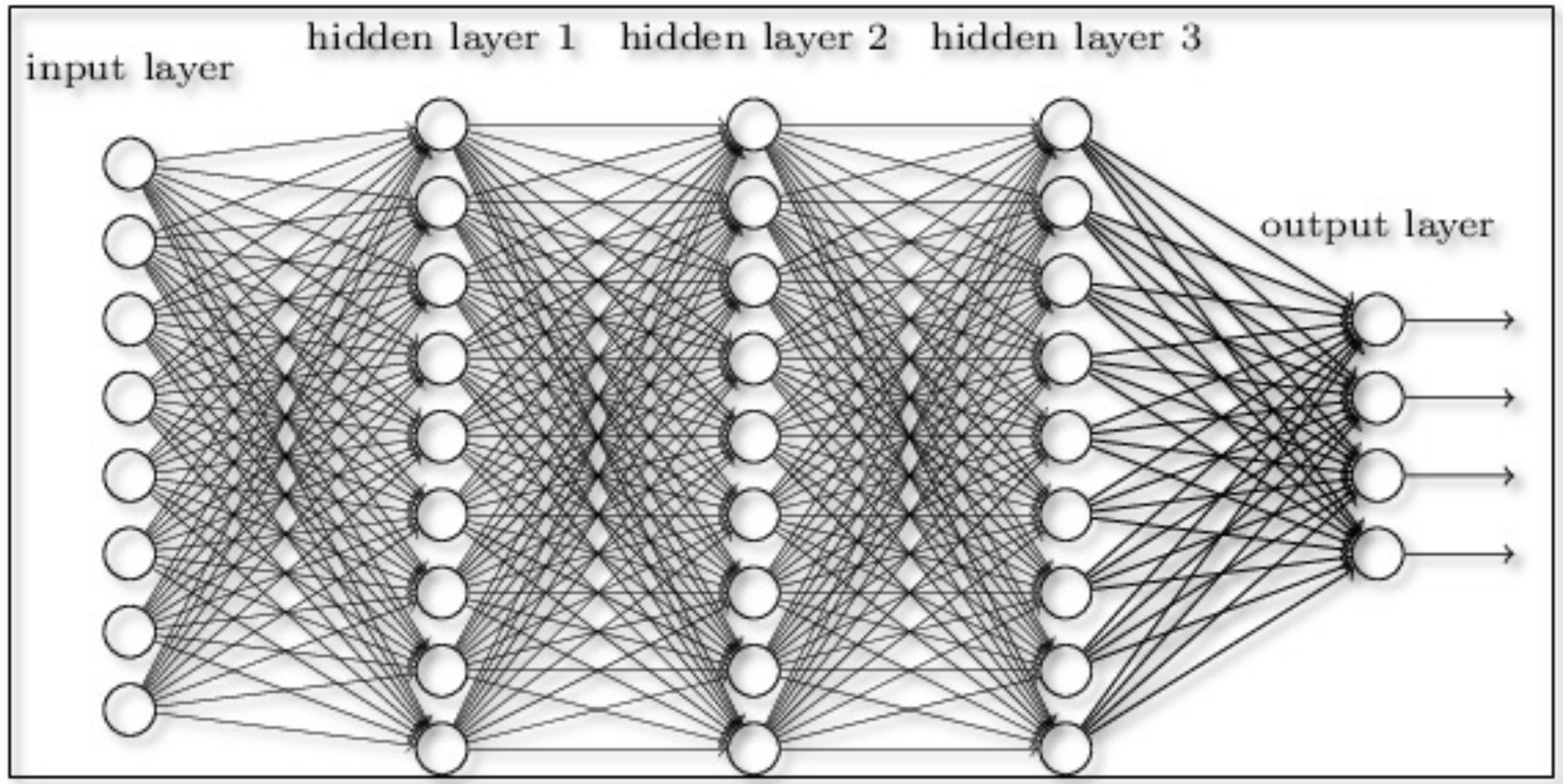
Neural Sequence Models: A Formal Lens

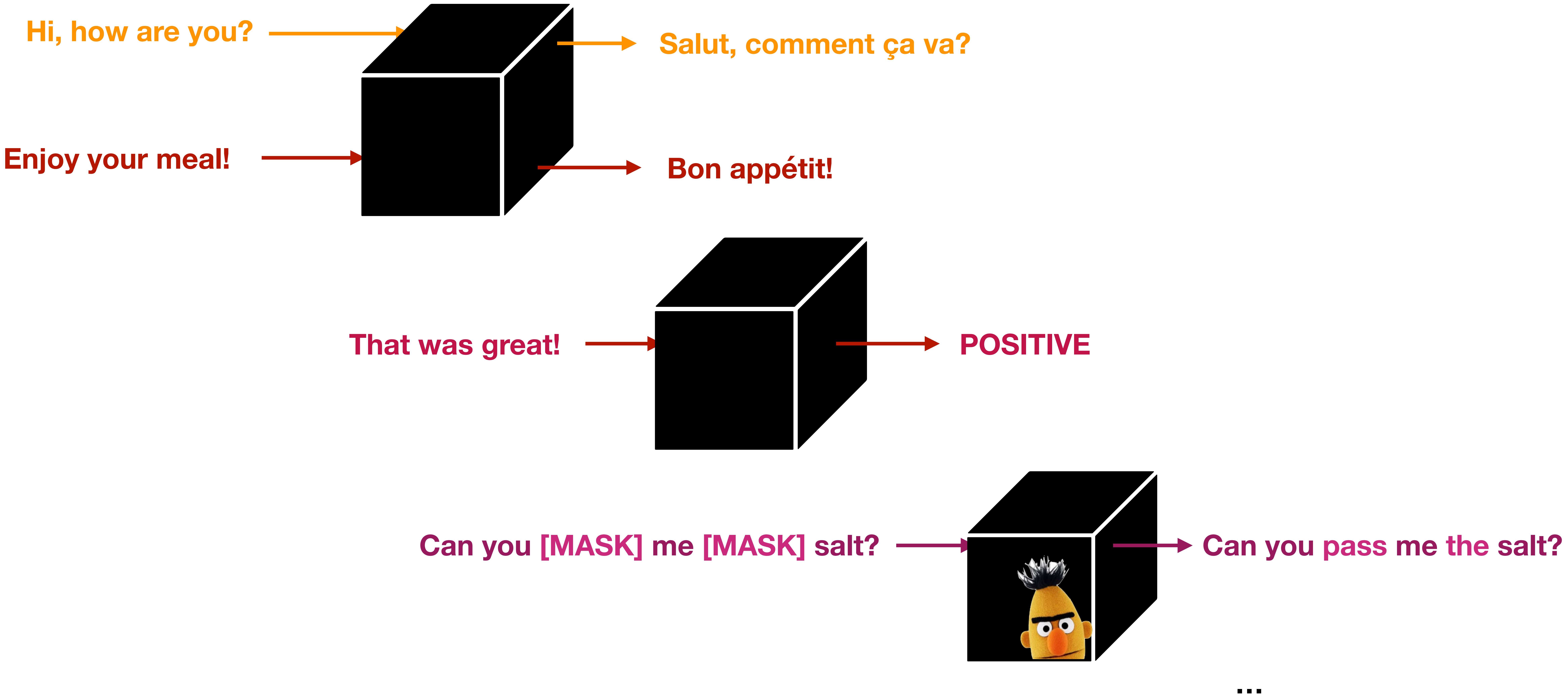
Gail Weiss

Yoav Goldberg, Eran Yahav

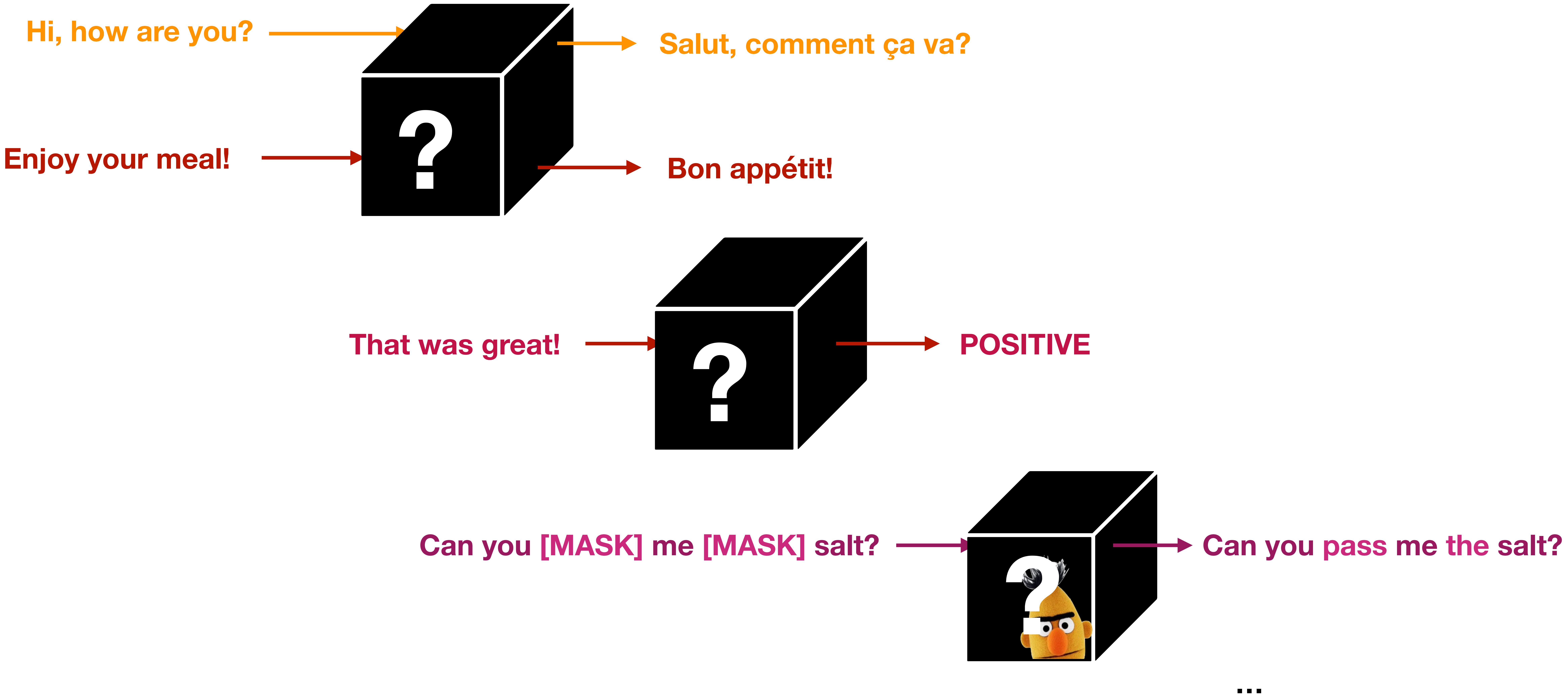


= ?

Neural Sequence Models

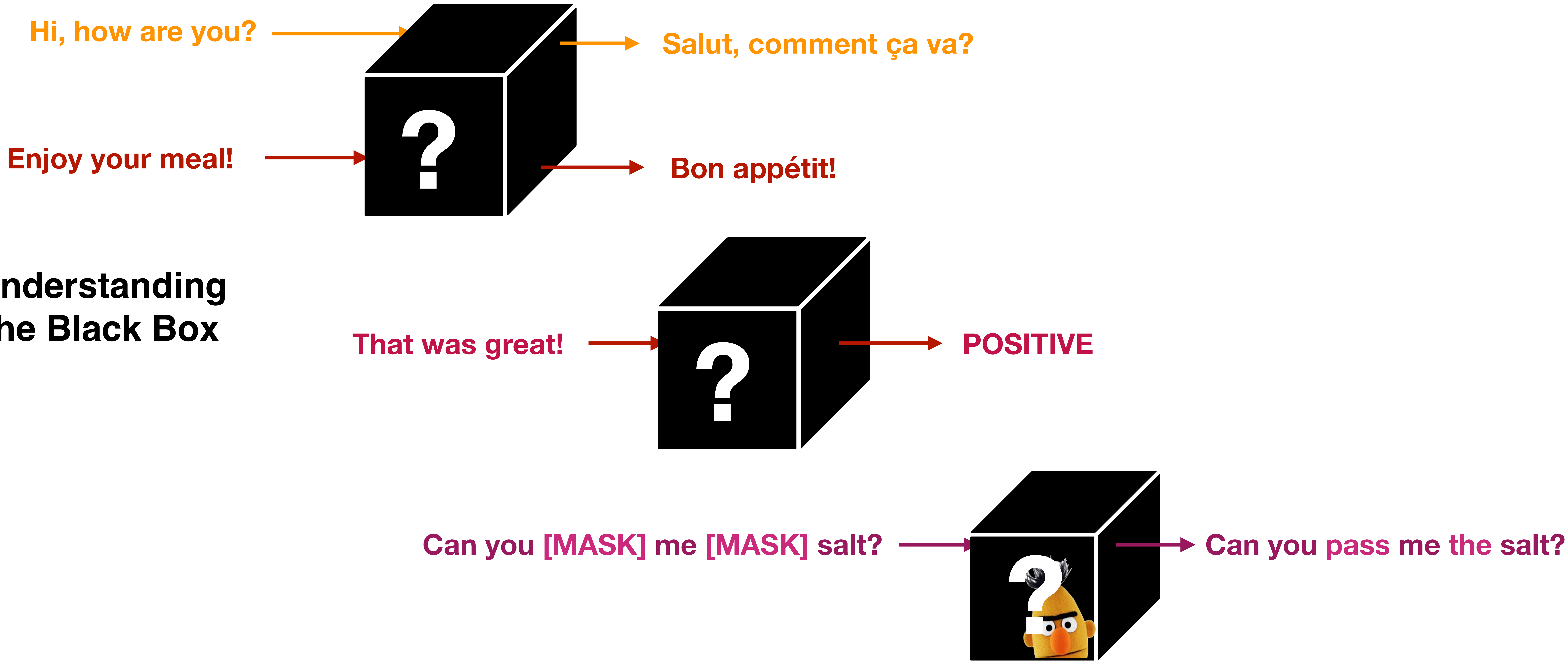


Neural Sequence Models



Neural Sequence Models

Understanding the Black Box

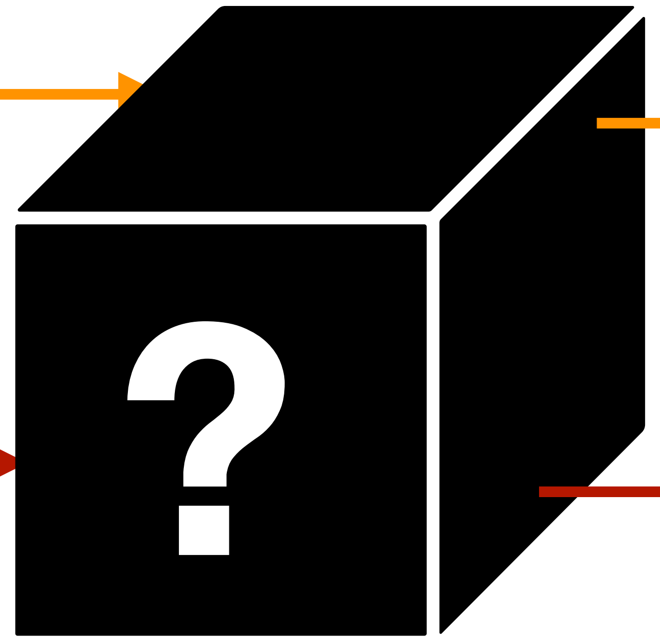


...

Neural Sequence Models

Hi, how are you? → Salut, comment ça va?

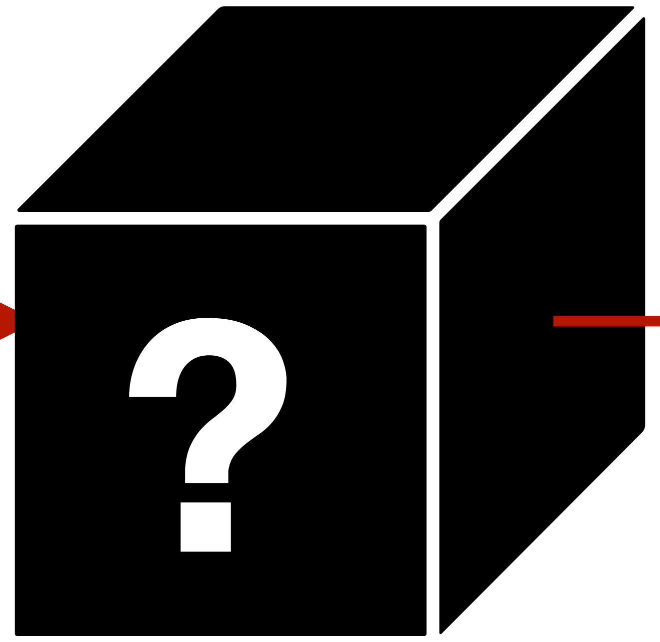
Enjoy your meal! → Bon appétit!



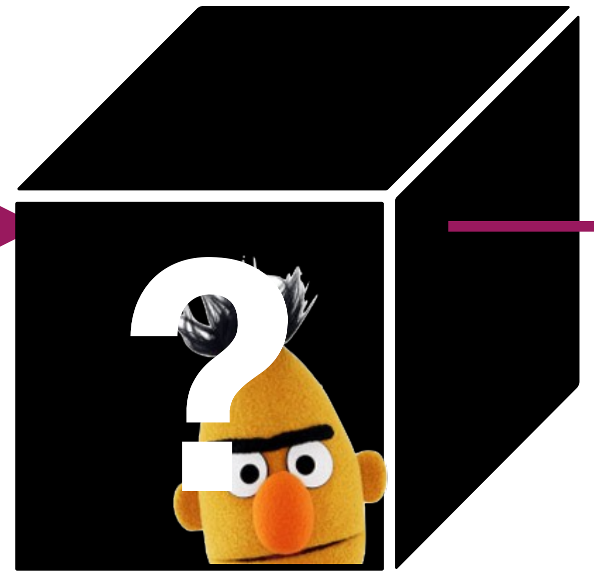
Understanding the Black Box

- Reliability (Verifiability)
- Intuition (biases, model selection)
- Knowledge extraction

That was great! → POSITIVE

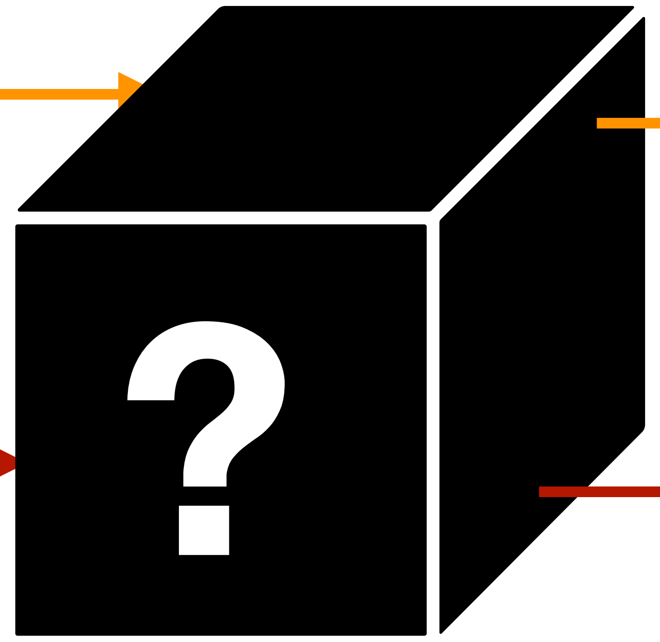


Can you [MASK] me [MASK] salt? → Can you pass me the salt?



...

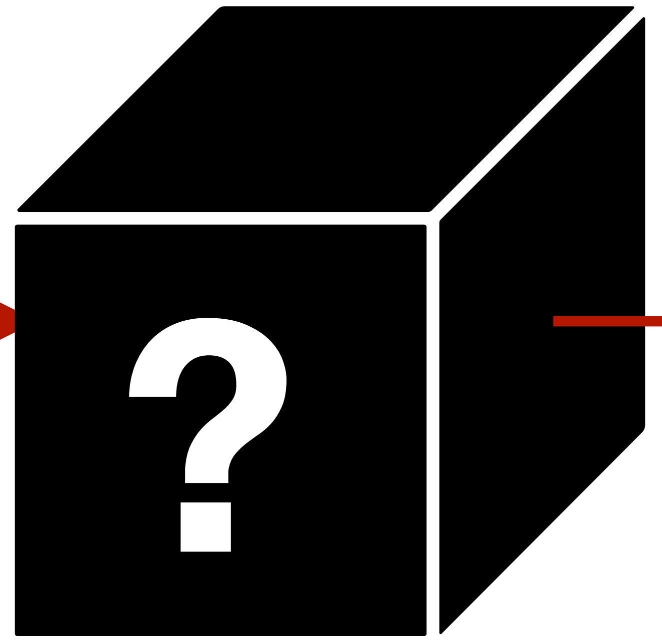
Neural Sequence Models

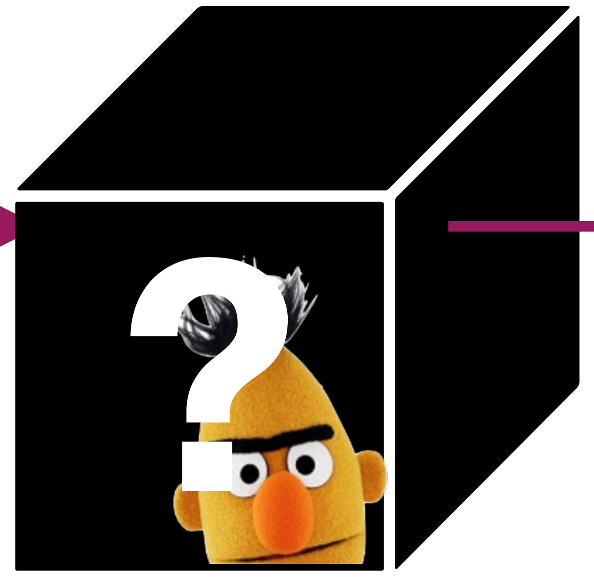


Understanding the Black Box

- Reliability (Verifiability)
- Intuition (biases, model selection)
- Knowledge extraction

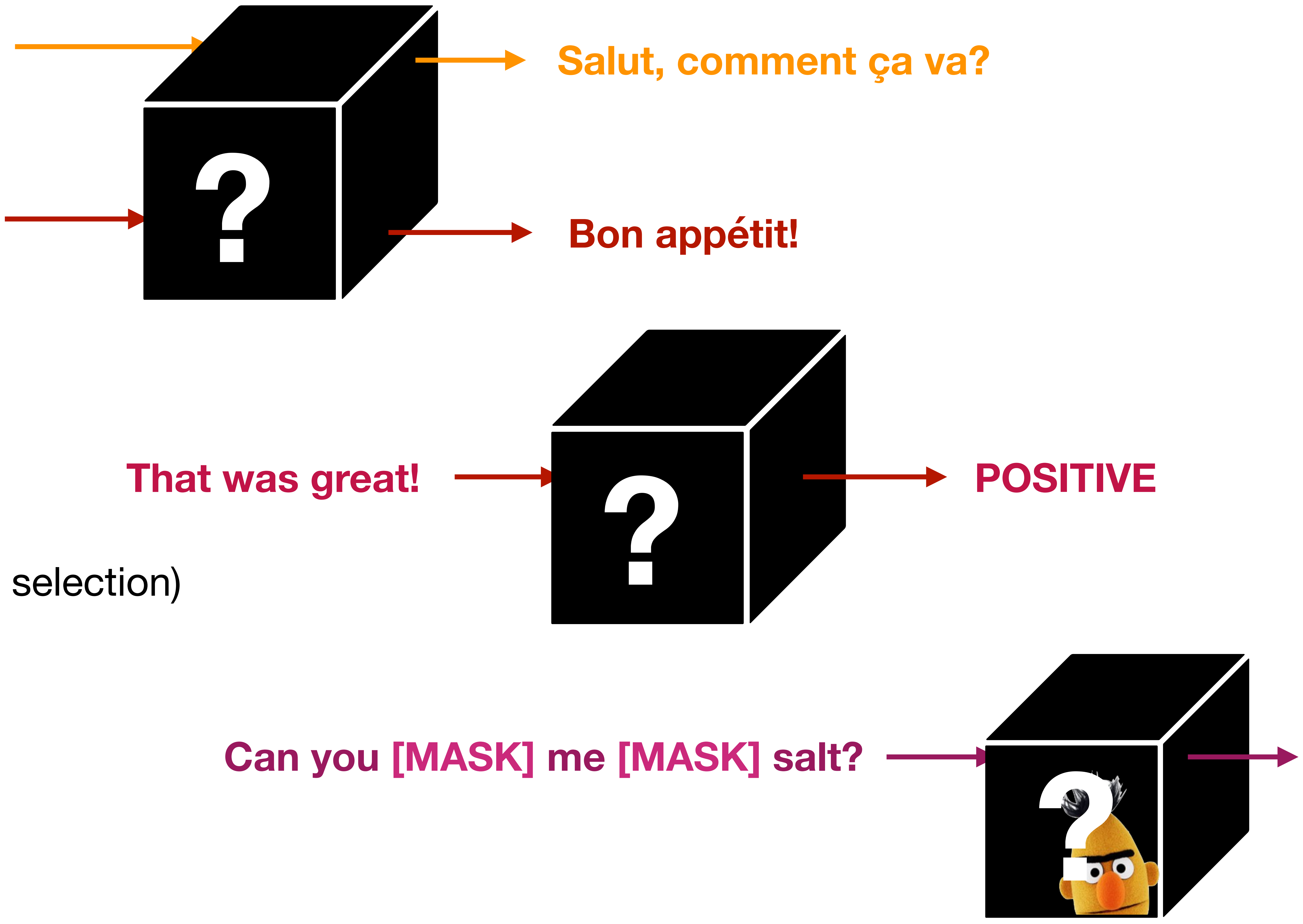
- Model design





…

Neural Sequence Models



Understanding the Black Box

- Reliability (Verifiability)
- Intuition (biases, model selection)
- Knowledge extraction

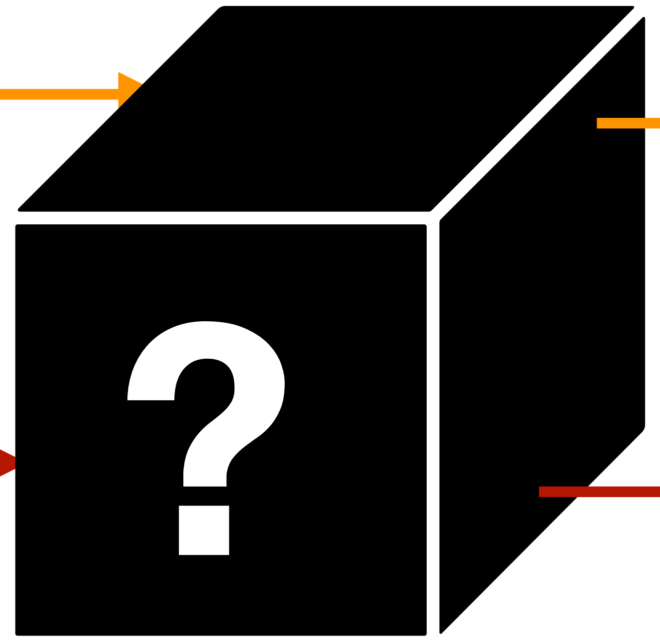
- Model design

- **Just kinda cool**

Neural Sequence Models

Hi, how are you? → Salut, comment ça va?

Enjoy your meal! → Bon appétit!



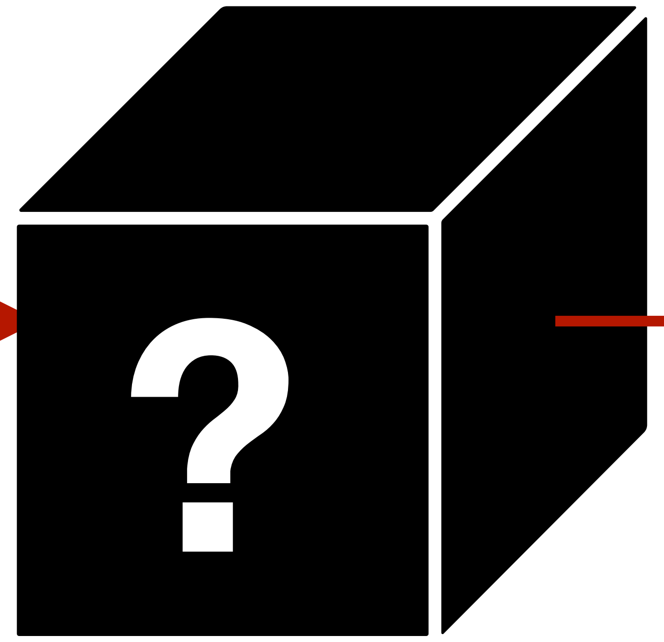
Understanding the Black Box

- Reliability (Verifiability)
- Intuition (biases, model selection)
- Knowledge extraction

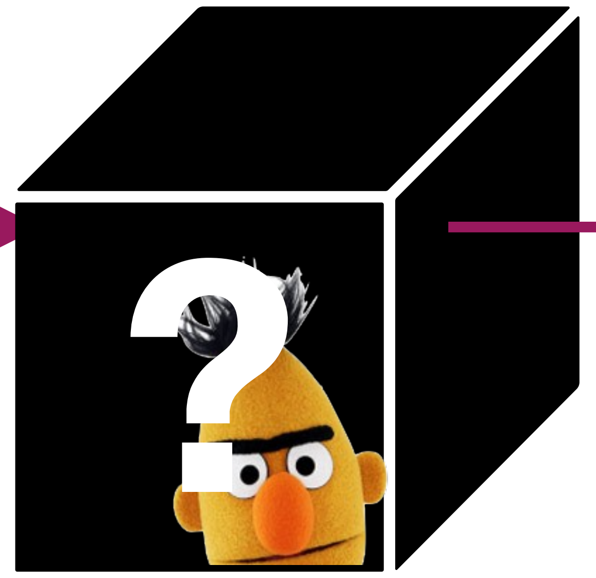
- Model design

- **Just kinda cool**

That was great! → POSITIVE



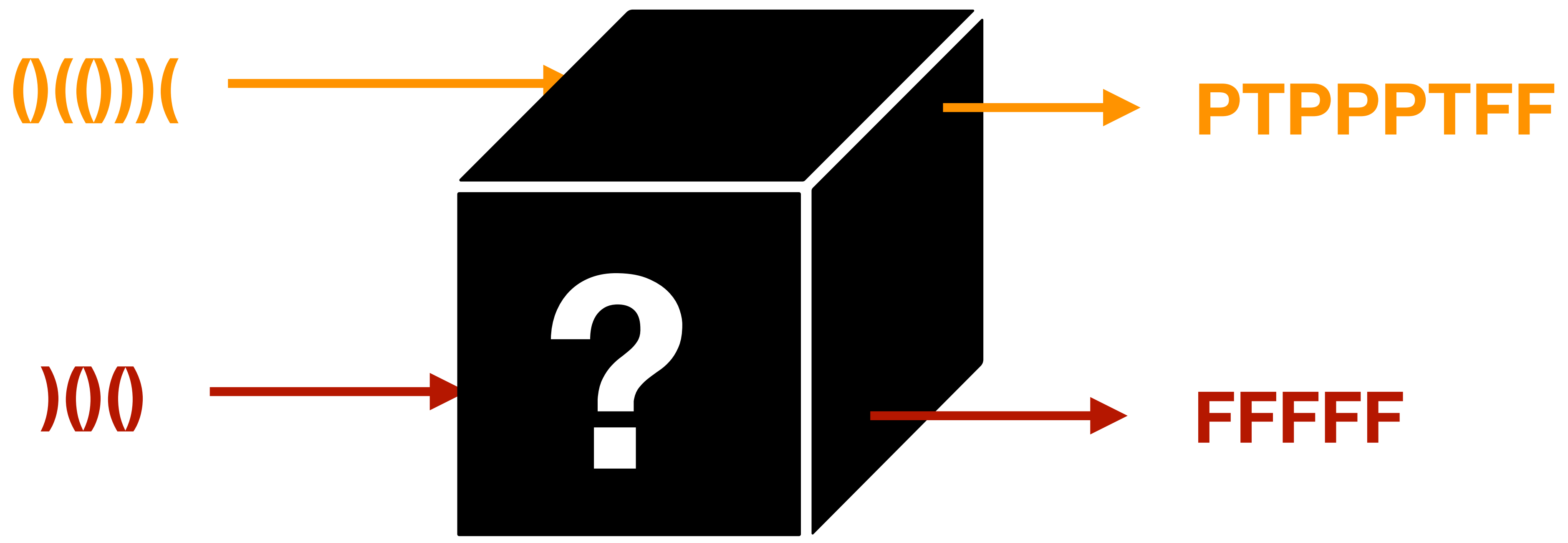
Can you [MASK] me [MASK] salt? → Can you pass me the salt?

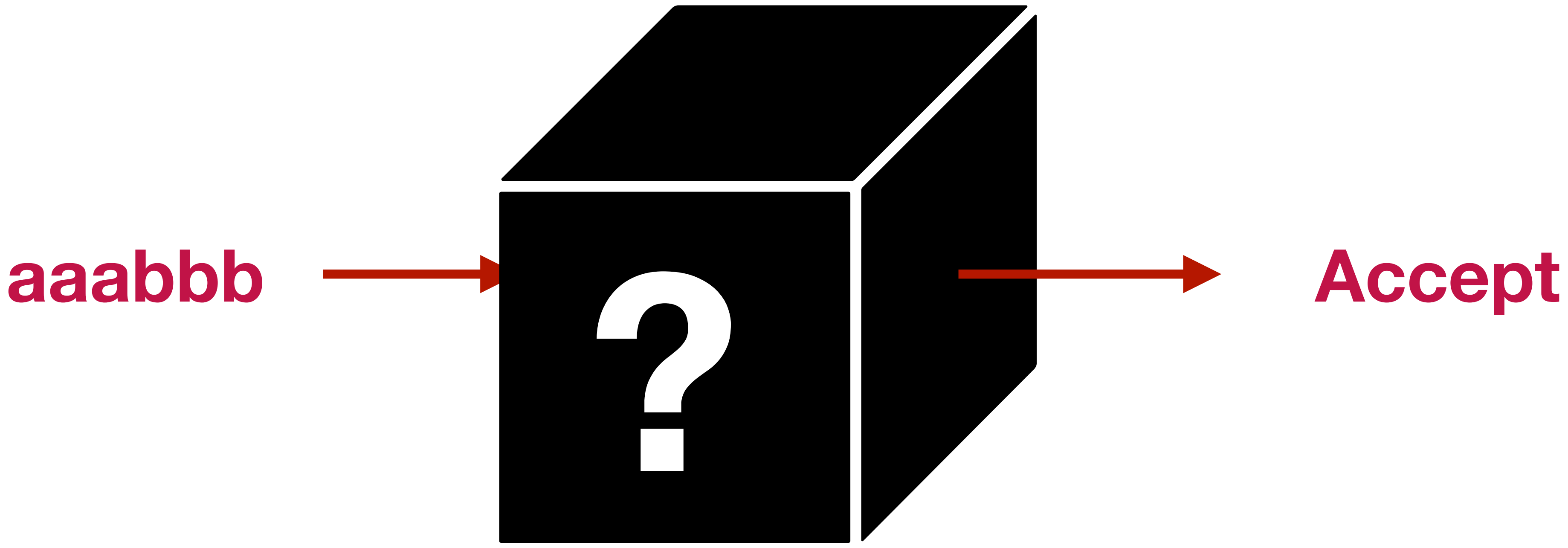


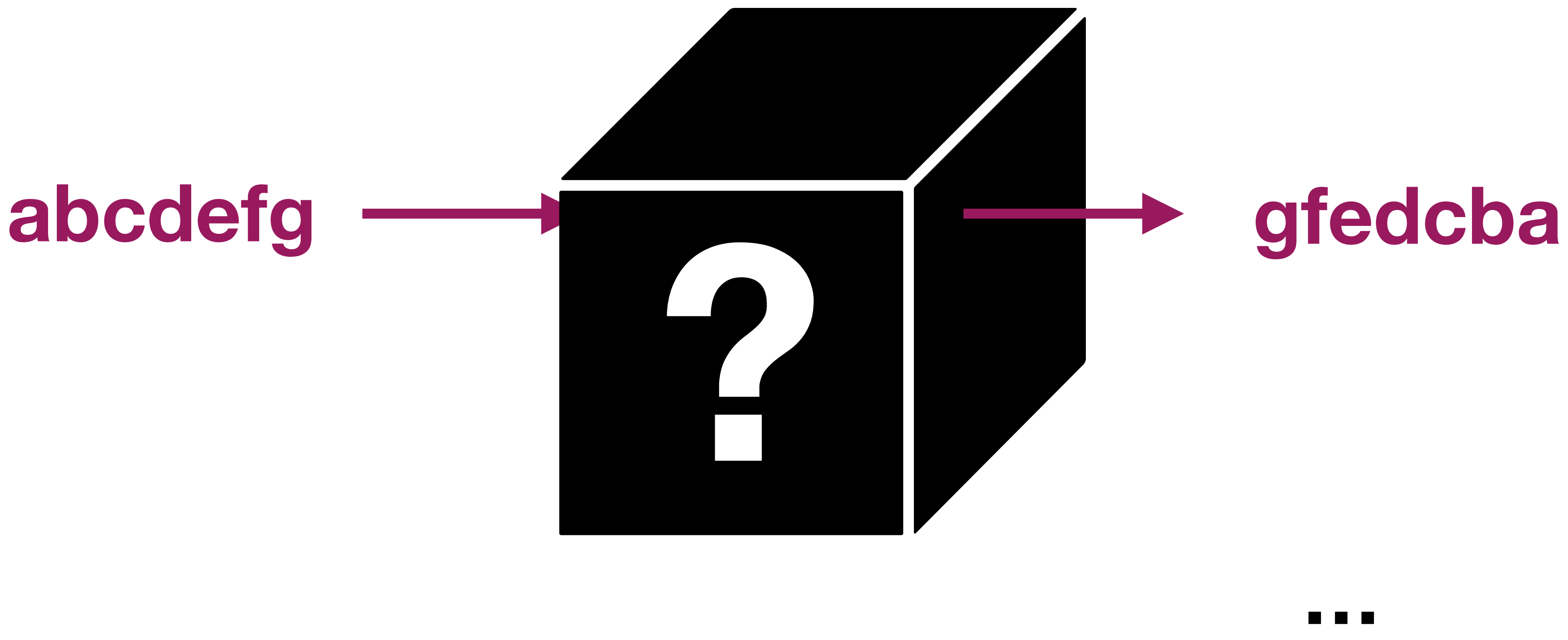
Natural language is hard…

...

Neural Sequence Models:
A Formal Lens







Understanding the Black Box

- Reliability (Verifiability)
- Intuition (biases, model selection)
- Knowledge extraction

- Model design

- **Just kinda cool**

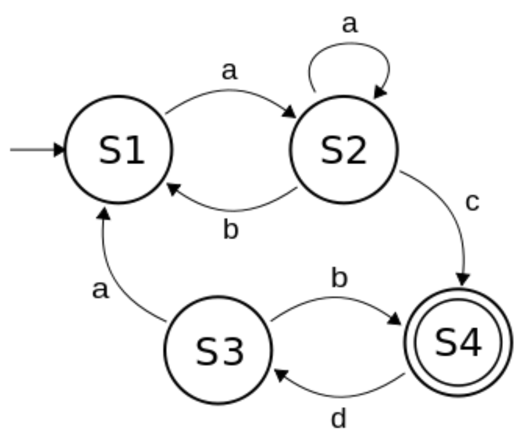
Neural Sequence Models: a Formal Lens

Counting

LSTMs are counter machines, GRUs aren't **(ACL 2018)**

RASP

Finding a formalism to describe transformers **(ICML 2021)**



DFAs from RNNs

Applying L* to learn DFAs from RNNs **(ICML 2018)**

+ using the result for CFGs **(TACAS 2021)**

Neural Sequence Models: a Formal Lens

Counting

LSTMs are counter machines, GRUs aren’t **(ACL 2018)**

RASP

Finding a formalism to describe transformers **(ICML 2021)**

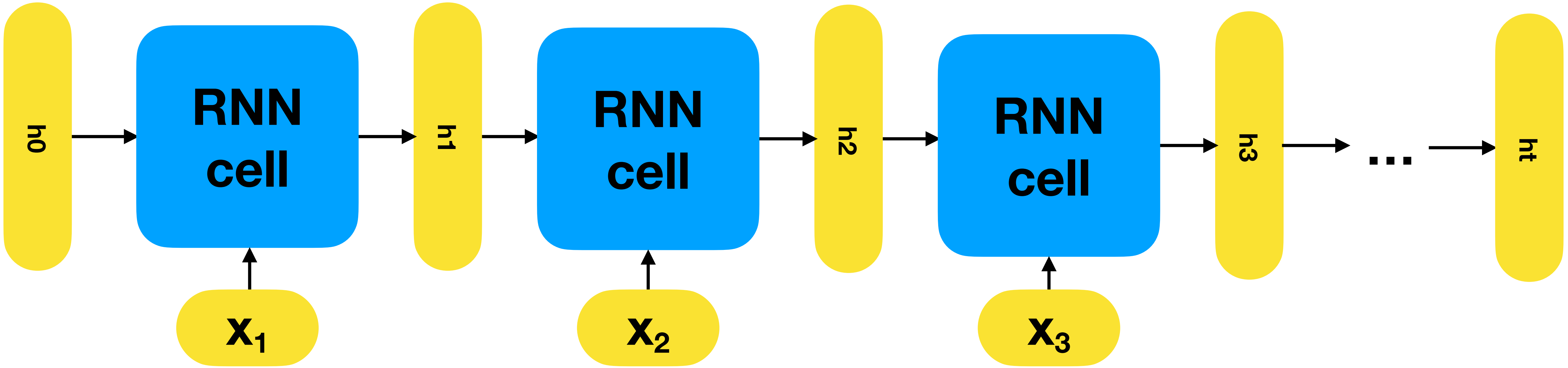
DFAs from RNNs

Applying L* to learn DFAs from RNNs **(ICML 2018)**

+ using the result for CFGs **(TACAS 2021)**

RNNs

(Elman, **1990**)
Introduction of RNNs

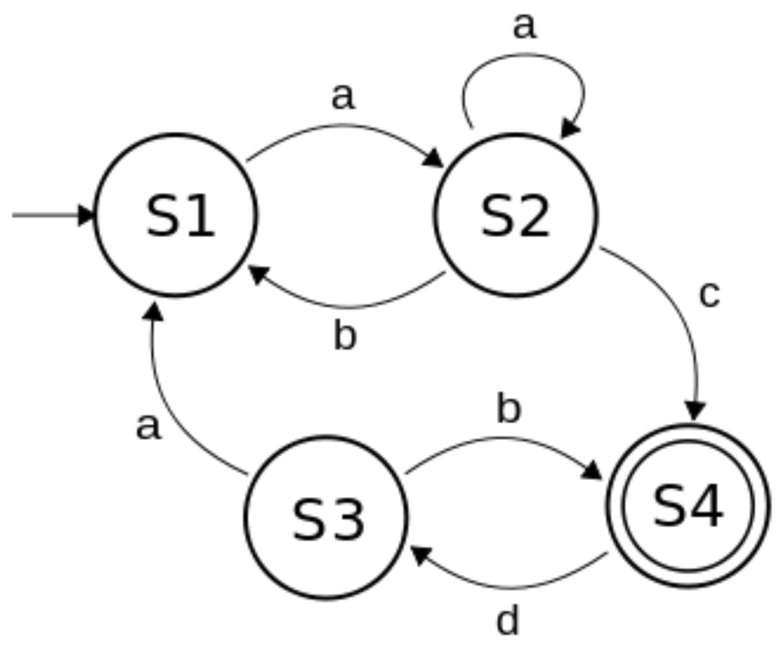


General RNN concept: $h_t = f(x_t, h_{t-1})$

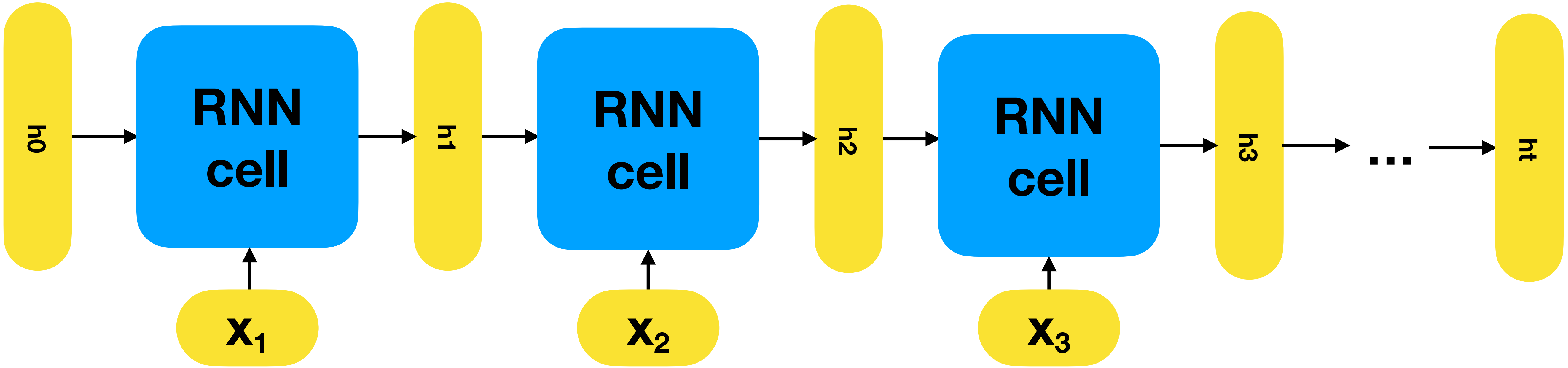
Elman RNN: $h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$

On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

RNNs



(Elman, **1990**)
Introduction of RNNs



General RNN concept: $h_t = f(x_t, h_{t-1})$

Elman RNN: $h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$

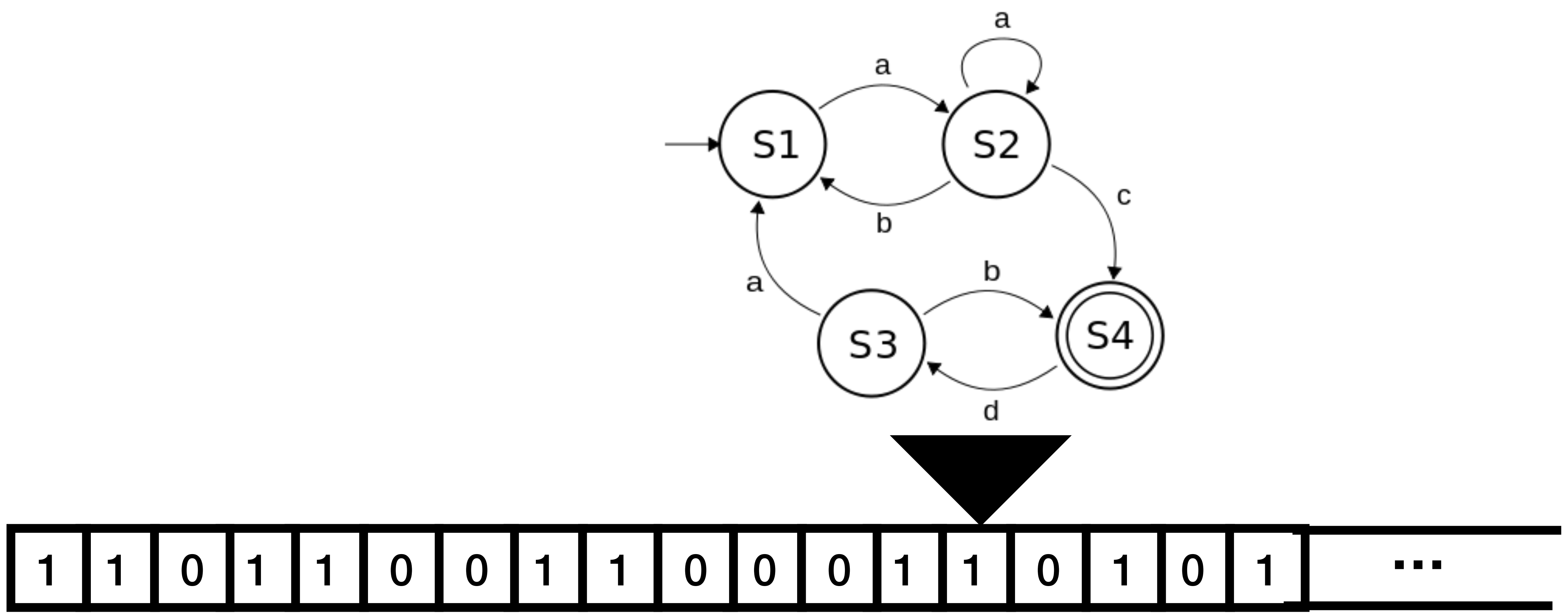
On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

RNNs

(Elman, **1990**)
Introduction of RNNs

(Siegelmann and Sonntag, **1993**)
RNNs are Turing Complete

Theoretical Power



On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

RNNs

(Elman, **1990**)
Introduction of RNNs

(Siegelmann and Sonntag, **1993**)
RNNs are Turing Complete

Theoretical Power

(Hochreiter and Schmidhuber, **1997**)
LSTMs

(Cho et al, **2014**)
GRUs

Practical Modifications

RNNs

(Elman, **1990**)
Introduction of RNNs

(Siegelmann and Sonntag, **1993**)
RNNs are Turing Complete

Theoretical Power

(Hochreiter and Schmidhuber, **1997**)
LSTMs

(Cho et al, **2014**)
GRUs

Practical Modifications

On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

RNNs

(Elman, **1990**)
Introduction of RNNs

(Siegelmann and Sonntag, **1993**)
RNNs are Turing Complete

Theoretical Power

RNN Turing Completeness Proof (1993):

1. **Requires Infinite Precision:**
   Uses stack(s), with zeros pushed using division: g = g/4 + 1/4
   *In 32 bits, this reaches the limit after **15** pushes*

2. **Requires Infinite Time:**
   And specifically, allows processing beyond reading input
   (Non standard use case!)

ldberg, Yahav, ACL 2018)

RNNs

(Elman, **1990**)
Introduction of RNNs

(Siegelmann and Sonntag, **1993**)
RNNs are Turing Complete

Theoretical Power

(Hochreiter and Schmidhuber, **1997**)
LSTMs

(Cho et al, **2014**)
GRUs

?

