Attention

- a visual tour

Martin Stoffel

Transformer

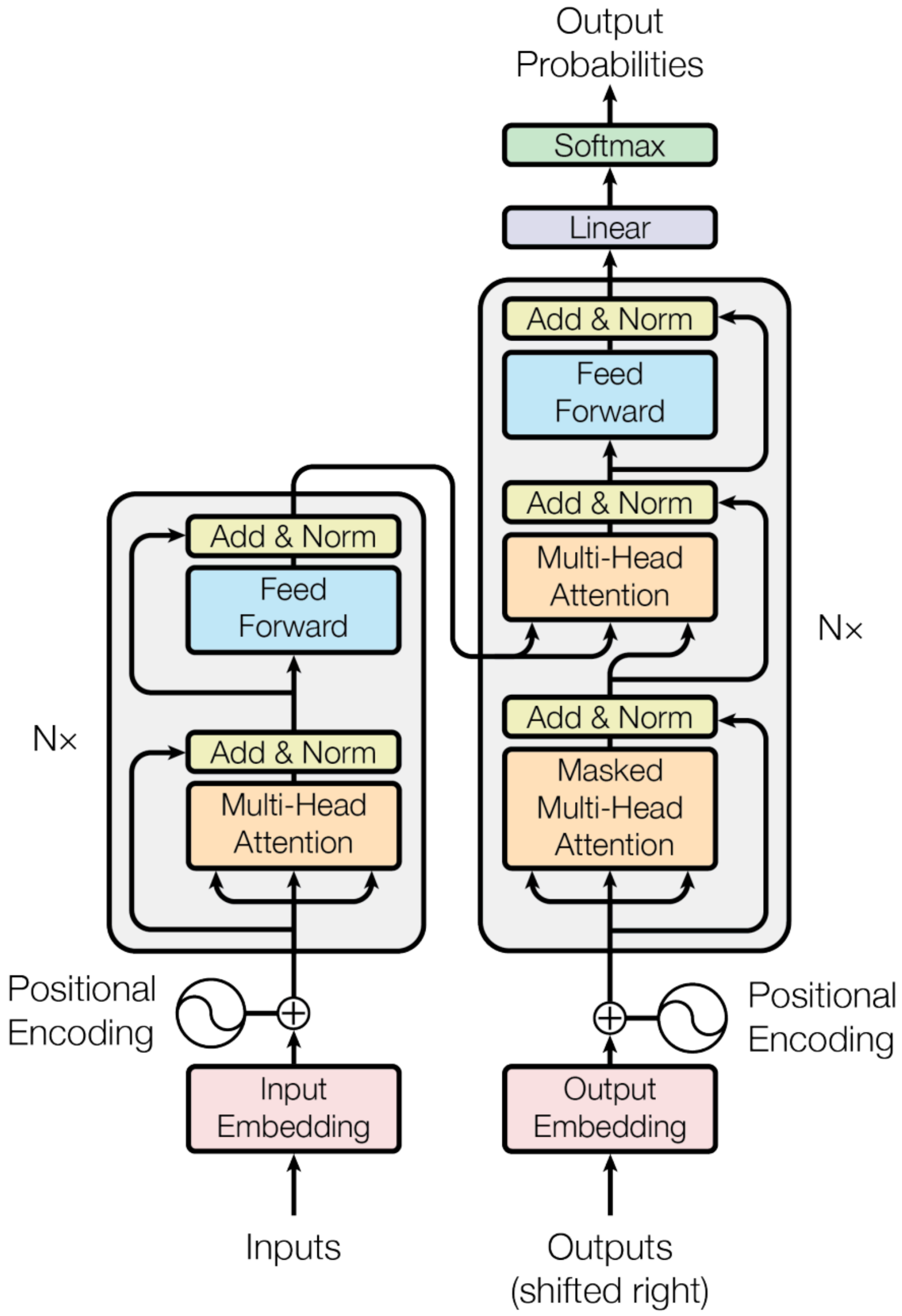
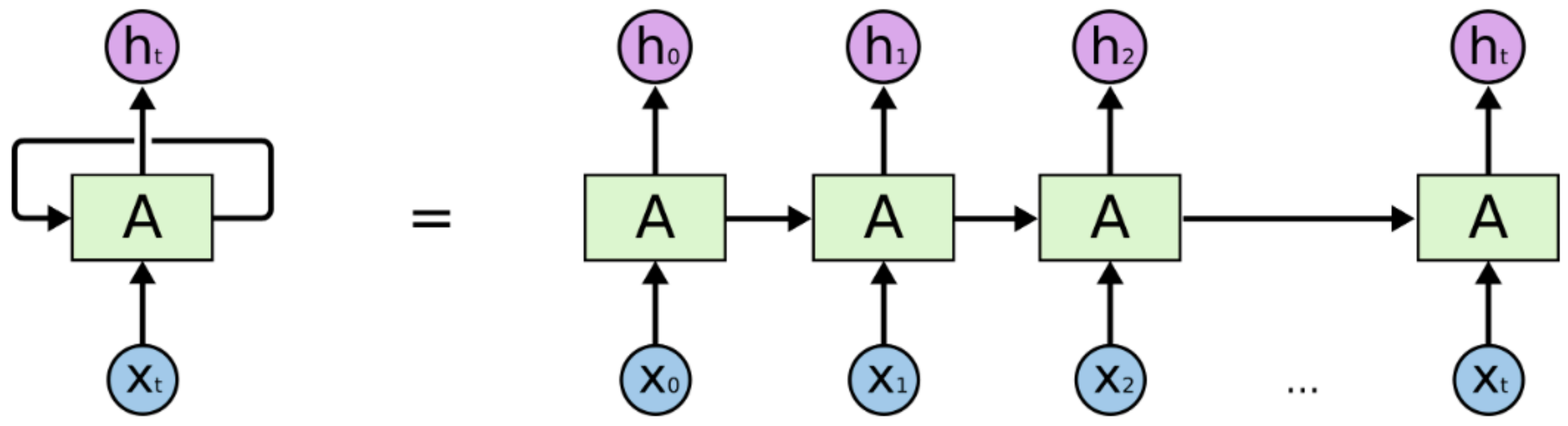


Figure 1: The Transformer - model architecture.

Vaswani et al. (2017) Attention is all you need

Why attention?



An unrolled recurrent neural network.

- Recurrence in RNNs and LSTMs has two fundamental weaknesses:
  - **Capturing long-range dependencies:** exploding/vanishing gradients
  - **Sequential computation:** makes parallelisation hard

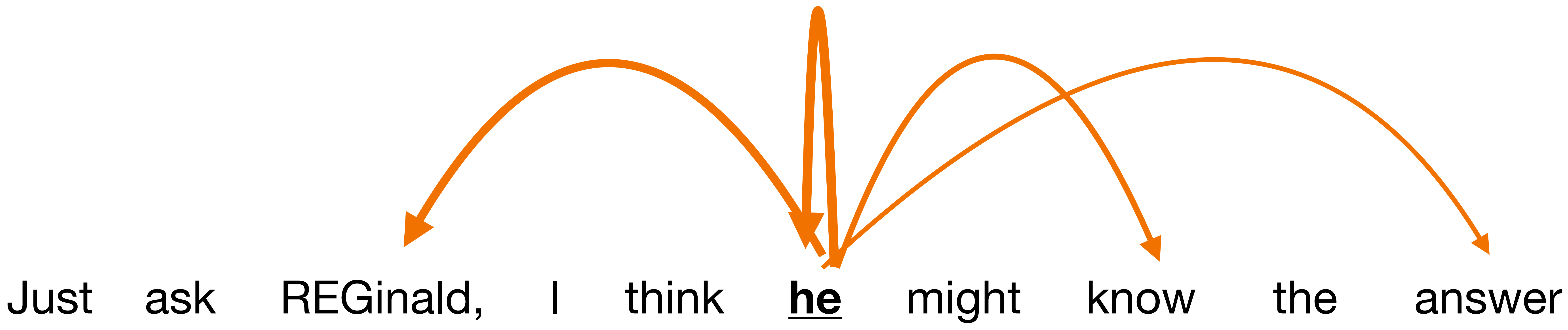
  - Also: attention seems to be a bit more *interpretable*

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

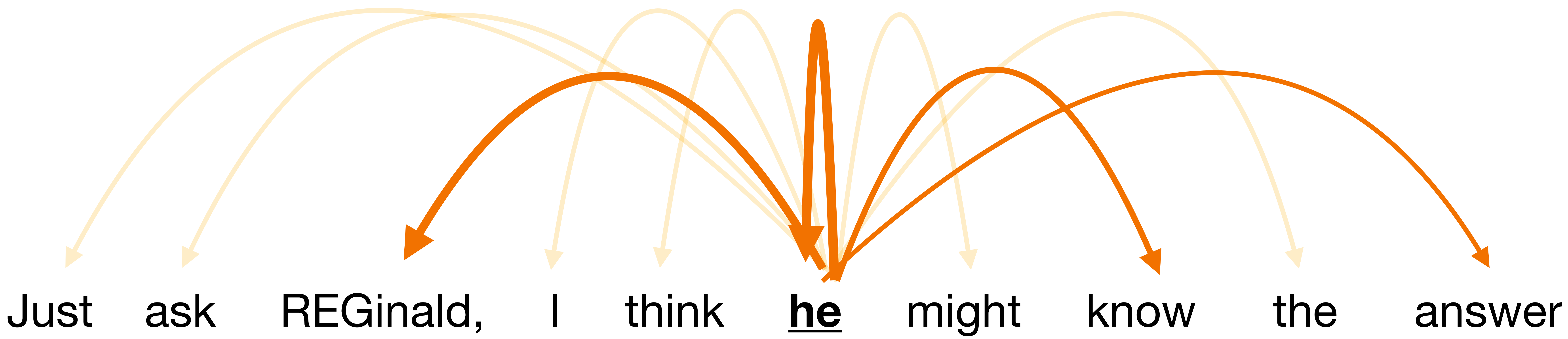
Attention: Goal

Just ask REGinald, I think **he** might know the answer

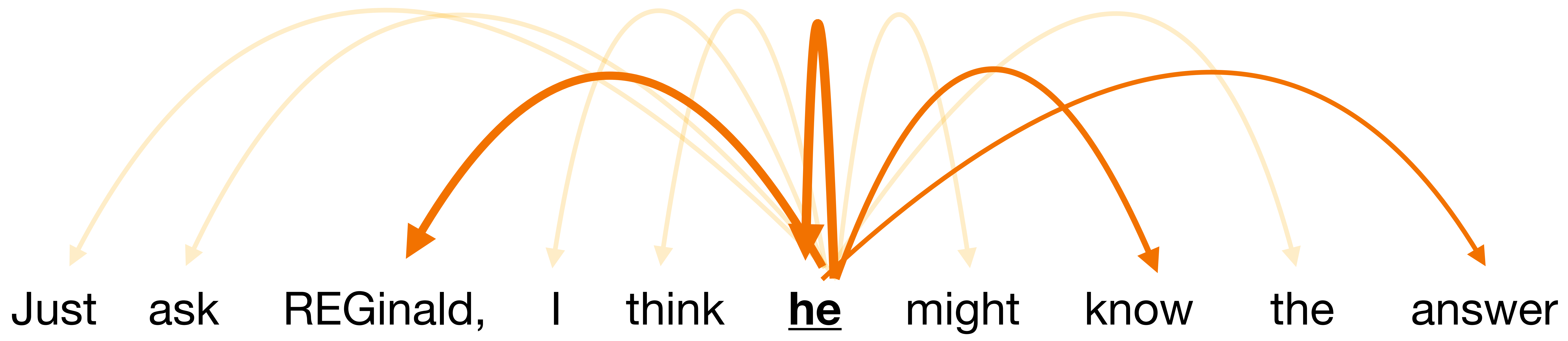
Attention: Goal



Attention: Goal

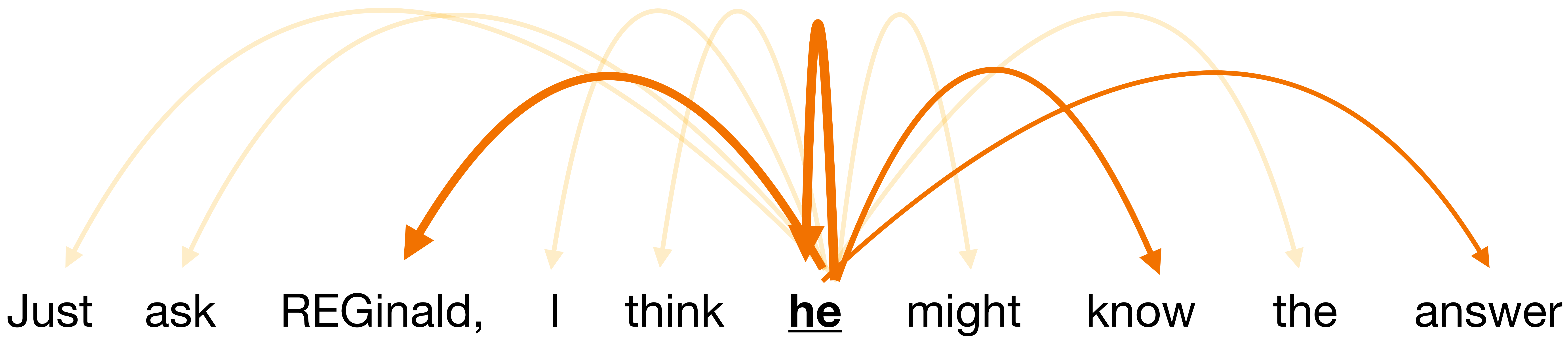


Attention: Goal



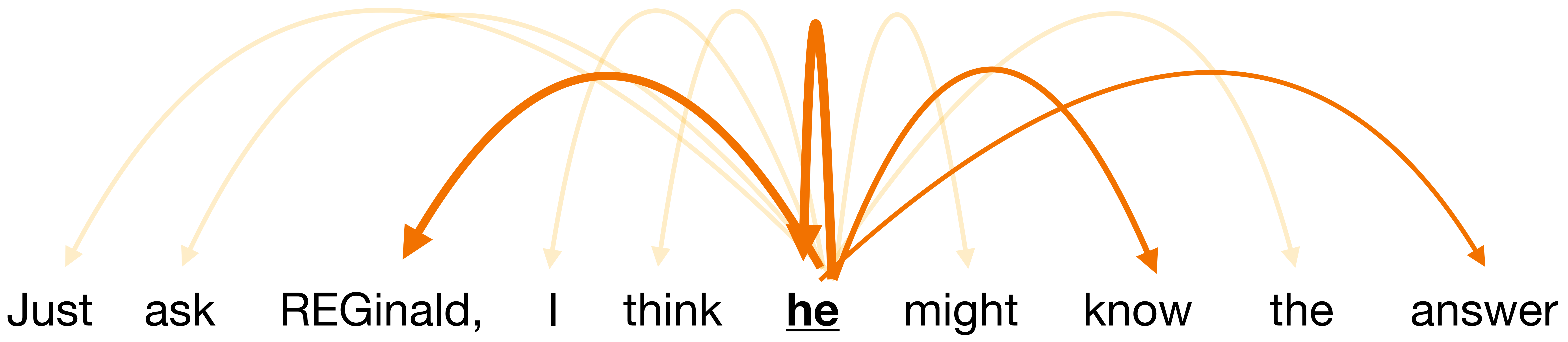
Me: What's the goal of the attention mechanism in transformers, in one illuminating sentence?

Attention: Goal



Me: What's the goal of the attention mechanism in transformers, in one illuminating sentence? Imagine you're not an AI, but a technologically advanced alien species, which doesn't use language to communicate.

Attention: Goal

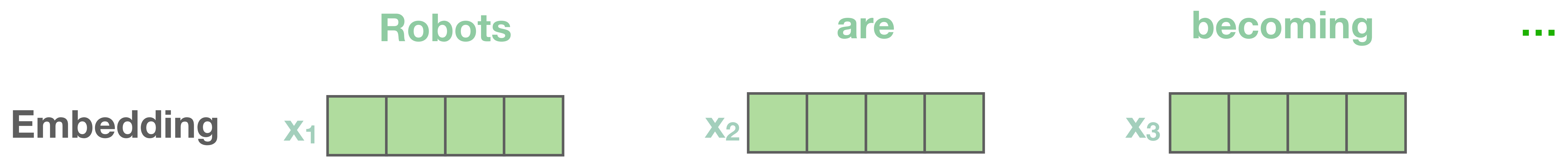


Me: What's the goal of the attention mechanism in transformers, in one illuminating sentence? Imagine you're not an AI, but a technologically advanced alien species, which doesn't use language to communicate.

ChatGP The attention mechanism in Transformers is akin to our advanced pattern recognition systems, **scanning and selectively amplifying** pertinent signals from a sea of information, regardless of their temporal sequence, to understand the **intricate dance of human symbolic sequences**, analogous to how we decipher cosmic vibrations.

