AI VIETNAM
All-in-One Course
(TA Session)

# Genetic Algorithm

## Extra

Dinh-Thang Duong – TA

Anh-Khoi Nguyen – STA

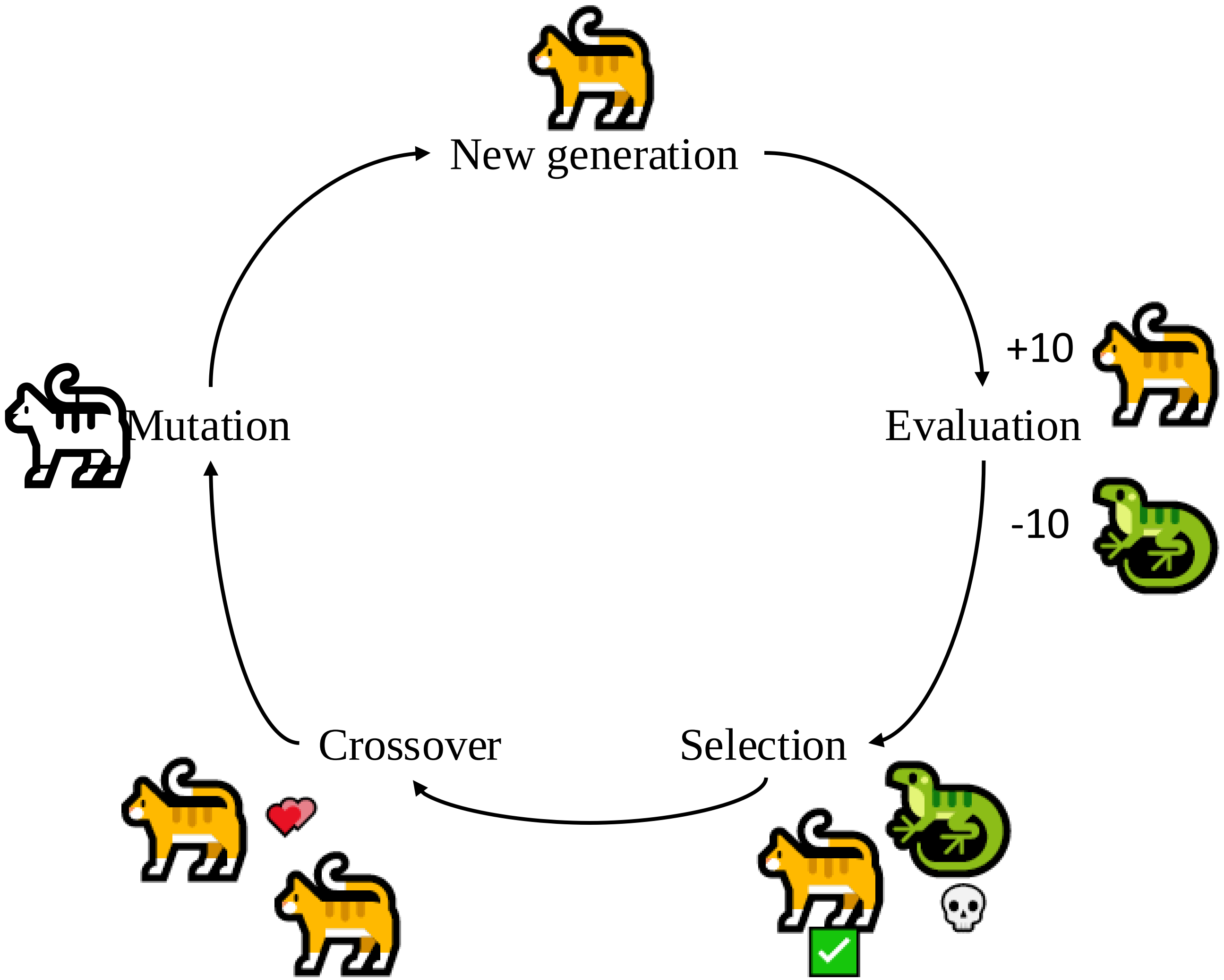
Yen-Linh Vu – STA

AI VIET NAM
@aivietnam.edu.vn

Year 2024

# Getting Started

## ❖ Objectives




**Our objectives:**

- Introduction to Genetic Algorithm.
- Discuss how Genetic Algorithm works.
- Implement a simple version of Genetic Algorithm to solve One-max problem.

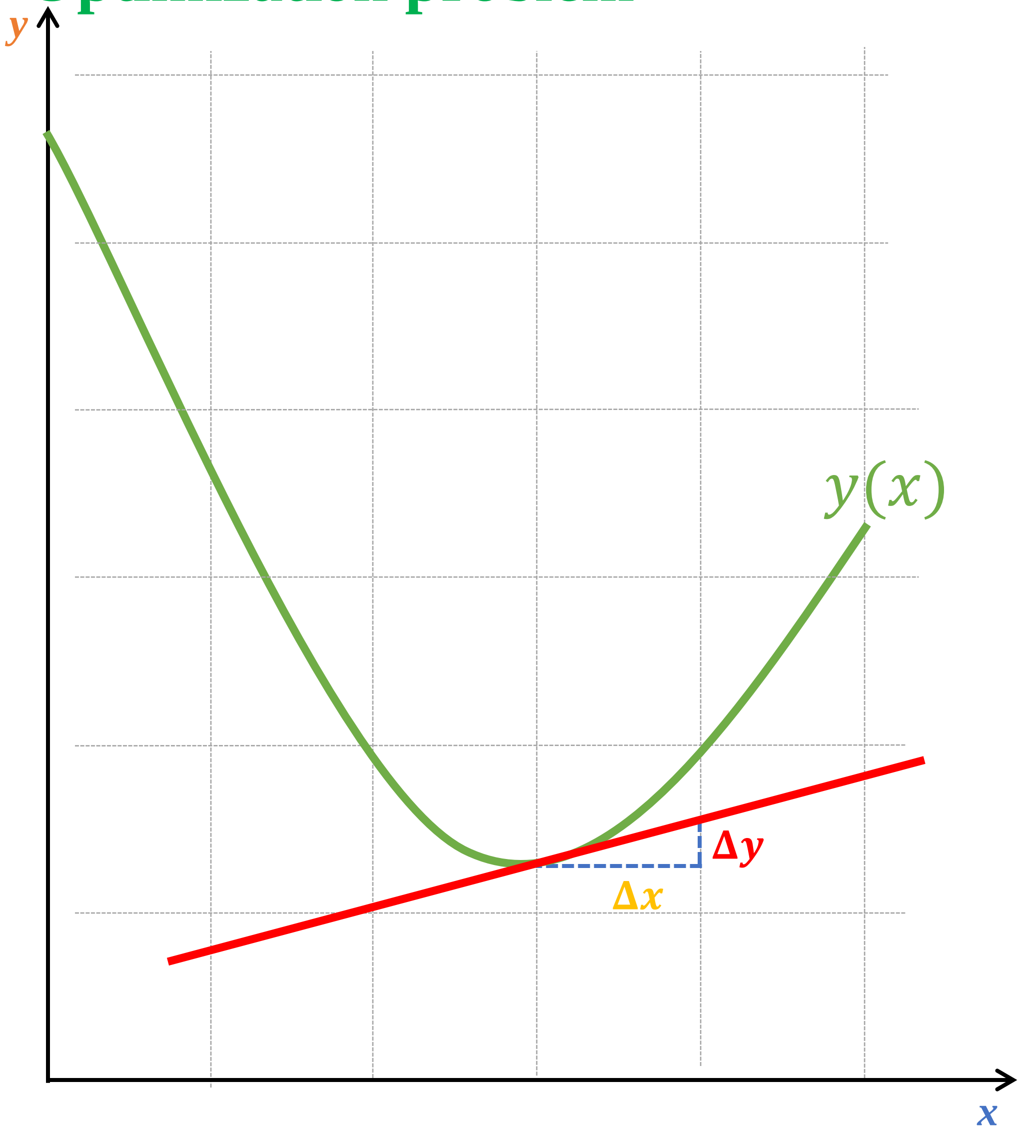
# Outline

- Introduction
- Genetic Algorithm
- Code Implementation
- Question

# Introduction

# Introduction

## ❖ Optimization problem



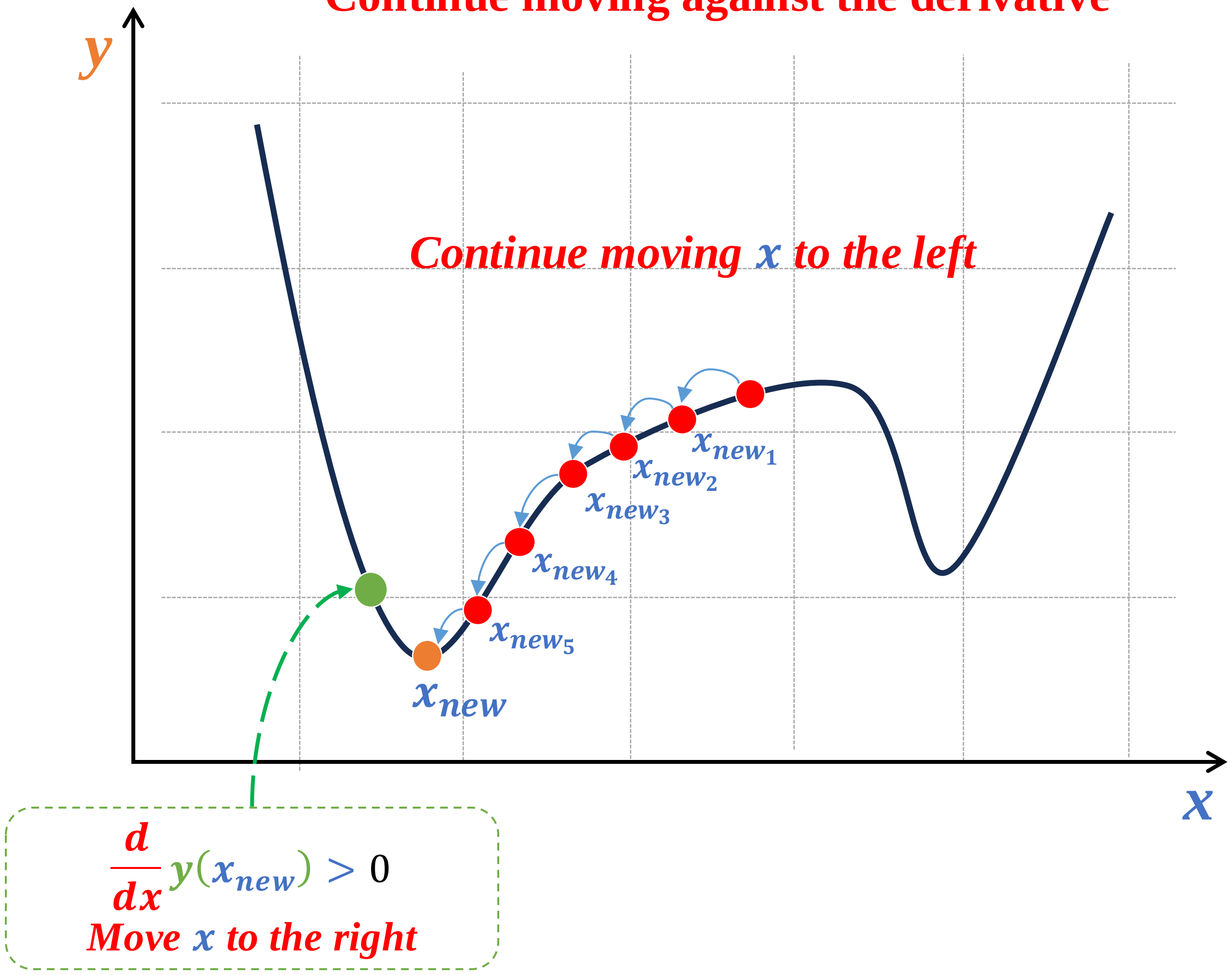

$$\frac{d}{dx}y(x) = \lim_{\Delta x \to 0} \frac{y(x+\Delta x) - y(x)}{\Delta x}$$

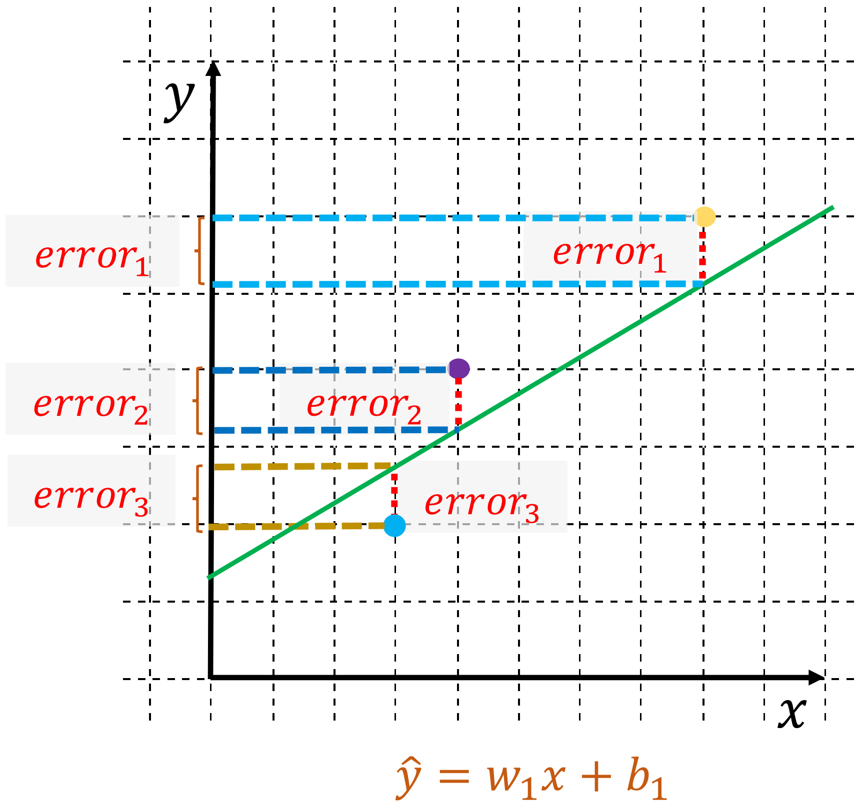
1. How to find minimal value of $y(x) = x^2$ on $\mathbb{R}$?
=> Take the derivative of $y(x)$.

# Introduction

## ❖ Optimization problem

**Continue moving against the derivative**



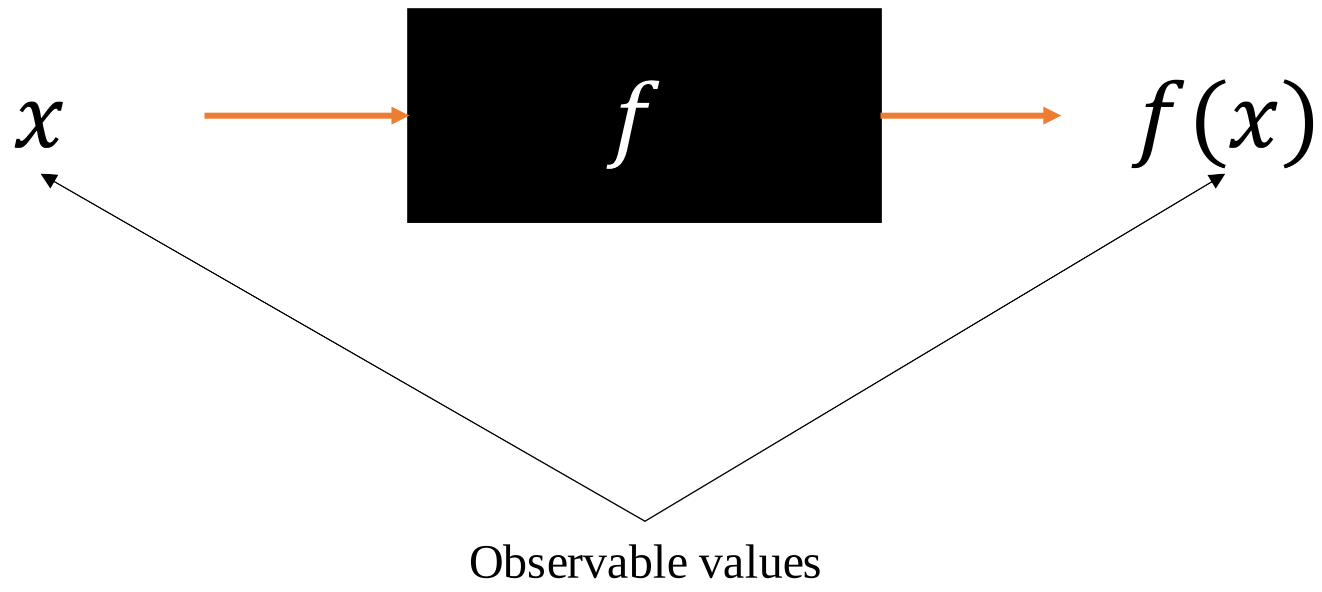




2. How to find weights of a linear function so that the loss is minimum?
=> Use gradient descent.

# Introduction

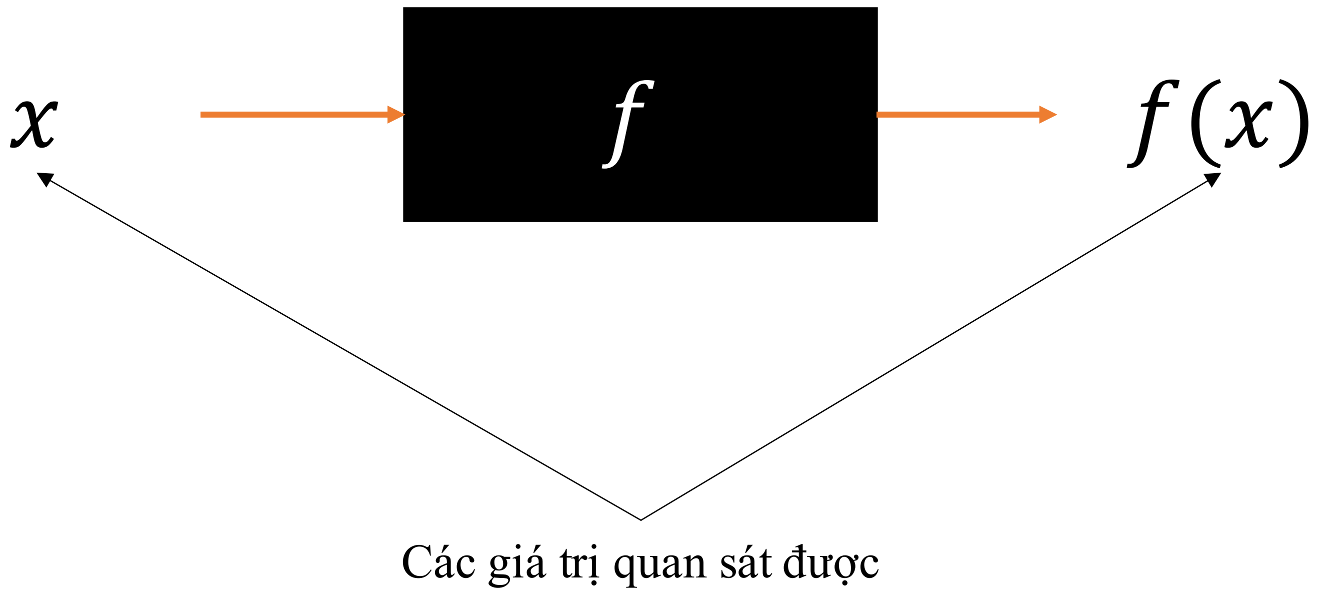
## ❖ Blackbox Optimization

In practice, we might not know the function of our problem to be able to optimize it using aforementioned methods. Trying to optimize in this case is called **black-box optimization.**



