From Attention to Transformers

Antoine Bosselut

Attention Recap

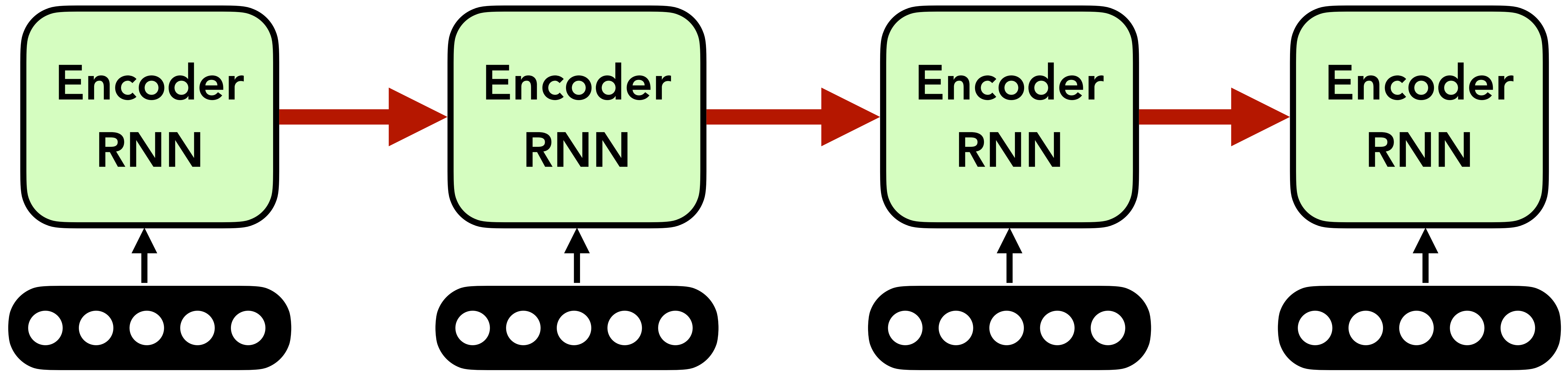
- **Main Idea:** Decoder computes a weighted sum of encoder outputs
  - Compute pairwise score between each encoder hidden state and initial decoder hidden state
- Many possible functions for computing scores (dot product, bilinear, etc.)
- **Temporal Bottleneck Fixed! Direct link** between decoder and encoder states
  - Helps with vanishing gradients and modelling long-term dependencies!
- Attention is **agnostic** to the type of RNN used in the encoder and decoder!

Question

Do any other inefficiencies remain in our sequence to sequence pipelines?

Encoder is still Recurrent

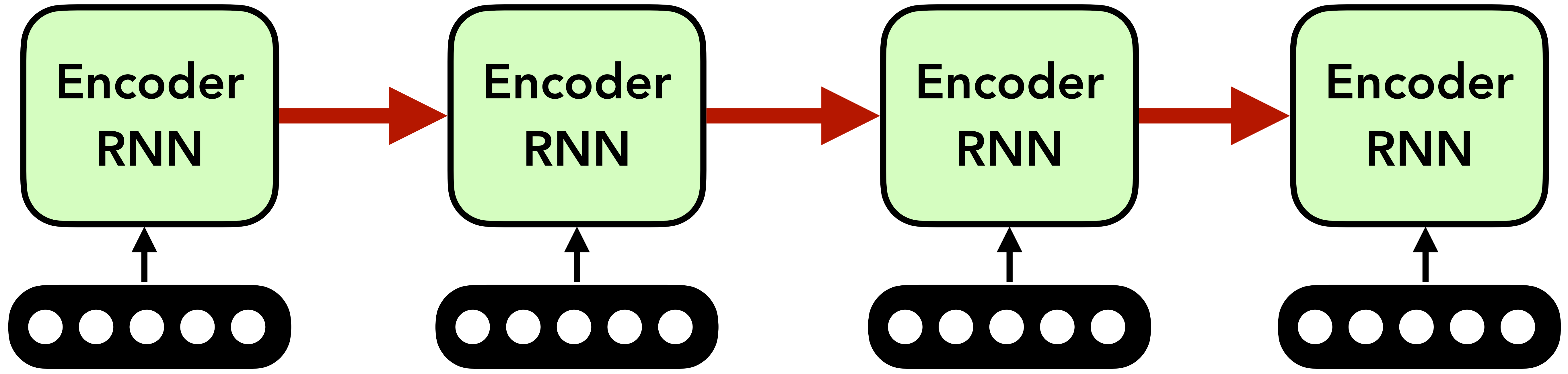
- **Encoder:** Recurrent functions can't be parallelized because previous state needs to be computed to encode next one



- **Problem: Encoder hidden states must still be computed in series**

Encoder is still Recurrent

- **Encoder:** Recurrent functions can't be parallelized because previous state needs to be computed to encode next one



- **Problem: Encoder hidden states must still be computed in series**

Who can think of a task where this might be a problem?

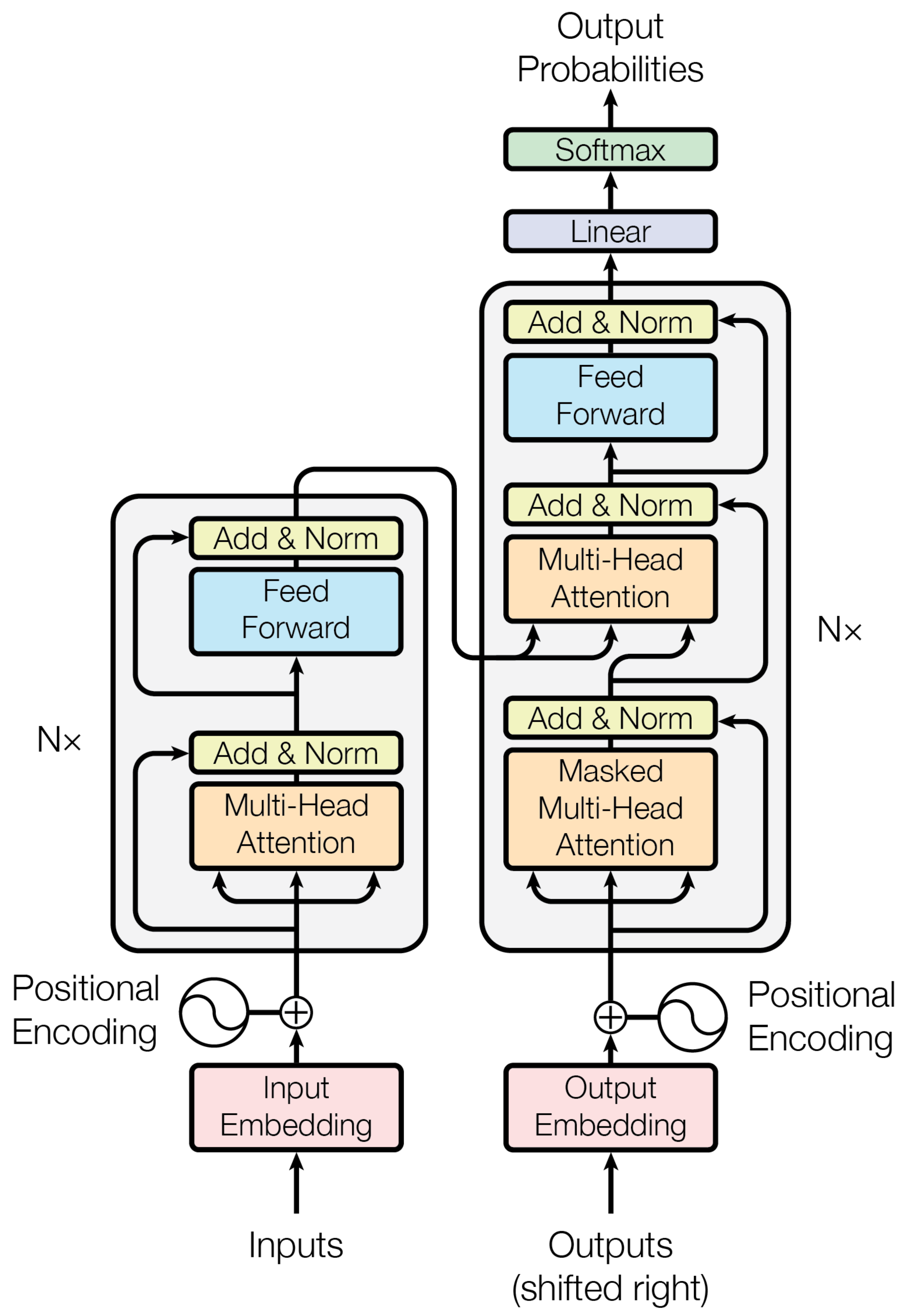
Solution:

Transformers!

Full Transformer

- Made up of encoder and decoder
- Both encoder and decoder made up of multiple cascaded transformer blocks
  - slightly different architecture in encoder and decoder transformer blocks
- Blocks generally made up **multi-headed attention** layers (self-attention) and **feedforward** layers
- No recurrent computations!

Encode sequences with self-attention

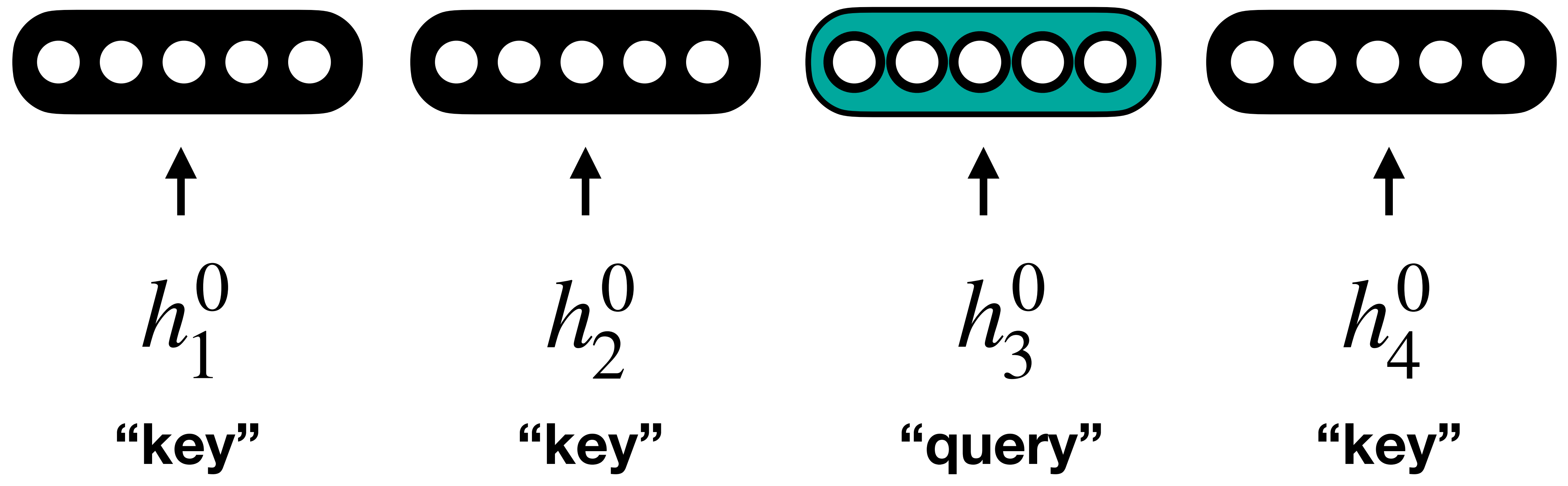


(Vaswani et al., 2017)

Self-Attention Toy Example

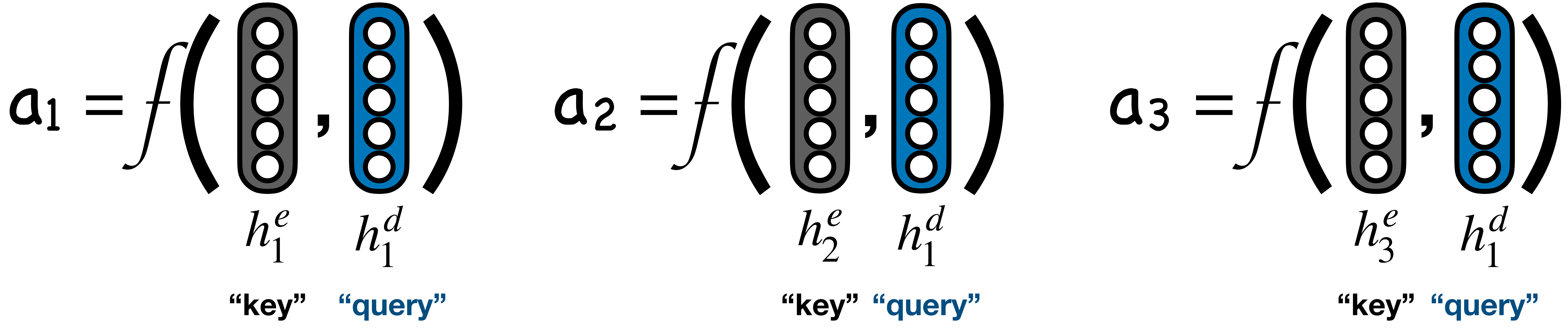
- **Original Idea:** Use decoder hidden state to compute attention distribution over encoder hidden states
- **New Idea: Could we use encoder hidden states to compute attention distribution over themselves?**
- **Ditch recurrence** and compute encoder state representations in parallel!

h_t^ℓ = encoder hidden state at time step *t* at layer ℓ



Recap: Attention with RNNs

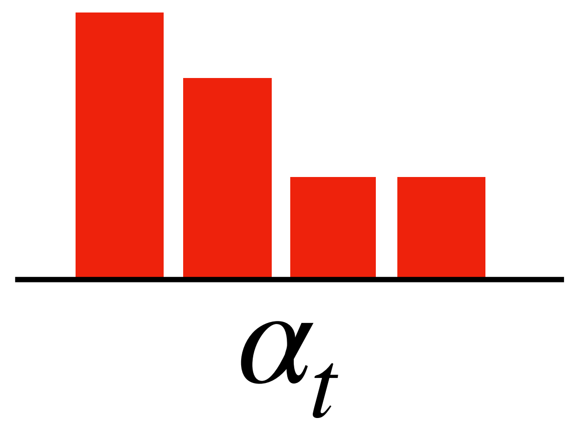
- **Compute** pairwise similarity between each encoder hidden state and decoder hidden state ("idea of what to decode")



- **Convert** pairwise similarity scores to probability **distribution** (using softmax!) over encoder hidden states and compute weighted average:

Softmax!

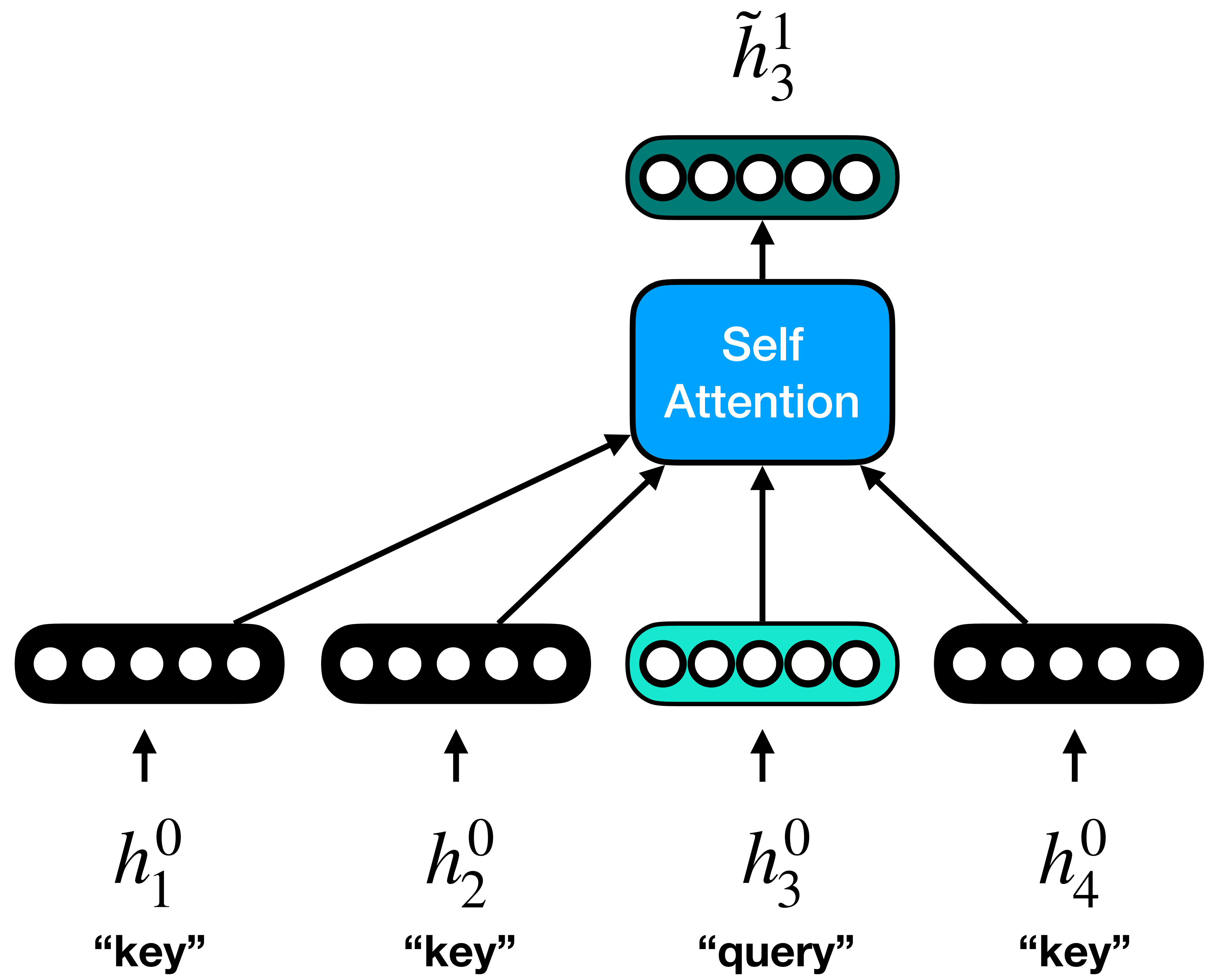
$$\boxed{\alpha_t = \frac{e^{a_t}}{\sum_j e^{a_j}}} \rightarrow$$