Attention 1)

Keys, Queries, Values



Based on https://jalammar.github.io/illustrated-transformer/

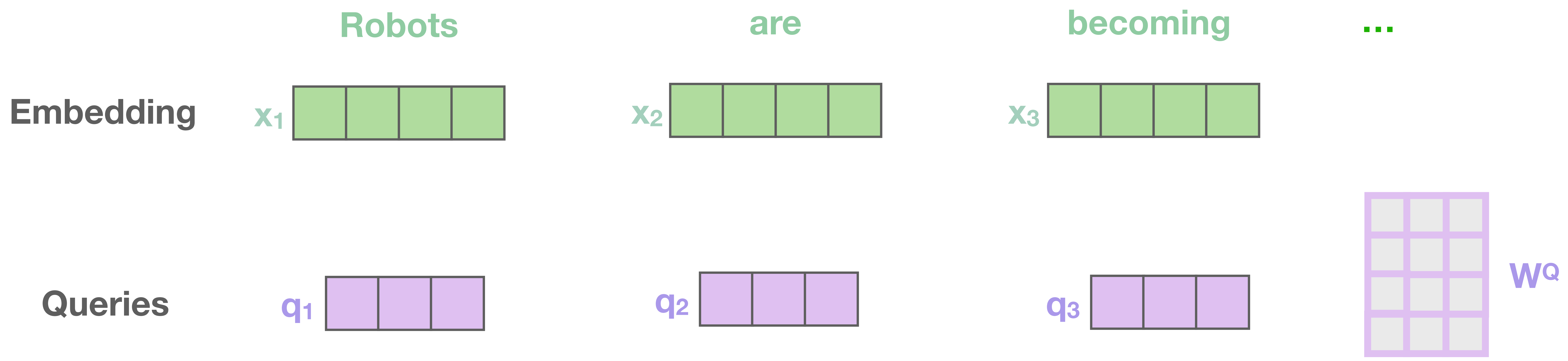
Attention 1)

Keys, Queries, Values

$q_1 = x_1 W^Q$

$q_2 = x_2 W^Q$

$q_3 = x_3 W^Q$



Based on
https://jalammar.github.io/illustrated-transformer/

Attention 1)

Keys, Queries, Values

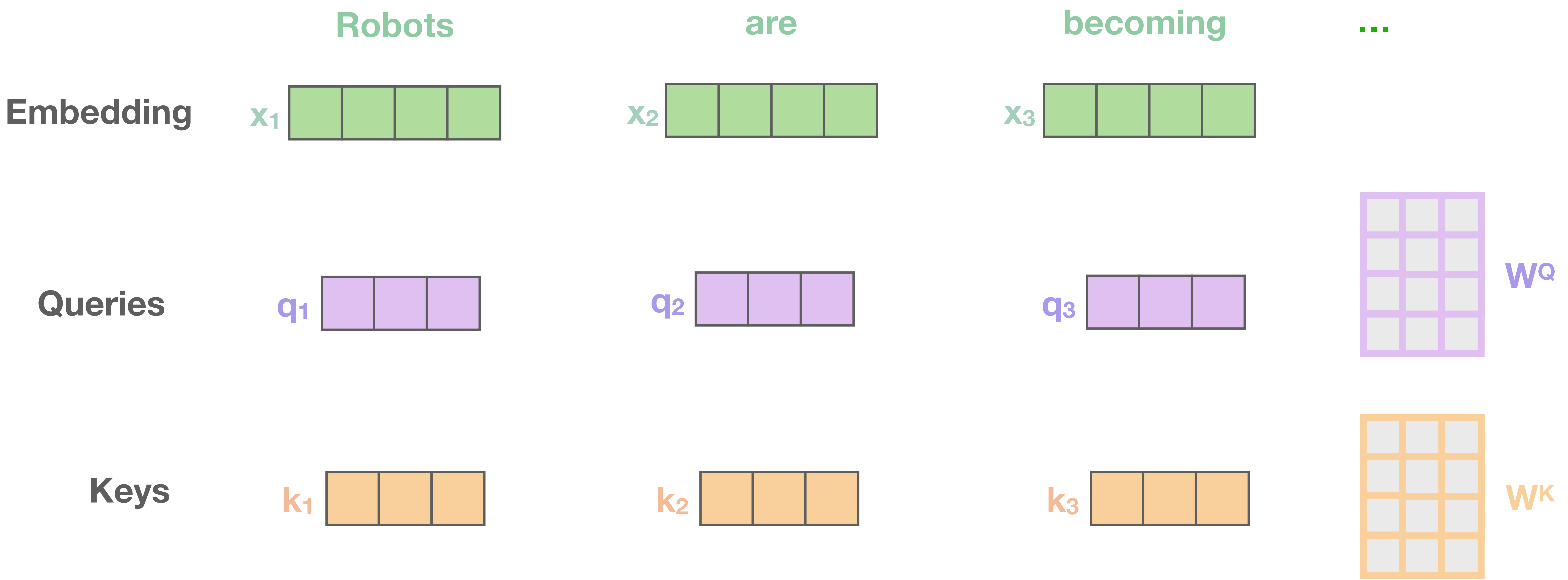
$q_1 = x_1 W^Q$

$q_2 = x_2 W^Q$

$q_3 = x_3 W^Q$

$k_1 = x_1 W^K$

$\vdots$



Based on
https://jalammar.github.io/illustrated-transformer/

Attention 1)

Keys, Queries, Values

$q_1 = x_1 W^Q$

$q_2 = x_2 W^Q$

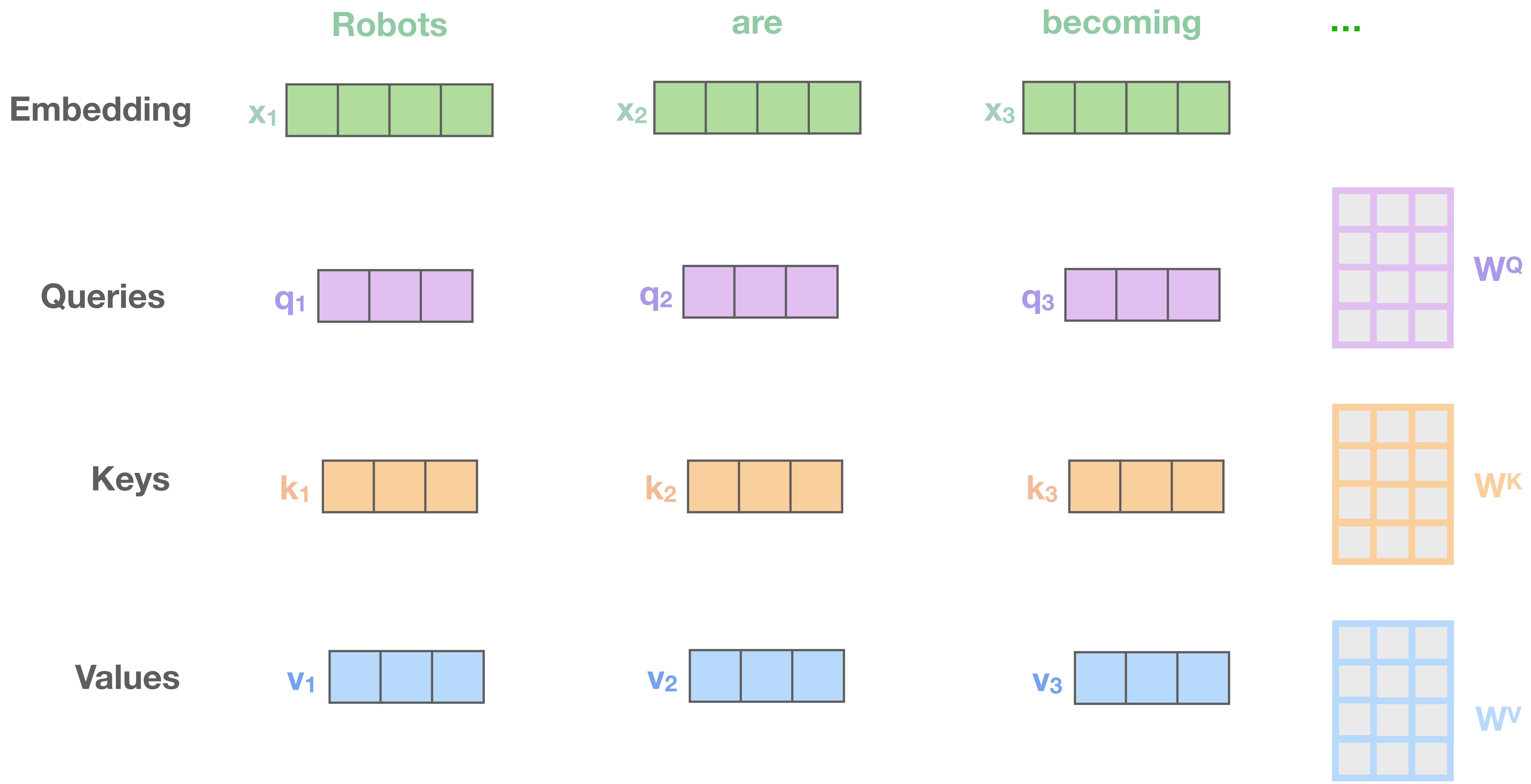
$q_3 = x_3 W^Q$

$k_1 = x_1 W^K$

$\vdots$

$v_1 = x_1 W^V$

$\vdots$



Based on
https://jalammar.github.io/illustrated-transformer/

Attention 1)

Keys, Queries, Values

$q_1 = x_1 W^Q$

$q_2 = x_2 W^Q$

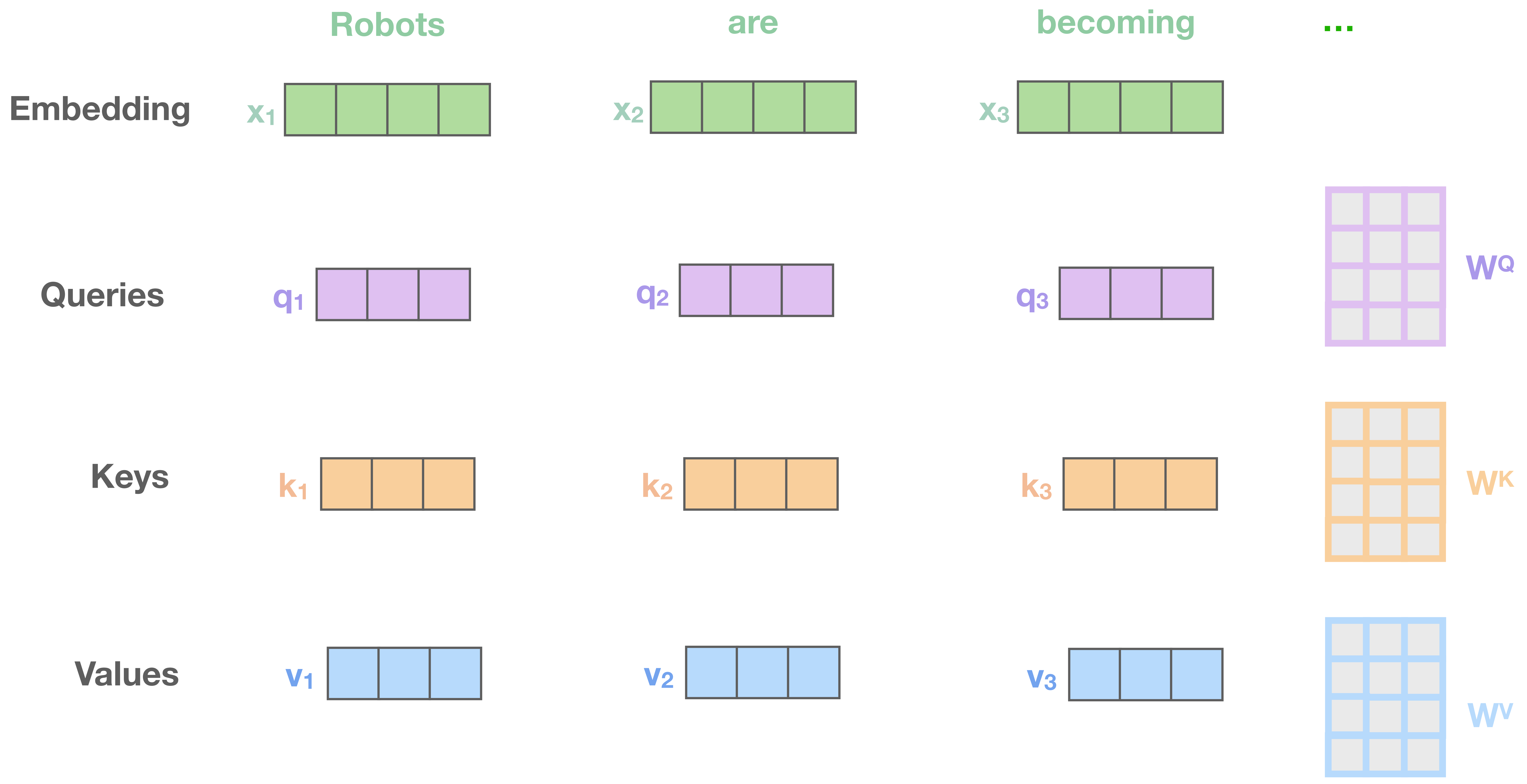
$q_3 = x_3 W^Q$

$k_1 = x_1 W^K$

⋮

$v_1 = x_1 W^V$

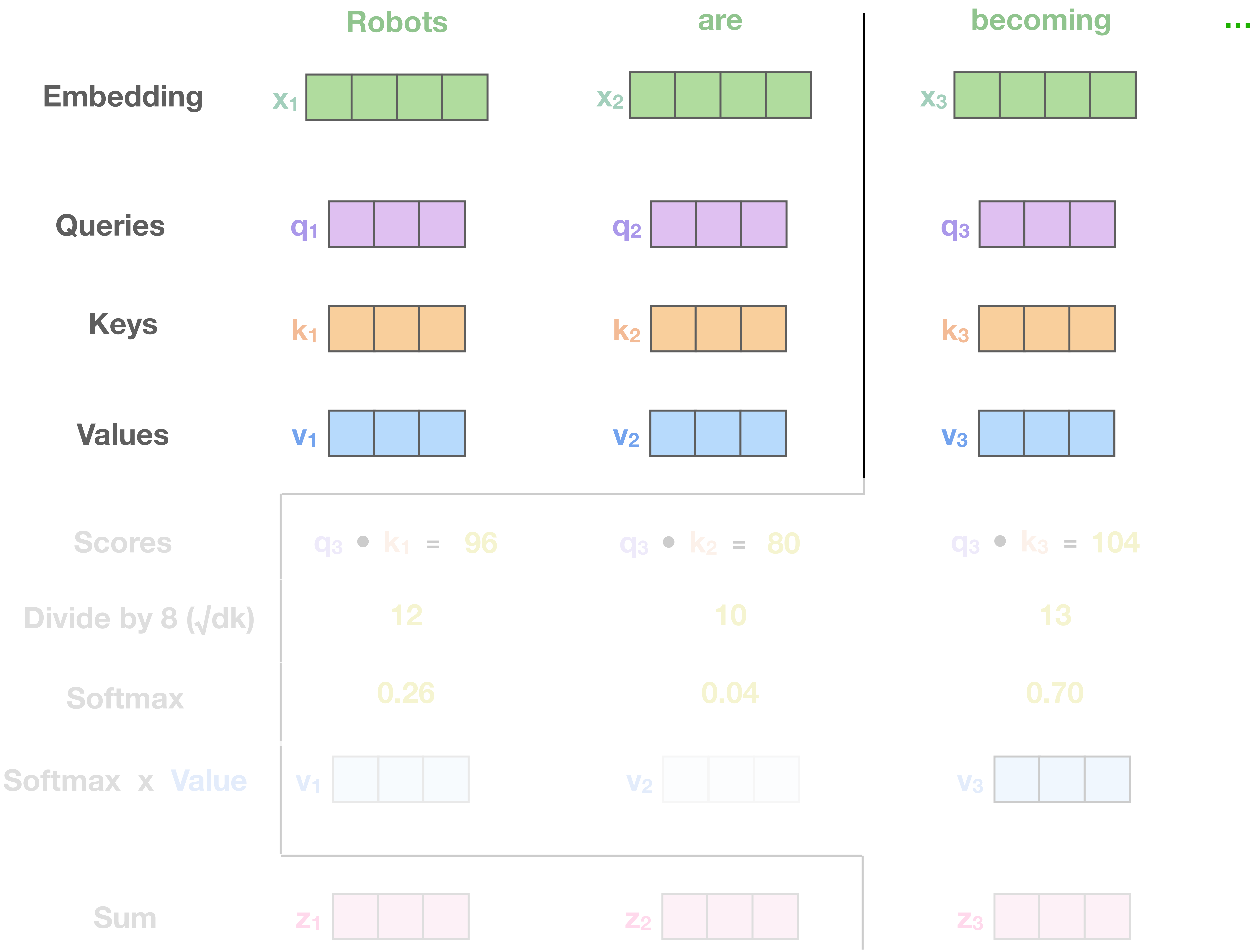
⋮



-> Linear projection of each input vector x into a query, key and value vector

Based on
https://jalammar.github.io/illustrated-transformer/

Attention 2)
Calculations



Based on
https://jalammar.github.io/illustrated-transformer/

Attention 2)
Calculations

	Robots	are	becoming	...
Embedding	x_1	x_2	x_3	
Queries	q_1	q_2	q_3	
Keys	k_1	k_2	k_3	
Values	v_1	v_2	v_3	
Scores	$q_3 \bullet k_1 = 96$	$q_3 \bullet k_2 = 80$	$q_3 \bullet k_3 = 104$	
Divide by 8 ($\sqrt{dk}$)	12	10	13	
Softmax	0.26	0.04	0.70	
Softmax x Value	v_1	v_2	v_3	
Sum	z_1	z_2	z_3	

