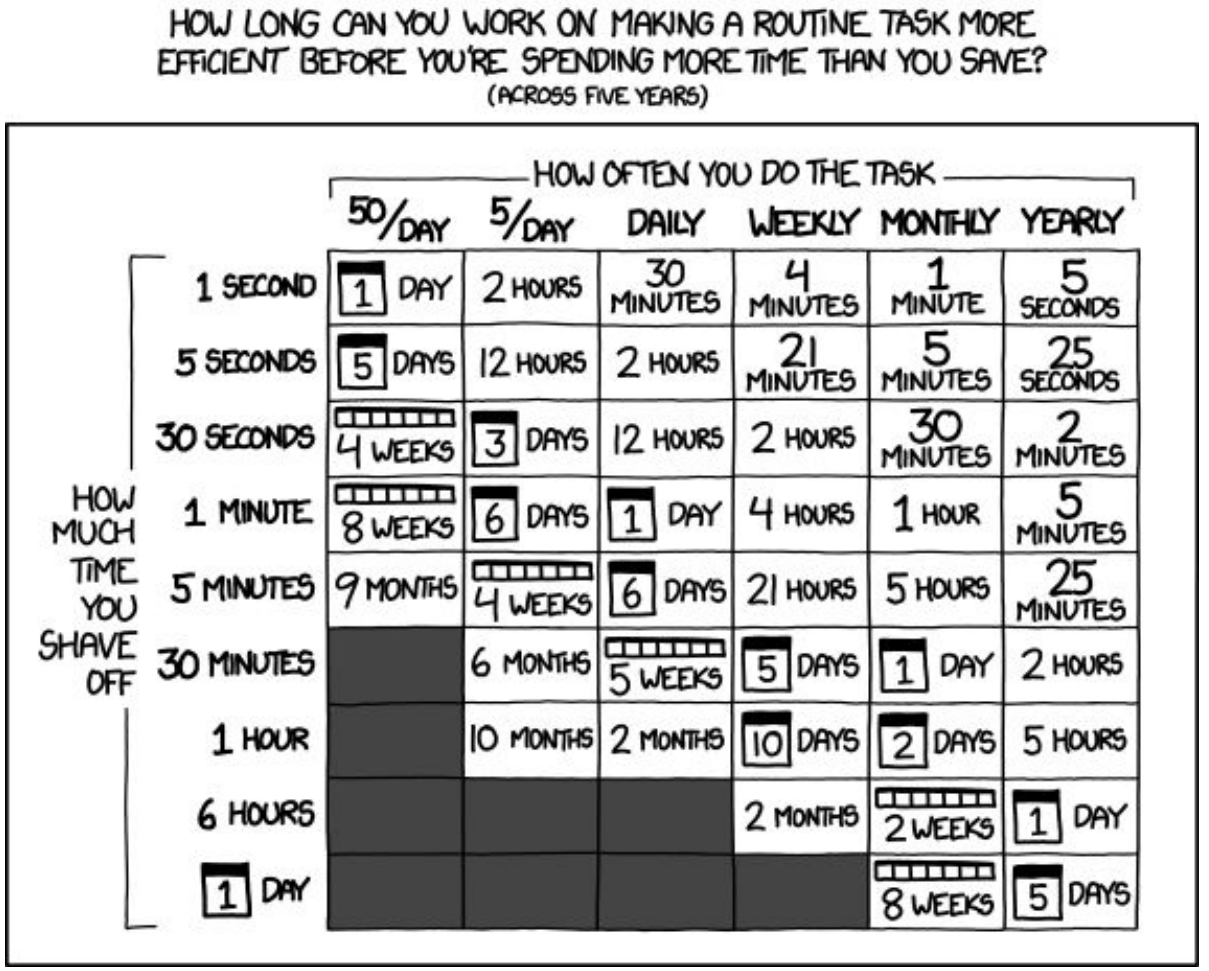
.Net Performance

Tips & Tricks

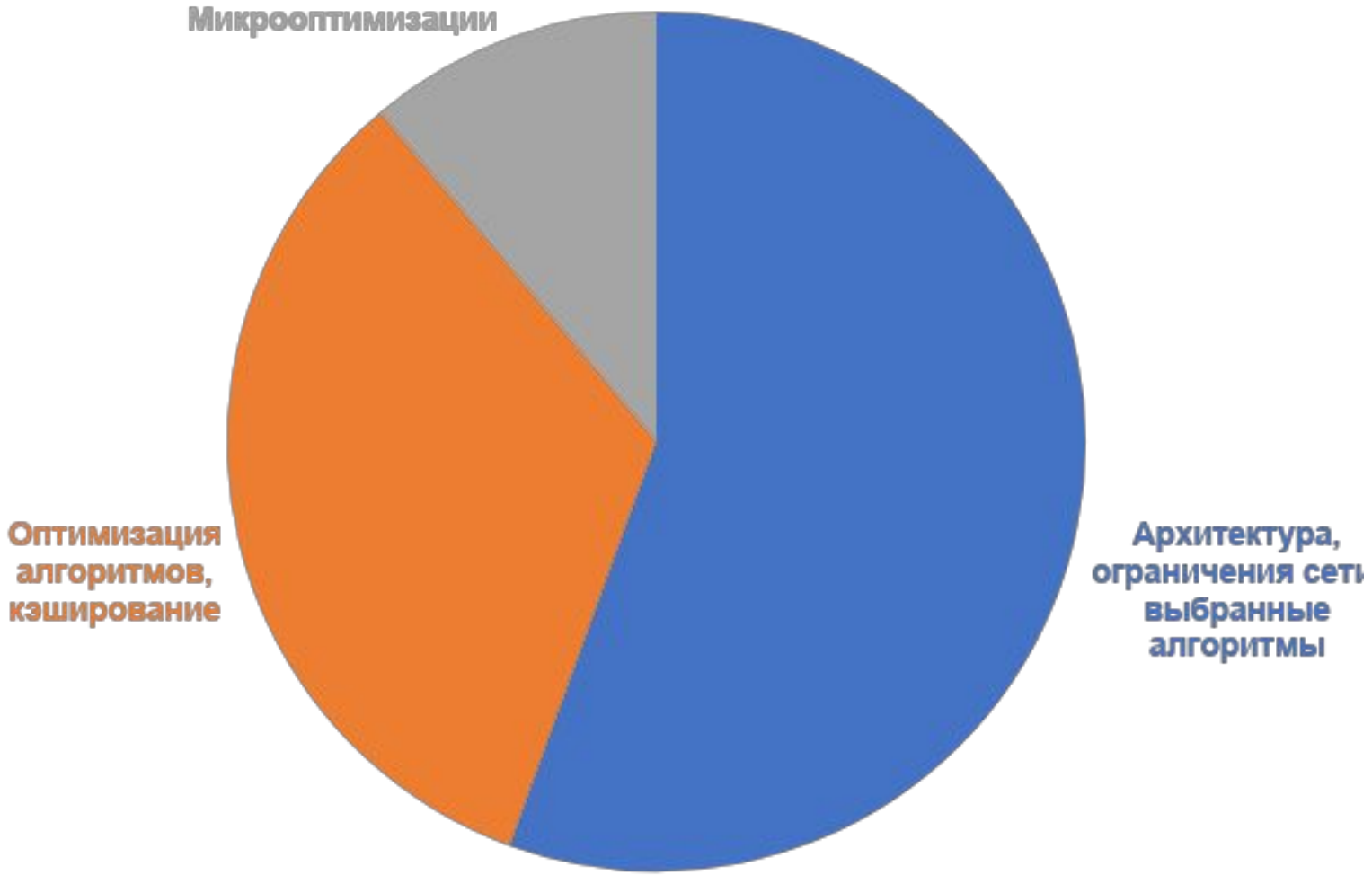
Сафин Рустам

DIRECTUM

Зачем нужна производительность?



Как понять, на каком уровне проблема?



