

Десятая независимая научно-практическая  
конференция «Разработка ПО 2014»

23 - 25 октября, Москва



# ReSharper inside out

Кирилл Скрыган

JetBrains

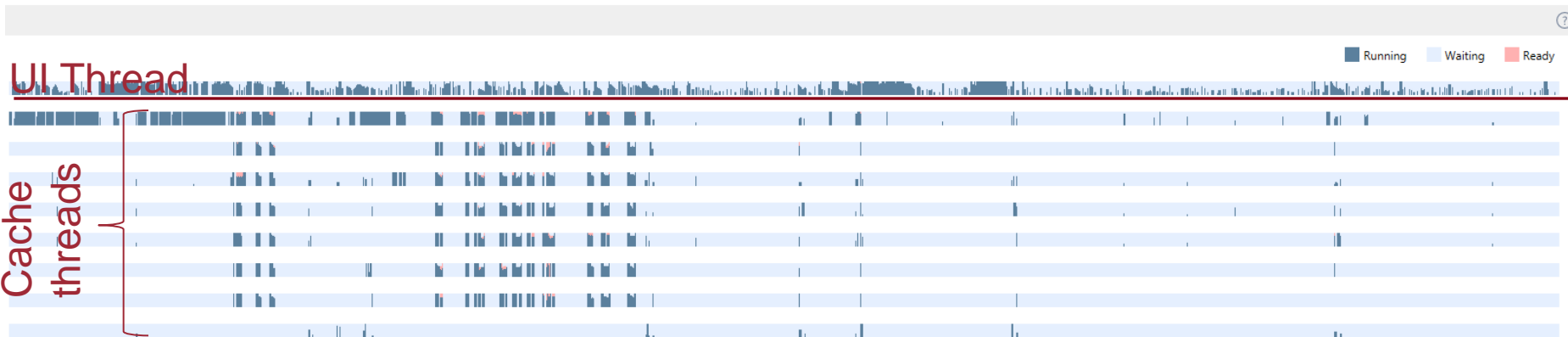
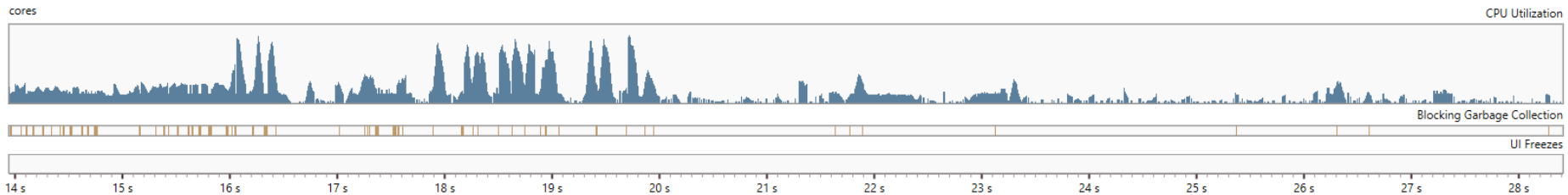
# Две большие технологические проблемы

- Производительность во всех проявлениях при мощной функциональности
  - Плавность UI
  - Memory Traffic
  - Общая память приложения
- Расширяемость
  - По языкам
  - По фичам
  - По плагинам
  - И теперь даже по продуктам...

# Performance: специфика IDE

- Чтобы быстро предоставлять пользователю фичи нужна большая, развесистая и, увы, сложная модель данных, кэш
- Кэш обновляется почти каждый раз при изменении кода и не только
- Обновление кэша – сложная операция, прямо или косвенно мешающая UI потоку