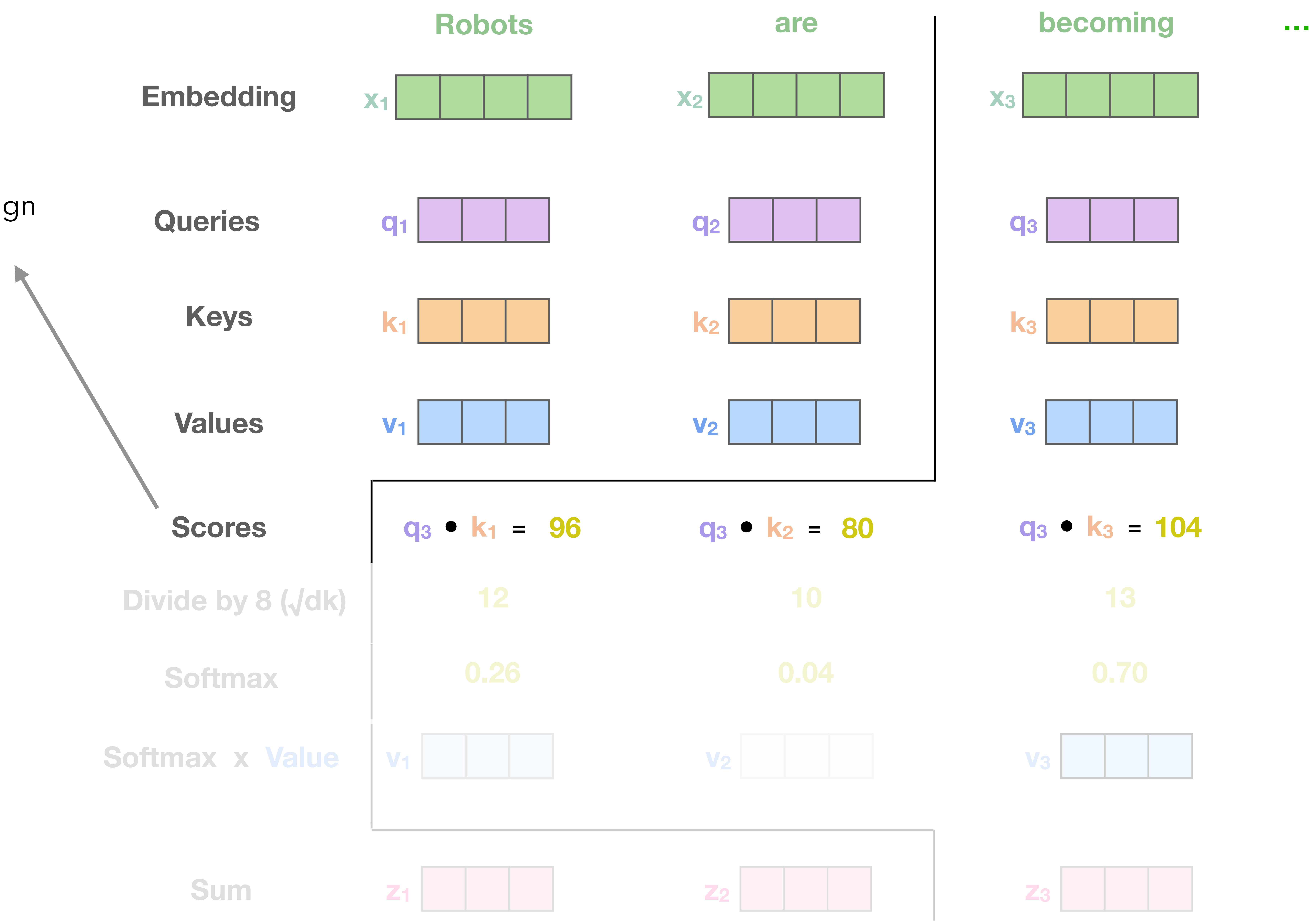
Based on
https://jalammar.github.io/illustrated-transformer/

Attention 2)

Calculations

Scores:

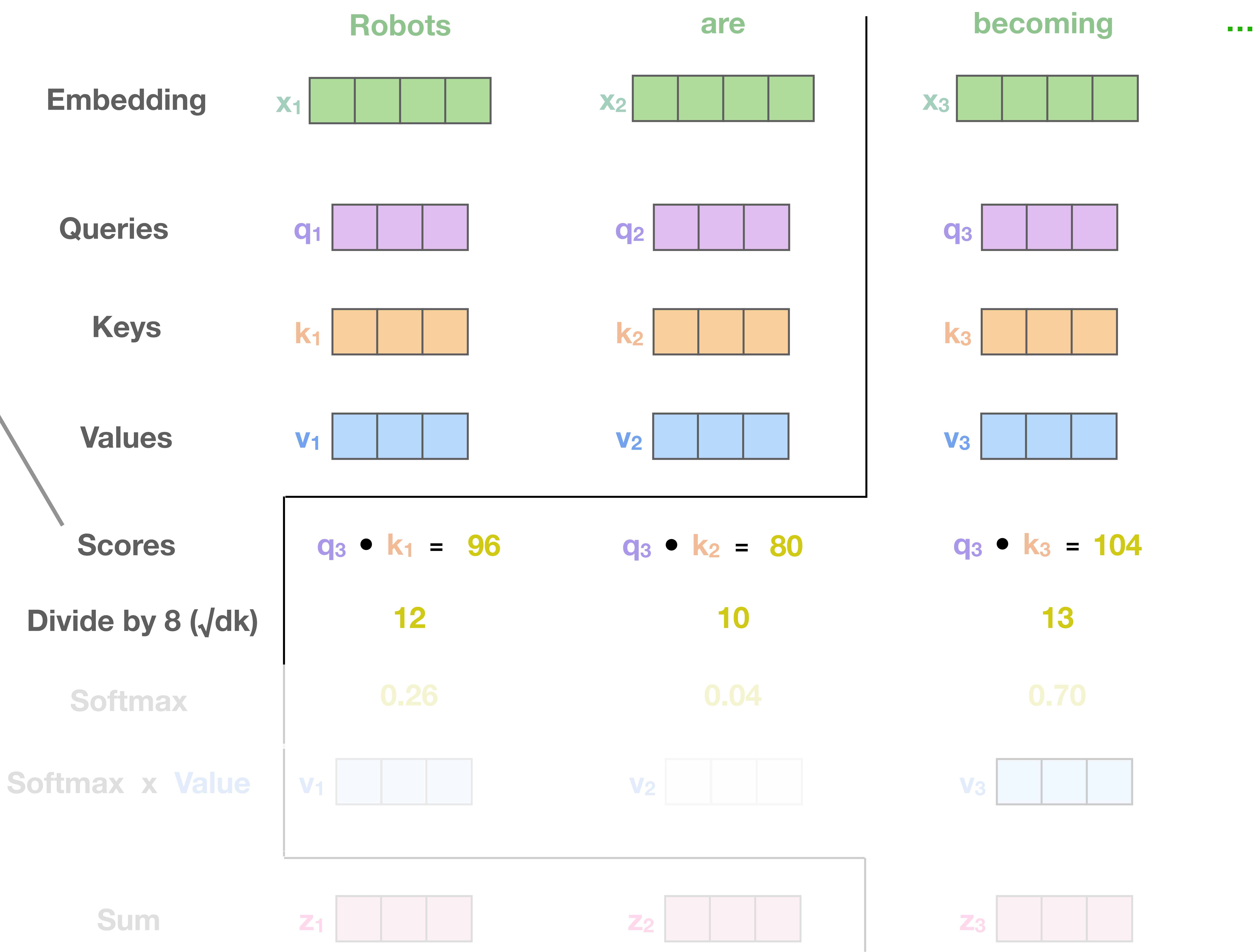
How well does the current **query** align with all the **keys** in the sequence?
(*raw attention scores*)



Based on
https://jalammar.github.io/illustrated-transformer/

Attention 2)
Calculations

Scores:
How well does the current **query** align with all the **keys** in the sequence? (*raw attention scores*)



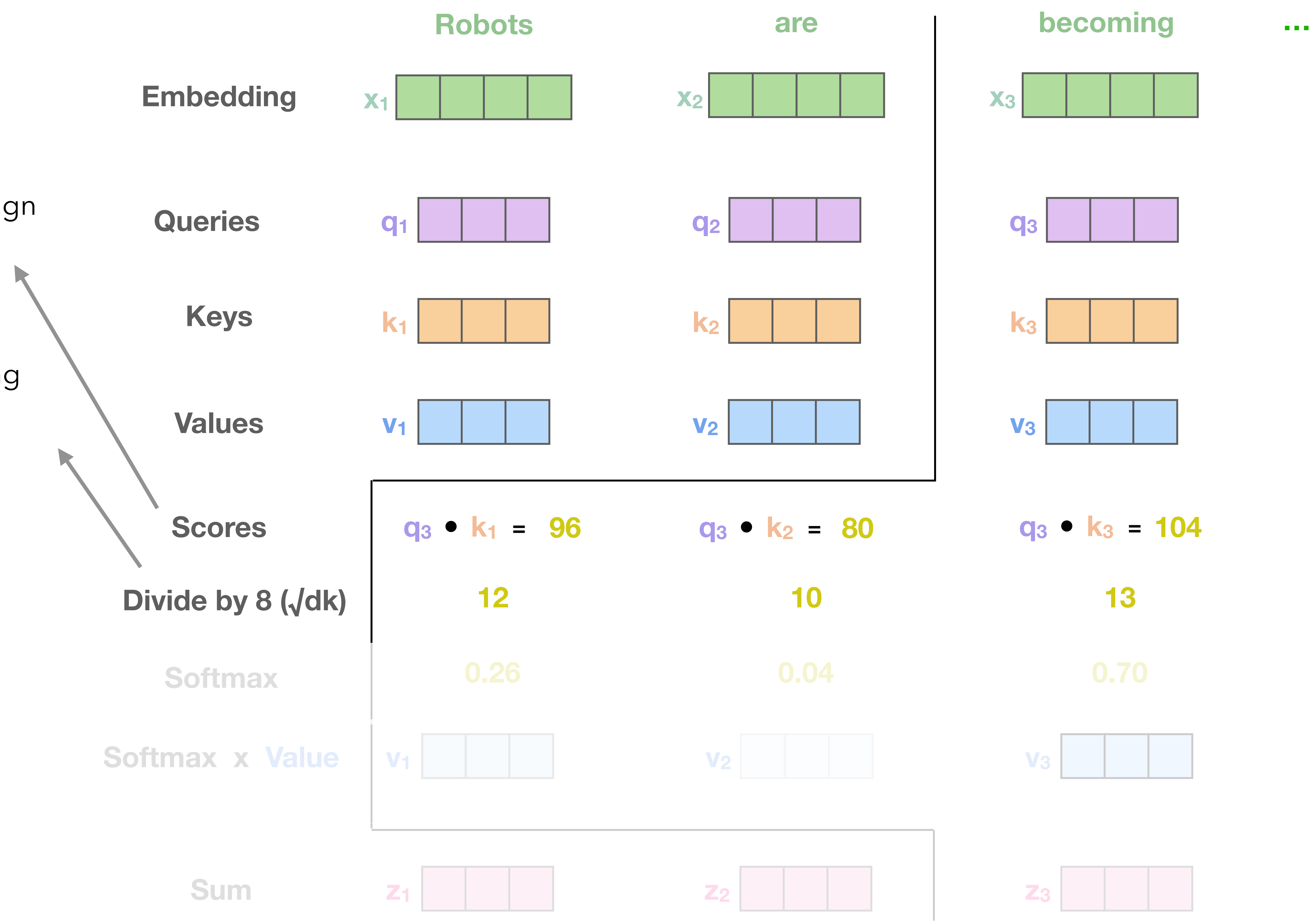
Based on
https://jalammar.github.io/illustrated-transformer/

Attention 2)

Calculations

Scores:
How well does the current **query** align with all the **keys** in the sequence? (*raw attention scores*)

Scaling factor:
Prevent dot-products from becoming too large and gradients too small.



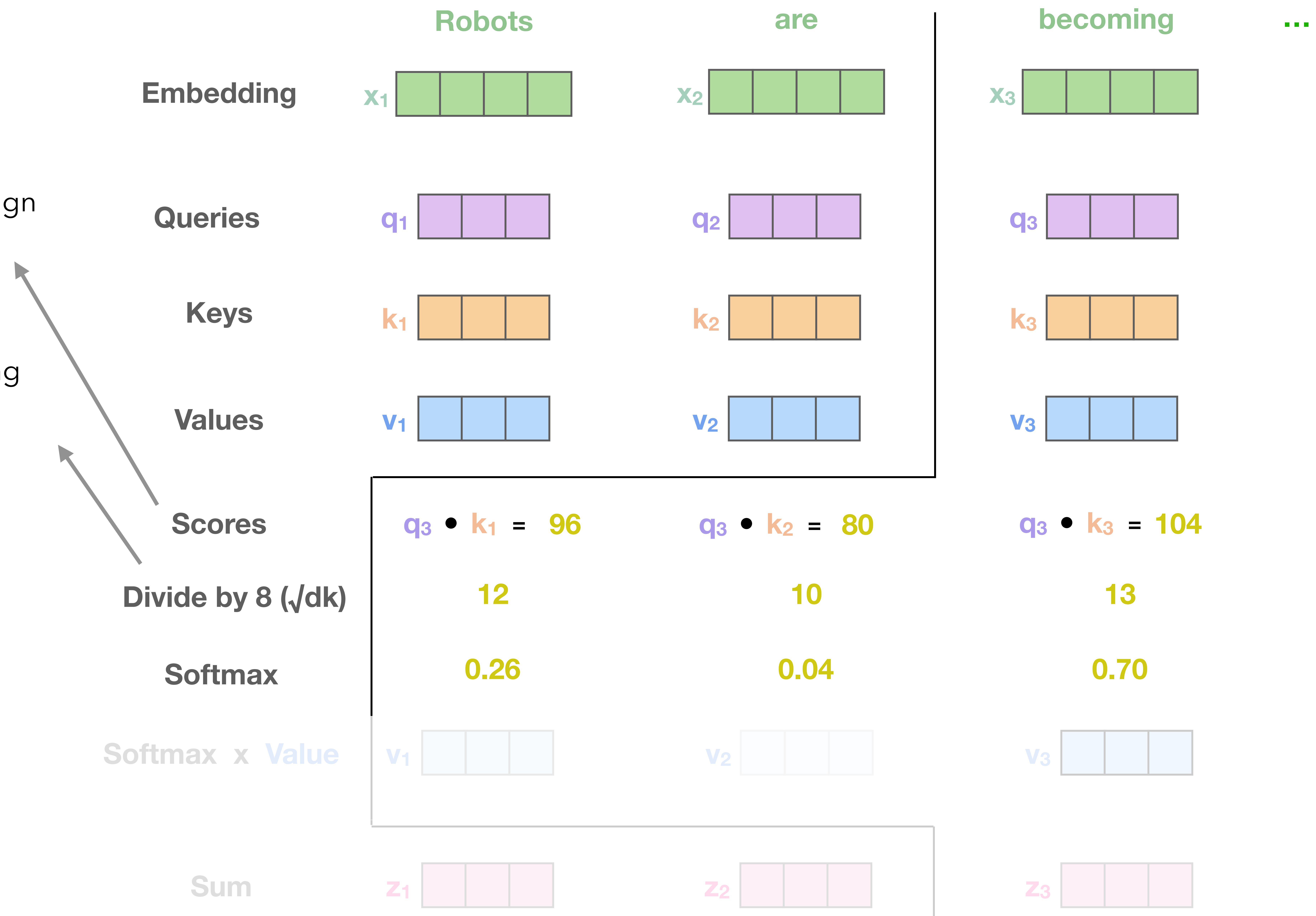
Based on
https://jalammar.github.io/illustrated-transformer/

Attention 2)

Calculations

Scores:
How well does the current **query** align with all the **keys** in the sequence? (*raw attention scores*)

Scaling factor:
Prevent dot-products from becoming too large and gradients too small.



Based on
https://jalammar.github.io/illustrated-transformer/

Attention 2)

Calculations

Scores:
How well does the current **query** align with all the **keys** in the sequence? (*raw attention scores*)

Scaling factor:
Prevent dot-products from becoming too large and gradients too small.

Softmax:
Probability distribution (summing to 1)
How much *attention* to pay to each token?
(scaled *attention scores*)

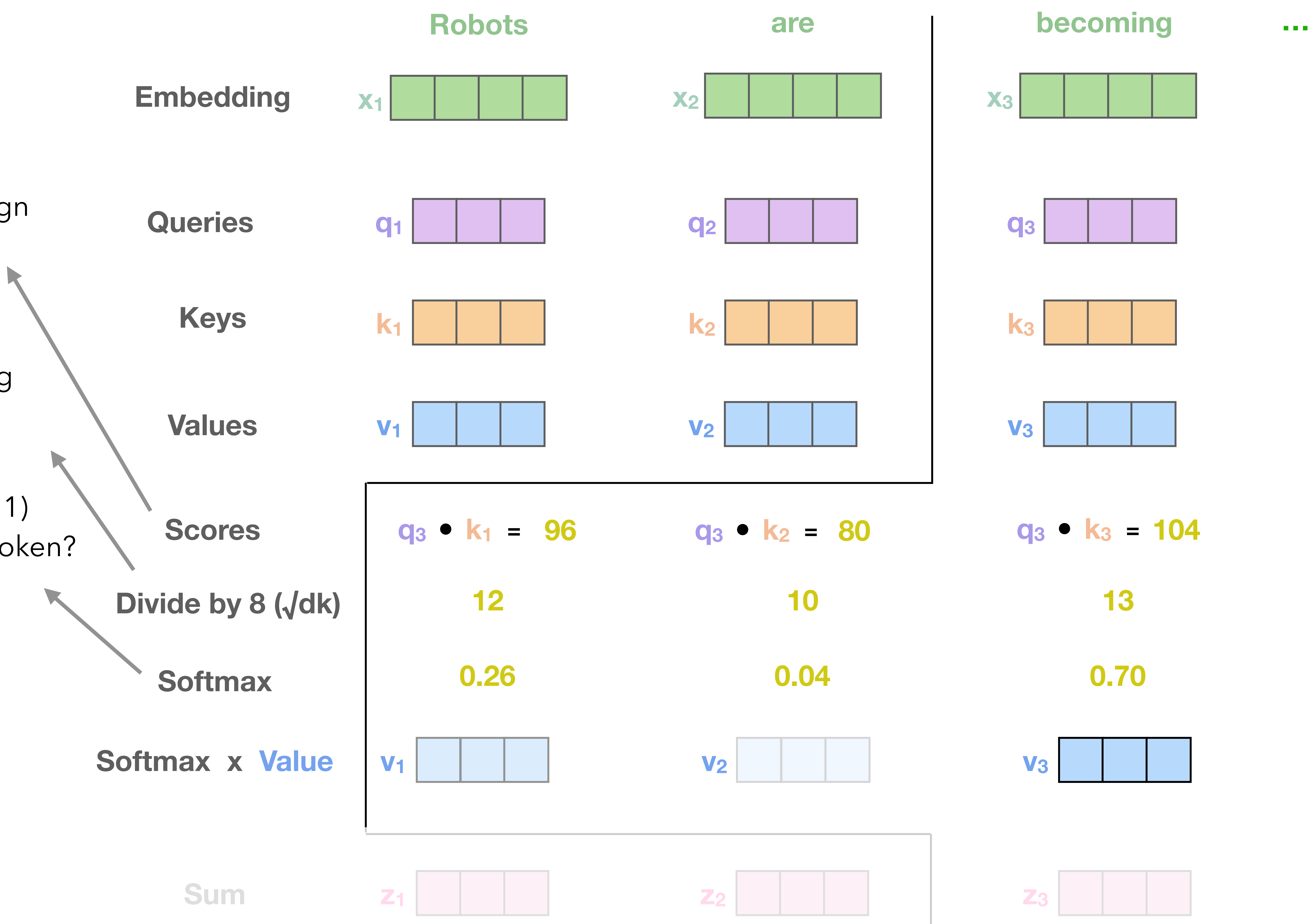
Based on
https://jalammar.github.io/illustrated-transformer/

Attention 2)
Calculations

Scores:
How well does the current **query** align with all the **keys** in the sequence? (*raw attention scores*)

Scaling factor:
Prevent dot-products from becoming too large and gradients too small.

