Learning Environment Models in Car Racing using Stateful Genetic Programming

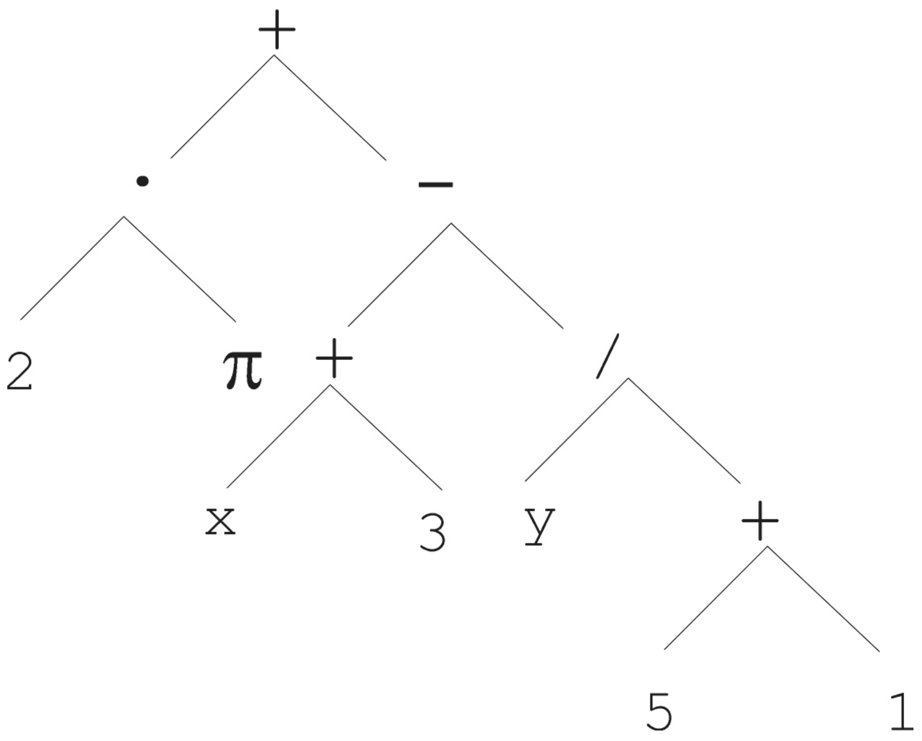
Alexandros Agapitos, Michael O'Neill, Anthony Brabazon
School Of Computer Science and Informatics
University College Dublin
Ireland

UCD Natural Computing Research & Applications Group (NCRA)

Outline

- Memory usage in Genetic Programming
- Method for Building Racing Track Models in TORCS
- Experimental Results
- Conclusions and Future Work

Traditional Expression-tree GP



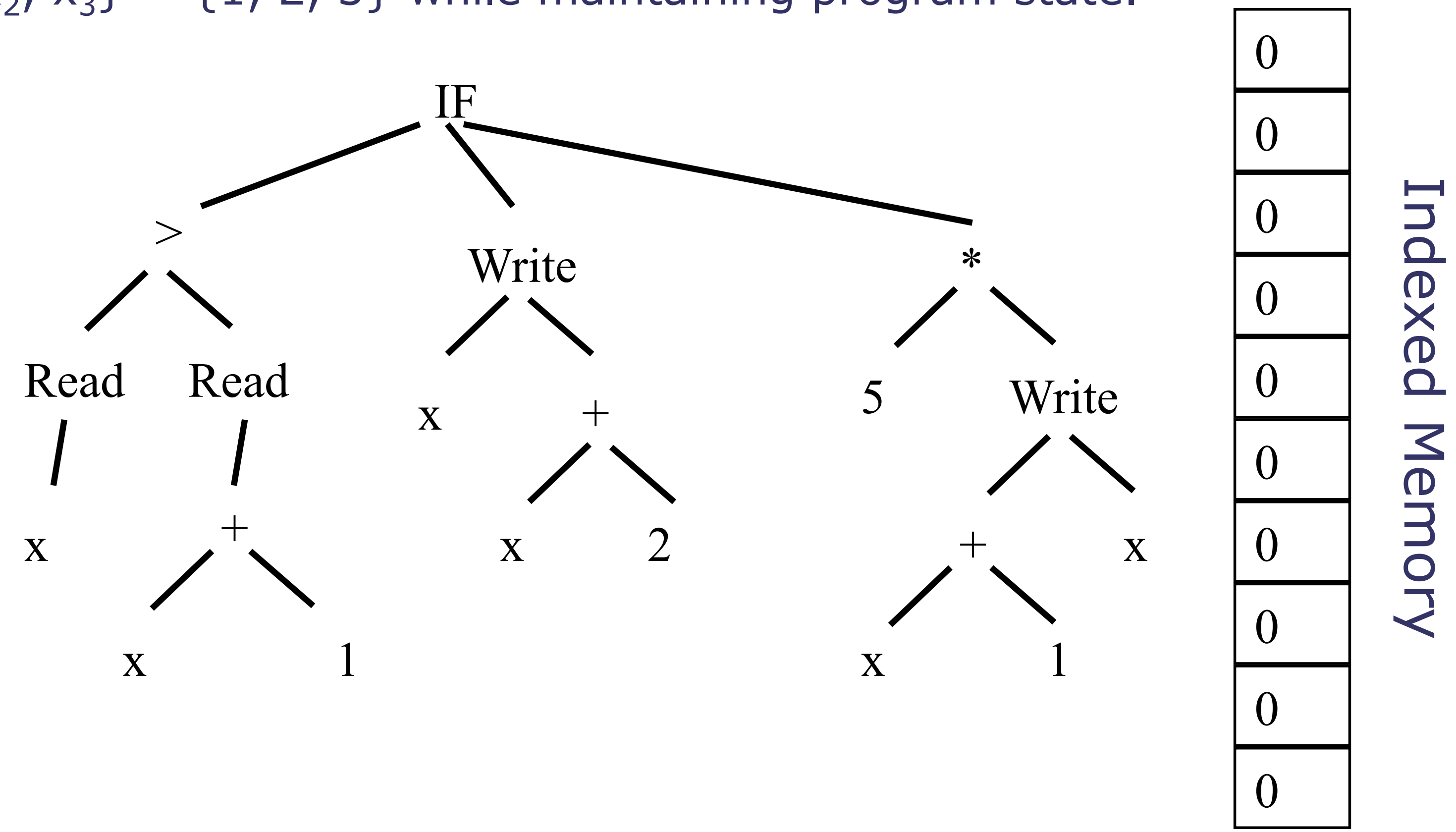
$$2\cdot\pi+\left((x+3)-\frac{y}{5+1}\right)$$

Indexed Memory: A Simple Addition to GP

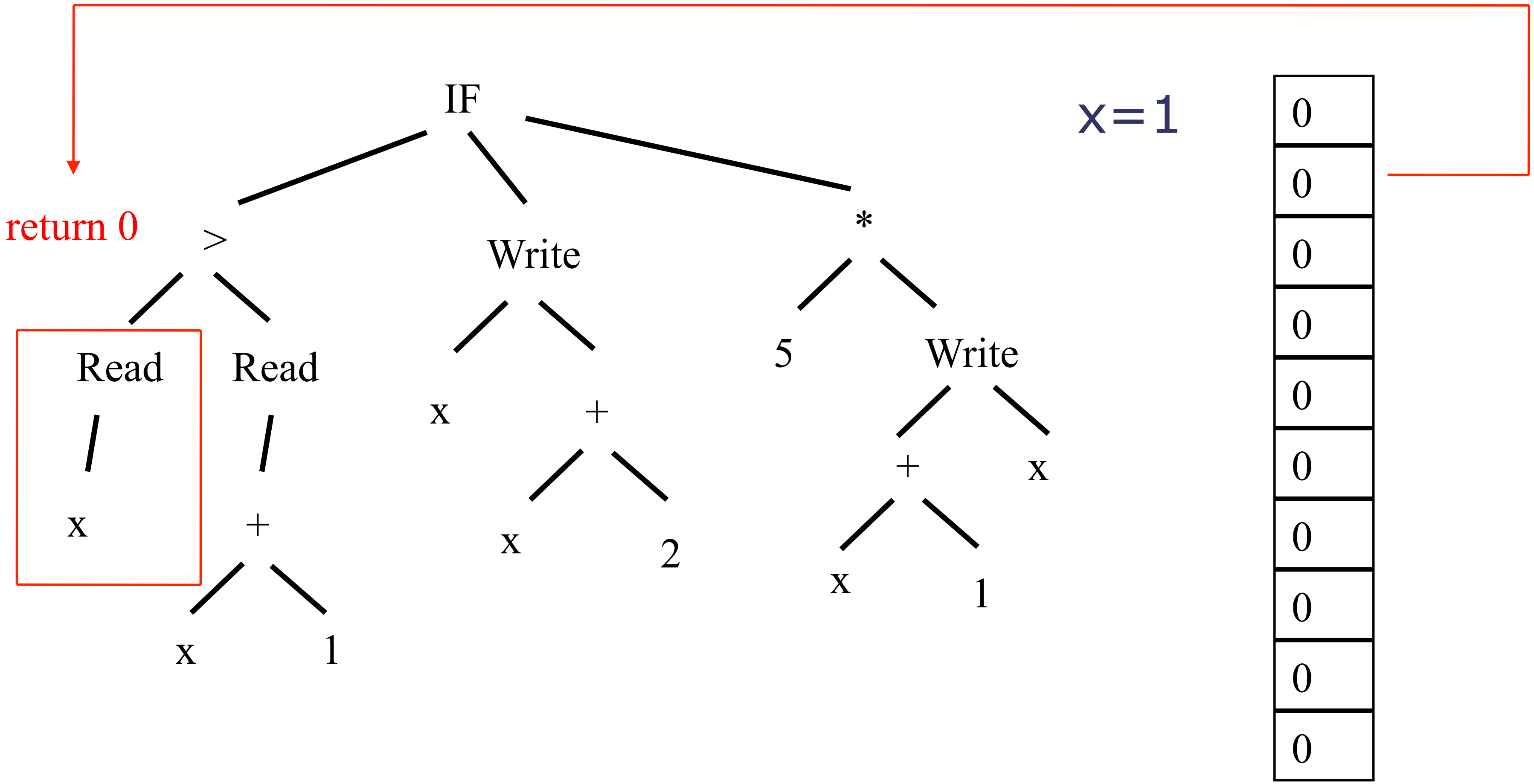
- **Read** and **Write** primitives are added as new non-terminals in the language.
- Each GP expression-tree is given access to its own array of integers, indexed over the integers.
- **Read(X)** returns the value stored in memory position X, where X is of same type as the type of memory elements (i.e. Read(4) returns the fifth element of the array).
- **Write(X, Y)** returns the old value of a memory position X, and has the side effect of changing the value of position X to Y (i.e. Write(10, 104) returns the value of memory position 10 and overwrites it to 104).
- *Program state* refers to the contents of array during program execution.