Инструменты

Visual Studio profiler

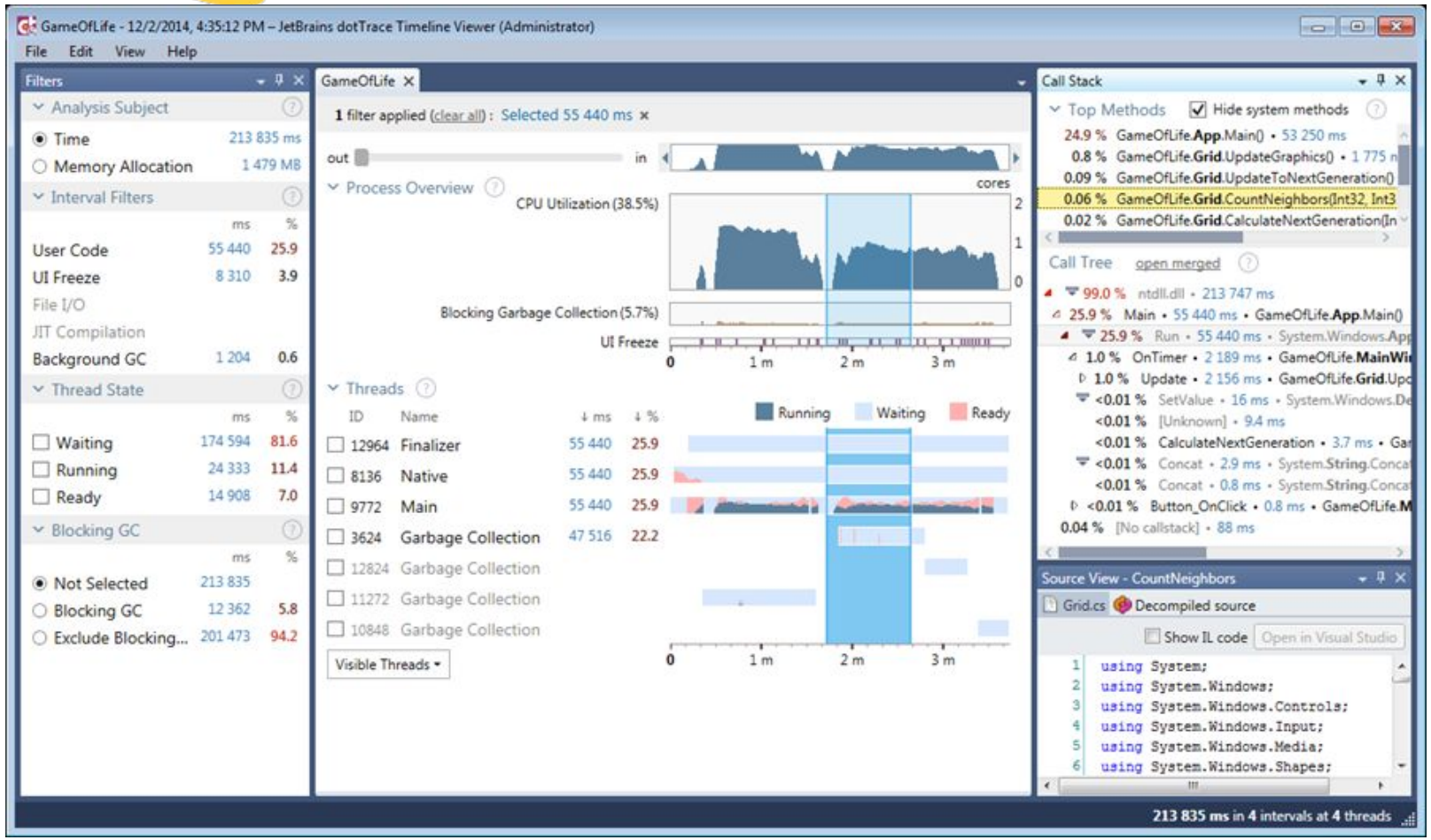
dotTrace

dotMemory

BenchmarkDotNet

CodeTrack

CLR Profiler

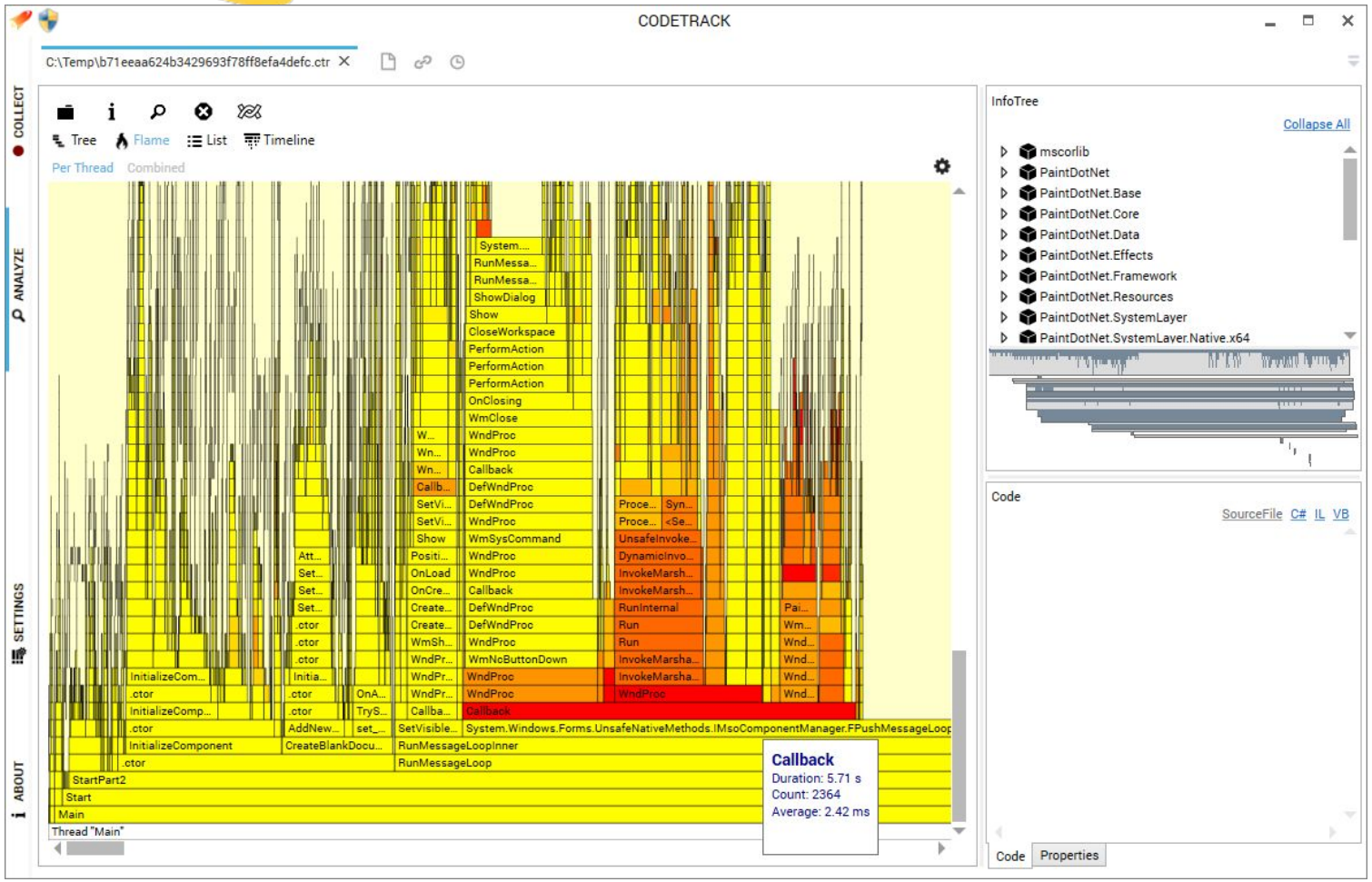


Инструменты

BenchmarkDotNet

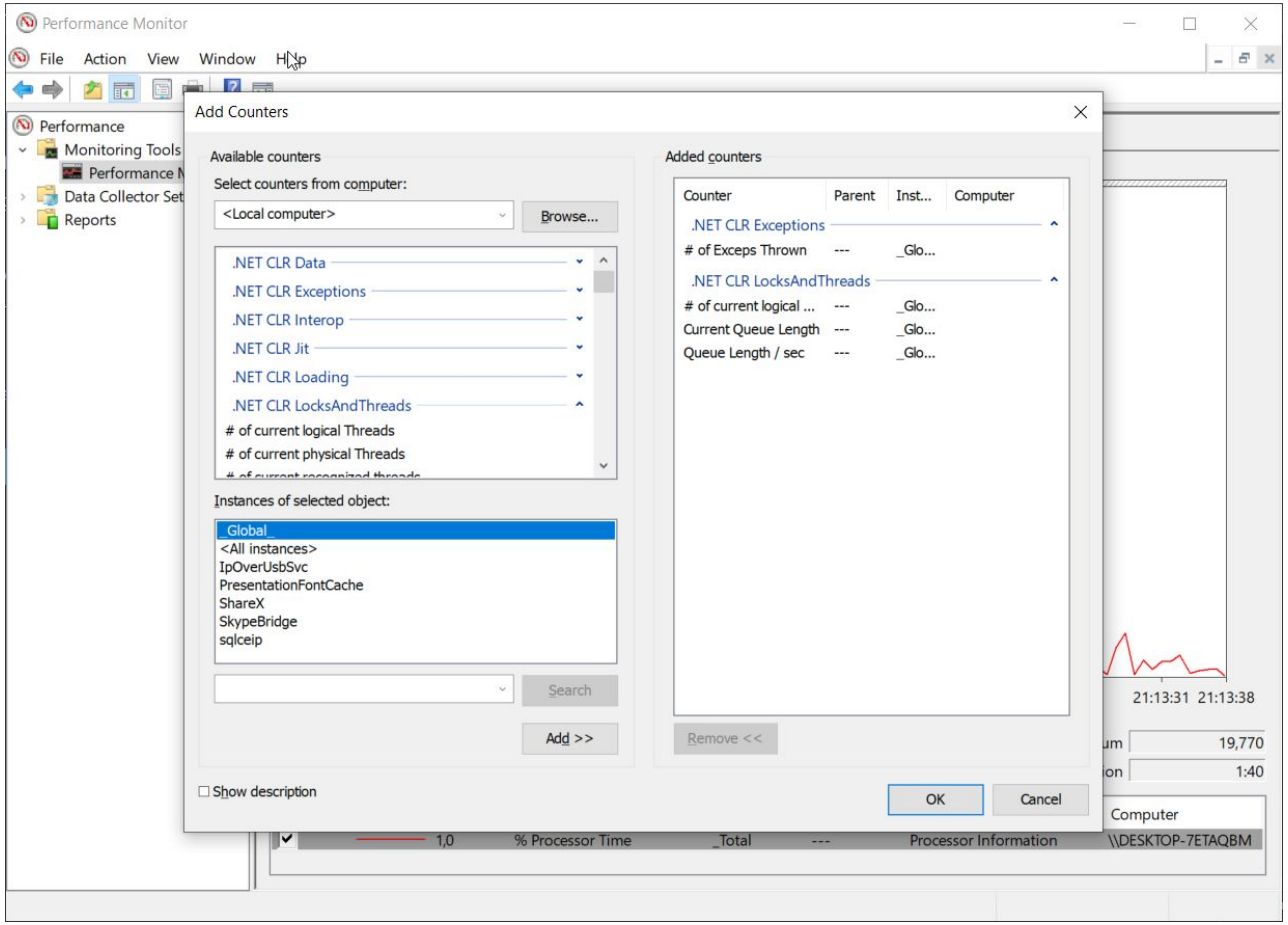
CodeTrack

CLR Profiler



Инструменты

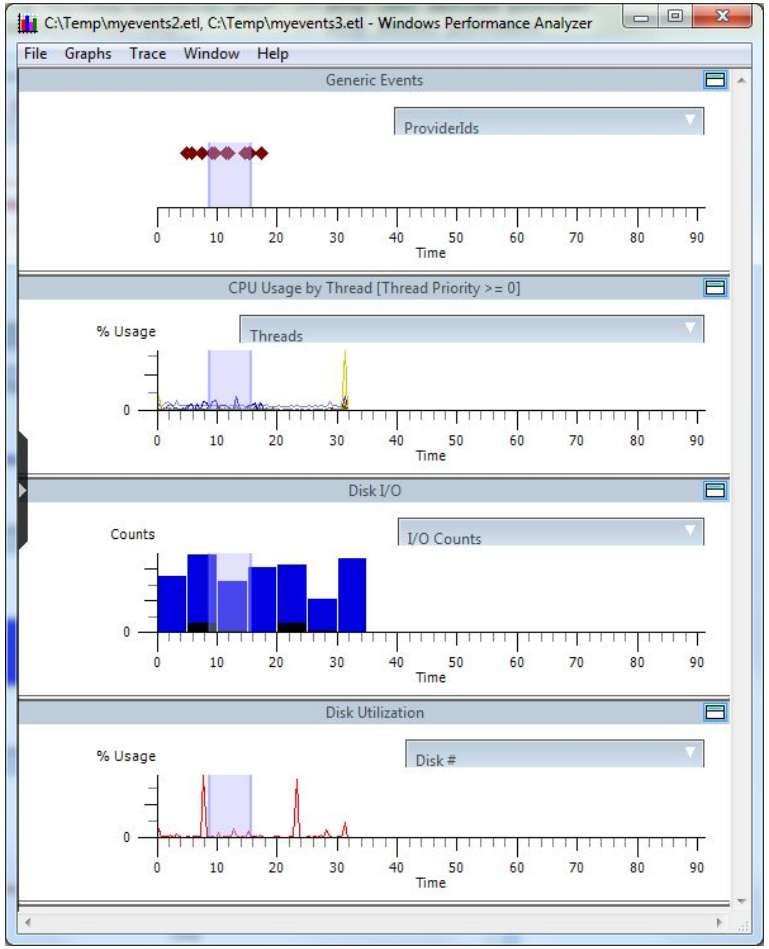
Performance Monitor