$$\rightarrow \tilde{h}_1^d = \sum_{t=1}^{T} \alpha_t h_t^e$$

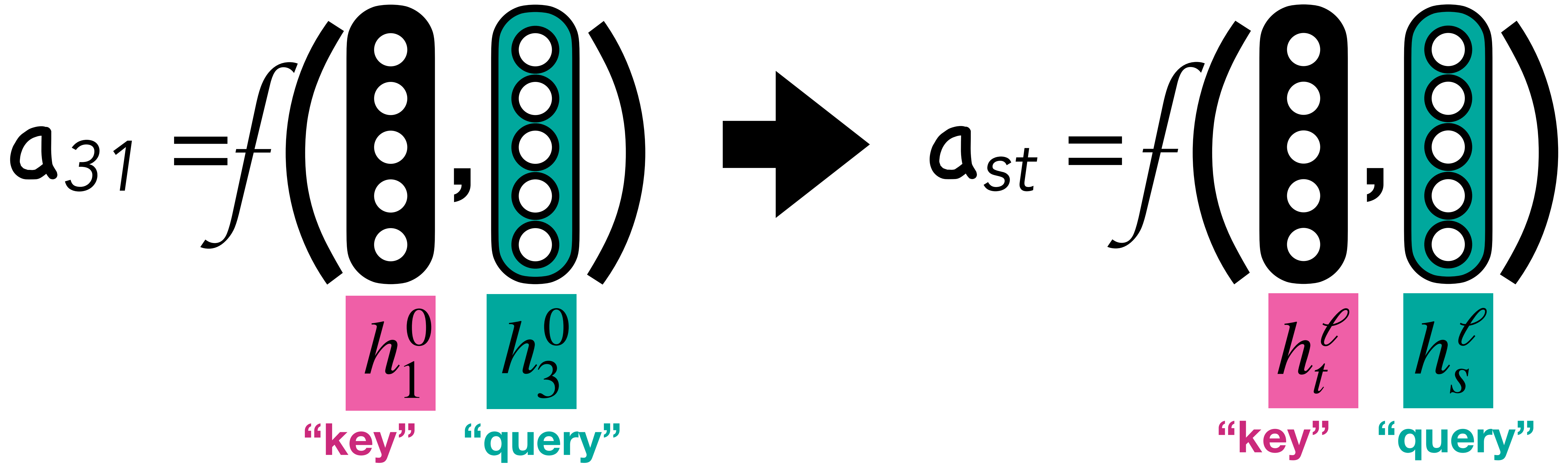
Here h_t^e is known as the "value"

Self-Attention Toy Example



Self-Attention Toy Example

h_t^ℓ = encoder hidden state at time step t at layer ℓ



$$a_{st} = \frac{(\mathbf{W}^Q h_s^\ell)^T(\mathbf{W}^K h_t^\ell)}{\sqrt{d}}$$

Compute pairwise scores

$$\alpha_{st} = \frac{e^{a_{st}}}{\sum_j e^{a_{sj}}}$$

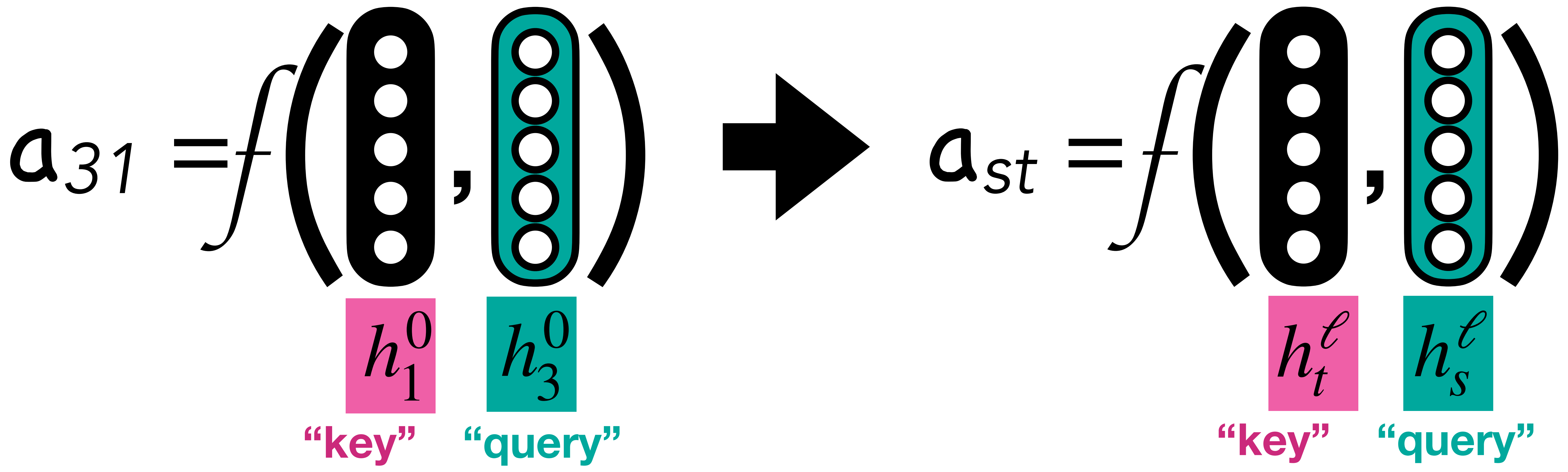
Get attention distribution

$$\tilde{h}_s^\ell = \sum_{t=1}^{T} \alpha_{st}\left(\mathbf{W}^V h_t^\ell\right)$$

Attend to values to get weighted sum

Self-Attention Toy Example

h_t^ℓ = encoder hidden state at time step *t* at layer ℓ



{1, …, *t*, …, T} includes *s*!

$$a_{st} = \frac{(\mathbf{W}^Q h_s^\ell)^T (\mathbf{W}^K h_t^\ell)}{\sqrt{d}}$$

Compute pairwise scores

$$\alpha_{st} = \frac{e^{a_{st}}}{\sum_j e^{a_{sj}}}$$

Get attention distribution

$$\boxed{\tilde{h}_s^\ell = \sum_{t=1}^{T} \alpha_{st}\left(\mathbf{W}^V h_t^\ell\right)}$$

Attend to values to get weighted sum

Self-attention!

Self-Attention Toy Example

Compute pairwise scores

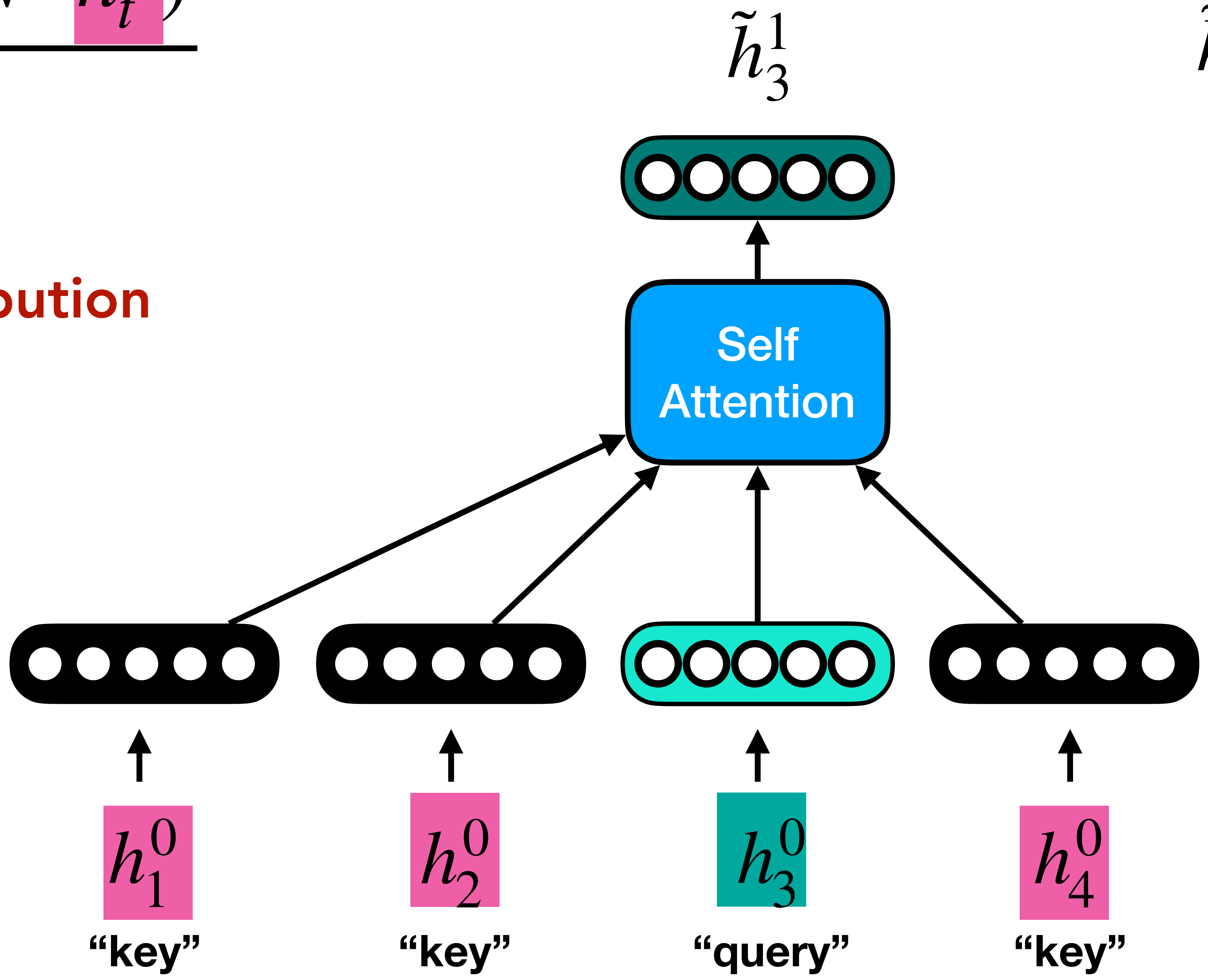
$$a_{st} = \frac{(\mathbf{W}^Q h_s^\ell)^T (\mathbf{W}^K h_t^\ell)}{\sqrt{d}}$$

Get attention distribution

$$\alpha_{st} = \frac{e^{a_{st}}}{\sum_j e^{a_{sj}}}$$

Attend to values to get weighted sum

$$\tilde{h}_s^\ell = \sum_{t=1}^{T} \alpha_{st}(\mathbf{W}^V h_t^\ell)$$



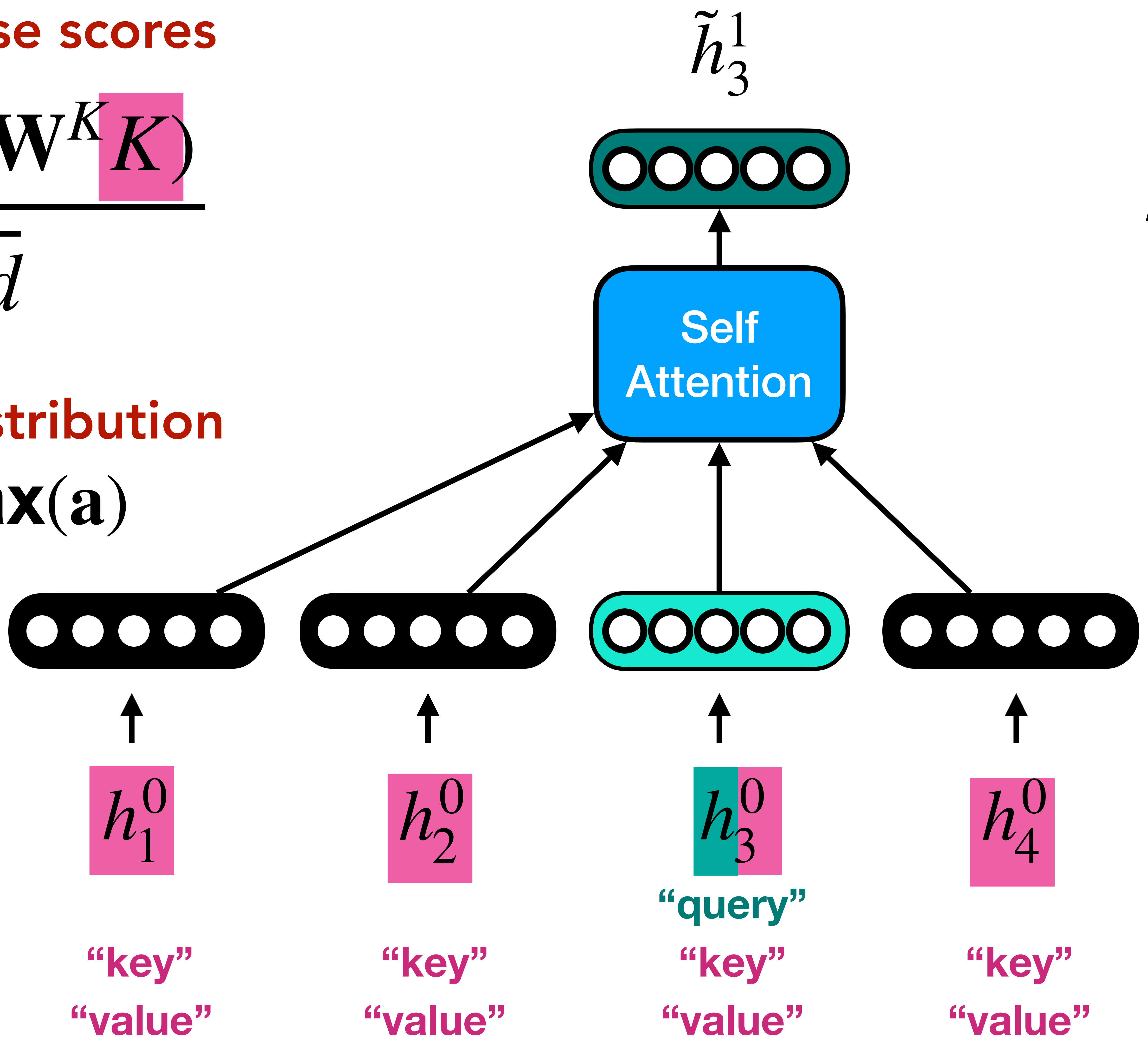
Self-Attention Toy Example

Compute pairwise scores

$$\mathbf{a} = \frac{(\mathbf{W}^Q q)(\mathbf{W}^K K)}{\sqrt{d}}$$

Get attention distribution

$$\alpha = \textbf{softmax}(\mathbf{a})$$



Attend to values to get weighted sum

$$\tilde{h}^\ell = W^O \alpha (V \mathbf{W}^V)$$

"query" $q = h_s^\ell$

"values"

$K = V = \{h_t^\ell\}_{t=0}^T$

"keys"

For each attention computation, every element is a key and value, and one element is a query

