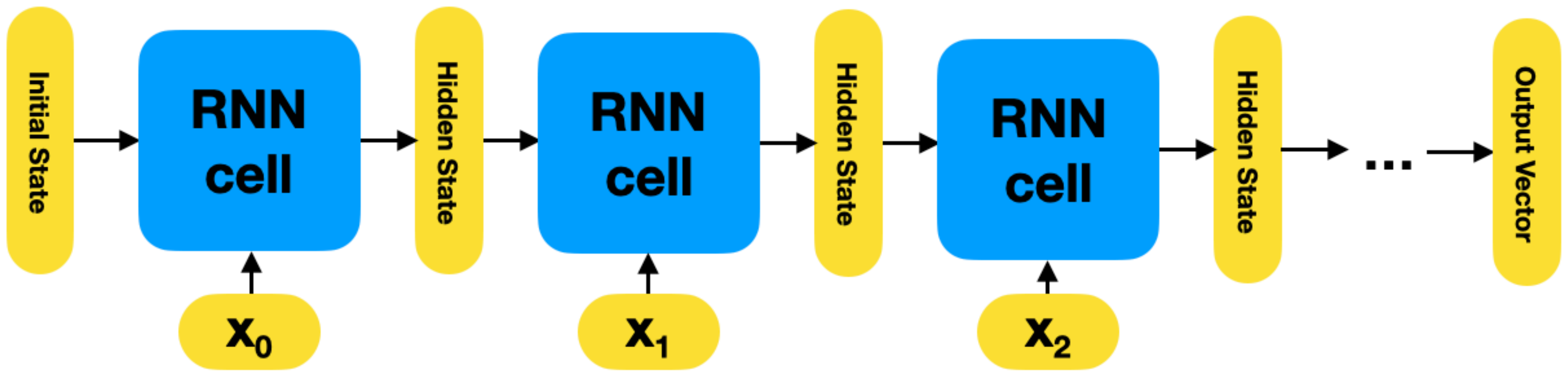
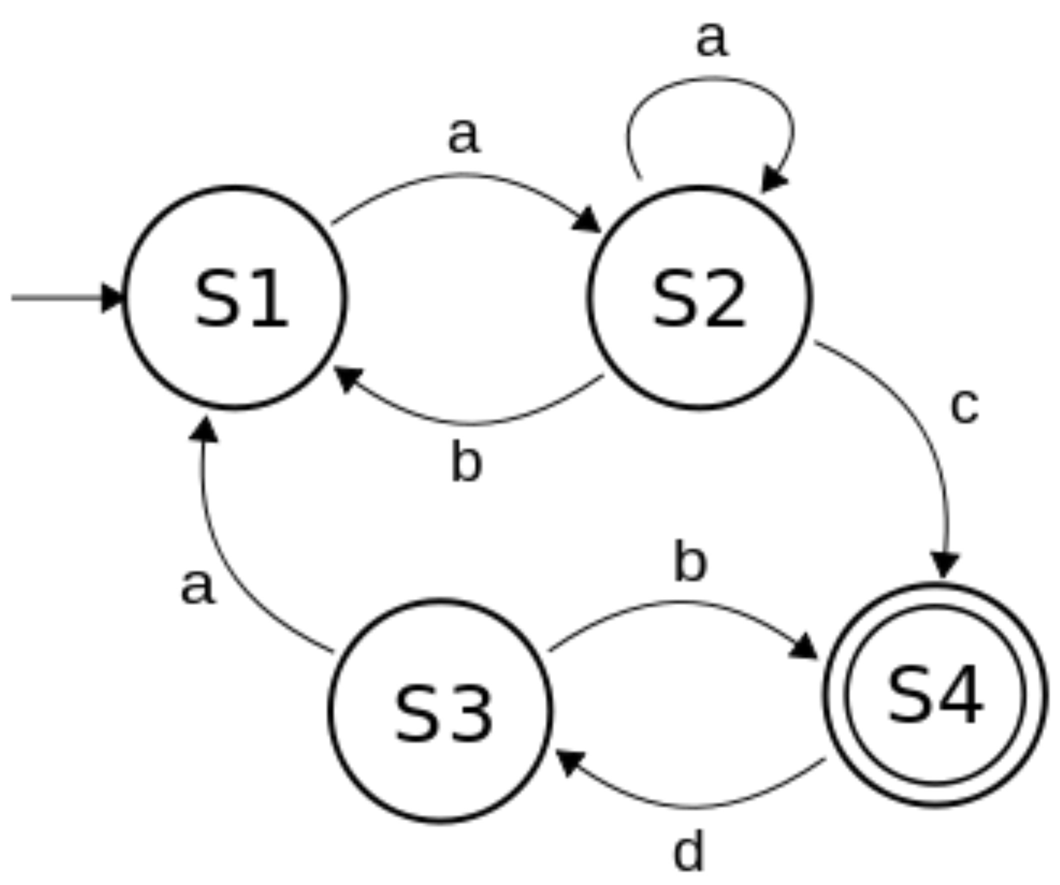
Theoretical Properties of RNNs

Gail Weiss



?



Outline

- **The state machine relation**
- Theoretical Power of RNNs: **Infinite Precision and Time**
- Theoretical Power of RNNs: **Finite Precision and Time**

Goal: appreciate effect of RNN architectures on practical expressive power

Outline

- **The state machine relation**
- Theoretical Power of RNNs: **Infinite Precision and Time**
- Theoretical Power of RNNs: **Finite Precision and Time**
    - Common Architectures (LSTMs, GRUs)
    - An Uncommon Architecture (QRNNs)
    - A Completely Made Up Architecture (Failed RNN 😔)
- Augmentations (Stack RNNs)

Goal: appreciate effect of RNN architectures on practical expressive power

Outline

- **The state machine relation**
- Theoretical Power of RNNs: **Infinite Precision and Time**
- Theoretical Power of RNNs: **Finite Precision and Time**
    - Common Architectures (LSTMs, GRUs)
    - An Uncommon Architecture (QRNNs)
    - A Completely Made Up Architecture (Failed RNN 😔)
- Augmentations (Stack RNNs)

Simple state machines…

Goal: appreciate effect of RNN architectures on practical expressive power

Outline

- **The state machine relation**
- Theoretical Power of RNNs: **Infinite Precision and Time**
- Theoretical Power of RNNs: **Finite Precision and Time**
  - Common Architectures (LSTMs, GRUs)
  - An Uncommon Architecture (QRNNs)
  - A Completely Made Up Architecture (Failed RNN 😔)
- Augmentations (Stack RNNs)

Simple state machines…

Turing complete!?

Goal: appreciate effect of RNN architectures on practical expressive power

Outline

- **The state machine relation**
- Theoretical Power of RNNs: **Infinite Precision and Time**
- Theoretical Power of RNNs: **Finite Precision and Time**
  - Common Architectures (LSTMs, GRUs)
  - An Uncommon Architecture (QRNNs)
  - A Completely Made Up Architecture (Failed RNN 😔)
- Augmentations (Stack RNNs)

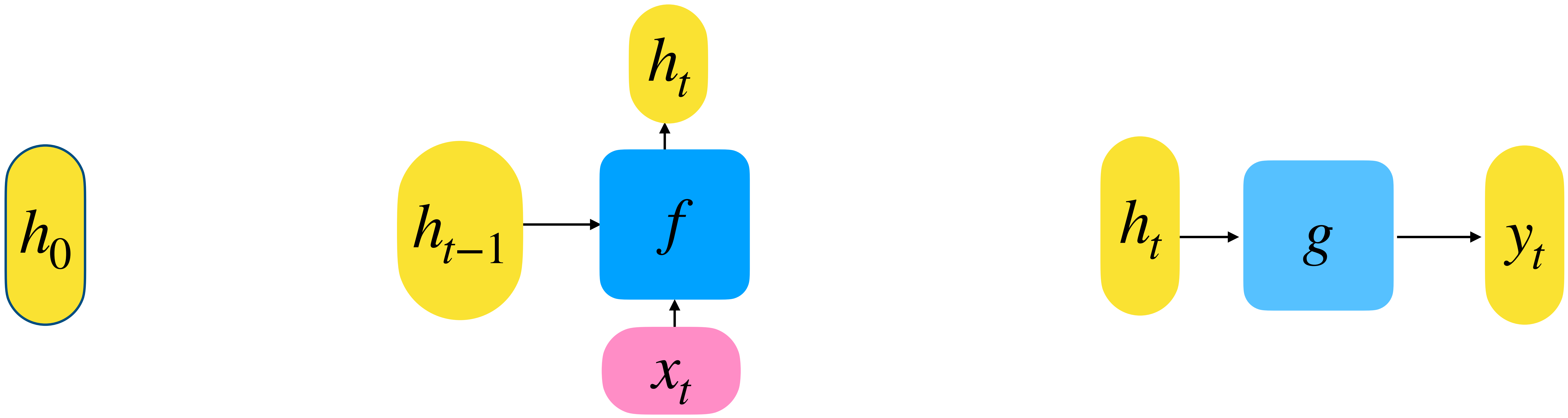
Simple state machines…

Turing complete!?

Oh, state machines… with extras

Goal: appreciate effect of RNN architectures on practical expressive power

Recap: RNNs

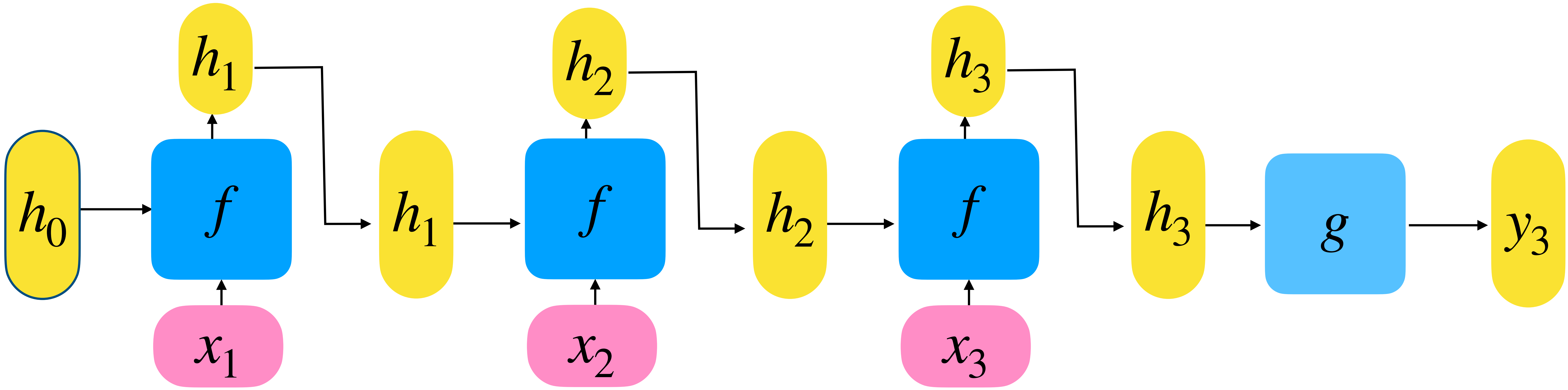


$R = (h_0, f, g)$

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

$h_t \in \mathbb{R}^d$ for some fixed dimension d

Recap: RNNs



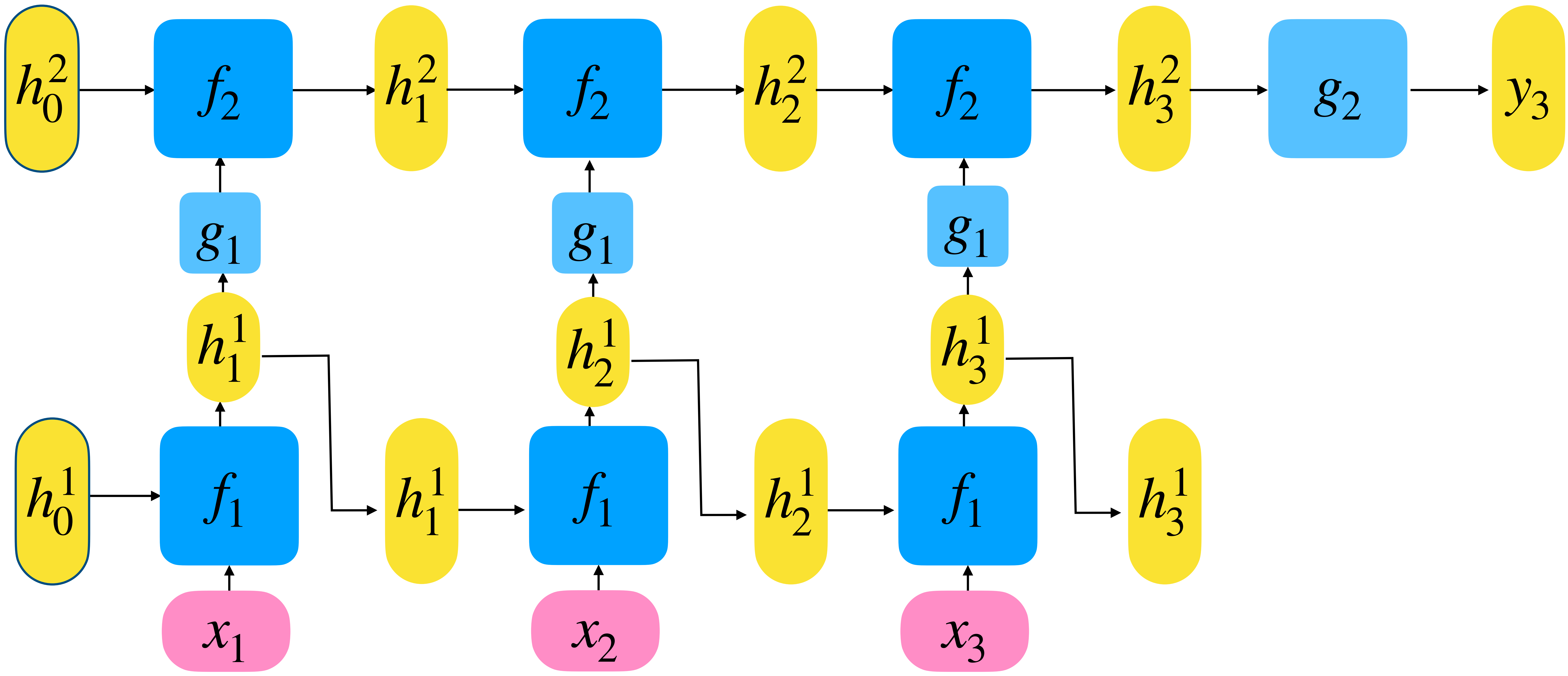
$R = (h_0, f, g)$

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

$h_t \in \mathbb{R}^d$ for some fixed dimension d

Recap: RNNs

(Multi-Layer RNNs)