Инструменты

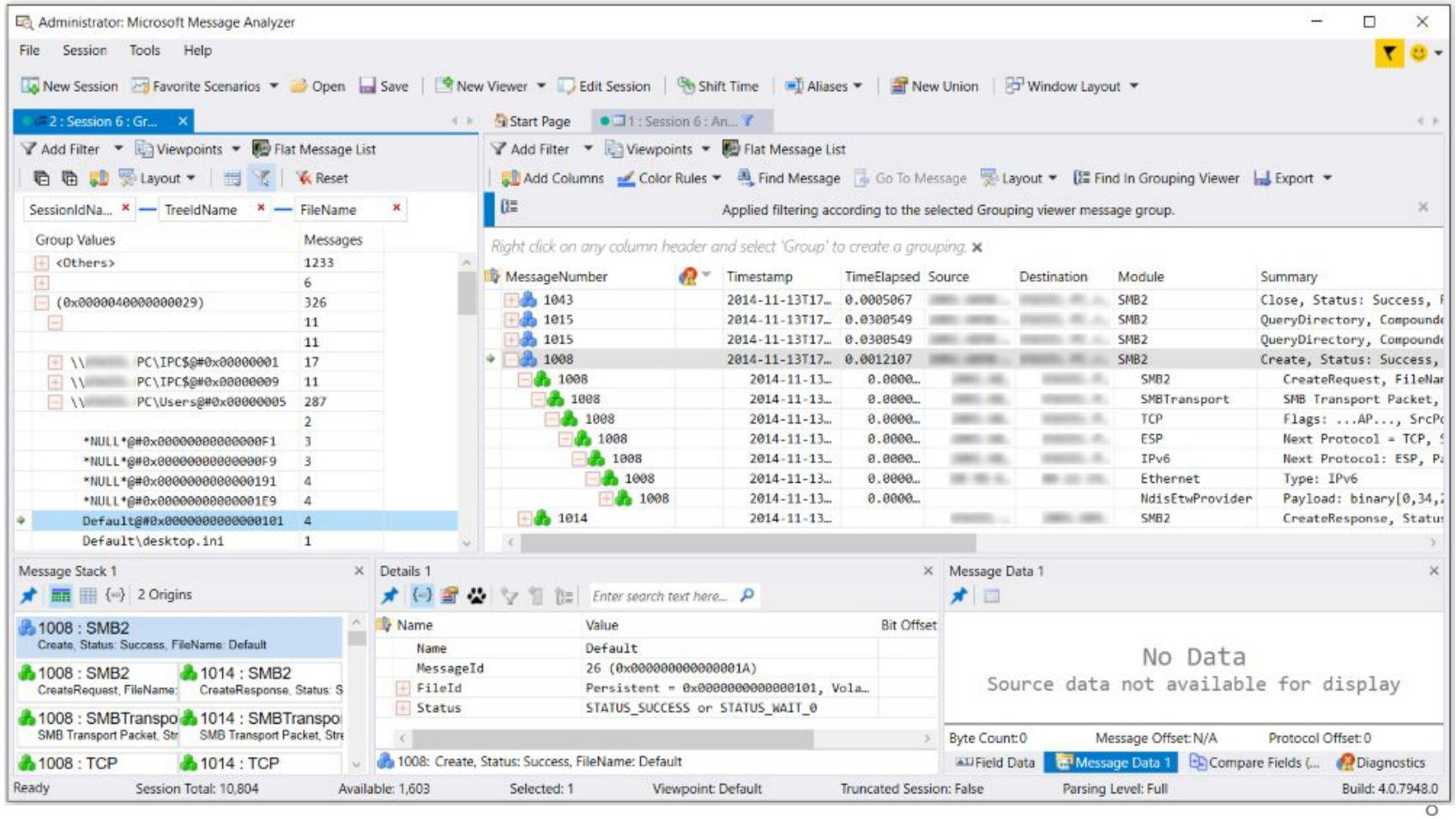
ETW

Windows Performance Analyzer

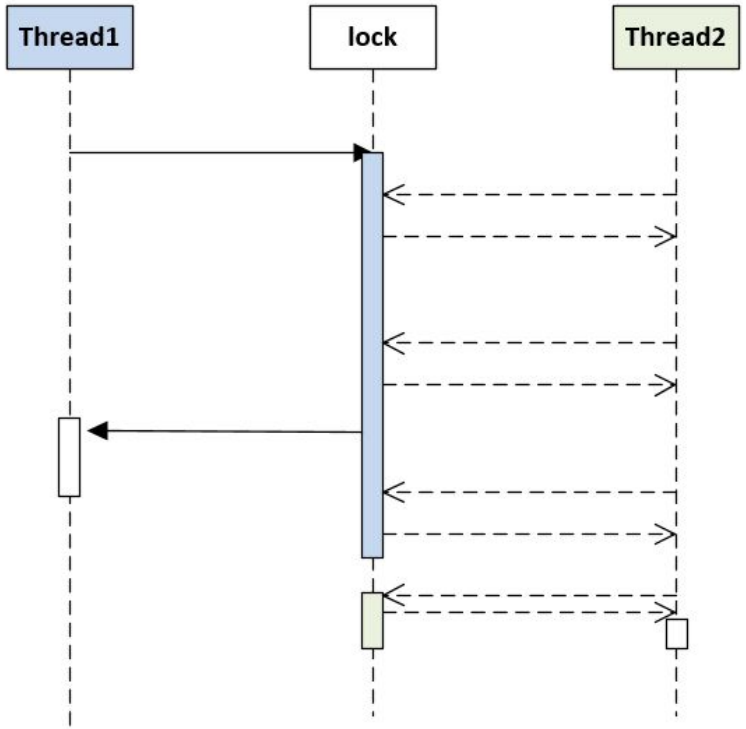


Инструменты

MS Message Analyzer



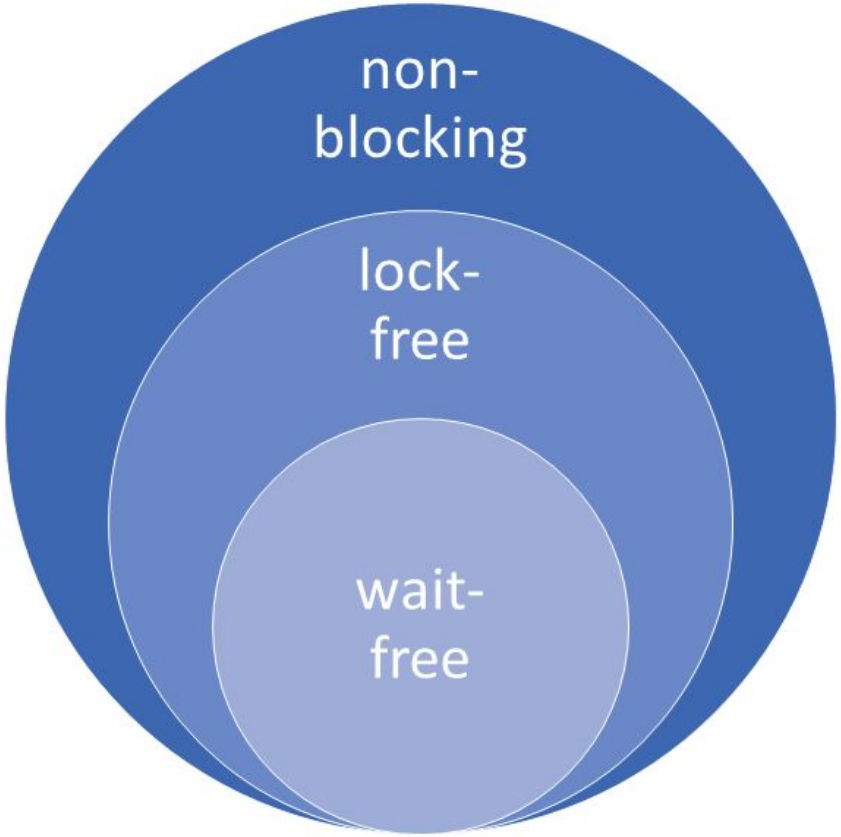
Lock-free & Wait-free



A Comparison of Locking Constructs

Construct	Overhead*
lock (**Monitor.Enter / Monitor.Exit**)	20ns
Mutex	1000ns