Practical Modifications

$$h_t = f(x_t, h_{t-1})$$

Practical RNNs

GRU

$$z_t = \sigma(W^z x_t + U^z h_{t-1} + b^z)$$
$$r_t = \sigma(W^r x_t + U^r h_{t-1} + b^r)$$
$$\tilde{h}_t = \tanh(W^h x_t + U^h (r_t \circ h_{t-1}) + b^h)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

LSTM

$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b^f)$$
$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b^i)$$
$$o_t = \sigma(W^o x_t + U^o h_{t-1} + b^o)$$
$$\tilde{c}_t = \tanh(W^c x_t + U^c h_{t-1} + b^c)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ g(c_t)$$

On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

Practical RNNs

GRU

$$z_t = \sigma(W^z x_t + U^z h_{t-1} + b^z)$$
$$r_t = \sigma(W^r x_t + U^r h_{t-1} + b^r)$$
$$\tilde{h}_t = \tanh(W^h x_t + U^h(r_t \circ h_{t-1}) + b^h)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

LSTM

$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b^f)$$
$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b^i)$$
$$o_t = \sigma(W^o x_t + U^o h_{t-1} + b^o)$$
$$\tilde{c}_t = \tanh(W^c x_t + U^c h_{t-1} + b^c)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ g(c_t)$$

gates

candidate vectors

update functions

On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

Practical RNNs

GRU

$$z_t \in (0,1)$$
$$r_t \in (0,1)$$
$$\tilde{h}_t = \tanh(W^h x_t + U^h(r_t \circ h_{t-1}) + b^h)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

LSTM

$$f_t \in (0,1)$$
$$i_t \in (0,1)$$
$$o_t \in (0,1)$$
$$\tilde{c}_t = \tanh(W^c x_t + U^c h_{t-1} + b^c)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ g(c_t)$$

gates

candidate vectors

update functions

On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

Practical RNNs

gates

candidate vectors

update functions

GRU

$$z_t \in (0,1)$$
$$r_t \in (0,1)$$
$$\tilde{h}_t \in (-1,1)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

LSTM

$$f_t \in (0,1)$$
$$i_t \in (0,1)$$
$$o_t \in (0,1)$$
$$\tilde{c}_t \in (-1,1)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ g(c_t)$$

On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

Practical RNNs

GRU

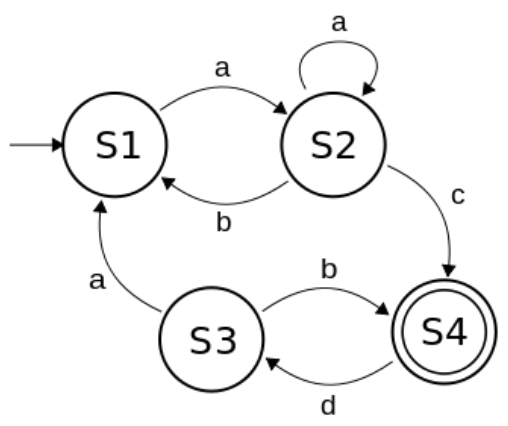
$z_t \in (0,1)$

$r_t \in (0,1)$

Bounded!

$\tilde{h}_t \in (-1,1)$

$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$



Interpolation

LSTM

$f_t \in (0,1)$

$i_t \in (0,1)$

$o_t \in (0,1)$

$\tilde{c}_t \in (-1,1)$

$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$

$h_t = o_t \circ g(c_t)$

On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

Practical RNNs

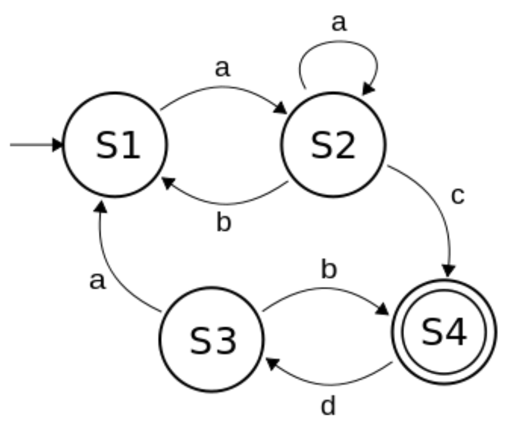
GRU

$z_t \in (0,1)$

$r_t \in (0,1)$

$\tilde{h}_t \in (-1,1)$

$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$



LSTM

$f_t \in (0,1)$ **reset/keep, then -**

$i_t \in (0,1)$ **stay/step, by -**

$o_t \in (0,1)$

$\tilde{c}_t \in (-1,1)$ **subtract/add**

$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$

$h_t = o_t \circ g(c_t)$

On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

Practical RNNs

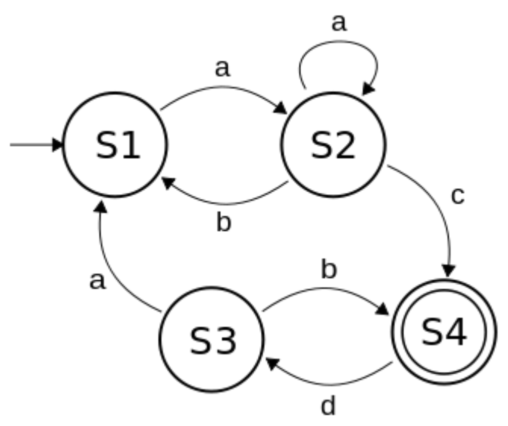
GRU

$z_t \in (0,1)$

$r_t \in (0,1)$

$\tilde{h}_t \in (-1,1)$

$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$



LSTM

$f_t \in (0,1)$ **reset/keep, then -**

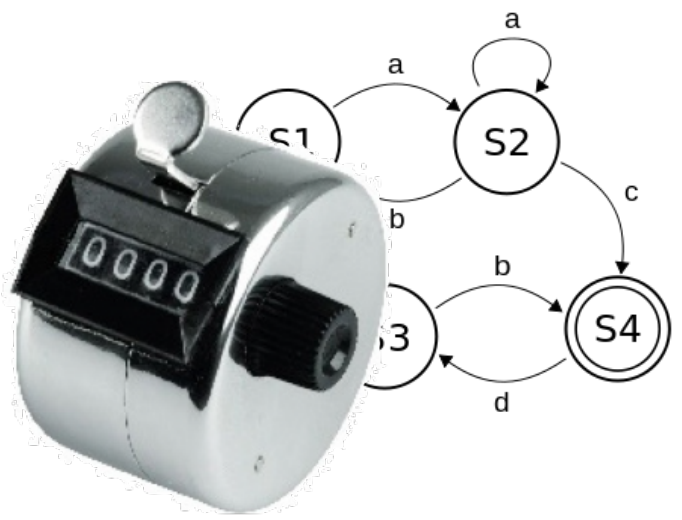
$i_t \in (0,1)$ **stay/step, by -**

$o_t \in (0,1)$

$\tilde{c}_t \in (-1,1)$ **subtract/add**

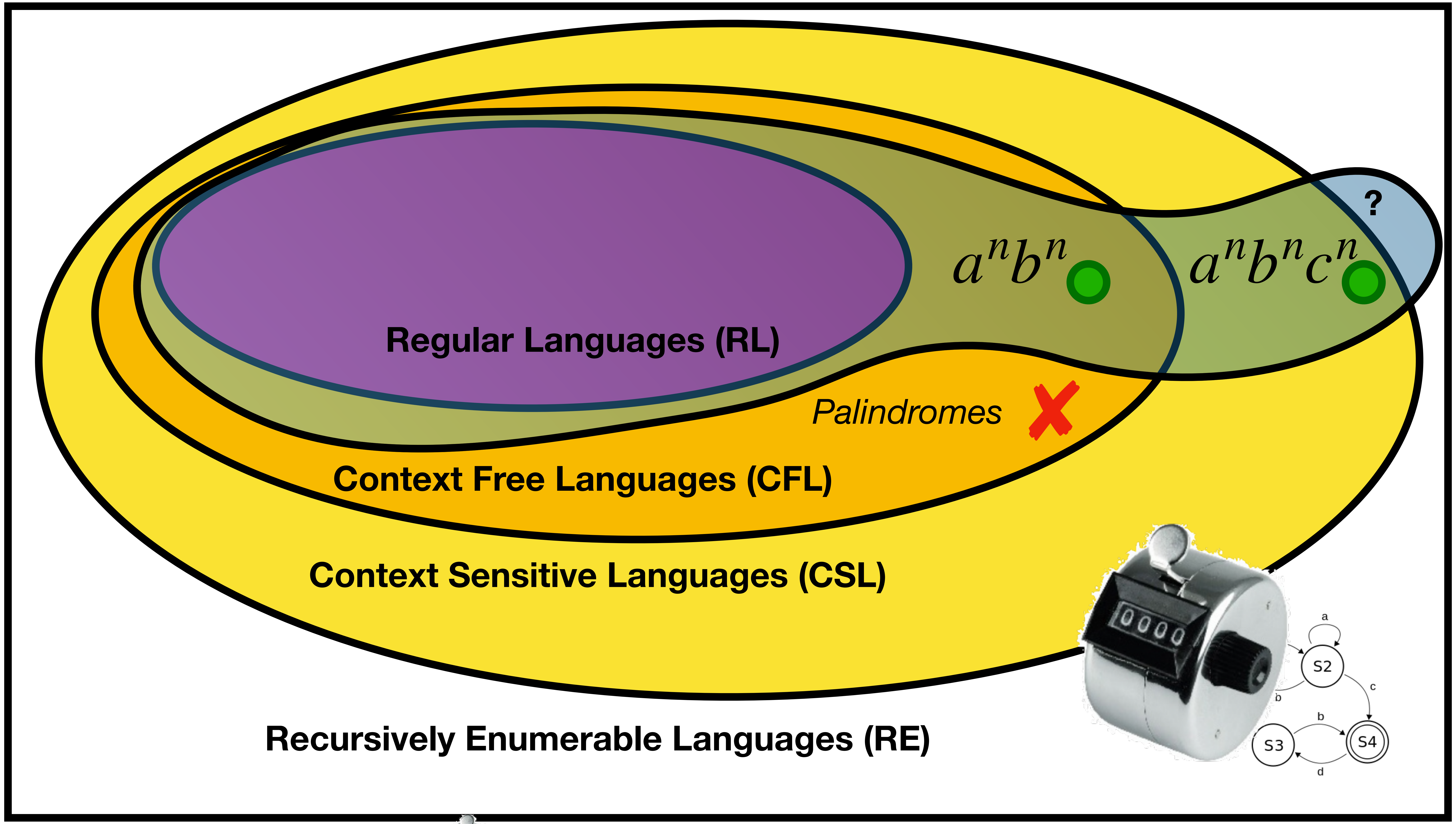
$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$

$h_t = o_t \circ g(c_t)$



On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

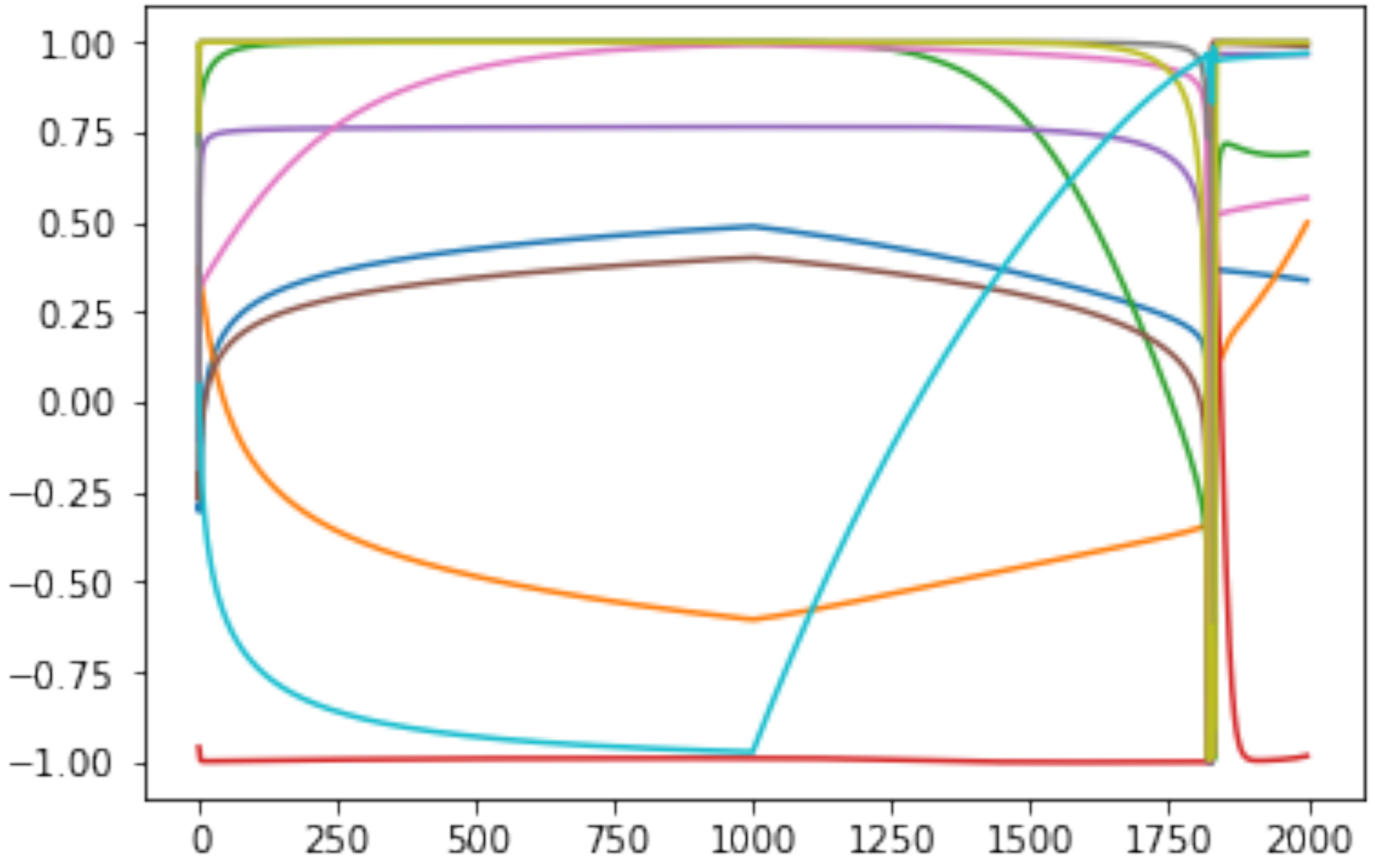
Counting



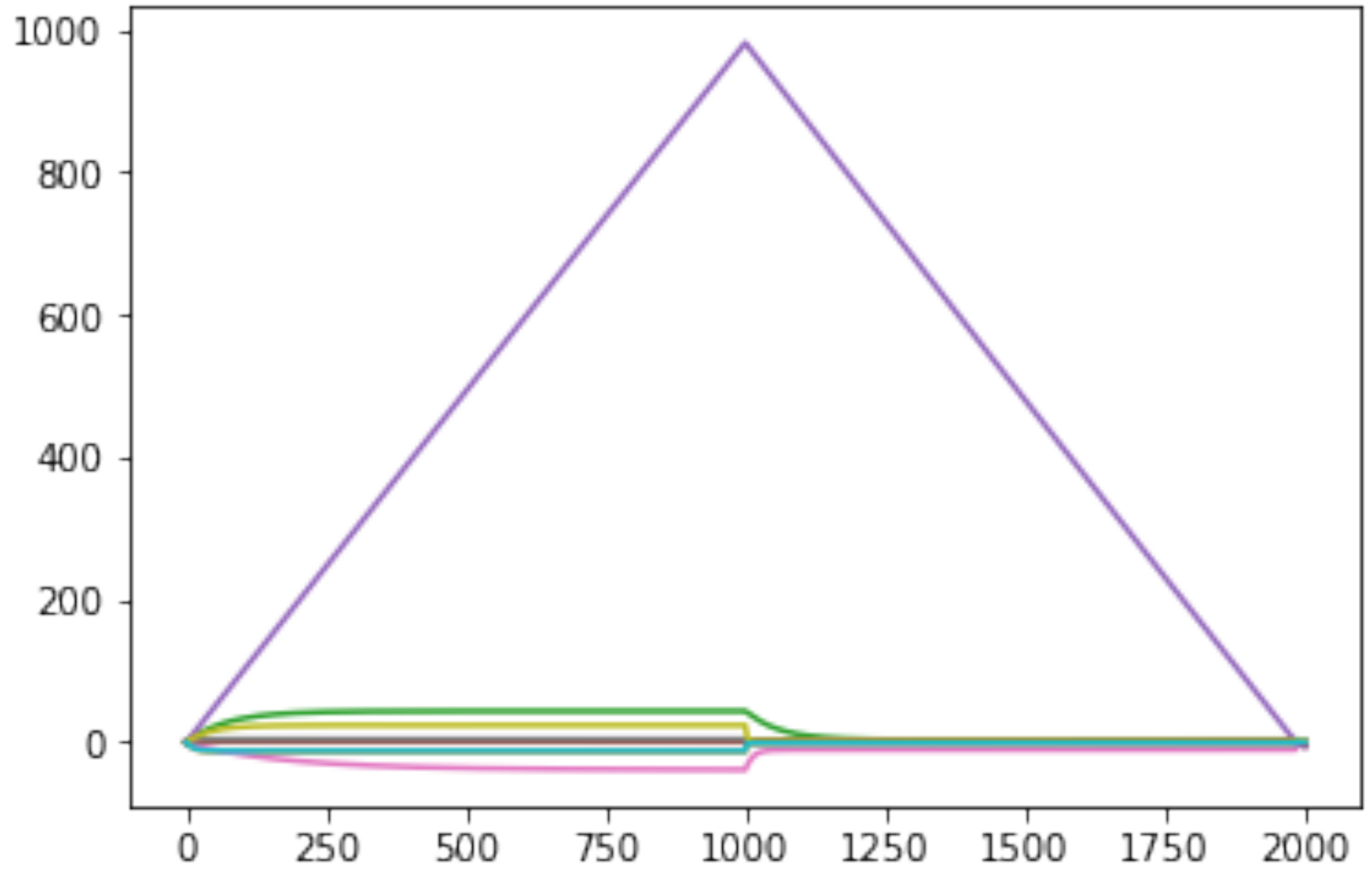
On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

Practical RNNs

GRU



LSTM



Activations on $a^{1000}b^{1000}$

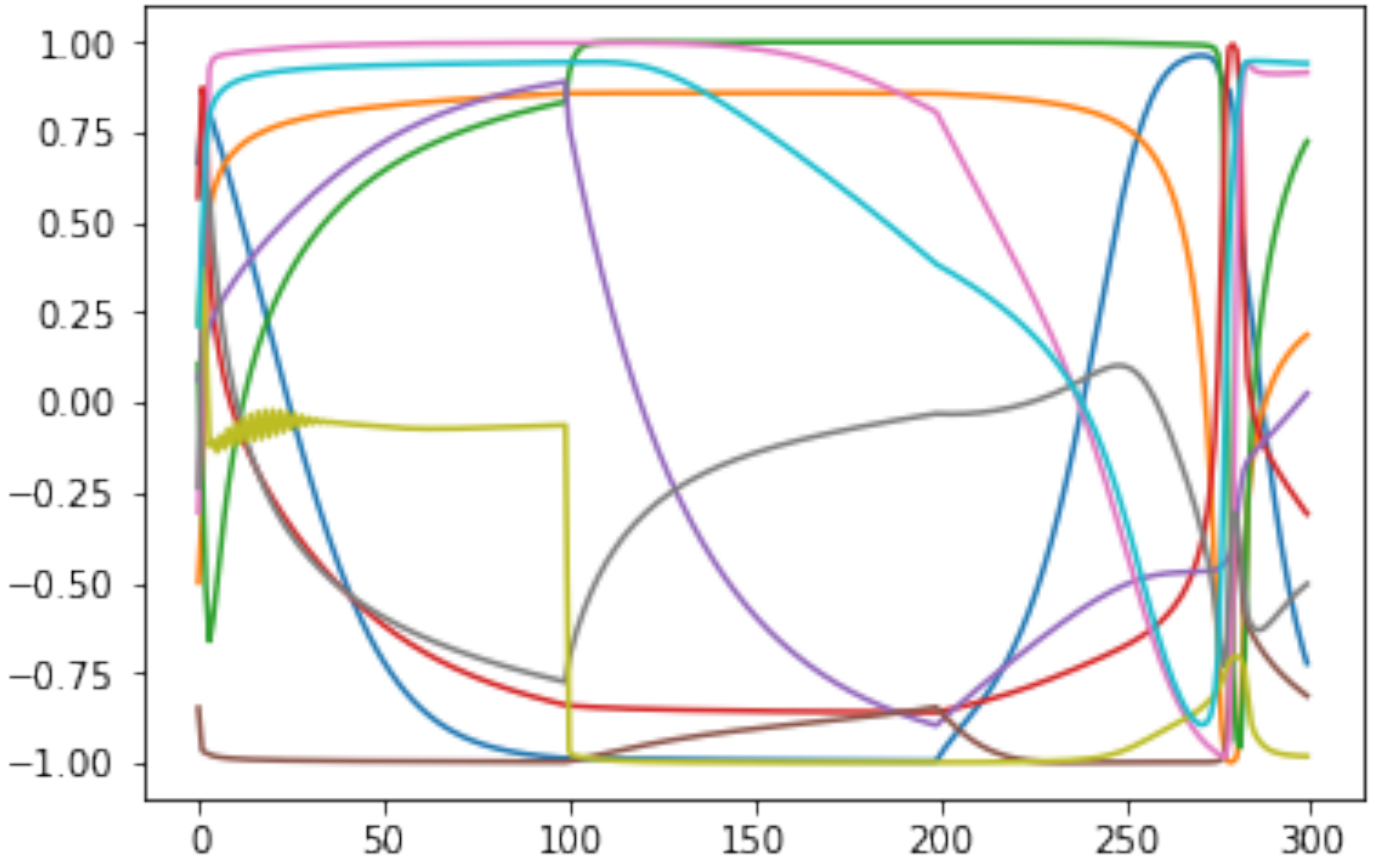
Trained $a^n b^n$ **,** (on positive examples up to length 100)

GRU begins failing at length 39

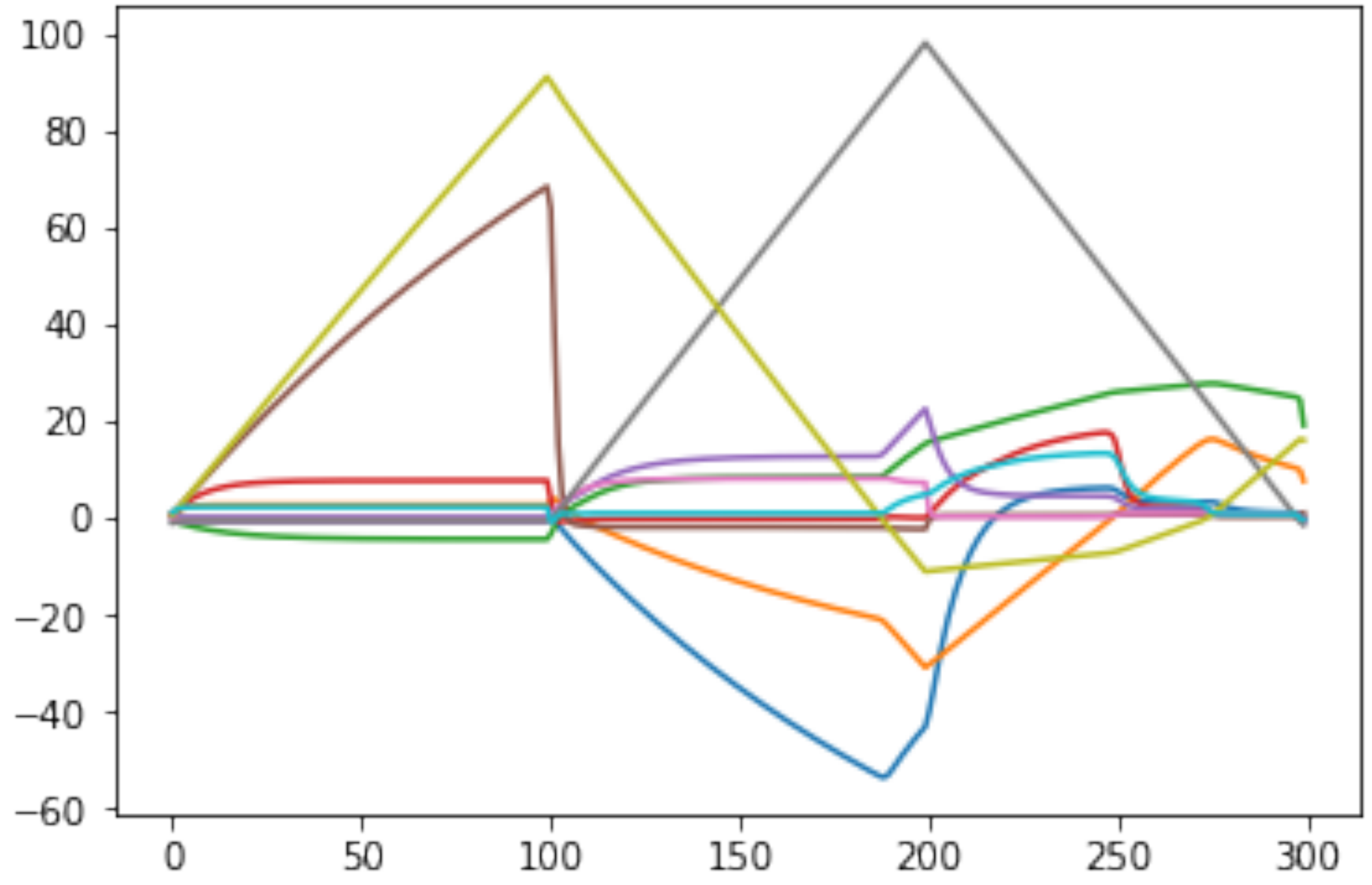
On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

Practical RNNs

GRU



LSTM



Activations on $a^{100}b^{100}c^{100}$

Trained $a^n b^n c^n$ **,** (on positive examples up to length 100)

GRU begins failing at length 9

On the Practical Computational Power of Finite Precision RNNs for Language Recognition (Weiss, Goldberg, Yahav, ACL 2018)

Neural Sequence Models:
a Formal Lens

Counting
LSTMs are counter machines, GRUs aren’t **(ACL 2018)**

RASP
Finding a formalism to describe transformers **(ICML 2021)**

DFAs from RNNs
Applying L* to learn DFAs from RNNs **(ICML 2018)**

+ using the result for CFGs **(TACAS 2021)**

Neural Sequence Models: a Formal Lens

Counting

LSTMs are counter machines, GRUs aren’t **(ACL 2018)**

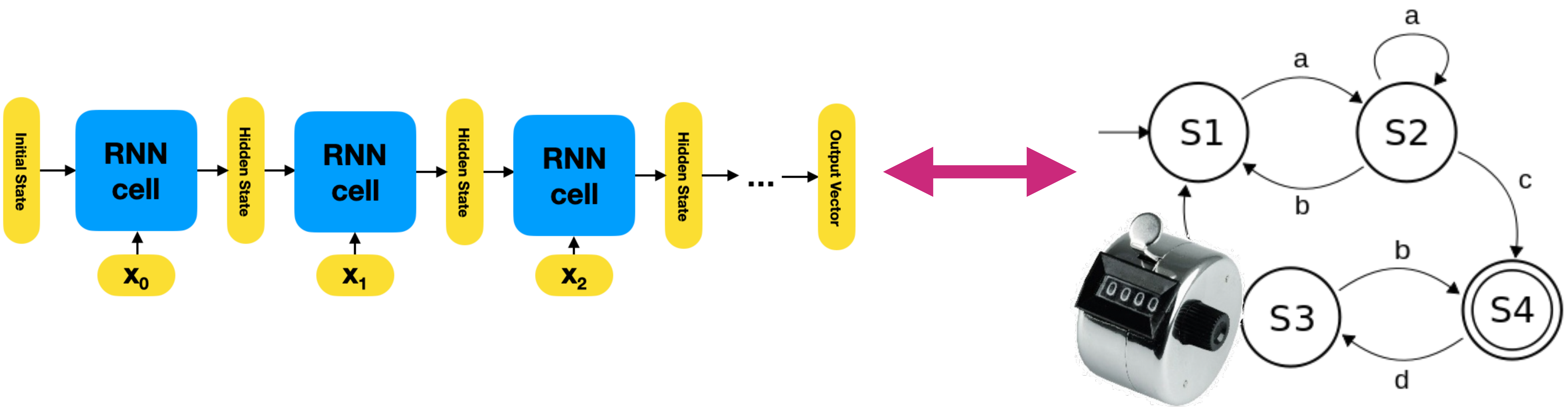
RASP

Finding a formalism to describe transformers **(ICML 2021)**

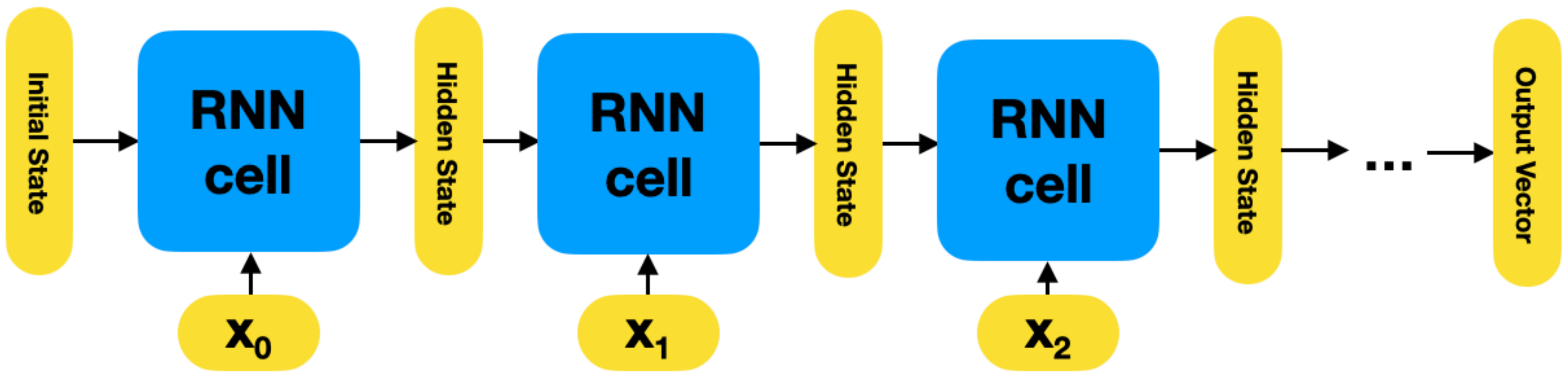
DFAs from RNNs

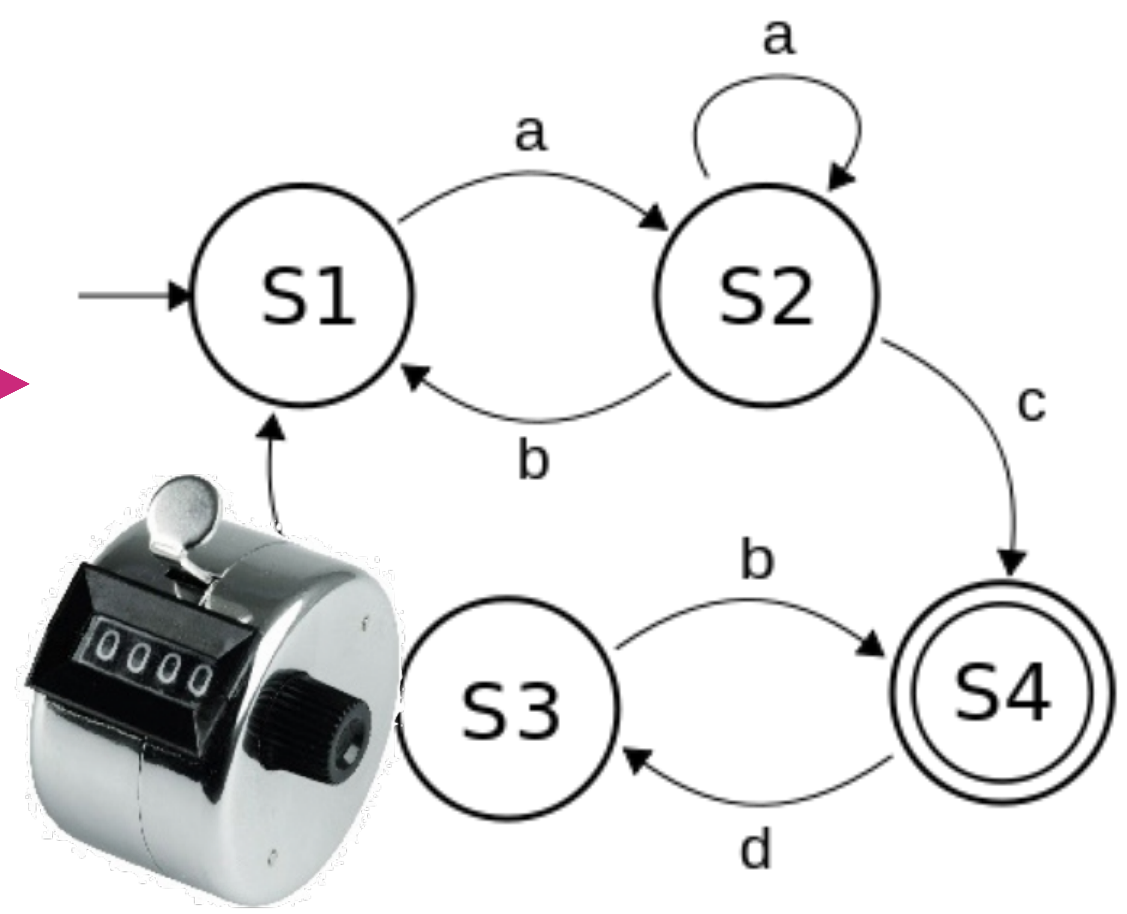
Applying L* to learn DFAs from RNNs **(ICML 2018)**

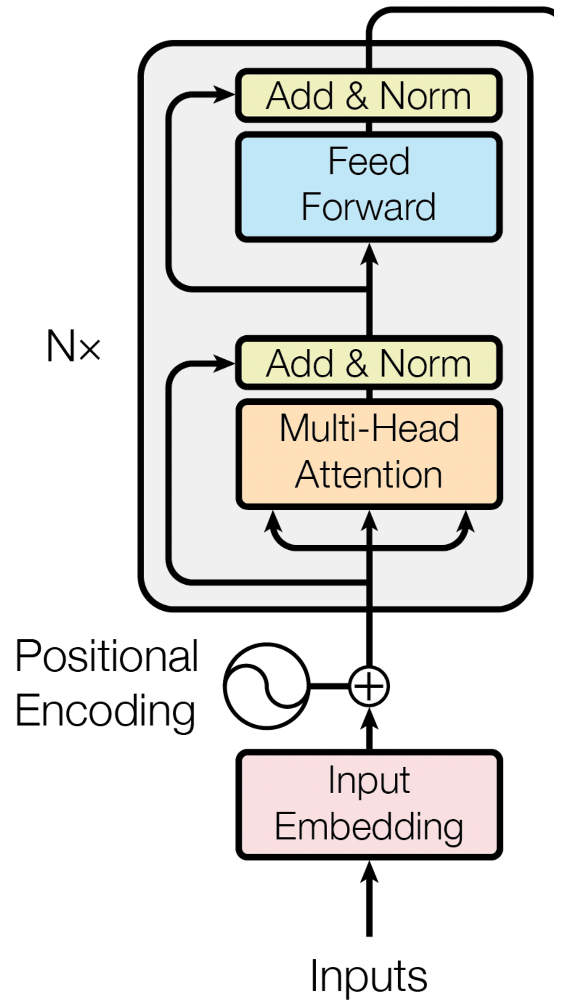
+ using the result for CFGs **(TACAS 2021)**


Initial State
RNN cell
Hidden State
RNN cell
Hidden State
RNN cell
Hidden State
...
Output Vector
x_0
x_1
x_2
S1
S2
S3
S4
a
a
b
c
b
d

RASP



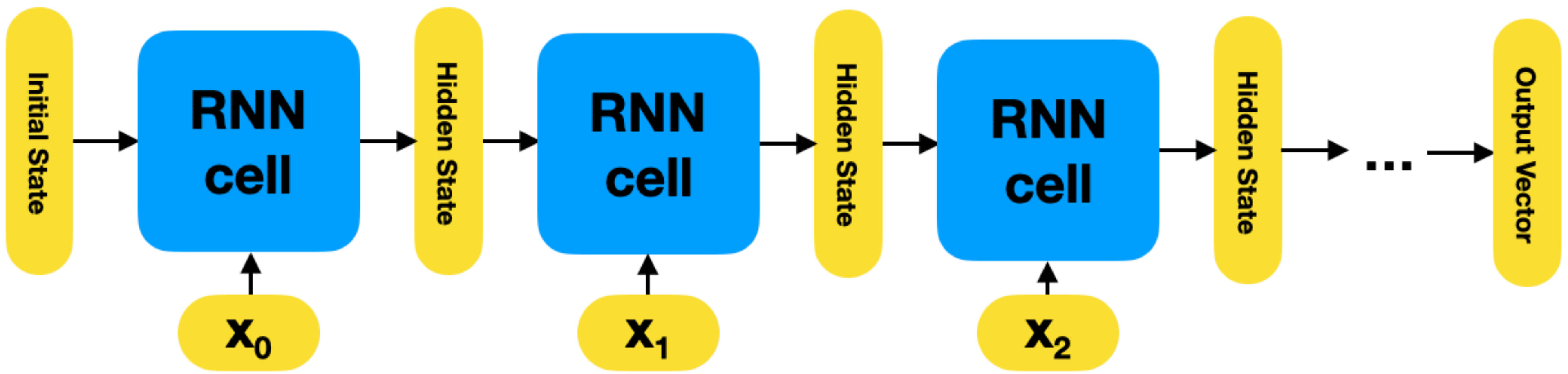


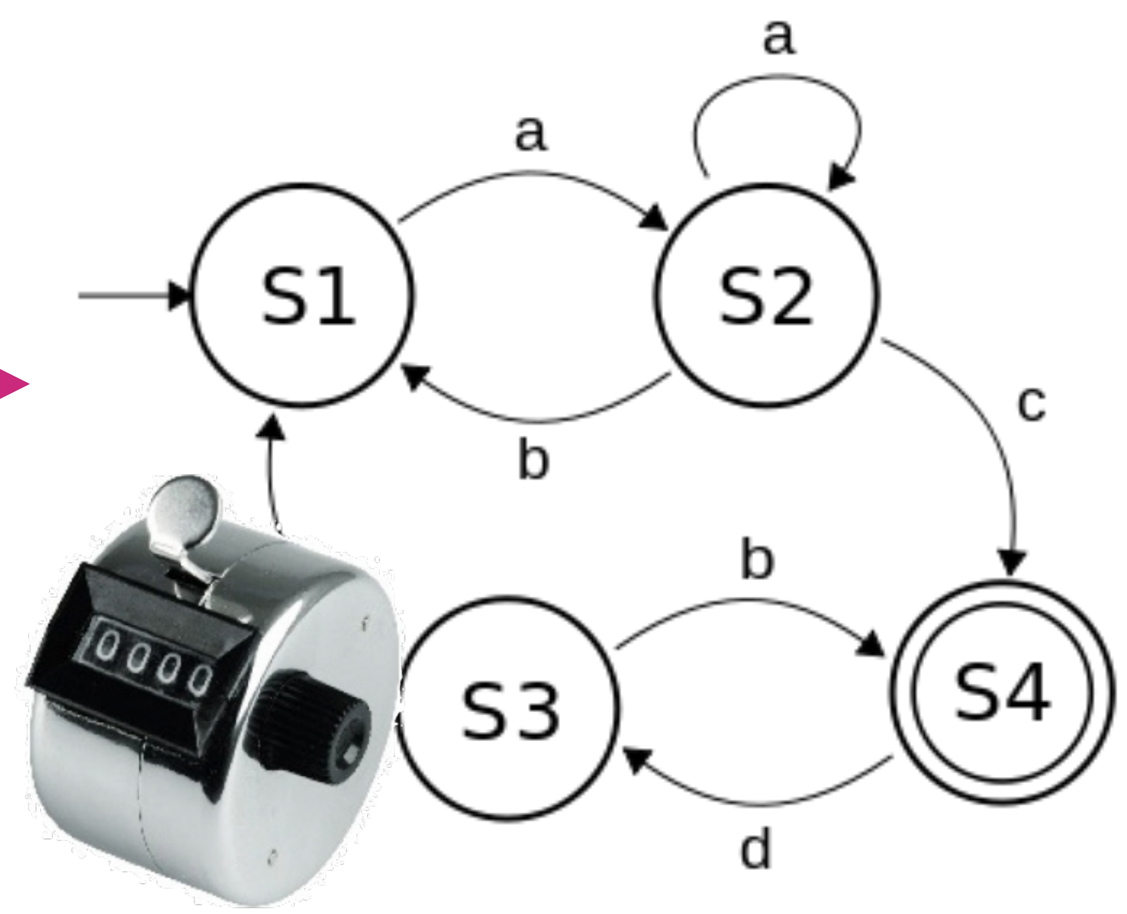


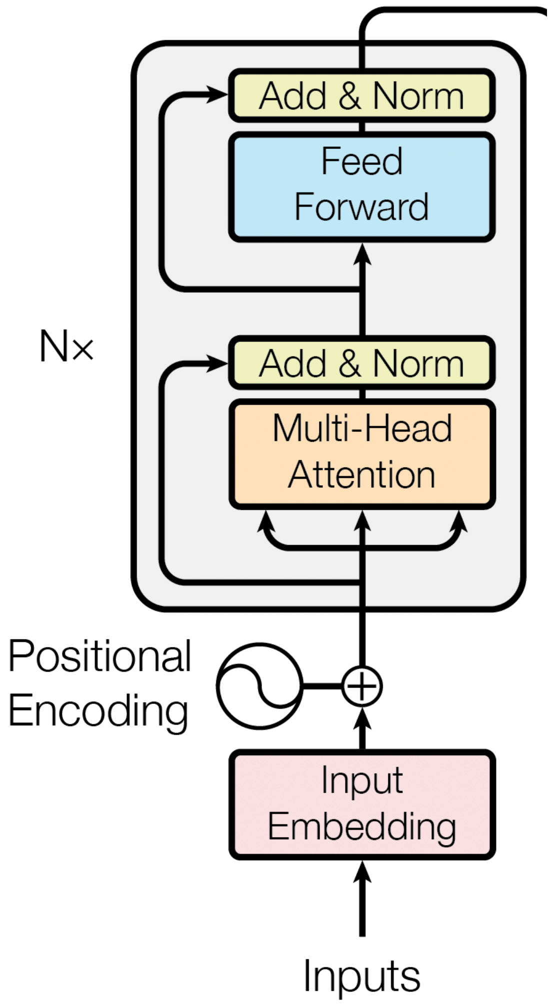
?

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

RASP







?

How is the transformer… doing things?

(How) does it count?

(How) does it reason?

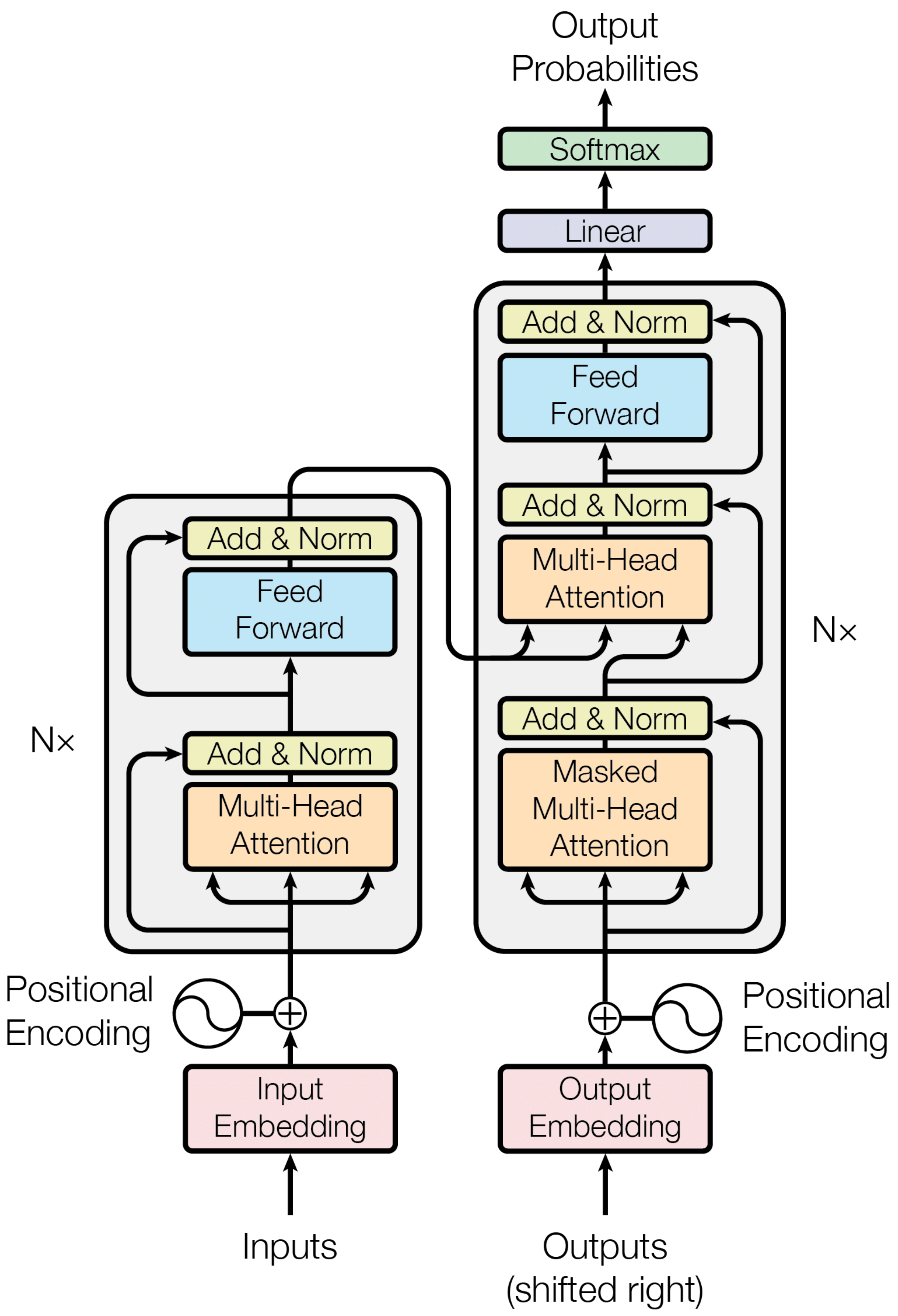
(What) does attention explain?

