Supervised Learning of Neural Random-Access Machines with Differential Evolution

Valerio Belli

April 10, 2018

Università degli Studi di Perugia

Neural Random-Access Machines

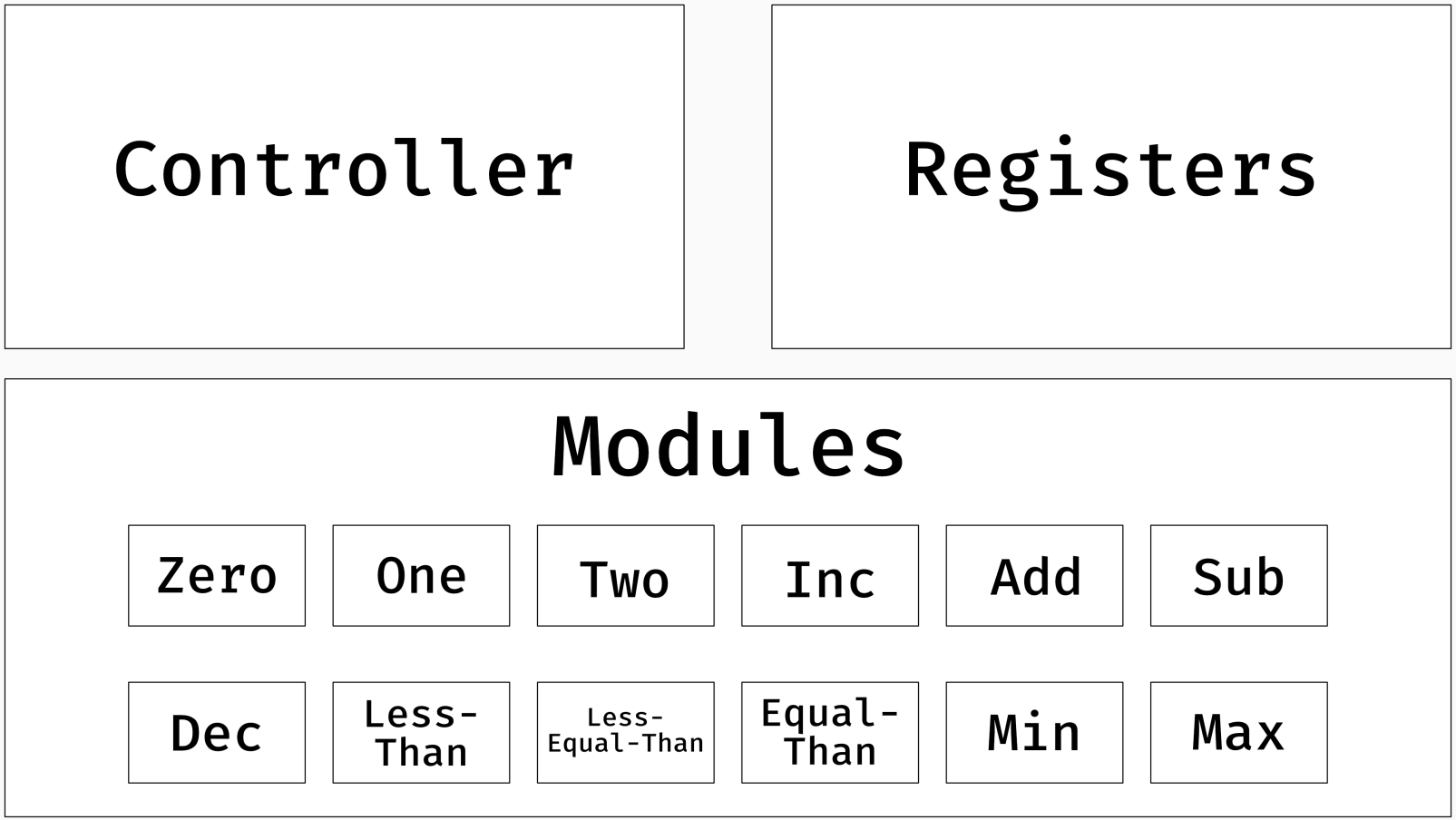
What is?

It is a machine introduced in [Kurach et al., 2015] based on a neural network which is capable of manipulating pointers and dereferencing them through “logical” circuits. Its objective is to solve a task on which it has been trained creating and executing those circuits.

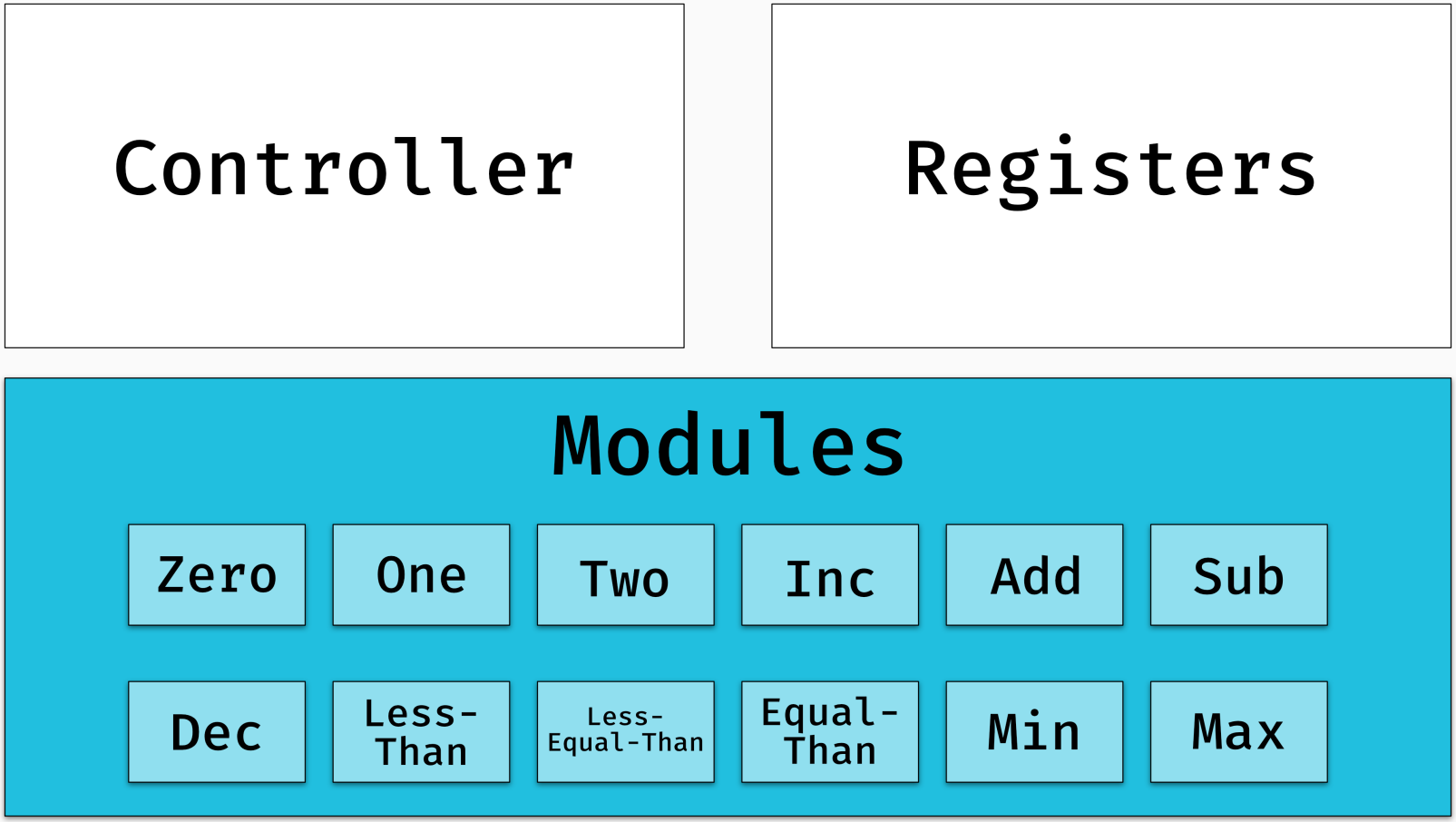
Introduction

Let $N = \{1, \ldots, I-1\}$, where I is an integer constant, the integers set over the NRAM should work. Since the training in [Kurach et al., 2015] is made through a gradient descent algorithm, the NRAM does not work directly over N but over stochastically independent probability distributions, defined as $p \in \mathbb{R}^I$ satisfying $0 \leq p_i \leq 1$ and $\sum_{i=0}^{I-1} p_i = 1$. Moreover, let T the maximum number of timestep of execution of the NRAM.

Registers only version



Modules


Controller
Registers
Modules
Zero
One
Two
Inc
Add
Sub
Dec
Less-Than
Less-Equal-Than
Equal-Than
Min
Max

Modules

The modules (or gates) are components through which the controller, connecting them, manipulates values and pointers. In the NRAM exist three types of modules, defined as follows:

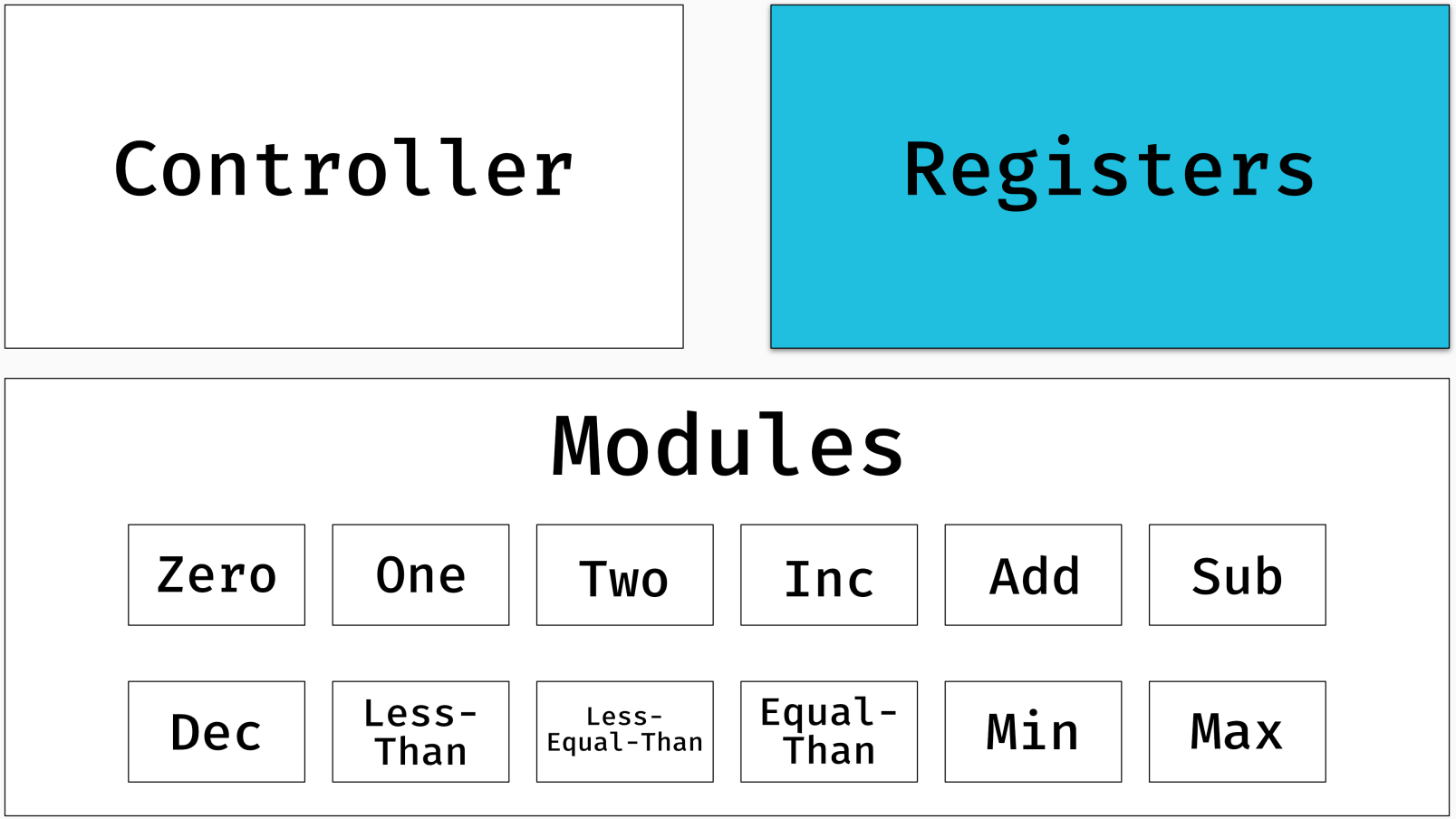
$$m_i \in N \ (\text{Constant modules}) \tag{1}$$

$$m_i : p \to p \ (\text{Unary modules}) \tag{2}$$

$$m_i : p \times p \to p \ (\text{Binary modules}) \tag{3}$$

Each of them takes as input and returns probability distributions, as exception are constant modules which return only probability distributions.

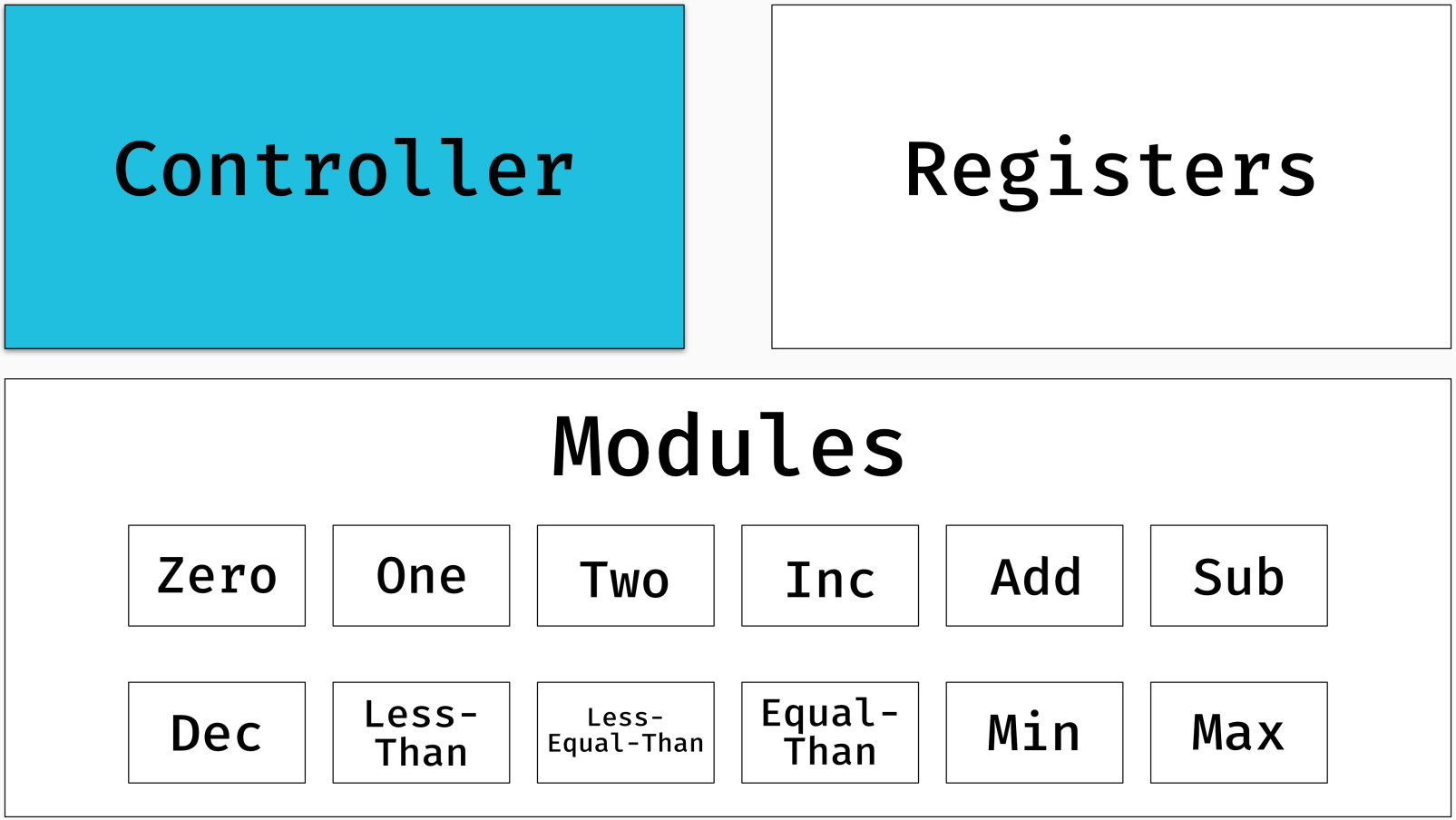
Registers


Controller
Registers
Modules
Zero
One
Two
Inc
Add
Sub
Dec
Less-Than
Less-Equal-Than
Equal-Than
Min
Max

Registers

The registers are a set of memory cells, where each of them contains a probability distribution. Hence, in other words, every register play the role of a random variable.

