# Building Regression Models using TensorFlow

## LEARNING USING NEURONS

**Vitthal Srinivasan**
CO-FOUNDER, LOONYCORN

www.loonycorn.com

# Overview

The most common use of TensorFlow is in deep learning

Neural networks are the most common type of deep learning algorithm

The basic building block of a neural network is a neuron

Linear regression can be “learnt” using a single neuron

Deep learning extends this idea to more complex, non-linear functions

# Prerequisites and Course Outline

# Course Outline

**Learning using Neurons**

**Linear Regression in TensorFlow**

**Logistic Regression in TensorFlow**

**Estimators**

# Related Courses

**Understanding the Foundations of TensorFlow**

**Understanding and Applying Linear Regression**

**Understanding and Applying Logistic Regression**

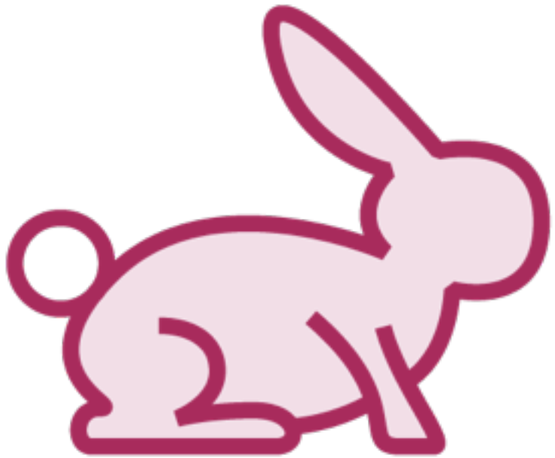
# Software and Skills

**Have TensorFlow installed**

**Know some Python programming**

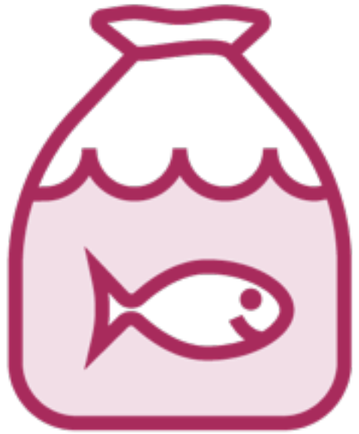
# Understanding Deep Learning

# Whales: Fish or Mammals?



**Mammals**

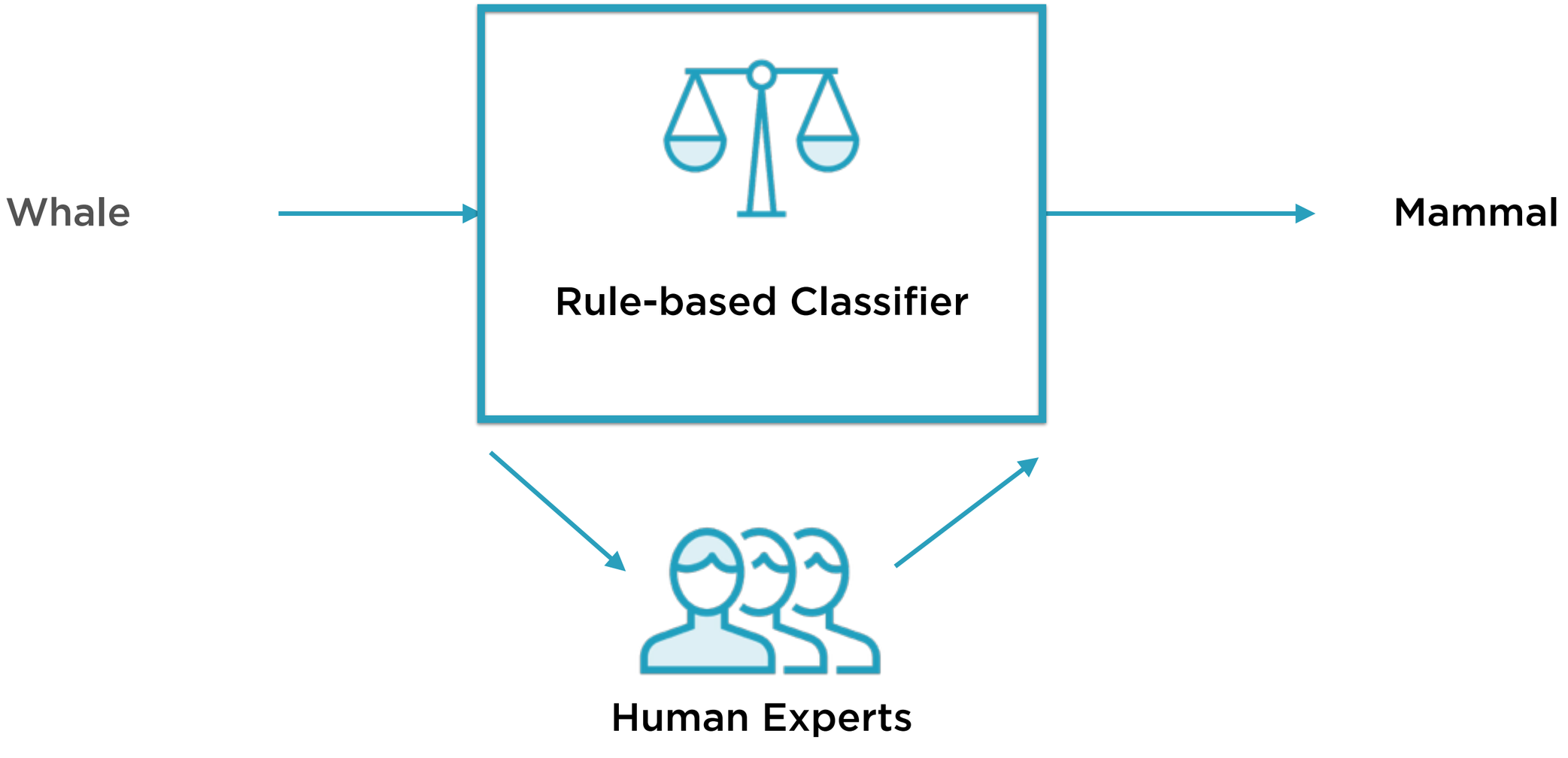
**Members of the infraorder *Cetacea***



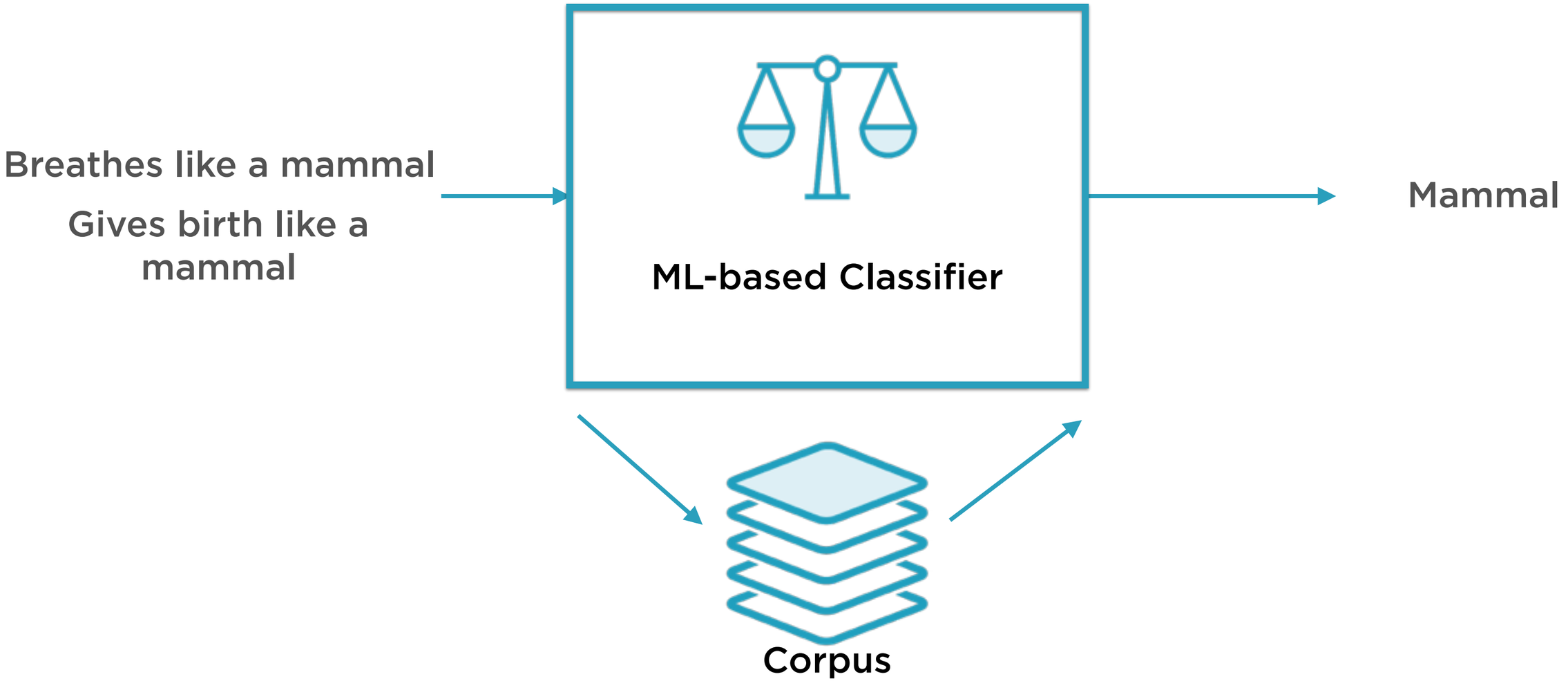
**Fish**

**Look like fish, swim like fish, move with fish**

# Rule-based Binary Classifier



# “Traditional” ML-based Binary Classifier



# “Traditional” ML-based Binary Classifier

**Corpus**

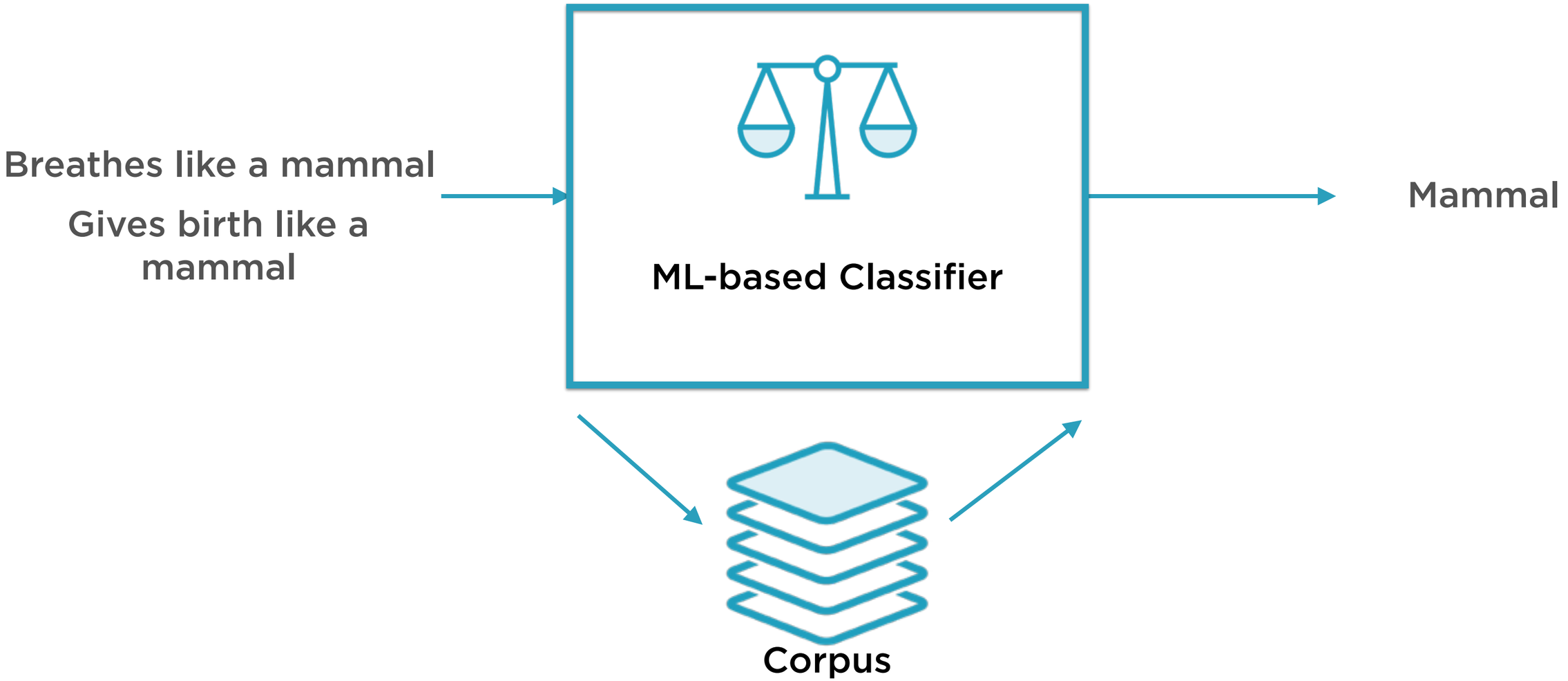
**Classification Algorithm**

**ML-based Classifier**

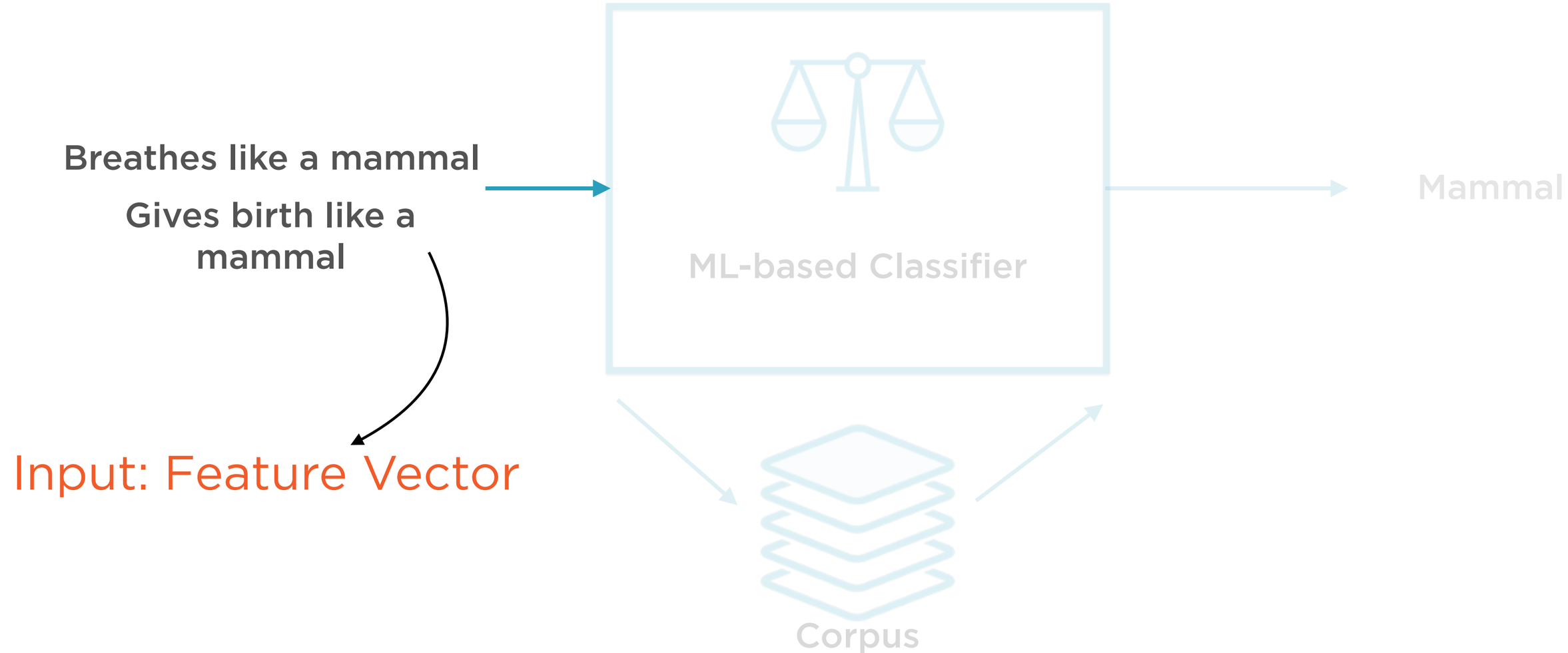
# “Traditional” ML-based vs. Rule-based

| “Traditional” ML-based | Rule-based |
| --- | --- |
| Dynamic | Static |
| Experts optional | Experts required |
| Corpus required | Corpus optional |
| Training step | No training step |

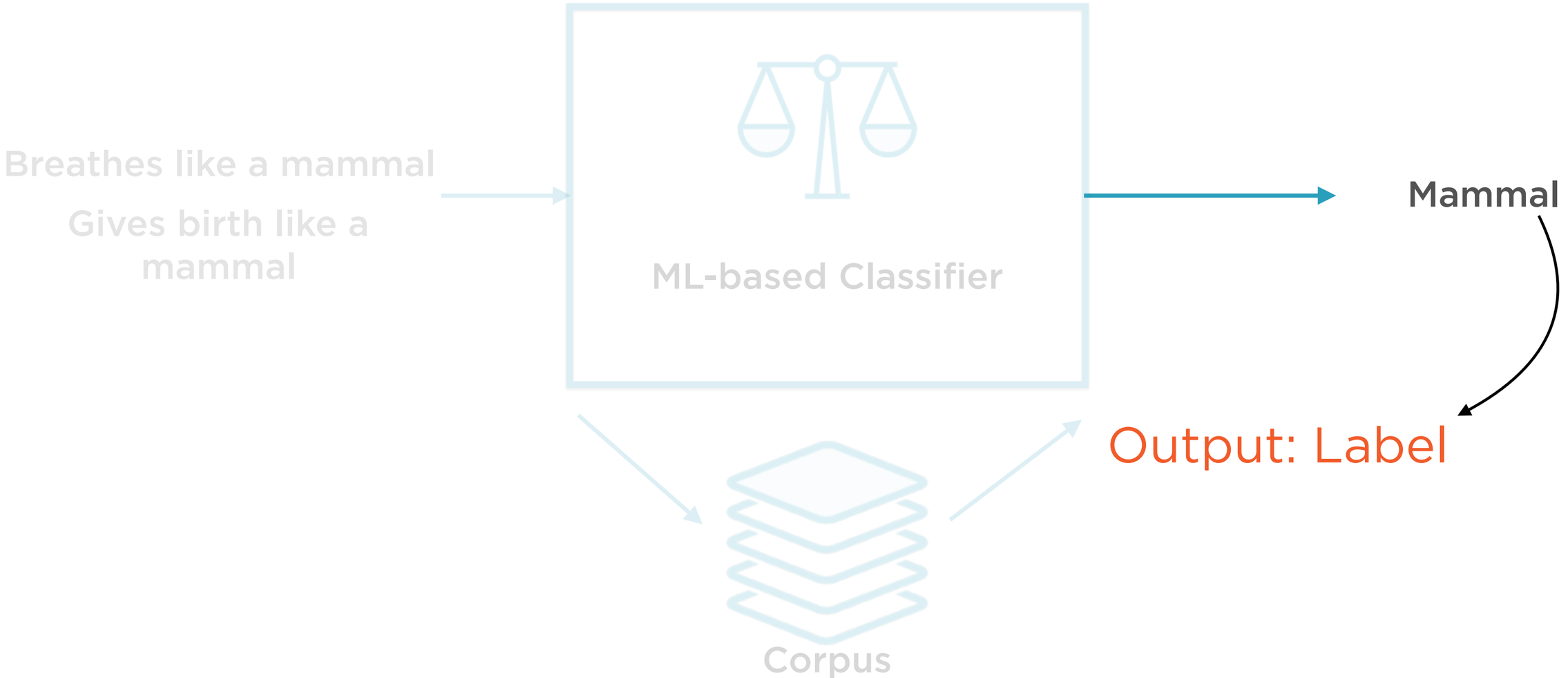
# “Traditional” ML-based Binary Classifier



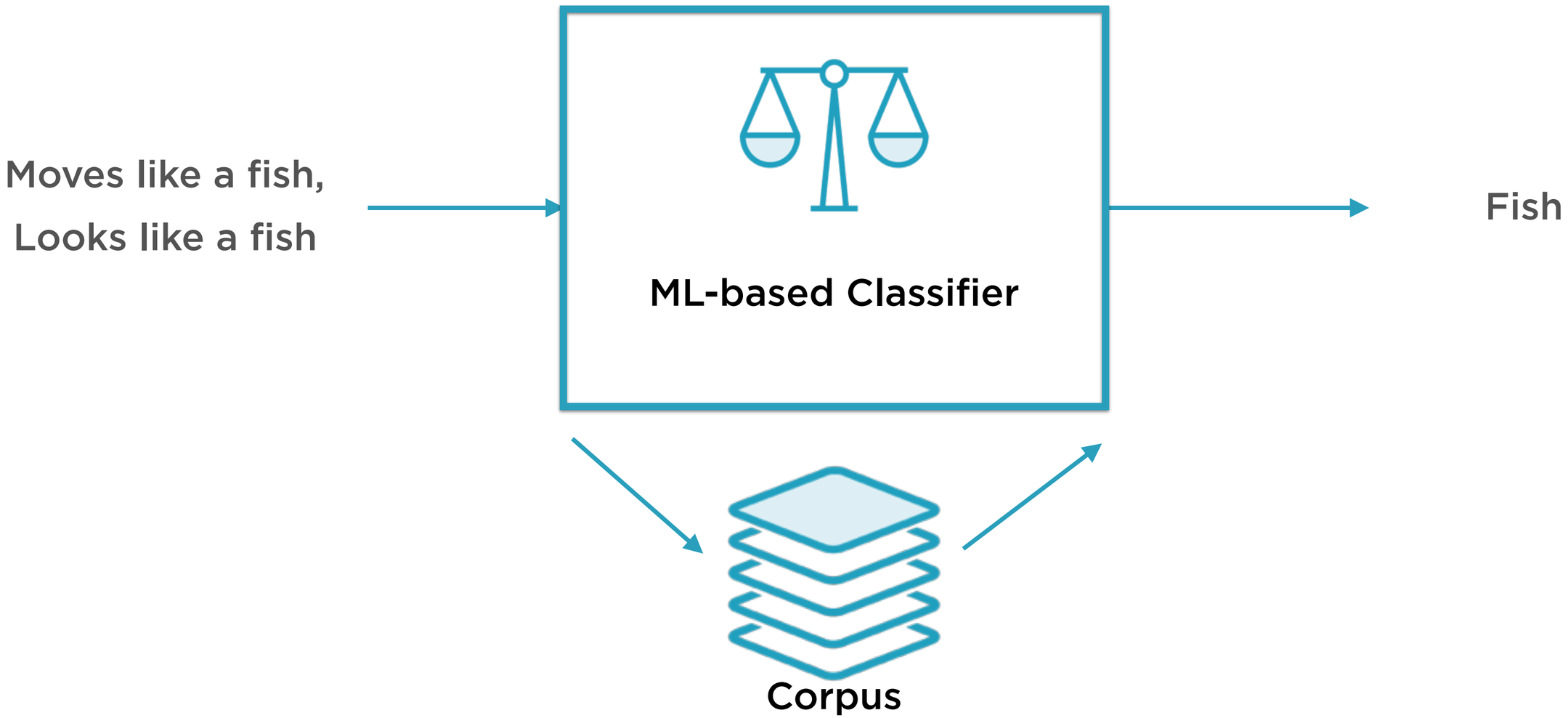
# “Traditional” ML-based Binary Classifier



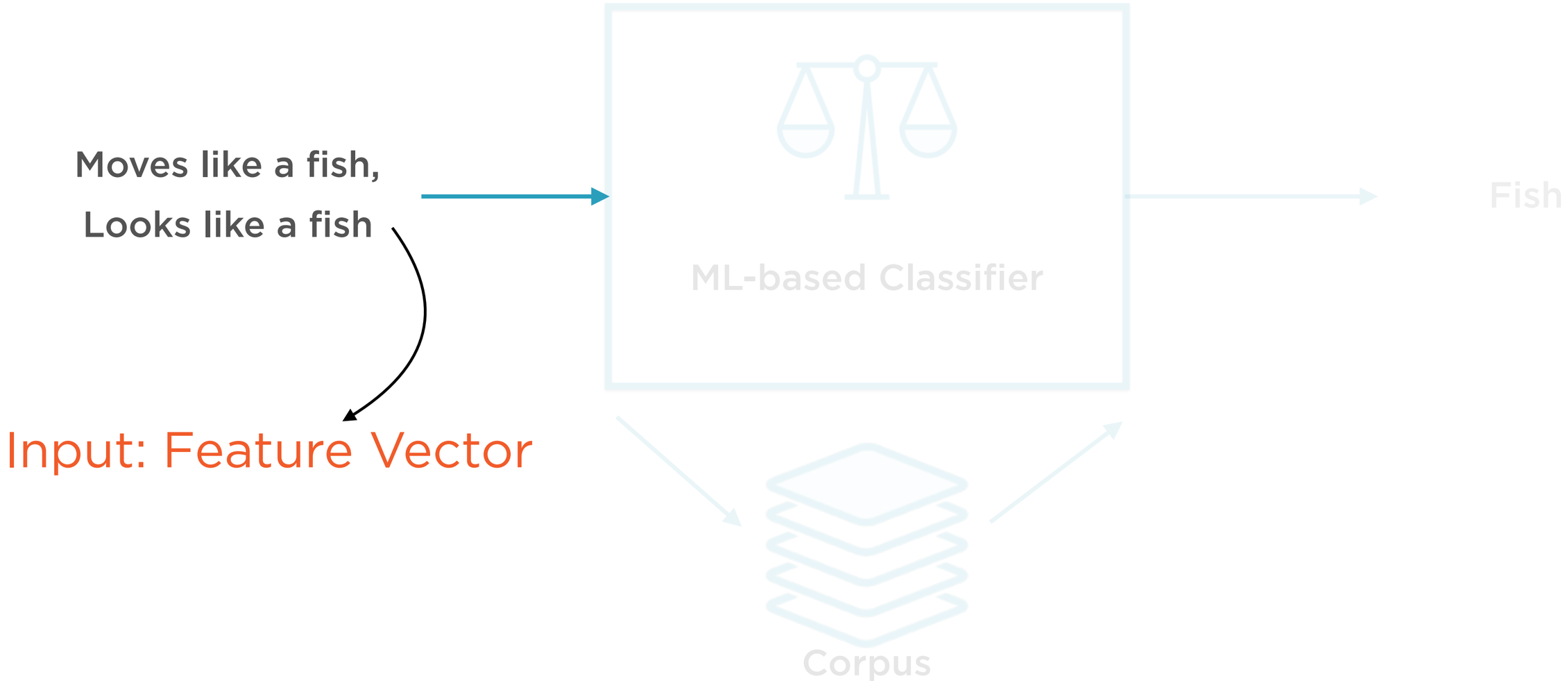
# “Traditional” ML-based Binary Classifier



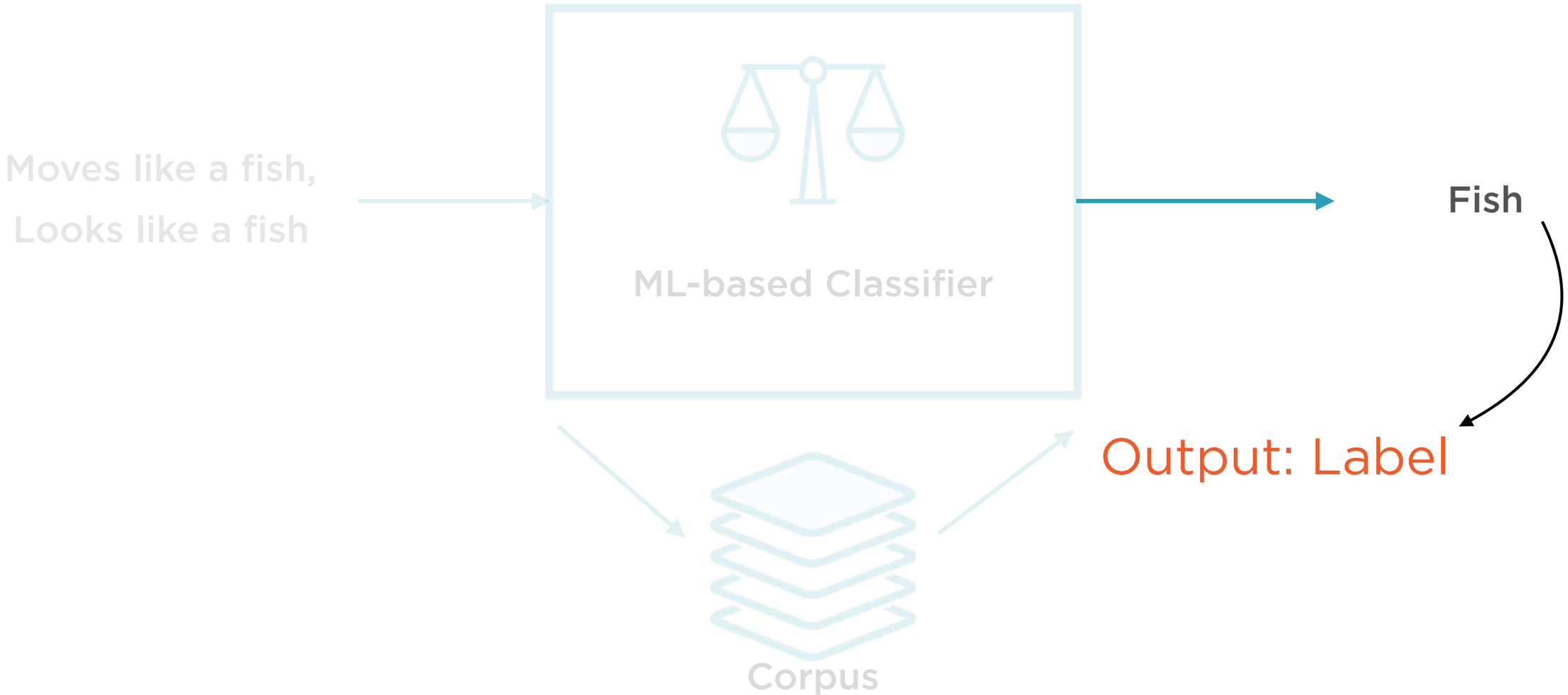
# “Traditional” ML-based Binary Classifier



# “Traditional” ML-based Binary Classifier



# “Traditional” ML-based Binary Classifier



# Feature Vectors

The attributes that the ML algorithm focuses on are called features

Each data point is a list - or vector - of such features

Thus, the input into an ML algorithm is a feature vector

“Traditional” ML-based systems still rely on experts to decide what features to pay attention to

“Representation” ML-based systems figure out by themselves what features to pay attention to

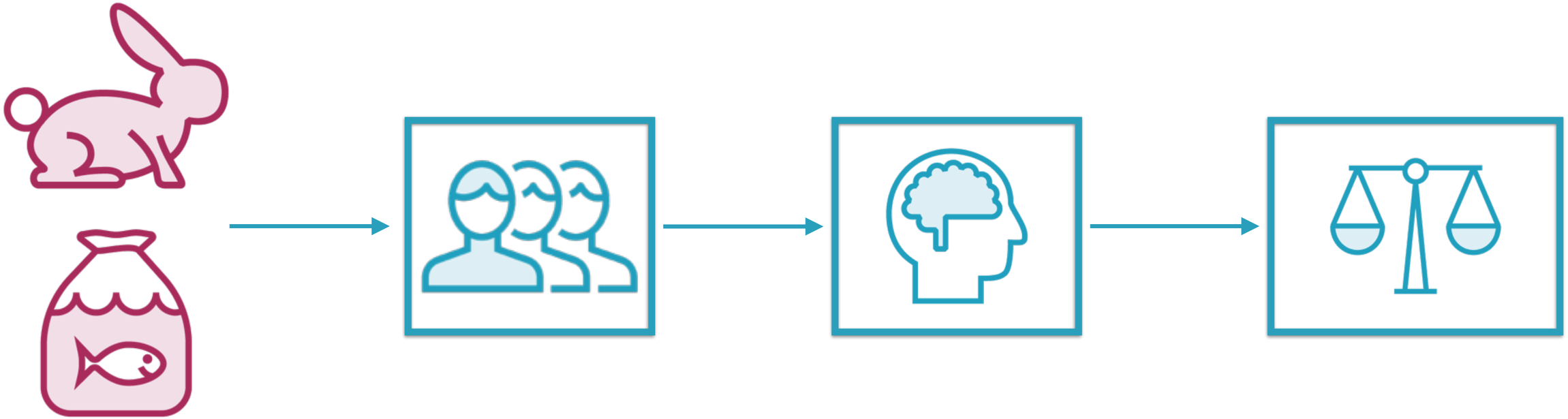
# “Traditional” ML-based Binary Classifier

**Corpus**

**Classification Algorithm**

**ML-based Classifier**

# “Traditional” ML-based Binary Classifier



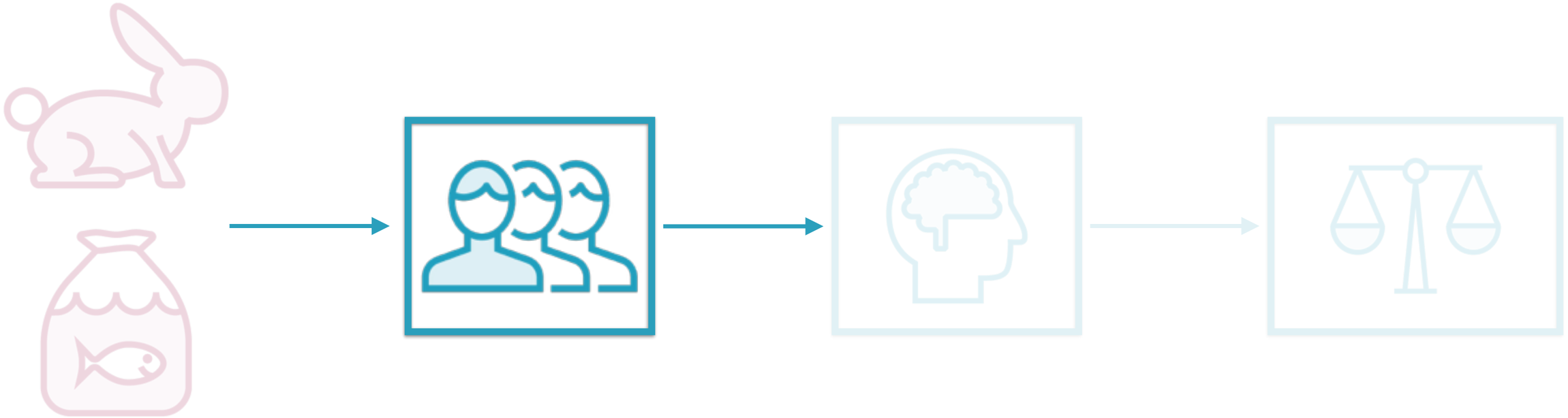
**Corpus**

**Feature Selection by Experts**

**Classification Algorithm**

**ML-based Classifier**

# “Traditional” ML-based Binary Classifier



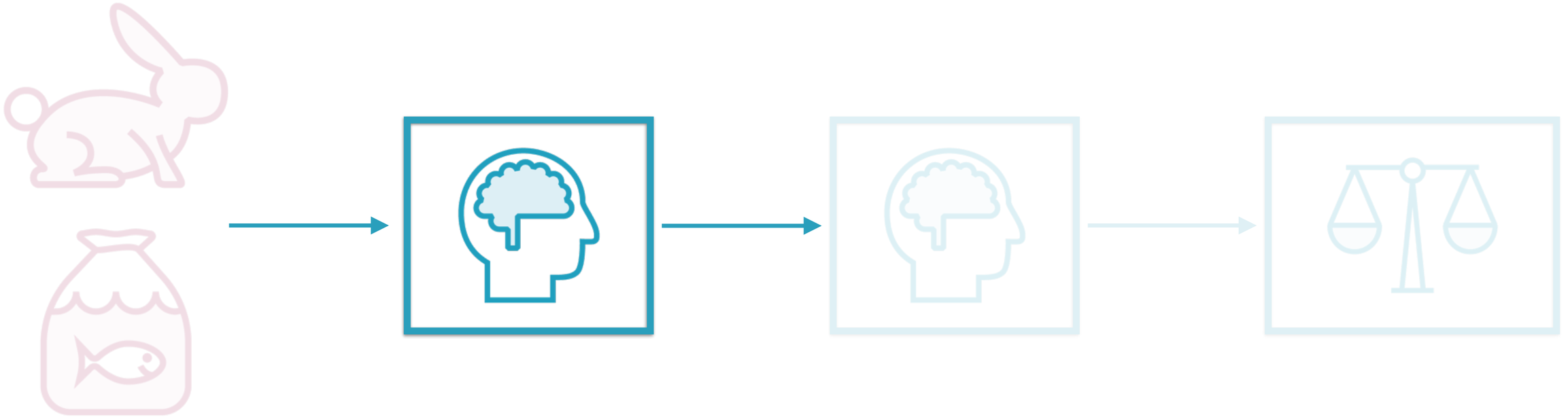
Corpus

**Feature Selection by Experts**

Classification Algorithm

ML-based Classifier

# “Representation” ML-based Binary Classifier



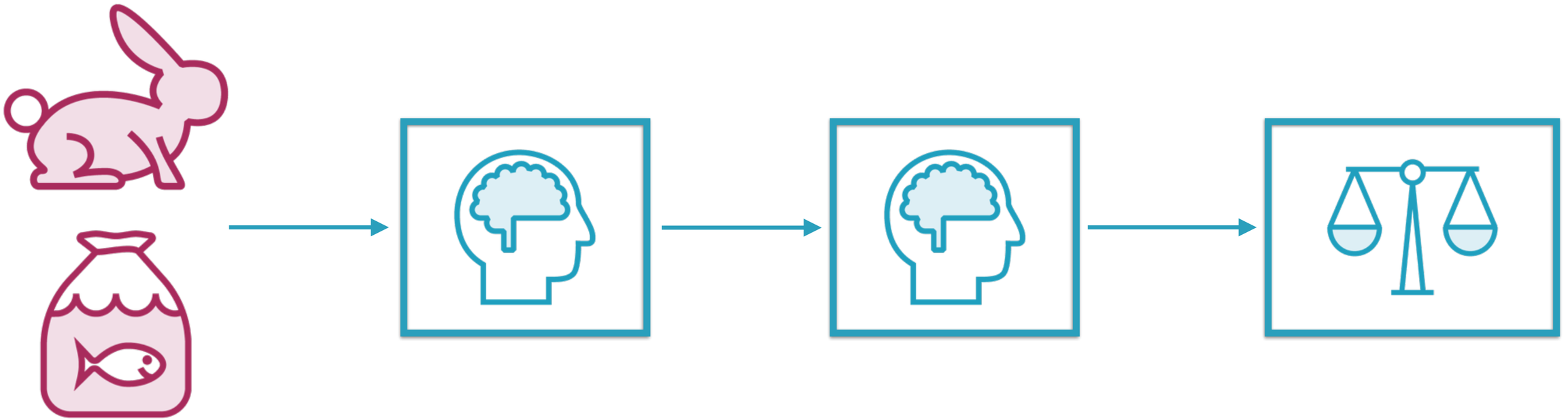
Corpus

**Feature Selection Algorithm**

Classification Algorithm

ML-based Classifier

# “Representation” ML-based Binary Classifier



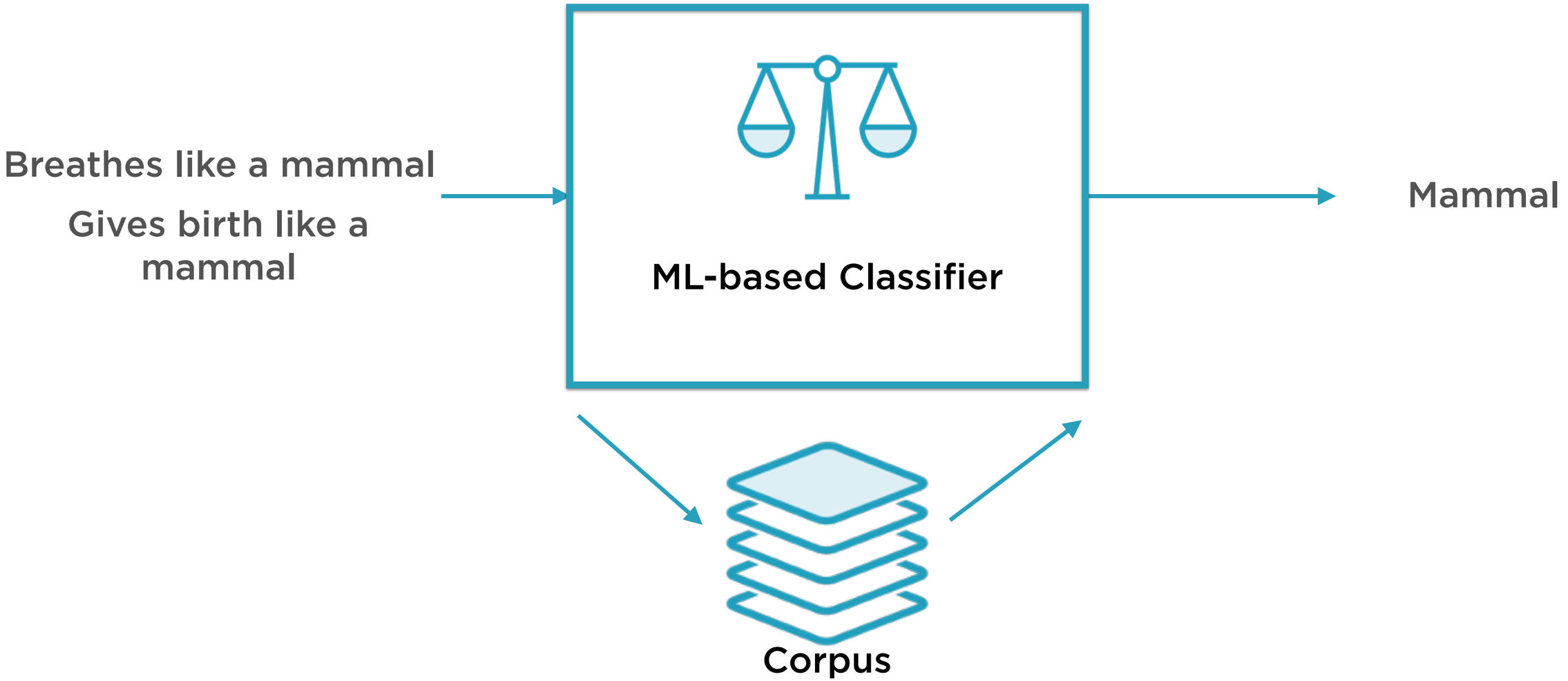
**Corpus**

**Feature Selection Algorithm**

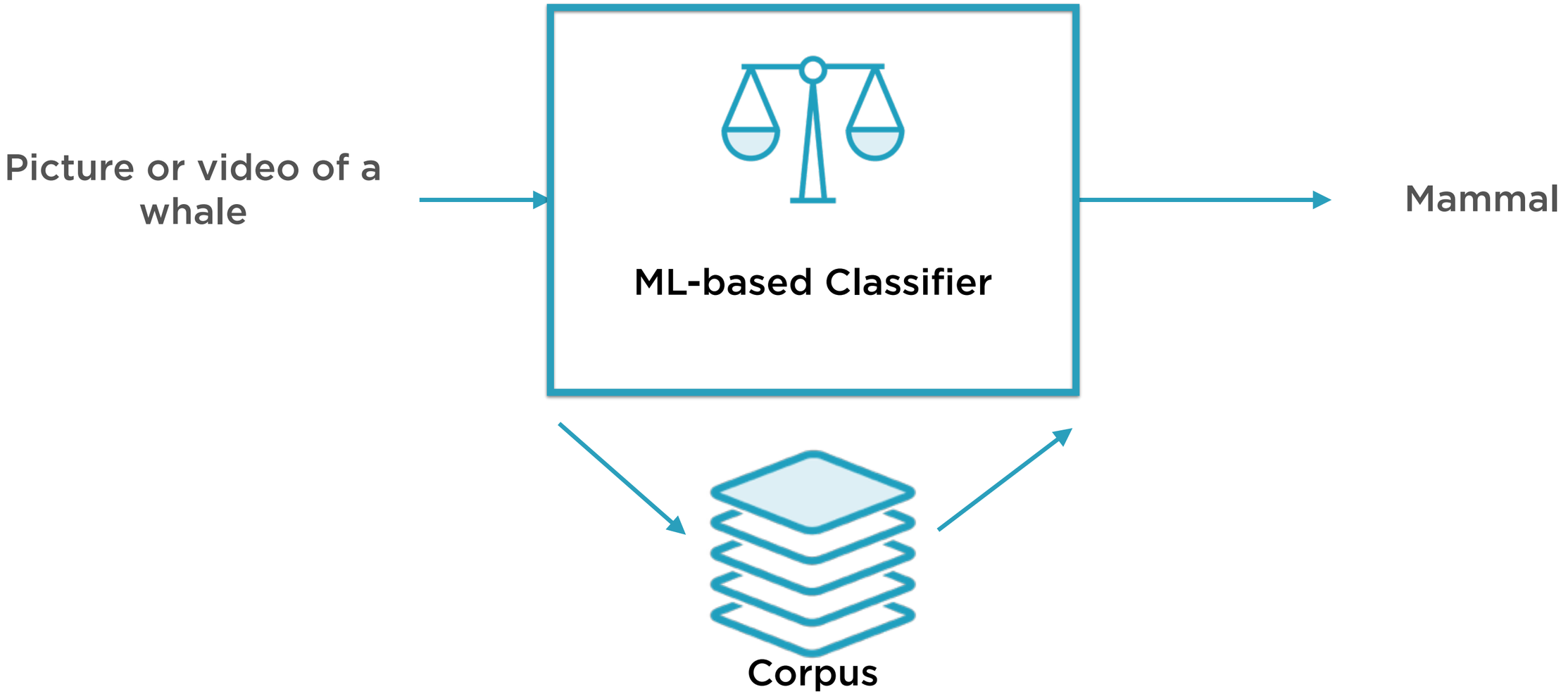
**Classification Algorithm**

**ML-based Classifier**

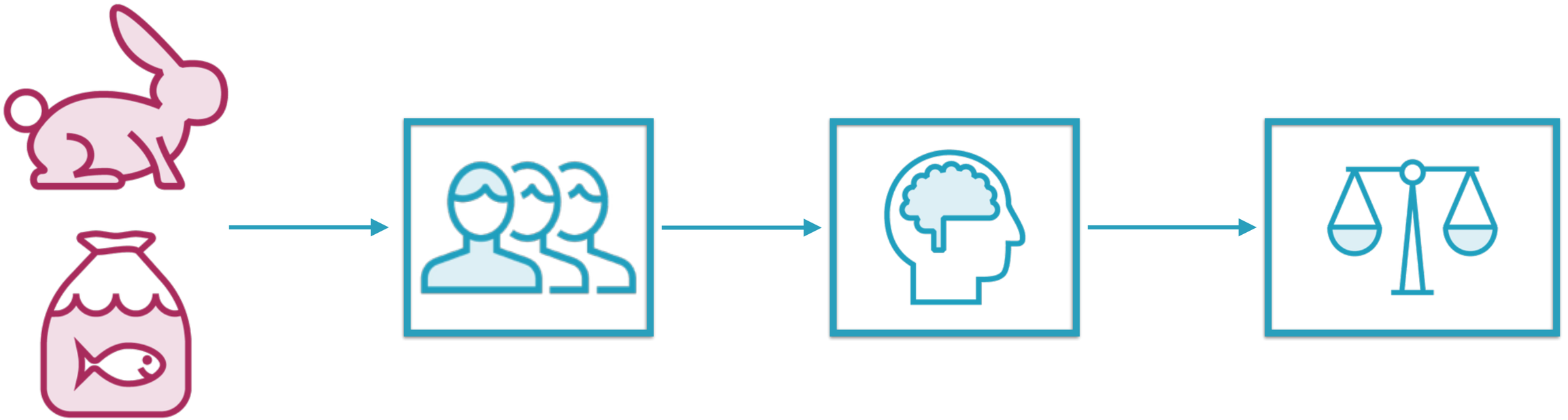
# “Traditional” ML-based Binary Classifier



# “Representation” ML-based Binary Classifier



# “Traditional” ML-based Binary Classifier



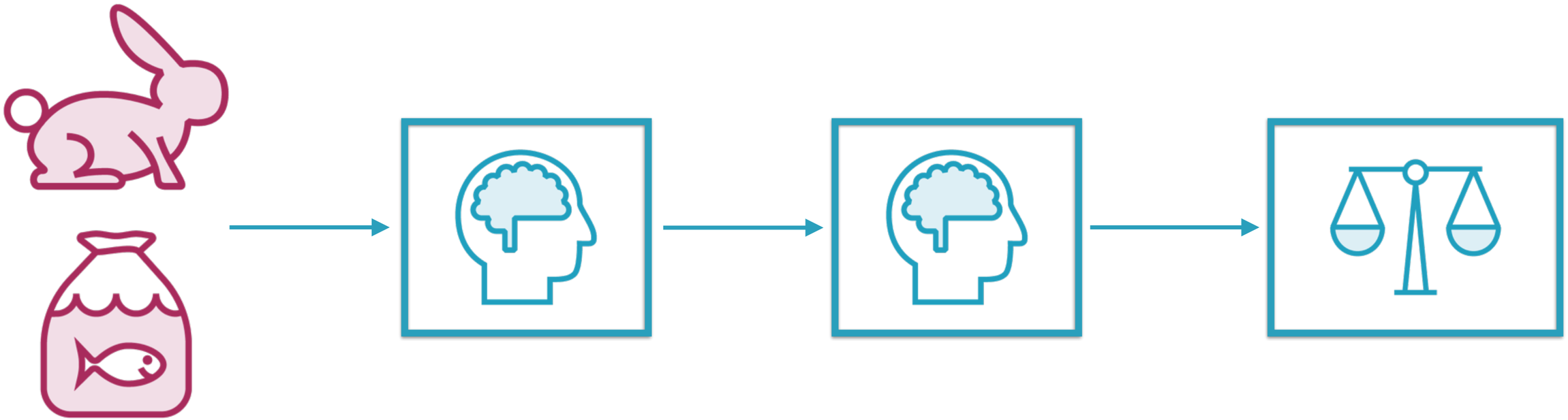
Corpus

**Feature Selection by Experts**

Classification Algorithm

ML-based Classifier

# “Representation” ML-based Binary Classifier



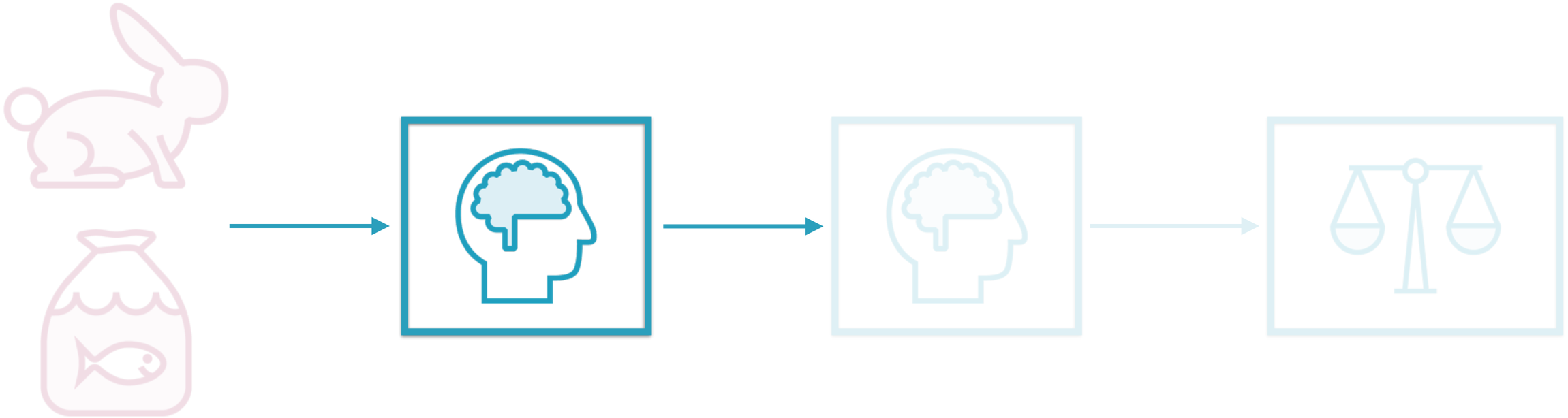
Corpus

**Feature Selection Algorithm**

Classification Algorithm

ML-based Classifier

# “Representation” ML-based Binary Classifier



Corpus

**Feature Selection Algorithm**

Classification Algorithm

ML-based Classifier

“Deep Learning” systems are one type of representation systems

# Deep Learning and Neural Networks

## Deep Learning

Algorithms that learn what features matter

## Neural Networks

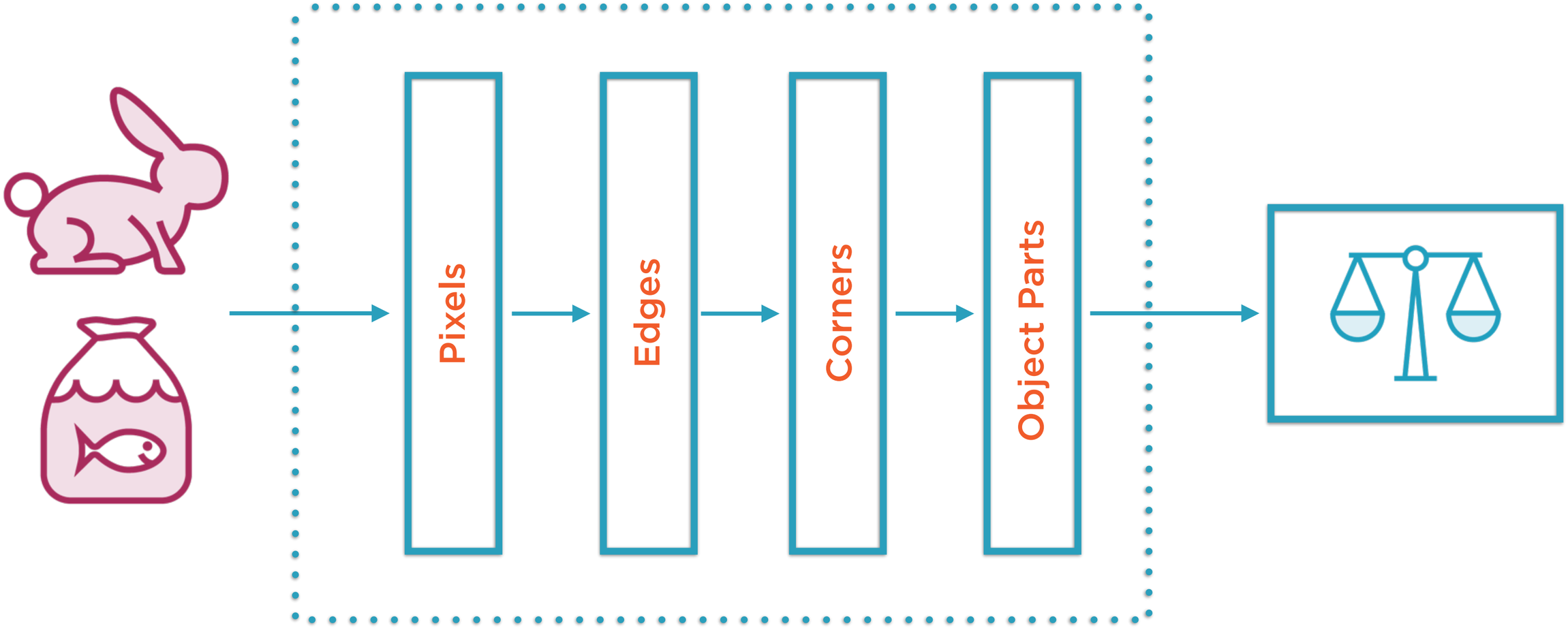
The most common class of deep learning algorithms

## Neurons

Simple building blocks that actually "learn"

