CMPU 366 · Natural Language Processing

Attention and Transformers

3 November 2025

www.buzzfeed.com/rsk17/if-you-were-elizabeth-bennet-could-you-ge

BuzzFeed

Sign In

Quizzes Shopping Trending News Celebrity Buzz Chat Arcade

QUIZ

Community • Posted on May 14, 2019

Subscribe to Quizzes Newsletter

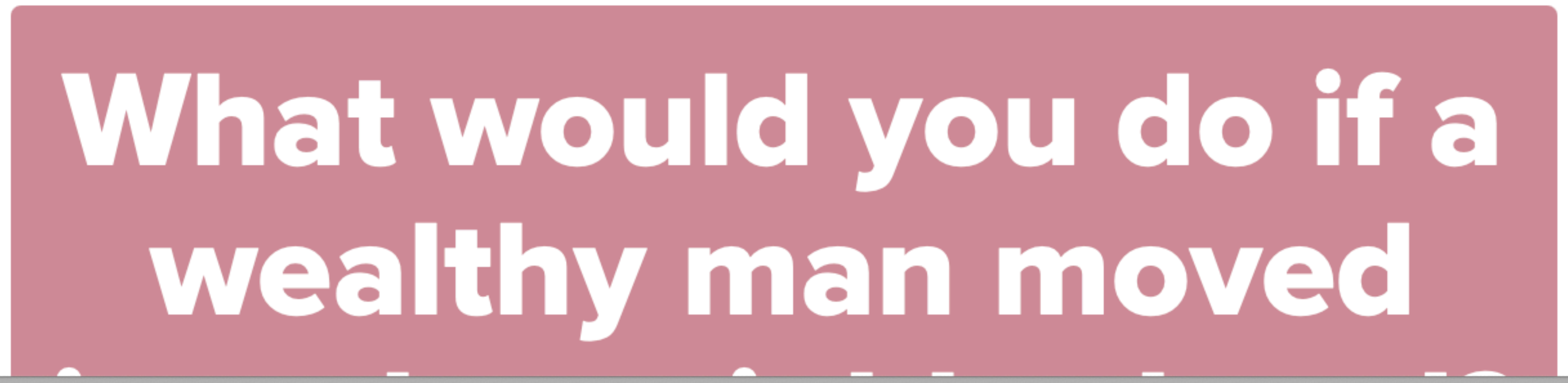
Can You Get Mr. Darcy From "Pride And Prejudice" To Propose To You?

If you can manage to avoid Mr. Collins, of course.

by rsk17
Community Contributor
820 points

Approved and edited by
BuzzFeed Community Team

View All 14 Comments



First half of the semester

Breadth

Individual work

Learn and practice foundational NLP concepts

- Rule-based systems (regular expressions)
- Language models
- Logistic regression
- Word embeddings
- Deep learning

First half of the semester

Breadth

Individual work

Learn and practice foundational NLP concepts

- Rule-based systems (regular expressions)
- Language models
- Logistic regression
- Word embeddings
- Deep learning

Second half of the semester

Depth

Group work

Deep and sustained creative problem-solving to build complex projects around one topic

Special topics

This is a chance to explore a topic we didn't cover yet. It's a good idea for this to be on the general area you'd like your final project to be in – though it doesn't need to be!

Sign up for a topic/day before the start of next class.

Projects

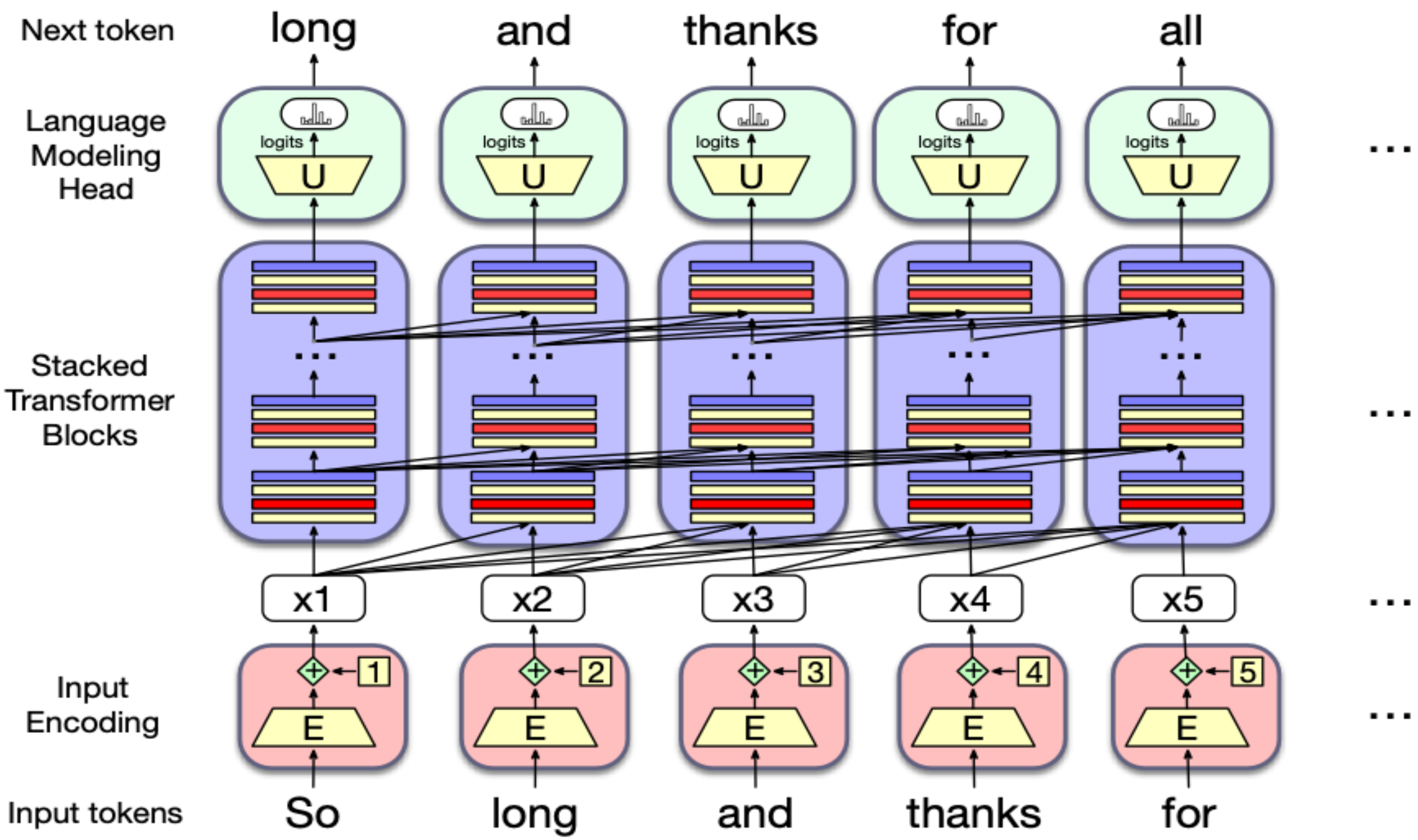
(Short) proposals will be due next Tuesday.

We'll use class time next Wednesday to start working on projects.

LLMs are built out of transformers.

A *transformer* is a specific kind of network architecture – like a fancier feedforward network, but based on *attention*.

Attention


Next token
long
and
thanks
for
all
Language Modeling Head
logits
U
Stacked Transformer Blocks
x1
x2
x3
x4
x5
Input Encoding
E
1
2
3
4
5
Input tokens
So
long
and
thanks
for


Next token
long
and
thanks
for
all
Language Modeling Head
logits
U
Stacked Transformer Blocks
Input Encoding
E
x1
x2
x3
x4
x5
Input tokens
So
long
and
thanks
for
Let's start by considering the embeddings for an individual word from a particular layer.

The problem with embeddings like Word2vec is they're *static*; the embedding for a word doesn't reflect how its meaning changes in *context*.

The chicken didn’t cross the road because it was too tired.

What does it mean in a static embedding?

The chicken didn't cross the road because it was too tired.

A better representation of the meaning of a word would be different in different contexts!

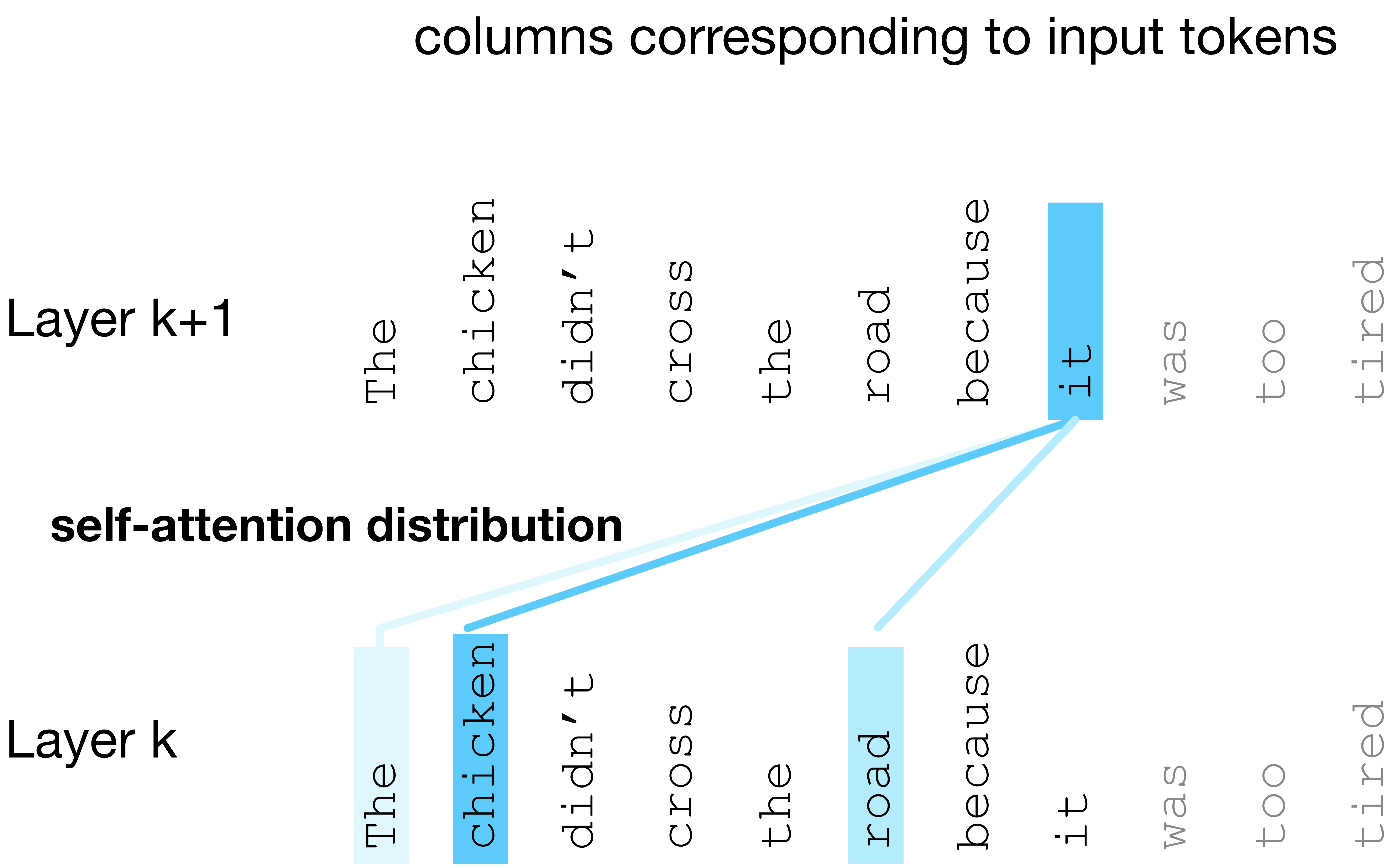
This is called a *contextual embedding*.

The chicken didn’t cross the road because it was too tired.

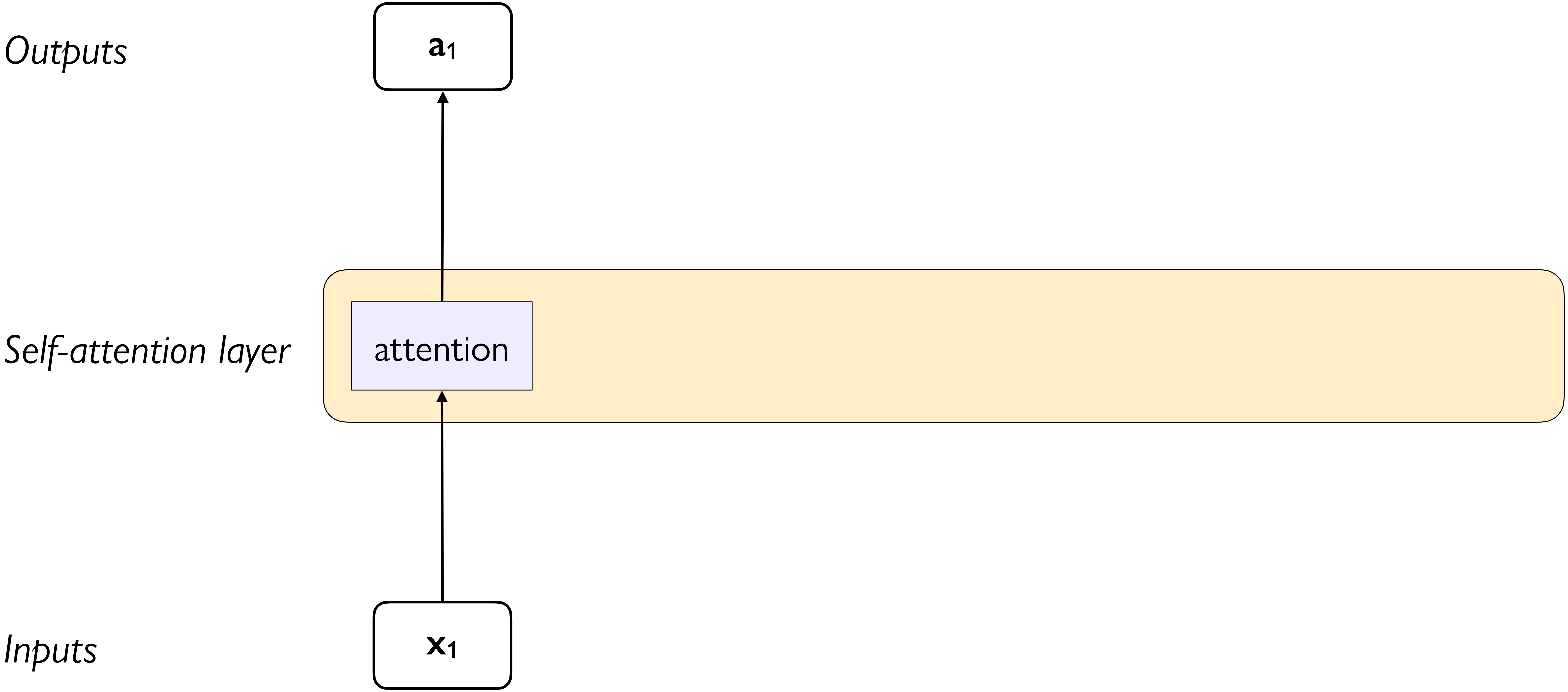
The chicken didn’t cross the road because it was too wide.

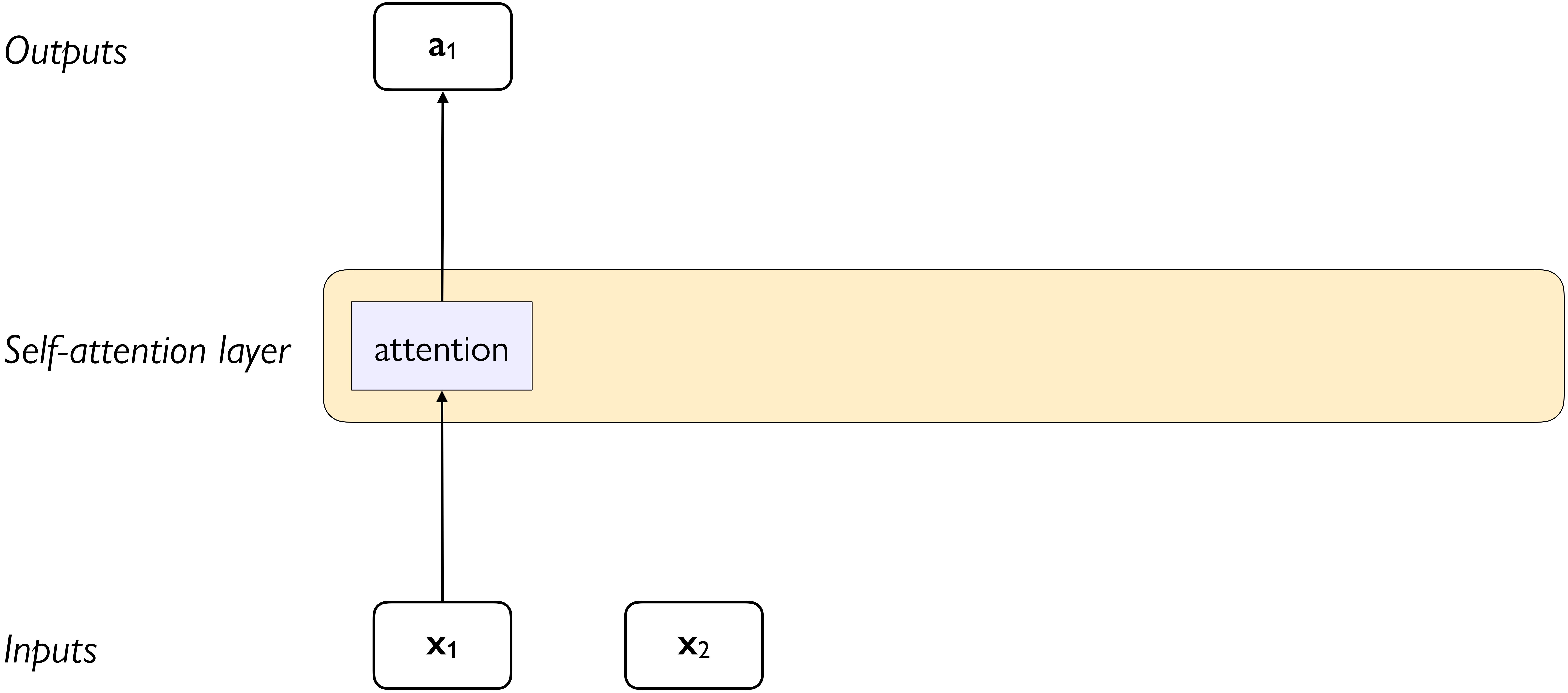
The chicken didn’t cross the road because it…

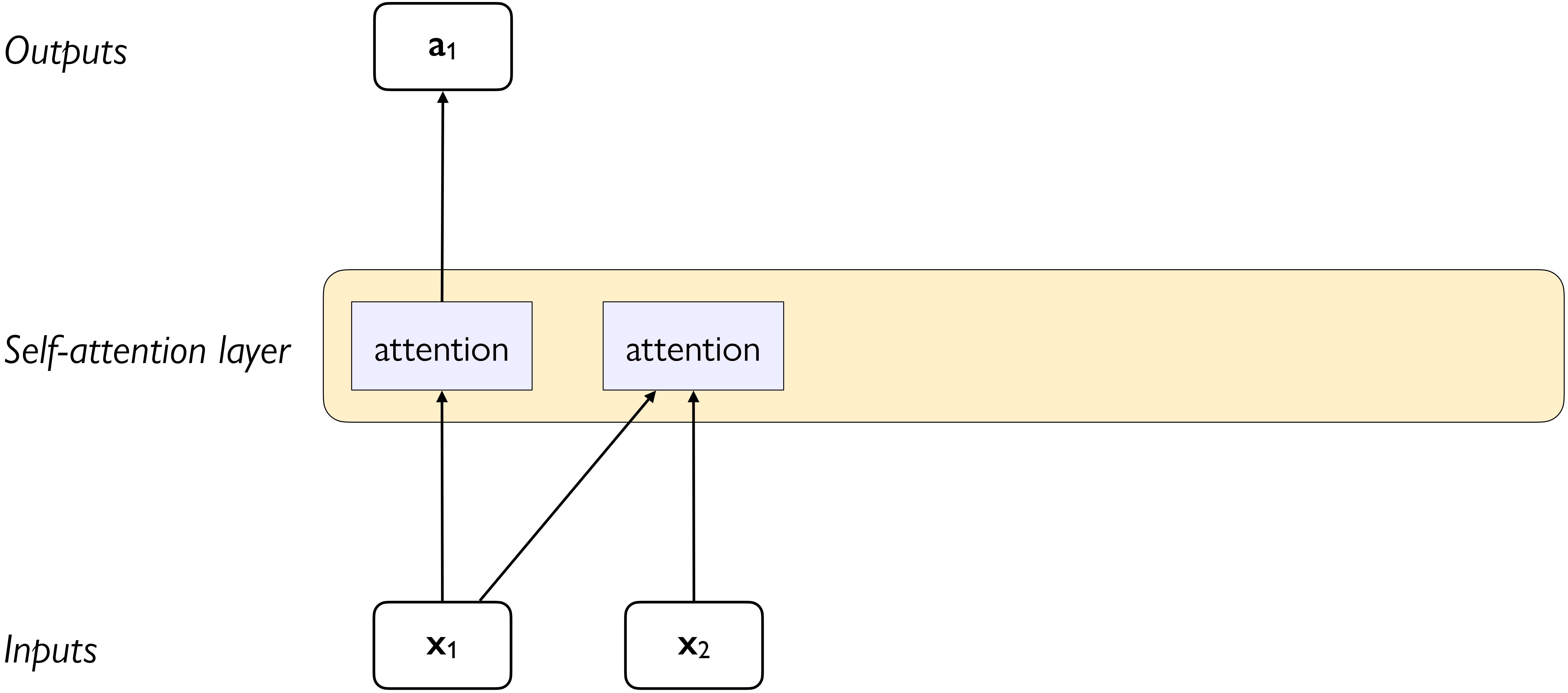
A transformer uses ***attention*** to build a contextual representation of a word by selectively integrating information from all the neighboring words.



