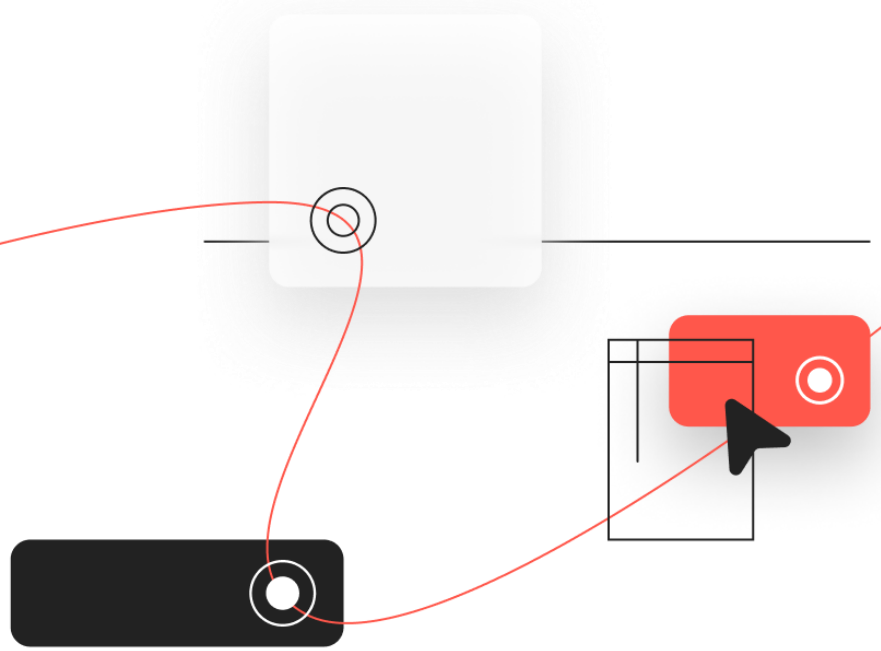
Как мы написали свой lock-free dictionary



Нечуговских Антон

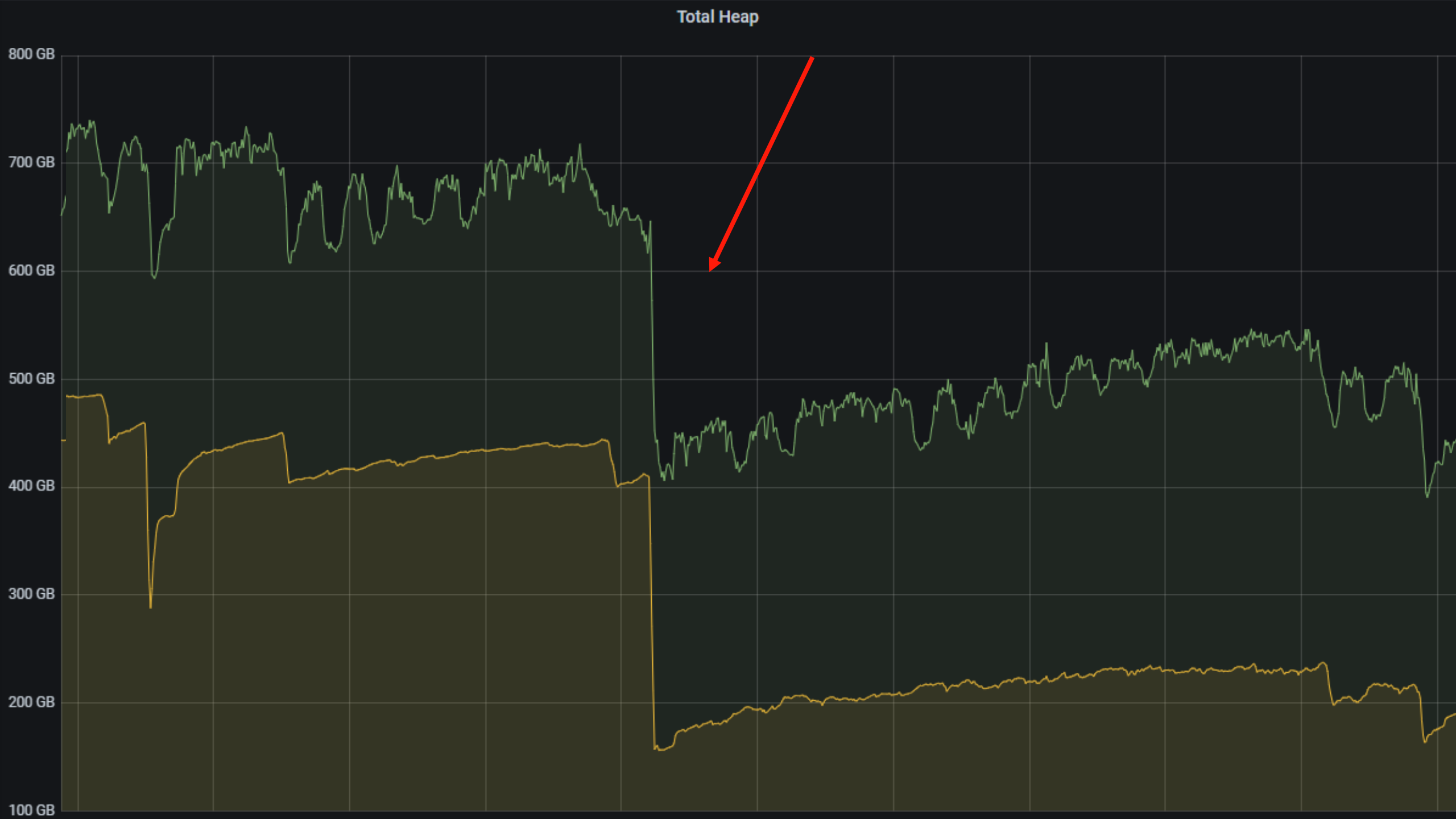
e-mail: nechugovskikh@kontur.ru

telegram: @ryzhes

Контур

О чём доклад

- Устройство словарей
- Многопоточность (lock-free, барьеры памяти etc.)
- Велосипеды