# Introduction

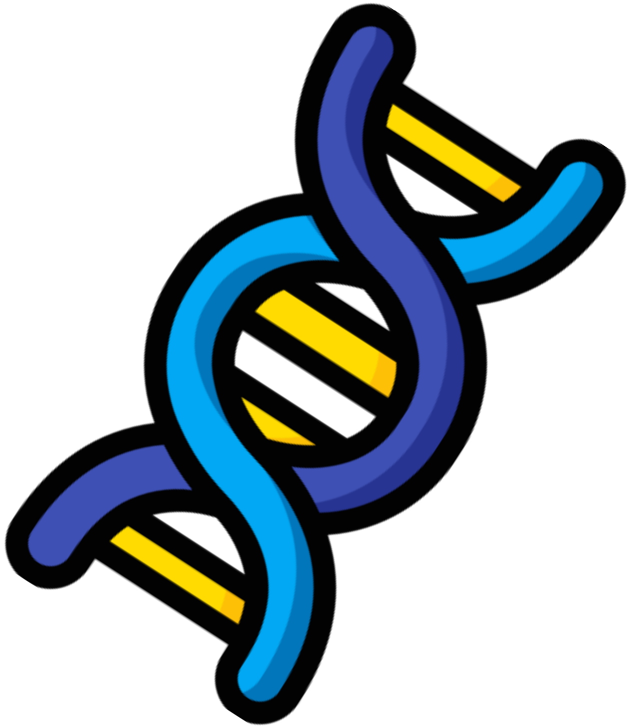
## ❖ Blackbox Optimization




How to solve an optimization problem where only know about the input and output values of the problem?

# Introduction

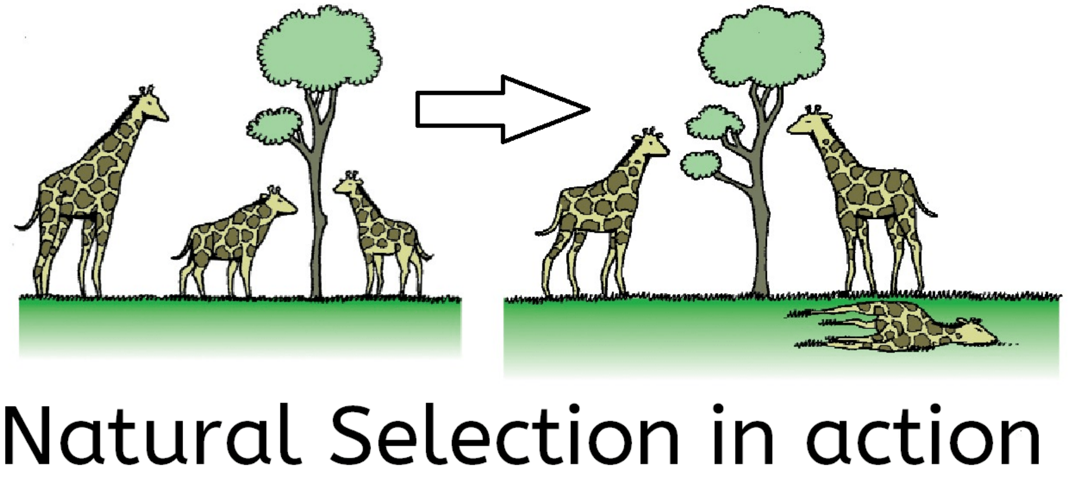
## ❖ Genetic Algorithm



**Genetic Algorithm (GA):** A search and optimization technique in Evolutionary Algorithms, inspired by the process of natural selection where a population of candidate solutions evolves over time.

# Introduction

## ❖ Genetic Algorithm



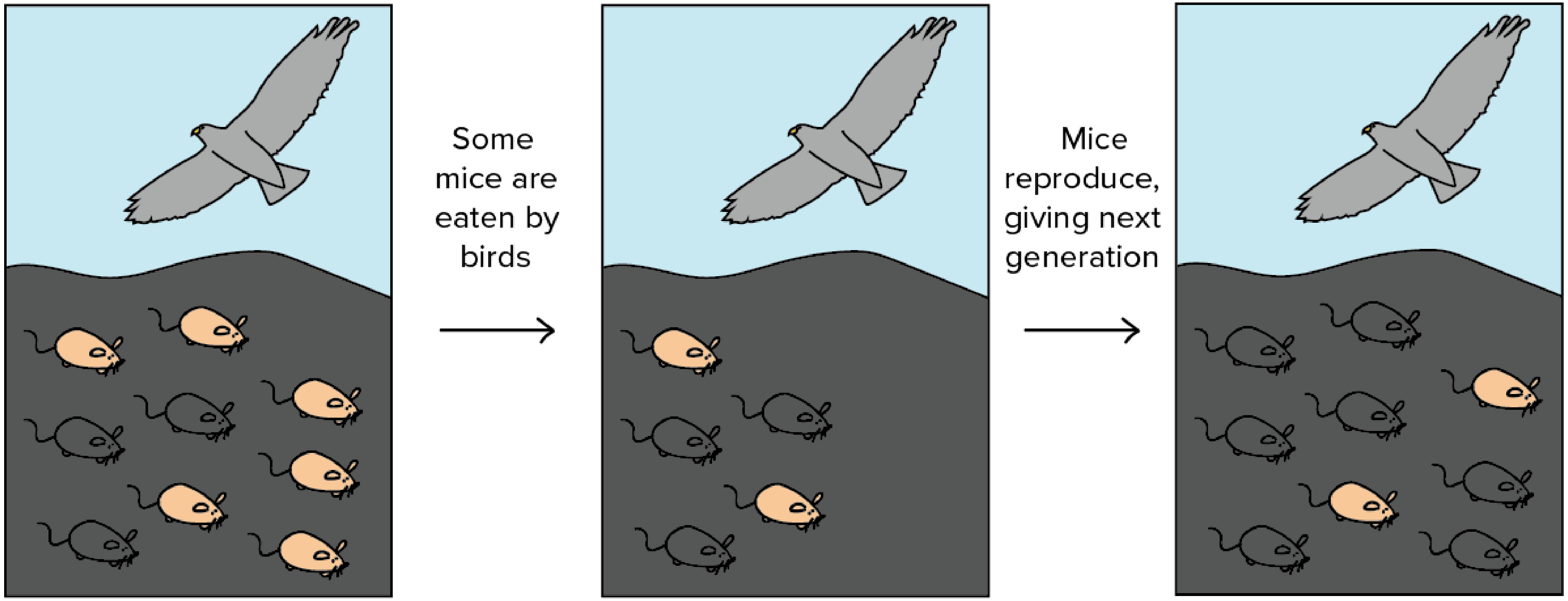

**Genetic Algorithm (GA)** mimic the process of natural selection, meaning that species who are able to adjust to changes in their surroundings will be able to endure, procreate, and pass on their genes to the following generation.



https://creationscience4kids.com/can-natural-selection-make-anything/

# Introduction

## ❖ Natural Selection Process Example




A population of mice has moved into a new area where the rocks are very dark. Due to natural genetic variation, some mice are black, while others are tan.

Tan mice are more visible to predatory birds than black mice. Thus, tan mice are eaten at higher frequency than black mice. Only the surviving mice reach reproductive age and leave offspring.

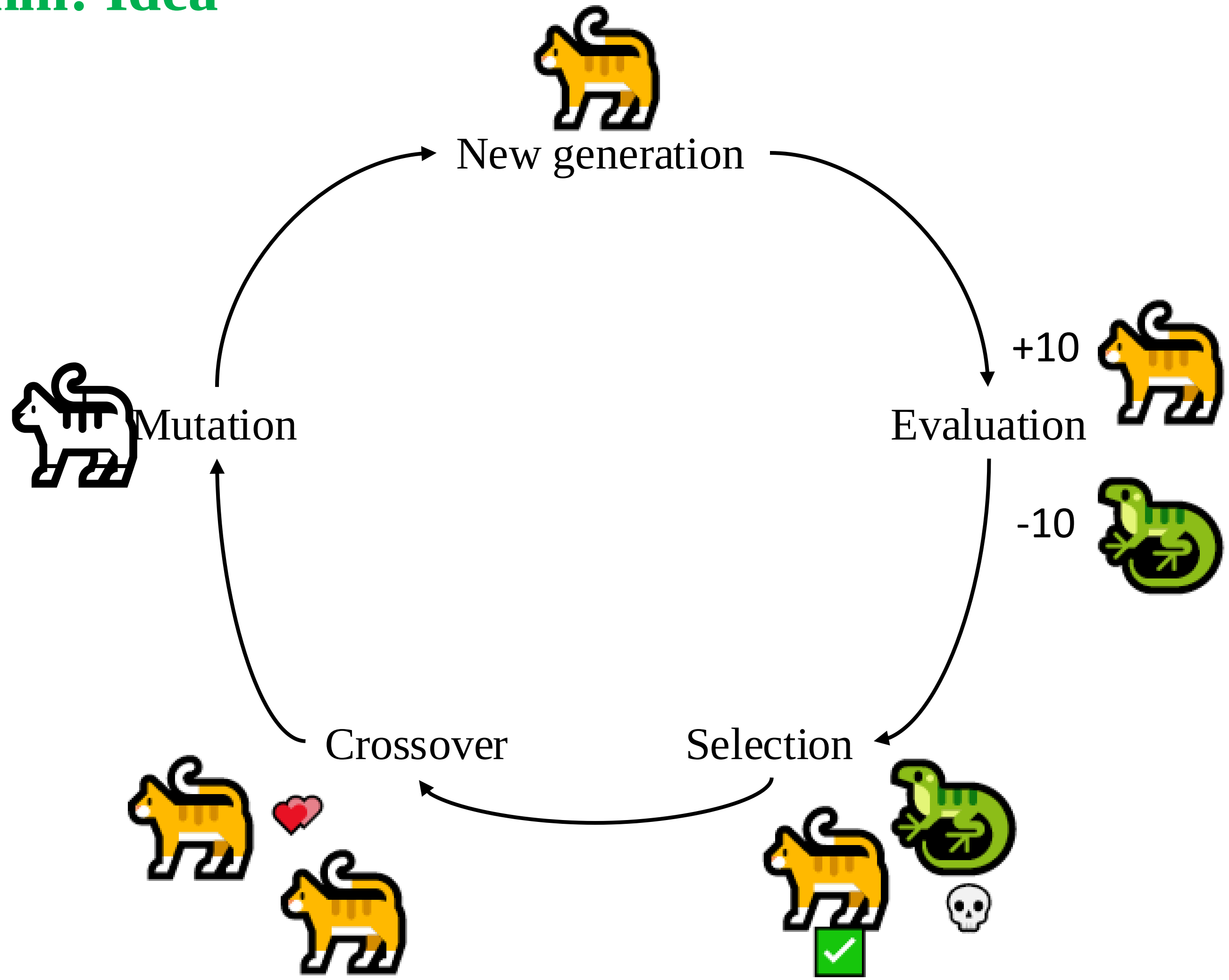
Because black mice had a higher chance of leaving offspring than tan mice, the next generation contains a higher fraction of black mice than the previous generation.

https://www.khanacademy.org/science/ap-biology/natural-selection/natural-selection-ap/a/darwin-evolution-natural-selection

# Genetic Algorithm

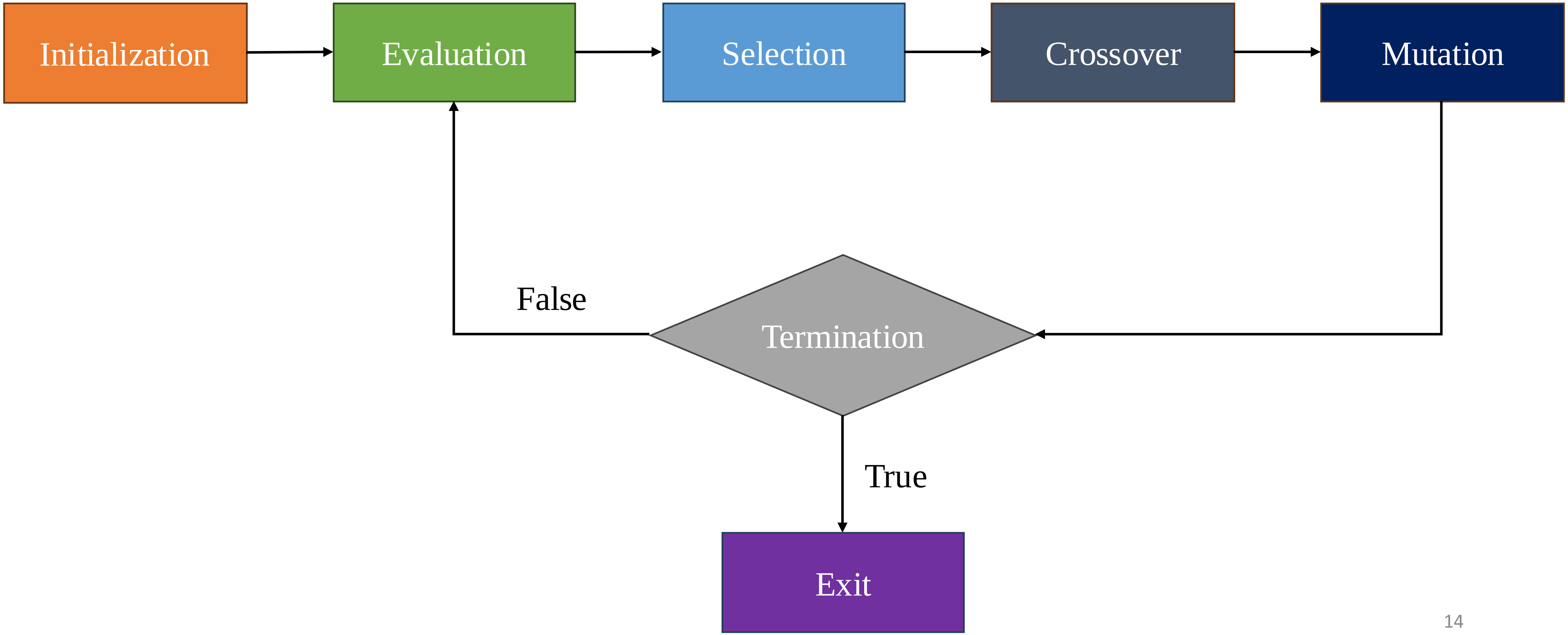
# Genetic Algorithm

## ❖ Genetic Algorithm: Idea



# Genetic Algorithm

## ❖ Simple GA Pipeline



# Genetic Algorithm

## ❖ Individual

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |

String of bits 0 - 1

An **individual** in a genetic algorithm represents a single candidate solution to the optimization problem.

We can consider a **chromosome** as a list of bits, where each bit represents a gene that encodes part of the solution.

# Genetic Algorithm

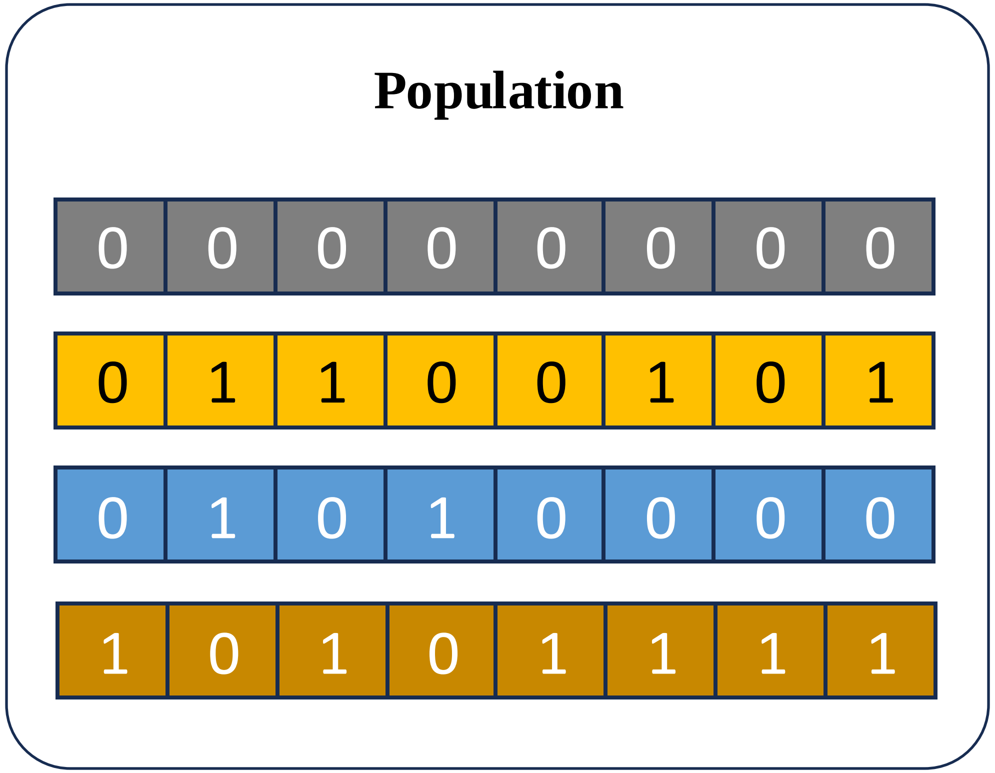
## ❖ Initialization

**Initial Population**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

## ❖ Population

A **population** in a genetic algorithm is a collection of individuals (chromosomes) that represent different possible solutions to the optimization problem.



