ES6 коллекции на примере V8: у ней внутре неонка

Андрей Печкуров

О докладчике

- Пишу на Java (очень долго), Node.js (долго)
- Node.js core collaborator
- Интересы: веб, архитектура, распределенные системы, производительность
- Можно найти тут:
  - https://twitter.com/AndreyPechkurov
  - https://github.com/puzpuzpuz
  - https://medium.com/@apechkurov

ECMAScript 2015 (ES6) привнес в JS стандартные коллекции:

- Map
- Set
- WeakMap
- WeakSet

```
const map = new Map();
map.set('foo', { bar: 'baz' });
for (let [key, value] of map) {
  console.log(`${key}:`, value);
}

const set = new Set();
set.add('foo');
set.add('bar');
set.forEach((item) => {
  console.log(item);
});
```

Спецификация не настаивает ни на чем конкретном:

> Map object must be implemented using **either hash tables or other mechanisms** that, on average, provide access times that are sublinear on the number of elements in the collection. The data structures used in this Map objects specification is only intended to describe the required observable semantics of Map objects. It is not intended to be a viable implementation model.

java.util

Interface Map<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

All Known Subinterfaces:

Bindings, ConcurrentMap<K,V>, ConcurrentNavigableMap<K,V>,
LogicalMessageContext, MessageContext, NavigableMap<K,V>,
SOAPMessageContext, SortedMap<K,V>

All Known Implementing Classes:

AbstractMap, Attributes, AuthProvider, ConcurrentHashMap,
ConcurrentSkipListMap, EnumMap, HashMap, Hashtable, IdentityHashMap,
LinkedHashMap, PrinterStateReasons, Properties, Provider, RenderingHints,
SimpleBindings, TabularDataSupport, TreeMap, UIDefaults, WeakHashMap

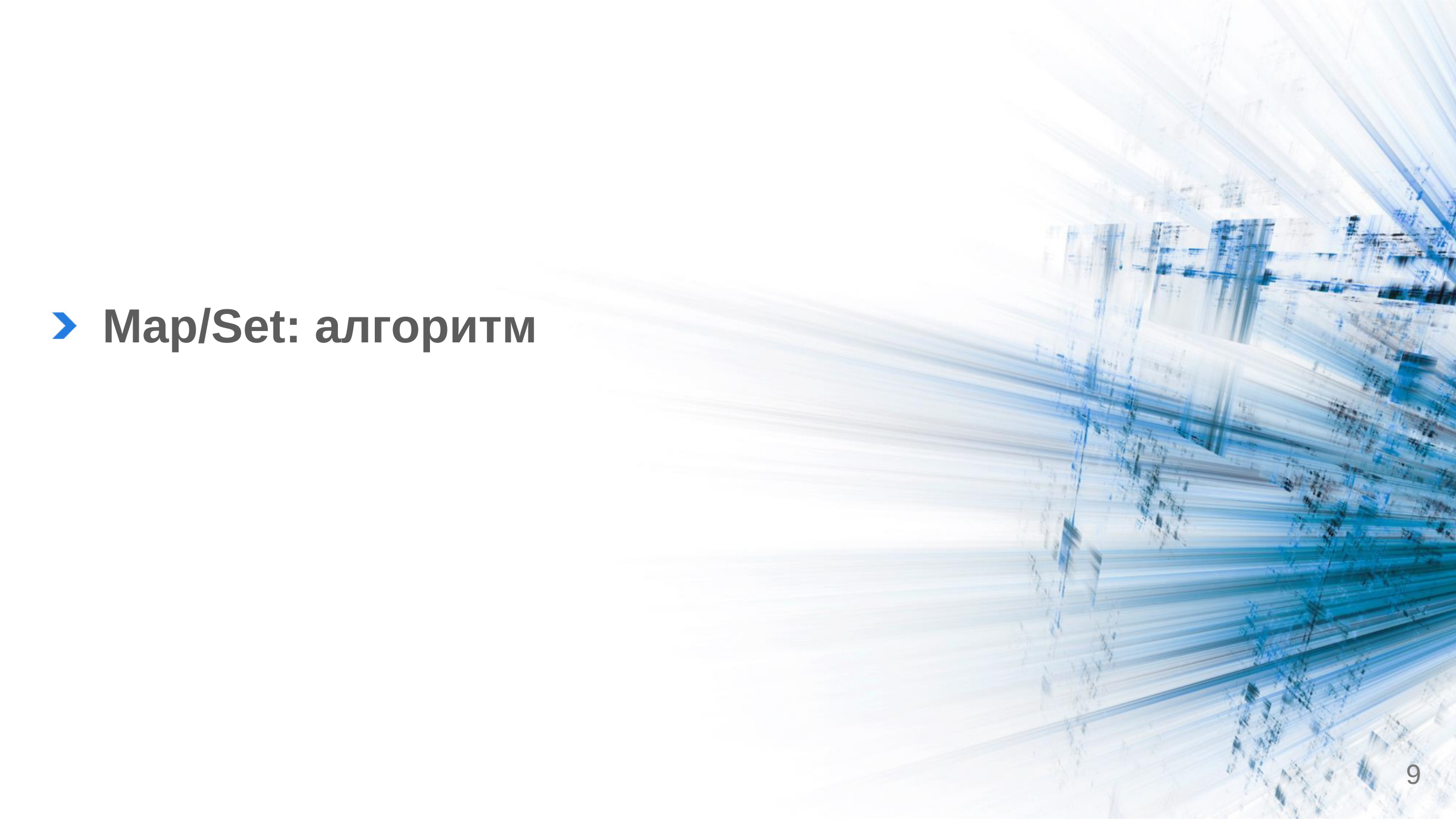
Disclamer

- Изложение основано на V8 8.4, Node.js commit 238104c
- Полагаться можно (и нужно) только на спецификацию ECMAScript
- Автор не работает в команде V8

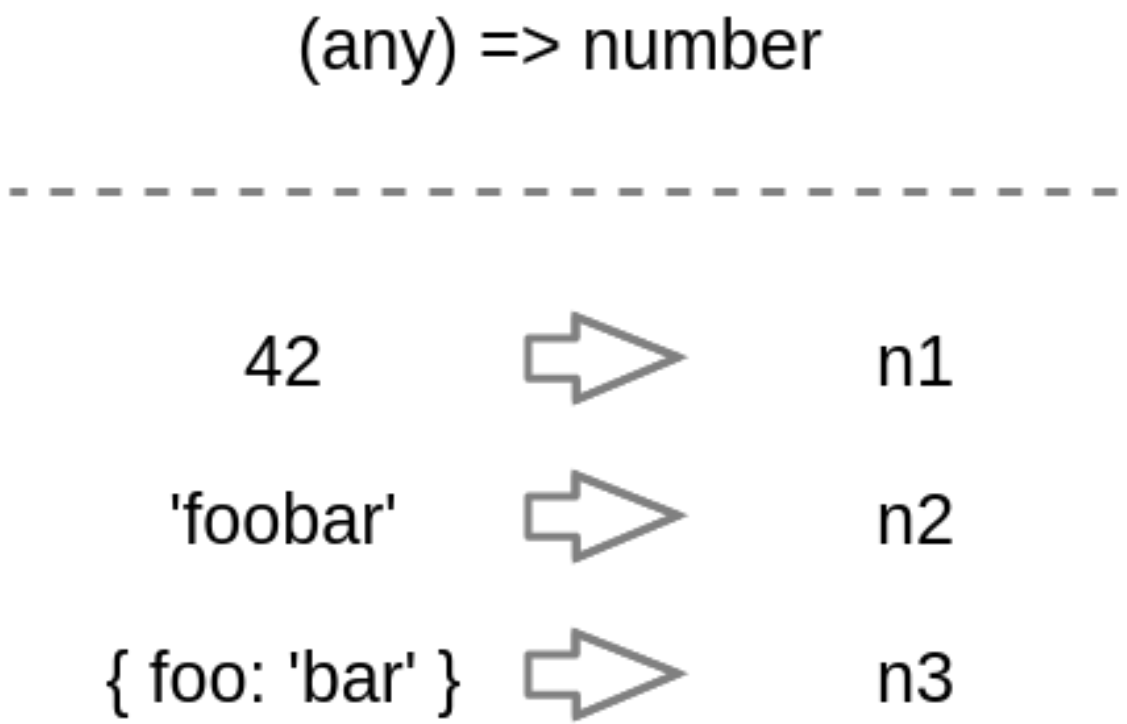
План на сегодня

- Map/Set
  - Алгоритм
  - Особенности реализации
  - Сложность
  - Память
- WeakMap/WeakSet
  - Алгоритм
  - Особенности реализации

Map/Set: алгоритм



Хеш-функция

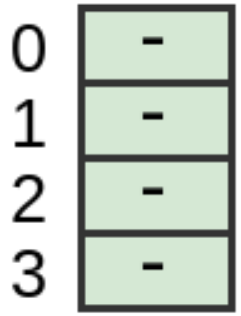


Пустая хеш-таблица

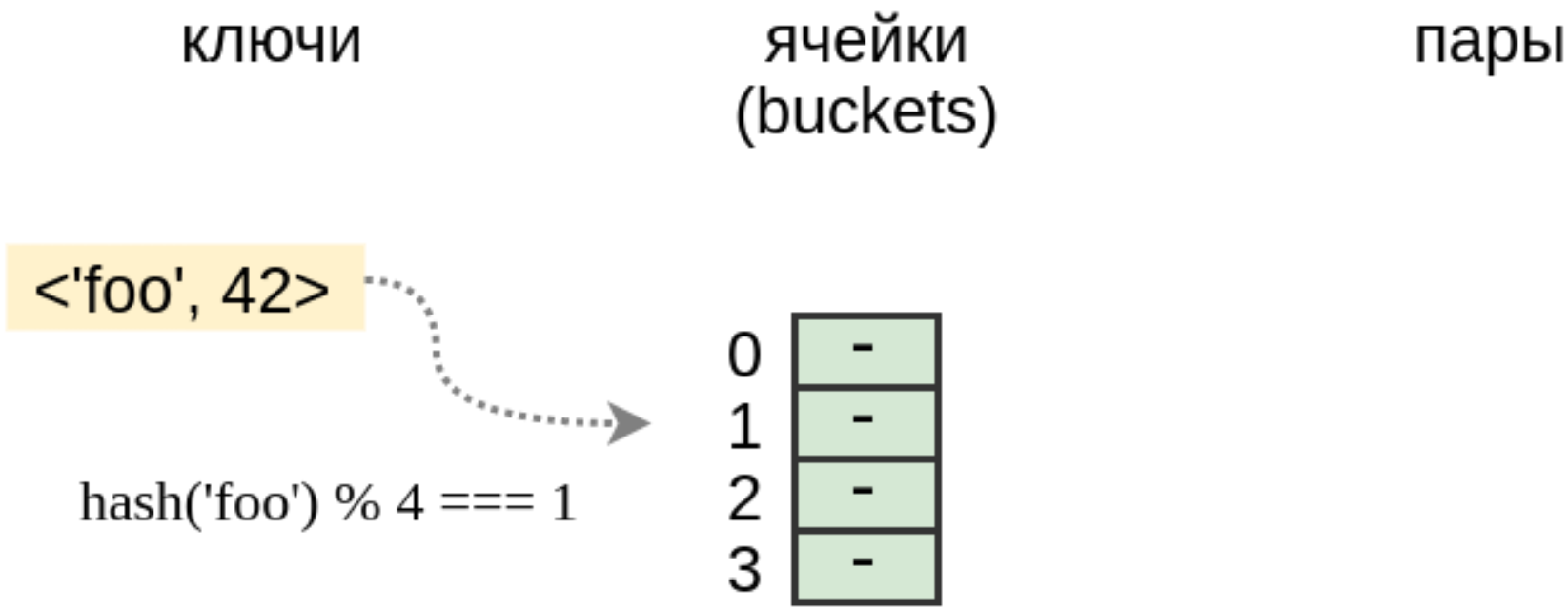
ключи

ячейки
(buckets)