Softmax:
Probability distribution (summing to 1)
How much *attention* to pay to each token?
(scaled *attention scores*)



Based on
https://jalammar.github.io/illustrated-transformer/

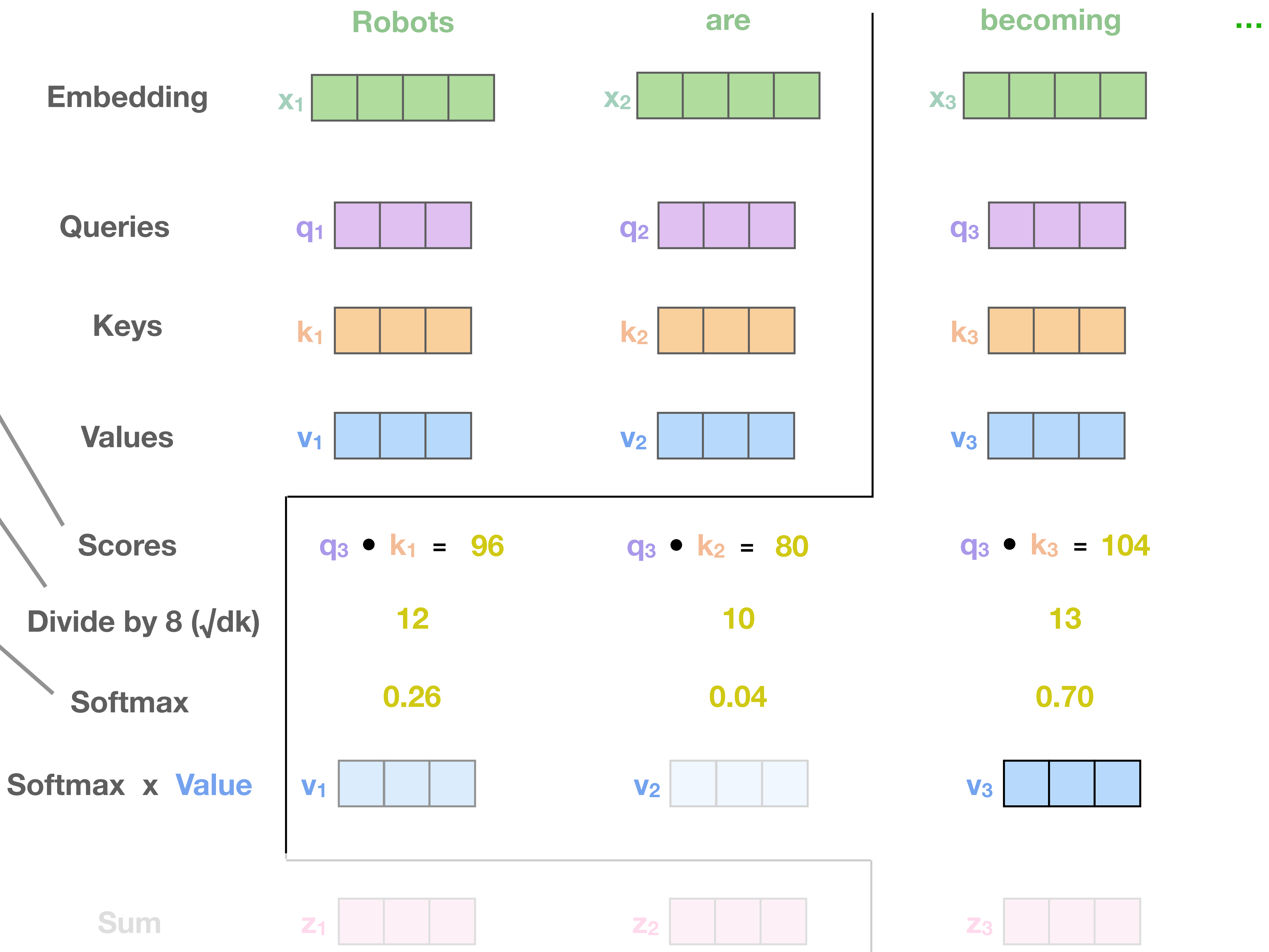
Attention 2)
Calculations

Scores:
How well does the current **query** align with all the **keys** in the sequence? (*raw attention scores*)

Scaling factor:
Prevent dot-products from becoming too large and gradients too small.

Softmax:
Probability distribution (summing to 1)
How much *attention* to pay to each token?
(scaled *attention scores*)

Attention-weighted values
Focus on relevant tokens, drown out irrelevant tokens.



Based on
https://jalammar.github.io/illustrated-transformer/

Attention 2)

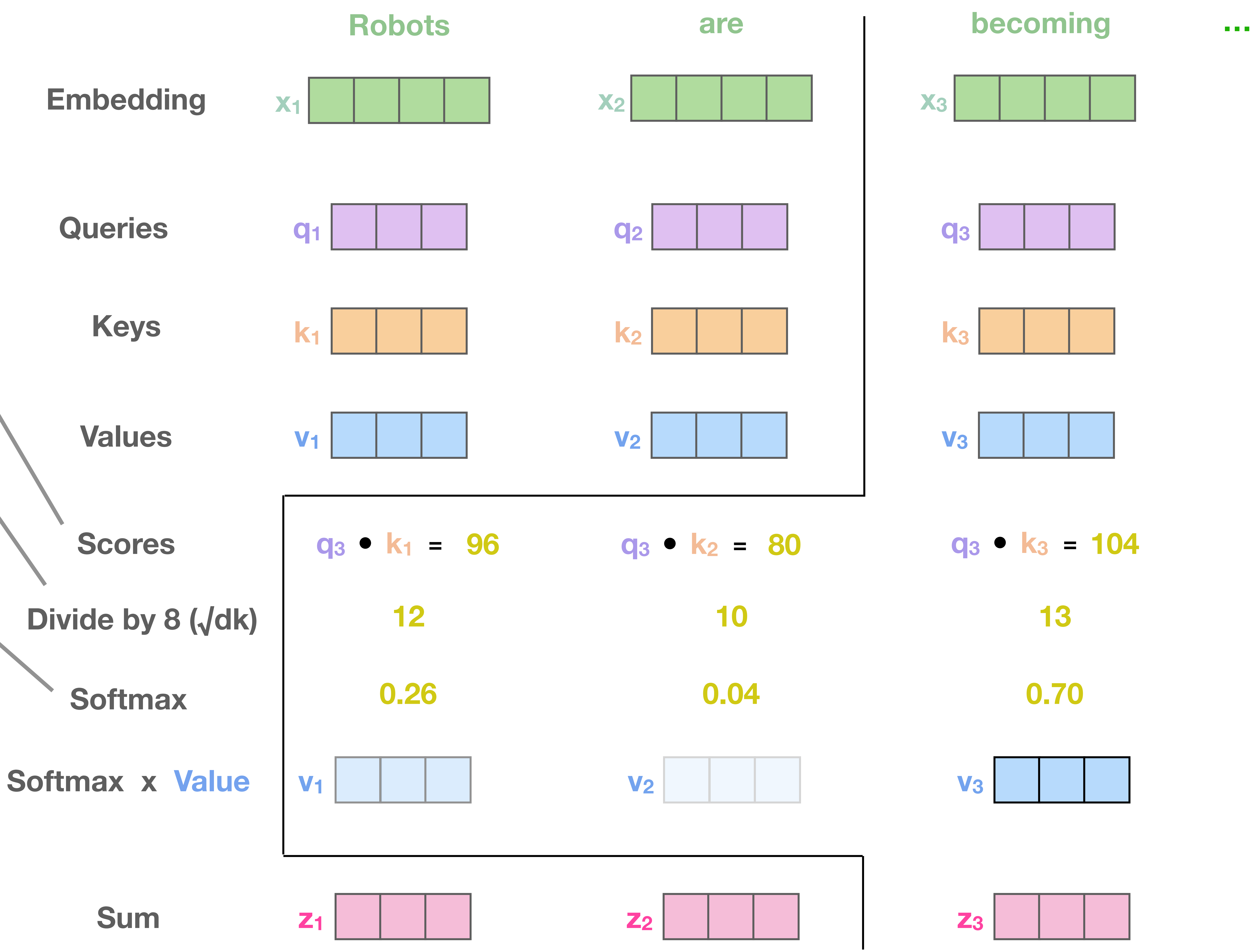
Calculations

Scores:
How well does the current **query** align with all the **keys** in the sequence? (*raw attention scores*)

Scaling factor:
Prevent dot-products from becoming too large and gradients too small.

Softmax:
Probability distribution (summing to 1)
How much *attention* to pay to each token?
(scaled *attention scores*)

Attention-weighted values
Focus on relevant tokens, drown out irrelevant tokens.



Based on
https://jalammar.github.io/illustrated-transformer/

Attention 2)

Calculations

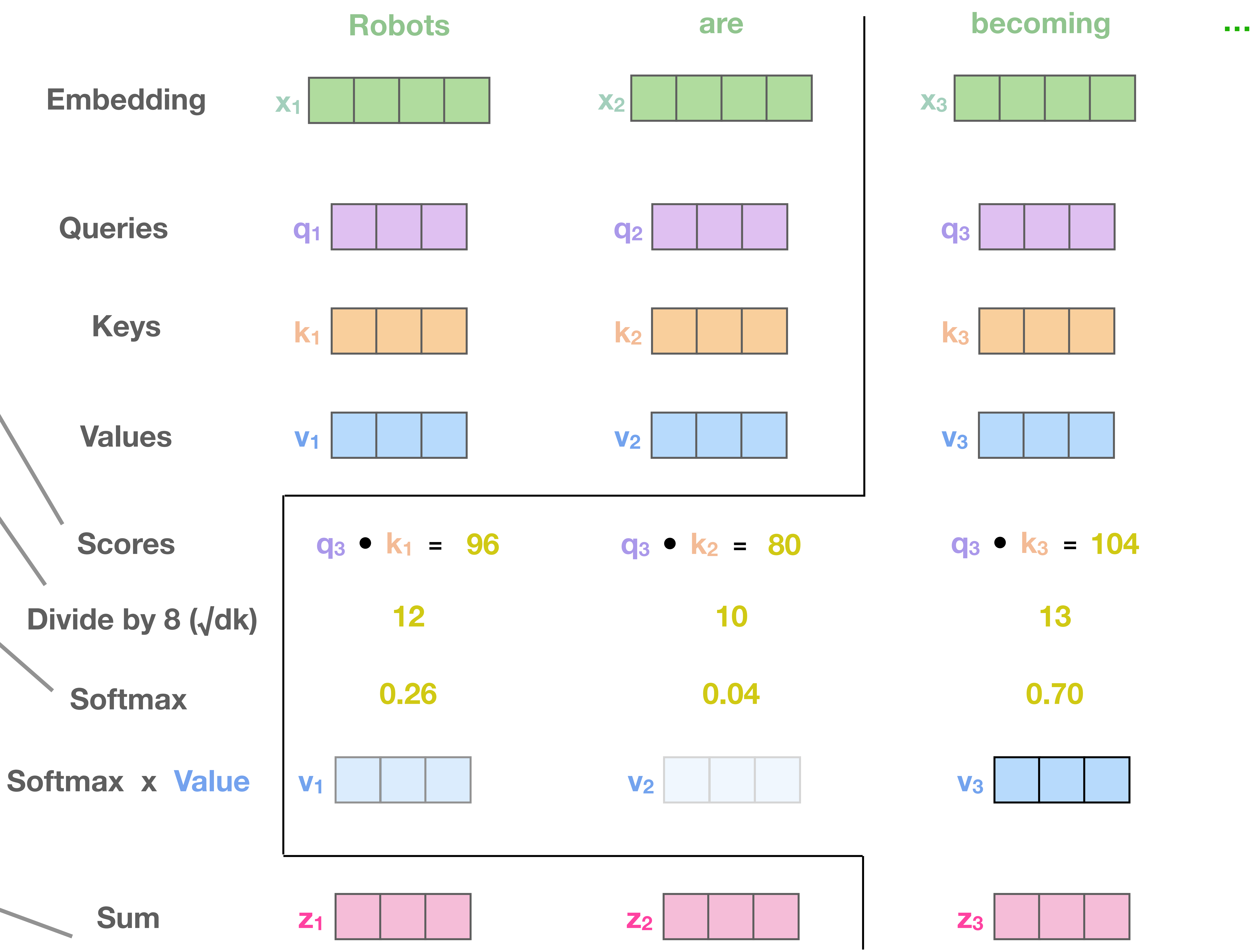
Scores:
How well does the current **query** align with all the **keys** in the sequence? (*raw attention scores*)

Scaling factor:
Prevent dot-products from becoming too large and gradients too small.

Softmax:
Probability distribution (summing to 1)
How much *attention* to pay to each token?
(scaled *attention scores*)

Attention-weighted values
Focus on relevant tokens, drown out irrelevant tokens.

Sum:
Aggregate attention-weighted **value** vectors into an output vector **z**

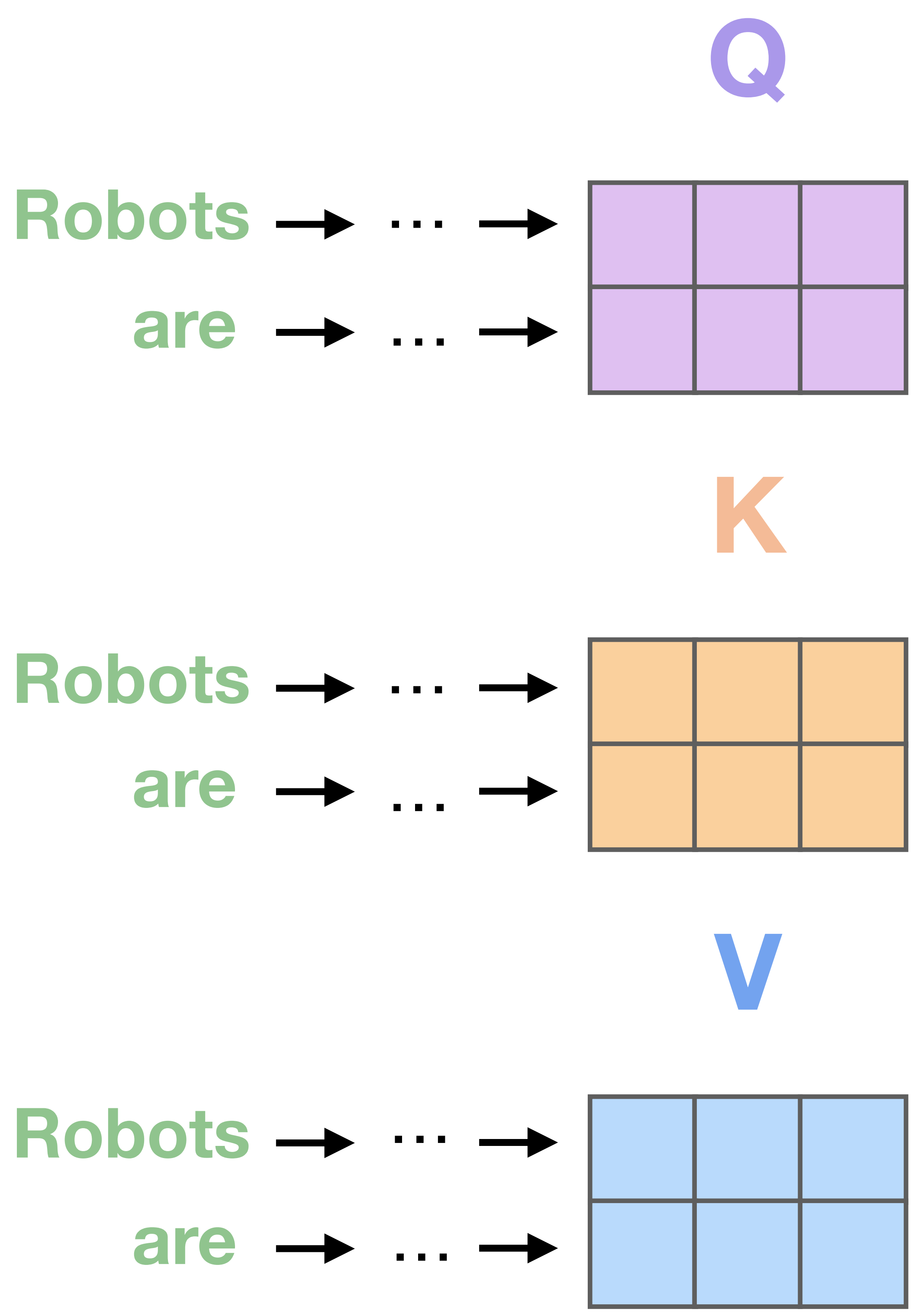


Based on
https://jalammar.github.io/illustrated-transformer/

Attention 3)