Total Heap
800 GB
700 GB
600 GB
500 GB
400 GB
300 GB
200 GB
100 GB

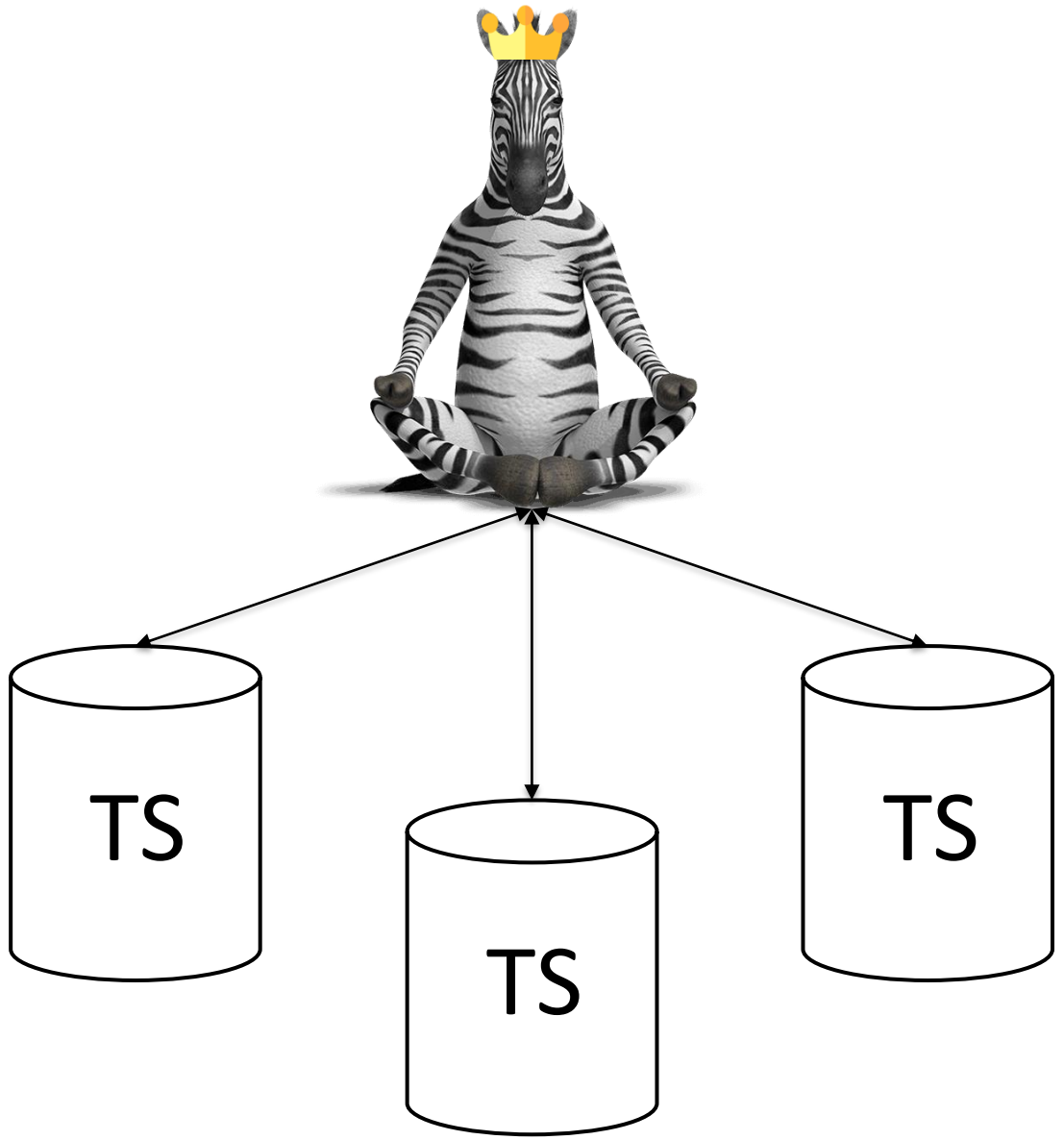
О чём доклад

- Устройство словарей
- Многопоточность (lock-free, барьеры памяти etc.)
- Велосипеды

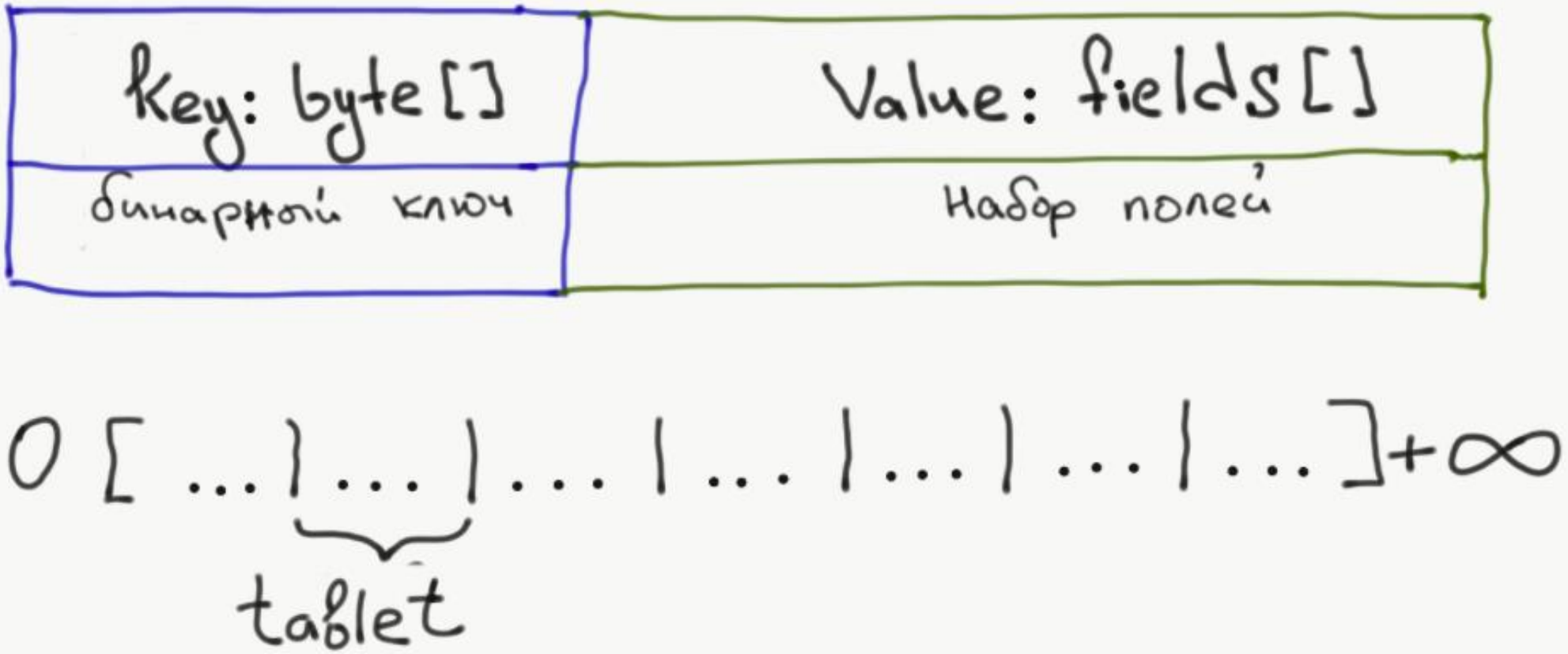
Zebra – распределённое
высоконагруженное отказоустойчивое
документно-ориентированное хранилище

Устройство Zebra

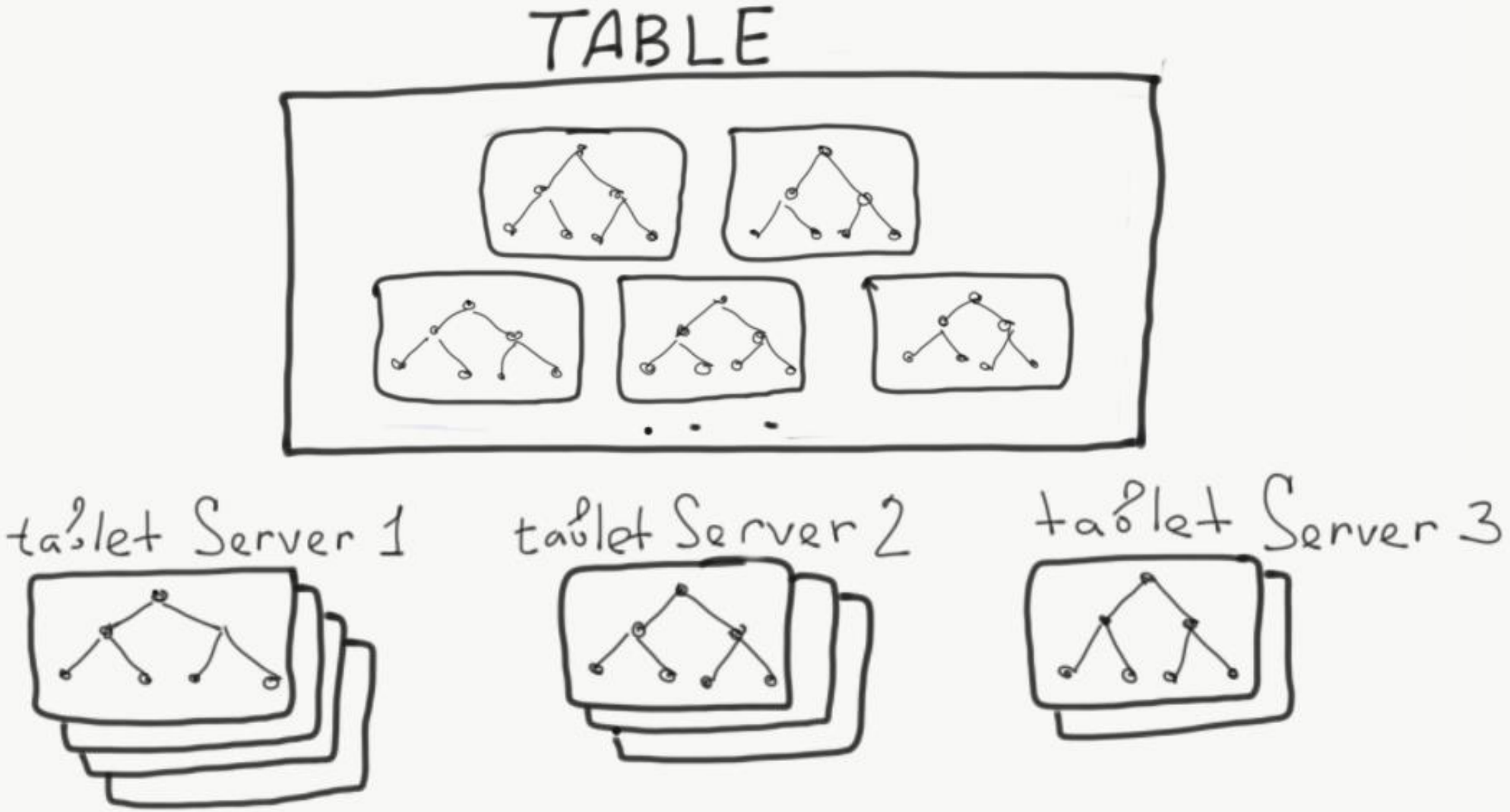
- **Мастер** – управляет лидерством, знает, где и что лежит
- **Таблет-серверы** (TS) – хранят данные, отвечают на запросы чтения/записи



