Пишем эффективную многопоточную хеш-таблицу

Никита Коваль

ndkoval *at* ya *dot* ru
twitter.com/nkoval_

DEVEXPERTS

Осторожно! Этот доклад про многопоточность и алгоритмы!*

* В других залах вы можете послушать про связку Spring и Reactor (зал 2) или про эксплуатацию Hadoop (зал 4)

DEVEXPERTS

О себе

- Инженер-исследователь в лаборатории dxLab, Devexperts
- Преподаю курс по многопоточному программированию в ИТМО
- Заканчиваю магистратуру ИТМО

DEVEXPERTS

Зачем мне это?

- Кто использует HashMap?

DEVEXPERTS

Зачем мне это?

- Кто использует HashMap?
  - Наверное, это самая популярная и полезная на практике структура данных

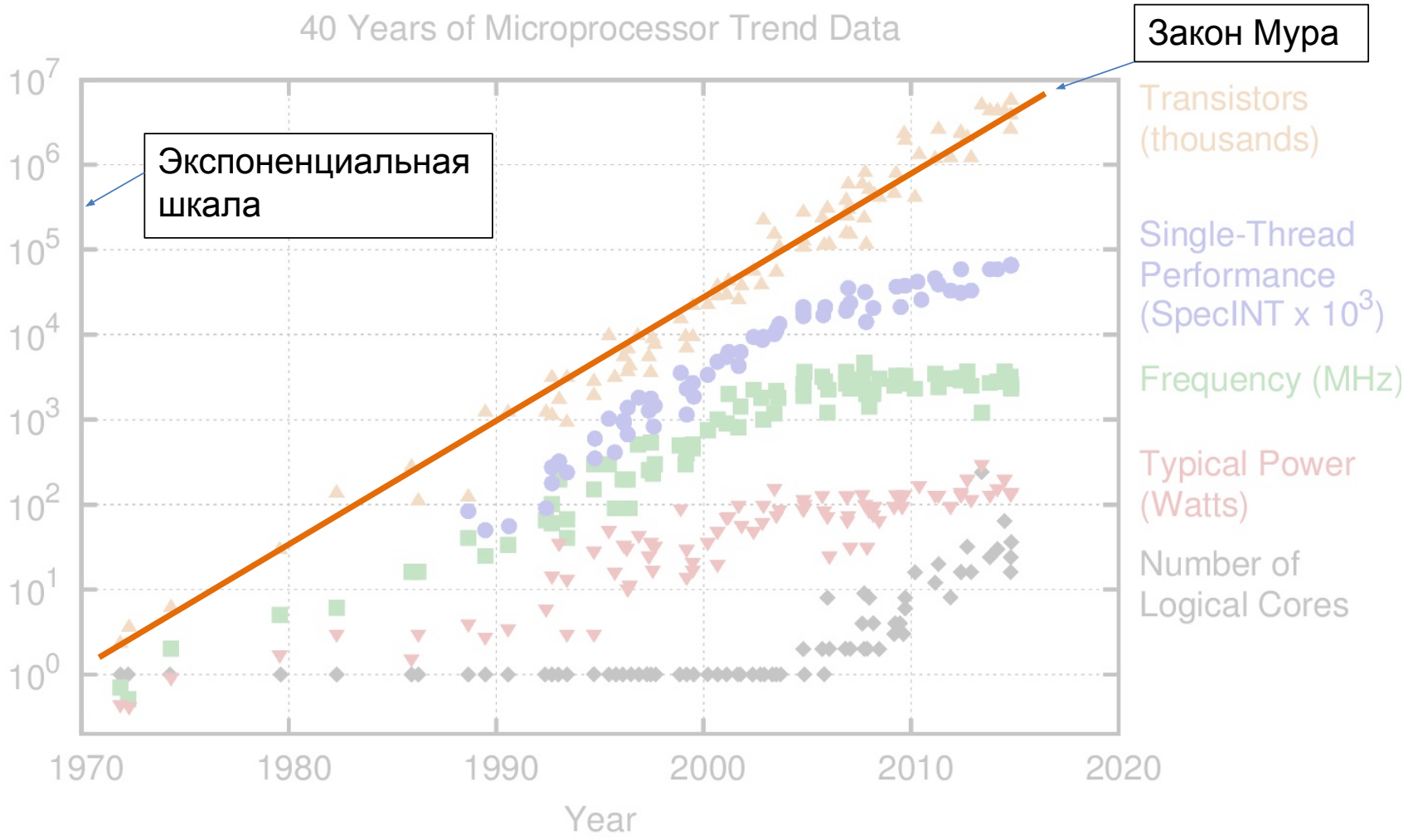
DEVEXPERTS

Зачем мне это?

- Кто использует HashMap?
  - Наверное, это самая популярная и полезная на практике структура данных

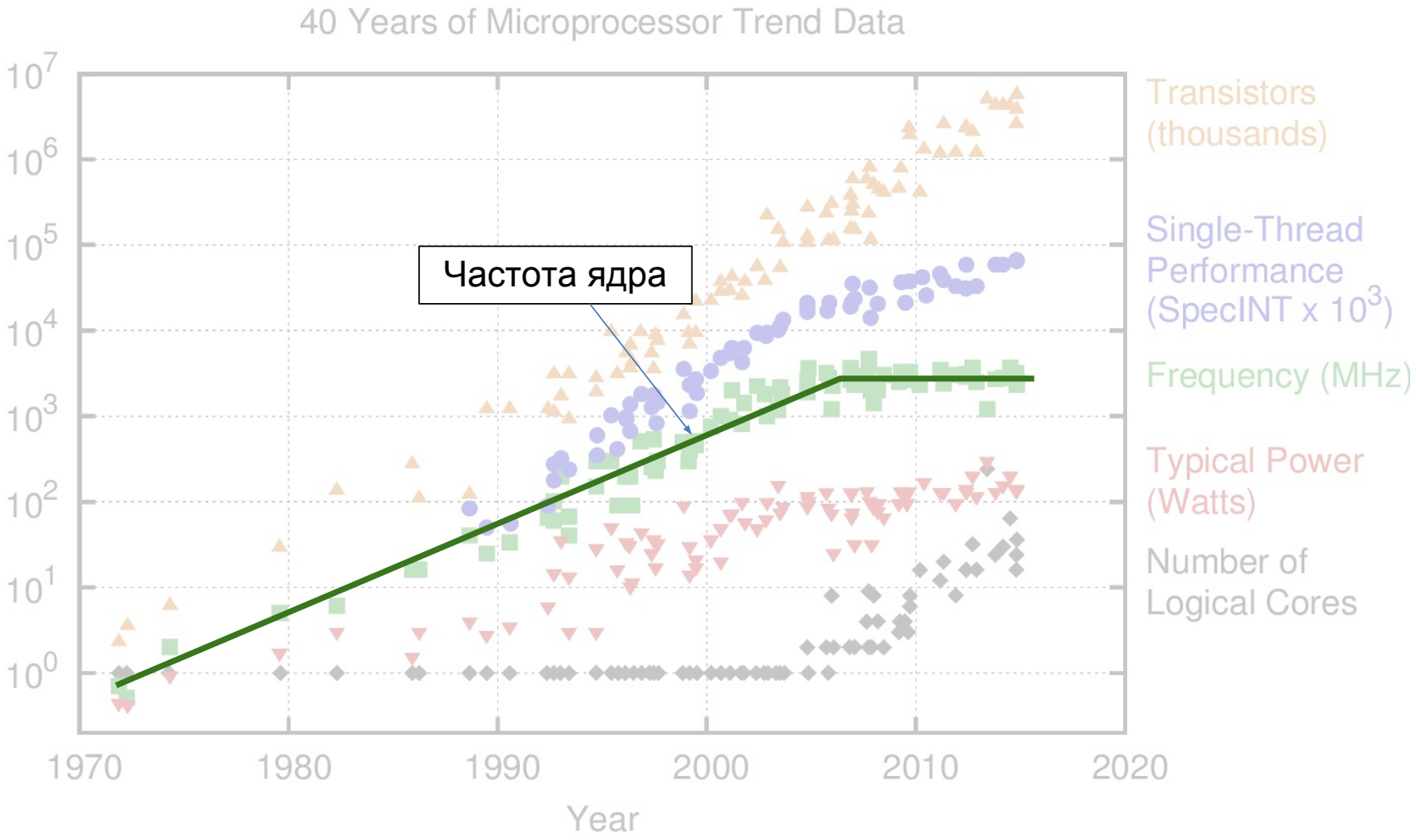
- Но при чём тут многопоточность?

Многопоточность: а надо ли?



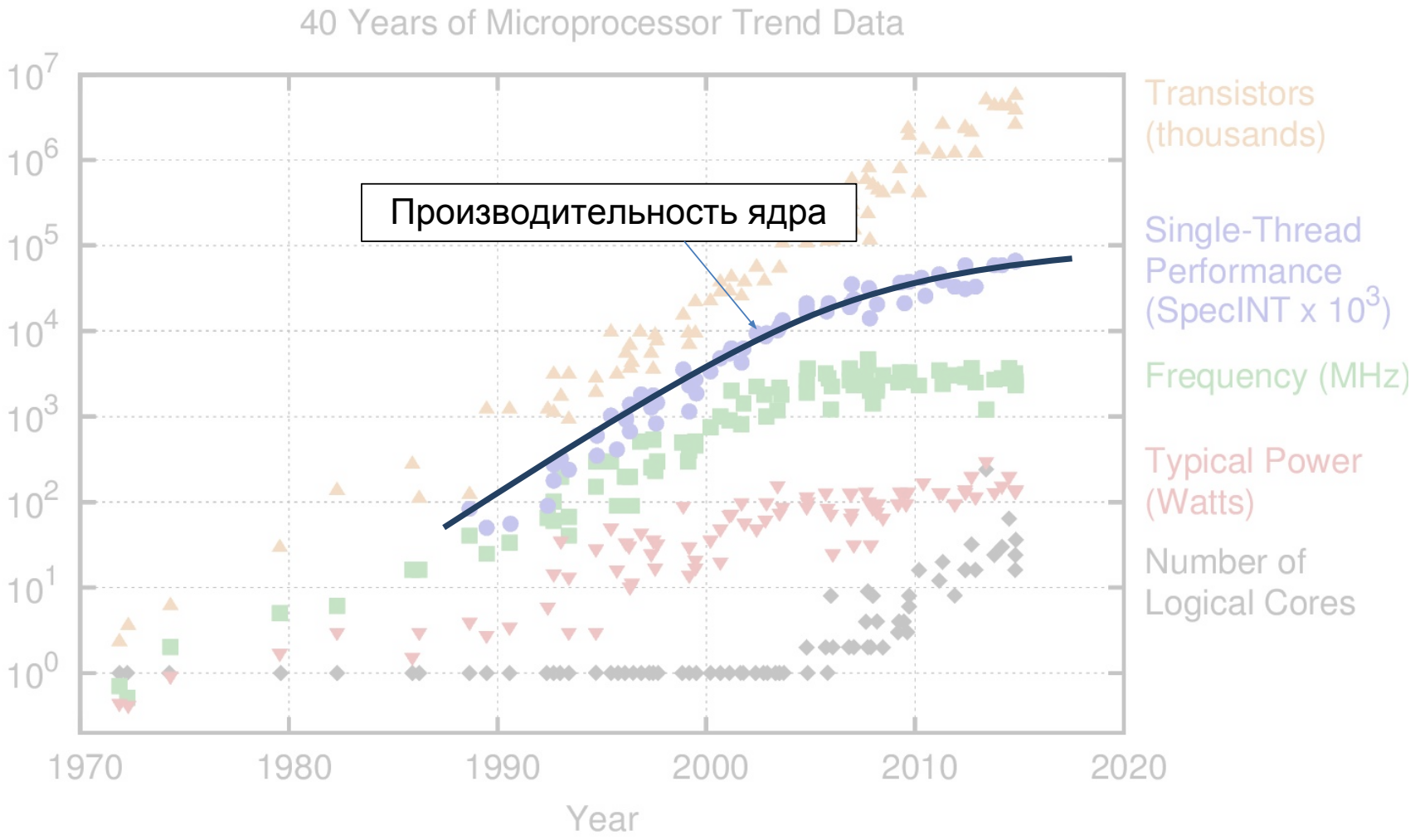
DEVEXPERTS

Многопоточность: а надо ли?



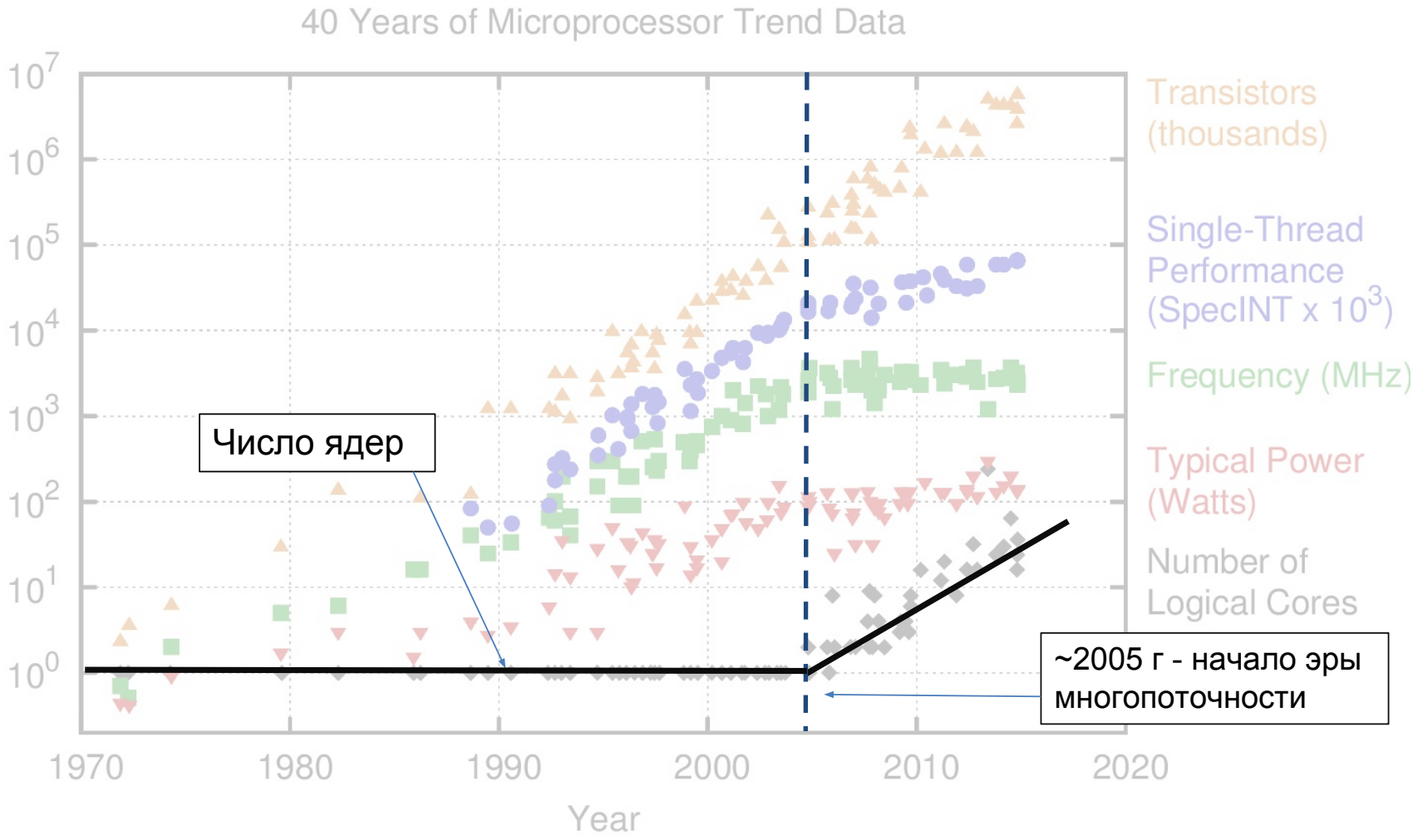
DEVEXPERTS

Многопоточность: а надо ли?



DEVEXPERTS

Многопоточность: а надо ли?



DEVEXPERTS

И что?

- Хеш-таблица – одна из самых популярных структур данных в приложениях
- Для увеличения производительности нужно писать многопоточный код

DEVEXPERTS

И что?

- Хеш-таблица – одна из самых популярных структур данных в приложениях
- Для увеличения производительности нужно писать многопоточный код

Нужна быстрая многопоточная хеш-таблица!

DEVEXPERTS

Модель: кешируем Account-ы

- Часто необходимо сопоставлять id (*long*) с каким-то объектом (*Object*)
  - При этом большинство операций – чтение
  - Время от времени – добавление

Модель: кешируем Account-ы

```
class Account(val id: Int) {
    val i1: Int = 0
    val i2: Int = 0
    val o1: Any? = null
    val o2: Any? = null
    val o3: Any? = null
    val o4: Any? = null
}
```


DEVEXPERTS

Модель: кешируем Account-ы

```
class Account(val id: Int) {
    val i1: Int = 0
    val i2: Int = 0
    val o1: Any? = null
    val o2: Any? = null
    val o3: Any? = null
    val o4: Any? = null
}
```

```
interface AccountsCache {
    fun addAccount(id: Int, account: Account)
    fun getById(id: Int): Account?
}
```


DEVEXPERTS

Как измерять?

Используем JMH:

- Стоимость отдельной операции ничего не значит, т.к. при перехешировании всё замедляется

DEVEXPERTS

Как измерять?

Используем JMH:

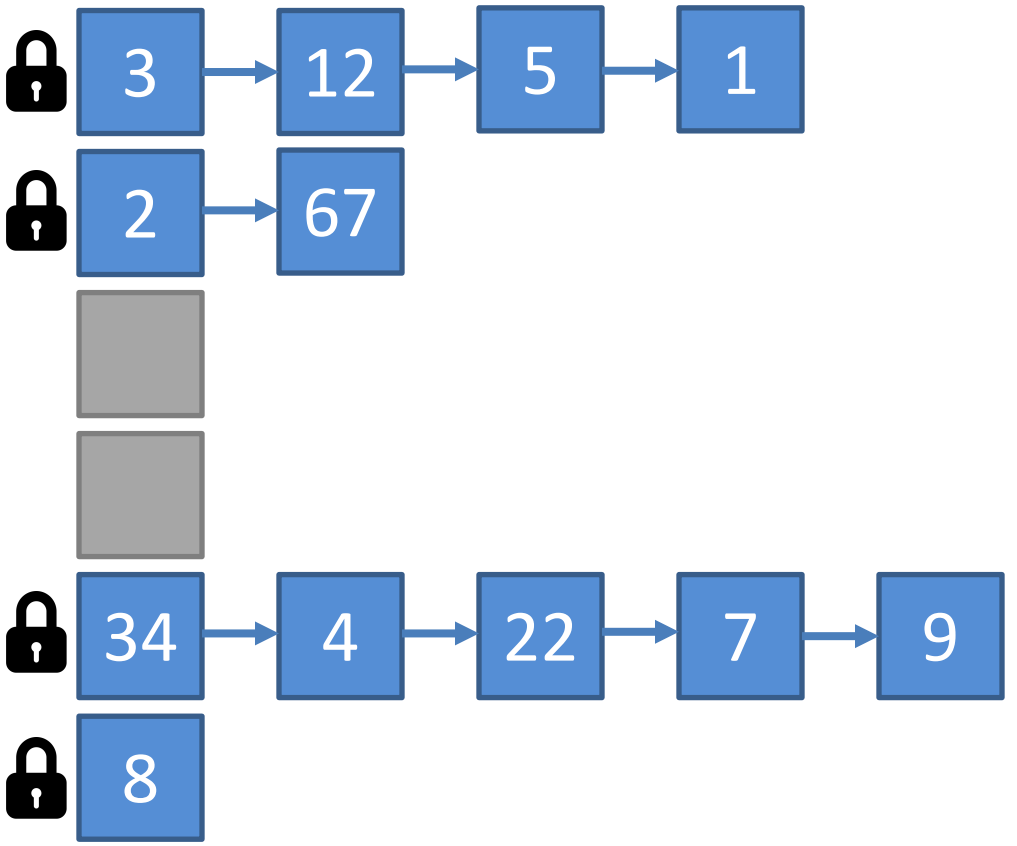
- Стоимость отдельной операции ничего не значит, т.к. при перехешировании всё замедляется
- ⇒ имеет смысл считать стоимость пачки операций

```
@Measurement(batchSize = 1000_000)
```


DEVEXPERTS

Что есть в Java: ConcurrentHashMap

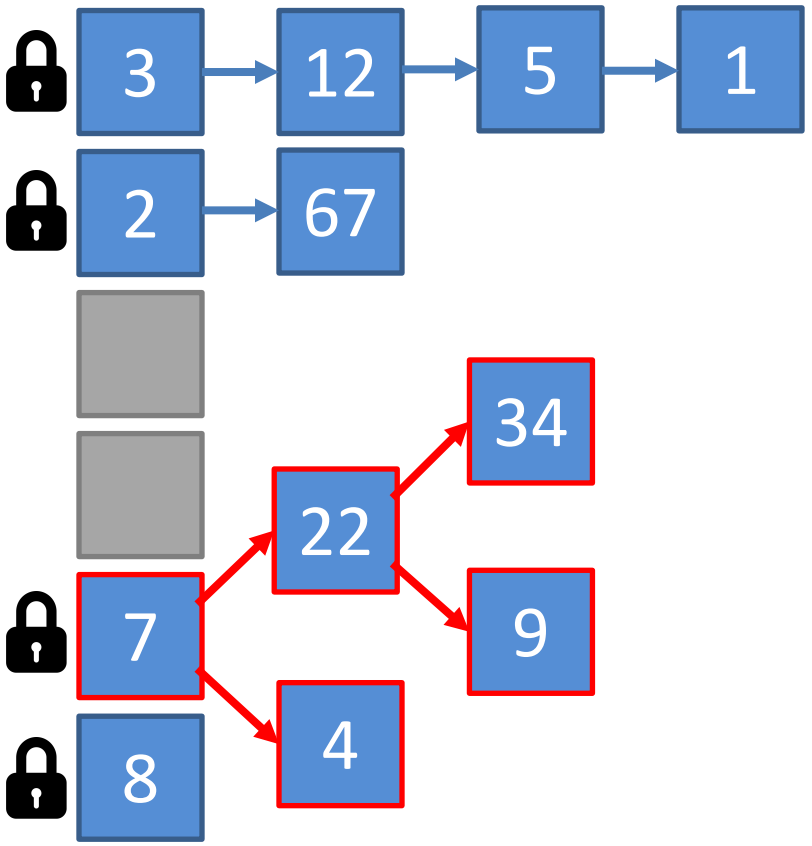
- Работает за O(1) в среднем
- Использует внутри блокировки
  - ⇒ не так хорошо масштабируется
- Хранит списки при коллизии
  - ⇒ лишние cache miss-ы



