

# Хэш-таблица

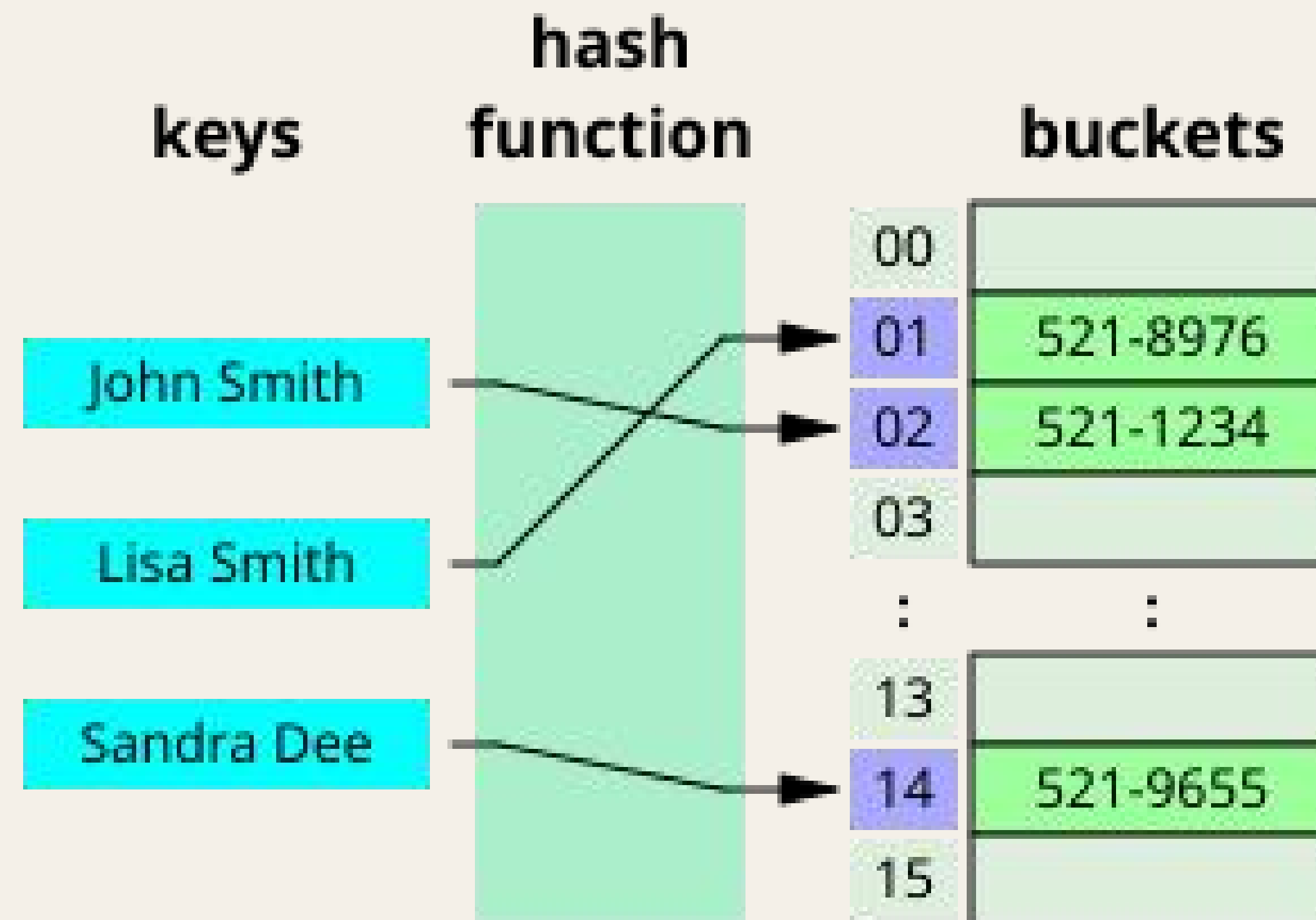
Практические занятия

БИБТ-24-17

**Надежда Анисимова**  
ms teams m2102039@edu.misis.ru

# Хэш-таблица

Хэш-таблица - тип данных, с помощью которого хранятся пары «ключ-значение». Однако индекс хранения значения определяется хэш-функцией



# А если просто словарь?

Как найти значение по  
ключу Orange?



