COMP 3361 Natural Language Processing

Lecture 9: Attention and Transformers

Spring 2024

Many materials from CSE447@UW (Liwei Jiang), COS 484@Princeton, and CS224n@Stanford with special thanks!

Transformers

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

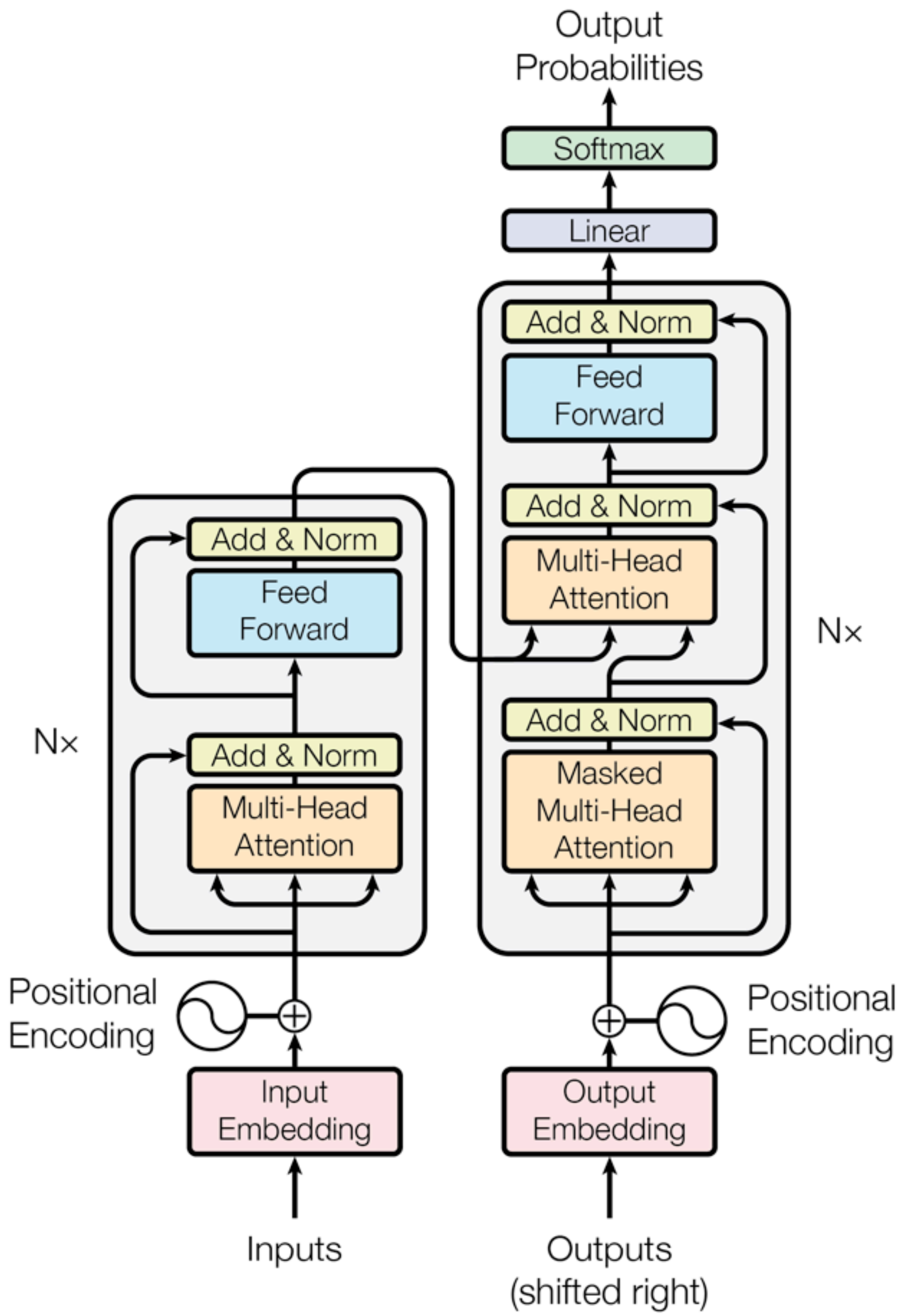
Aidan N. Gomez[*, †]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*, ‡]
illia.polosukhin@gmail.com

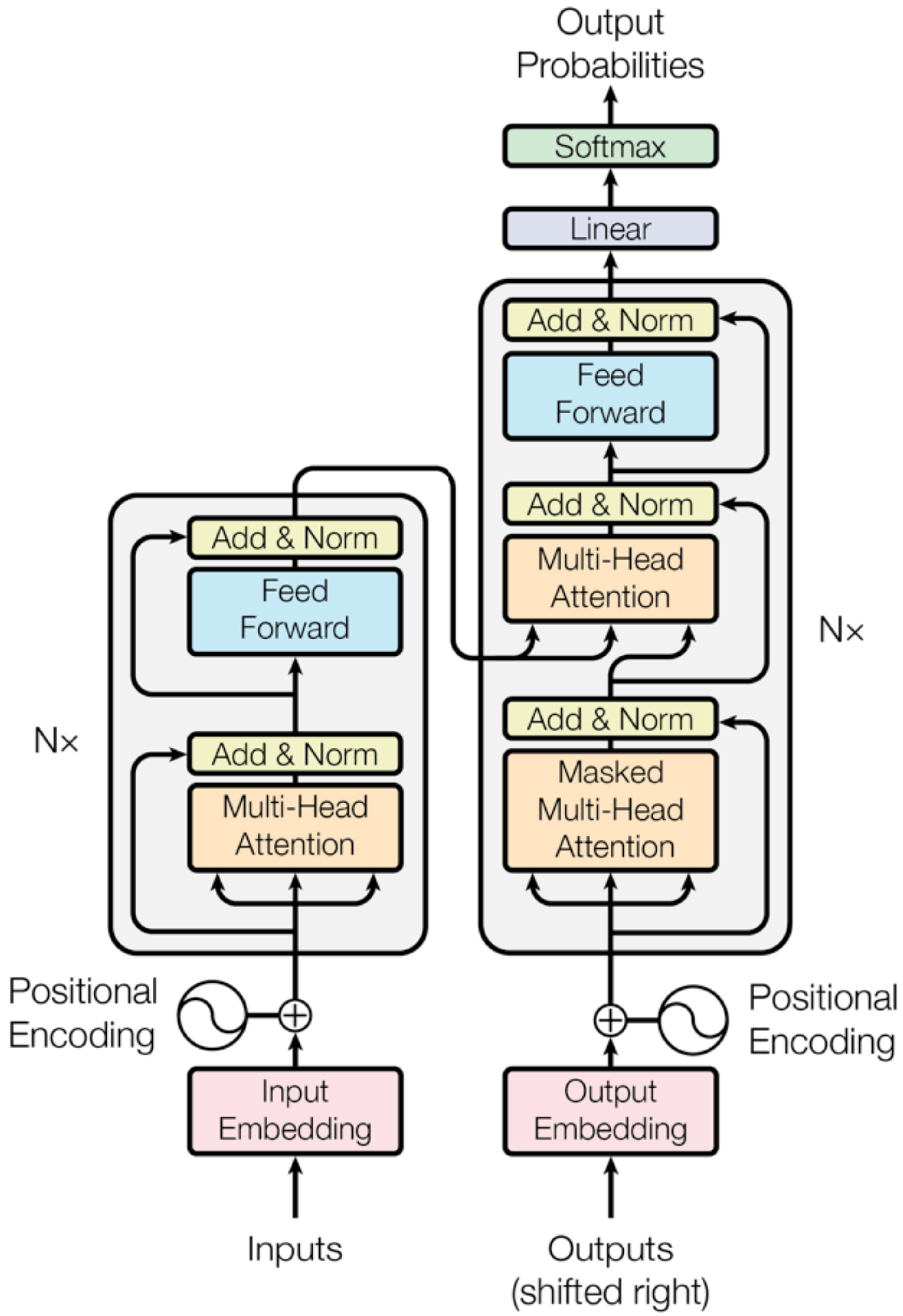
(Vaswani et al., 2017)

Transformer encoder-decoder



- Transformer encoder + Transformer decoder
- First designed and experimented on NMT

Transformer encoder-decoder



- Transformer encoder = a stack of **encoder layers**
- Transformer decoder = a stack of **decoder layers**

Transformer encoder: BERT, RoBERTa, ELECTRA

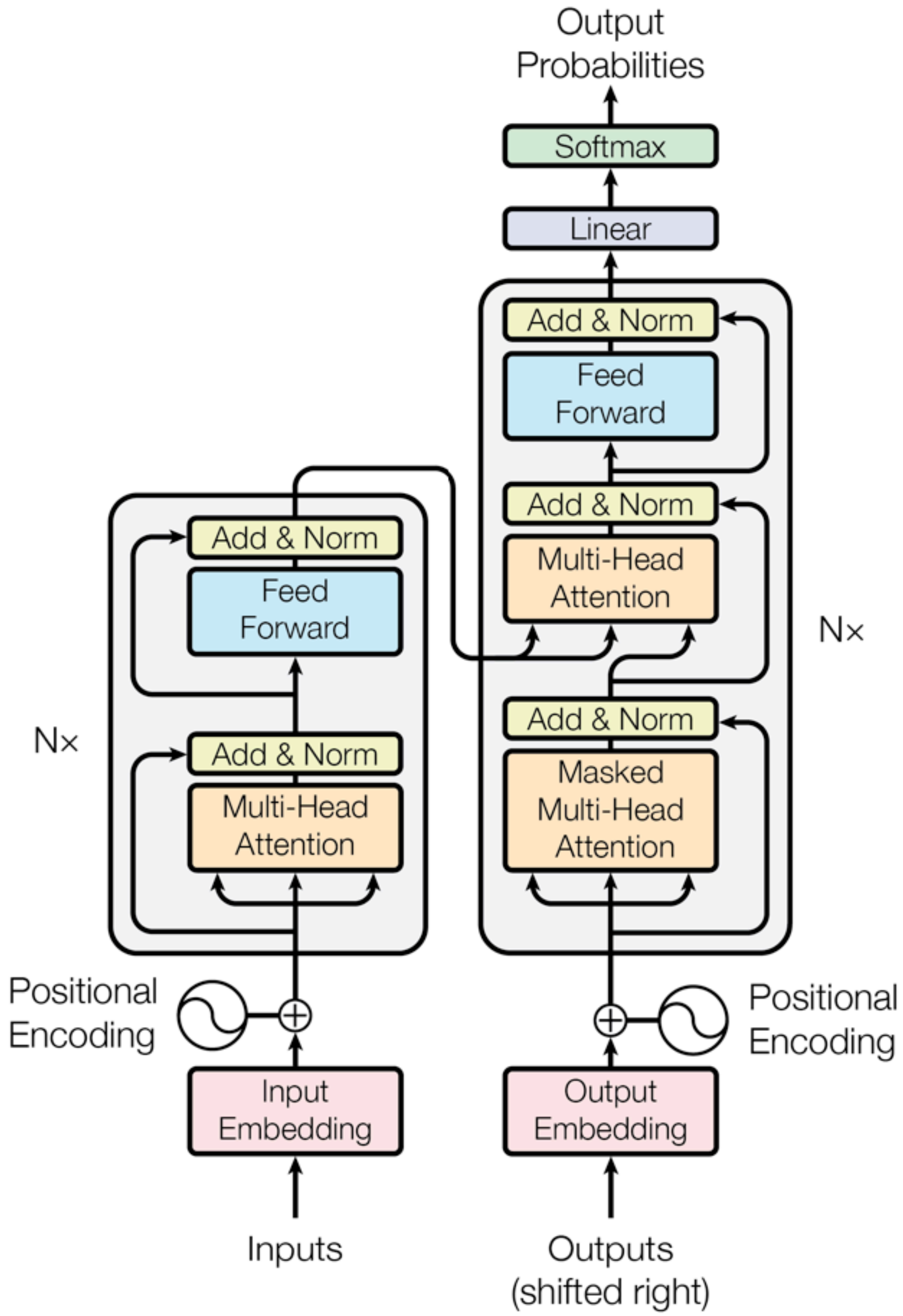
Transformer decoder: GPT-3, ChatGPT, Palm

Transformer encoder-decoder: T5, BART

- Key innovation: **multi-head, self-attention**
- Transformers don't have any recurrence structures!

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^h$$

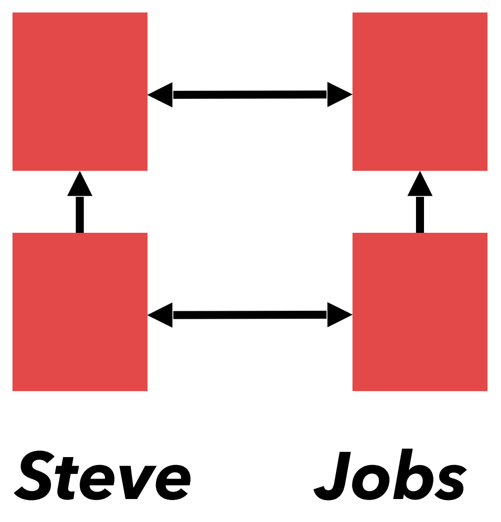
Transformers: roadmap



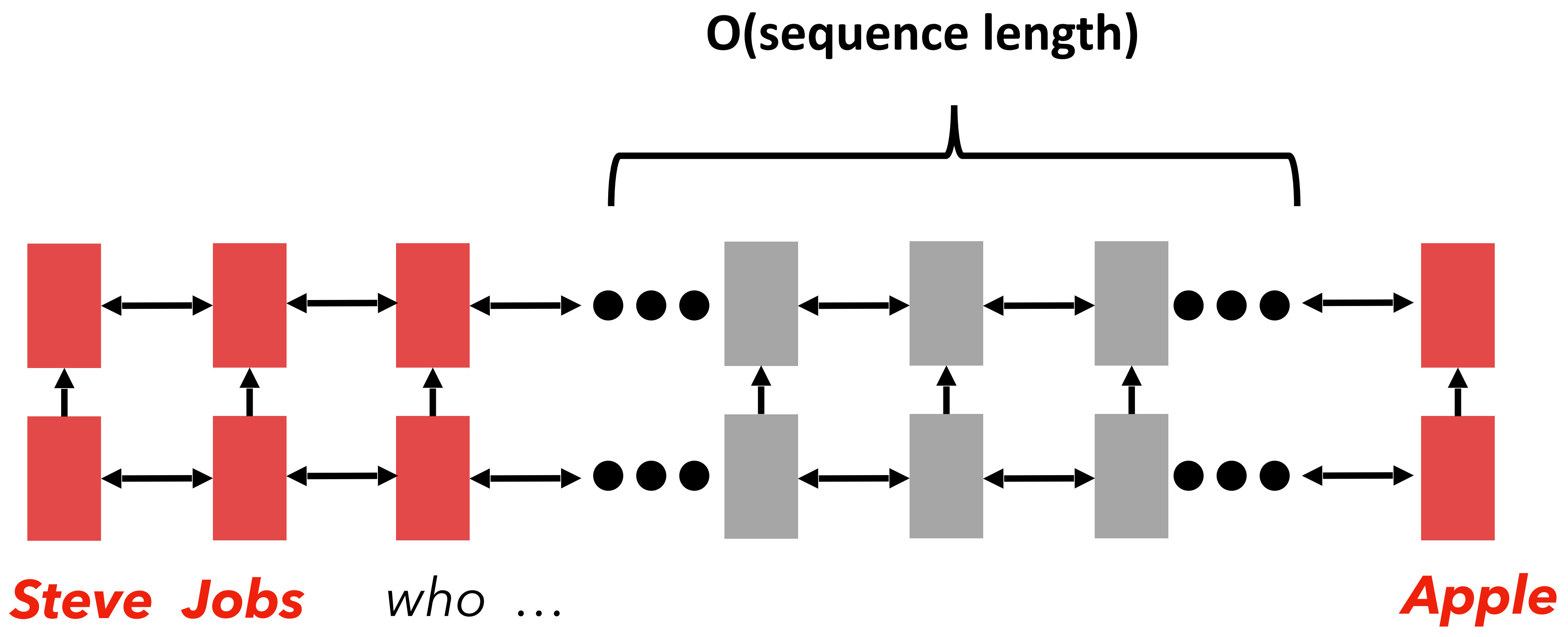
- From attention to self-attention
- From self-attention to multi-head self-attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

Issues with RNNs: Linear Interaction Distance

- RNNs are unrolled **left-to-right**.
  - **Linear locality** is a useful heuristic: nearby words often affect each other's meaning!



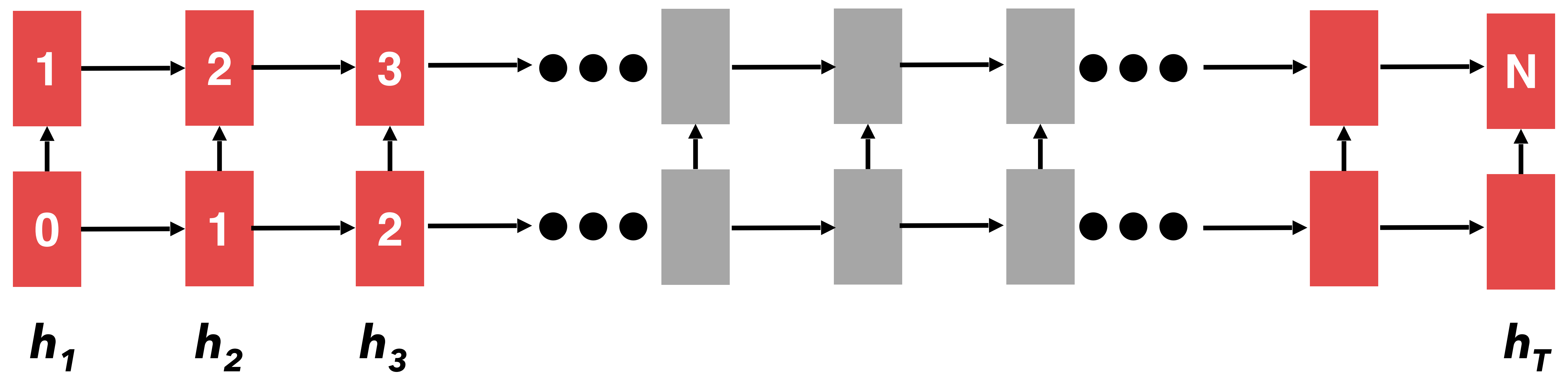
- However, there's the **vanishing gradient problem** for long sequences.
  - The gradients that are used to update the network become extremely small or "vanish" as they are backpropogated from the output layers to the earlier layers.



Failing to capture **long-term dependences**.

Issues with RNNs: Lack of Parallelizability

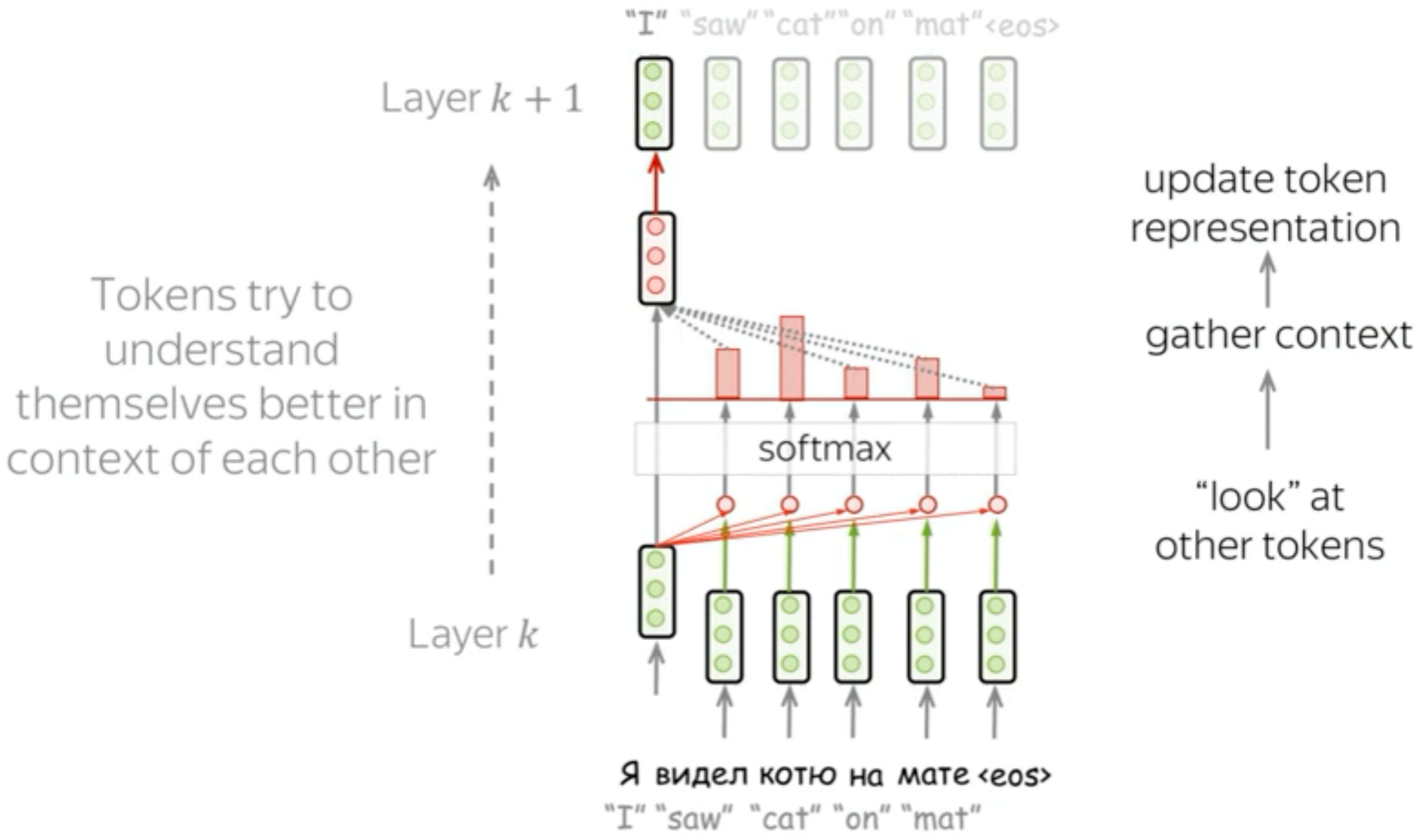
- Forward and backward passes have **O(sequence length)** **unparallelizable** operations
  - GPUs can perform many independent computations (like addition) at once!
  - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed.
  - Training and inference are slow; inhibits on very large datasets!



Numbers indicate min # of steps before a state can be computed

The New De Facto Method: Attention

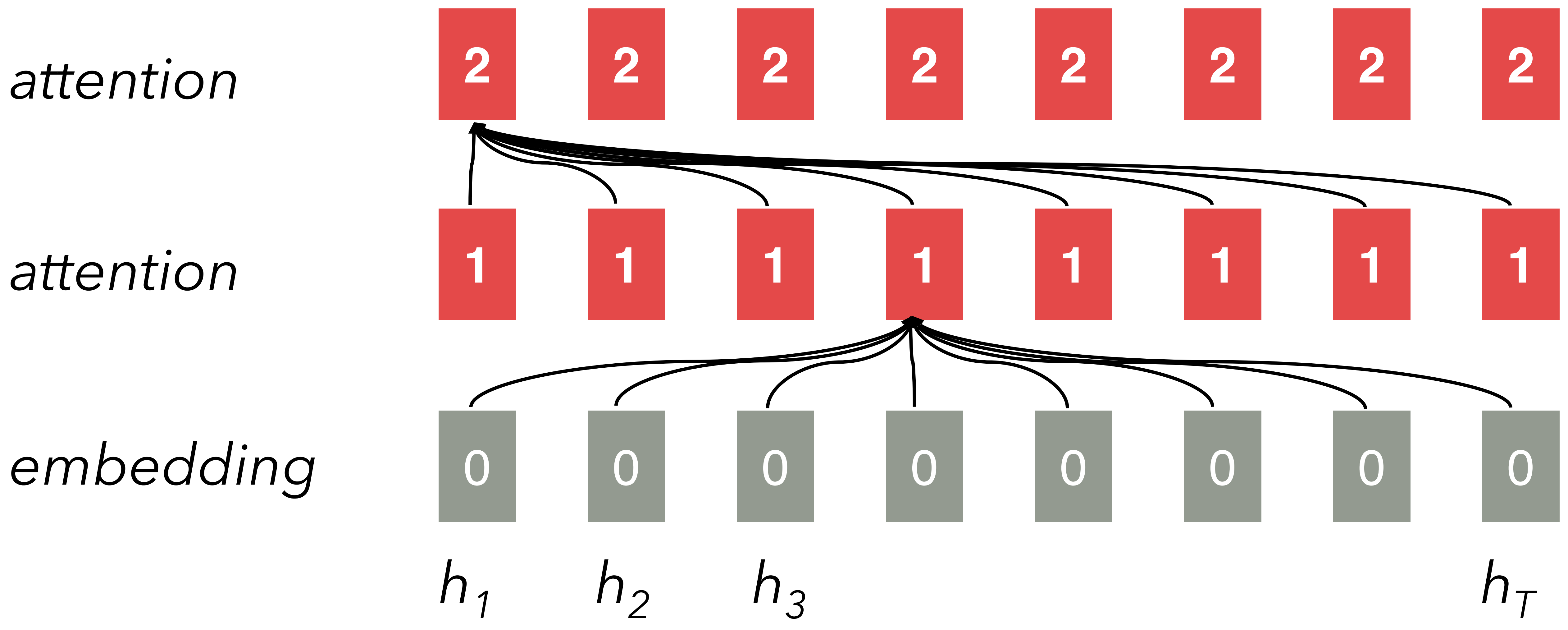
Instead of deciding the next token solely based on the previously seen tokens, **each token will "look at" all input tokens at the same to decide which ones are most important** to decide the next token.