DEVEXPERTS

Что есть в Java: ConcurrentHashMap

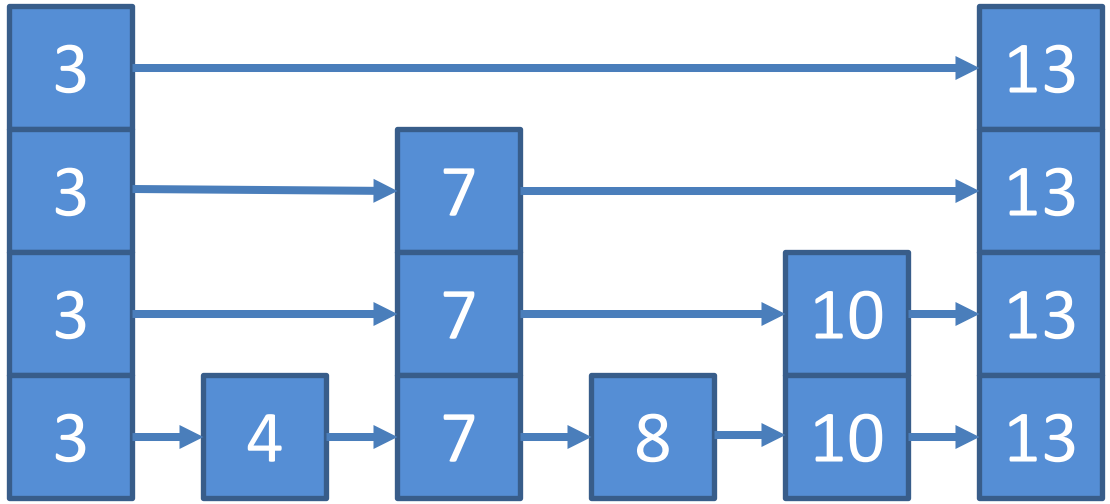
- Работает за O(1) в среднем
- Использует внутри блокировки
  - ⇒ не так хорошо масштабируется
- Хранит списки при коллизии
  - ⇒ лишние cache miss-ы
  - зато может хранить дерево при постоянных коллизиях*



DEVEXPERTS

Что ещё есть в Java: ConcurrentSkipListMap

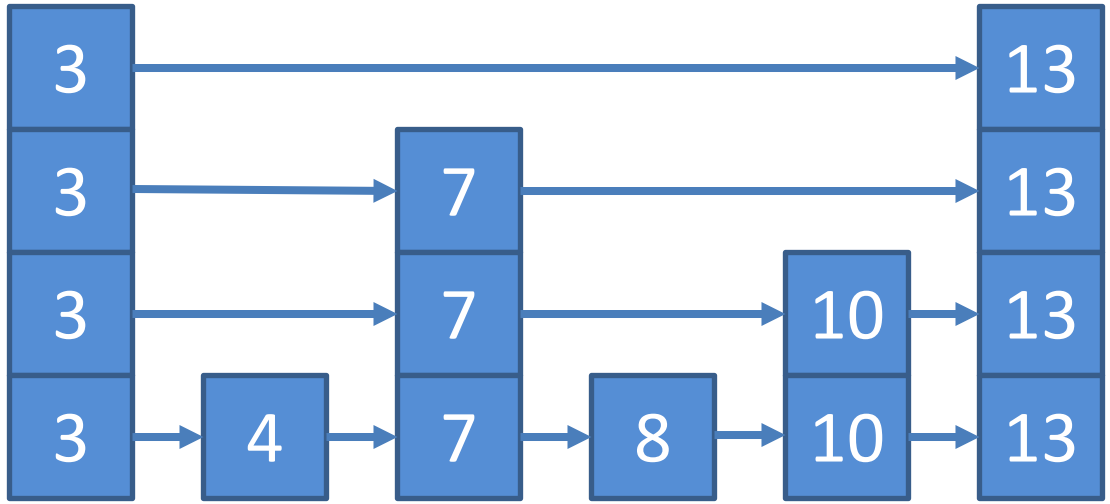
- Работает за `O(log N)` в среднем
- «Дерево» списков



DEVEXPERTS

Что ещё есть в Java: ConcurrentSkipListMap

- Работает за `O(log N)` в среднем
- «Дерево» списков

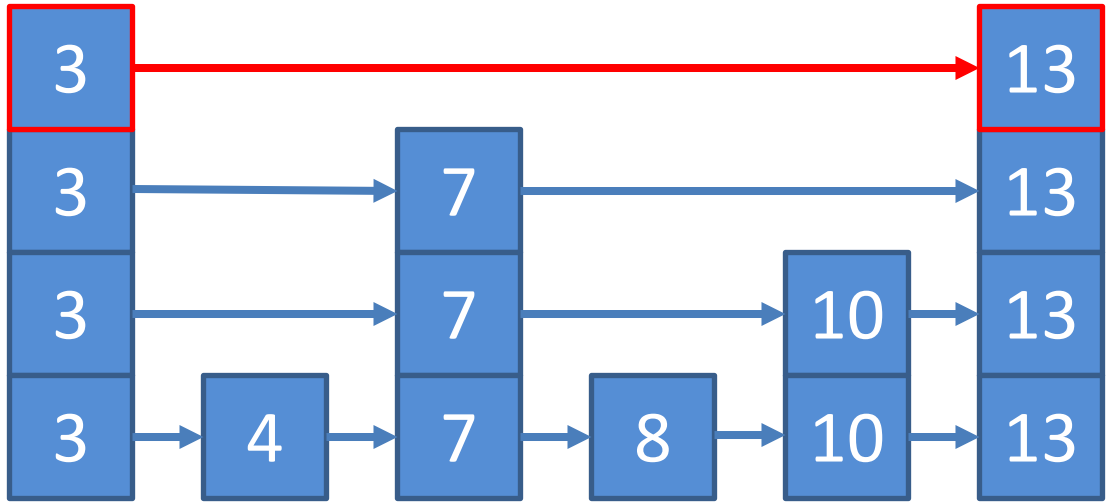


Ищем ключ «8»

DEVEXPERTS

Что ещё есть в Java: ConcurrentSkipListMap

- Работает за `O(log N)` в среднем
- «Дерево» списков

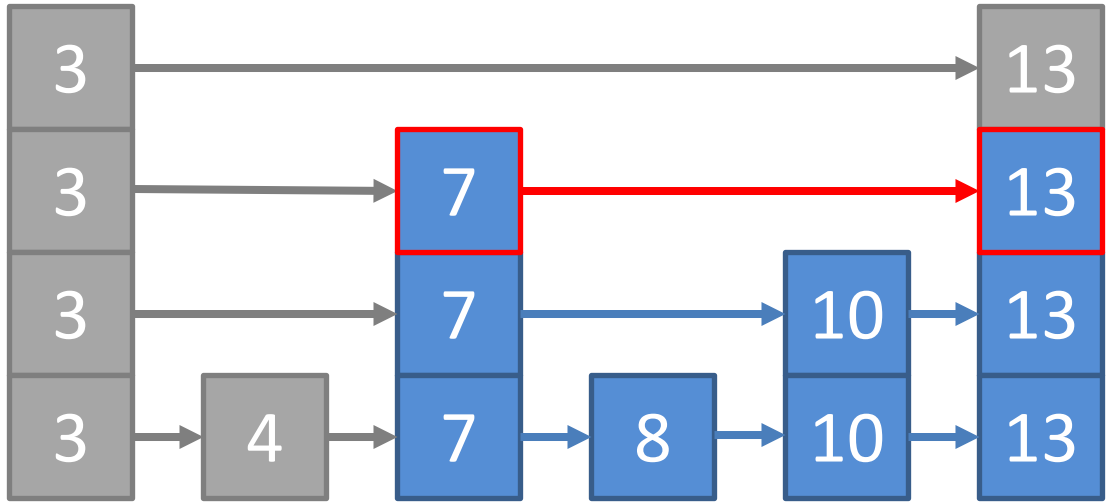


Ищем ключ «8»

DEVEXPERTS

Что ещё есть в Java: ConcurrentSkipListMap

- Работает за `O(log N)` в среднем
- «Дерево» списков

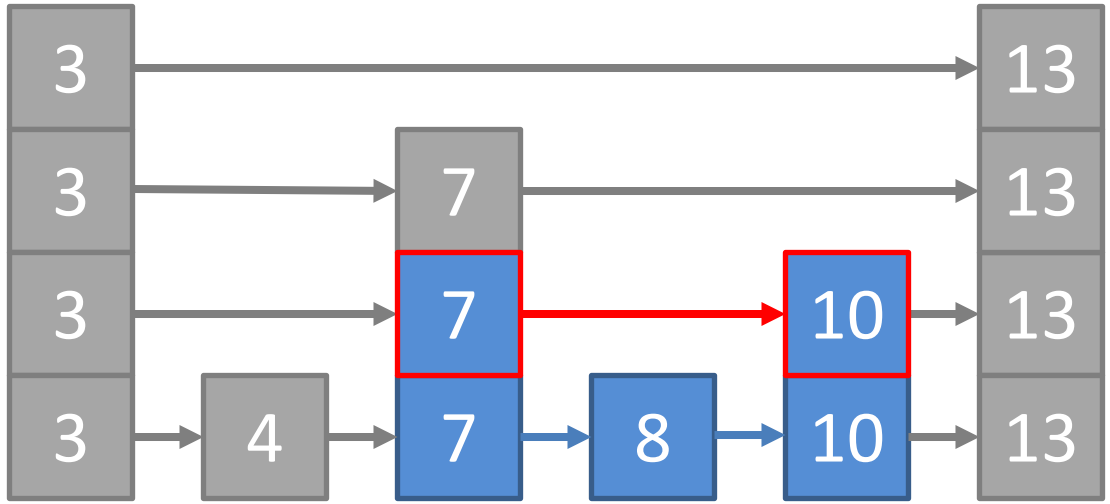


Ищем ключ «8»

DEVEXPERTS

Что ещё есть в Java: ConcurrentSkipListMap

- Работает за `O(log N)` в среднем
- «Дерево» списков

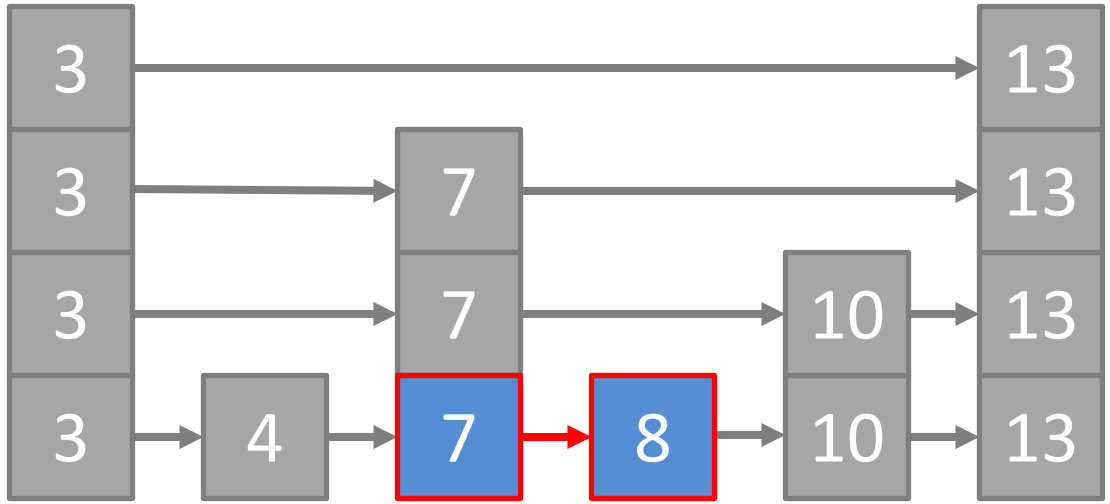


Ищем ключ «8»

DEVEXPERTS

Что ещё есть в Java: ConcurrentSkipListMap

- Работает за `O(log N)` в среднем
- «Дерево» списков

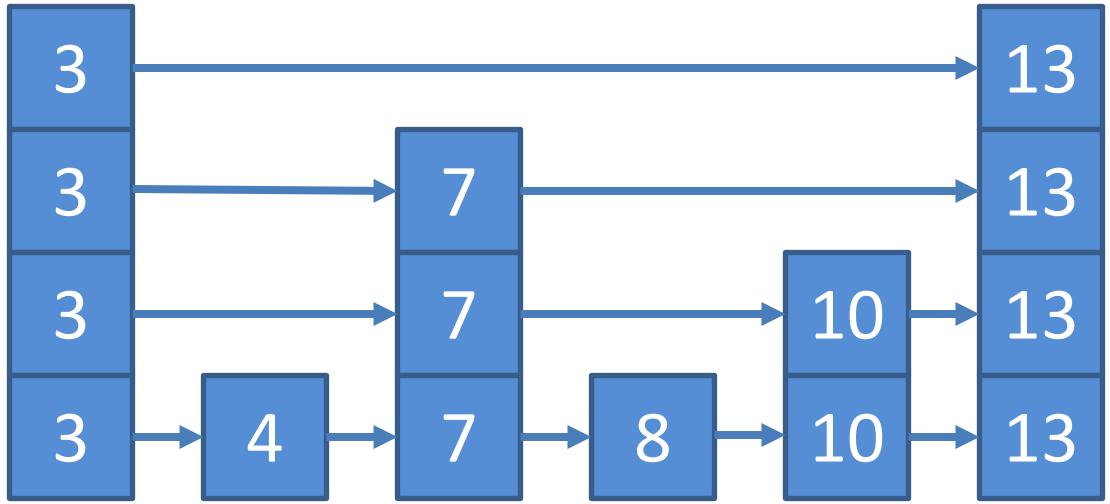


Ищем ключ «8»

DEVEXPERTS

Что ещё есть в Java: ConcurrentSkipListMap

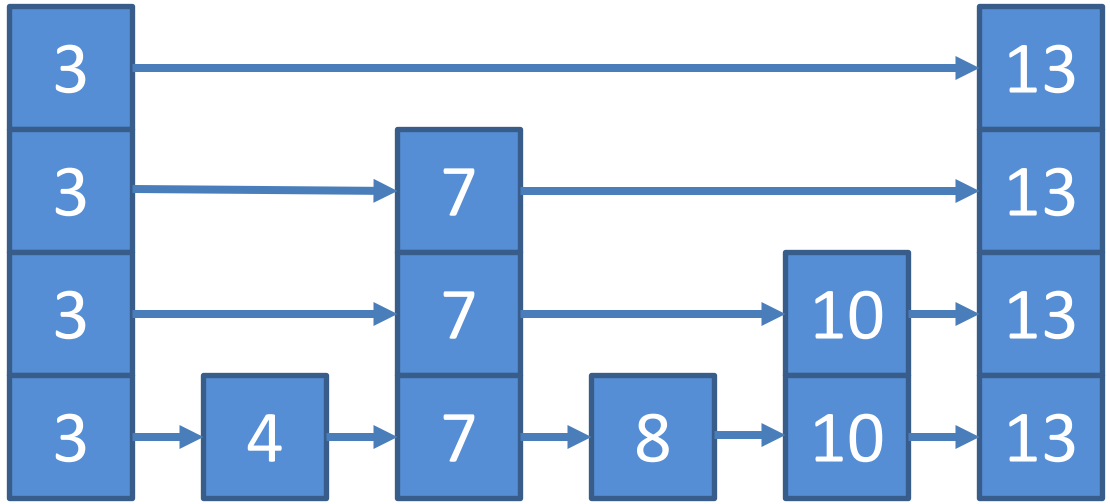
- Работает за `O(log N)` в среднем
- «Дерево» списков
  - Много лишних объектов
  - ⇒ постоянные cache miss-ы и нагружает GC



DEVEXPERTS

Что ещё есть в Java: ConcurrentSkipListMap

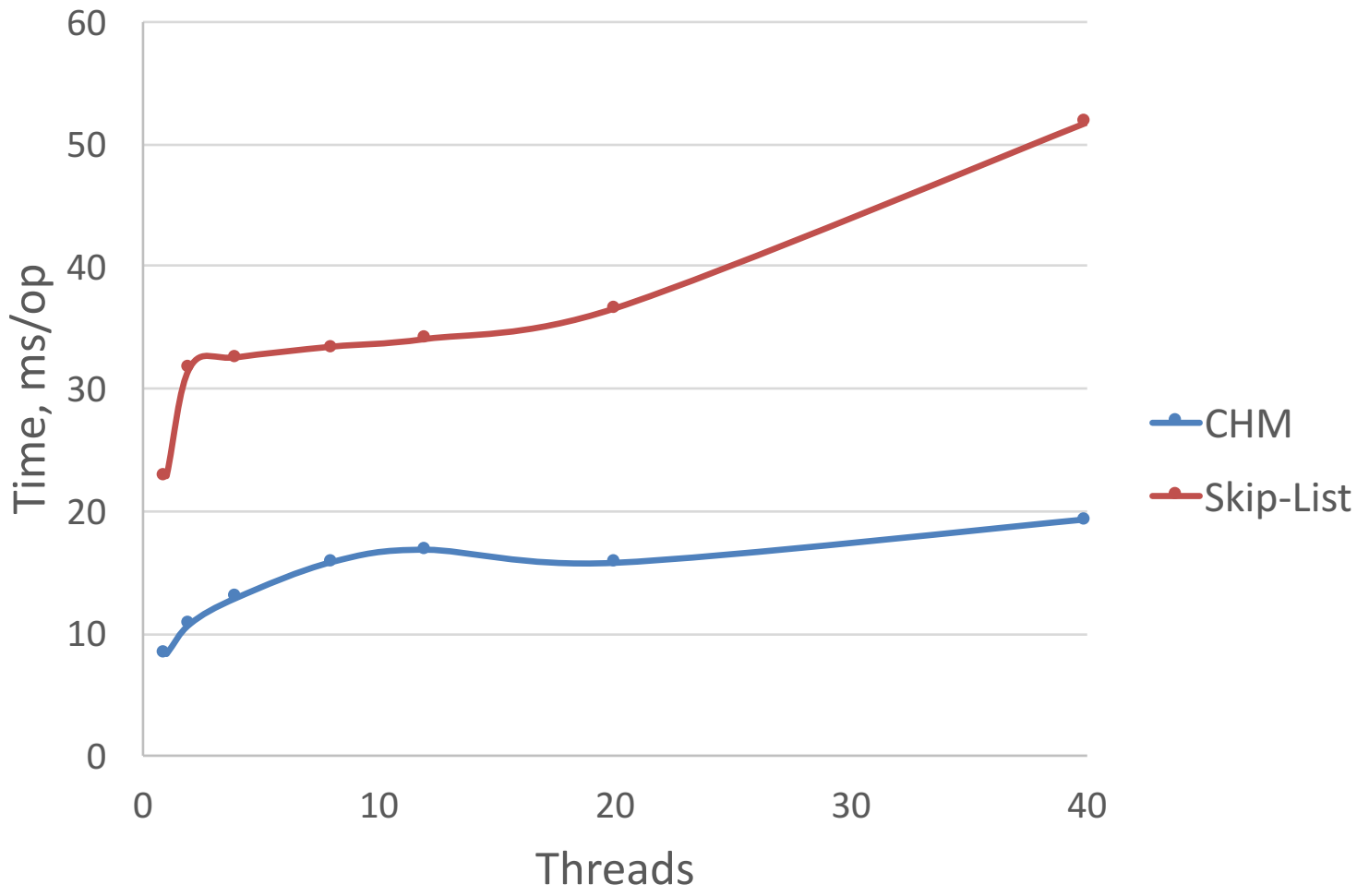
- Работает за `O(log N)` в среднем
- «Дерево» списков
  - Много лишних объектов
  - ⇒ постоянные cache miss-ы и нагружает GC



- И, вообще, это не хеш-таблица!

