Туда и обратно.
Тёмная сторона сериализации

Константин Рудниченко
CTO, Cadwise


СТОРОНА ТЕМНАЯ
СЛАЙДЫ БЕЛЫЕ
imgflip.com

О себе

- Зарабатываю на жизнь программированием с 2005 года
- Программировал еще на .net framework 1.0
- Помню C# без Generics
  - Складывал объекты ArrayList
- Использовал BinaryFormatter ¯_(ツ)_/¯

“ЗОЧЕМ?
Есть же <*выберите свой сериализатор*>?”

Варианты

- Protocol Buffers
- Json
  - Newtonsoft.Json
  - System.Text.Json
  - etc
- BinaryFormatter
- XmlFormatter
- другие

Причины

Причины

- Экзотические форматы

Причины

- Экзотические форматы
- Лучшая производительность

Причины

- Экзотические форматы
- Лучшая производительность
- Другие сценарии использования Reflection:
  - Data-binding
  - Object-object mapping
  - Логирование: Destructing

О докладе

- Доступ к данным: Reflection и альтернативы
- Производительность различных способов доступа
- Участие доступа к данным в процессе сериализации

Прямой доступ vs. Reflection

System.Reflection Namespace

The System.Reflection namespace contains types that retrieve information about assemblies, modules, members, parameters, and other entities in managed code by examining their metadata. These types also can be used to manipulate instances of loaded types, for example to hook up events or to invoke methods. To dynamically create types, use the System.Reflection.Emit namespace.

System.Reflection

- System.Object
  - GetType()

System.Reflection

- System.Object
  - GetType()
- System.Type
  - GetProperties(...)

System.Reflection

- System.Object
  - GetType()
- System.Type
  - GetProperties(...)
- System.Reflection.PropertyInfo
  - Name
  - PropertyType
  - GetValue/SetValue

“Все знают, что
Reflection это **Медленно!!!**”

Type System Overview

Design Goals and Non-goals

Goals

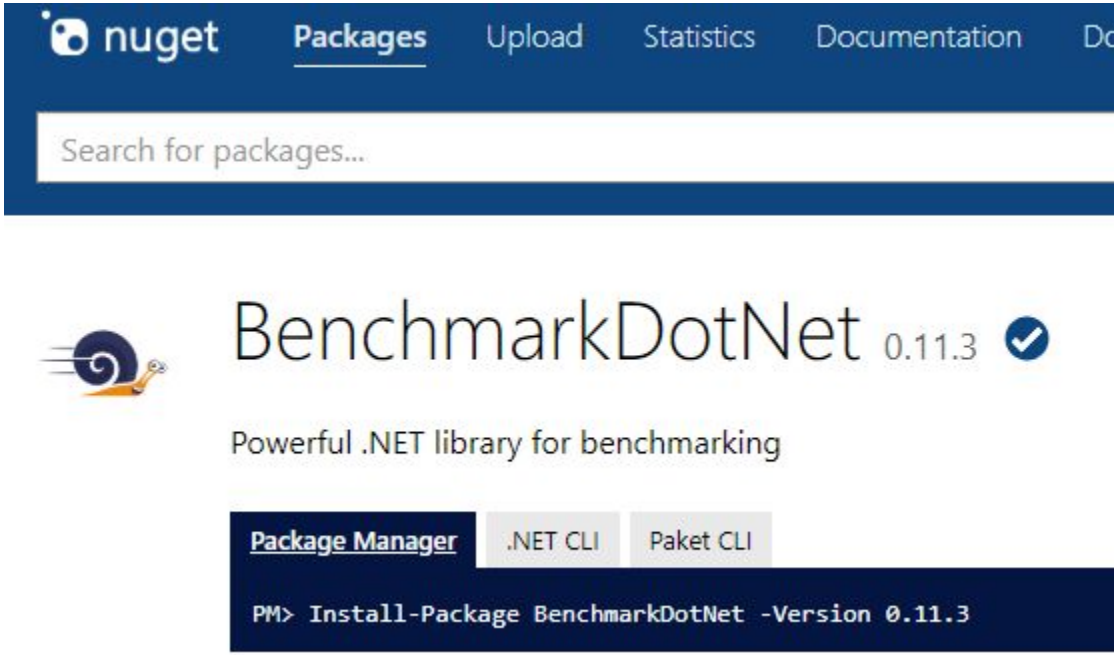
- Accessing information needed at runtime from executing (non-reflection) code is very fast.
- Accessing information needed at compilation time for generating code is straightforward.
- The garbage collector/stackwalker is able to access necessary information without taking locks, or allocating memory.
- Minimal amounts of types are loaded at a time.
- Minimal amounts of a given type are loaded at type load time.
- Type system data structures must be storable in NGEN images.

Non-Goals

- All information in the metadata is directly reflected in the CLR data structures.
- All uses of reflection are fast.

Насколько Reflection это медленно?

Цифры


nuget
Packages
Upload
Statistics
Documentation
Search for packages...
BenchmarkDotNet 0.11.3
Powerful .NET library for benchmarking
Package Manager
.NET CLI
Paket CLI
PM> Install-Package BenchmarkDotNet -Version 0.11.3

```
Test _test = new Test("Hello world!");
```

```
[BenchmarkCategory(GetCategory), Benchmark(Baseline = true)]
public string GetViaProperty()
    => _test.StringProperty;
```

```
[BenchmarkCategory(GetCategory), Benchmark(Baseline = true)]
public string GetViaProperty()
    => _test.StringProperty;

[BenchmarkCategory(SetCategory), Benchmark(Baseline = true)]
public void SetViaProperty()
    => _test.StringProperty = nameof(SetViaProperty);
```

```
Test _test = new Test("Hello world!");

Type TargetType = typeof(Test);
```

```
Test _test = new Test("Hello world!");

Type TargetType = typeof(Test);

PropertyName = nameof(Test.StringProperty);
```

```
Test _test = new Test("Hello world!");

Type TargetType = typeof(Test);

PropertyName = nameof(Test.StringProperty);

BindingFlags BindingFlags = System.Reflection.BindingFlags.Instance
                          | System.Reflection.BindingFlags.Public;
```

```
[BenchmarkCategory(GetCategory), Benchmark]
public string GetViaReflection()
{
    var property = TargetType.GetProperty(PropertyName, BindingFlags);
    return (string)property.GetValue(_test, null);
}
```

```
[BenchmarkCategory(GetCategory), Benchmark]
public string GetViaReflection()
{
    var property = TargetType.GetProperty(PropertyName, BindingFlags);
    return (string)property.GetValue(_test, null);
}

[BenchmarkCategory(SetCategory), Benchmark]
public void SetViaReflection()
{
    var property = TargetType.GetProperty(PropertyName, BindingFlags);
    property.SetValue(_test, nameof(SetViaReflection));
}
```

Ba Dum Tss!

Method	Mean	Error	StdDev	Ratio
SetViaProperty	1.8766 ns	0.0818 ns	0.0909 ns	1.00
SetViaReflection	196.0413 ns	1.6216 ns	1.3541 ns	105.35

Ba Dum Tss!

Method	Mean	Error	StdDev	Ratio
GetViaProperty	0.3240 ns	0.0635 ns	0.0563 ns	1.00
GetViaReflection	131.9475 ns	1.0697 ns	0.9483 ns	418.20

“Давай закэшируем PropertyInfo!”

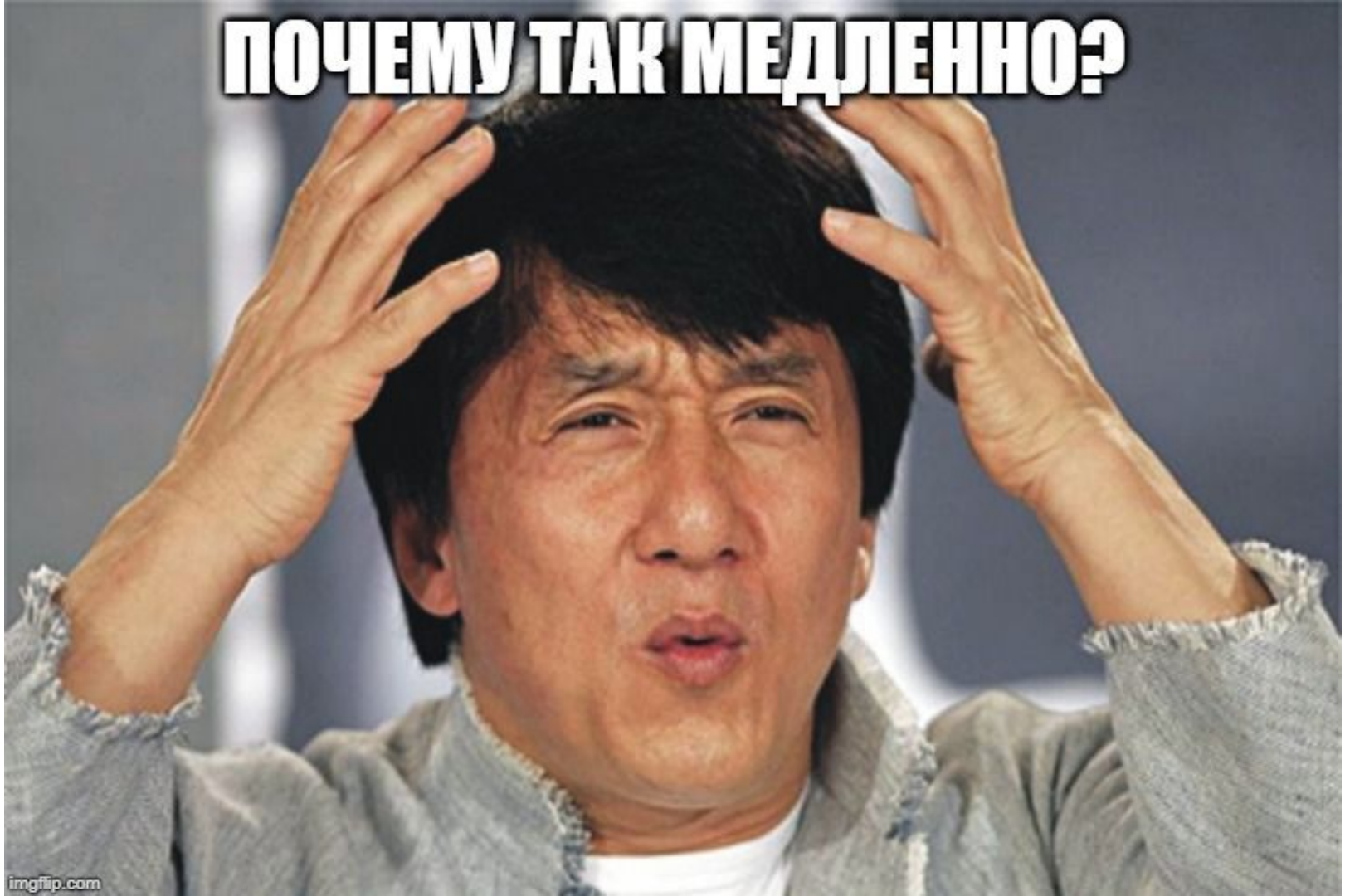
```
PropertyInfo CachedPropertyInfo
= TargetType.GetProperty(PropertyName, BindingFlags);

[BenchmarkCategory(GetCategory), Benchmark]
public string GetViaReflectionCached()
    => (string)CachedPropertyInfo.GetValue(_test, null);

[BenchmarkCategory(SetCategory), Benchmark]
public void SetViaReflectionCached()
    => CachedPropertyInfo.SetValue(_test, nameof(SetViaReflectionCached));
```

Cached PropertyInfo

Method	Mean	Error	StdDev	Ratio
SetViaProperty	1.8766 ns	0.0818 ns	0.0909 ns	1.00
SetViaReflectionCached	143.1944 ns	0.6018 ns	0.5629 ns	76.44
SetViaReflection	196.0413 ns	1.6216 ns	1.3541 ns	105.35

Cached PropertyInfo

Method	Mean	Error	StdDev	Ratio
GetViaProperty	0.3240 ns	0.0635 ns	0.0563 ns	1.00
GetViaReflectionCached	80.6136 ns	0.5869 ns	0.5490 ns	255.63
GetViaReflection	131.9475 ns	1.0697 ns	0.9483 ns	418.20


ПОЧЕМУ ТАК МЕДЛЕННО?
imgflip.com

“Неужели нет другого способа?!”

FastMember 1.5.0

In .NET reflection is slow... well, kinda slow. If you need access to the members of an arbitrary type, with the type and member-names known only at runtime - then it is frankly hard (especially for DLR types). This library makes such access easy and fast.