An example

We wish to execute the program with consecutive input $\{x_1, x_2, x_3\} = \{1, 2, 3\}$ while maintaining program state.

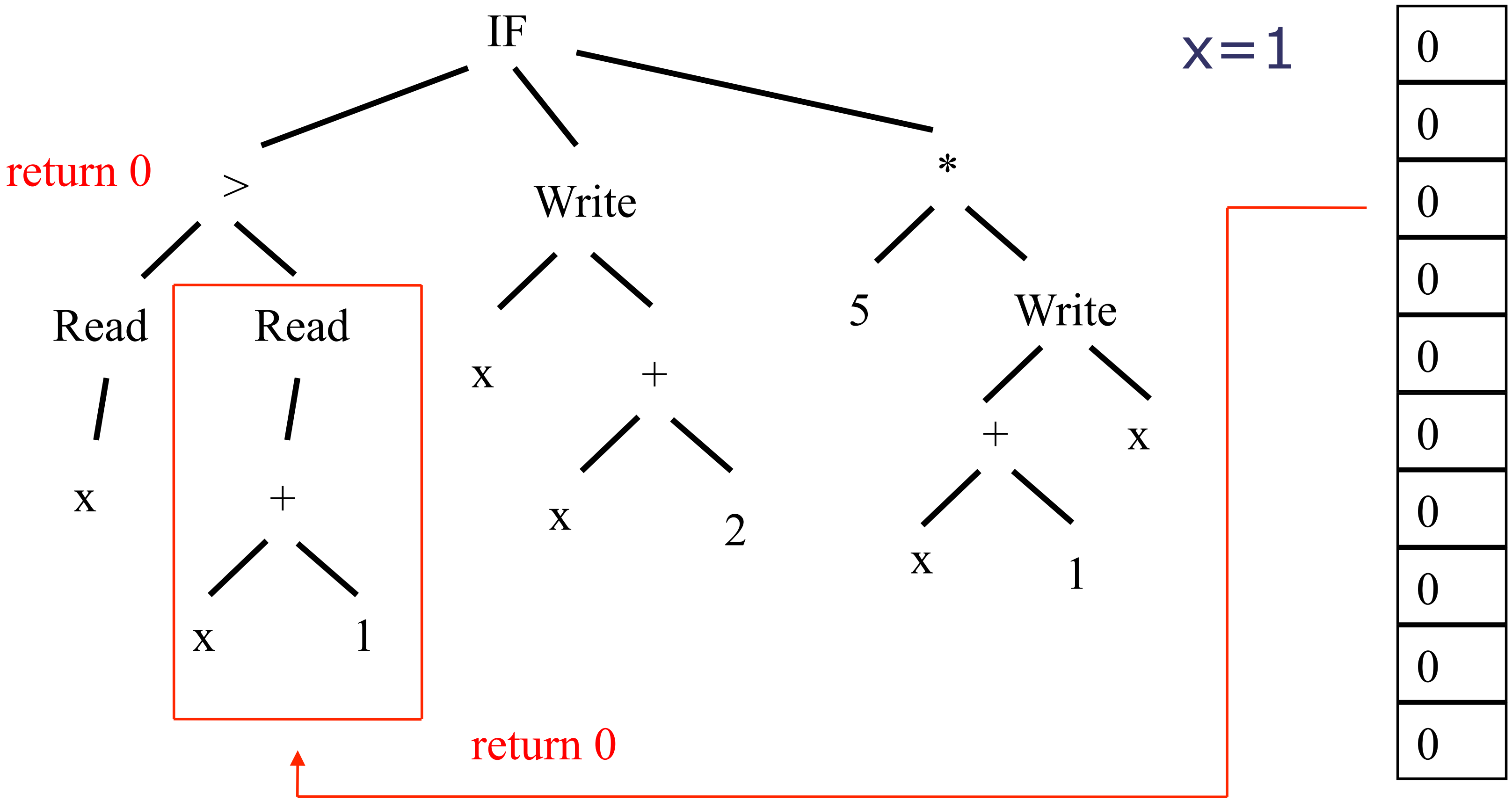


State Maintenance During Program Execution (1)

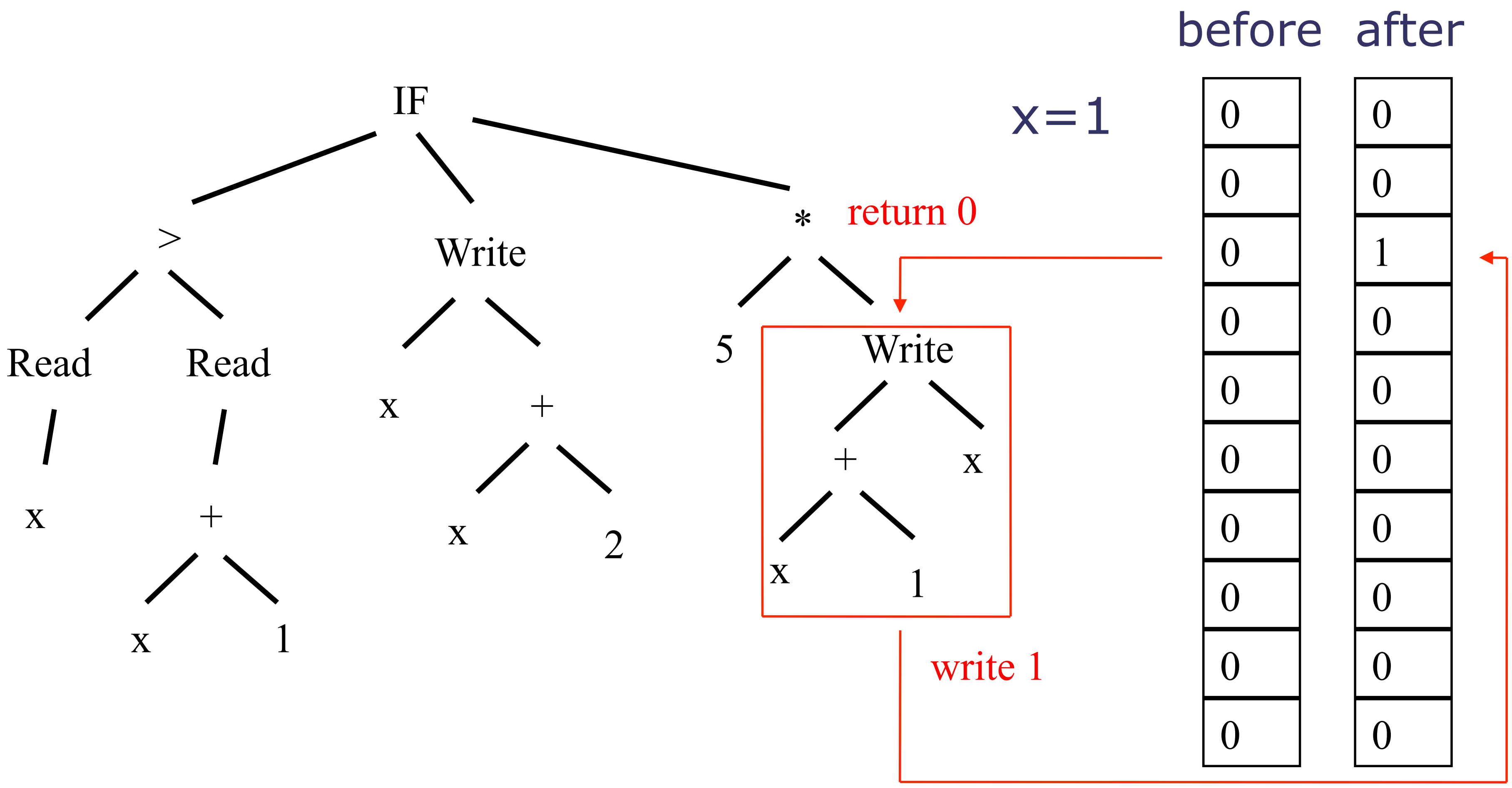


UCD Natural Computing Research & Applications Group (NCRA)