$R = (h_0, f, g)$

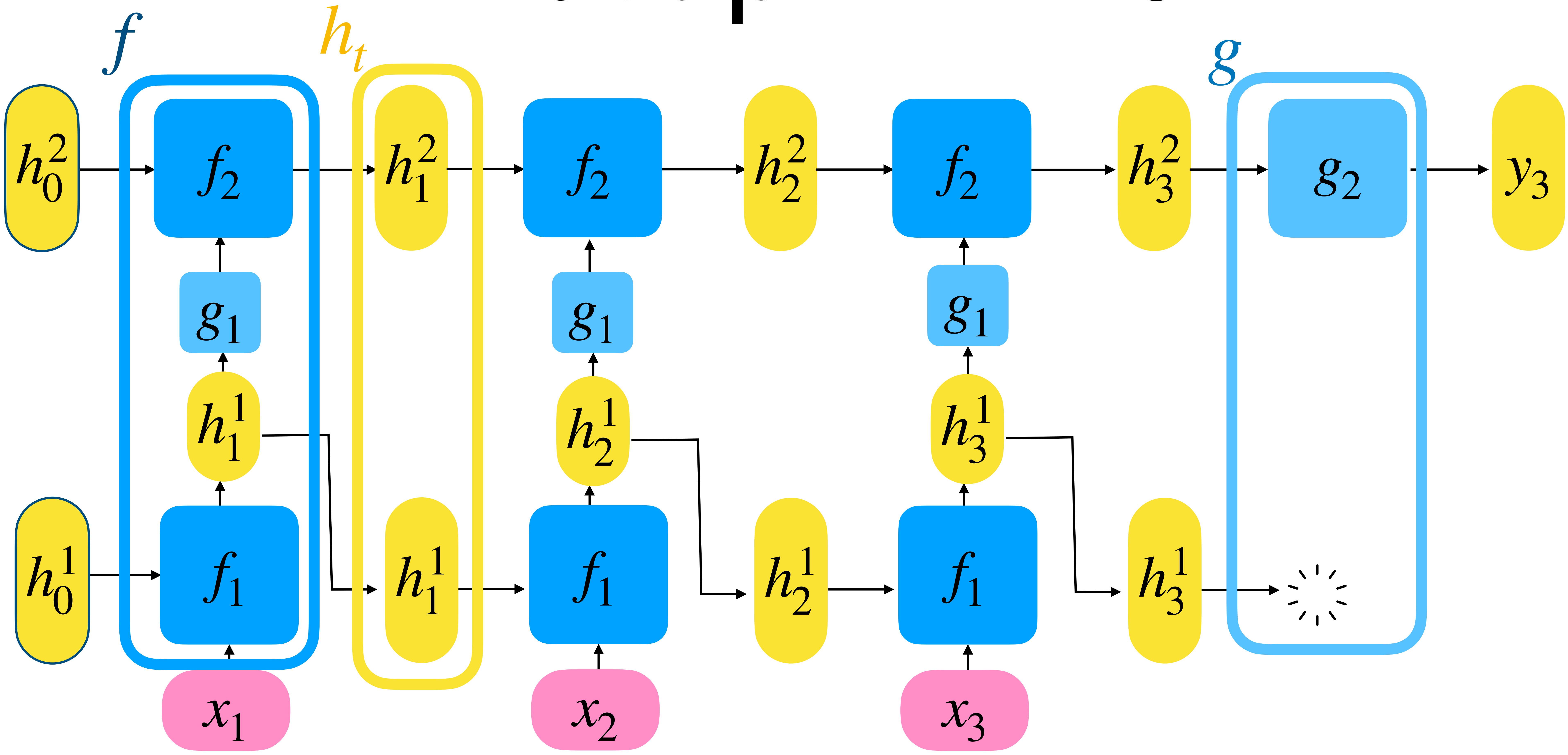
$h_t = f(h_{t-1}, x_t)$

$y_t = g(h_t)$

$h_t \in \mathbb{R}^d$ for some fixed dimension d

Recap: RNNs

(Multi-Layer RNNs)



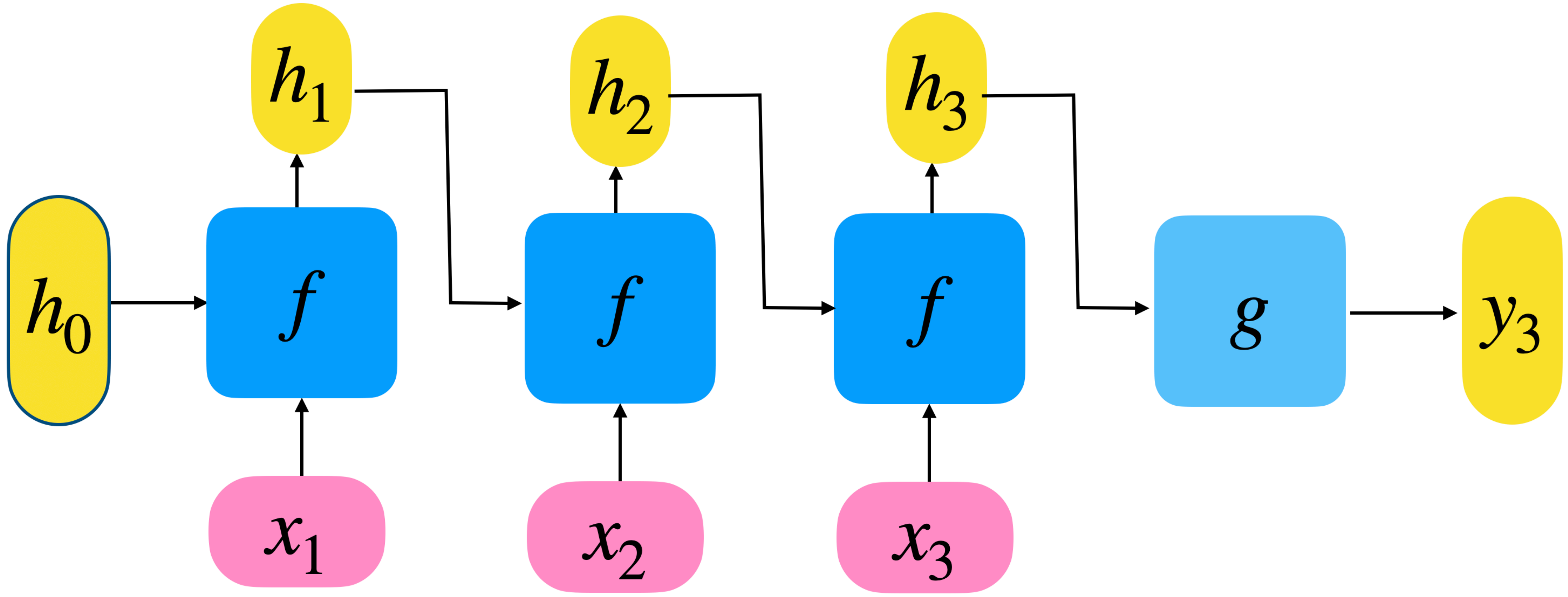
$R = (h_0, f, g)$

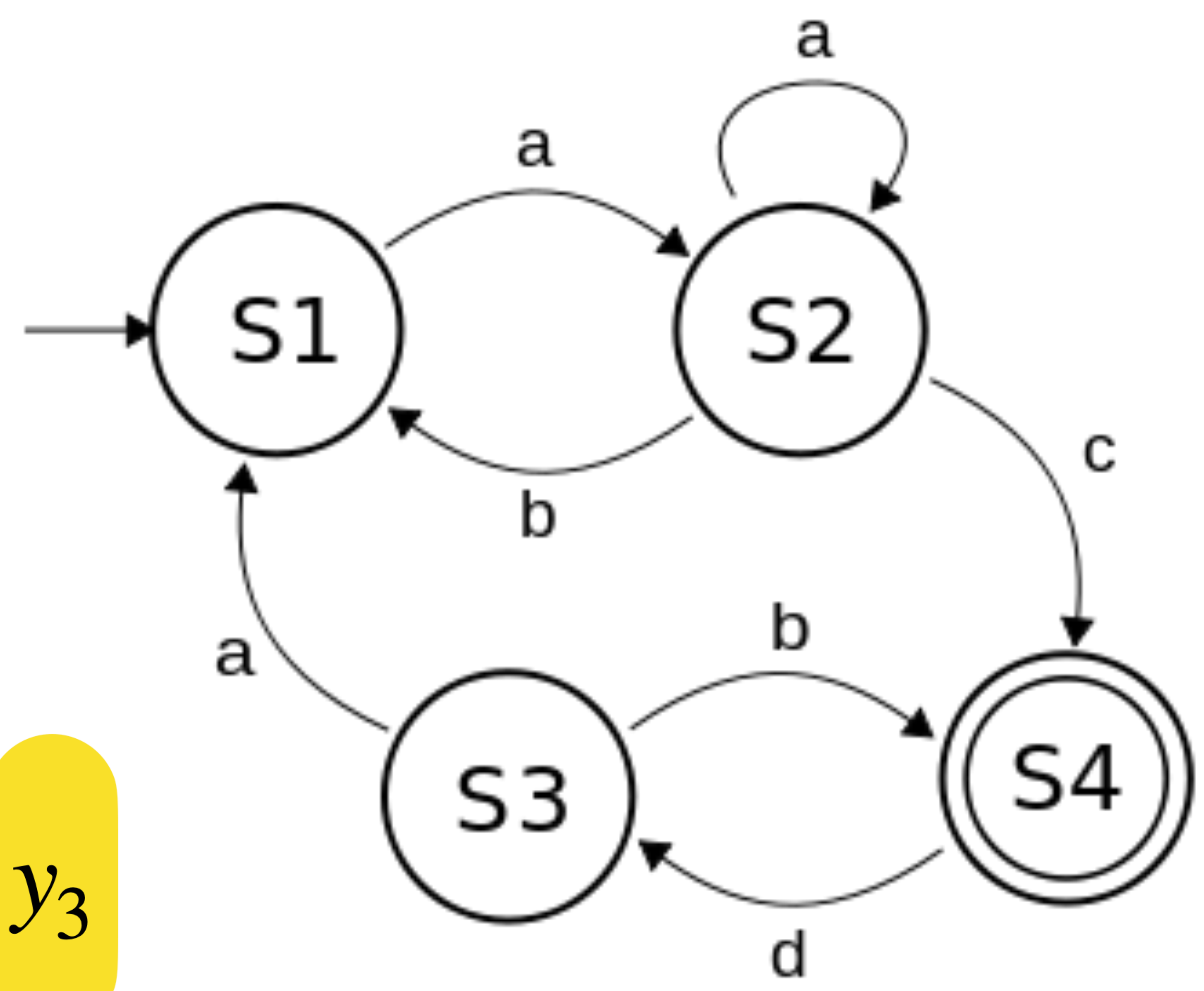
$h_t = f(h_{t-1}, x_t)$

$y_t = g(h_t)$

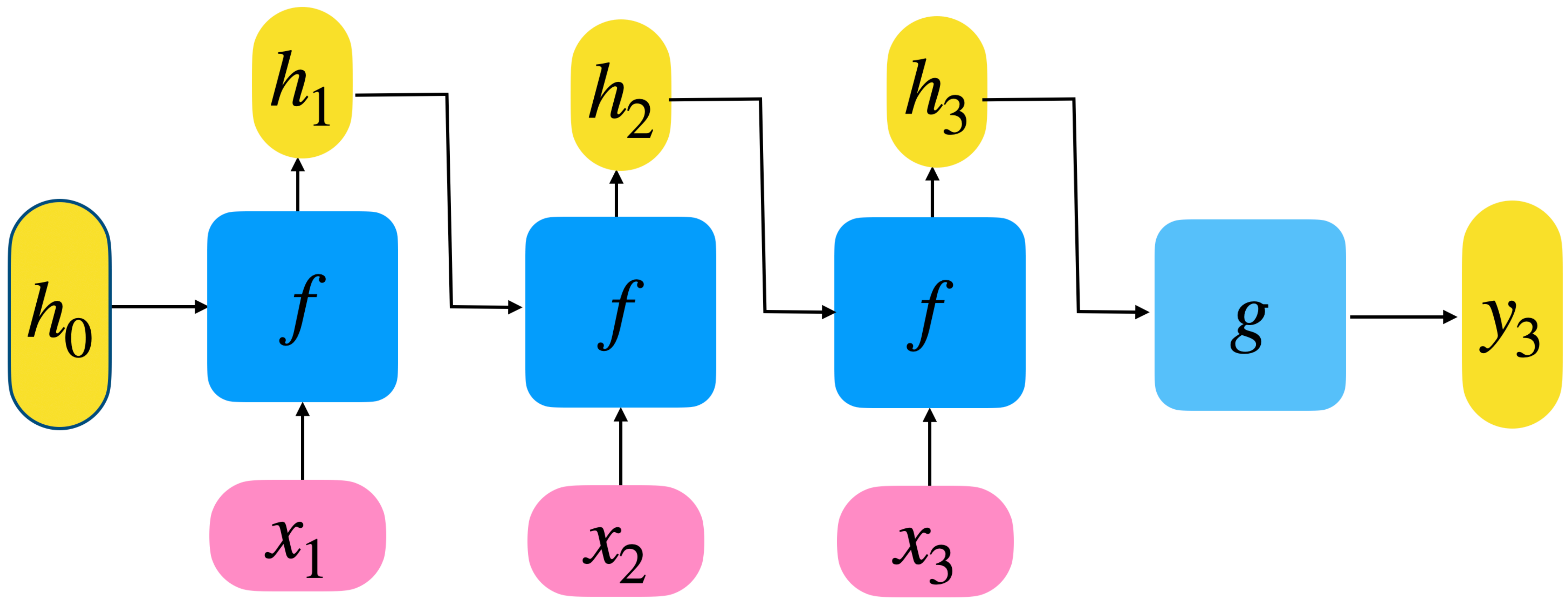
$h_t \in \mathbb{R}^d$ for some fixed dimension d