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Transformers

Attention Is All You Need

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

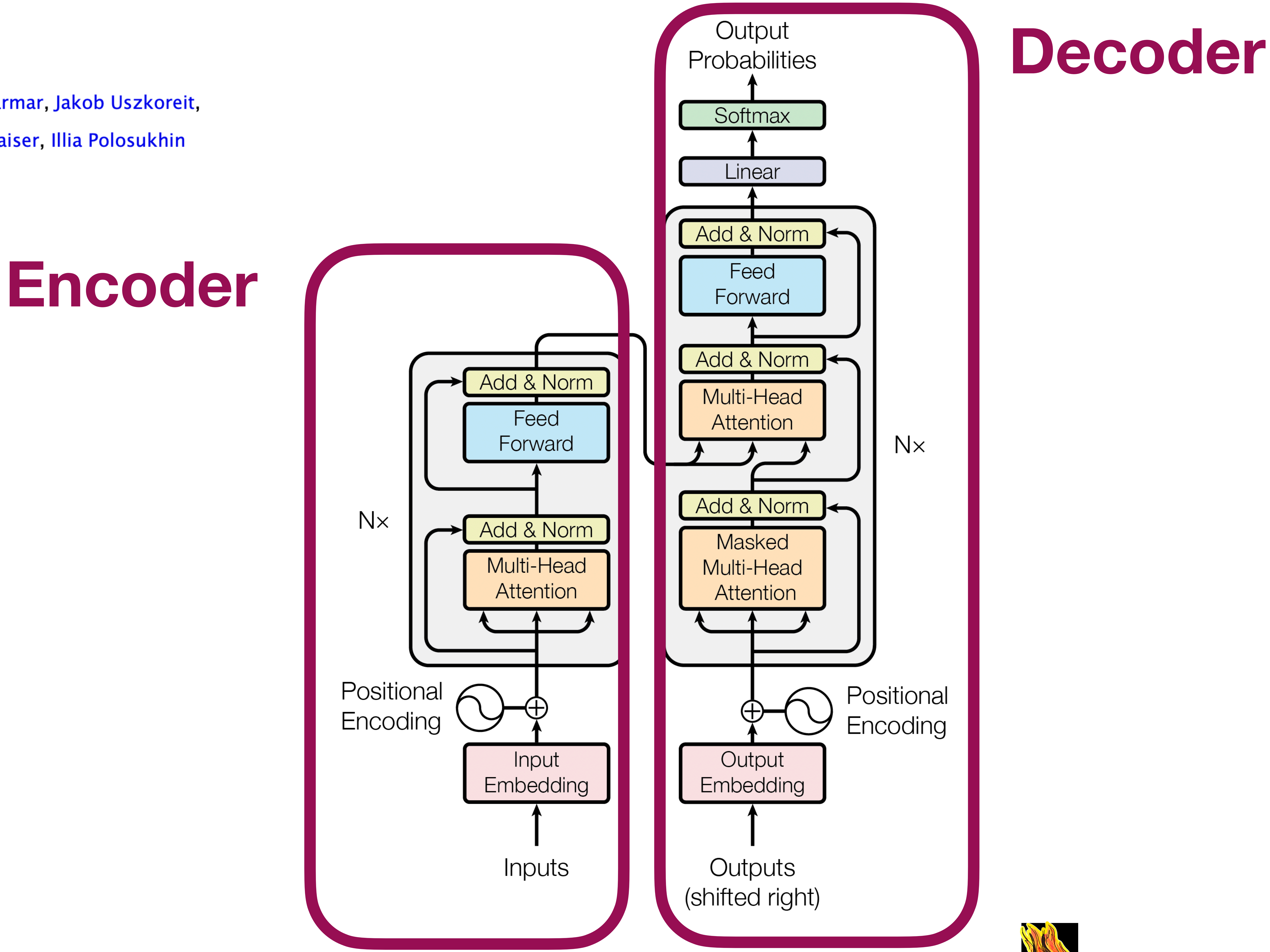


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Transformers

Attention Is All You Need

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

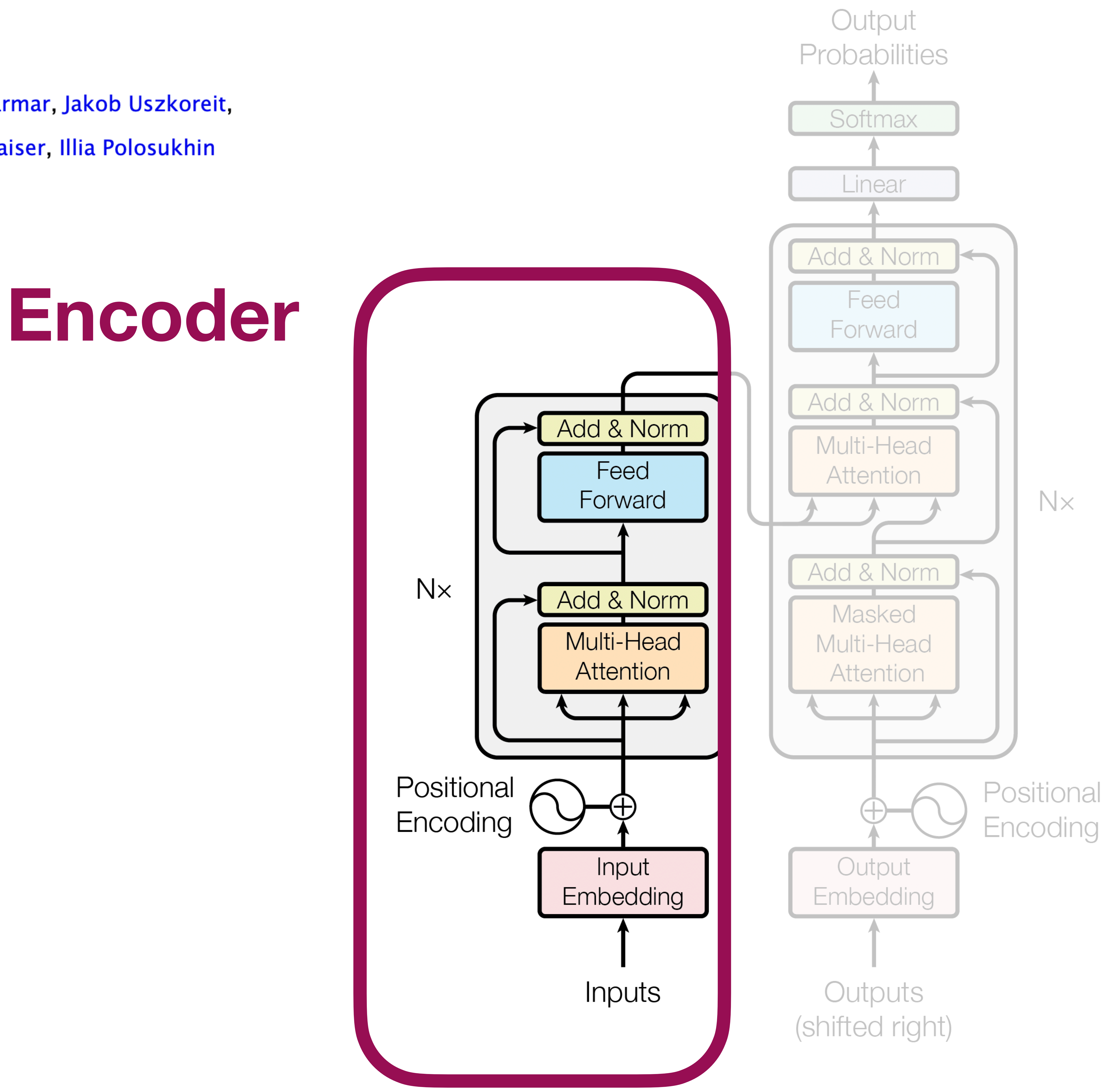


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Transformers

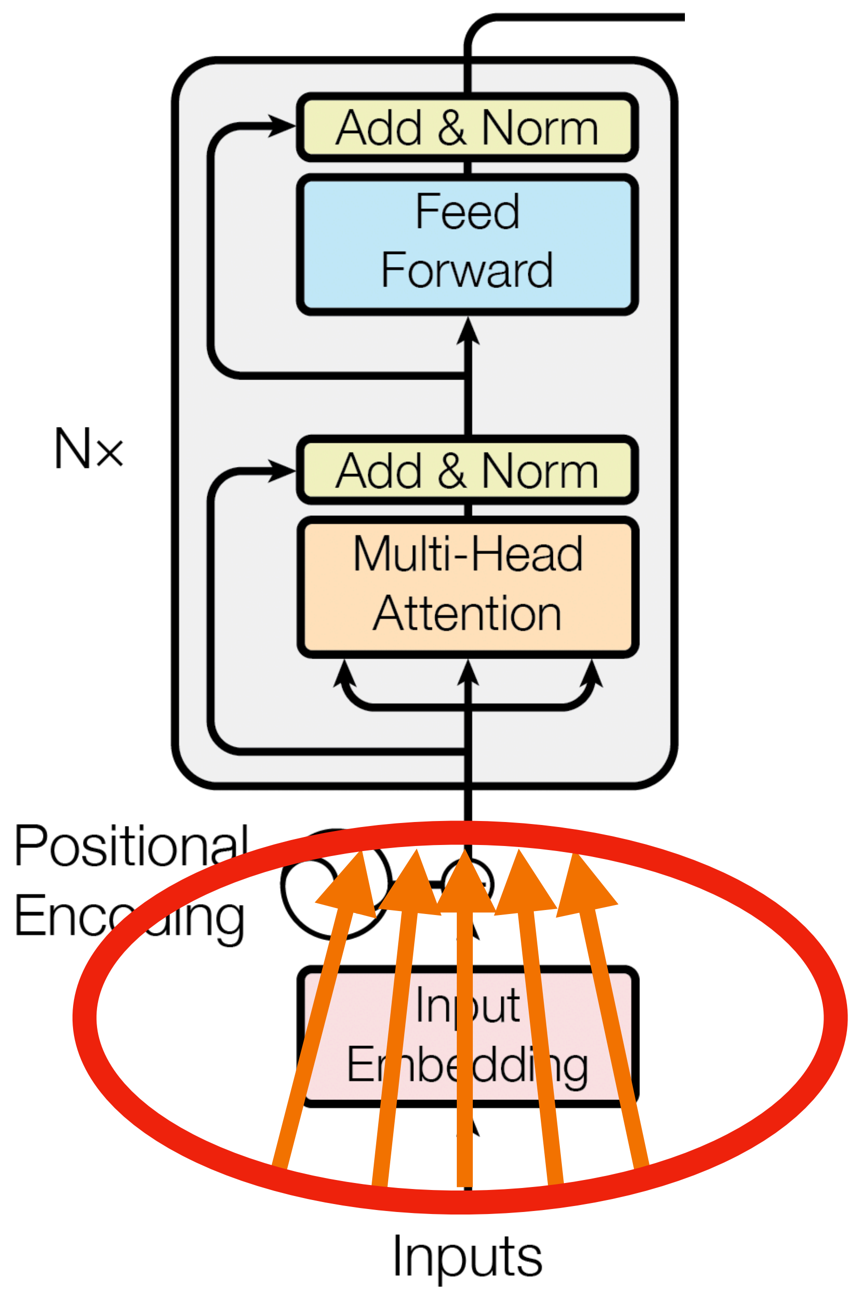
Attention Is All You Need

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



Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

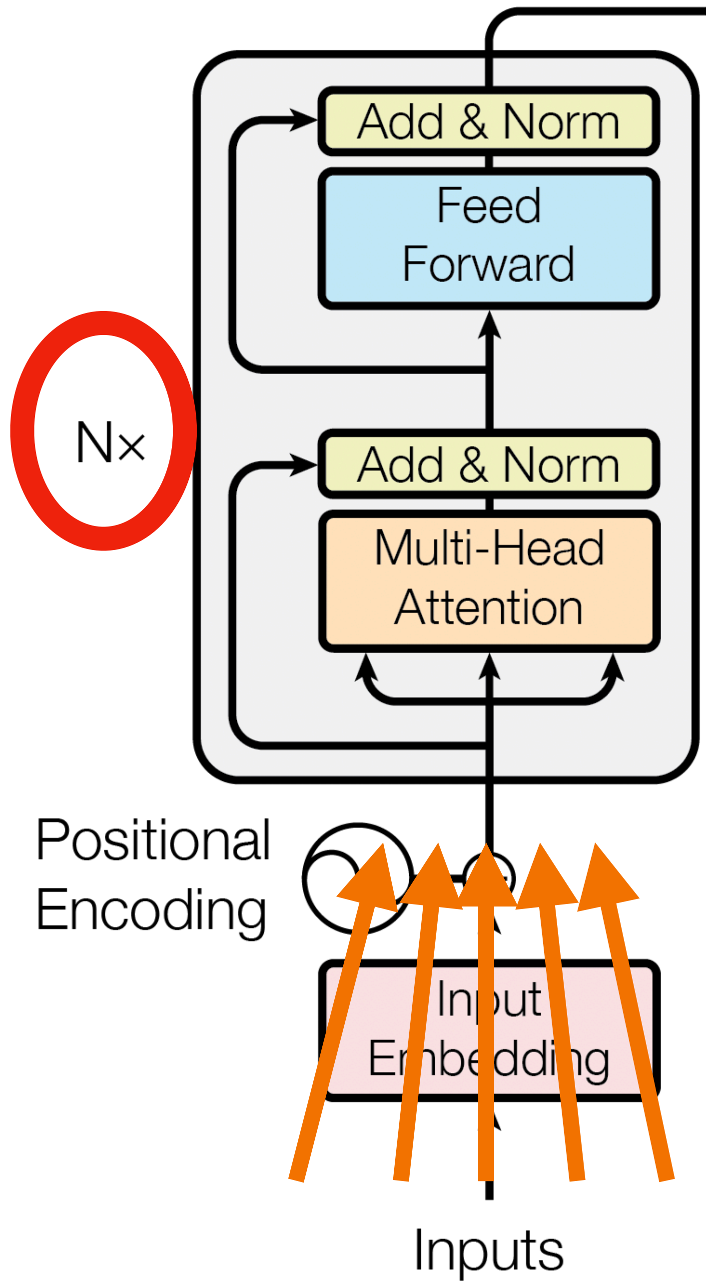
Transformers



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

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

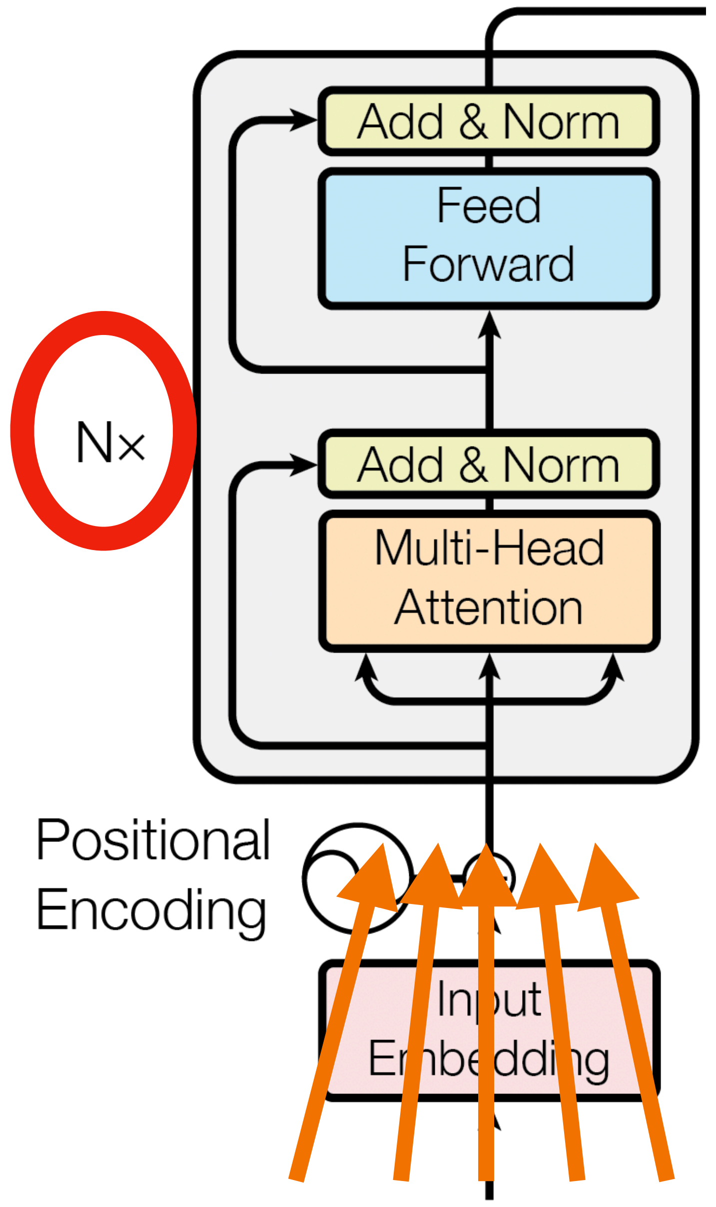
Transformers



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

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

Transformers

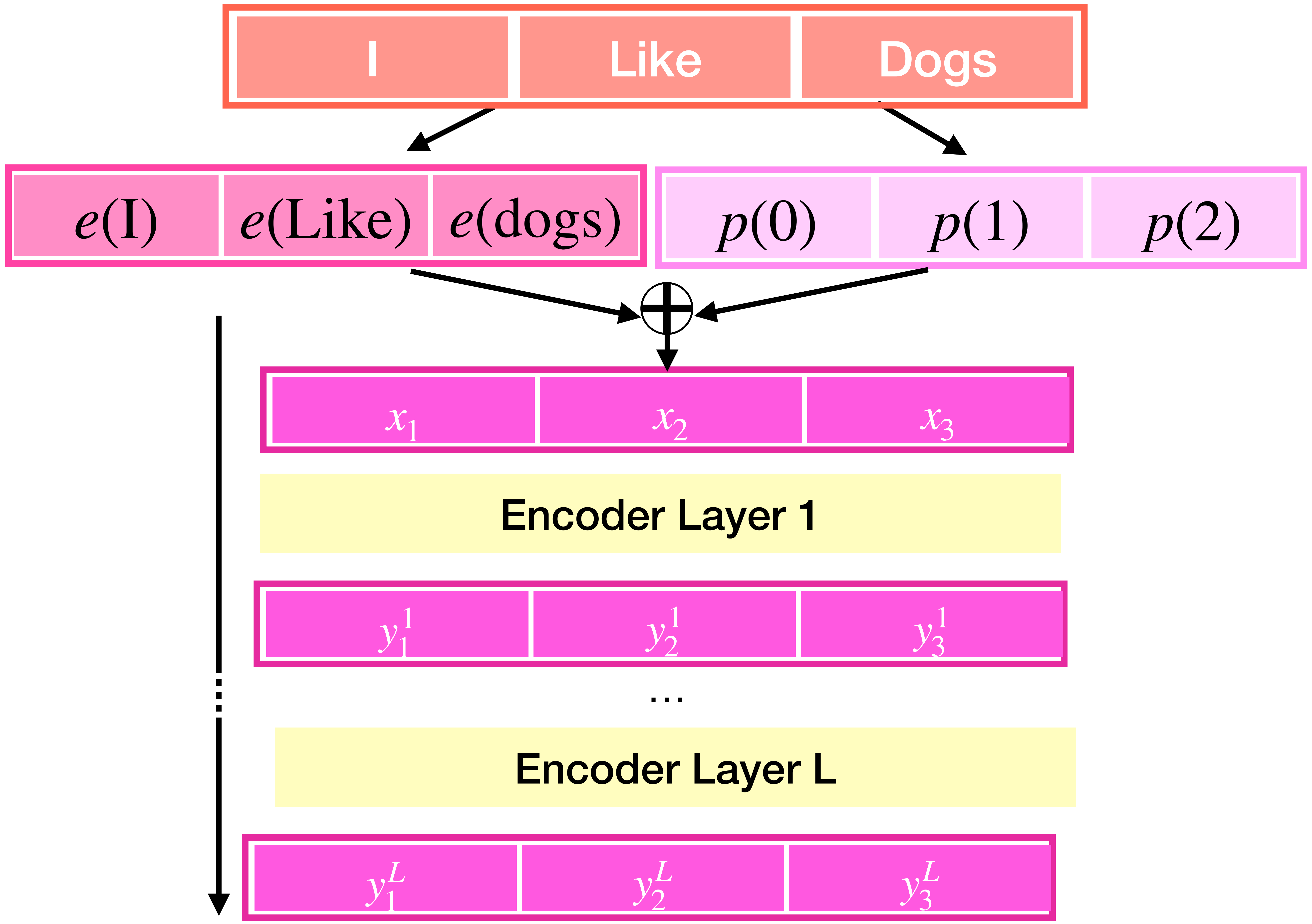


- Receive their entire input 'at once', processing all tokens in parallel
- Have a fixed number of layers, such that the output of one is the input of the next

Computation "progresses" along network depth… not input length

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Transformers



tokens = positionwise_embeddings(input)

indices = positionwise_indices(input)

x = tokens+indices

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

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

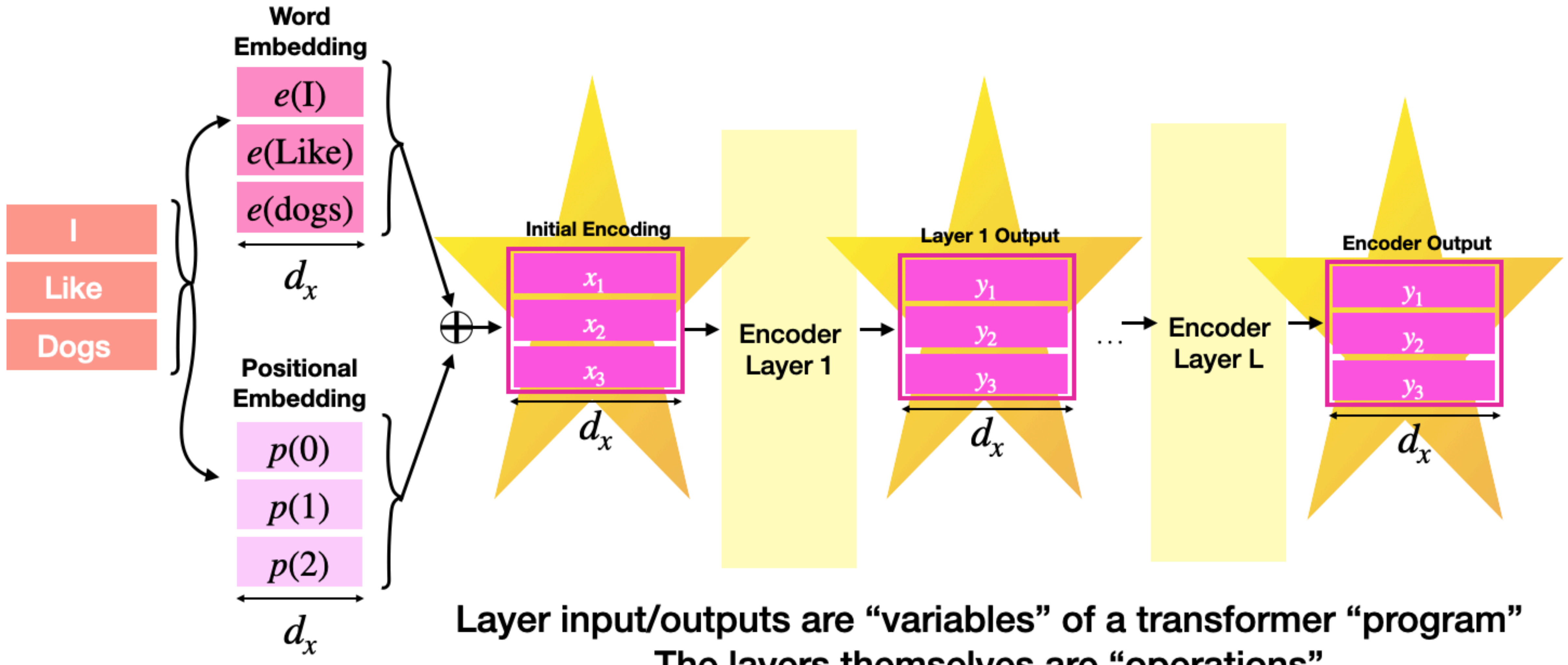
…

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

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

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

RASP (Restricted Access Sequence Processing)

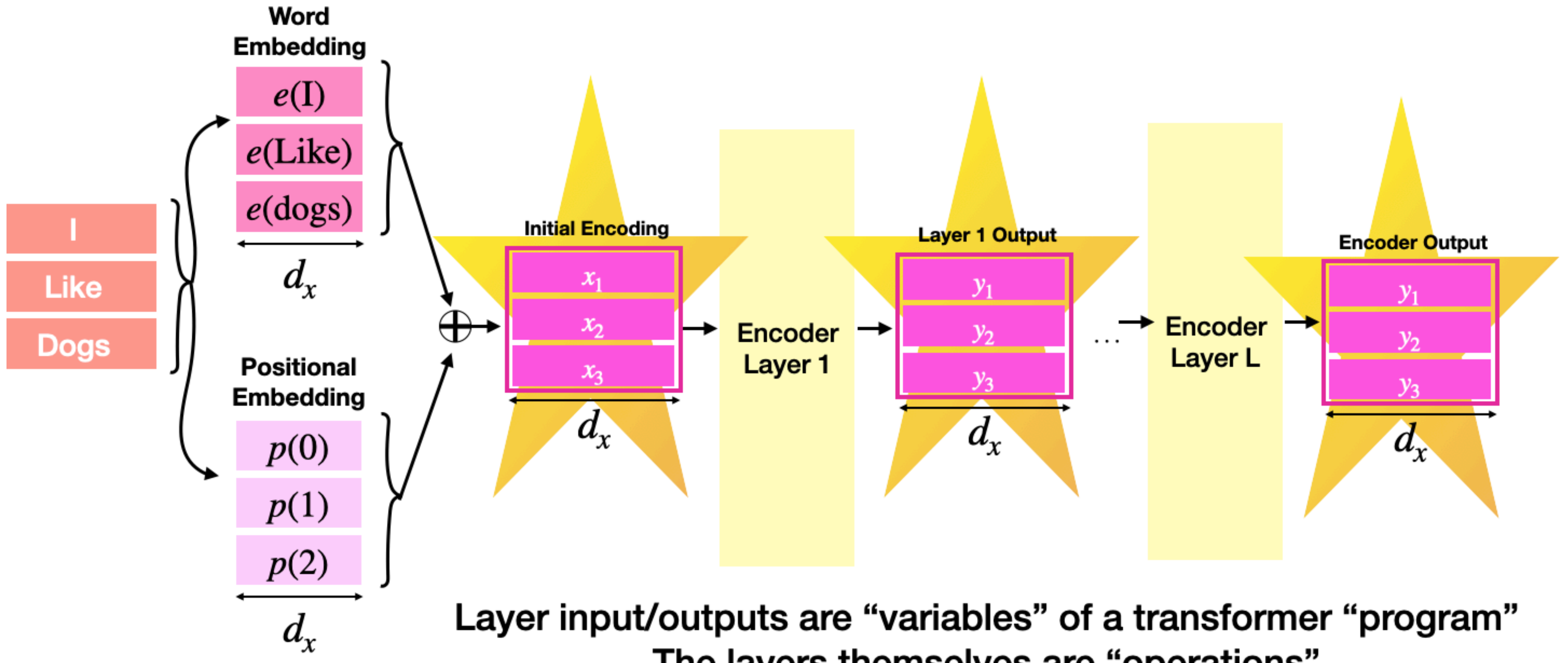


Layer input/outputs are "variables" of a transformer "program"
The layers themselves are "operations"

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

RASP (Restricted Access Sequence Processing)

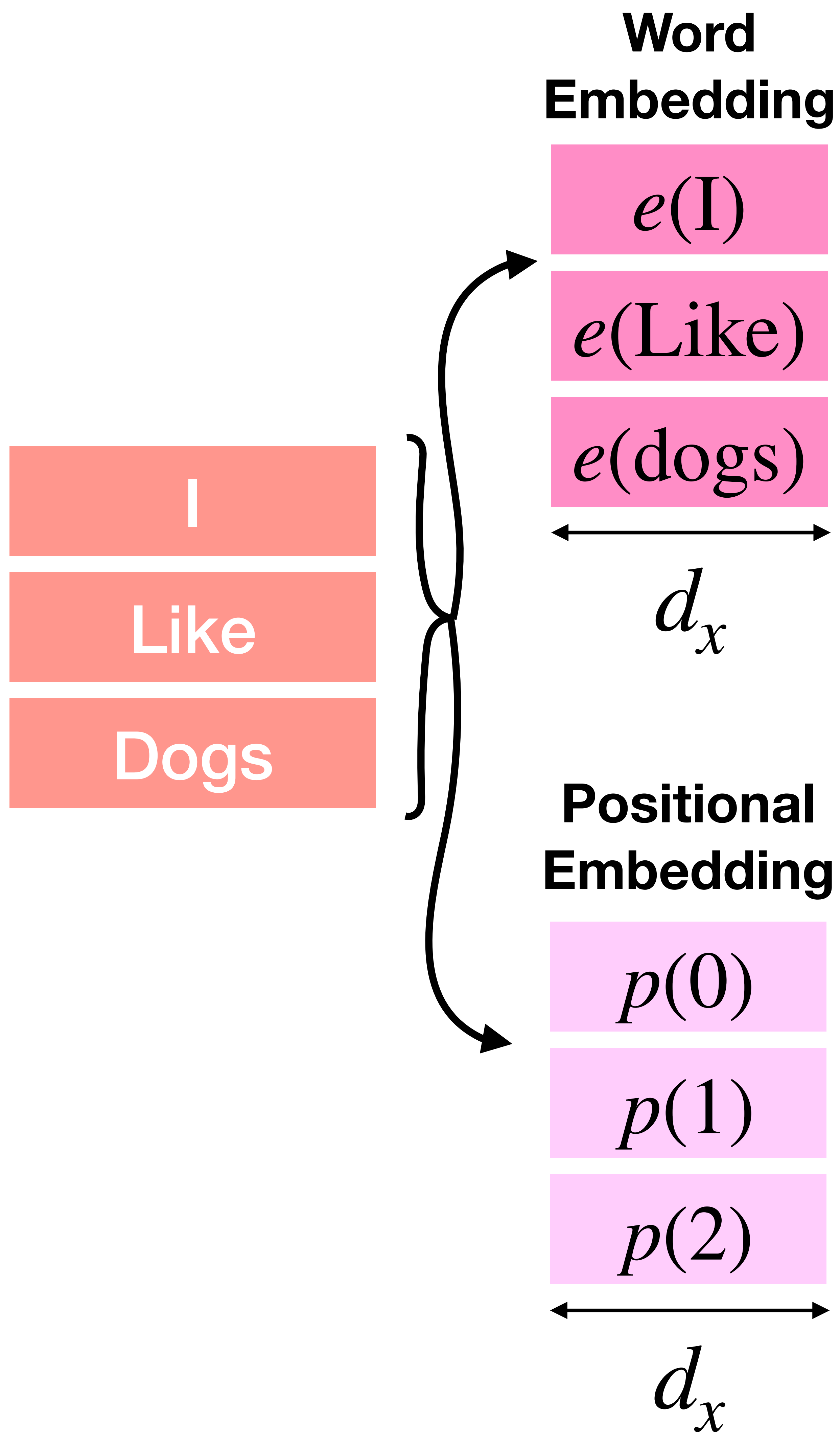
- A transformer-encoder is a sequence to sequence function (“sequence operator”, or, “**s-op**”)
- Its **layers apply operations** to the sequences
- **RASP builds s-ops**, constrained to a transformer’s inputs and possible operations
    - (The s-ops are the transformer abstractions!)



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

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

RASP base s-ops



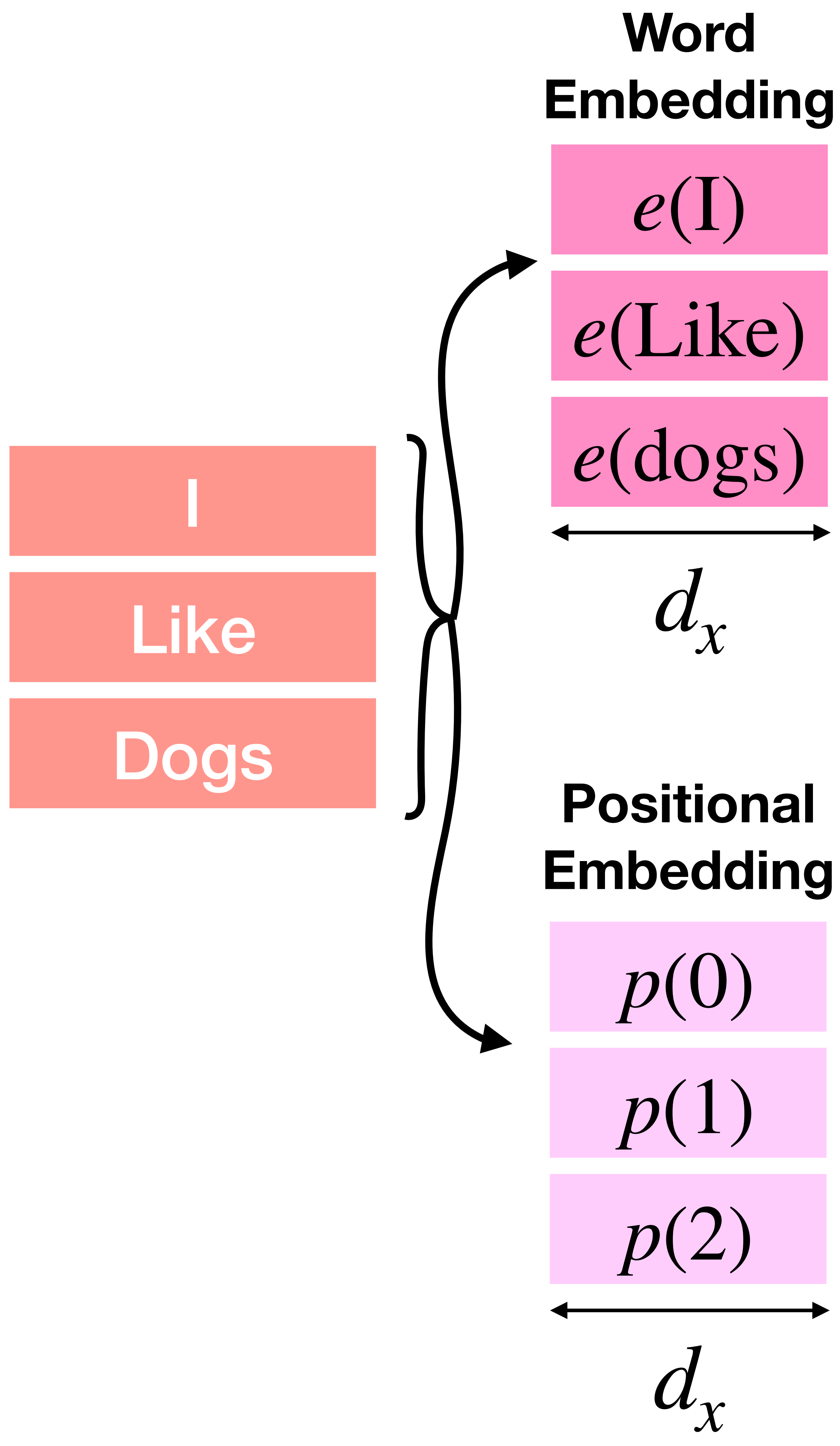
The information before a transformer has done anything ("0 layer transformer")

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)

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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)
```

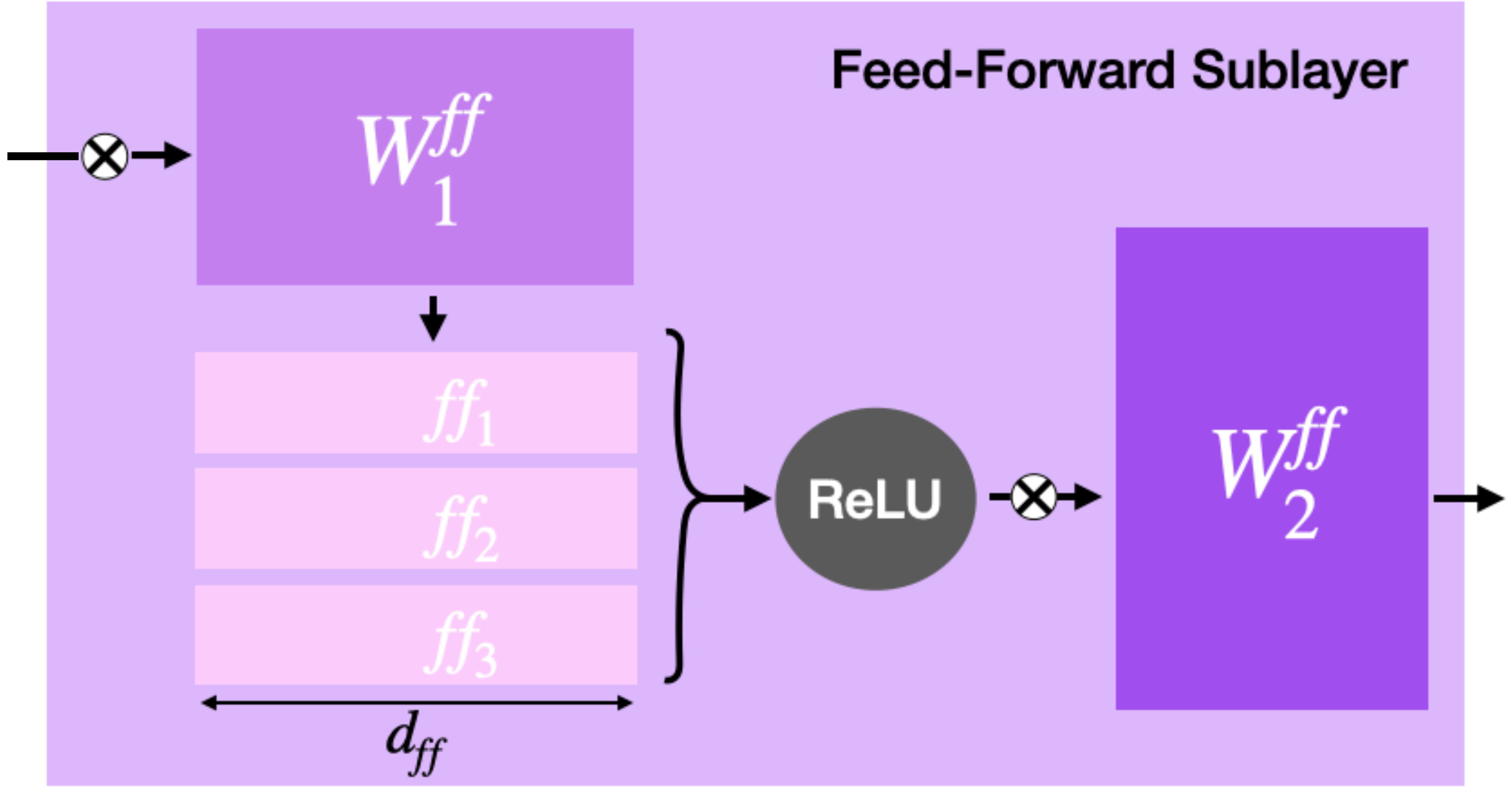
To know what operations RASP may have, we must inspect the transformer-encoder layers!

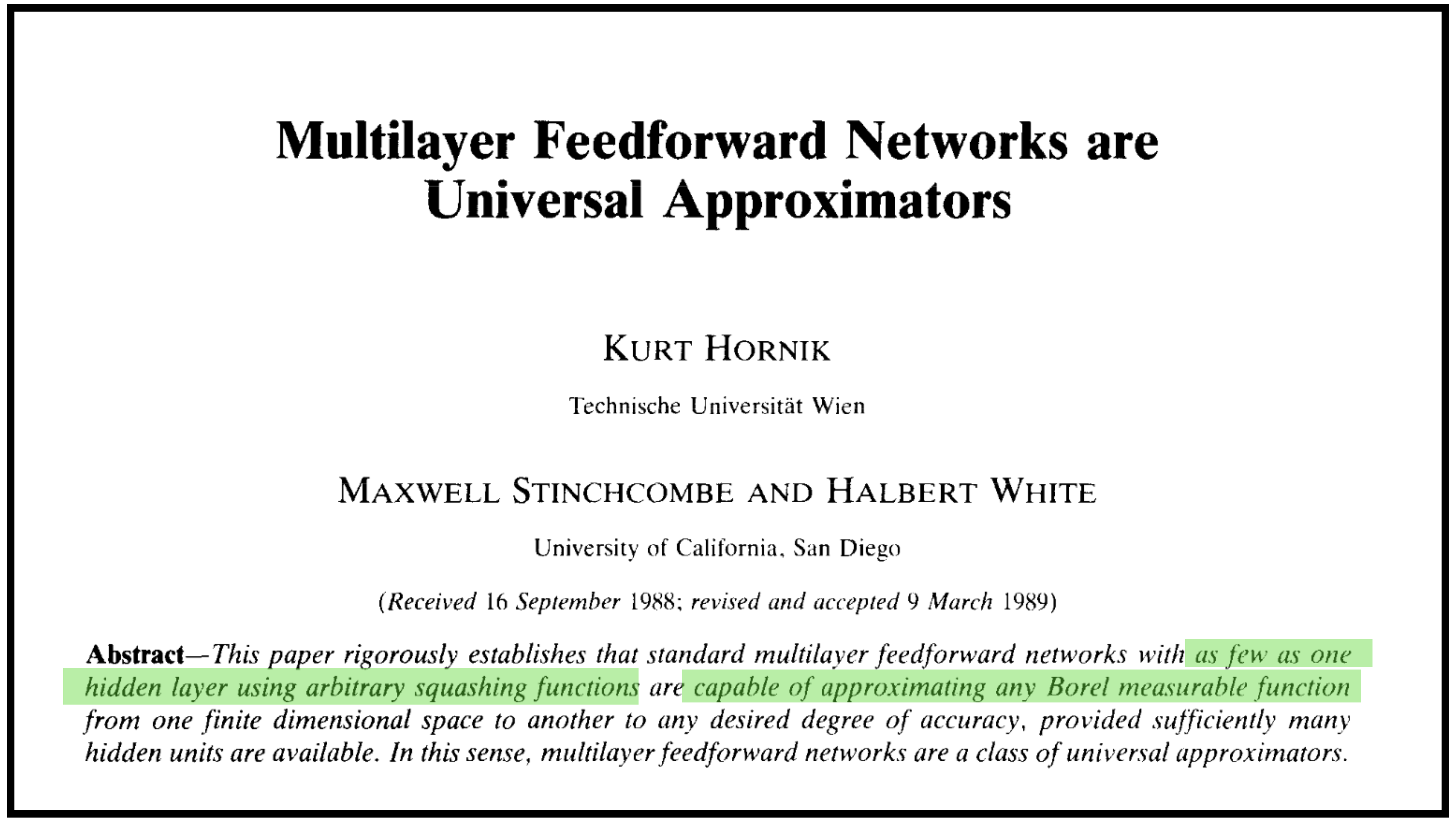
Transformer-Encoder Layer

Feed-Forward Sublayer

Multilayer Feedforward Networks are Universal Approximators (Hornik et al, 1989)

Feed-Forward Sublayer





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)
```

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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?

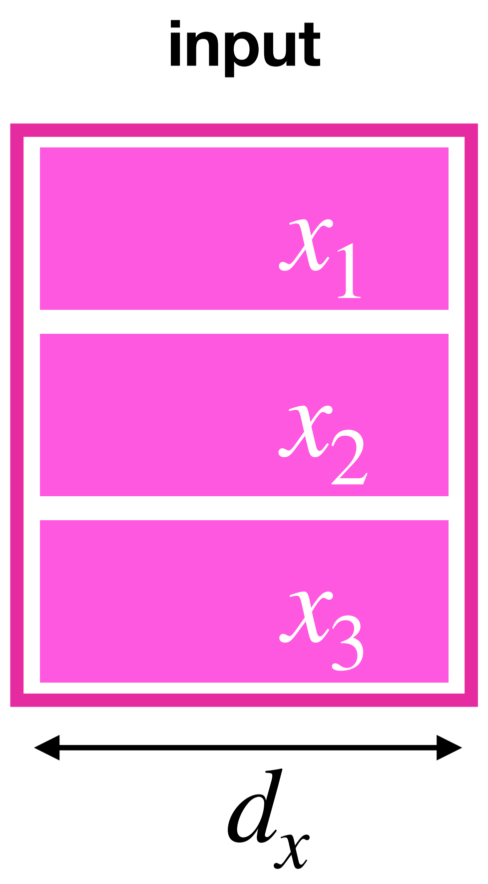
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Attention Sublayer

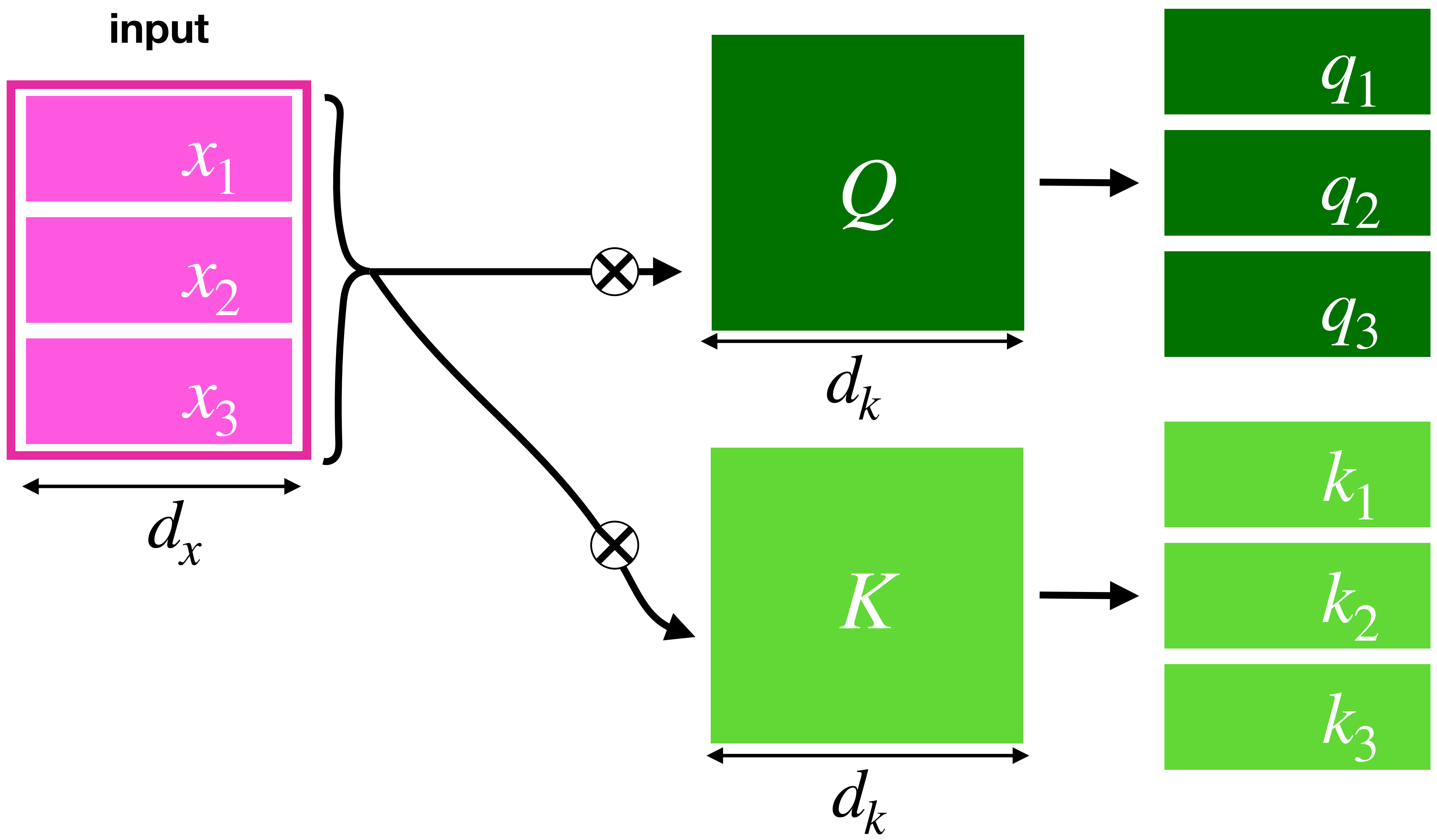
Background - Multi Head Attention

Starting from single-head attention…

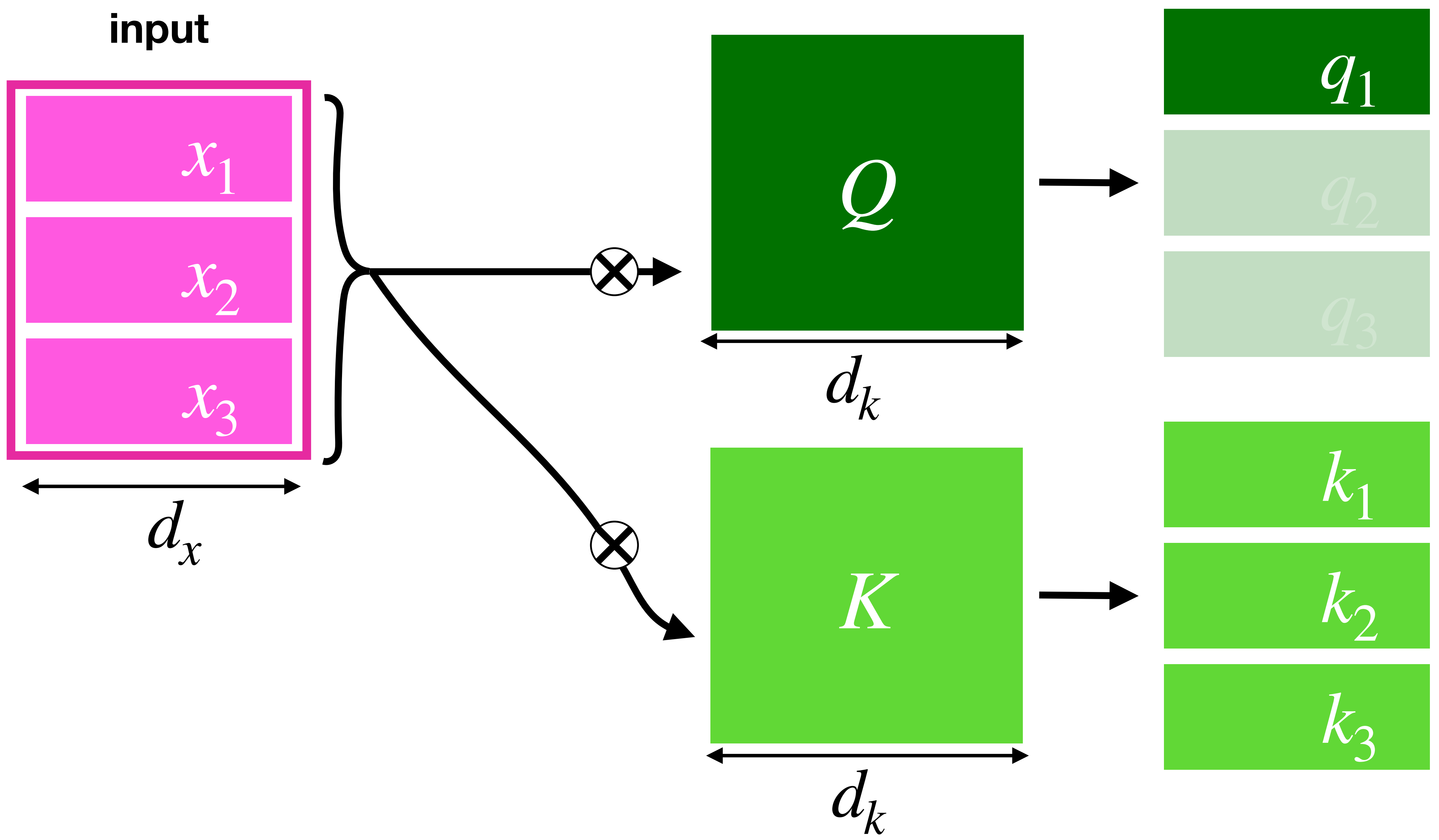
Background - Self Attention (Single Head)