Package Manager | .NET CLI | PackageReference | Paket CLI

```
PM> Install-Package FastMember -Version 1.5.0
```

FastMember

```
TypeAccessor _accessor
    = TypeAccessor.Create(TargetType, allowNonPublicAccessors: true);
```

FastMember

```
TypeAccessor _accessor
    = TypeAccessor.Create(TargetType, allowNonPublicAccessors: true);

[BenchmarkCategory(GetCategory), Benchmark]
public string GetViaFastMember()
     => (string)_accessor[_test, PropertyName];
```

FastMember

```
TypeAccessor _accessor
    = TypeAccessor.Create(TargetType, allowNonPublicAccessors: true);

[BenchmarkCategory(GetCategory), Benchmark]
public string GetViaFastMember()
    => (string)_accessor[_test, PropertyName];

[BenchmarkCategory(SetCategory), Benchmark]
public void SetViaFastMember()
    => _accessor[_test, PropertyName] = nameof(SetViaFastMember);
```

Method	Mean	Error	StdDev	Ratio
SetViaProperty	1.8766 ns	0.0818 ns	0.0909 ns	1.00
SetViaFastMember	25.9617 ns	0.2093 ns	0.1958 ns	13.86
SetViaReflectionCached	143.1944 ns	0.6018 ns	0.5629 ns	76.44
SetViaReflection	196.0413 ns	1.6216 ns	1.3541 ns	105.35

Method	Mean	Error	StdDev	Ratio
GetViaProperty	0.3240 ns	0.0635 ns	0.0563 ns	1.00
GetViaFastMember	24.0186 ns	0.1106 ns	0.1035 ns	76.05
GetViaReflectionCached	80.6136 ns	0.5869 ns	0.5490 ns	255.63
GetViaReflection	131.9475 ns	1.0697 ns	0.9483 ns	418.20

“Безнадёга?”

Delegate.CreateDelegate

Delegate

Открытые / Закрытые

Delegate: открытые/закрытые

```
var foo = new Foo();

Action closed = foo.InstanceMethod;
```

Delegate: открытые/закрытые

```
var foo = new Foo();

Action closed = foo.InstanceMethod;
Action opened = Foo.StaticMethod;
```

Delegate: открытые/закрытые

```
var foo = new Foo();

Action closed = foo.InstanceMethod; // closed.Target == foo
Action opened = Foo.StaticMethod;   // opened.Target == null
```

Delegate.CreateDelegate

CreateDelegate(Type, MethodInfo)

Creates a delegate of the specified type to represent the specified static method.

C# Copy

```
public static Delegate CreateDelegate (Type type, System.Reflection.MethodInfo method);
```

Delegate.CreateDelegate

CreateDelegate(Type, MethodInfo)

Creates a delegate of the specified type to represent the specified static method.

C# Copy

```
public static Delegate CreateDelegate (Type type, System.Reflection.MethodInfo method);
```

PropertyInfo → MethodInfo

PropertyInfo → MethodInfo

GetGetMethod() / GetSetMethod()

Delegate.CreateDelegate

```
MethodInfo PropertyGetMethod = CachedPropertyInfo.GetGetMethod();
```

Delegate.CreateDelegate

```
MethodInfo PropertyGetMethod = CachedPropertyInfo.GetGetMethod();

Delegate.CreateDelegate(typeof(Func<Test, string>), PropertyGetMethod);
```

Delegate.CreateDelegate

```
MethodInfo PropertyGetMethod = CachedPropertyInfo.GetGetMethod();

Delegate.CreateDelegate(typeof(Func<Test, string>), PropertyGetMethod);
```

Delegate.CreateDelegate

```
MethodInfo PropertySetMethod = CachedPropertyInfo.GetSetMethod();
Delegate.CreateDelegate(typeof(Action<Test, string>), PropertySetMethod);
```

Delegate.CreateDelegate

```
MethodInfo PropertySetMethod = CachedPropertyInfo.GetSetMethod();
Delegate.CreateDelegate(typeof(Action<Test, string>), PropertySetMethod);
```

Delegate.CreateDelegate

```
[BenchmarkCategory(GetCategory), Benchmark]
public void GetViaDelegate() => _getDelegate(_test);

[BenchmarkCategory(SetCategory), Benchmark]
public void SetViaDelegate() => _setDelegate(_test, nameof(SetViaDelegate));
```

Method	Mean	Error	StdDev	Ratio
SetViaProperty	1.8766 ns	0.0818 ns	0.0909 ns	1.00
SetViaDelegate	3.6128 ns	0.0387 ns	0.0343 ns	1.93
SetViaFastMember	25.9617 ns	0.2093 ns	0.1958 ns	13.86
SetViaReflectionCached	143.1944 ns	0.6018 ns	0.5629 ns	76.44
SetViaReflection	196.0413 ns	1.6216 ns	1.3541 ns	105.35

Method	Mean	Error	StdDev	Ratio
GetViaProperty	0.3240 ns	0.0635 ns	0.0563 ns	1.00
GetViaDelegate	2.7044 ns	0.0256 ns	0.0240 ns	8.57
GetViaFastMember	24.0186 ns	0.1106 ns	0.1035 ns	76.05
GetViaReflectionCached	80.6136 ns	0.5869 ns	0.5490 ns	255.63
GetViaReflection	131.9475 ns	1.0697 ns	0.9483 ns	418.20

“А может генерировать код в runtime?”

Генерация кода в runtime

Генерация кода в runtime

- IL Emit

Генерация кода в runtime

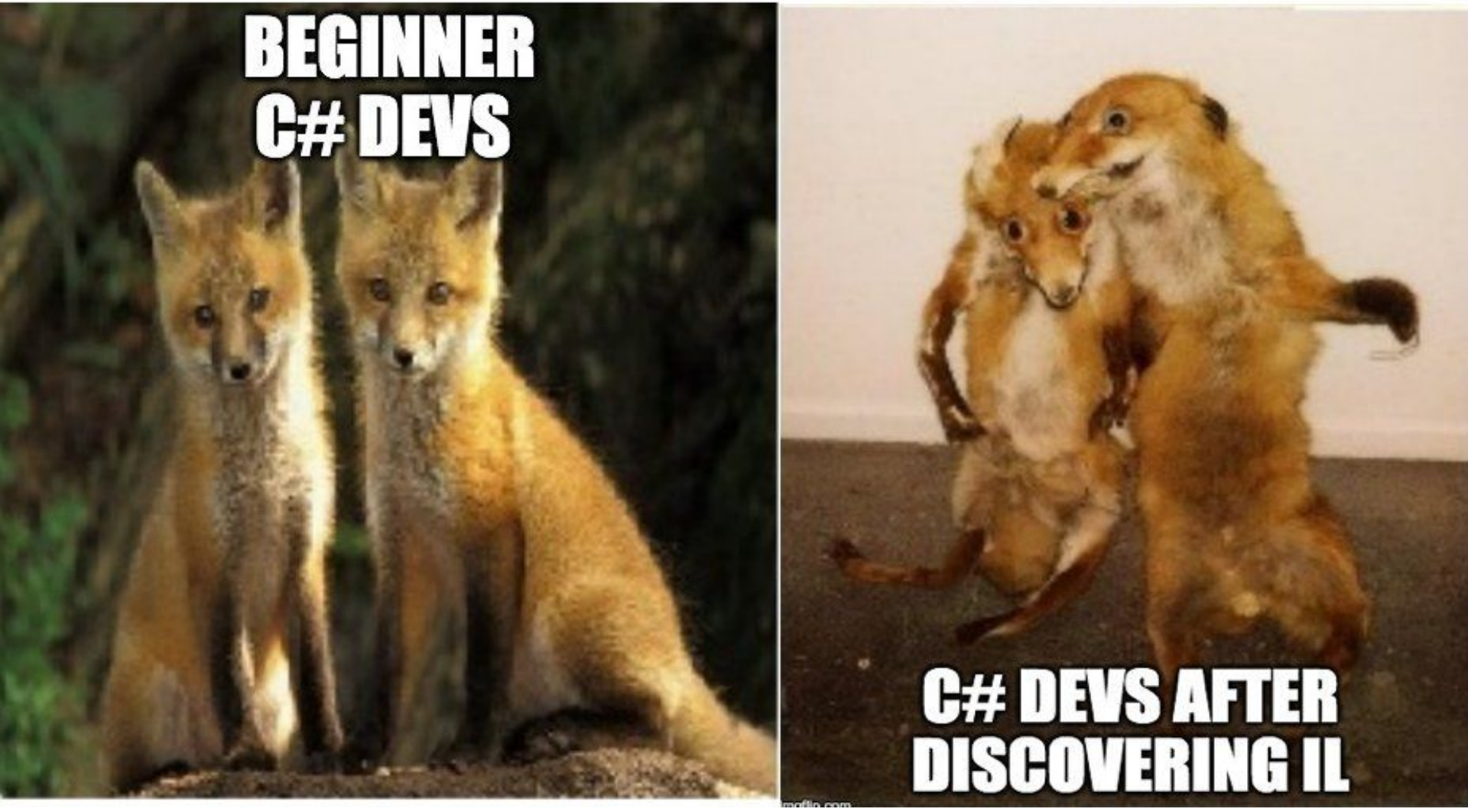
- IL Emit
- Compiled Expression Trees

IL Emit

Intermediate Language

Common Intermediate Language (**CIL**), formerly called **Microsoft Intermediate Language** (**MSIL**) or **Intermediate Language** (**IL**) [1]., is the intermediate language binary instruction set defined within the Common Language Infrastructure (CLI) specification [2]. CIL instructions are executed by a CLI-compatible runtime environment such as the Common Language Runtime. Languages which target the CLI compile to CIL. CIL is object-oriented, stack-based bytecode. Runtimes typically just-in-time compile CIL instructions into native code.

CIL was originally known as Microsoft Intermediate Language (MSIL) during the beta releases of the .NET languages. Due to standardization of C# and the Common Language Infrastructure, the bytecode is now officially known as CIL.[3] Windows Defender virus definitions continue to refer to binaries compiled with it as MSIL.[4]


