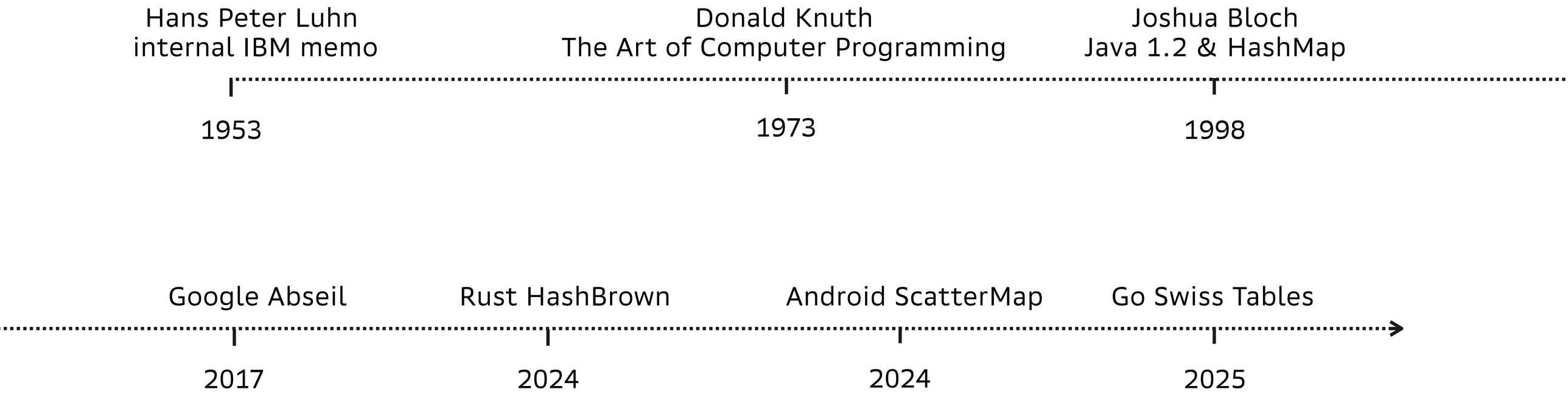
1 byte вместо 32
анатомия ScatterMap

Митропольский Александр / СберИнвестиции Android Platform

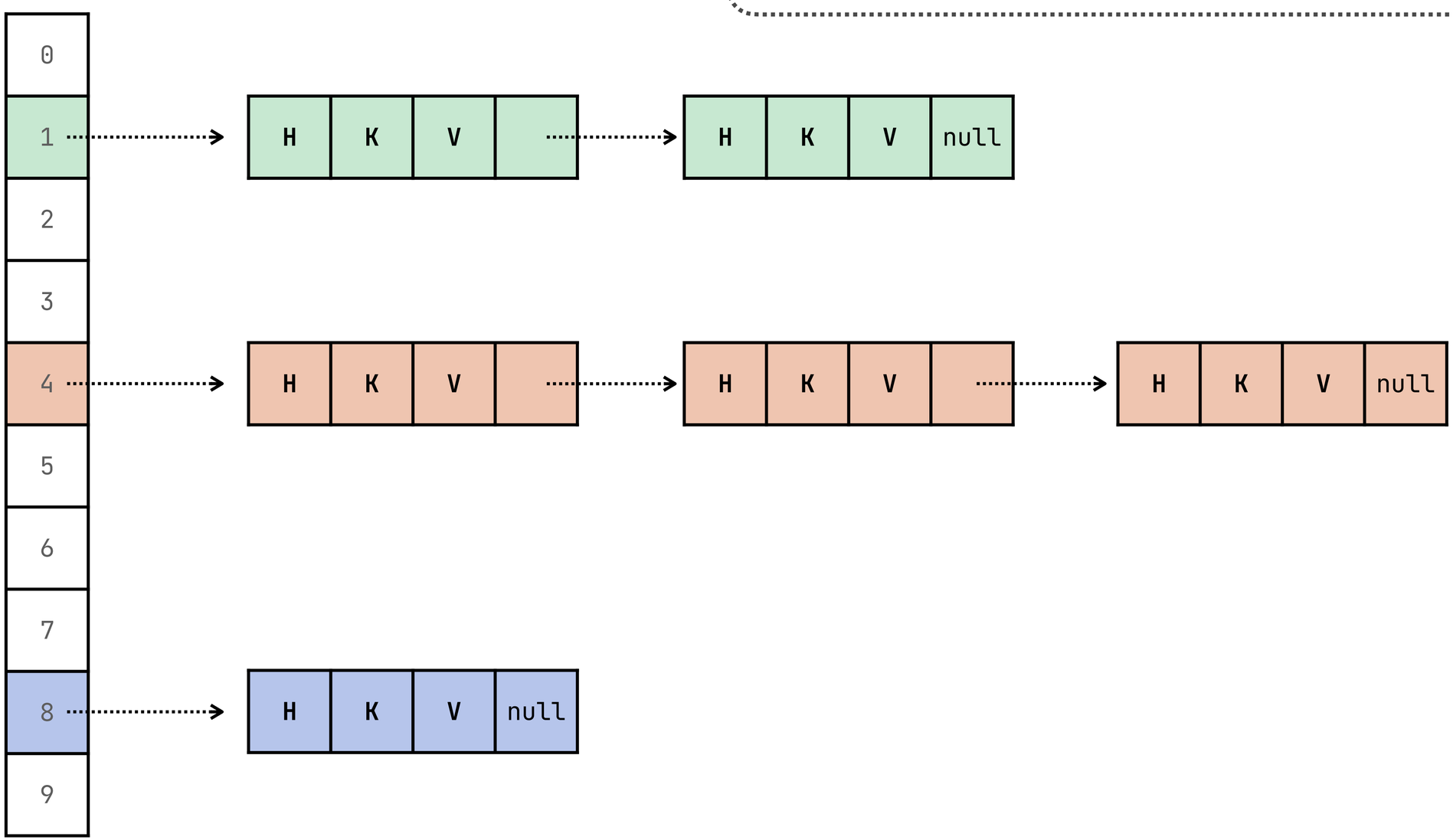

Hans Peter Luhn
internal IBM memo
1953
Donald Knuth
The Art of Computer Programming
1973
Joshua Bloch
Java 1.2 & HashMap
1998
Google Abseil
2017
Rust HashBrown
2024
Android ScatterMap
2024
Go Swiss Tables
2025

Что будет в докладе

- наносекундные оптимизации hot path
- аллокации и нагрузка на GC
- паттерны доступа к памяти и кеши процессора

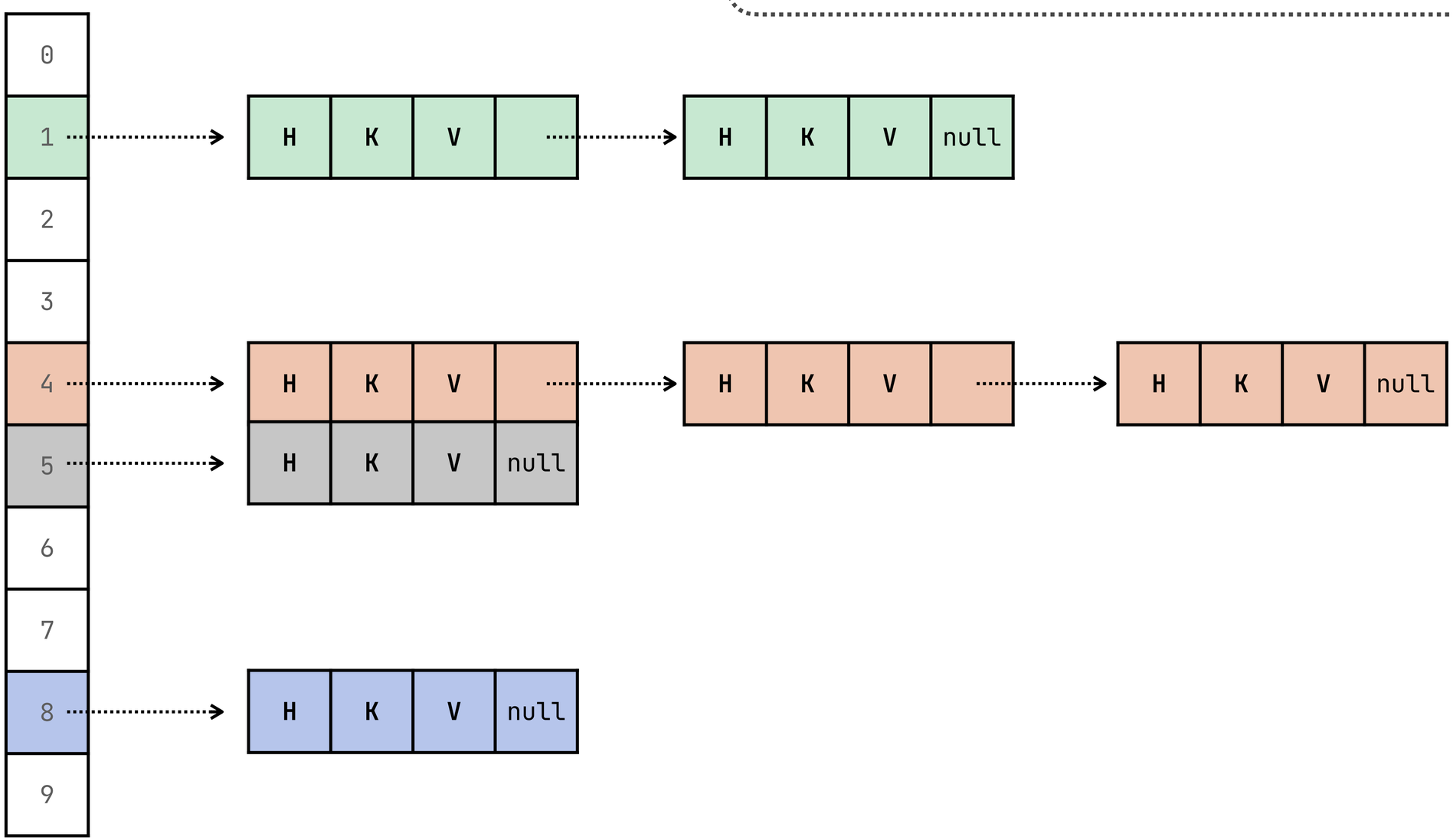
HashMap

H - hash: Int, K - key, V - value



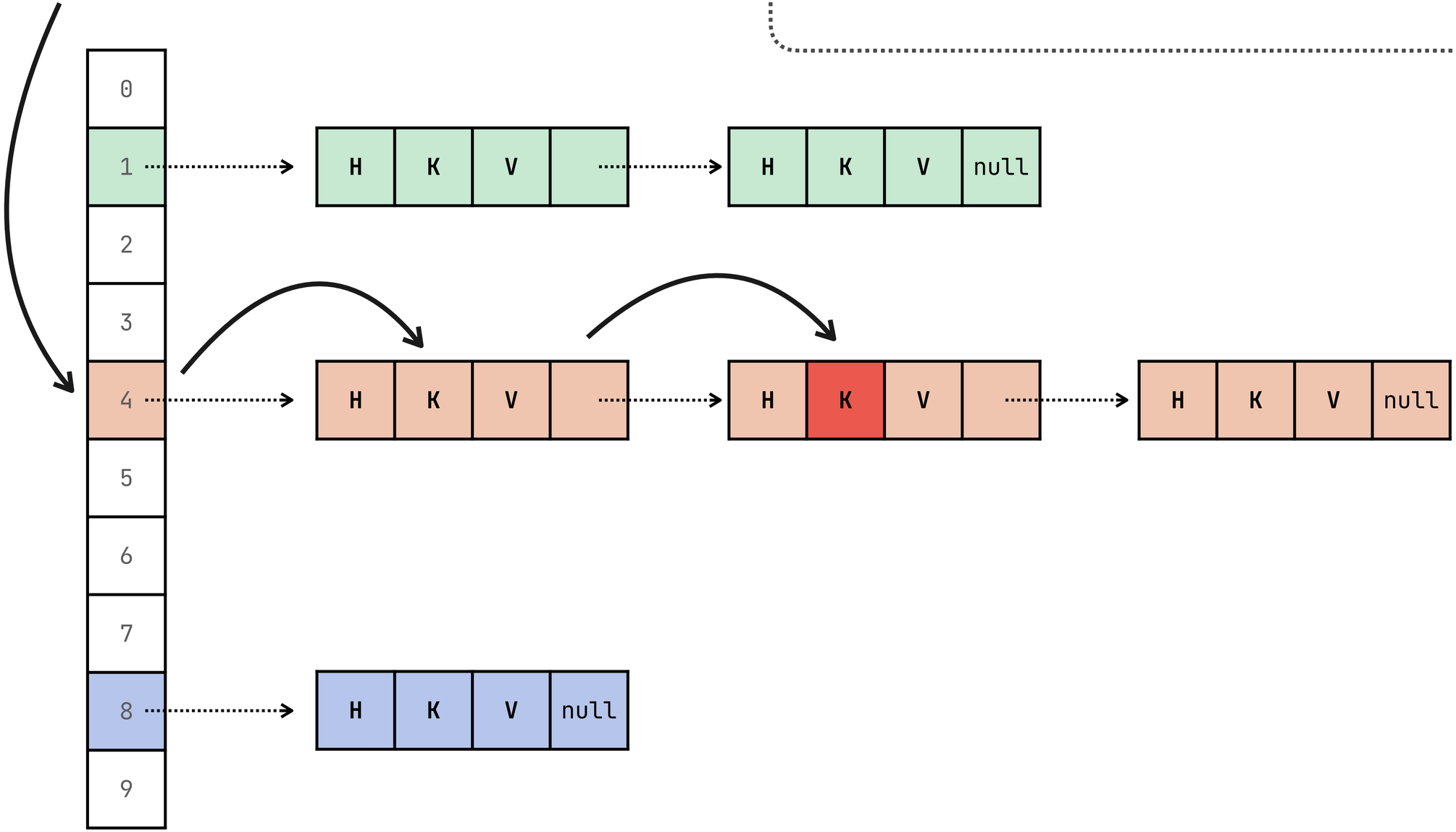
HashMap.put(K | V)