In practice, the actions of all tokens are done in parallel!

Building the Intuition of Attention

- **Attention** treats each token's representation as a **query** to access and incorporate information from **a set of values**.
  - Today we look at attention **within a single sequence**.
- Number of unparallelizable operations does **NOT** increase with sequence length.
- Maximum interaction distance: O(1), since all tokens interact at every layer!

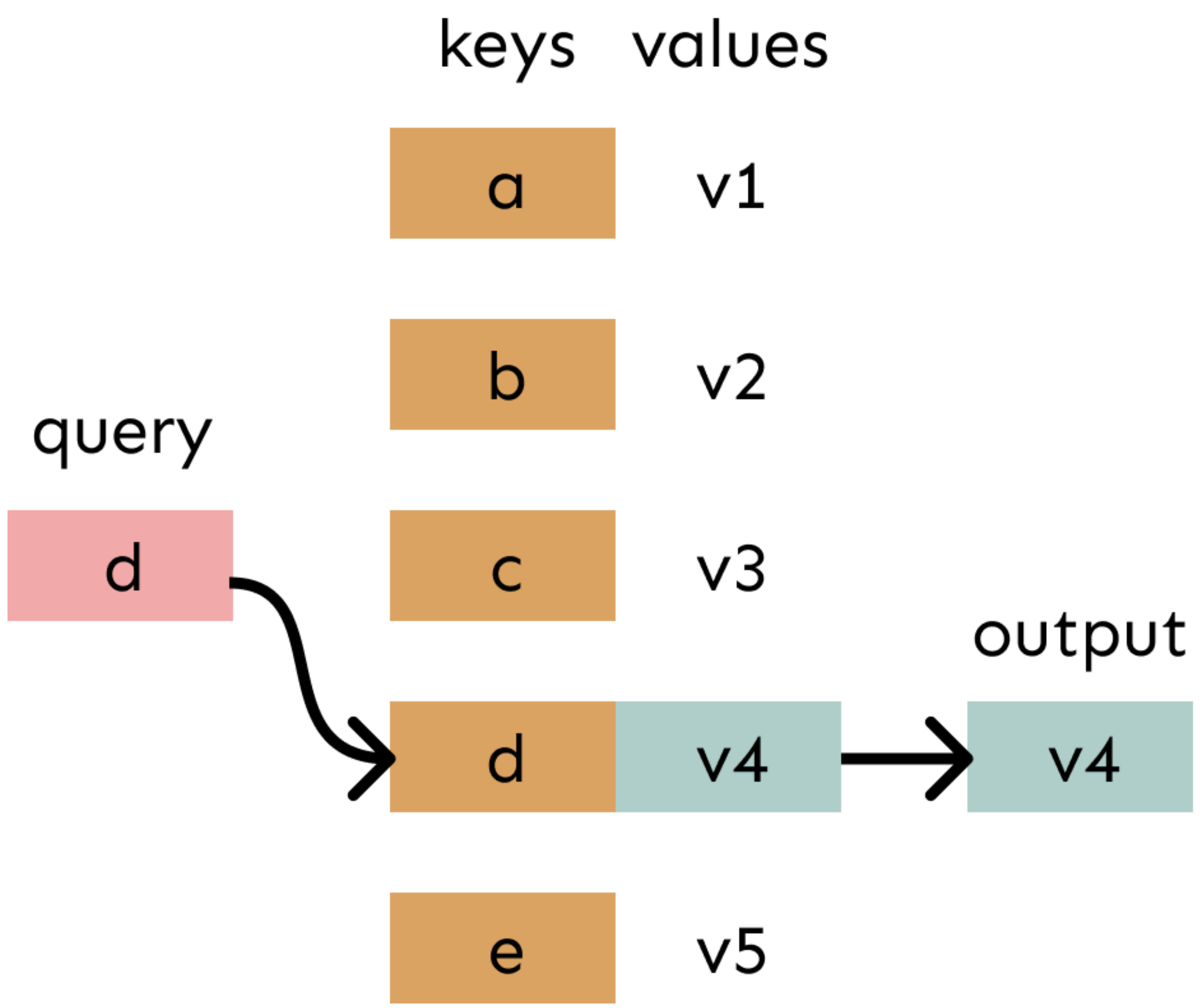


All tokens attend to all tokens in previous layer; most arrows here are omitted

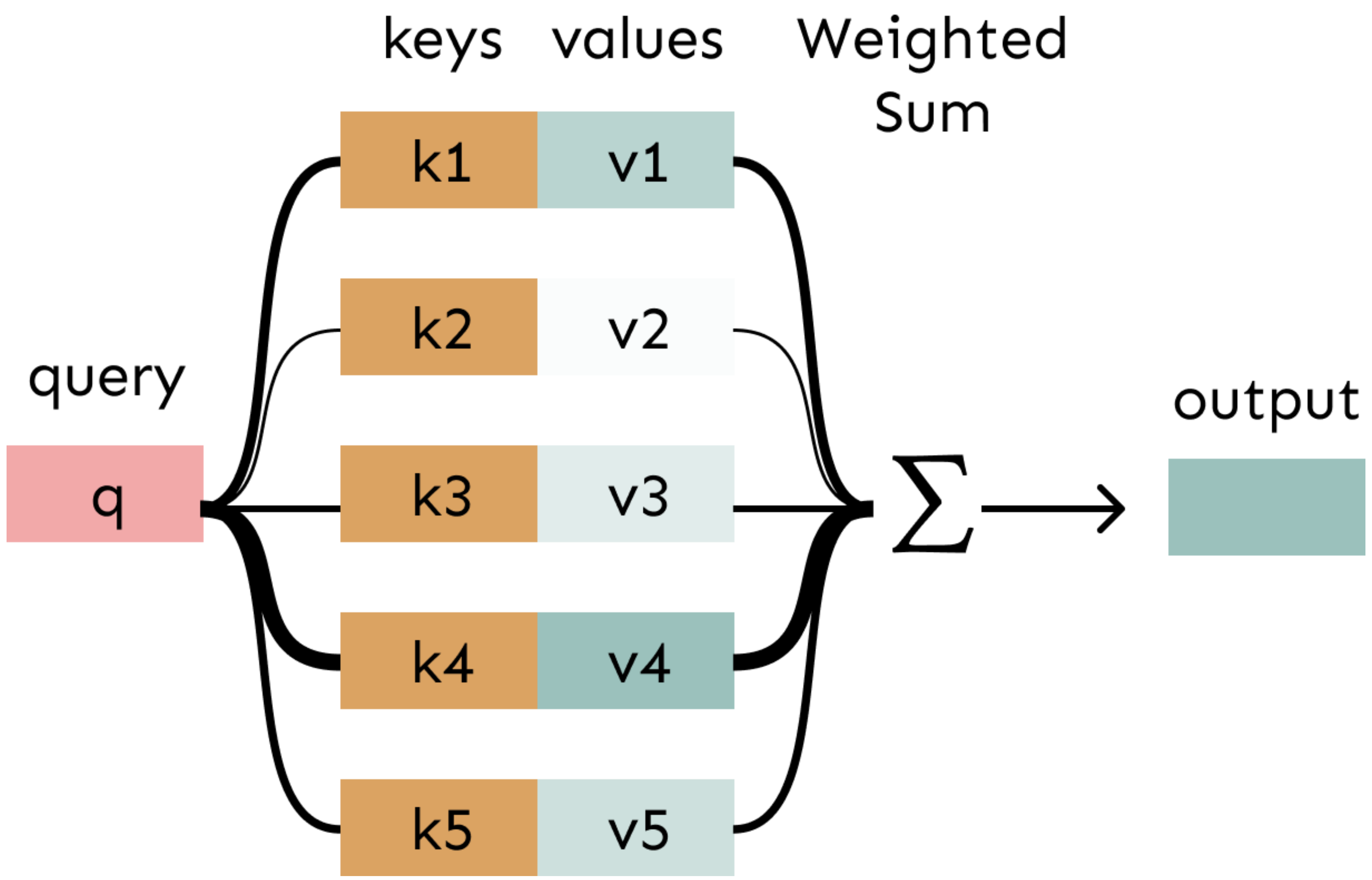
Attention as a soft, averaging lookup table

We can think of **attention** as performing **fuzzy lookup** in a key-value store.

In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



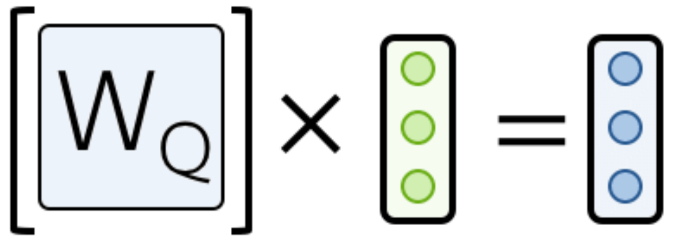
In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



Self-Attention: Basic Concepts

[Lena Viota Blog]

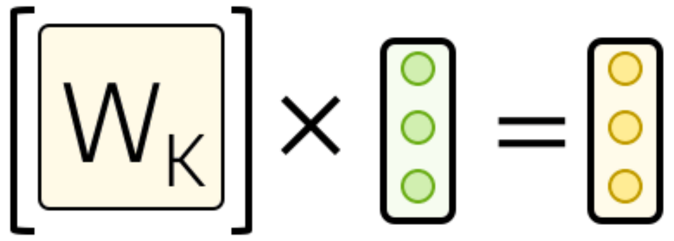
Each vector receives t



Qu y: vector **from** which the atte

“Hey there, do you have

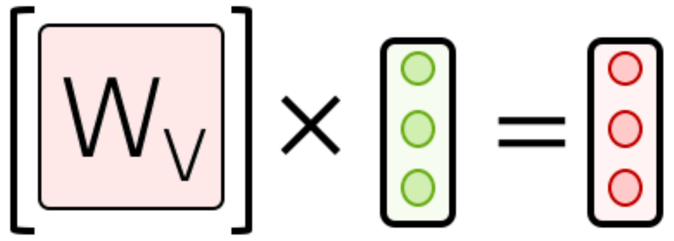
Query: asking for information



Key: v ctor **at** which the query looks to co

“Hi, I have this informa

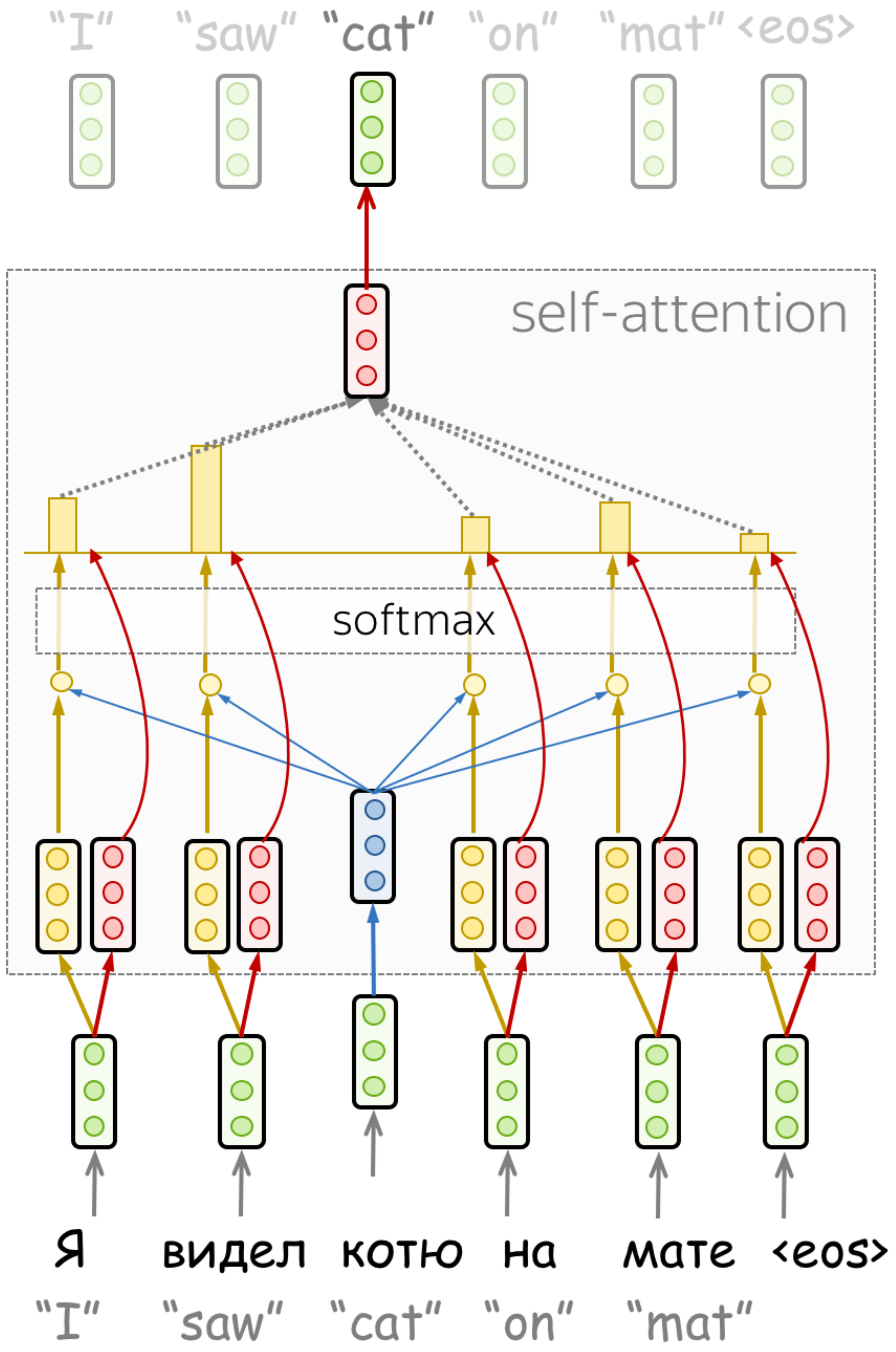
Key: saying that it has some information



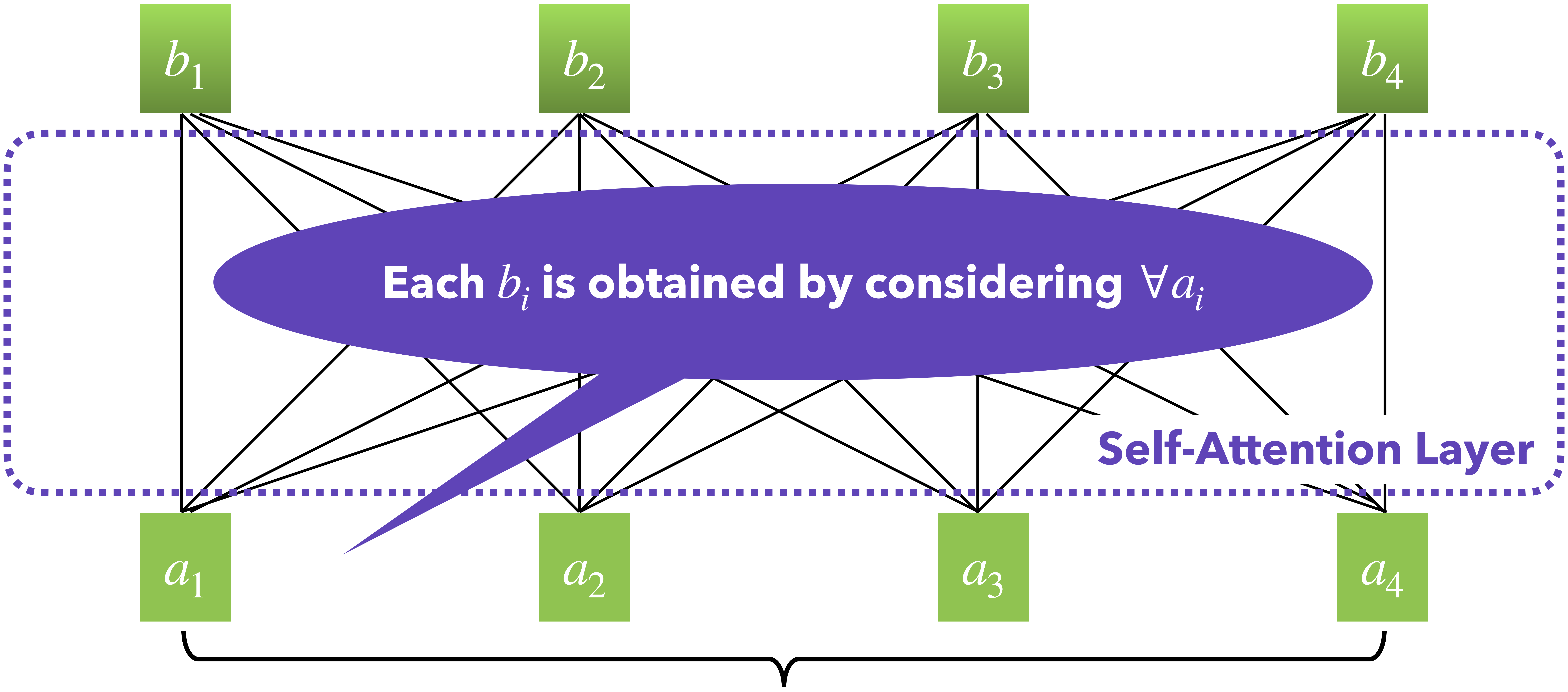
Value:
att tion output

“Here's the information I have!”

Value: giving the information



Self-Attention: Walk-through

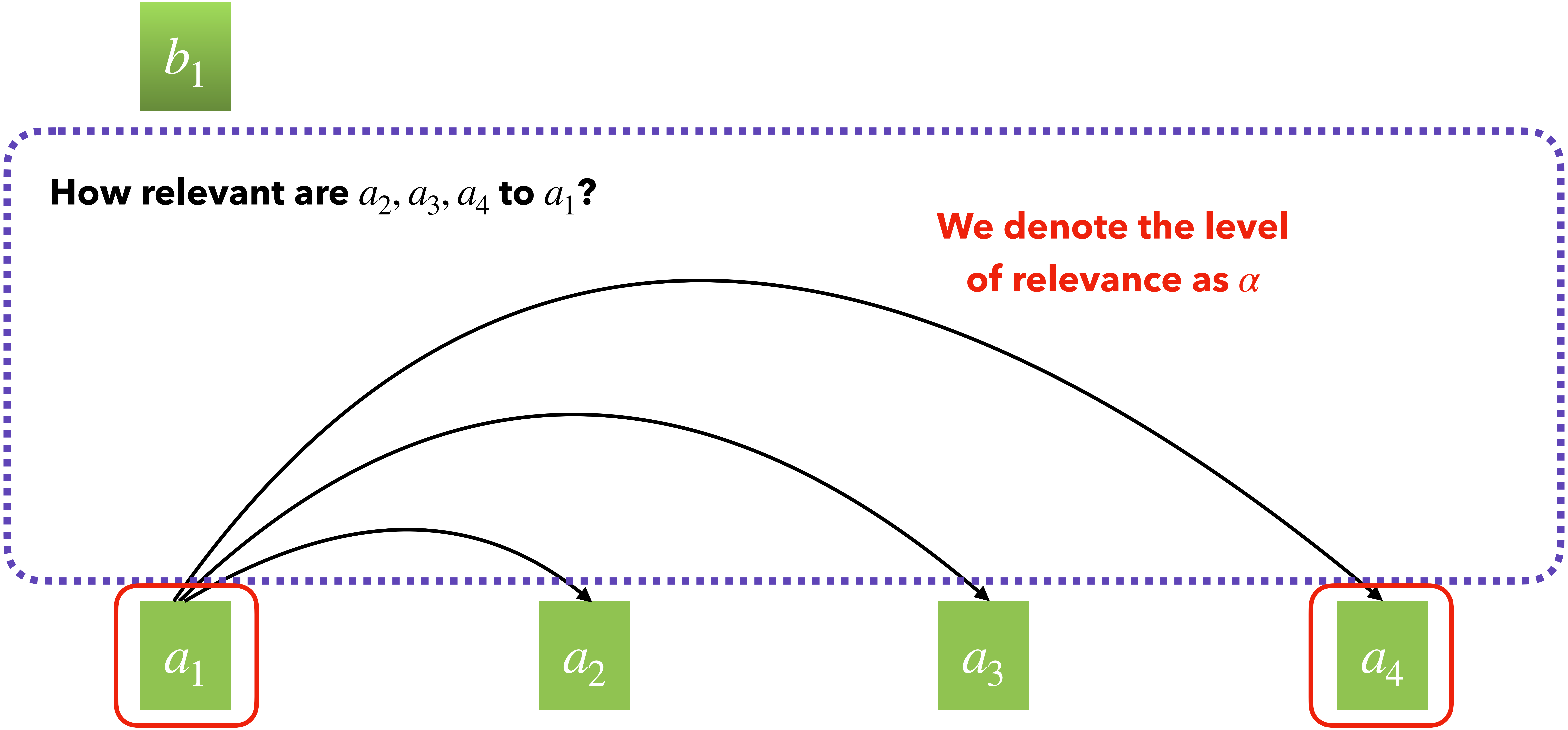


Each b_i is obtained by considering $\forall a_i$

Self-Attention Layer

Can be either **input** or a **hidden layer**

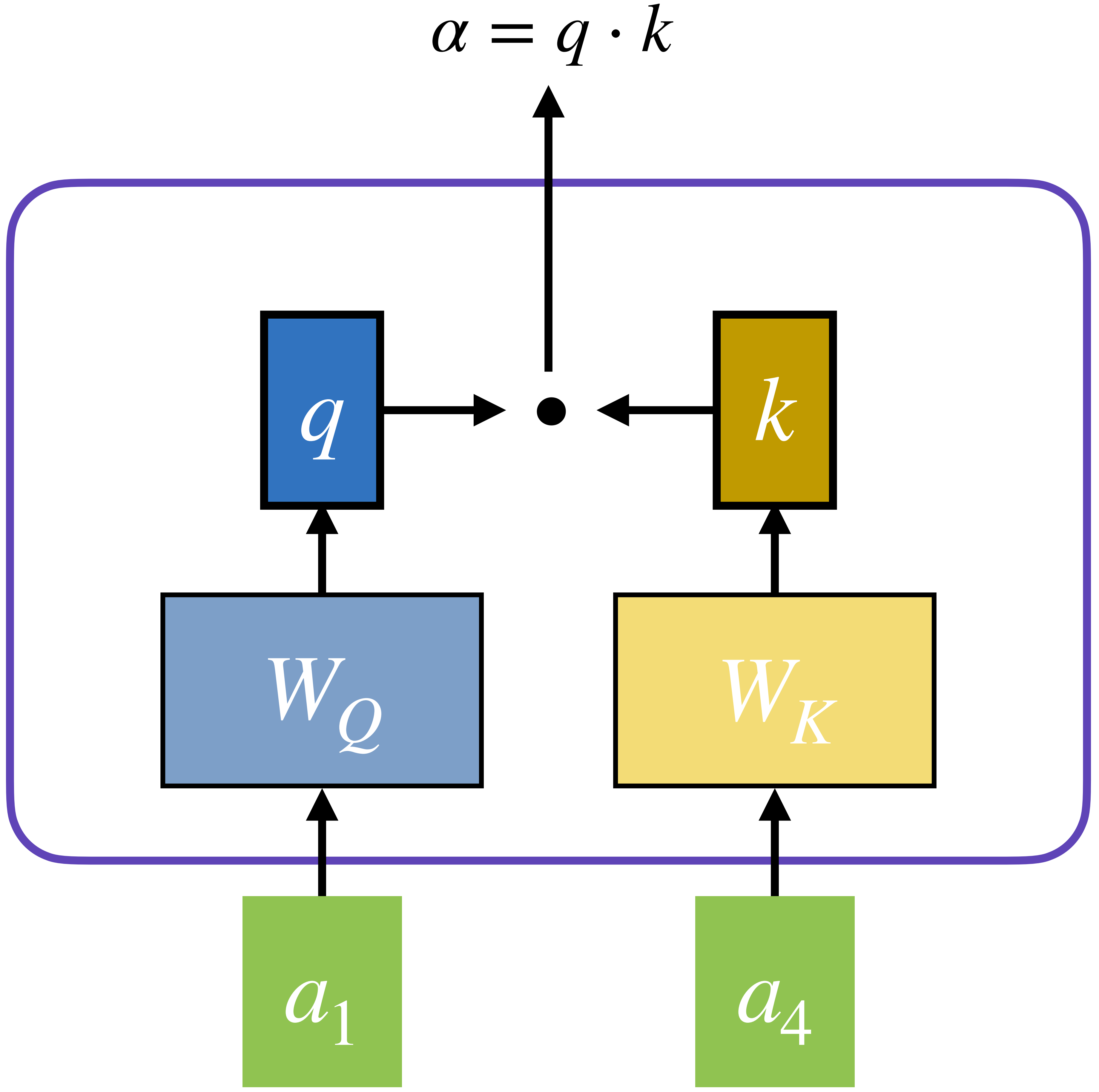
Self-Attention: Walk-through



How relevant are a_2, a_3, a_4 to a_1?