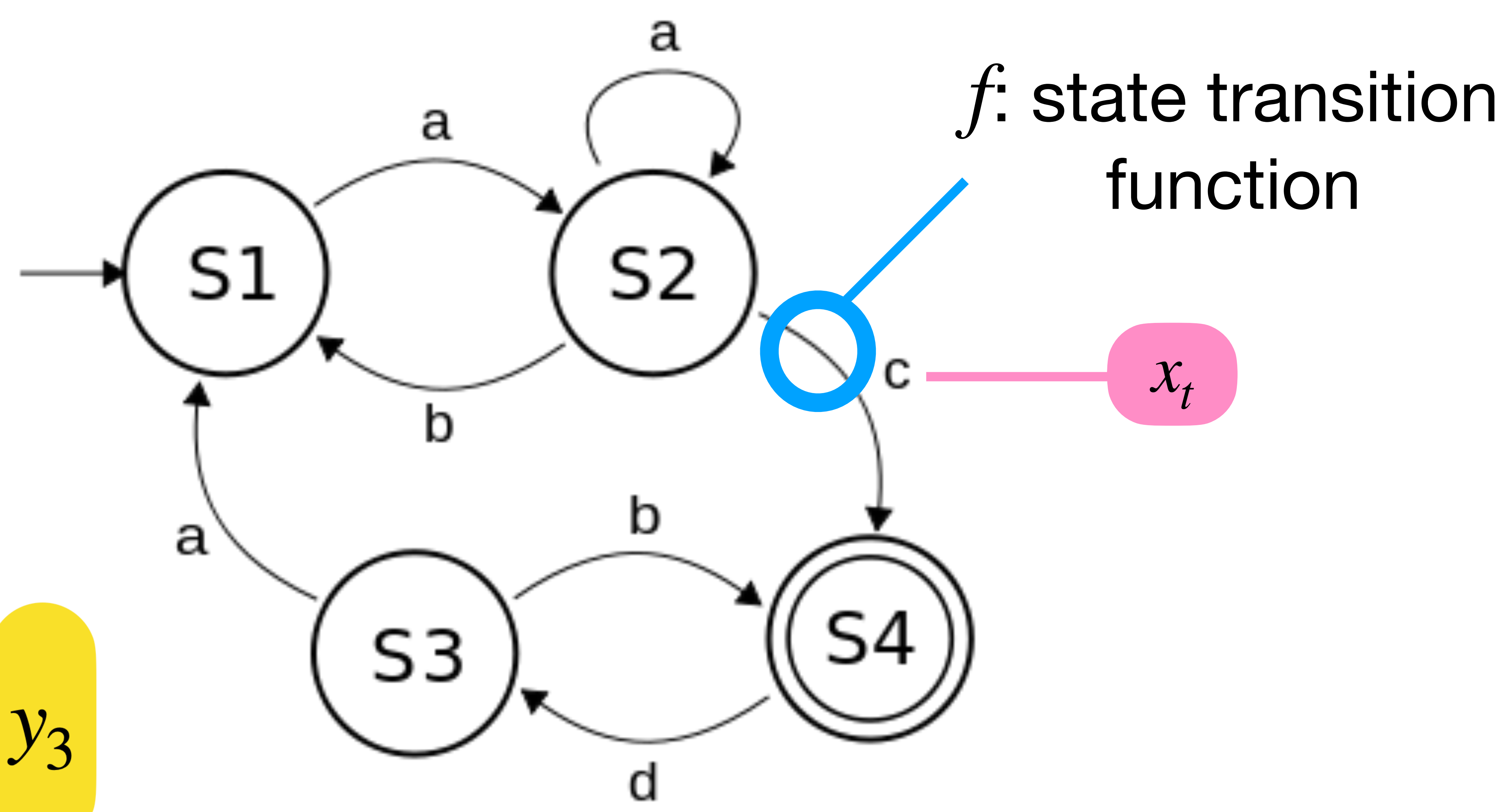
RNNs and State Machines



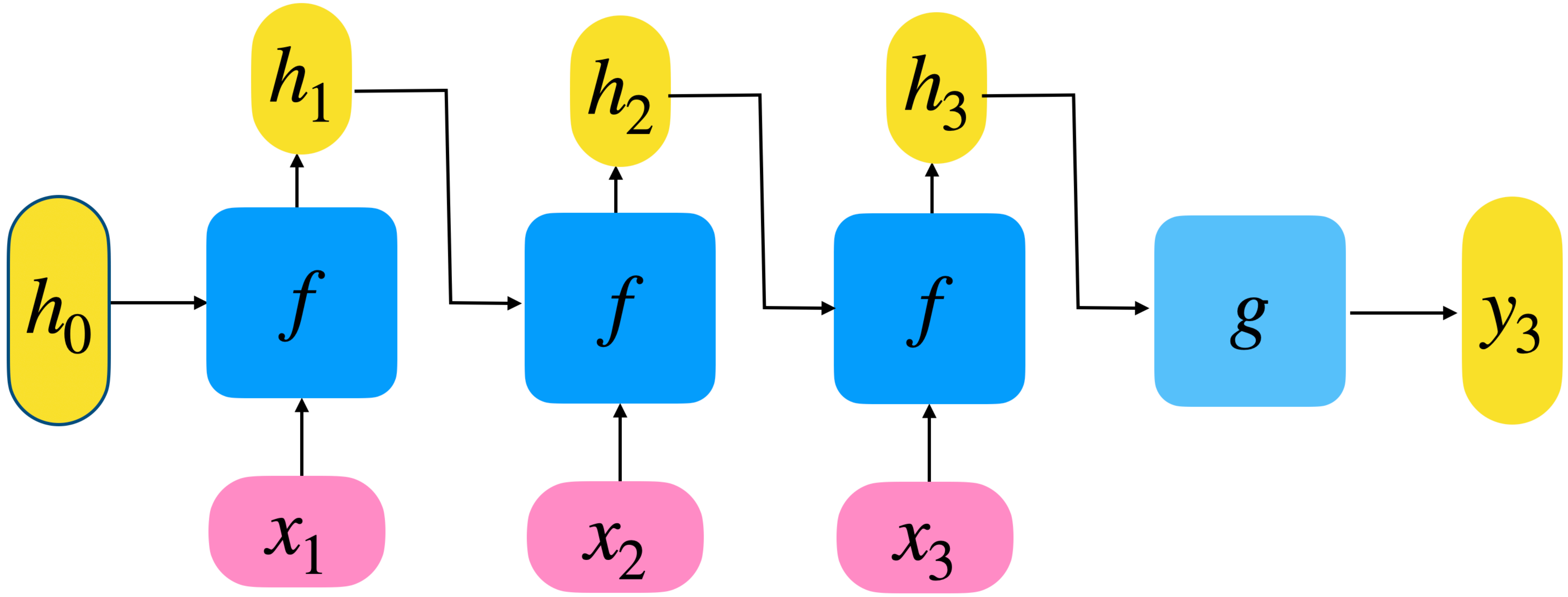


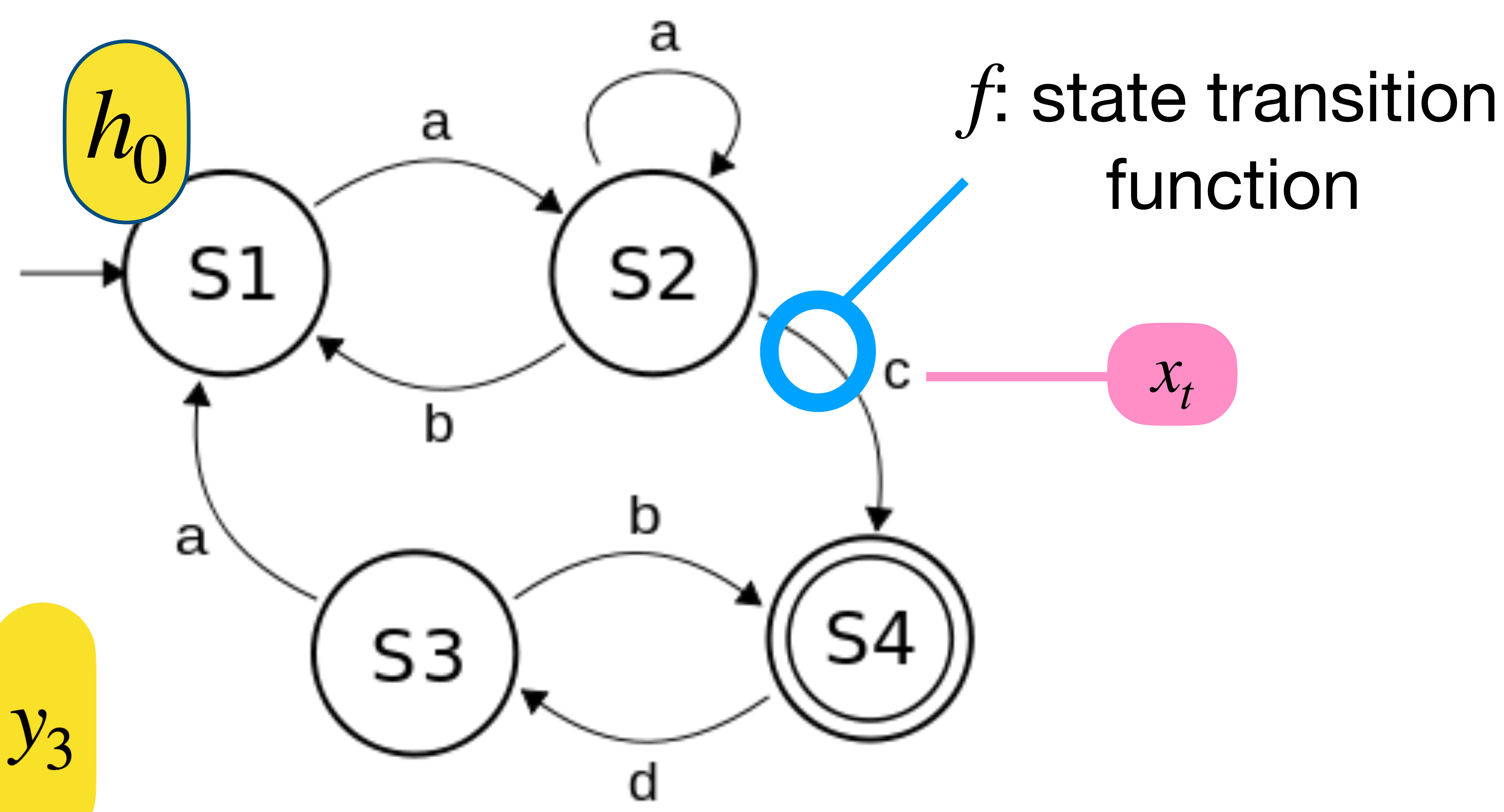
RNNs and State Machines



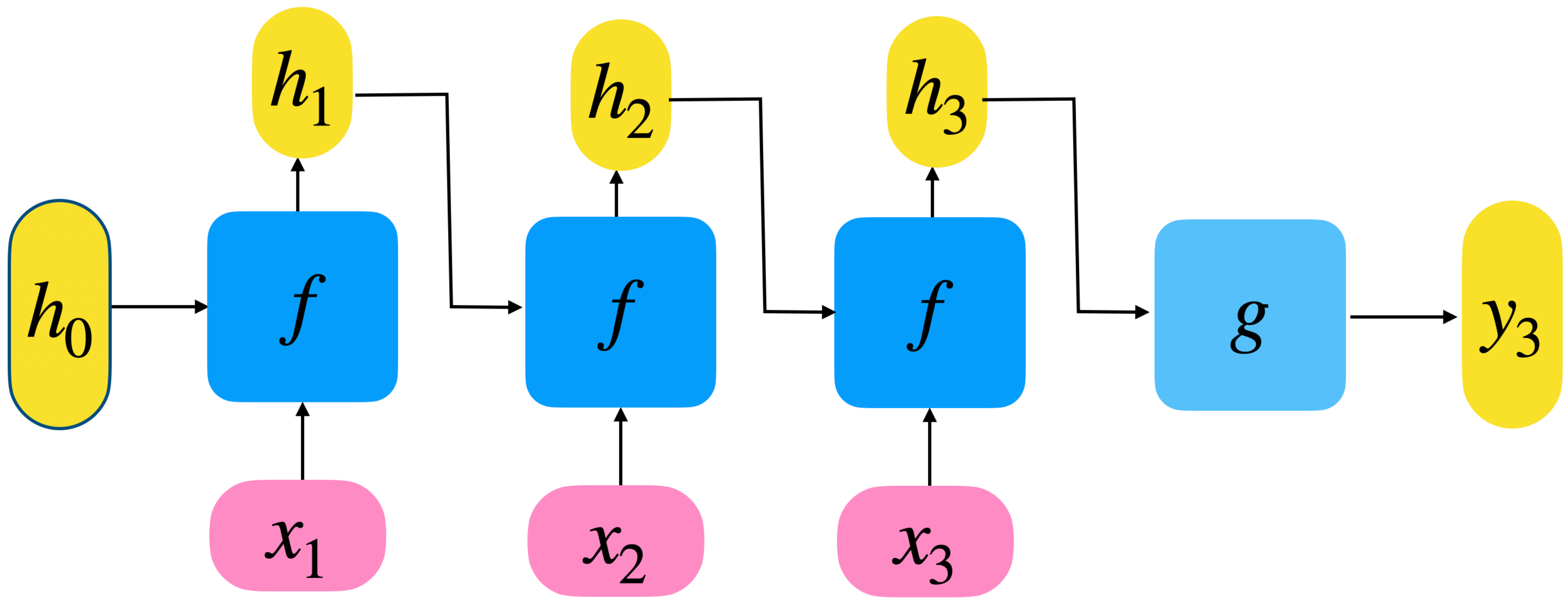


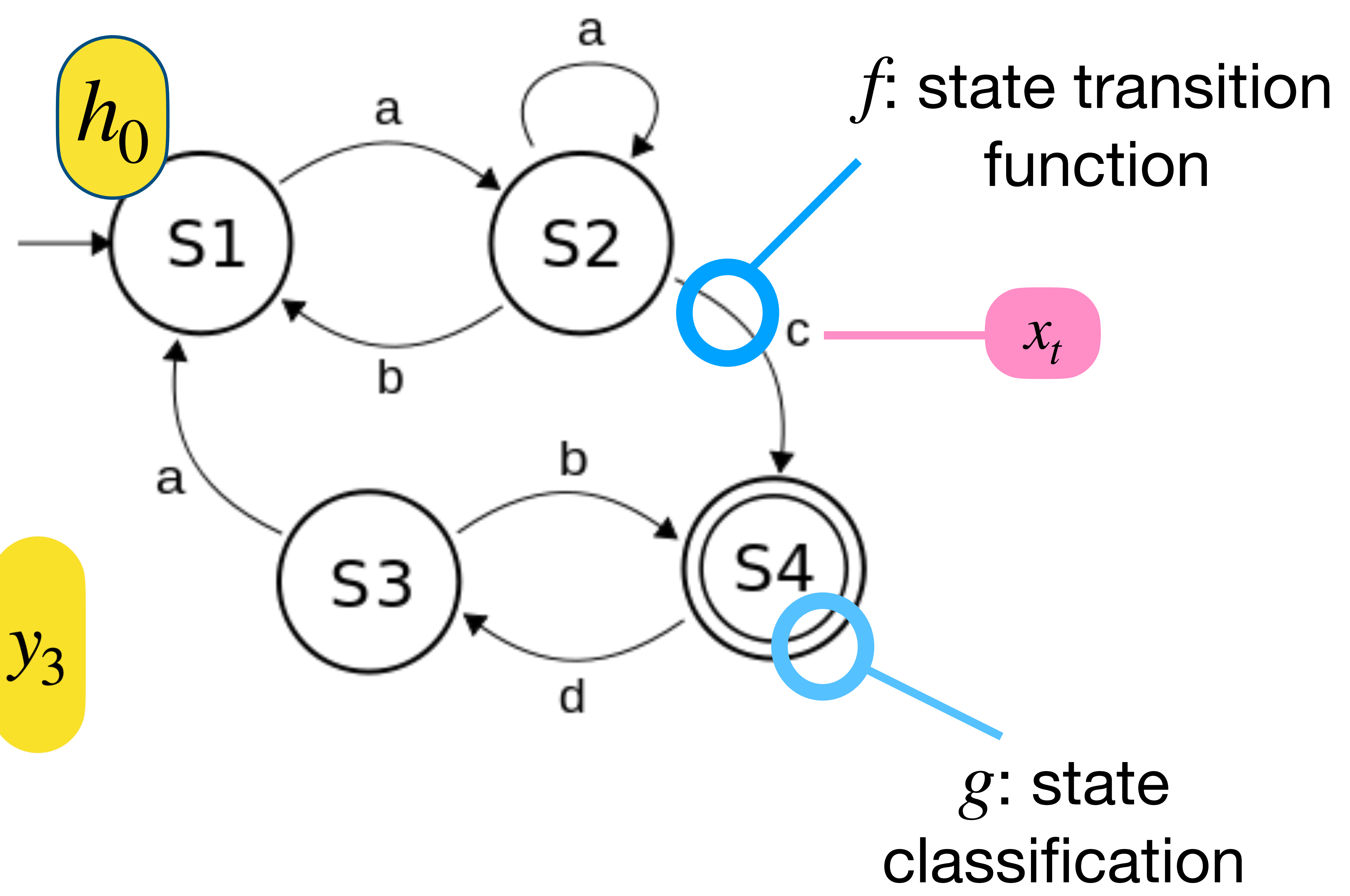
RNNs and State Machines





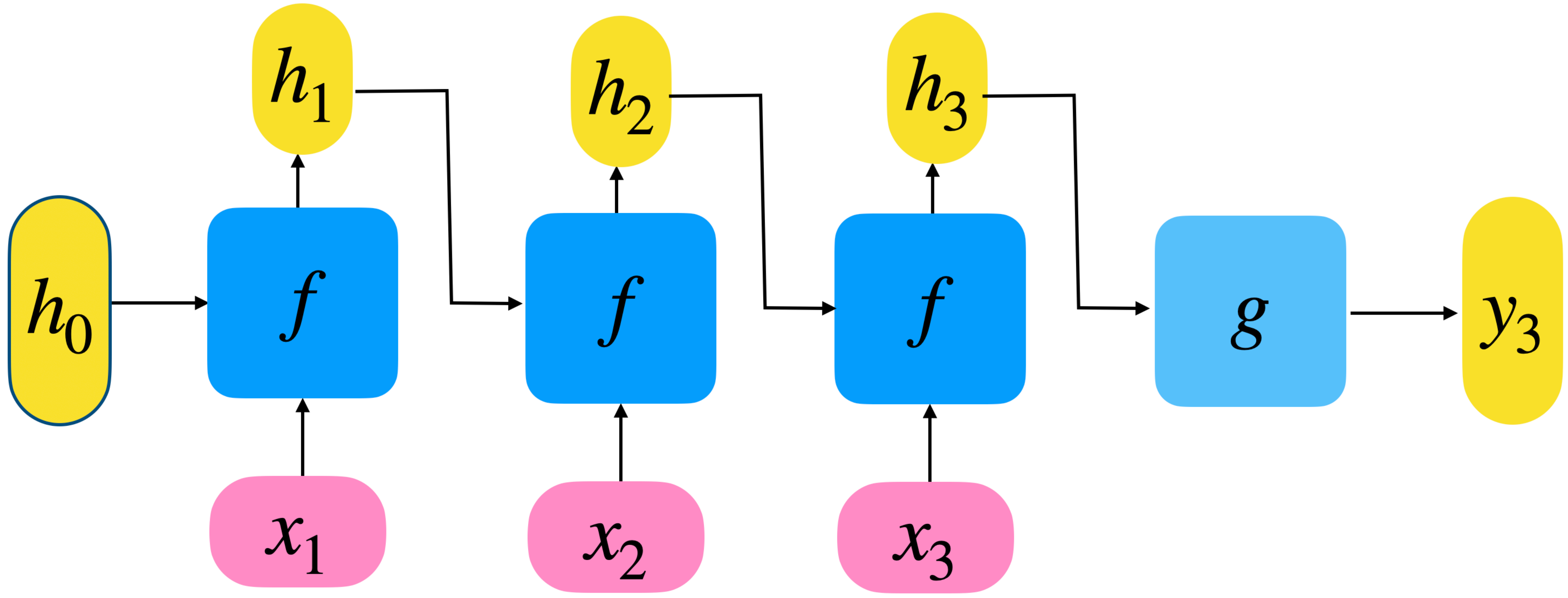
RNNs and State Machines





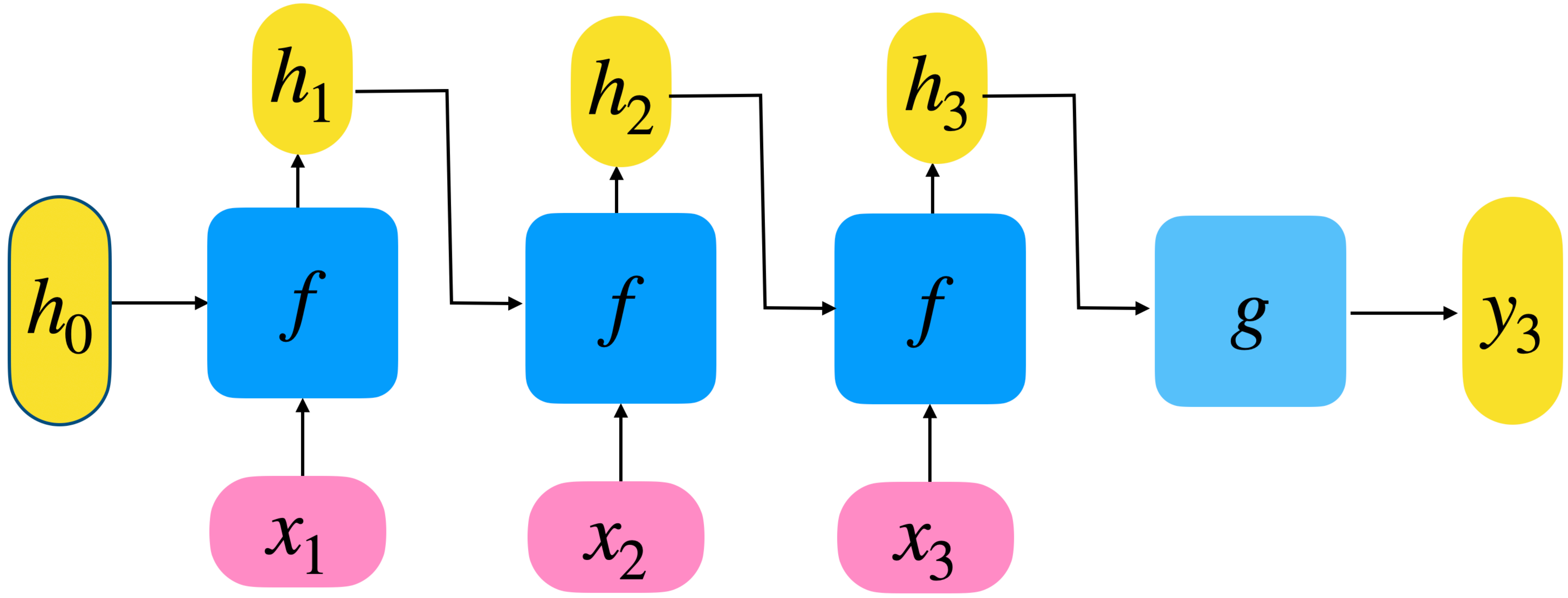
Several works use this parallel as inspiration to extract deterministic finite automata from trained RNNs!

RNNs and State Machines



Wait, are RNNs just deterministic finite automata? :/

What's in a state?



Wait, are RNNs just deterministic finite automata? :/

Let's assume $h_t \in \mathbb{Q}^d$ for some fixed d, and $f: \mathbb{Q}^d \rightarrow \mathbb{Q}^d$ perfectly precise

1. How big, potentially, is the set of reachable states?
2. How strong, potentially, is such an RNN?

High Level: Turing Completeness of RNNs

A simple RNN, with (1) rational states and weights and (2) clipped ReLU (piecewise linear sigmoid), **is Turing complete**. *Siegelmann and Sonntag, 1992*

$$h_{t+1} = \sigma(Wx_t + Uh_t + b)$$

$$\sigma(x) = \begin{cases} 0 & : x \leq 0 \\ x & : 0 < x < 1 \\ 1 & : 1 \leq x \end{cases}$$

High Level: Turing Completeness of RNNs

