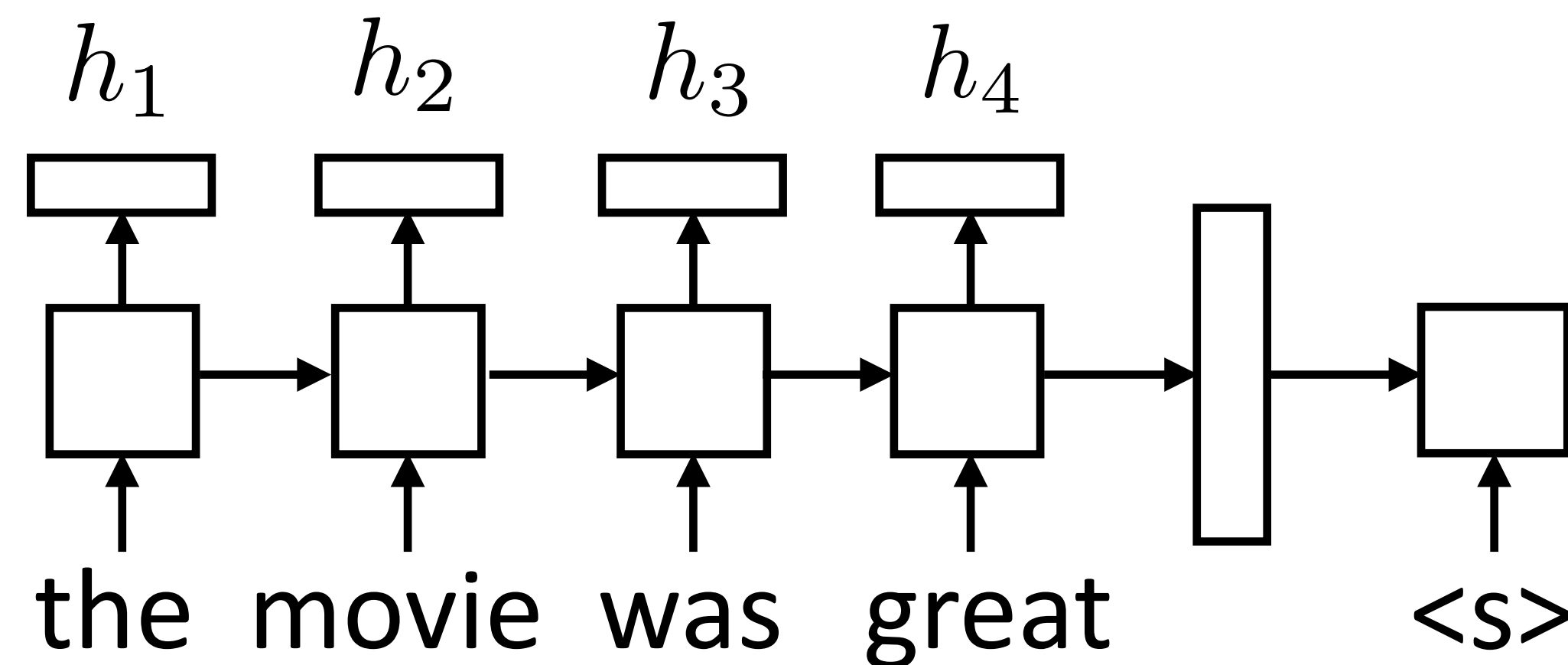
# Attention



- ▶ At each decoder state, compute a distribution over source inputs based on current decoder state
- ▶ Use that in output layer

# Attention

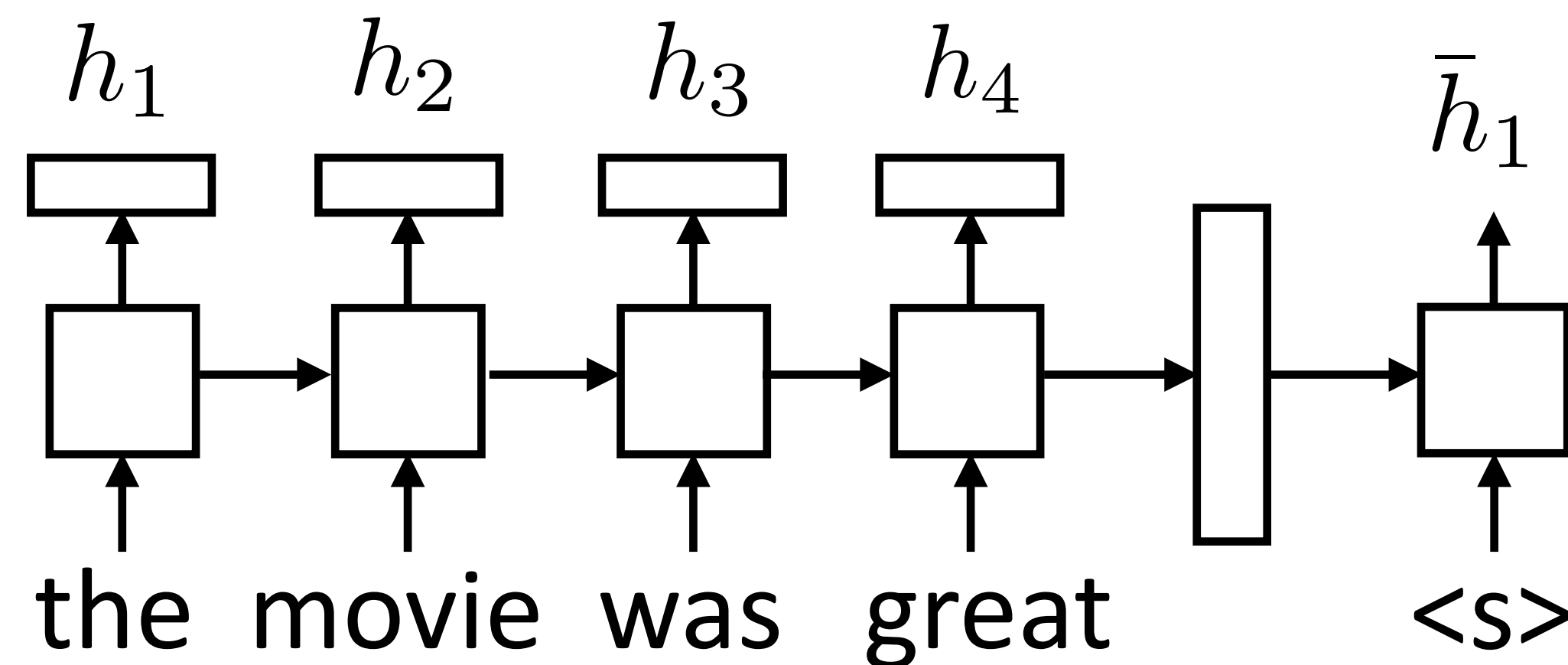
- ▶ For each decoder state, compute weighted sum of input states