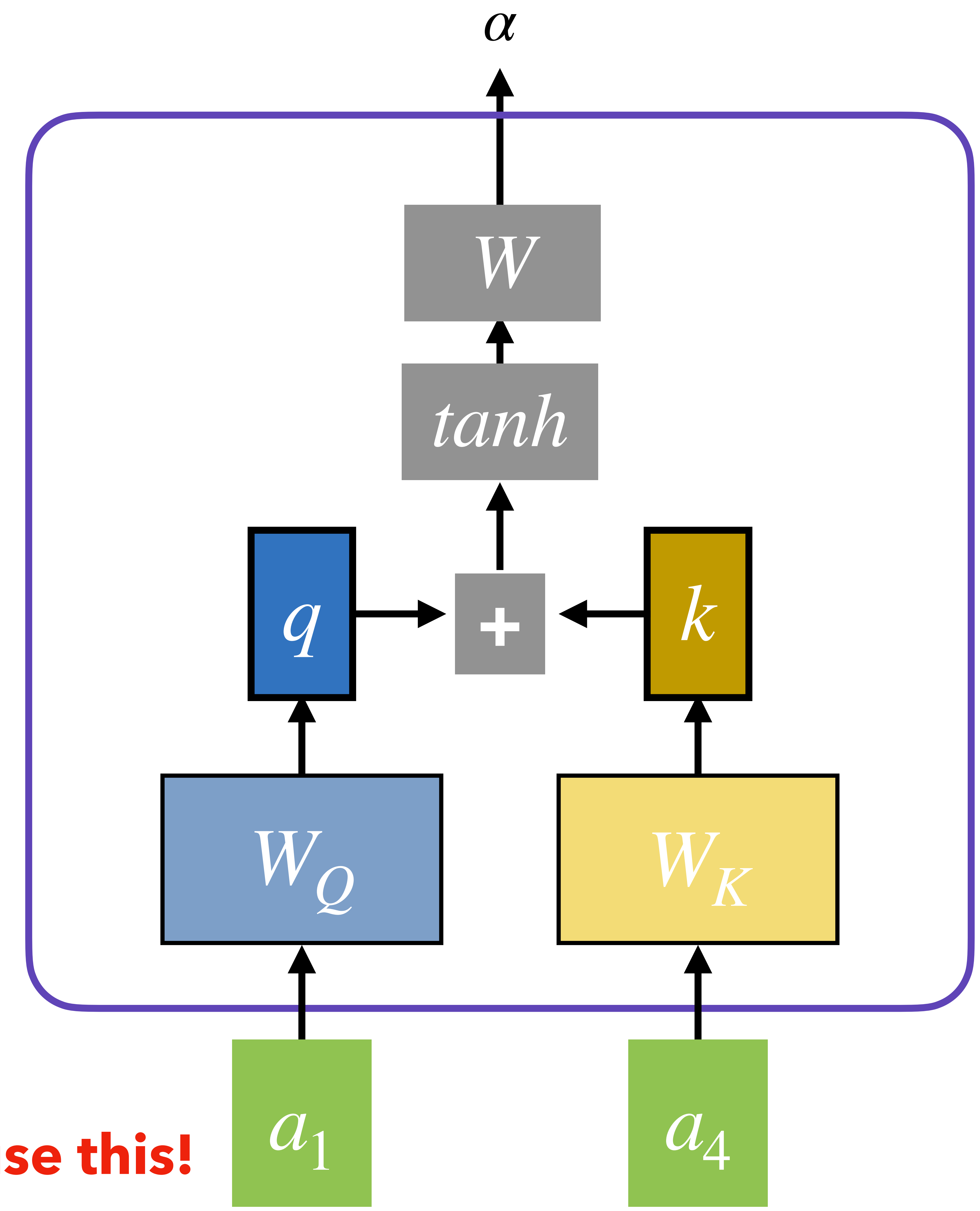
We denote the level of relevance as α

How to compute α?



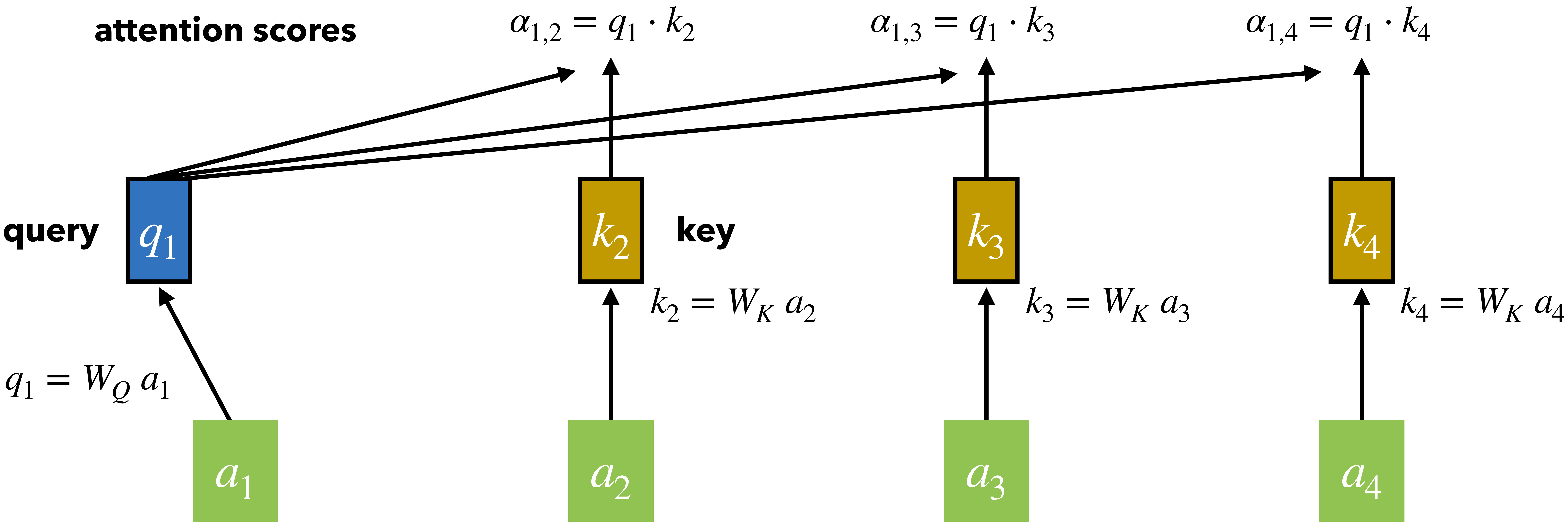
Method 1 **(most common): Dot product**

We'll use this!



Method 2: **Additive**

Self-Attention: Walk-through



attention scores

$\alpha_{1,2} = q_1 \cdot k_2$

$\alpha_{1,3} = q_1 \cdot k_3$

$\alpha_{1,4} = q_1 \cdot k_4$

query

key

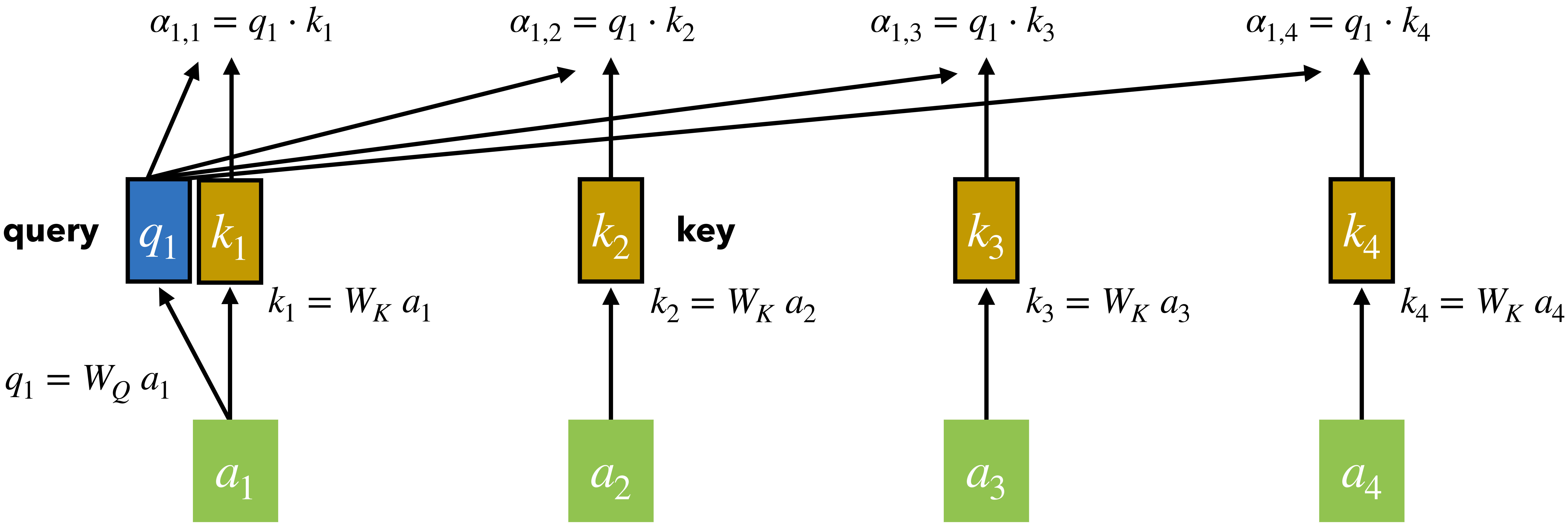
$q_1 = W_Q\, a_1$

$k_2 = W_K\, a_2$

$k_3 = W_K\, a_3$

$k_4 = W_K\, a_4$

Self-Attention: Walk-through



$\alpha_{1,1} = q_1 \cdot k_1$

$\alpha_{1,2} = q_1 \cdot k_2$

$\alpha_{1,3} = q_1 \cdot k_3$

$\alpha_{1,4} = q_1 \cdot k_4$

query

key

$k_1 = W_K\, a_1$

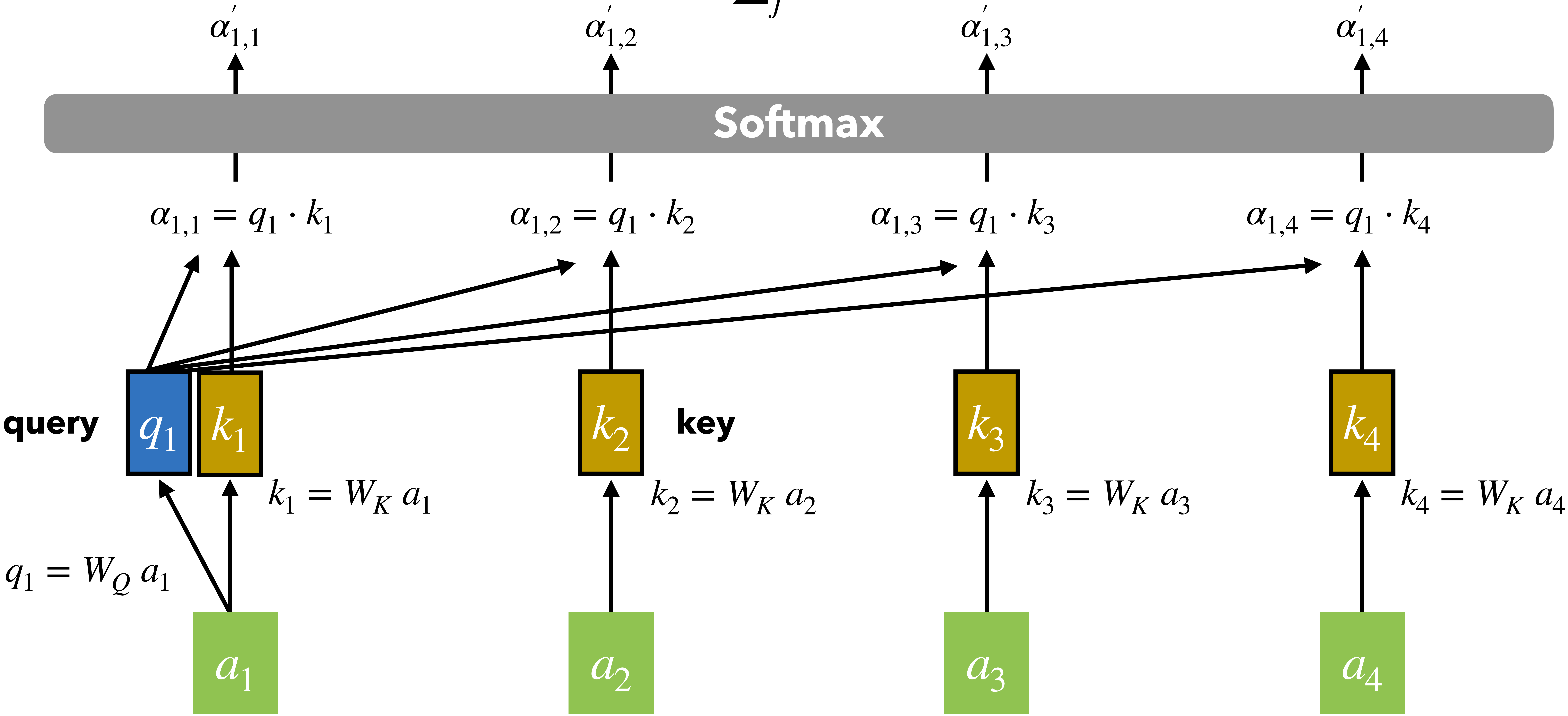
$k_2 = W_K\, a_2$

$k_3 = W_K\, a_3$

$k_4 = W_K\, a_4$

$q_1 = W_Q\, a_1$

$$\alpha'_{1,i} = \frac{e^{\alpha_{1,i}}}{\sum_j e^{\alpha_{1,j}}}$$



$\alpha'_{1,1}$ $\alpha'_{1,2}$ $\alpha'_{1,3}$ $\alpha'_{1,4}$

Softmax

$\alpha_{1,1} = q_1 \cdot k_1$ $\alpha_{1,2} = q_1 \cdot k_2$ $\alpha_{1,3} = q_1 \cdot k_3$ $\alpha_{1,4} = q_1 \cdot k_4$

query q_1 k_1 k_2 **key** k_3 k_4

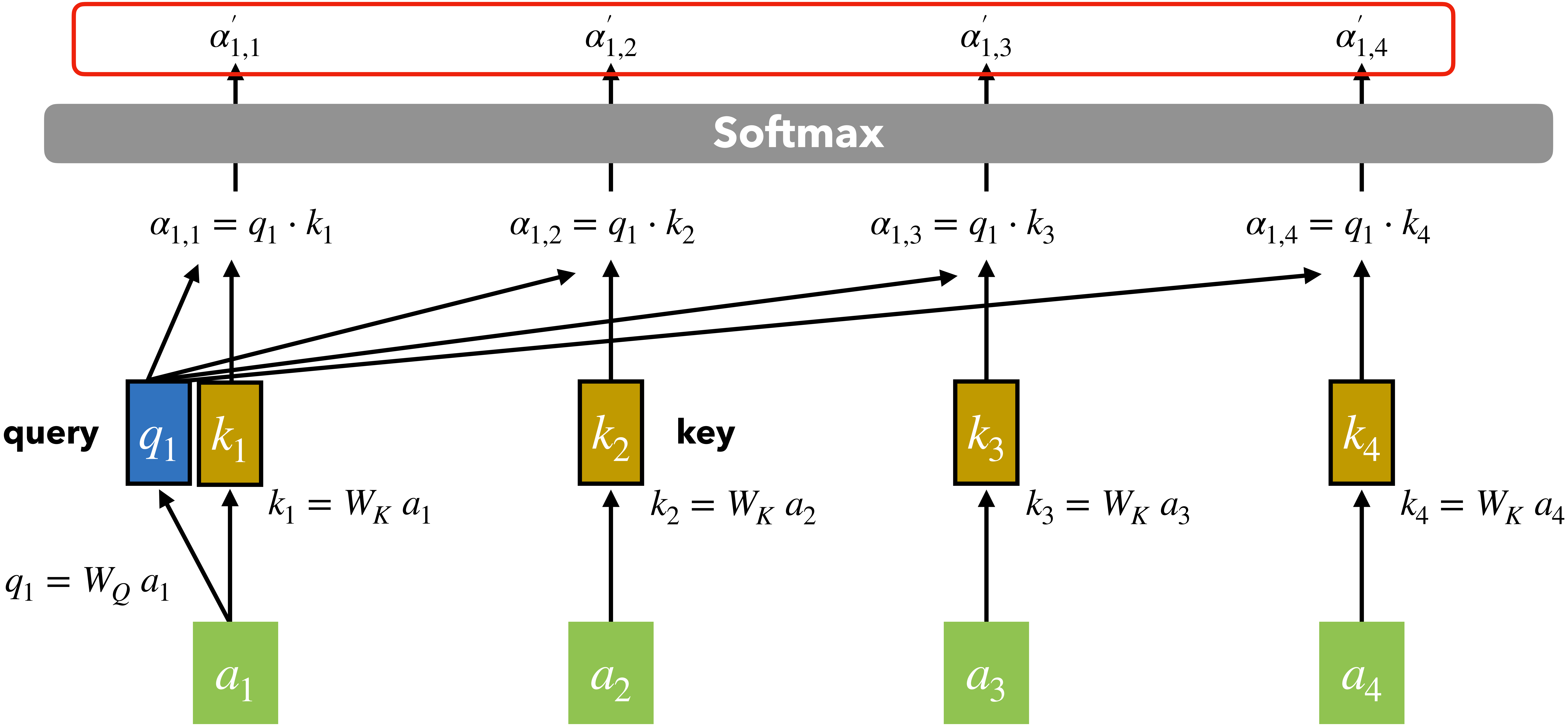
$k_1 = W_K\, a_1$ $k_2 = W_K\, a_2$ $k_3 = W_K\, a_3$ $k_4 = W_K\, a_4$

$q_1 = W_Q\, a_1$

a_1 a_2 a_3 a_4

Denote how relevant each token are to a_1!

Use attention scores to extract information



$\alpha'_{1,1}$ $\alpha'_{1,2}$ $\alpha'_{1,3}$ $\alpha'_{1,4}$

Softmax

$\alpha_{1,1} = q_1 \cdot k_1$ $\alpha_{1,2} = q_1 \cdot k_2$ $\alpha_{1,3} = q_1 \cdot k_3$ $\alpha_{1,4} = q_1 \cdot k_4$

query q_1 k_1 k_2 **key** k_3 k_4

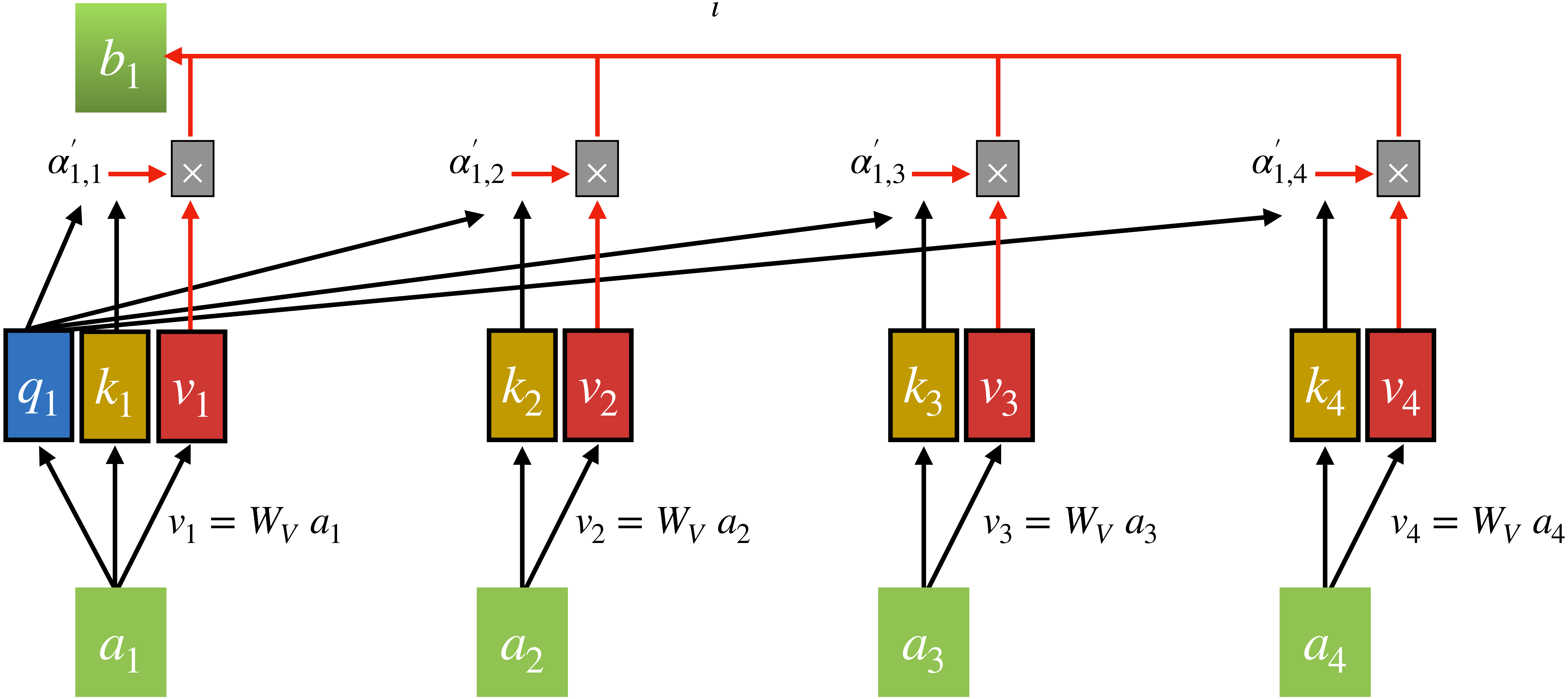
$k_1 = W_K\, a_1$ $k_2 = W_K\, a_2$ $k_3 = W_K\, a_3$ $k_4 = W_K\, a_4$