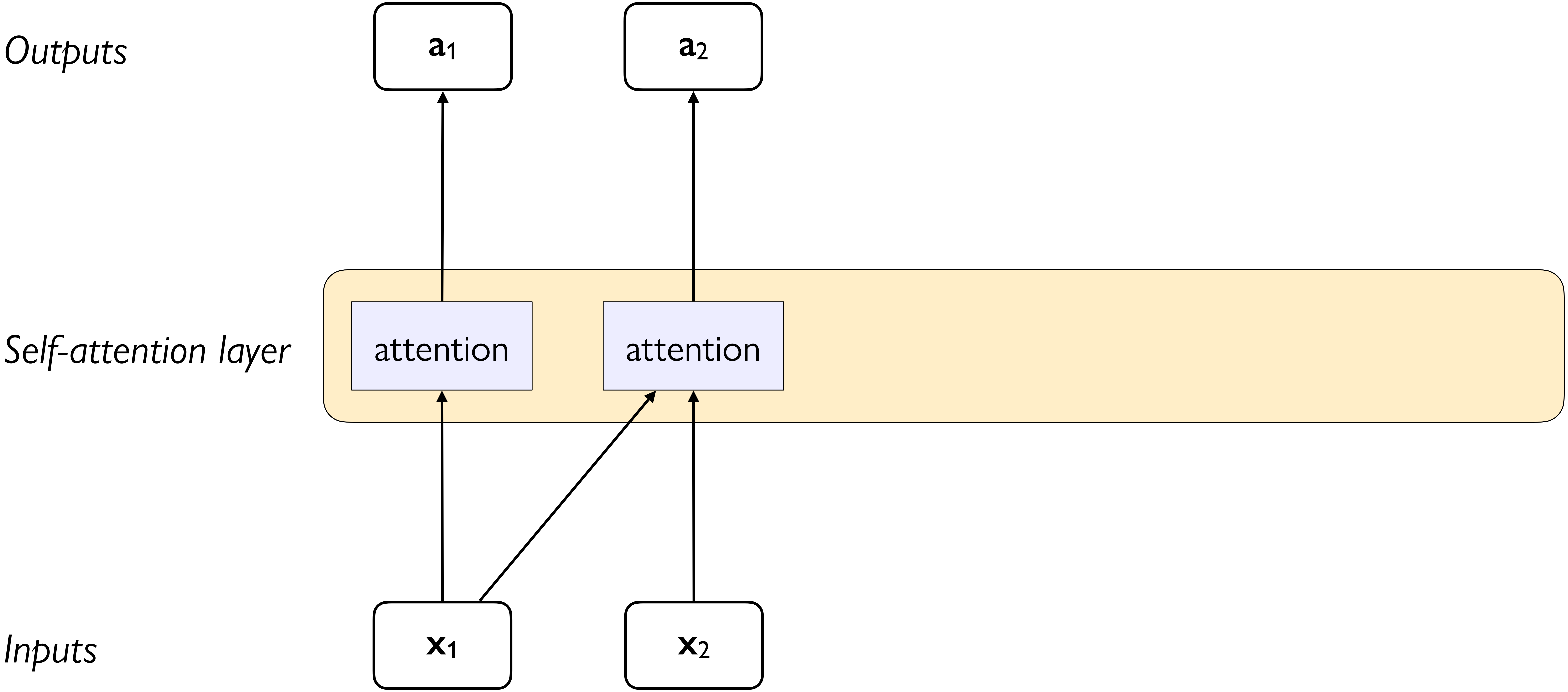
columns corresponding to input tokens
Layer k+1
The
chicken
didn't
cross
the
road
because
it
was
too
tired
self-attention distribution
Layer k
The
chicken
didn't
cross
the
road
because
it
was
too
tired