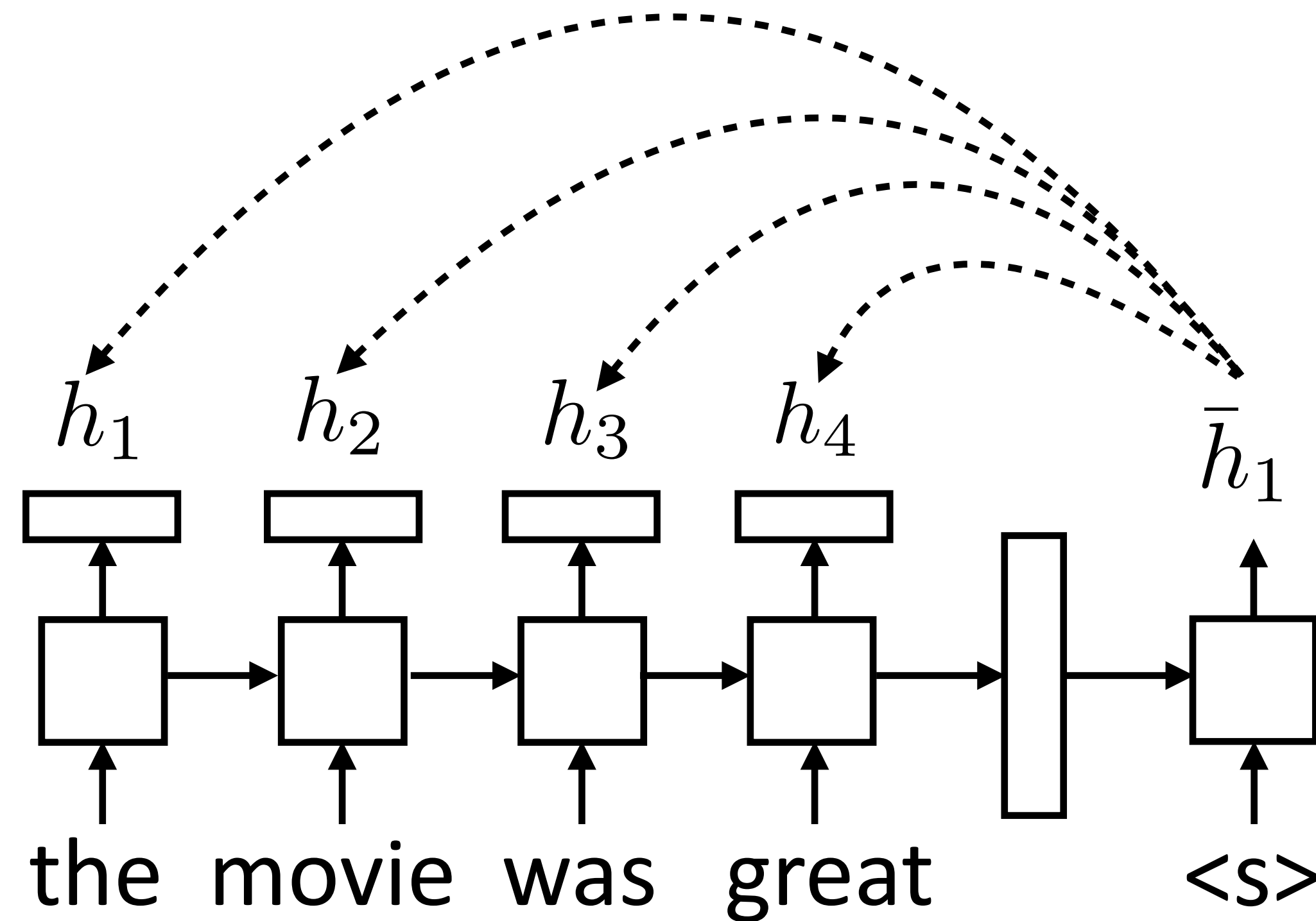
# Attention

- For each decoder state, compute weighted sum of input states



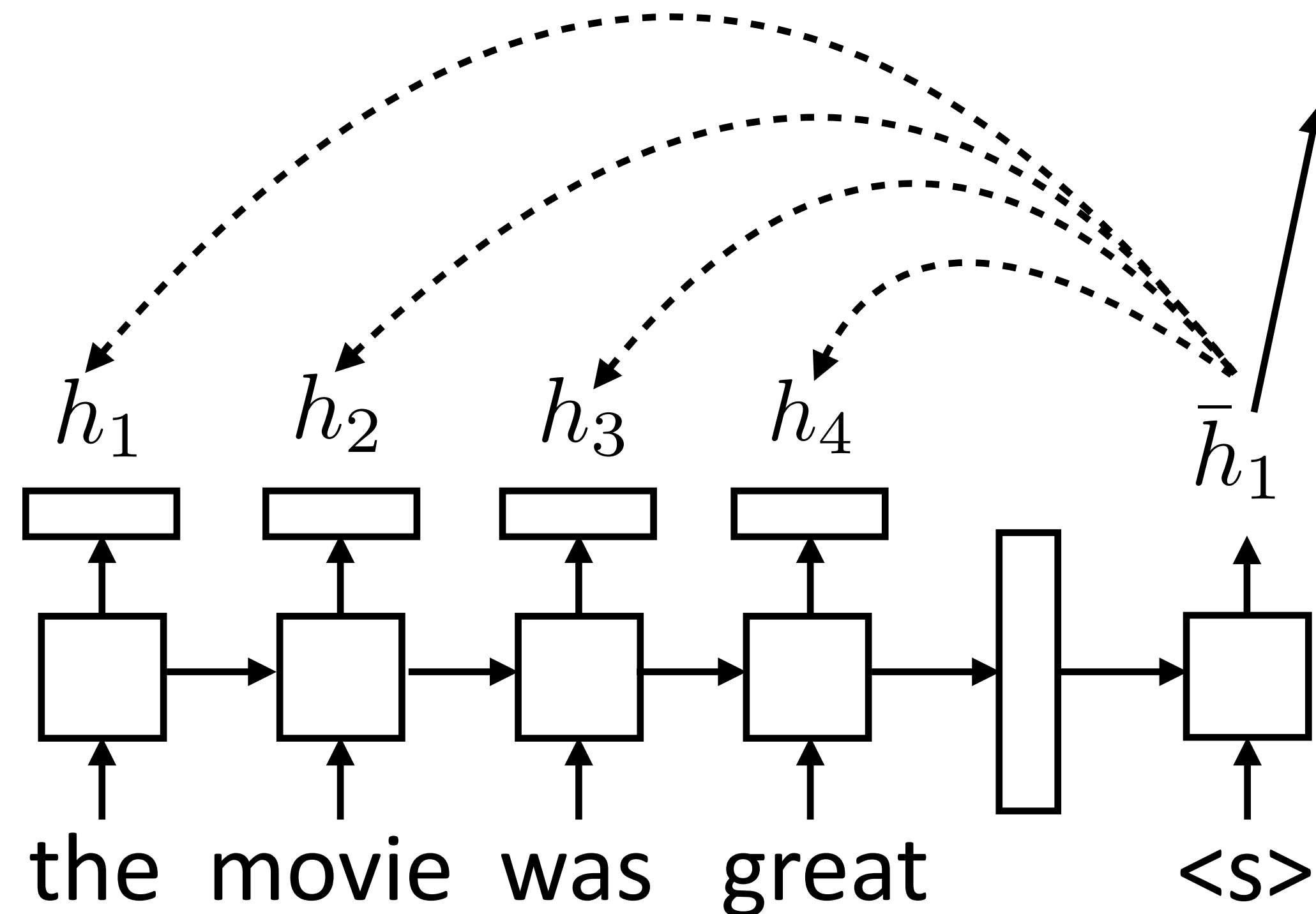
# Attention

- ▶ For each decoder state, compute weighted sum of input states



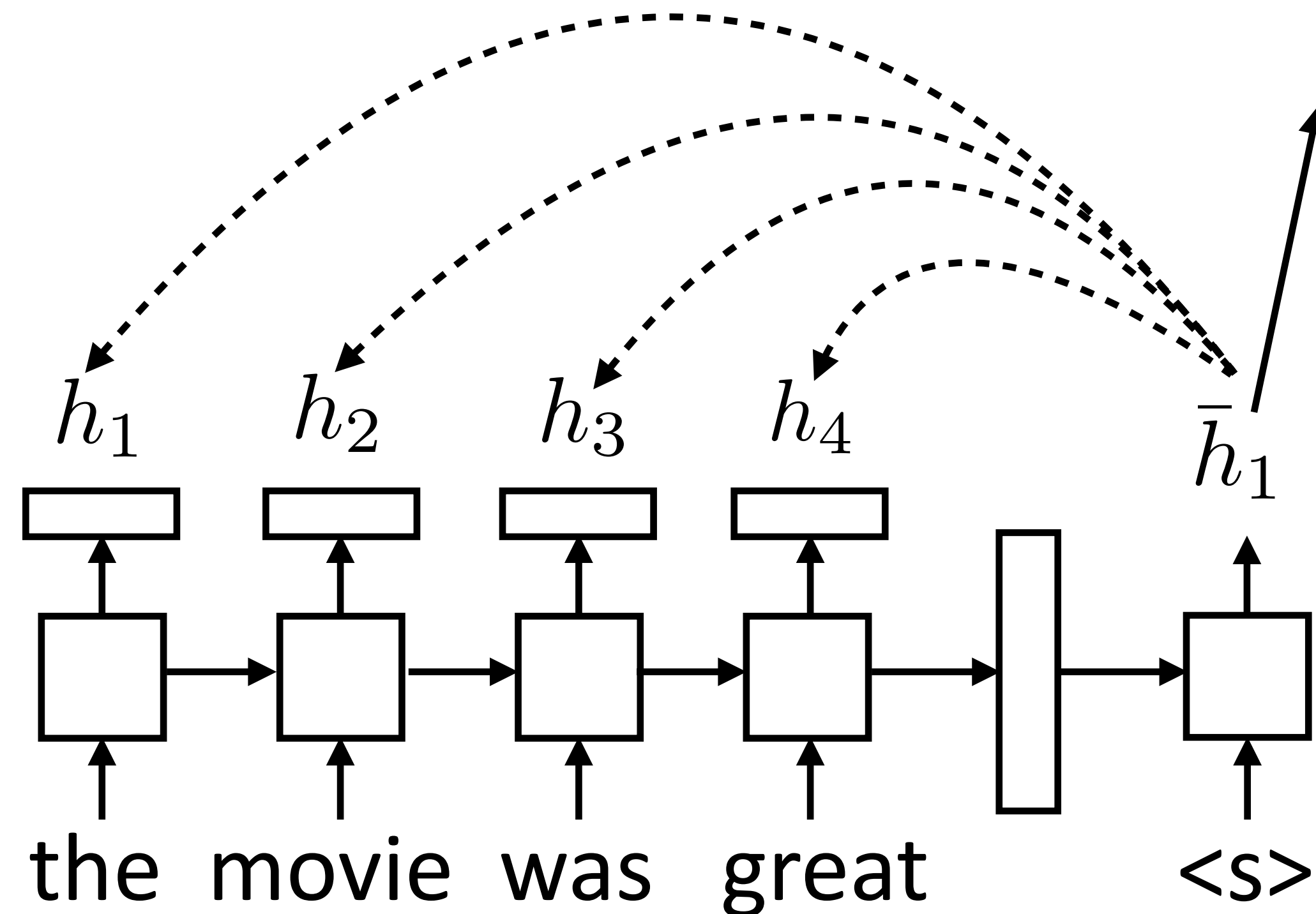
# Attention

- ▶ For each decoder state, compute weighted sum of input states



# Attention

- ▶ For each decoder state, compute weighted sum of input states

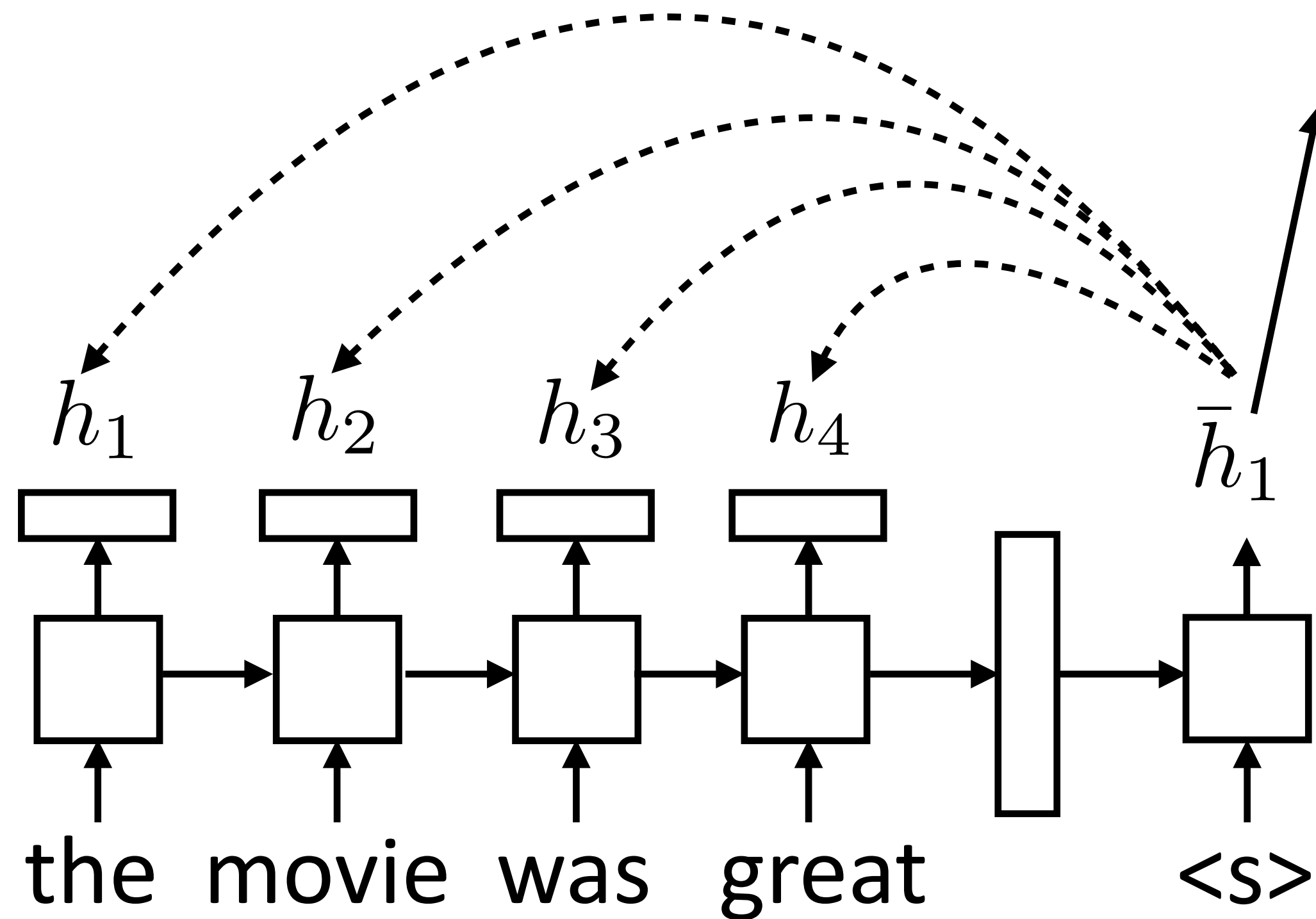


$$e_{ij} = f(\bar{h}_i, h_j)$$

- ▶ Unnormalized scalar weight

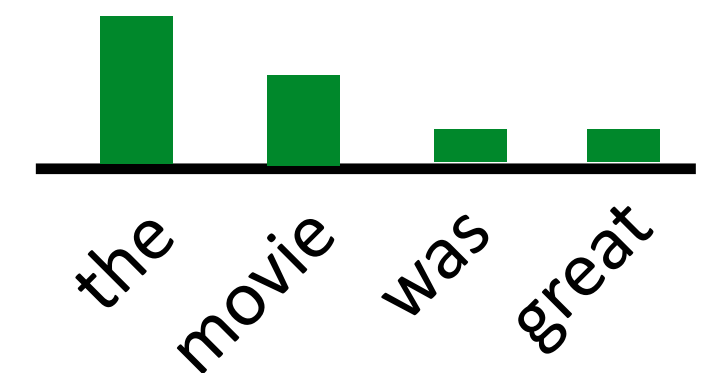
# Attention

- For each decoder state, compute weighted sum of input states



$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

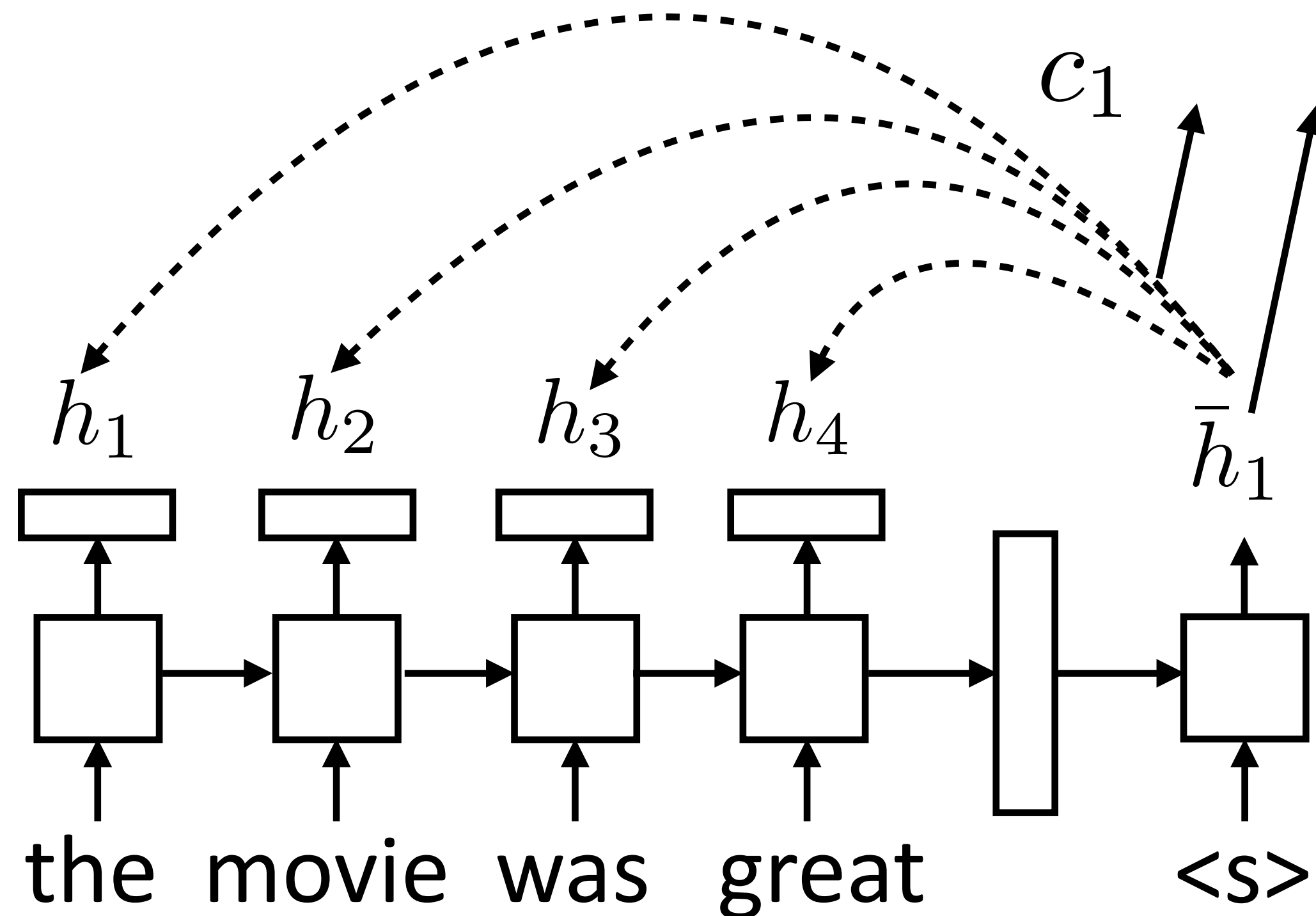
$$e_{ij} = f(\bar{h}_i, h_j)$$



- Unnormalized scalar weight

# Attention

- For each decoder state, compute weighted sum of input states

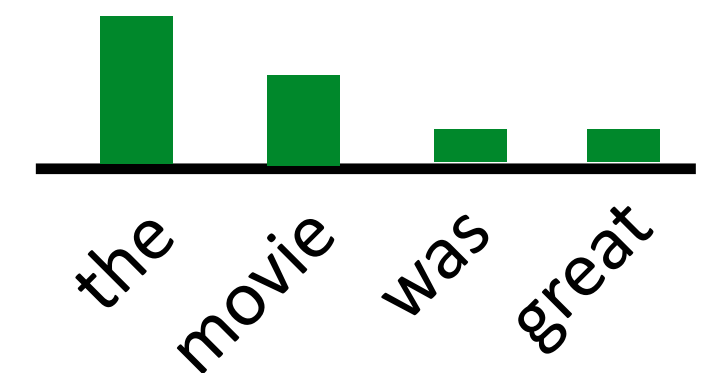


$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

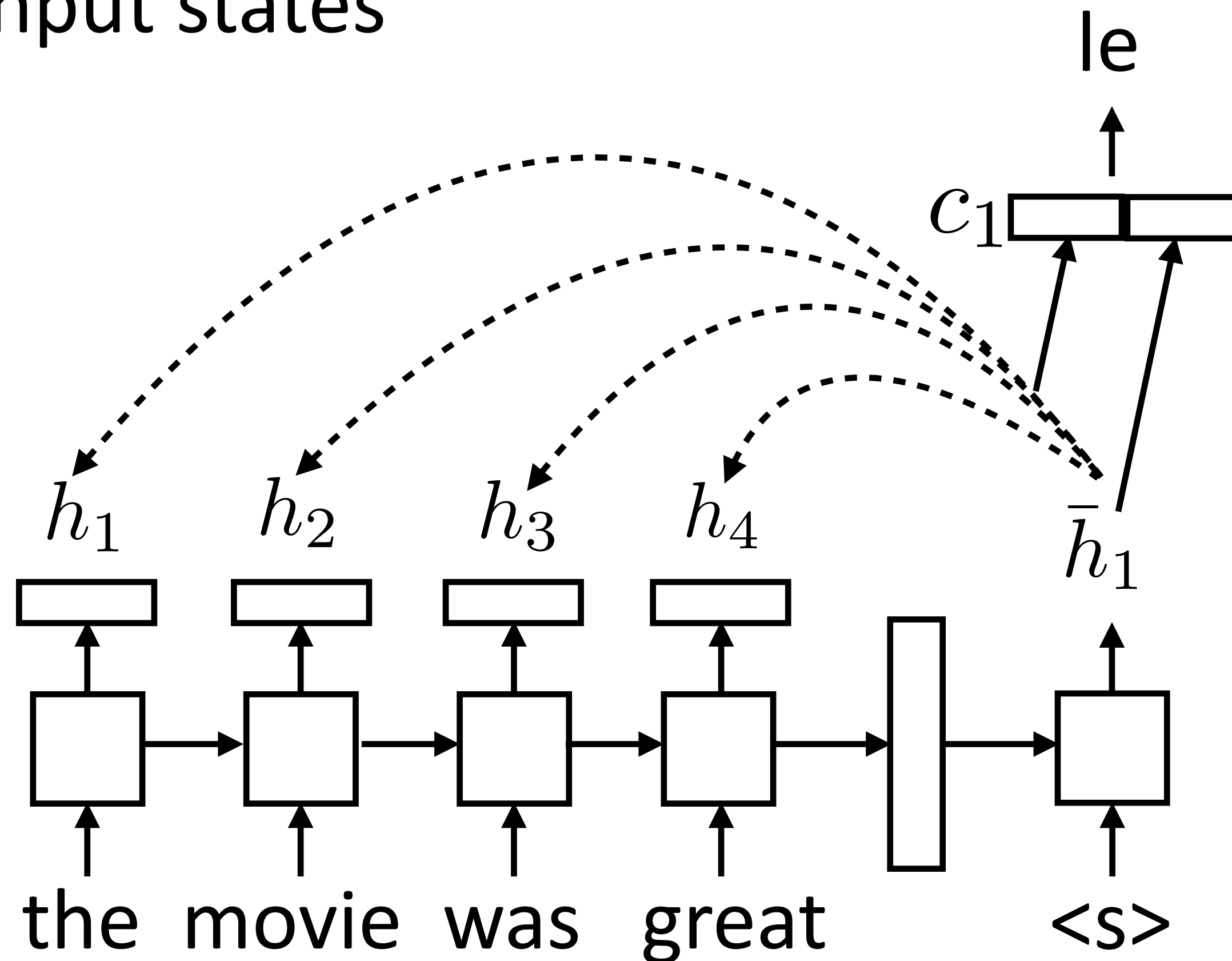
- Weighted sum of input hidden states (vector)



- Unnormalized scalar weight

# Attention

- For each decoder state, compute weighted sum of input states

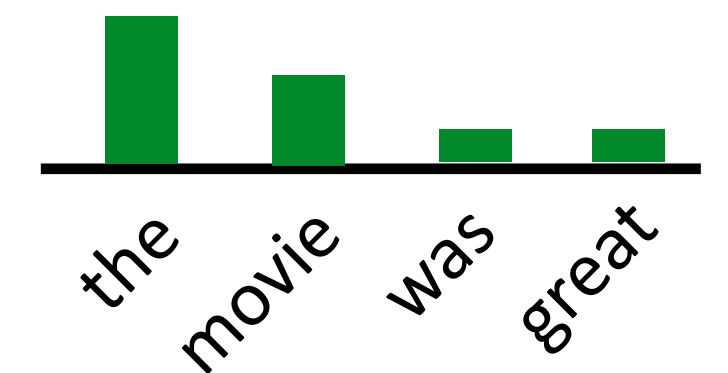


$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

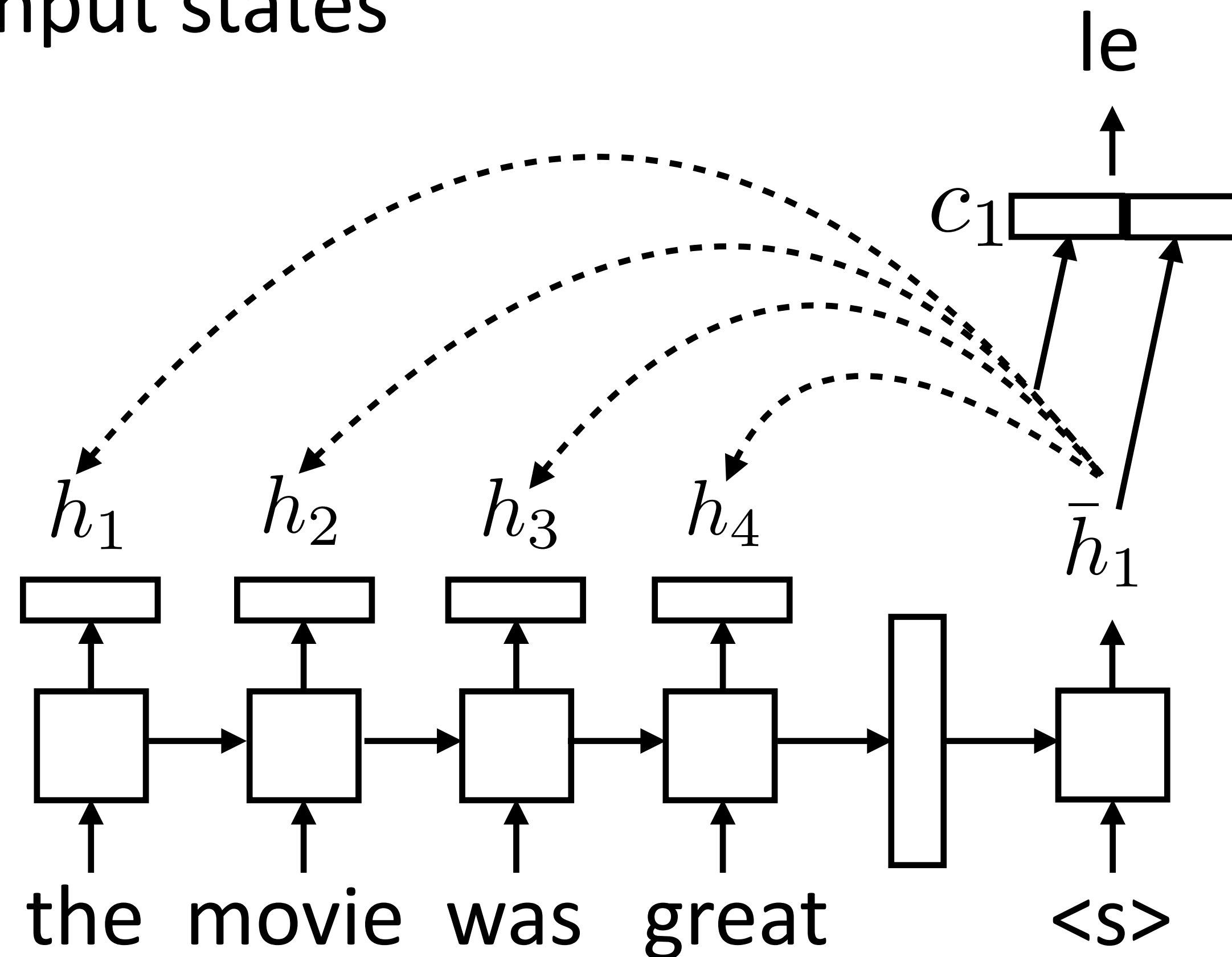
- Weighted sum of input hidden states (vector)



- Unnormalized scalar weight

# Attention

- For each decoder state, compute weighted sum of input states



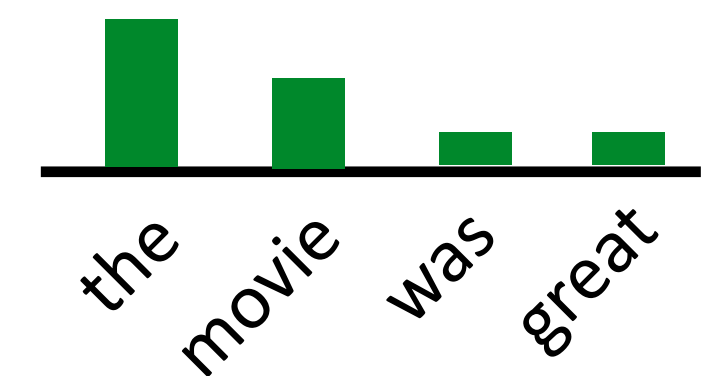
$$P(y_i | \mathbf{x}, y_1, \dots, y_{i-1}) = \text{softmax}(W[c_i; \bar{h}_i])$$

$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

- Weighted sum of input hidden states (vector)



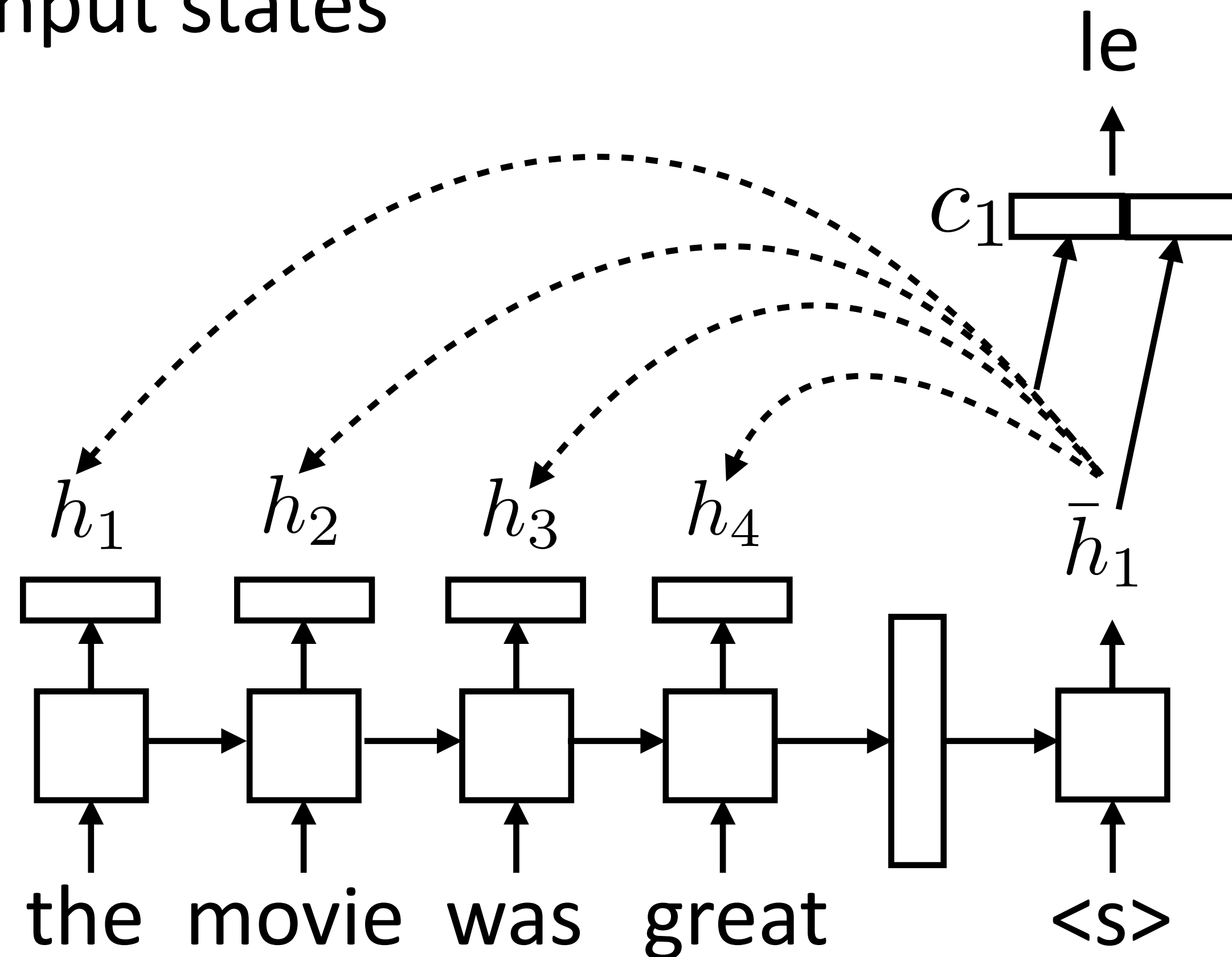
- Unnormalized scalar weight



# Attention

- For each decoder state, compute weighted sum of input states

- No attn:  $P(y_i | \mathbf{x}, y_1, \dots, y_{i-1}) = \text{softmax}(W \bar{h}_i)$



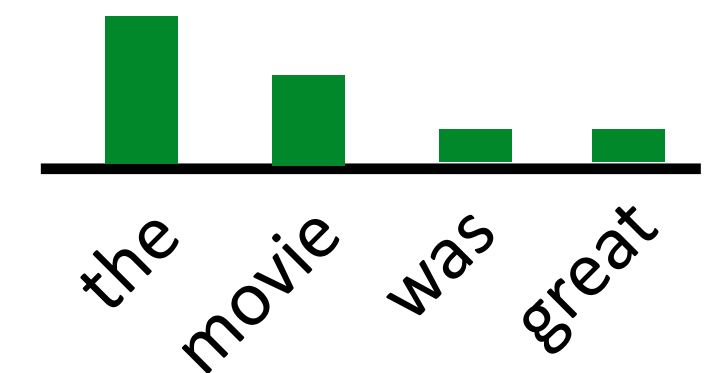
$$P(y_i | \mathbf{x}, y_1, \dots, y_{i-1}) = \text{softmax}(W[c_i; \bar{h}_i])$$

$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

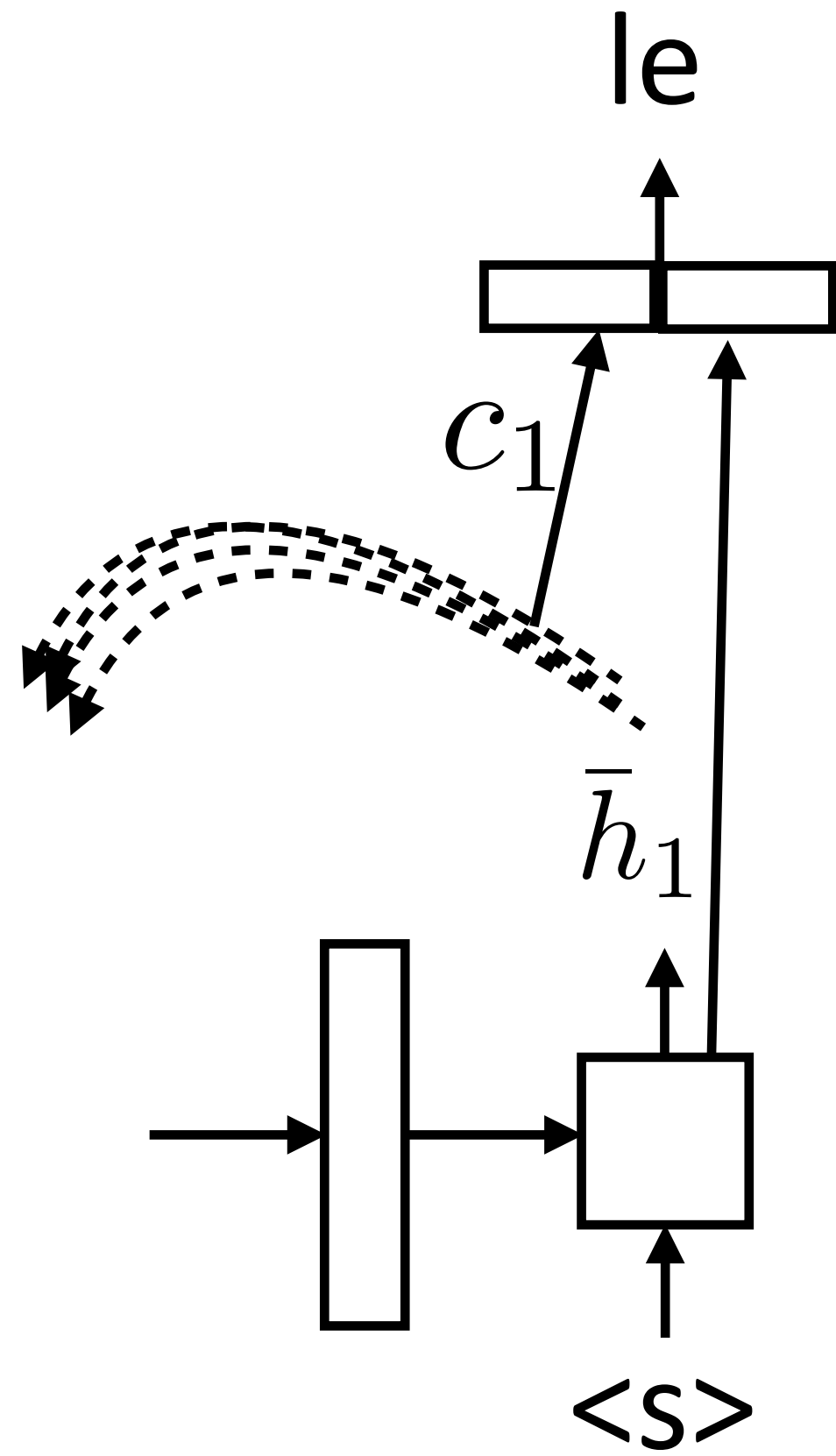
$$e_{ij} = f(\bar{h}_i, h_j)$$

- Weighted sum of input hidden states (vector)



- Unnormalized scalar weight

# Attention

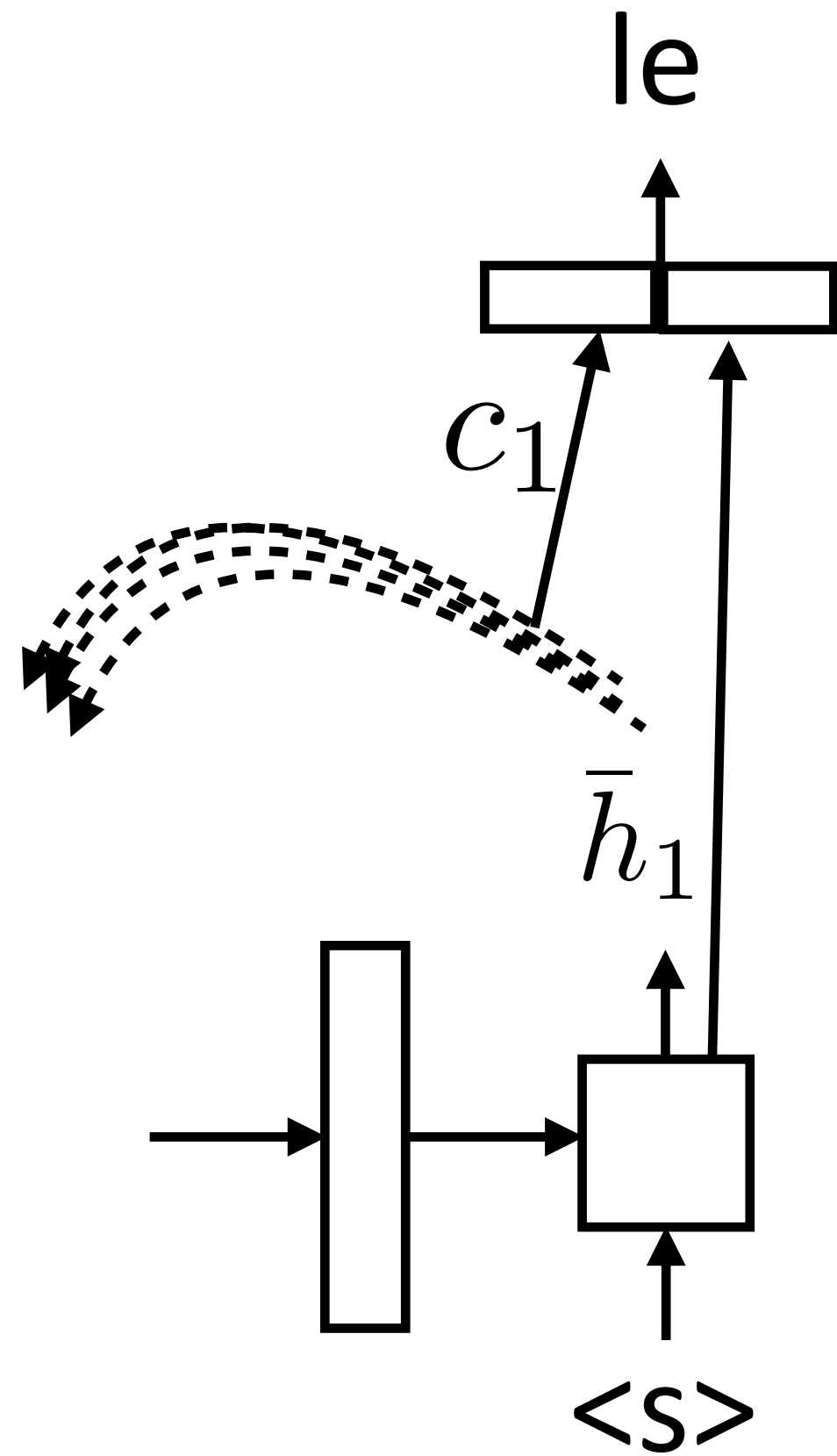


$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

# Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

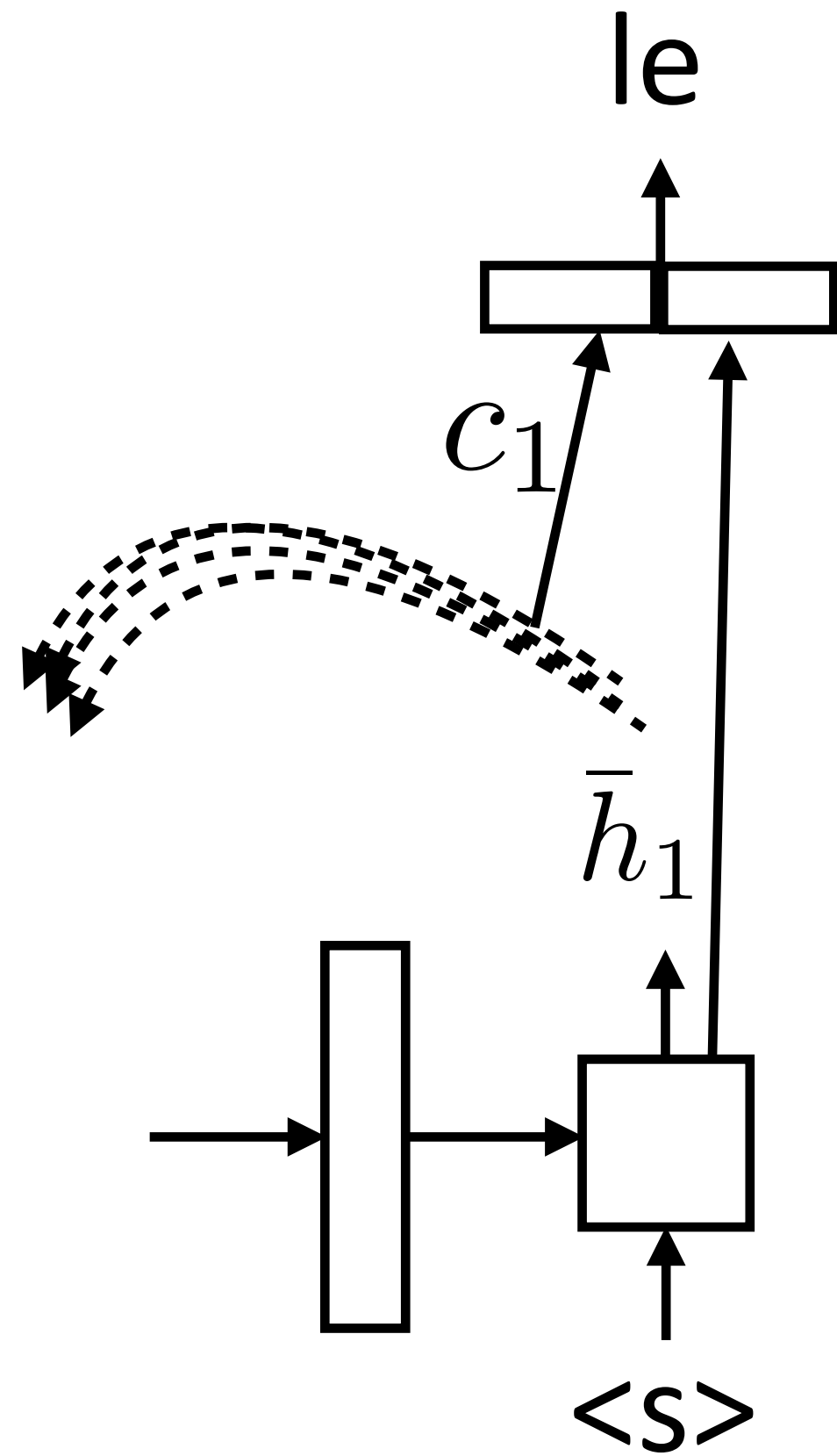
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