Deep Learning Book - Chapter 1 (intro), page 6

# “Deep Learning”-based Binary Classifier

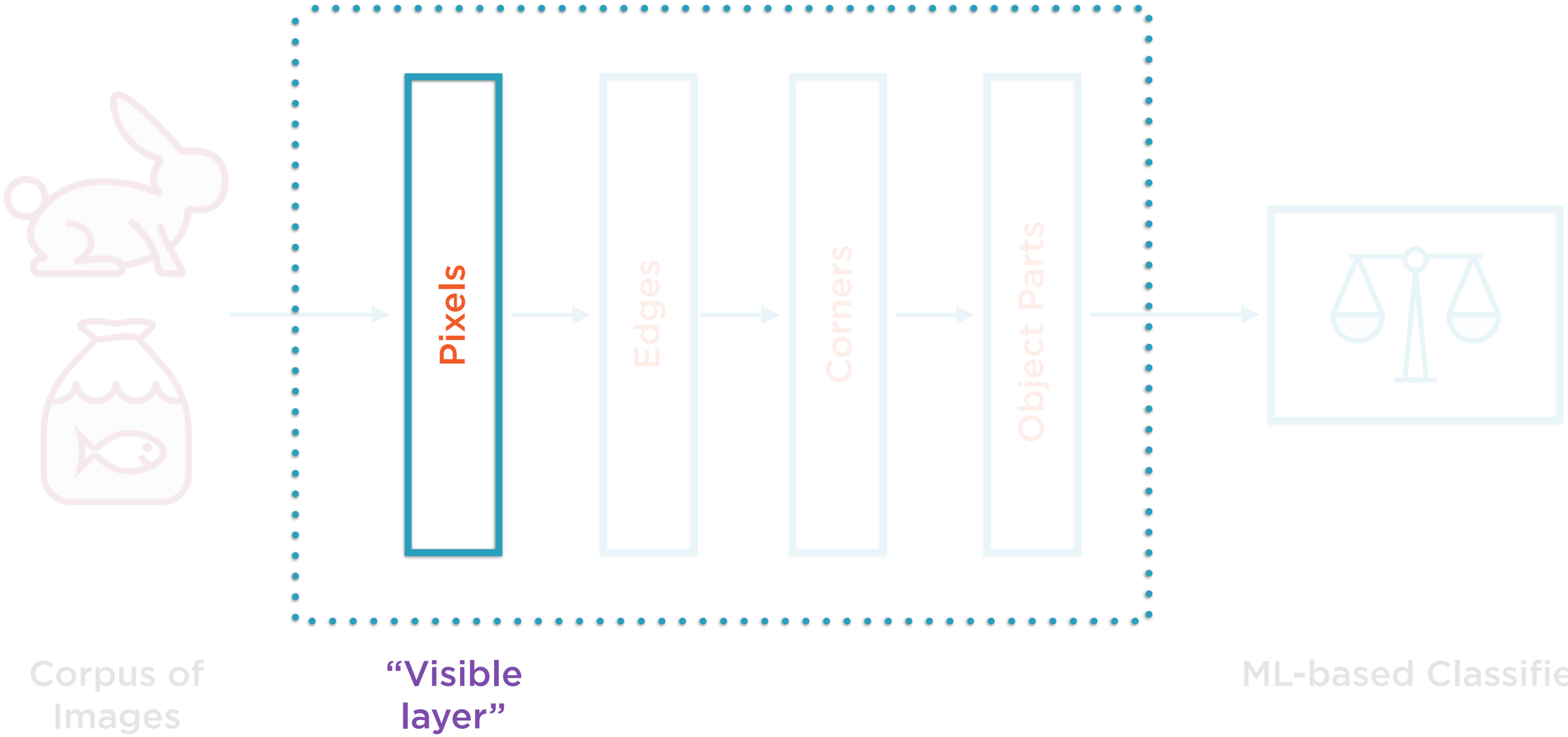



Corpus of Images

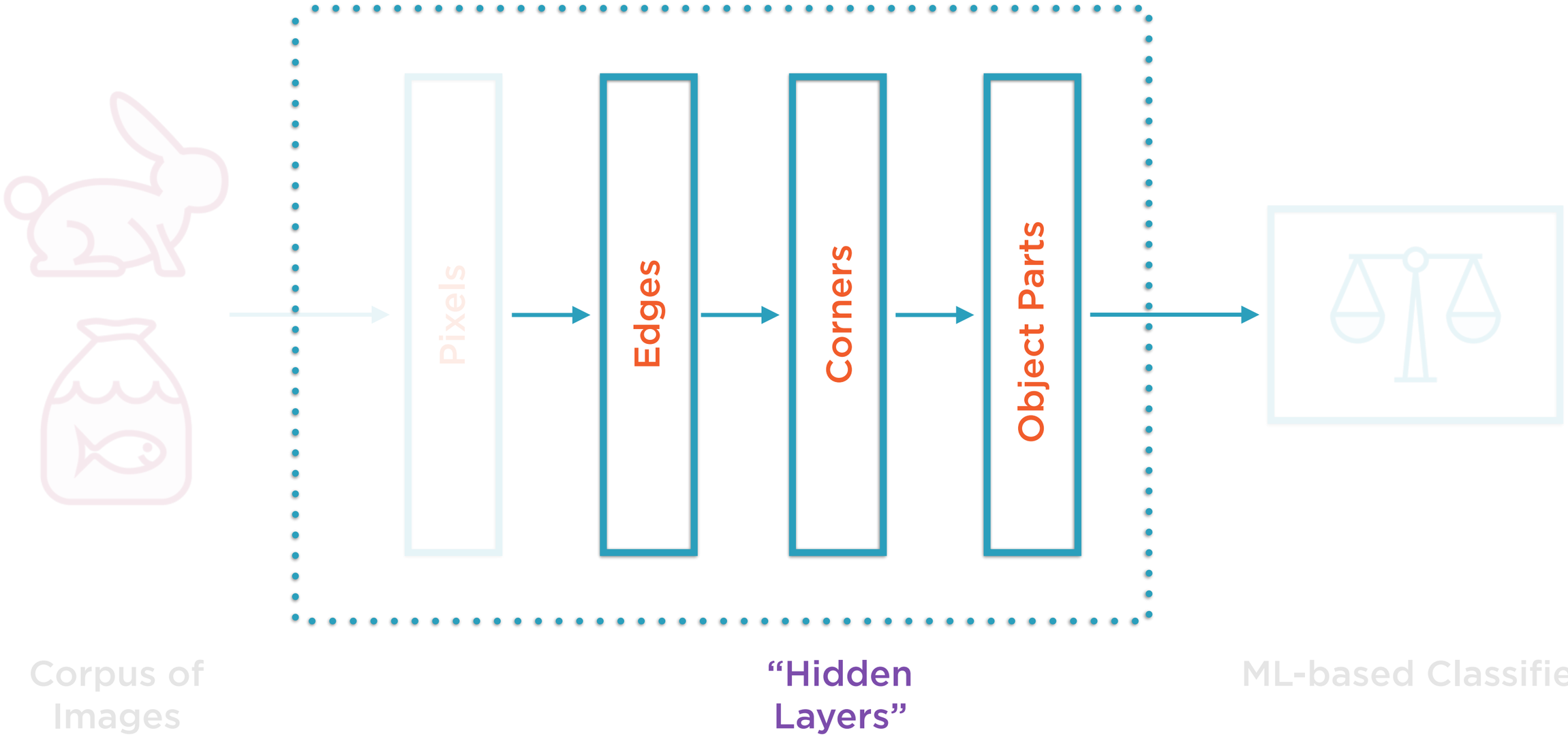
Feature Selection & Classification Algorithm

ML-based Classifier

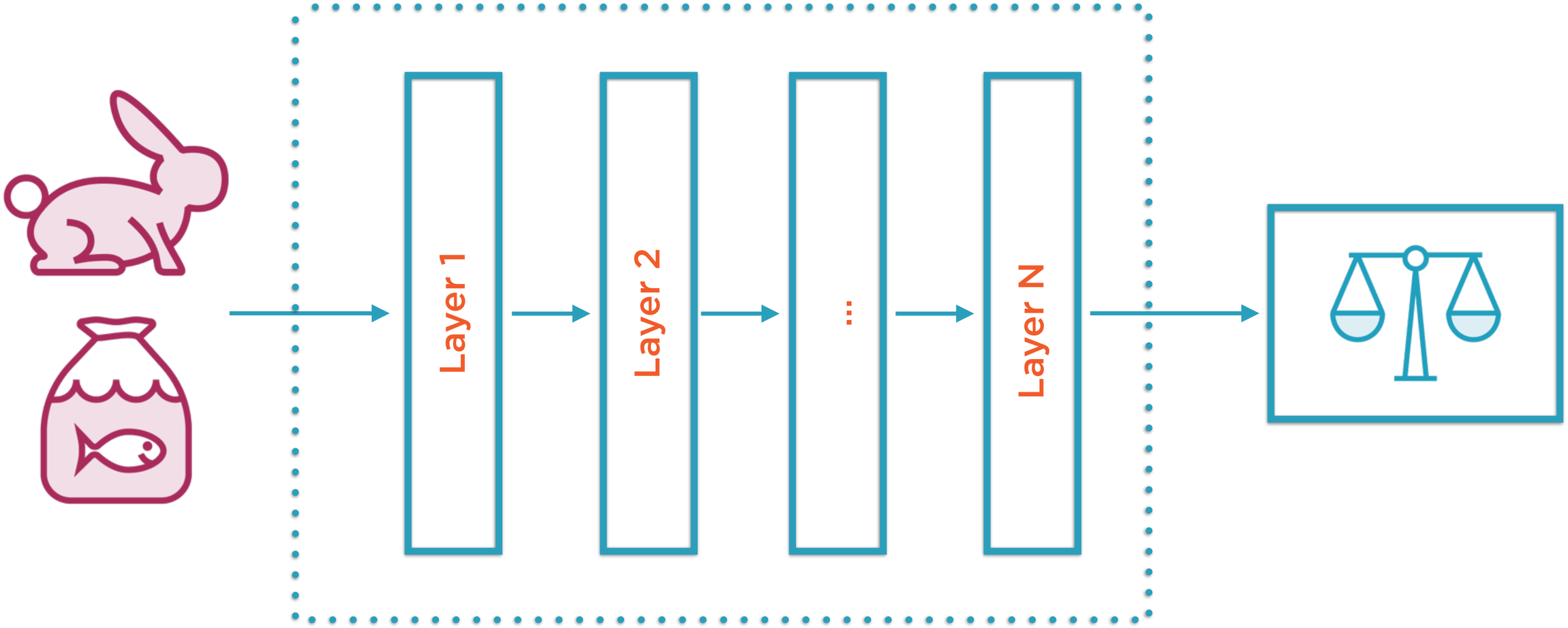
# “Deep Learning”-based Binary Classifier



# “Deep Learning”-based Binary Classifier



# Neural Networks Introduced

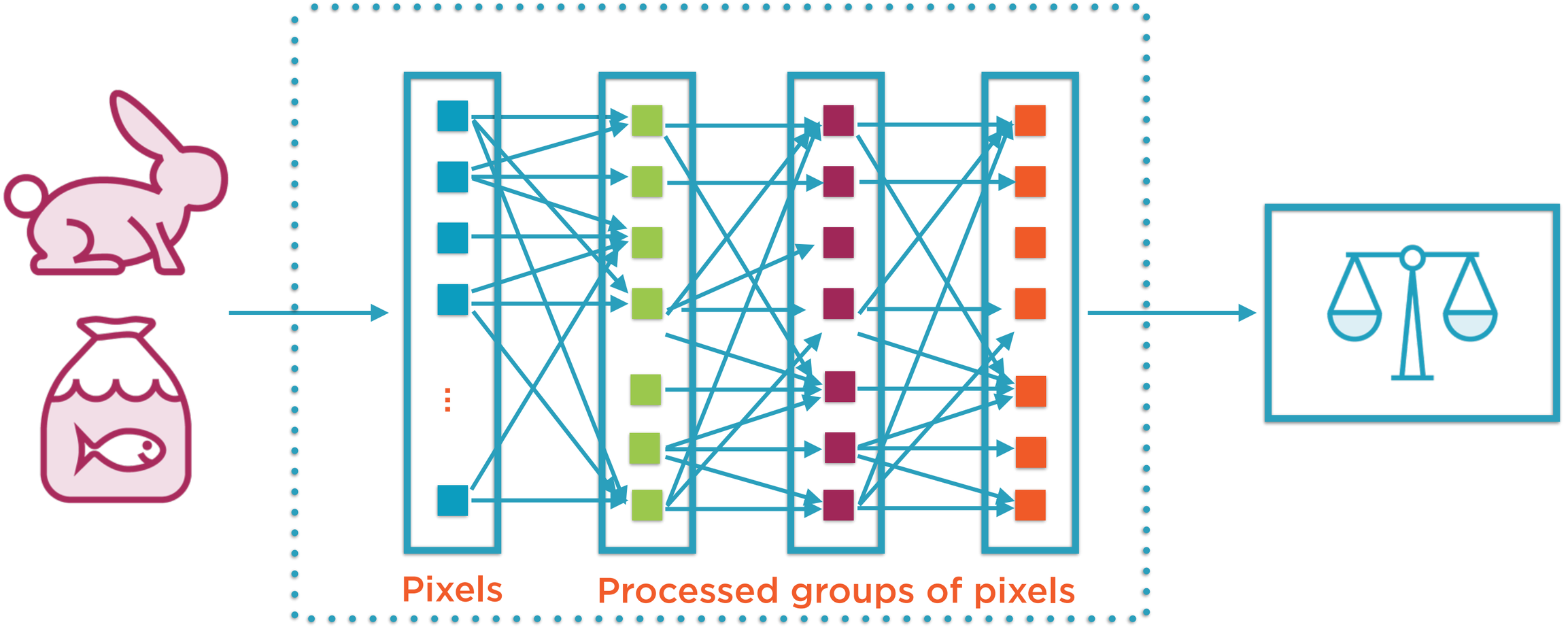



Corpus of Images

Layers in a neural network

ML-based Classifier

# Neural Networks Introduced




Corpus of Images

Each layer consists of individual interconnected neurons

ML-based Classifier

# Neurons as Learning Units

A machine learning algorithm is an algorithm that is able to learn from data

# Learning Algorithms

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E

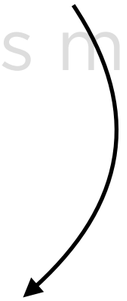
# Learning Algorithms

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E

Most common tasks in ML:
Classification, regression

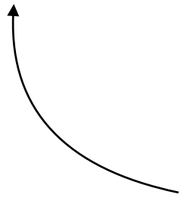
# Learning Algorithms

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E



Accuracy in classification,
residual variance in regression

Training using a corpus of labelled instances



# Learning Algorithms

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E

# Learning Algorithms

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E

# Deep Learning and Neural Networks

## Deep Learning

Algorithms that learn what features matter

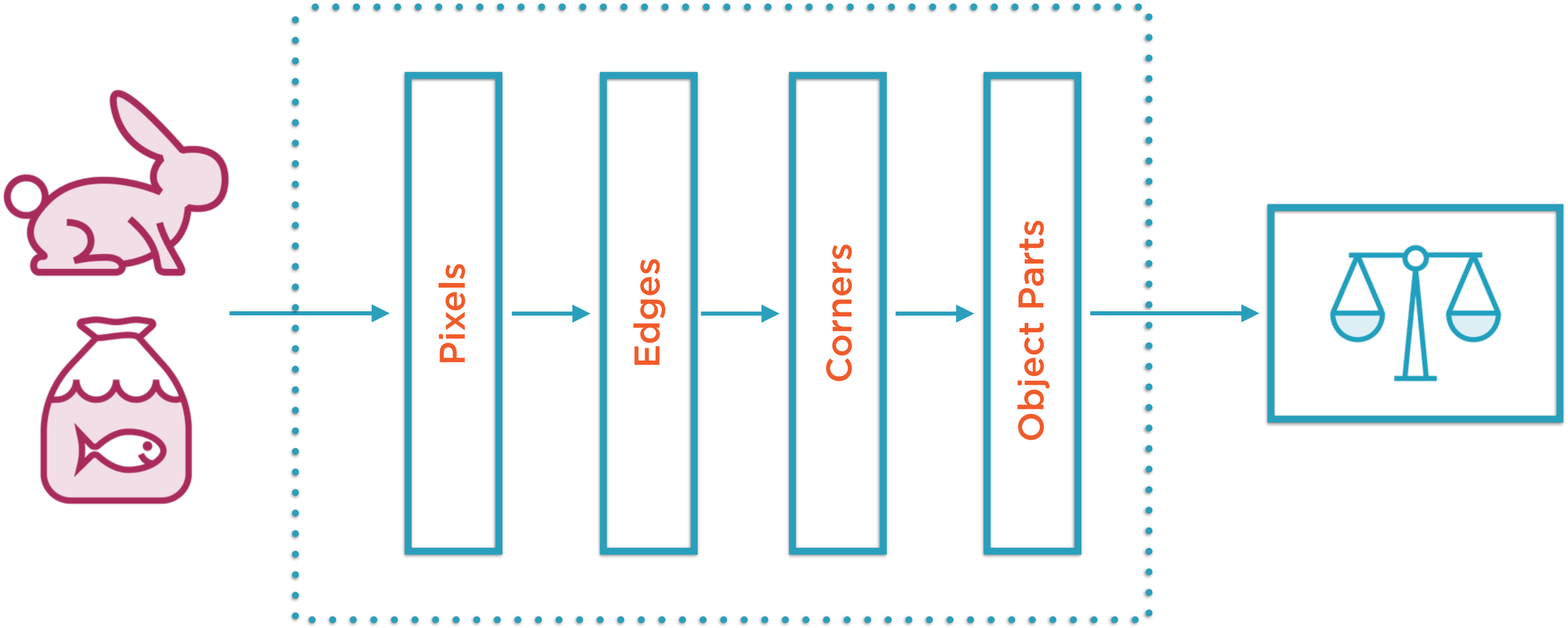
## Neural Networks

The most common class of deep learning algorithms

## Neurons

Simple building blocks that actually "learn"

# “Deep Learning”-based Binary Classifier

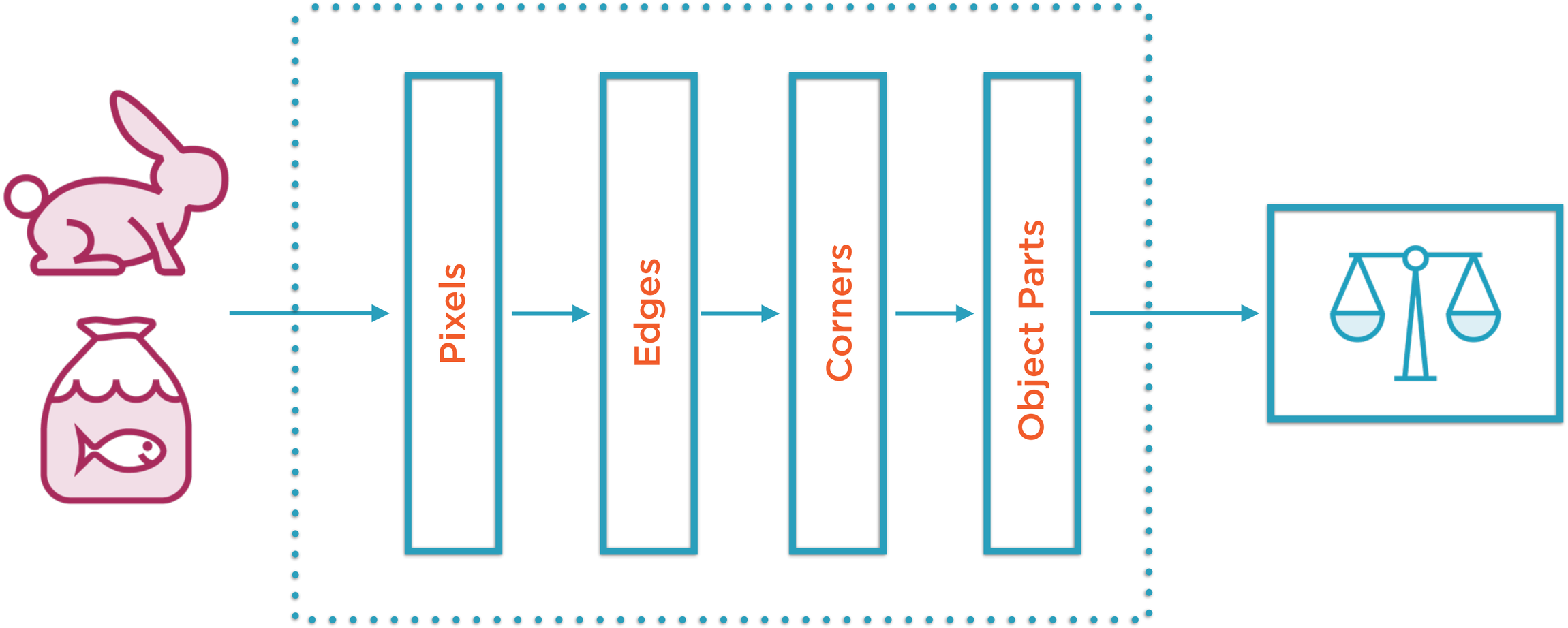



Corpus of Images

Feature Selection & Classification Algorithm

ML-based Classifier

# Layers in the Computation Graph

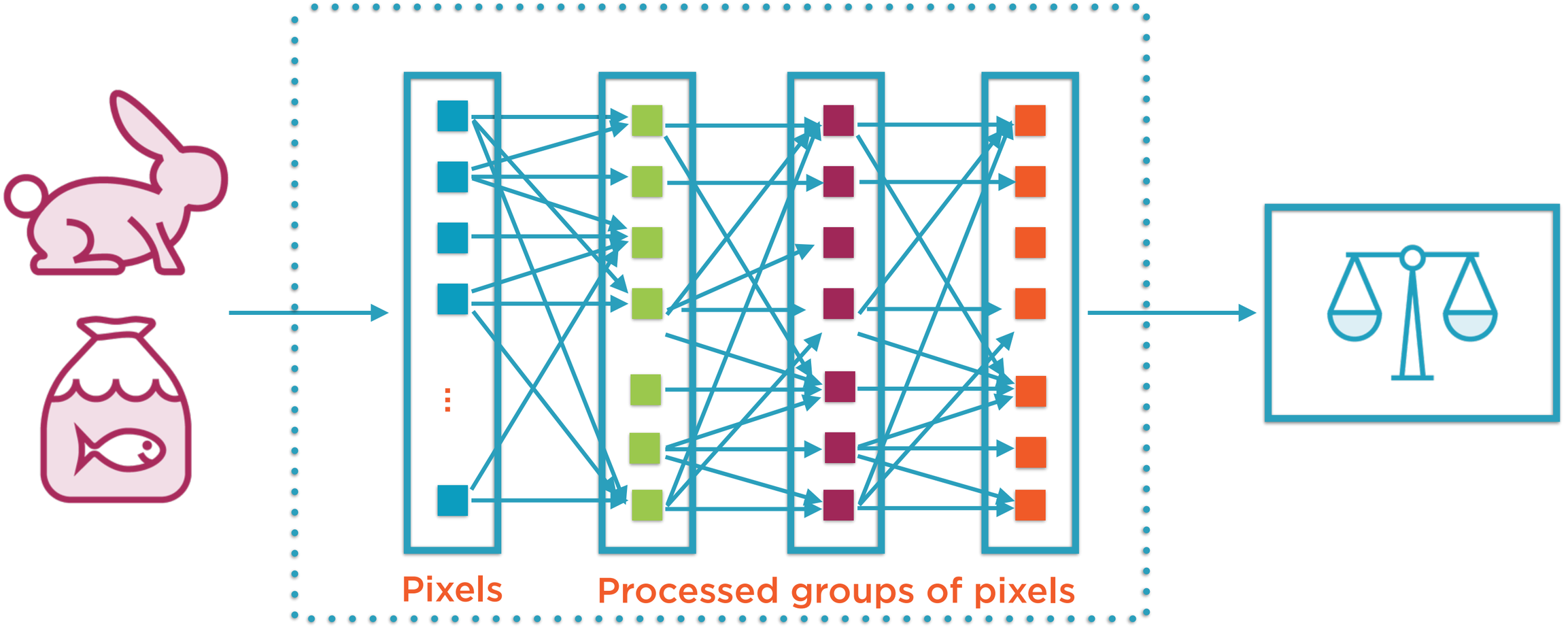



Corpus of Images

Groups of neurons that perform similar functions are aggregated into layers

ML-based Classifier

# Neural Networks: Networks of Neurons




Corpus of Images

Each layer consists of individual interconnected neurons

ML-based Classifier

# Deep Learning

**Directed computation graphs “learn” relationships between data**

**The more complex the graph, the more the relationships it can “learn”**

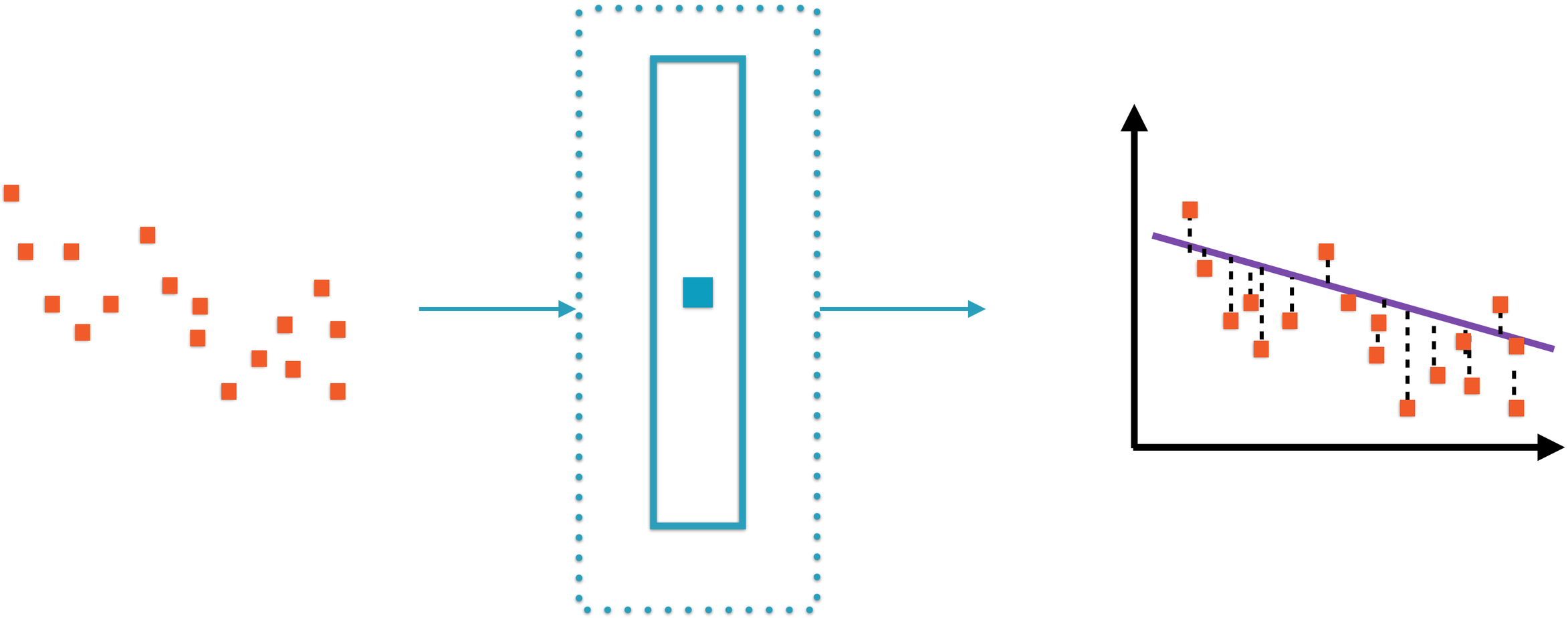
**“Deep” Learning: Depth of the computation graph**

```
y = Wx + b
```

# “Learning” Regression

**Regression can be reverse-engineered by a single neuron**

# Regression: The Simplest Neural Network



Set of Points

Single Neuron

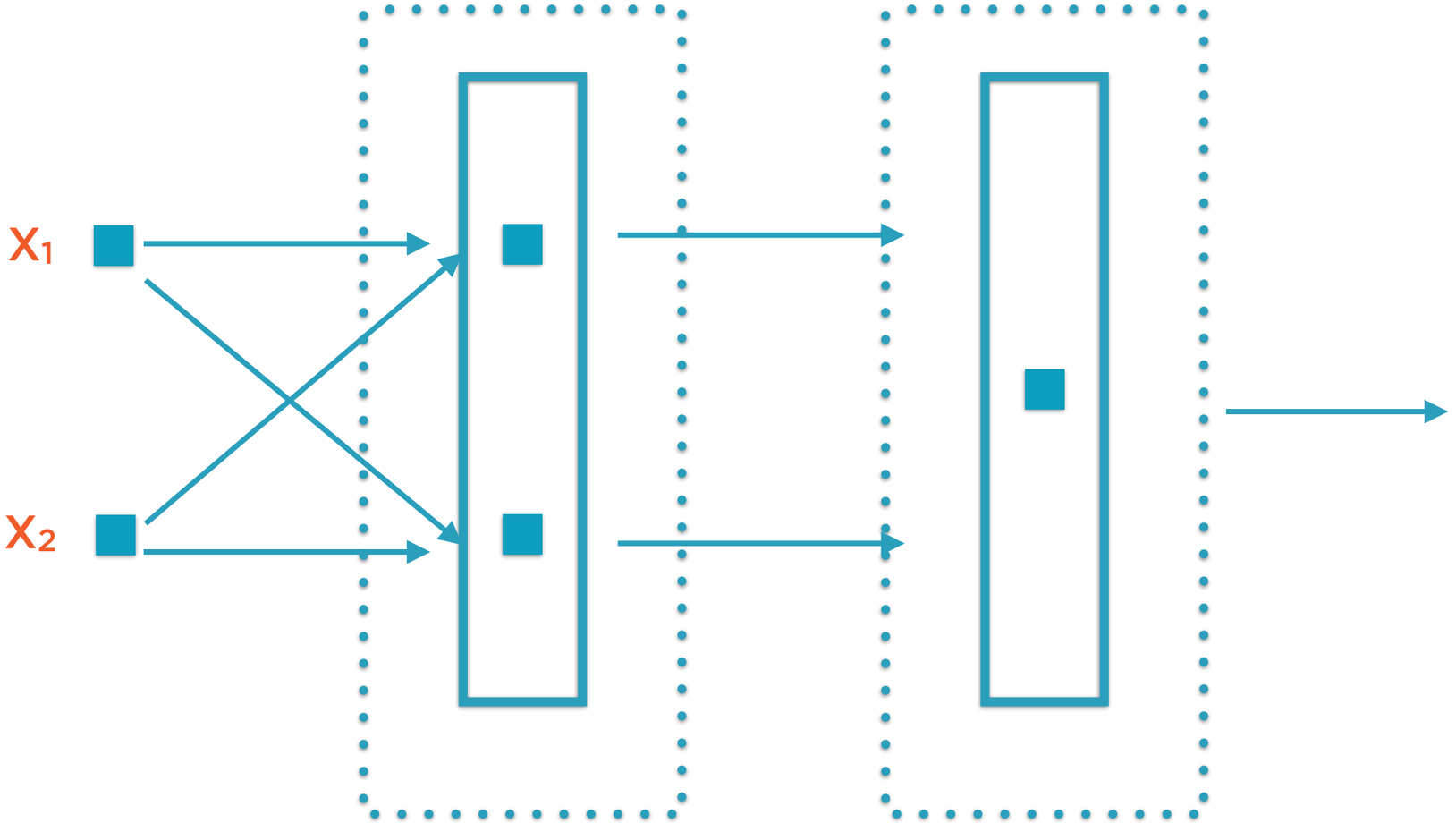
Regression Line

```
def XOR(x1,x2):
  if (x1 == x2):
     return 0
  return 1
```

# “Learning” XOR

**The XOR function can be reverse-engineered using 3 neurons arranged in 2 layers**

# XOR: 3 Neurons, 2 Layers




| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

2 Inputs

Layer 1

Layer 2

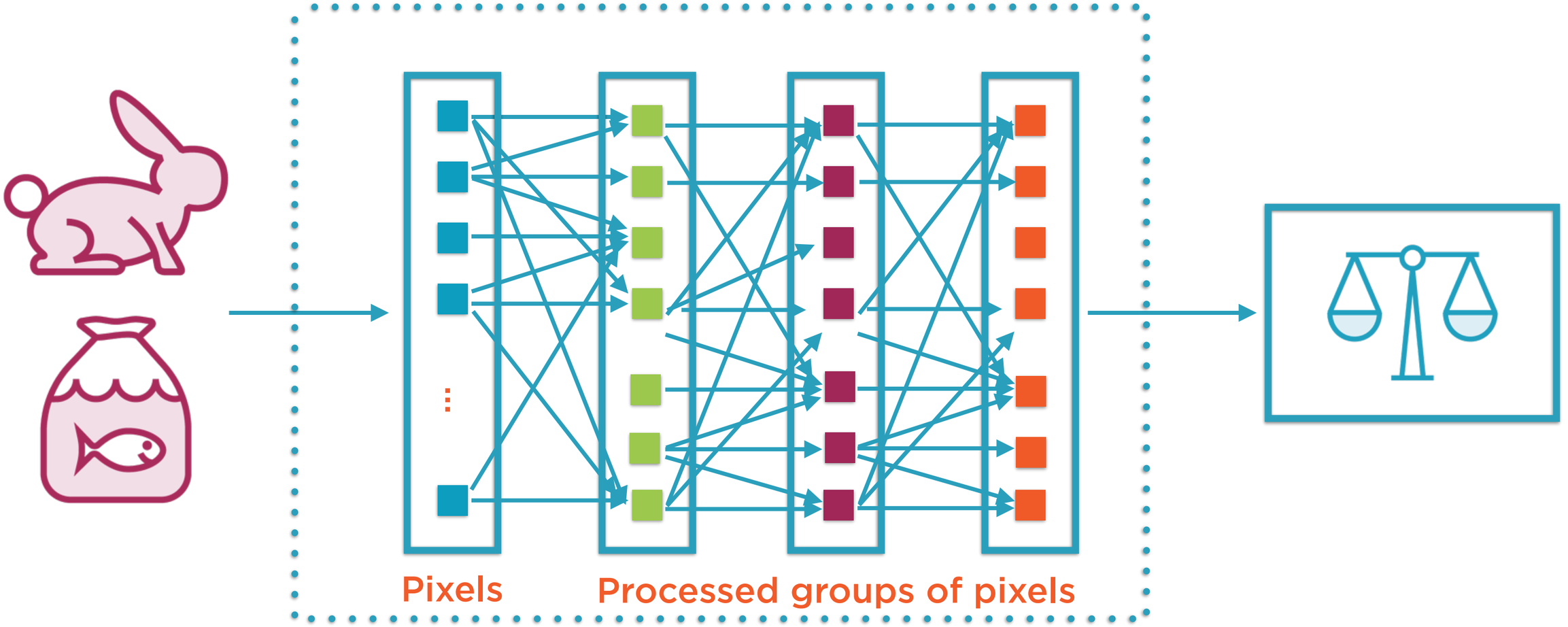
Output Truth Table

```
def doSomethingReallyComplicated(x1,x2…):
   …
   …
   …
   return complicatedResult
```

# “Learning” Arbitrarily Complex Functions

**Adding layers to a neural network can “learn” (reverse-engineer) pretty much anything**

# Arbitrarily Complex Function

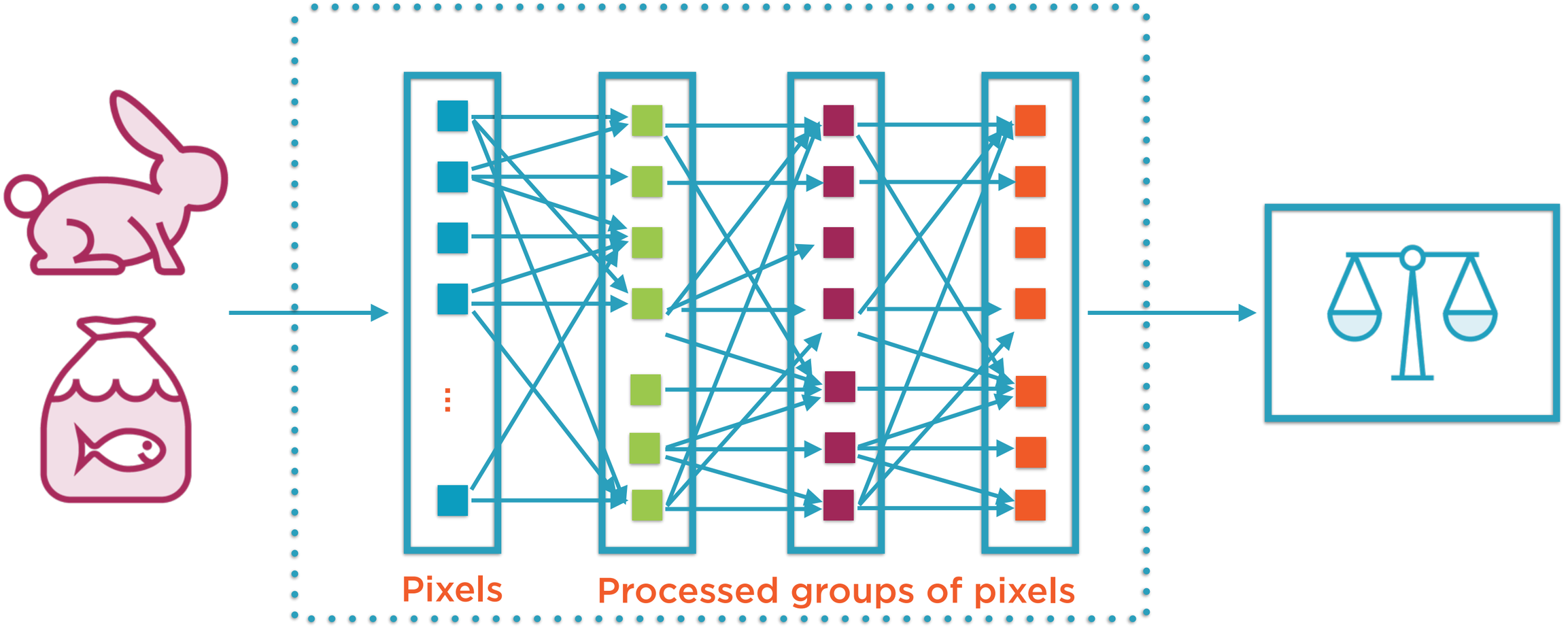



Corpus of Images

Operations (nodes) on data (edges)

ML-based Classifier

# The Computational Graph

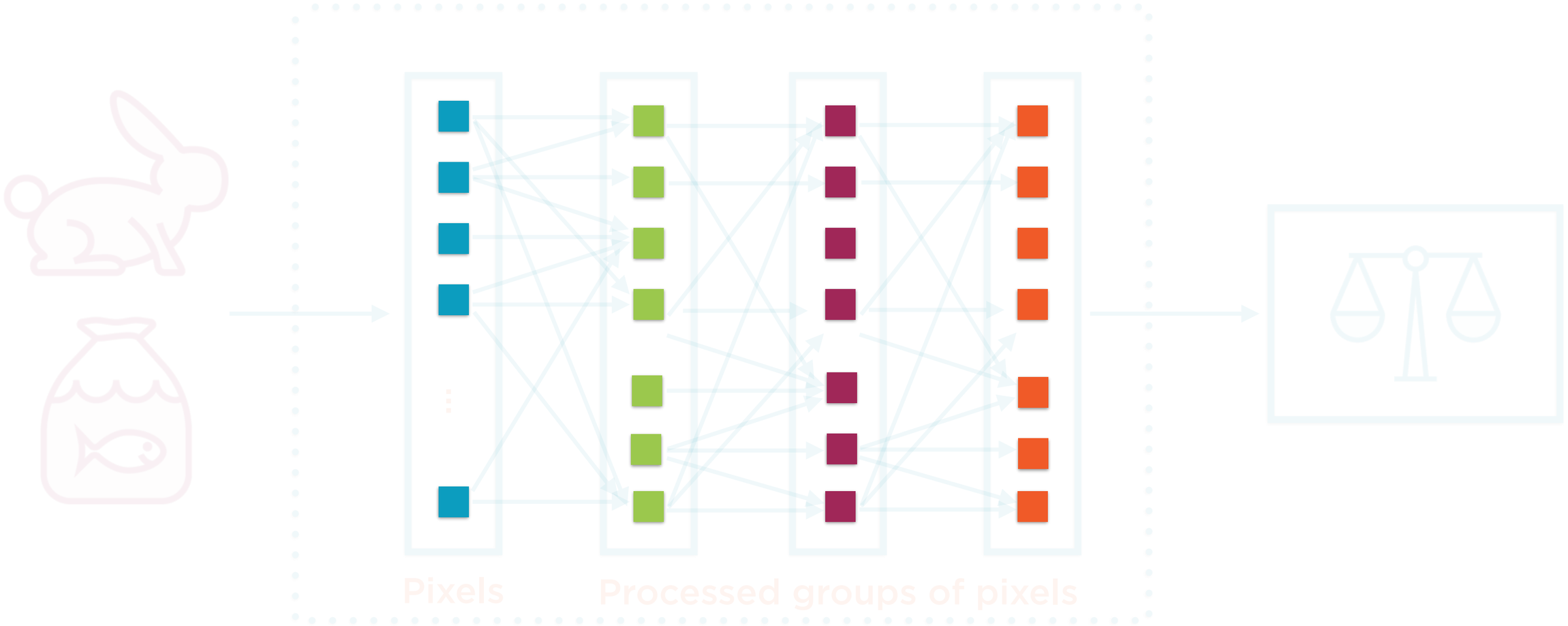



Corpus of Images

Operations (nodes) on data (edges)

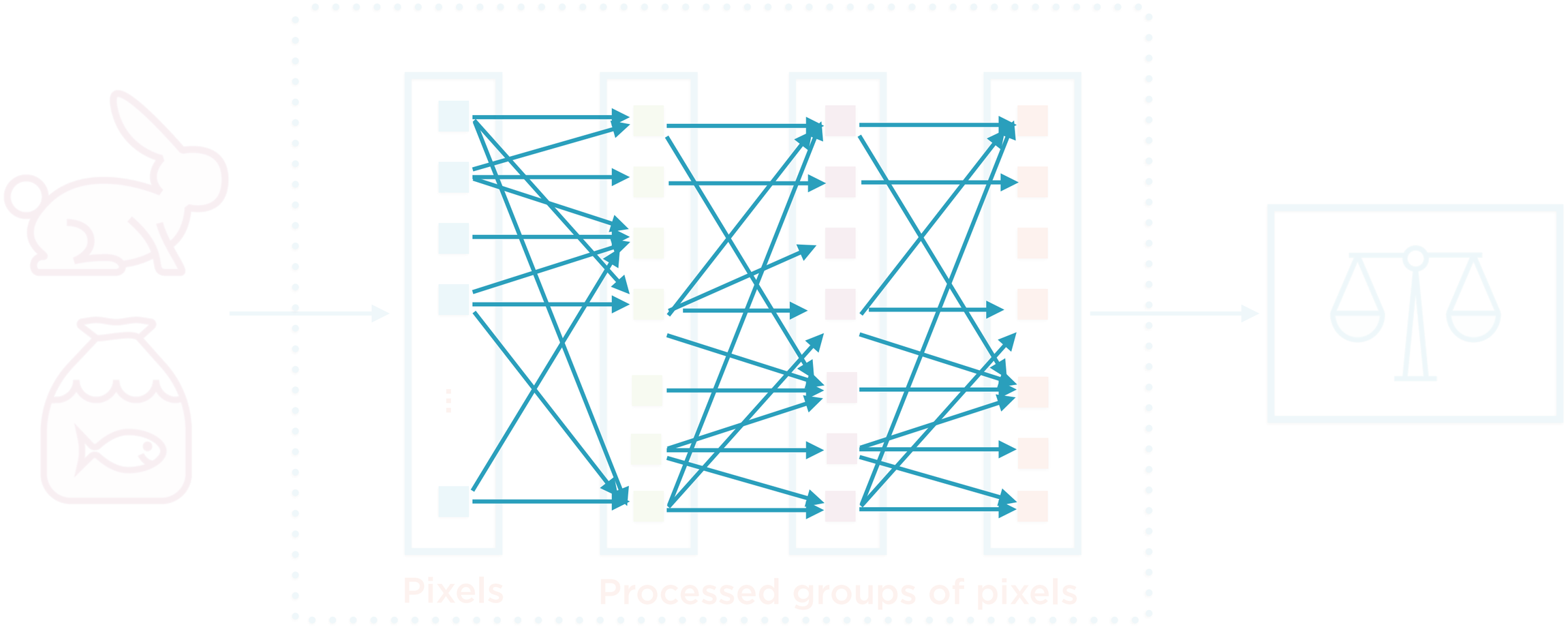
ML-based Classifier

# The Computational Graph




**The nodes in the computation graph are neurons (simple building blocks)**

# The Computational Graph




**The edges in the computation graph are data items called tensors**

# Neurons

**The nodes in the computation graph are simple entities called neurons**

**Each neuron performs very simple operations on data**

**The neurons are connected in very complex, sophisticated ways**

# Neural Networks

**The complex interconnections between simple neurons**

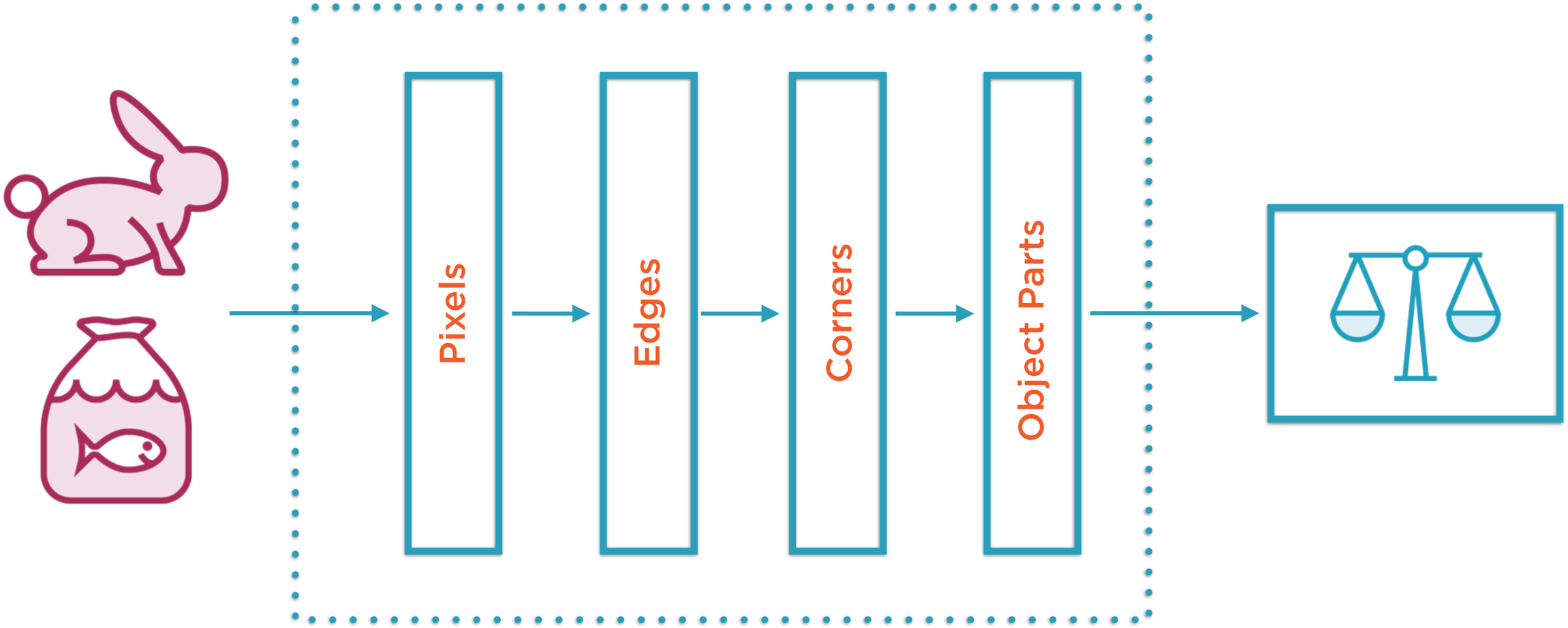
**Different network configurations => different types of neural networks**

- Convolutional
- Recurrent

# Neural Networks

**Groups of neurons that perform similar functions are aggregated into layers**

# Layers in the Computation Graph

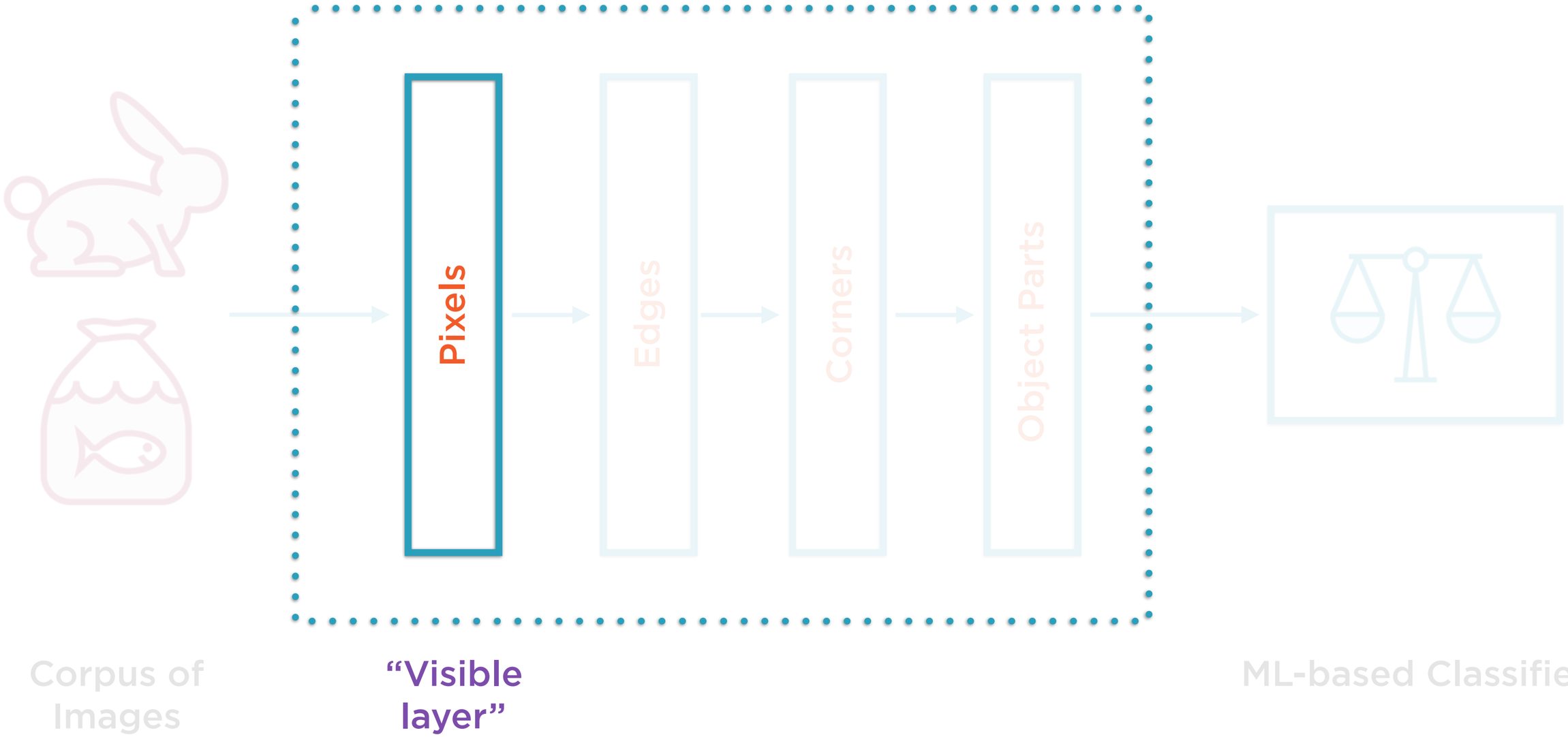



Corpus of Images

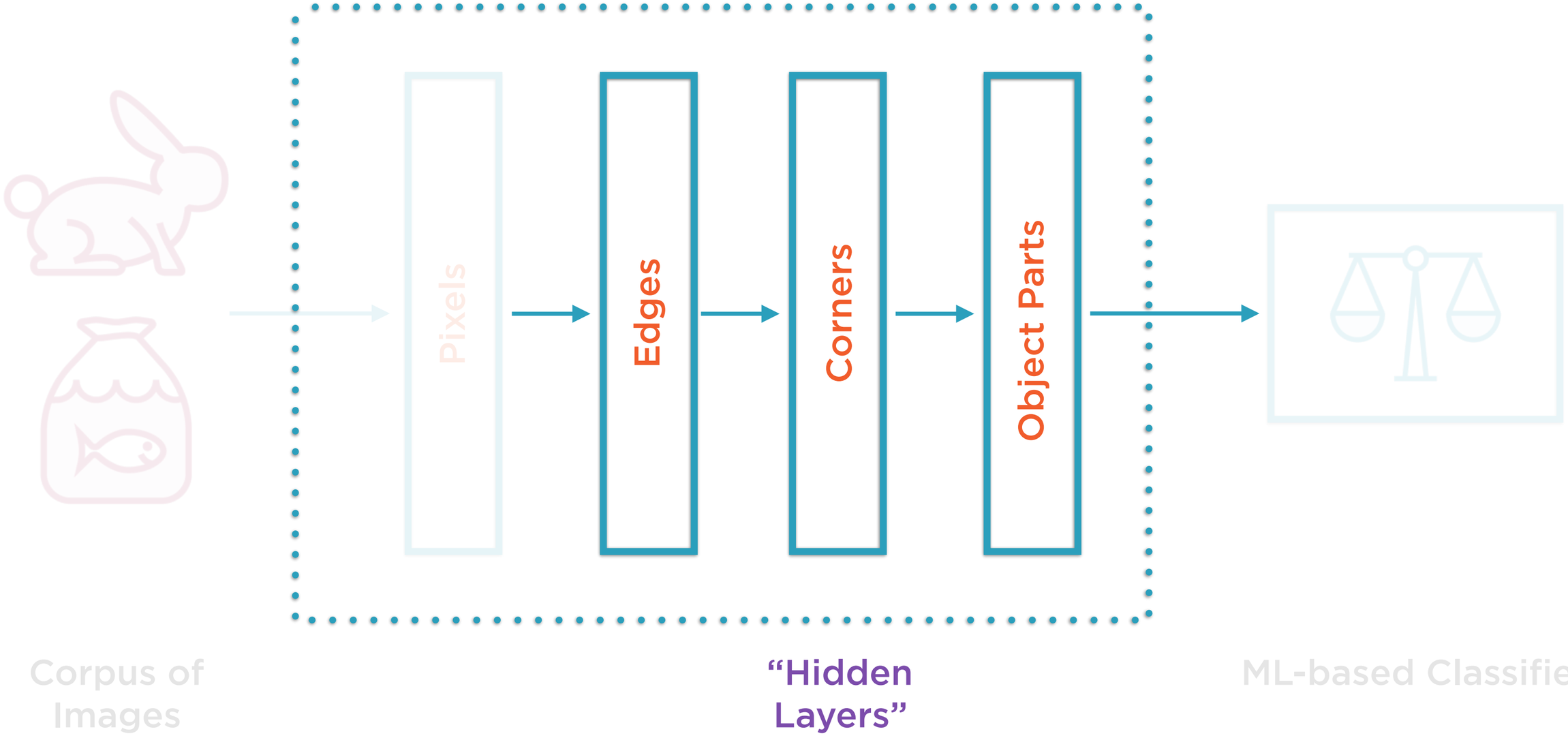
Groups of neurons that perform similar functions are aggregated into layers