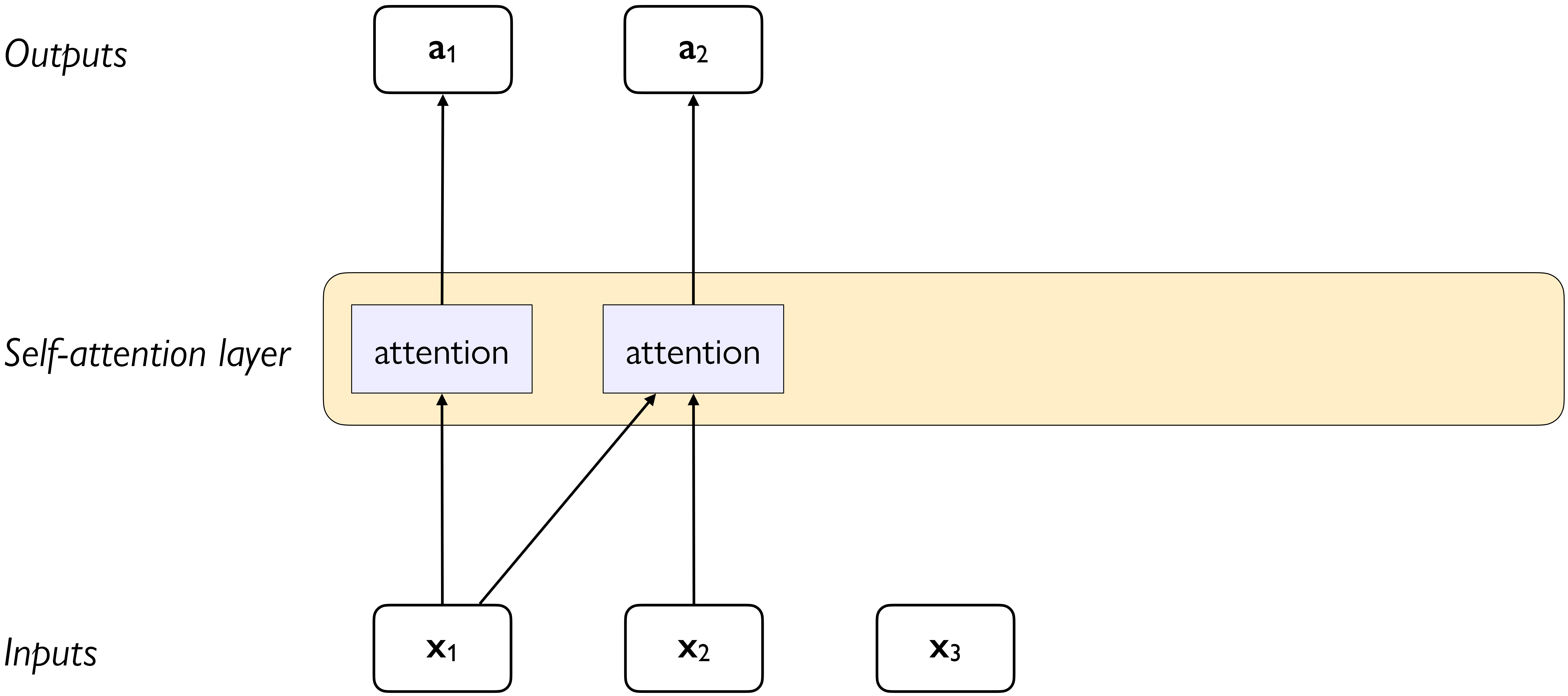

Outputs
a1
Self-attention layer
attention
Inputs
x1

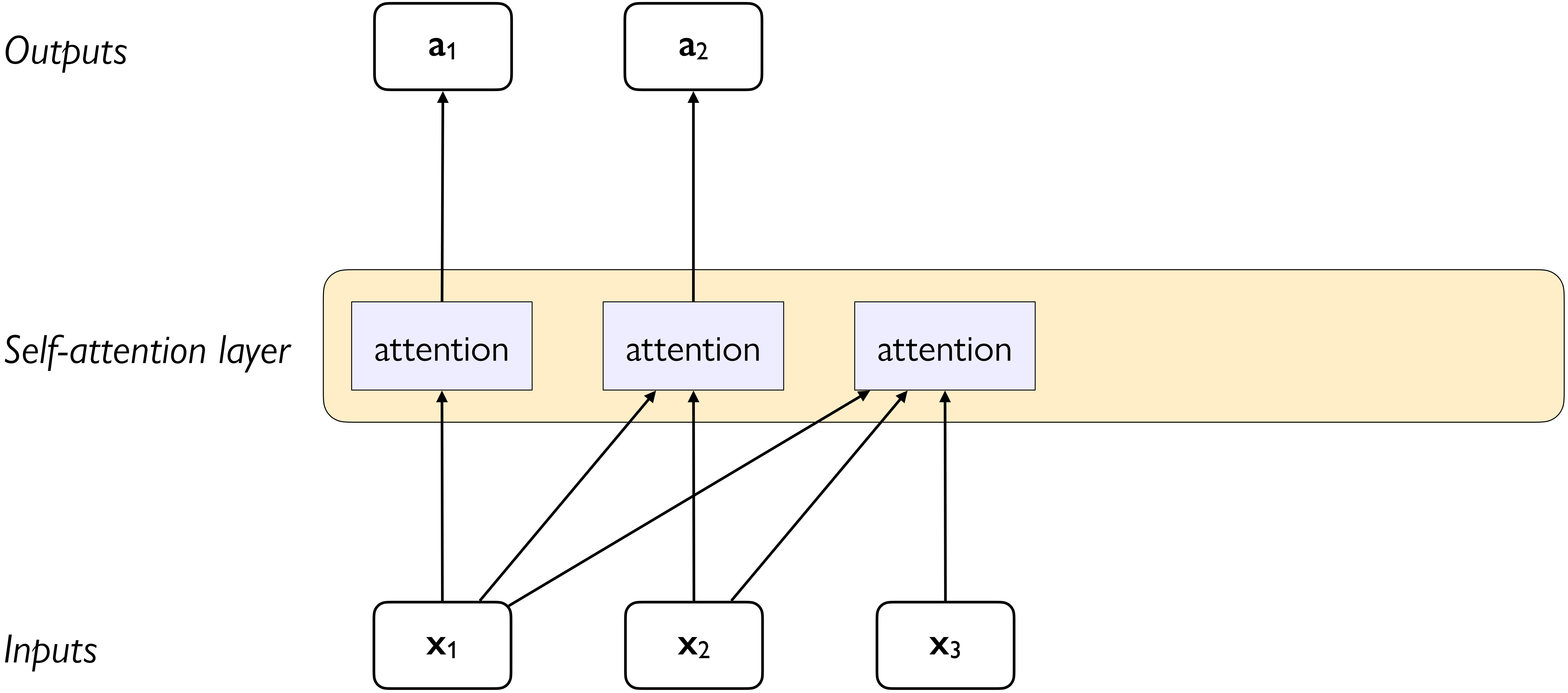

Outputs
a1
Self-attention layer
attention
Inputs
x1
x2

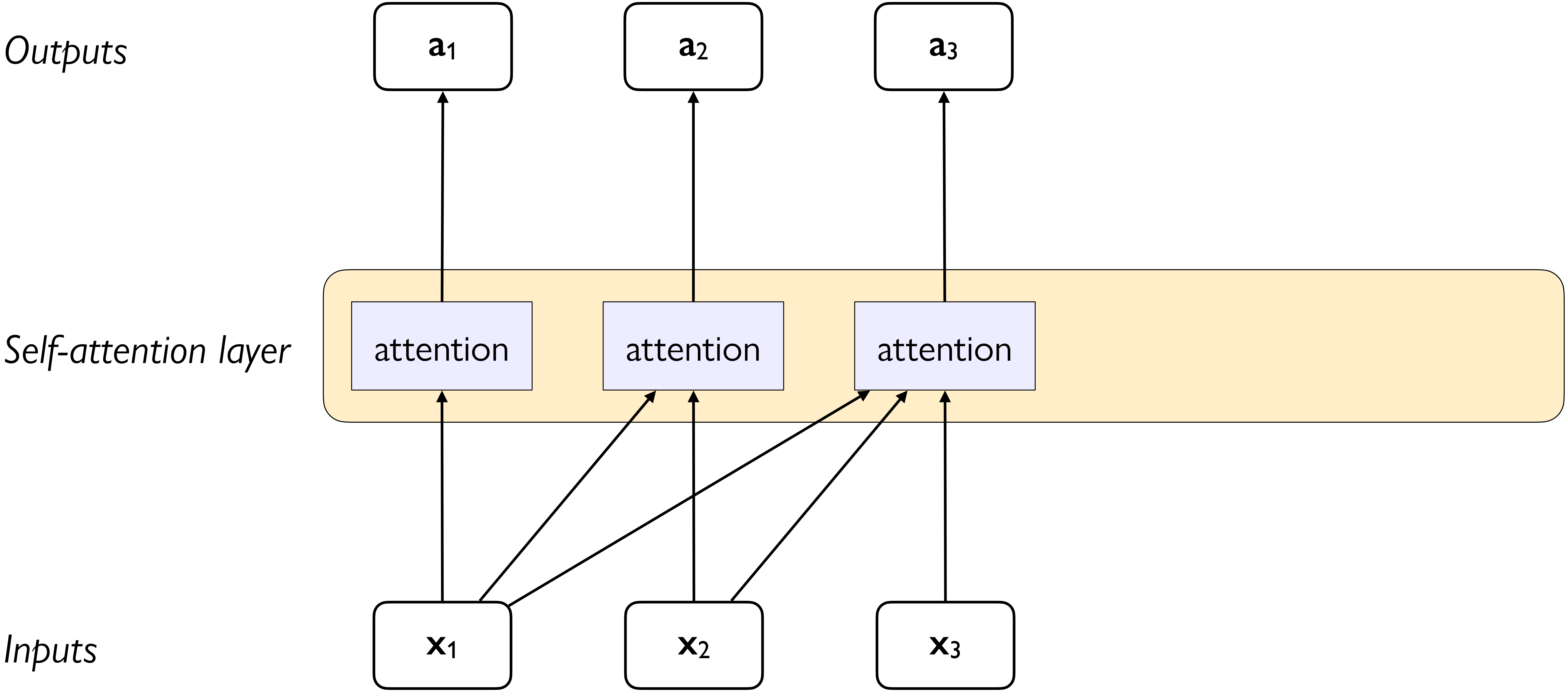


Outputs
a1
Self-attention layer
attention
attention
Inputs
x1
x2

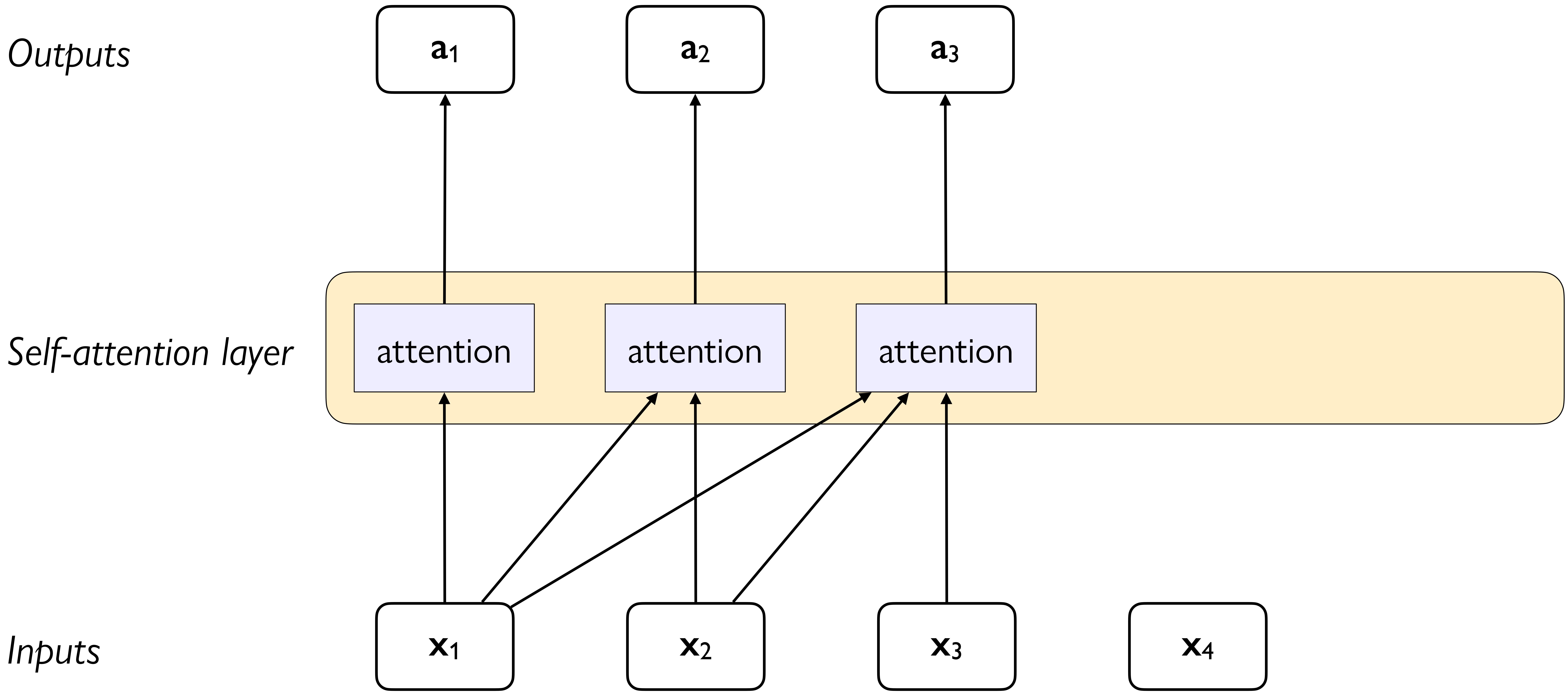

Outputs
a1
a2
Self-attention layer
attention
attention
Inputs
x1
x2


Outputs
a1
a2
Self-attention layer
attention
attention
Inputs
x1
x2
x3

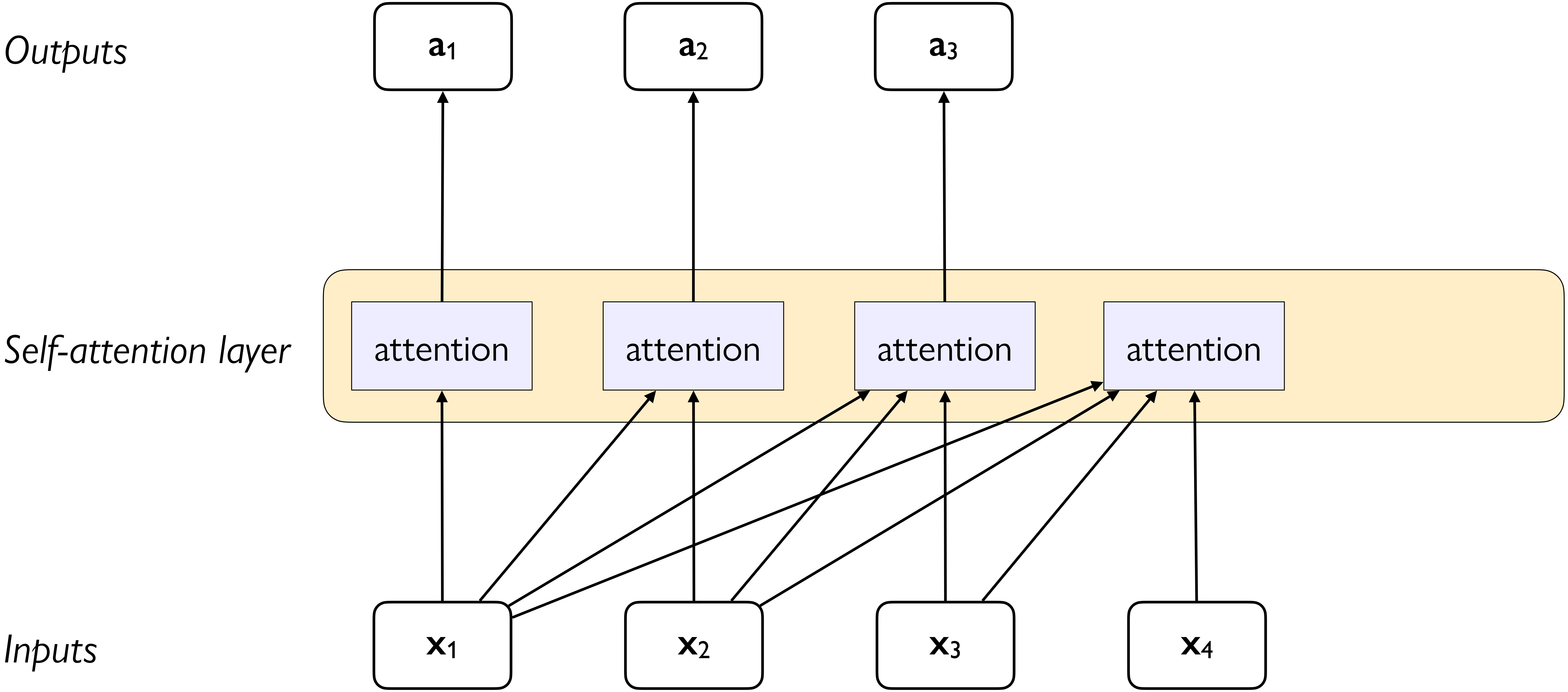

Outputs
a1
a2
Self-attention layer
attention
attention
attention
Inputs
x1
x2
x3

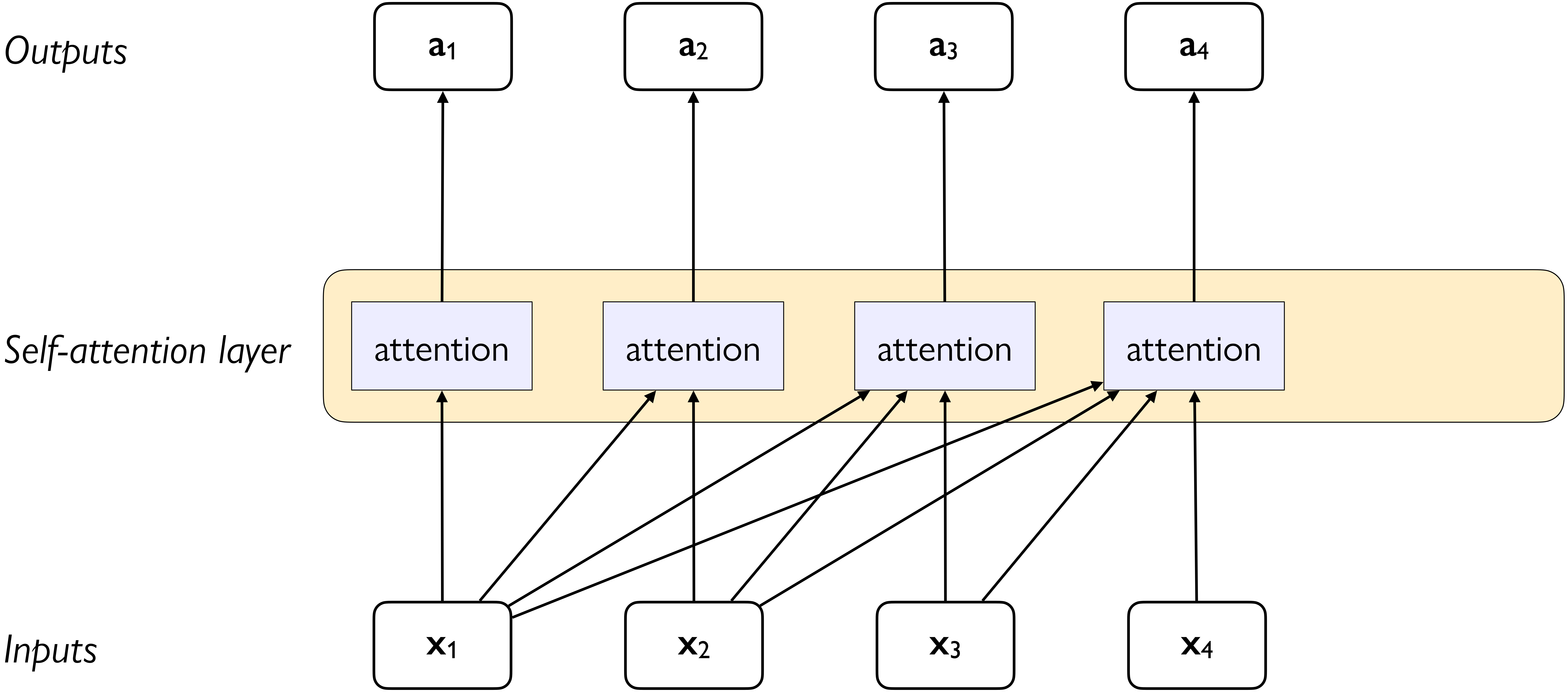


Outputs
a1
a2
a3
Self-attention layer
attention
attention
attention
Inputs
x1
x2
x3

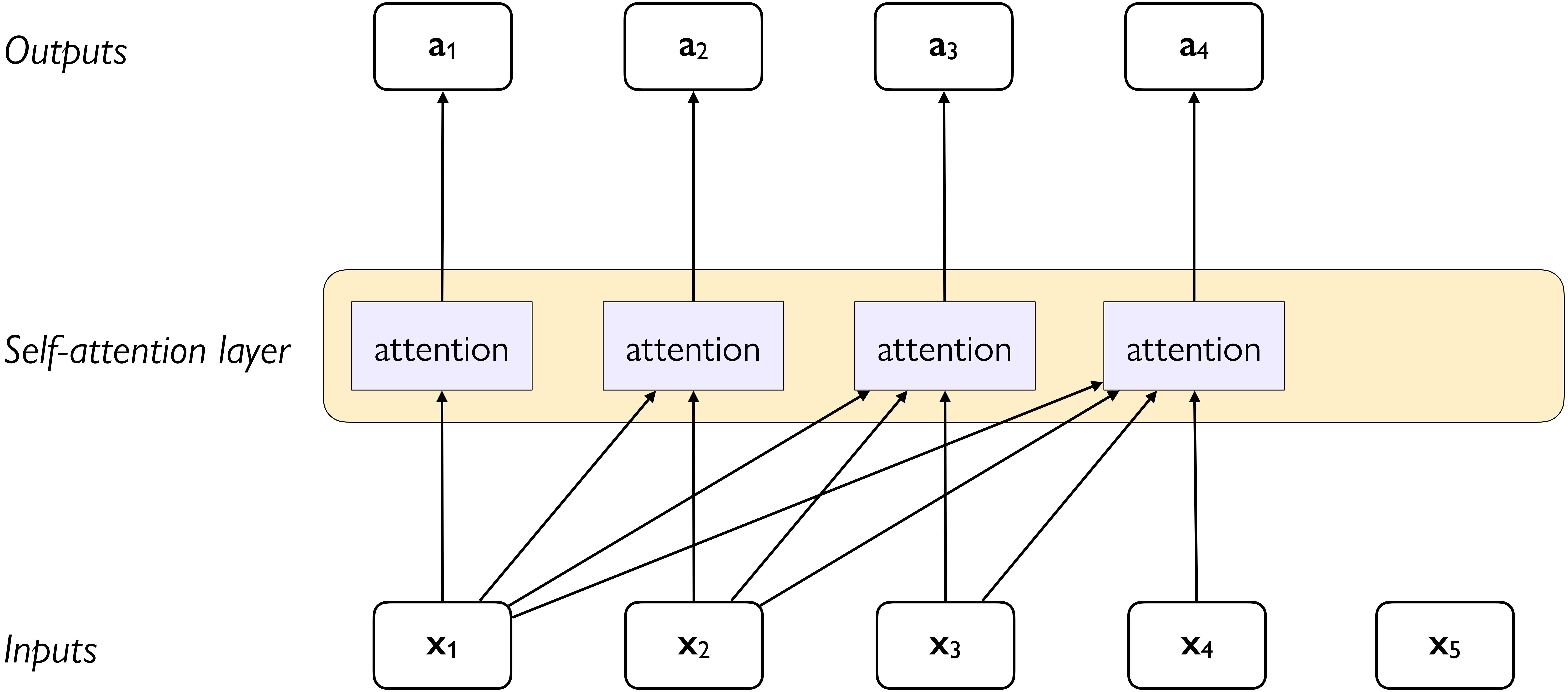


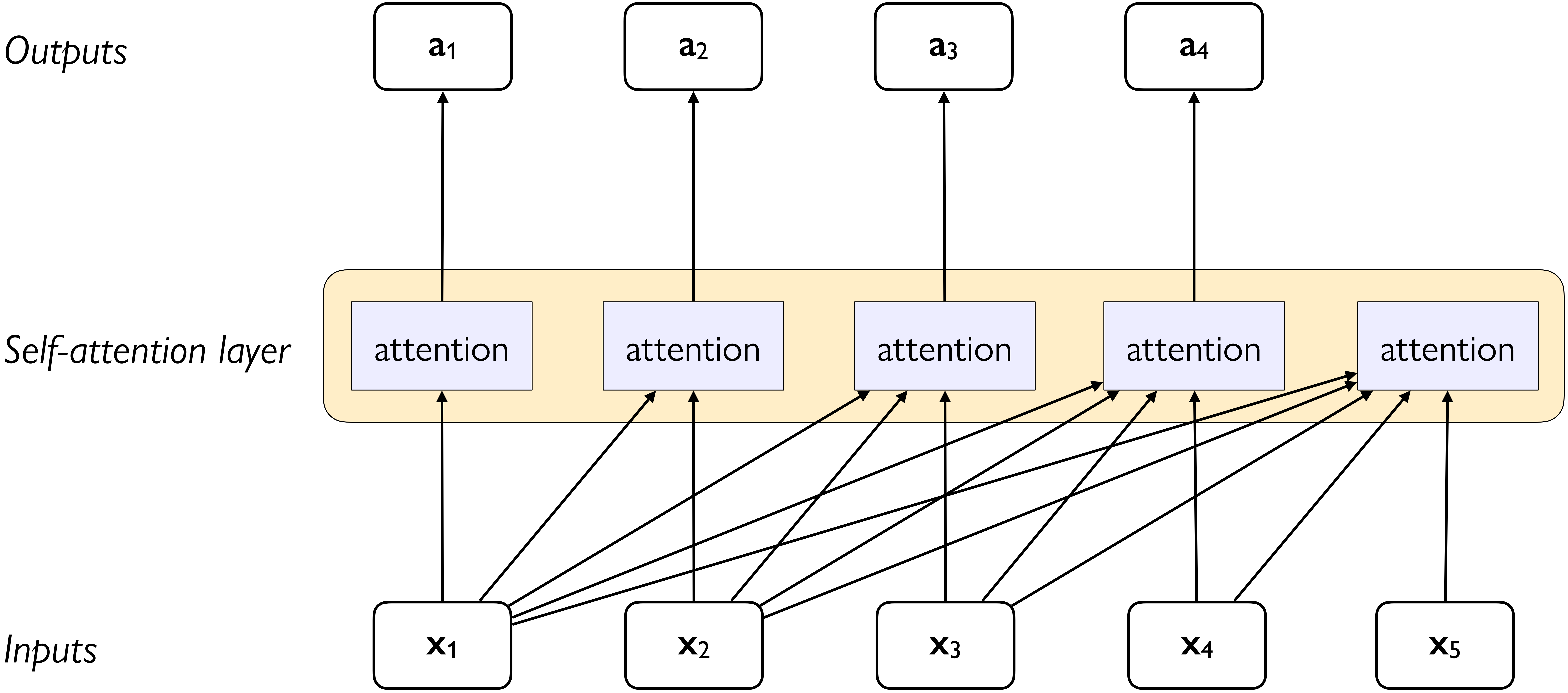
Outputs
a1
a2
a3
Self-attention layer
attention
attention
attention
Inputs
x1
x2
x3
x4


Outputs
a1
a2
a3
Self-attention layer
attention
attention
attention
attention
Inputs
x1
x2
x3
x4

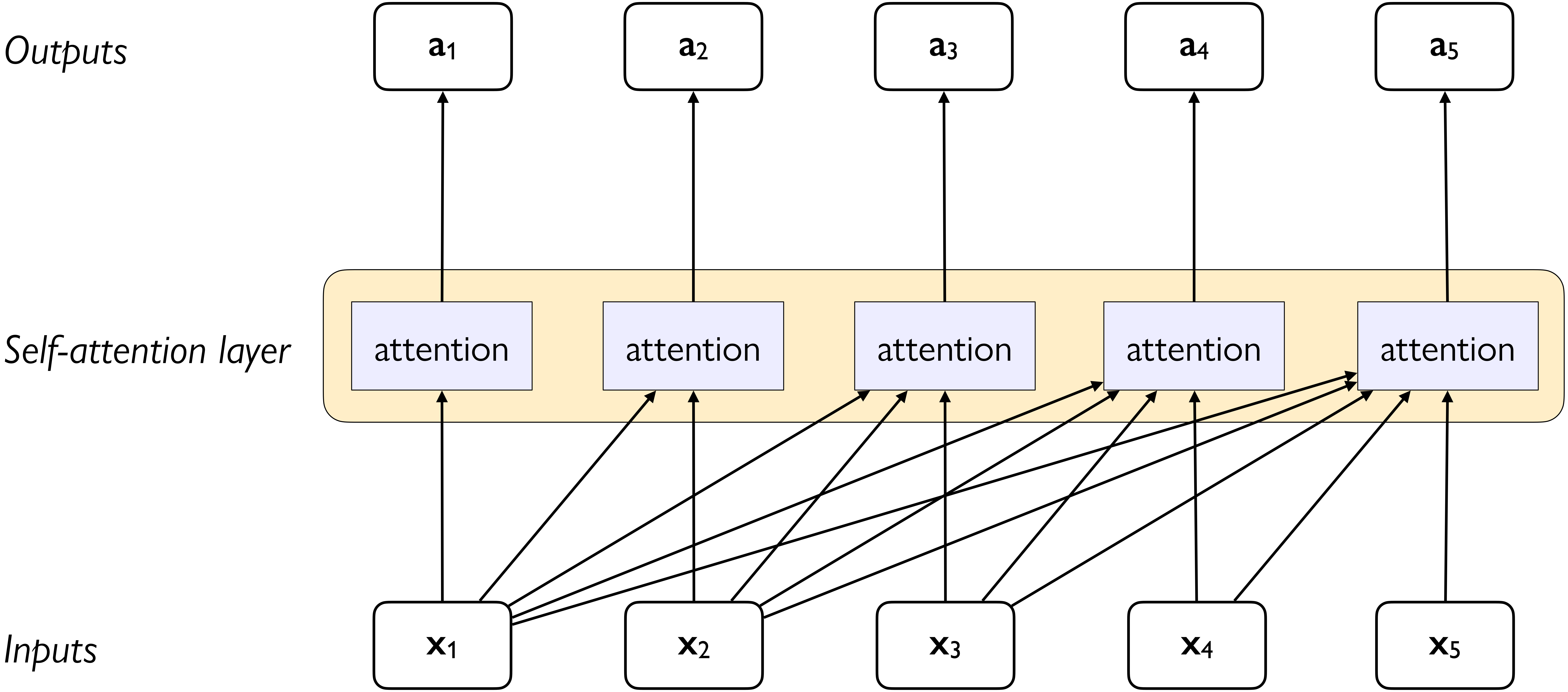


Outputs
Self-attention layer
Inputs
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{a}_3$
$\mathbf{a}_4$
attention
attention
attention
attention
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$


Outputs
a1
a2
a3
a4
Self-attention layer
attention
attention
attention
attention
Inputs
x1
x2
x3
x4
x5



Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{a}_3$
$\mathbf{a}_4$
Self-attention layer
attention
attention
attention
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$



Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{a}_3$
$\mathbf{a}_4$
$\mathbf{a}_5$
Self-attention layer
attention
attention
attention
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$

Given a sequence of token embeddings

$$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_6 \quad \mathbf{x}_7 \quad \mathbf{x}_i$$

we can produce a sum of the embeddings weighted by their similarity to $\mathbf{x}_i$:

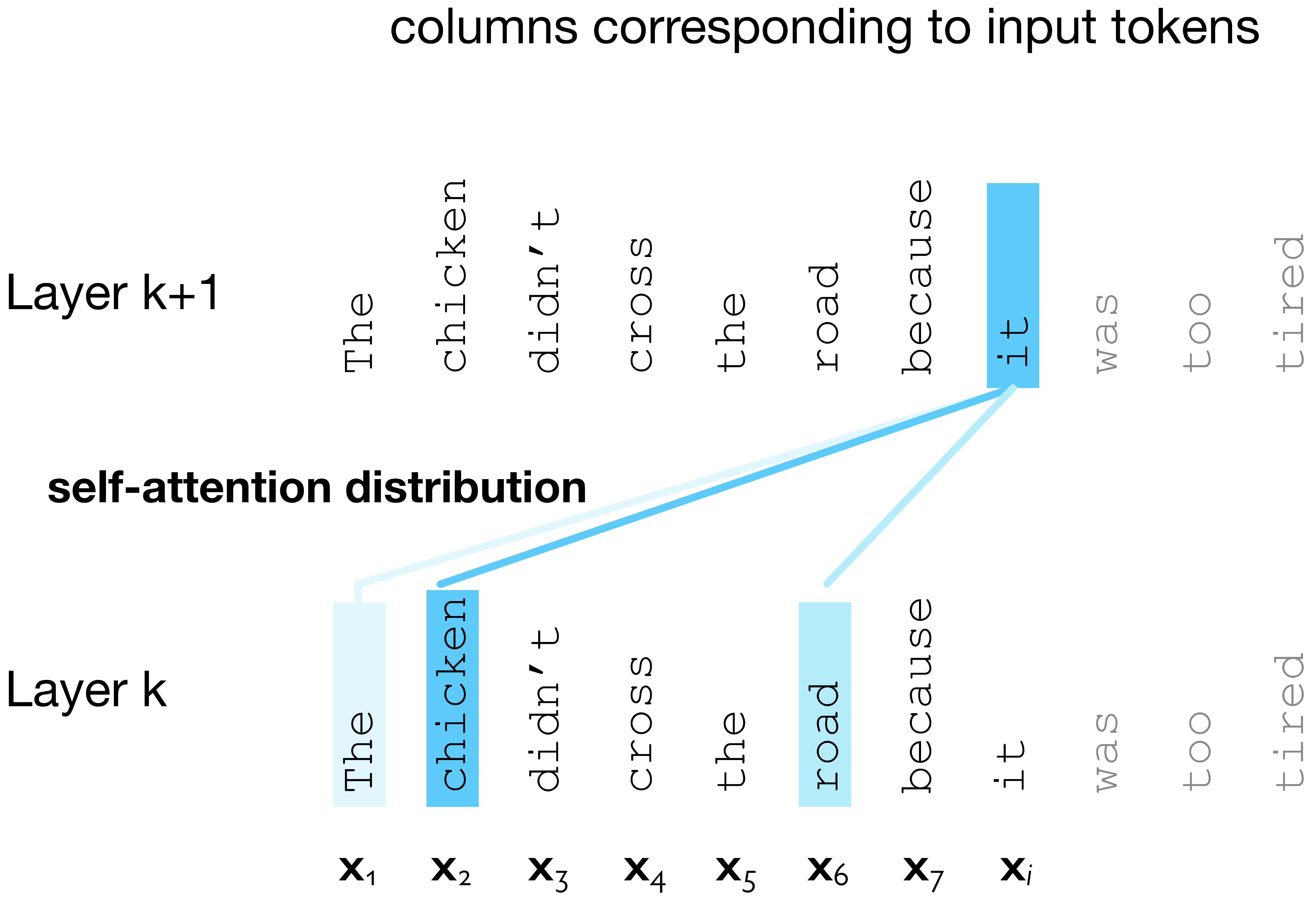
$$\mathbf{a}_i = \sum_{j \le i} \alpha_{ij} \mathbf{x}_j$$

where the weight

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \le i,$$

and we can simply define

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j.$$


columns corresponding to input tokens
Layer k+1
The
chicken
didn't
cross
the
road
because
it
was
too
tired
self-attention distribution
Layer k
The
chicken
didn't
cross
the
road
because
it
was
too
tired
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$ $\mathbf{x}_5$ $\mathbf{x}_6$ $\mathbf{x}_7$ $\mathbf{x}_i$