i = K .hashCode() % capacity = ..5 % 10 = **5**



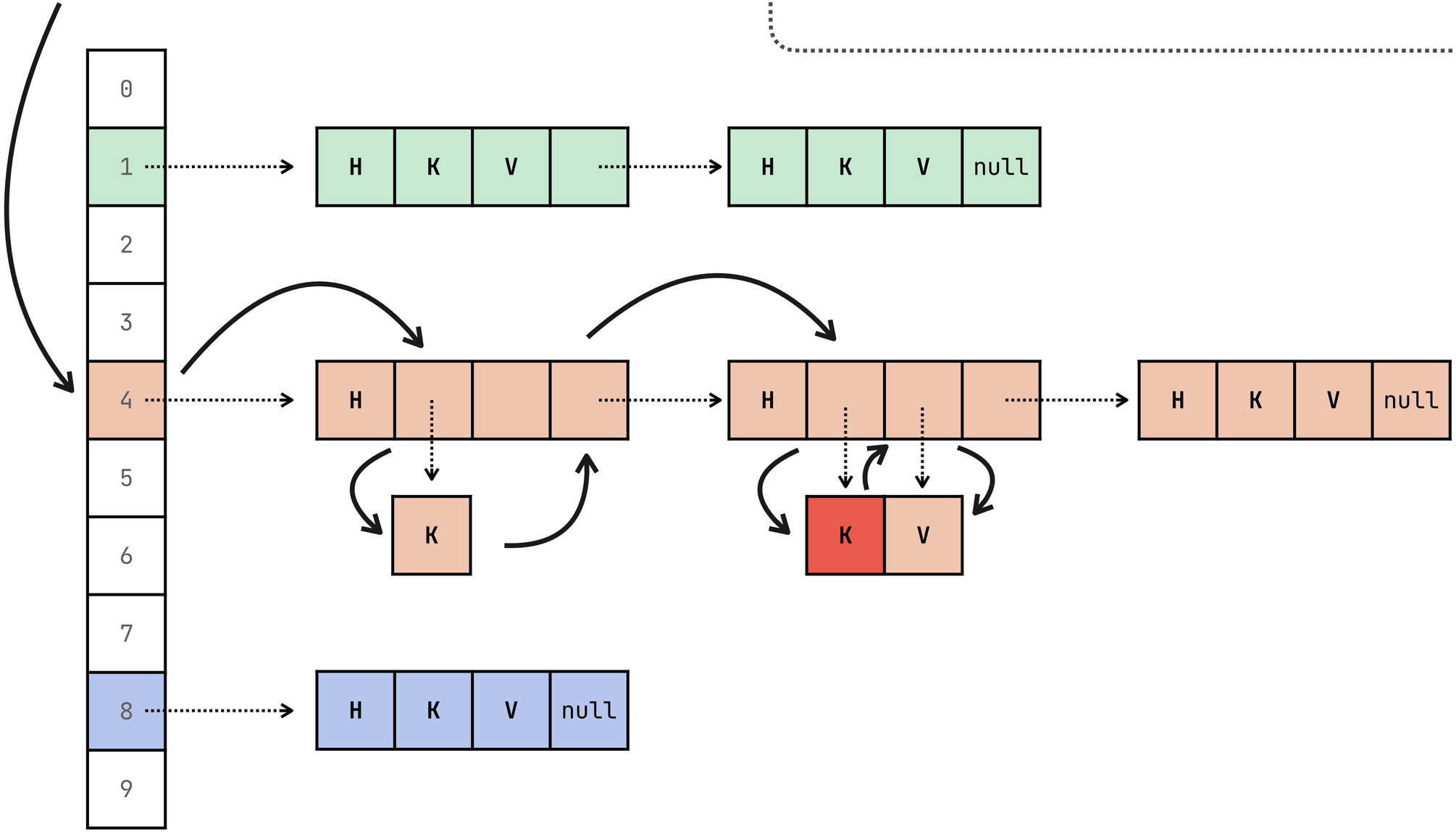
HashMap.get(K)

i = K .hashCode() % capacity = ..4 % 10 = 4

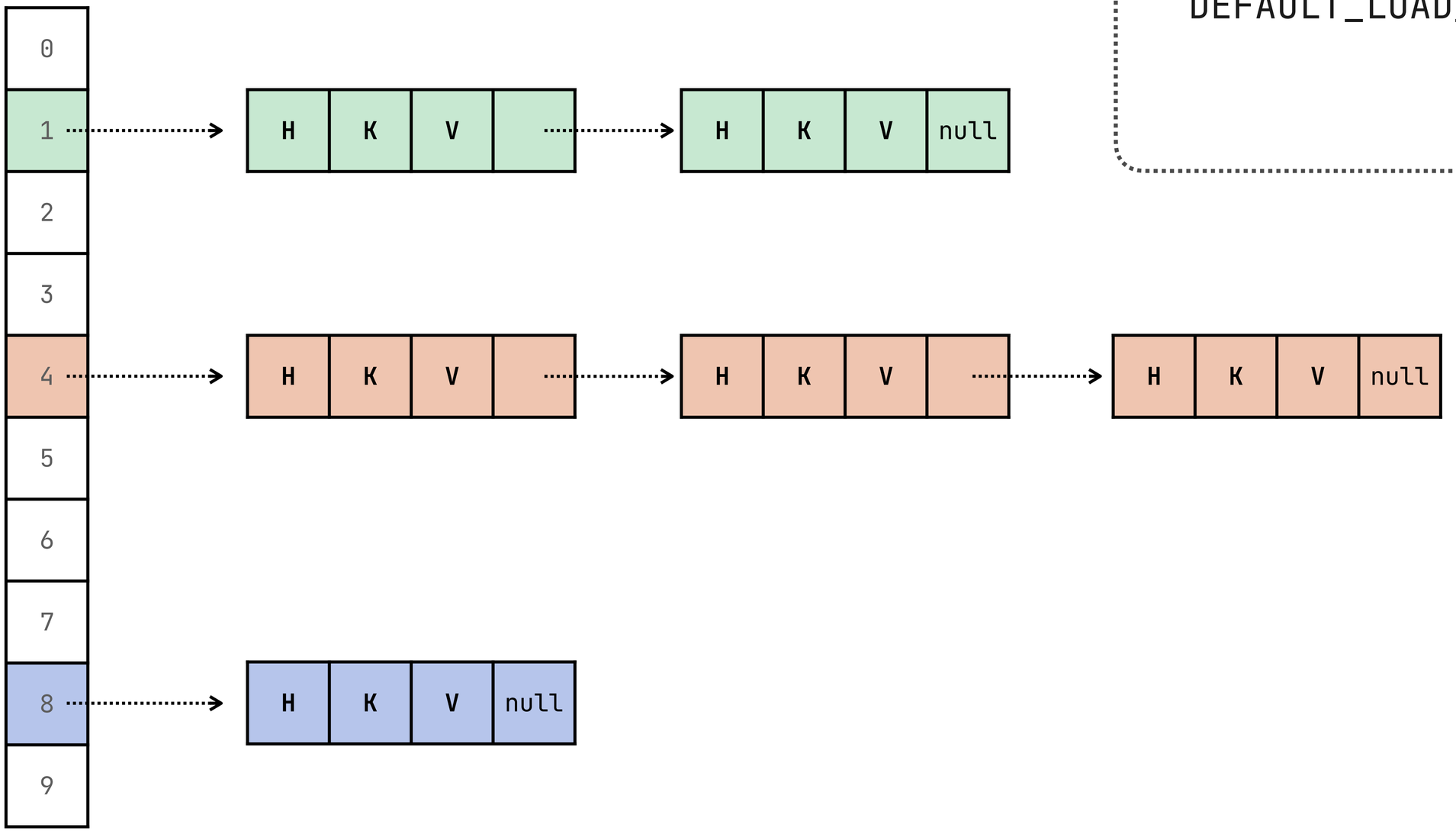


HashMap.get(K)

i = K .hashCode() % capacity = ..4 % 10 = 4



HashMap




```
loadFactor = size / capacity
DEFAULT_LOAD_FACTOR = 0.75
```

HashMap.Node

```
static class Node<K,V> {
    final int hash;
    final K key;
    V value;
    Node<K,V> next;
}
```

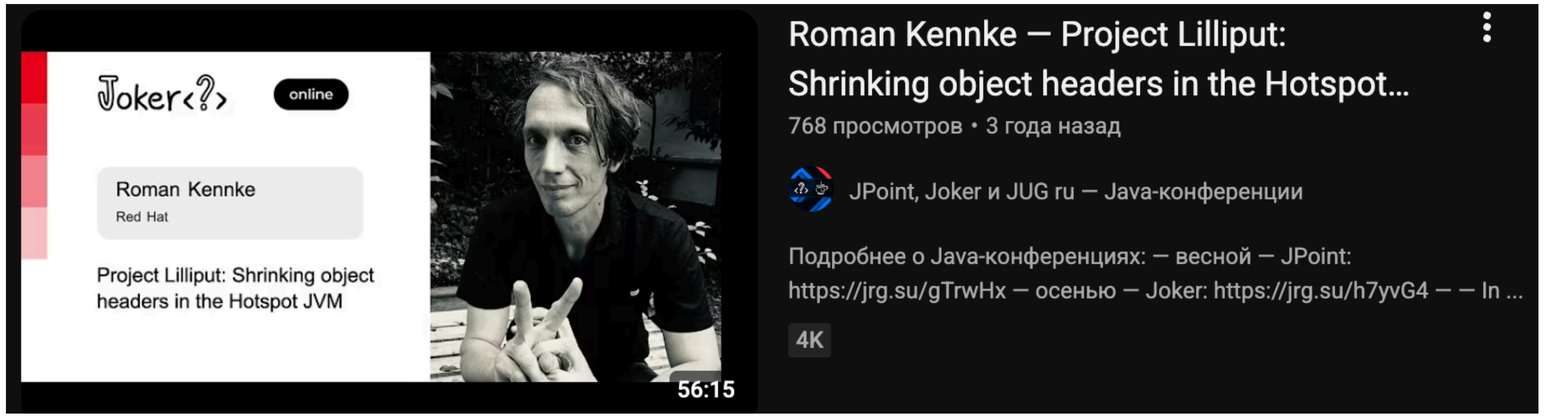
HashMap.Node

```
static class Node<K,V> {     16 bytes или 12
    final int hash;
    final K key;
    V value;
    Node<K,V> next;
}
```

HashMap.Node

```
static class Node<K,V> {
    final int hash;
    final K key;
    V value;
    Node<K,V> next;
}
```

16 bytes или 12 или 8 (в HotSpot)



HashMap.Node

```
static class Node<K,V> {     8 bytes (в ART)
    final int hash;        + 4 bytes
    final K key;           + 4 bytes
    V value;               + 4 bytes
    Node<K,V> next;        + 4 bytes
}                           ----------
                            24 bytes
```

```
inline fun <K, V> mutableMapOf(): MutableMap<K, V> = ?
```

LinkedHashMap.Node

```
static class Node<K,V> {     8 bytes (в ART)
    final int hash;        + 4 bytes
    final K key;           + 4 bytes
    V value;               + 4 bytes
    Node<K,V> next;        + 4 bytes
    Node<K,V> before;      + 4 bytes
    Node<K,V> after;       + 4 bytes
}                          ---------
                           32 bytes
```

HashMap.Node<Int, Int>

```
static class Node<Integer, Integer> {
    final int hash;
    final Integer key;
    Integer value;
    Node<Integer, Integer> next;
}
```

```
  8 bytes (в ART)
+ 4 bytes
+ 4 + 16 - 4 = 16 bytes
+ 4 + 16 - 4 = 16 bytes
+ 4 bytes
-------------------
 56 bytes
```

HashMap.Node<Int, Int>

```
static class Node<Integer, Integer> {
    final int hash;
    final Integer key;
    Integer value;
    Node<Integer, Integer> next;
}
```

```
  8 bytes (в ART)
+ 4 bytes
+ 4 + 16 - 4 = 16 bytes
+ 4 + 16 - 4 = 16 bytes
+ 4 bytes
 ----------------------
 56 bytes
+ 4 × 1 / 0.75 ≈ 5.3
 ----------------------
 61 bytes
```

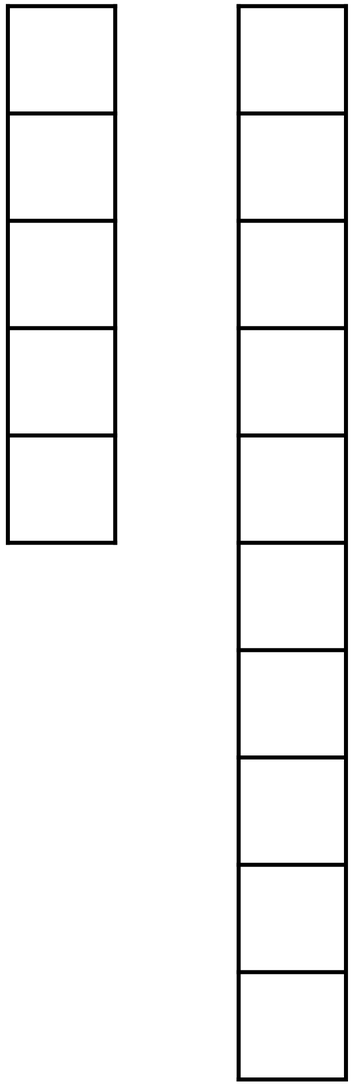
HashMap

- O(1)

- Каждая пара - отдельный Node (overhead 24 или 32 байта)
- Autoboxing примитивов
- Pointer chasing и риск cache miss

ArrayMap

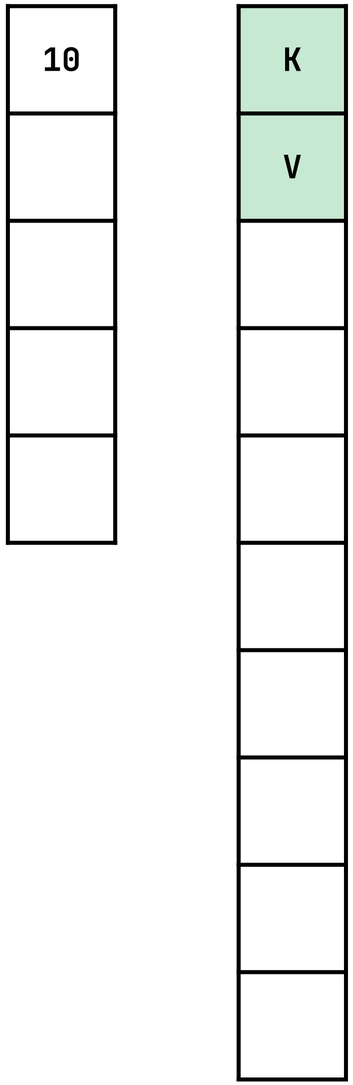
ArrayMap



H - hash: Int, K - key, V - value

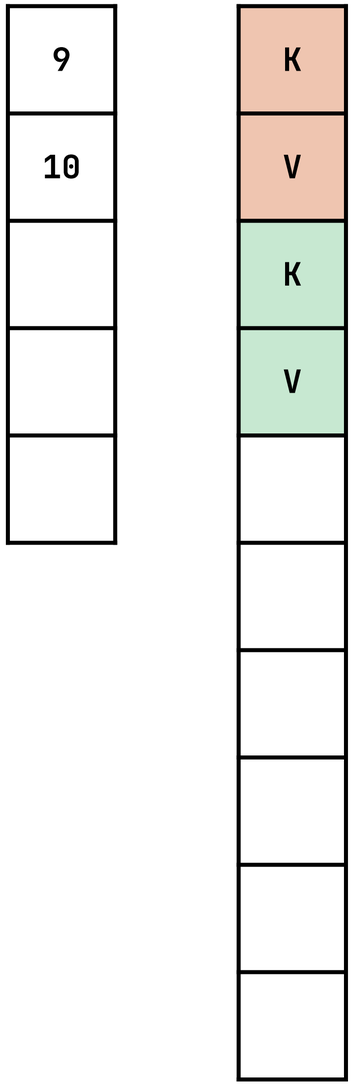
ArrayMap.put(K | V)

H = K .hashCode() = 10



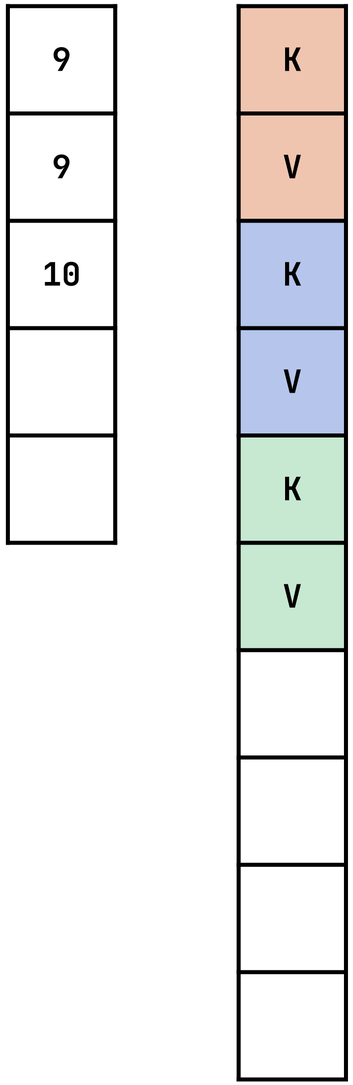
ArrayMap.put(K | V)