Apple	2
Banana	8
Pear	3
Orange	1
Plum	10

# Если хэш-таблица, то посчитать хэш

Apple
Banana
Pear
Orange
Plum

$k = \text{Orange}$

$h(\text{Orange}) = 3$

индекс  
массива

0	2
1	8
2	3
3	1
4	10

Пусть  $k$  - ключ, тогда  $h(k) = \text{ind}$  хранения данных по ключу

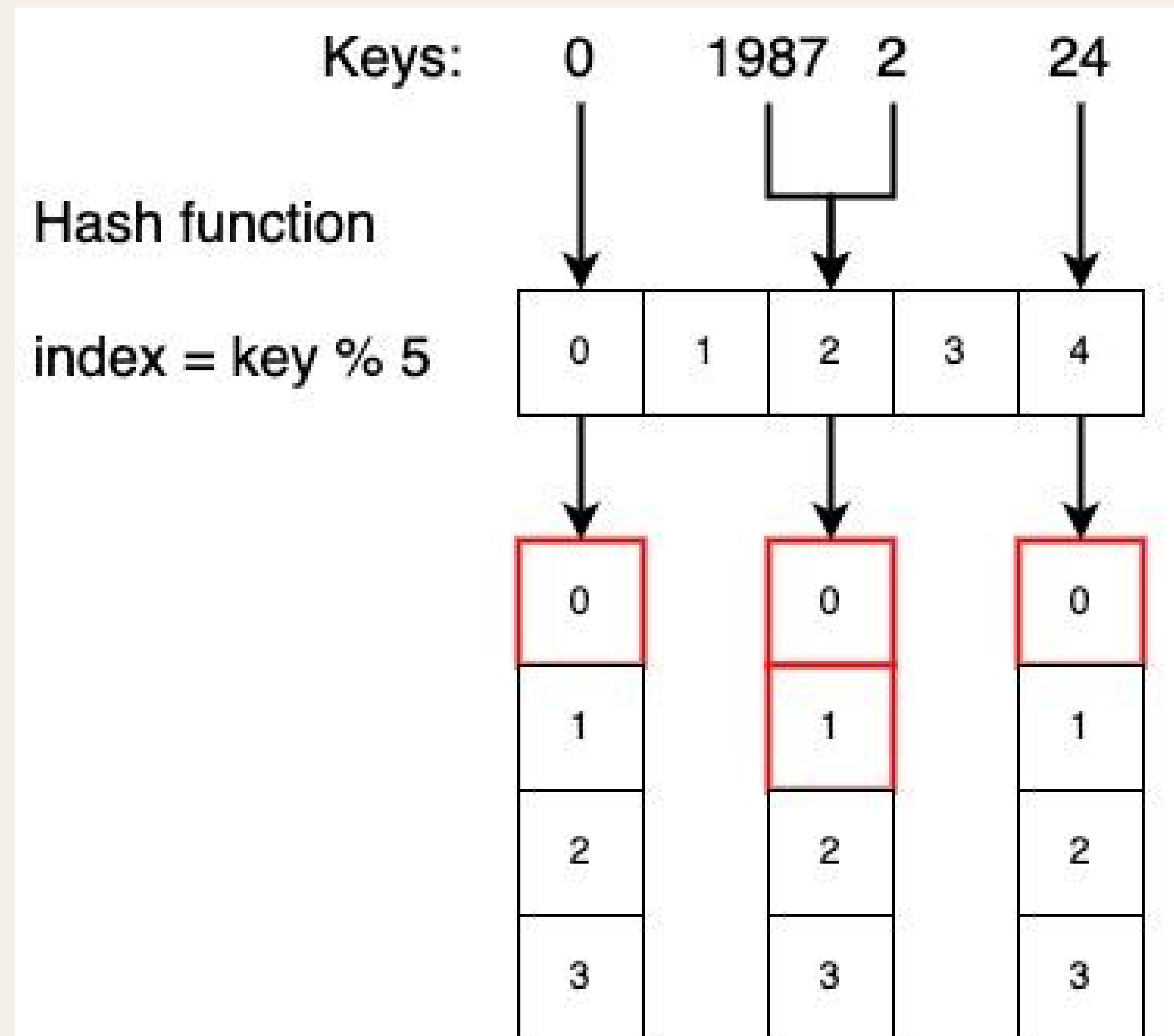
# Примеры хэш-функций

1. Метод деления:  $h(K) = k \bmod M$