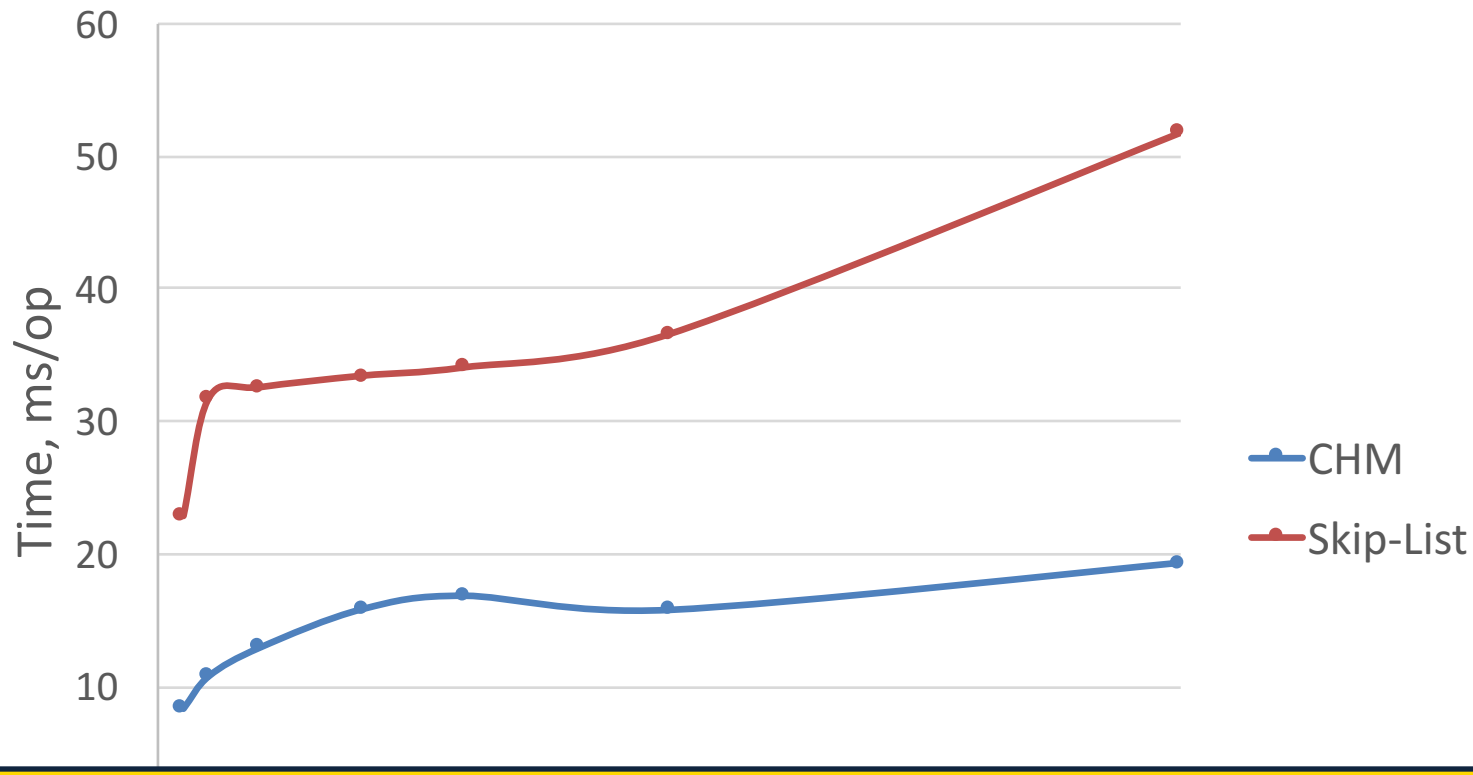
DEVEXPERTS

ConcurrentHashMap vs ConcurrentSkipListMap



DEVEXPERTS

ConcurrentHashMap vs ConcurrentSkipListMap



Даёшь больше производительности!

DEVEXPERTS

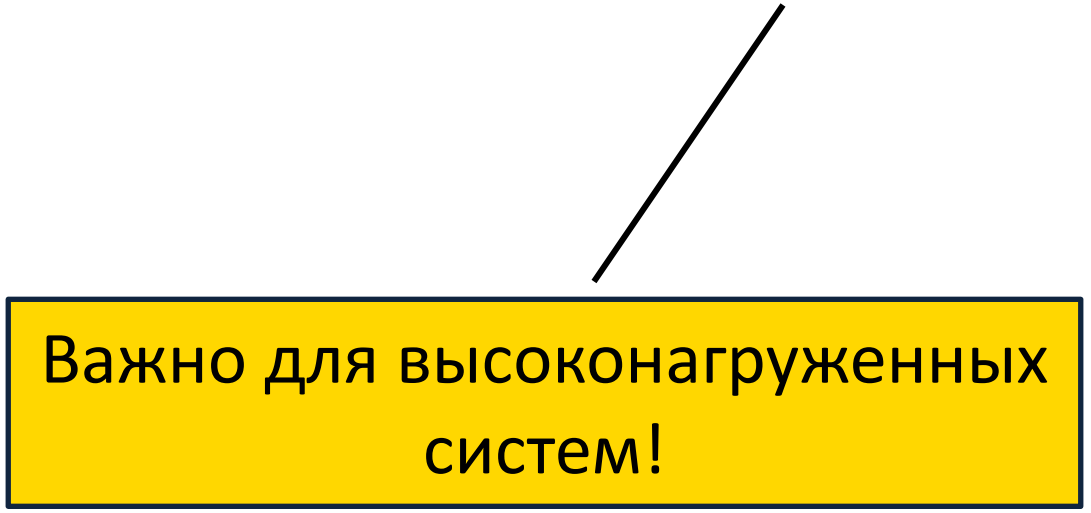
Потенциальные проблемы

- ConcurrentHashMap
  - Использует внутри блокировки ⇒ не lock-free

DEVEXPERTS

Потенциальные проблемы

- ConcurrentHashMap
  - Использует внутри блокировки ⇒ не lock-free



DEVEXPERTS

Потенциальные проблемы

- ConcurrentHashMap
  - Использует внутри блокировки ⇒ не lock-free
  - Использует списки ⇒ cache miss-ы

DEVEXPERTS

Потенциальные проблемы

- ConcurrentHashMap
  - Использует внутри блокировки ⇒ не lock-free
  - Использует списки ⇒ cache miss-ы
- ConcurrentSkipListMap
  - Много «лишних» объектов
  - Постоянные cache miss-ы

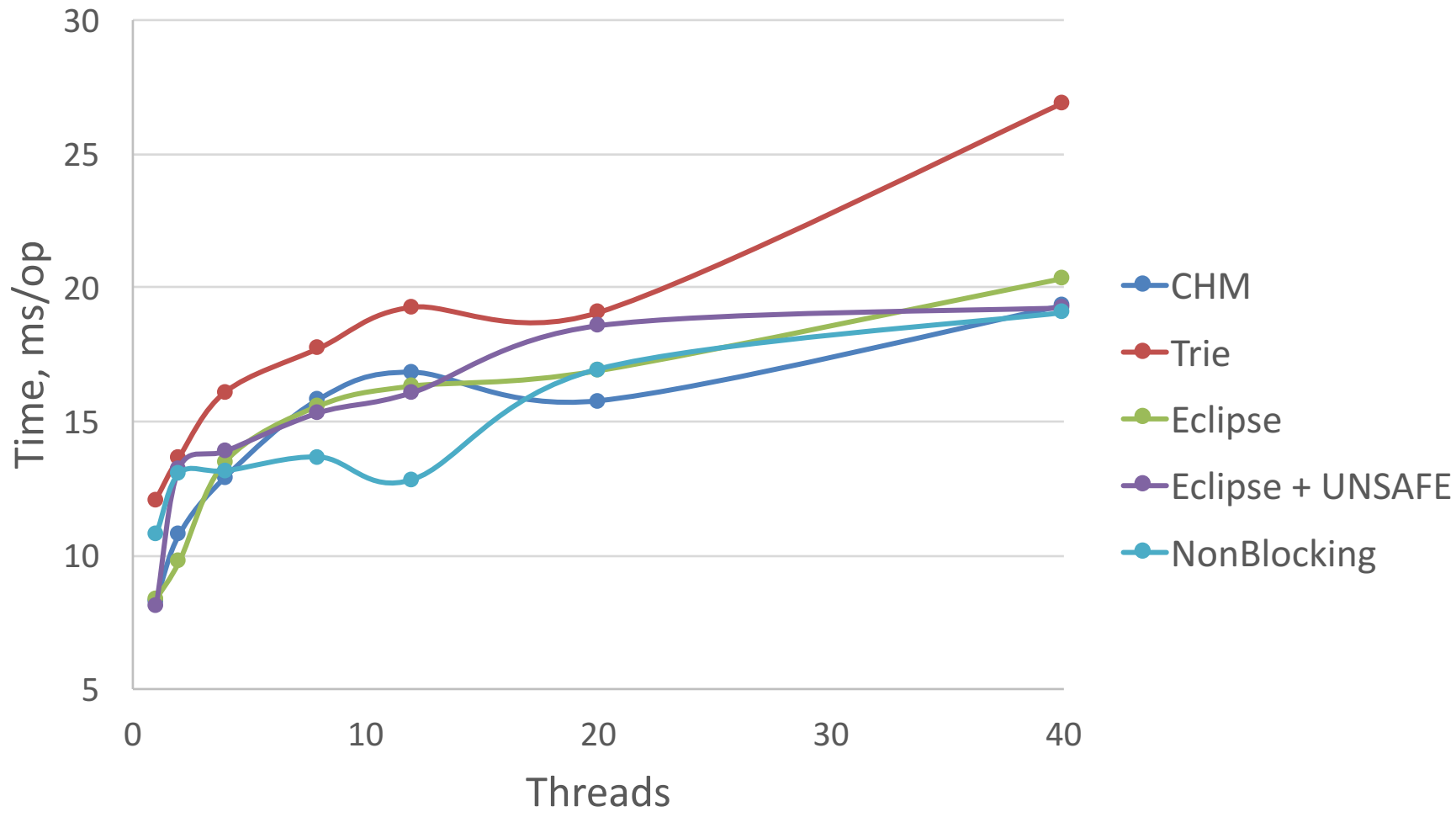
DEVEXPERTS

Альтернативы

- https://github.com/edwardw/high-scale-java-lib
  - `NonBlockingHashMap`
- https://github.com/romix/java-concurrent-hash-trie-map
- https://github.com/eclipse/eclipse-collections
  - `ConcurrentHashMap` и `ConcurrentHashMapUnsafe`
- https://github.com/google/guava
- https://github.com/javolution/javolution
- …

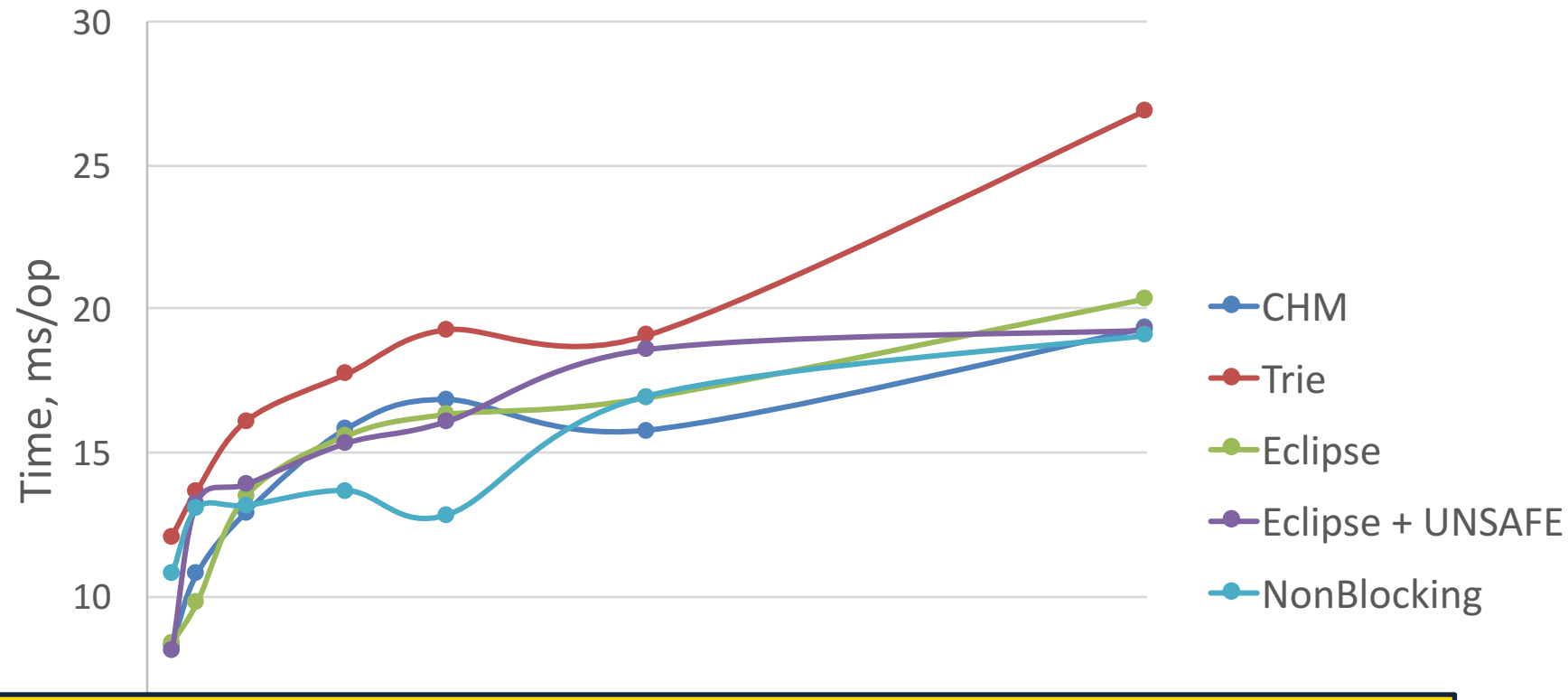
DEVEXPERTS

Альтернативы: бенчмарки



DEVEXPERTS

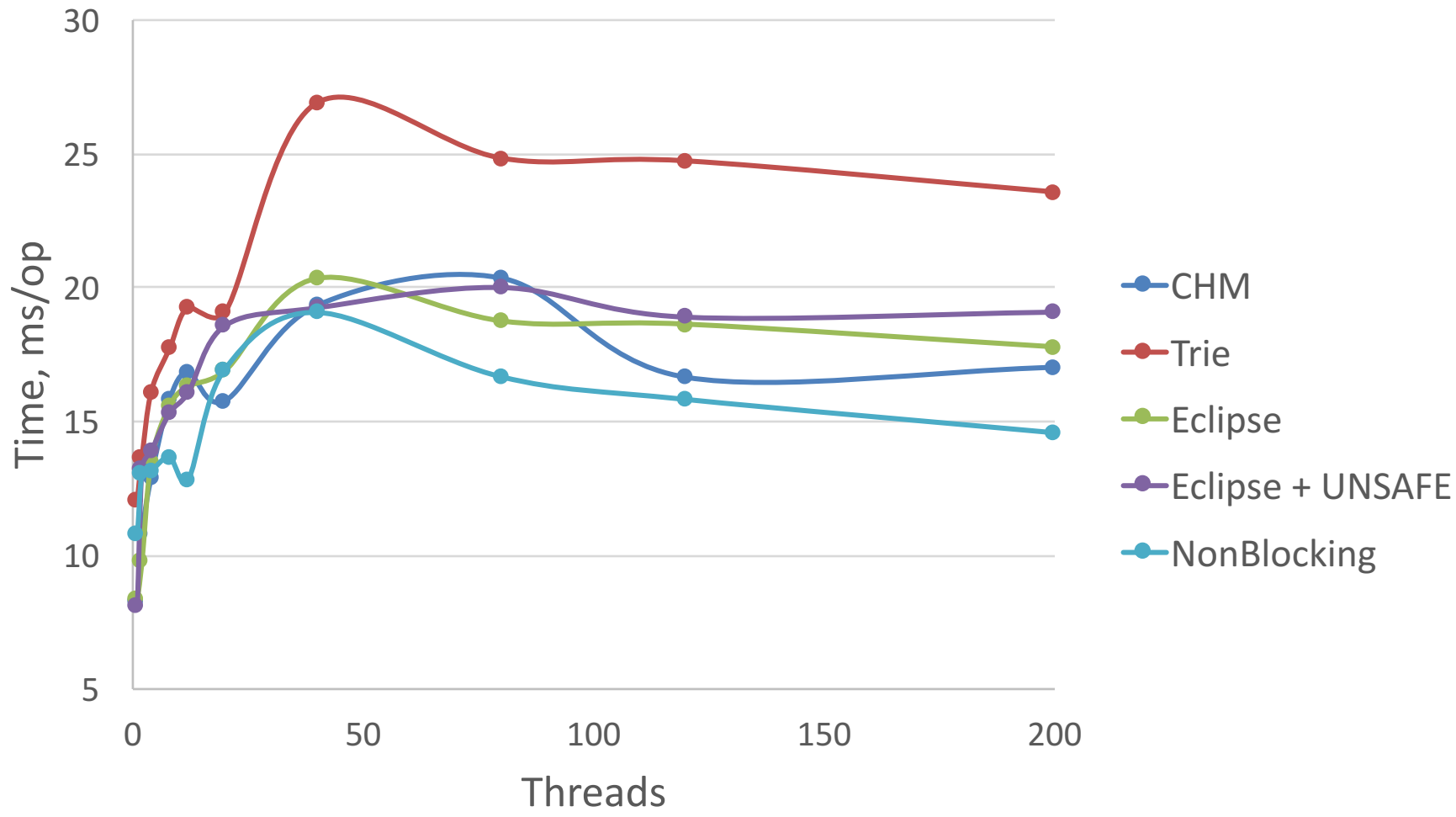
Альтернативы: бенчмарки



А если потоков больше, чем ядер?

