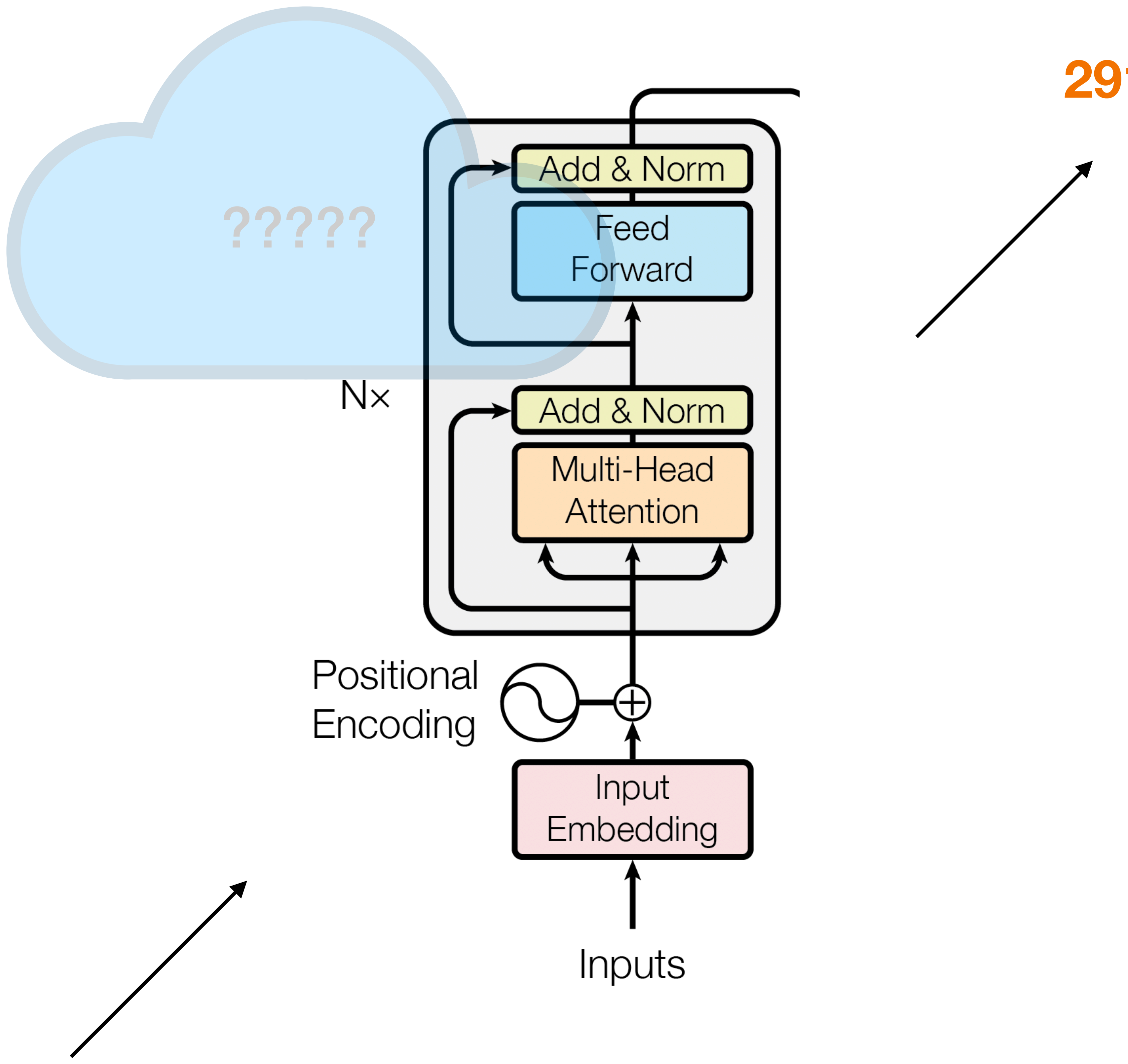
TECHNION
Israel Institute
of Technology

BIU NLP

Thinking Like Transformers

2917427



13945 + 2903482

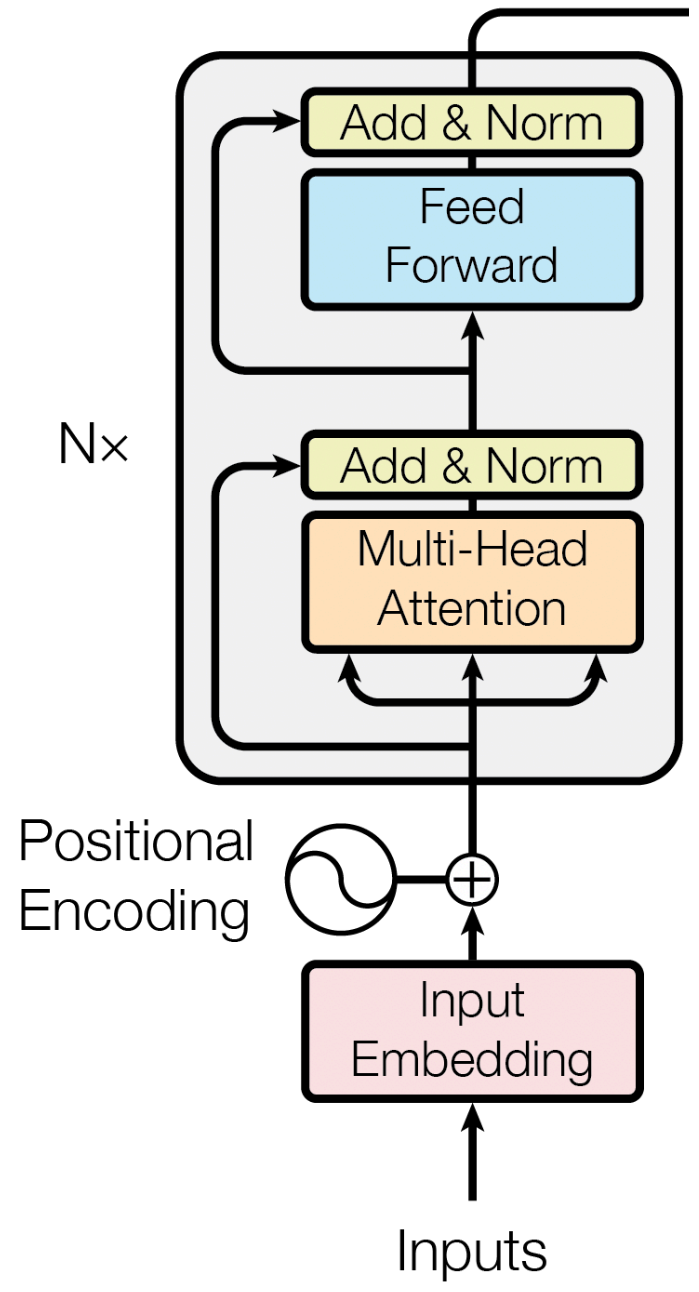
Gail Weiss, Yoav Goldberg, Eran Yahav

Transformers

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

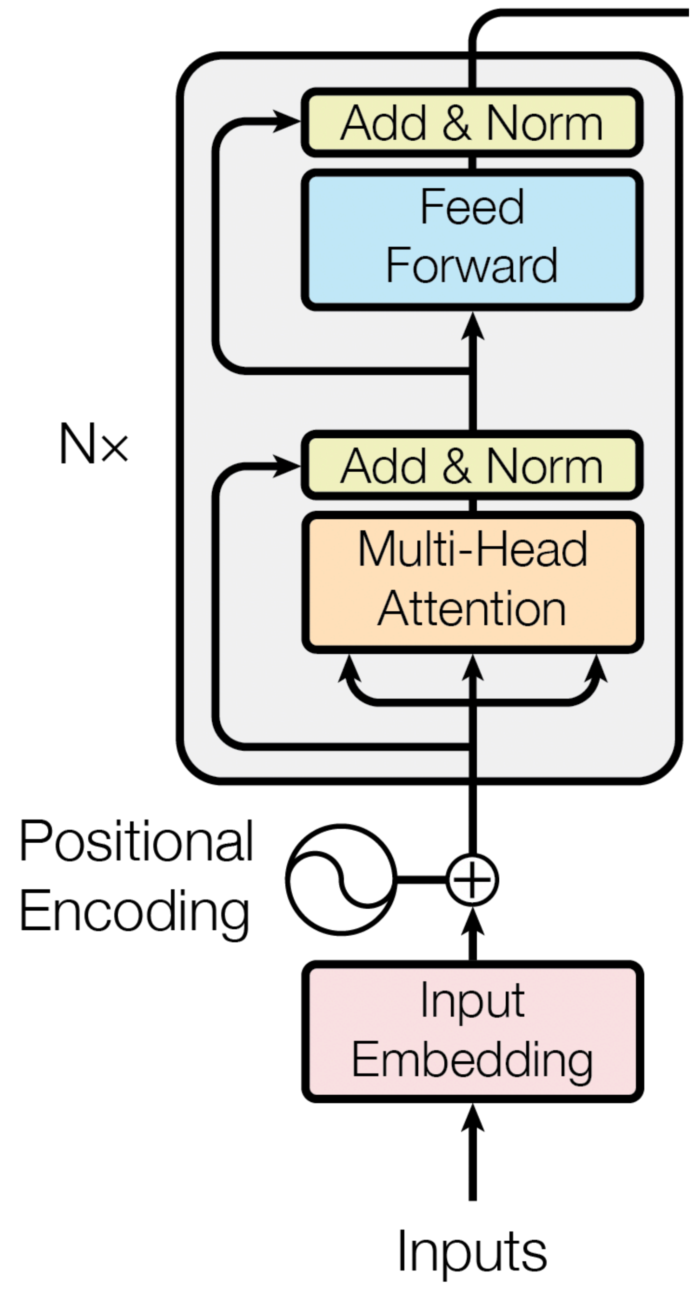
Highly parallel architecture
Strong performance



Transformers

Attention Is All You Need

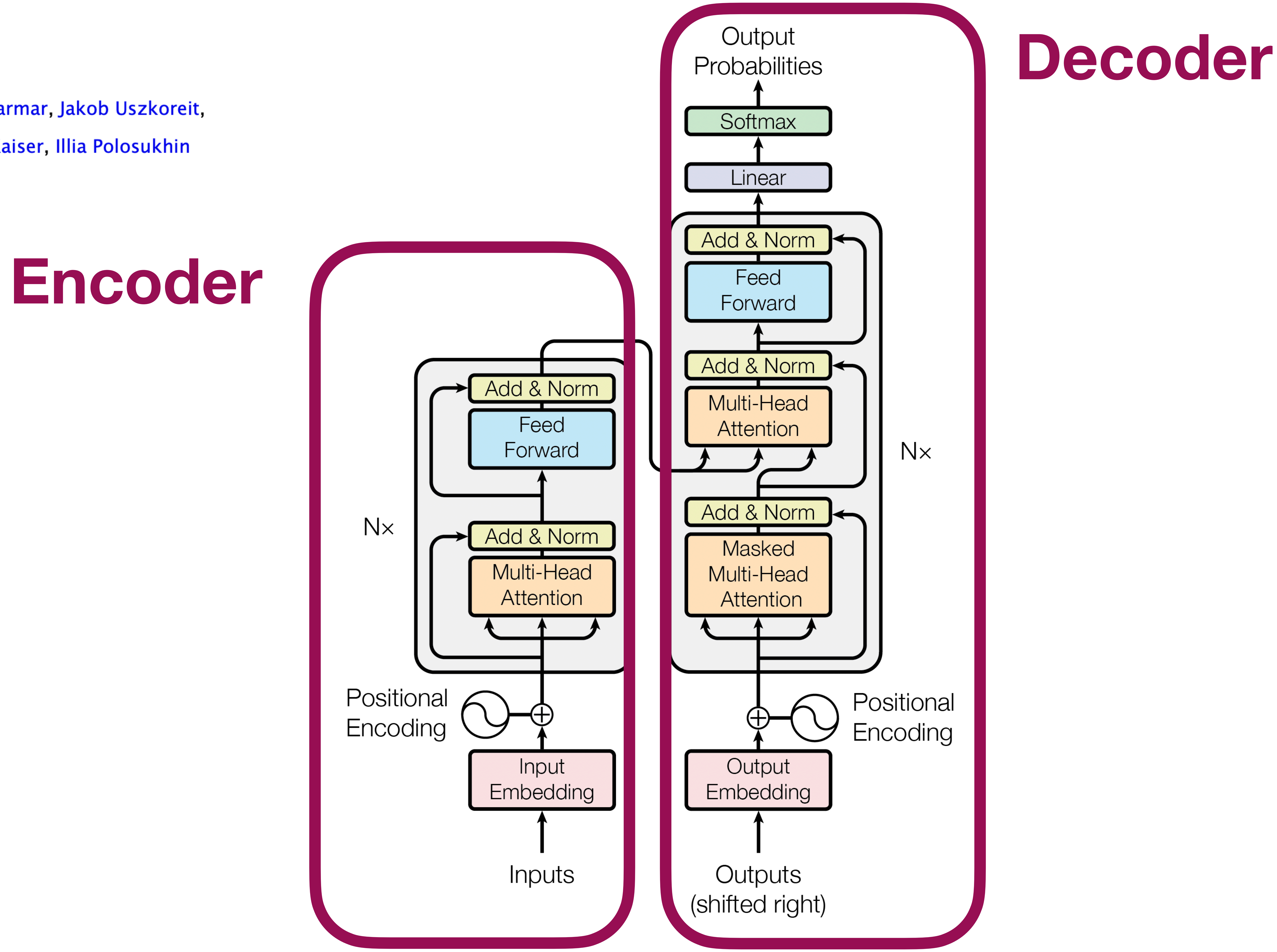
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin



Transformers

Attention Is All You Need

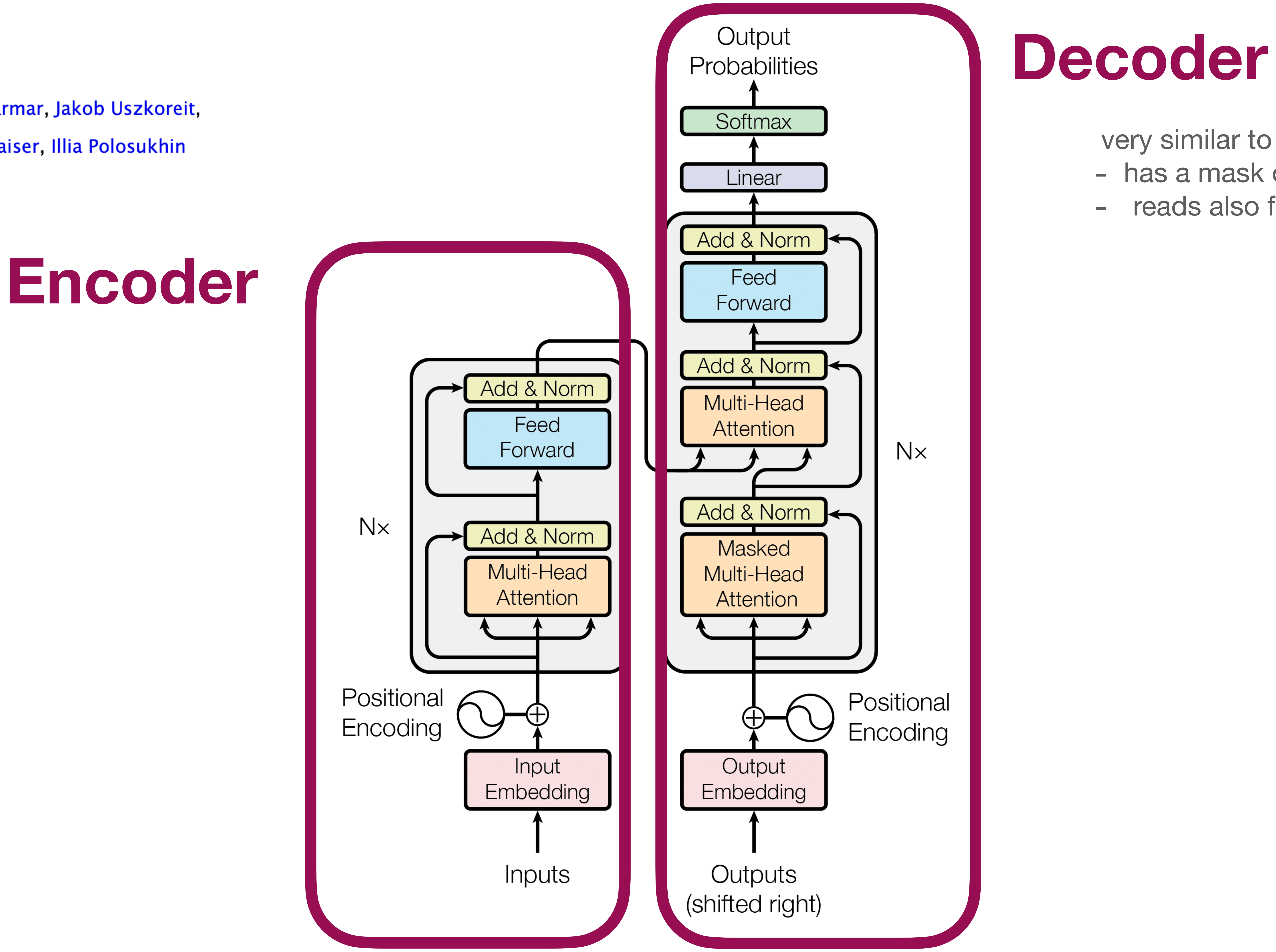
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin



Transformers

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin



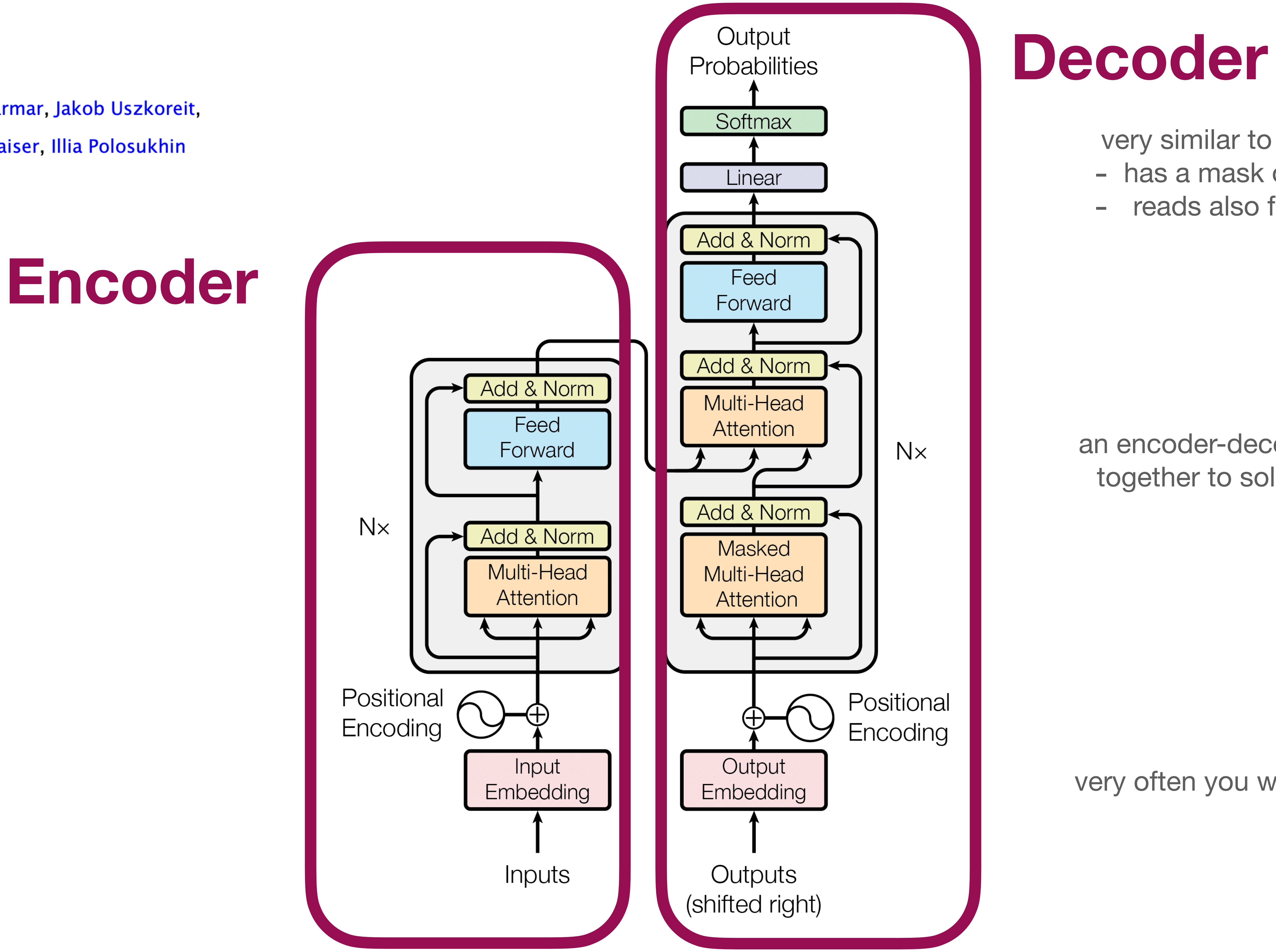
very similar to the encoder but:

- has a mask on the “attention”
- reads also from the decoder

Transformers

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin



very similar to the encoder but:

- has a mask on the “attention”
- reads also from the decoder

an encoder-decoder pair can be trained to work together to solve tasks such as translation or summarisation

very often you will see them independently…

Transformers

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

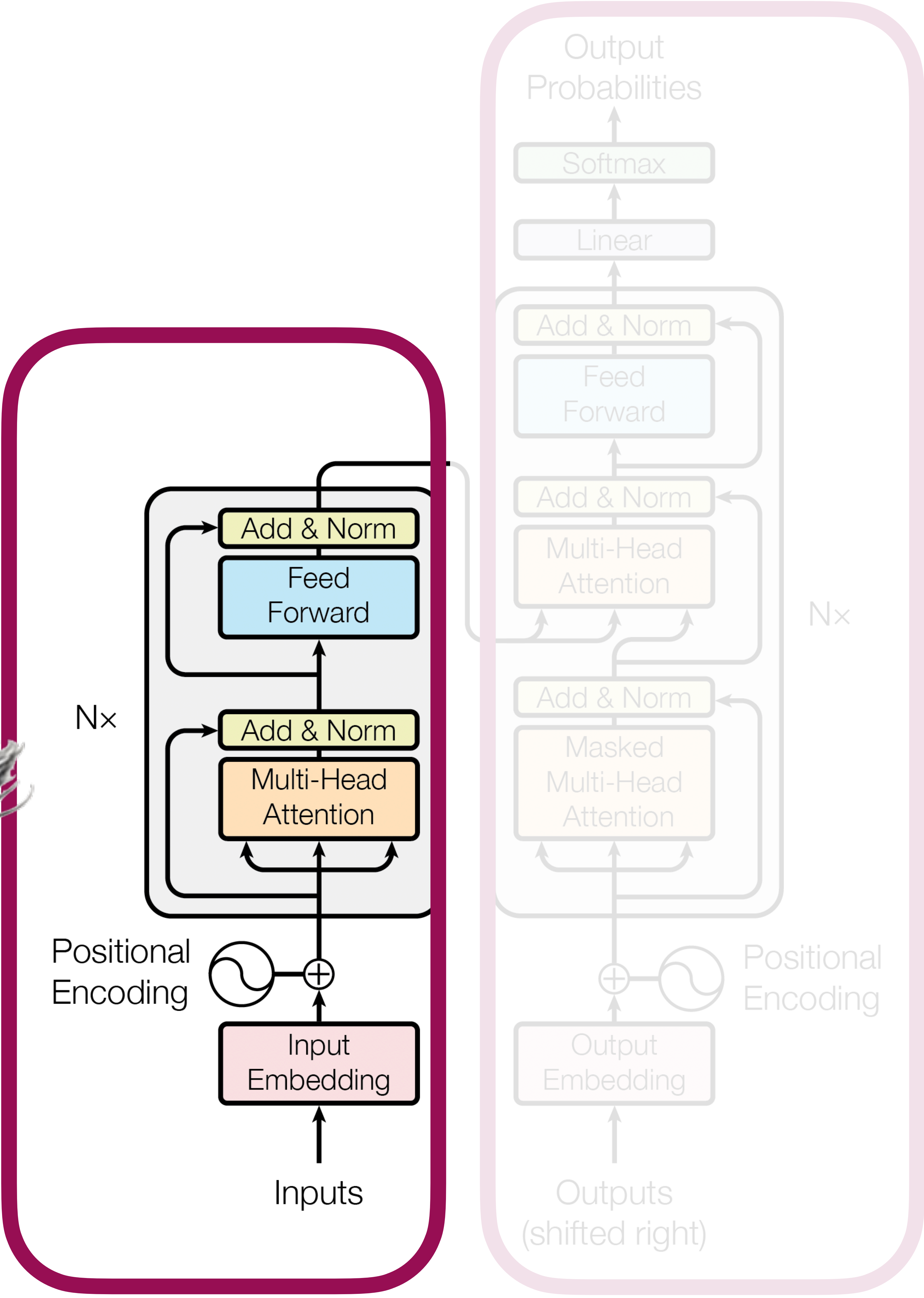
Encoder

Example: BERT

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

Held sota on multiple NLP leaderboards
(now has more competition…)



Decoder

Transformers

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

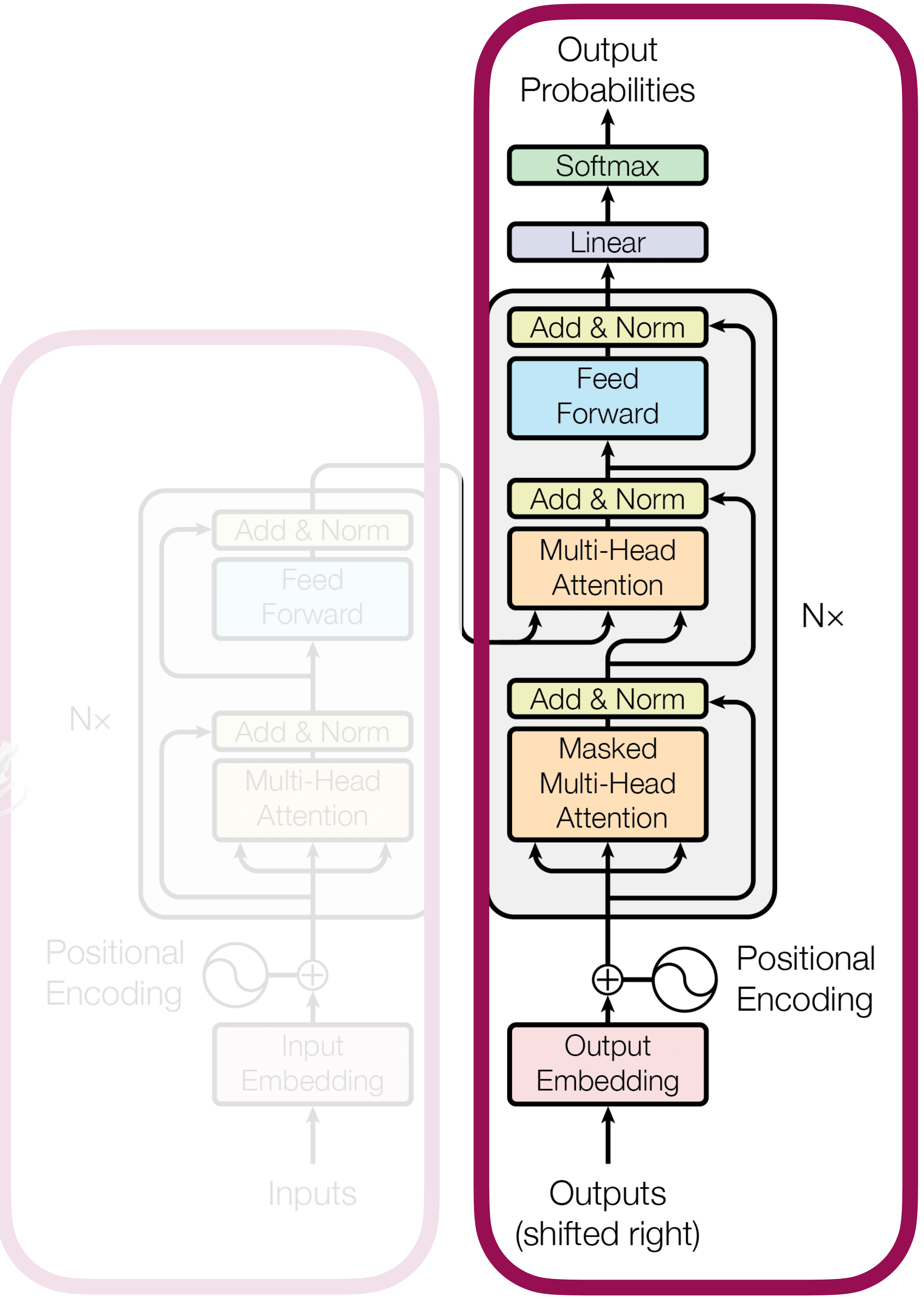
Encoder

Example: BERT

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

Held sota on multiple NLP leaderboards
(now has more competition…)



Decoder

Example: The GPT family of transformers

Improving Language Understanding by Generative Pre-Training

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever

wint but AI @dril_gpt2 · Aug 11
im getting real sick of having to scroll through a bunch of old news to find my name. lets make the news today be about me for once

28 199

MICROSOFT \ TECH \ ARTIFICIAL INTELLIGENCE

Microsoft exclusively licenses OpenAI's groundbreaking GPT-3 text generation model

Forbes
https://www.forbes.com › bernardmarr › 2023/03/01

The Best Examples Of What You Can Do With ChatGPT

1 Mar 2023 — **ChatGPT** is a versatile tool that can be used in a myriad of ways to enhance your productivity and learning. Whether you're looking for quick ...

Wikipedia
https://en.wikipedia.org › wiki › GPT-4

GPT-4

Rumors claim that GPT-4 has **1.76 trillion parameters**, which was first

Transformers

How do they think?????

x+5=11. What is x?

Cool! How did you do that?

Transformers

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

Encoder

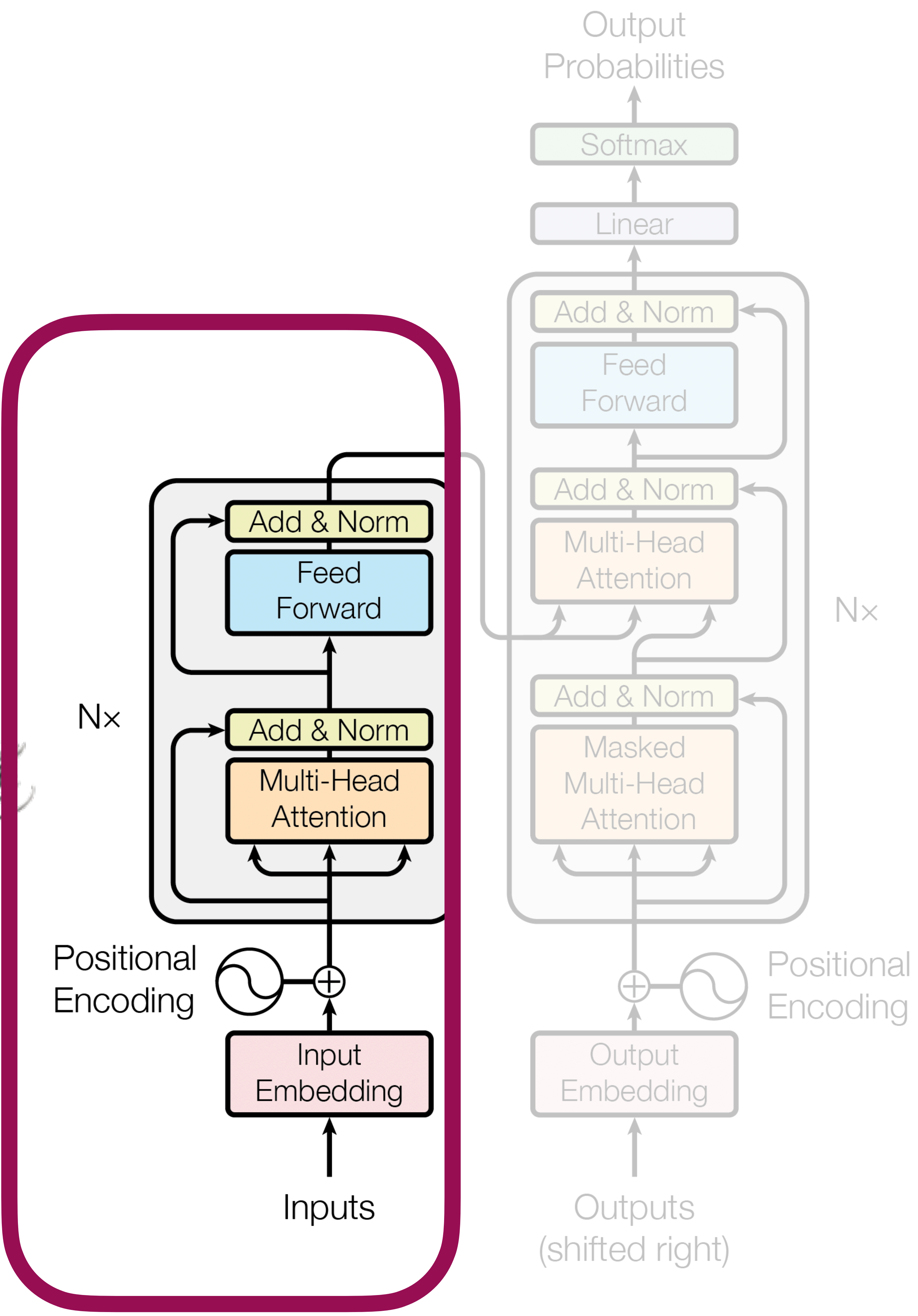
Example: BERT

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

Held sota on multiple NLP leaderboards
(now has more competition… eg XLNet)

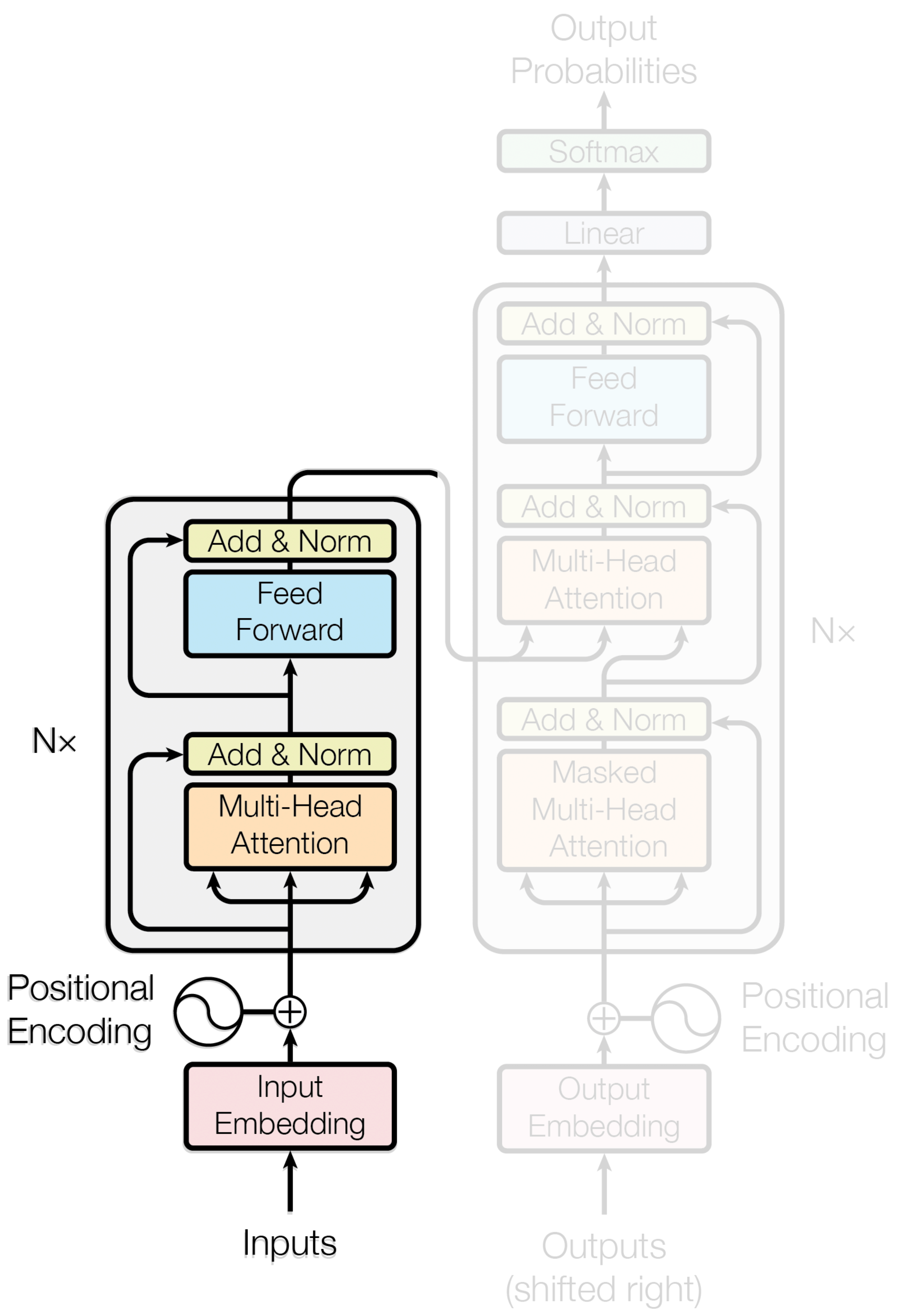
used in Google search!



We will focus on encoders

(understanding decoders will be very simple after encoders)

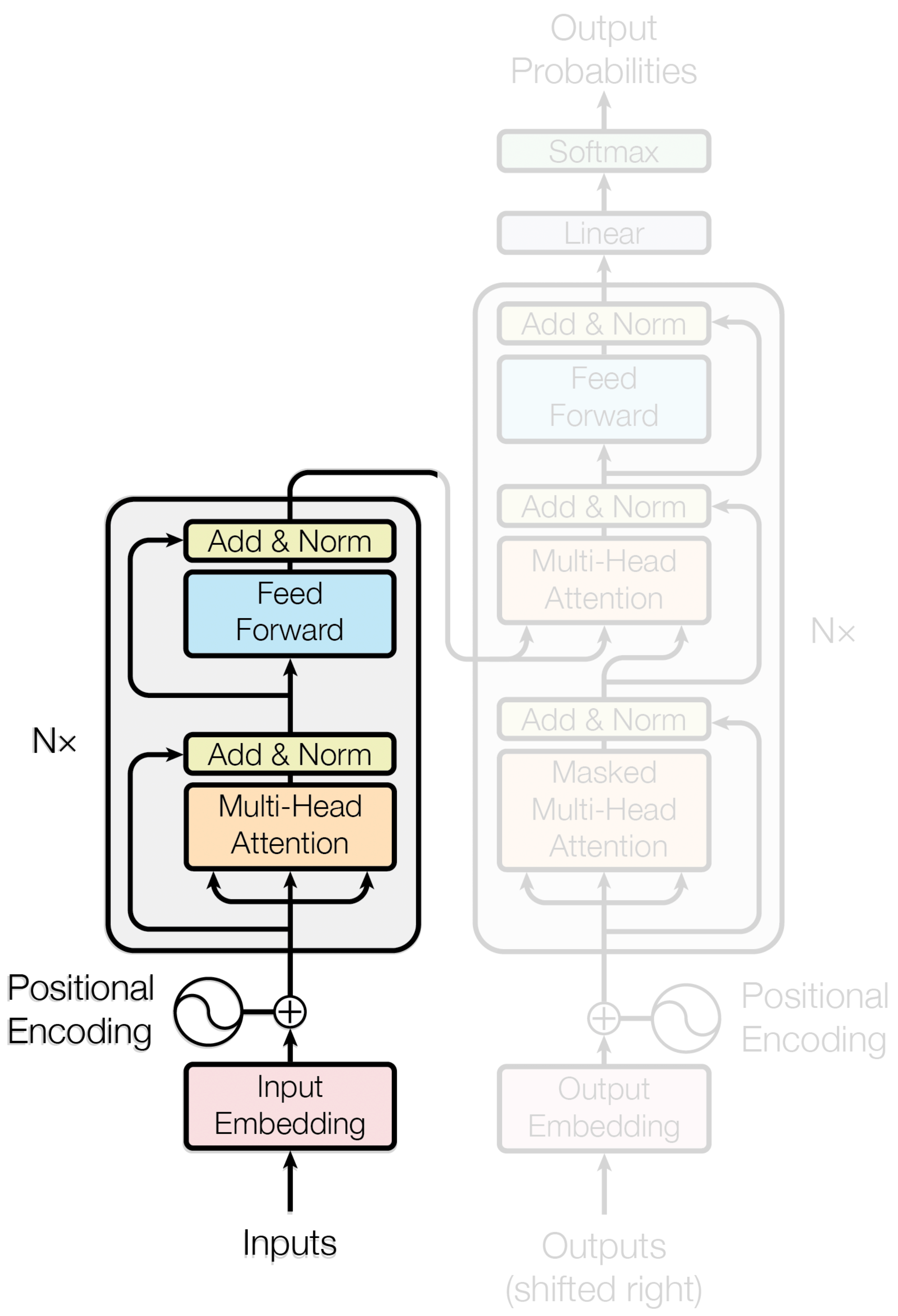
Motivation: Transformer Encoders



How do they think?

We're figuring out all kinds of things…

Motivation: Transformer Encoders



How do they think?

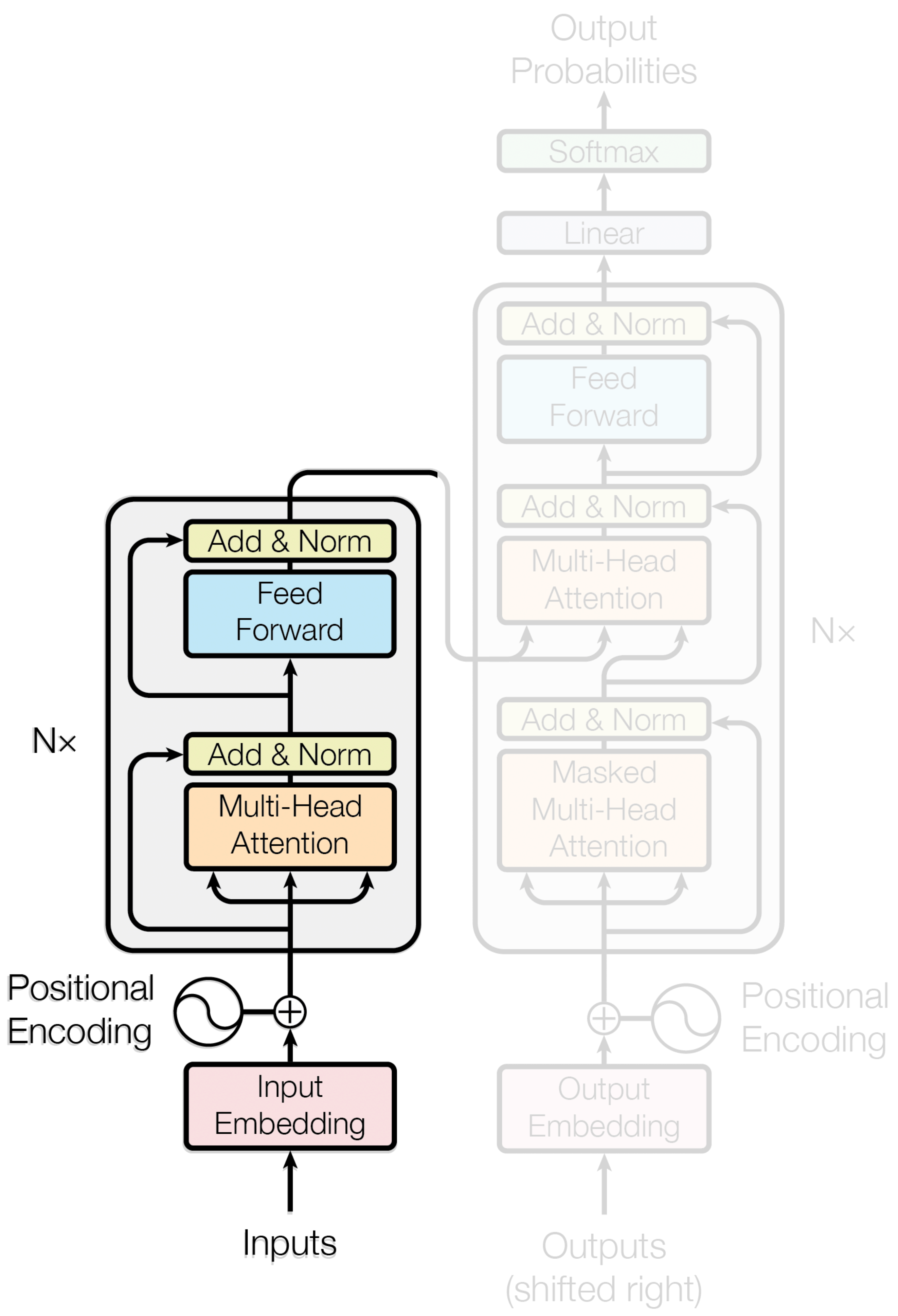
We're figuring out all kinds of things...

Are Transformers universal approximators of sequence-to-sequence functions?

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, Sanjiv Kumar

...but that's not how they *think*!

Motivation: Transformer Encoders



How do they think?

We're figuring out all kinds of things…

Are Transformers universal approximators of sequence-to-sequence functions?

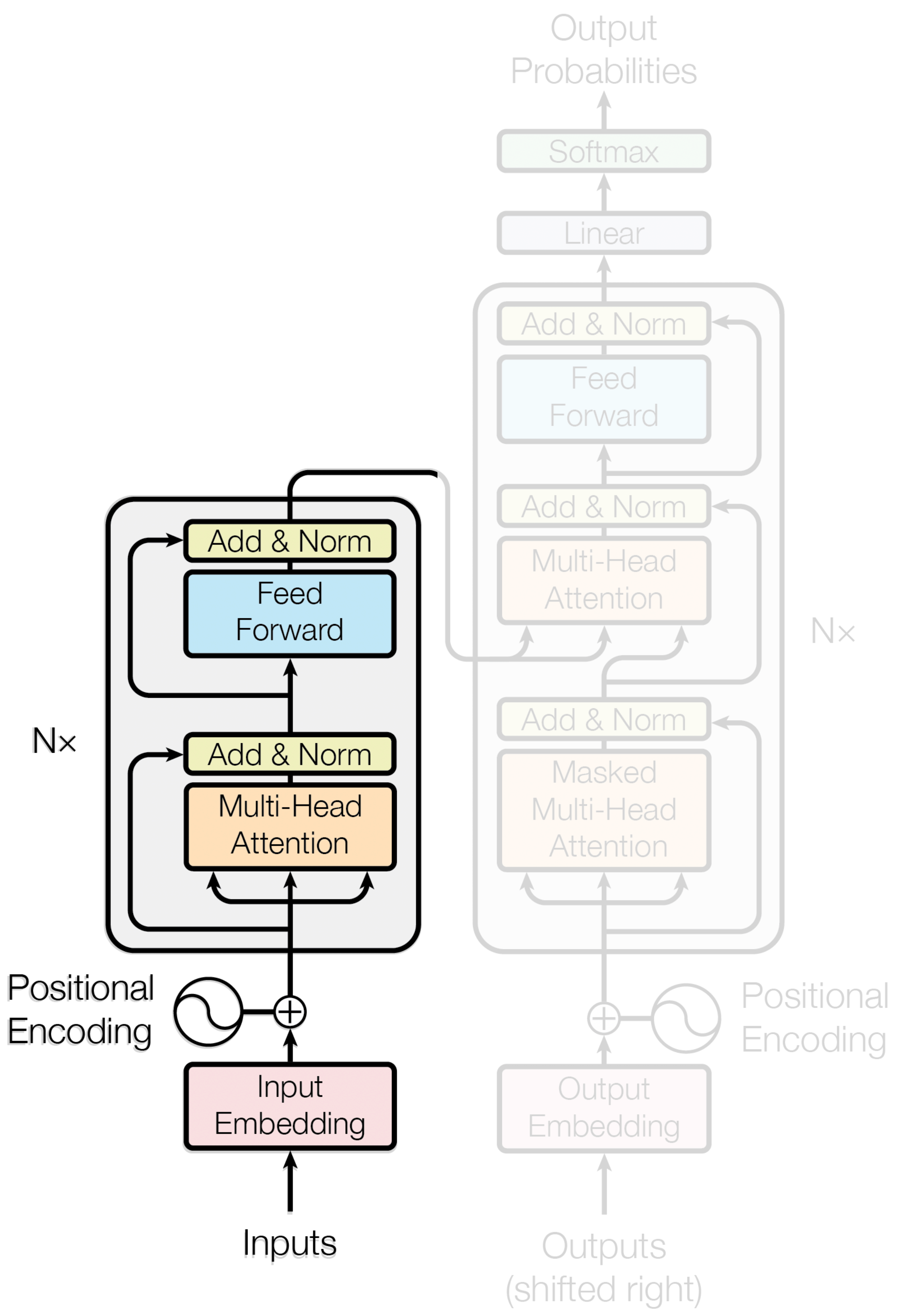
Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, Sanjiv Kumar

Theoretical Limitations of Self-Attention in Neural Sequence Models

Michael Hahn

…but that's not how they *think*!

Motivation: Transformer Encoders



How do they think?

We're figuring out all kinds of things…

Are Transformers universal approximators of sequence-to-sequence functions?