Matrix form

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

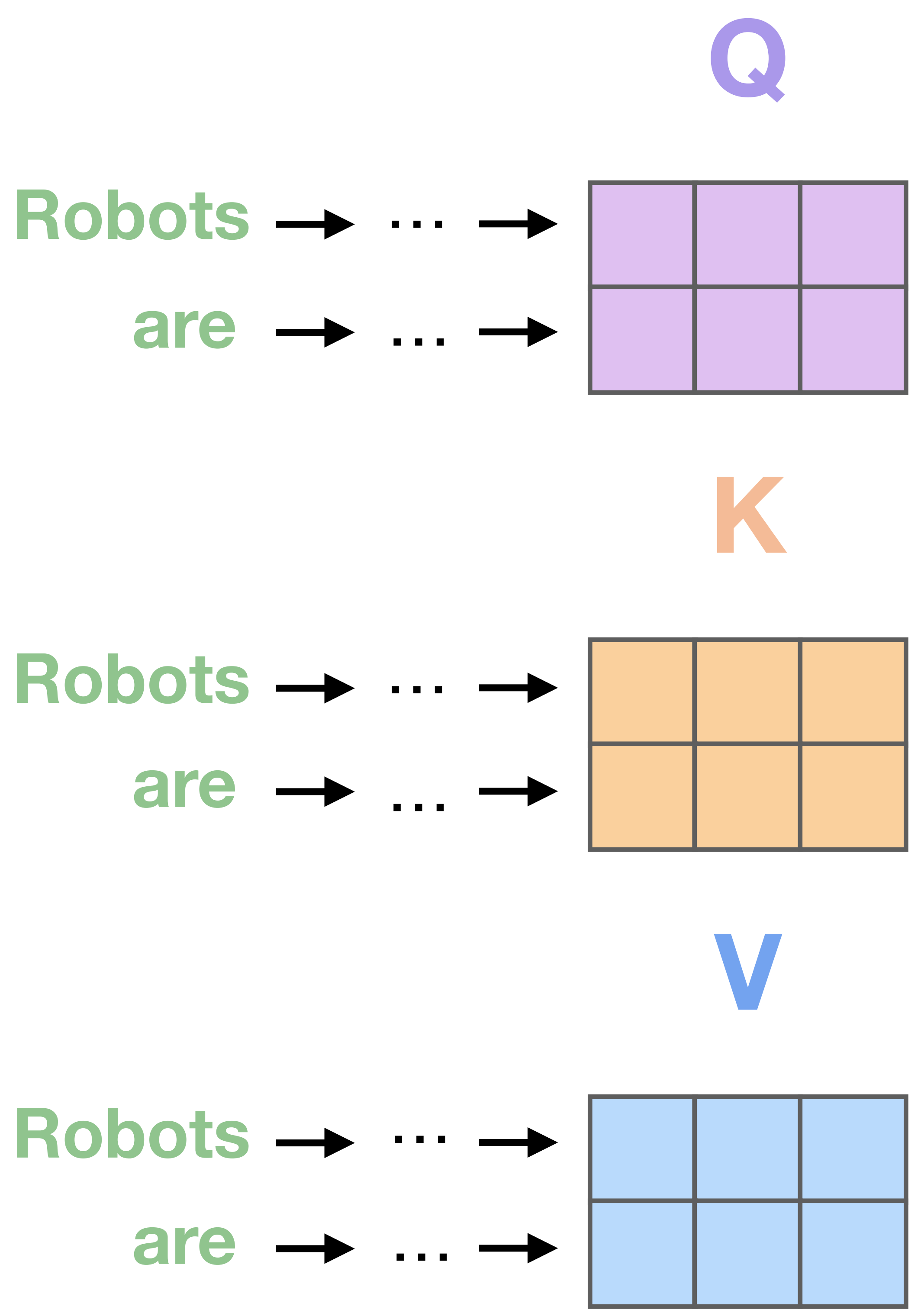


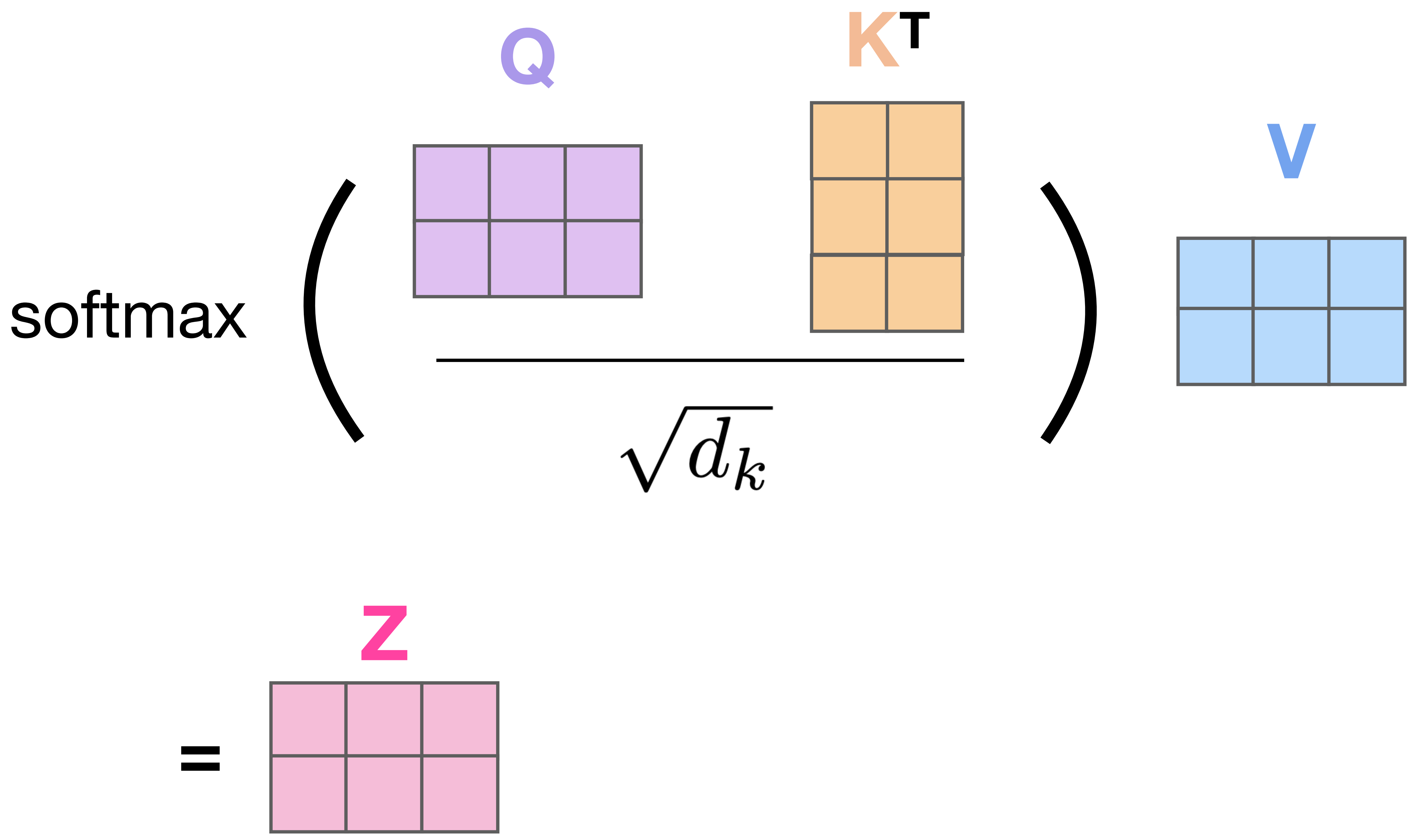
Based on
https://jalammar.github.io/illustrated-transformer/

Attention 3)

Matrix form

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

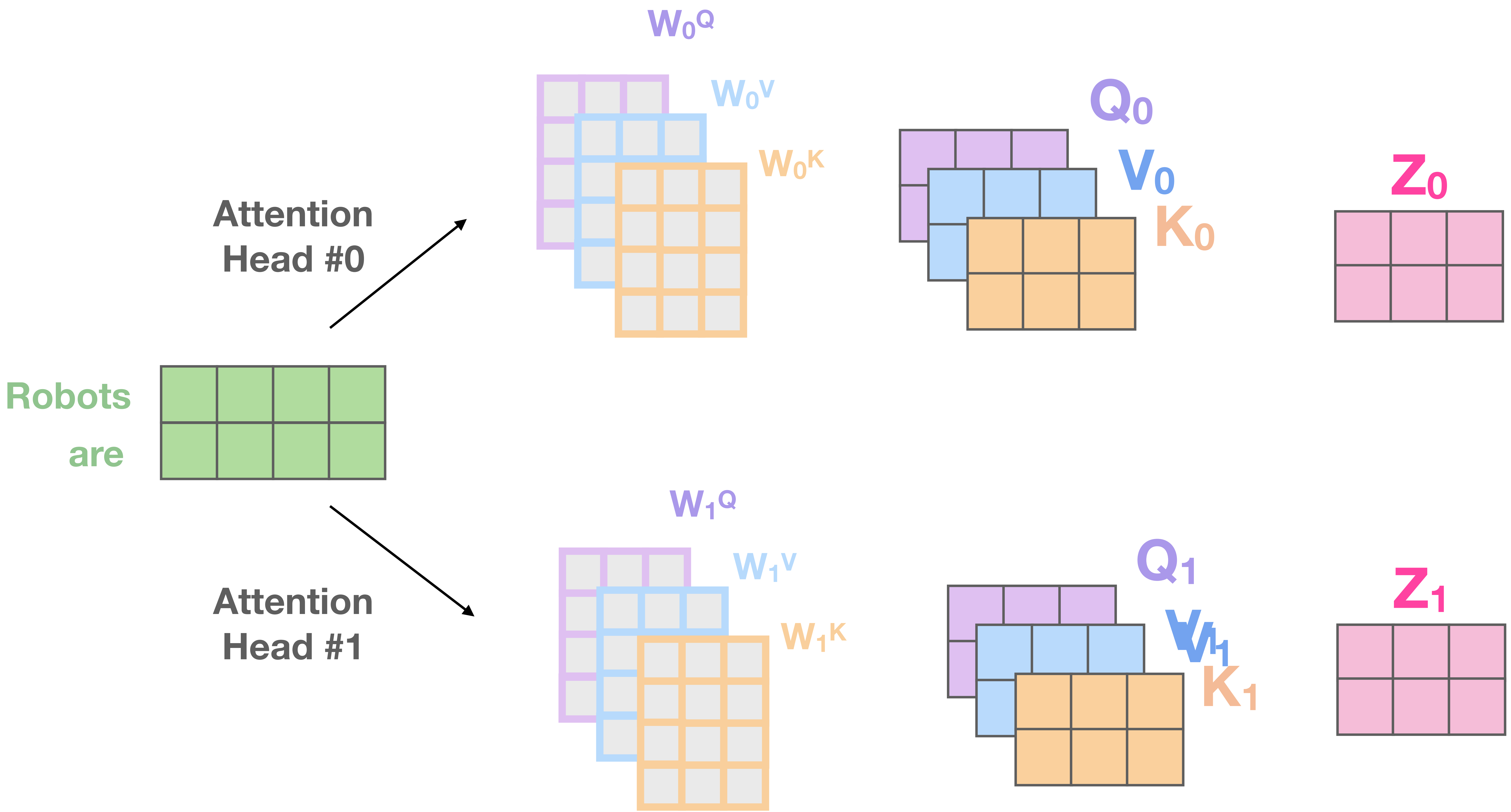




Based on
https://jalammar.github.io/illustrated-transformer/

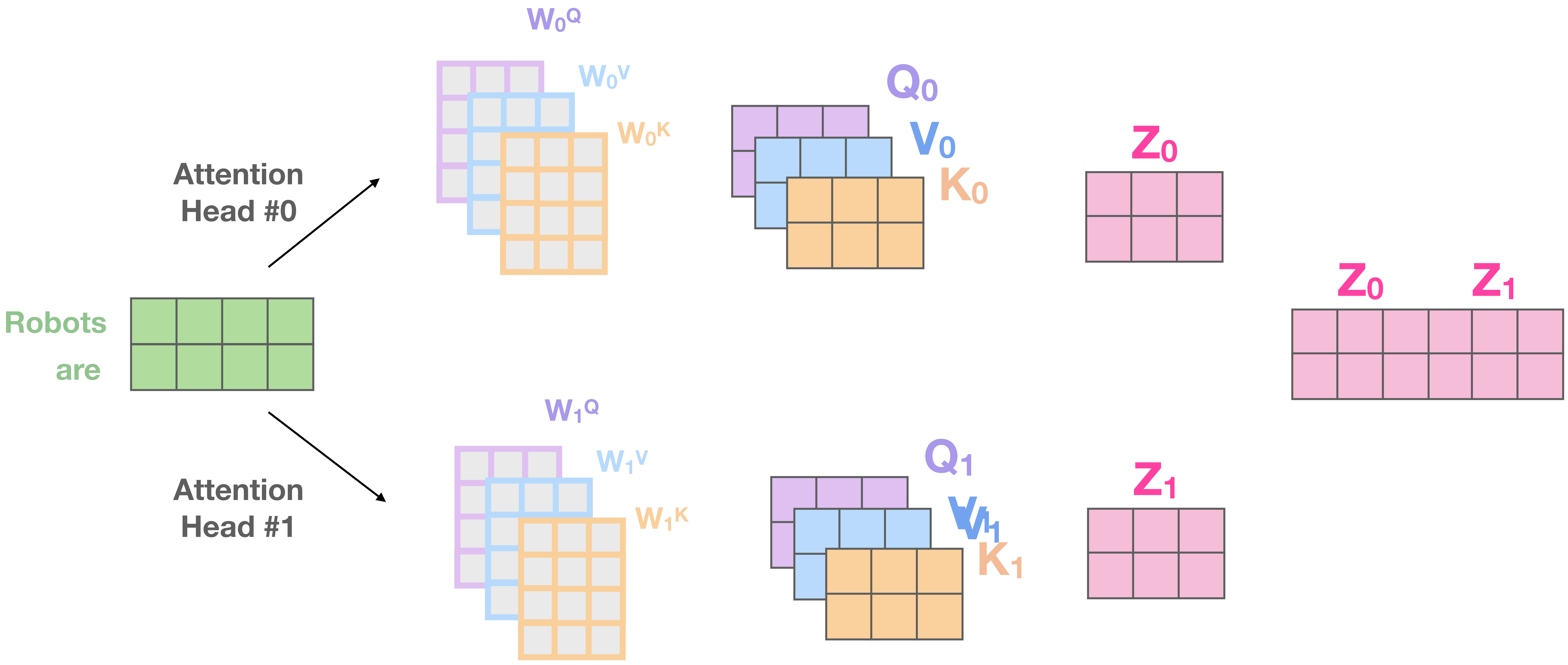
Attention 4)

Multiple heads



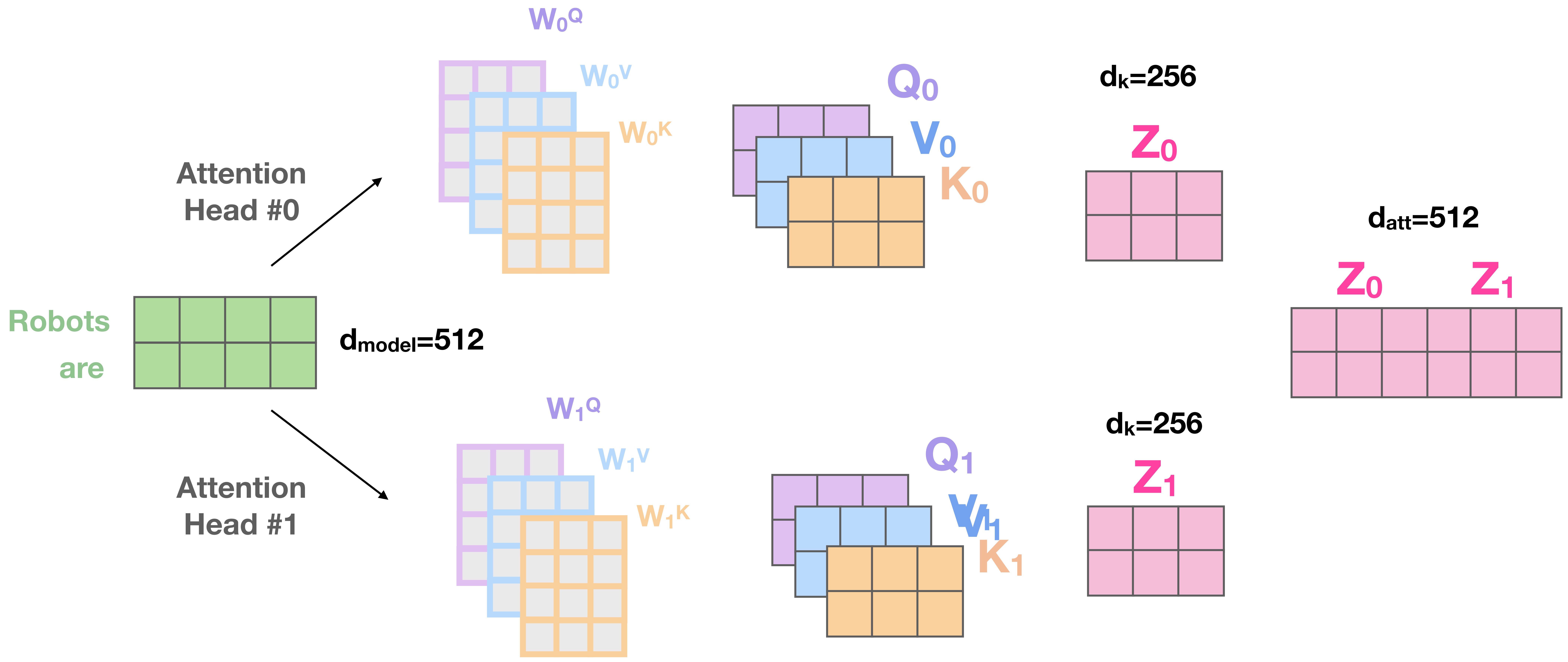
Attention 4)