ML-based Classifier

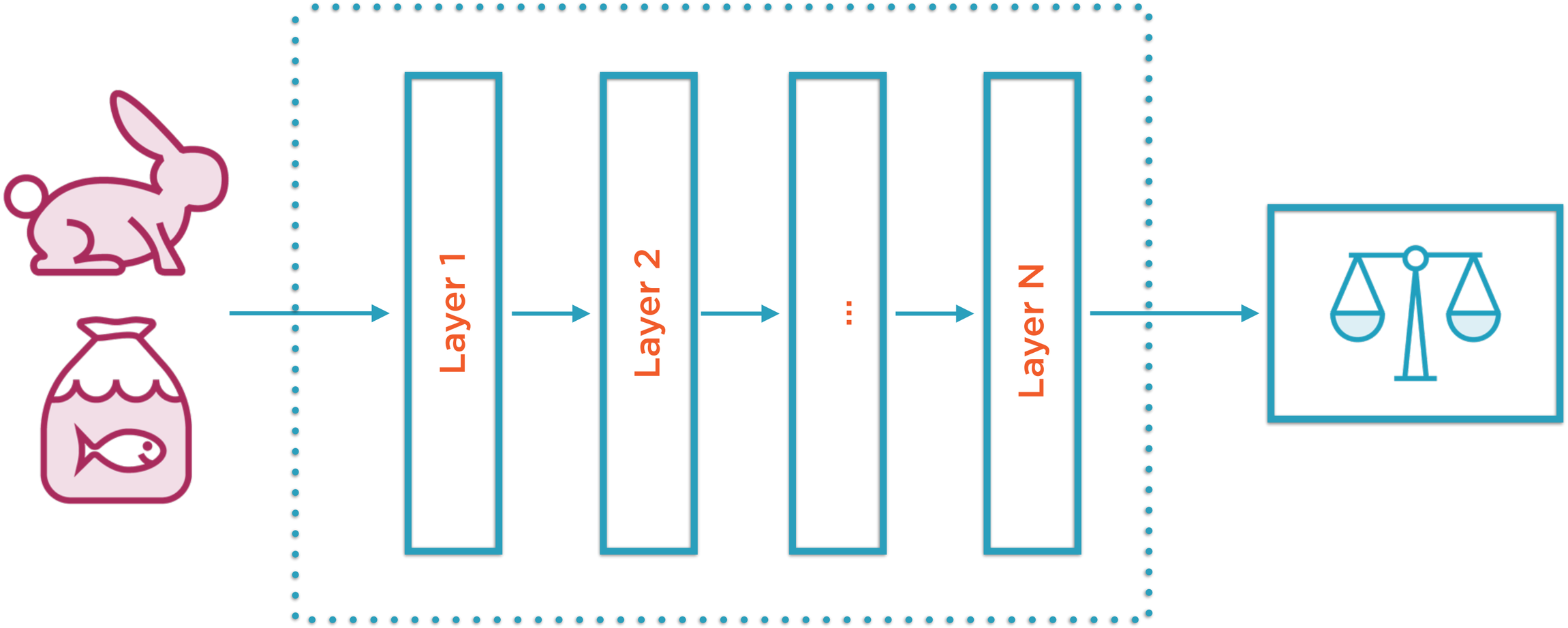
# Layers in the Computation Graph



# Layers in the Computation Graph



# Layers in the Computation Graph




Corpus of Images

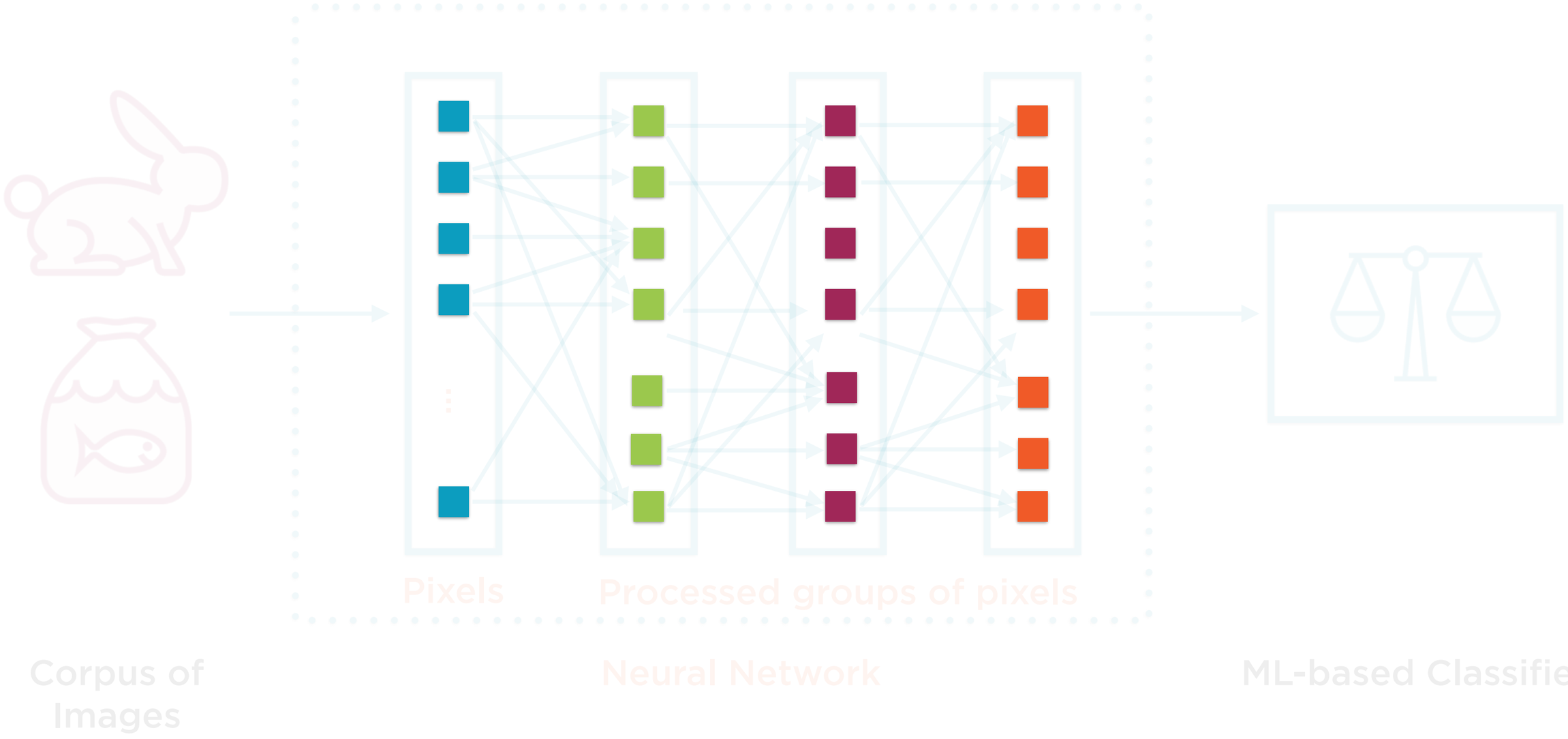
Layers in a neural network

ML-based Classifier

# Neurons

**Each layer consists of units called neurons**

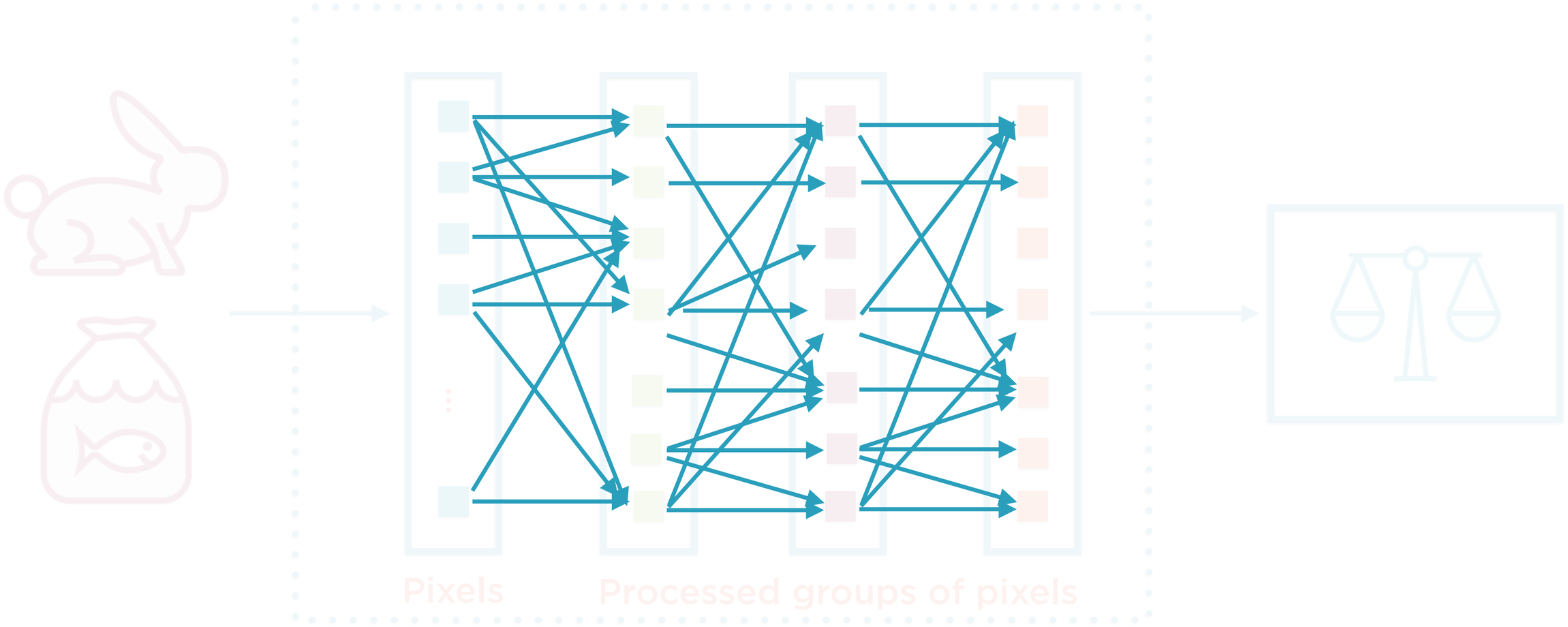
# Neurons


Pixels
Processed groups of pixels
Corpus of Images
Neural Network
ML-based Classifier

# Neural Networks

**Neurons in a neural network can be connected in very complex ways...**

# Neural Networks Introduced




**Neurons in a neural network can be connected in very complex ways…**

# Neural Networks

**Neurons in a neural network can be connected in very complex ways...**

**...But each neuron only applies two simple functions to its inputs**

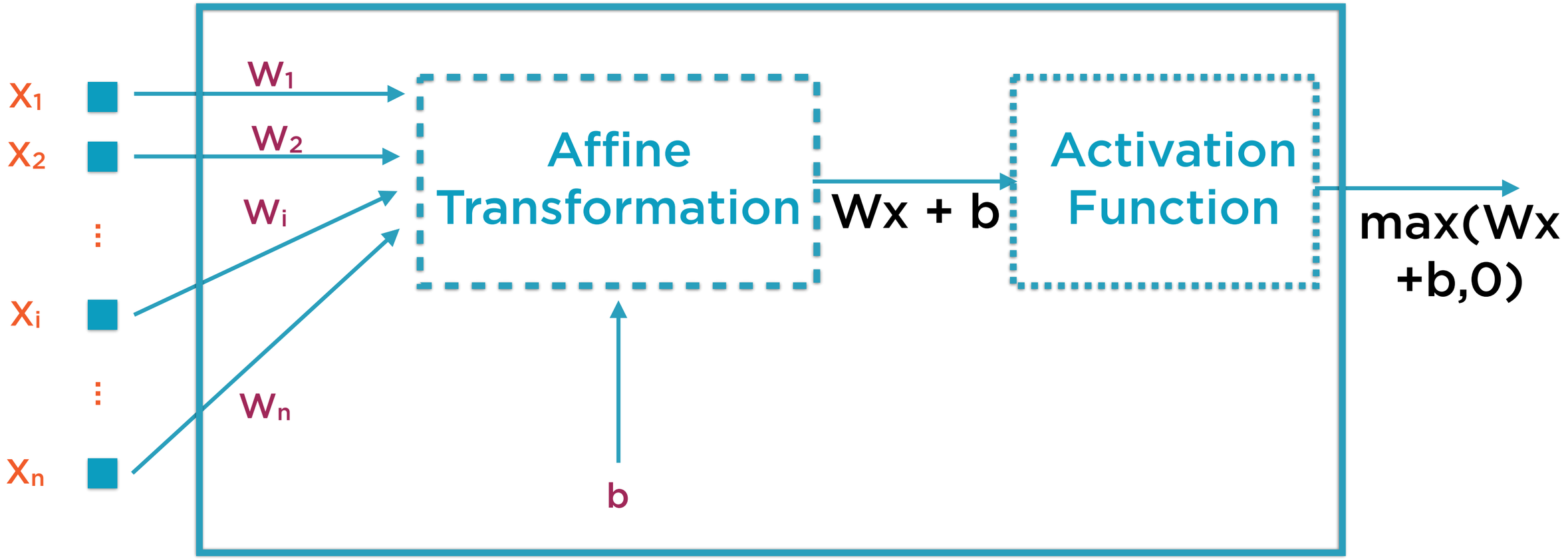
# Neural Networks

**Neurons in a neural network can be connected in very complex ways...**

**...But each neuron only applies two simple functions to its inputs**

- A linear (affine) transformation
- An activation function

# Operation of a Single Neuron




**Each neuron only applies two simple functions to its inputs**

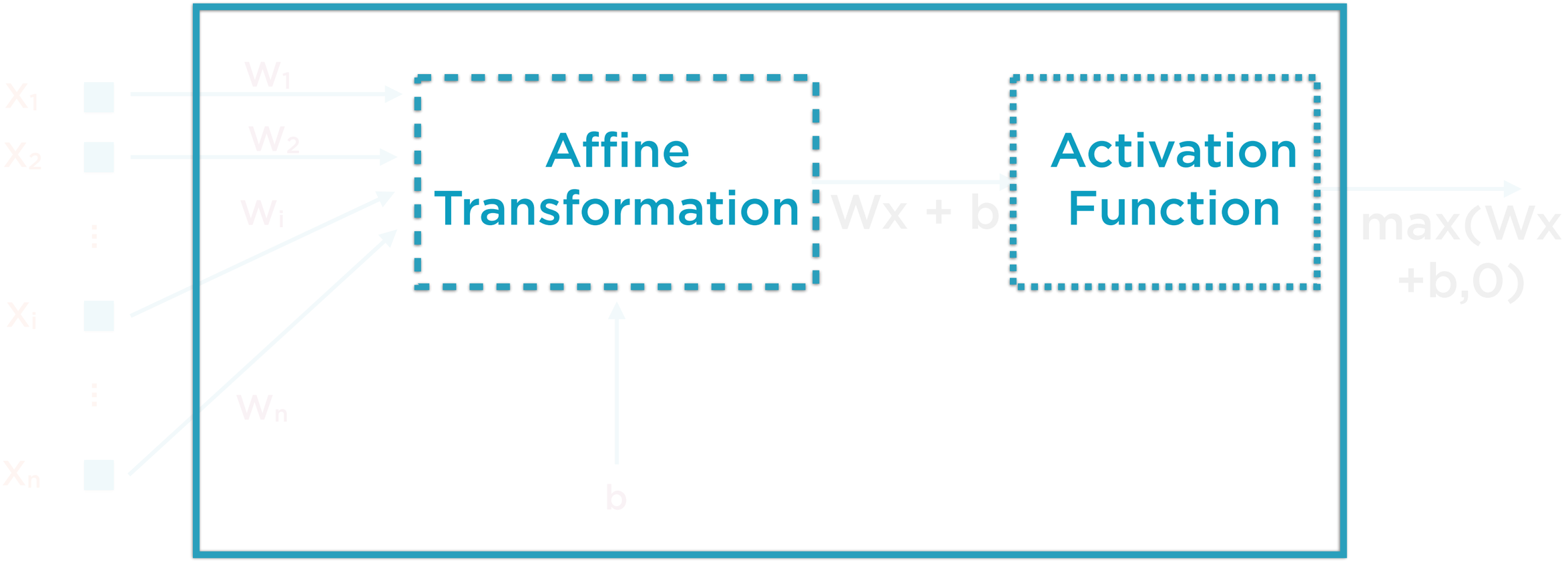
# Operation of a Single Neuron


**Inputs into the neuron**

# Operation of a Single Neuron

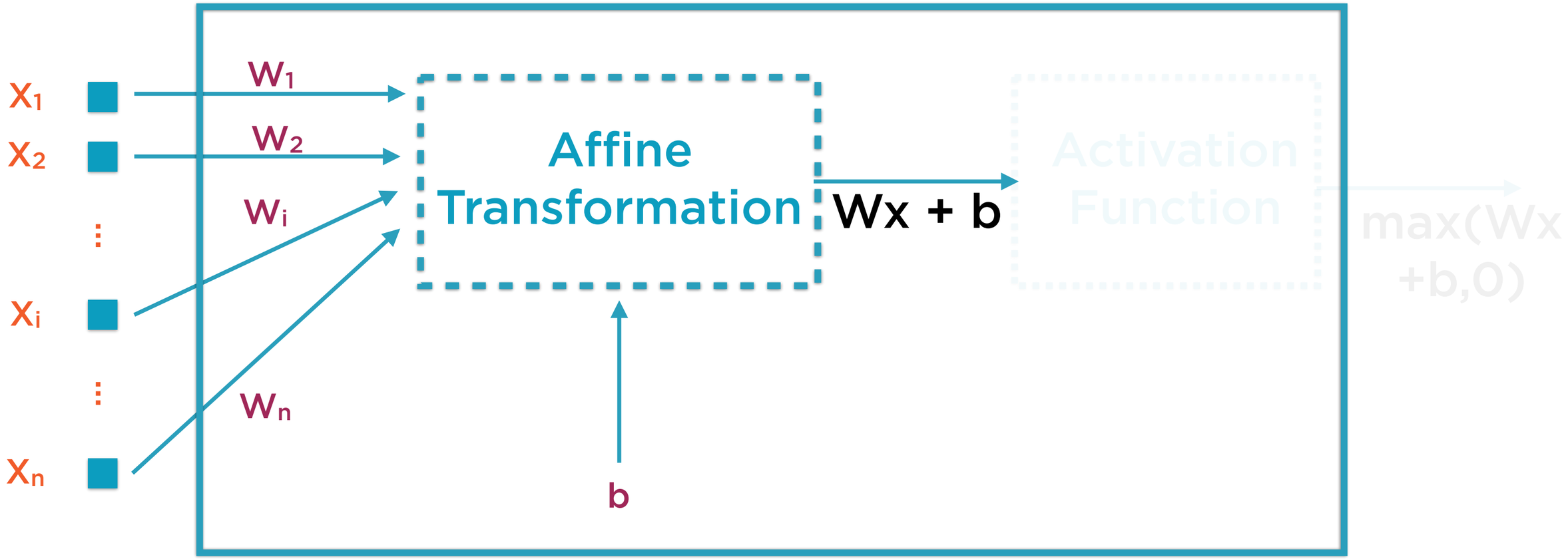

**Output from the neuron**

# Operation of a Single Neuron



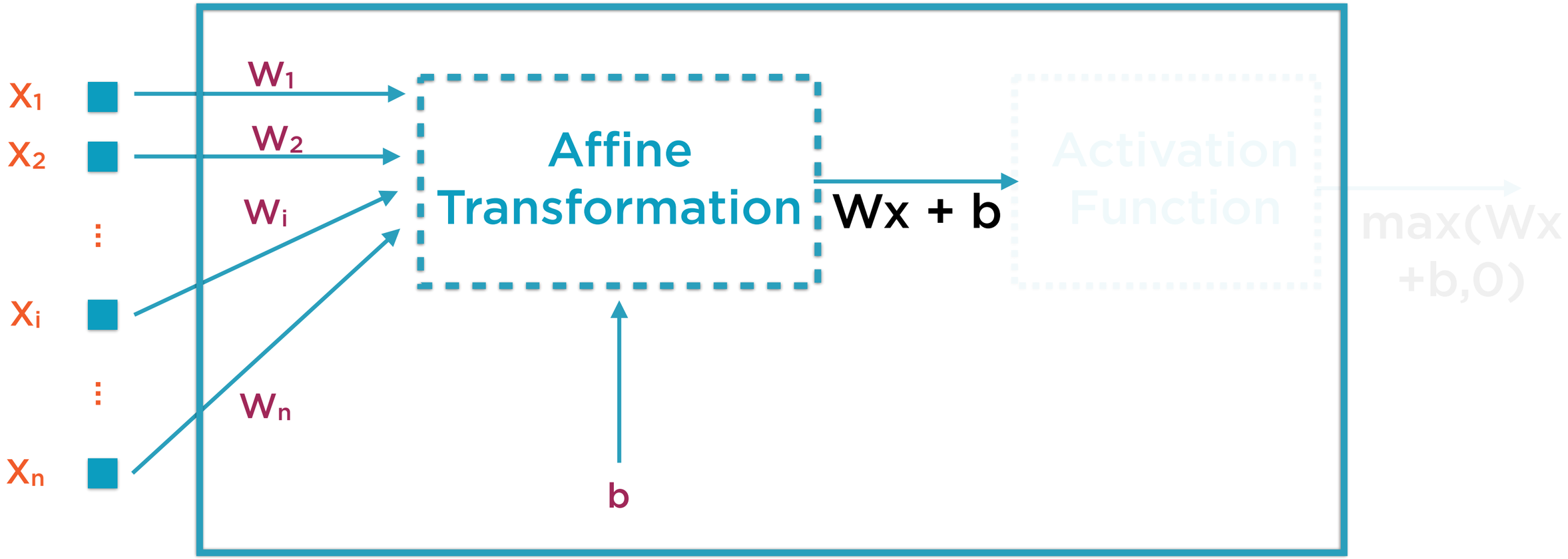

**Each neuron only applies two simple functions to its inputs**

# Operation of a Single Neuron



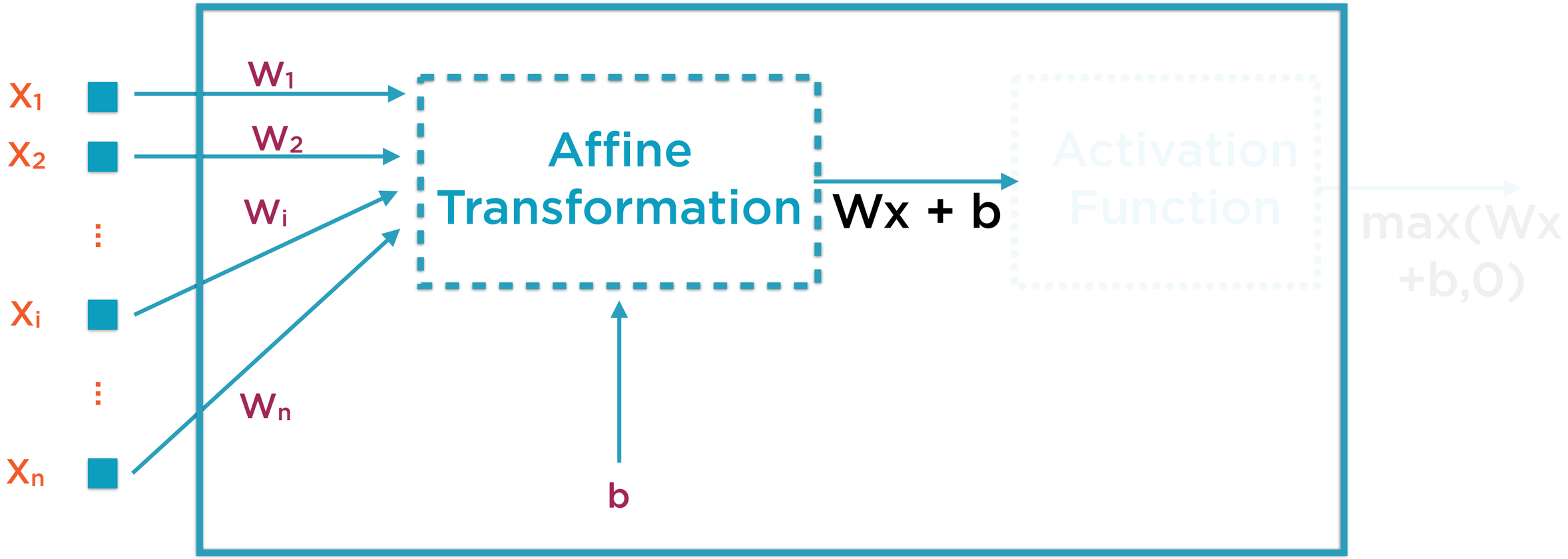

**The affine transformation is just a weighted sum with a bias added**

# Operation of a Single Neuron



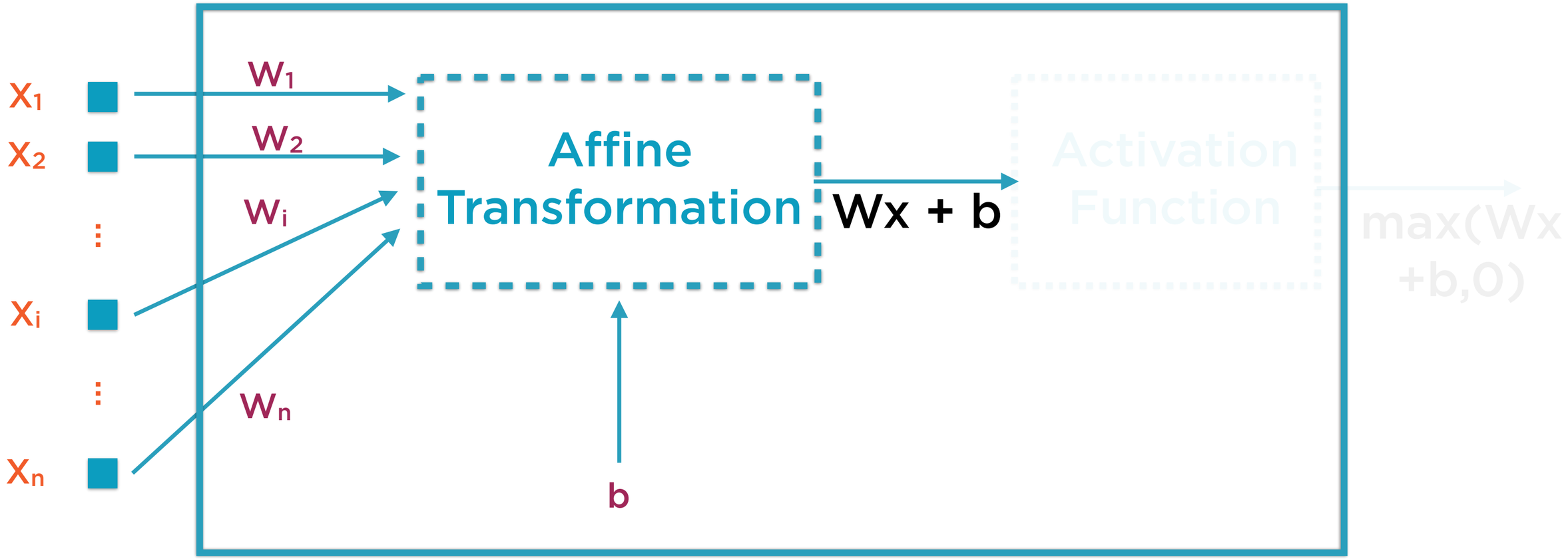

**The values $W_1$, $W_2$...$W_n$ are called the weights**

# Operation of a Single Neuron




**The value b is called the bias**

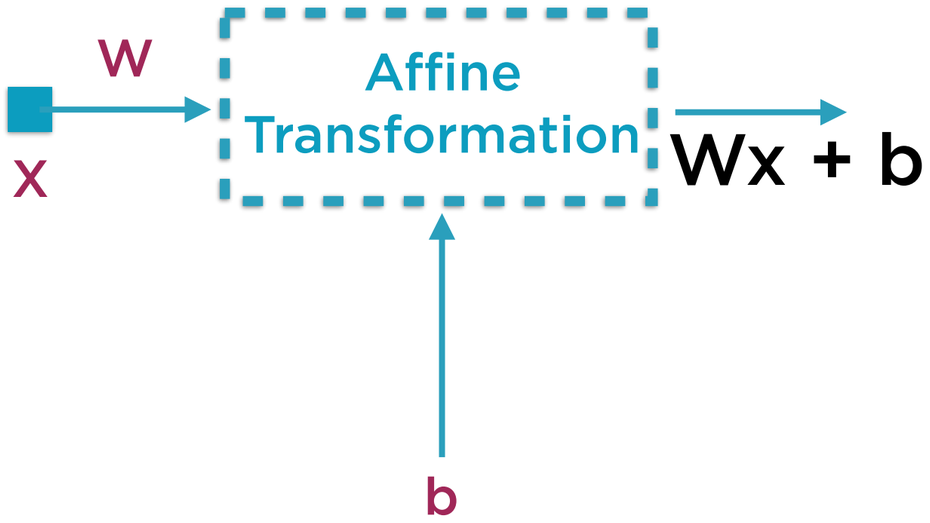
# Operation of a Single Neuron




**Where do the values of W and b come from?**

The weights and biases of individual neurons are determined during the training process

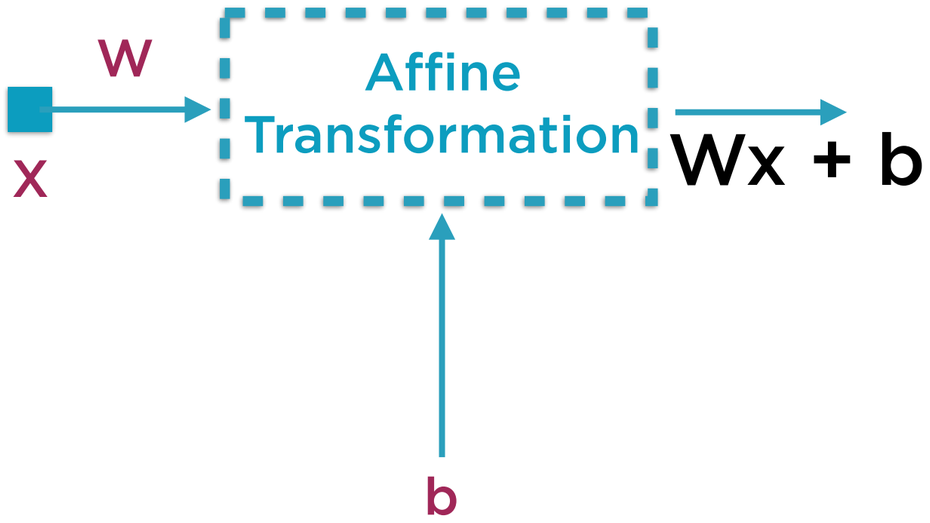
The actual training of a neural network is managed by TensorFlow




Finding the “best” values of W and b for each neuron is crucial

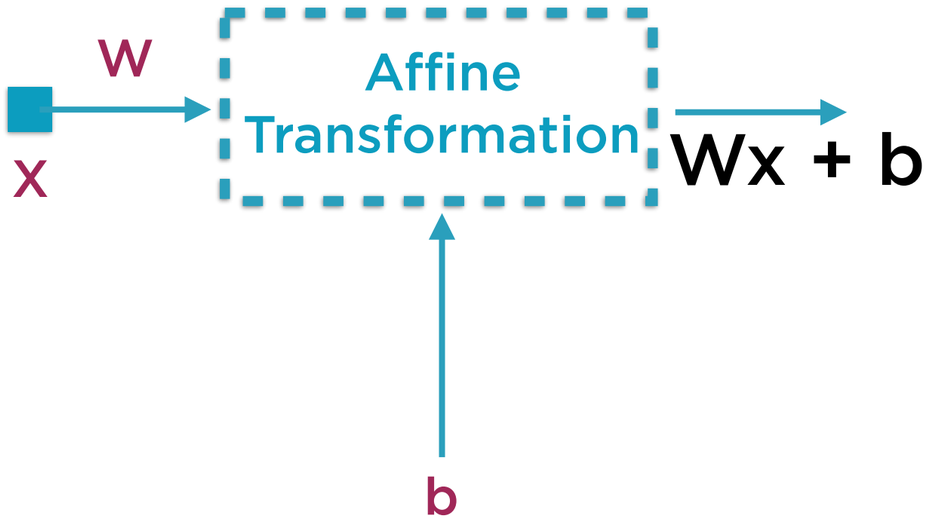
The “best” values are found using the cost function, optimiser and corpus...

...and the process of finding them is called the training process




Different types of neural networks wire up neurons in different ways

These interconnections can get very sophisticated...

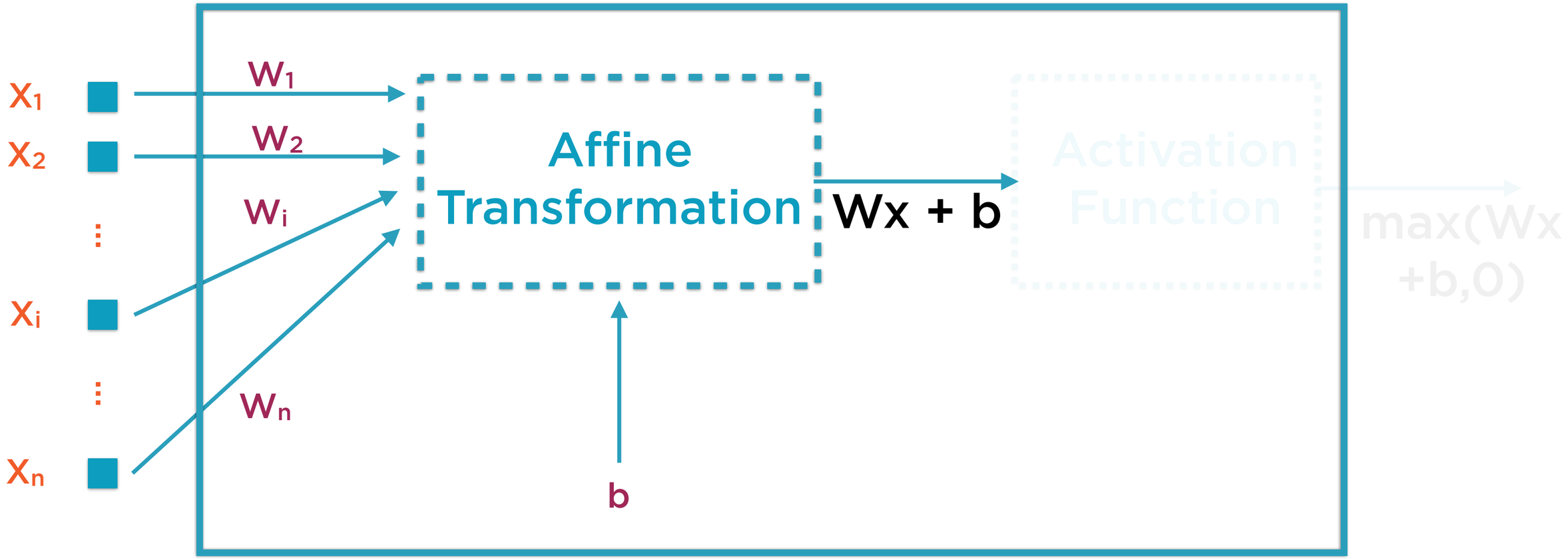



During training, the output of deeper layers may be “fed back” to find the best W, b

This is called back propagation

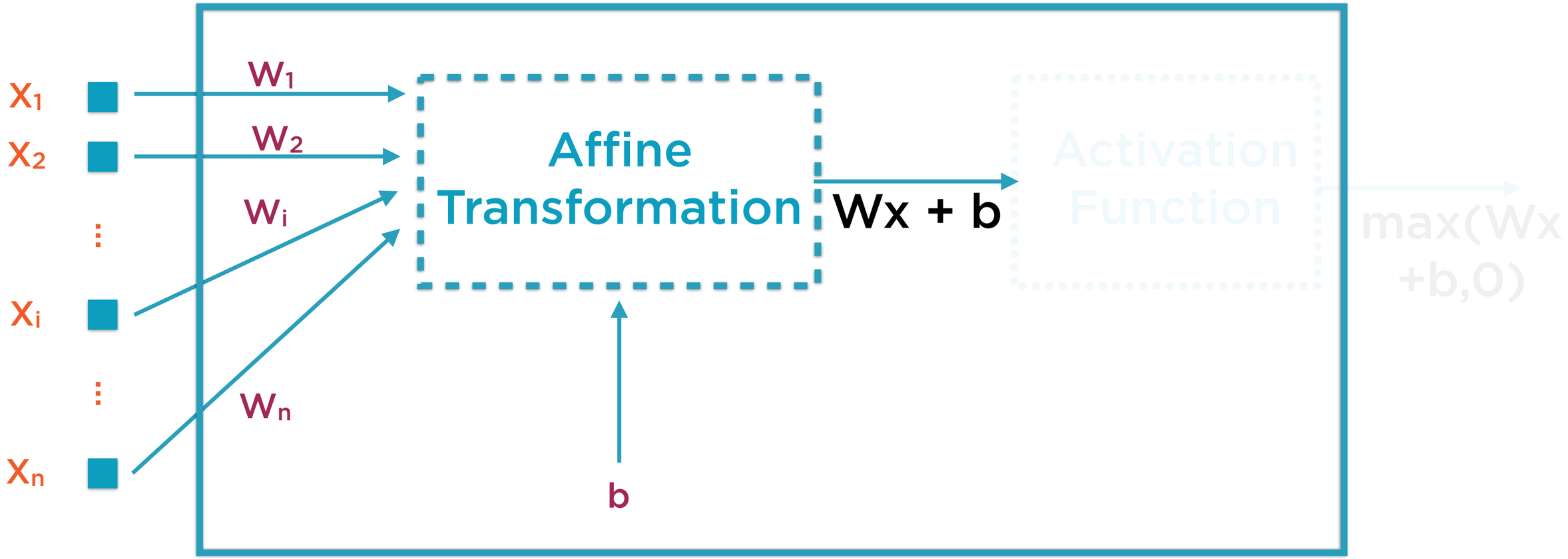
Back propagation is the standard algorithm for training neural networks

# Operation of a Single Neuron



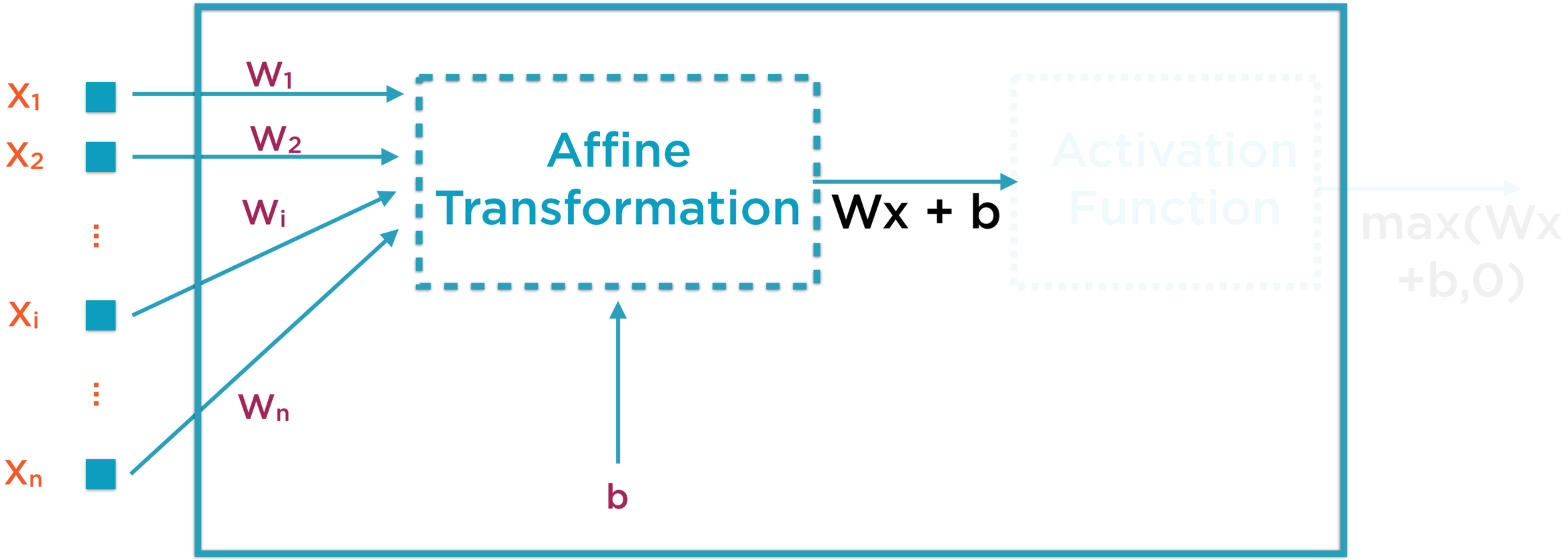

**The training algorithm will use the weights to tell the neuron which inputs matter, and which do not...**

# Operation of a Single Neuron



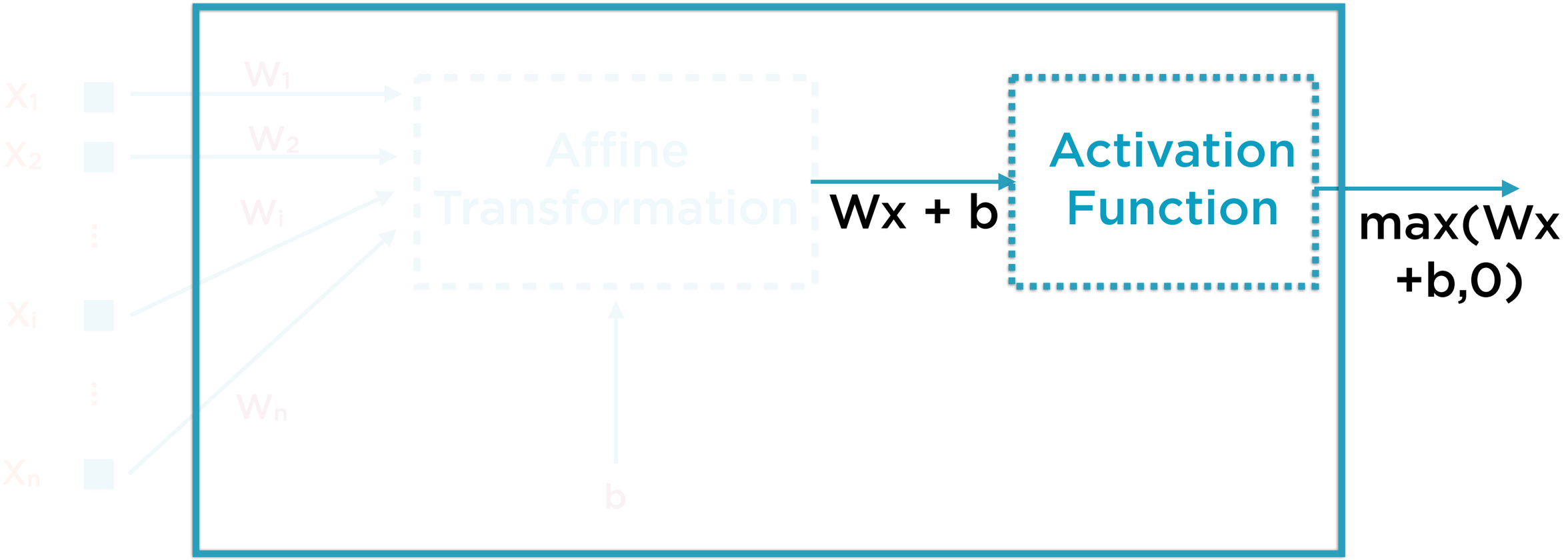

**...and apply a corrective bias if needed**

# Operation of a Single Neuron




**The linear output can only be used to learn linear functions, but we can easily generalize this...**

# Operation of a Single Neuron




**The Activation function is a non-linear function, very often simply the max(0,...) function**


The output of the affine transformation is chained into an activation function


The activation function is needed for the neural network to predict non-linear functions


The most common form of the activation function is the ReLU

ReLU : Rectified Linear Unit

ReLU(x) = max(0,x)

# Regression: The Simplest Neural Network

```
def doSomethingReallyComplicated(x1,x2…):
   …
   …
   …
   return complicatedResult
```

# “Learning” Arbitrarily Complex Functions

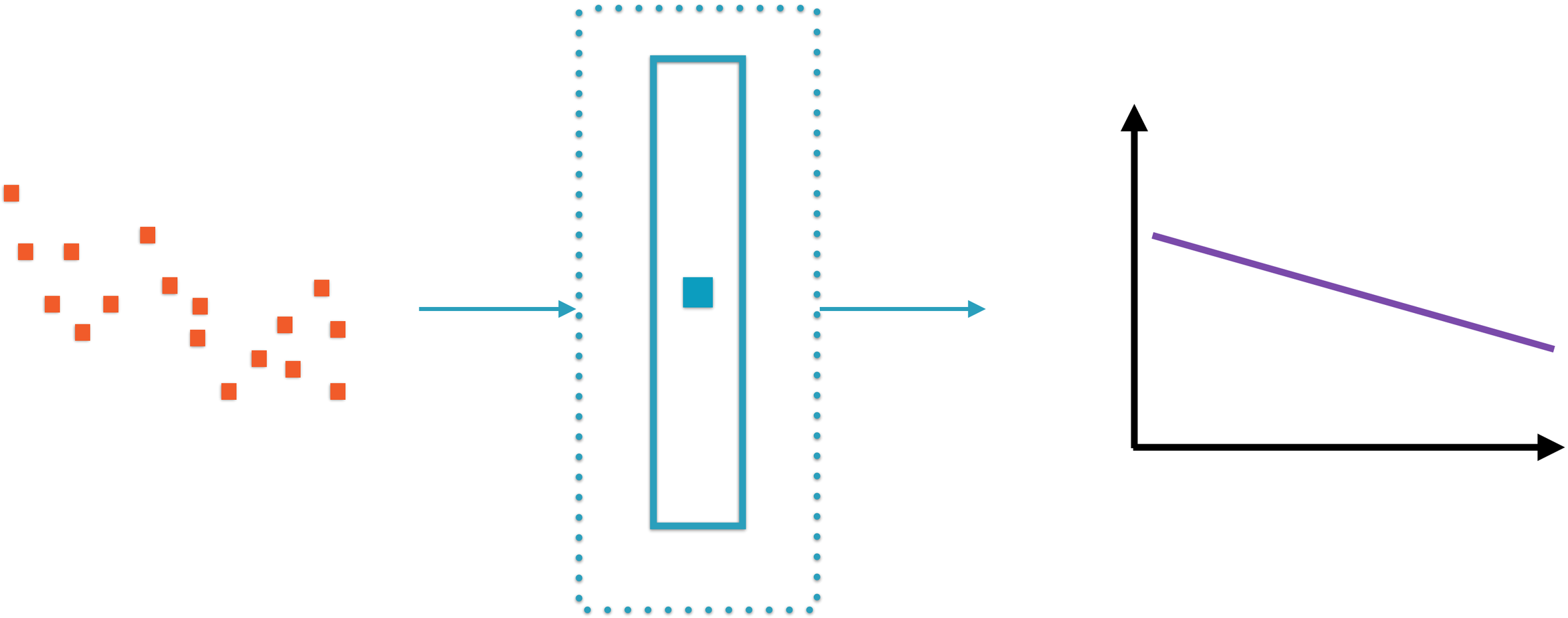
**Adding layers to a neural network can “learn” (reverse-engineer) pretty much anything**

```
y = Wx + b
```