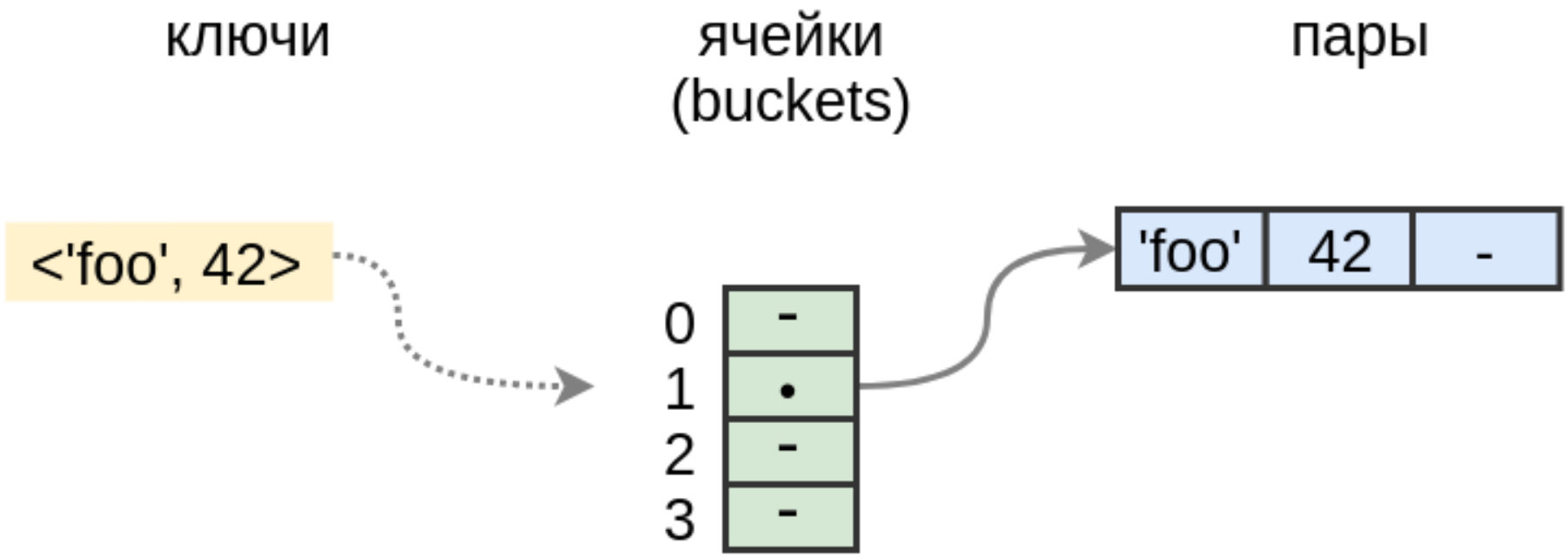
пары



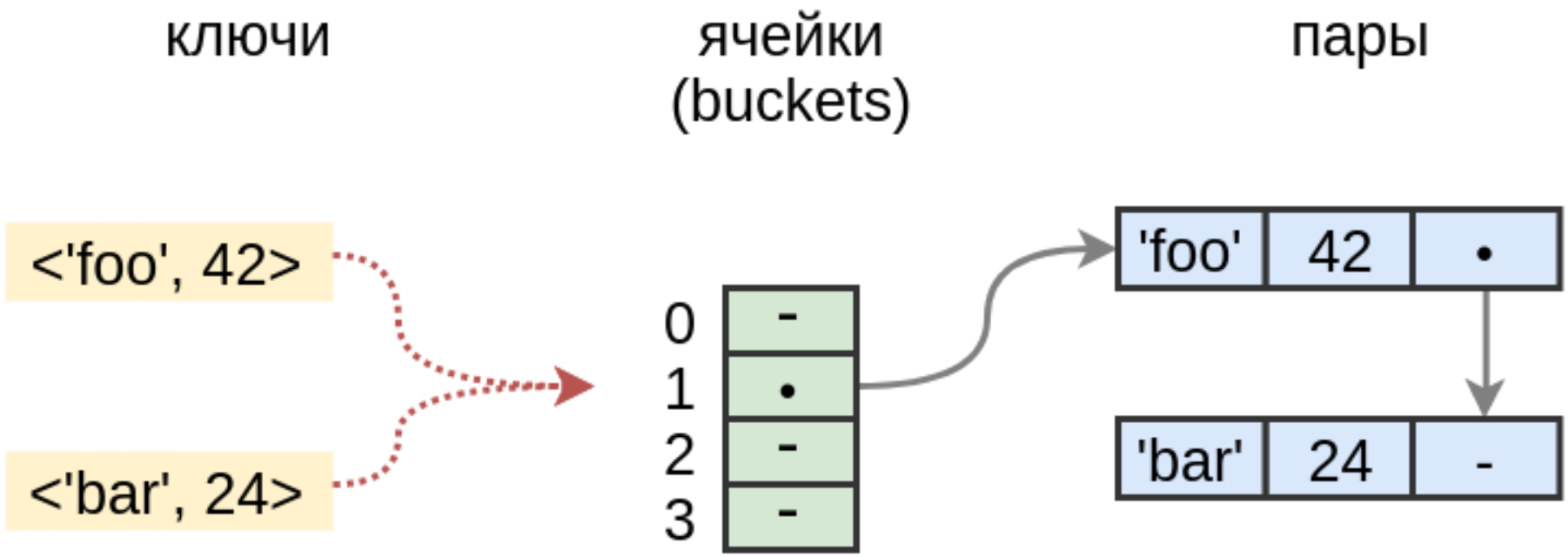
Вставка



Результат вставки



Обработка коллизий



Что внутри у Map/Set?

В силу спецификации в Map/Set не может быть "классической" хеш-таблицы

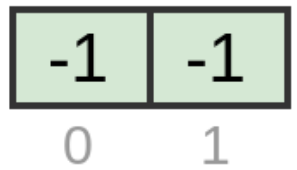
> When the forEach method is called with one or two arguments, the following steps are taken:
> ...
> 7. Repeat for each Record {[[key]], [[value]]} e that is an element of entries, in original key **insertion order**

V8 реализует deterministic hash tables (Tyler Close)

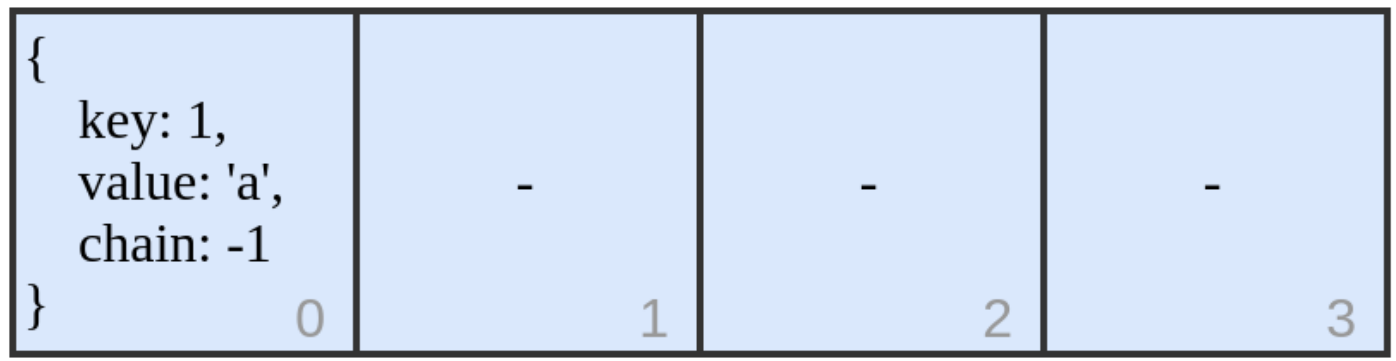
```
interface CloseTable {
    hashTable: number[];
    dataTable: Entry[];
    nextSlot: number;
    size: number;
}

interface Entry {
    key: any;
    value: any;
    chain: number;
}
```

hashTable:



dataTable:



nextSlot: 0

size: 0

table.set(1, 'a')

hashTable:

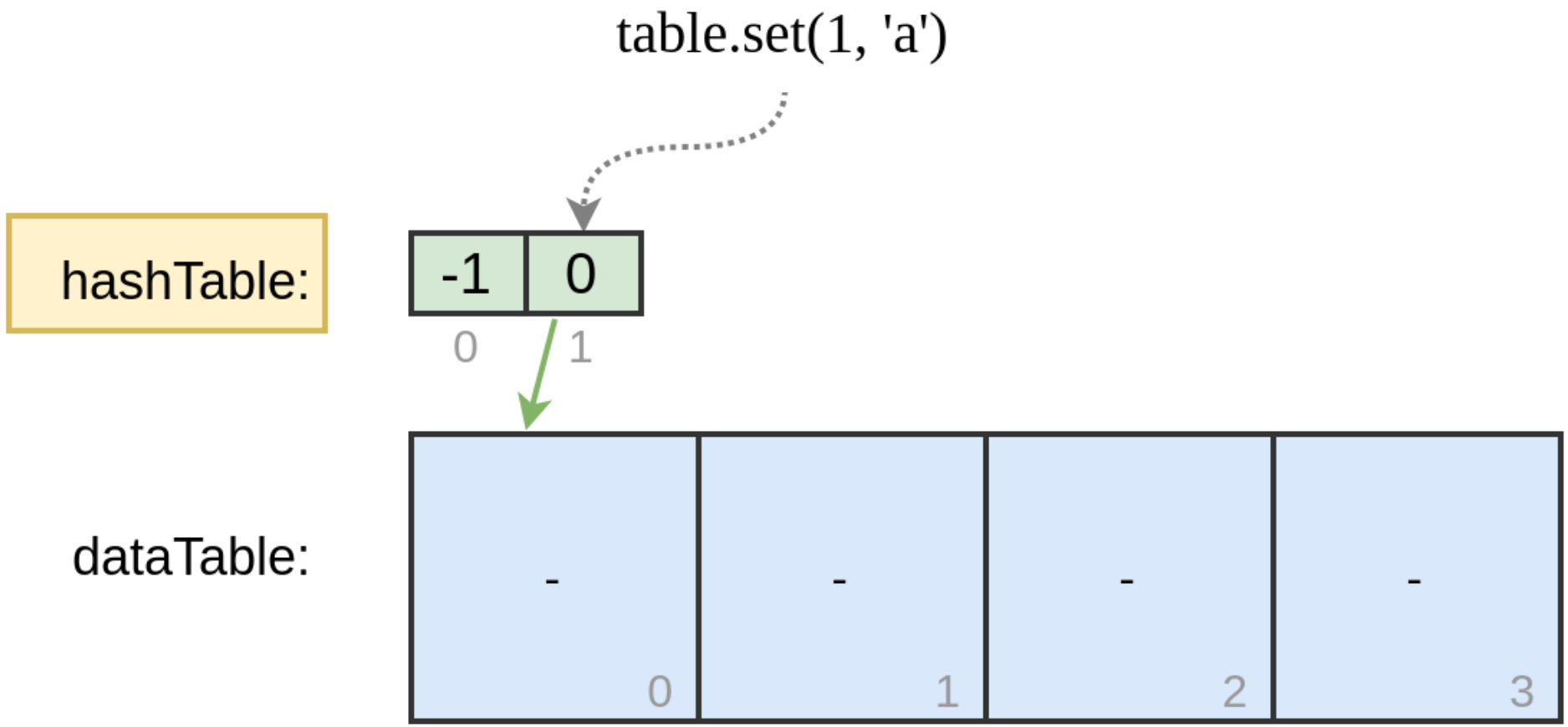
-1	-1
0	1

dataTable:

-	-	-	-
0	1	2	3

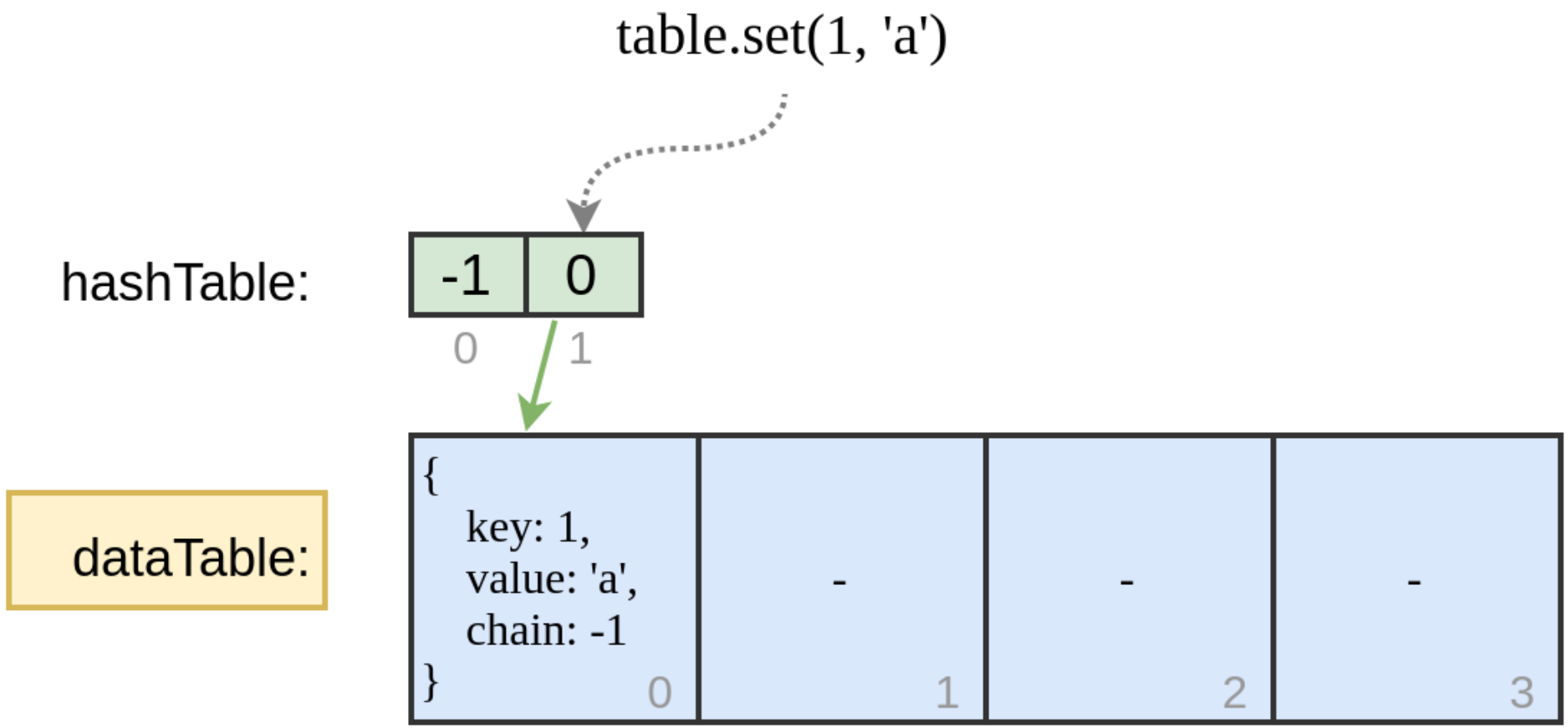
nextSlot: 0

size: 0



nextSlot: 0

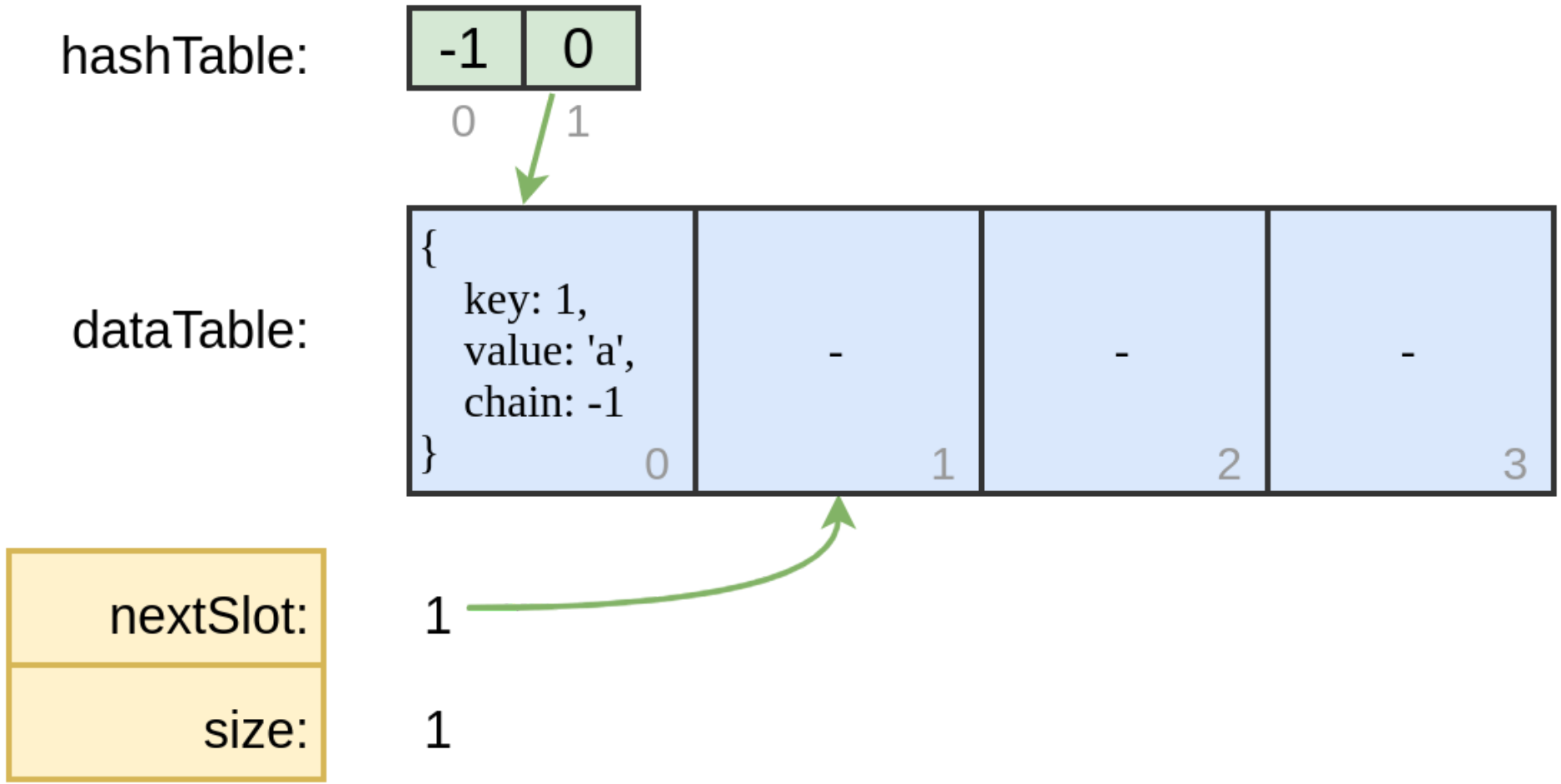
size: 0

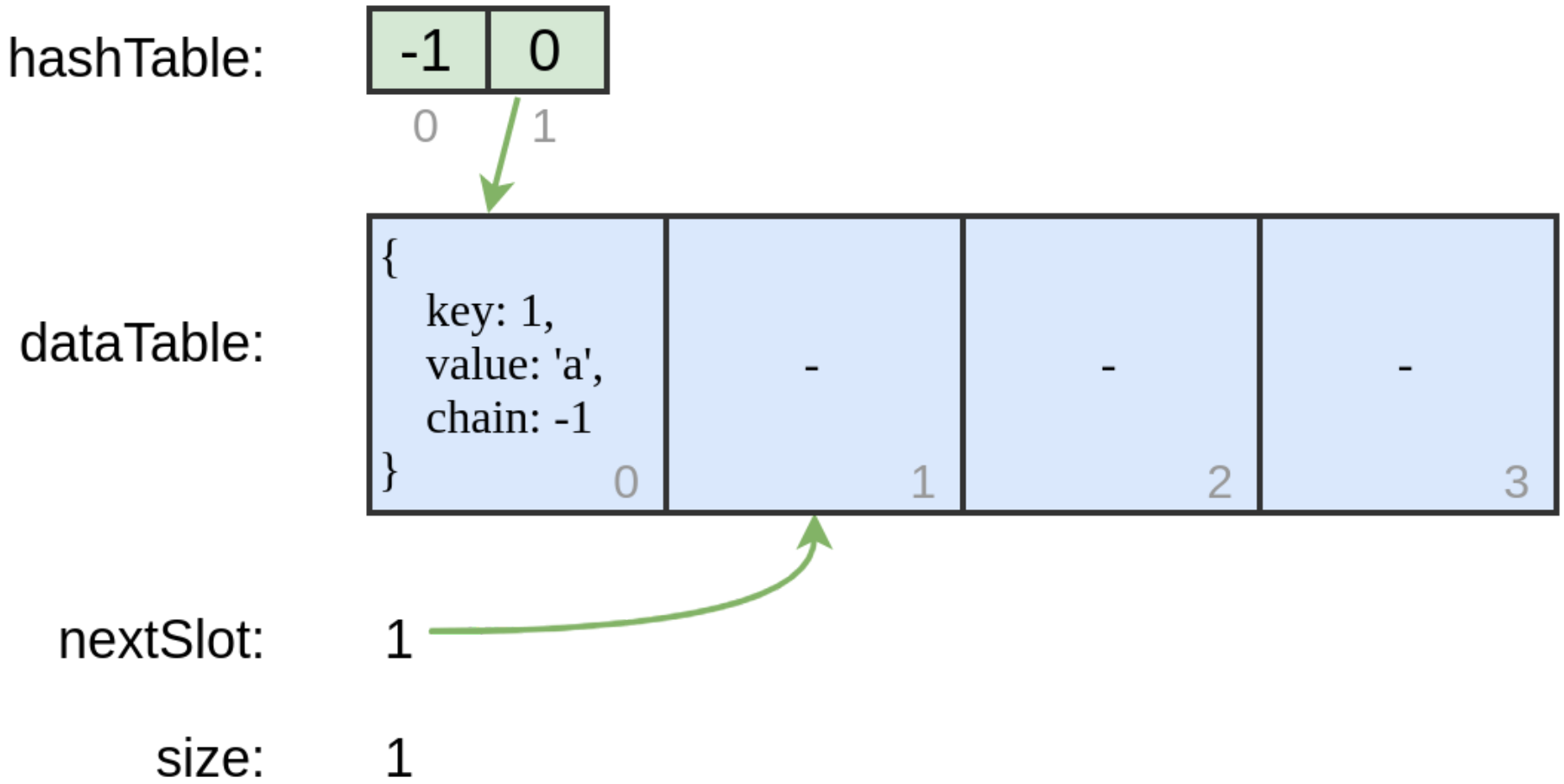


nextSlot: 0

size: 0

table.set(1, 'a')