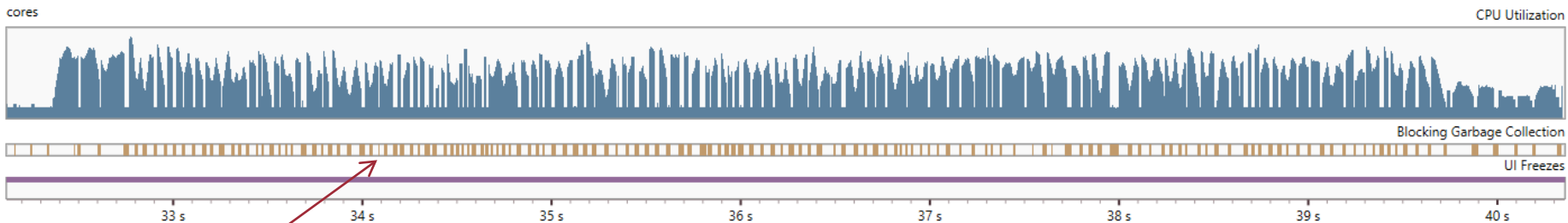
# Performance: UI smoothness



# Performance: проблемы

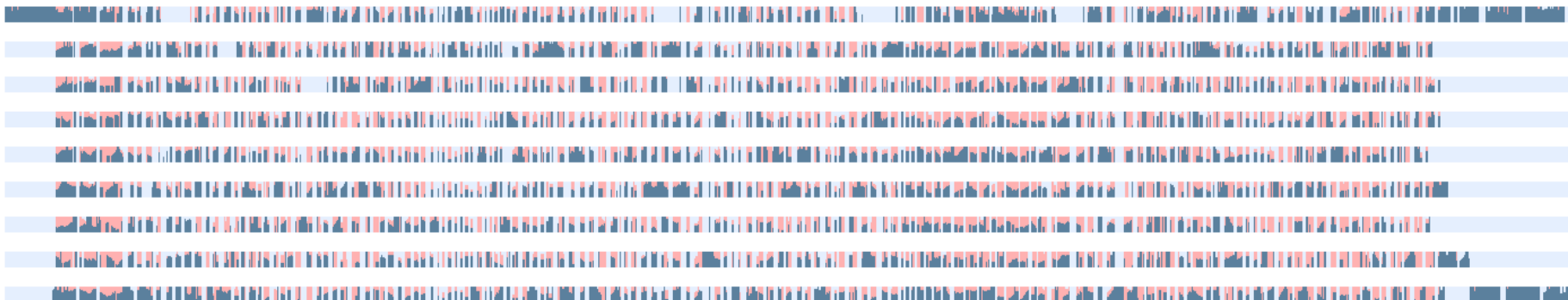
- UI thread должен быть почти всё время свободным
- Размеры кэша ограничены, так как
  - 32-битный процесс Visual Studio, всего ~3 Гб на процесс
  - Увеличивается нагрузка на GC
- Обновление кэша может аллоцировать много памяти
- Большое количество одновременно работающих потоков усугубляют «Кризис среднего возраста» объектов

# Performance: GC impact



Blocking GC

Running Waiting Ready



# Примеры: замыкание

```
namespace SECR_DEMO
{
    public class Classs1
    {
        private void Foo(Func<String, String> id)
        {
        }

        private void Test()
        {
            string localVariable = "sample";
            this.Foo((Func<string, string>)(s => localVariable + "+1"));
        }
    }
}
```

# Примеры: ближе к реальности

```
private void Test()  
{  
    Class1.<>c__DisplayClass1 cDisplayClass1 = new Class1.<>c__DisplayClass1();  
    cDisplayClass1.s = "sample";  
    // ISSUE: method pointer  
    this.Foo(new Func<string, string>((object) cDisplayClass1, __methodptr(<Test>b__0)));  
}
```

```
[CompilerGenerated]  
private sealed class <>c__DisplayClass1  
{  
    public string s;  
  
    public <>c__DisplayClass1()  
    {  
        base..ctor();  
    }  
  
    public string <Test>b__0(string s1)  
    {  
        return this.s + "+1";  
    }  
}
```



# Примеры: две лямбды...

```
public void Execute(IDataContext context, DelegateExecute nextExecute)
{
    var s1 = "toBeCaptured1";
    var s2 = "toBeCaptured2";
    myLifetime.AddBracket(() =>
    {
        var v = s1 + "smth";
    },
    () =>
    {
        var v = s2 + "smthelse";
    });
}
```