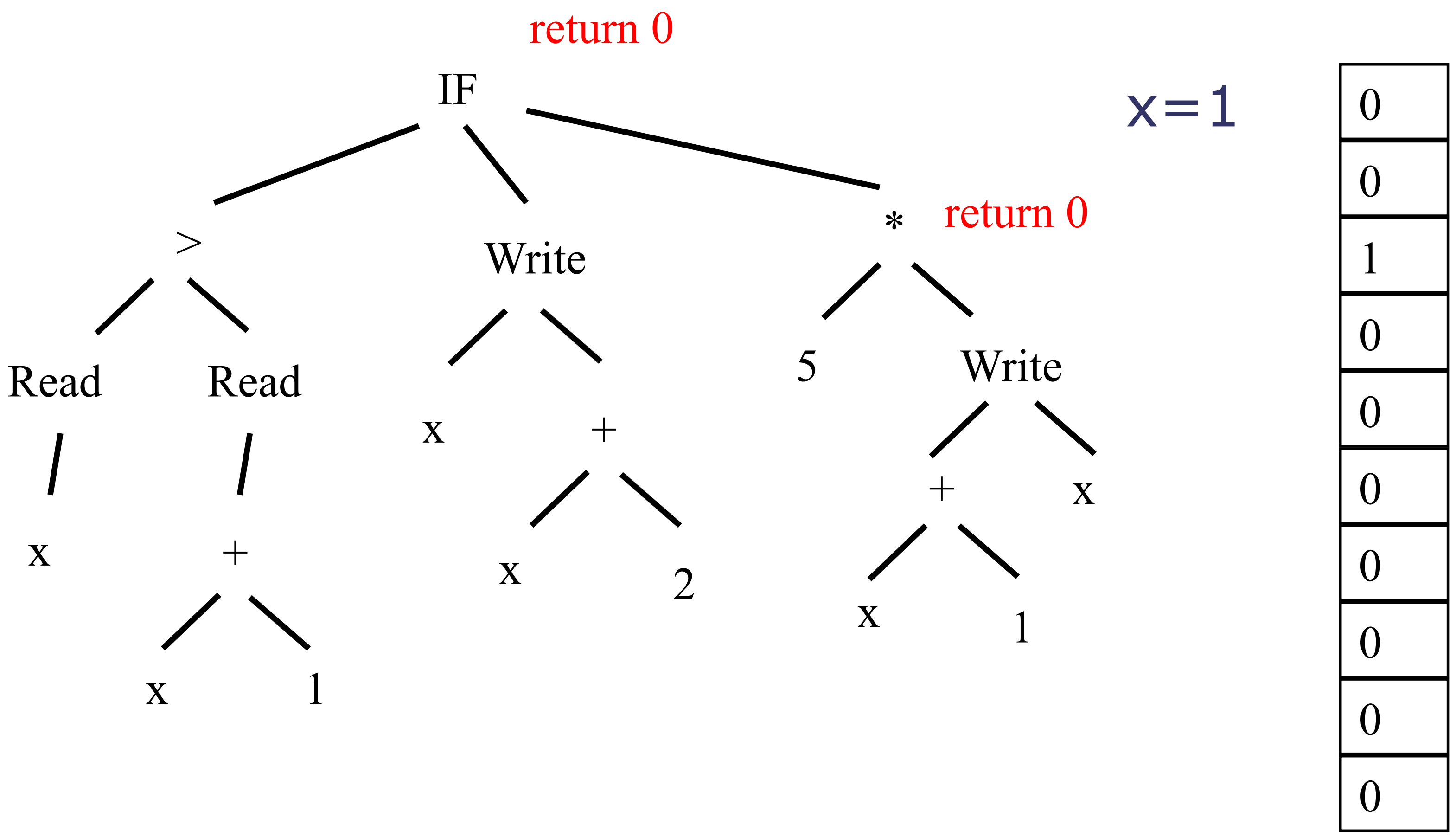
State Maintenance During Program Execution (2)



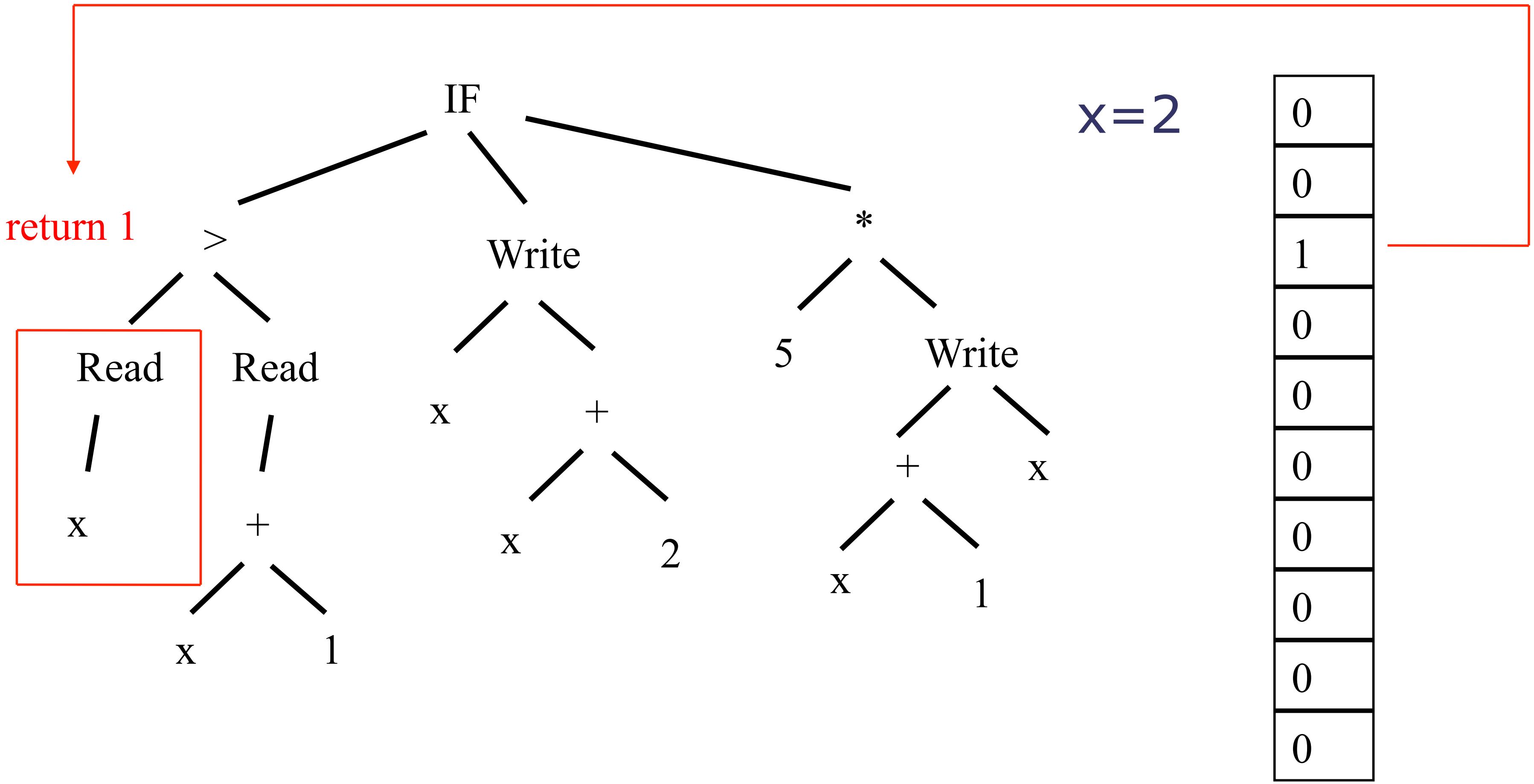
State Maintenance During Program Execution (3)



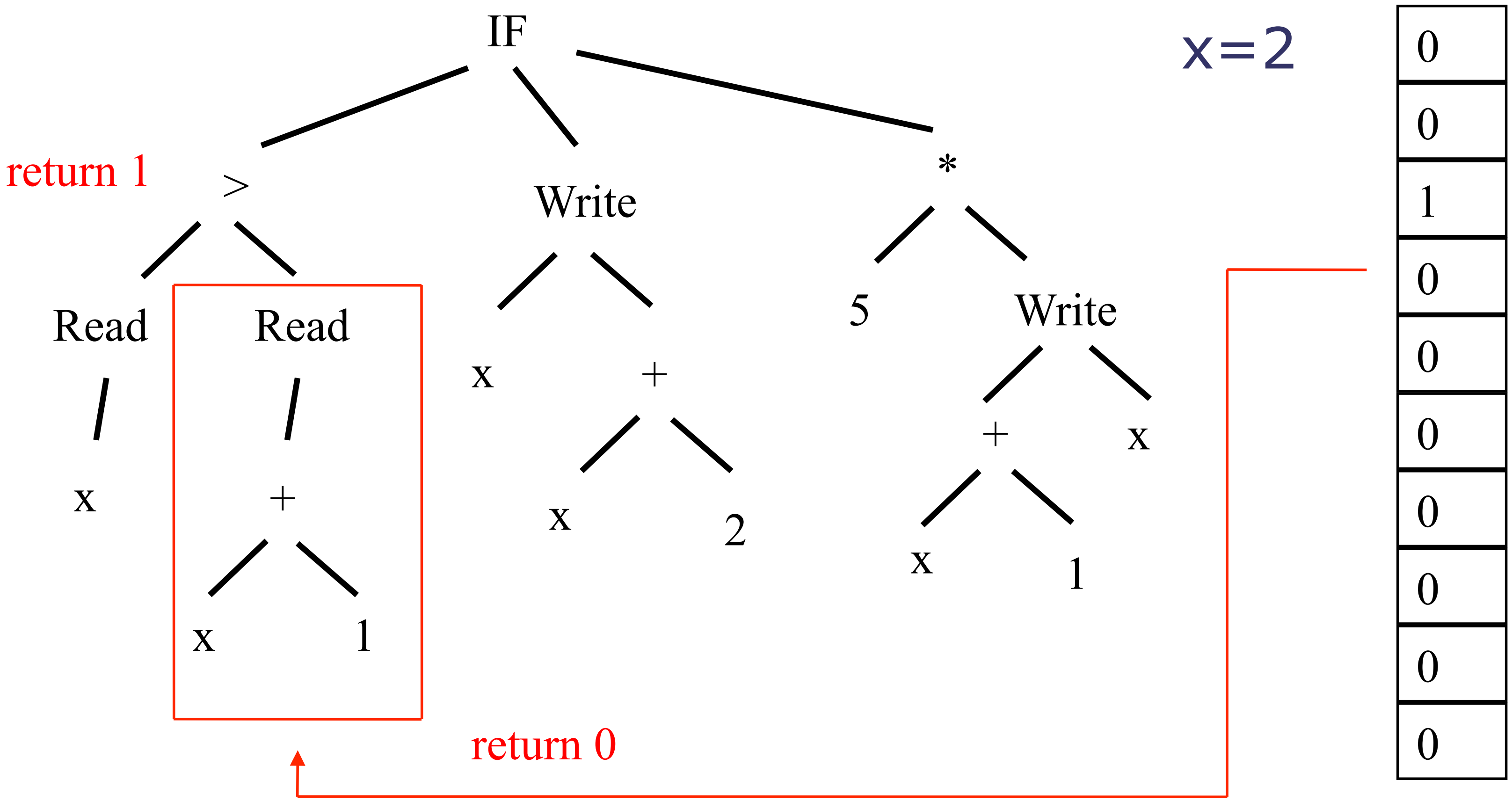
State Maintenance During Program Execution (4)