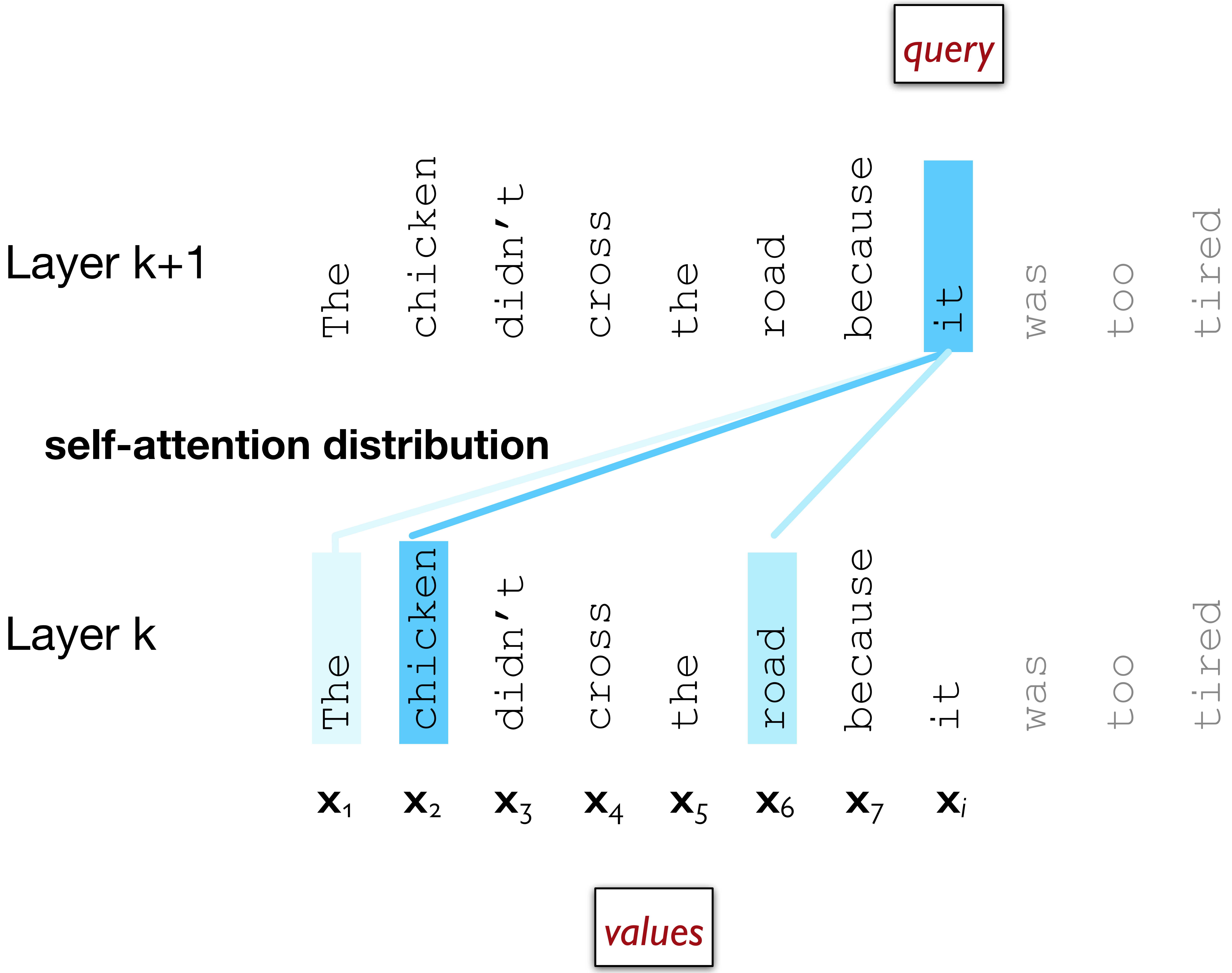
An actual attention head is slightly more complicated.

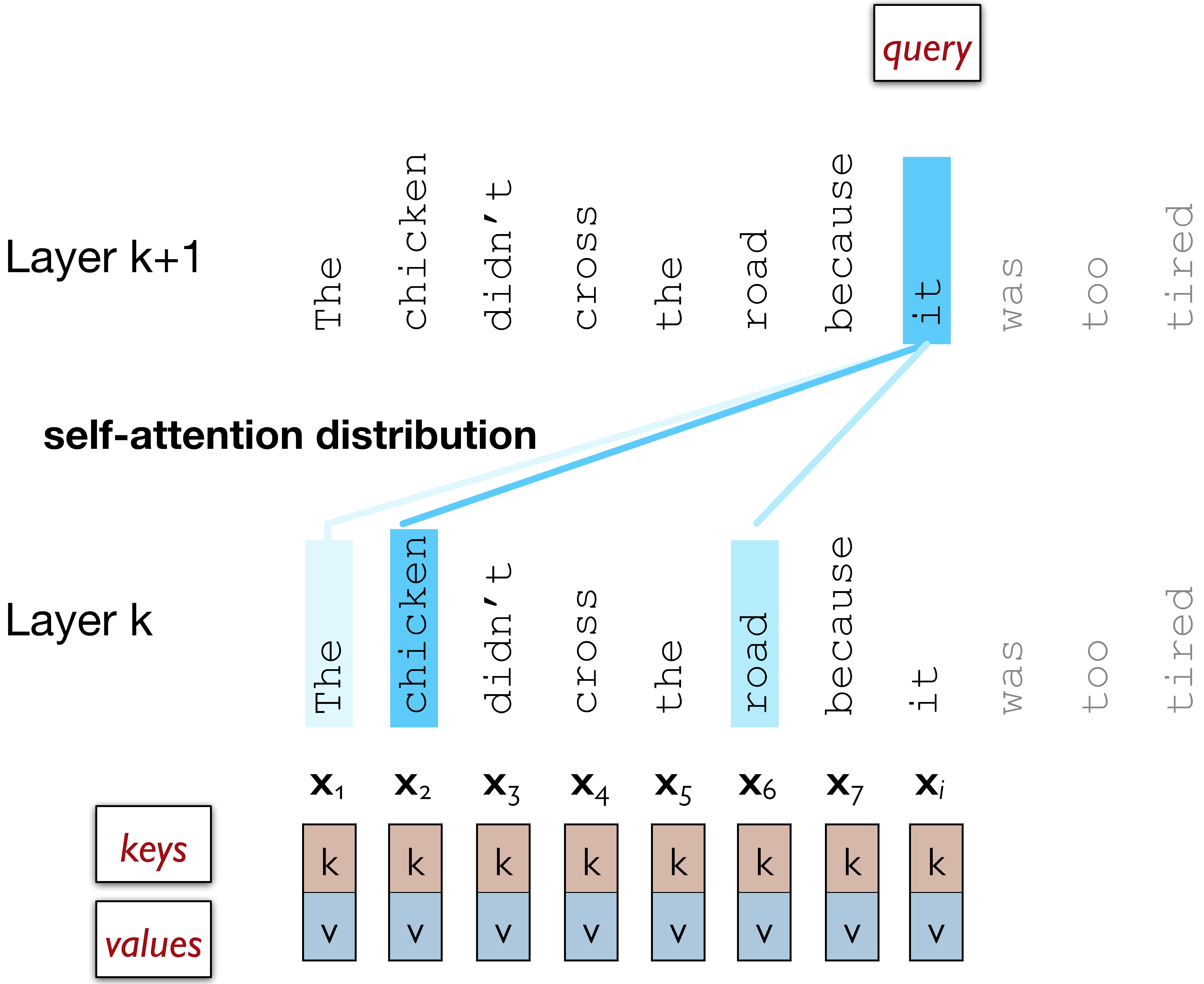
Instead of using vectors like $\mathbf{x}_i$ and $\mathbf{x}_4$ directly, we'll represent three separate ***roles*** each vector $\mathbf{x}_i$ plays:

Query: As the ***current element*** being compared to the preceding inputs

Key: As a ***preceding input*** that is being compared to the current element to determine a similarity

Value: a value of a preceding element that gets ***weighted and summed***.


query
Layer k+1
The chicken didn't cross the road because it was too tired
self-attention distribution
Layer k
The chicken didn't cross the road because it was too tired
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$ $\mathbf{x}_5$ $\mathbf{x}_6$ $\mathbf{x}_7$ $\mathbf{x}_i$
values


query
Layer k+1
The
chicken
didn't
cross
the
road
because
it
was
too
tired
self-attention distribution
Layer k
The
chicken
didn't
cross
the
road
because
it
was
too
tired
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_i
keys
values
k
v

We'll use weight matrices $\mathbf{W^Q}$, $\mathbf{W^K}$, and $\mathbf{W^V}$ to project each vector $\mathbf{x}_i$ into a representation of its role as a

query: $\mathbf{q}_i = \mathbf{x}_i\mathbf{W^Q}$

key: $\mathbf{k}_i = \mathbf{x}_i\mathbf{W^K}$

value: $\mathbf{v}_i = \mathbf{x}_i\mathbf{W^V}$

To compute the similarity of the current element $\mathbf{x}_i$ with some prior element $\mathbf{x}_j$, we'll use the dot product between the projections $\mathbf{q}_i$ and $\mathbf{k}_j$.

And instead of summing up $\mathbf{x}_j$, we'll sum up the projection $\mathbf{v}_j$.

Simplified	De-simplified
	$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}}$
	$\mathbf{k}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{K}}$
	$\mathbf{v}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{V}}$
$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$	$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$
$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \leq i$	$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \leq i$
$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$	$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}}$$

$$\mathbf{k}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{K}}$$

$$\mathbf{v}_j = \mathbf{v}_j \mathbf{W}^{\mathbf{V}}$$

d_k is the dimensionality of the query and key vectors

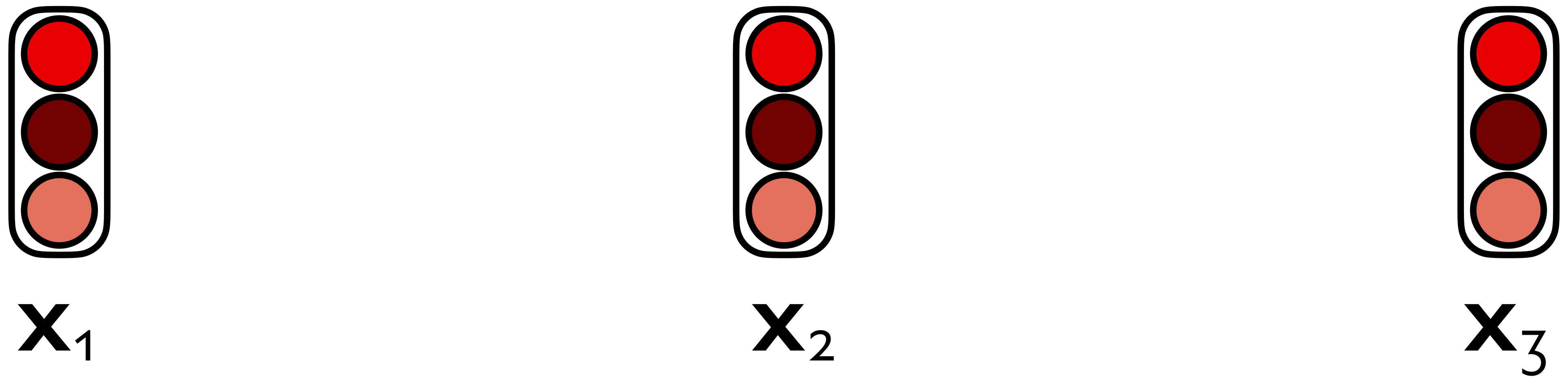
$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \leftarrow \text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$$

This is a practical change to avoid numerical issues during training.

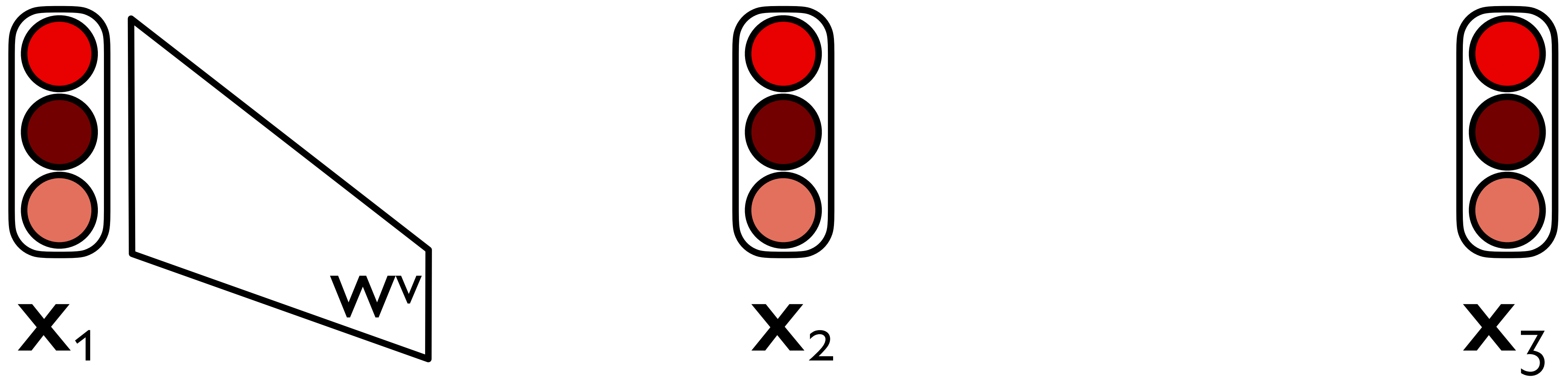
$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \ \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

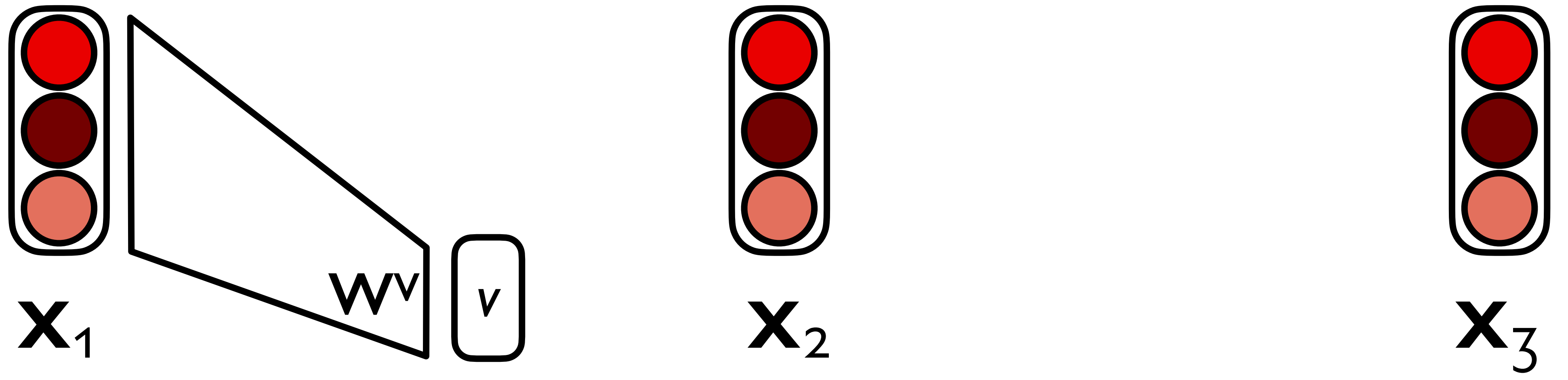
Project each $\mathbf{x}_i$ into key, query, and value vectors



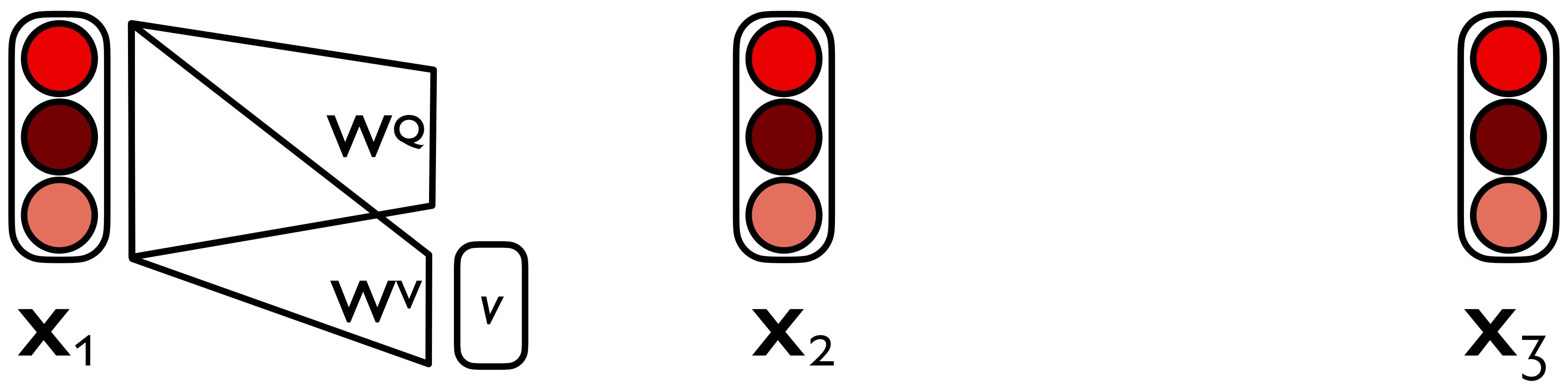
Project each $\mathbf{x}_i$ into key, query, and value vectors



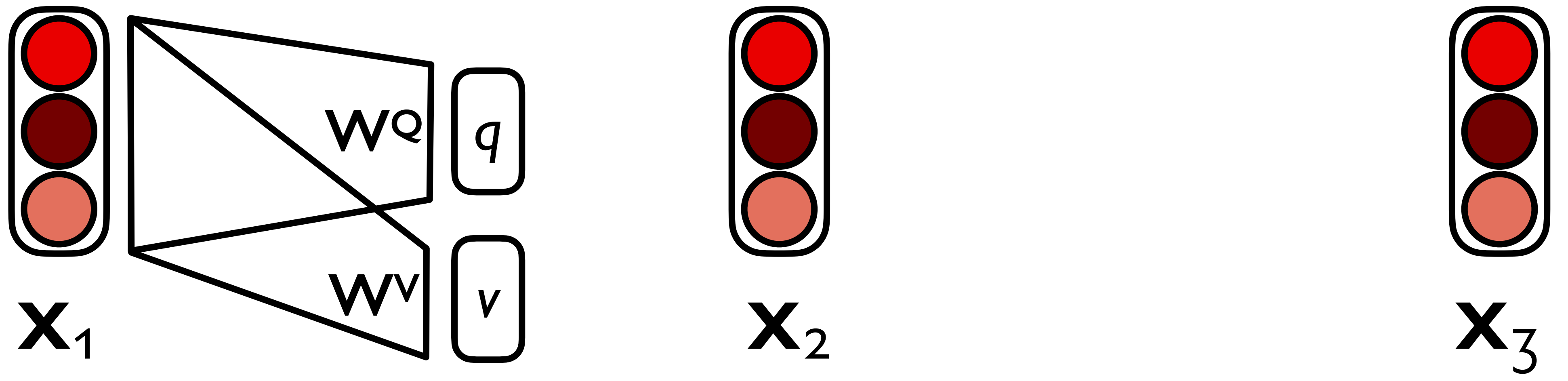
Project each $\mathbf{x}_i$ into key, query, and value vectors