Устройство Zebra



Устройство Zebra



А в чём проблема?

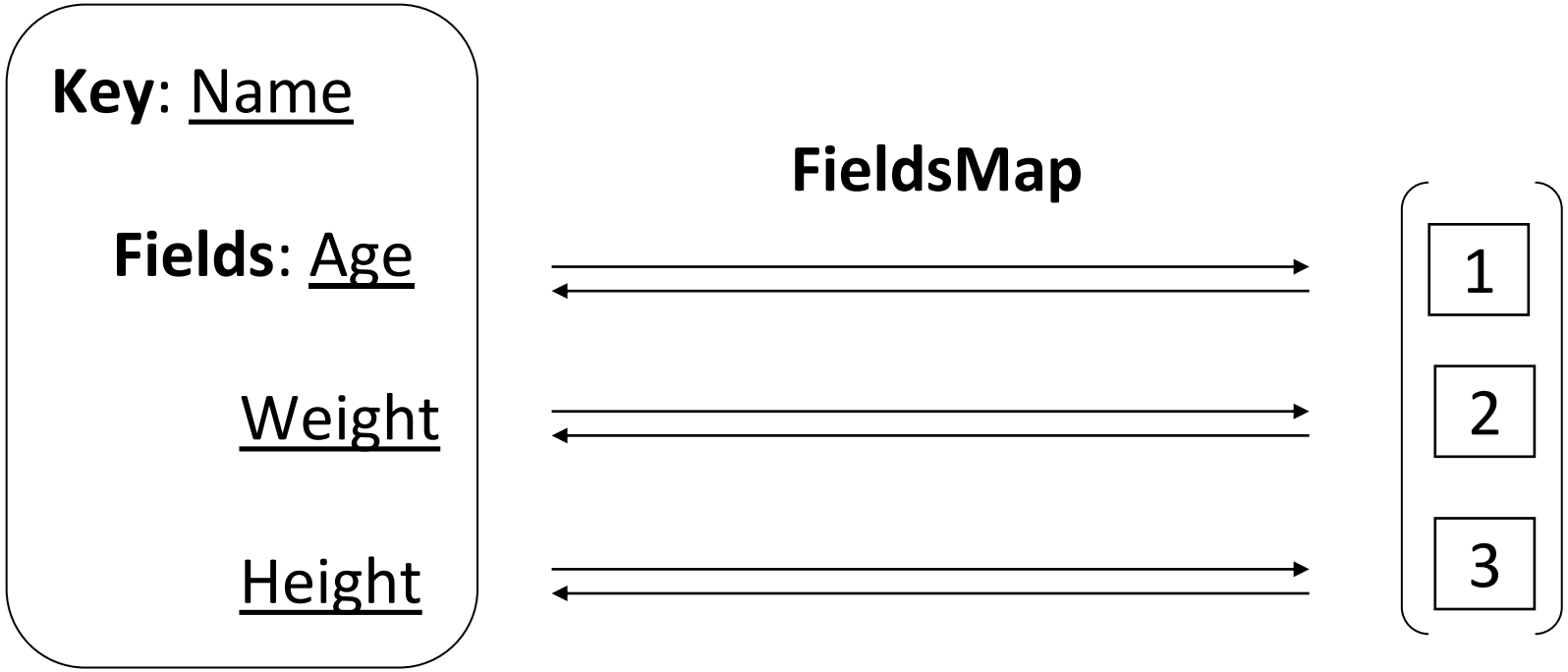
Дамп с проблемной машины

```
  Count         Size ClassName
 475148     19005920 ConcurrentDictionary`2+Node[String, UInt32]
 475148     22807104 ConcurrentDictionary`2+Node[UInt32, String]
 165667     25765328 System.Char[]
 473931     54997872 System.Object[]
 207928     57476304 System.Int32[]
4590319    110167656 System.Object
1505479    136176586 System.String
 173753    969062931 System.Byte[]
```

Дамп с проблемной машины

```
  Count         Size ClassName
 475148     19005920 ConcurrentDictionary`2+Node[String, UInt32]
 475148     22807104 ConcurrentDictionary`2+Node[UInt32, String]
 165667     25765328 System.Char[]
 473931     54997872 System.Object[]
 207928     57476304 System.Int32[]
4590319    110167656 System.Object
1505479    136176586 System.String
 173753    969062931 System.Byte[]
```

Поля в таблицах



Наблюдения про ConcurrentDictionary

- Overhead в 1 объект на куче на каждую запись
- Пересоздание всех Node на каждый Resize
- Lock при записи
- Больше потребляет чем обычный Dictionary

Перед написанием велосипеда

Какую локальную задачу решаем?

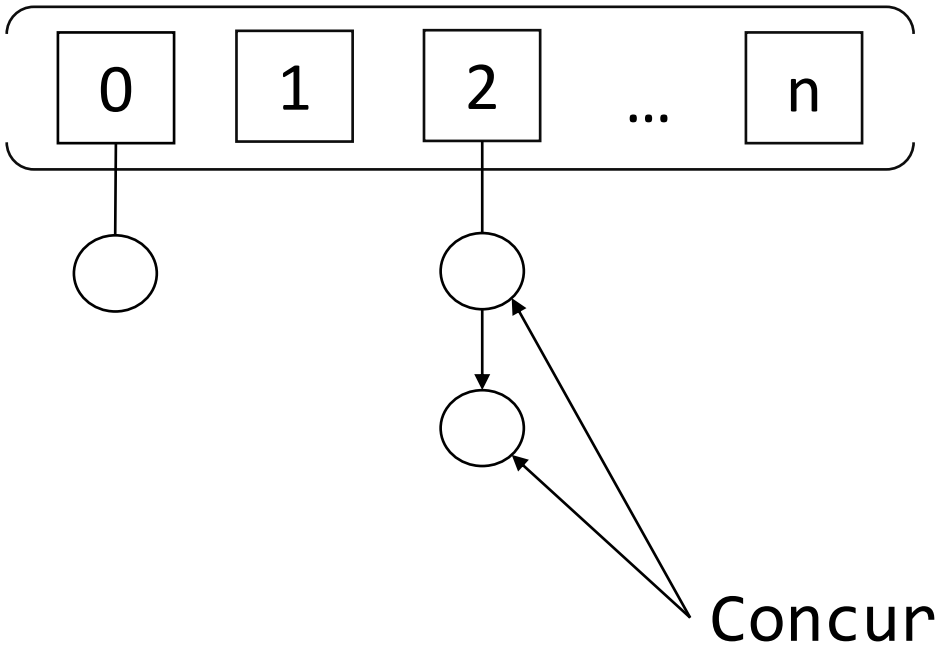
- Поля таблиц
- Append-only
- Multiple readers
- **Single** writer

Какие требования предъявляем?

