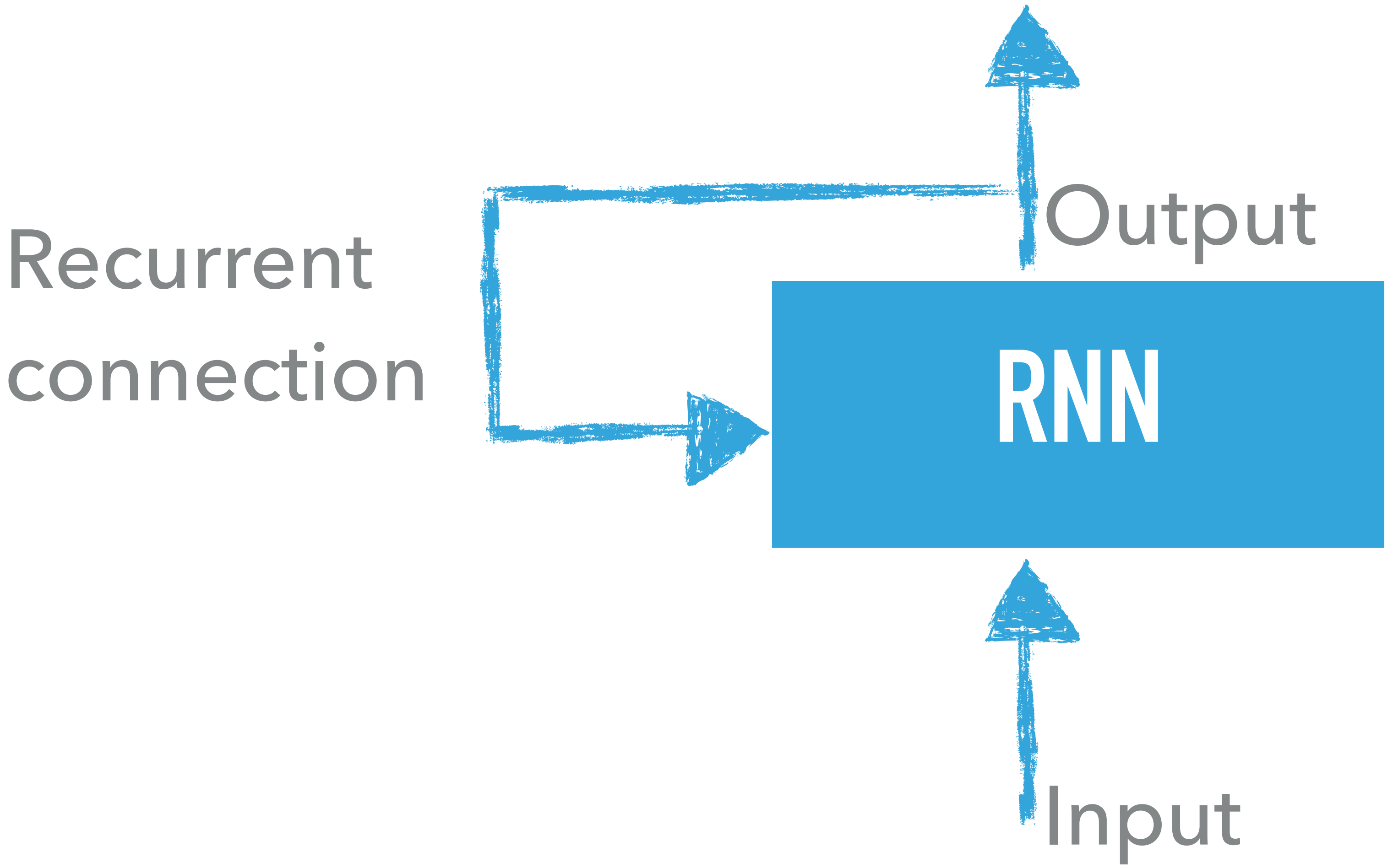
LONG SHORT TERM MEMORY

LSTM NETWORKS

Simple RNN - recurrent neural network



Loop. State

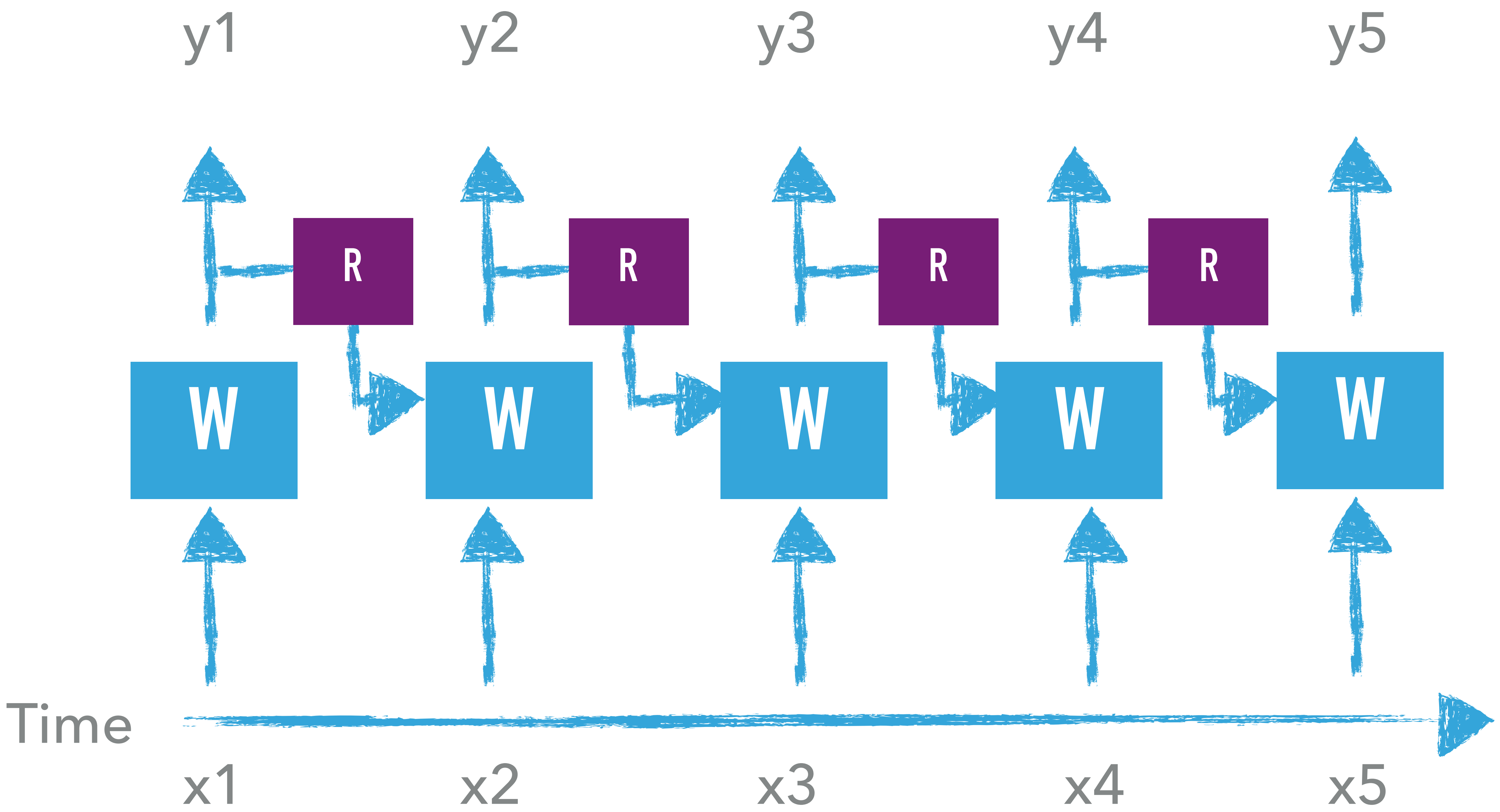
PLAIN OLD NN

```
activation(dot(W, input_t) + b)
```

RNN

```
activation(dot(W, input_t) +
                dot(U, state_t) + b)
```

RUN



MAIN PURPOSE

Remember what is has seen so far (state)

Capture Long Distance Dependencies

MAIN PURPOSE

Remember what is has seen so far (state)

Capture Long Distance Dependencies

Not so good

TURNS OUT . . .

Simple RNNs are too simple to deal with long-distance dependencies

A RNN variant, LSTM work much better.