$$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$$

► Bahdanau+ (2014): additive

# Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

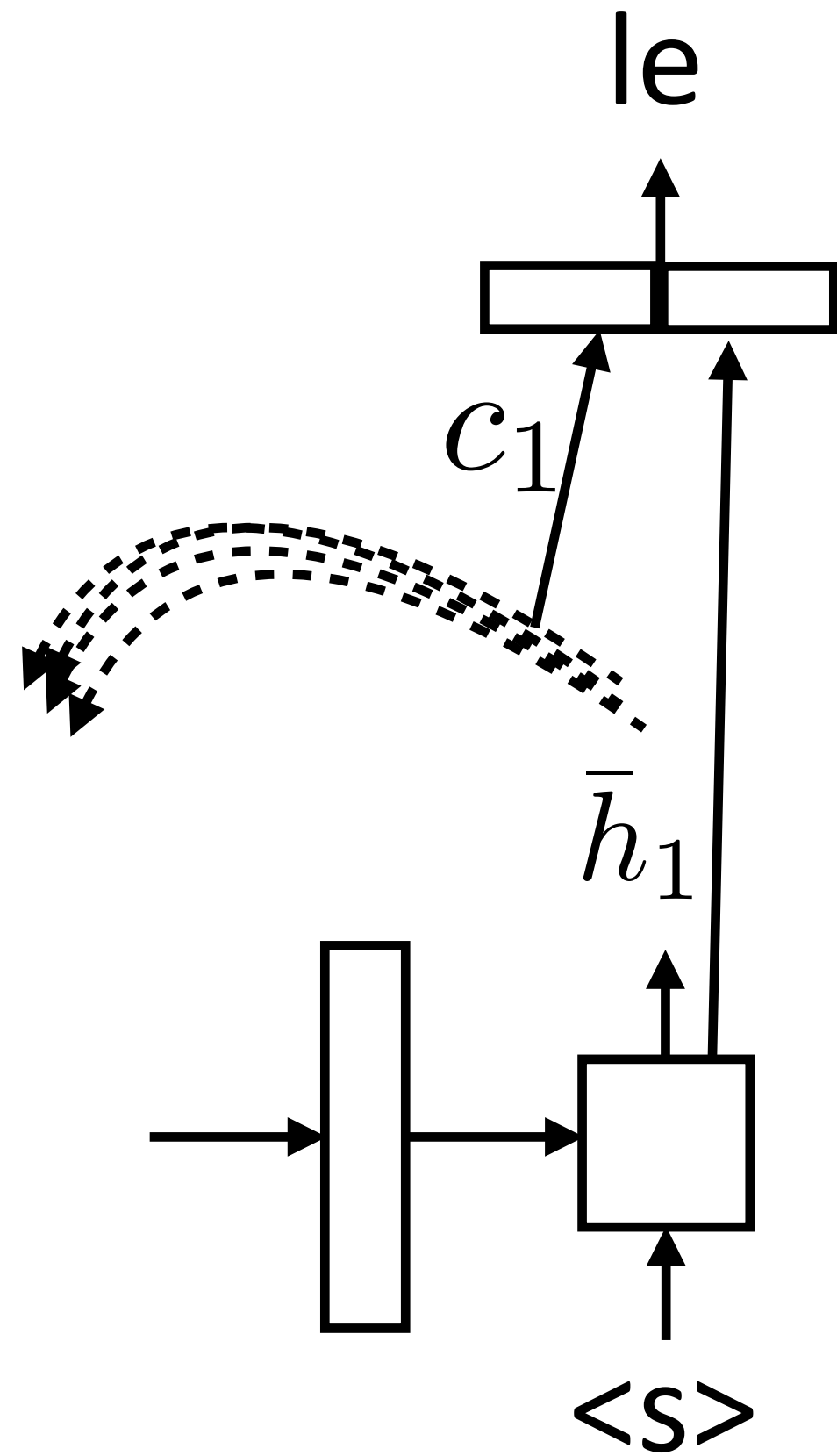
$$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$$

► Bahdanau+ (2014): additive

$$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$$

► Luong+ (2015): dot product

# Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

$$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$$

► Bahdanau+ (2014): additive

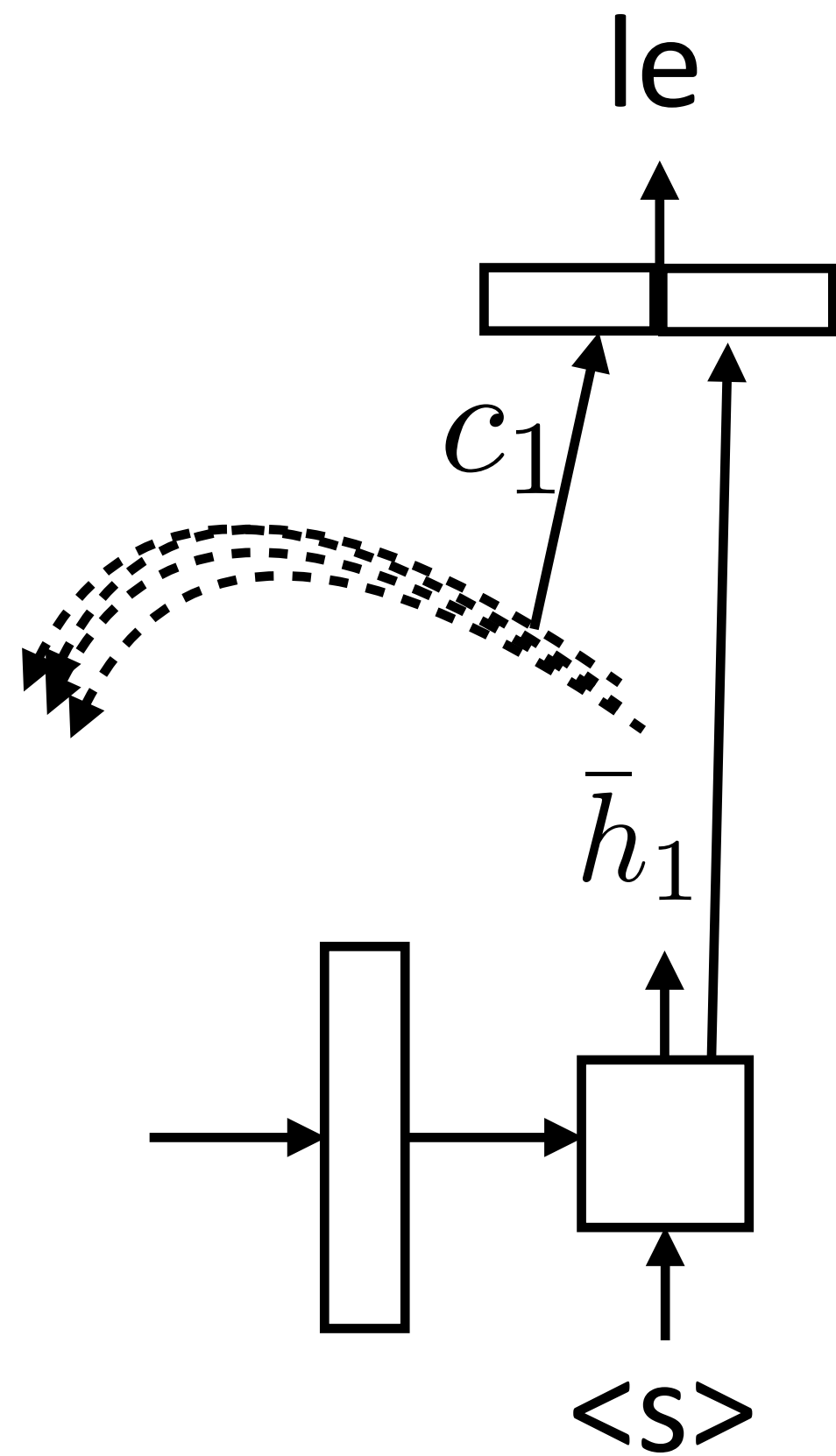
$$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$$

► Luong+ (2015): dot product

$$f(\bar{h}_i, h_j) = \bar{h}_i^\top W h_j$$

► Luong+ (2015): bilinear

# Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

$$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$$

► Bahdanau+ (2014): additive

$$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$$

► Luong+ (2015): dot product

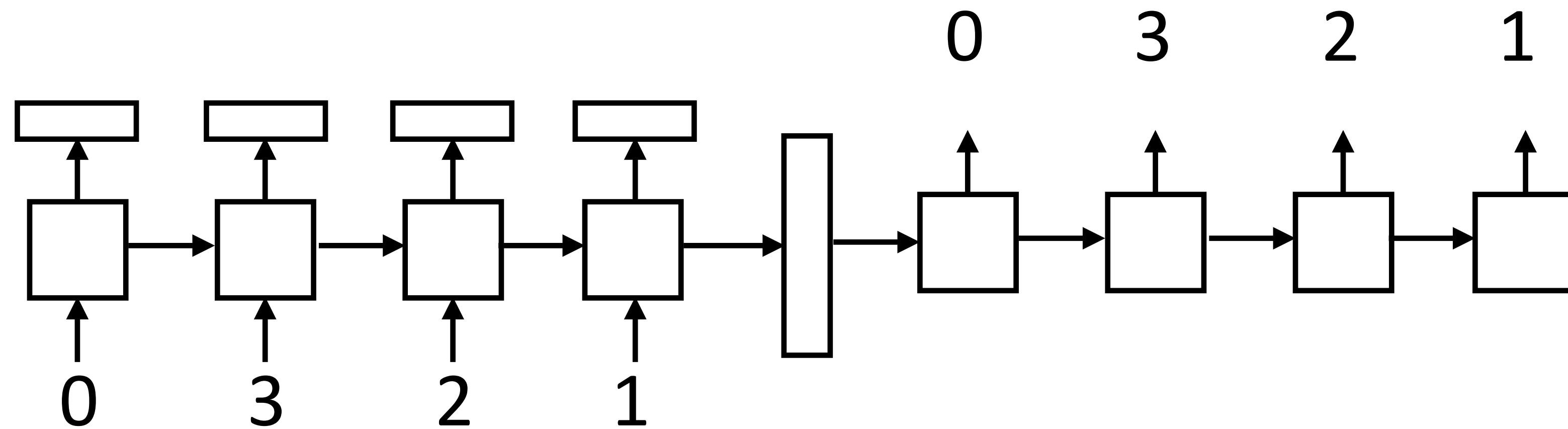
$$f(\bar{h}_i, h_j) = \bar{h}_i^\top W h_j$$

► Luong+ (2015): bilinear

- Note that this all uses outputs of hidden layers

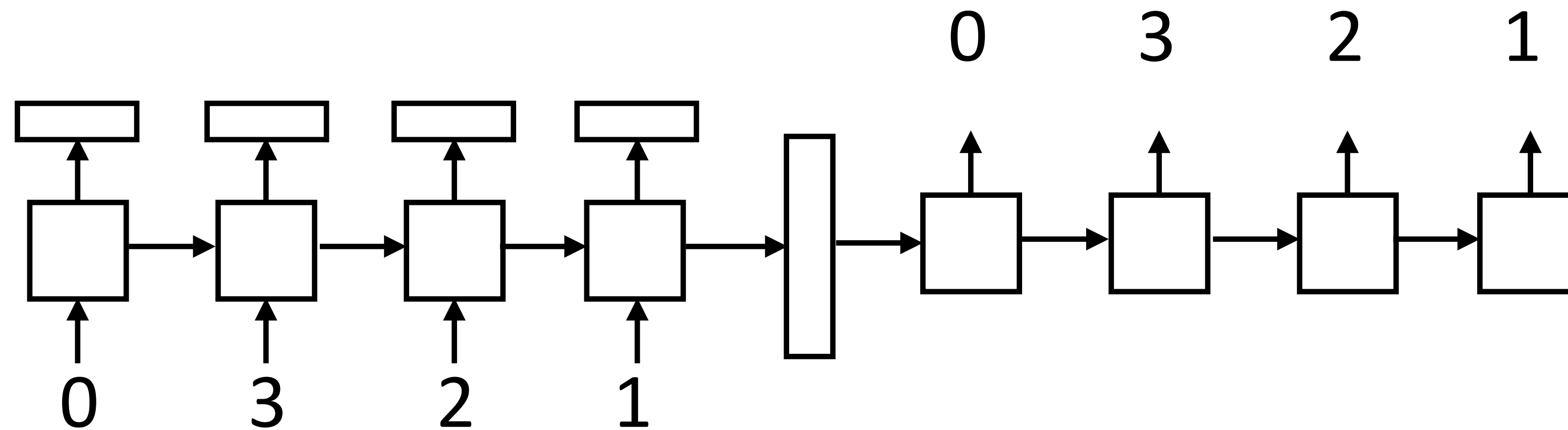
# What can attention do?

---



# What can attention do?

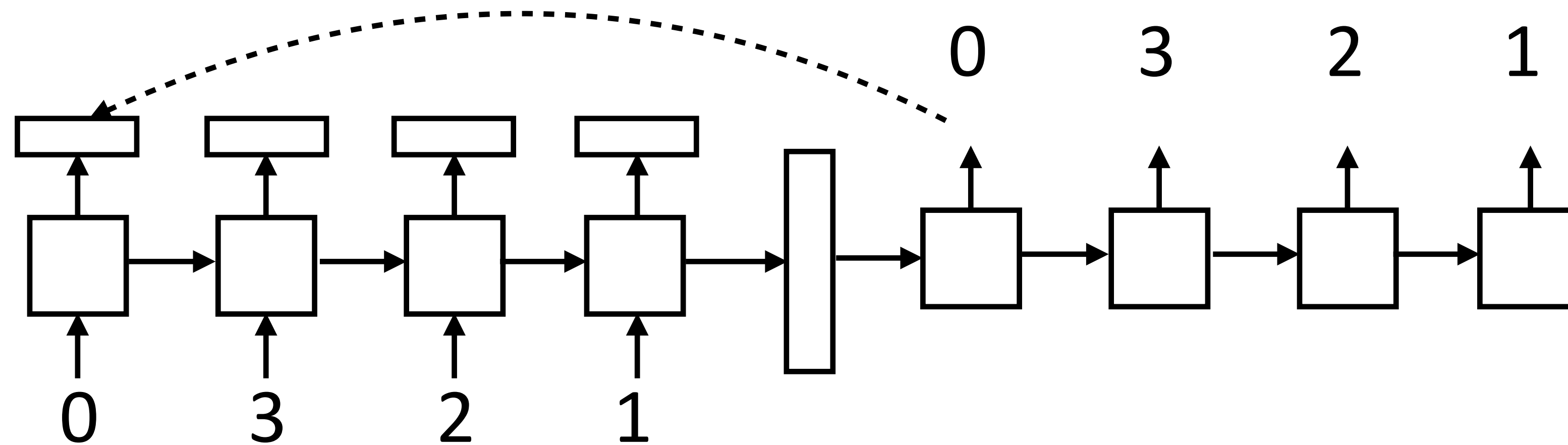
- ▶ Learning to copy — how might this work?





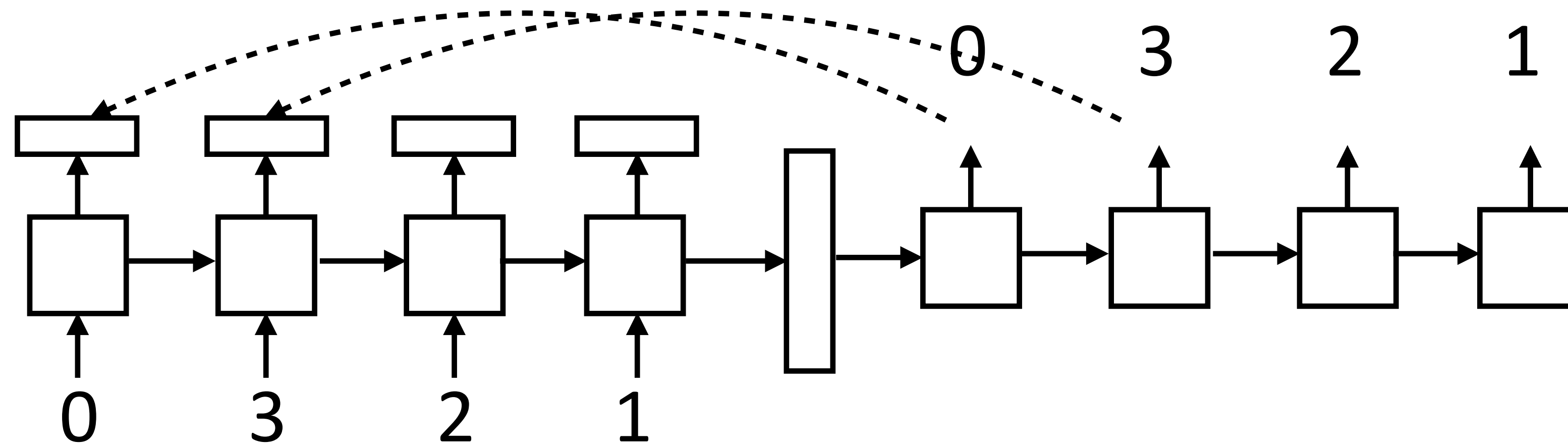
# What can attention do?

- ▶ Learning to copy — how might this work?



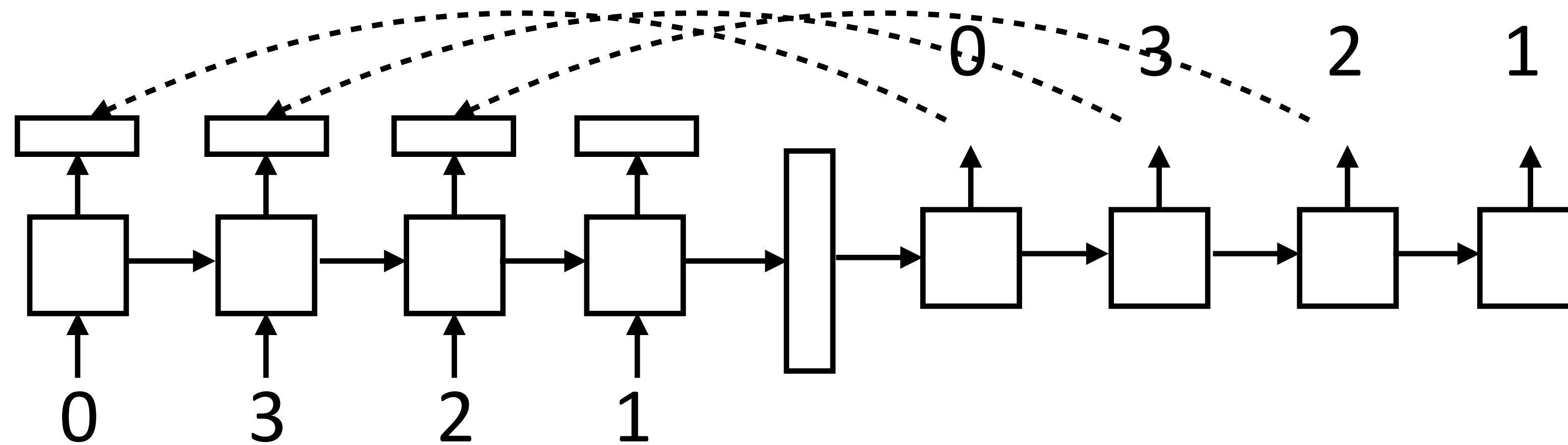
# What can attention do?

- ▶ Learning to copy — how might this work?



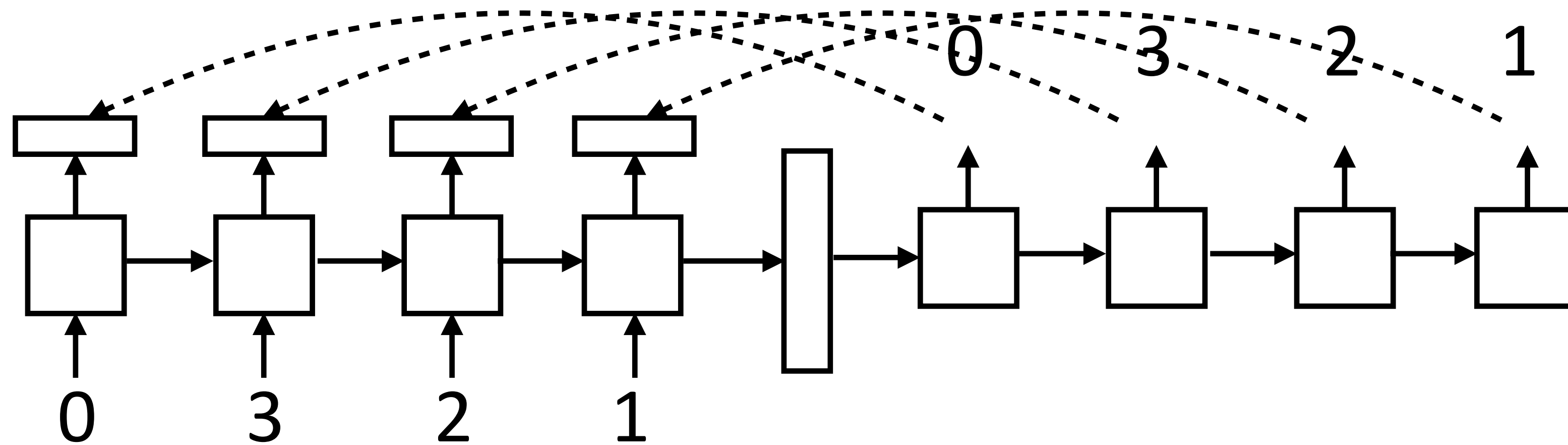
# What can attention do?

- ▶ Learning to copy — how might this work?



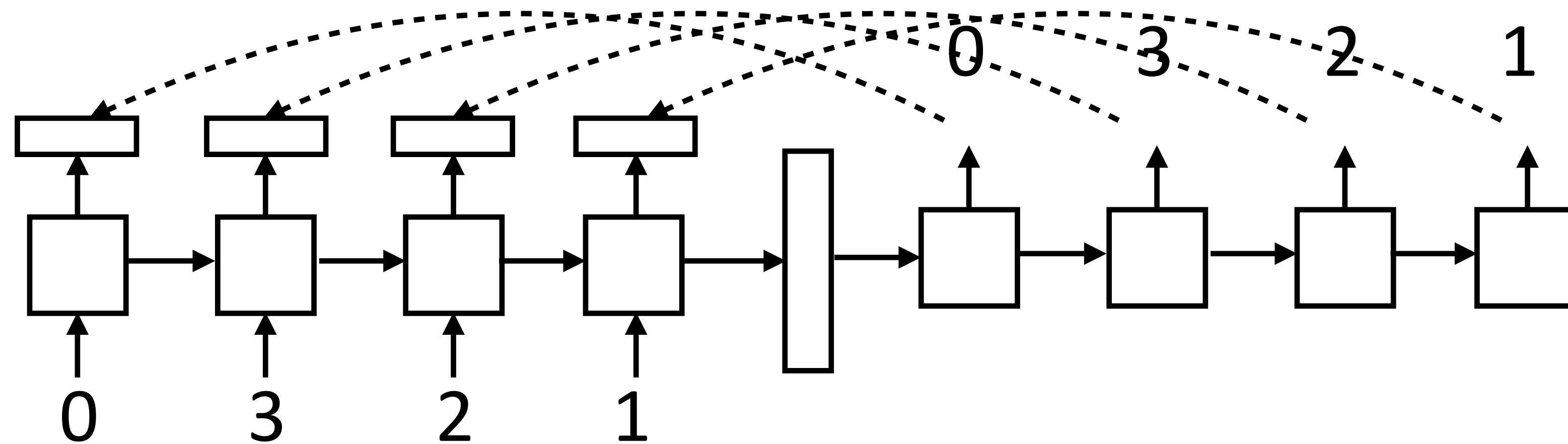
# What can attention do?

- ▶ Learning to copy — how might this work?



# What can attention do?

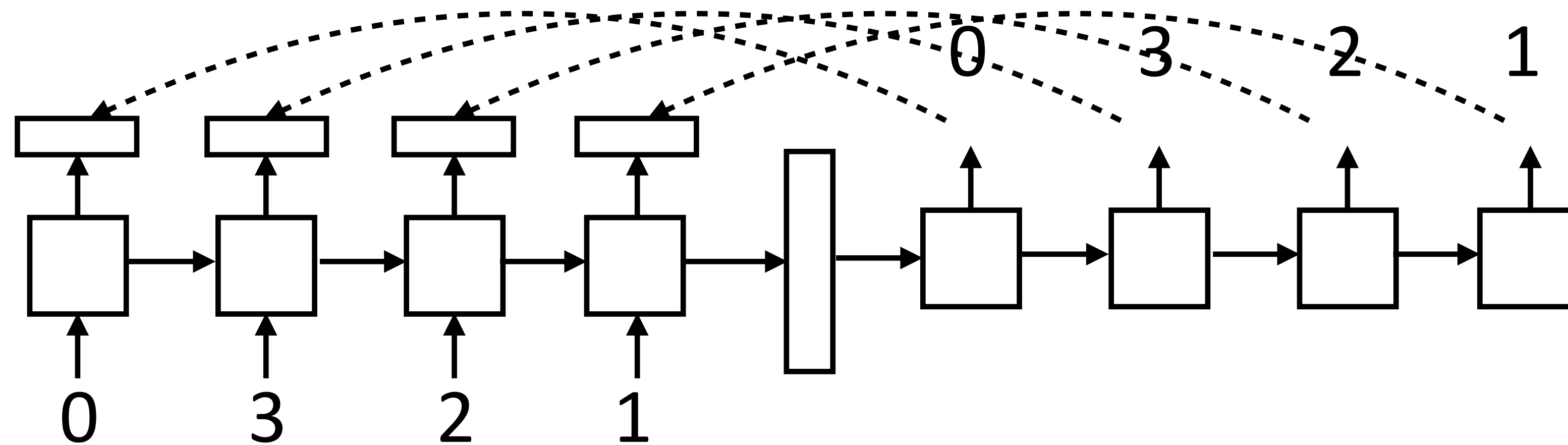
- ▶ Learning to copy — how might this work?