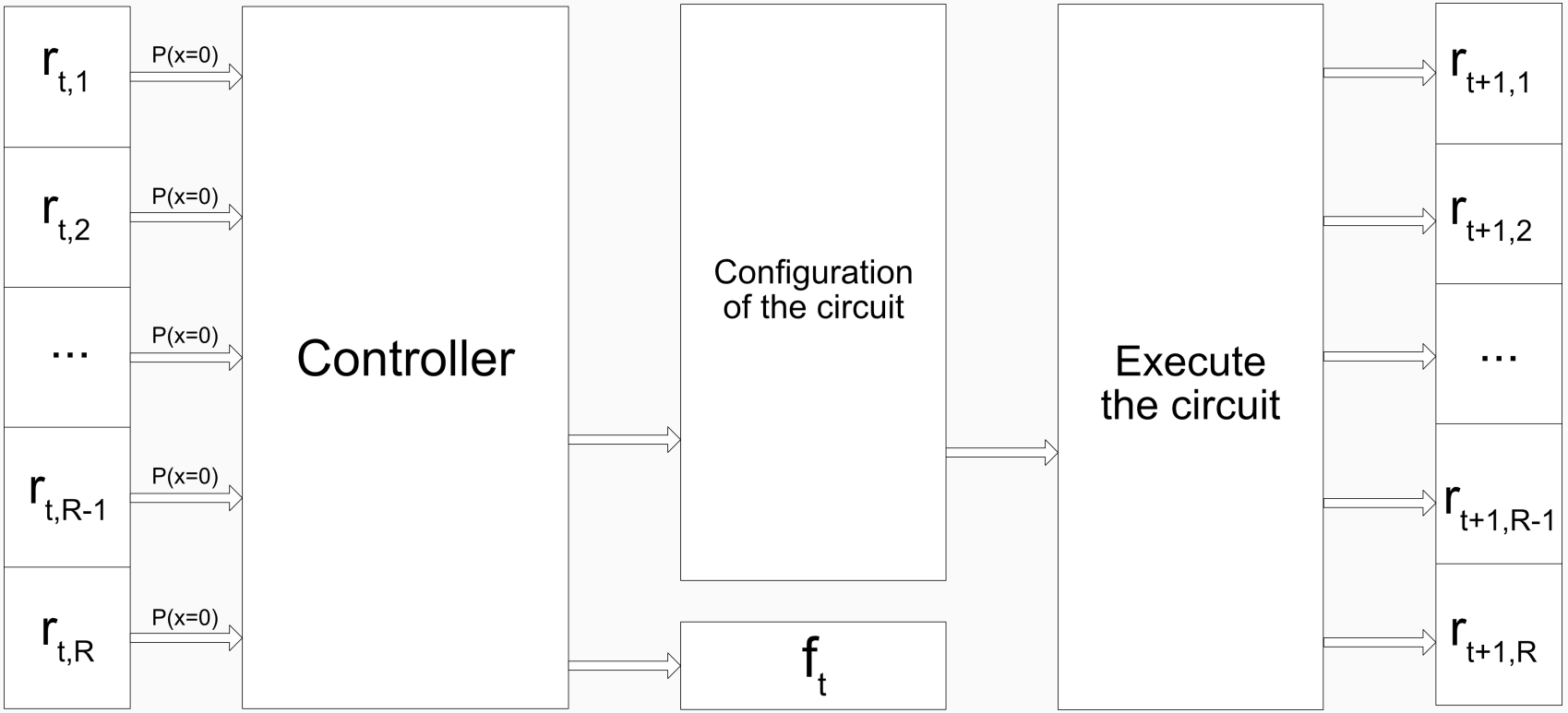
Controller


Controller
Registers
Modules
Zero
One
Two
Inc
Add
Sub
Dec
Less-Than
Less-Equal-Than
Equal-Than
Min
Max

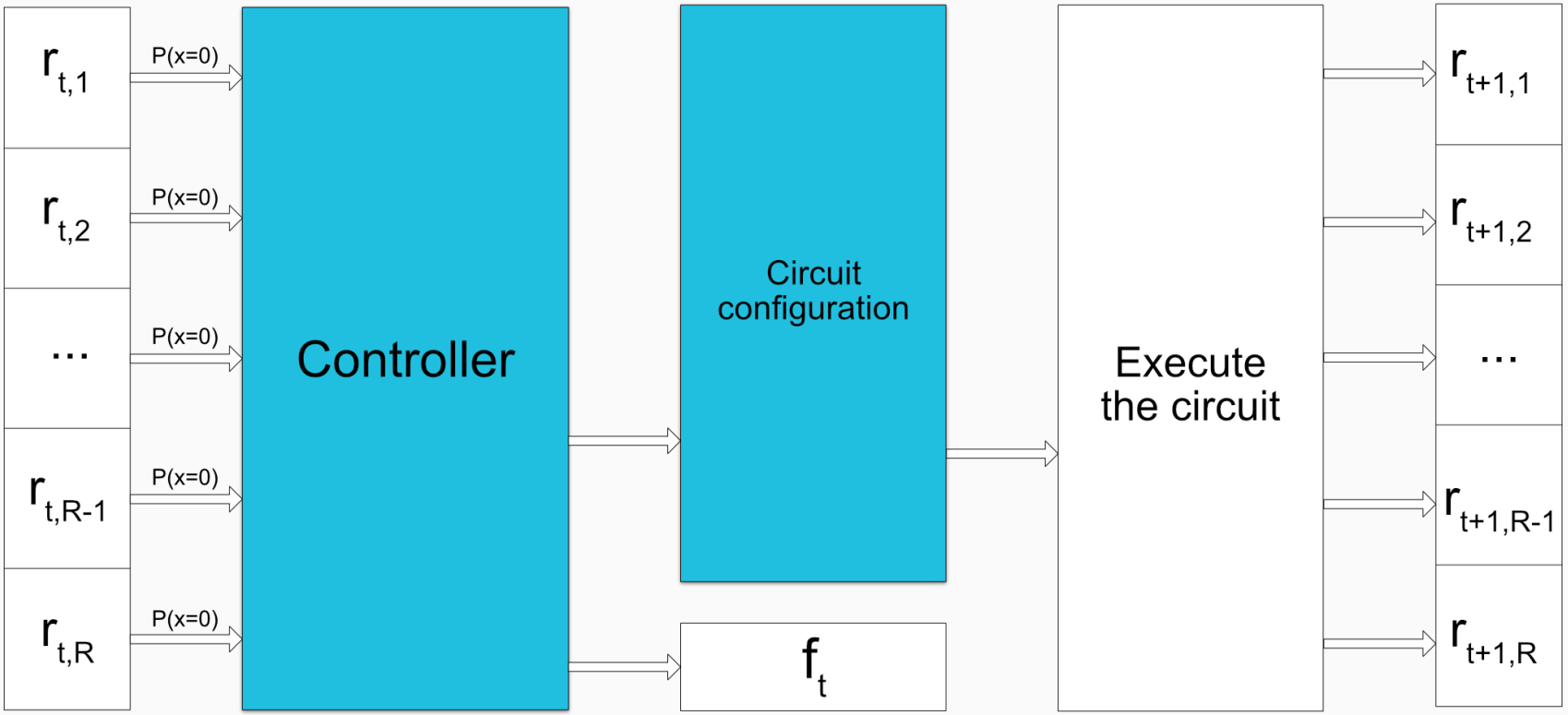
Controller

The controller is a neural network (Multi Layer Perceptron) whose objective is to emit a circuit configuration, i.e. how the circuits are connected together in form of a Directed Acyclic Graph. To do this the controller takes as input the $\mathbb{P}(r_i = 0)$ of each register.

Timestep execution

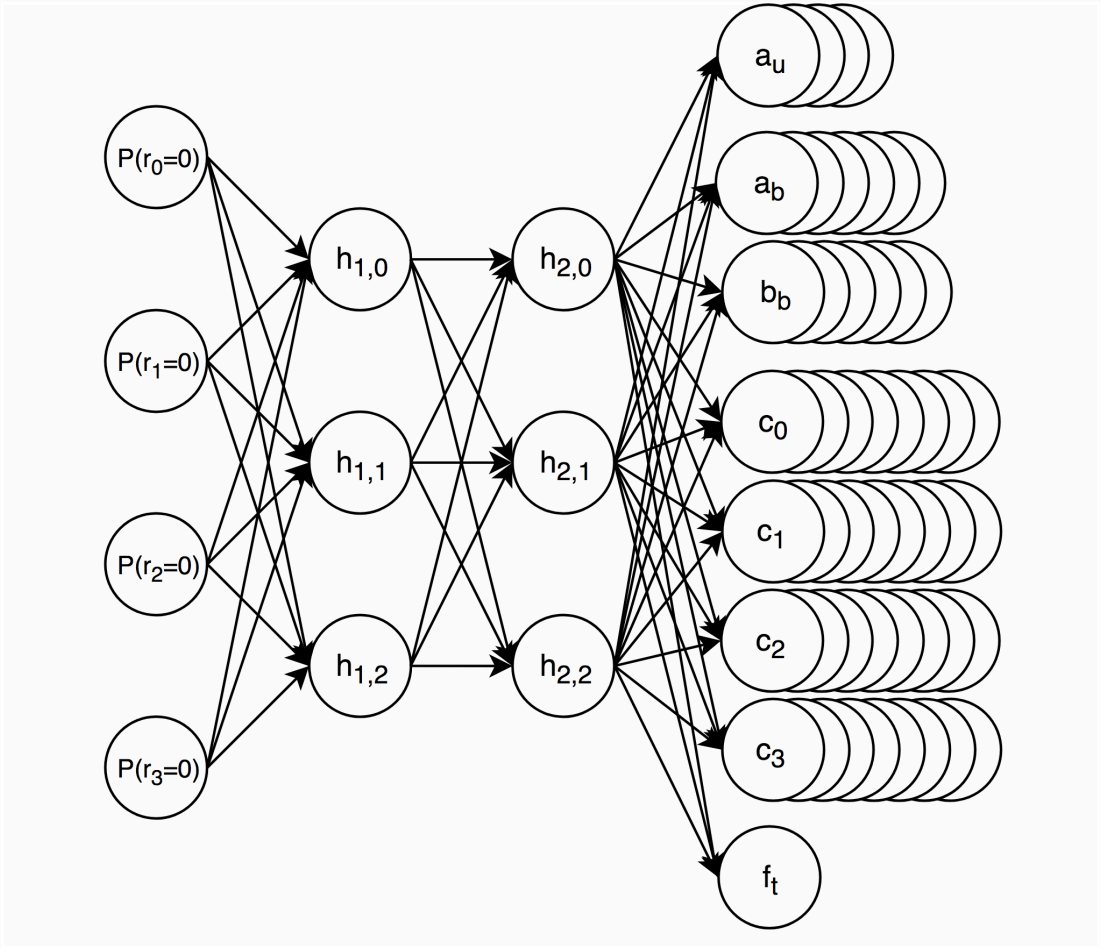


Timestep execution - Controller & Circuit configuration structure



Timestep execution - Controller

Let $R = 4$ and Gates$= \{m_u, m_c, m_b\}$, the controller structure appear as follows



Timestep execution - Circuit configuration structure

Using the same assumptions seen in the previous slide, the structure of the circuit configuration is represented as a list of vectors as follows

$softmax(a_u)$	0	0	0	1			
$softmax(a_b)$	0	0	0.4	0	0.7	0	
$softmax(b_b)$	0	0.4	0.1	0.1	0.3	0.1	
$softmax(c_0)$	0.2	0.2	0.3	0.1	0.2	0	0
$softmax(c_1)$	0.3	0.1	0.1	0.4	0	0.1	0
$softmax(c_2)$	0.6	0.3	0.1	0	0	0	0
$softmax(c_3)$	0	1	0	0	0	0	0
$\sigma(f_t)$	0.3						

Timestep execution - Selection of the input of the gates

An input for the gate m_i is chosen by the controller from the set $\{r_1, \ldots, r_R, o_1, \ldots, o_{i-1}\}$ where:

- r_j is the content of the j^{th} register, $j = 1, \ldots, R$;
- o_k is the output of the k^{th} previous gate with respect to the current m_i, for $k = 1, \ldots, |Gates| - 1$.

selected through a weighted average with a coefficient s_i as follows

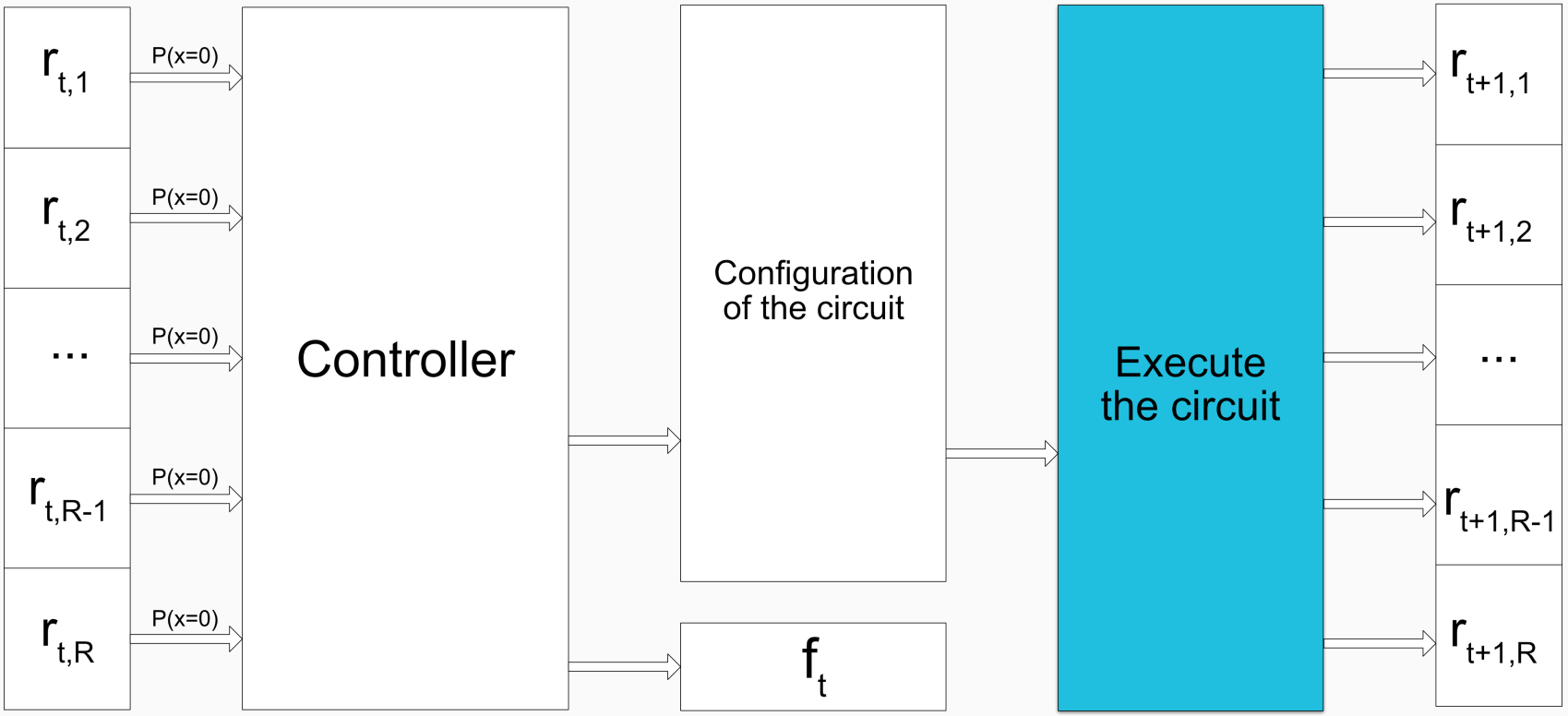
$$(r_1, r_2, \ldots, r_R, o_1, o_2, \ldots, o_{i-1})^T \mathbf{softmax}(s_i)$$

Timestep execution - Selection of the input of the registers

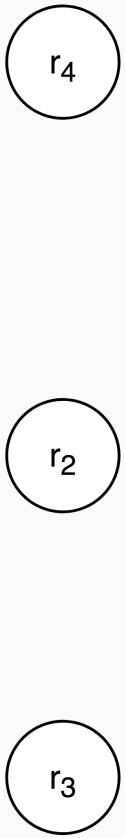
Similarly to the gates, also the content of the registers is updated as follows:

$$r_i^{(t+1)} = (r_1^{(t)}, \ldots, r_R^{(t)}, o_1^{(t)}, \ldots, o_{|\mathrm{Gates}|}^{(t)})^T \mathbf{softmax}(s_i)$$

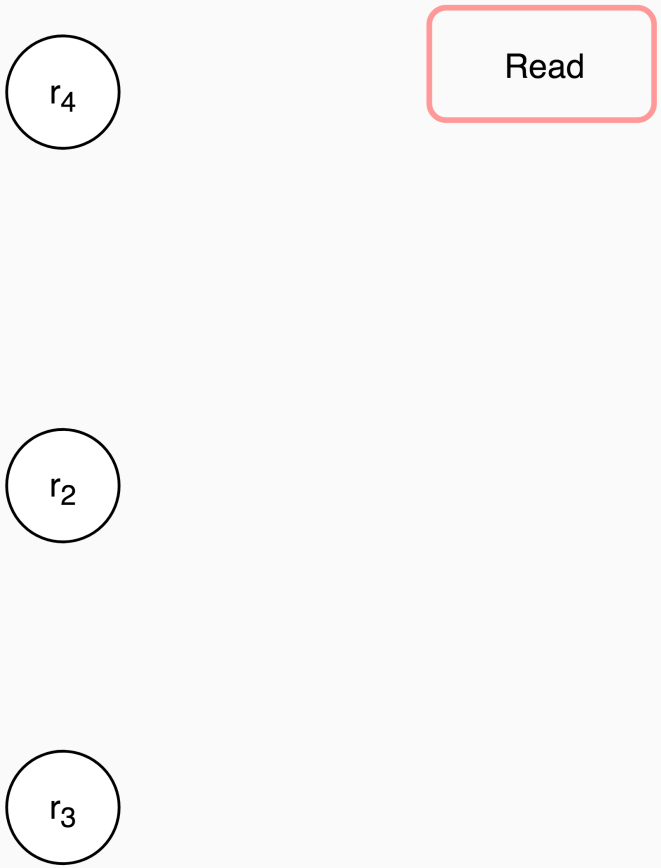
Timestep execution - Circuit execution example



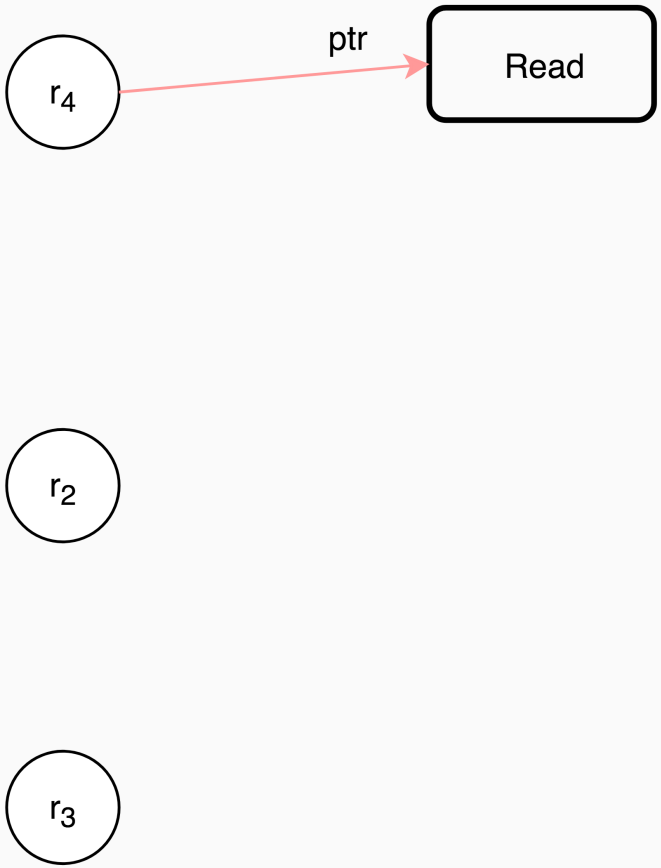
Timestep execution - Circuit execution example



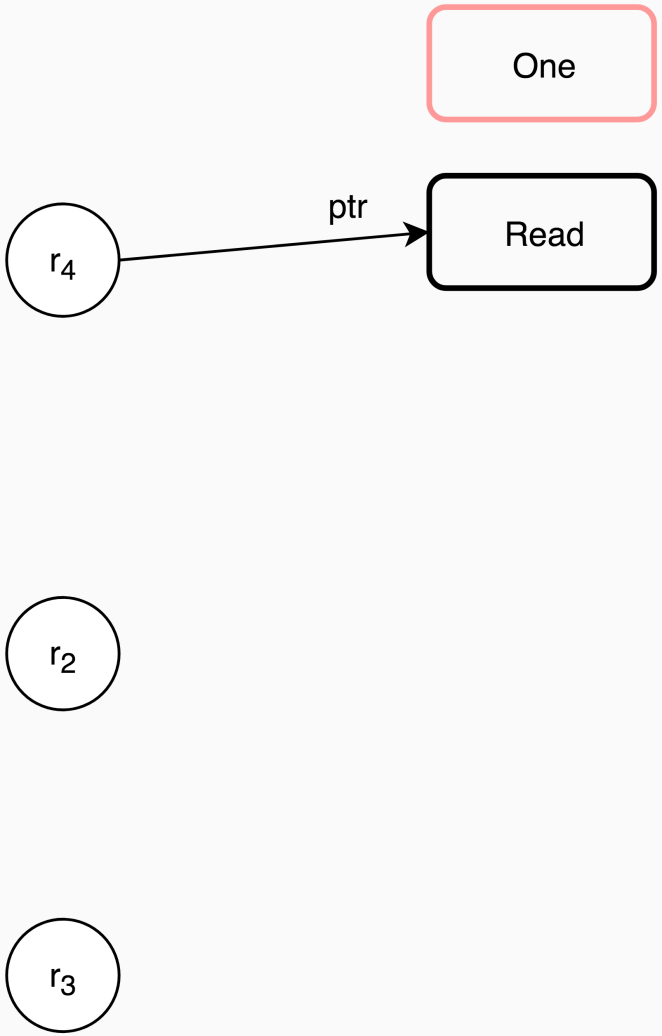
Timestep execution - Circuit execution example