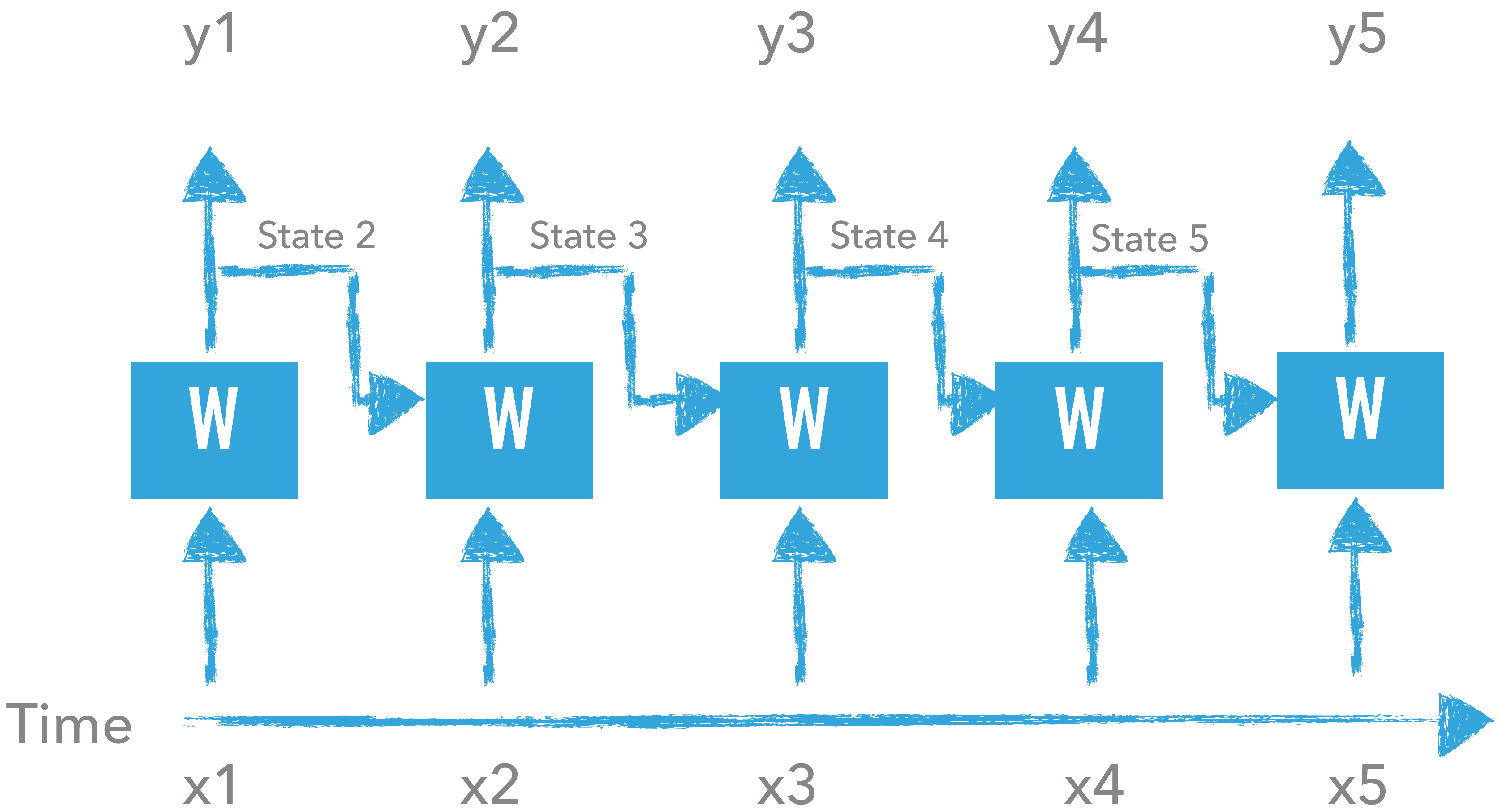
ALMOST ALL THE APPLICATIONS OF RNN USE LSTM

- Speech recognition
- Machine translation
- Image captioning
- …

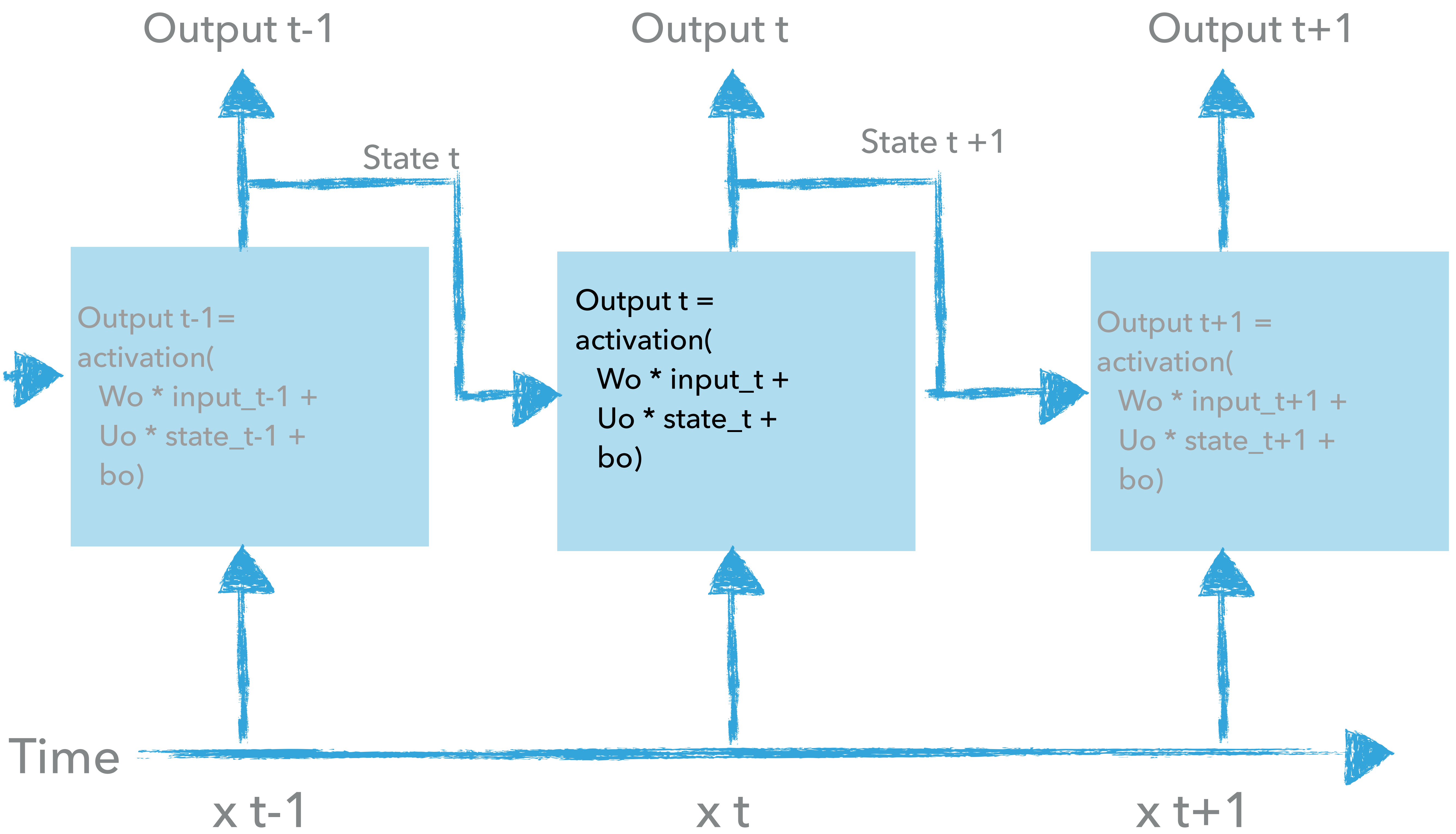
AS ALWAYS

May look complicated

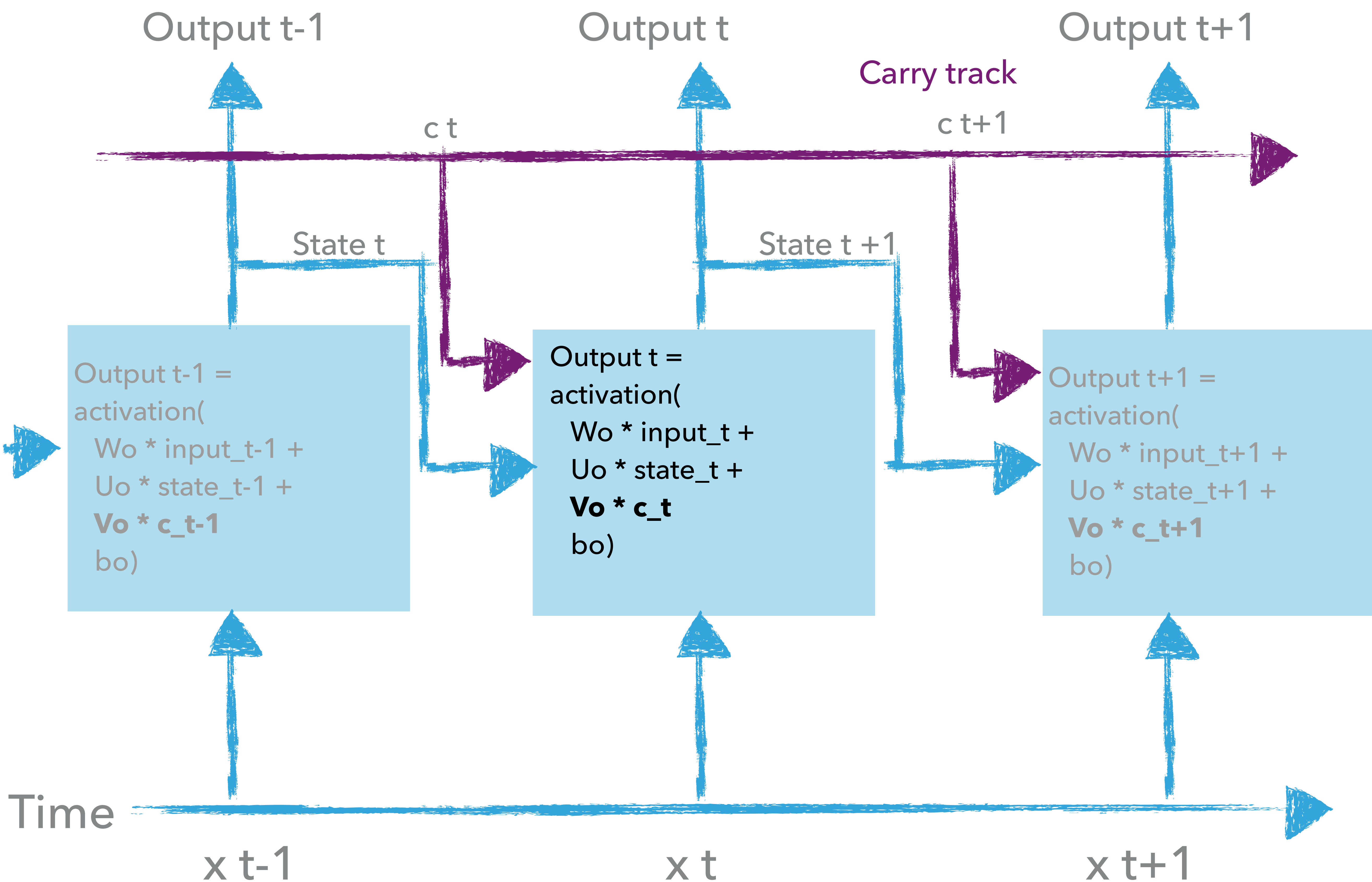
“Not so complicated—merely complex.” - Chollet