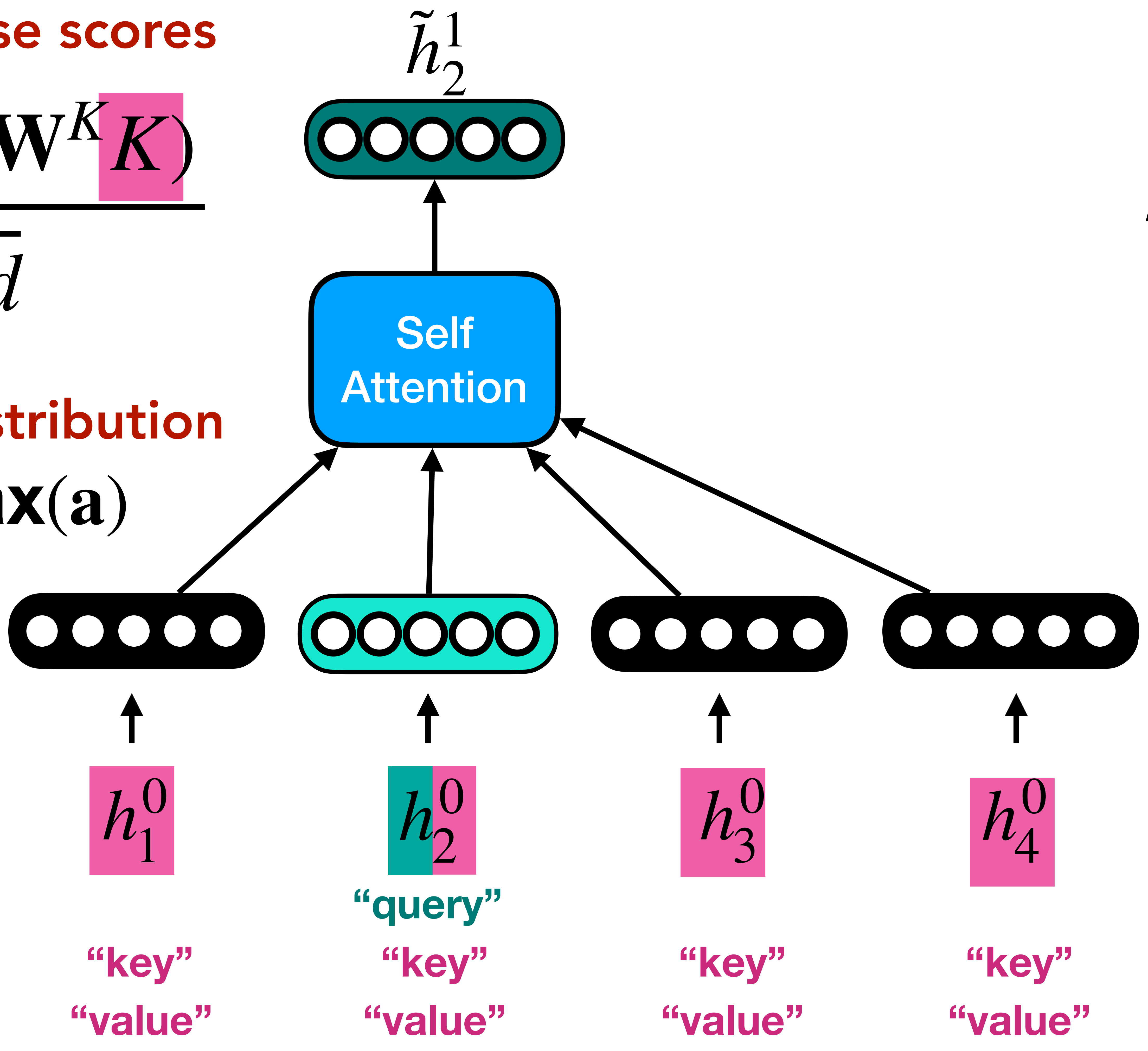
Self-Attention Toy Example

Compute pairwise scores

$$\mathbf{a} = \frac{(\mathbf{W}^Q q)(\mathbf{W}^K K)}{\sqrt{d}}$$

Get attention distribution

$$\alpha = \textbf{softmax}(\mathbf{a})$$



Attend to values to get weighted sum

$$\tilde{h}^\ell = W^O \alpha (V \mathbf{W}^V)$$

"query" $q = h_s^\ell$

"values"

$K = V = \{h_t^\ell\}_{t=0}^T$

"keys"

For each attention computation, every element is a key and value, and one element is a query

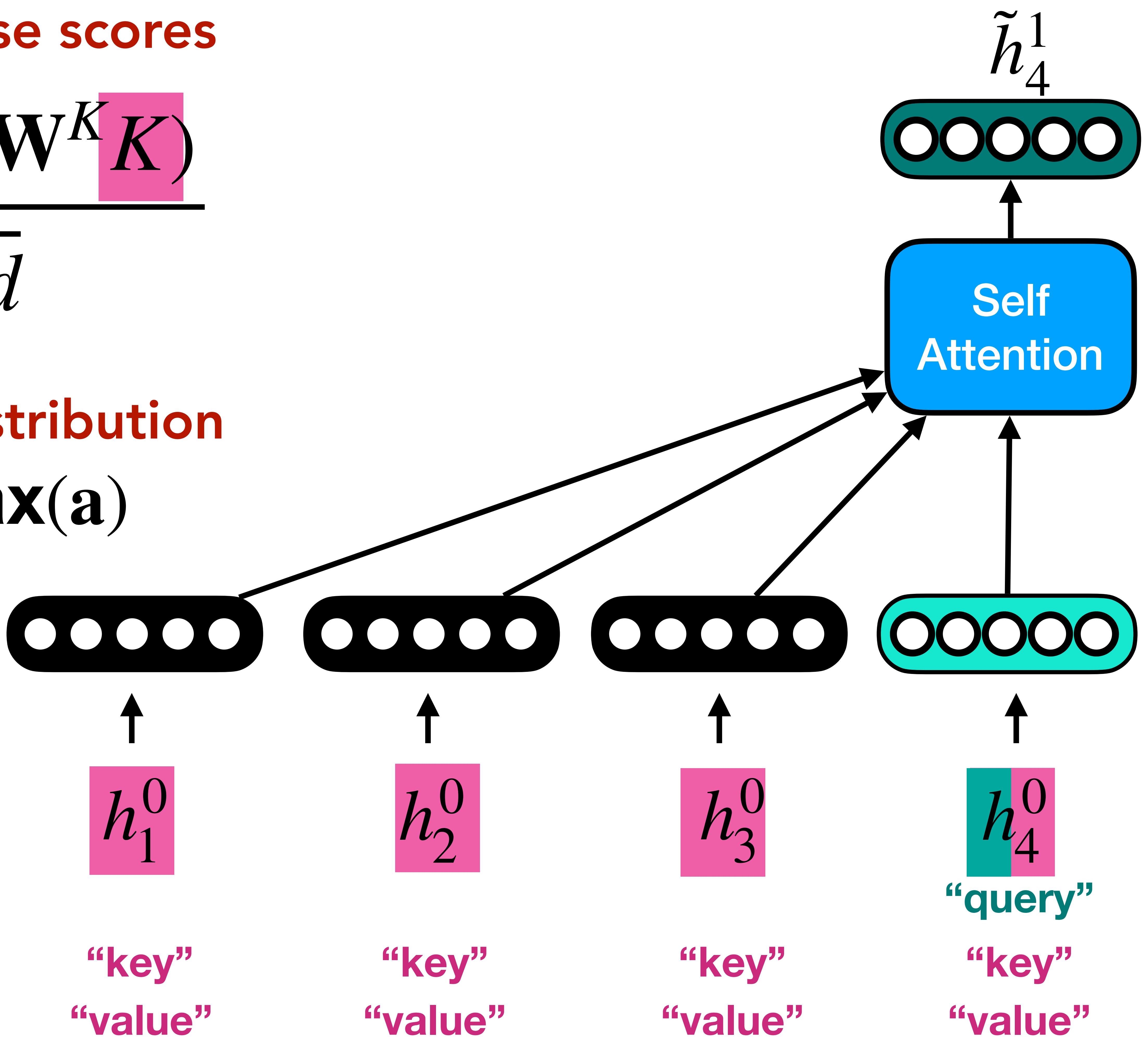
Self-Attention Toy Example

Compute pairwise scores

$$\mathbf{a} = \frac{(\mathbf{W}^Q q)(\mathbf{W}^K K)}{\sqrt{d}}$$

Get attention distribution

$$\alpha = \mathbf{softmax}(\mathbf{a})$$



Attend to values to get weighted sum

$$\tilde{h}^\ell = W^O \alpha(V\mathbf{W}^V)$$

"query" $q = h_s^\ell$

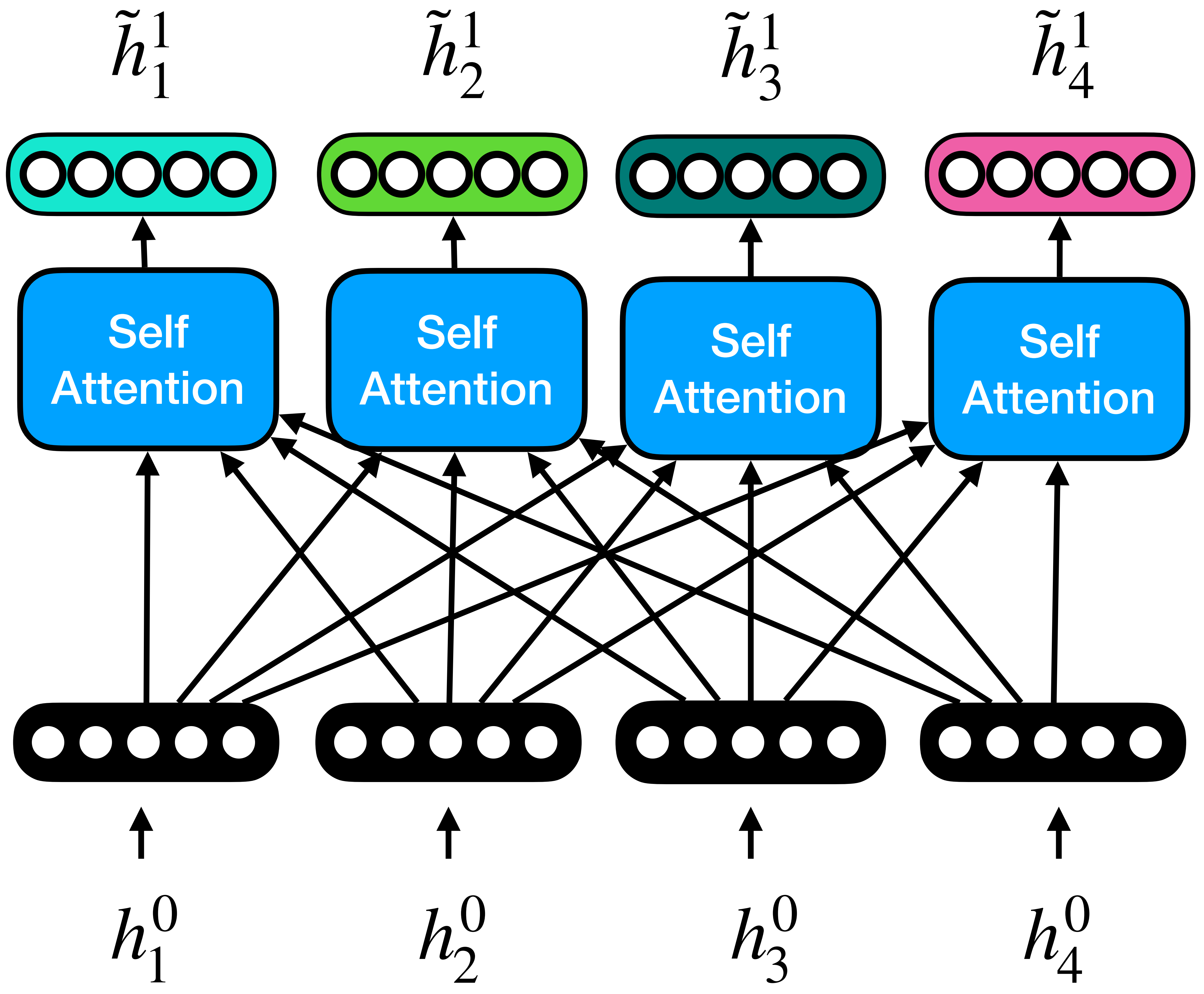
"values"

$K = V = \{h_t^\ell\}_{t=0}^T$

"keys"

For each attention computation, every element is a key and value, and one element is a query

Self-Attention Toy Example



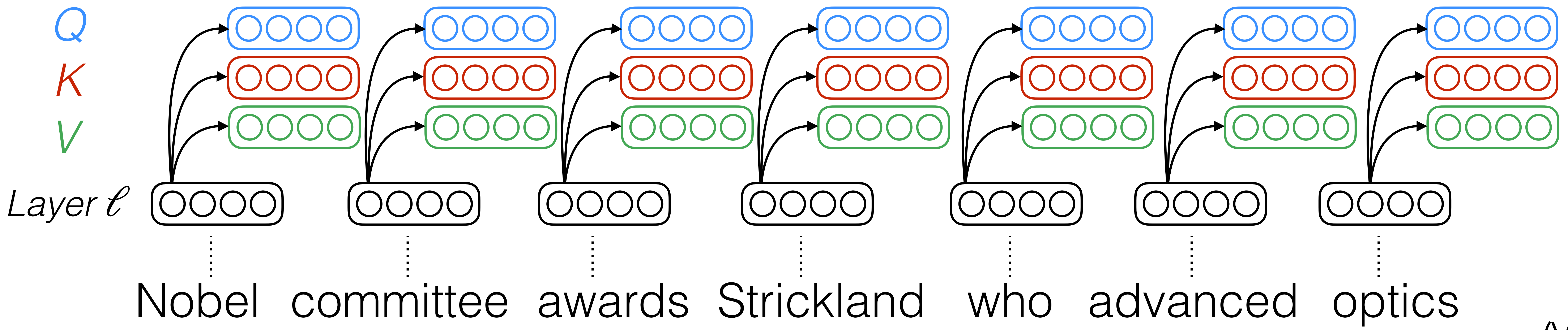
$$\tilde{h}_1^1 = \textbf{Attention}\left(h_1^0, \{h_t^0\}_{t=0}^{t=3}\right)$$

$$\tilde{h}_1^2 = \textbf{Attention}\left(h_2^0, \{h_t^0\}_{t=0}^{t=3}\right)$$

$$\tilde{h}_1^3 = \textbf{Attention}\left(h_3^0, \{h_t^0\}_{t=0}^{t=3}\right)$$

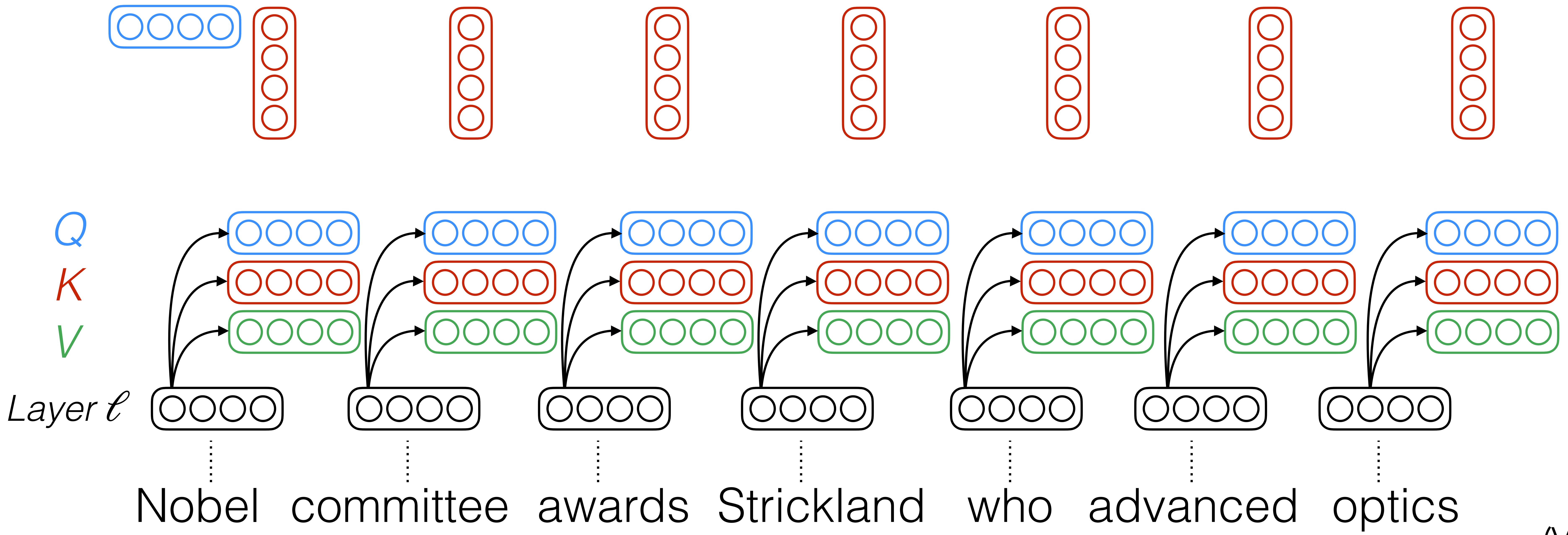
$$\tilde{h}_1^4 = \textbf{Attention}\left(h_4^0, \{h_t^0\}_{t=0}^{t=3}\right)$$

Self-attention (in encoder)