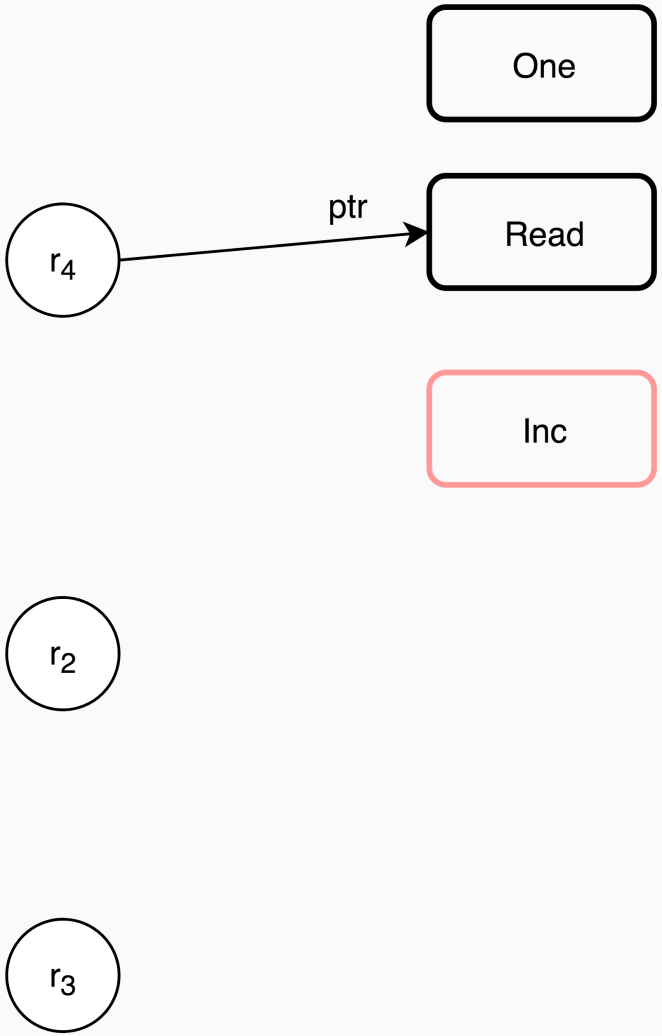
Timestep execution - Circuit execution example



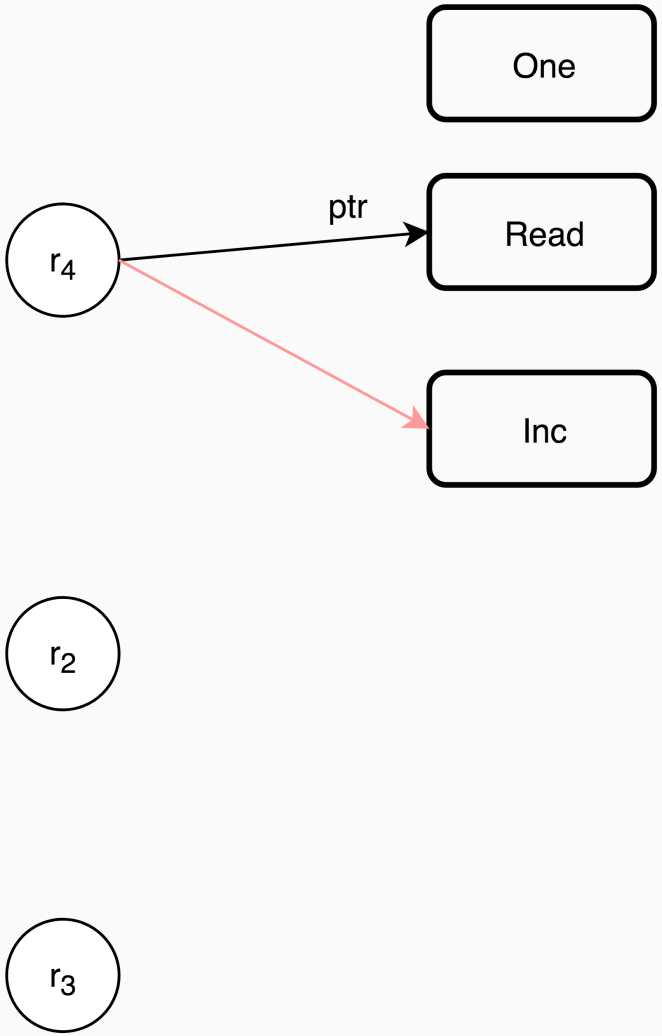
Timestep execution - Circuit execution example



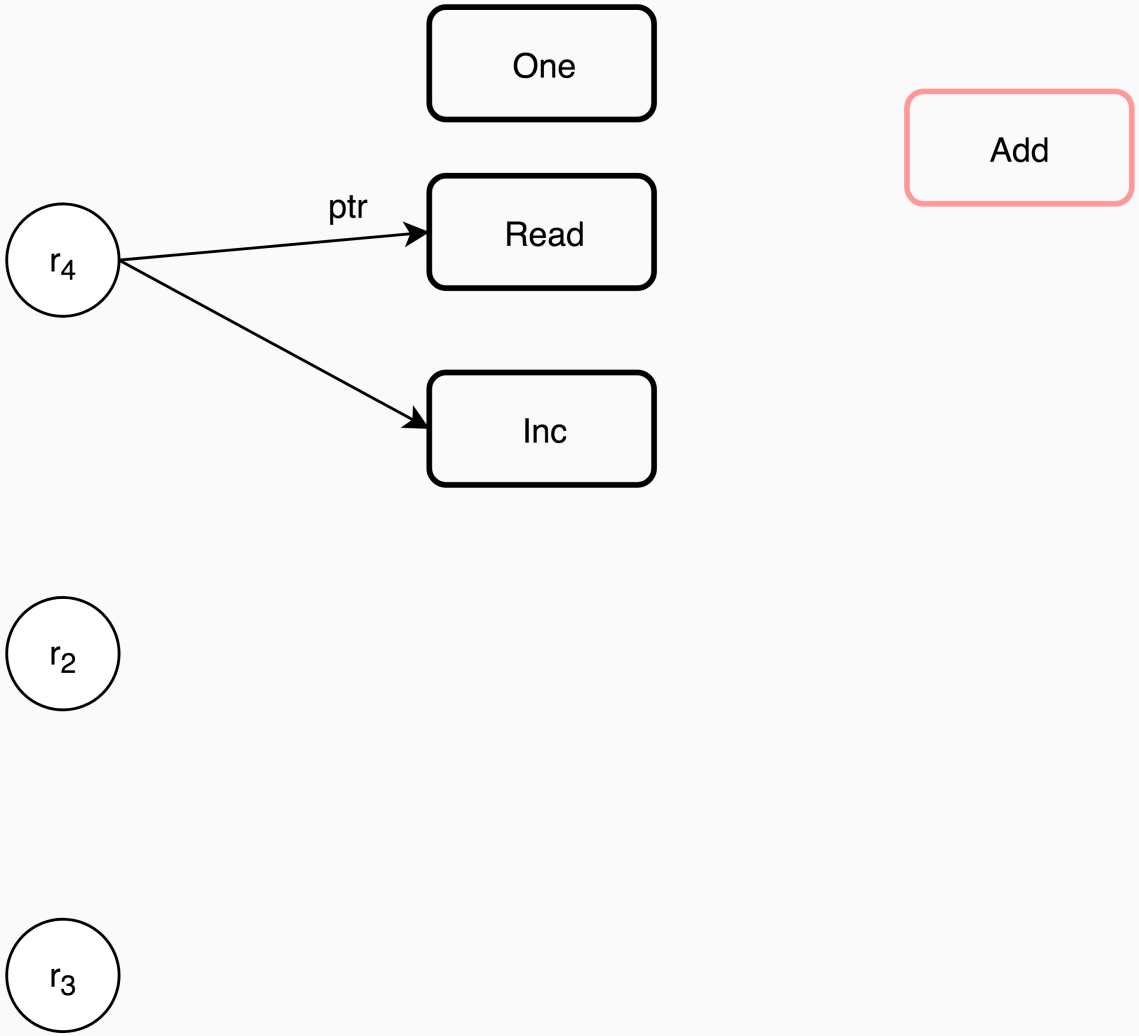
Timestep execution - Circuit execution example



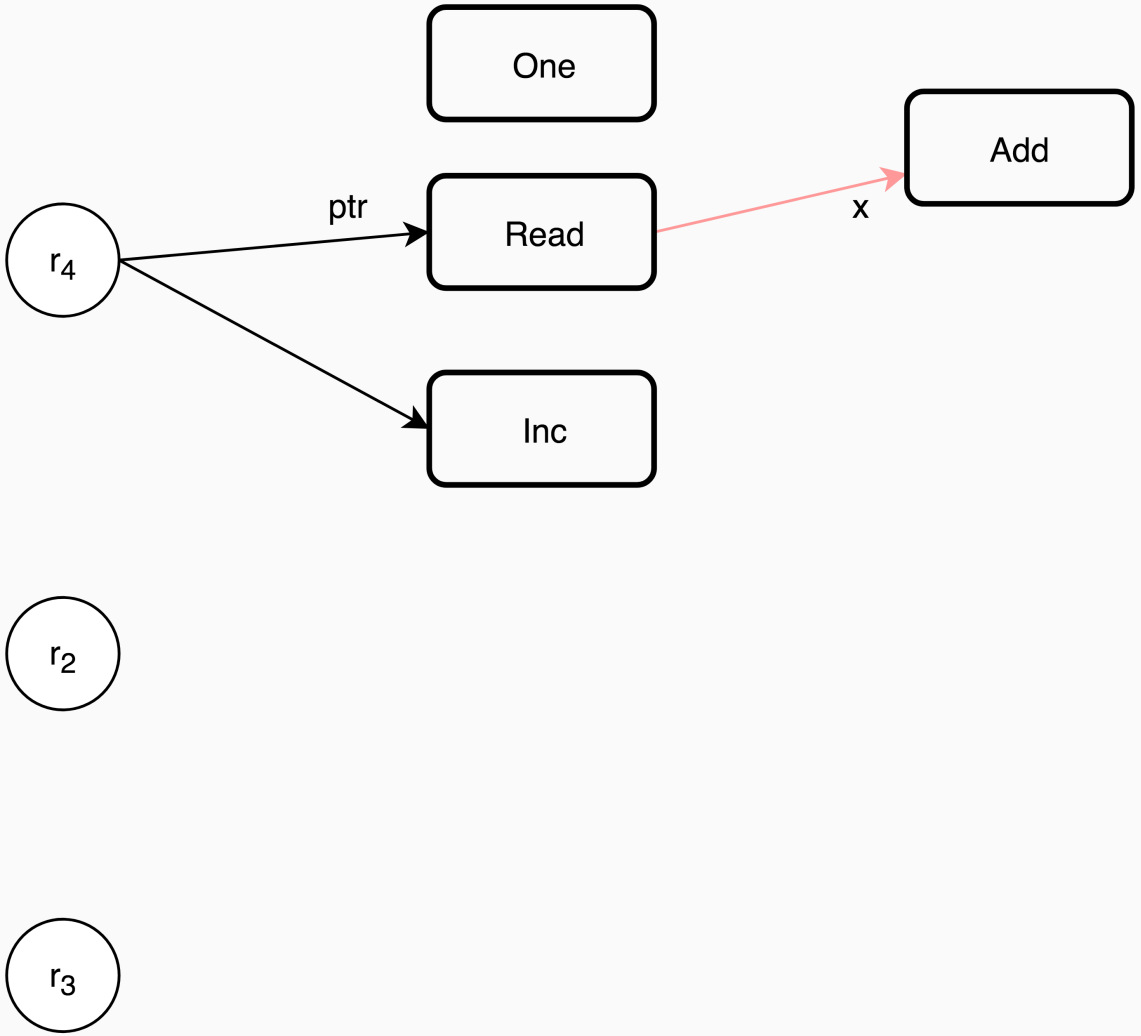
Timestep execution - Circuit execution example



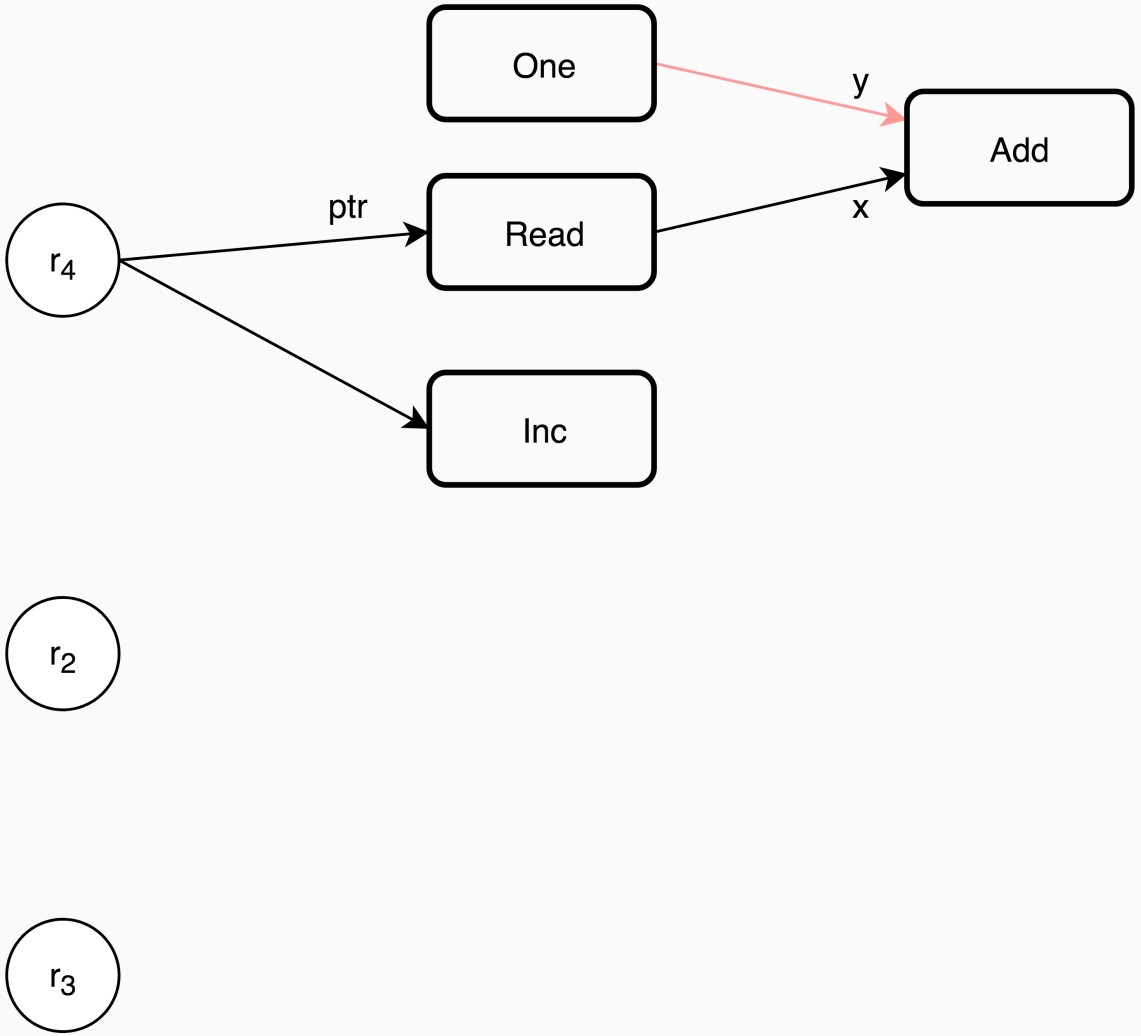
Timestep execution - Circuit execution example



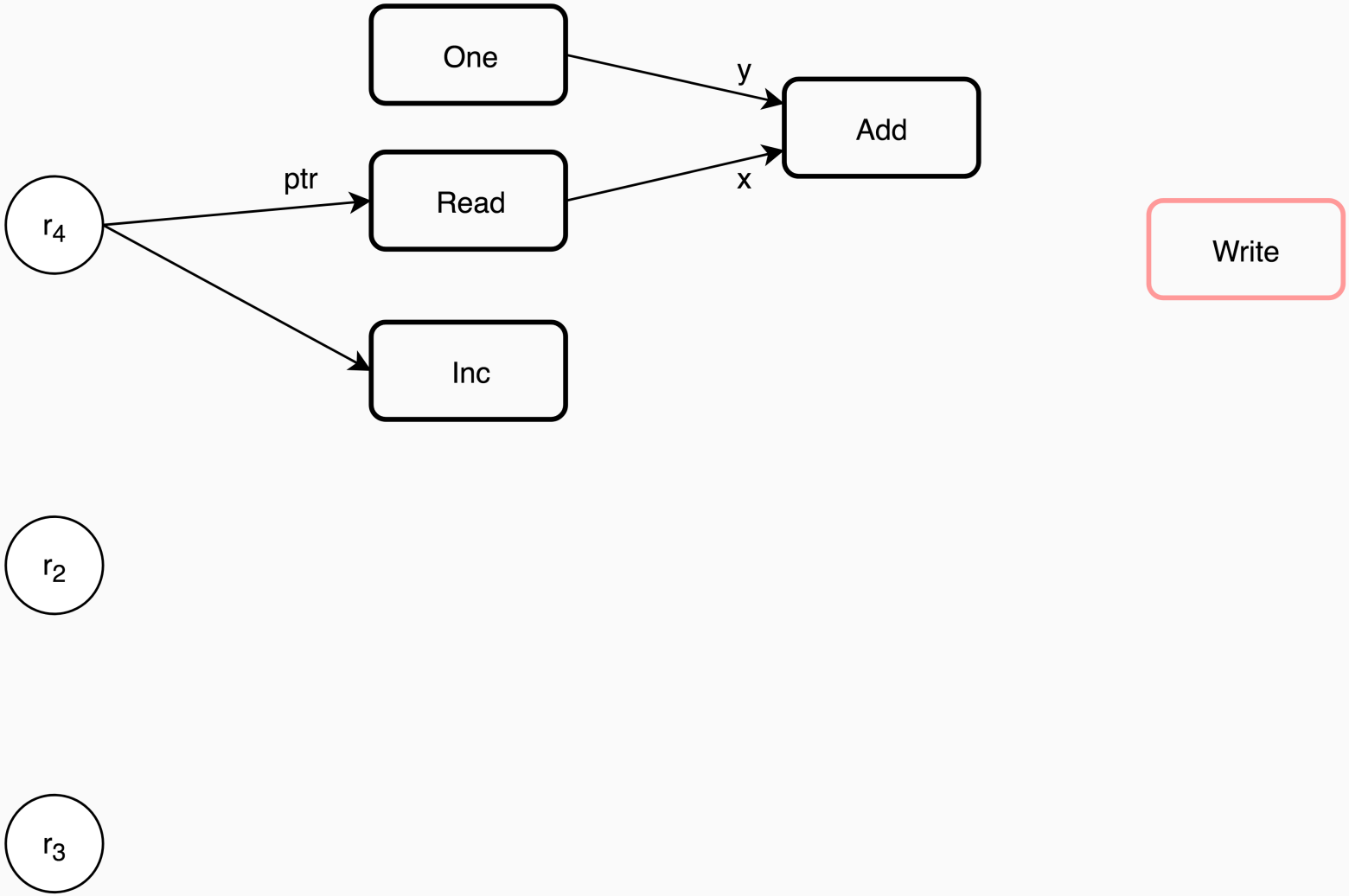
Timestep execution - Circuit execution example