y1
y2
y3
y4
y5
State 2
State 3
State 4
State 5
W
W
W
W
W
Time
x1
x2
x3
x4
x5

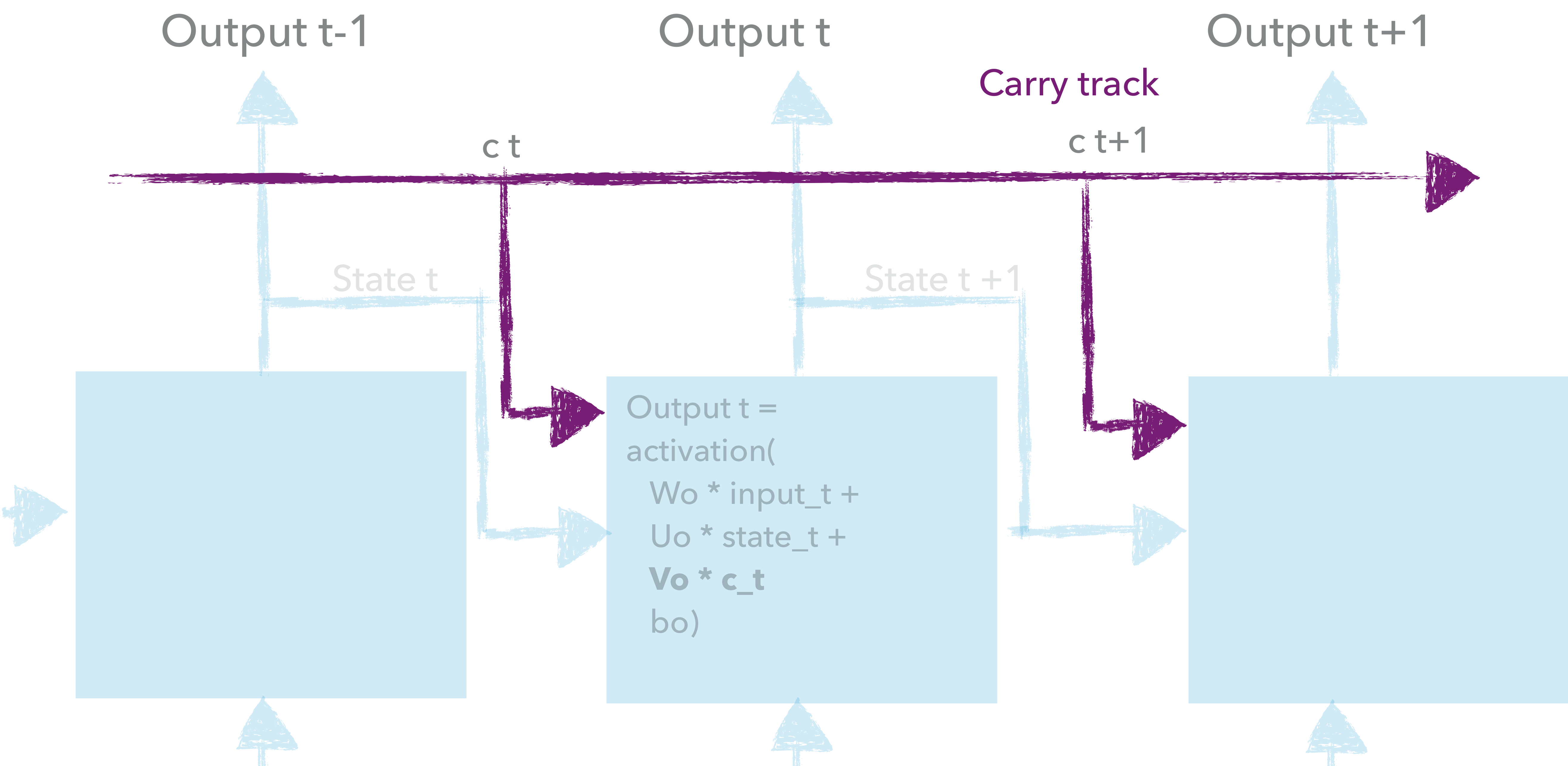
Plain Vanilla RNN



LSTM

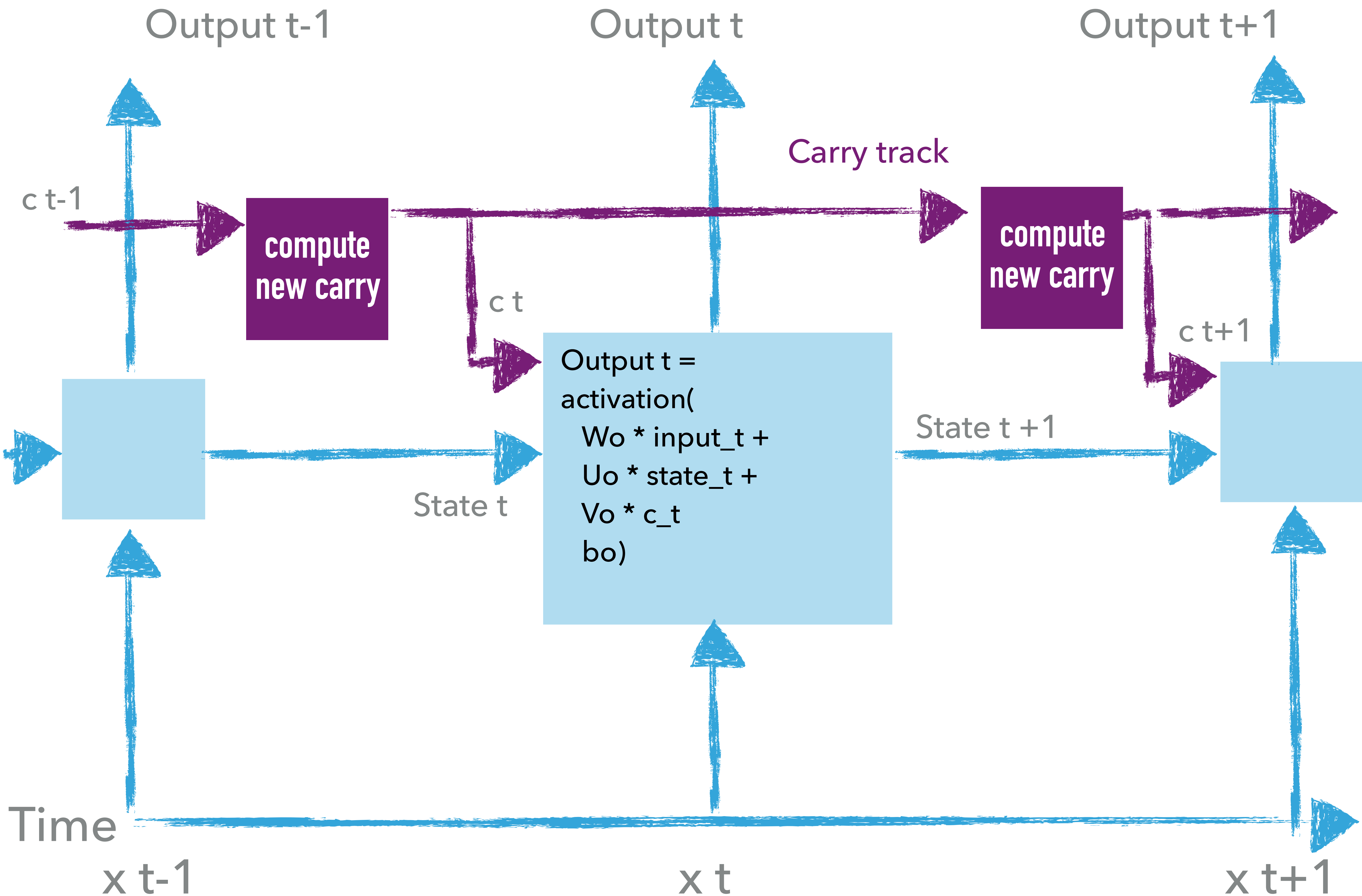

Output t-1
Output t
Output t+1
Carry track
c t
c t+1
State t
State t +1
Output t-1 =
activation(
Wo * input_t-1 +
Uo * state_t-1 +
Vo * c_t-1
bo)
Output t =
activation(