# “Learning” Regression

**Regression can be reverse-engineered by a single neuron**

# Regression: The Simplest Neural Network

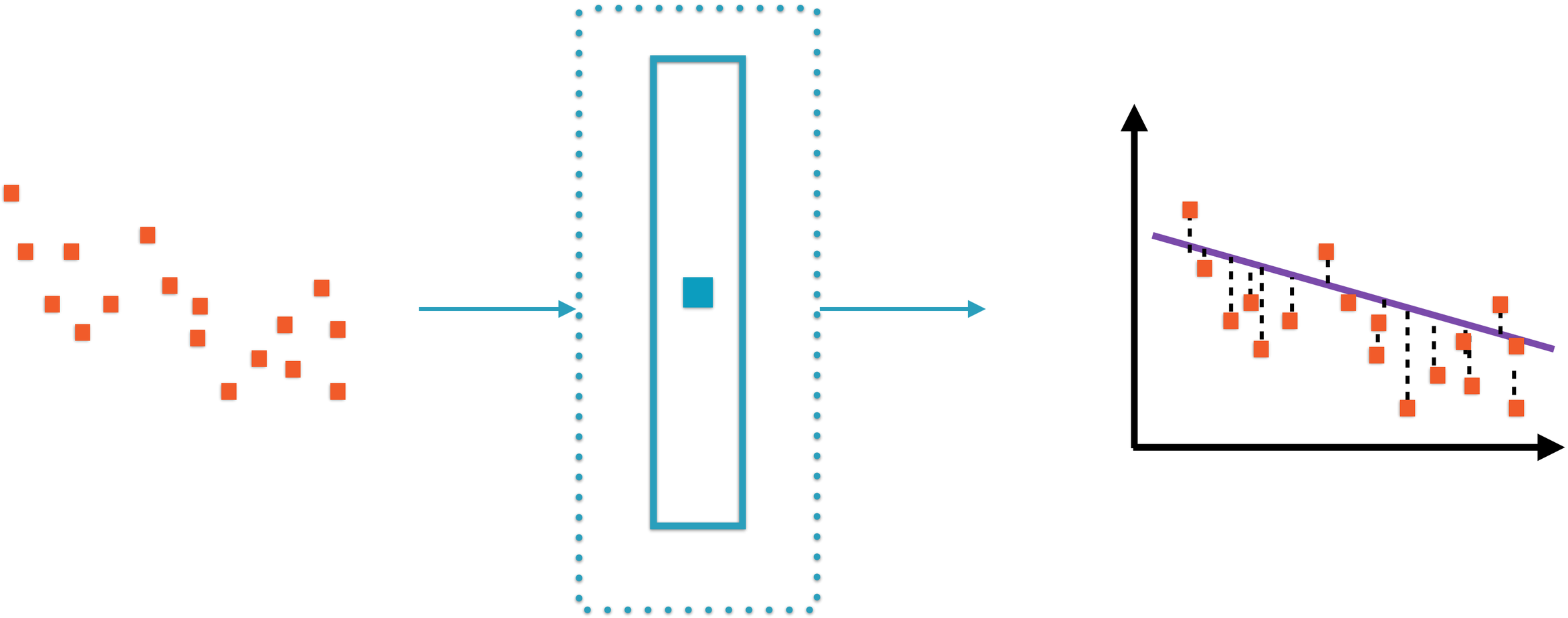


Set of Points

Single Neuron

Regression Line

# Regression: The Simplest Neural Network

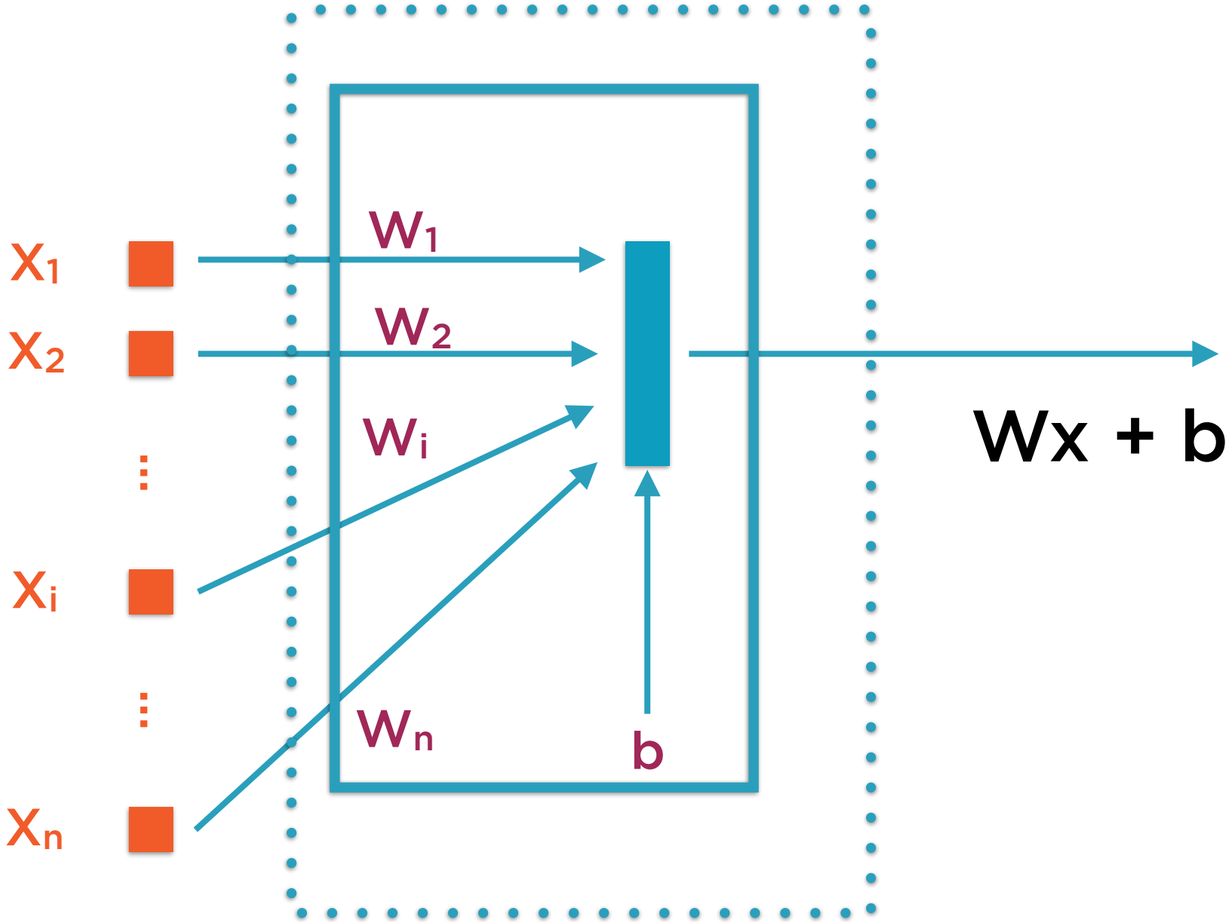


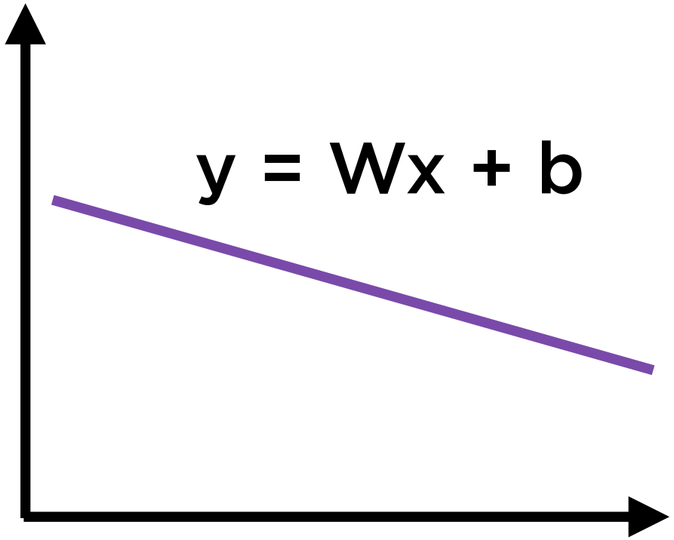
Set of Points

Single Neuron

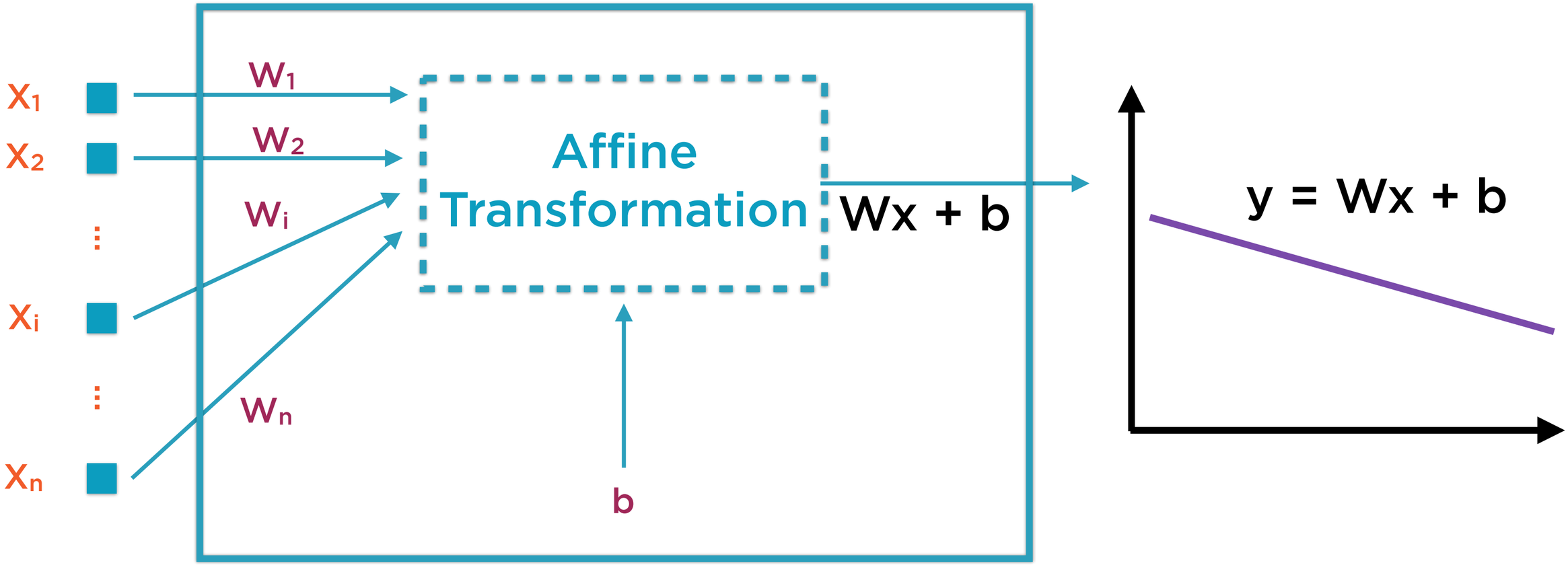
Regression Line

# Operation of a Single Neuron



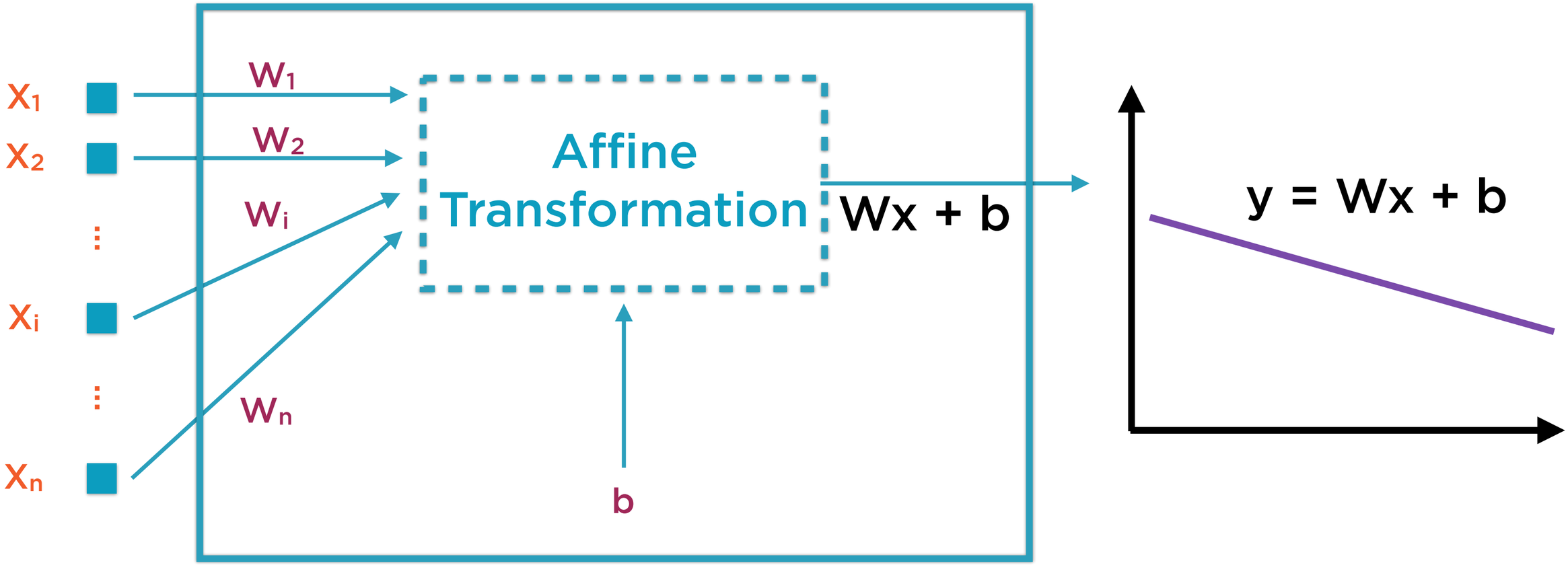



# Operation of a Single Neuron



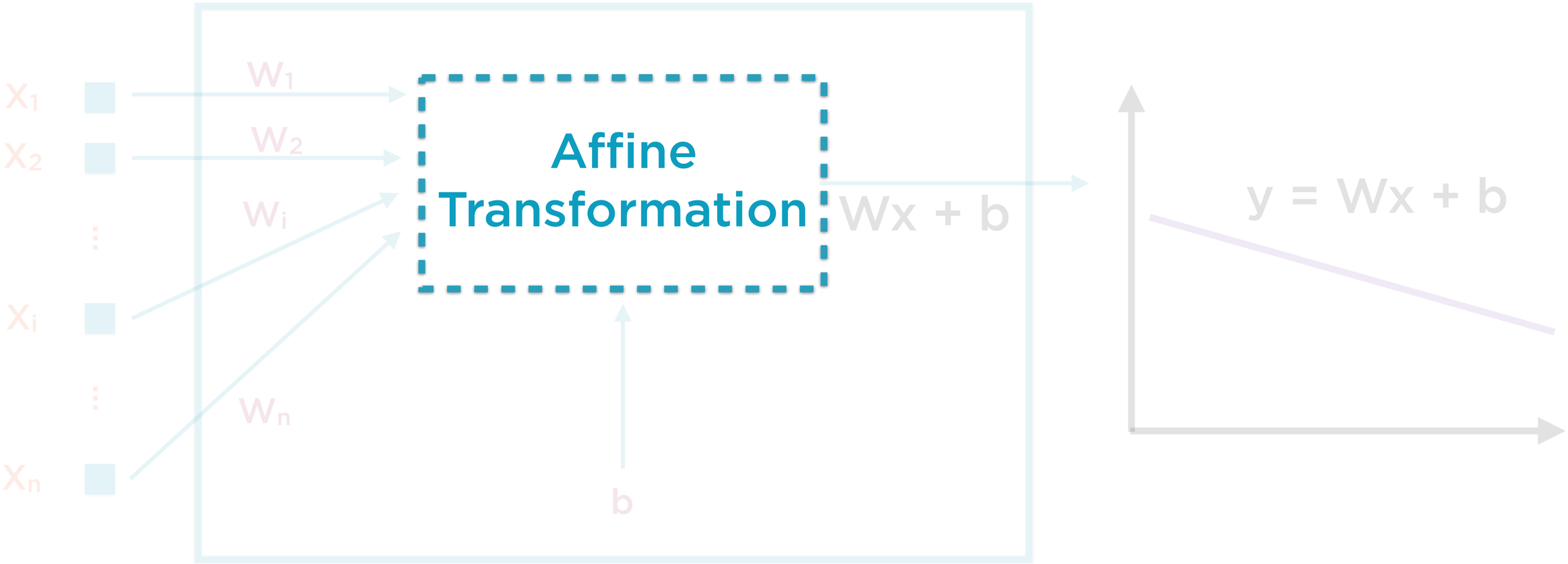

**Here the neuron is an entity that finds the "best fit" line through a set of points**

# Operation of a Single Neuron



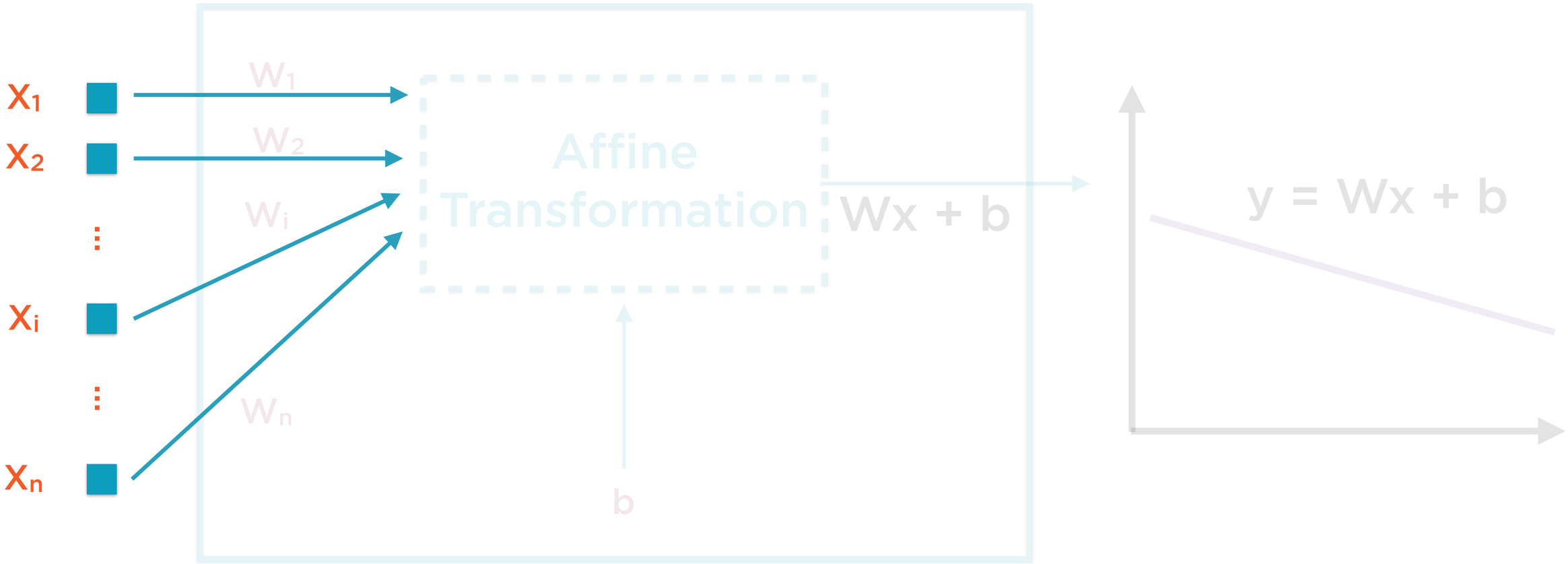

**We are instructing the neuron to learn a linear function**
**- so no activation function is required at all**

# Operation of a Single Neuron



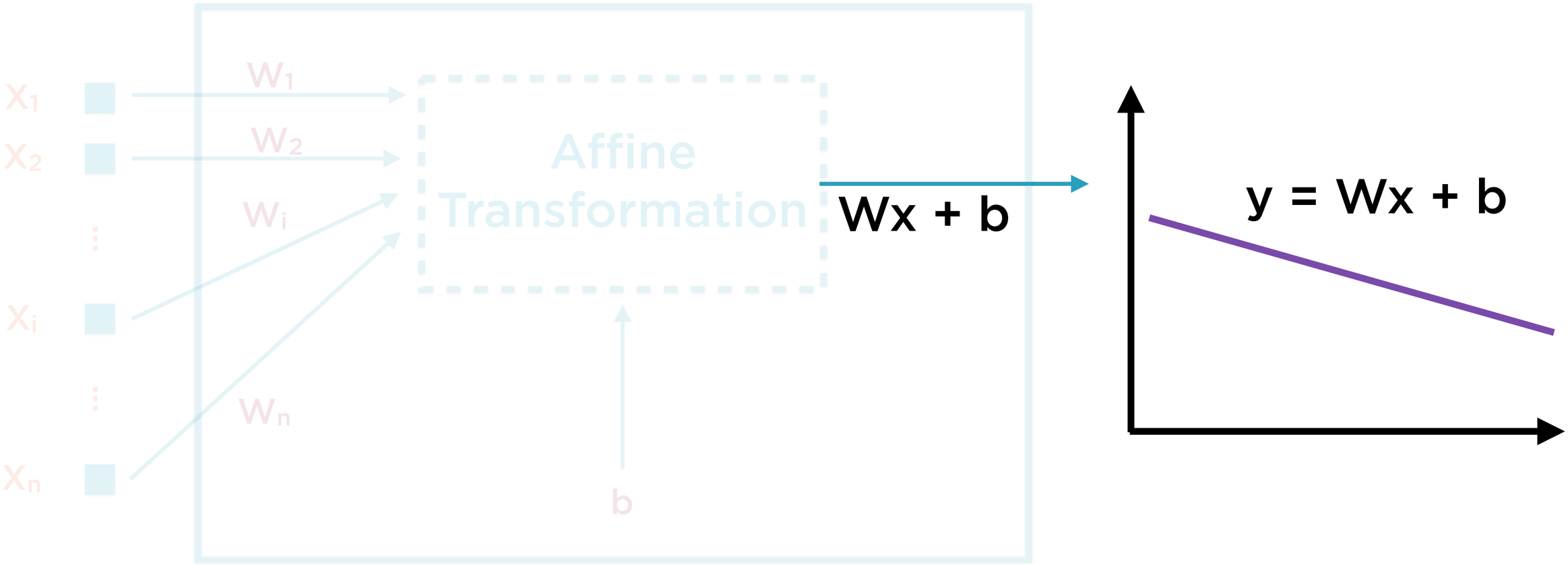

**The affine transformation is just a weighted sum of the inputs with a bias added**

# Operation of a Single Neuron



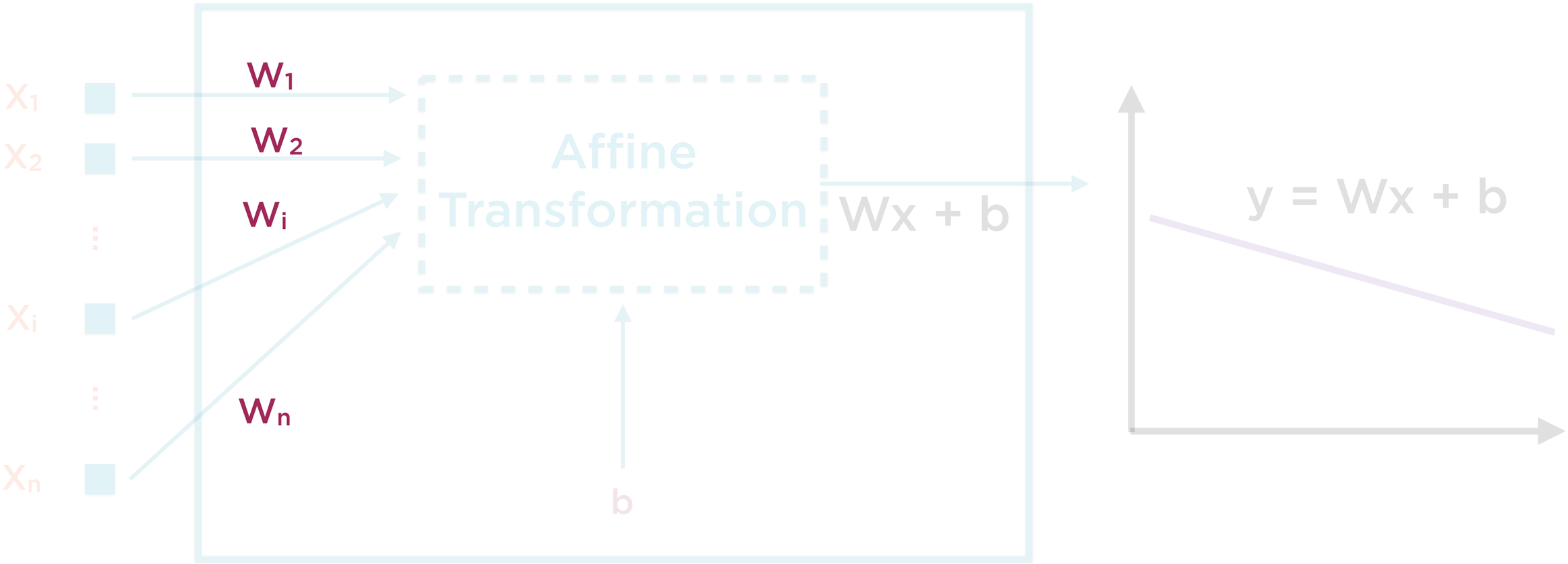

**Inputs into the neuron**

# Operation of a Single Neuron



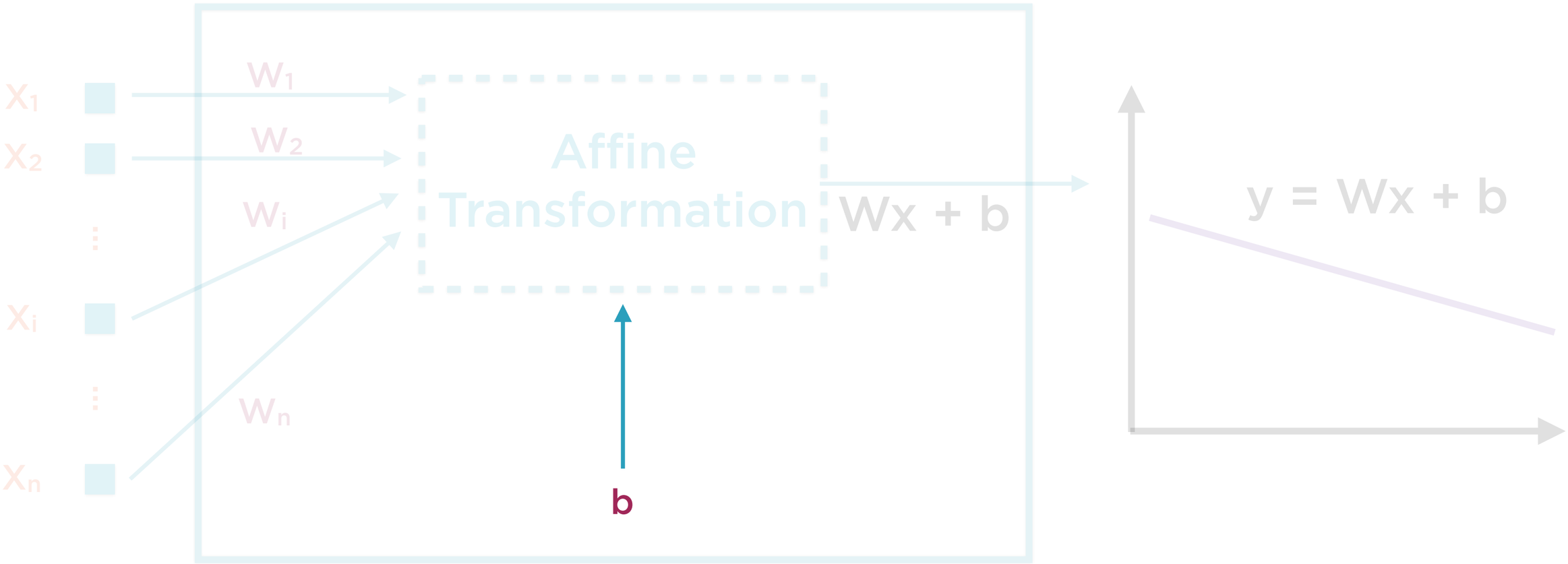

Output from the neuron

# Operation of a Single Neuron



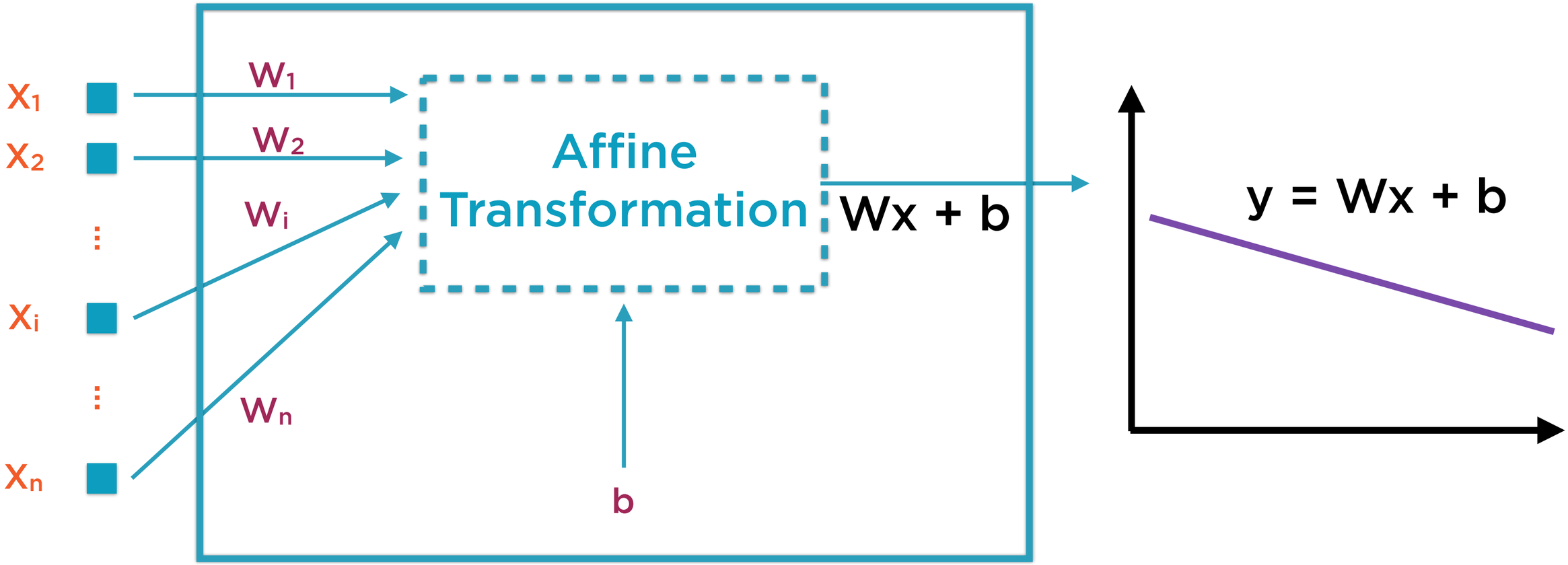

**The values $W_1$, $W_2$...$W_n$ are called the weights**

# Operation of a Single Neuron




**The value b is called the bias**

# Operation of a Single Neuron




**Where do the weights W and the bias b come from?**

The weights and biases of individual neurons are determined during the training process

The actual training of a neural network is managed by TensorFlow

# Simple Regression

**Regression Equation:**

$$y = A + Bx$$

$$
\begin{aligned}
y_1 &= A + Bx_1 \\
y_2 &= A + Bx_2 \\
y_3 &= A + Bx_3 \\
&\dots \quad\quad \dots \\
y_n &= A + Bx_n
\end{aligned}
$$

# Simple Regression

**Regression Equation:**

$$y = A + Bx$$

$$
\begin{aligned}
y_1 &= A + Bx_1 + e_1 \\
y_2 &= A + Bx_2 + e_2 \\
y_3 &= A + Bx_3 + e_3 \\
\dots &\quad \dots \\
y_n &= A + Bx_n + e_n
\end{aligned}
$$

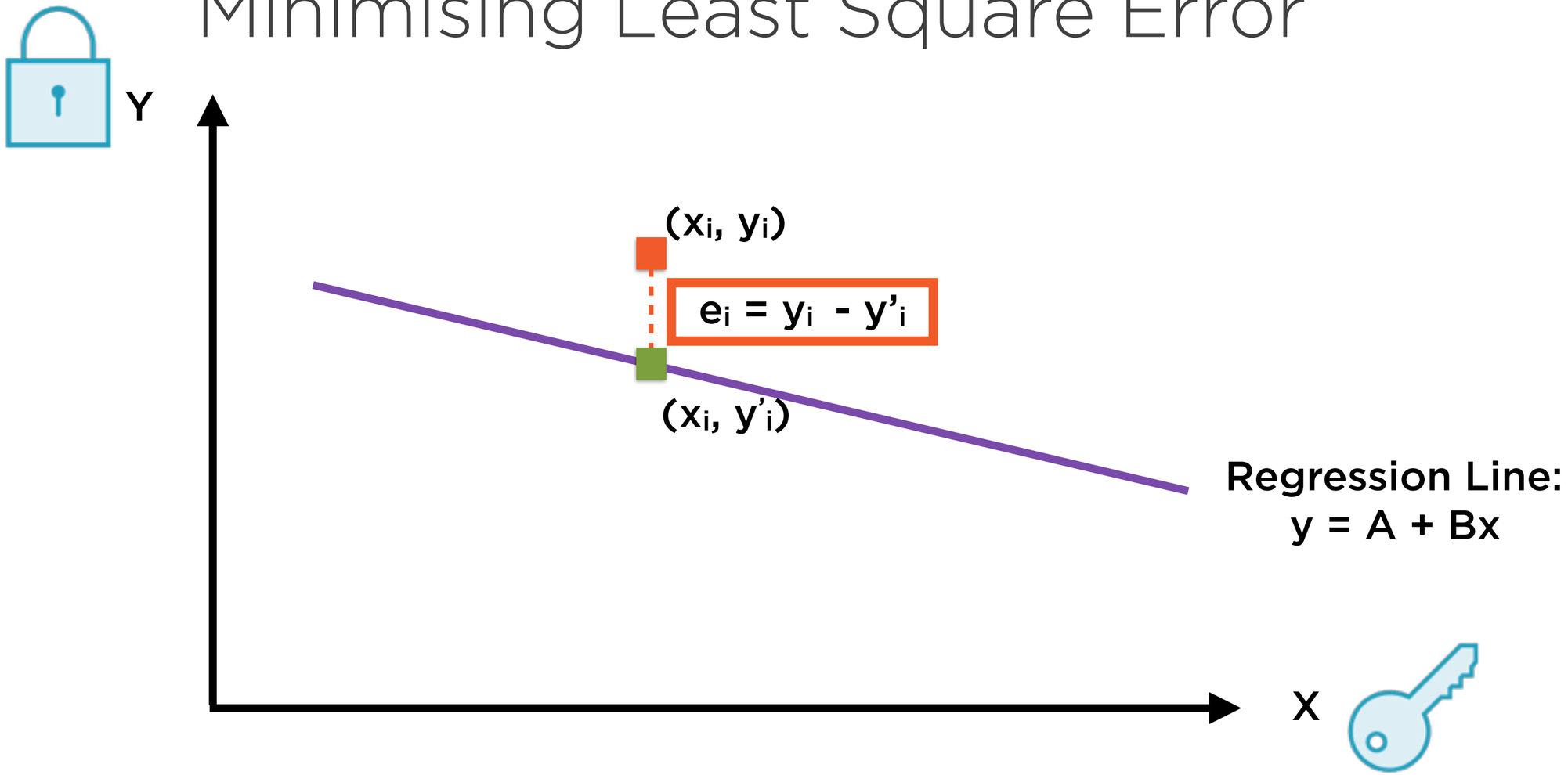
# Simple Regression

**Regression Equation:**

$$y = A + Bx$$

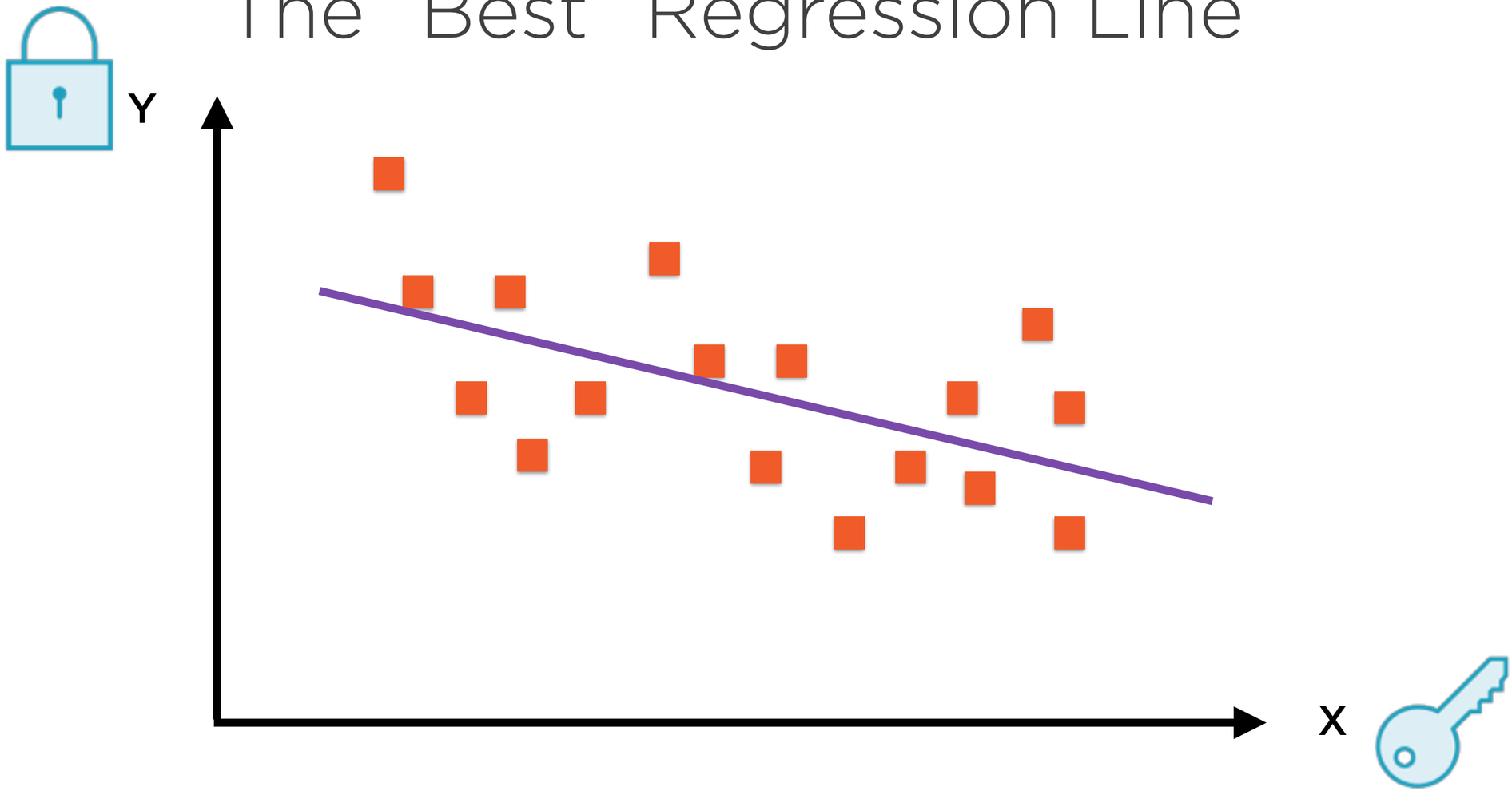
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_n \end{bmatrix} = A \begin{bmatrix} 1 \\ 1 \\ 1 \\ \dots \\ 1 \end{bmatrix} + B \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \dots \\ e_n \end{bmatrix}$$

# Minimising Least Square Error



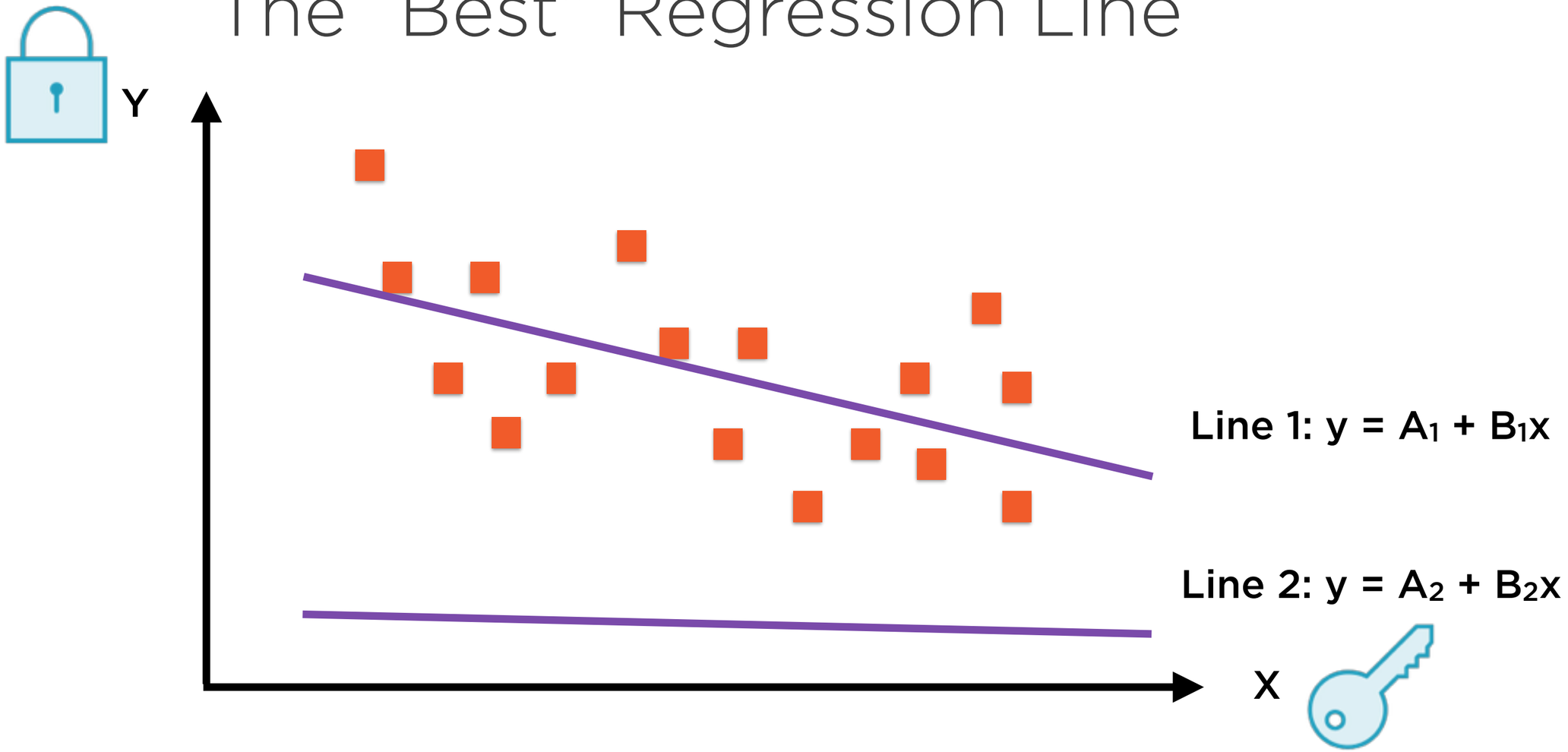

Residuals of a regression are the difference between actual and fitted values of the dependent variable

# The “Best” Regression Line



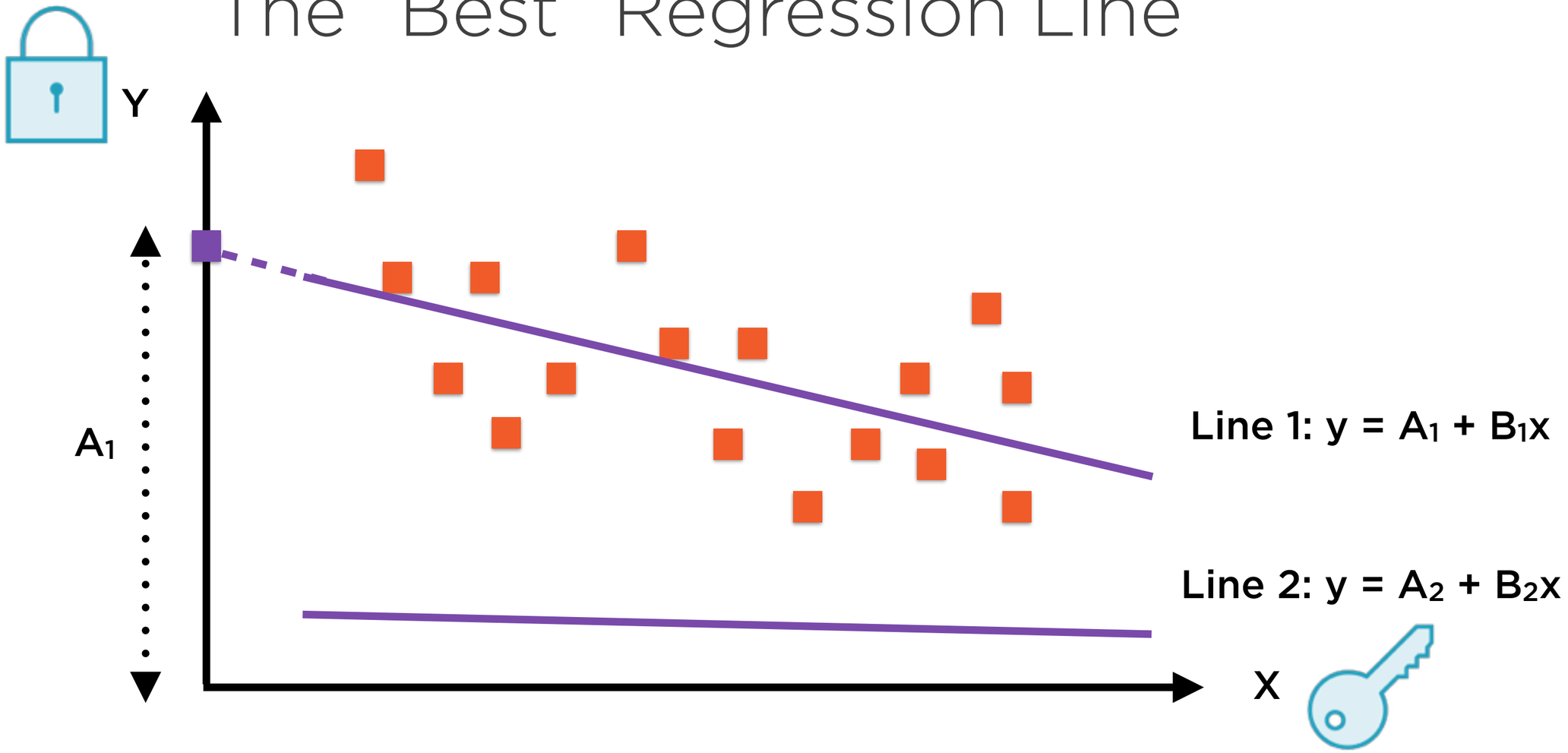

Linear Regression involves finding the “best fit” line

# The “Best” Regression Line



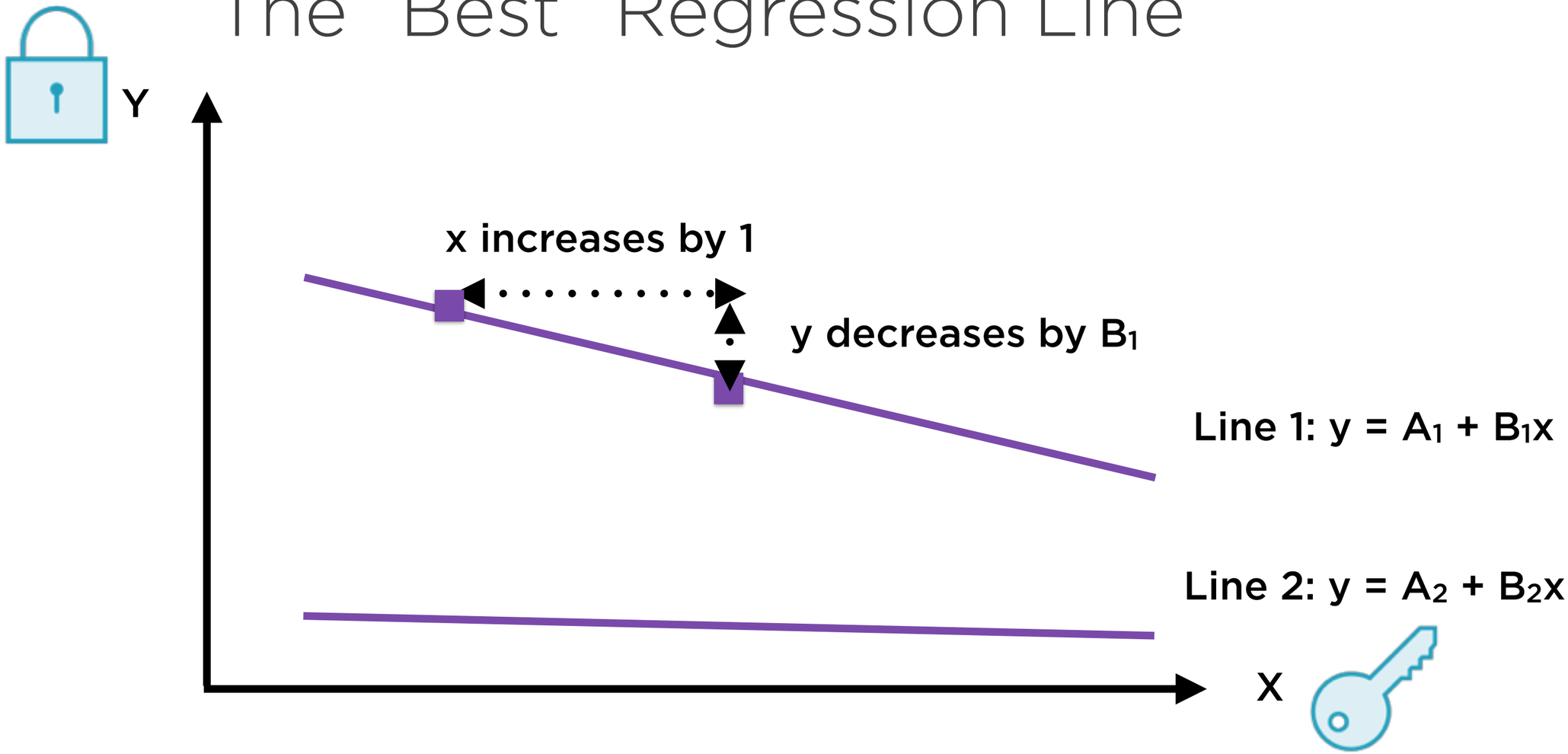

Let’s compare two lines, Line 1 and Line 2

# The “Best” Regression Line



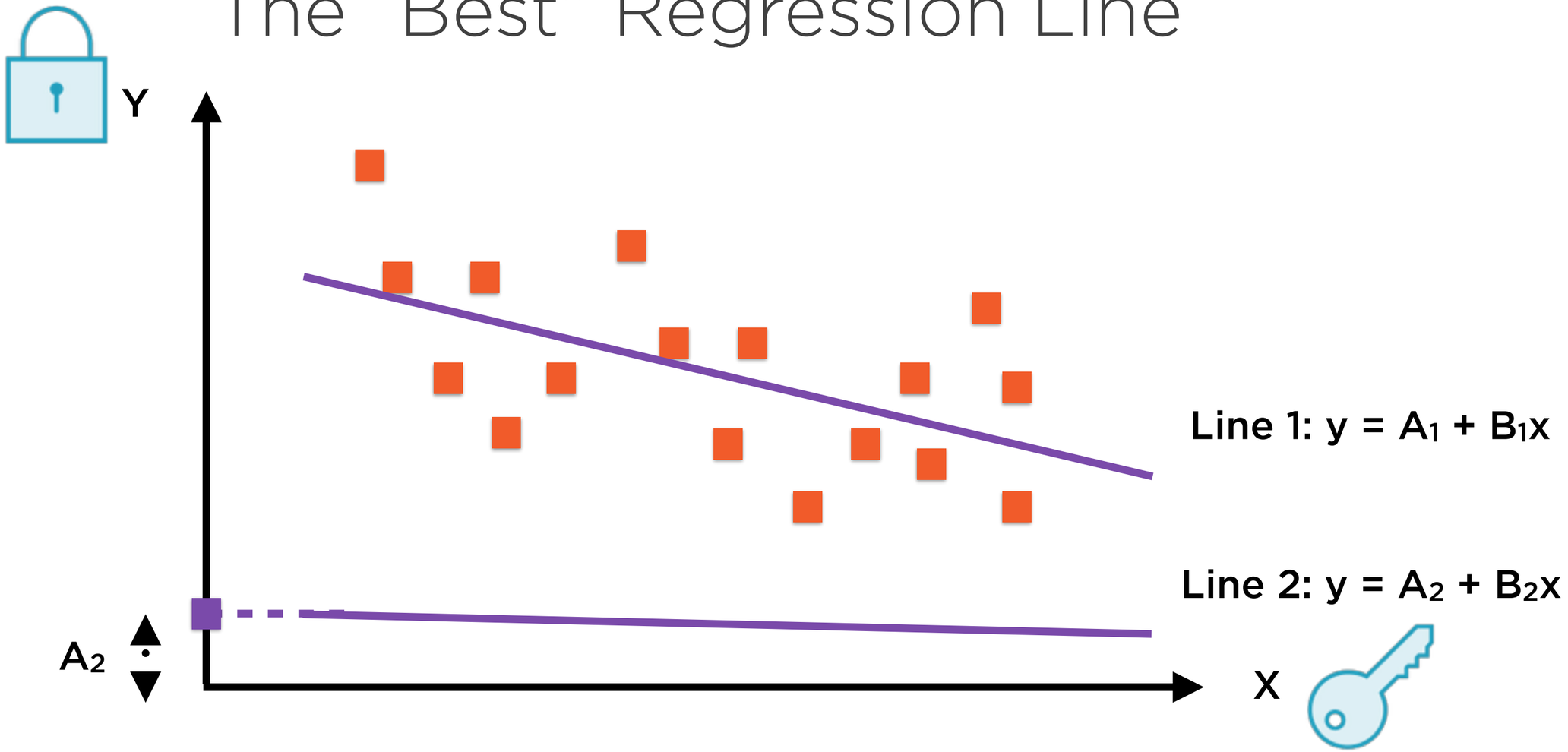

The first line has y-intercept $A_1$

# The "Best" Regression Line



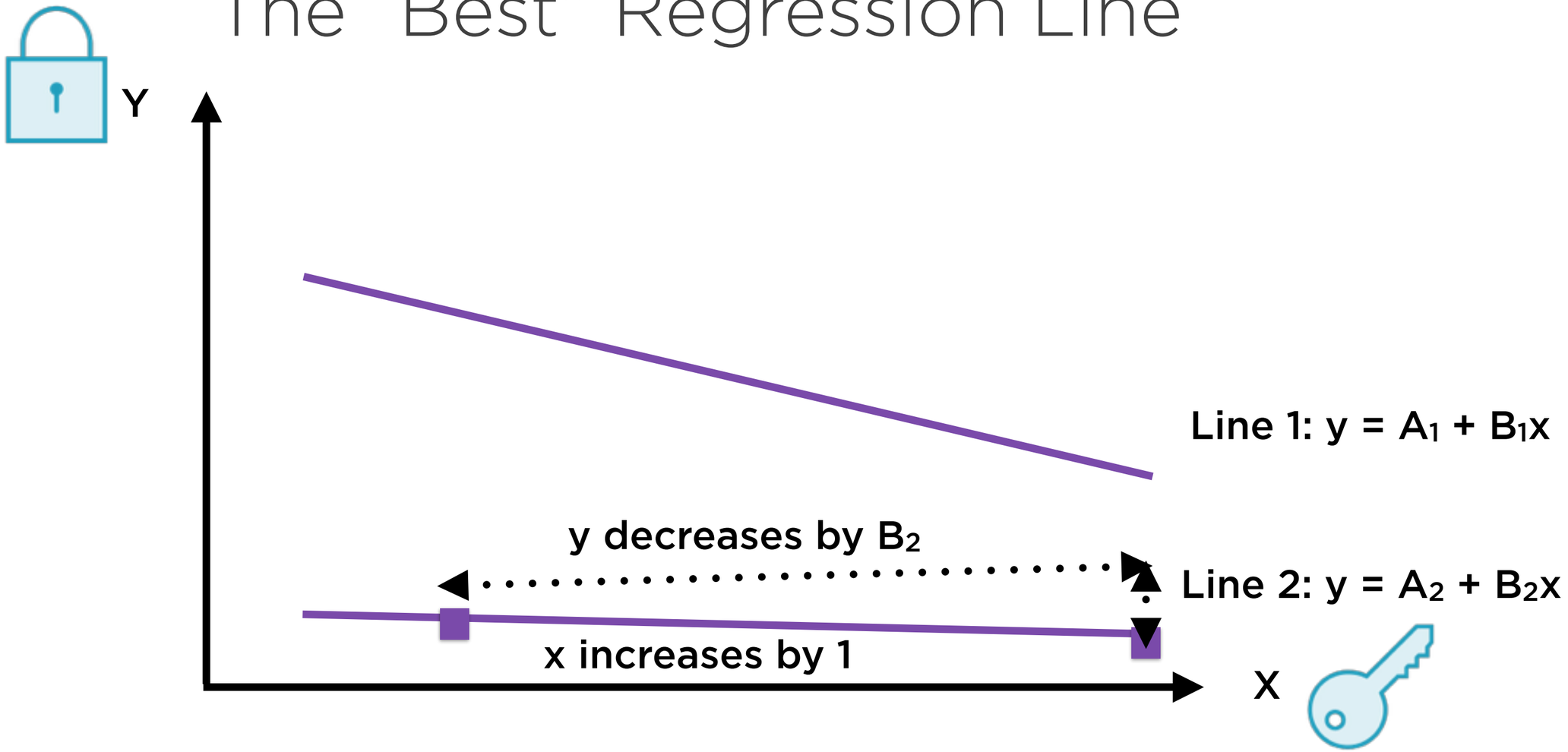

In the first line, if x increases by 1 unit, y decreases by $B_1$ units

# The “Best” Regression Line



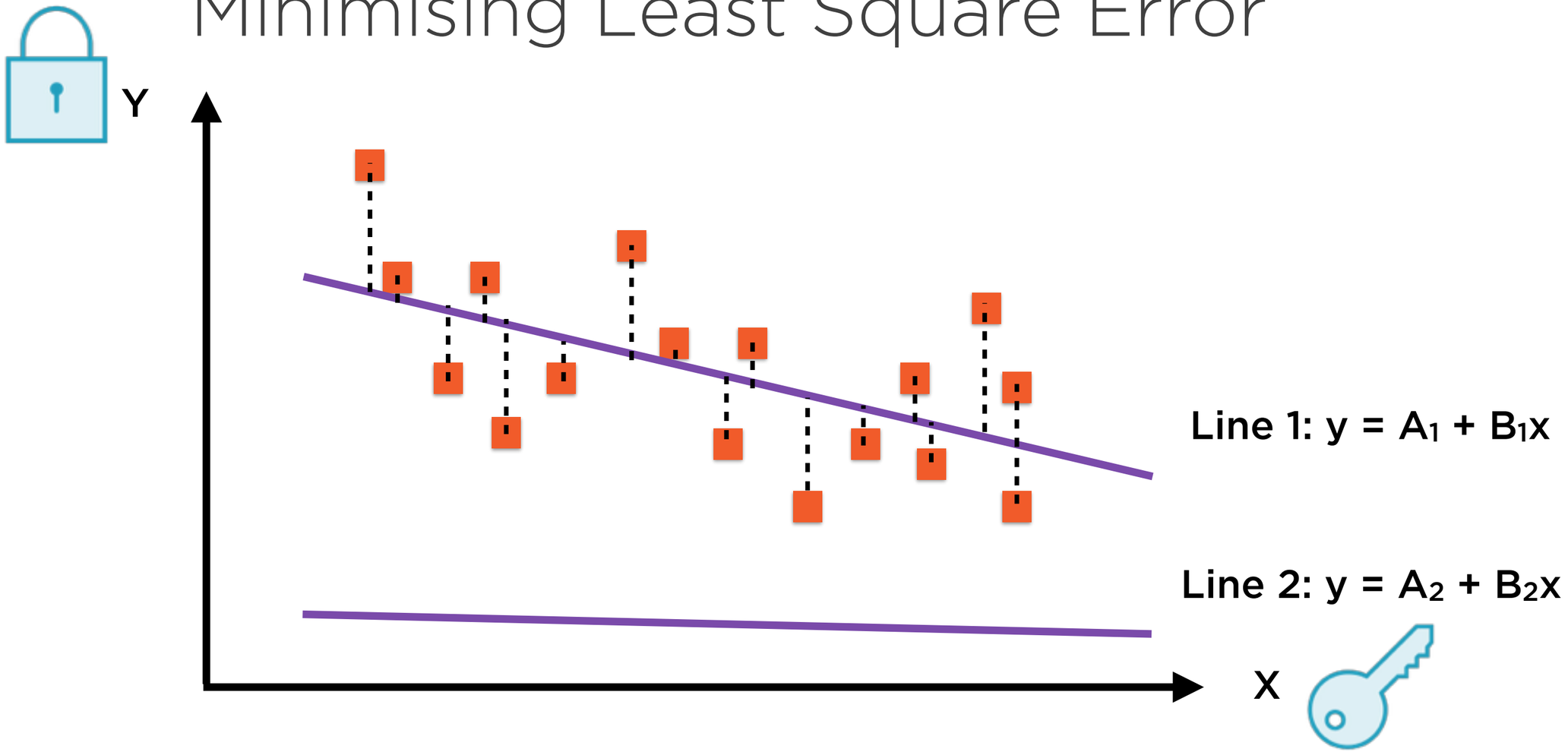

The second line has y-intercept $A_2$

# The "Best" Regression Line



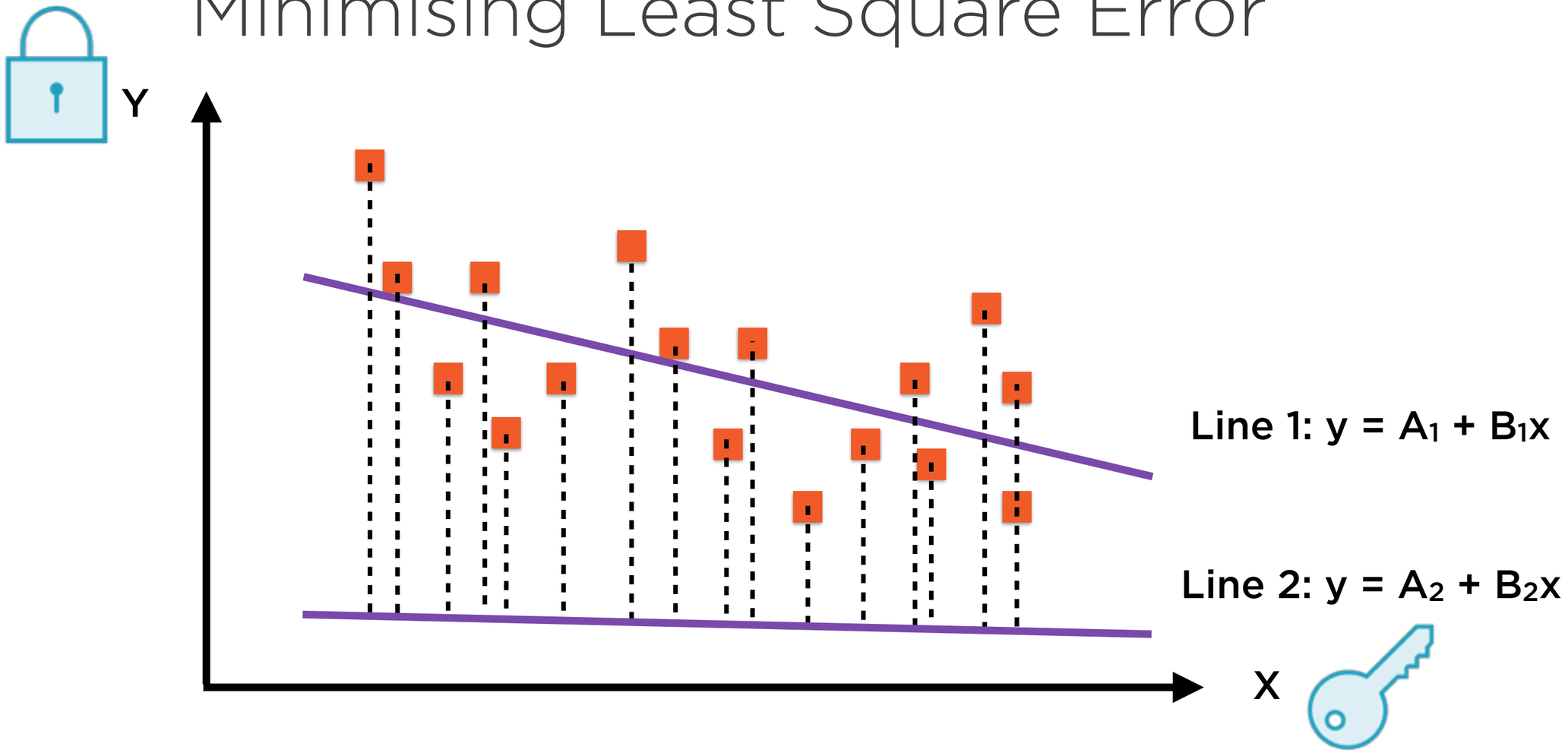

In the second line, if x increases by 1 unit, y decreases by $B_2$ units

# Minimising Least Square Error




Drop vertical lines from each point to the lines A and B

# Minimising Least Square Error




Drop vertical lines from each point to the lines A and B

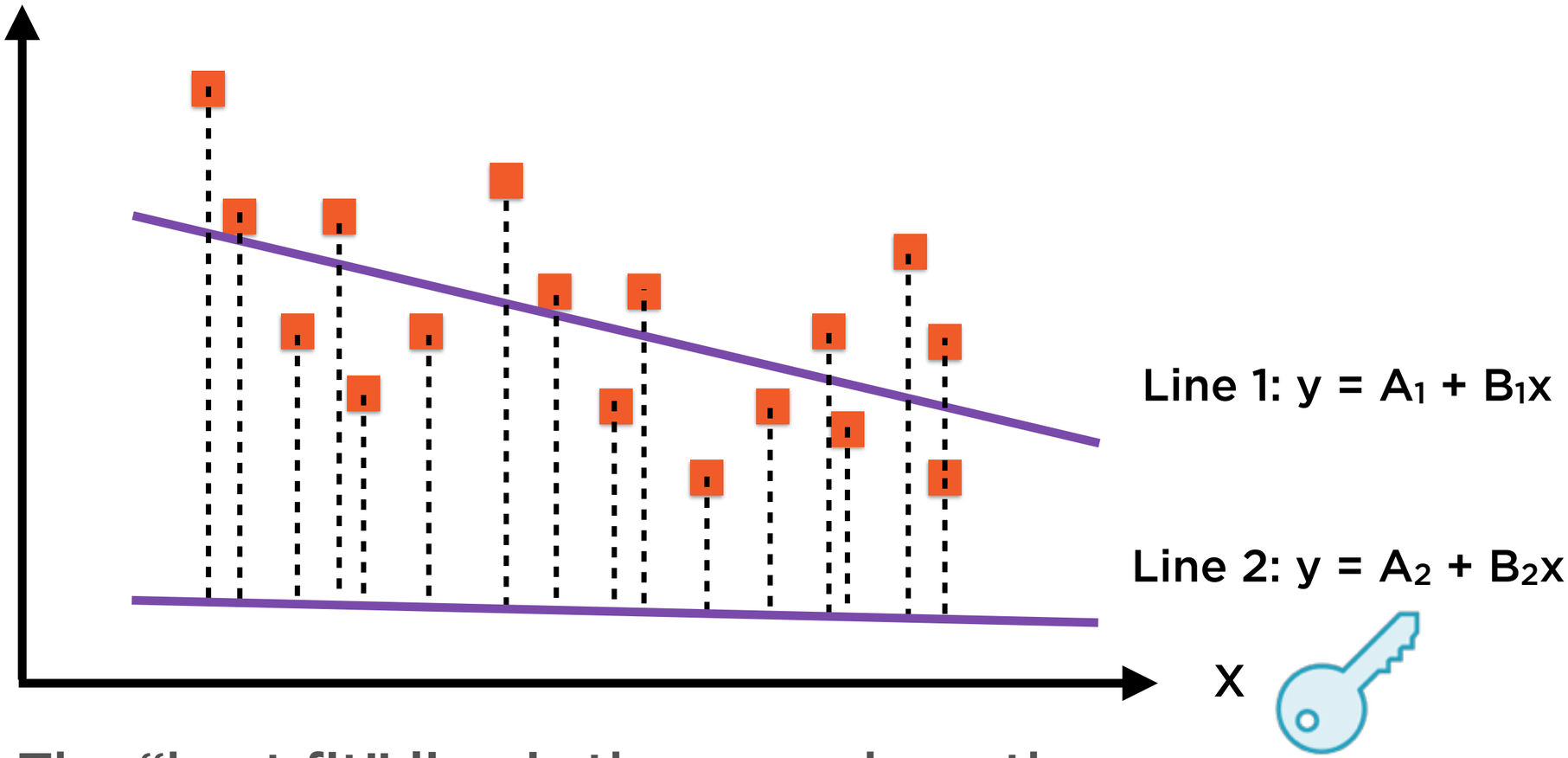
# Simple Regression

**Regression Equation:**

$$y = A + Bx$$

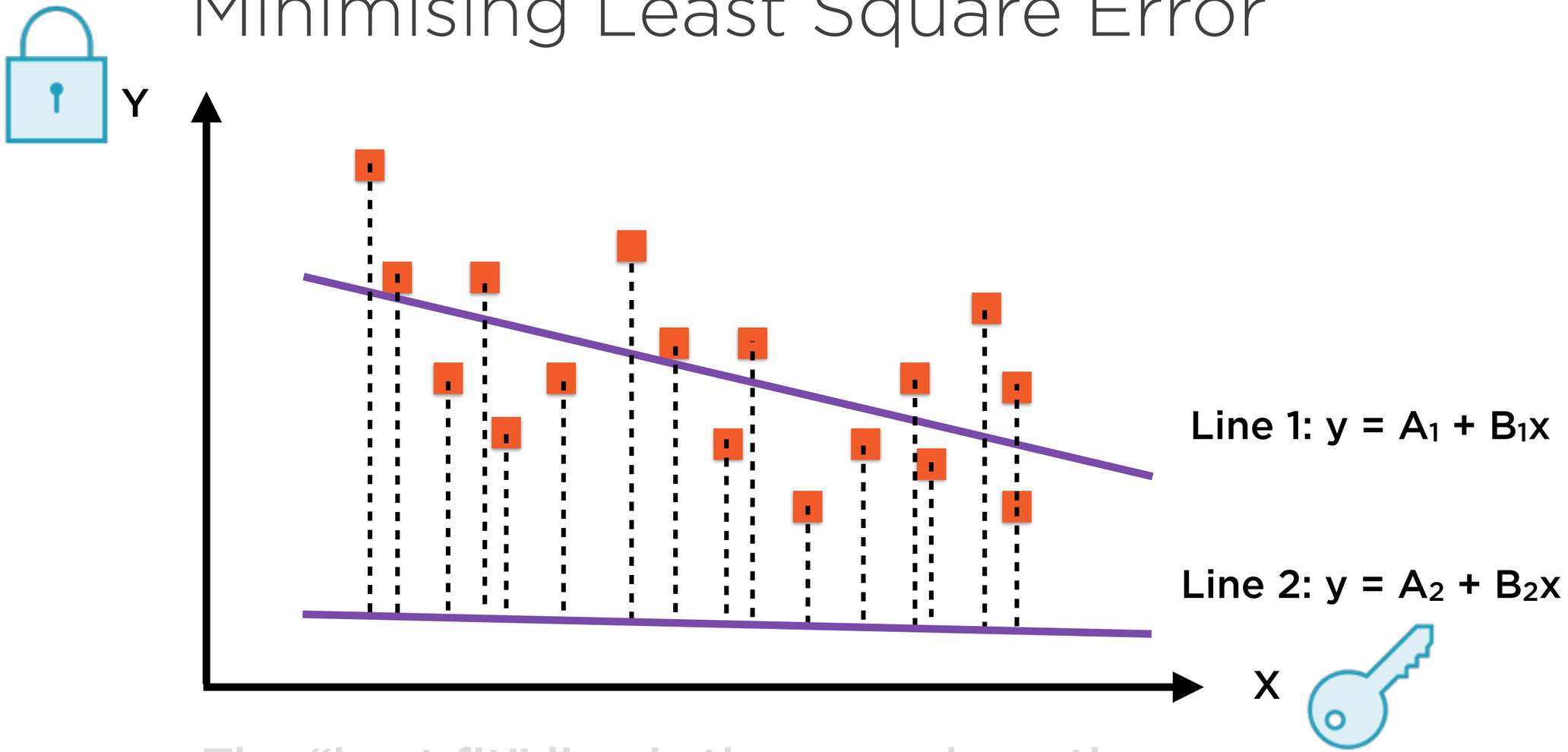
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_n \end{bmatrix} = A \begin{bmatrix} 1 \\ 1 \\ 1 \\ \dots \\ 1 \end{bmatrix} + B \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \dots \\ e_n \end{bmatrix}$$