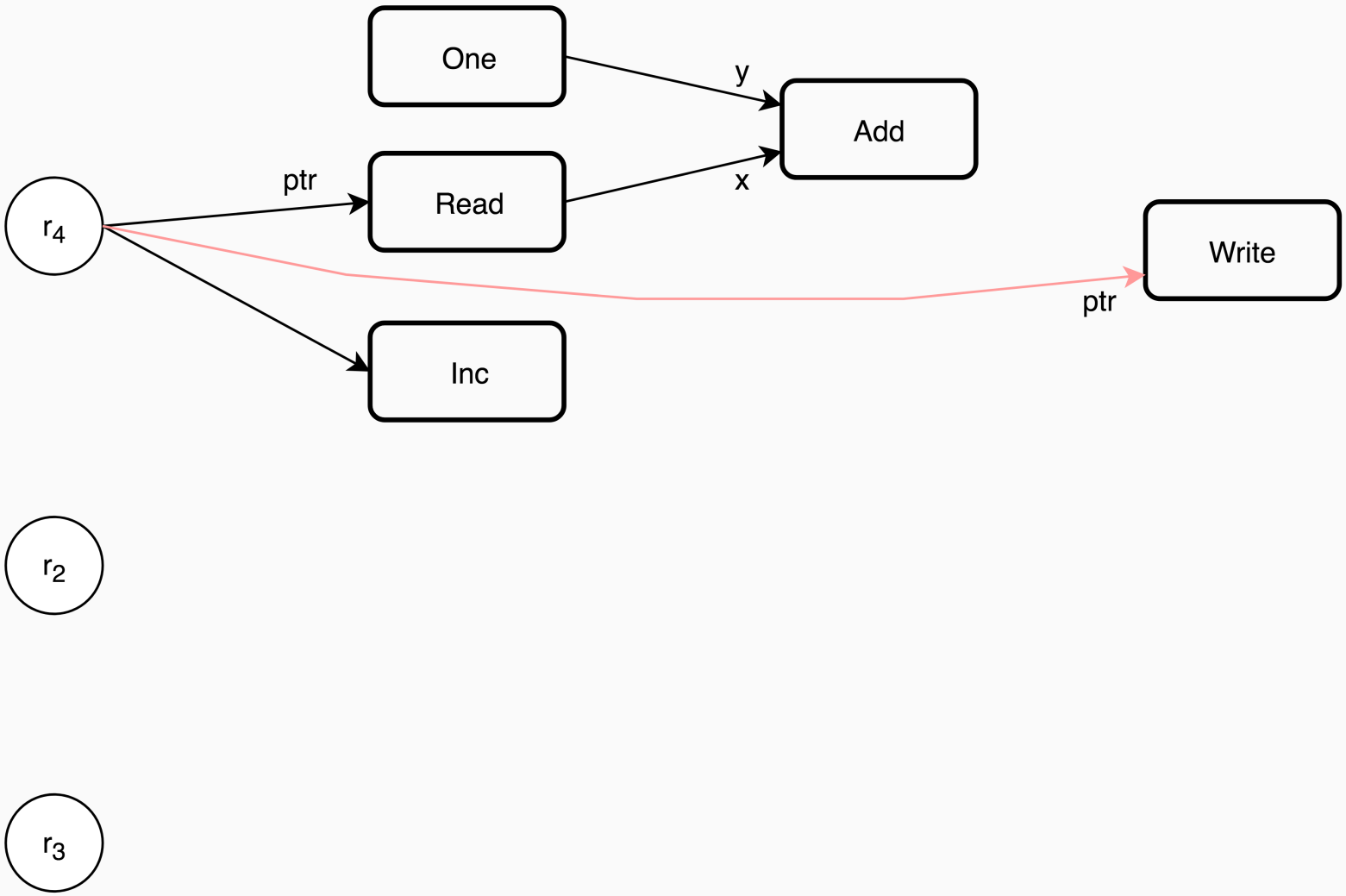
Timestep execution - Circuit execution example



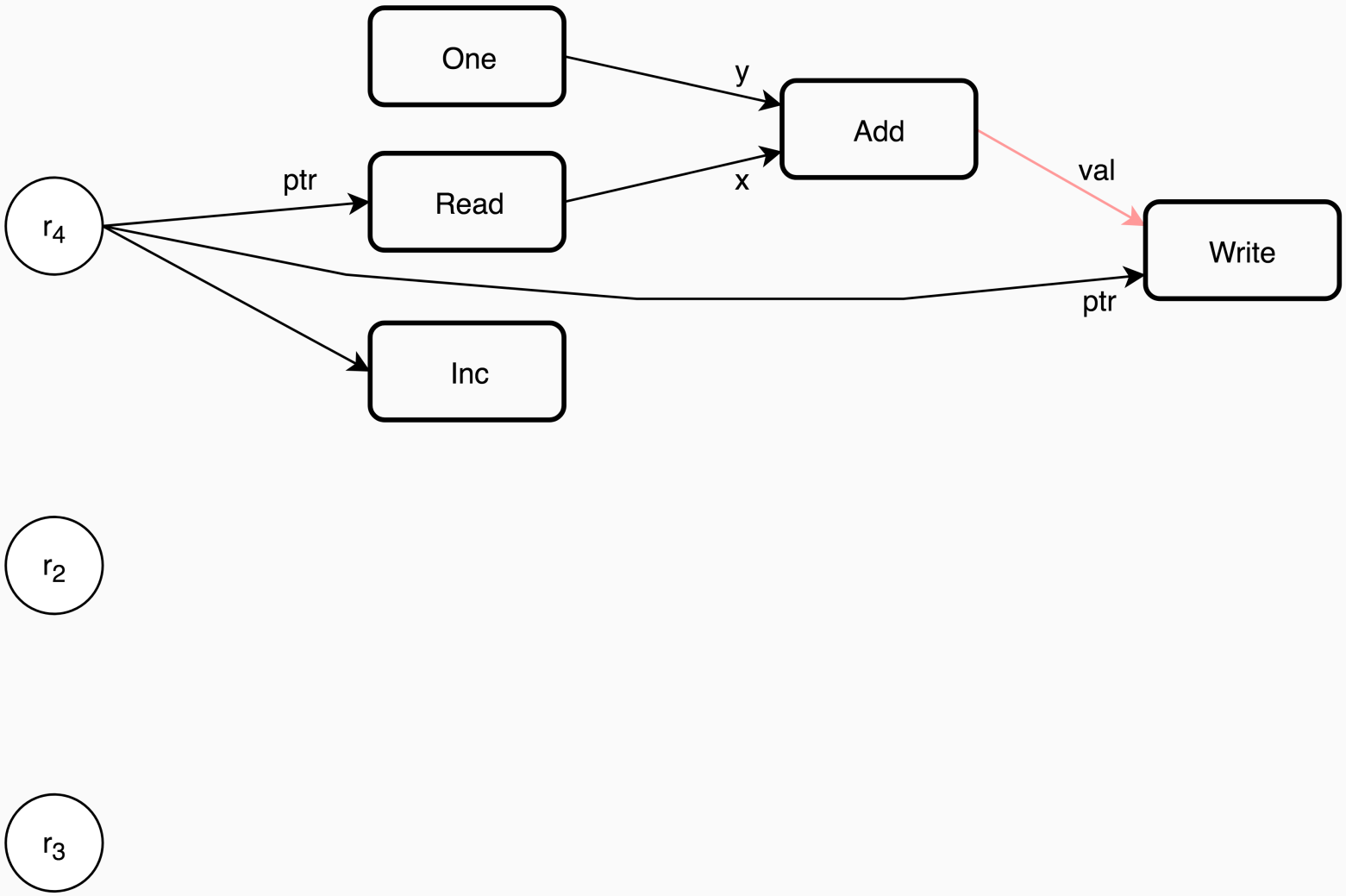
Timestep execution - Circuit execution example



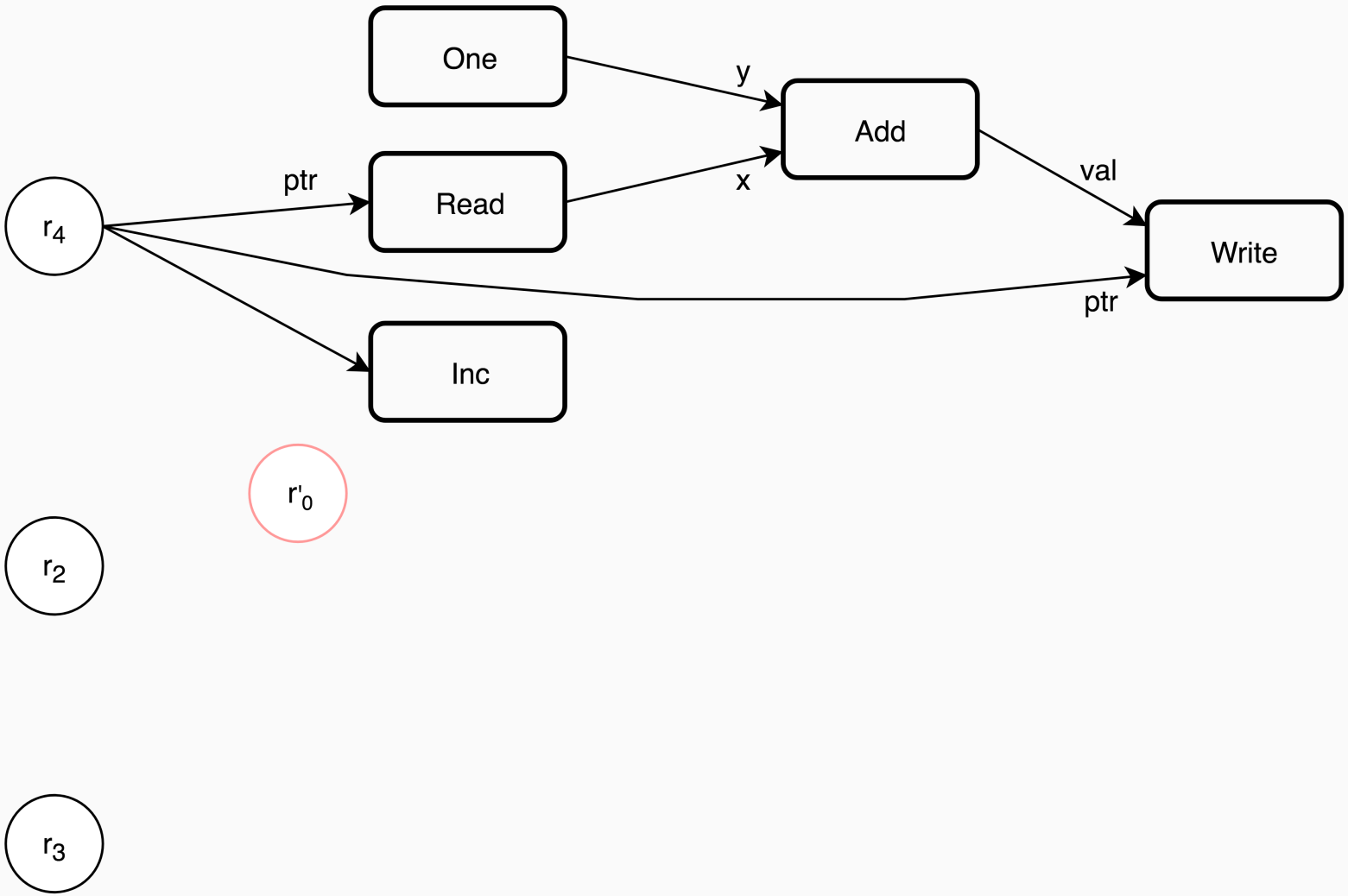
Timestep execution - Circuit execution example



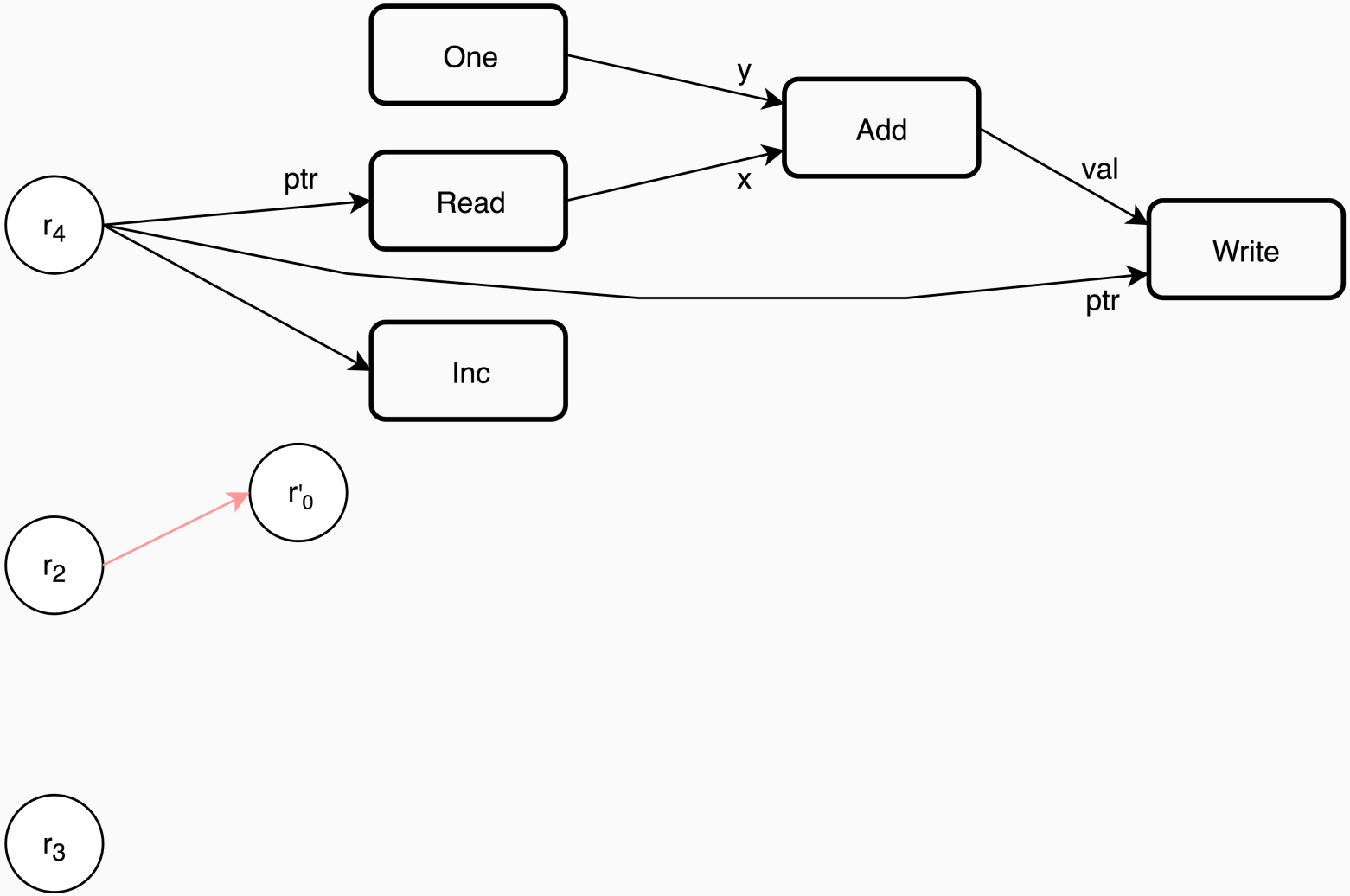
Timestep execution - Circuit execution example



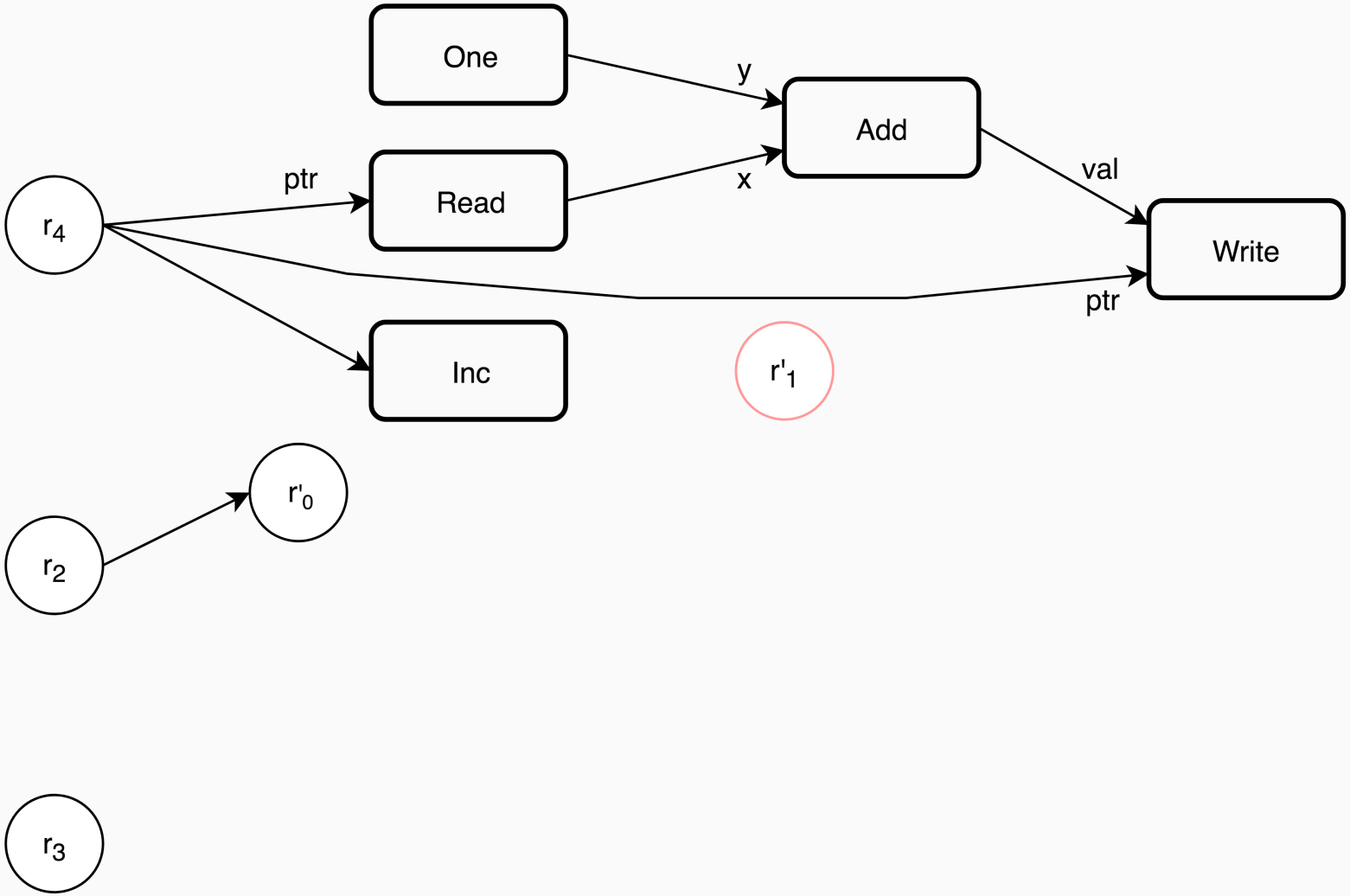
Timestep execution - Circuit execution example