BEGINNER
C# DEVS
C# DEVS AFTER
DISCOVERING IL

```
public string GetViaProperty() => _test.StringProperty;
```

```
public string GetViaProperty() => _test.StringProperty;

.method private hidebysig
static string GetViaProperty( class Test target)
cil managed
{
  .maxstack 1
  ldarg.0
  callvirt instance string Test::get_StringProperty()
  ret
}
```

```
var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
        new[] { typeof(TTarget) },
        Module, true);
```

```
var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
        new[] { typeof(TTarget) },
        Module, true);
```

```
var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
        new[] { typeof(TTarget) },
        Module, true);
```

```
var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
        new[] { typeof(TTarget) },
        Module, true);
```

```
var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
        new[] { typeof(TTarget) },
        Module, true);

var gen = method.GetILGenerator();
```

```
var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
        new[] { typeof(TTarget) },
        Module, true);

var gen = method.GetILGenerator();

var getMethod = property.GetGetMethod();
```

```
var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
        new[] { typeof(TTarget) },
        Module, true);

var gen = method.GetILGenerator();

var getMethod = property.GetGetMethod();

gen.Emit(OpCodes.Ldarg_0);
```

```
var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
        new[] { typeof(TTarget) },
        Module, true);

var gen = method.GetILGenerator();

var getMethod = property.GetGetMethod();

gen.Emit(OpCodes.Ldarg_0);
gen.Emit(OpCodes.Call, getMethod);
```

```
var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
        new[] { typeof(TTarget) },
        Module, true);

var gen = method.GetILGenerator();

var getMethod = property.GetGetMethod();

gen.Emit(OpCodes.Ldarg_0);
gen.Emit(OpCodes.Call, getMethod);
gen.Emit(OpCodes.Ret);
```

```
var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
        new[] { typeof(TTarget) },
        Module, true);

var gen = method.GetILGenerator();

var getMethod = property.GetGetMethod();

gen.Emit(OpCodes.Ldarg_0);
gen.Emit(OpCodes.Call, getMethod);
gen.Emit(OpCodes.Ret);

                                        method.CreateDelegate(typeof(Func<TTarget, TParam>));
```

```
public static Func<TTarget, TParam> GenerateGetter<TTarget, TParam>(PropertyInfo property)
{
	var method = new DynamicMethod(property.Name + "GetterTyped", typeof(TParam),
		new[] { typeof(TTarget) },
		Module, true);

	var gen = method.GetILGenerator();

	var getMethod = property.GetGetMethod();

	gen.Emit(OpCodes.Ldarg_0);
	gen.Emit(OpCodes.Call, getMethod);
	gen.Emit(OpCodes.Ret);

	return (Func<TTarget, TParam>)method.CreateDelegate(typeof(Func<TTarget, TParam>));
}
```

```
var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
        new[] { typeof(TTarget), typeof(TParam) }, Module, true);
```

```
var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
        new[] { typeof(TTarget), typeof(TParam) }, Module, true);
```

```
var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
        new[] { typeof(TTarget), typeof(TParam) }, Module, true);
```

```
var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
        new[] { typeof(TTarget), typeof(TParam) }, Module, true);

var gen = method.GetILGenerator();
```

```
var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
        new[] { typeof(TTarget), typeof(TParam) }, Module, true);

var gen = method.GetILGenerator();

var setMethod = propertyInfo.GetSetMethod();
```

```
var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
        new[] { typeof(TTarget), typeof(TParam) }, Module, true);

var gen = method.GetILGenerator();

var setMethod = propertyInfo.GetSetMethod();

gen.Emit(OpCodes.Ldarg_0);
```

```
var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
        new[] { typeof(TTarget), typeof(TParam) }, Module, true);

var gen = method.GetILGenerator();

var setMethod = propertyInfo.GetSetMethod();

gen.Emit(OpCodes.Ldarg_0);
gen.Emit(OpCodes.Ldarg_1);
```

```
var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
        new[] { typeof(TTarget), typeof(TParam) }, Module, true);

var gen = method.GetILGenerator();

var setMethod = propertyInfo.GetSetMethod();

gen.Emit(OpCodes.Ldarg_0);
gen.Emit(OpCodes.Ldarg_1);
gen.Emit(OpCodes.Call, setMethod);
```

```
var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
        new[] { typeof(TTarget), typeof(TParam) }, Module, true);

var gen = method.GetILGenerator();

var setMethod = propertyInfo.GetSetMethod();

gen.Emit(OpCodes.Ldarg_0);
gen.Emit(OpCodes.Ldarg_1);
gen.Emit(OpCodes.Call, setMethod);
gen.Emit(OpCodes.Ret);
```

```
var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
        new[] { typeof(TTarget), typeof(TParam) }, Module, true);

var gen = method.GetILGenerator();

var setMethod = propertyInfo.GetSetMethod();

gen.Emit(OpCodes.Ldarg_0);
gen.Emit(OpCodes.Ldarg_1);
gen.Emit(OpCodes.Call, setMethod);
gen.Emit(OpCodes.Ret);

                                        method.CreateDelegate(typeof(Action<TTarget, TParam>));
```

```
static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo propertyInfo)
{
	var method = new DynamicMethod(propertyInfo.Name + "SetterTyped", null,
		new[] { typeof(TTarget), typeof(TParam) }, Module, true);

	var gen = method.GetILGenerator();

	var setMethod = propertyInfo.GetSetMethod();

	gen.Emit(OpCodes.Ldarg_0);
	gen.Emit(OpCodes.Ldarg_1);
	gen.Emit(OpCodes.Call, setMethod);
	gen.Emit(OpCodes.Ret);

	return (Action<TTarget, TParam>)method.CreateDelegate(typeof(Action<TTarget, TParam>));
}
```

Intermediate Language

```
[BenchmarkCategory(GetCategory), Benchmark]
public string GetViaILGen()
    => _ilGenGetter(_test);

[BenchmarkCategory(SetCategory), Benchmark]
public void SetViaILGen()
    => _ilGenSetter(_test, nameof(SetViaILGen));
```

Intermediate Language

Method	Mean	Error	StdDev	Ratio
SetViaProperty	1.8766 ns	0.0818 ns	0.0909 ns	1.00
SetViaDelegate	3.6128 ns	0.0387 ns	0.0343 ns	1.93
SetViaILGen	4.6452 ns	0.1195 ns	0.1117 ns	2.48
SetViaFastMember	25.9617 ns	0.2093 ns	0.1958 ns	13.86
SetViaReflectionCached	143.1944 ns	0.6018 ns	0.5629 ns	76.44
SetViaReflection	196.0413 ns	1.6216 ns	1.3541 ns	105.35

Intermediate Language

Method	Mean	Error	StdDev	Ratio
GetViaProperty	0.3240 ns	0.0635 ns	0.0563 ns	1.00
GetViaDelegate	2.7044 ns	0.0256 ns	0.0240 ns	8.57
GetViaILGen	3.1889 ns	0.1557 ns	0.3109 ns	10.18
GetViaFastMember	24.0186 ns	0.1106 ns	0.1035 ns	76.05
GetViaReflectionCached	80.6136 ns	0.5869 ns	0.5490 ns	255.63
GetViaReflection	131.9475 ns	1.0697 ns	0.9483 ns	418.20

Compiled Expression Trees

Expression Trees

Expression trees represent code in a tree-like data structure, where each node is an expression, for example, a method call or a binary operation such as `x < y`.

You can compile and run code represented by expression trees. This enables dynamic modification of executable code, the execution of LINQ queries in various databases, and the creation of dynamic queries. For more information about expression trees in LINQ, see How to use expression trees to build dynamic queries (C#).

Expression Trees

- Lambda Expressions

Expression Trees

- Lambda Expressions
- API

Expression Trees: Lambda

```
Func<int, bool>  lambda = num => num < 5;
```

Expression Trees: Lambda

```
Expression<Func<int, bool>> lambda = num => num < 5;
```

Expression Trees: API

```
ParameterExpression numParam = Expression.Parameter(typeof(int), "num");
ConstantExpression five = Expression.Constant(5, typeof(int));
BinaryExpression numLessThanFive = Expression.LessThan(numParam, five);
Expression<Func<int, bool>> lambda =
    Expression.Lambda<Func<int, bool>>(
        numLessThanFive,
        new ParameterExpression[] { numParam });
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));



}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(                                    );


}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(targetParam,                    );


}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(targetParam, setMethod,           );


}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(targetParam, setMethod, valueParam);



}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(targetParam, setMethod, valueParam);

        Expression
    .Lambda

}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(targetParam, setMethod, valueParam);

        Expression
    .Lambda<Action<TTarget, TParam>>(callSetter, targetParam,          )

}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(targetParam, setMethod, valueParam);

        Expression
    .Lambda<Action<TTarget, TParam>>(callSetter, targetParam, valueParam)

}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(targetParam, setMethod, valueParam);

        Expression
        .Lambda<Action<TTarget, TParam>>(callSetter, targetParam, valueParam)
        .Compile();
}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(targetParam, setMethod, valueParam);

    return Expression
        .Lambda<Action<TTarget, TParam>>(callSetter, targetParam, valueParam)
        .Compile();
}
```

```
public static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    var setMethod = property.GetSetMethod();
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(targetParam, setMethod, valueParam);

    return Expression
        .Lambda<Action<TTarget, TParam>>(callSetter, targetParam, valueParam)
        .Compile();
}
```

Expression Trees: API

```
[BenchmarkCategory(GetCategory), Benchmark]
public string GetViaCompiledExpressionTrees()
    => _compiledExpressionTreesGetter(_test);
```

Expression Trees: API

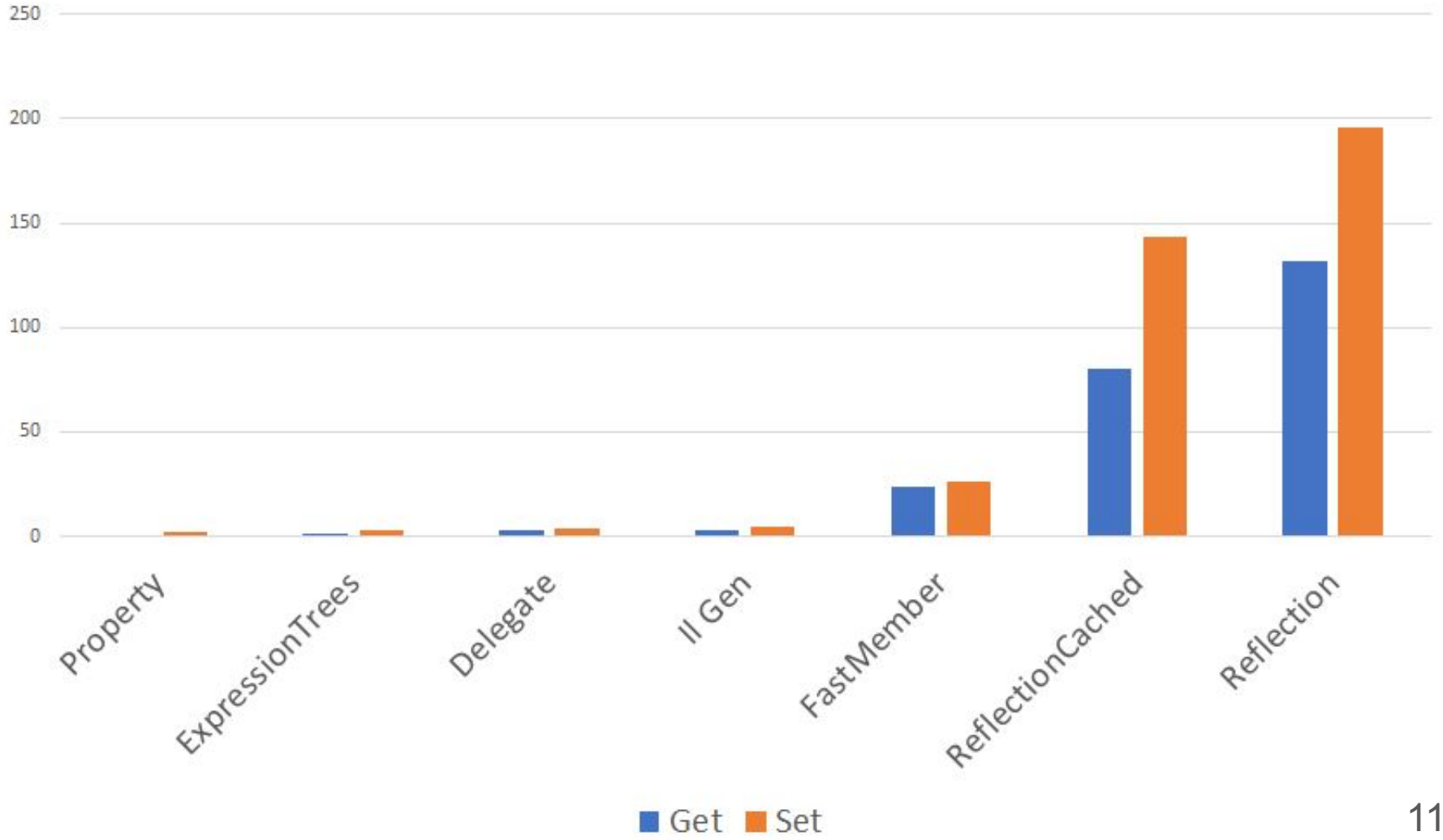
```
[BenchmarkCategory(GetCategory), Benchmark]
public string GetViaCompiledExpressionTrees()
    => _compiledExpressionTreesGetter(_test);

[BenchmarkCategory(SetCategory), Benchmark]
public void SetViaCompiledExpressionTrees()
    => _compiledExpressionTreesSetter(_test, nameof(SetViaCompiledExpressionTrees));
```