$q_1 = W_Q\, a_1$

a_1 a_2 a_3 a_4

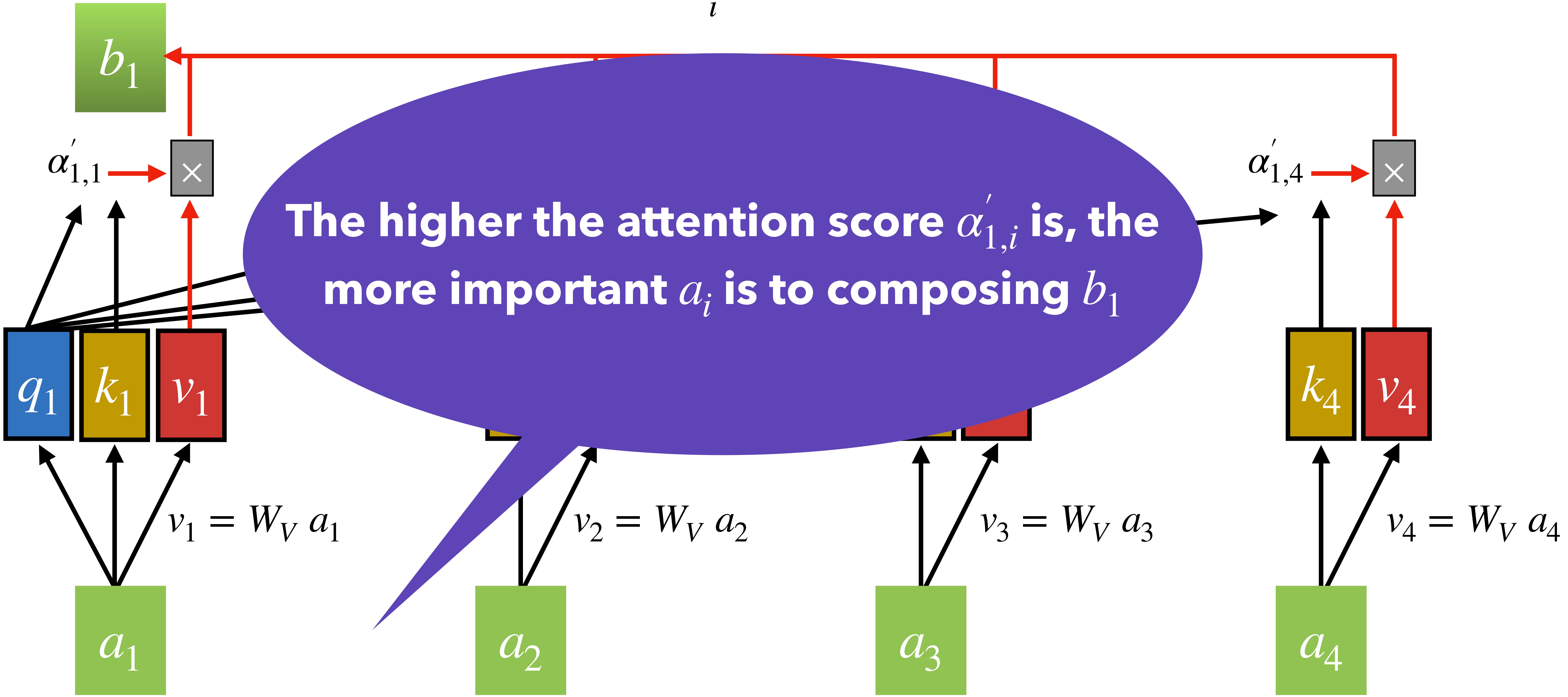
Use attention scores to extract information

$$b_1 = \sum_i \alpha'_{1,i} \, v_i$$



Use attention scores to extract information

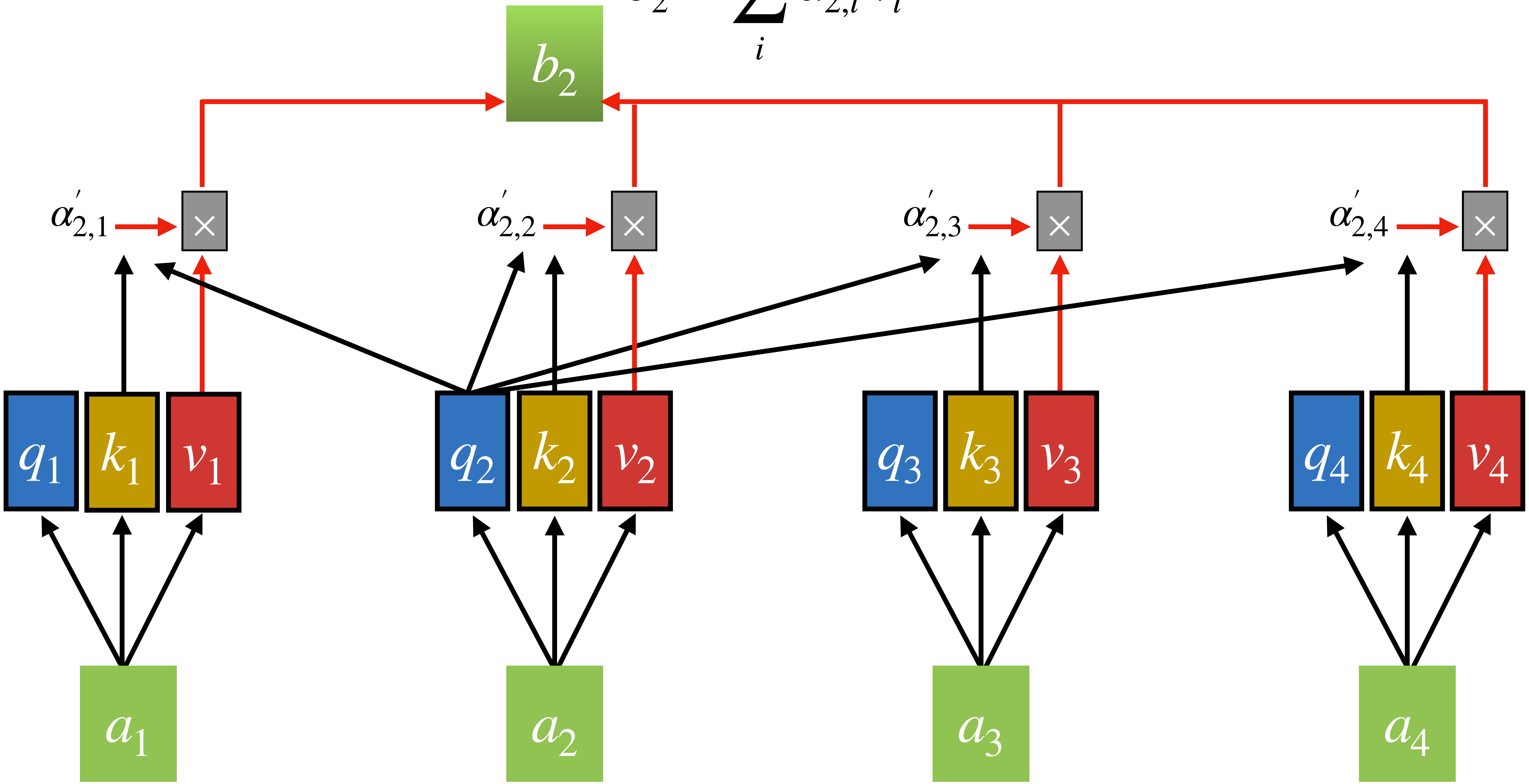
$$b_1 = \sum_i \alpha'_{1,i} v_i$$



The higher the attention score $\alpha'_{1,i}$ is, the more important a_i is to composing b_1

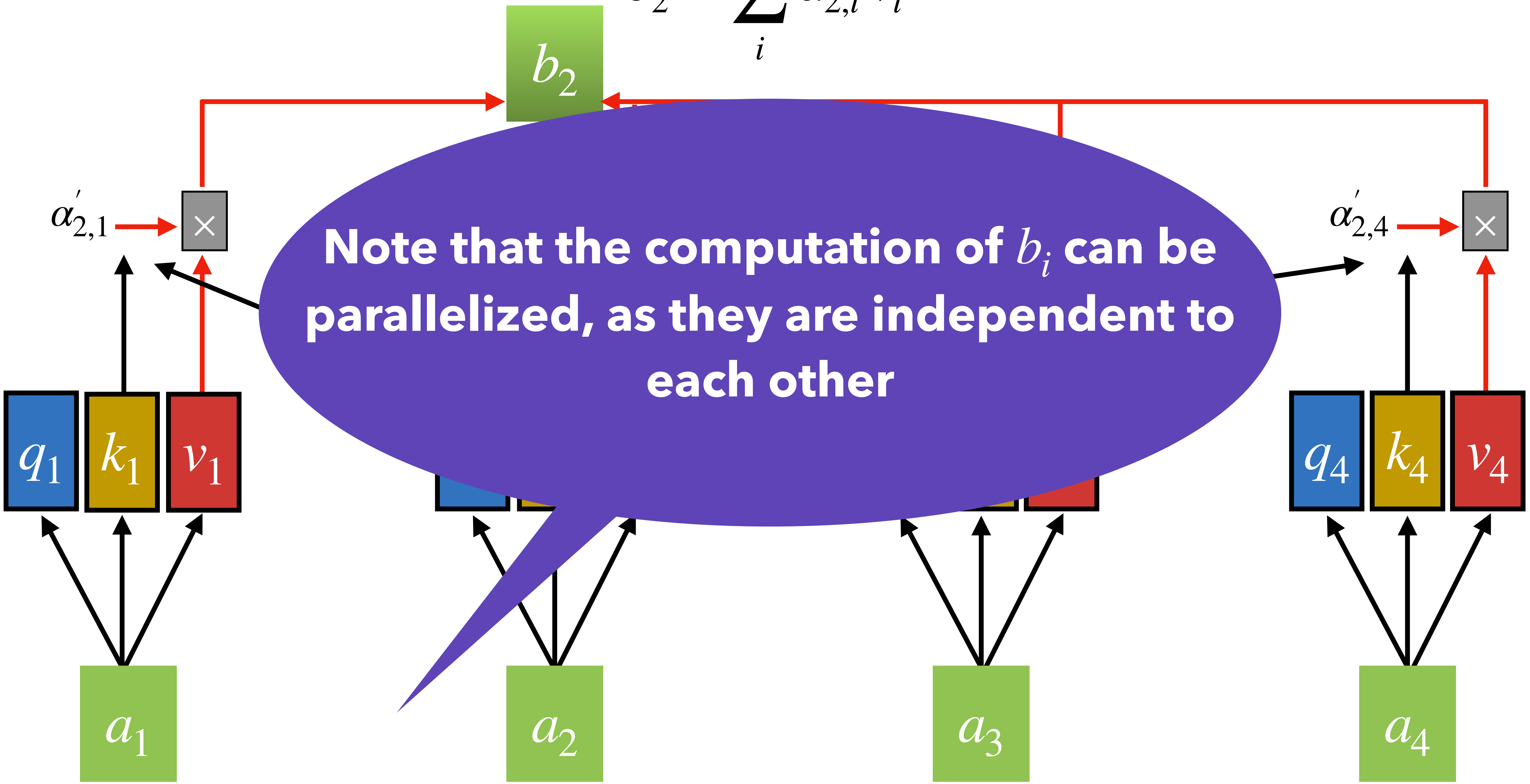
Repeat the same calculation for all a_i to obtain b_i

$$b_2 = \sum_i \alpha'_{2,i} \, v_i$$



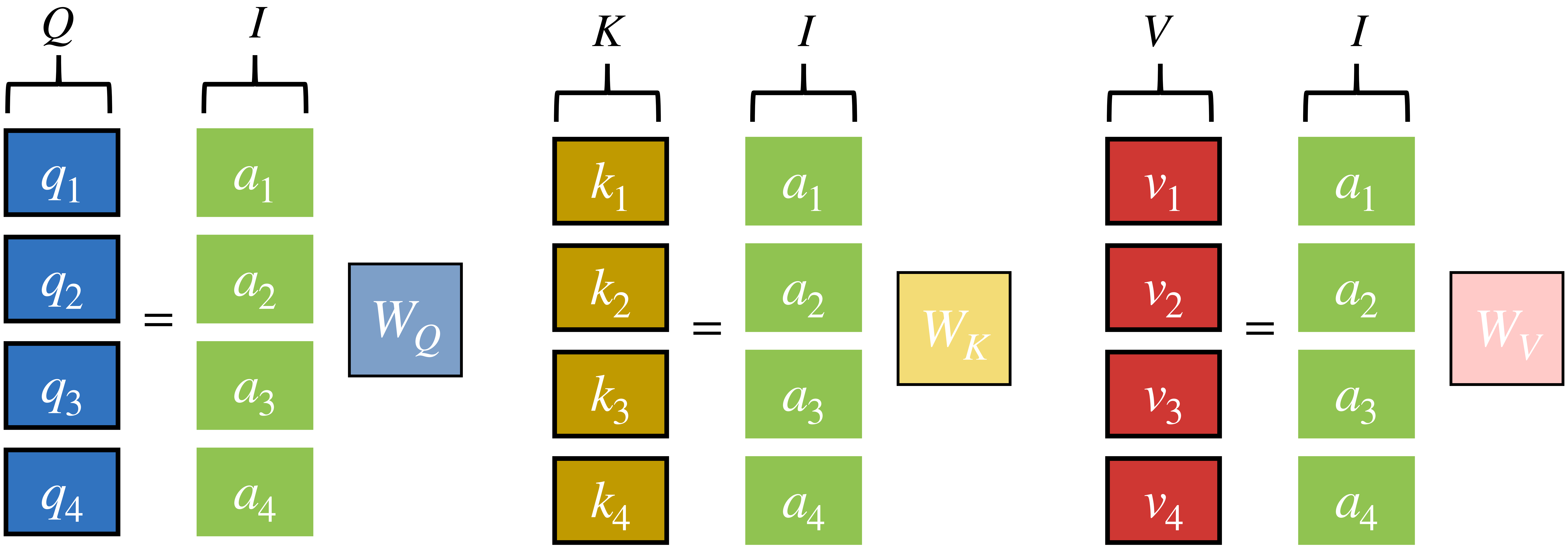
Repeat the same calculation for all a_i to obtain b_i

$$b_2 = \sum_i \alpha'_{2,i} v_i$$



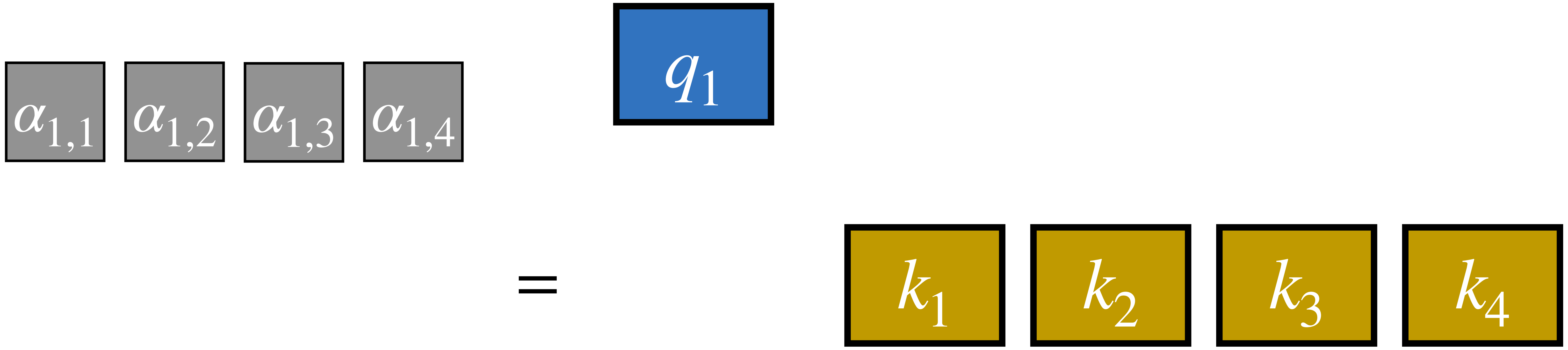
Note that the computation of b_i can be parallelized, as they are independent to each other

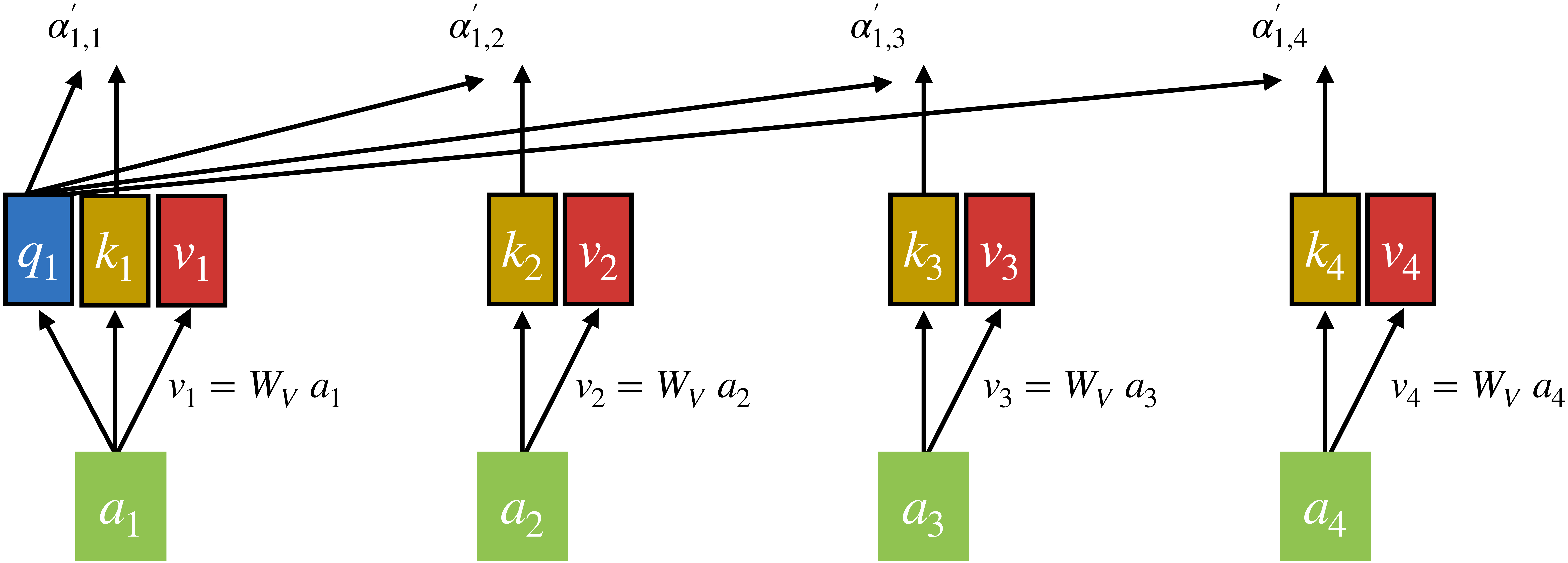
Parallelize the computation!
QKV



Parallelize the computation!

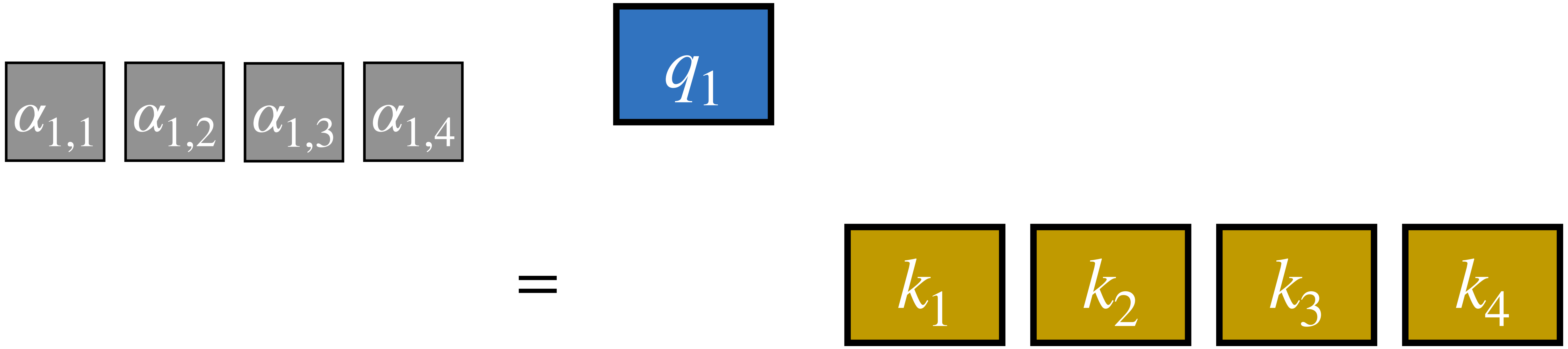
Attention Scores





Parallelize the computation!

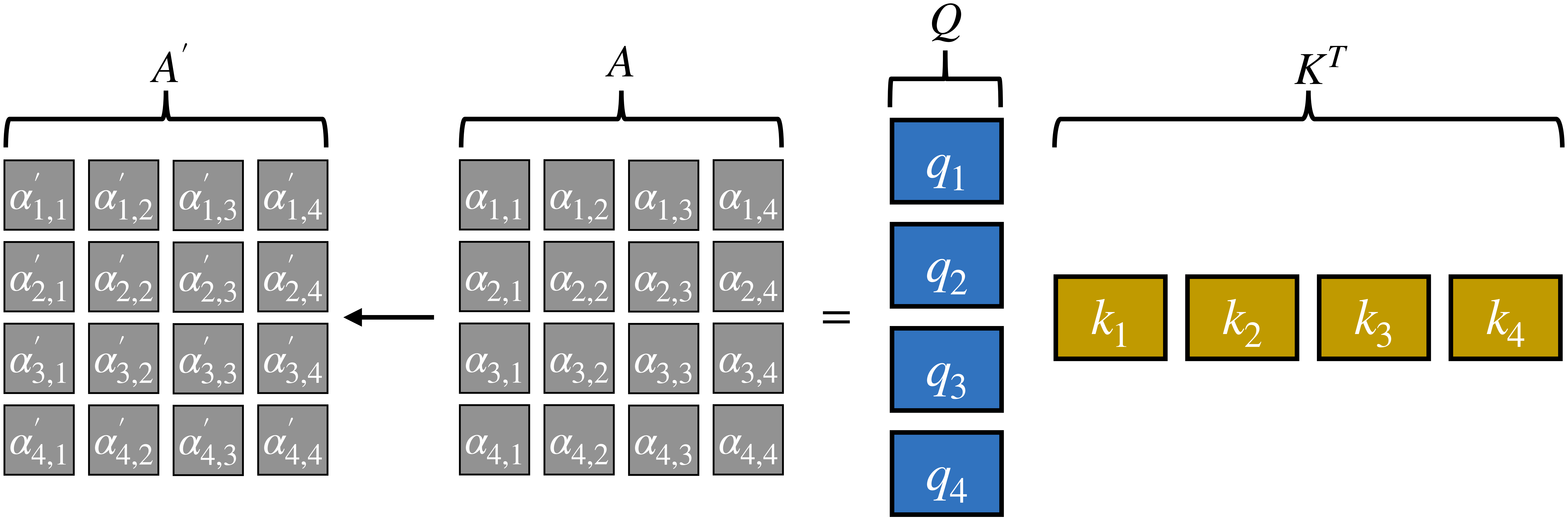
Attention Scores

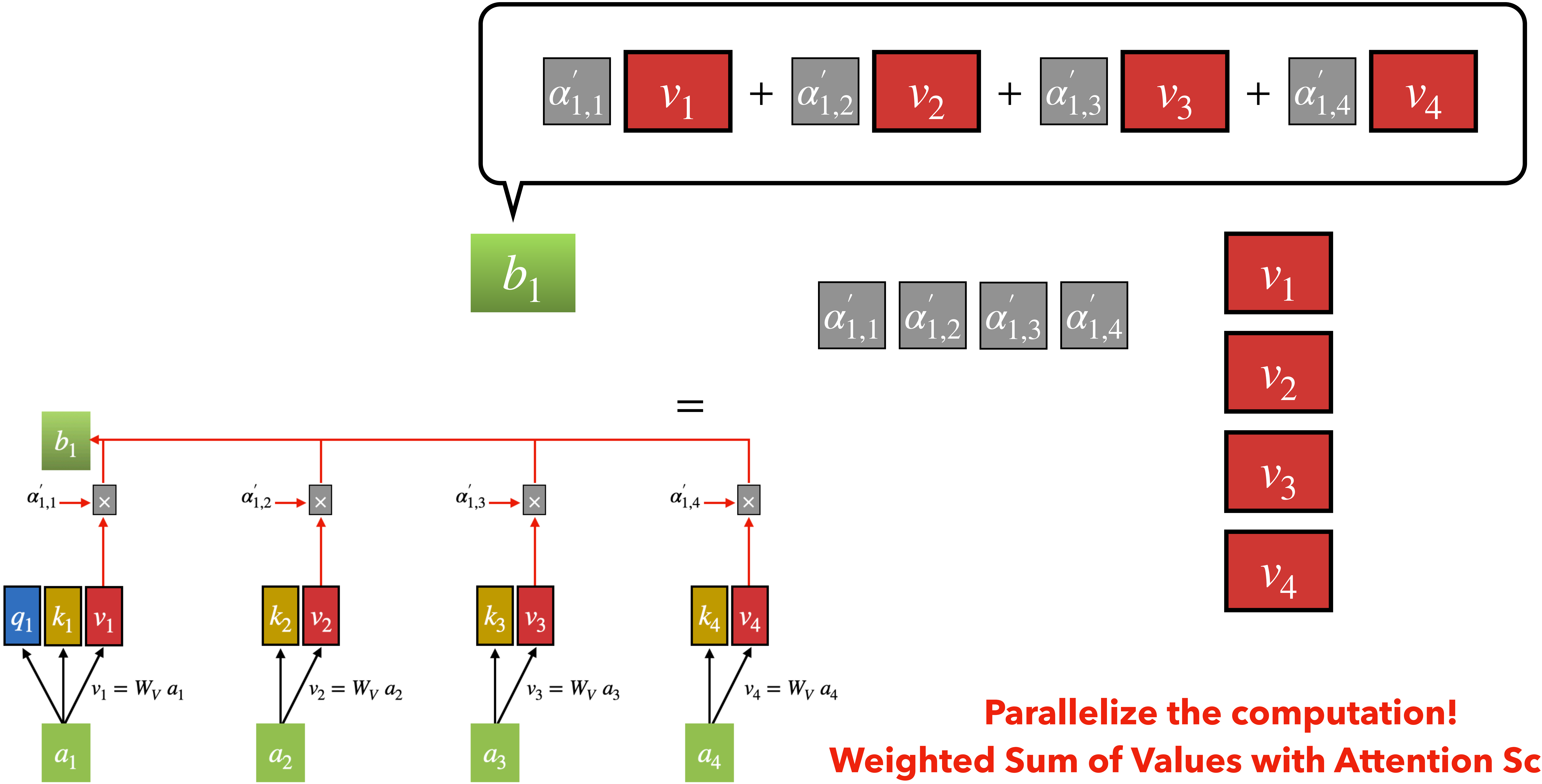


$$\alpha_{1,1} \quad \alpha_{1,2} \quad \alpha_{1,3} \quad \alpha_{1,4} = q_1 \quad k_1 \quad k_2 \quad k_3 \quad k_4$$

Parallelize the computation!