Multiple heads



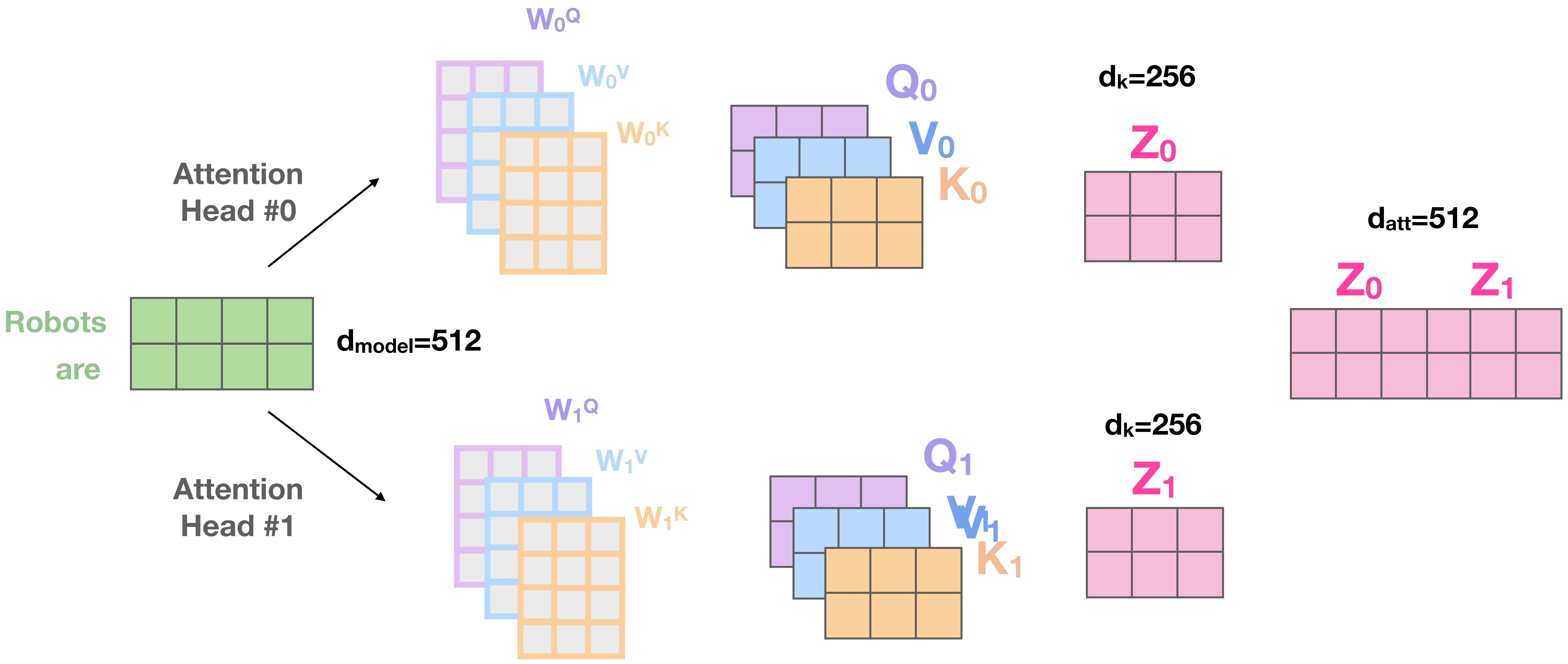
Attention 4)

Multiple heads



Attention 4)

Multiple heads



-> Allows the model to focus on different positions at each head

Three variants of Attention

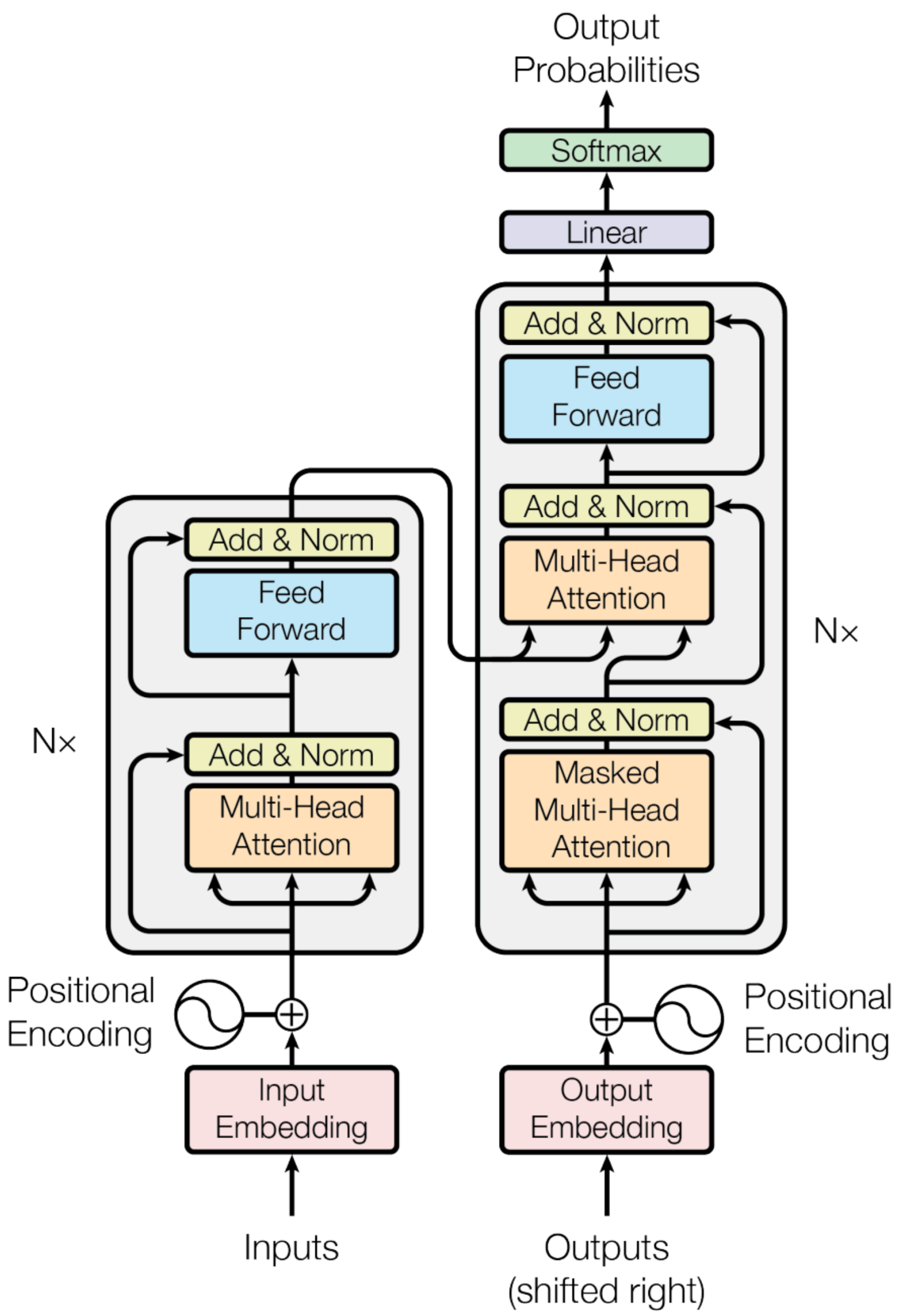


Figure 1: The Transformer - model architecture.

Three variants of Attention

Self-attention

- keys, queries, values come from the *same source x*
- each token attends to *all tokens* in the input sequence

-> e.g. sentiment analysis

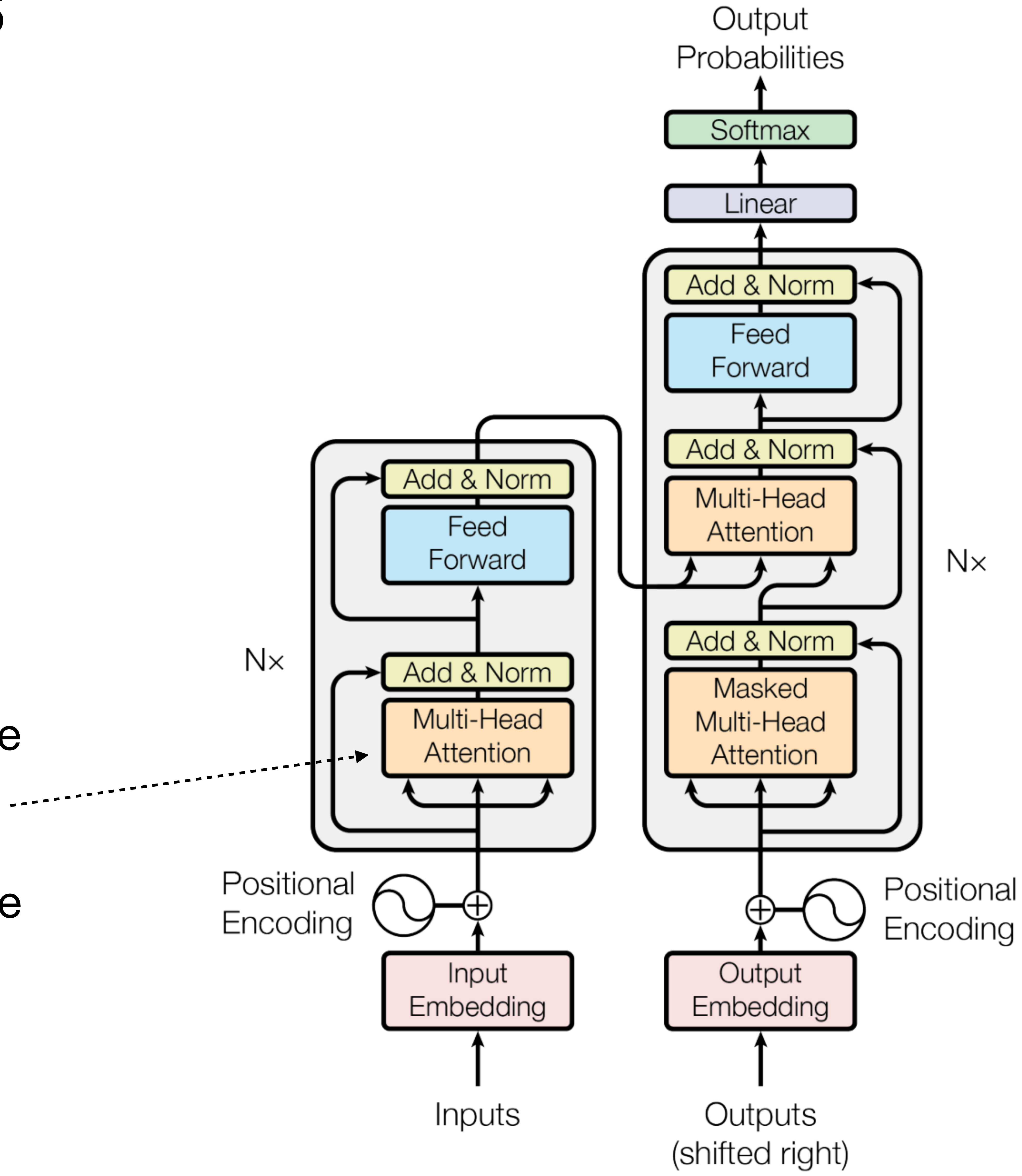


Figure 1: The Transformer - model architecture.

Three variants of Attention

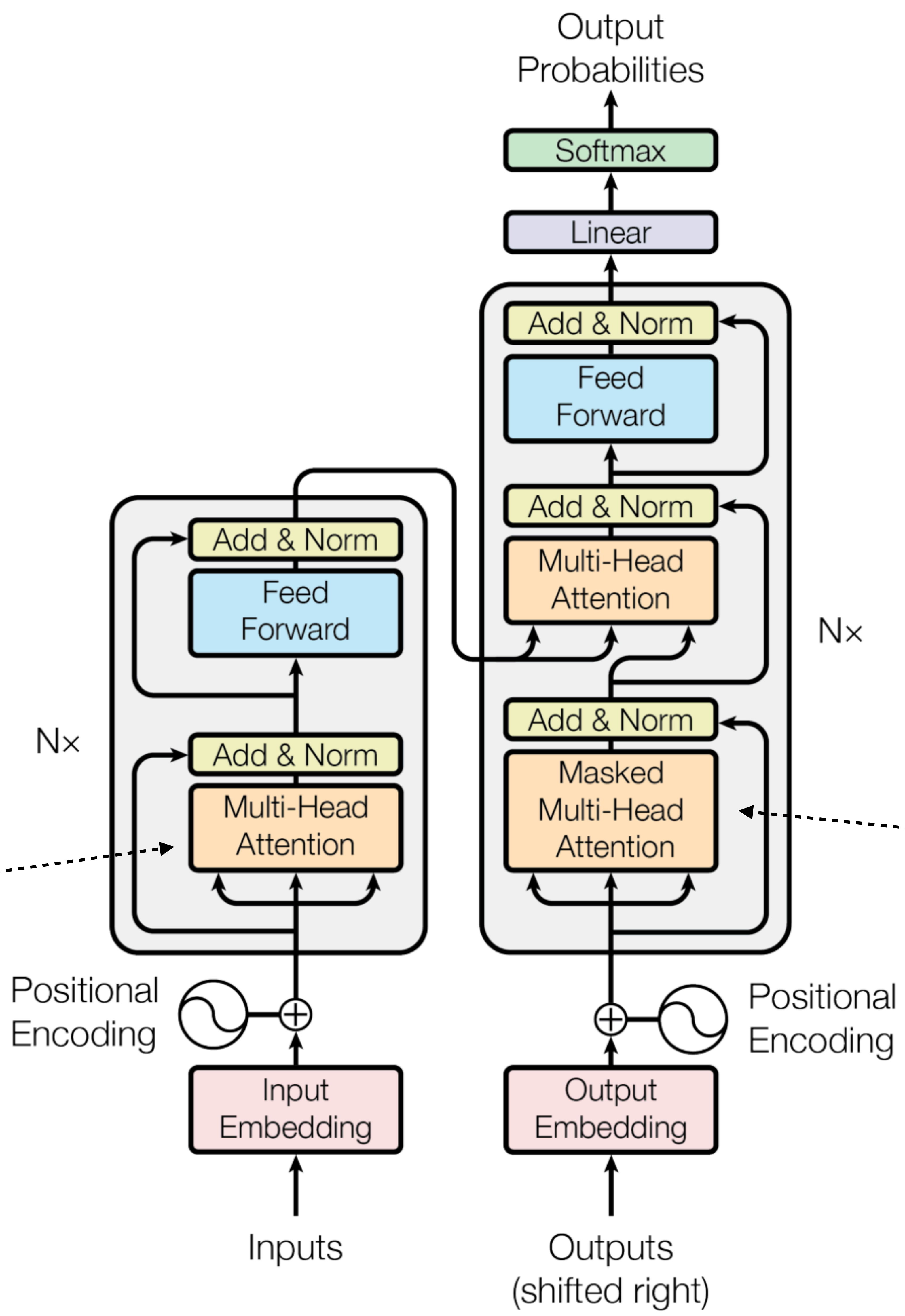


Figure 1: The Transformer - model architecture.

Self-attention

- keys, queries, values come from the *same source x*
- each token attends to *all tokens* in the input sequence

-> e.g. sentiment analysis

Masked Self-attention

- keys, queries, values come from the *same source x*
- each token attends to all *previous tokens* in the input sequence

-> e.g. text generation

Three variants of Attention

Self-attention

- keys, queries, values come from the *same source x*
- each token attends to *all tokens* in the input sequence

-> e.g. sentiment analysis

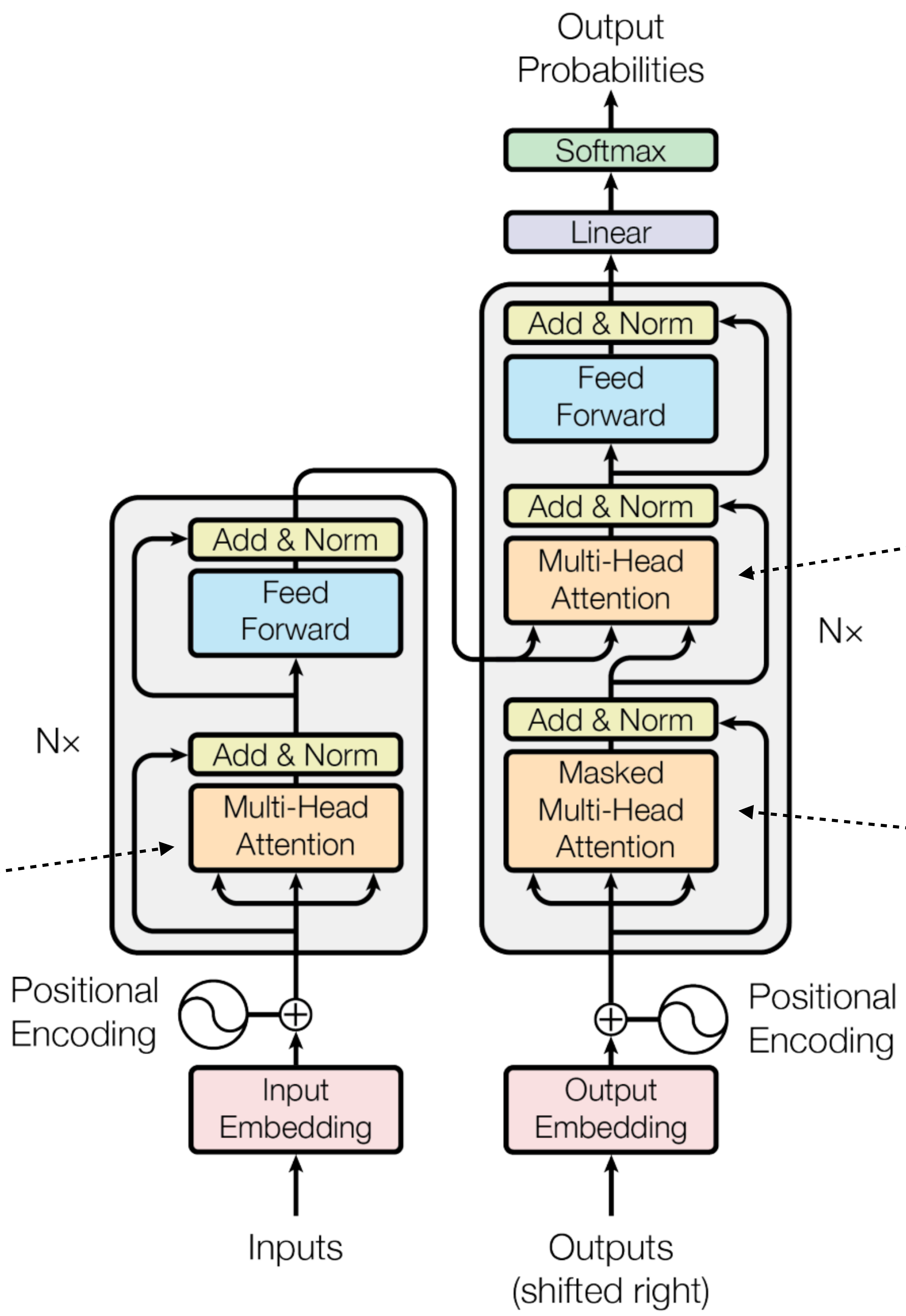


Figure 1: The Transformer - model architecture.