# Примеры: ...и один замыкающий класс!

```
public void Execute(IDataContext context, DelegateExecute nextExecute)
{
    AboutAction.<>c__DisplayClass2 cDisplayClass2 = new AboutAction.<>c__DisplayClass2();
    cDisplayClass2.s1 = "toBeCaptured1";
    cDisplayClass2.s2 = "toBeCaptured2";
    // ISSUE: method pointer
    // ISSUE: method pointer
    this.myLifetime.AddBracket(new Action((object) cDisplayClass2, __methodptr(<Execute>b_
}
```

# Примеры: yield

```
namespace ConsoleApplication3
{
    public class YieldExample
    {
        private readonly IEnumerable<string> myEnumerable = new List<string>();

        IEnumerable<string> Enumerate()
        {
            foreach (var item in myEnumerable)
            {
                yield return item;
            }
        }
    }
}
```

# Примеры: yield в реальности

```
private IEnumerable<string> Enumerate()  
{  
    Yield.<Enumerate>d__0 enumerateD0 = new Yield.<Enumerate>d__0(-2);  
    enumerateD0.<>4__this = this;  
    return (IEnumerable<string>) enumerateD0;  
}
```

[CompilerGenerated]

```
private sealed class <Enumerate>d__0 : IEnumerable<string>, IEnumerable  
{  
    private string <>2__current;  
    private int <>1__state;  
    private int <>1__initialThreadId;  
    public Yield <>4__this;  
    public string <item>5__1;  
    public IEnumerator<string> <>7__wrap2;
```

# Примеры: IList vs List

```
public class IListTest
{
    private List<string> myValues = new List<string>();

    public IList<string> GetValues()
    {
        return myValues;
    }

    void Main(IListTest test)
    {
        foreach (var value in test.GetValues())
        {

        }
    }
}
```

# Примеры: IList - неожиданный boxing

- List

```
IL_0002: ldarg.1
IL_0003: callvirt instance class [mscorlib]System.Collections.Generic.List`1<string> CodeFestDe
IL_0008: callvirt instance valuetype [mscorlib]System.Collections.Generic.List`1/Enumerator<!0>
IL_000d: stloc.1
.try
```

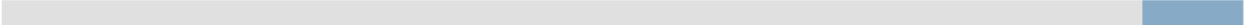





- IList

```
IL_0002: ldarg.1
IL_0003: callvirt instance class [mscorlib]System.Collections.Generic.IList`1<string> Cod
IL_0008: callvirt instance class [mscorlib]System.Collections.Generic.IEnumerator`1<!0> c
IL_000d: stloc.1
.try
```

# Мораль

- Обязательно учитывать memory traffic
- Знать подводные камни и избегать синтаксического сахара в высокопроизводительном коде
- Профилировать:
  - dotMemory
  - TimeLine tools
  - dotTrace

# Memory: проблема с LOH-ом

Map	Total bytes ▾	Utilization	Objects count	Fragmentation
Large Object Heap				
	16,476,920	8 %	3	0 %
	16,387,944	67 %	7	32 %
	15,805,472	1 %	1	0 %
	15,444,632	76 %	31	32 %
	14,929,368	59 %	19	13 %
	11,977,640	46 %	26	37 %



# Memory: уменьшаем размер кэшей

- Интернирование
- Свои «велосипеды» - структуры данных, экономящие память
- **Выносим кэши на диск**
- Теоретически можно вынести вообще из процесса, но пока только в теории

# Memory: leveldb

- Of Google, open source
- In-process, not sharable, thread safe
- Benchmarks:
  - Read: 250, 150, 7 MB/sec
  - Write: 62, 45, 47 MB/sec