# Genetic Algorithm

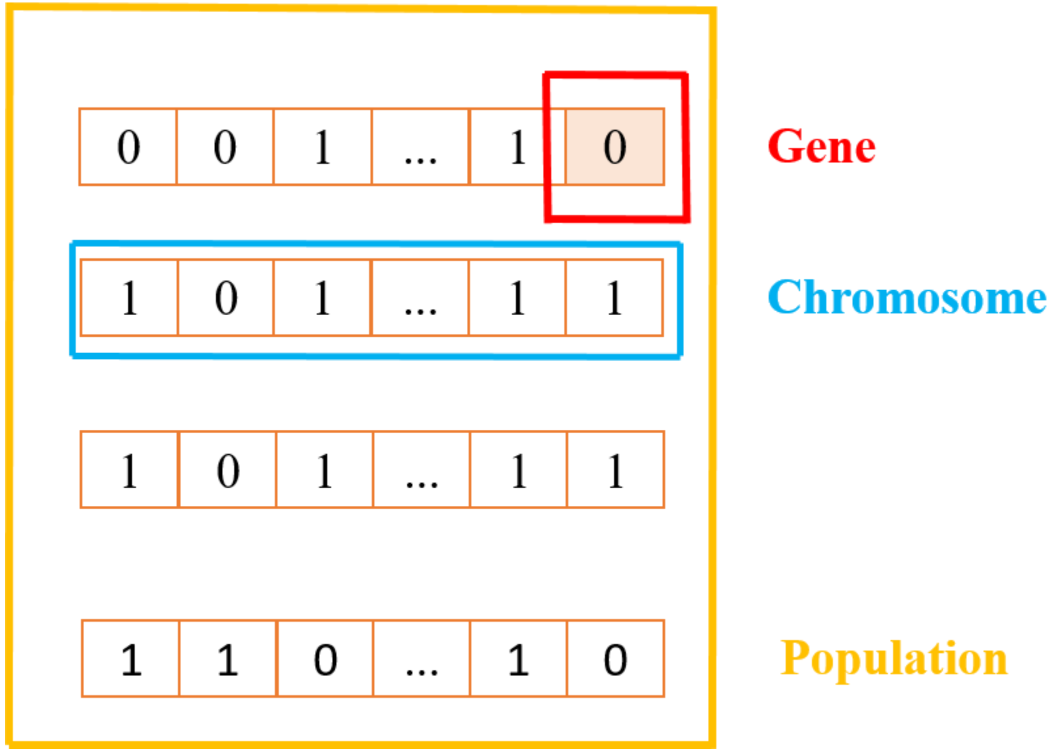
## ❖ Initialization

**Initial Population**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

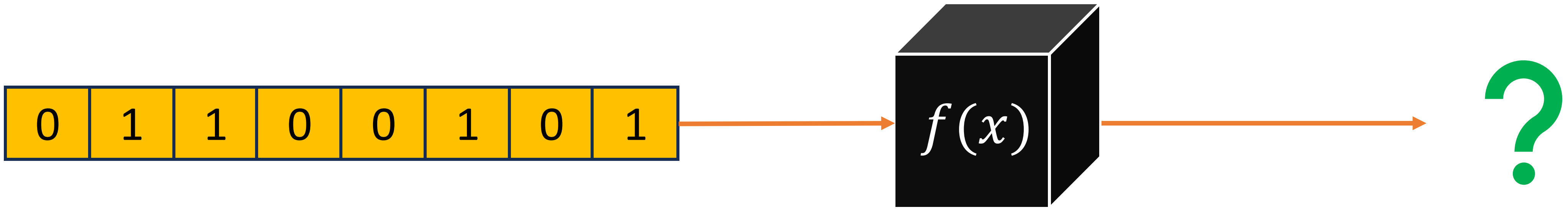
# Genetic Algorithm

## ❖ Summary of a gene, chromosome and population



# Genetic Algorithm

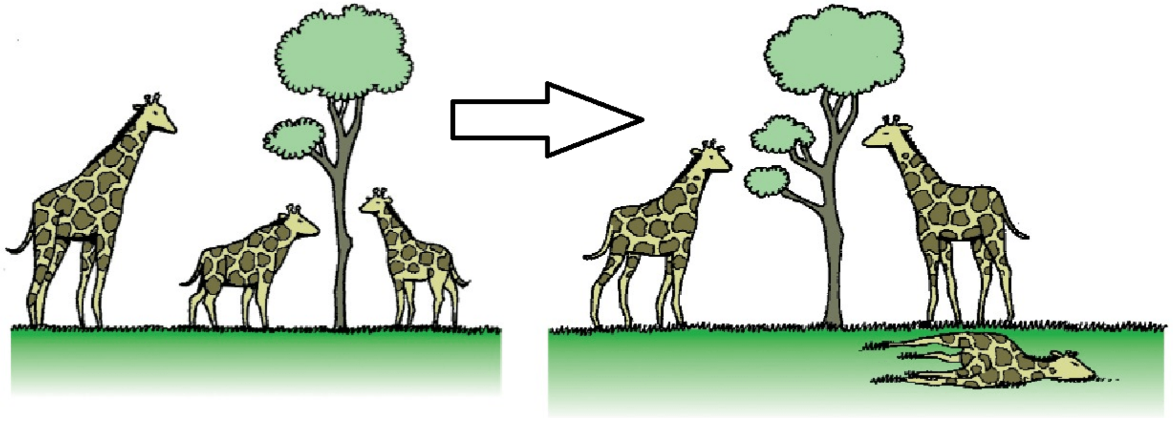
## ❖ Evaluate



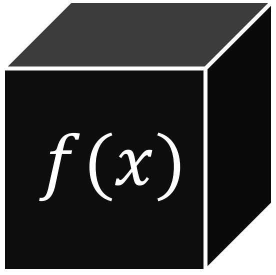

Given an individual (solution), how can we determine how well it is?

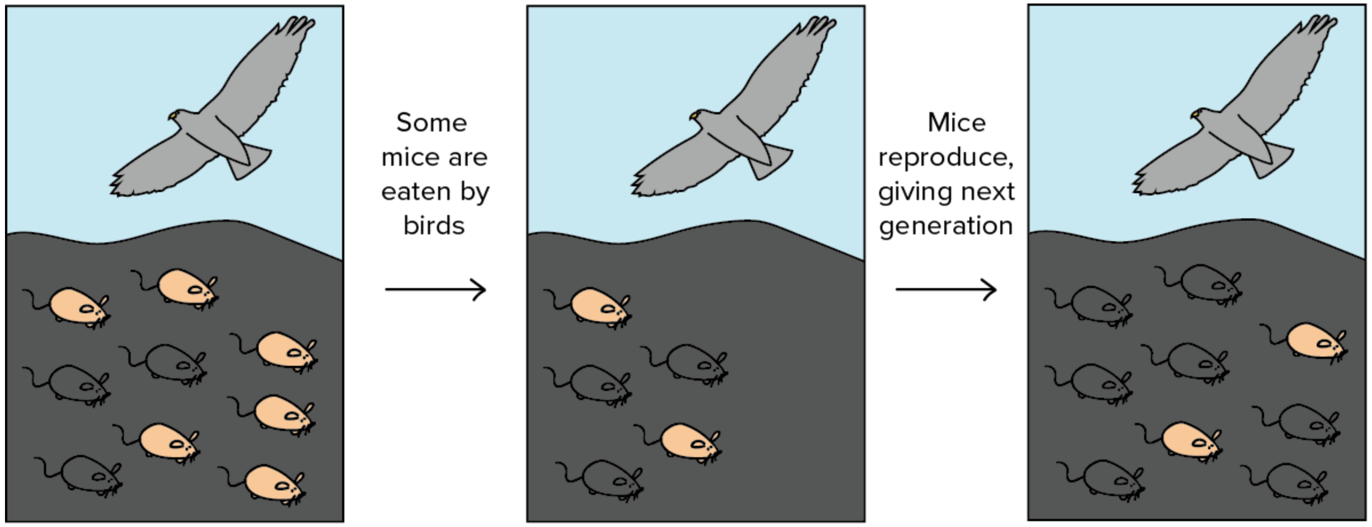
# Genetic Algorithm

## ❖ Evaluate



$f(x) = High$



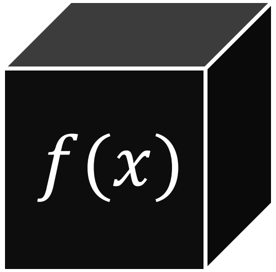




$f(x) = Color$

# Genetic Algorithm

## ❖ Fitness

$$f(x) = High$$

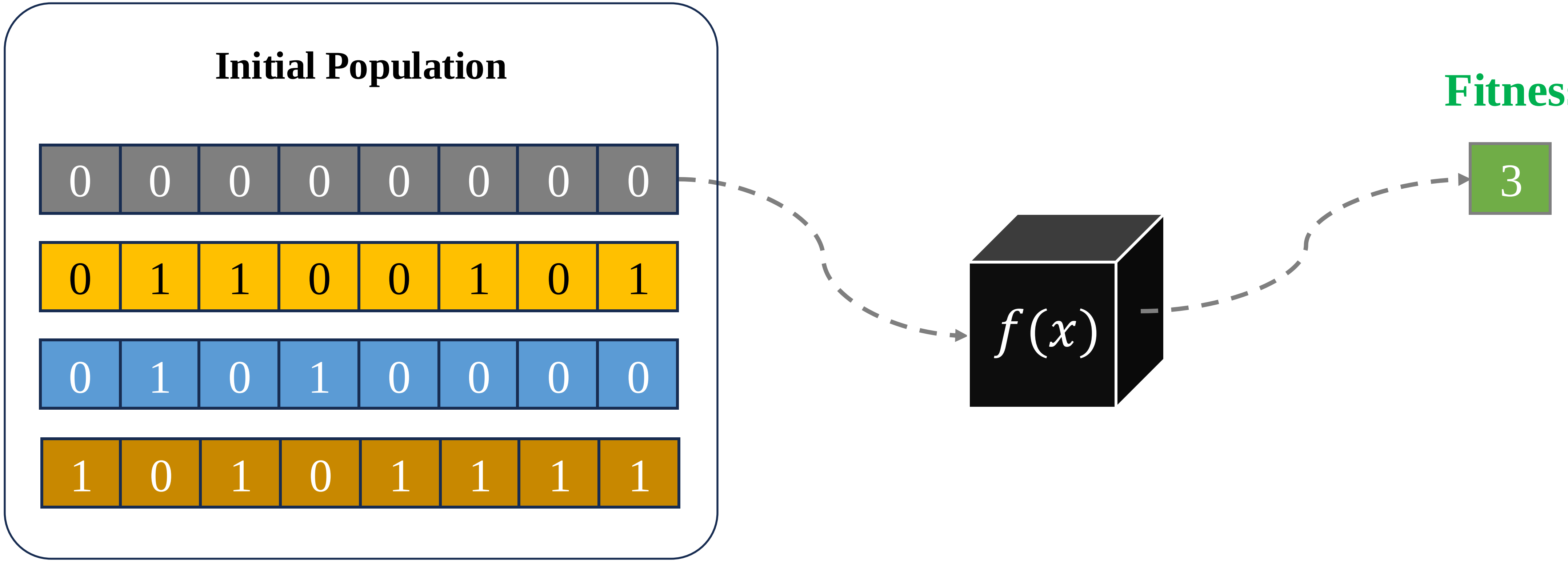



$$f(x) = Color$$

**Fitness** in a genetic algorithm is a measure of how well an individual (chromosome) performs in solving the given problem, with higher fitness indicating a better solution.