H = K .hashCode() = 9



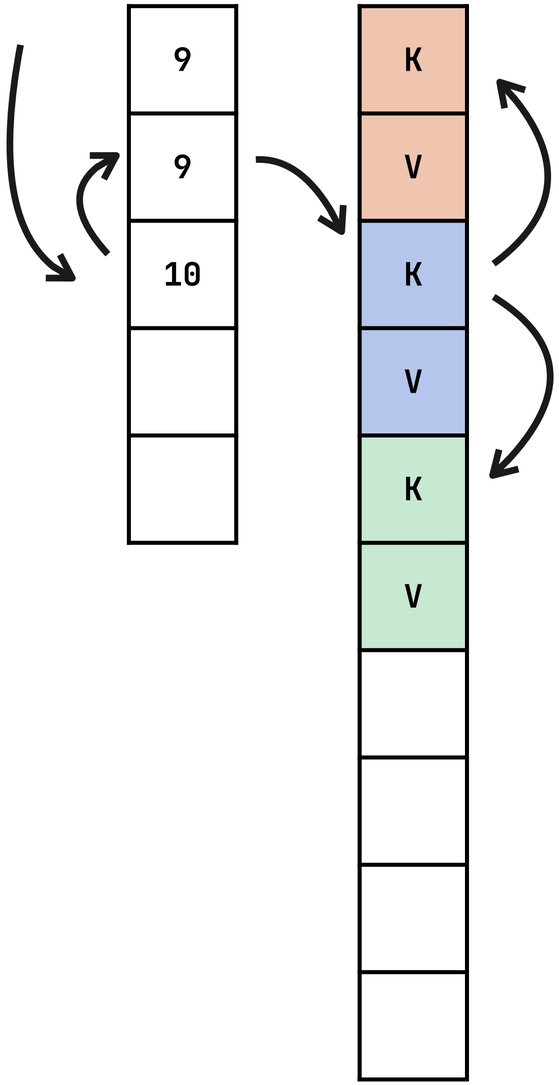
ArrayMap.put(K | V)

H = K .hashCode() = 9

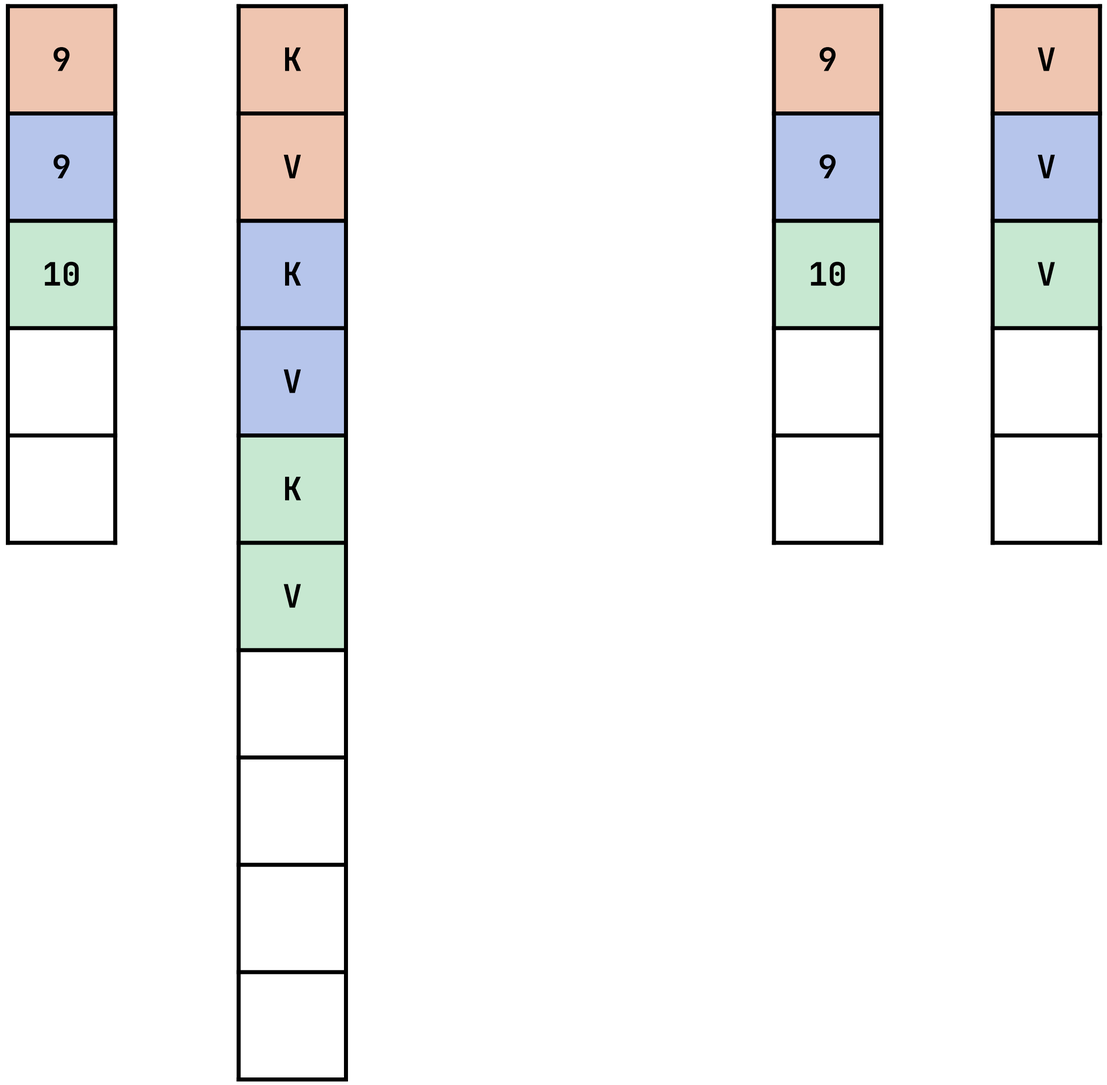


ArrayMap.get(K)

H = K .hashCode() = 9



ArrayMap & SparseArray



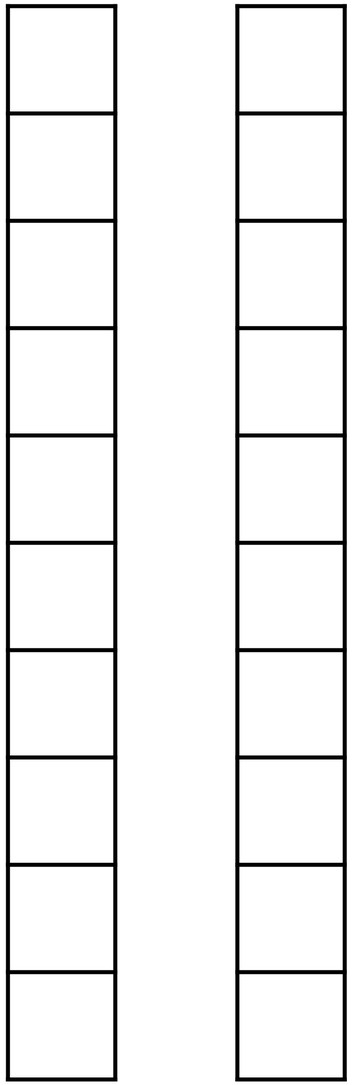
ArrayMap

- O(log n)

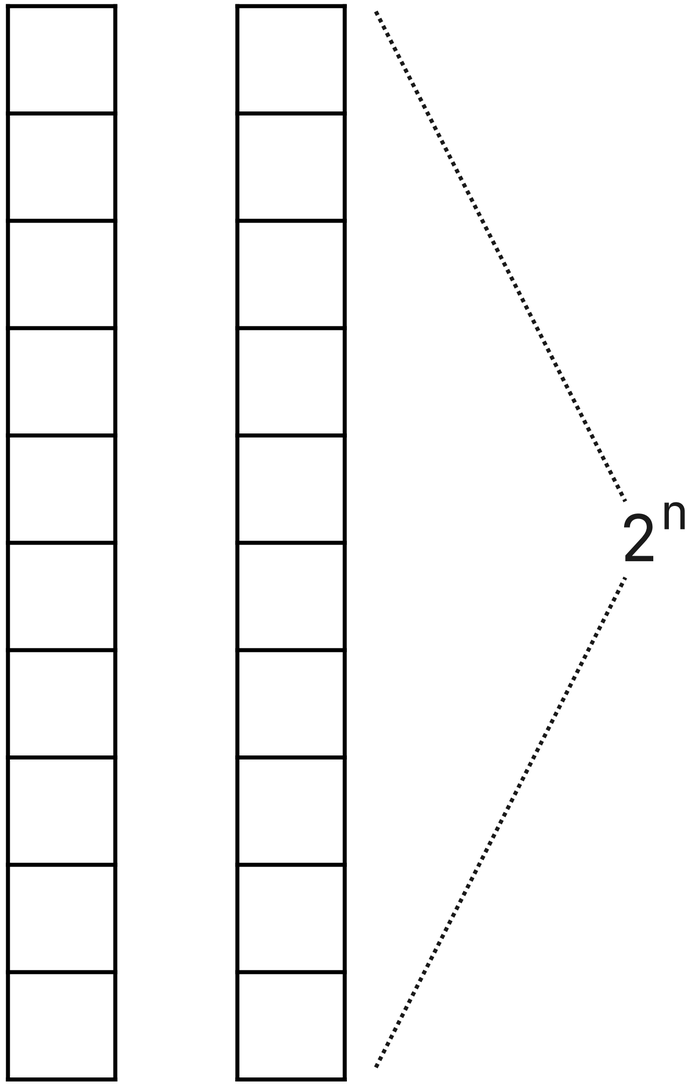
- 2 массива вместо Node (overhead 4 байта)
- Вставка без аллокаций
- SparseArray для примитивных ключей

Open Addressing / FlatHashMap

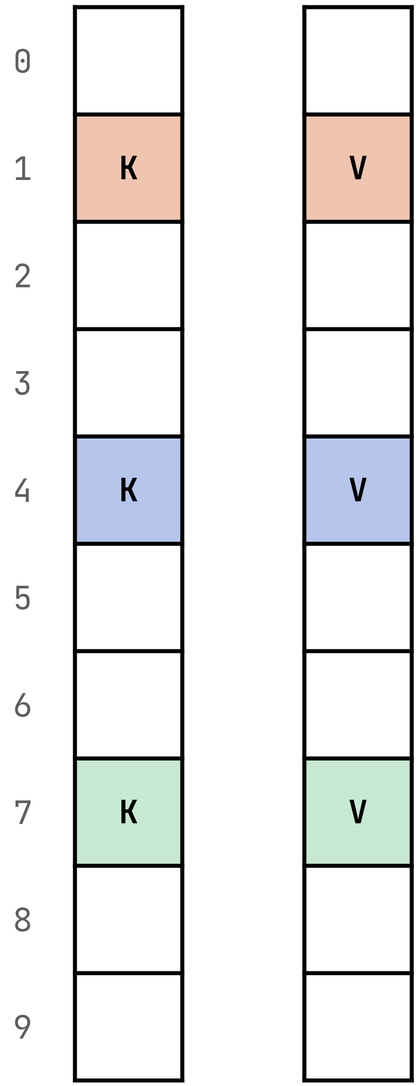
FlatHashMap



FlatHashMap



FlatHashMap



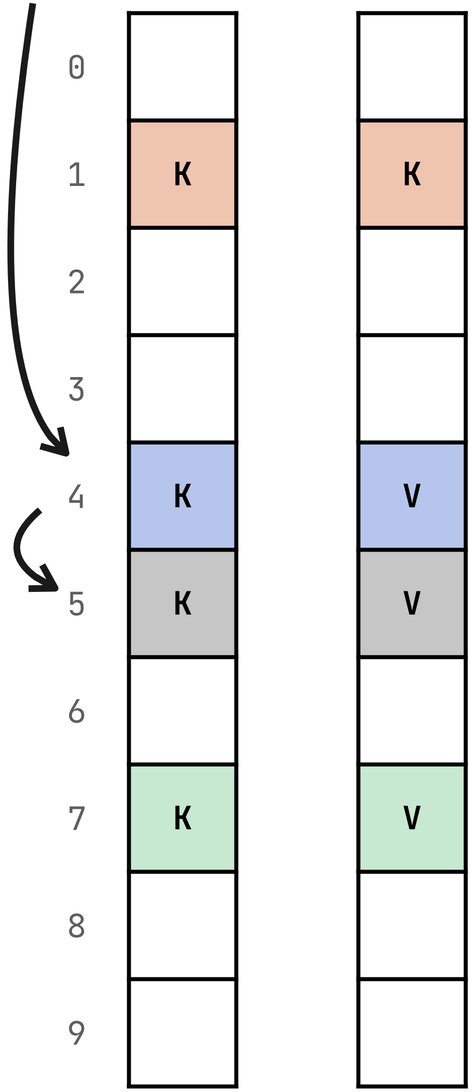
i = K .hashCode() % 10 = 1

i = K .hashCode() % 10 = 4

i = K .hashCode() % 10 = 7

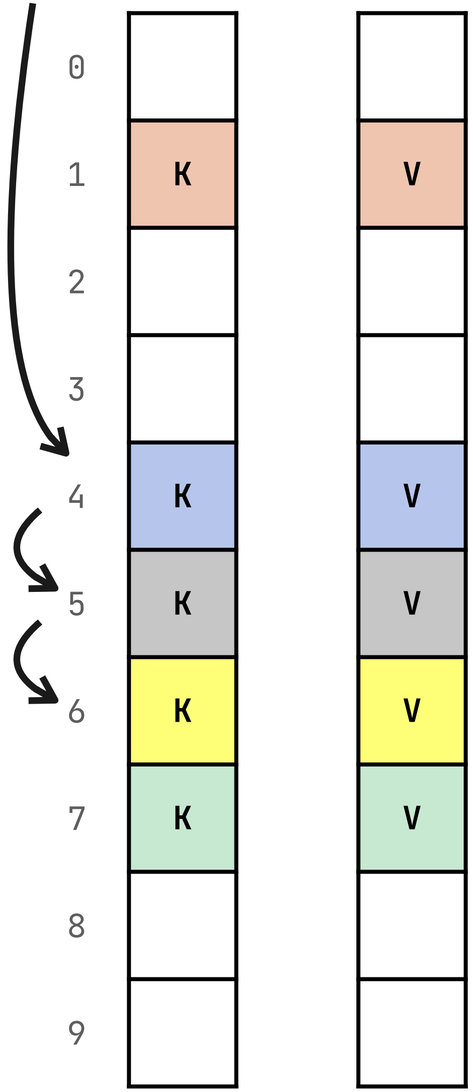
FlatHashMap.put(K | V)