Wo * input_t +
Uo * state_t +
Vo * c_t
bo)
Output t+1 =
activation(
Wo * input_t+1 +
Uo * state_t+1 +
Vo * c_t+1
bo)
Time
x t-1
x t
x t+1

LSTM

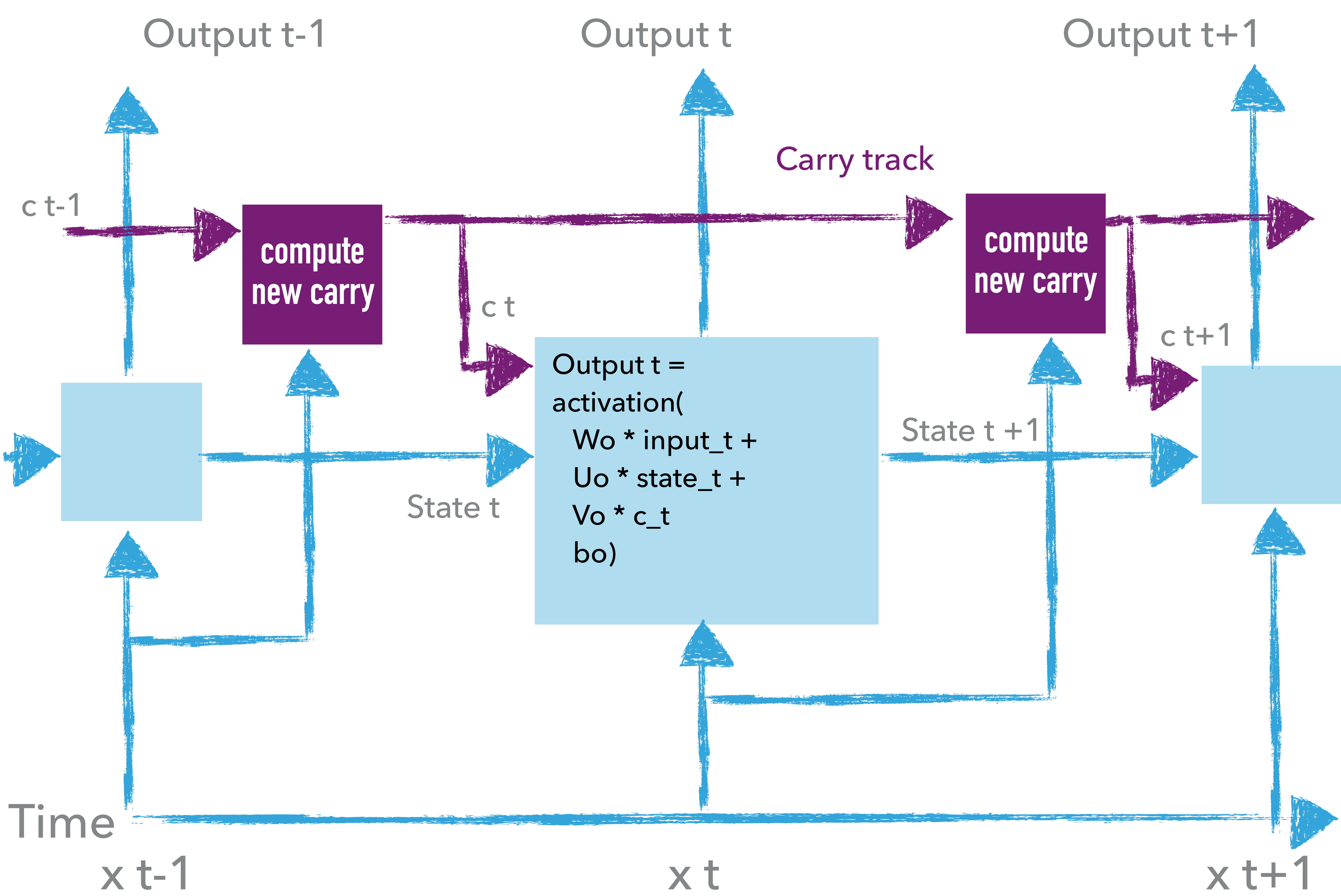


CARRY TRACK: information flows -> add new info -> forget selected info

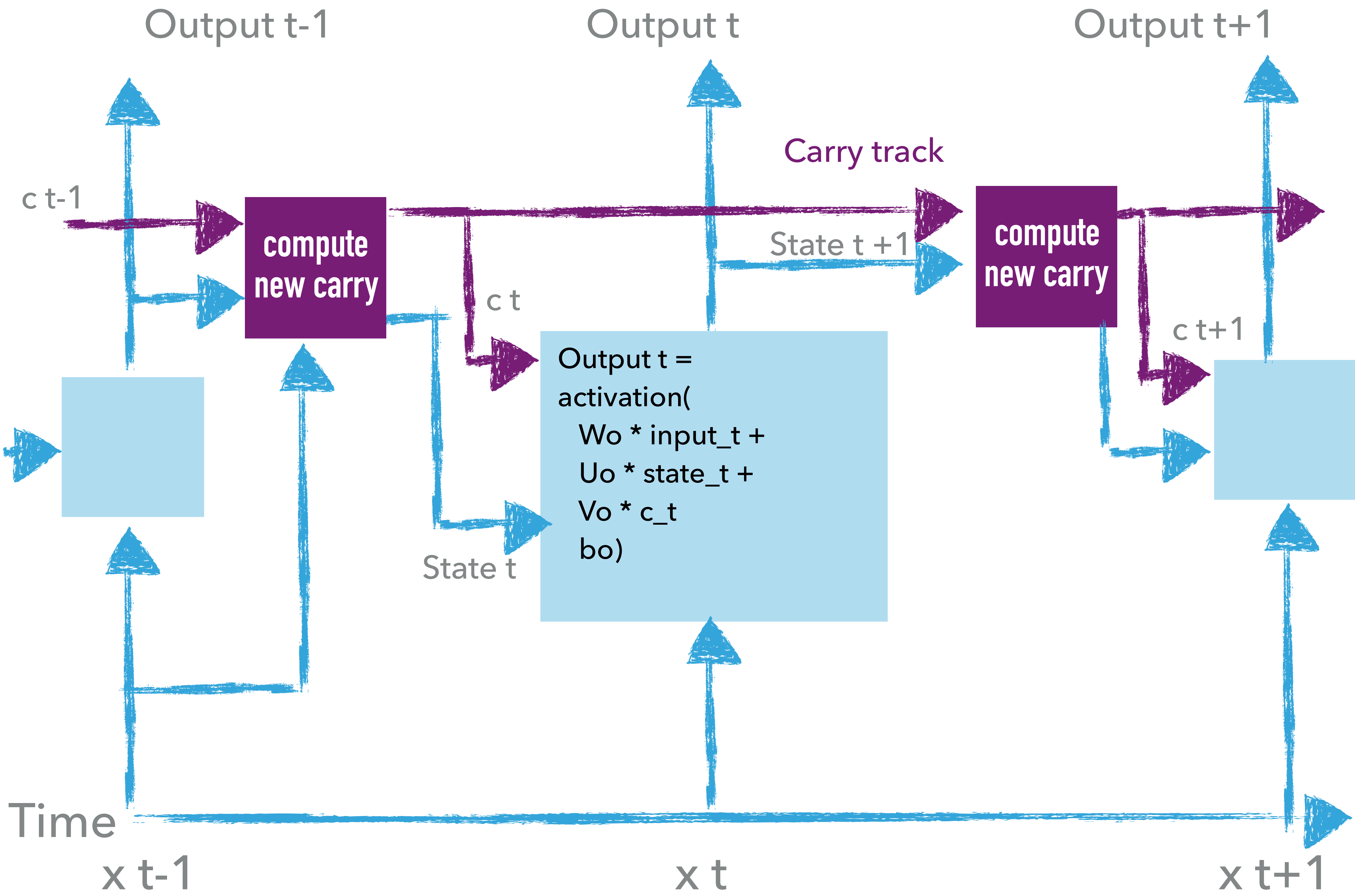
Compute new carry based on previous carry