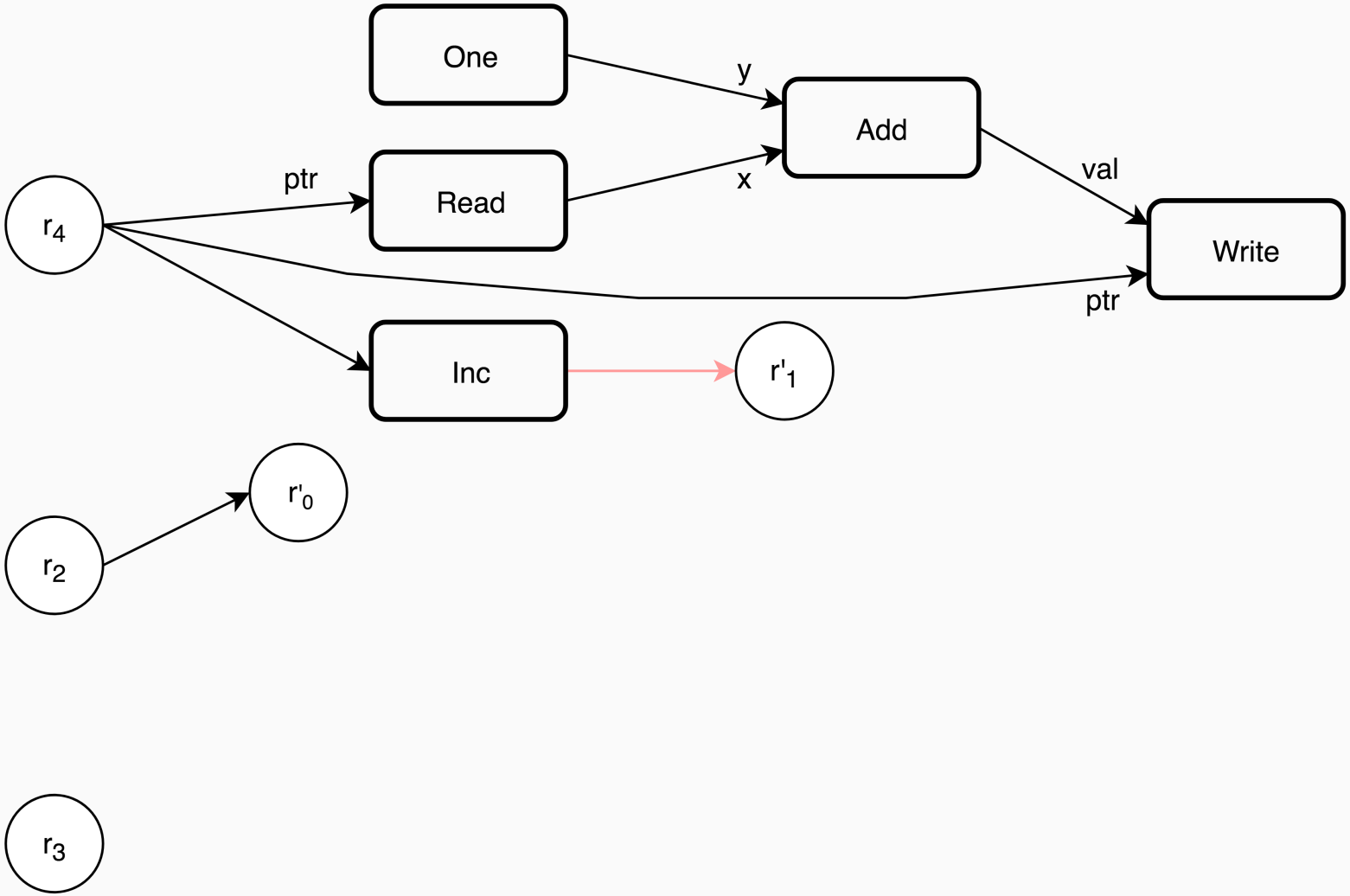
Timestep execution - Circuit execution example



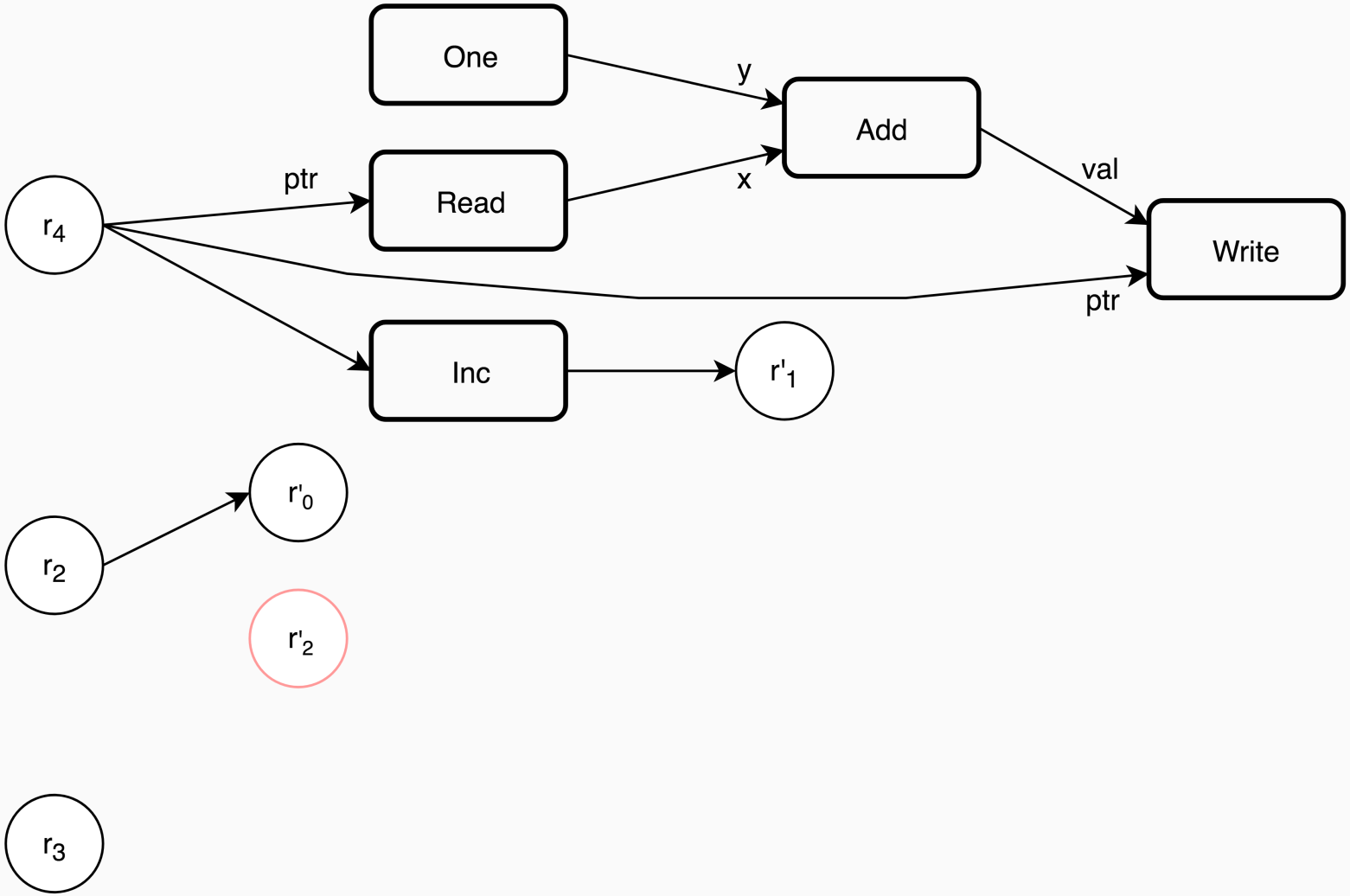
Timestep execution - Circuit execution example



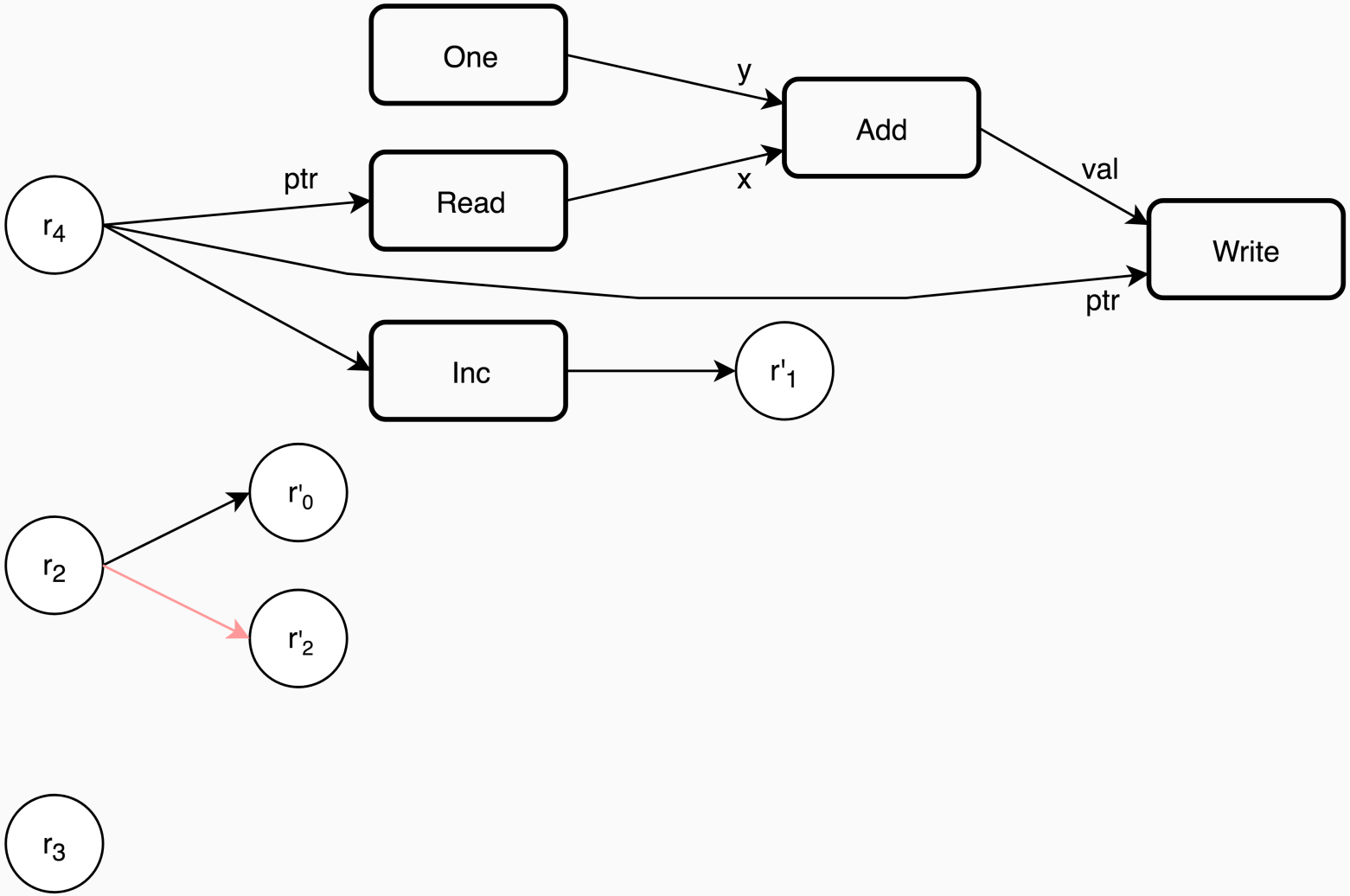
Timestep execution - Circuit execution example



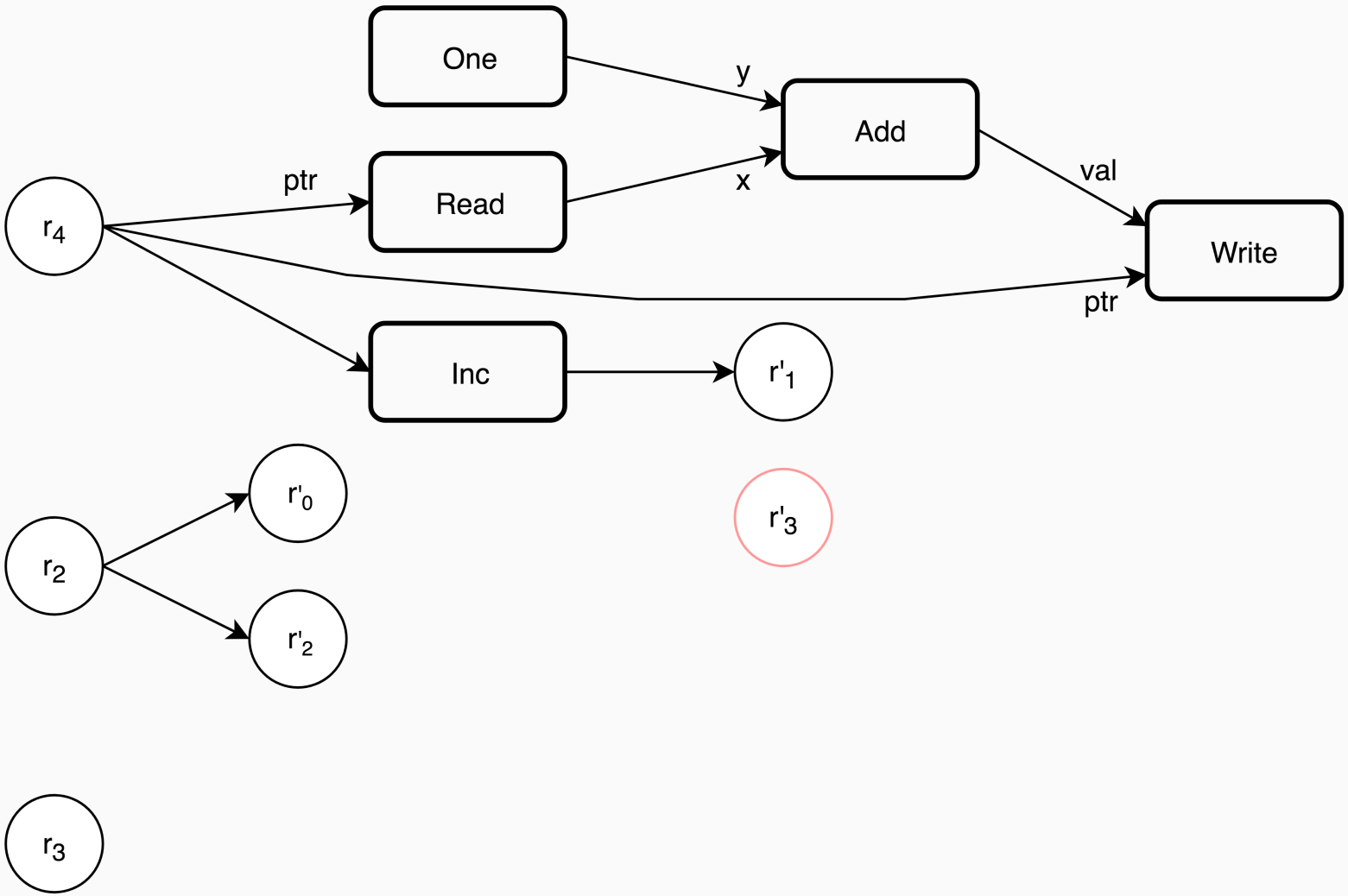
Timestep execution - Circuit execution example



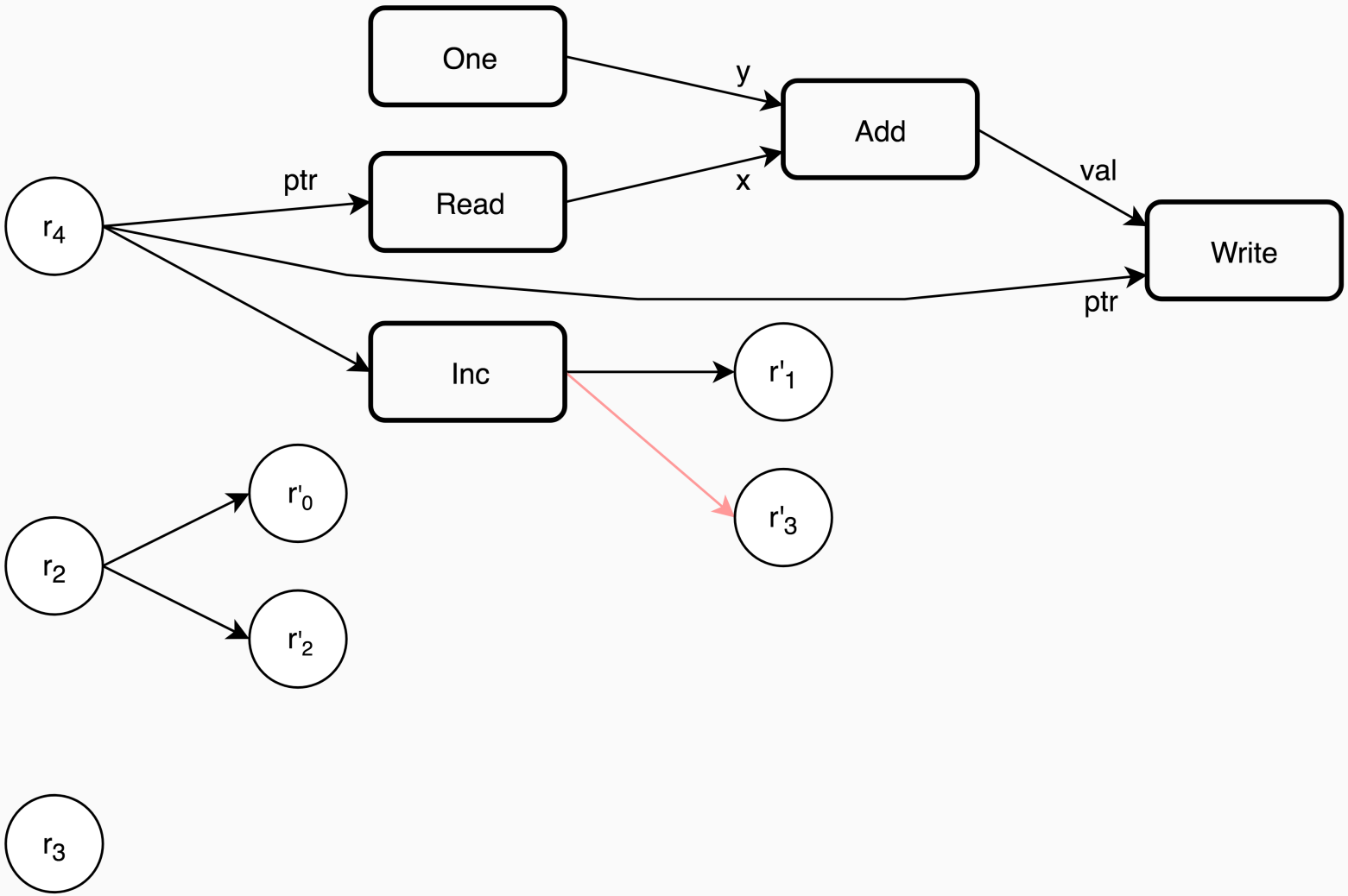
Timestep execution - Circuit execution example