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, Sanjiv Kumar

Theoretical Limitations of Self-Attention in Neural Sequence Models

Michael Hahn

Overcoming a Theoretical Limitation of Self-Attention

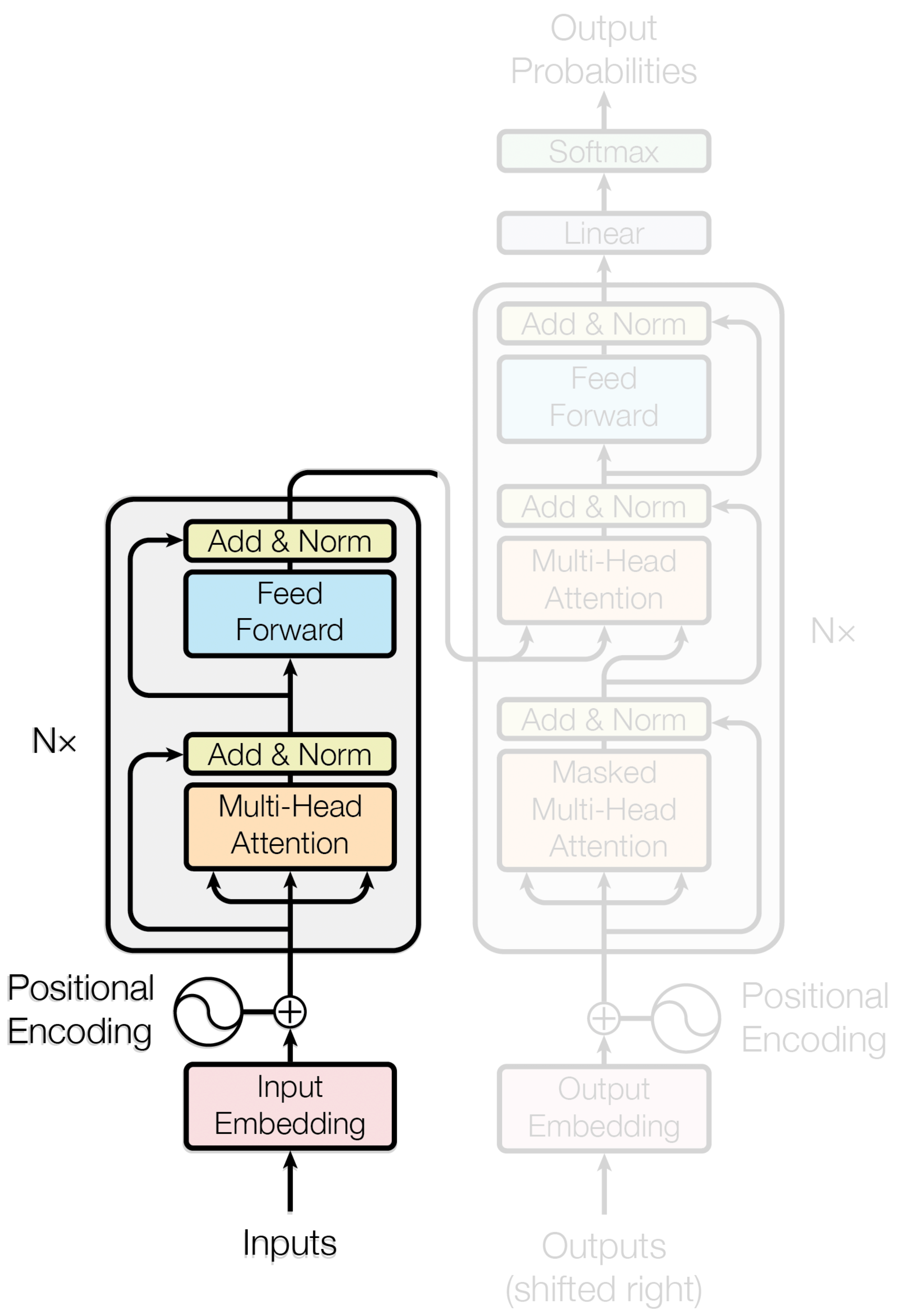
David Chiang, Peter Cholak

Formal Language Recognition by Hard Attention Transformers: Perspectives from Circuit Complexity

Yiding Hao, Dana Angluin, Robert Frank

…but that's not how they *think*!

Motivation: Transformer Encoders



How do they think?

We're figuring out all kinds of things…

Are Transformers universal approximators of sequence-to-sequence functions?

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, Sanjiv Kumar

Theoretical Limitations of Self-Attention in Neural Sequence Models

Michael Hahn

Overcoming a Theoretical Limitation of Self-Attention

David Chiang, Peter Cholak

Formal Language Recognition by Hard Attention Transformers: Perspectives from Circuit Complexity

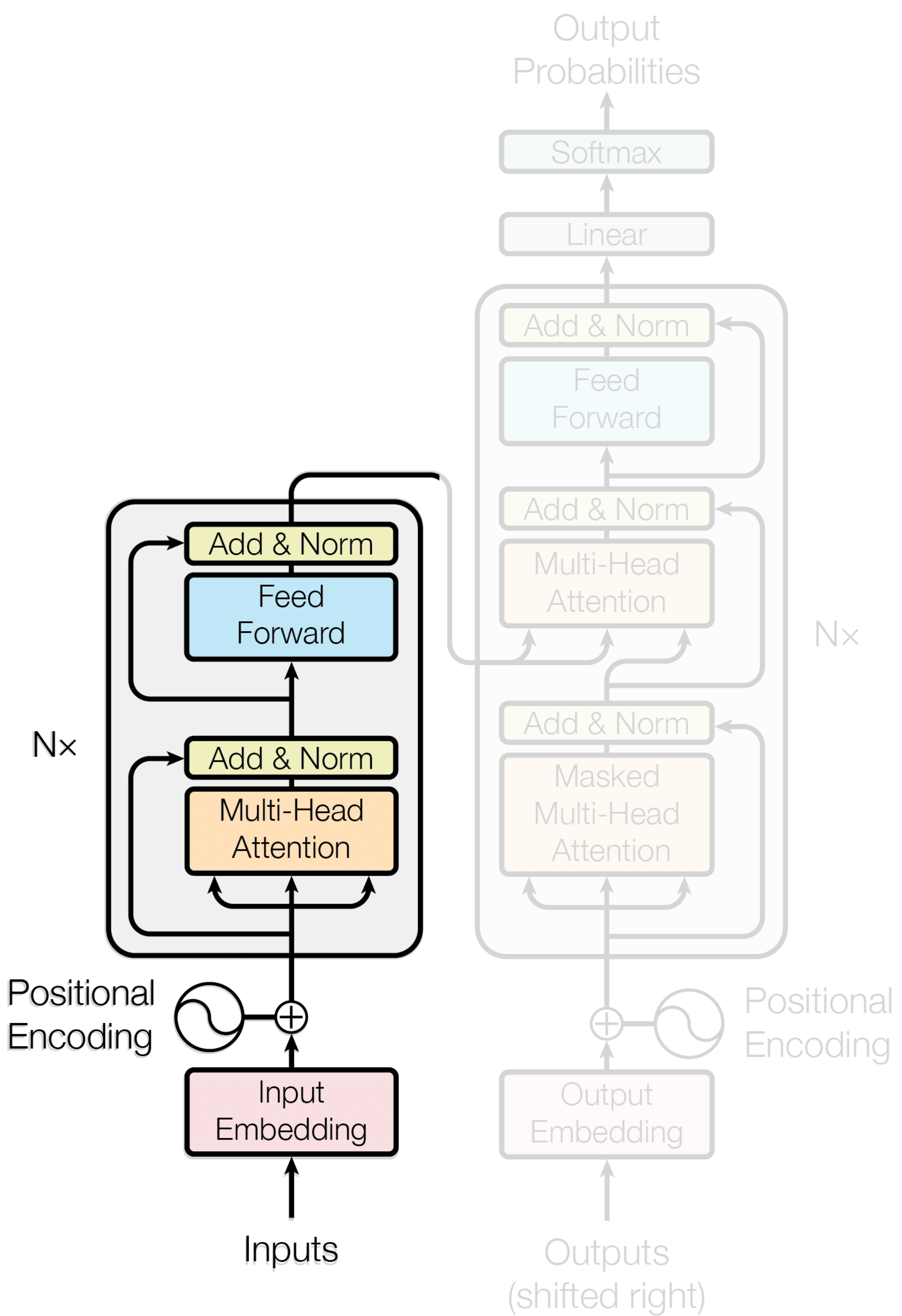
Yiding Hao, Dana Angluin, Robert Frank

On the Ability and Limitations of Transformers to Recognize Formal Languages

Satwik Bhattamishra, Kabir Ahuja, Navin Goyal

…but that's not how they *think*!

Motivation: Transformer Encoders



Transformers as Recognizers of Formal Languages: A Survey on Expressivity

Lena Strobl, William Merrill, Gail Weiss, David Chiang, Dana Angluin

How do they think?

We're figuring out all kinds of things…

Are Transformers universal approximators of sequence-to-sequence functions?

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, Sanjiv Kumar

Theoretical Limitations of Self-Attention in Neural Sequence Models

Michael Hahn

Overcoming a Theoretical Limitation of Self-Attention

David Chiang, Peter Cholak

Formal Language Recognition by Hard Attention Transformers: Perspectives from Circuit Complexity

Yiding Hao, Dana Angluin, Robert Frank

On the Ability and Limitations of Transformers to Recognize Formal Languages

Satwik Bhattamishra, Kabir Ahuja, Navin Goyal

Attention is Turing-Complete

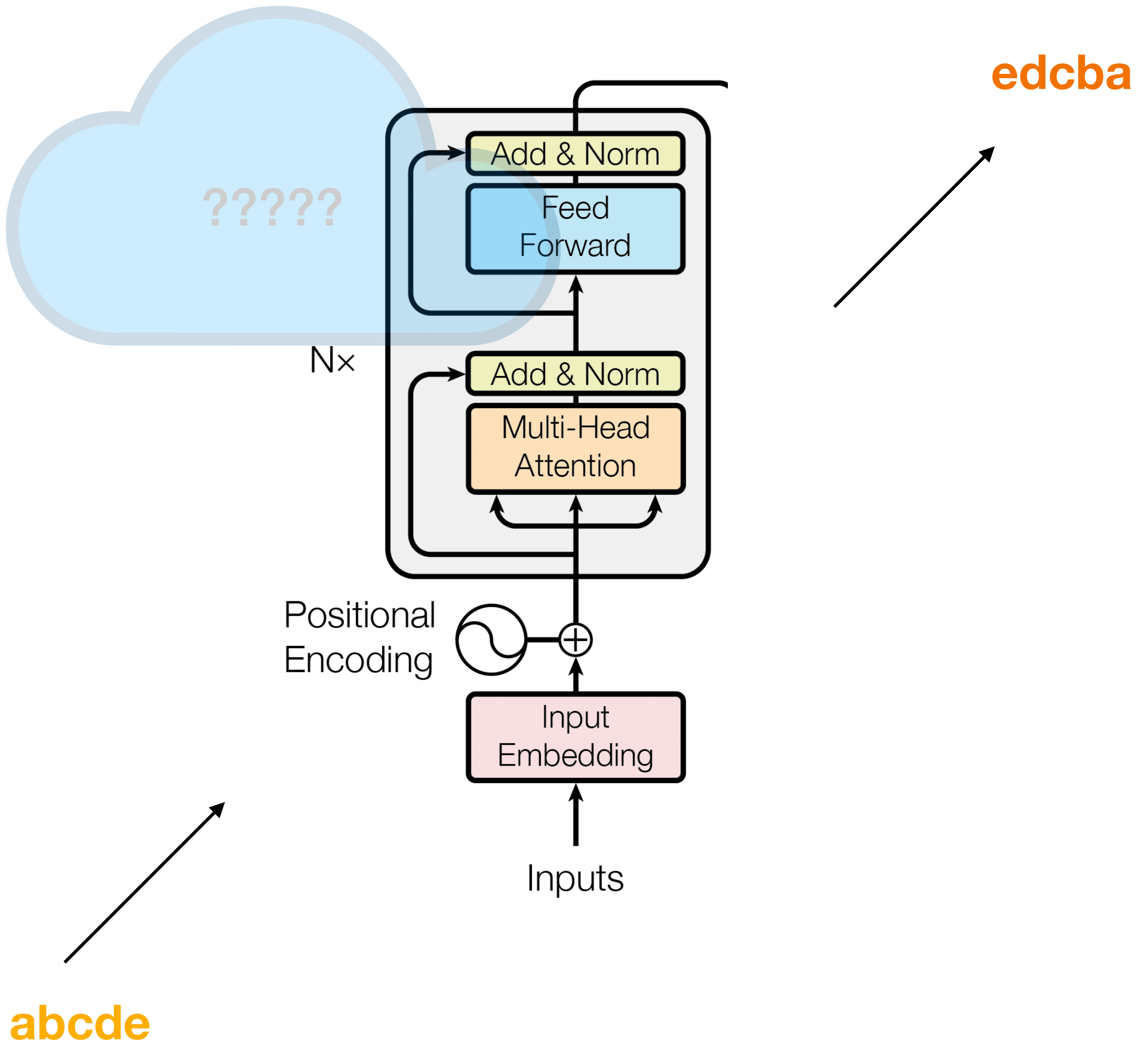
Jorge Pérez, Pablo Barceló, Javier Marinkovic; 22(75):1−35, 2021.

Statistically Meaningful Approximation: a Case Study on Approximating Turing Machines with Transformers

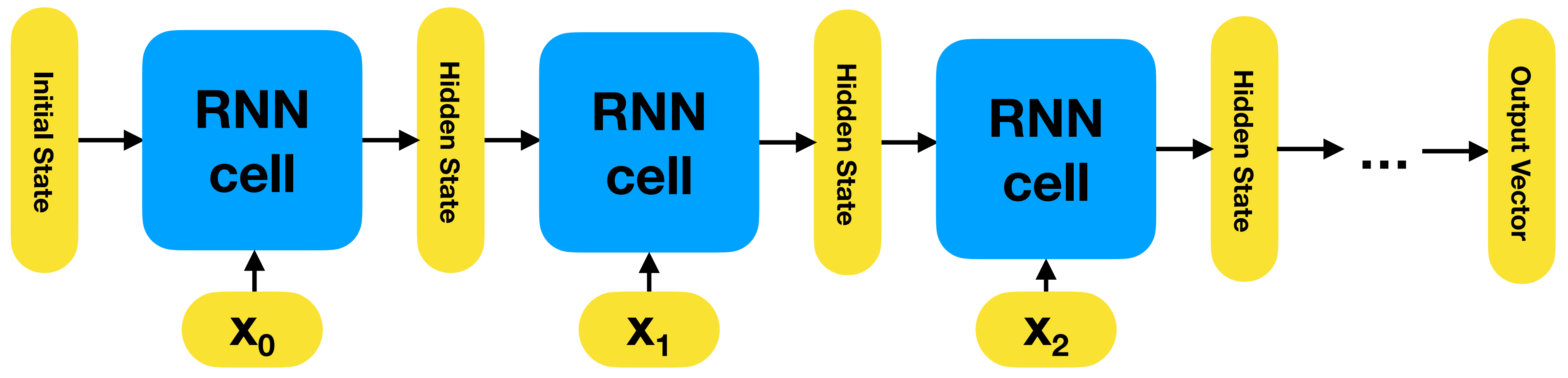
Colin Wei, Yining Chen, Tengyu Ma

…but that's not how they *think*!

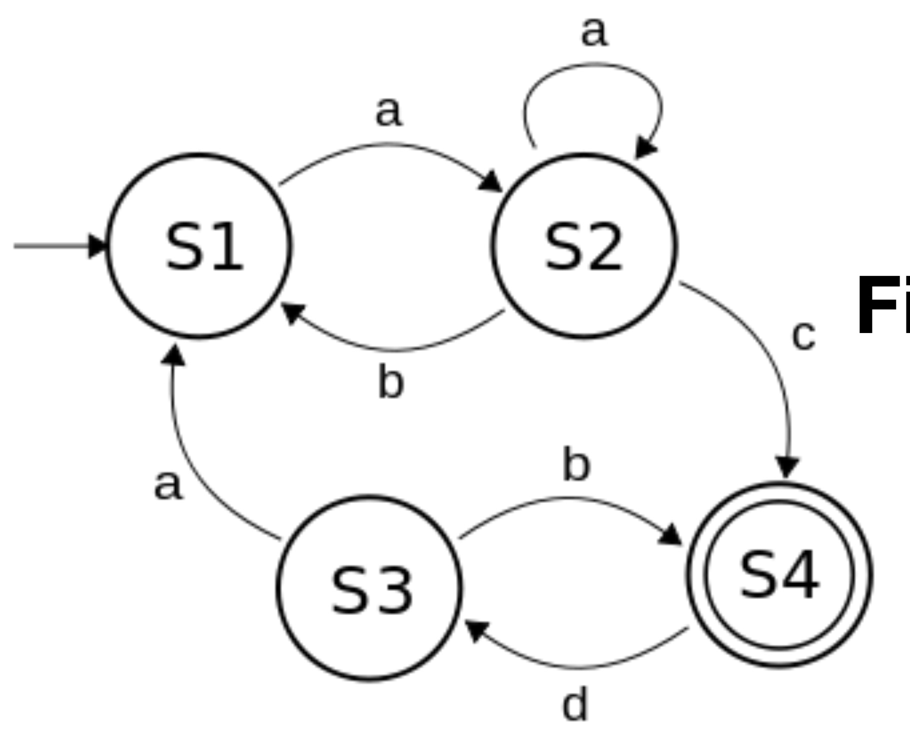
Teaser: Reverse



Motivation: What RNNs have

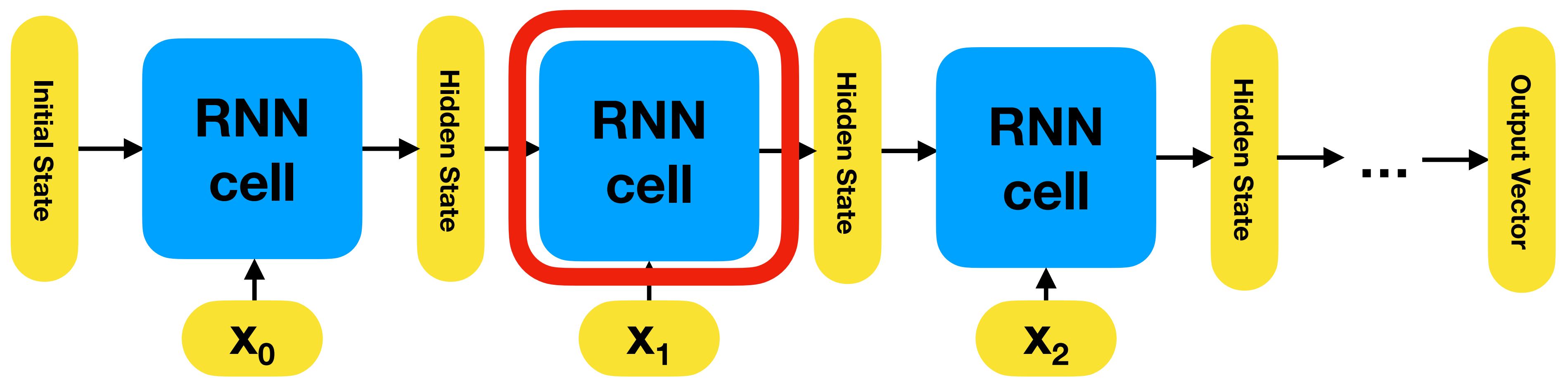


Computational Model(s)!

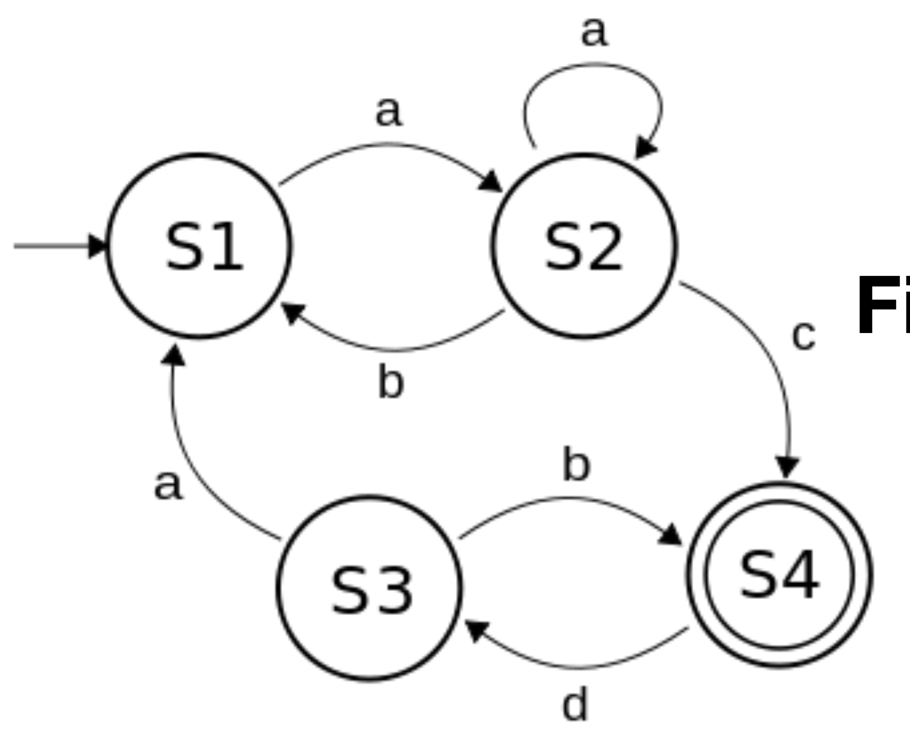


Deterministic Finite Automata (DFAs)

Motivation: What RNNs have

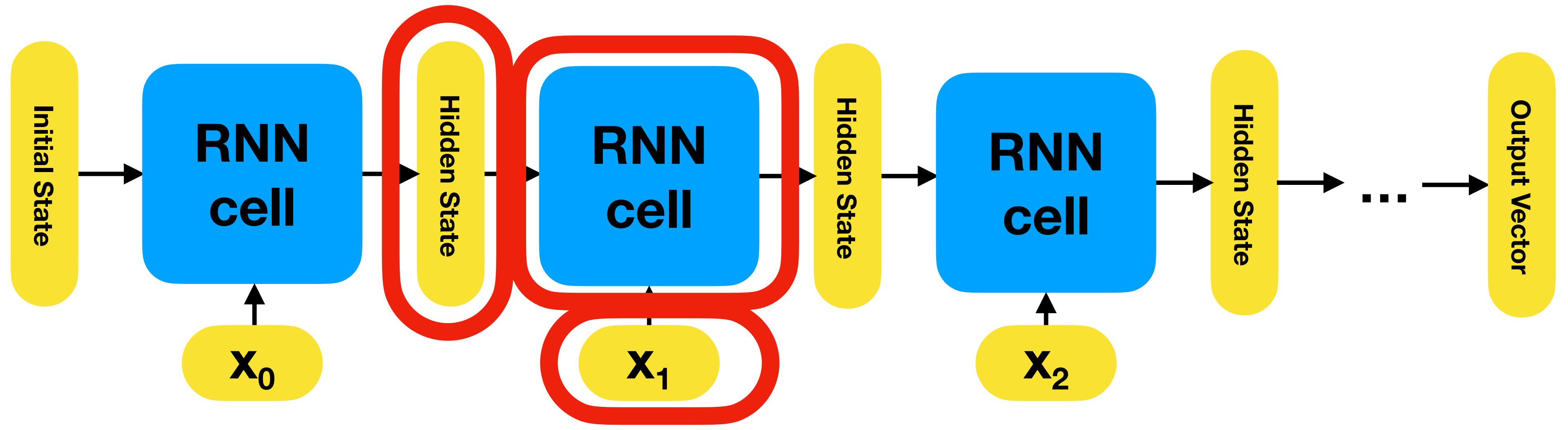


Computational Model(s)!

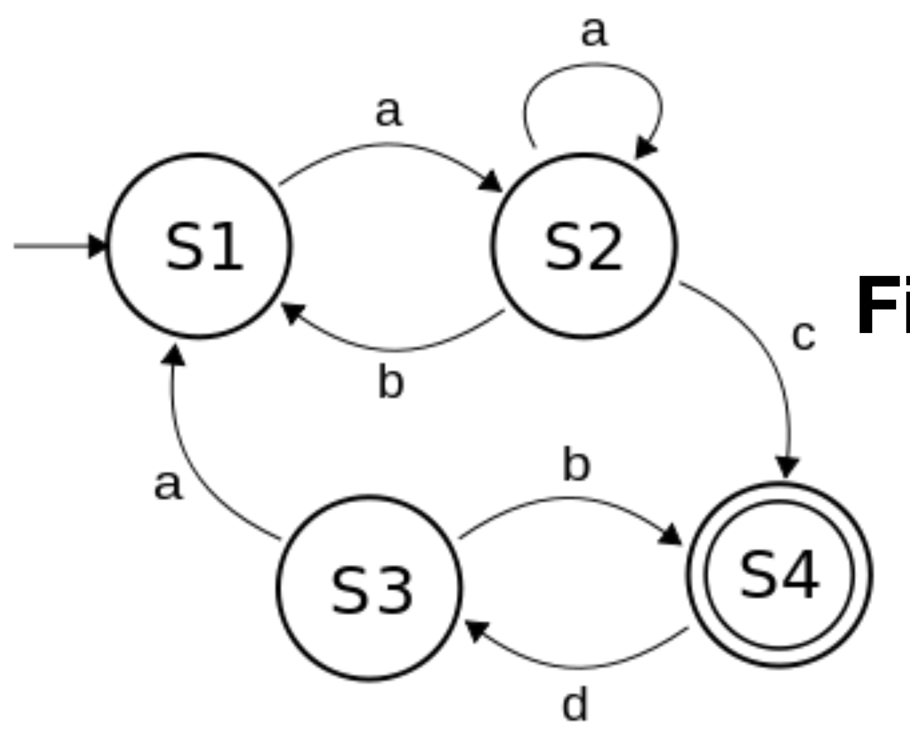


Deterministic Finite Automata (DFAs)

Motivation: What RNNs have

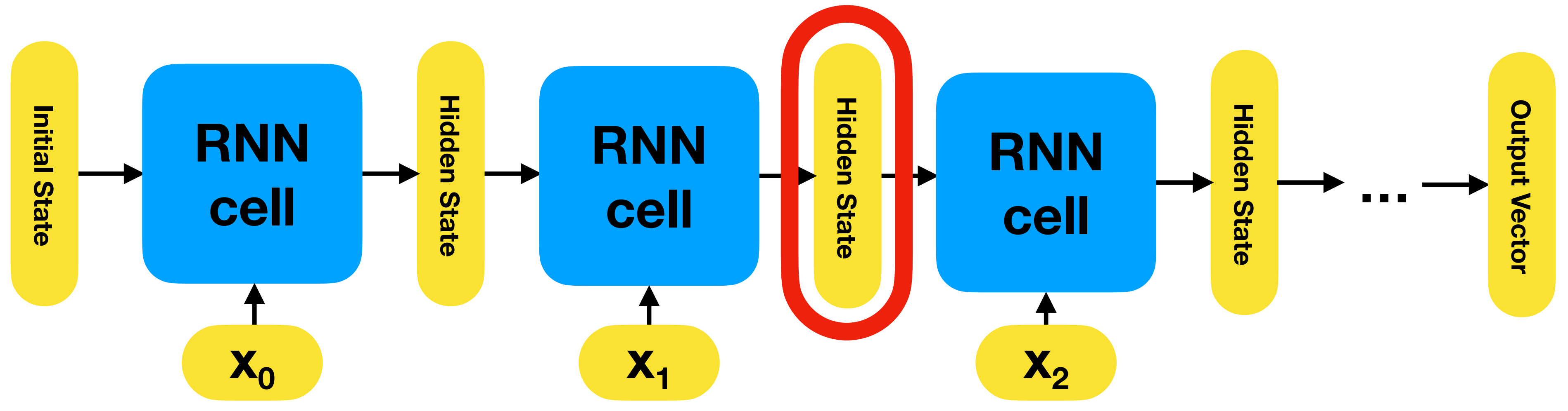


Computational Model(s)!

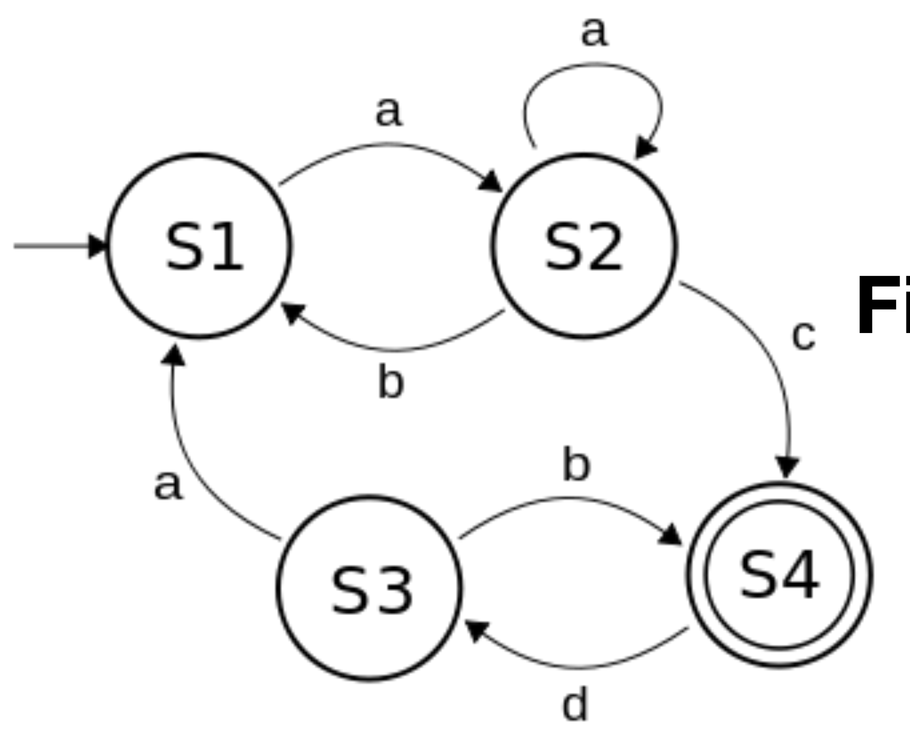


Deterministic Finite Automata (DFAs)

Motivation: What RNNs have

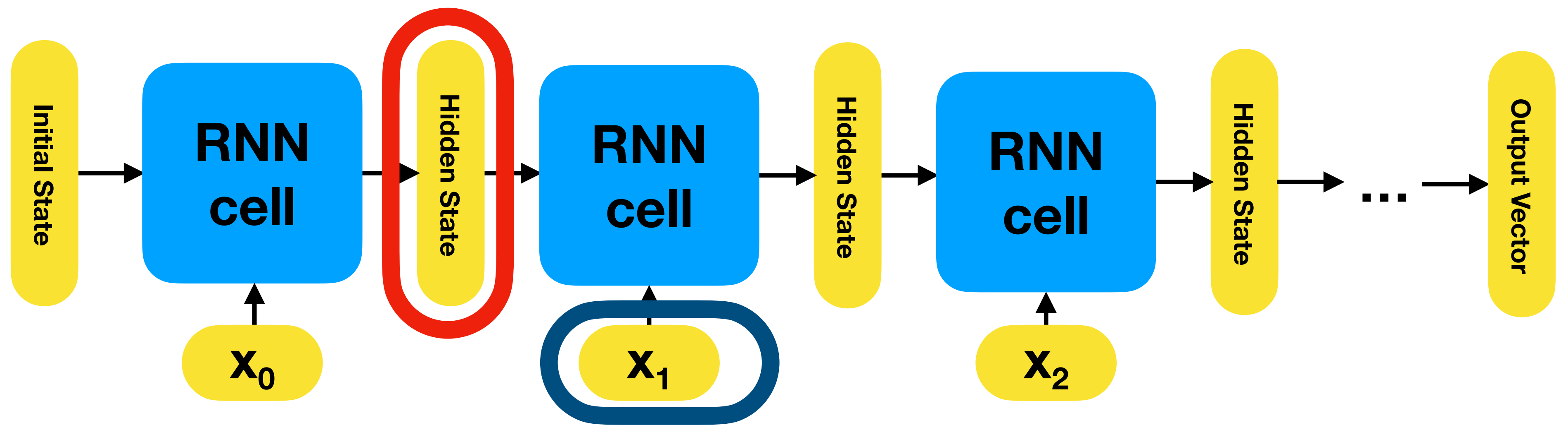


Computational Model(s)!

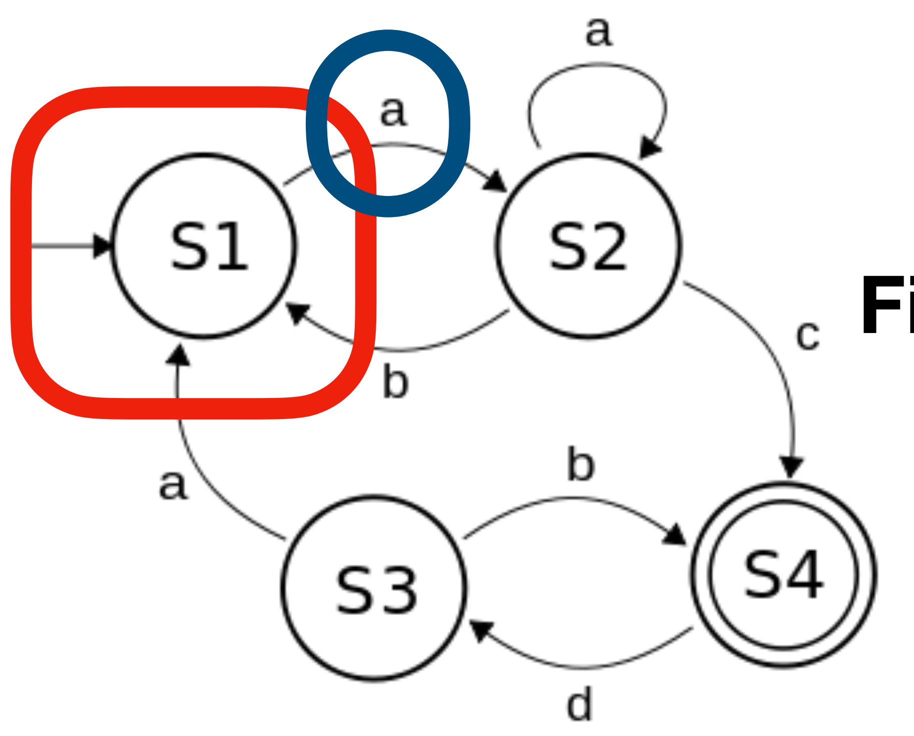


Deterministic Finite Automata (DFAs)

Motivation: What RNNs have

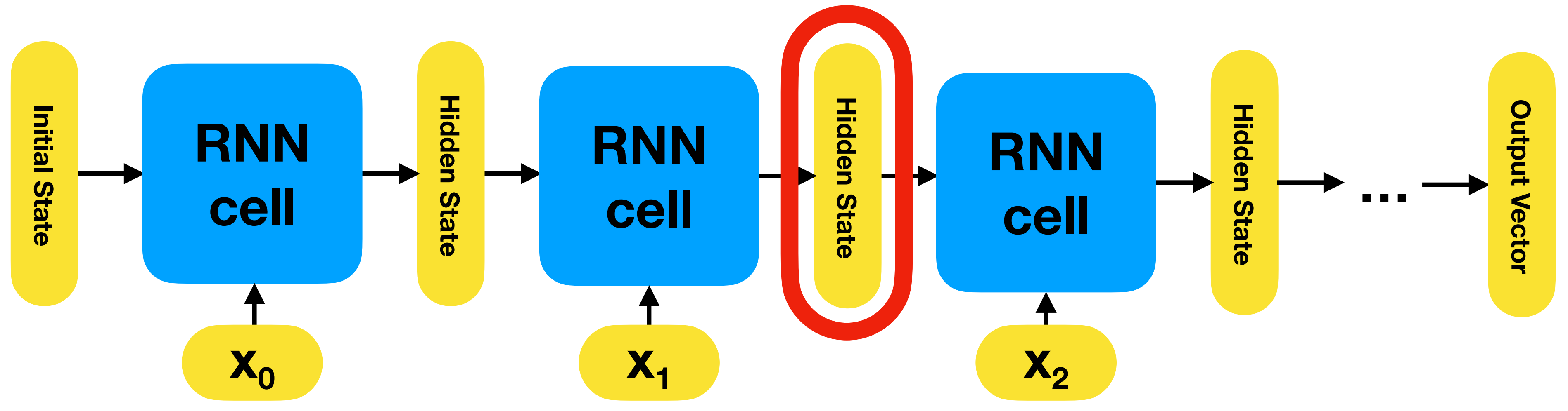


Computational Model(s)!

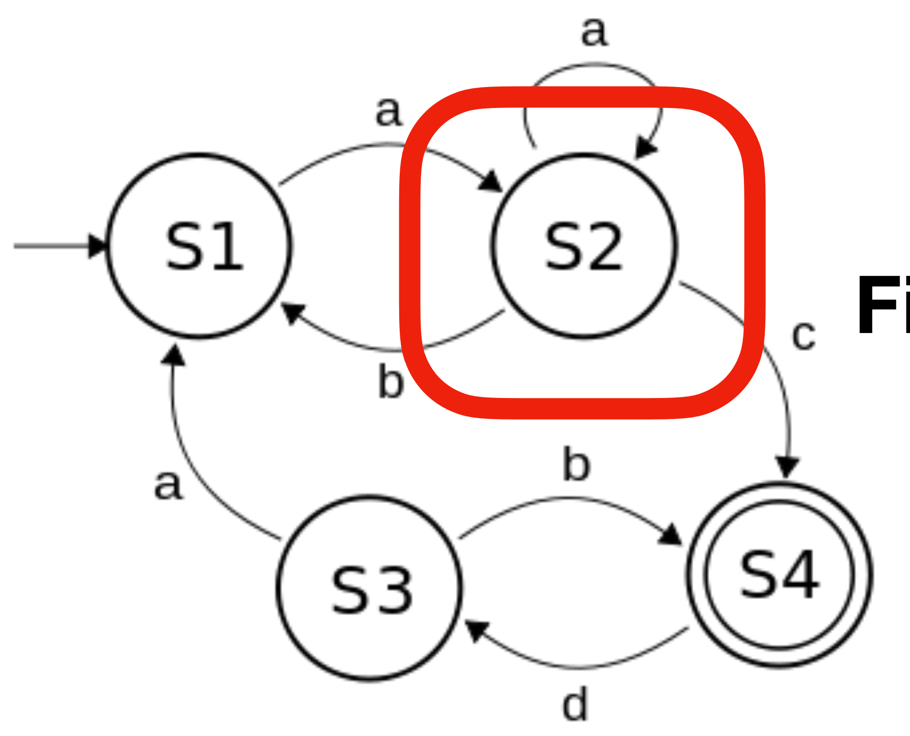


Deterministic Finite Automata (DFAs)

Motivation: What RNNs have

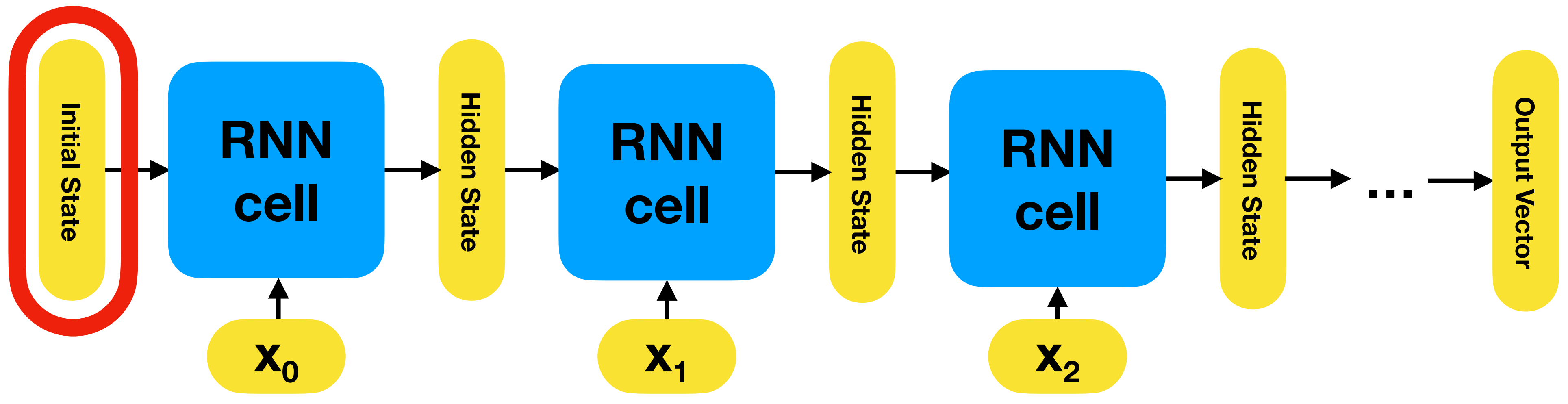


Computational Model(s)!

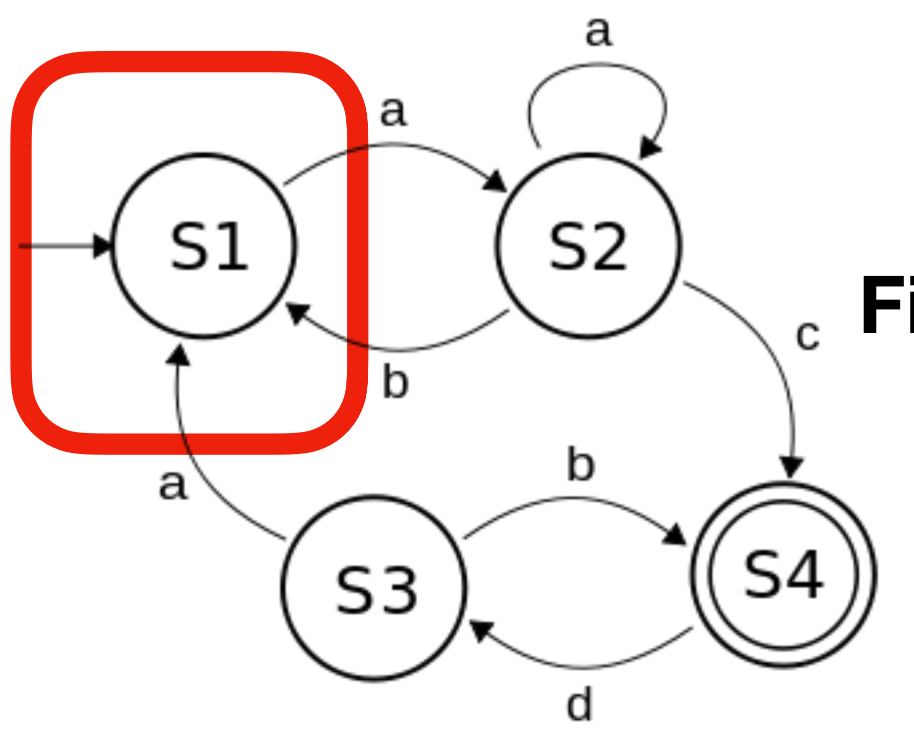


Deterministic Finite Automata (DFAs)

Motivation: What RNNs have

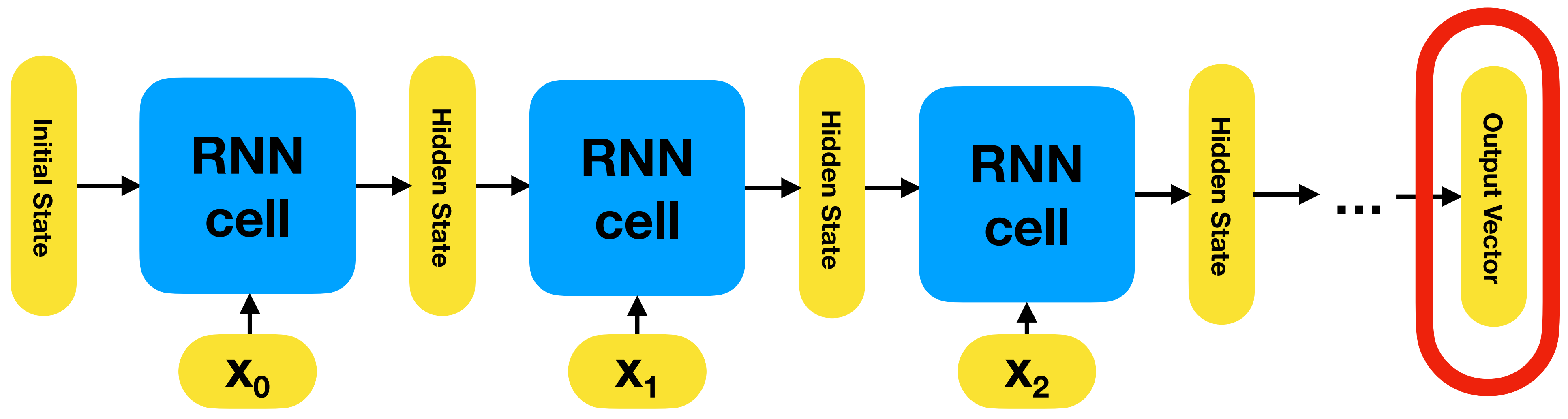


Computational Model(s)!

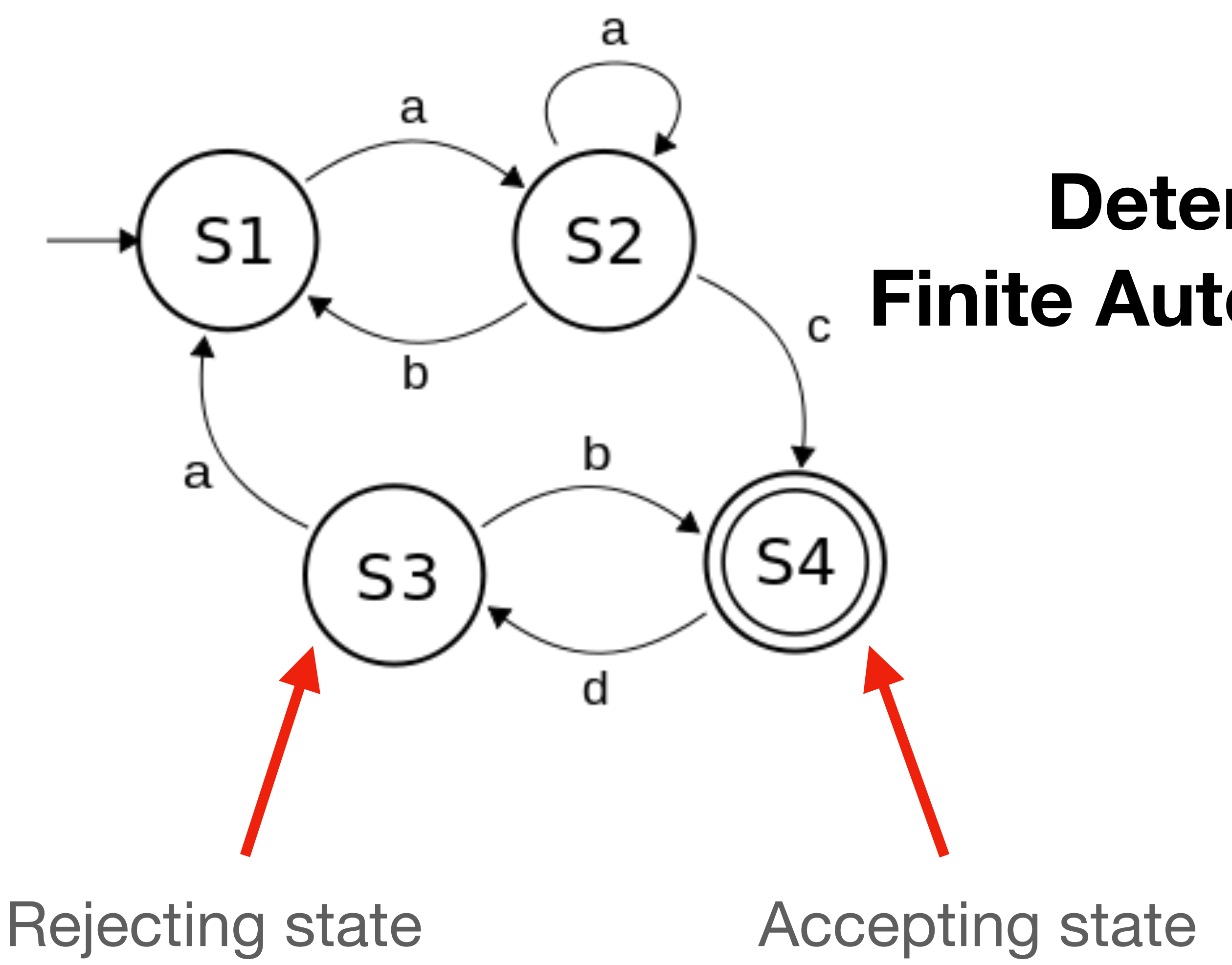


Deterministic Finite Automata (DFAs)

Motivation: What RNNs have

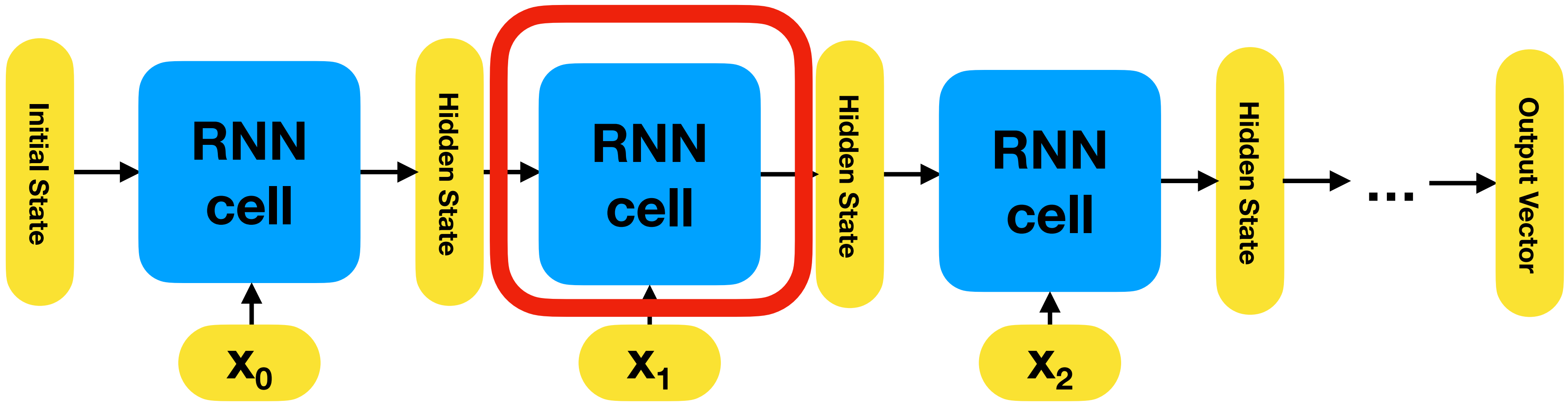


Computational Model(s)!

