Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

Lecture 28
Collision Resolution

FIT 1008&2085
Introduction to Computer Science

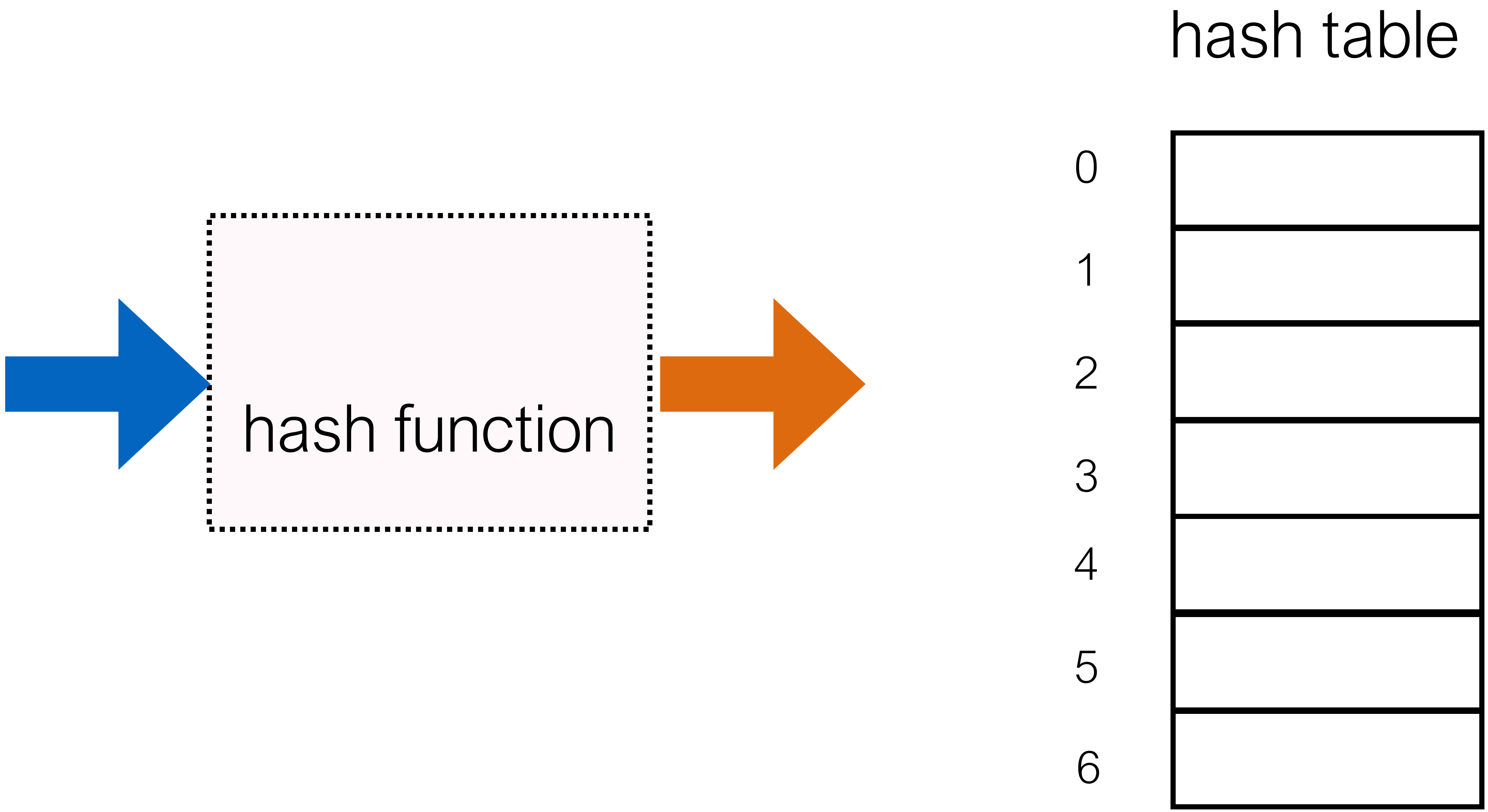
MONASH University
Information Technology

Hash Table operations: Insert

- Apply the hash function to get a position N
- Try to insert key at position N
- Deal with collision if any

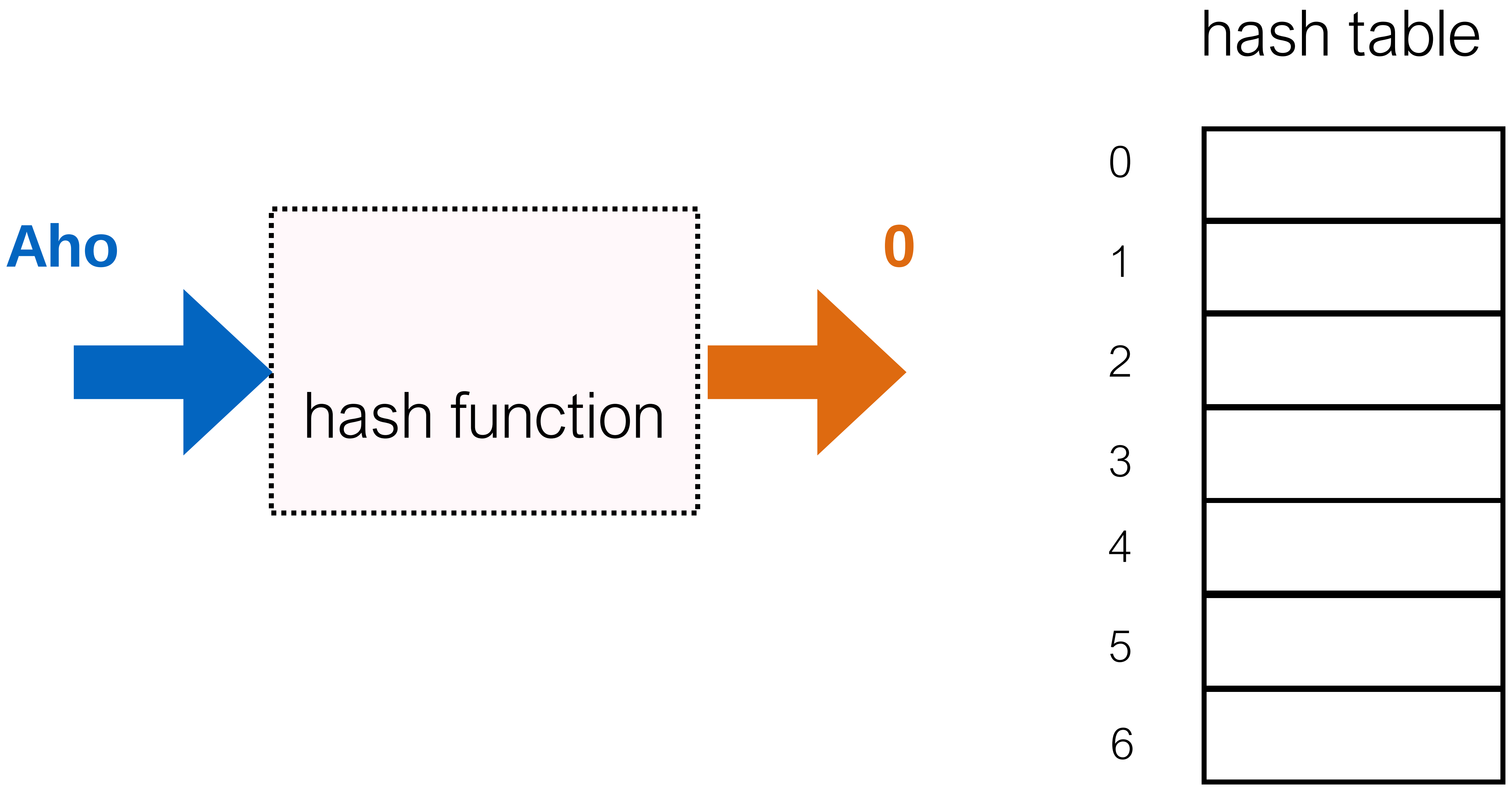
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



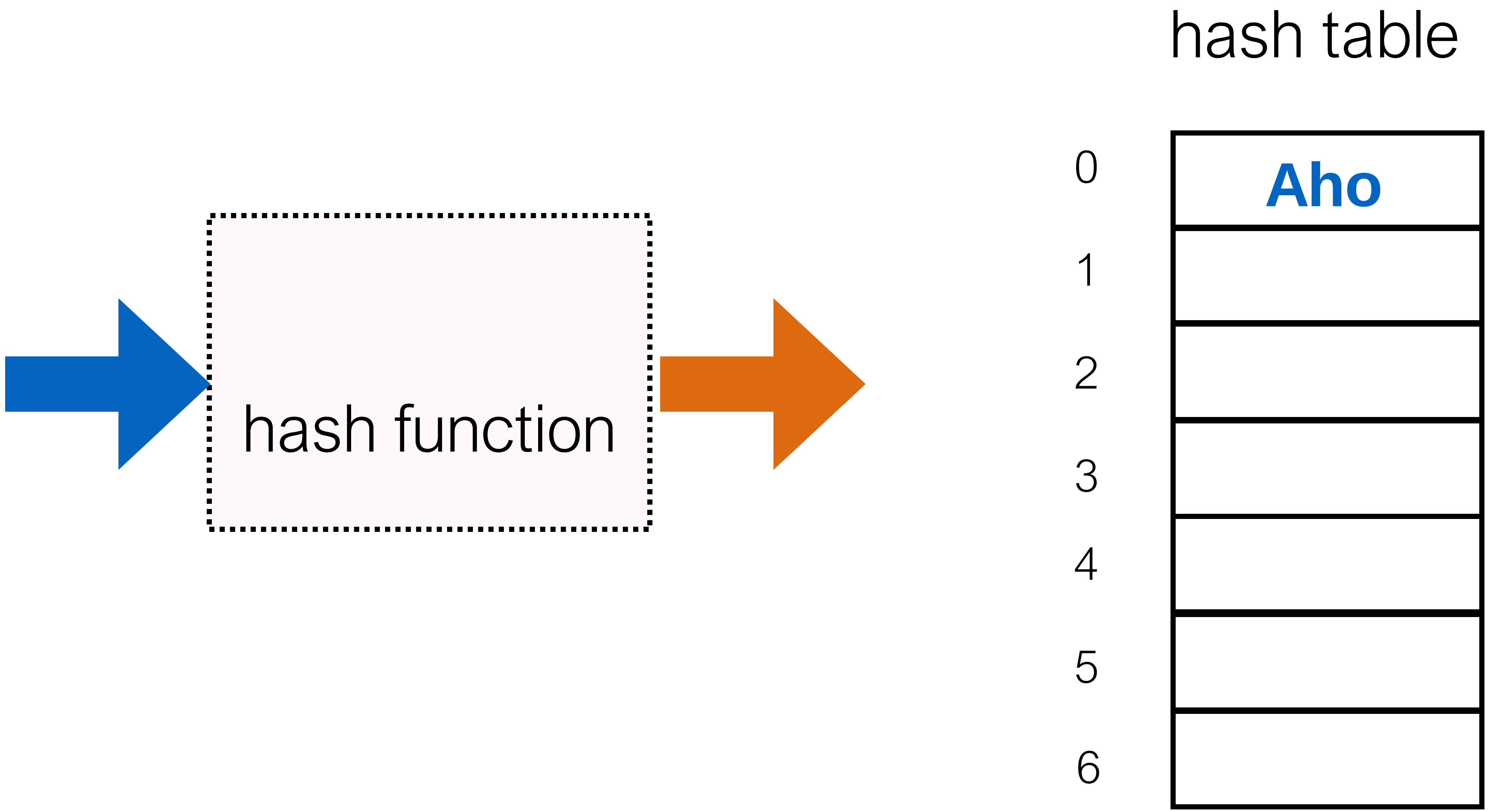
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



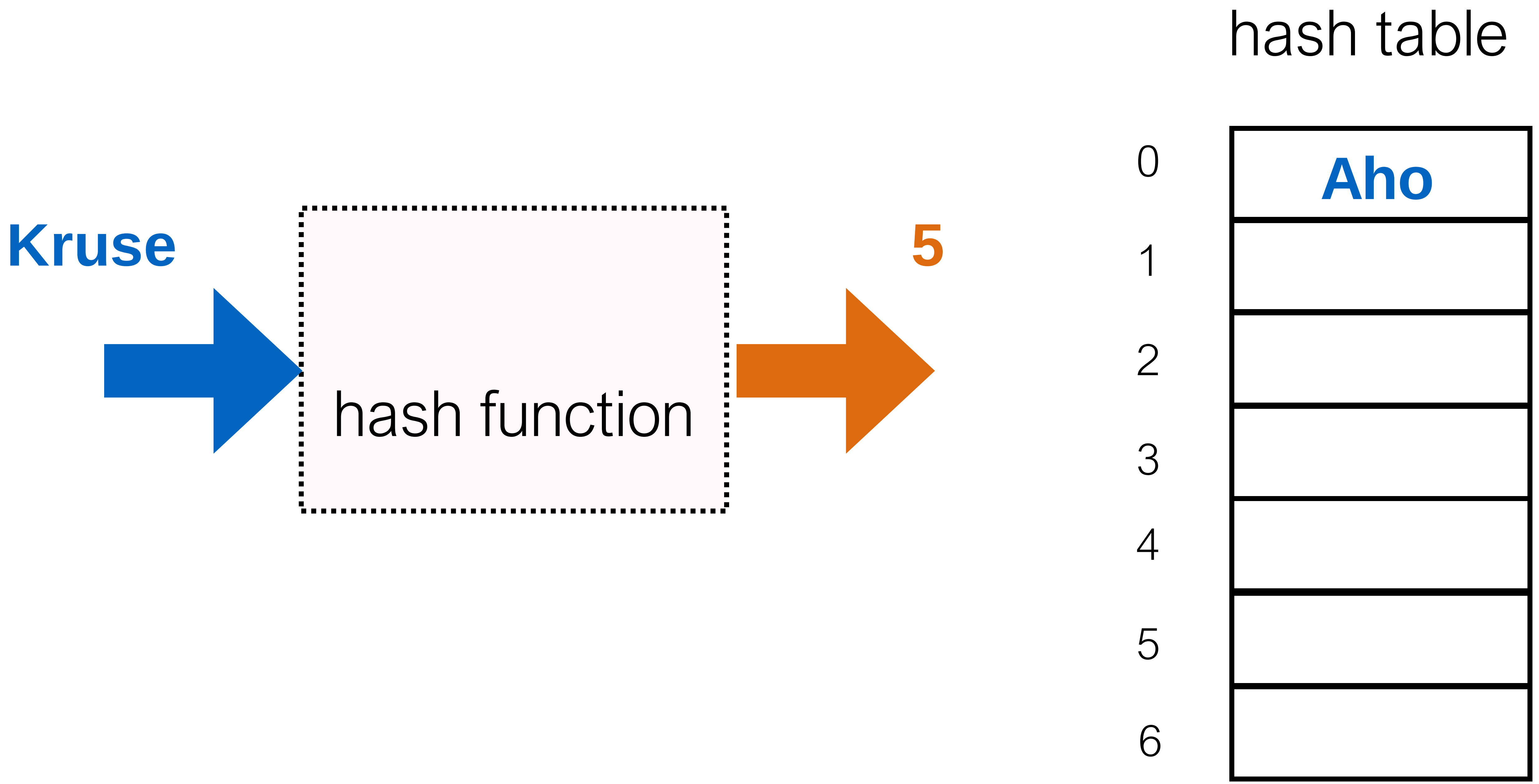
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