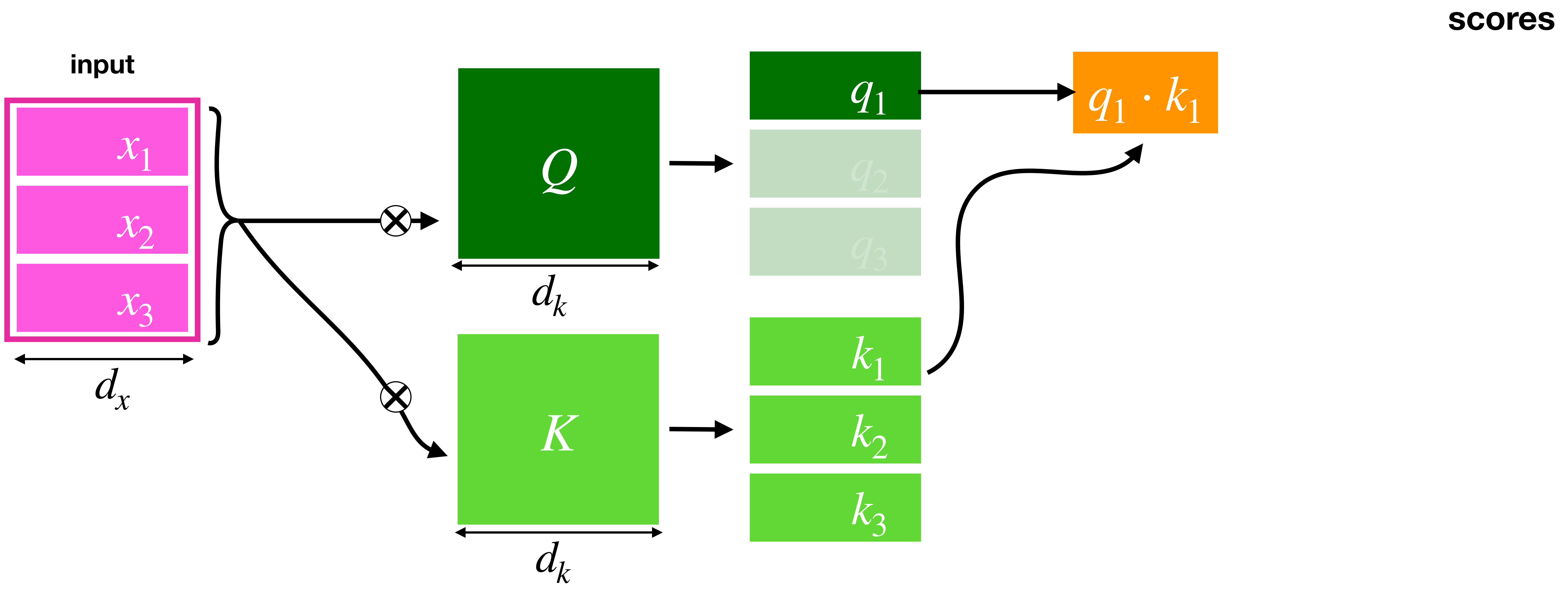
Background - Self Attention (Single Head)



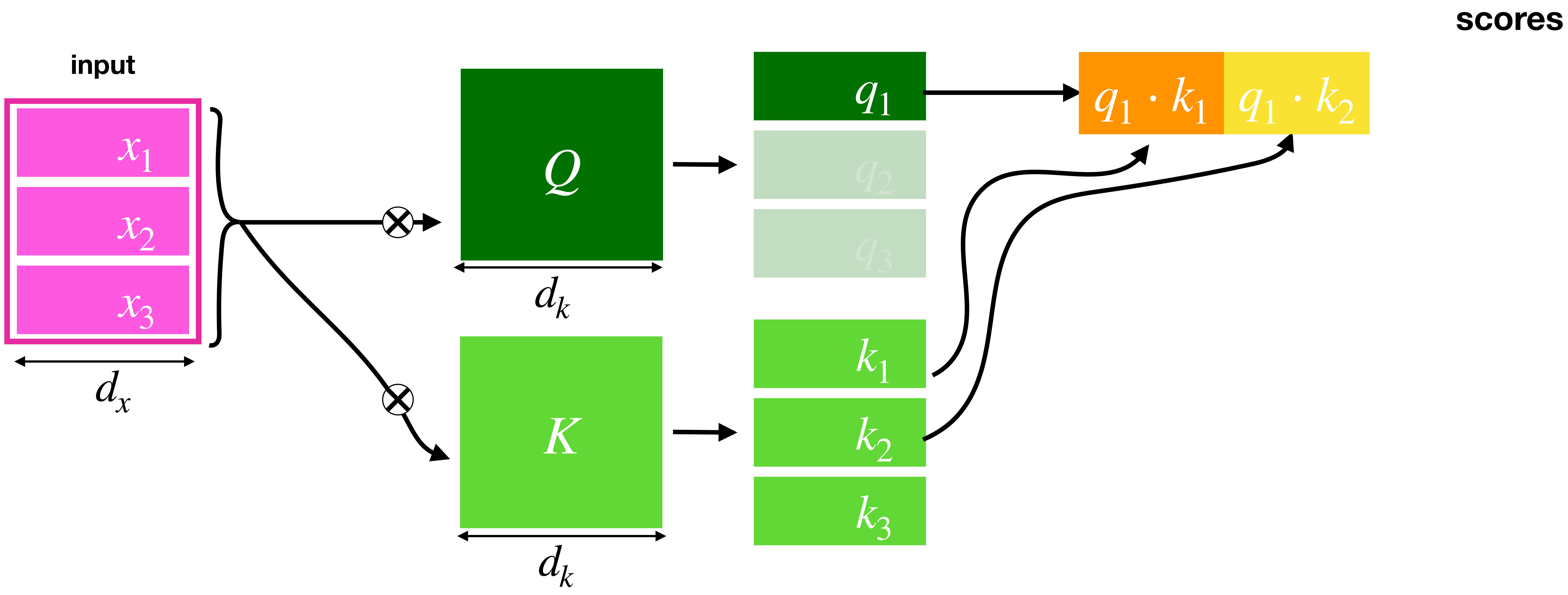
Background - Self Attention (Single Head)



Background - Self Attention (Single Head)

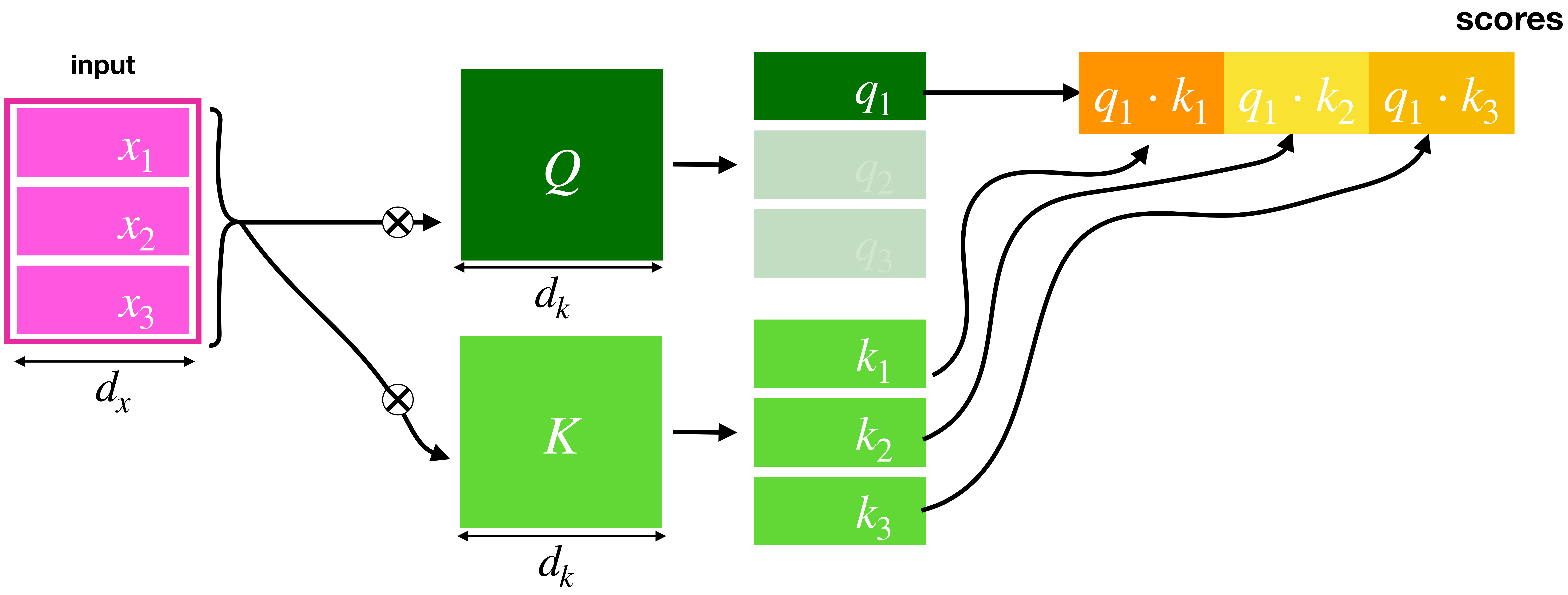


Background - Self Attention (Single Head)



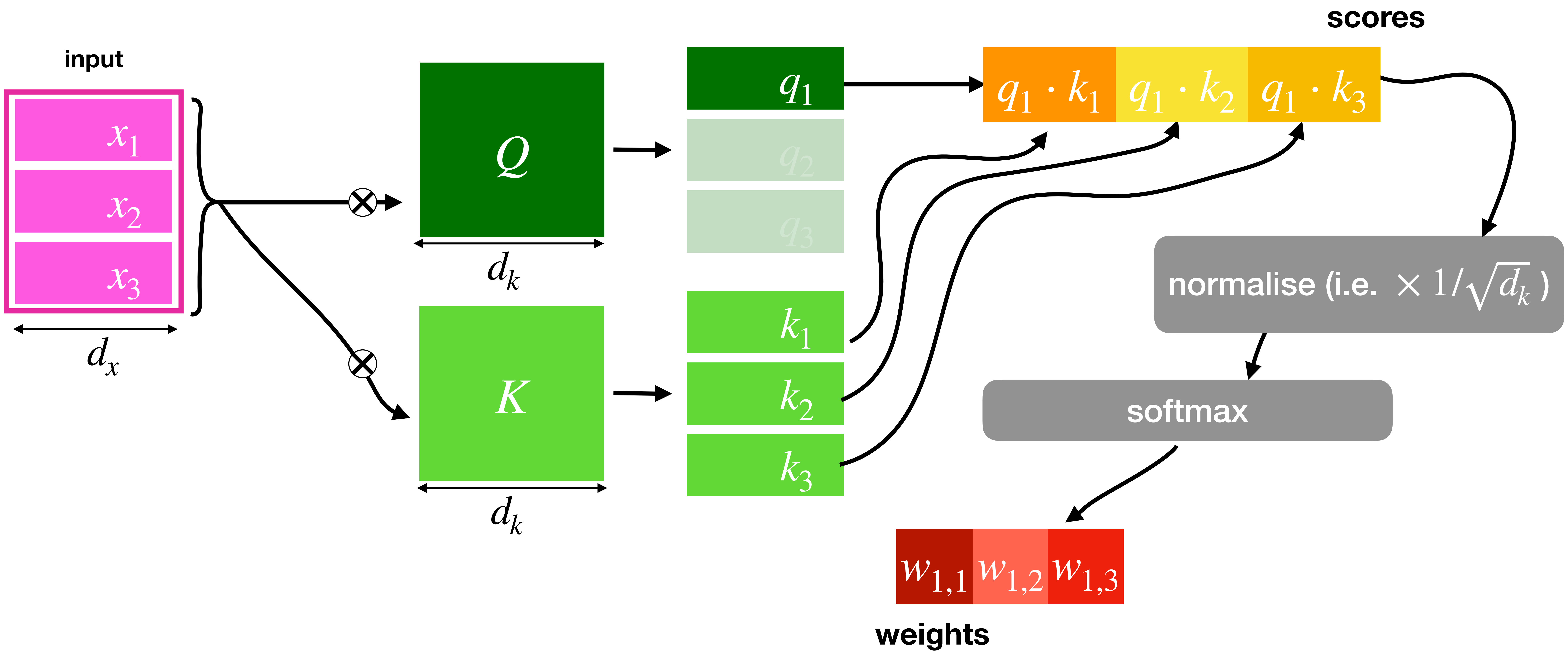
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Background - Self Attention (Single Head)

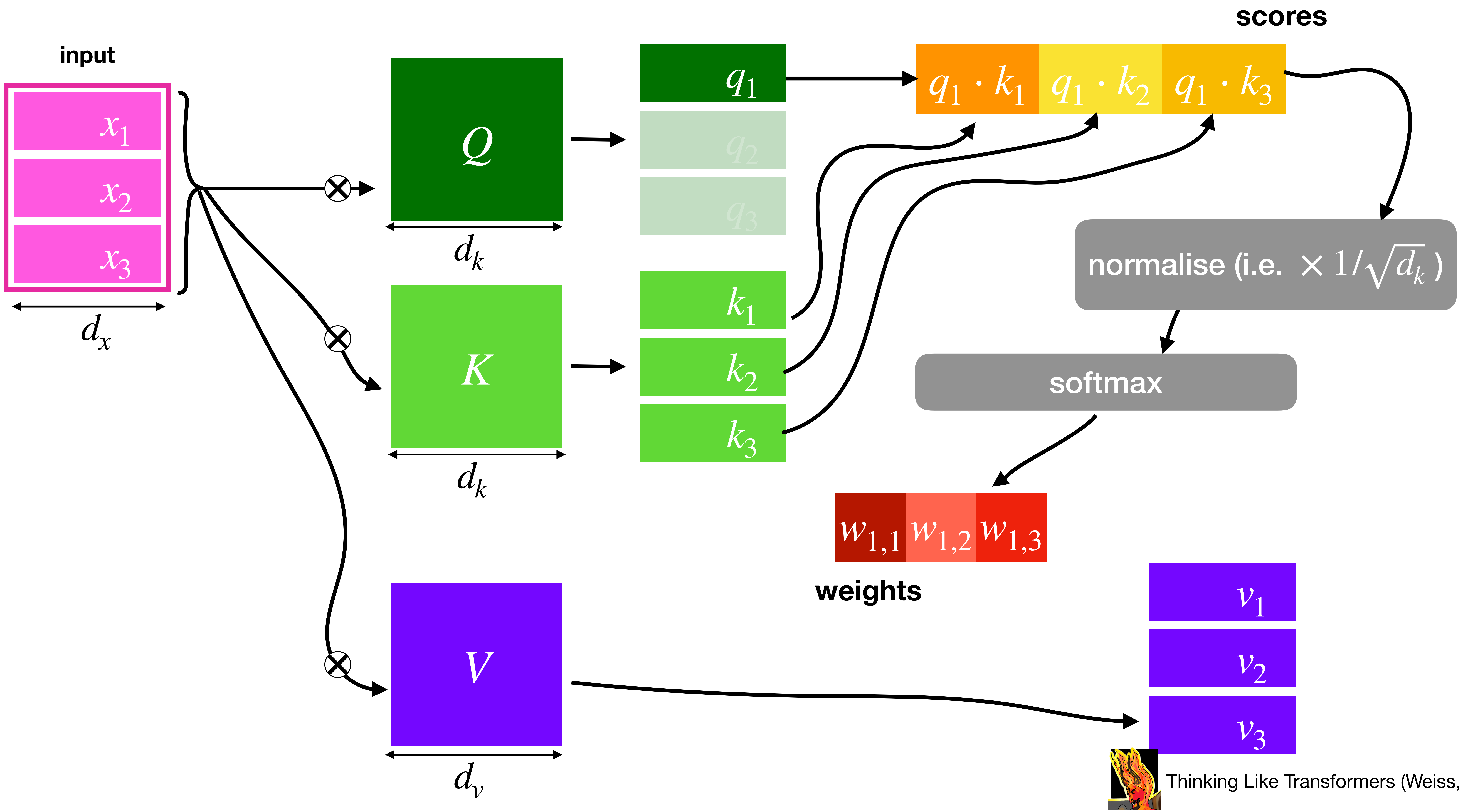


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Background - Self Attention (Single Head)

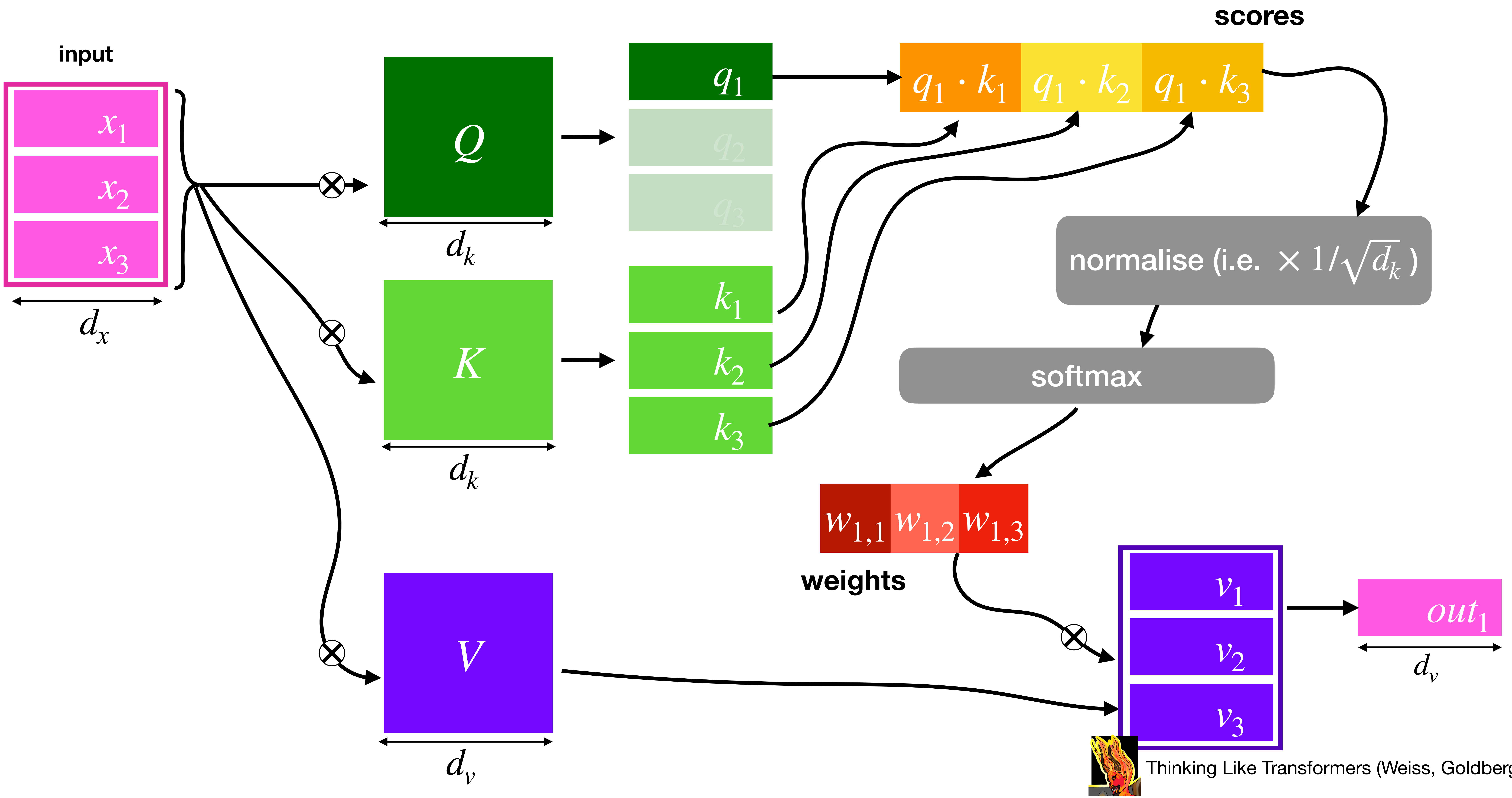


Background - Self Attention (Single Head)



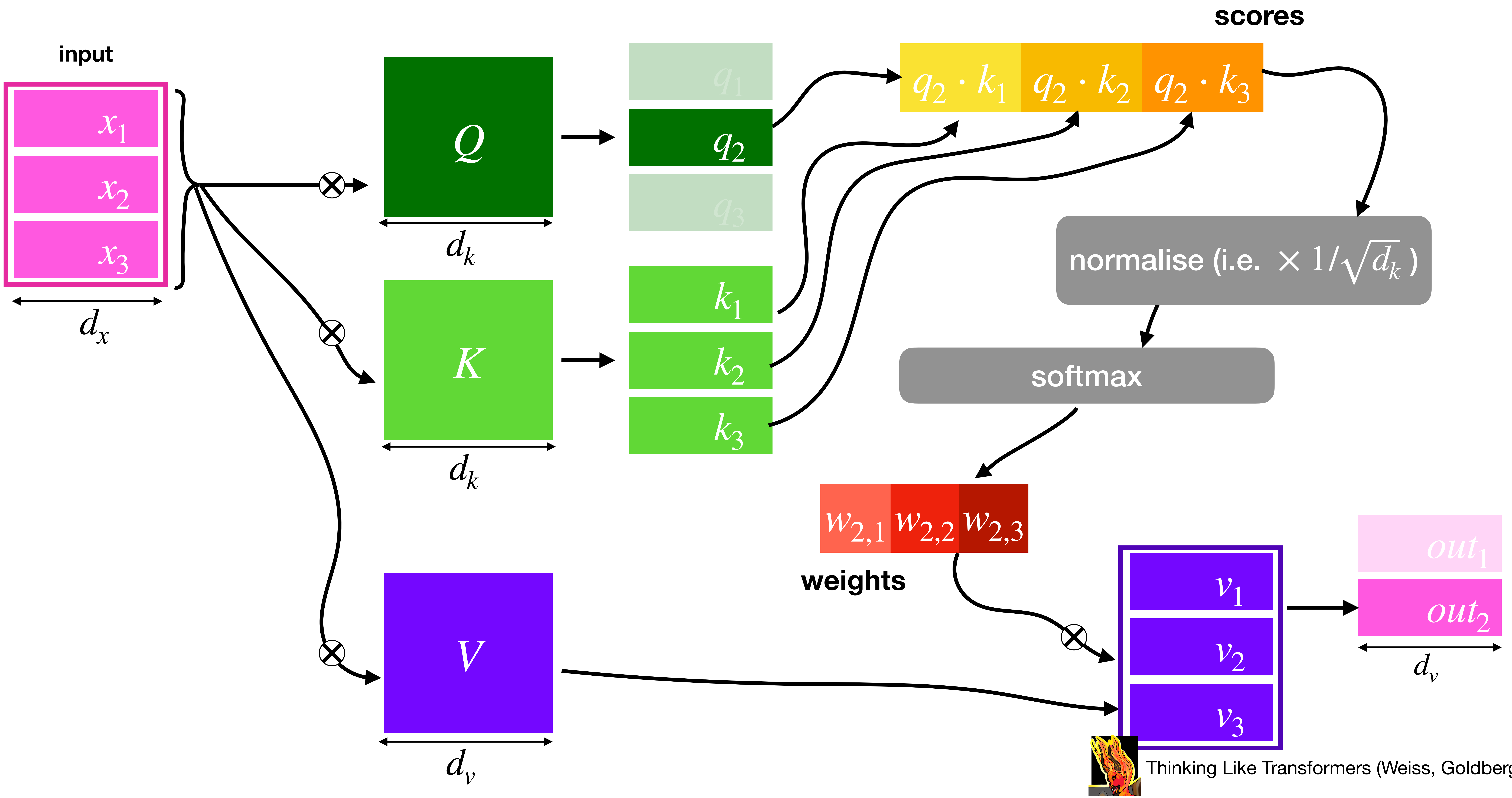
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Background - Self Attention (Single Head)



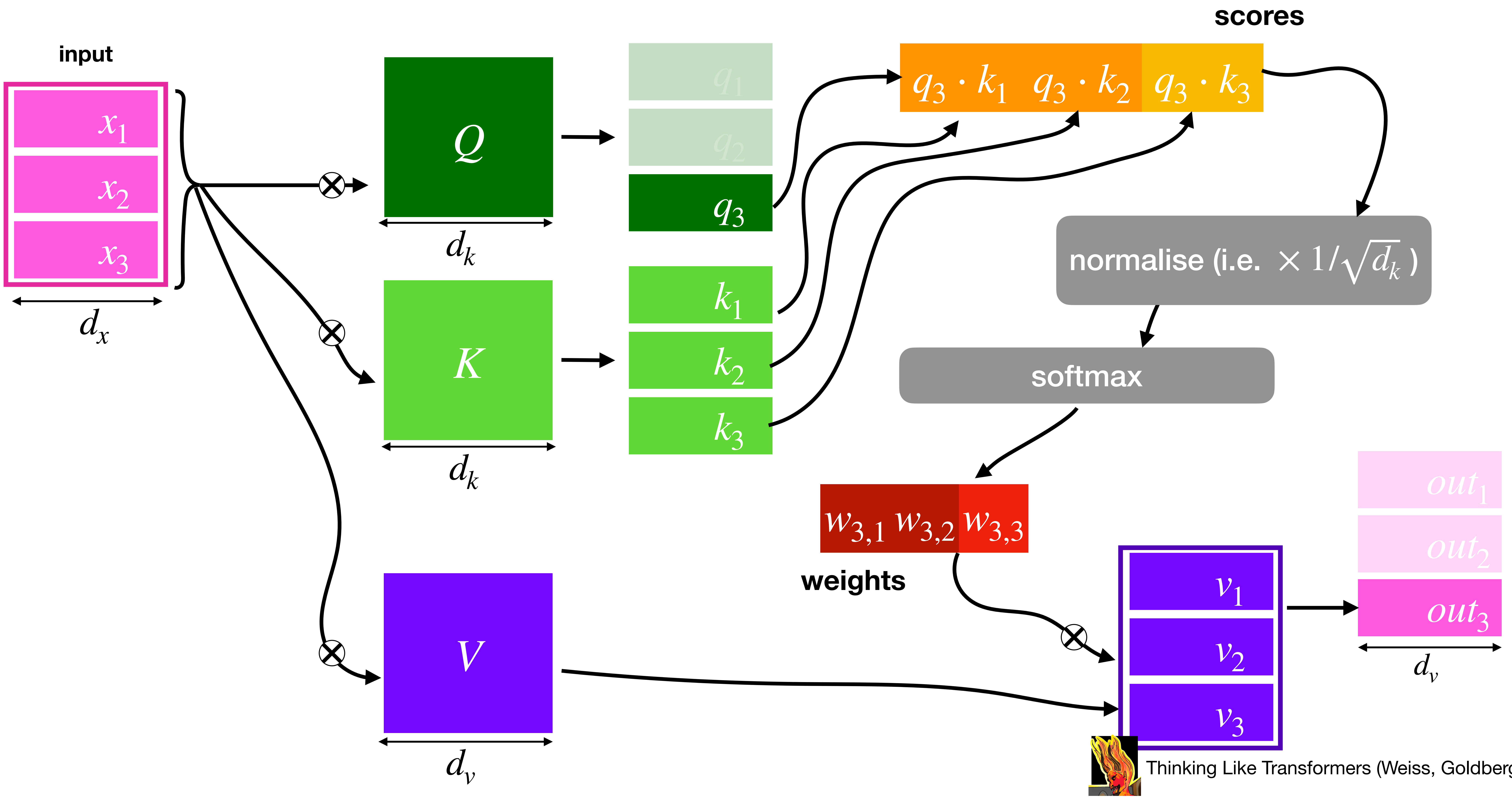
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Background - Self Attention (Single Head)



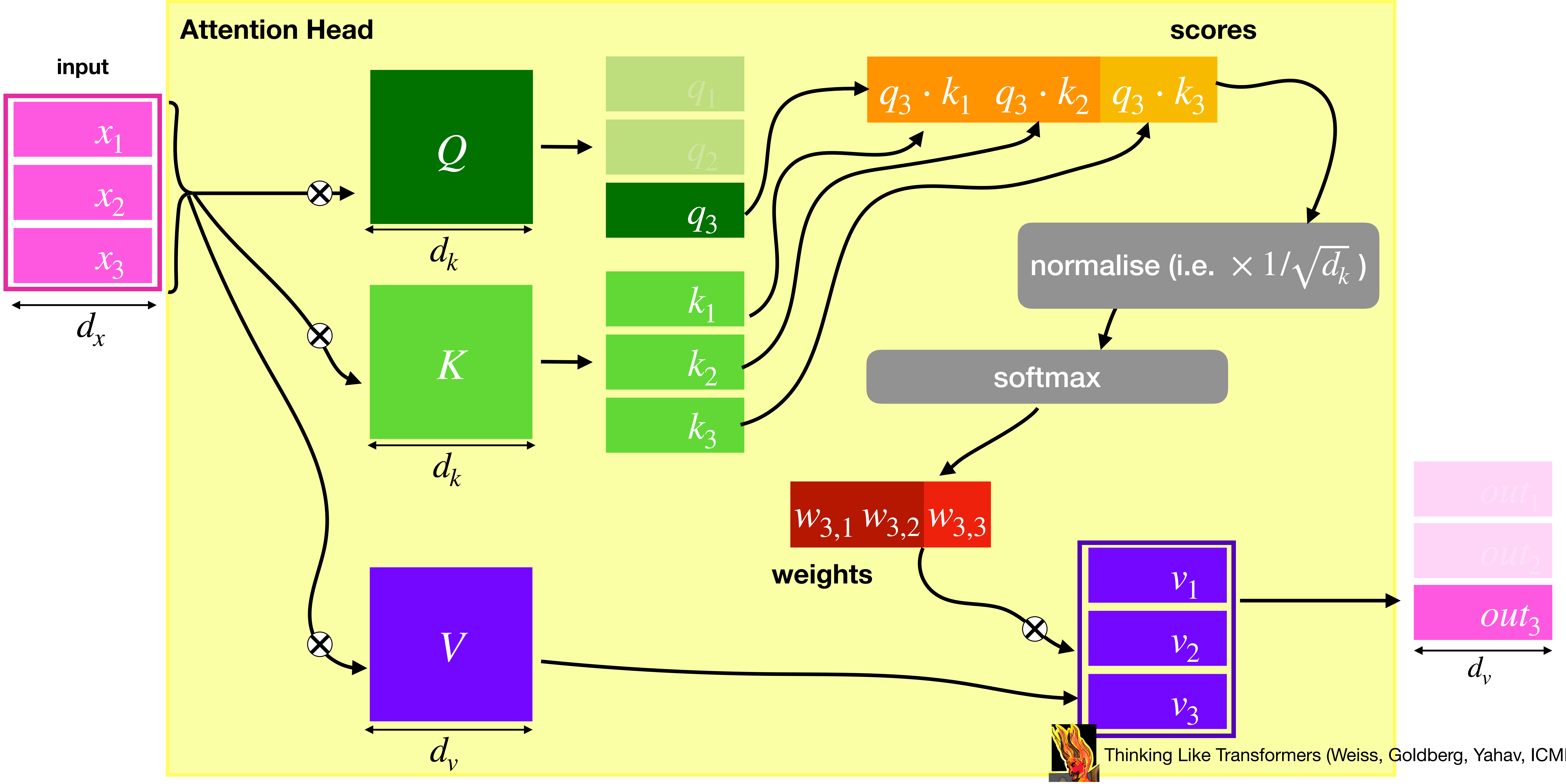
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Background - Self Attention (Single Head)



Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

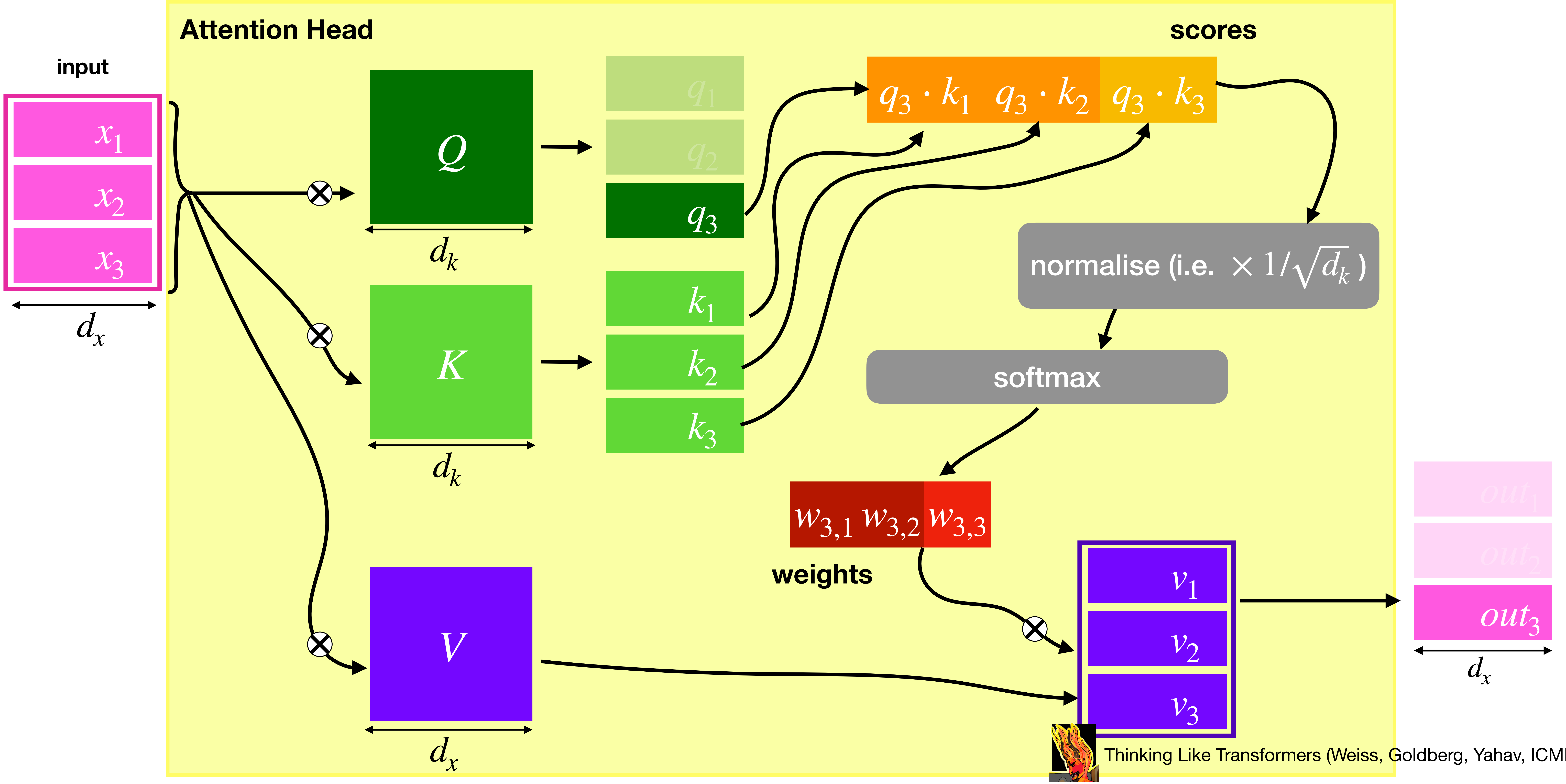
Background - Self Attention (Single Head)



Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

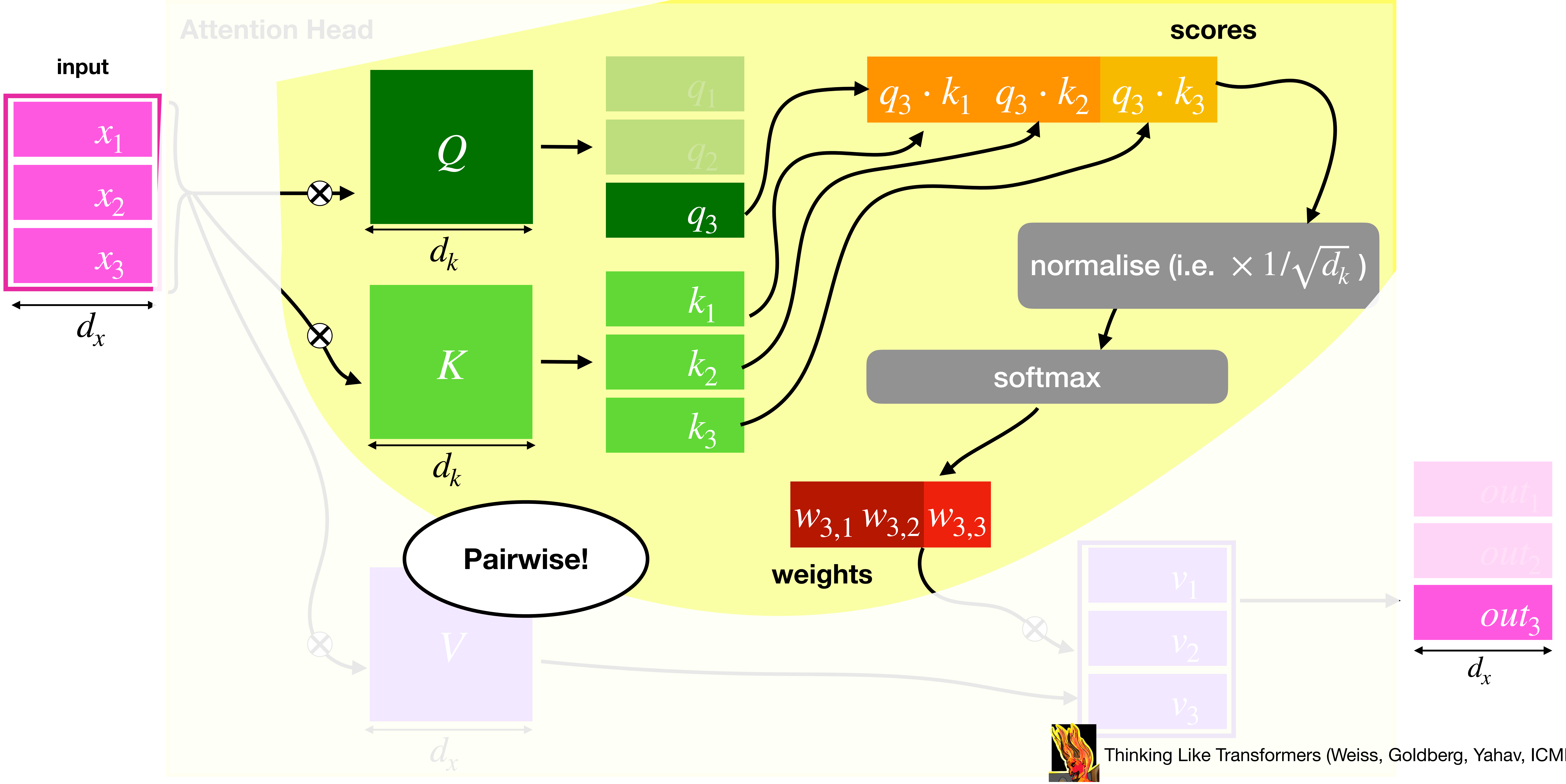
So, how do we present an attention head?

Self Attention (Single Head)



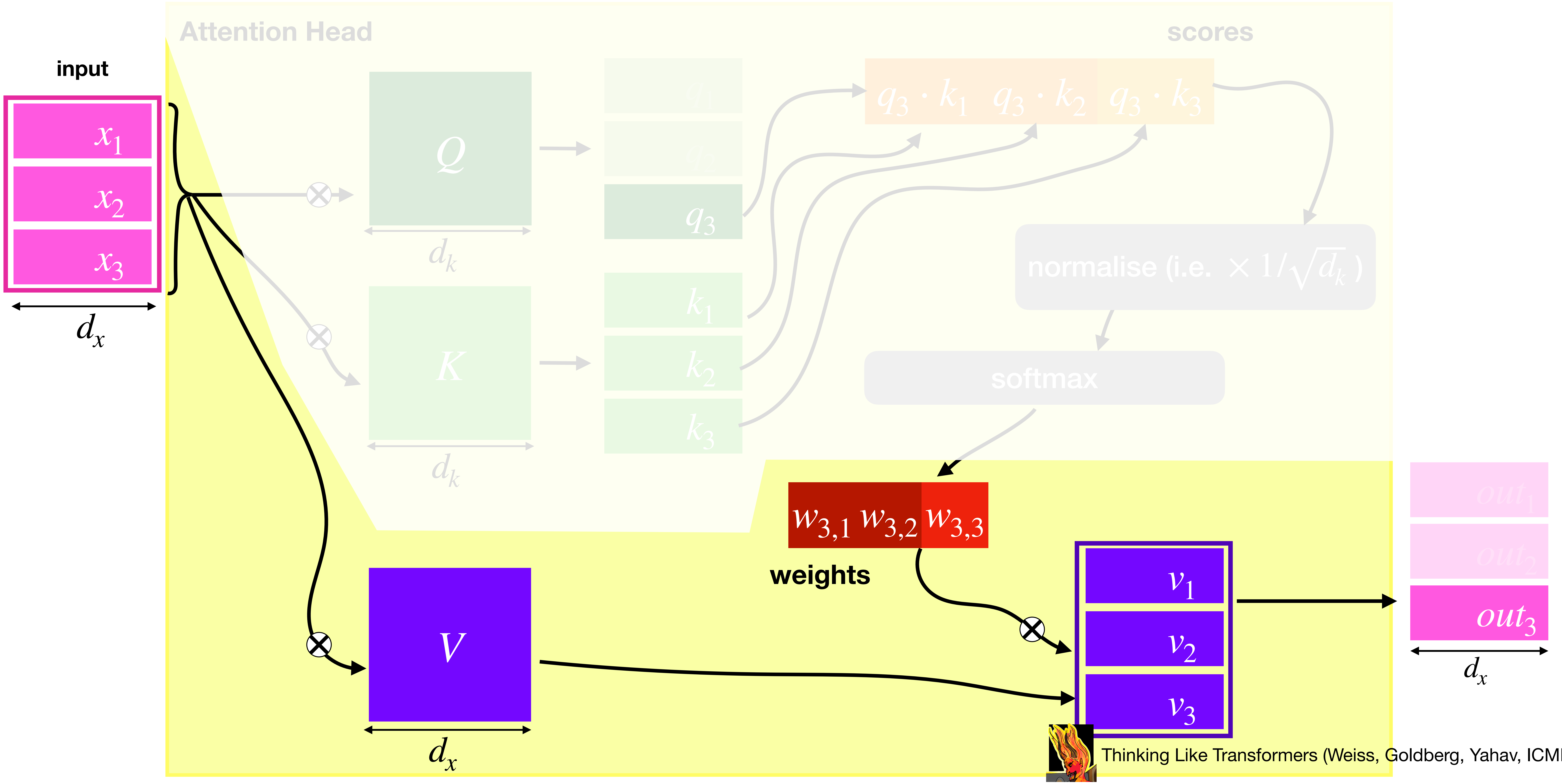
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Self Attention (Single Head)



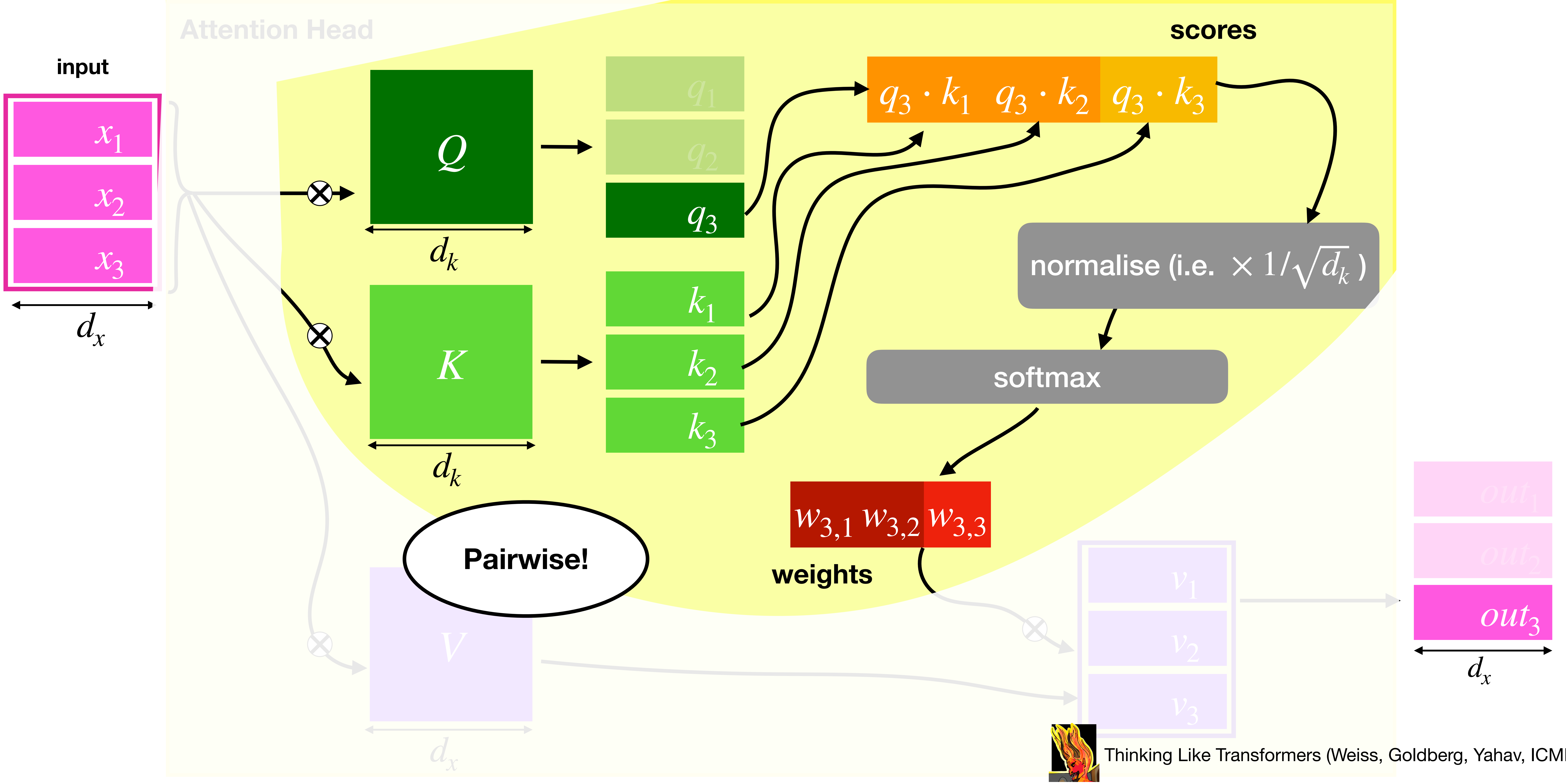
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Self Attention (Single Head)



Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Single Head: Scoring ↔ Selecting



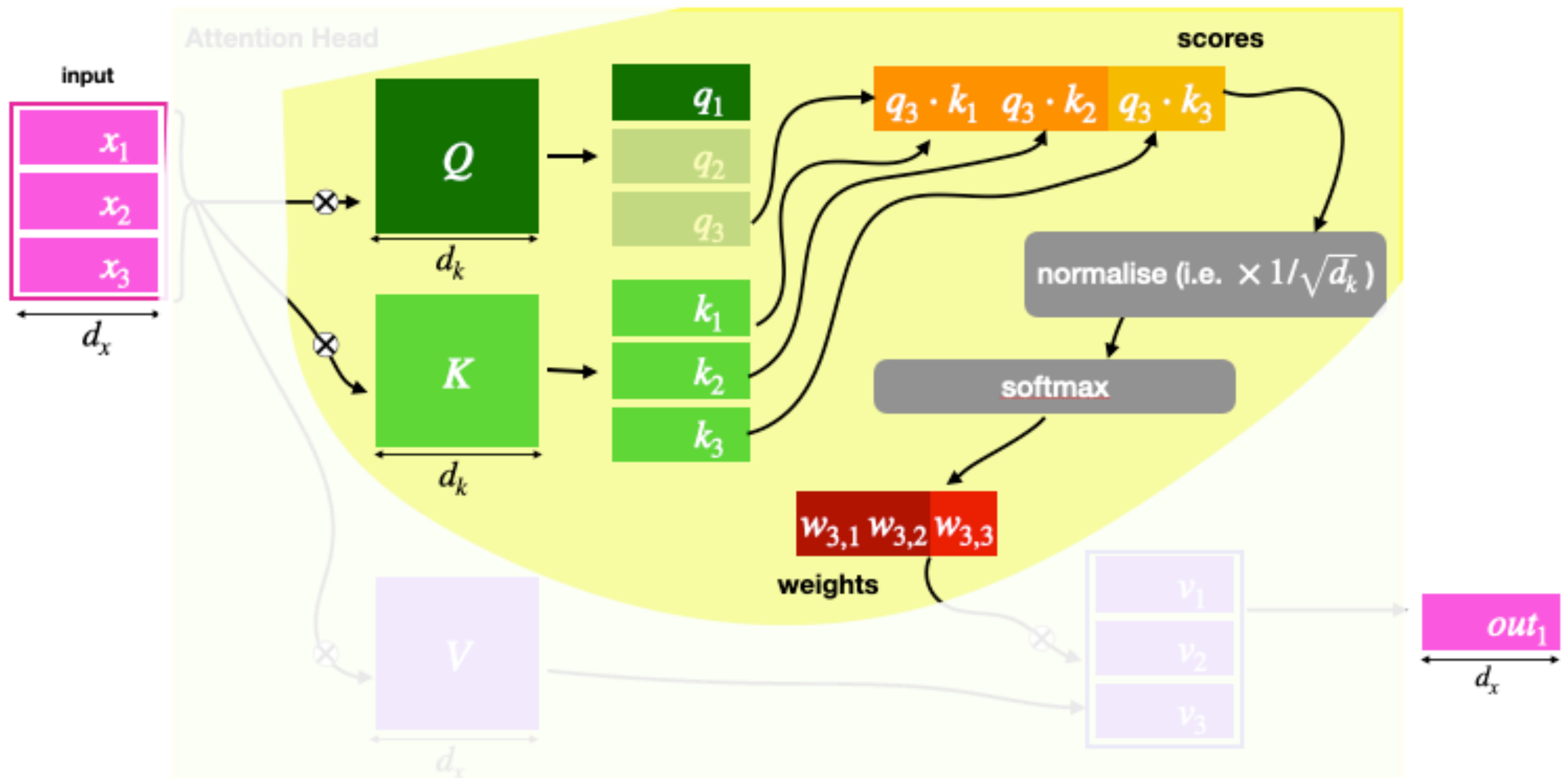
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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