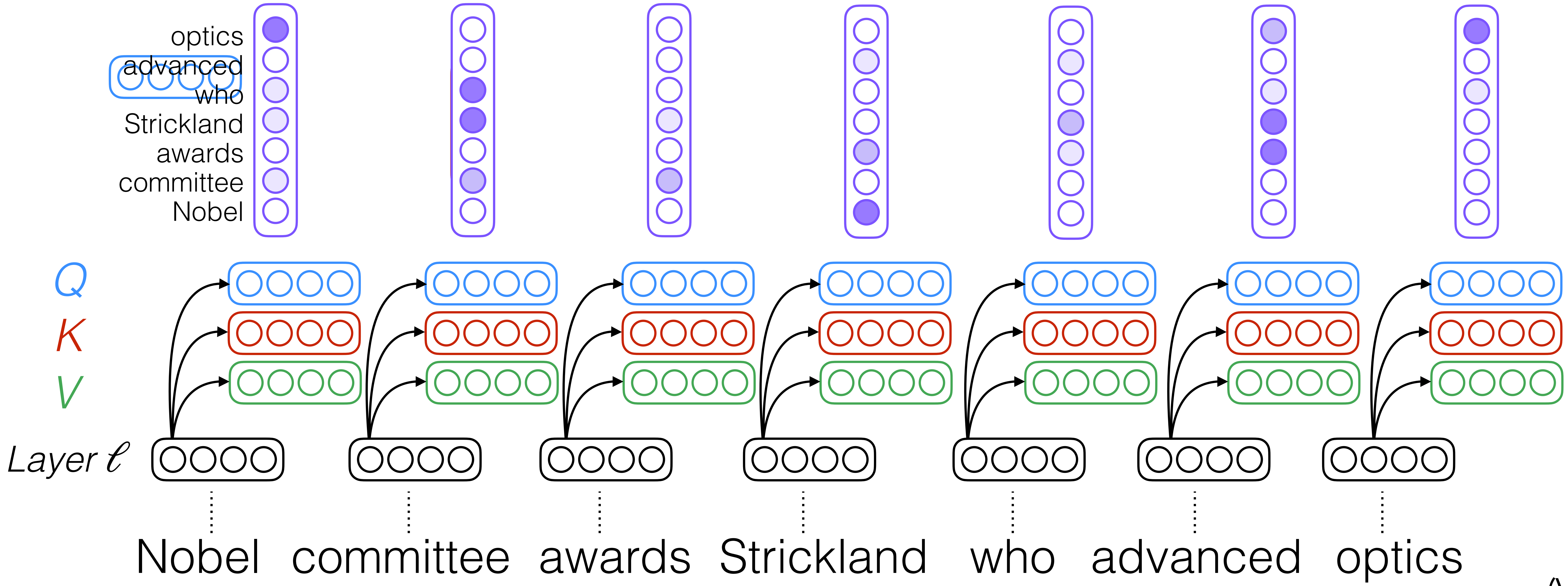
(Vaswani et al., 2017)

Self-attention (in encoder)



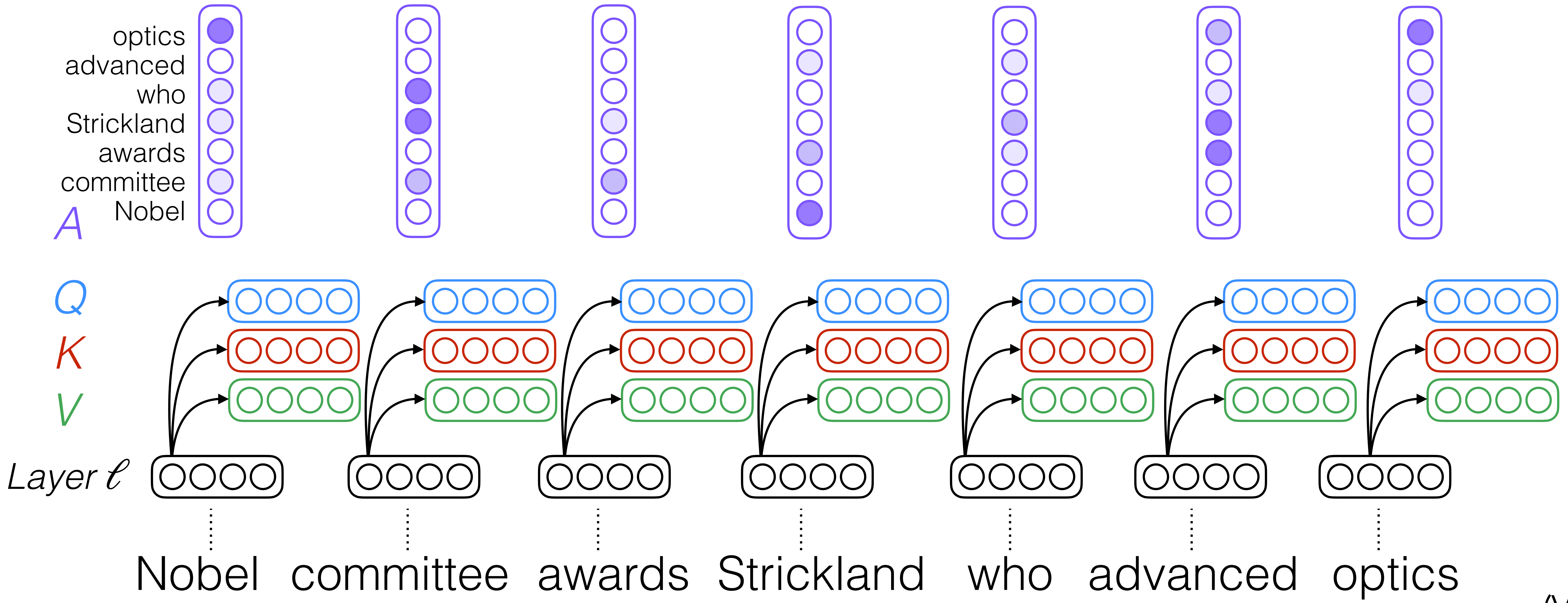
(Vaswani et al., 2017)

Self-attention (in encoder)



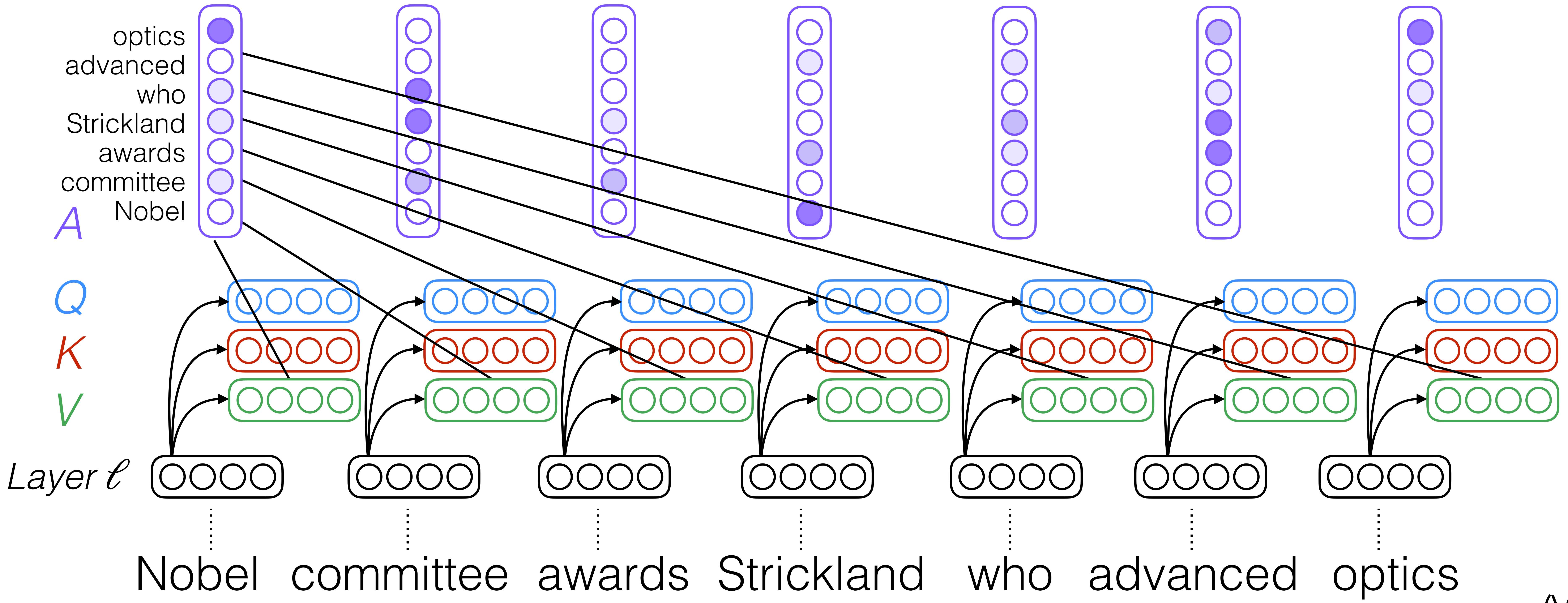
(Vaswani et al., 2017)

Self-attention (in encoder)



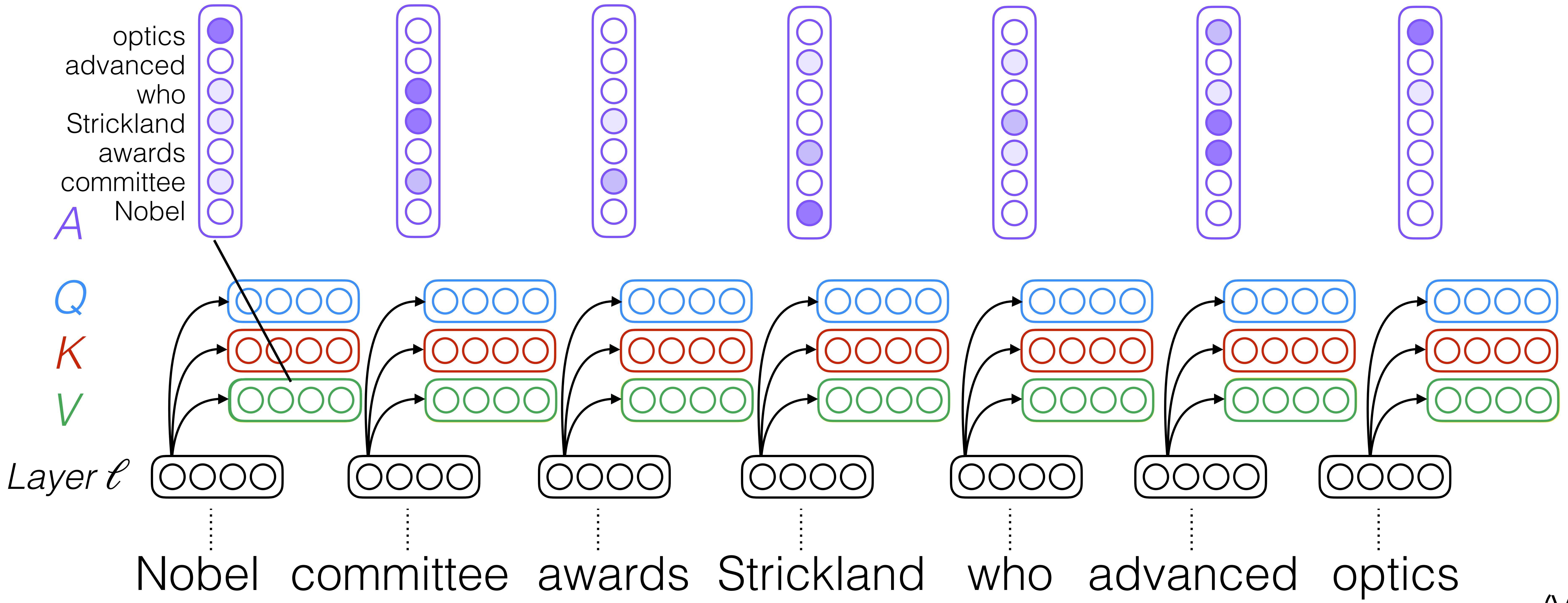
(Vaswani et al., 2017)

Self-attention (in encoder)



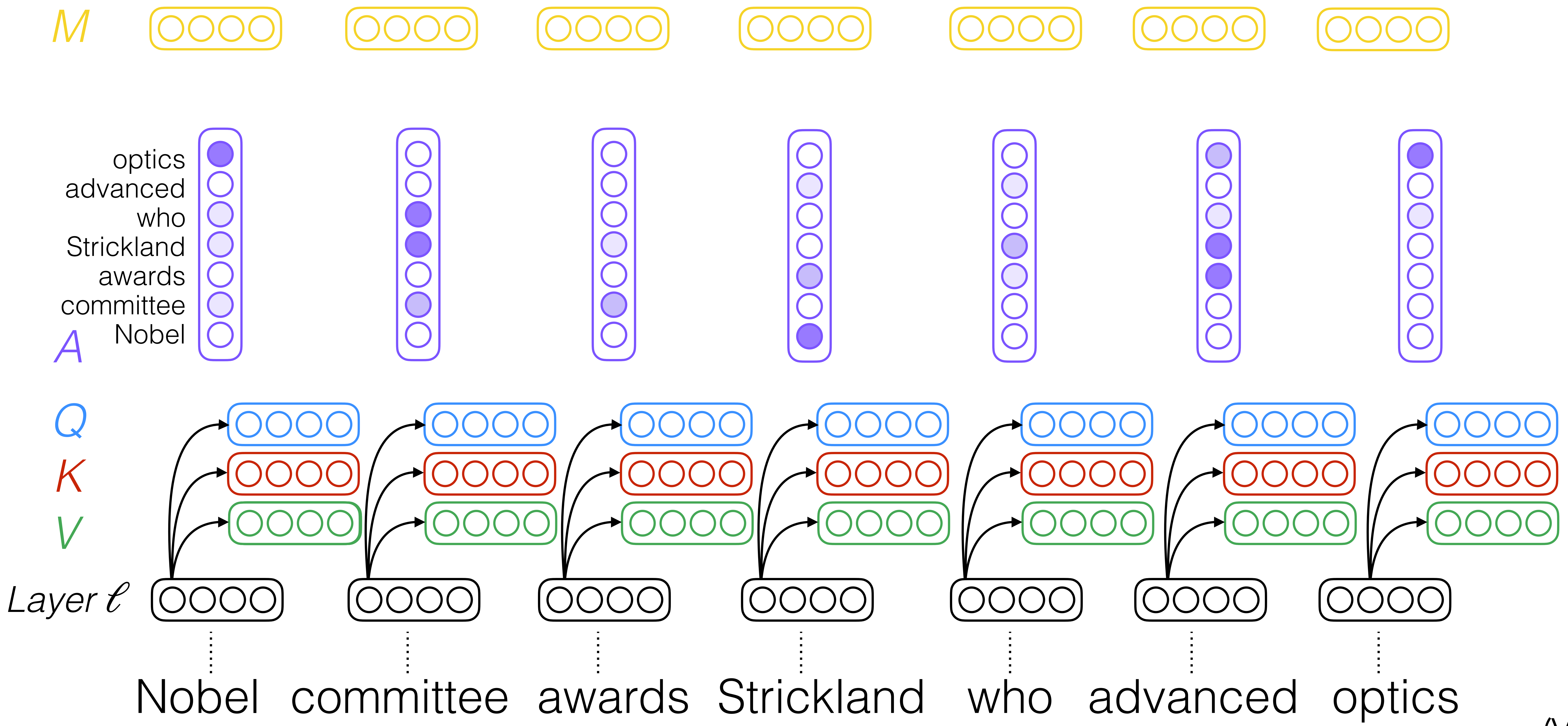
(Vaswani et al., 2017)

Self-attention (in encoder)



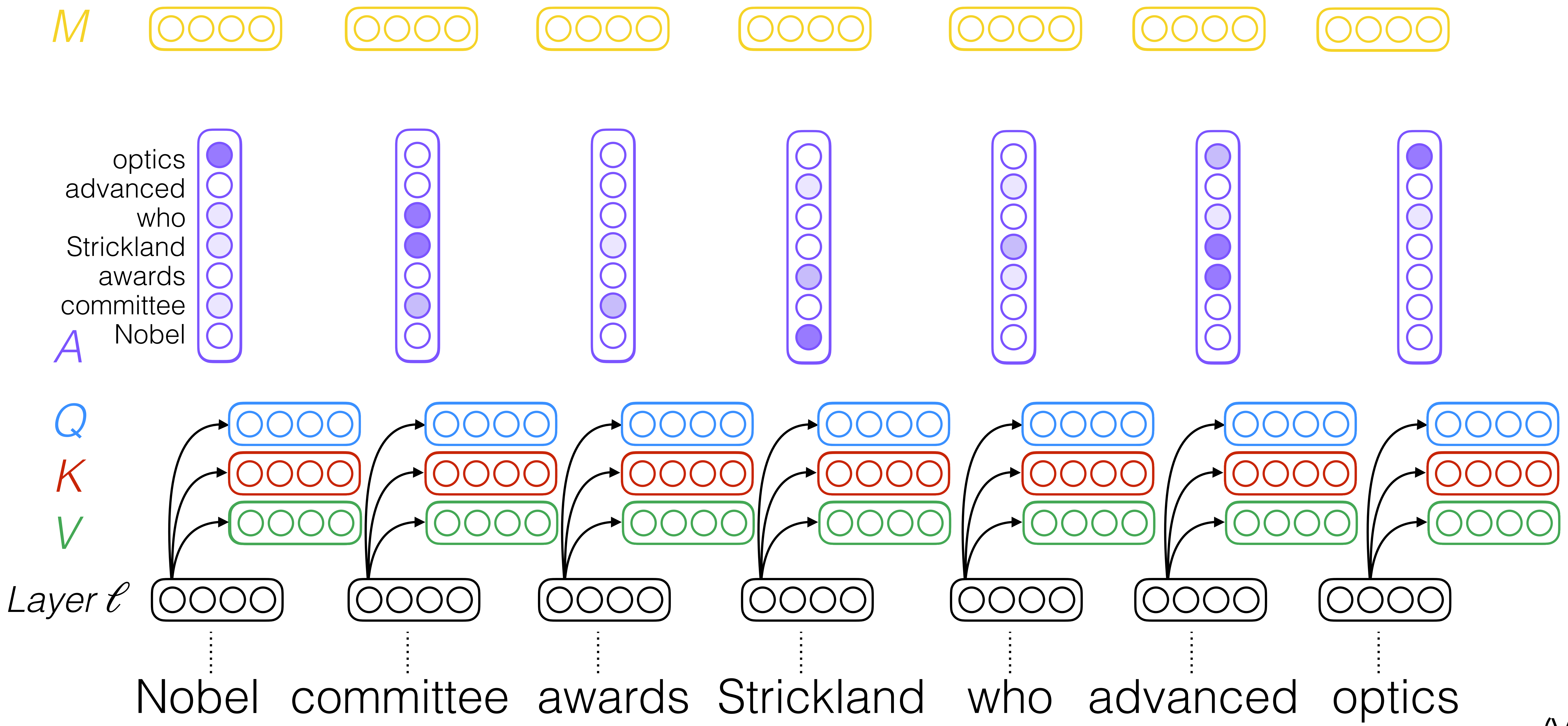
(Vaswani et al., 2017)