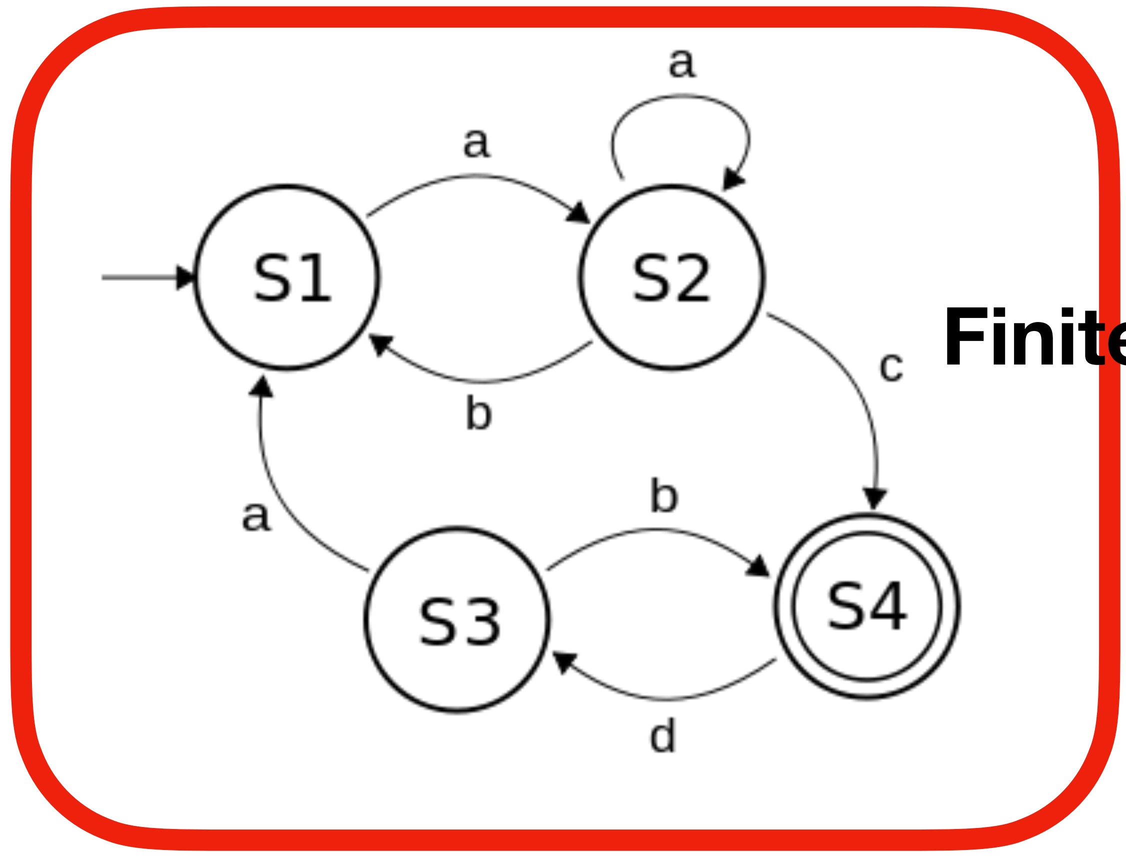


Deterministic Finite Automata (DFAs)

Motivation: What RNNs have

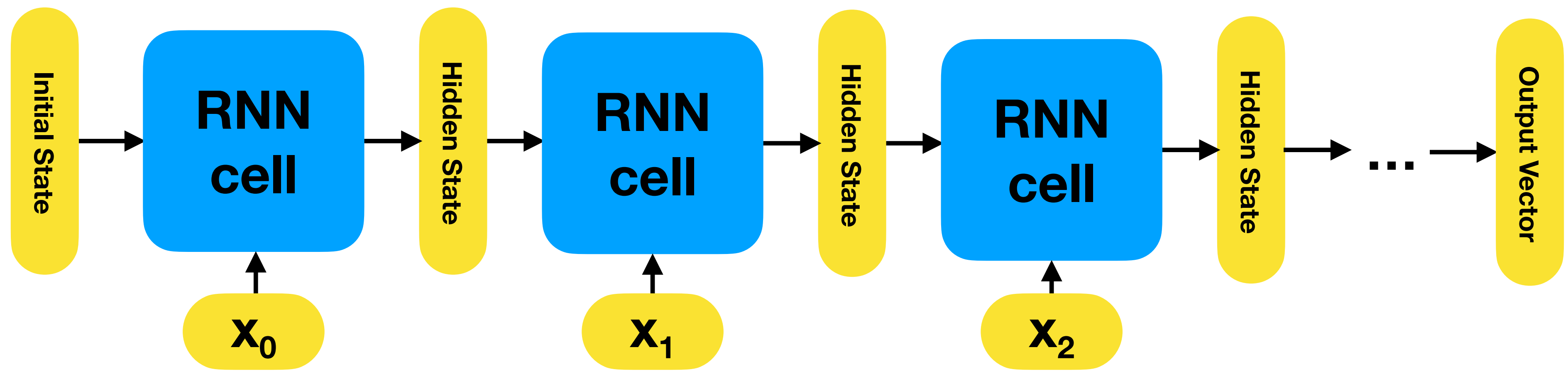


Computational Model(s)!

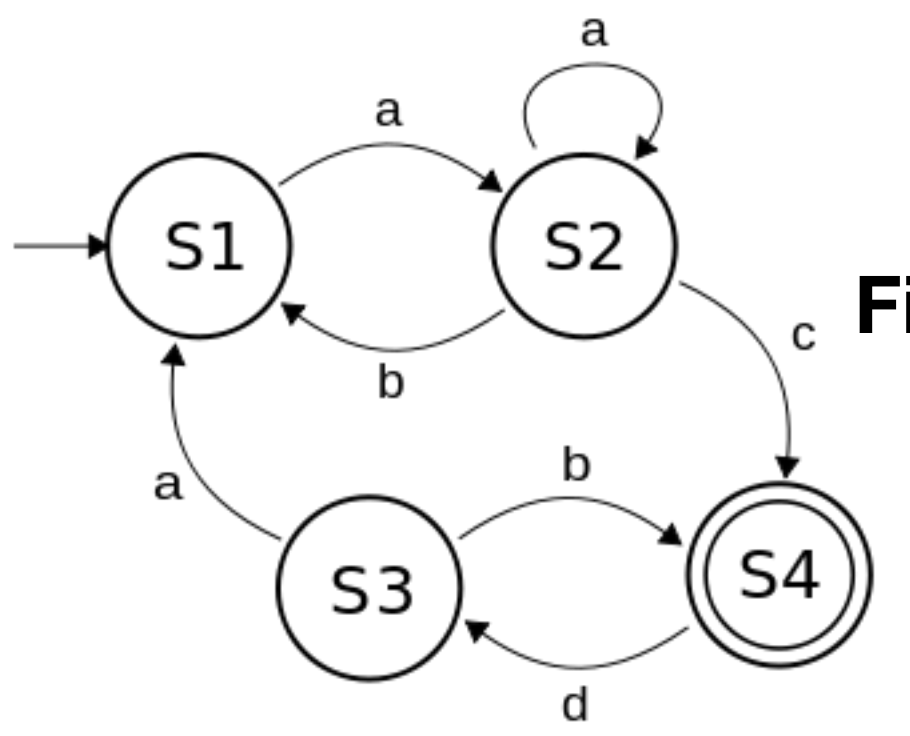


Deterministic Finite Automata (DFAs)

Motivation: What RNNs have

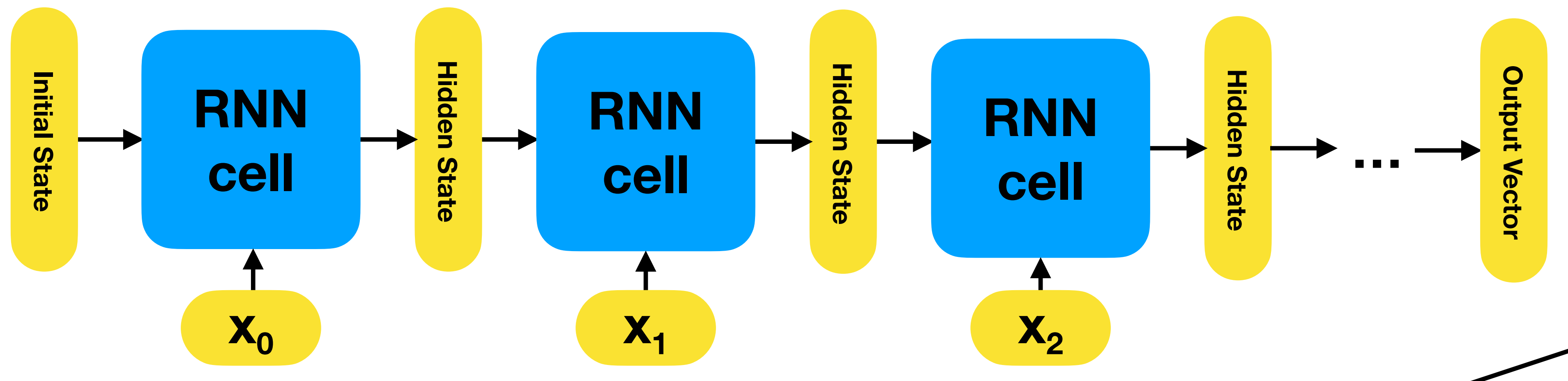


Computational Model(s)!

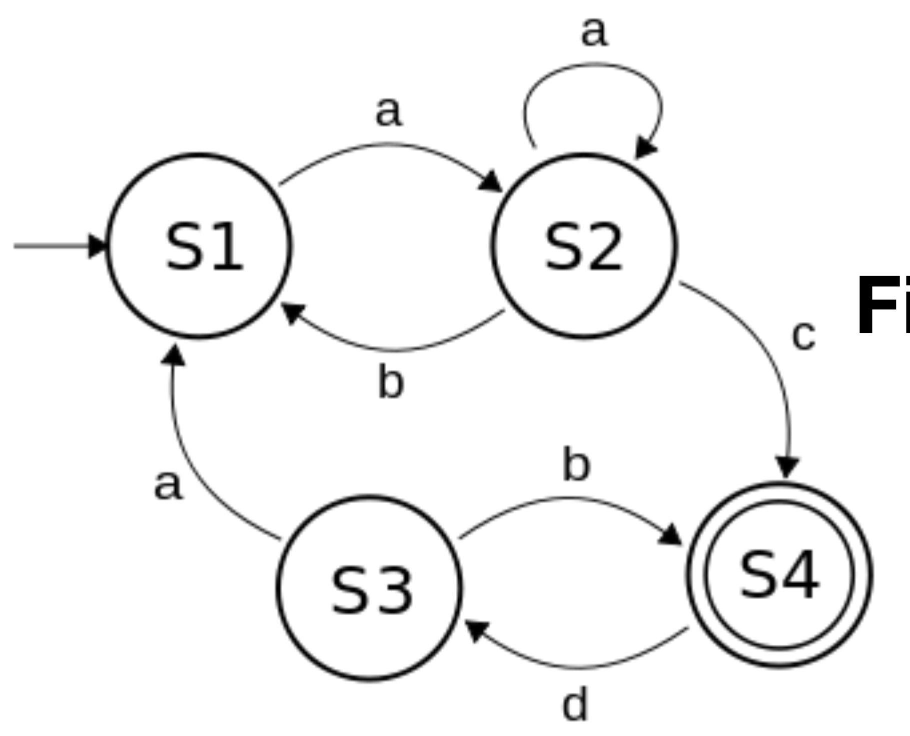


Deterministic Finite Automata (DFAs)

Motivation: What RNNs have



Computational Model(s)!



Deterministic Finite Automata (DFAs)

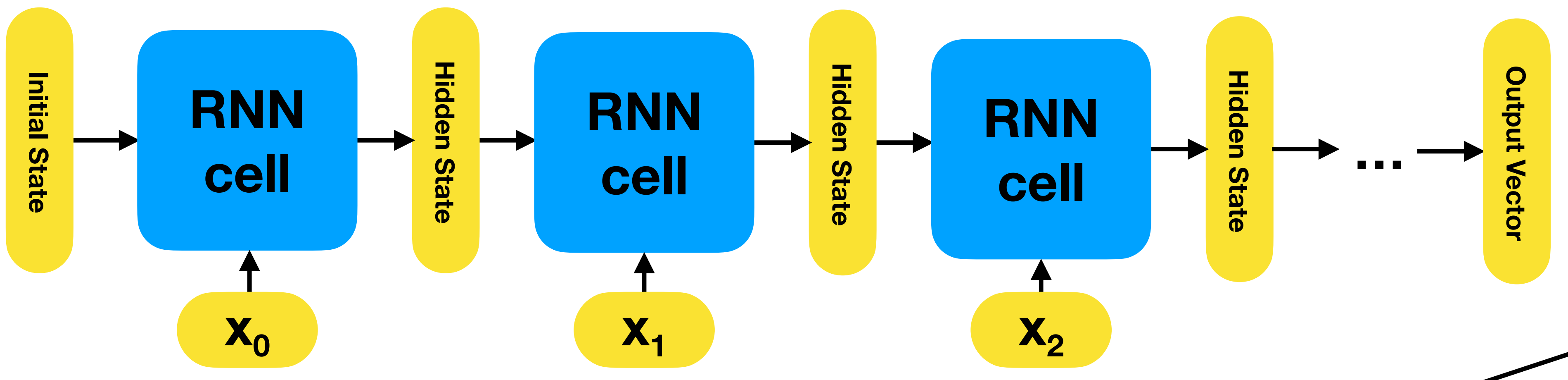
Extraction!

Spectral extraction:
RNNs to WFAs

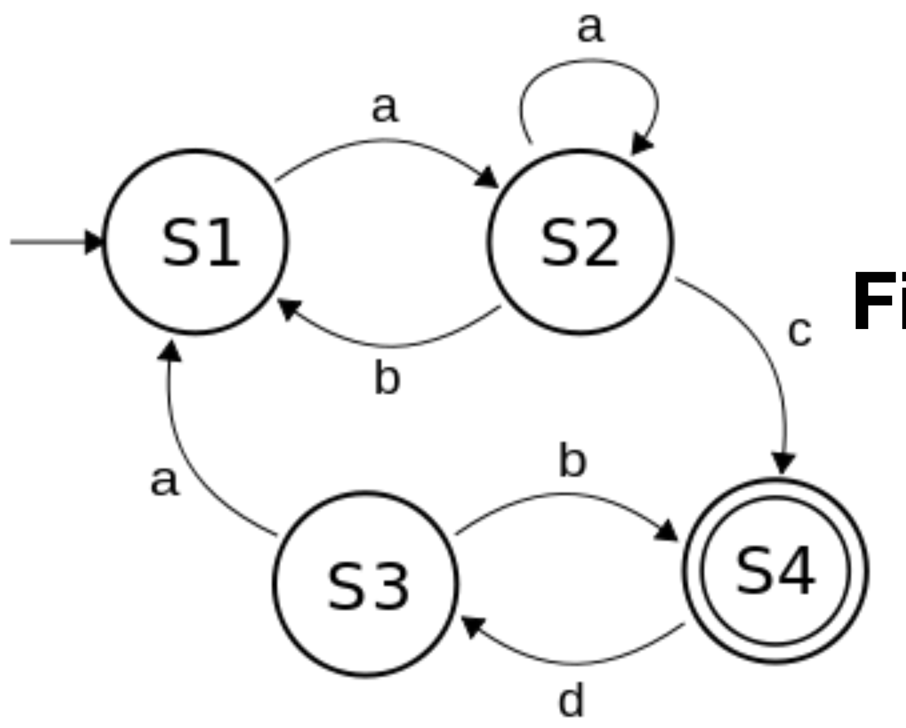
DFA extraction:
Clustering

DFA and WDFA extraction:
L-star variants

Motivation: What RNNs have



Computational Model(s)!



Deterministic Finite Automata (DFAs)

Extraction!

Spectral extraction:
RNNs to WFAs

DFA extraction:
Clustering

DFA and WDFA extraction:
L-star variants

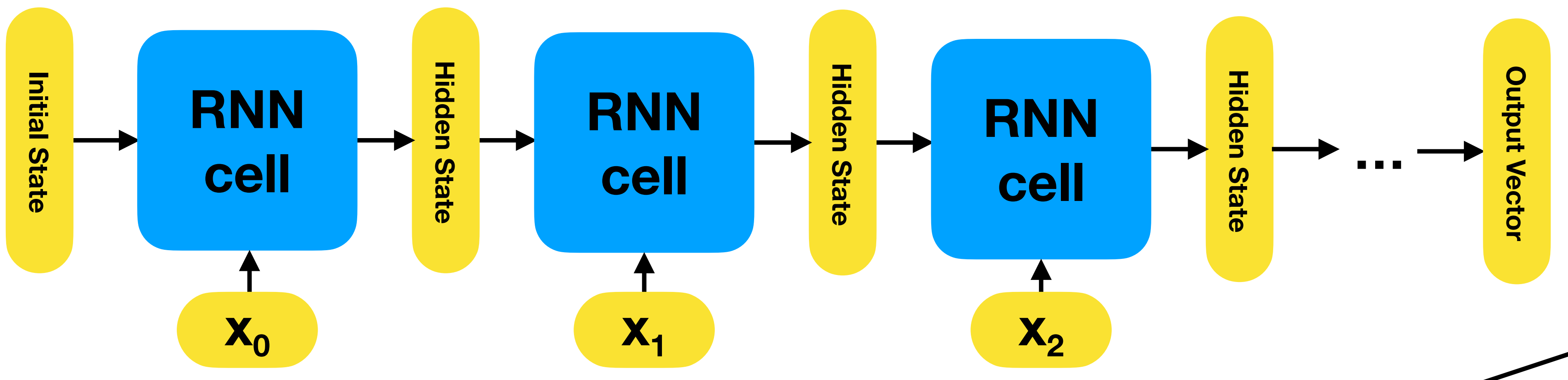
Analysis of Expressive Power!

2-RNNs are WFAs

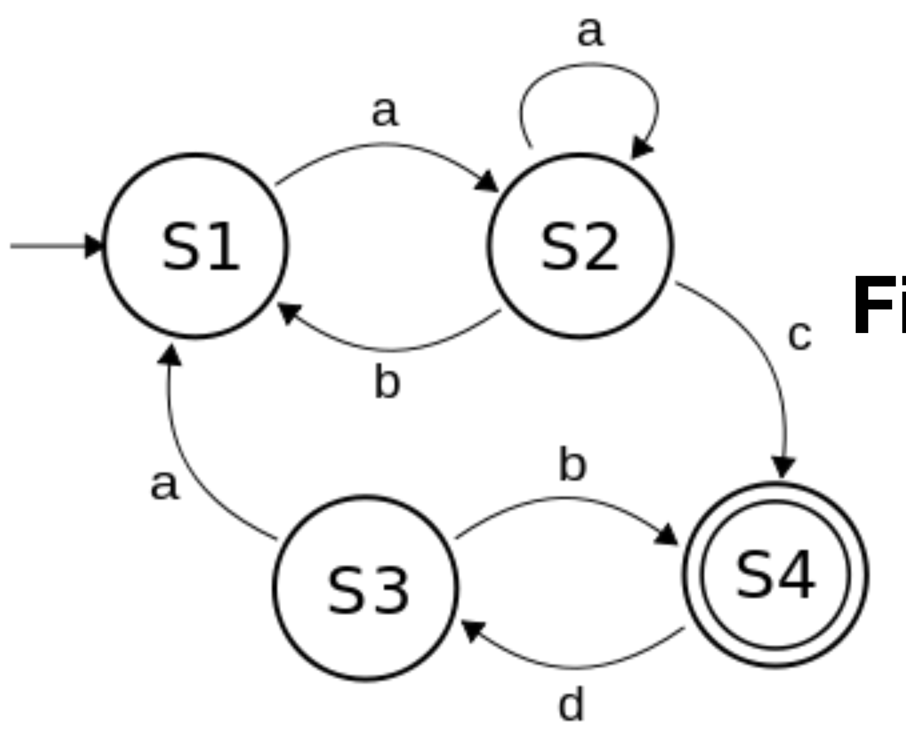
LSTMs are counter machines

GRUs are DFAs

Motivation: What RNNs have



Computational Model(s)!



Deterministic Finite Automata (DFAs)

Extraction!

Spectral extraction:
RNNs to WFAs

DFA extraction:
Clustering

DFA and WDFA extraction:
L-star variants

Analysis of Expressive Power!

2-RNNs are WFAs

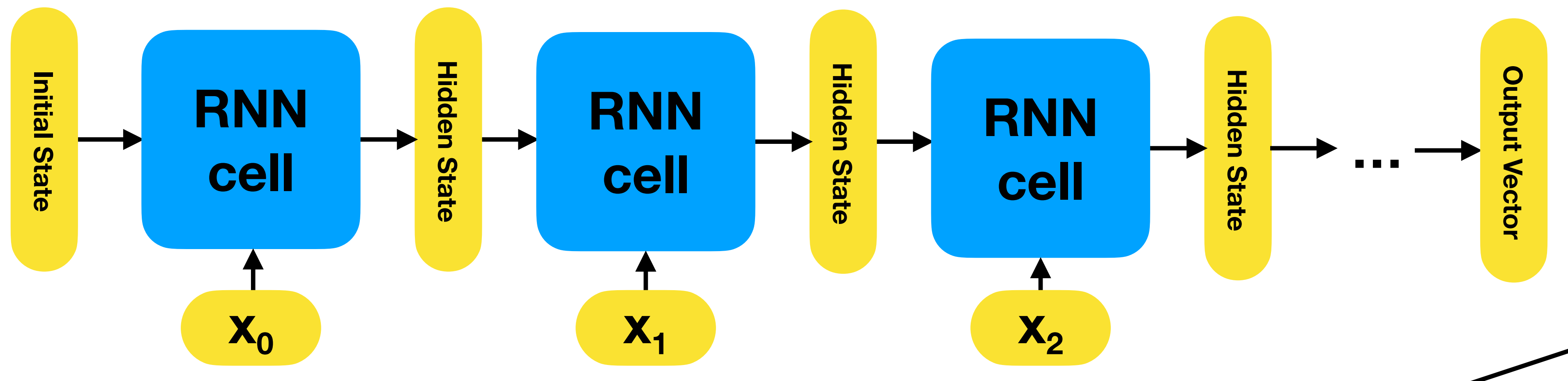
LSTMs are counter machines

GRUs are DFAs

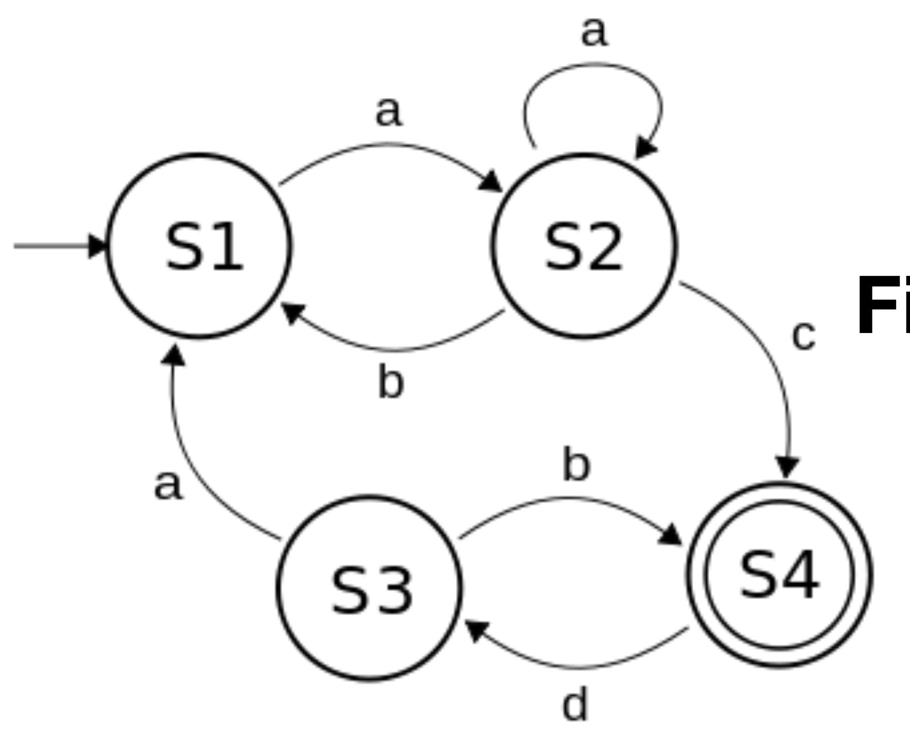
Inspiration from existing theory!

Stack-RNNs

Motivation: What RNNs have



Computational Model(s)!



Deterministic Finite Automata (DFAs)

Extraction!

Spectral extraction: RNNs to WFAs

DFA extraction: Clustering

DFA and WDFA extraction: L-star variants

Analysis of Expressive Power!

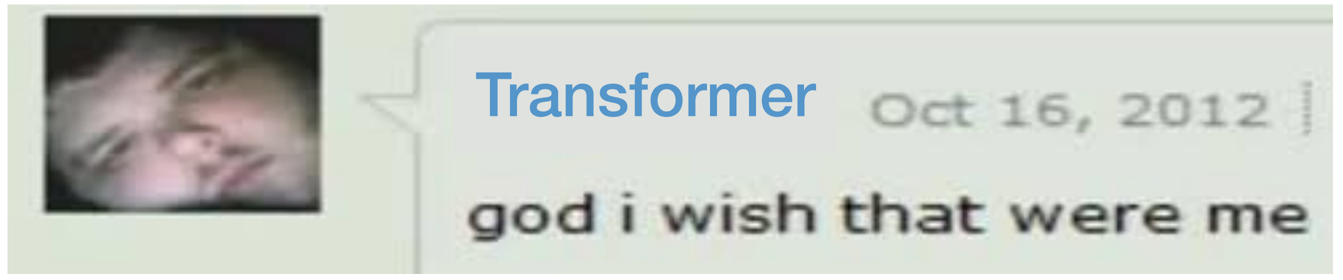
2-RNNs are WFAs

LSTMs are counter machines

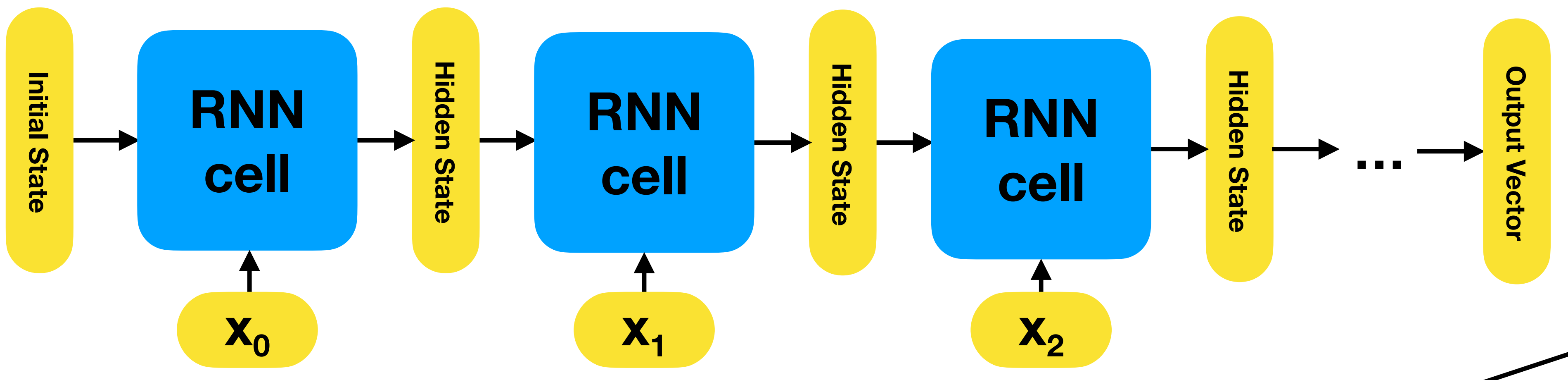
GRUs are DFAs

Inspiration from existing theory!

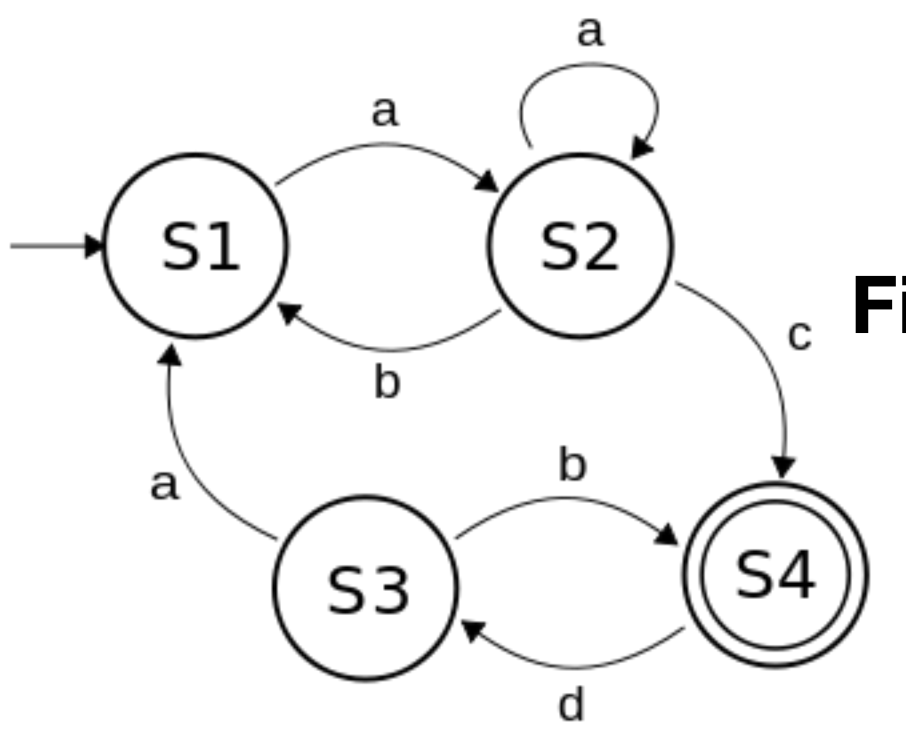
Stack-RNNs



(References for the Interested)



Computational Model(s)!



Deterministic Finite Automata (DFAs)

Extraction!

Explaining Black Boxes on Sequential Data using Weighted Automata

Extraction of Rules from Discrete-Time Recurrent Neural Networks

Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples

Analysis of Expressive Power!

Connecting Weighted Automata and Recurrent Neural Networks through Spectral Learning

On the Practical Computational Power of Finite Precision RNNs for Language Recognition

Sequential Neural Networks as Automata

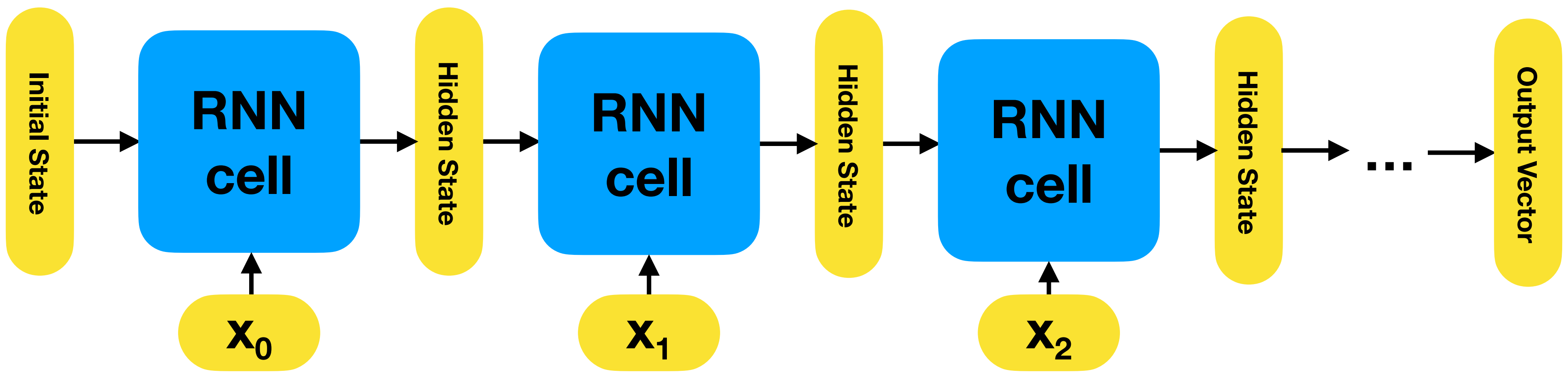
A Formal Hierarchy of RNN Architectures

Inspiration from existing theory!

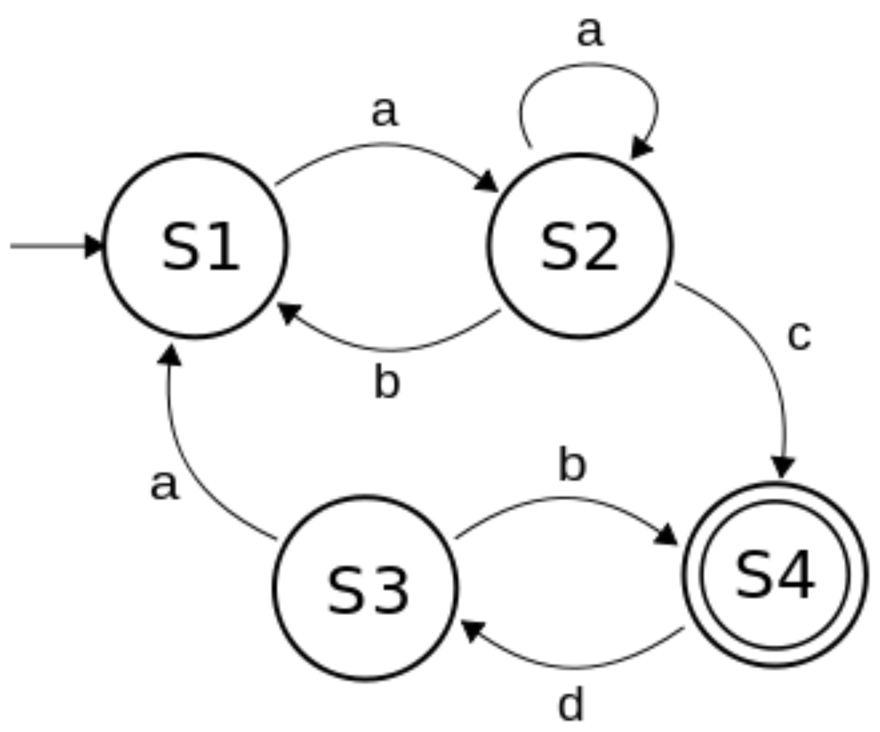
Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets

Learning to Transduce with Unbounded Memory

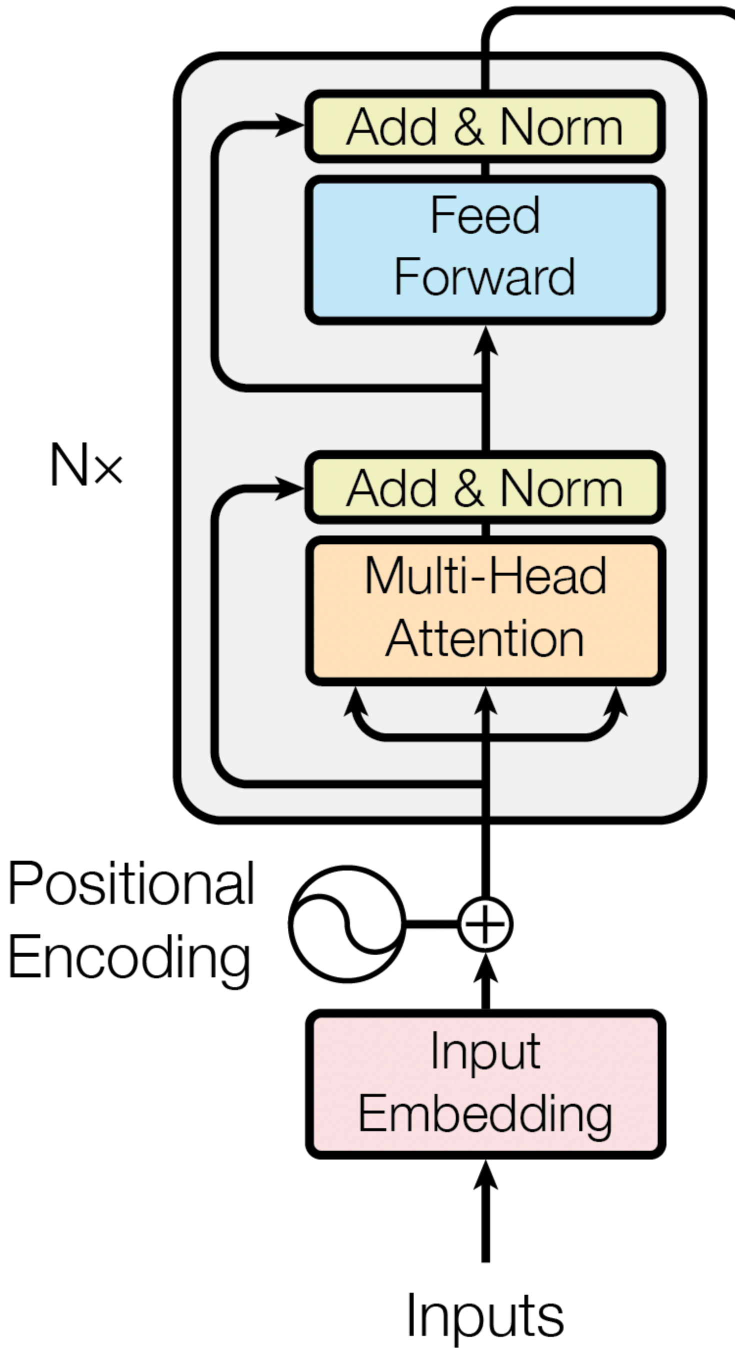
But what are Transformer-Encoders?



Computational Model(s)!



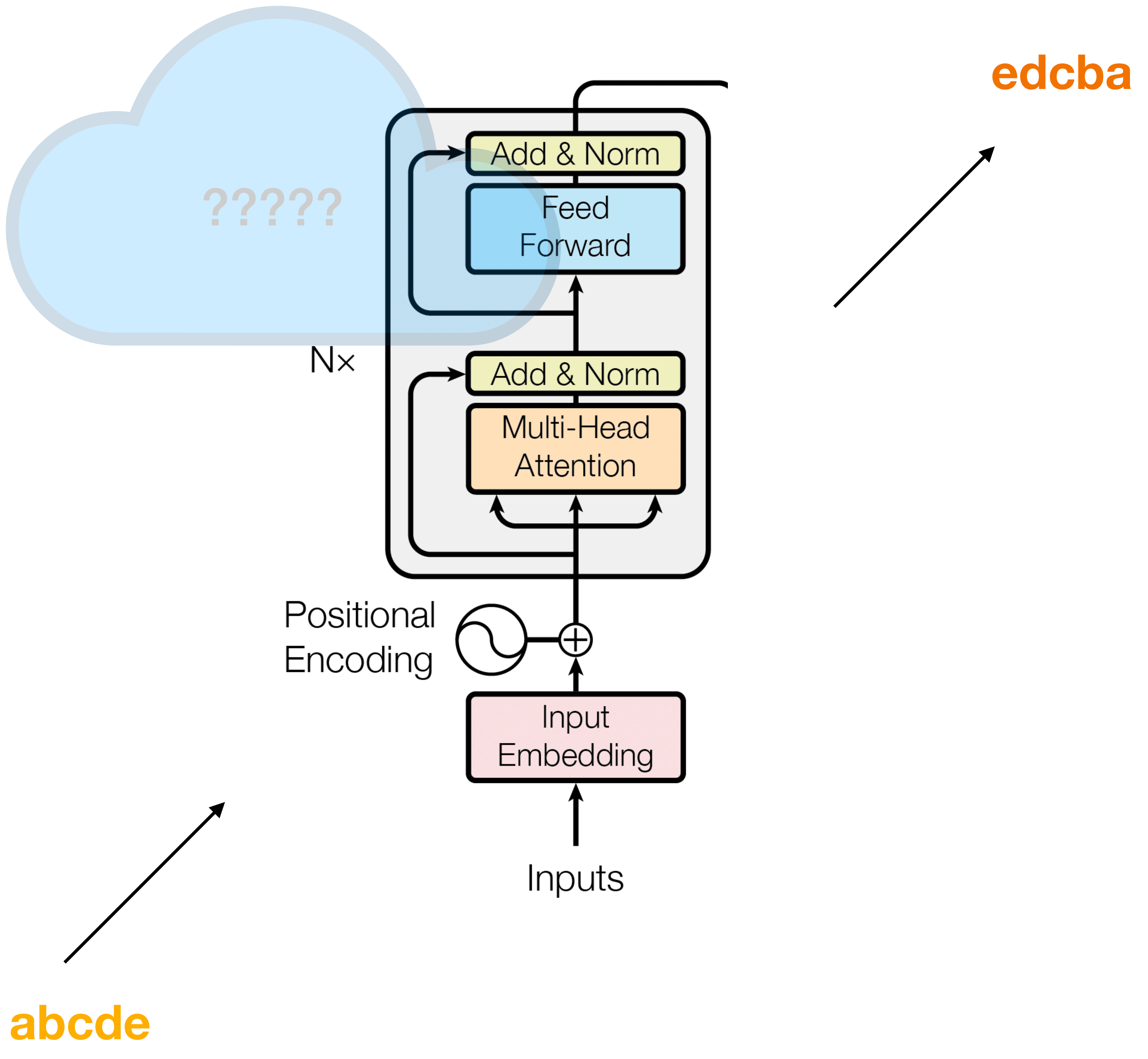
Transformer-Encoder



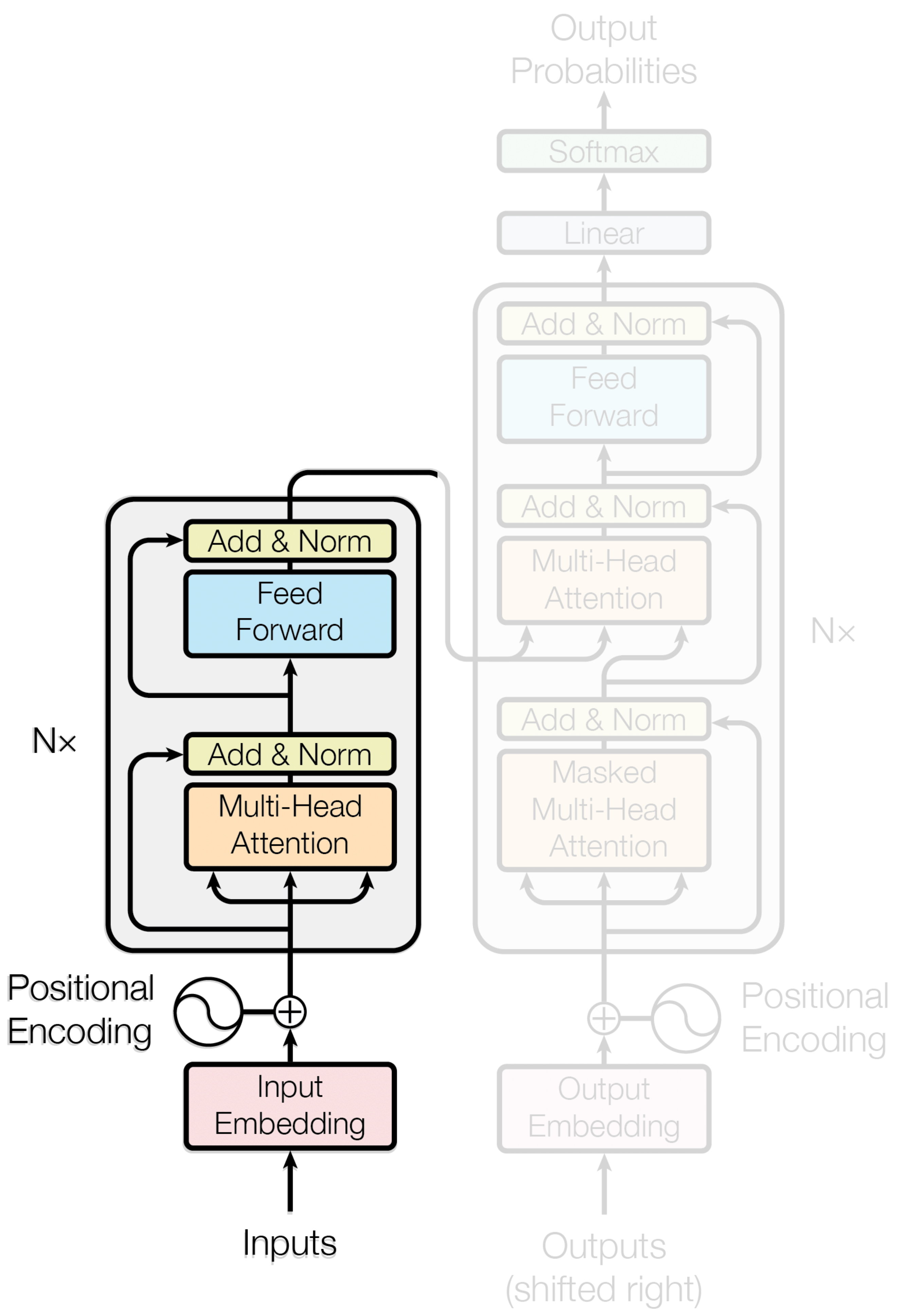
?

Any ideas?