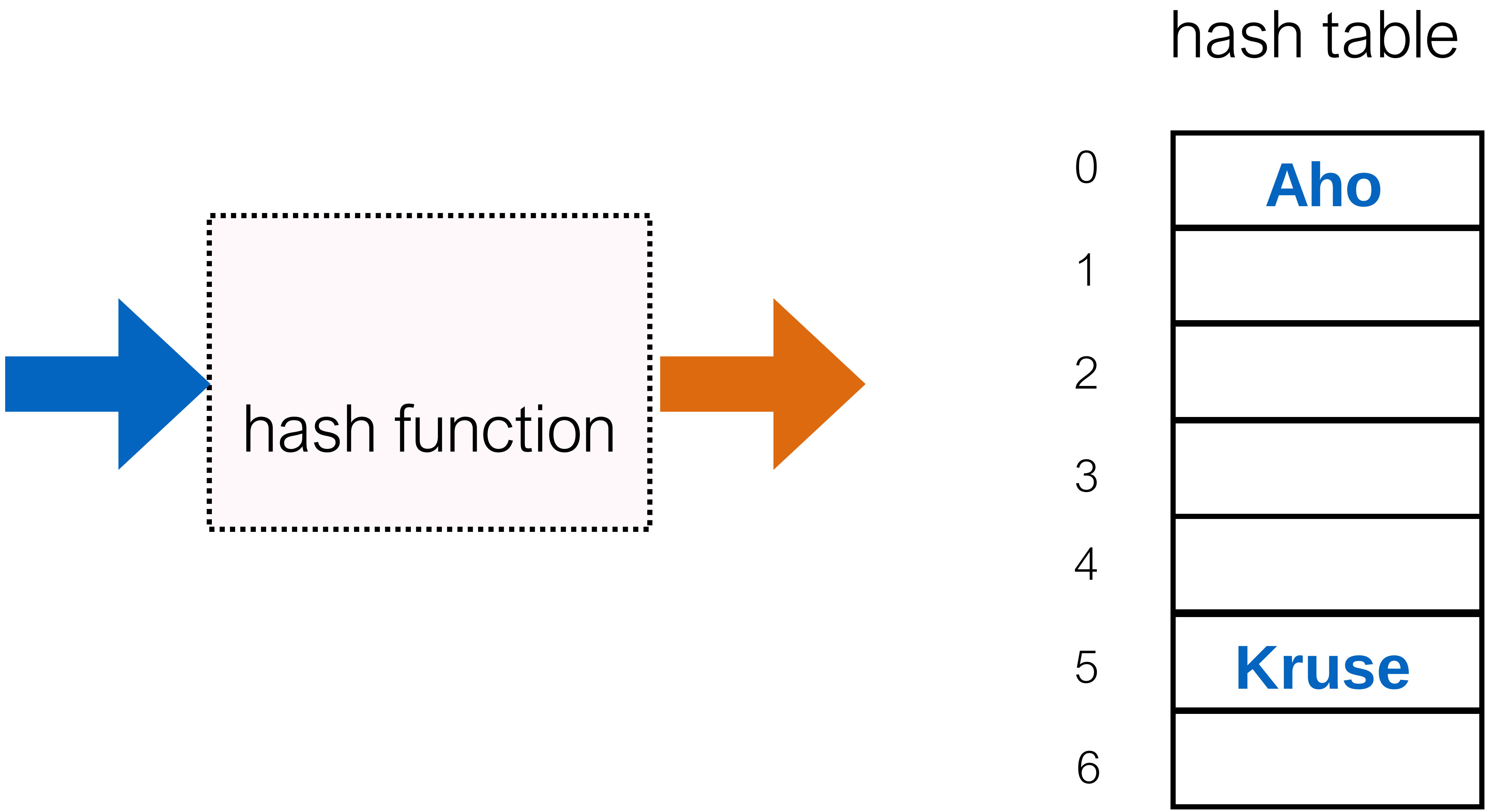
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



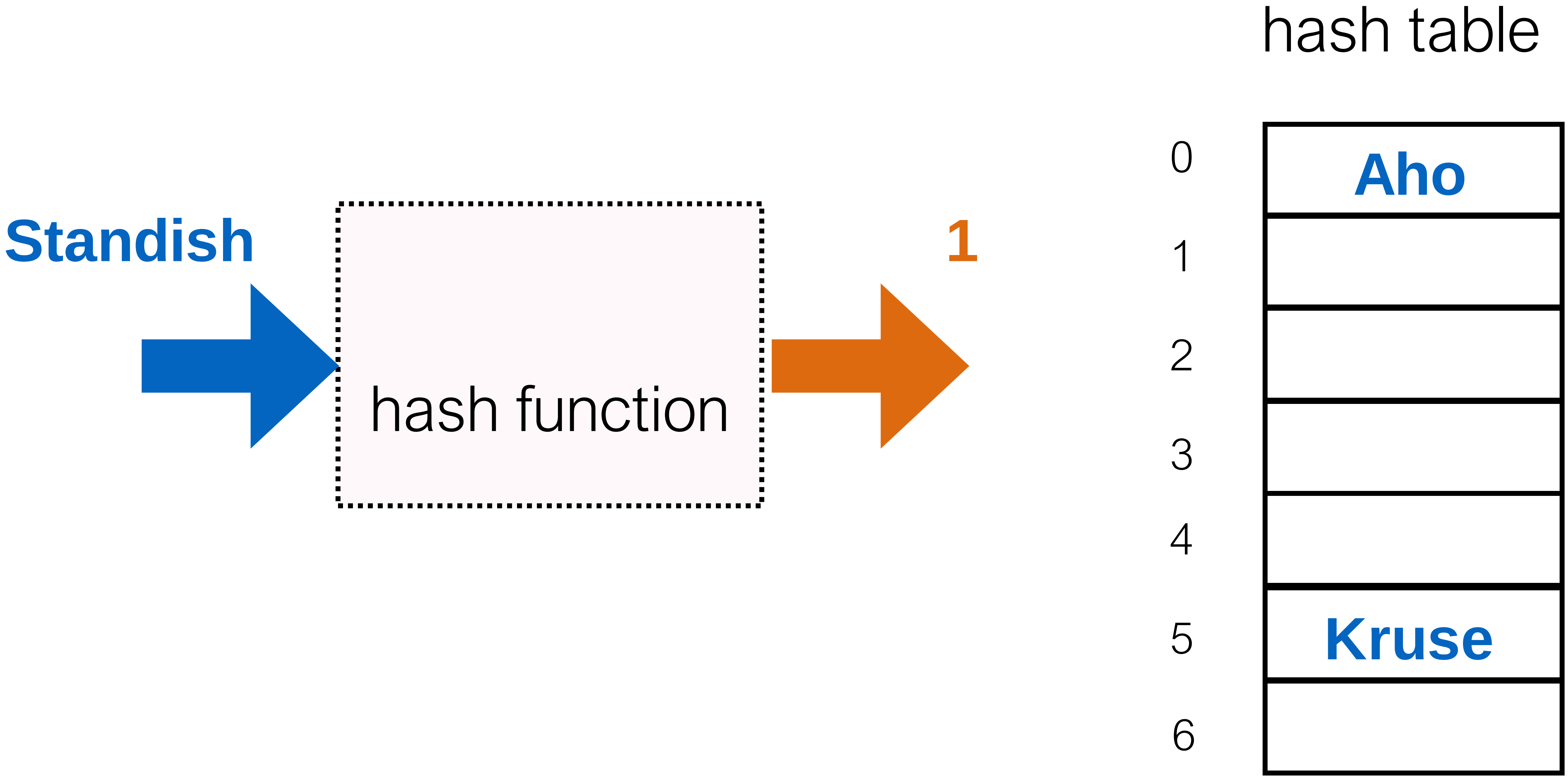
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



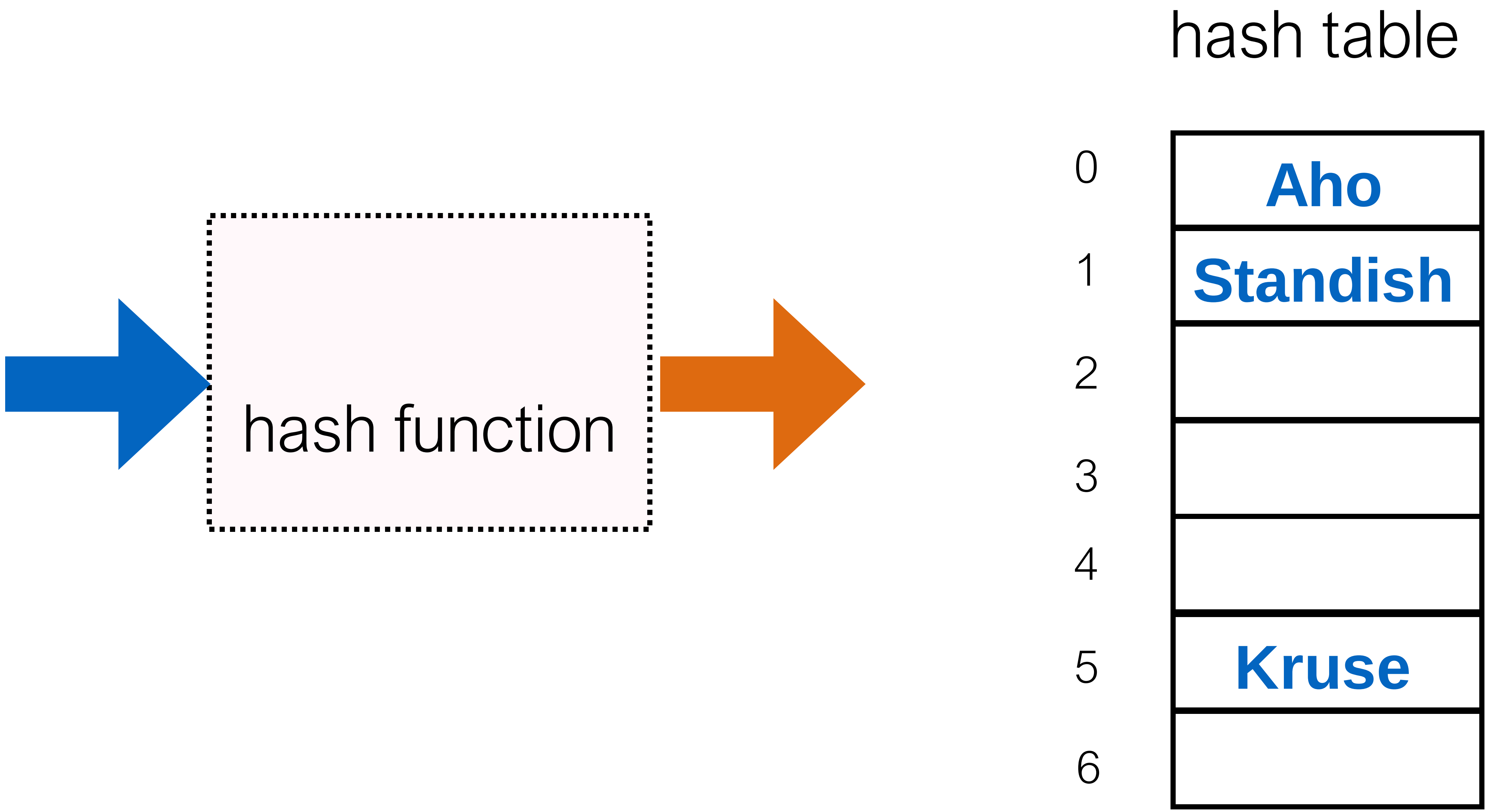
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



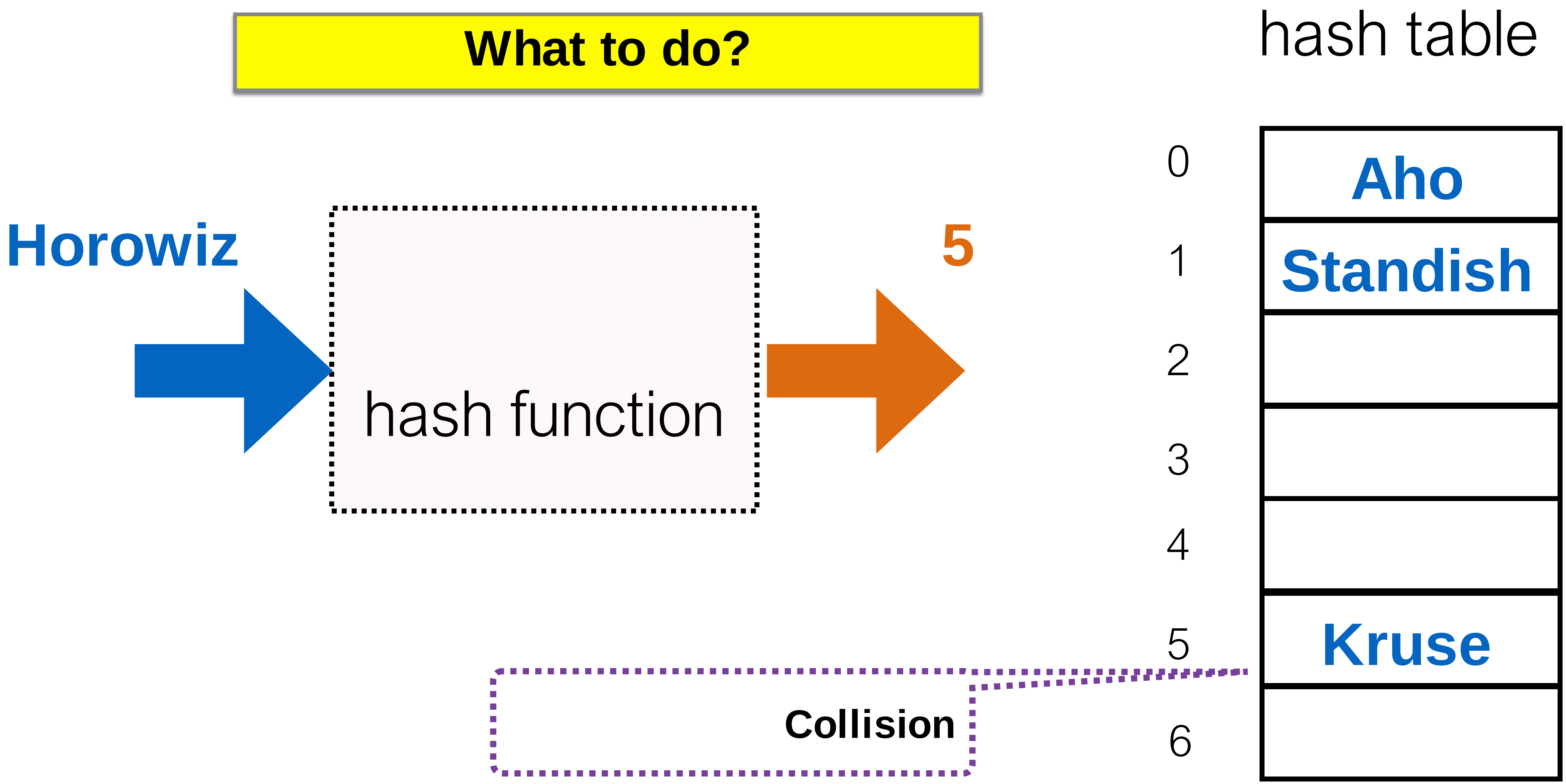
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