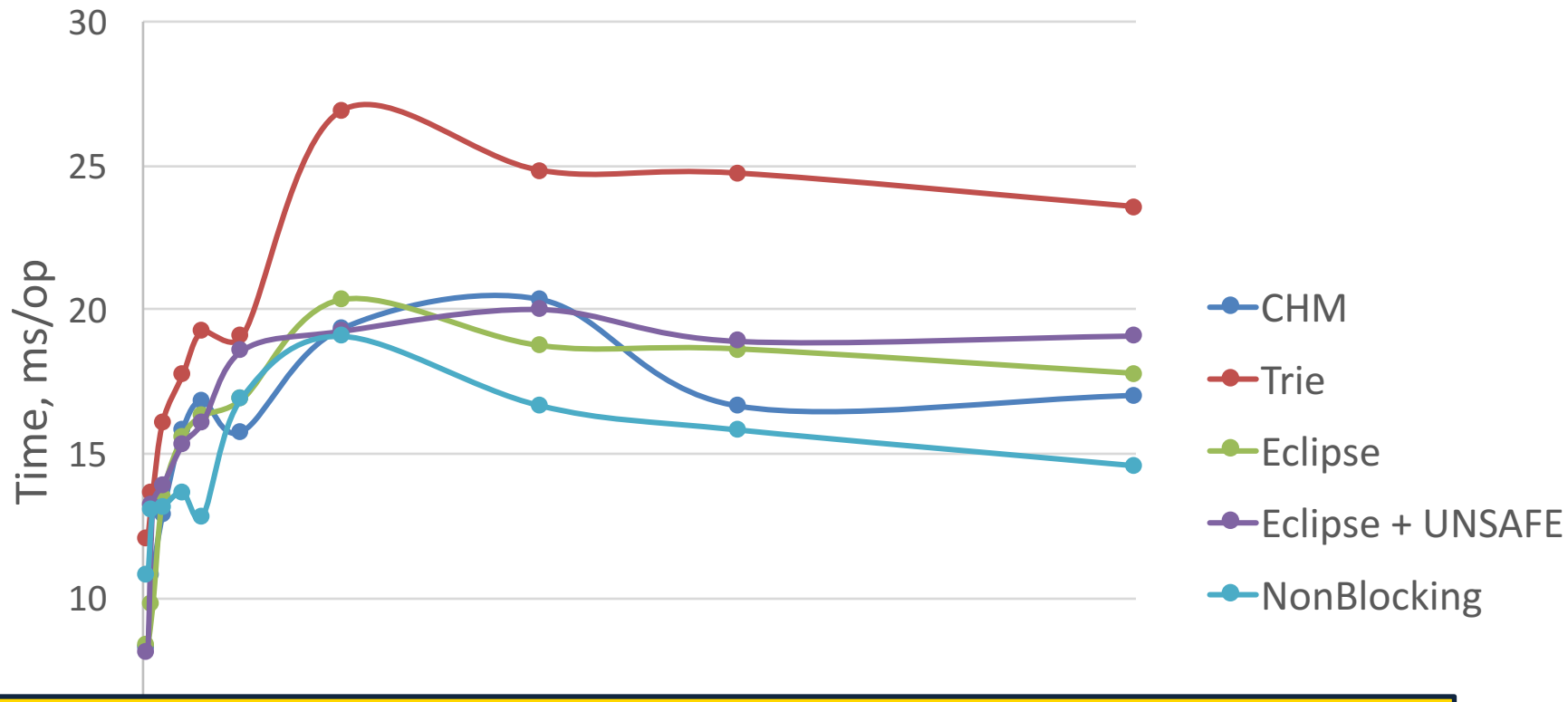
DEVEXPERTS

Альтернативы: бенчмарки



DEVEXPERTS

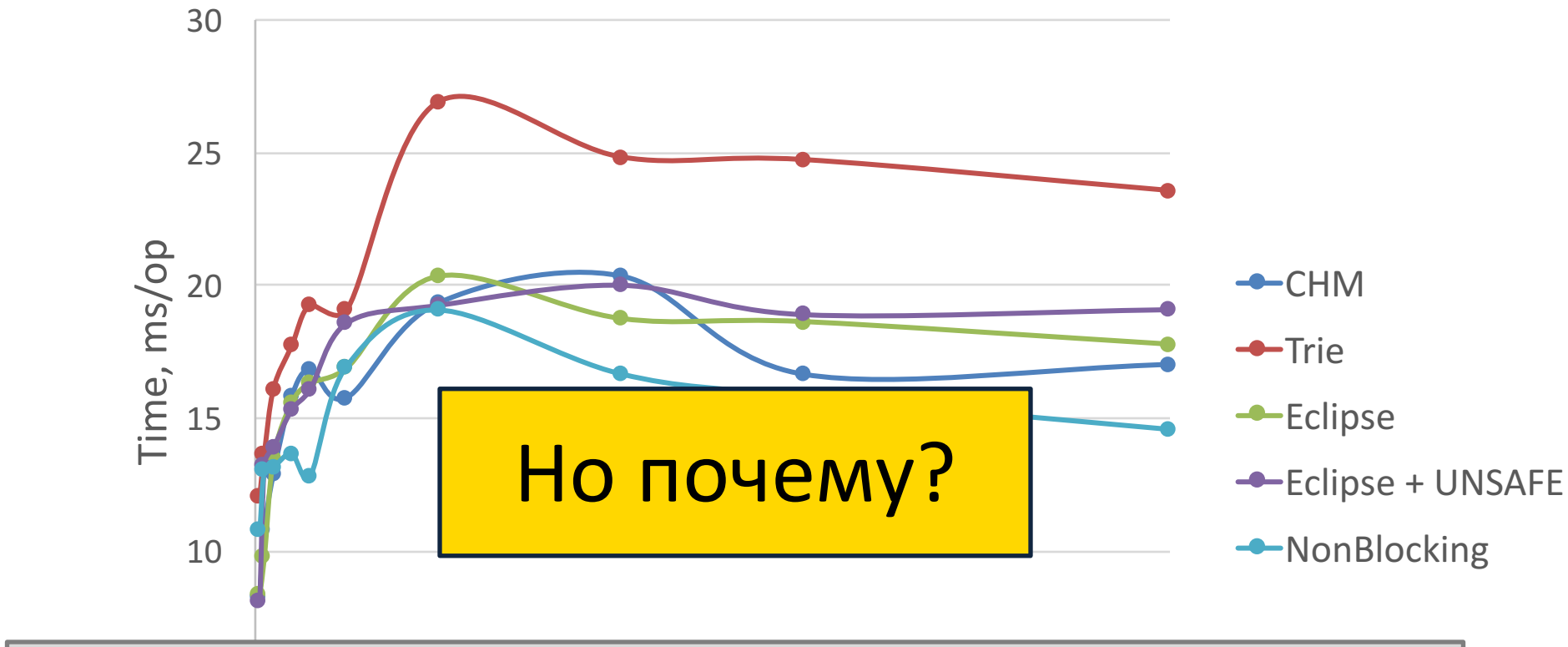
Альтернативы: бенчмарки



NonBlocking чуть-чуть всех рвёт

DEVEXPERTS

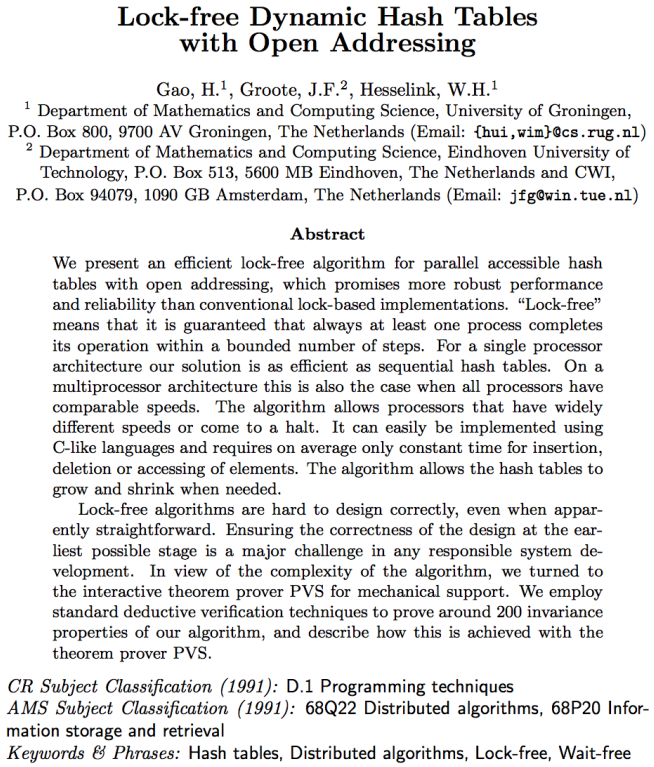
Альтернативы: бенчмарки



NonBlocking чуть-чуть всех рвёт

DEVEXPERTS

NonBlockingHashMap



Lock-free Dynamic Hash Tables with Open Addressing

Gao, H.[1], Groote, J.F.[2], Hesselink, W.H.[1]
[1] Department of Mathematics and Computing Science, University of Groningen, P.O. Box 800, 9700 AV Groningen, The Netherlands (Email: {hui,wim}@cs.rug.nl)
[2] Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands and CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands (Email: jfg@win.tue.nl)

Abstract

We present an efficient lock-free algorithm for parallel accessible hash tables with open addressing, which promises more robust performance and reliability than conventional lock-based implementations. "Lock-free" means that it is guaranteed that always at least one process completes its operation within a bounded number of steps. For a single processor architecture our solution is as efficient as sequential hash tables. On a multiprocessor architecture this is also the case when all processors have comparable speeds. The algorithm allows processors that have widely different speeds or come to a halt. It can easily be implemented using C-like languages and requires on average only constant time for insertion, deletion or accessing of elements. The algorithm allows the hash tables to grow and shrink when needed.

Lock-free algorithms are hard to design correctly, even when apparently straightforward. Ensuring the correctness of the design at the earliest possible stage is a major challenge in any responsible system development. In view of the complexity of the algorithm, we turned to the interactive theorem prover PVS for mechanical support. We employ standard deductive verification techniques to prove around 200 invariance properties of our algorithm, and describe how this is achieved with the theorem prover PVS.

CR Subject Classification (1991): D.1 Programming techniques
AMS Subject Classification (1991): 68Q22 Distributed algorithms, 68P20 Information storage and retrieval
Keywords & Phrases: Hash tables, Distributed algorithms, Lock-free, Wait-free

Lock-free Dynamic Hash Tables with Open Addressing by Gao et al.

A Lock-Free Wait-Free Hash Table by C. Click

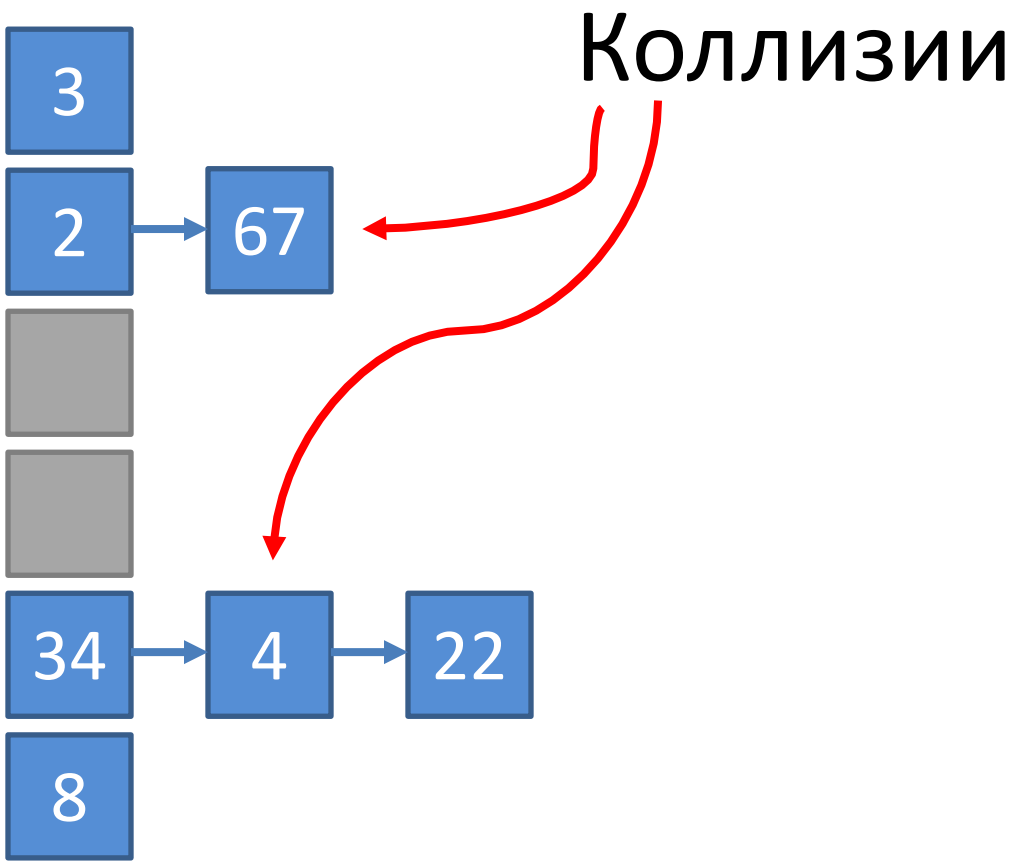
DEVEXPERTS

NonBlockingHashMap

- Использует открытую адресацию

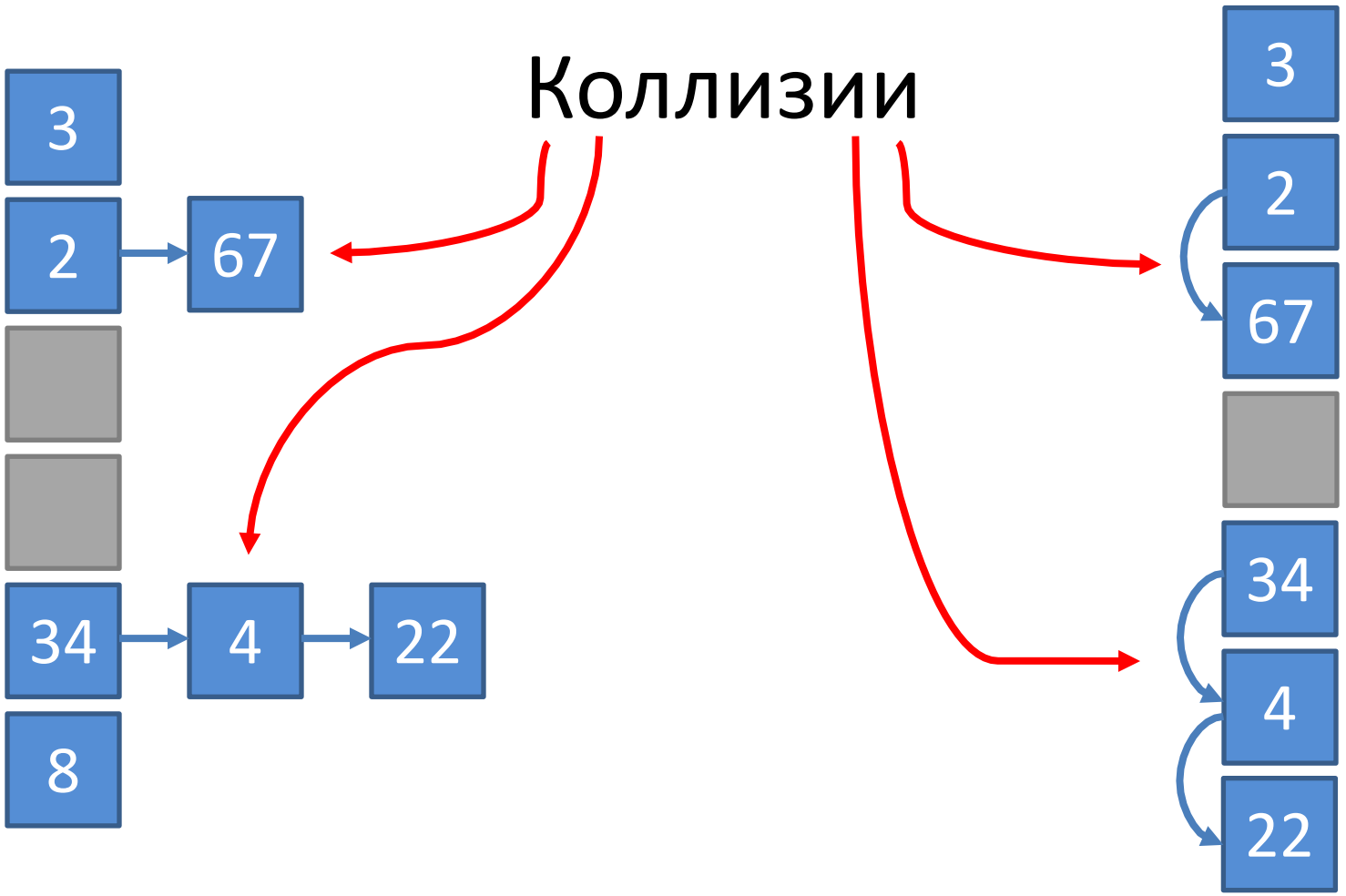
DEVEXPERTS

Открытая и закрытая адресации



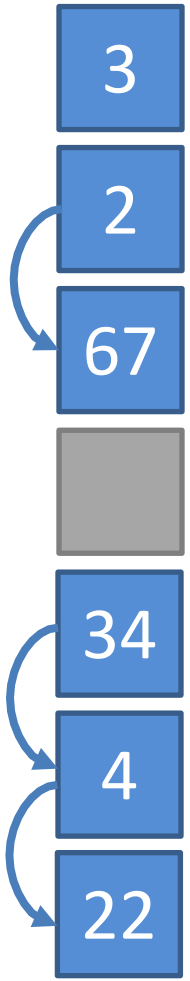
DEVEXPERTS

Открытая и закрытая адресации



DEVEXPERTS

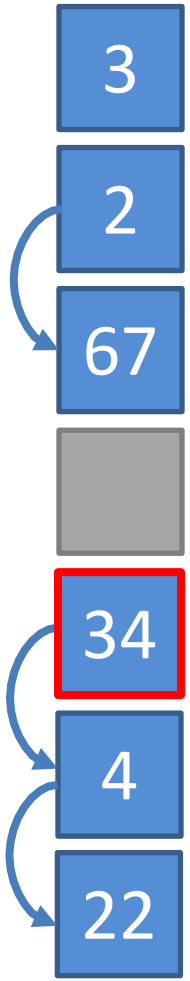
Открытая адресация: поиск



Ищем «22»

DEVEXPERTS

Открытая адресация: поиск

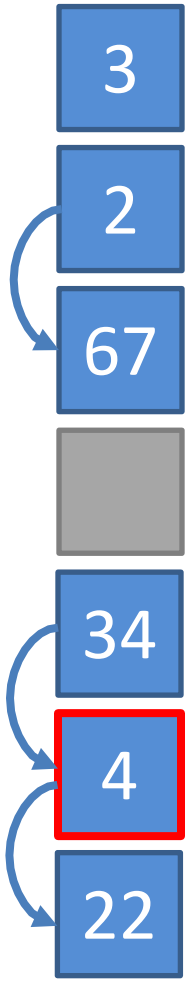


Ищем «22»:

1. Пришли в ячейку с «34»

DEVEXPERTS

Открытая адресация: поиск

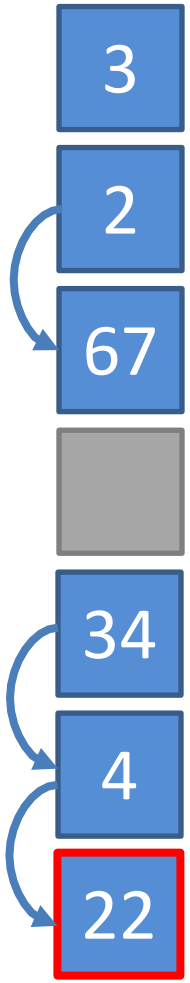


Ищем «22»:

1. Пришли в ячейку с «34»
2. Идём дальше пока не найдём «22»

DEVEXPERTS

Открытая адресация: поиск



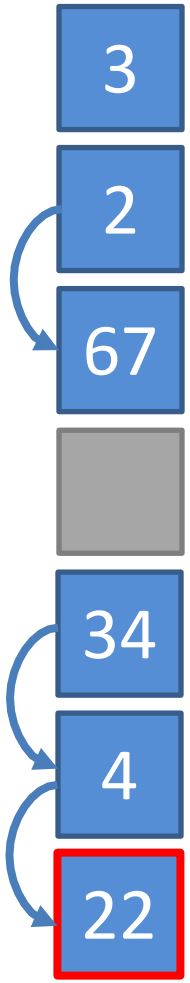
Ищем «22»:

1. Пришли в ячейку с «34»
2. Идём дальше пока не найдём «22»

Ура, нашли!

DEVEXPERTS

Открытая адресация: поиск



Ищем «22»:

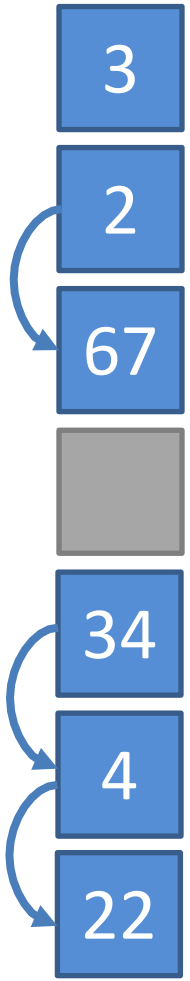
1. Пришли в ячейку с «34»
2. Идём дальше пока не найдём «22»

Ура, нашли!

А если бы «4» была удалена?

DEVEXPERTS

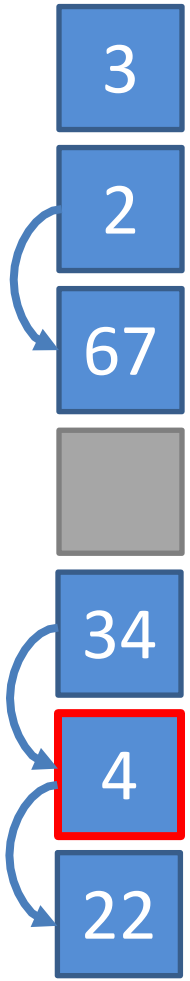
Открытая адресация: удаление



Удаляем «4»

DEVEXPERTS

Открытая адресация: удаление

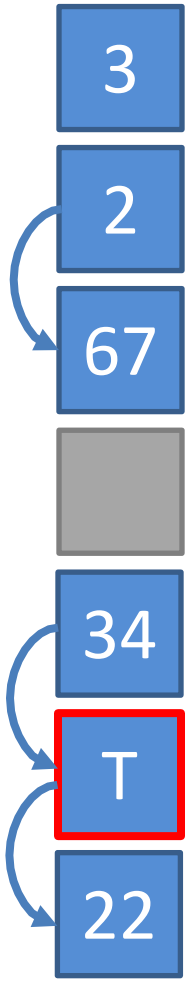


Удаляем «4»:

1. Находим ячейку

DEVEXPERTS

Открытая адресация: удаление

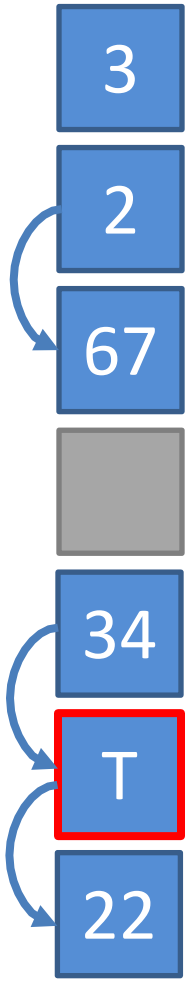


Удаляем «4»:

1. Находим ячейку
2. Меняем на специальное значение «раньше тут был элемент»

DEVEXPERTS

Открытая адресация: удаление



Удаляем «4»:

1. Находим ячейку
2. Меняем на специальное значение «раньше тут был элемент»

Теперь мы сможем найти «22»!

DEVEXPERTS

Пара деталей: размер таблицы

- Как выбрать правильный размер таблицы?
- Теория говорит «давайте возьмём простое число»

Пара деталей: размер таблицы

- Как выбрать правильный размер таблицы?
- Теория говорит «давайте возьмём простое число»
  - Тогда нужно всегда делать `hash(key) % N`

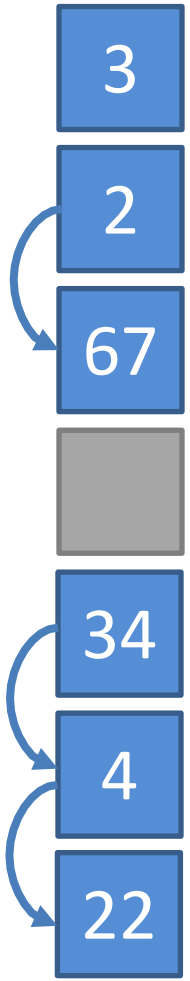
DEVEXPERTS

Пара деталей: размер таблицы

- Как выбрать правильный размер таблицы?
- Теория говорит «давайте возьмём простое число»
  - Тогда нужно всегда делать `hash(key) % N`
  - Операция взятия по модулю слишком дорогая
  - На практике степень двойки лучше (это просто сдвиг)

DEVEXPERTS

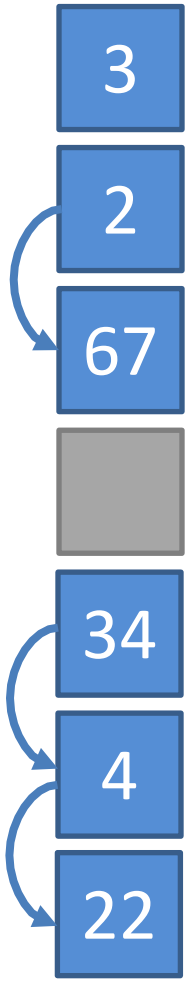
Пара деталей: поиск элемента



- Можно смотреть не на следующий элемент
  - И в теории это лучше!

DEVEXPERTS

Пара деталей: поиск элемента



- Можно смотреть не на следующий элемент
  - И в теории это лучше!
- На практике посмотреть на следующий элемент дешевле, т.к. он скорее всего уже закеширован

DEVEXPERTS

Открытая адресация: get(key)

```
public T getInternal(long key) {
    int i = index(key);
    long k;
    int probes = 0;
    while ((k = keys.get(i)) != key) {
        if (k == NULL_KEY)
            return null;
        if (++probes >= MAX_PROBES)
            return null;
        if (i == 0)
            i = length;
        i--;
    }
    return values.get(i);
}
```


DEVEXPERTS

Открытая адресация: get(key)

```
public T getInternal(long key) {
    int i = index(key);
    long k;
    int probes = 0;
    while ((k = keys.get(i)) != key) {
        if (k == NULL_KEY)
            return null;
        if (++probes >= MAX_PROBES)
            return null;
        if (i == 0)
            i = length;
        i--;
    }
    return values.get(i);
}
```


DEVEXPERTS

Открытая адресация: get(key)

```
public T getInternal(long key) {
    int i = index(key);
    long k;
    int probes = 0;
    while ((k = keys.get(i)) != key) {
        if (k == NULL_KEY)
            return null;
        if (++probes >= MAX_PROBES)
            return null;
        if (i == 0)
            i = length;
        i--;
    }
    return values.get(i);
}
```


DEVEXPERTS

Открытая адресация: get(key)

```
public T getInternal(long key) {
    int i = index(key);
    long k;
    int probes = 0;
    while ((k = keys.get(i)) != key) {
        if (k == NULL_KEY)
            return null;
        if (++probes >= MAX_PROBES)
            return null;
        if (i == 0)
            i = length;
        i--;
    }
    return values.get(i);
}
```


DEVEXPERTS

NonBlockingHashMap

- Использует открытую адресацию
- Lock-free
  - Гарантирует, что система не стоит на месте даже при неудачном scheduling-е
  - Можно вставлять/читать во время перехеширования

NonBlockingHashMap

- Использует открытую адресацию
- Lock-free
  - Гарантирует, что система не стоит на месте даже при неудачном scheduling-е
  - Можно вставлять/читать во время перехеширования

Как этого добиться?