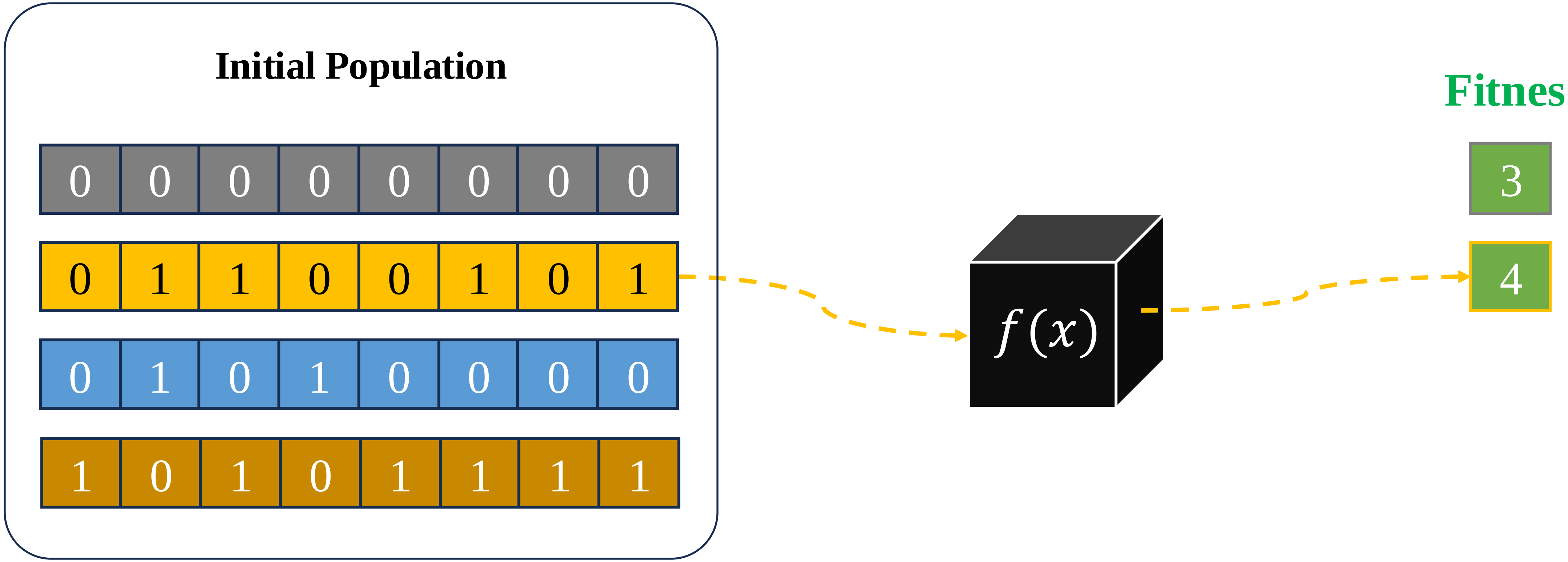
# Genetic Algorithm

## ❖ Evaluate



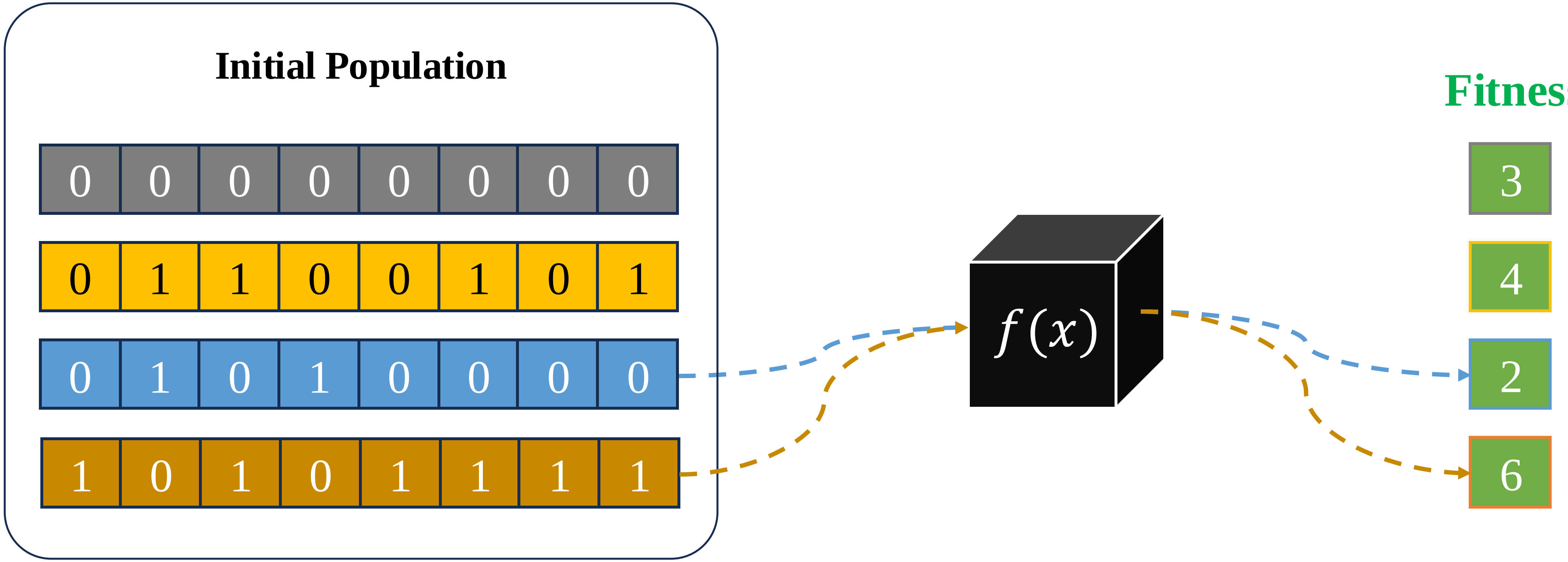
# Genetic Algorithm

## ❖ Evaluate



# Genetic Algorithm

## ❖ Evaluate

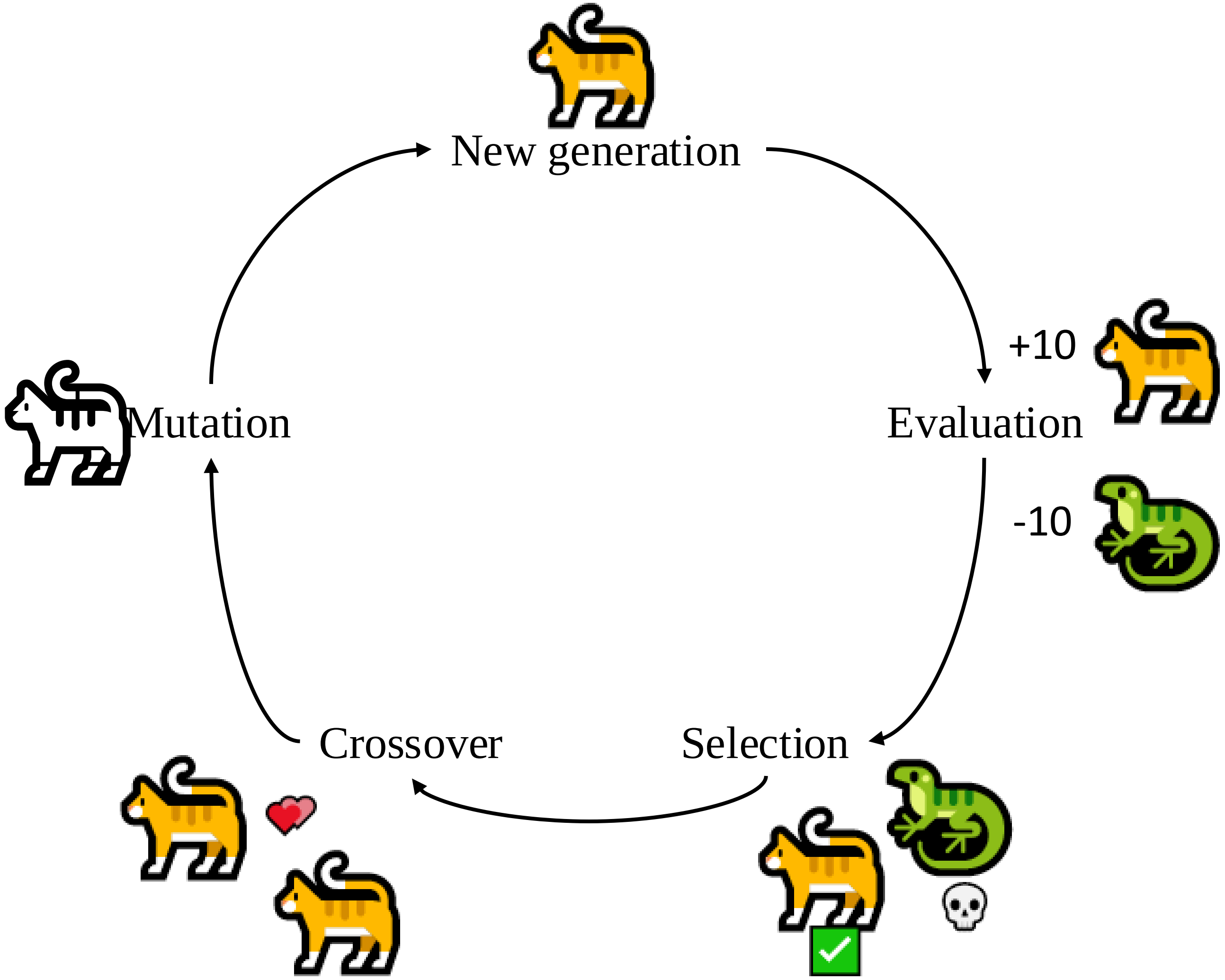


# Genetic Algorithm

## ❖ How to choose individuals?

**Initial Population**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |



# Genetic Algorithm

## ❖ Selection

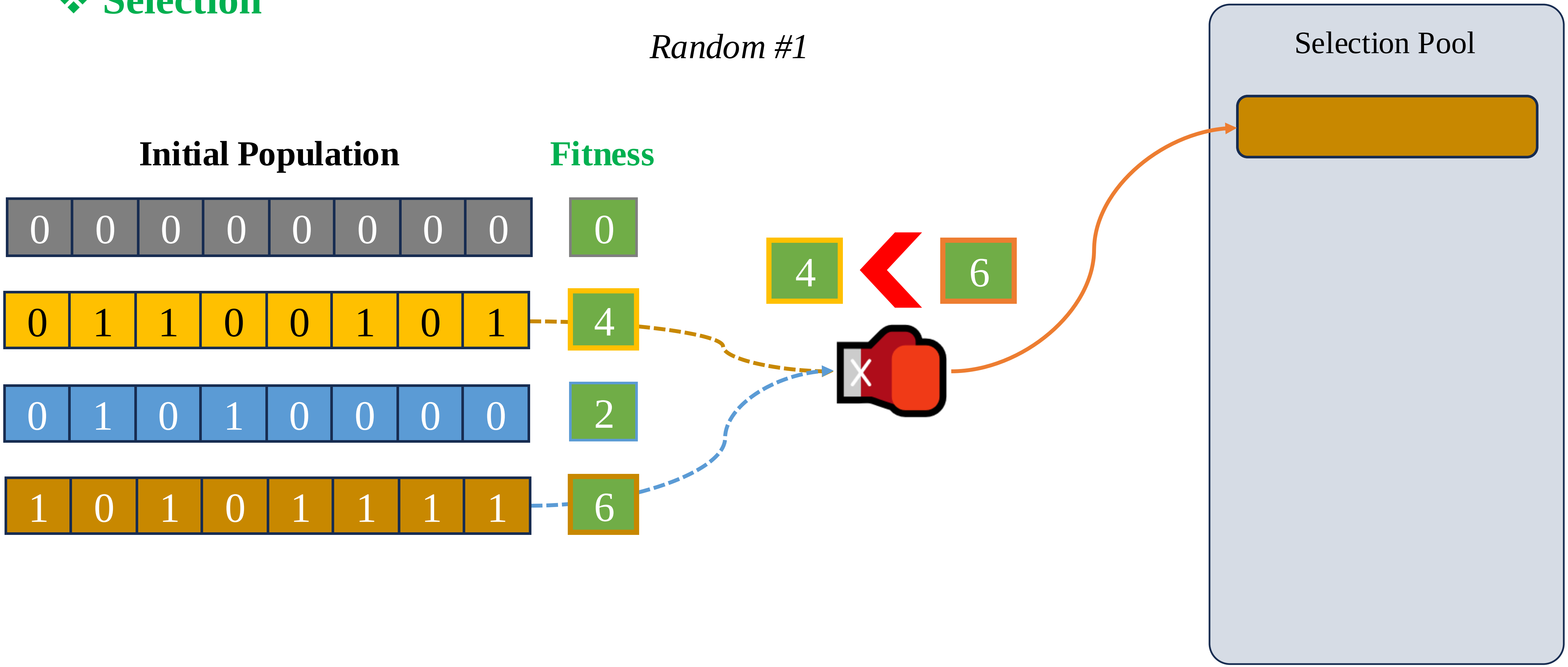
**Initial Population**

| | | | | | | | | **Fitness** |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 4 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 2 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 6 |

**Selection** in a genetic algorithm is the process of choosing individuals from the population based on their fitness to serve as parents for the next generation.