Objectives for this lecture

- To understand two of the main methods of conflict resolution:
  - Open addressing:
    - Linear Probing
    - Quadratic probing
    - Double Hashing
  - Separate Chaining

- To understand their advantages and disadvantages

- To be able to implement them

Collisions: two main approaches

- **Open addressing**:
  - Each array position contains a single item
  - Upon collision, use an empty space to store the item (which empty space depends on which technique)

- **Separate chaining**:
  - Each array position contains a linked list of items
  - Upon collision, the element is added to the linked list

Open Addressing

Open Addressing: Linear Probing

- **Insert item with hash value N**:
  - If array[N] is empty just put item there.
  - If there is already an item there:
    look for the **first empty space in the array** from **N+k** (if any) and add it there

- Linear search from N until an empty slot is found *(moving along k at a time)*

- **Things to think about**:
  - Full table (to avoid going into an infinite loop)
  - Restarting from position 0 if the end of table is reached
  - Finding an item with the same key.

Insert the following keys into the Hash Table, in the order they appear, using linear probing. Is the following table correct? *(assume k=1)*

Key	Hash value
Aho	0
Kruse	5
Standish	1
Horowitz	5
Langsam	5
Sedgewick	2
Knuth	1

hash table

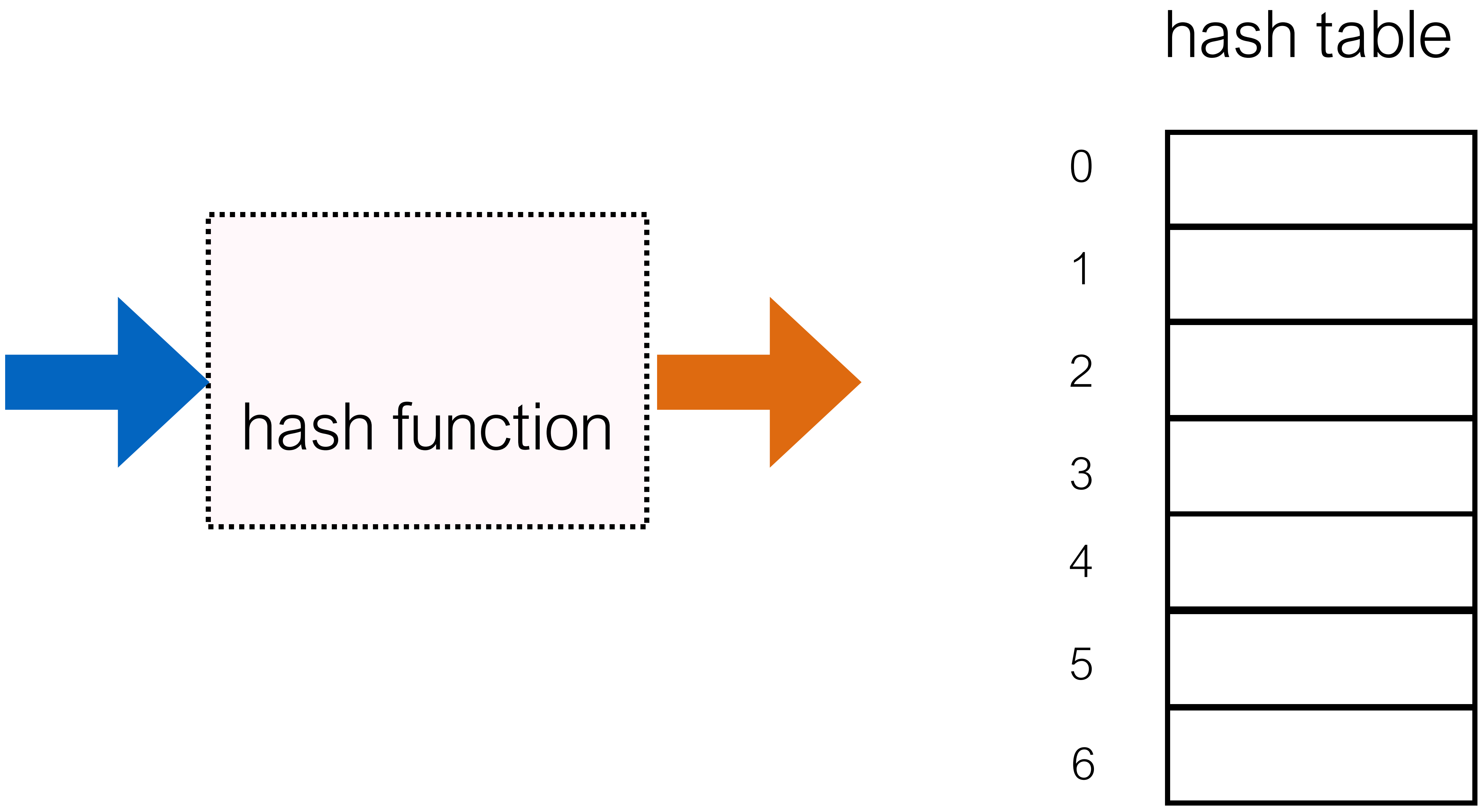
0	**Aho**
1	**Standish**
2	**Langsam**
3	**Knuth**
4	**Sedgewick**
5	**Kruse**
6	**Horowitz**

A) True

B) False

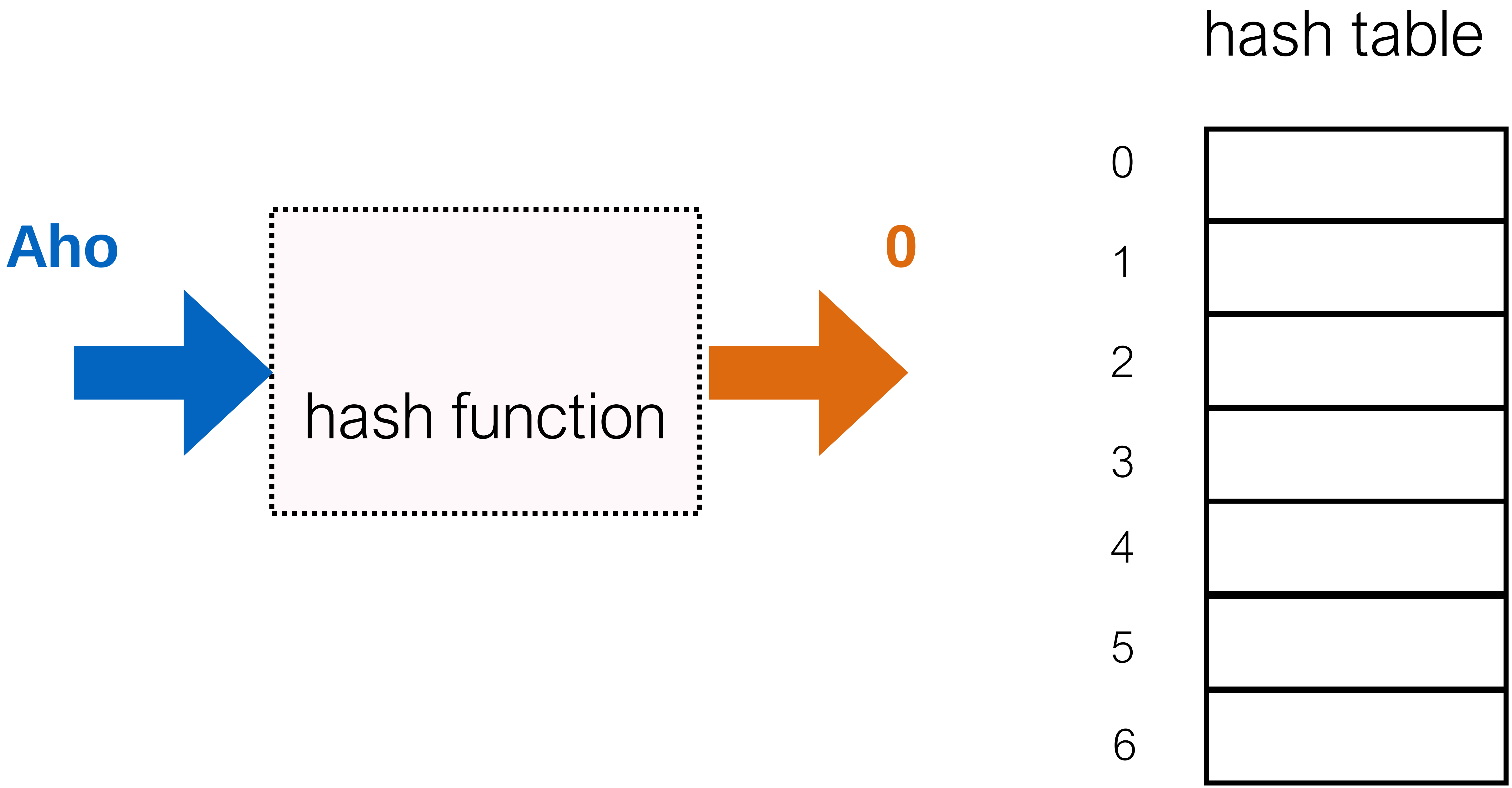
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



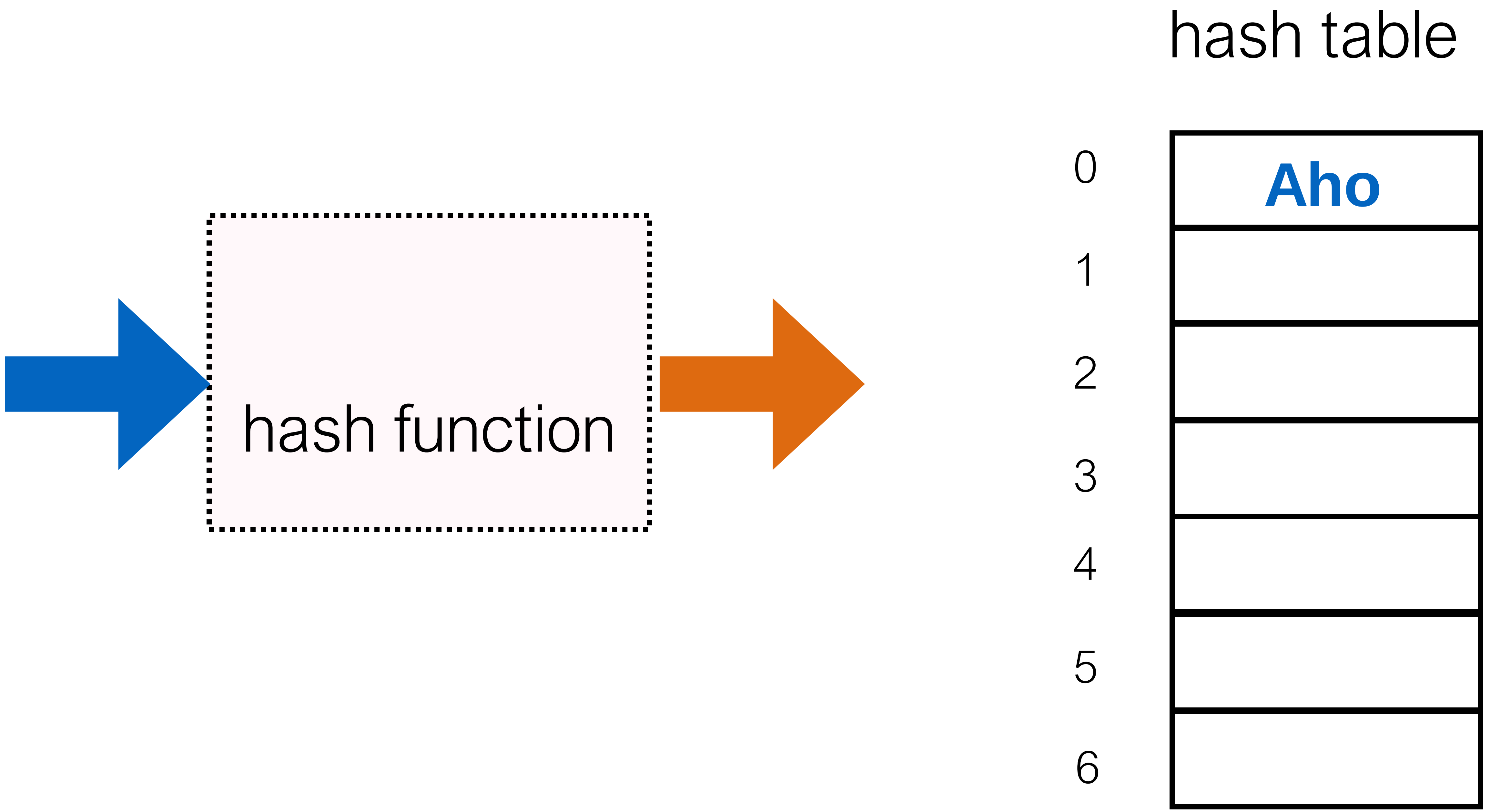
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



