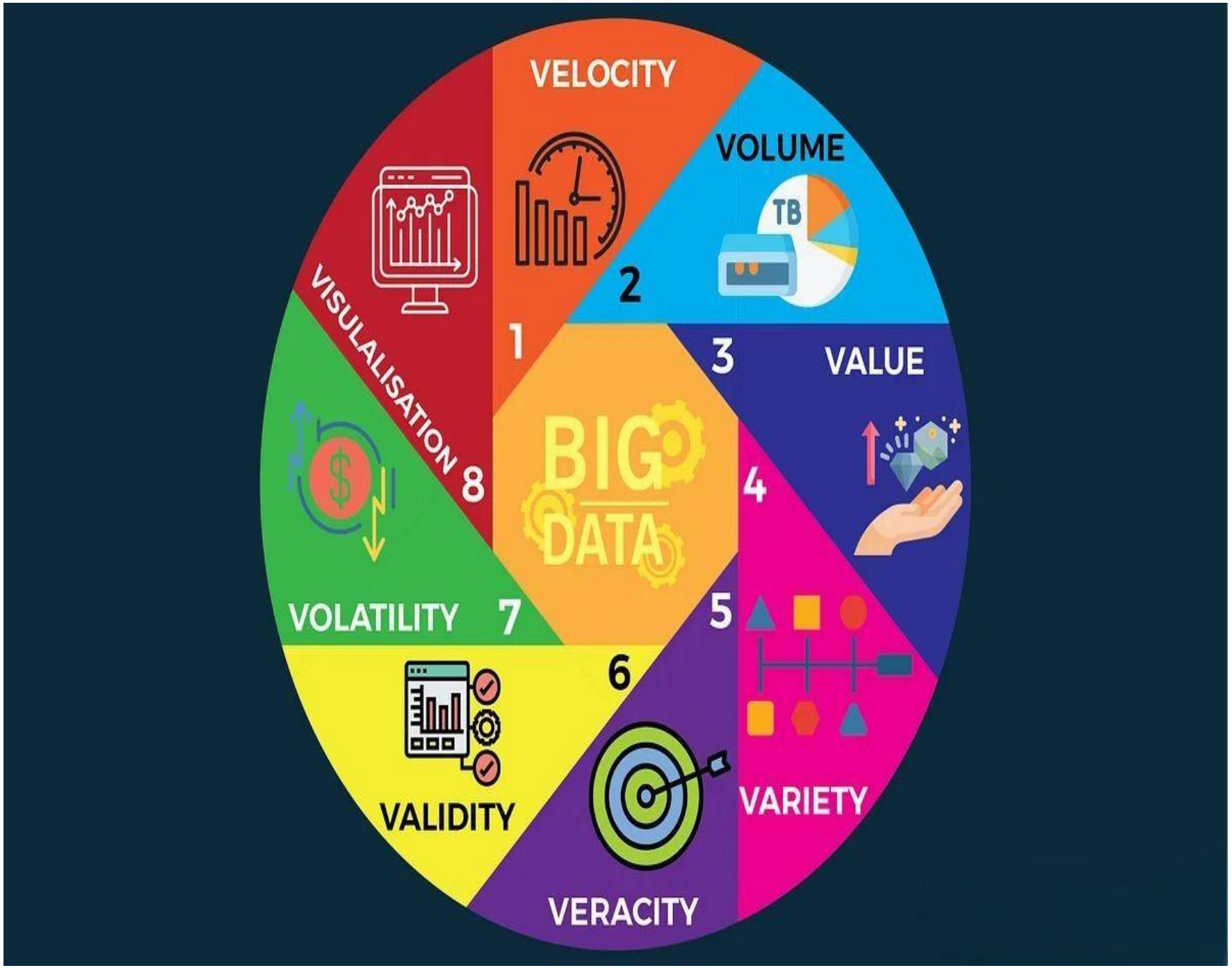


CS 3032: Big Data

Lec-10

In this Discussion . . .

- Data Streams
  - Bloom Filter
    - Introduction
    - Requirements
    - What is a Bloom Filter
    - How does it work ?
    - Special Properties
    - How to add an item to the bloom filter?
    - How to check the membership of an item in the bloom filter?
    - How to update an item to the bloom filter?

Introduction

- Data structures such as HashSet can be used in a small data set to test if an item is a member of the data set.
- However, the operation to test if an item is a member of a large dataset can be expensive. The time complexity and space complexity can be linear in the worst case.

Probabilistic data structures offer constant time complexity and constant space complexity at the expense of providing an answer that is non-deterministic.

Requirements

- Design a data structure with the following characteristics:
  - constant time complexity to test membership
  - a small amount of memory to test membership
  - insert and query operations should be parallelizable
  - test membership can yield approximate results

What is a Bloom Filter?

- A Bloom filter is a space-efficient probabilistic data structure that is used to test whether an item is a member of a set.
- The bloom filter will always say yes if an item is a set member.
- However, the bloom filter might still say yes although an item is not a member of the set (false positive).

What is a Bloom Filter? (Contd.)

- The items can be added to the bloom filter but the items cannot be removed.
- **The bloom filter supports the following operations:**
  - adding an item to the set
  - test the membership of an item in the set
  - updating an item to the set

We can't delete from a bloom filter

Bloom Filter

The Bloom filter is a space efficient, probabilistic data structure, designed to test the membership of elements to a set.

- **The trade-off for being a space efficient data structure is it may return false positives, but always returns definite negatives: Meaning Bloom filters can accurately test an element for non-membership to a set, but can only with probability test an element for membership.**

Bloom Filter (Contd.)

The Bloom filter is a space efficient, probabilistic data structure, designed to test the membership of elements to a set.

- **Bloom filters find application in circumstances where testing for non-membership saves resources such as calls to a web server, checking a proxy cache. Google uses Bloom filters in the Chrome browser as a preliminary check for malicious URLs.**
- With the rise of big data since the mid-2000s, there's been increased interest in Bloom filter.

Bloom Filter (Contd.)

Bloom filter is a space-efficient probabilistic data structure conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set.

- **False positive matches are possible, but False Negatives are not – in other words, a query returns either "possibly in set" or "definitely not in set"**
- **False Positive = "possibly in set" or "definitely not in set"**
  False Negative = "possibly not in set" or "definitely in set"

Bloom Filter (Contd.)

Bloom filter is a space-efficient probabilistic data structure conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set.