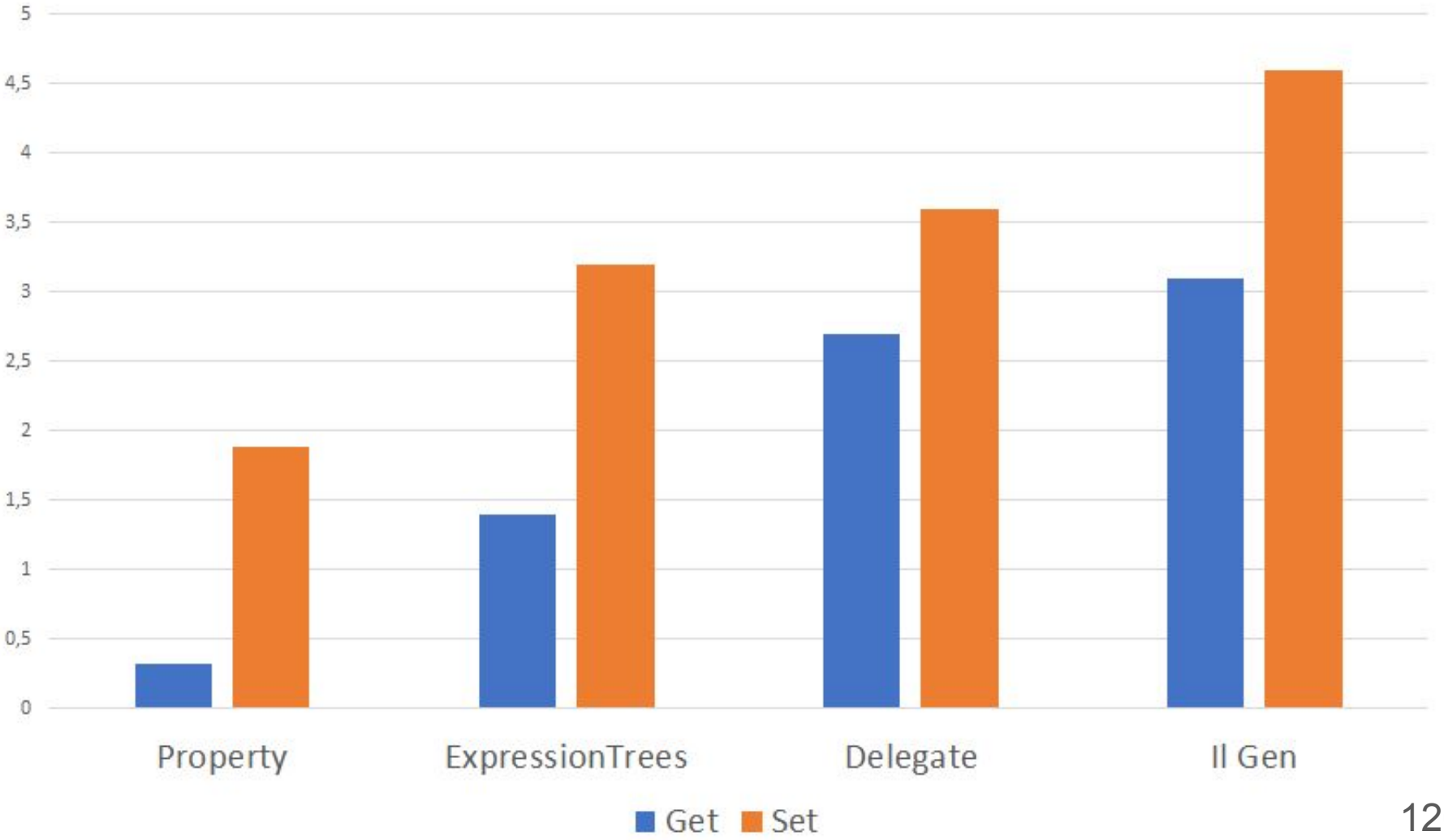
Expression Trees: API

Method	Mean	Error	StdDev	Ratio
SetViaProperty	1.8766 ns	0.0818 ns	0.0909 ns	1.00
SetViaCompiledExpressionTrees	3.2353 ns	0.1052 ns	0.1211 ns	1.73
SetViaDelegate	3.6128 ns	0.0387 ns	0.0343 ns	1.93
SetViaILGen	4.6452 ns	0.1195 ns	0.1117 ns	2.48
SetViaFastMember	25.9617 ns	0.2093 ns	0.1958 ns	13.86
SetViaReflectionCached	143.1944 ns	0.6018 ns	0.5629 ns	76.44
SetViaReflection	196.0413 ns	1.6216 ns	1.3541 ns	105.35

Expression Trees: API

Method	Mean	Error	StdDev	Ratio
GetViaProperty	0.3240 ns	0.0635 ns	0.0563 ns	1.00
GetViaCompiledExpressionTrees	1.3426 ns	0.1167 ns	0.1981 ns	4.18
GetViaDelegate	2.7044 ns	0.0256 ns	0.0240 ns	8.57
GetViaILGen	3.1889 ns	0.1557 ns	0.3109 ns	10.18
GetViaFastMember	24.0186 ns	0.1106 ns	0.1035 ns	76.05
GetViaReflectionCached	80.6136 ns	0.5869 ns	0.5490 ns	255.63
GetViaReflection	131.9475 ns	1.0697 ns	0.9483 ns	418.20


250
200
150
100
50
0
Property
ExpressionTrees
Delegate
Il Gen
FastMember
ReflectionCached
Reflection
Get
Set


5
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
Property
ExpressionTrees
Delegate
Il Gen
Get
Set

Ни рефлексией единой

- Reflection - медленно

Ни рефлексией единой

- Reflection - медленно
- Альтернативы “из коробки” - быстрее

Ни рефлексией единой

- Reflection - медленно
- Альтернативы “из коробки” - быстрее
  - Требуют усилий

“Погоди! Но ...”

“Делегаты специфицированы конкретными типами целевого объекта и его свойства!”

```
Delegate.CreateDelegate(typeof(Func<Test, string>), PropertyGetMethod);
```

```
Delegate.CreateDelegate(typeof(Func<Test, string>), PropertyGetMethod);

Func<Test, string> _compiledExpressionTreesGetter
    = CompiledExpressionTreesHelper.GenerateGetter<Test, string>(CachedPropertyInfo);
```

```
Delegate.CreateDelegate(typeof(Func<Test, string>), PropertyGetMethod);

Func<Test, string> _compiledExpressionTreesGetter
    = CompiledExpressionTreesHelper.GenerateGetter<Test, string>(CachedPropertyInfo);

Func<Test, string> _ilGenGetter
    = ILGenHelper.GenerateGetter<Test, string>(CachedPropertyInfo);
```

PropertyInfo.GetValue

GetValue(Object)

Returns the property value of a specified object.

C# Copy

```
public object GetValue (object obj);
```

Delegate.CreateDelegate

```
Delegate.CreateDelegate(typeof(Func<Test, string>), PropertyGetMethod);
```

Delegate.CreateDelegate

```
Delegate.CreateDelegate(typeof(Func<object, object>), PropertyGetMethod);
```

System.ArgumentException

Cannot bind to the target method because its signature is not compatible with that of the delegate type.

at System.Delegate.CreateDelegate(Type type, MethodInfo method, Boolean throwOnBindFailure)

at System.Delegate.CreateDelegate(Type type, MethodInfo method)

at Program.Main() in Program.cs

“Эх! А счастье было так близко...”

Delegate (object, object)

```
var setter = CreateDelegate<Action<TTarget, TParam>>(method);
```

Delegate (object, object)

```
var setter = CreateDelegate<Action<TTarget, TParam>>(method);

        (object target, object value) => setter((TTarget)target, (TParam)value);
```

Delegate (object, object)

```
static Action<object, object> GetSetMethodHelper<TTarget, TParam>(MethodInfo method)
    where TTarget : class
{
    var setter = CreateDelegate<Action<TTarget, TParam>>(method);

    return (object target, object value) => setter((TTarget)target, (TParam)value);
}
```

Delegate (object, object)

```
var genericHelper = helperClass.GetMethod(nameof(GetSetMethodHelper), bindingFlags);
```

Delegate (object, object)

```
var genericHelper = helperClass.GetMethod(nameof(GetSetMethodHelper), bindingFlags);

var targetType = propertyInfo.DeclaringType;
var propertyType = propertyInfo.PropertyType;
```

Delegate (object, object)

```
var genericHelper = helperClass.GetMethod(nameof(GetSetMethodHelper), bindingFlags);

var targetType = propertyInfo.DeclaringType;
var propertyType = propertyInfo.PropertyType;

var constructedHelper = genericHelper.MakeGenericMethod(targetType, propertyType);
```

Delegate (object, object)

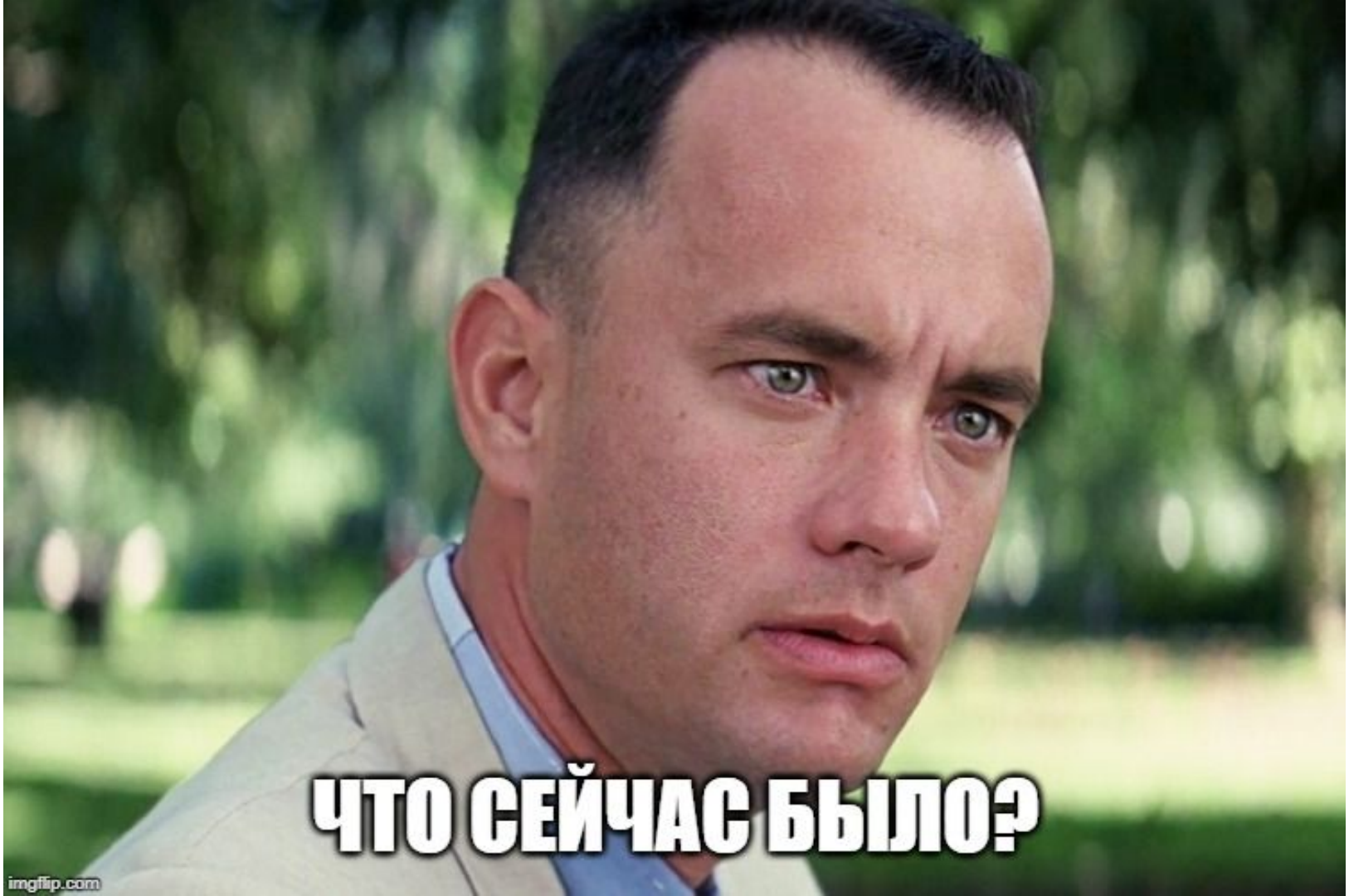
```
var setMethod = propertyInfo.GetSetMethod(true);

var genericHelper = helperClass.GetMethod(nameof(GetSetMethodHelper), bindingFlags);

var targetType = propertyInfo.DeclaringType;
var propertyType = propertyInfo.PropertyType;

var constructedHelper = genericHelper.MakeGenericMethod(targetType, propertyType);
```

Delegate (object, object)

```
var setMethod = propertyInfo.GetSetMethod(true);

var genericHelper = helperClass.GetMethod(nameof(GetSetMethodHelper), bindingFlags);

var targetType = propertyInfo.DeclaringType;
var propertyType = propertyInfo.PropertyType;

var constructedHelper = genericHelper.MakeGenericMethod(targetType, propertyType);

var result = constructedHelper.Invoke(null, new object[] { setMethod });
```