i = K .hashCode() % 10 = 4

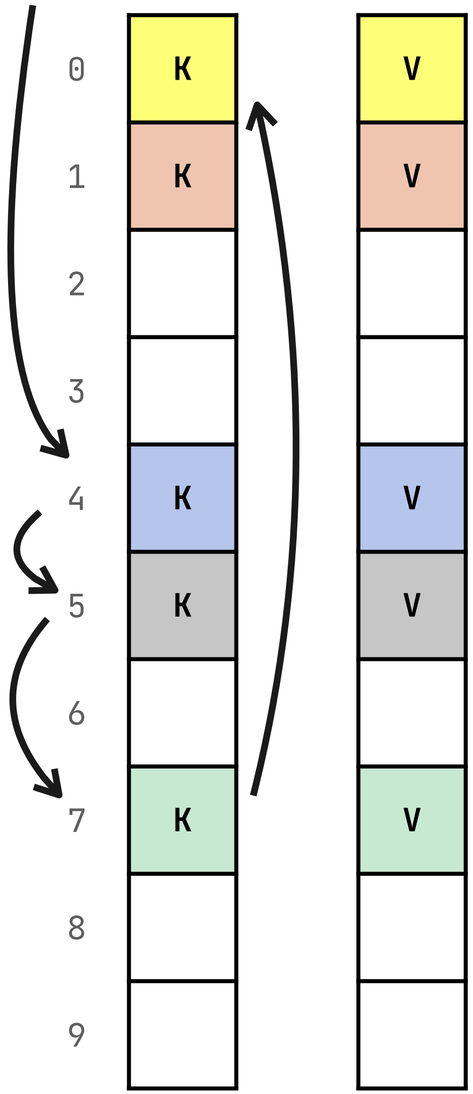


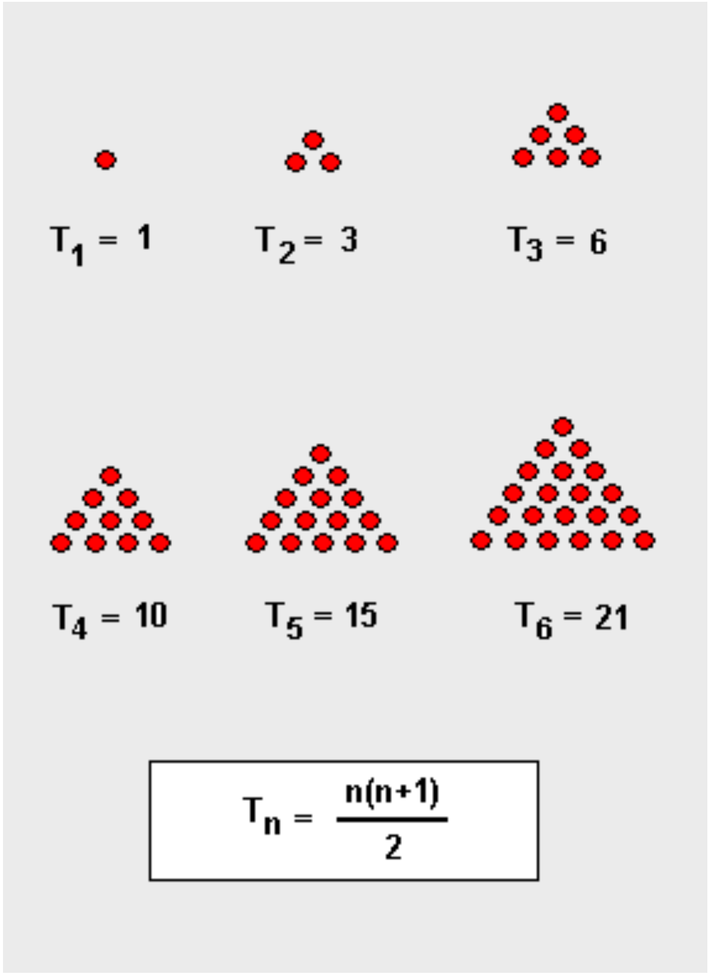
FlatHashMap.put(K V)

i = K .hashCode() % 10 = 4



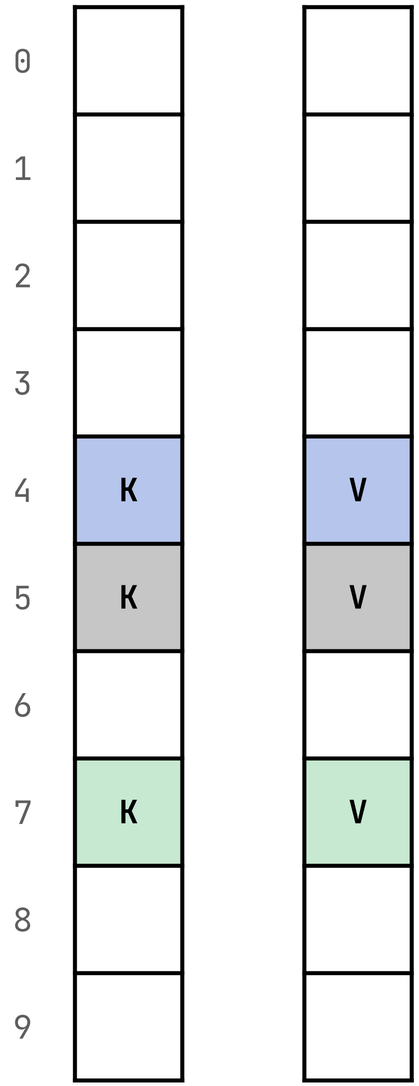
FlatHashMap





https://ru.wikipedia.org/wiki/Треугольное_число

FlatHashMap

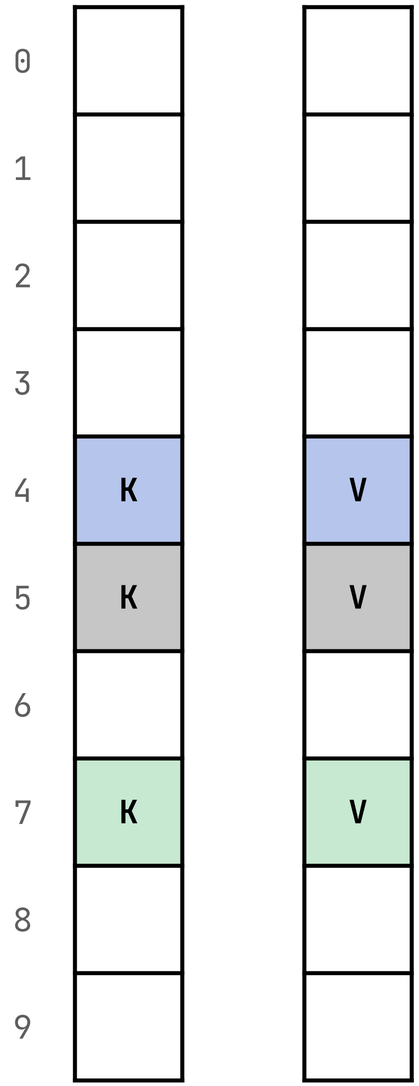


i = K .hashCode() % 10 = 4

i = K .hashCode() % 10 = 4

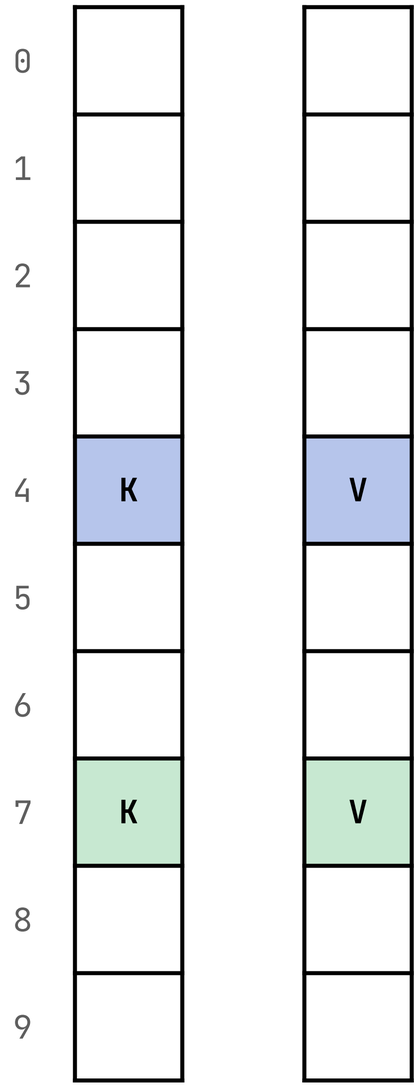
i = K .hashCode() % 10 = 4

FlatHashMap.remove(K)



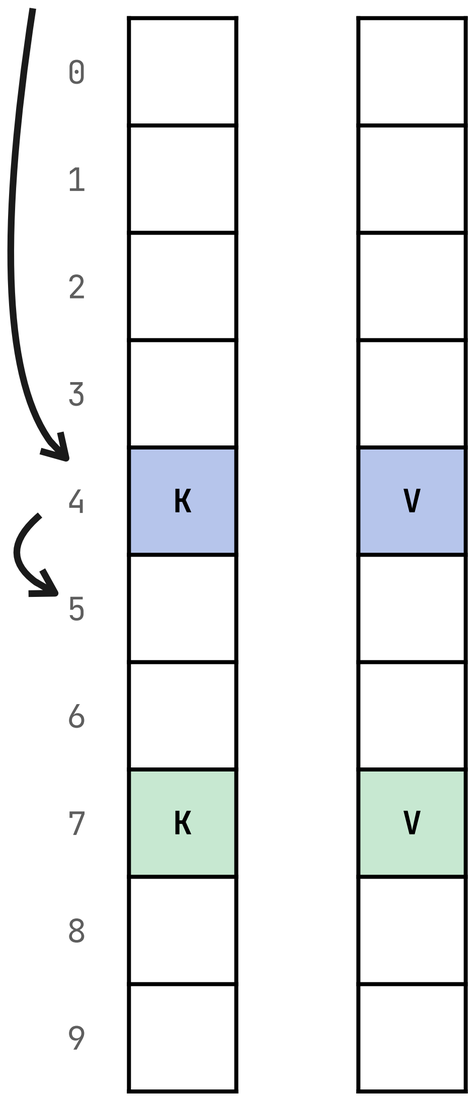

```
fun remove(key: K) {

    var i = key.hashCode() % capacity

    while (keys[i] ≠ null) {
        if (keys[i] == key) {
            // TODO delete
            return
        }
        i = next(i)
    }
}
```

FlatHashMap.remove(K)



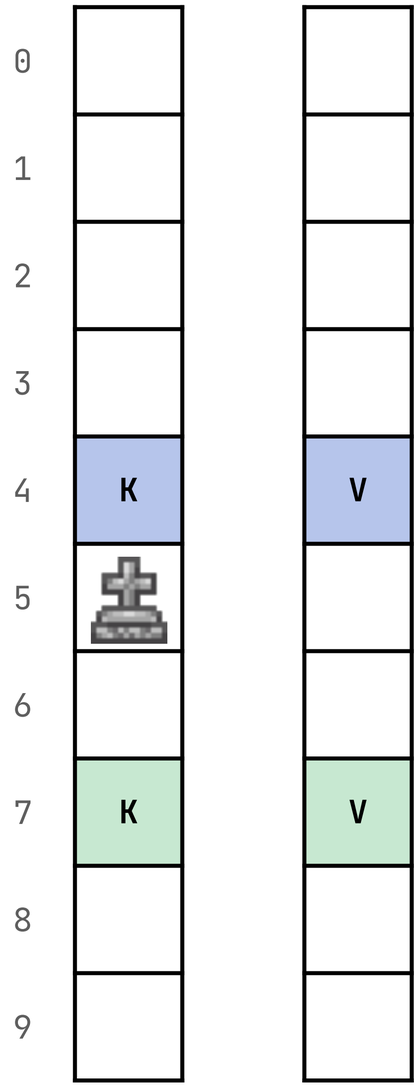

```
fun remove(key: K) {

    var i = key.hashCode() % capacity

    while (keys[i] ≠ null) {
        if (keys[i] == key) {
            keys[i] = null
            values[i] = null
            return
        }
        i = next(i)
    }
}
```

FlatHashMap.get(K)



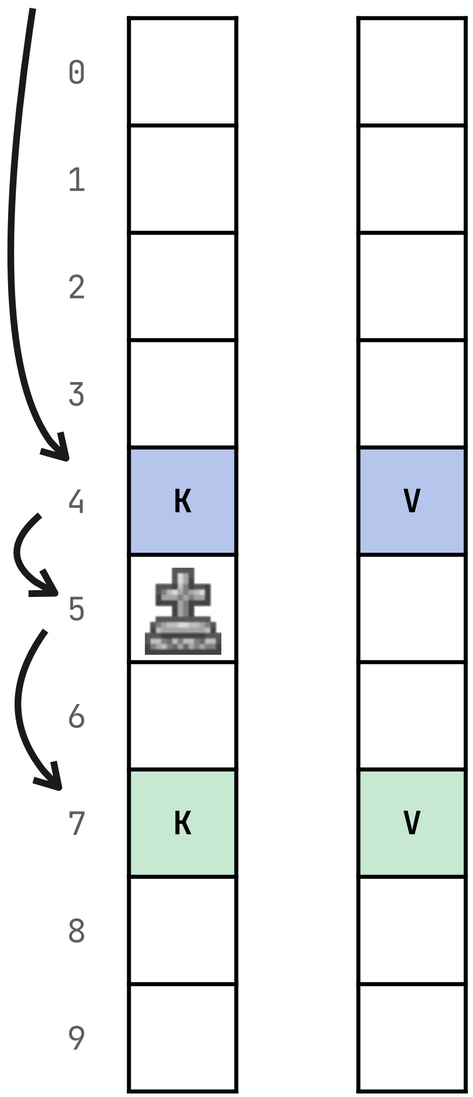

```
fun get(key: K): V? {

    var i = key.hashCode() % capacity

    while (keys[i] ≠ null) {
        if (keys[i] == key) {
            return values[i]
        }
        i = next(i)
    }

    return null
}
```

FlatHashMap.remove(K)




```
fun remove(key: K) {

    var i = key.hashCode() % capacity

    while (keys[i] ≠ null) {
        if (keys[i] == key) {
            keys[i] = TOMBSTONE
            values[i] = null
            return
        }
        i = next(i)
    }
}


val TOMBSTONE = Any()
```

FlatHashMap.get(K)




```
fun get(key: K): V? {

    var i = key.hashCode() % capacity

    while (keys[i] ≠ null) {
        if (keys[i] == key) {
            return values[i]
        }
        i = next(i)
    }

    return null
}
```

FlatHashMap

- O(1)