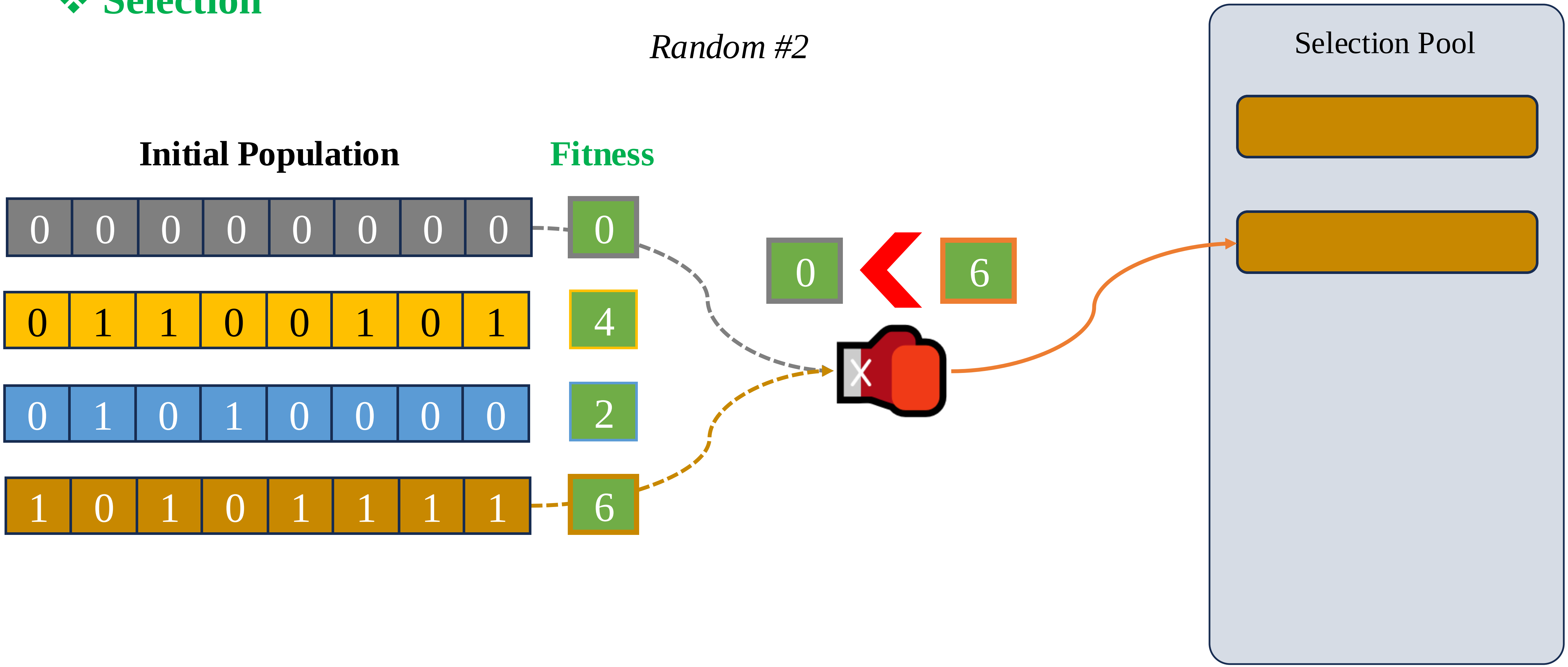
# Genetic Algorithm

## ❖ Selection



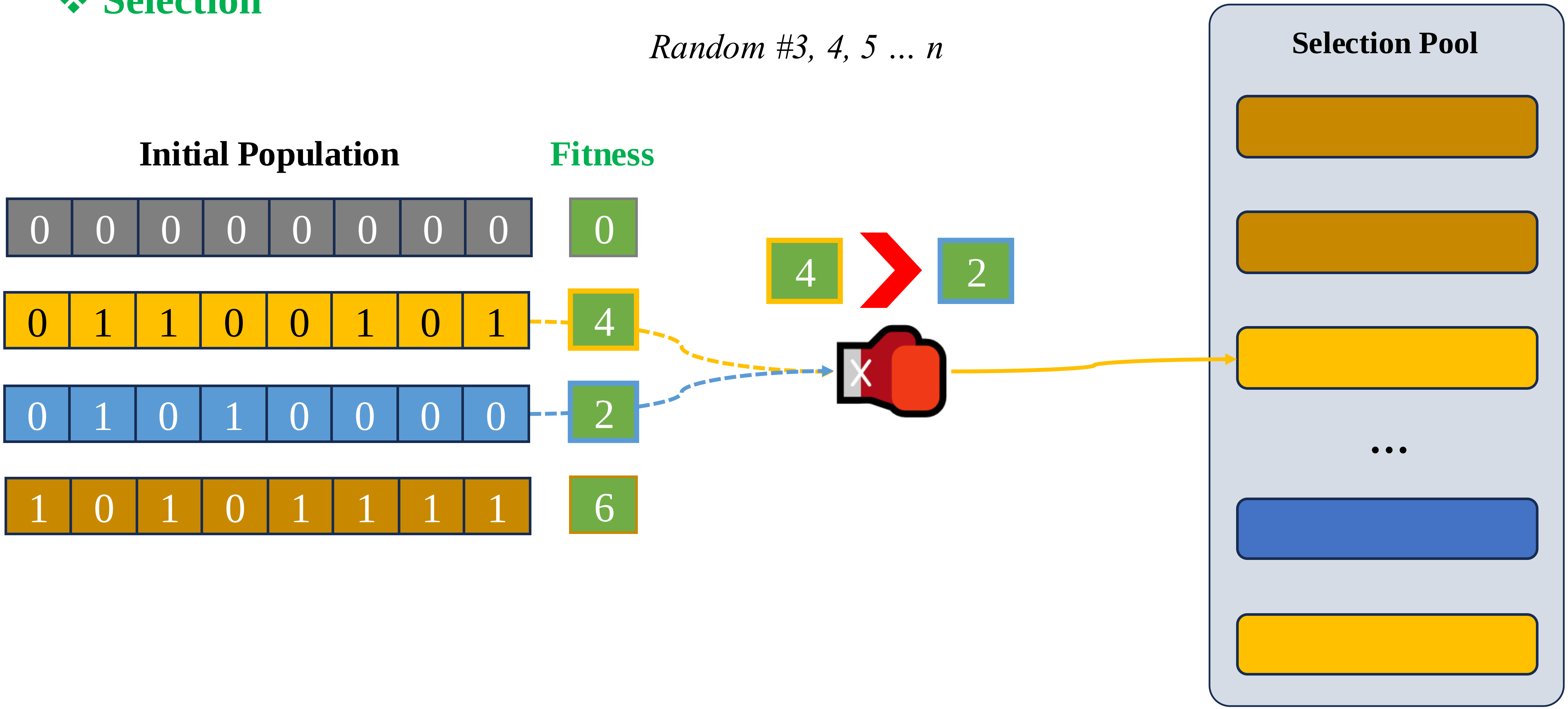
# Genetic Algorithm

## ❖ Selection



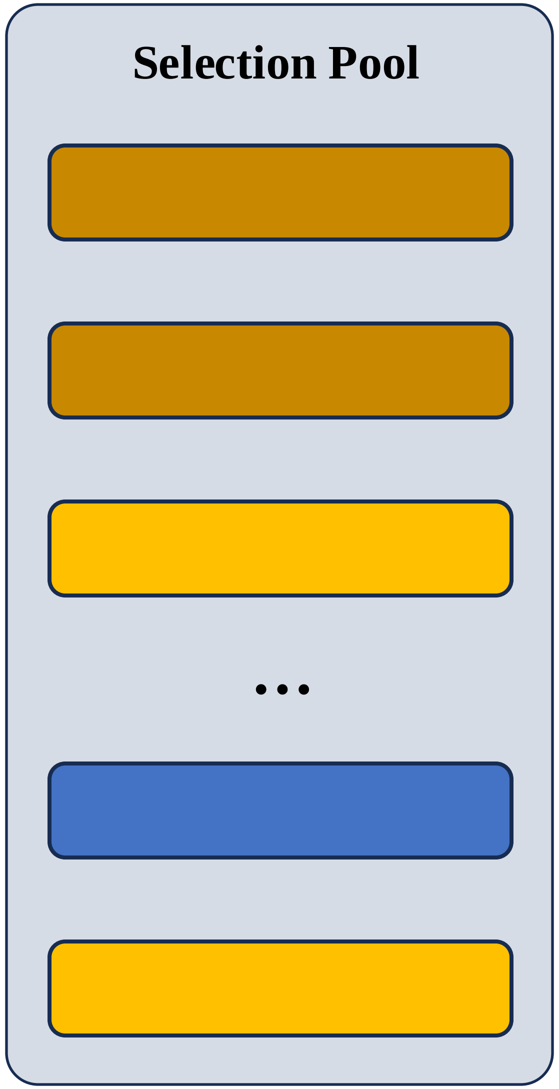
# Genetic Algorithm

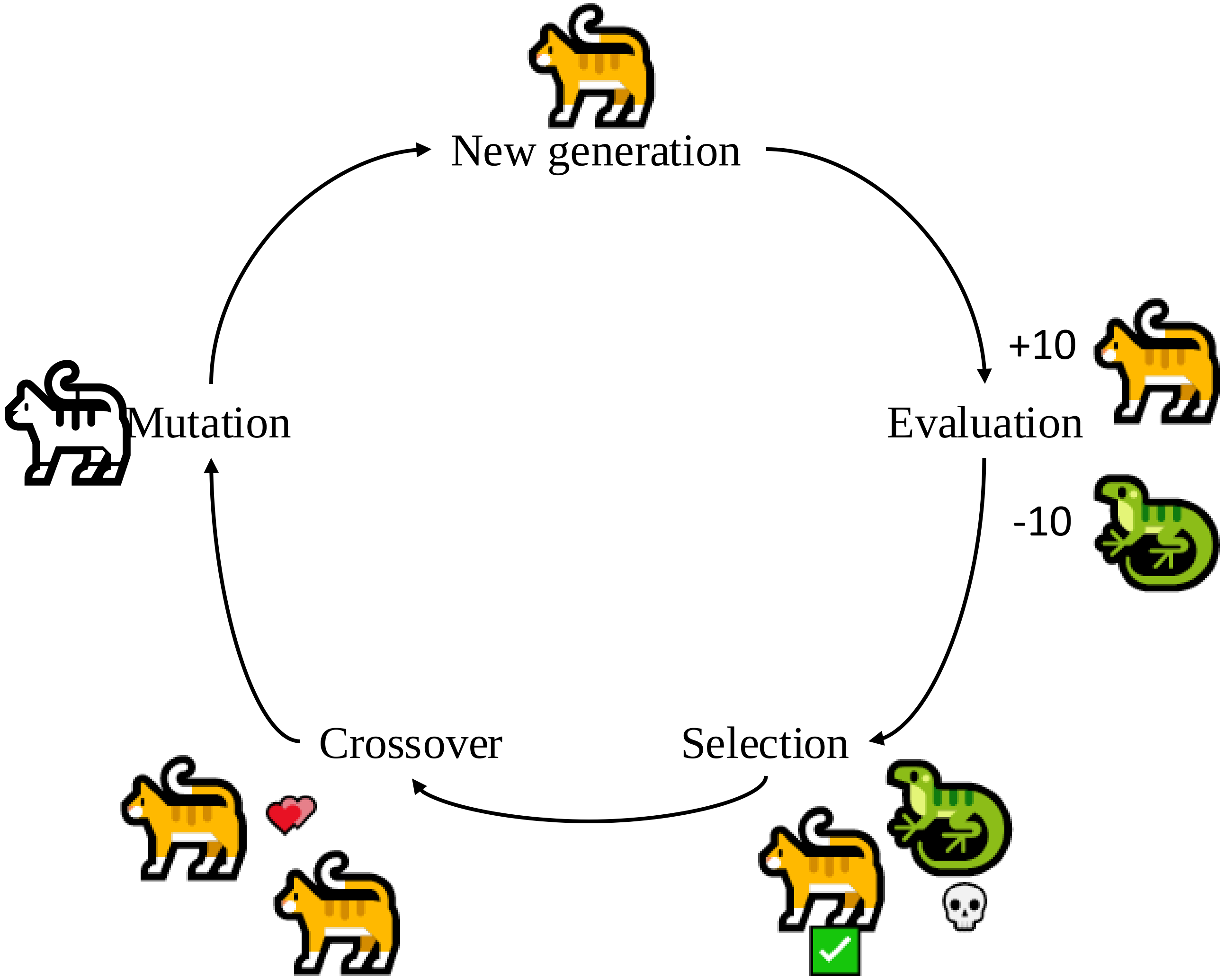
## ❖ Selection



# Genetic Algorithm

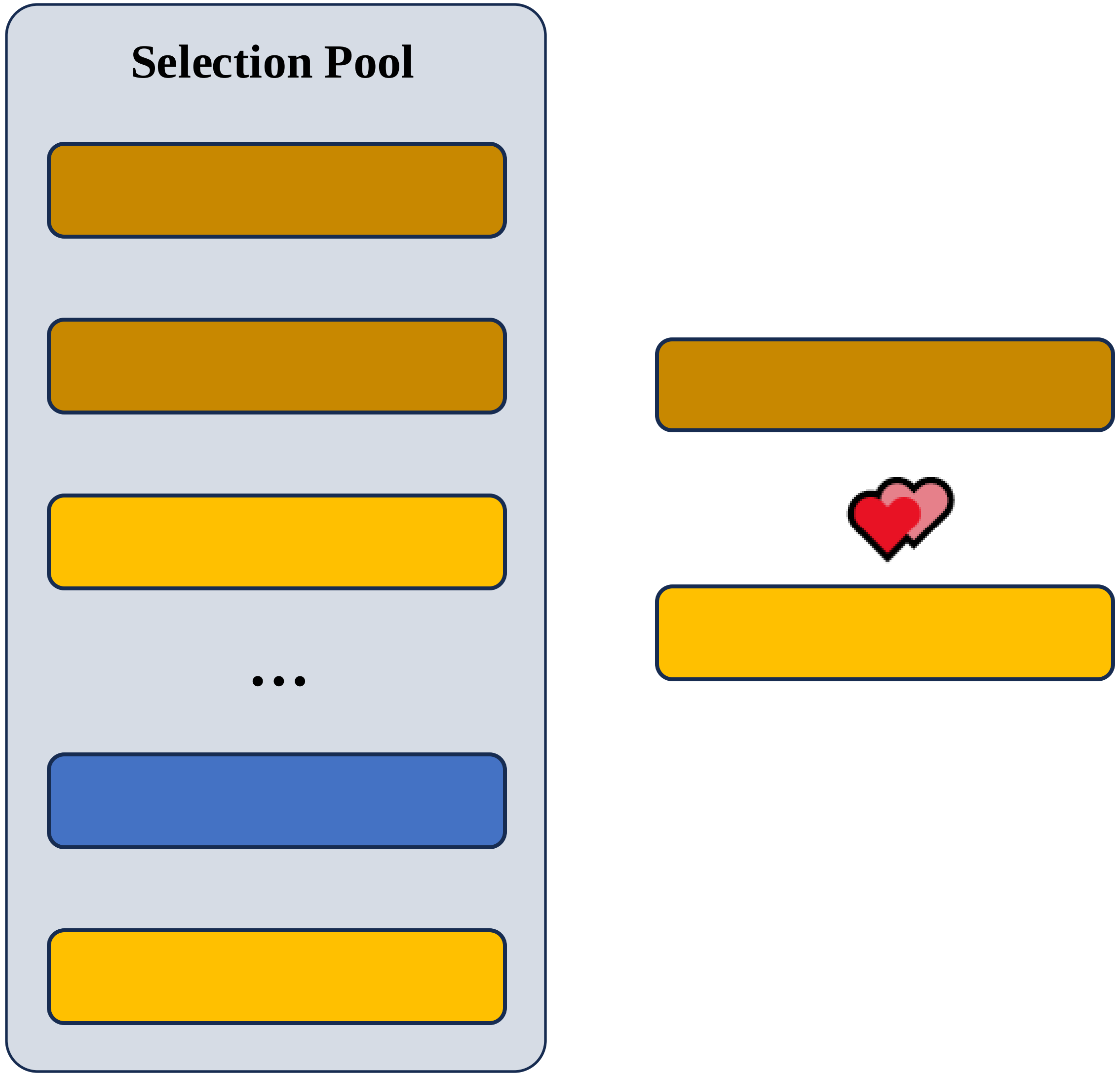
## ❖ How can we create an offspring?



# Genetic Algorithm

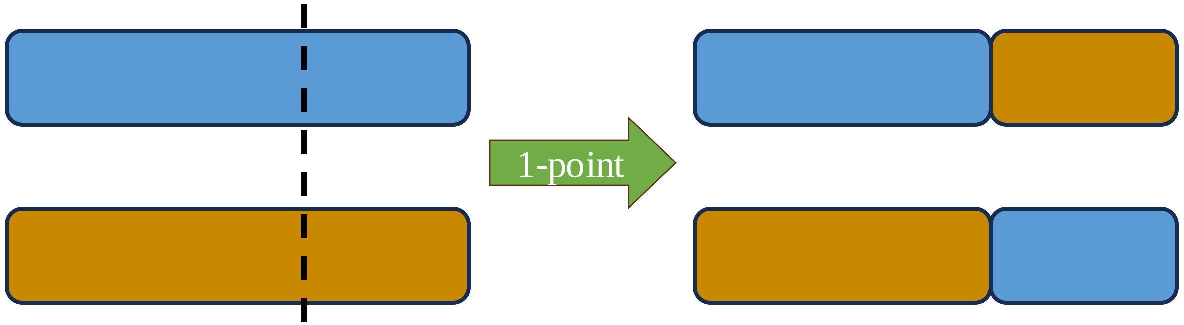
## ❖ Crossover



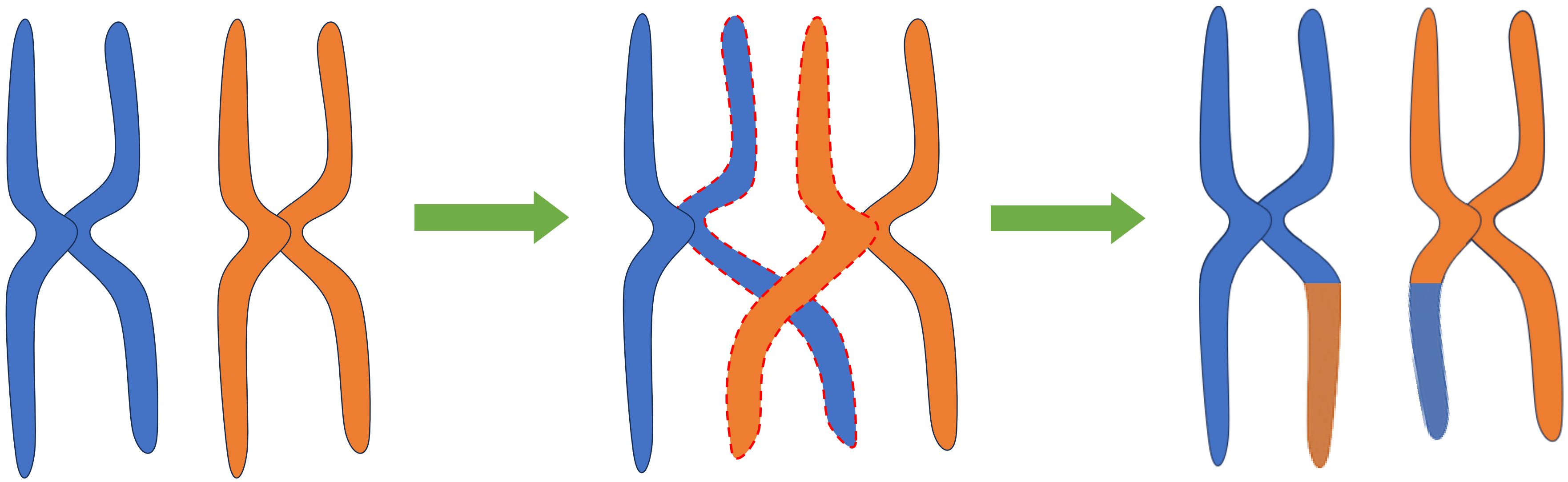

**Crossover** in a genetic algorithm is the process of combining the genes of two parent individuals to create offspring, exchanging genetic material to explore new solutions.

# Genetic Algorithm

## ❖ One-point Crossover - 1X

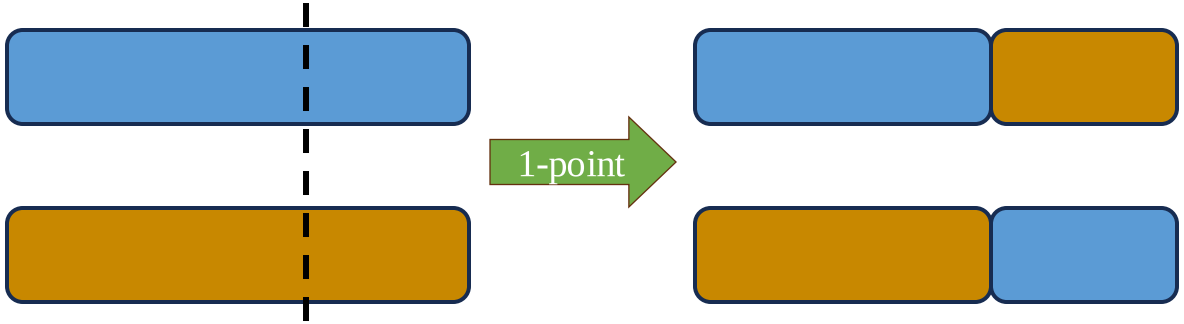


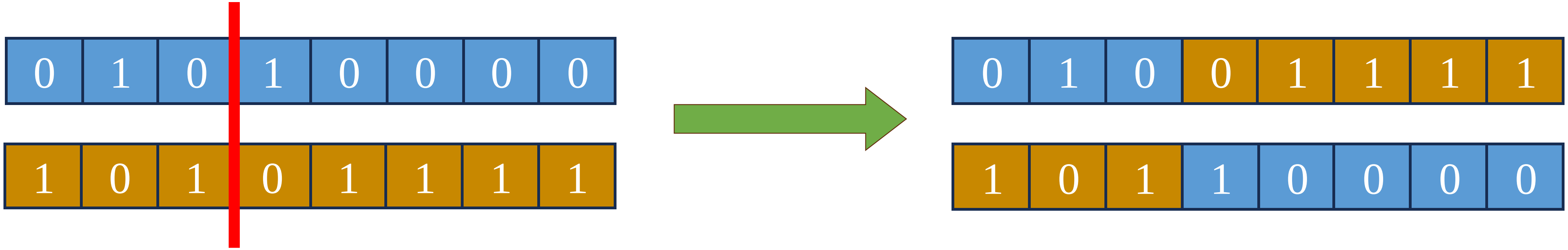

**1-point crossover** swaps genetic material between two parents at a single point, combining segments from both to create offspring.



# Genetic Algorithm

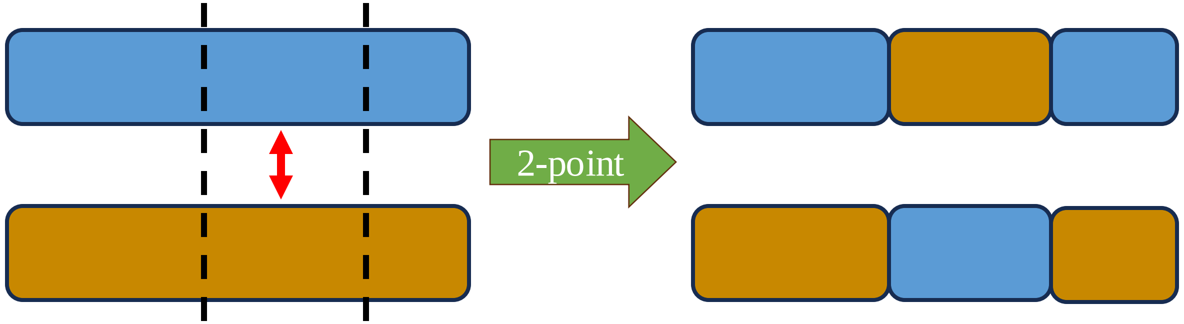
## ❖ One-point Crossover Example



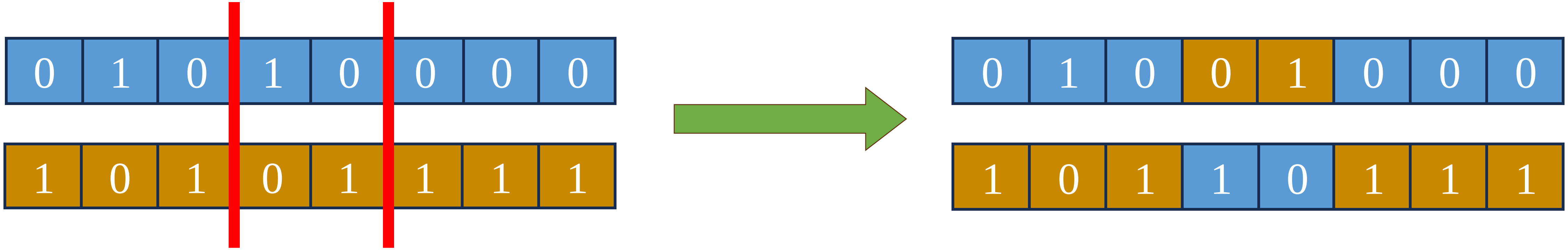



# Genetic Algorithm

## ❖ Two-point Crossover - 2X

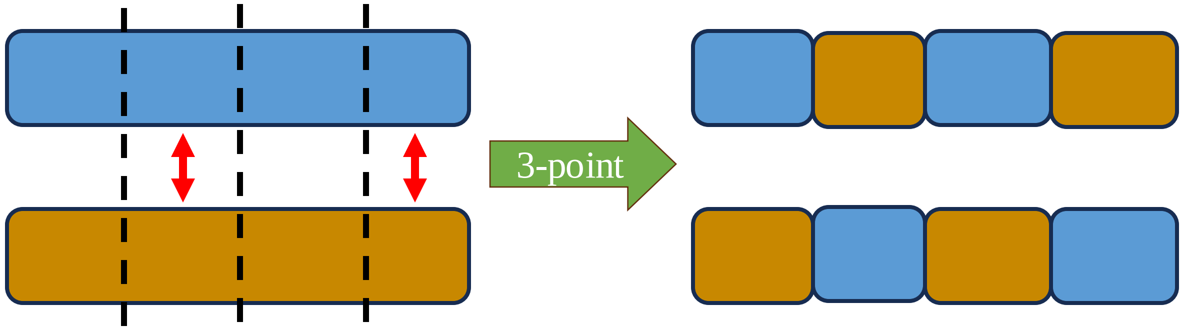



**2-point crossover** swaps genetic material between two parents by selecting two crossover points, exchanging the segment between them to create offspring.



# Genetic Algorithm

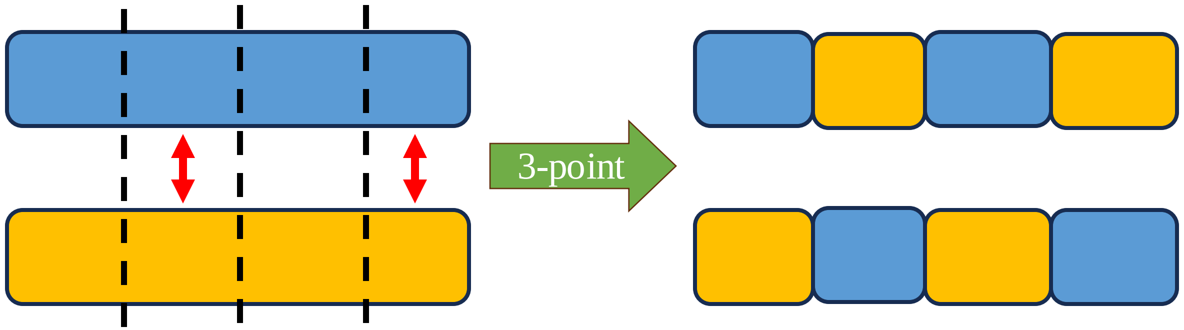
## ❖ Three-point Crossover - 3X

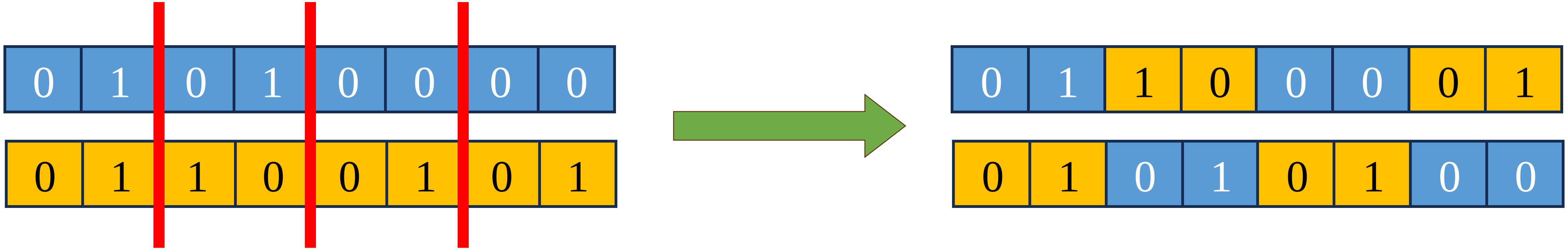



**3-point crossover** exchanges genetic material between two parents by selecting three crossover points, alternating the segments between them to create offspring.

