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

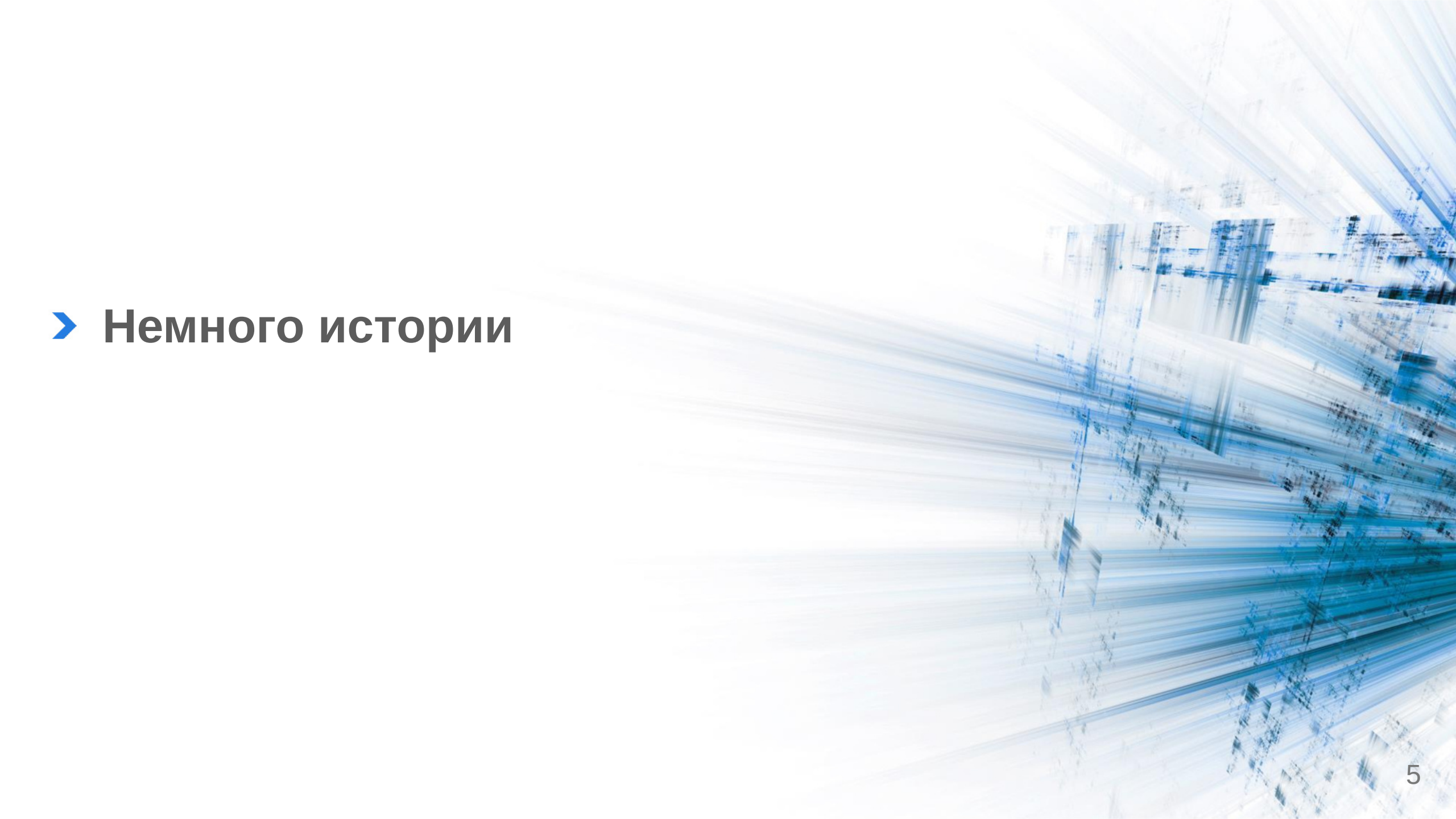
Disclamer

- 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

- Немного истории
- Map/Set
  - Алгоритм
  - Особенности реализации
  - Сложность
  - Память
  - Map vs Object
- WeakMap/WeakSet
  - TODO