DEVEXPERTS

MRSW хеш-таблица

- Multiple Readers, Single Writer
  - Часто только один поток меняет данные!

DEVEXPERTS

MRSW хеш-таблица

- Как читать в процессе увеличения?

DEVEXPERTS

MRSW хеш-таблица

- Как читать в процессе увеличения?

- Увеличение в случае одного писателя тривиально:
  1. Создаём новую таблицу

DEVEXPERTS

MRSW хеш-таблица

- Как читать в процессе увеличения?

- Увеличение в случае одного писателя тривиально:
  1. Создаём новую таблицу
  2. Копируем элементы

DEVEXPERTS

MRSW хеш-таблица

- Как читать в процессе увеличения?

- Увеличение в случае одного писателя тривиально:
  1. Создаём новую таблицу
  2. Копируем элементы
  3. Меняем ссылку на таблицу

DEVEXPERTS

MRSW хеш-таблица

- Как читать в процессе увеличения?

- Увеличение в случае одного писателя тривиально:
  1. Создаём новую таблицу
  2. Копируем элементы
  3. Меняем ссылку на таблицу

MRSW хеш-таблица: get(key)

```
public class MRSWLongObjectHashMap<T> {
    private volatile Core core = new Core(…);

    public T get(long key) {
        return core.getInternal(key);
    }

    private class Core {
        final AtomicLongArray keys;
        final AtomicReferenceArray<T> values;

        public T getInternal(long key) { … }
    }
}
```


DEVEXPERTS

MRSW хеш-таблица: get(key)

```
public class MRSWLongObjectHashMap<T> {
    private volatile Core core = new Core(…);

    public T get(long key) {
        return core.getInternal(key);
    }

    private class Core {
        final AtomicLongArray keys;
        final AtomicReferenceArray<T> values;

        public T getInternal(long key) { … }
    }
}
```


DEVEXPERTS

MRSW хеш-таблица: get(key)

```
public class MRSWLongObjectHashMap<T> {
    private volatile Core core = new Core(…);

    public T get(long key) {
        return core.getInternal(key);
    }

    private class Core {
        final AtomicLongArray keys;
        final AtomicReferenceArray<T> values;

        public T getInternal(long key) { … }
    }
}
```

Меняется после перехешировании

DEVEXPERTS

MRSW хеш-таблица: get(key)

```
public class MRSWLongObjectHashMap<T> {
    private volatile Core core = new Core(…);

    public T get(long key) {
        return core.getInternal(key);
    }

    private class Core {
        final AtomicLongArray keys;
        final AtomicReferenceArray<T> values;

        public T getInternal(long key) { … }
    }
}
```

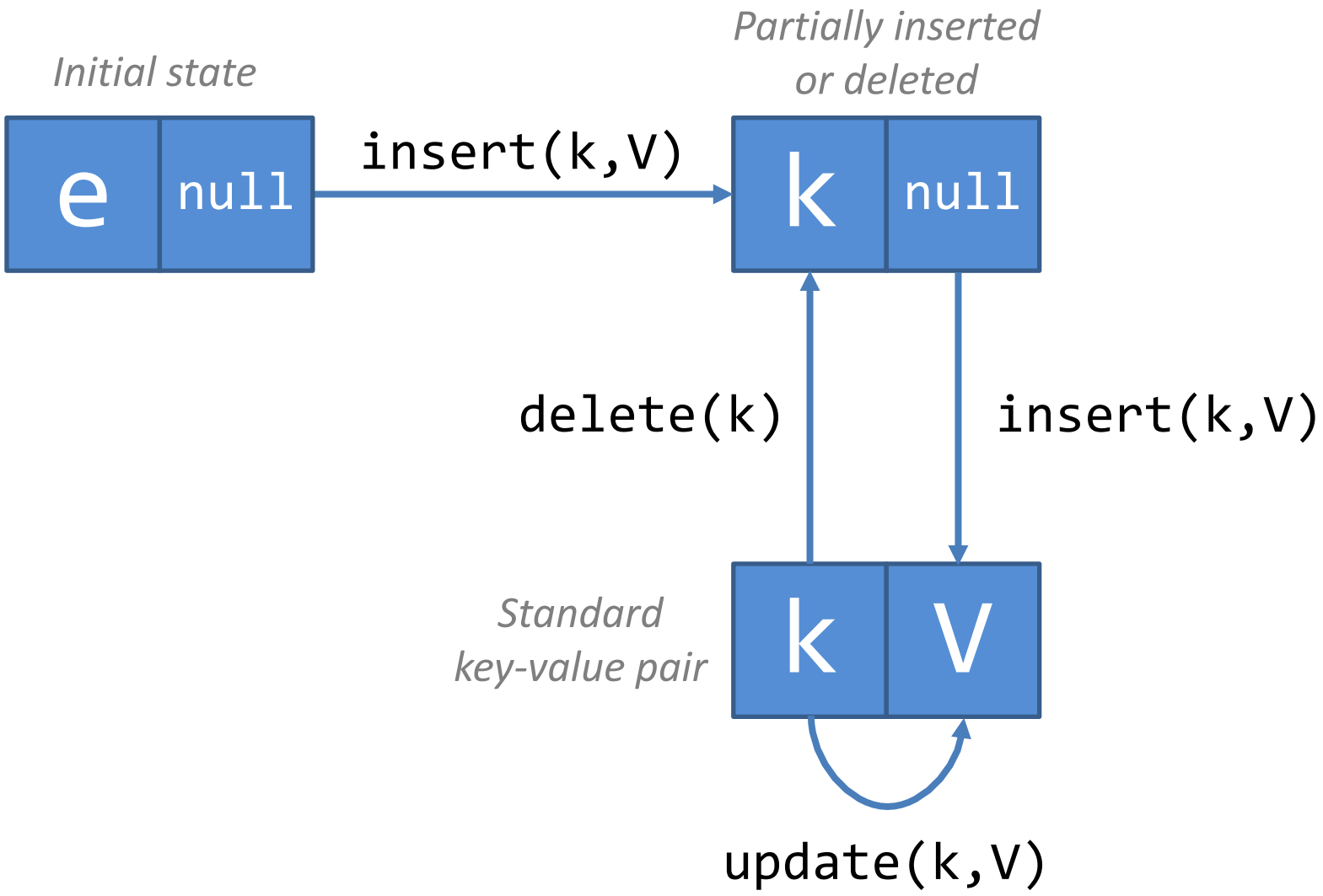

DEVEXPERTS

MRSW хеш-таблица: get(key)

```
public class MRSWLongObjectHashMap<T> {
    private volatile Core core = new Core(…);

    public T get(long key) {
        return core.getInternal(key);
    }

    private class Core {
        final AtomicLongArray keys;
        final AtomicReferenceArray<T> values;

        public T getInternal(long key) { … }
    }
}
```


DEVEXPERTS

Жизнь ячейки



e = empty key
k = some key

V = some value
null = tombstone / empty

DEVEXPERTS

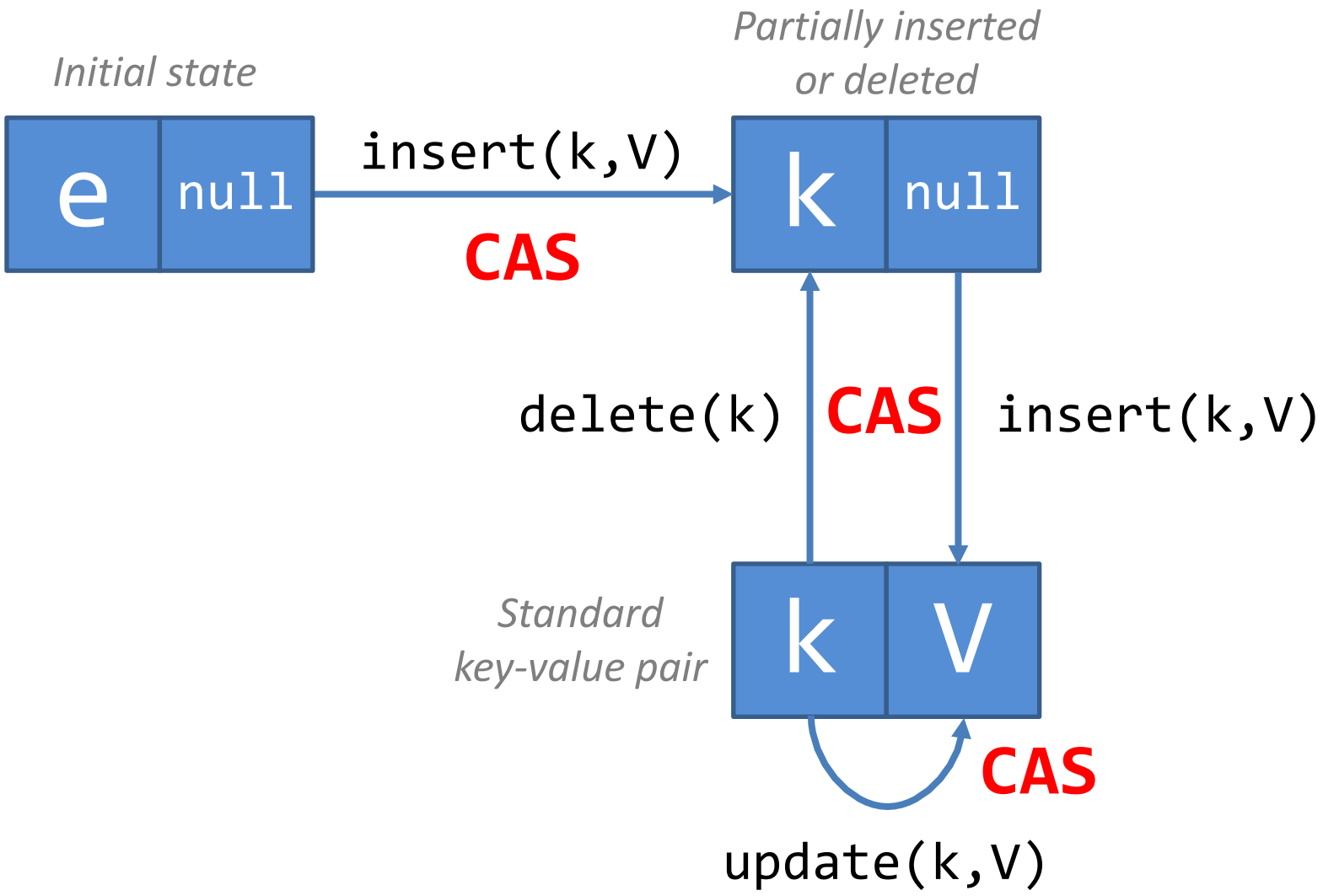
MRMW хеш-таблица

- Multiple Readers, Multiple Writers
  - То, чего мы хотим в общем случае

- Как организовать изменение из разных потоков?
- Как параллельно делать перенос элементов?