- Single writer multiple readers модель
- Нет overhead`а по объектам
- Lock-free чтения
- Уменьшение потребления памяти

Про устройство словаря

Buckets:

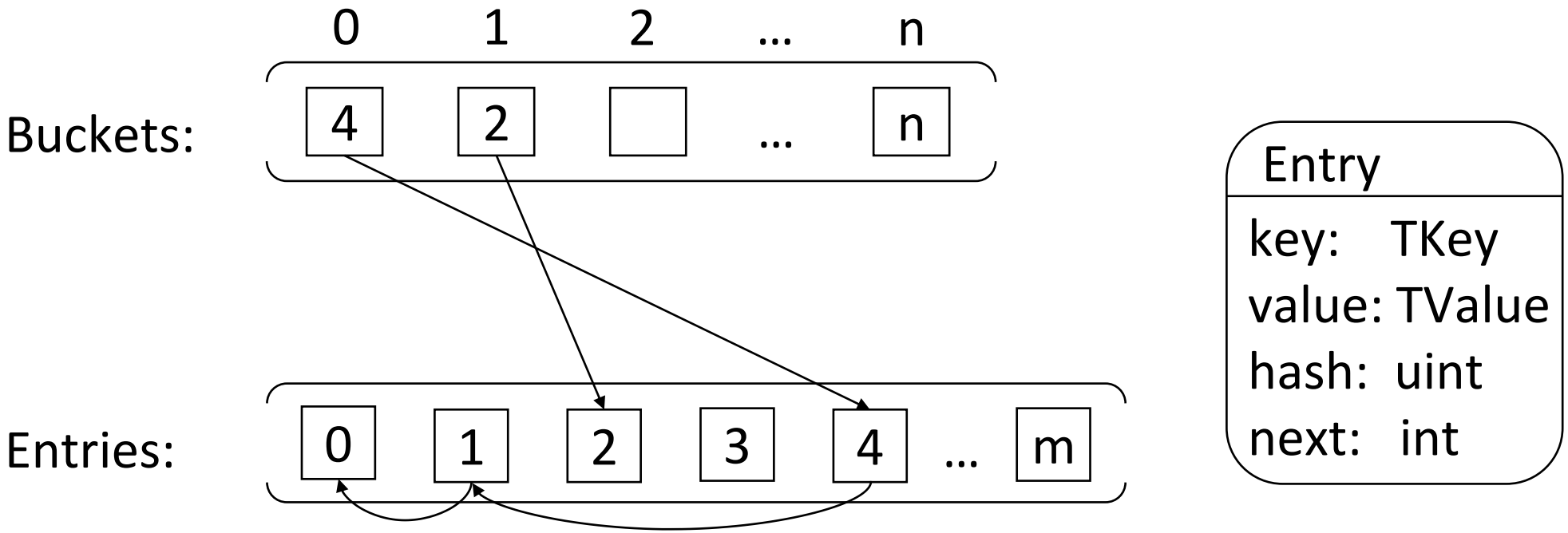


Key: ***
Value: *****

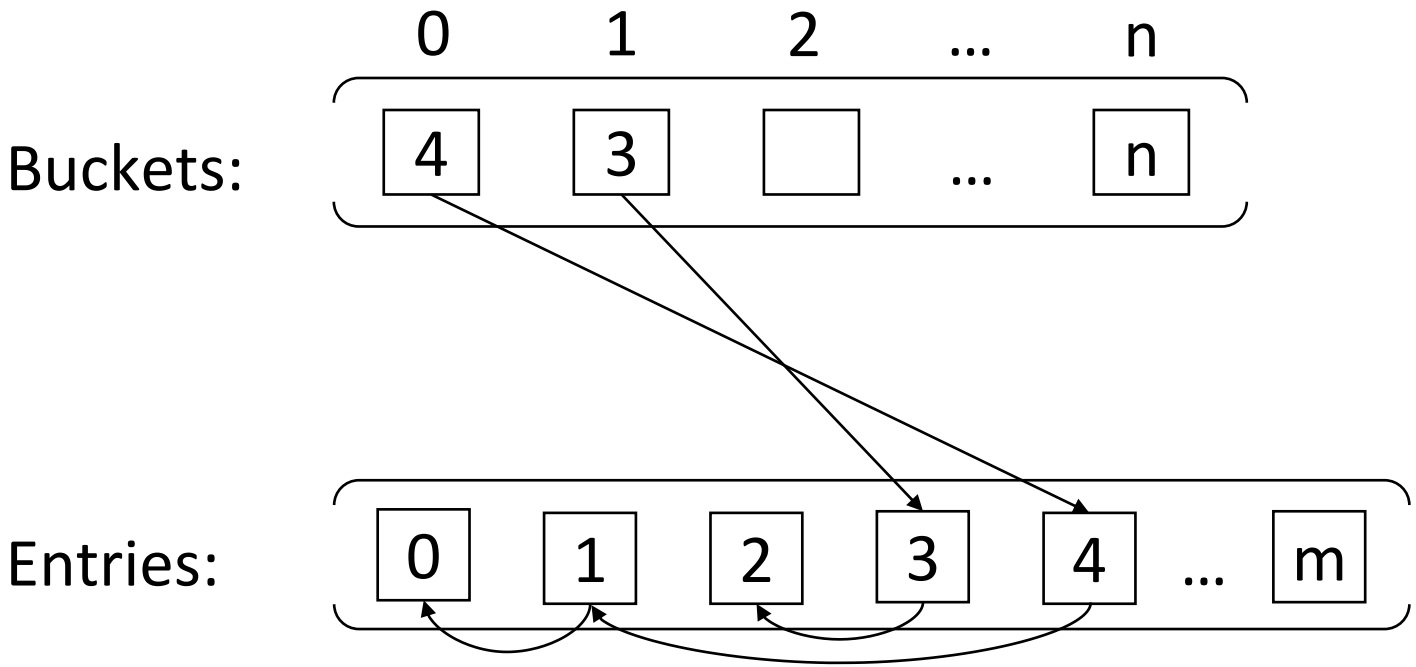
Hash: 2

`ConcurrentDictionary.Node`

Как в Dictionary на самом деле



Как в Dictionary на самом деле



Entry: **struct**

key: TKey
value: TValue
hash: uint
next: int

Добавление

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(buckets, hashCode);
ref Entry entry = ref entries[count];

entry.hashCode = hashCode;
entry.next = bucket;
entry.key = key;
entry.value = value;

bucket = count;
```

Добавление

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(buckets, hashCode);
ref Entry entry = ref entries[count];

entry.hashCode = hashCode;
entry.next = bucket;
entry.key = key;
entry.value = value;

bucket = count;
```

Добавление

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(buckets, hashCode);
ref Entry entry = ref entries[count];

entry.hashCode = hashCode;
entry.next = bucket;
entry.key = key;
entry.value = value;

bucket = count;
```

Добавление

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(buckets, hashCode);
ref Entry entry = ref entries[count];

entry.hashCode = hashCode;
entry.next = bucket;
entry.key = key;
entry.value = value;

bucket = count;
```