hashTable:
-1
0
0
1
dataTable:
{
key: 1,
value: 'a',
chain: -1
}
0
-
1
-
2
-
3
nextSlot:
1
size:
1


hashTable:
-1
0
0
1
dataTable:
{
key: 1,
value: 'a',
chain: -1
}
0
-
1
-
2
-
3
nextSlot:
1
size:
1

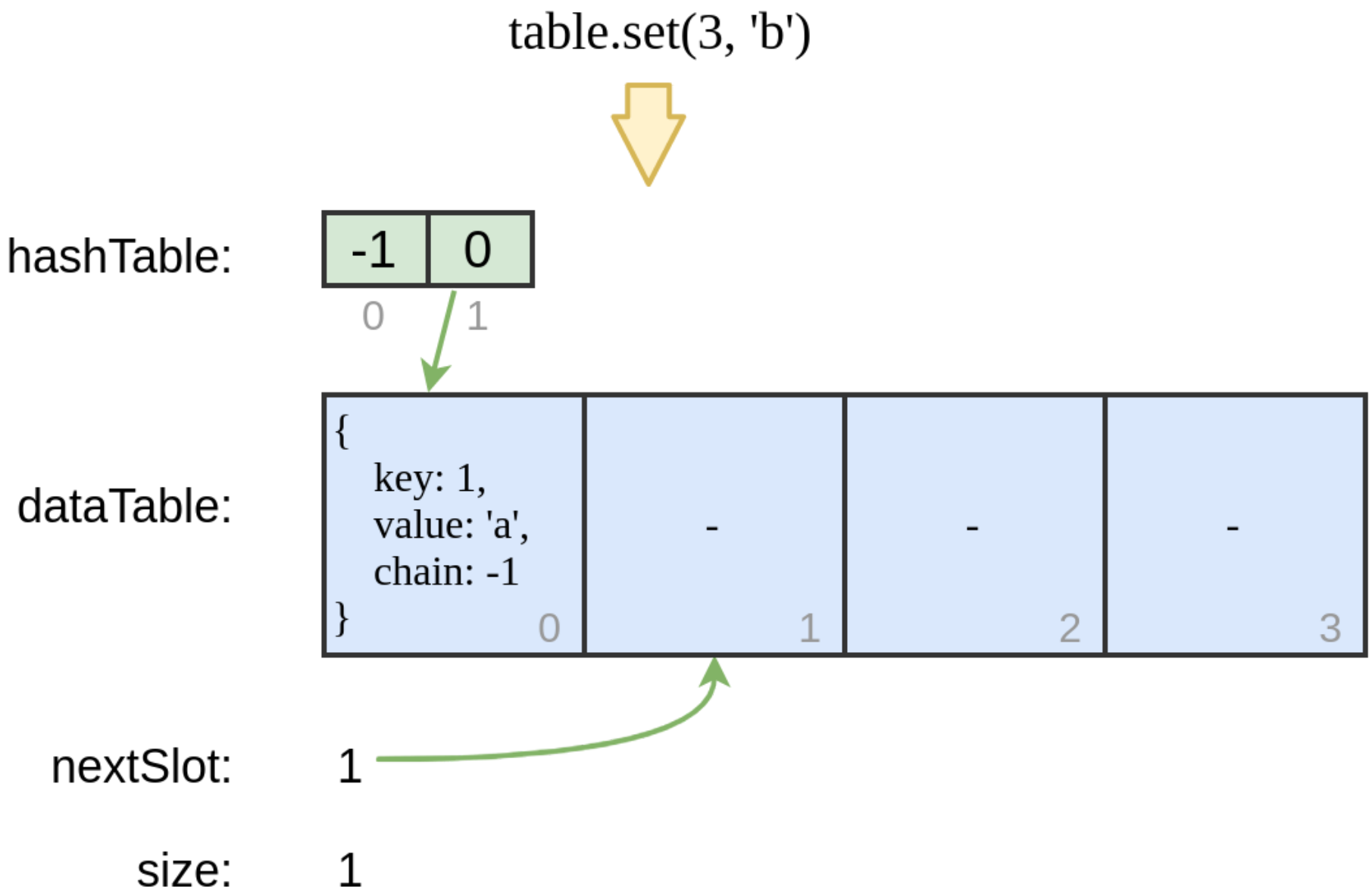


table.set(3, 'b')
hashTable:
-1
0
0
1
dataTable:
{
key: 1,
value: 'a',
chain: -1
}
0
-
1
-
2
-
3
nextSlot:
1
size:
1

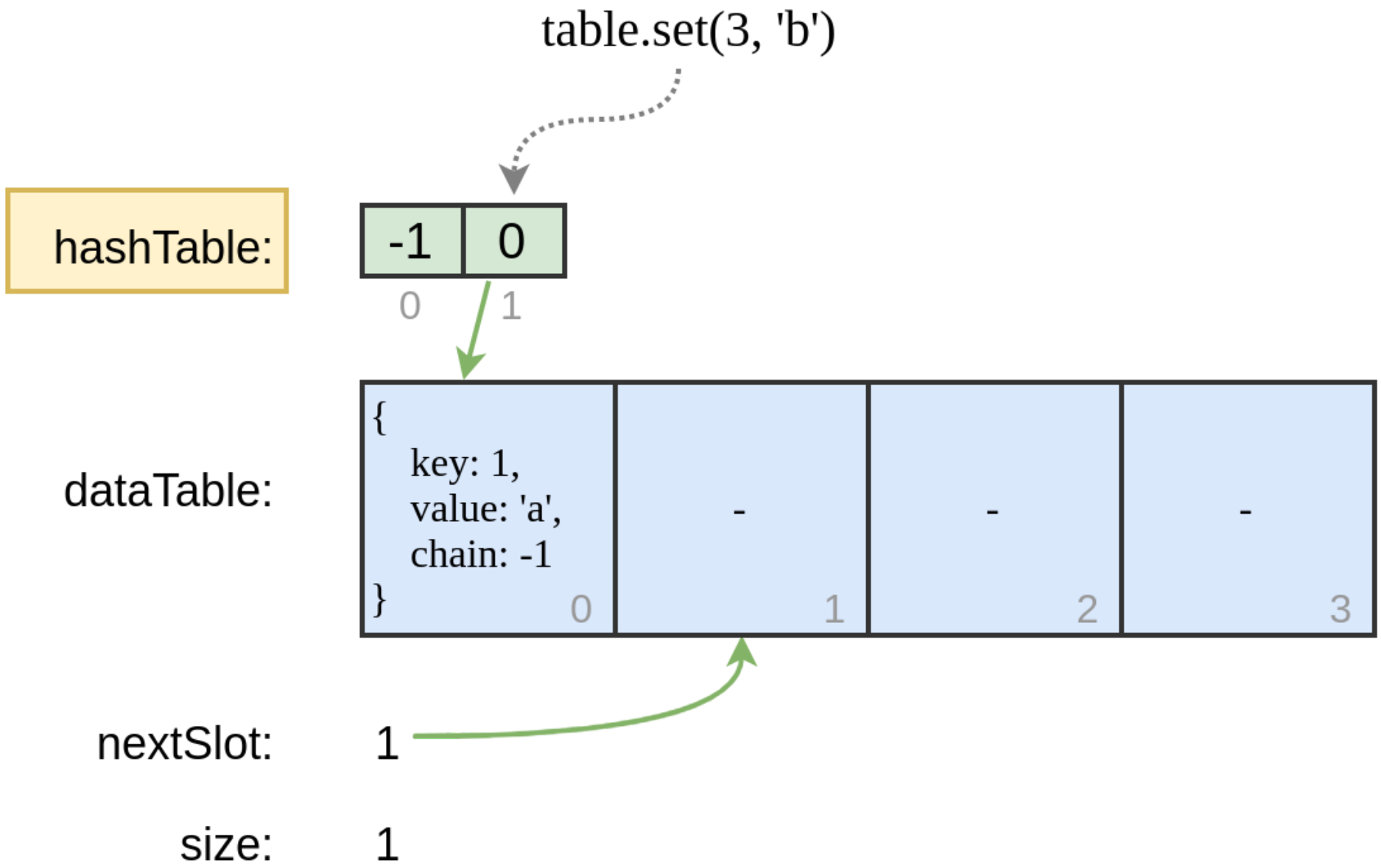
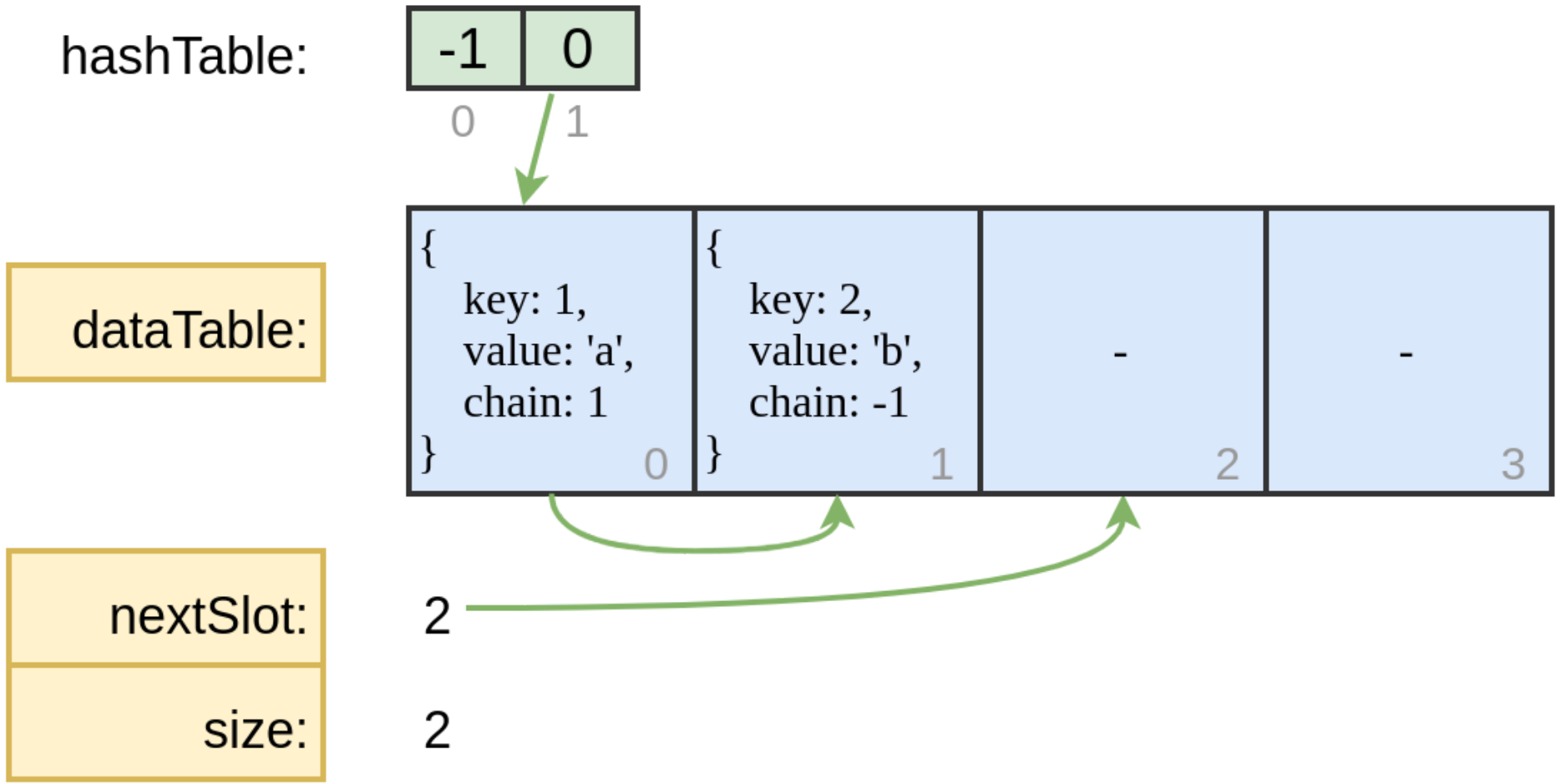