Attention Scores

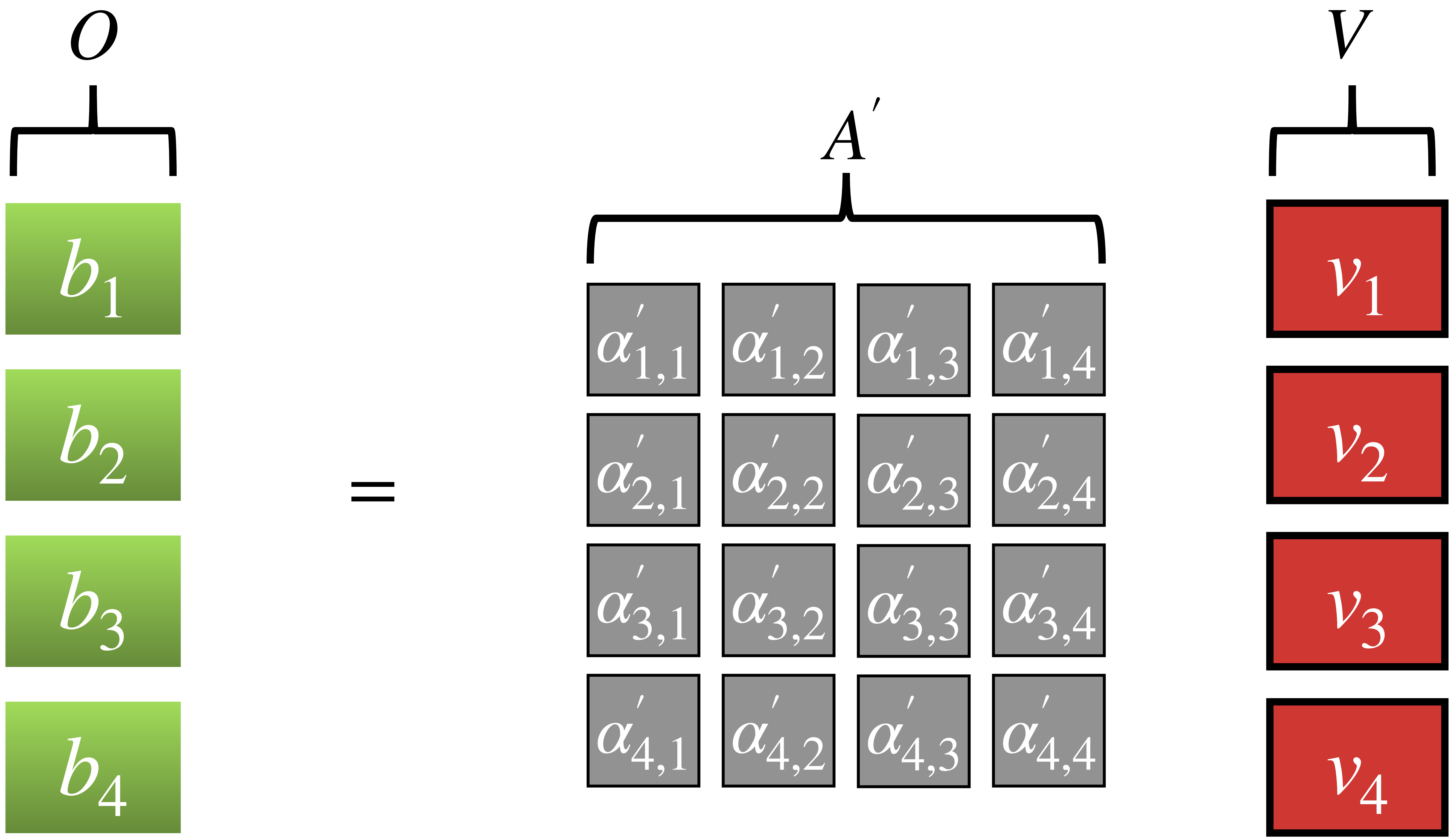




Parallelize the computation!

Weighted Sum of Values with Attention Scores

Parallelize the computation!



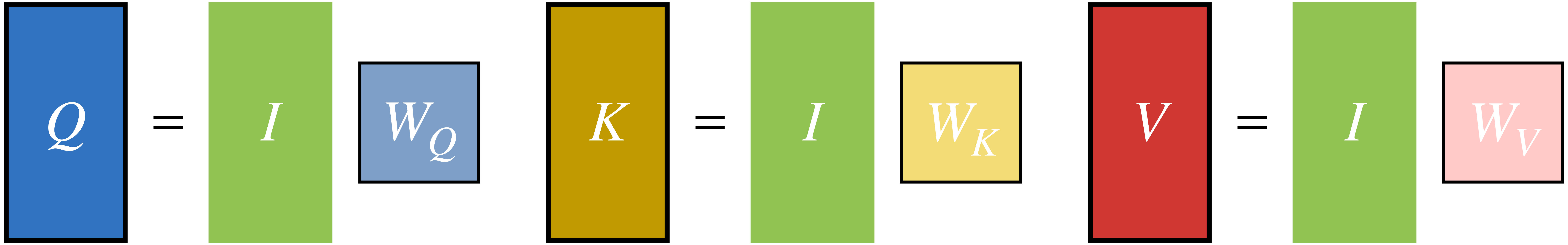
Parallelize the computation!

Weighted Sum of Values with Attention Scores

$$Q = I\, W_Q$$

$$K = I\, W_K$$

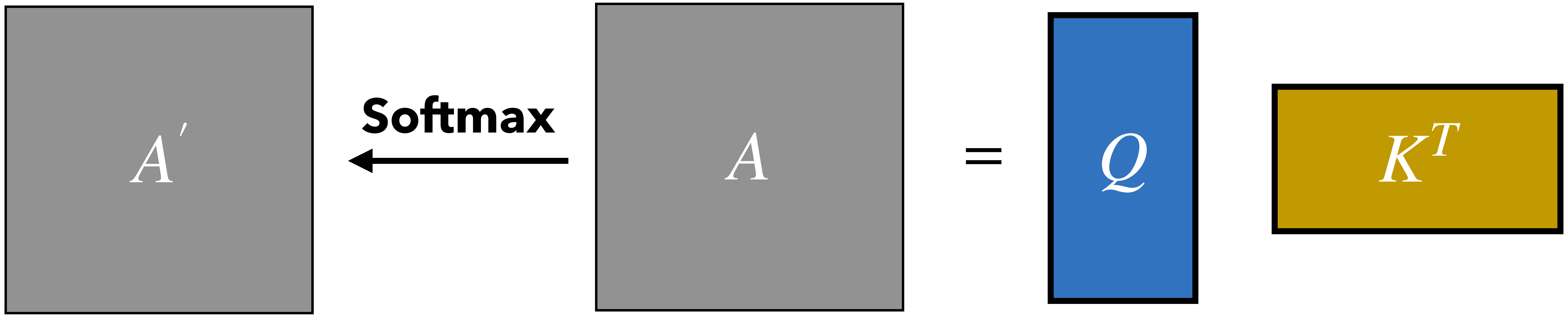
$$V = I\, W_V$$



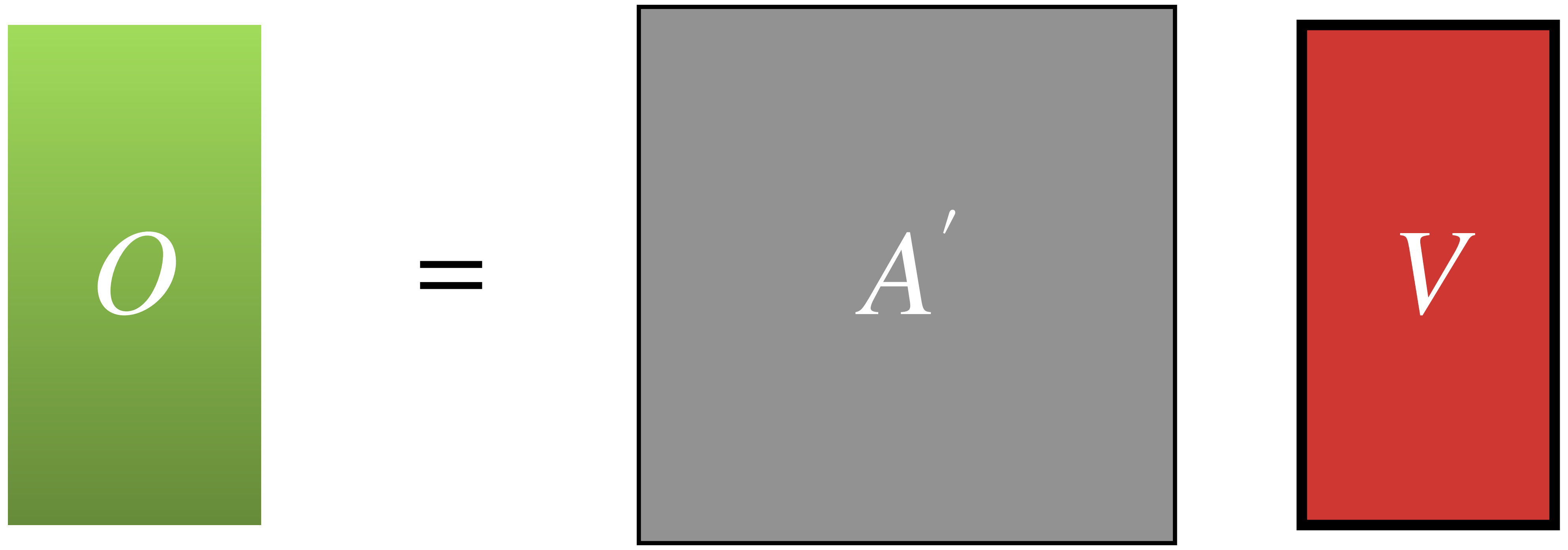
$$A = Q\, K^T$$

$$A = I\, W_Q\, (I\, W_K)^T = I\, W_Q\, W_K^T\, I^T$$

$$A^{'} = \mathrm{softmax}(A)$$



$$O = A^{'}\, V$$



The Matrices Form of Self-Attention

$$Q = I\,W_Q$$
$$K = I\,W_K$$
$$V = I\,W_V$$

$I = \{a_1, \ldots, a_n\} \in \mathbb{R}^{n \times d}$, where $a_i \in \mathbb{R}^d$

$W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$

$Q, K, V \in$ **?**

$$A = Q\,K^T$$
$$A = I\,W_Q\,(I\,W_K)^T = I\,W_Q\,W_K^T\,I^T$$
$$A' = \text{softmax}(A)$$

$A', A \in$ **?**

Dimensions?

$$O = A'\,V$$

$O \in$ **?**

The Matrices Form of Self-Attention

$$Q = I\,W_Q$$

$$K = I\,W_K$$

$$V = I\,W_V$$

$I = \{a_1, \ldots, a_n\} \in \mathbb{R}^{n \times d}$, where $a_i \in \mathbb{R}^d$

$W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$

$Q, K, V \in \mathbb{R}^{n \times d}$

$$A = Q\,K^T$$

$$A = I\,W_Q\,(I\,W_K)^T = I\,W_Q\,W_K^T\,I^T$$

$$A' = \text{softmax}(A)$$

$A', A \in \mathbb{R}^{n \times n}$

Dimensions?

$$O = A'\,V$$

$O \in \mathbb{R}^{n \times d}$

Self-Attention: Summary

Let $w_{1:n}$ be a sequence of words in vocabulary V, like *Steve Jobs founded Apple*.

For each w_i, let $a_i = Ew_i$, where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix.

1. Transform each word embedding with weight matrices W_Q, W_K, W_V, each in $\mathbb{R}^{d \times d}$

$q_i = W_Q\, a_i$ **(queries)** $k_i = W_K\, a_i$ **(keys)** $v_i = W_V\, a_i$ **(values)**

2. Compute pairwise similarities between keys and queries; normalize with softmax

$$\alpha_{i,j} = k_j\, q_i \qquad \alpha'_{i,j} = \frac{e^{\alpha_{i,j}}}{\sum_j e^{\alpha_{i,j}}}$$

3. Compute output for each word as weighted sum of values

$$b_i = \sum_j \alpha'_{i,j}\, v_j$$

Limitations and Solutions of Self-Attention

👎		👍
No Sequence Order	→	**Position Embedding**
No Nonlinearities	→	**Adding Feed-forward Networks**
Looking into the Future	→	**Masking**

Limitations and Solutions of Self-Attention

👎 👍

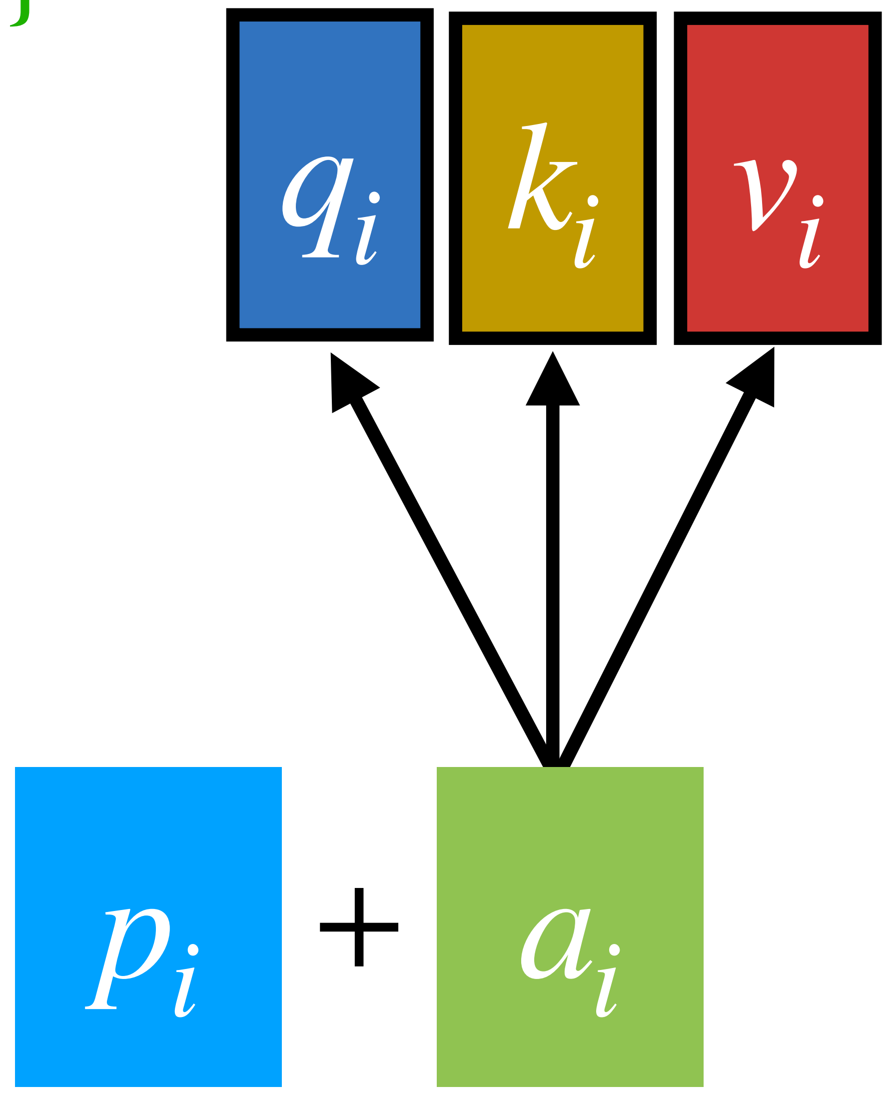
No Sequence Order → Position Embedding

No Nonlinearities → Adding Feed-forward Networks

Looking into the Future → Masking

No Sequence Order → Position Embedding

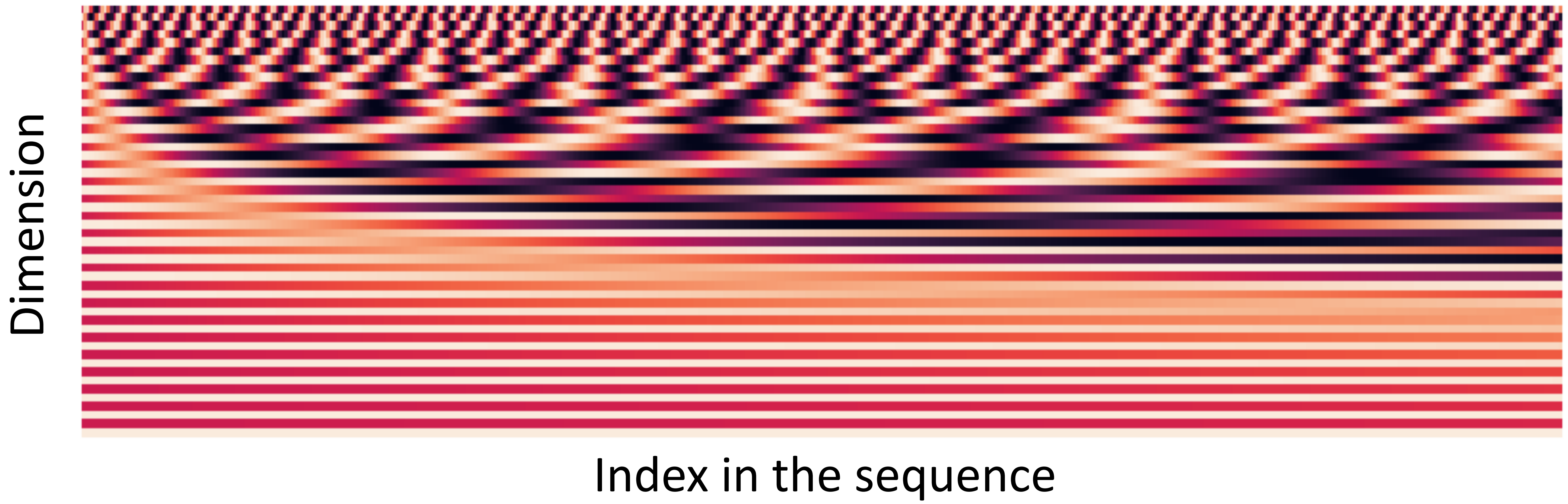
- All tokens in an input sequence are **simultaneously** fed into self-attention blocks. Thus, there's no difference between tokens at different positions.
  - **We lose the position info!**
- **How do we bring the position info back, just like in RNNs?**
  - **Representing each sequence index as a vector:** $p_i \in \mathbb{R}^d$, for $i \in \{1,\ldots,n\}$
- **How to incorporate the position info into the self-attention blocks?**
  - **Just add the p_i to the input:** $\hat{a}_i = a_i + p_i$
  - where a_i is the embedding of the word at index i.
  - In deep self-attention networks, we do this at the **first layer**.
  - We can also concatenate a_i and p_i, but more commonly we add them.



Position Representation Vectors via Sinusoids

Sinusoidal Position Representations (from the original Transformer paper): concatenate sinusoidal functions of varying periods.

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



https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

👍
- Periodicity indicates that maybe "absolute position" isn't as important
- Maybe can extrapolate to longer sequences as periods restart!

👎
- Not learnable; also the extrapolation doesn't really work!

Learnable Position Representation Vectors

Learned absolute position representations: p_i contains learnable parameters.

- Learn a matrix $p \in \mathbb{R}^{d \times n}$, and let each p_i be a column of that matrix
- Most systems use this method.

👍
- **Flexibility:** each position gets to be learned to fit the data

👎
- Cannot extrapolate to indices outside $1, \ldots, n$.

Sometimes people try more flexible representations of position:

- Relative linear position attention [Shaw et al., 2018]
- Dependency syntax-based position [Wang et al., 2019]

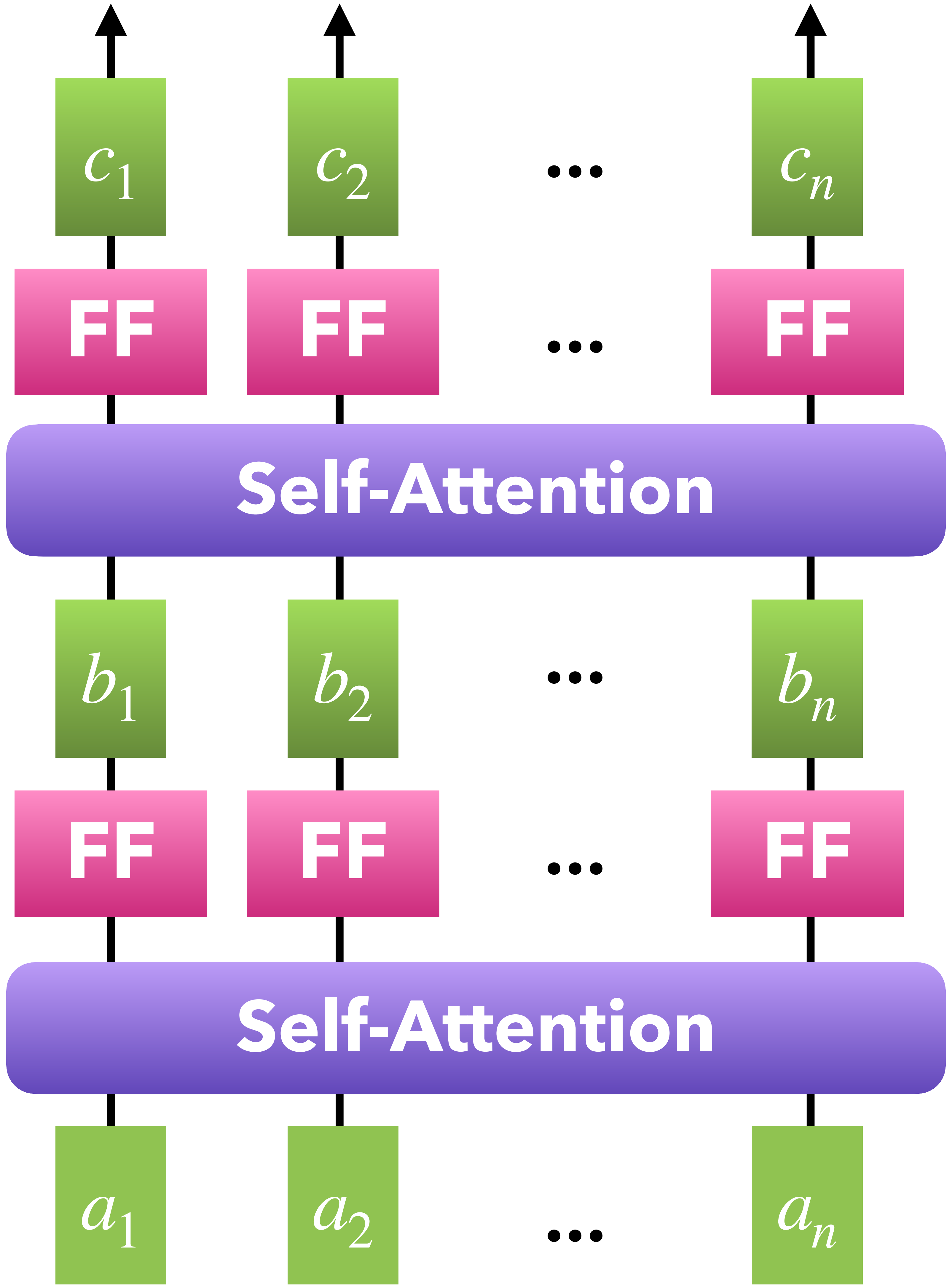
Limitations and Solutions of Self-Attention

👎		👍
No Sequence Order	→	**Position Embedding**
No Nonlinearities	→	**Adding Feed-forward Networks**
Looking into the Future	→	**Masking**

No Nonlinearities → Add Feed-forward Networks

There are **no element-wise nonlinearities** in self-attention; stacking more self-attention layers just re-averages value vectors.

Easy Fix: add a feed-forward network to post-process each output vector.