Teaser: Reverse



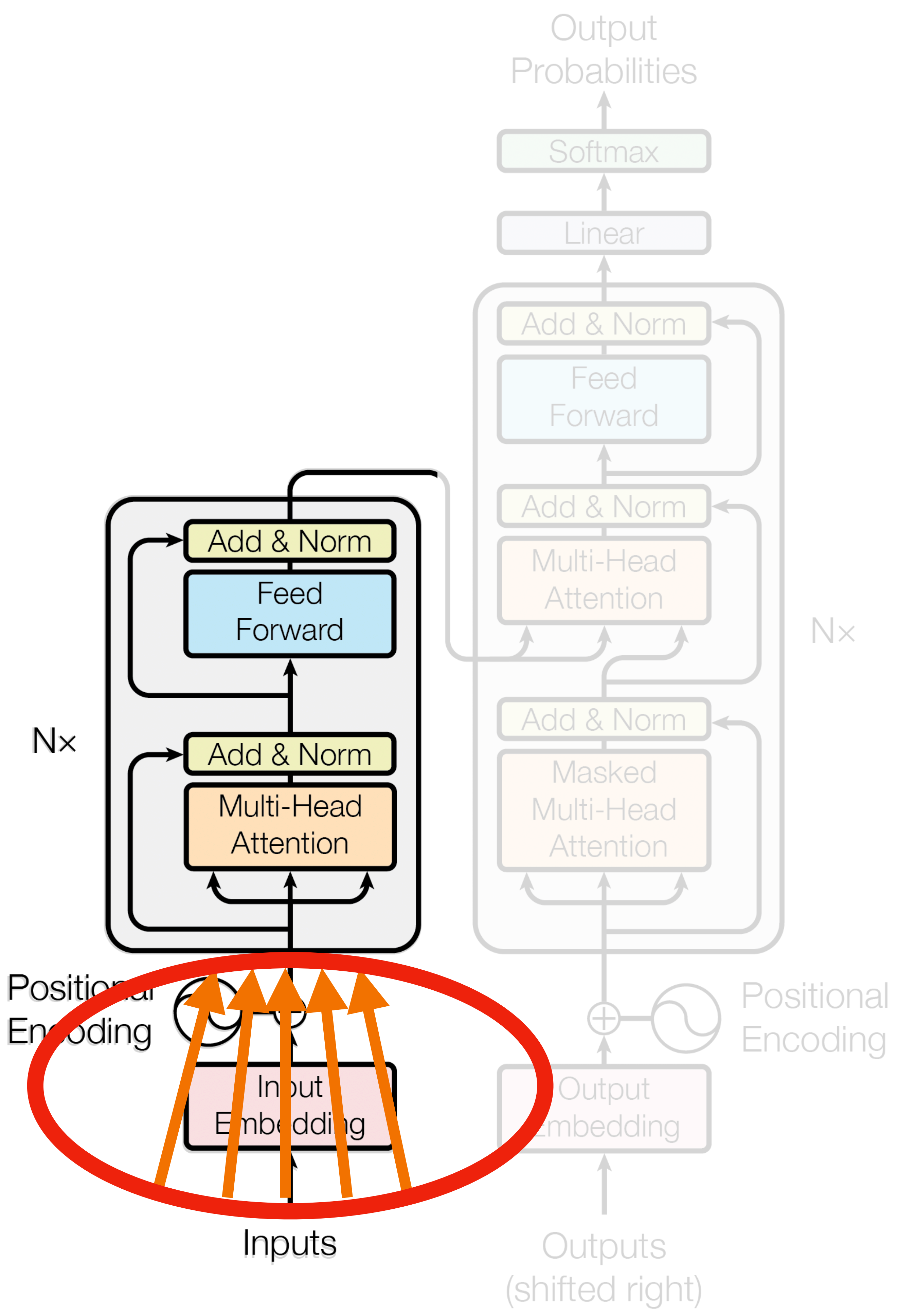
Transformer Encoders



Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

Transformer Encoders

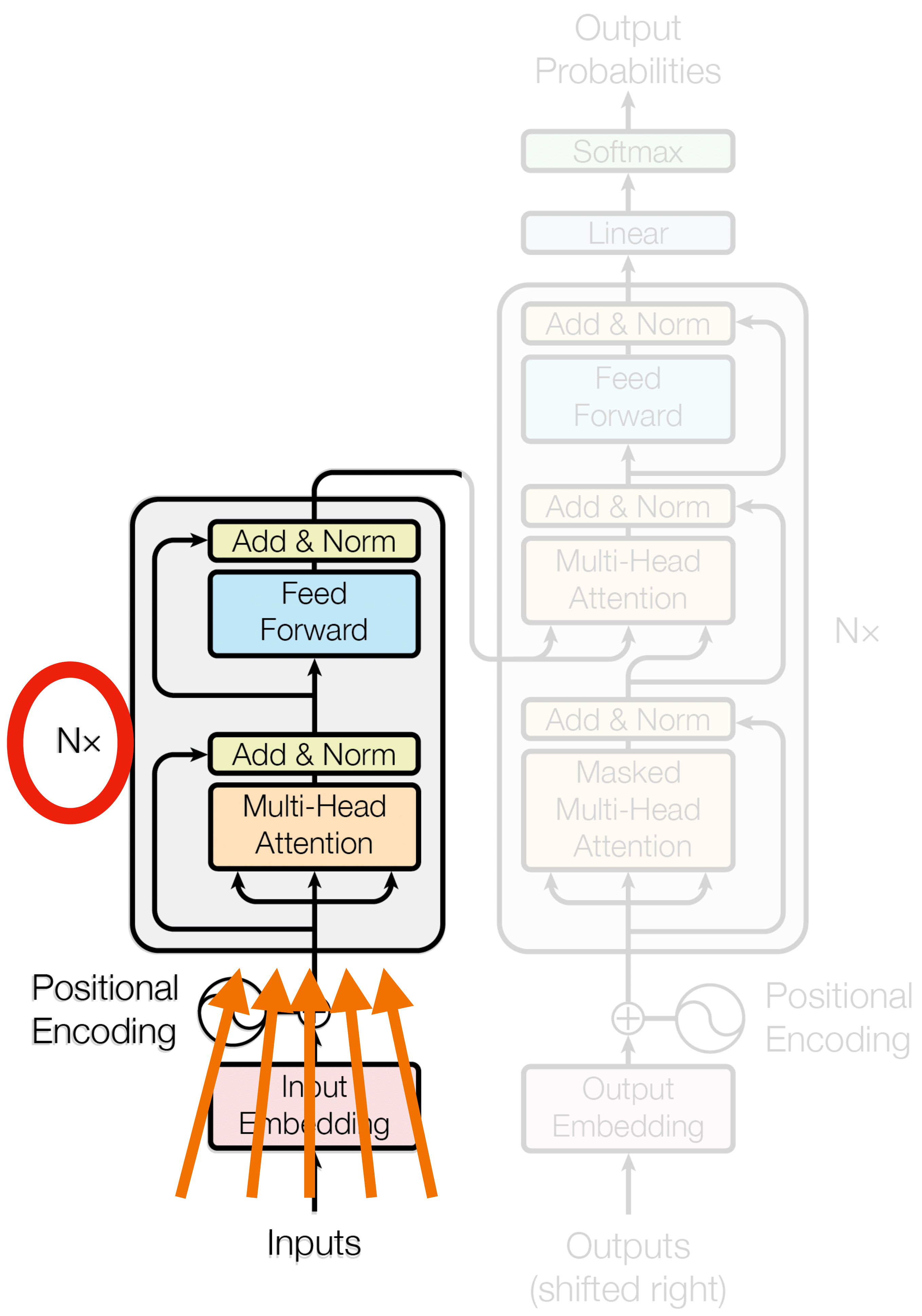


- Receive their entire input 'at once', processing all tokens in parallel

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

Transformer Encoders



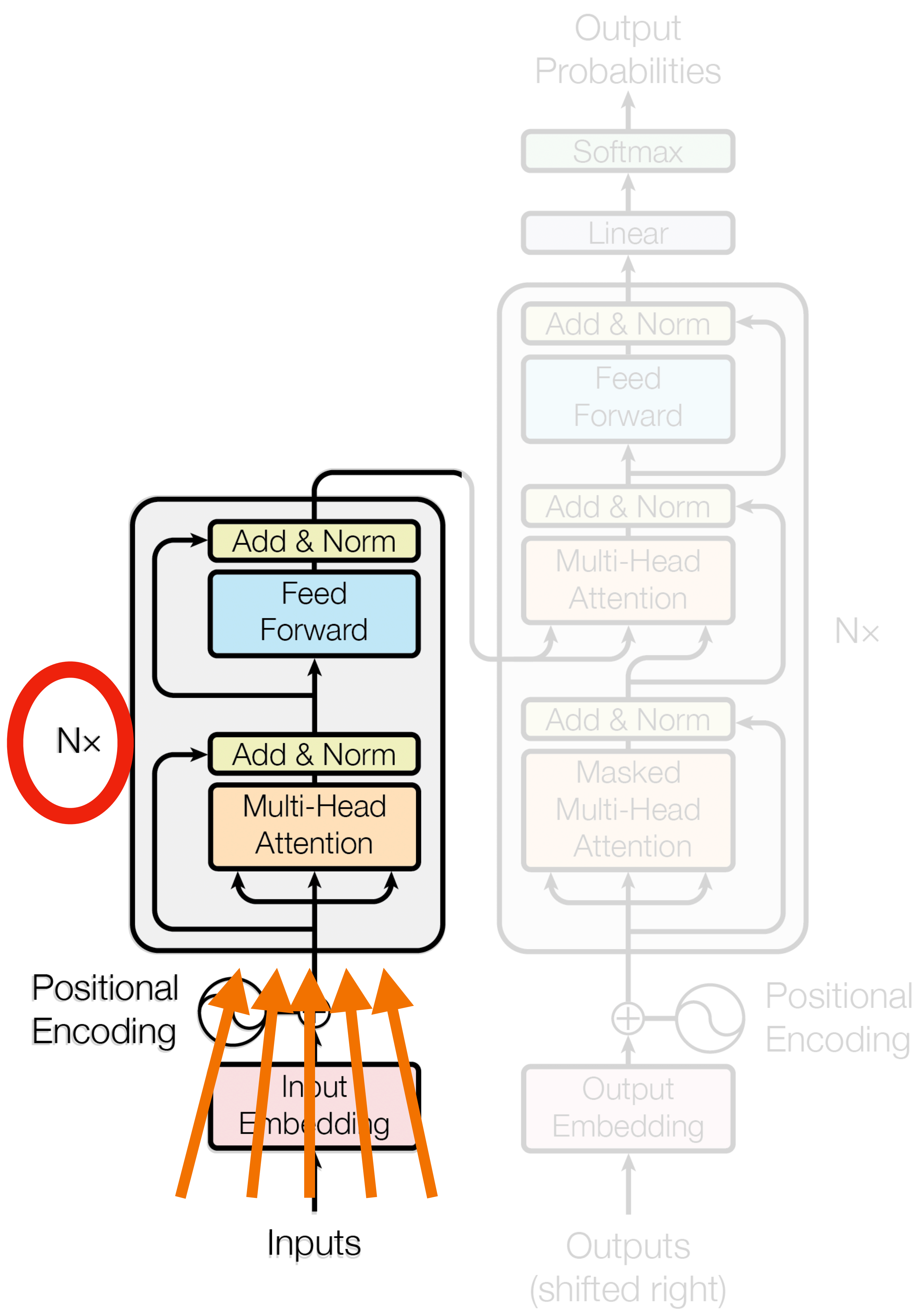
- Receive their entire input ‘at once’, processing all tokens in parallel

- Have a fixed number of layers, where the output of one is the input of the next

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

Transformer Encoders



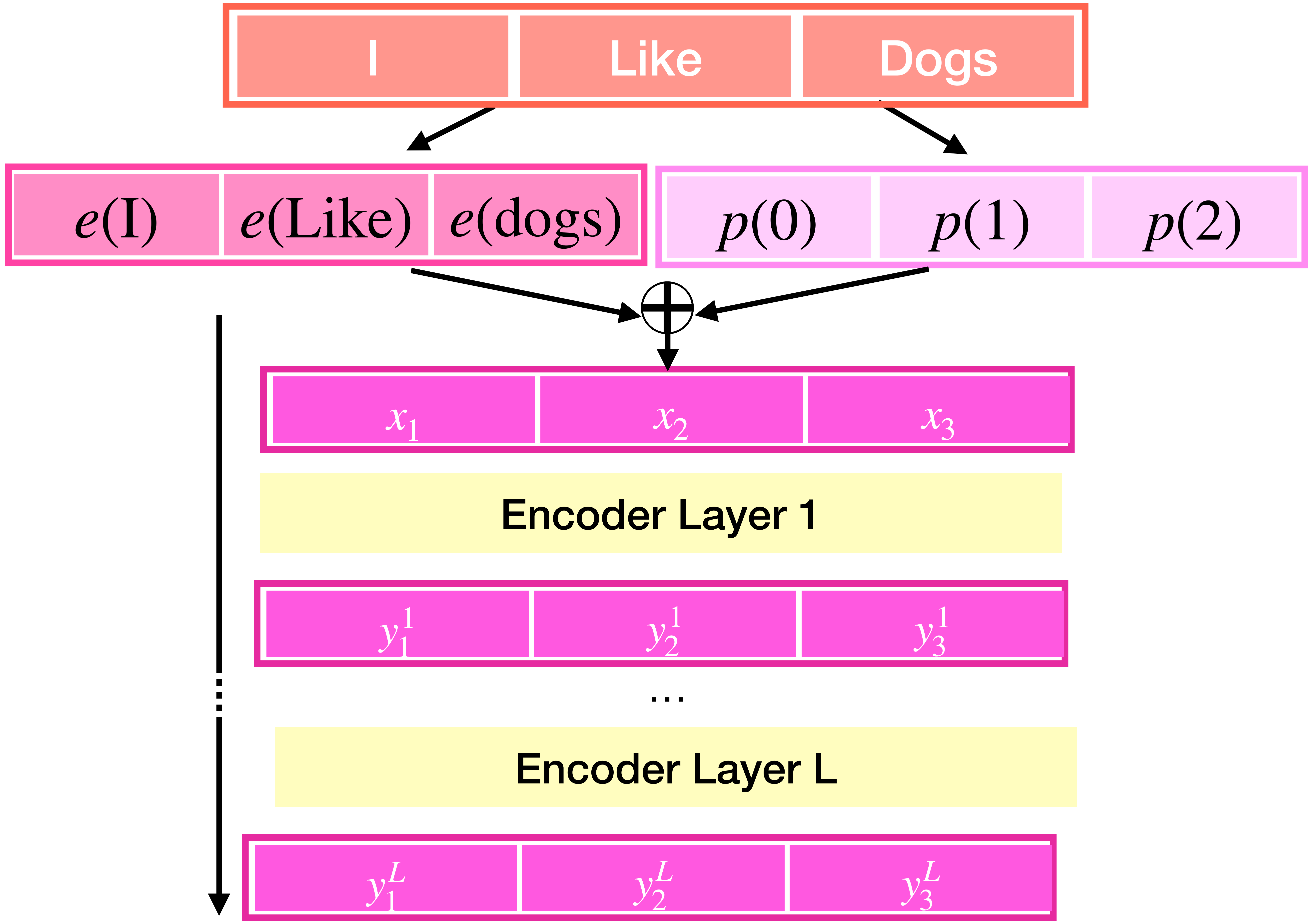
- Receive their entire input ‘at once’, processing all tokens in parallel
- Have a fixed number of layers, where the output of one is the input of the next

Computation “progresses” along network depth… not input length

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

Transformers



tokens = positionwise_embeddings(input)

indices = positionwise_indices(input)

x = tokens+indices

$$y^1 = L_1(x)$$

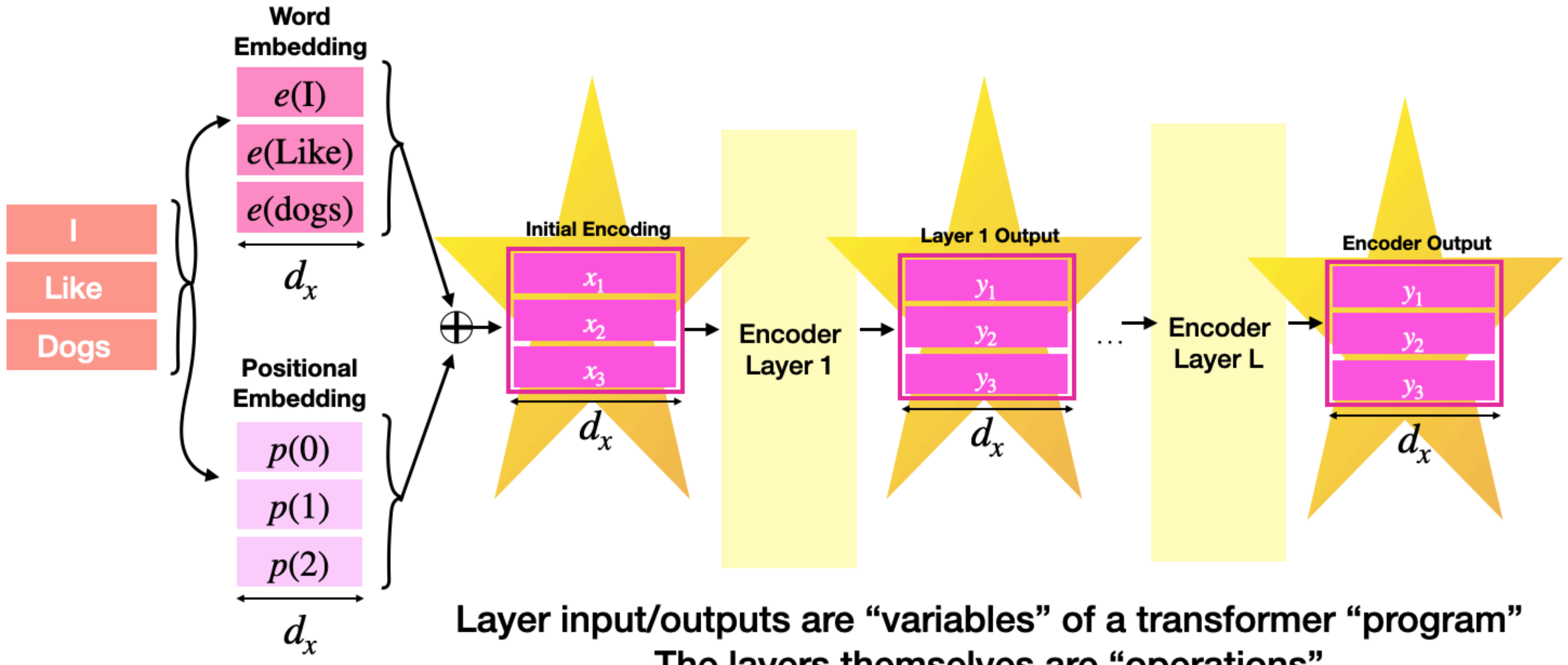
$$y^2 = L_2(y^1)$$

…

$$y = y^L = L_L\,(y^{L-1})$$

Layer input/outputs are "variables" of a transformer "program"
The layers themselves are "operations"

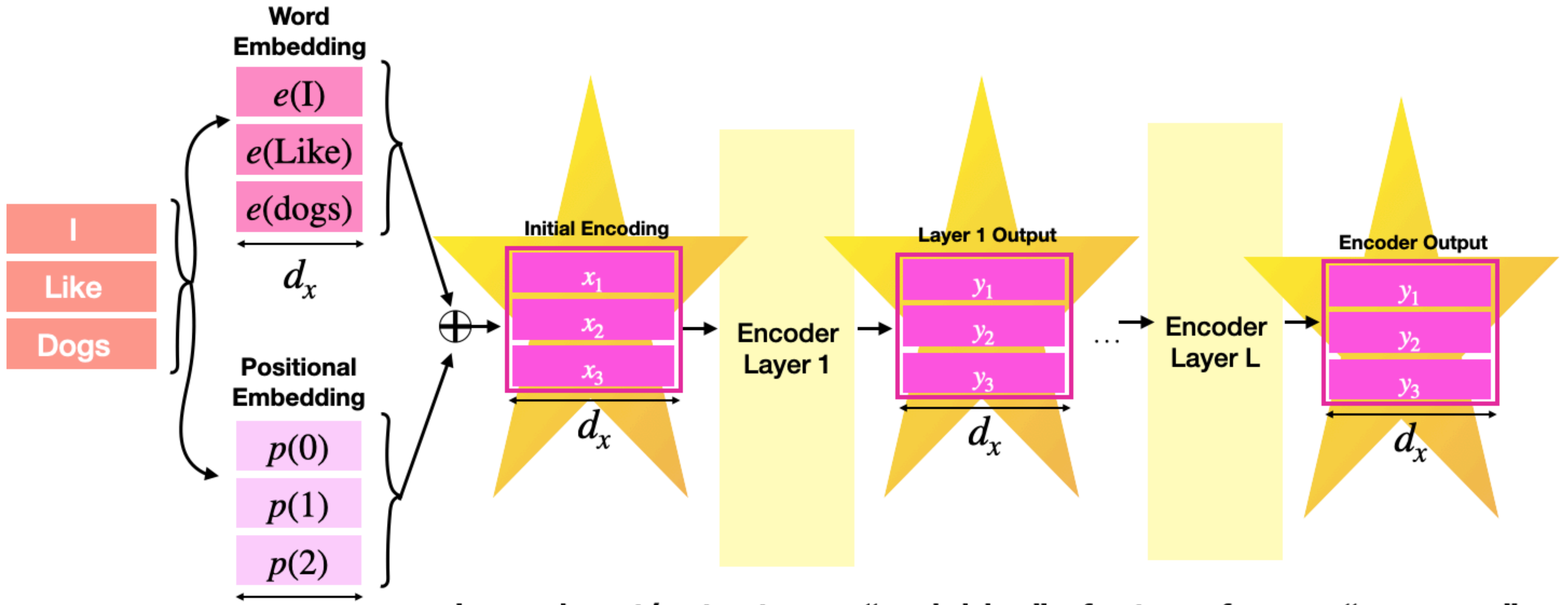
RASP (Restricted Access Sequence Processing)



Layer input/outputs are “variables” of a transformer “program”
The layers themselves are “operations”

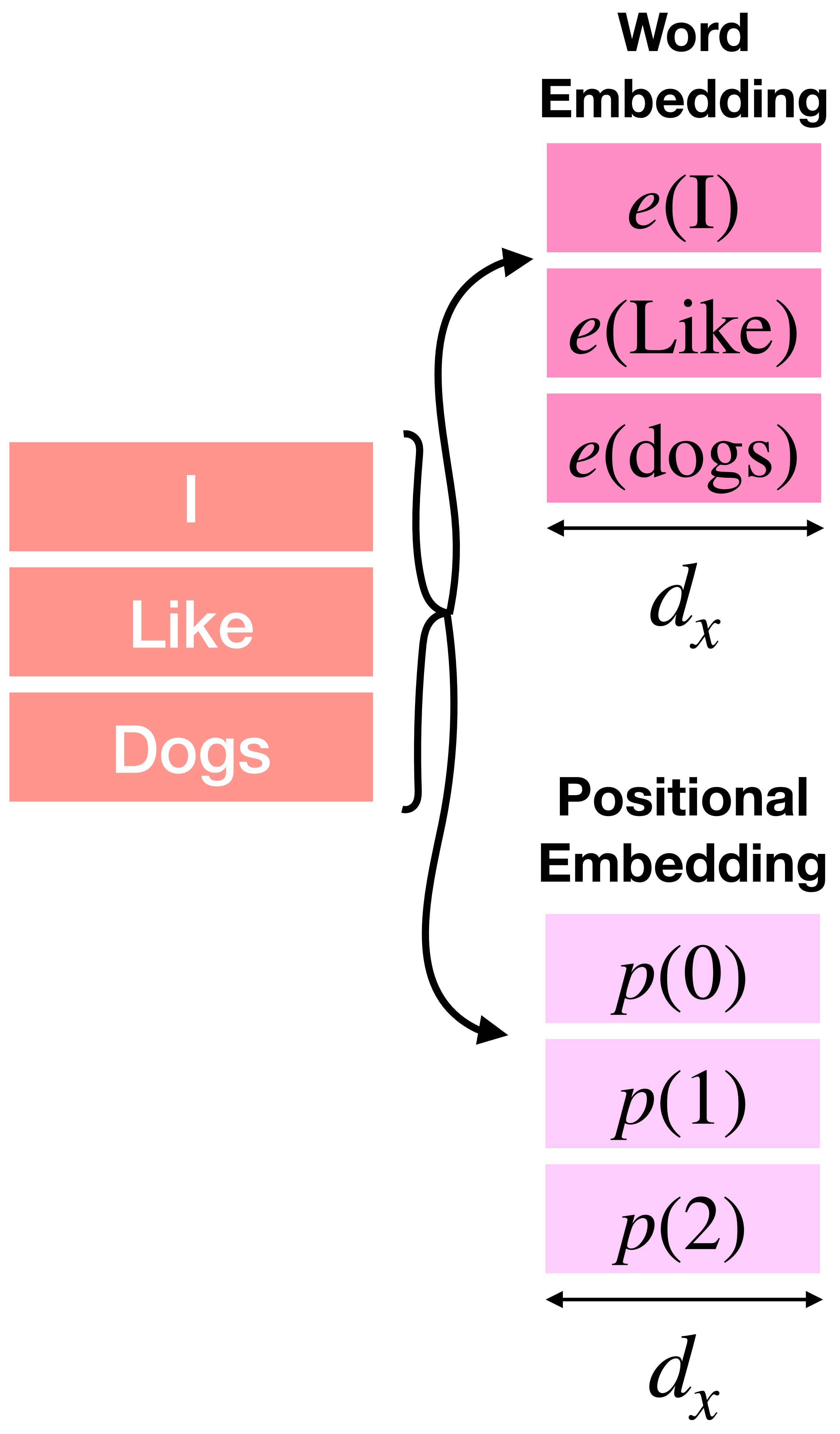
RASP (Restricted Access Sequence Processing)

- A transformer-encoder is a sequence to sequence function (“sequence operator”, or, “**s-op**”)
- Its **layers apply operations** to the sequences
- **RASP builds s-ops**, constrained to a transformer’s inputs and possible operations
  - (The s-ops are the transformer abstractions!)



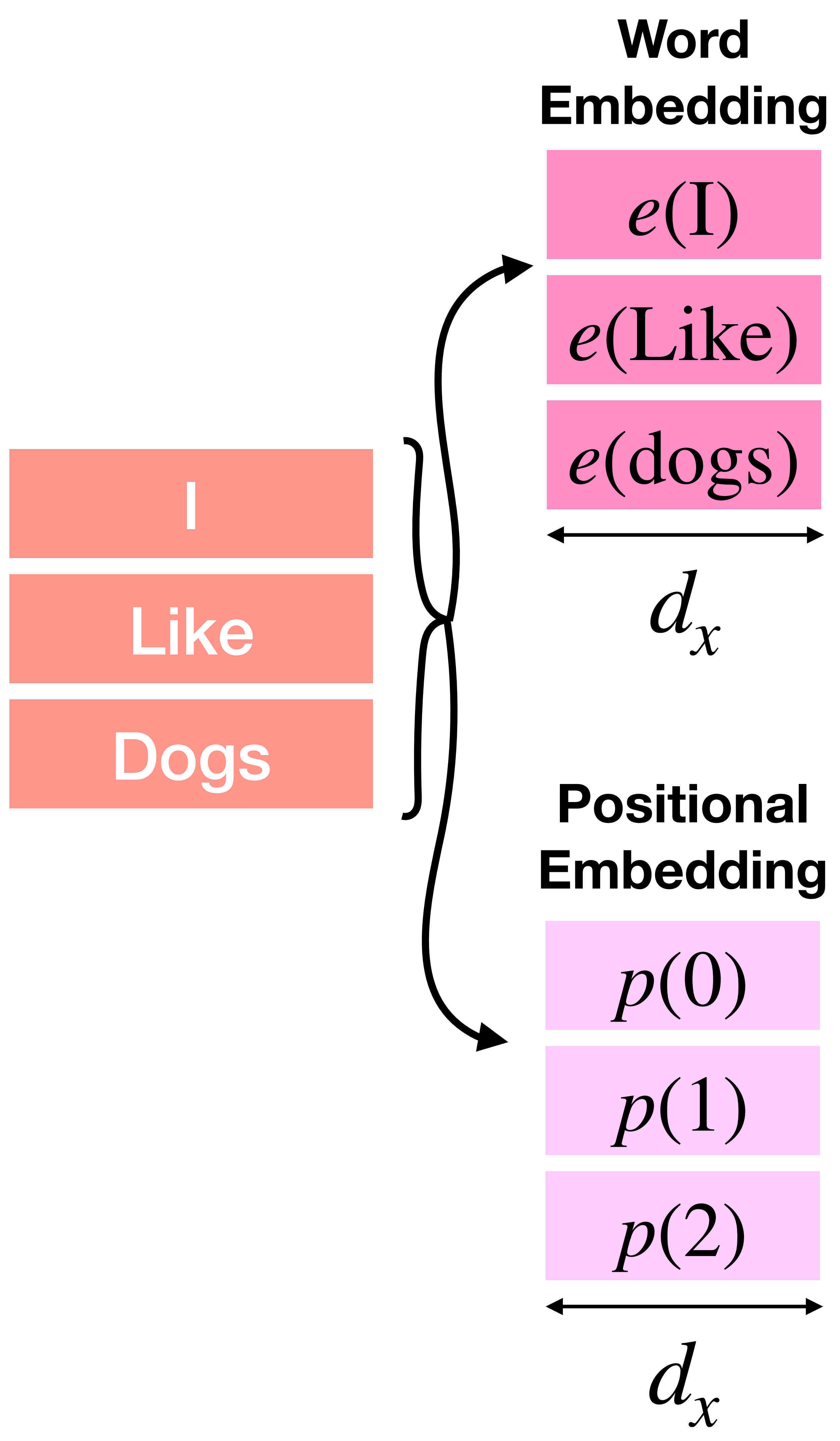
Layer input/outputs are “variables” of a transformer “program”
The layers themselves are “operations”

RASP base s-ops



The information before a transformer has done anything ("0 layer transformer")

RASP base s-ops



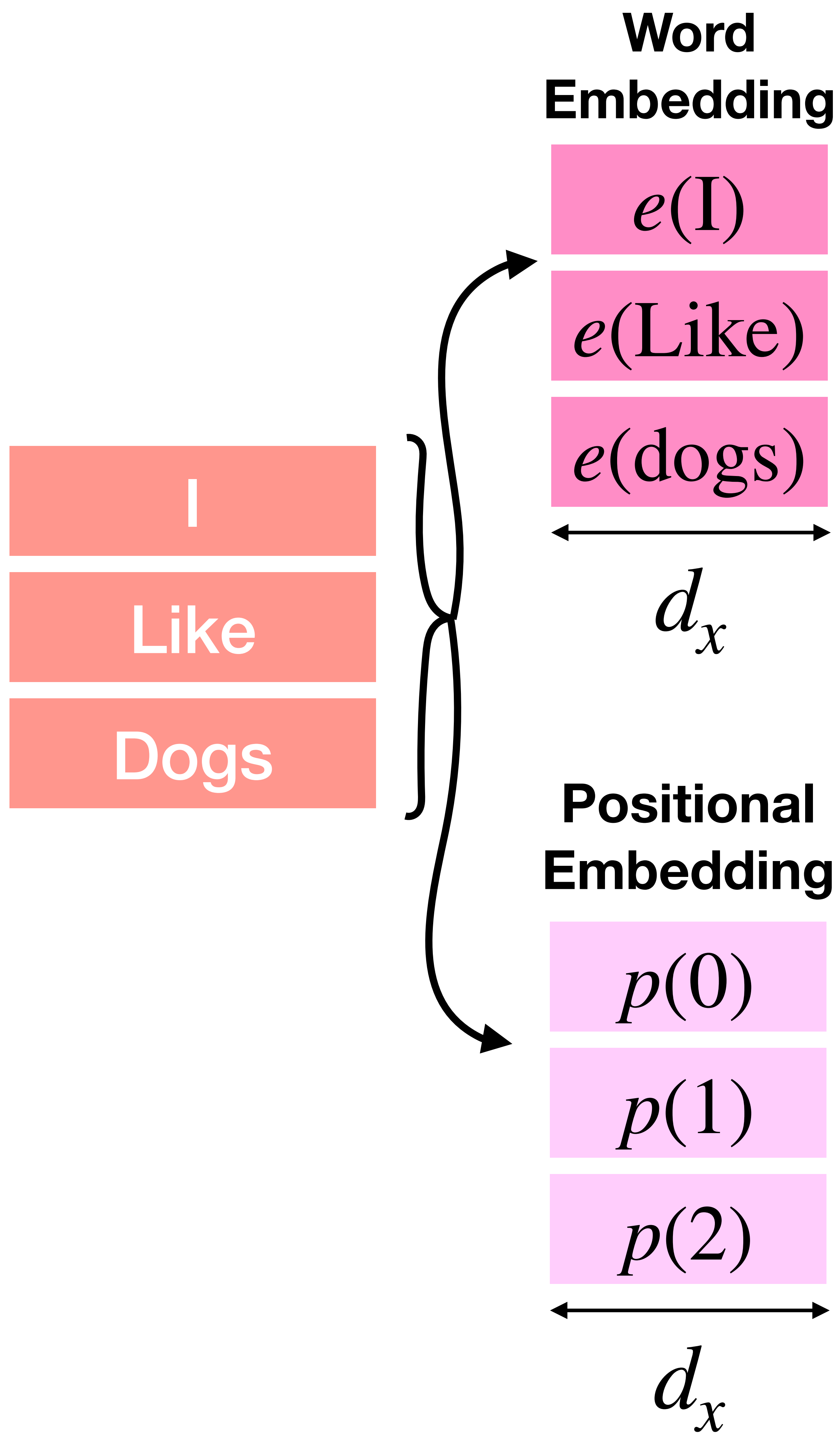
The information before a transformer has done anything ("0 layer transformer")

tokens and *indices* are RASP built-ins:

```
>> tokens;
     s-op: tokens

>> indices;
     s-op: indices
```

RASP base s-ops



The information before a transformer has done anything (“0 layer transformer”)

tokens and *indices* are RASP built-ins:

```
>> tokens;
     s-op: tokens
         Example: tokens("hello") = [h, e, l, l, o] (strings)
>> indices;
     s-op: indices
         Example: indices("hello") = [0, 1, 2, 3, 4] (ints)
```

The RASP REPL gives you examples (until you ask it not to)

Okay, now what?

```
>> tokens;
     s-op: tokens
         Example: tokens("hello") = [h, e, l, l, o] (strings)
>> indices;
     s-op: indices
         Example: indices("hello") = [0, 1, 2, 3, 4] (ints)
```

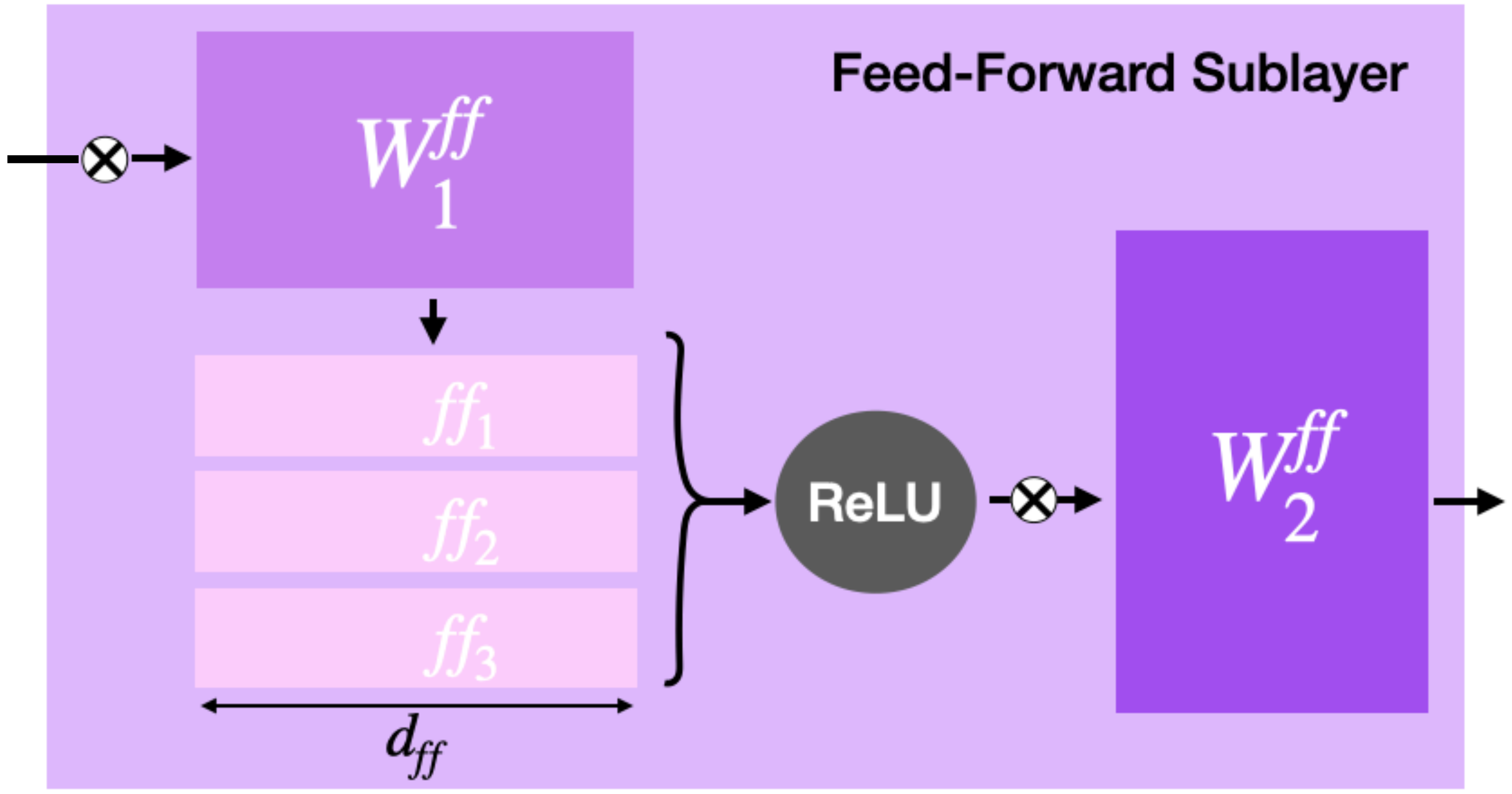
To know what operations RASP may have, we must inspect the transformer-encoder layers!

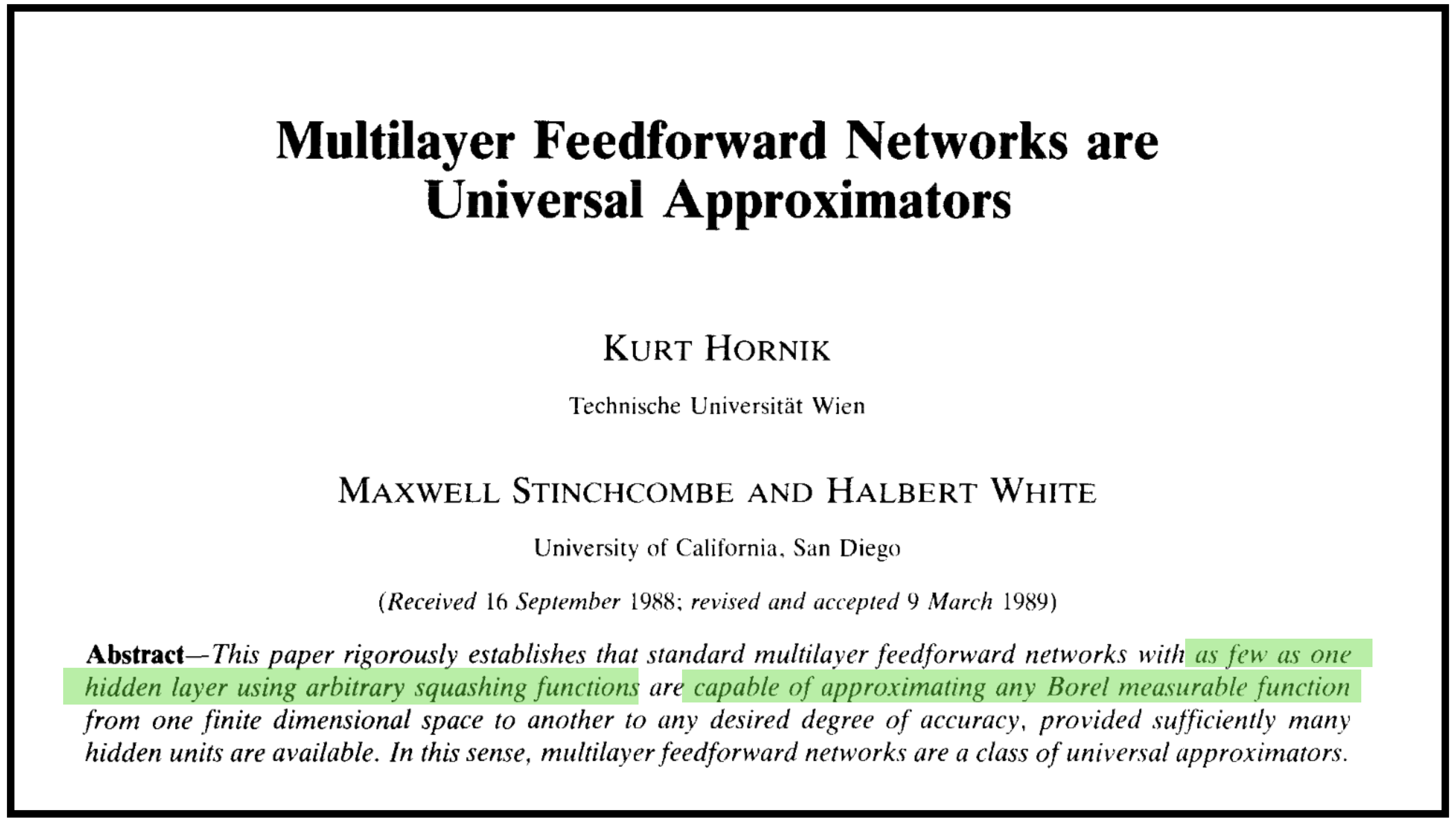
Transformer-Encoder Layer

Feed-Forward Sublayer

Feed-Forward Sublayer

Feed-Forward gives us (Many) Elementwise Operations





Multilayer Feedforward Networks are Universal Approximators

KURT HORNIK
Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBERT WHITE
University of California, San Diego

(*Received 16 September 1988; revised and accepted 9 March 1989*)

Abstract—*This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.*

```
>> indices+1;
     s-op: out
         Example: out("hello") = [1, 2, 3, 4, 5] (ints)
>> tokens=="e" or tokens=="o";
     s-op: out
         Example: out("hello") = [F, T, F, F, T] (bools)
```

So far

```
>> tokens;
     s-op: tokens
         Example: tokens("hello") = [h, e, l, l, o] (strings)
>> indices;
     s-op: indices
         Example: indices("hello") = [0, 1, 2, 3, 4] (ints)


>> indices+1;
     s-op: out
         Example: out("hello") = [1, 2, 3, 4, 5] (ints)
>> tokens=="e" or tokens=="o";
     s-op: out
         Example: out("hello") = [F, T, F, F, T] (bools)
```

Are we all-powerful (well, transformer-powerful) yet?

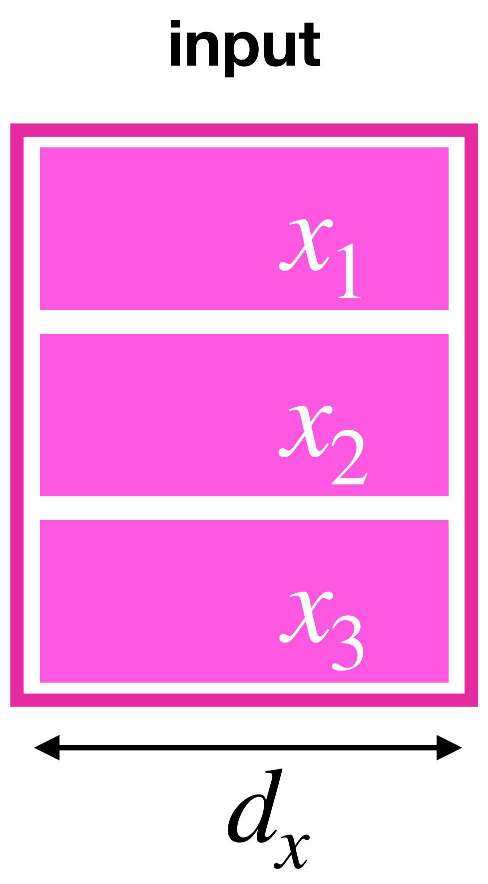
Transformer-Encoder Layer

Attention Sublayer

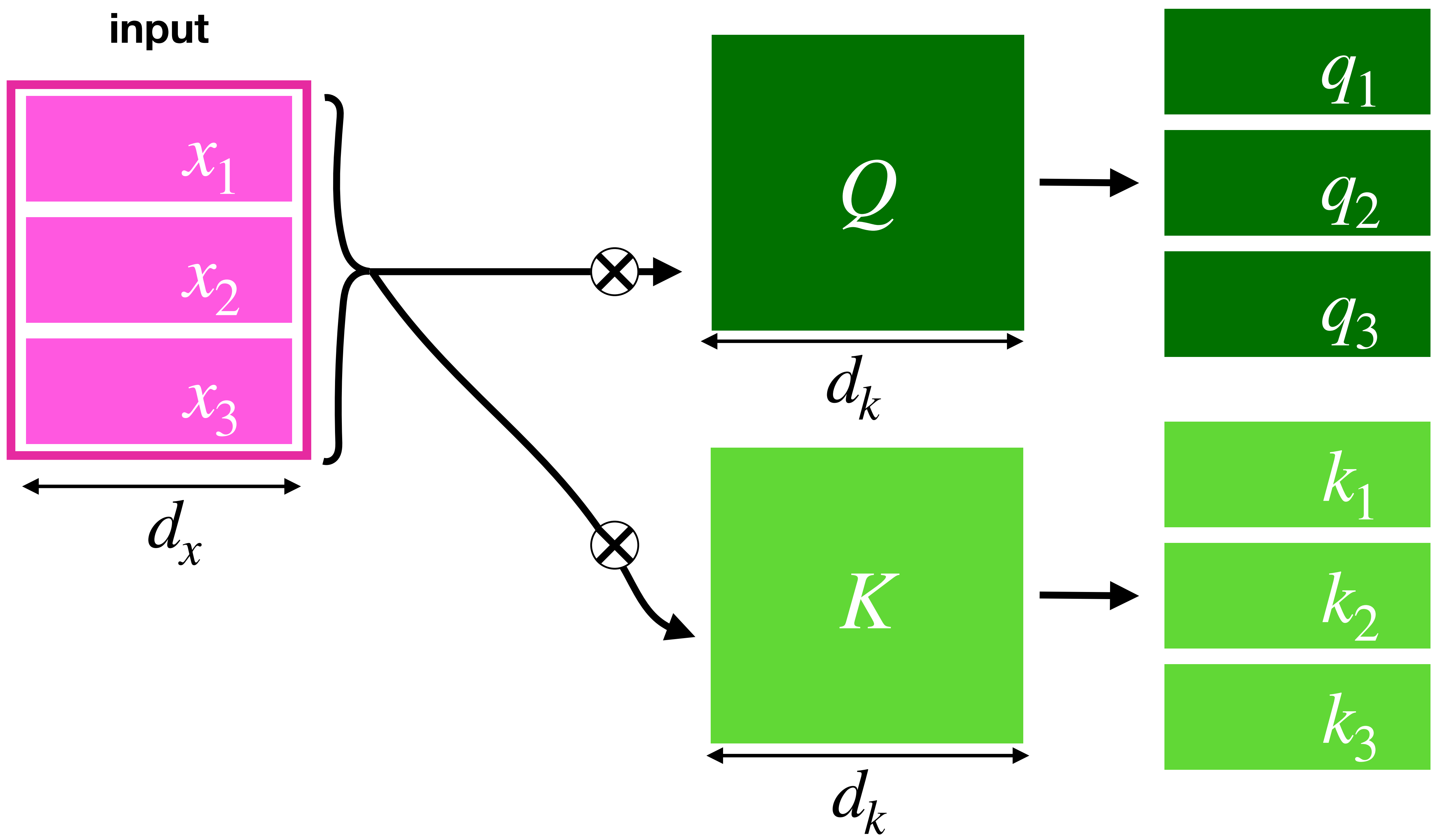
Background - Multi Head Attention

Starting from single-head attention…

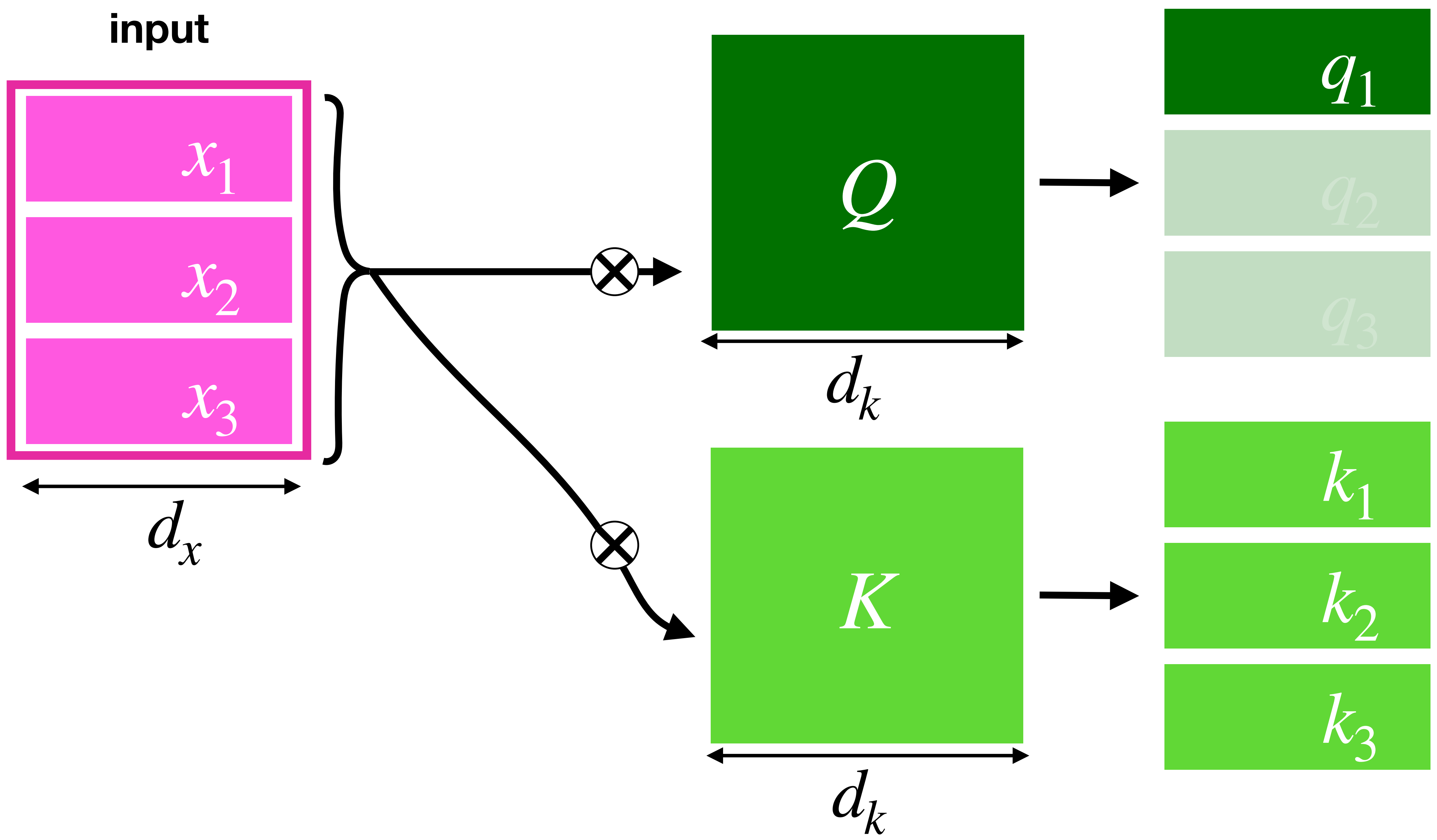
Background - Self Attention (Single Head)