table.set(3, 'b')
hashTable:
-1
0
0
1
dataTable:
{
key: 1,
value: 'a',
chain: -1
}
0
-
1
-
2
-
3
nextSlot:
1
size:
1

table.set(3, 'b')


hashTable:
-1
0
0
1
dataTable:
{
key: 1,
value: 'a',
chain: 1
}
0
{
key: 2,
value: 'b',
chain: -1
}
1
-
2
-
3
nextSlot:
2
size:
2

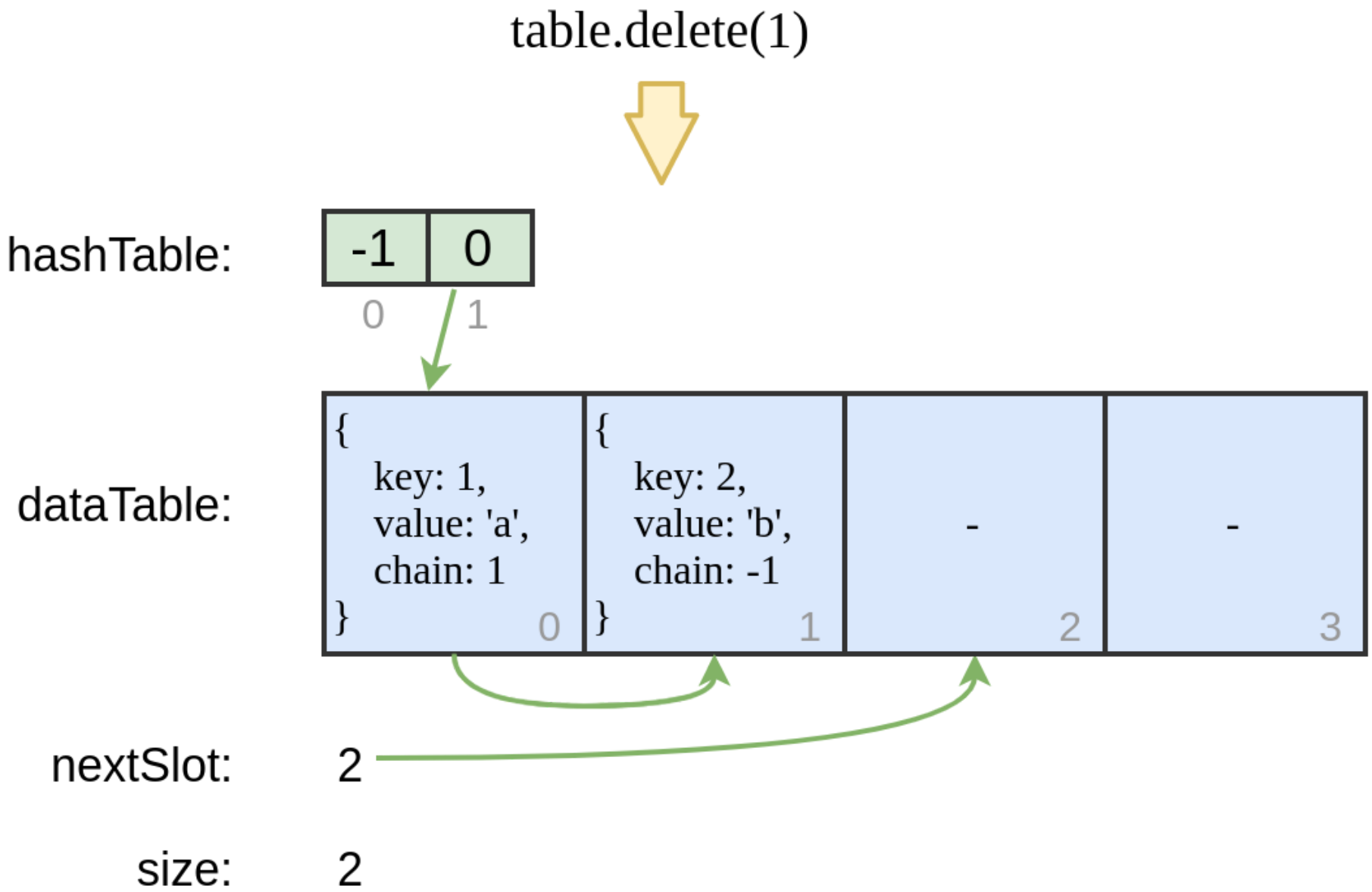


table.delete(1)
hashTable:
-1
0
0
1
dataTable:
{
key: 1,
value: 'a',
chain: 1
}
0
{
key: 2,
value: 'b',
chain: -1
}
1
-
2
-
3
nextSlot:
2
size:
2

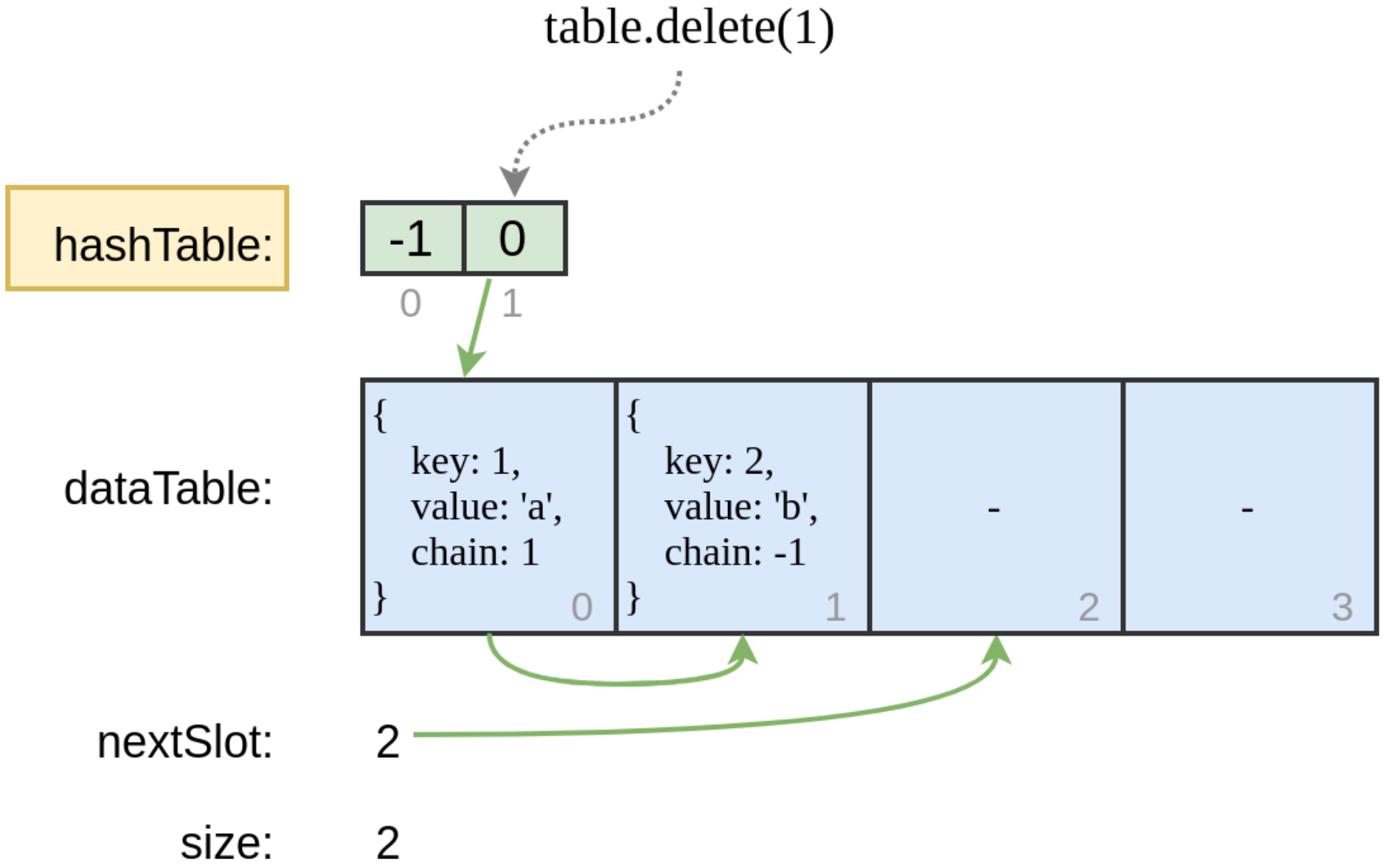


table.delete(1)
hashTable:
-1
0
0
1
dataTable:
{
key: 1,
value: 'a',
chain: 1
}
0
{
key: 2,
value: 'b',
chain: -1
}
1
-
2
-
3
nextSlot:
2
size:
2