Self-attention (in encoder)



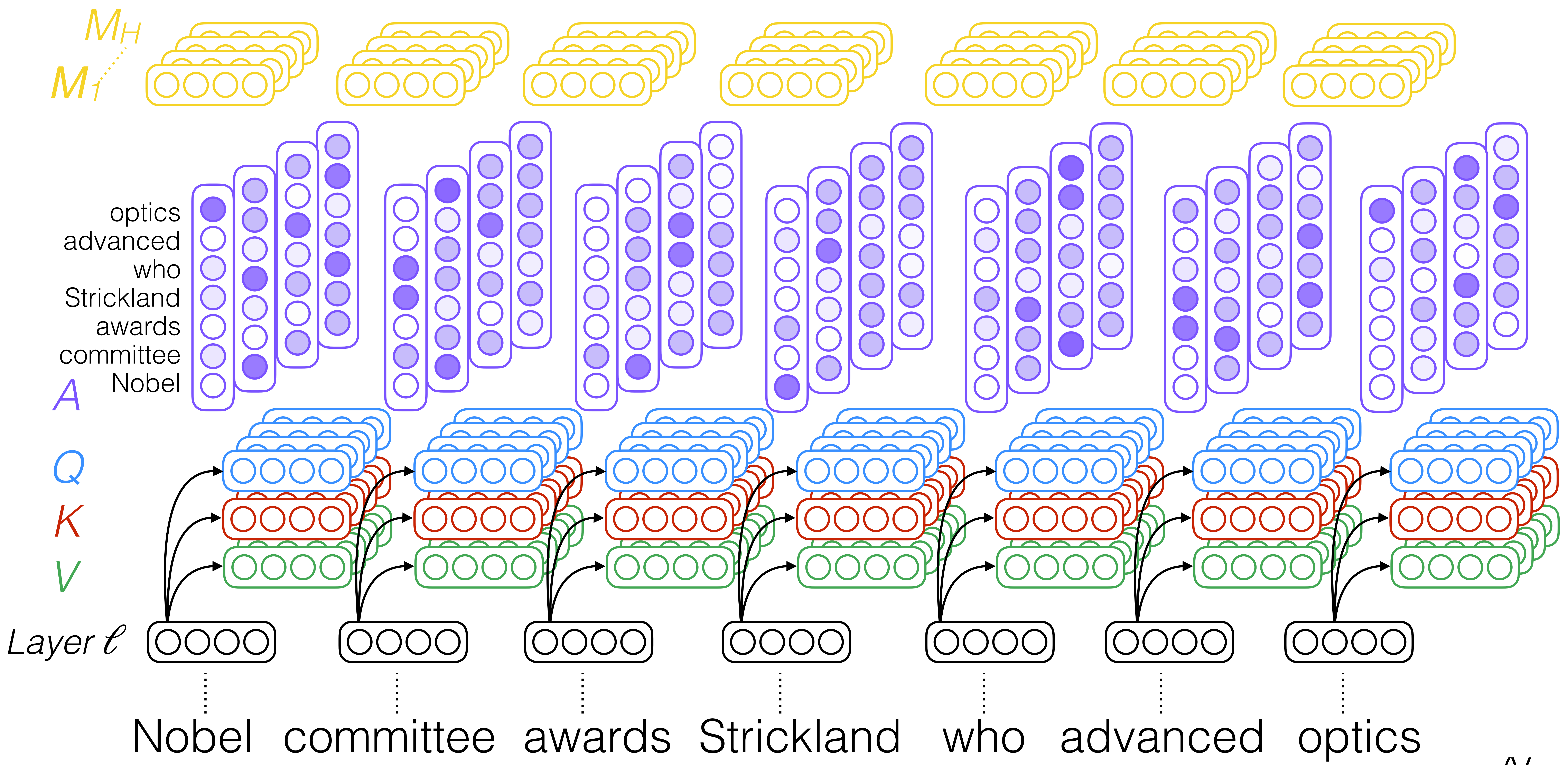
(Vaswani et al., 2017)

Self-attention (in encoder)



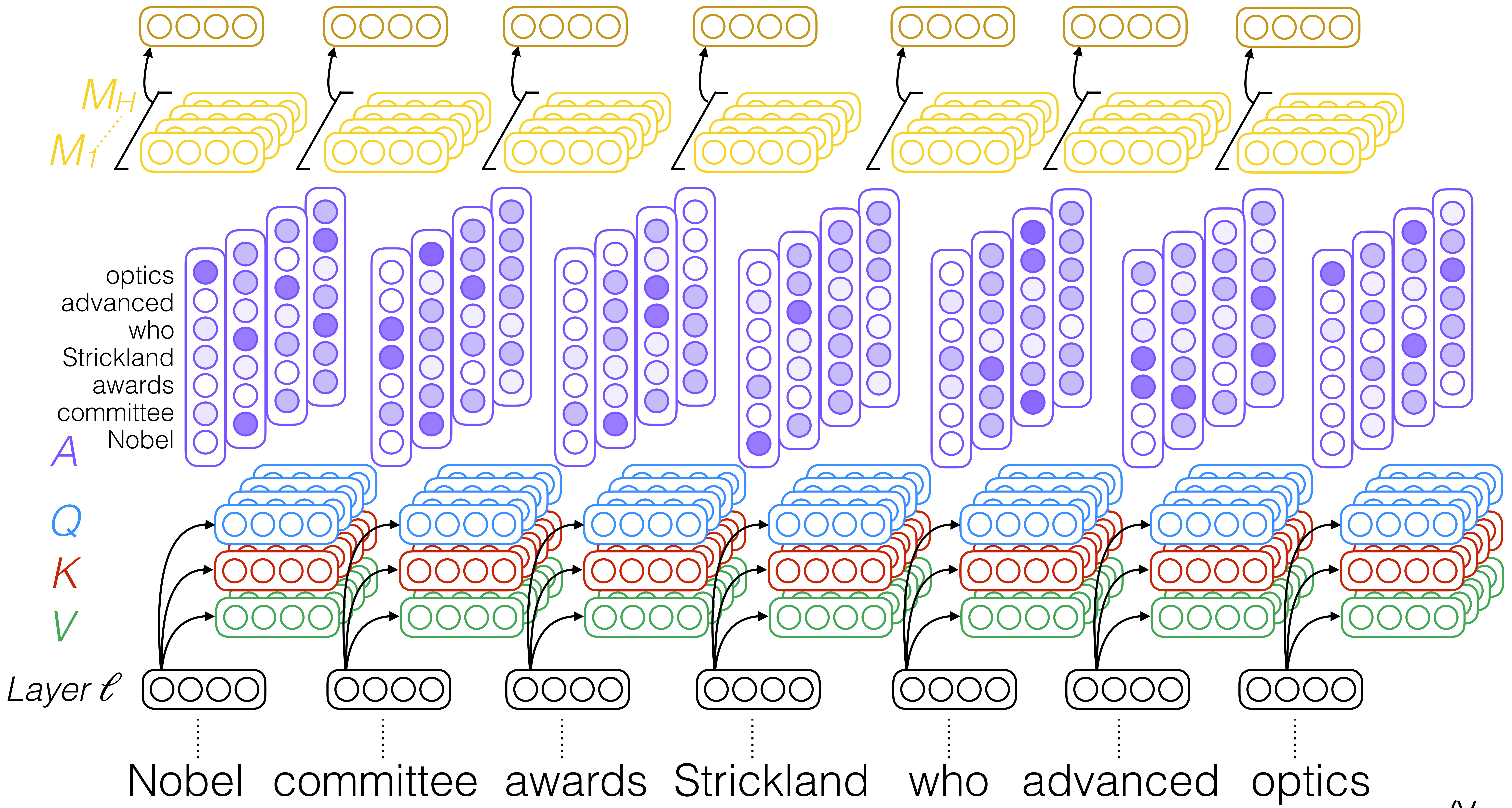
(Vaswani et al., 2017)

Multi-head self-attention



(Vaswani et al., 2017)

Multi-head self-attention



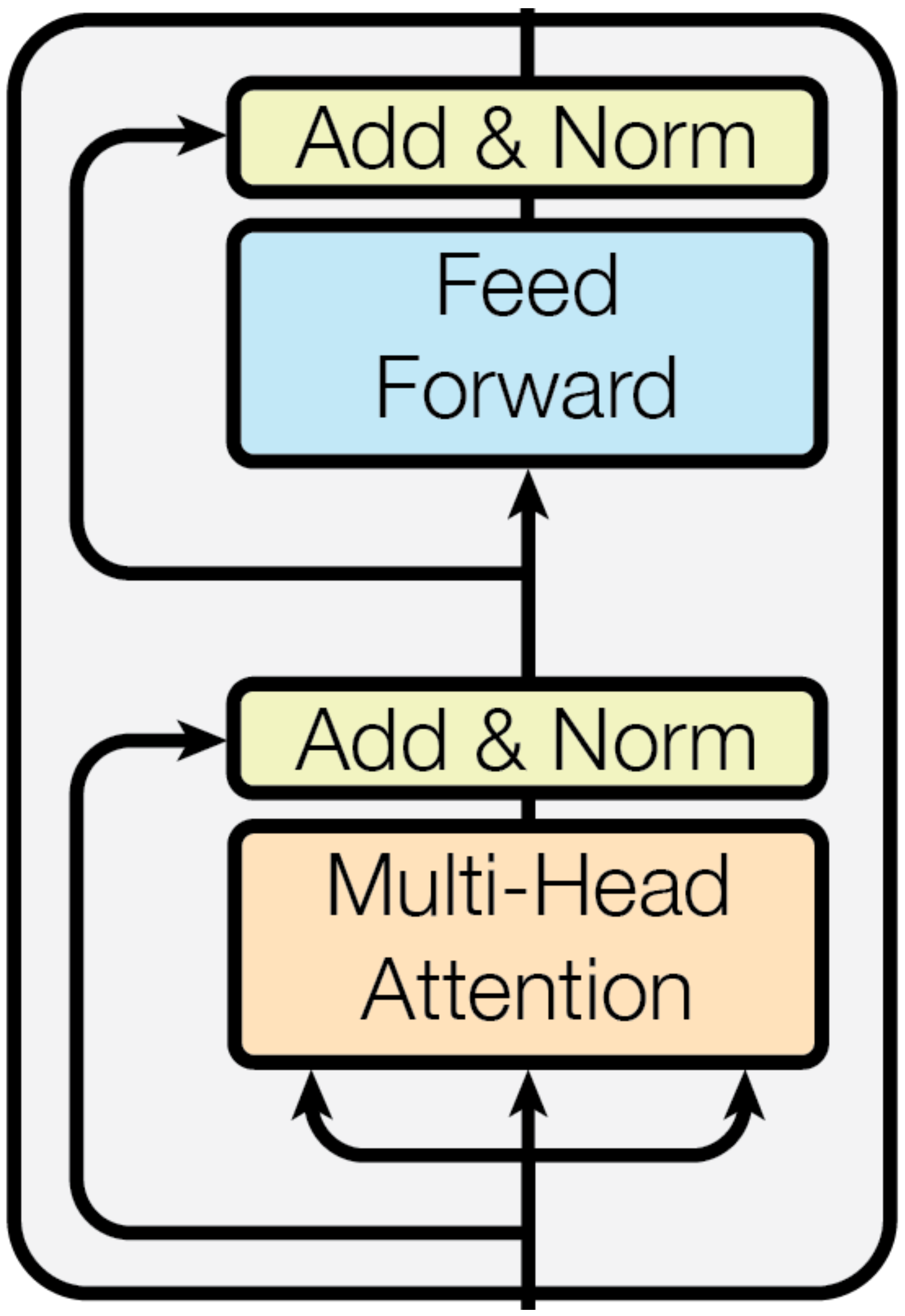
(Vaswani et al., 2017)

Question

What are two advantages of self-attention over recurrent models?

Transformer Block

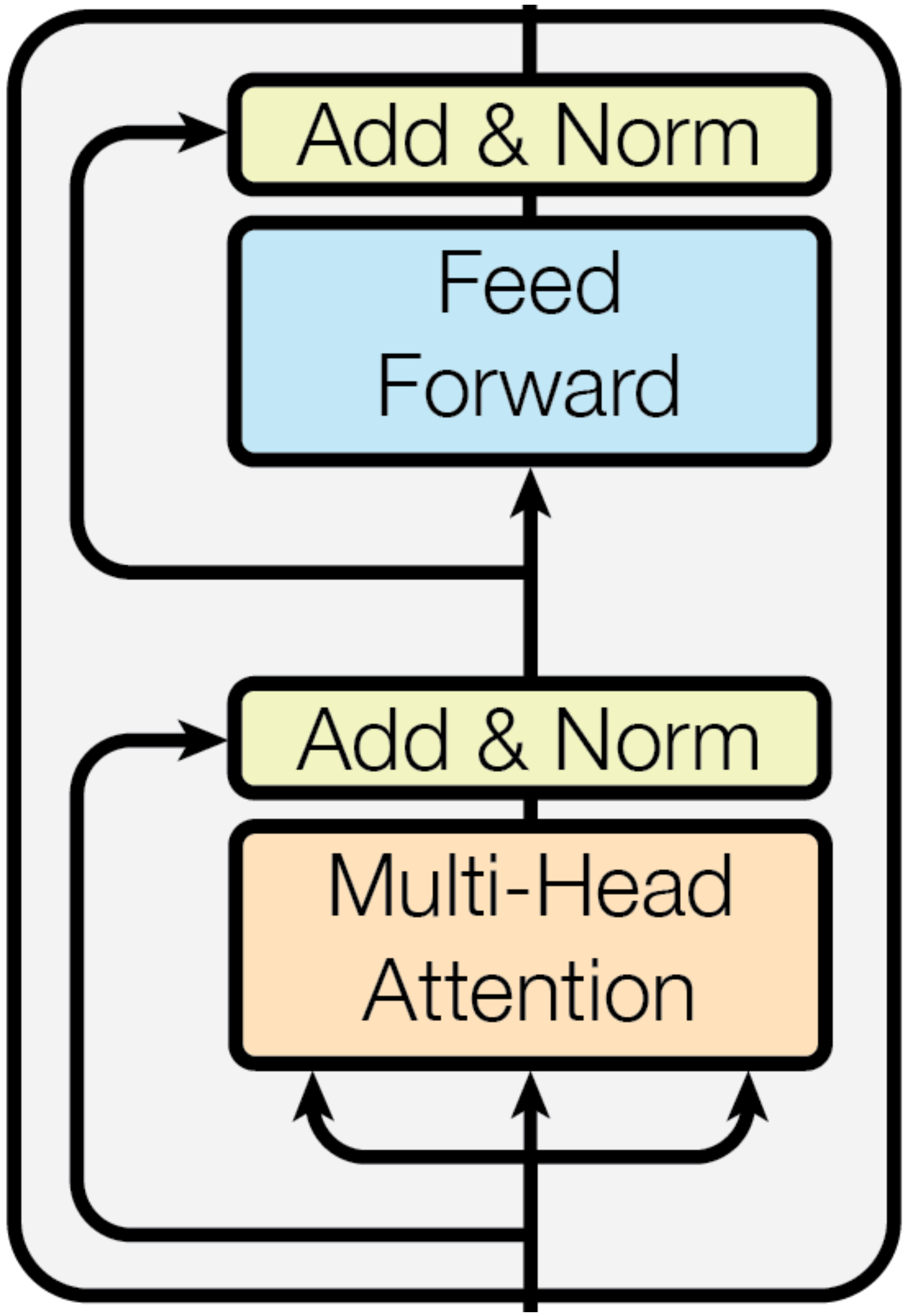
- Multi-headed attention is the main innovation of the transformer model!



Transformer Block

- Multi-headed attention is the main innovation of the transformer model!