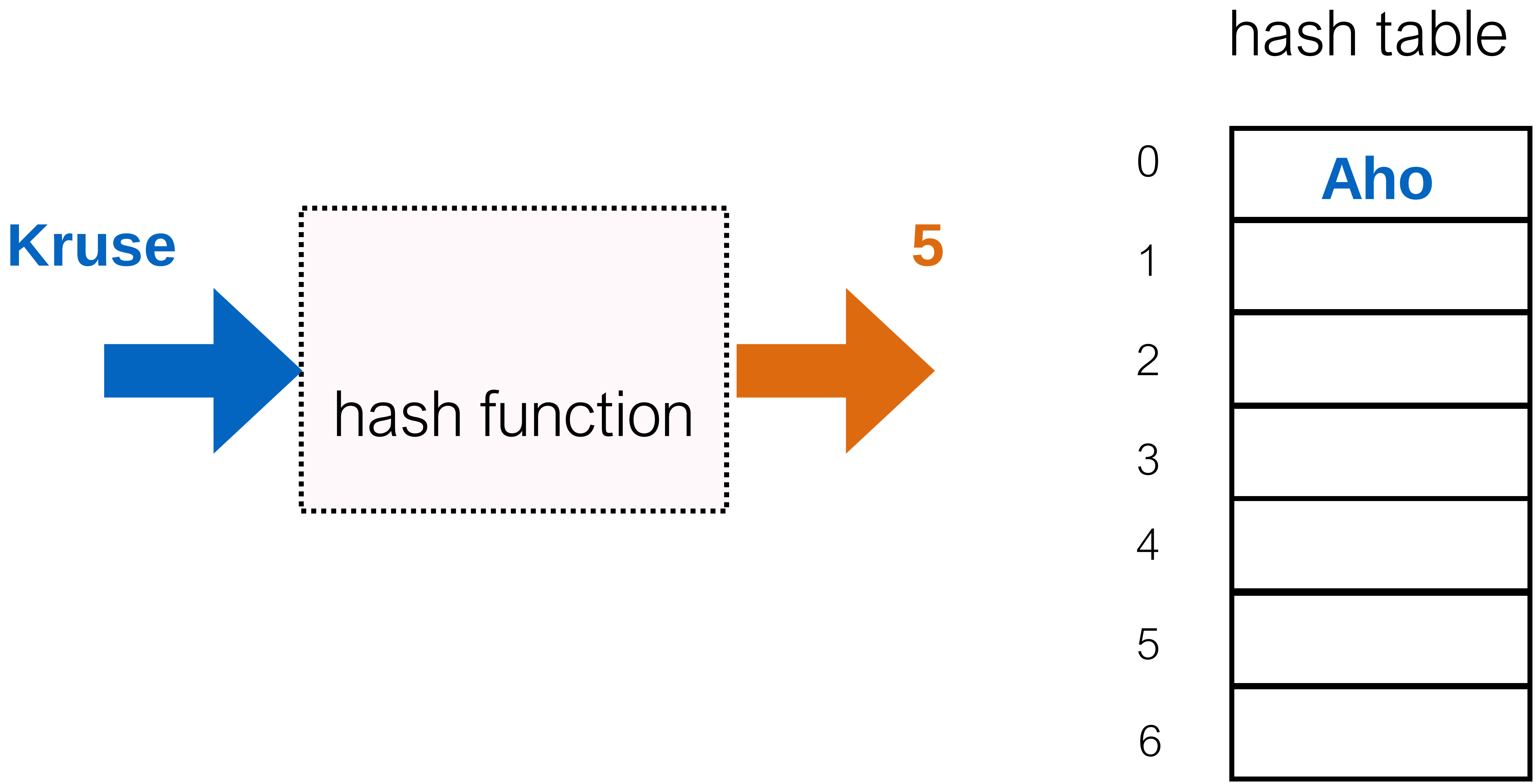
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



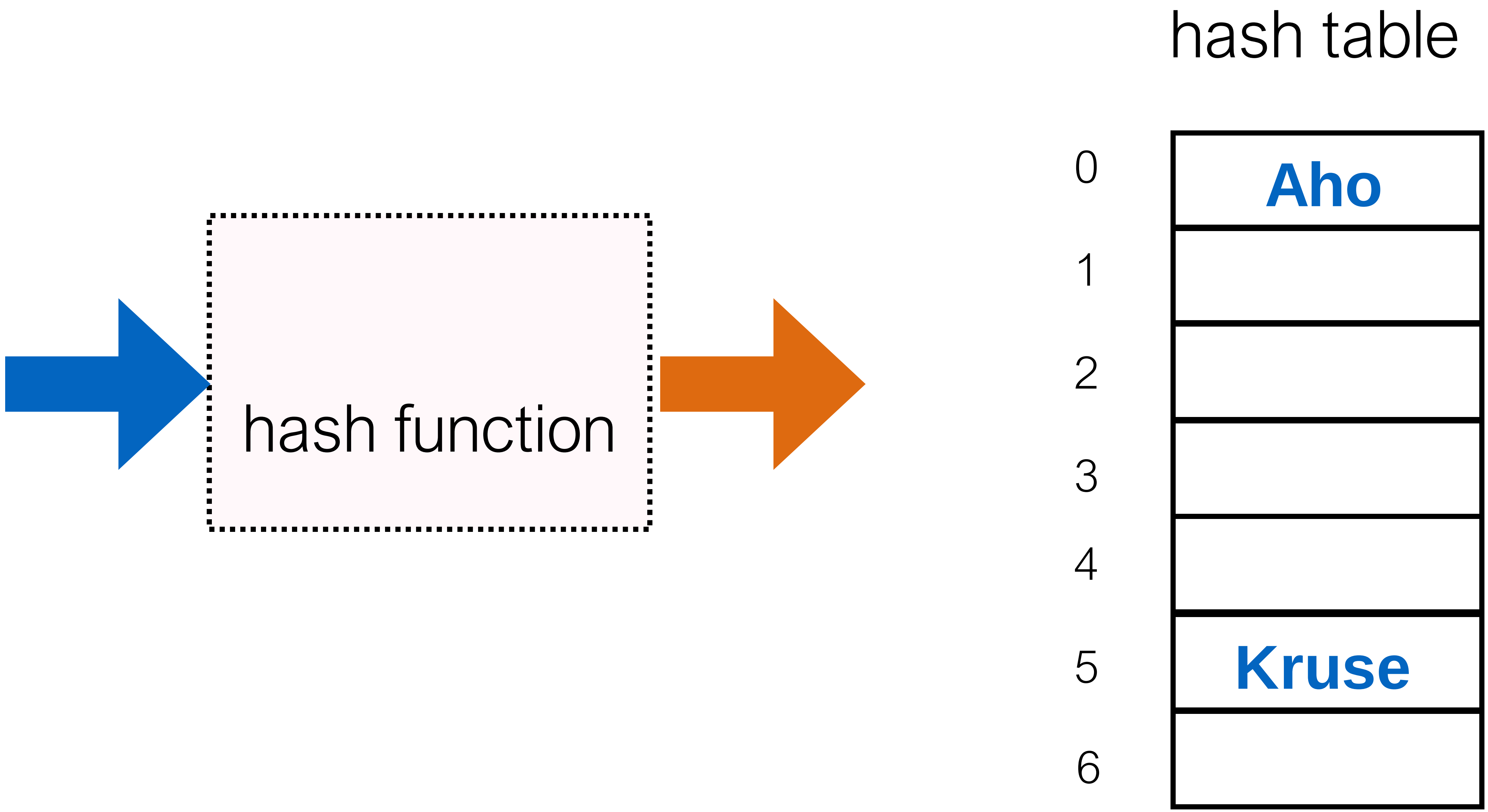
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



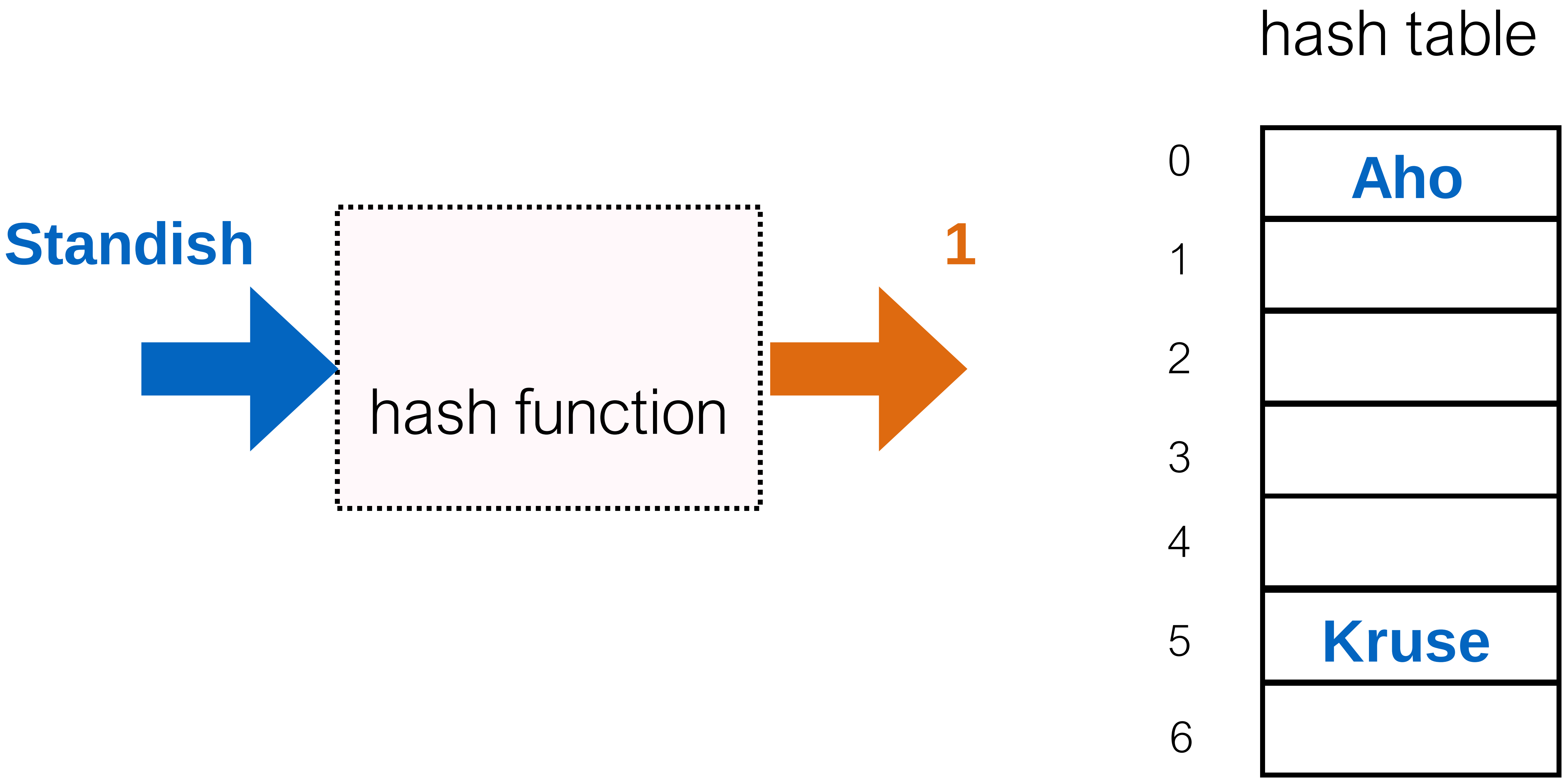
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



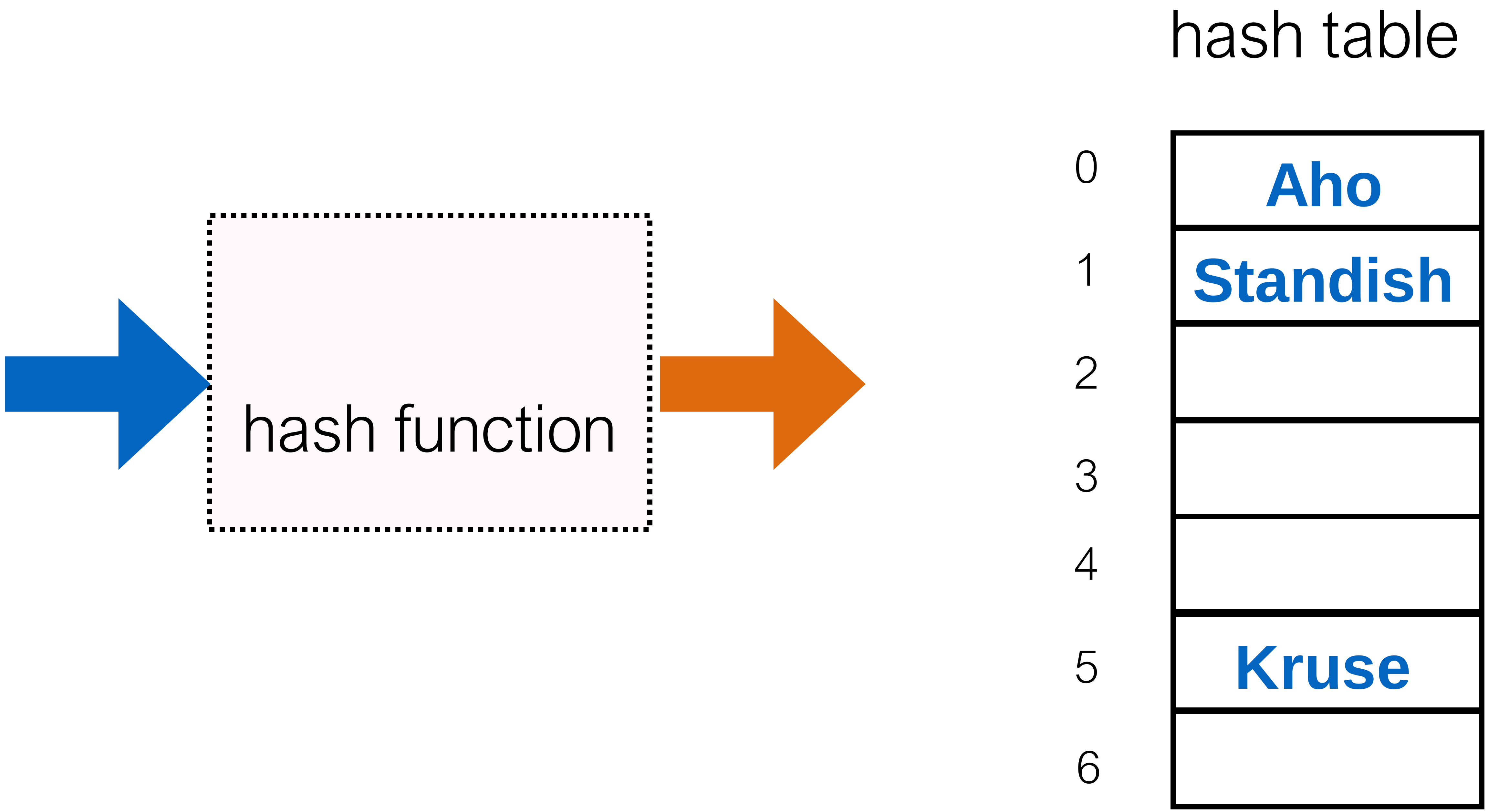
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