As well as current input



And the current state



THAT'S THE

Mid-level description

A CLOSER LOOK

Pseudocode

PSEUDOCODE

```
output_t = activation(dot(state_t, Uo) +
                      dot(input_t, Wo) +
                      dot(c_t, Vo) + bo)
```

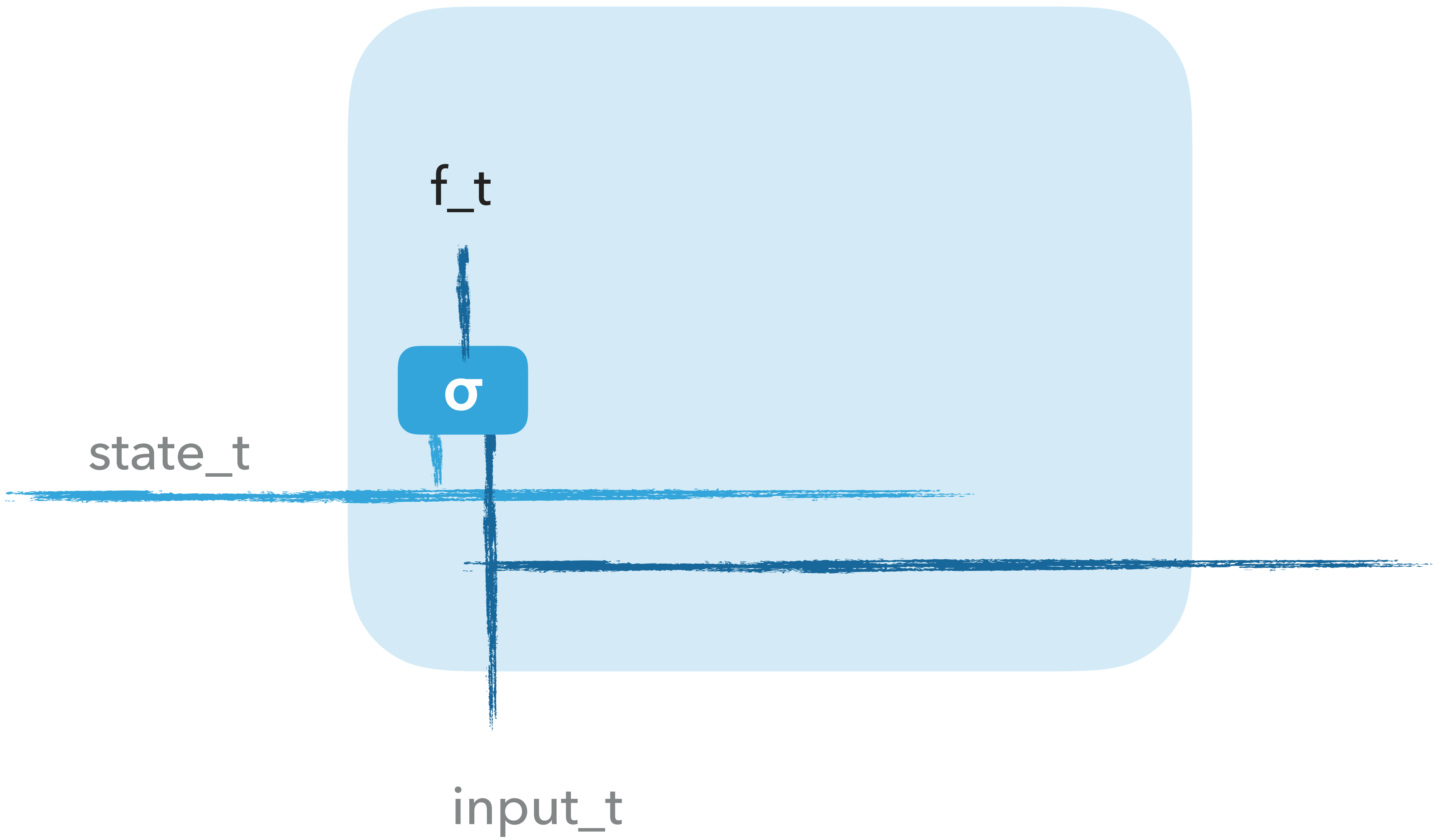
PSEUDOCODE

```
output_t = activation(dot(state_t, Uo) +
                      dot(input_t, Wo) +
                      dot(c_t, Vo) + bo)


dot([1, 2, 3, 4, 5], [10, 20, 30, 40, 50])
   = 1 * 10 + 2 * 20 + 3 * 30 + 4 * 40 + 5 * 50
   = 10 + 40 + 90 + 160 + 250
   = 590
```

Forget gate layer -> what information are we going to throw out

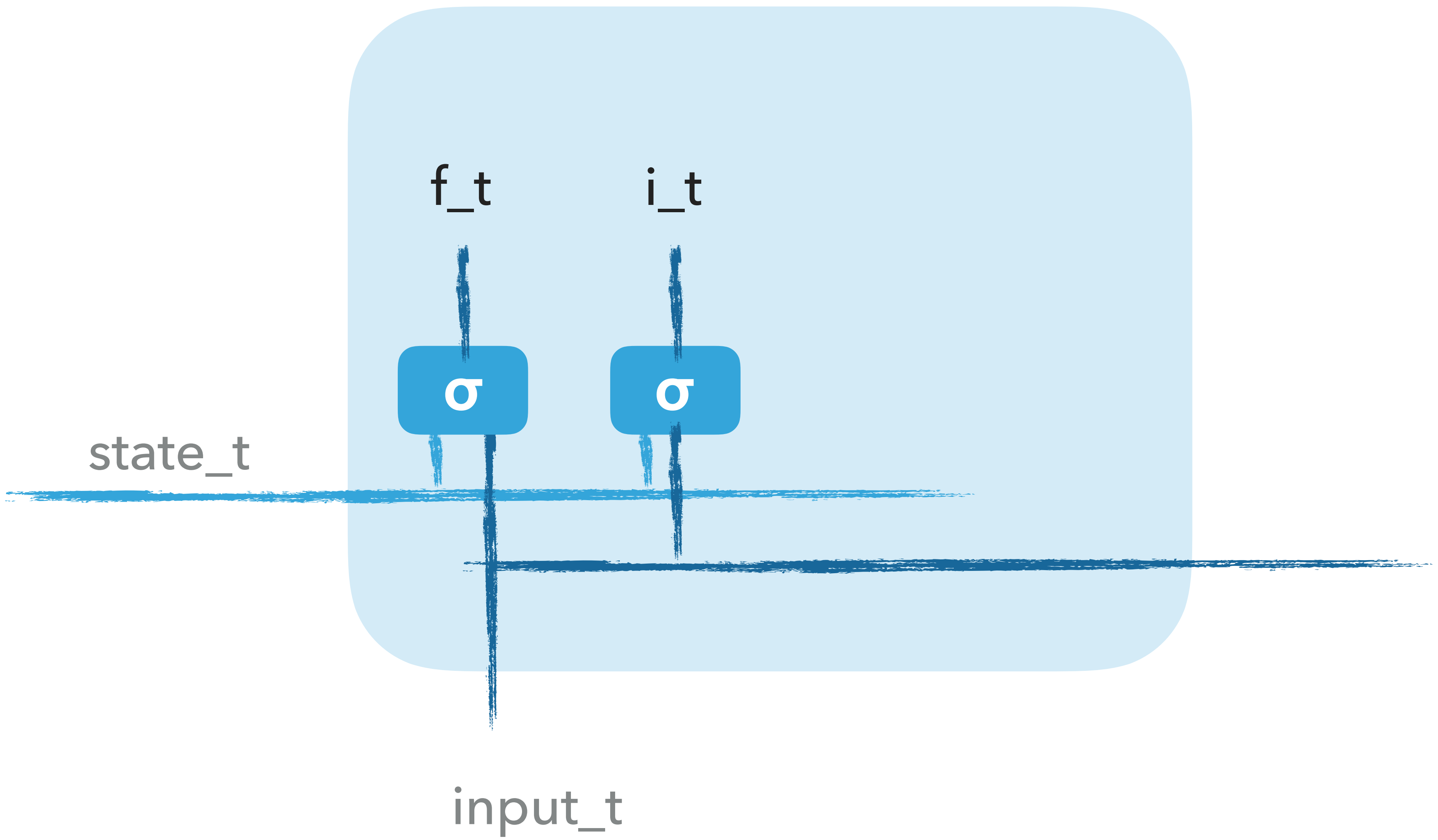
```
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
```