- Each block also composed of:
  - a layer normalisations
  - a feedforward network
  - residual connections



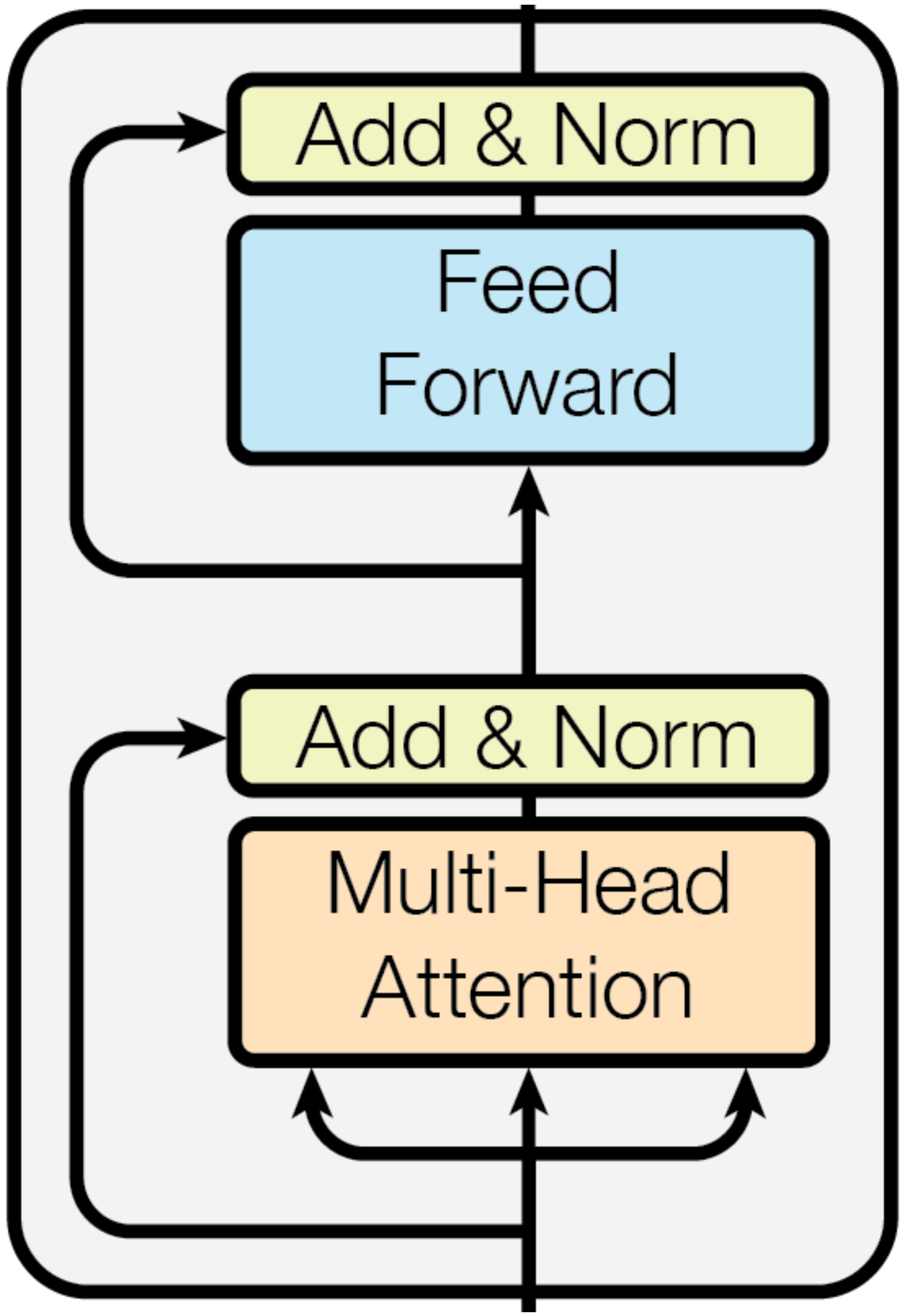
Vaswani et al., 2017

LayerNorm & Residual Connections

- **Layer Normalisation**
  - Normalize the outputs of different modules

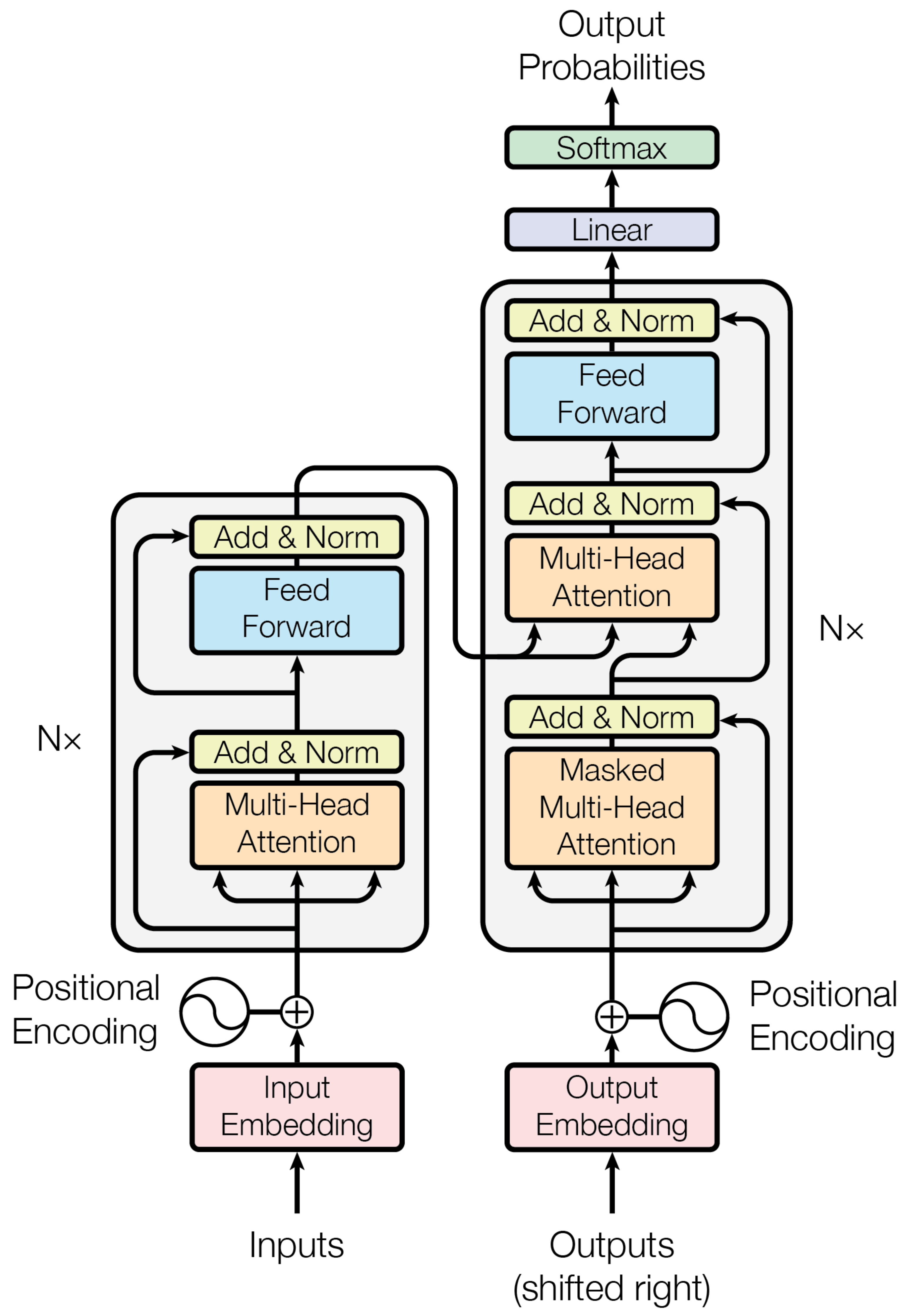
$$\boxed{y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta}$$

- **Residual Connections**
  - Add the input of a module to its output
  - $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$



Full Transformer

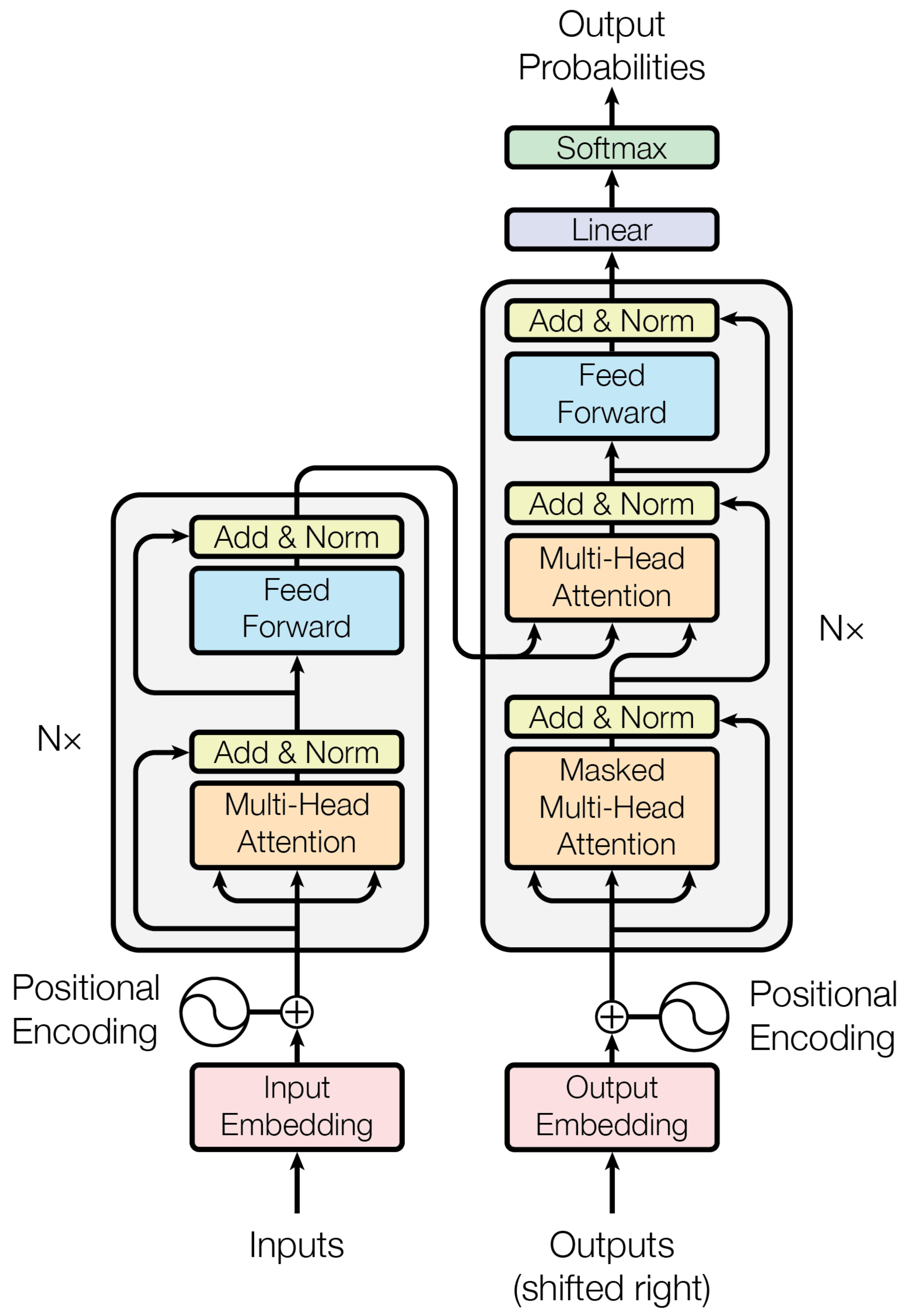
- Full transformer encoder is multiple cascaded transformer blocks
  - **build up compositional representations of inputs**



Vaswani et al., 2017

Full Transformer

- Full transformer encoder is multiple cascaded transformer blocks
  - **build up compositional representations of inputs**
- Transformer decoder (right) similar to encoder
  - First layer of block is **masked** multi-headed attention
  - Second layer is multi-headed attention over *final-layer* encoder outputs **(cross-attention)**
  - Third layer is feed-forward network



Vaswani et al., 2017

Question

What is an issue with self-attention for the decoder?

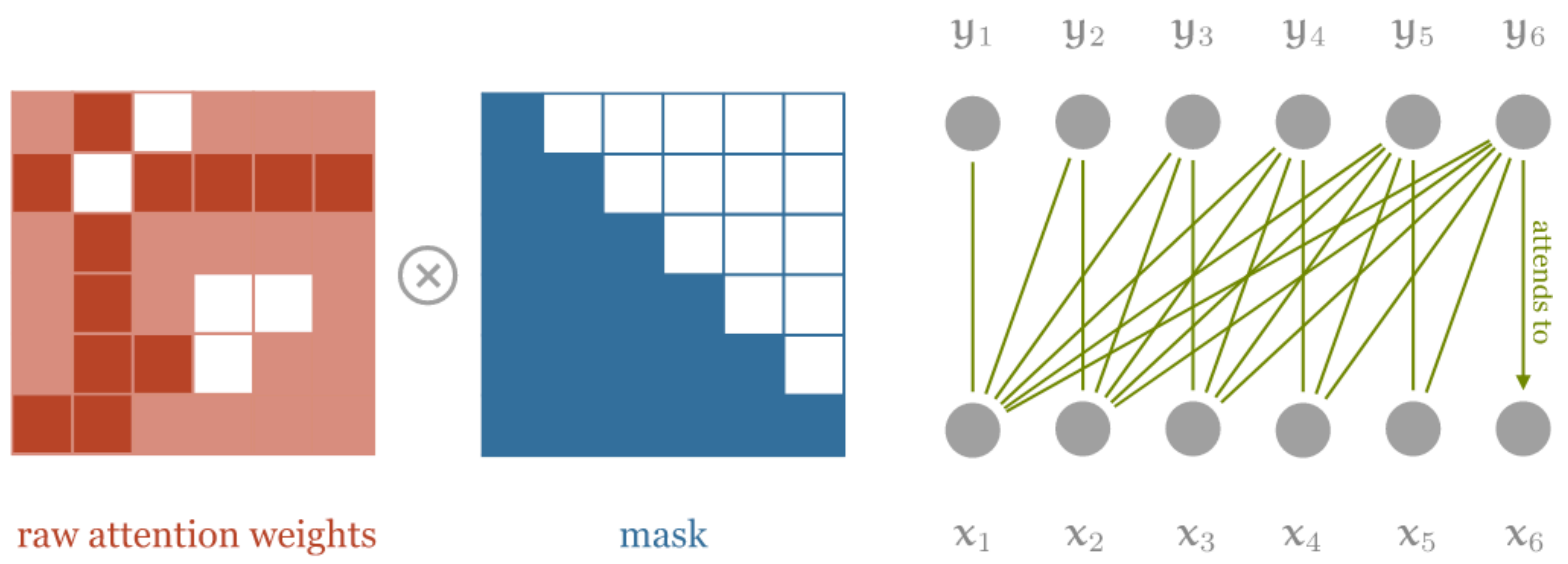
Masked Multi-headed Attention

- Self-attention can attend to any token in the sequence
- For the decoder, **you don't want tokens to attend to future tokens**
  - Decoder used to generate text (i.e., machine translation)

Masked Multi-headed Attention

- Self-attention can attend to any token in the sequence
- For the decoder, **you don't want tokens to attend to future tokens**
  - Decoder used to generate text (i.e., machine translation)

Mask the attention scores of future tokens so their attention = 0


raw attention weights
mask
y_1 y_2 y_3 y_4 y_5 y_6
attends to
x_1 x_2 x_3 x_4 x_5 x_6

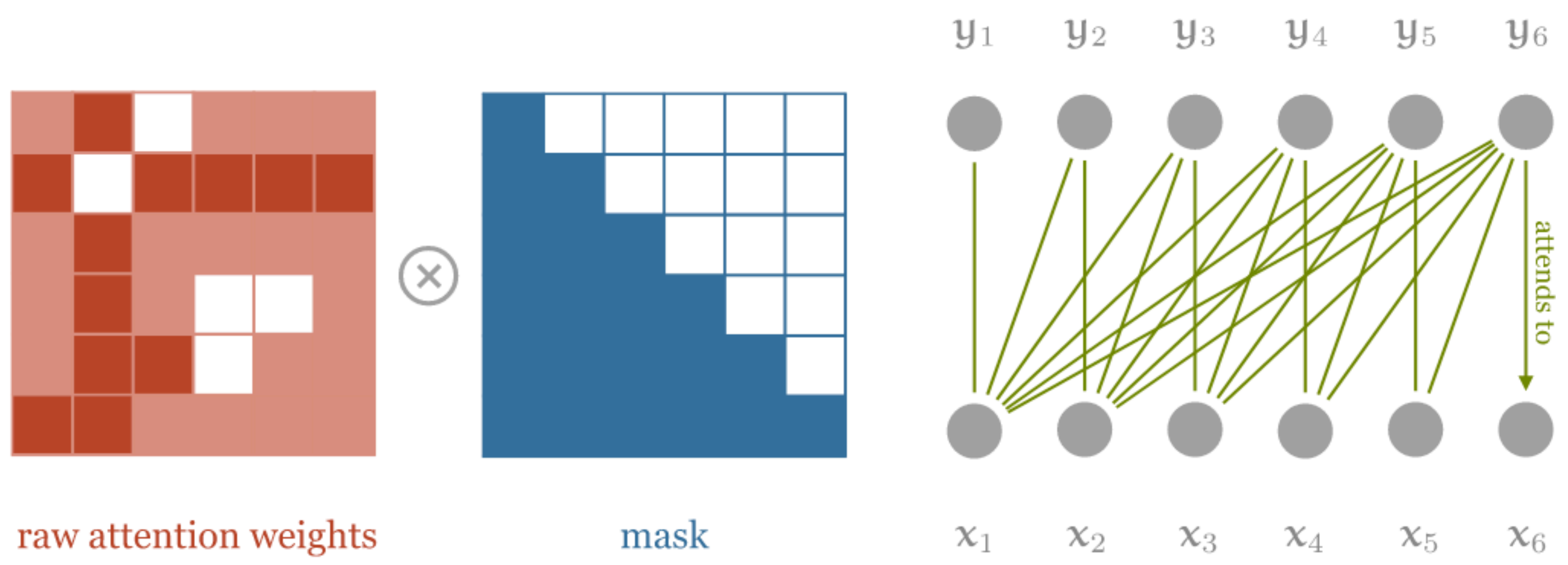
$$a_{st} = \frac{(\mathbf{W}^Q h_s^\ell)^T (\mathbf{W}^K h_t^\ell)}{\sqrt{d}} \quad \Rightarrow \quad a_{st} := a_{st} - \infty \; ; \; s < t$$