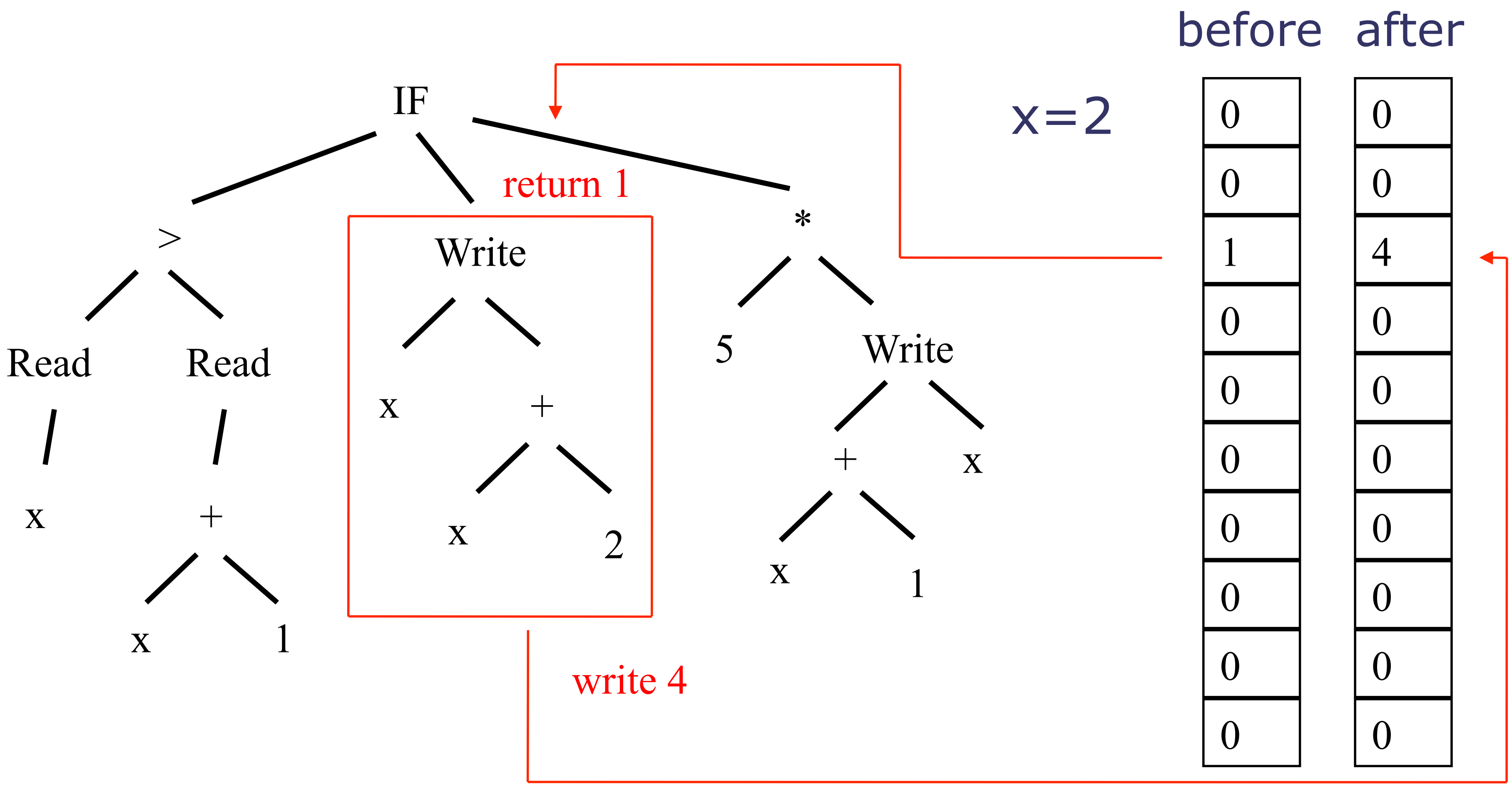
State Maintenance During Program Execution (5)



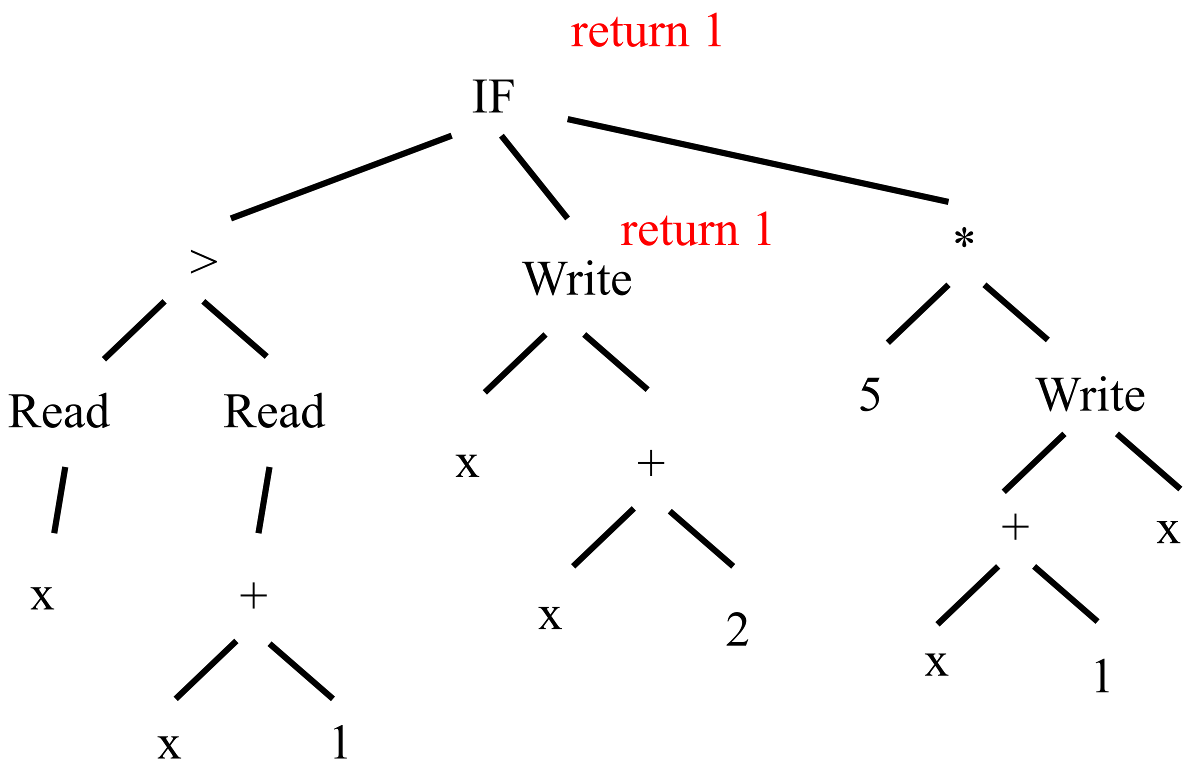
State Maintenance During Program Execution (6)



State Maintenance During Program Execution (7)