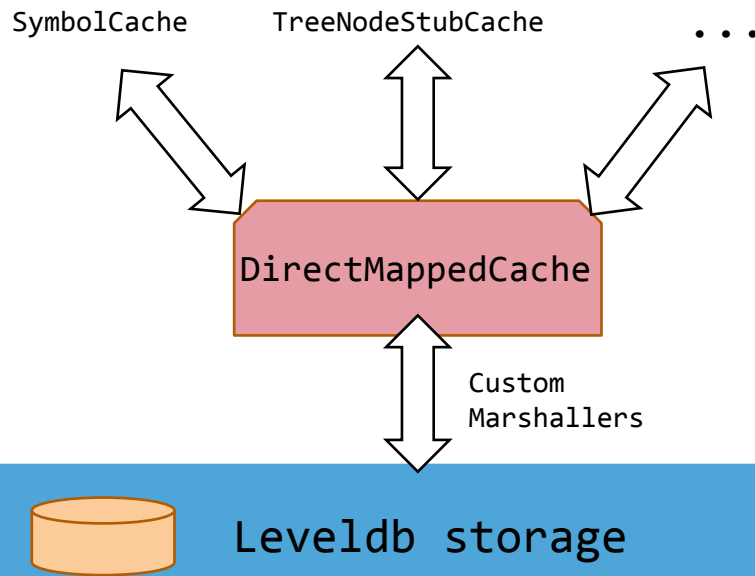
# Memory: маршалеры

- Stackalloc в .NET тормозит, так как принудительно обнуляет всю выделяемую память
- Собственные маршалеры
  - Однопоточные
  - Aggressive inlining
  - Kernel32Dll.HeapReAlloc
  - Минимум в 2,5 раза быстрее, чем BinaryWriter/Reader

# Memory: опять memory traffic

- Чем больше маршалим – тем больше аллоцируется памяти
- Выход: «процессорные кэши» в оперативной памяти поверх базы данных
  - DirectMappedCache
  - Более сложные ассоциативные кэши
  - Многоурвневые кэши (IDEA)

# Memory: общая картина



# Полезные .NET инструменты

- dotTrace – алгоритмы
- dotTimeline – блокировки потоков, starvation, GC фильтры
- dotMemory – memory traffic, утечки, общее исследование managed памяти
- dotPeek – декомпилятор
- Ildasm – истина в последней инстанции
- ReSharper
  - Boxing
  - Memory allocations

# Multithreading: основная проблема

- Обновление кэша требует ReadLock и может быть долгим
- Однако UI-поток должен иметь возможность моментально брать WriteLock...

# Multithreading: решение

- Chunked operations: разбить длинный процесс обновления кэшэЙ на много мелких частей
- Прерываться как только кто-то просит WriteLock или по другим условиям (**checkForInterrupt**)
- Прерываемся → перекладываем недоделанное, с учётом новых изменений
- Timeline для выявления длинных и возможно ненужных ReadLock-ов



# Extensibility: проблема тысячи сборок

- Ось: некий однородный набор конфигураций продукта для его выпуска
- Например ось окружений
  - Visual Studio 2008, 2010, 2012, 2013, vs14, ...
  - Standalone
  - Server-side?
- Если единица deployment'a – это сборка, то сборок может быть декартово произведение осей

# Extensibility: наше решение

- Собственный Component Container (loc)
- Единица deployment'a – пространства имён, а точнее компоненты внутри них
- Декларативно помечаем пространства имён атрибутами, какой оси требуются компоненты внутри
- Component Container просто собирает те компоненты которые подходят под требования конфигурации

# Extensibility: zone example

```
[- namespace JetBrains.ReSharper.Intentions.JavaScript
{
  [Zone]
  public class ZoneMarker :
    IRequire<ILanguageJavaScriptZone>,
    IRequire<ICodeEditingZone>,
    IRequire<DaemonZone>
  {
  }
}
```

# Вопросы?

- Кирилл Скрыган
- [Kirill.skrygan@jetbrains.com](mailto:Kirill.skrygan@jetbrains.com)
- [twitter.com/kskrygan](https://twitter.com/kskrygan)