Masked Multi-headed Attention

- Self-attention can attend to any token in the sequence

- For the decoder, **you don't want tokens to attend to future tokens**
  - Decoder used to generate text (i.e., machine translation)

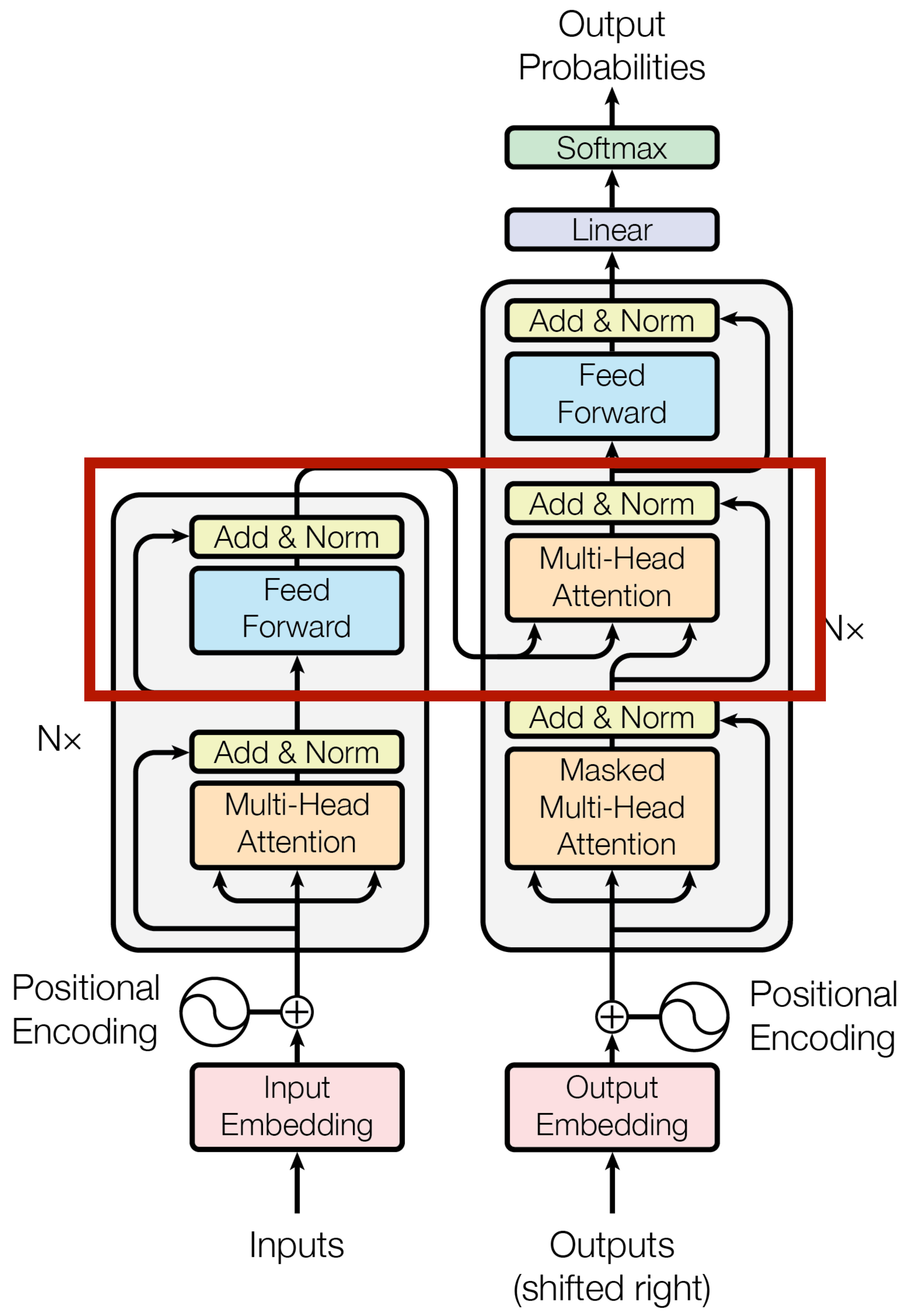
Mask the attention scores of future tokens so their attention = 0



$$a_{st} = \frac{(\mathbf{W}^Q h_s^\ell)^T (\mathbf{W}^K h_t^\ell)}{\sqrt{d}} \quad \Rightarrow \quad a_{st} := a_{st} - \infty \; ; \; s < t \quad \Rightarrow \quad \alpha_{st} = \frac{e^{a_{st}}}{\sum_j e^{a_{sj}}} = 0$$

Cross-attention

- **Cross attention** is the same classical attention as in the RNN encoder-decoder model
- The query to the attention function is the output of the masked multi-headed attention in the decoder (i.e., a decoder state)
- The keys and values are the output of the **final** encoder transformer
- Once again, a representation from the decoder is used to **attend** to the encoder outputs



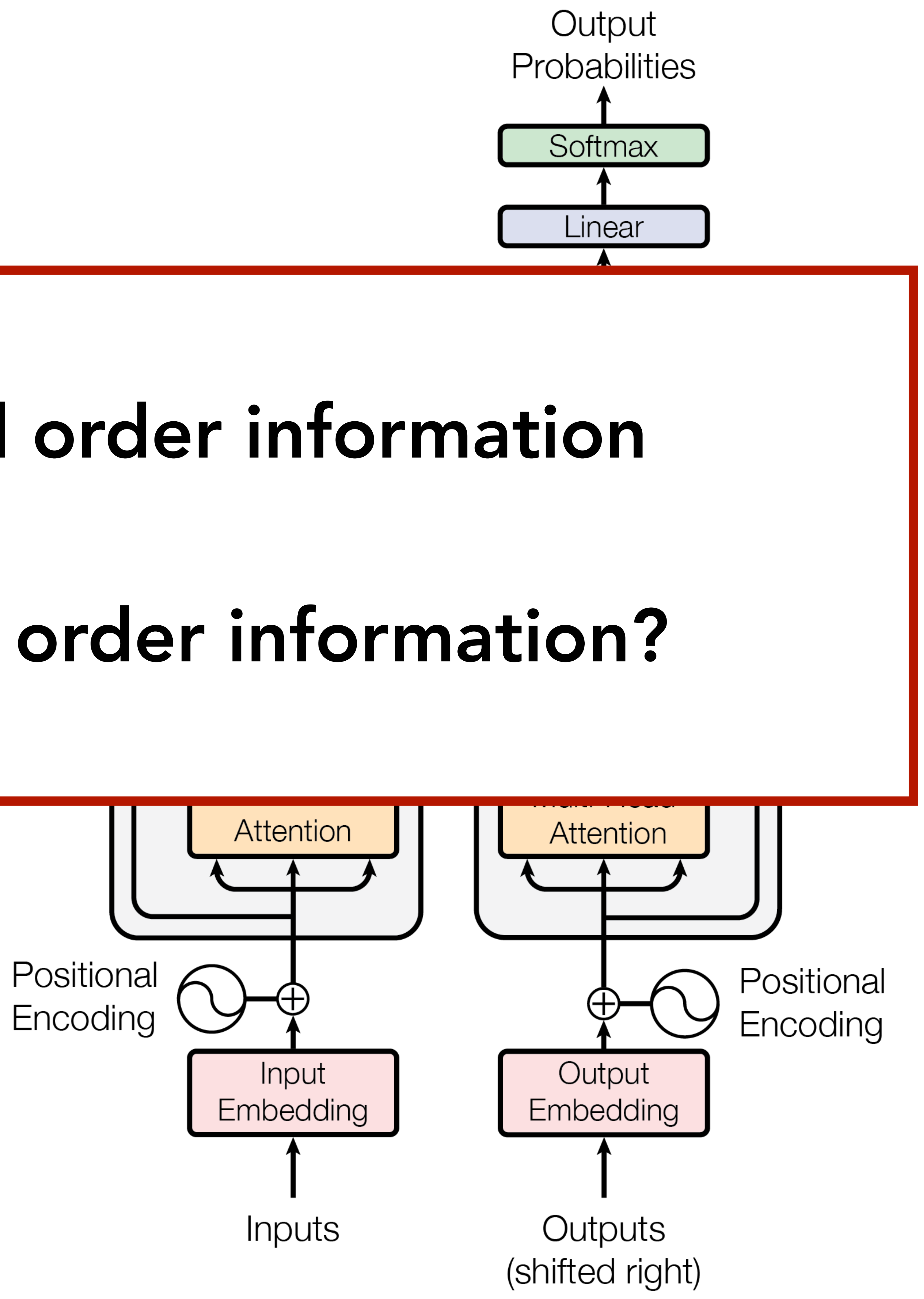
Vaswani et al., 2017

Full Transformer

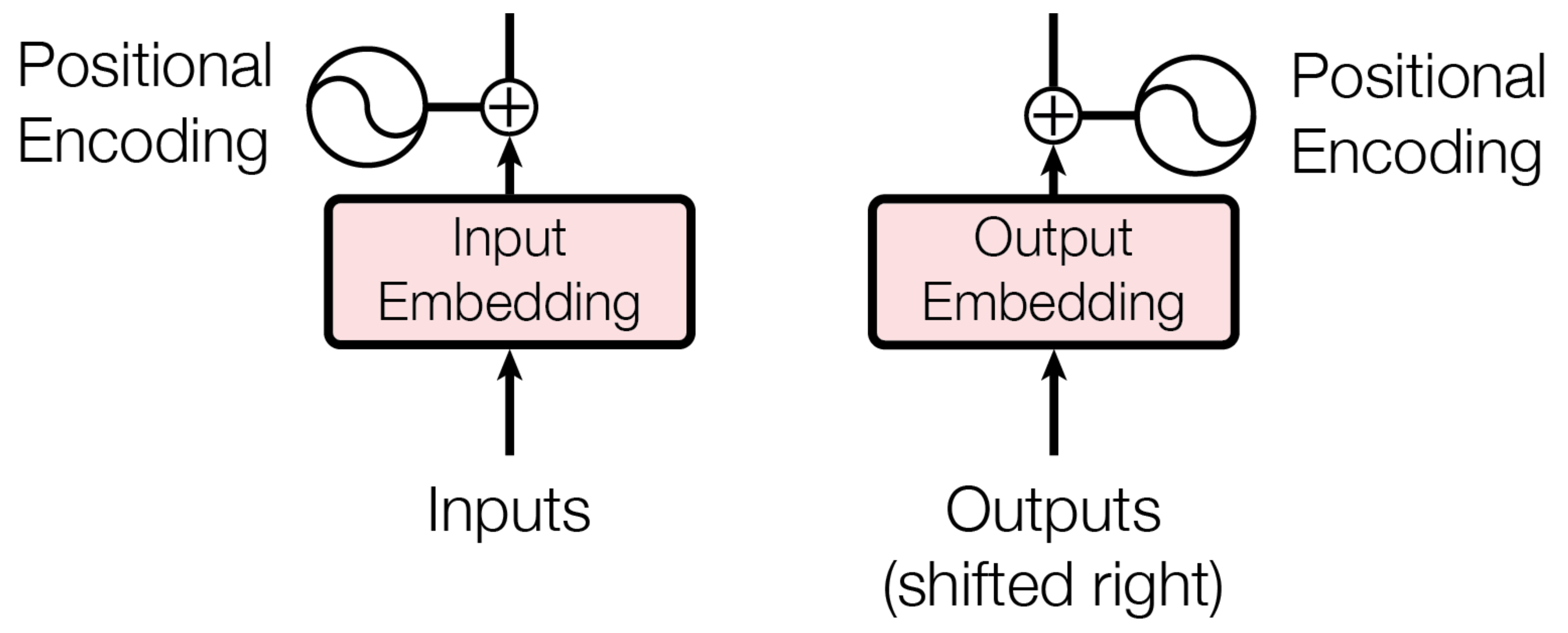
- Full transformer encoder is multiple cascaded transformer blocks
  -
- Tr
  -
  - Second layer is multi-headed attention over encoder outputs **(cross-attention)**
  - Third layer is feed-forward network

Recurrent models provided word order information

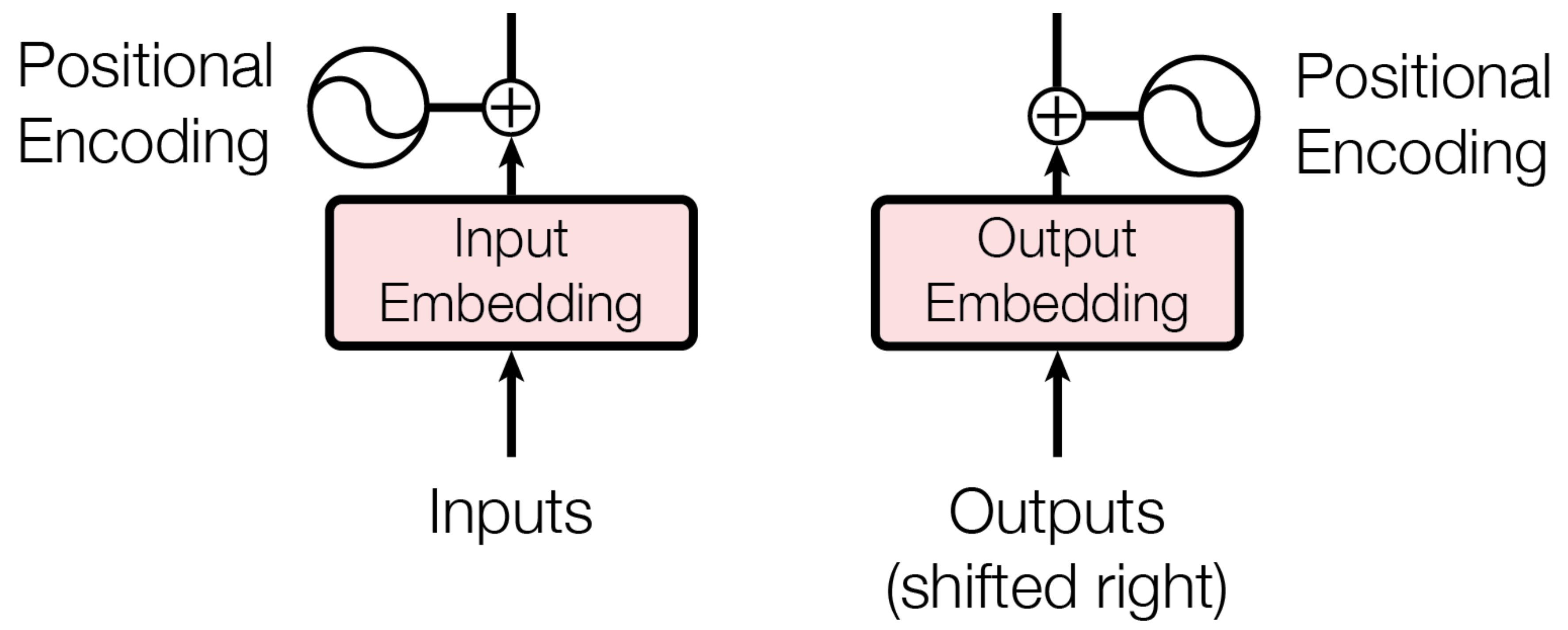
Does self-attention provide word order information?