- ▶ LSTM can learn to count with the right weight matrix

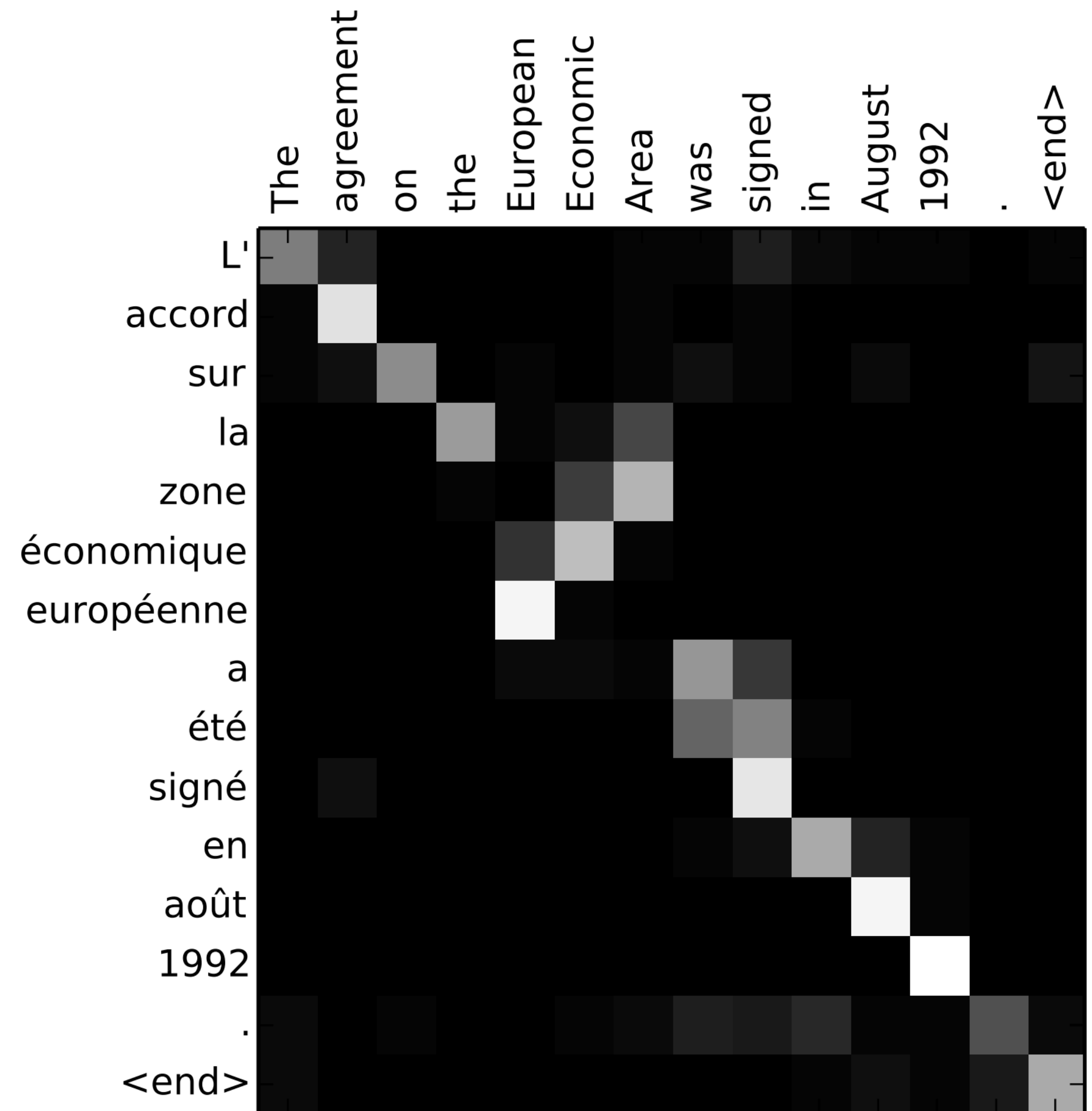
# What can attention do?

- ▶ Learning to copy — how might this work?



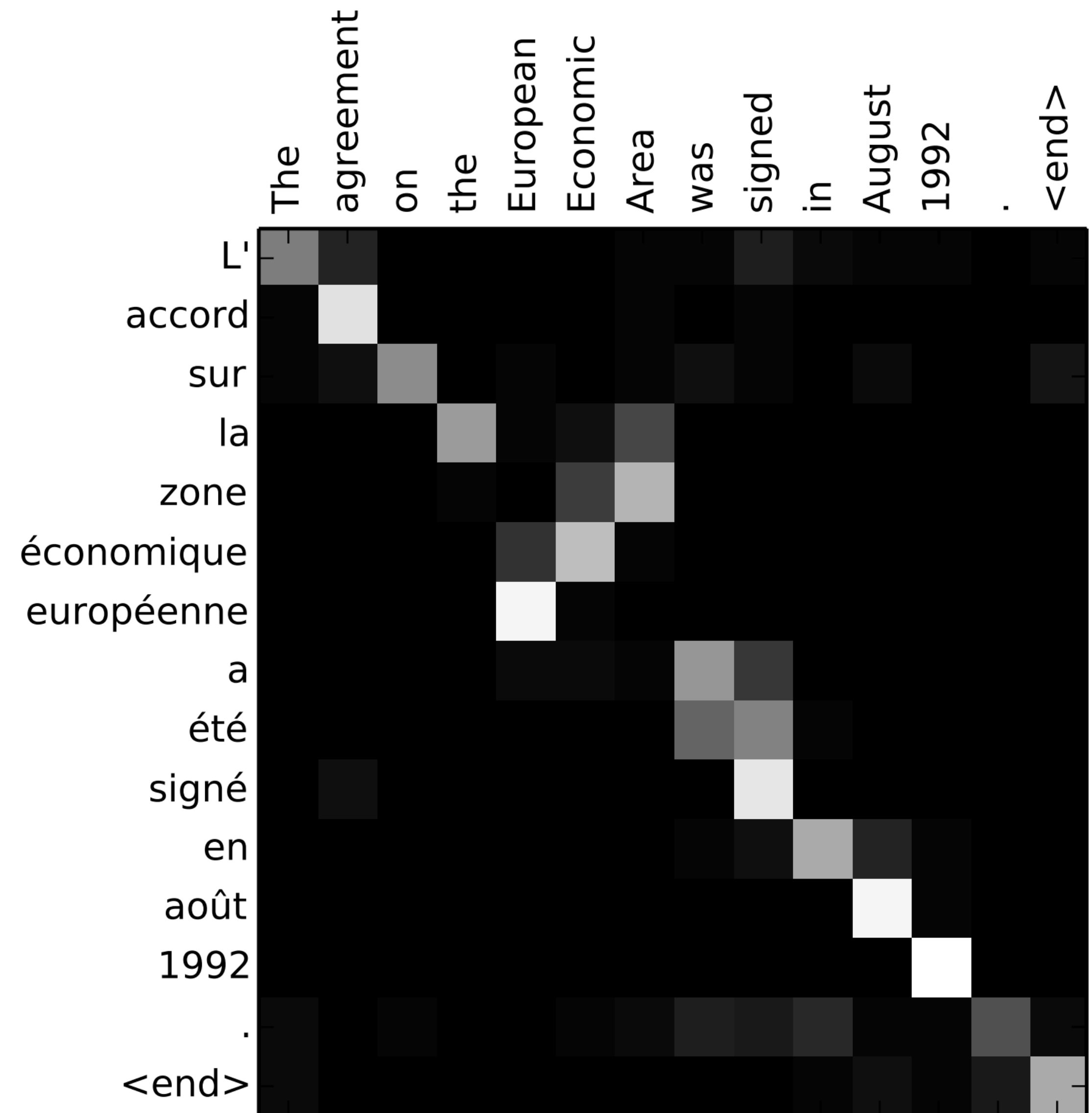
- ▶ LSTM can learn to count with the right weight matrix
- ▶ This is effectively position-based addressing

# Attention



# Attention

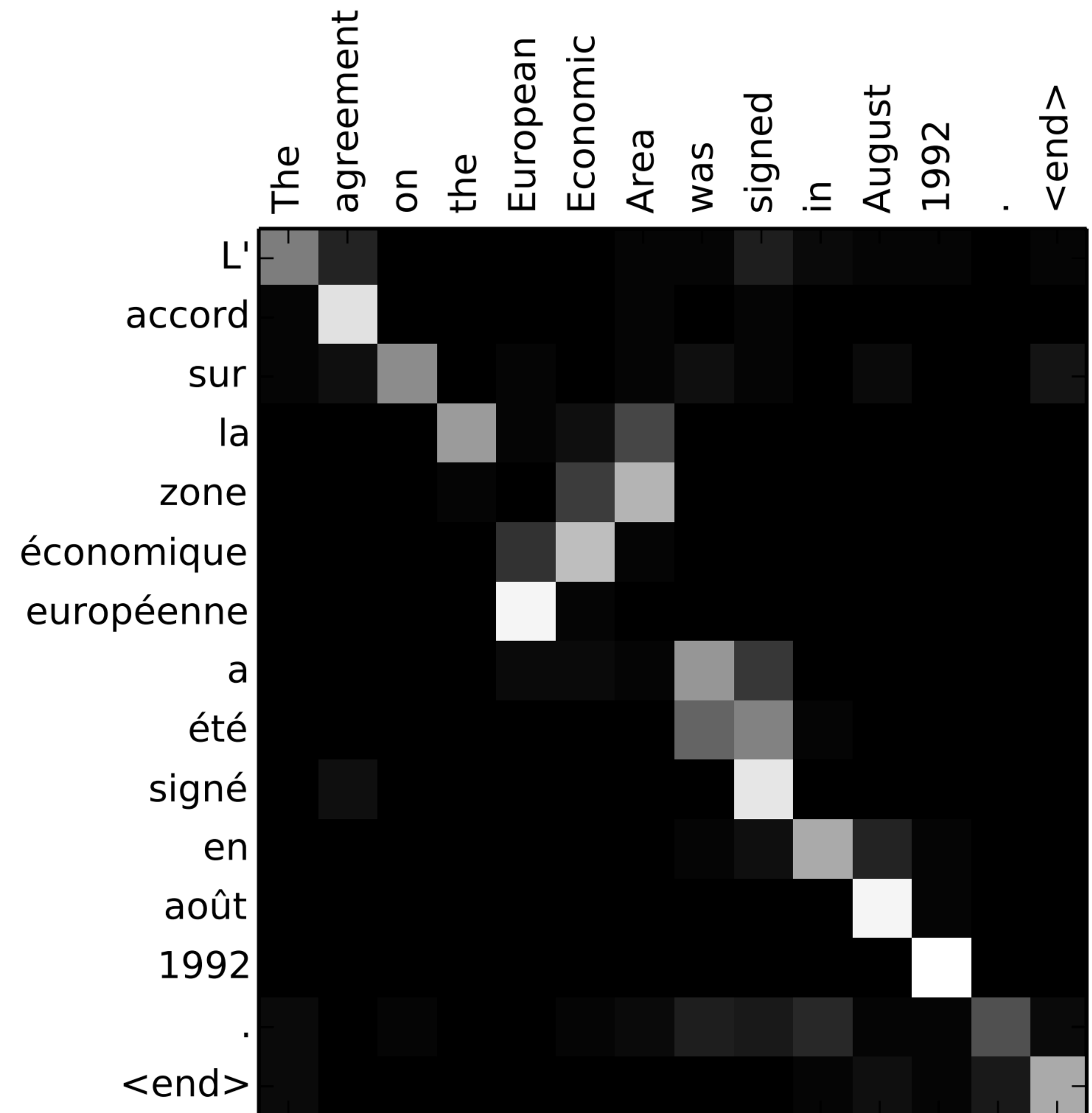
- ▶ Encoder hidden states capture contextual source word identity





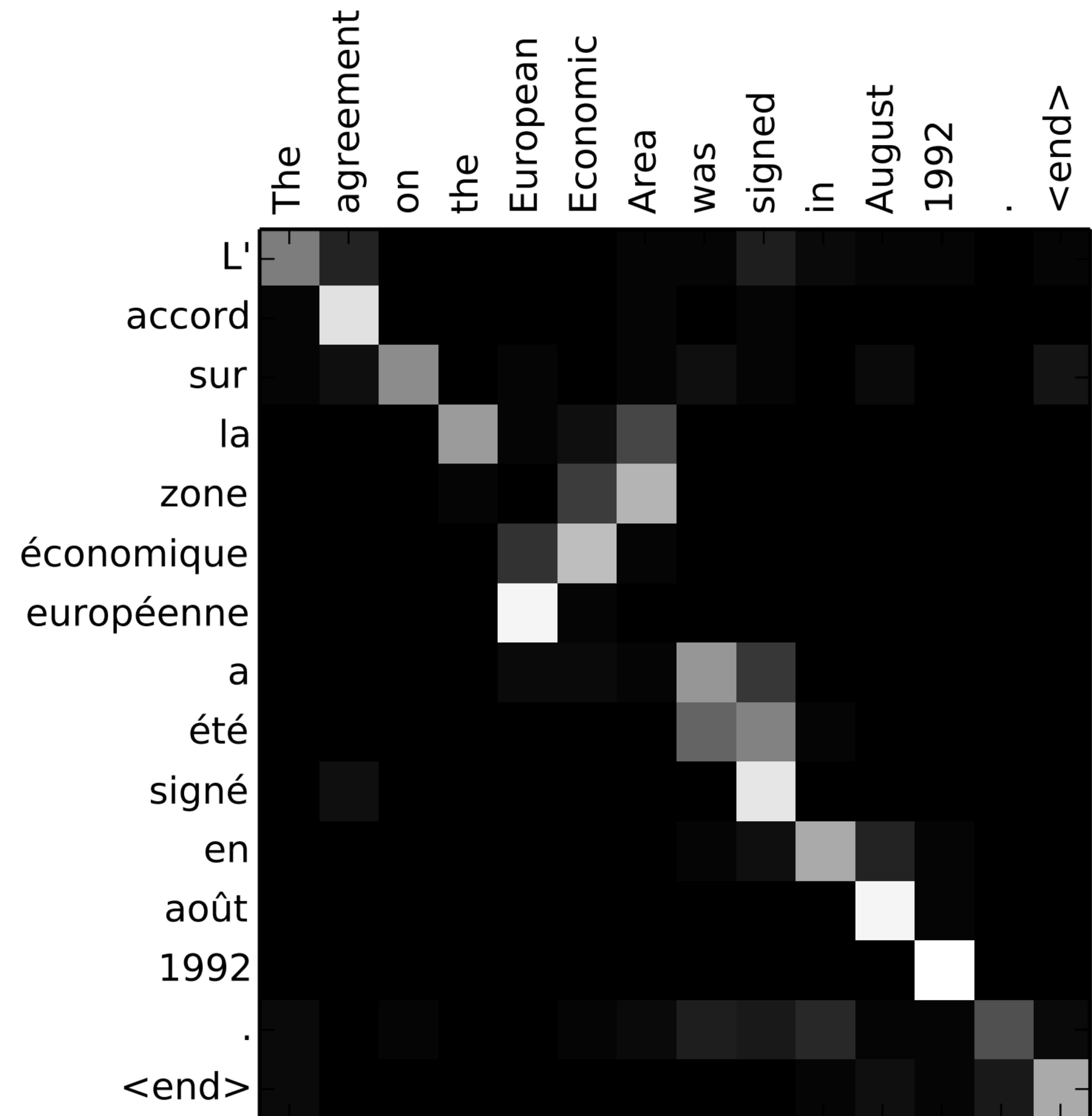
# Attention

- ▶ Encoder hidden states capture contextual source word identity
- ▶ Decoder hidden states are now mostly responsible for selecting what to attend to



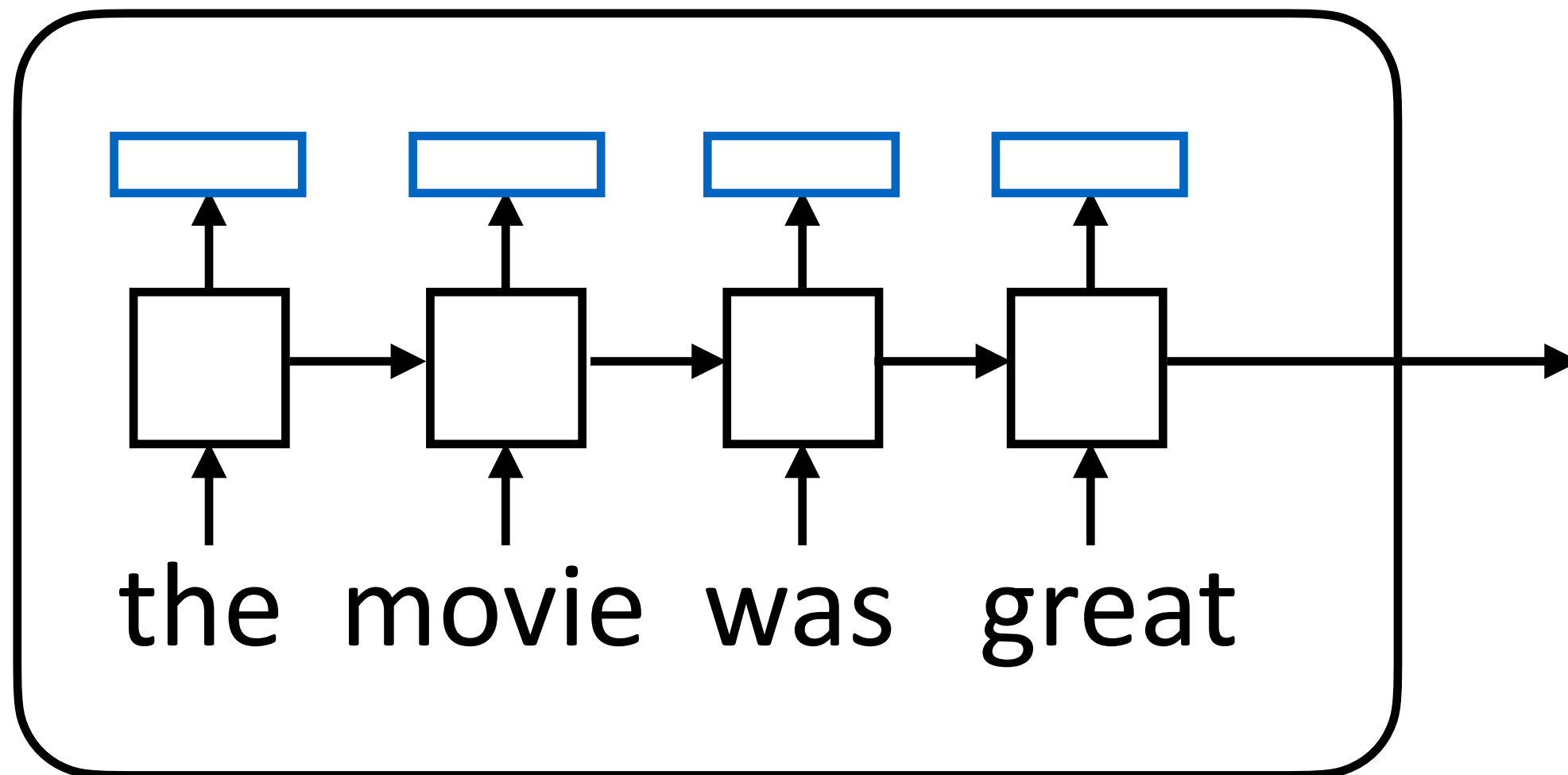
# Attention

- ▶ Encoder hidden states capture contextual source word identity
- ▶ Decoder hidden states are now mostly responsible for selecting what to attend to
- ▶ Doesn't take a complex hidden state to walk monotonically through a sentence and spit out word-by-word translations