# Minimising Least Square Error



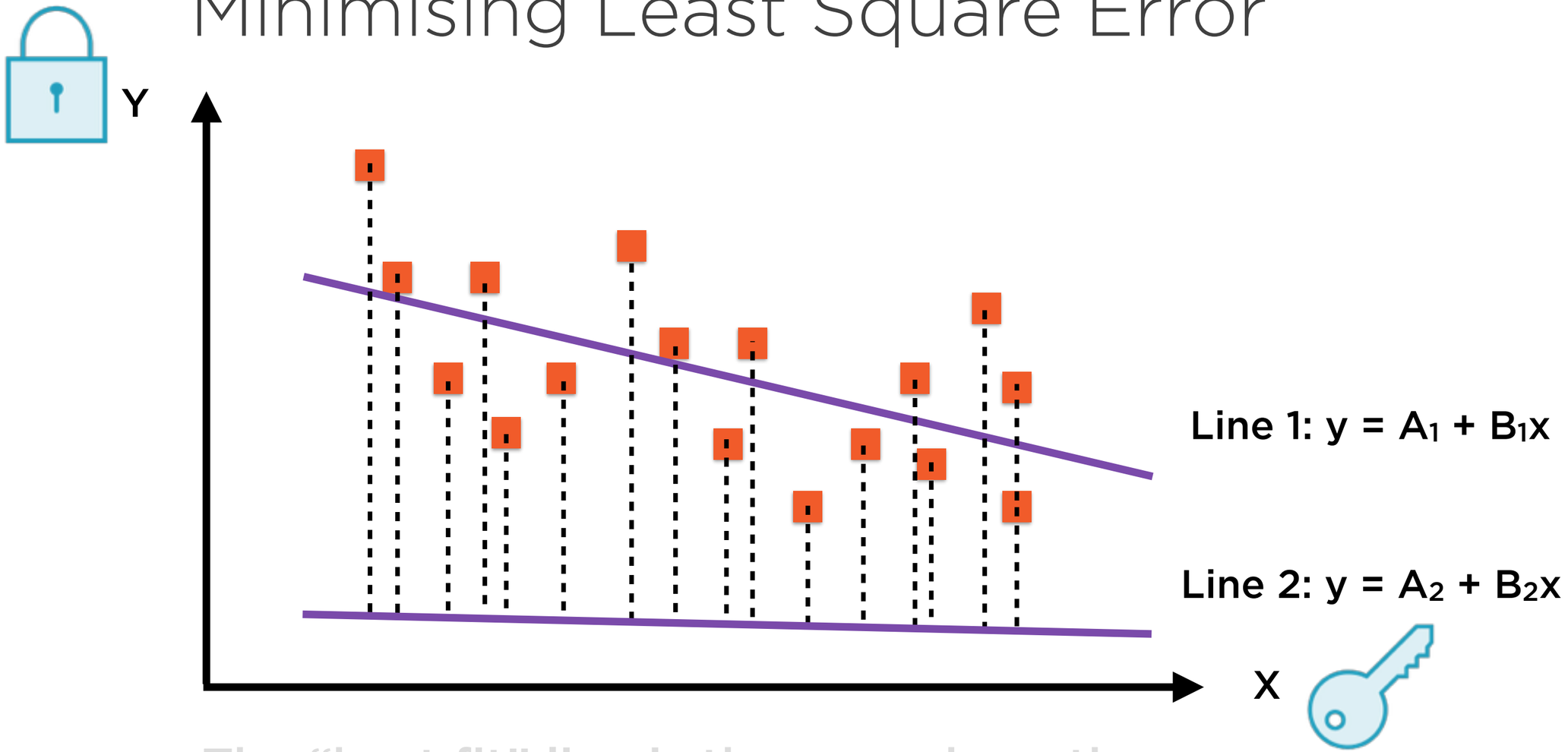

The “best fit” line is the one where the sum of the squares of the lengths of these dotted lines is minimum

# Minimising Least Square Error



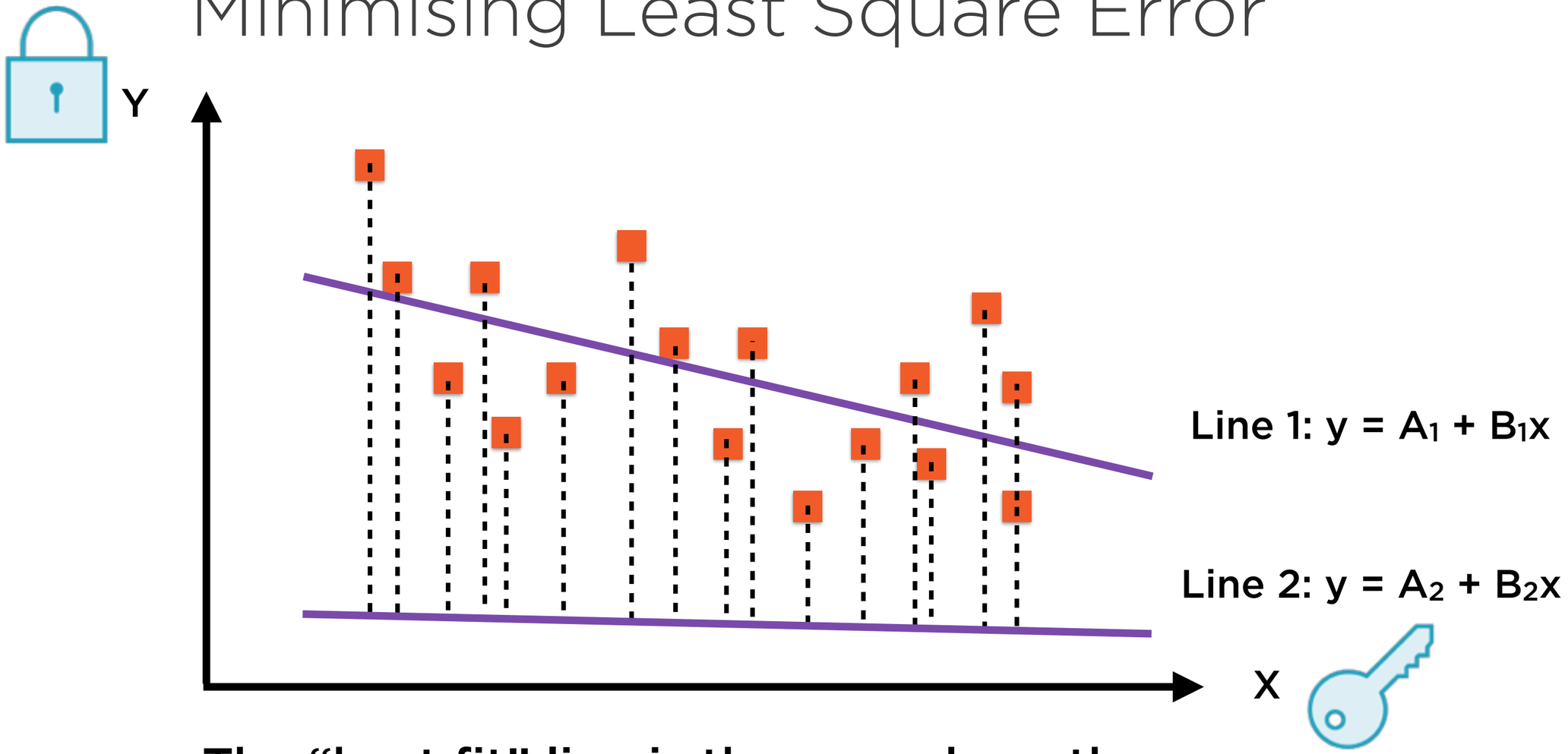

The "best fit" line is the one where the sum of the squares of the lengths of **these dotted lines** is minimum

# Minimising Least Square Error




The "best fit" line is the one where the sum of the squares of the lengths of **the errors** is minimum

# Minimising Least Square Error




**The "best fit" line is the one where the sum of the squares of the lengths of the errors is minimum**

# Simple Regression

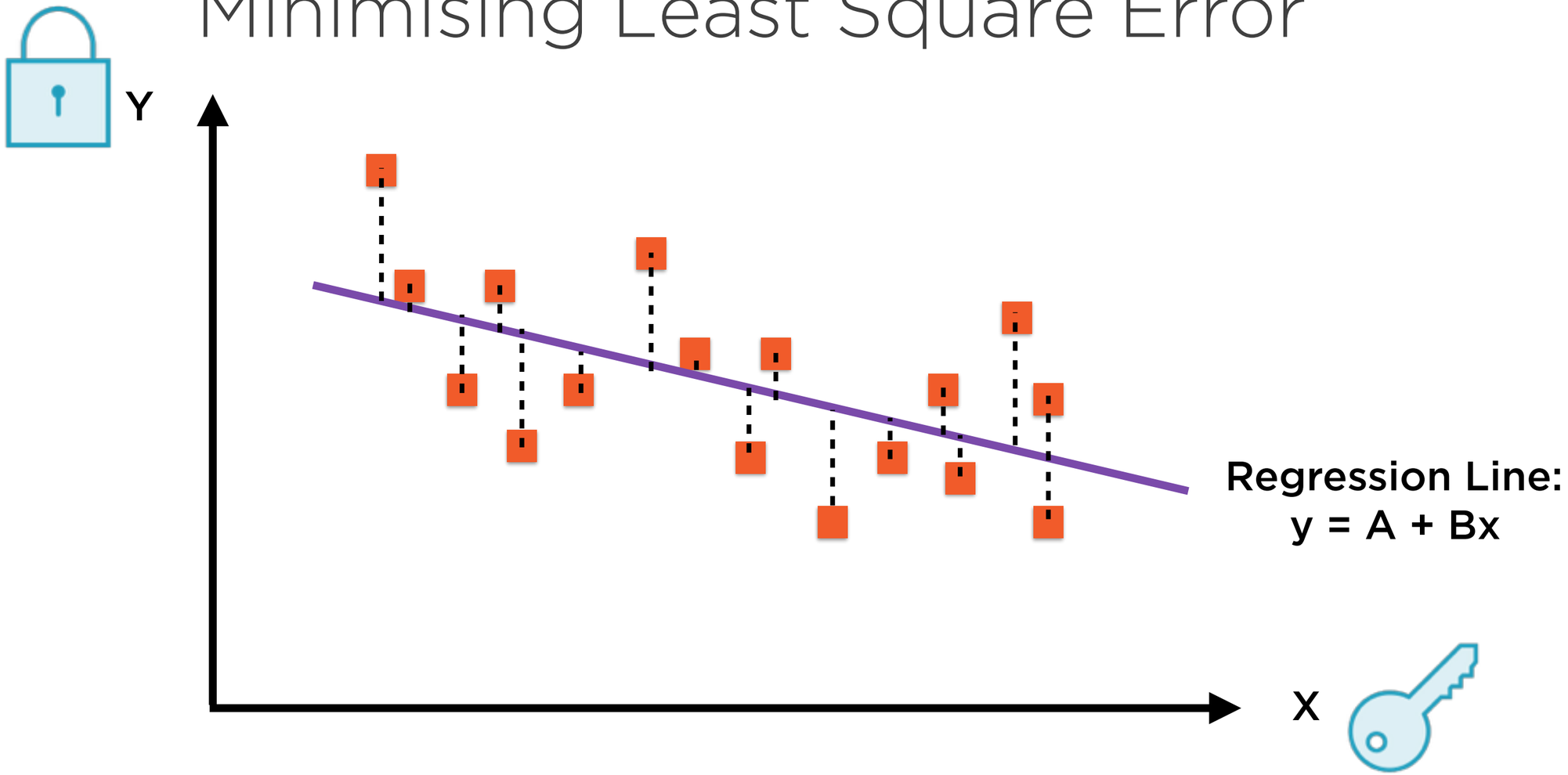
**Regression Equation:**

$$y = A + Bx$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_n \end{bmatrix} = A \begin{bmatrix} 1 \\ 1 \\ 1 \\ \dots \\ 1 \end{bmatrix} + B \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \dots \\ e_n \end{bmatrix}$$

**The “best fit” line is the one where the sum of the squares of the lengths of the errors is minimum**

# Minimising Least Square Error




The “best fit” line is called the regression line

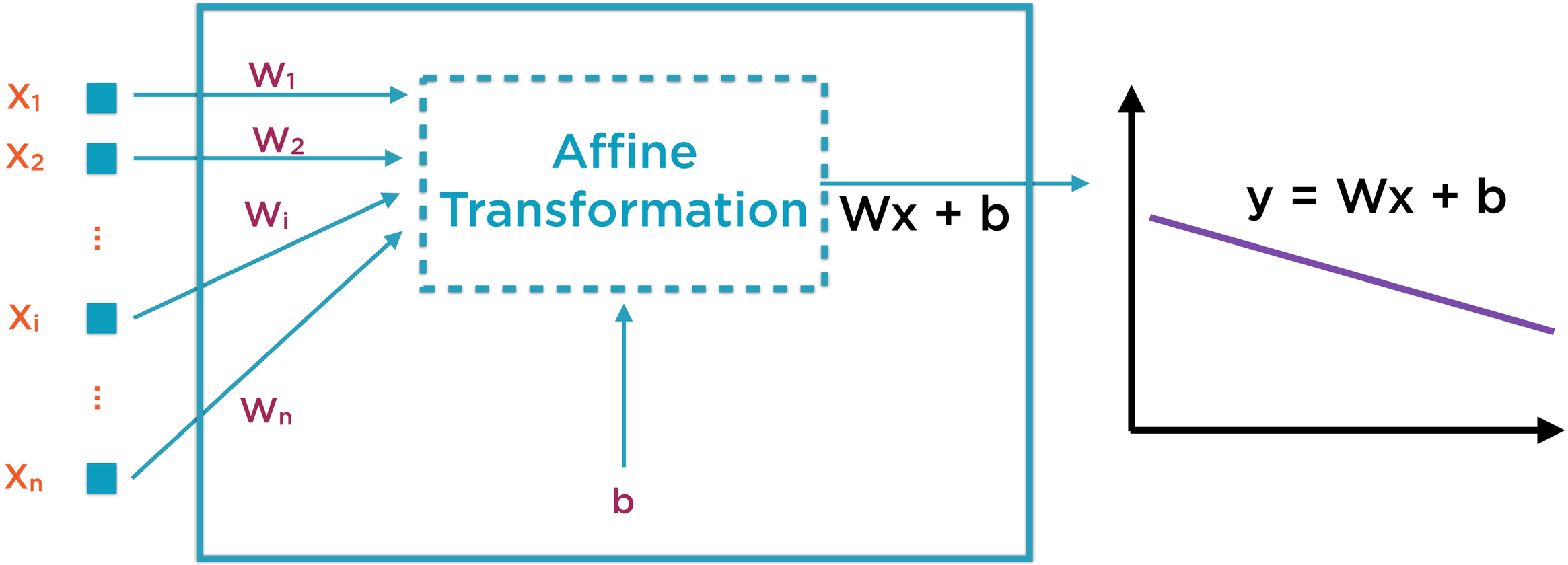
# Optimizers for the “Best-fit”

Method of moments

Method of least squares

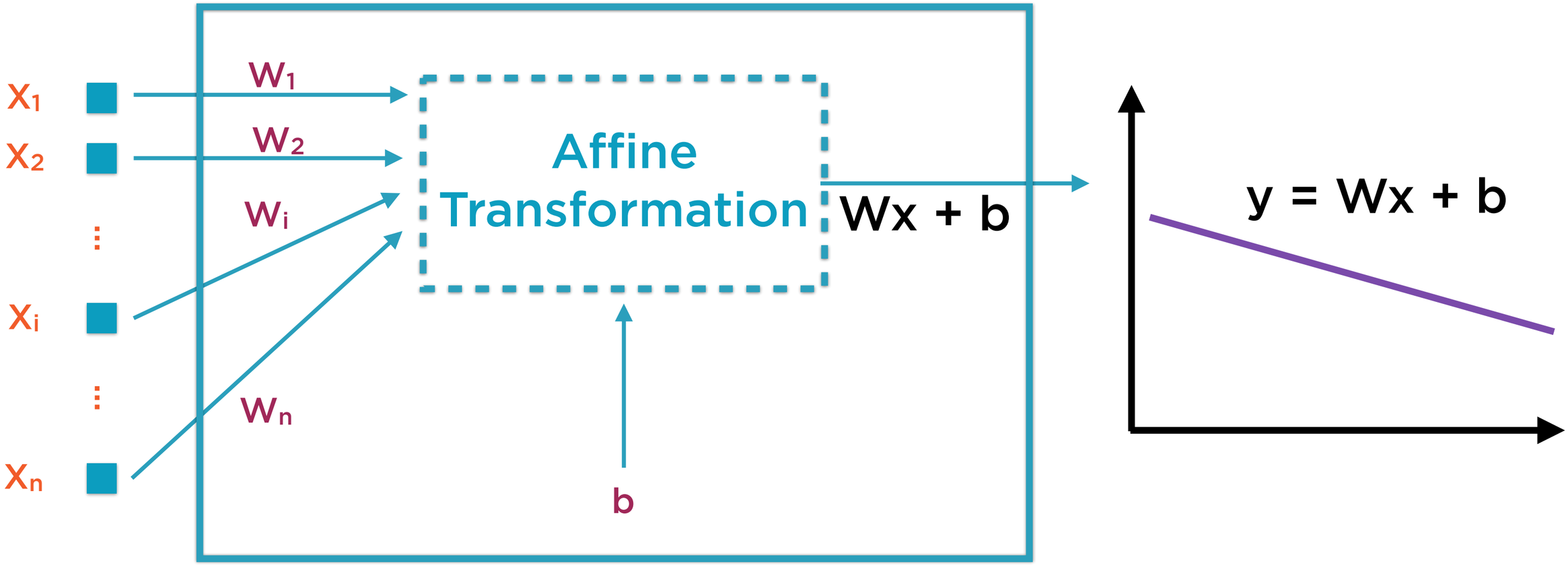
Maximum likelihood estimation

# Operation of a Single Neuron




**Where do the weights W and the bias b come from?**

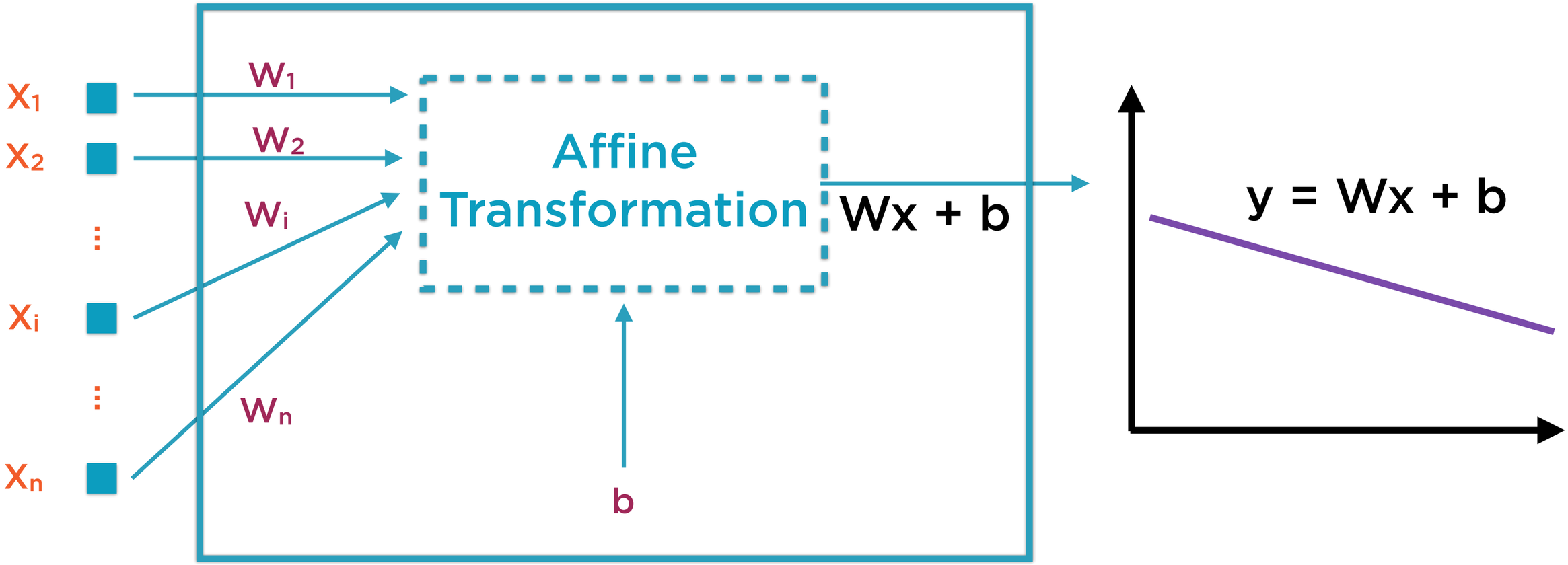
# Operation of a Single Neuron




**Where do the weights W and the bias b come from?**

**They are determined during the training process**

# Operation of a Single Neuron




**This optimization is not carried out by the individual neuron, rather a training algorithm will take care of this**

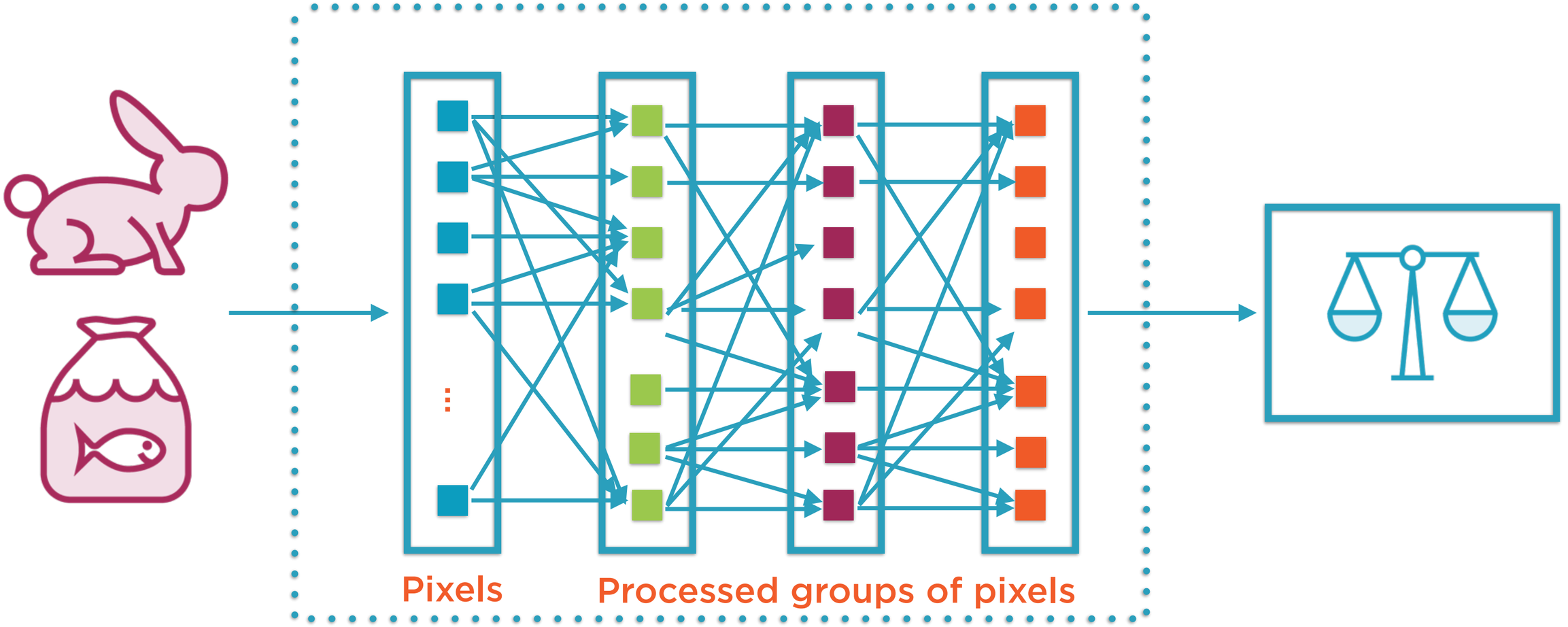
# A Slightly More Complex Neural Network

```
def doSomethingReallyComplicated(x1,x2…):
   …
   …
   …
   return complicatedResult
```

# “Learning” Arbitrarily Complex Functions

**Adding layers to a neural network can “learn” (reverse-engineer) pretty much anything**

# An Arbitrarily Complex Function




Corpus of Images

Each layer consists of individual interconnected neurons

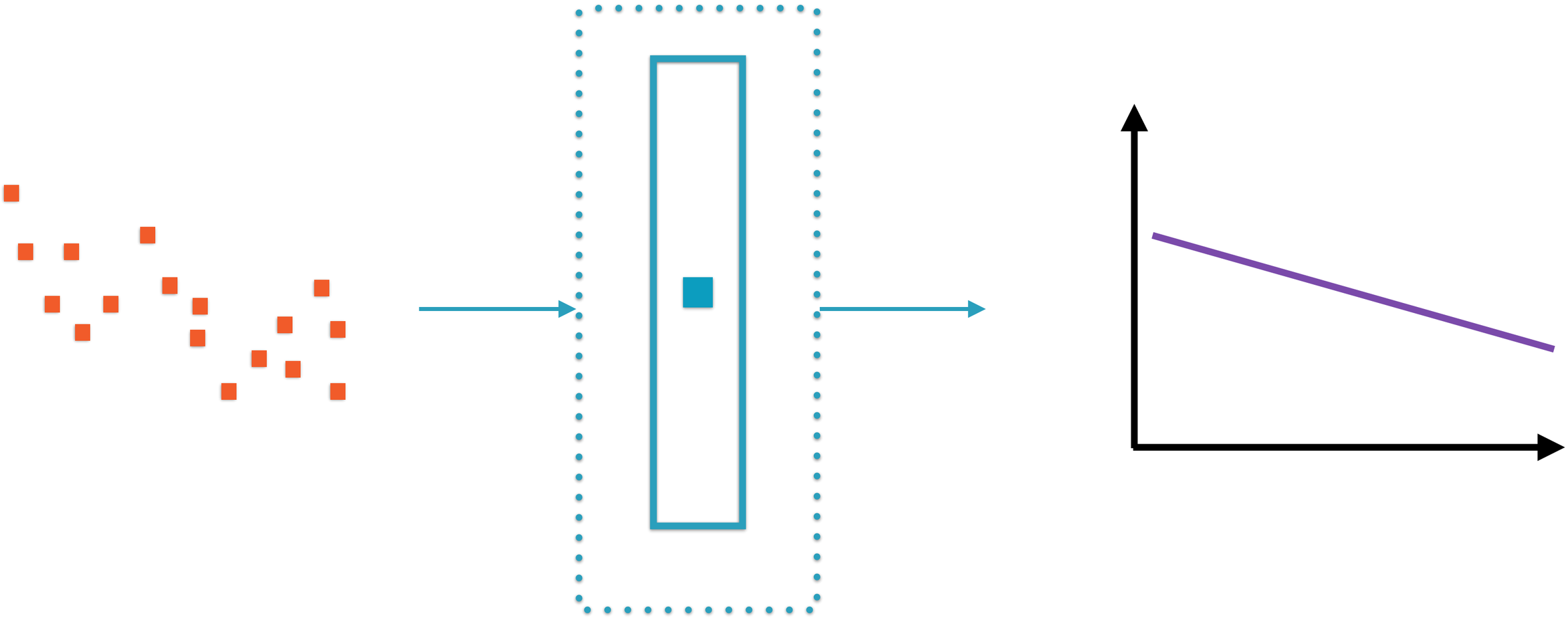
ML-based Classifier

```
y = Wx + b
```

# “Learning” Regression

**Regression can be learnt by a single neuron using an affine transformation alone**

# Regression: The Simplest Neural Network



Set of Points

Single Neuron

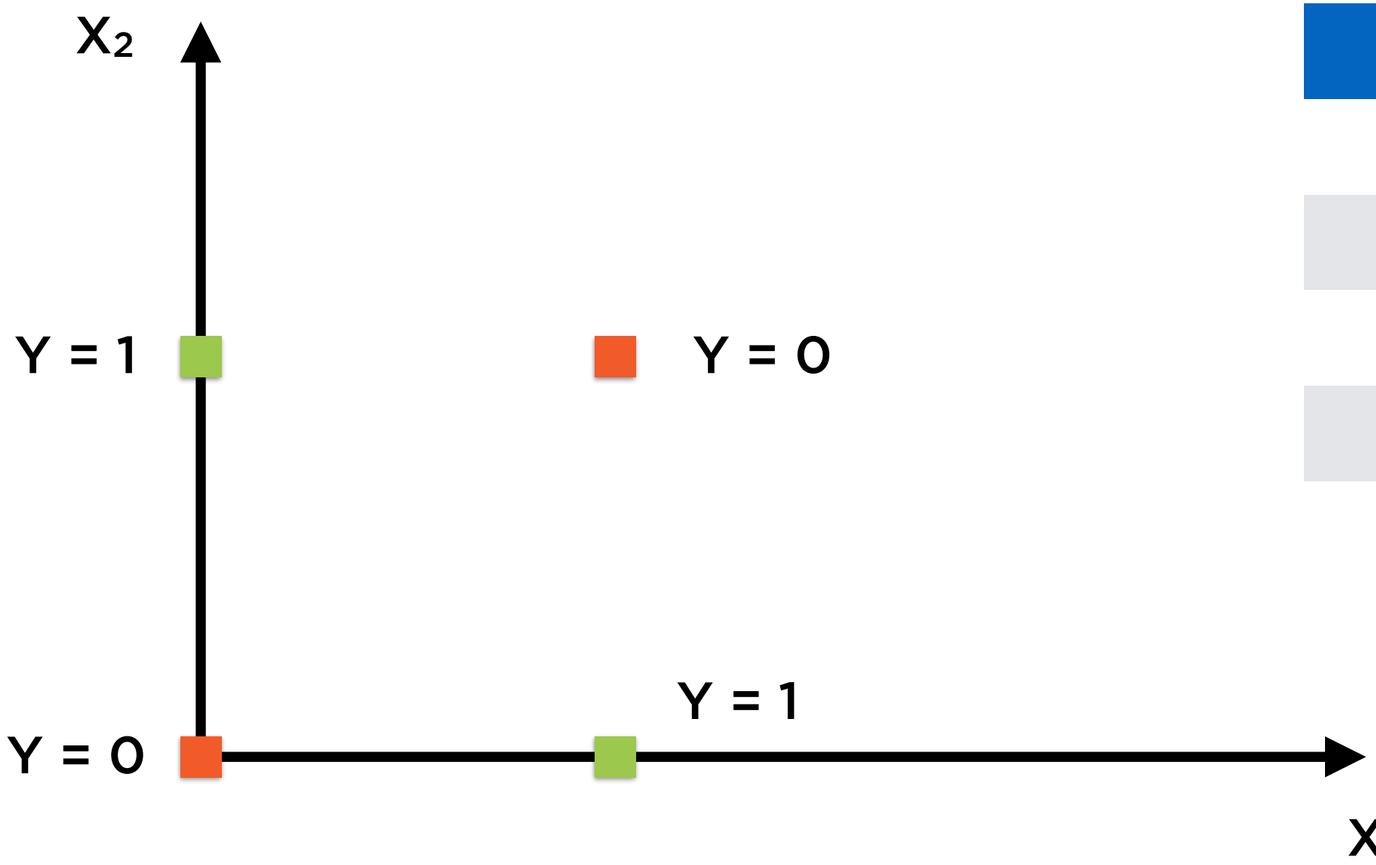
Regression Line

```
def XOR(x1,x2):
  if (x1 == x2):
     return 0
  return 1
```

# “Learning” XOR

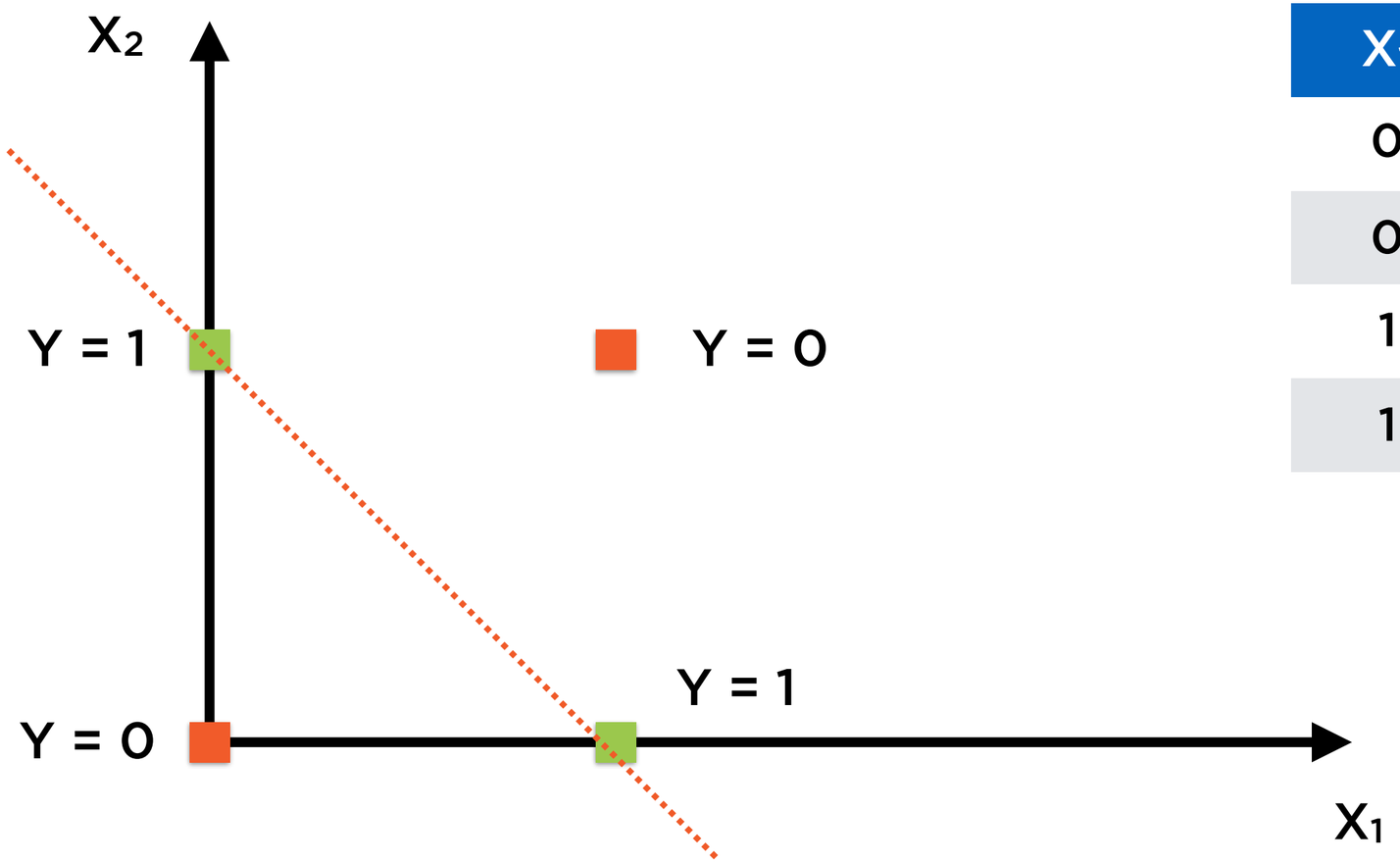
**Reverse-engineering XOR requires 3 neurons (arranged in 2 layers) as well as a non-linear activation function**

# XOR: Not Linearly Separable




| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

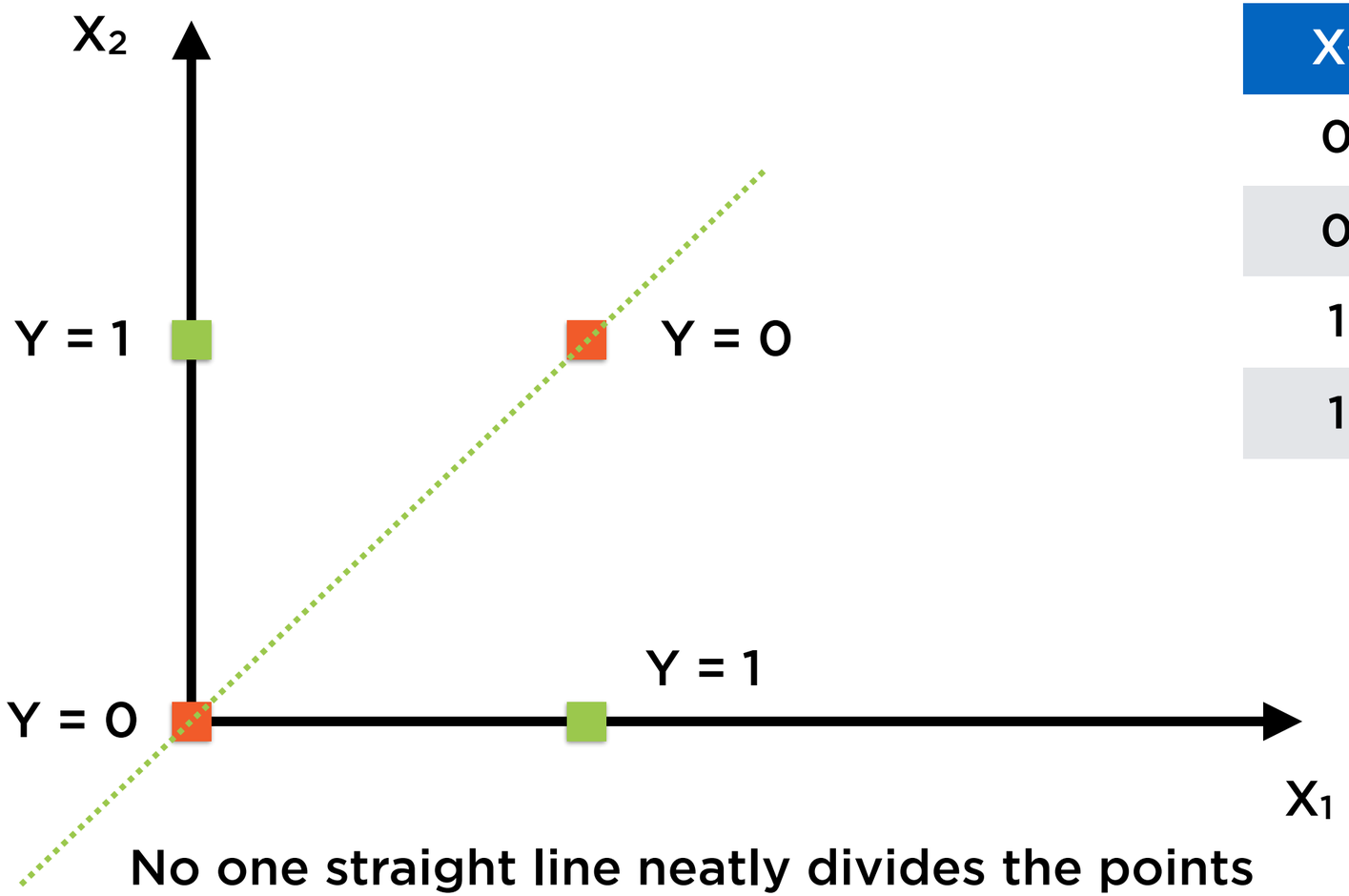
# XOR: Not Linearly Separable




| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**No one straight line neatly divides the points into disjoint regions where Y = 0 and Y = 1**

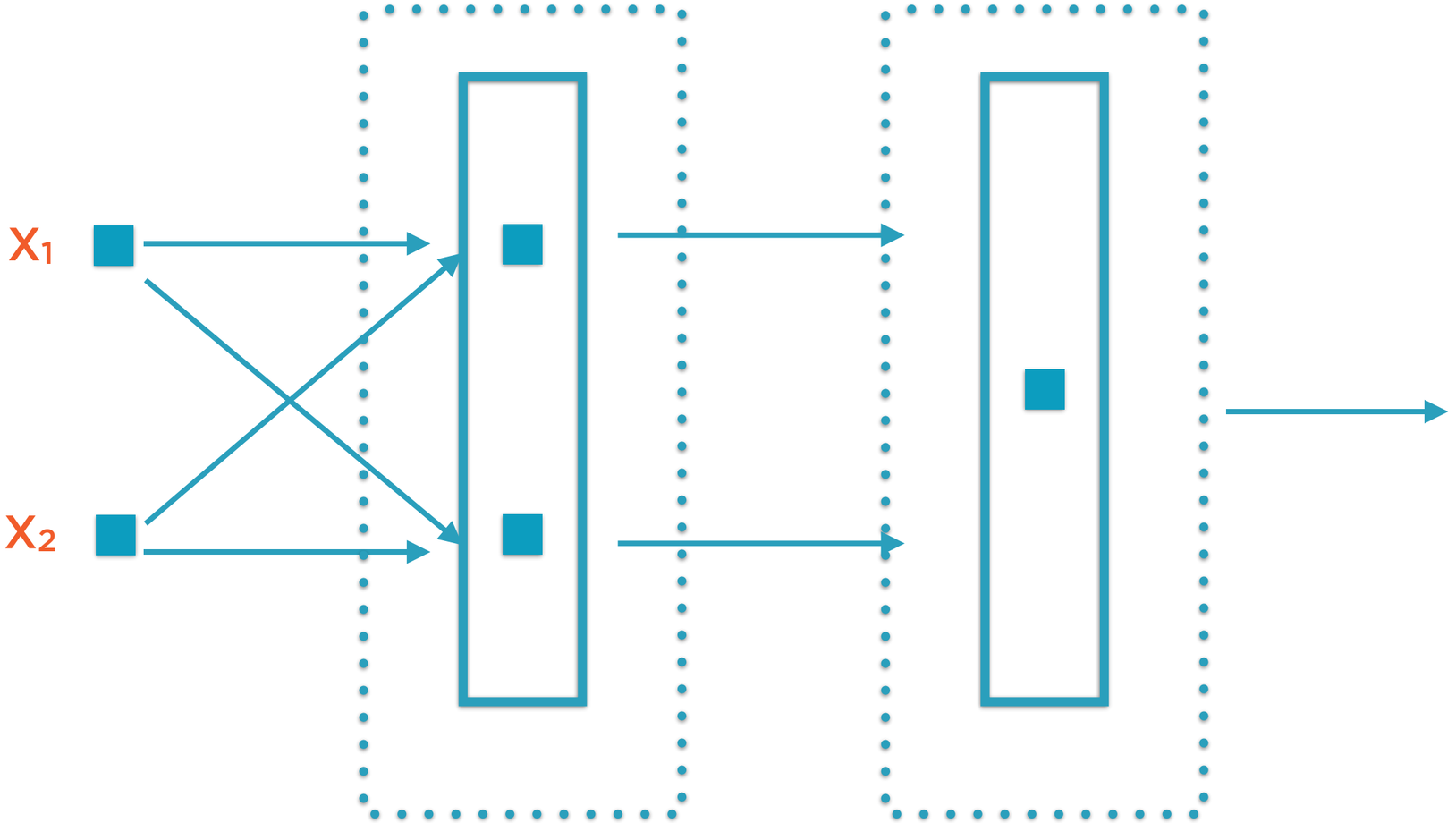
# XOR: Not Linearly Separable




| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**No one straight line neatly divides the points into disjoint regions where Y = 0 and Y = 1**

# XOR: 3 Neurons, 2 Layers




| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

2 Inputs

Layer 1

Layer 2

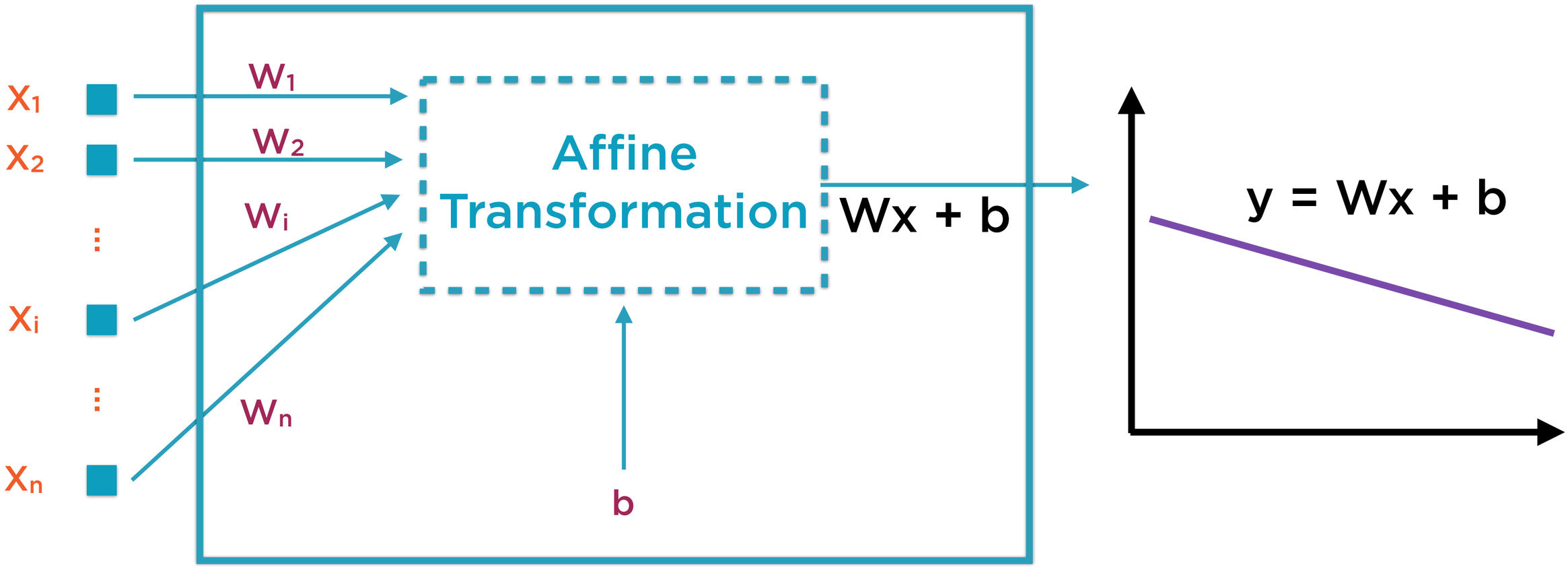
Output Truth Table

| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# “Learning” XOR

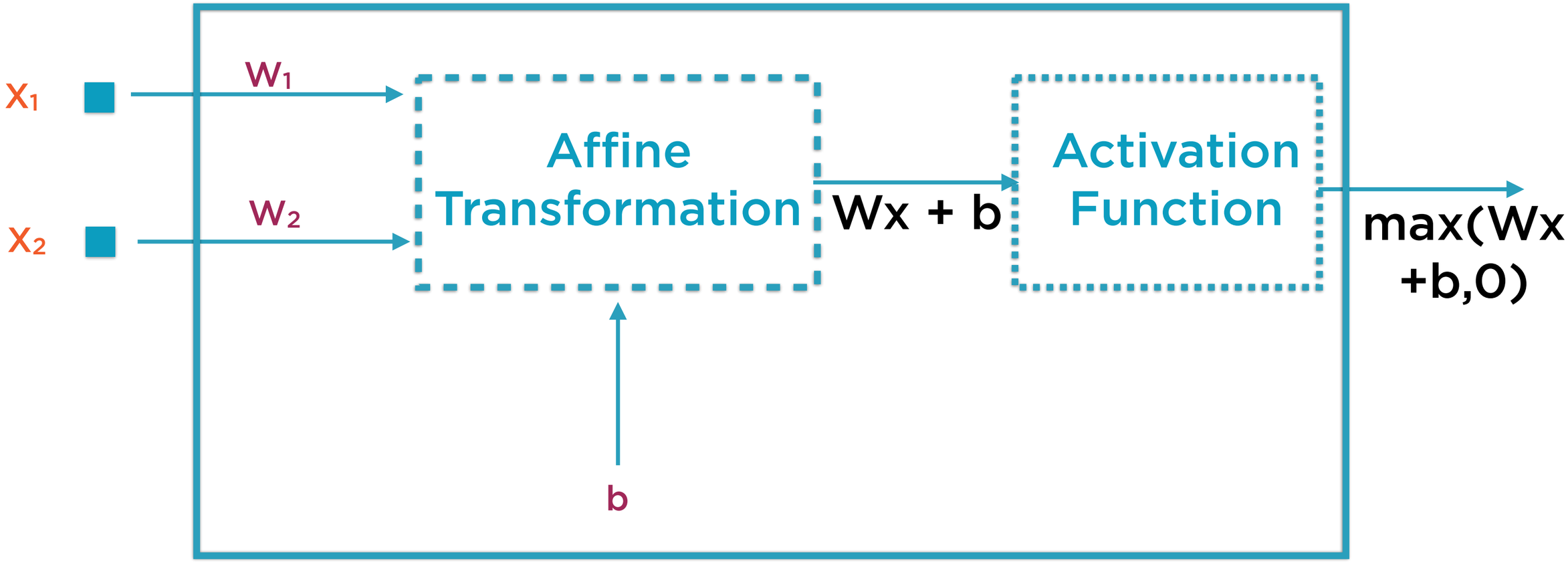
**Reverse-engineering XOR requires 3 neurons (arranged in 2 layers) as well as a non-linear activation function**

# 1-Neuron Regression




**Regression could be learnt by a single neuron using a single, linear operation**

# Adding an Activation Function




**XOR, a simple non-linear function, can be learnt by 3 neurons if we add an appropriate activation function**

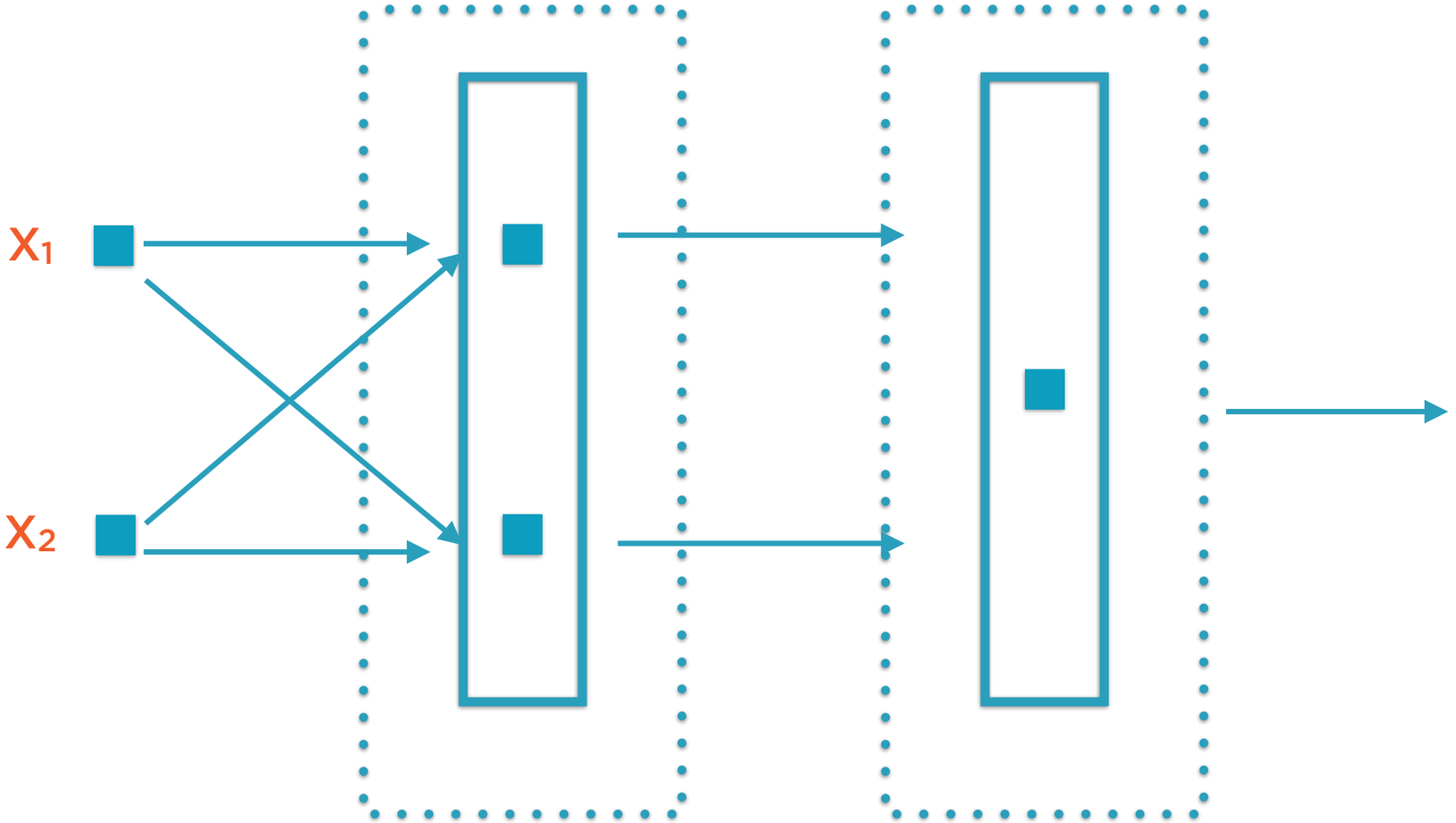

The activation function is needed for the neural network to predict non-linear functions


The most common form of the activation function is the ReLU

ReLU : Rectified Linear Unit

ReLU(x) = max(0,x)