SemaphoreSlim (introduced in Framework 4.0)	200ns
Semaphore	1000ns
ReaderWriterLockSlim (introduced in Framework 3.5)	40ns
ReaderWriterLock (effectively deprecated)	100ns

Lock-free & Wait-free

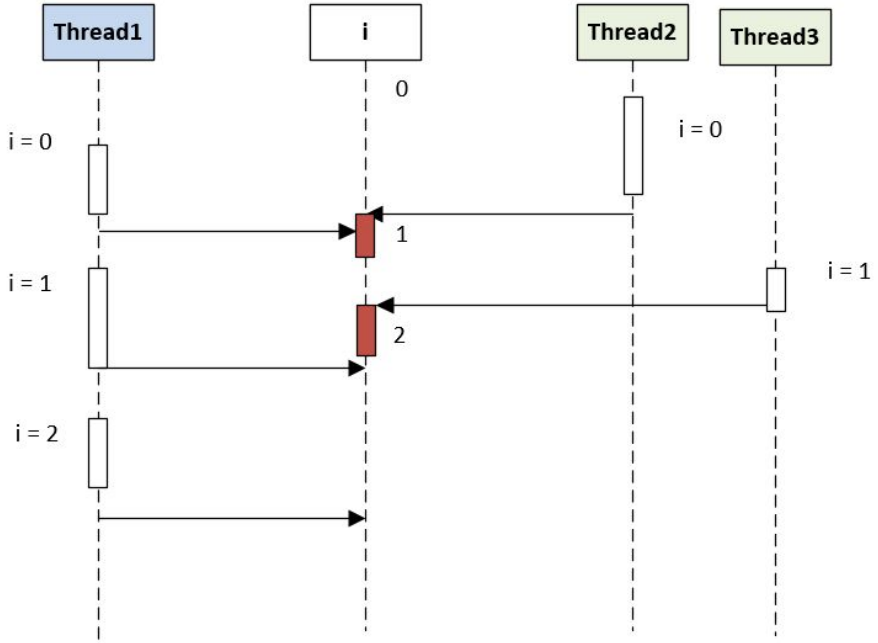


Lock-free & Wait-free

Compare and Swap

`Interlocked`

```
double CompareExchange (ref double location1, double value, double comparand);
```