(Source : http://colah.github.io/posts/2015-08-Understanding-LSTMs)

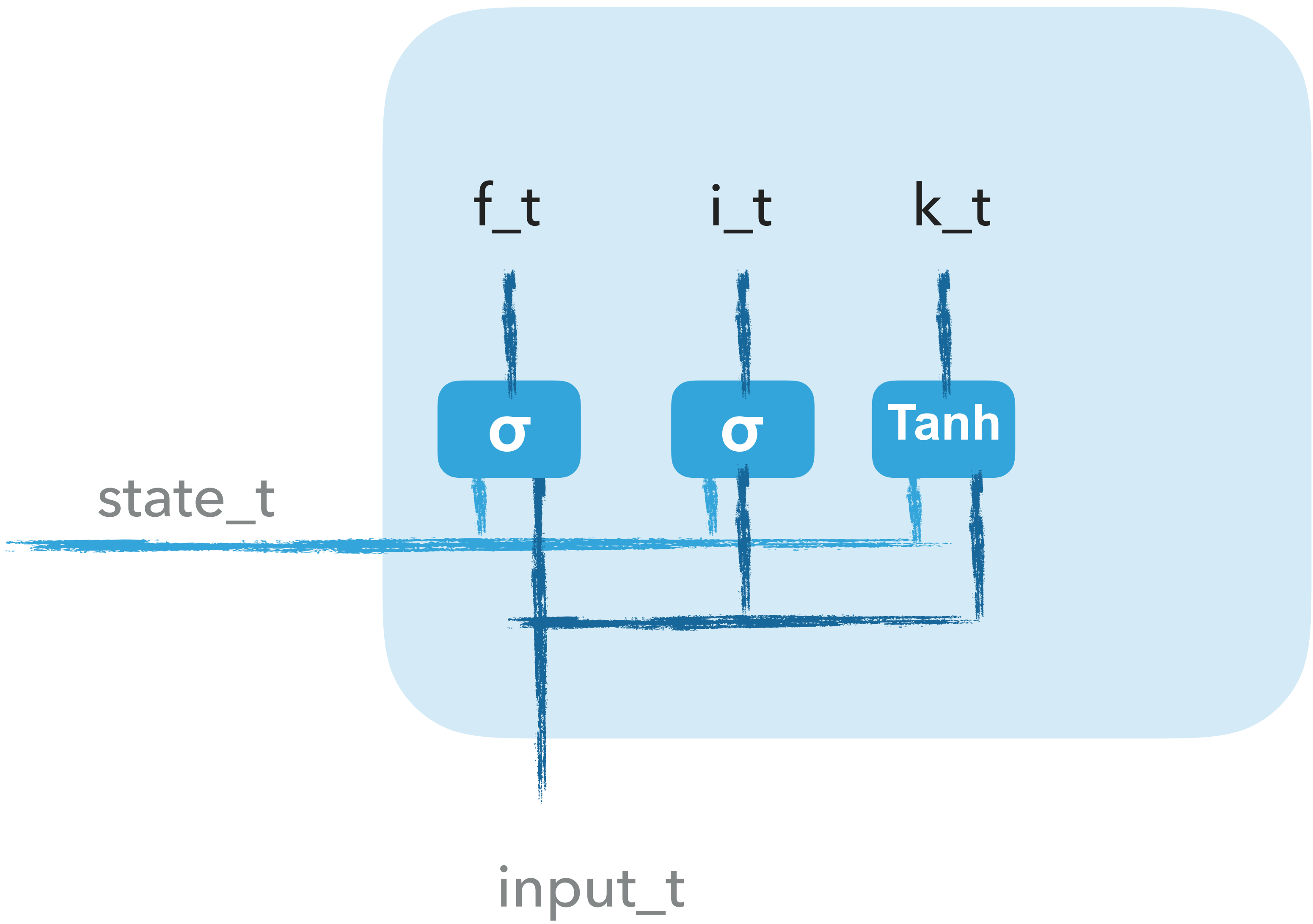
Input gate layer -> contributes to the information are we going to keep

```
i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
```



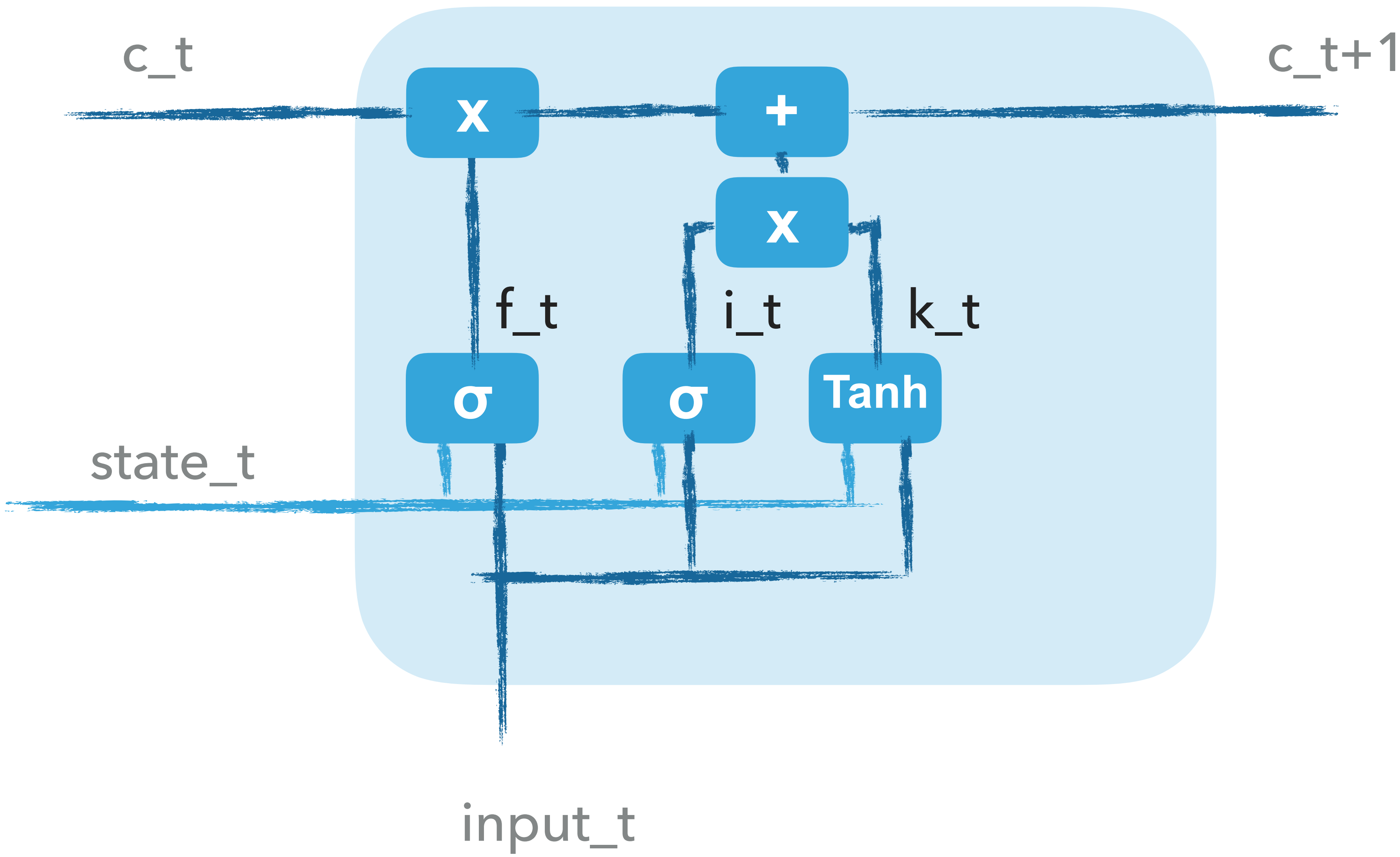
K-gate. - Keep gate

```
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)
```



Compute new carry

```
c+t+1 = i_t * k_t + c_t * f_t
```



In 2009, deep multidimensional LSTM networks demonstrated the power of deep learning with many nonlinear layers, by winning three ICDAR 2009 competitions in connected handwriting recognition, without any prior knowledge about the three different languages to be learned