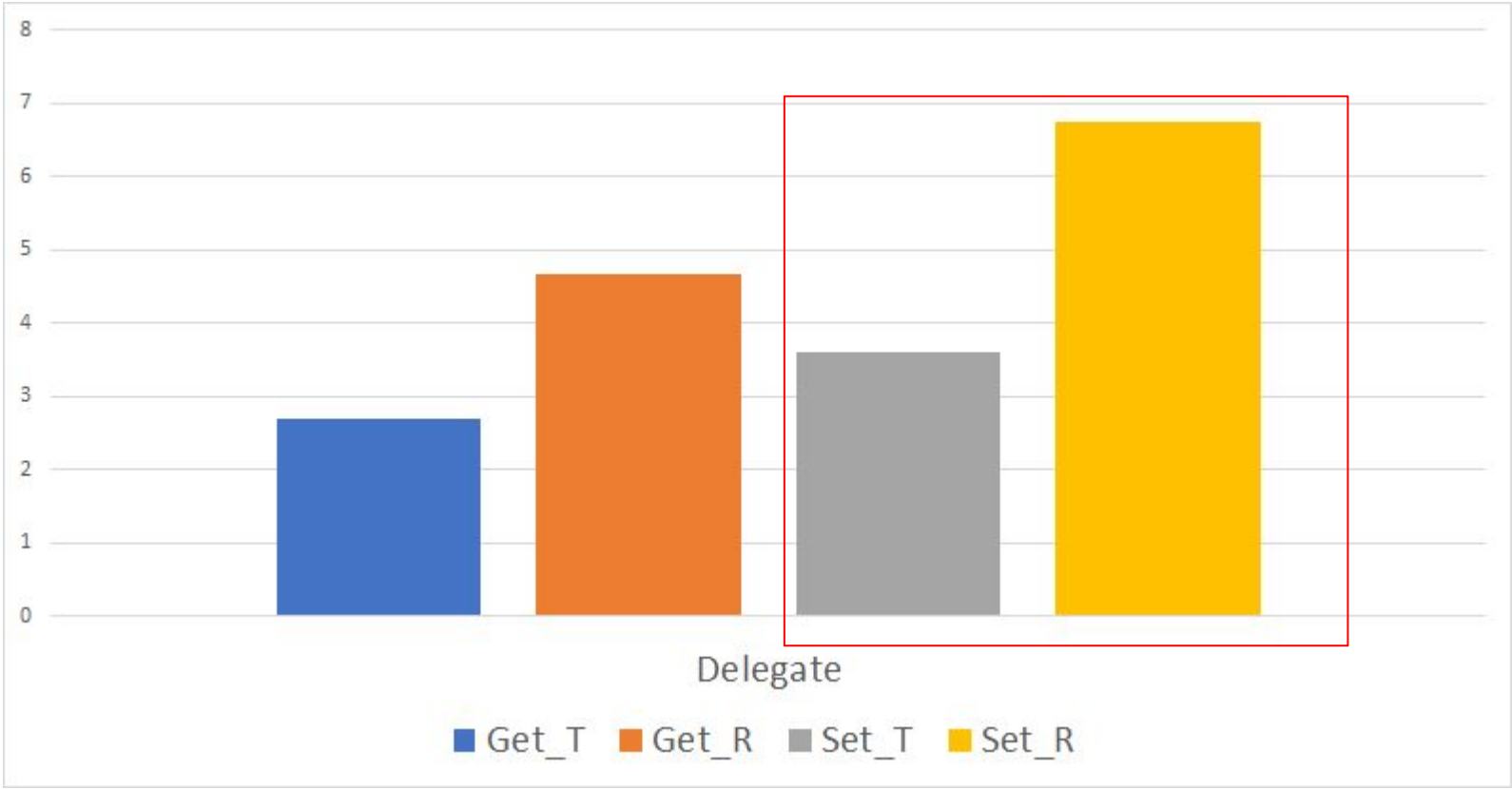
Delegate (object, object)

```
public static Action<object, object> GetSetMethod(PropertyInfo propertyInfo)
{
    var setMethod = propertyInfo.GetSetMethod(true);

    var genericHelper = helperClass.GetMethod(nameof(GetSetMethodHelper), bindingFlags);

    var targetType = propertyInfo.DeclaringType;
    var propertyType = propertyInfo.PropertyType;

    var constructedHelper = genericHelper.MakeGenericMethod(targetType, propertyType);

    var result = constructedHelper.Invoke(null, new object[] { setMethod });

    return (Action<object, object>)result;
}
```


ЧТО СЕЙЧАС БЫЛО?
imgflip.com

Delegate (object, object)

Delegate (object, object)



ILGen (object, object)

```
static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo property)
{
    ...prepare...

    gen.Emit(OpCodes.Ldarg_0);
    gen.Emit(OpCodes.Ldarg_1);
    gen.Emit(OpCodes.Call, setMethod);
    gen.Emit(OpCodes.Ret);

    return (Action<TTarget, TParam>)method.CreateDelegate(typeof(Action<TTarget, TParam>));
}
```

ILGen (object, object)

```
static Action<object, object> GenerateSetter(PropertyInfo property)
{
    ...prepare...

    gen.Emit(OpCodes.Ldarg_0);
    gen.Emit(OpCodes.Castclass, property.DeclaringType);
    gen.Emit(OpCodes.Ldarg_1);
    gen.Emit(propertyType.IsValueType ?
        OpCodes.Unbox_Any : OpCodes.Castclass, propertyType);
    gen.Emit(OpCodes.Call, setMethod);
    gen.Emit(OpCodes.Ret);

    return (Action<object, object>)method.CreateDelegate(typeof(Action<object, object>));
}
```

ILGen (object, object)

```
static Action<object, object> GenerateSetter(PropertyInfo property)
{
    ...prepare...

    gen.Emit(OpCodes.Ldarg_0);
    gen.Emit(OpCodes.Castclass, property.DeclaringType);
    gen.Emit(OpCodes.Ldarg_1);
    gen.Emit(propertyType.IsValueType ?
        OpCodes.Unbox_Any : OpCodes.Castclass, propertyType);
    gen.Emit(OpCodes.Call, setMethod);
    gen.Emit(OpCodes.Ret);

    return (Action<object, object>)method.CreateDelegate(typeof(Action<object, object>));
}
```

ILGen (object, object)

```
static Action<object, object> GenerateSetter(PropertyInfo property)
{
    ...prepare...

    gen.Emit(OpCodes.Ldarg_0);
    gen.Emit(OpCodes.Castclass, property.DeclaringType);
    gen.Emit(OpCodes.Ldarg_1);
    gen.Emit(propertyType.IsValueType ?
        OpCodes.Unbox_Any : OpCodes.Castclass, propertyType);
    gen.Emit(OpCodes.Call, setMethod);
    gen.Emit(OpCodes.Ret);

    return (Action<object, object>)method.CreateDelegate(typeof(Action<object, object>));
}
```

ILGen (object, object)

ILGen (object, object)

Expression Trees (object, object)

```
static Action<TTarget, TParam> GenerateSetter<TTarget, TParam>(PropertyInfo propertyInfo)
{
    var setMethod = propertyInfo.GetSetMethod(true);
    var targetParam = Expression.Parameter(typeof(TTarget));
    var valueParam = Expression.Parameter(typeof(TParam));

    var callSetter = Expression.Call(targetParam, setMethod, valueParam);

    return Expression
        .Lambda<Action<TTarget, TParam>>(callSetter, targetParam, valueParam)
        .Compile();
}
```

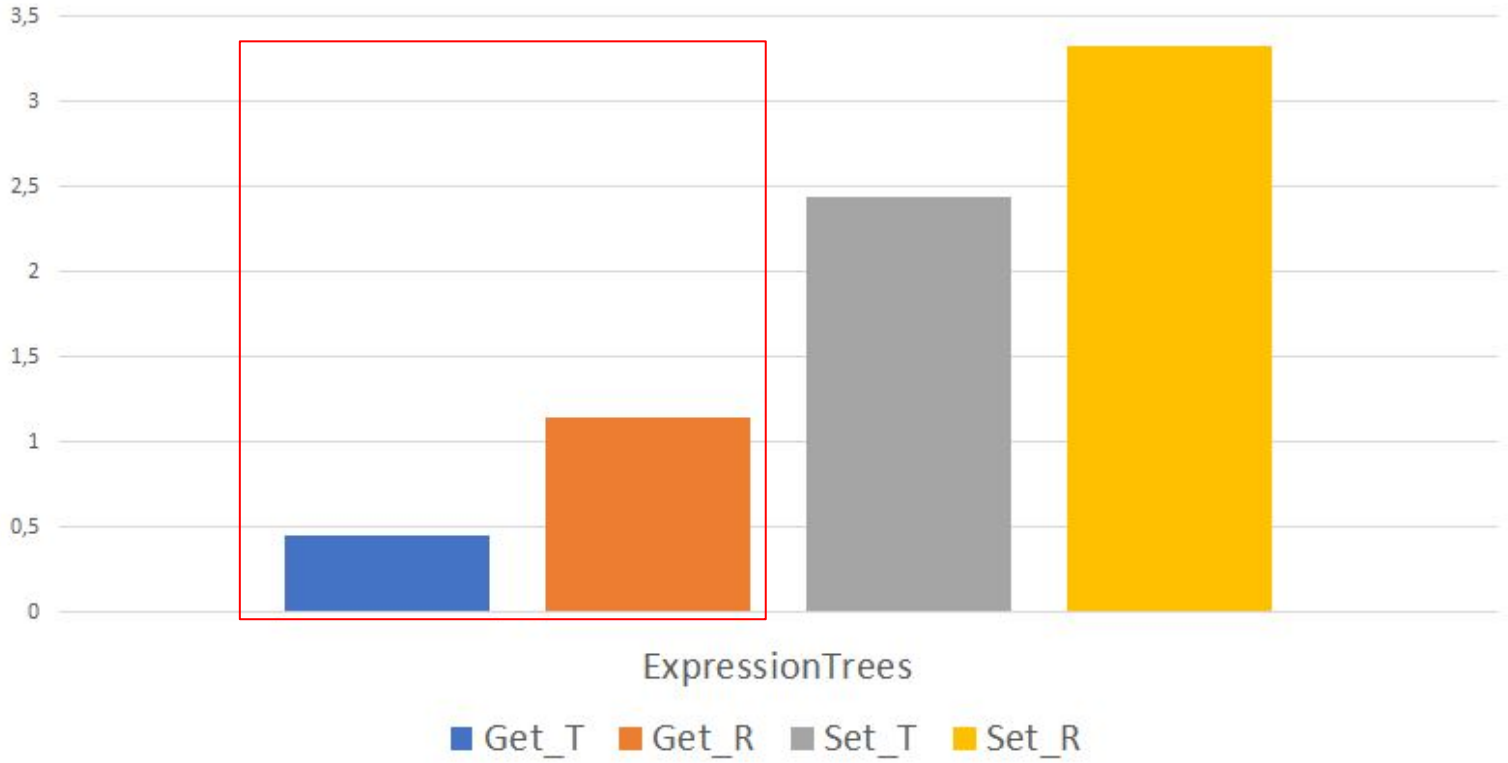
Expression Trees (object, object)

```
static Action<object, object> GenerateSetter(PropertyInfo propertyInfo)
{
    var setMethod = propertyInfo.GetSetMethod(true);
    var targetParam = Expression.Parameter(typeof(object));
    var valueParam = Expression.Parameter(typeof(object));

    var convertedTarget = Expression.Convert(targetParam, propertyInfo.DeclaringType);
    var convertedValue = Expression.Convert(valueParam, propertyInfo.PropertyType);

    var callSetter = Expression.Call(convertedTarget, setMethod, convertedValue);

    return Expression
        .Lambda<Action<object, object>>(callSetter, targetParam, valueParam)
        .Compile();
}
```