# Batching Attention

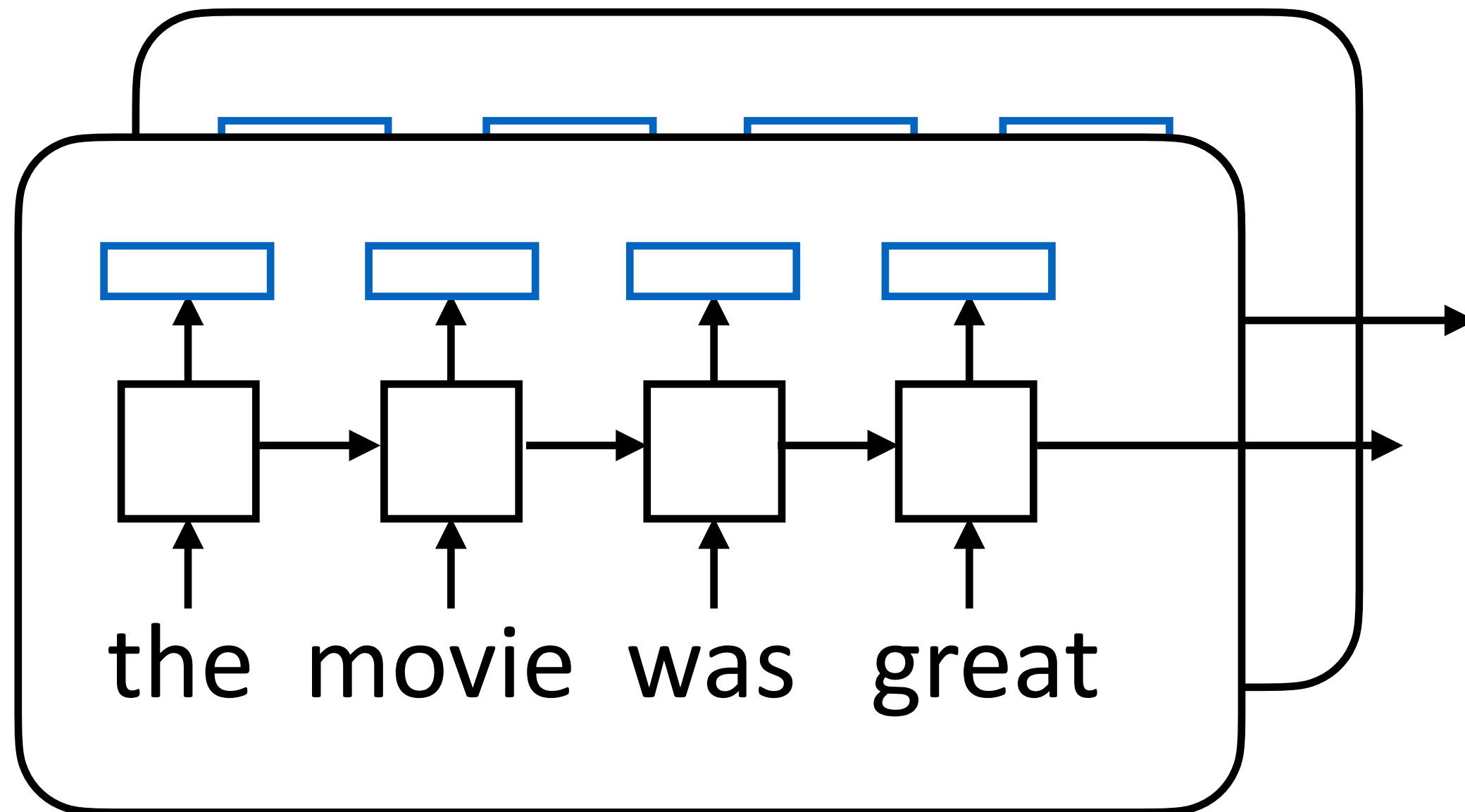
---



# Batching Attention

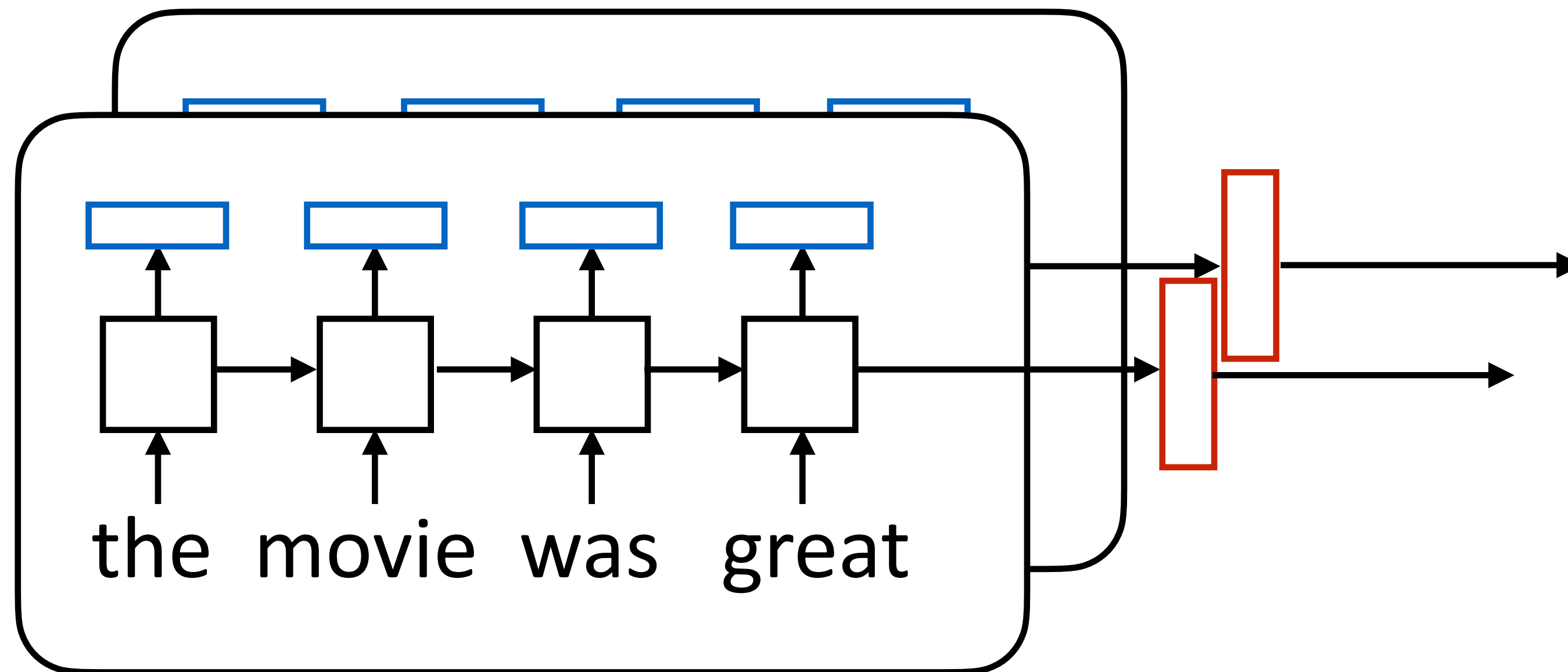
---

token outputs: batch size x sentence length x dimension



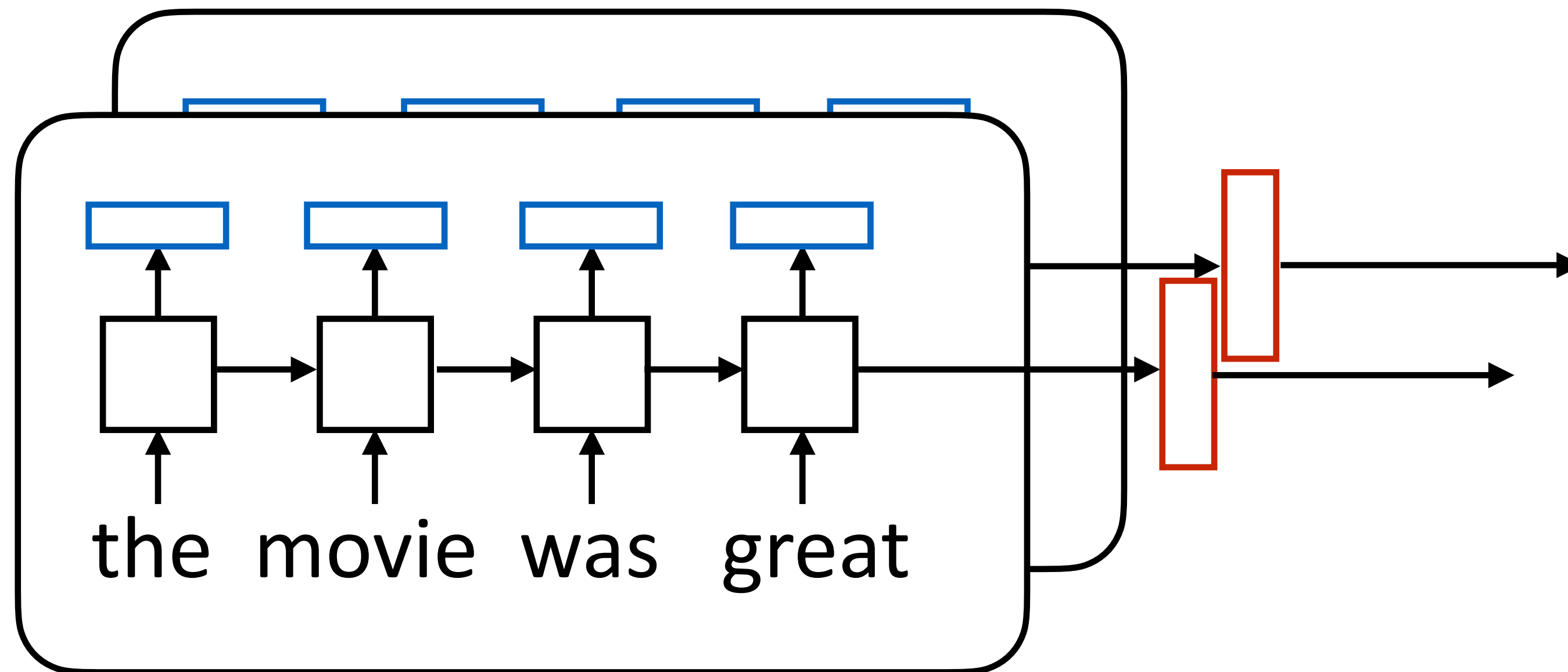
# Batching Attention

token outputs: batch size x sentence length x dimension



# Batching Attention

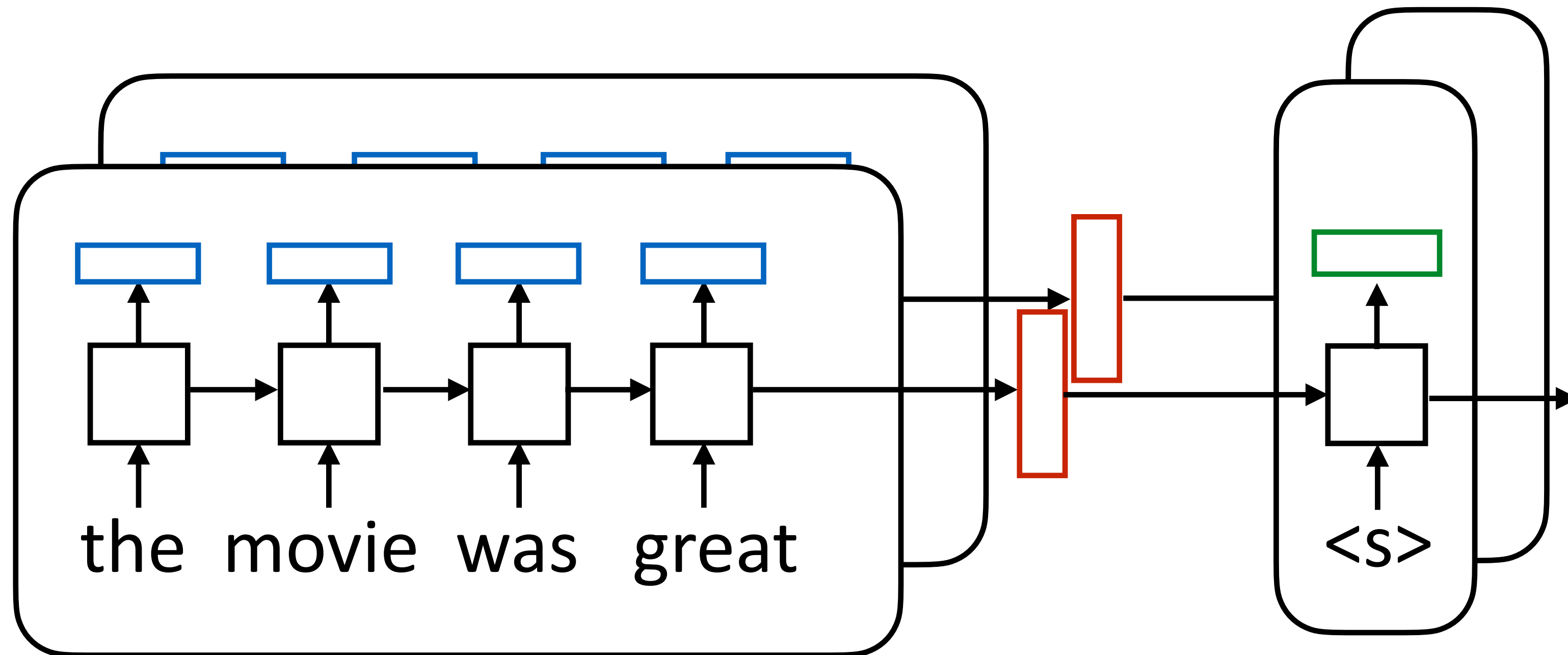
token outputs: batch size x sentence length x dimension



sentence outputs:  
batch size x hidden size

# Batching Attention

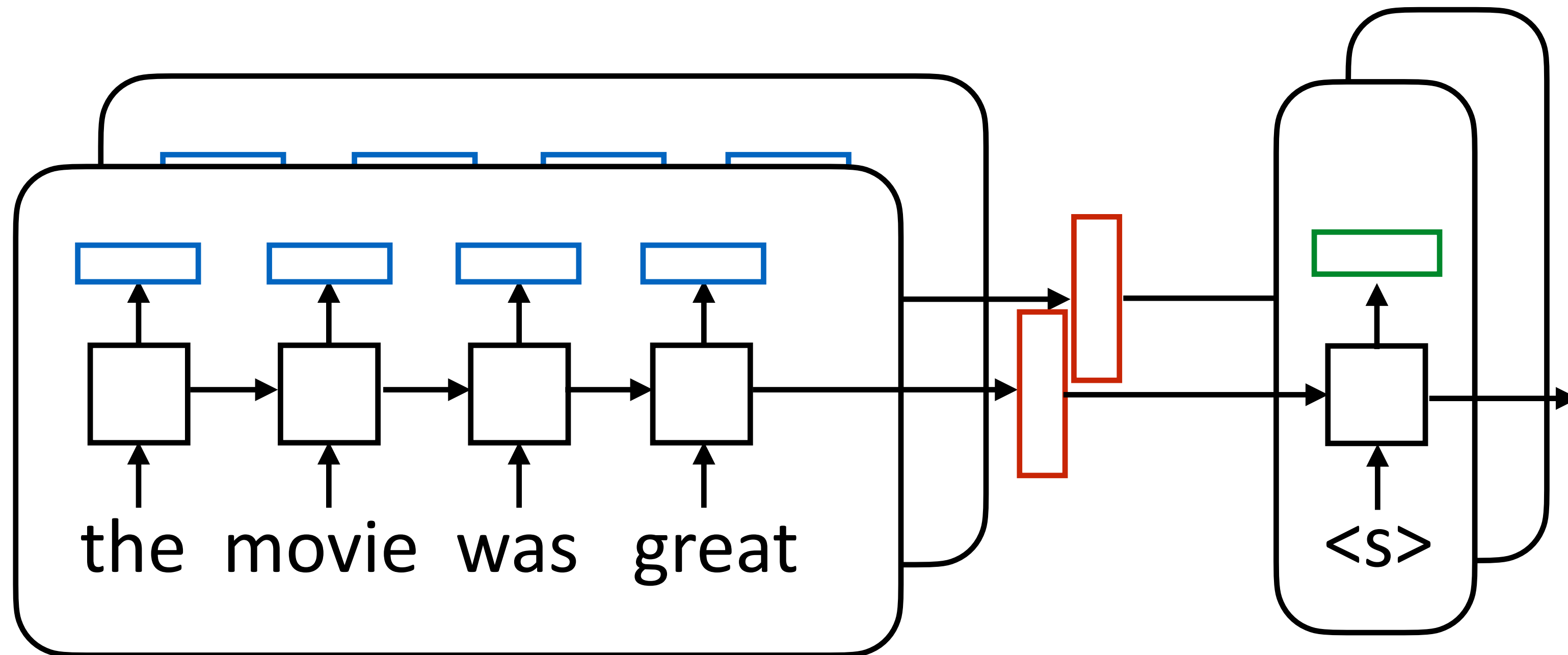
token outputs: batch size x sentence length x dimension



sentence outputs:  
batch size x hidden size

# Batching Attention

token outputs: batch size x sentence length x dimension



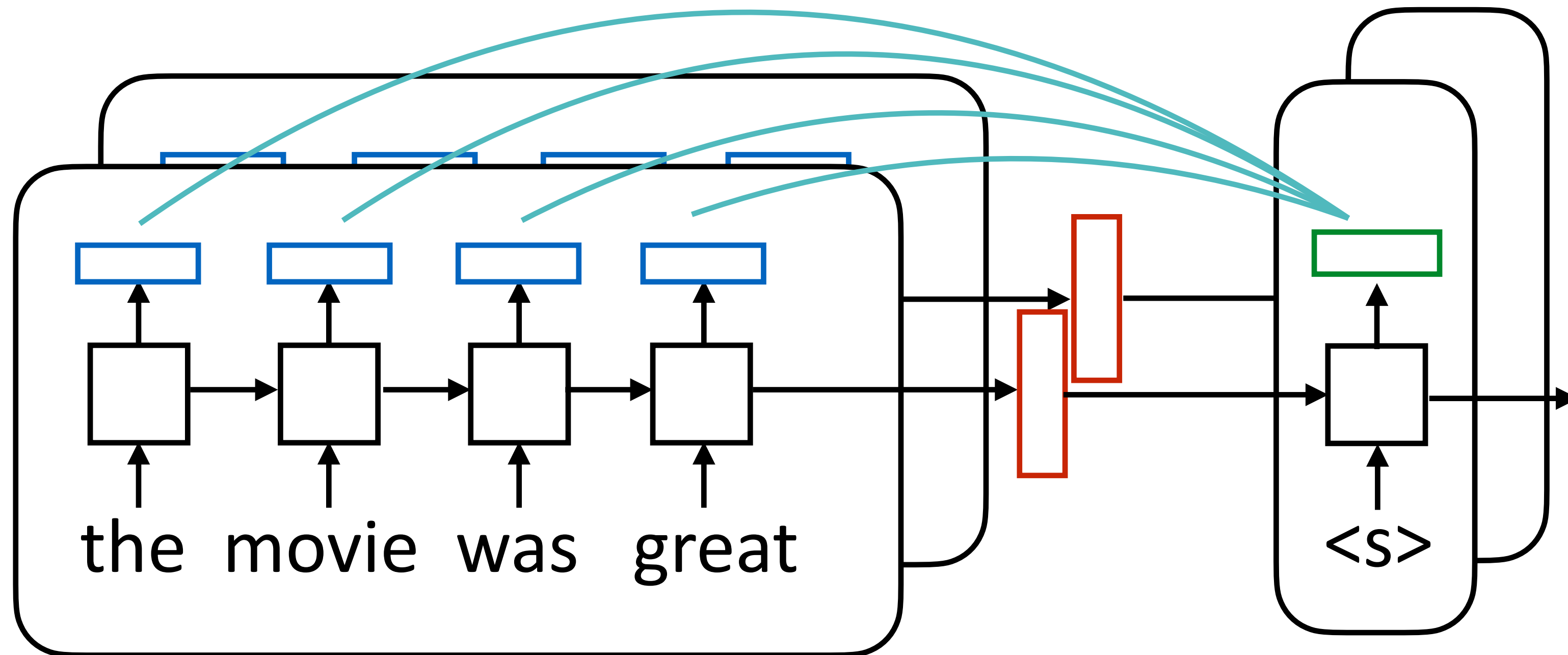
hidden state: batch size  
x hidden size

sentence outputs:  
batch size x hidden size



# Batching Attention

token outputs: batch size x sentence length x dimension

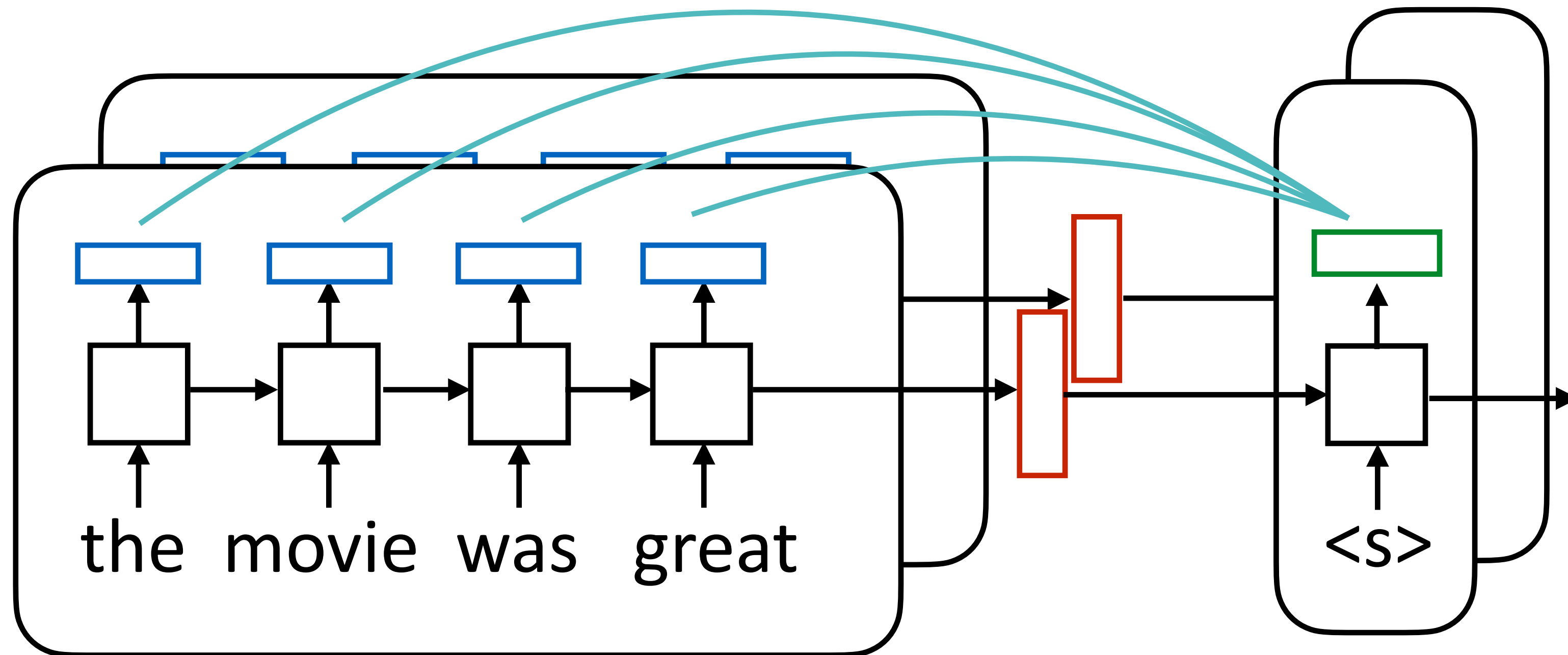


hidden state: batch size  
x hidden size

sentence outputs:  
batch size x hidden size

# Batching Attention

token outputs: batch size x sentence length x dimension



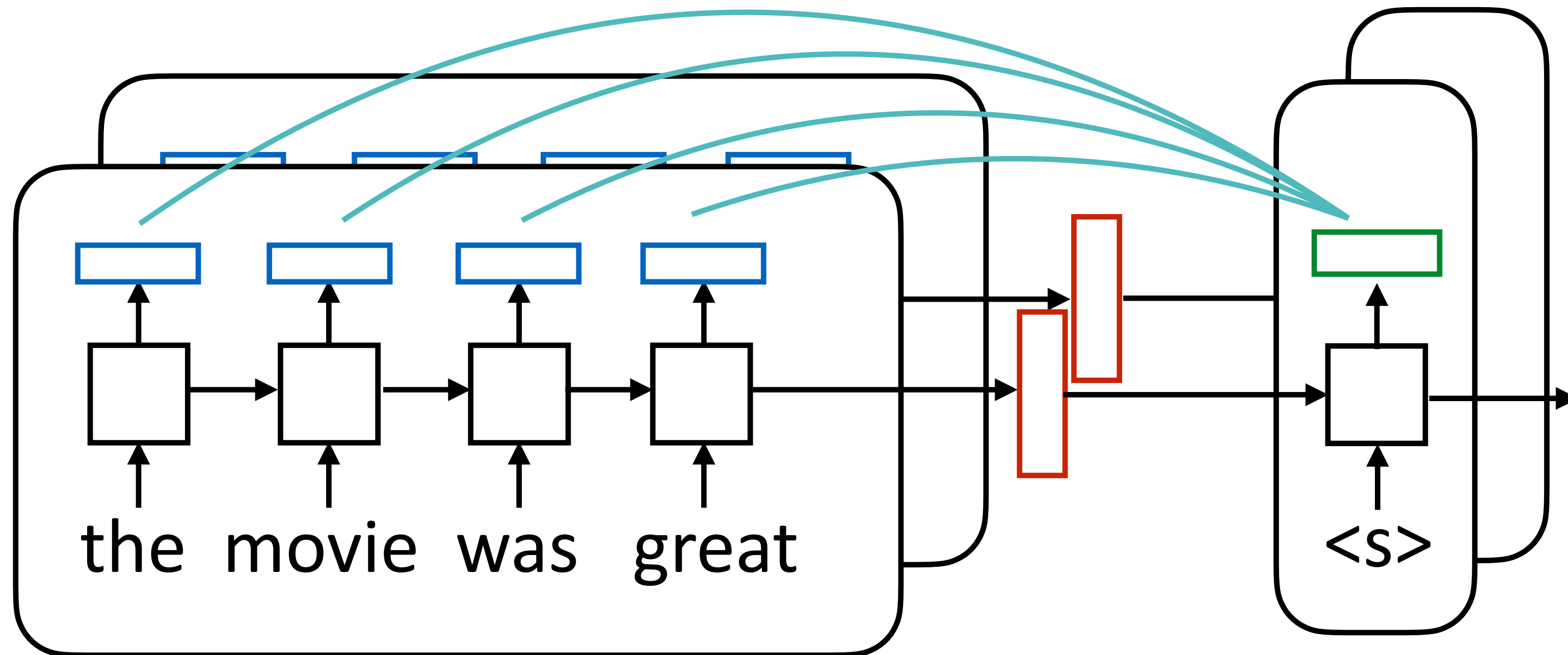
sentence outputs:  
batch size x hidden size

hidden state: batch size  
x hidden size

$$e_{ij} = f(\bar{h}_i, h_j)$$

# Batching Attention

token outputs: batch size x sentence length x dimension



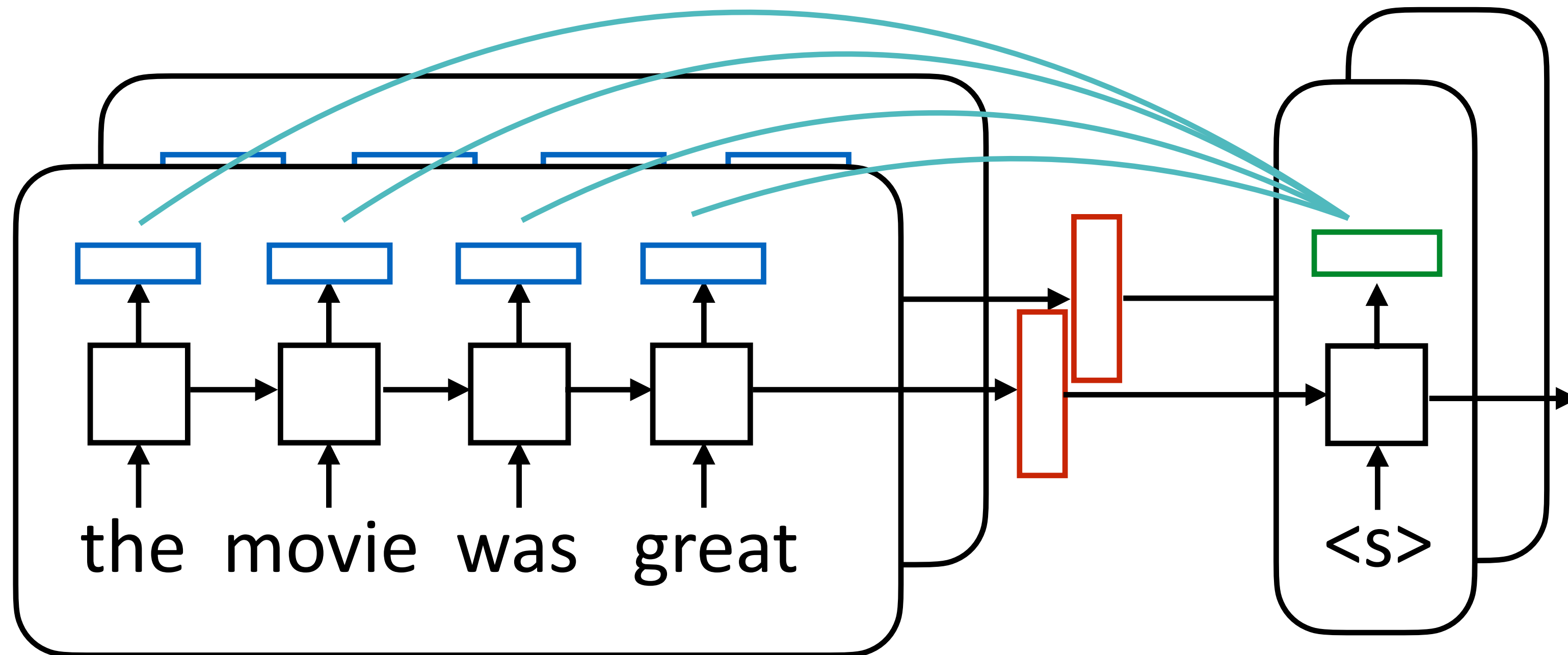
sentence outputs:  
batch size x hidden size

hidden state: batch size  
x hidden size

$$e_{ij} = f(\bar{h}_i, h_j)$$
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