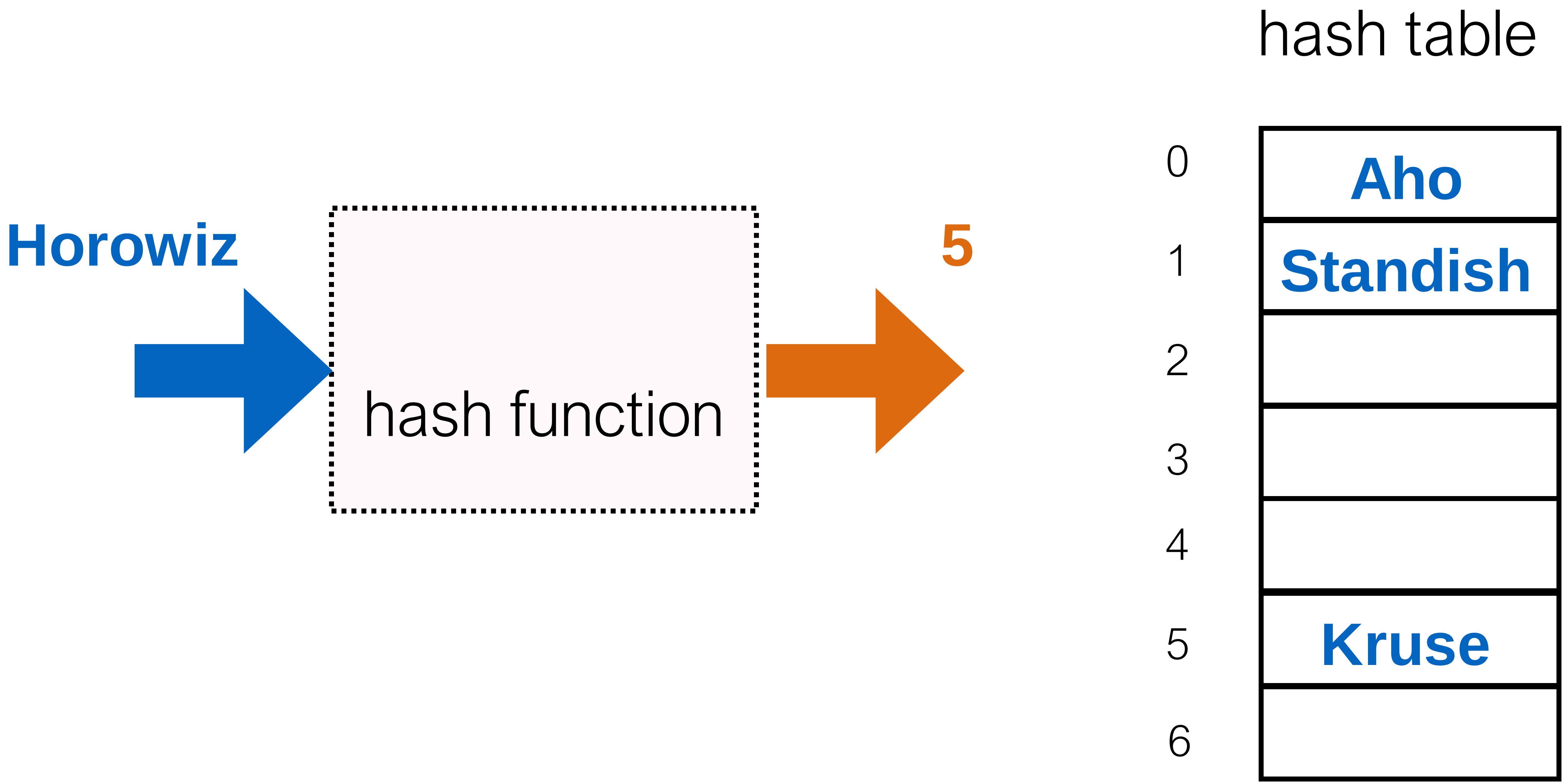
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



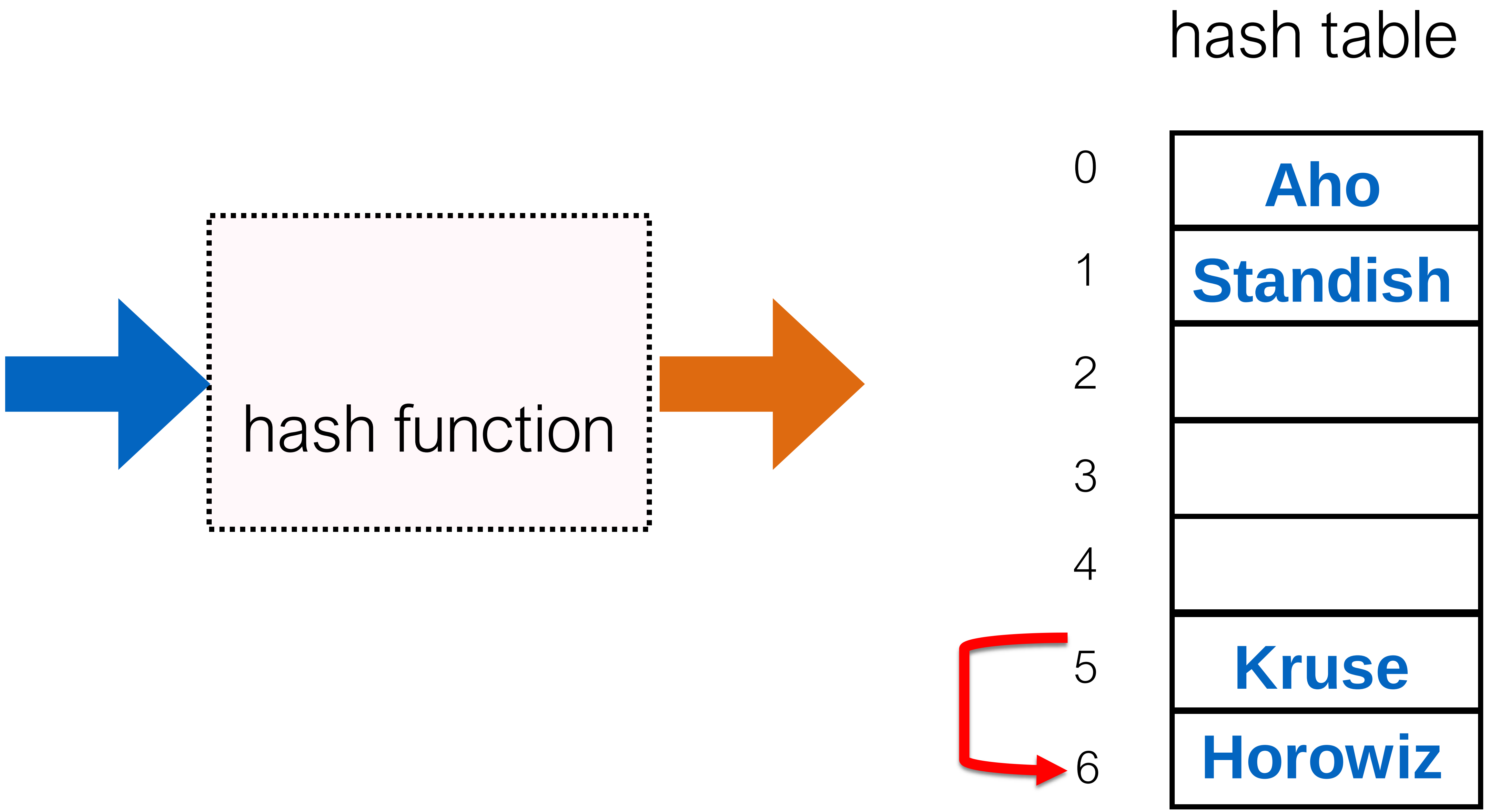
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



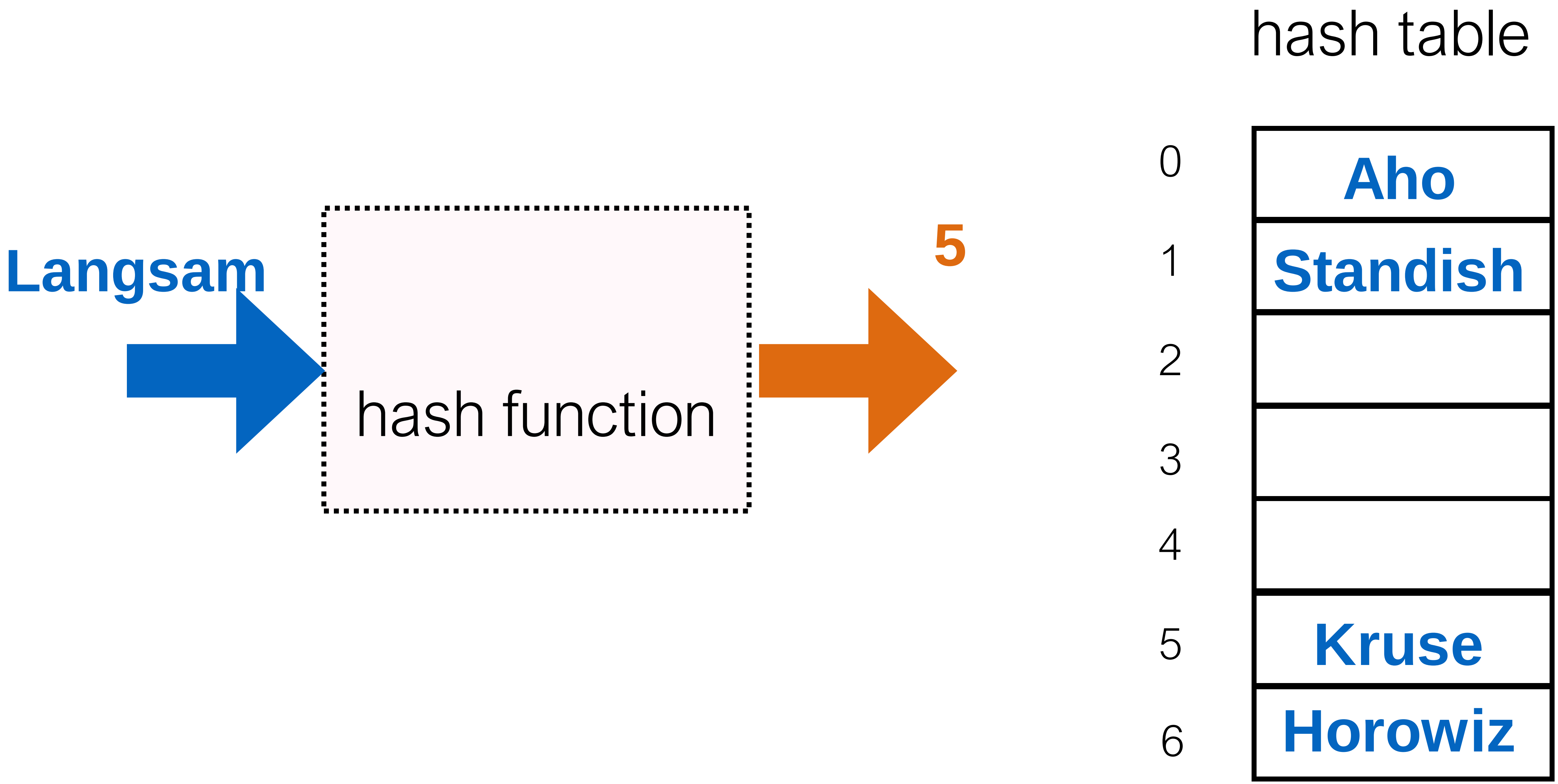
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



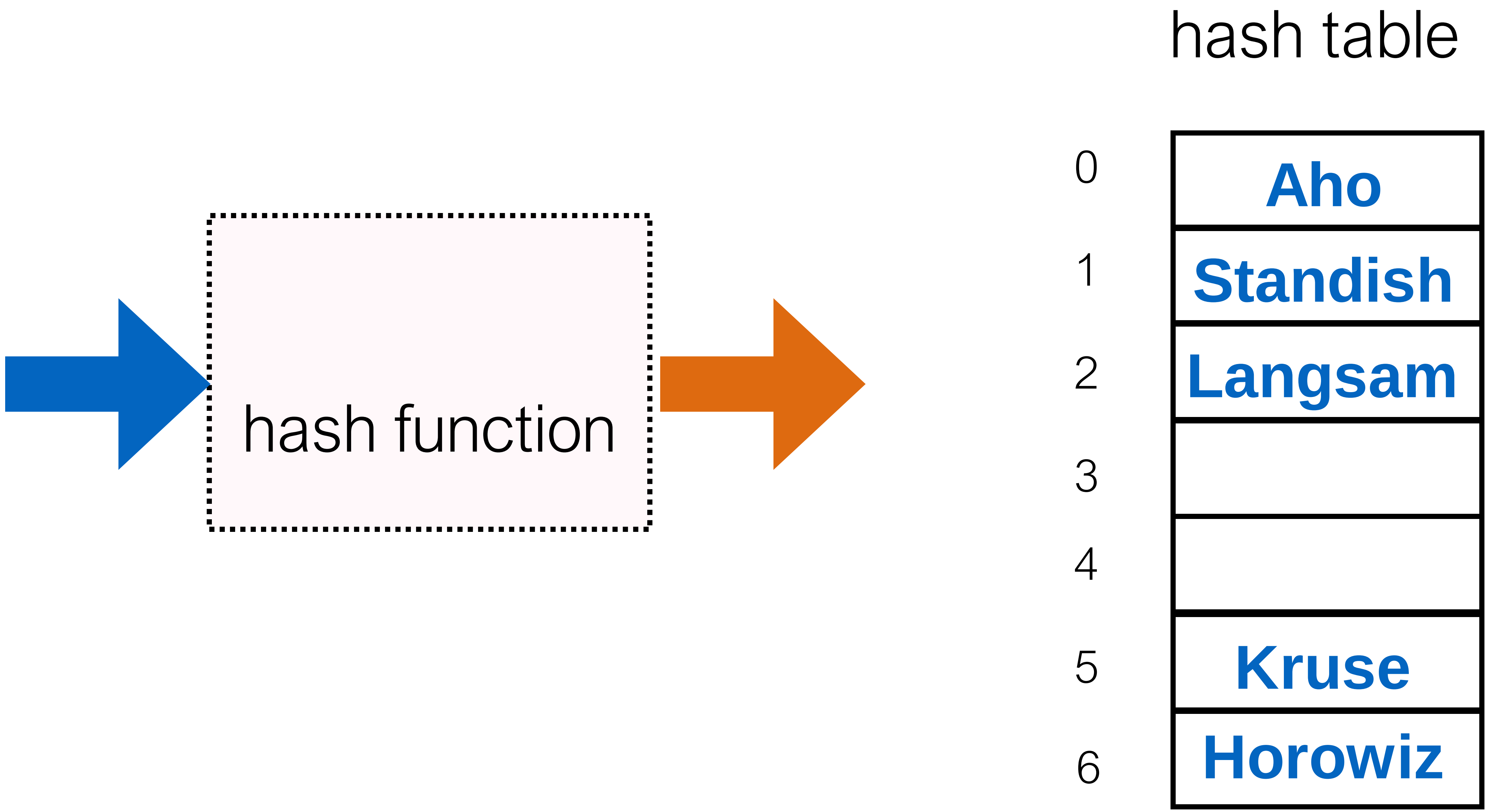
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