# Genetic Algorithm

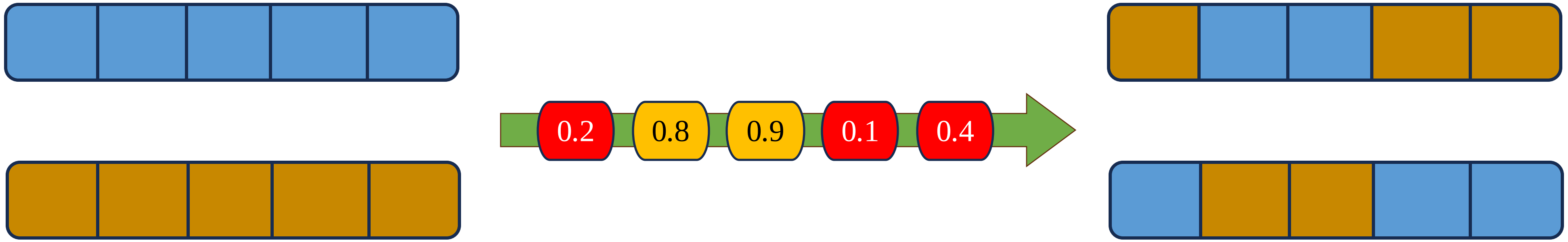
## ❖ Three-point Crossover Example



# Genetic Algorithm

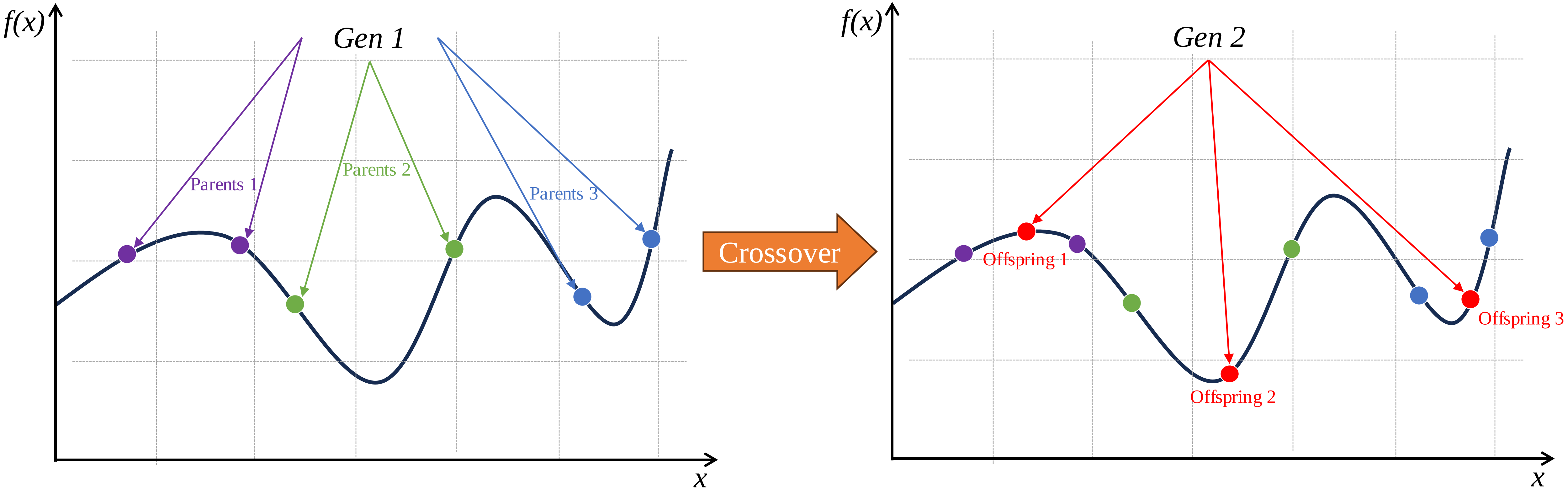
## ❖ Uniform Crossover




**Uniform crossover** exchanges genetic material between two parents by randomly deciding for each gene whether to swap it, resulting in offspring with a mix of genes from both parents.

# Genetic Algorithm

## ❖ New offsprings result in new generation

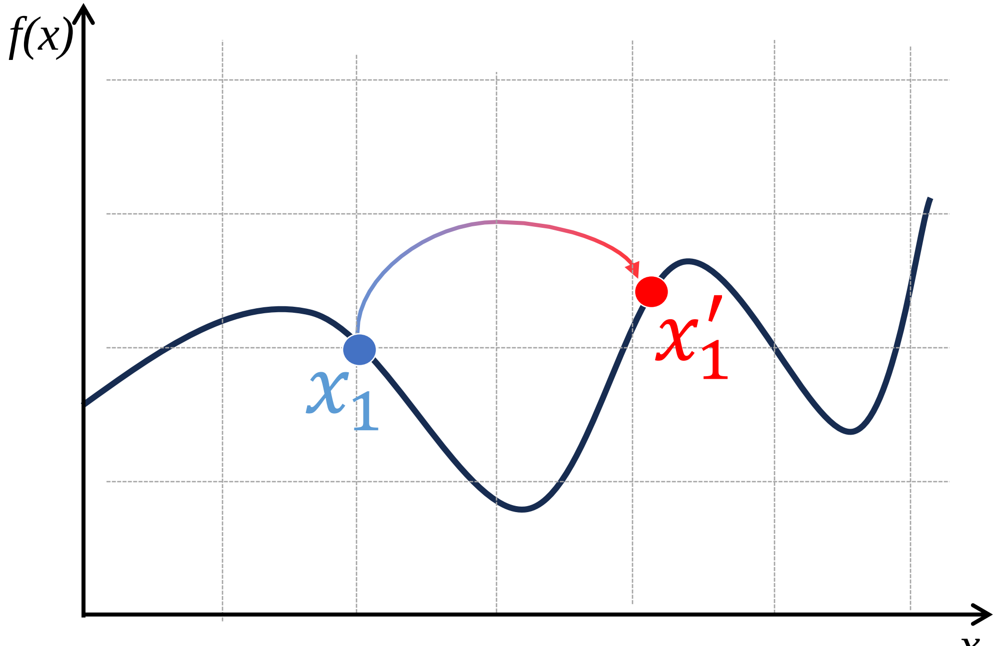



In theory, the new offsprings will be better than their parents.

# Genetic Algorithm

## ❖ Mutation

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $x_1$ | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| $x_{1-mutation}$ | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| $x_1'$ | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |




**Mutation:** Introduces small random changes (or "random genes") in the offspring to maintain diversity within the population.

# Genetic Algorithm

## ❖ Mutation: How it works

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $x_1$ | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| $x_{1-mutation}$ | 0.52 | **0.02** | 0.78 | 0.15 | **0.04** | 0.59 | 0.81 | **0.01** |
| $x_1'$ | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

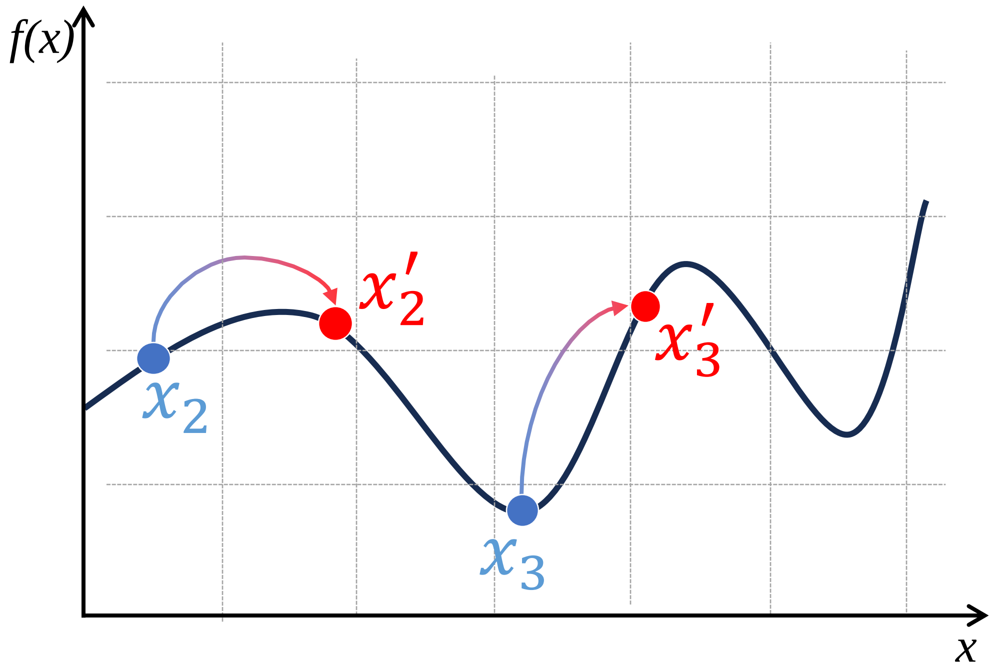
***mutation_rate=0.05***

# Genetic Algorithm

## ❖ Mutation Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $x_2$ | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| $x_2'$ | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $x_3$ | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| $x_3'$ | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |



# Genetic Algorithm

## ❖ Mutation Example

**After Mutation**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

**Initial Population**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

# Genetic Algorithm

## ❖ Fitness Evaluation

**One – max** Fitness: $f(x) = \sum_{i=0}^{n} x_i$

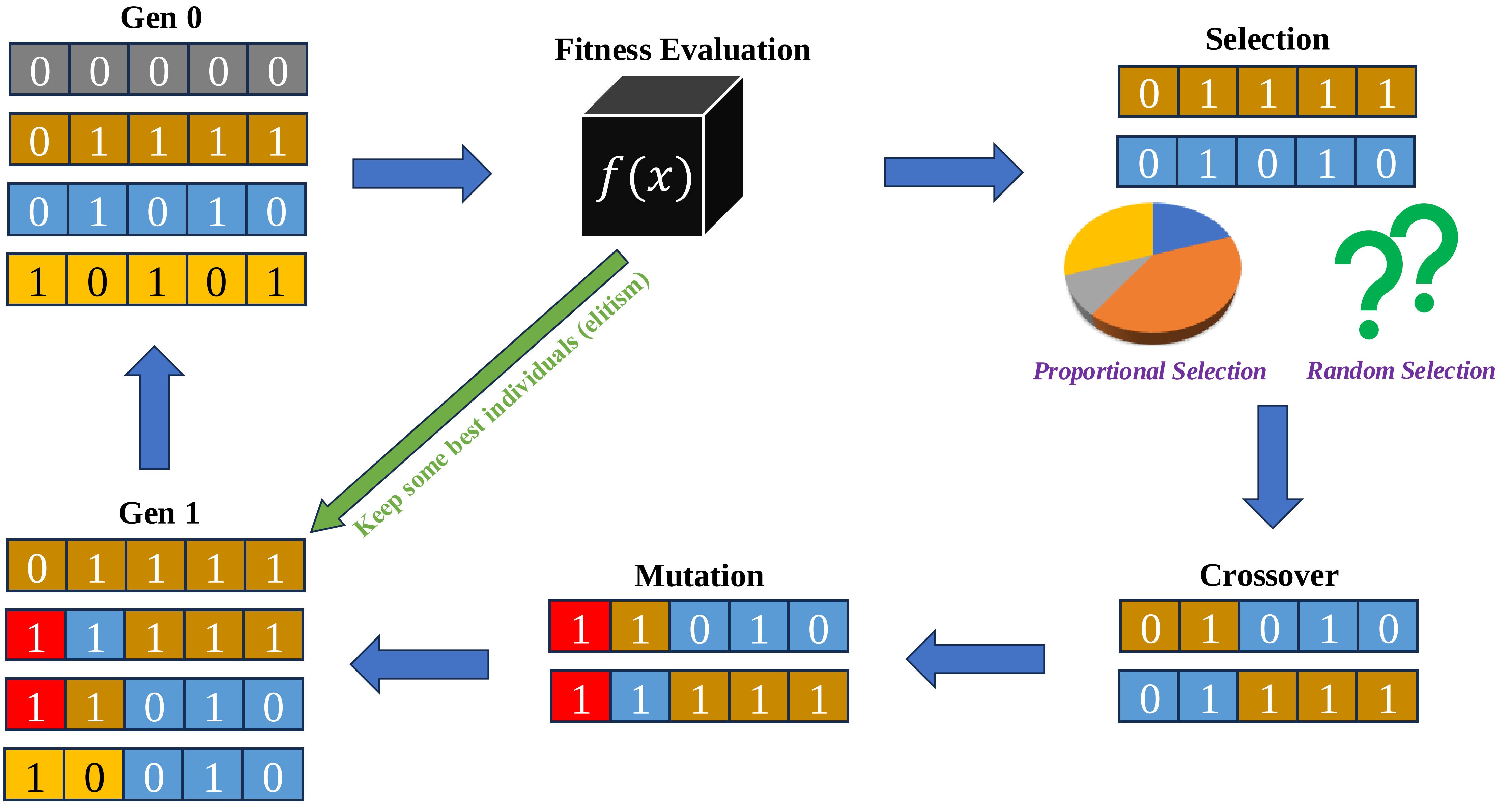
**After Mutation**

| | | | | | | | | Fitness |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 2 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 5 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 5 |

**Initial Population**

| | | | | | | | | Fitness |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 4 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 2 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 6 |

# Genetic Algorithm



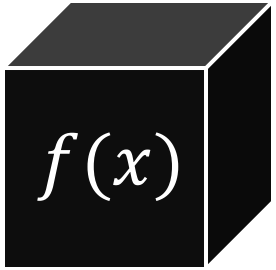
# QUIZ

# Code Implementation

# Code Implementation

## ❖ Problem Statement

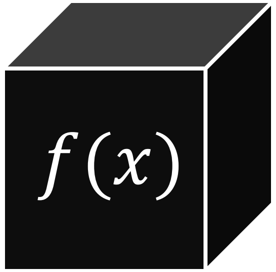
**Description:** Solve the **One-Max problem** using a genetic algorithm. Implement a genetic algorithm that evolves a population of binary strings, where the goal is to maximize the number of 1s in each string. Your algorithm should include the following steps: initialize a population of random binary strings, evaluate fitness based on the number of 1s, apply selection, crossover, and mutation operators, and iterate until a string of all 1s is found or a specified number of generations is reached.




**One – max** Fitness: $f(x) = \sum_{i=0}^{n} x_i$

# Code Implementation

## ❖ One-max Fitness Evaluation




One – max Fitness: $f(x) = \sum_{i=0}^{n} x_i$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

$f(x) = 0 + 0 + 0 + 0 + 0 + 1 + 1 + 0$
$= 2$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

$f(x) = 0 + 0 + 1 + 0 + 0 + 0 + 1 + 0$
$= 2$

# Code Implementation

## ❖ One-max Fitness Evaluation

$$\text{One} - \text{max Fitness: } f(x) = \sum_{i=0}^{n} x_i$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

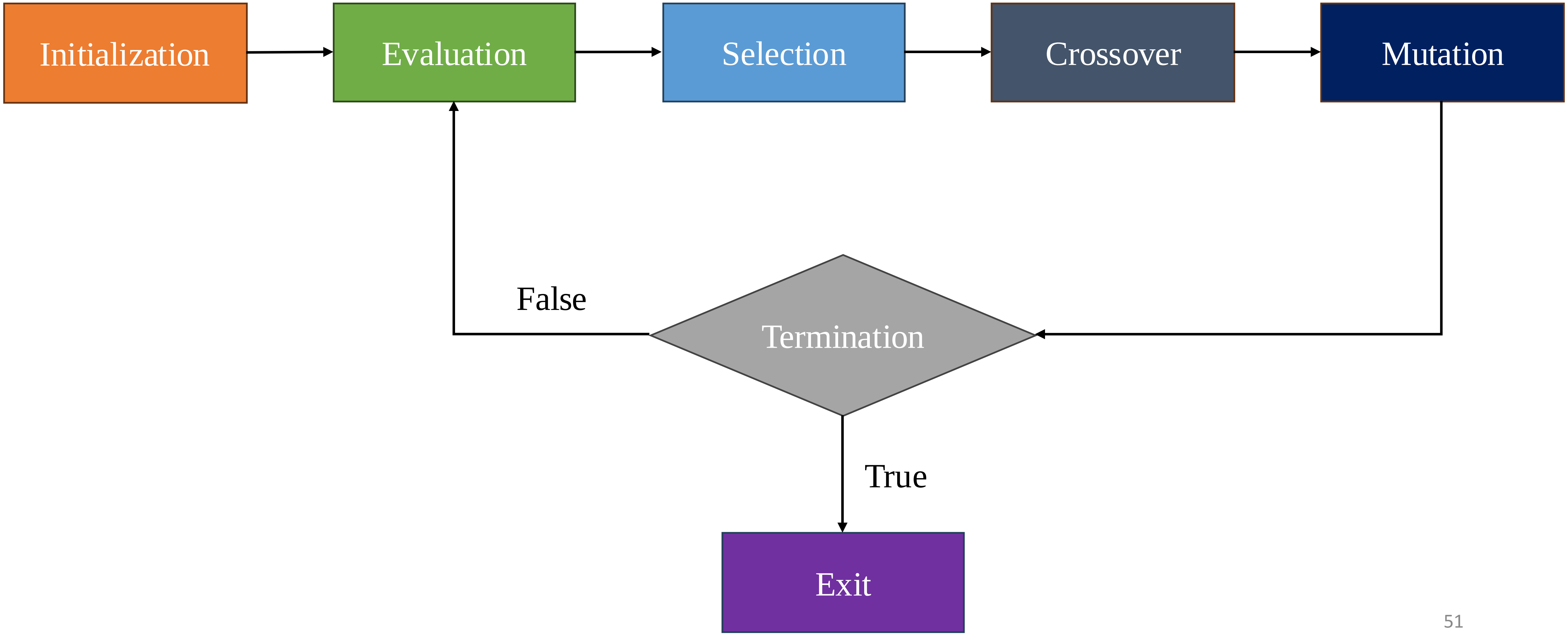
$$f(x) = 0 + 0 + 1 + 0 + 1 + 1 + 1 + 1 = 5$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