# XOR: 3 Neurons, 2 Layers




| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

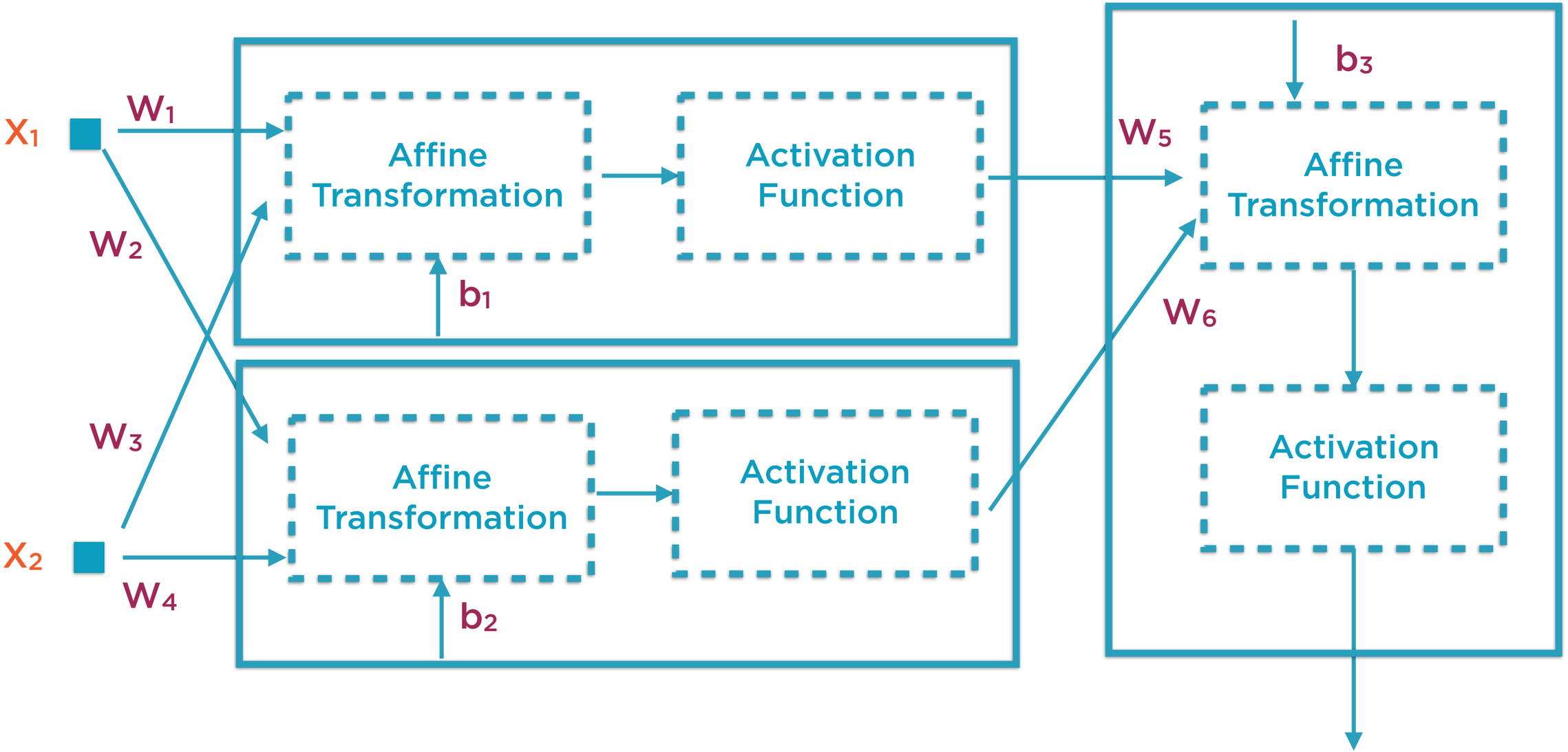
2 Inputs

Layer 1

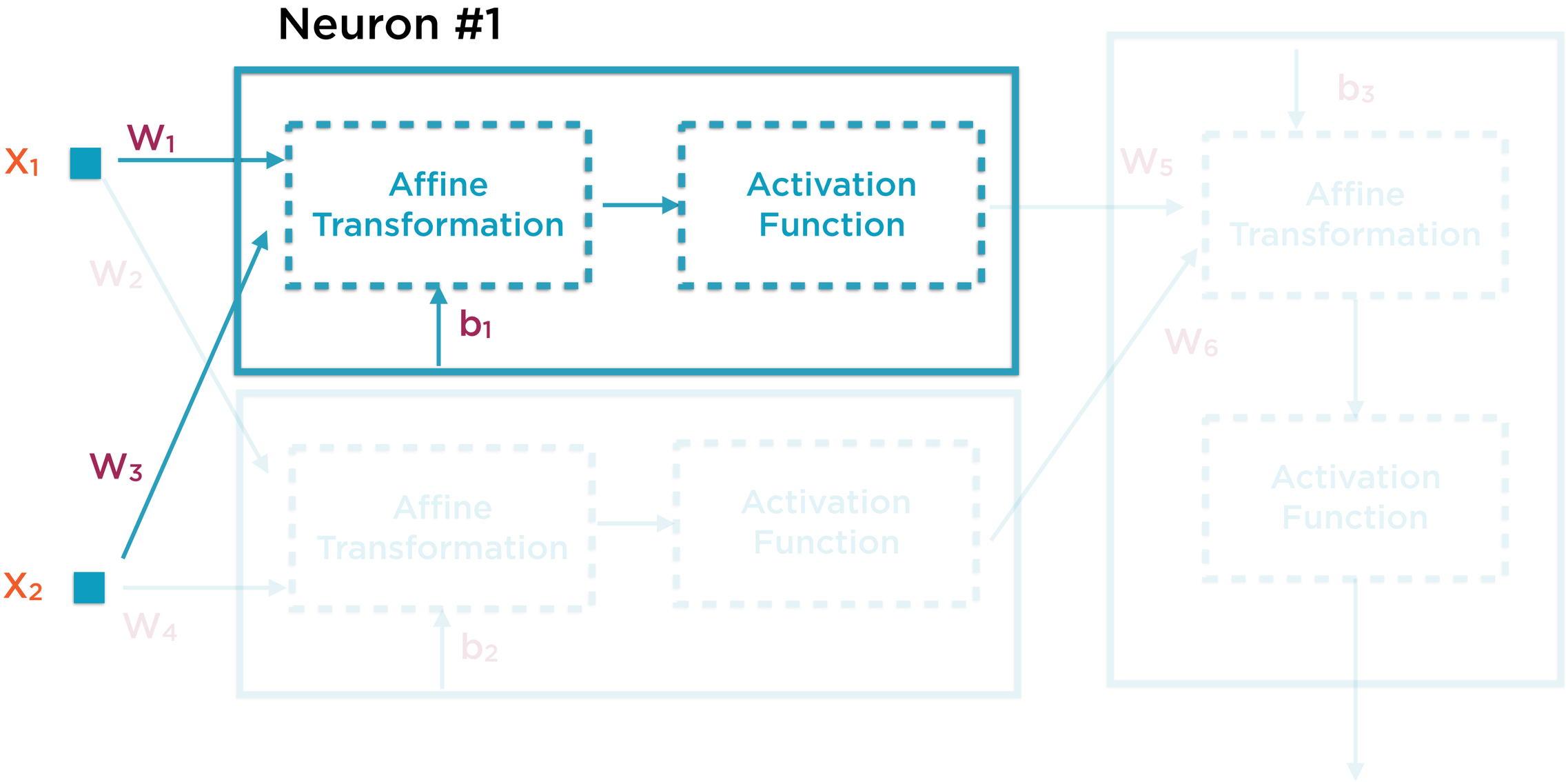
Layer 2

Output Truth Table

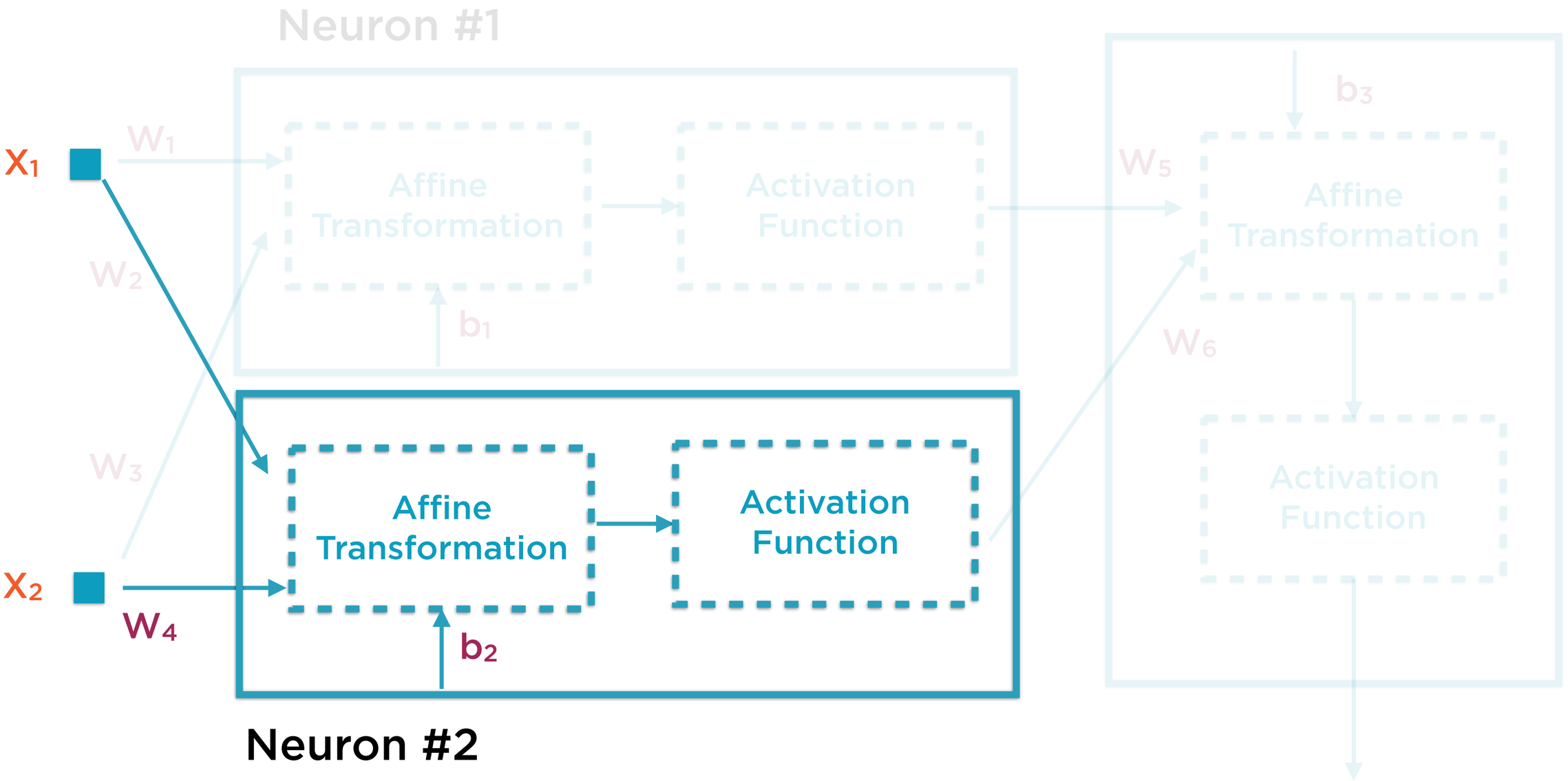
# 3-Neuron XOR


$x_1$
$x_2$
$W_1$
$W_2$
$W_3$
$W_4$
$W_5$
$W_6$
$b_1$
$b_2$
$b_3$
Affine Transformation
Activation Function
Affine Transformation
Activation Function
Affine Transformation
Activation Function

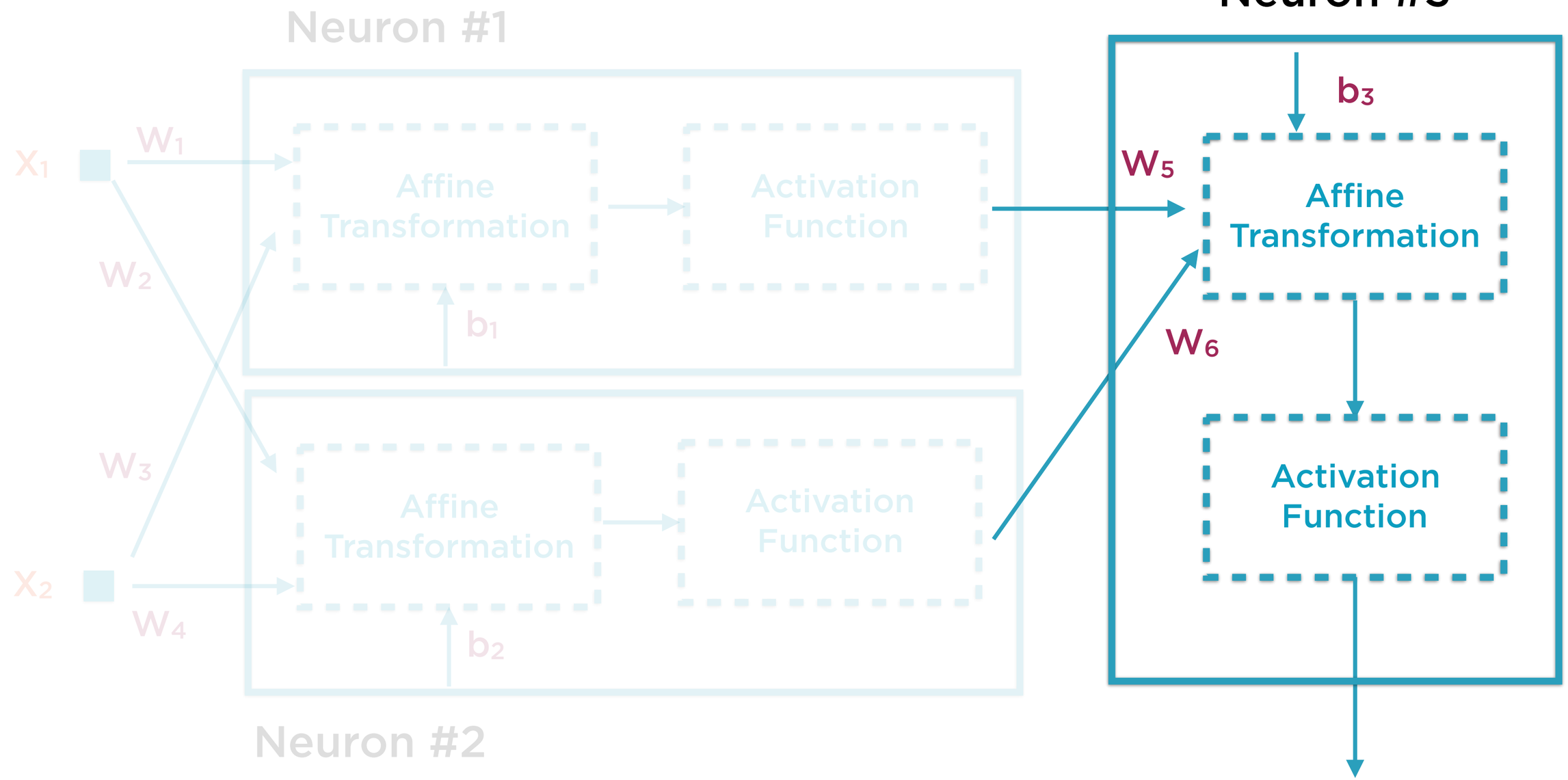
# 3-Neuron XOR


Neuron #1
$X_1$
$X_2$
$W_1$
$W_2$
$W_3$
$W_4$
$W_5$
$W_6$
Affine Transformation
Activation Function
$b_1$
$b_2$
$b_3$
Affine Transformation
Activation Function
Affine Transformation
Activation Function

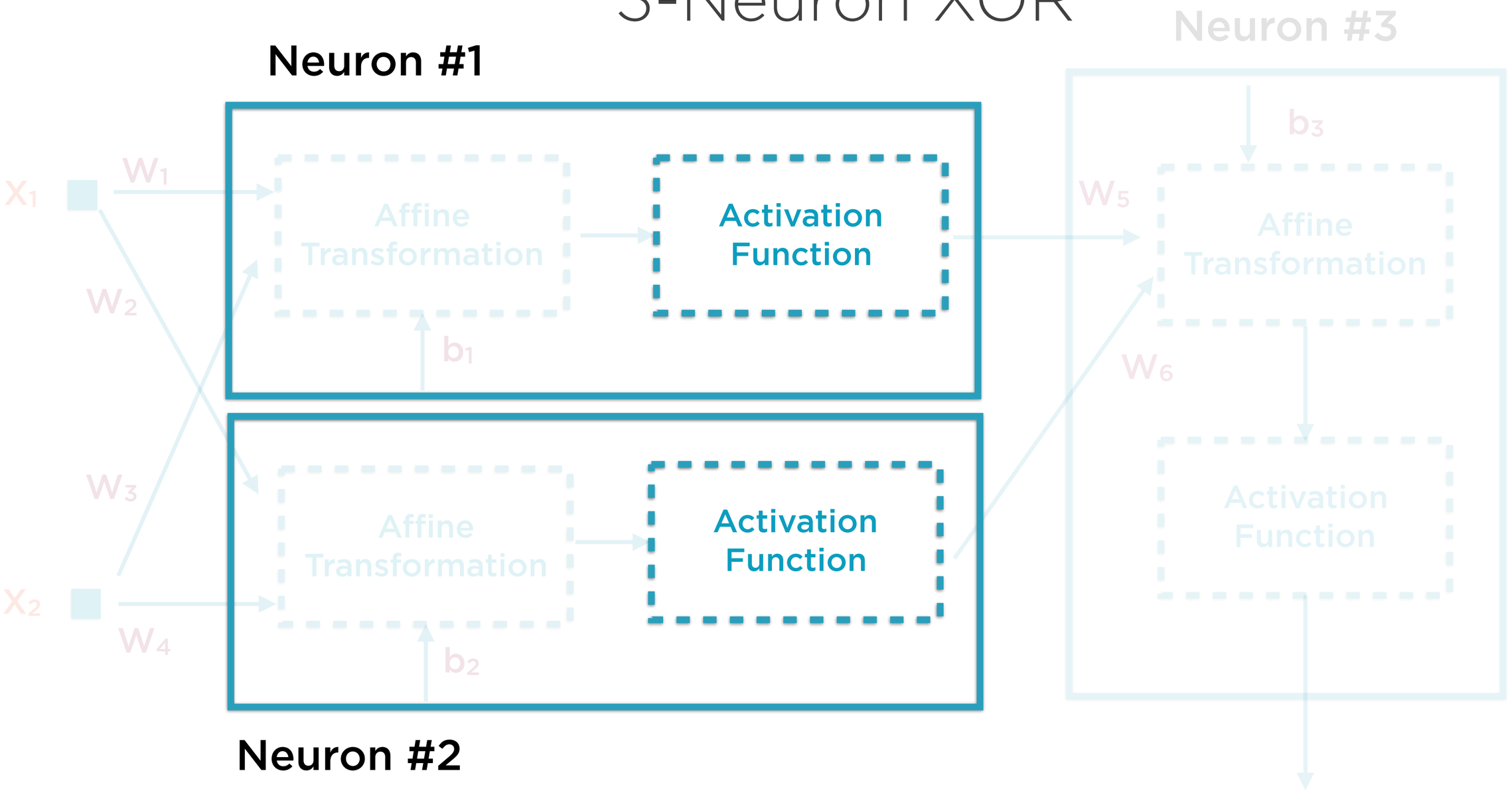
# 3-Neuron XOR


Neuron #1
Affine Transformation
Activation Function
$b_1$
$W_1$
$W_2$
$W_3$
$W_4$
$W_5$
$W_6$
$b_3$
$X_1$
$X_2$
Affine Transformation
Activation Function
$b_2$
Neuron #2

# 3-Neuron XOR


Neuron #1
Affine Transformation
Activation Function
$b_1$
$W_1$
$W_2$
$X_1$
$X_2$
$W_3$
$W_4$
Affine Transformation
Activation Function
$b_2$
Neuron #2
Neuron #3
$b_3$
$W_5$
Affine Transformation
$W_6$
Activation Function

# 3-Neuron XOR

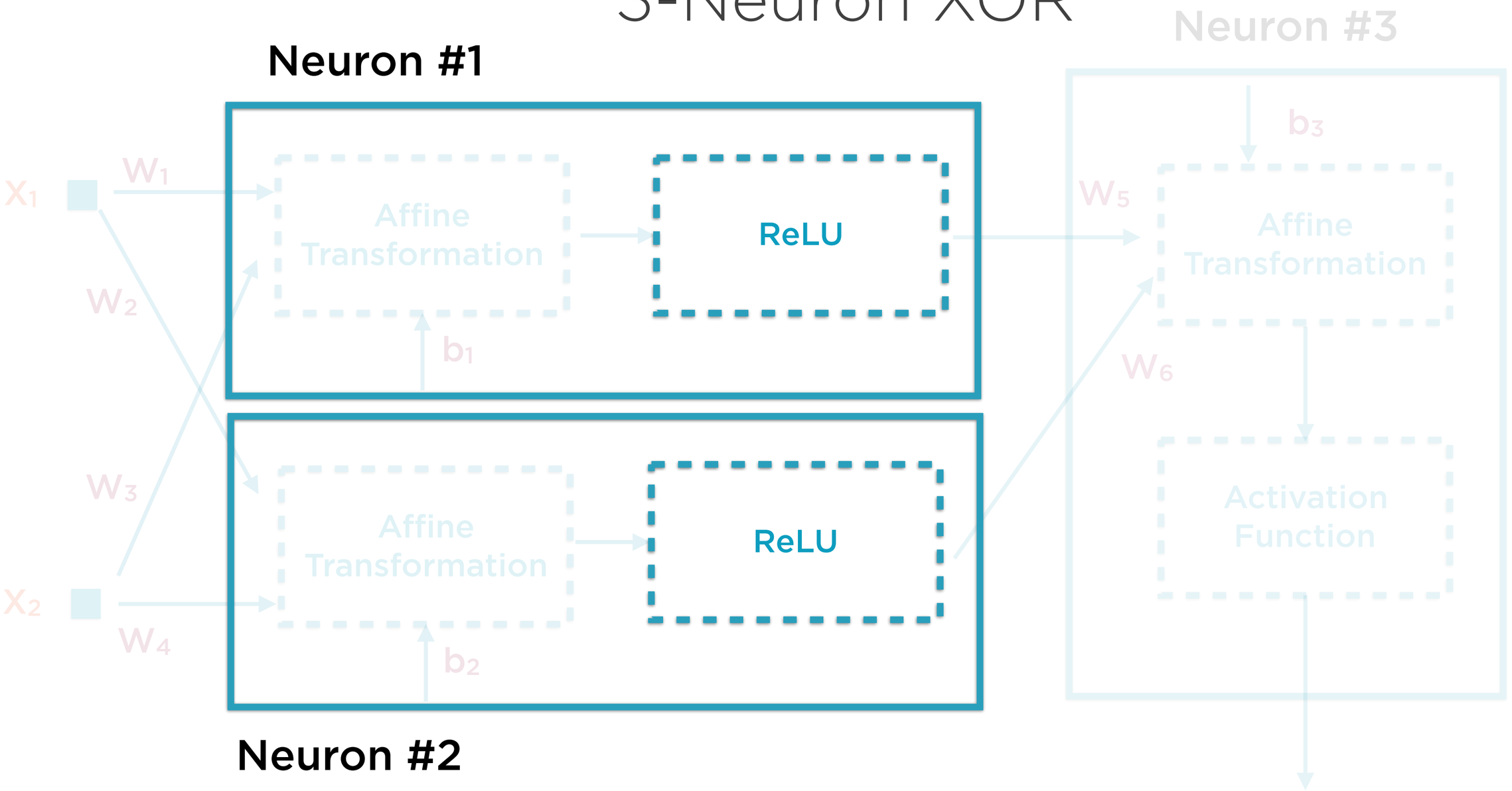

Neuron #1
Affine Transformation
Activation Function
Neuron #2
Affine Transformation
Activation Function
Neuron #3
Affine Transformation
Activation Function
$x_1$
$x_2$
$w_1$
$w_2$
$w_3$
$w_4$
$w_5$
$w_6$
$b_1$
$b_2$
$b_3$


The most common form of the activation function is the ReLU

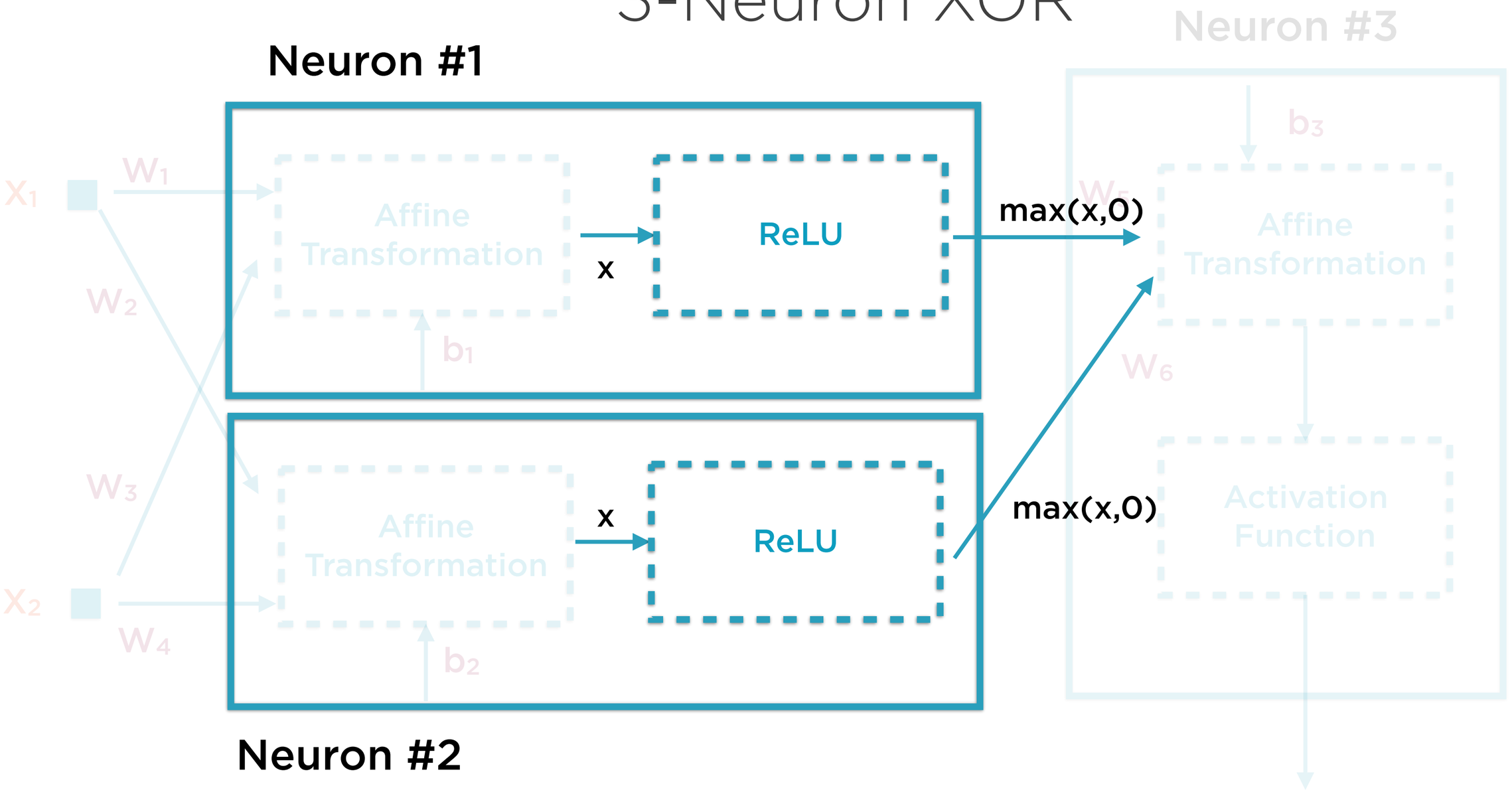
ReLU : Rectified Linear Unit

ReLU(x) = max(0,x)

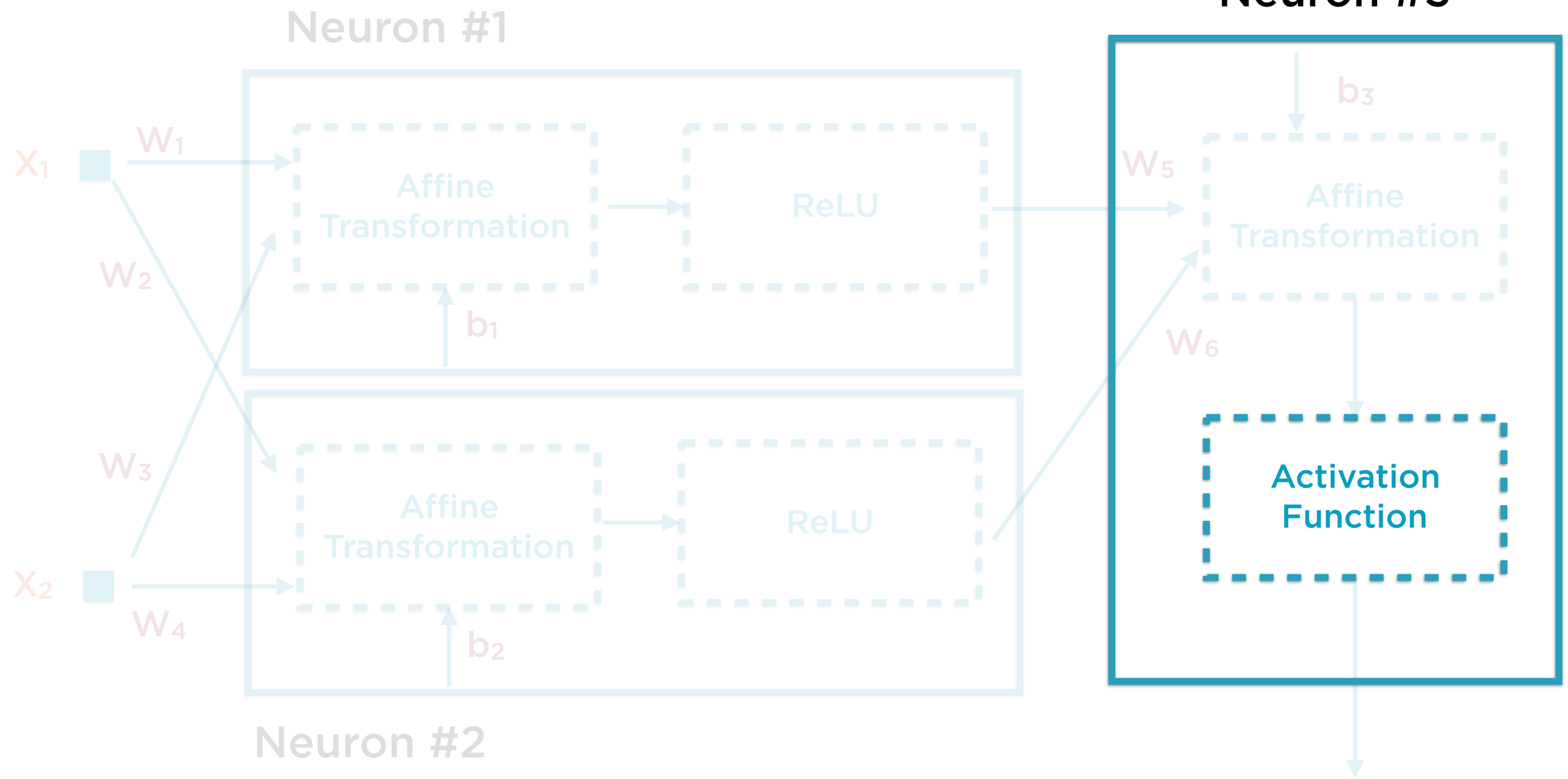
# 3-Neuron XOR


Neuron #1
Neuron #3
$b_3$
$W_1$
$X_1$
$W_5$
Affine Transformation
ReLU
$W_2$
$b_1$
$W_6$
Activation Function
$W_3$
Affine Transformation
ReLU
$X_2$
$W_4$
$b_2$
Neuron #2

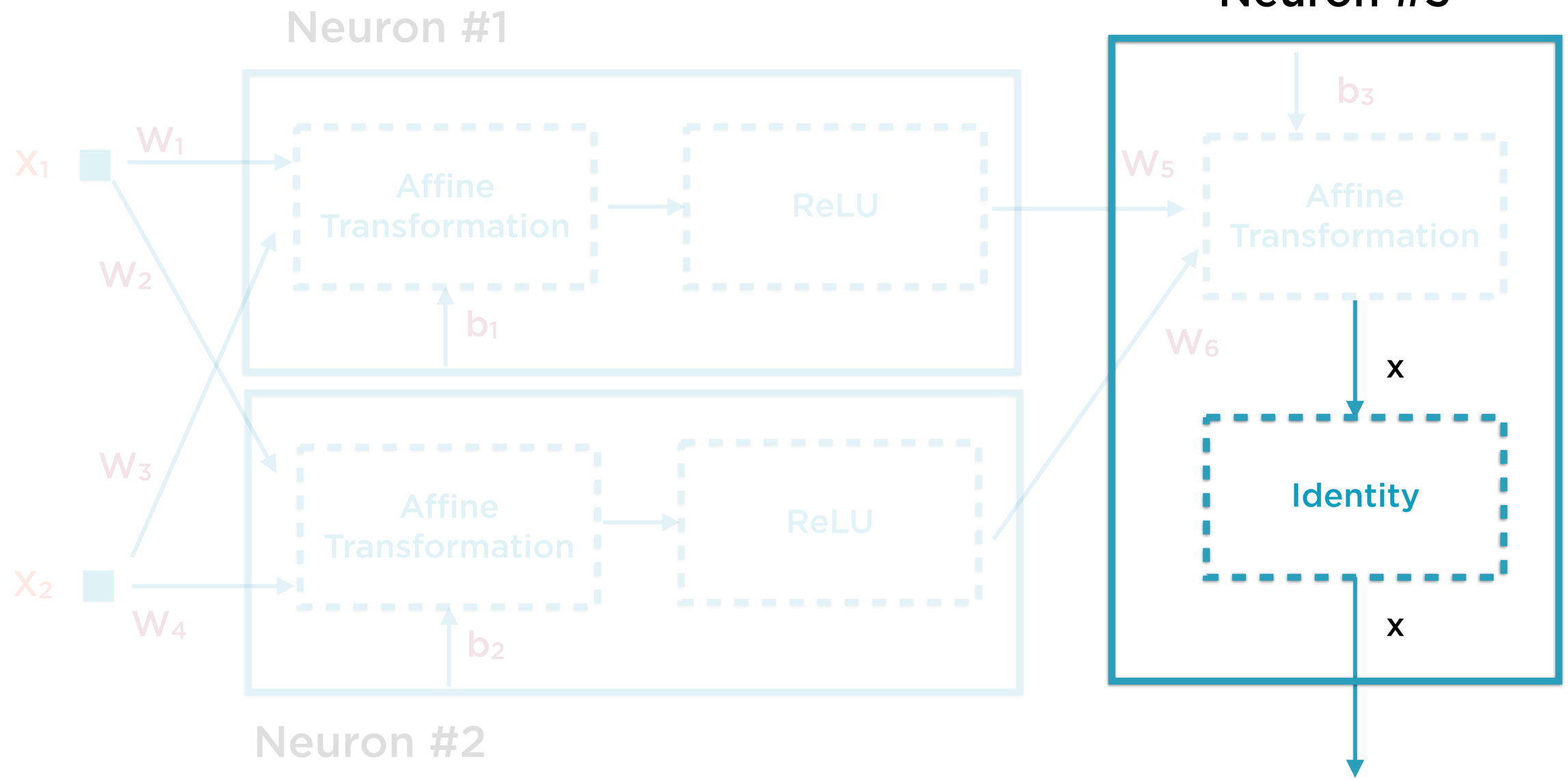
# 3-Neuron XOR


Neuron #1
Neuron #3
$b_3$
$W_5$
Affine Transformation
max(x,0)
$X_1$
$W_1$
Affine Transformation
ReLU
x
$W_2$
$b_1$
$W_6$
Activation Function
$W_3$
x
Affine Transformation
ReLU
max(x,0)
$X_2$
$W_4$
$b_2$
Neuron #2

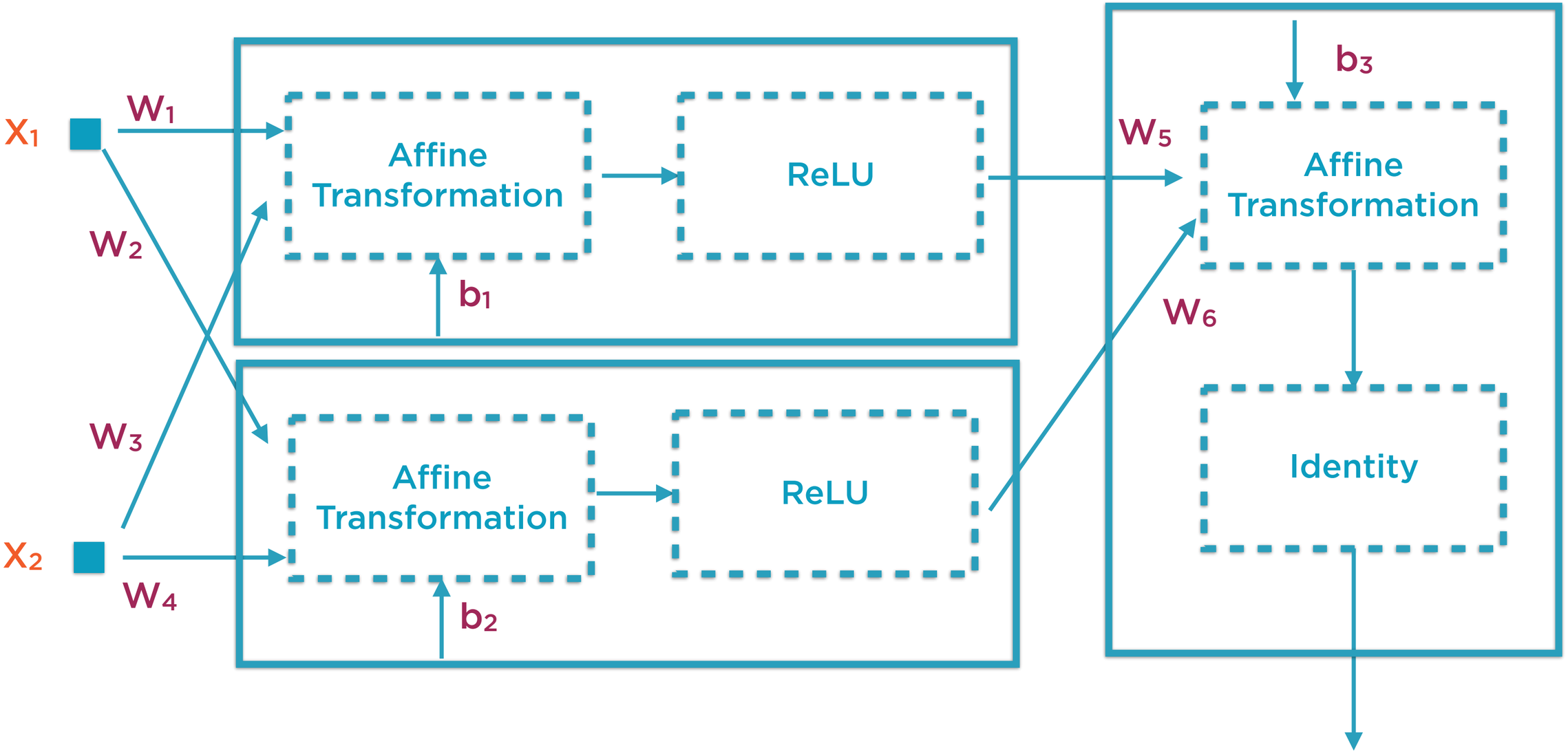
# 3-Neuron XOR


Neuron #1
Neuron #2
Neuron #3
$x_1$
$x_2$
$W_1$
$W_2$
$W_3$
$W_4$
$W_5$
$W_6$
$b_1$
$b_2$
$b_3$
Affine Transformation
ReLU
Affine Transformation
ReLU
Affine Transformation
Activation Function

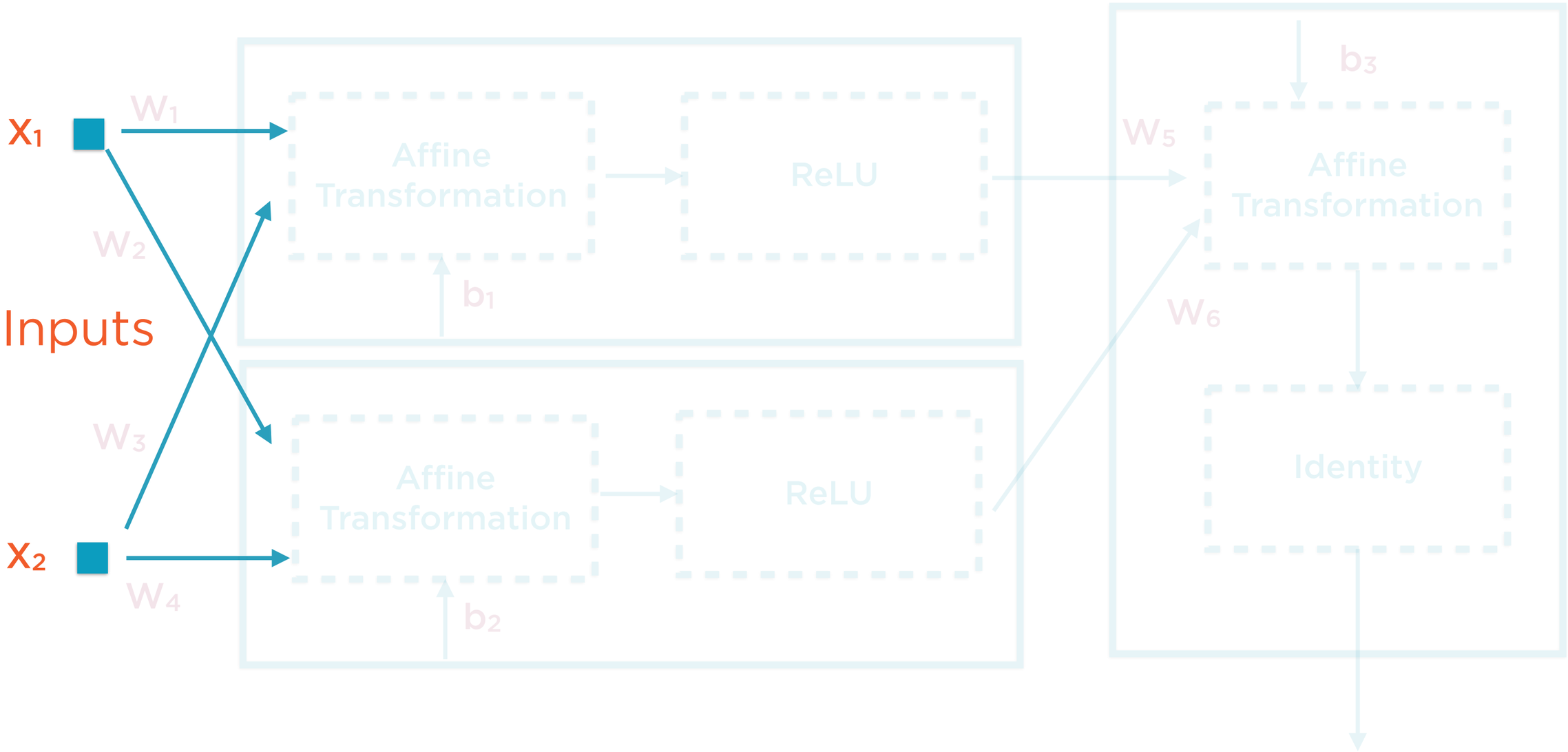
# 3-Neuron XOR


Neuron #1
$x_1$
$w_1$
$w_2$
Affine Transformation
ReLU
$b_1$
$w_3$
$x_2$
$w_4$
Affine Transformation
ReLU
$b_2$
Neuron #2
Neuron #3
$b_3$
$w_5$
Affine Transformation
$w_6$
x
Identity
x

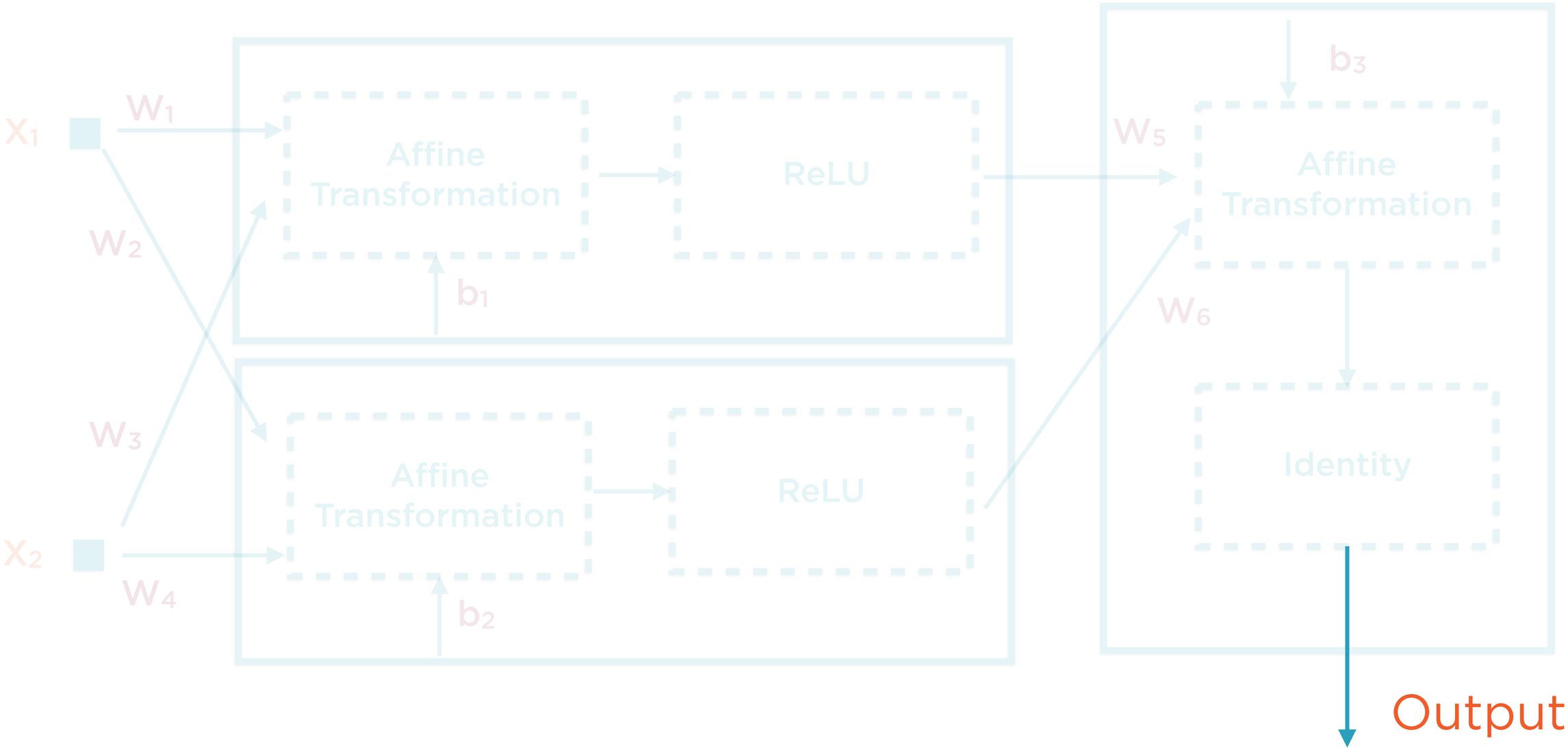
# 3-Neuron XOR


$x_1$
$x_2$
$W_1$
$W_2$
$W_3$
$W_4$
Affine Transformation
ReLU
$b_1$
Affine Transformation
ReLU
$b_2$
$b_3$
$W_5$
Affine Transformation
$W_6$
Identity

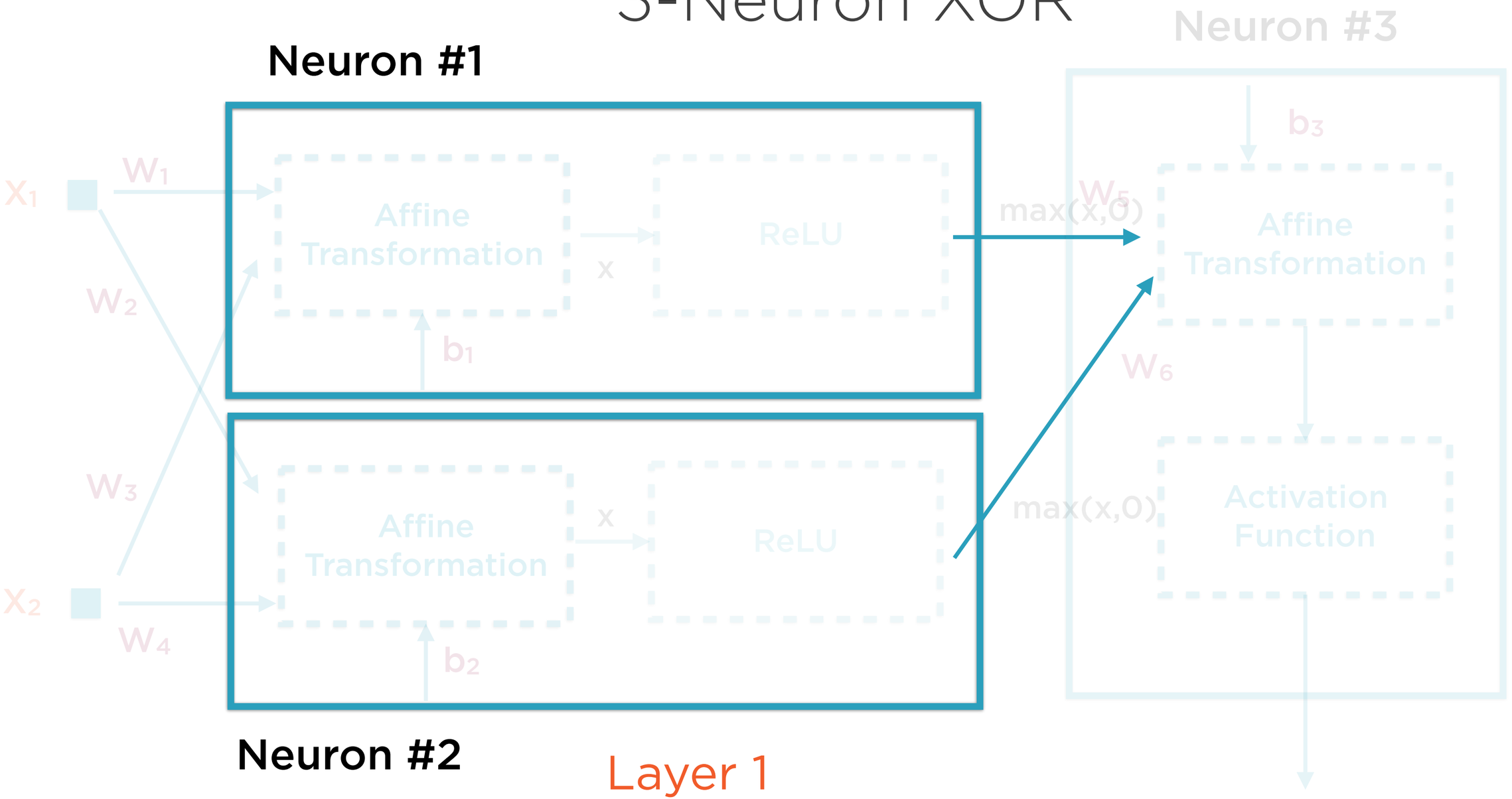
# 3-Neuron XOR


$x_1$
$x_2$
Inputs
$W_1$
$W_2$
$W_3$
$W_4$
Affine Transformation
ReLU
$b_1$
Affine Transformation
ReLU
$b_2$
$b_3$
$W_5$
$W_6$
Affine Transformation
Identity

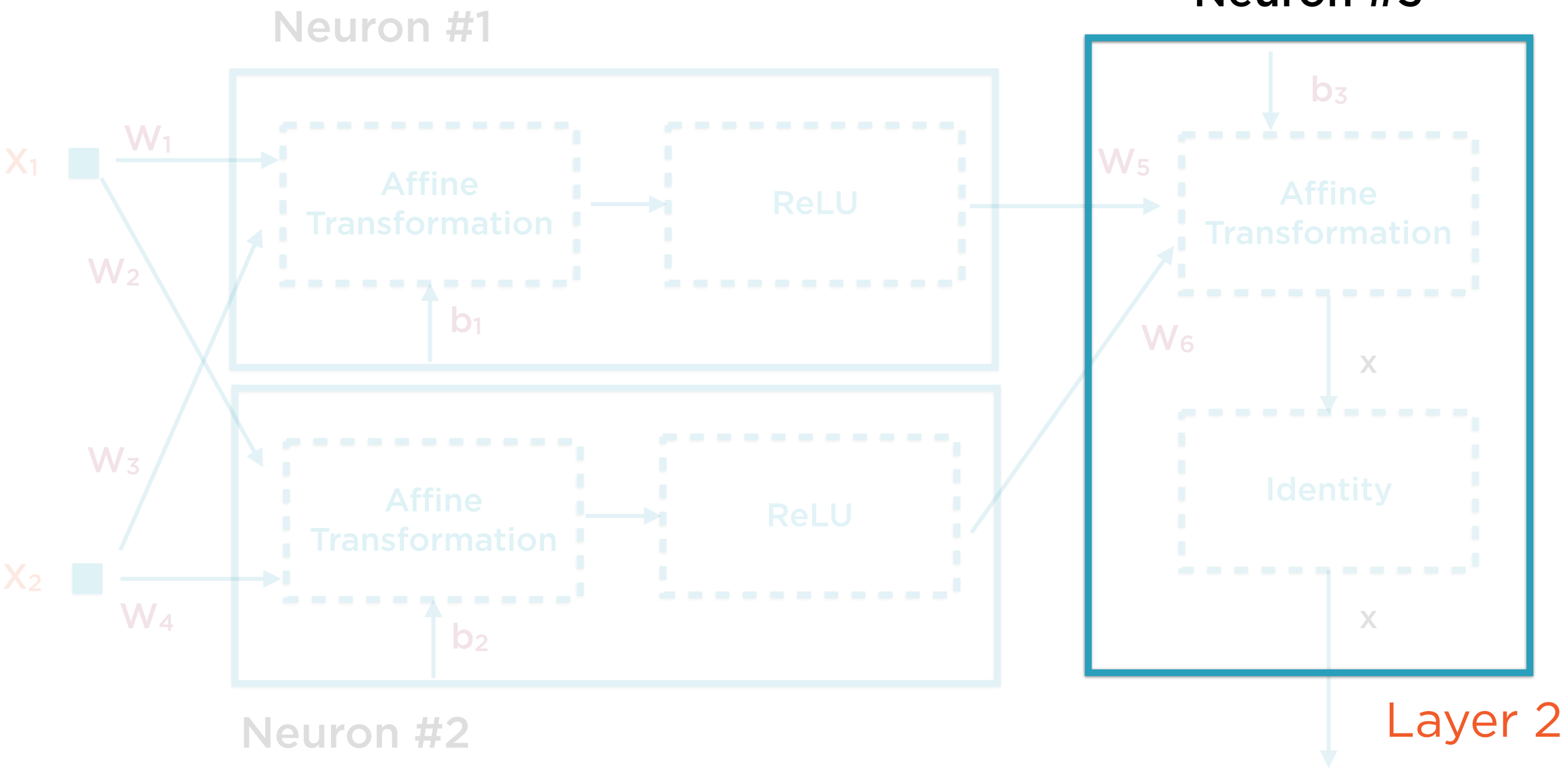
# 3-Neuron XOR


$x_1$
$x_2$
$W_1$
$W_2$
$W_3$
$W_4$
Affine Transformation
ReLU
$b_1$
Affine Transformation
ReLU
$b_2$
$b_3$
$W_5$
$W_6$
Affine Transformation
Identity
Output

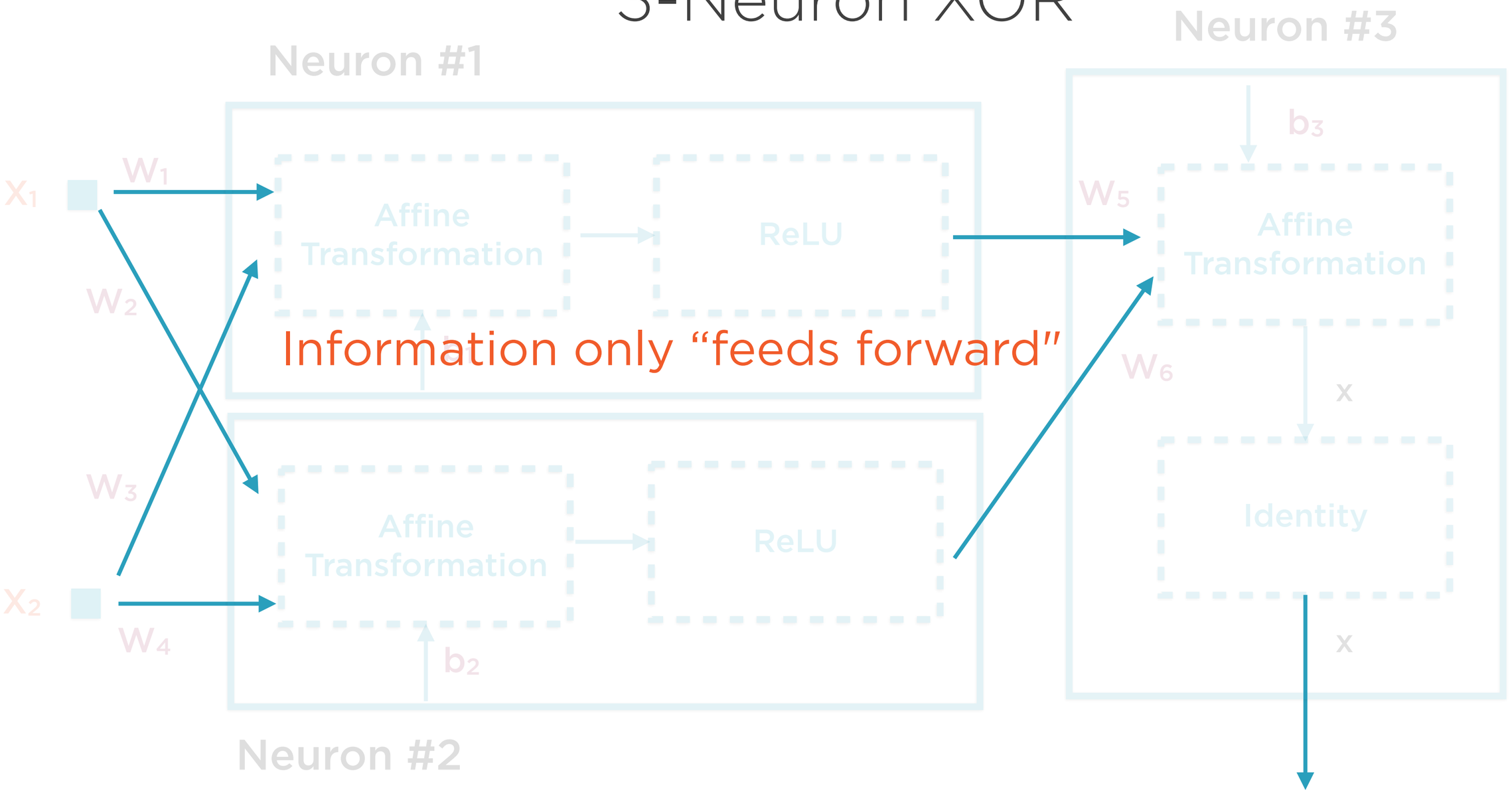
# 3-Neuron XOR


Neuron #1
Neuron #3
$b_3$
$W_5$
$X_1$
$W_1$
Affine Transformation
x
ReLU
max(x,0)
Affine Transformation
$W_2$
$b_1$
$W_6$
$W_3$
Affine Transformation
x
ReLU
max(x,0)
Activation Function
$X_2$
$W_4$
$b_2$
Neuron #2
Layer 1

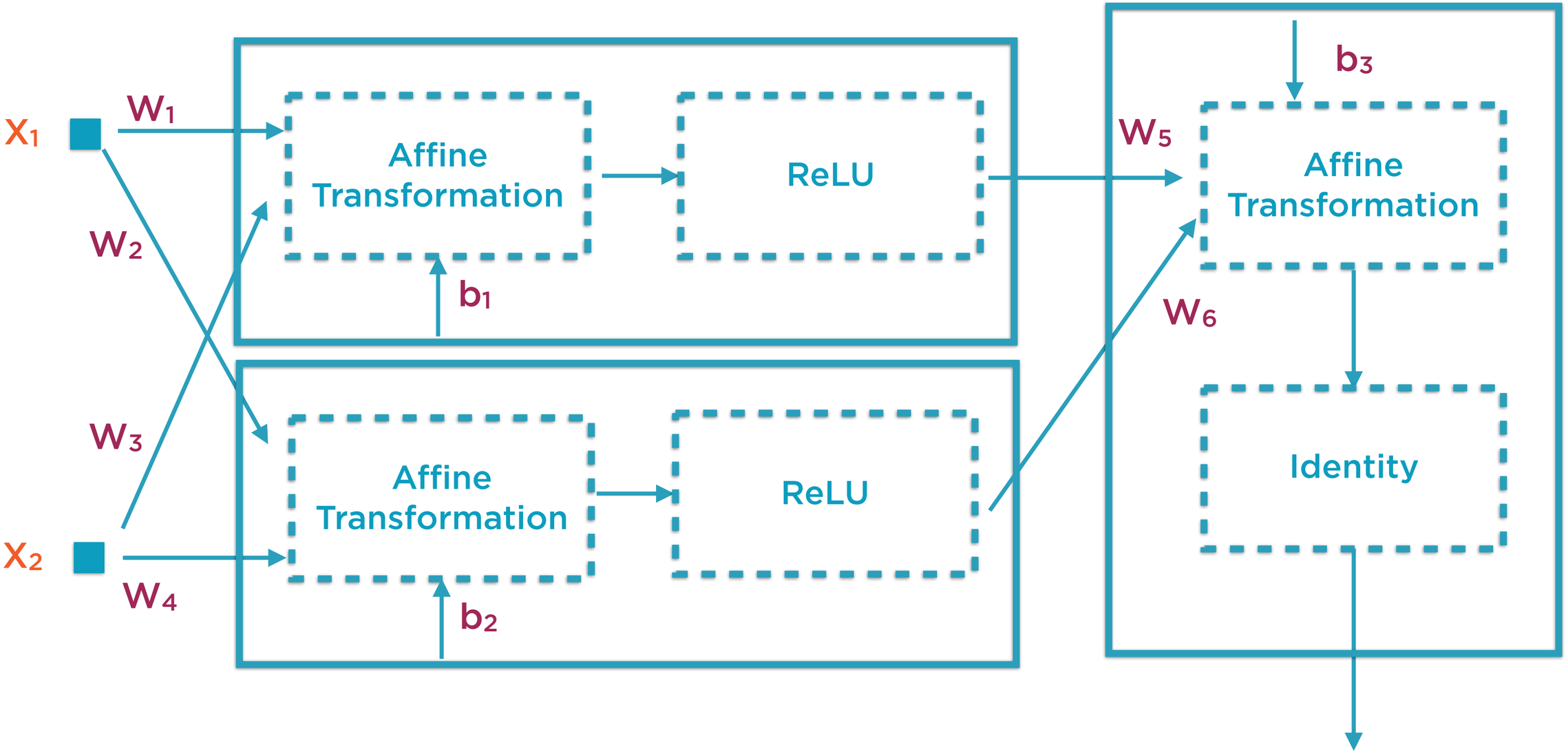
# 3-Neuron XOR


Neuron #1
Neuron #2
Neuron #3
$x_1$
$x_2$
$W_1$
$W_2$
$W_3$
$W_4$
$W_5$
$W_6$
$b_1$
$b_2$
$b_3$
Affine Transformation
ReLU
Affine Transformation
ReLU
Affine Transformation
x
Identity
x
Layer 2