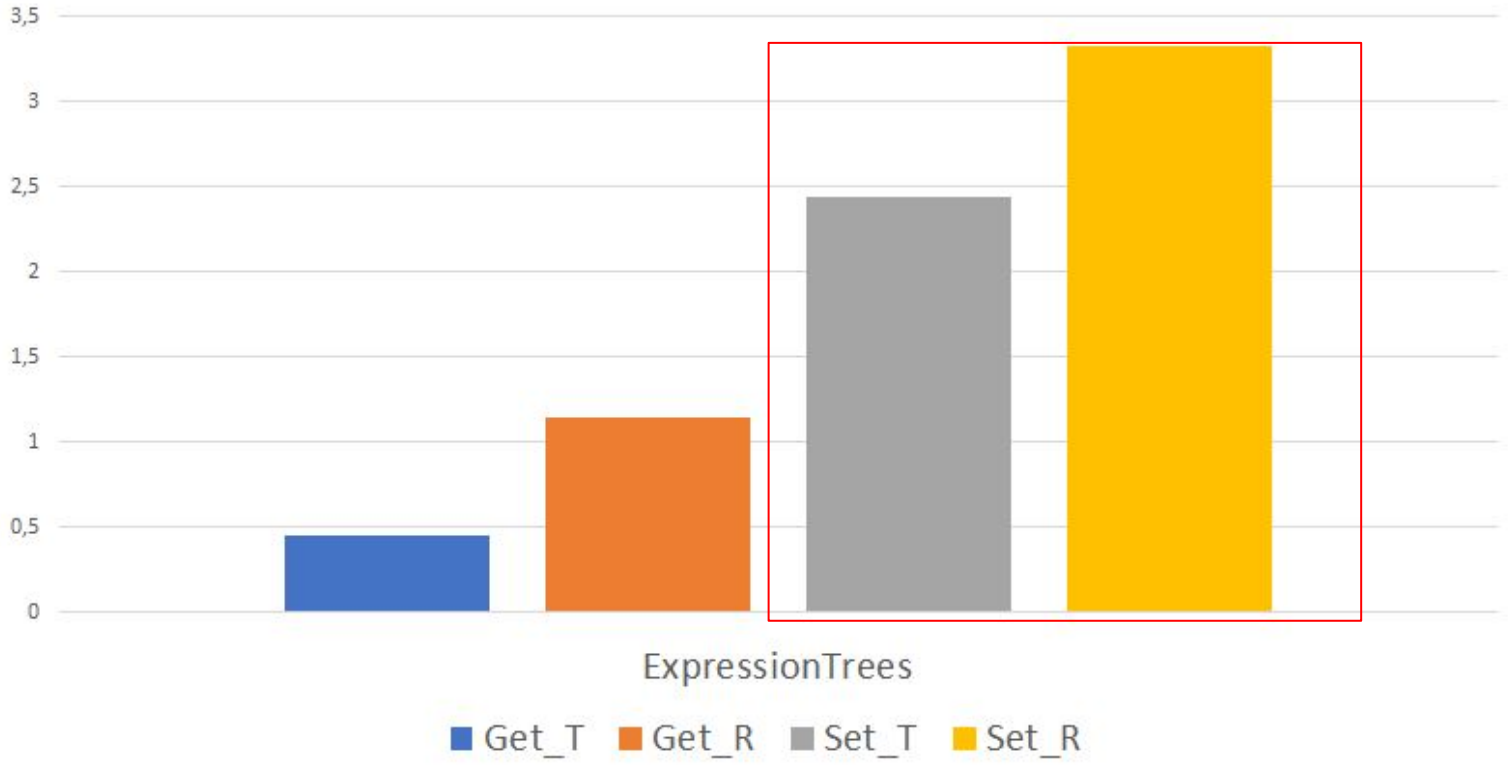
Expression Trees (object, object)

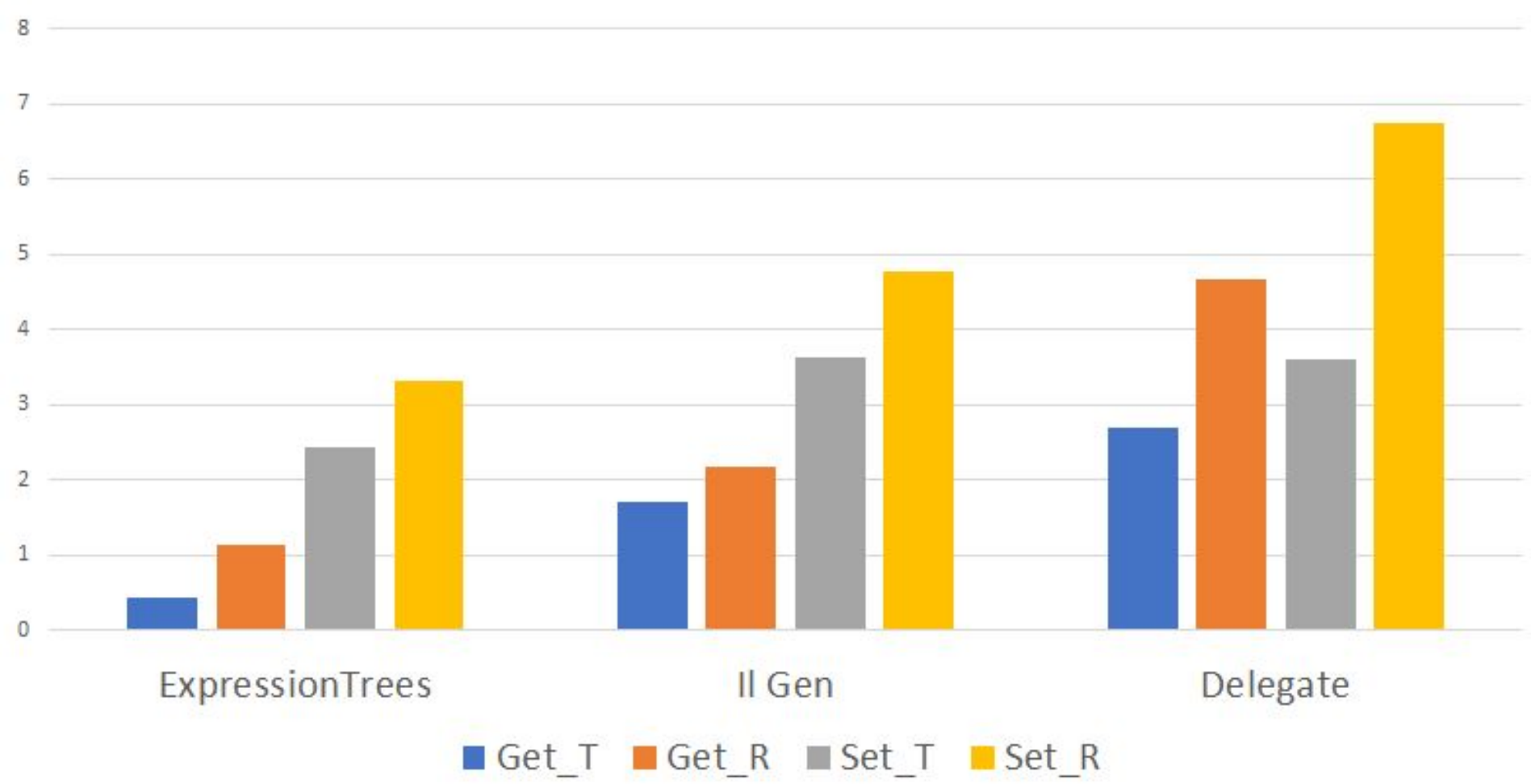
```
static Action<object, object> GenerateSetter(PropertyInfo propertyInfo)
{
    var setMethod = propertyInfo.GetSetMethod(true);
    var targetParam = Expression.Parameter(typeof(object));
    var valueParam = Expression.Parameter(typeof(object));

    var convertedTarget = Expression.Convert(targetParam, propertyInfo.DeclaringType);
    var convertedValue = Expression.Convert(valueParam, propertyInfo.PropertyType);

    var callSetter = Expression.Call(convertedTarget, setMethod, convertedValue);

    return Expression
        .Lambda<Action<object, object>>(callSetter, targetParam, valueParam)
        .Compile();
}
```

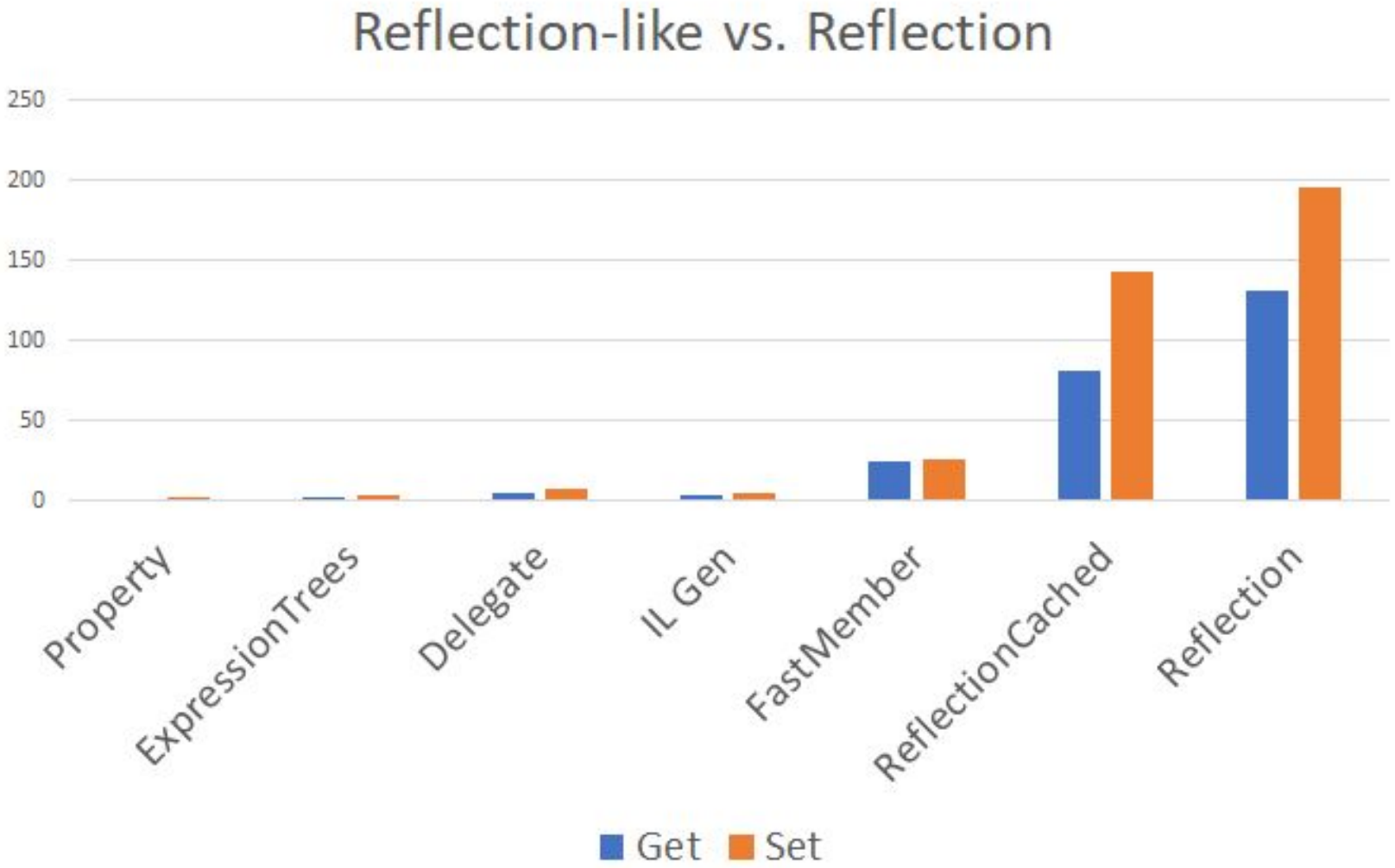
Expression Trees (object, object)



Expression Trees (object, object)




8
7
6
5
4
3
2
1
0
ExpressionTrees
Il Gen
Delegate
Get_T
Get_R
Set_T
Set_R


Reflection-like vs. Reflection
250
200
150
100
50
0
Property
ExpressionTrees
Delegate
IL Gen
FastMember
ReflectionCached
Reflection
Get
Set

Ни рефлексией единой

- Reflection - медленно

Ни рефлексией единой

- Reflection - медленно
  - Нетипизированный доступ

Ни рефлексией единой

- Reflection - медленно
  - Нетипизированный доступ
- Альтернативы “из коробки” - быстрее

Ни рефлексией единой

- Reflection - медленно
  - Нетипизированный доступ
- Альтернативы “из коробки” - быстрее
  - Типизированный доступ

Ни рефлексией единой

- Reflection - медленно
  - Нетипизированный доступ
- Альтернативы “из коробки” - быстрее
  - Типизированный доступ
  - Reflection-like

Ни рефлексией единой

- Reflection - медленно
  - Нетипизированный доступ
- Альтернативы “из коробки” - быстрее
  - Типизированный доступ
  - Reflection-like
  - Требуют усилий

“А что если ValueType? Boxing же!”