# Batching Attention

token outputs: batch size x sentence length x dimension



sentence outputs:  
batch size x hidden size

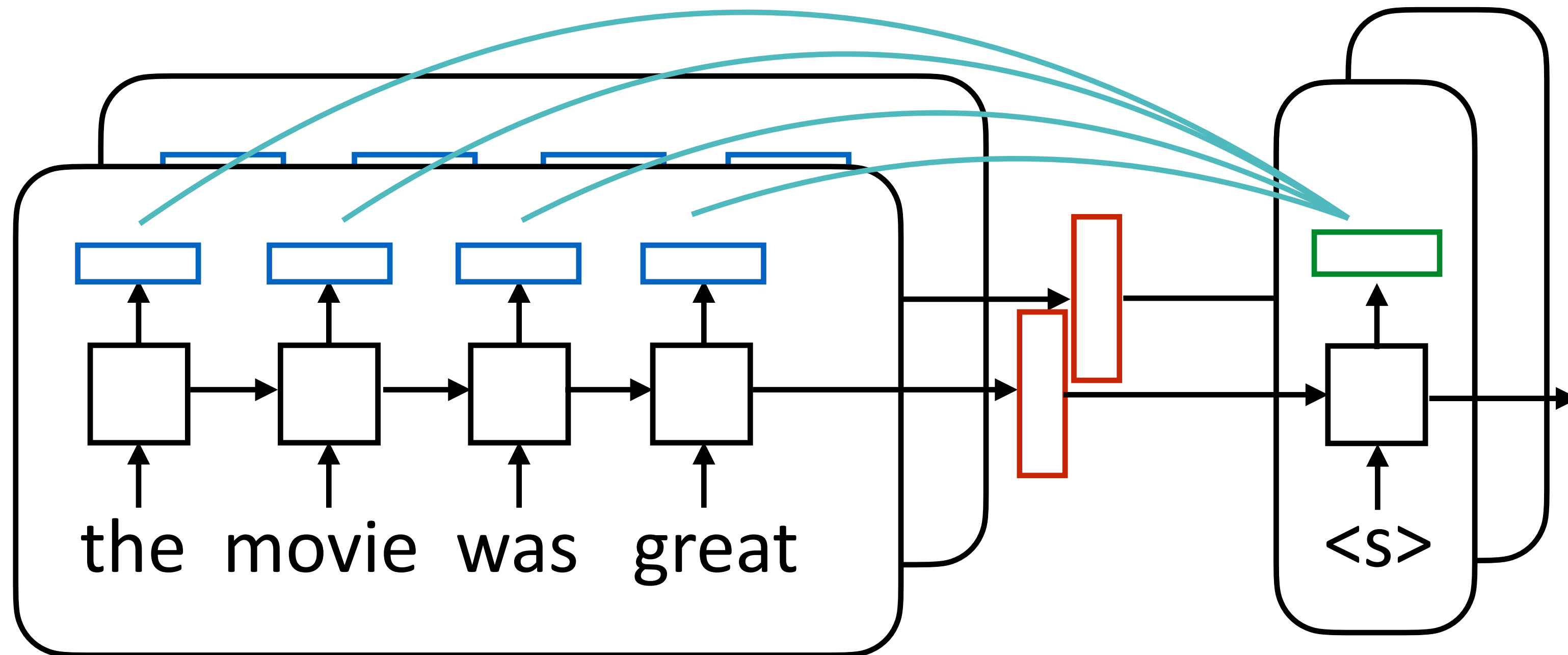
hidden state: batch size  
x hidden size

$$e_{ij} = f(\bar{h}_i, h_j)$$
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

attention scores = batch size x sentence length

# Batching Attention

token outputs: batch size x sentence length x dimension



sentence outputs:  
batch size x hidden size

hidden state: batch size  
x hidden size

$$e_{ij} = f(\bar{h}_i, h_j)$$
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

attention scores = batch size x sentence length

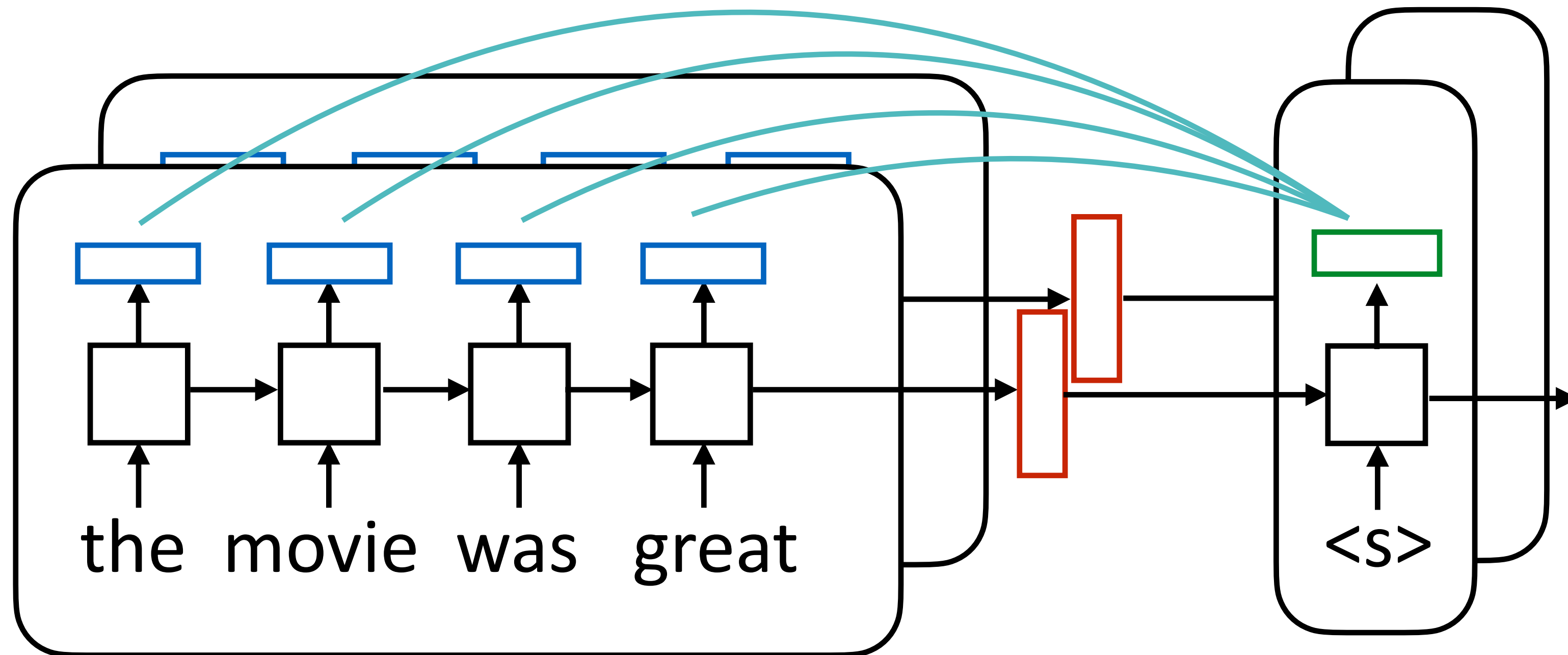
c = batch size x hidden size

$$c_i = \sum_j \alpha_{ij} h_j$$

Luong et al. (2015)

# Batching Attention

token outputs: batch size x sentence length x dimension



sentence outputs:  
batch size x hidden size

attention scores = batch size x sentence length

$c$  = batch size x hidden size

$$e_{ij} = f(\bar{h}_i, h_j)$$
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$c_i = \sum_j \alpha_{ij} h_j$$

- Make sure tensors are the right size!

Luong et al. (2015)

# Results

---

Luong et al. (2015)  
Chopra et al. (2016)  
Jia and Liang (2016)

# Results

---

- ▶ Machine translation: BLEU score of 14.0 on English-German -> 16.8 with attention, 19.0 with smarter attention (we'll come back to this later)

Luong et al. (2015)  
Chopra et al. (2016)  
Jia and Liang (2016)



# Results

---

- ▶ Machine translation: BLEU score of 14.0 on English-German -> 16.8 with attention, 19.0 with smarter attention (we'll come back to this later)
- ▶ Summarization/headline generation: bigram recall from 11% -> 15%

Luong et al. (2015)  
Chopra et al. (2016)  
Jia and Liang (2016)

# Results

---

- ▶ Machine translation: BLEU score of 14.0 on English-German -> 16.8 with attention, 19.0 with smarter attention (we'll come back to this later)
- ▶ Summarization/headline generation: bigram recall from 11% -> 15%
- ▶ Semantic parsing: ~30% accuracy -> 70+% accuracy on Geoquery

Luong et al. (2015)  
Chopra et al. (2016)  
Jia and Liang (2016)

# Copying Input/Pointers

# Unknown Words

*en*: The ecotax portico in Pont-de-Buis , ... [truncated] ... , was taken down on Thursday morning

*fr*: Le portique écotaxe de Pont-de-Buis , ... [truncated] ... , a été démonté jeudi matin

*nn*: Le unk de unk à unk , ... [truncated] ... , a été pris le jeudi matin

# Unknown Words

*en*: The ecotax portico in Pont-de-Buis , ... [truncated] ... , was taken down on Thursday morning

*fr*: Le portique écotaxe de Pont-de-Buis , ... [truncated] ... , a été démonté jeudi matin

*nn*: Le unk de unk à unk , ... [truncated] ... , a été pris le jeudi matin

# Unknown Words

*en*: The ecotax portico in Pont-de-Buis , ... [truncated] ... , was taken down on Thursday morning

*fr*: Le portique écotaxe de Pont-de-Buis , ... [truncated] ... , a été démonté jeudi matin

*nn*: Le unk de unk à unk , ... [truncated] ... , a été pris le jeudi matin

- Want to be able to copy named entities like Pont-de-Buis



# Unknown Words

*en*: The ecotax portico in Pont-de-Buis , ... [truncated] ... , was taken down on Thursday morning

*fr*: Le portique écotaxe de Pont-de-Buis , ... [truncated] ... , a été démonté jeudi matin

*nn*: Le unk de unk à unk , ... [truncated] ... , a été pris le jeudi matin

- Want to be able to copy named entities like Pont-de-Buis

$$P(y_i | \mathbf{x}, y_1, \dots, y_{i-1}) = \text{softmax}(W[c_i; \bar{h}_i])$$

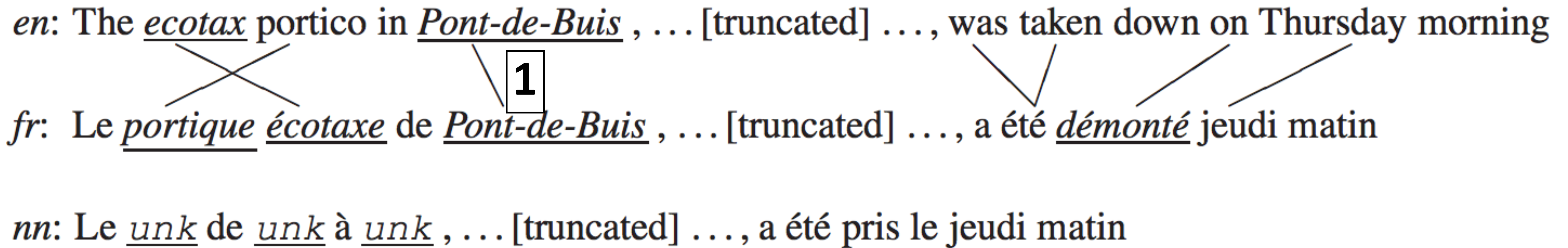
from attention      from RNN hidden state

# Unknown Words

*en*: The ecotax portico in Pont-de-Buis , ... [truncated] ..., was taken down on Thursday morning

*fr*: Le portique écotaxe de Pont-de-Buis , ... [truncated] ..., a été démonté jeudi matin

*nn*: Le unk de unk à unk , ... [truncated] ..., a été pris le jeudi matin



- Want to be able to copy named entities like Pont-de-Buis

$$P(y_i | \mathbf{x}, y_1, \dots, y_{i-1}) = \text{softmax}(W[c_i; \bar{h}_i])$$

from attention

from RNN  
hidden state

- Still can only generate from the vocabulary

Jean et al. (2015), Luong et al. (2015)



# Copying

---

*en*: The ecotax portico in Pont-de-Buis , ... [truncated] ..

*fr*: Le portique écotaxe de Pont-de-Buis , ... [truncated] .

*nn*: Le unk de unk à unk , ... [truncated] ..., a été pris

# Copying

---

*en*: The ecotax portico in Pont-de-Buis , ... [truncated] ..

*fr*: Le portique écotaxe de Pont-de-Buis , ... [truncated] .

*nn*: Le unk de unk à unk , ... [truncated] ..., a été pris

- Vocabulary contains “normal” vocab as well as words in input. Normalizes over both of these:

# Copying

*en*: The ecotax portico in Pont-de-Buis , ... [truncated] ..

*fr*: Le portique écotaxe de Pont-de-Buis , ... [truncated] .

*nn*: Le unk de unk à unk , ... [truncated] ..., a été pris

- Vocabulary contains “normal” vocab as well as words in input. Normalizes over both of these:

{ the  
a  
...  
zebra  
-----  
Pont-de-Buis  
ecotax }

# Copying

*en*: The ecotax portico in Pont-de-Buis , ... [truncated] ..

*fr*: Le portique écotaxe de Pont-de-Buis , ... [truncated] .

*nn*: Le unk de unk à unk , ... [truncated] ..., a été pris

- Vocabulary contains “normal” vocab as well as words in input. Normalizes over both of these:

$$P(y_i = w | \mathbf{x}, y_1, \dots, y_{i-1}) \propto \begin{cases} \exp W_w [c_i; \bar{h}_i] & \text{if } w \text{ in vocab} \\ h_j^\top V \bar{h}_i & \text{if } w = x_j \end{cases}$$

{ the  
a  
...  
zebra  
-----  
Pont-de-Buis  
ecotax }



# Copying

*en*: The ecotax portico in Pont-de-Buis , ... [truncated] ..

*fr*: Le portique écotaxe de Pont-de-Buis , ... [truncated] .

*nn*: Le unk de unk à unk , ... [truncated] ..., a été pris

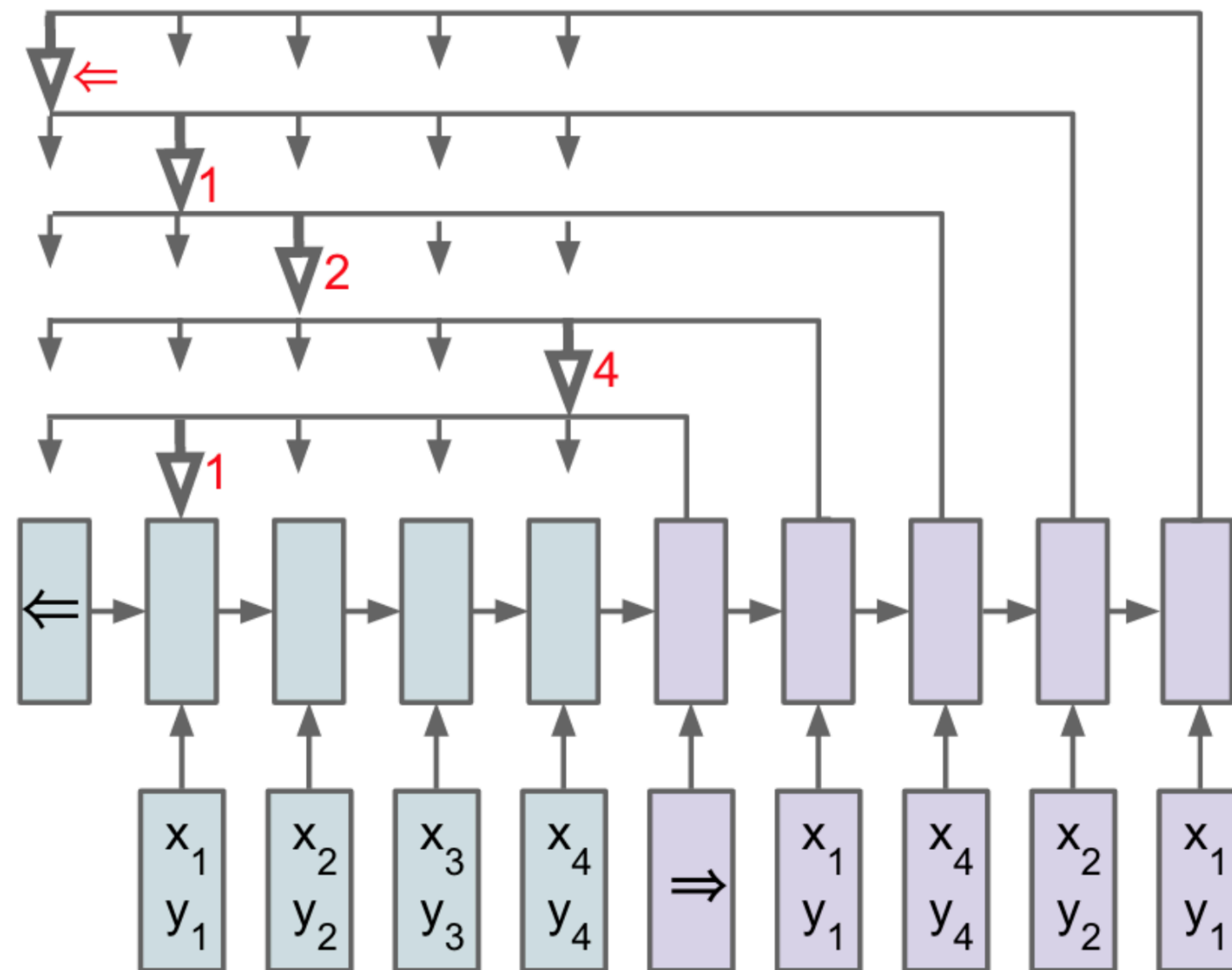
- Vocabulary contains “normal” vocab as well as words in input. Normalizes over both of these:

$$P(y_i = w | \mathbf{x}, y_1, \dots, y_{i-1}) \propto \begin{cases} \exp W_w [c_i; \bar{h}_i] & \text{if } w \text{ in vocab} \\ h_j^\top V \bar{h}_i & \text{if } w = x_j \end{cases}$$

- Bilinear function of input representation + output hidden state

{  
the  
a  
...  
zebra  
-----  
Pont-de-Buis  
ecotax  
}

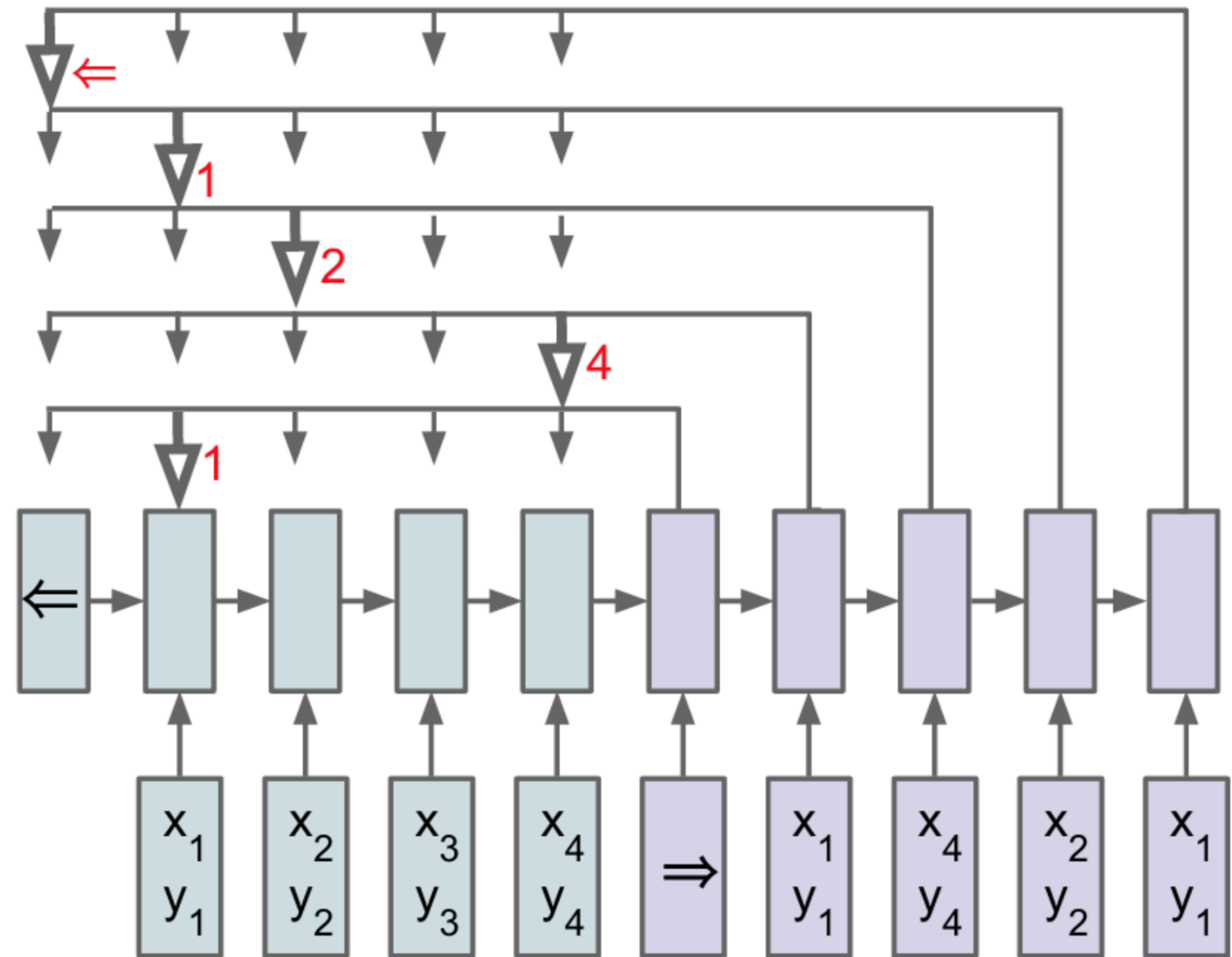
# Pointer Networks



Vinyals et al. (2015)

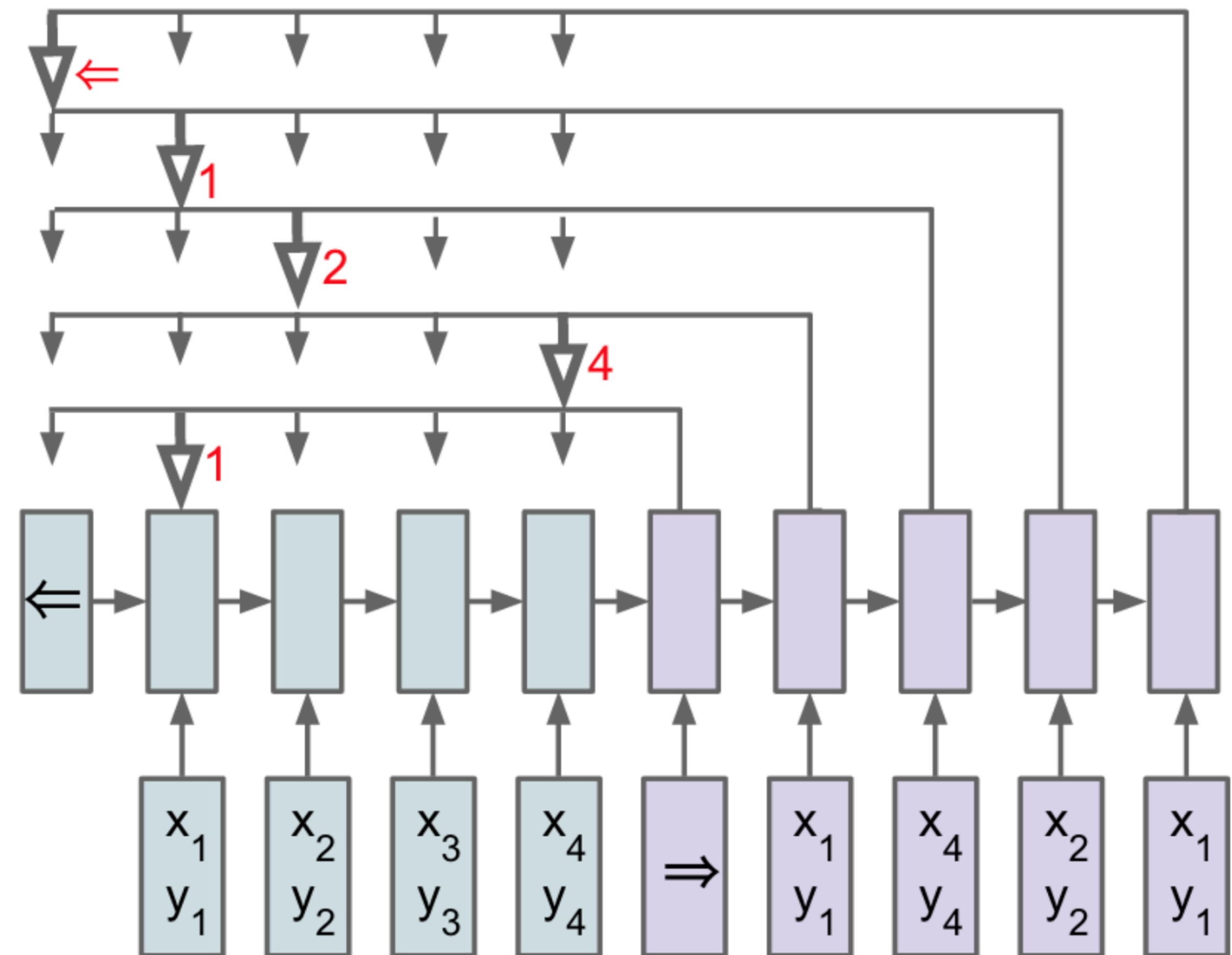
# Pointer Networks

- Only point to the input, don't have any notion of vocabulary



# Pointer Networks

- ▶ Only point to the input, don't have any notion of vocabulary
- ▶ Used for tasks including summarization and sentence ordering