False Positive = "possibly in set" or "definitely not in set"
False Negative = "possibly not in set" or "definitely in set"

Overview
x : An element
S: A set of elements
Input: x, S **Output:**
-TRUE if x in S
-FALSE if x not in S

Bloom Filter (Contd.)

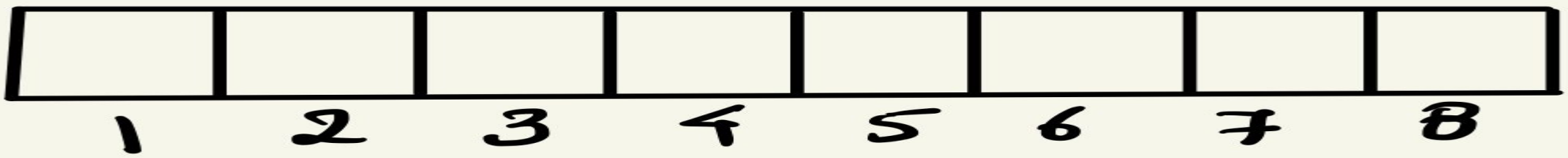
The Bloom filter is a space efficient, probabilistic data structure, designed to test the membership of elements to a set.

- To understand Bloom Filters better, let's first discuss about the two concepts Bloom Filter depends on:

1. Bit Array
2. Hash Functions

Bloom Filter (Contd.)

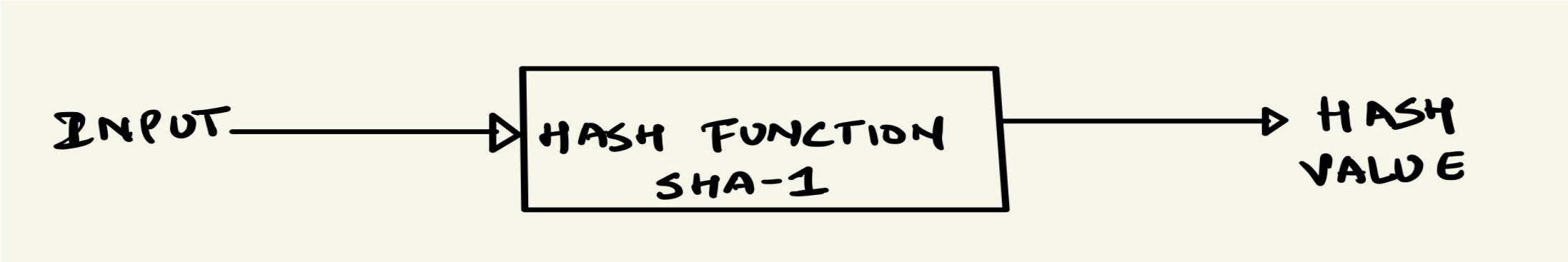
- **Bit Array:** It is an array data structure that stores only boolean values in it. It is used to map values of some domain with {0, 1} in the bit array.



This is an **8-bit** bit array.

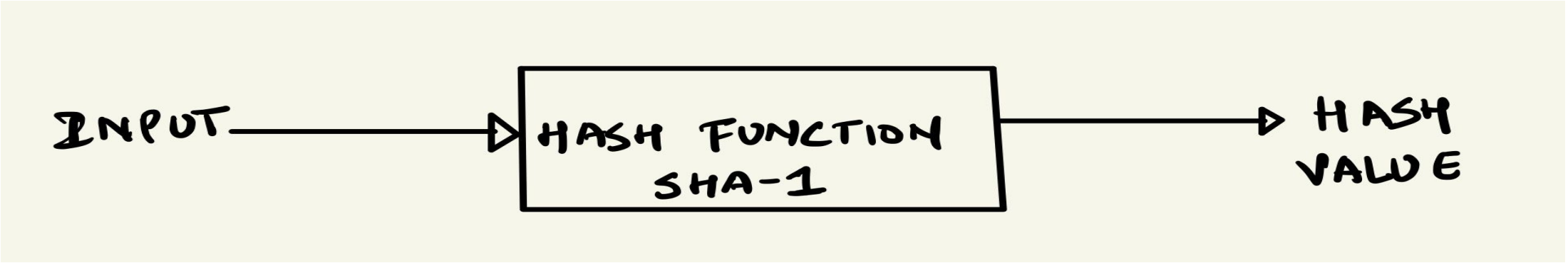
Bloom Filter (Contd.)

- **Hash Functions :** Hash functions, like any other function, take input and apply some algorithm to change the input to the output called hash value.



Bloom Filter (Contd.)

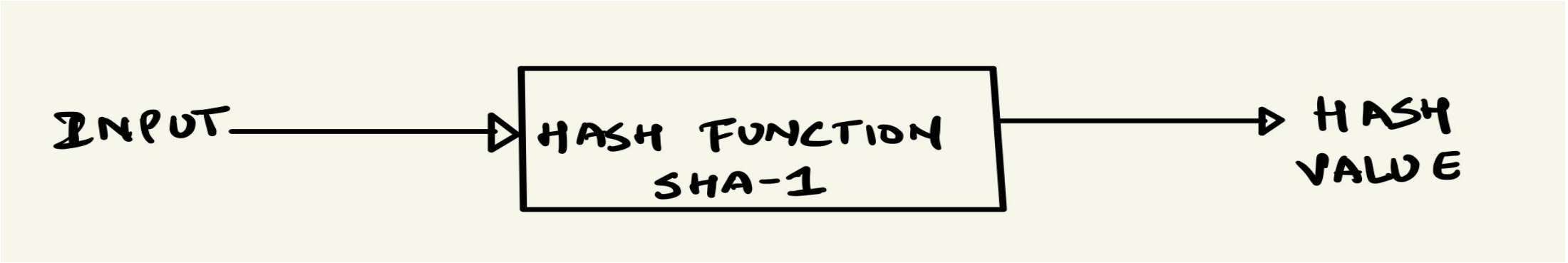
- **Hash Functions :** Hash functions, like any other function, take input and apply some algorithm to change the input to the output called hash value.



- There are various applications of the hash value, one of the most common ones is storing the hash value in a hash table for faster retrieval.
- One example of the algorithm applied to transform the input is SHA-1 (Secure Hash Algorithm 1).

Bloom Filter (Contd.)

- **Hash Functions :** Hash functions, like any other function, take input and apply some algorithm to change the input to the output called hash value.



- **The properties of hash functions that make them ideal to use it for bloom filters are**:
  - The length of the output remains the same no matter what is the input.
  - Every time you pass the same input, it gives the same output.