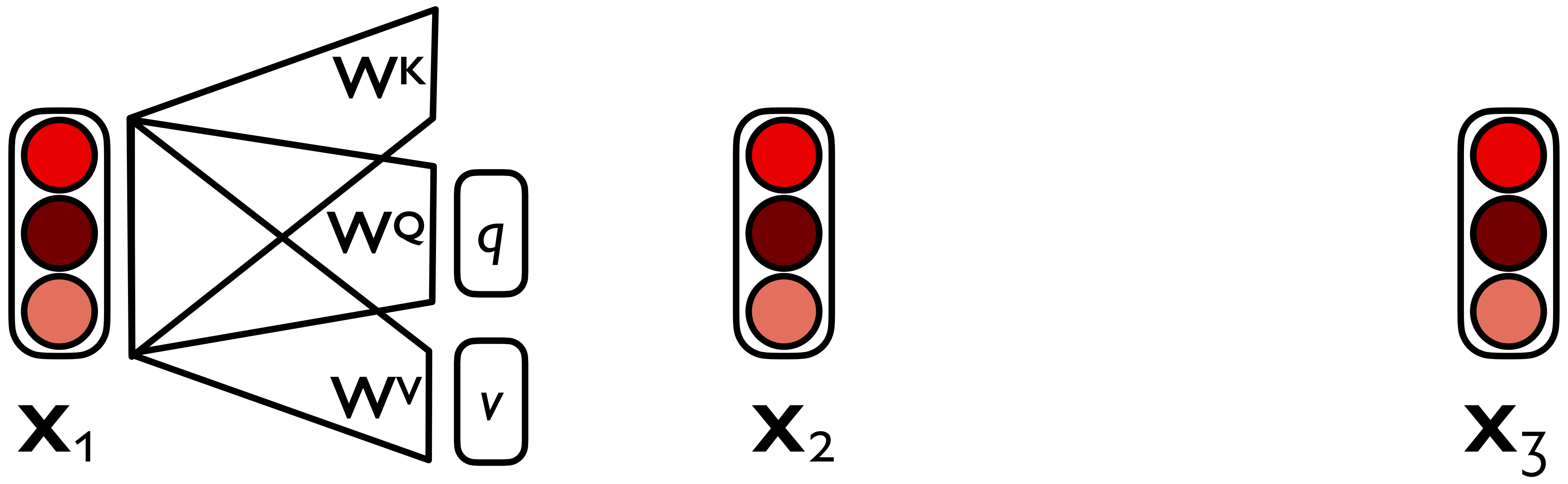
Project each $\mathbf{x}_i$ *into key, query, and value vectors*



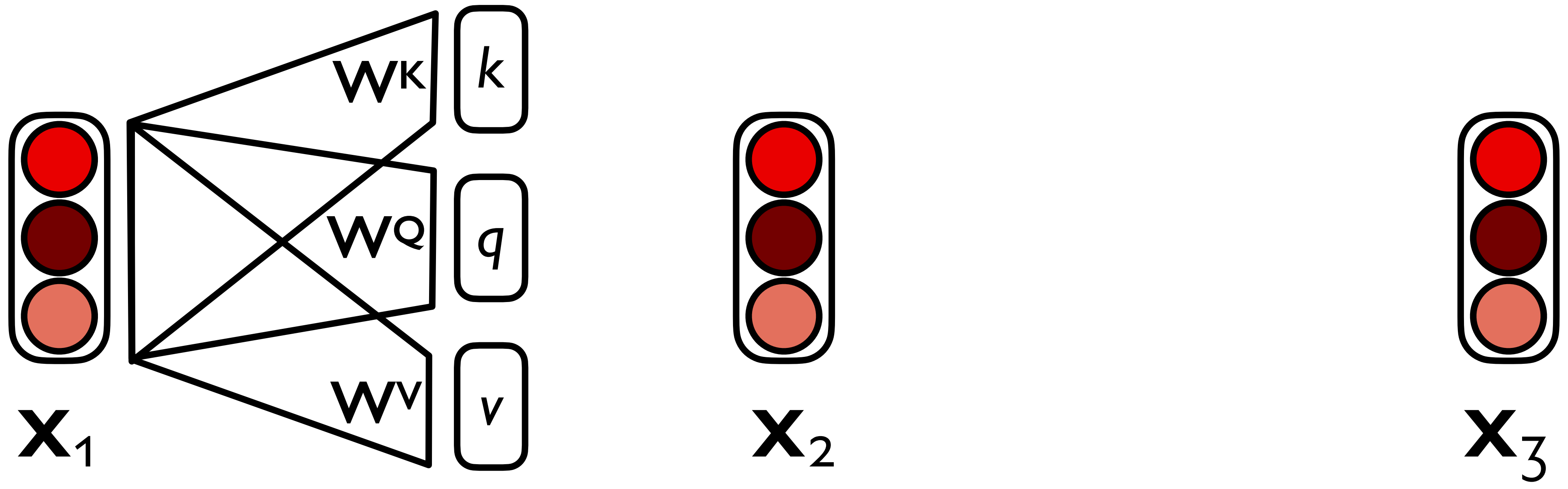
Project each $\mathbf{x}_i$ into key, query, and value vectors



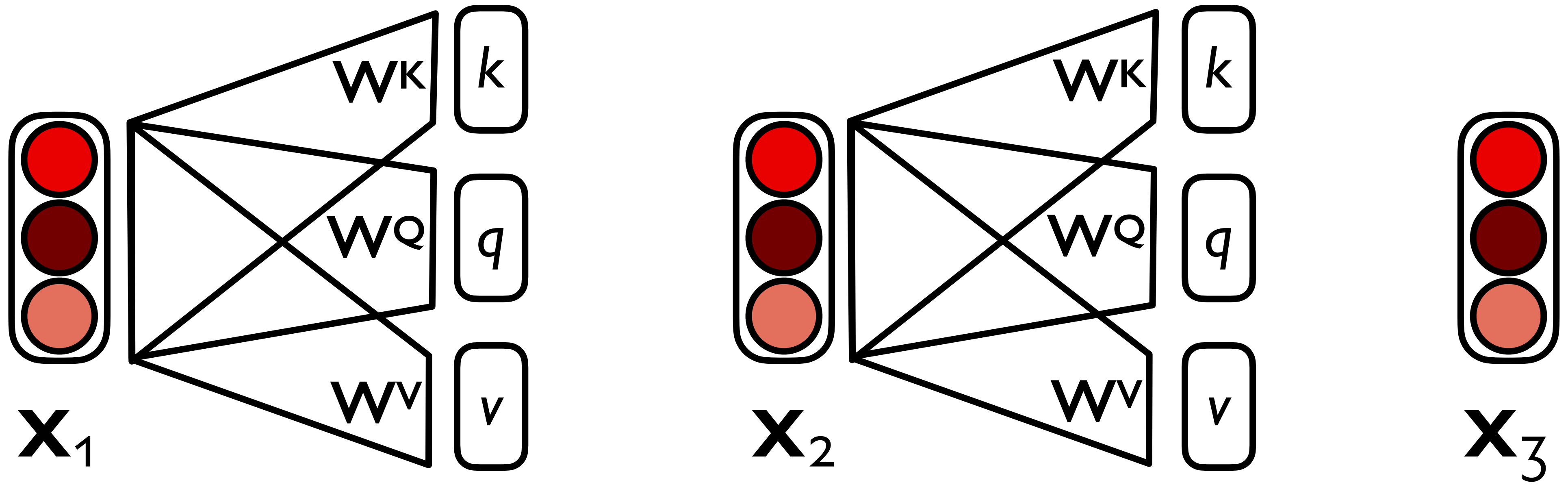
Project each $\mathbf{x}_i$ into key, query, and value vectors



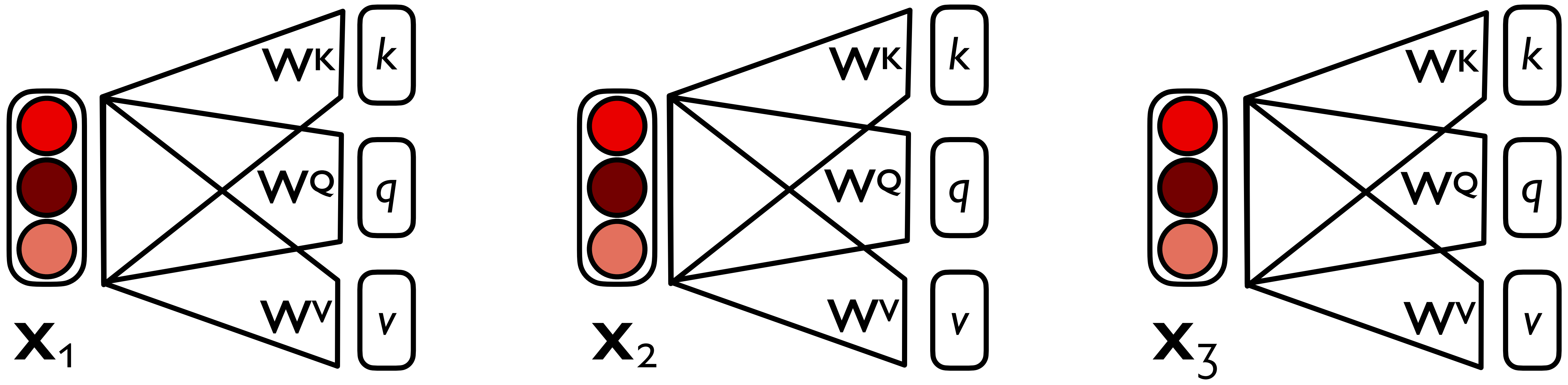
Project each $\mathbf{x}_i$ into key, query, and value vectors



Project each $\mathbf{x}_i$ into key, query, and value vectors

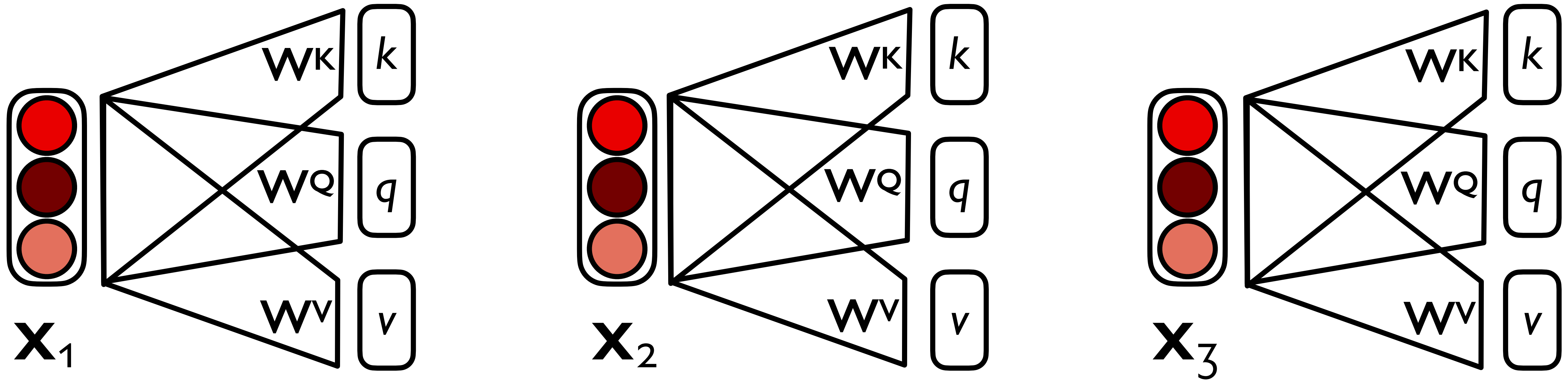


Project each $\mathbf{x}_i$ into key, query, and value vectors



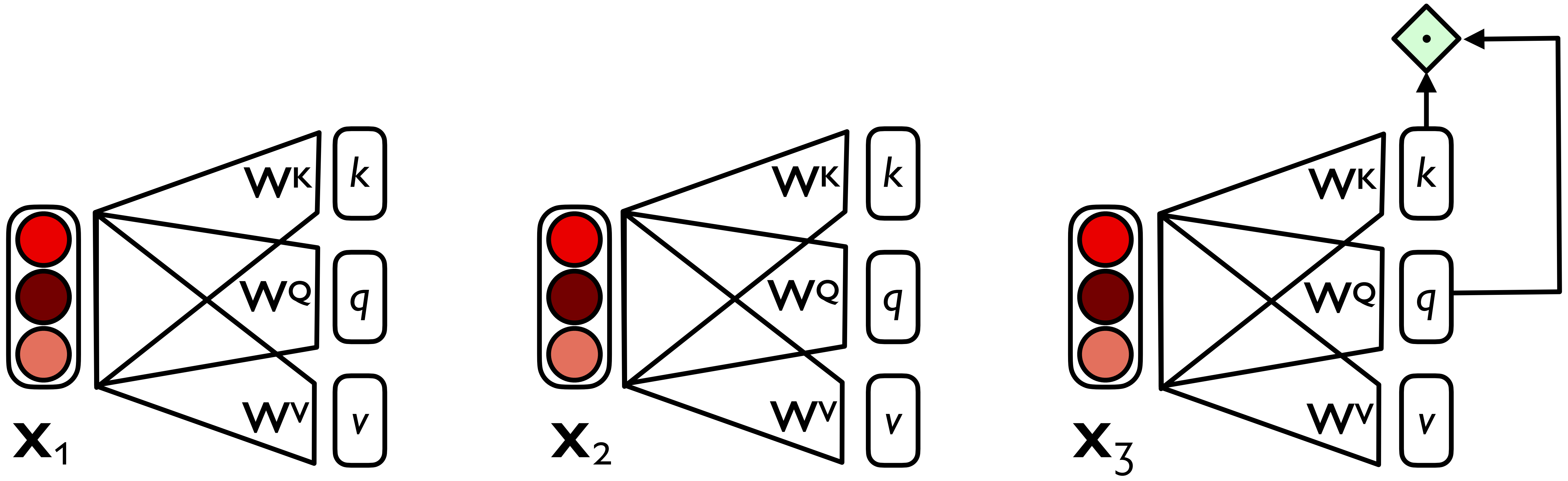
Compare $\mathbf{x}_3$*'s query with the keys for* $\mathbf{x}_1$*,* $\mathbf{x}_2$*, and* $\mathbf{x}_3$*.*

Project each $\mathbf{x}_i$ *into key, query, and value vectors*



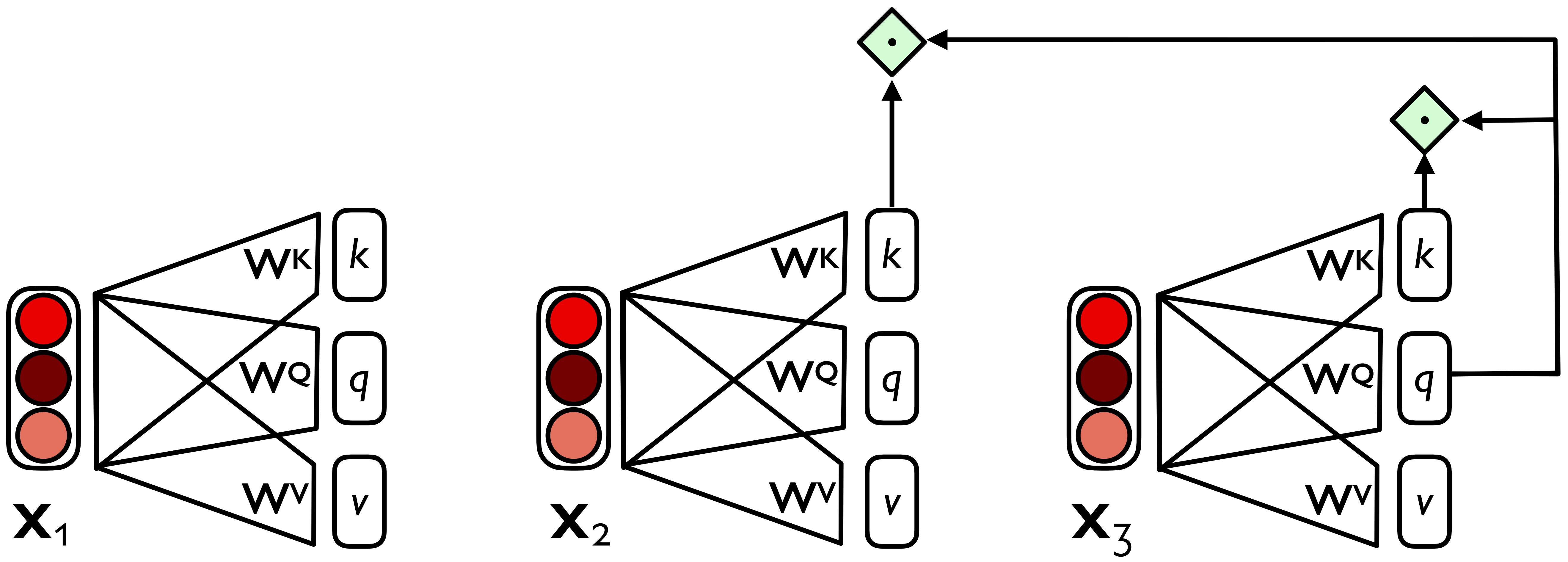
Compare $\mathbf{x}_3$*'s query with the keys for* $\mathbf{x}_1$, $\mathbf{x}_2$, *and* $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ *into key, query, and value vectors*



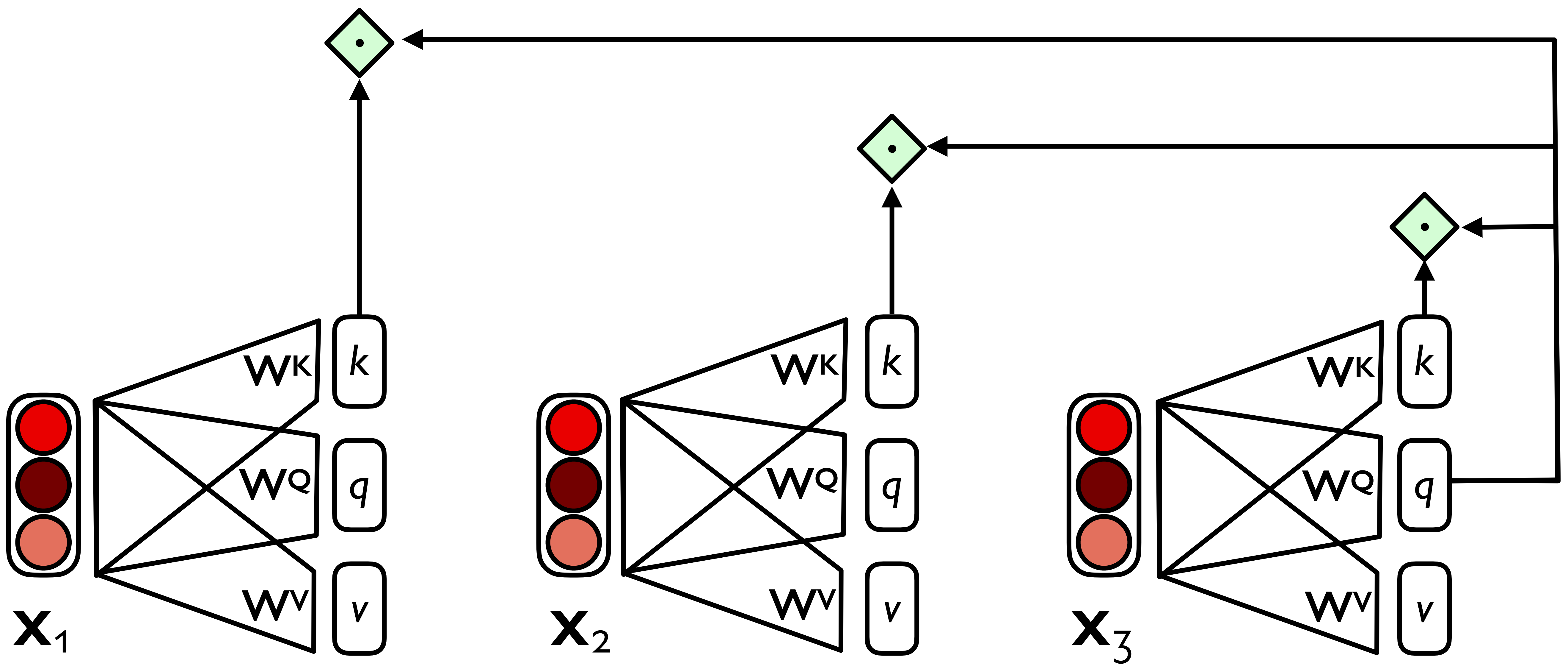
Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ into key, query, and value vectors



Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

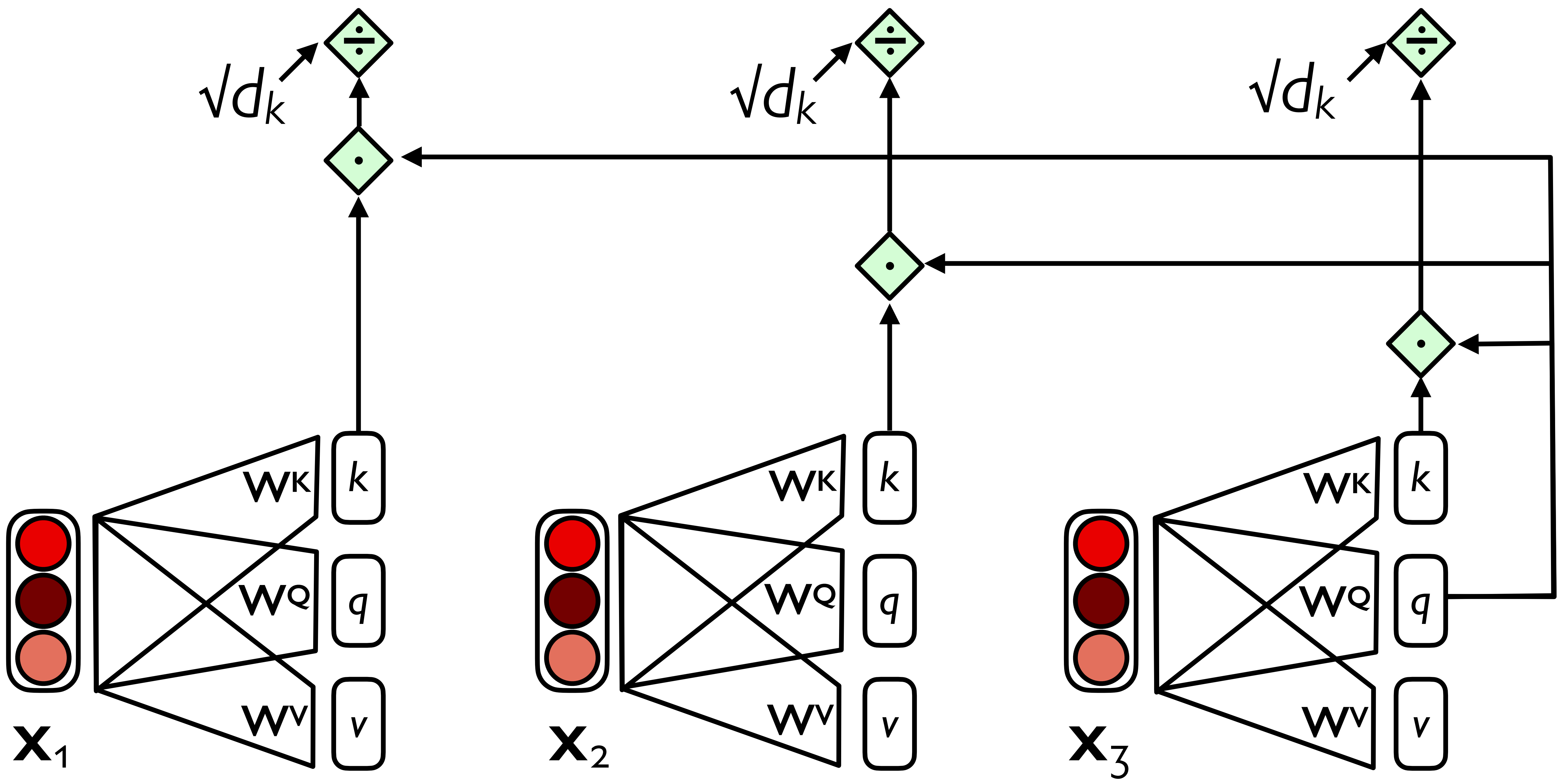
Project each $\mathbf{x}_i$ into key, query, and value vectors



Divide score by $\sqrt{d_k}$

Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ into key, query, and value vectors

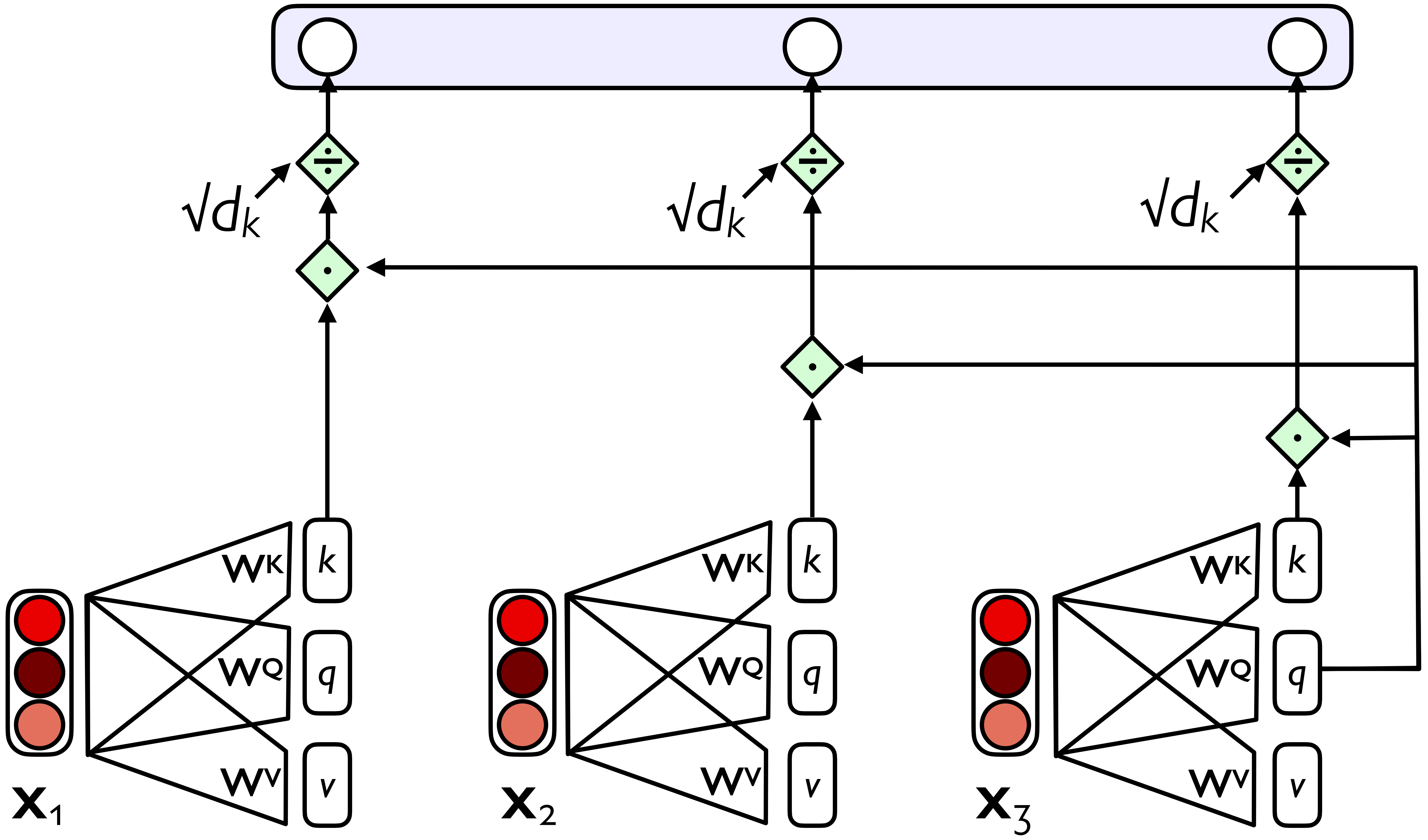


Turn into a_{ij} weights via softmax

Divide score by $\sqrt{d_k}$

Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ into key, query, and value vectors



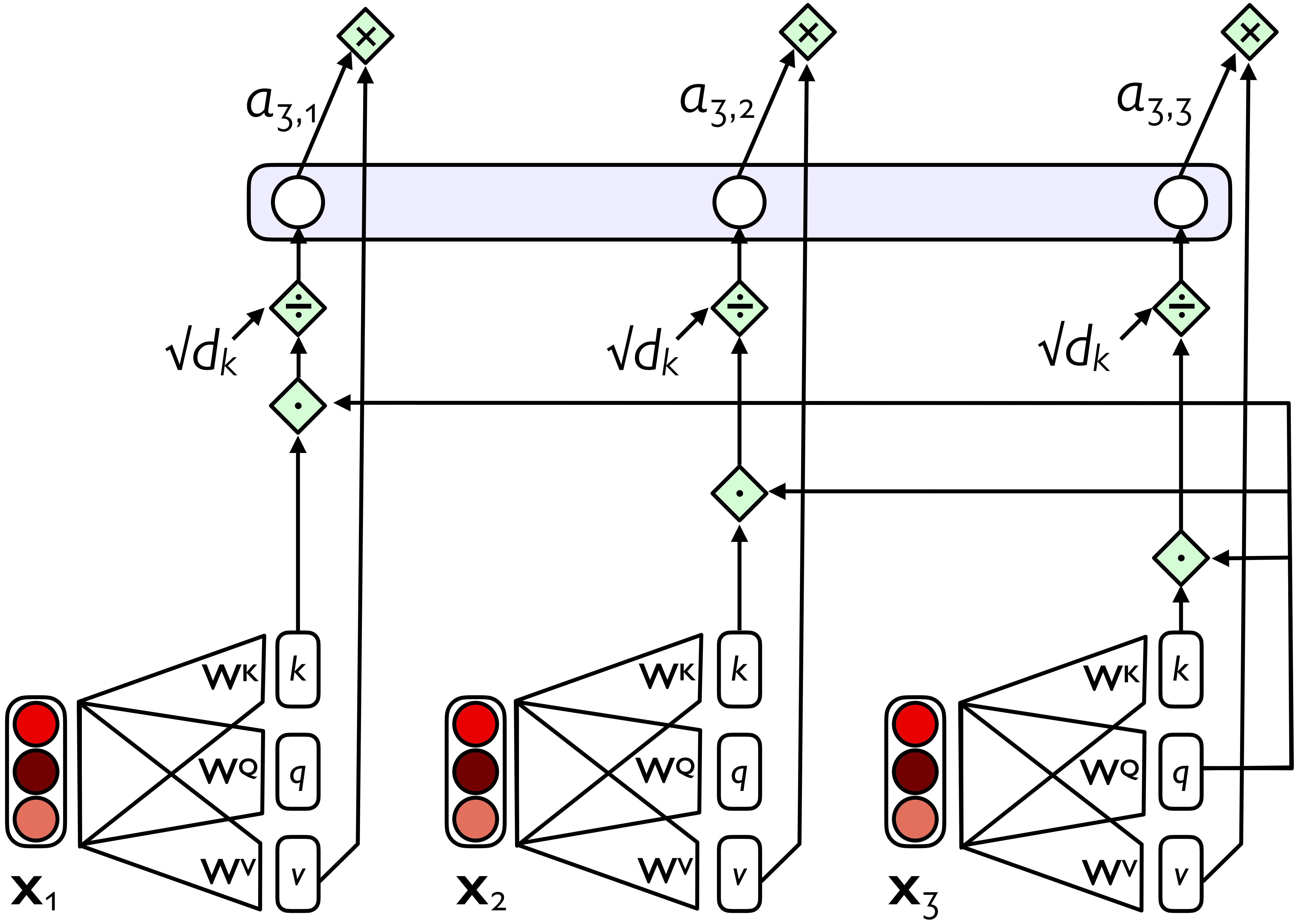
Weigh each value vector

Turn into a_{ij} weights via softmax

Divide score by $\sqrt{d_k}$

*Compare **x**3's query with the keys for **x**1, **x**2, and **x**3.*

*Project each **x**i into key, query, and value vectors*



Sum the weighted value vectors

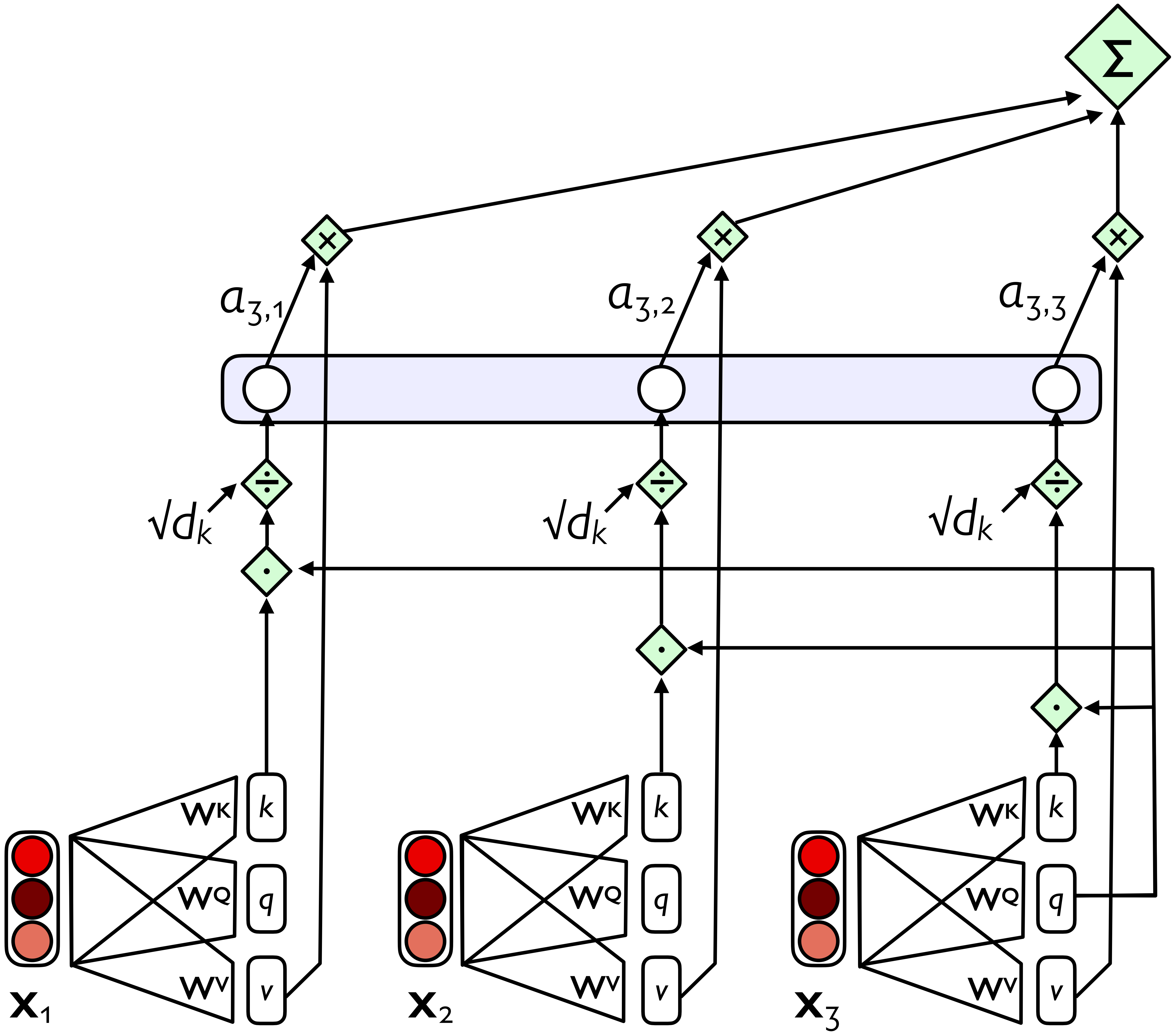
Weigh each value vector

Turn into a_{ij} weights via softmax

Divide score by $\sqrt{d_k}$

Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ into key, query, and value vectors



Output of self-attention

Sum the weighted value vectors

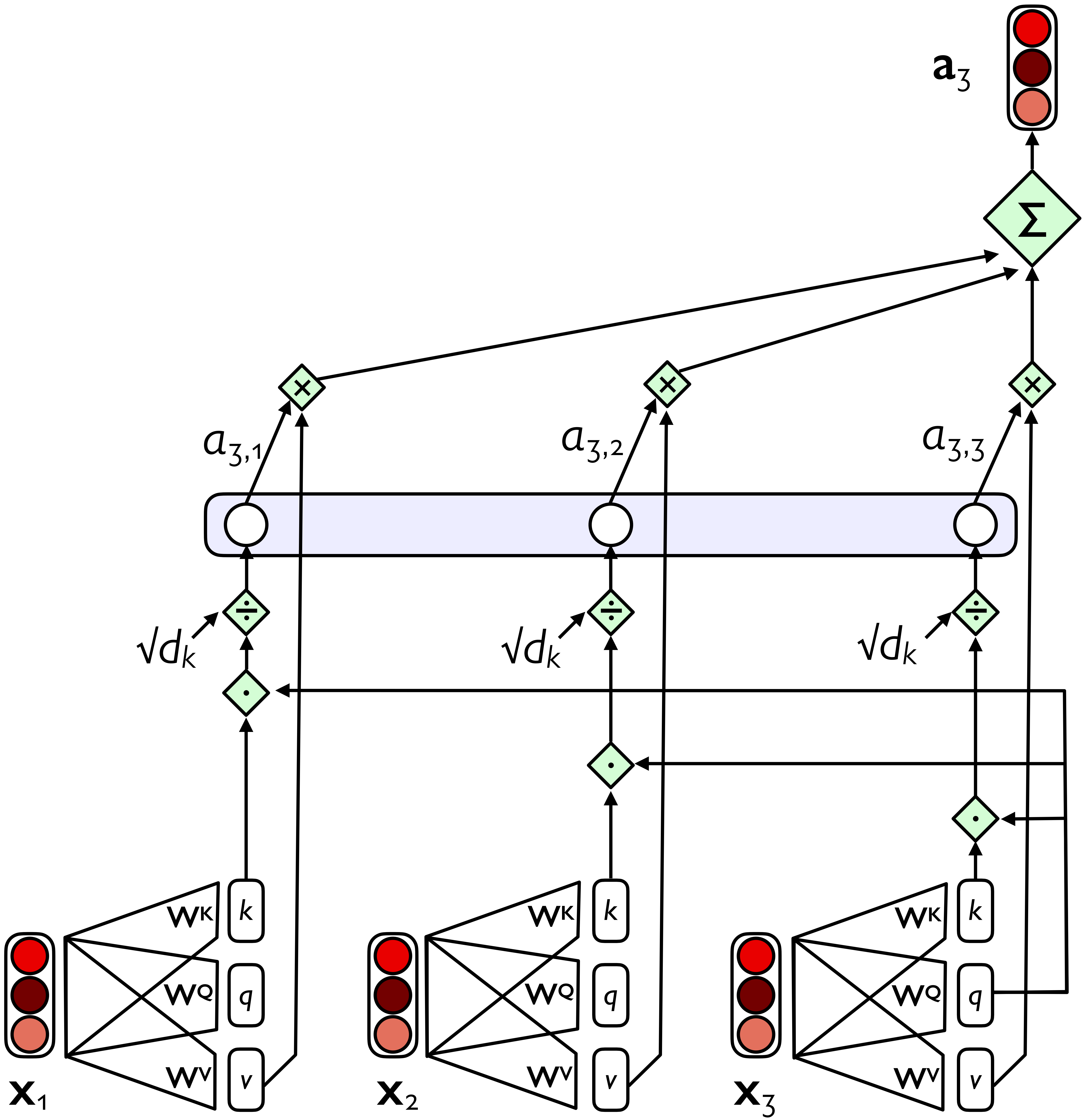
Weigh each value vector

Turn into a_{ij} weights via softmax

Divide score by $\sqrt{d_k}$

Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ into key, query, and value vectors



vicki
@vboykis

They don’t tell you this in the paper (well they do but you have to read it like 15 times)

6:20 PM · Feb 22, 2023 · **88.3K** Views

In practice, instead of one attention head, we'll have lots of them: ***multi-head attention***.