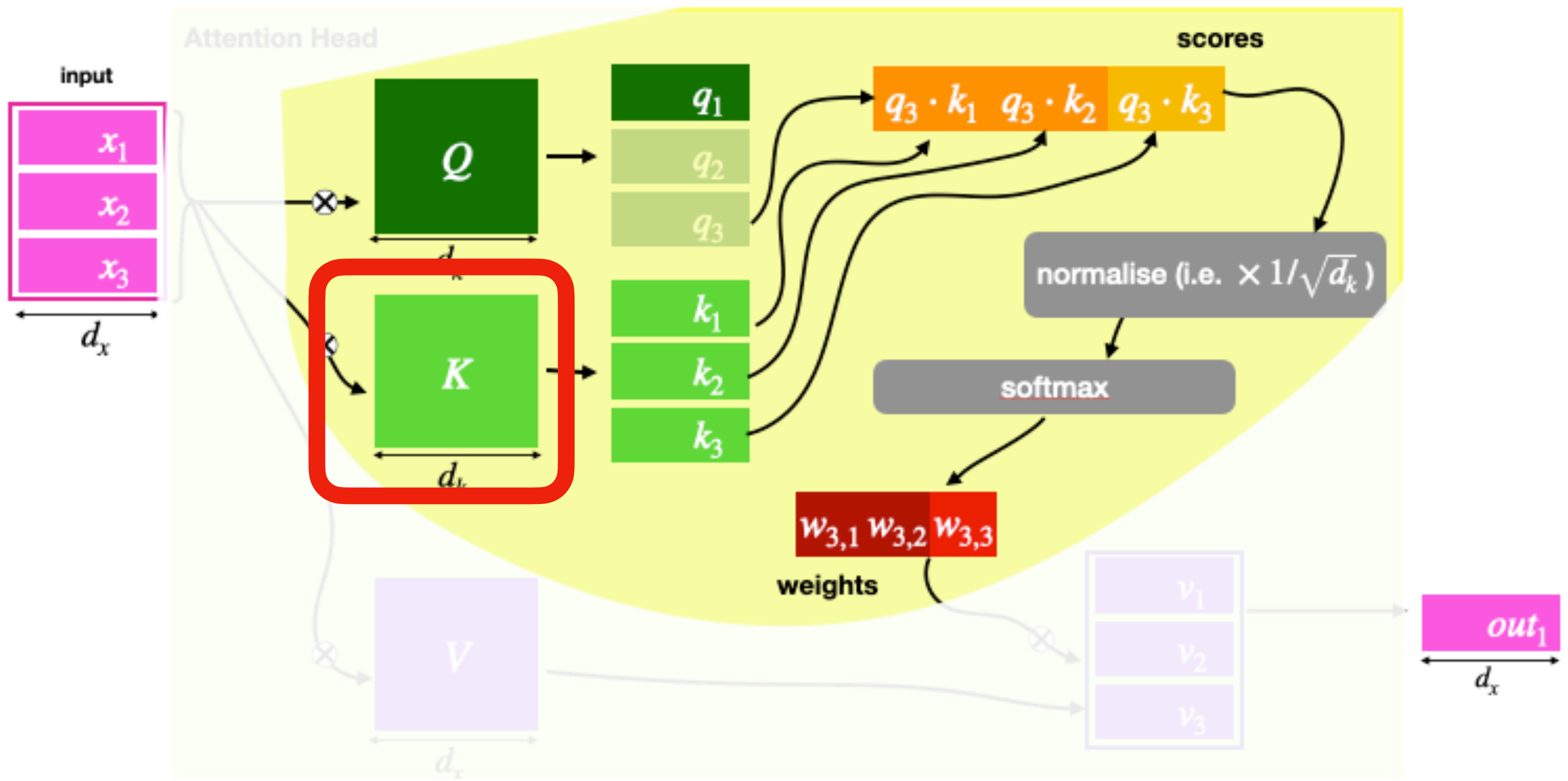
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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



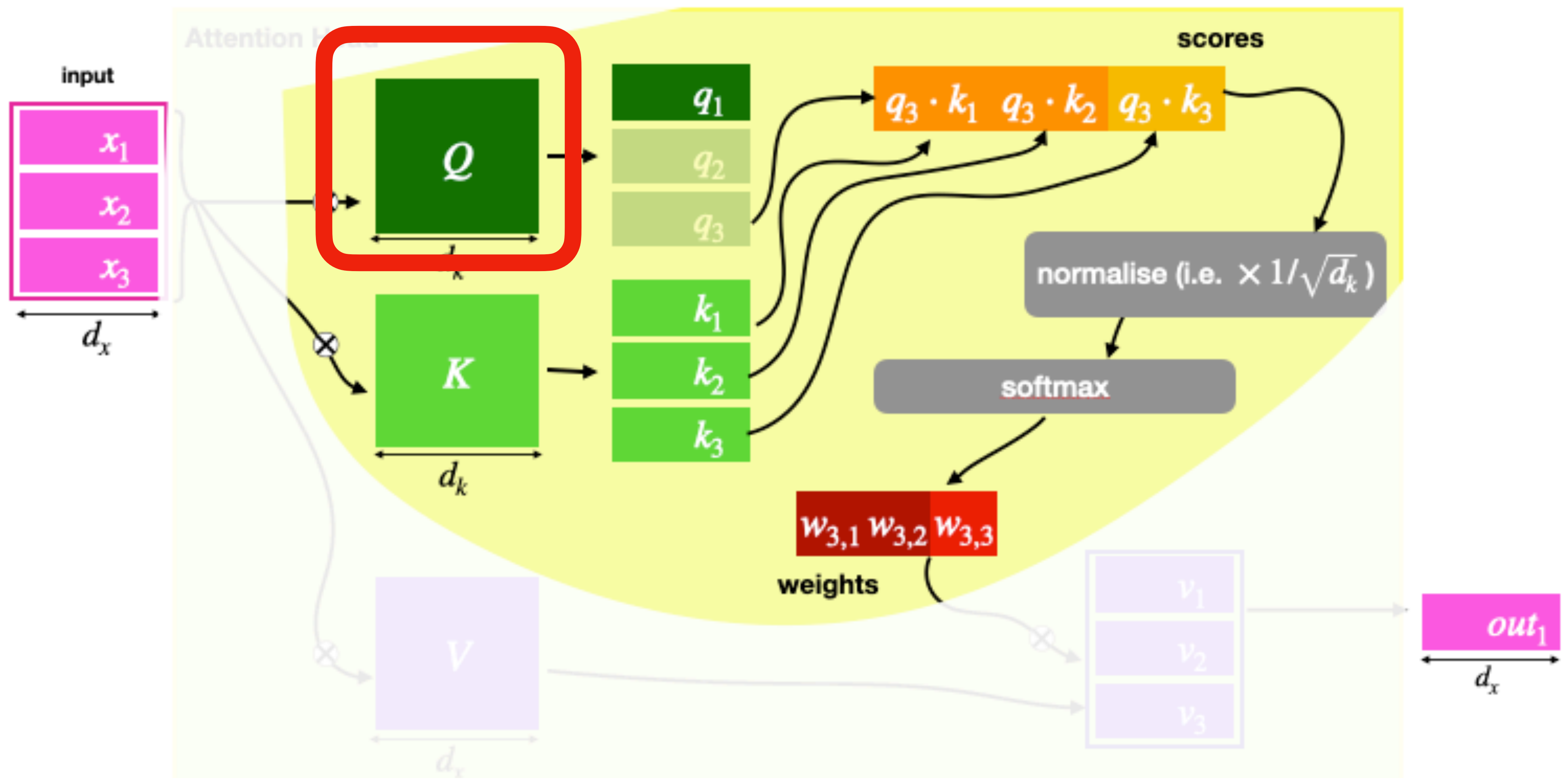
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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



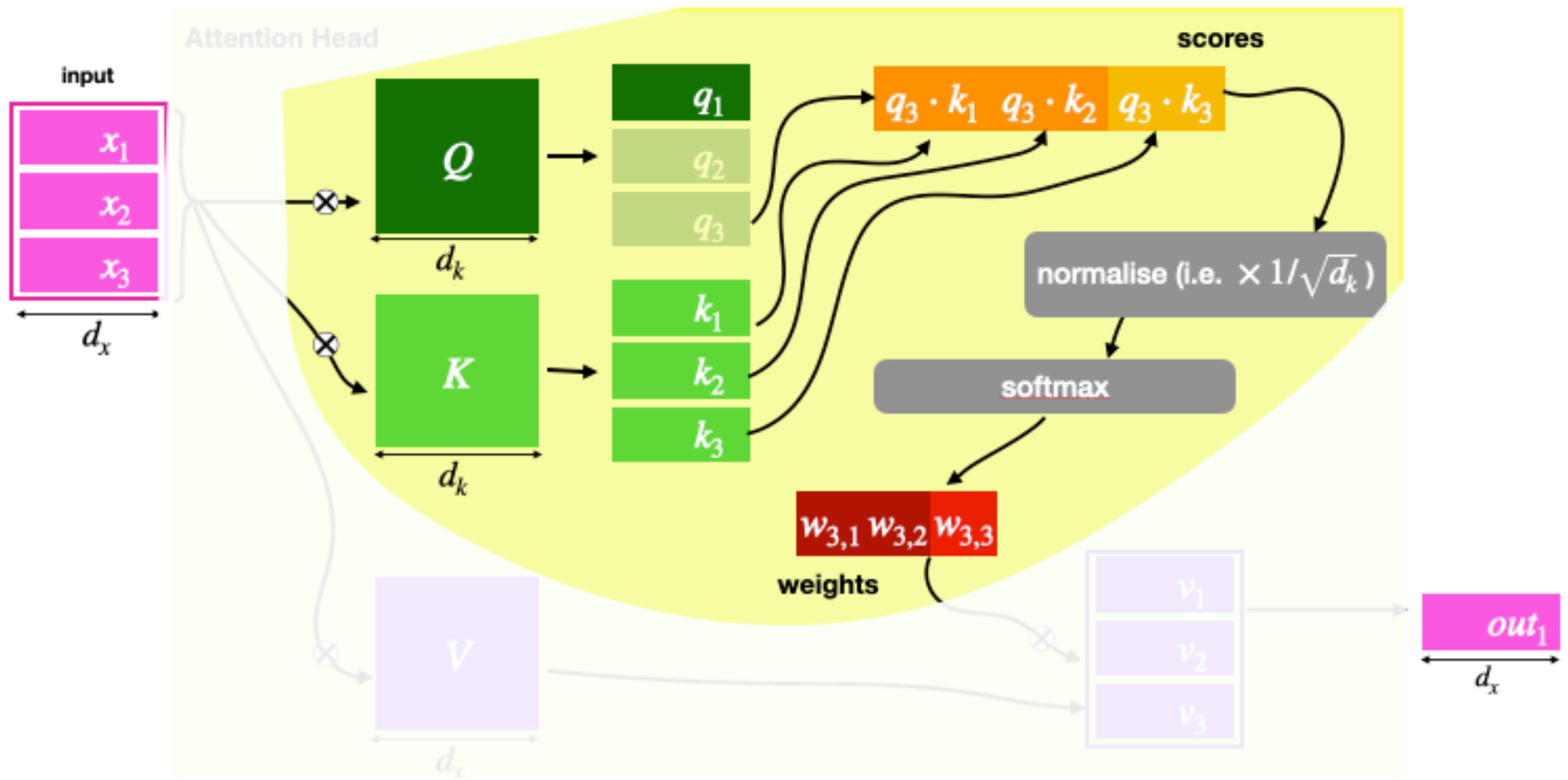
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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



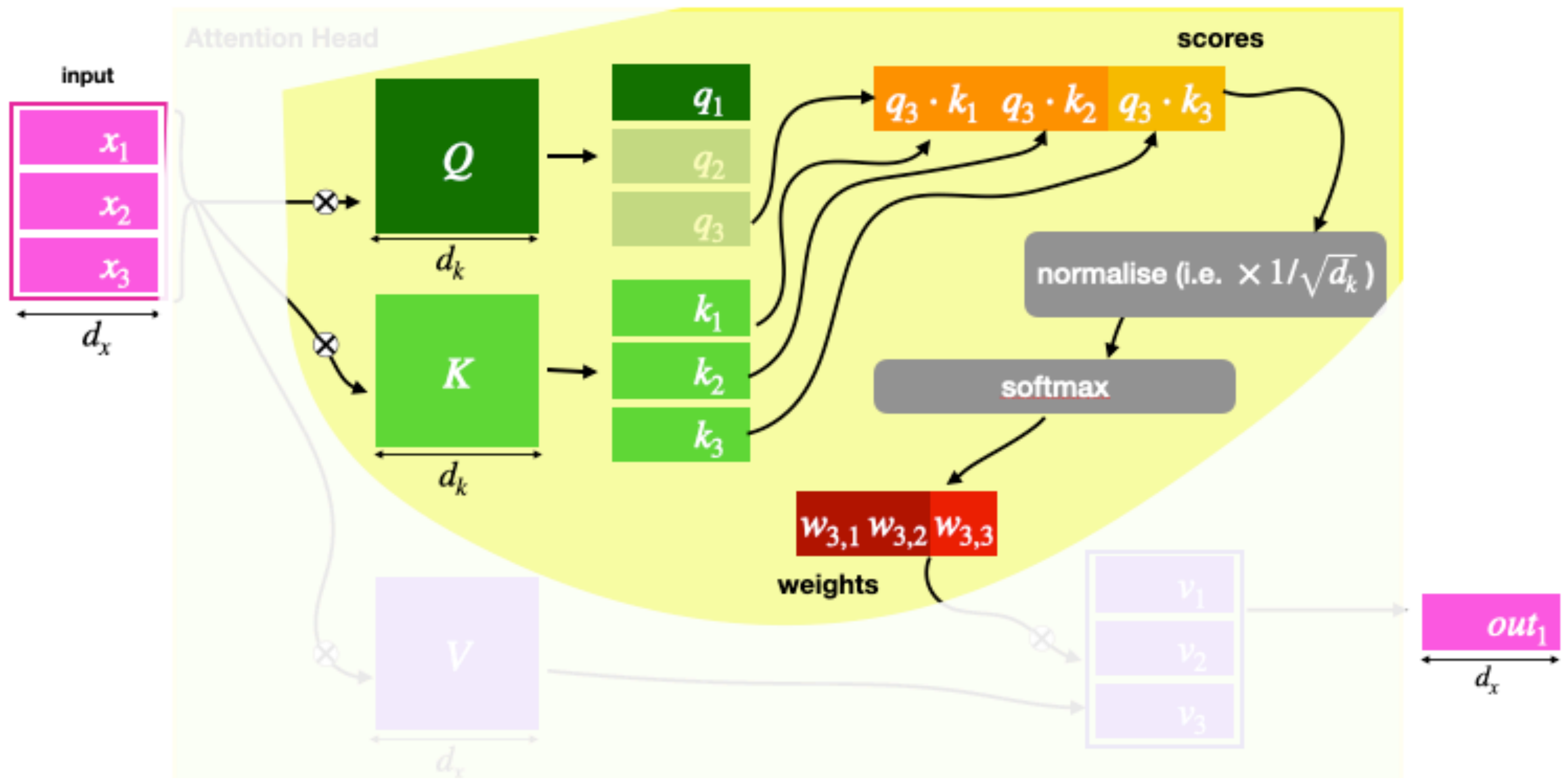
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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



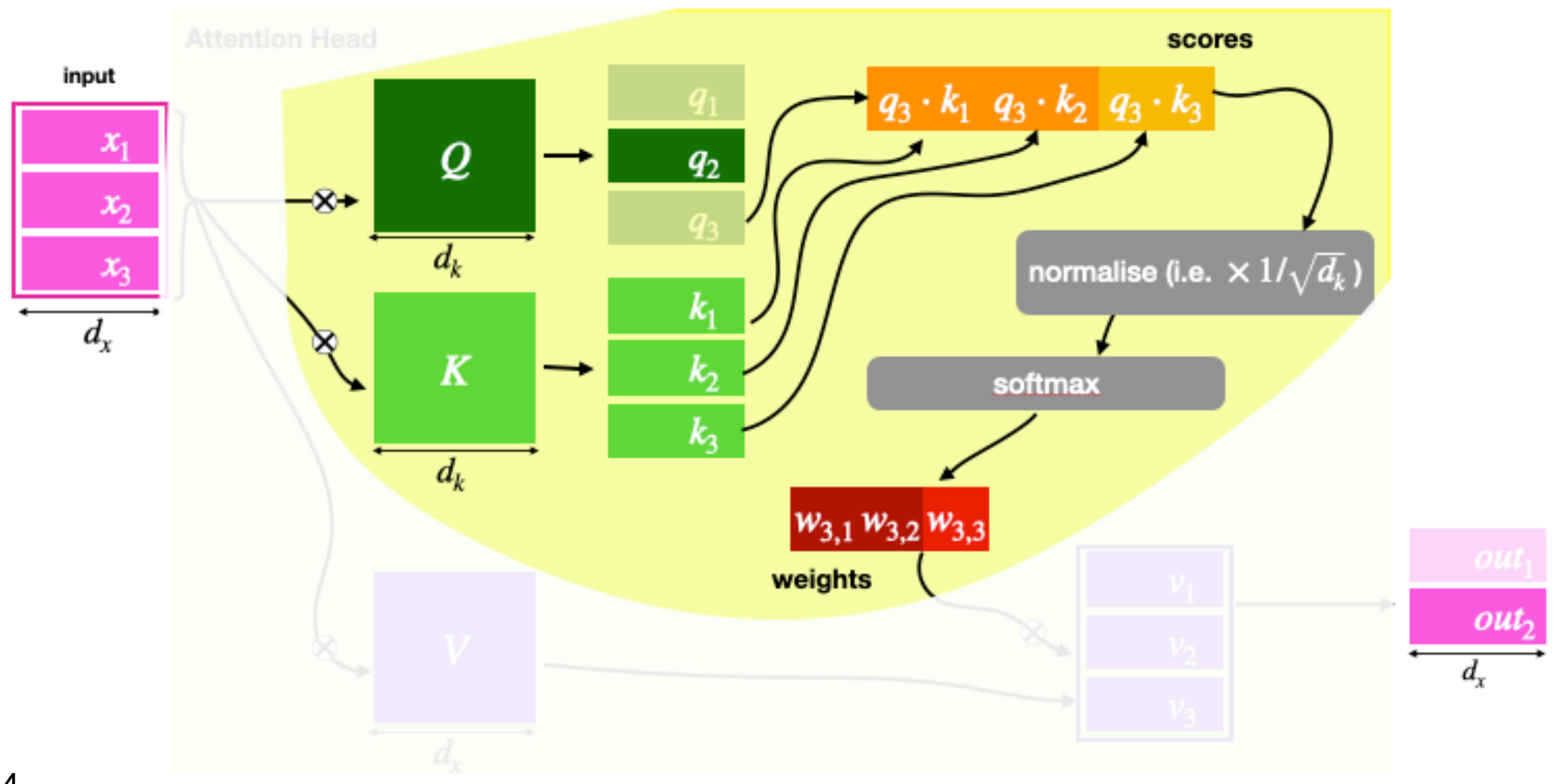
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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
$\boxed{1}$	$\boxed{F}$	$\boxed{F}$	$\boxed{F}$
2	T	F	F



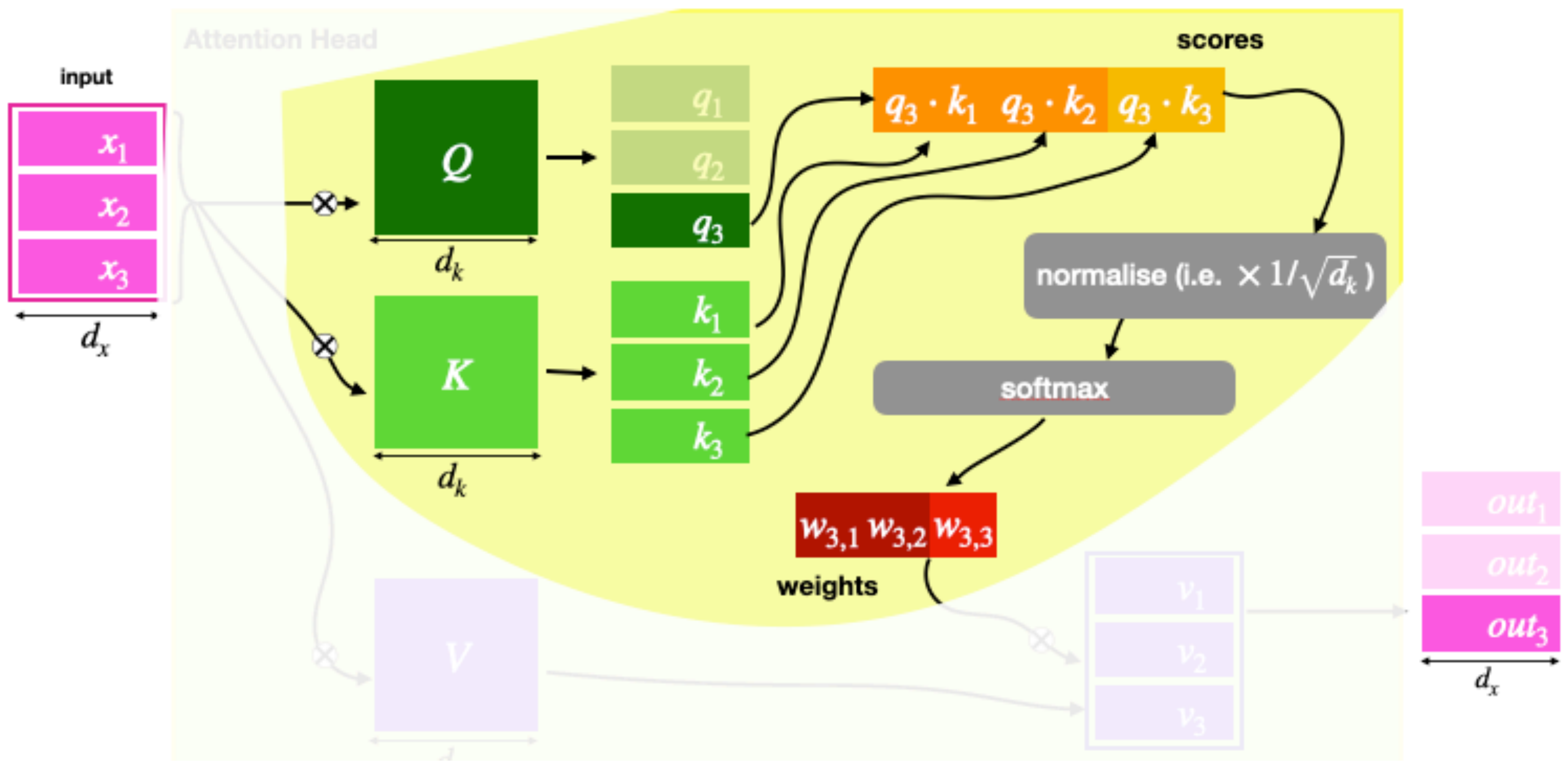
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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



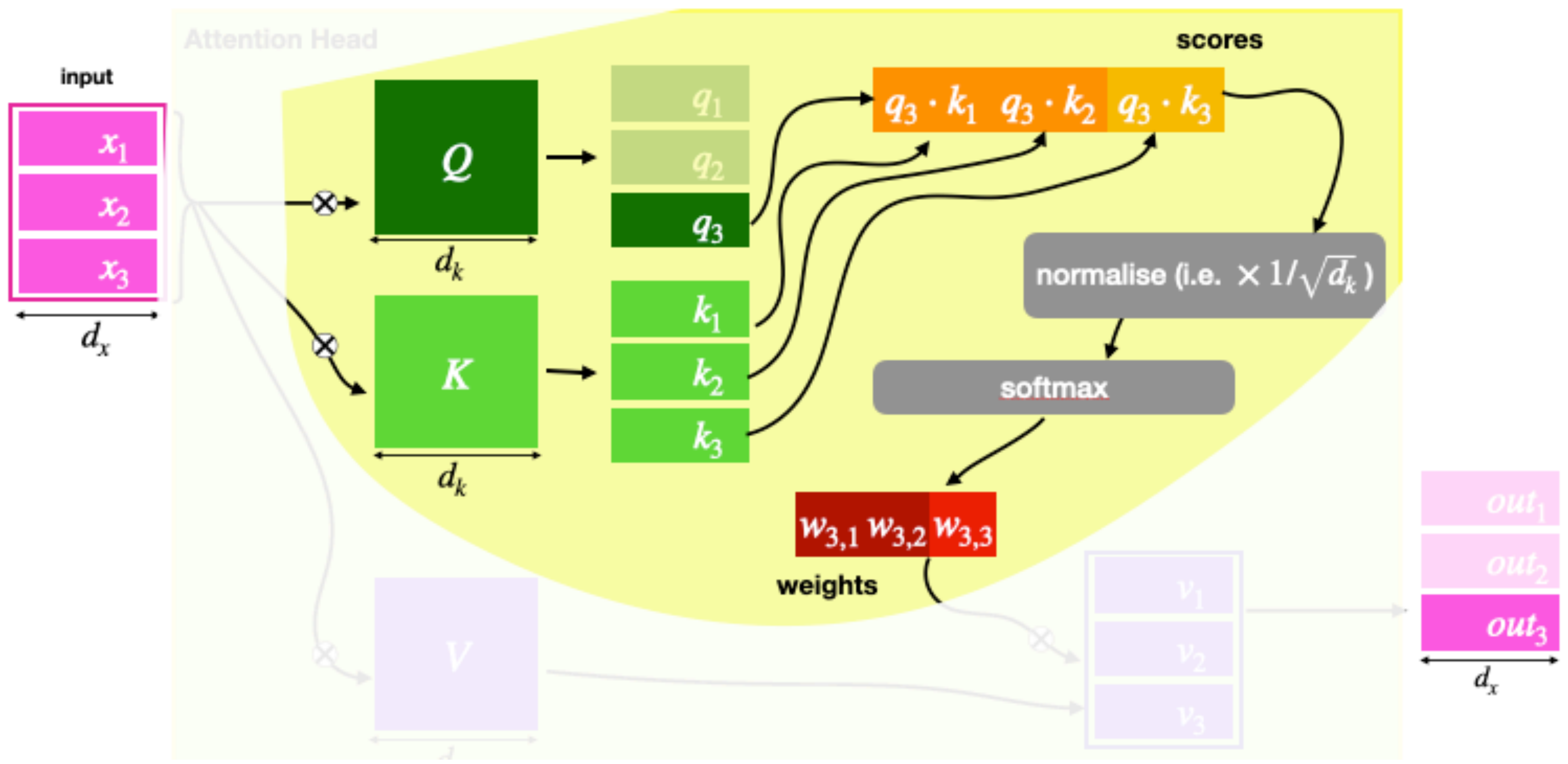
Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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],>=)

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

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Single Head: Scoring $\leftrightarrow$ Selecting

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

	0	1	2
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 ↔ Selecting

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

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

(1, 0, 0, …) k_1

(0, 1, 0, …) k_2

(0, 0, 1, …) k_3

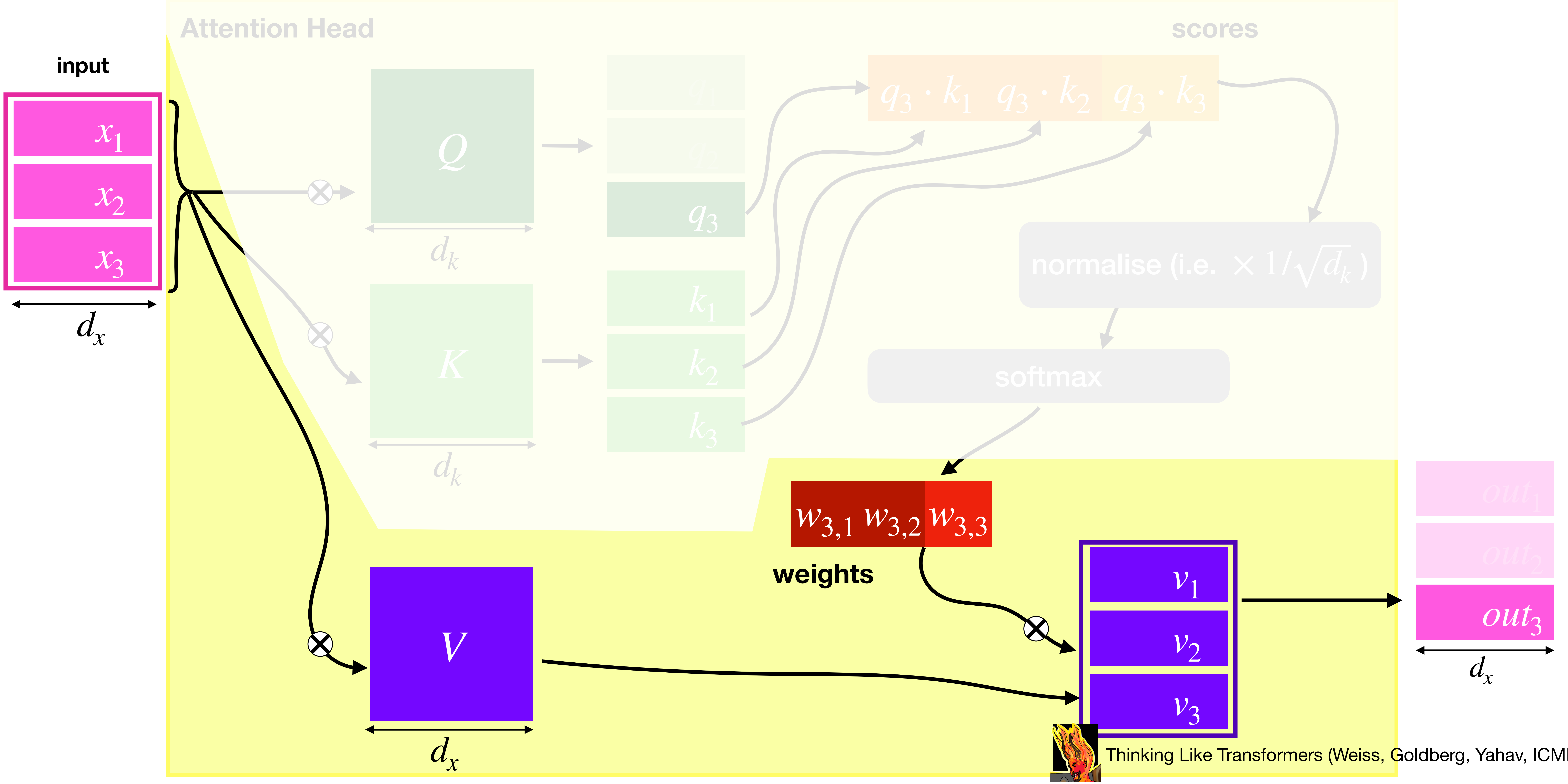
(1, 0, 0, …) q_1

(1, 1, 0, …) q_2

(1, 1, 1, …) q_3

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Single Head: Weighted Average ↔ Aggregation

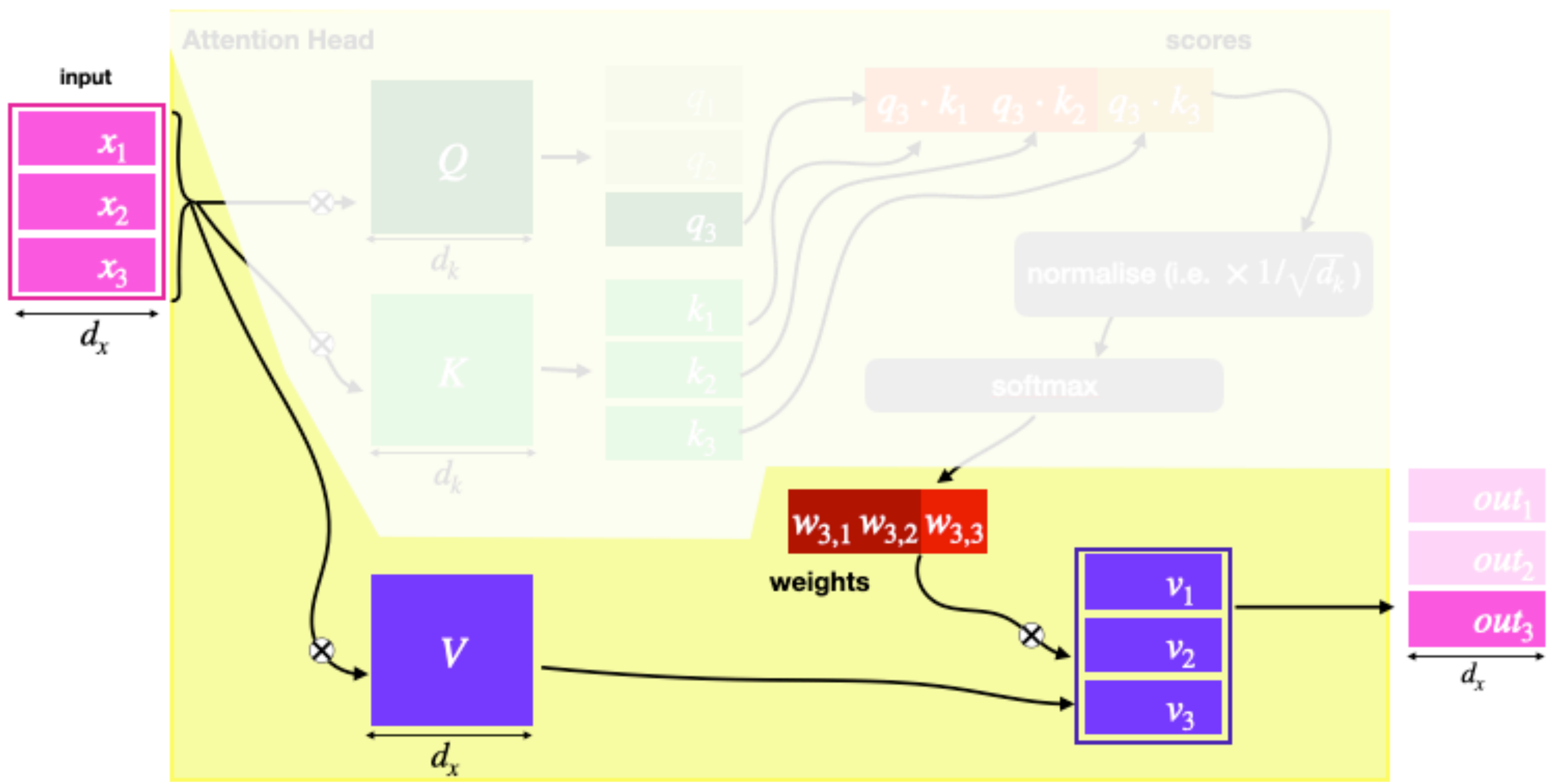


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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		

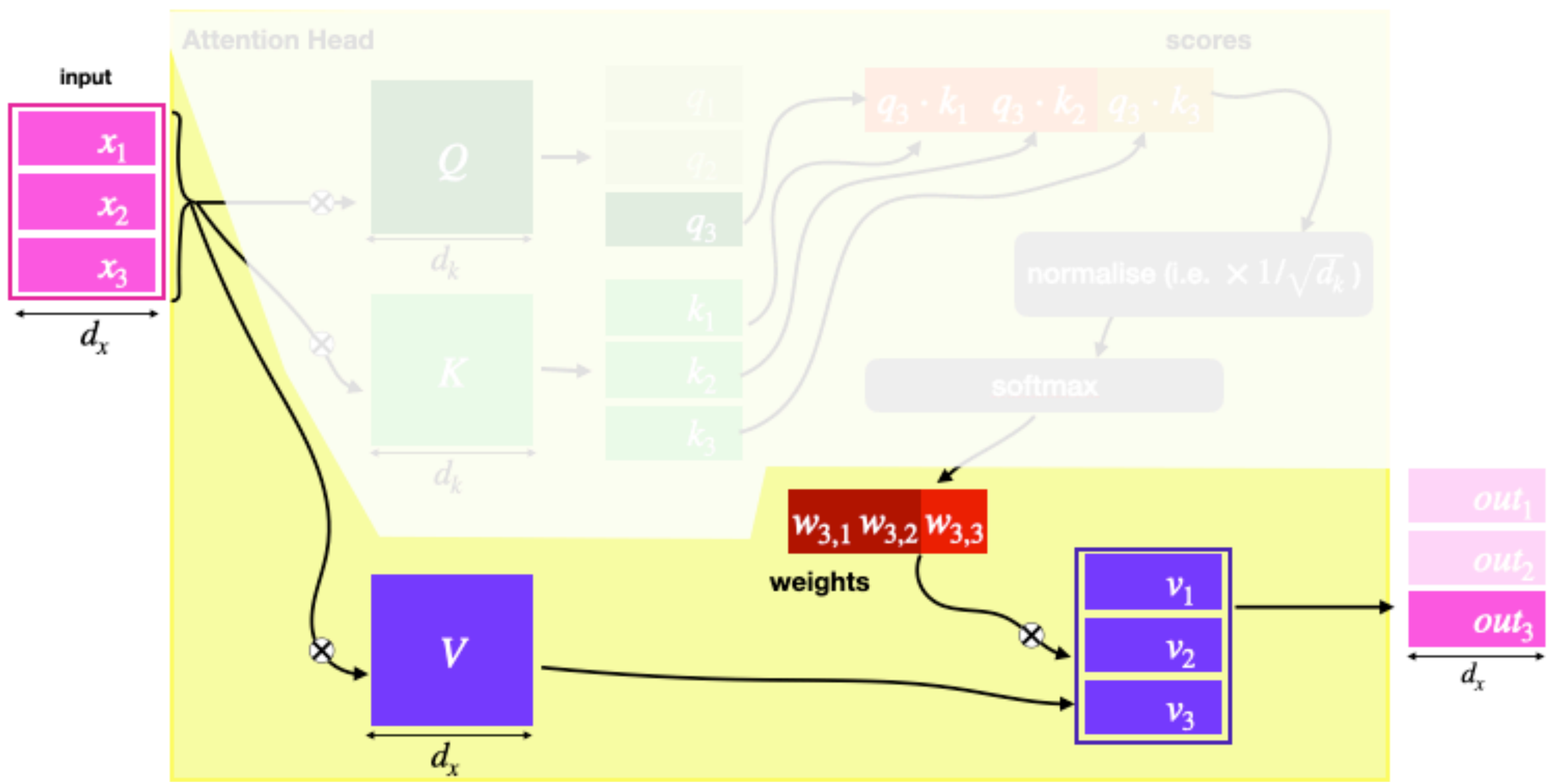


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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		

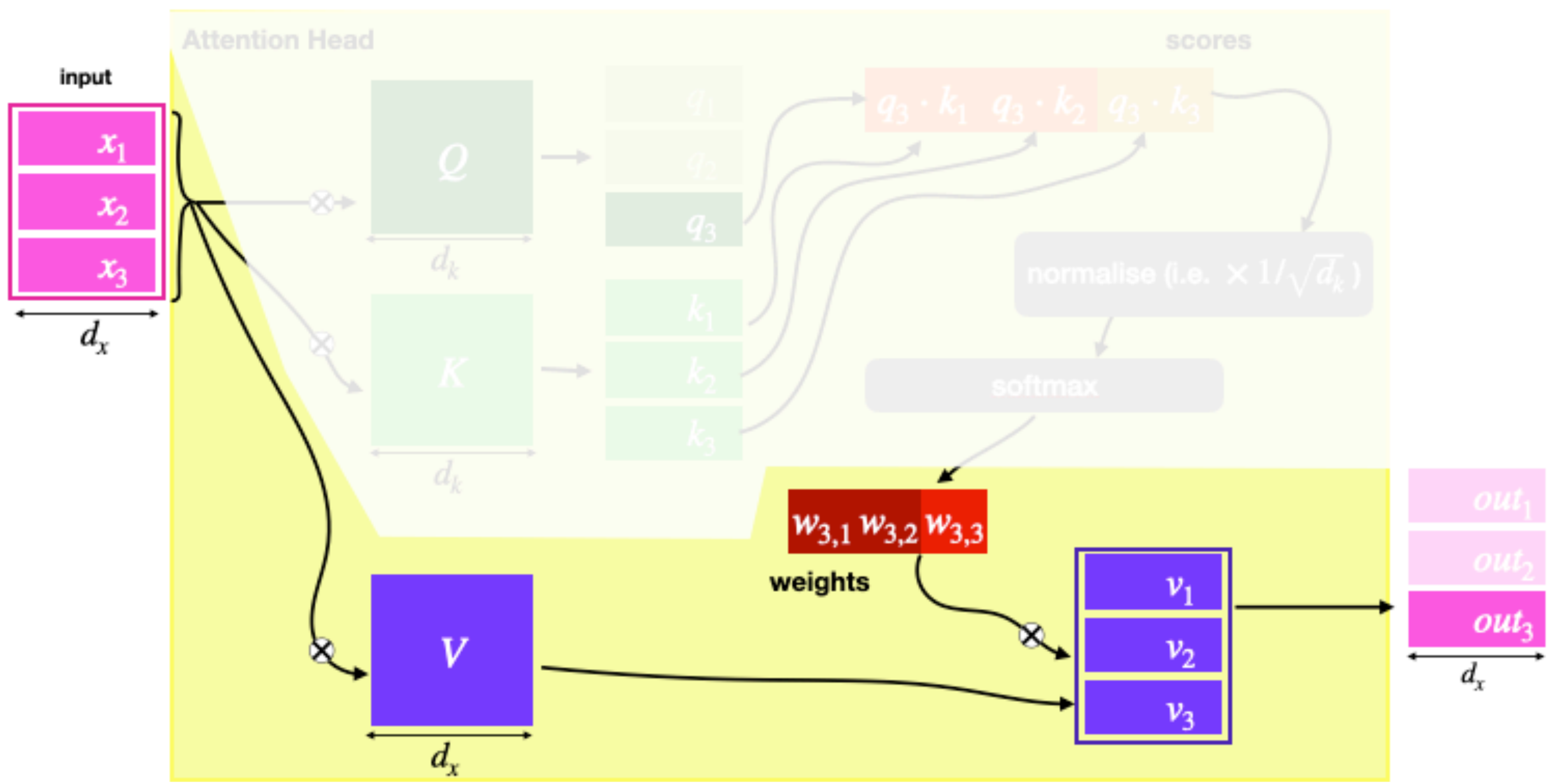


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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				

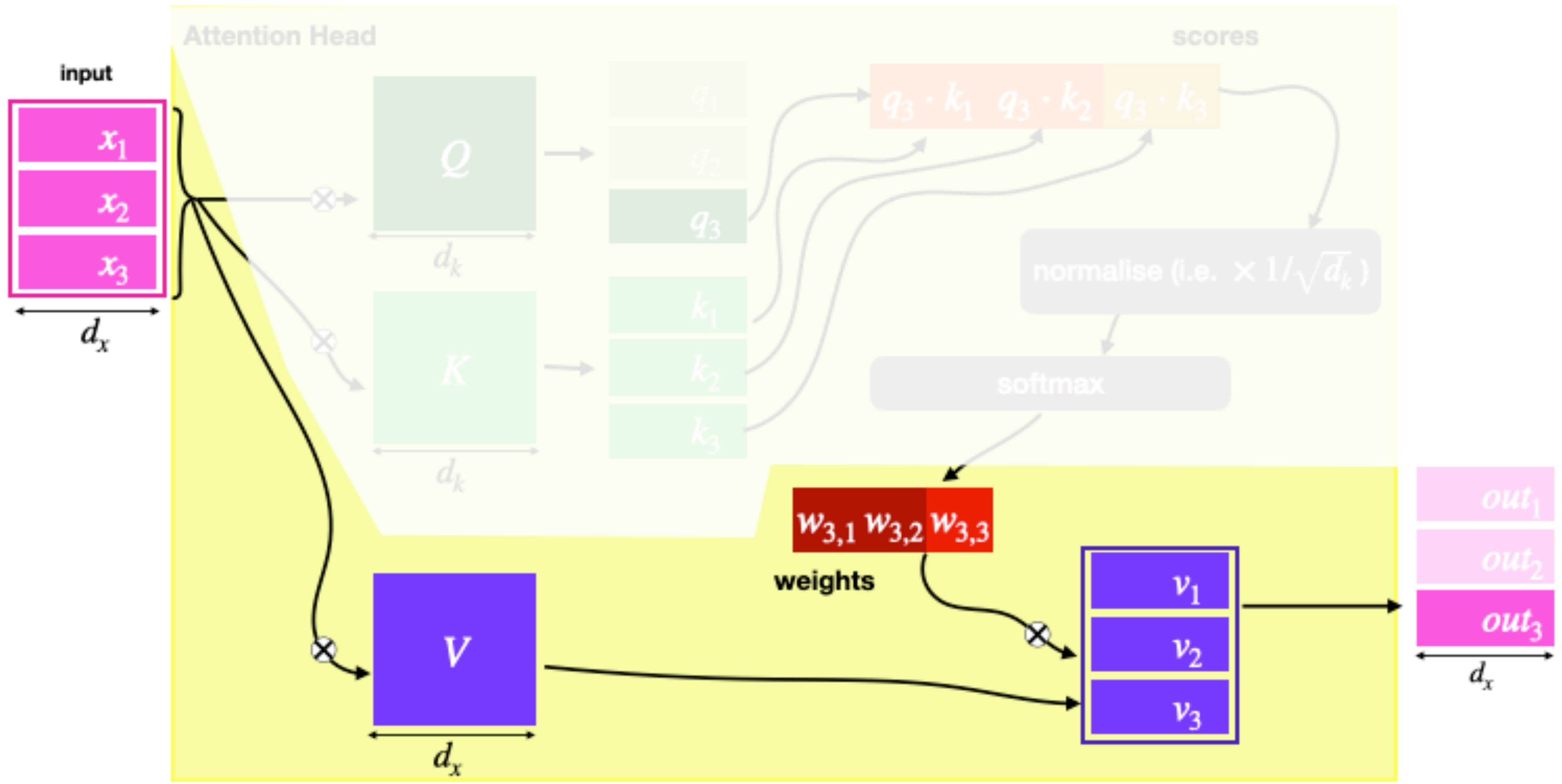


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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		

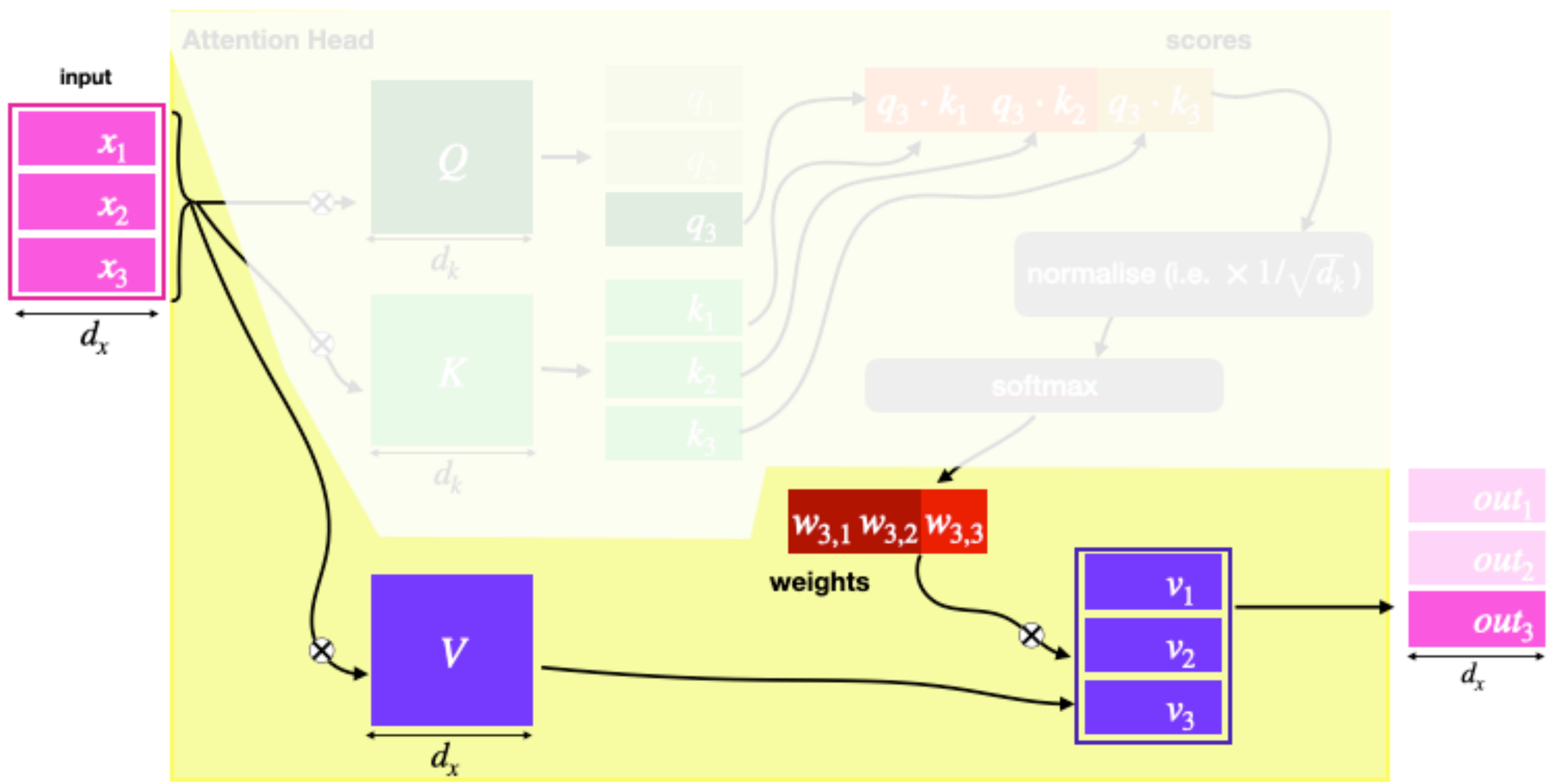


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Single Head: Weighted Average ↔ Aggregation

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

			1 2 4			
F	T	T	1 $\boxed{2\ 4}$	=>	3	
F	F	F	1 2 4	=>	0	=> [3,0,1]
T	F	F	1 2 4	=>	1	