$$f(x) = 0 + 1 + 1 + 1 + 1 + 1 + 0 + 0 = 5$$

# Code Implementation

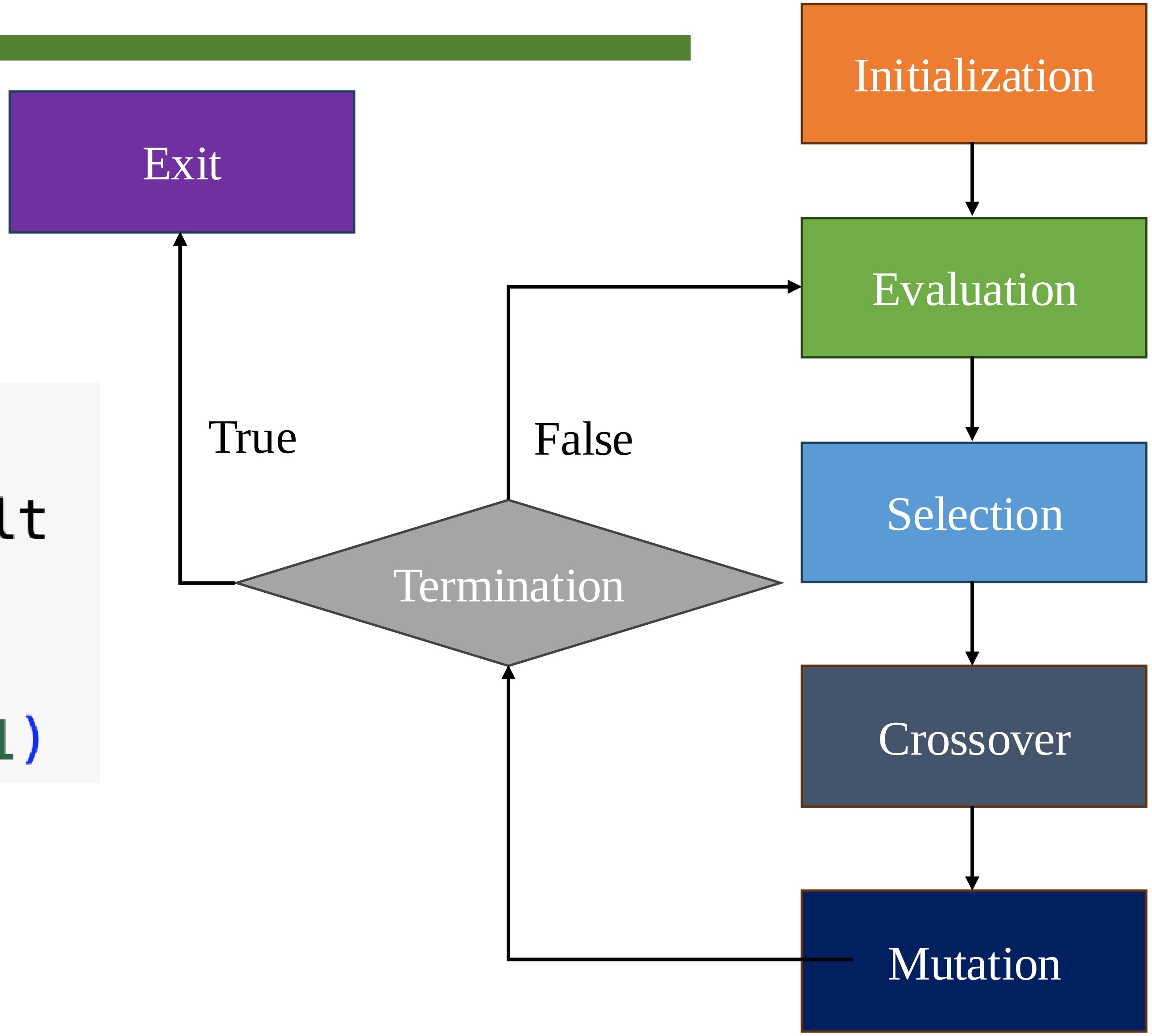
## ❖ Simple GA Pipeline



# Code Implementation

## ❖ Step 1

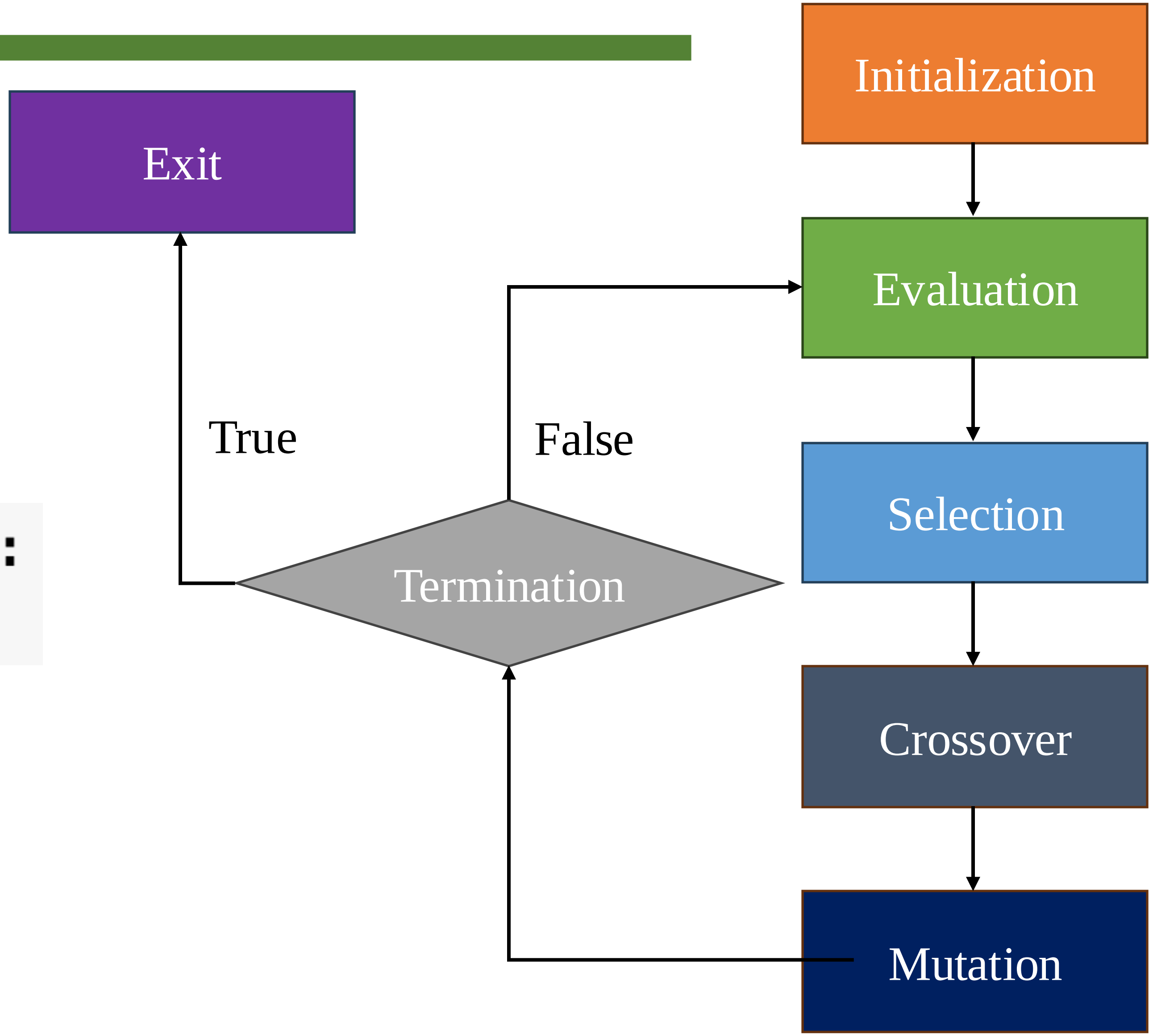
```python
import random
import matplotlib.pyplot as plt

def generate_random_value():
    return random.randint(0, 1)
```



# Code Implementation

❖ Step 2

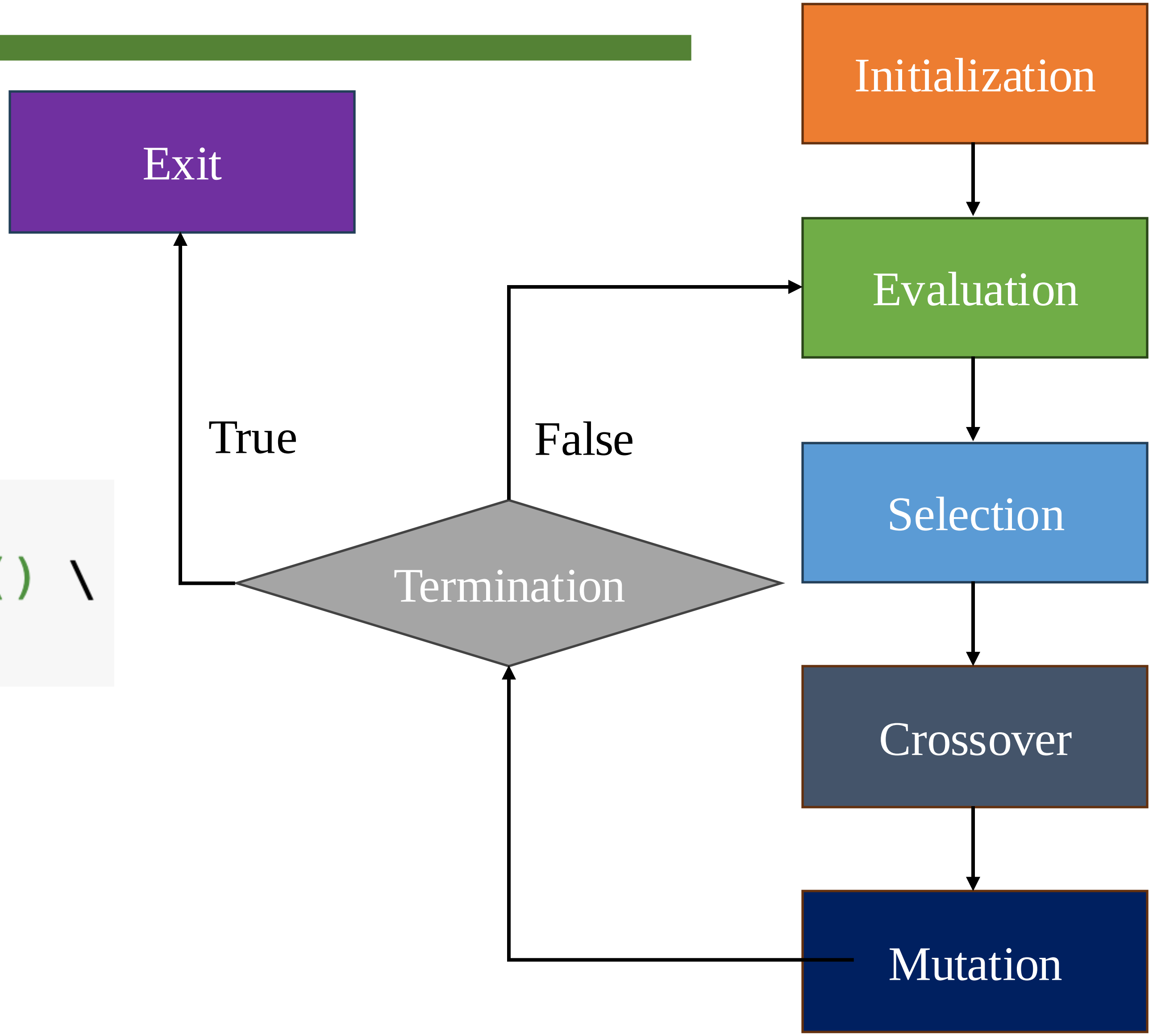
```python
def compute_fitness(individual):
    return sum(individual)
```



# Code Implementation

## ❖ Step 3

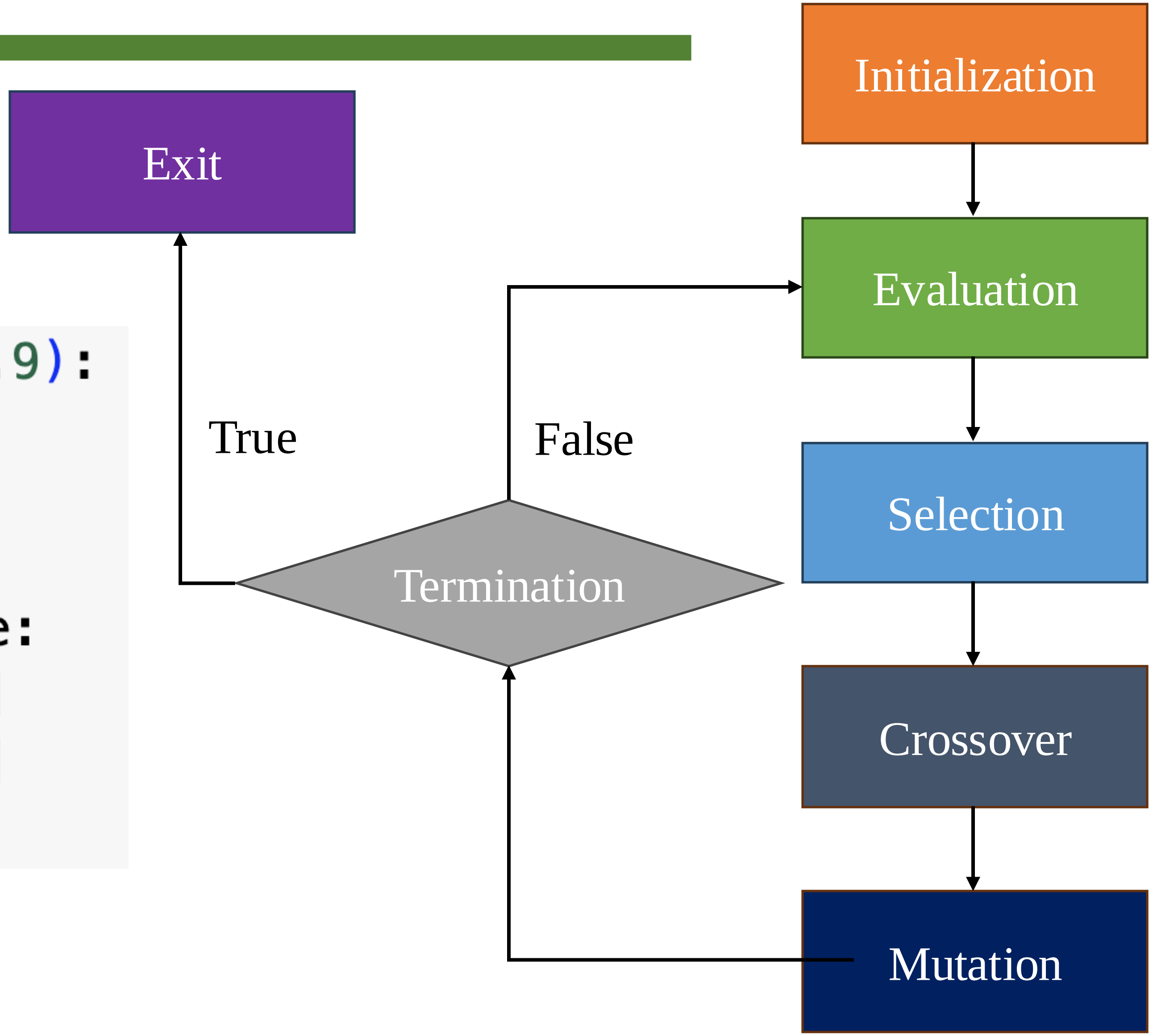
```
def create_individual(n):
    return [generate_random_value() \
            for _ in range(n)]
```



# Code Implementation

❖ **Step 4**

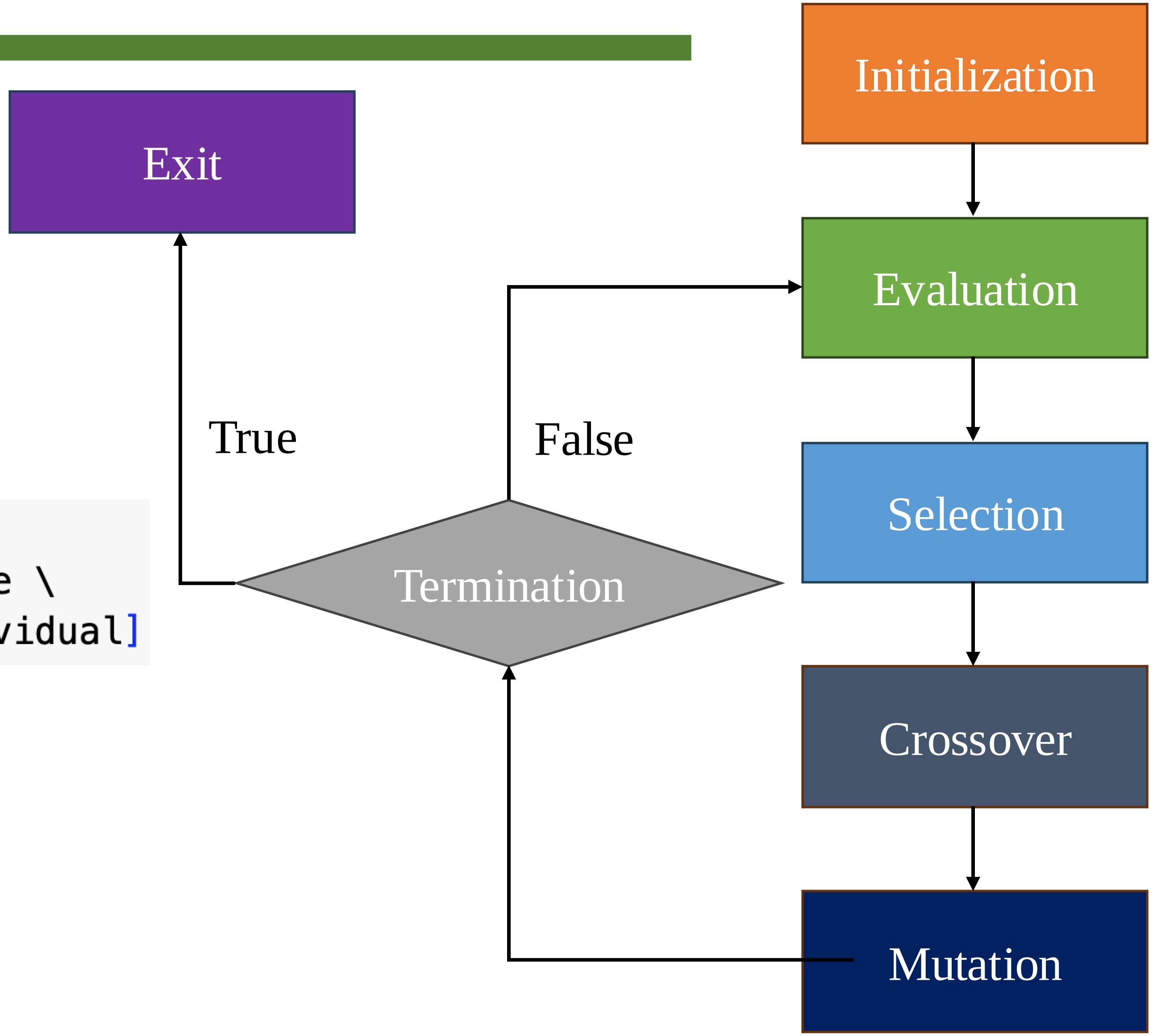
```python
def crossover(ind1, ind2, rate=0.9):
    ind1_new = ind1.copy()
    ind2_new = ind2.copy()
    for i in range(len(ind1)):
        if random.random() < rate:
            ind1_new[i] = ind1[i]
            ind2_new[i] = ind2[i]
    return ind1_new, ind2_new
```