Добавление

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(buckets, hashCode);
ref Entry entry = ref entries[count];

entry.hashCode = hashCode;
entry.next = bucket;
entry.key = key;
entry.value = value;

bucket = count;
```

Поиск

```
var buckets = _buckets;
var entries = _entries;
var hashCode = (uint)key.GetHashCode();

int bucket = ref GetBucket(buckets, hashCode);

// линейный поиск по бакету
```

Поиск

```
var buckets = _buckets;
var entries = _entries;
var hashCode = (uint)key.GetHashCode();

int bucket = ref GetBucket(buckets, hashCode);

// линейный поиск по бакету
```

Поиск

```
var buckets = _buckets;
var entries = _entries;
var hashCode = (uint)key.GetHashCode();

int bucket = ref GetBucket(buckets, hashCode);

// линейный поиск по бакету
```

Модель памяти*

Conforming implementations of the CLI are **free to execute** programs **using any technology** that guarantees, within a **single thread of execution**, that side-effects and exceptions generated by a thread **are visible in the order specified by the CIL**.

*ECMA-335 1.12.6.4

Добавление

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(buckets, hashCode);
ref Entry entry = ref entries[count];

entry.hashCode = hashCode;
entry.next = bucket;

entry.key = key;
entry.value = value;
bucket = count;
```

Добавление

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(buckets, hashCode);
ref Entry entry = ref entries[count];

entry.hashCode = hashCode;
entry.next = bucket;
bucket = count;
entry.key = key;
entry.value = value;
```

Расширение

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(buckets, hashCode);
ref Entry entry = ref entries[count];

ResizeIfNeeded();

// запись в entry
```

Расширение

```
void ResizeIfNeeded()
{
    Array.Copy(entries, newEntries);

    // расположение в новые бакеты

    entries = newEntries;
    buckets = newBuckets;
}
```

Гонка при расширении

```
void ResizeIfNeeded()
{
    Array.Copy(entries, newEntries);

    // расположение в новые бакеты

    entries = newEntries;
    buckets = newBuckets;
}
```

Гонка при расширении

```
TValue TryGetValue(Tkey key)
{
    var entries = _entries;
    var buckets = _buckets;
    var hashCode = (uint)key.GetHashCode();

    int bucket = ref GetBucket(buckets, hashCode);

    // линейный поиск по бакету
}
```

count для перечисления

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(buckets, hashCode);
ref Entry entry = ref entries[count];


// запись в entry

bucket = count;
count++;
```

count для перечисления

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(buckets, hashCode);
ref Entry entry = ref entries[count];

count++;
// запись в entry

bucket = count;
```

Боремся с гонками

Решаем проблему при расширении

```
ResizeIfNeeded()
{
    Array.Copy(entries, newEntries);

    // расположение в новые бакеты

    entries = newEntries;
    buckets = newBuckets;

}
```

Решаем проблему при расширении

```
ResizeIfNeeded()
{
    Array.Copy(entries, newEntries);

    // расположение в новые бакеты

-   entries = newEntries;
-   buckets = newBuckets;
+   tables = new Tables(newEntries, newBuckets);
}
```

Решаем проблему при расширении

```
TValue TryGetValue(Tkey key)
{
    var entries = _entries;
    var buckets = _buckets;

    var hashCode = (uint)key.GetHashCode();

    int bucket = ref GetBucket(buckets, hashCode);


    // линейный поиск по бакету
}
```

Решаем проблему при расширении

```
TValue TryGetValue(Tkey key)
{
-   var entries = _entries;
-   var buckets = _buckets;
+   var tables = _tables;
    var hashCode = (uint)key.GetHashCode();

-   int bucket = ref GetBucket(buckets, hashCode);
+   int bucket = ref GetBucket(tables.buckets, hashCode);

    // линейный поиск по бакету
}
```

Решаем проблему при расширении

```
TValue TryGetValue(Tkey key)
{
    var tables = _tables;
    var hashCode = (uint)key.GetHashCode();

    int bucket = ref GetBucket(tables.buckets, hashCode);

    // линейный поиск по бакету
}
```

Оптимизации компилятора

- instructions reordering

Q: завершится ли?

```
static int data = 0;

void Run()
{
    Task.Delay(1).ContinueWith(_ => data = 1);


    var iterations = 0;
    while (data == 0)
        iterations++;

    Console.WriteLine(iterations);
}
```

Q: завершится ли?

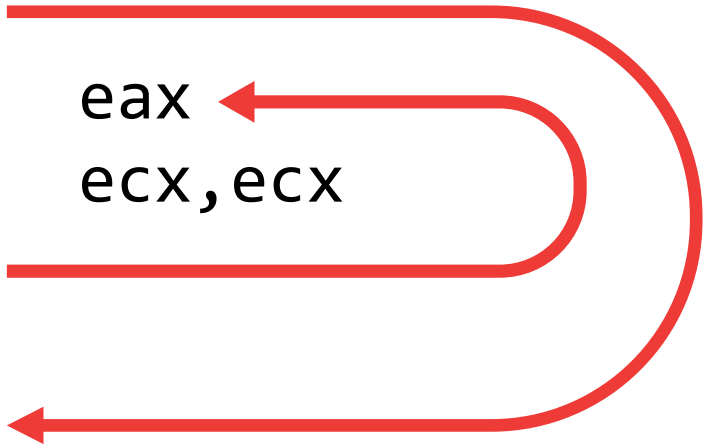
```
static int data = 0;

void Run()
{
    Task.Delay(1).ContinueWith(_ => data = 1);


    var iterations = 0;
    while (data == 0)
        iterations++;

    Console.WriteLine(iterations);
}
```

Как сделает компилятор

```
static int data = 0;

void Run()
{
    Task.Delay(1).ContinueWith(_ => data = 1);

    var dataLocal = data;
    var iterations = 0;
    while (dataLocal == 0)
        iterations++;

    Console.WriteLine(iterations);
}
```

Во что скомпилируется

```
mov   ecx,dword ptr [rsi+8]

test  ecx,ecx

jne
inc   eax
test  ecx,ecx
je
...
ret
```



```
ecx = data

if (ecx != 0)
   return 0;

do {
   eax++
} while (ecx != 0);

return eax;
```

Оптимизации компилятора

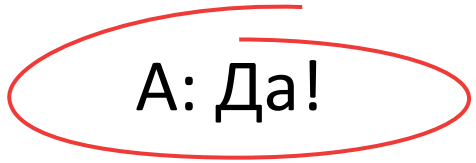
- instructions reordering
- loop read hoisting

Q: сломается, если зависит от порядка чтения переменных?

```
public void Foo()
{

   if (B < 0)
      throw new Exception();

   var a = A;
   var b = B;

   // your algorithm
}
```

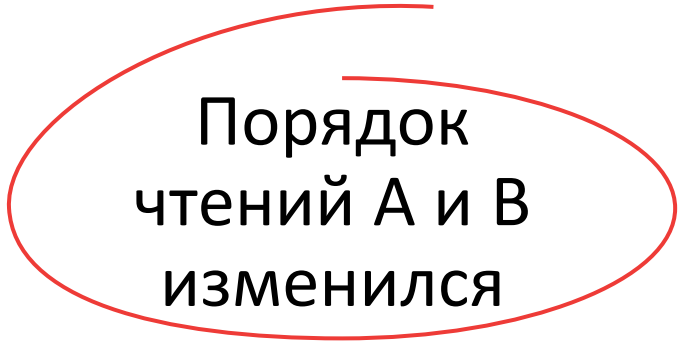
Q: сломается, если зависит от порядка чтения переменных?

```
public void Foo()
{

   if (B < 0)
      throw new Exception();

   var a = A;
   var b = B;

   // your algorithm
}
```



Как сделает компилятор

```
public void Foo()
{
   var bLocal = B;
   if (bLocal < 0)
      throw new Exception();

   var a = A;
   var b = bLocal;

   // your algorithm
}
```



Оптимизации компилятора

- instructions reordering
- loop read hoisting
- read elimination

Q: возможен NullReferenceException?

```
private Object _obj = new Object();

void PrintObj() {
    Object obj = _obj;
    if (obj != null)
        Console.WriteLine(obj.ToString());
}

void Uninitialize() {
    _obj = null;
}
```

Q: возможен NullReferenceException?

```
private Object _obj = new Object();

void PrintObj() {
    Object obj = _obj;
    if (obj != null)
        Console.WriteLine(obj.ToString());
}

void Uninitialize() {
    _obj = null;
}
```

A: Да!