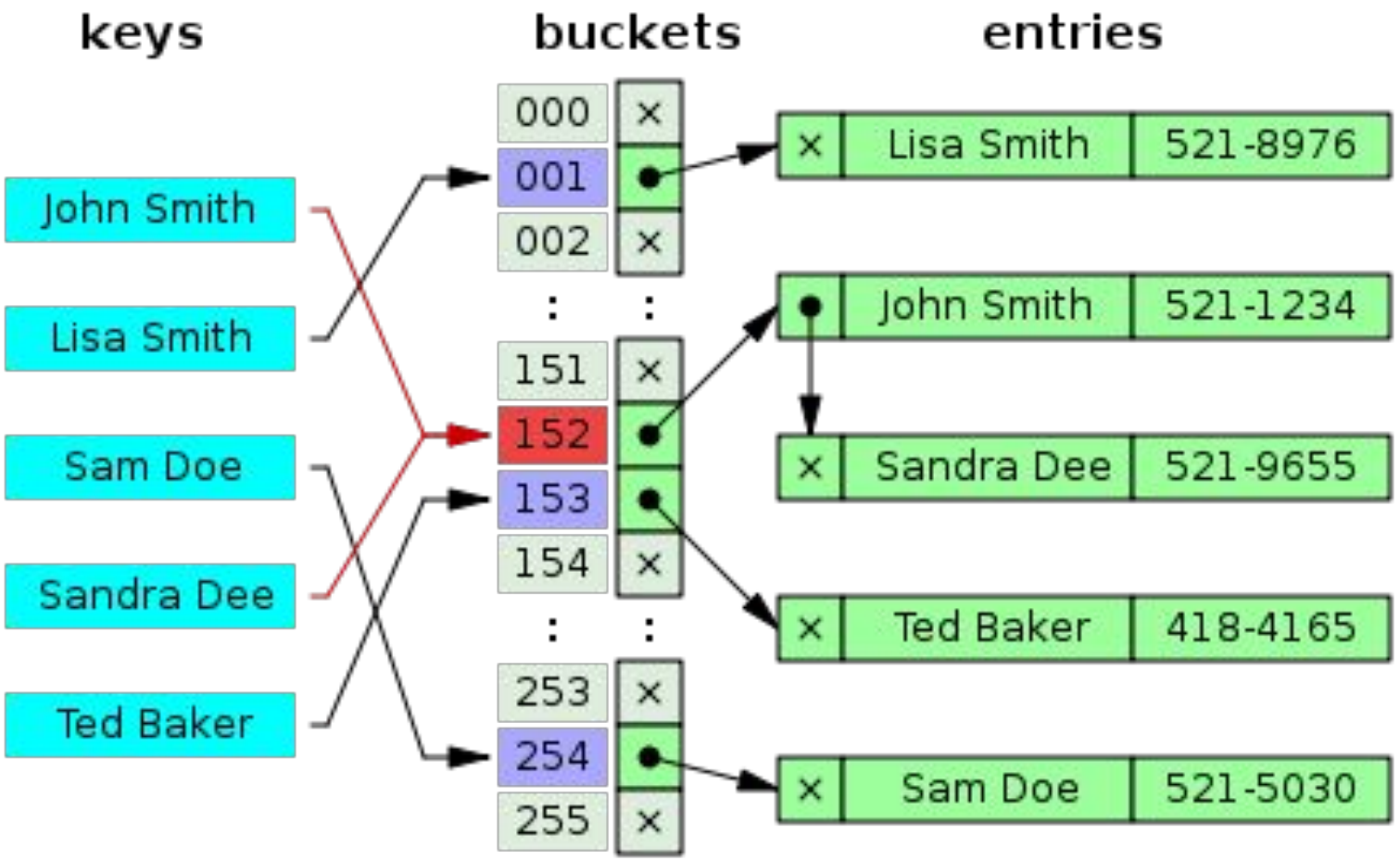



```
int InterlockedMultiply(ref int x, int y) {
  int t, r;
  do {
    t = x;
    r = t * y;
  }
  while (Interlocked.CompareExchange(ref x, r, t) != t);
  return r;
}
```

New Value

Comparand

ConcurrentDictionary



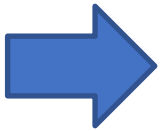
ConcurrentDictionary

Сильная деградация при попытке получить все блокировки

Method	Time	Memory	Locks
ContainsKey	O(1)	O(1)	none
TryGetValue	O(1)	O(1)	none
TryAdd	O(1)	O(1)	1
TryRemove	O(1)	O(1)	1
TryUpdate	O(1)	O(1)	1
Clear	O(K)	O(K)	K
Count	O(K)	O(1)	K
IsEmpty	O(K)	O(1)	K
Keys	O(N)	O(N)	K
Values	O(N)	O(N)	K
Enumeration	O(N)	O(1)	none

Single instruction, Multiple data

```
float[] values = GetValues();
float increment = GetIncrement();

for (int i = 0; i < values.Length; i++)
{
   values[i] += increment;
}
```

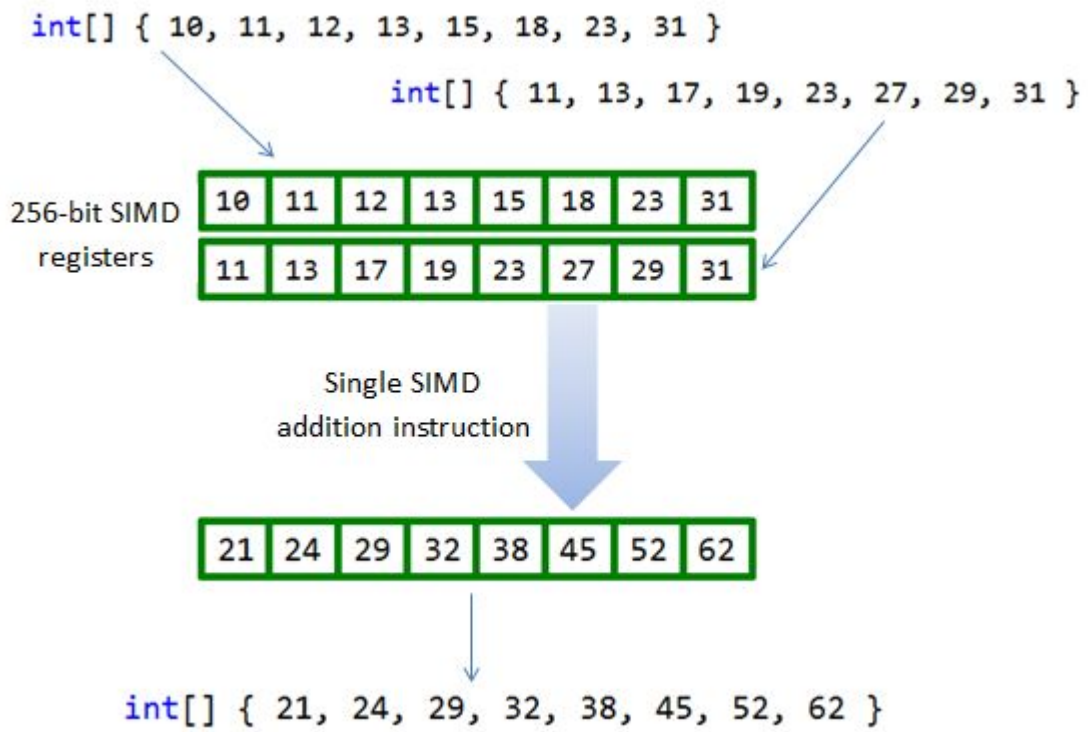


```
Vector<float> values = GetValues();
Vector<float> increment =
GetIncrement();

Vector<float> result = values +
increment;
```

Vector.IsHardwareAccelerated

SIMD

Single instruction, Multiple data



Task.Yield

```
async Task LongRunningCpuBoundWorkAsync()
{

    for (int i = 0; i != 1000000; ++i)
    {
        ... // CPU-bound work.
    }
}
```

Task.Yield

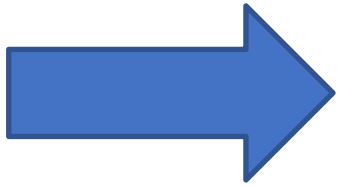
```
async Task LongRunningCpuBoundWorkAsync()
{

    for (int i = 0; i != 1000000; ++i)
    {
        ... // CPU-bound work.
        await Task.Yield();
    }
}
```

Long switch-case

```
switch (packetID)
{
  case BncsPacketId.Null:
    break;

  case BncsPacketId.EnterChat:
    string ecUniqueName = pck.ReadNTString();
    string ecStatstring = pck.ReadNTString();
    string ecAcctName = pck.ReadNTString();
    var args = new EventArgs(ecUniqueName, ecStatstring, ecAcctName);
    OnEnteredChat(args);
    break;

  ...
  // ну и так и далее
}
```