# Code Implementation

## ❖ Step 5

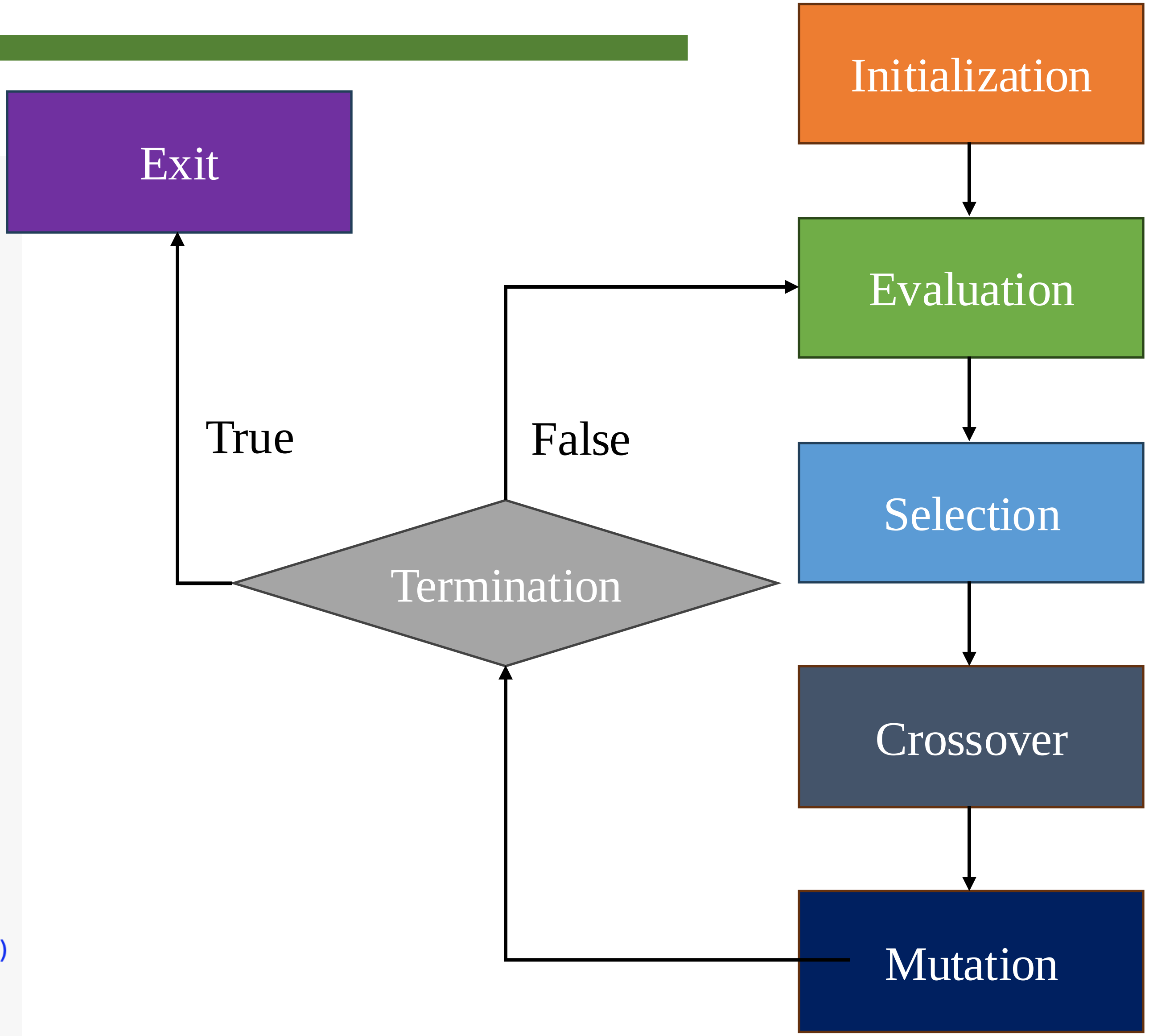
```python
def mutate(individual, rate=0.05):
    return [1 - val if random.random() < rate \
                    else val for val in individual]
```



# Code Implementation

## ❖ Step 6

```python
n = 100  # Length of each individual (binary string)
m = 100  # Population size
generations = 100  # Number of generations
elitism = 2  # Number of elite individuals to carry over

population = [create_individual(n) for _ in range(m)]
fitness_history = []

for gen in range(generations):
    population = sorted(population,
                        key=compute_fitness,
                        reverse=True)

    # Track the best fitness in this generation
    if gen % 10 == 0:
        best_fitness = compute_fitness(population[0])
        fitness_history.append(best_fitness)
        print(f"Generation {gen} - Best Number of 1s: {best_fitness}")

    # Elitism: retain the best individuals
    new_population = population[:elitism]

    while len(new_population) < m:
        parent1, parent2 = selection(population), selection(population)
        child1, child2 = crossover(parent1, parent2)
        new_population.append(mutate(child1))
        new_population.append(mutate(child2))

    population = new_population[:m]
```

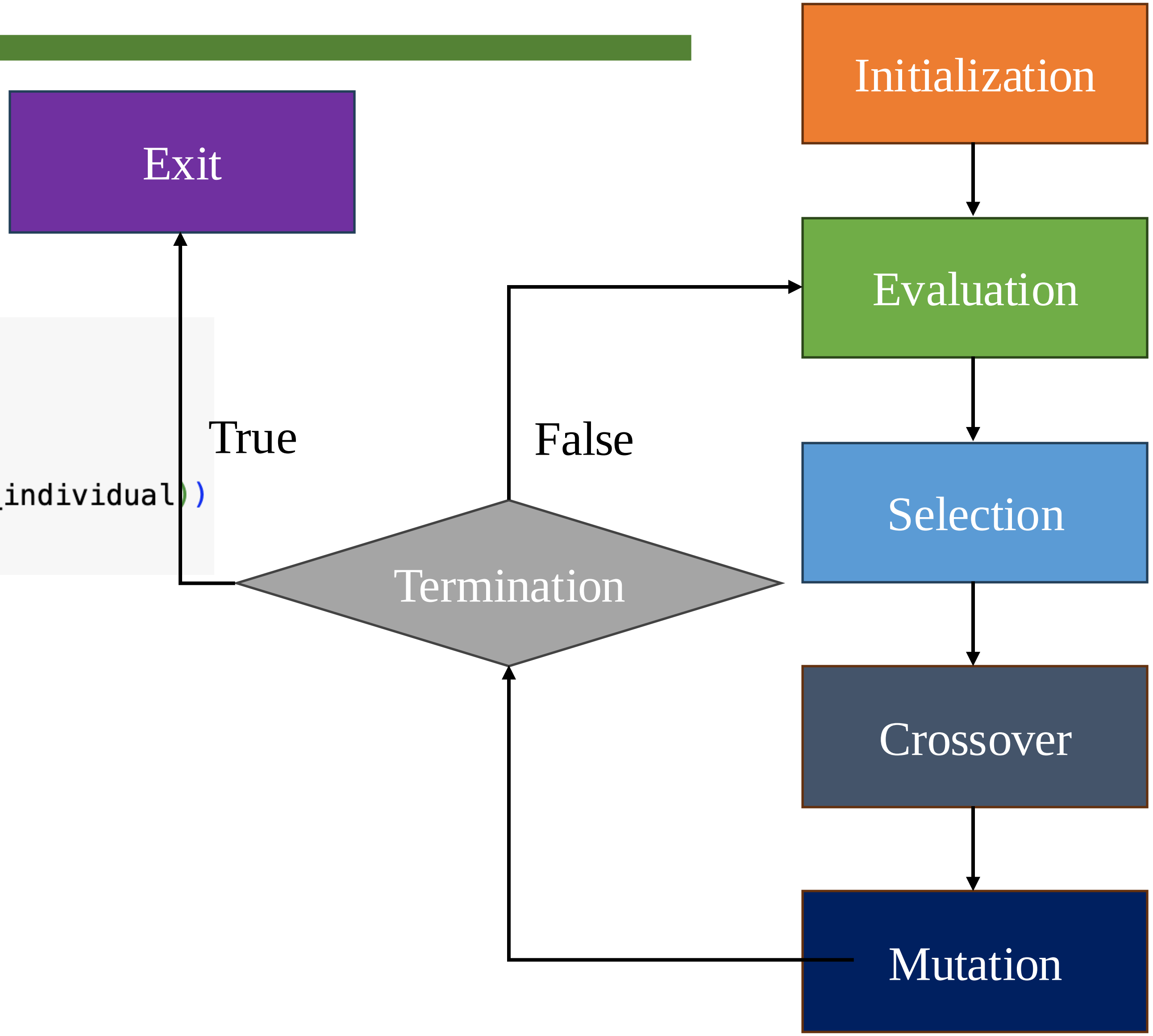


# Code Implementation

## ❖ Step 7

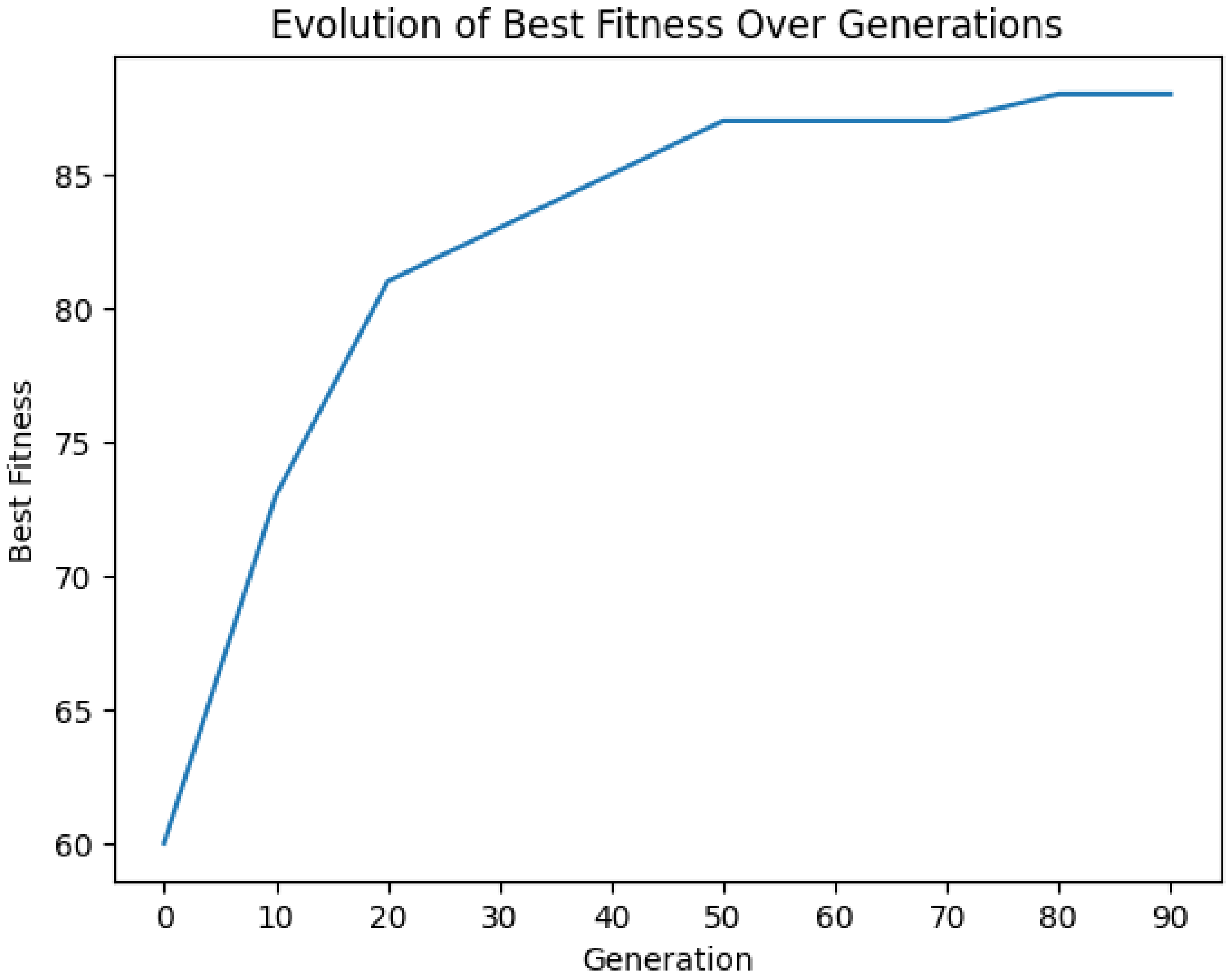
```
best_individual = sorted(population,
                         key=compute_fitness,
                         reverse=True)[0]
print("Best individual:", best_individual)
print("Best fitness (number of 1s):", compute_fitness(best_individual))
print("Fitness history:", fitness_history)
```

```
Generation 0 - Best Number of 1s: 60
Generation 10 - Best Number of 1s: 73
Generation 20 - Best Number of 1s: 81
Generation 30 - Best Number of 1s: 83
Generation 40 - Best Number of 1s: 85
Generation 50 - Best Number of 1s: 87
Generation 60 - Best Number of 1s: 87
Generation 70 - Best Number of 1s: 87
Generation 80 - Best Number of 1s: 88
Generation 90 - Best Number of 1s: 88
```



# Code Implementation

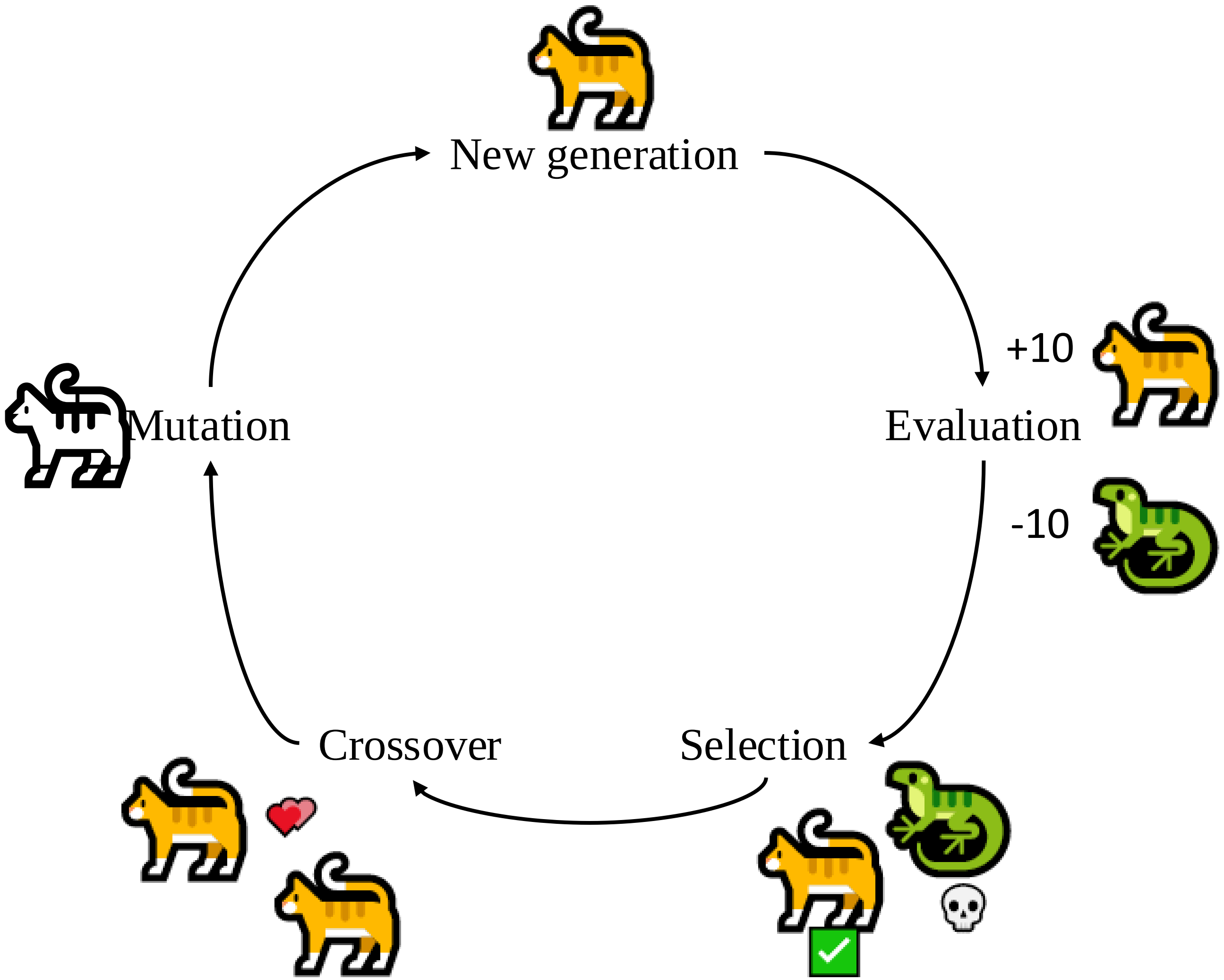
## ❖ Fitness over generations



# Summarization and QA

# Summarization

## ❖ Content




**In this lecture, we have discussed:**

1. Introduction to Genetic Algorithm.
2. Discuss about how a simple Genetic Algorithm works.
3. Implement a simple Genetic Algorithm pipeline to solve one-max problem.

# Question

?


Thank you!