Как сделает компилятор

```
private Object _obj = new Object();

void PrintObj() {

    if (_obj != null)
        Console.WriteLine(_obj.ToString());
}

void Uninitialize() {
    _obj = null;
}
```

Оптимизации компилятора

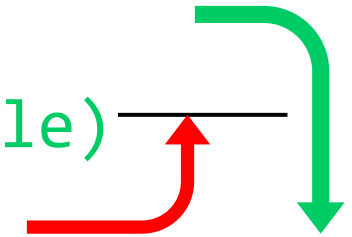
- instructions reordering
- loop read hoisting
- read elimination
- read introduction

Про volatile

- instructions reordering
- ~~loop read hoisting~~
- ~~read elimination~~
- ~~read introduction~~

Acquire семантика*

```
int _a;
volatile int _b;
int _c;

void Foo()
{
    int a = _a; // Read 1
    int b = _b; // Read 2(volatile)
    int c = _c; // Read 3
}
```



*ECMA-334 15.5.4
ECMA-335 1.12.6.7

Acquire семантика*

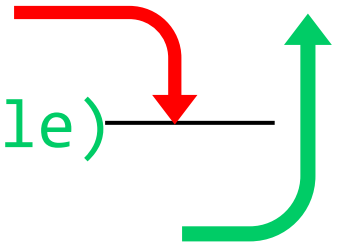
```
int _a;
volatile int _b;
int _c;

void Foo()
{
    int b = _b; // Read 2(volatile)
    int a = _a; // Read 1
    int c = _c; // Read 3
}
```

Acquire семантика*

```
int _a;
volatile int _b;
int _c;

void Foo()
{
    int b = _b; // Read 2(volatile)
    int c = _c; // Read 3
    int a = _a; // Read 1
}
```

Release семантика*

```
int a;
volatile int b;
int c;

void Foo()
{
    a = 1; // Write 1
    b = 1; // Write 2(volatile)
    c = 1; // Write 3
}
```



*ECMA-334 15.5.4
ECMA-335 1.12.6.7

Release семантика*

```
int a;
volatile int b;
int c;

void Foo()
{
    a = 1; // Write 1
    c = 1; // Write 3
    b = 1; // Write 2(volatile)
}
```

Release семантика*

```
int a;
volatile int b;
int c;

void Foo()
{
    c = 1; // Write 3
    a = 1; // Write 1
    b = 1; // Write 2(volatile)
}
```

Решаем проблему с перестановками

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(tables, hashCode);
ref Entry entry = ref tables.entries[count];

ResizeIfNeeded();

// обновляем entry

bucket = count;
count++;
```

Решаем проблему с перестановками

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(tables, hashCode);
ref Entry entry = ref tables.entries[count];

ResizeIfNeeded();

// обновляем entry

bucket = count;
count++; // volatile write
```

Решаем проблему с перестановками

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(tables, hashCode);
ref Entry entry = ref tables.entries[count];

ResizeIfNeeded();

// обновляем entry

bucket = count;
count++; // volatile write
```

Решаем проблему с перестановками

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(tables, hashCode);
ref Entry entry = ref tables.entries[count];

ResizeIfNeeded();

// обновляем entry

Volatile.Write(ref bucket, count);
count++; // volatile write
```

Внутри Volatile.Write()

```
void Write(ref int location, int value)
{
    Thread.MemoryBarrier();
    location = value;
}
```

Разные архитектуры процессоров

Memory ordering in some architectures[8][9]

Type	Alpha	ARMv7	MIPS	RISC-V		PA-RISC	POWER	SPARC			x86 [a]	AMD64	IA-64	z/Architecture
				WMO	TSO			RMO	PSO	TSO				
Loads can be reordered after loads	Y	Y	depend on implementation	Y		Y	Y	Y					Y	
Loads can be reordered after stores	Y	Y		Y		Y	Y	Y					Y	
Stores can be reordered after stores	Y	Y		Y		Y	Y	Y	Y				Y	
Stores can be reordered after loads	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Atomic can be reordered with loads	Y	Y		Y			Y	Y					Y	
Atomic can be reordered with stores	Y	Y		Y			Y	Y	Y				Y	
Dependent loads can be reordered	Y													
Incoherent instruction cache/pipeline	Y	Y		Y	Y		Y	Y	Y	Y	Y		Y	

https://en.wikipedia.org/wiki/Memory_ordering

Перестановки в разных архитектурах

	X86, X64	ARM
LOAD - LOAD	**Нет**	**Да**
LOAD - STORE	**Нет**	**Да**
STORE – STORE	**Нет**	**Да**
STORE - LOAD	**Да**	**Да**

Operations without reordering

```
volatile int _a;
volatile int _b;

{
    _a = 1;
    _b = 1;
}
```

```
volatile int _a;
volatile int _b;

{
    var a = _a;
    var b = _b;
}
```

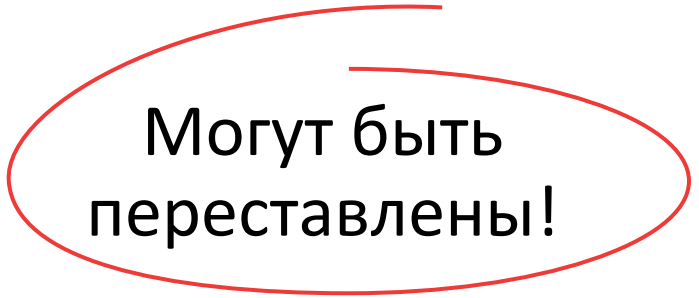
```
volatile int _a;
volatile int _b;

{
    var a = _a;
    _b = 1;
}
```

Volatile write-read reordering

```
volatile int _a;
volatile int _b;

void Foo()
{
    _a = 1; // Write (volatile)

    var b = _b; // Read (volatile)
}
```

Volatile write-read reordering

```
volatile int _a;
volatile int _b;

void Foo()
{
    _a = 1; // Write (volatile)

    var b = _b; // Read (volatile)
}
```



Архитектура x86, x64

- Чтения и записи имею acuire/release-семантику
- JIT не генерирует дополнительных инструкций для *volatile*-полей
- Префикс *lock* (aka full-fence) для:
  - Thread.MemoryBarrier()
  - Interlocked.[Exchange|CompareExchange]()
  - …

Перестановки в разных архитектурах

	X86, X64	ARM
LOAD - LOAD	**Нет**	**Да**
LOAD - STORE	**Нет**	**Да**
STORE – STORE	**Нет**	**Да**
STORE - LOAD	**Да**	**Да**

Архитектура ARM

- JIT может* генерировать инструкцию *DMB* для:
  - volatile fields
  - Thread.MemoryBarrier()
  - Interlocked.[Exchange|CompareExchange]()
  - …
- *Для ARM64 (ARMv8) *LDAR STLR*

*ARM Barriers

Работающее добавление

```
var hashCode = (uint)key.GetHashCode();

ref int bucket = ref GetBucket(tables, hashCode);
ref Entry entry = ref tables.entries[count];

ResizeIfNeeded();

// обновляем entry

Volatile.Write(ref bucket, count);
count++; // volatile write
```

Результаты

Бенчмарки

```
BenchmarkDotNet=v0.13.1
OS=Windows 10.0.19043.1415 (21H1/May2021Update)
AMD Ryzen 7 4700U with Radeon Graphics
     1 CPU, 8 logical and 8 physical cores
.NET SDK=5.0.303
[Host] : .NET Core 3.1.22, X64 RyuJIT
```