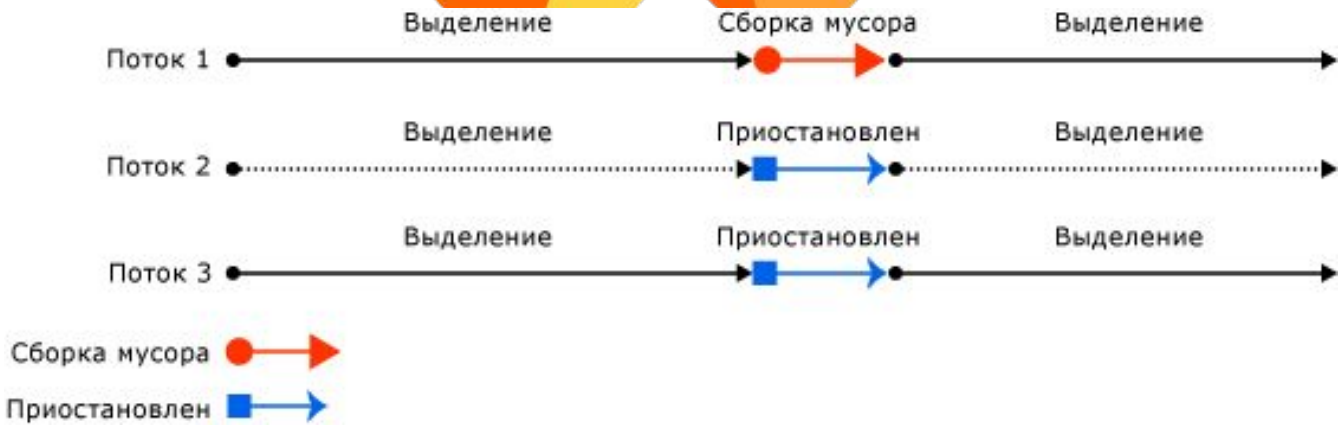
Long switch-case

```
switch (packetID)
{
  case BncsPacketId.Null:
    break;

  case BncsPacketId.EnterChat:
    string ecUniqueName = pck.ReadNTString();
    string ecStatstring = pck.ReadNTString();
    string ecAcctName = pck.ReadNTString();
    var args = new EventArgs(ecUniqueName, ecStatstring, ecAcctName);
    OnEnteredChat(args);
    break;

  ...
  // ну и так и далее
}
```

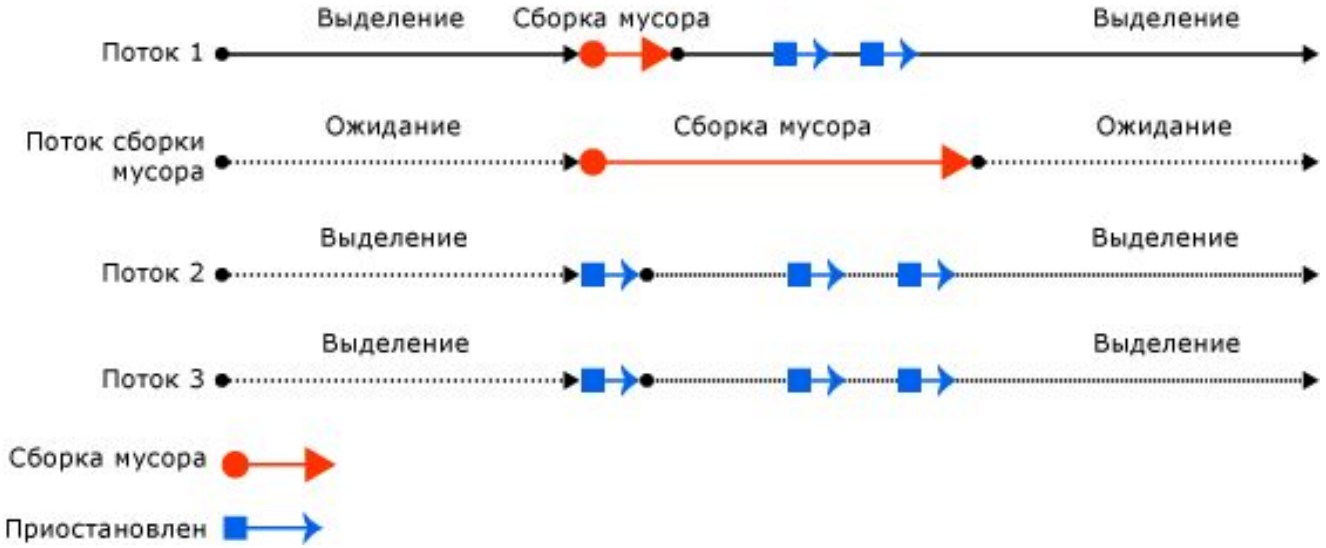


Dictionary<string, Action<T>>

Клиентский режим

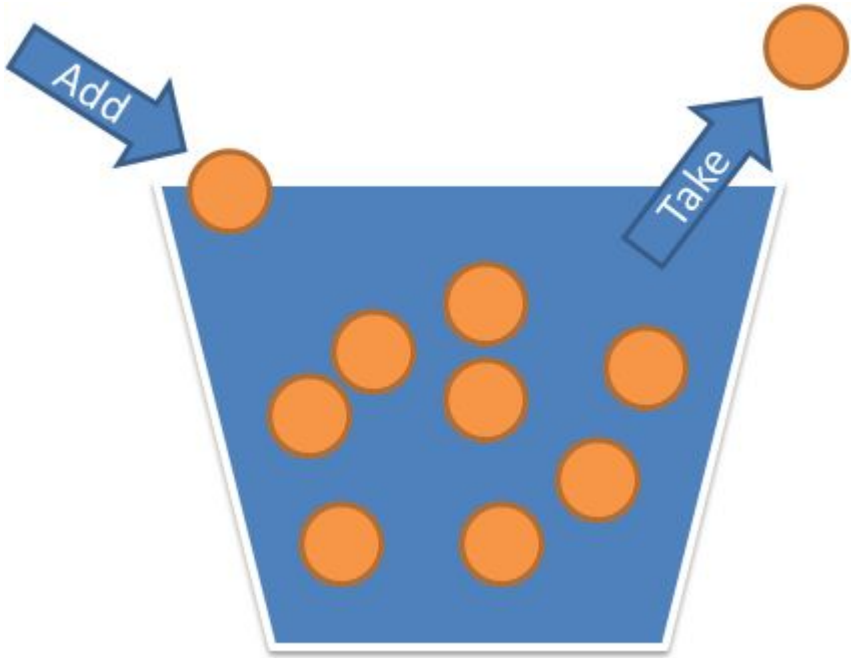


Серверный режим



ConcurrentBag

Оптимизация потребления памяти



async/await

```
private readonly Dictionary<string, string> cache = new Dictionary<string, string>();

public Task<string> GetValueAsync(string key)
{
  string result;
  if (cache.TryGetValue(key, out result))
    return Task.FromResult(result);
  return DoGetValueAsync(key);
}

private async Task<string> DoGetValueAsync(string key)
{
  HttpClient client = new HttpClient();
  var response =  await client.GetAsync(key);
  var text = await response.Content.ReadAsStringAsync();
  this.cache[key] = text;
  return text;
}
```

async/await

```
public struct TaskAwaiter<TResult>: INotifyCompletion
  {
    private readonly Task<TResult> m_task;

    internal TaskAwaiter(Task<TResult> task)
    {
      this.m_task = task;
    }

    public bool IsCompleted
       {
         get
         {
           return this.m_task.IsCompleted;
         }
       }
```

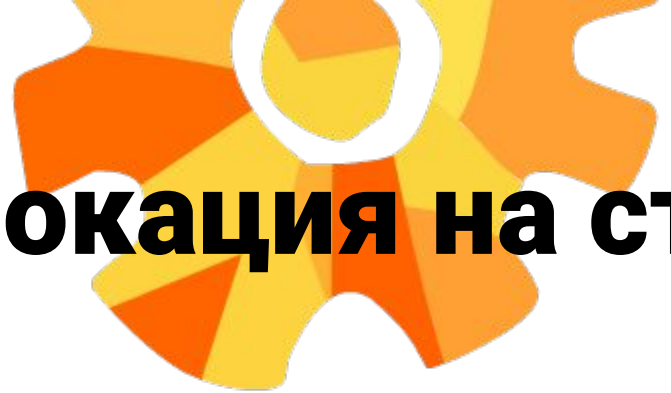
async/await

```
private readonly Dictionary<string, Task<string>> cache = new Dictionary<string, Task<string>>();

public Task<string> GetValueAsync(string key)
{
  Task<string> result;
  if (cache.TryGetValue(key, out result))
    return result;
  return DoGetValueAsync(key);
}

private async Task<string> DoGetValueAsync(string key)
{
  HttpClient client = new HttpClient();
  var response =  await client.GetAsync(key);
  var text = response.Content.ReadAsStringAsync();
  this.cache[key] = text;
  return await text;
}
```

async/await

.Net 4.6

`Task.FromResult(0)` → `Task.CompletedResult`



Аллокация на стеке

.net core 2.1

Span<T>

```
C# 7.2
int* block = stackalloc int[100];

Span<byte> span = size <= 128 ? stackalloc byte[size] : new byte[size];
```