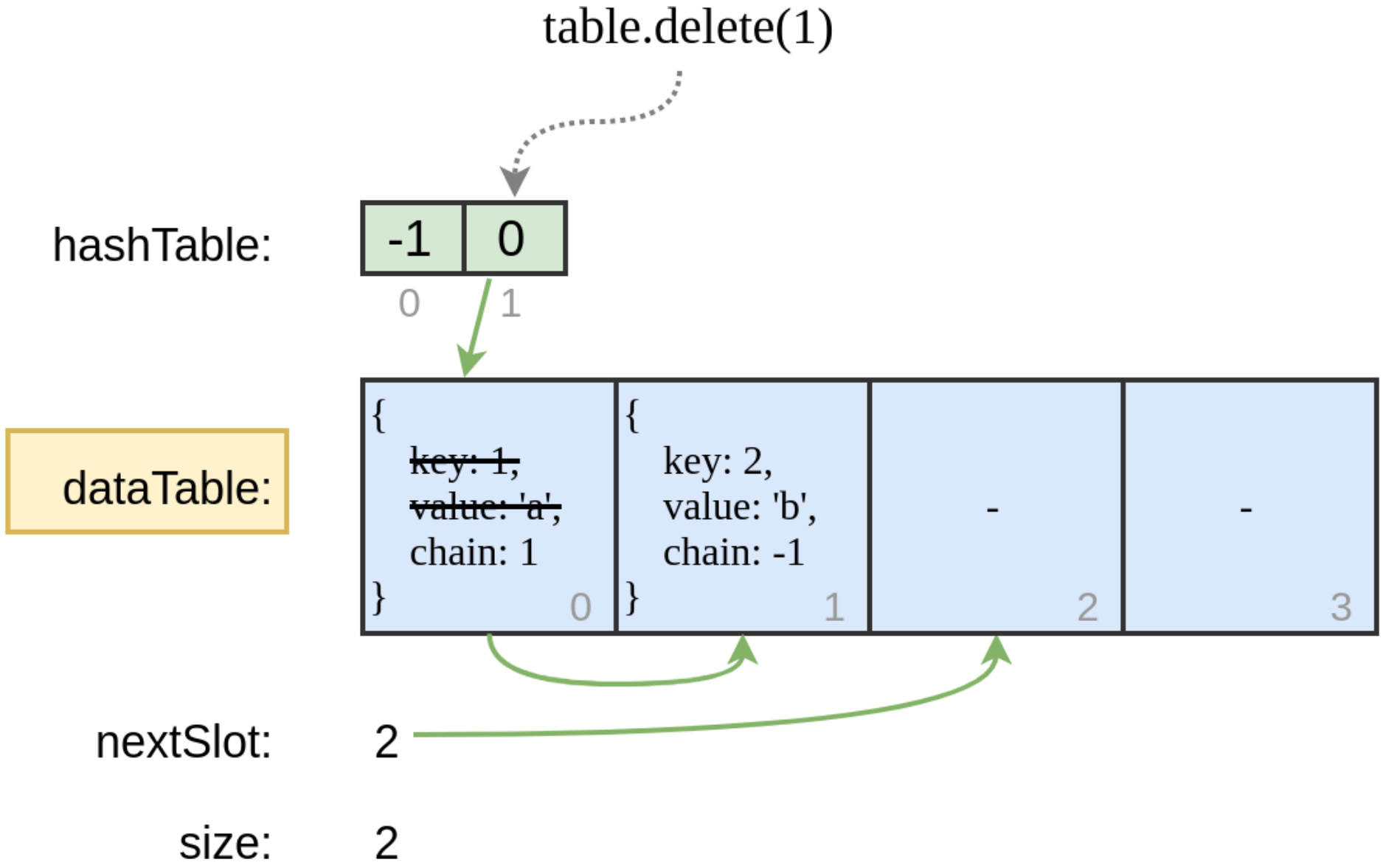


table.delete(1)
hashTable:
-1
0
0
1
dataTable:
{
key: 1,
value: 'a',
chain: 1
}
0
{
key: 2,
value: 'b',
chain: -1
}
1
-
2
-
3
nextSlot:
2
size:
2

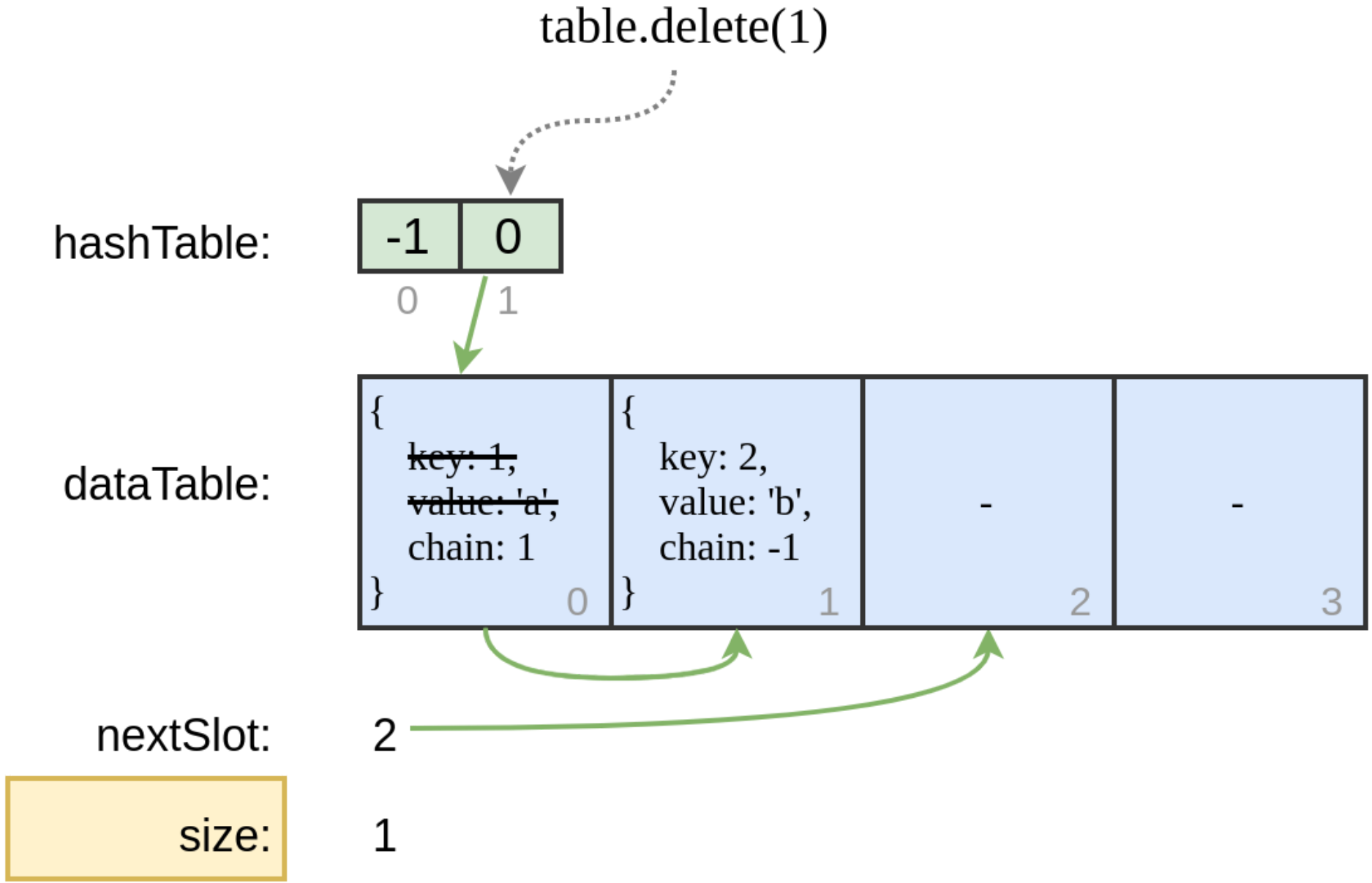
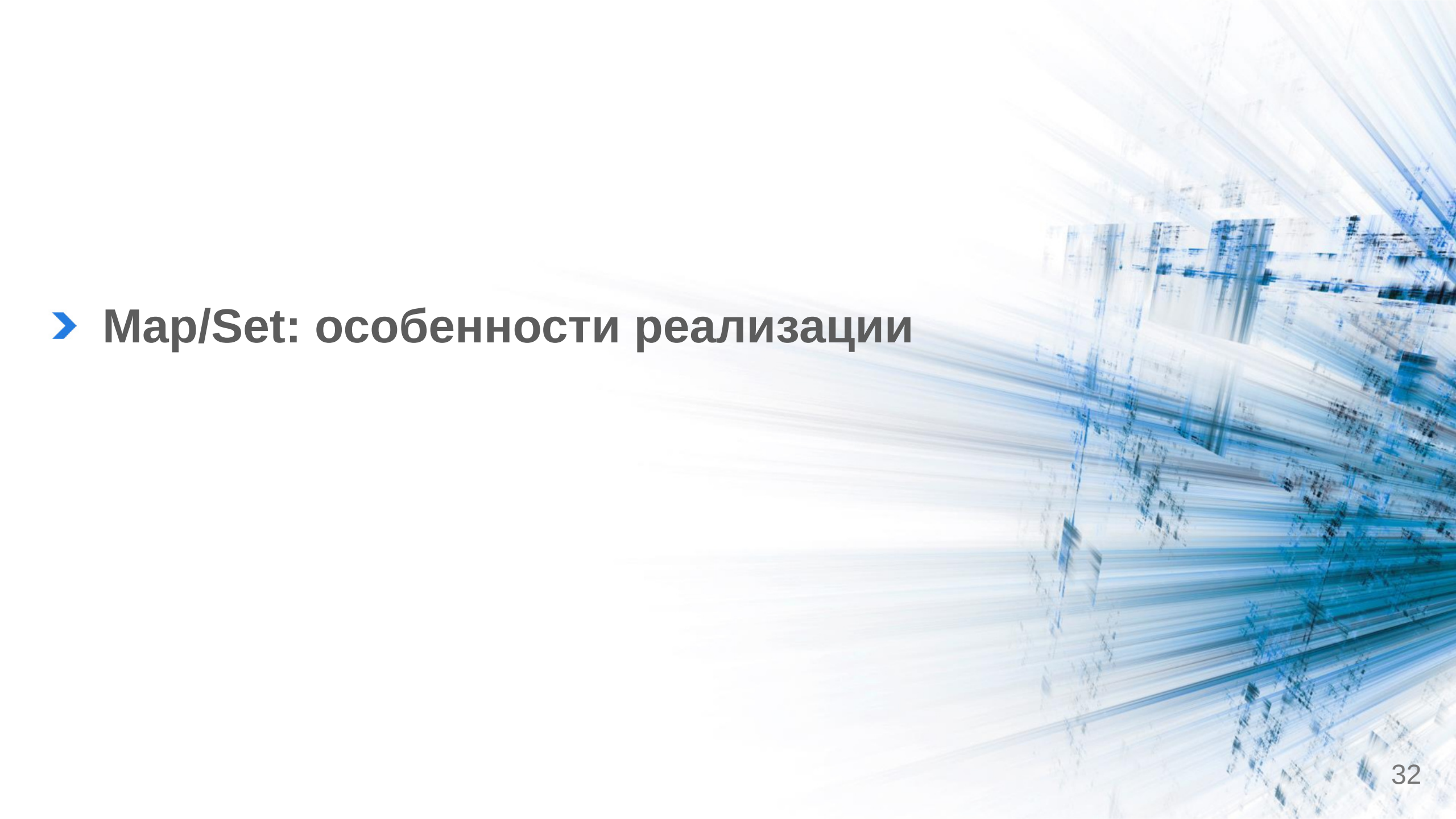


table.delete(1)
hashTable:
-1
0
0
1
dataTable:
{
key: 1,
value: 'a',
chain: 1
}
0
{
key: 2,
value: 'b',
chain: -1
}
1
-
2
-
3
nextSlot:
2
size:
1