Немного истории



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

Хеш-таблица

TODO

Упомянуть load factor

Хеш-функция

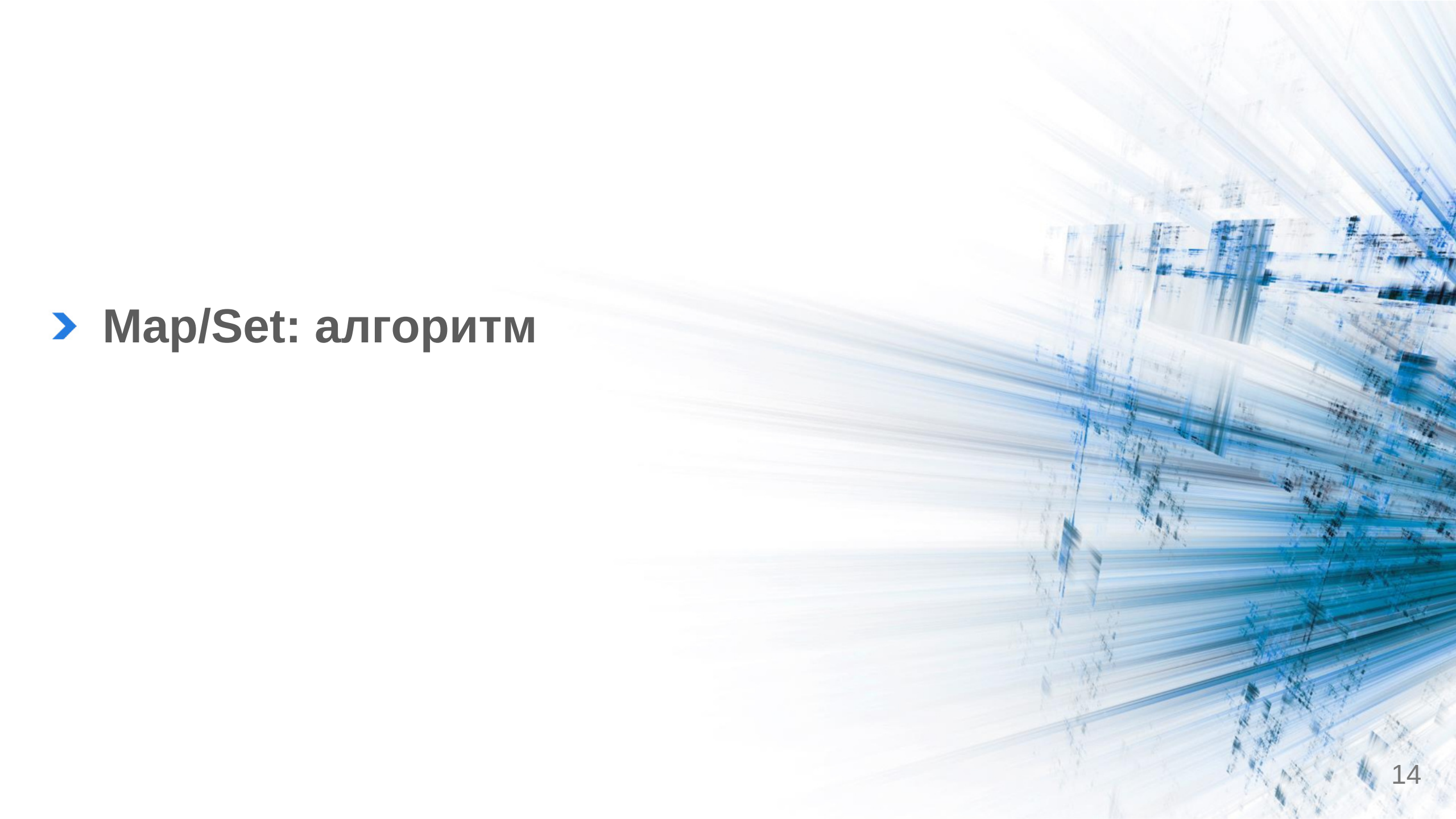
TODO

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**

Map/Set: алгоритм



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;
}
```

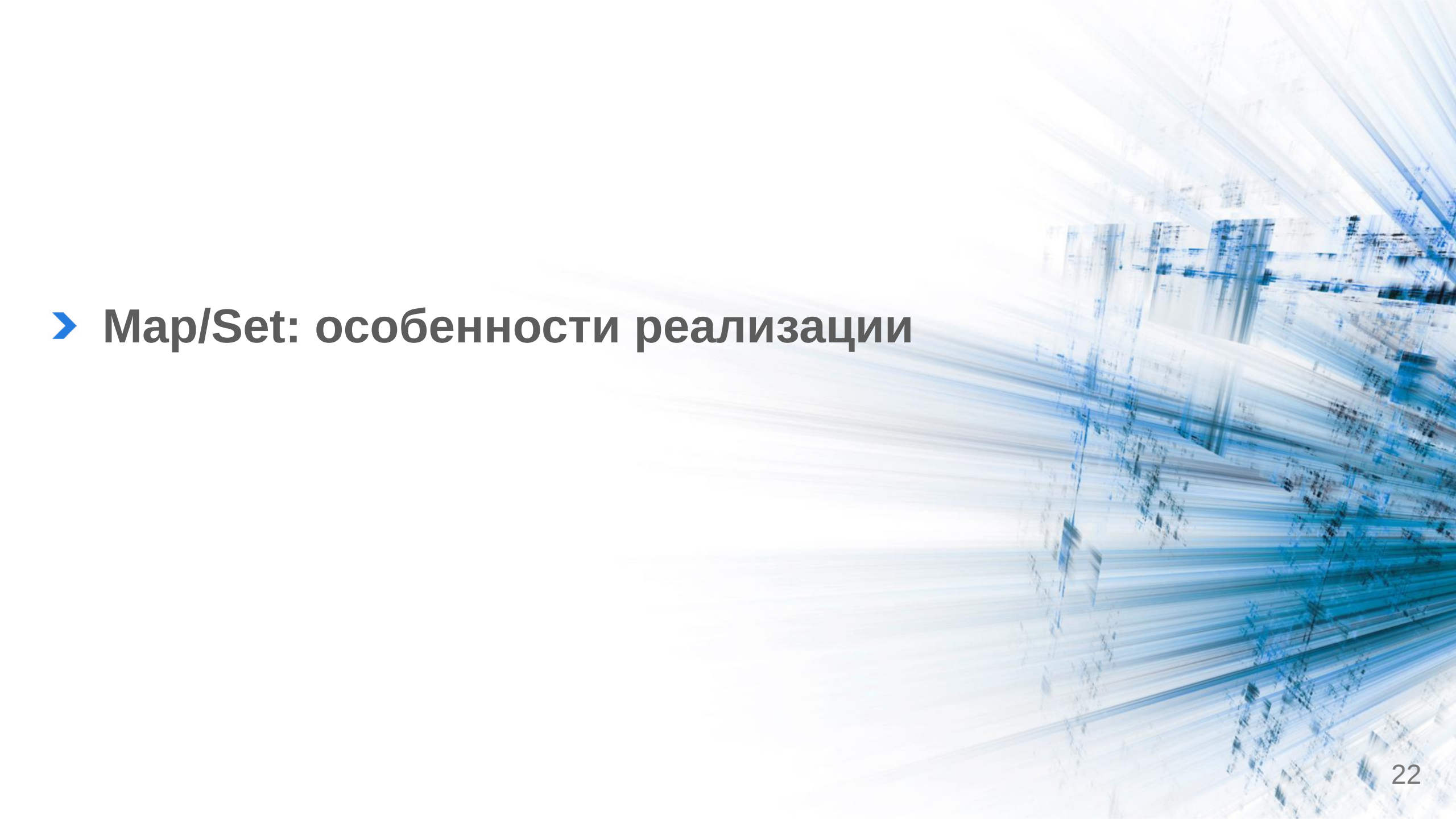
```
// пусть хеш-функция:
// function hashCode(n) { return n; }
table.set(0, 'a'); // => bucket 0 (0 % 2)
table.set(1, 'b'); // => bucket 1 (1 % 2)
table.set(2, 'c'); // => bucket 0 (2 % 2)
```

```
const tableInternals = {
    hashTable: [0, 1],
    dataTable: [
        {
            key: 0,
            value: 'a',
            chain: 2 // индекс <2, 'c'>
        },
        {
            key: 1,
            value: 'b',
            chain: -1 // -1 значит хвост
        },
        {
            key: 2,
            value: 'c',
            chain: -1
        },
        // пустой слот
    ],
    nextSlot: 3, // индекс пустого слота
    size: 3
}
```