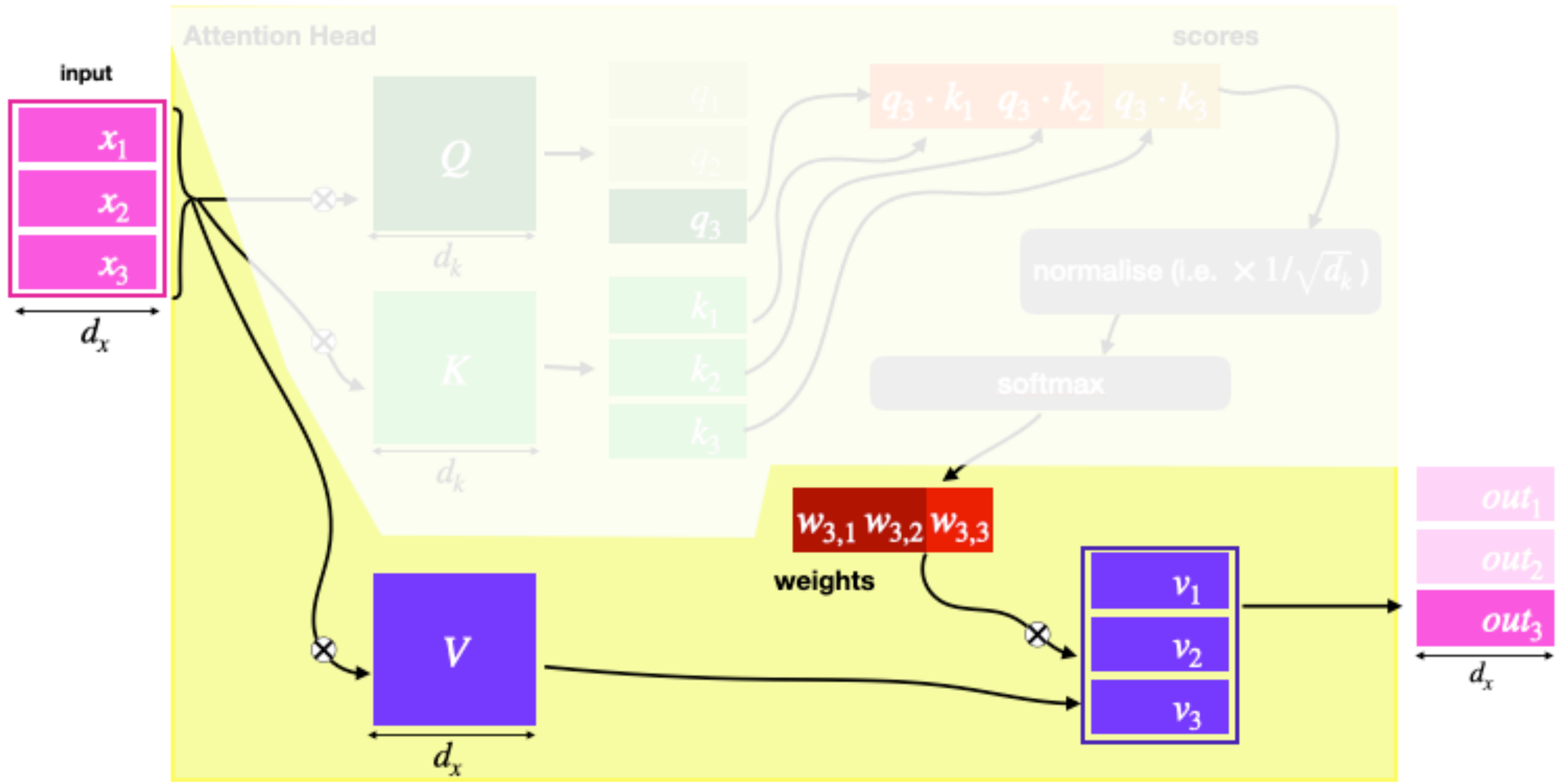


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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		

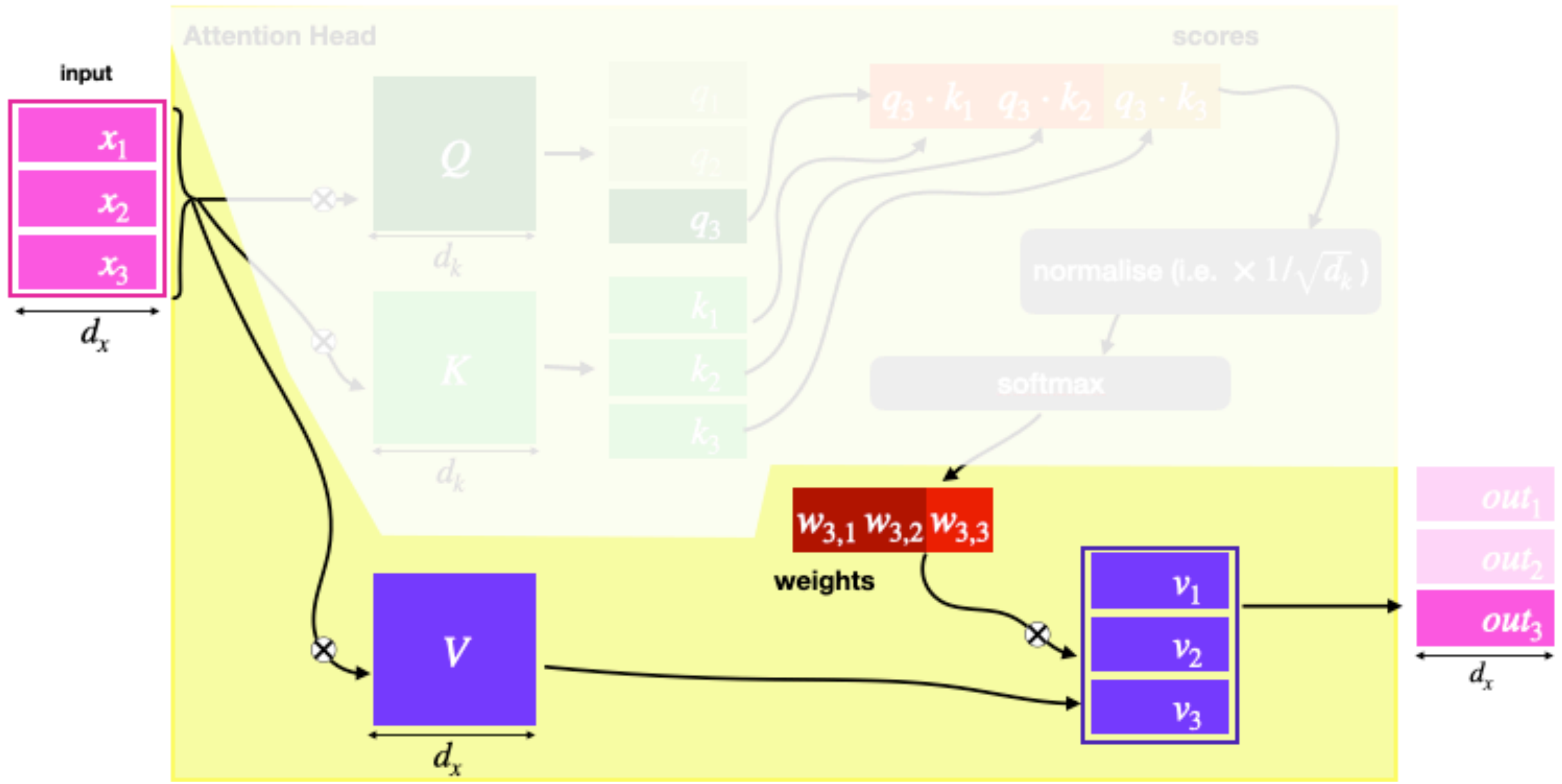


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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])

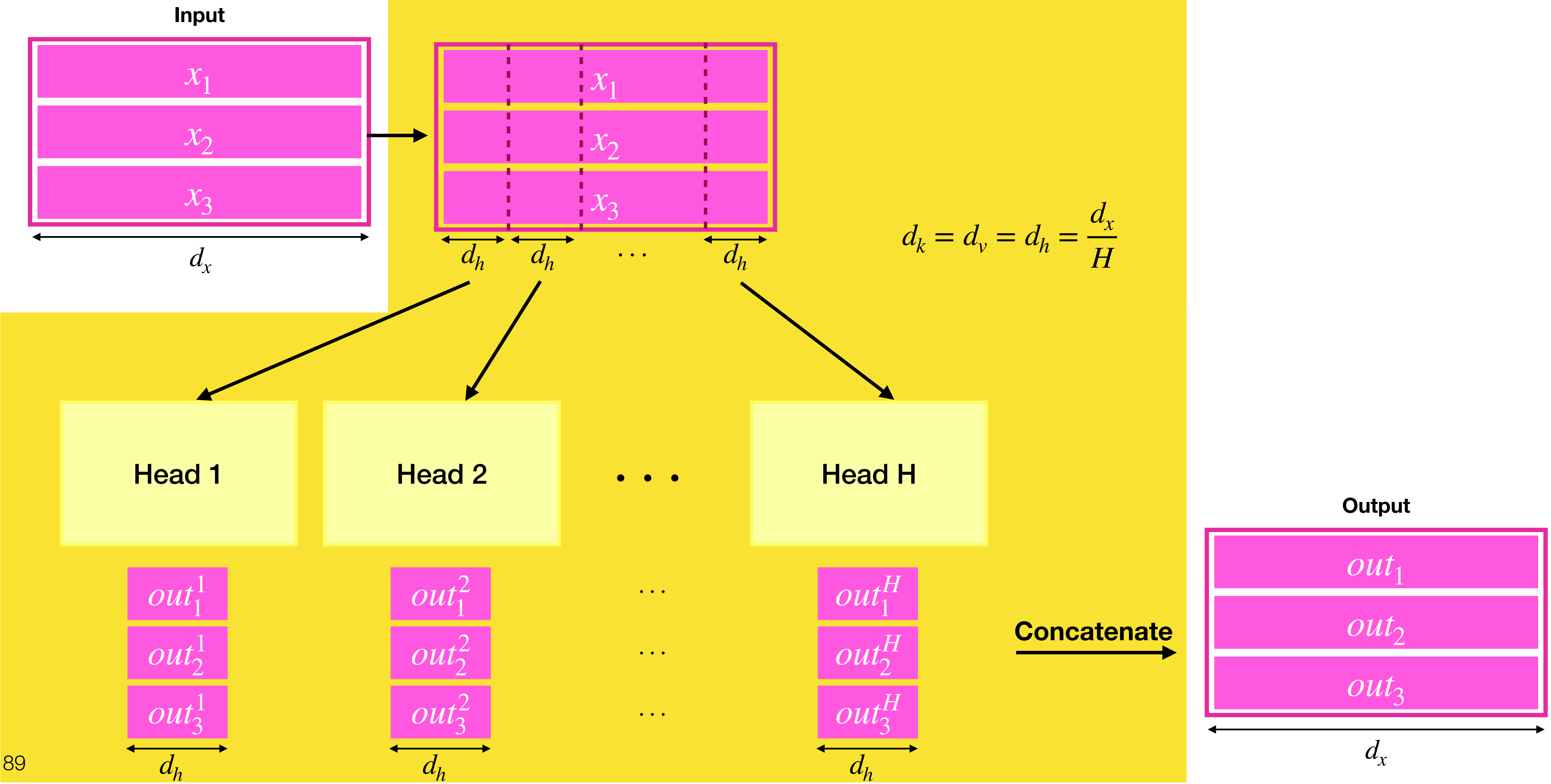
			A B C			
F	F	T	A B C	=> C		
F	T	F	A B C	=> B	=>	[C,B,A]
T	F	F	A B C	=> A		

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Great!

Now do multi-headed attention

Background - Multi-Headed Self Attention



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)

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]
```


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

RASP Extras

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)
```

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
```

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]
```

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
```

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]
```

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)

>                 indices   ):
                                        T, F]
```

Selector Compositions

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


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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)
```


Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

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)
```

Can you see how to use this trick for selector_width?

Connection to Reality?

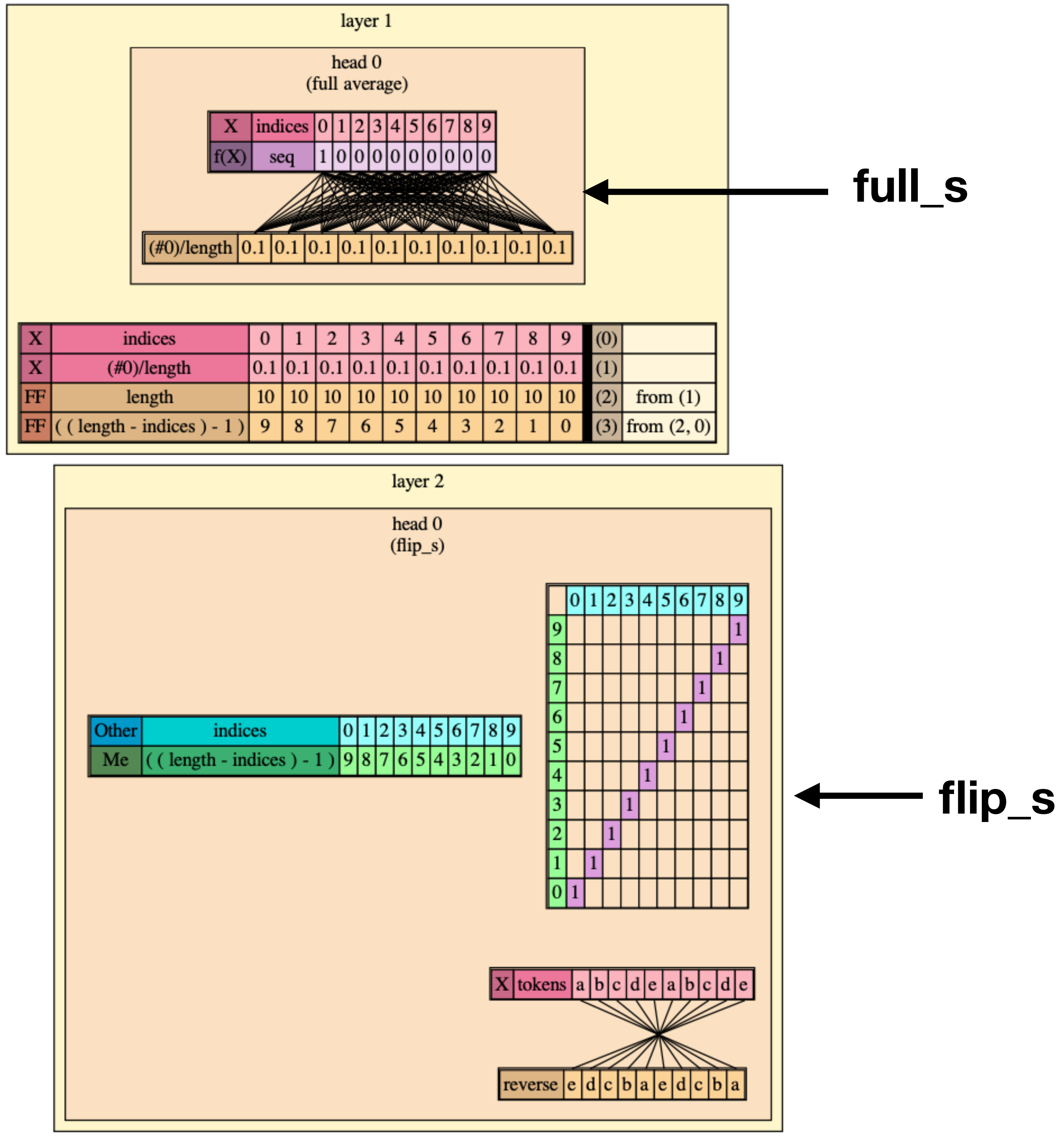
RASP expects 2 layers for arbitrary-length reverse

```
>> 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)
```

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Connection to Reality?

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

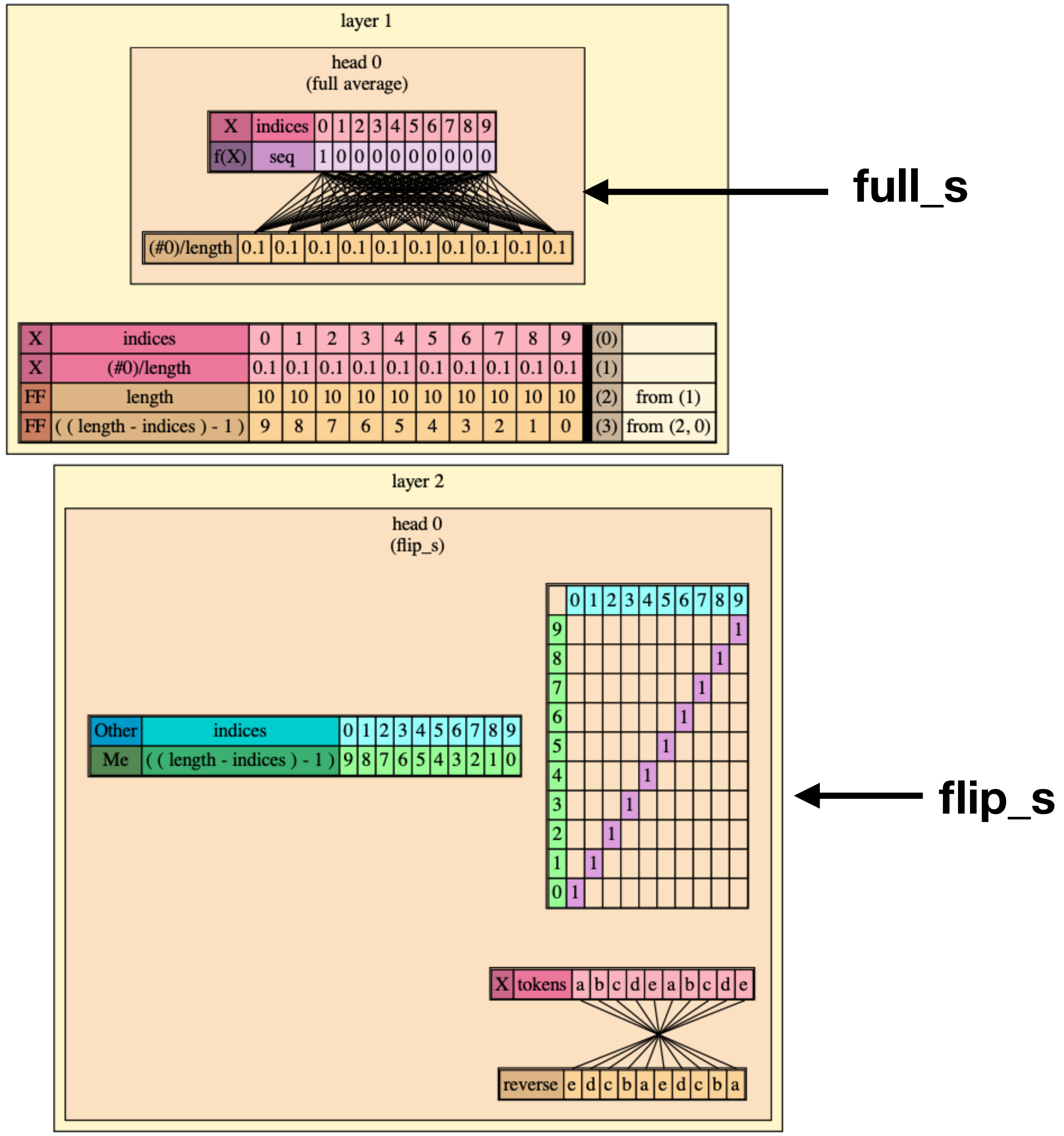


RASP expects 2 layers for arbitrary-length reverse

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Connection to Reality?

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



RASP expects 2 layers for arbitrary-length reverse

Test:
Training small transformers on lengths 0-100:

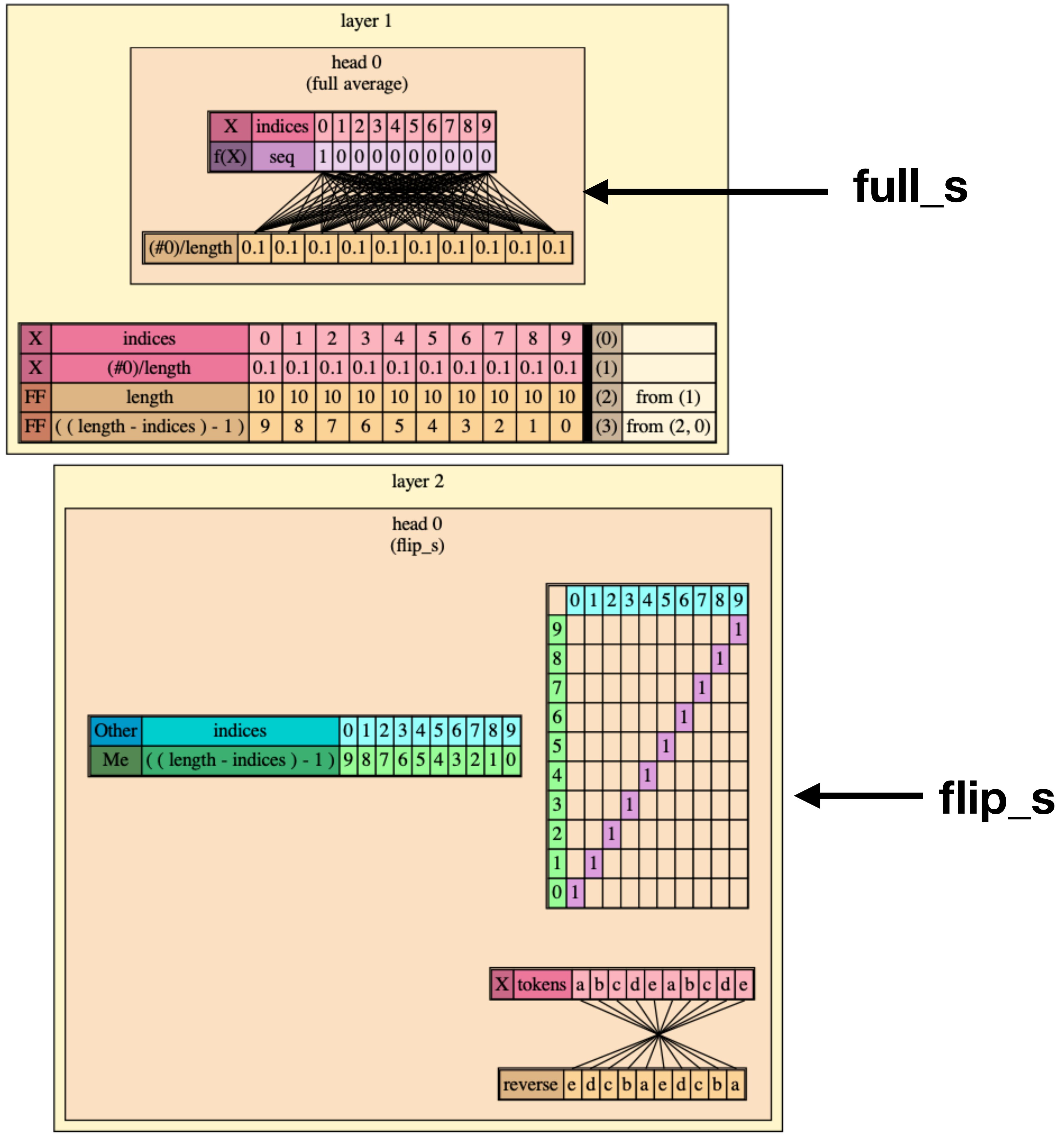
2 layers: **99.6**% accuracy after 20 epochs
1 layer: **39.6**% accuracy after 50 epochs

Even with compensation for number of heads and parameters!

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Connection to Reality?

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



RASP expects 2 layers for arbitrary-length reverse

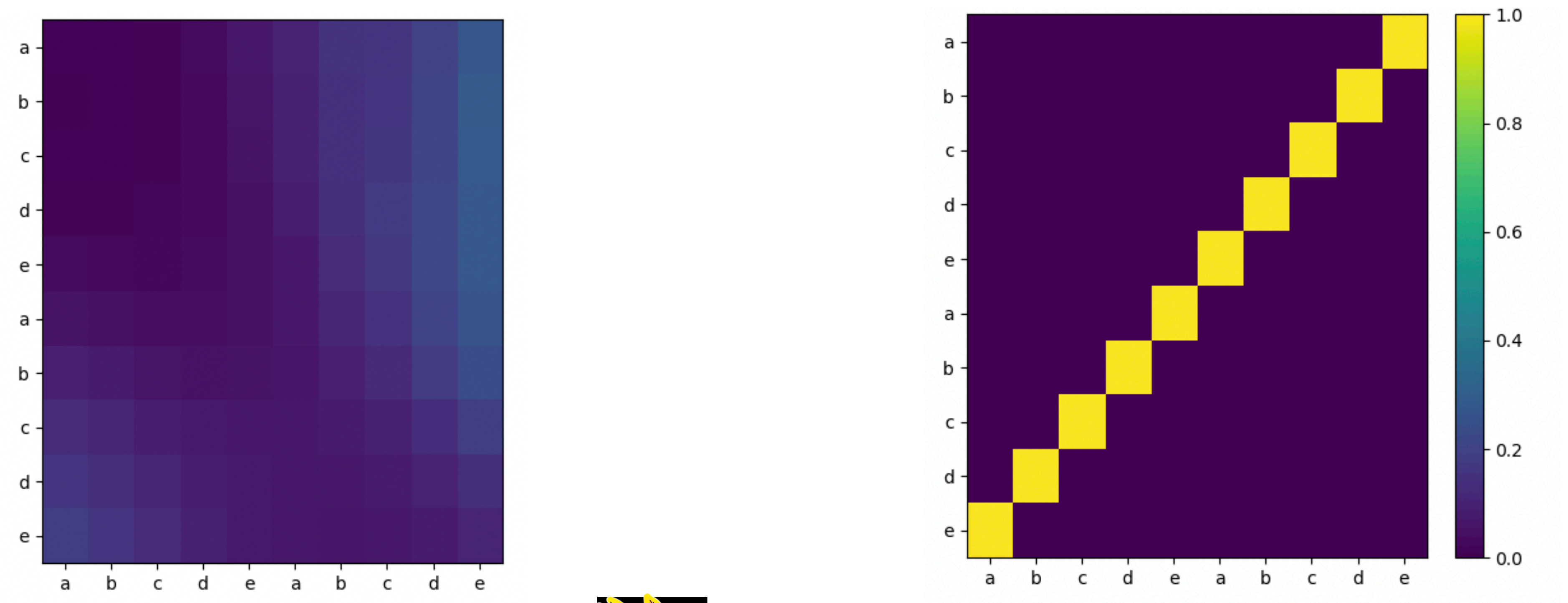
Test:
Training small transformers on lengths 0-100:

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!

Bonus: the 2 layer transformer's attention patterns:

Layer 1 (*full_s*) Layer 2 (*flip_s*)



Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Connection to Reality?

Example 2: *histogram* (assuming BOS)

in place histogram,
with BOS - examples:

[§,a,a,a,b] -> [0,3,3,3,1]
[§,a,b,a,c] -> [0,2,1,2,1]
[§,a,b,c,c] -> [0,1,1,2,2]

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)
```

**in place histogram,
with BOS - examples:**

[§,a,a,a,b] -> [0,3,3,3,1]
[§,a,b,a,c] -> [0,2,1,2,1]
[§,a,b,c,c] -> [0,1,1,2,2]

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)

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Connection to Reality?

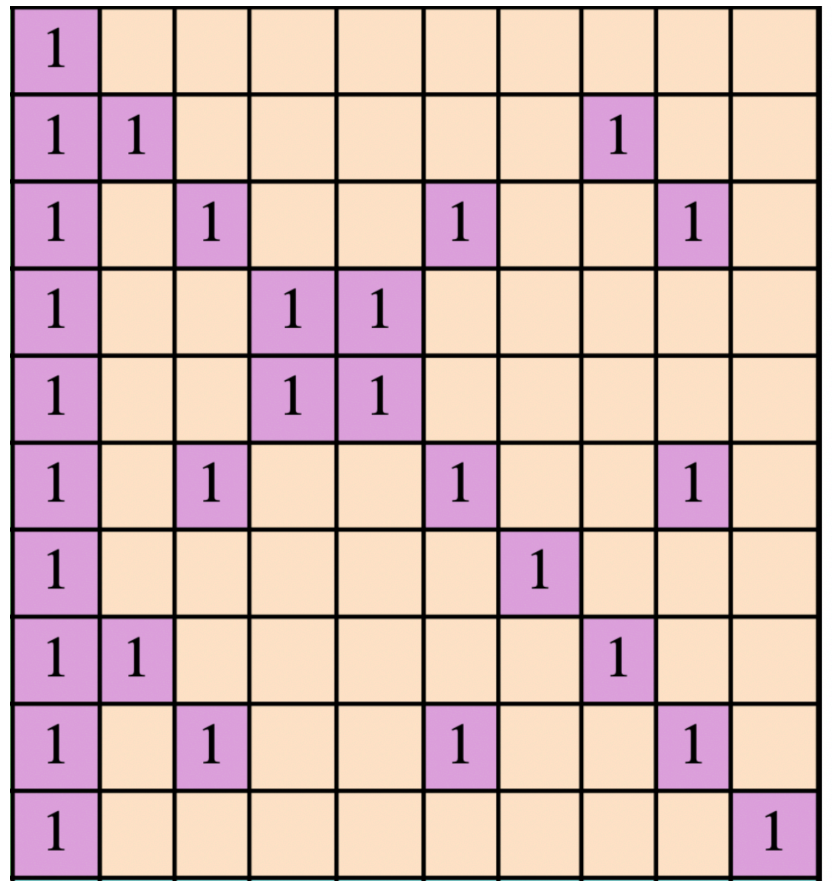
Example 2: *histogram* (assuming BOS)

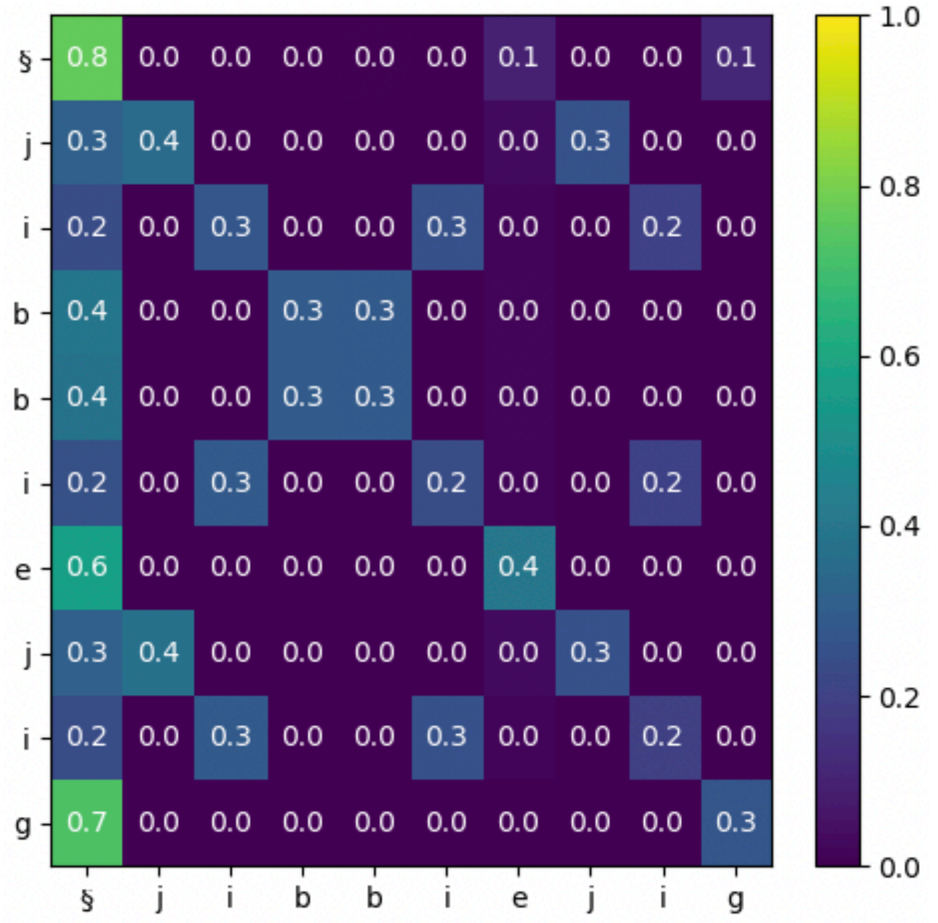
```
>> examples off
>> same_or_0 = select(tokens,tokens,==) or select(indices,0,==);
     selector: same_or_0
>> frac_with_0 = aggregate(same_or_0,indicator(indices==0));
     s-op: frac_with_0
>> histogram_assuming_bos = round(1/frac_with_0)-1;
     s-op: histogram_assuming_bos
>> histogram_assuming_bos("§hello");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





Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

Connection to Reality?

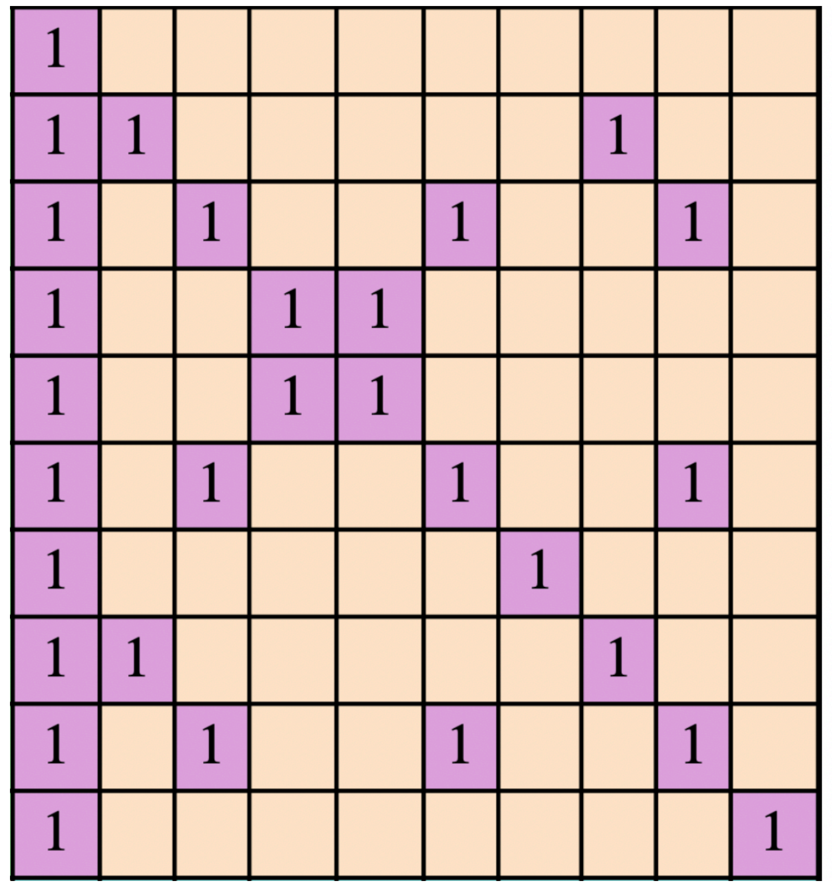
Example 2: *histogram* (assuming BOS)

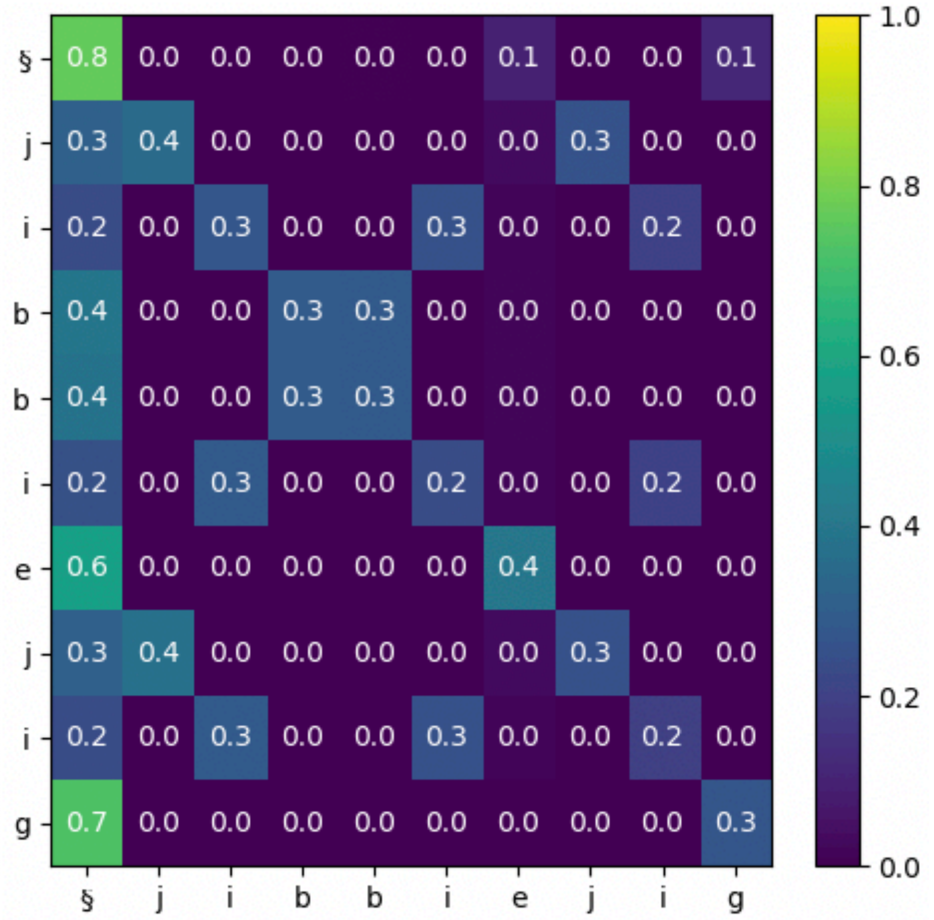
```
>> examples off
>> same_or_0 = select(tokens,tokens,==) or select(indices,0,==);
     selector: same_or_0
>> frac_with_0 = aggregate(same_or_0,indicator(indices==0));
     s-op: frac_with_0
>> histogram_assuming_bos = round(1/frac_with_0)-1;
     s-op: histogram_assuming_bos
>> histogram_assuming_bos("§hello");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





Try it out!

github.com/tech-srl/RASP

Thinking Like Transformers (Weiss, Goldberg, Yahav, ICML 2021)

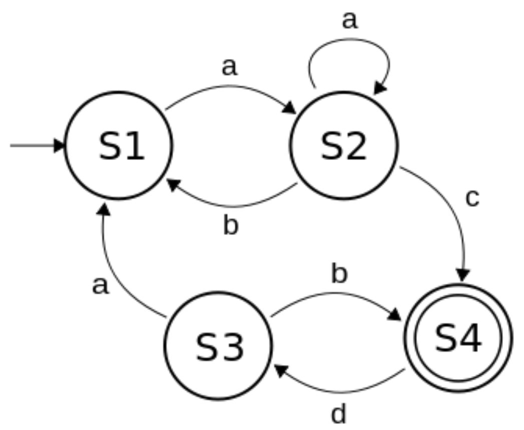
Neural Sequence Models:
a Formal Lens

Counting

LSTMs are counter machines, GRUs aren't **(ACL 2018)**

RASP

Finding a formalism to describe transformers **(ICML 2021)**

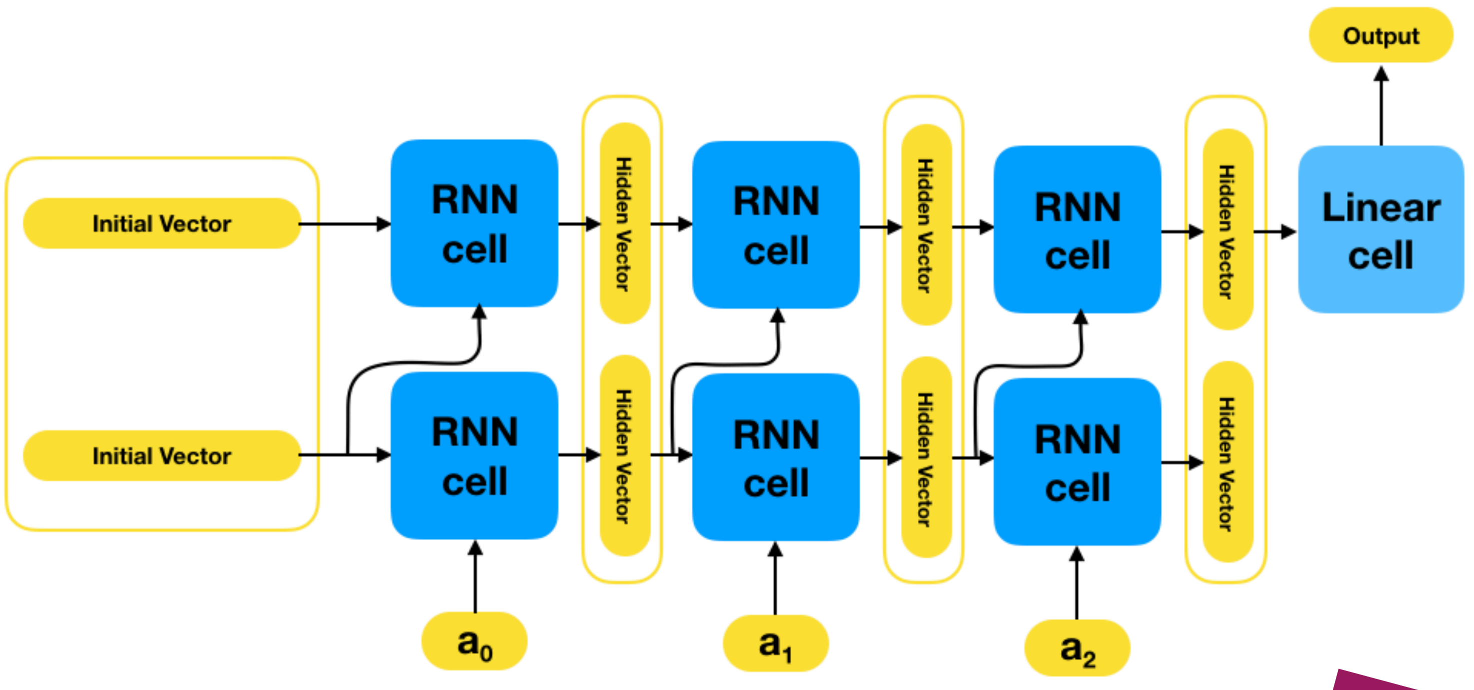


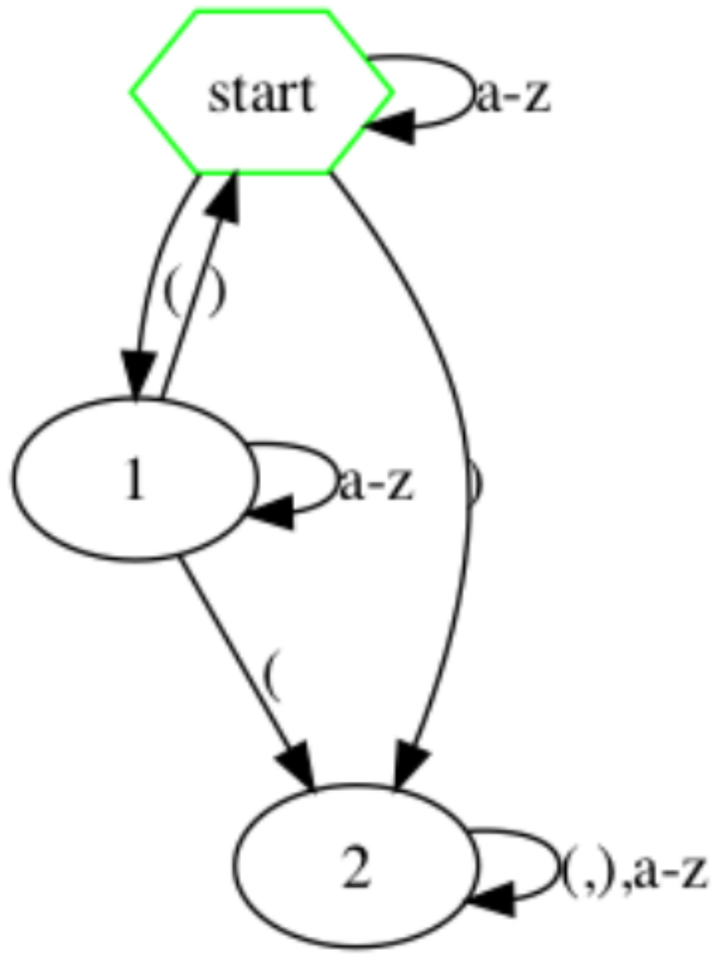
DFAs from RNNs

Applying L* to learn DFAs from RNNs **(ICML 2018)**

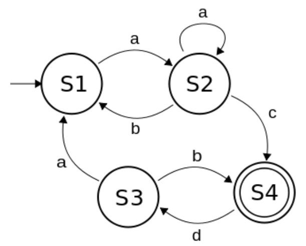
+ using the result for CFGs **(TACAS 2021)**

DFAs from RNNs





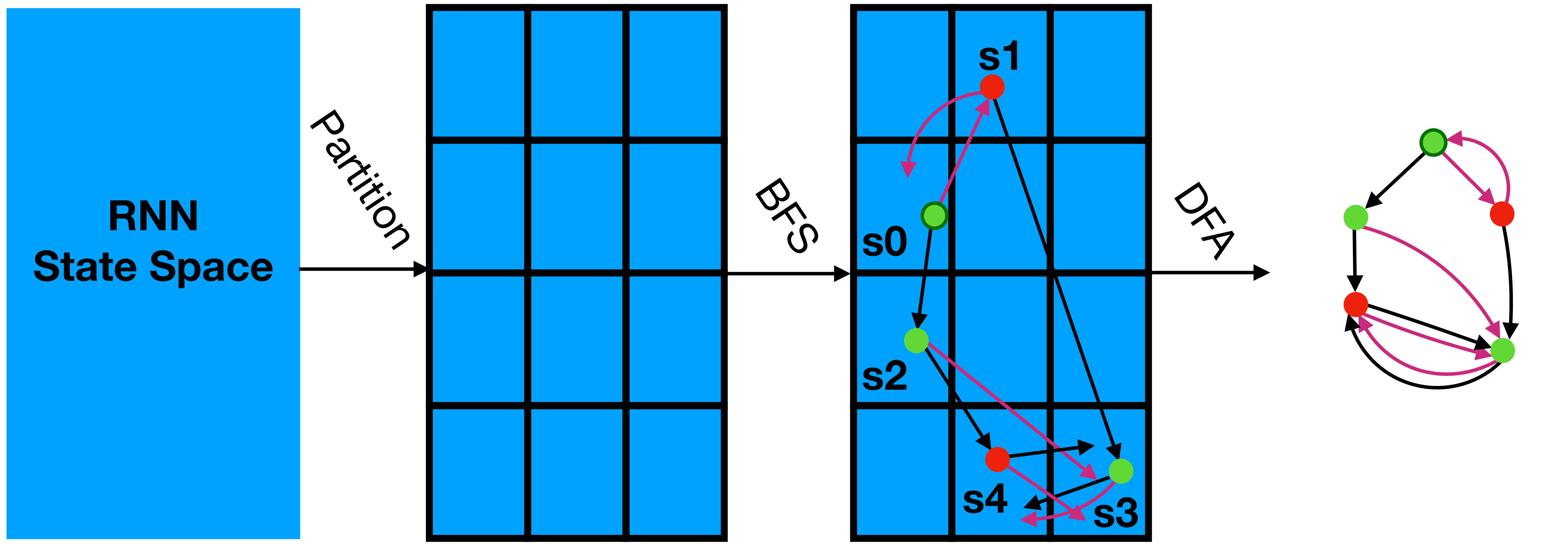
Goal:
Concise (Meaningful) Model
from Trained RNN



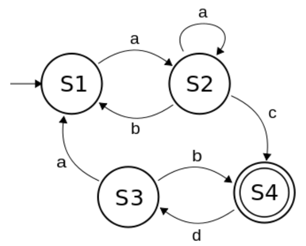
Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

Previous Approaches

1. **Partition** RNN state space
2. Explore using **pruned BFS** or transition sampling



e.g.: Omlin and Giles (1996), Cechin et al. (2003)

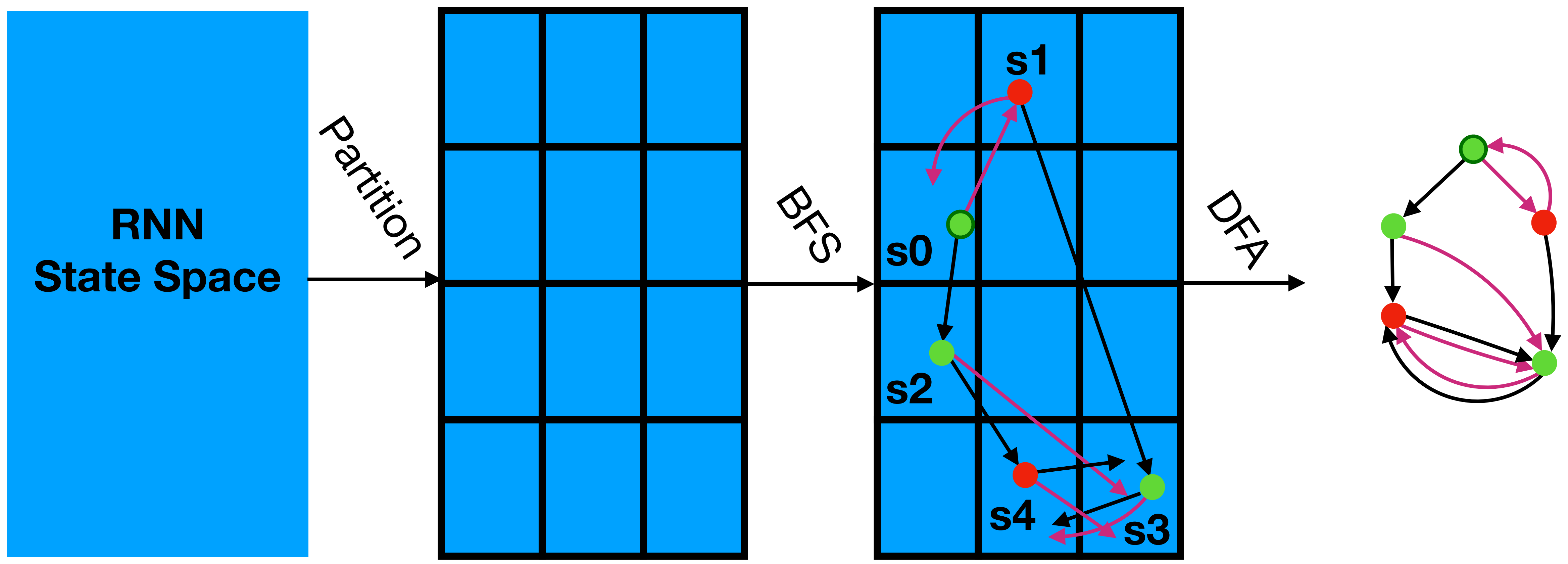


Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

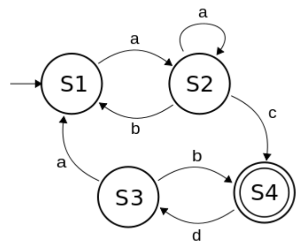
Previous Approaches

1. **Too coarse:** not representative
2. **Too fine:** very large: slow & memory consuming extraction