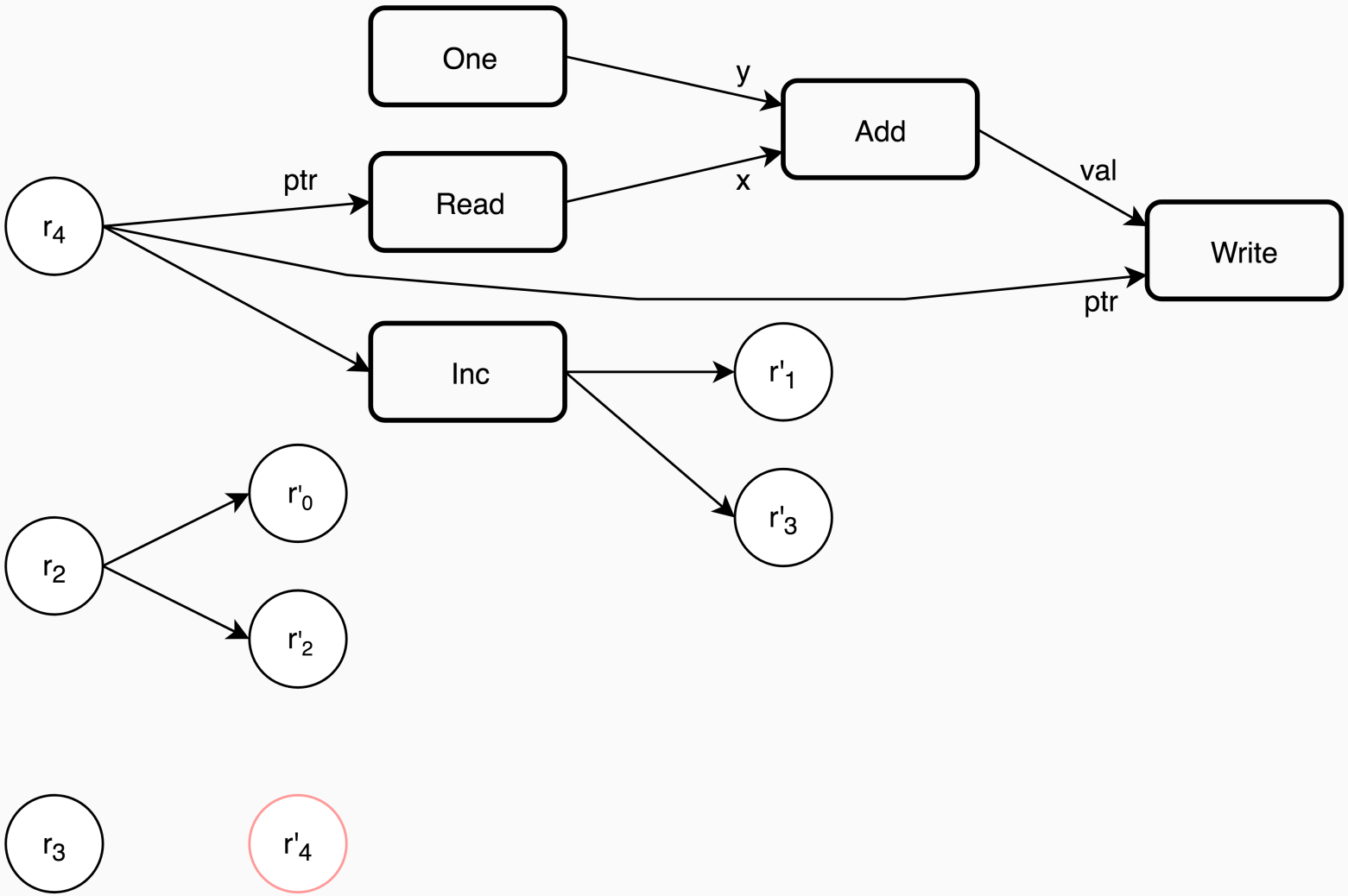
Timestep execution - Circuit execution example



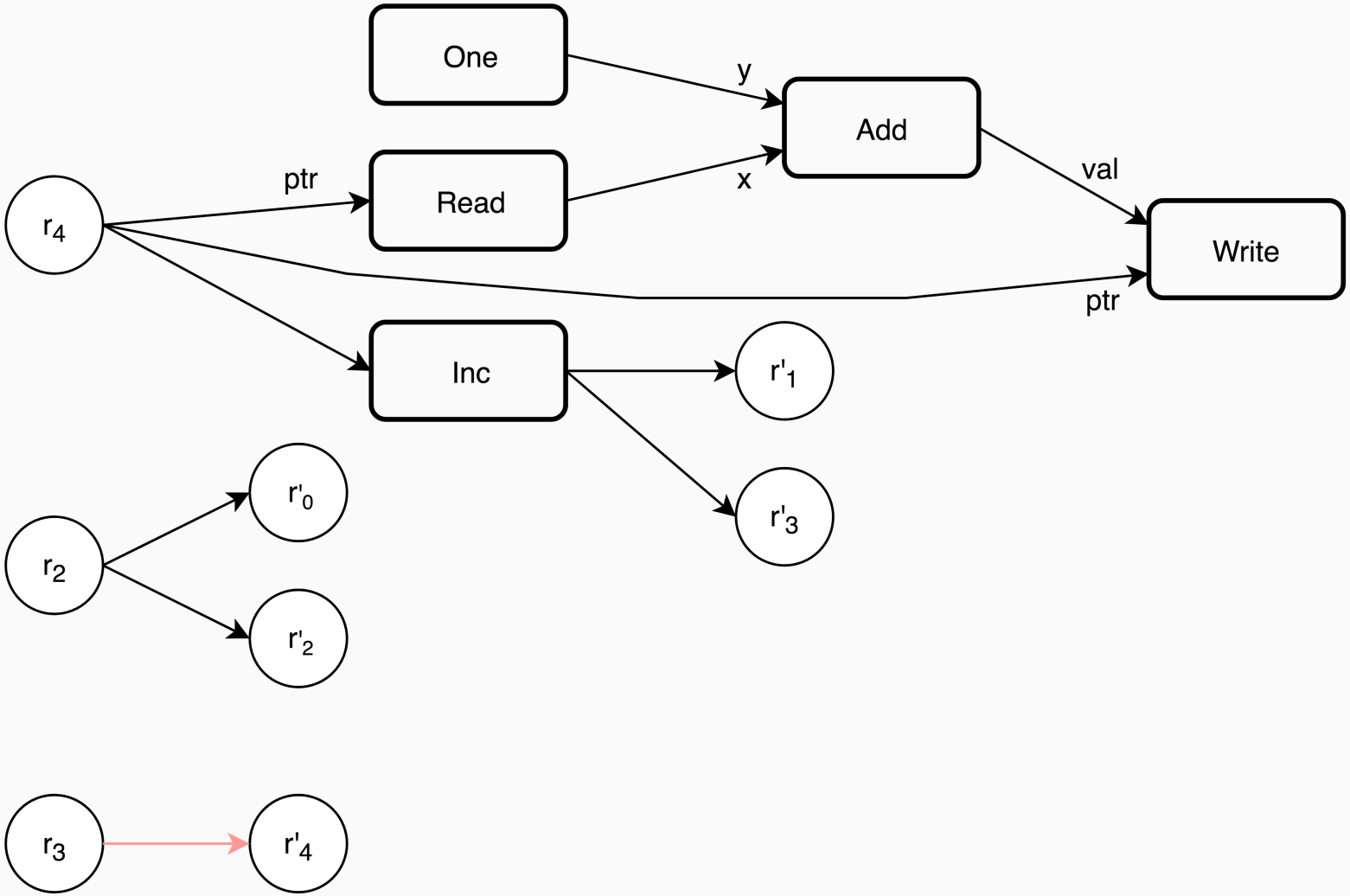
Timestep execution - Circuit execution example



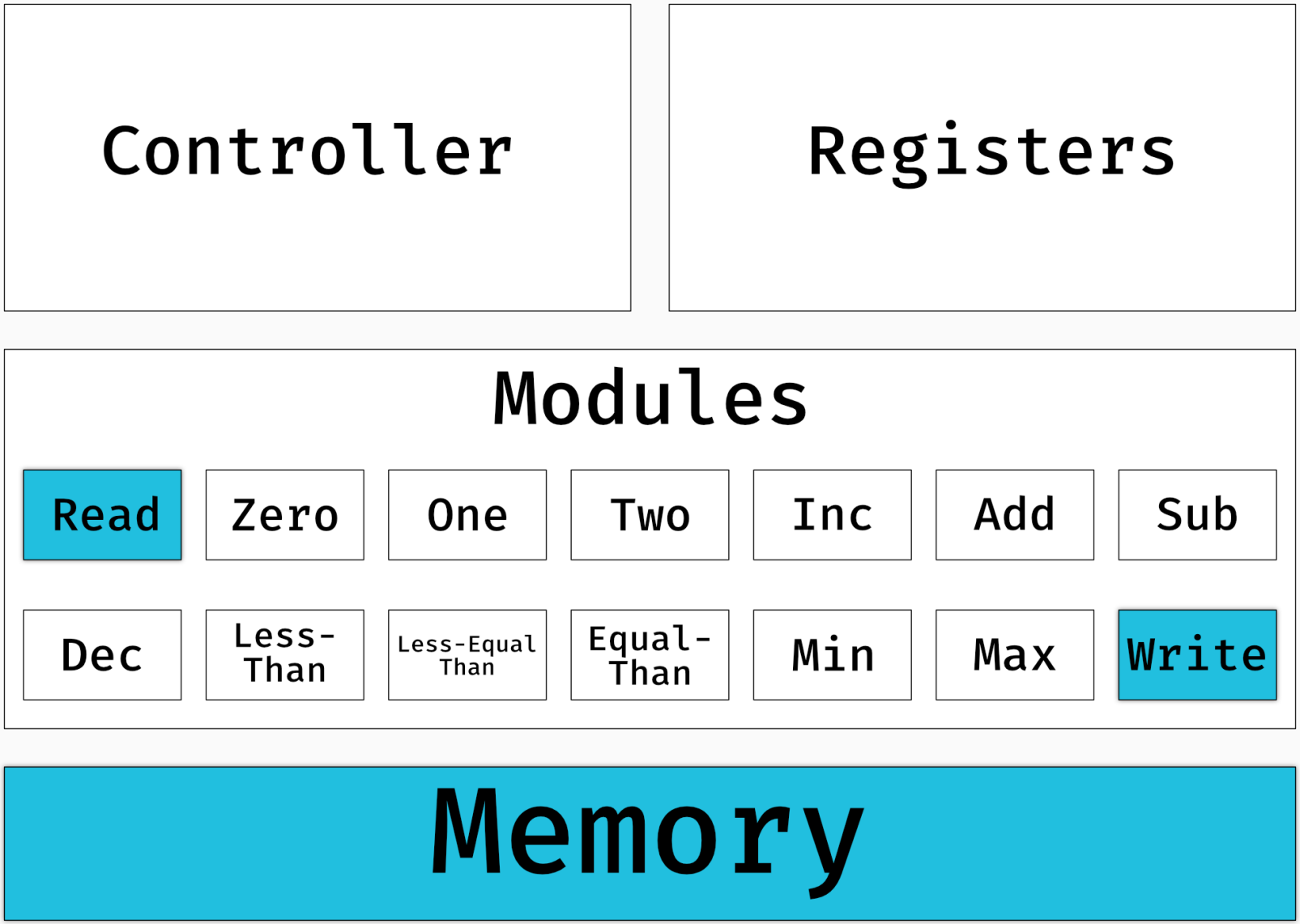
Timestep execution - Circuit execution example



Timestep execution - Circuit execution example



Memory augmented version



Memory

The memory is a support of the Neural Random-Access Machines of I memory cells, formalized as $\mathcal{M} \in \mathbb{R}^{I}_{J}$. Every value $\mathcal{M}_{i,j}$ is the probability that the i^{th} cell contains the j^{th} integer value of the set N.

Read & Write modules

To interact with the memory, the NRAM has two further modules:

- **Read**: takes a pointer and returns $\mathcal{M}^T p$. In other words, the value(s) pointed by the pointer.

Read & Write modules

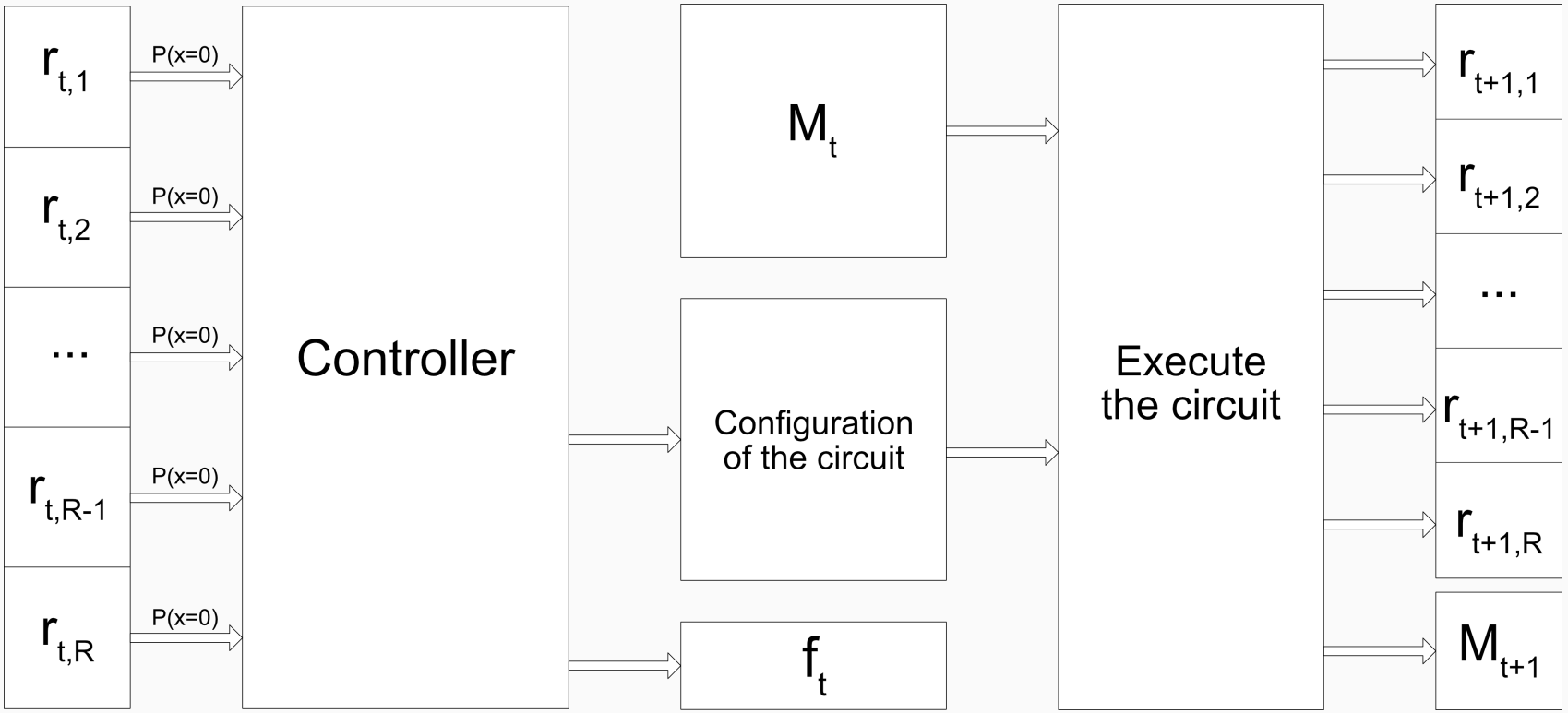
To interact with the memory, the NRAM has two further modules:

- **Read**: takes a pointer and returns $\mathcal{M}^T p$. In other words, the value(s) pointed by the pointer.
- **Write**: takes a pointer and a value, returns zero. Behave as follows:

  $$\mathcal{M} = (J - p)J^T \cdot M + pa^T$$

  where $J \in 1^I$ and $\cdot$ is the coordinate-wise multiplication.

Timestep execution



Termination of NRAM

The NRAM could terminate its execution in two ways:

- Reaching the last timestep T

Termination of NRAM

The NRAM could terminate its execution in two ways:

- Reaching the last timestep T
- Through its internal system of termination

Internal system of termination

NRAM uses a novel way of termination. In each timestep emits along the circuit configuration an $f_t = \sigma(x_i)$ which represents the willingness of terminating the execution. Although not specified in [Kurach et al., 2015], we set the termination if $f_t \geq 1.0$.