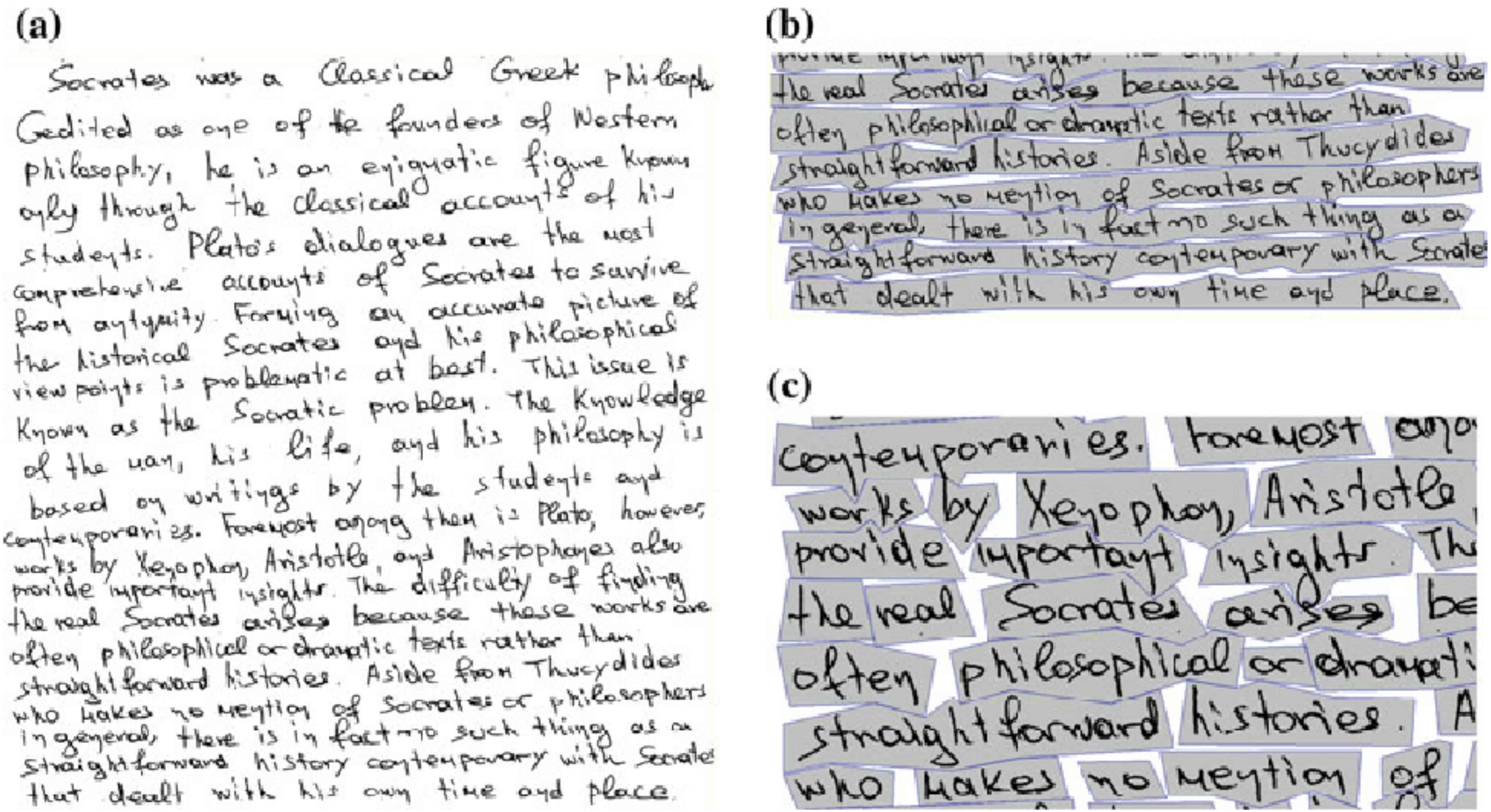
A Novel Connectionist System for Improved Unconstrained Handwriting Recognition

The fire brigade has arrived.

Adenauer is in a tough spot. Waiting.

bring support and comfort to

Commonwealth countries do


(a)
(b)
(c)

LSTM

LAB

SEQ2SEQ

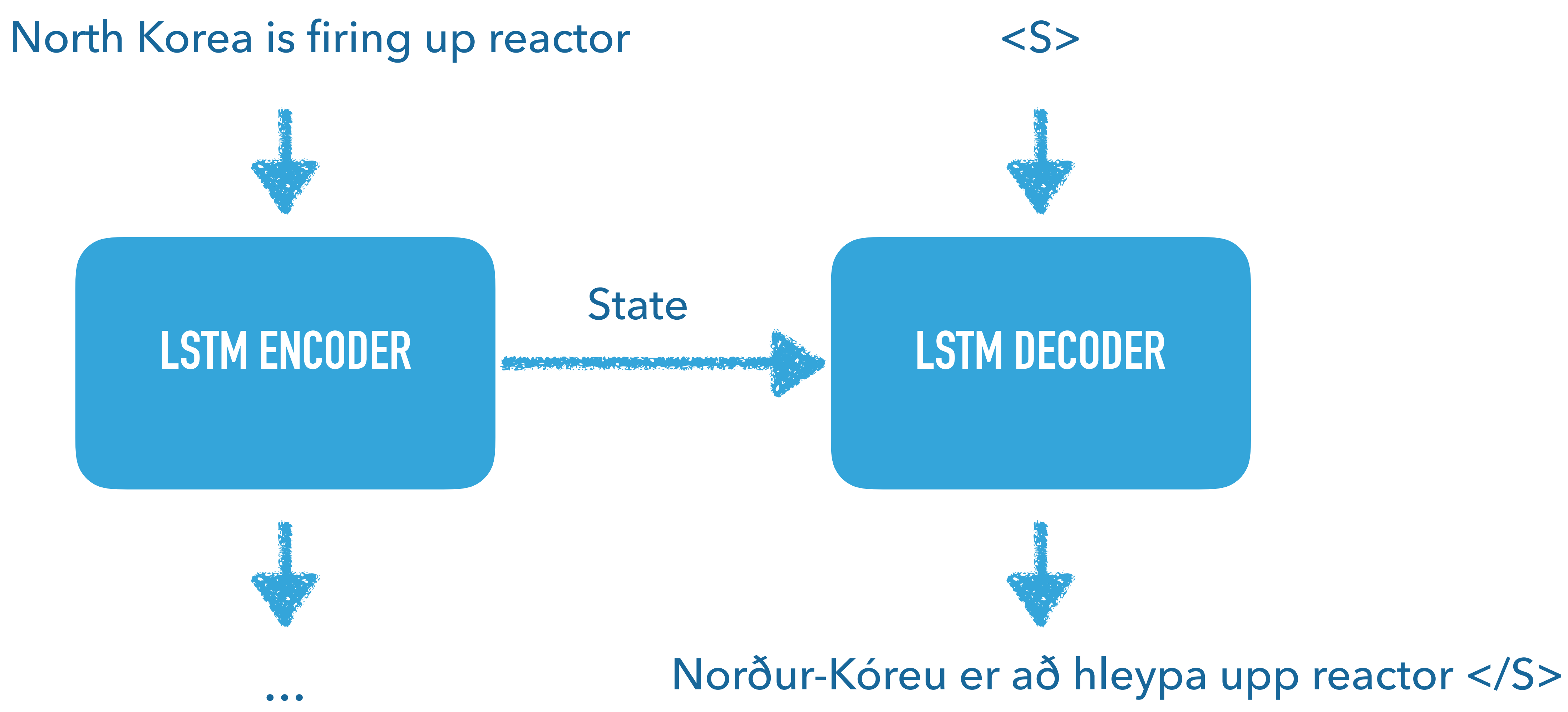
CANONICAL SEQ2SEQ LEARNING PHASE

- Encoder RNN layer(s)
  - processes the input sequence and returns its own internal state.
  - We discard the outputs of the encoder RNN, only recovering the state.
- Decoder RNN
  - trained to predict the next characters of the target sequence, given previous characters of the target sequence.
  - uses as initial state the state vectors from the encoder, which is how the decoder obtains information about what it is supposed to generate.