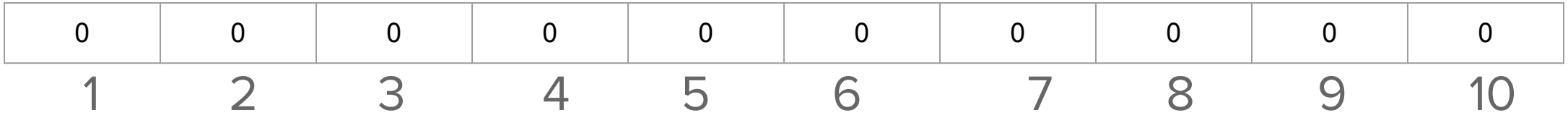
Bloom Filter: How does it work?

- To understand how a bloom filter works, let's take a use case in that we want to store the word "Marvel" in the bit array.
- Let's break down the working of it into steps:

  - Initialize the bit array.
  - Pass the argument into a group of hash functions.
  - Collect the output of each hash function.
  - Apply some mathematical logic to obtain bits to update, in our case we are using the modulus operation.
  - Update the bits obtained in the previous step with value 1.

Bloom Filter: How does it work? (Contd.)

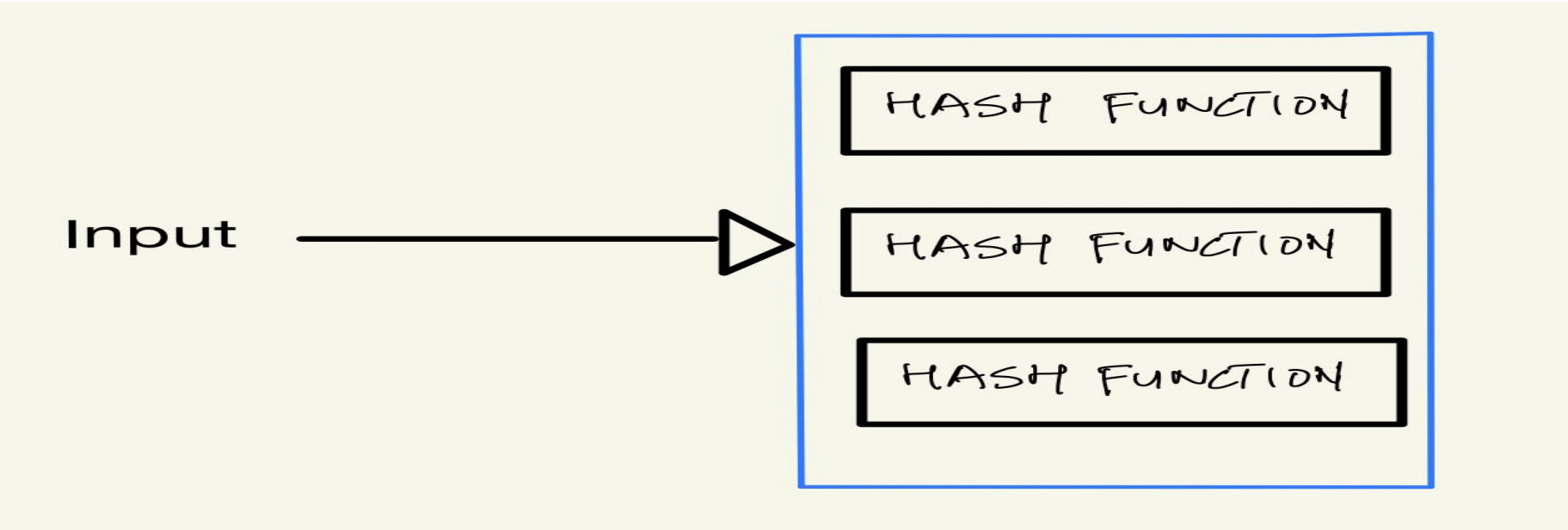
- Let’s look at the diagram to see the same process in action.
- We have initialized the bit array of size 10 with default values as 0.



Empty Bloom Filter

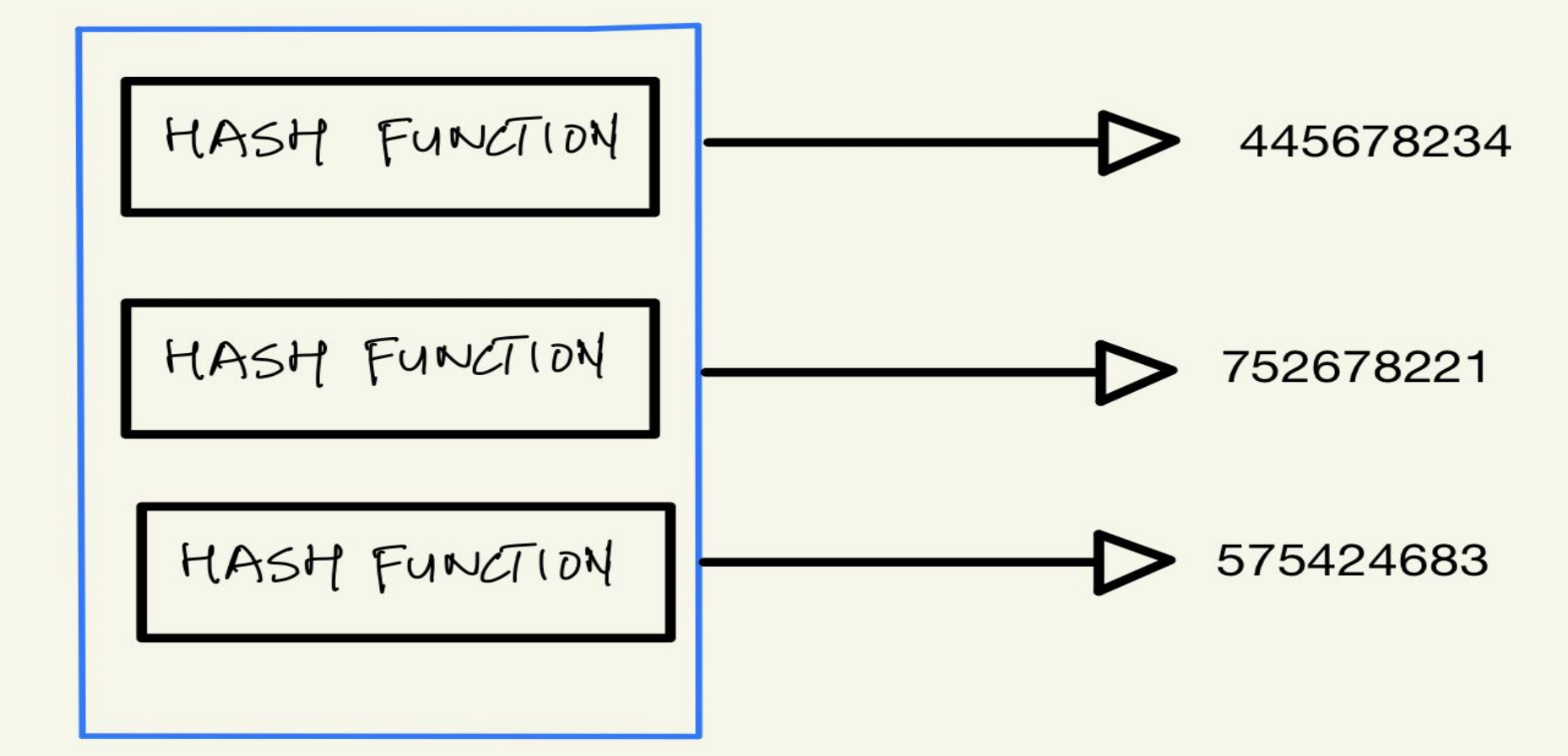
Bloom Filter: How does it work? (Contd.)