Why? Each head might be attending to the context for different purposes – different linguistic relationships or patterns in the context

Summary

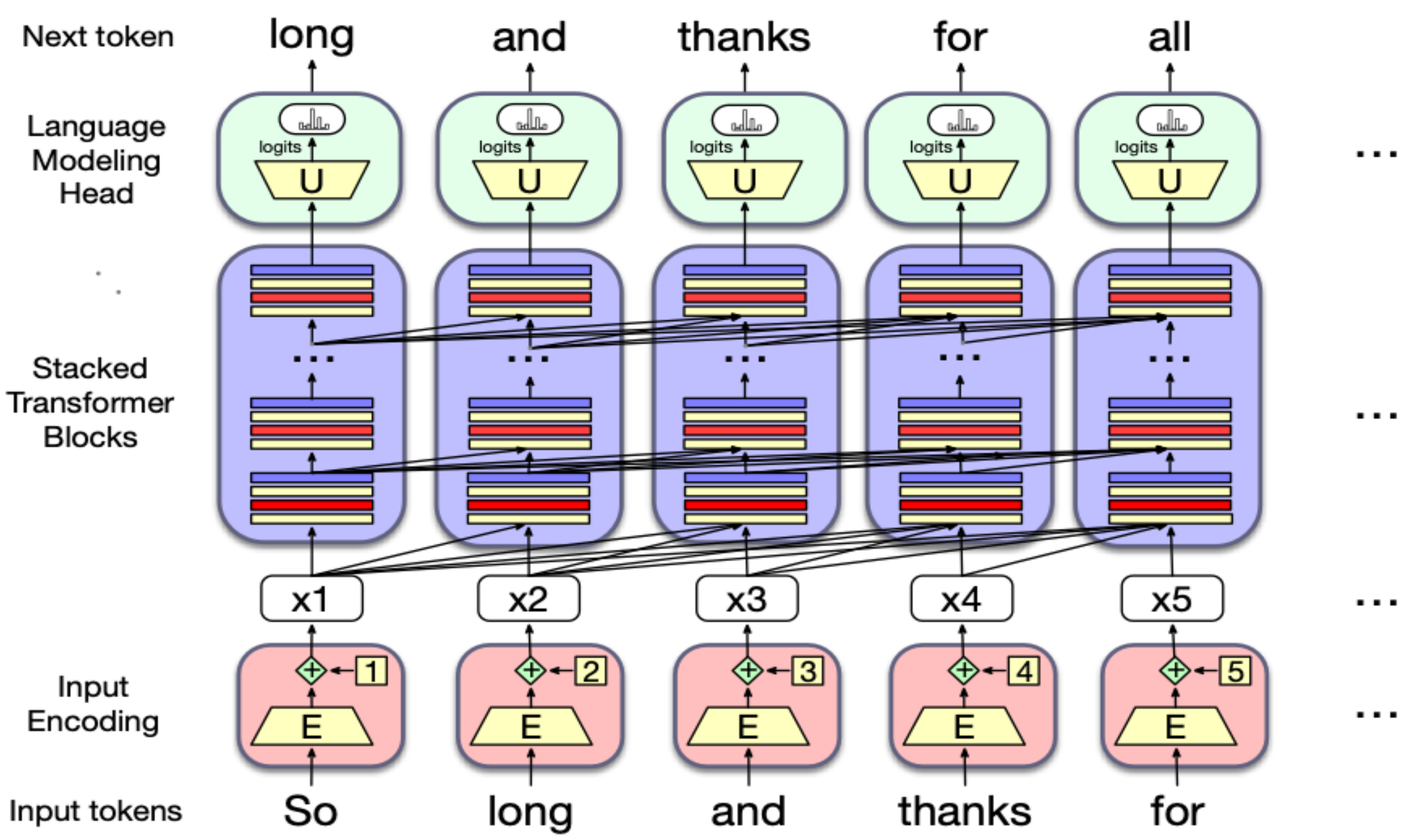
Attention is a method for enriching the representation of a token by incorporating contextual information

The result: the embedding for each word will be different in different contexts!

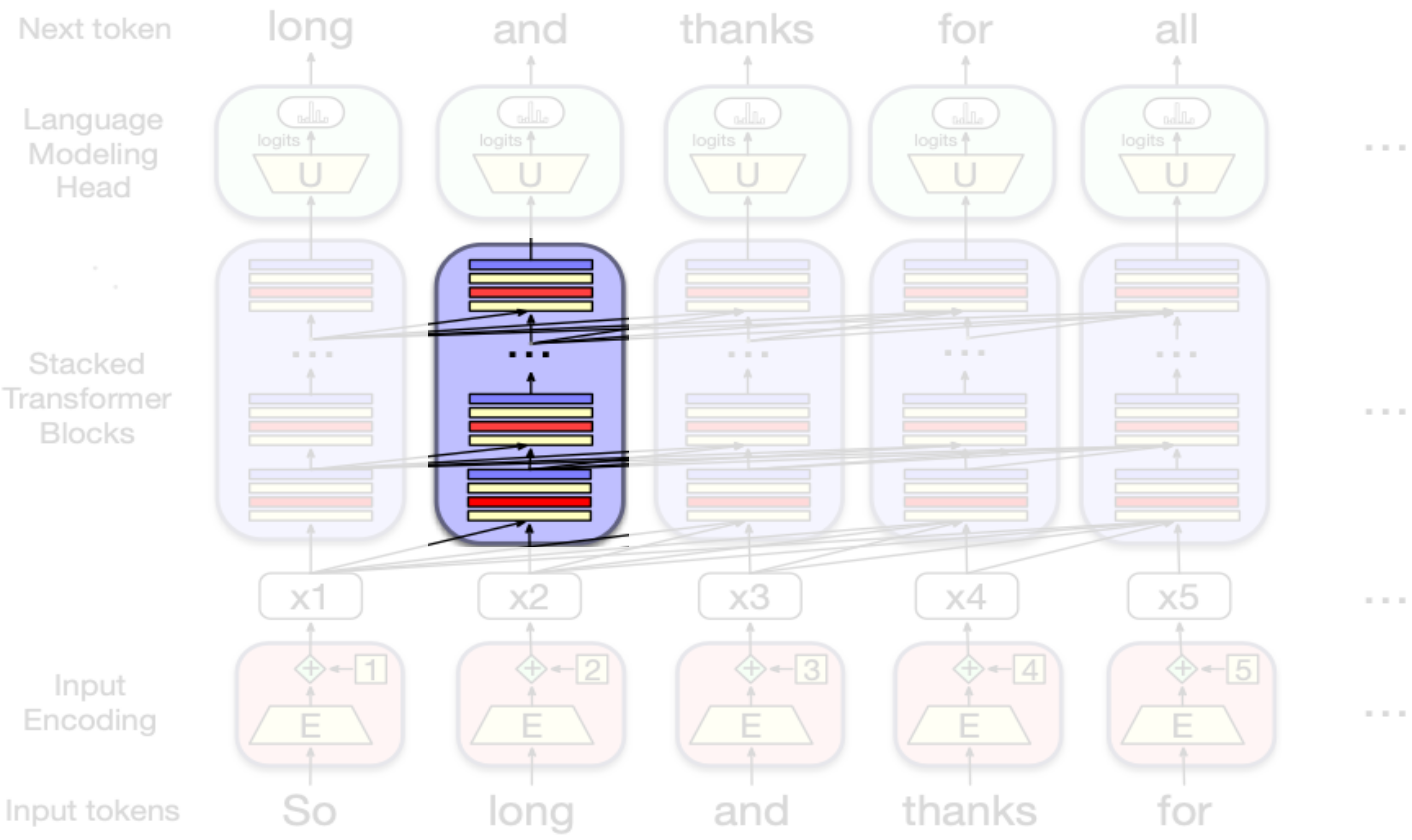
Contextual embeddings: a representation of word meaning in its context.

We'll see next that attention can also be viewed as a way to move information from one token to another.

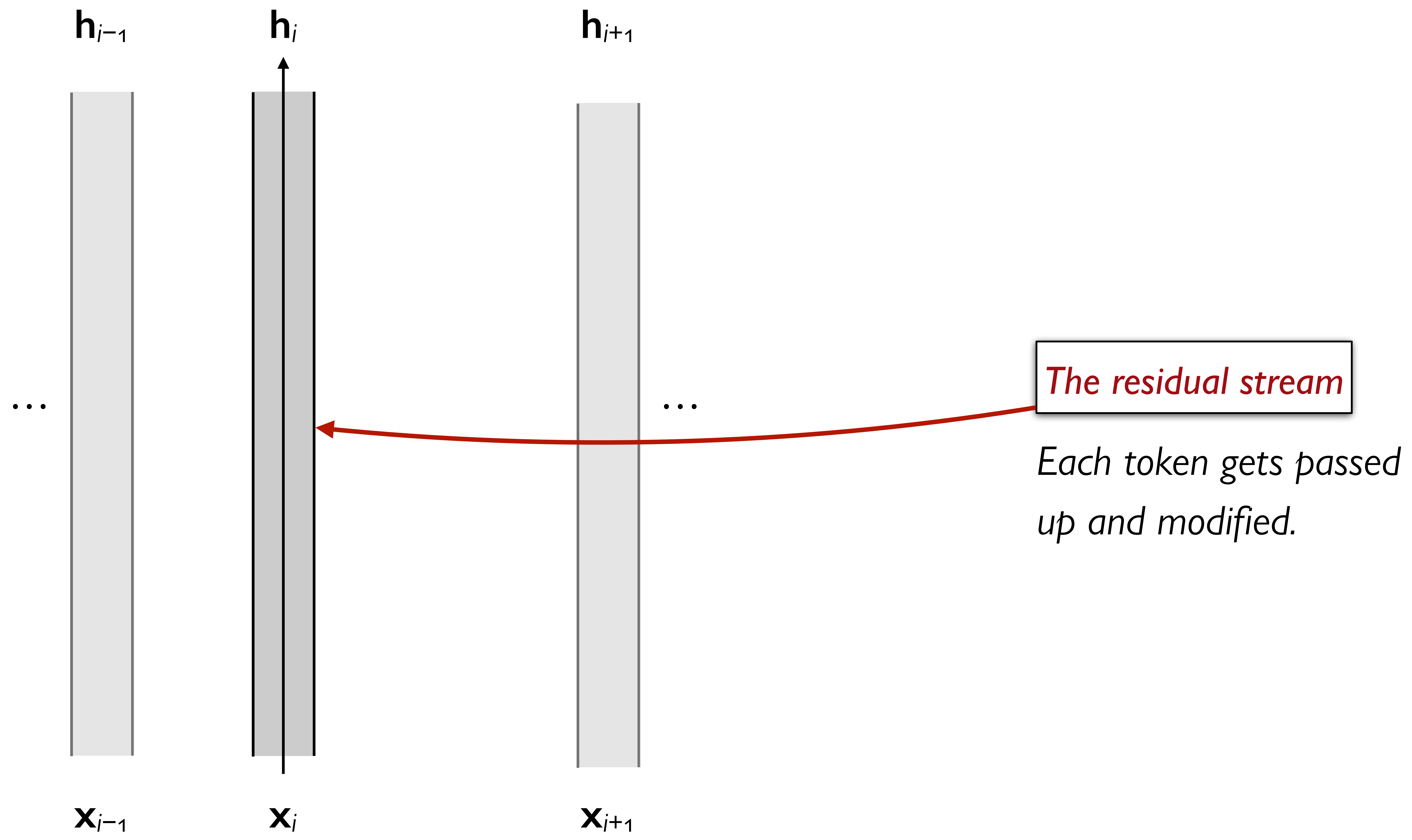
The transformer block



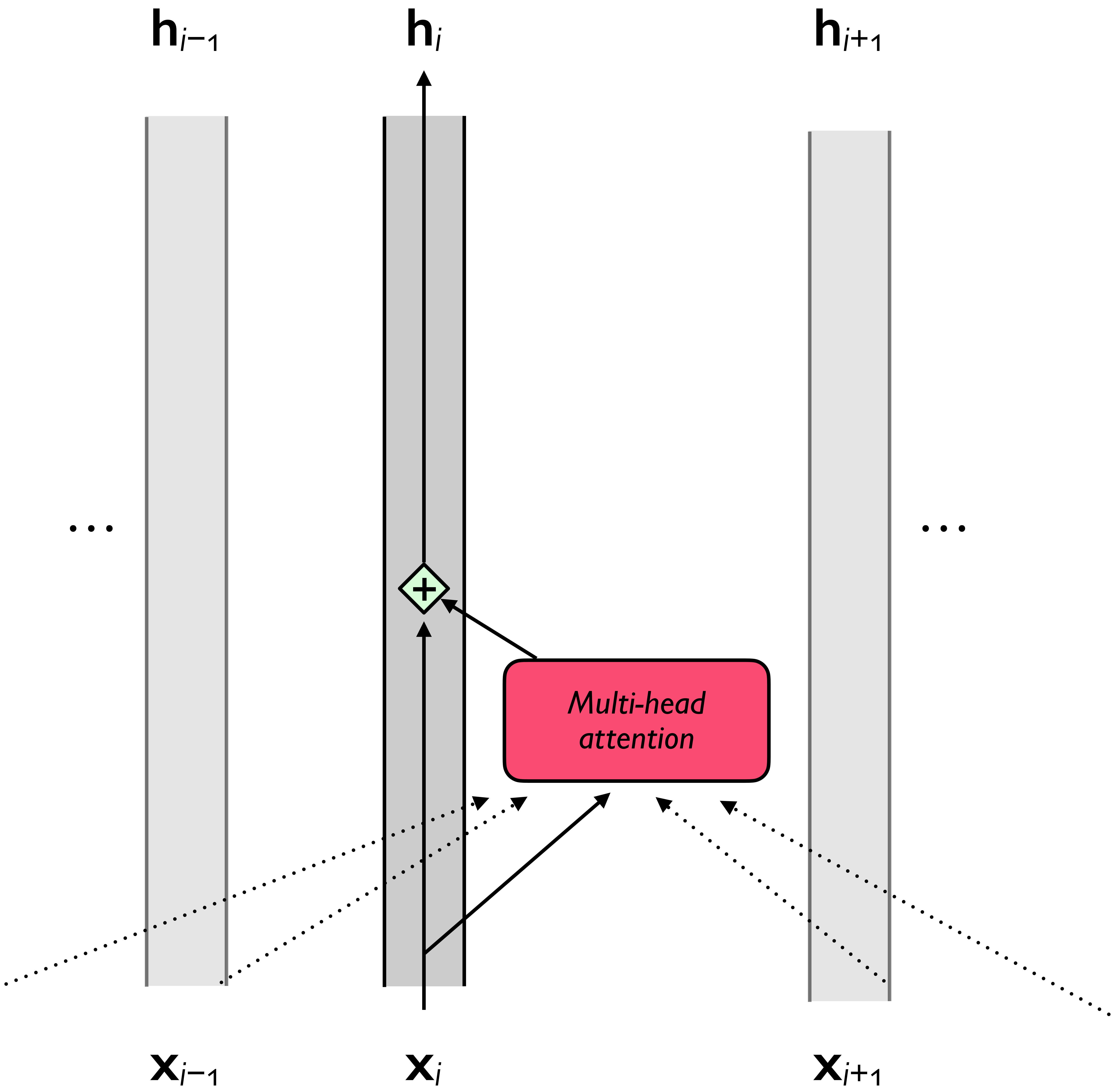
Next token
long
and
thanks
for
all
Language Modeling Head
logits
U
Stacked Transformer Blocks
x1
x2
x3
x4
x5
Input Encoding
E
1
2
3
4
5
Input tokens
So
long
and
thanks
for



Next token
long
and
thanks
for
all
Language Modeling Head
logits
U
Stacked Transformer Blocks
x1
x2
x3
x4
x5
Input Encoding
E
1
2
3
4
5
Input tokens
So
long
and
thanks
for

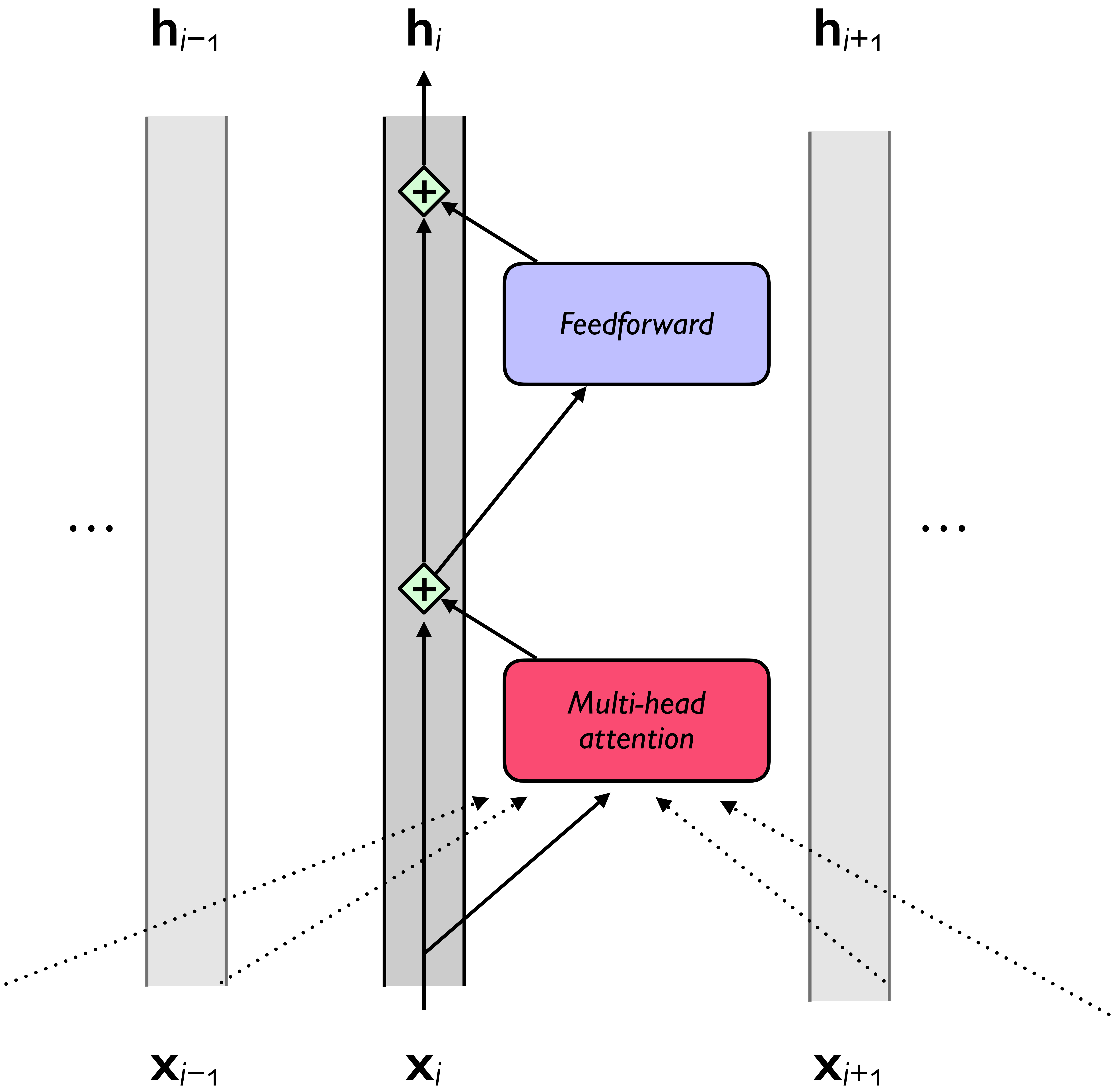


$\mathbf{h}_{i-1}$
$\mathbf{h}_i$
$\mathbf{h}_{i+1}$
…
…
The residual stream
Each token gets passed up and modified.
$\mathbf{x}_{i-1}$
$\mathbf{x}_i$
$\mathbf{x}_{i+1}$