A simple RNN, with (1) rational states and weights and (2) clipped ReLU (piecewise linear sigmoid), **is Turing complete**. *Siegelmann and Sonntag, 1992*

$$h_{t+1} = \sigma(Wx_t + Uh_t + b)$$

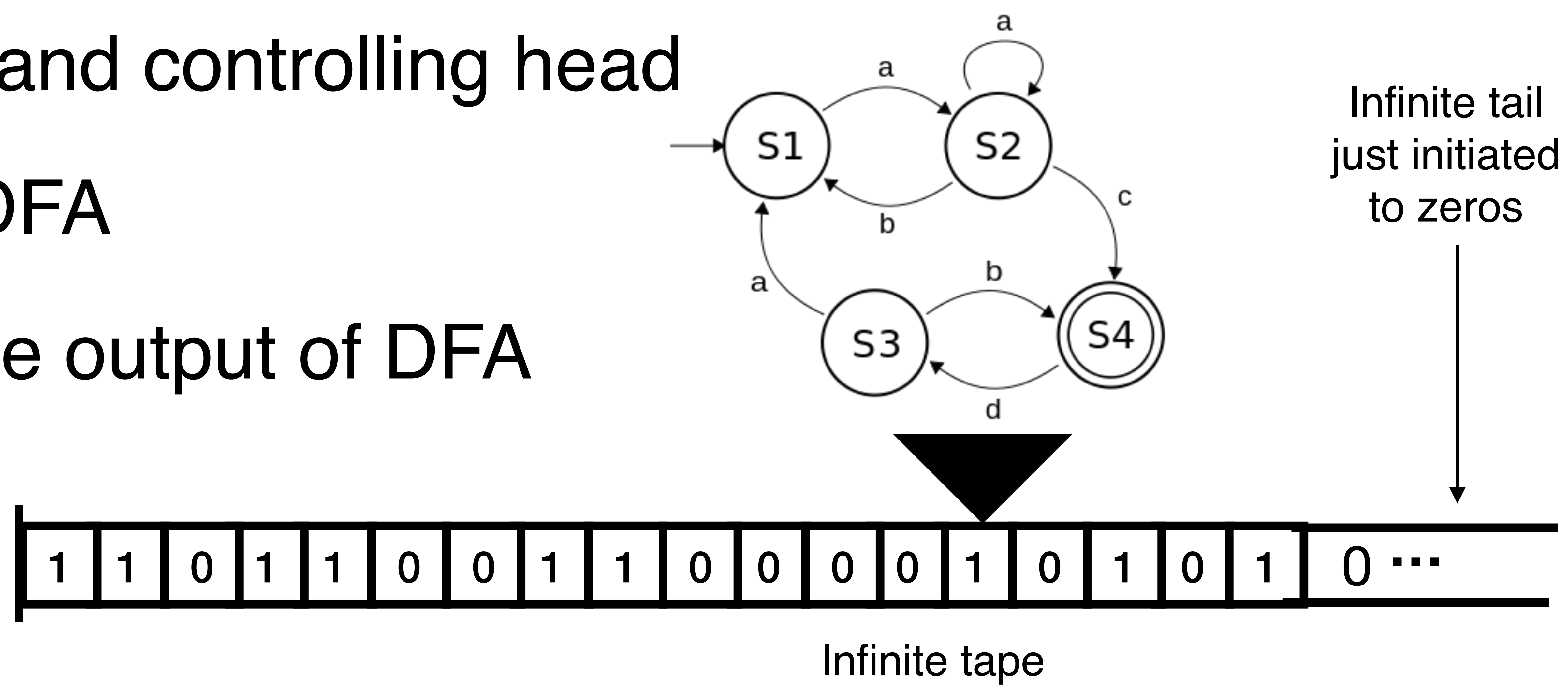
$$\sigma(x) = \begin{cases} 0 & : x \leq 0 \\ x & : 0 < x < 1 \\ 1 & : 1 \leq x \end{cases}$$

Theory time! Buckle up

High Level: Turing Completeness of RNNs

Recall: **Turing Machines** (**most basic version**)

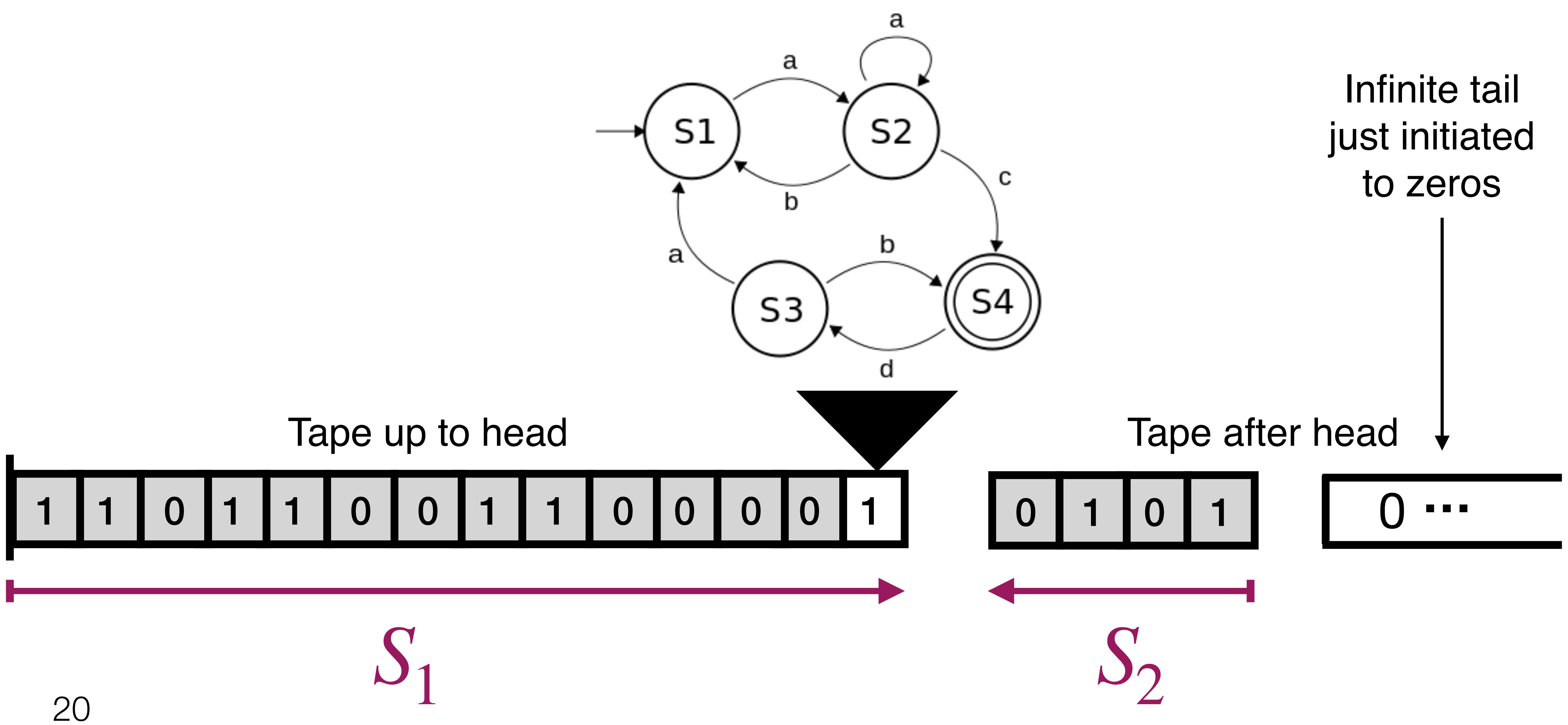
- Infinite tape (infinite in one direction), binary (for simplicity)
- Head moving on and reading from/writing to tape
- Controller DFA responding to tape and controlling head
    - Current head value is input to DFA
    - Head movements and writes are output of DFA



High Level: Turing Completeness of RNNs

The tape can be described with two stacks:

- One from the bottom of the tape, up to the head (S_1)
- One from the highest location written to in the tape, down to the head (S_2)

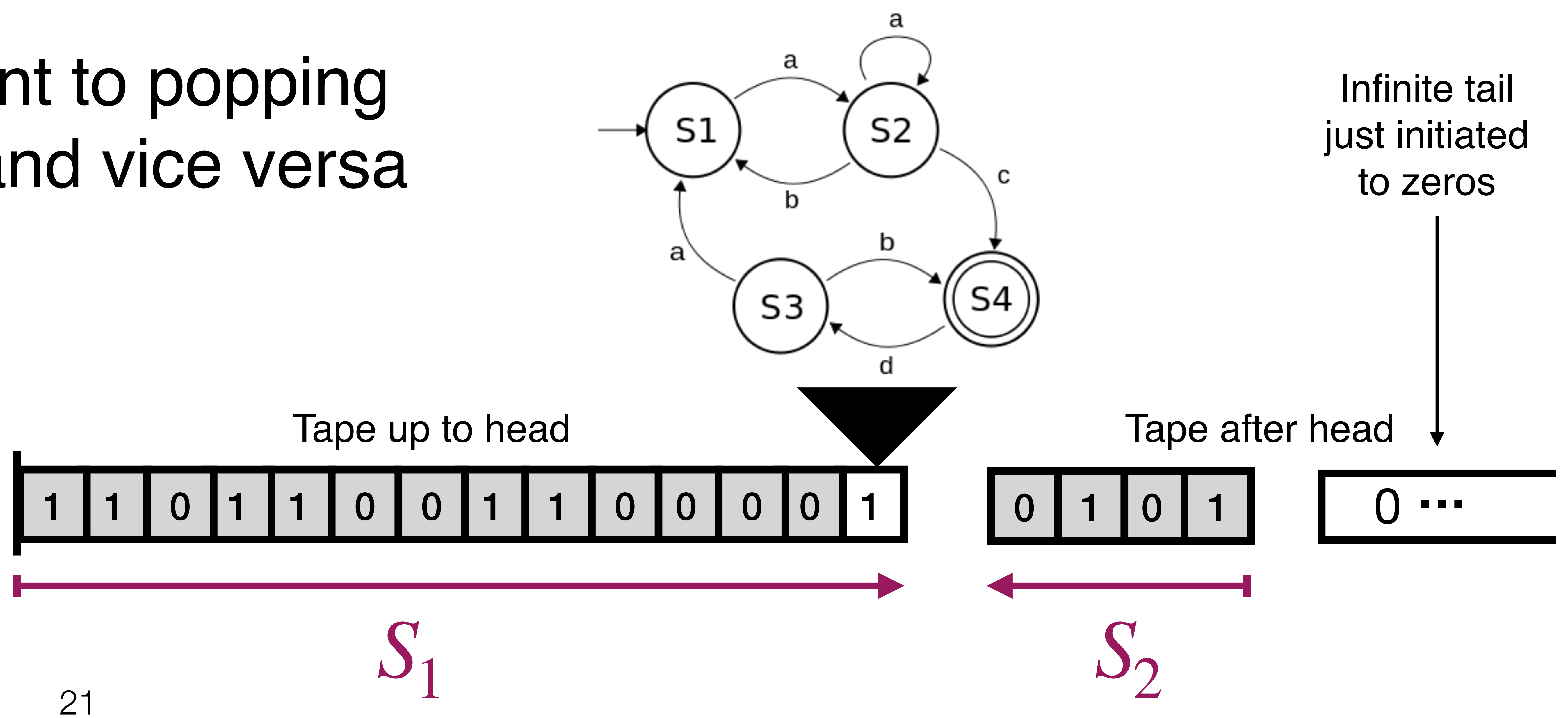


High Level: Turing Completeness of RNNs

The tape can be described with two stacks:

- One from the bottom of the tape, up to the head (S_1)
- One from the highest location written to in the tape, down to the head (S_2)

Moving left or right on the tape is equivalent to popping from one stack and pushing to the other, and vice versa