Impractical!



e.g.: Omlin and Giles (1996), Cechin et al. (2003)



Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

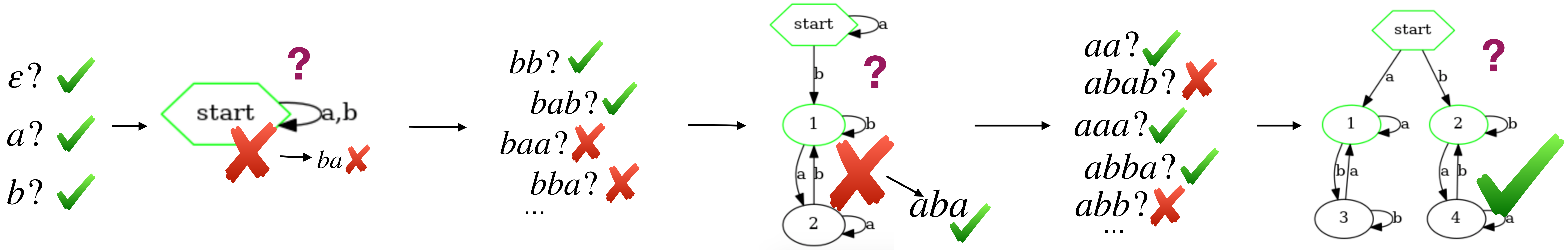
L* (Angluin, 1987)

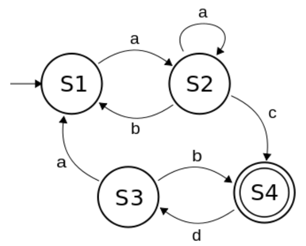
An exact learning algorithm for DFAs

Learns using:

- **Membership Queries** (request to label input sequence) and
- **Equivalence Queries** (request to accept/reject DFA)

Creates hypothesis DFA and improves it until accepted by teacher





Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

Iterative Approach

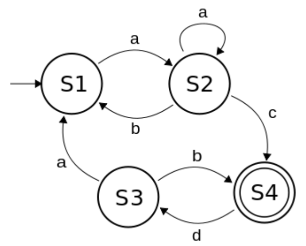
Apply L* to RNN:

Membership queries are trivial

(**Equivalence queries** are hard)

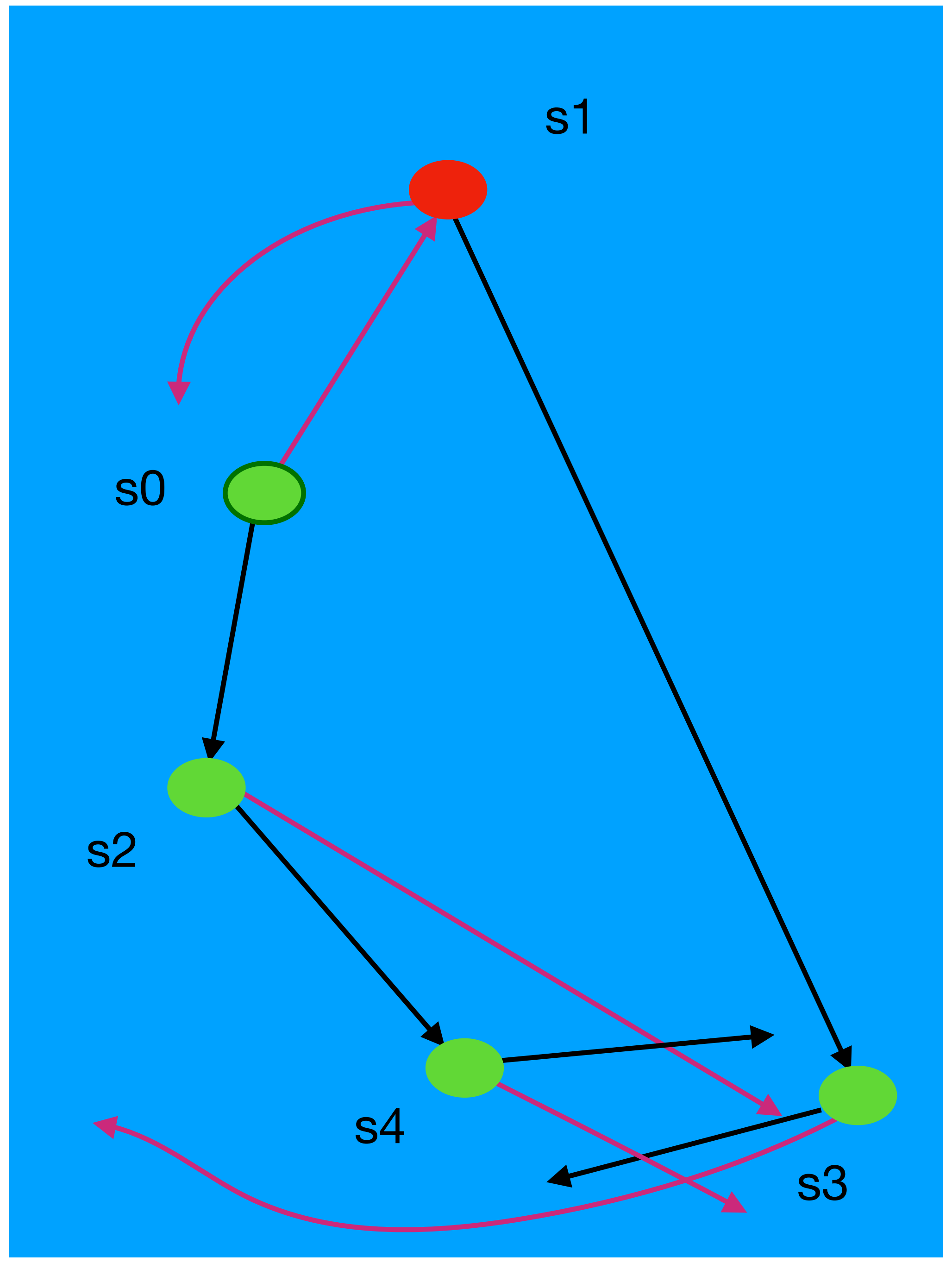
Use **equivalence queries** to induce the **partitioning** of the RNN state space

Use the **partitioning** to answer the **equivalence queries**

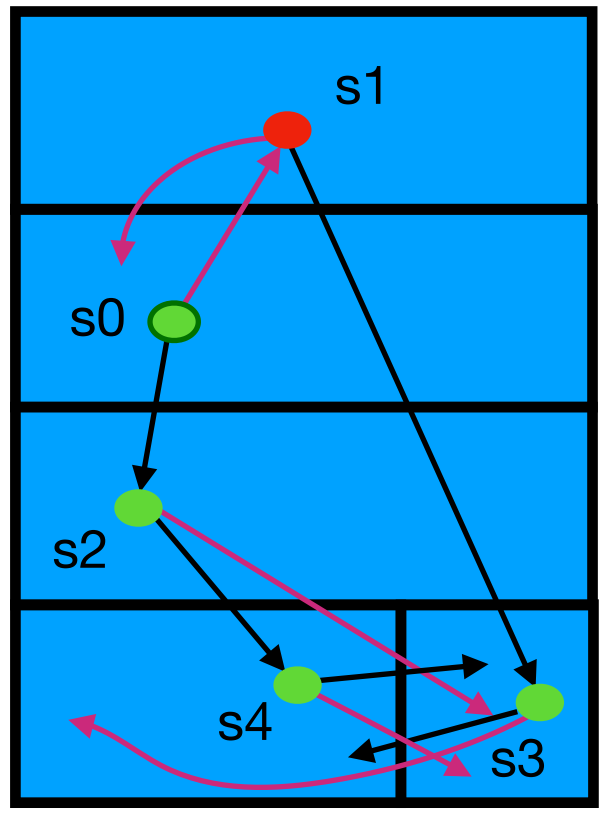


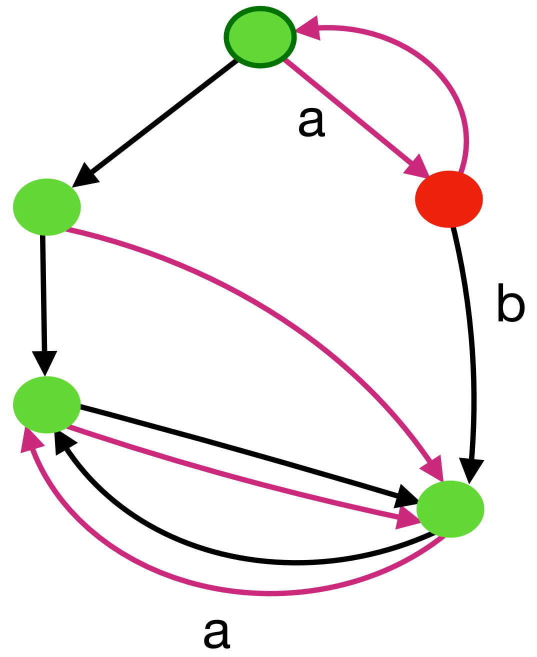
Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

Iterative Approach



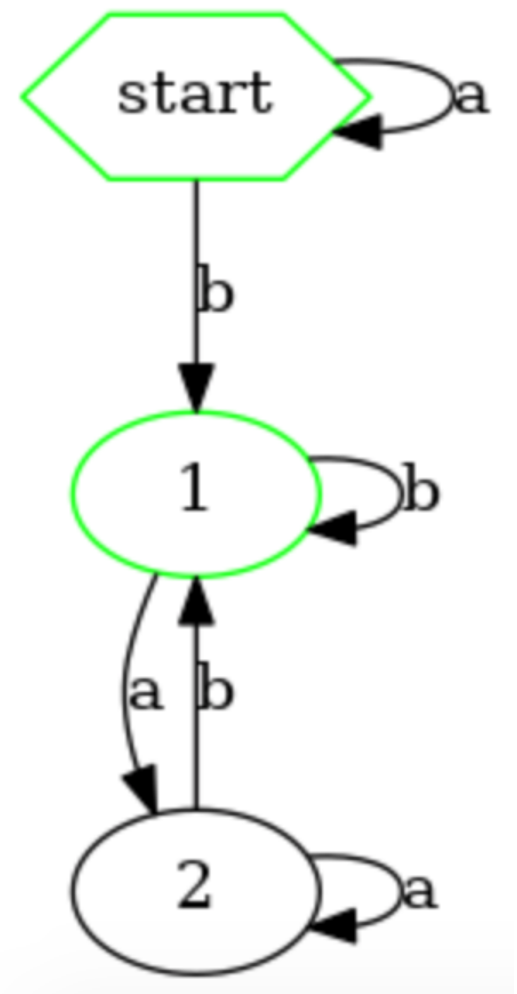
Partitioning



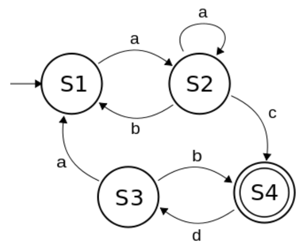


L*

ε? ✓ bb? ✓
a? ✓ bab? ✓
b? ✓ baa? ✗
bba? ✗
...

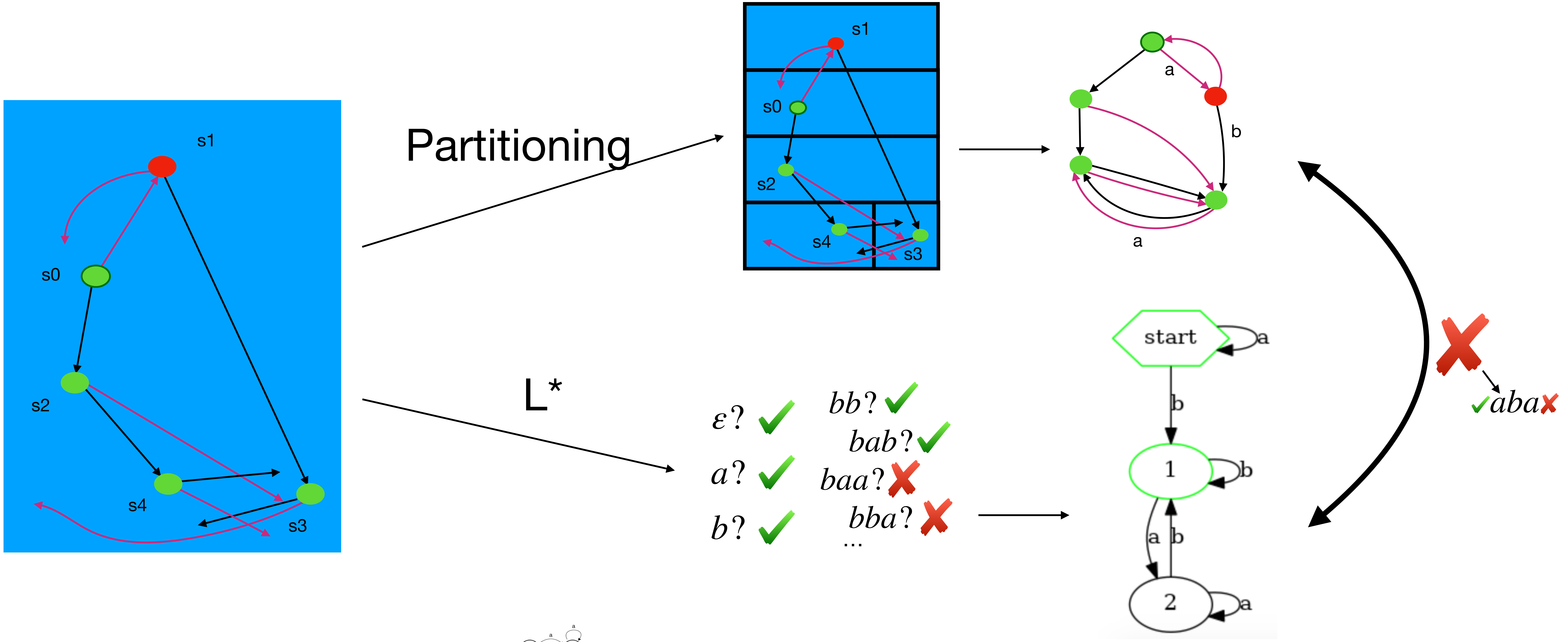


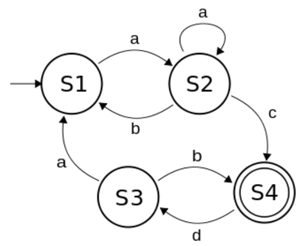
?



Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

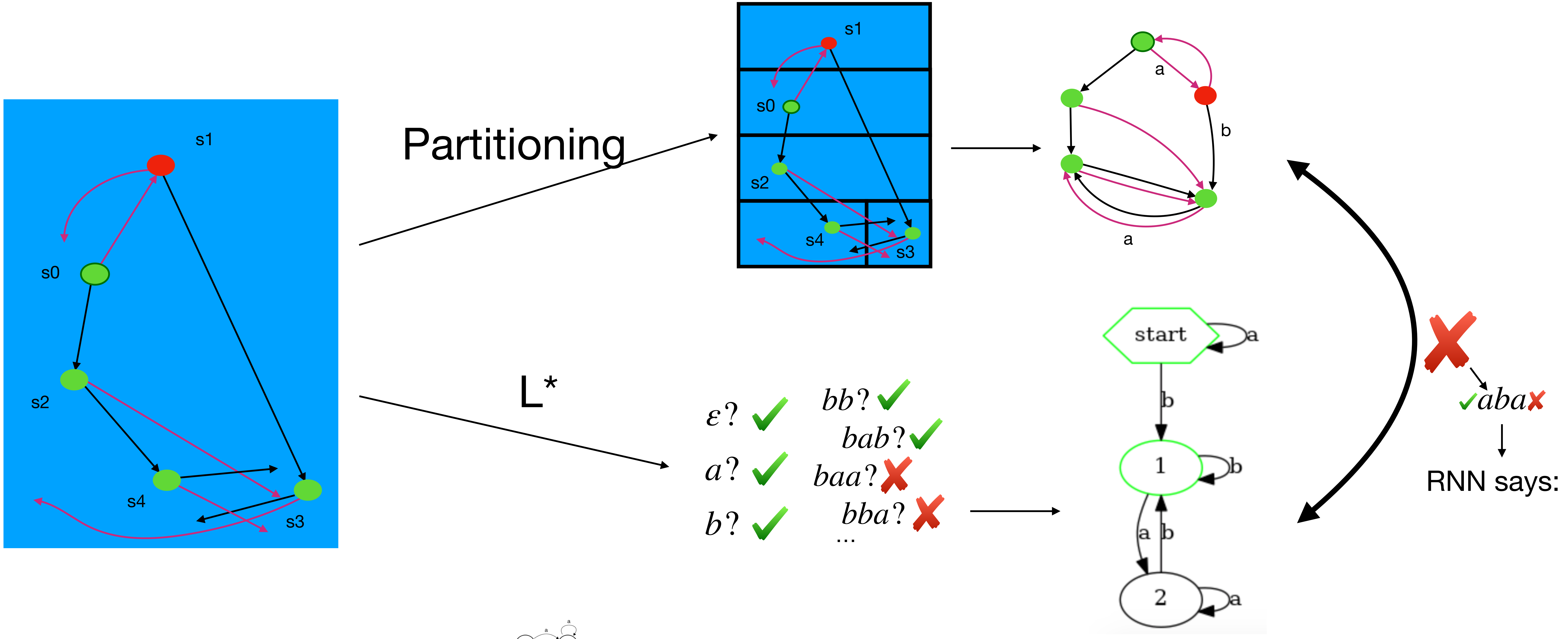
Iterative Approach

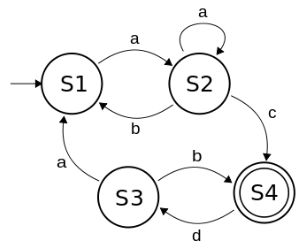




Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

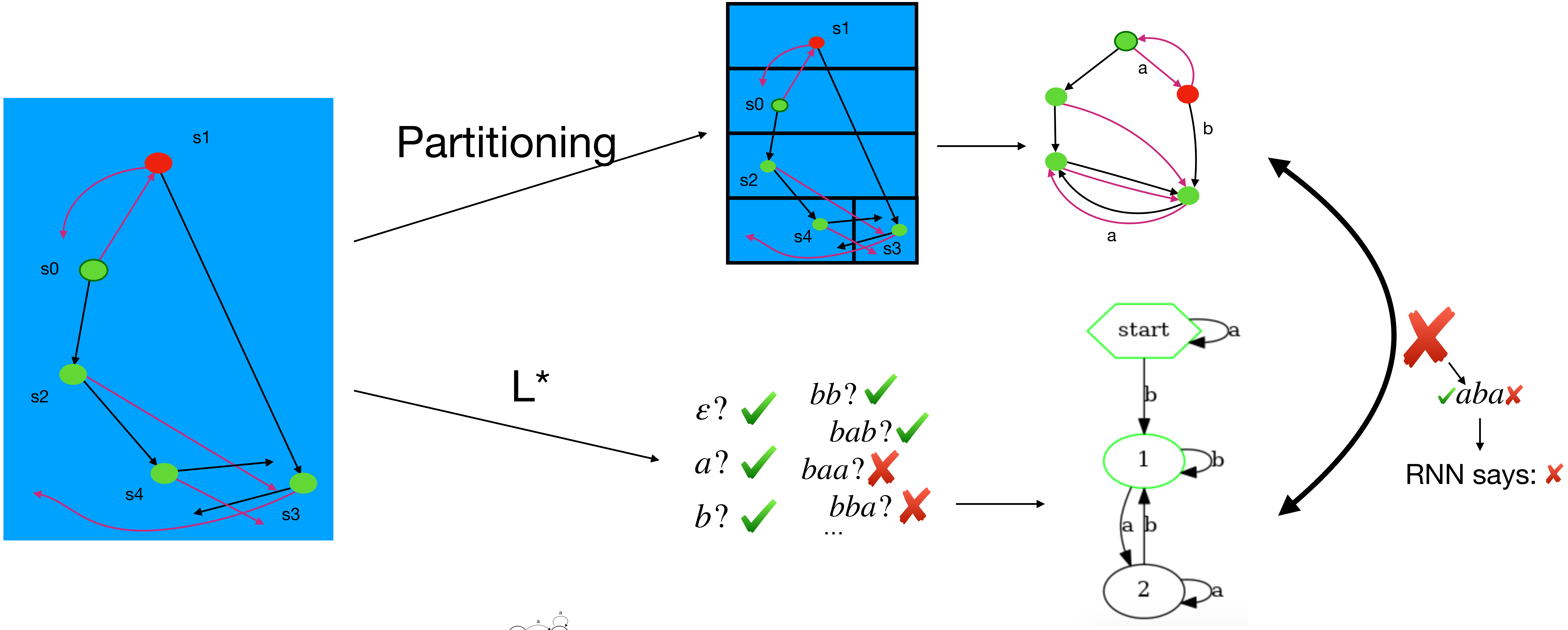
Iterative Approach

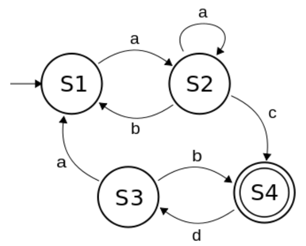




Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

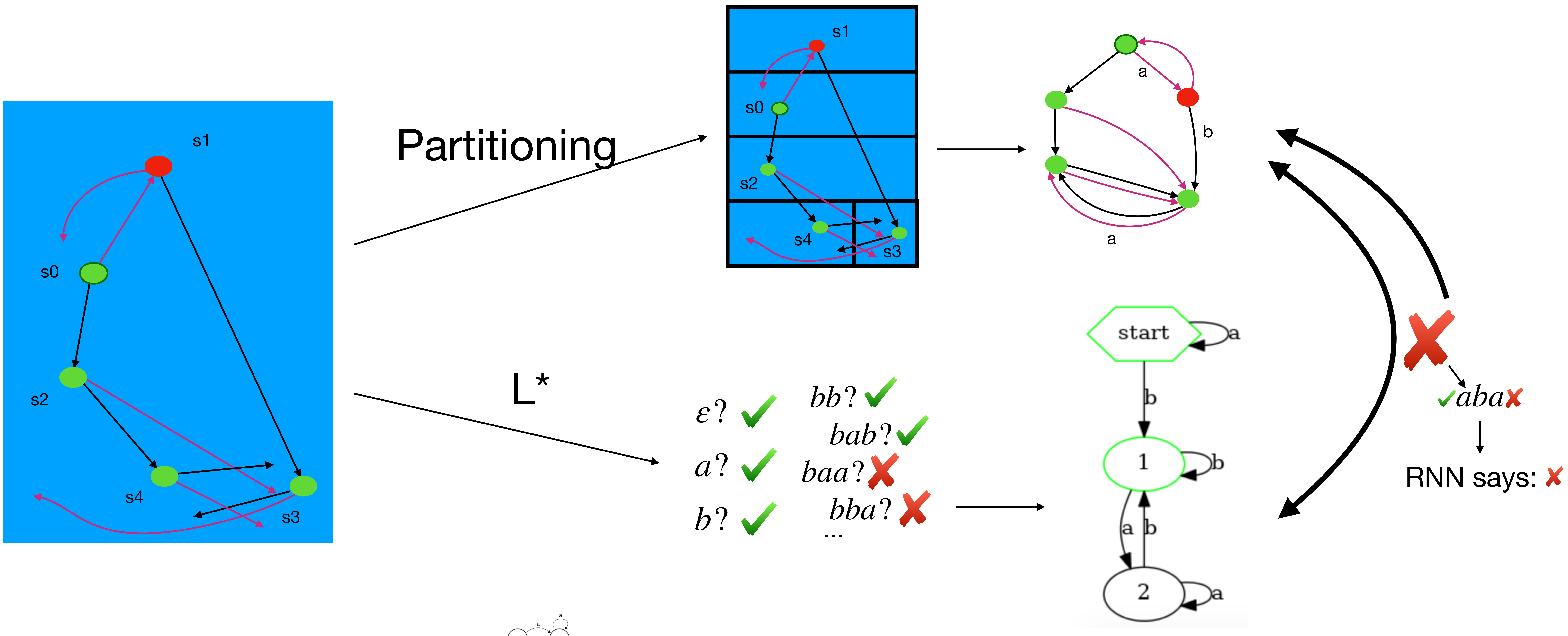
Iterative Approach

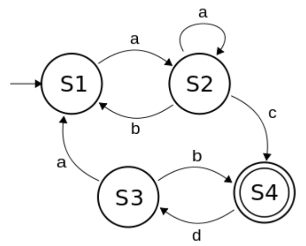




Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

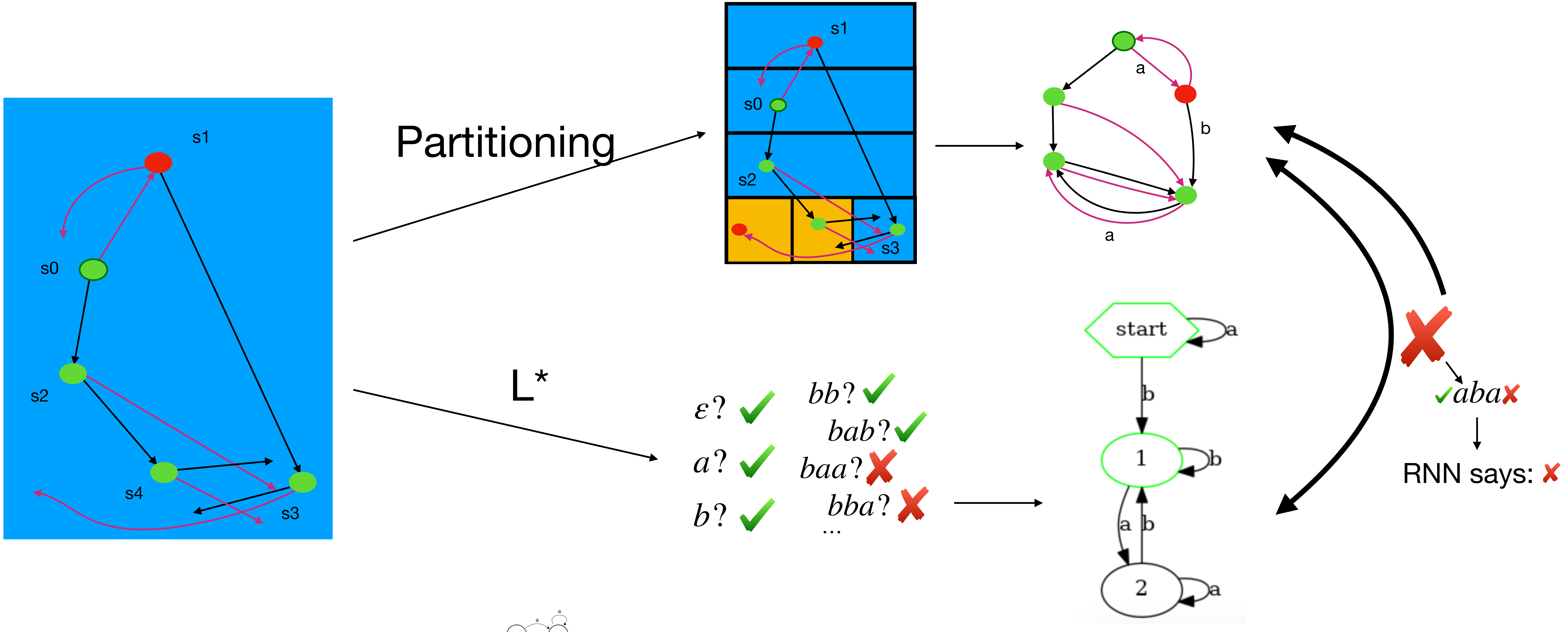
Iterative Approach

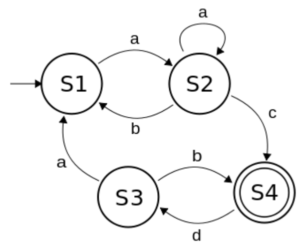




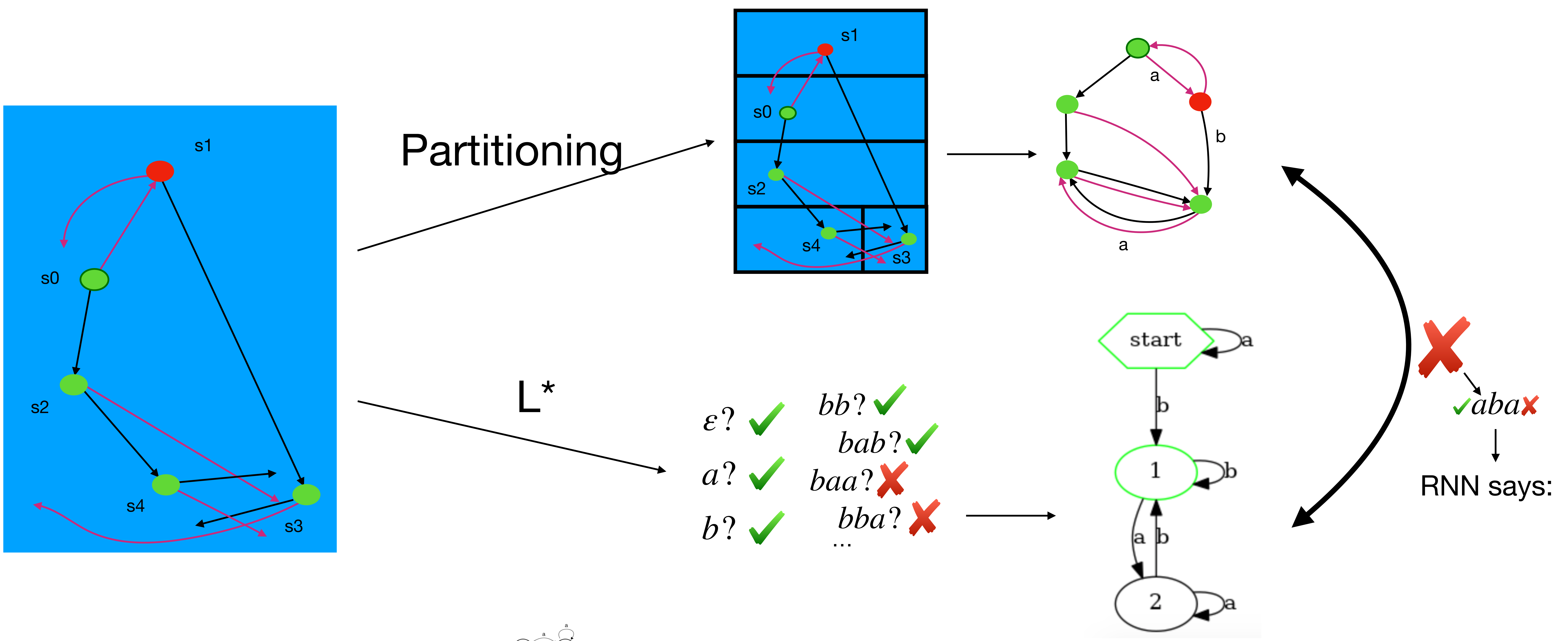
Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

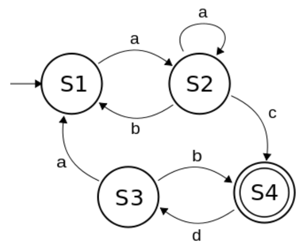
Iterative Approach




Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

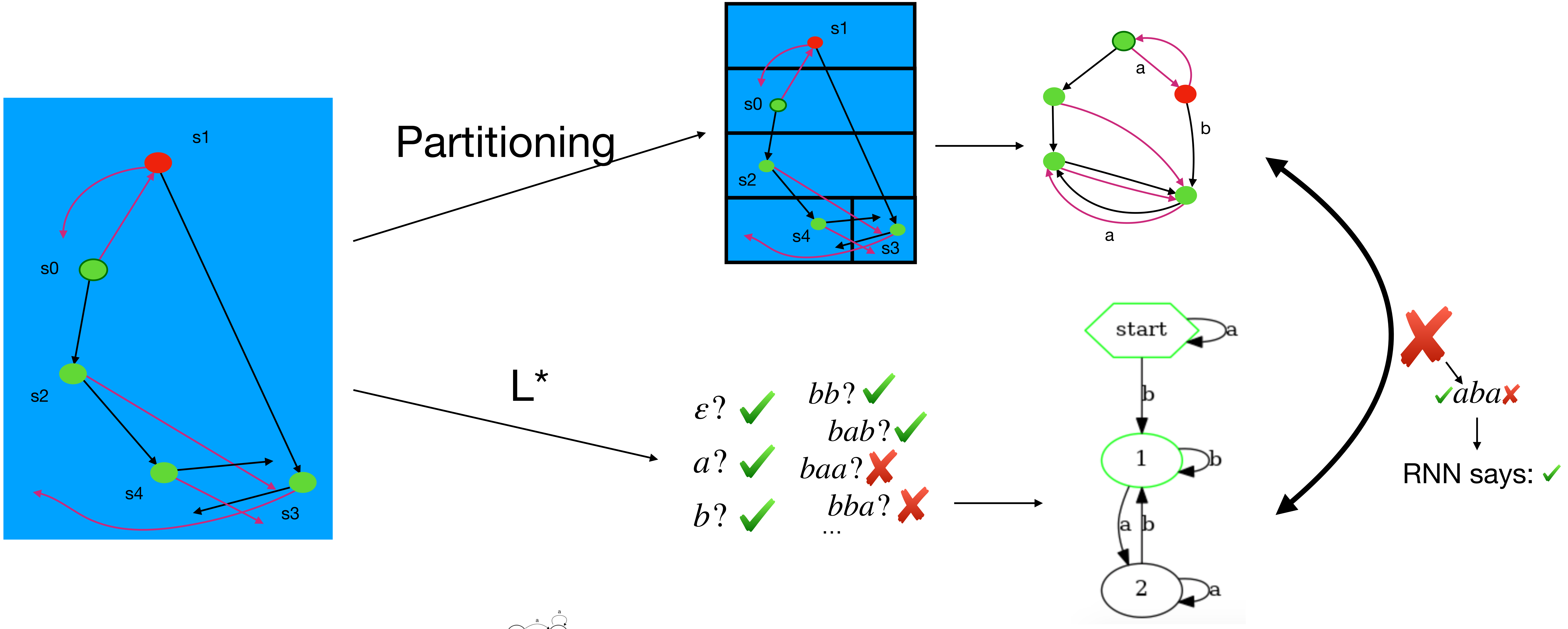
Iterative Approach

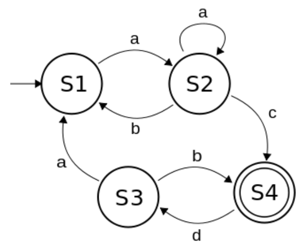




Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

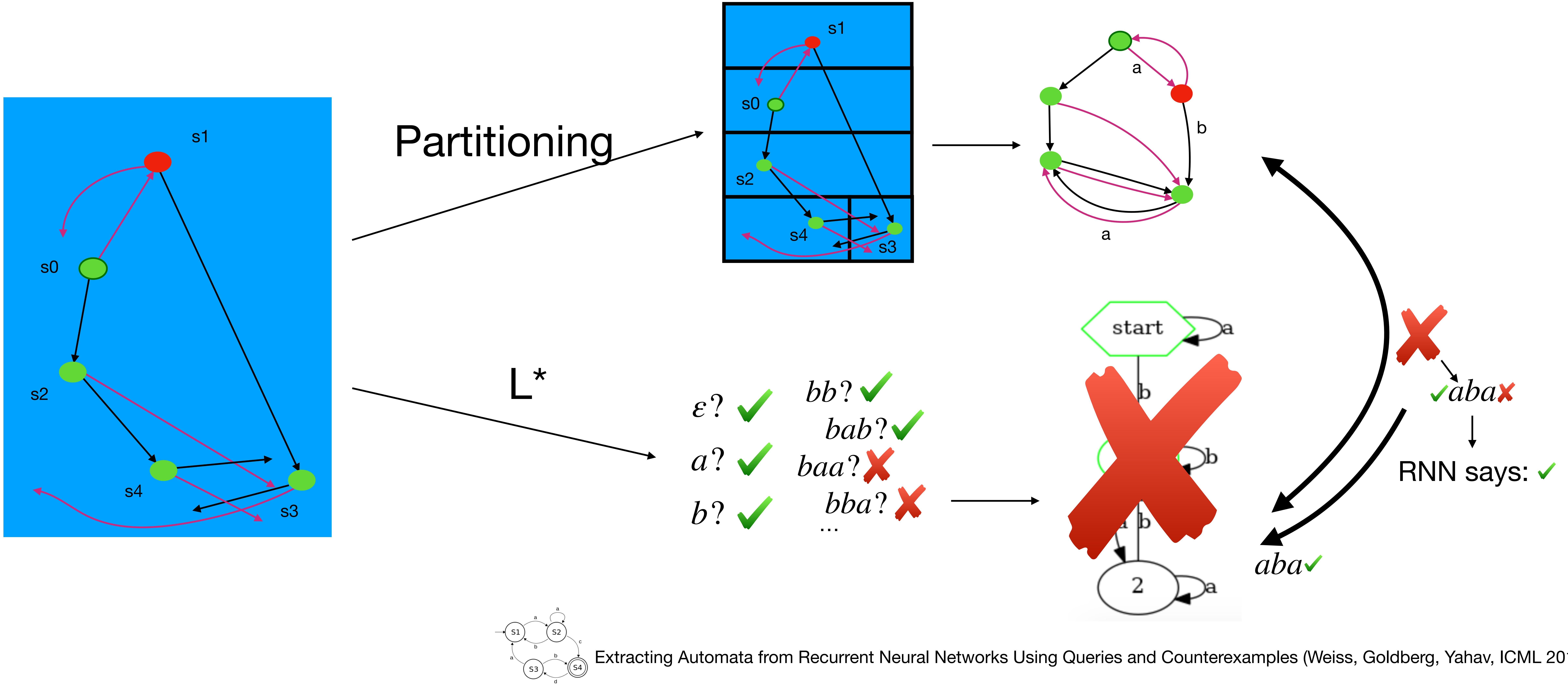
Iterative Approach





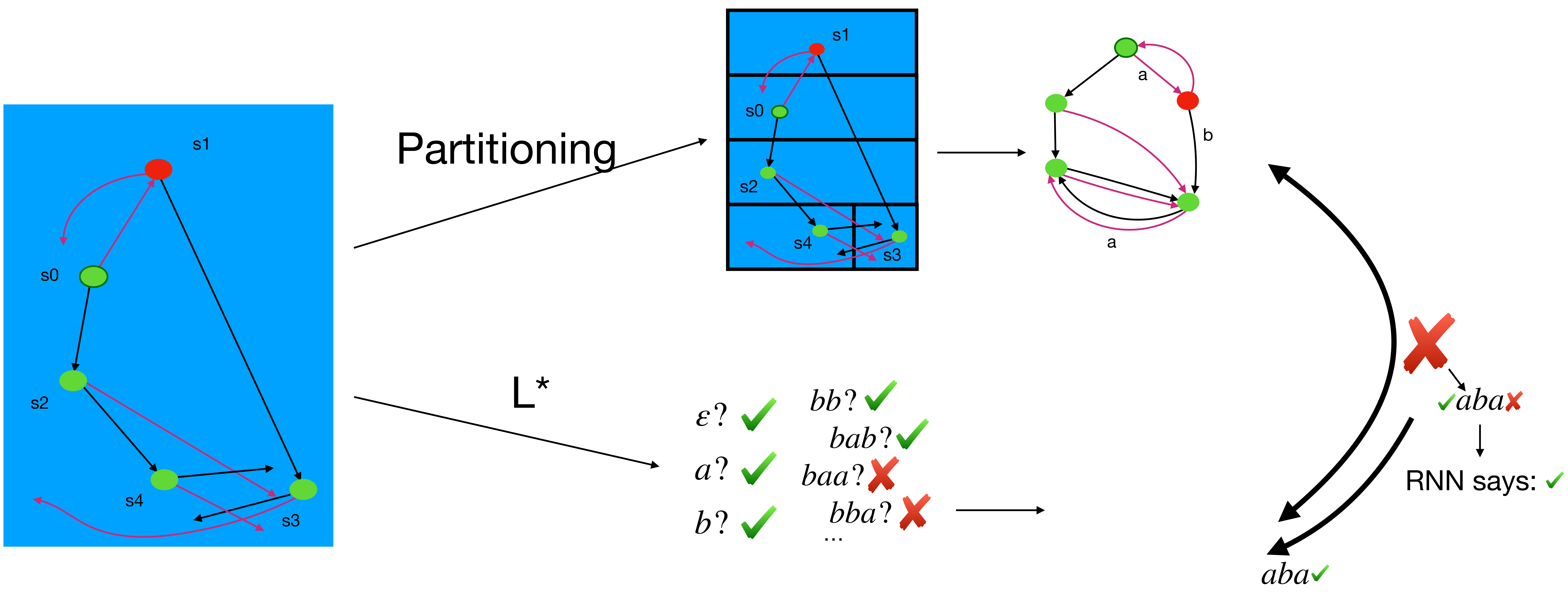
Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

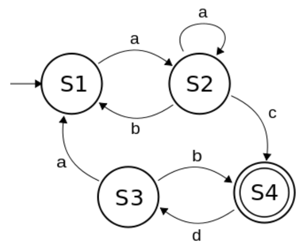
Iterative Approach



Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

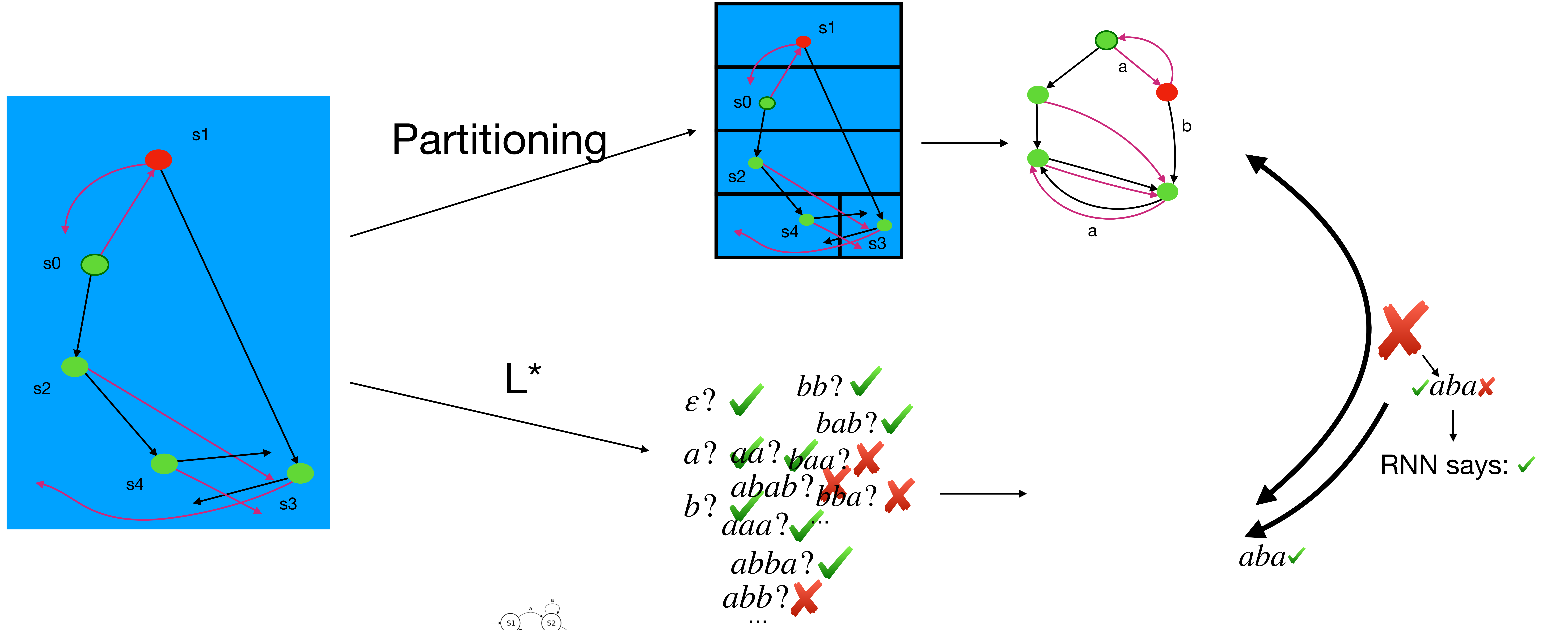
Iterative Approach

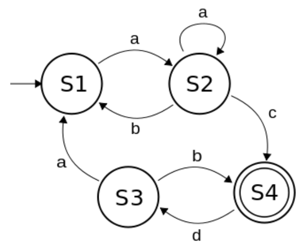




Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

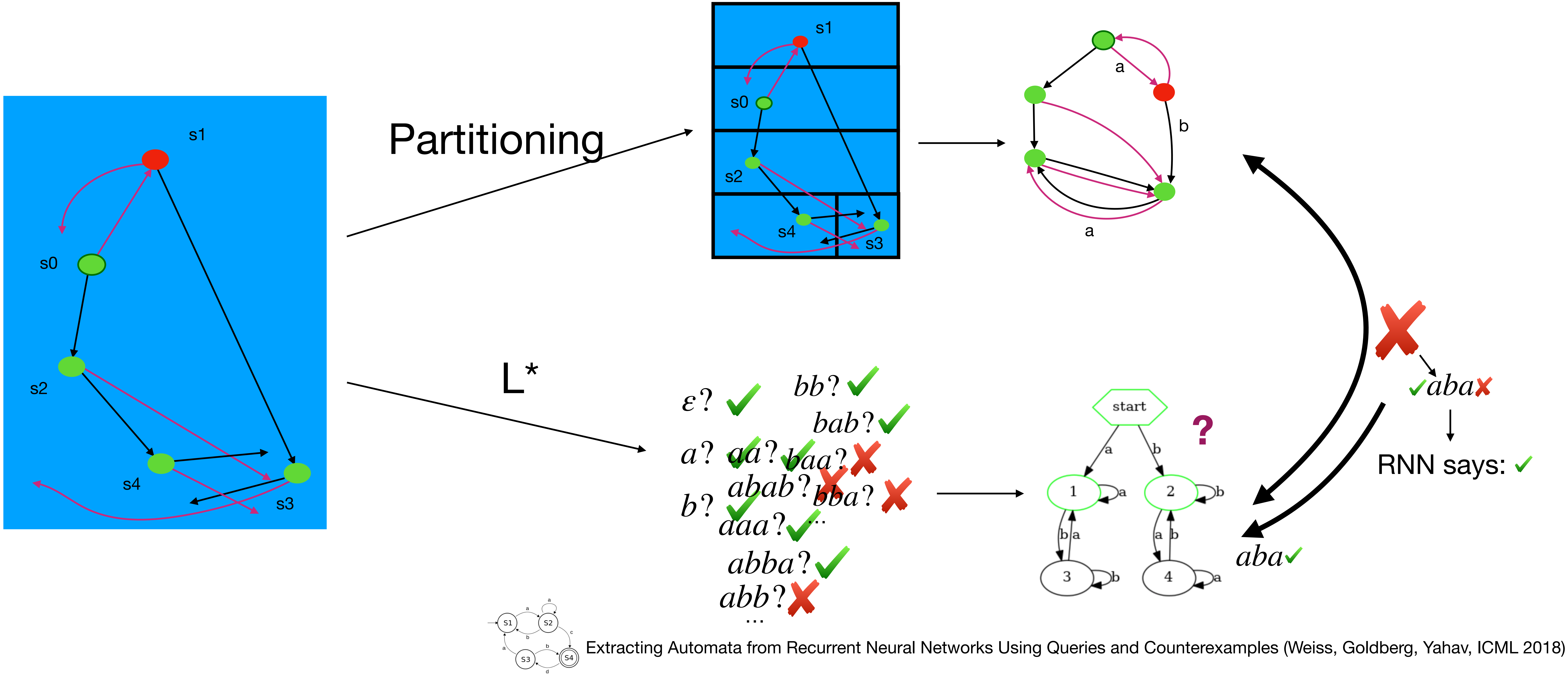
Iterative Approach





Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

Iterative Approach



Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

Results

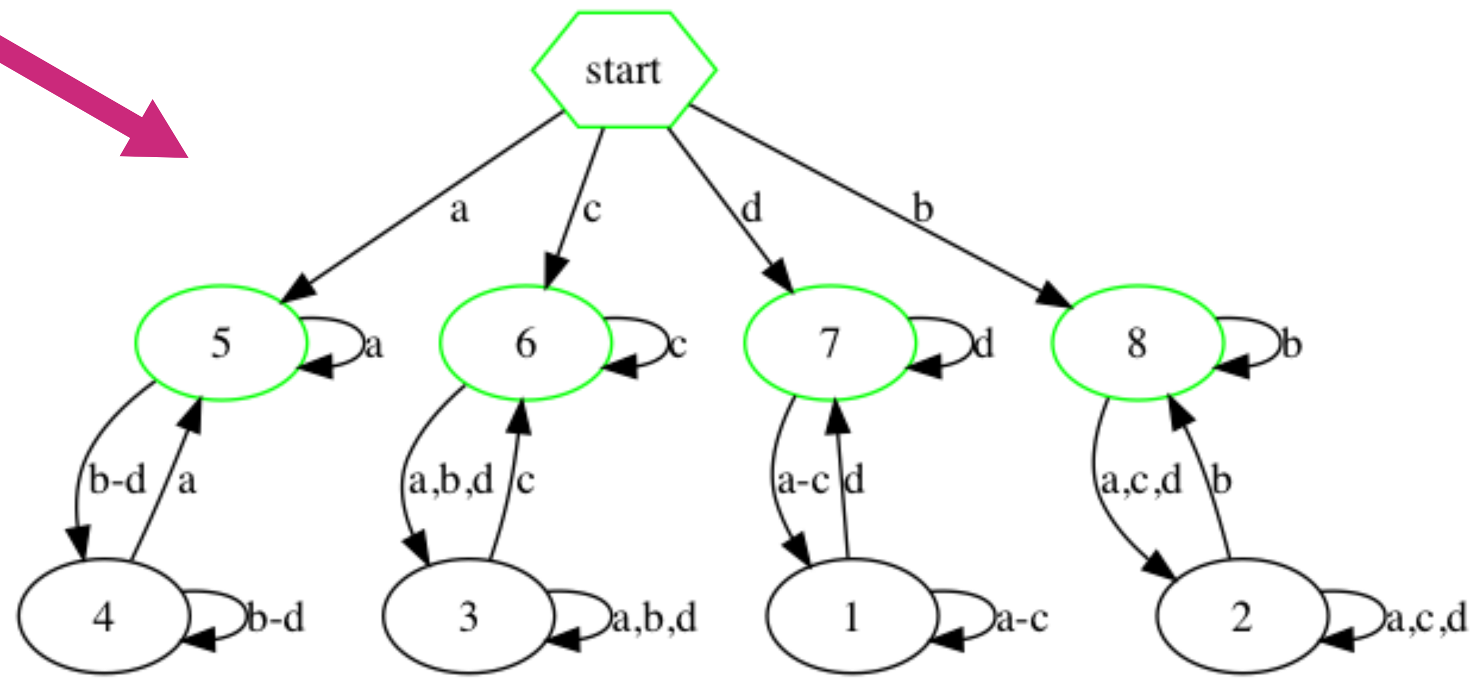
1. Concise, Exact Models in Short Time:

```
def target(w):
    if len(w)==0:
        return True
    return w[0]==w[-1]
alphabet = "abcd"
```