# Results

---

	GEO	ATIS
No Copying	74.6	69.9
With Copying	85.0	76.3

# Results

---

	GEO	ATIS
No Copying	74.6	69.9
With Copying	85.0	76.3

- For semantic parsing, copying tokens from the input (texas) can be very useful

# Results

---

	GEO	ATIS
No Copying	74.6	69.9
With Copying	85.0	76.3

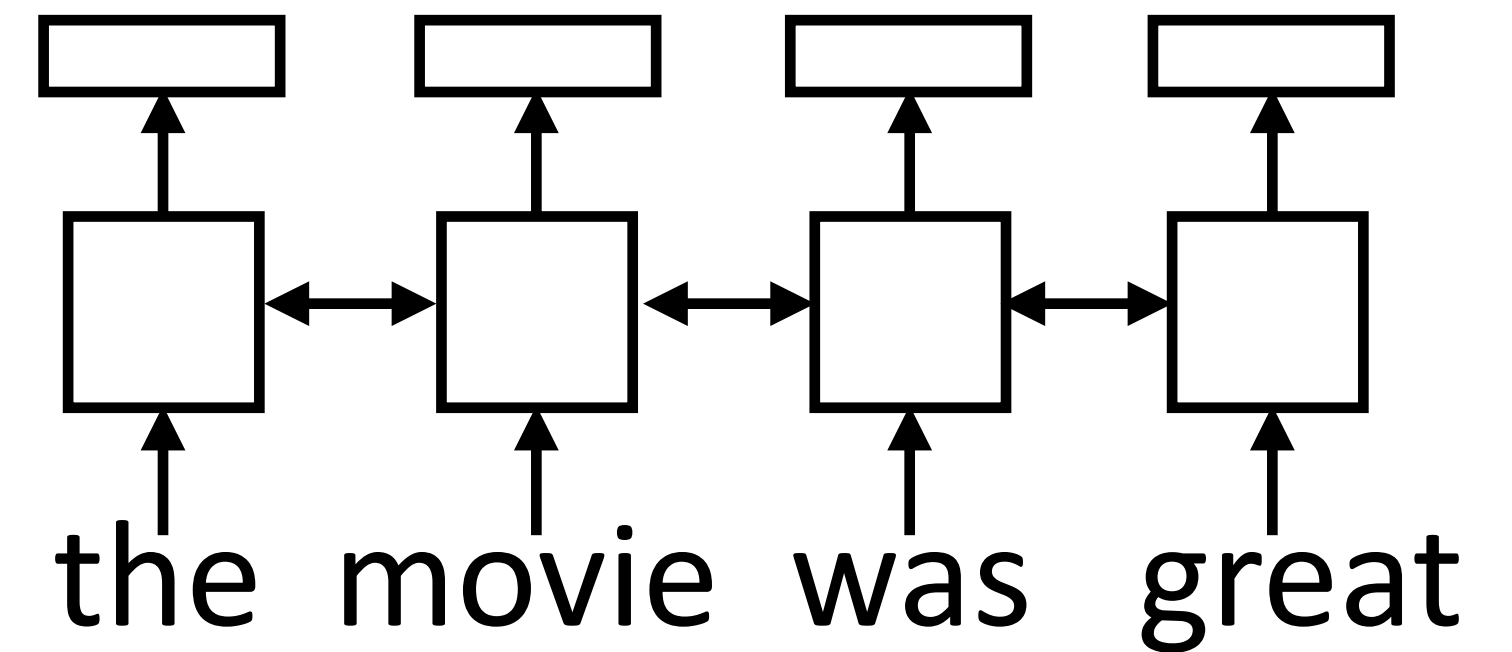
- ▶ For semantic parsing, copying tokens from the input (texas) can be very useful
- ▶ In many settings, attention can roughly do the same things as copying

# Transformers

# Self-Attention

---

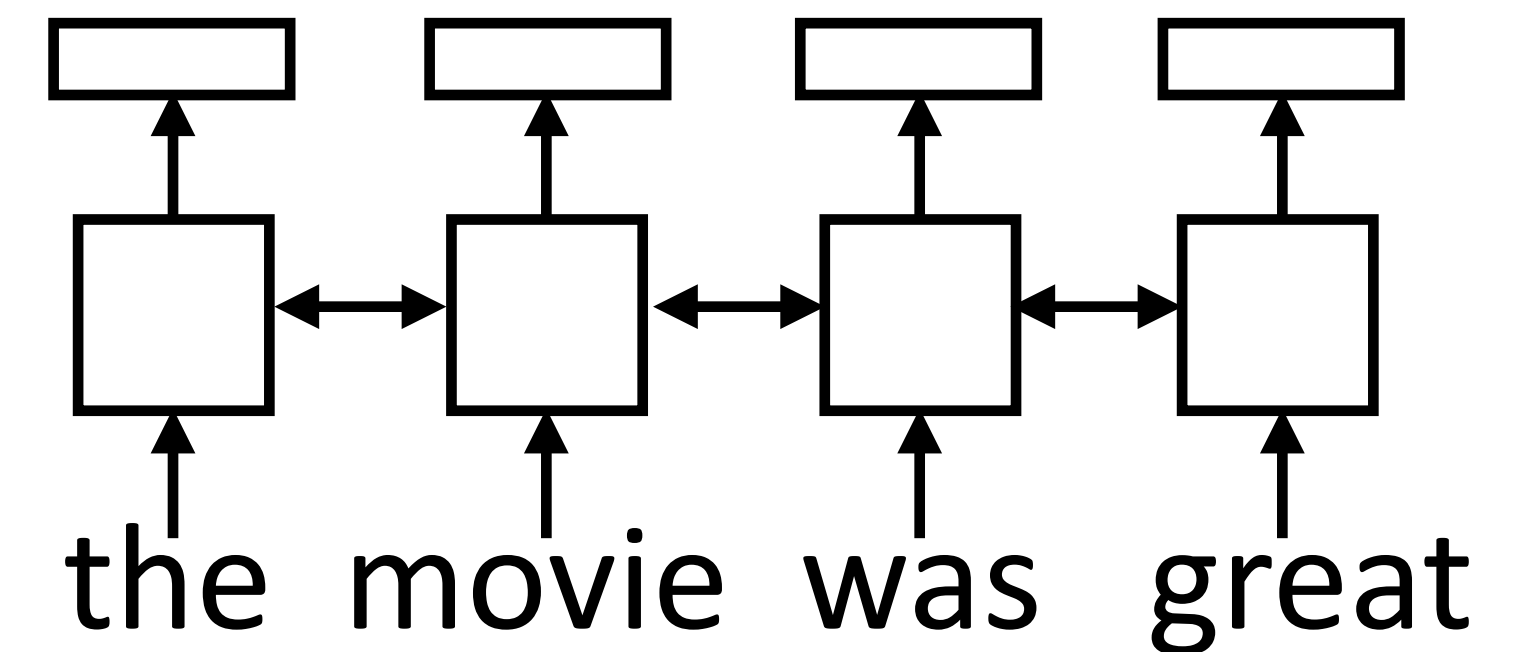
- ▶ LSTM abstraction: maps each vector in a sentence to a new, context-aware vector



# Self-Attention

---

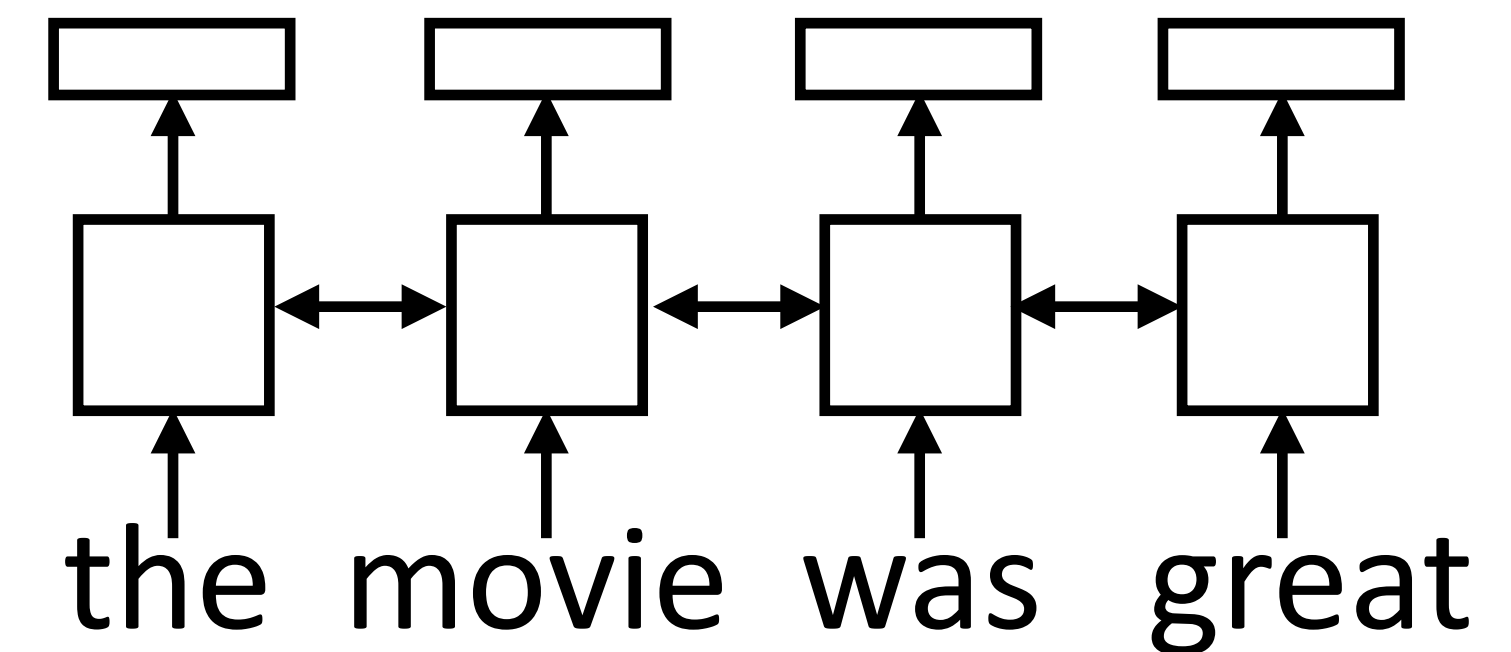
- ▶ LSTM abstraction: maps each vector in a sentence to a new, context-aware vector
- ▶ CNNs did something similar with filters



# Self-Attention

---

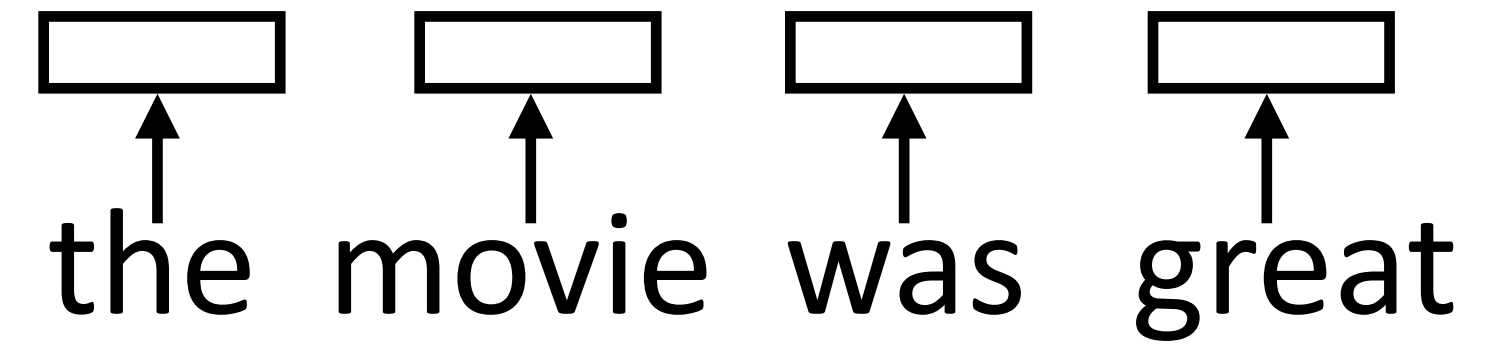
- ▶ LSTM abstraction: maps each vector in a sentence to a new, context-aware vector
- ▶ CNNs did something similar with filters
- ▶ Attention can give us a third way to do this



# Self-Attention

---

- ▶ Each word forms a “query” which then computes attention over each word

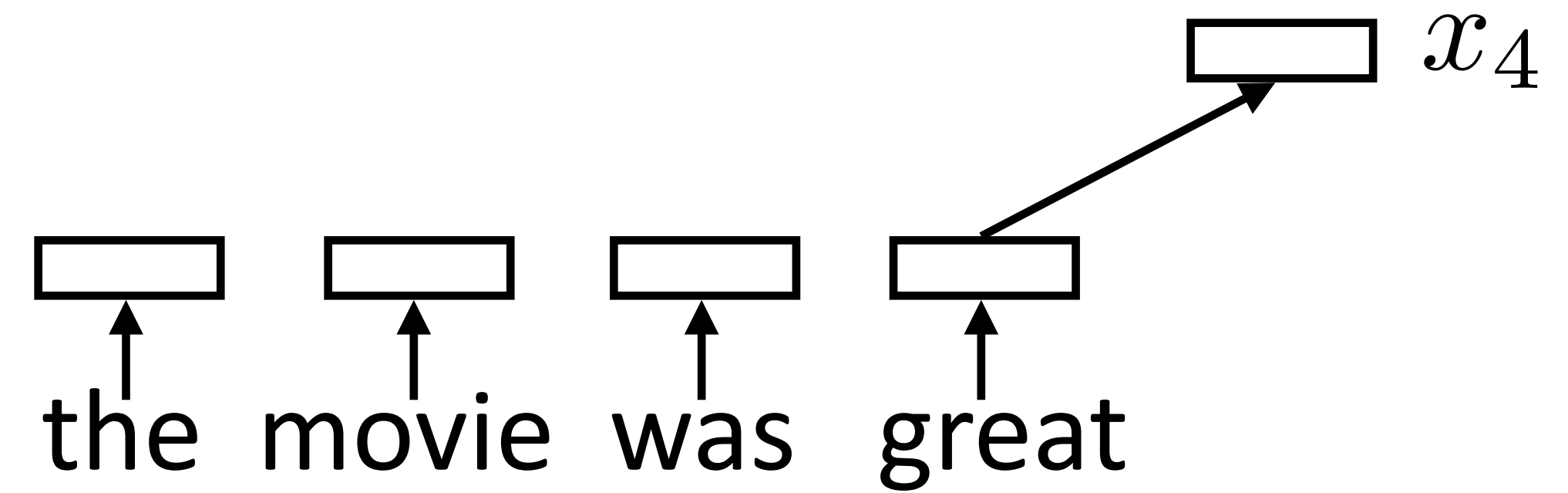




# Self-Attention

---

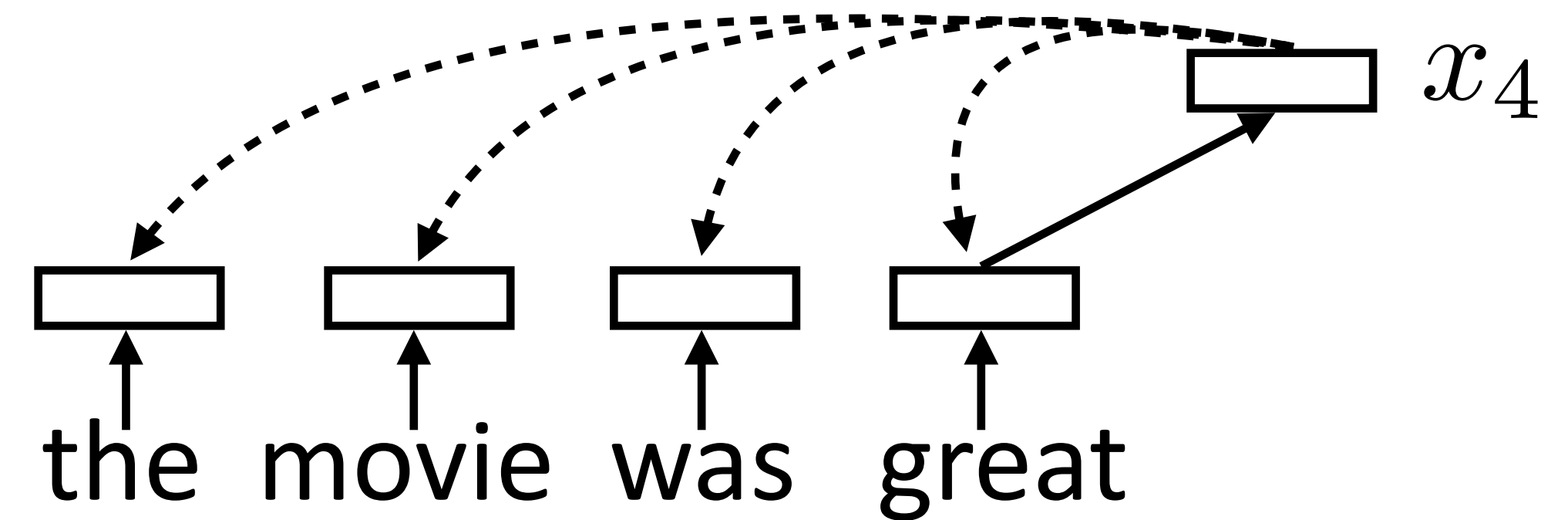
- ▶ Each word forms a “query” which then computes attention over each word



# Self-Attention

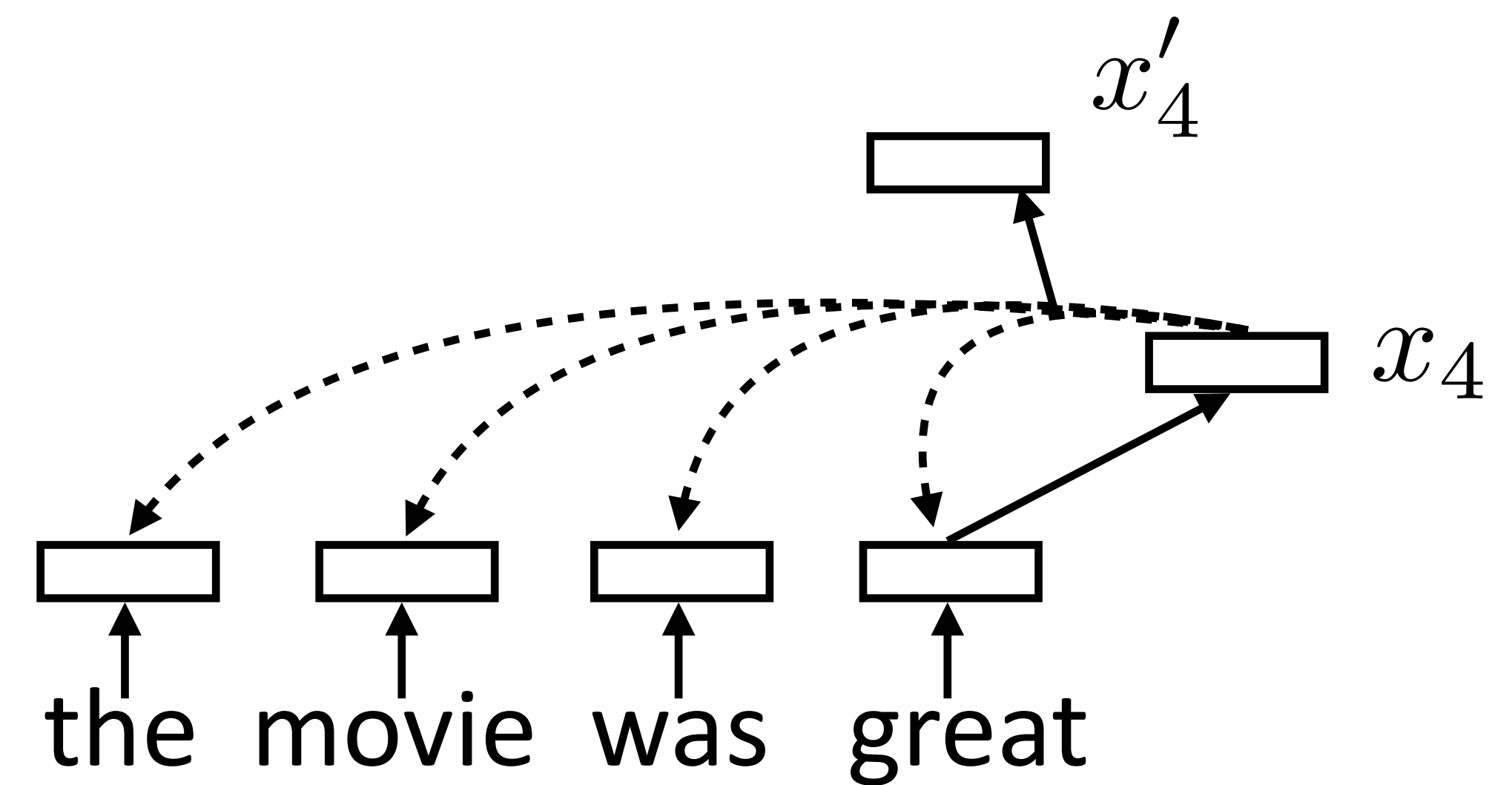
---

- ▶ Each word forms a “query” which then computes attention over each word



# Self-Attention

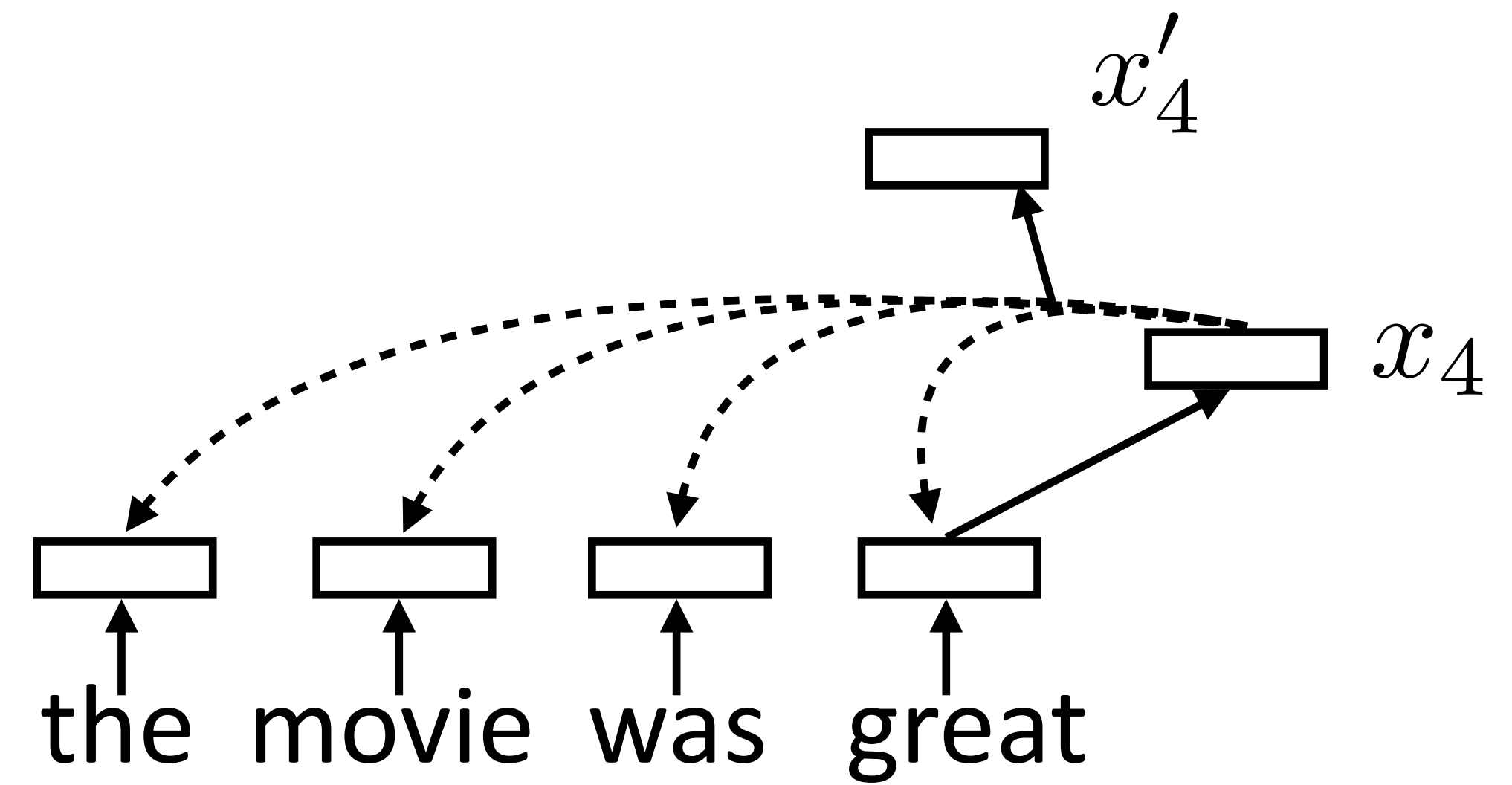
- ▶ Each word forms a “query” which then computes attention over each word



# Self-Attention

- Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

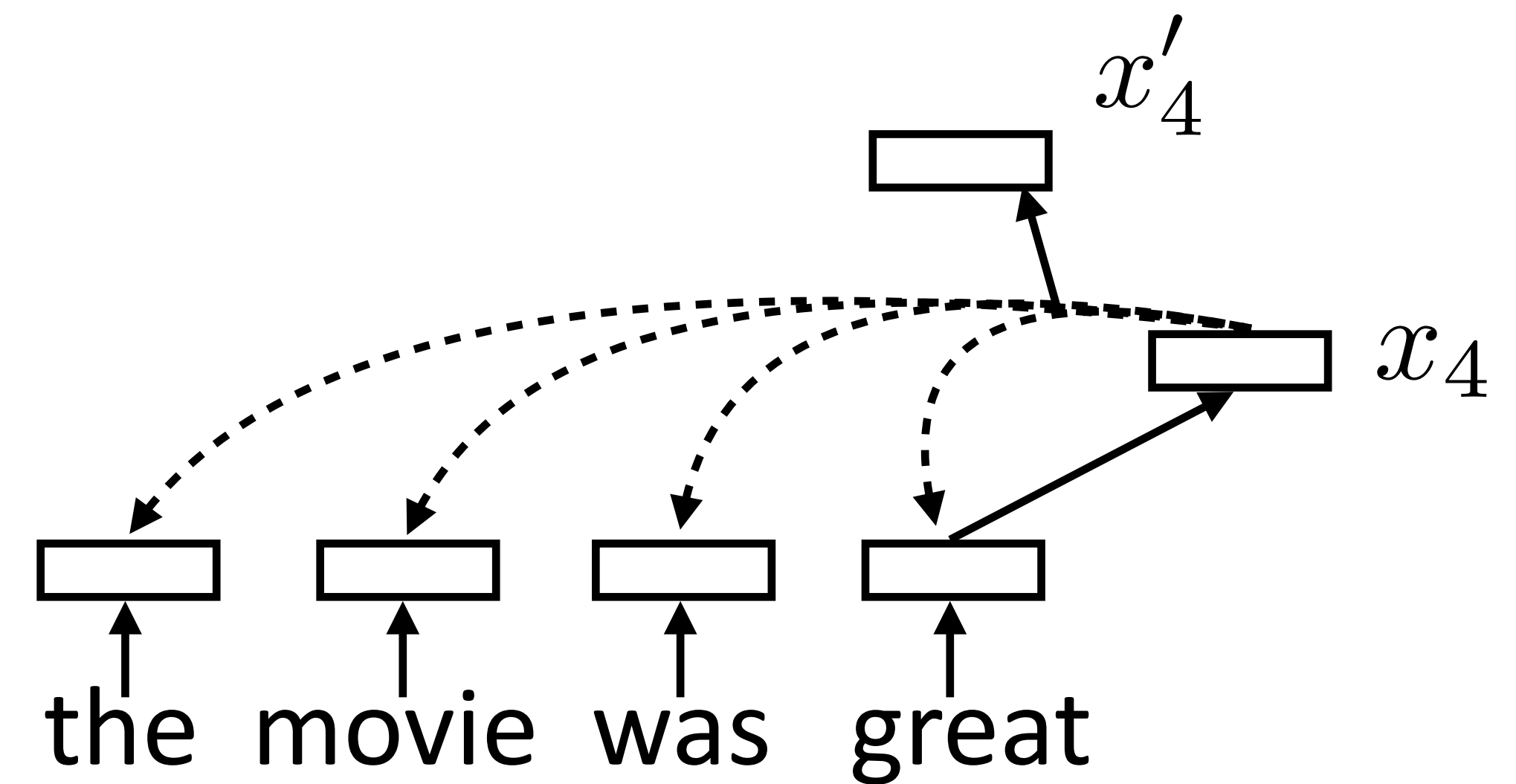


# Self-Attention

- Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^n \alpha_{i,j} x_j \quad \text{vector} = \text{sum of scalar} * \text{vector}$$