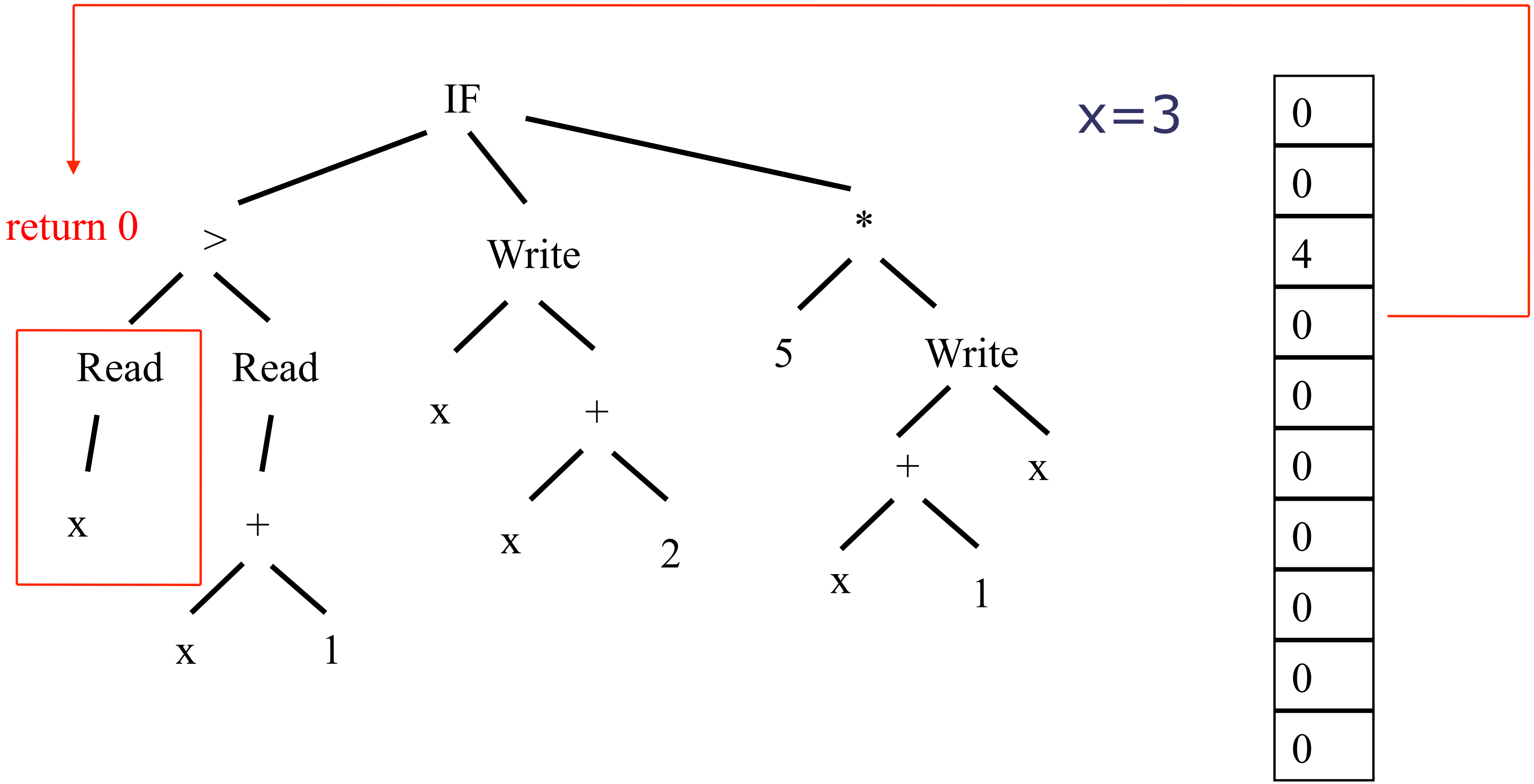
State Maintenance During Program Execution (8)



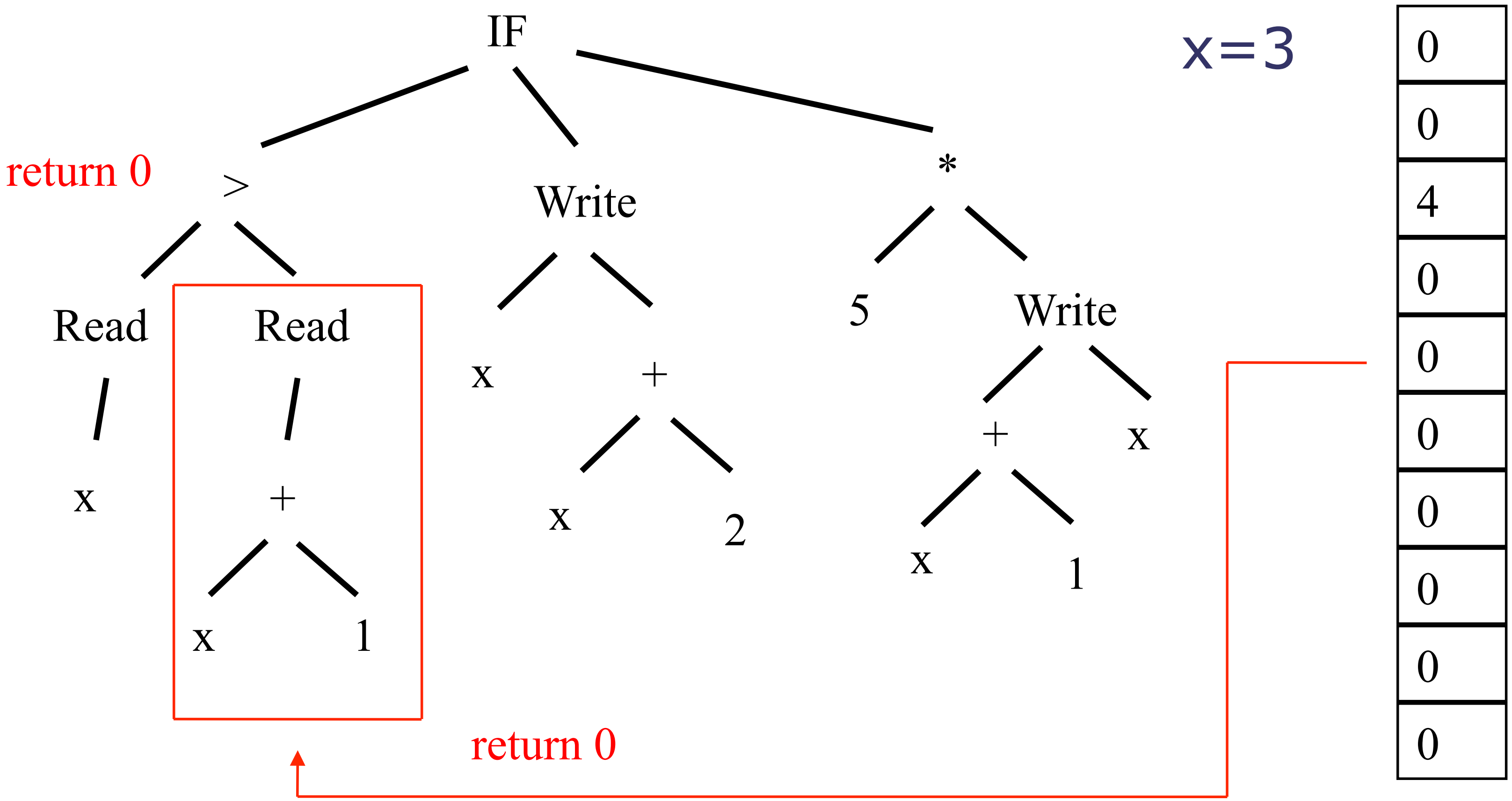
x=2

0
0
4
0
0
0
0
0
0
0

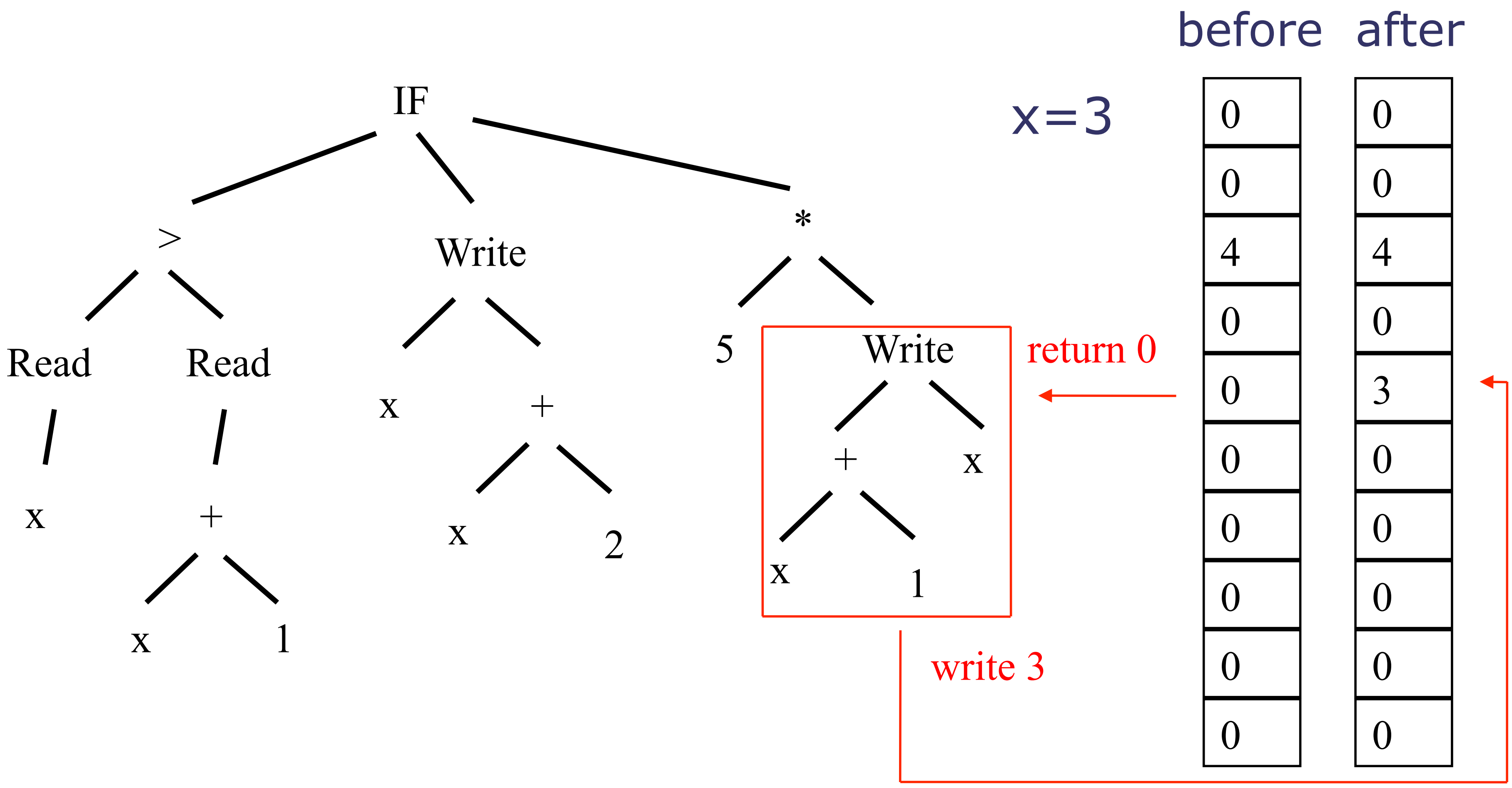
State Maintenance During Program Execution (9)