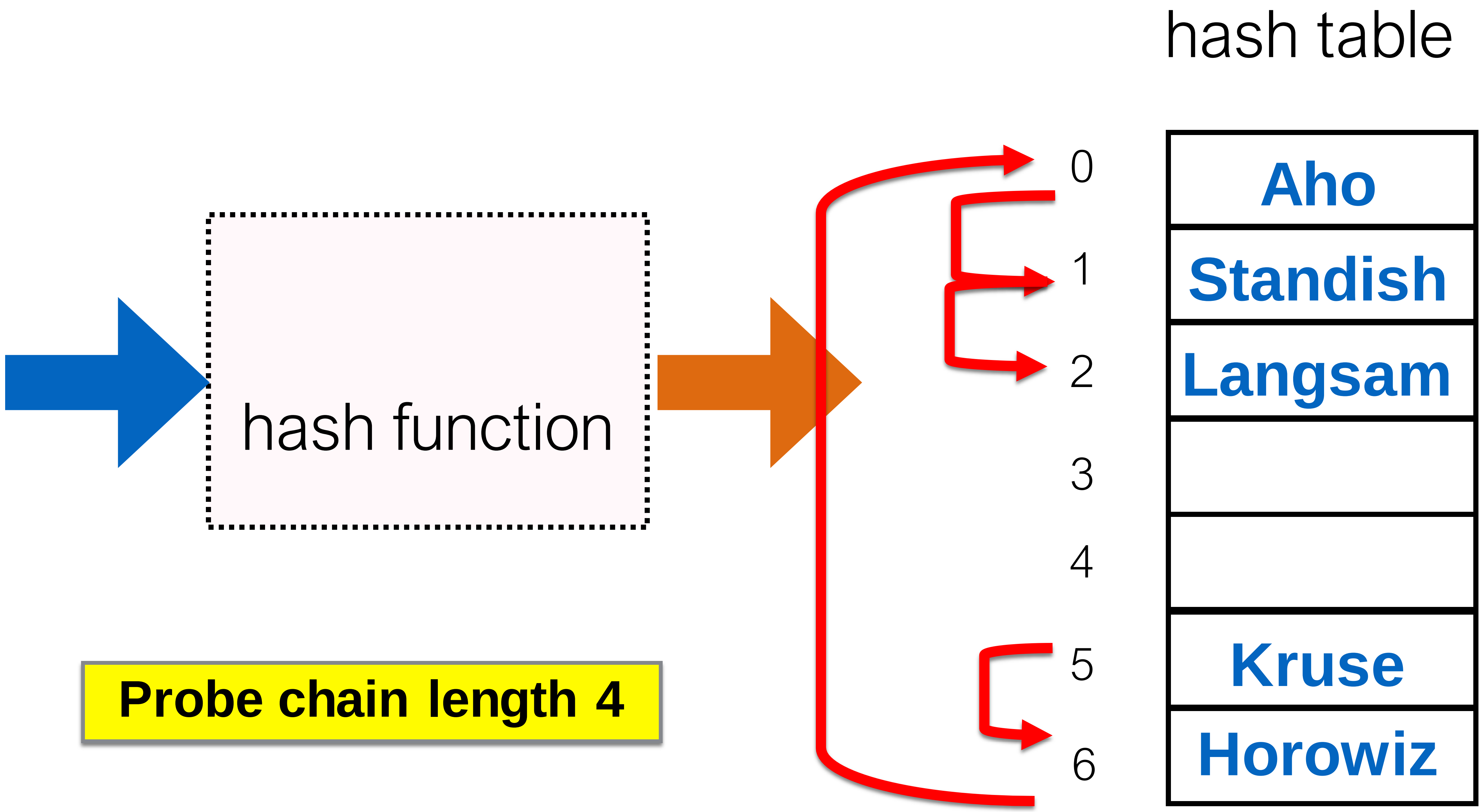
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



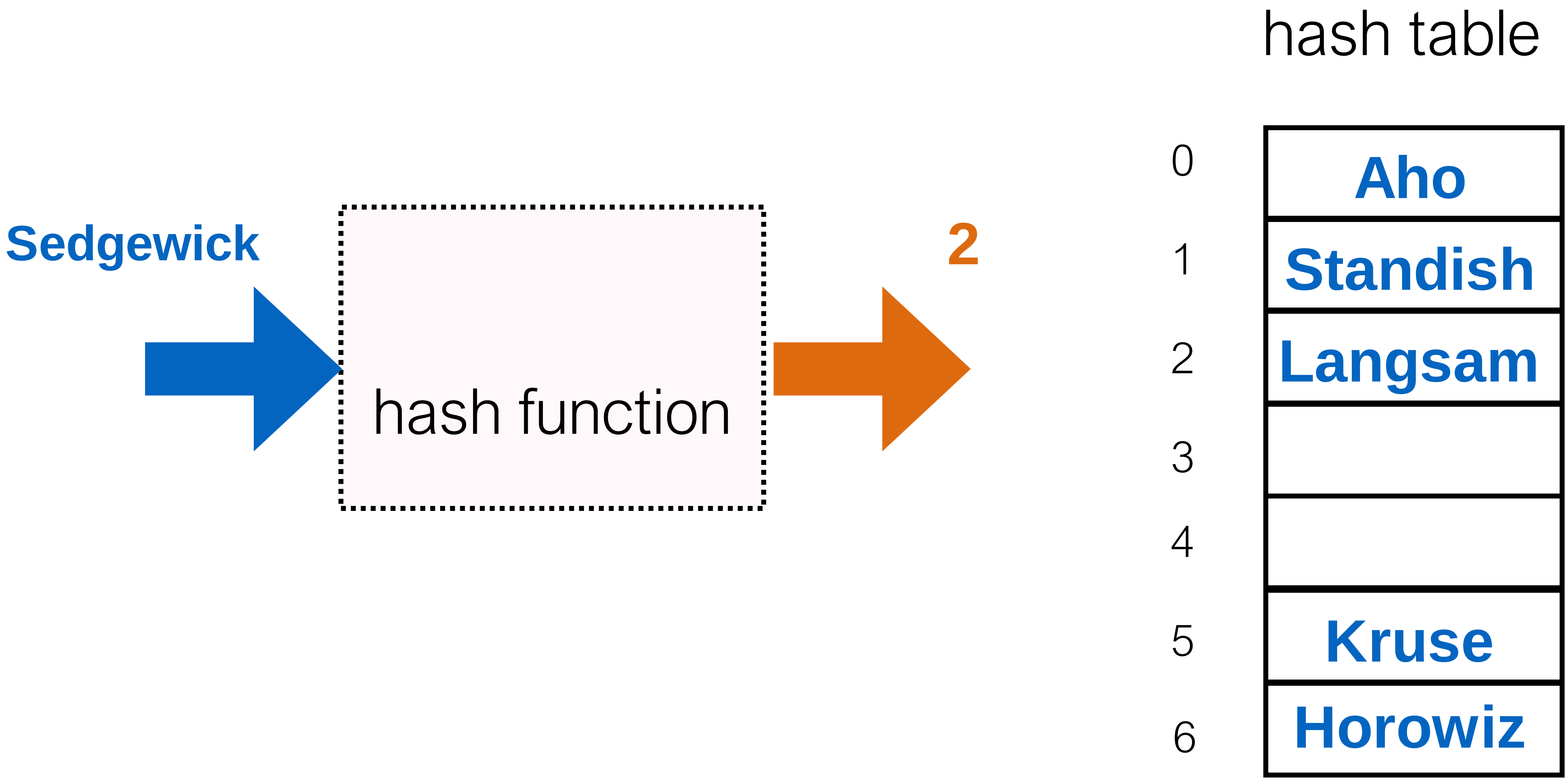
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



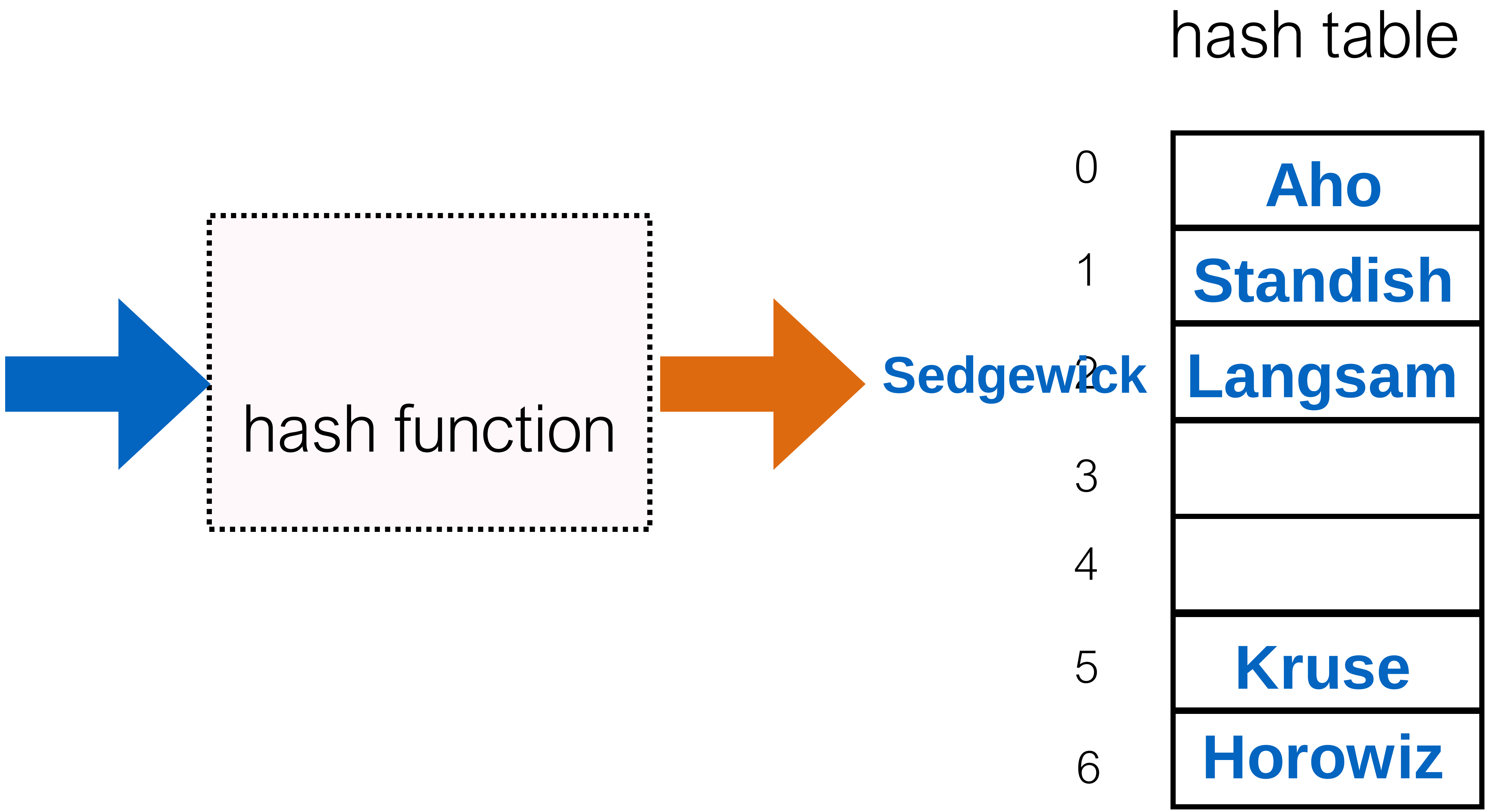
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



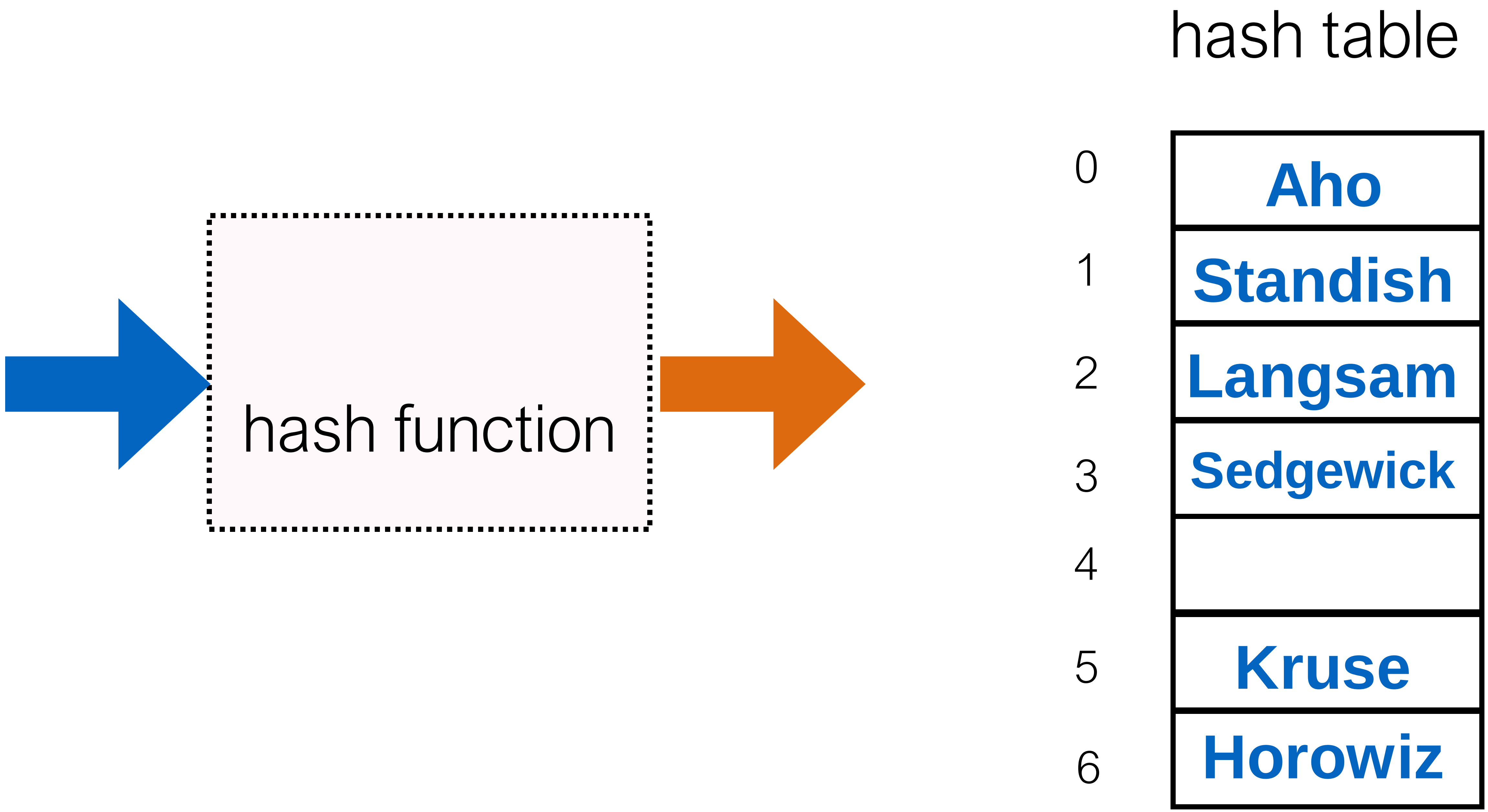
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