```
// table.delete(0)
const tableInternals = {
    hashTable: [0, 1],
    dataTable: [
        {
            key: undefined, // удаленный элемент
            value: undefined,
            chain: 2
        },
        {
            key: 1,
            value: 'b',
            chain: -1
        },
        {
            key: 2,
            value: 'c',
            chain: -1
        },
        // пустой слот
    ],
    nextSlot: 3,
    size: 2 // новый размер
}
```

TODO нарисовать картинки и описать перехеширование

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
- Load factor равен 2 => емкость === 2 * кол_во_бакетов

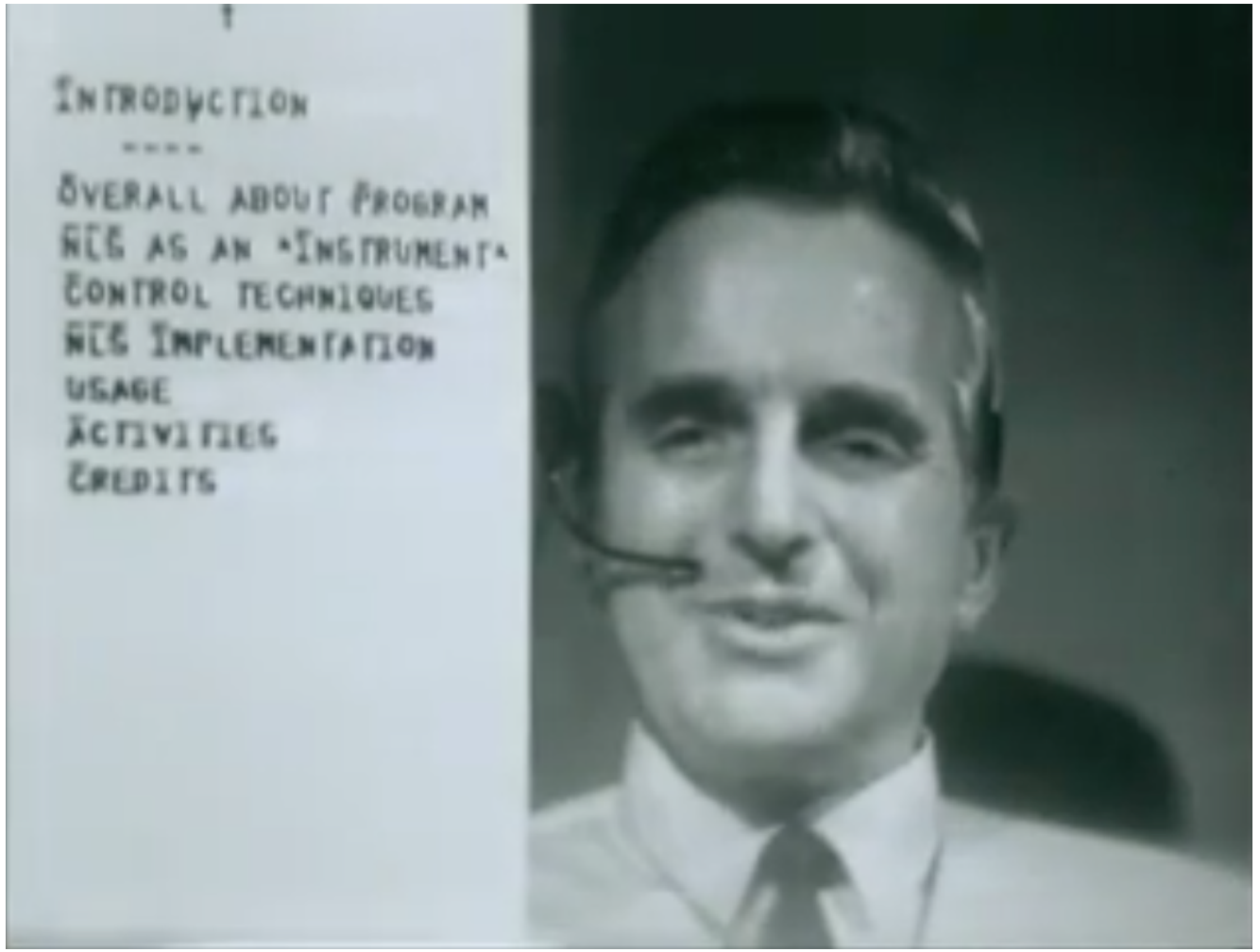
Границы