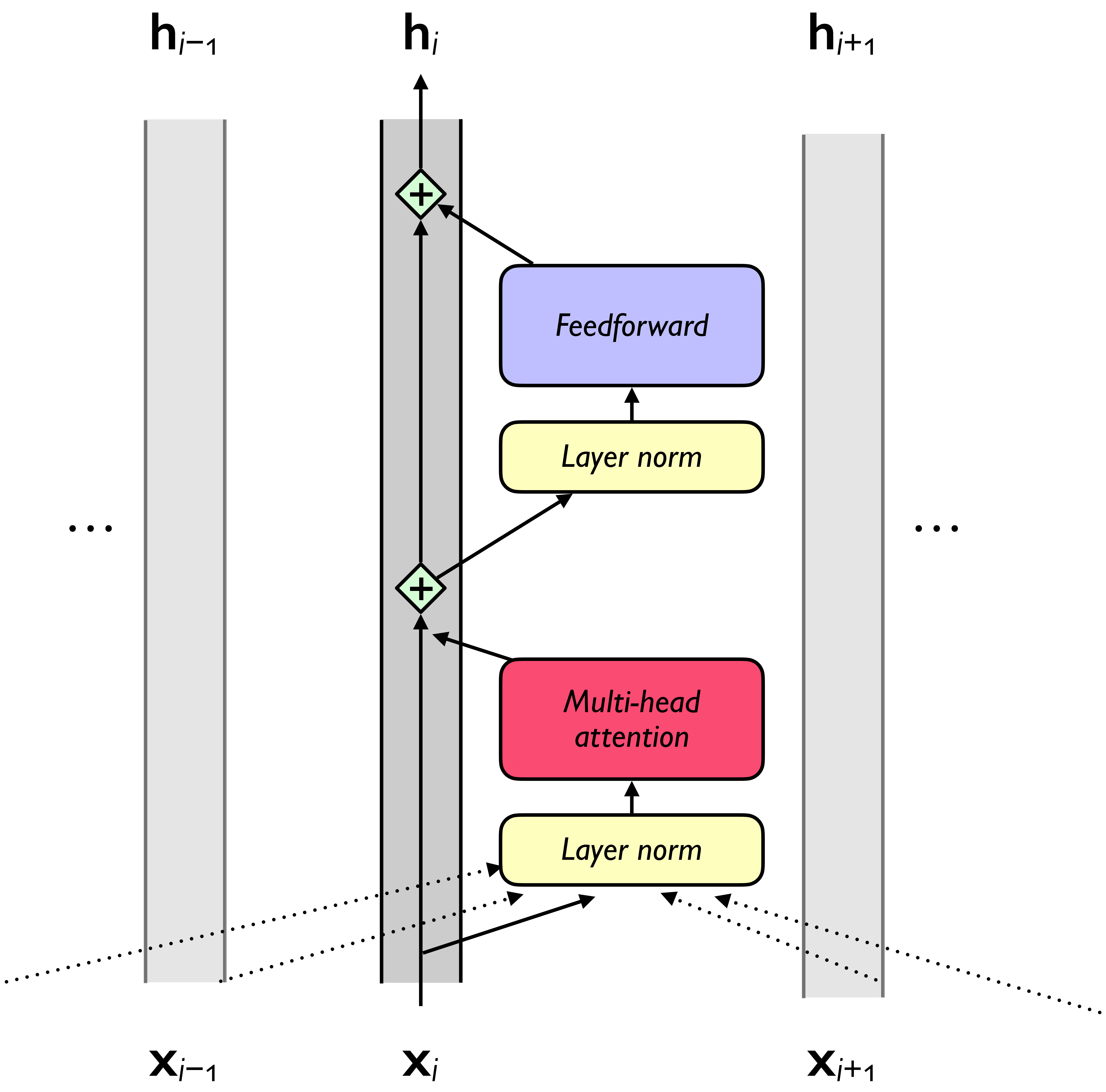
$\mathbf{h}_{i-1}$
$\mathbf{h}_i$
$\mathbf{h}_{i+1}$
...
...
Multi-head attention
$\mathbf{x}_{i-1}$
$\mathbf{x}_i$
$\mathbf{x}_{i+1}$

Analogy time!



We'll need nonlinearities, so add a feedforward layer

$$\text{FFN}(\mathbf{x}_i) = \text{ReLU}(\mathbf{x}_i\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$$



Layer norm

The vector $\mathbf{x}_i$ is normalized twice

Layer norm is a variation of the z-score from statistics, applied to a single vector in a hidden layer:

mean

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$

standard deviation

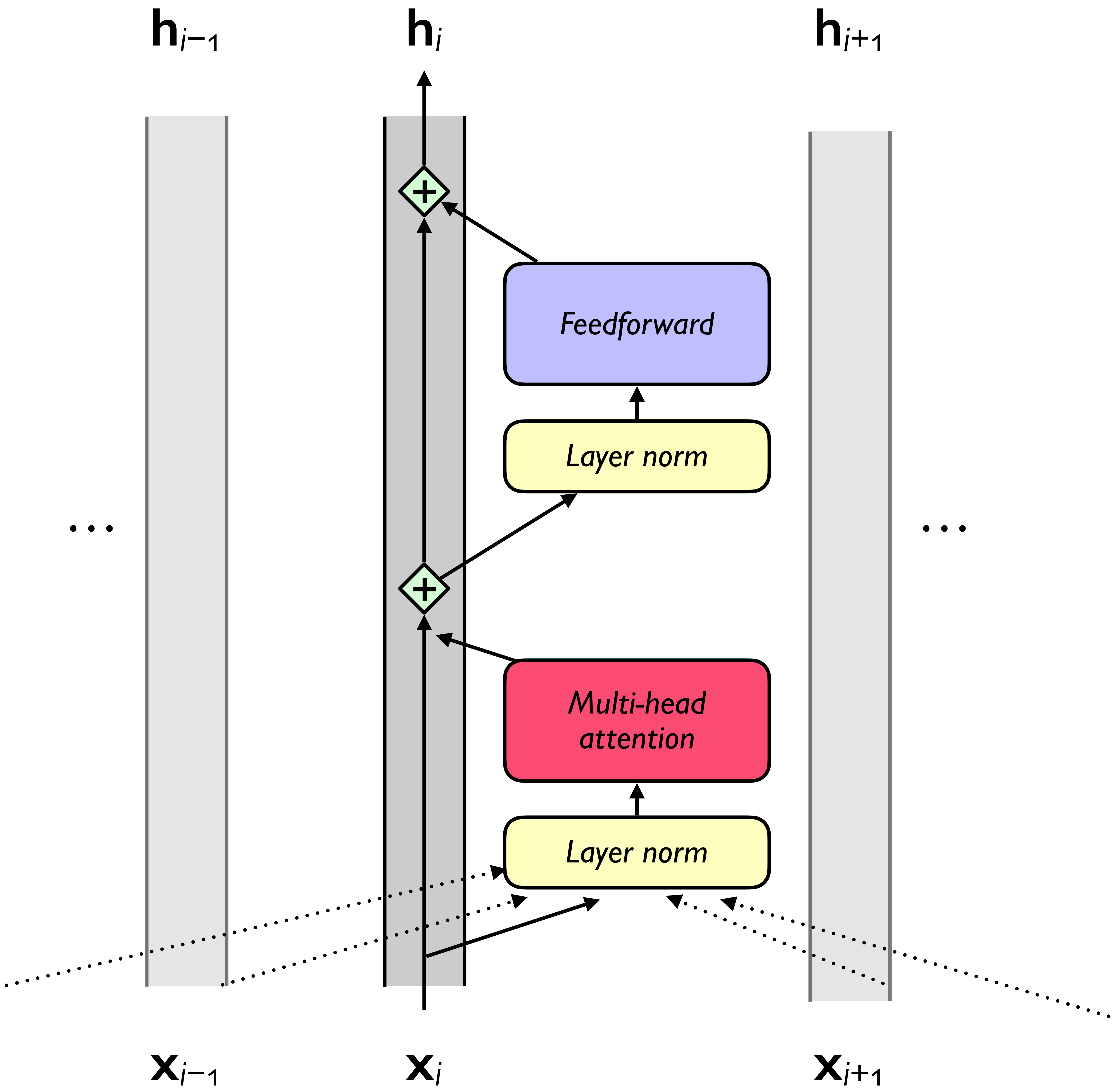
$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i-\mu)^2}$$

normalized

$$\hat{\mathbf{x}} = \frac{(\mathbf{x}-\mu)}{\sigma}$$

normalized in a tunable way

$$\text{LayerNorm}(\mathbf{x}) = \gamma\frac{(\mathbf{x}-\mu)}{\sigma}+\beta$$



$$\mathbf{h}_i = \mathbf{t}_i^5 + \mathbf{t}_i^3$$

$$\mathbf{t}_i^5 = \mathrm{FFN}(\mathbf{t}_i^4)$$

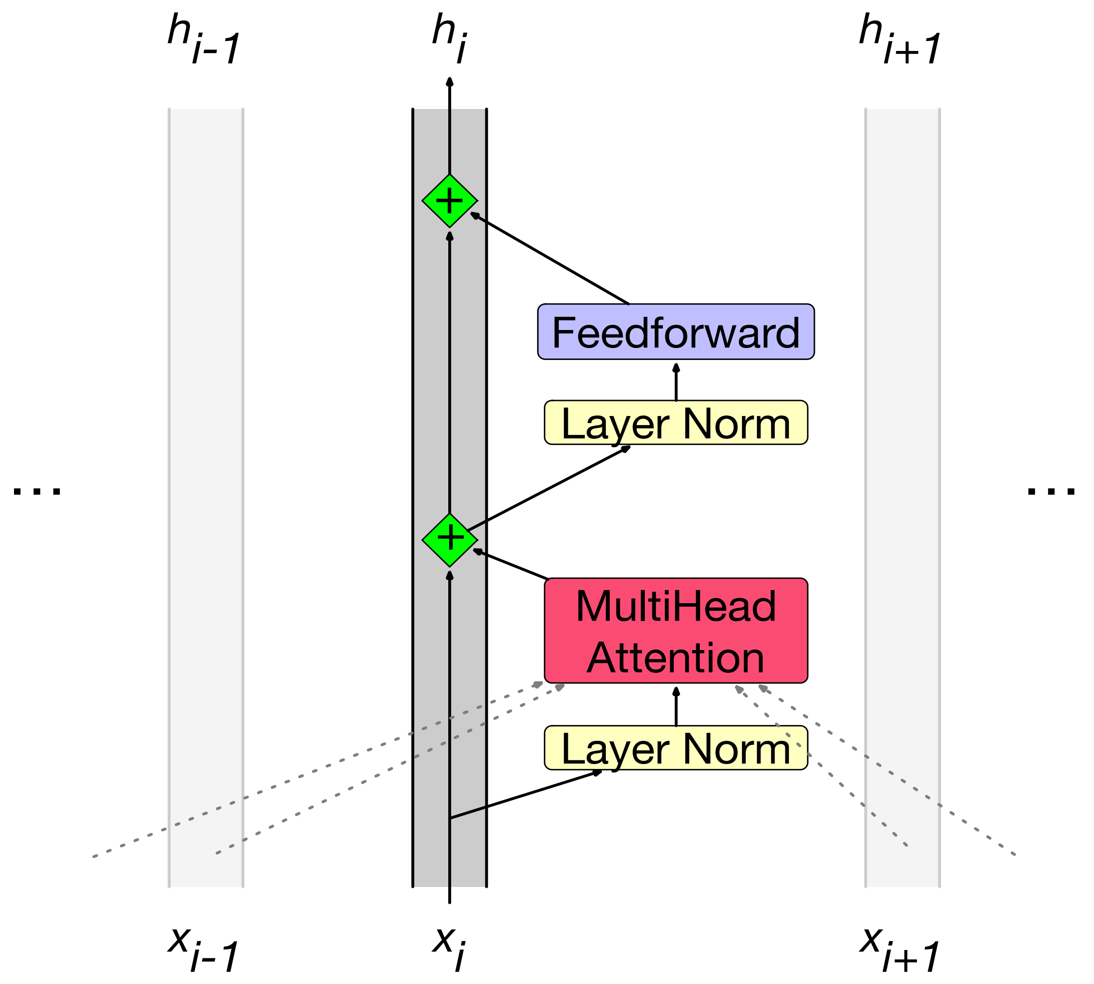
$$\mathbf{t}_i^4 = \mathrm{LayerNorm}(\mathbf{t}_i^3)$$

$$\mathbf{t}_i^3 = \mathbf{t}_i^2 + \mathbf{x}_i$$

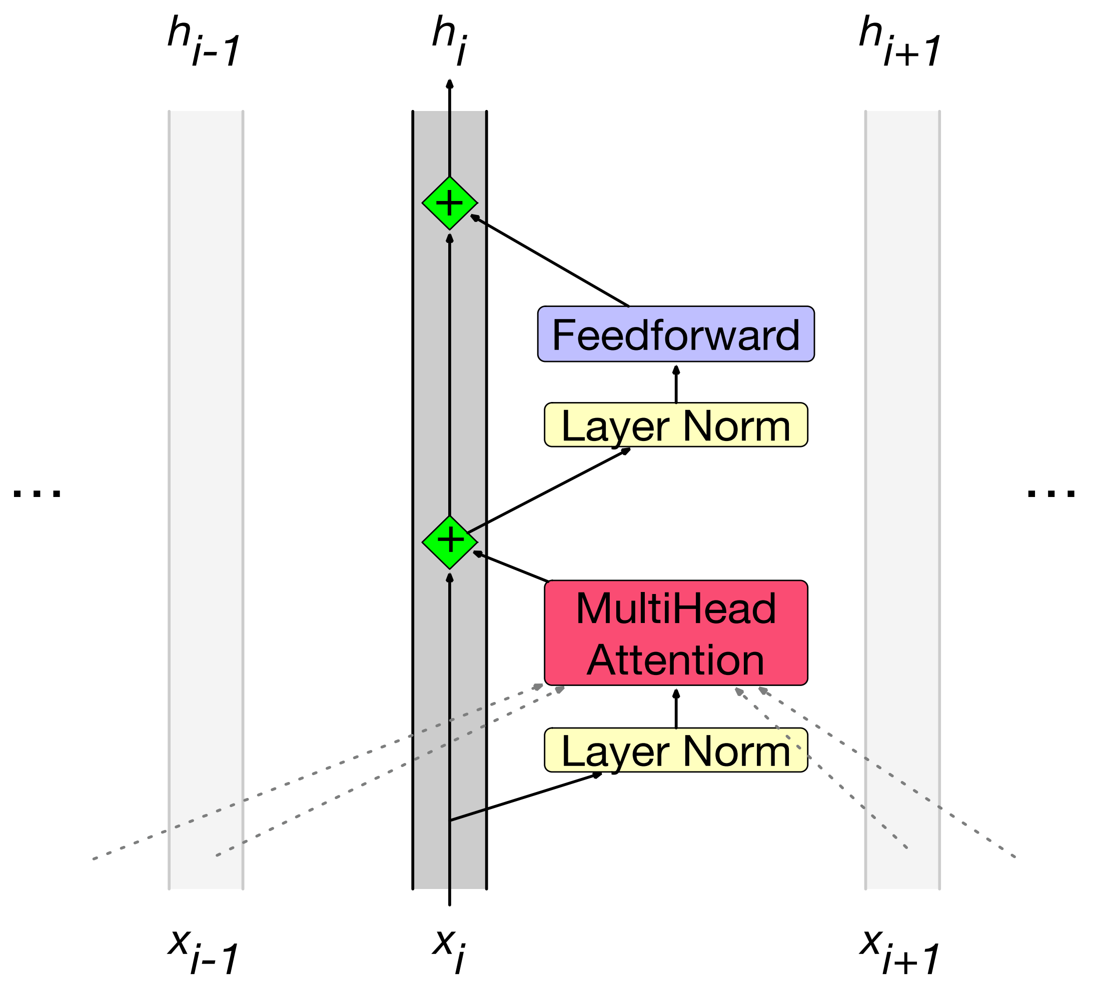
$$\mathbf{t}_i^2 = \mathrm{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{t}_1^1, \ldots, \mathbf{t}_N^1])$$

$$\mathbf{t}_i^1 = \mathrm{LayerNorm}(\mathbf{x}_i)$$

Block 2



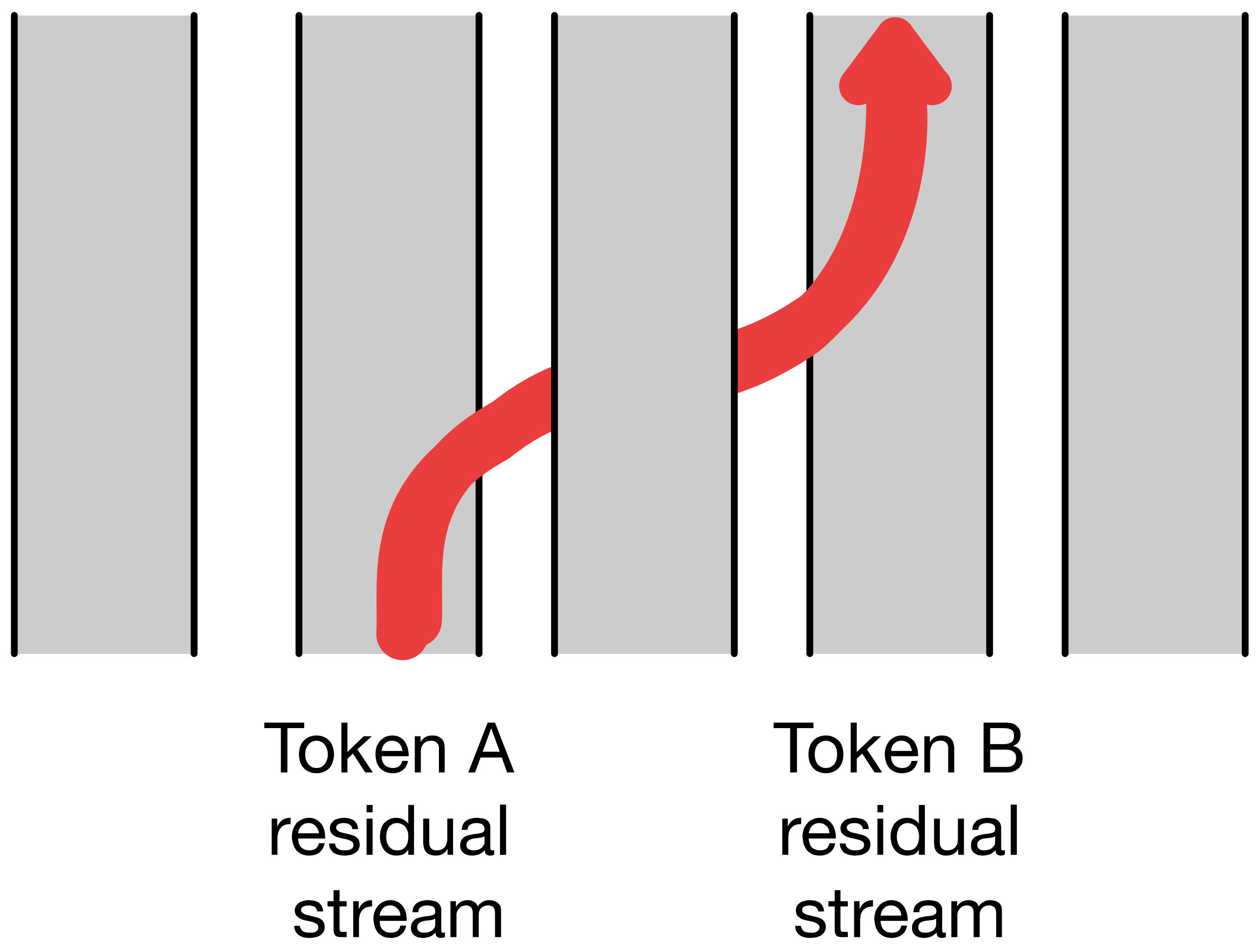
Block 1



A transformer is a stack of these blocks – so all the vectors are of the same dimensionality d

Notice that all parts of the transformer block apply to 1 residual stream (1 token) – except attention, which takes information from other tokens.

Elhage et al. (2021) show that we can view attention heads as literally moving information from the residual stream of a neighboring token into the current stream:



Acknowledgments

The lecture incorporates material from:

Jurafsky & Martin, *Speech and Language Processing*, 3rd ed. draft