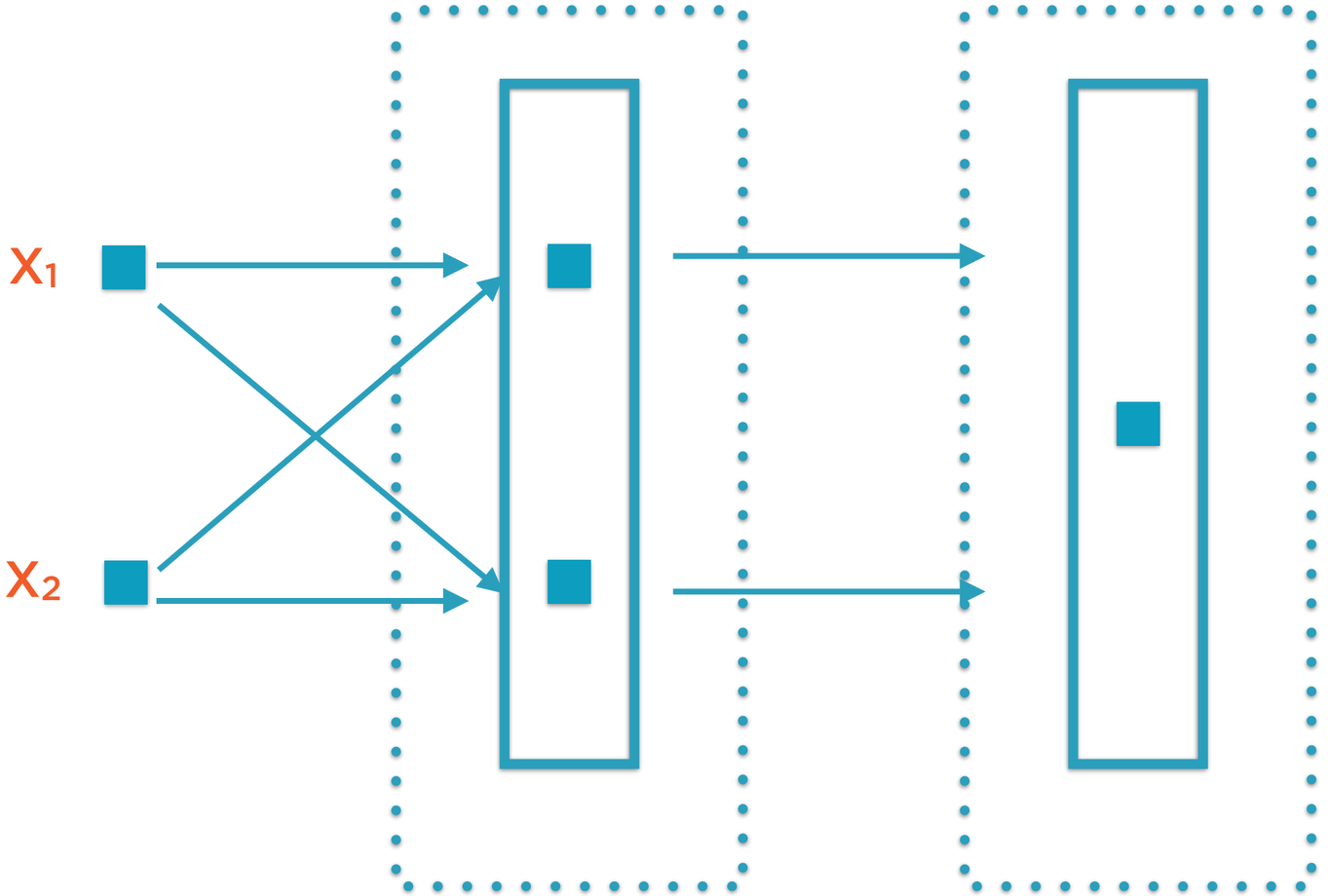
# 3-Neuron XOR


Neuron #1
Neuron #3
$b_3$
$W_1$
$X_1$
Affine Transformation
ReLU
$W_5$
Affine Transformation
$W_2$
Information only “feeds forward"
$W_6$
x
$W_3$
Affine Transformation
ReLU
Identity
$X_2$
x
$W_4$
$b_2$
Neuron #2

# “2-Layer Feed-forward Neural Network”



# XOR: 3 Neurons, 2 Layers




| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

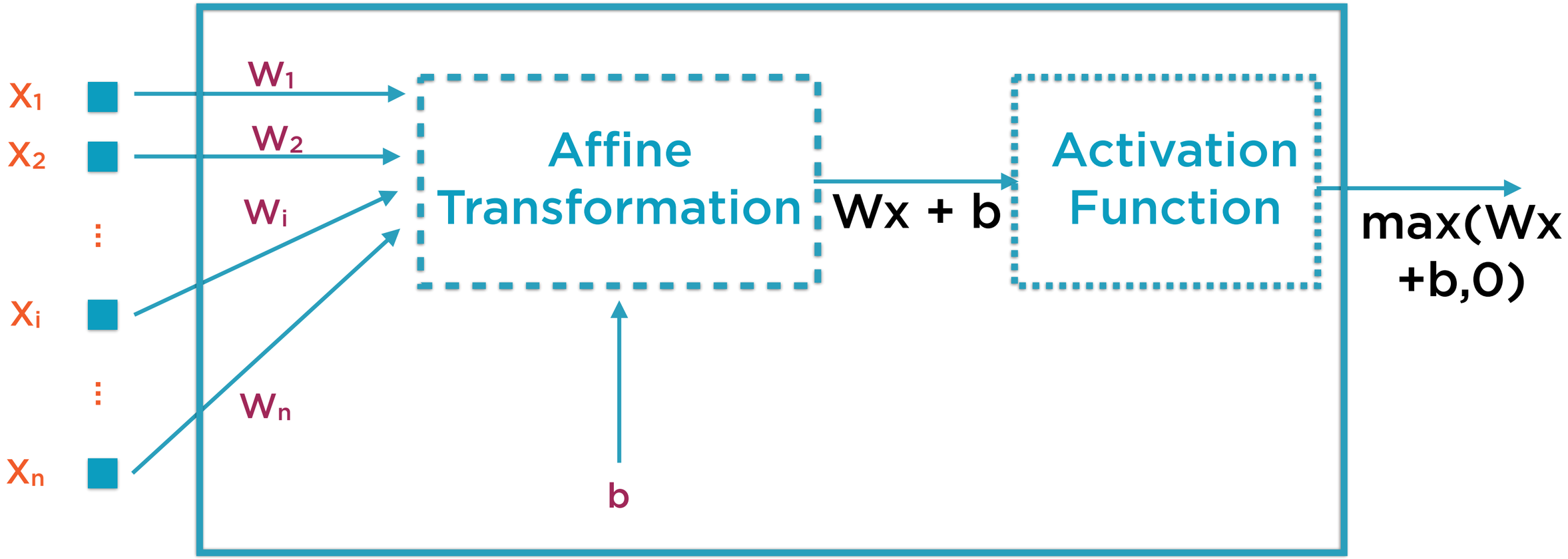
2 Inputs

Layer 1

Layer 2

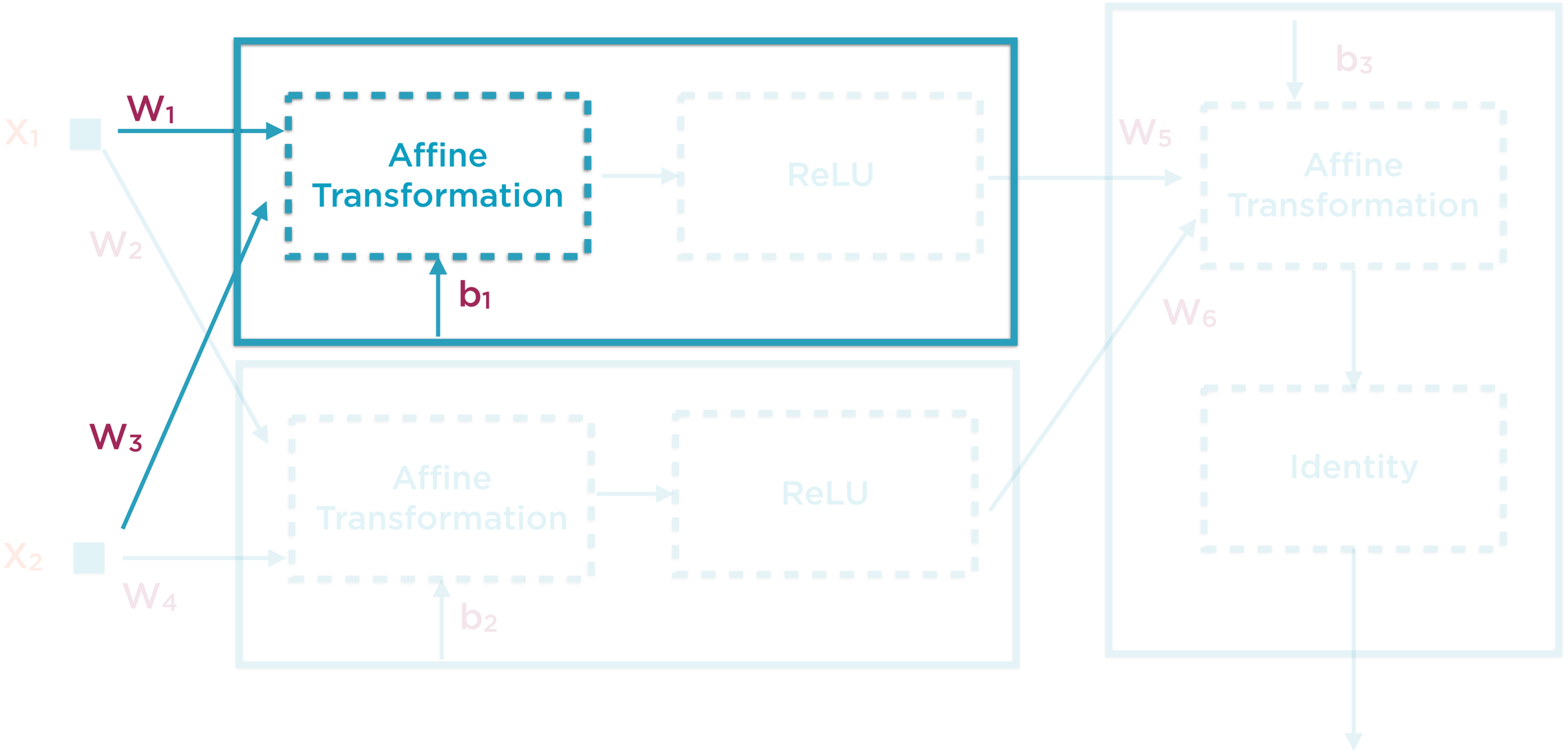
Output Truth Table

# Operation of a Single Neuron

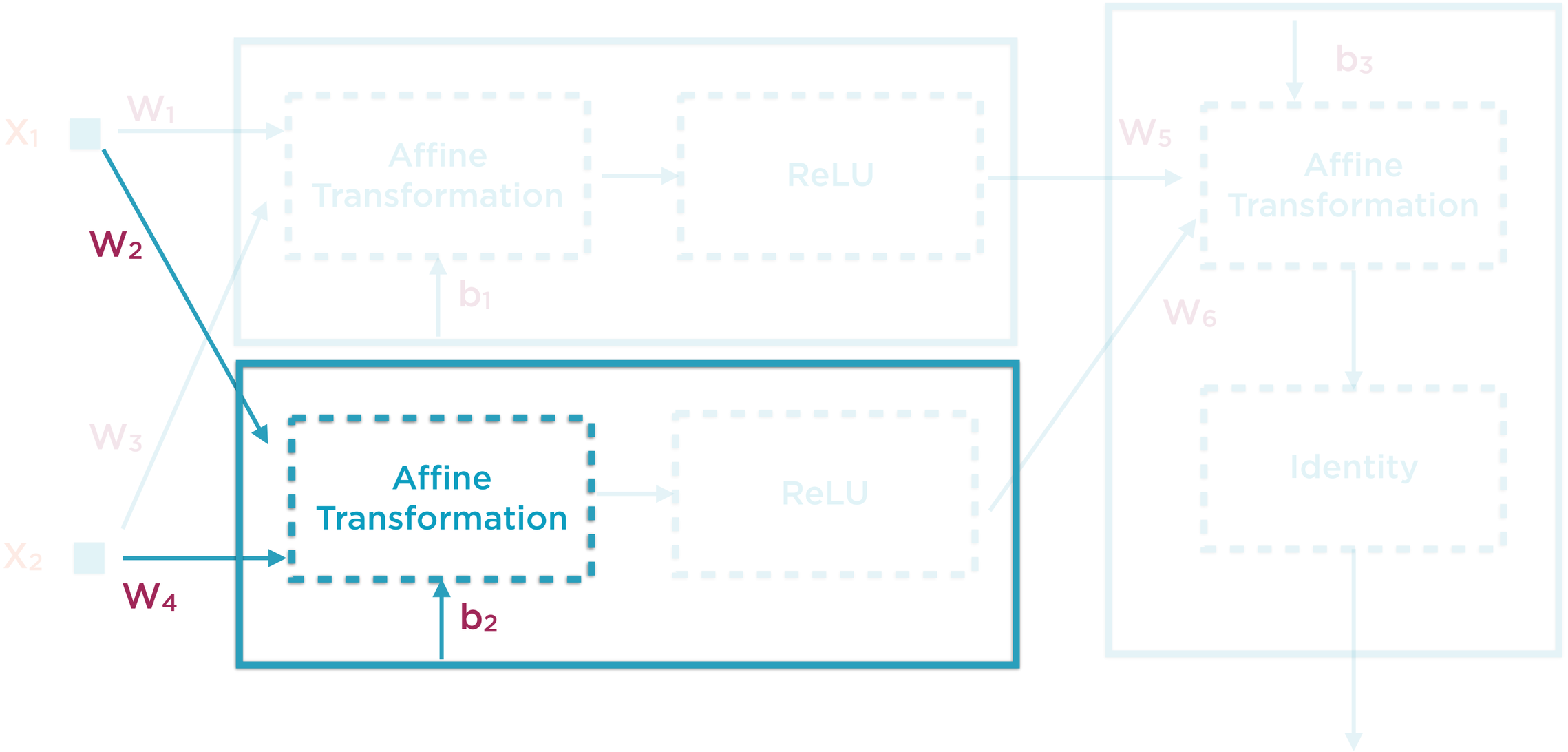



**Each neuron has weights and a bias that must be calculated by the training algorithm (done for us by TensorFlow)**

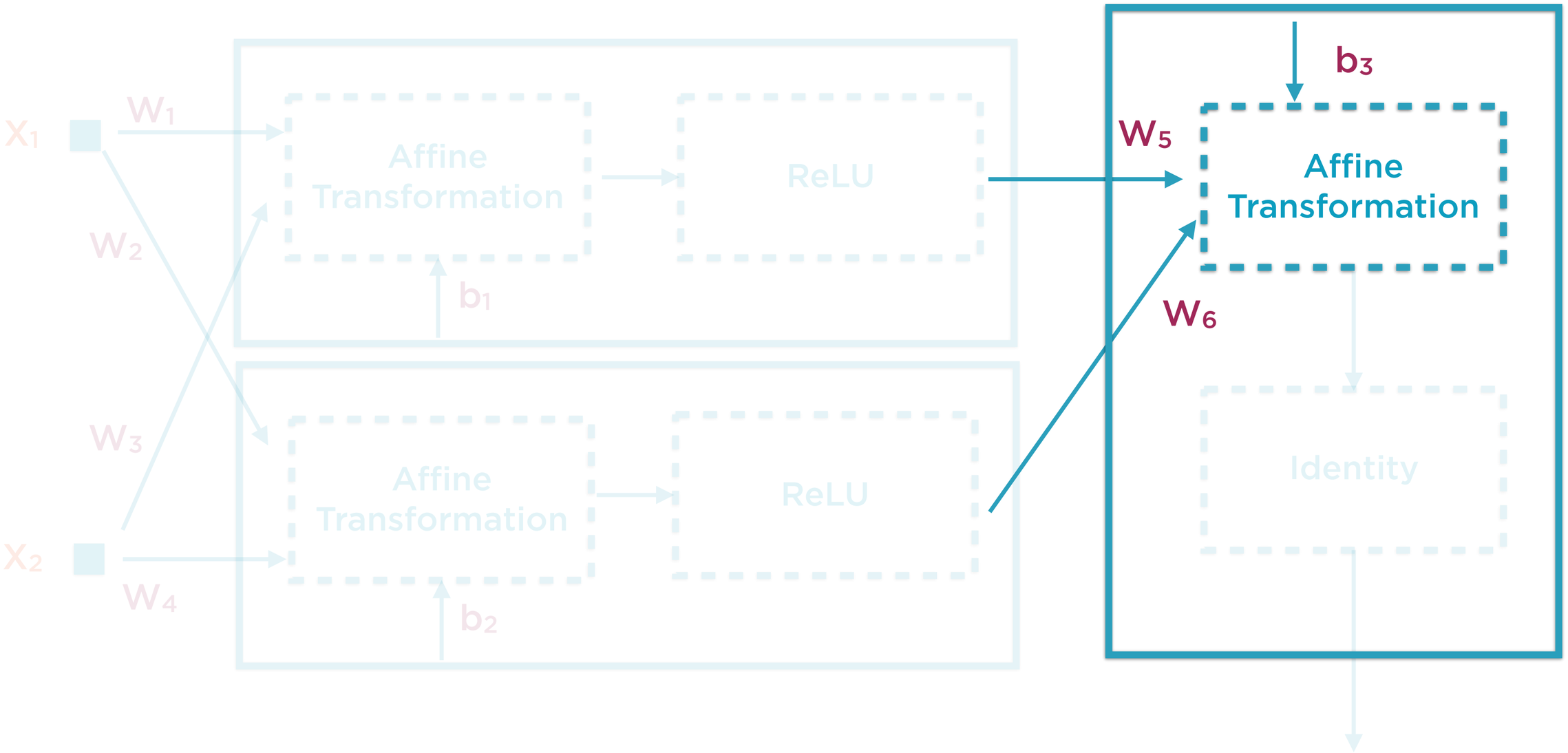
# Weights and Bias of Neuron #1



# Weights and Bias of Neuron #2

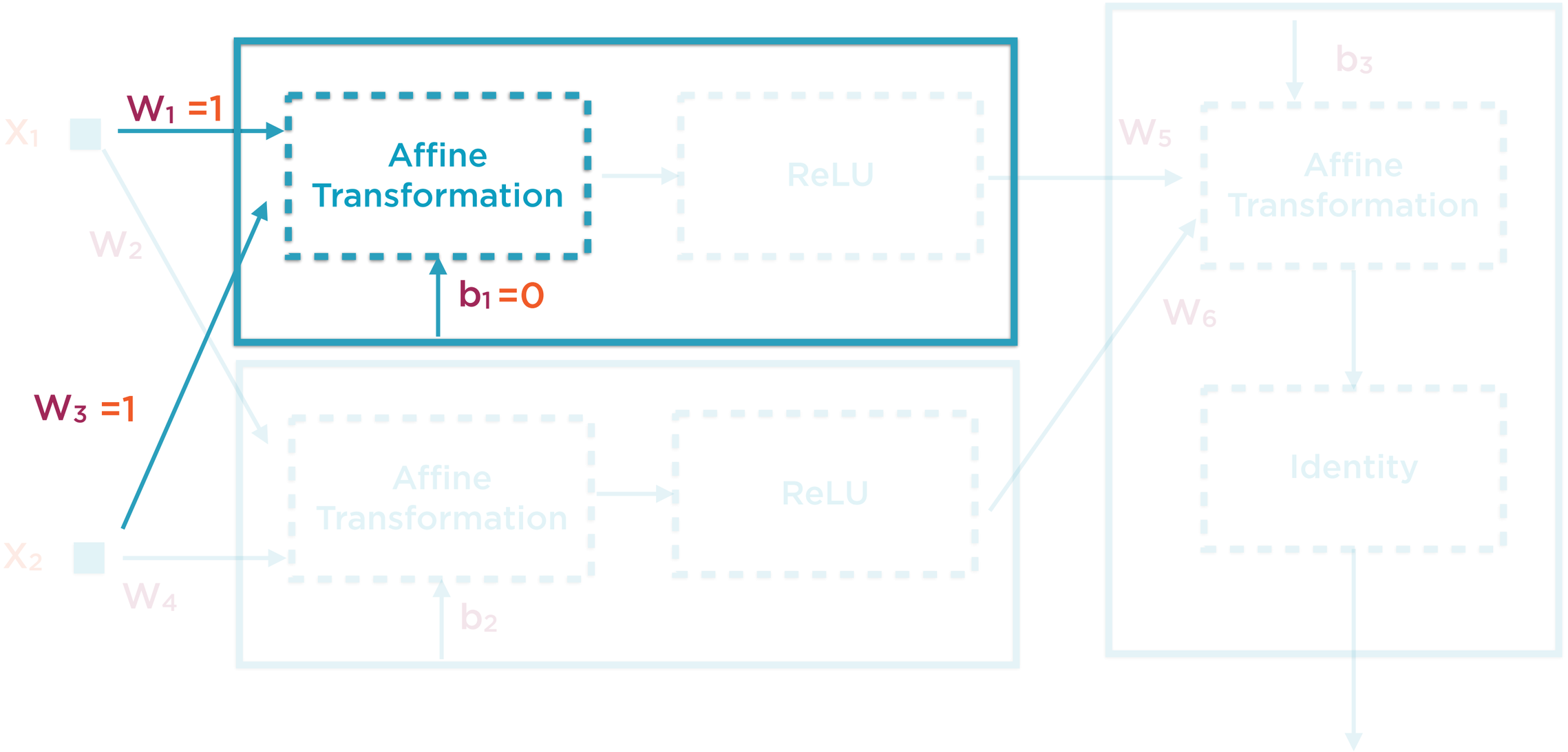


# Weights and Bias of Neuron #3

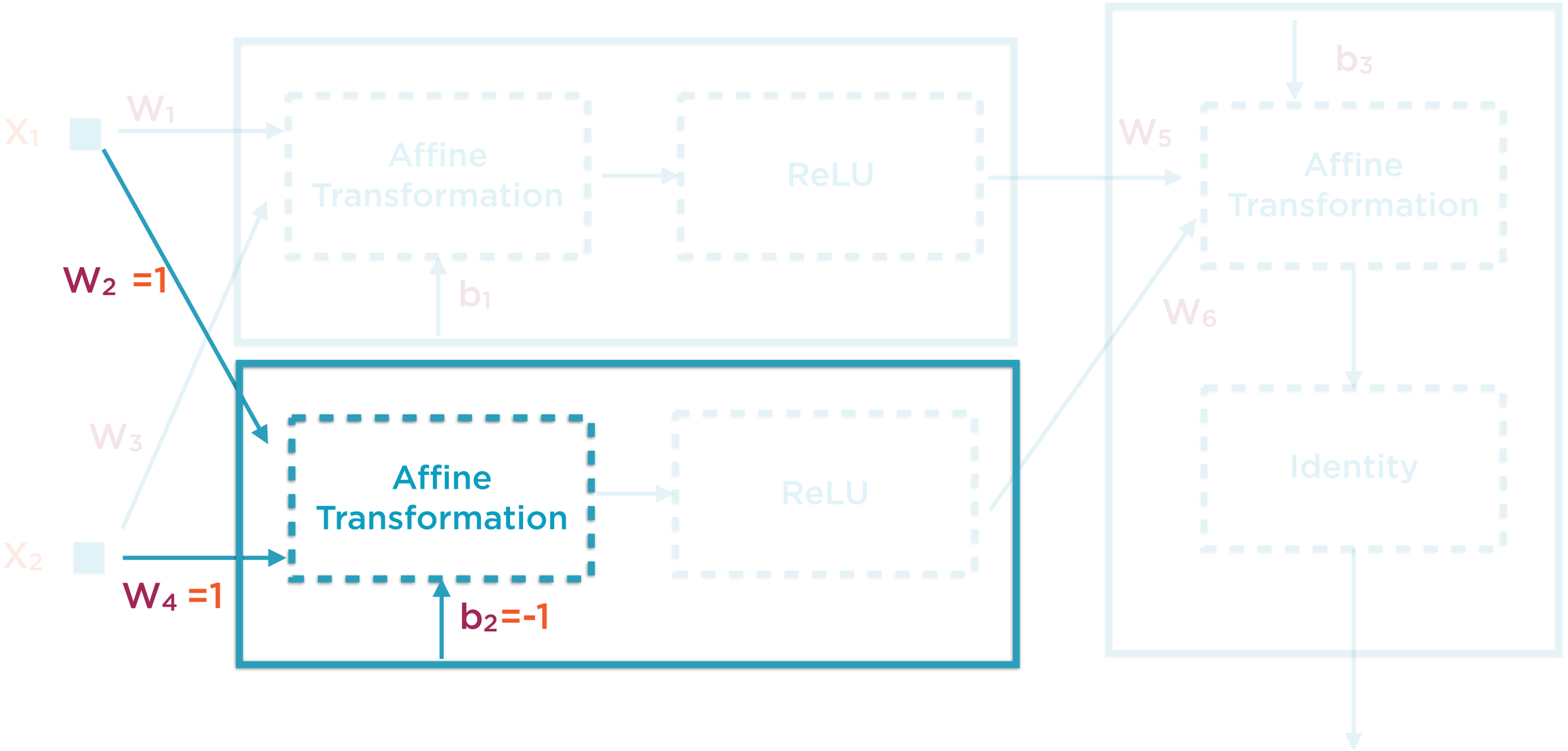


The weights and biases of individual neurons are determined during the training process

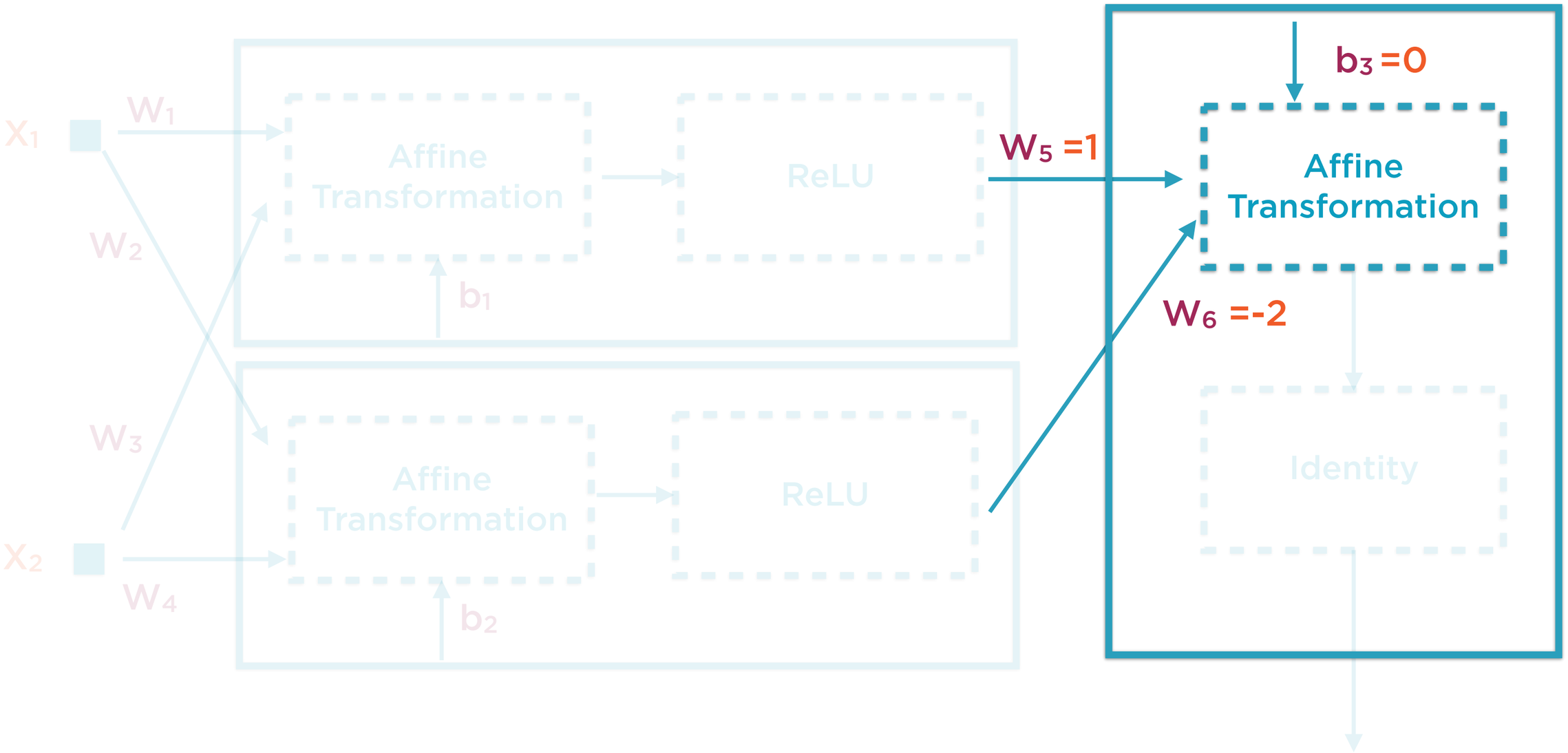
# Weights and Bias of Neuron #1



# Weights and Bias of Neuron #2



# Weights and Bias of Neuron #3

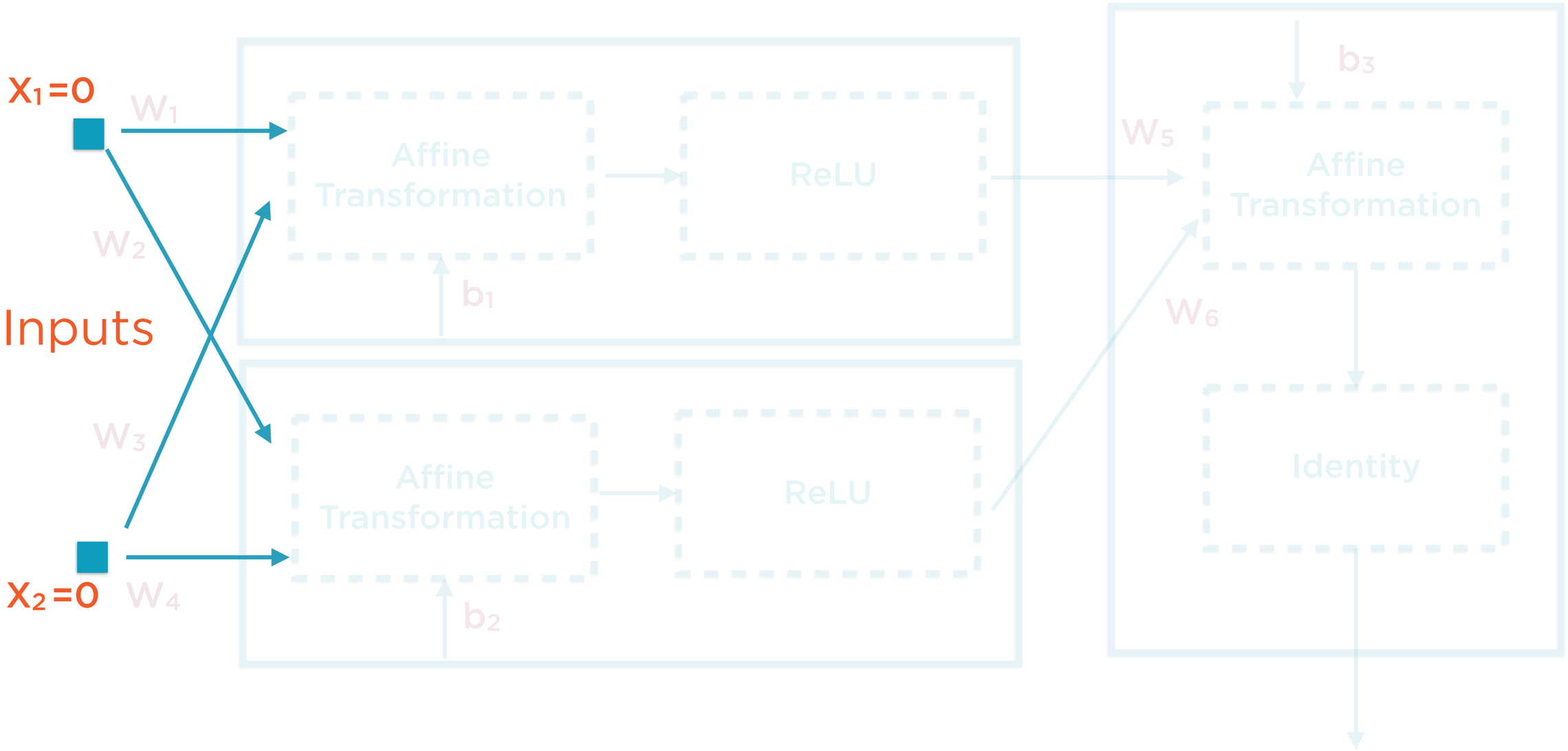


| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

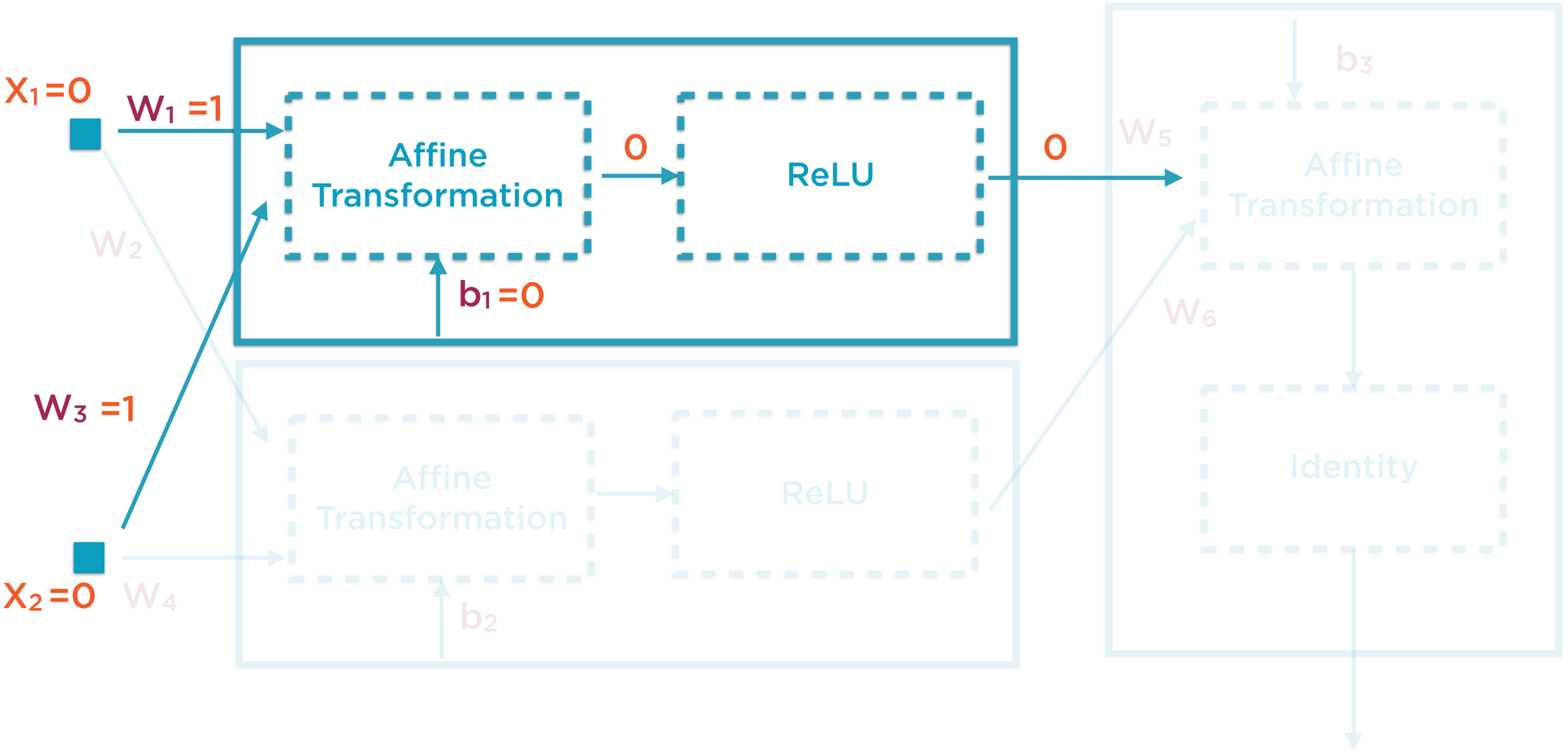
# “Learning” XOR

**Reverse-engineering XOR requires 3 neurons (arranged in 2 layers) as well as a non-linear activation function**

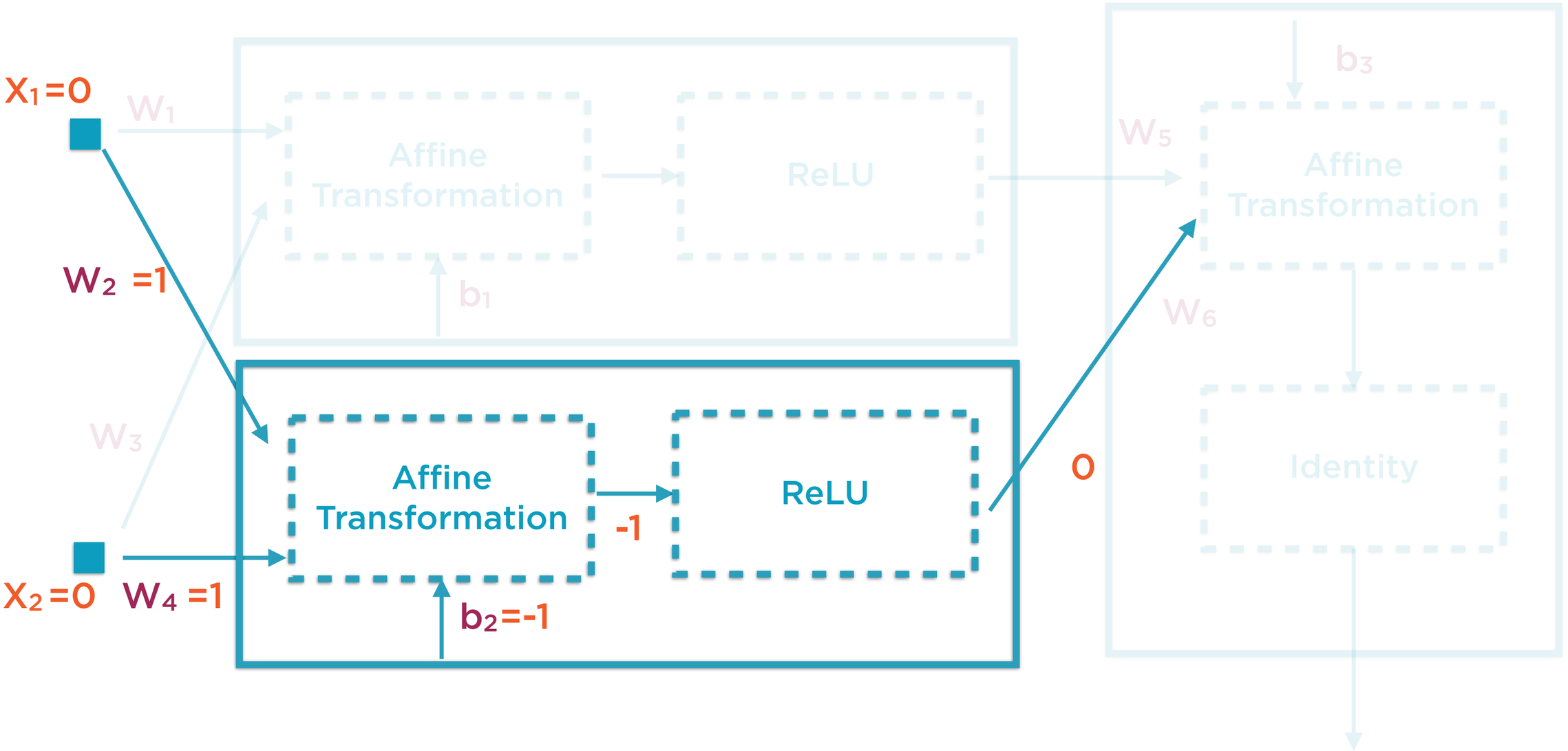
# 3-Neuron XOR


$x_1=0$
$W_1$
$W_2$
Inputs
$W_3$
$x_2=0$
$W_4$
Affine Transformation
ReLU
$b_1$
Affine Transformation
ReLU
$b_2$
$b_3$
$W_5$
Affine Transformation
$W_6$
Identity

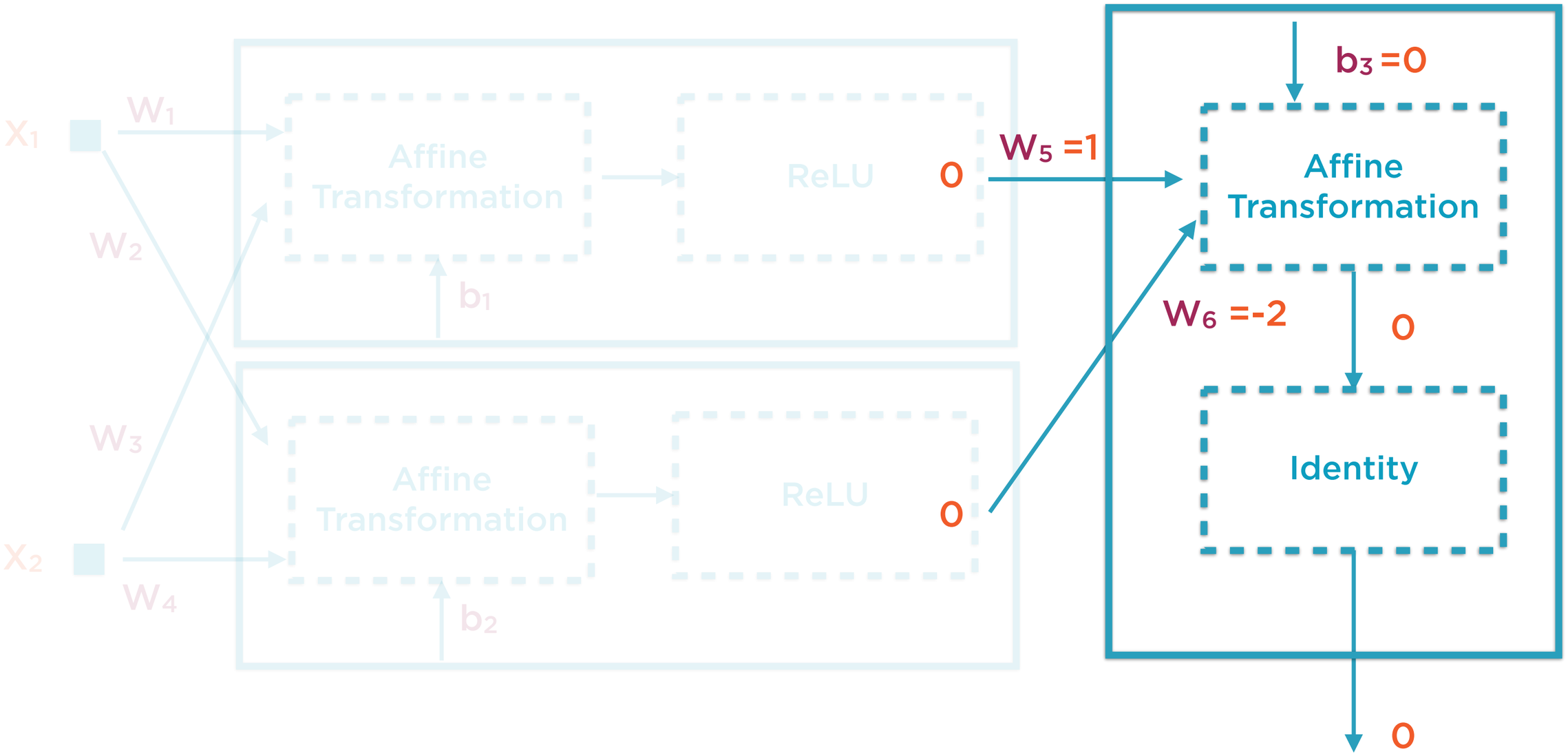
# Weights and Bias of Neuron #1



# Weights and Bias of Neuron #2



# Weights and Bias of Neuron #3

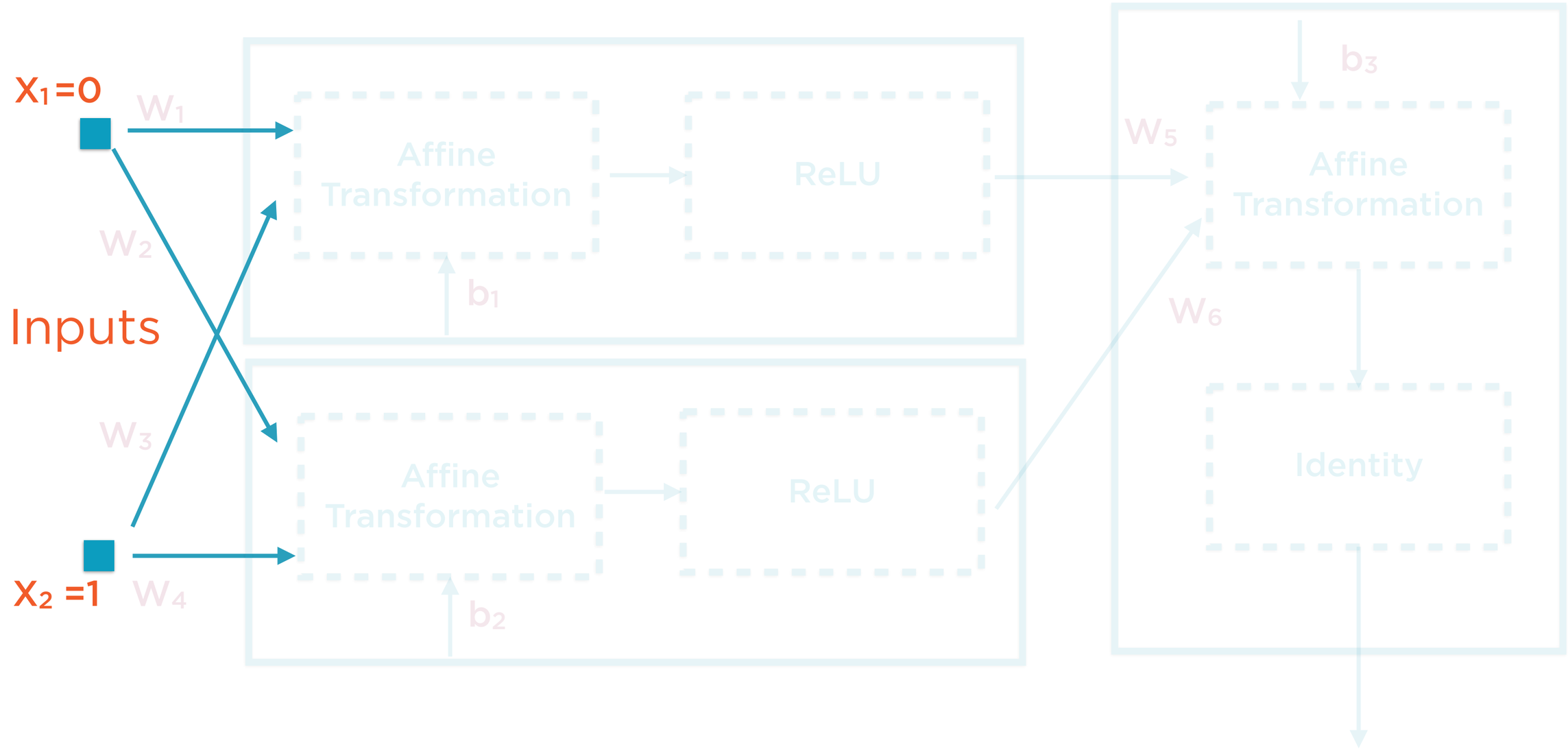


| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

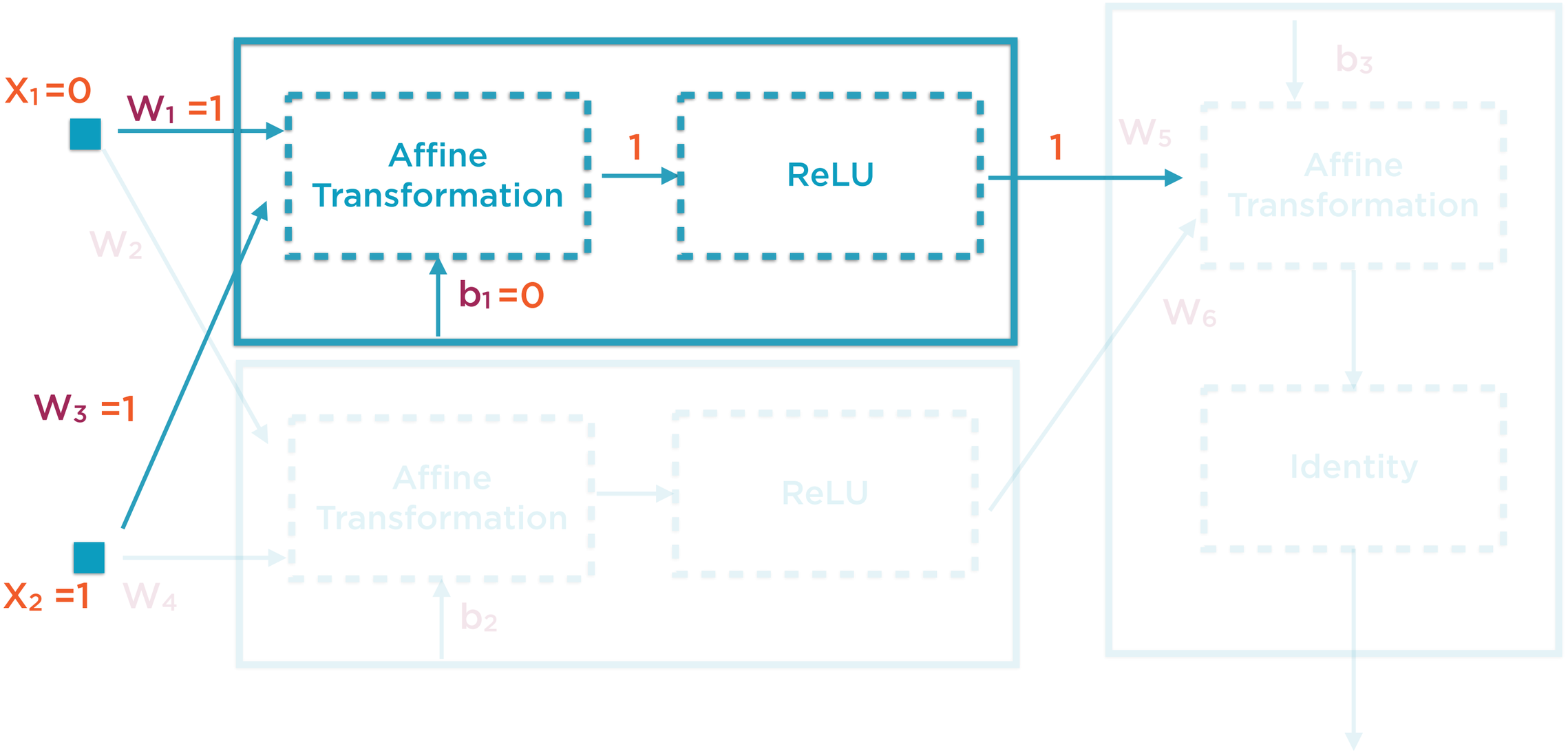
# “Learning” XOR

**Reverse-engineering XOR requires 3 neurons (arranged in 2 layers) as well as a non-linear activation function**

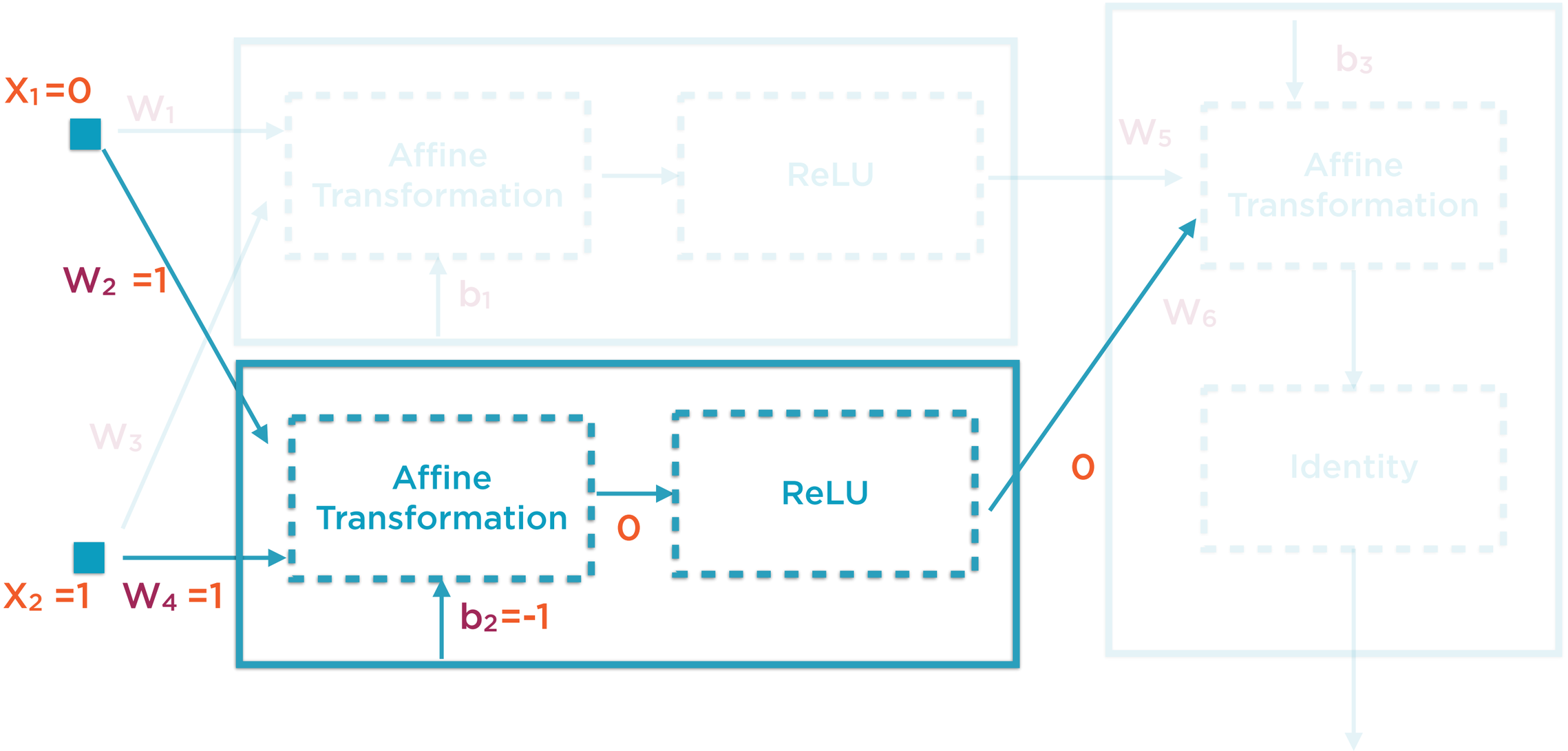
# 3-Neuron XOR


$x_1$=0
$W_1$
$W_2$
Inputs
$W_3$
$x_2$ =1
$W_4$
Affine Transformation
ReLU
$b_1$
Affine Transformation
ReLU
$b_2$
$b_3$
$W_5$
Affine Transformation
$W_6$
Identity