- We pass the argument to a group of hash functions:



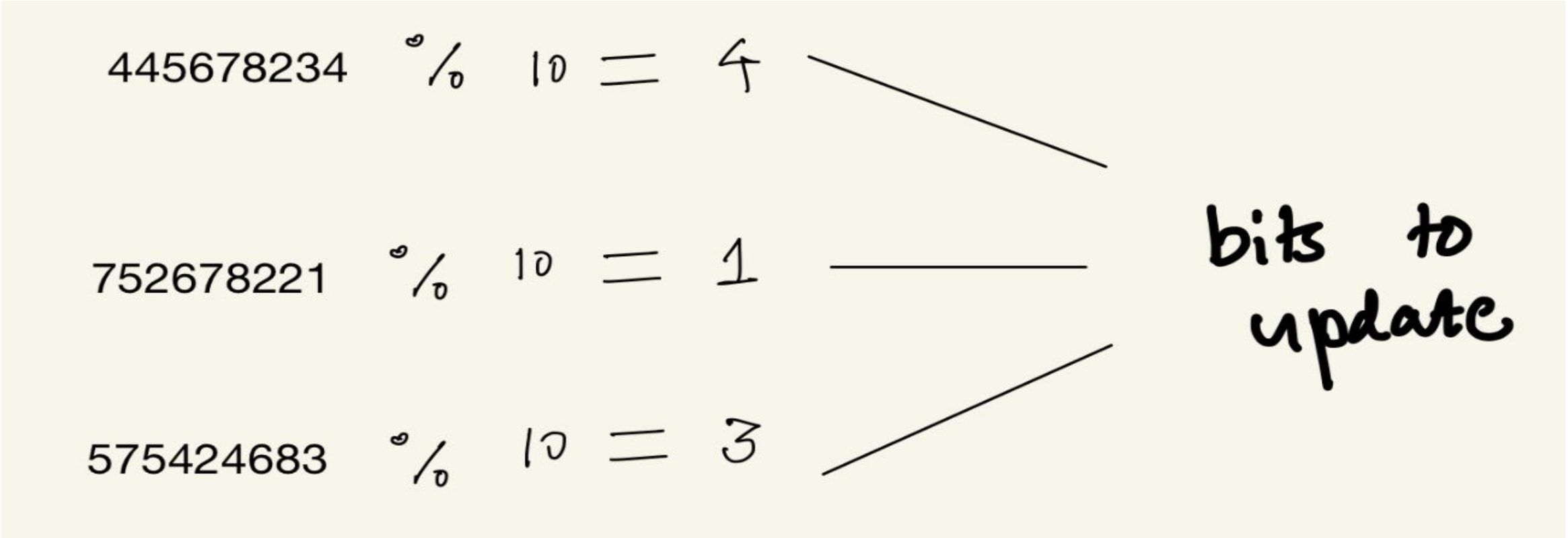
Bloom Filter: How does it work? (Contd.)

- The hash functions give some output:



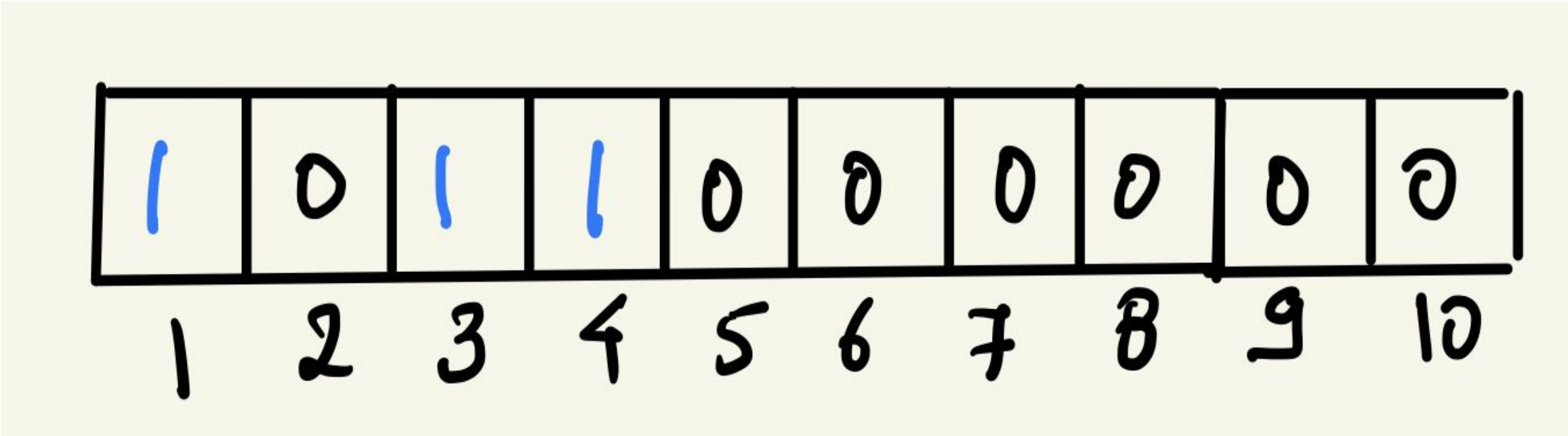
Bloom Filter: How does it work? (Contd.)