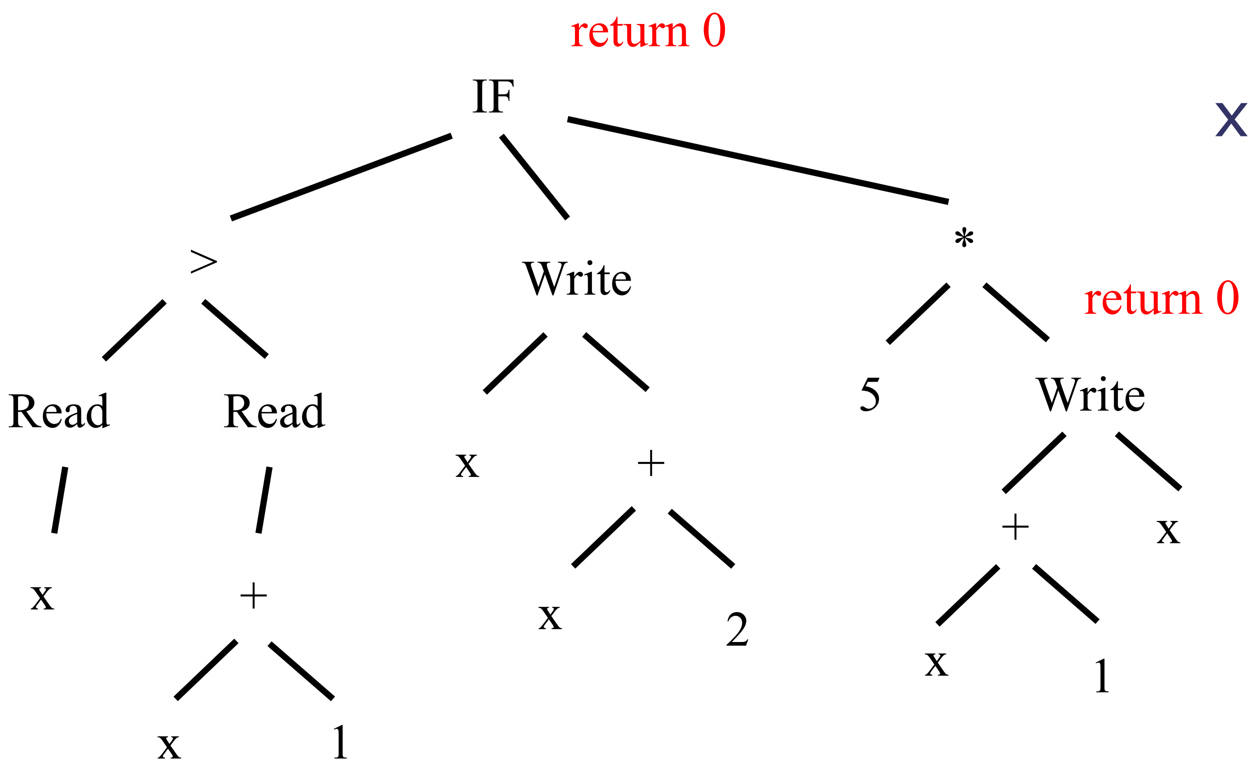
State Maintenance During Program Execution (10)



State Maintenance During Program Execution (11)



State Maintenance During Program Execution (12)



x=3

0
0
4
0
3
0
0
0
0
0

Program Execution Overview

	initial state	state a	state b	state c
input		1	2	3
output		0	1	0
	0	0	0	0
	0	0	0	0
	0	**1**	**4**	**4**
	0	0	0	0
	0	0	0	**3**
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

Scope for Research

- Is it possible to use program state to represent a model of a racing track?
- Can this model be utilised for navigation purposes?
- What is the best way to evolve programs with state using GP?
- What are the effects of stateful program representations to the evolutionary search?

- Test-bed used: **T**he **O**pen **C**ar **R**acing **S**imulator

Cooperative Coevolution of Model-builder and Car-controller Programs

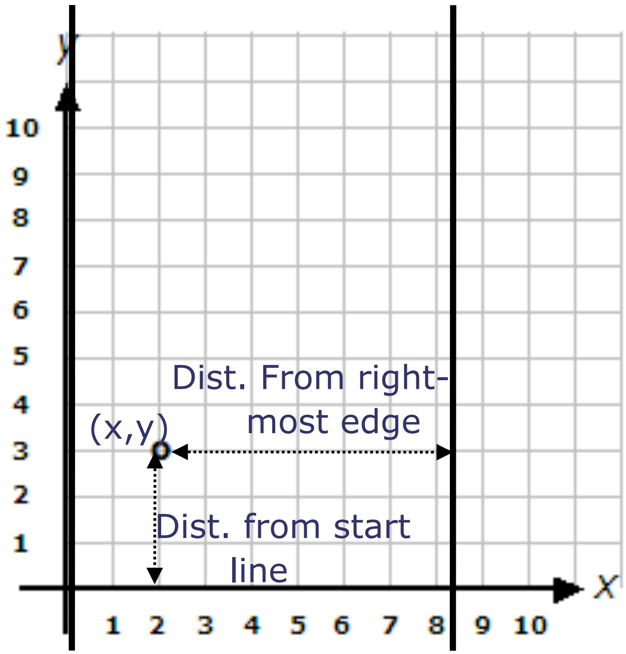
- A multi-phasic fitness evaluation procedure:
  - Phase A: **Model Building** based on sensory information.
  - Phase B: **Car Controlling** with deprivation of sensory information.

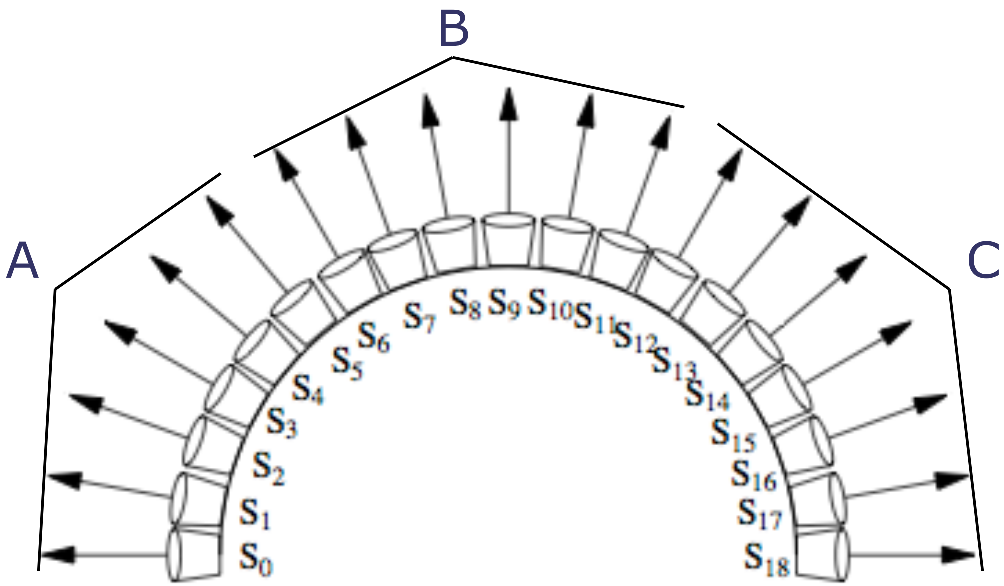
- Fitness assignment is based on the second phase of fitness evaluation.

- Program representation employs a modular architecture that consists of two individual expression-tree branches that are coupled with a general-purpose two-dimensional data-structure.

Phase A: Information for Model Building

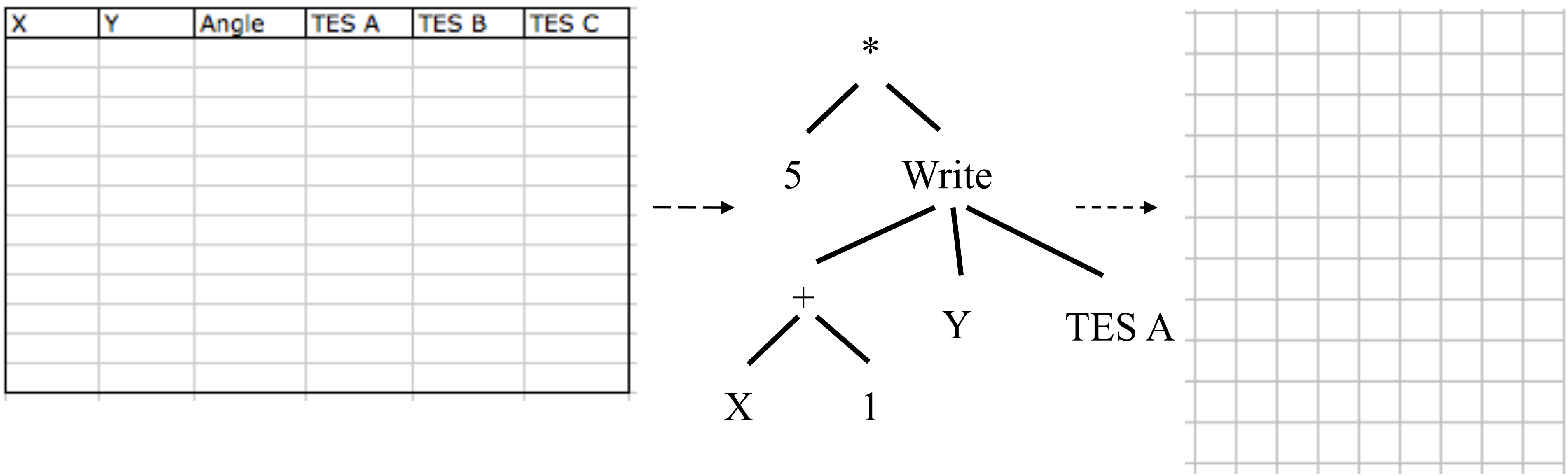
- Information required to build a model of the racing track:
  - X (normalised within the range [0, 163] for ETrack5)
  - Y (normalised within the range [0, 1621] for ETrack5)
  - Angle between the car direction and track axis
  - TrackEdgeSensors A, TrackEdgeSensors B, TrackEdgeSensors C