ValueType

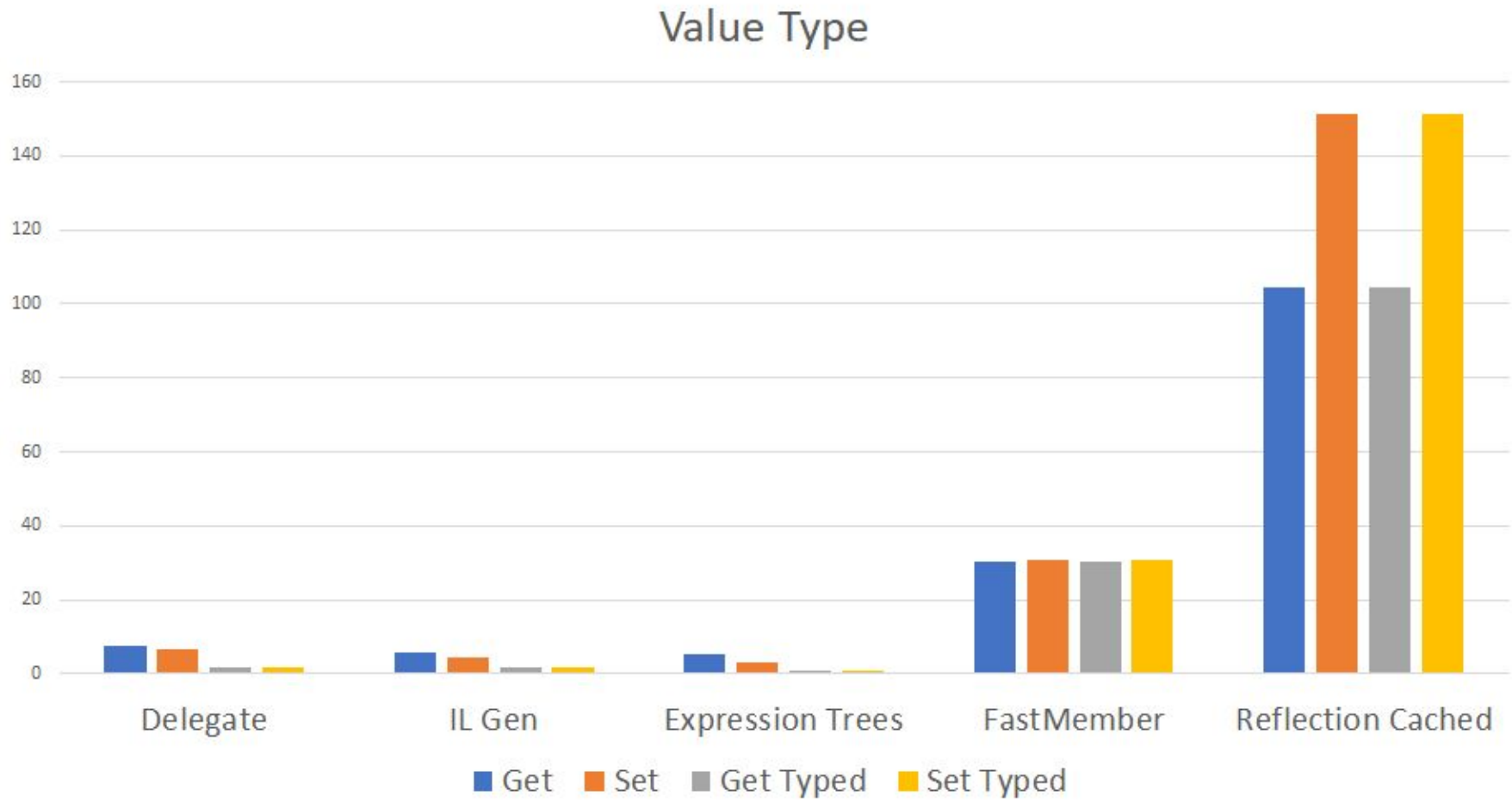
Method	Mean	Allocated
GetViaReflectionCached	104.4789 ns	24 B
SetViaReflectionCached	151.3353 ns	88 B
GetViaFastMember	30.3511 ns	24 B
SetViaFastMember	30.9700 ns	24 B

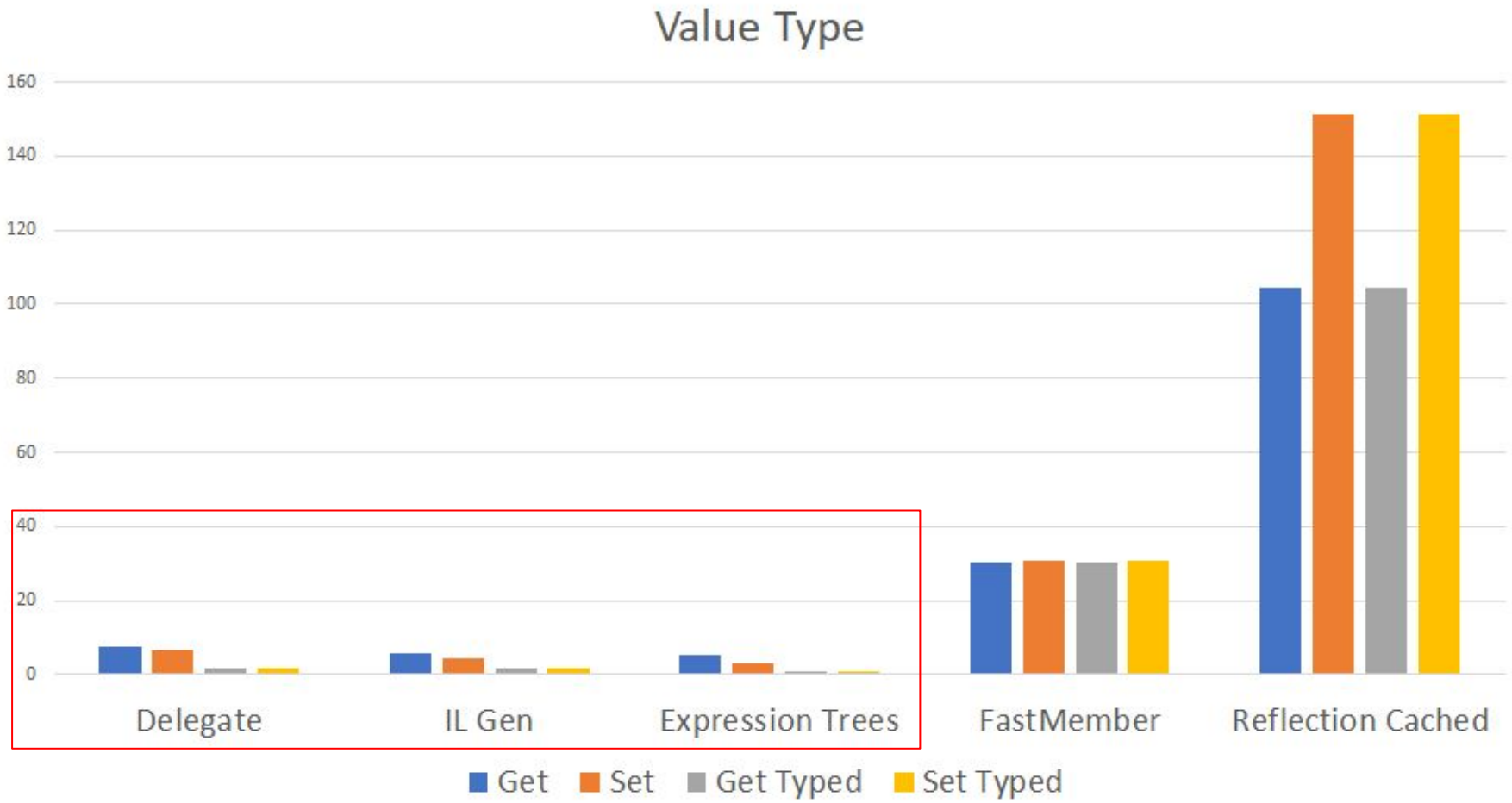
ValueType

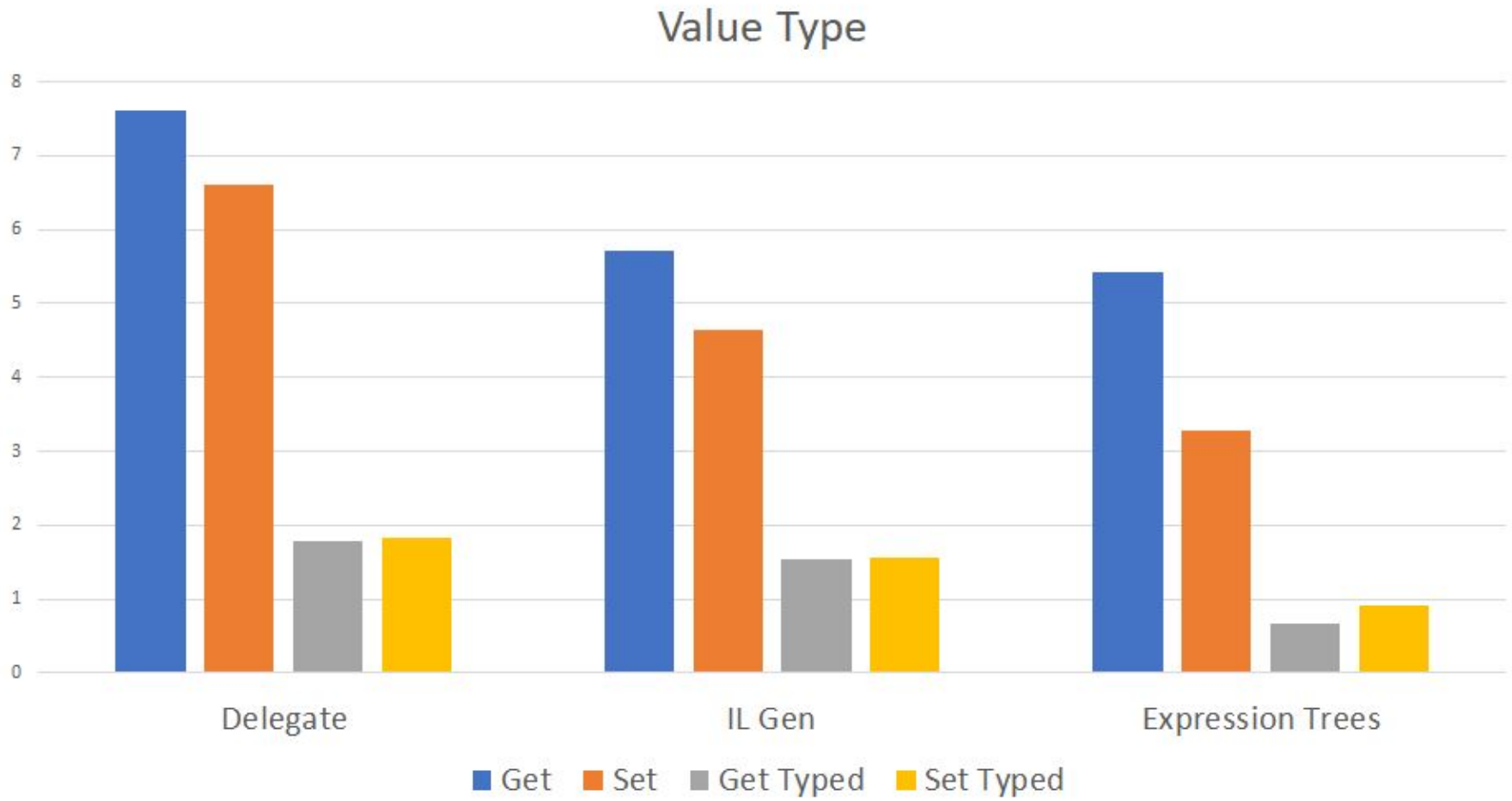
Method	Mean	Allocated
GetViaDelegateReflectionStyle	6.6041 ns	24 B
SetViaDelegateReflectionStyle	7.6047 ns	24 B
GetViaILGenReflectionStyle	4.6524 ns	24 B
SetViaILGenReflectionStyle	5.7267 ns	24 B
GetViaCompiledExpressionTreesReflectionStyle	3.2900 ns	24 B
SetViaCompiledExpressionTreesReflectionStyle	5.4248 ns	24 B

ValueType

Method	Mean	Allocated
GetViaDelegateTyped	1.7885 ns	-
SetViaDelegateTyped	1.8371 ns	-
GetViaILGenTyped	1.5500 ns	-
SetViaILGenTyped	1.5691 ns	-
GetViaCompiledExpressionTreesTyped	0.6634 ns	-
SetViaCompiledExpressionTreesTyped	0.9252 ns	-


Value Type
160
140
120
100
80
60
40
20
0
Delegate
IL Gen
Expression Trees
FastMember
Reflection Cached
Get
Set
Get Typed
Set Typed


Value Type
160
140
120
100
80
60
40
20
0
Delegate
IL Gen
Expression Trees
FastMember
Reflection Cached
Get
Set
Get Typed
Set Typed