Position Embeddings

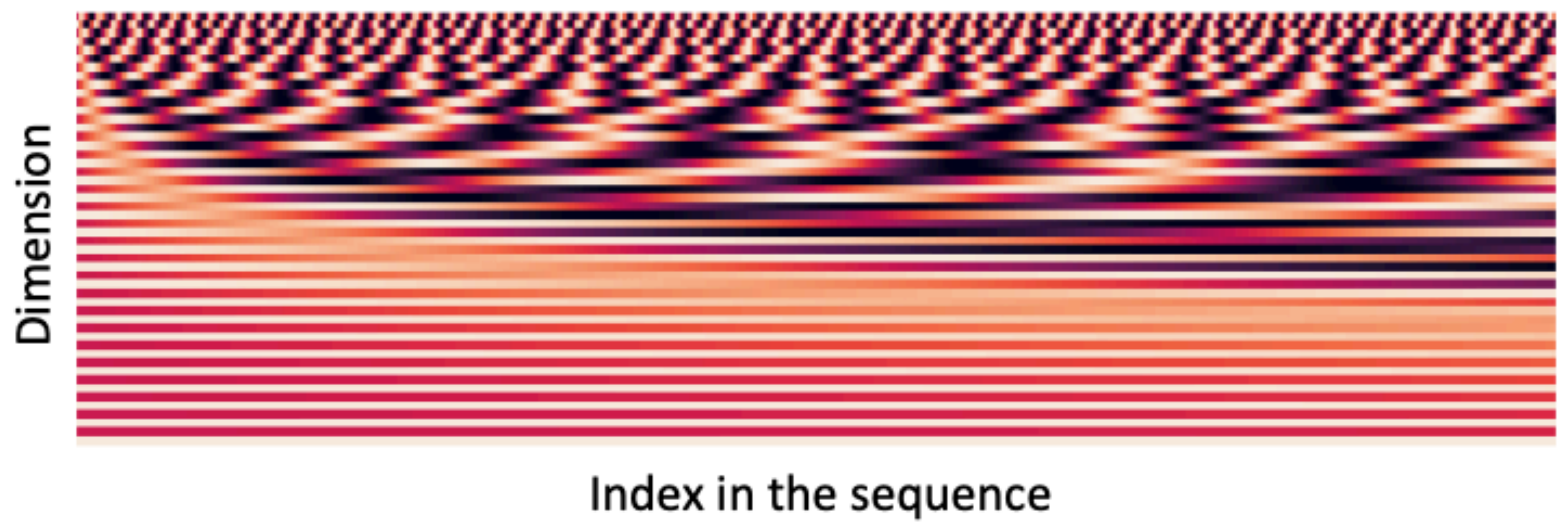


Position Embeddings

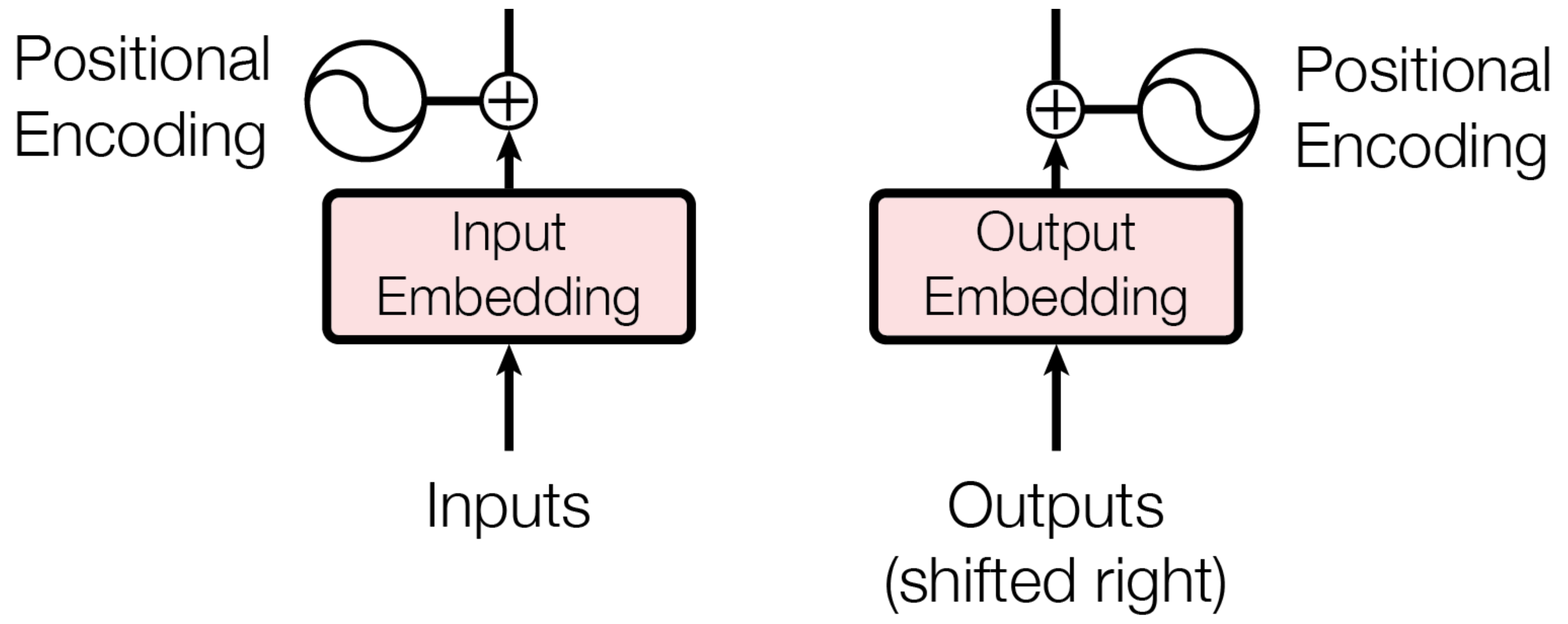


- Early position embeddings encoded a sinusoid function that was offset by a phase shift proportional to sequence position

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$

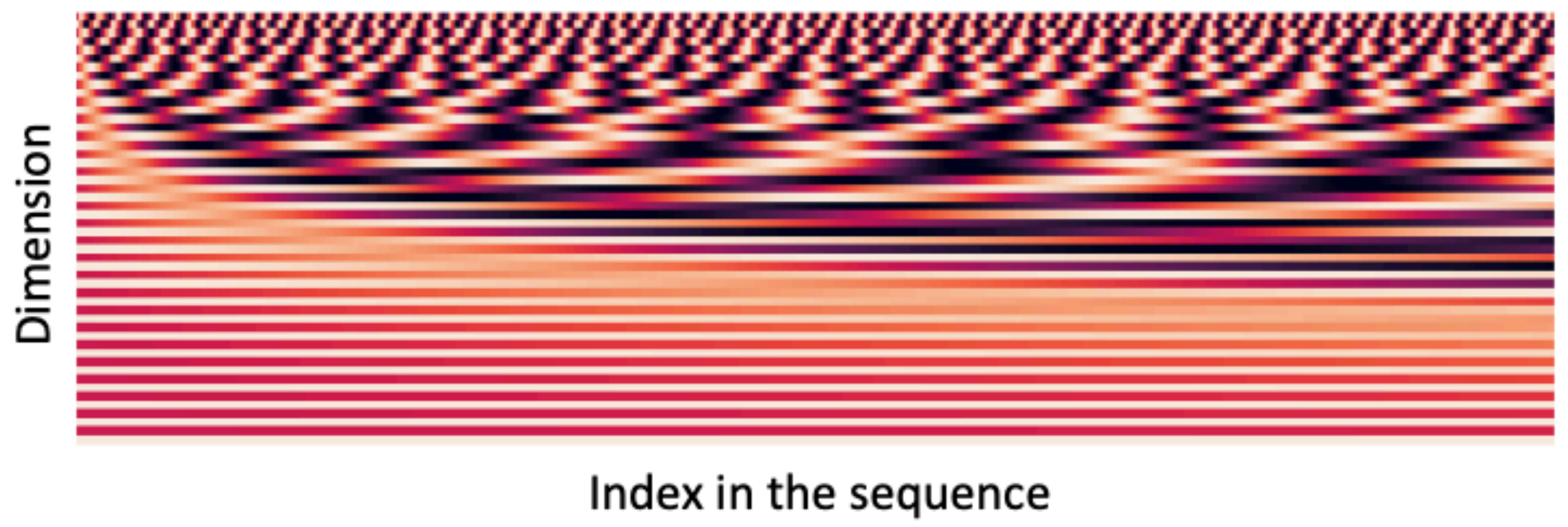


Position Embeddings



- Early position embeddings encoded a sinusoid function that was offset by a phase shift proportional to sequence position
- **In practice, easiest is to learn position embeddings from scratch**

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



Question

What might be a disadvantage of using learned position embeddings?

Poor generalisation to sequences longer than the maximum position embedding you have learned

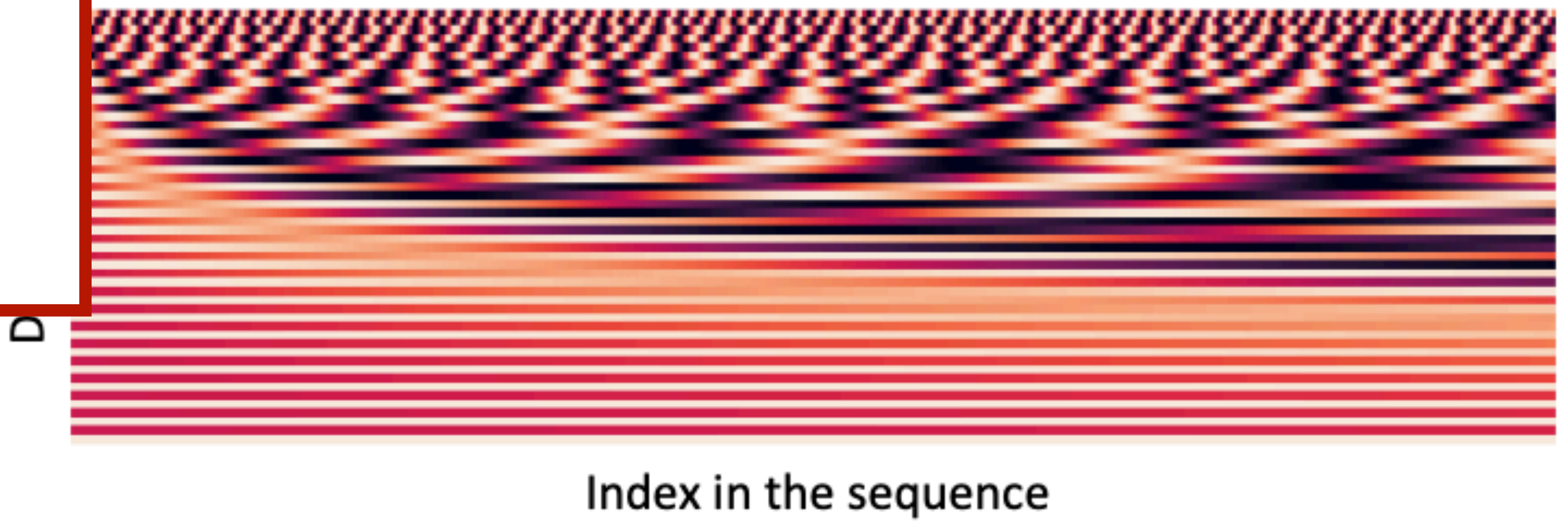
Position Embeddings

Lots of potential for new methods that generalise to longer sequences

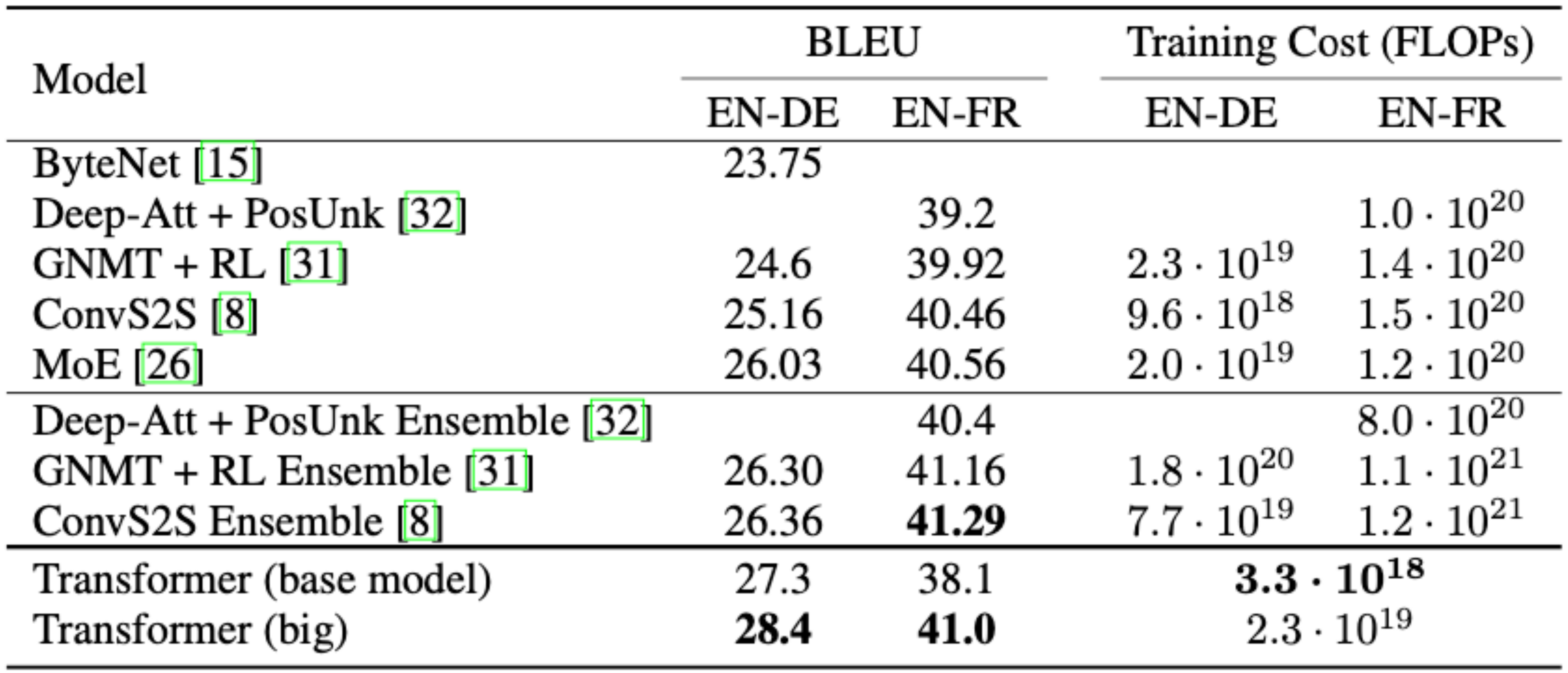
Position embeddings remain an active area of research

- Early position embeddings encoded a sinusoid function that was offset by a phase shift proportional to sequence position
- **In practice, easiest is to learn position embeddings from scratch**

$$\begin{pmatrix} \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



Performance: Machine Translation

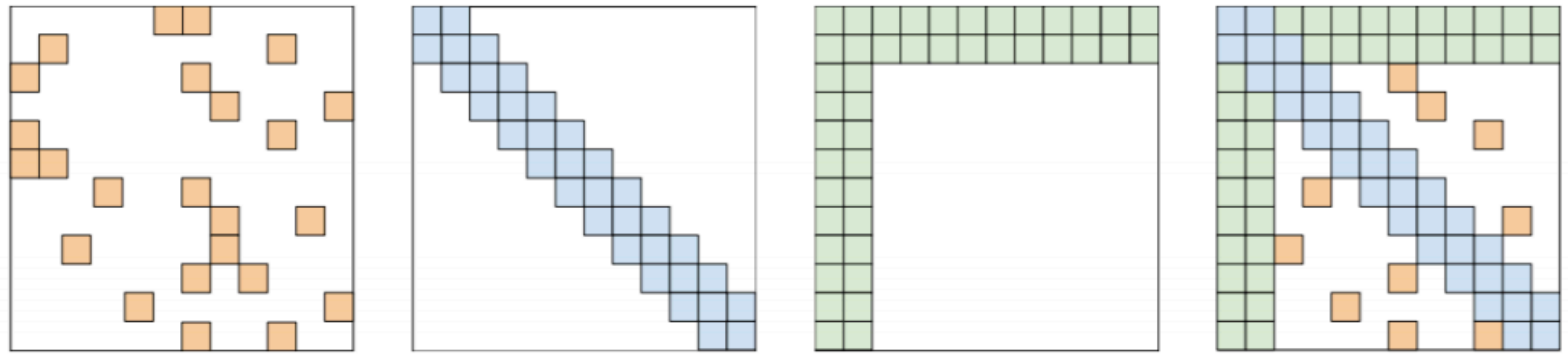


Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

(Vaswani et al., 2017)

Question

What could be a disadvantage of transformers over RNNs?



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Other Resources of Interest

- The Annotated Transformer
  - https://nlp.seas.harvard.edu/2018/04/03/attention.html
- The Illustrated Transformer
  - https://jalammar.github.io/illustrated-transformer/
- Only basics presented here today! Many modifications to initial transformers exist

Recap

- **Temporal Bottleneck**: **Vanishing gradients** stop many RNN architectures from learning **long-range dependencies**

- **Parallelisation Bottleneck:** RNN states depend on previous time step hidden state, so must be **computed in series**

- **Attention**: Direct connections between output states and inputs (solves temporal bottleneck)

- **Self-Attention**: Remove recurrence, allowing parallel computation

- Modern **Transformers** use attention, but require position embeddings to capture sequence order

References

- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR, abs/1409.0473*.
- Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. *ArXiv, abs/1706.03762*.
- Wu et al., Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arxiv 2016