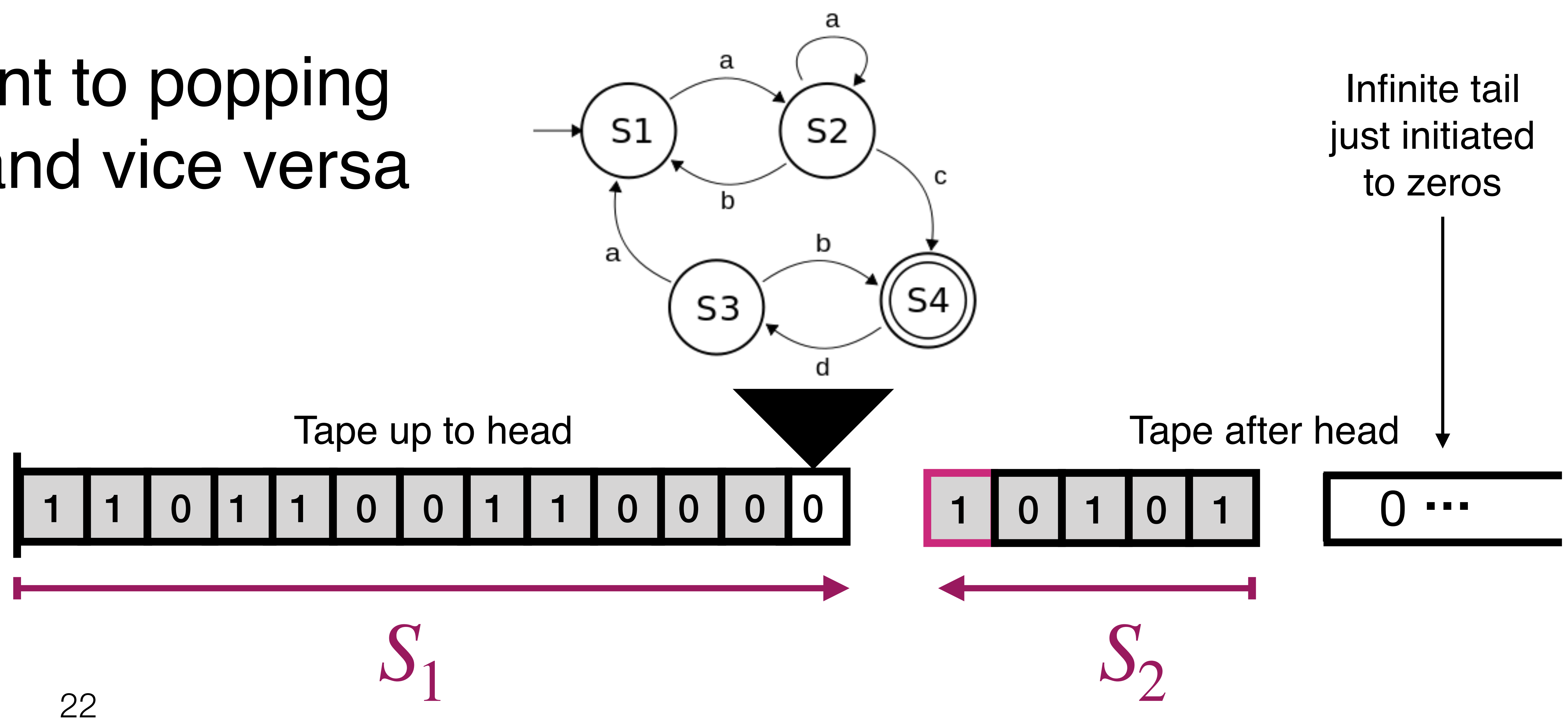


High Level: Turing Completeness of RNNs

The tape can be described with two stacks:

- One from the bottom of the tape, up to the head (S_1)
- One from the highest location written to in the tape, down to the head (S_2)

Moving left or right on the tape is equivalent to popping from one stack and pushing to the other, and vice versa

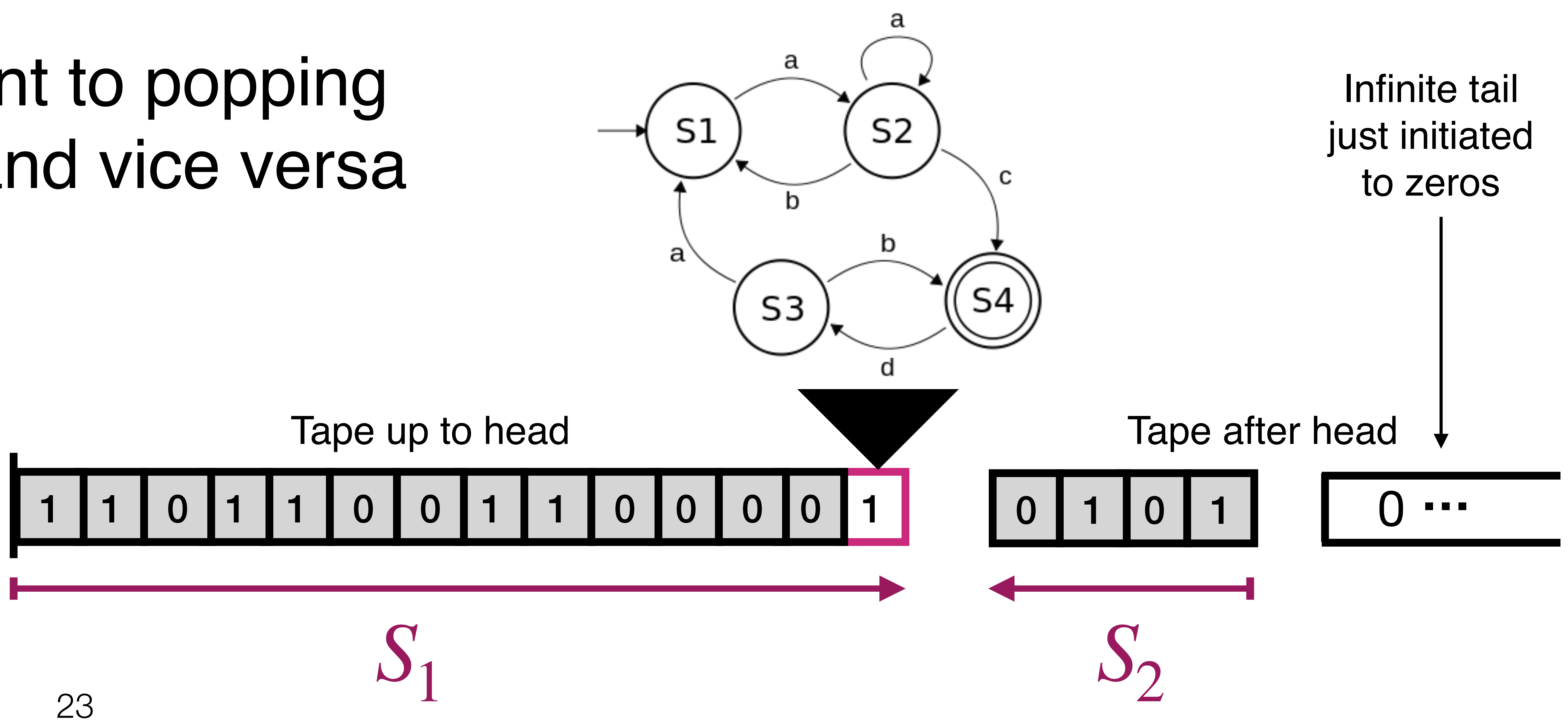


High Level: Turing Completeness of RNNs

The tape can be described with two stacks:

- One from the bottom of the tape, up to the head (S_1)
- One from the highest location written to in the tape, down to the head (S_2)

Moving left or right on the tape is equivalent to popping from one stack and pushing to the other, and vice versa

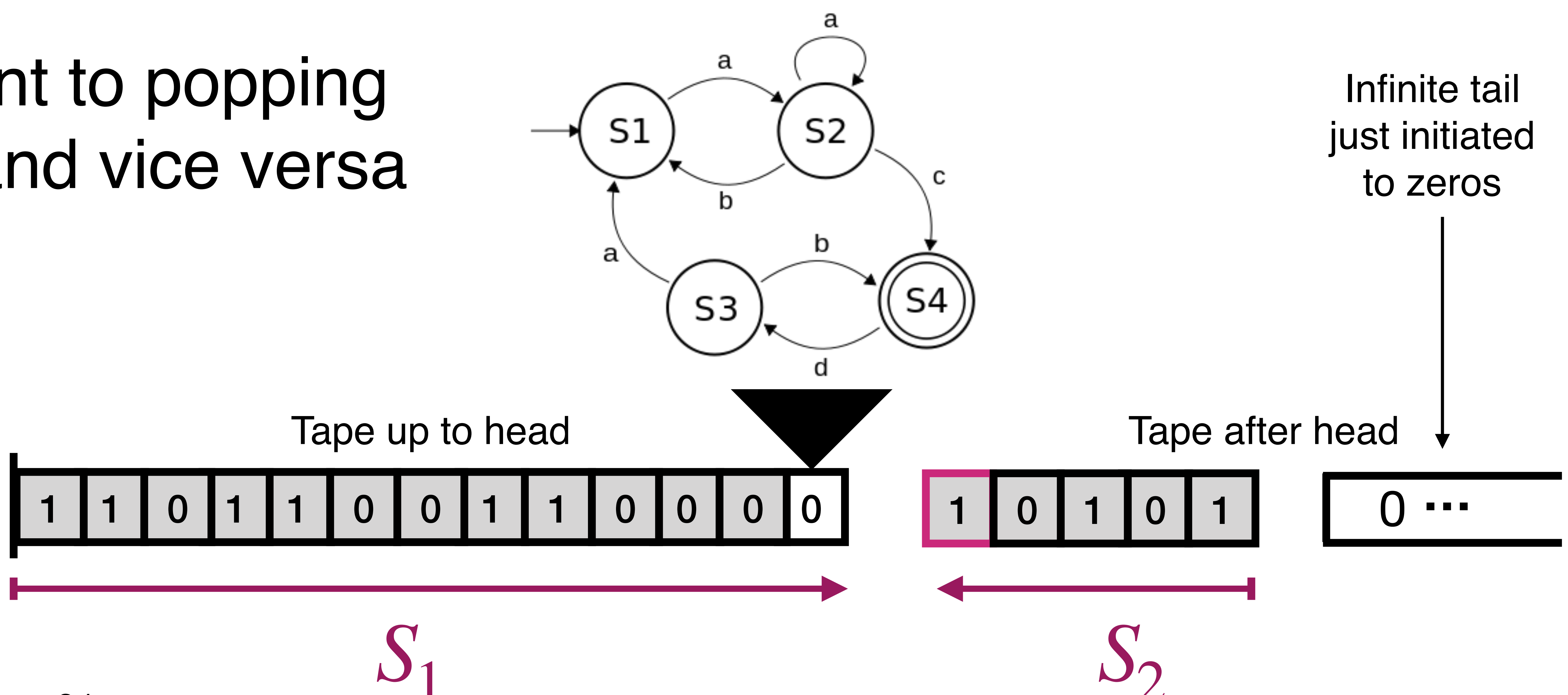


High Level: Turing Completeness of RNNs

The tape can be described with two stacks:

- One from the bottom of the tape, up to the head (S_1)
- One from the highest location written to in the tape, down to the head (S_2)

Moving left or right on the tape is equivalent to popping from one stack and pushing to the other, and vice versa

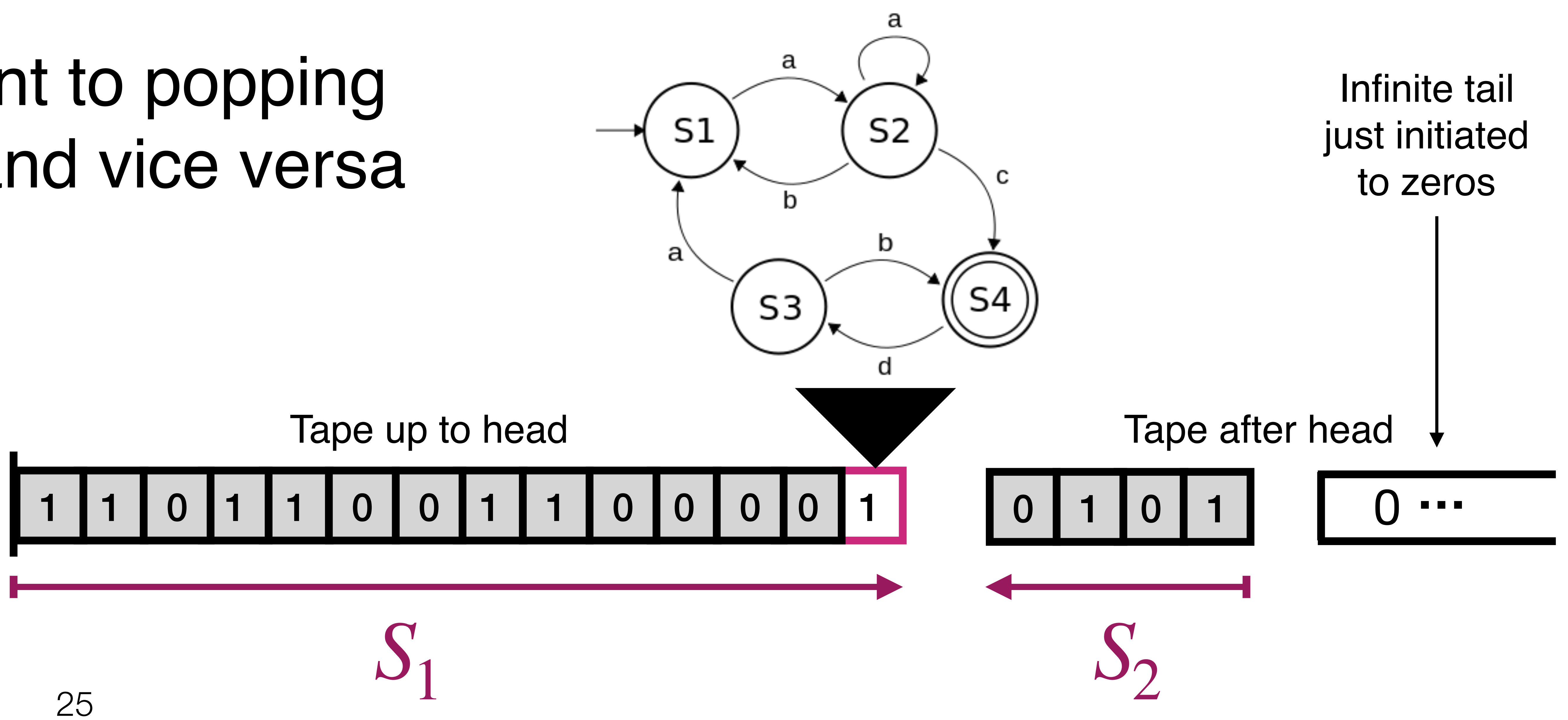


High Level: Turing Completeness of RNNs

The tape can be described with two stacks:

- One from the bottom of the tape, up to the head (S_1)
- One from the highest location written to in the tape, down to the head (S_2)

Moving left or right on the tape is equivalent to popping from one stack and pushing to the other, and vice versa