- Now we are going to apply the modulus operation to each output of the hash function and we will modulate it by the size of the bit array.



Bloom Filter: How does it work? (Contd.)

- These are the bits that we need to update to store “Marvel” in the bit array.



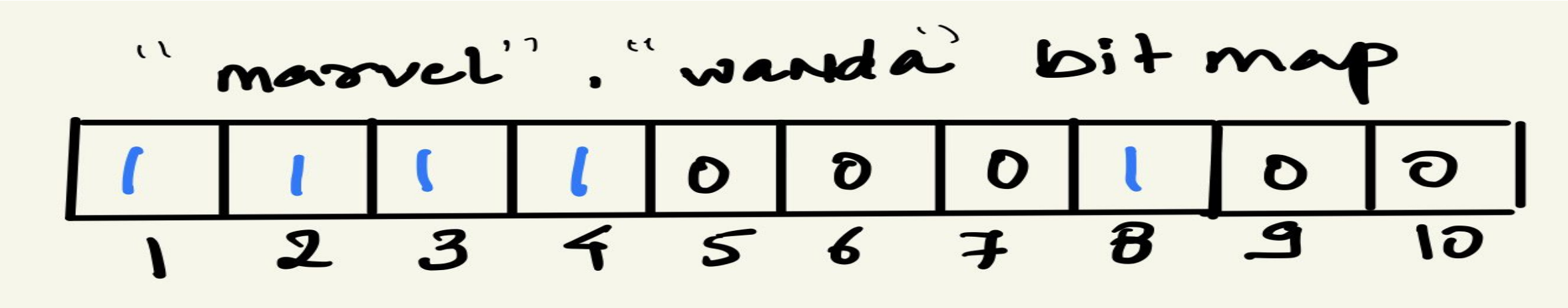
Now if we want to search any word in our bloom filter, we follow the same process except instead of updating the bits with 1, we get the stored value in those bits and if all the values are 1, that means the element is present in the set.

Bloom Filter: Special Properties

- We have already come across the point that bloom filters are being space efficient and probabilistic in nature.
- Now, we have seen that when we want to store any data:
    - We transform that data into bit positions and then update only those bit positions.
    - **This update happens over the same bit array**, and **that makes it space-efficient as storing boolean values in a single-bit array takes up far less space compared to storing actual text values in the database.**

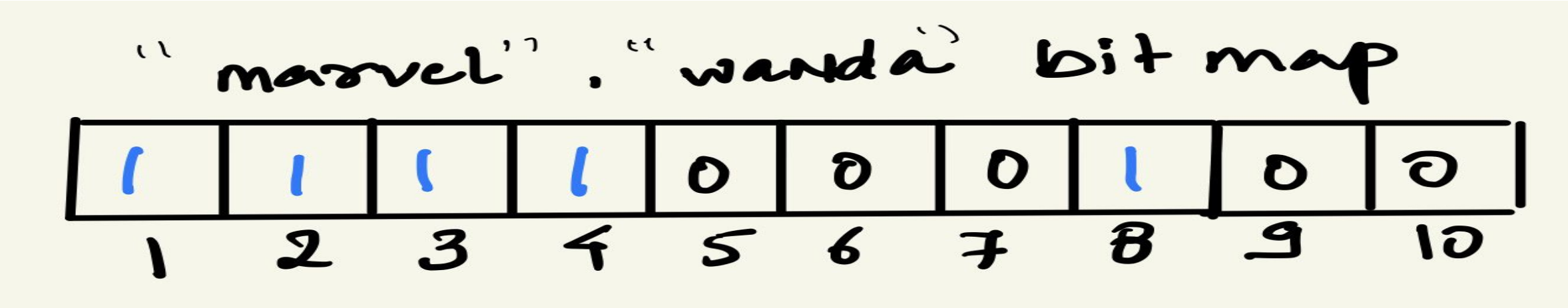
Bloom Filter: Special Properties (Contd.)

- The other special property is that it is probabilistic in nature.
- So let's consider this example, we have a bit array of 8 bits and it is storing two words right now, "Marvel" and "Wanda". This is what the representation of these two words looks like in a single-bit array:



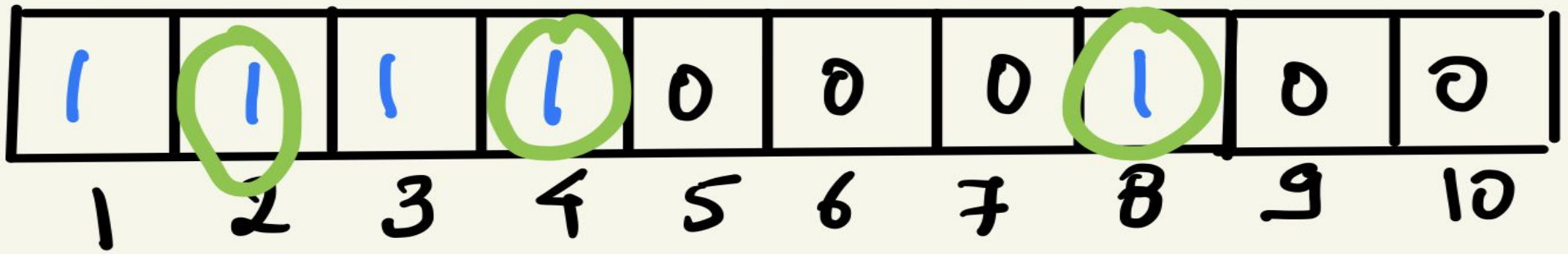
Bloom Filter: Special Properties (Contd.)

- Now, if we were to search for the word “Thanos”,
  - We would have to pass the word through the hash functions and obtain the bits.
  - We would then check if those bits have a value of 1 and, if all the bits have a value of 1, it would mean that the word exists.