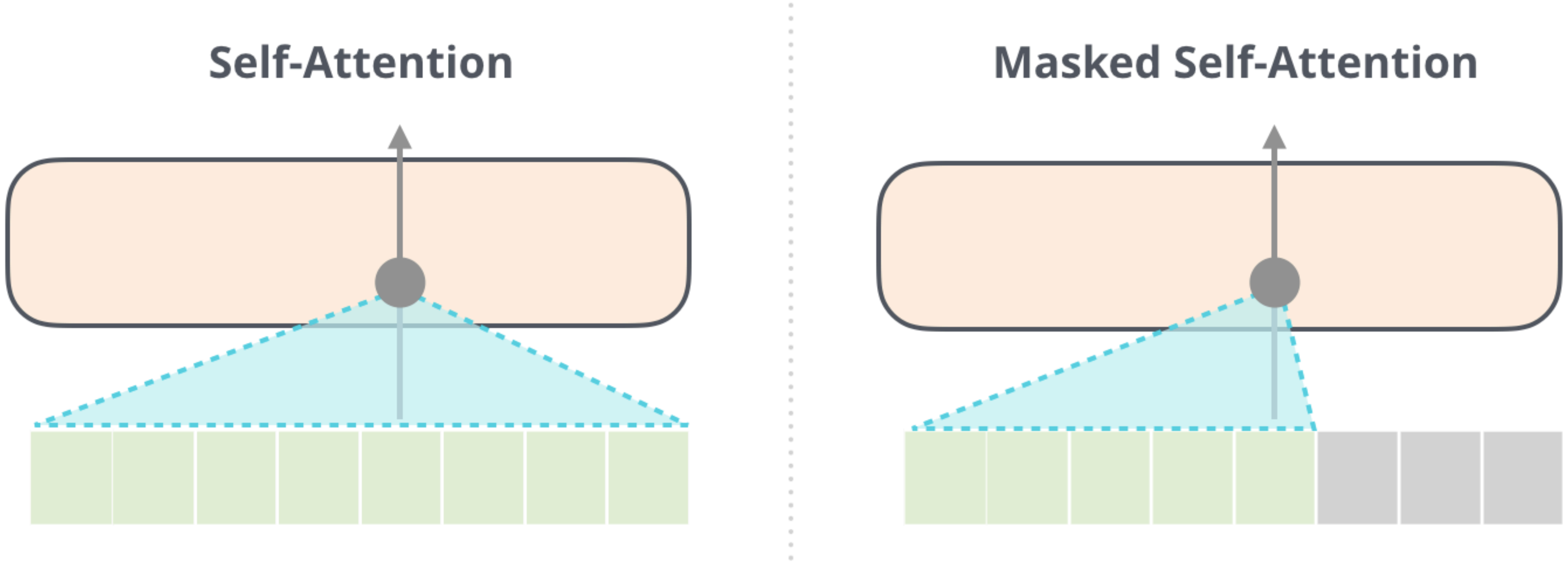


Limitations and Solutions of Self-Attention

👎		👍
No Sequence Order	→	**Position Embedding**
No Nonlinearities	→	**Adding Feed-forward Networks**
Looking into the Future	→	**Masking**

Looking into the Future → Masking

- In **decoders** (language modeling, producing the next word given previous context), we need to ensure we **don't** peek at the future.

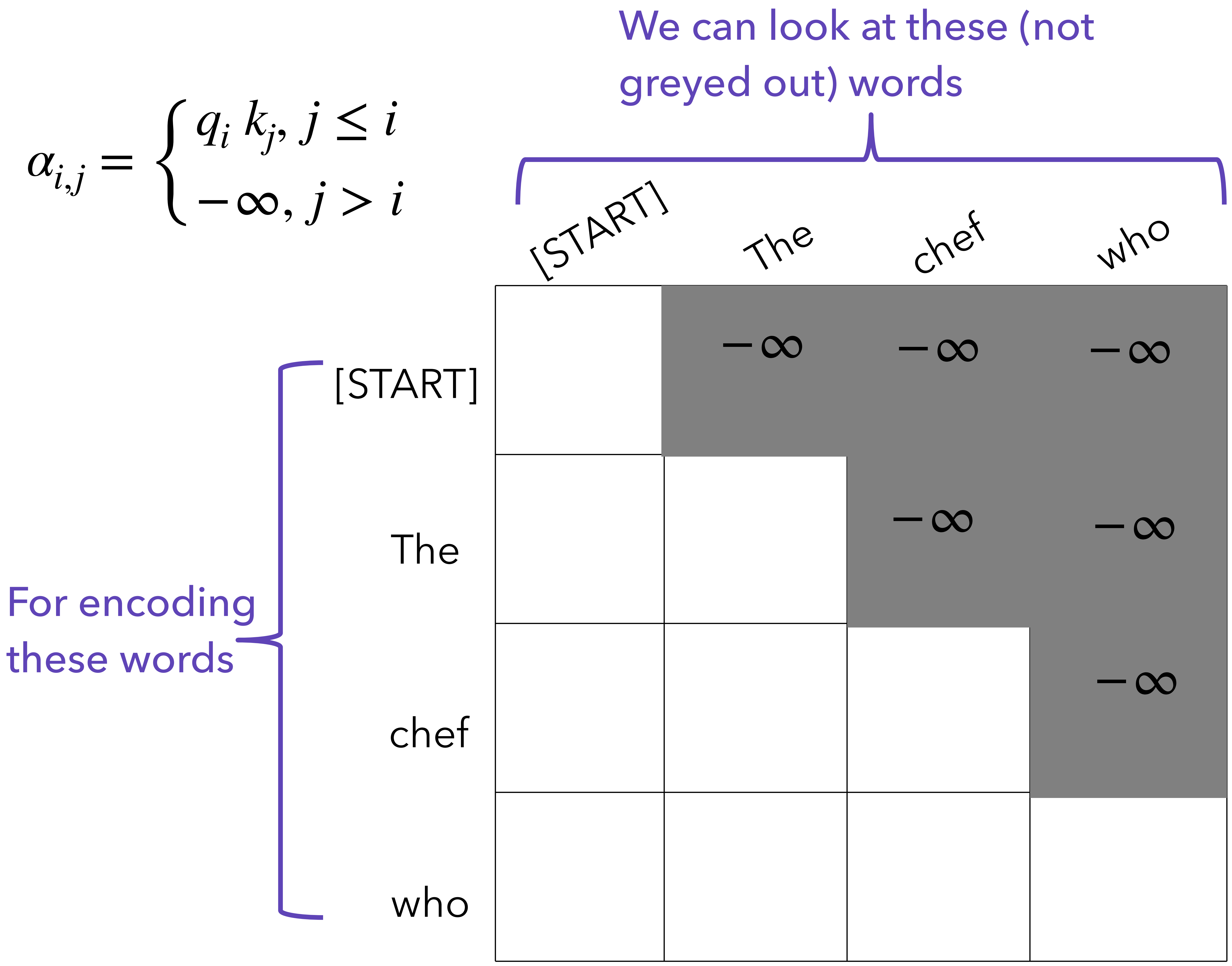


https://jalammar.github.io/illustrated-gpt2/

Looking into the Future → Masking

- In **decoders** (language modeling, producing the next word given previous context), we need to ensure we **don't** peek at the future.
- To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$.

$$\alpha_{i,j} = \begin{cases} q_i k_j, j \leq i \\ -\infty, j > i \end{cases}$$



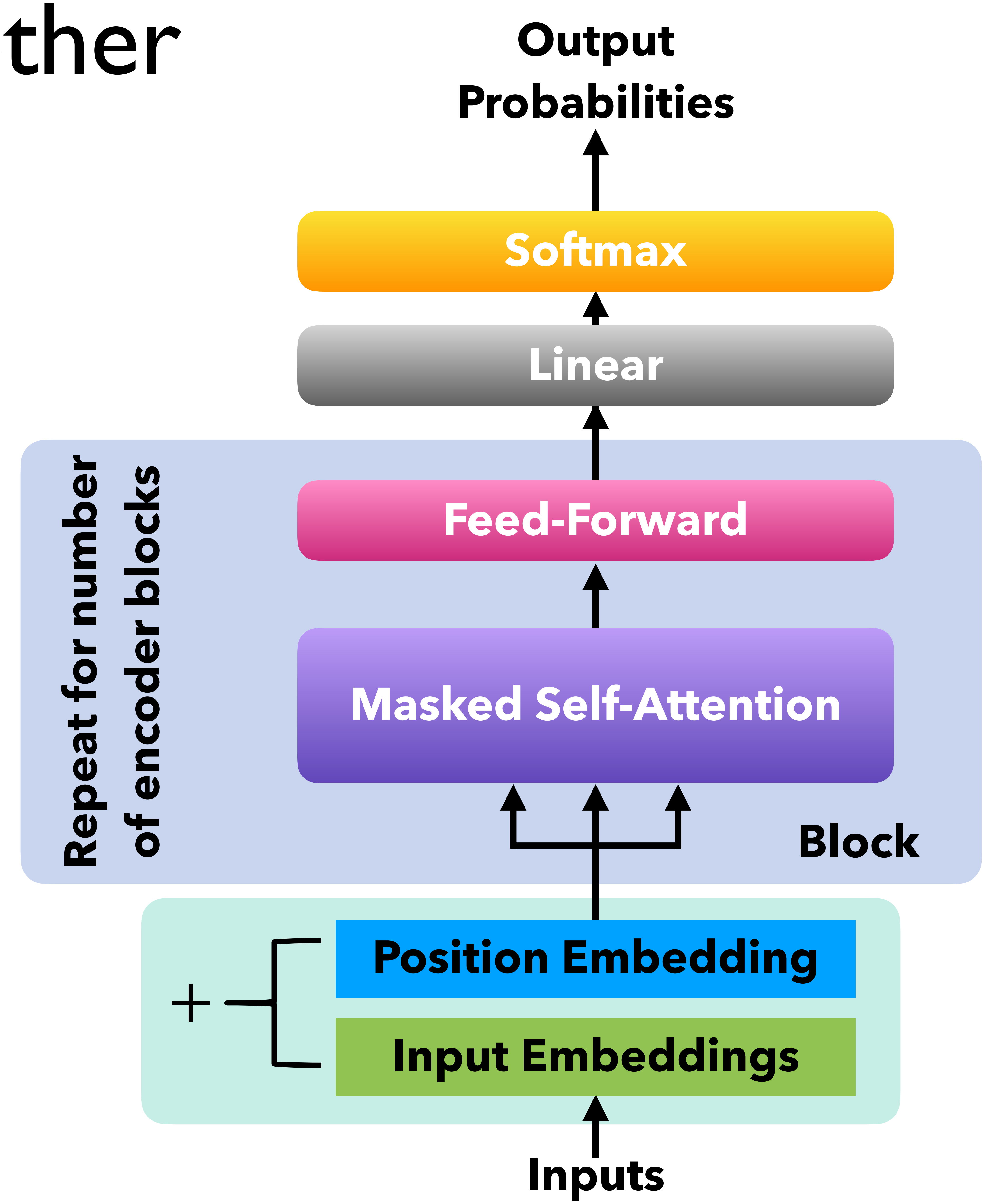
We can look at these (not greyed out) words

For encoding these words

	[START]	The	chef	who
[START]		$-\infty$	$-\infty$	$-\infty$
The			$-\infty$	$-\infty$
chef				$-\infty$
who				

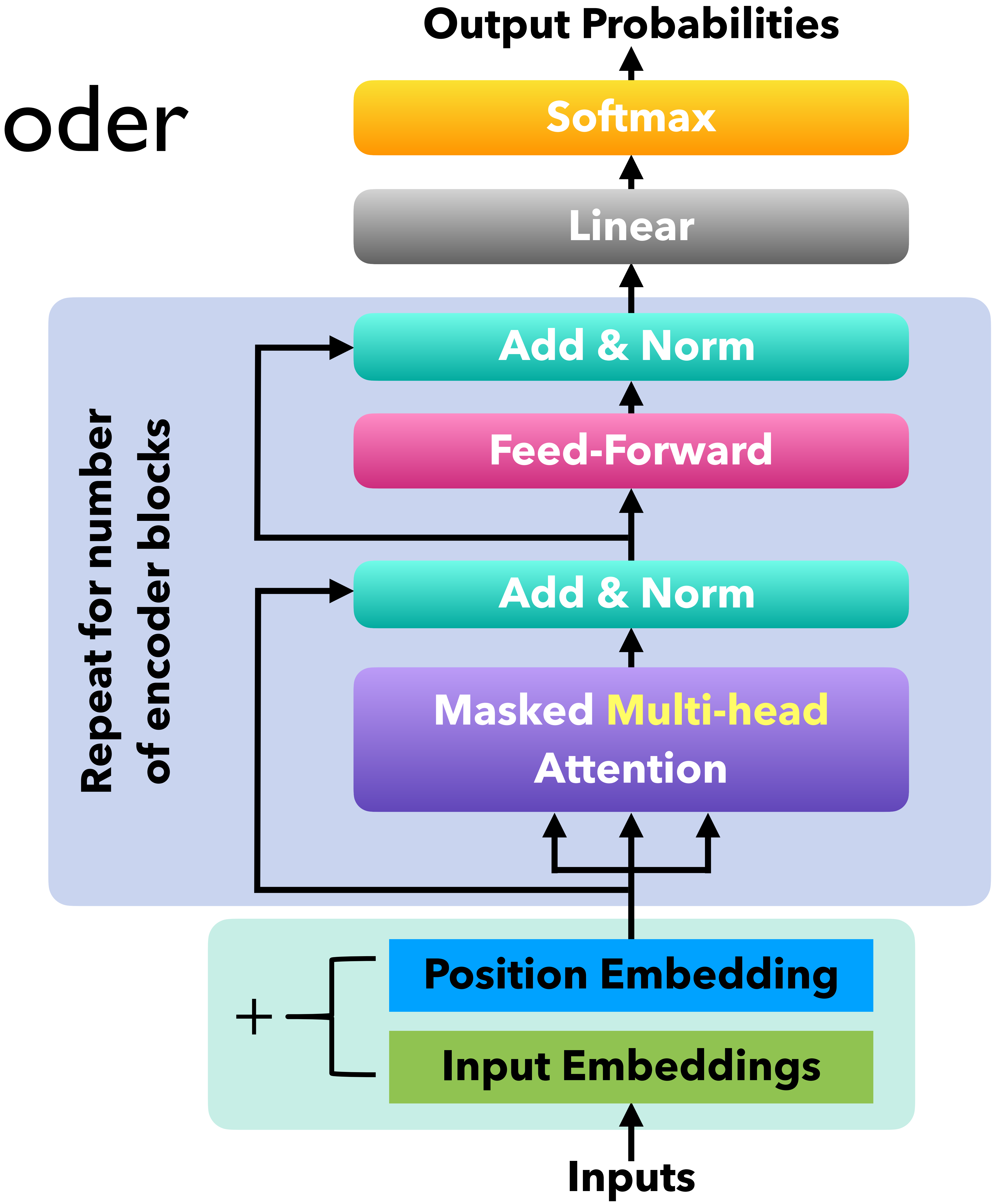
Now We Put Things Together

- **Self-attention**
  - The basic computation
- **Positional Encoding**
  - Specify the sequence order
- **Nonlinearities**
  - Adding a feed-forward network at the output of the self-attention block
- **Masking**
  - Parallelize operations (looking at all tokens) while not leaking info from the future



The Transformer Decoder

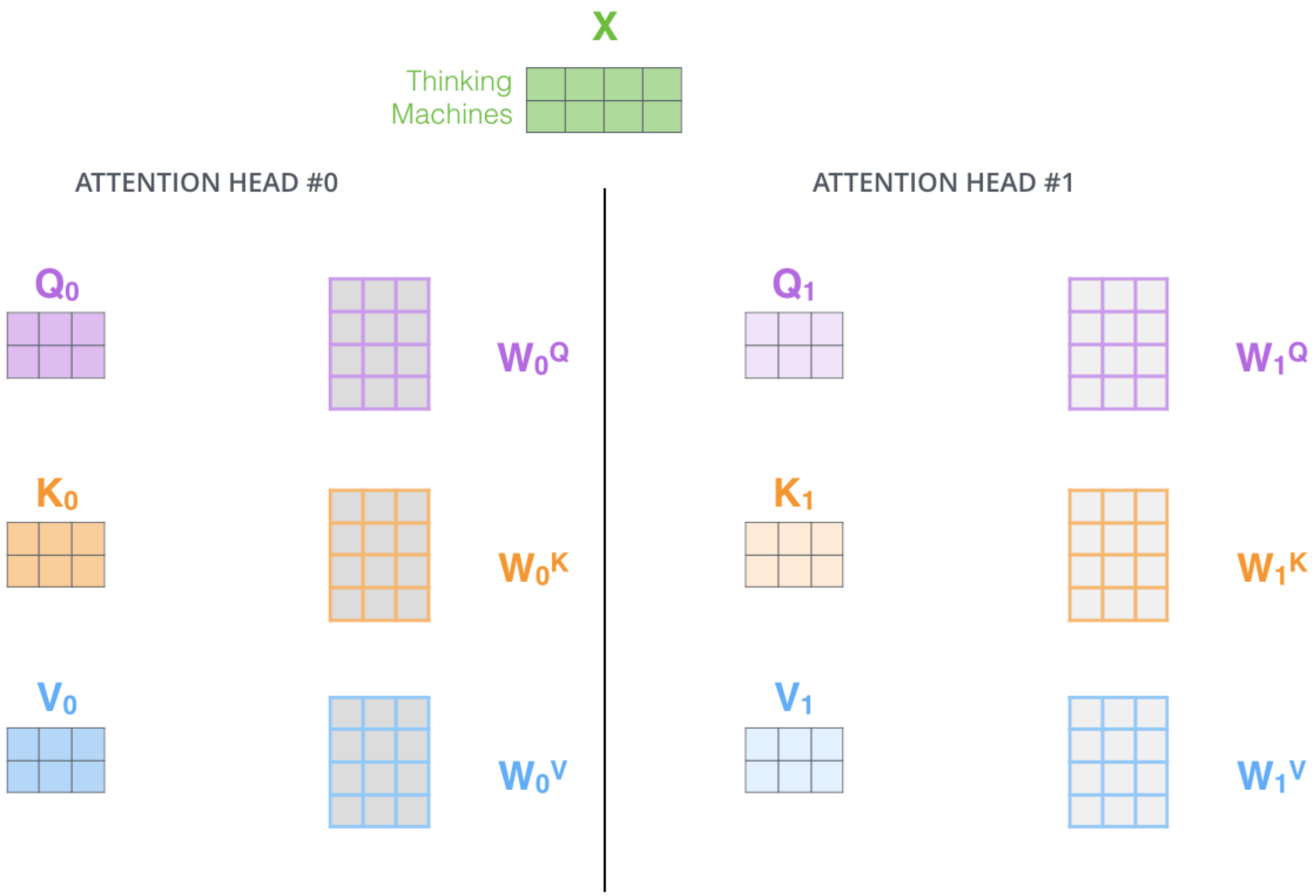
- A **Transformer decoder** is what we use to build systems like language models.
- It's a lot like our minimal self-attention architecture, but with a few more components.
  - ***Residual connection ("Add")***
  - ***Layer normalization ("Norm")***
- Replace self-attention with **multi-head self-attention**.

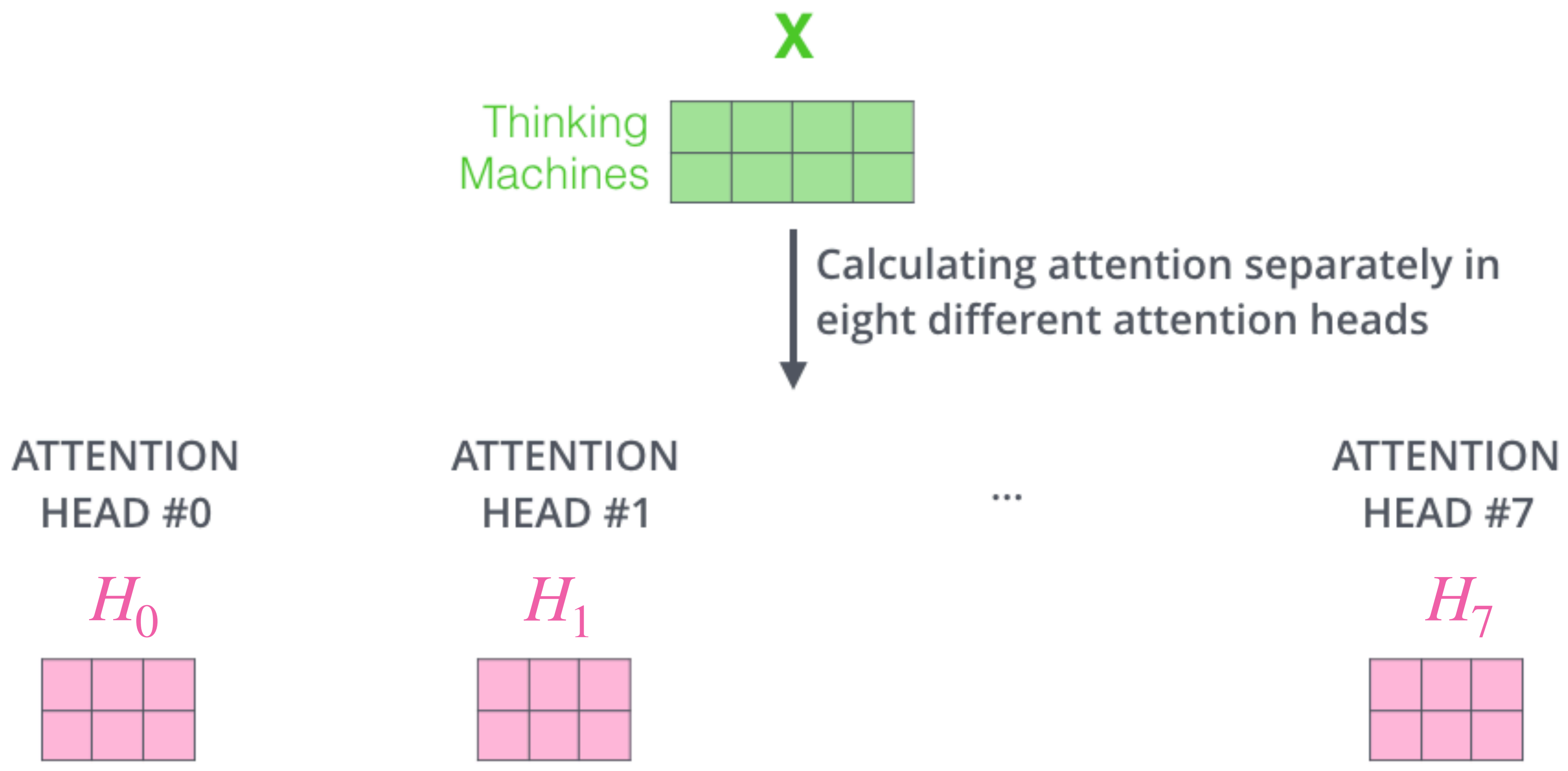


Multi-head Attention

"The Beast with Many Heads"

- It is better to use multiple attention functions instead of one!
  - Each attention function ("head") can focus on different positions.