Cross-attention

- query from decoder, keys/values from encoder
- Each token attends to *all tokens* in sequence

Masked Self-attention

- keys, queries, values come from the *same source x*
- each token attends to all *previous tokens* in the input sequence

-> e.g. text generation

Summary

- Attention captures dependencies between words in a sequence, regardless of their distance

Summary

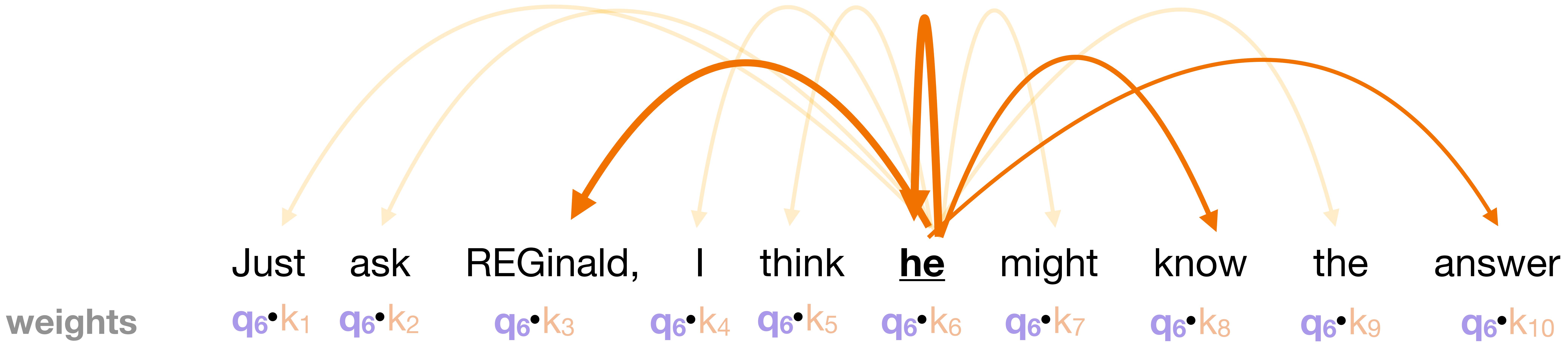
- Attention captures dependencies between words in a sequence, regardless of their distance

- **Query** **Key** dot-products produce attention scores / weights

Summary

- Attention captures dependencies between words in a sequence, regardless of their distance

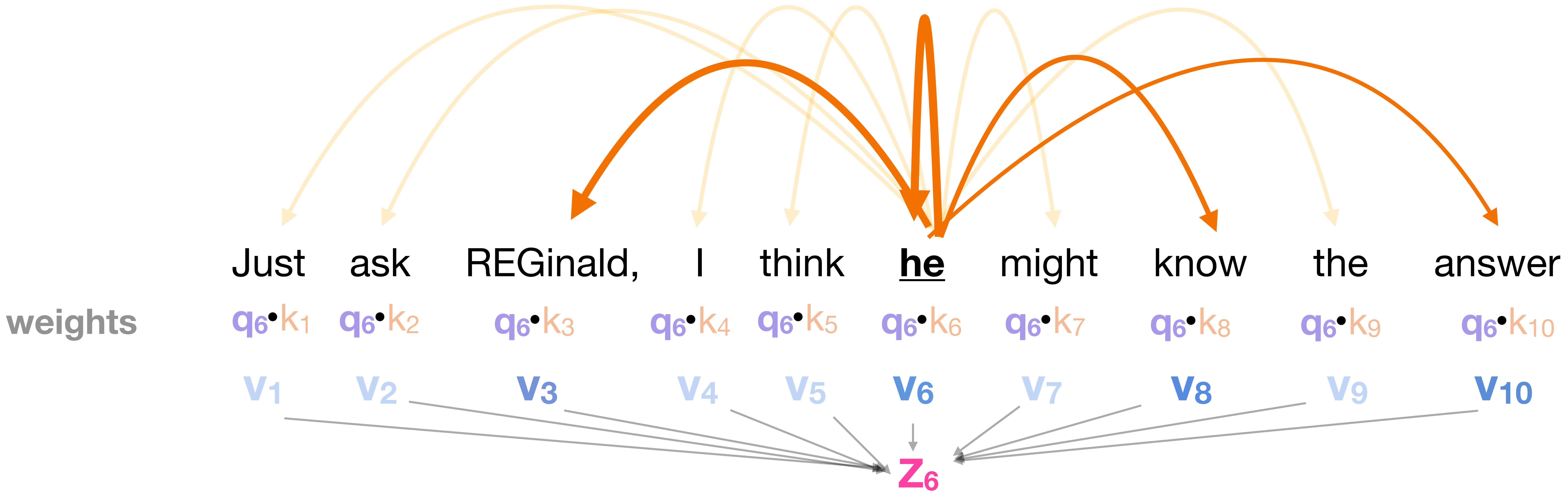
- Query Key dot-products produce attention scores / weights



Summary

- Attention captures dependencies between words in a sequence, regardless of their distance

- **Query Key** dot-products produce attention scores / weights



- Attention output **z** is a sum of weighted **values**

Notes

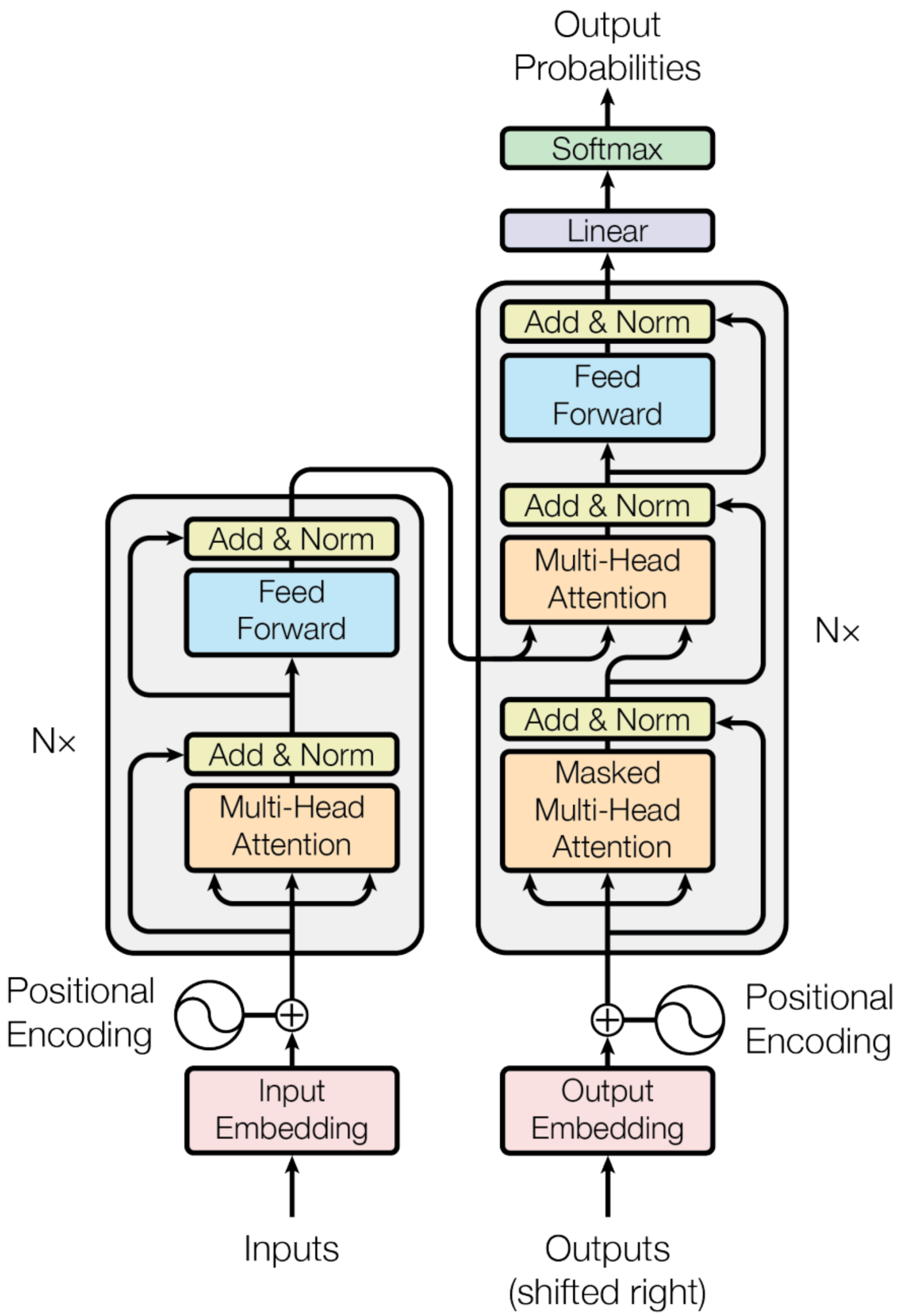


Figure 1: The Transformer - model architecture.

* https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c_fvtXnx-?usp=sharing

Notes

- Attention is indifferent to the *position* of words, it simply acts over a set of vectors

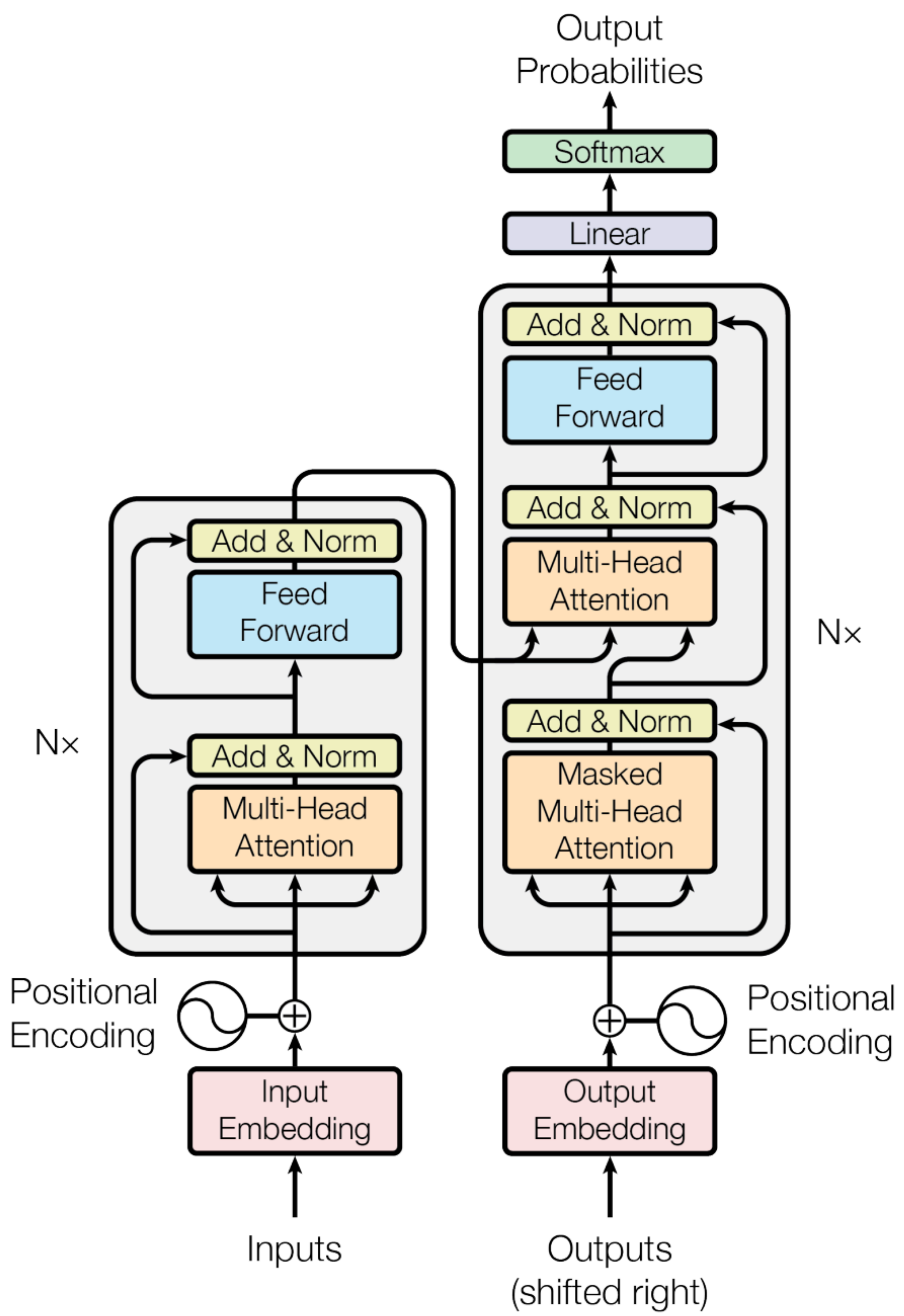


Figure 1: The Transformer - model architecture.

* https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c_fvtXnx-?usp=sharing

Notes

- Attention is indifferent to the *position* of words, it simply acts over a set of vectors
  - -> Needs positional encoders

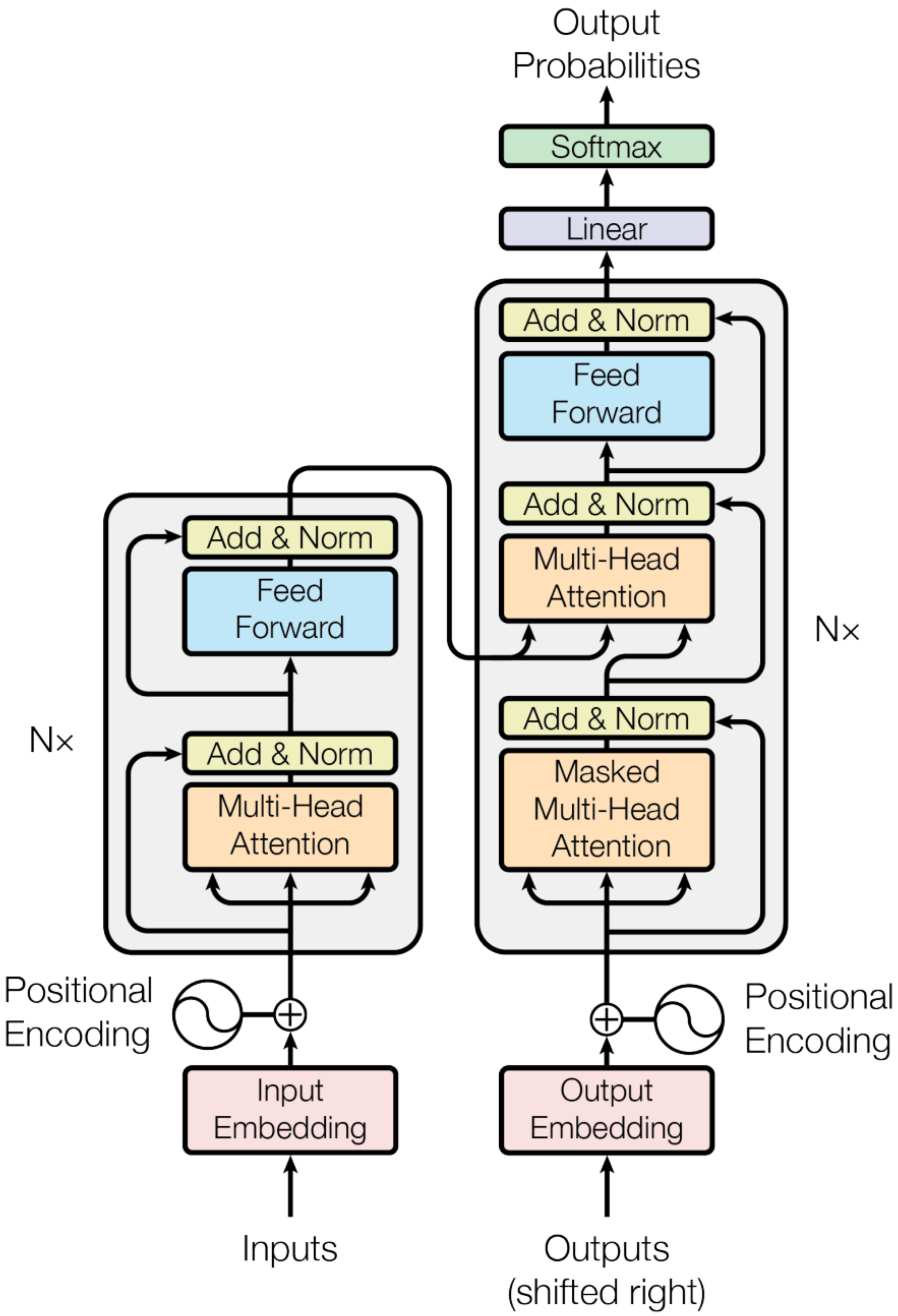


Figure 1: The Transformer - model architecture.