Бенчмарки

Fill

Method	Mean	Allocated
AppendOnly_without_initial_capacity	24.632 ms	43 MB
Concurrent_without_initial_capacity	99.960 ms	50 MB
AppendOnly_with_initial_capacity	6.486 ms	15 MB
Concurrent_with_initial_capacity	49.816 ms	23 MB

Бенчмарки

Fill

Method	Mean	Allocated
AppendOnly_without_initial_capacity	24.632 ms	43 MB
Concurrent_without_initial_capacity	99.960 ms	50 MB
AppendOnly_with_initial_capacity	6.486 ms	15 MB
Concurrent_with_initial_capacity	49.816 ms	23 MB

Бенчмарки

Fill

Method	Mean	Allocated
AppendOnly_without_initial_capacity	24.632 ms	43 MB
Concurrent_without_initial_capacity	99.960 ms	50 MB
AppendOnly_with_initial_capacity	6.486 ms	15 MB
Concurrent_with_initial_capacity	49.816 ms	23 MB

Бенчмарки

TryGetValue_Missing

Method	Mean	Allocated
AppendOnly	3.762 ns	-
Concurrent	9.102 ns	-

TryGetValue_Existing

Method	Mean	Allocated
AppendOnly	5.325 ns	-
Concurrent	13.277 ns	-

Бенчмарки

TryGetValue_Missing

Method	Mean	Allocated
AppendOnly	3.762 ns	-
Concurrent	9.102 ns	-

TryGetValue_Existing

Method	Mean	Allocated
AppendOnly	5.325 ns	-
Concurrent	13.277 ns	-

Бенчмарки

TryGetValue_Missing

Method	Mean	Allocated
AppendOnly	3.762 ns	-
Concurrent	9.102 ns	-

TryGetValue_Existing

Method	Mean	Allocated
AppendOnly	5.325 ns	-
Concurrent	13.277 ns	-

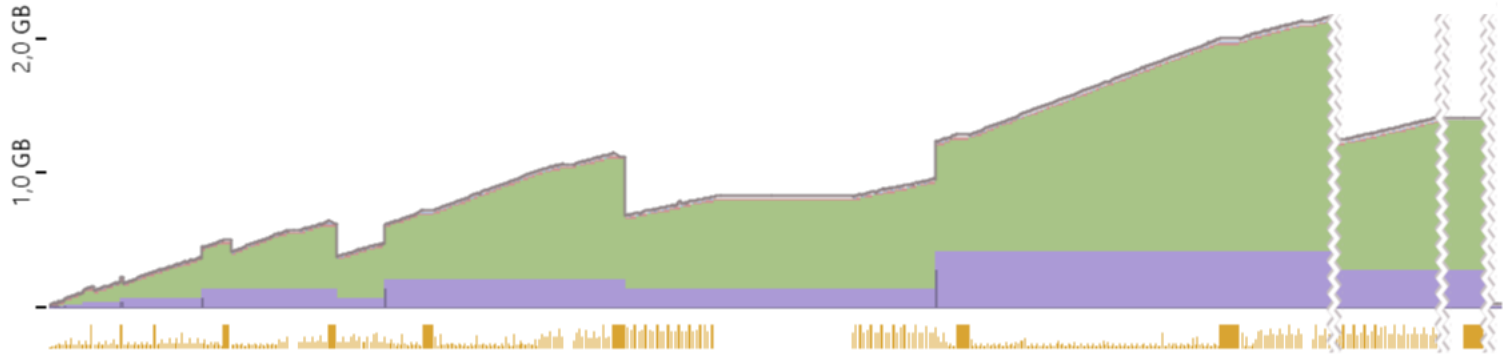
Бенчмарки

```
Enumeration
|       Method |         Mean | Allocated |
|------------- |-------------:|----------:|
|   AppendOnly |   719.8 us |       64 B |
|   Concurrent | 1,199.1 us |       64 B |
```

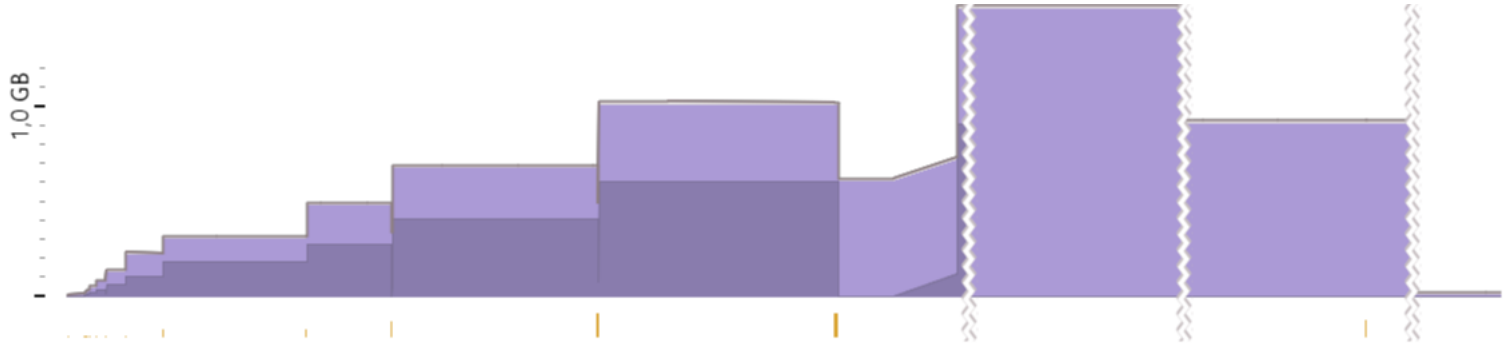
Бенчмарки

```
Enumeration
|        Method |         Mean | Allocated |
|-------------- |-------------:|----------:|
|    AppendOnly |    719.8 us |       64 B |
|    Concurrent |  1,199.1 us |       64 B |
```