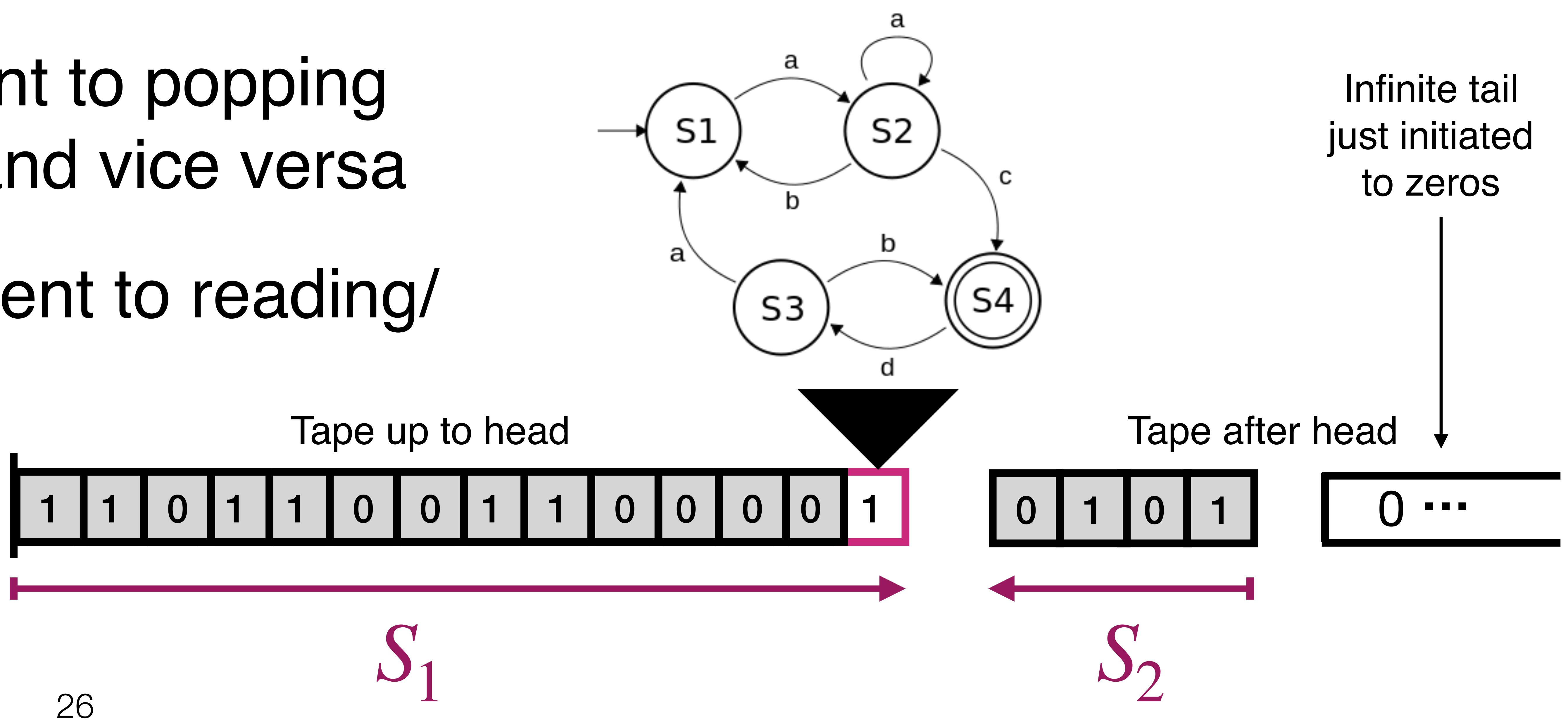
High Level: Turing Completeness of RNNs

The tape can be described with two stacks:

- One from the bottom of the tape, up to the head (S_1)
- One from the highest location written to in the tape, down to the head (S_2)

Moving left or right on the tape is equivalent to popping from one stack and pushing to the other, and vice versa

Reading from/writing to the tape is equivalent to reading/writing the top value in S_1



High Level: Turing Completeness of RNNs

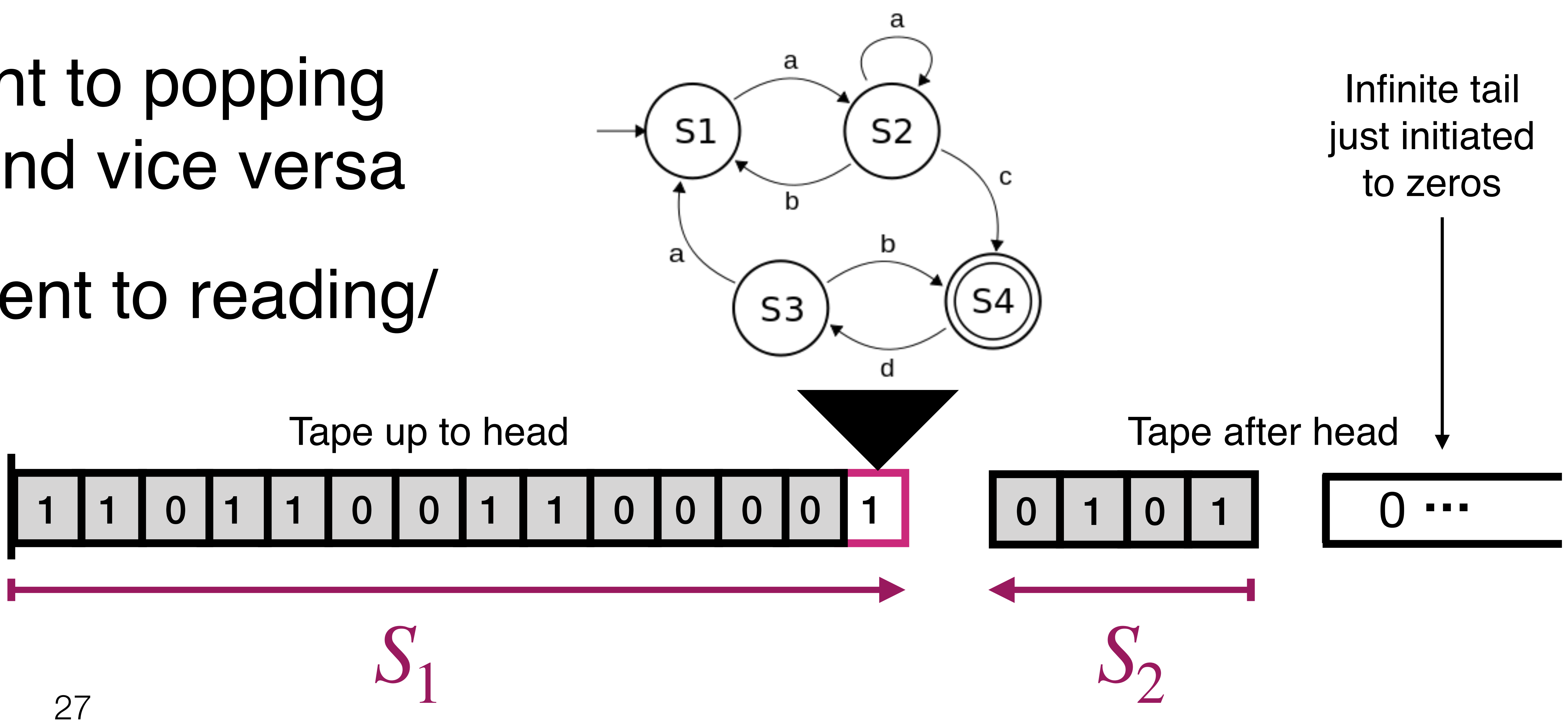
The tape can be described with two stacks:

- One from the bottom of the tape, up to the head (S_1)
- One from the highest location written to in the tape, down to the head (S_2)

Moving left or right on the tape is equivalent to popping from one stack and pushing to the other, and vice versa

Reading from/writing to the tape is equivalent to reading/ writing the top value in S_1

The Turing completeness proof hinges on simulating S_1 and S_2



High Level: Turing Completeness of RNNs

The stacks S_1 and S_2 can be encoded with a pair of fractions, q_1 and q_2

High Level: Turing Completeness of RNNs

The stacks S_1 and S_2 can be encoded with a pair of fractions, q_1 and q_2

- Both stacks are initially empty, and for this q_1 and q_2 are initiated to 0

High Level: Turing Completeness of RNNs

The stacks S_1 and S_2 can be encoded with a pair of fractions, q_1 and q_2

- Both stacks are initially empty, and for this q_1 and q_2 are initiated to 0
- To push $z \in \{0,1\}$ into q_i, we compute: $q_i' = \frac{q_i}{4} + \frac{z}{2} + \frac{1}{4}$.

High Level: Turing Completeness of RNNs

The stacks S_1 and S_2 can be encoded with a pair of fractions, $q_1, q_2 \in [0,1)$

- Both stacks are initially empty, and for this q_1 and q_2 are initiated to 0
- To push $z \in \{0,1\}$ into q_i, we compute: $q_i' = \frac{q_i}{4} + \frac{z}{2} + \frac{1}{4}$.
- To recover the top value z from non-empty q_i, we compute $z = \sigma(4q_i - 2)$

$$= \sigma(q_i^{prev} + 2z - 1)$$

High Level: Turing Completeness of RNNs

The stacks S_1 and S_2 can be encoded with a pair of fractions, q_1 and q_2

- Both stacks are initially empty, and for this q_1 and q_2 are initiated to 0
- To push $z \in \{0,1\}$ into q_i, we compute: $q_i' = \dfrac{q_i}{4} + \dfrac{z}{2} + \dfrac{1}{4}$.
- To recover the top value z from non-empty q_i, we compute $z = \sigma(4q_i - 2)$
- To pop the (recovered) top value z from q_i, we compute $q_i' = 4q_i - 2z - 1$

High Level: Turing Completeness of RNNs

The stacks S_1 and S_2 can be encoded with a pair of fractions, q_1 and q_2

- Both stacks are initially empty, and for this q_1 and q_2 are initiated to 0
- To push $z \in \{0,1\}$ into q_i, we compute: $q_i' = \frac{q_i}{4} + \frac{z}{2} + \frac{1}{4}$.

Why add $1/4$? And why compute $q_i/4$ instead of $q_i/2$?

- To recover the top value z from non-empty q_i, we compute $z = \sigma(4q_i - 2)$
- To pop the (recovered) top value z from q_i, we compute $q_i' = 4q_i - 2z - 1$

High Level: Turing Completeness of RNNs

We can simulate an entire Turing machine in our RNN by:

1. Using specific dimensions in the state h_t for:
    1. The stacks, q_1 and q_2
    2. The current head values, z, and value to be written, z', and
    3. The controller states
2. Padding the input, to allow the RNN time at each ‘actual’ input token to compute the full Turing machine update (read, decide updates, write, move)

High Level: Turing Completeness of RNNs

So RNNs are Turing complete!

But we also just said they were plain state machines…

(Well, we didn't say they were *finite* state machines)

Still, where's the truth?

High Level: Turing Completeness of RNNs

Do you see any practical problems with the stack fractions q_1 and q_2 ?

$$q_i' = \frac{q_i}{4} + \frac{z}{2} + \frac{1}{4}$$

push z to S_i

$$q_i' = 4q_i - 2z - 1$$

pop z from S_i

$$z = \sigma(4q_i - 2)$$

read from top of S_i

High Level: Turing Completeness of RNNs

Do you see any practical problems with the stack fractions q_1 and q_2 ?

$$q_i' = \frac{q_i}{4} + \frac{z}{2} + \frac{1}{4}$$

push z to S_i

$$q_i' = 4q_i - 2z - 1$$

pop z from S_i

$$z = \sigma(4q_i - 2)$$

read from top of S_i

Do we normally provide RNNs with heavily padded input?

Do we normally allow RNNs arbitrary additional time to 'finish' a computation before we read the output?

High Level: Turing Completeness of RNNs

Do you see any practical problems with the stack fractions q_1 and q_2 ?

$$q_i' = \frac{q_i}{4} + \frac{z}{2} + \frac{1}{4}$$

push z to S_i

$$q_i' = 4q_i - 2z - 1$$

pop z from S_i

$$z = \sigma(4q_i - 2)$$

read from top of S_i

Do we normally provid

Do we normally allow RN
computation

This result does not say much about how we use RNNs ***in practice*** *(finite precision floating point calculations, bounded inference time)*

Practical Power of RNNs

Instead of checking their maximum potential, let's consider what different RNN mechanisms are really set up for…

1. Common RNN Architectures

GRU

LSTM

Teaser: these are not the same…

1. Common RNN Architectures

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

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

1. Common RNN Architectures

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

1. Common RNN Architectures

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

1. Common RNN Architectures

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

$f_t \in (0,1)$

$i_t \in (0,1)$

$o_t \in (0,1)$

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

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

$h_t = o_t \circ g(c_t)$

gates

candidate vectors

update functions

1. Common RNN Architectures

GRU

$z_t \in (0,1)$

$r_t \in (0,1)$

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

Bounded!

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

Interpolation

LSTM

$f_t \in (0,1)$

$i_t \in (0,1)$

$o_t \in (0,1)$

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

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

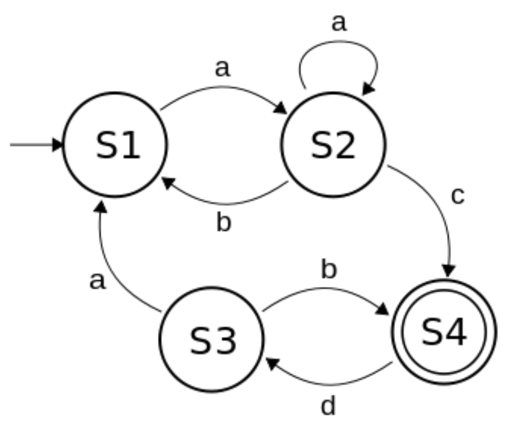
$h_t = o_t \circ g(c_t)$

1. Common RNN Architectures

GRU

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

Bounded!



Interpolation

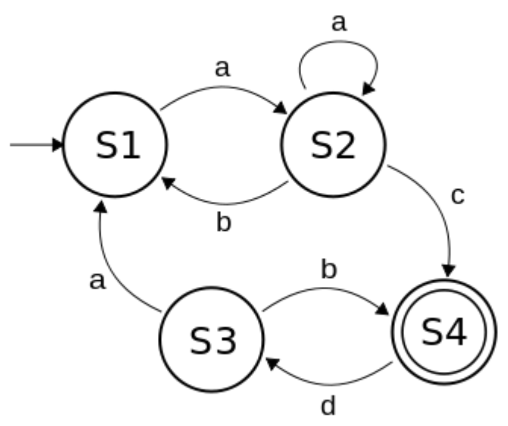
LSTM

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

1. Common RNN Architectures

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…**

$o_t \in (0,1)$

$\tilde{c}_t \in (-1,1)$ **subtracting/adding 1**

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

1. Common RNN Architectures

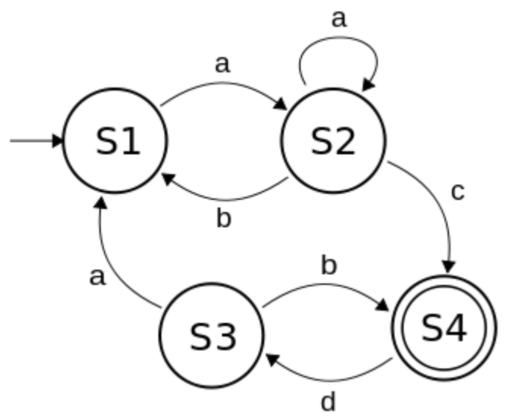
GRU

$z_t \in (0,1)$

$r_t \in (0,1)$

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

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



LSTM

$f_t \in (0,1)$ **reset/keep, then…**

$i_t \in (0,1)$ **stay/step, by…**

$o_t \in (0,1)$

$\tilde{c}_t \in (-1,1)$ **subtracting/adding 1**

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

$h_t = o_t \circ g(c_t)$

Counts!