DEVEXPERTS

Жизнь ячейки: без переноса

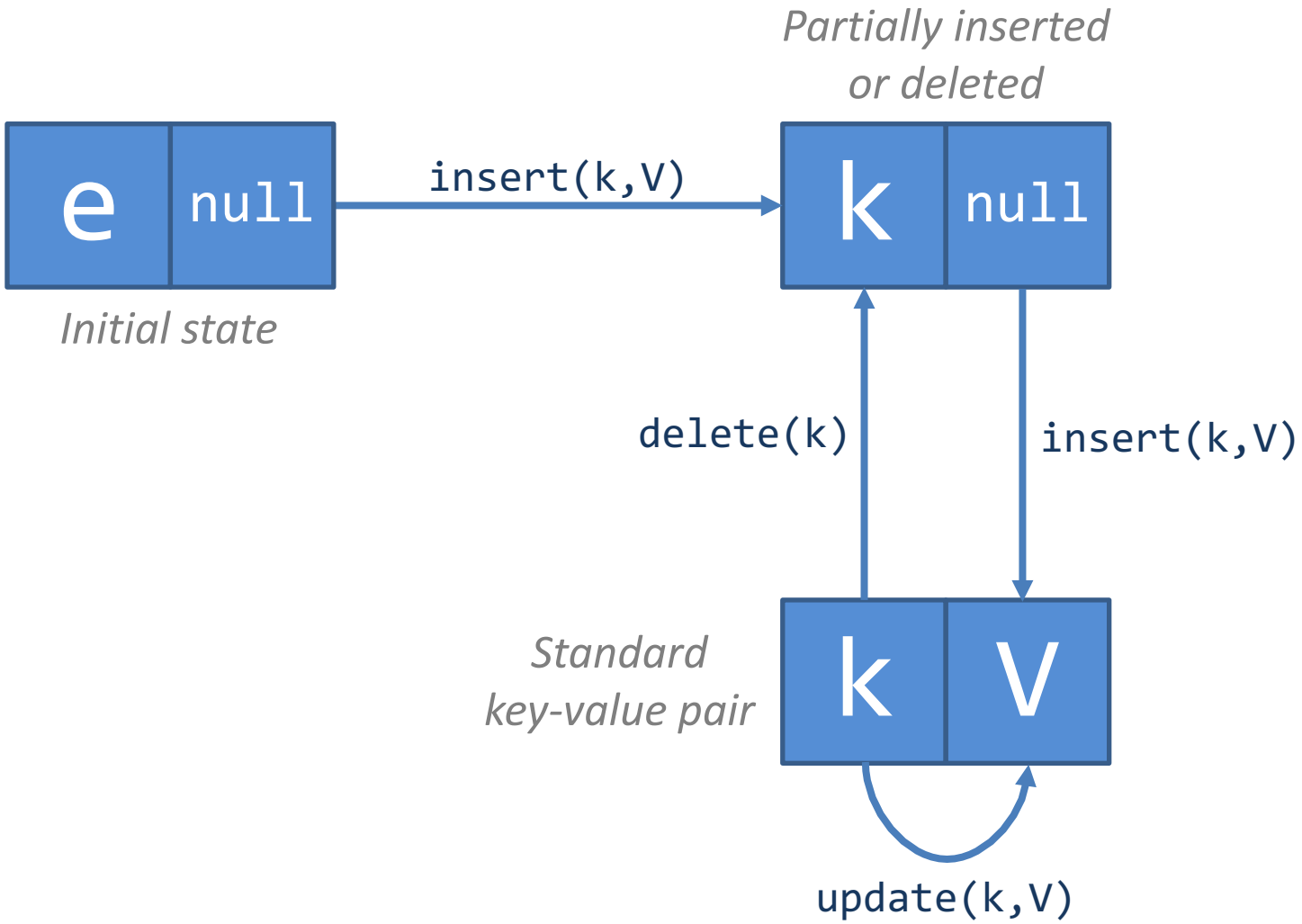


e = empty key
k = some key

V = some value
null = tombstone / empty

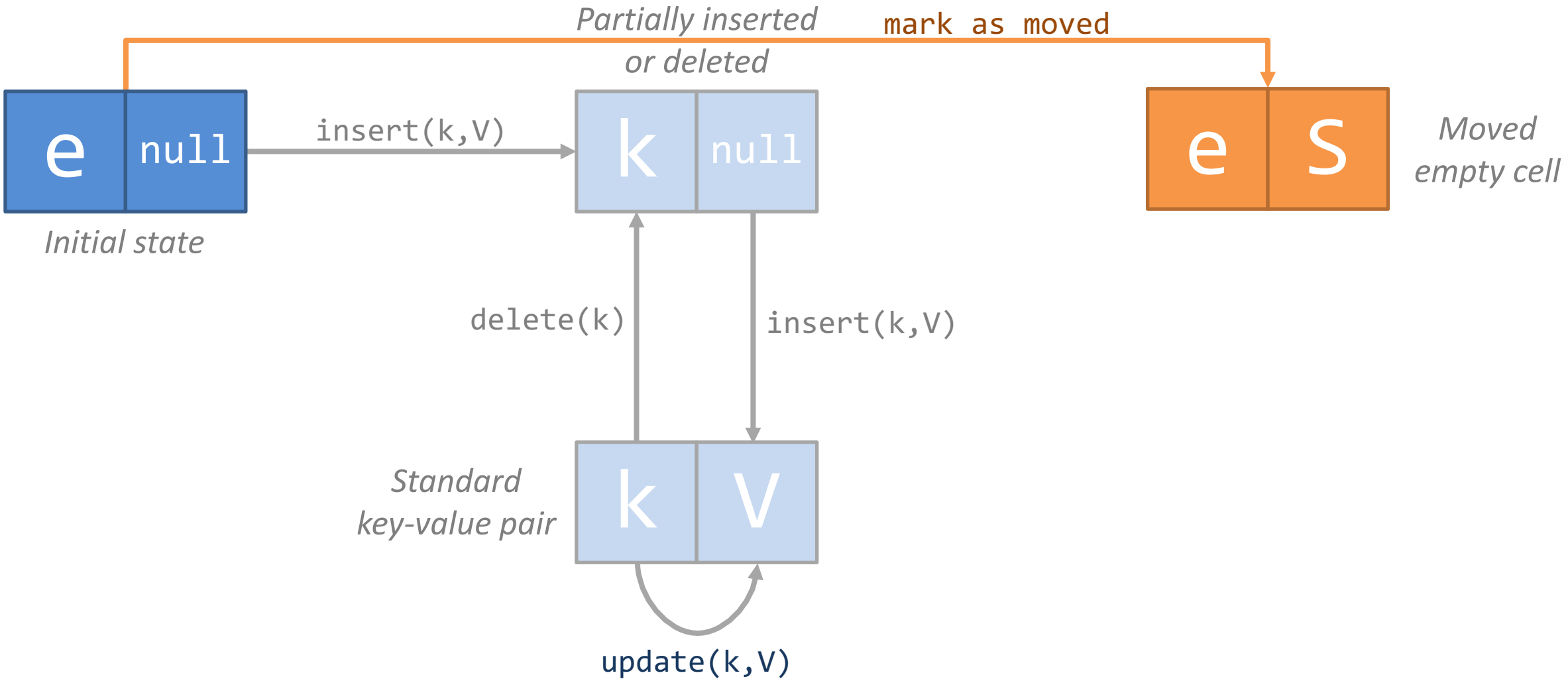
DEVEXPERTS

Жизнь ячейки: добавляем перенос



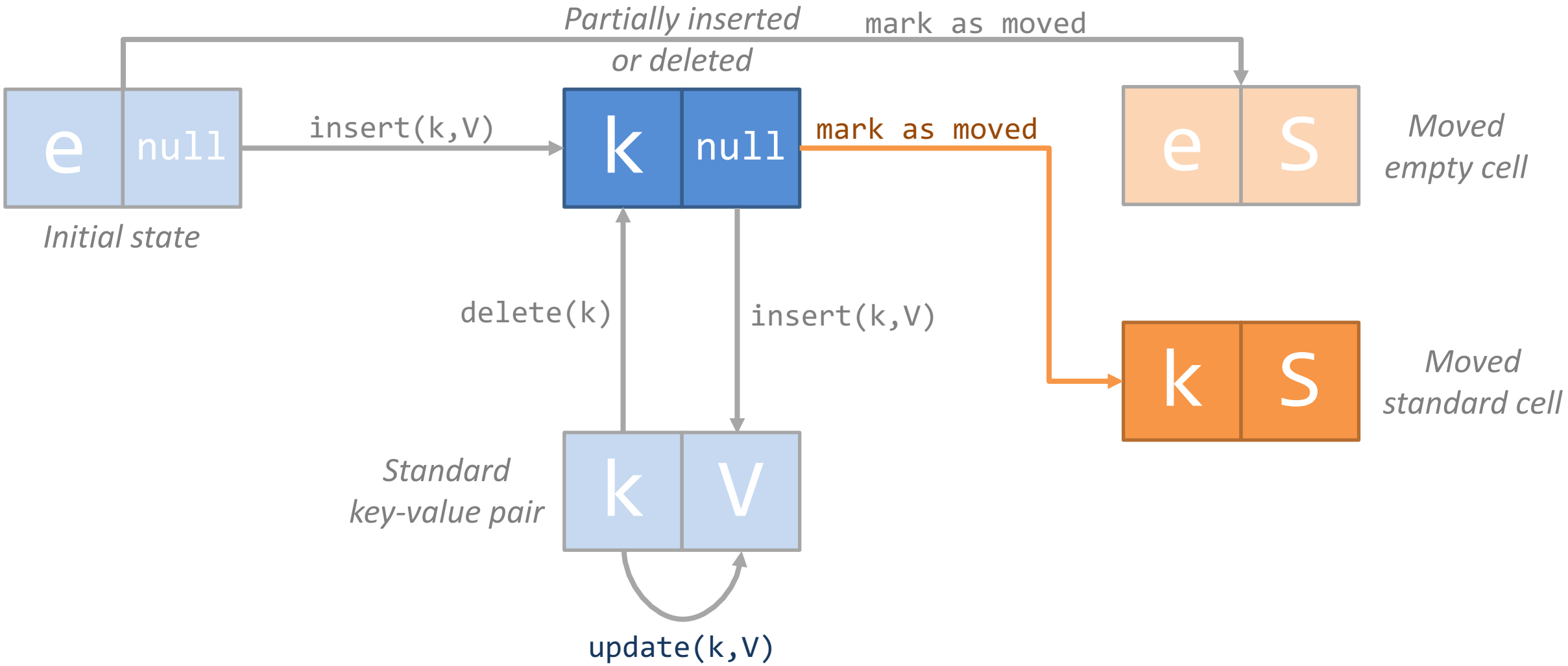
DEVEXPERTS

Жизнь ячейки: добавляем перенос



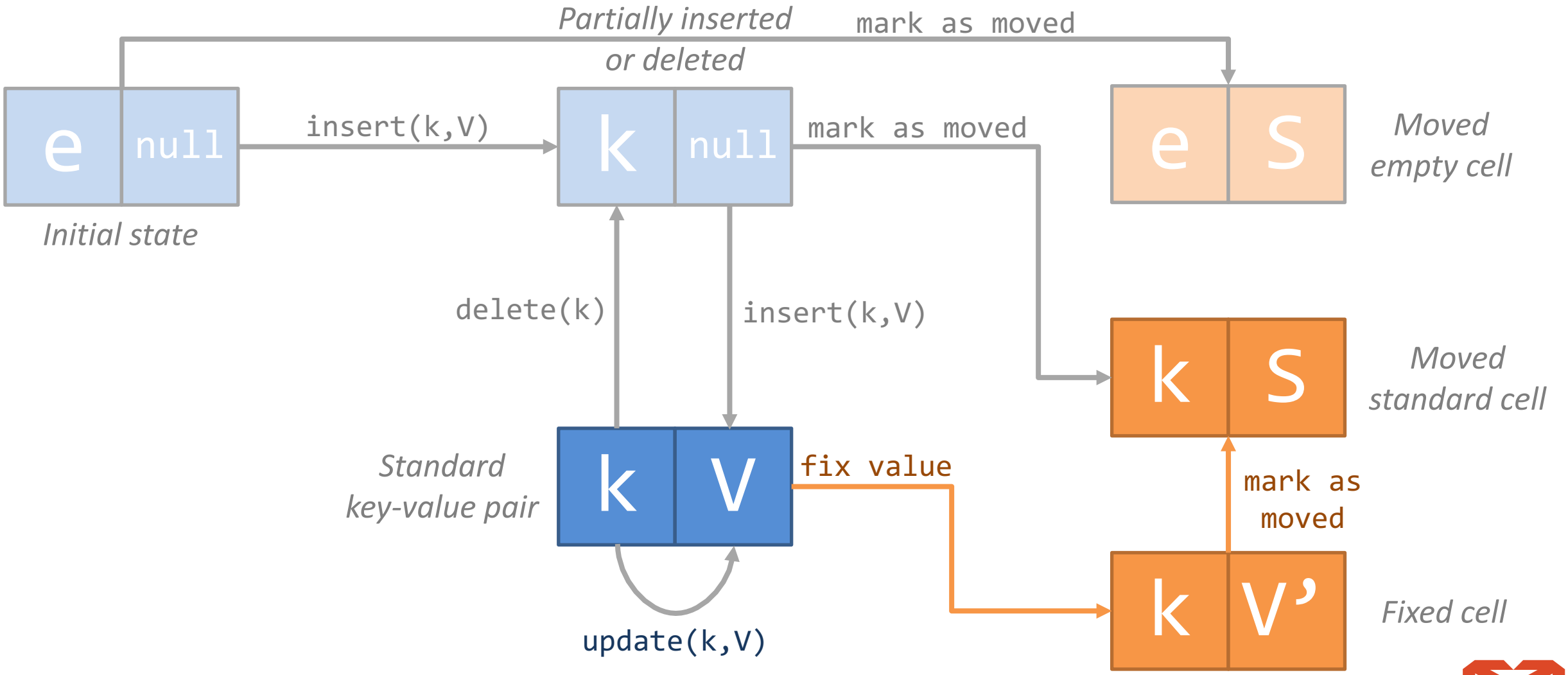
DEVEXPERTS

Жизнь ячейки: добавляем перенос



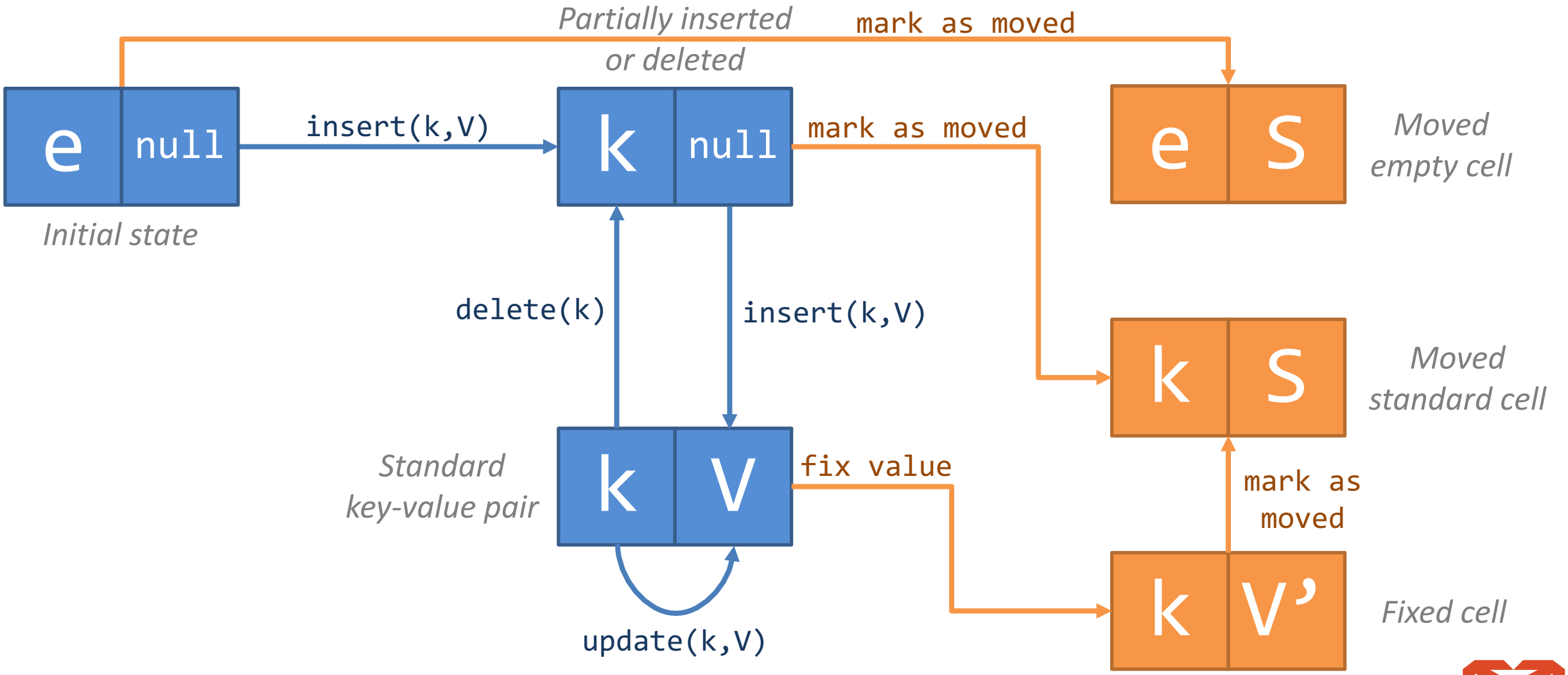
DEVEXPERTS

Жизнь ячейки: добавляем перенос



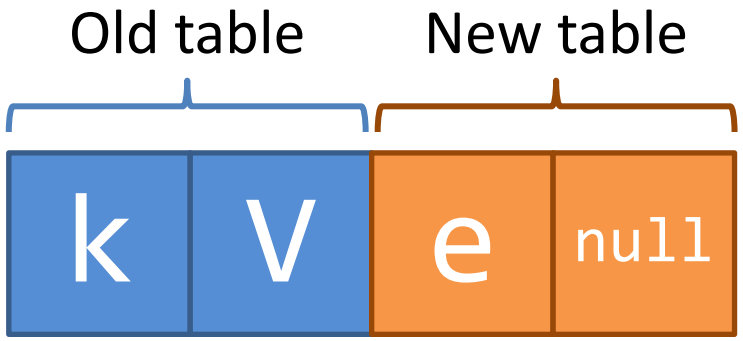
DEVEXPERTS

Жизнь ячейки: добавляем перенос



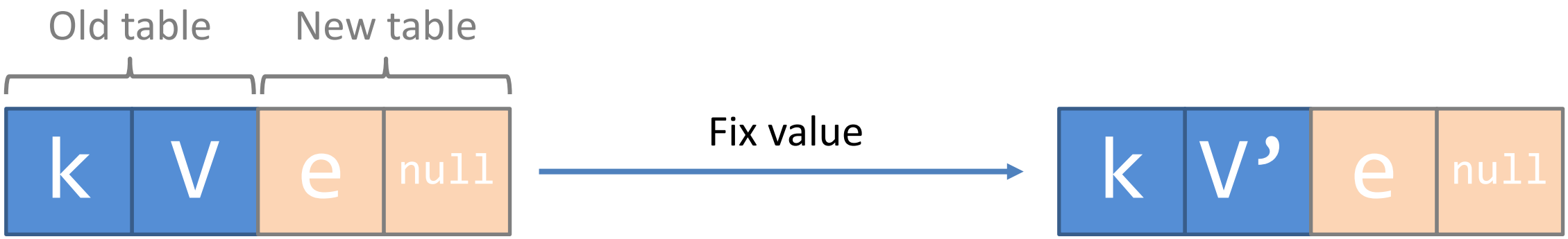
DEVEXPERTS

Перенос: с другого ракурса



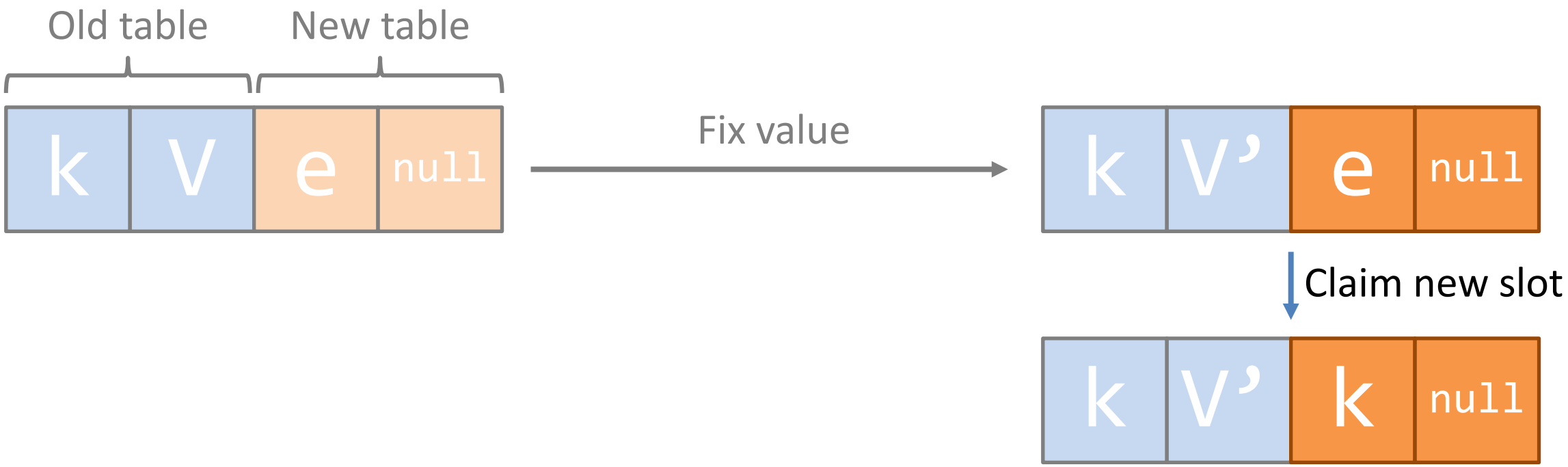
DEVEXPERTS

Перенос: с другого ракурса



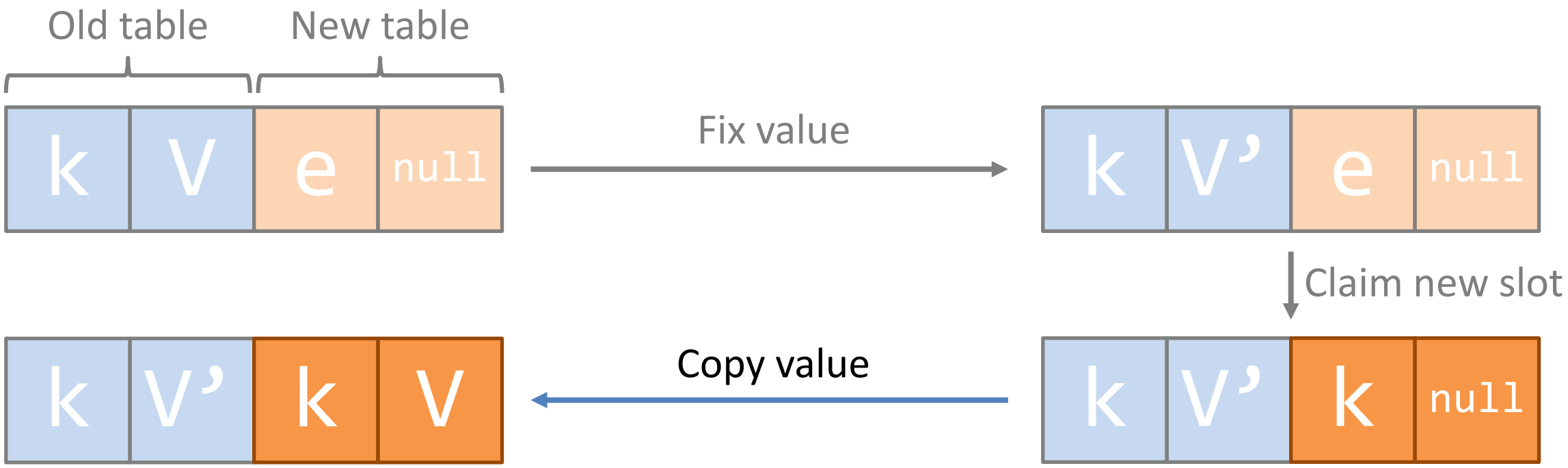
DEVEXPERTS

Перенос: с другого ракурса



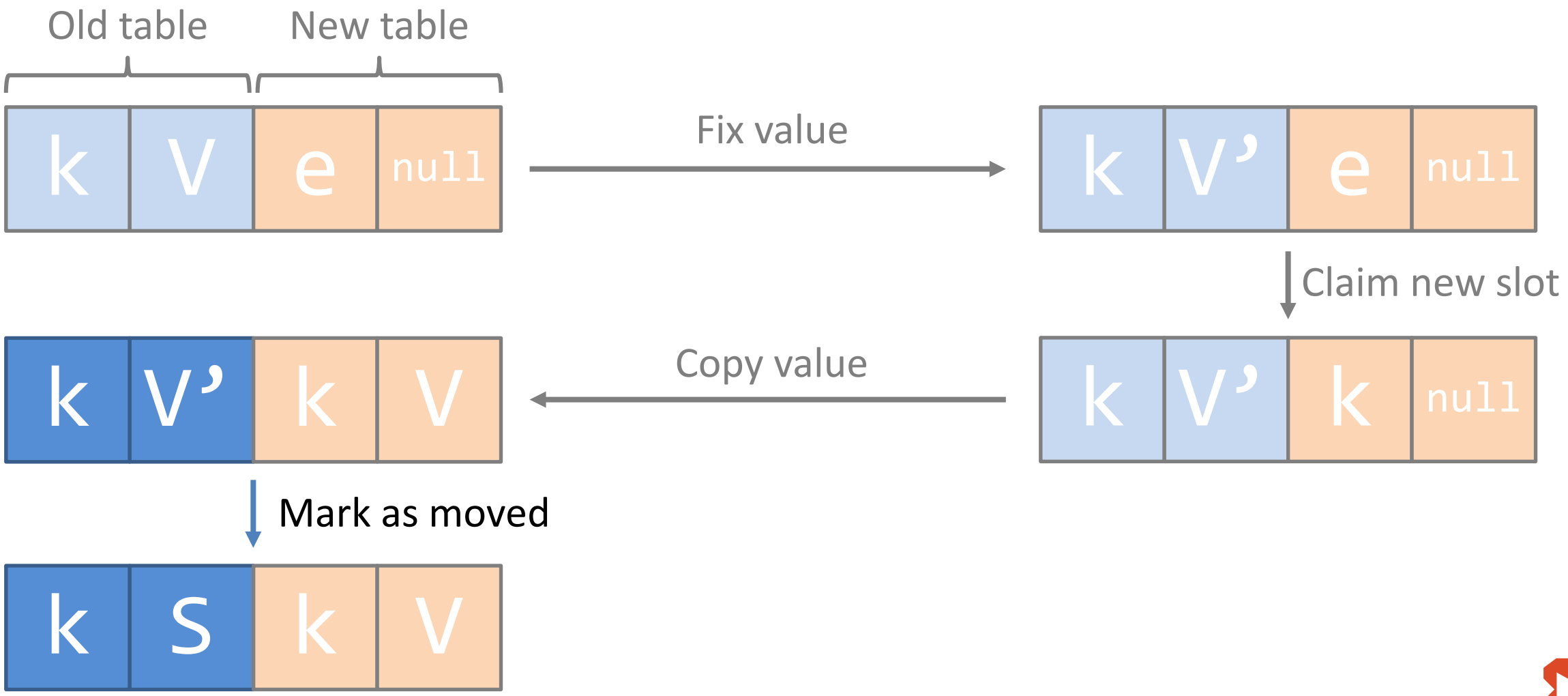
DEVEXPERTS

Перенос: с другого ракурса



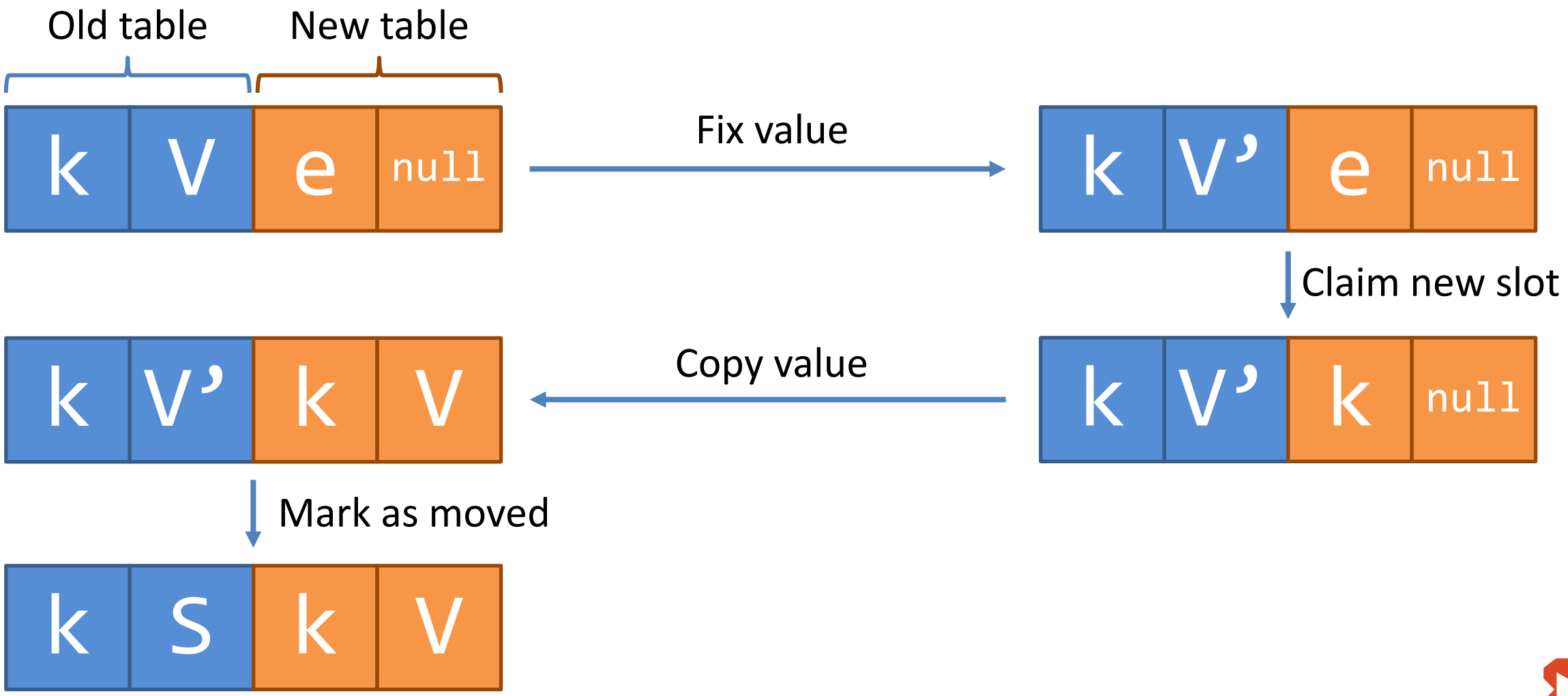
DEVEXPERTS

Перенос: с другого ракурса



DEVEXPERTS

Перенос: с другого ракурса



DEVEXPERTS

Кооперация при переносе

- Как несколько потоков могут переносить элементы и не мешать друг другу?
  - Блоками по K элементов (так делает Cliff Click)
  - Переносом занимается выделенный поток

НУ И ЗАЧЕМ МНЕ ЭТО ВСЁ ЗНАТЬ???