2. Метод умножения:  $\text{hash} = ((k \times A) \bmod P) \bmod N$   $N$  - размер таблицы
3. Просто сумма чисел/подстрок

# Остаток по модулю, чтобы не выйти за границы массива

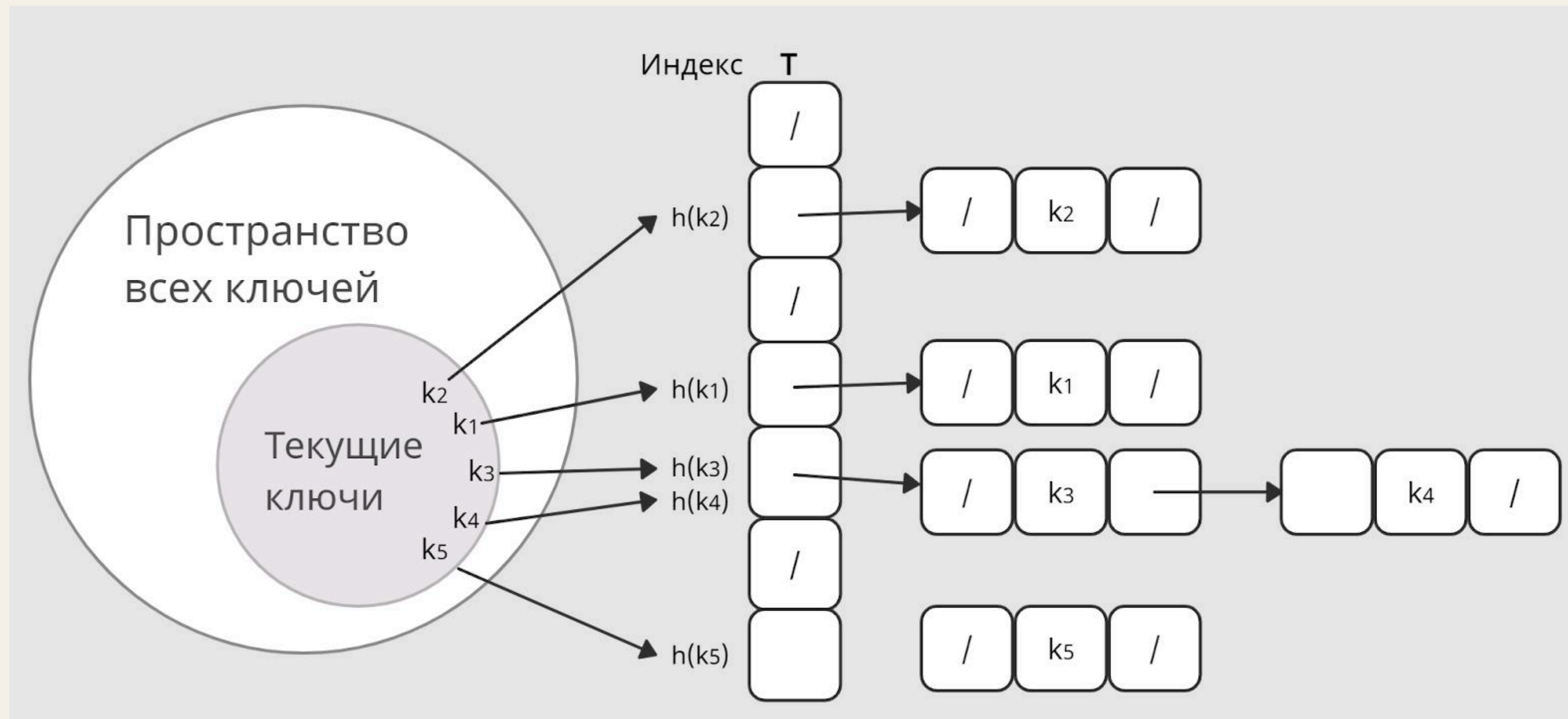
В нашем случае будем не просто остаток брать, чтобы ключи могли быть строками:

```
hash_function(key) % table.size();
```