1. Common RNN Architectures

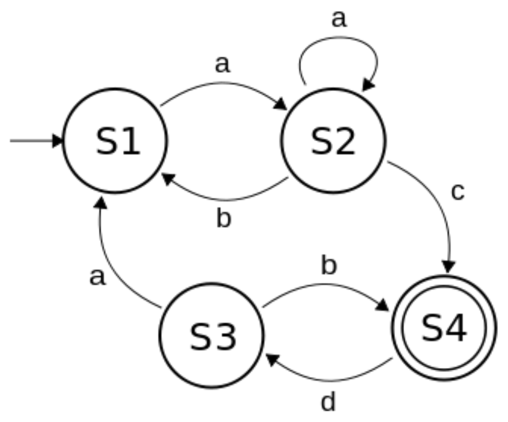
GRU

$z_t \in (0,1)$

$r_t \in (0,1)$

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

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



LSTM

$f_t \in (0,1)$ **reset/keep, then…**

$i_t \in (0,1)$ **stay/step, by…**

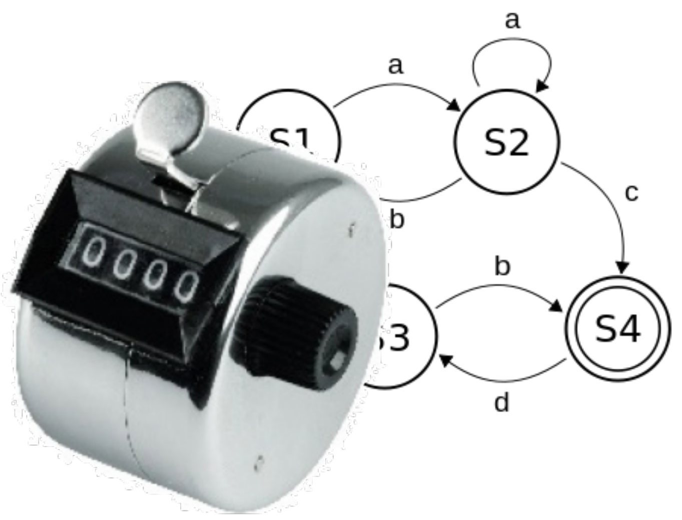
$o_t \in (0,1)$

$\tilde{c}_t \in (-1,1)$ **subtracting/adding 1**

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

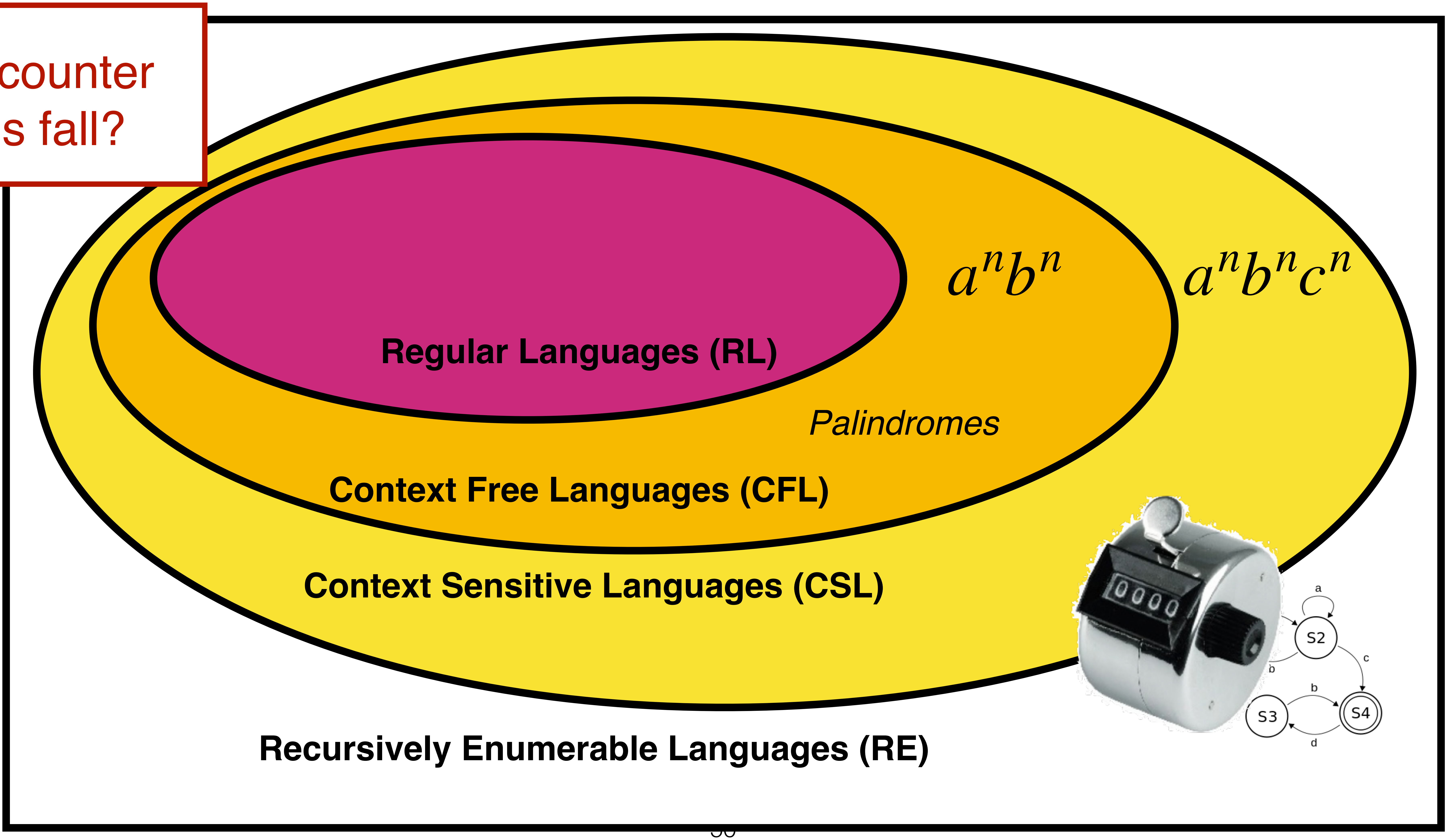
$h_t = o_t \circ g(c_t)$

Counts!



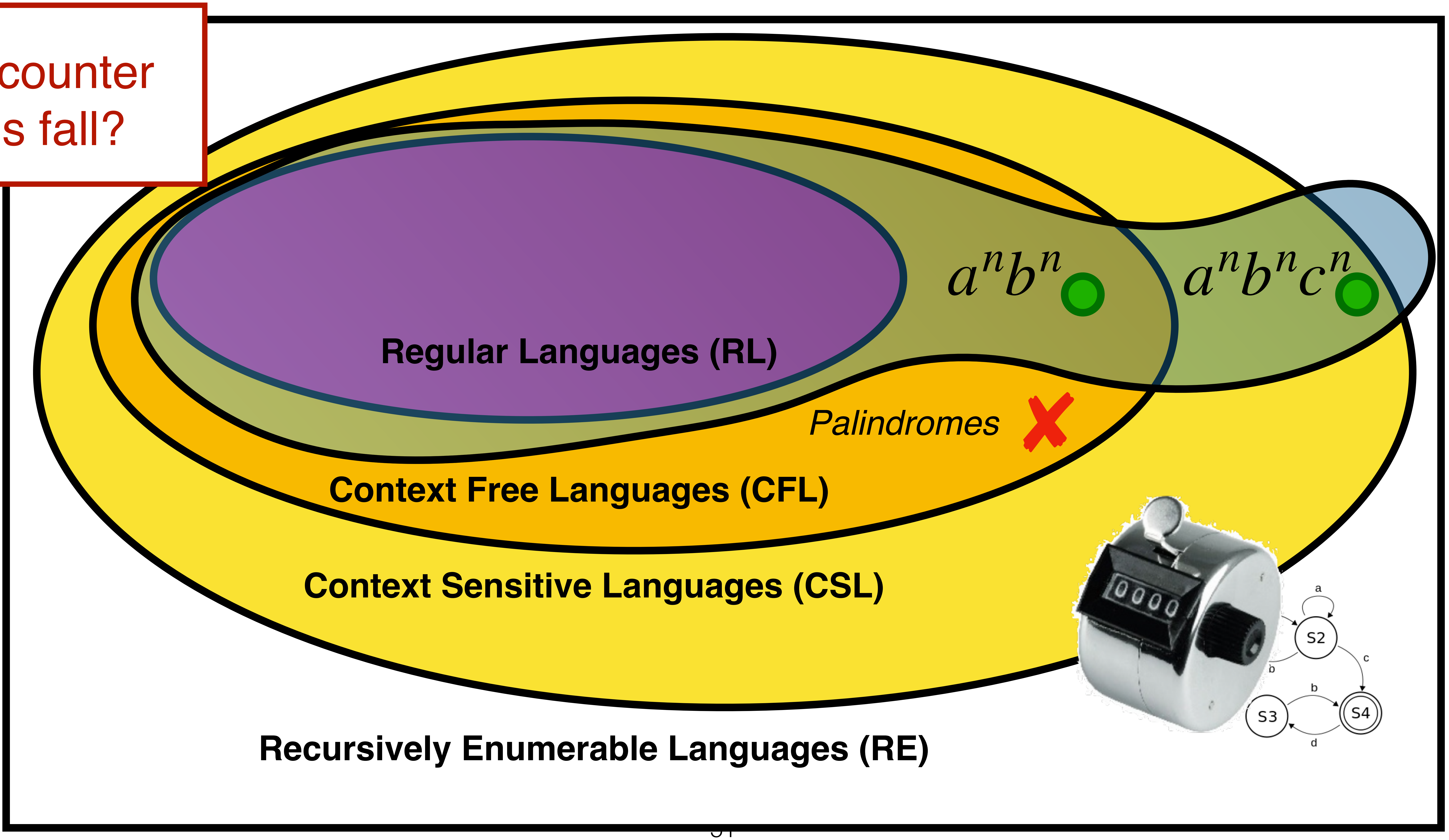
Aside: The Chomsky Hierarchy and Counting

Where do counter languages fall?



Aside: The Chomsky Hierarchy and Counting

Where do counter languages fall?



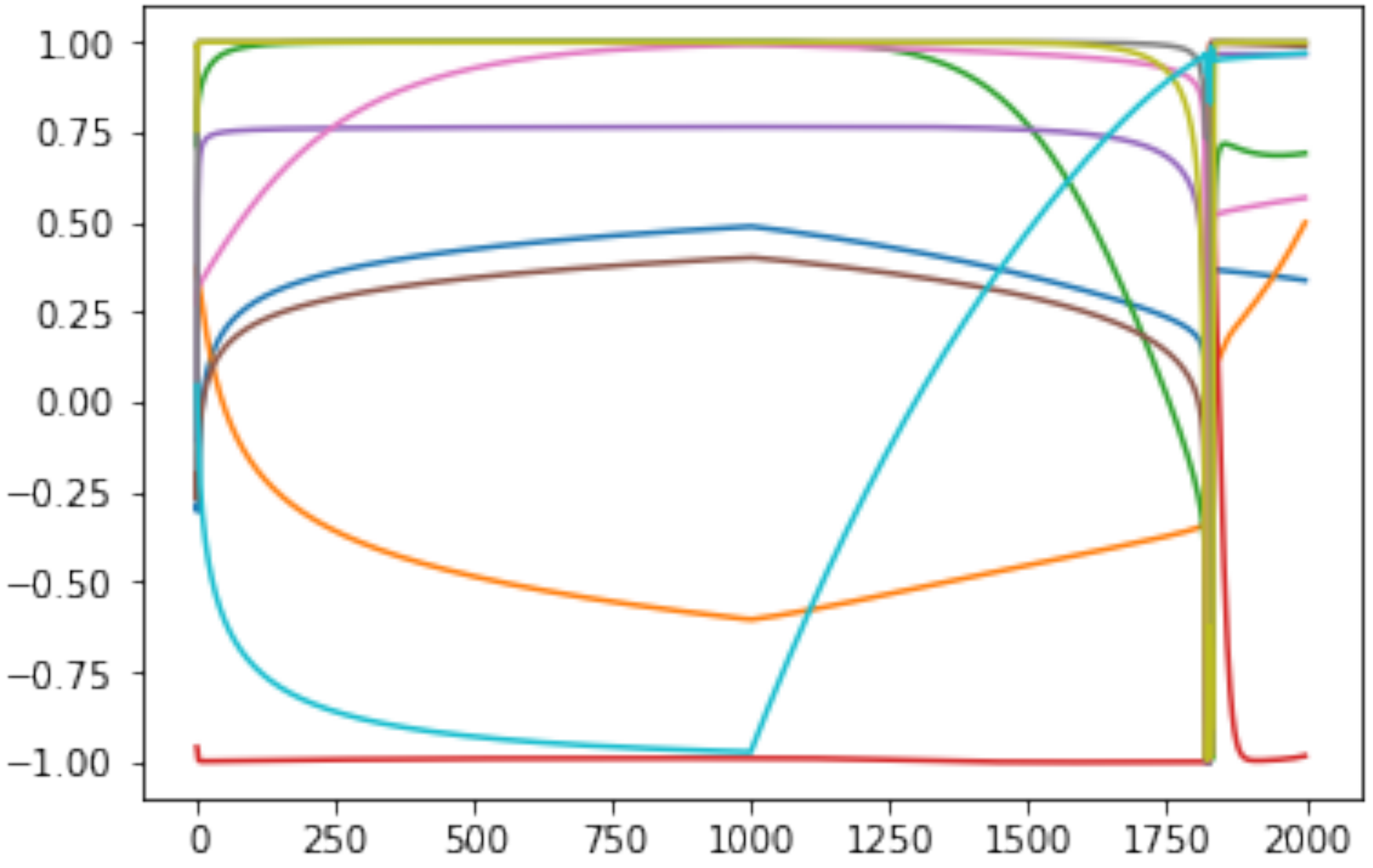
1. Common RNN Architectures

What will happen if we train GRUs and LSTMs on counting-based tasks, like $a^n b^n$ and $a^n b^n c^n$?

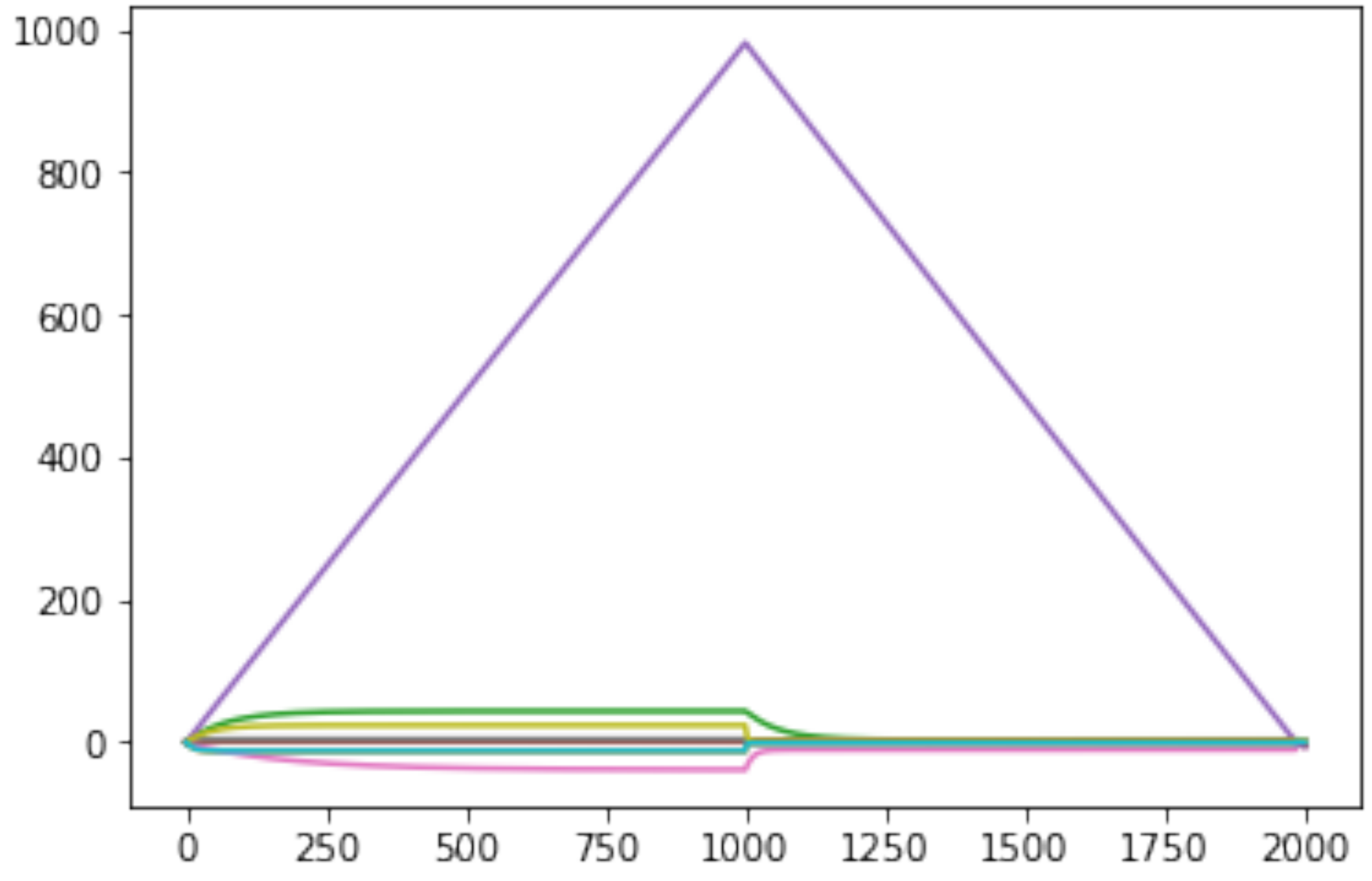
What kind of counters do these tasks need?