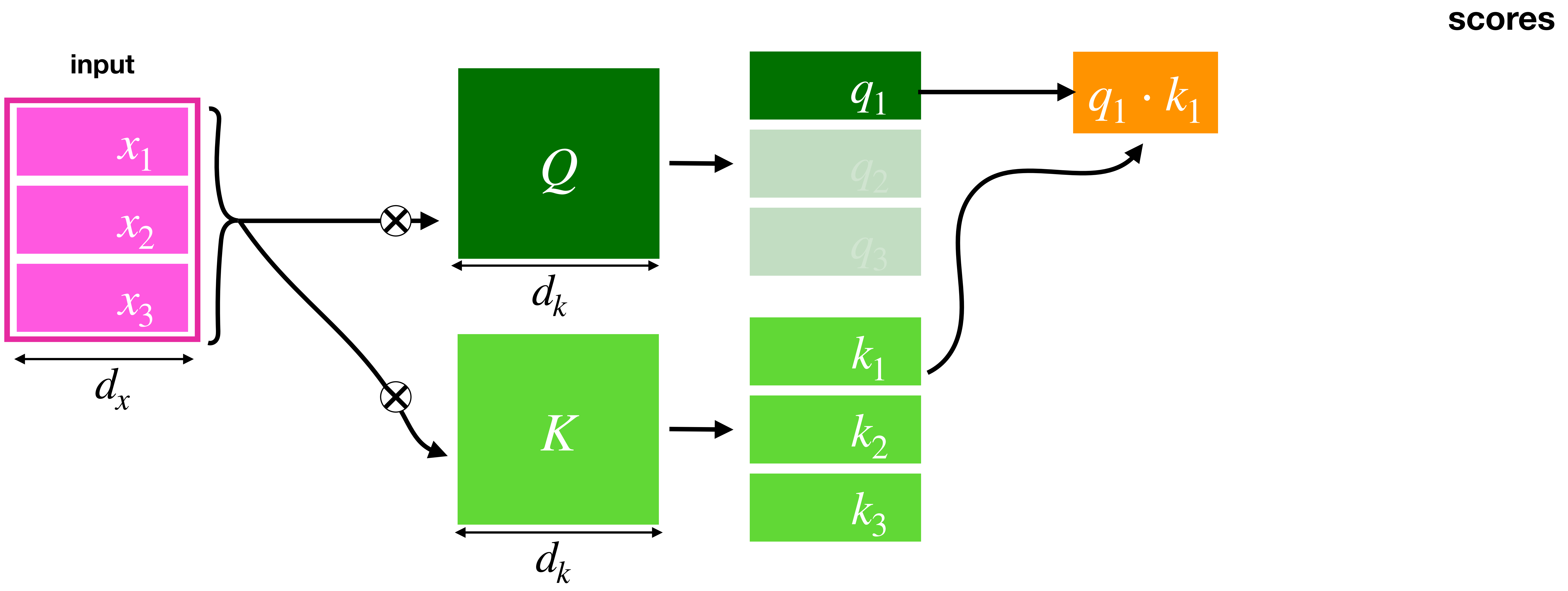
Background - Self Attention (Single Head)



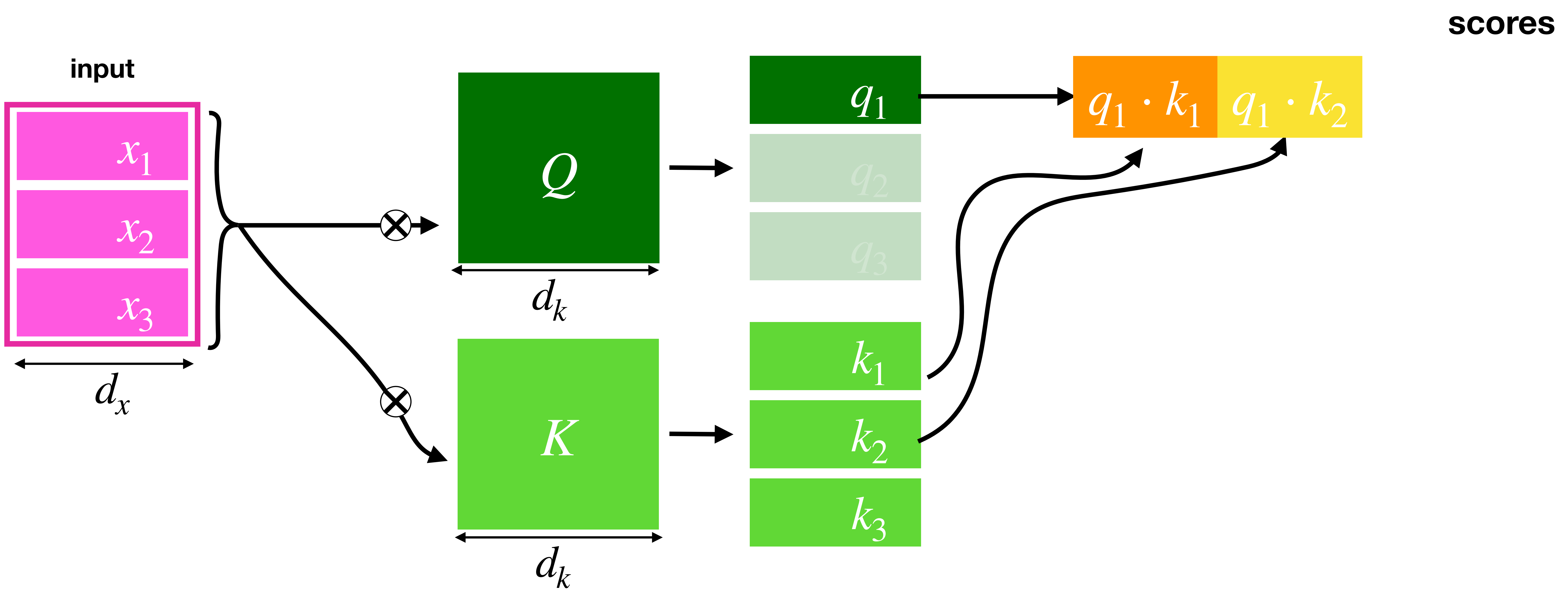
Background - Self Attention (Single Head)



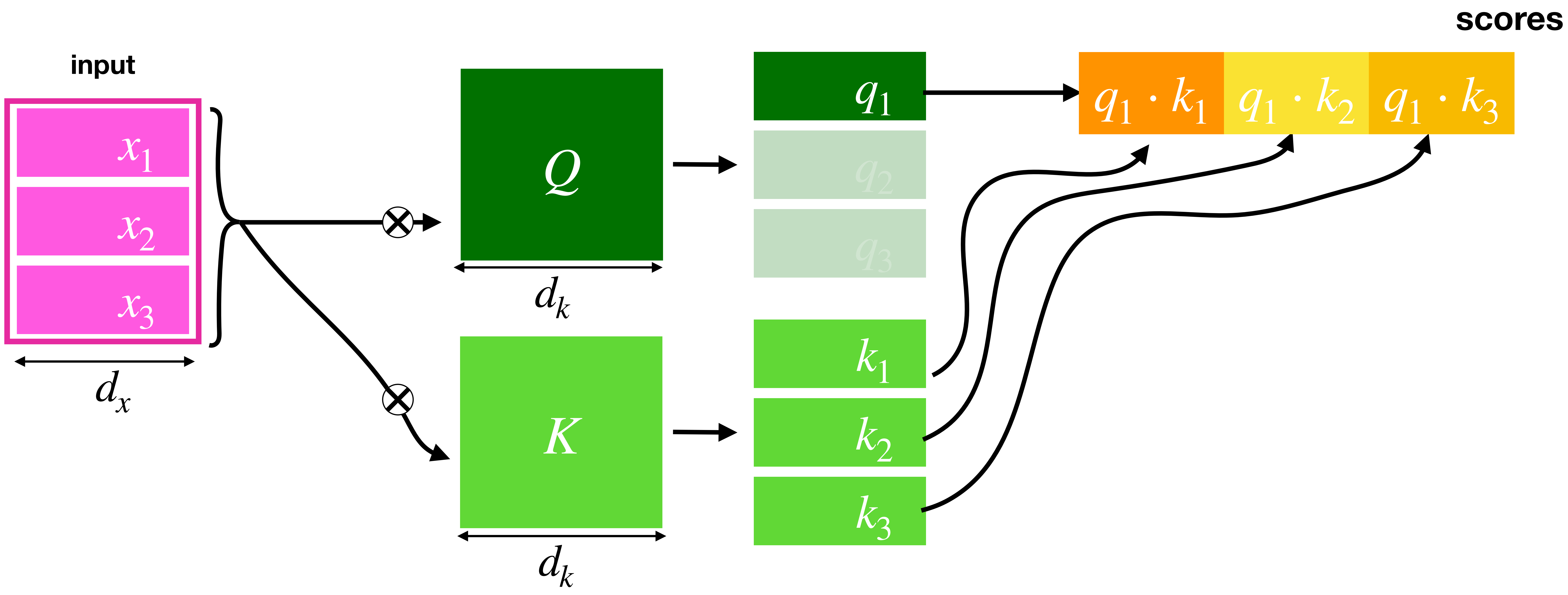
Background - Self Attention (Single Head)



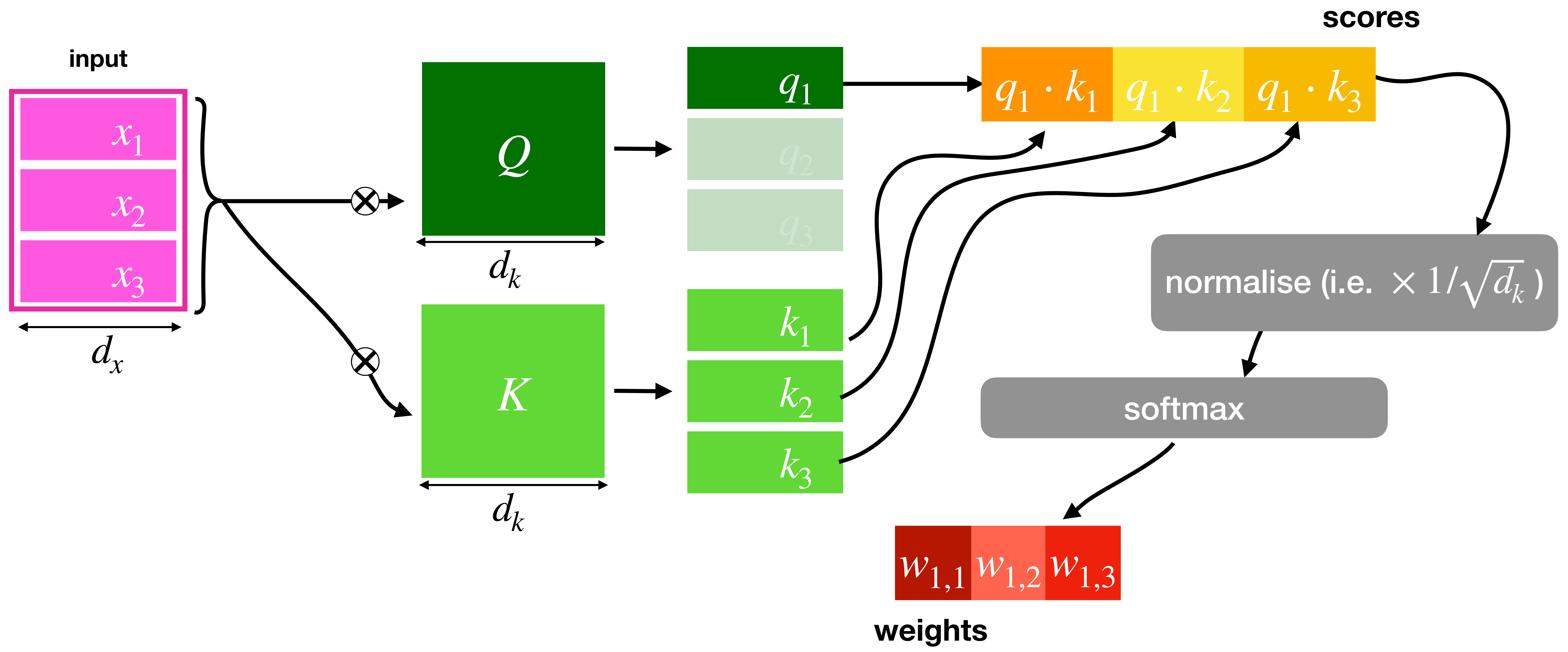
Background - Self Attention (Single Head)



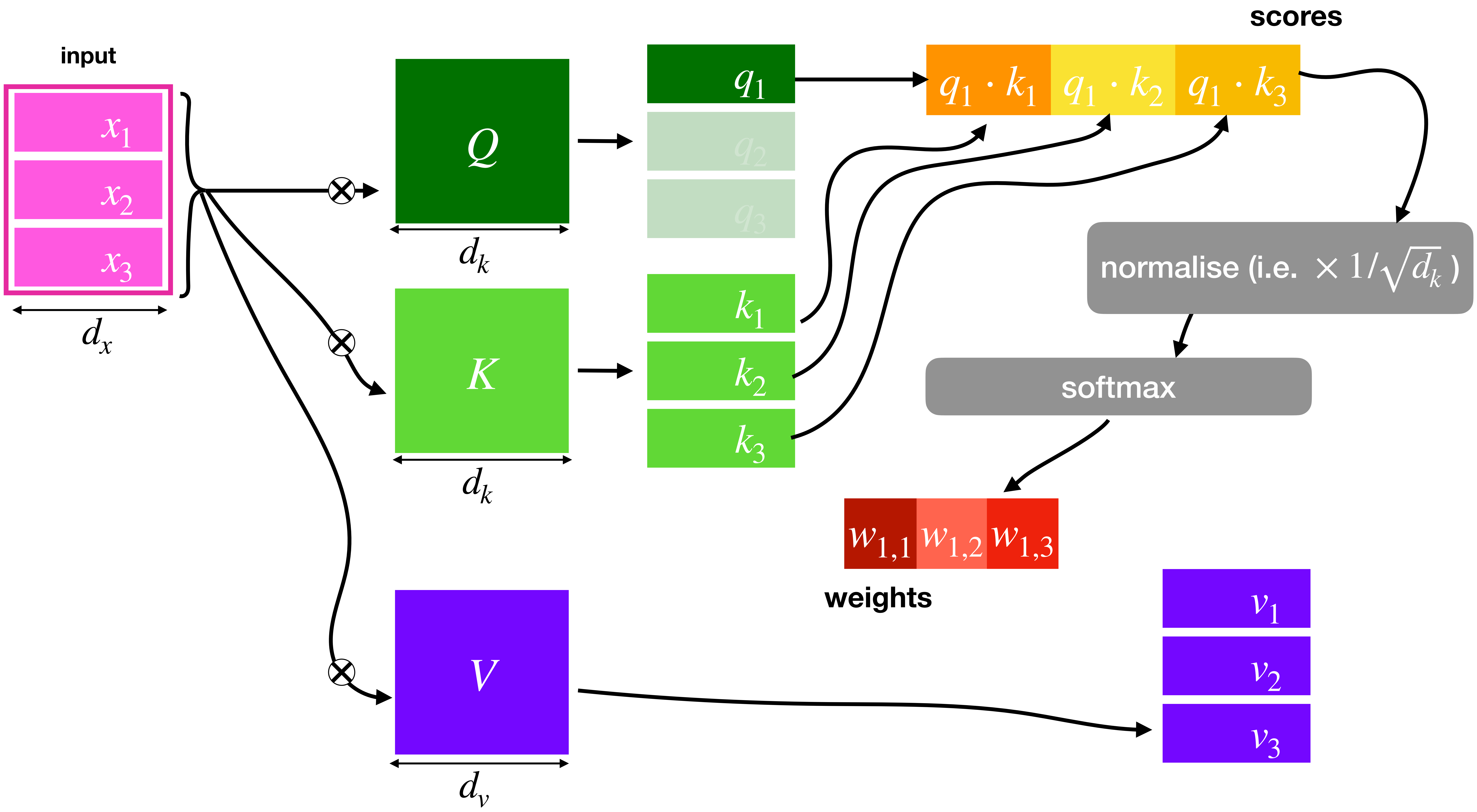
Background - Self Attention (Single Head)



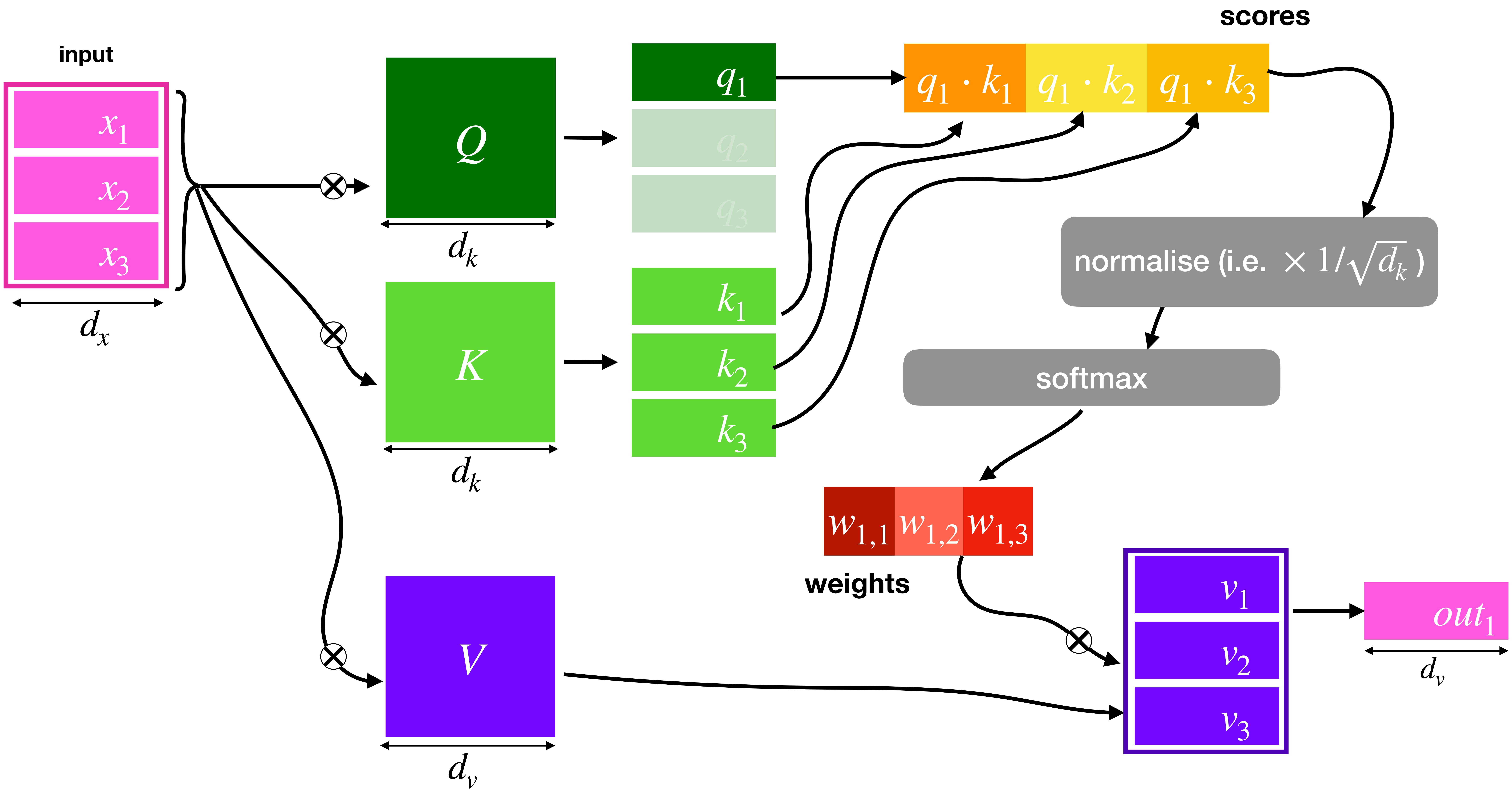
Background - Self Attention (Single Head)



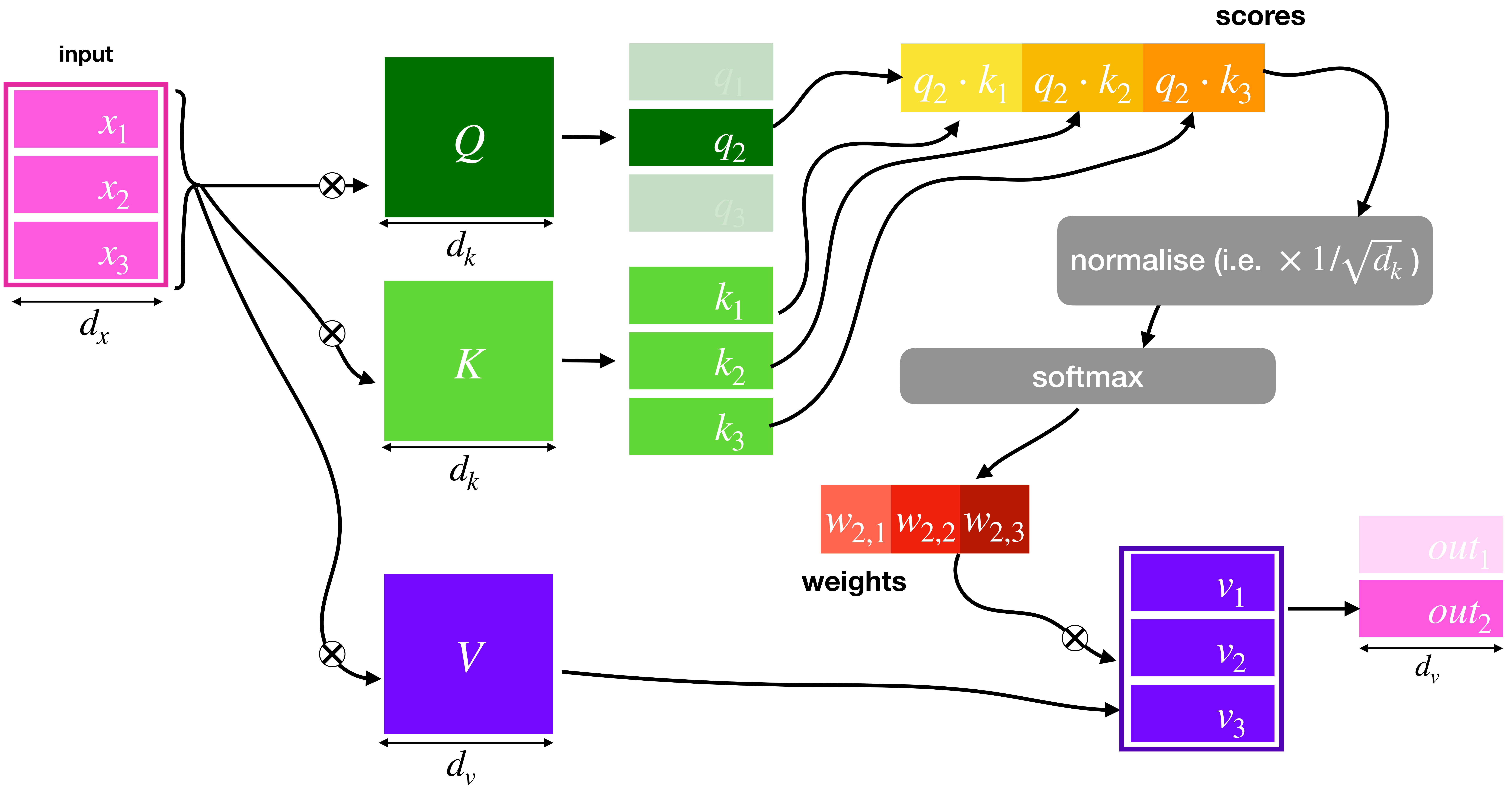
Background - Self Attention (Single Head)



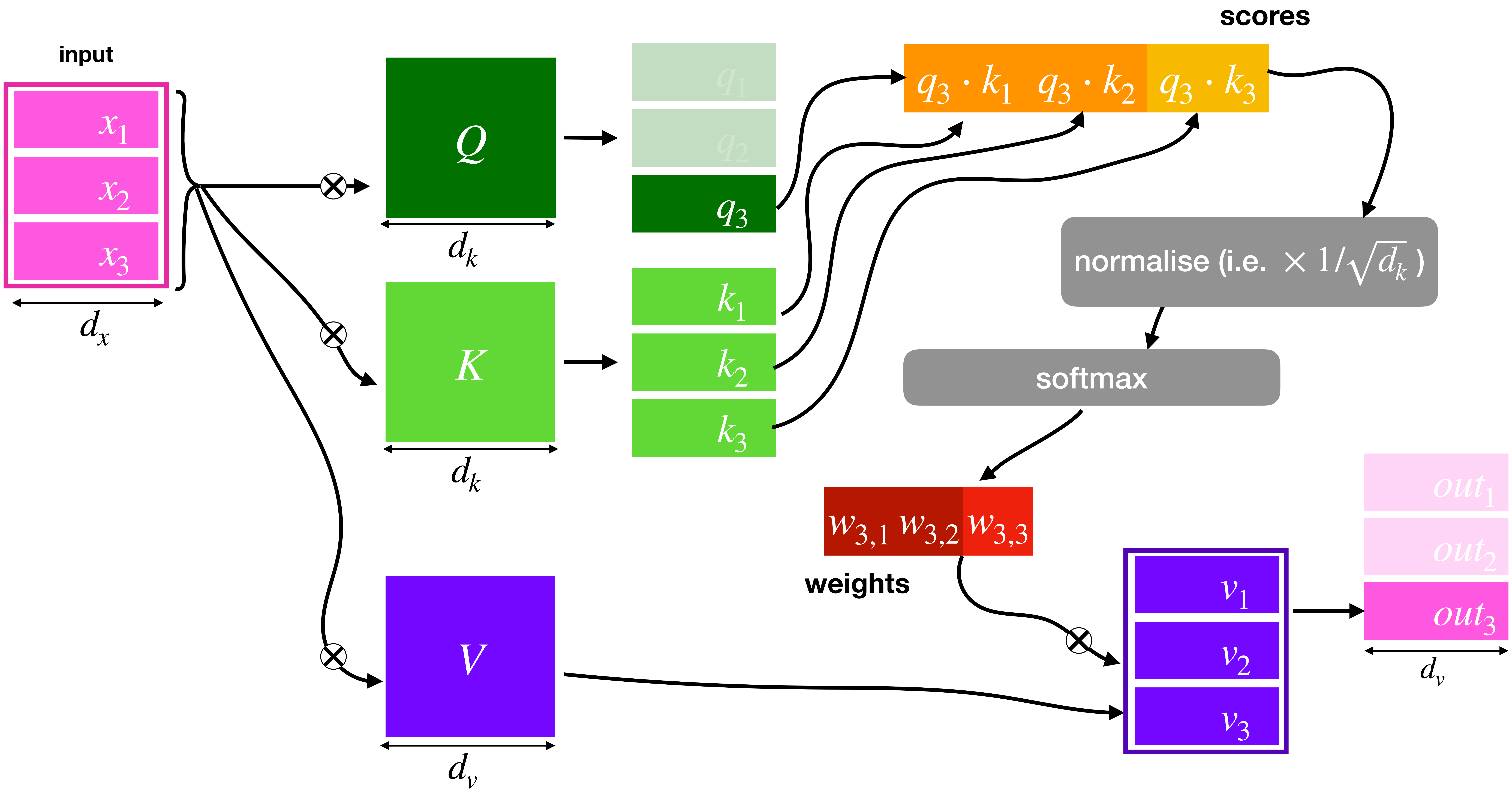
Background - Self Attention (Single Head)



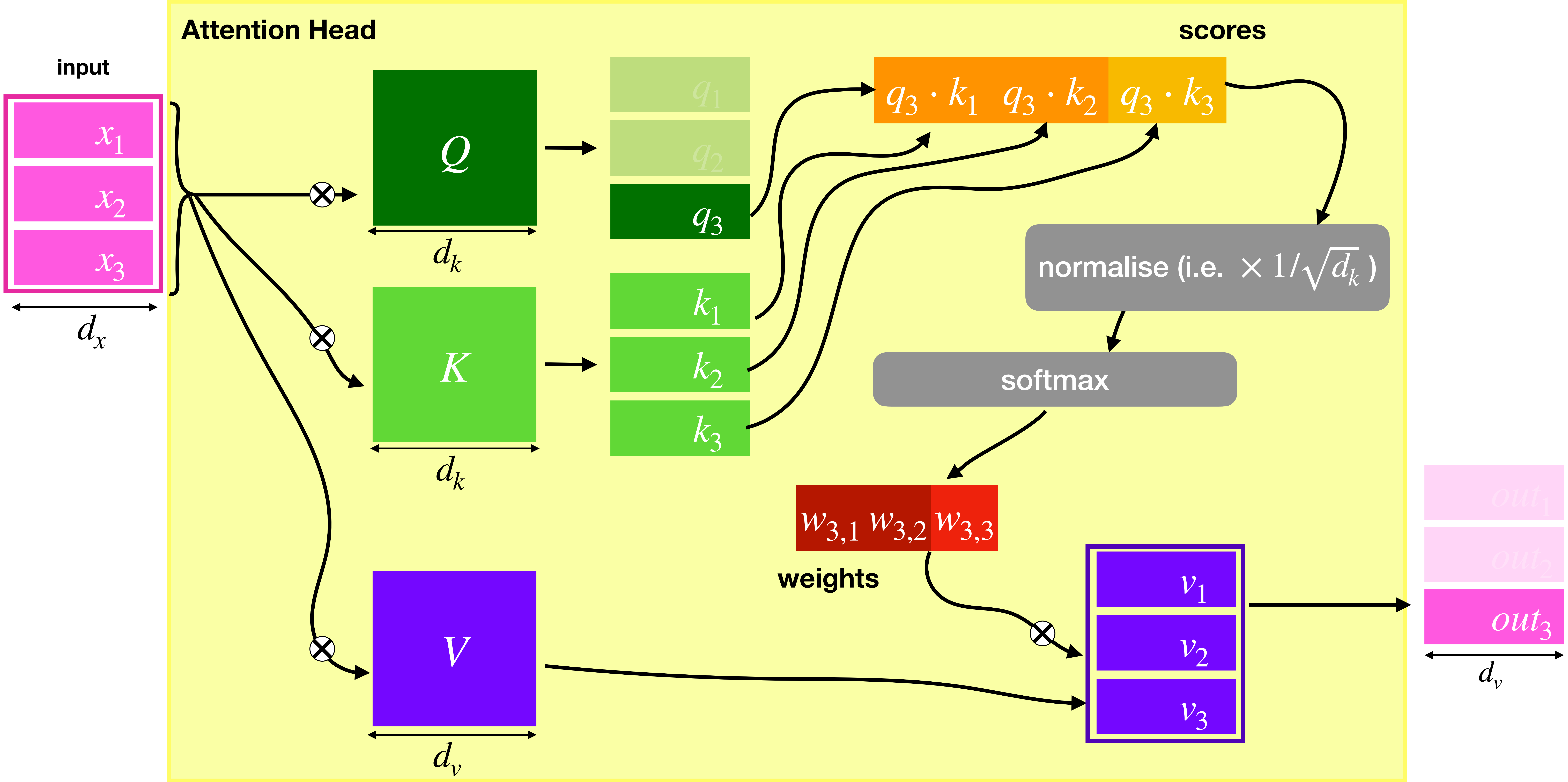
Background - Self Attention (Single Head)



Background - Self Attention (Single Head)

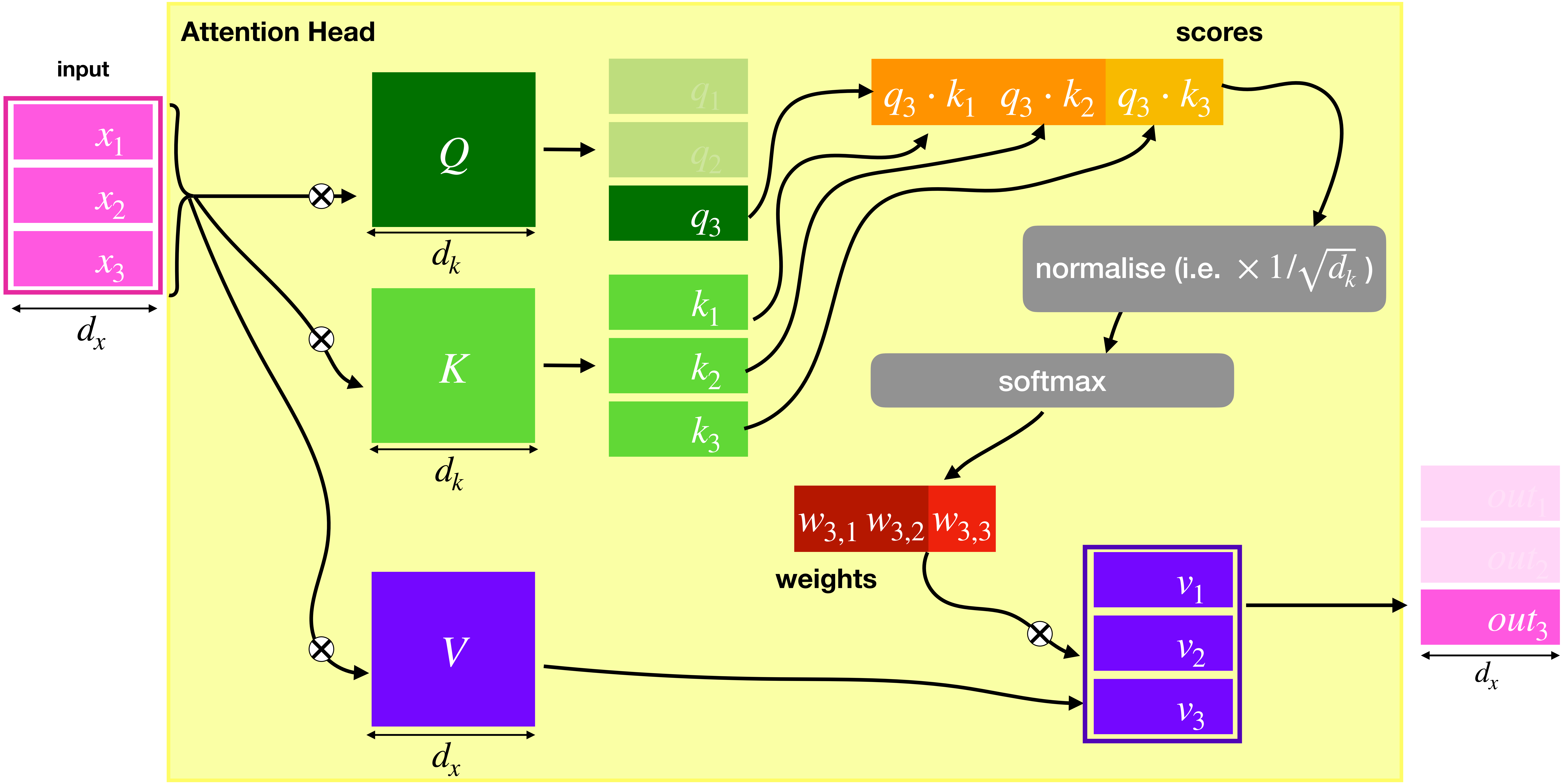


Background - Self Attention (Single Head)

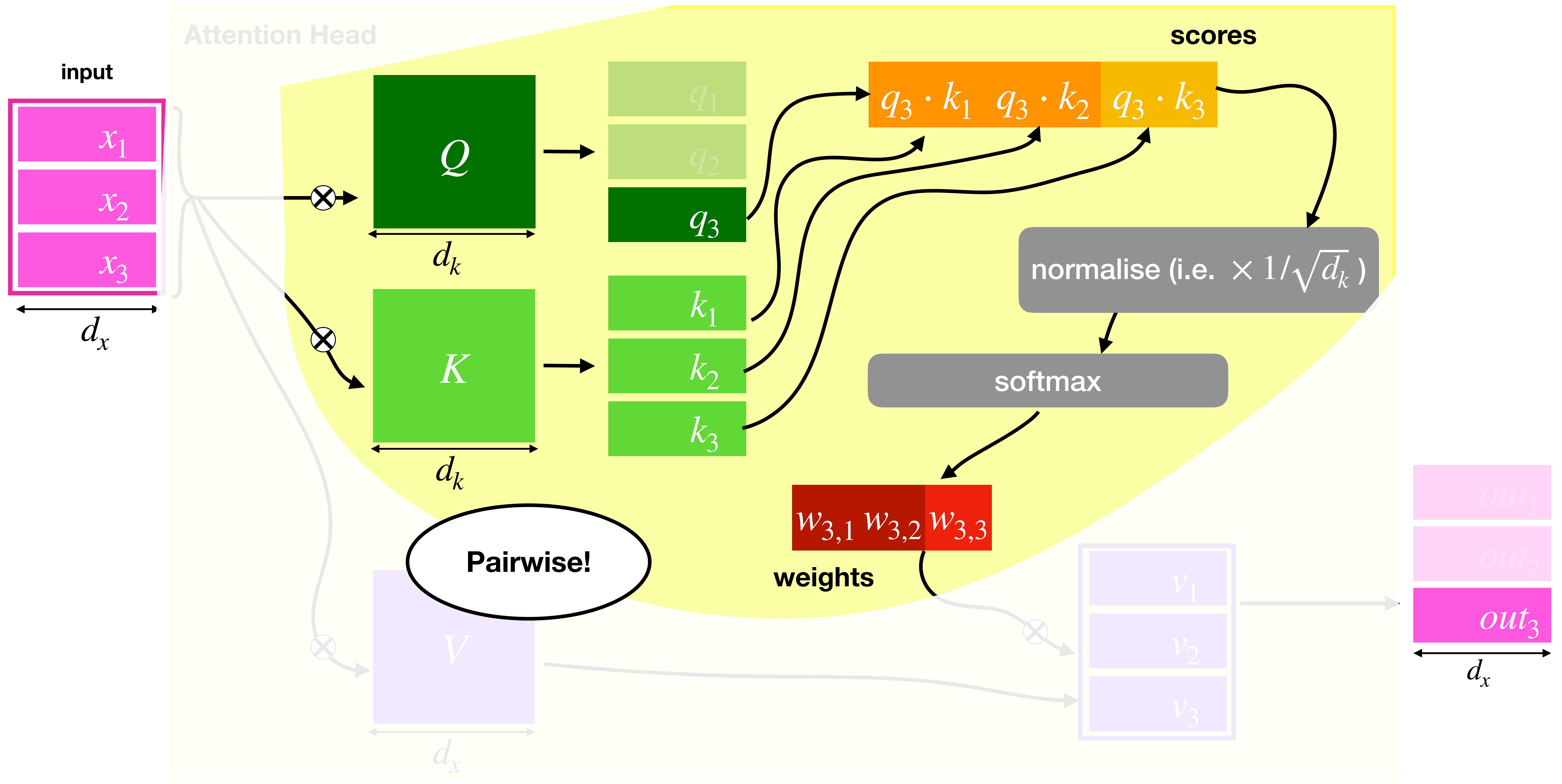


So, how do we present an attention head?

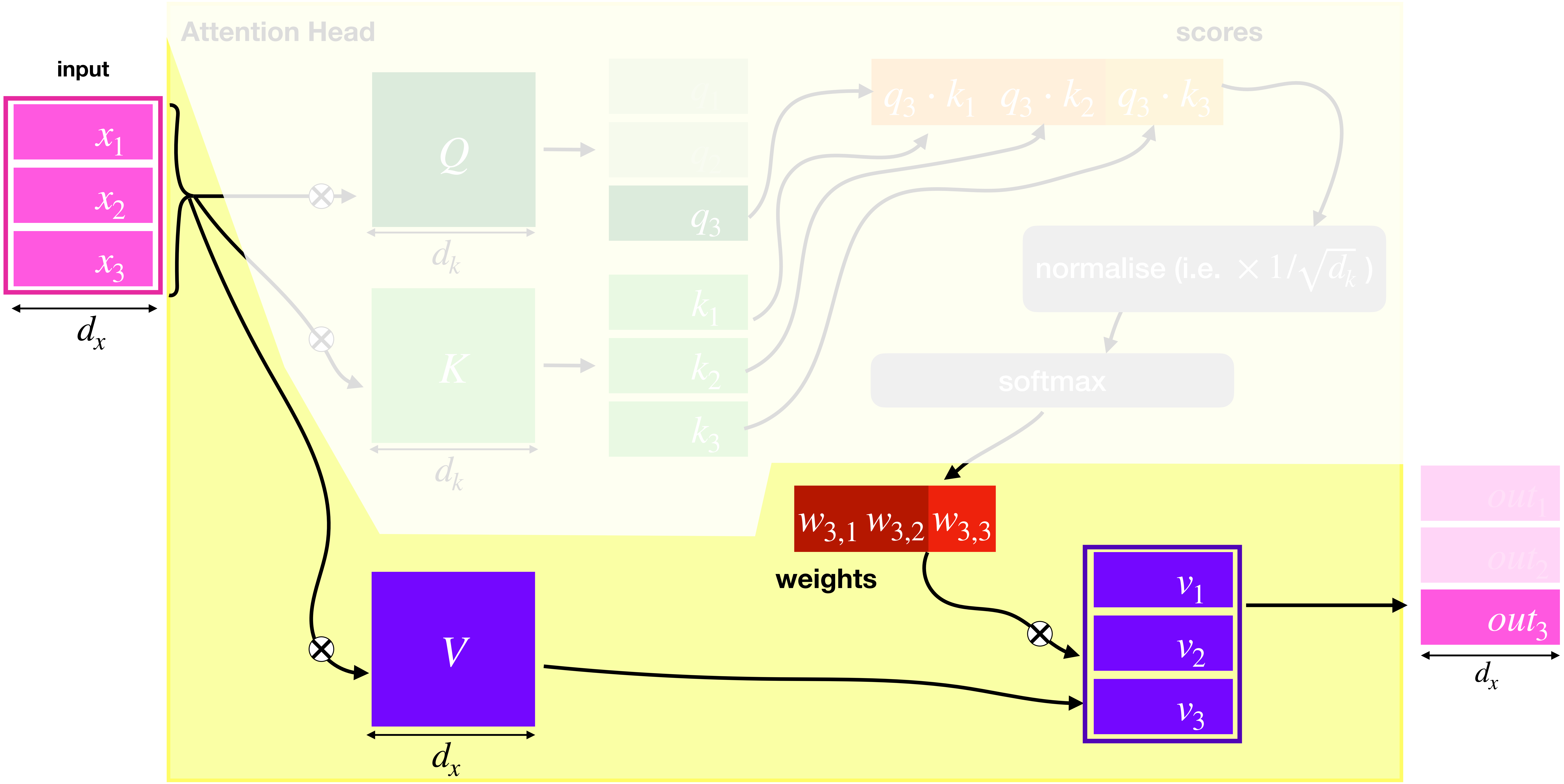
Self Attention (Single Head)



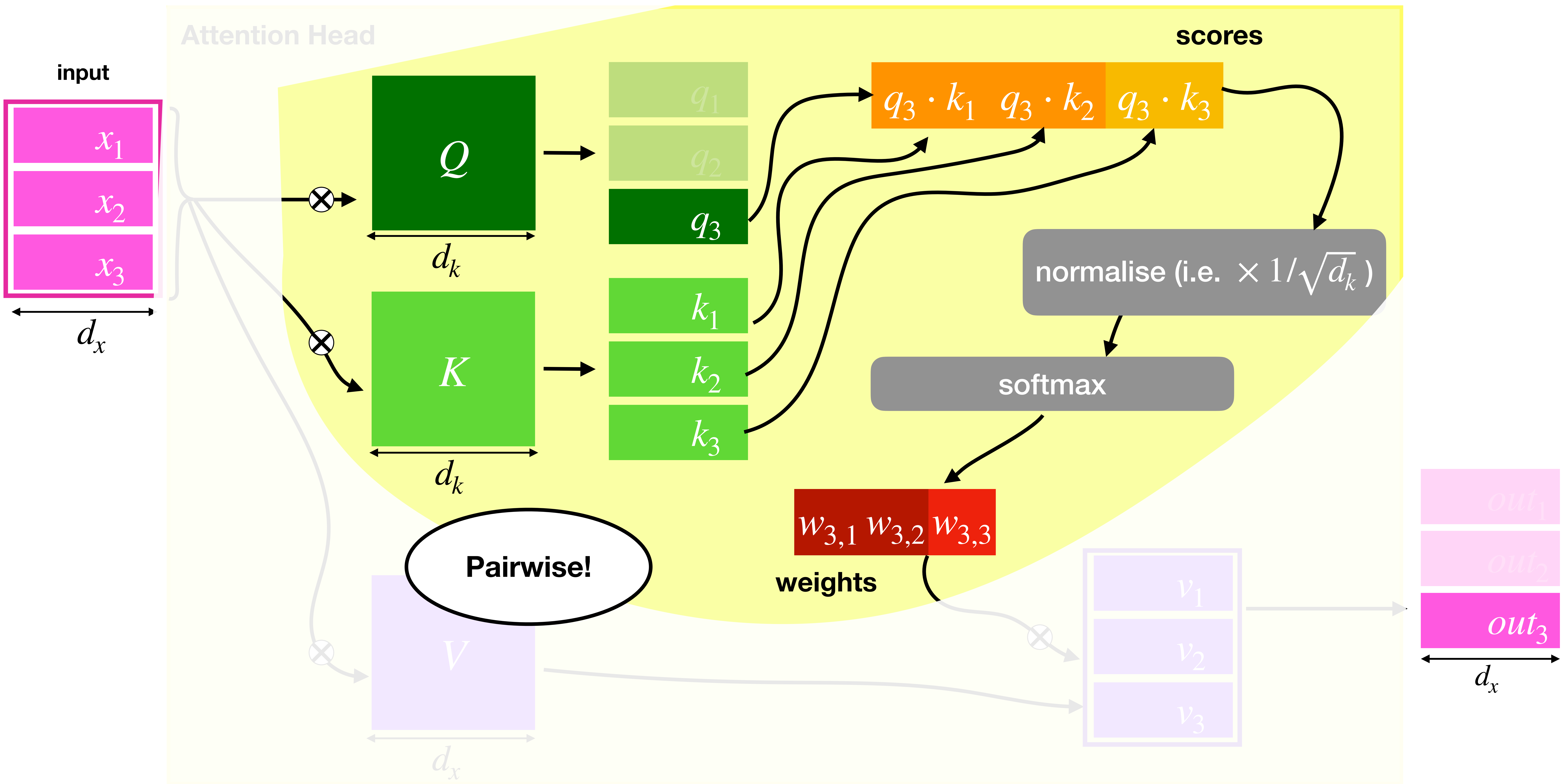
Self Attention (Single Head)



Self Attention (Single Head)