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

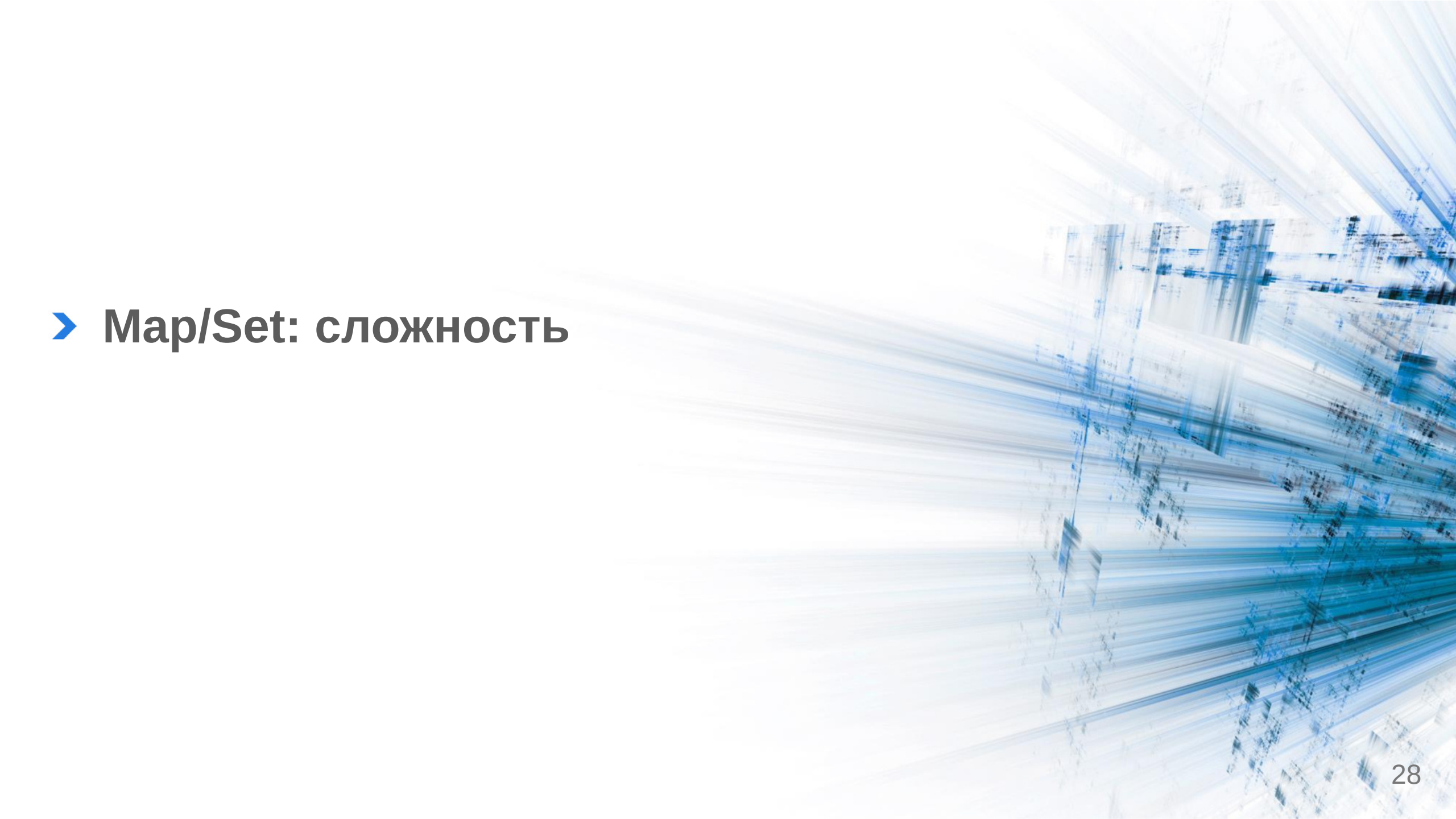
Перехеширование

- Множитель при перехешировании тоже 2

Проверим-ка!


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

- **Map/Set: сложность**



TODO

Перечислить в таблице (упомянуть перехеширование)

Map/Set: память

локализовать картинку

https://miro.medium.com/max/2880/1*lrmWw6cULyCZ6sRmXnsZyQ.png

Немного арифметики

Размер массива можно оценить как `N*3.5`, где `N` - емкость Map

Еще немного арифметики

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

Map/Set: Map vs Object

TODO

TODO WeakMap/WeakSet

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