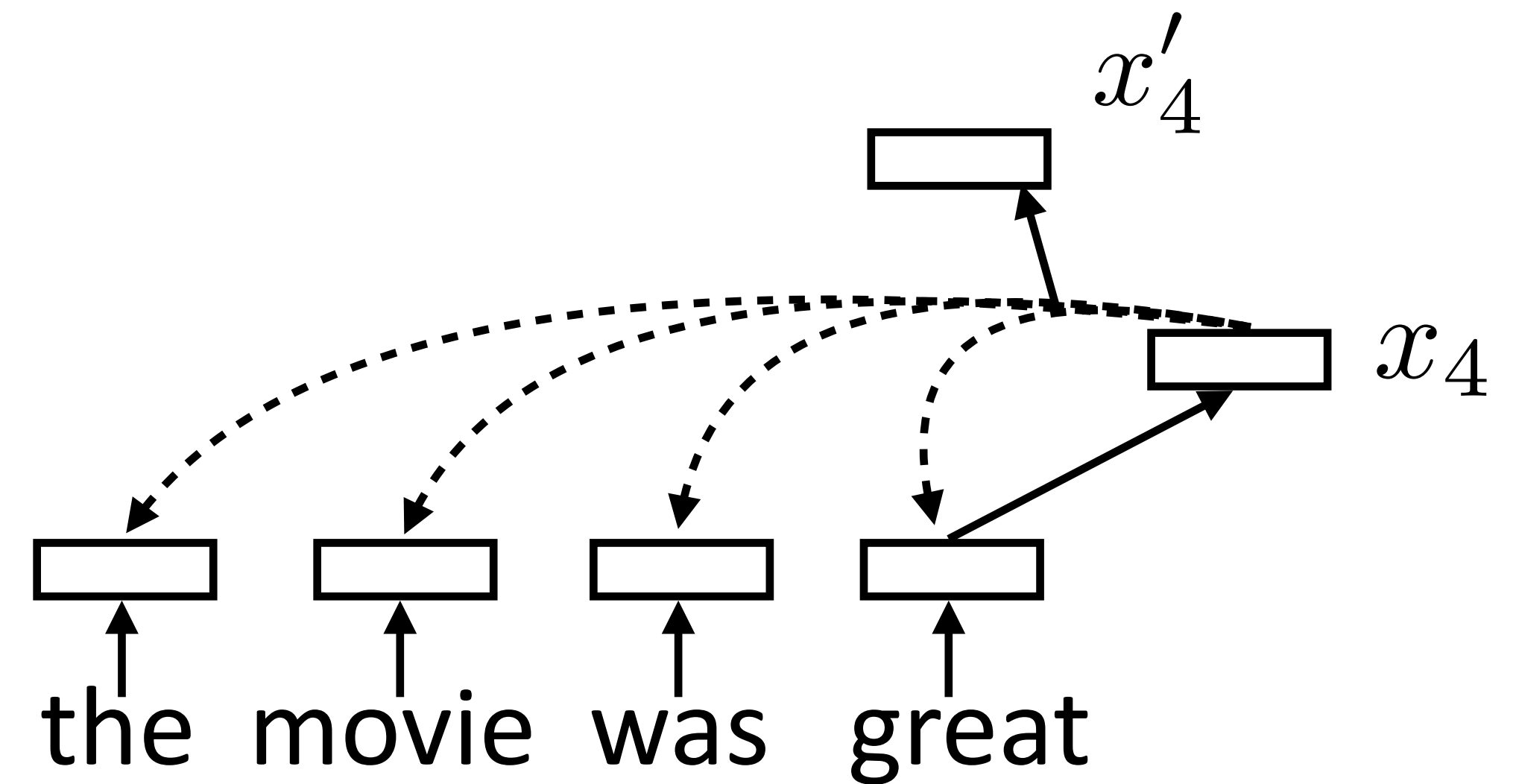


# Self-Attention

- Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^n \alpha_{i,j} x_j \quad \text{vector} = \text{sum of scalar} * \text{vector}$$



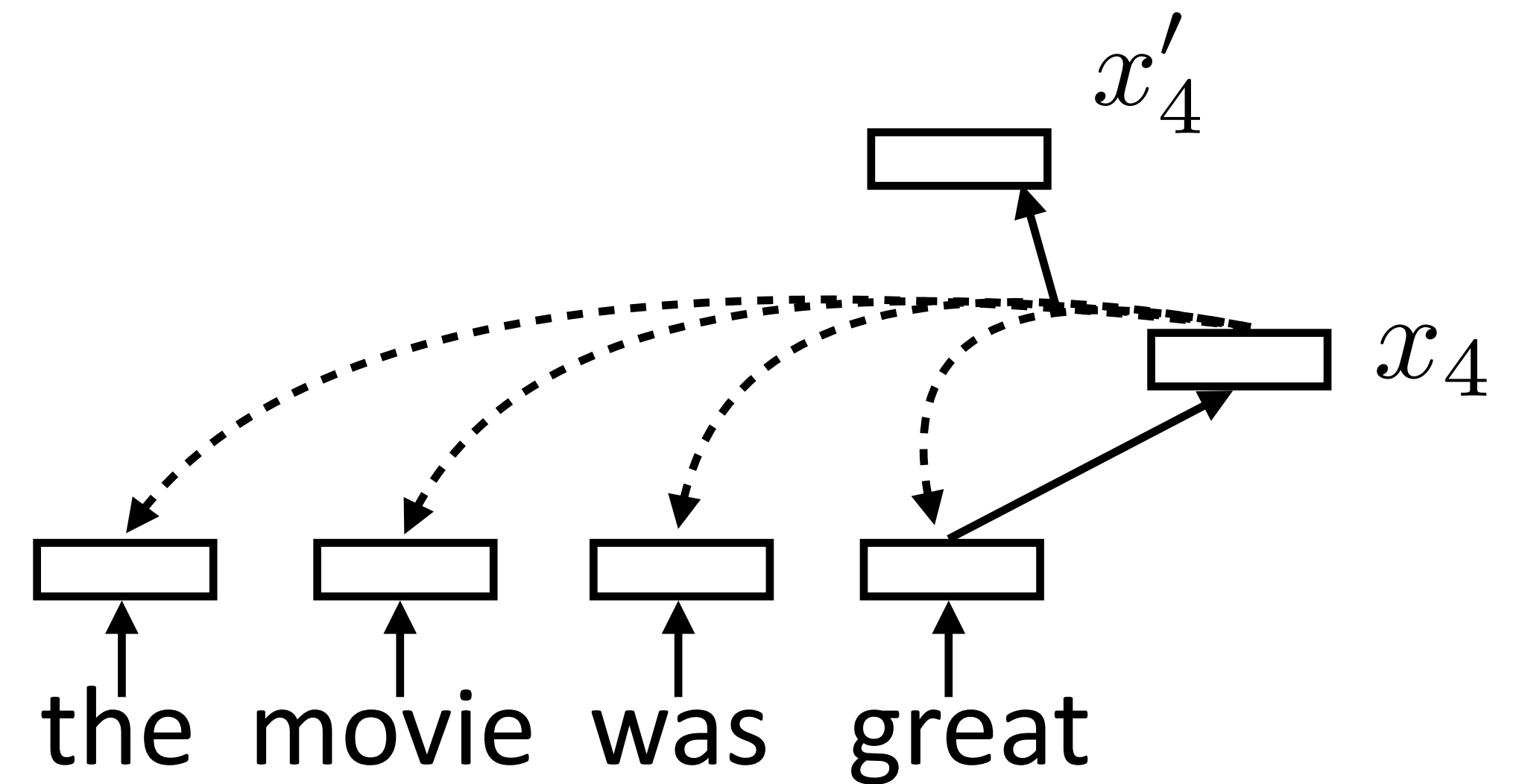
- Multiple “heads” analogous to different convolutional filters. Use parameters  $W_k$  and  $V_k$  to get different attention values + transform vectors

# Self-Attention

- Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^n \alpha_{i,j} x_j \quad \text{vector} = \text{sum of scalar} * \text{vector}$$



- Multiple “heads” analogous to different convolutional filters. Use parameters  $W_k$  and  $V_k$  to get different attention values + transform vectors

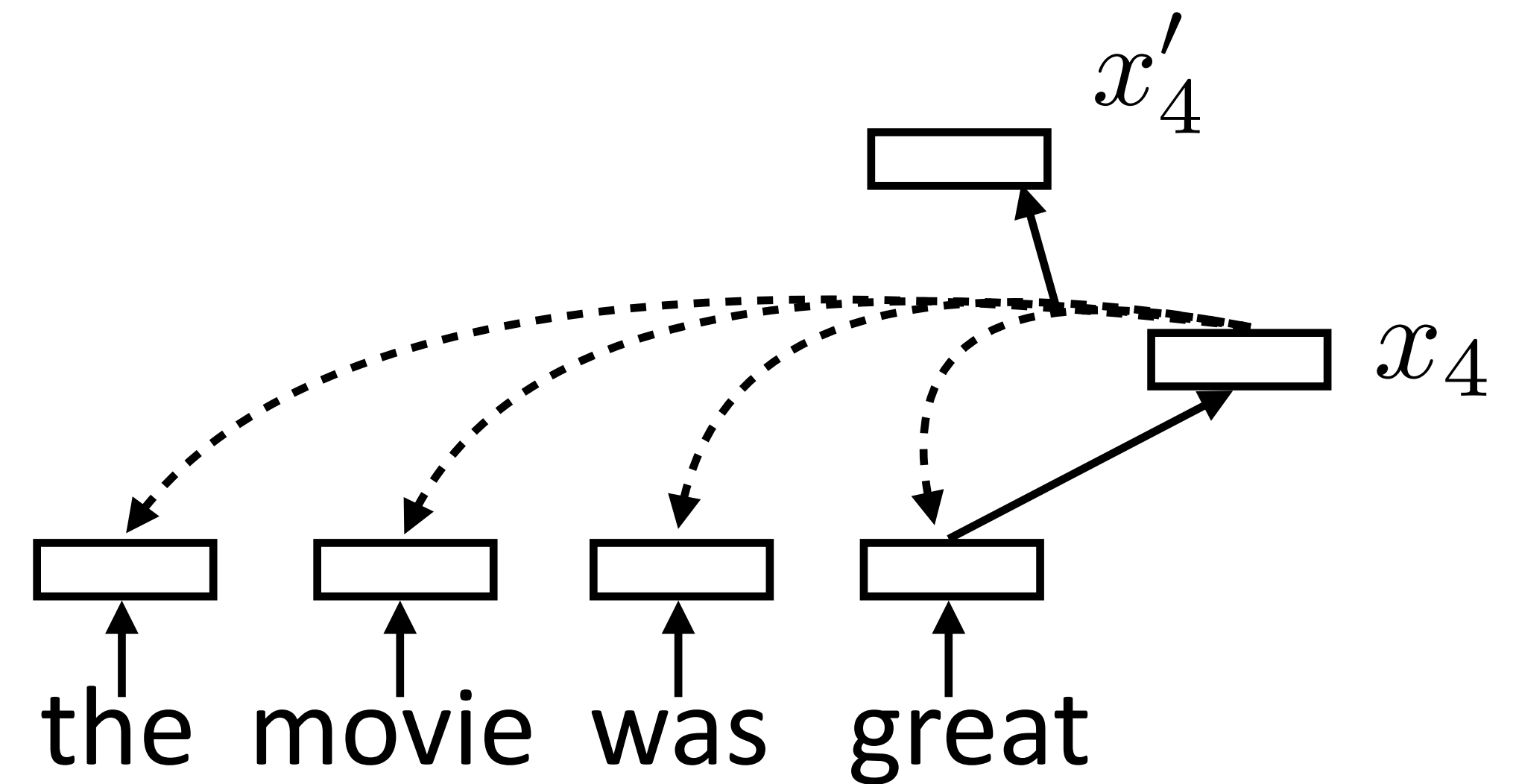
$$\alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j)$$

# Self-Attention

- Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^n \alpha_{i,j} x_j \quad \text{vector} = \text{sum of scalar} * \text{vector}$$



- Multiple “heads” analogous to different convolutional filters. Use parameters  $W_k$  and  $V_k$  to get different attention values + transform vectors

$$\alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j) \quad x'_{k,i} = \sum_{j=1}^n \alpha_{k,i,j} V_k x_j$$

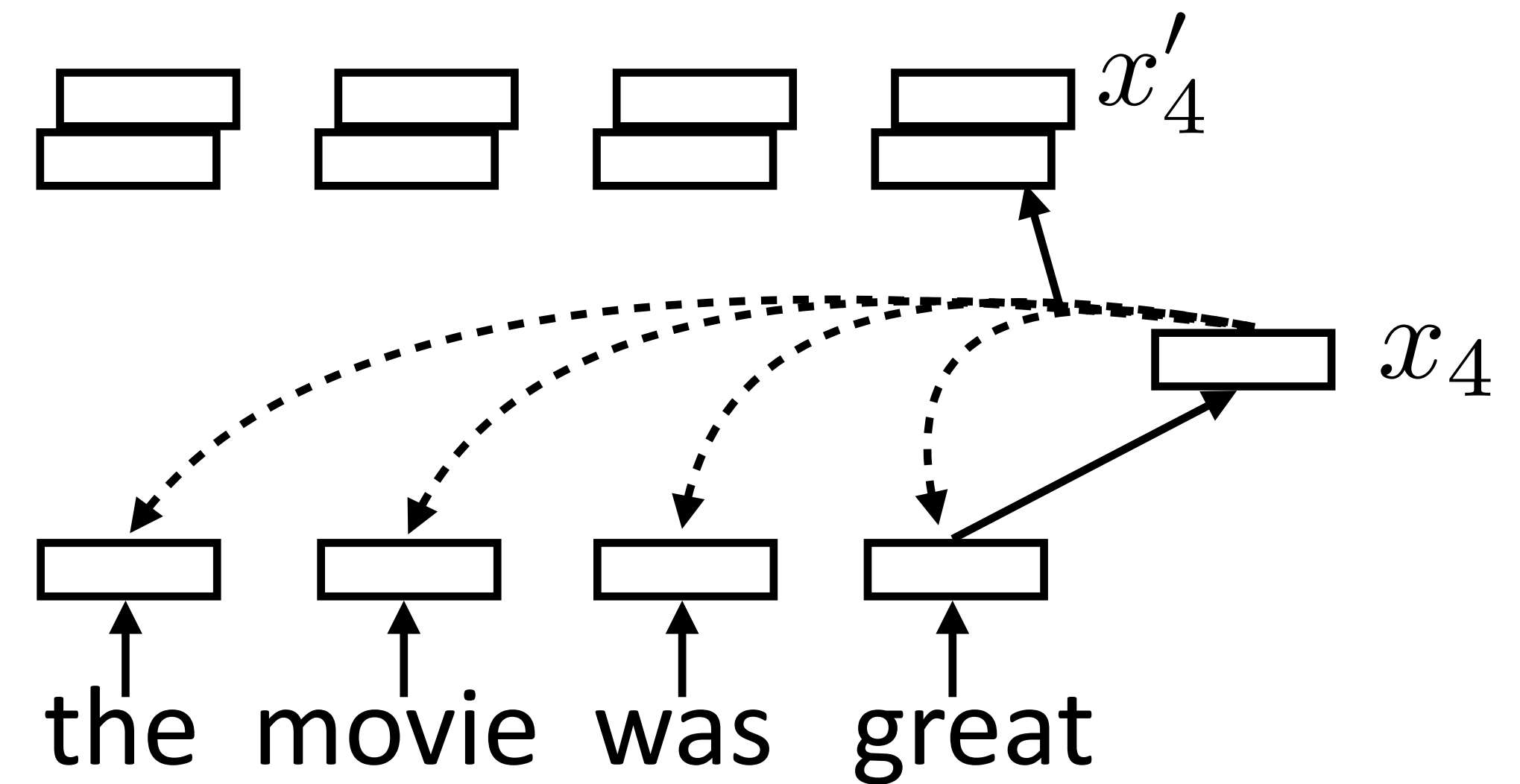


# Self-Attention

- Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^n \alpha_{i,j} x_j \quad \text{vector} = \text{sum of scalar} * \text{vector}$$



- Multiple “heads” analogous to different convolutional filters. Use parameters  $W_k$  and  $V_k$  to get different attention values + transform vectors

$$\alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j) \quad x'_{k,i} = \sum_{j=1}^n \alpha_{k,i,j} V_k x_j$$

# Deep Transformers

---

- ▶ Supervised: transformer can replace LSTM; will revisit this when we discuss MT

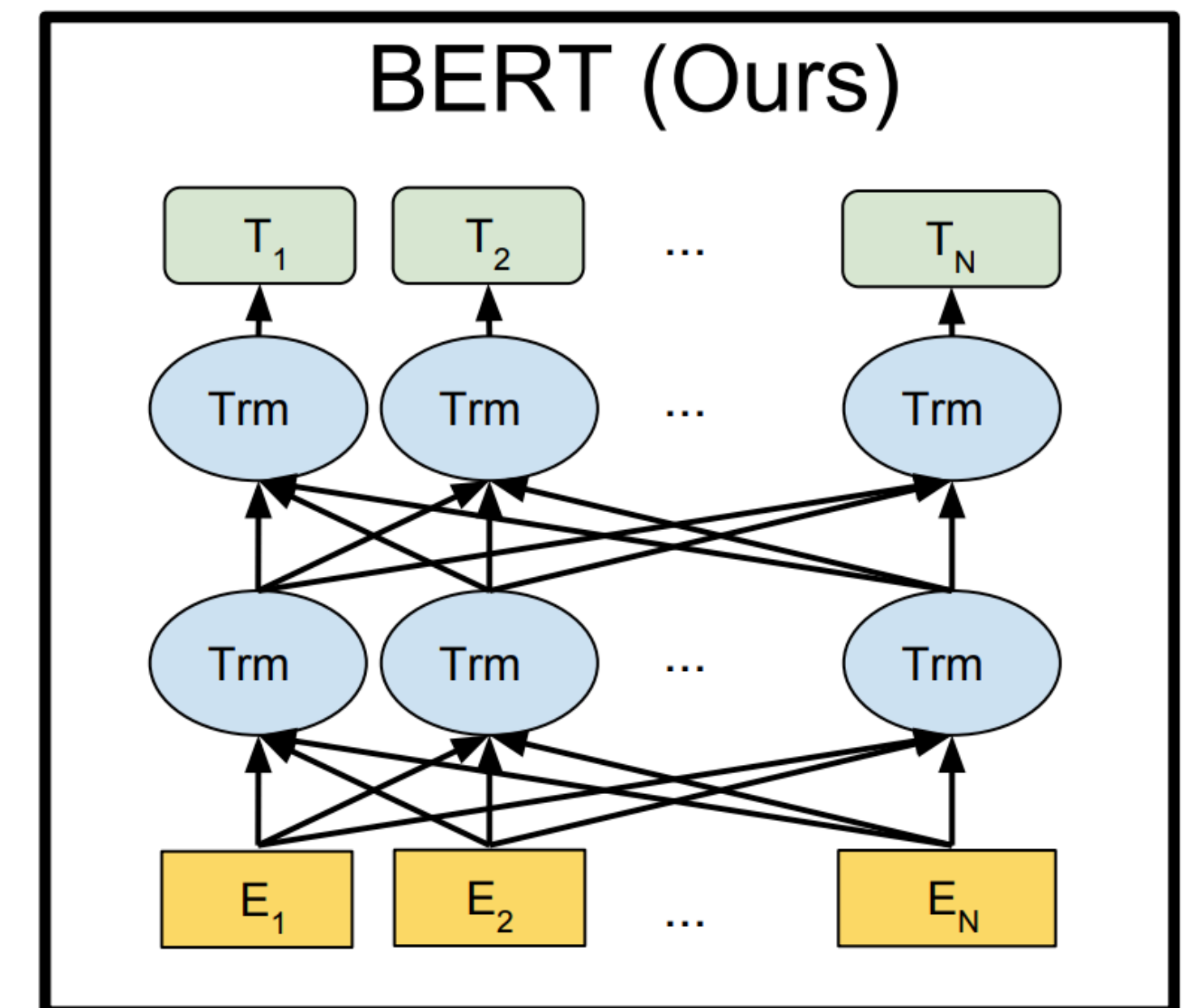
# Deep Transformers

---

- ▶ Supervised: transformer can replace LSTM; will revisit this when we discuss MT
- ▶ Unsupervised: transformers work better than LSTM for unsupervised pre-training of embeddings: predict word given context words

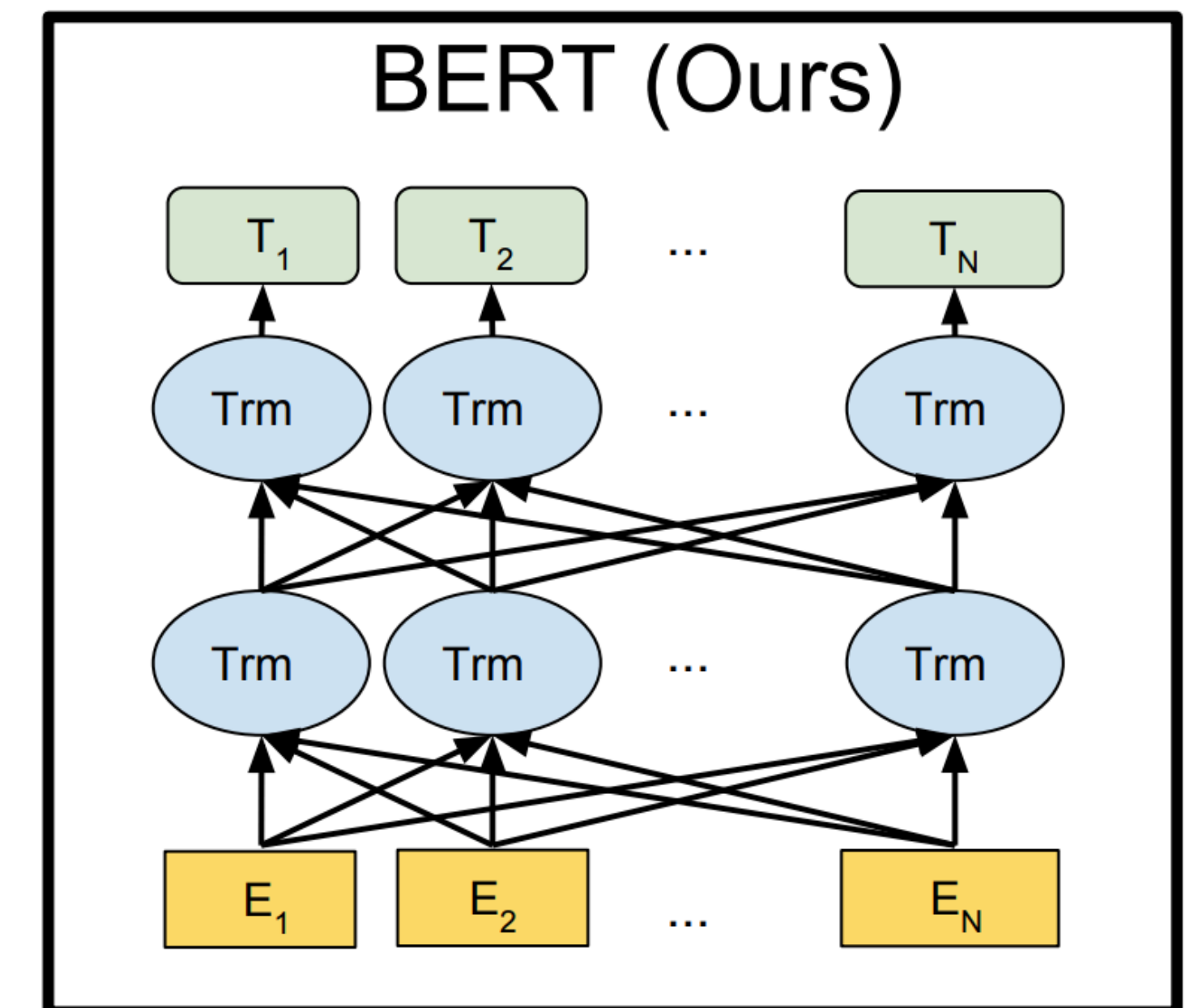
# Deep Transformers

- ▶ Supervised: transformer can replace LSTM; will revisit this when we discuss MT
- ▶ Unsupervised: transformers work better than LSTM for unsupervised pre-training of embeddings: predict word given context words
- ▶ Devlin et al. October 11, 2018  
“BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”



# Deep Transformers

- ▶ Supervised: transformer can replace LSTM; will revisit this when we discuss MT
- ▶ Unsupervised: transformers work better than LSTM for unsupervised pre-training of embeddings: predict word given context words
- ▶ Devlin et al. October 11, 2018  
“BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”
- ▶ Stronger than similar methods, SOTA on ~11 tasks (including NER — 92.8 F1)



# Takeaways

---

# Takeaways

---

- ▶ Attention is very helpful for seq2seq models

# Takeaways

---

- ▶ Attention is very helpful for seq2seq models
- ▶ Used for tasks including summarization and sentence ordering



# Takeaways

---

- ▶ Attention is very helpful for seq2seq models
- ▶ Used for tasks including summarization and sentence ordering
- ▶ Explicitly copying input can be beneficial as well

# Takeaways

---

- ▶ Attention is very helpful for seq2seq models
- ▶ Used for tasks including summarization and sentence ordering
- ▶ Explicitly copying input can be beneficial as well
- ▶ Transformers are strong models we'll come back to later