Single Head: Scoring ↔ Selecting

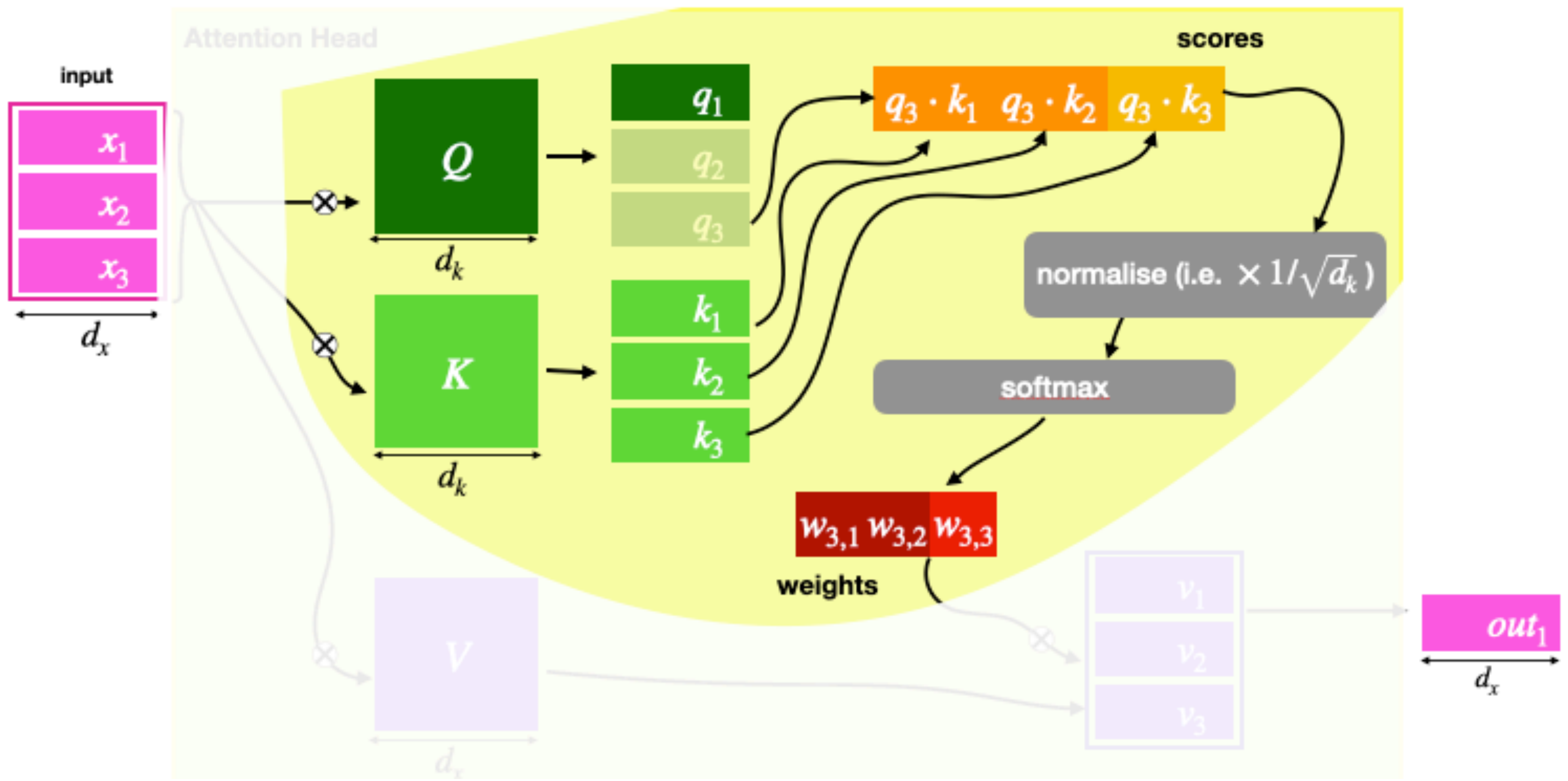


Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

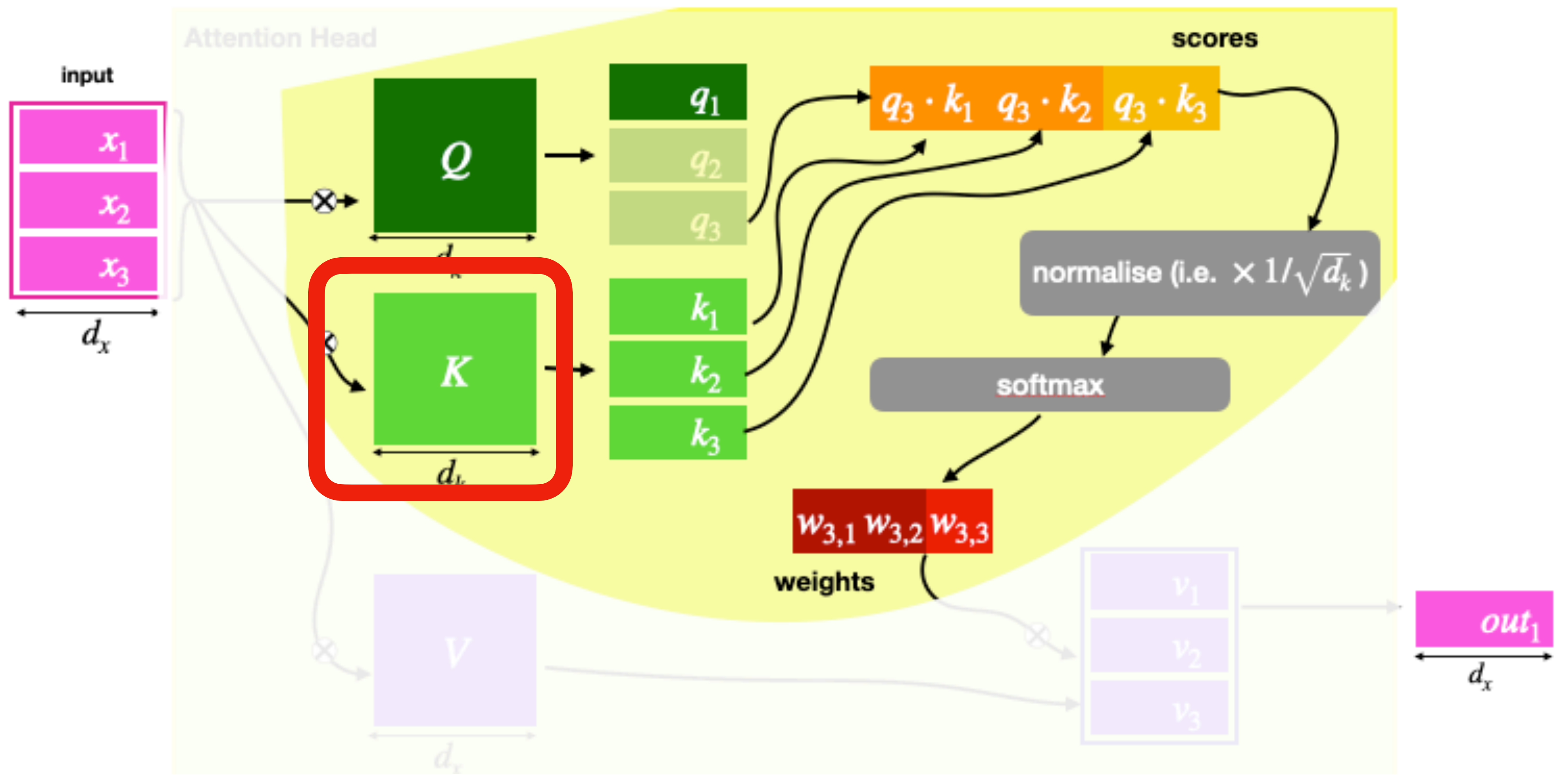


Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

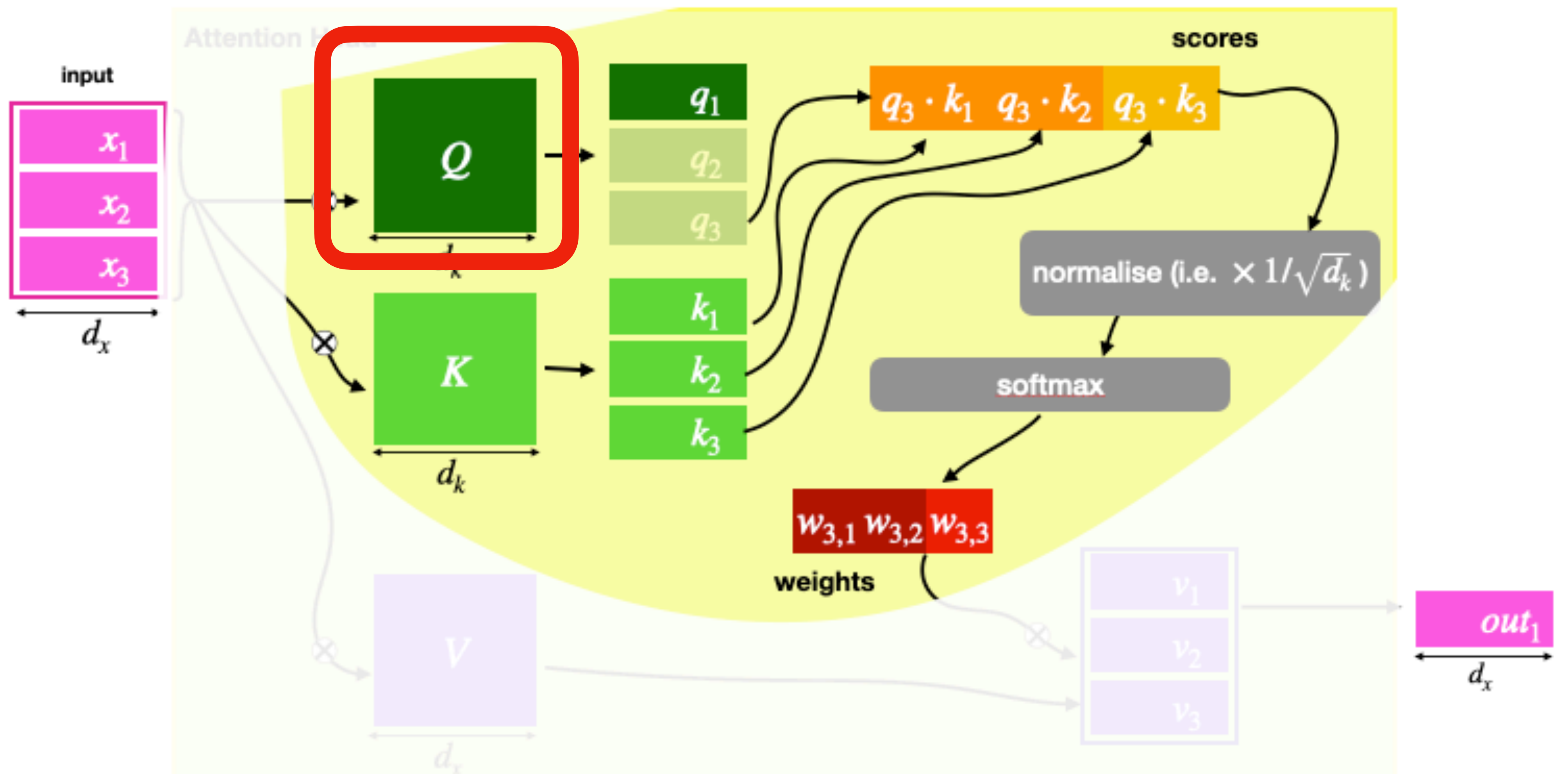


Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

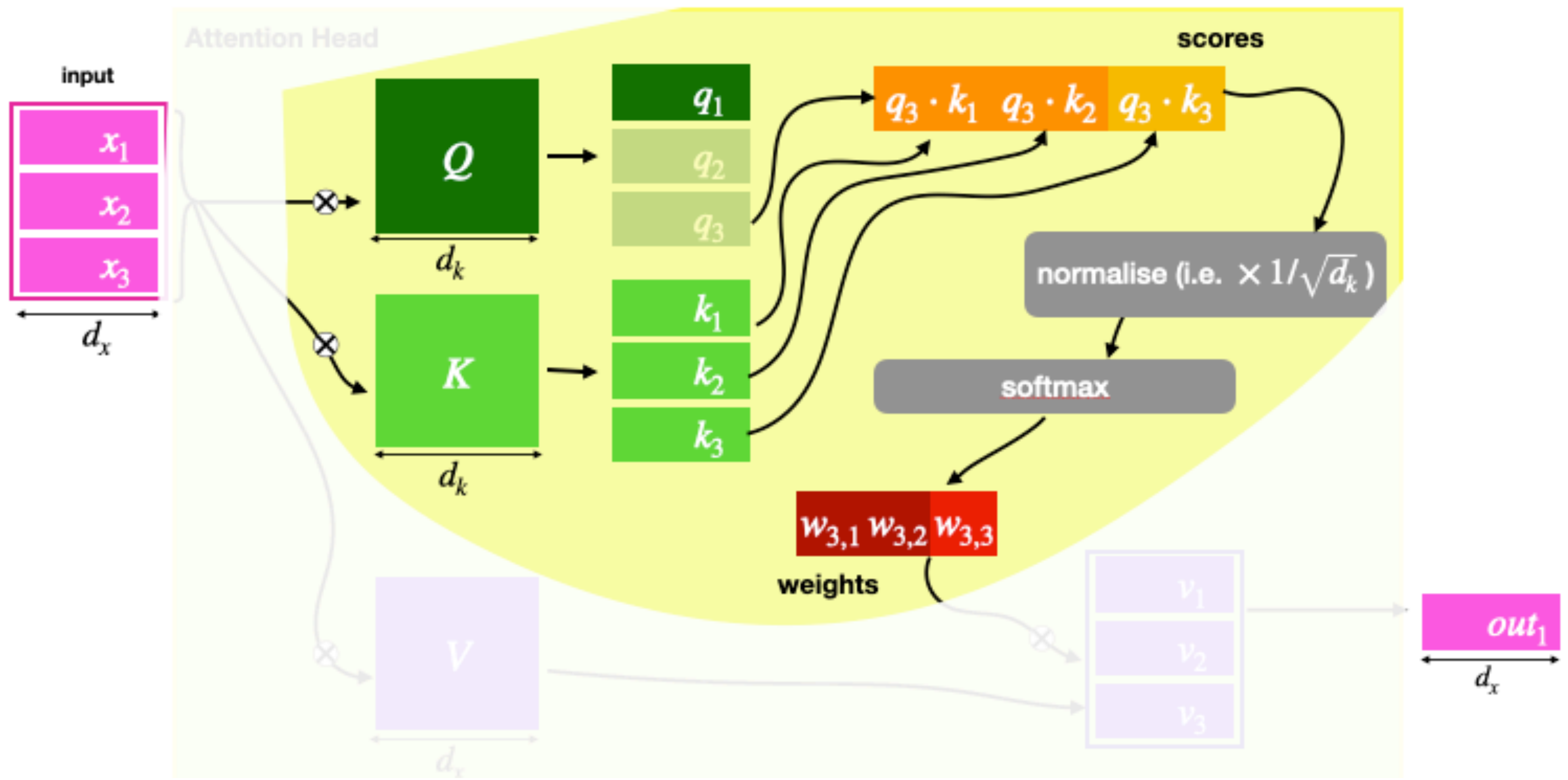


Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

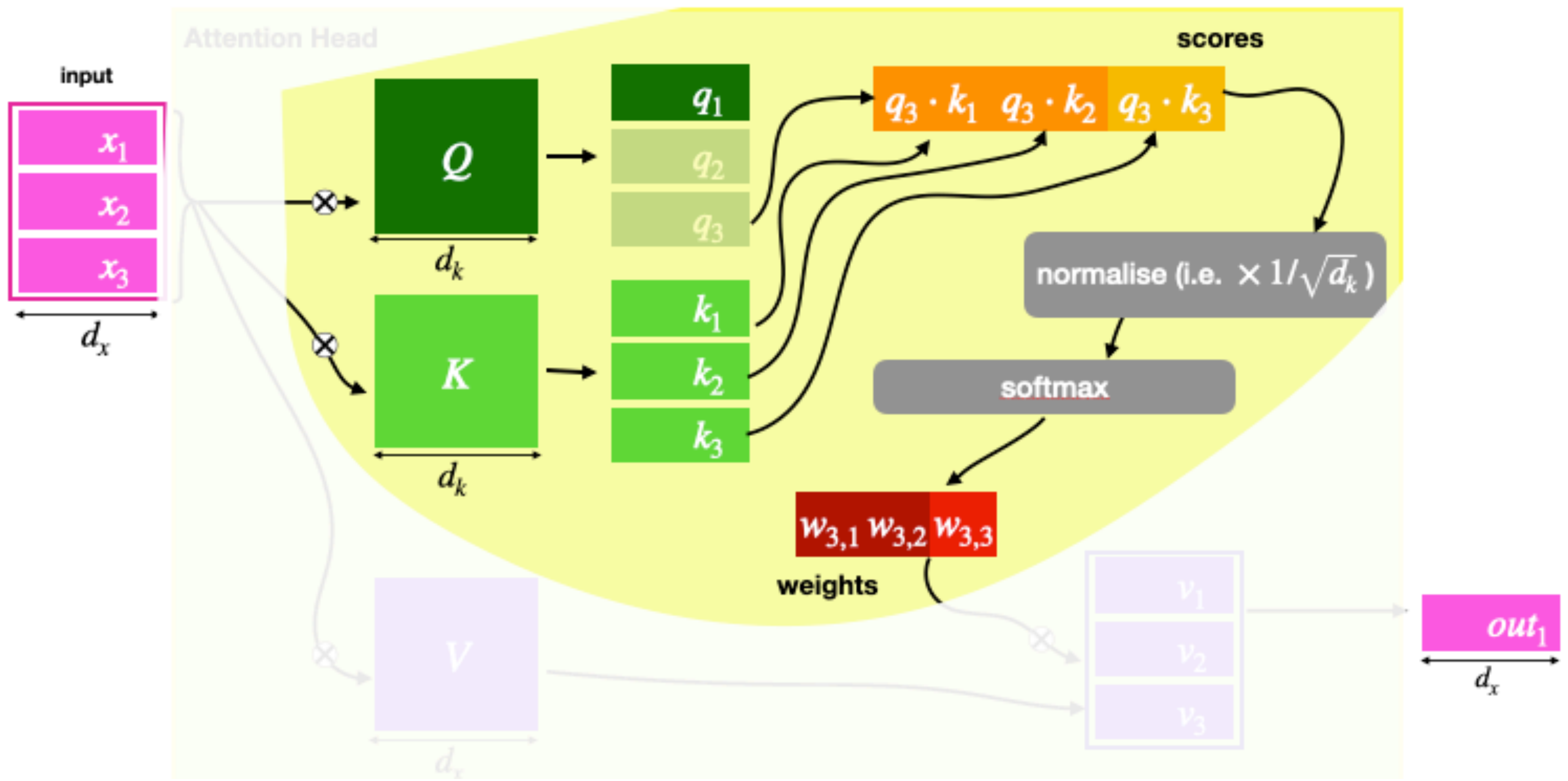


Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

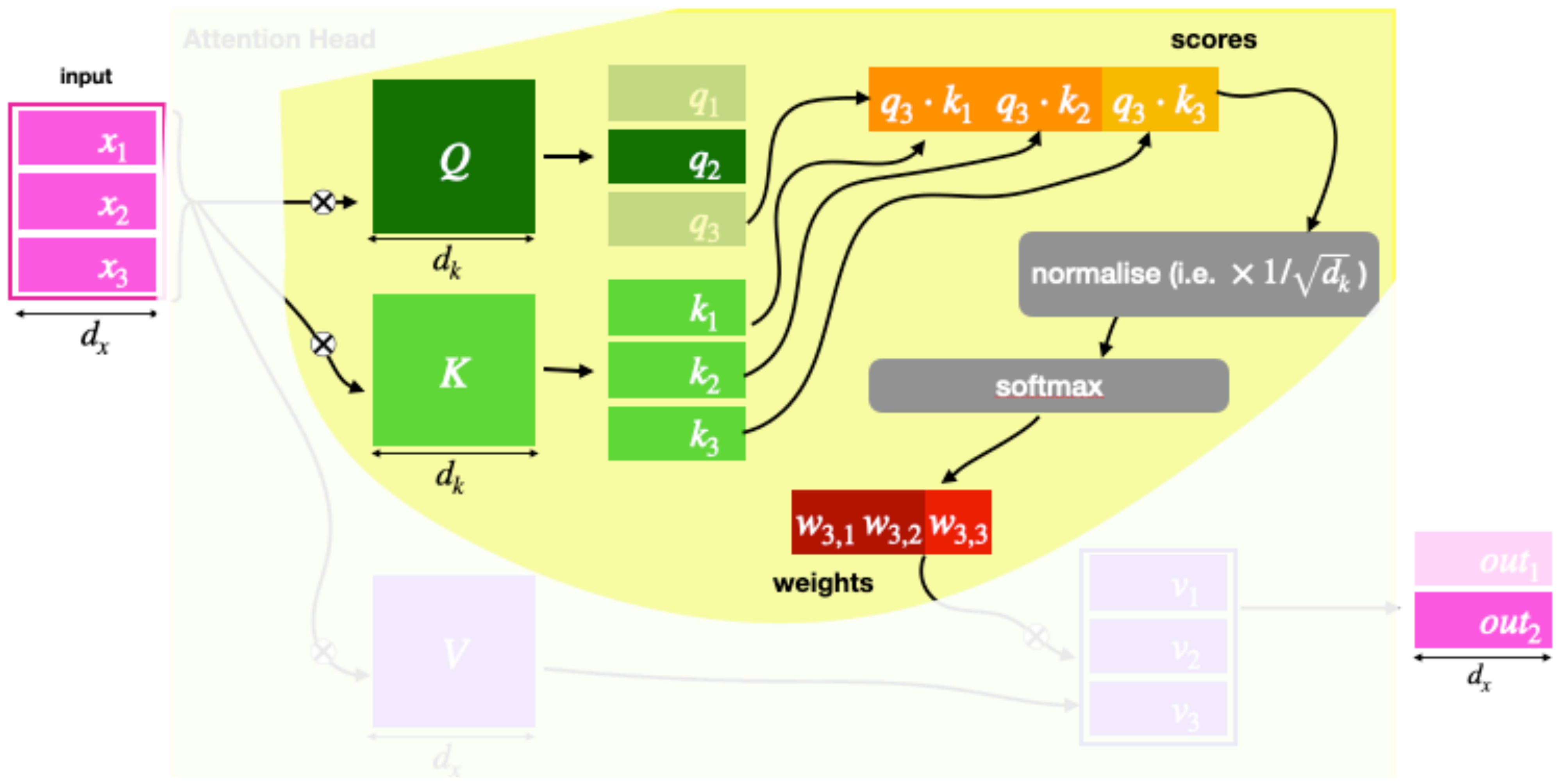


Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

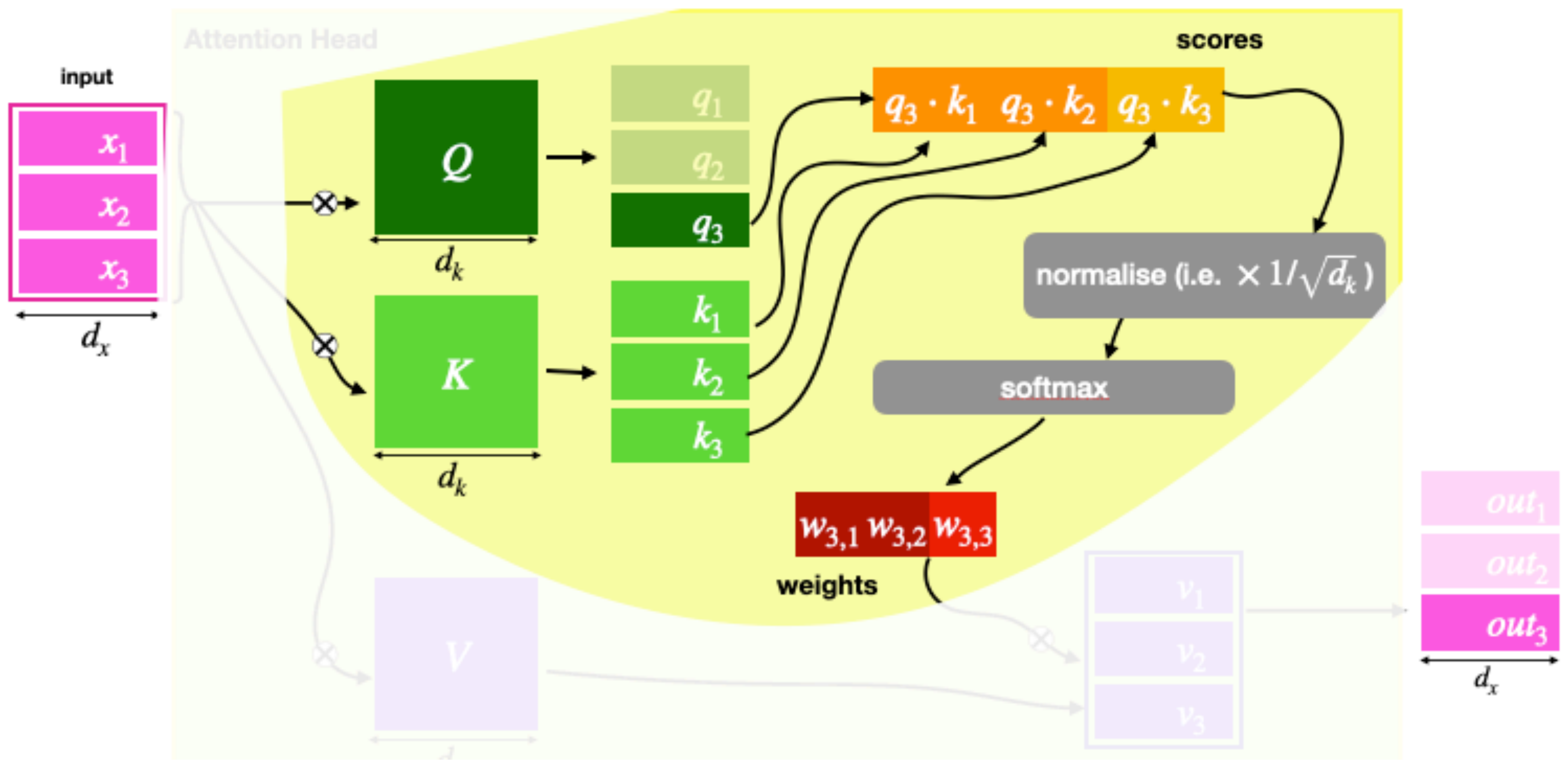


Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

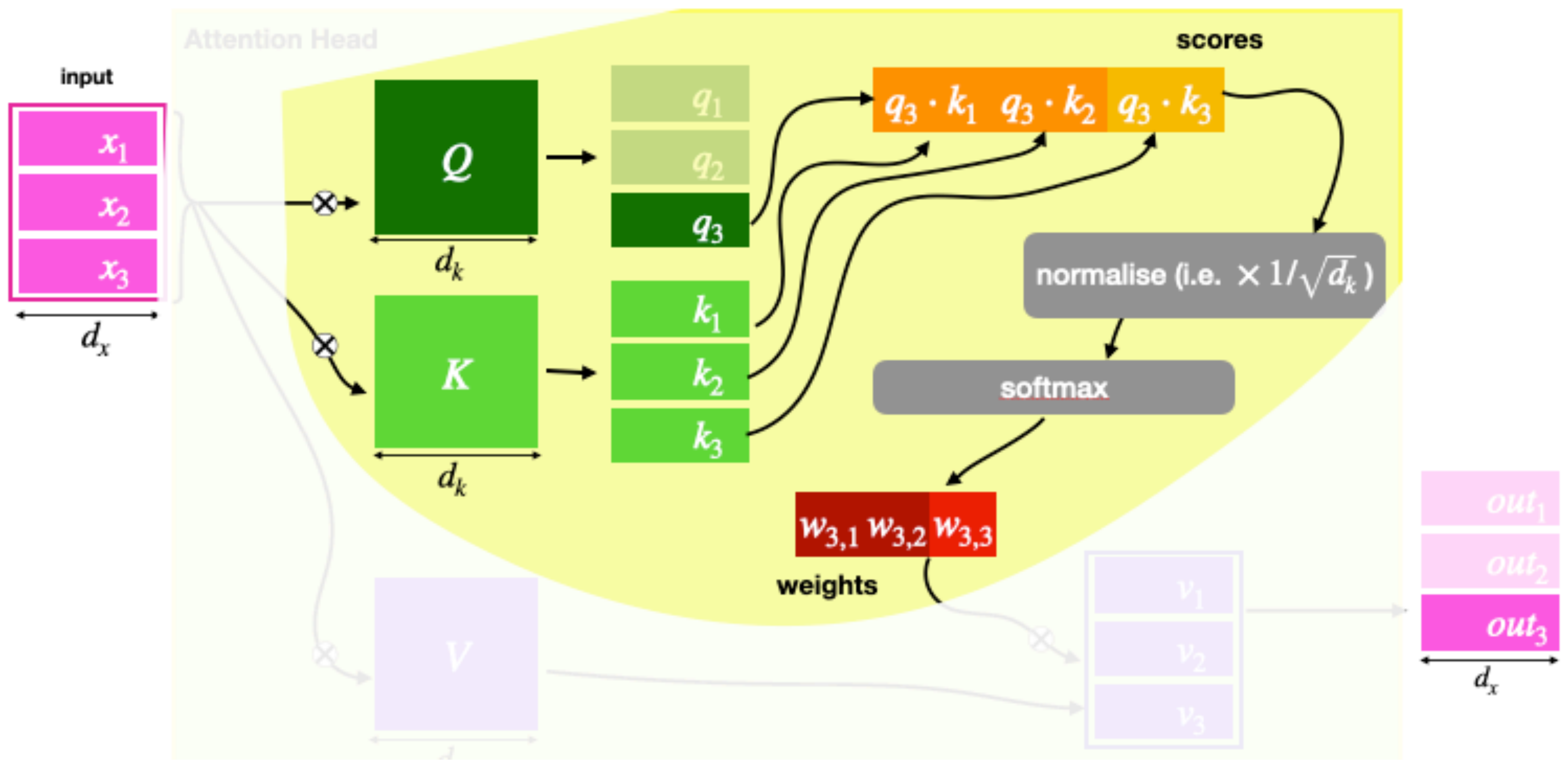
sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F



Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions



sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

Another example:

sel2 = select([2,0,0],[0,1,2],$\boxed{>=}$)

	2	0	0
0	T	T	T
1	T	F	F
2	T	F	F

Single Head: Scoring ↔ Selecting

```
prevs = select([0,1,2],[0,1,2],<=)
```

	0	1	0
0	T	F	F
1	T	T	F
2	T	T	T

Single Head: Scoring $\leftrightarrow$ Selecting

prevs = select([0,1,2],[0,1,2],<=)

	0	1	0
0	T	F	F
1	T	T	F
2	T	T	T

$(1, 0, 0, \ldots)$ k_1

$(0, 1, 0, \ldots)$ k_2

$(0, 0, 1, \ldots)$ k_3

Single Head: Scoring $\leftrightarrow$ Selecting

prevs = select([0,1,2],[0,1,2],<=)

	0	1	0
0	T	F	F
1	T	T	F
2	T	T	T

$(1, 0, 0, \ldots)$ k_1

$(0, 1, 0, \ldots)$ k_2

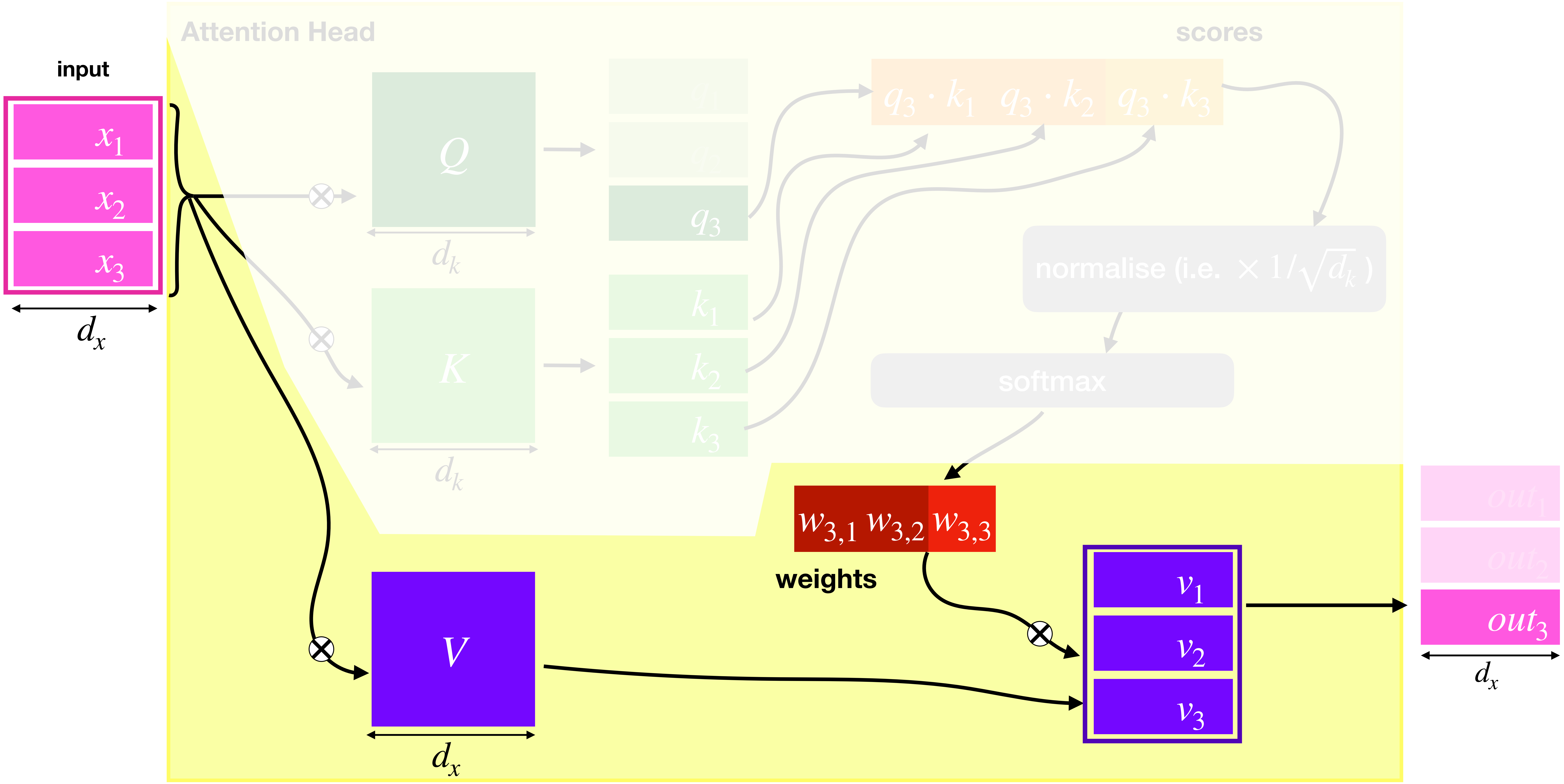
$(0, 0, 1, \ldots)$ k_3

$(1, 0, 0, \ldots)$ q_1

$(1, 1, 0, \ldots)$ q_2

$(1, 1, 1, \ldots)$ q_3

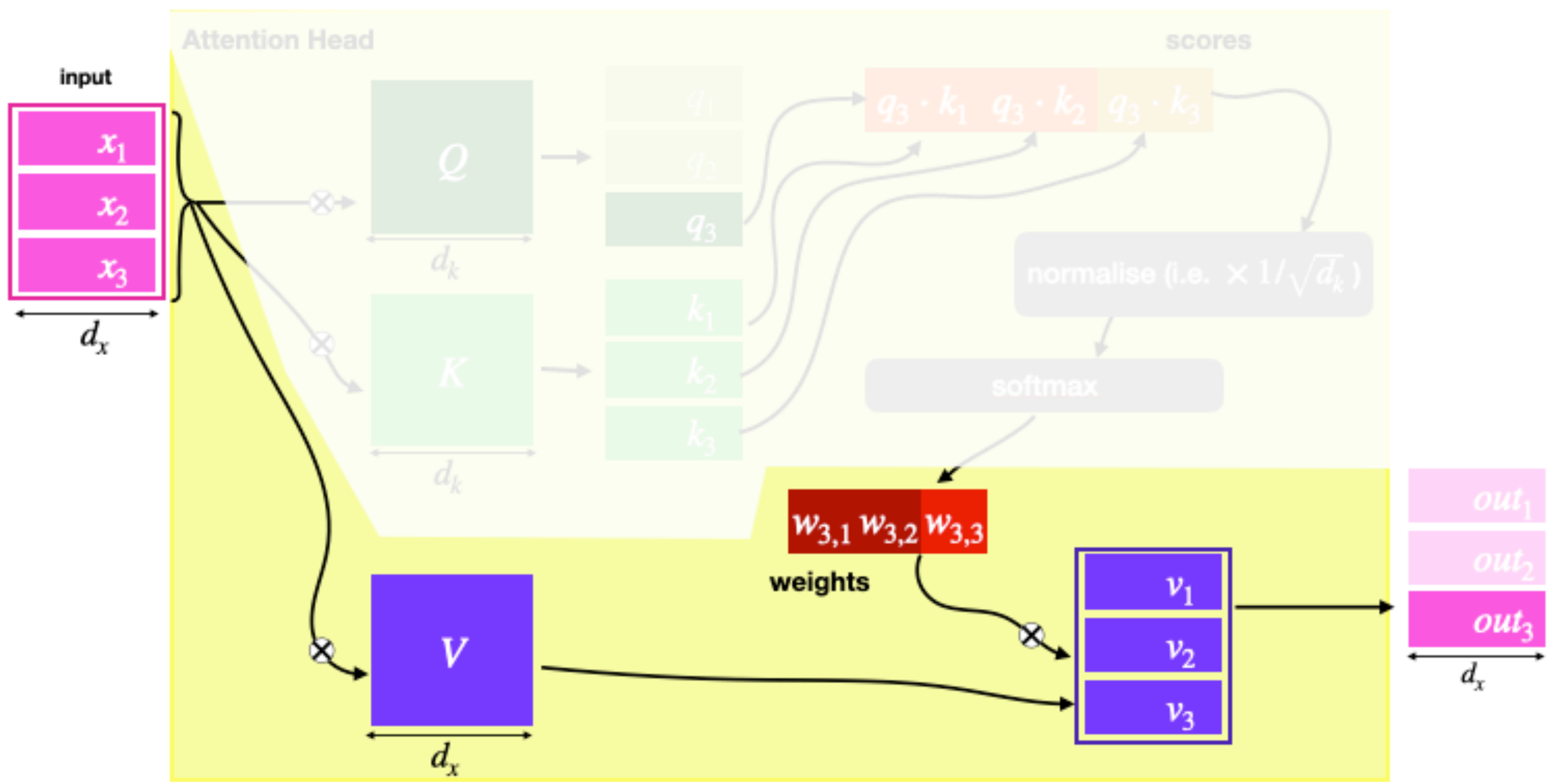
Single Head: Weighted Average ↔ Aggregation



Single Head: Weighted Average ↔ Aggregation

new=aggregate(sel, [1,2,4])

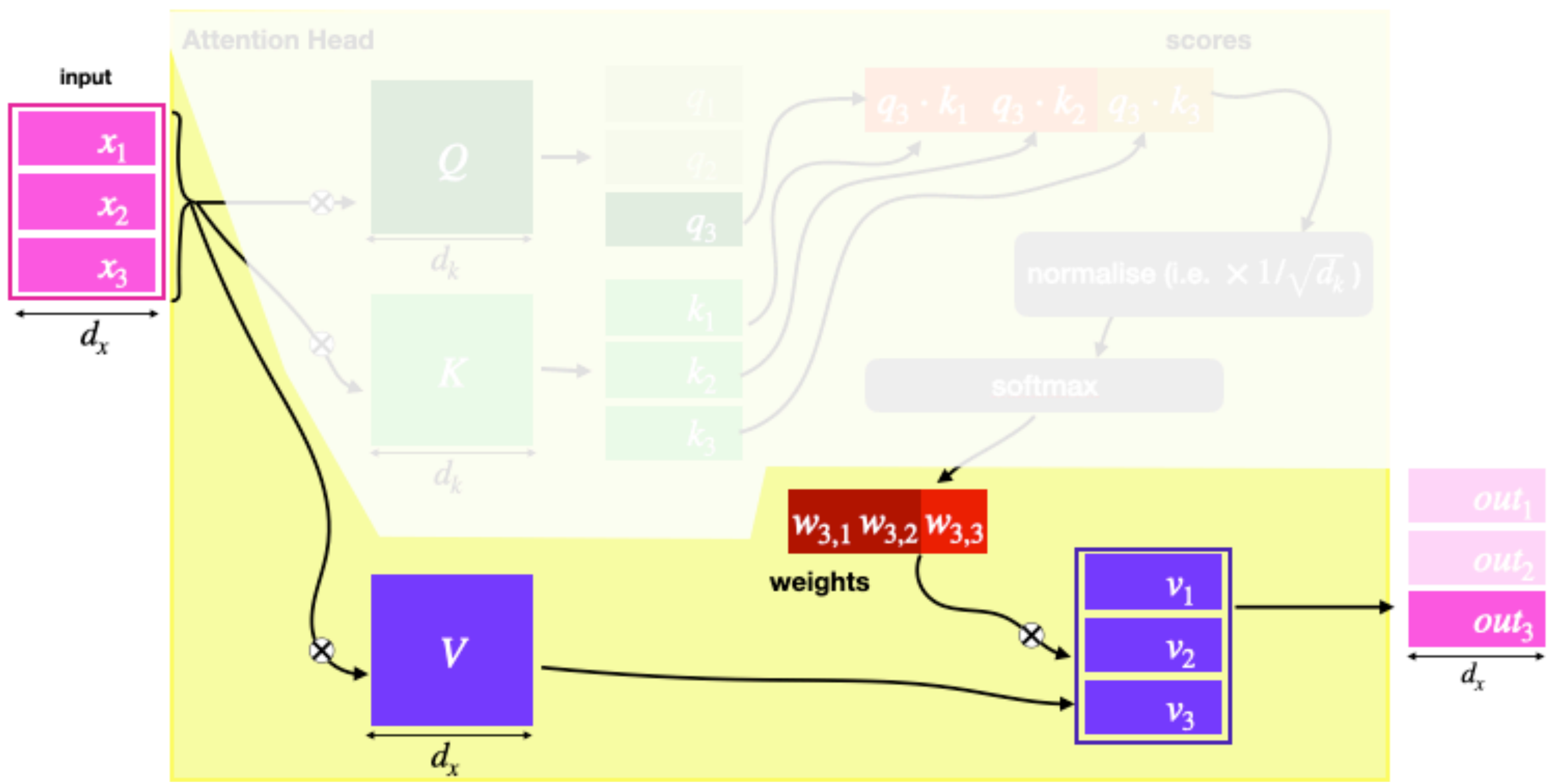
			1	2	4					
F	T	T	1	2	4	=>	3			
F	F	F	1	2	4	=>	0	=>	[3,0,1]	
T	F	F	1	2	4	=>	1			



Single Head: Weighted Average ↔ Aggregation

new=aggregate(sel, [1,2,4])

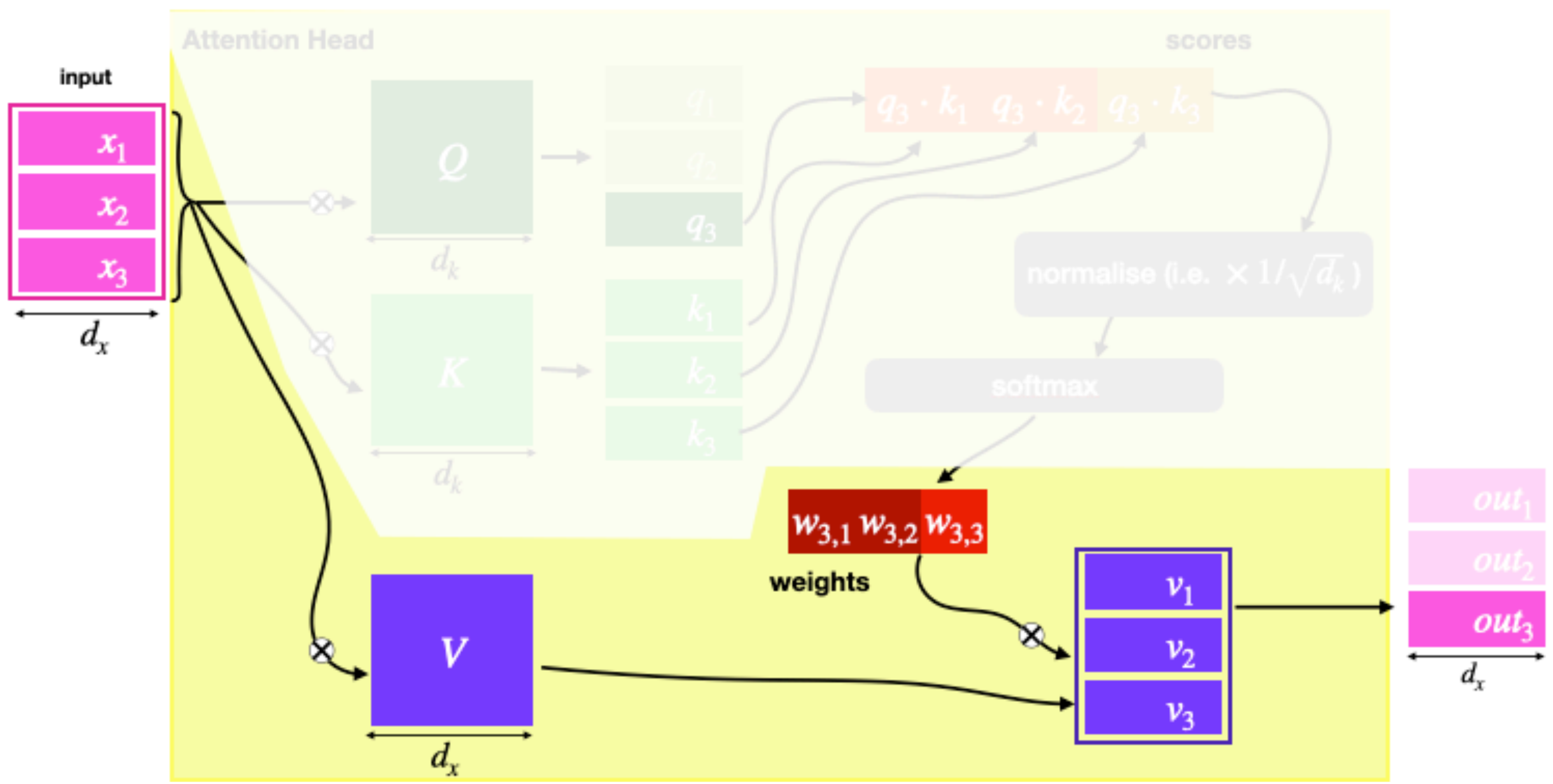
			1 2 4				
F	T	T	1 2 4	=>	3		
F	F	F	1 2 4	=>	0	=>	[3,0,1]
T	F	F	1 2 4	=>	1		



Single Head: Weighted Average ↔ Aggregation

new=aggregate(sel, [1,2,4])

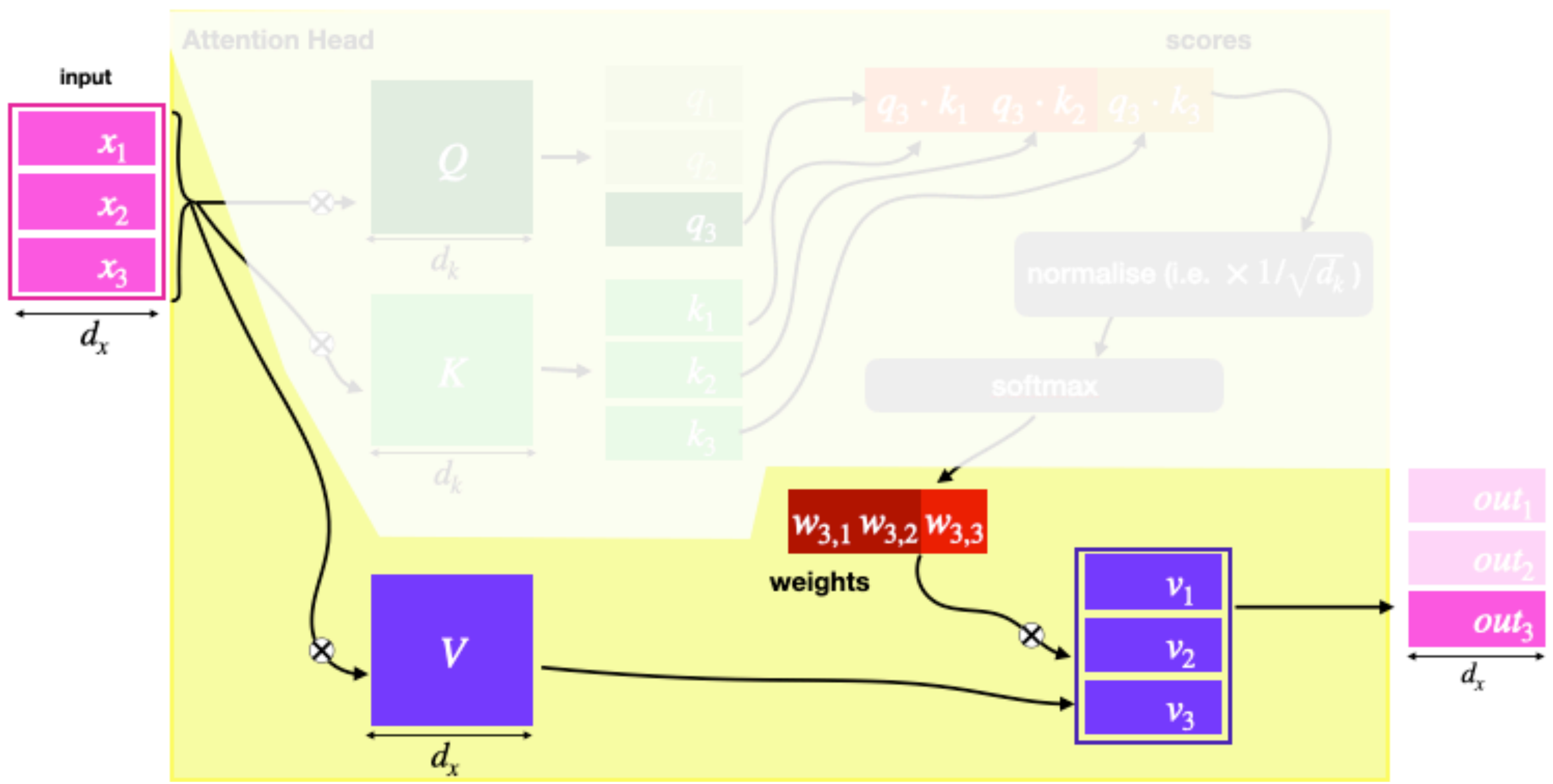
			1	2	4				
F	T	T	1	2	4	=>	3		
F	F	F	1	2	4	=>	0	=>	[3,0,1]
T	F	F	1	2	4	=>	1		



Single Head: Weighted Average ↔ Aggregation

new=aggregate(sel, [1,2,4])