Cost calculation

Starting from f_t, the probability that the execution is not finished in the previous timestep is $\prod_{i=1}^{t-1}(1-f_i)$, from which are computed

- the probability that the execution is produced in the current timestep

$$p_t = f_t \cdot \prod_{i=1}^{t-1}(1-f_i)$$

Cost calculation

Starting from f_t, the probability that the execution is not finished in the previous timestep is $\prod_{i=1}^{t-1}(1-f_i)$, from which are computed

- the probability that the execution is produced in the current timestep

$$p_t = f_t \cdot \prod_{i=1}^{t-1}(1-f_i)$$

- the probability that the execution is produced in the last timestep

$$p_T = 1 - \sum_{i=1}^{T-1} p_i$$

Cost calculation

Let $\mathcal{M} \in \mathbb{R}_l^l$ the memory and $\mathcal{EM} \in N^l$, the expected negative log-likelihood is used as cost function and is defined as follows

$$-\sum_{i=1}^{T}\left(p_t \cdot \sum_{i=1}^{|N|} log\left(\mathcal{M}_{i,\mathcal{EM}_i}^{(t)}\right)\right)$$

Cost calculation - Example

Let $t = 0$, $I = 4$ and $p_0 = 0.5$, we might have

Expected Memory

2	0	3	1

Memory

0.3	0.2	0.1	0.2
0.7	0.1	0.0	0.2
0.2	0.2	0.6	0.0
0.1	0.1	0.1	0.7

$$= -(0.5(\log(0.1) + \log(0.7) + \log(0.0 + \epsilon) + \log(0.1)) + \dots)$$

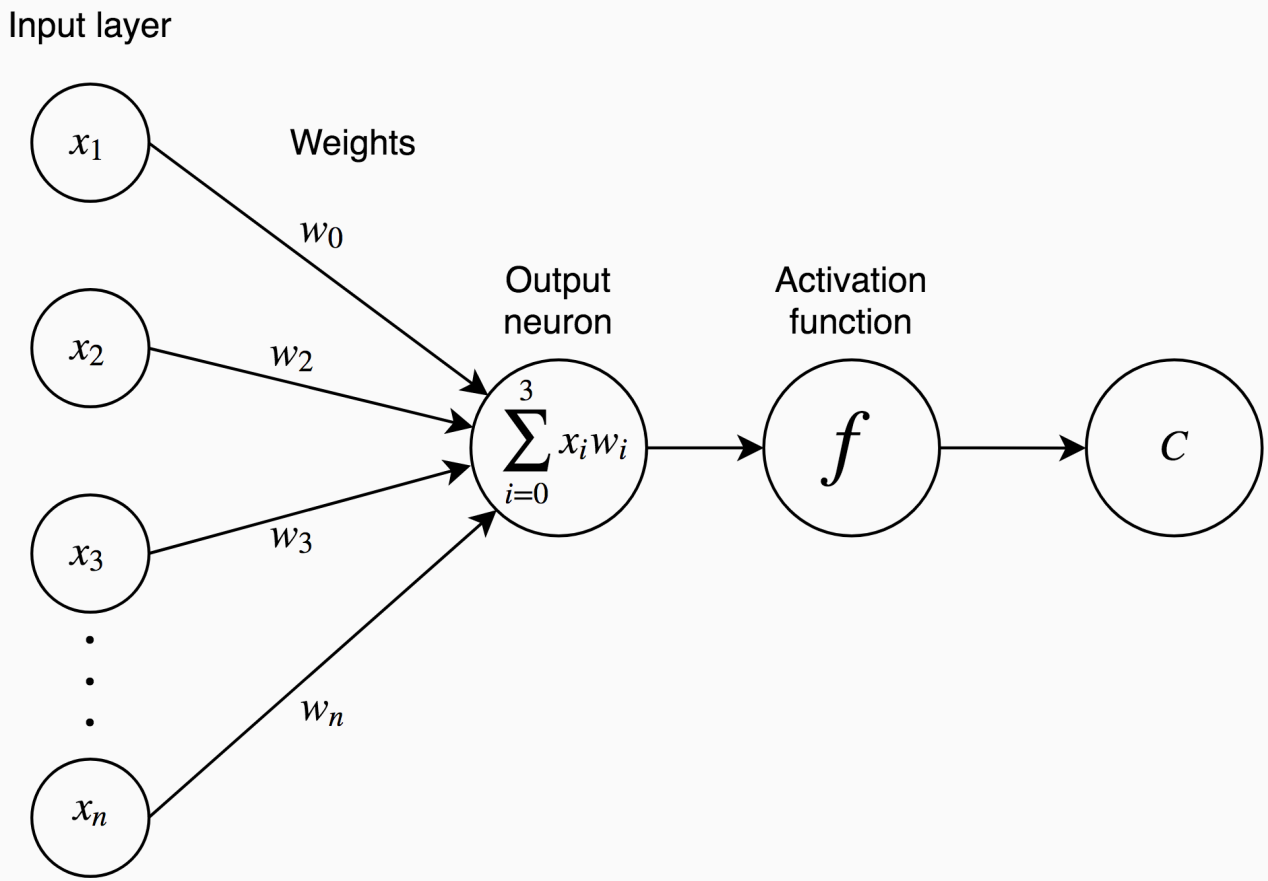
Artificial Neural Network

What are?

The Artificial Neural Network are computing systems inspired by biology and belonging to classification techniques set. Classification systems learn patterns by view examples, i.e. approximate a function f that maps example x to the label y, where $(x, y) \in D$.

Perceptron

The most simple and old ANN [Rosenblatt, 1958] is the Perceptron, formed as follows



Perceptron objective

The Perceptron objective is to divide the examples in an hyperplane

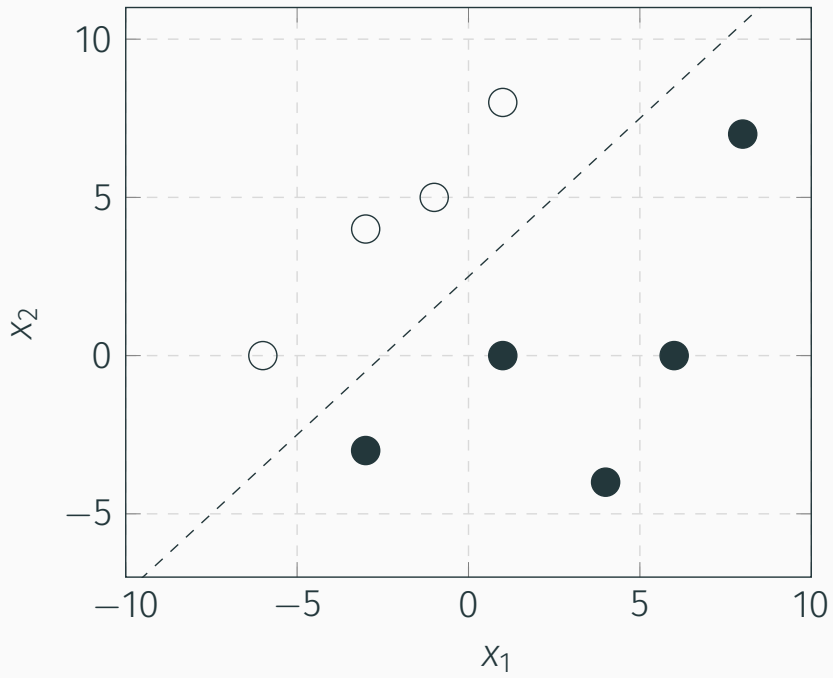
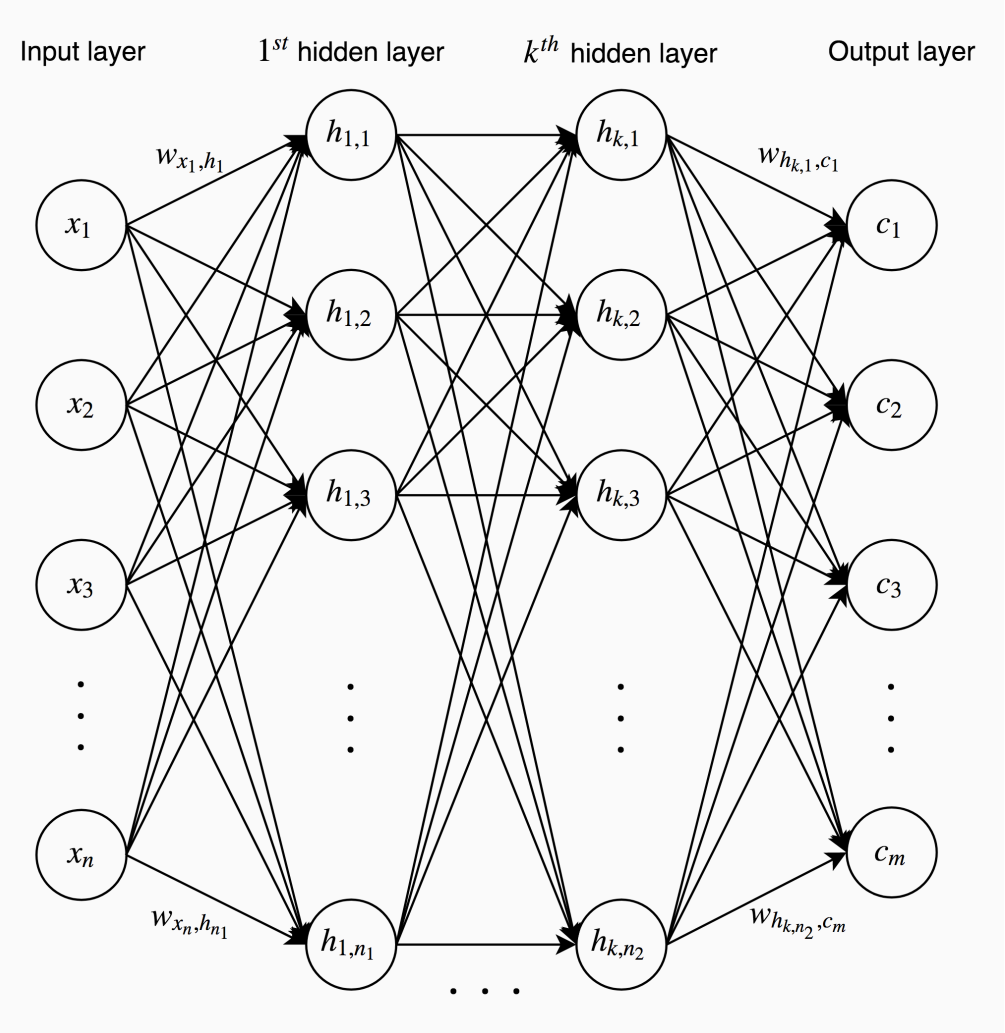


Figure 1: Example with only two features, x_1 and x_2, and two classes, represented by the black and white circles.

Multi Layer Perceptron (MLP)

An evolution of the Perceptron is the Multi Layer Perceptron, formed as follows



Training of an ANN

The training of an ANN follows two steps:

- The initialization of the weights/parameters

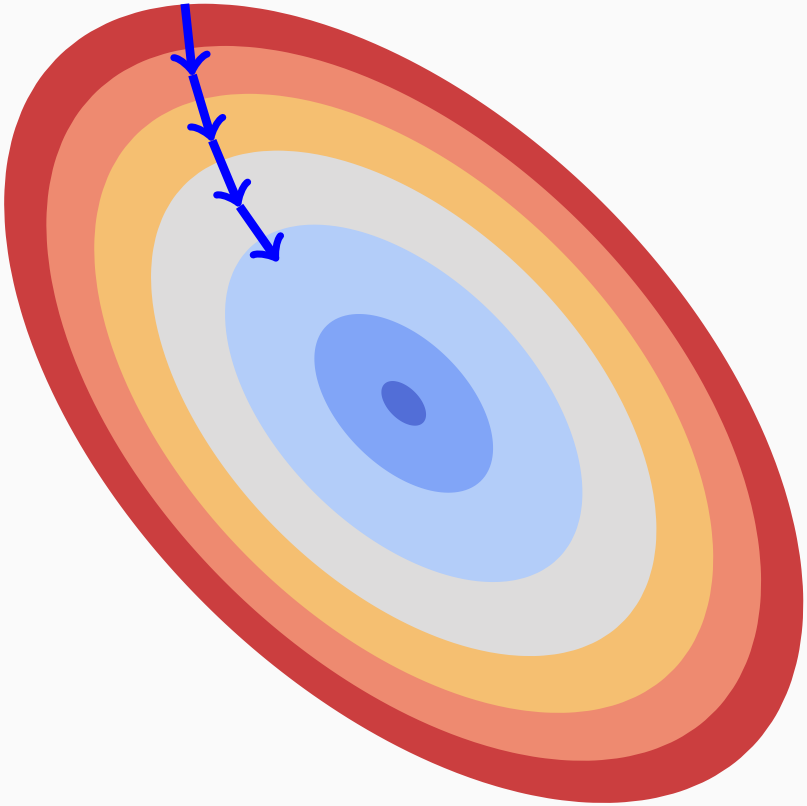
Training of an ANN

The training of an ANN follows two steps:

- The initialization of the weights/parameters
- The re-computation of these searching to minimize the value of an objective function

Gradient Descent & back-propagation

One method to search a optimal/sub-optimal configuration of the parameters through the minimization of the cost function through the Gradient Descent and the back-propagation, with a behaviour similar to the descent of a bowl by an hypothetical ball, where the bowl bottom corresponds to the global minimum of the function.



Back-propagation

The search of those minima is done firstly through the computing of the ANN gradient through the back-propagation, which it is divided into two phases:

- Forward propagation

Back-propagation

The search of those minima is done firstly through the computing of the ANN gradient through the back-propagation, which it is divided into two phases:

- Forward propagation
- Backward propagation

Forward phase

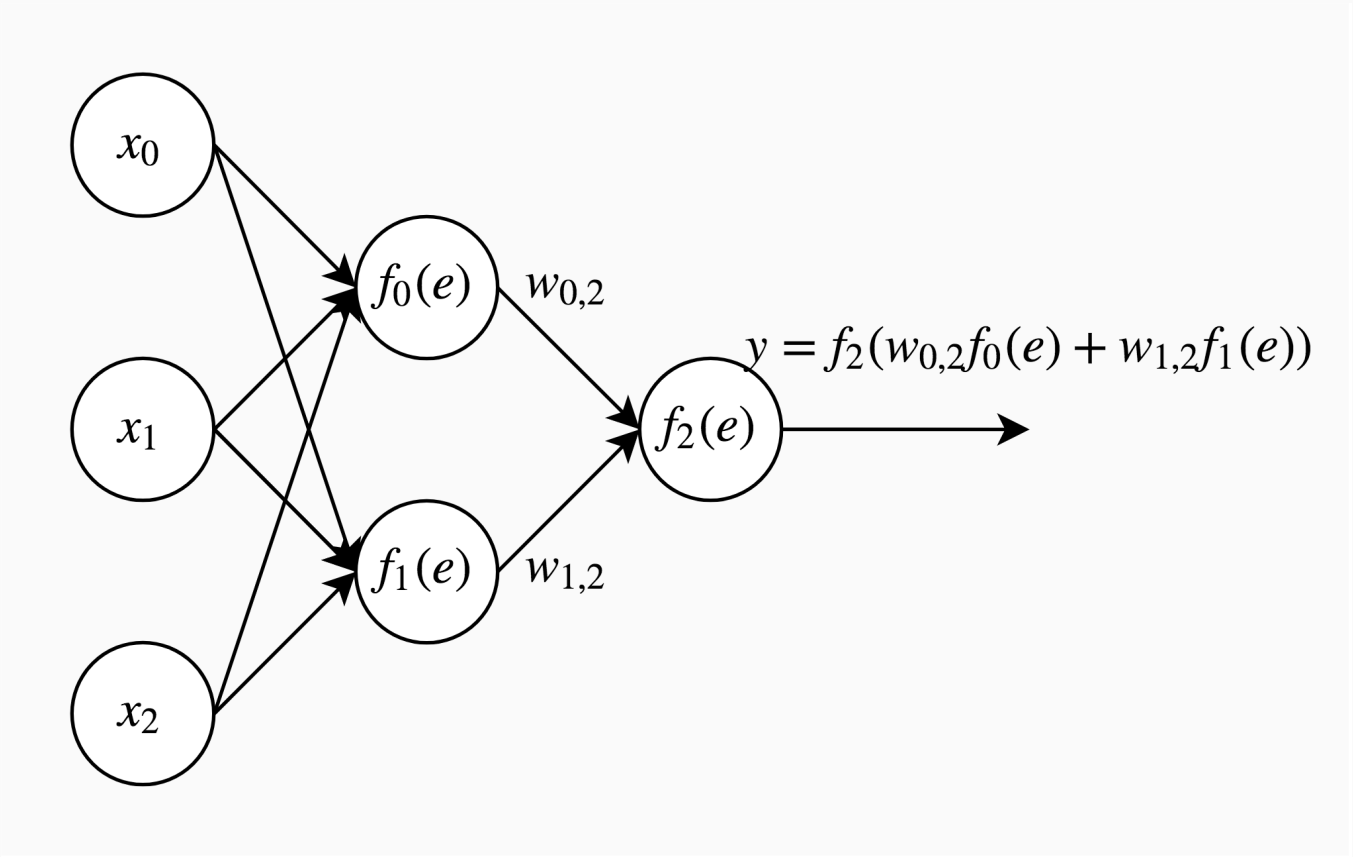


Figure 2: Last pass of forward phase, when the output of ANN is computed

Backpropagation phase

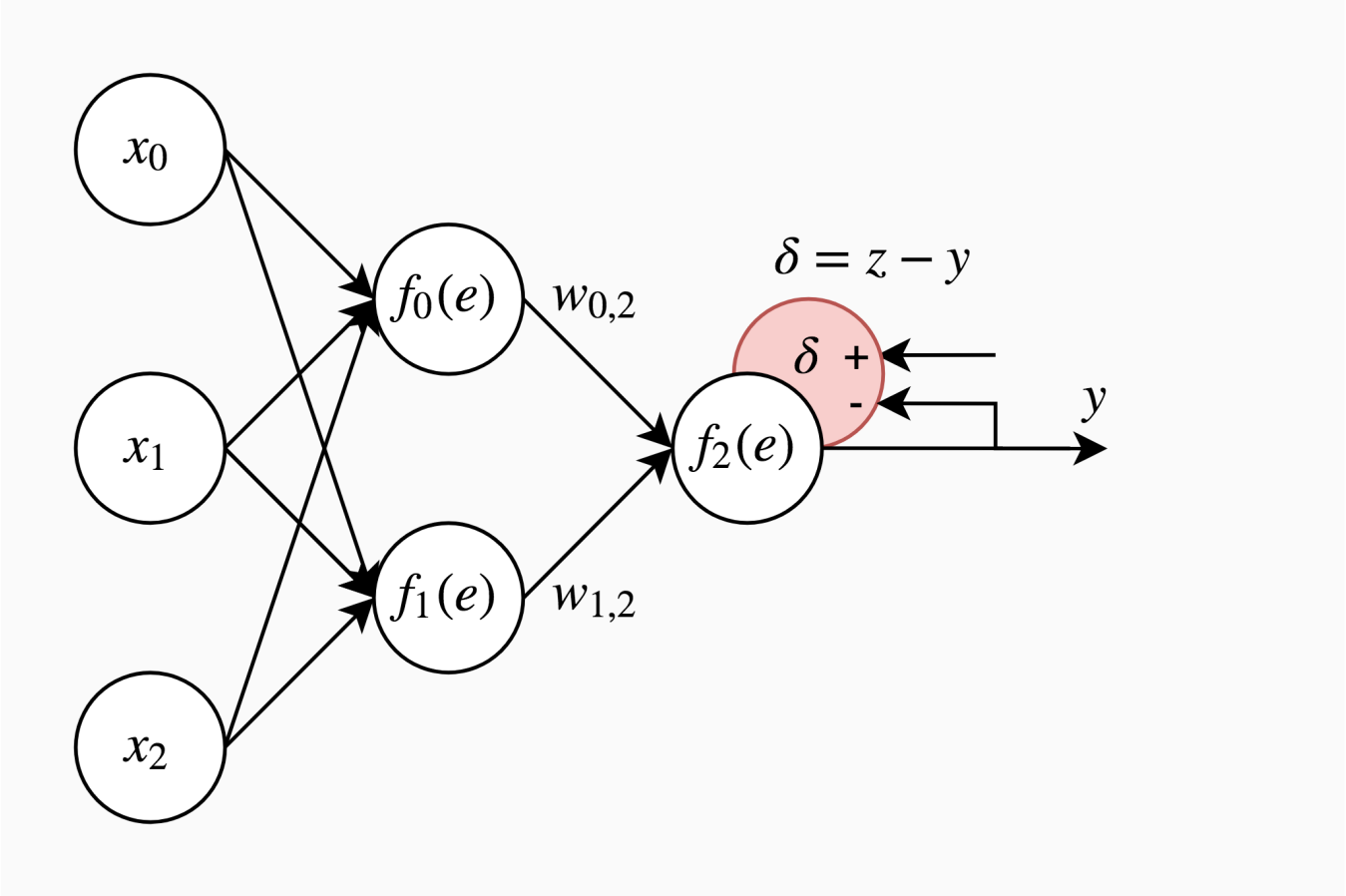


Figure 3: Computing of δ. It can be computed directly because is available a class that can be compared to the ANN exit.