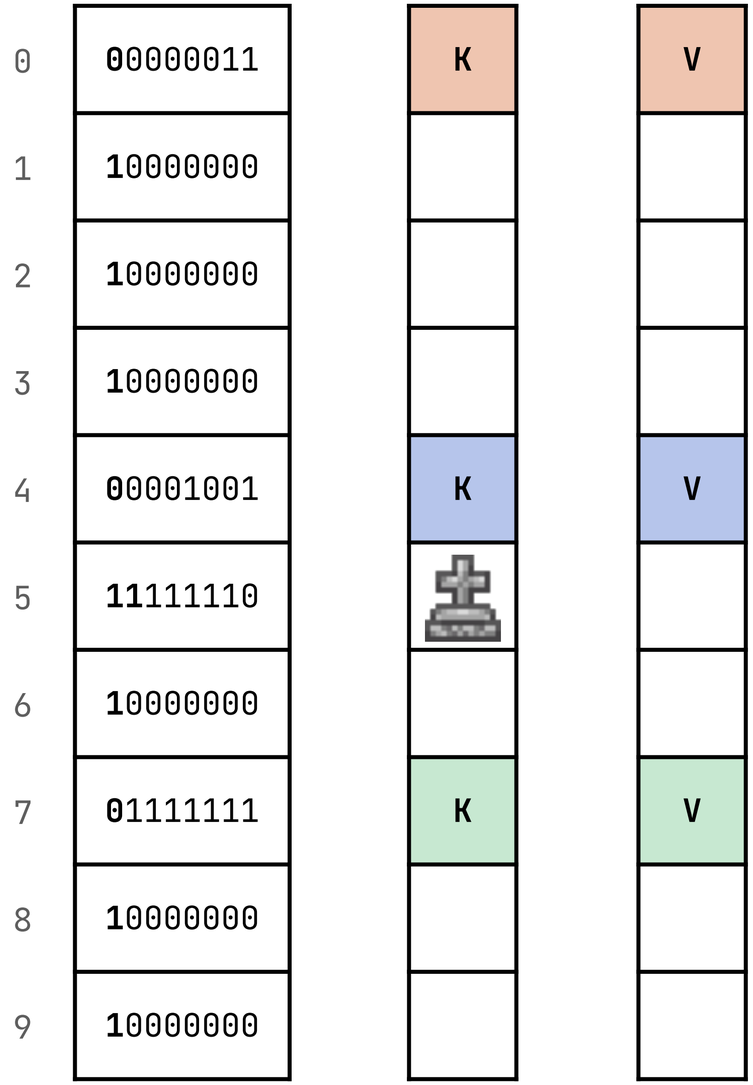
- 2 массива вместо Node
- Autoboxing примитивов
- Pointer chasing и риск cache miss
- 

почти ScatterMap

почти ScatterMap

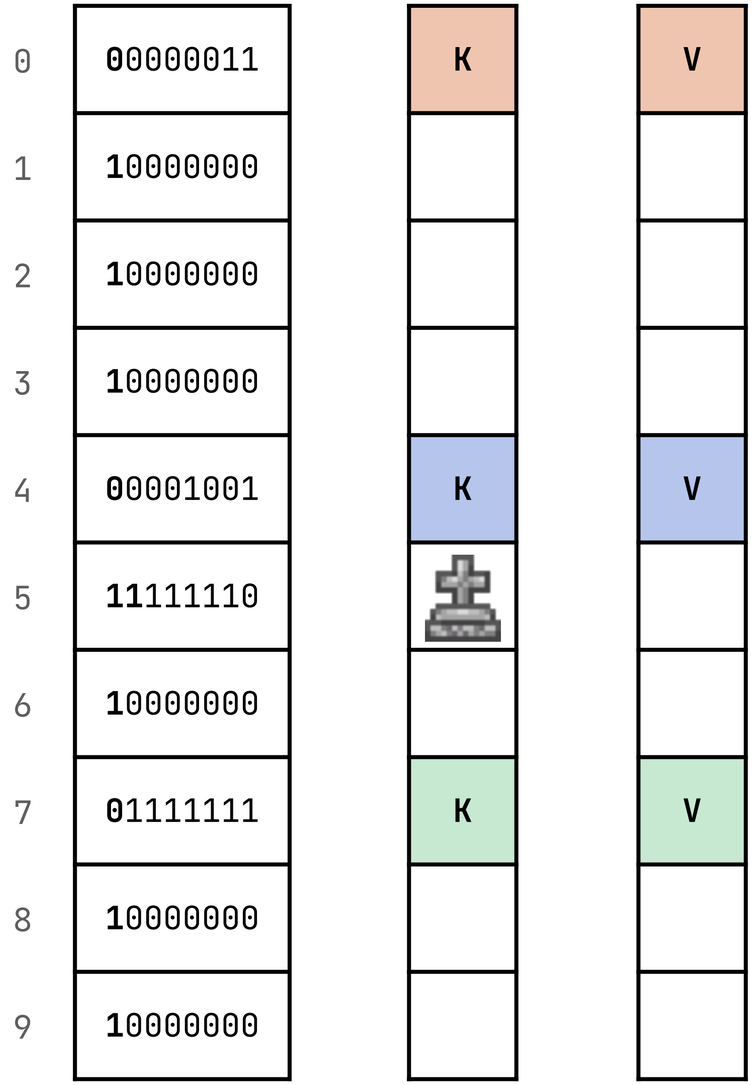
0	0	K	V
1	10		
2	10		
3	10		
4	0	K	V
5	11		
6	10		
7	0	K	V
8	10		
9	10		

почти ScatterMap



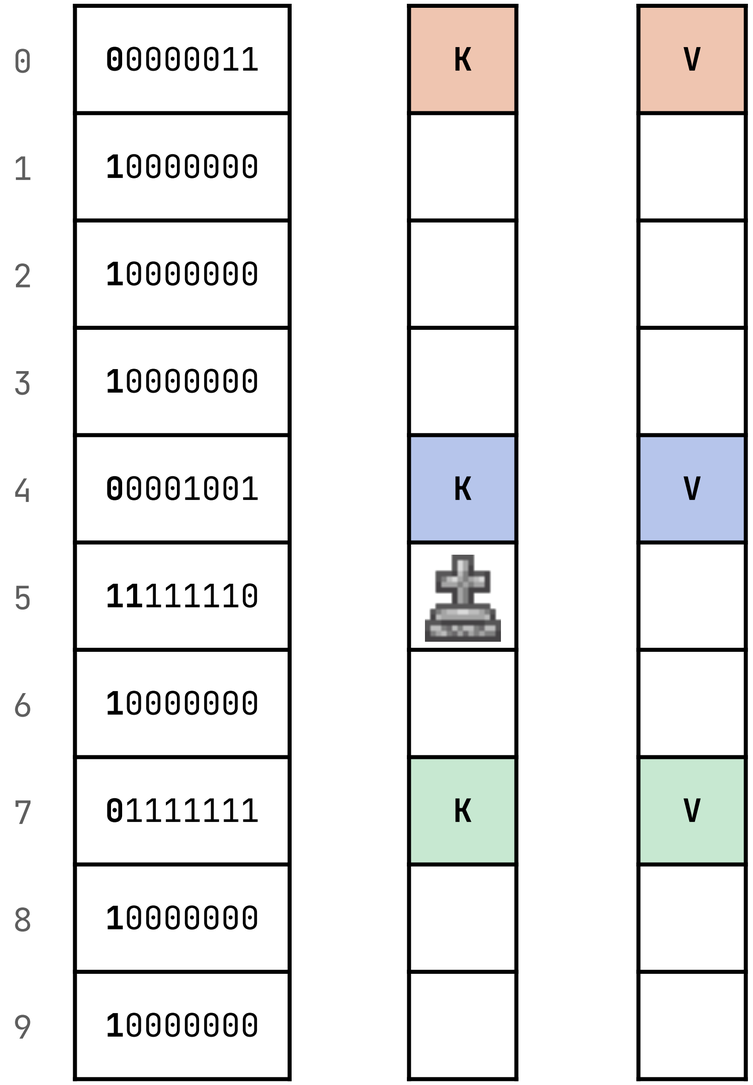

```
hashCode = 0b 0101_1011_0010_1110_1101_0011_1010_0111
             ├──────────────────── H1 ──────────────────┤├─ H2 ─┤
                                 25 бит                  7 бит
```

почти ScatterMap




```
val Any.h1: Int get = ...
val Any.h2: Int get = ... // 0xxxxxxx


fun get(key: Any): Any? {
    var i = key.h1 % capacity

    while (meta[i] ≠ 0x80) { // 10000000
        if (meta[i] == key.h2 && keys[i] == key) {
            return values[i]
        }

        i = next(i)
    }

    // Дошли до пустого слота = ключа нет
    return null
}
```

почти ScatterMap

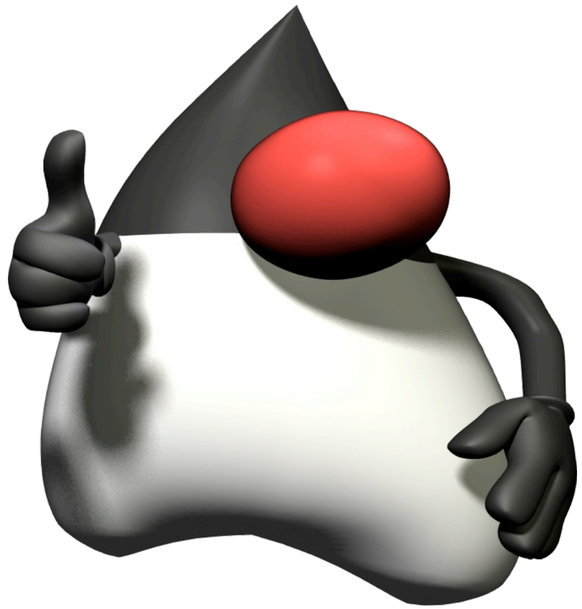



```
BitMask<uint32_t> Match(h2_t hash) const {
  auto match = _mm_set1_epi8(hash);
  return BitMask<uint32_t>(
      _mm_movemask_epi8(_mm_cmpeq_epi8(match, ctrl)));
}
```

почти ScatterMap

		K	V
0	00000011	K	V
1	10000000		
2	10000000		
3	10000000		
4	00001001	K	V
5	11111110		
6	10000000		
7	01111111	K	V
8	10000000		
9	10000000		

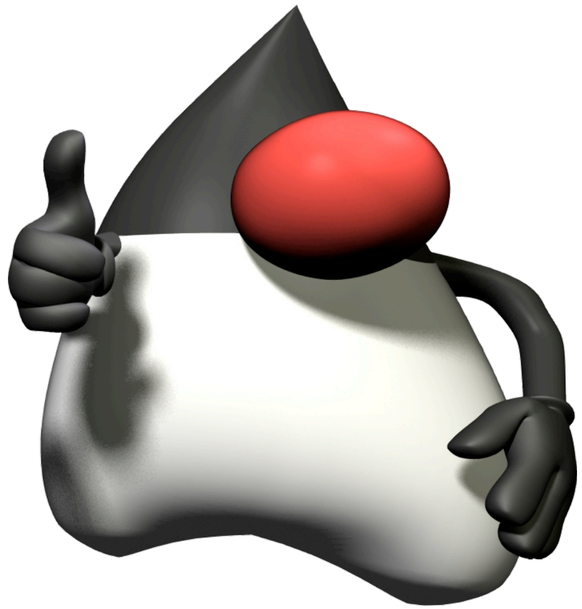
```
BitMask<uint32_t> Match(h2_t hash) const {
  auto match = _mm_set1_epi8(hash);
  return BitMask<uint32_t>(
      _mm_movemask_epi8(_mm_cmpeq_epi8(match, ctrl)));
}
```



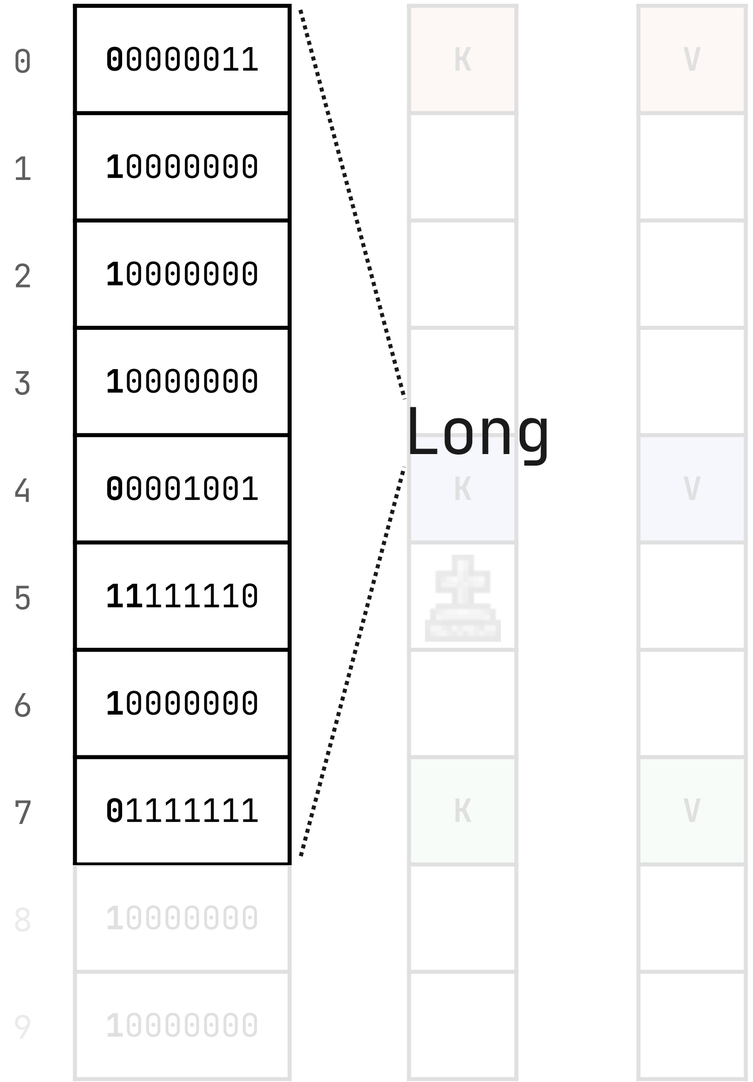
почти ScatterMap

		K	V
0	00000011		
1	10000000		
2	10000000		
3	10000000		
4	00001001	K	V
5	11111110		
6	10000000		
7	01111111	K	V
8	10000000		
9	10000000		

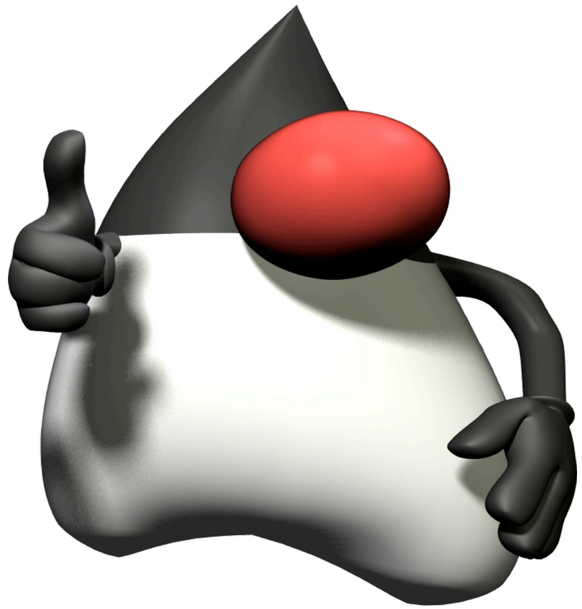
```
BitMask<uint32_t> Match(h2_t hash) const {
  auto match = _mm_set1_epi8(hash);
  return BitMask<uint32_t>(
      _mm_movemask_epi8(_mm_cmpeq_epi8(match, ctrl)));
}
```