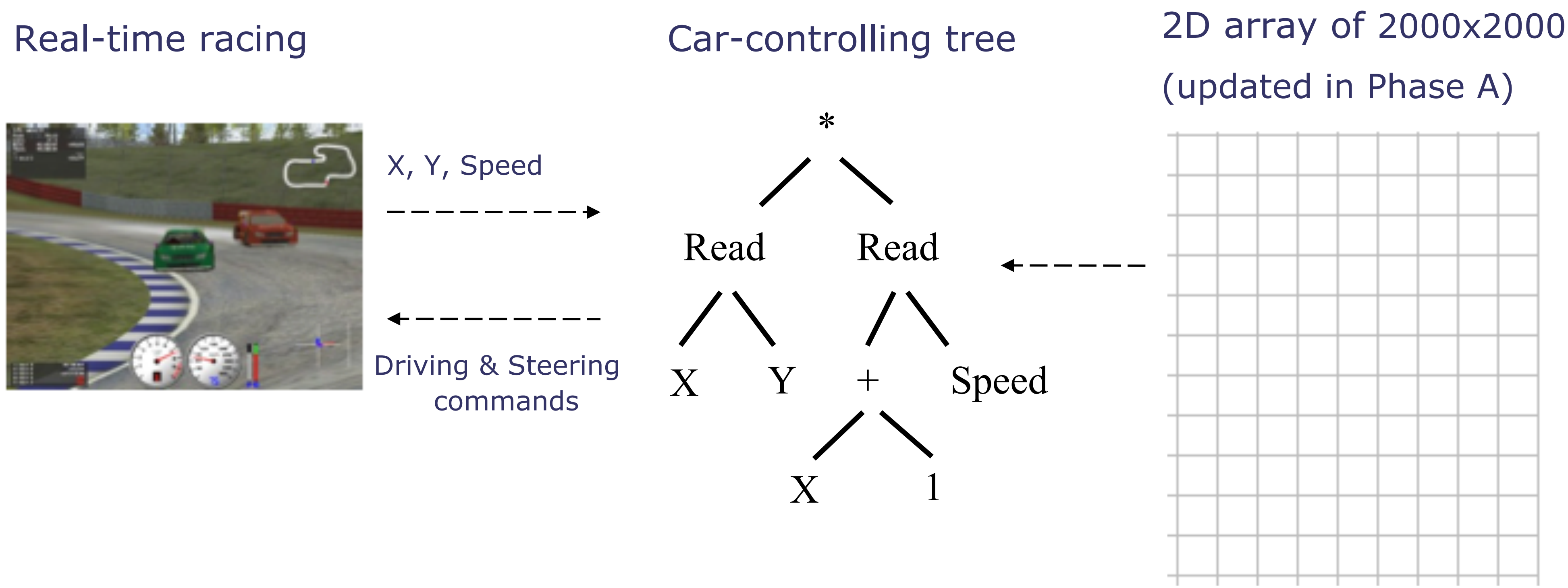


Phase A: Model Building Flow-chart

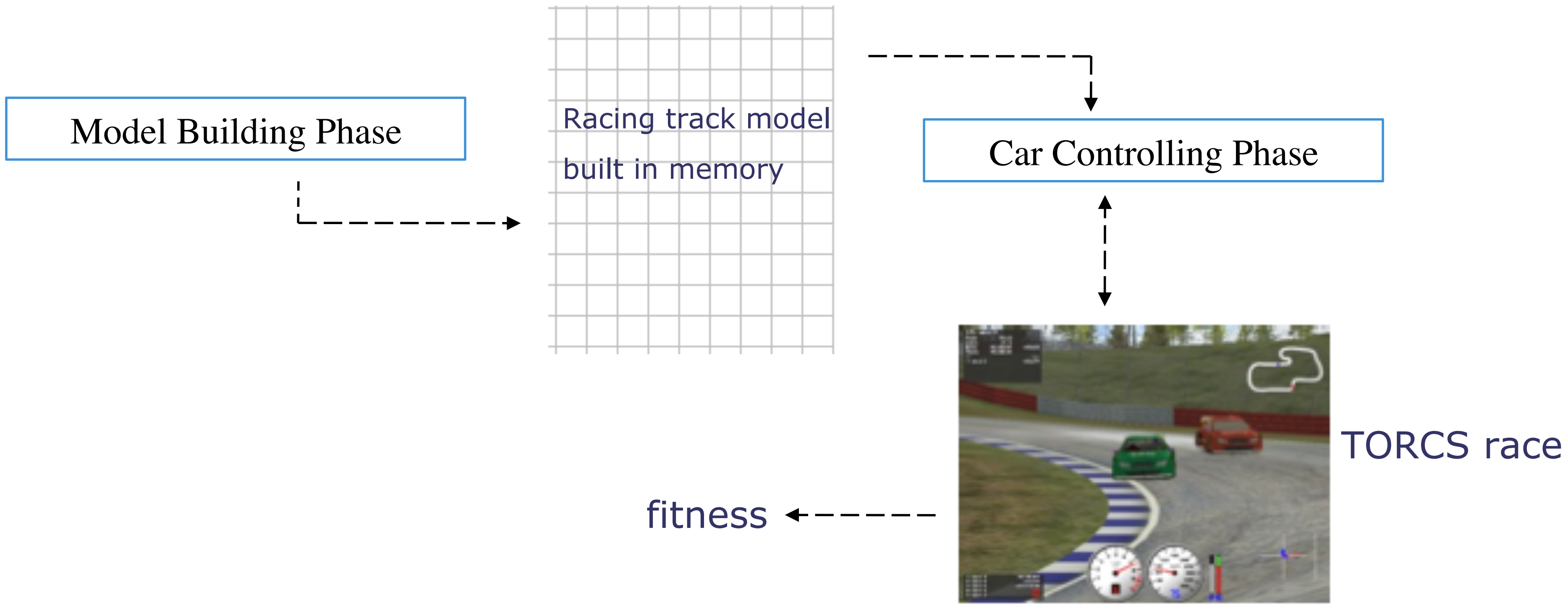
Input file generated offline | Model-building tree | 2D array of 2000x2000



Phase B: Car-controlling Flow-chart



Fitness assignment: Combining Phases A & B

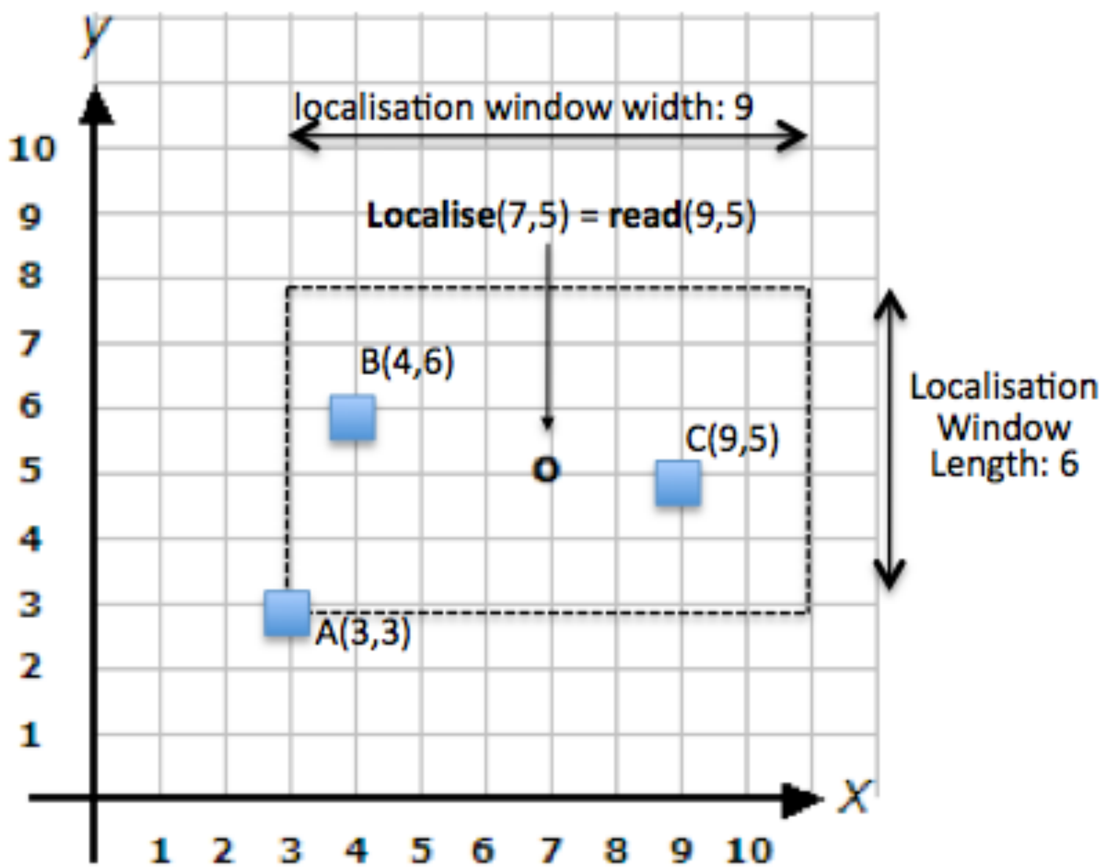


Program Representation Language

Model-Building Branch	
Non-Terminal set	**Terminal set**
add(x, y) sub(x, y) mul(x, y) div(x, y) read(x, y) write(x, y, z)	X, Y, Angle, TrackEdgeSensorsA, TrackEdgeSensorsB, TrackEdgeSensorsC