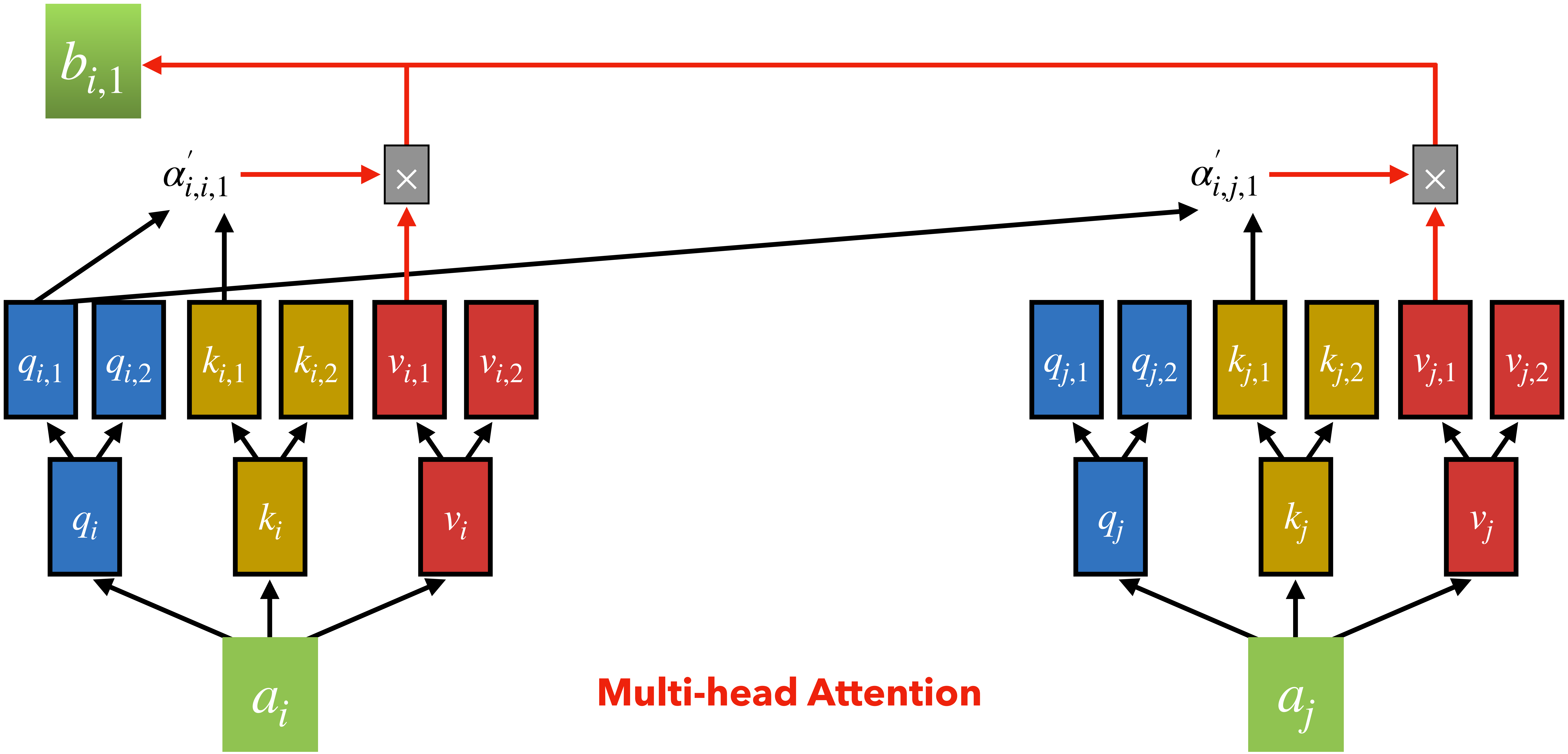




Calculating attention separately in eight different attention heads

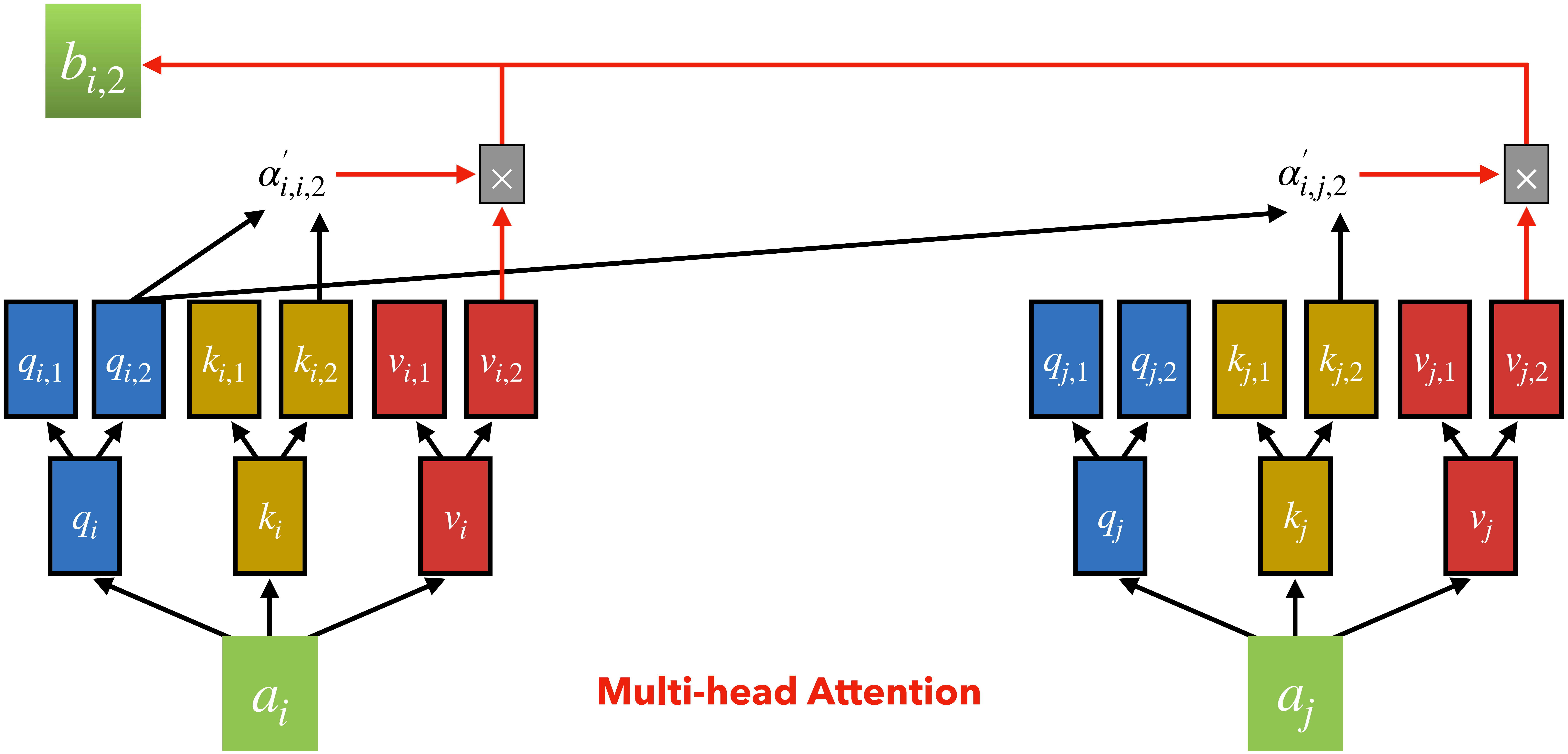
https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention: Walk-through



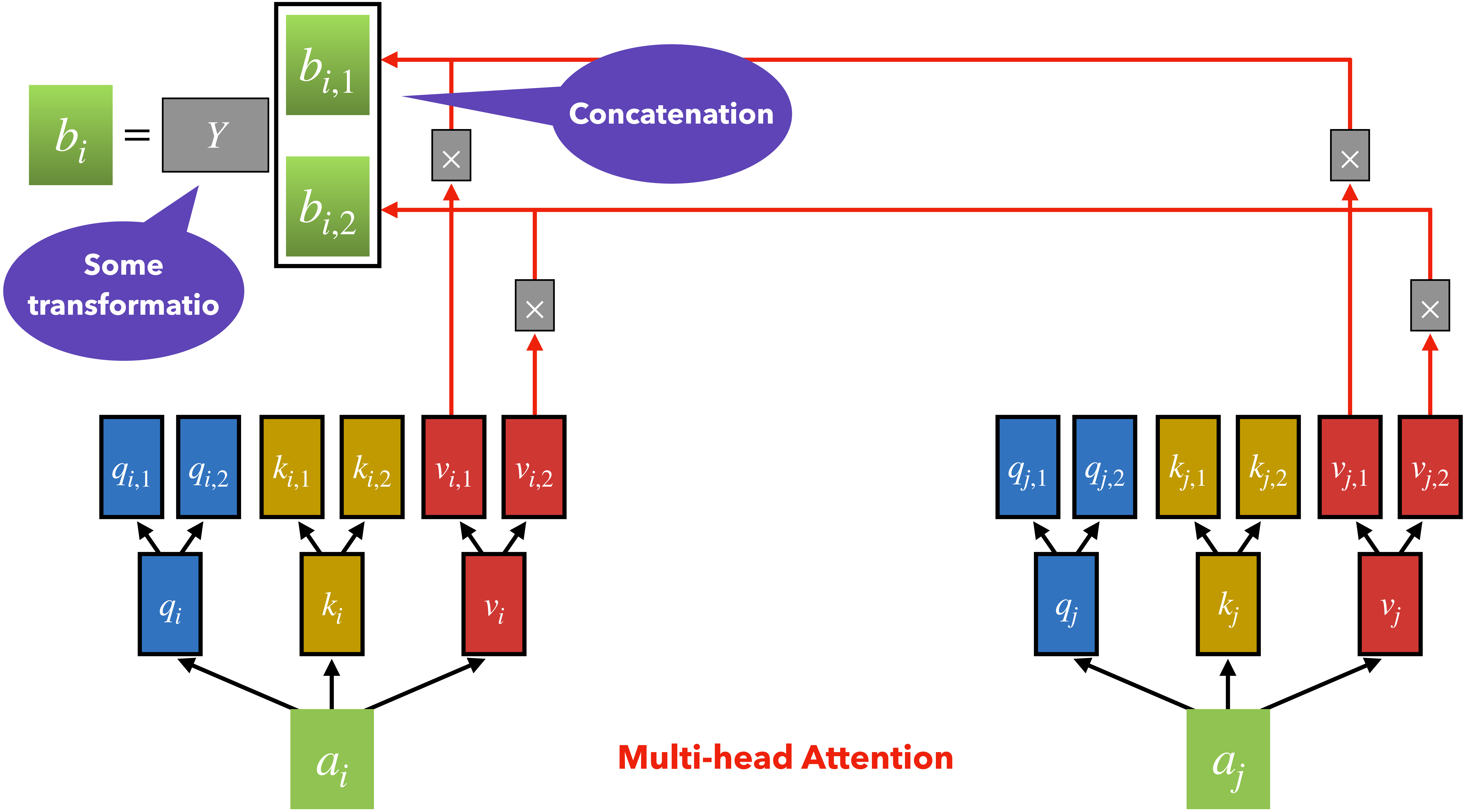
Multi-head Attention

Multi-Head Attention: Walk-through



Multi-head Attention

Multi-head Attention



Concatenation

Some transformatio

Recall the Matrices Form of Self-Attention

$$Q = I\ W_Q$$

$$K = I\ W_K$$

$$V = I\ W_V$$

$I = \{a_1, \ldots, a_n\} \in \mathbb{R}^{n \times d}$, where $a_i \in \mathbb{R}^d$

$$W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$$

$$Q, K, V \in \mathbb{R}^{n \times d}$$

$$A = Q\ K^T$$

$$A = I\ W_Q\ (I\ W_K)^T = I\ W_Q\ W_K^T\ I^T$$

$$A^{'} = \text{softmax}(A)$$

$$A^{'}, A \in \mathbb{R}^{n \times n}$$

$$O = A^{'}\ V$$

$$O \in \mathbb{R}^{n \times d}$$

Multi-head Attention in Matrices

- Multiple attention “heads” can be defined via multiple W_Q, W_K, W_V matrices
- Let $W_Q^l, W_K^l, W_V^l \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and l ranges from 1 to h.
- Each attention head performs attention independently:
  - $O^l = \text{softmax}(I\ W_Q^l\ {W_K^l}^T\ I^T)\ I\ W_V^l$
- Concatenating different O^l from different attention heads.
  - $O = [O^1; \ldots; O^n]\ Y$, where $Y \in \mathbb{R}^{d \times d}$

The Matrices Form of Multi-head Attention

$$Q^l = I\,W_Q^l$$

$$K^l = I\,W_K^l$$

$$V^l = I\,W_V^l$$

$I = \{a_1, \ldots, a_n\} \in \mathbb{R}^{n\times d}$, where $a_i \in \mathbb{R}^d$

$W_Q^l, W_K^l, W_V^l \in \mathbb{R}^{d\times\frac{d}{h}}$

$Q^l, K^l, V^l \in$ **?**

$$A^l = Q^l\,{K^l}^T$$

$$A^{l'} = \text{softmax}(A^l)$$

$A^{l'}, A^l \in \mathbb{R}$ **?**

$$O^l = A^{l'}\,V^l$$

$O^l \in \mathbb{R}$ **?**

$$O = [O^1; \ldots; O^h]\,Y$$

$Y \in \mathbb{R}^{d\times d}$

$[O^1; \ldots; O^h] \in$ **?**

$O \in \mathbb{R}$ **?**

Dimensions?

The Matrices Form of Multi-head Attention

$$Q^l = I\, W_Q^l$$
$$K^l = I\, W_K^l$$
$$V^l = I\, W_V^l$$

$I = \{a_1, \ldots, a_n\} \in \mathbb{R}^{n \times d}$, where $a_i \in \mathbb{R}^d$

$$W_Q^l, W_K^l, W_V^l \in \mathbb{R}^{d \times \frac{d}{h}}$$
$$Q^l, K^l, V^l \in \mathbb{R}^{n \times \frac{d}{h}}$$

$$A^l = Q^l\, K^{l^T}$$
$$A^{l'} = \text{softmax}(A^l)$$

$$A^{l'}, A^l \in \mathbb{R}^{n \times n}$$

$$O^l = A^{l'}\, V^l$$

$$O^l \in \mathbb{R}^{n \times \frac{d}{h}}$$

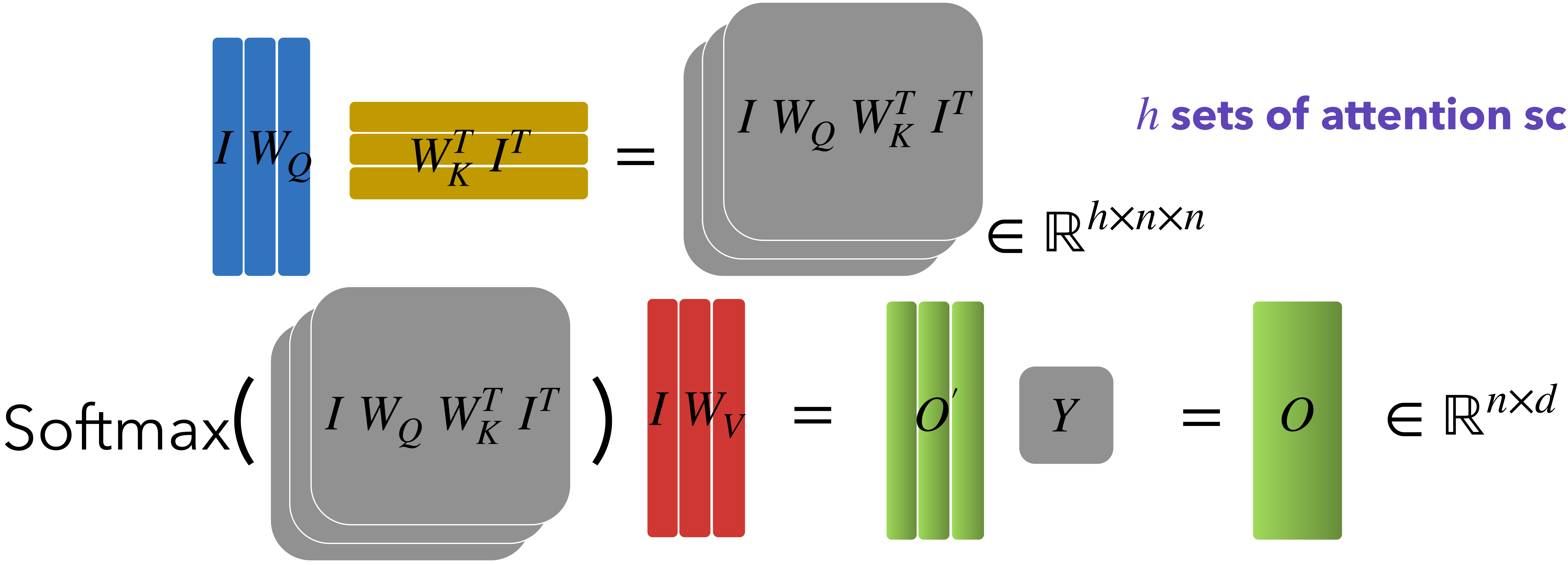
$$O = [O^1; \ldots; O^h]\, Y$$

$$Y \in \mathbb{R}^{d \times d}$$
$$[O^1; \ldots; O^h] \in \mathbb{R}^{n \times d}$$
$$O \in \mathbb{R}^{n \times d}$$

Dimensions?

Multi-head Attention is Computationally Efficient

- Even though we compute h many attention heads, it's not more costly.
  - We compute $I\,W_Q \in \mathbb{R}^{n \times d}$, and then reshape to $\mathbb{R}^{n \times h \times \frac{d}{h}}$.
  - Likewise for $I\,W_K$ and $I\,W_V$.
  - Then we transpose to $\mathbb{R}^{h \times n \times \frac{d}{h}}$; **now the head axis is like a batch axis.**
  - Almost everything else is identical. All we need to do is to reshape the tensors!



h sets of attention scores!

Scaled Dot Product

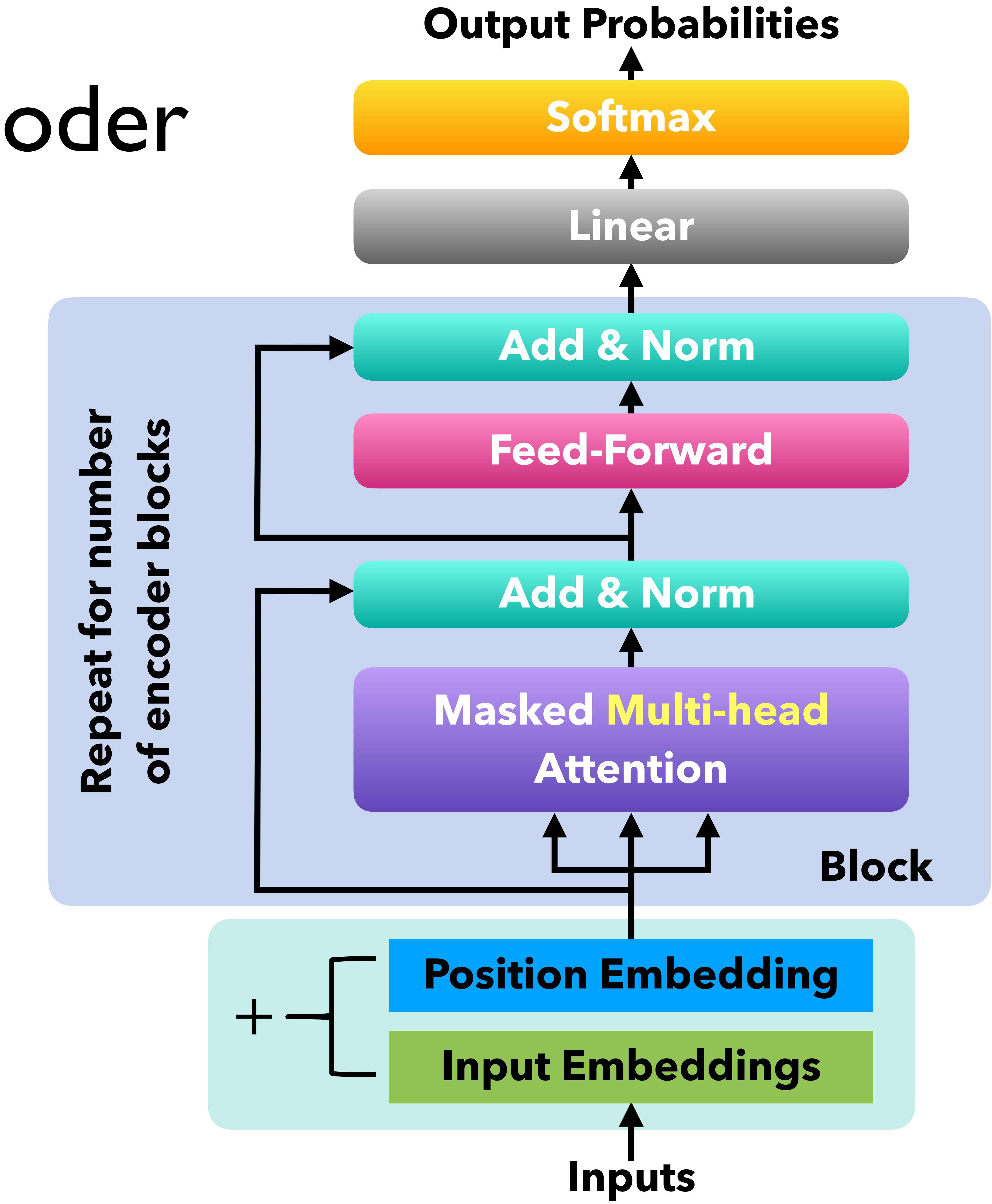
- **"Scaled Dot Product"** attention aids in training.
- When dimensionality d becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.

- Instead of the self-attention function we've seen:
  - $O^l = \text{softmax}(I\ W_Q^l\ {W_K^l}^T\ I^T)\ I\ W_V^l$
- **We divide the attention scores by $\sqrt{d/h}$**, to stop the scores from becoming large just as a function of d/h (the dimensionality divided by the number of heads).

$$O^l = \textbf{softmax}\left(\frac{I\ W_Q^l\ {W_K^l}^T\ I^T}{\sqrt{d/h}}\right)\ I\ W_V^l$$

The Transformer Decoder

- Now that we've replaced self-attention with multi-head self-attention, we'll go through two **optimization tricks**:
  - ***Residual connection ("Add")***
  - ***Layer normalization ("Norm")***

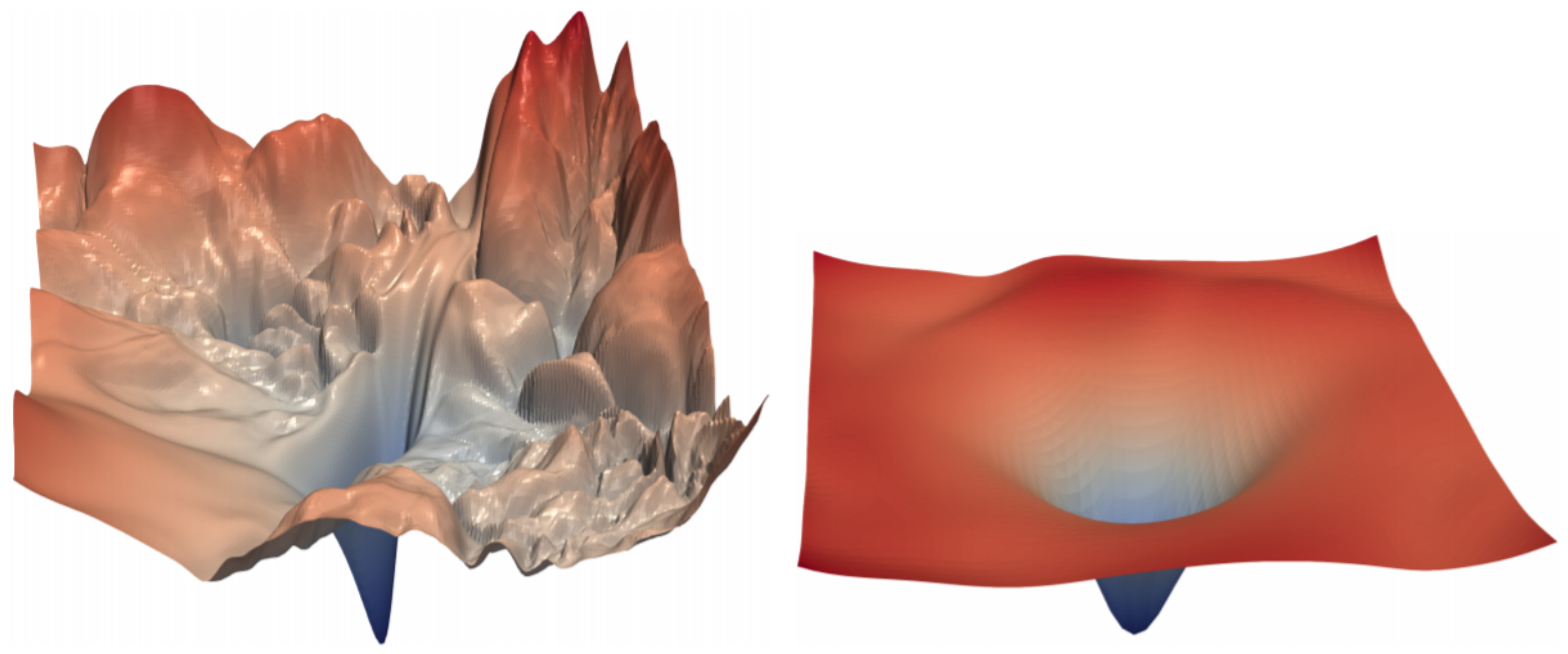


Residual Connections

- Residual connections are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)

  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)

  - Gradient is great through the residual connection; it's 1!
  - Bias towards the identity function!