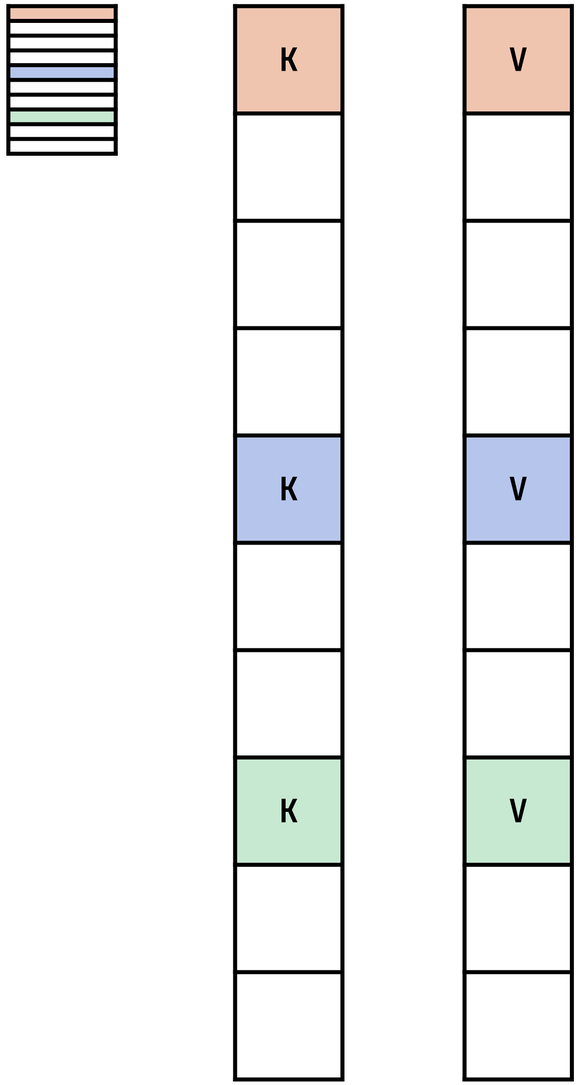
почти ScatterMap



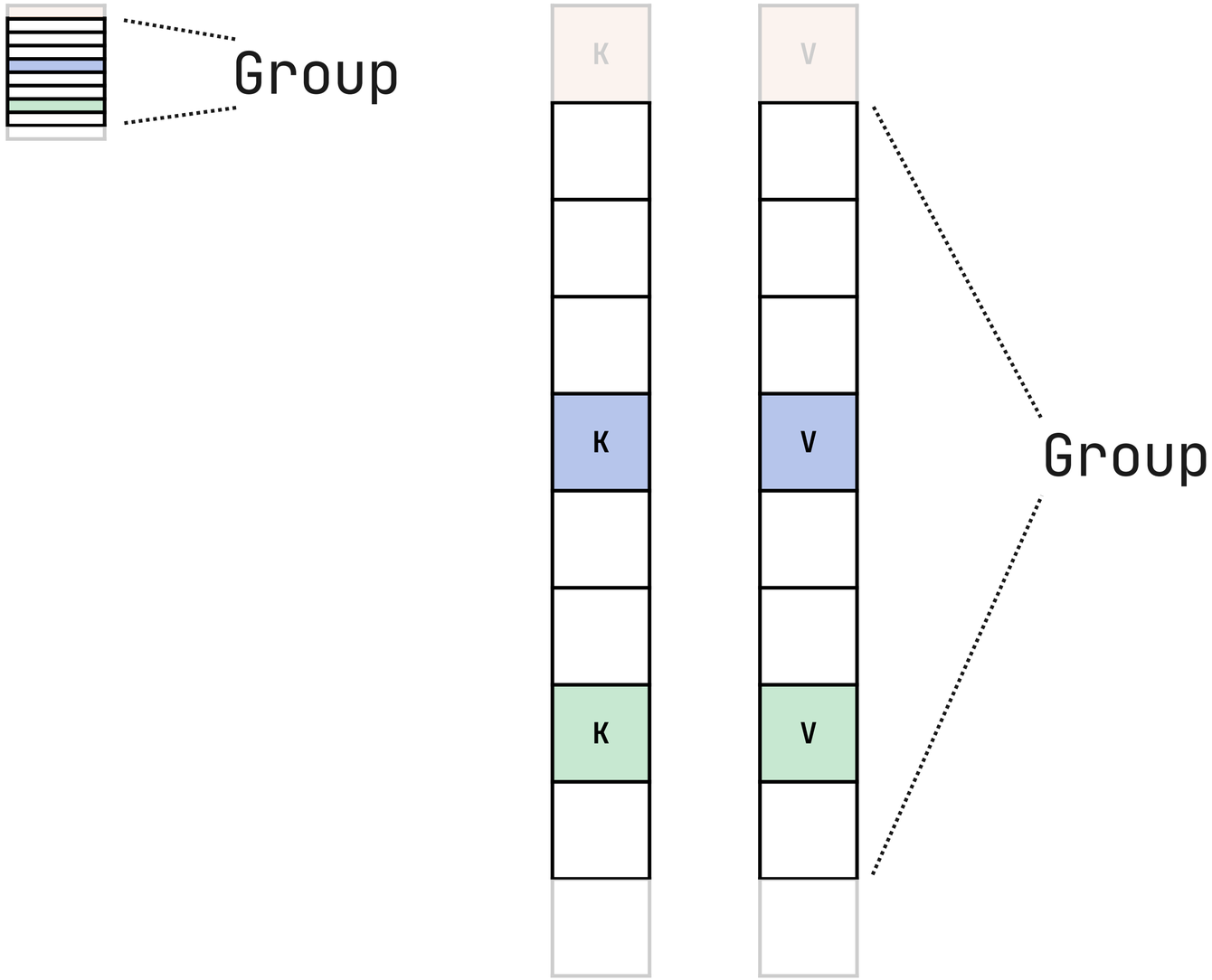

```
BitMask<uint32_t> Match(h2_t hash) const {
  auto match = _mm_set1_epi8(hash);
  return BitMask<uint32_t>(
      _mm_movemask_epi8(_mm_cmpeq_epi8(match, ctrl)));
}
```



ScatterMap



ScatterMap



ScatterMap

```
fun get(key: K): V? {
    val index = findKeyIndex(key)
    return if (index ⩾ 0) values[index] as V?
    else null
}
```

ScatterMap

```
internal inline fun findKeyIndex(key: K): Int {
    val hash = hash(key)
    val hash2 = h2(hash)

    val probeMask = _capacity
    var probeOffset = h1(hash) and probeMask
    var probeIndex = 0

    while (true) {
        val g = group(metadata, probeOffset)
        var m = g.match(hash2)
        while (m.hasNext()) {
            val index = (probeOffset + m.get()) and probeMask
            if (keys[index] == key) {
                return index
            }
            m = m.next()
        }

        if (g.maskEmpty() != 0L) {
            break
        }

        probeIndex += GroupWidth
        probeOffset = (probeOffset + probeIndex) and probeMask
    }

    return -1
}
```

ScatterMap

```
fun get(key: K): V? {

    var i = key.hashCode() % capacity

    while (keys[i] ≠ null) {
        if (keys[i] == key) {
            return values[i]
        }
        i = next(i)
    }

    return null
}
```

```
internal inline fun findKeyIndex(key: K): Int {
    val hash = hash(key)
    val hash2 = h2(hash)

    val probeMask = _capacity
    var probeOffset = h1(hash) and probeMask
    var probeIndex = 0

    while (true) {
        val g = group(metadata, probeOffset)
        var m = g.match(hash2)
        while (m.hasNext()) {
            val index = (probeOffset + m.get()) and probeMask
            if (keys[index] == key) {
                return index
            }
            m = m.next()
        }

        if (g.maskEmpty() ≠ 0L) {
            break
        }

        probeIndex += GroupWidth
        probeOffset = (probeOffset + probeIndex) and probeMask
    }

    return -1
}
```

ScatterMap

```
fun get(key: K): V? {

    var i = key.hashCode() % capacity

    while (keys[i] ≠ null) {
        if (keys[i] == key) {
            return values[i]
        }
        i = next(i)
    }

    return null
}
```

```
internal inline fun findKeyIndex(key: K): Int {
    val hash = hash(key)
    val hash2 = h2(hash)

    val probeMask = _capacity
    var probeOffset = h1(hash) and probeMask // % capacity
    var probeIndex = 0

    while (true) {
        val g = group(metadata, probeOffset)
        var m = g.match(hash2)
        while (m.hasNext()) {
            val index = (probeOffset + m.get()) and probeMask
            if (keys[index] == key) {
                return index
            }
            m = m.next()
        }

        if (g.maskEmpty() ≠ 0L) {
            break
        }

        probeIndex += GroupWidth
        probeOffset = (probeOffset + probeIndex) and probeMask
    }

    return -1
}
```

ScatterMap

```
fun get(key: K): V? {

    var i = key.hashCode() % capacity

    while (keys[i] ≠ null) {
        if (keys[i] == key) {
            return values[i]
        }
        i = next(i)
    }

    return null
}
```

```
internal inline fun findKeyIndex(key: K): Int {
    val hash = hash(key)
    val hash2 = h2(hash)

    val probeMask = _capacity
    var probeOffset = h1(hash) and probeMask
    var probeIndex = 0

    while (true) {
        val g = group(metadata, probeOffset)
        var m = g.match(hash2)
        while (m.hasNext()) {
            val index = (probeOffset + m.get()) and probeMask
            if (keys[index] == key) {
                return index
            }
            m = m.next()
        }

        if (g.maskEmpty() ≠ 0L) {
            break
        }

        probeIndex += GroupWidth
        probeOffset = (probeOffset + probeIndex) and probeMask
    }

    return -1
}
```

ScatterMap

```
fun get(key: K): V? {

    var i = key.hashCode() % capacity

    while (keys[i] ≠ null) {
        if (keys[i] == key) {
            return values[i]
        }
        i = next(i)
    }

    return null
}
```

```
internal inline fun findKeyIndex(key: K): Int {
    val hash = hash(key)
    val hash2 = h2(hash)

    val probeMask = _capacity
    var probeOffset = h1(hash) and probeMask
    var probeIndex = 0

    while (true) {
        val g = group(metadata, probeOffset)
        var m = g.match(hash2)
        while (m.hasNext()) {
            val index = (probeOffset + m.get()) and probeMask
            if (keys[index] == key) {
                return index
            }
            m = m.next()
        }

        if (g.maskEmpty() ≠ 0L) {
            break
        }

        probeIndex += GroupWidth
        probeOffset = (probeOffset + probeIndex) and probeMask
    }

    return -1
}
```

ScatterMap

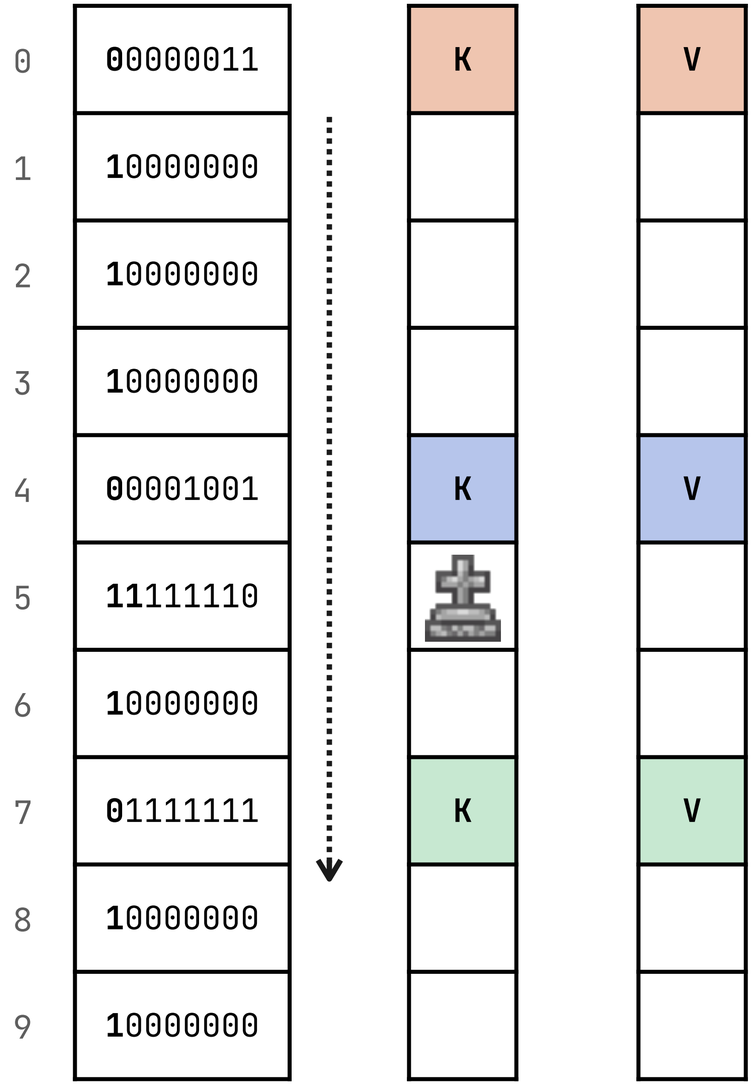
```
fun get(key: K): V? {

    var i = key.hashCode() % capacity

    while (keys[i] ≠ null) {
        if (keys[i] == key) {
            return values[i]
        }
        i = next(i)
    }

    return null
}
```

```
internal inline fun findKeyIndex(key: K): Int {
    val hash = hash(key)
    val hash2 = h2(hash)

    val probeMask = _capacity
    var probeOffset = h1(hash) and probeMask
    var probeIndex = 0

    while (true) {
        val g = group(metadata, probeOffset)
        var m = g.match(hash2)
        while (m.hasNext()) {
            val index = (probeOffset + m.get()) and probeMask
            if (keys[index] == key) {
                return index
            }
            m = m.next()
        }

        if (g.maskEmpty() ≠ 0L) {
            break
        }

        probeIndex += GroupWidth
        probeOffset = (probeOffset + probeIndex) and probeMask
    }

    return -1
}
```

ScatterMap

```
internal typealias Group = Long
internal typealias Bitmask = Long

const val BitmaskLsb: Long = 0x0101010101010101L // 0000 0001 0000 0001 ...
const val BitmaskMsb: Long = 0x8080808080808080L // 1000 0000 1000 0000 ...
```

ScatterMap



Group = 01111111 10000000 11111110 00001001 10000000 10000000 10000000 00000011

ScatterMap

```
Empty: 1 0 0 0 0 0 0 0
Deleted: 1 1 1 1 1 1 1 0
Full: 0 h h h h h h h
```

```
while (true) {
    ...
    if (g.maskEmpty() ≠ 0L) {
        break
    }
}
```

```
inline fun Group.maskEmpty(): Bitmask {
    return (this and (this.inv() shl 6)) and BitmaskMsb
}
```

ScatterMap

```
Empty: 1 0 0 0 0 0 0 0
Deleted: 1 1 1 1 1 1 1 0
Full: 0 h h h h h h h
```

```
inline fun Group.maskEmpty(): Bitmask {
    return (this and (this.inv() shl 6)) and BitmaskMsb
}
```

0. 00001001 11111110 10000000

ScatterMap

```
Empty: 1 0 0 0 0 0 0 0
Deleted: 1 1 1 1 1 1 1 0
Full: 0 h h h h h h h
```

```
inline fun Group.maskEmpty(): Bitmask {
    return (this and (this.inv() shl 6)) and BitmaskMsb
}
```

0. 00001001 11111110 10000000

1. 11110110 00000001 01111111 = inv this

2. 10000000 01011111 11000000 = 1. shl 6

3. 00000000 01011110 10000000 = 2. and this

4. 00000000 00000000 10000000 = 3. and BitmaskMsb

ScatterMap

```
inline fun group(metadata: LongArray, offset: Int): Group {
    val i = offset shr 3
    val b = (offset and 0x7) shl 3
    return (metadata[i] ushr b) or (metadata[i + 1] shl (64 - b) and (-(b.toLong()) shr 63))
}
```

ScatterMap

```
inline fun group(metadata: LongArray, offset: Int): Group {
    val i = offset shr 3
    val b = (offset and 0x7) shl 3
    return (metadata[i] ushr b) or (metadata[i + 1] shl (64 - b) and (-(b.toLong()) shr 63))
}
```