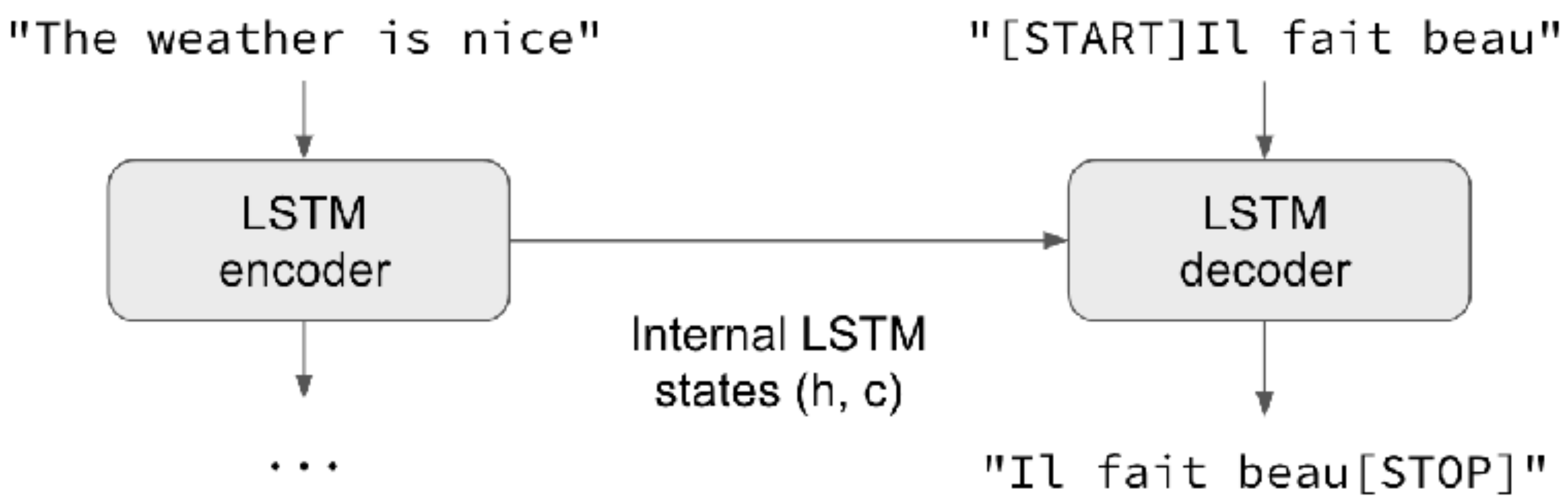
INFERENCE PHASE

- Run source language sentence through encoder to get state
- Feed state and start symbol (for ex., <S>) to decoder to get get first character/word.
- Repeat until you generate </S> (end of sequence symbol)

INFERENCE PHASE



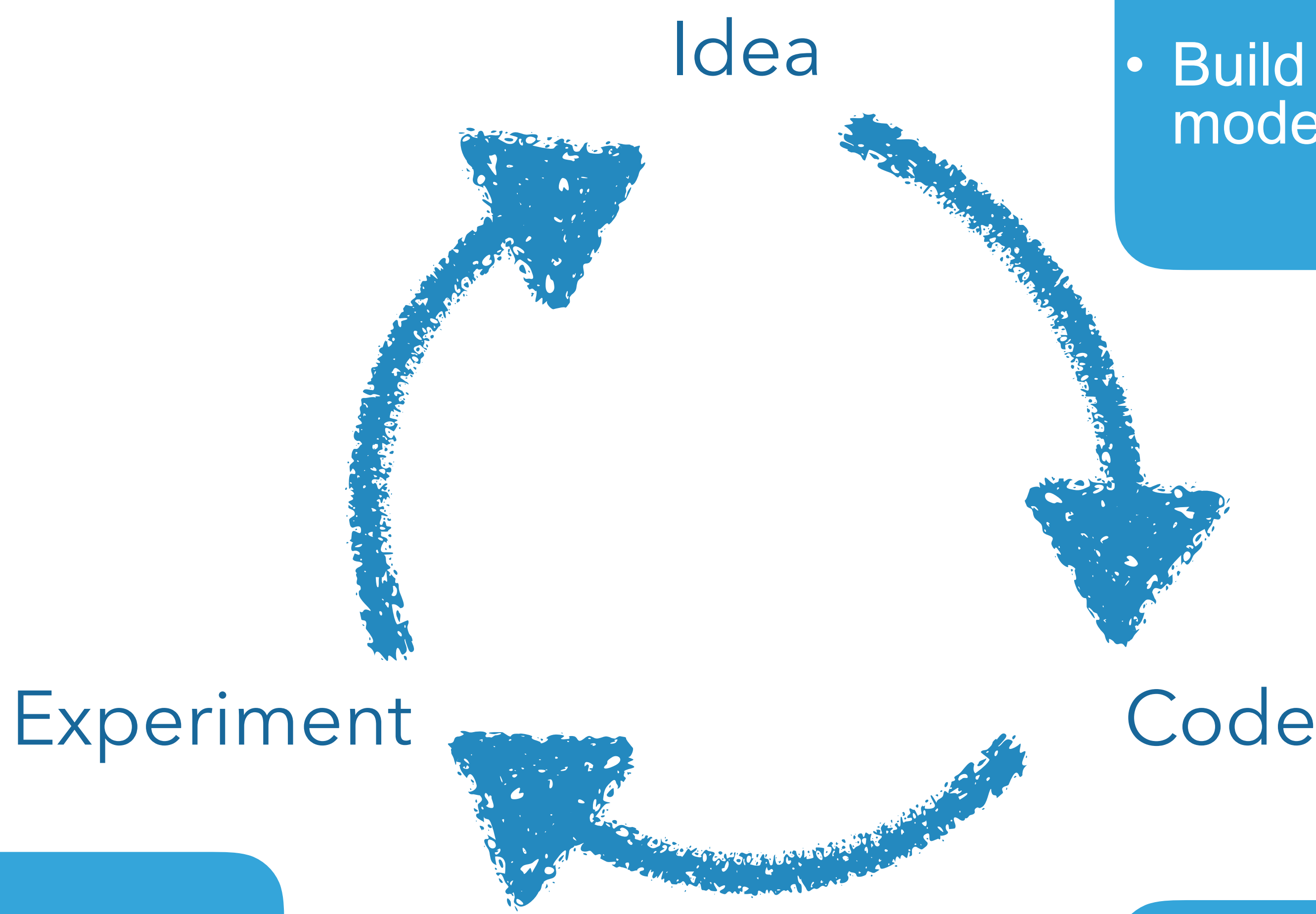
INFERENCE PHASE



```
output_t = activation(dot(state_t, Uo) +
                      dot(input_t, Wo) +
                      dot(c_t, Vo) + bo)

state_t+1 = dot(output_t, activation(c_t))
```

APPLIED DEEP LEARNING IS A VERY EMPIRICAL PROCESS



- Build first model quickly

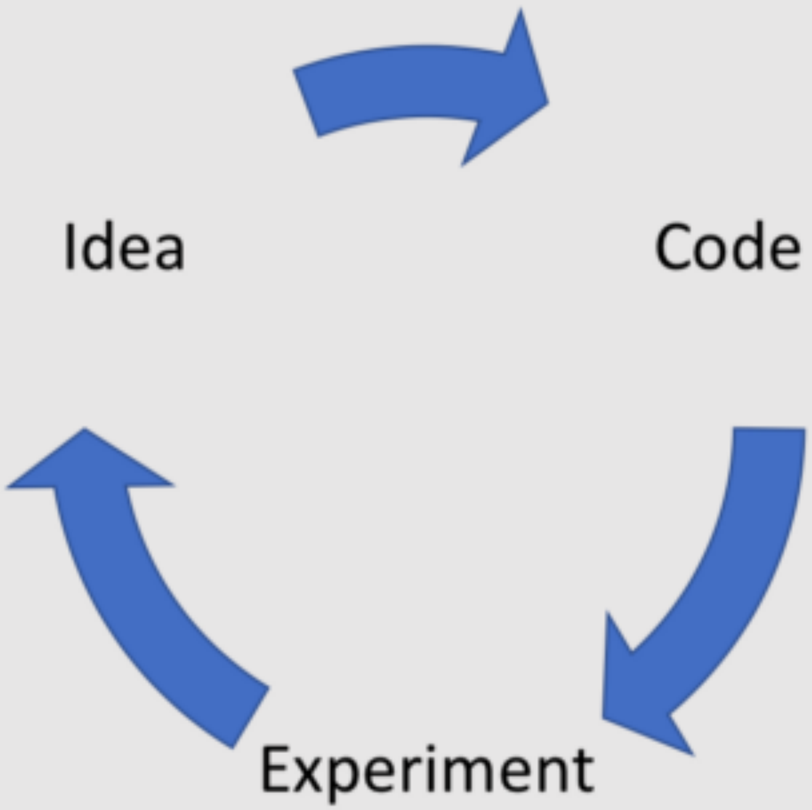
- Use train/ validation/test set

- Evaluate and adjust parameters

From Andrew Ng

Quickly iterate over your ML project following these strategic guidelines

- Build your first model quickly
- Define a single optimizing metric
- Define constraints and satisficing metrics



- Use a 98/1/1% train/dev/test set distribution
- Dev/test sets have the same data distribution
- Dev/test sets contain only examples to optimize for

- Evaluate errors to estimate bias, variance and data mismatch
- Manually analyze and label 100 misclassified examples
- Consider end-to-end, transfer or multitask learning

PSEUDOCODE

```
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)

c+t+1 = i_t * k_t + c_t * f_t
```

PSEUDOCODE

```
output_t = activation(dot(state_t, Uo) +
                      dot(input_t, Wo) +
                      dot(c_t, Vo) + bo)
```

```
dot([1, 2, 3, 4, 5], [10, 20, 30, 40, 50])
   = 1 * 10 + 2 * 20 + 3 * 30 + 4 * 40 + 5 * 50
   = 10 + 40 + 90 + 160 + 250
   = 590
```