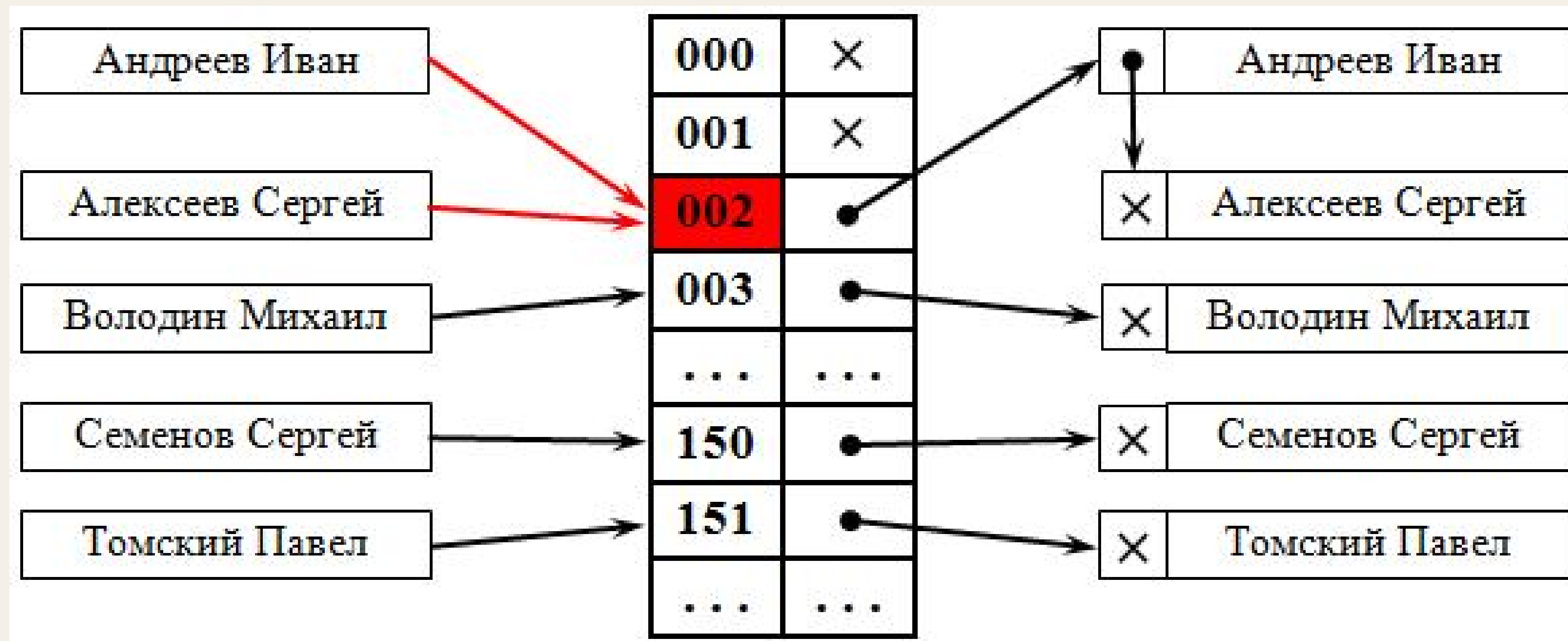
# Проблемы с коллизиями

Для одного и того же ключа может быть одинаковое значение хэша(индекса в таблице)



# Проблемы с коллизиями

Для одного и того же ключа может быть одинаковое значение хэша(индекса в таблице)