Map/Set: особенности реализации



Основа реализации Map/Set - классы `OrderedHashTable` и `OrderedHashMap`:

- ordered-hash-table.h
- ordered-hash-table.cc
- builtins-collections-gen.cc

Емкость

- Емкость это всегда степерь двойки
- Коэффициент заполнения равен 2
  - Емкость равна 2 * кол_во_ячеек

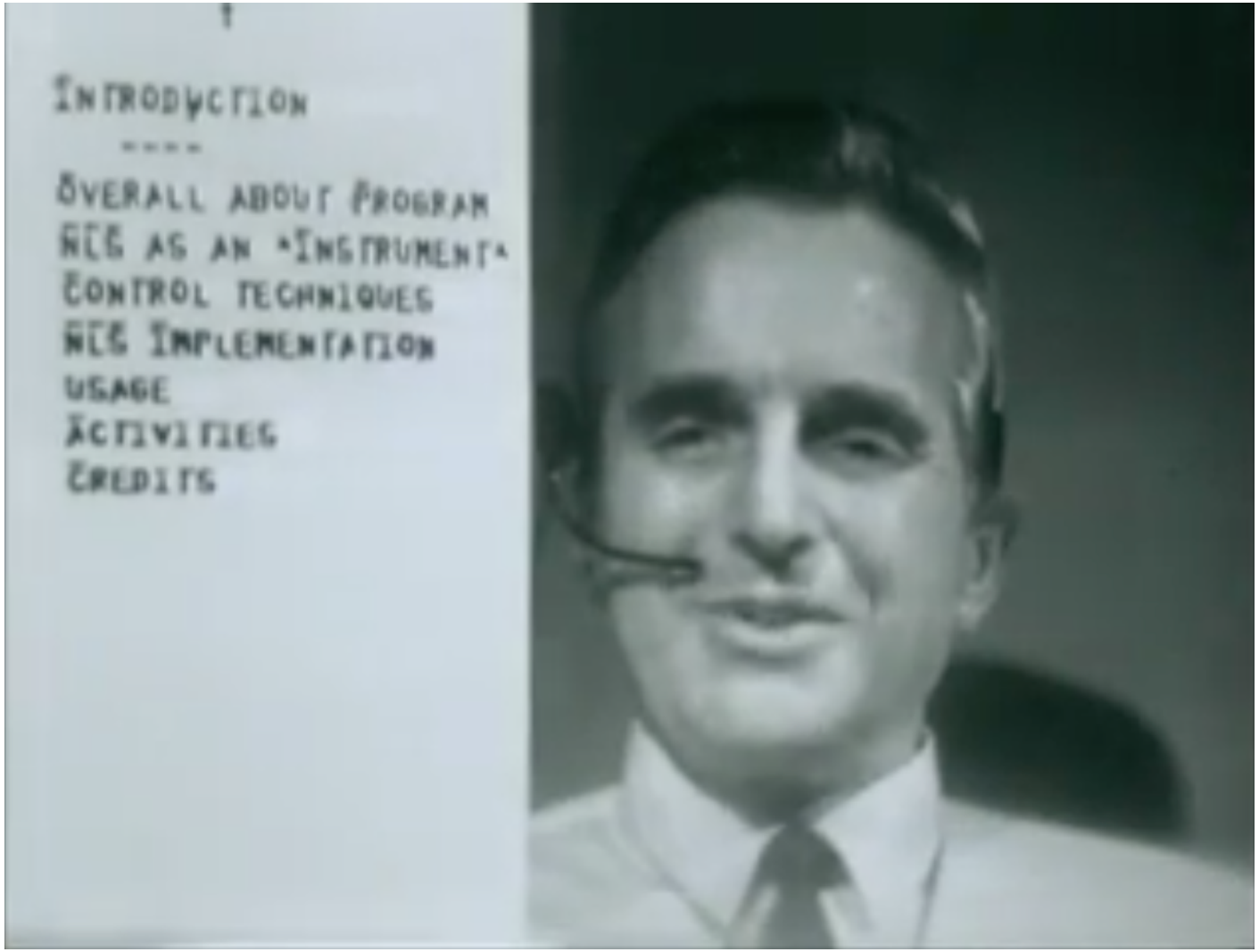
Границы

- Начальная емкость: `new Map()` содержит 2 ячейки (емкость равна 4)
- Максимальная емкость: на 64-битной системе емкость Map ограничена 2^{27} (~16.7 млн. пар)

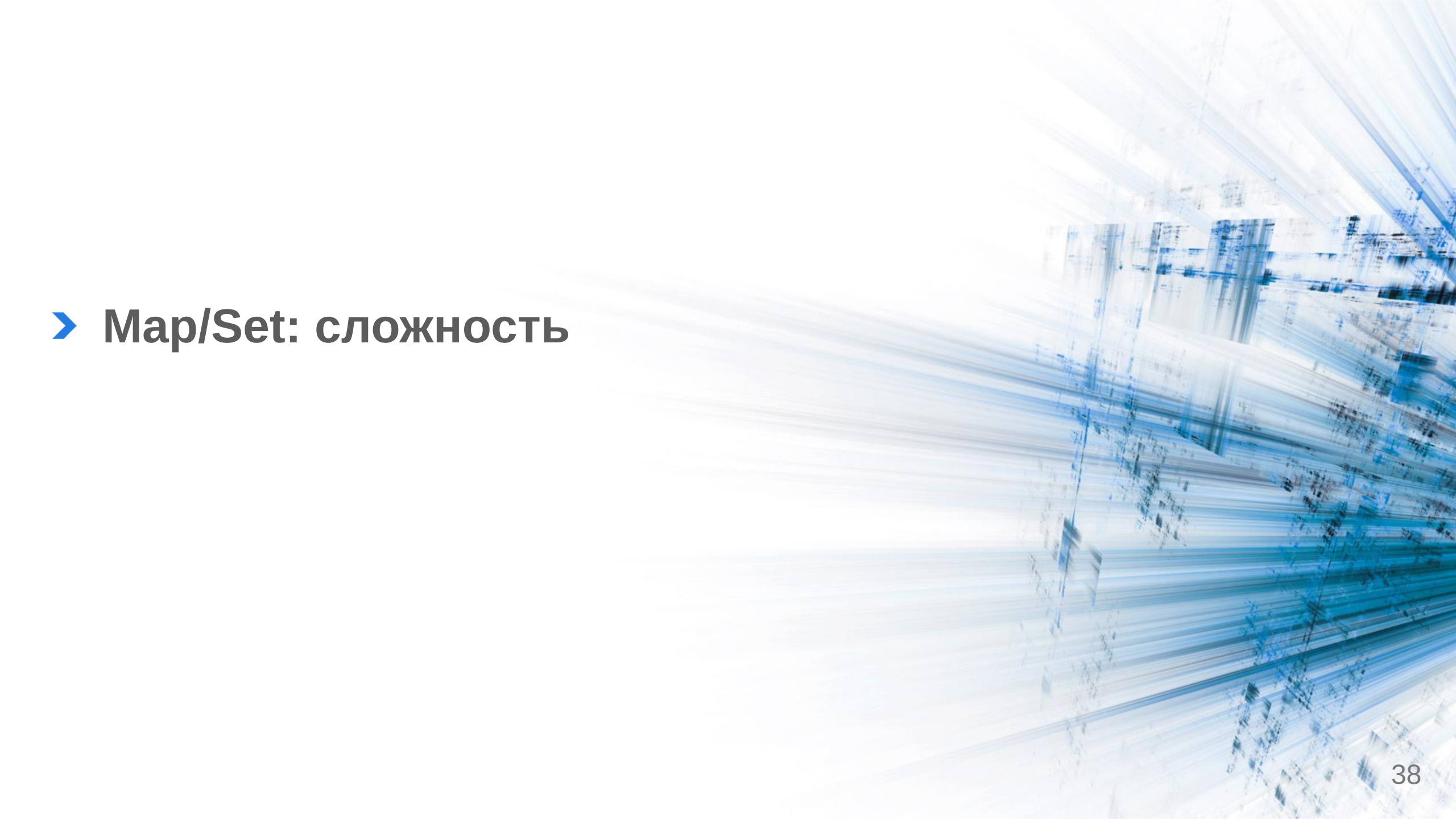
Перехеширование

- Множитель при перехешировании тоже 2
  - Таблица увеличивается/уменьшается в 2 раза

Проверим-ка!


1
INTRODUCTION
OVERALL ABOUT PROGRAM
NLS AS AN "INSTRUMENT"
CONTROL TECHNIQUES
NLS IMPLEMENTATION
USAGE
ACTIVITIES
CREDITS

- **Map/Set: сложность**



Big O

	В среднем случае	**В худшем случае**
Поиск	O(1)	O(n)
Вставка*	O(1)	O(n)
Удаление**	O(1)	O(n)

* Может привести к увеличению таблицы

** Может привести к уменьшению таблицы

Map/Set: память

Как Map/Set хранятся в памяти?

```
interface CloseTable {
    hashTable: number[];
    dataTable: Entry[];
    nextSlot: number;
    size: number;
}
```

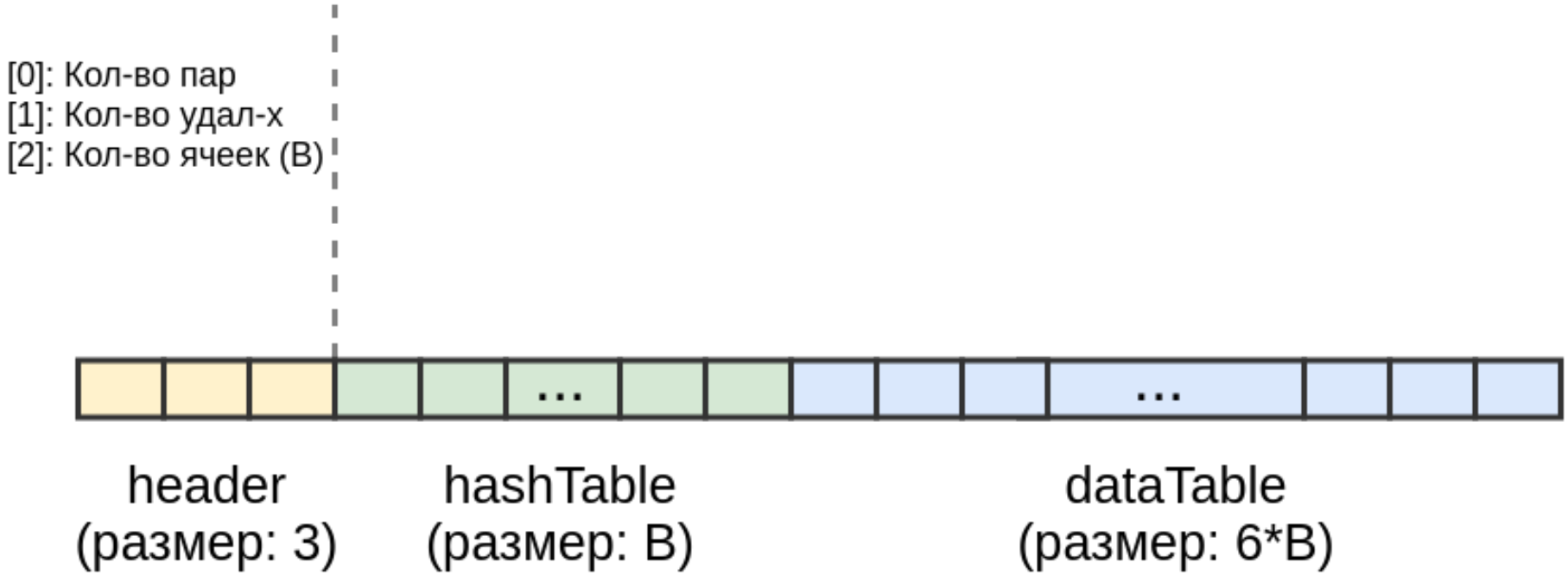
С точки зрения хранения данных в куче V8 Map/Set это всего лишь массивы