Backpropagation phase

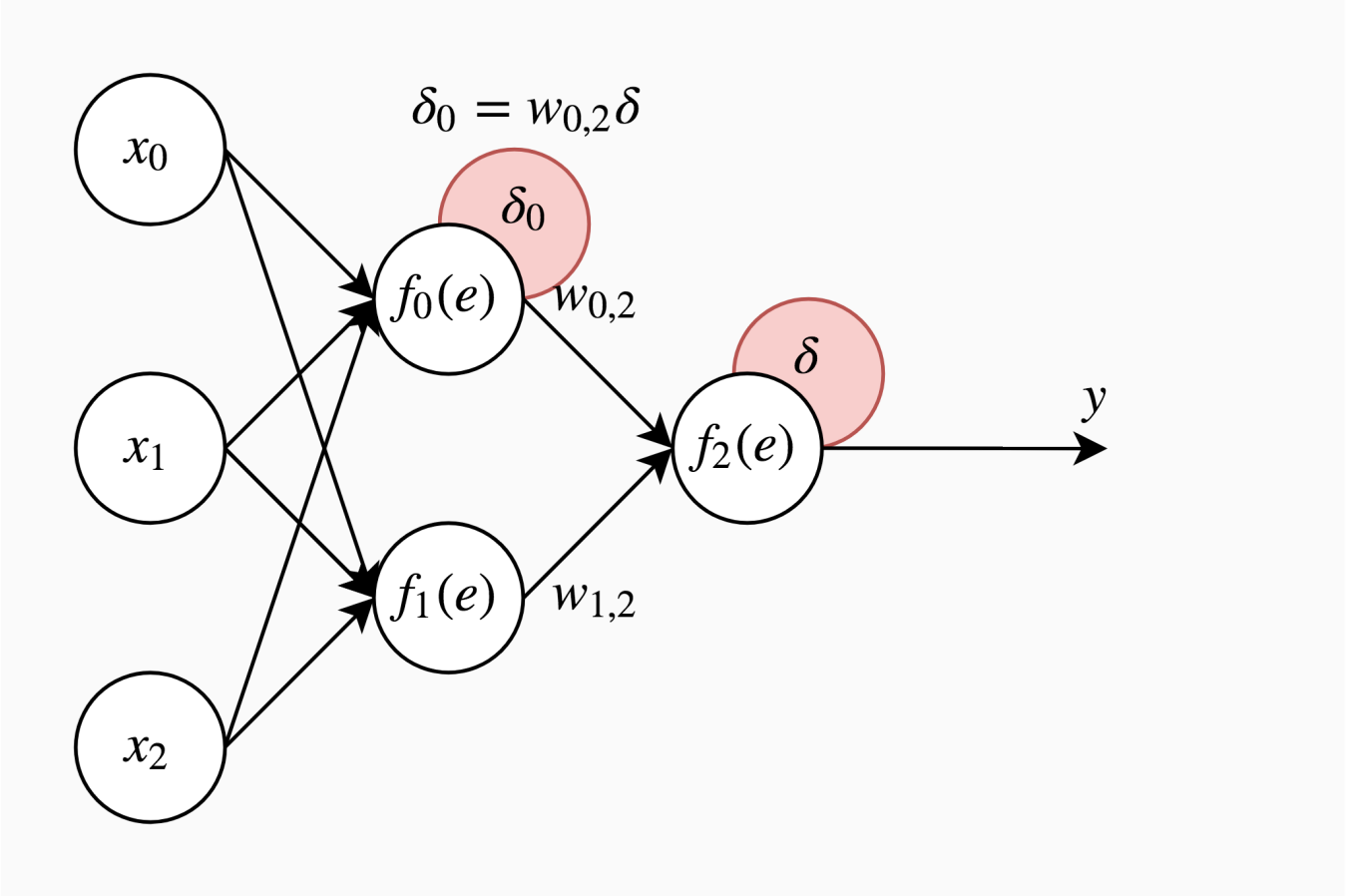


Figure 4: Computing of δ_0 using δ.

Backpropagation phase

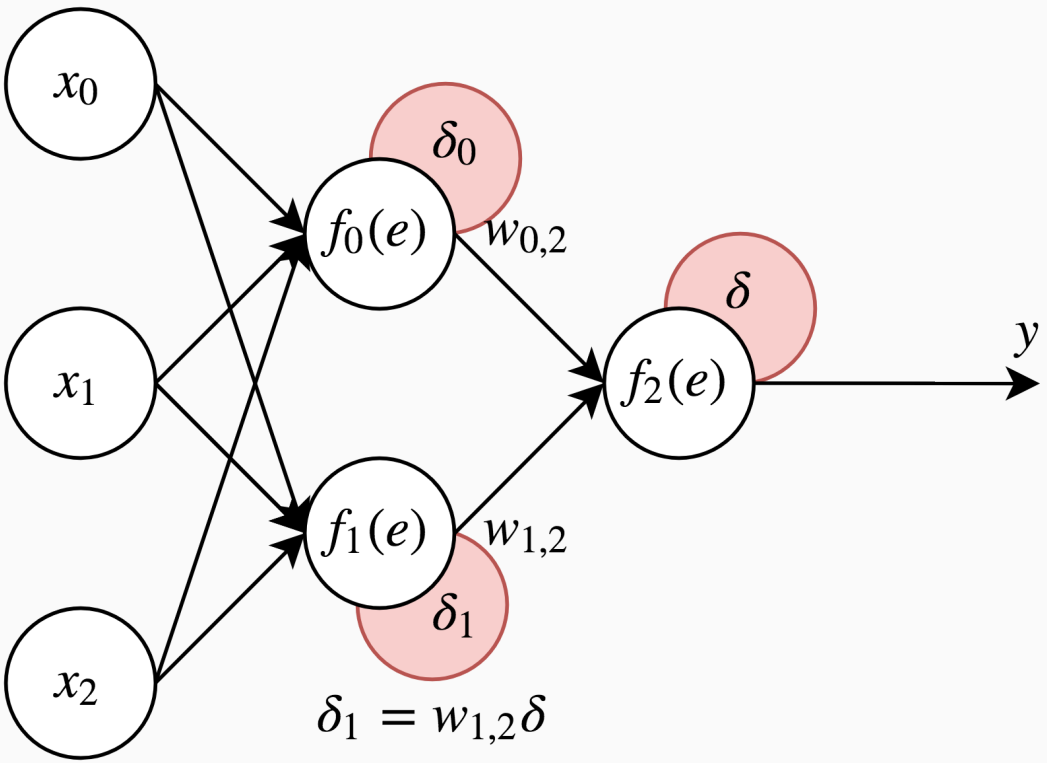


Figure 5: Computing of δ_1 using δ.

Gradient Descent & ADAM

ADAM is a gradient-based optimization algorithm that uses the concept of momentum to regularize the descent of the gradient.

ADAM

Let β_1 and β_2 two exponential decay rates used for the momentum estimates, ϵ a small real-valued scalar, λ the learning rate and w_{ij} the reference weight. The steps to update a parameter are the following

- **Compute** $m_{ij}^{(t)}$: $m_{ij}^{(t)} = \beta_1 m_{ij}^{(t-1)} + (1 - \beta_1)\delta_{ij}^{(t)}$

ADAM

Let β_1 and β_2 two exponential decay rates used for the momentum estimates, ϵ a small real-valued scalar, λ the learning rate and w_{ij} the reference weight. The steps to update a parameter are the following

- **Compute** $m_{ij}^{(t)}$: $m_{ij}^{(t)} = \beta_1 m_{ij}^{(t-1)} + (1 - \beta_1)\delta_{ij}^{(t)}$
- **Compute** $v_{ij}^{(t)}$: $v_{ij}^{(t)} = \beta_2 v_{ij}^{(t-1)} + (1 - \beta_2)\delta_{ij}^{2,(t)}$

ADAM

Let β_1 and β_2 two exponential decay rates used for the momentum estimates, ϵ a small real-valued scalar, λ the learning rate and w_{ij} the reference weight. The steps to update a parameter are the following

- **Compute** $m_{ij}^{(t)}$: $m_{ij}^{(t)} = \beta_1 m_{ij}^{(t-1)} + (1 - \beta_1)\delta_{ij}^{(t)}$
- **Compute** $v_{ij}^{(t)}$: $v_{ij}^{(t)} = \beta_2 v_{ij}^{(t-1)} + (1 - \beta_2)\delta_{ij}^{2,(t)}$
- **Regularize** $m_{ij}^{(t)}$ **w.r.t.** β_1: $\hat{m}_{ij}^{(t)} = \frac{m_{ij}^{(t)}}{1-\beta_1}$

ADAM

Let β_1 and β_2 two exponential decay rates used for the momentum estimates, ϵ a small real-valued scalar, λ the learning rate and w_{ij} the reference weight. The steps to update a parameter are the following

- **Compute** $m_{ij}^{(t)}$: $m_{ij}^{(t)} = \beta_1 m_{ij}^{(t-1)} + (1 - \beta_1)\delta_{ij}^{(t)}$
- **Compute** $v_{ij}^{(t)}$: $v_{ij}^{(t)} = \beta_2 v_{ij}^{(t-1)} + (1 - \beta_2)\delta_{ij}^{2,(t)}$
- **Regularize** $m_{ij}^{(t)}$ **w.r.t.** β_1: $\hat{m}_{ij}^{(t)} = \frac{m_{ij}^{(t)}}{1-\beta_1}$
- **Regularize** $v_{ij}^{(t)}$ **w.r.t.** β_2: $\hat{m}_{ij}^{(t)} = \frac{m_{ij}^{(t)}}{1-\beta_1}$

ADAM

Let β_1 and β_2 two exponential decay rates used for the momentum estimates, ϵ a small real-valued scalar, λ the learning rate and w_{ij} the reference weight. The steps to update a parameter are the following

- **Compute** $m_{ij}^{(t)}$: $m_{ij}^{(t)} = \beta_1 m_{ij}^{(t-1)} + (1-\beta_1)\delta_{ij}^{(t)}$
- **Compute** $v_{ij}^{(t)}$: $v_{ij}^{(t)} = \beta_2 v_{ij}^{(t-1)} + (1-\beta_2)\delta_{ij}^{2,(t)}$
- **Regularize** $m_{ij}^{(t)}$ **w.r.t.** β_1: $\hat{m}_{ij}^{(t)} = \frac{m_{ij}^{(t)}}{1-\beta_1}$
- **Regularize** $v_{ij}^{(t)}$ **w.r.t.** β_2: $\hat{m}_{ij}^{(t)} = \frac{m_{ij}^{(t)}}{1-\beta_1}$
- **Update** $w_{ij}^{(t)}$: $w_{ij}^{(t)} = w_{ij}^{(t)} - \lambda \frac{\hat{m}_{ij}^{(t)}}{\sqrt{\hat{v}_{ij}^{(t)}}+\epsilon}$

Differential Evolution & DENN

Description

Differential Evolution (DE) is a metaheuristic introduced in [**?**], belonging to the family of Evolutionary Algorithms (EAs), that has as objective the searching of a solution through the parallel evolution of a set of candidate solutions.

This set is defined as **population** and it is composed by NP D-dimensional numerical vectors where each of them is called individual, chromosome or genoma.

Description

Its functioning is iterative and after a first phase of initialization of the individuals the following step is executed

- **Mutation**

Description

Its functioning is iterative and after a first phase of initialization of the individuals the following step is executed

- Mutation
- Crossover

Description

Its functioning is iterative and after a first phase of initialization of the individuals the following step is executed

- Mutation
- Crossover
- Selection

Mutation

Through the mutation phase a new population, called donor set, is created. Let x an individual at the generation G, also called target. A donor is generated combining x with some other individuals through a differential mutation operator.

Mutation - Example

For example, the method **rand/1** work as follows

$$v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{x_3,G})$$

where $r_1, r_2, r_3 \in \{1, 2, \ldots, NP\}$ are mutually exclusive indices and F is a real-valued constant defined by the user.

Crossover

The Crossover phase does nothing else than mixing up one-by-one the donor vectors components with the target vectors with some crossover methods, generating the **trials** vectors set and enhancing the potential diversity of the population (a better exploration).

Crossover - Example

For example, let the target vector $x_{i,G}$ and the mutant $v_{i,G}$ the **bin** strategy work as follows

$$u_{ji,G} = \begin{cases} v_{ji,G}, & \text{if } (randb(j) \leq CR) \text{ or } j = rnbr(i) \\ x_{ji,G}, & \text{if } (randb(j) > CR) \text{ and } j \neq rnbr(i) \end{cases} i = 1, 2, \ldots, D \tag{4}$$

where $randb(j)$ is a function which generate a real-valued number for the j^{th} parameter according to binomial distribution, $\mathrm{CR} \in [0, 1]$ is the global user-defined crossover constant used as a threshold and $rnbr(i)$ is a function that generate a random index which ensures that is selected at least one parameter of the mutant v_i.

Selection

Each trial is compared one-by-one to the corresponding target using the fitness function - if the target have a smaller cost with respect to the trial, than it is retained as individual of the population of the next generation and vice-versa.

Differential Evolution variants

There exist various DE variants; some of these that are used in the thesis are

- **JADE**: generate for each target x_i, F_i and CR_i with an adaptation system which is based on two memories of *infinite* size called S_{F} and S_{CR};

Differential Evolution variants

There exist various DE variants; some of these that are used in the thesis are

- **JADE**: generate for each target x_i, F_i and CR_i with an adaptation system which is based on two memories of *infinite* size called S_{F} and S_{CR};
- **SHADE**: generate for each target x_i, F_i and CR_i with an adaptation system which is based on two memories of *finite* size M_{F} and M_{CR} called *historical memories*;

Differential Evolution variants

There exist various DE variants; some of these that are used in the thesis are

- **JADE**: generate for each target x_i, F_i and CR_i with an adaptation system which is based on two memories of *infinite* size called S_{F} and S_{CR};
- **SHADE**: generate for each target x_i, F_i and CR_i with an adaptation system which is based on two memories of *finite* size M_{F} and M_{CR} called *historical memories*;
- **L-SHADE**: similar to SHADE, but bases is function also in a linear reduction of the population.

Mutation variants

Crossover variants

DENN (Differential Evolution for Neural Network)

Implementation and results

Implementation

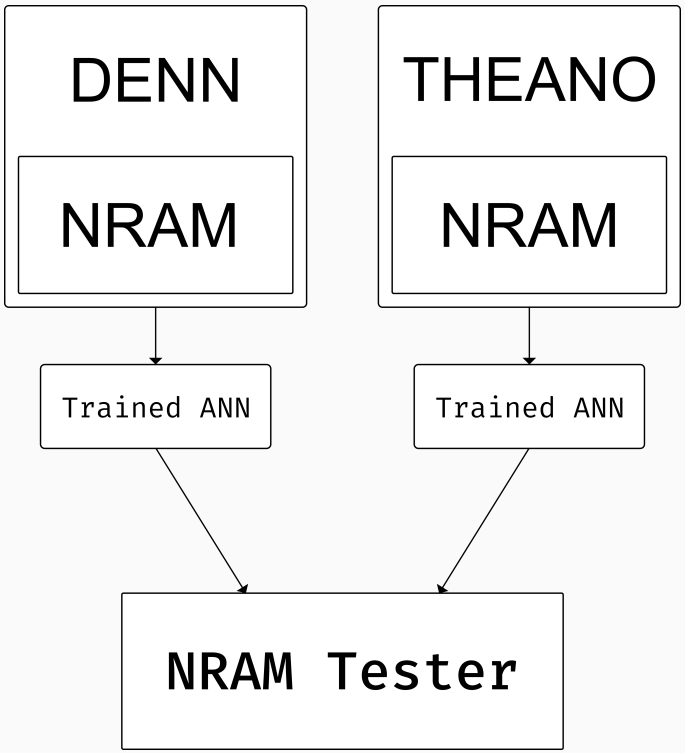


Figure 6: Three implementations were developed: the first two are used to train the controllers and the last is used to test the generalization ability of the discovered controllers.

Datasets

The datasets used during the training are automatically generated at runtime. Each batch of samples is composed by:

- initial memories: the memories which are manipulated by NRAM during the execution;
- expected memories: the memories which are compared to the modified initial memories for cost computation;
- cost masks: used to take into account only parts of the memories during the computing of the cost;
- error rate masks: used to take into account only parts of the memories during the computing of the error rate;

Problems

The following are some of the problems introduced in [Kurach et al., 2015]:

- Access

Problems

The following are some of the problems introduced in [Kurach et al., 2015]:

- Access
- Increment

Problems

The following are some of the problems introduced in [Kurach et al., 2015]:

- Access
- Increment
- Copy

Problems

The following are some of the problems introduced in [Kurach et al., 2015]:

- Access
- Increment
- Copy
- Reverse

Access

Given a value k and an array $\mathbf{A}$, return $\mathbf{A}[k]$. Input is given as $k,\ A[0],\ \ldots,$
$\mathbf{A}[n-1],\ NULL$ and the network should replace the first memory cell with $\mathbf{A}[k]$.

Initial memory									
4	5	1	4	$\underline{7}$	2	8	3	6	0

Desired memory									
7	5	1	4	7	2	8	3	6	0

Cost mask									
1	1	1	1	1	1	1	1	1	1

Error mask									
1	0	0	0	0	0	0	0	0	0

Increment

Given an array **A**, increment all its elements by 1. Input is given as $\mathbf{A}[0], \ldots, \mathbf{A}[n-1], \mathit{NULL}$ and the expected output is $\mathbf{A}[0]+1, \ldots, A[n-1]+1$.

Initial memory									
5	5	9	4	7	8	0	0	0	0

Desired memory									
6	6	0	4	8	9	0	0	0	0

Cost mask									
1	1	1	1	1	1	1	1	1	1

Error mask									
1	1	1	1	1	1	0	0	0	0

Copy

Given an array and a pointer to the destination, copy all elements from the array to the given location. Input is given as $p, \mathsf{A}[0], \ldots, \mathsf{A}[n-1]$ where p points to one element after $\mathsf{A}[n-1]$. The expected output is $\mathsf{A}[0], \ldots, \mathsf{A}[n-1]$ at positions $p, \ldots, p+n-1$ respectively.

Initial memory									
5	5	1	4	7	0	0	0	0	0

Desired memory									
5	5	1	4	7	5	1	4	7	0

Cost mask									
0	1	1	1	1	1	1	1	1	1

Error mask									
0	0	0	0	0	1	1	1	1	0

reverse

Given an array and a pointer to the destination, copy all elements from the array in reversed order. Input is given as $p, \mathsf{A}[0], \ldots, \mathsf{A}[n-1]$ where p points one element after $\mathsf{A}[n-1]$. The expected output is $\mathsf{A}[n-1], \ldots, \mathsf{A}[0]$ at positions $p, \ldots, p+n-1$.

Initial memory									
5	5	1	4	7	0	0	0	0	0

Desired memory									
5	5	1	4	7	7	4	1	5	0

Cost mask									
0	1	1	1	1	1	1	1	1	1

Error mask									
0	0	0	0	0	1	1	1	1	0

Results

References

- Kurach, K., Andrychowicz, M., and Sutskever, I. (2015).
  Neural random-access machines.
  CoRR, abs/1511.06392.
- Rosenblatt, F. (1958).
  The perceptron: a probabilistic model for information storage and organization in the brain.
  Psychological review, 65 6:386–408.