# Способы борьбы с коллизиями

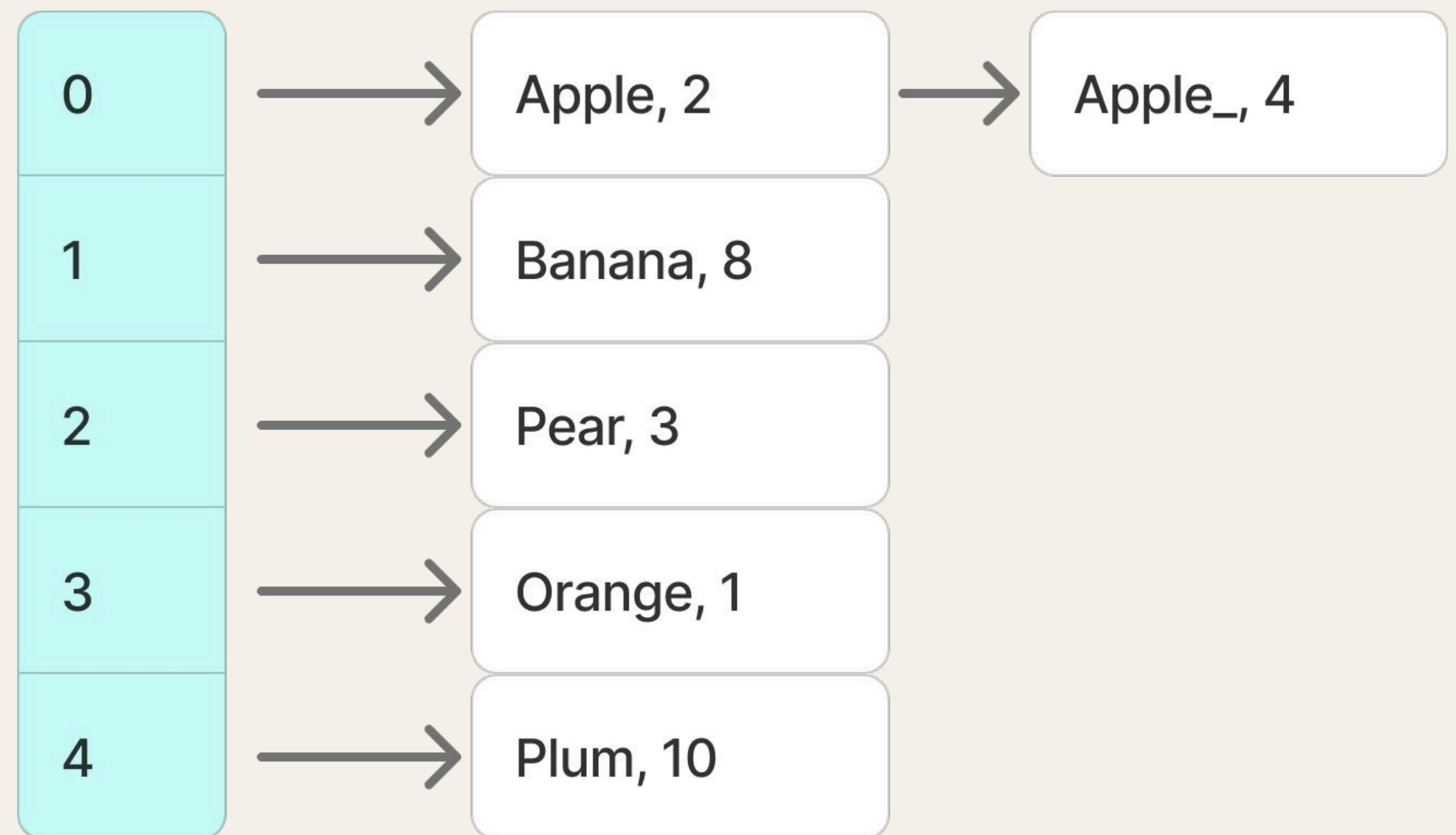
# O2

# Хэш-таблица с цепочками (Chaining)

Просто храним  
значение

0	Apple, 2
1	Banana, 8
2	Pear, 3
3	Orange, 1
4	Plum, 10

Список из элементов (key,  
value)



Проблема наивной итерации по такому массиву - <https://habr.com/ru/companies/vk/articles/660265/>

# Открытая адресация (Open Addressing)

Нет цепочки - только  
таблица, однако особым  
способом выбирается  
какой следующий индекс  
заполнить