[no residuals] **[residuals]**

[Loss landscape visualization, Li et al., 2018, on a ResNet]

Layer Normalization

- Layer normalization is a trick to help models train faster.
- **Idea:** cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within each layer.
  - LayerNorm's success may be due to its normalizing gradients [Xu et al., 2019]
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.
- Let $\mu = \sum_{j=1}^{d} x_j$; this is the mean; $\mu \in \mathbb{R}$.
- Let $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^{d} \left(x_j - \mu\right)^2}$; this is the standard deviation; $\sigma \in \mathbb{R}$.
- Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)
- Then layer normalization computes:

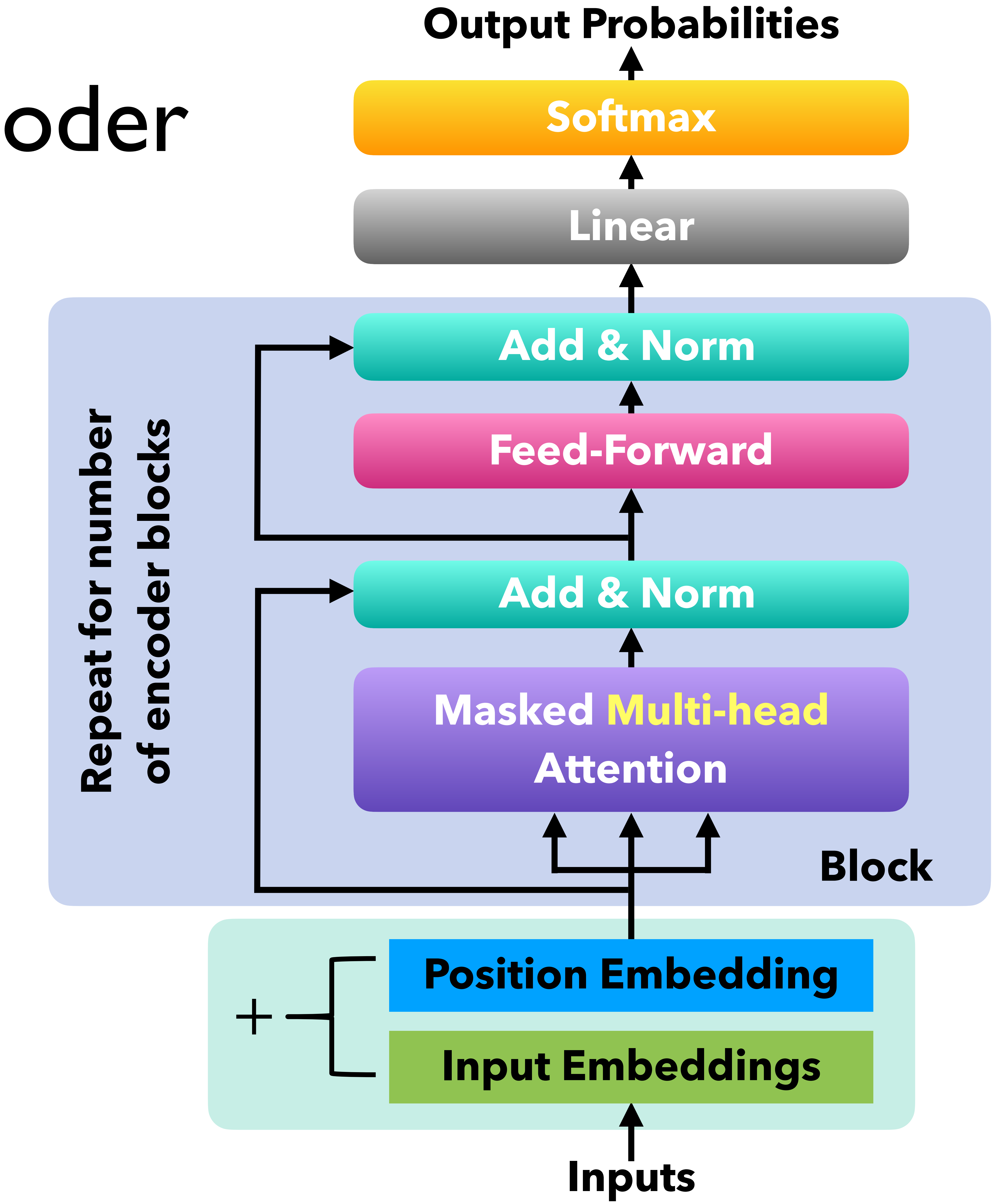
Normalize by scalar mean and variance

- $\text{output} = \frac{x - \mu}{\sqrt{\sigma} + \epsilon} * \gamma + \beta$

Modulate by learned element-wise gain and bias

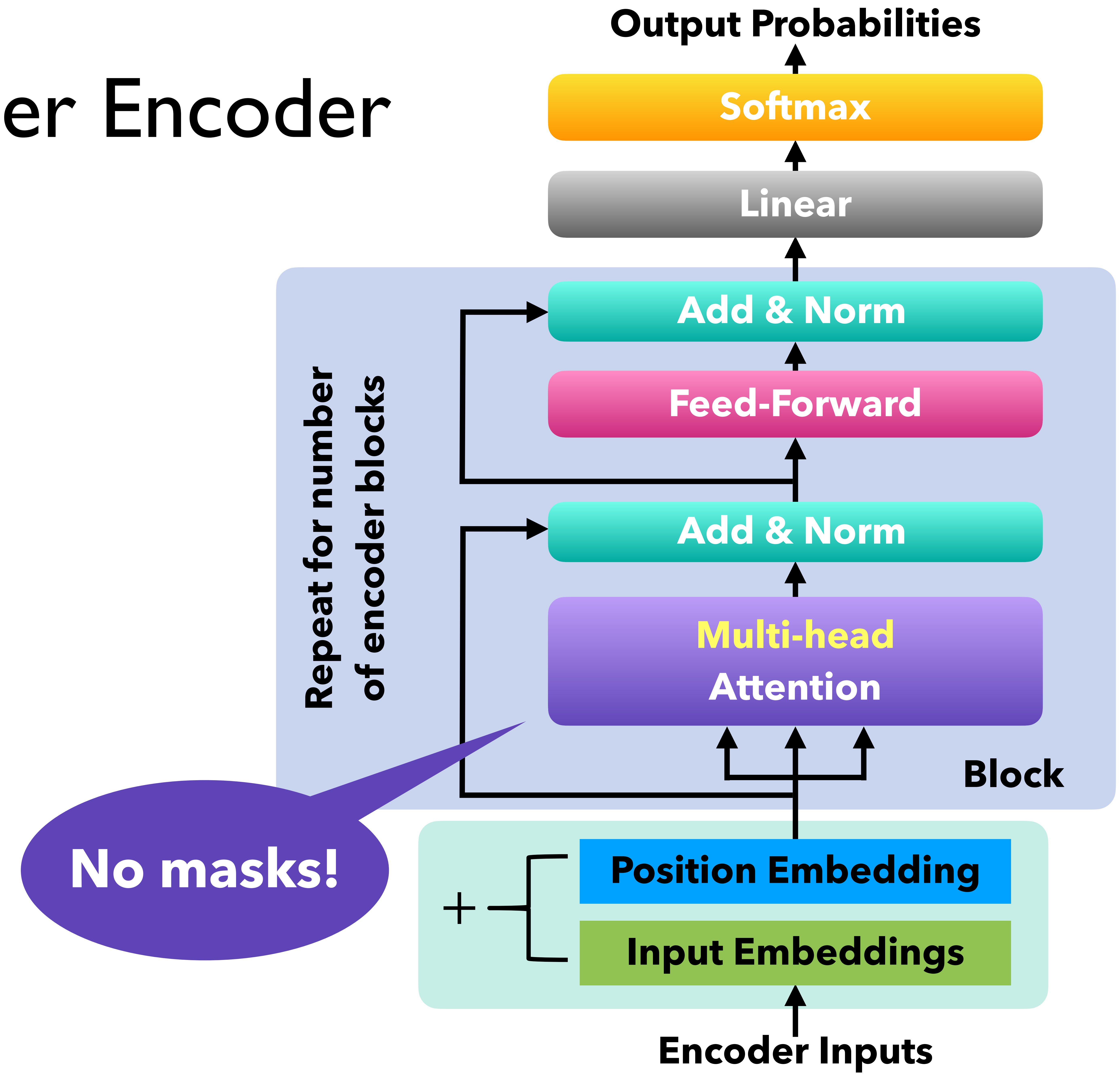
The Transformer Decoder

- The Transformer Decoder is a stack of Transformer Decoder **Blocks**.
- Each Block consists of:
  - Masked Multi-head Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm



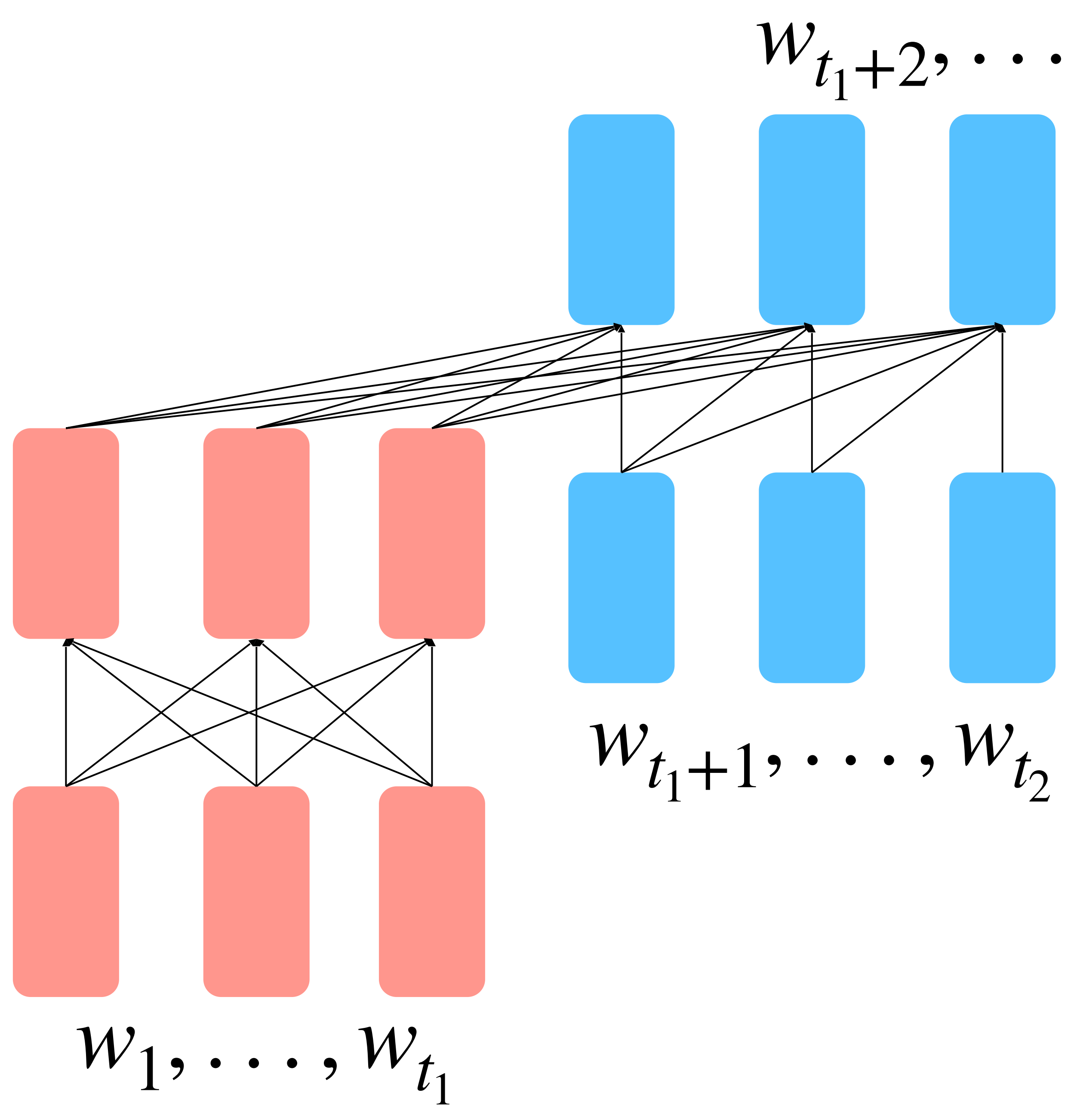
The Transformer Encoder

- The Transformer **Decoder** constrains to **unidirectional** context, as for language models.
- What if we want **bidirectional** context, like in a bidirectional RNN?
- We use **Transformer Encoder** – the ONLY difference is that we **remove the masking** in self-attention.

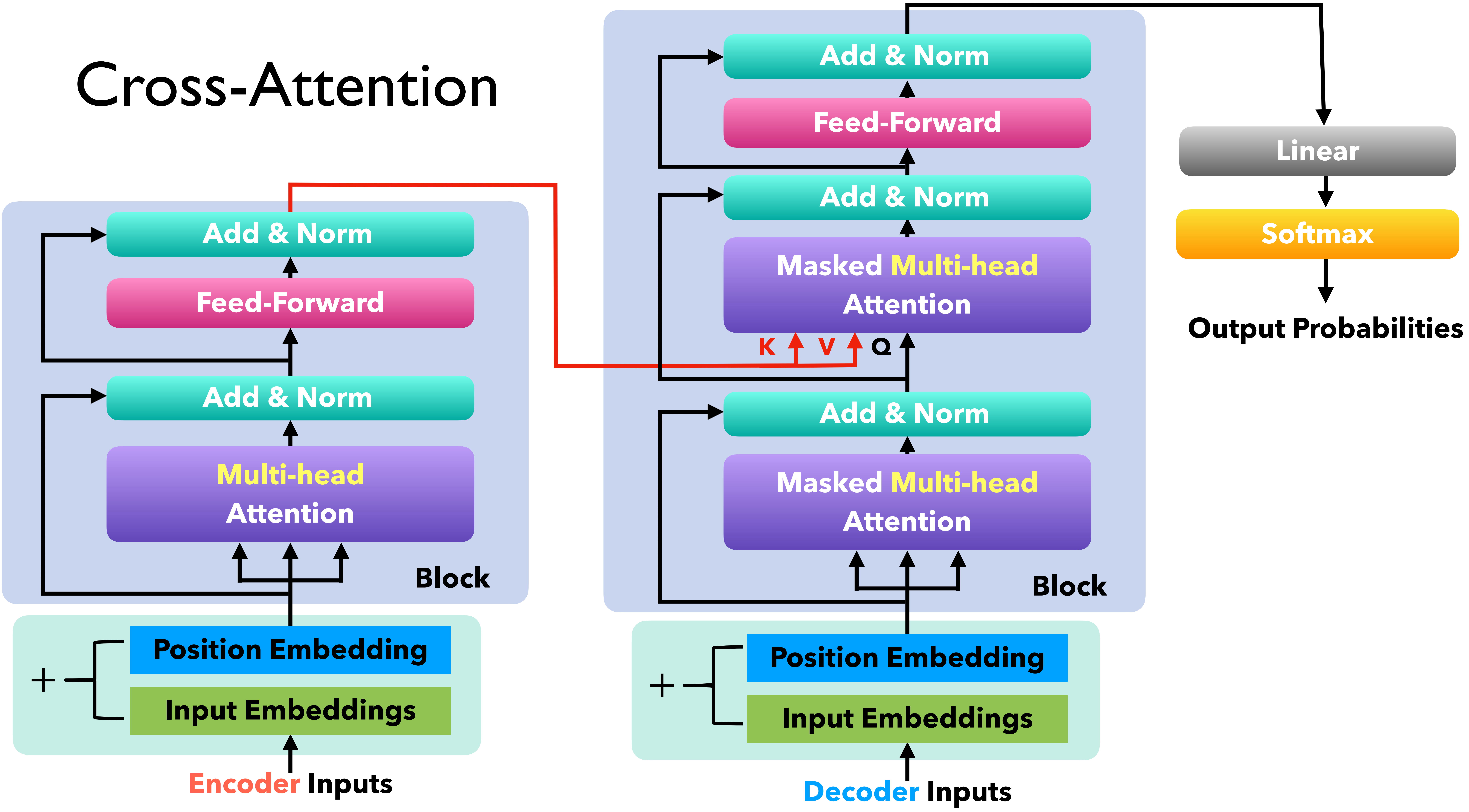


The Transformer Encoder-Decoder

- More on Encoder-Decoder models will be introduced in the next lecture!
- Right now we only need to know that it processes the source sentence with a **bidirectional** model (**Encoder**) and generates the target with a **unidirectional** model (**Decoder**).
- The Transformer Decoder is modified to perform **cross-attention** to the output of the Encoder.



Cross-Attention



Cross-Attention Details

- **Self-attention:** queries, keys, and values come from the same source.
- **Cross-Attention:** *keys* and *values* are from **Encoder** (like a memory); *queries* are from **Decoder**.
- Let $h_1, \ldots, h_n$ be output vectors from the Transformer **encoder**, $h_i \in \mathbb{R}^d$.
- Let $z_1, \ldots, \ z_n$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$.
- **Keys** and **values** from the **encoder**:
  - $k_i = W_K \, h_i$
  - $v_i = W_V \, h_i$
- **Queries** are drawn from the **decoder**:
  - $q_i = W_Q \, z_i$

Transformers: pros and cons

- **Easier to capture long-range dependencies**: we draw attention between every pair of words!
- **Easier to parallelize:**

$$Q = XW^Q \qquad K = XW^K \qquad V = XW^V$$

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

- **Are positional encodings enough to capture positional information?**

Otherwise self-attention is an unordered function of its input

- **Quadratic computation in self-attention**

Can become very slow when the sequence length is large

Quadratic computation as a function of sequence length

$$Q = XW^Q \qquad K = XW^K \qquad V = XW^V$$

$$\boxed{\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{\overset{n \times d_q}{Q}\,\overset{d_k \times n}{K^T}}{\sqrt{d_k}}\right)V} \longleftarrow n \times d_v$$

Need to compute n^2 pairs of scores (= dot product) $O(n^2 d)$

RNNs only require $O(nd^2)$ running time:

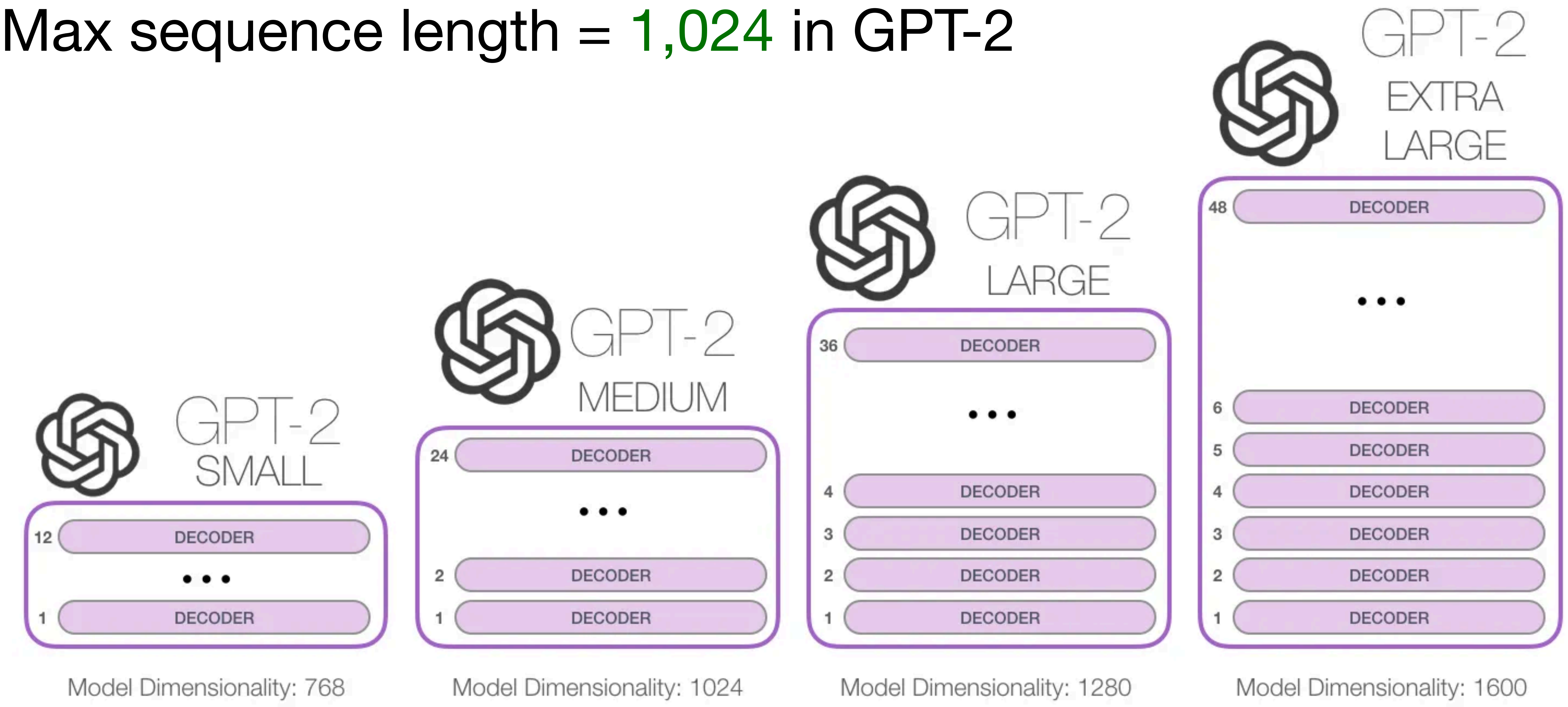
$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

(assuming input dimension = hidden dimension = d)

Quadratic computation as a function of sequence length

Need to compute n^2 pairs of scores (= dot product) O(n^2d)

Max sequence length = 1,024 in GPT-2

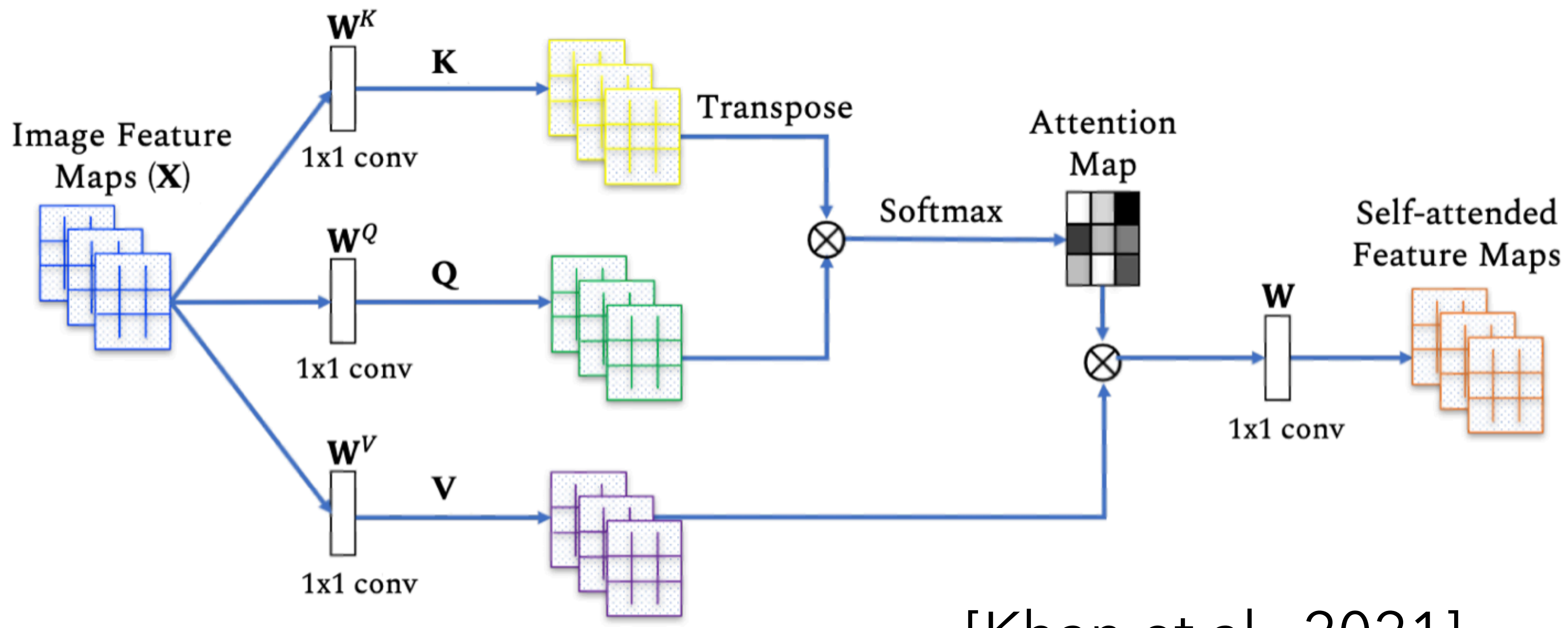


What if we want to scale $n \geq 50{,}000$? For example, to work on long documents?

The Revolutionary Impact of Transformers

- **Almost all current-day leading language models** use Transformer building blocks.
  - E.g., GPT1/2/3/4, T5, Llama 1/2, BERT, … almost anything we can name
  - Transformer-based models dominate nearly all NLP leaderboards.

- Since Transformer has been popularized in language applications, computer vision also adapted Transformers, e.g., **Vision Transformers**.



[Khan et al., 2021]

What's next after Transformers?