[**77 66 55 44 33 22** 11 00], [FF EE DD CC BB AA **99 88**]; offset = 2

ScatterMap

```
inline fun group(metadata: LongArray, offset: Int): Group {
    val i = offset shr 3
    val b = (offset and 0x7) shl 3
    return (metadata[i] ushr b) or (metadata[i + 1] shl (64 - b) and (-(b.toLong()) shr 63))
}
```

[**77 66 55 44 33 22** 11 00], [FF EE DD CC BB AA **99 88**]; offset = 2

```
i = offset / 8 = 0         // индекс Long, в котором начинается offset
b = (offset % 8) * 8 = 16  // сдвиг от границ Long в битах
```

ScatterMap

```
inline fun group(metadata: LongArray, offset: Int): Group {
    val i = offset shr 3
    val b = (offset and 0x7) shl 3
    return (metadata[i] ushr b) or (metadata[i + 1] shl (64 - b) and (-(b.toLong()) shr 63))
}
```

```
[77 66 55 44 33 22 11 00], [FF EE DD CC BB AA 99 88]; offset = 2

i = offset / 8 = 0          // индекс Long, в котором начинается offset
b = (offset % 8) * 8 = 16   // сдвиг от границ Long в битах


LSBs = metadata[0] ushr 16        = 0x0000776655443322
MSBs = metadata[1] shl (64 - 16) = 0x9988000000000000
```

ScatterMap

```
inline fun group(metadata: LongArray, offset: Int): Group {
    val i = offset shr 3
    val b = (offset and 0x7) shl 3
    return (metadata[i] ushr b) or (metadata[i + 1] shl (64 - b) and (-(b.toLong()) shr 63))
}
```

```
[77 66 55 44 33 22 11 00], [FF EE DD CC BB AA 99 88]; offset = 2

i = offset / 8 = 0          // индекс Long, в котором начинается offset
b = (offset % 8) * 8 = 16   // сдвиг от границ Long в битах


LSBs = metadata[0] ushr 16        = 0x0000776655443322
MSBs = metadata[1] shl (64 - 16) = 0x9988000000000000
```

ScatterMap

```
inline fun group(metadata: LongArray, offset: Int): Group {
    val i = offset shr 3
    val b = (offset and 0x7) shl 3
    return (metadata[i] ushr b) or (metadata[i + 1] shl (64 - b) and (-(b.toLong()) shr 63))
}
```

```
[77 66 55 44 33 22 11 00], [FF EE DD CC BB AA 99 88]; offset = 2

i = offset / 8 = 0          // индекс Long, в котором начинается offset
b = (offset % 8) * 8 = 16   // сдвиг от границ Long в битах


LSBs = metadata[0] ushr 16        = 0x0000776655443322
MSBs = metadata[1] shl (64 - 16) = 0x9988000000000000
```

```
x shl 64 = x
```

ScatterMap

```
inline fun group(metadata: LongArray, offset: Int): Group {
    val i = offset shr 3
    val b = (offset and 0x7) shl 3
    return (metadata[i] ushr b) or (metadata[i + 1] shl (64 - b) and (-(b.toLong()) shr 63))
}
```

[**77 66 55 44 33 22** 11 00], [FF EE DD CC BB AA **99 88**]; offset = 2

```
i = offset / 8 = 0          // индекс Long, в котором начинается offset
b = (offset % 8) * 8 = 16   // сдвиг от границ Long в битах
```

```
LSBs = metadata[0] ushr 16        = 0x0000776655443322
MSBs = metadata[1] shl (64 - 16) = 0x998800000000000
```

```
x shl 64 = x          if (b == 0) return metadata[i]
```

ScatterMap

```
(-b shr 63)


b  = 0   = 00000000_00000000_...._00000000_00000000
-b = 0   = 00000000_00000000_...._00000000_00000000
shr 63   = 00000000_00000000_...._00000000_00000000
           ^

b  =  18 = 00000000_00000000_...._00000000_00010010
-b = -18 = 11111111_11111111_...._11111111_11101110
shr 63   = 11111111_11111111_...._11111111_11111111
           ^
```

ScatterMap

```
inline fun group(metadata: LongArray, offset: Int): Group {
    val i = offset shr 3
    val b = (offset and 0x7) shl 3
    return (metadata[i] ushr b) or (metadata[i + 1] shl (64 - b) and (-(b.toLong()) shr 63))
}
```

```
[77 66 55 44 33 22 11 00], [FF EE DD CC BB AA 99 88]; offset = 2

i = offset / 8 = 0          // индекс Long, в котором начинается offset
b = (offset % 8) * 8 = 16   // сдвиг от границ Long в битах


LSBs = metadata[0] ushr 16         = 0x0000776655443322
MSBs = metadata[1] shl (64 - 16) = 0x9988000000000000


return 0x0000776655443322 + (0x9988000000000000 * (if (b == 0) 0 else 1))
```

ScatterMap

```
var m = group.match(hash2)
while (m.hasNext()) {
    val index = (probeOffset + m.get()) and probeMask
    if (keys[index] == key) {
        return index
    }
    m = m.next()
}
```

ScatterMap

```
Empty: 1 0 0 0 0 0 0 0
Deleted: 1 1 1 1 1 1 1 0
Full: 0 h h h h h h h
```

```
inline fun Group.match(m: Int): Bitmask {
    val x = this xor (BitmaskLsb * m)
    return (x - BitmaskLsb) and x.inv() and BitmaskMsb
}
```

```
0. 11111110 00001001 10000000 = this, 1001 = m
```

ScatterMap

```
Empty: 1 0 0 0 0 0 0 0
Deleted: 1 1 1 1 1 1 1 0
Full: 0 h h h h h h h
```

```
inline fun Group.match(m: Int): Bitmask {
    val x = this xor (BitmaskLsb * m)
    return (x - BitmaskLsb) and x.inv() and BitmaskMsb
}
```

0. 11111110 00001001 10000000 = this, 1001 = m

1. 00001001 00001001 00001001 = m * *BitmaskLsb*

2. 11110111 00000000 10001001 = 1. xor this = x

3. ...

4. ...

5. 00000000 10000000 00000000 = 3. and 4. and *BitmaskMsb*

ScatterMap

```
(x - BitmaskLsb) and x.inv() and BitmaskMsb
```

Свойства нулевого байта:

- Старший бит всегда 0, исключаются числа 10000000..11111111
- После вычитания 1: старший бит всегда 1, исключаются числа 00000001..10000000

```
0. 11110111 00000000 10001001 = x

1. 00001000 11111111 01110110 = inv x

2. 11110101 11111111 10001000 = x - 00000001 00000001 00000001

3. 00000000 11111111 00000000 = 1. and 2.

4. 00000000 10000000 00000000 = 3. and BitmaskMsb
```

ScatterMap

```
var m = group.match(hash2) // 10000000 10000000 00000000
while (m.hasNext()) {
    val index = (probeOffset + m.get()) and probeMask
    if (keys[index] == key) {
        return index
    }
    m = m.next()
}
```

ScatterMap

```
var m = group.match(hash2) // 10000000 10000000 00000000
while (m.hasNext()) {
    val index = (probeOffset + m.get()) and probeMask
    if (keys[index] == key) {
        return index
    }
    m = m.next()
}
```

```
inline fun Bitmask.hasNext() = this != 0L
```

ScatterMap

```
var m = group.match(hash2) // 10000000 10000000 00000000
while (m.hasNext()) {
    val index = (probeOffset + m.get()) and probeMask
    if (keys[index] == key) {
        return index
    }
    m = m.next()
}
```

```
inline fun Bitmask.hasNext() = this ≠ 0L

inline fun Bitmask.get() = countTrailingZeroBits() shr 3
```

ScatterMap

```
var m = group.match(hash2) // 10000000 10000000 00000000
while (m.hasNext()) {
    val index = (probeOffset + m.get()) and probeMask
    if (keys[index] == key) {
        return index
    }
    m = m.next()
}
```

```
inline fun Bitmask.hasNext() = this != 0L

inline fun Bitmask.get() = countTrailingZeroBits() shr 3

inline fun Bitmask.next() = this and (this - 1L)
```

```
0. 10000000 10000000 00000000 = this
1. 10000000 01111111 11111111 = this - 1
2. 10000000 00000000 00000000 = 0. and 1.
```

ScatterMap

```
internal inline fun findKeyIndex(key: K): Int {
    val hash = hash(key)
    val hash2 = h2(hash)

    val probeMask = _capacity
    var probeOffset = h1(hash) and probeMask
    var probeIndex = 0

    while (true) {
        val g = group(metadata, probeOffset)
        var m = g.match(hash2)
        while (m.hasNext()) {
            val index = (probeOffset + m.get()) and probeMask
            if (keys[index] == key) {
                return index
            }
            m = m.next()
        }

        if (g.maskEmpty() != 0L) {
            break
        }

        probeIndex += GroupWidth
        probeOffset = (probeOffset + probeIndex) and probeMask
    }

    return -1
}
```

ScatterMap

```
internal fun adjustStorage() {
    if (_capacity > GroupWidth && _size ⩽ 25/32 * _capacity) {
        dropDeletes()
    } else {
        resizeStorage(nextCapacity(_capacity)) // LOAD_FACTOR = 7/8
    }
}
```

```
25/32 = 0.78125
```

```
7/8 = 0.875
```

https://github.com/abseil/abseil-cpp/.../raw_hash_set.cc#L1587

ScatterMap

- O(1)

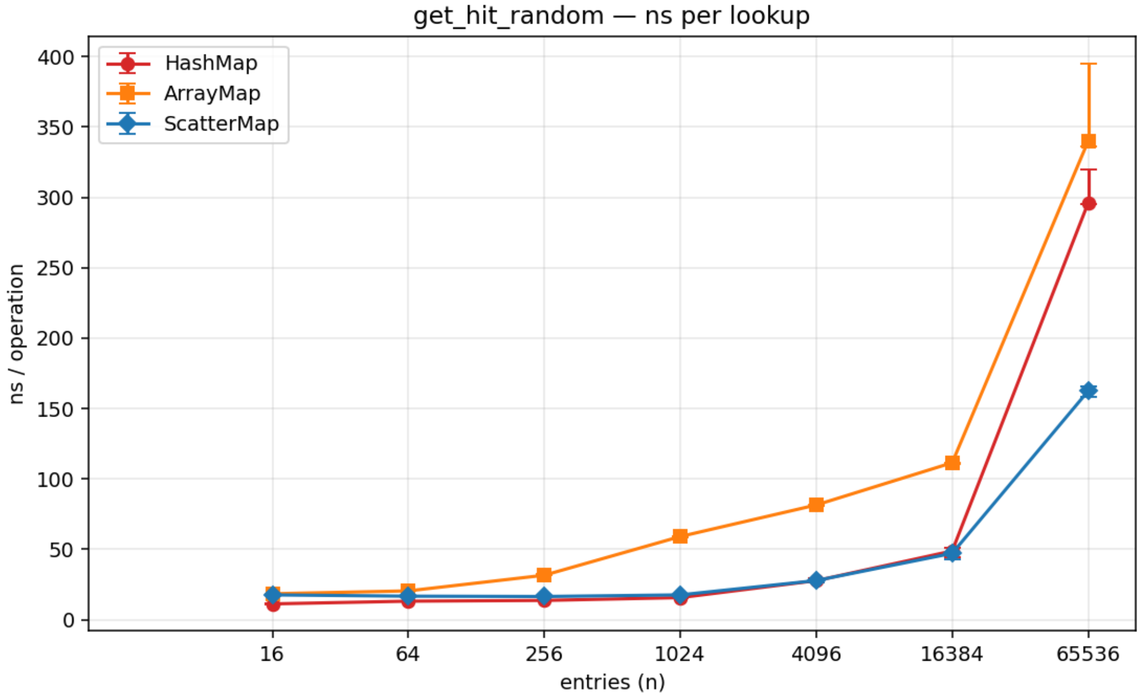
- 1 байт вместо 32
- Вставка без аллокаций
- IntIntMap, LongLongMap, ... - без boxing
- Cache friendly, проверки по 8 слотов за раз в LongArray
- Нет pointer chasing
- Load factory 0.875, на 17% больше чем в HashMap

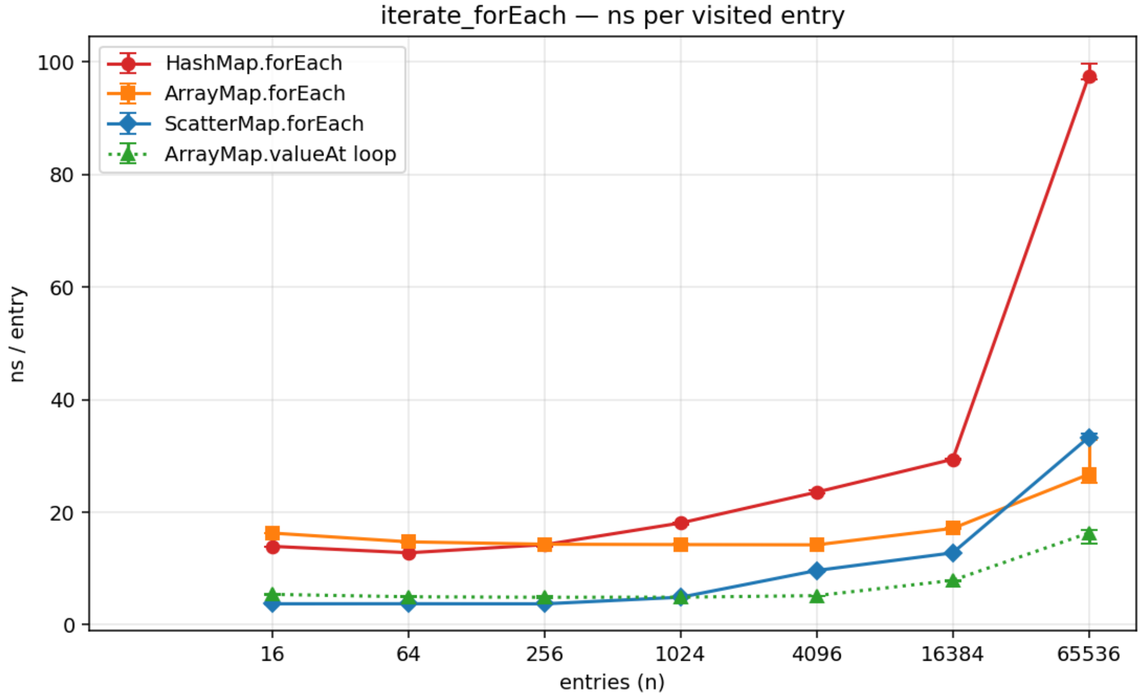
Когда использовать ScatterMap

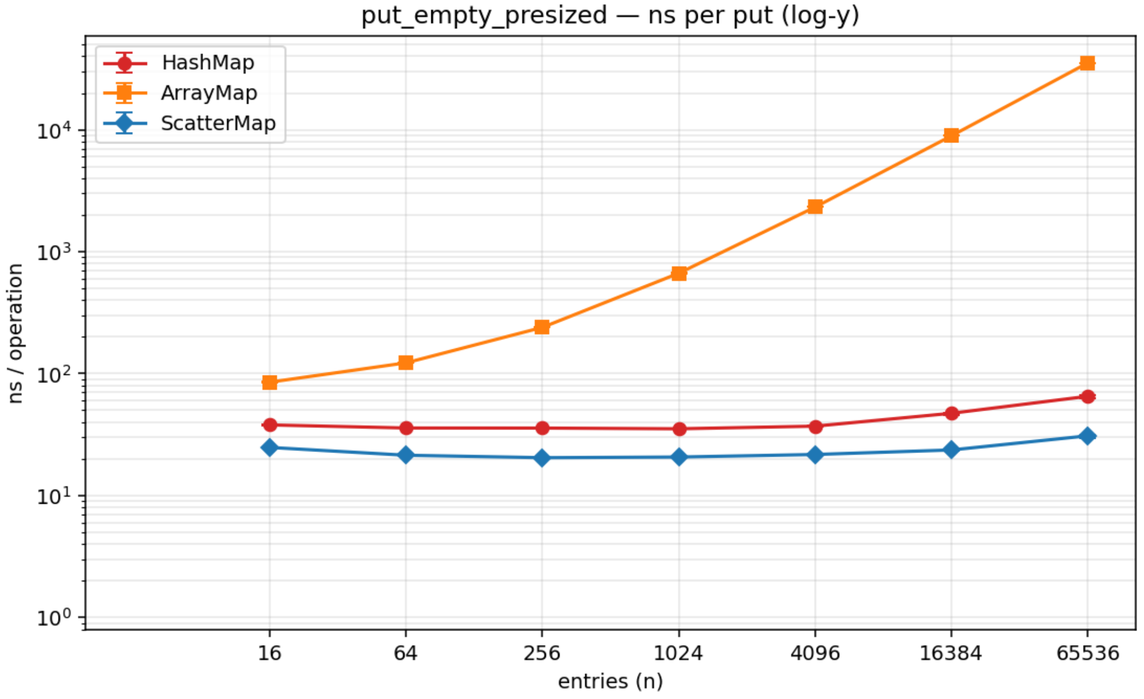
- hot path с частыми put/get
- долгоживущая коллекция, чувствительная к GC
- примитивные ключи или значения (IntIntMap и т.д.)