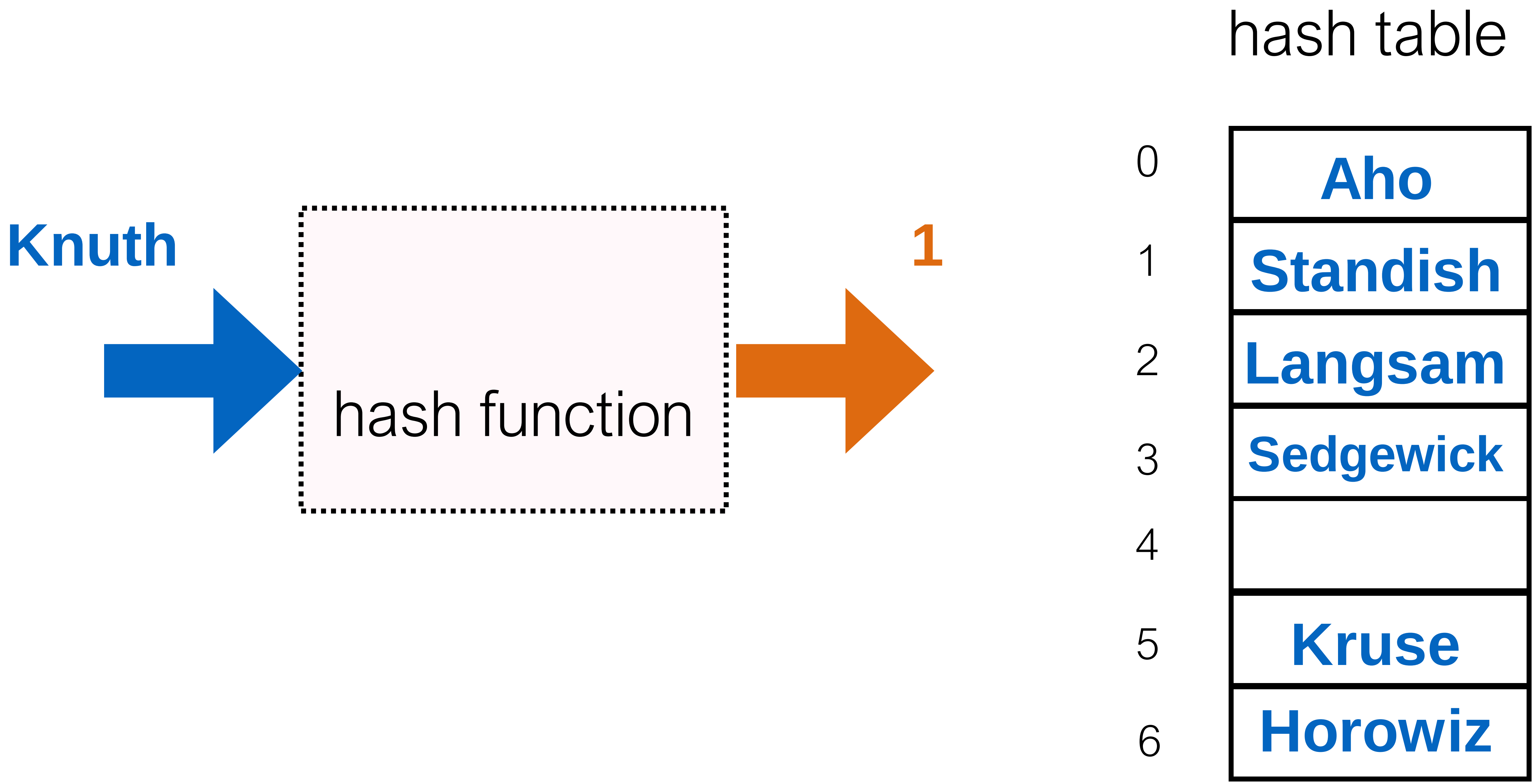
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



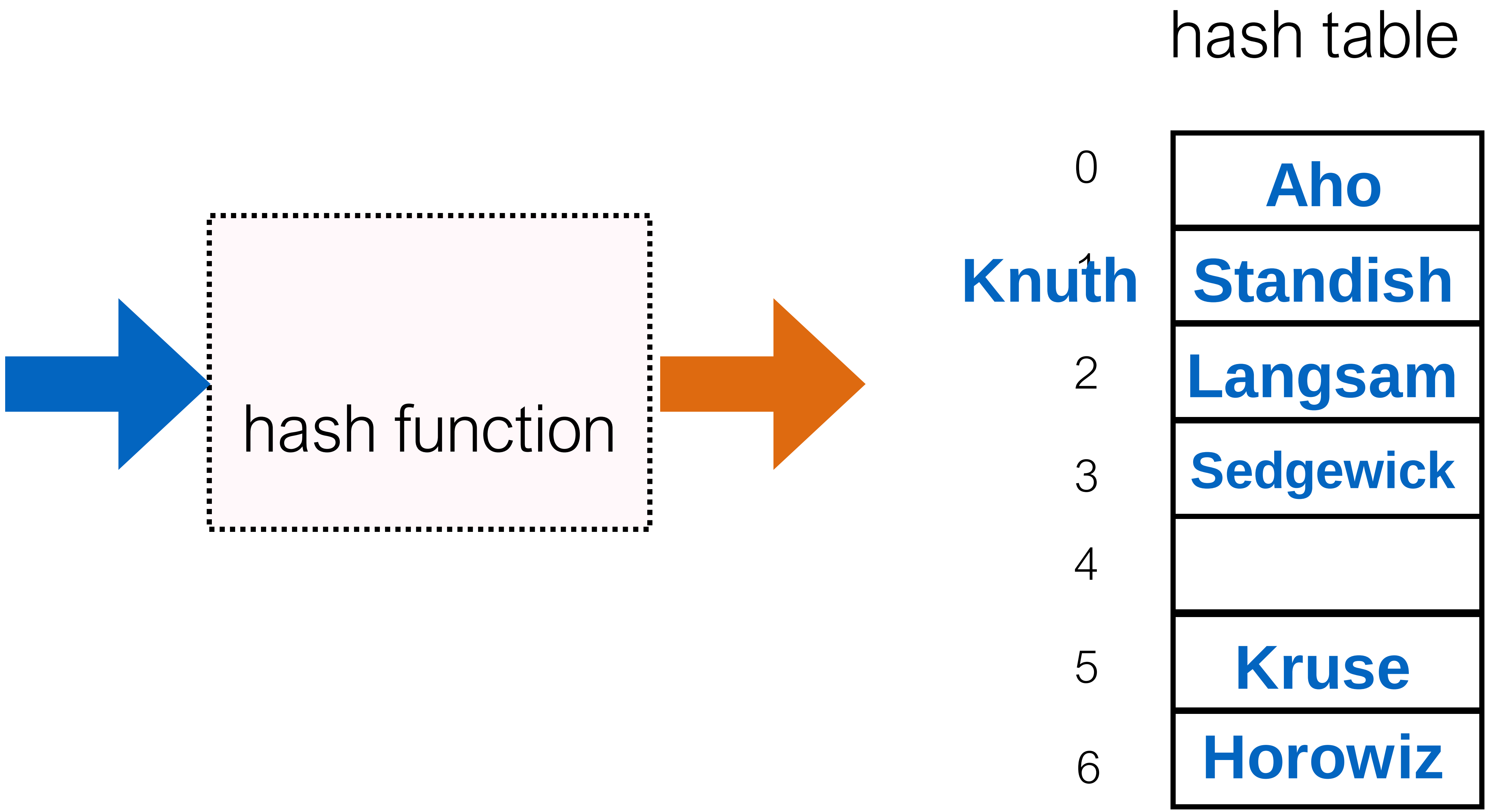
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



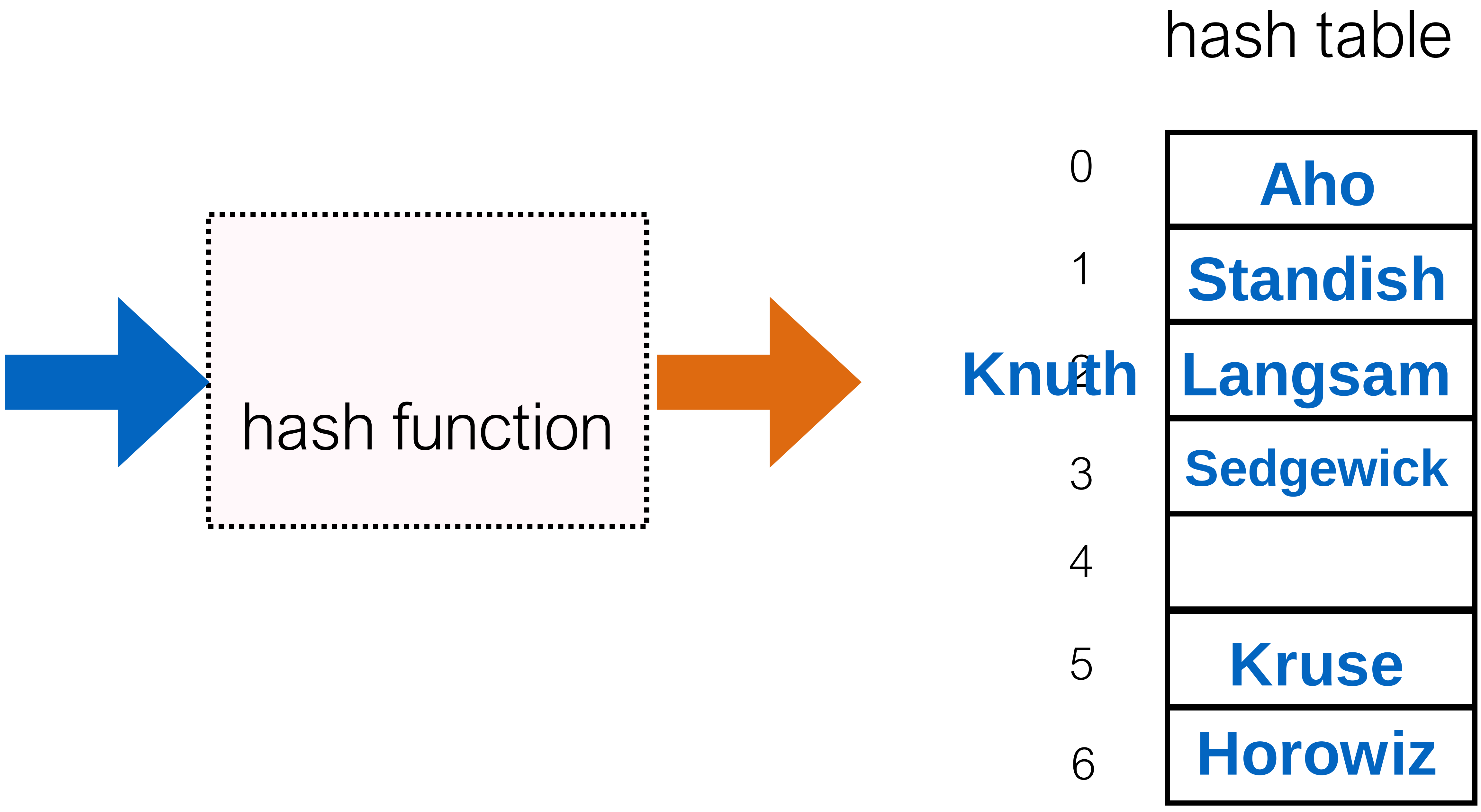
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



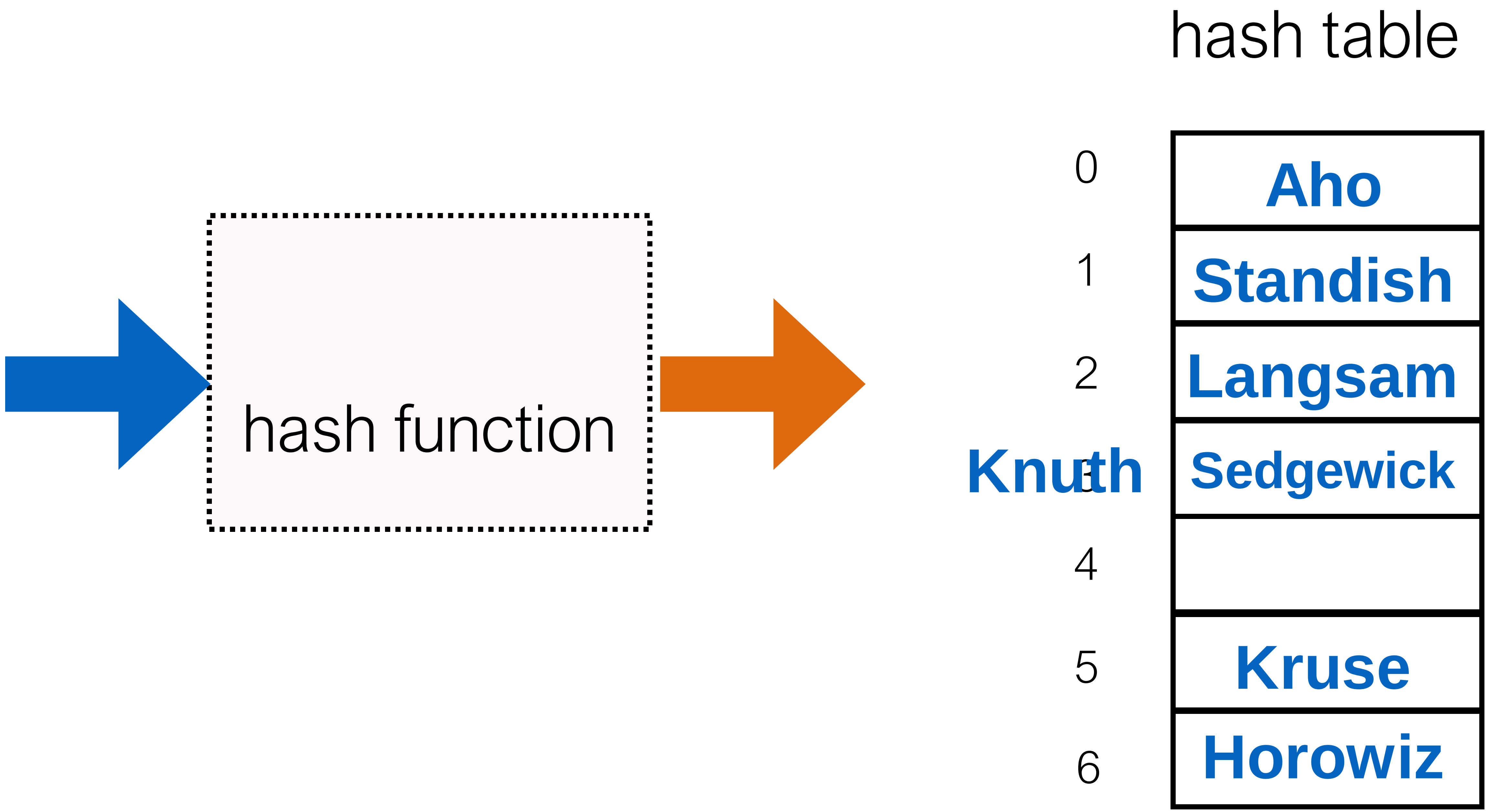
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