ValueTask<TResult>

JIT оптимизация

Типичные заблуждения:

[AggressiveInlining] спасет всех

Generic типы инлайнятся

Компилятор умный

```
interface IChildItems
{
  Collection<IDisposable> ChildItems { get; }
}

internal static class FreeObjectUtils
{
  public void Free<T>(T instance)
    where T : IChildItems
  {
    bool needDispose = typeof(IDisposable).IsAssignableFrom(typeof(T));

    foreach (var element in instance.ChildItems)
      element.Dispose();

    if (needDispose)
      DisposableUtils.Clean(instance);
    else
      instance.ChildItems.Clear();
  }
}
```

JIT оптимизация

Типичные заблуждения:

[AggressiveInlining] спасет всех

Generic типы инлайнятся

Компилятор умный

```
interface IChildItems
{
  Collection<IDisposable> ChildItems { get; }
}

internal static class FreeObjectUtils
{
  public void Free<T>(T instance)
    where T : IChildItems
  {


    foreach (var element in instance.ChildItems)
      element.Dispose();

    if (false)
      DisposableUtils.Clean(instance);
    else
      instance.ChildItems.Clear();
  }
}
```

JIT оптимизация

Типичные заблуждения:

[AggressiveInlining] спасет всех

Generic типы инлайнятся

Компилятор умный

```
interface IChildItems
{
  Collection<IDisposable> ChildItems { get; }
}

internal static class FreeObjectUtils
{
  public void Free<T>(T instance)
    where T : IChildItems
  {


    foreach (var element in instance.ChildItems)
      element.Dispose();




    instance.ChildItems.Clear();
  }
}
```

JIT оптимизация

```
public Dictionary(IEqualityComparer<TKey> comparer);
```

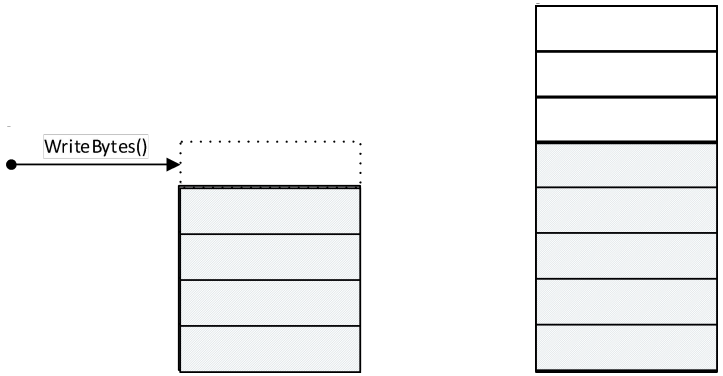
```
public struct LongEqualityComparer : IEqualityComparer<long>
{
        public static readonly IEqualityComparer<long> BoxedInstanceInt64 = new LongEqualityComparer();

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool Equals(long x, long y)
        {
            return x == y;
        }
}
```

JIT оптимизация

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public void Clear()
{
    if (typeof(T) == typeof(int) || typeof(T) == typeof(uint) || typeof(T) == typeof(byte) ||
        typeof(T) == typeof(short) || typeof(T) == typeof(long) || typeof(T) == typeof(ulong))
    {
        _size = 0;
        _version++;
    }
    else
    {
        int size = (int)_size;

        _size = 0;
        _version++;
        if (size > 0)
            Array.Clear(_items, 0, size);
    }
}
```

MemoryStream

MemoryStream

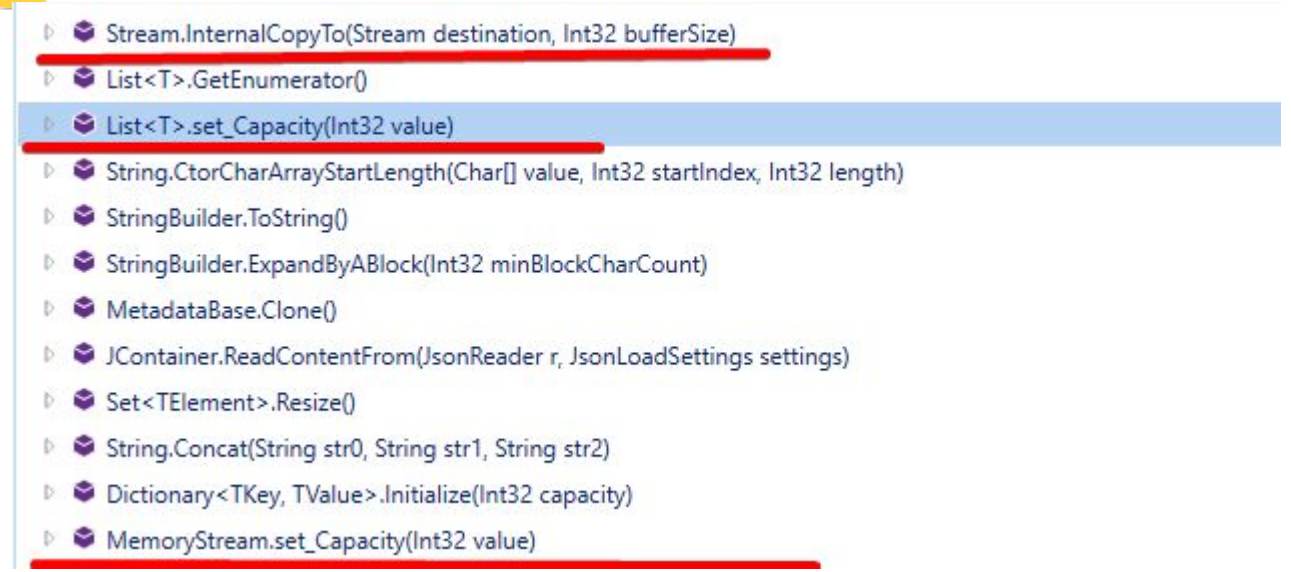


MemoryStream()	Initializes a new instance of the MemoryStream class with an expandable capacity initialized to zero.
MemoryStream(Byte[])	Initializes a new non-resizable instance of the MemoryStream class based on the specified byte array.
MemoryStream(Int32)	Initializes a new instance of the MemoryStream class with an expandable capacity initialized as specified.

MemoryStream

То же самое для:

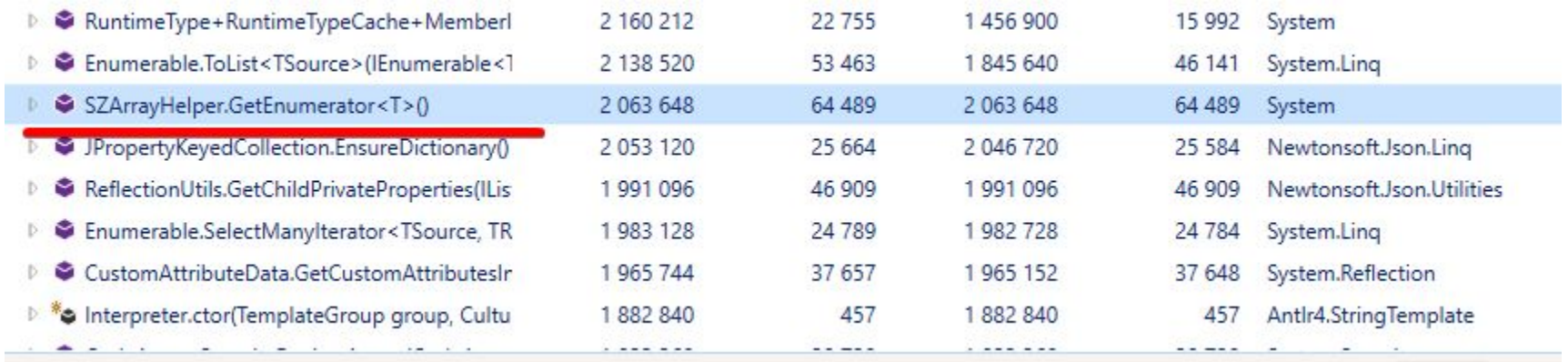
List<T>
Collection<T>
Dictionary<TKey, TValue>
и прочих коллекций



List<T>()	Initializes a new instance of the List<T> class that is empty and has the default initial capacity.
List<T>(IEnumerable<T>)	Initializes a new instance of the List<T> class that contains elements copied from the specified collection and has sufficient capacity to accommodate the number of elements copied.
List<T>(Int32)	Initializes a new instance of the List<T> class that is empty and has the specified initial capacity.

struct[] -> IList