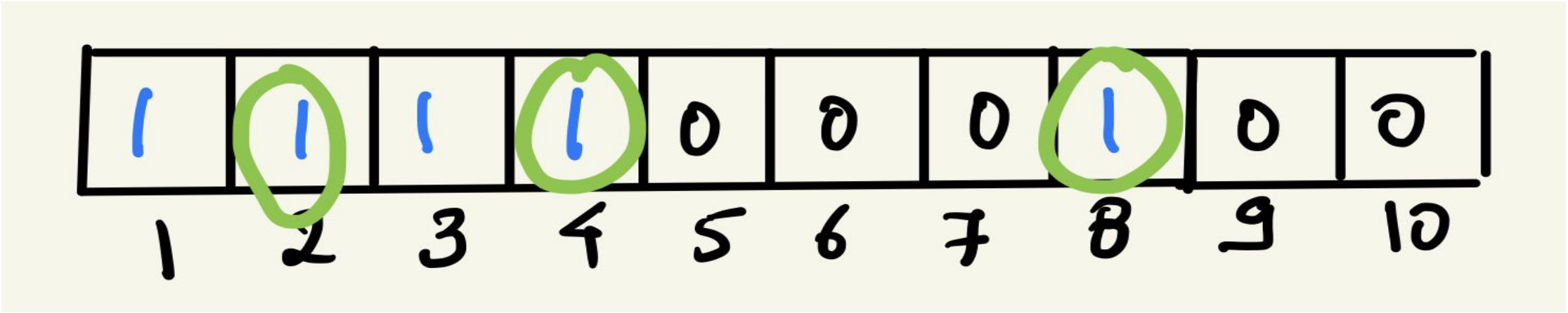
Bloom Filter: Special Properties (Contd.)

- So let's say "Thanos" when passed through hash functions gives bits 2, 4, 8.
    - If we look at the bit array, the positions at 2, 4, and 8 have a value of 1.
    - So this would mean that the word does exist in the filter but, in reality, it does not. These bits were set to 1 while storing the words "Marvel" and "Wanda".



Bloom Filter: Special Properties (Contd.)

This is what makes the data structure probabilistic in nature.



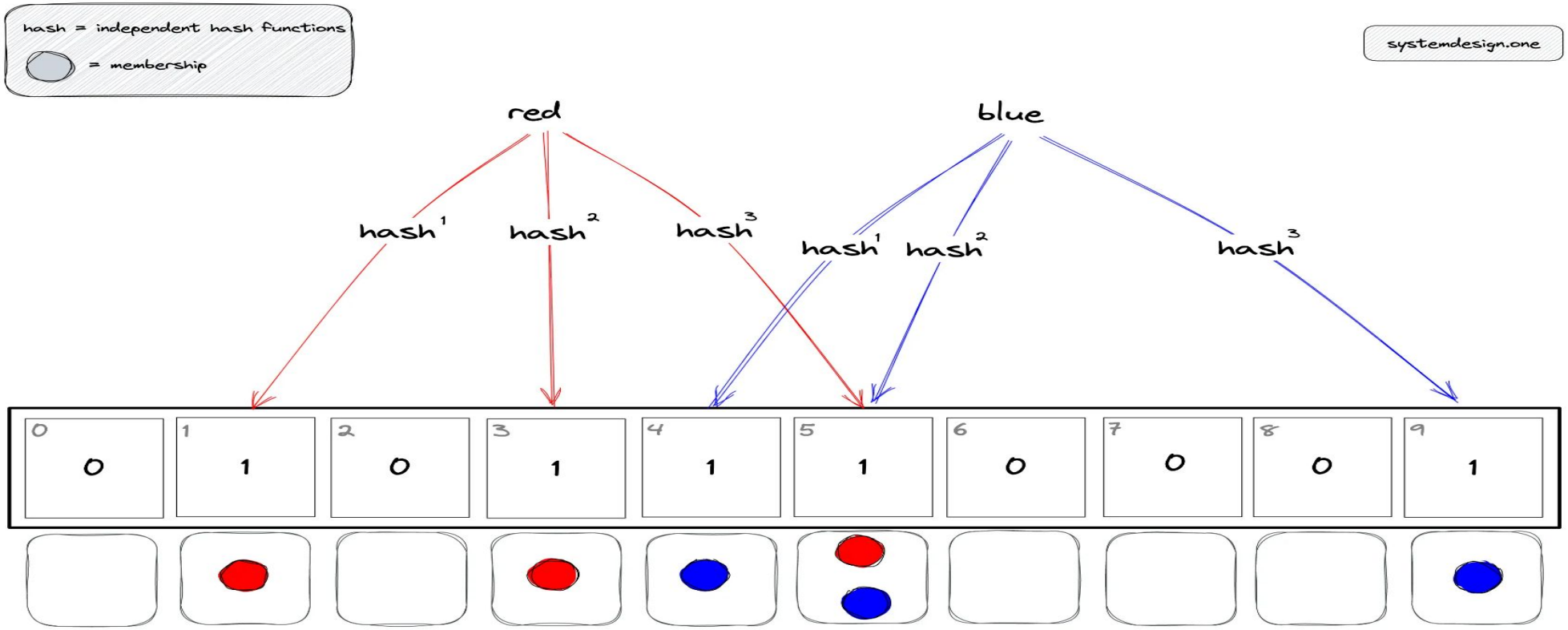
- **When the output of the bloom filter is positive, i.e the element is found in the set**, **it may or may not be true**. So it could also be a **false positive**.

But if the output of the bloom filter is negative, i.e **the element is not found in the set**, **it definitely does not exist in the set.**

How to add an item to the bloom filter?