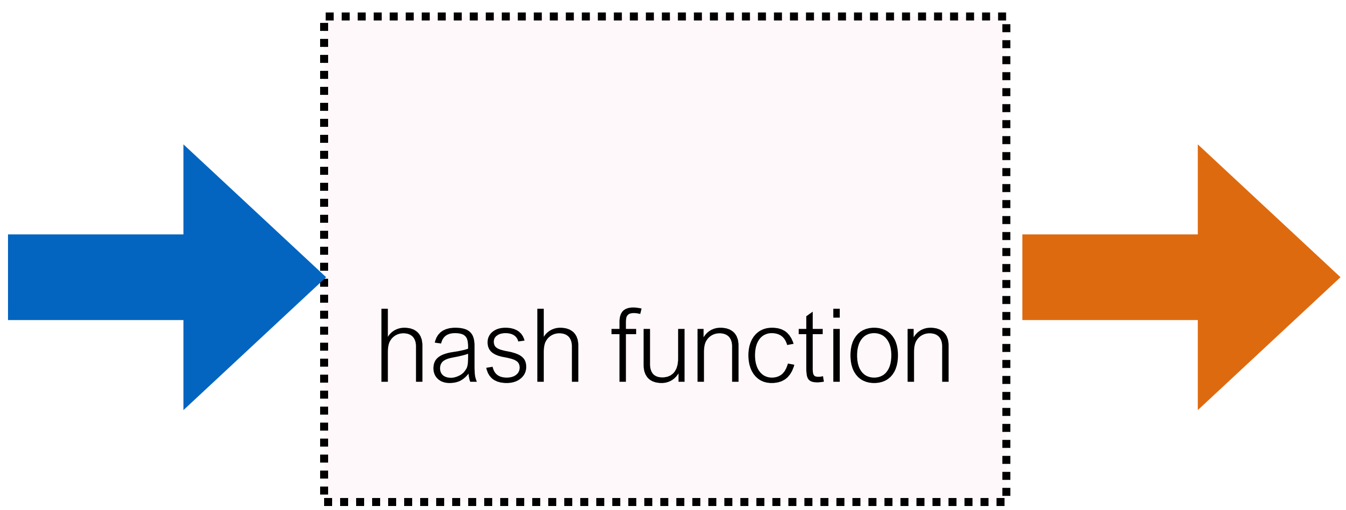
Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth

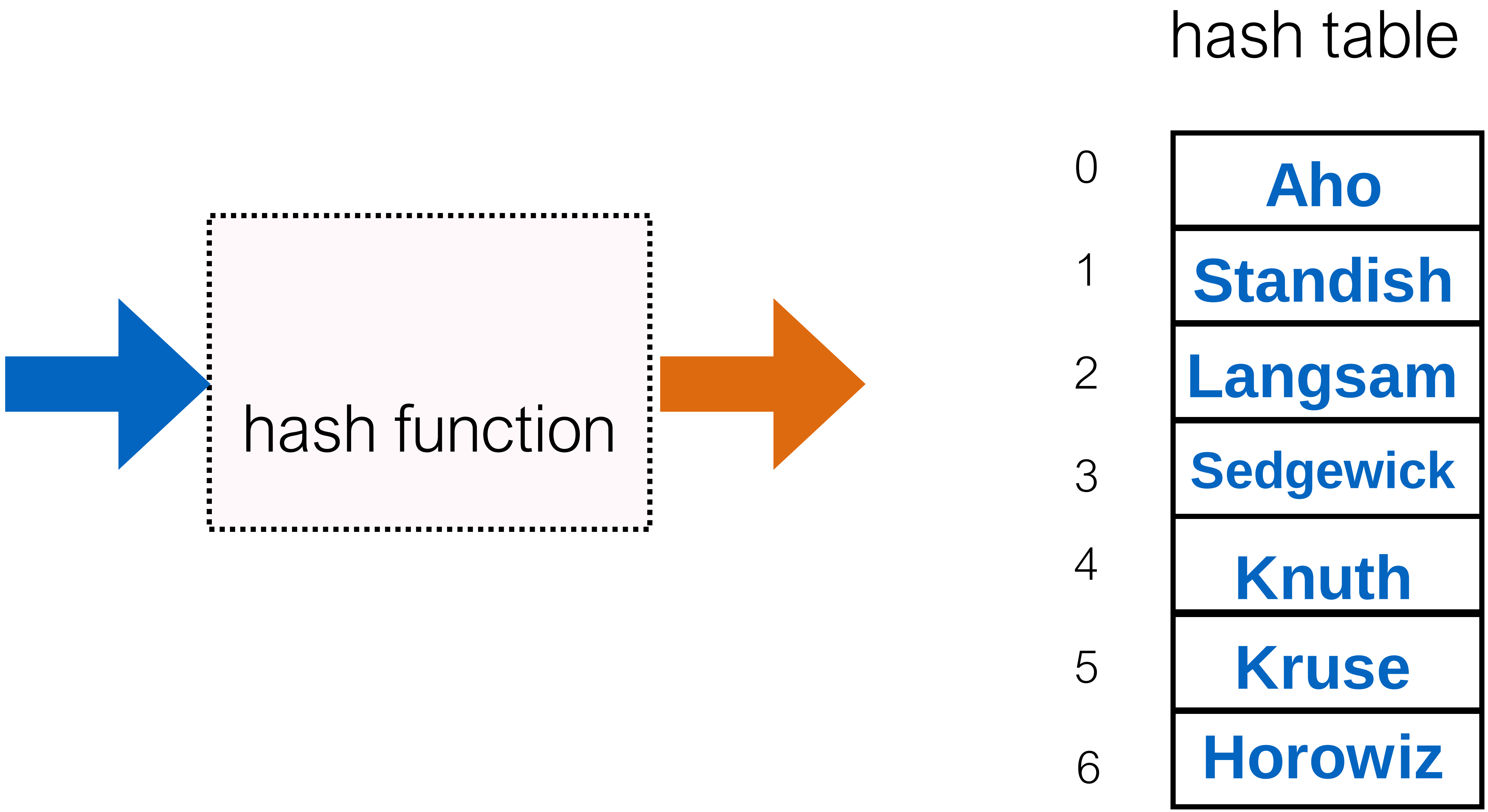


hash table

0	**Aho**
1	**Standish**
2	**Langsam**
3	**Sedgewick**
4	**Knuth**
5	**Kruse**
6	**Horowiz**

Example

Aho, Kruse, Standish, Horowiz, Langsam, Sedgewick, Knuth



Let’s implement a Table with Linear Probing….

```
class HashTableLinear:
```

```python
from referential_array import build_array


class HashTableLinear:

    def __init__(self, size=7919):
        self.count = 0
        self.table_size = size
        self.array = build_array(self.table_size)
```

default size,
a prime number

count: how many items I have in store

array: where I will store things

table_size: size of the underlying array, a prime…

```python
from referential_array import build_array


class HashTableLinear:

    def __init__(self, size=7919):
        self.count = 0
        self.table_size = size
        self.array = build_array(self.table_size)

    def __len__(self):
        return self.count
```

count: how many items I have in store

overloading operator **`len`** by implementing **`__len__`**

```python
from referential_array import build_array


class HashTableLinear:

    def __init__(self, size=7919):
        self.count = 0
        self.table_size = size
        self.array = build_array(self.table_size)

    def __len__(self):
        return self.count

    def hash_value(self, key):
        h = 0
        a = 31415
        for i in range(len(key)):
            h = (h * a + ord(key[i])) % self.table_size
        return h
```

Hash function with appropriately chosen constants

Open Addressing: Linear Probing

- **Insert item with hash value N**:
  - If array[N] is empty just put **item** there.
  - If there is already an item there:
    look for the first empty space in the array from N+1 (if any) and add it there

- Linear search from N until an empty slot is found

- **Things to think about**:
  - Full table (to avoid going into an infinite loop)
  - Restarting from position 0 if the end of table is reached
  - Finding an item with the same key.

Key	Hash value
Aho	0
Kruse	5
Standish	1
Horowitz	5
Langsam	5
Sedgewick	2
Knuth	1

hash table

0	**Aho**
1	**Standish**
2	**Langsam**
3	**Sedgewick**
4	**Knuth**
5	**Kruse**
6	**Horowiz**

We are storing the key only.

In practice you want to store also some data that you associate to each key.

Key	Data	Hash value
Aho	Data structures and algorithms	0
Kruse	Data structures and program design in C++	5
Standish	Data structures in Java	1
Horowitz	Fundamentals of Data Structures	5
Langsam	Data structures using C and C++	5
Sedgewick	Algorithms in C++	2
Knuth	The art of computer programming	1

hash table

We are storing the key only.

In practice you want to store also some data that you associate to each key.

hash table

	key	data
0	Aho	Data structures and algorithms
1	Standish	Data structures in Java
2	Langsam	Data structures using C and C++
3	Sedgewick	Algorithms in C++
4	Knuth	The art of computer programming
5	Kruse	Data structures and program design
6	Horowiz	Fundamentals of Data Structures

hash table

	key	data
0	(Aho ,	Data structures and algorithms)
1	(Standish ,	Data structures in Java)
2	(Langsam ,	Data structures using C and C++)
3	(Sedgewick ,	Algorithms in C++)
4	(Knuth ,	The art of computer programming)
5	(Kruse ,	Data structures and program design)
6	(Horowiz ,	Fundamentals of Data Structures)

(key , data)

Python tuple

my_tuple = (key , data)

Python tuple

my_tuple[0] = key

my_tuple[1] = data

Open Addressing: Linear Probing

(**key** , **data**)

- **Insert item with hash value N**:
  - If array[N] is empty just put **item** there.
  - If there is already an item there:
    look for the first empty space in the array from N+1 (if any) and add it there

- Linear search from N until an empty slot is found

- **Things to think about**:
  - Full table (to avoid going into an infinite loop)
  - Restarting from position 0 if the end of table is reached
  - Finding an item with the same key.

insert(key, data)

- Get the position N using the hash function, **N = hash(key)**
- If **array[N] is empty** just put the item **(key, data)** there.
- If there is <u>already an item there</u>:
  - If there is already something there, with the **same key** the user is **updating** the data
  - If there is already something there with a **different key**, you need to **find an empty spot**

What if the Table is full?

```
def insert(self, key, data):
```

```python
def insert(self, key, data):
    position = self.hash(key)
    for _ in range(self.table_size):
        if self.array[position] is None:  # found empty slot
            self.array[position] = (key, data)
            self.count += 1
            return
        elif self.array[position][0] == key:  # found key
            self.array[position] = (key, data)
            return
        else:  # not found, try next
            position = (position + 1) % self.table_size
    self.rehash()
    self.insert(key, data)
```

limit iterations to size of the table

the **key** of the tuple currently living at array[position]

(if full) move everything to a new larger table and try again

__setitem__

object.**__setitem__**(*self*, *key*, *value*)

Called to implement assignment to `self[key]`. Same note as for `__getitem__()`. This should only be implemented for mappings if the objects support changes to the values for keys, or if new keys can be added, or for sequences if elements can be replaced. The same exceptions should be raised for improper *key* values as for the `__getitem__()` method.

https://docs.python.org/3/reference/datamodel.html#object.__setitem__

```
>>> a = dict()
>>> a[123465] = "Julian"
>>> a[133123] = "Nicole"
>>> a[982211] = "David"
>>>
>>> a
{123465: 'Julian', 133123: 'Nicole', 982211: 'David'}
```

```python
def insert(self, key, data):
    position = self.hash(key)
    for _ in range(self.table_size):
        if self.array[position] is None:  # found empty slot
            self.array[position] = (key, data)
            self.count += 1
            return
        elif self.array[position][0] == key:  # found key
            self.array[position] = (key, data)
            return
        else:  # not found, try next
            position = (position + 1) % self.table_size
    self.rehash()
    self.insert(key, data)
```

```python
def __setitem__(self, key, data):
    position = self.hash(key)
    for _ in range(self.table_size):
        if self.array[position] is None:  # found empty slot
            self.array[position] = (key, data)
            self.count += 1
            return
        elif self.array[position][0] == key:  # found key
            self.array[position] = (key, data)
            return
        else:  # not found, try next
            position = (position + 1) % self.table_size
    self.rehash()
    self.__setitem__(key, data)
```

What is the best case time complexity of __setitem__ ?

A) O(1)

B) O(log N)

C) O(N)

D) $O(N^2)$

What is the worst case time complexity of __setitem__ ?

A) O(1)

B) O(log N)

C) O(N)

D) $O(N^2)$

Note: typical array sizes >> number of items

```python
def __str__(self):
    result = ""
    for item in self.array:
        if item is not None:
            (key, value) = item
            result += "(" + str(key) + "," + str(value) + ")"
    return result
```

Conclusion

- Hash Tables are one of the most used data type: You have a very good chance of using them in your career.
- They are very simple conceptually and very powerful in practice.
- A significant amount of **experimental evaluation** is usually needed to fine **tune the hash function** and the TABLESIZE