```
int[] a = new int[10];
DoSomething(a);


public void DoSomething<T>(IList<T> list)
{
}
```



RuntimeType+RuntimeTypeCache+MemberI	2 160 212	22 755	1 456 900	15 992	System
Enumerable.ToList<TSource>(IEnumerable<T	2 138 520	53 463	1 845 640	46 141	System.Linq
SZArrayHelper.GetEnumerator<T>()	2 063 648	64 489	2 063 648	64 489	System
JPropertyKeyedCollection.EnsureDictionary()	2 053 120	25 664	2 046 720	25 584	Newtonsoft.Json.Linq
ReflectionUtils.GetChildPrivateProperties(ILis	1 991 096	46 909	1 991 096	46 909	Newtonsoft.Json.Utilities
Enumerable.SelectManyIterator<TSource, TR	1 983 128	24 789	1 982 728	24 784	System.Linq
CustomAttributeData.GetCustomAttributesIr	1 965 744	37 657	1 965 152	37 648	System.Reflection
Interpreter.ctor(TemplateGroup group, Cultu	1 882 840	457	1 882 840	457	Antlr4.StringTemplate


```
public IEnumerable<RibbonPartMetadata> FindChildren(Guid parentId)
{
      return this.AllParts.Where(part => part.ParentGuid == parentId).ToList();
}
```

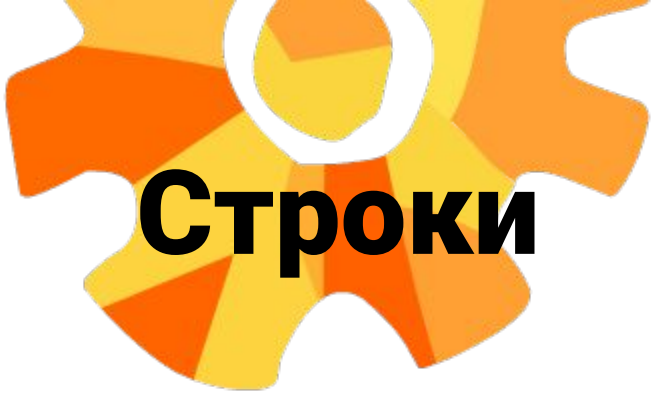
```
// [18 7 - 18 80]
    IL_000e: ldarg.0      // this
    IL_000f: call         instance class [mscorlib]System.Collections.Generic.IEnumerable`1<class WpfApp3.RibbonPartMetadata>
WpfApp3.Class2::get_AllParts()
    IL_0014: ldloc.0      // 'CS$<>8__locals0'
    IL_0015: ldftn        instance bool WpfApp3.Class2/'<>c__DisplayClass2_0'::'<FindChilds>b__0'(class WpfApp3.RibbonPartMetadata)
    IL_001b: newobj       instance void class [mscorlib]System.Func`2<class WpfApp3.RibbonPartMetadata, bool>::.ctor(object, native int)
    IL_0020: call         class [mscorlib]System.Collections.Generic.IEnumerable`1<!!0/*class WpfApp3.RibbonPartMetadata*/>
[System.Core]System.Linq.Enumerable::Where<class WpfApp3.RibbonPartMetadata>(class
[mscorlib]System.Collections.Generic.IEnumerable`1<!!0/*class WpfApp3.RibbonPartMetadata*/>, class [mscorlib]System.Func`2<!!0/*class
WpfApp3.RibbonPartMetadata*/, bool>)
    IL_0025: call         class [mscorlib]System.Collections.Generic.List`1<!!0/*class WpfApp3.RibbonPartMetadata*/>
[System.Core]System.Linq.Enumerable::ToList<class WpfApp3.RibbonPartMetadata>(class
[mscorlib]System.Collections.Generic.IEnumerable`1<!!0/*class WpfApp3.RibbonPartMetadata*/>)
    IL_002a: stloc.1      // V_1
    IL_002b: br.s         IL_002d
```


```
public IEnumerable<RibbonPartMetadata> FindChildren(Guid parentId)
{
      return this.AllParts.Where(part => part.ParentGuid == parentId).ToList();
}
```

```
public IEnumerable<RibbonPartMetadata> FindChildren(Guid parentId)
{
      var list = new List<RibbonPartMetadata>();

      foreach (RibbonPartMetadata part in this.AllParts)
      {
        if (part.ParentGuid == parentId)
        {
          list.Add(part);
        }
      }

      return list;
}
```


```
public IEnumerable<RibbonPartMetadata> FindChildren(Guid parentId)
{
      return this.AllParts.Where(part => part.ParentGuid == parentId).ToList();
}
```

```
public IEnumerable<RibbonPartMetadata> FindChildren(Guid parentId)
{
      var list = new List<RibbonPartMetadata>(this.AllPart.Count);

      foreach (RibbonPartMetadata part in this.AllParts)
      {
        if (part.ParentGuid == parentId)
        {
          list.Add(part);
        }
      }

      return list;
}
```

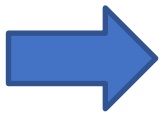


Строки

Вместо конкатенаций используй StringBuilder

Вместо string.Format используй конкатенацию

```
string first = "first";
string second = "second";
return $"{first}{second}";
```



```
string first = "first";
string second = "second";
return first + second;
```

Boxing

```
static Logger log = new Logger();
public void Info(params object[] args)

log.Info(1, 2, 3, 4, 5);
log.Info();
```

```
new object[0]
```

Как надо?

```
public static void Info();
public static void Info<T1>(T1 arg1);
…
public static void Info<T1, T2, T3>(T1 arg1, T2 arg2, T3 arg3);
```

Heap Allocation Viewer

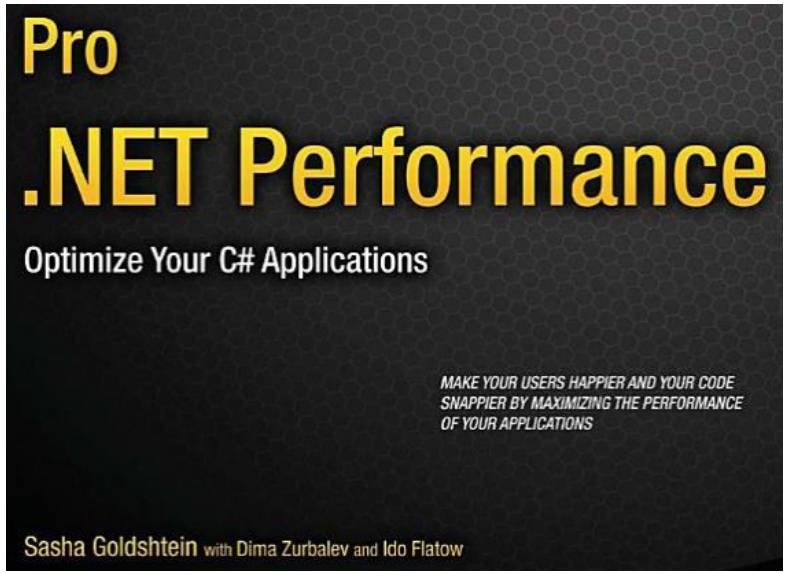
```
0 references
public class ParamsArgs
{
    1 reference
    public static void Method1(params string[] args)...

    0 references
    public void Method2()
    {
        Method1("Hello", "World");
    }
}
```

Object allocation: parameters array 'args' creation

```
var strings = new string[] { "x", "y"};
foreach (var s in strings)
{
    Task.Run(() => Console.WriteLine(s));
}
```

Delegate allocation: capture of 's' variable

Почитать?



Federico Lois
Sasha Goldstein
Андрей Акиньшин

Сафин Рустам

Разработчик
DIRECTUM в г. Уфа

safin_rf@directum.ru