DEVEXPERTS

Вспомним модель!

```
interface AccountsCache {
    fun addAccount(id: Int, account: Account)
    fun getById(id: Int): Account?
}
```


DEVEXPERTS

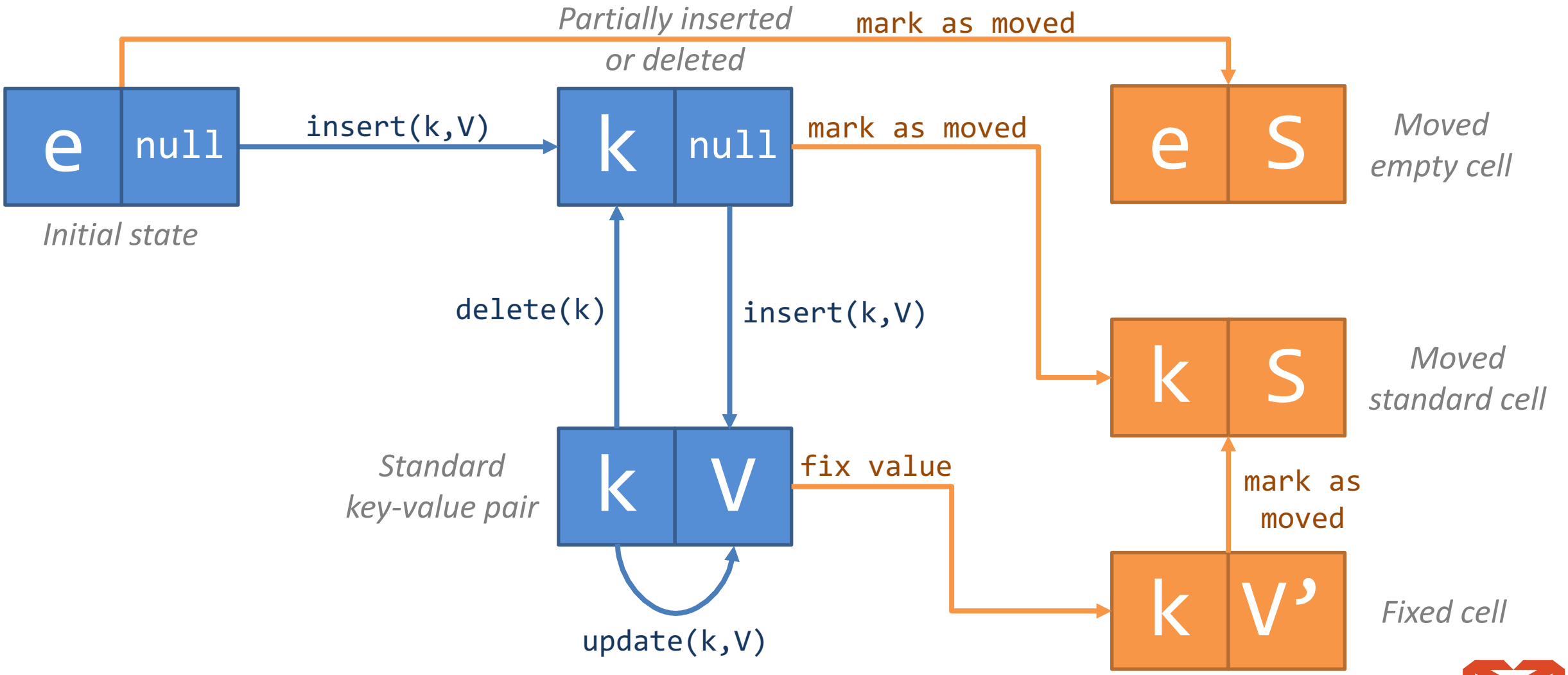
Вспомним модель!

```
interface AccountsCache {
    fun addAccount(id: Int, account: Account)
    fun getById(id: Int): Account?
}
```