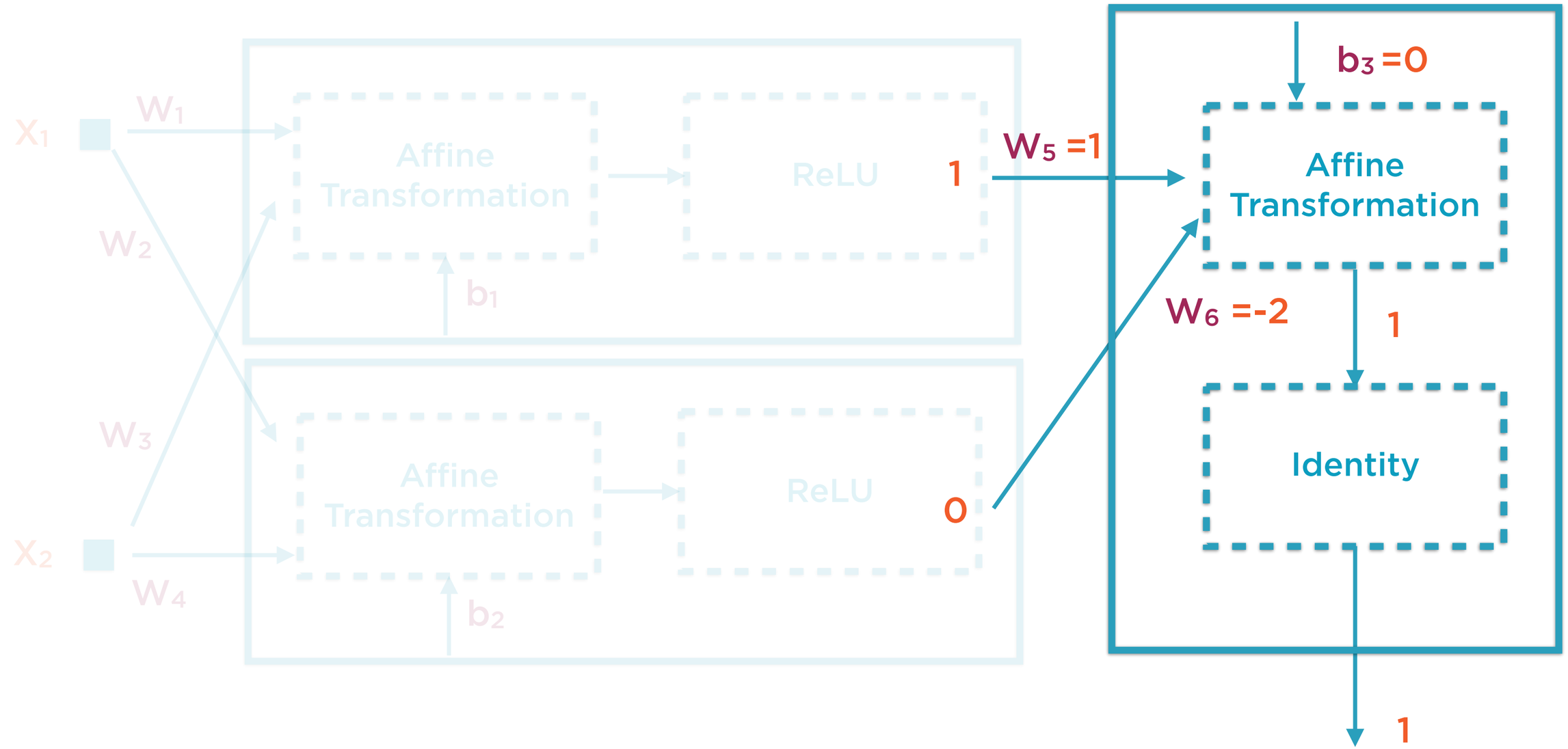
# Weights and Bias of Neuron #1



# Weights and Bias of Neuron #2



# Weights and Bias of Neuron #3

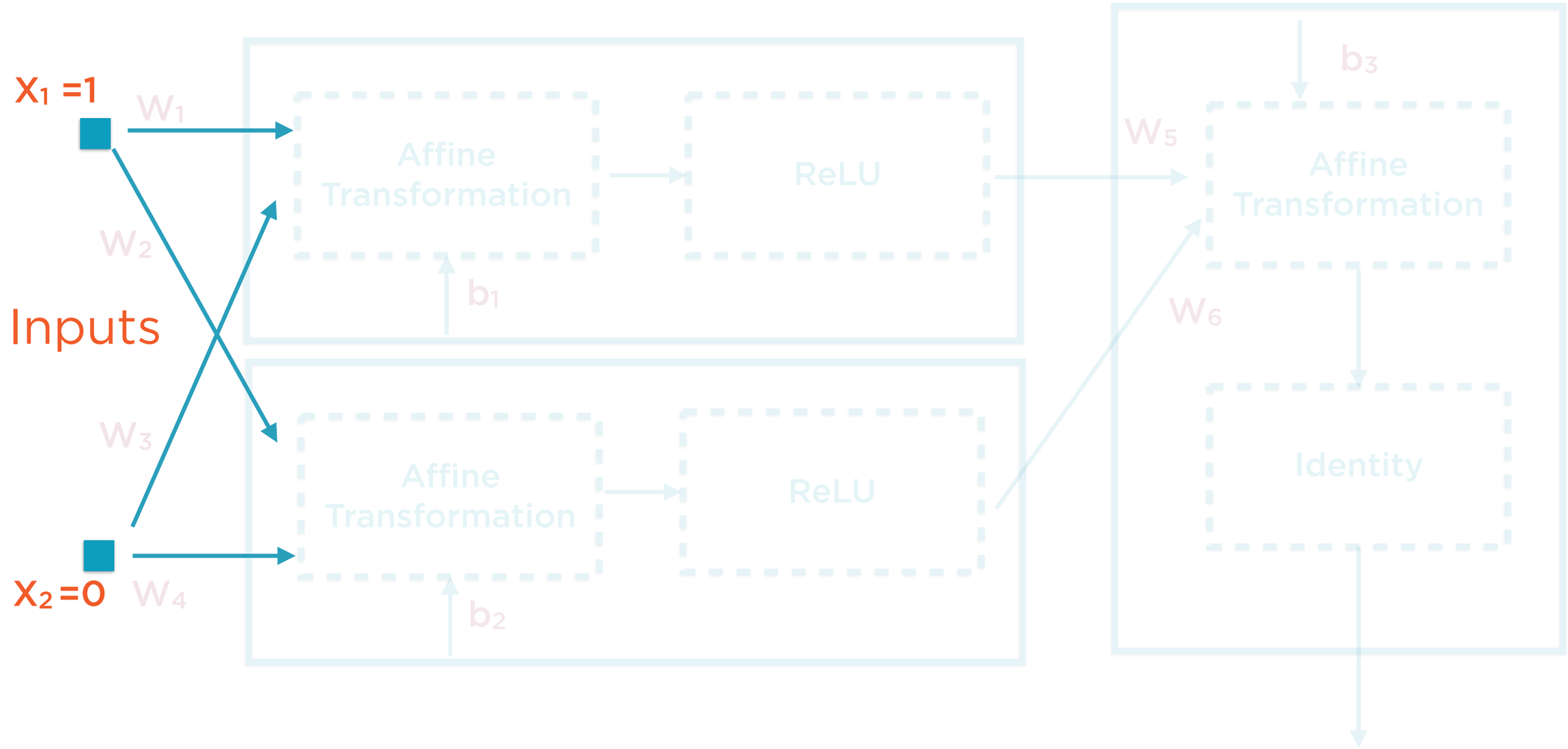


| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

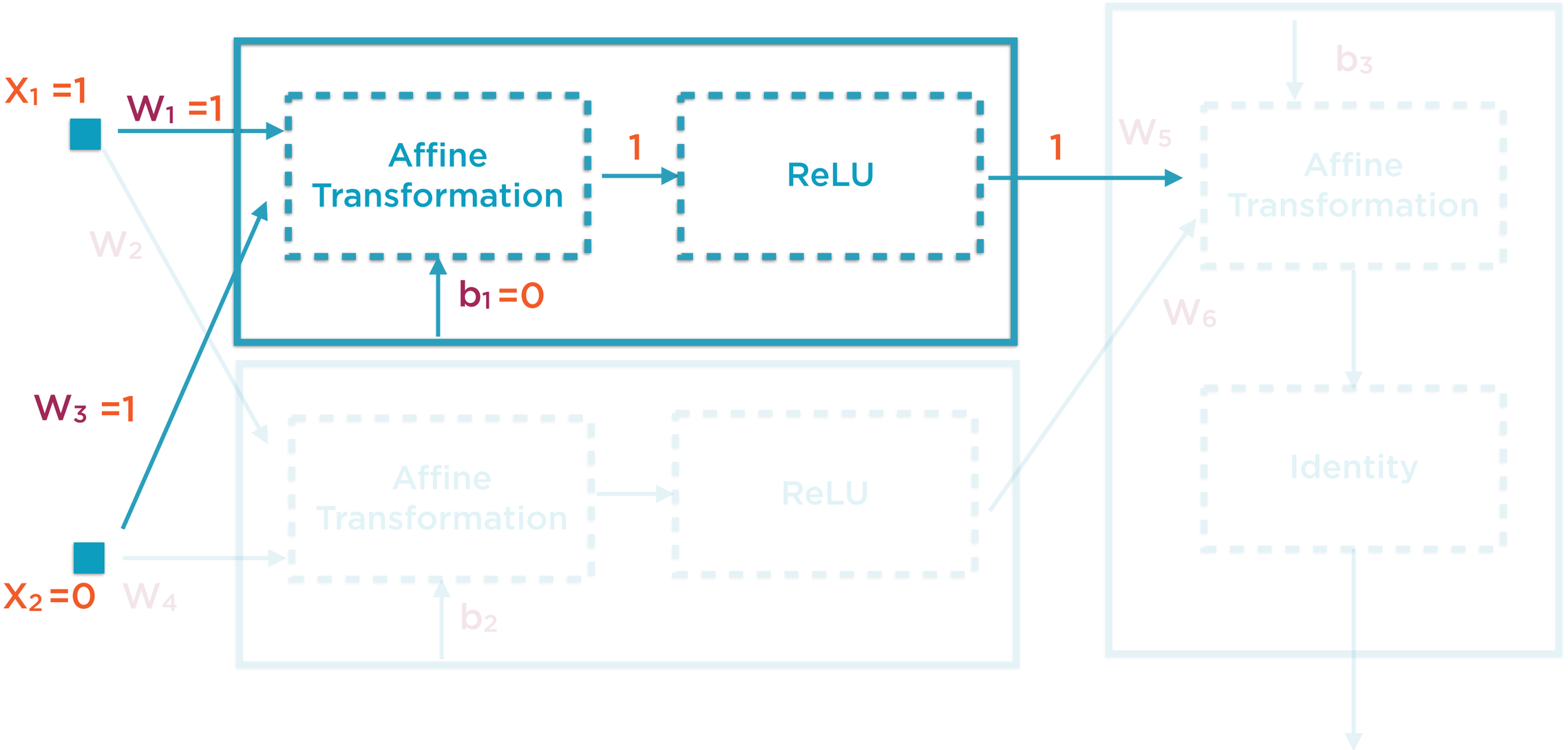
# “Learning” XOR

**Reverse-engineering XOR requires 3 neurons (arranged in 2 layers) as well as a non-linear activation function**

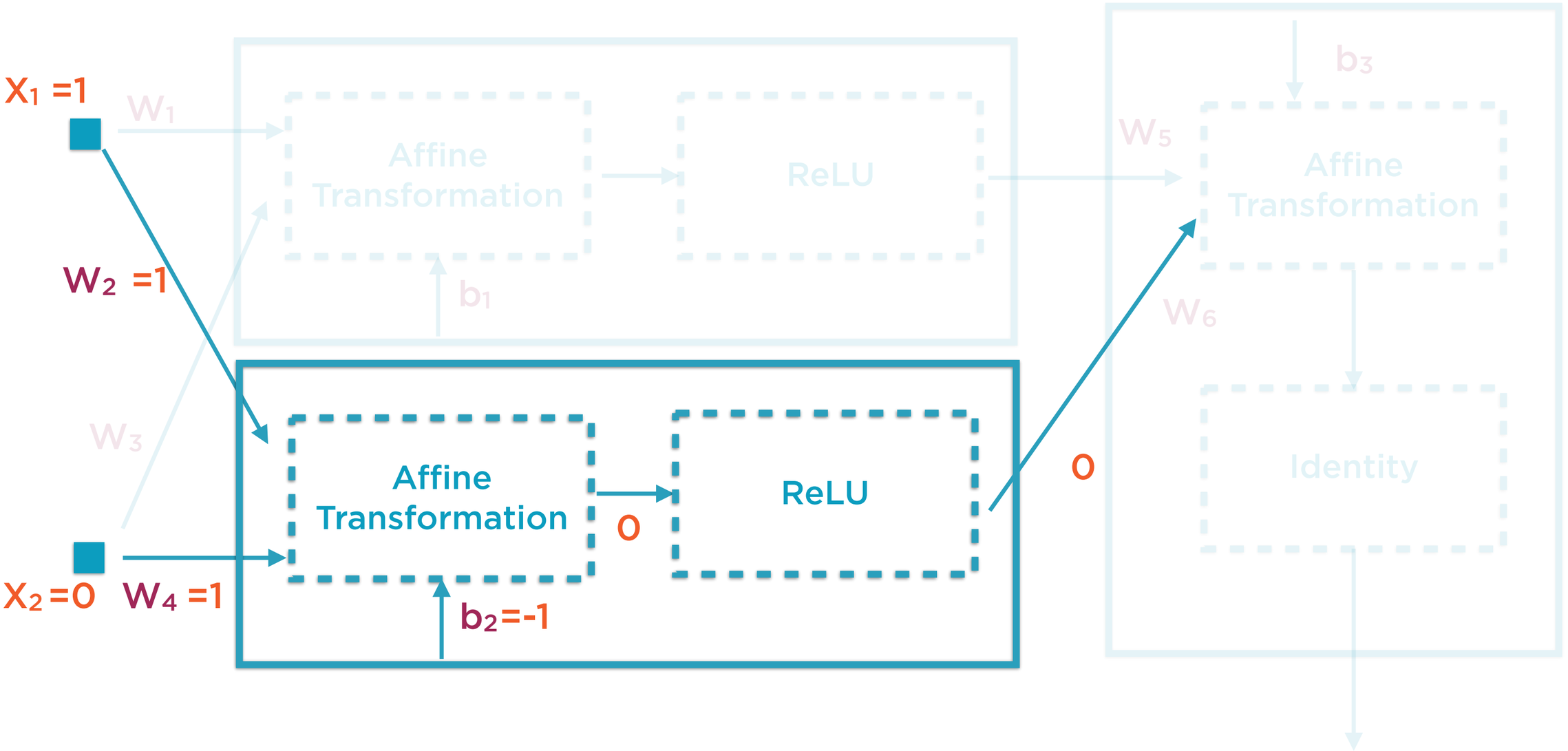
# 3-Neuron XOR


$x_1 = 1$
$W_1$
$W_2$
Inputs
$W_3$
$x_2 = 0$
$W_4$
Affine Transformation
ReLU
$b_1$
Affine Transformation
ReLU
$b_2$
$b_3$
$W_5$
Affine Transformation
$W_6$
Identity

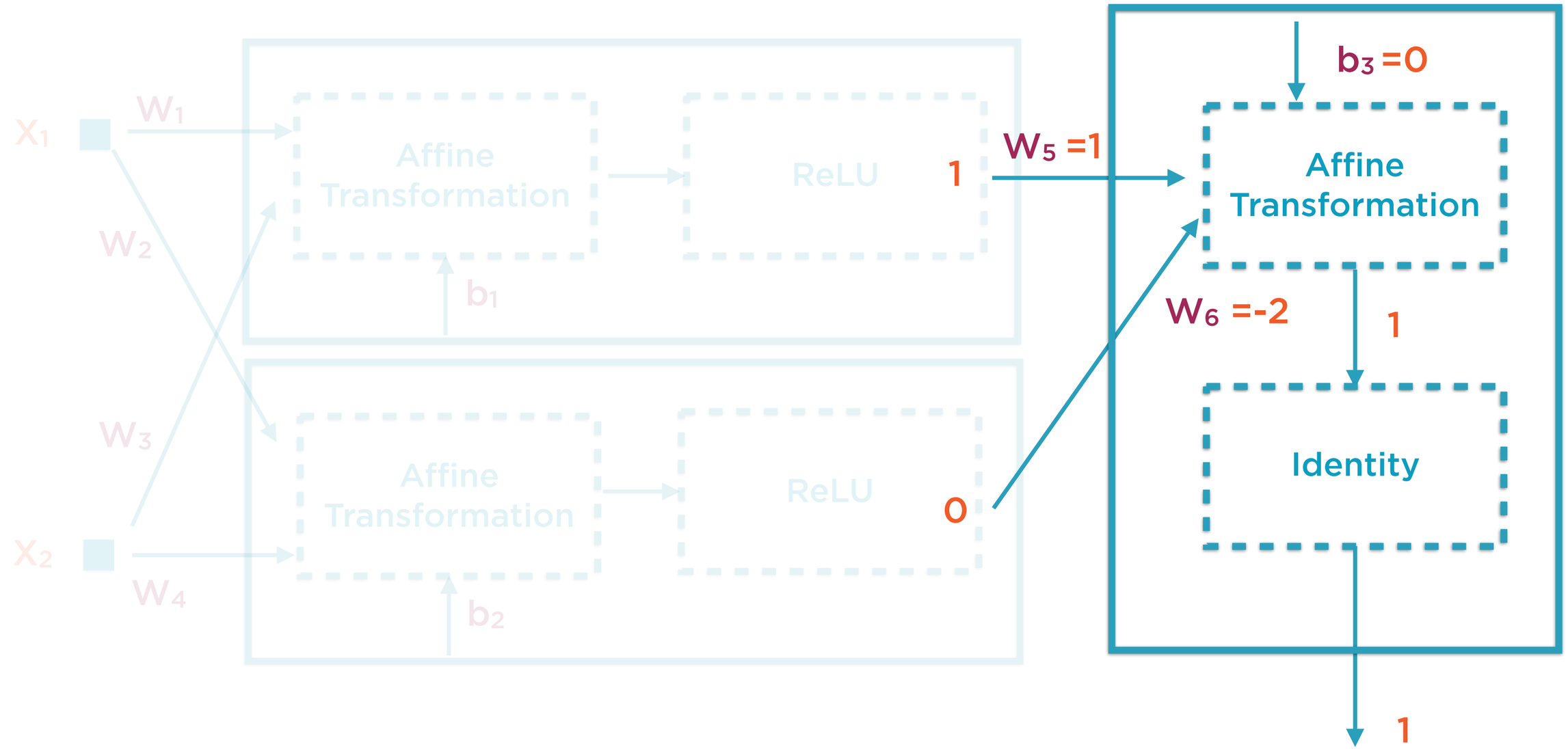
# Weights and Bias of Neuron #1



# Weights and Bias of Neuron #2



# Weights and Bias of Neuron #3

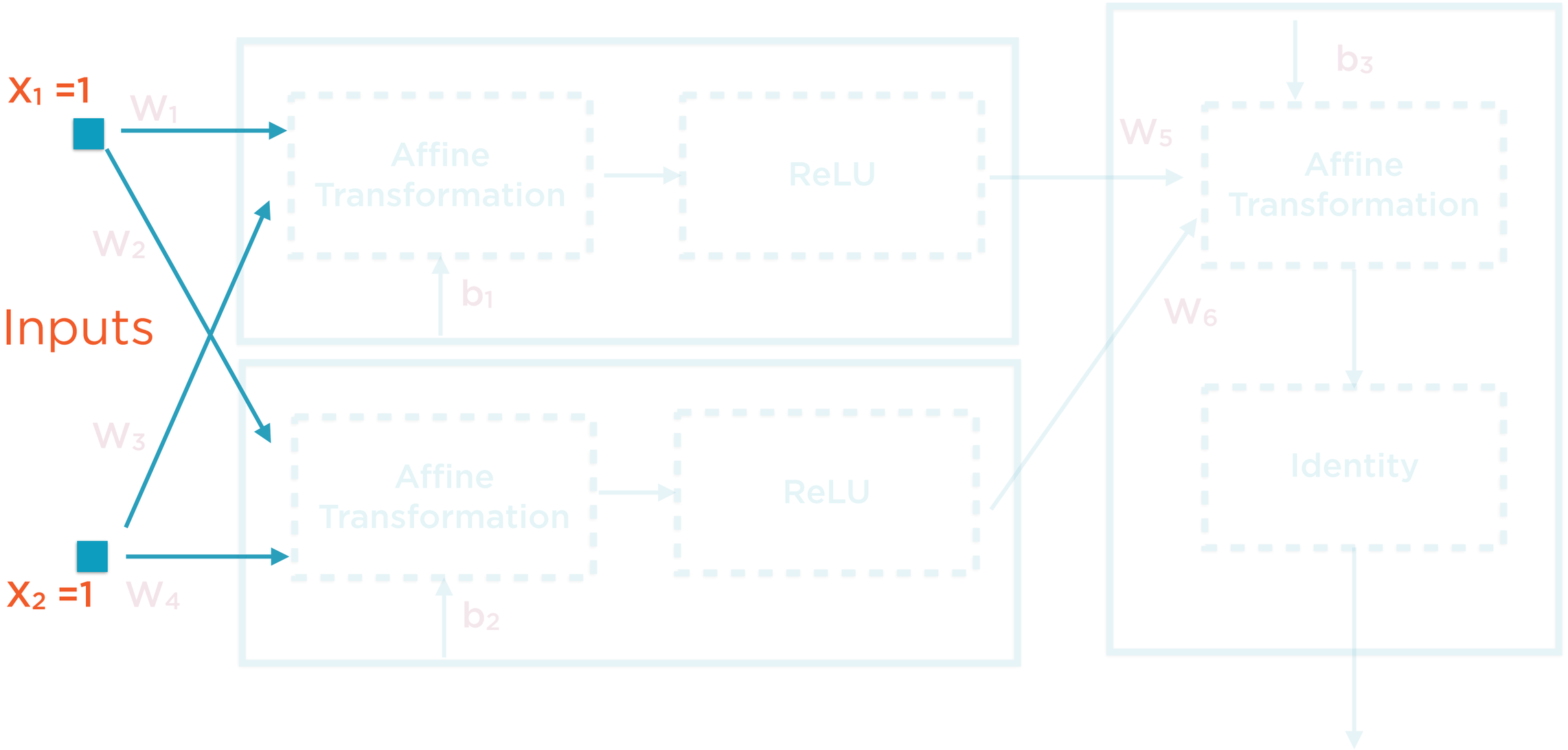


| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

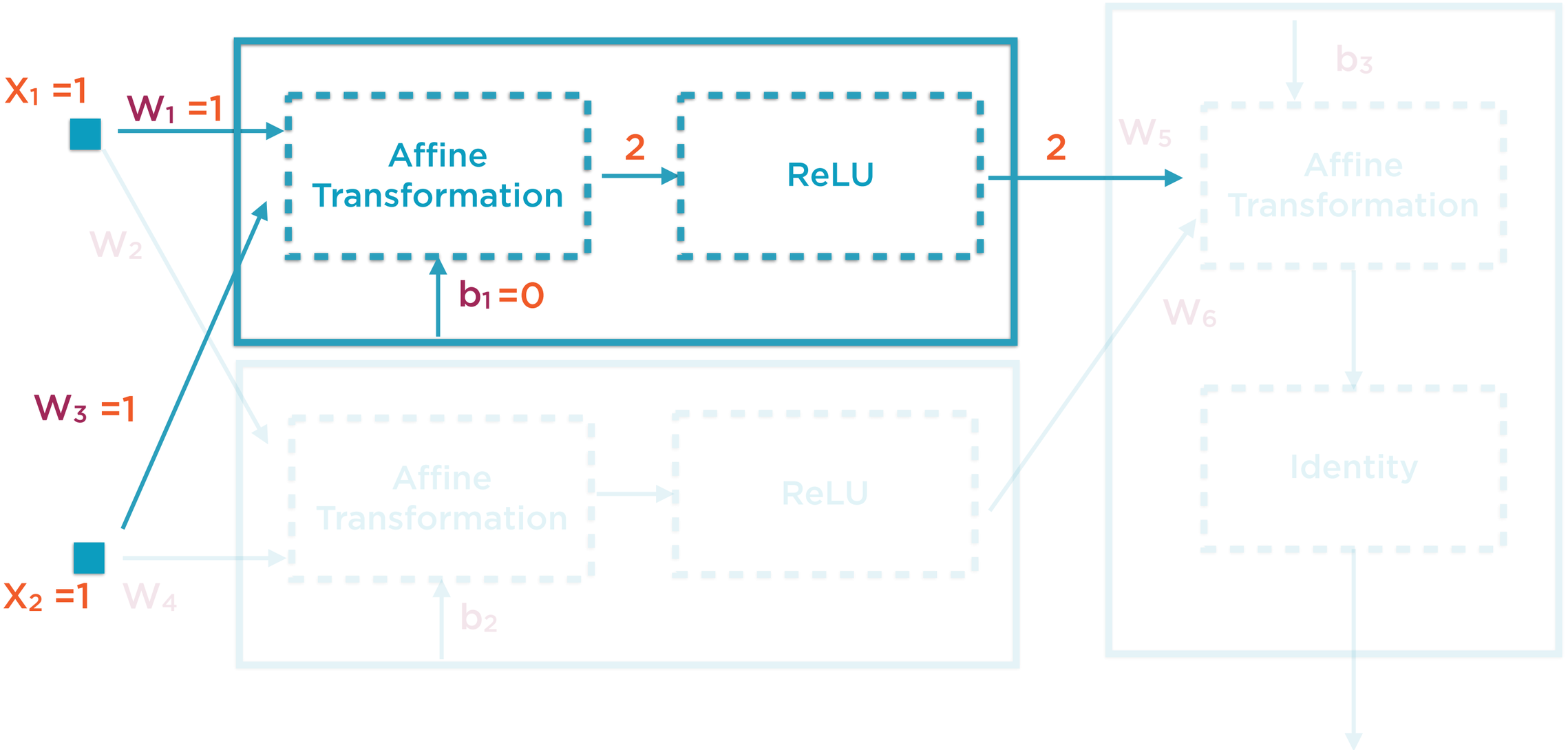
# “Learning” XOR

**Reverse-engineering XOR requires 3 neurons (arranged in 2 layers) as well as a non-linear activation function**

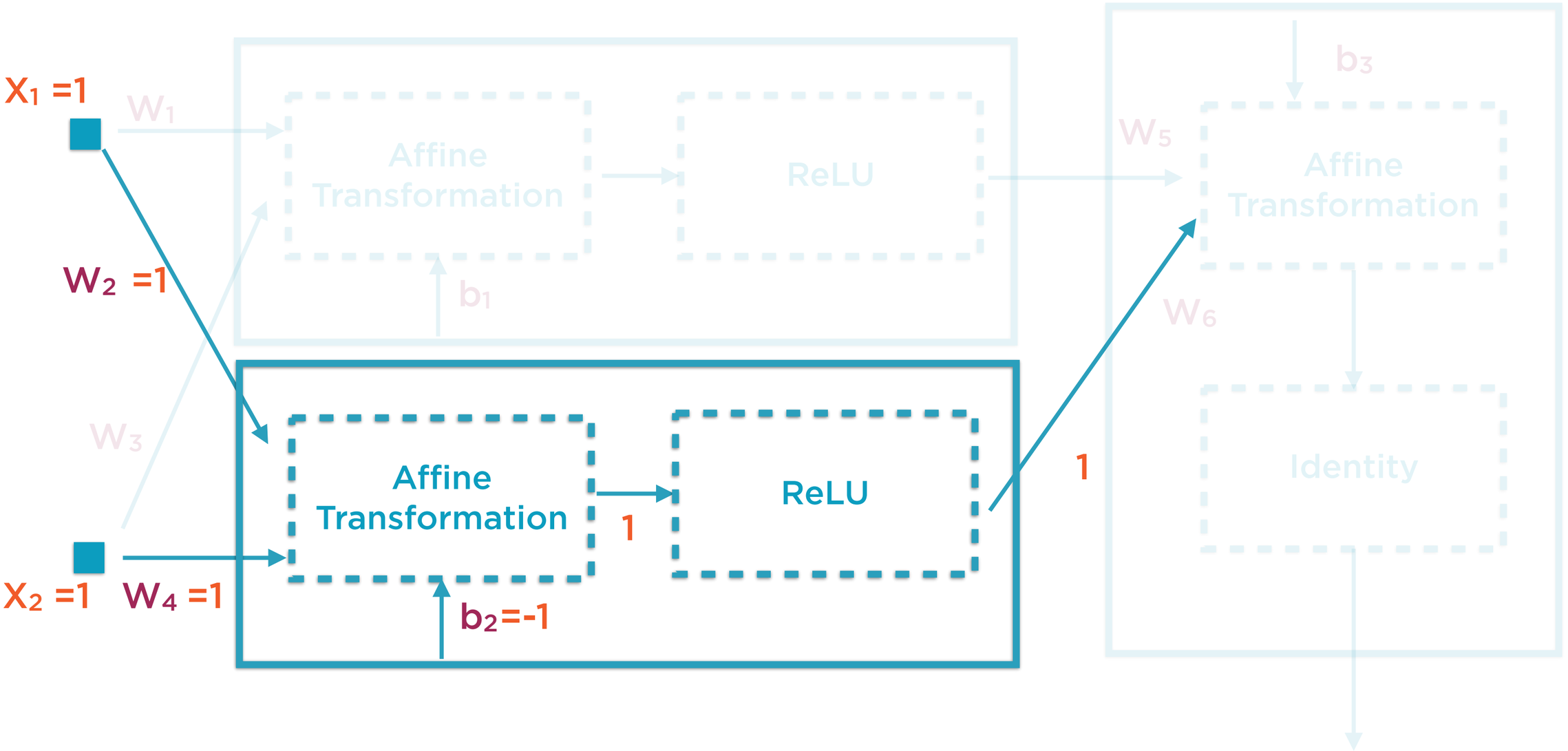
# 3-Neuron XOR


$x_1 = 1$
$W_1$
$W_2$
Inputs
$W_3$
$x_2 = 1$
$W_4$
Affine Transformation
ReLU
$b_1$
Affine Transformation
ReLU
$b_2$
$b_3$
$W_5$
Affine Transformation
$W_6$
Identity

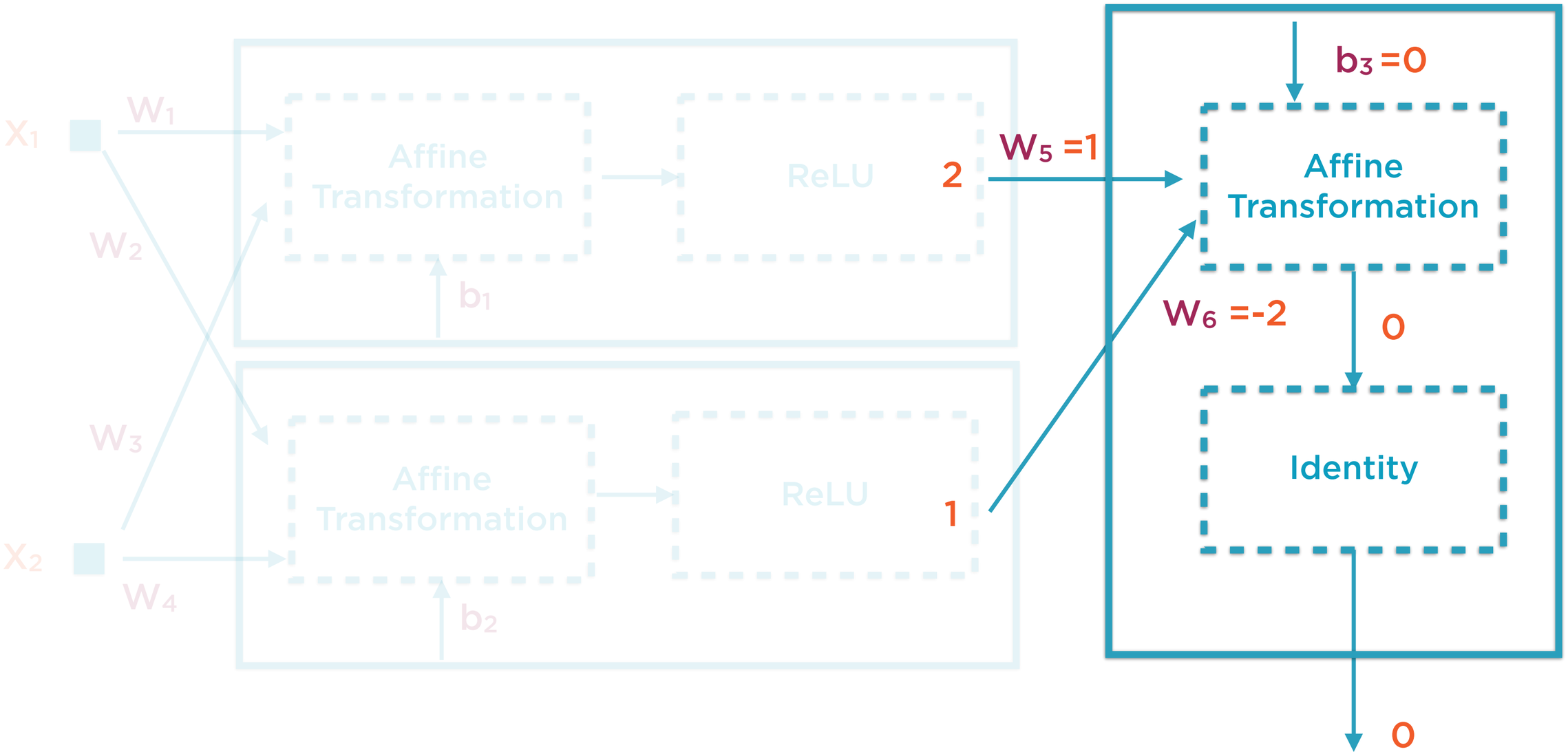
# Weights and Bias of Neuron #1



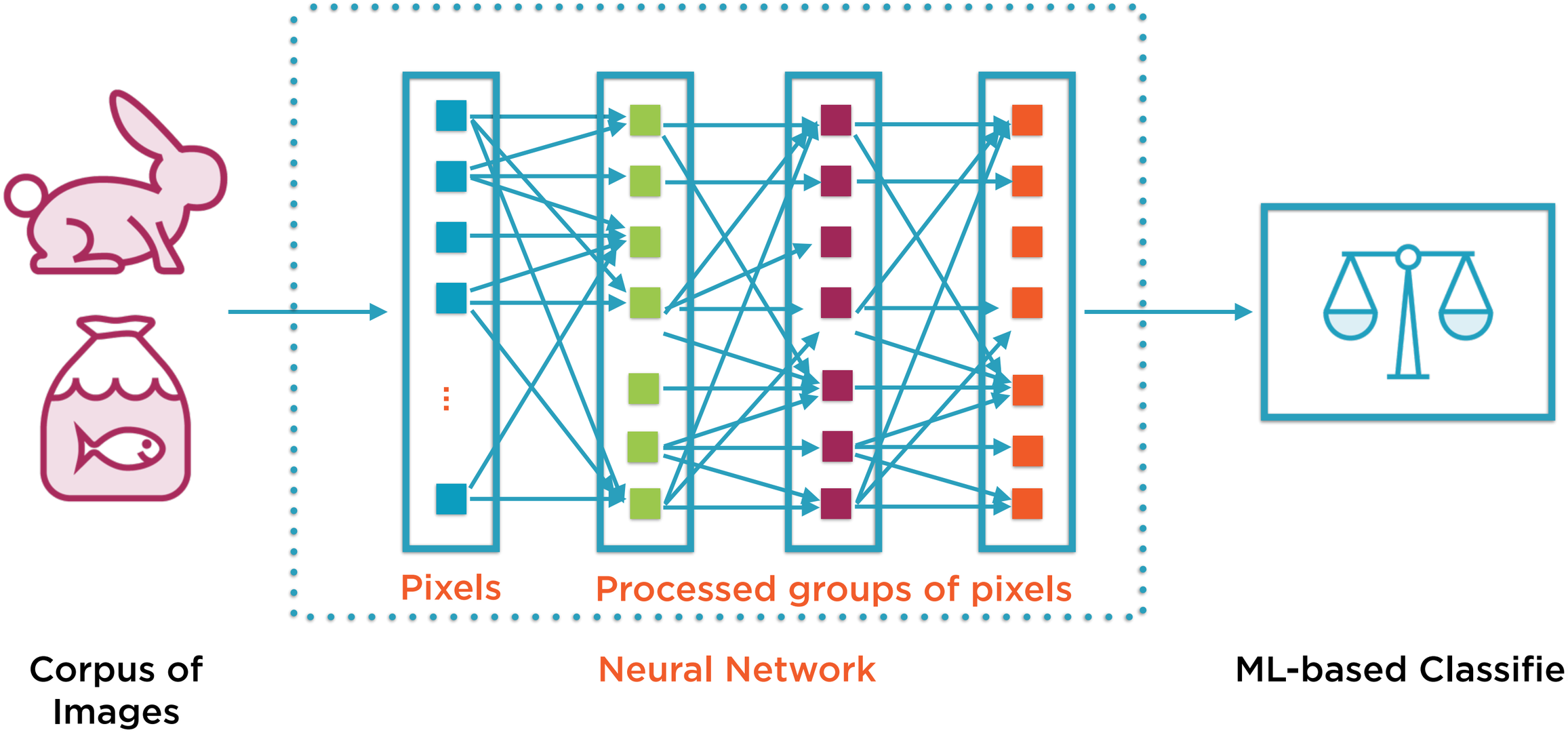
# Weights and Bias of Neuron #2



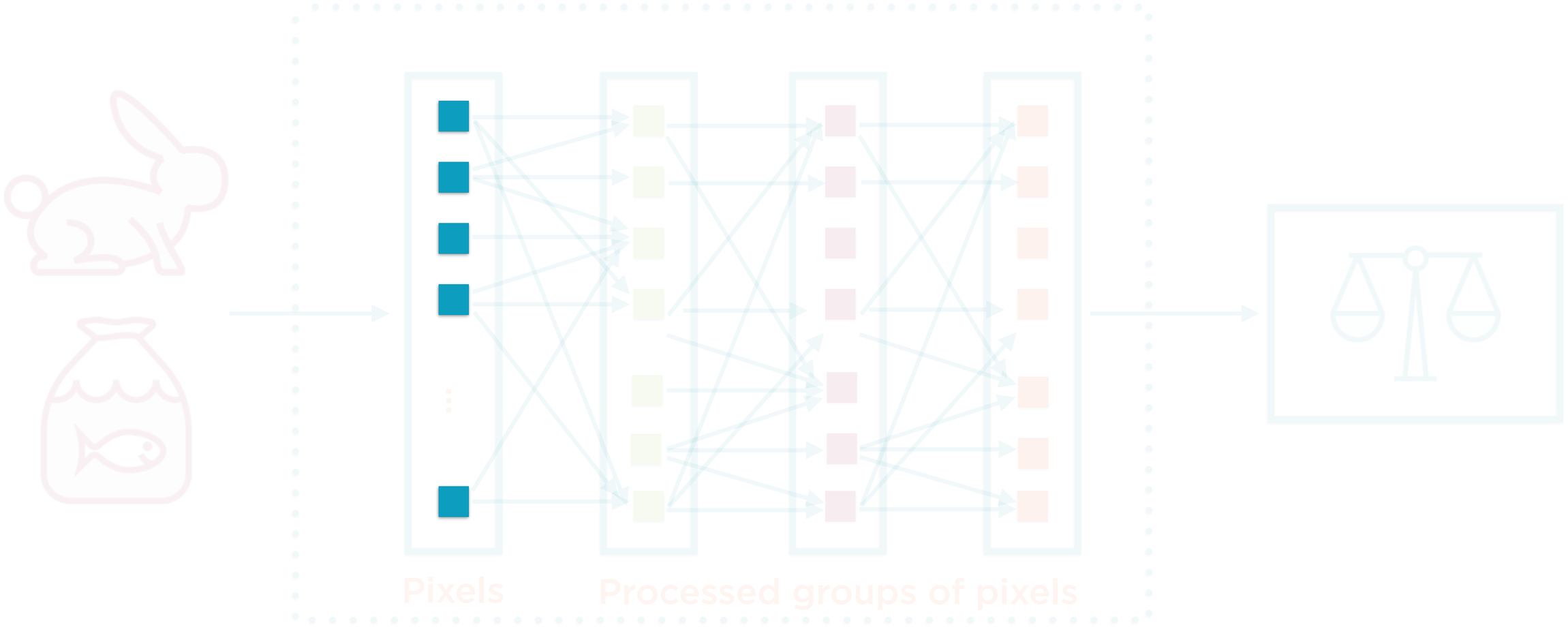
# Weights and Bias of Neuron #3



# Choice of Activation Function

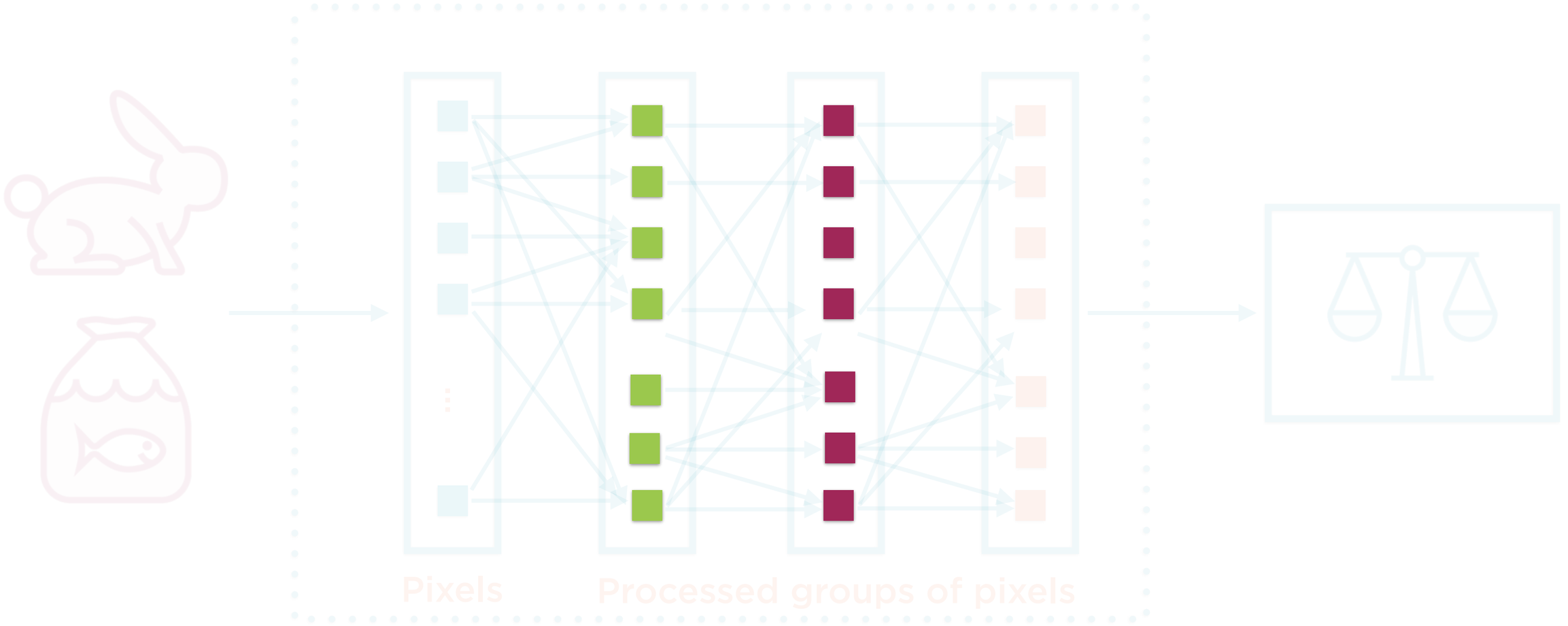


# Choice of Activation Function



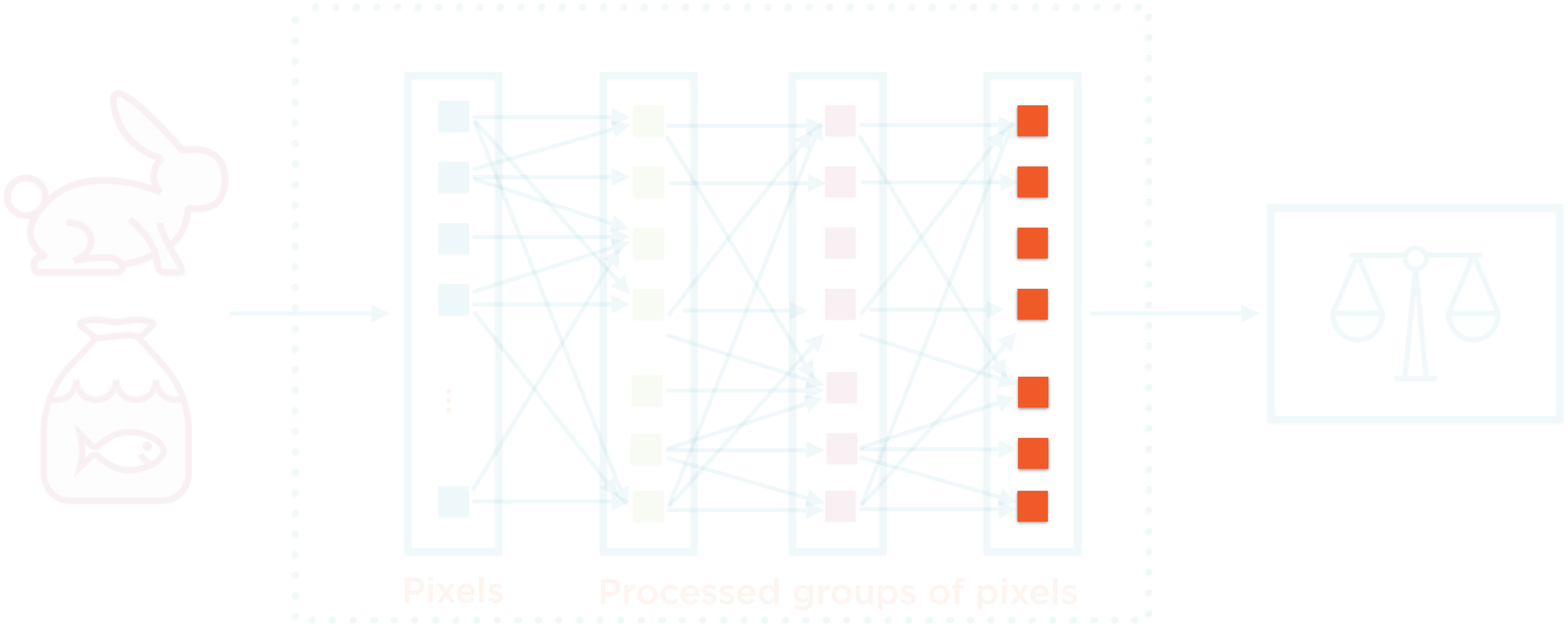

**Input layers use identity function as activation: f(x) = x**

# Choice of Activation Function



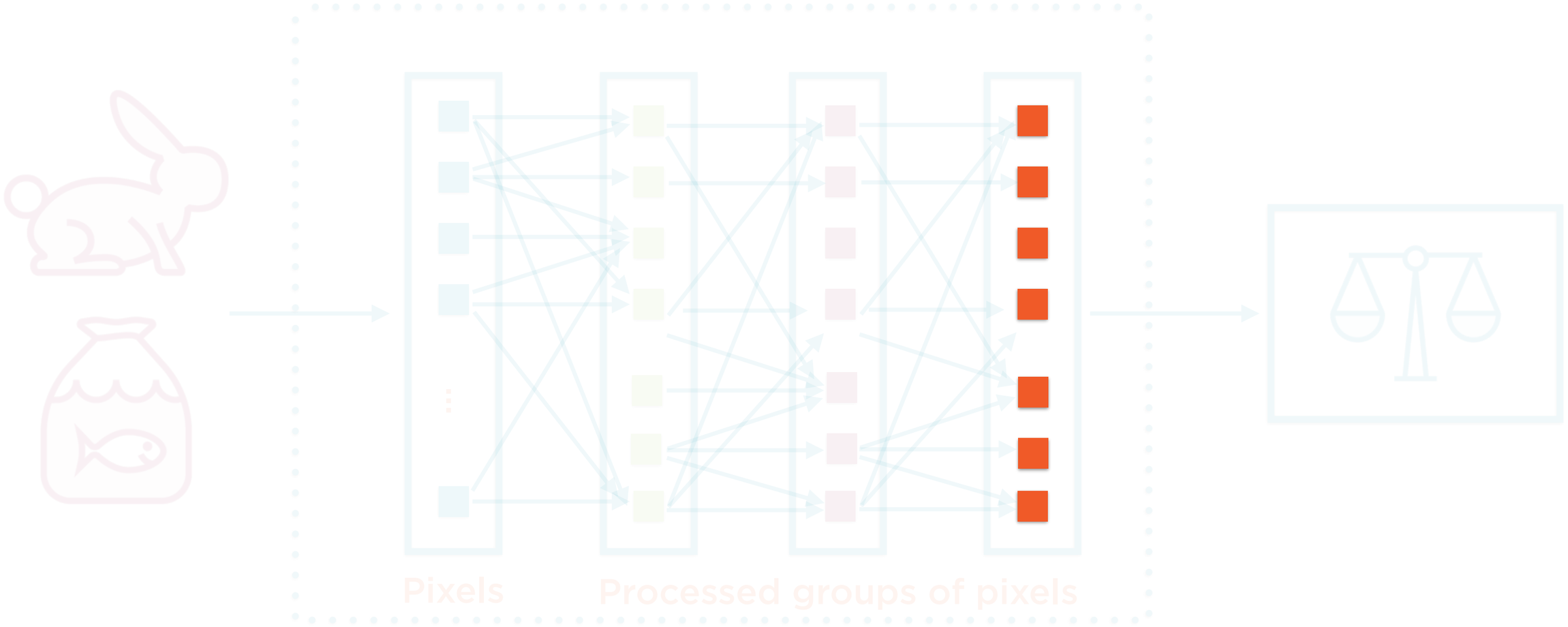

**Inner hidden layers typically use ReLU as activation function**

# Choice of Activation Function




**Output layer in our XOR example used the identity function**

# Choice of Activation Function




**Output layer in classification will often use SoftMax**


Another very common form of the activation function is the SoftMax

SoftMax(x) outputs a number between 0 and 1

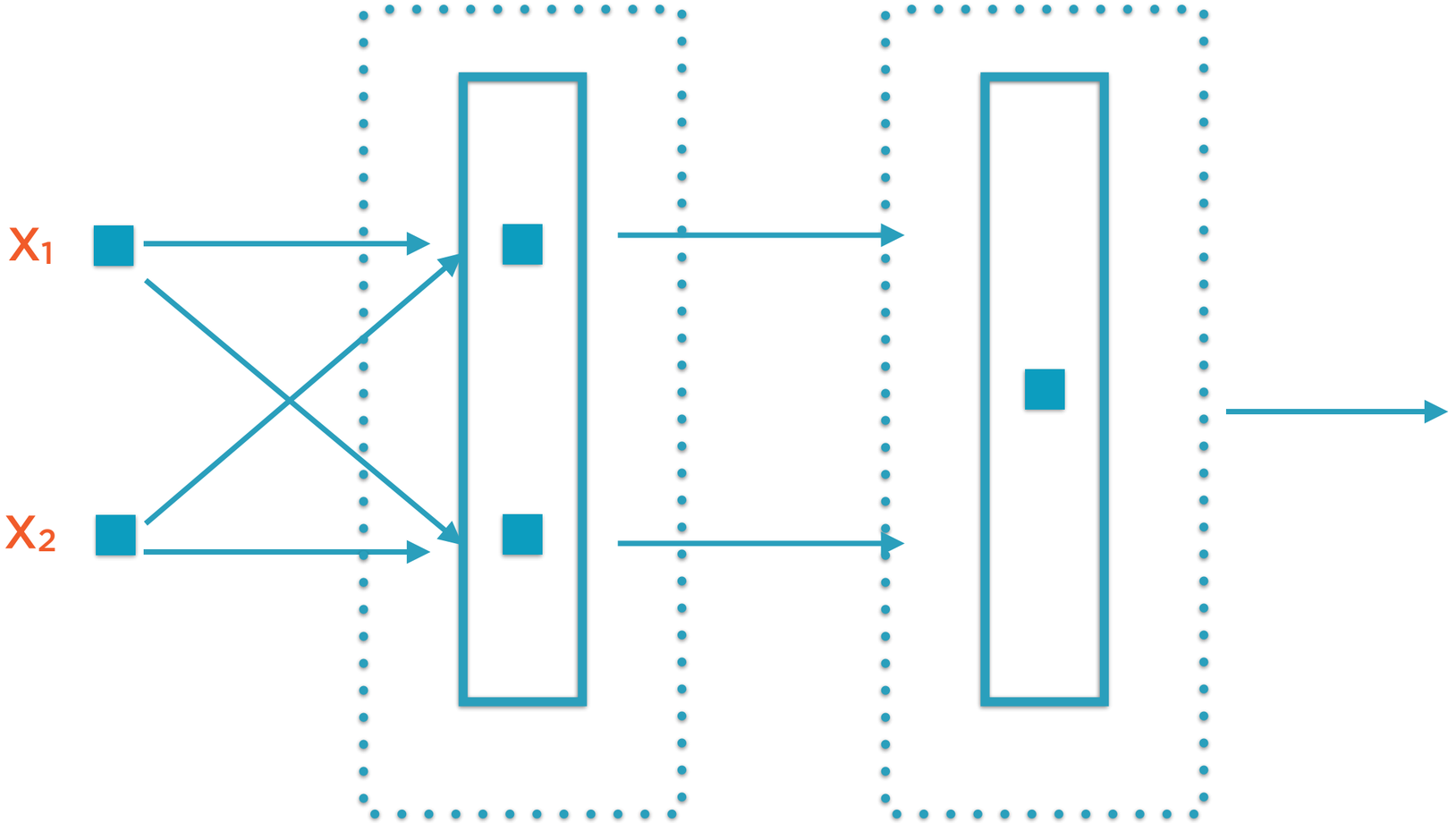
This output can be interpreted as a probability

```
def XOR(x1,x2):
  if (x1 == x2):
     return 0
  return 1
```

# “Learning” XOR

**Reverse-engineering XOR requires 3 neurons (arranged in 2 layers) as well as a non-linear activation function**

# XOR: 3 Neurons, 2 Layers




| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

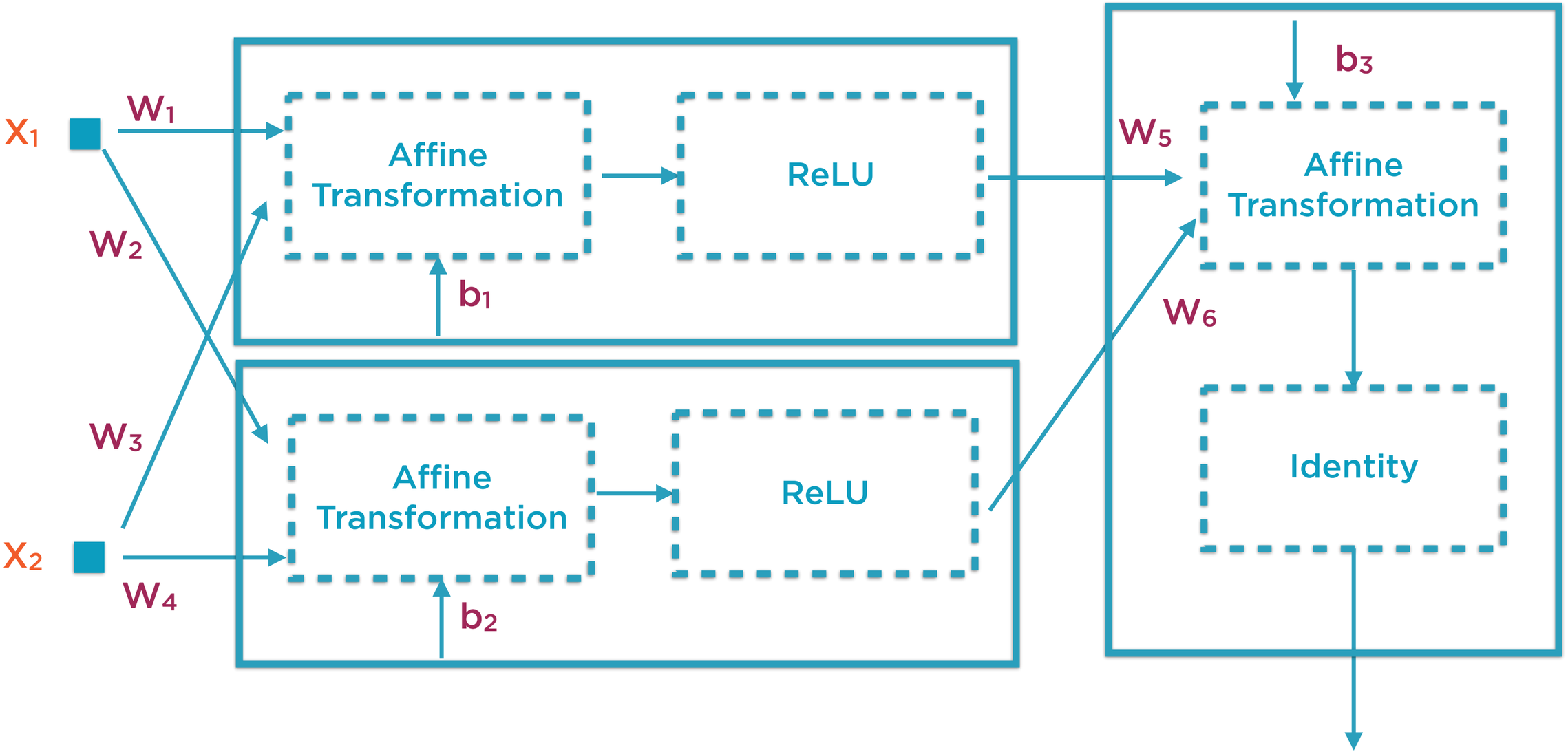
2 Inputs

Layer 1

Layer 2

Output Truth Table

# 3-Neuron XOR


$x_1$
$x_2$
$W_1$
$W_2$
$W_3$
$W_4$
Affine Transformation
ReLU
$b_1$
Affine Transformation
ReLU
$b_2$
$b_3$
$W_5$
Affine Transformation
$W_6$
Identity

```
def doSomethingReallyComplicated(x1,x2…):
   …
   …
   …
   return complicatedResult
```

# “Learning” Arbitrarily Complex Functions

**Adding layers to a neural network can “learn” (reverse-engineer) pretty much anything**

# Summary

A neuron is the smallest entity in a neural network

Linear regression can be learnt by a single neuron

A more complex function such as XOR requires more neurons

Combinations of interconnected neurons can “learn” virtually anything

Training such networks to use the “best” parameter values is vital