			1	2	4					
F	T	T	1	2	4	=>	3			
F	F	F	1	2	4	=>	0	=>	[3,0,1]	
T	F	F	1	2	4	=>	1			



Single Head: Weighted Average ↔ Aggregation

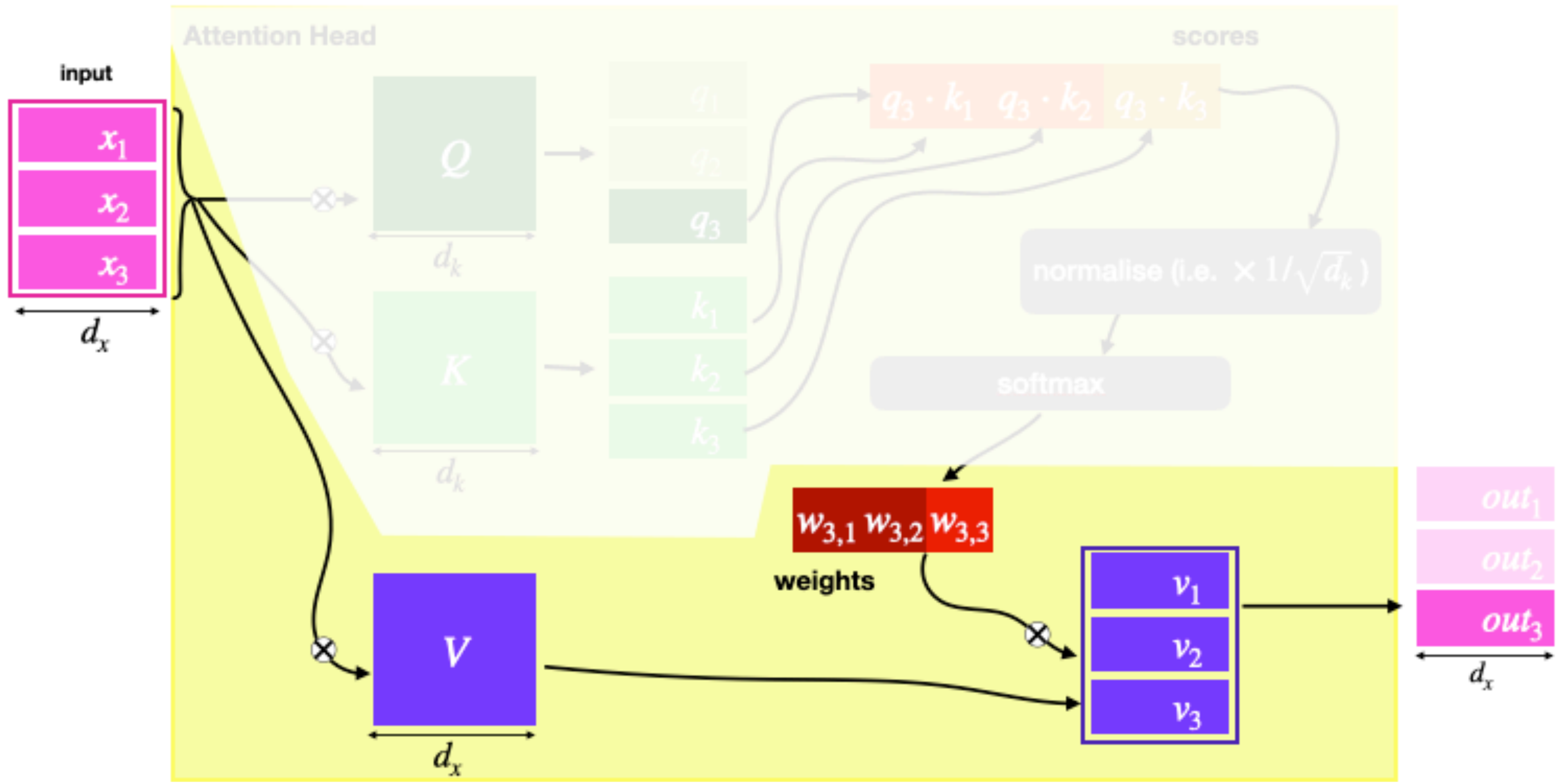
new=aggregate(sel, [1,2,4])

1 2 4

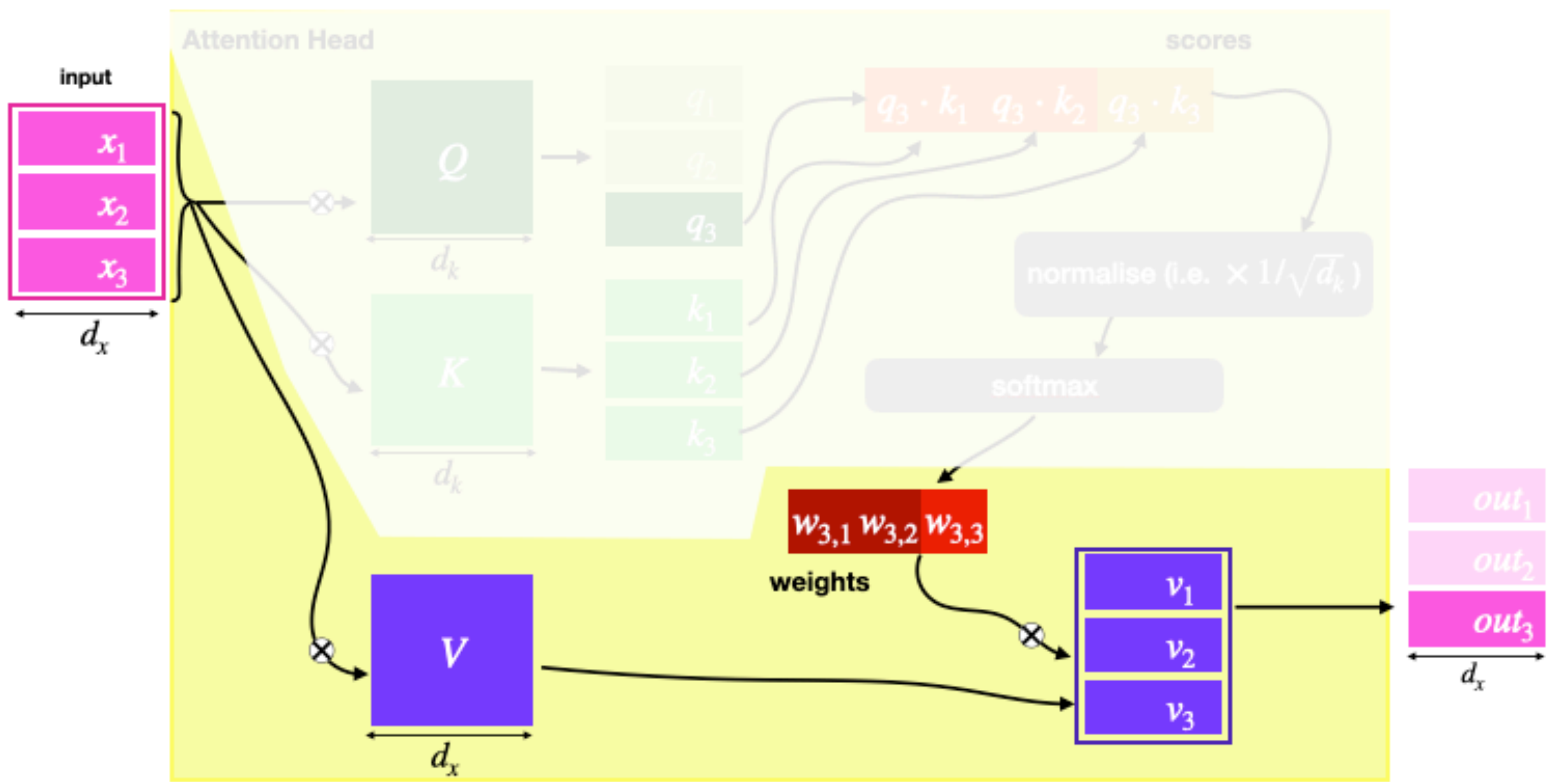
F T T 1 2 4 => 3

F F F 1 2 4 => 0 => [3,0,1]

T F F 1 2 4 => 1



Single Head: Weighted Average ↔ Aggregation



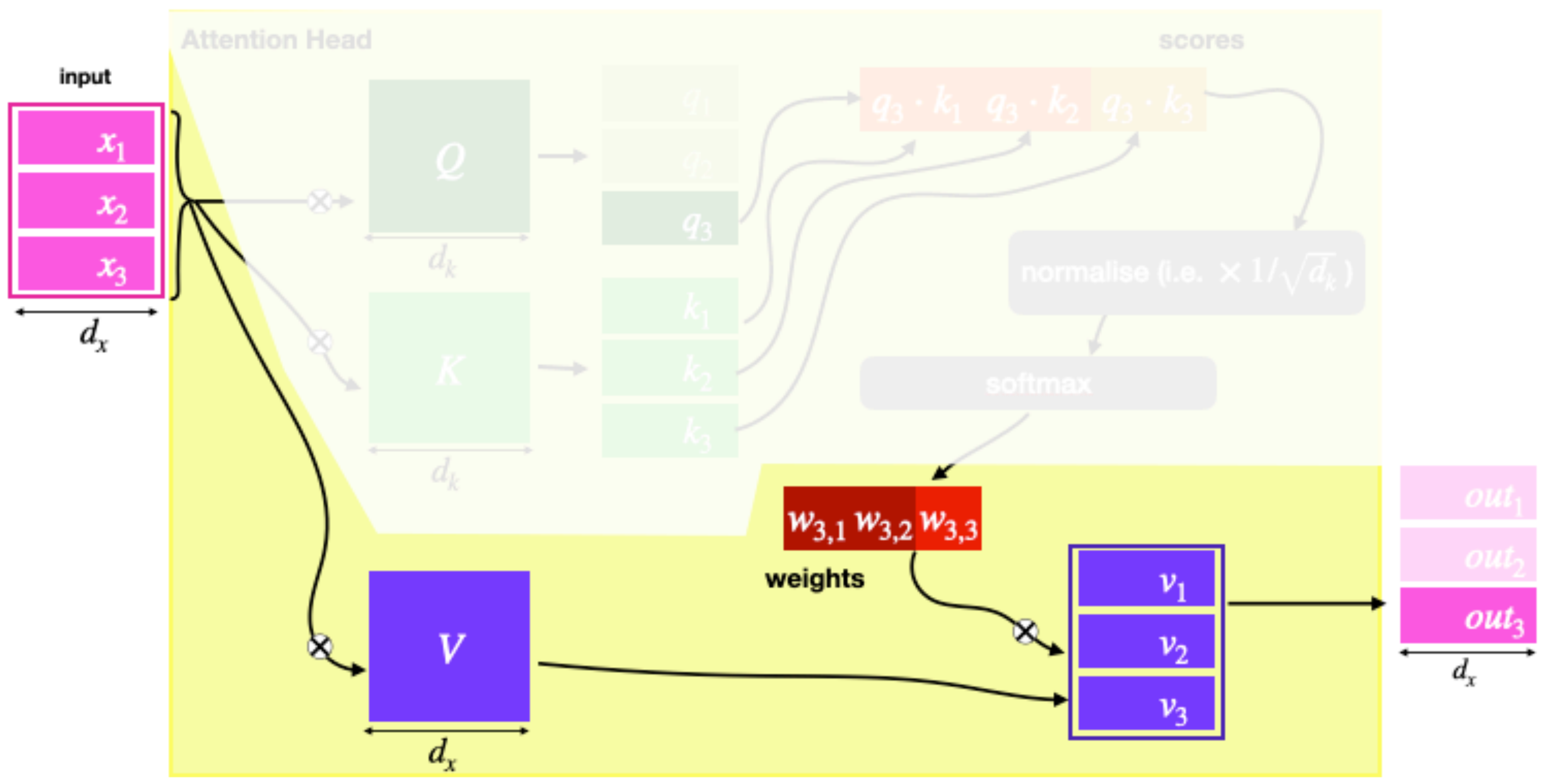
new=aggregate(sel, [1,2,4])

			1 2 4				
F	T	T	1 2 4	=>	3		
F	F	F	1 2 4	=>	0	=>	[3,0,1]
T	F	F	1 2 4	=>	1		

Single Head: Weighted Average ↔ Aggregation

new=aggregate(sel, [1,2,4])

			1	2	4					
F	T	T	1	2	4	=>	3			
F	F	F	1	2	4	=>	0	=>	[3,0,1]
T	F	F	1	2	4	=>	1			



Symbolic language + no averaging when only one position selected allows (for example):

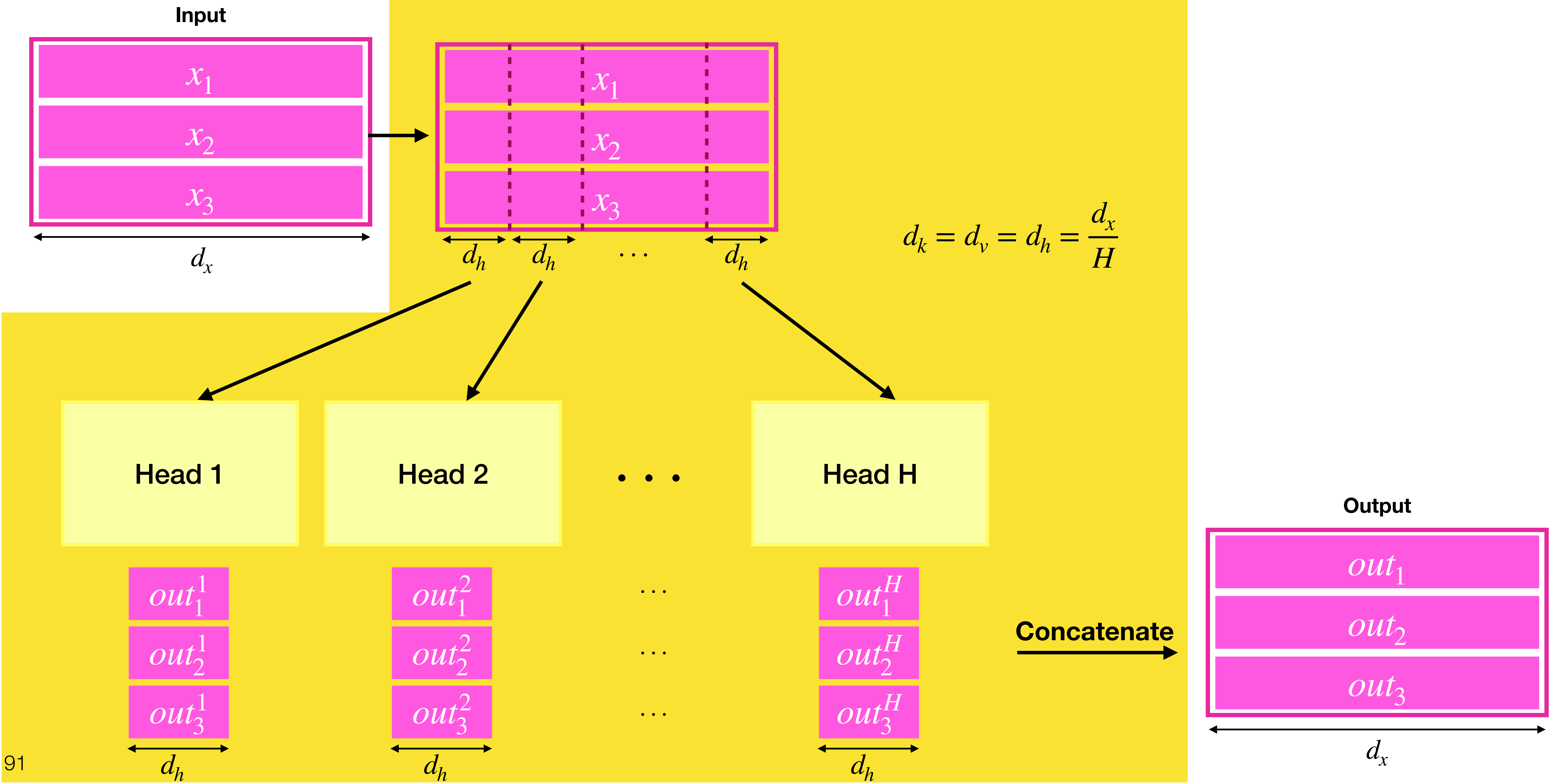
reverse=aggregate(flip, [A,B,C])

			A	B	C				
F	F	T	A	B	C	=>	C		
F	T	F	A	B	C	=>	B	=>	[C,B,A]
T	F	F	A	B	C	=>	A		

Great!

Now do multi-headed attention

Background - Multi-Headed Self Attention



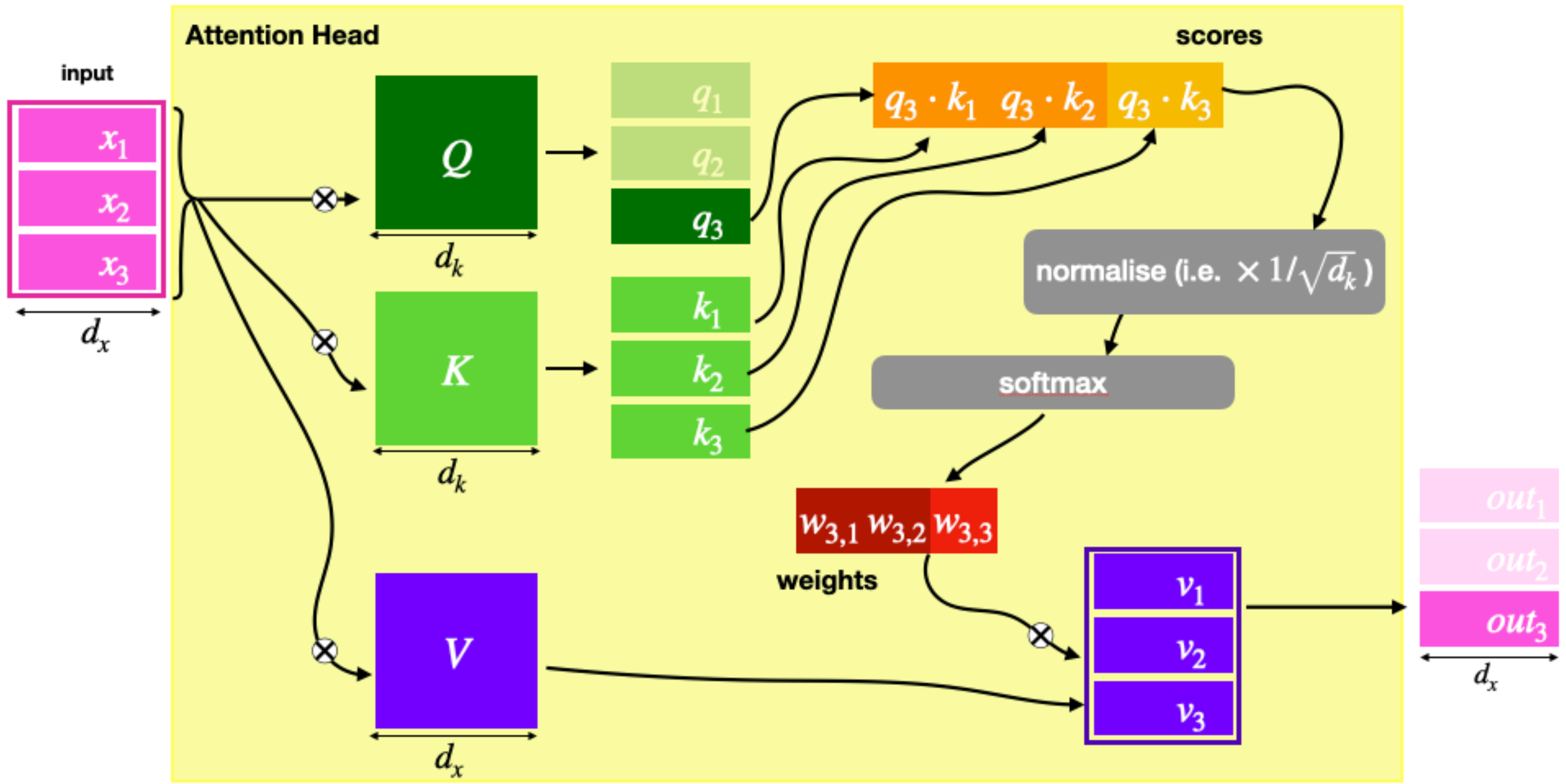
$$d_k = d_v = d_h = \frac{d_x}{H}$$

The multi-headed attention lets one layer do multiple single head operations

We do not need ‘new’ RASP operations to describe it!

(We will just let the RASP compiler know it can place multiple heads on the same layer)

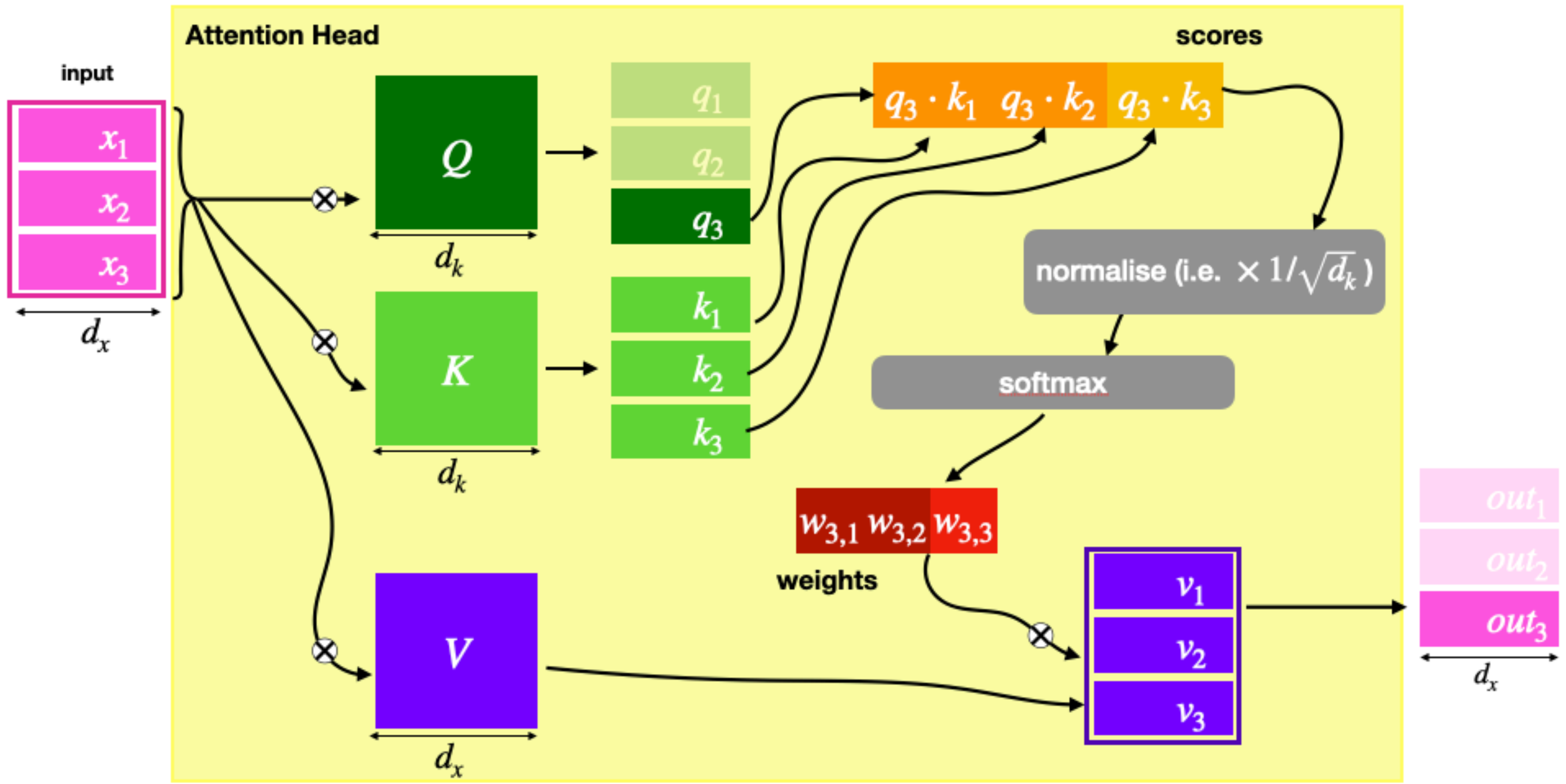
Example: Reverse



reverse=aggregate(flip, [A,B,C])

	A B C					
F F **T**	A B **C**	=>	C			
F **T** F	A **B** C	=>	B	=>	[C,B,A]	
T F F	**A** B C	=>	A			

Example: Reverse

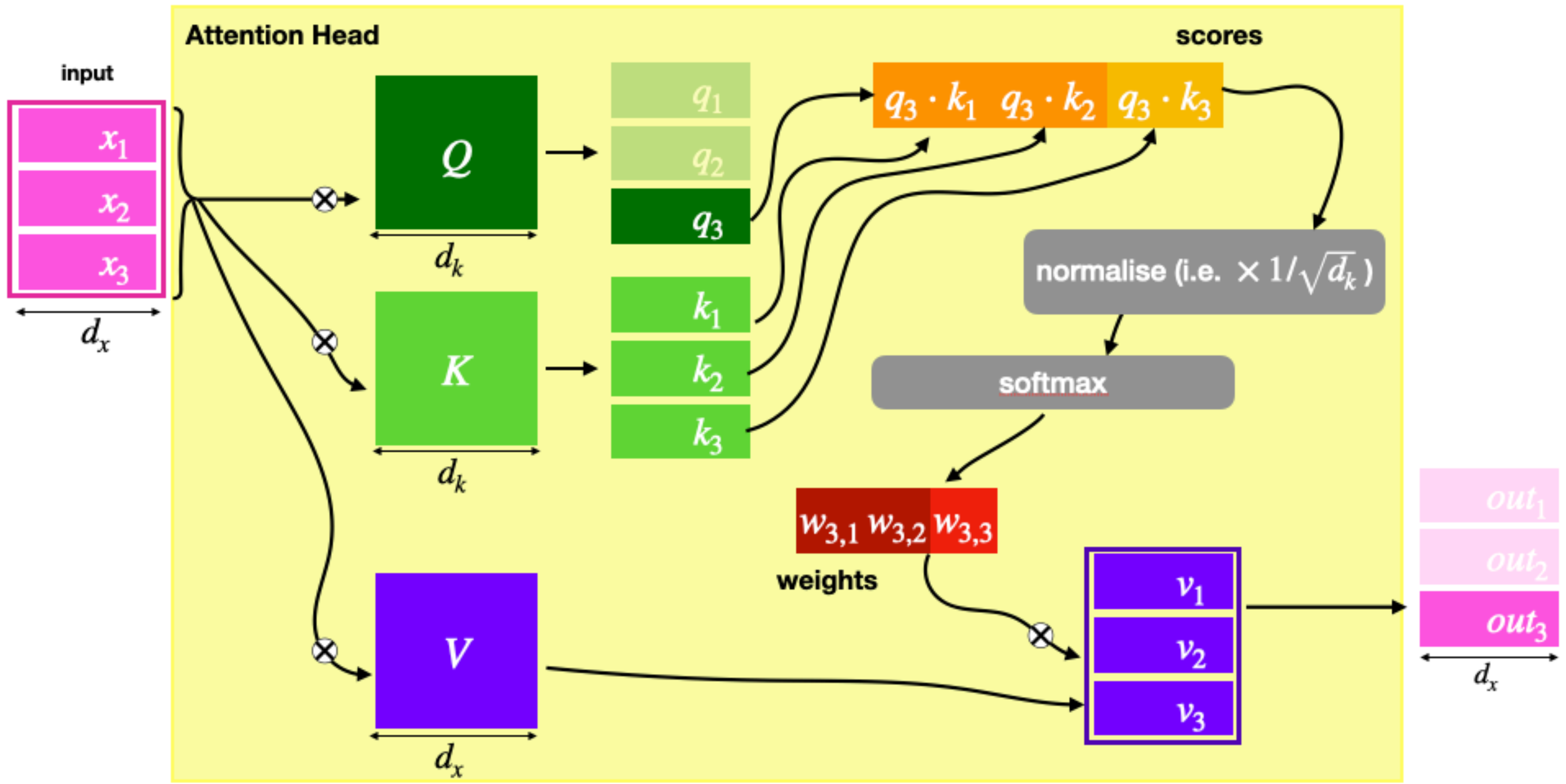



```
>>        = select(length-indices-1,        ,   );
```

reverse=aggregate(flip, [A,B,C])

			A B C					
F	F	T	A B C	=>	C			
F	T	F	A B C	=>	B	=>	[C,B,A]
T	F	F	A B C	=>	A			

Example: Reverse




```
>>        = select(length-indices-1, indices,    );
```

reverse=aggregate(flip, [A,B,C])

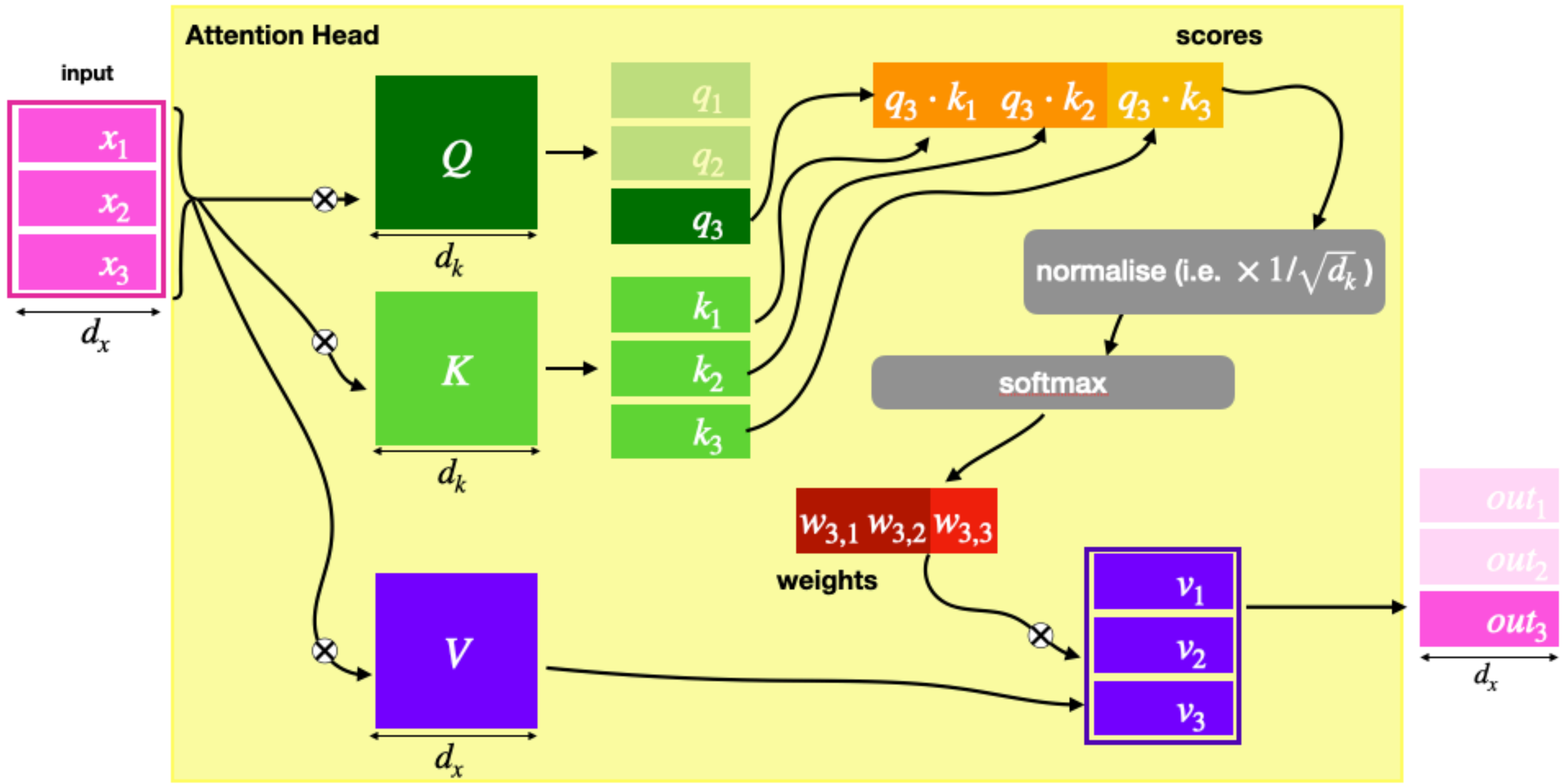
A B C

F F T   A B C => C

F T F   A B C => B => [C,B,A]

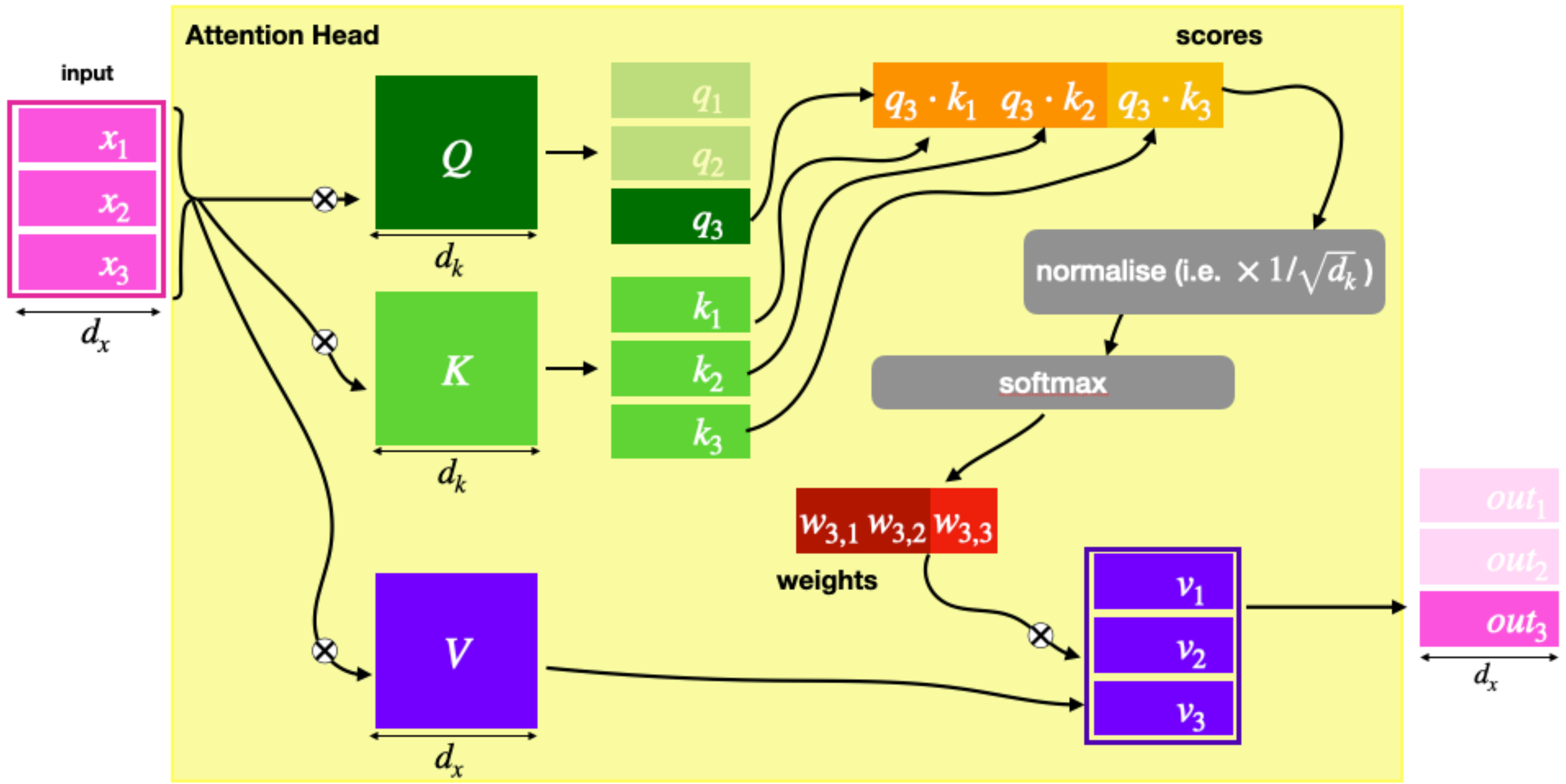
T F F   A B C => A

Example: Reverse




```
>> flip = select(length-indices-1,indices,==);
     selector: flip
         Example:
                 h e l l o
             h |         1
             e |       1
             l |     1
             l |   1
             o | 1
```

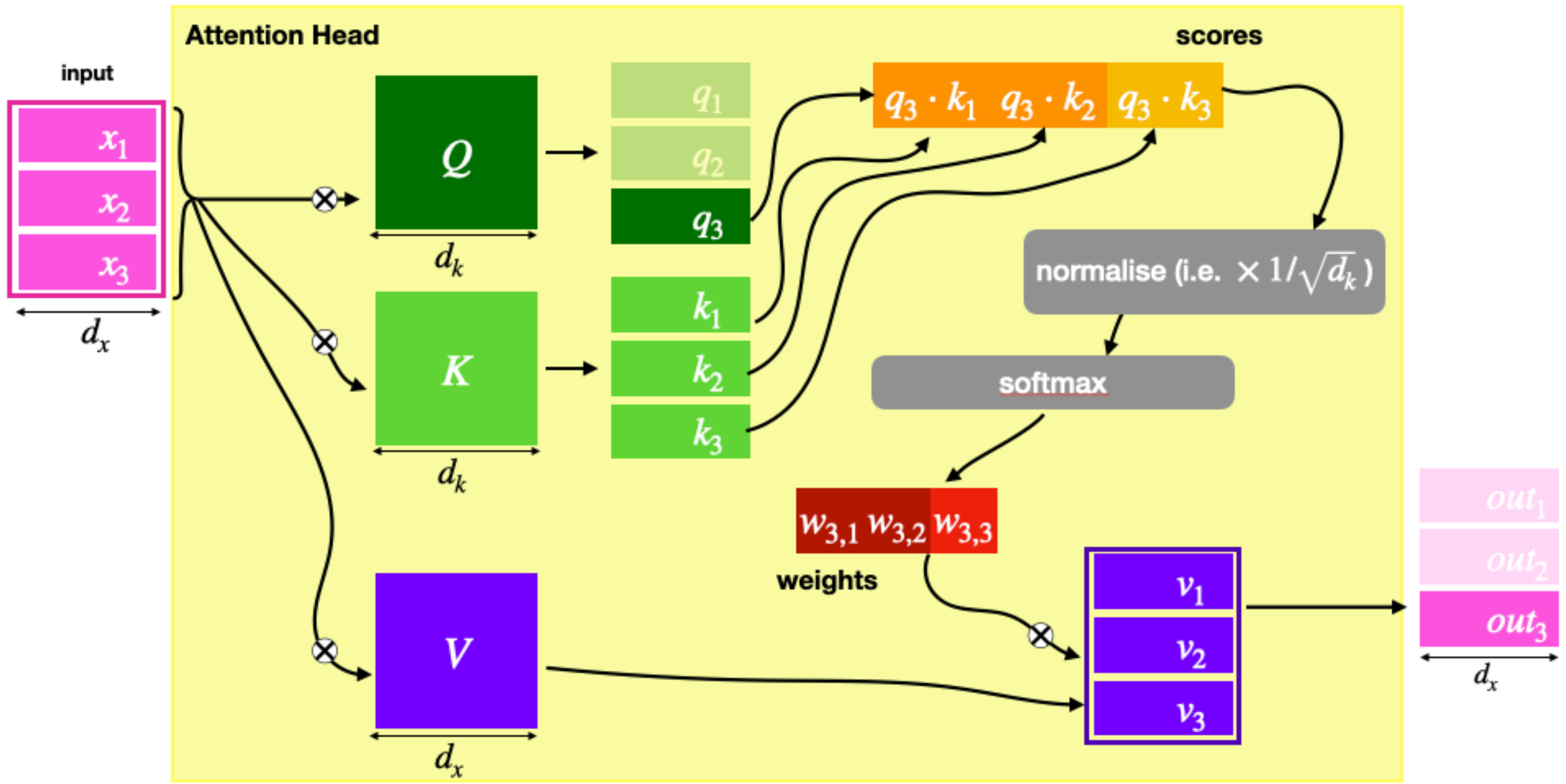
Example: Reverse




```
>> flip = select(length-indices-1,indices,==);
     selector: flip
         Example:
                  h e l l o
              h |         1
              e |       1
              l |     1
              l |   1
              o | 1

>> reverse = aggregate(flip,tokens);
```

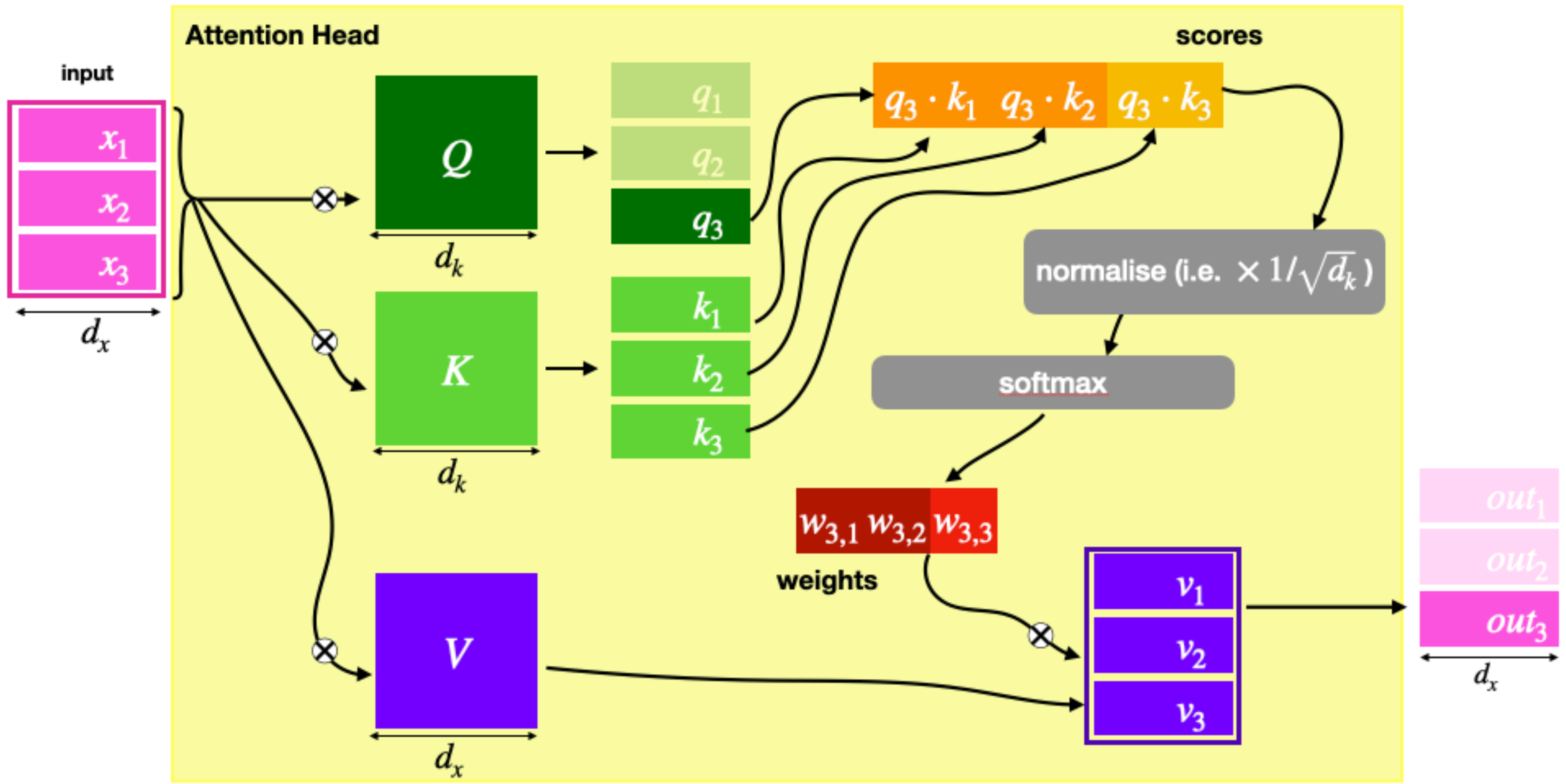
Example: Reverse




```
>> flip = select(length-indices-1,indices,==);
     selector: flip
         Example:
                    h e l l o
                h |         1
                e |       1
                l |     1
                l |   1
                o | 1

>> reverse = aggregate(flip,tokens);
     s-op: reverse
         Example: reverse("hello") = [o, l, l, e, h] (strings)
```

Example: Reverse

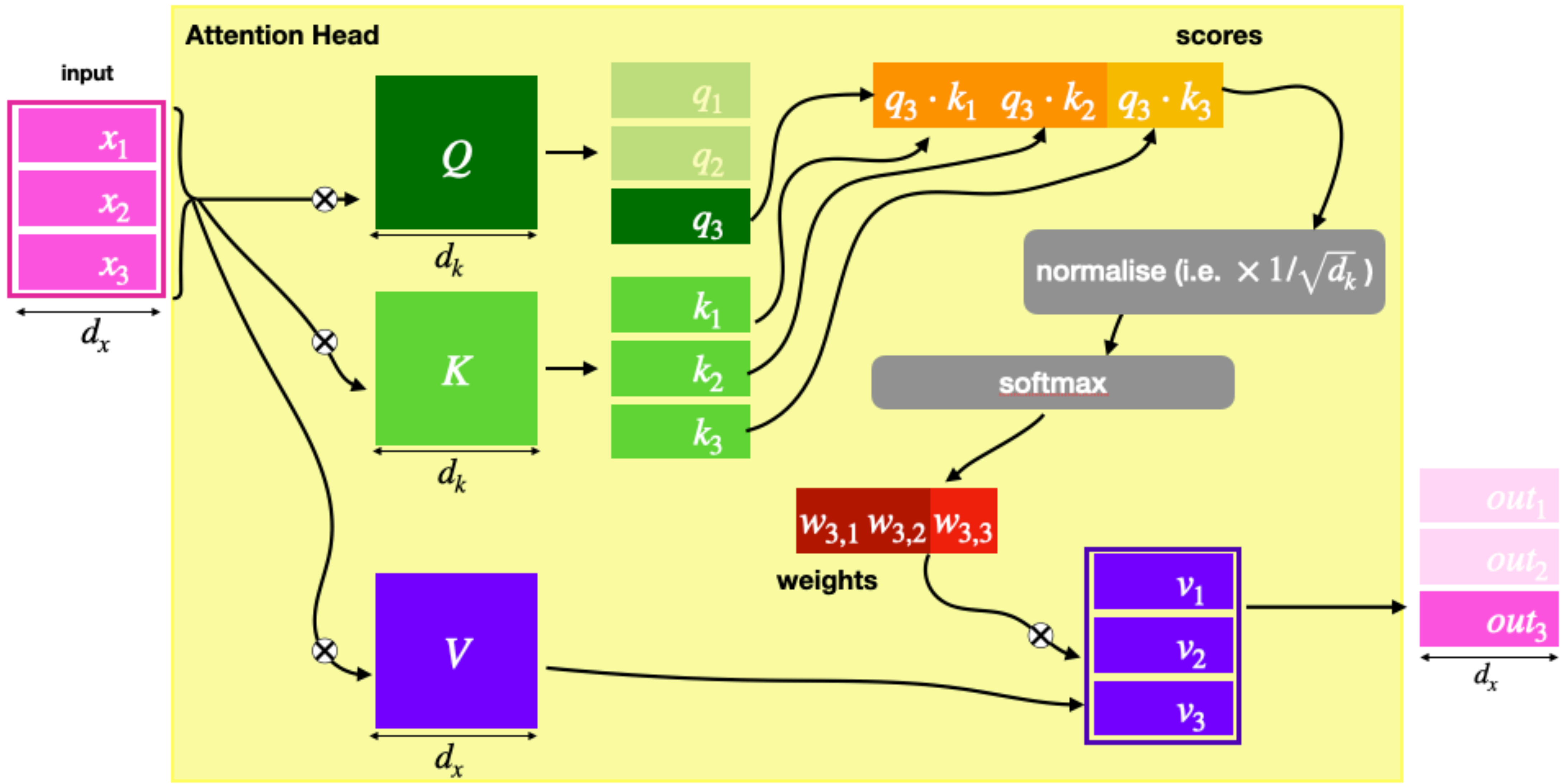



```
>> flip = select(length-indices-1,indices,==);
     selector: flip
         Example:
                       h e l l o
                   h |         1
                   e |       1
                   l |     1
                   l |   1
                   o | 1

>> reverse = aggregate(flip,tokens);
     s-op: reverse
         Example: reverse("hello") = [o, l, l, e, h] (strings)
```

See anything suspicious in the example?