Работа с памятью



ConcurrentDictionary



AppendOnlyDictionary

Heap generation 0

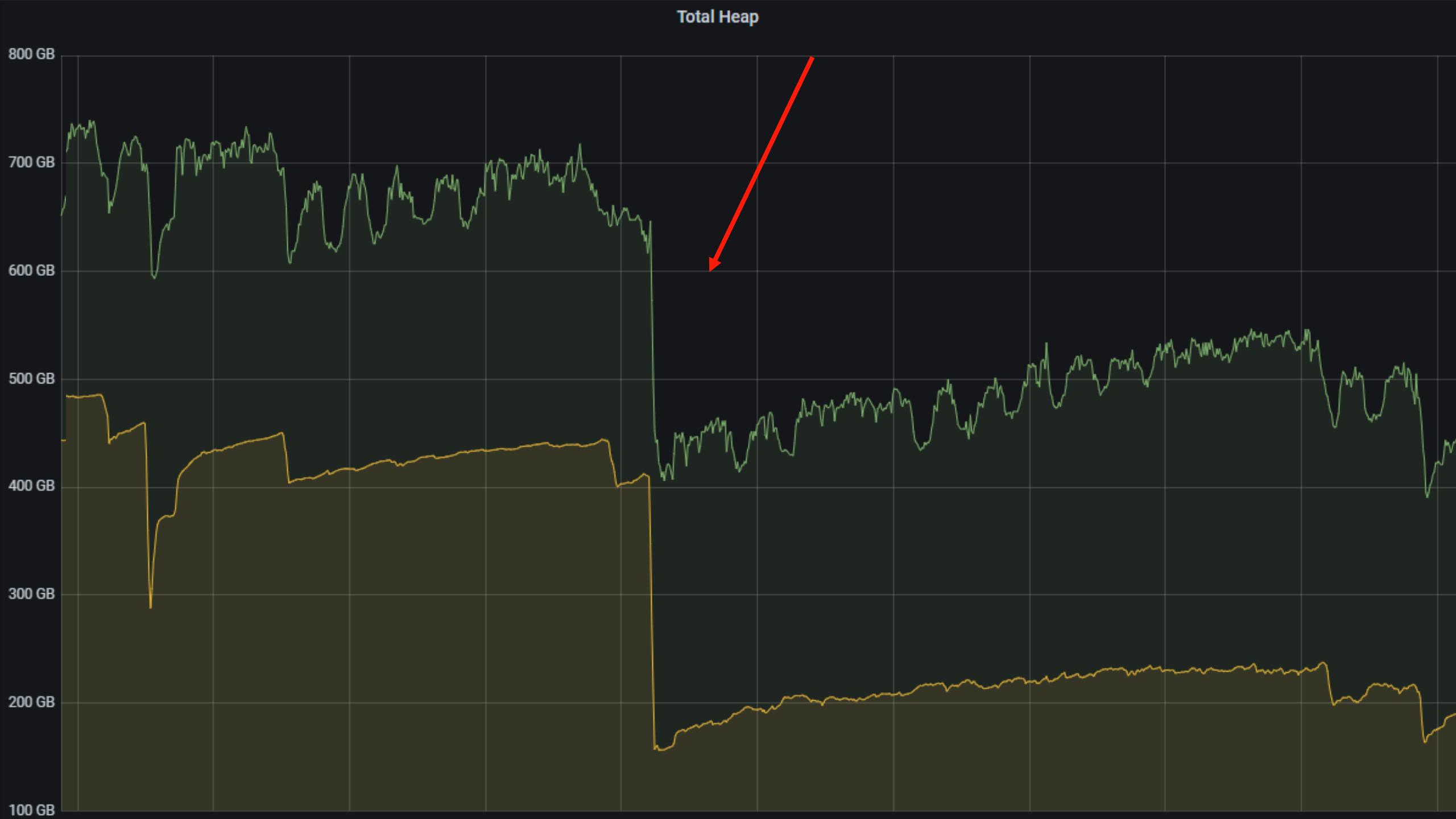
Heap generation 1

Heap generation 2

LOH and POH

Сравнение с ConcurrentDictionary

	Lock-free reads	Lock-free single-writer	Objects overhead
ConcurrentDictionary	**Да**	**Нет**	**Да**
AppendOnlyDictionary	**Да**	**Да**	**Нет**


Total Heap
800 GB
700 GB
600 GB
500 GB
400 GB
300 GB
200 GB
100 GB

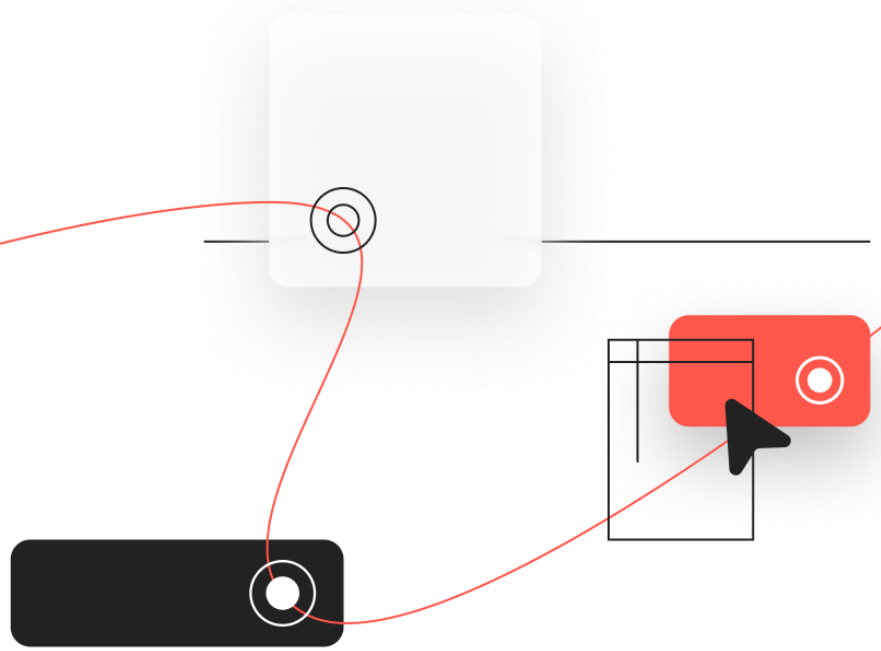
Напоследок

- Пробуй писать своё, если готовое не устраивает
- Решай конкретную задачу
- Будет непросто, но интересно!

Что почитать?

- Стандарт ECMA-334 (C# Language Specification)
- Стандарт ECMA-335 (Common Language Infrastructure)
- C# Memory Model in Theory and Practice pt. 1
- C# Memory Model in Theory and Practice pt. 2
- Сборка статей и докладов про memory model и volatile

Вопросы

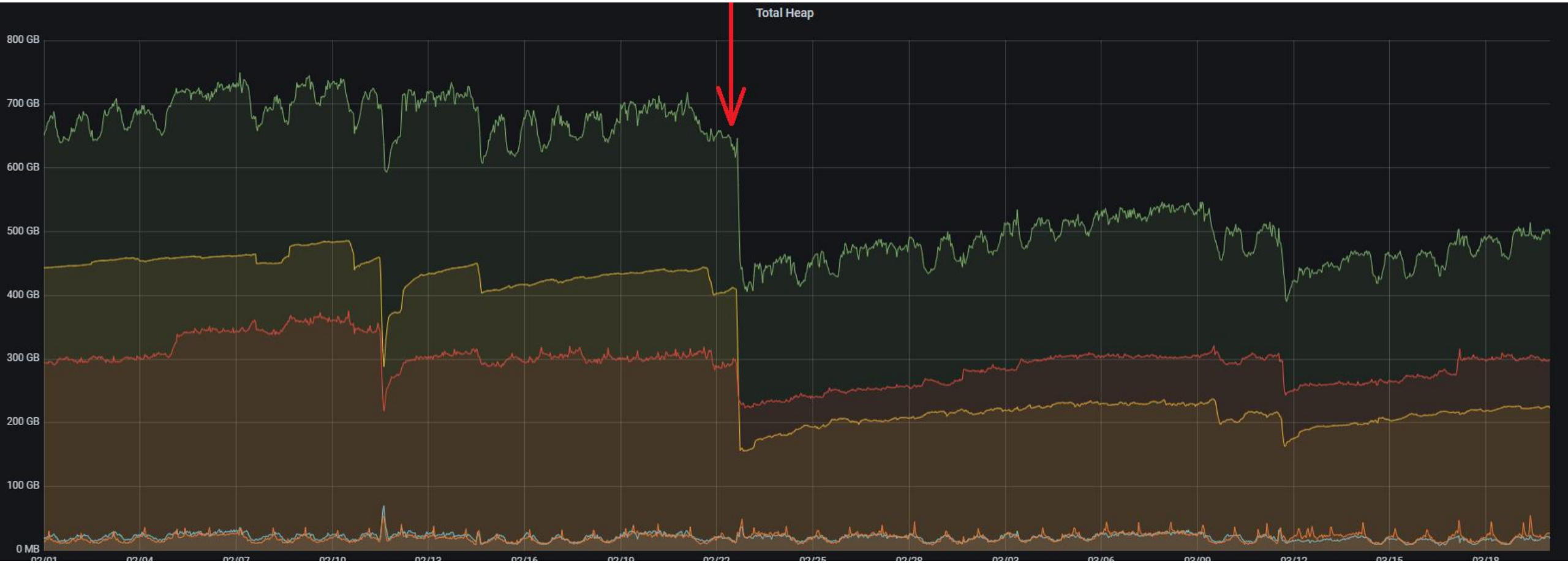


Нечуговских Антон

e-mail: nechugovskikh@kontur.ru

telegram: @ryzhes

Контур


Total Heap
800 GB
700 GB
600 GB
500 GB
400 GB
300 GB
200 GB
100 GB
0 MB