Car-Controlling Branch	
Non-Terminal set	**Terminal set**
add(x, y) sub(x, y) mul(x, y) div(x, y) read(x, y) localise(x, y)	X, Y, Speed, LateralSpeed 10 random constants in [-1.0, 1.0] interval

Method for Localisation

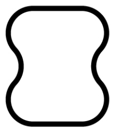


Experiment design

- **Generational, Elitist GA**
- **Population size:** 300
- **Generations:** 40
- **Tournament size:** 3
- **Expression-tree initialisation:** Ramped-half-and-half
- **Subtree mutation and crossover** (prob. set to 0.7 in favour of mutation)
- **Multiobjective fitness function to be maximised:**

$$f = w_1 DR - w_2 \frac{1}{5000} \sum_{i=1}^{5000} TCD_i$$

- **Race duration:** 5,000 time-steps
- **Racing track:** Etrack5

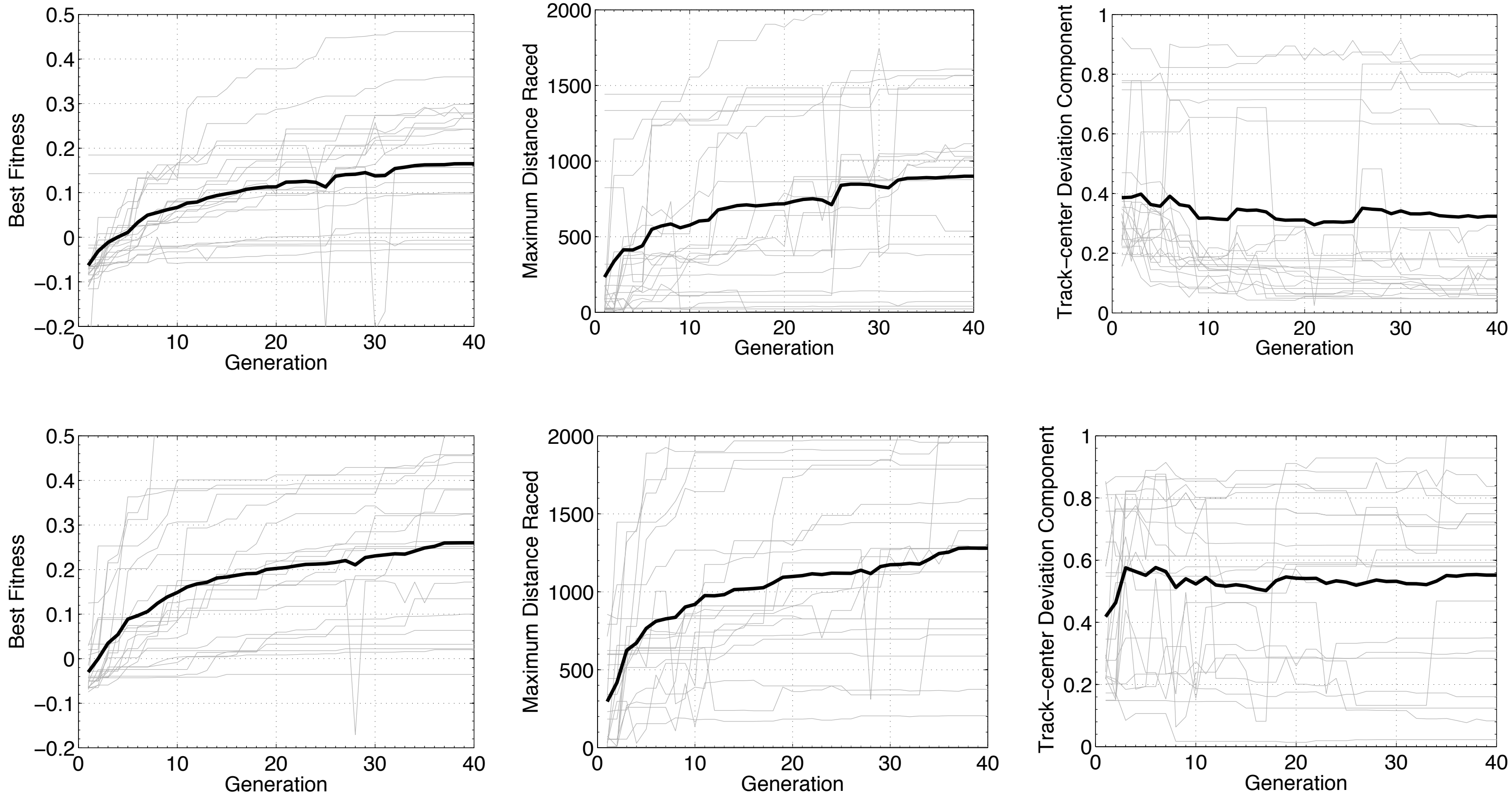


- **Maximum gear:** constrained to gear 1
- **Car1-trb1 speed:** approx. 82Km/h
- **Distance covered:** approx. 2,320m within 5,000 time-steps

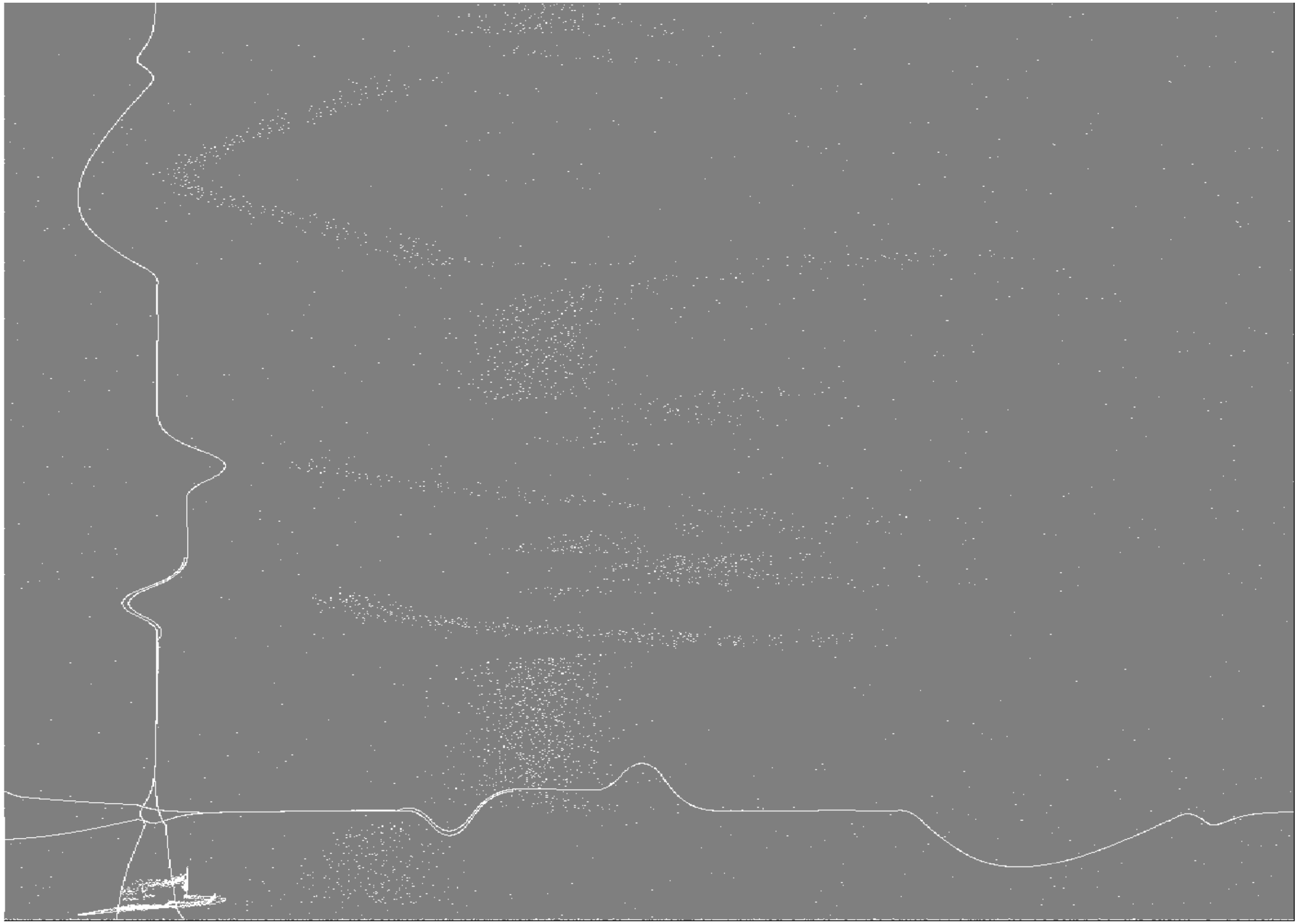
Performance Histograms

$(w_1,w_2)=(0.65, 0.35)$

$(w_1,w_2)=(0.75, 0.25)$



Heatmap of Read-Write Overlap



UCD Natural Computing Research & Applications Group (NCRA)

Conclusions

- Successful cooperative coevolution of two programs that share memory.
- A fitness function that penalises a racing line that deviates heavily from the track's center provides the necessary search bias towards the effective use of memory.
- Most evolved racing track models exploited a roughly isomorphic relation between the environment and the memory by mapping the car's move in the racing track to an equivalent position in the 2D array.
- Indexed memory is a powerful extension to GP for the effective storage and retrieval of information.
- For future work:
  - Study methods to allow agents to utilise the track model to plan.
  - Generalisation of model-building ability.

Thank you