Example: Reverse

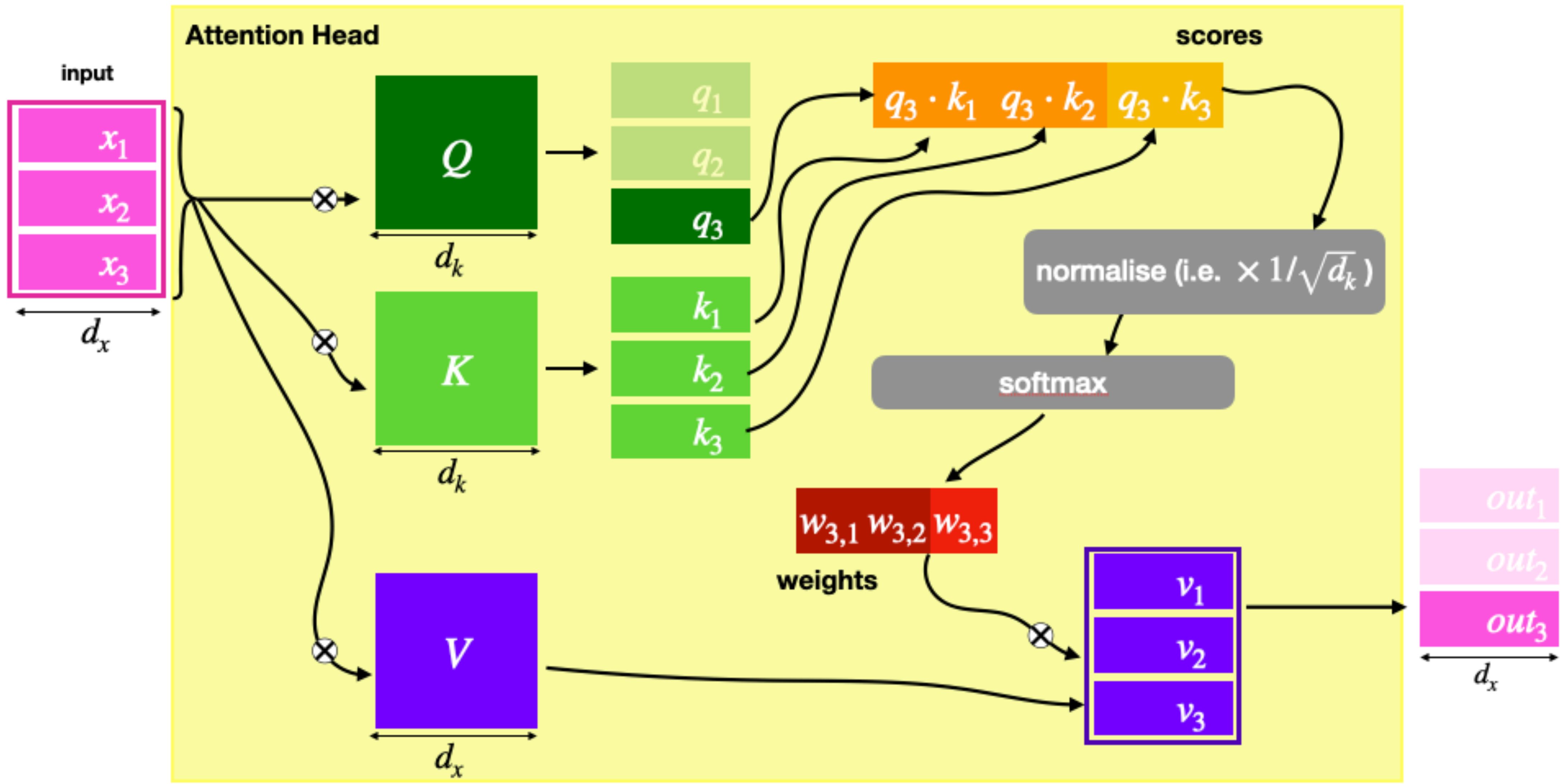



```
>> flip = select(length-indices-1,indices,==);
   selector: flip
       Example:
                     h e l l o
                 h |         1
                 e |       1
                 l |     1
                 l |   1
                 o | 1
>> reverse = aggregate(flip,tokens);
   s-op: reverse
       Example: reverse("hello") = [o, l, l, e, h] (strings)
```

See anything suspicious in the example?

It's length!

Example: Reverse




```
>> flip = select(length-indices-1,indices,==);
     selector: flip
         Example:
                     h e l l o
                 h |         1
                 e |       1
                 l |     1
                 l |   1
                 o | 1
>> reverse = aggregate(flip,tokens);
     s-op: reverse
         Example: reverse("hello") = [o, l, l, e, h] (strings)
```

The select decisions are pairwise!!

What would happen if they were arbitrarily powerful?

Transformer-Encoder Layer

Transformer-Encoder Layer

RASP (Restricted Access Sequence Processing)

Initial Sequences

```
>> tokens;
     s-op: tokens
         Example: tokens("hello") = [h, e, l, l, o] (strings)
>> indices;
     s-op: indices
         Example: indices("hello") = [0, 1, 2, 3, 4] (ints)
```

Elementwise application of atomic operations

```
>> indices+1;
     s-op: out
         Example: out("hello") = [1, 2, 3, 4, 5] (ints)
>> tokens=="e" or tokens=="o";
     s-op: out
         Example: out("hello") = [F, T, F, F, T] (bools)
```

Selectors, and aggregate

sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

new=aggregate(sel, [1,2,4])

			1 2 4				
F	T	T	1 2 4	=>	3		
F	F	F	1 2 4	=>	0	=>	[3,0,1]
T	F	F	1 2 4	=>	1		

```
>> flip = select(length-indices-1,indices,==);
     selector: flip
         Example:
                     h e l l o
                 h |         1
                 e |       1
                 l |     1
                 l |   1
                 o | 1
>> reverse = aggregate(flip,tokens);
     s-op: reverse
         Example: reverse("hello") = [o, l, l, e, h]
```

RASP Extras

Extra Sequences

```
>> length;
     s-op: length
         Example: length("hello") = [5]*5 (ints)
```

RASP Extras

Extra Sequences

```
>> length;
     s-op: length
         Example: length("hello") = [5]*5 (ints)
```

Selector Compositions

```
>> select(indices,3,==) or select(indices,indices,<=);
     selector: out
         Example:
                            h e l l o
                        h | 1     1
                        e | 1 1   1
                        l | 1 1 1 1
                        l | 1 1 1 1
                        o | 1 1 1 1 1
```

RASP Extras

Extra Sequences

```
>> length;
     s-op: length
         Example: length("hello") = [5]*5 (ints)
```

Functions

```
>> def in_range(min,val,max) {
..        return (min<=val) and (val<=max);
..    }
      console function: in_range(min, val, max)


>> in_range(1,indices,3);
      s-op: out
          Example: out("hello") = [F, T, T, T, F]
```

Selector Compositions

```
>> select(indices,3,==) or select(indices,indices,<=);
     selector: out
         Example:
                         h e l l o
                     h | 1     1
                     e | 1 1   1
                     l | 1 1 1 1
                     l | 1 1 1 1
                     o | 1 1 1 1 1
```

RASP Extras

Extra Sequences

```
>> length;
     s-op: length
         Example: length("hello") = [5]*5 (ints)
```

Functions

```
>> def in_range(min,val,max) {
..         return (min<=val) and (val<=max);
..    }
      console function: in_range(min, val, max)

>> in_range(1,indices,3);
     s-op: out
         Example: out("hello") = [F, T, T, T, F]
```

Selector Compositions

```
>> select(indices,3,==) or select(indices,indices,<=);
     selector: out
         Example:
                         h e l l o
                     h | 1       1
                     e | 1 1     1
                     l | 1 1 1 1
                     l | 1 1 1 1
                     o | 1 1 1 1 1
```

Library Functions

```
>> selector_width(select(tokens,tokens,==));
     s-op: out
         Example: out("hello") = [1, 1, 2, 2, 1] (ints)

>> count(tokens,"l");
     s-op: out
         Example: out("hello") = [2]*5 (ints)
```

RASP Extras

Extra Sequences

```
>> length;
     s-op: length
         Example: length("hello") = [5]*5 (ints)
```

Functions

```
>> def in_range(min,val,max) {
..         return (min<=val) and (val<=max);
..     }
     consc       tion: in_range(mi       max)

>            ngs   indices   ):

                      o                   T, F]
```

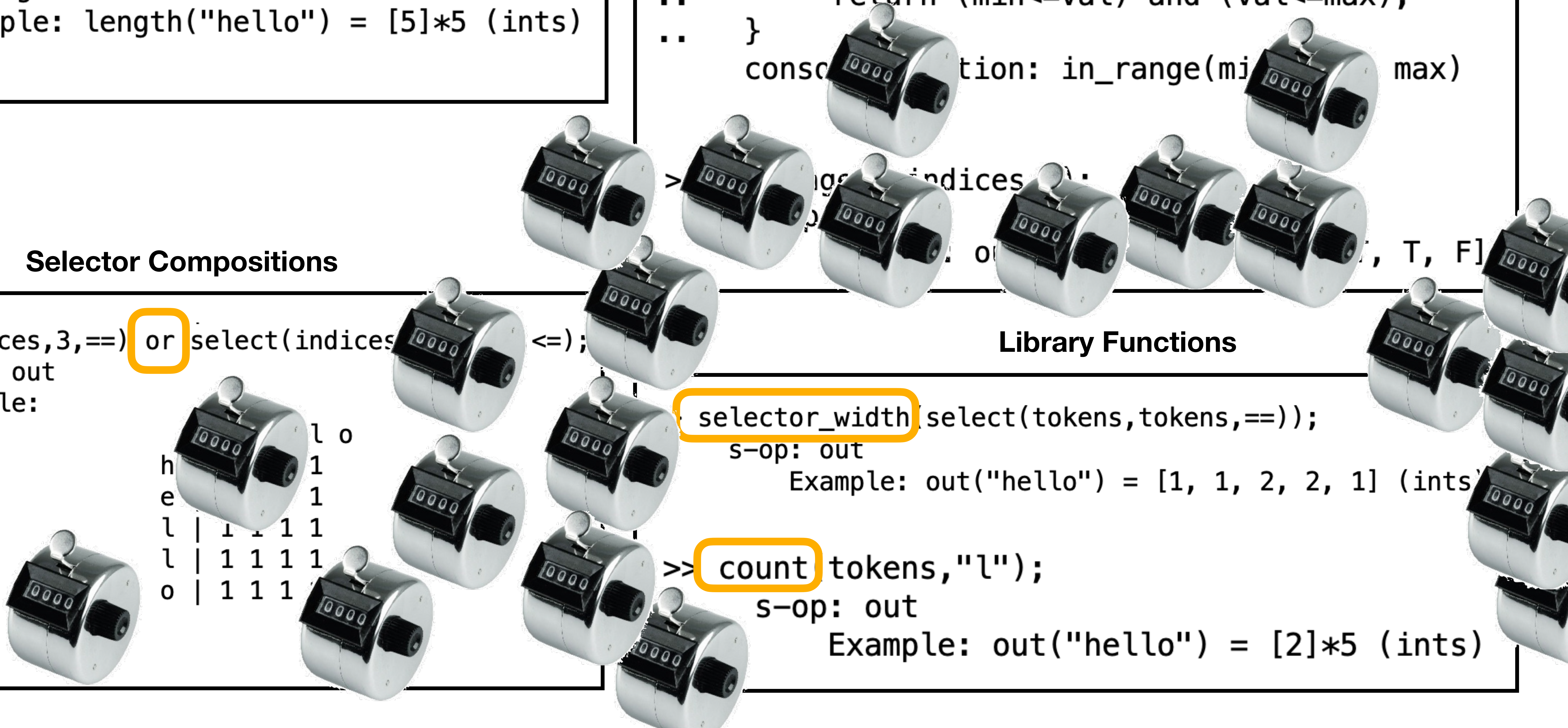
Selector Compositions

```
>> select(indices,3,==) or select(indices          <=);
     selector: out
         Example:
                                         l o
                            h            1
                            e            1
                            l | 1 1 1 1
                            l | 1 1 1 1
                            o | 1 1 1
```

Library Functions

```
 selector_width(select(tokens,tokens,==));
     s-op: out
         Example: out("hello") = [1, 1, 2, 2, 1] (ints)

>> count(tokens,"l");
     s-op: out
         Example: out("hello") = [2]*5 (ints)
```



Small Example

Computing *length*:

Small Example

Computing *length*:

```
>> full_s = select(1,1,==);
     selector: full_s
         Example:
                              h e l l o
                          h | 1 1 1 1 1
                          e | 1 1 1 1 1
                          l | 1 1 1 1 1
                          l | 1 1 1 1 1
                          o | 1 1 1 1 1
```

Small Example

Computing *length*:

```
>> full_s = select(1,1,==);
     selector: full_s
         Example:
                                h e l l o
                            h | 1 1 1 1 1
                            e | 1 1 1 1 1
                            l | 1 1 1 1 1
                            l | 1 1 1 1 1
                            o | 1 1 1 1 1
                              indicator(indices==0)
```

```
>> indicator(indices==0);
     s-op: out
         Example: out("hello") = [1, 0, 0, 0, 0] (ints)
```

Small Example

Computing *length*:

```
>> full_s = select(1,1,==);
     selector: full_s
         Example:
                             h e l l o
                         h | 1 1 1 1 1
                         e | 1 1 1 1 1
                         l | 1 1 1 1 1
                         l | 1 1 1 1 1
                         o | 1 1 1 1 1
>> frac_0=aggregate(full_s,indicator(indices==0));
     s-op: frac_0
         Example: frac_0("hello") = [0.2]*5 (floats)
```

```
>> indicator(indices==0);
     s-op: out
         Example: out("hello") = [1, 0, 0, 0, 0] (ints)
```

Small Example

Computing *length*:

```
>> full_s = select(1,1,==);
     selector: full_s
         Example:
                              h e l l o
                          h | 1 1 1 1 1
                          e | 1 1 1 1 1
                          l | 1 1 1 1 1
                          l | 1 1 1 1 1
                          o | 1 1 1 1 1
>> frac_0=aggregate(full_s,indicator(indices==0));
     s-op: frac_0
         Example: frac_0("hello") = [0.2]*5 (floats)
>> round(1/frac_0);
     s-op: out
         Example: out("hello") = [5]*5 (ints)
```

```
>> indicator(indices==0);
     s-op: out
         Example: out("hello") = [1, 0, 0, 0, 0] (ints)
```

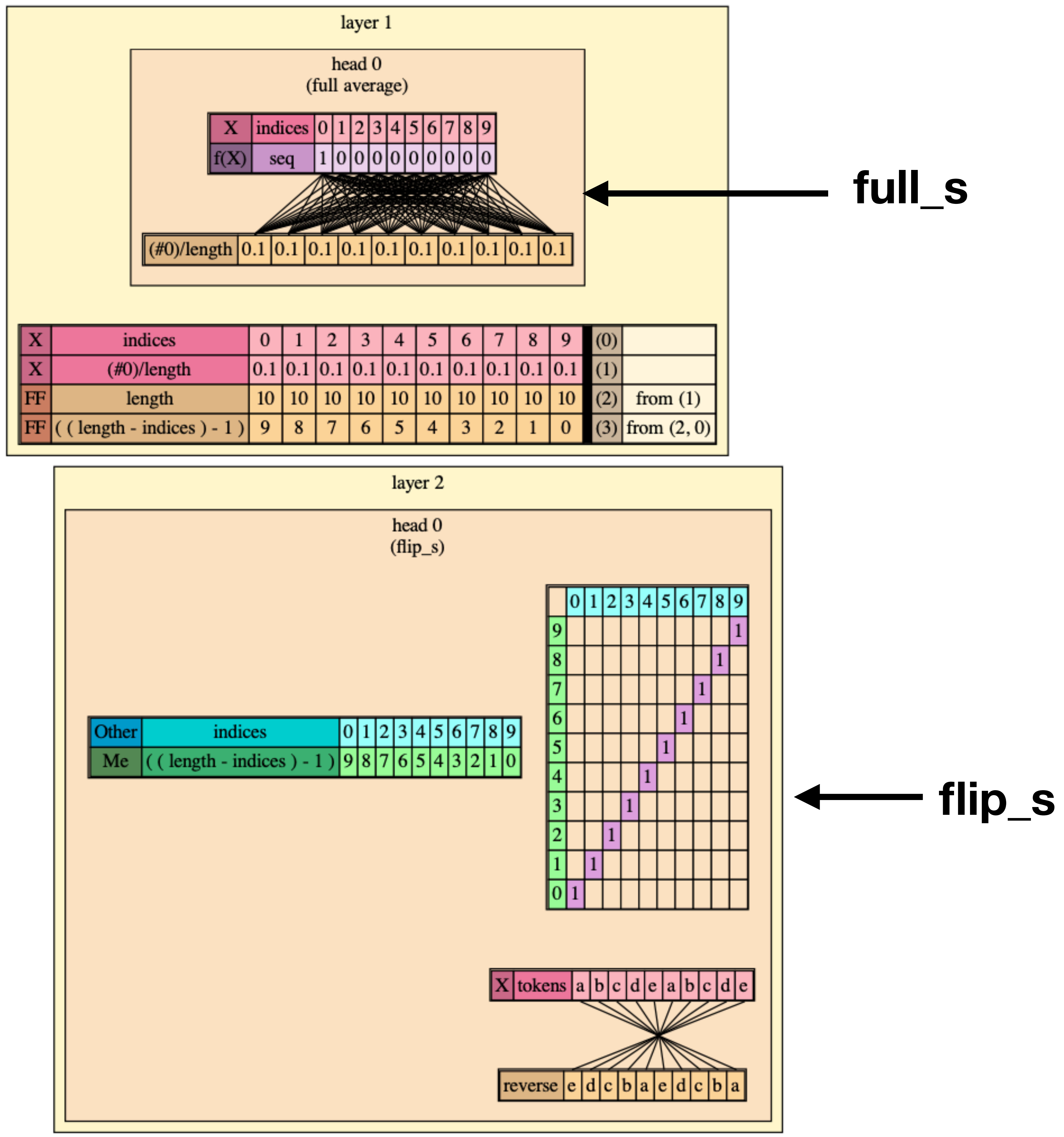
Connection to Reality?

Are our RASP programs predicting the right number of layers?

Are our RASP programs predicting relevant selector patterns?

Connection to Reality?

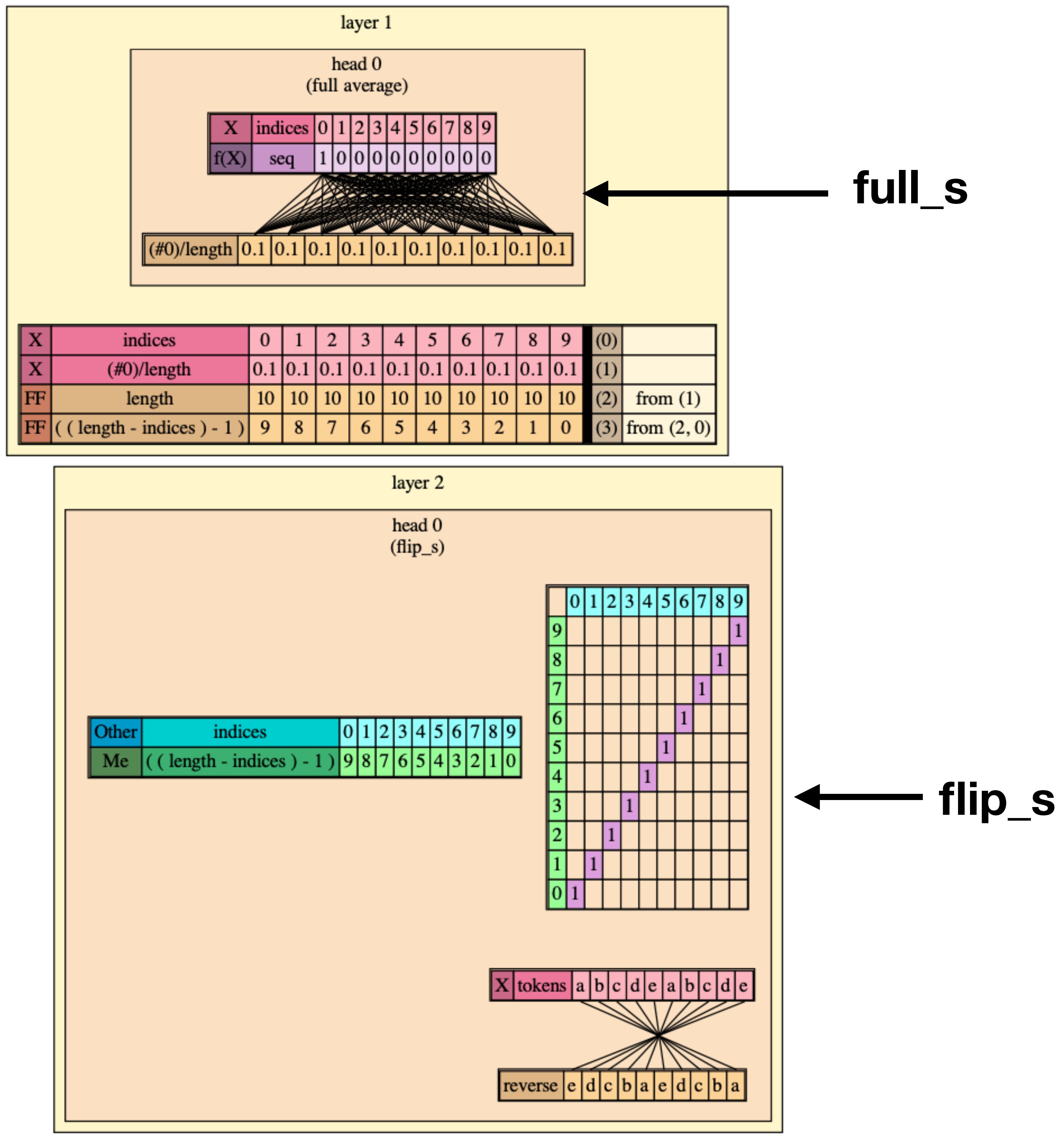
```
>> draw(reverse,"abcdeabcde")
```



RASP expects 2 layers for arbitrary-length reverse

Connection to Reality?

```
>> draw(reverse,"abcdeabcde")
```



RASP expects 2 layers for arbitrary-length reverse

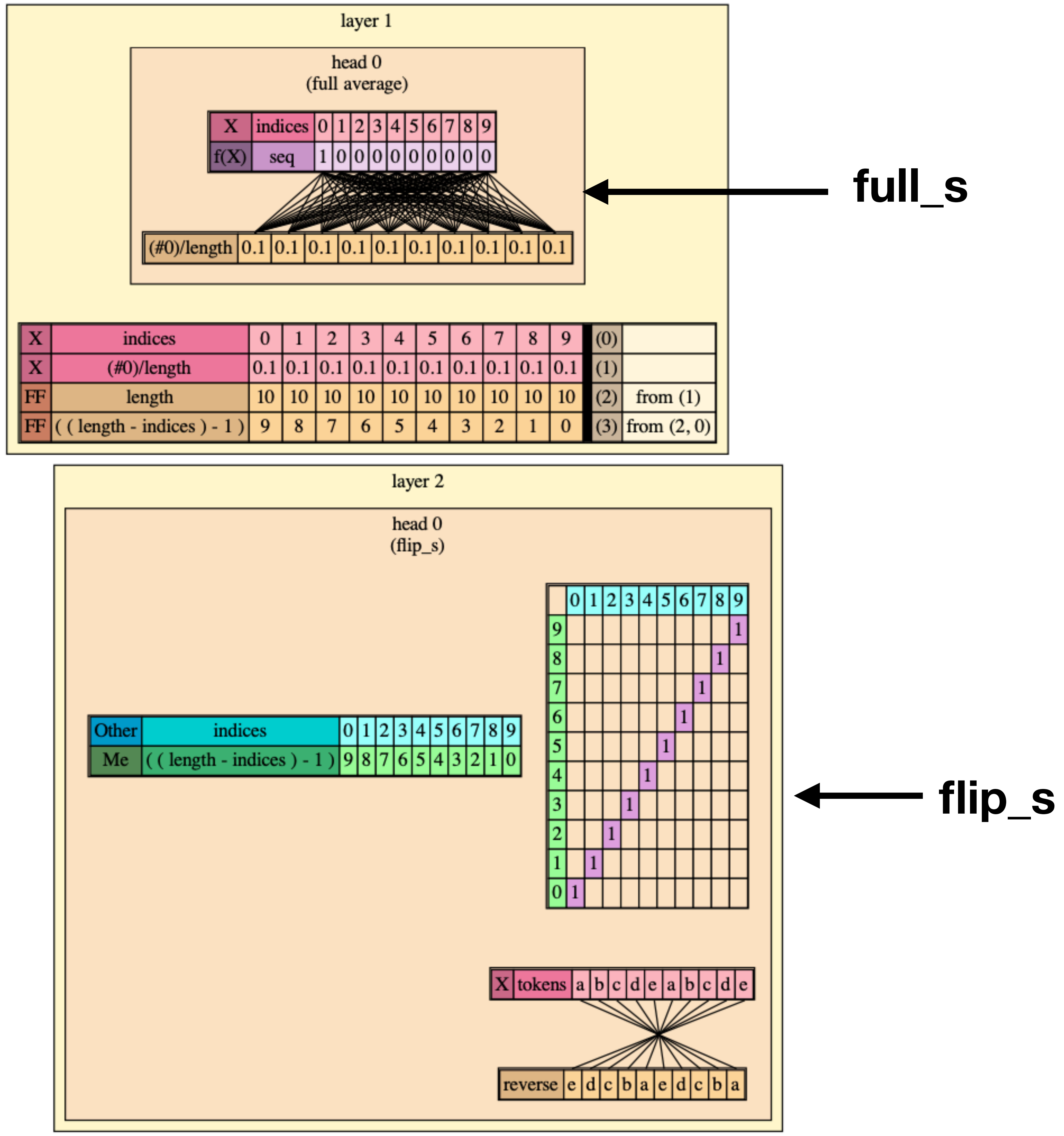
Test:
Training small transformers on lengths 0-100:

2 layers: **99.6**% accuracy after 20 epochs
1 layer: **39.6**% accuracy after 50 epochs

Even with compensation for
number of heads and parameters!

Connection to Reality?

```
>> draw(reverse,"abcdeabcde")
```



RASP expects 2 layers for arbitrary-length reverse

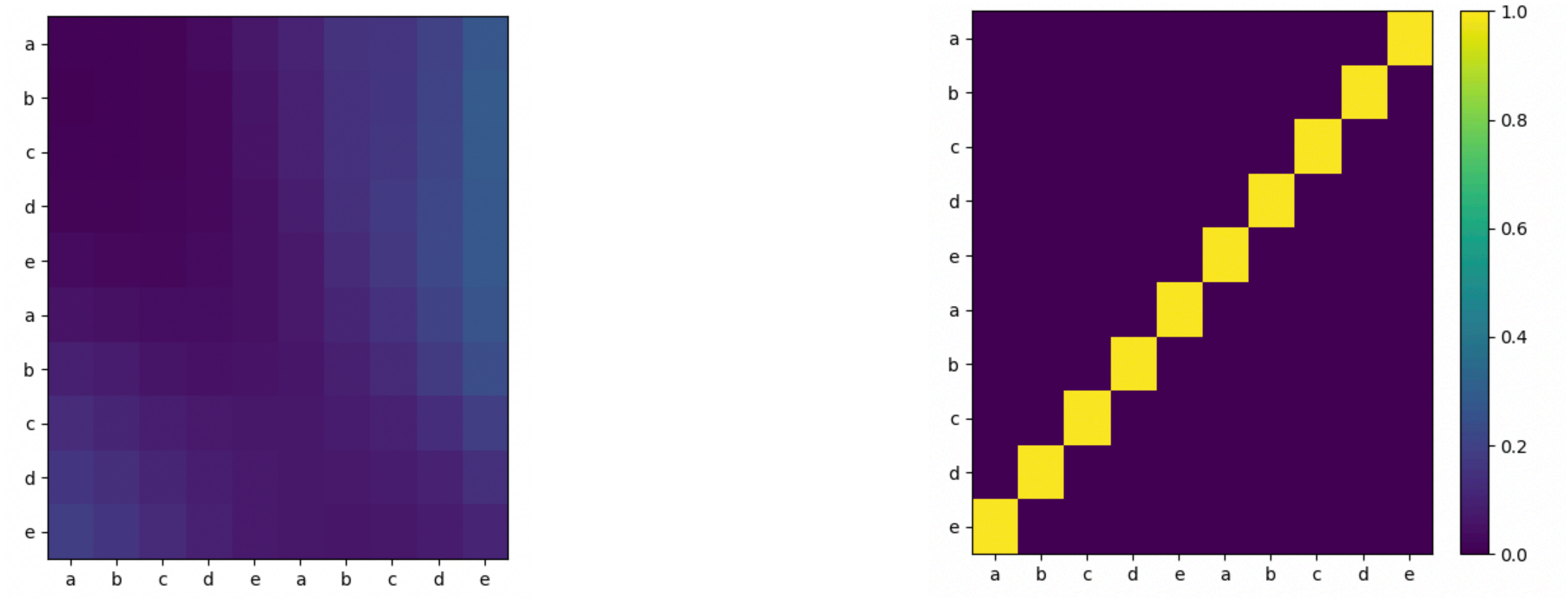
Test:
Training small transformers on lengths 0-100:

2 layers: **99.6**% accuracy after 20 epochs
1 layer: **39.6**% accuracy after 50 epochs

Bonus: the 2 layer transformer's attention patterns:

Layer 1 (*full_s*)

Layer 2 (*flip_s*)



Connection to Reality?

Example 2: ***histogram*** (assuming BOS)

Eg:

[§,h,e,l,l,o] $\mapsto$ [0,1,1,2,2,1]

[§,a,b,a] $\mapsto$ [0,2,1,2]

[§,a,b,c,c,c] $\mapsto$ [0,1,1,3,3,3]

Connection to Reality?

Example 2: *histogram* (assuming BOS)

Eg:

[§,h,e,l,l,o] $\mapsto$ [0,1,1,2,2,1]

[§,a,b,a] $\mapsto$ [0,2,1,2]

[§,a,b,c,c,c] $\mapsto$ [0,1,1,3,3,3]

```
>> selector_width(select(tokens,tokens,==));
     s-op: out
         Example: out("hello") = [1, 1, 2, 2, 1] (ints)
```

Connection to Reality?

Example 2: *histogram* (assuming BOS)

```
>> set example "§hello"
>> same_or_0 = select(tokens,tokens,==)
     selector: same_or_0
         Example:
                     § h e l l o
                 § | 1
                 h |   1
                 e |     1
                 l |       1 1
                 l |       1 1
                 o |           1
```

Connection to Reality?

Example 2: *histogram* (assuming BOS)

```
>> set example "§hello"
>> same_or_0 = select(tokens,tokens,==) or select(indices,0,==);
     selector: same_or_0
         Example:
                     § h e l l o
                 § | 1
                 h | 1 1
                 e | 1   1
                 l | 1     1 1
                 l | 1     1 1
                 o | 1         1
```

Connection to Reality?

Example 2: *histogram* (assuming BOS)

```
>> set example "§hello"
>> same_or_0 = select(tokens,tokens,==) or select(indices,0,==);
     selector: same_or_0
         Example:
                             § h e l l o
                         § | 1
                         h | 1 1
                         e | 1   1
                         l | 1     1 1
                         l | 1     1 1
                         o | 1         1
>> frac_with_0 = aggregate(same_or_0,indicator(indices==0));
     s-op: frac_with_0
         Example: frac_with_0("§hello") = [1, 0.5, 0.5, 0.333, 0.333, 0.5] (floats)
```

Connection to Reality?

Example 2: *histogram* (assuming BOS)

```
>> set example "§hello"
>> same_or_0 = select(tokens,tokens,==) or select(indices,0,==);
     selector: same_or_0
         Example:
                                          § h e l l o
                                      § | 1
                                      h | 1 1
                                      e | 1   1
                                      l | 1     1 1
                                      l | 1     1 1
                                      o | 1         1
>> frac_with_0 = aggregate(same_or_0,indicator(indices==0));
     s-op: frac_with_0
         Example: frac_with_0("§hello") = [1, 0.5, 0.5, 0.333, 0.333, 0.5] (floats)
>> histogram_assuming_bos = round(1/frac_with_0)-1;
     s-op: histogram_assuming_bos
         Example: histogram_assuming_bos("§hello") = [0, 1, 1, 2, 2, 1] (ints)
```

Connection to Reality?

Example 2: *histogram* (assuming BOS)

```
>> examples off
>> same_or_0 = select(tokens,tokens,==) or select(indices,0,==);
     selector: same_or_0
>> frac_with_0 = aggregate(same_or_0,indicator(indices==0));
     s-op: frac_with_0
>> histogram_assuming_bos = round(1/frac_with_0)-1;
     s-op: histogram_assuming_bos
>> histogram_assuming_bos("§hello");
         =  [0, 1, 1, 2, 2, 1] (ints)
```

RASP analysis:

- Just one attention head
- It focuses on:
  1. All positions with same token, and:
  2. Position 0 (regardless of content)

Connection to Reality?

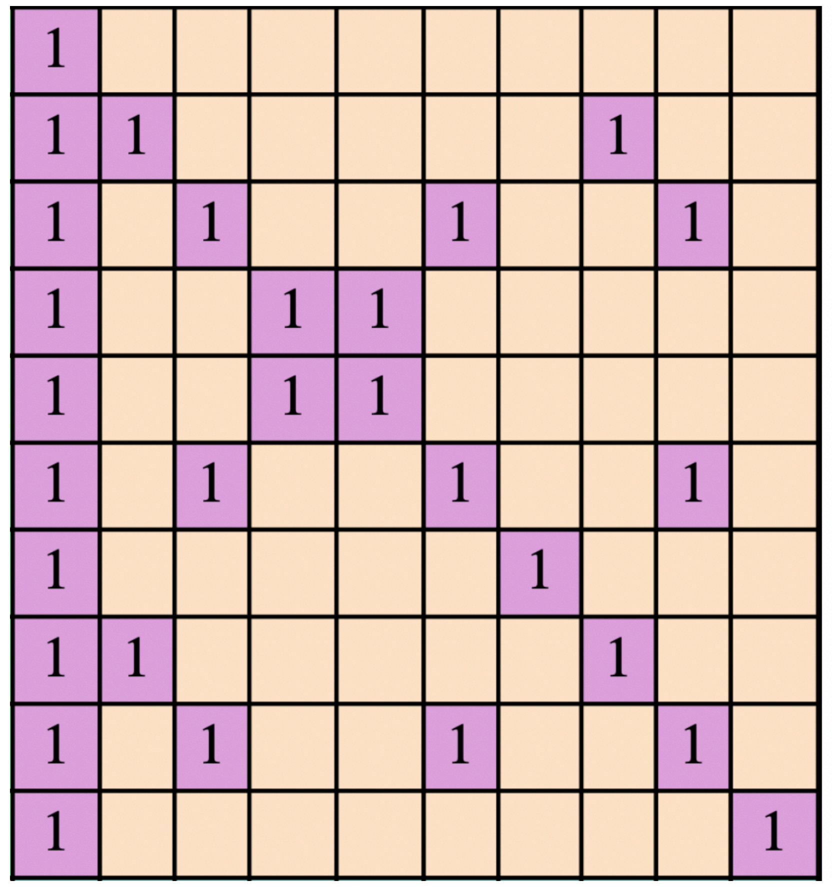
Example 2: *histogram* (assuming BOS)

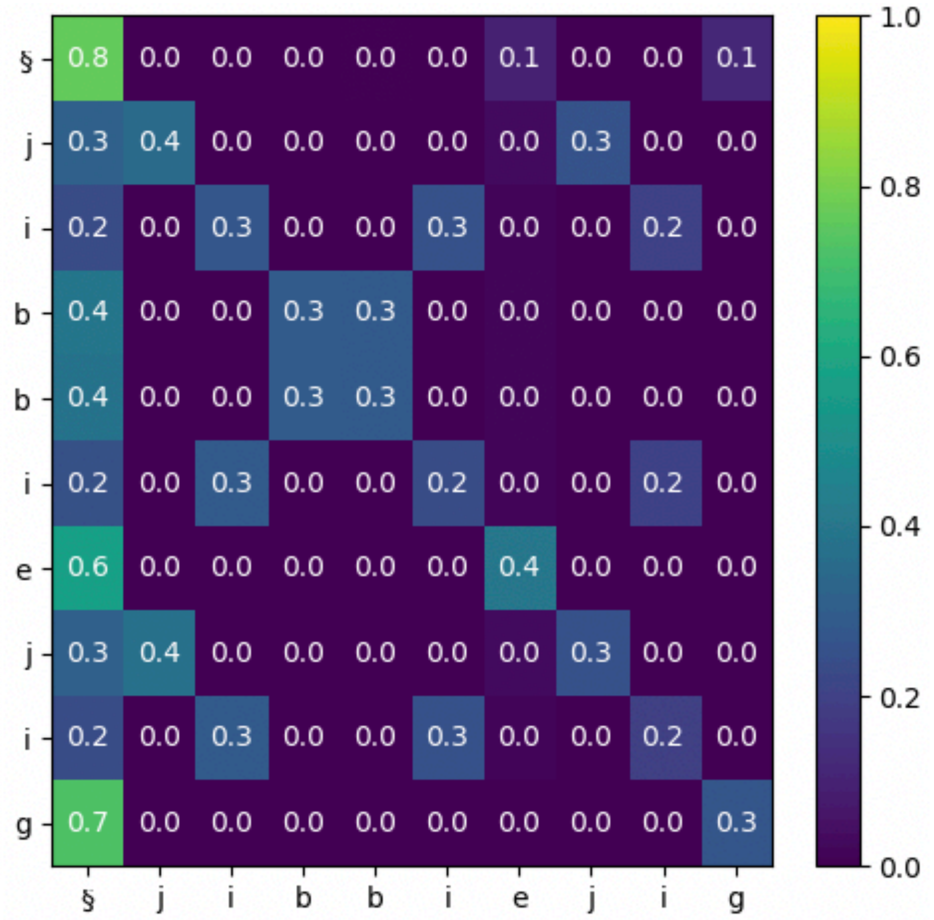
```
>> examples off
>> same_or_0 = select(tokens,tokens,==) or select(indices,0,==);
     selector: same_or_0
>> frac_with_0 = aggregate(same_or_0,indicator(indices==0));
     s-op: frac_with_0
>> histogram_assuming_bos = round(1/frac_with_0)-1;
     s-op: histogram_assuming_bos
>> histogram_assuming_bos("§hello");
         =  [0, 1, 1, 2, 2, 1] (ints)
```

RASP analysis:

- Just one attention head
- It focuses on:
    1. All positions with same token, and:
    2. Position 0 (regardless of content)

Selector pattern vs trained transformer's attention for same input sequence:





RASP (Restricted Access Sequence Processing)

Initial Sequences

```
>> tokens;
     s-op: tokens
         Example: tokens("hello") = [h, e, l, l, o] (strings)
>> indices;
     s-op: indices
         Example: indices("hello") = [0, 1, 2, 3, 4] (ints)
```

Elementwise application of atomic operations

```
>> indices+1;
     s-op: out
         Example: out("hello") = [1, 2, 3, 4, 5] (ints)
>> tokens=="e" or tokens=="o";
     s-op: out
         Example: out("hello") = [F, T, F, F, T] (bools)
```

Selectors, and aggregate

sel = select([2,0,0],[0,1,2],==)

```
   2 0 0
0  F T T
1  F F F
2  T F F
```

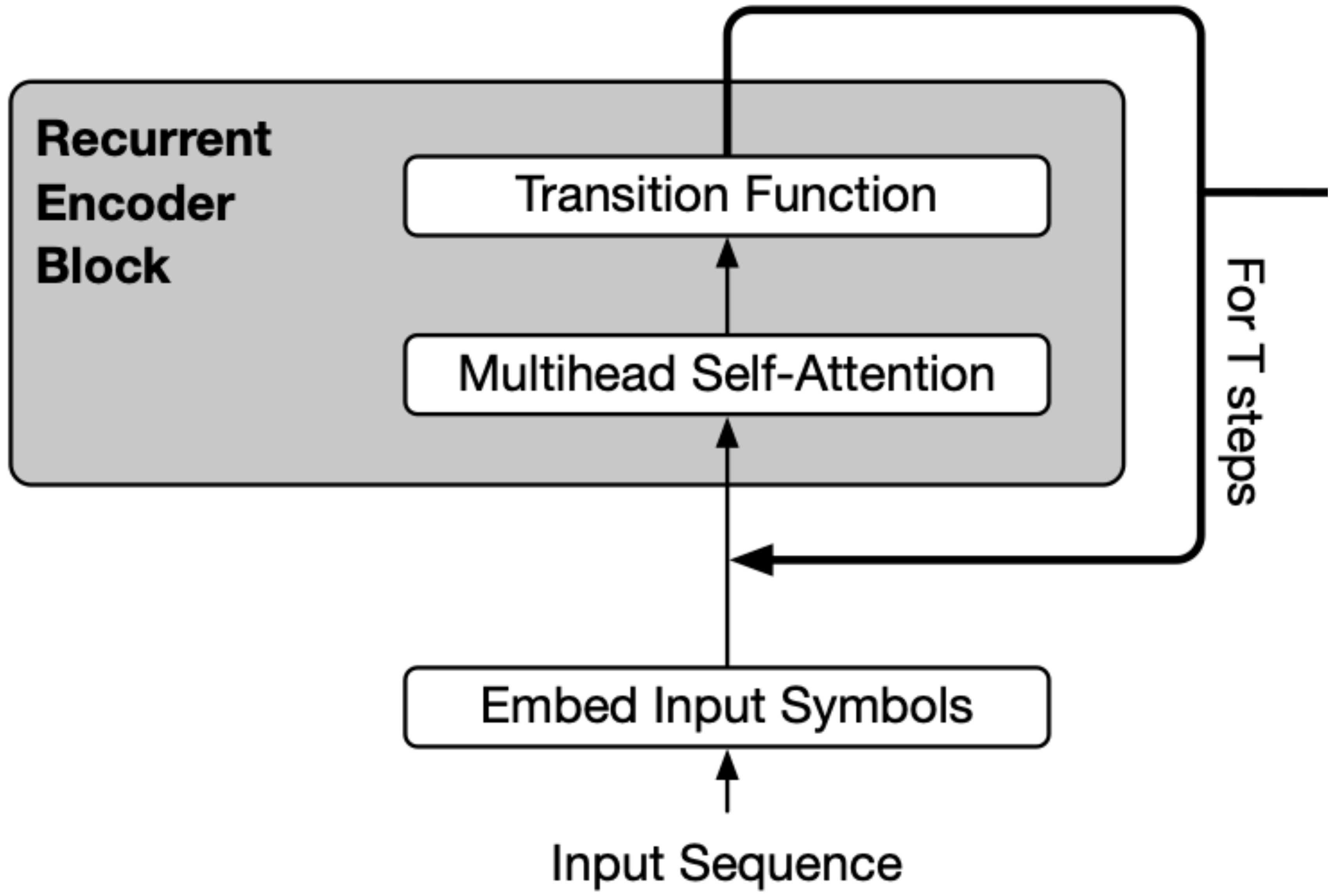
new=aggregate(sel, [1,2,4])

```
         1 2 4
F T T    1 2 4  =>  3
F F F    1 2 4  =>  0  =>  [3,0,1]
T F F    1 2 4  =>  1
```

```
>> flip = select(length-indices-1,indices,==);
     selector: flip
         Example:
                     h e l l o
                 h |         1
                 e |       1
                 l |     1
                 l |   1
                 o | 1
>> reverse = aggregate(flip,tokens);
     s-op: reverse
         Example: reverse("hello") = [o, l, l, e, h]
```

Insight

1. Further motivates the *Universal Transformer*



Recurrent blocks are like allowing loops in RASP!

Universal Transformers

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, Łukasz Kaiser

Insight

2. Explains results of the *Sandwich Transformer*

Improving Transformer Models by Reordering their Sublayers

Ofir Press, Noah A. Smith, Omer Levy

If re-ordering and switching attention and feed-forward layers of a transformer (while adjusting to keep same number of parameters):

1. Better to have attention earlier, and feed-forward later
2. Only attention not enough

Model	**PPL** ↓
sfffssfsfsfssffffsfsffsffffff	22.80
sffssfsssssssssssssfsfsssfsffsssfsssfs	21.02
ssssssffsffffssfffffsssfsfsssssssss	20.98
fffffffffsffssffsffssssfsfsssf	20.75
fssfsssffffffssfsssfsfffssssfsfss	20.43
sffsffffffsfsfssfsssfsfsfssfssfs	20.28
sffssffsfffsfsfssssffffffssssff	20.02
fsffsfssffffsfsfffsfffssfffsss	19.93
sffsffssffsfsffsssfsssssfsssfffsss	19.85
ssfffffffssfffssfssffsfsfsffsf	19.82
sfsfsfffsfffssfsfffsffssfsfsfss	19.77
sfsffsssffsffsssfssfffffssssfsssf	19.55
sffsfssfffsffsfssssfsfsffffsfsss	19.49
sffffsffssssfsssfssfffsssfssssfsfs	19.47
fsssffssssssfsfsfsffsffffssfsfssss	19.25
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**19.13**
fssssssfsfsfsfffsfsssfssffssssfsff	18.86
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.83**
ssfsfsssfsssssffsfsfsssfssfsfsssssssf	18.62
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.54**
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.49**
sssfsffsfssfsssffsffffffssfsfff	18.34
sssfsfsffsssfsfffffsfsffffsssff	18.31
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.25**
ssssssfsssffffsfsfffffffffffsf	18.12

s self-attention

f feed-forward

Insight

3. Transformers can "use" at least $n\log(n)$ of the n^2 computational cost they have:

selector_width can be used to implement sort:

```
>> selector examples off
>> earlier_token = select(tokens,tokens,<) or
..          (select(tokens,tokens,==) and select(indices,indices,<));
     selector: earlier_token
>> num_prev = selector_width(earlier_token);
     s-op: num_prev
         Example: num_prev("hello") = [1, 0, 2, 3, 4] (ints)
>> sorted = aggregate(select(num_prev,indices,==),tokens);
     s-op: sorted
         Example: sorted("hello") = [e, h, l, l, o] (strings)
```

which we know requires at least $n\log(n)$ operations
(if making no assumptions on input data)

Open Question: is there something that "uses" *all* n^2 of the attention head cost?

Tracr

Researchers at Deepmind built an actual compiler for (a large subset of) RASP!!?!

Tracr: Compiled Transformers as a Laboratory for Interpretability

David Lindner, János Kramár, Matthew Rahtz, Thomas McGrath, Vladimir Mikulik

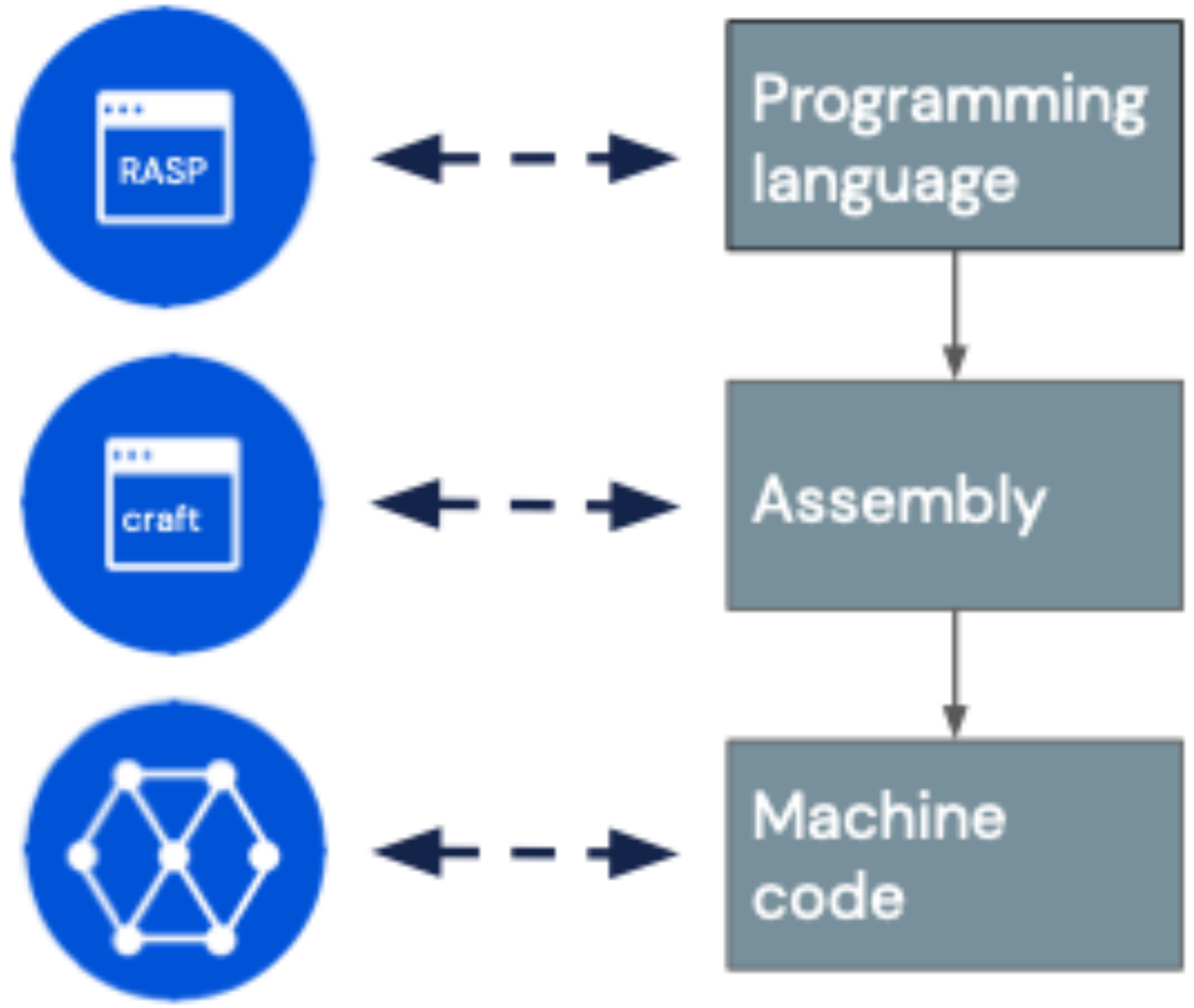


Figure 3 | `Tracr` translates RASP to `craft` and then to model weights, analogous to how programming languages are first translated to assembly then to machine code.

End

Try it out!

🌟 github.com/tech-srl/RASP 🌟

(or email me if you can't get on github)

Do a challenge!

🌟 https://srush.github.io/raspy/ 🌟

"Thinking Like Transformers" - ICML 2021
(Available on Arxiv)

Optional Talking Points

- Bhattamishra et al (2020) note that, unlike LSTMs, transformers struggle with some regular languages. Why might that be? (What would a general method for encoding a DFA in a transformer be?)

- Hahn (2019) proves that transformers with hard attention cannot compute Parity with hard attention. RASP can compute parity. What is the difference?

- How should we convert a RASP program to ‘real’ transformers? How big does our head-dimension need to be for “select(indices,indices,<)”? How do we implement *and*, *or*, and *not* between selectors?

- Do our selectors cover all the possible attention patterns? What is missing?

- How can aggregating on no positions be achieved in a transformer?