Pear



0	Apple, 2
1	Banana, 8
2	
3	
4	Plum, 10

# Открытая адресация (Open Addressing)

## Линейное пробирование (Linear Probing)

Pear



Перебираем пока не найдем  
свободную ячейку

0	Apple, 2
1	Banana, 8
2	
3	
4	Plum, 10

# Открытая адресация (Open Addressing)

## Квадратичное пробирование (Quadratic Probing)

Pear

1 - занято

$$1 + 1^2$$

Перебираем с квадратичным  
шагом  $i + i^2$

0	Apple, 2
1	Banana, 8
2	
3	
4	Plum, 10

# Открытая адресация (Open Addressing)

## Двойное хэширование (Double Hashing)

Pear

1 - занято

$1 + \text{hash\_2}(1)$

Двойное хэширование  
использует вторую хэш-  
функцию для вычисления  
следующего индекса при  
коллизии  $i + \text{hash\_2}(i)$

0	Apple, 2
1	Banana, 8
2	
3	
4	Plum, 10

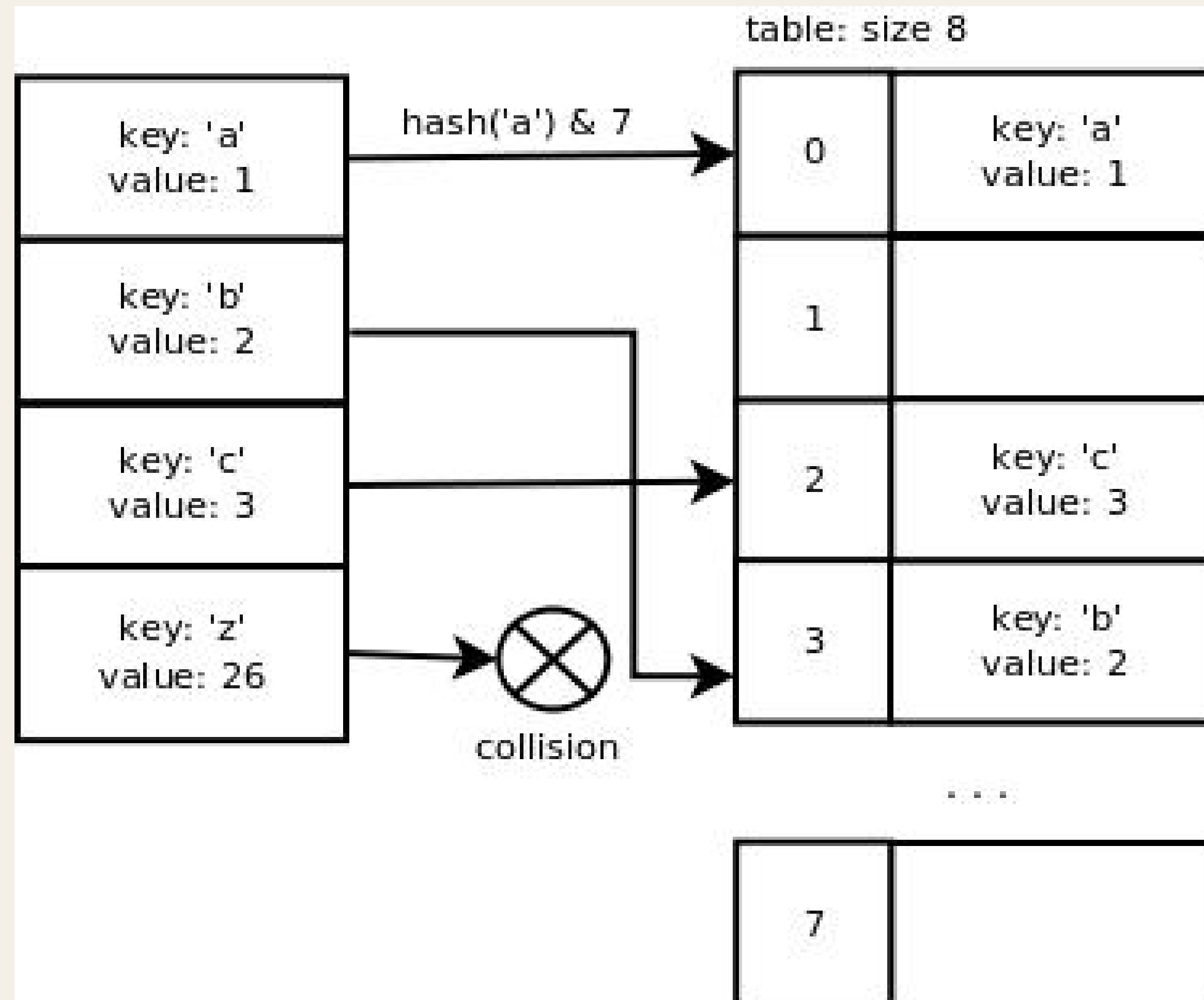
# Хэш-таблицы в питоне

03

# Словарь dict

+ квадратичное  
пробирование с  
модификациями

- поэтому ключ  
должен быть  
hashable



<https://habr.com/ru/companies/otus/articles/448350/>

<https://github.com/python/cpython/blob/main/Objects/dictobject.c>

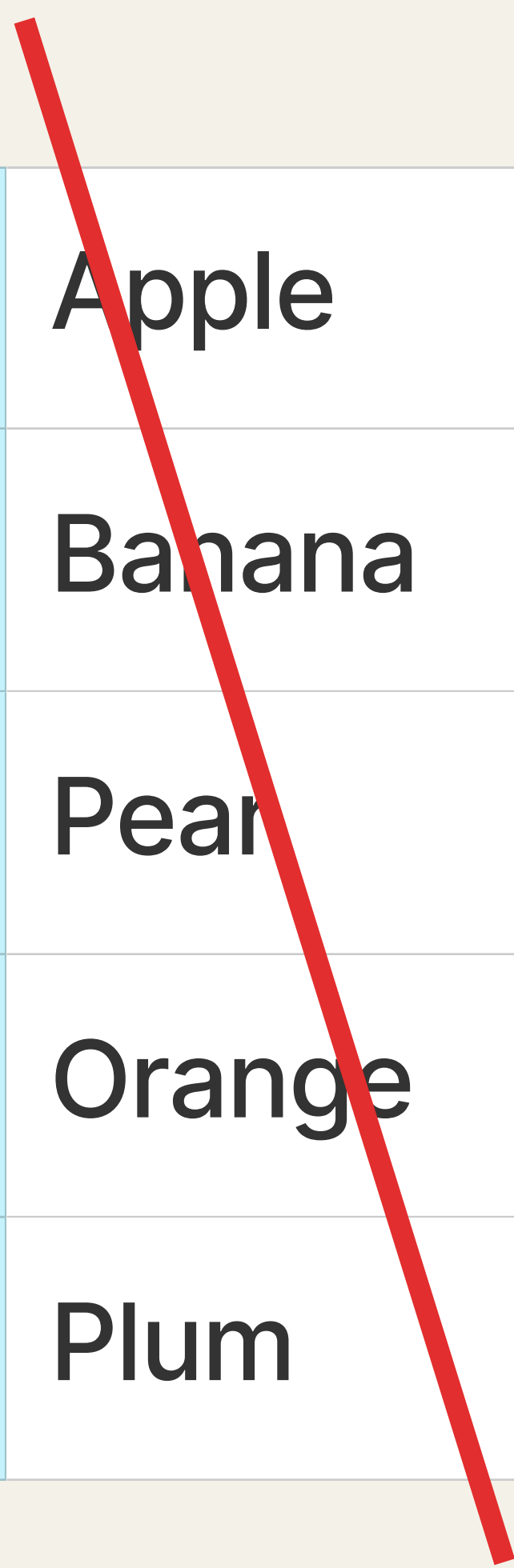


# Множество set и frozenset

Храним только  
значение → не ключ, а  
значение должно быть  
hashable

- сам set не hashable,  
а вот frozenset –  
hashable

Только значения



0	Apple	2
1	Banana	8
2	Pear	3
3	Orange	1
4	Plum	10

# Дз – словарик хэш- таблица