Training
(4,400 samples to 100% accuracy)

RNN

Extraction
0.2s



Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

Results

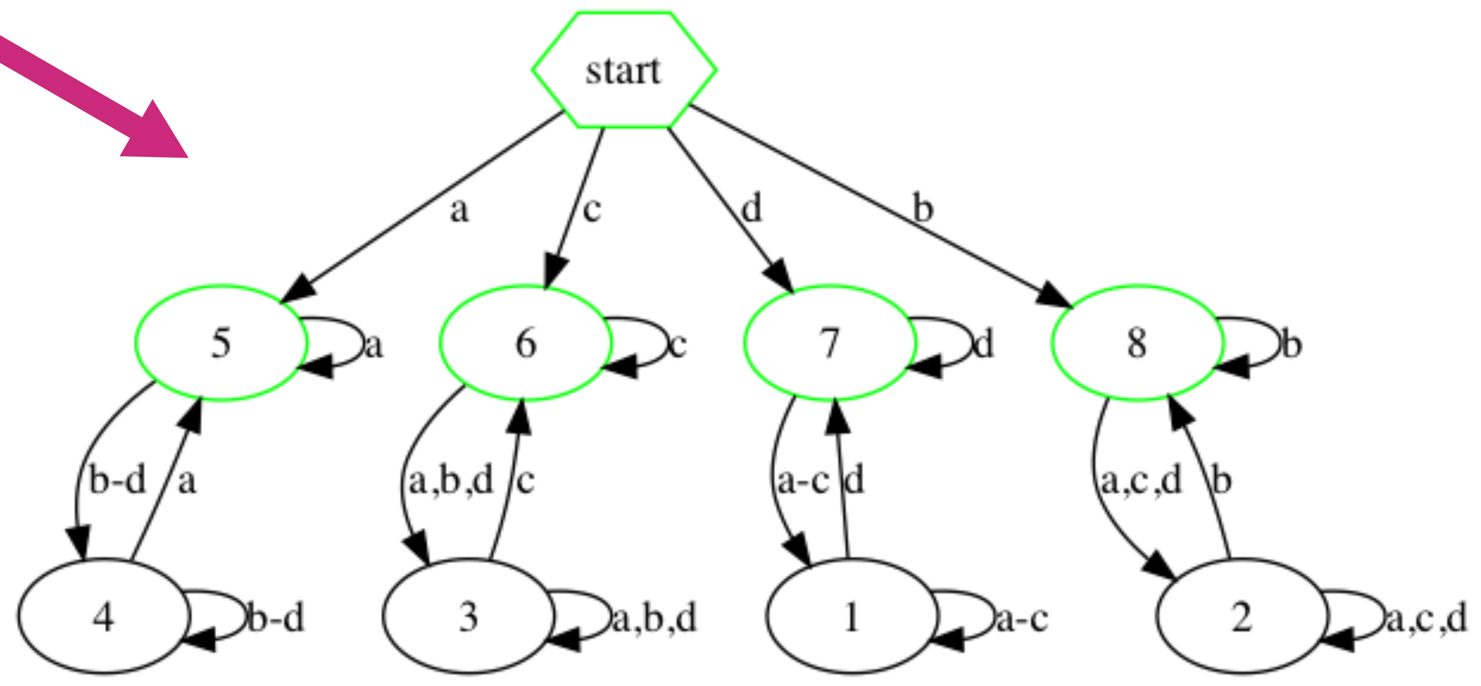
1. Concise, Exact Models in Short Time:

```python
def target(w):
    if len(w)==0:
        return True
    return w[0]==w[-1]
alphabet = "abcd"
```

Training
(4,400 samples to 100% accuracy)

RNN

Extraction
0.2s



2. **Adversarial Examples** (finding flaws)

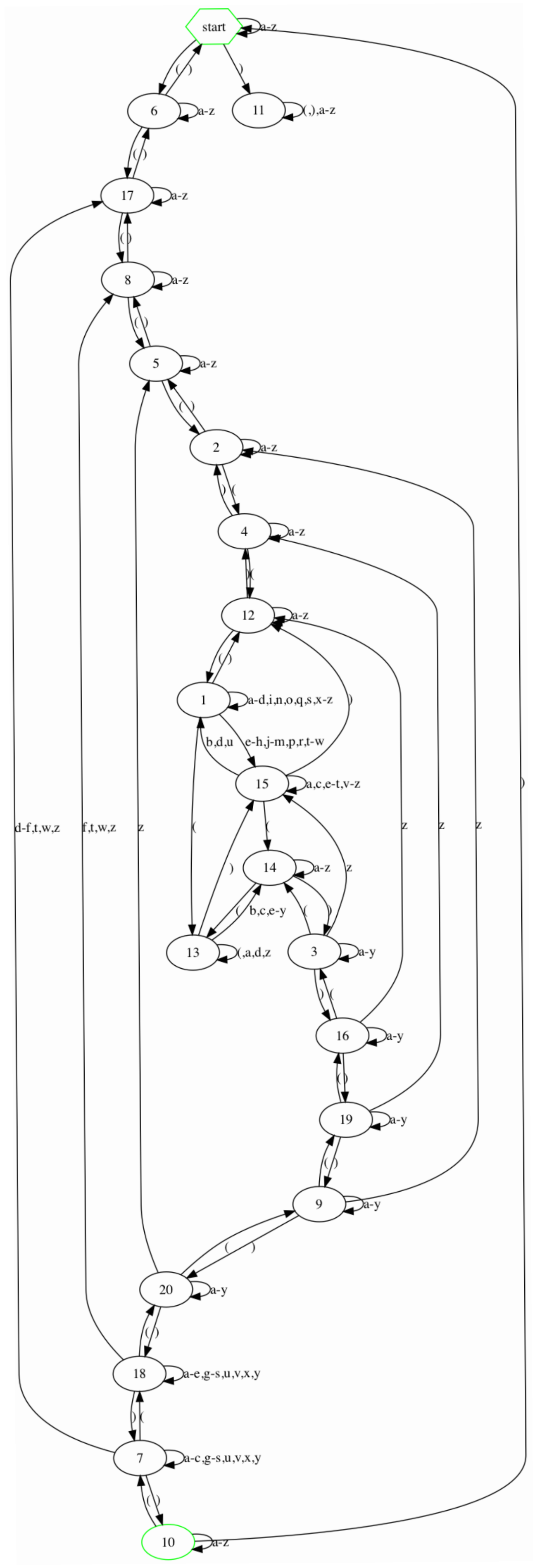
Balanced Parentheses GRU
100% train set accuracy
BP up to depth 11, over alphabet: ()a-z

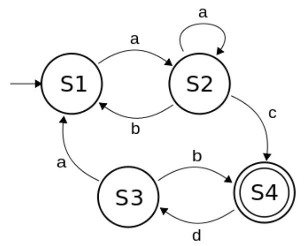
Counterexamples:

))	(1.1s)
(())	(1.2s)
((()))	(2.1s)
(((())))	(3.1s)
((((()))))	(3.8s)
(((((())))))	(4.4s)
((((((()))))))	(6.6s)
(((((((())))))))	(9.2s)
((((((((v()))))))))	(10.7s)
((((((((a()z))))))))	(8.3s)

Comparison:
Random sampling counterexamples:

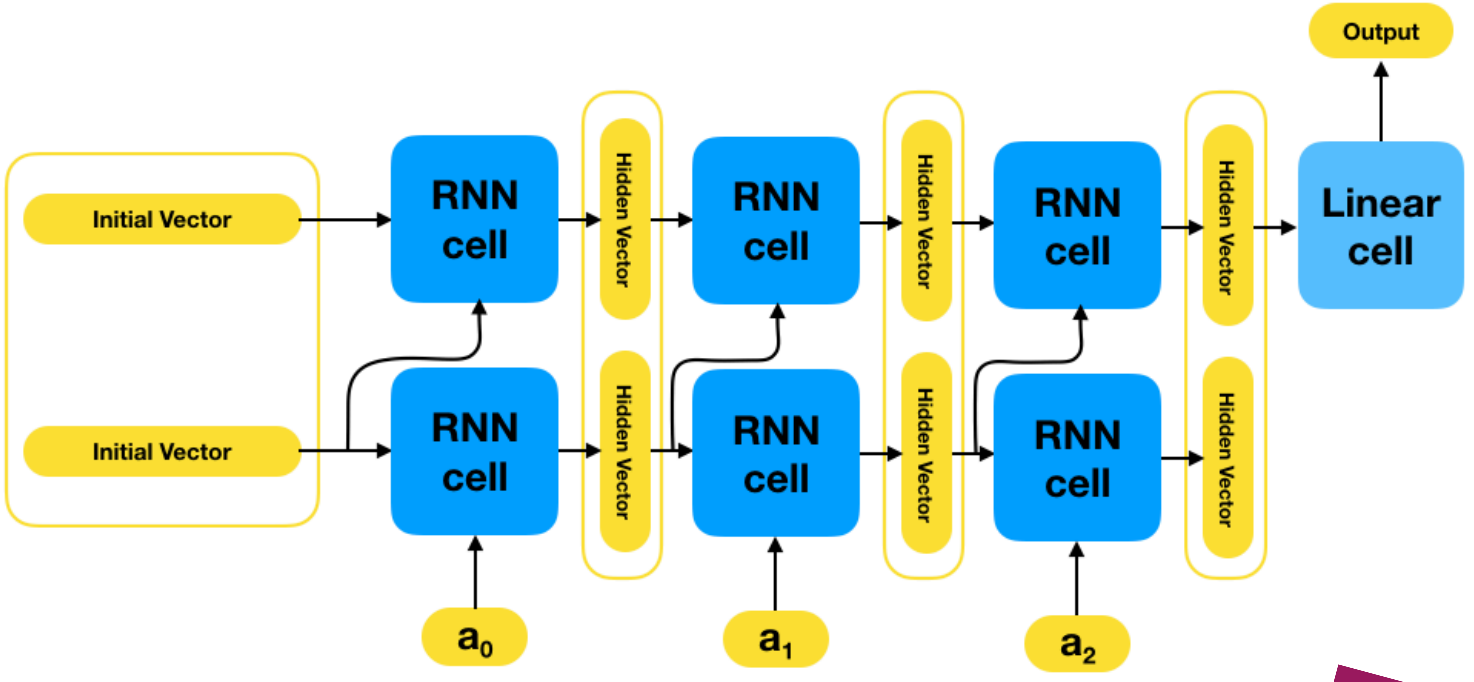
))	(0.4s)
(()i)ma	(32.6s)

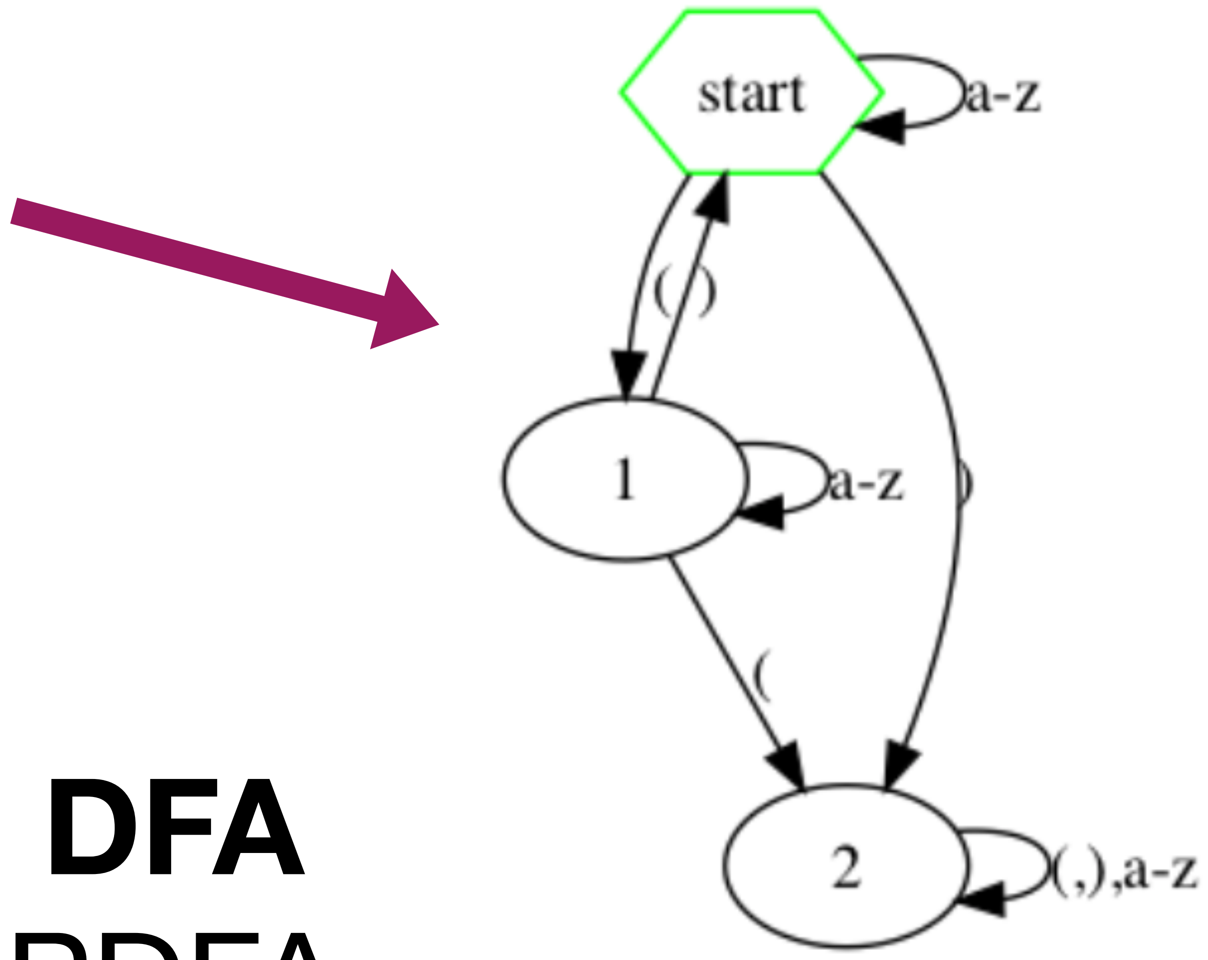




Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

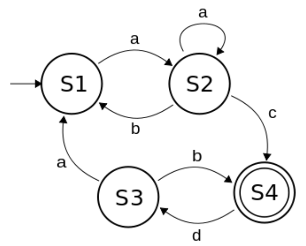
DFAs from RNNs





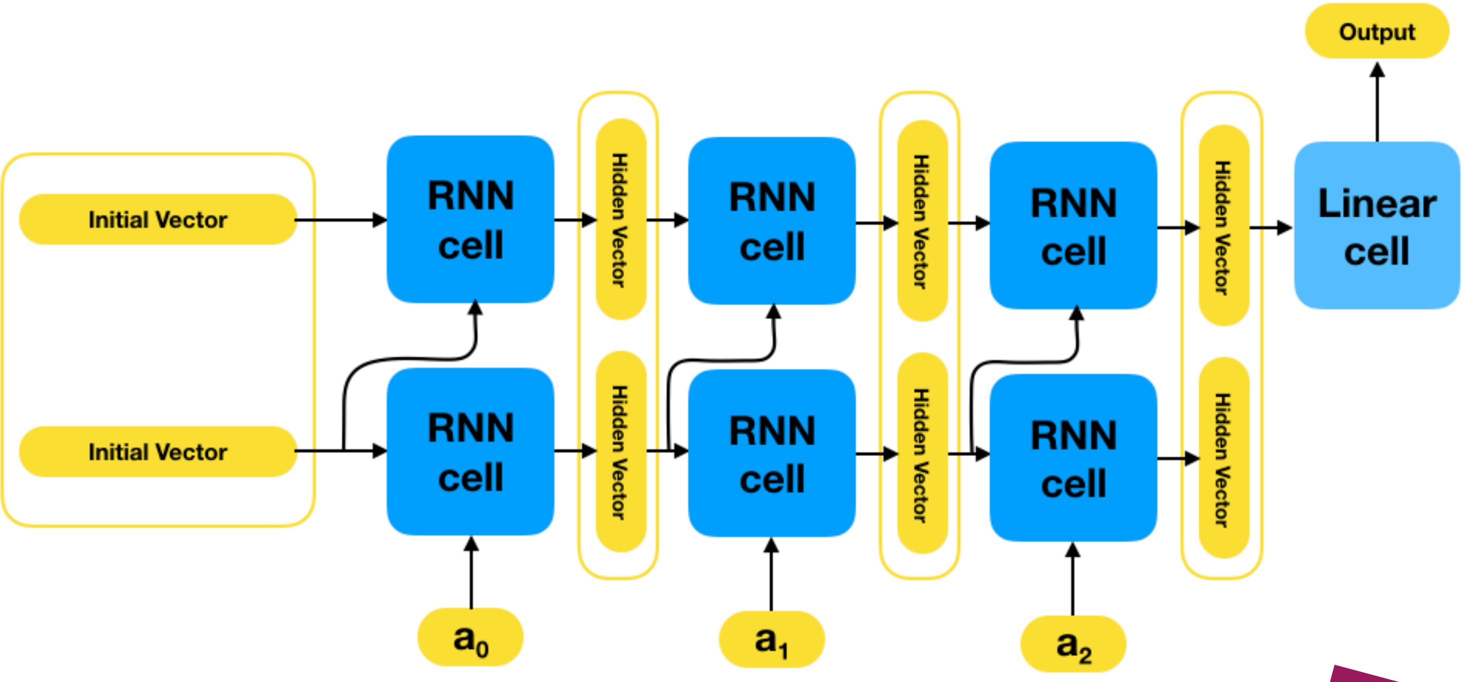
Goal:
Concise (Meaningful) **Model**
from Trained RNN

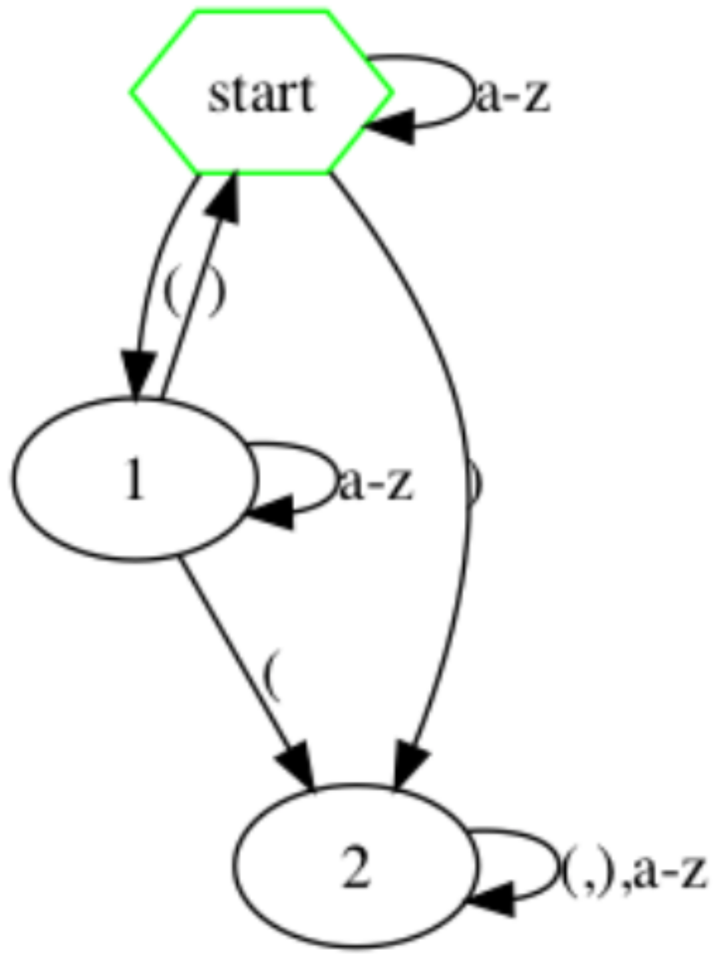
DFA
PDFA
CFG



Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Weiss, Goldberg, Yahav, ICML 2018)

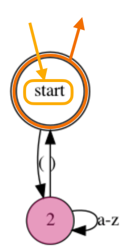
DFAs from RNNs





Goal:
Concise (Meaningful) **Model**
from Trained RNN

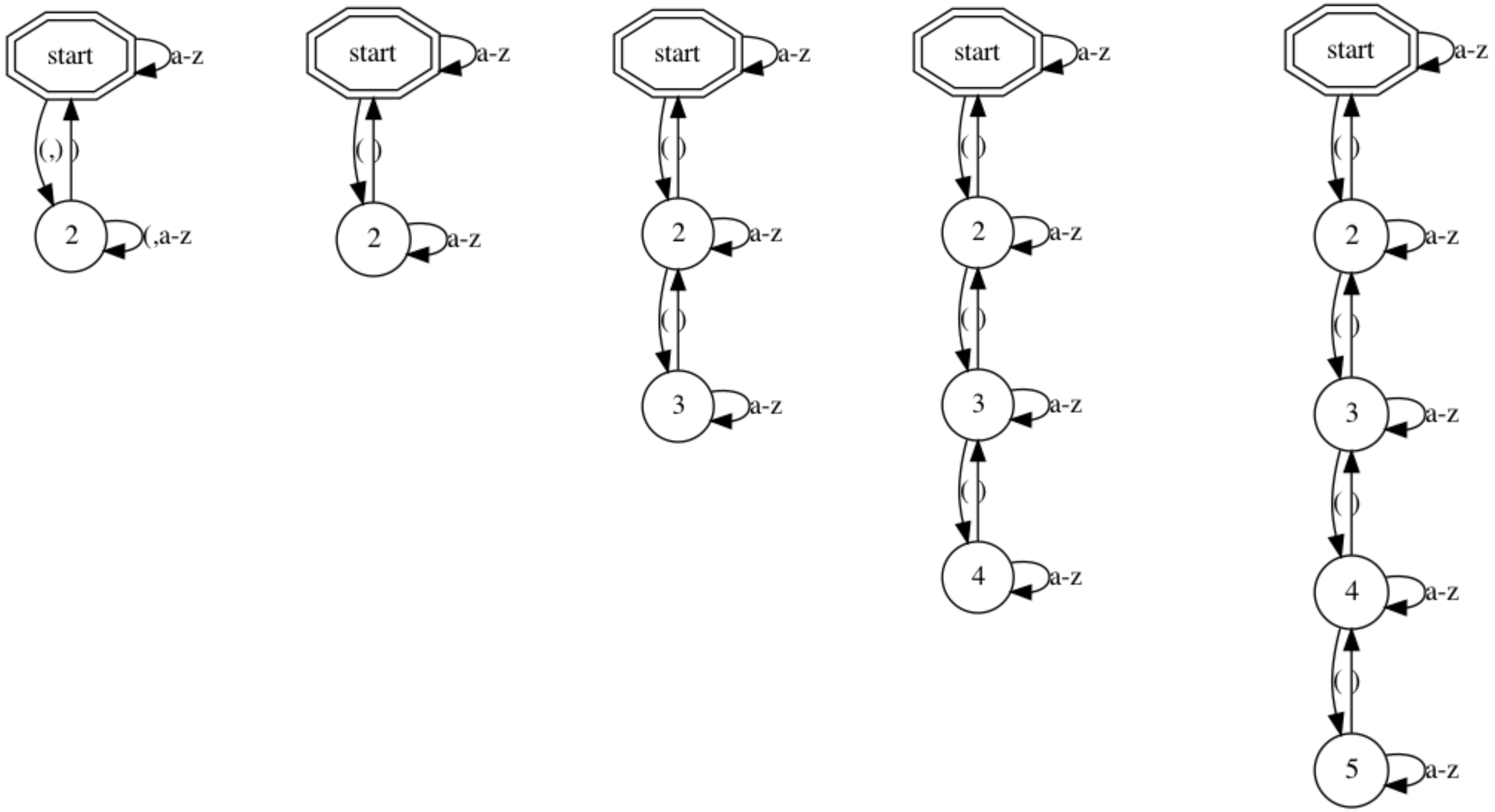
DFA
PDFA
CFG



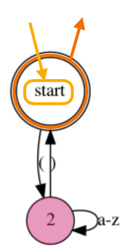
Synthesising Context Free Grammars from Recurrent Neural Networks (Yellin, Weiss, TACAS 2021)

CFGs from RNNs

Observation: L-star learning a CFG seems to have **structured increases** (example on BP)

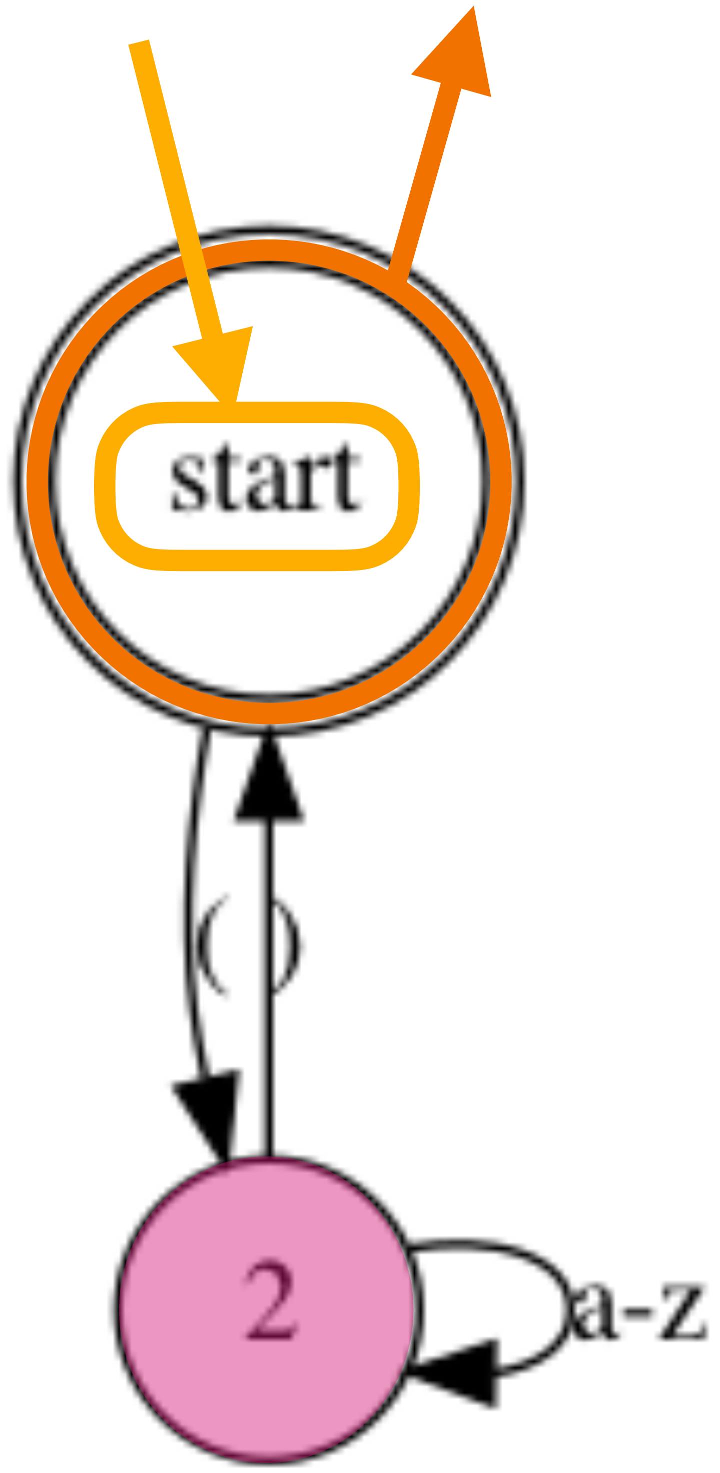


etc…



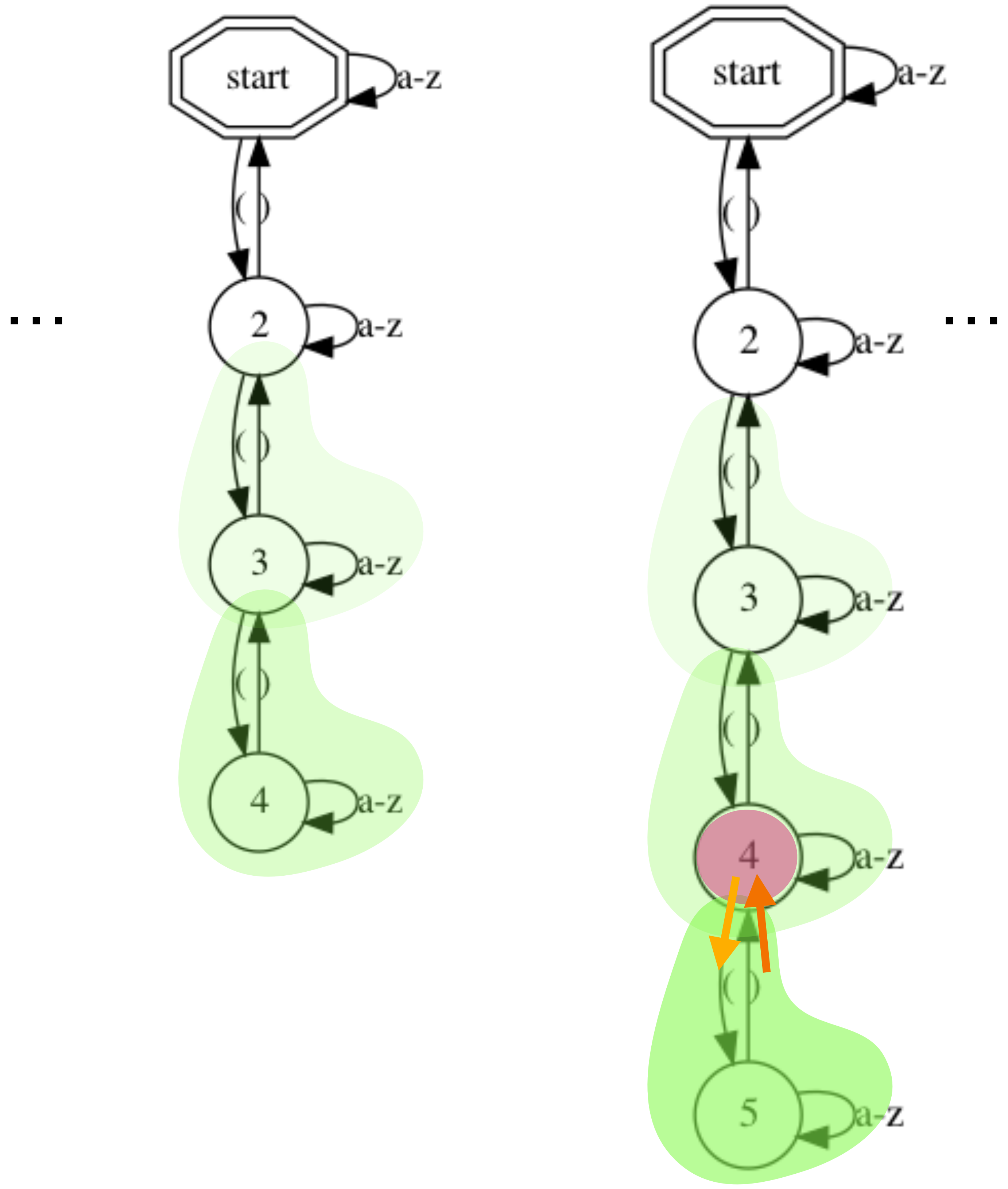
Synthesising Context Free Grammars from Recurrent Neural Networks (Yellin, Weiss, TACAS 2021)

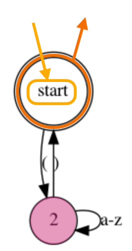
CFGs from RNNs



Patterns

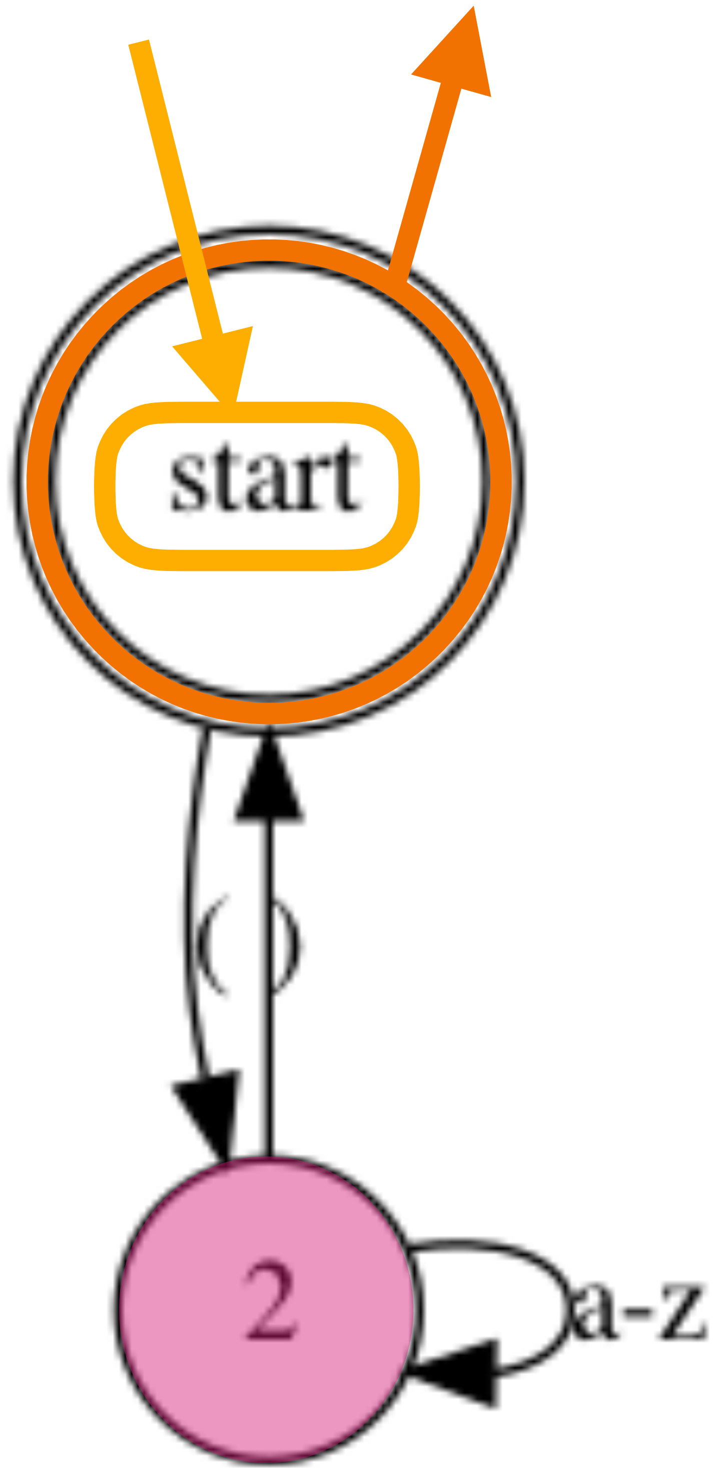
- Structure
  - Entry
  - Exit
- Connection Point(s)
- Composable





Synthesising Context Free Grammars from Recurrent Neural Networks (Yellin, Weiss, TACAS 2021)

CFGs from RNNs

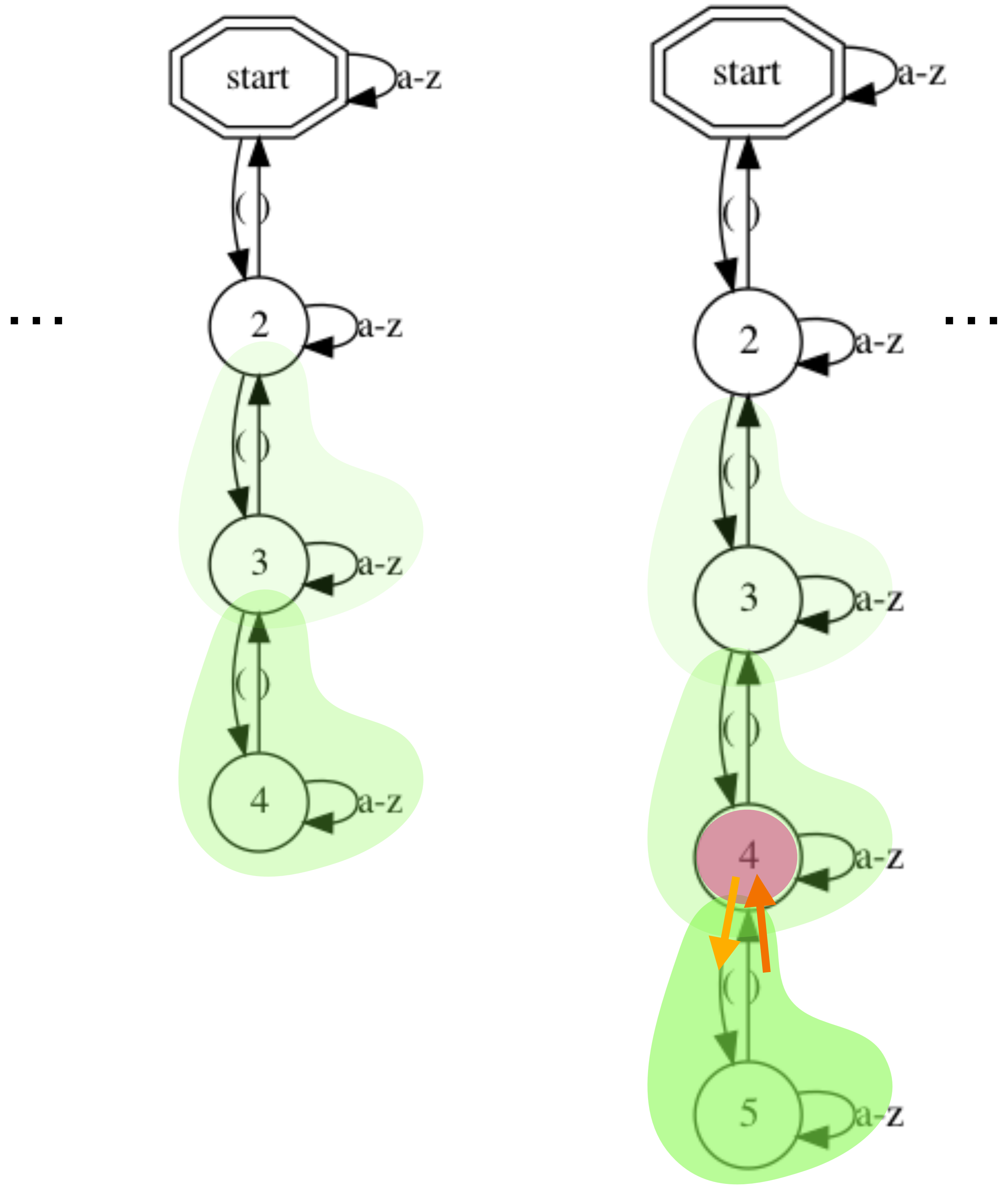


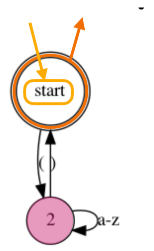
Patterns

- Structure
  - Entry
  - Exit
- Connection Point(s)
- Composable

Rules

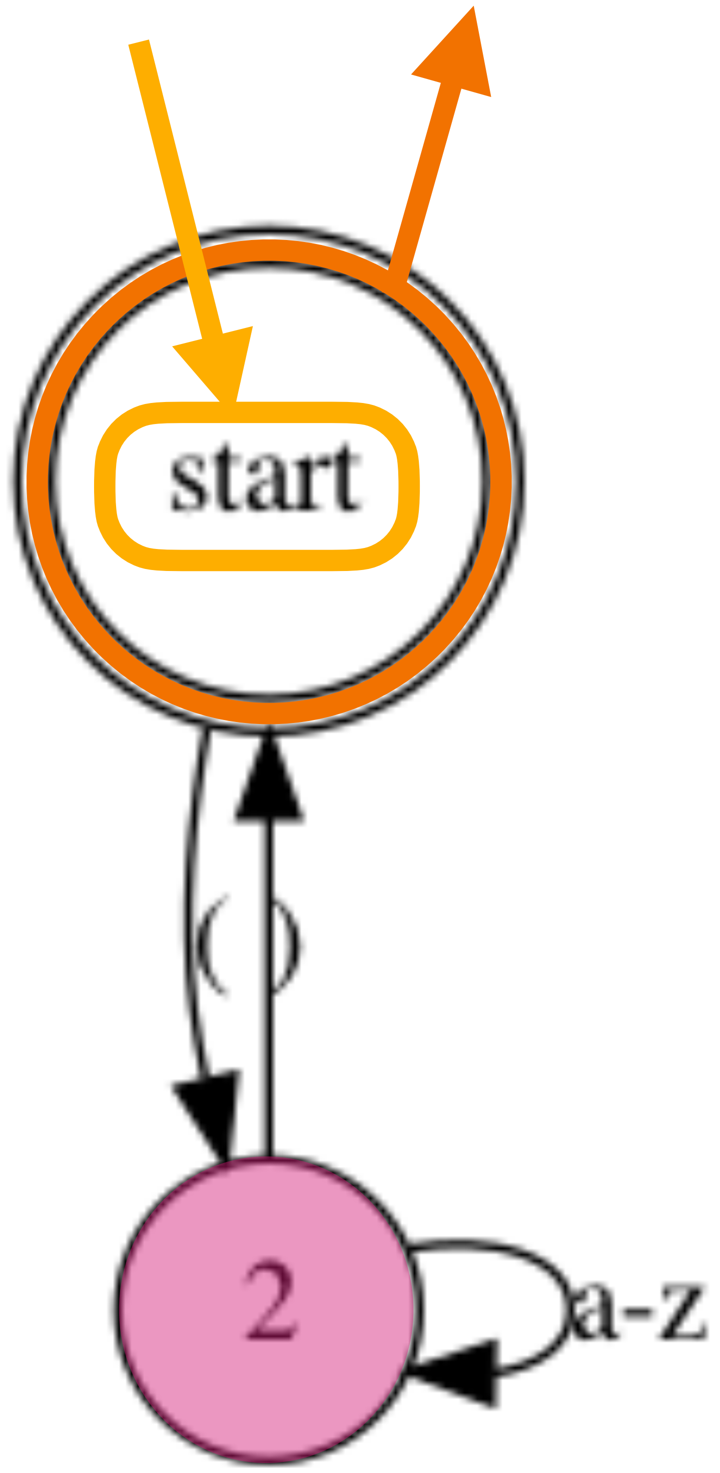
- Describe legal compositions
  - Legal sequences of DFAs





Synthesising Context Free Grammars from Recurrent Neural Networks (Yellin, Weiss, TACAS 2021)

CFGs from RNNs

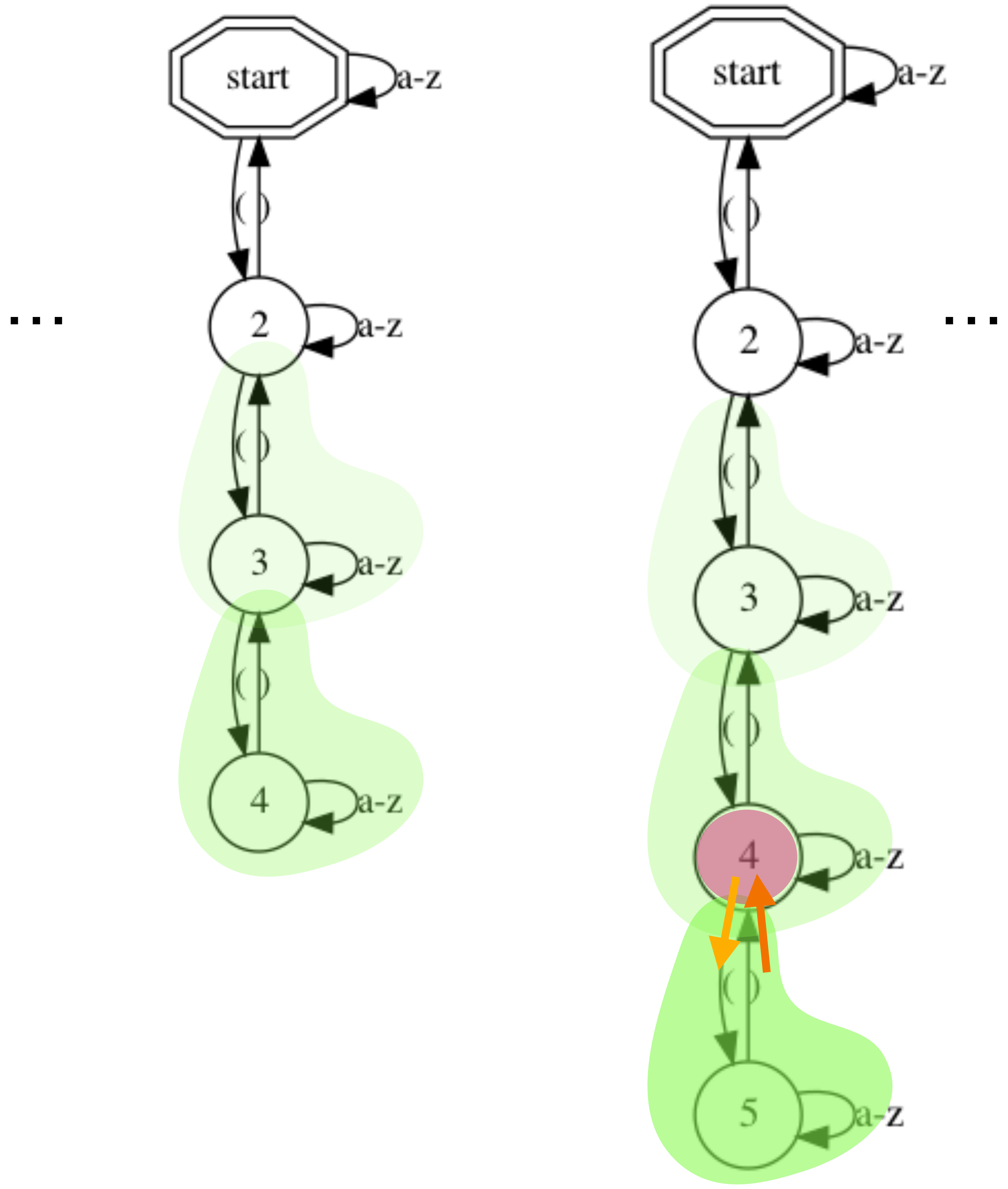


Patterns

- Structure
  - Entry
  - Exit
- Connection Point(s)
- Composable

Rules

- Describe legal compositions
  - Legal sequences of DFAs

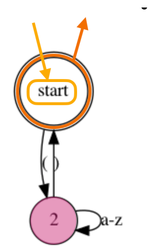


Result:

Algorithm to recover ***Pattern Rule Sets*** from a sequence of DFAs

Sequence can be obtained from L-star extraction

Some tolerance to noise!

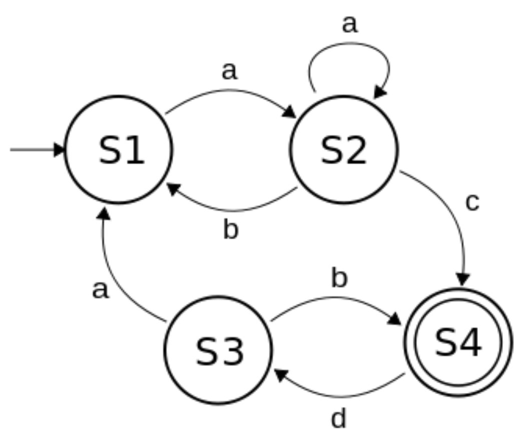


Synthesising Context Free Grammars from Recurrent Neural Networks (Yellin, Weiss, TACAS 2021)

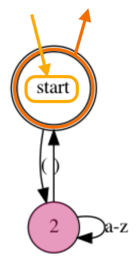
Neural Sequence Models: a Formal Lens

Counting
LSTMs are counter machines, GRUs aren't **(ACL 2018)**

RASP
Finding a formalism to describe transformers **(ICML 2021)**



DFAs from RNNs
Applying L* to learn DFAs from RNNs **(ICML 2018)**

\+  using the result for CFGs **(TACAS 2021)**

Neural Sequence Models:
a Formal Lens

WDFAs from RNNs

Adapting L* to the (noisy!) weighted case (Neurips 2019)

A Hierarchy of RNNs

Comparing more RNN architectures, with different angles (ACL 2020)

Thanks!