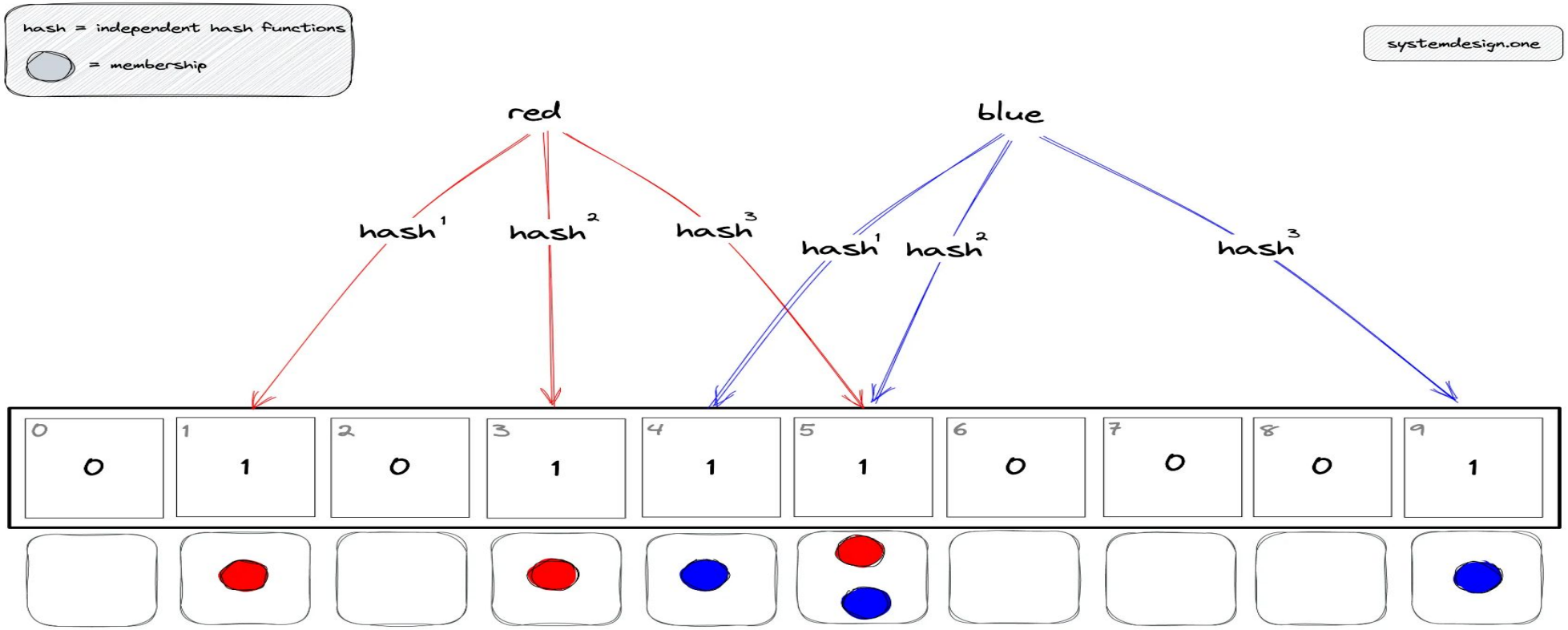
- The following operations are executed to add an item to the bloom filter:
  - the item is hashed through *k* hash functions
  - the modulo n (length of bit array) operation is executed on the output of the hash functions to identify the k array positions (indexes)
  - the bits at all identified indexes are set to one
- There is a probability that some bits on the array are set to one multiple times due to hash collisions.

How to add an item to the bloom filter? (Contd.)



The items red and blue are added to the bloom filter. The indexes that should be set to one for the item red are identified by the execution of the modulo operator on the computed hash value.

How to add an item to the bloom filter? (Contd.)



• h1(red) mod 10 = 1 • h2(red) mod 10 = 3 • h3(red) mod 10 = 5	The indexes that should be set to one for the item *blue* are identified by the execution of the modulo operator on the computed hash value. The index at position five is set to one by distinct items *red* and *blue*.	• h1(blue) mod 10 = 4 • h2(blue) mod 10 = 5 • h3(blue) mod 10 = 9

How to check the membership of an item in the bloom filter?

- The following operations are executed to check if an item is a member of the bloom filter:
  - the item is hashed through *k* hash functions
  - the modulo n (length of bit array) operation is executed on the output of the hash functions to identify the k array positions (indexes)
  - verify if all the bits at identified indexes are set to one

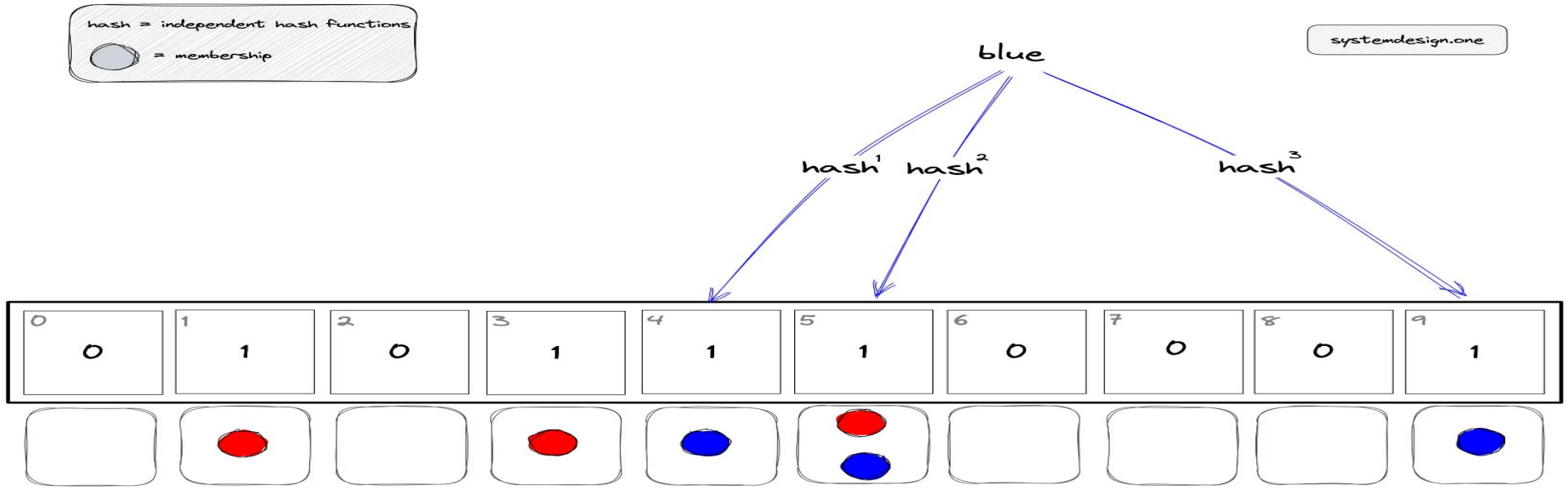
How to check the membership of an item in the bloom filter? (Contd.)

- **If any of the identified bits are set to zero, the item is not a member of the bloom filter.**
- *If all the bits are set to one, the item might be a member of the bloom filter.*
- The uncertainty about the membership of an item is due to the possibility of some bits being set to one by different items or due to hash function collisions.

How to update an item to the bloom filter?

- The following operations are executed to update an item to the bloom filter:
  - the new item is hashed through *k* hash functions
  - the modulo n (length of bit array) operation is executed on the output of the hash functions to identify the k array positions (indexes)
  - the bits at all identified indexes are set to one
- There is a probability that some bits on the array are set to one multiple times due to hash collisions.

How to check the membership of an item in the bloom filter? (Contd.)



The bloom filter is queried to check the membership of item blue. The buckets that should be checked are identified by the execution of the modulo operator on the computed hash value..