- Account-ы никогда не меняются
- Удалений нет

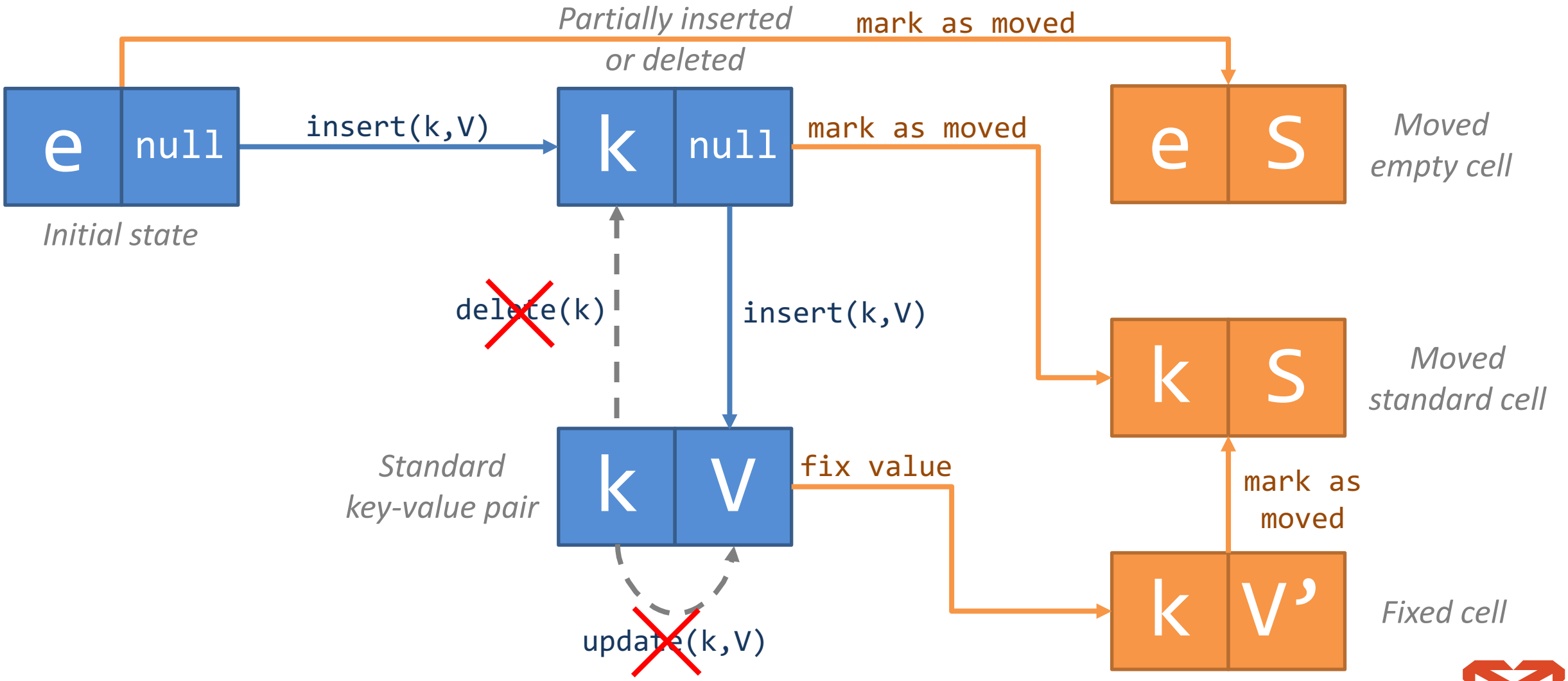
DEVEXPERTS

Упрощаем жизнь ячейки



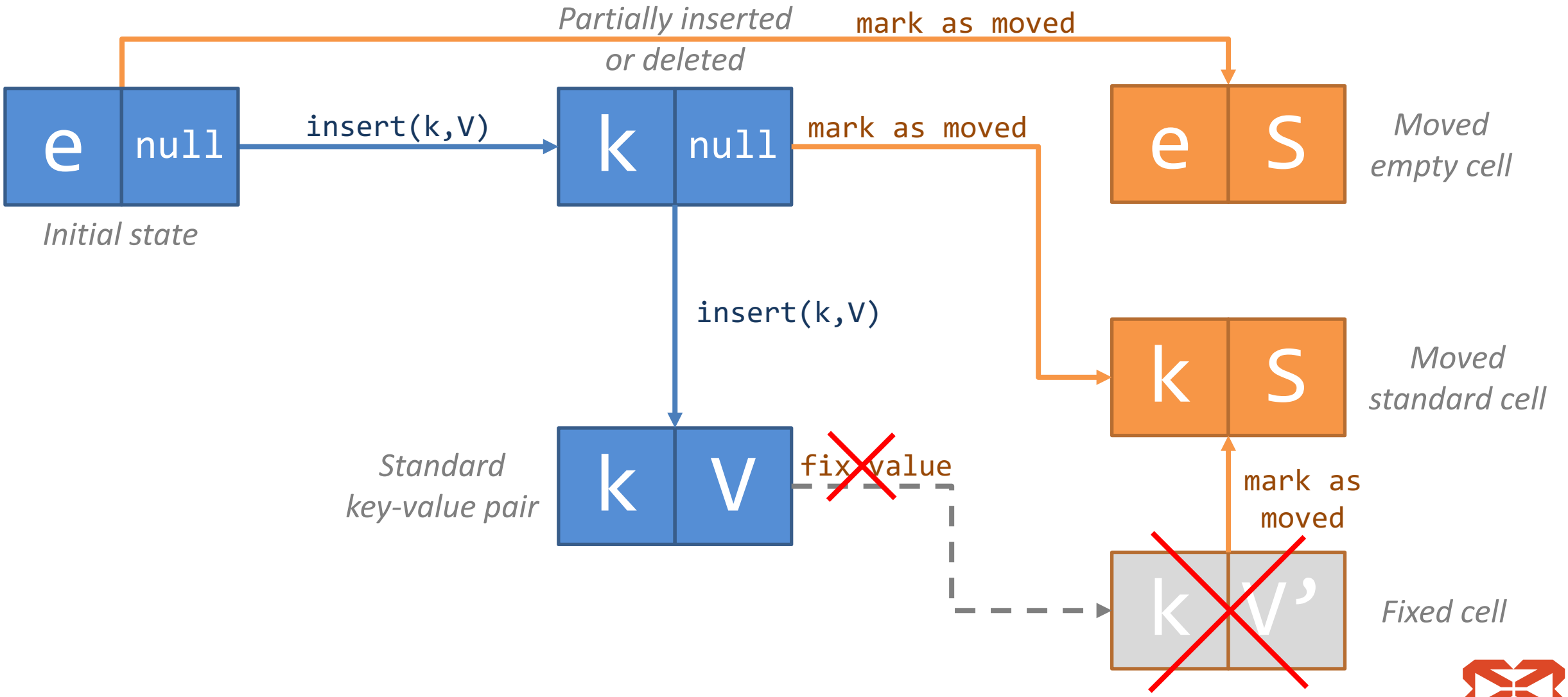
DEVEXPERTS

Упрощаем жизнь ячейки



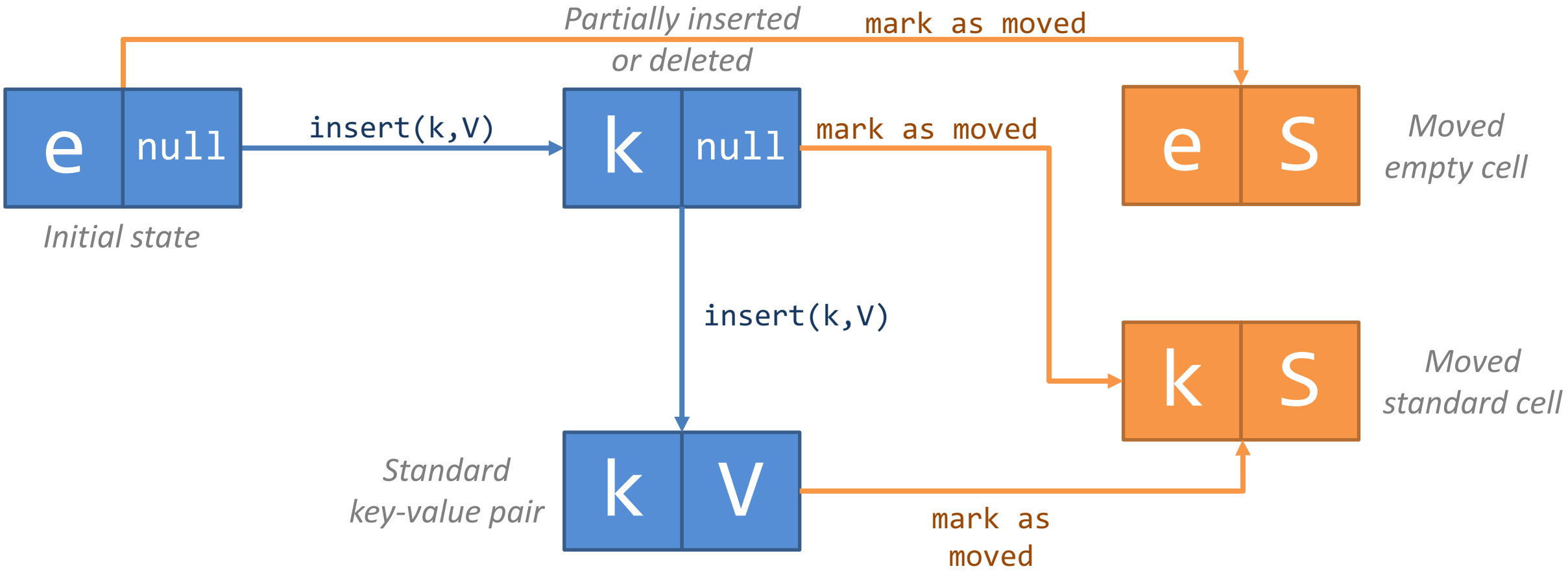
DEVEXPERTS

Упрощаем жизнь ячейки



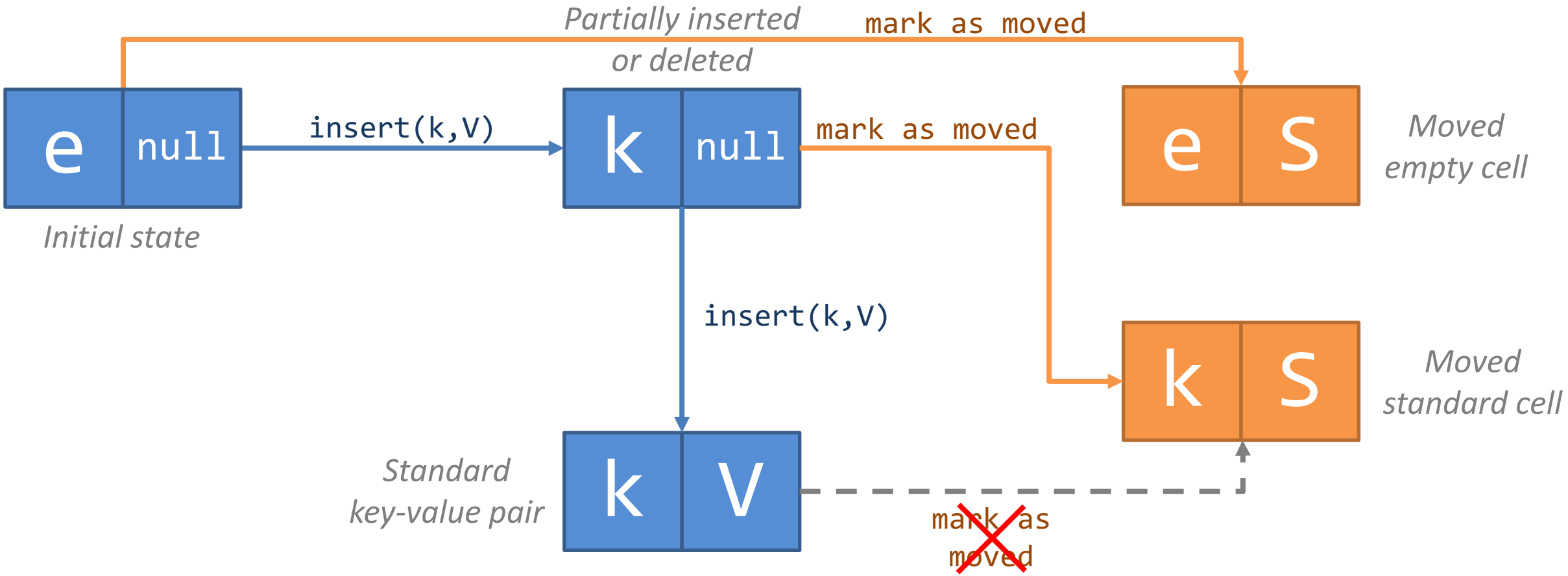
DEVEXPERTS

Упрощаем жизнь ячейки



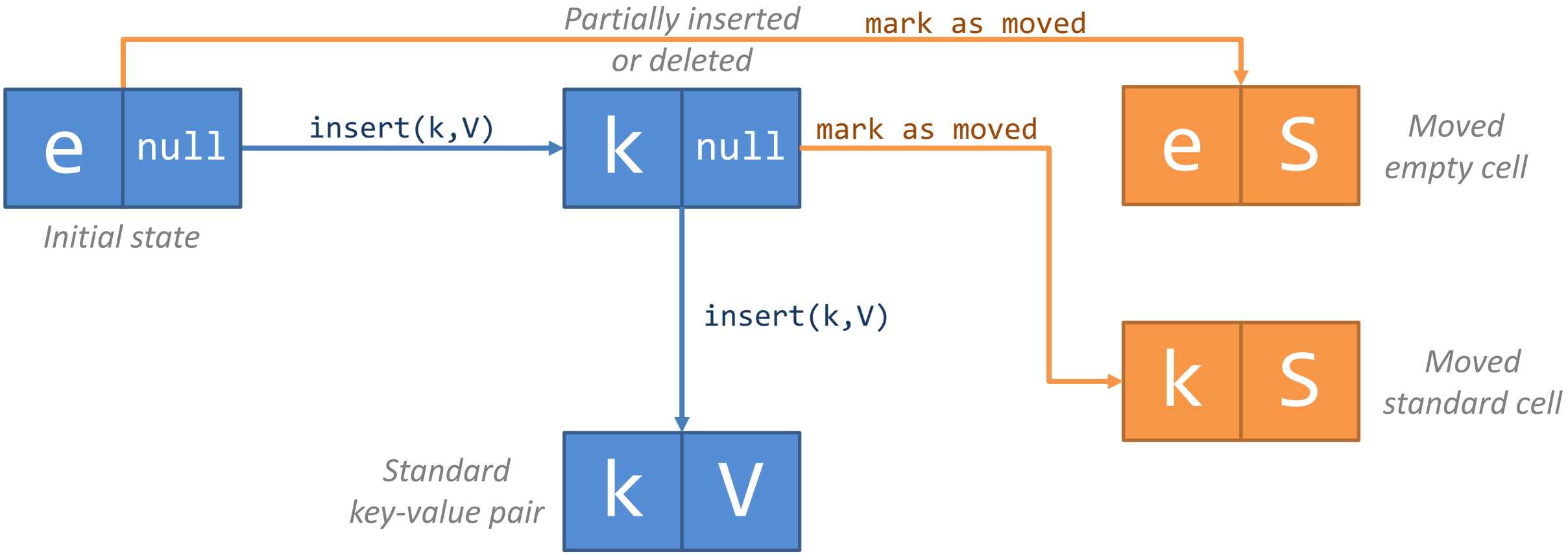
DEVEXPERTS

Упрощаем жизнь ячейки



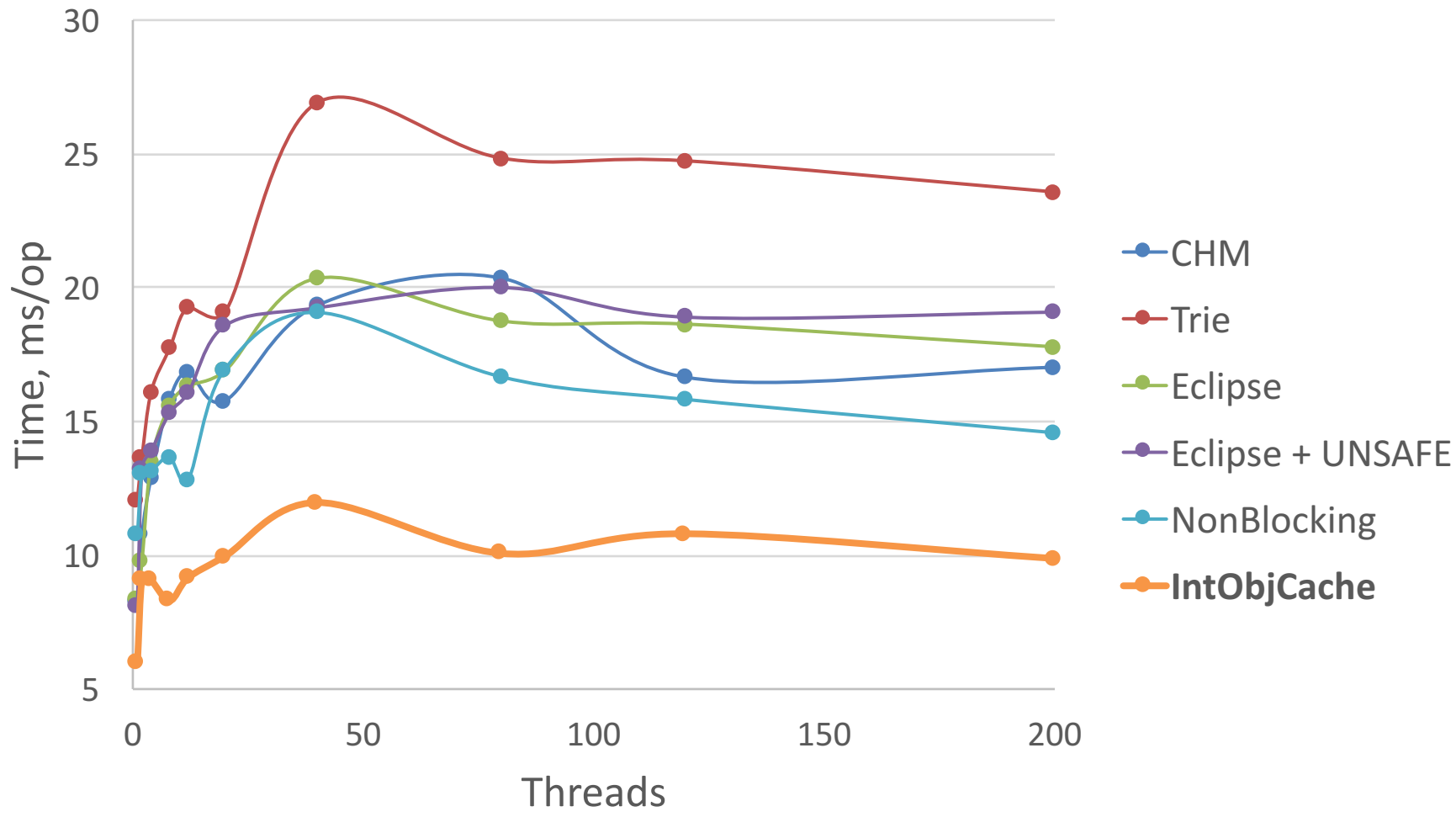
DEVEXPERTS

Упрощаем жизнь ячейки



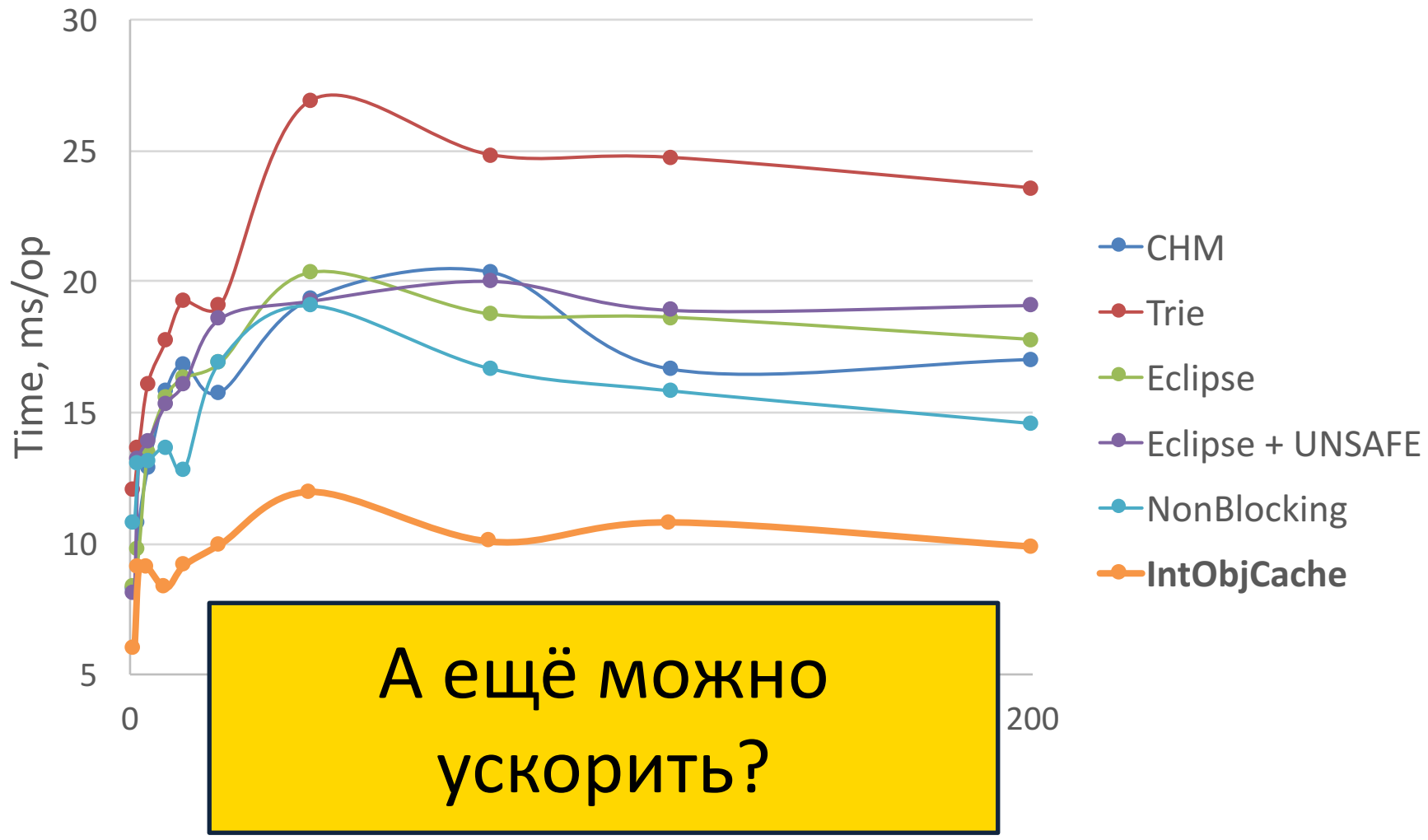
DEVEXPERTS

Производительность



DEVEXPERTS

Производительность



DEVEXPERTS

Посмотрим на модель ещё раз!

```
class Account(val id: Int) {
    private val i1: Int = 0
    private val i2: Int = 0
    private val o1: Any? = null
    private val o2: Any? = null
    private val o3: Any? = null
    private val o4: Any? = null
}
```


DEVEXPERTS

Посмотрим на модель ещё раз!

```
class Account(val id: Int) {
    private val i1: Int = 0
    private val i2: Int = 0
    private val o1: Any? = null
    private val o2: Any? = null
    private val o3: Any? = null
    private val o4: Any? = null
}
```

DEVEXPERTS

Посмотрим на модель ещё раз!

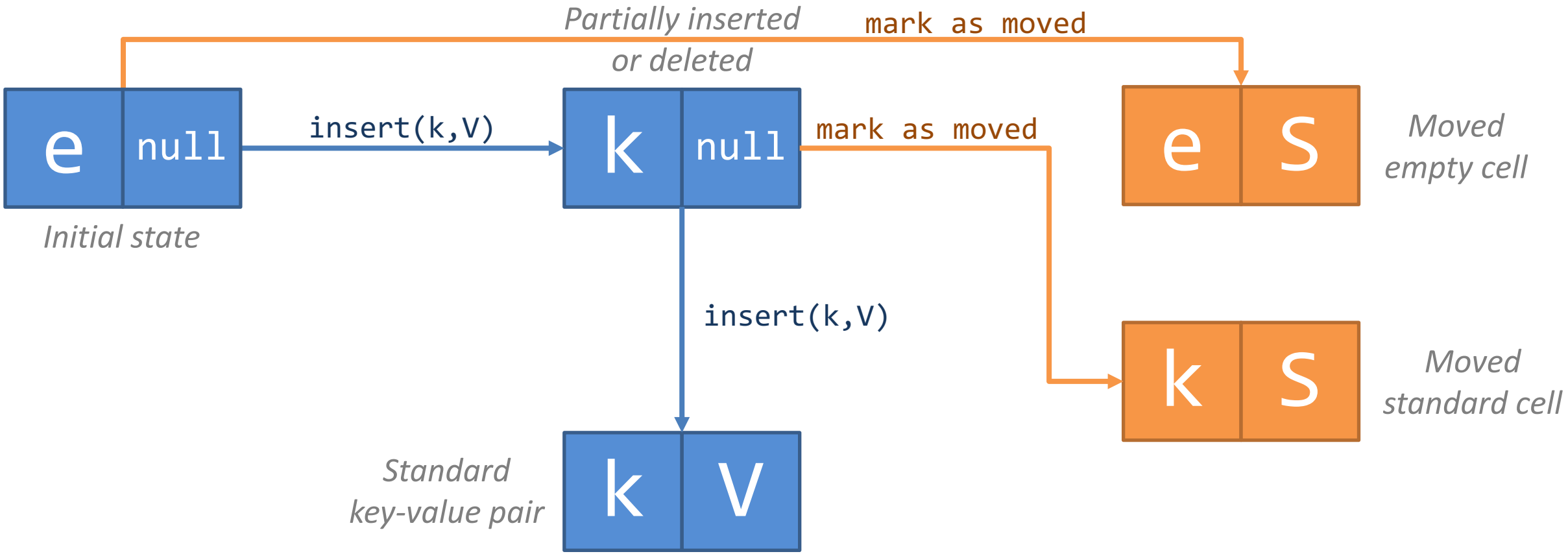
```
class Account(val id: Int) {
    private val i1: Int = 0
    private val i2: Int = 0
    private val o1: Any? = null
    private val o2: Any? = null
    private val o3: Any? = null
    private val o4: Any? = null
}
```

Уже храним id

Может не хранить ключ отдельно?

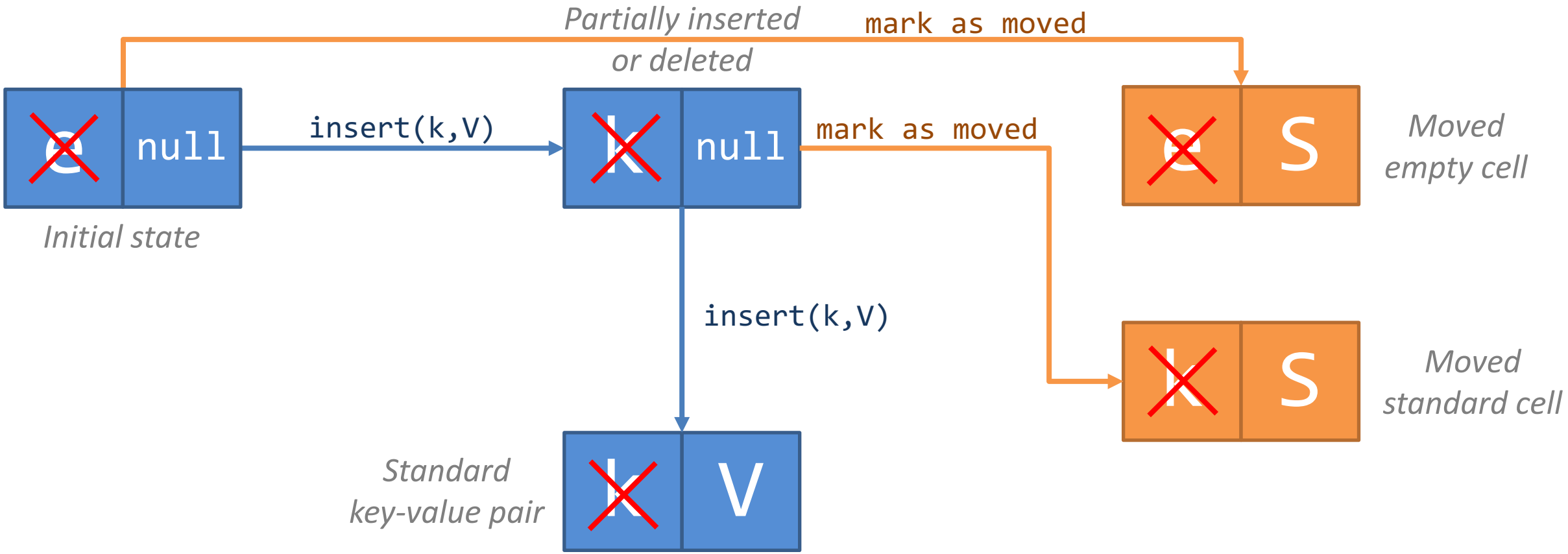
DEVEXPERTS

Упрощаем ячейку



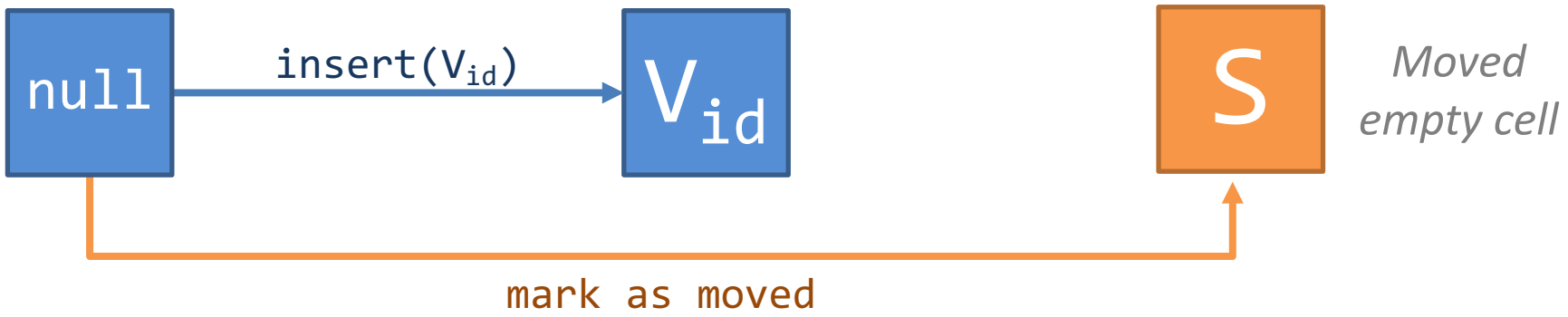
DEVEXPERTS

Упрощаем ячейку



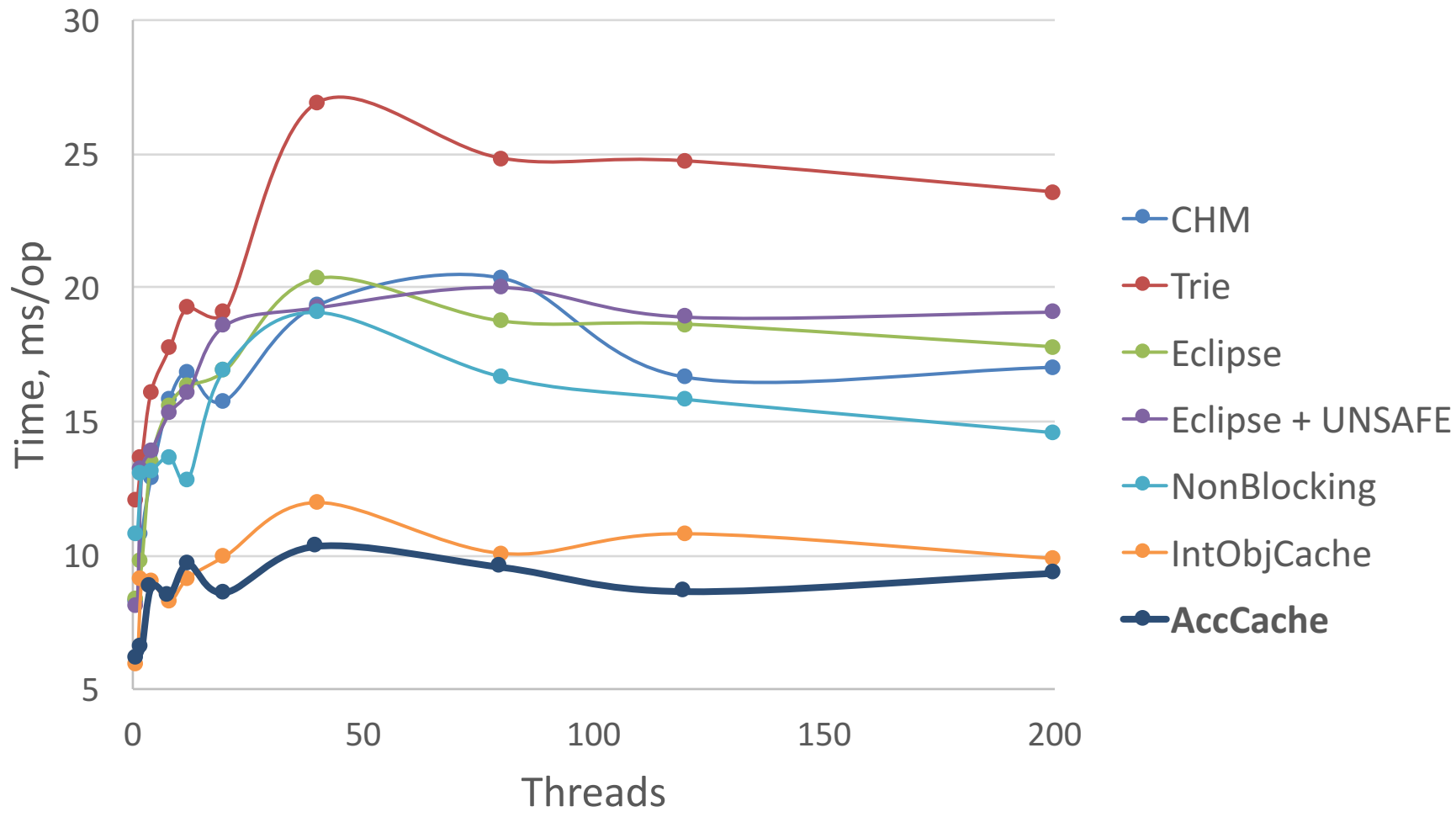
DEVEXPERTS

Упрощаем ячейку



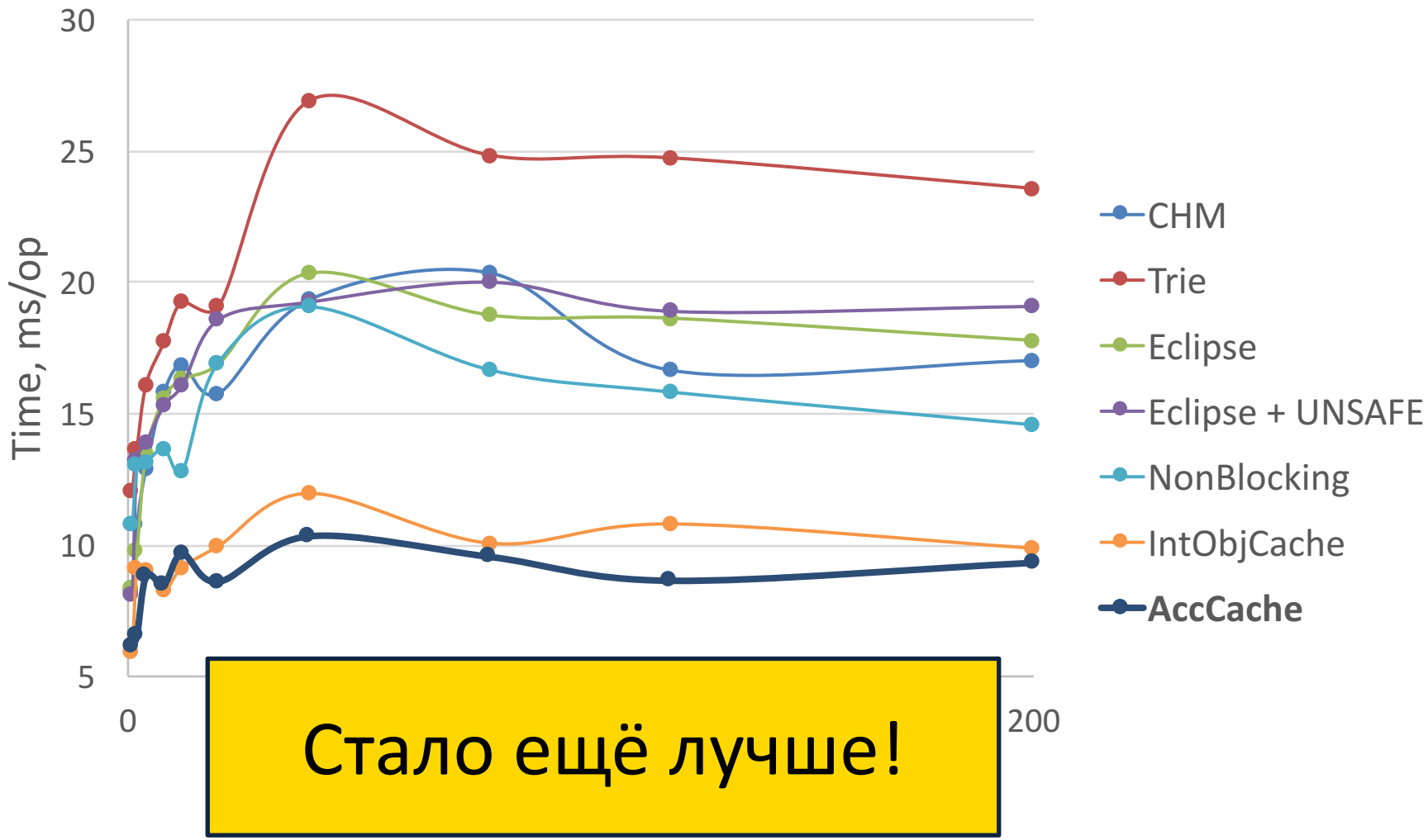
DEVEXPERTS

Производительность



DEVEXPERTS

Производительность



Стало ещё лучше!

DEVEXPERTS

Можно ли ещё улучшить?

- IntObjCache и AccCache «плохо» делают rehash

Можно ли ещё улучшить?

- IntObjCache и AccCache «плохо» делают rehash
- VarHandle позволяет избежать лишних чтений и синхронизаций
  - compareAndExchange помогает не делать лишних чтений после неудачного CAS-а

DEVEXPERTS

Можно ли ещё улучшить?

- IntObjCache и AccCache «плохо» делают rehash
- VarHandle позволяет избежать лишних чтений и синхронизаций
  - compareAndExchange помогает не делать лишних чтений после неудачного CAS-а
  - Что-то вроде release-acquire semantics, но не специфицировано в JMM (есть «спека» от Doug Lea)
    - Высокий порог вхождения

DEVEXPERTS

Итоги

- Как правило достаточно стандартных реализаций СД
  - Если недостаточно стандартных – поищите альтернативы

DEVEXPERTS

Итоги

- Как правило достаточно стандартных реализаций СД
  - Если недостаточно стандартных – поищите альтернативы

- При разработке своих алгоритмов важно
  - Понимать существующие подходы
  - Иметь простую и понятную модель
  - Использовать вводимые моделью ограничения

DEVEXPERTS

Спасибо за внимание!

Никита Коваль

ndkoval *at* ya *dot* ru
twitter.com/nkoval_

DEVEXPERTS