* https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c_fvtXnx-?usp=sharing

Notes

- Attention is indifferent to the *position* of words, it simply acts over a set of vectors
  - -> Needs positional encoders
- Attention weights are data-dependent and *change during runtime* (unlike Feed Forward NNs / MLPs)

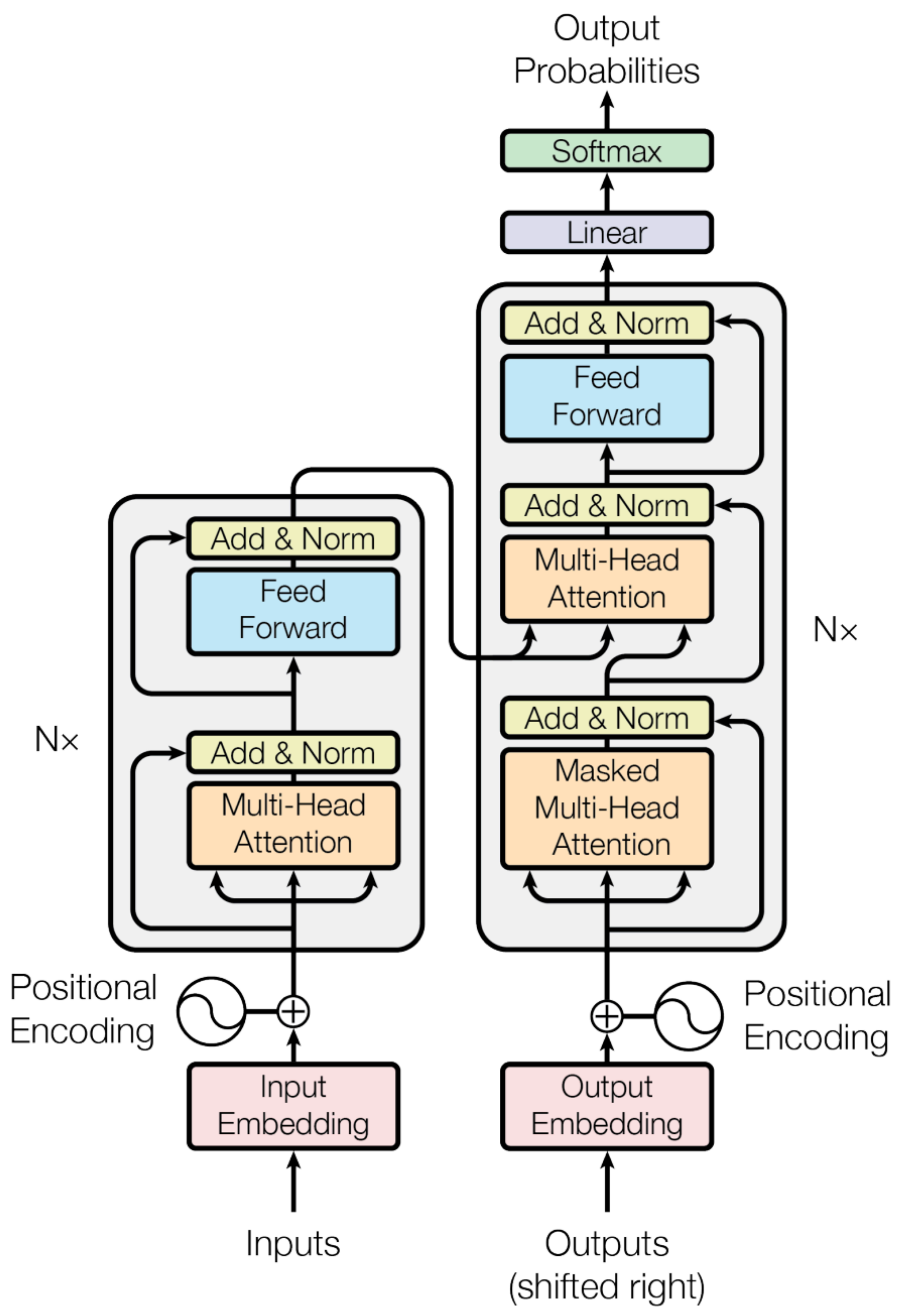


Figure 1: The Transformer - model architecture.

* https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c_fvtXnx-?usp=sharing

Notes

- Attention is indifferent to the *position* of words, it simply acts over a set of vectors
  - -> Needs positional encoders
- Attention weights are data-dependent and *change during runtime* (unlike Feed Forward NNs / MLPs)
- No communication *between batches* of data

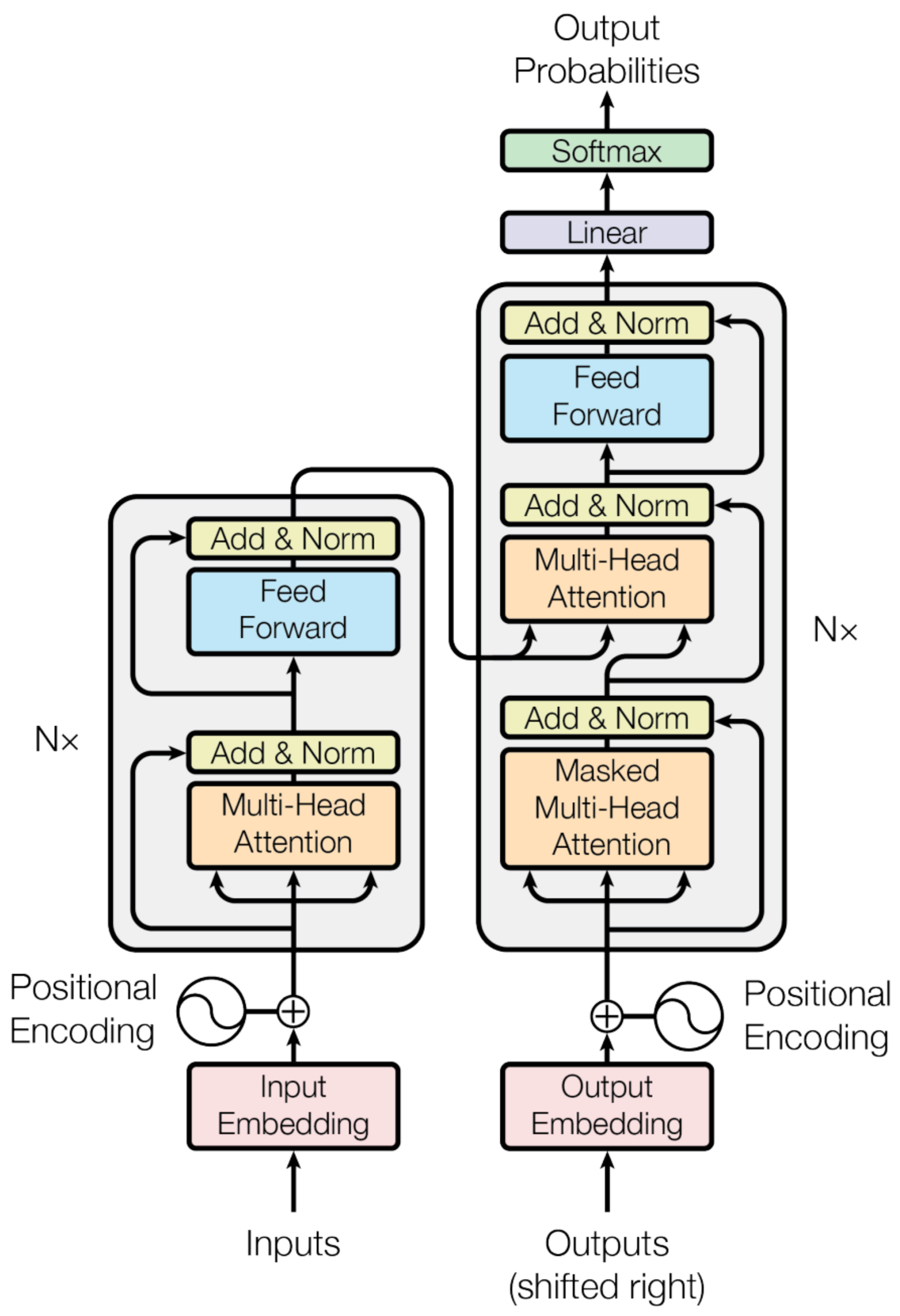


Figure 1: The Transformer - model architecture.

* https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c_fvtXnx-?usp=sharing

Notes

- Attention is indifferent to the *position* of words, it simply acts over a set of vectors
  - -> Needs positional encoders
- Attention weights are data-dependent and *change during runtime* (unlike Feed Forward NNs / MLPs)
- No communication *between batches* of data
- Attention is a general communication mechanism, tokens can be seen as nodes in a directed graph*

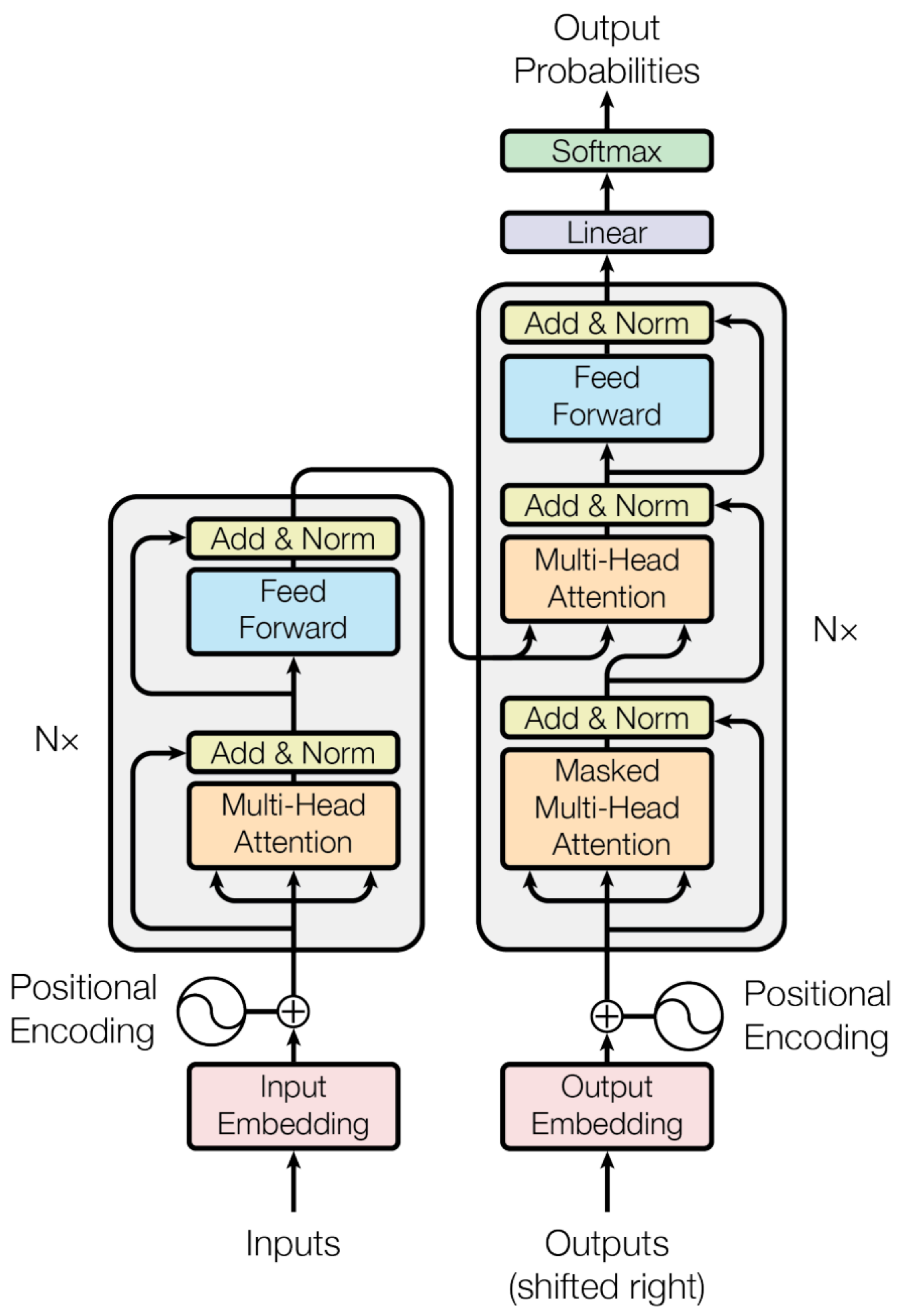


Figure 1: The Transformer - model architecture.

* https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c_fvtXnx-?usp=sharing

Appendix

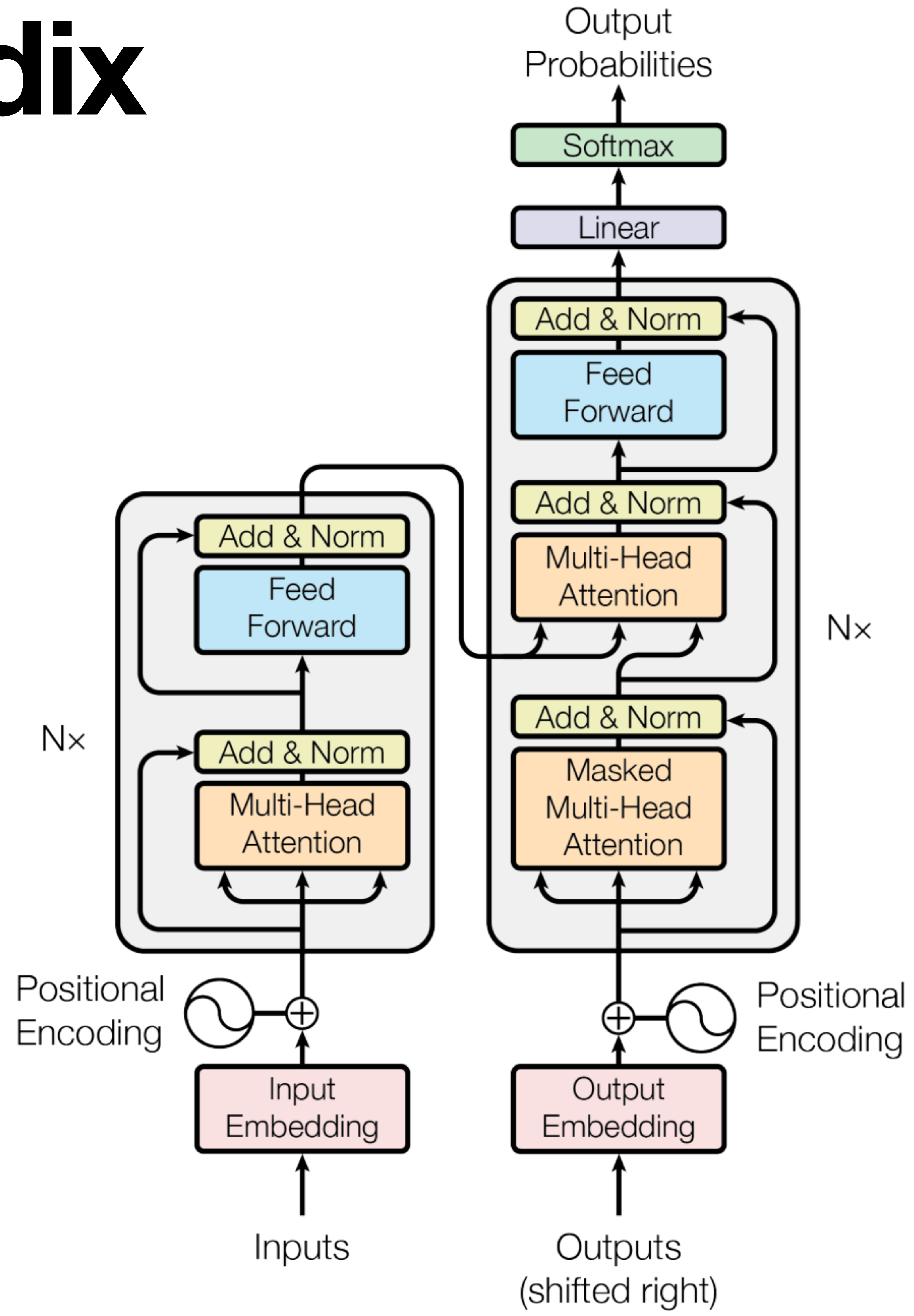
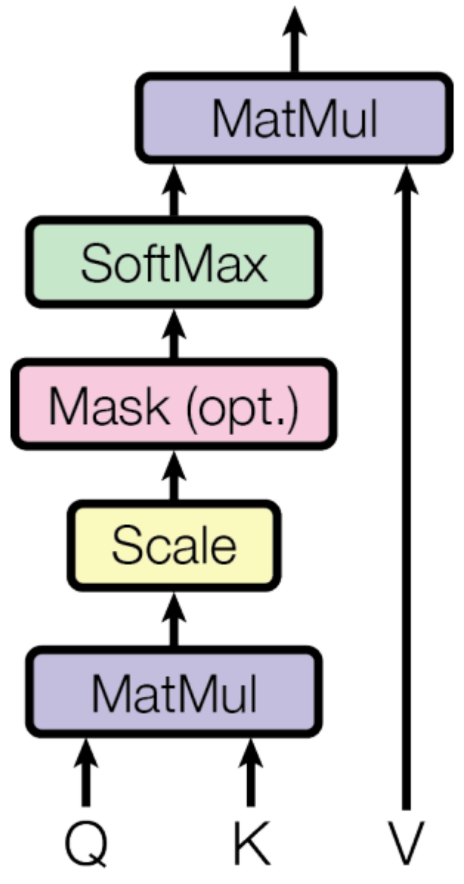


Figure 1: The Transformer - model architecture.

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention



Multi-Head Attention