1. Common RNN Architectures

GRU



LSTM



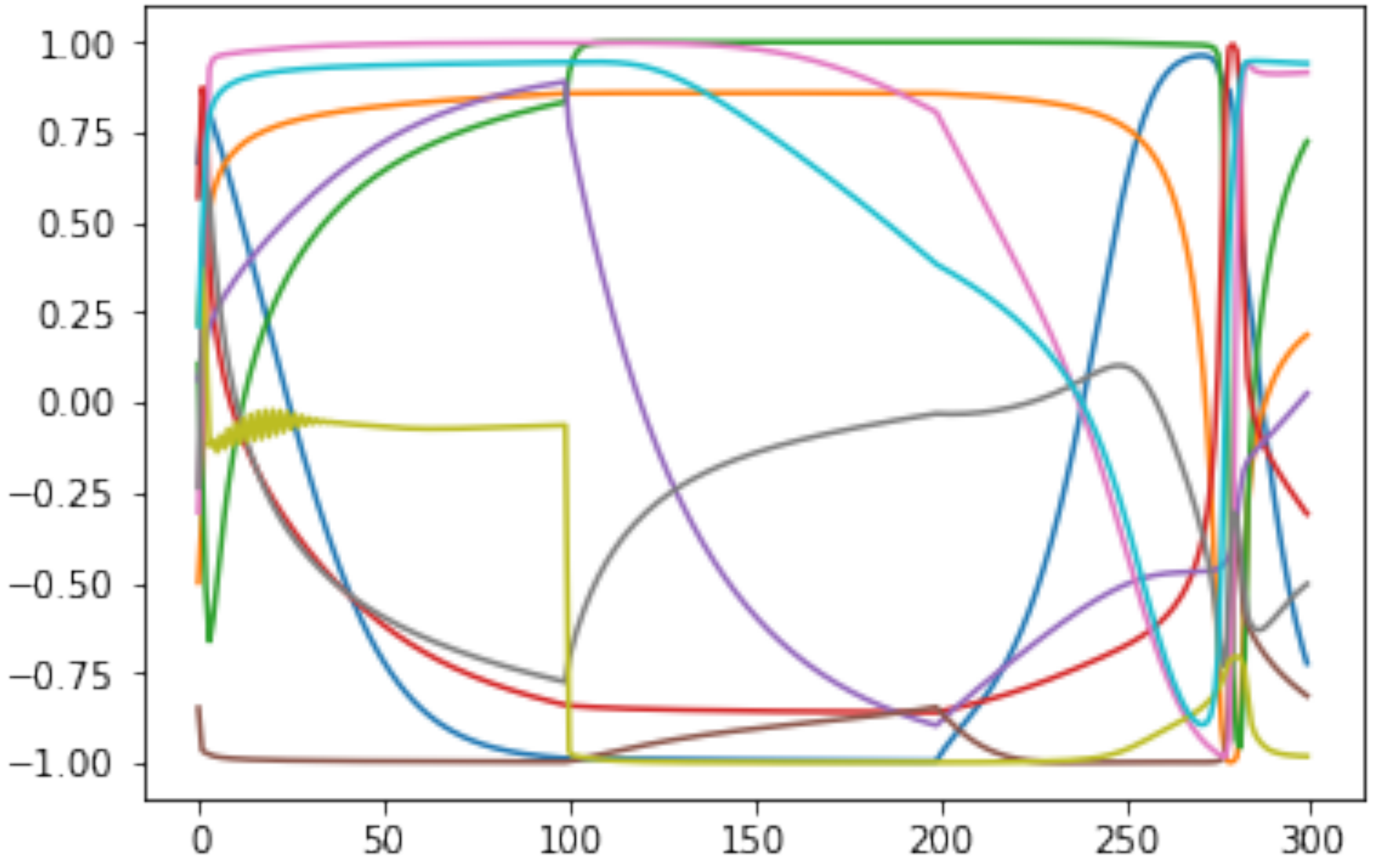
Activations on $a^{1000}b^{1000}$

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

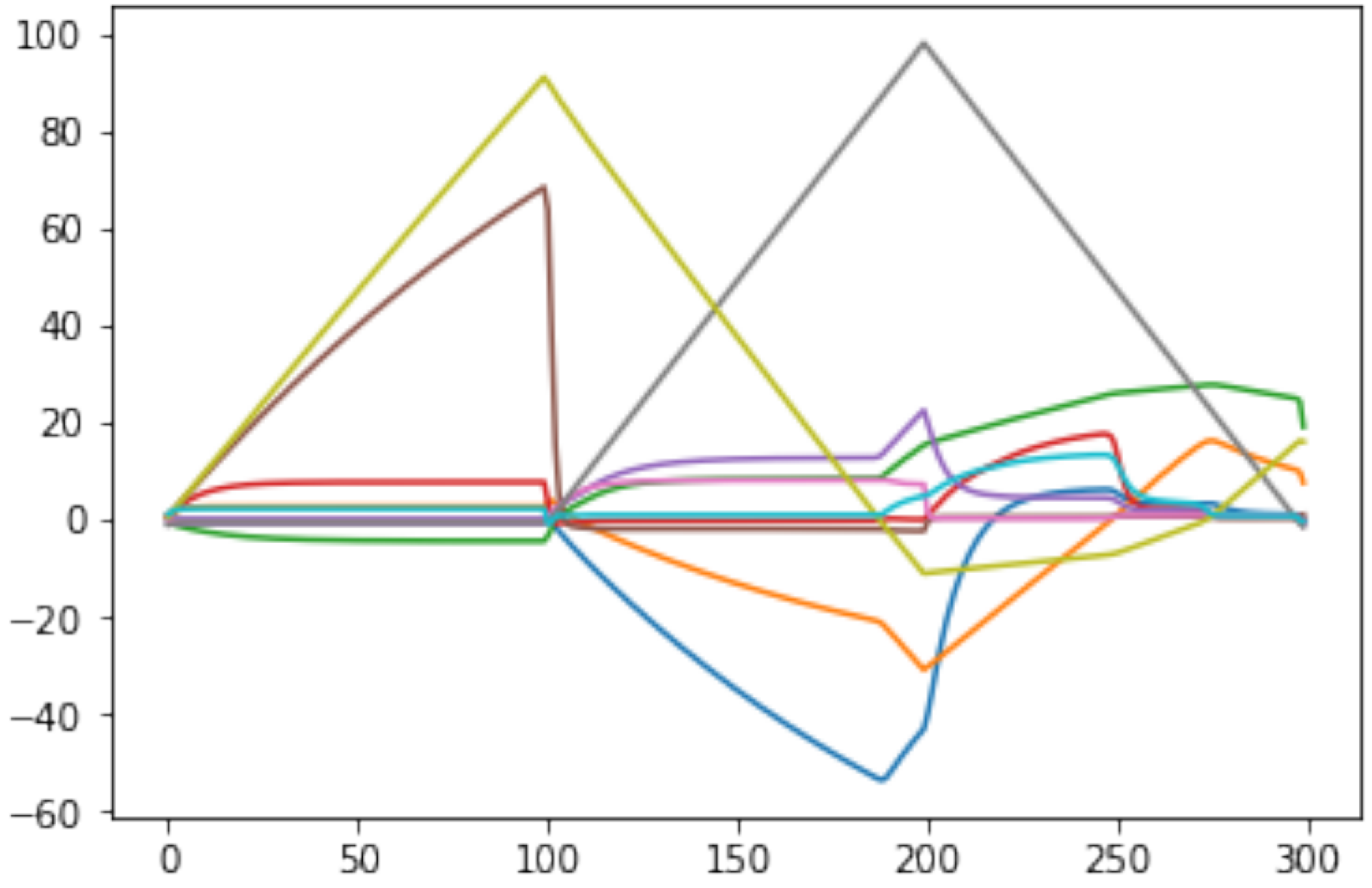
GRU begins failing at length 39

1. Common RNN Architectures

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

1. Common RNN Architectures

Conclusion:

LSTMs can dedicate any single dimension to counting, and can learn to do so when necessary

GRUs cannot, and seem to struggle on counting tasks overall

Are LSTMs always better than GRUs?

Other RNN Architectures

Let’s have a little fun

2. An Uncommon RNN Architecture

QRNNs (Quasi-Recurrent Neural Networks)

A proposed efficient alternative to LSTMs

Idea: state *update* is function of last k input tokens - without current state

(This way on large sequence, heavy calculation of all state updates can be computed in parallel, and only the updates themselves need to be chained)

High level:

$$x^{t,k} = (x_t, x_{t-1}, \ldots, x_{t+1-k})$$

$$z_t, f_t, o_t, i_t = w(x^{t,k})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot z_t$$

$$h_t = o_t \odot c_t$$

Could count - like an LSTM!

Can it be as strong as an LSTM?

2. An Uncommon RNN Architecture

QRNNs (Quasi-Recurrent Neural Networks)

High level:

$$x^{t,k} = (x_t, x_{t-1}, \ldots, x_{t+1-k})$$

$$z_t, f_t, o_t, i_t = w(x^{t,k})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot z_t$$

$$h_t = o_t \odot c_t$$

Could count - like an LSTM!

But, counter updates won't be state-dependent

2. An Uncommon RNN Architecture

QRNNs (Quasi-Recurrent Neural Networks)

High level:

$$x^{t,k} = (x_t, x_{t-1}, \ldots, x_{t+1-k})$$

$$z_t, f_t, o_t, i_t = w(x^{t,k})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot z_t$$

$$h_t = o_t \odot c_t$$

Could count - like an LSTM!

But, counter updates won't be state-dependent

For languages like $a^n b^n$, can do all the needed counting without 'self awareness'
- RNN classifier function g can finish the job

2. An Uncommon RNN Architecture

QRNNs (Quasi-Recurrent Neural Networks)

High level:

$$x^{t,k} = (x_t, x_{t-1}, \ldots, x_{t+1-k})$$

$$z_t, f_t, o_t, i_t = w(x^{t,k})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot z_t$$

$$h_t = o_t \odot c_t$$

Could count - like an LSTM!

But, counter updates won't be state-dependent

For languages like $a^n b^n$, can do all the needed counting without 'self awareness': RNN classifier function g can finish the job

Does this mean a QRNN is as powerful as an LSTM?

2. An Uncommon RNN Architecture

QRNNs (Quasi-Recurrent Neural Networks)

High level:

$$x^{t,k} = (x_t, x_{t-1}, \ldots, x_{t+1-k})$$

$$z_t, f_t, o_t, i_t = w(x^{t,k})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot z_t$$

$$h_t = o_t \odot c_t$$

Could count - like an LSTM!

But, counter updates won't be state-dependent

Some counter languages require state/counter-dependent updates!

Consider $L = \{a^n b^n w \,|\, n \geq 1, w \in \{a, b\}^*\}$

Need to know to stop updating the counters after reading $a^n b^n$!

3. A Completely Made Up RNN Architecture

3. A Completely Made Up RNN Architecture

Motivation:

$abcdef\$fedcba$

where am I going to cram all this?

LSTMs can count! But that's not *that* strong

State values increase at most linearly with input sequence length:
state "space" increases only polynomially

Need much more space to recognise things like palindromes (which requires holding entire prefix)

3. A Completely Made Up RNN Architecture

Motivation:

$abcdef\$fedcba$

where am I going to cram all this?

Need state space to grow much faster…

What if I allow state values to double??

3. A Completely Made Up RNN Architecture

Motivation:

$$abcdef\$fedcba$$

where am I going to cram all this?

Need state space to grow much faster…

What if I allow state values to double??

Doubling-LSTM

$$f_t^1 = \sigma(W^f x_t + U^f h_{t-1} + b^f)$$
$$f_t^2 = \sigma(W^f x_t + U^f h_{t-1} + b^f)$$
$$f_t = f_t^1 + f_t^2$$
$$\dots$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ g(c_t)$$

3. A Completely Made Up RNN Architecture

Motivation:

$abcdef\$fedcba$

where am I going to cram all this?

Doubling-LSTM

$$f_t^1 = \sigma(W^f x_t + U^f h_{t-1} + b^f)$$

$$f_t^2 = \sigma(W^f x_t + U^f h_{t-1} + b^f)$$

$$f_t = f_t^1 + f_t^2$$

$$\dots$$

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

$$h_t = o_t \circ g(c_t)$$

Now the RNN can implement reset ($f_t = 0$), keep ($f_t = 1$) **and** double ($f_t = 2$)

If we also assume it manages to stabilise on halve ($f_t = 1/2$), it effectively has a simple stack to push and pop binary tokens to/from!

Did it work?

3. A Completely Made Up RNN Architecture

Motivation:

$abcdef\$fedcba$

where am I going to cram all this?

Doubling-LSTM

$$f_t^1 = \sigma(W^f x_t + U^f h_{t-1} + b^f)$$
$$f_t^2 = \sigma(W^f x_t + U^f h_{t-1} + b^f)$$
$$f_t = f_t^1 + f_t^2$$
$$\dots$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ g(c_t)$$

Now the RNN can implement reset ($f_t = 0$), keep ($f_t = 1$) **and** double ($f_t = 2$)

If we also assume it manages to stabilise on halve ($f_t = 1/2$), it effectively has a simple stack to push and pop binary tokens to/from!

Did it work?

It turns out doubling the values in RNN states quickly gets too big for training…

RNNs with Augmentations

The doubling LSTM was a failure… but getting a stack doesn't have to be doomed

Several works propose augmenting an RNN with an external stack

(Other works propose adding attention, but you've already seen that :))

The stack is *not* encoded as some horrifically large number

Instead an actual stack, with differentiable push, pop, and read operations, is carefully designed

(That's the hard part)

The RNN learns to interact with the stack to solve tasks

Stack RNNs

1. Stratification (layer cake stack)

Items in stack have a “thickness” - layers of a cake

RNN computes a distribution d over push, pop, do nothing

Push, pop, and nothing all(!) happen, **each with thickness according to** d

When read width covers several stack layers: they are **averaged (by thickness)**

Expectation

Reality

Stack RNNs

2. Superposition (Try everything stack)

Items in stack get pushed, popped, or left alone completely - like a true stack

RNN still computes a distribution d over push, pop, do nothing

Push, pop, and nothing all(!) happen, each in **different stack - all weighted by** d

Creates polynomial (!) number of potential stacks over time, d is composed with the existing distribution

Reading from stack takes **weighted average of stack tops**

Stack RNNs

2. Superposition (Try everything stack)

Items in stack get pushed, popped, or left alone completely - like a true stack

RNN still computes a distribution d over push, pop, do nothing

Push, pop, and nothing all(!) happen, each in **different stack - all weighted by d**

Creates polynomial (!) number of potential stacks over time, d is composed with the existing distribution

Reading from stack takes **weighted average of stack tops**

(3. Nondeterminism)

New: similar to superposition, but achieved with simulation of a nondeterministic pushdown automaton

References

RNN Architectures

- Elman RNN (s-RNN) - *Finding Structure in Time* - Elman, 1990
- LSTM - *Long short-term memory* - Hochreiter and Schmidhuber, 1997
- GRU - *On the properties of neural machine translation: encoder-decoder approaches* - Cho et al, 2014; *Empirical evaluation of gated recurrent neural networks on sequence modeling* - Chung et al, 2014
- QRNN - *Quasi-Recurrent Neural Networks* - Bradbury et al, 2016
- Stack RNNs - *The Neural Network Pushdown Automaton: Model, Stack and Learning Simulations* - Sun et al, 1995; *Learning to Transduce with Unbounded Memory* - Grefenstette et al, 2015; *Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets* - Joulin and Mikolov, 2015; *Learning Context-Free Languages with Nondeterministic Stack RNNs* - DuSell and Chiang, 2022

Theoretical Results

- Turing Completeness - *On the Computational Power of Neural Nets*, Siegelmann and Sontag 1992
- RNNs as DFAs - *Finite state automata and simple recurrent networks,* Cleeremans et al, 1989
- Counting - *On the practical computational power of finite precision rnns for language recognition* - Weiss et al, 2018
- Saturated RNNs as a framework - *Sequential neural networks as automata* - Merrill, 2019
- QRNNs and more - *A formal hierarchy of RNN architectures* - Merrill et al, 2020
- 2RNNs as WFAs - *Connecting Weighted Automata and Recurrent Neural Networks through Spectral Learning* - Rabusseau et al, 2019