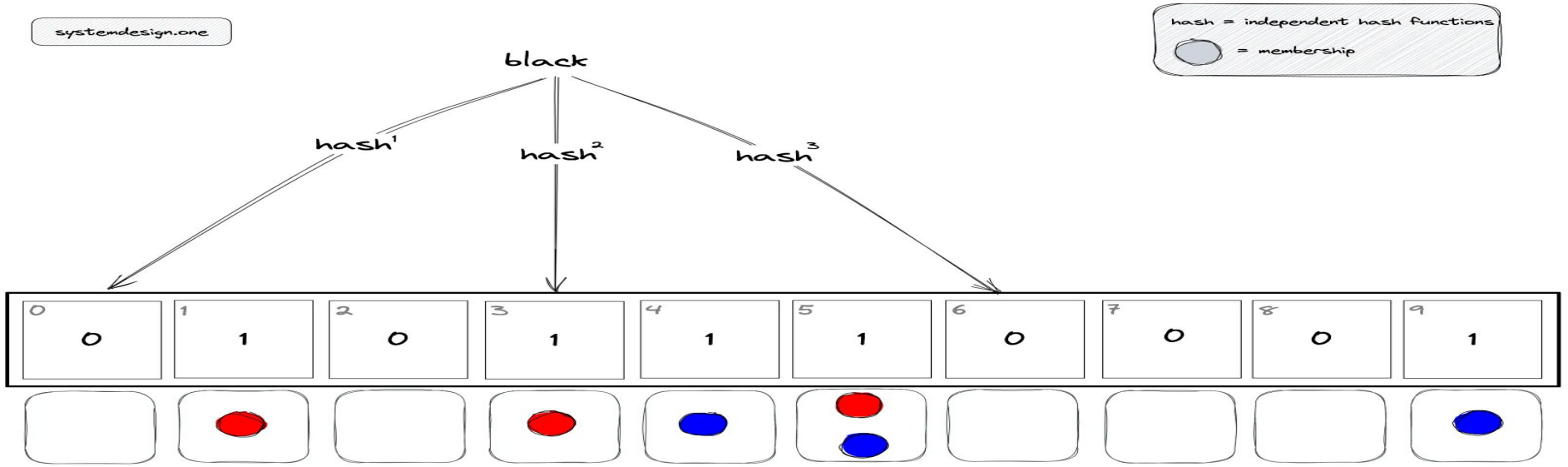
h1(blue) mod 10 = 4

h2(blue) mod 10 = 5

h3(blue) mod 10 = 9

The item blue might be a member of the bloom filter as all the bits are set to one.

How to check the membership of an item in the bloom filter? (Contd.)



The bloom filter is queried to check the membership of item black, which is not a member of the bloom filter.

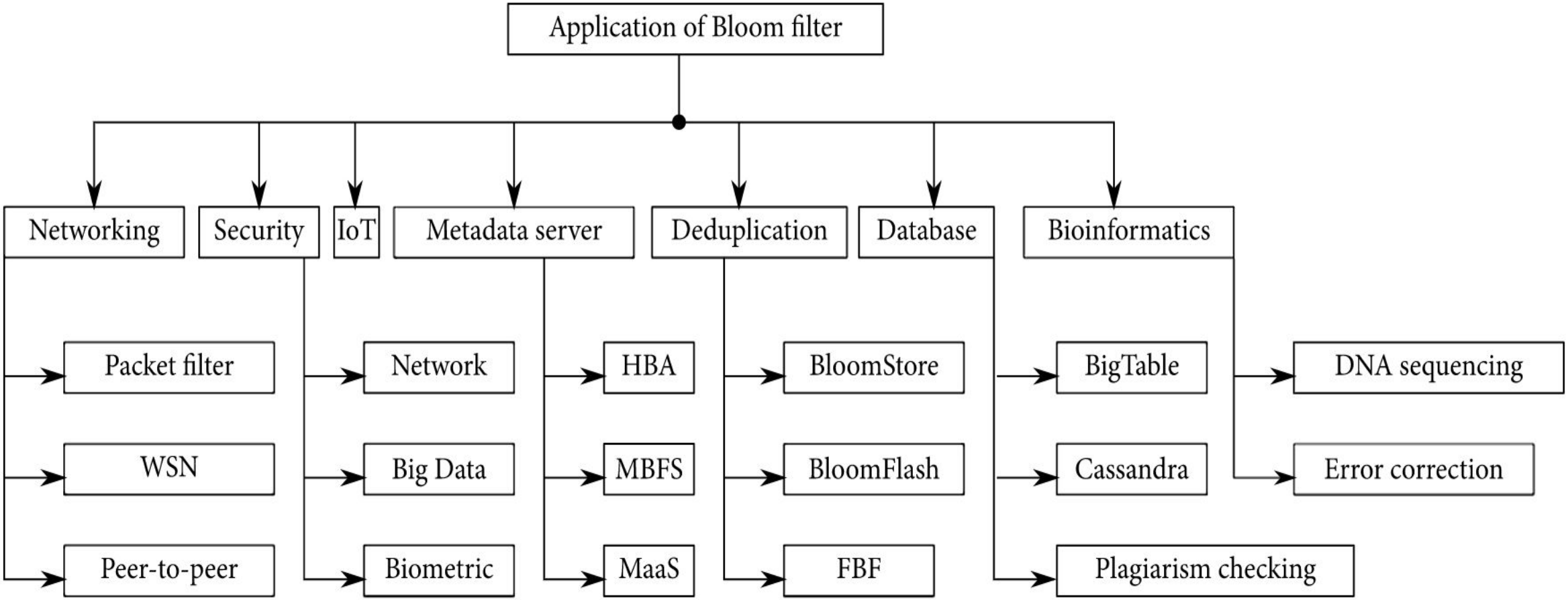
h1(black) mod 10 = 0

h2(black) mod 10 = 3

h3(black) mod 10 = 6

The item black is not a member of the bloom filter as the bit at position zero is set to zero. The verification of the remaining bits can be skipped.

Applications of Bloom Filter



References

1. https://omereingold.files.wordpress.com/2017/06/p97-cormode.pdf
2. https://devopedia.org/bloom-filter
3. https://levelup.gitconnected.com/what-is-bloom-filter-413c1485104f
4. https://techeffigytutorials.blogspot.com/2015/01/bloom-filters-explained.html
5. https://www.ombulabs.com/blog/systemdesign/ruby/bloom-filter-and-what-makes-them-special.html
6. https://systemdesign.one/bloom-filters-explained/#introduction
7. https://www.math.umd.edu/~immortal/CMSC420/notes/bloomfilters.pdf
8.