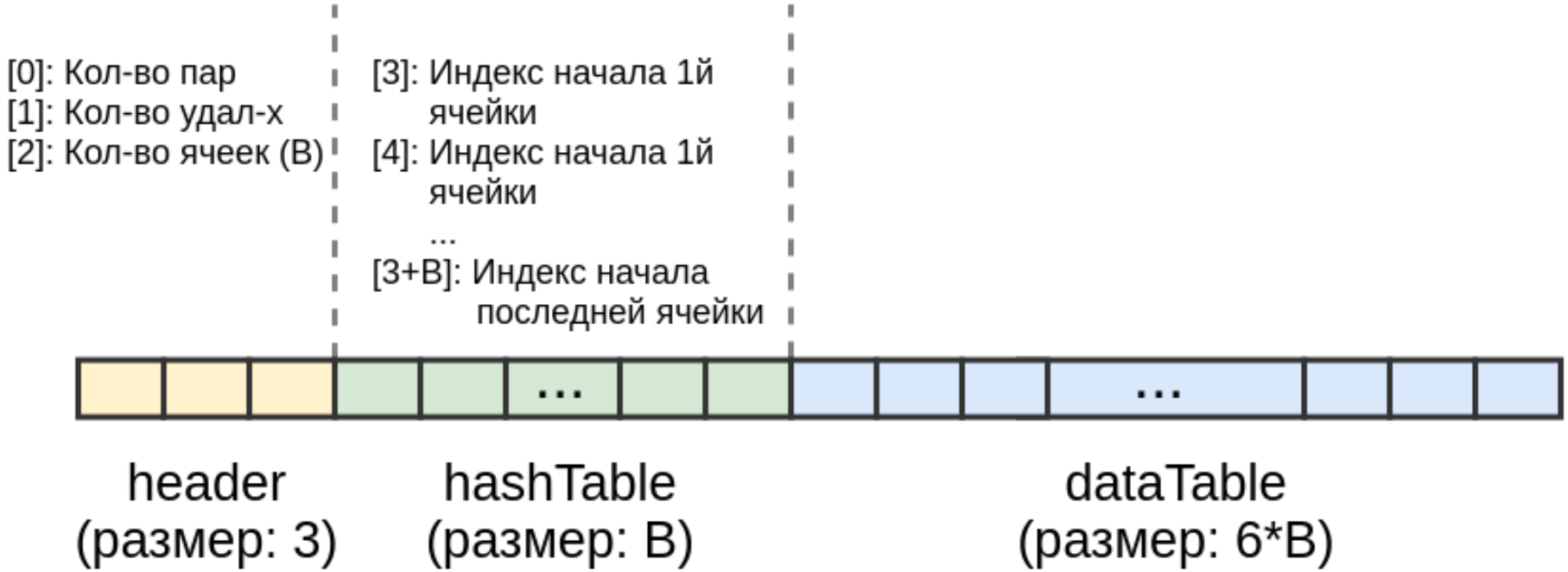
P.S. Отсюда следуют упомянутые ограничения на размеры

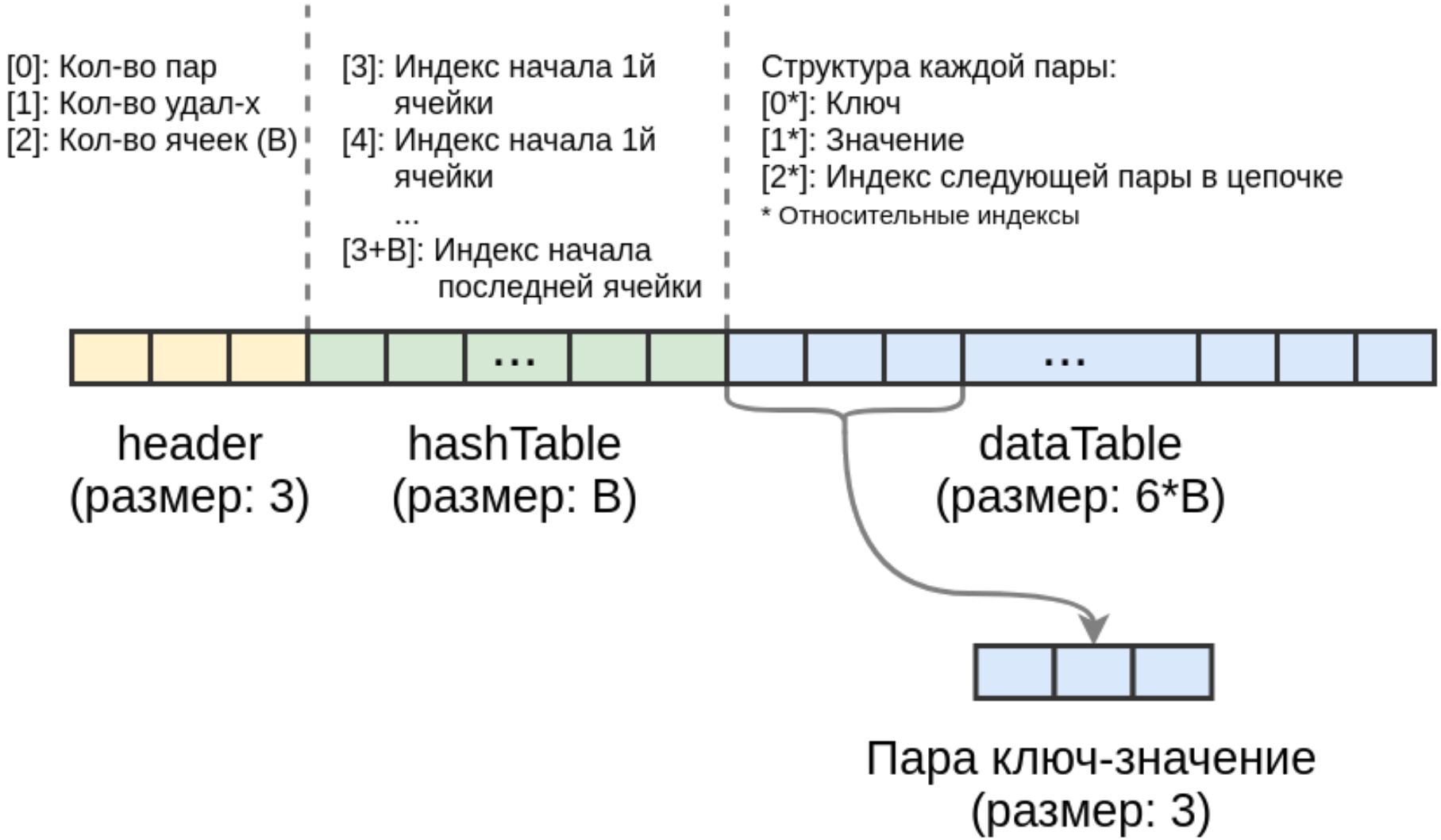
header
(размер: 3)

hashTable
(размер: B)

dataTable
(размер: 6*B)


[0]: Кол-во пар
[1]: Кол-во удал-х
[2]: Кол-во ячеек (B)
...
...
header
(размер: 3)
hashTable
(размер: B)
dataTable
(размер: 6*B)


[0]: Кол-во пар
[1]: Кол-во удал-х
[2]: Кол-во ячеек (B)
[3]: Индекс начала 1й ячейки
[4]: Индекс начала 1й ячейки
...
[3+B]: Индекс начала последней ячейки
...
...
header (размер: 3)
hashTable (размер: B)
dataTable (размер: 6*B)


[0]: Кол-во пар
[1]: Кол-во удал-х
[2]: Кол-во ячеек (B)
[3]: Индекс начала 1й ячейки
[4]: Индекс начала 1й ячейки
...
[3+B]: Индекс начала последней ячейки
Структура каждой пары:
[0*]: Ключ
[1*]: Значение
[2*]: Индекс следующей пары в цепочке
* Относительные индексы
header (размер: 3)
hashTable (размер: B)
dataTable (размер: 6*B)
Пара ключ-значение (размер: 3)

Немного арифметики

Размер массива можно оценить примерно как:

- `N*3.5`, где `N` - емкость Map
- `N*2.5`, где `N` - емкость Set

Входные условия

Для 64-битной системы (без учета pointer compression) каждый элемент массива занимает 8 байтов

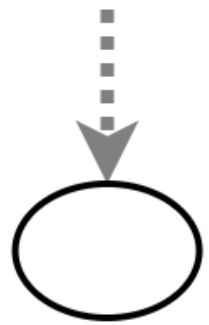
Итого

- Map с 2^{20} (~1 млн.) пар займет ~29MB памяти
- Set с 2^{20} (~1 млн.) элементов займет ~21MB памяти

WeakMap/WeakSet: алгоритм

Не подлежит GC

Подлежит GC



strong reference

weak reference

Почему бы не взять Map + WeakRef* в основу WeakMap?

* Речь о ключах, но для значений тоже есть нюансы

- **WeakMap/WeakSet: особенности реализации**

- WeakMap/WeakSet:
  - EphemeronHashTable
  - MarkingVisitorBase::VisitEphemeronHashTable
  - MarkCompactCollector::ProcessEphemeronsUntilFixpoint, ProcessEphemeronsLinear, ProcessEphemeron
- WeakRef:
  - MarkingVisitorBase::VisitJSWeakRef
  - MarkCompactCollector::ClearJSWeakRefs

Что же такое WeakMap/WeakSet?

Формально - "классическая" хеш-таблица с открытой адресацией и квадратичным пробированием

TODO

Спасибо за внимание!

Полезные ссылки

- https://itnext.io/v8-deep-dives-understanding-map-internals-45eb94a183df
- http://www.jucs.org/jucs_14_21/eliminating_cycles_in_weak/jucs_14_21_3481_3497_barros.pdf
- https://v8.dev/blog/concurrent-marking

- Bonus unlocked 🏅🏅🏅

Map/Set: Map vs Object

	Object	Map
Заранее известная структура	✓	
Гарантированный порядок обхода		✓*
Ключи произвольного типа		✓
Предсказуемая производительность при частых вставках/удалениях		✓

* С недавних пор (ES 2015, 2020) у объектов порядок обхода тоже гарантирован, но правила обхода сложнее