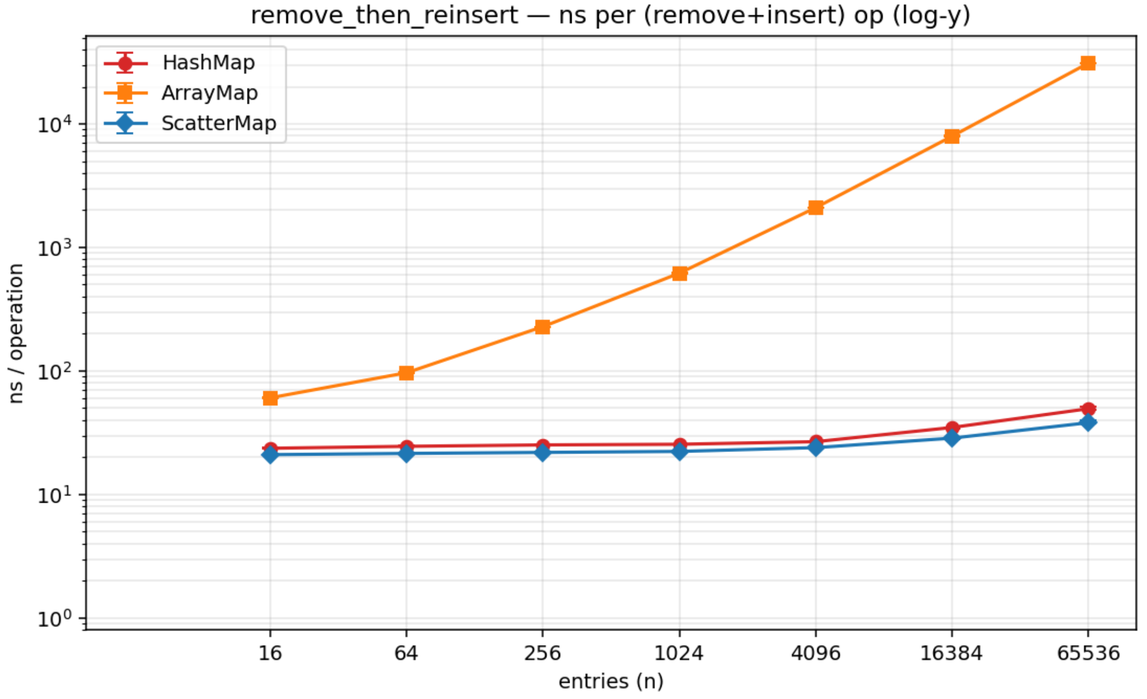
Когда не использовать ScatterMap

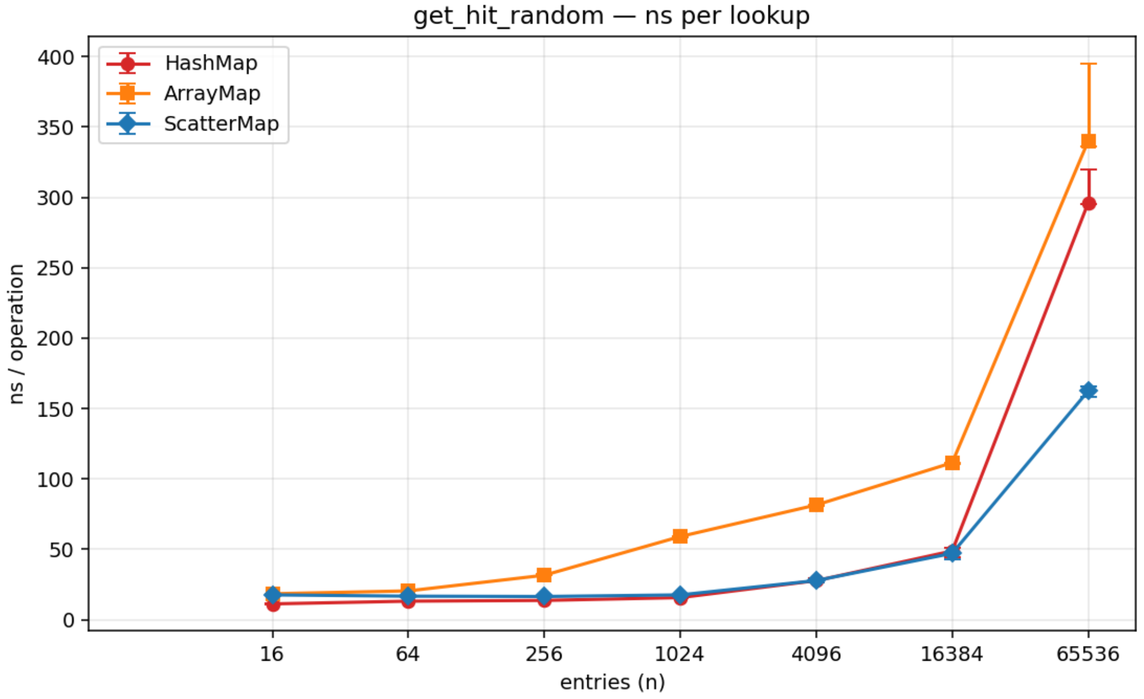
- нужен порядок вставки (LinkedHashMap)
- коллекция маленькая и короткоживущая

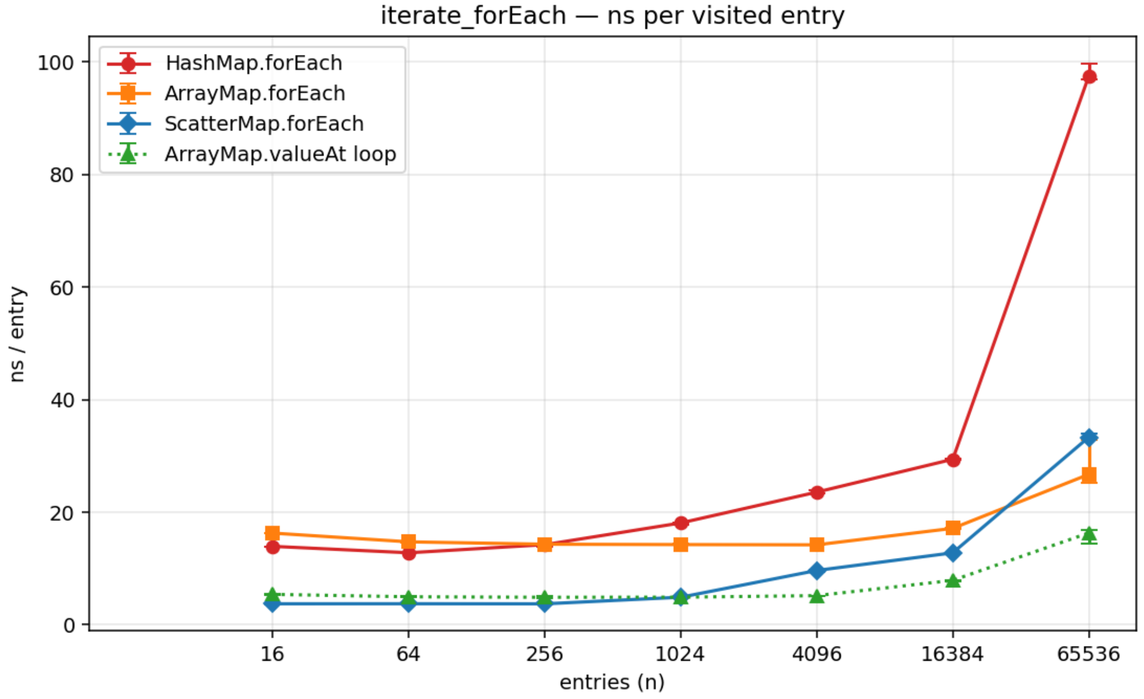

get_hit_random — ns per lookup
HashMap
ArrayMap
ScatterMap
ns / operation
entries (n)
16
64
256
1024
4096
16384
65536

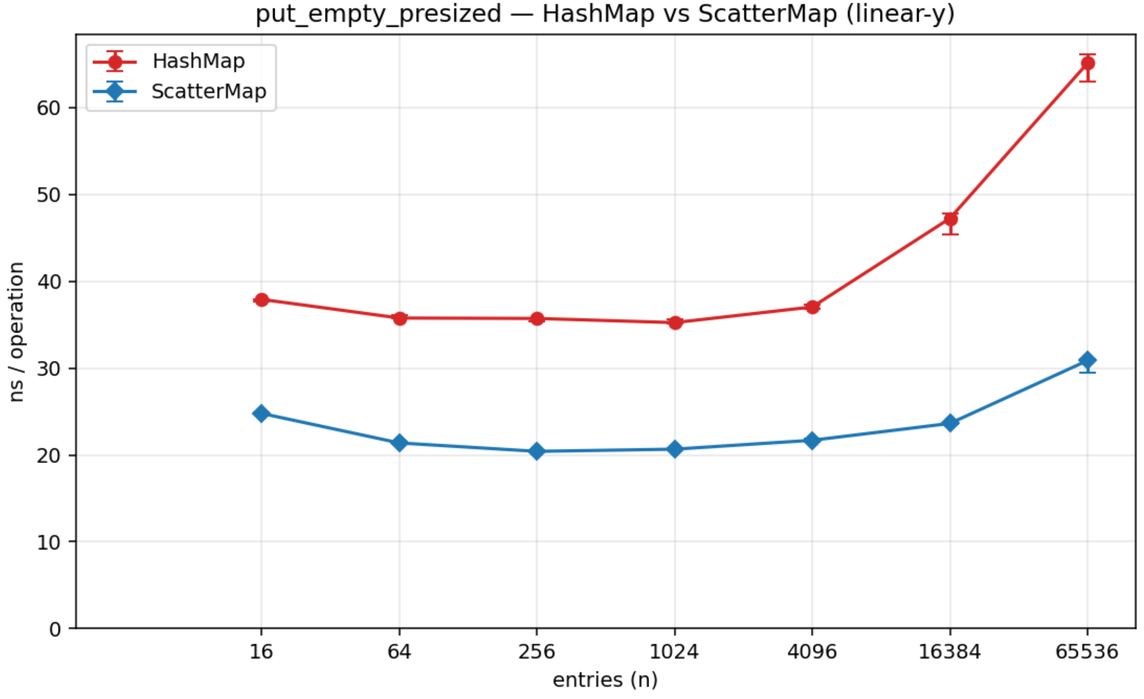

iterate_forEach — ns per visited entry
HashMap.forEach
ArrayMap.forEach
ScatterMap.forEach
ArrayMap.valueAt loop
ns / entry
entries (n)

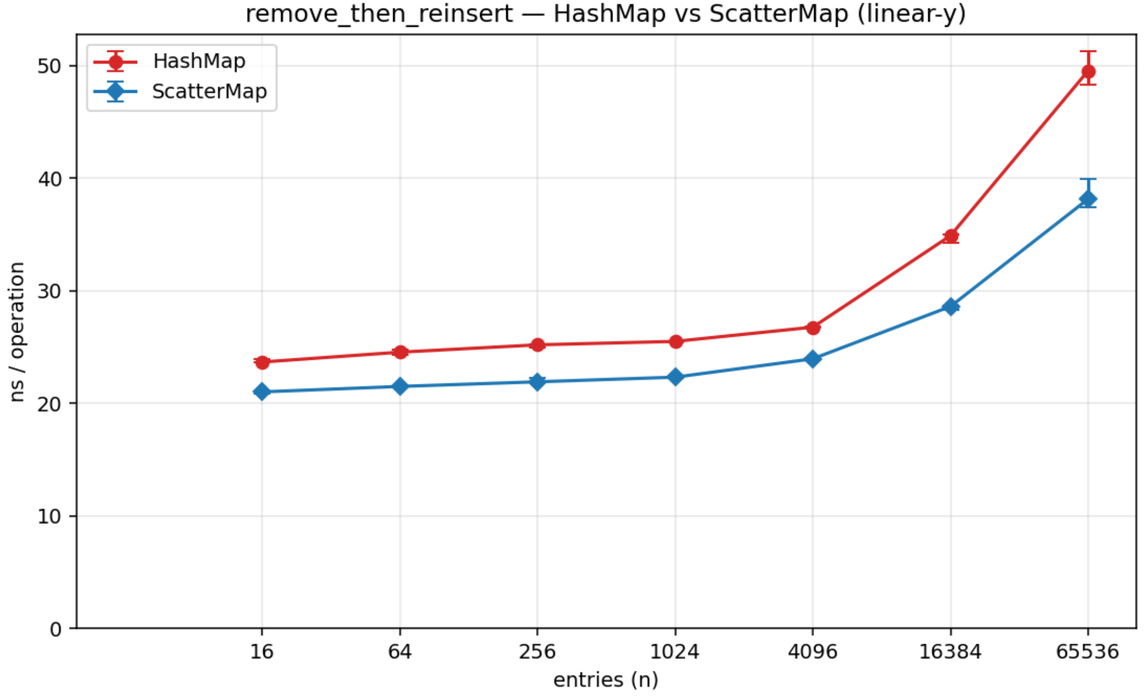

put_empty_presized — ns per put (log-y)
HashMap
ArrayMap
ScatterMap
ns / operation
entries (n)


remove_then_reinsert — ns per (remove+insert) op (log-y)
HashMap
ArrayMap
ScatterMap
ns / operation
entries (n)


get_hit_random — ns per lookup
HashMap
ArrayMap
ScatterMap
ns / operation
entries (n)
16
64
256
1024
4096
16384
65536


iterate_forEach — ns per visited entry
HashMap.forEach
ArrayMap.forEach
ScatterMap.forEach
ArrayMap.valueAt loop
ns / entry
entries (n)


put_empty_presized — HashMap vs ScatterMap (linear-y)
HashMap
ScatterMap
ns / operation
entries (n)


remove_then_reinsert — HashMap vs ScatterMap (linear-y)
HashMap
ScatterMap
ns / operation
entries (n)