Value Type
8
7
6
5
4
3
2
1
0
Delegate
IL Gen
Expression Trees
Get
Set
Get Typed
Set Typed

Ни рефлексией единой: Value Type

Ни рефлексией единой: Value Type

- Reflection: boxing

Ни рефлексией единой: Value Type

- Reflection: boxing
- Альтернатива
  - Reflection-like: boxing

Ни рефлексией единой: Value Type

- Reflection: boxing
- Альтернатива
  - Reflection-like: boxing
  - Типизированный доступ: no boxing

Приватные данные

```
BindingFlags BindingFlags = System.Reflection.BindingFlags.Instance
                          | System.Reflection.BindingFlags.Public;
```

```
BindingFlags BindingFlags = System.Reflection.BindingFlags.Instance
                          | System.Reflection.BindingFlags.NonPublic
                          | System.Reflection.BindingFlags.Public;
```

```
BindingFlags BindingFlags = System.Reflection.BindingFlags.Instance
                          | System.Reflection.BindingFlags.NonPublic
                          | System.Reflection.BindingFlags.Public;

propertyInfo.GetGetMethod(nonPublic:true);
```

```
BindingFlags BindingFlags = System.Reflection.BindingFlags.Instance
                          | System.Reflection.BindingFlags.NonPublic
                          | System.Reflection.BindingFlags.Public;

FieldInfo field = TargetType.GetField(FieldName, BindingFlags);
```

Доступ к данным

Доступ к данным

- Reflection

Доступ к данным

- Reflection:
  - Медленно

Доступ к данным

- Reflection:
  - Медленно
  - Boxing

Доступ к данным

- Reflection:
  - Медленно
  - Boxing
- Альтернативы

Доступ к данным

- Reflection:
  - Медленно
  - Boxing
- Альтернативы:
  - Сильно быстрее Reflection

Доступ к данным

- Reflection:
  - Медленно
  - Boxing
- Альтернативы:
  - Сильно быстрее Reflection
  - Boxing

Доступ к данным

- Reflection:
  - Медленно
  - Boxing
- Альтернативы:
  - Сильно быстрее Reflection
  - Boxing / No Boxing

Доступ к данным

- Reflection:
  - Медленно
  - Boxing
- Альтернативы:
  - Сильно быстрее Reflection
  - Boxing / No Boxing
- Можно и приватные данные (поля, свойства)

Доступ к данным

- Reflection:
  - Медленно
  - Boxing
- Альтернативы:
  - Сильно быстрее Reflection
  - Boxing / No Boxing
- Можно и приватные данные (поля, свойства)
- Прямое чтение/запись не обойти


КОГДА УЖЕ
ПРО СЕРИАЛИЗАЦИЮ?
imgflip.com

Сценарий использования

Сериализация

Сериализация

```
_stream = new MemoryStream();
_writer = new BinaryWriter(_stream, Encoding.Default, true);
_reader = new BinaryReader(_stream, Encoding.Default, true);
```

Сериализация

```
[Benchmark(Baseline = true), BenchmarkCategory(WriteCategory)]
public void WriteViaProperty()
{
    _stream.Position = 0;
    _writer.Write(_test.StringProperty);
}

[Benchmark(Baseline = true), BenchmarkCategory(ReadCategory)]
public void ReadViaProperty()
{
    _stream.Position = 0;
    _test.StringProperty = _reader.ReadString();
}
```

Сериализация

```
[Benchmark(Baseline = true), BenchmarkCategory(WriteCategory)]
public void WriteViaProperty()
{
    _stream.Position = 0;
    _writer.Write(_test.StringProperty);
}

[Benchmark(Baseline = true), BenchmarkCategory(ReadCategory)]
public void ReadViaProperty()
{
    _stream.Position = 0;
    _test.StringProperty = _reader.ReadString();
}
```

Сериализация

```
[Benchmark, BenchmarkCategory(WriteCategory)]
public void WriteViaReflection()
{
    _stream.Position = 0;
    _writer.Write((string)CachedPropertyInfo.GetValue(_test, null));
}

[Benchmark, BenchmarkCategory(ReadCategory)]
public void ReadViaReflection()
{
    _stream.Position = 0;
    CachedPropertyInfo.SetValue(_test, _reader.ReadString());
}
```

Сериализация

```
[Benchmark, BenchmarkCategory(WriteCategory)]
public void WriteViaILGenReflection()
{
    _stream.Position = 0;
    _writer.Write((string)_ilGenGet(_test));
}

[Benchmark, BenchmarkCategory(ReadCategory)]
public void ReadViaIlGenReflection()
{
    _stream.Position = 0;
    _ilGenSet(_test, _reader.ReadString());
}
```

Ho!

Сгенерированный сериализатор

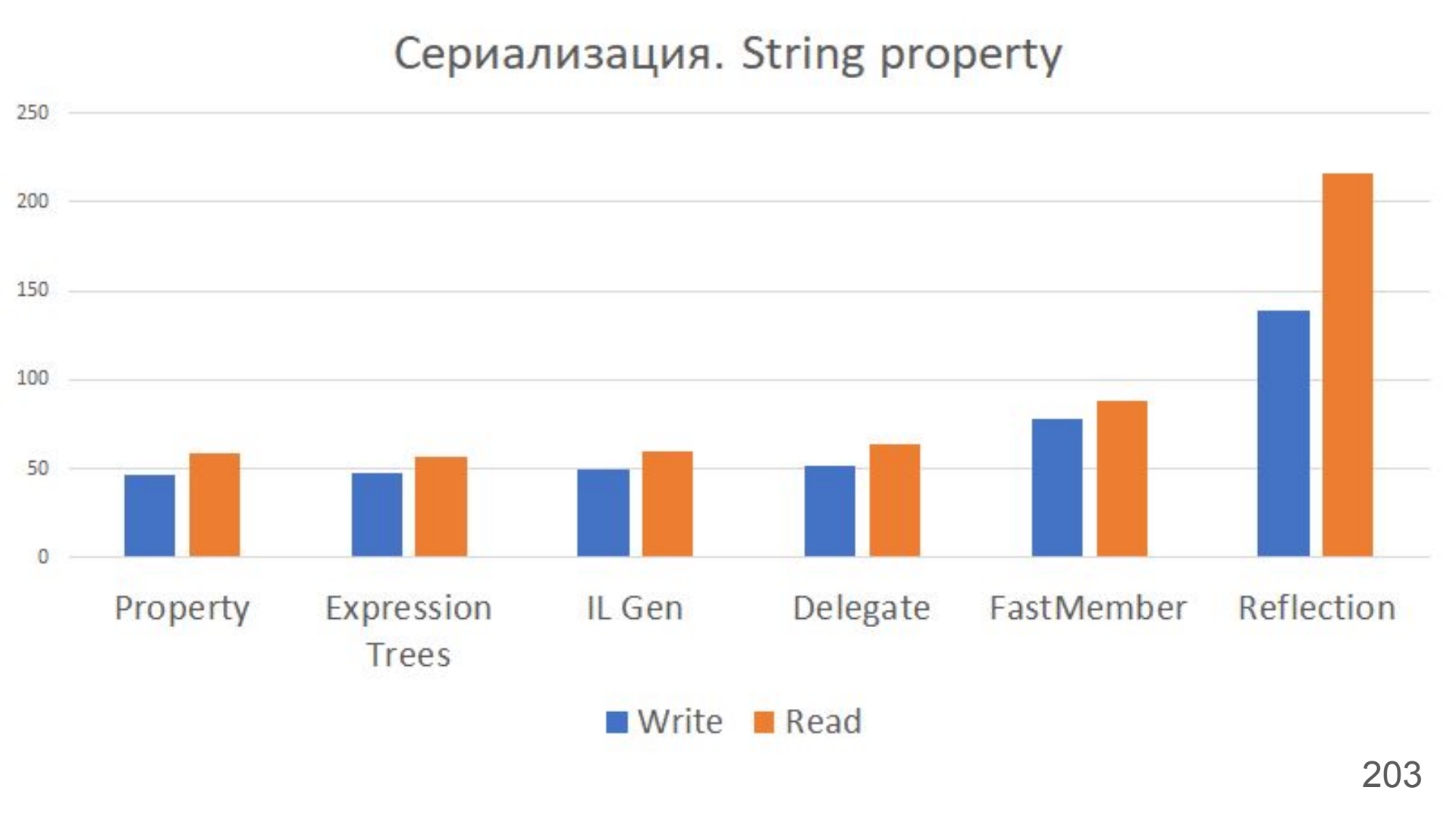
```
static void Write(Test target, BinaryWriter writer)
{
    writer.Write(target.StringProperty);
}
```

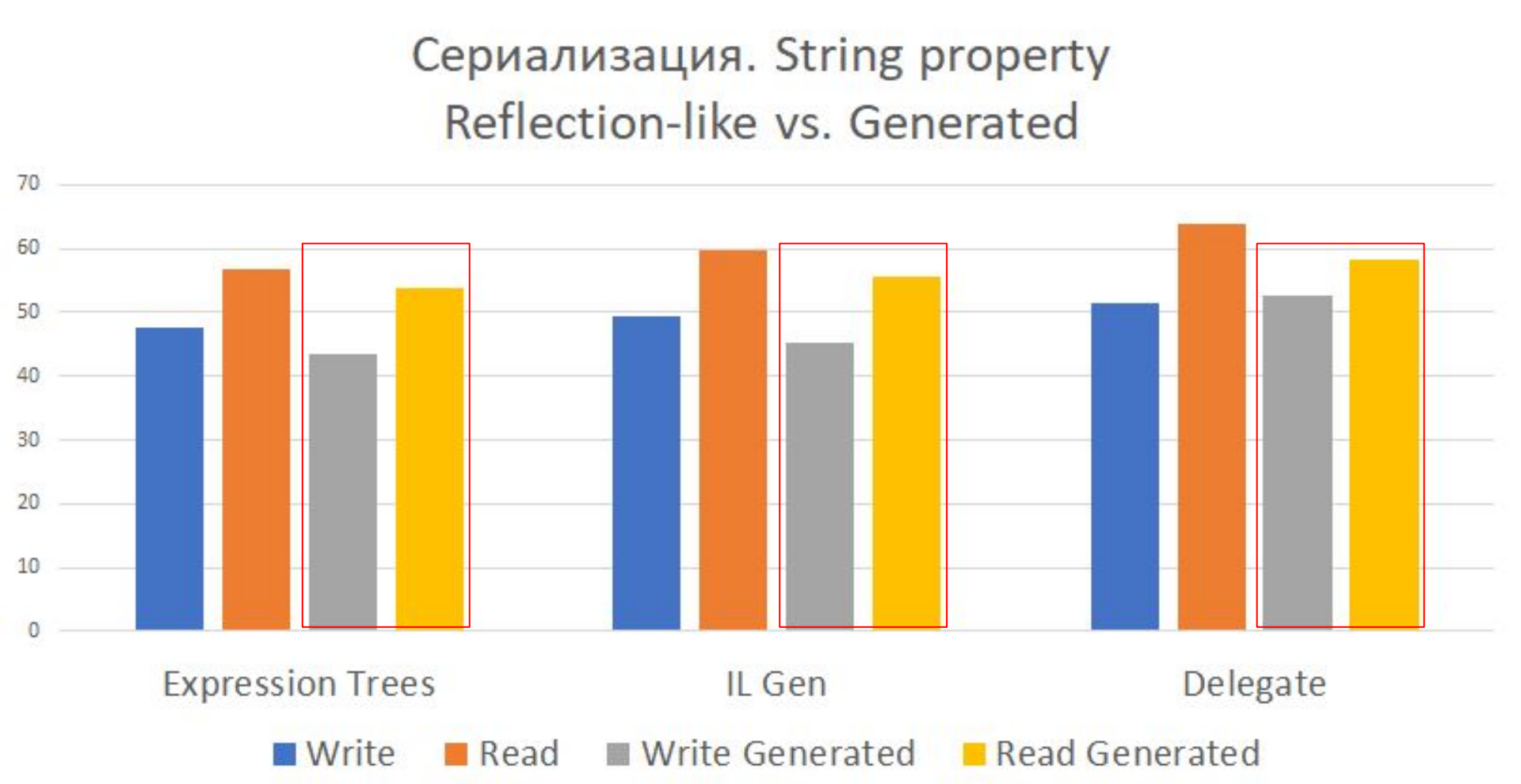
Сгенерированный сериализатор

```
static void Write(Test target, BinaryWriter writer)
{
    writer.Write(target.StringProperty);
}

static void Read(Test target, BinaryReader reader)
{
    target.StringProperty = reader.ReadString();
}
```

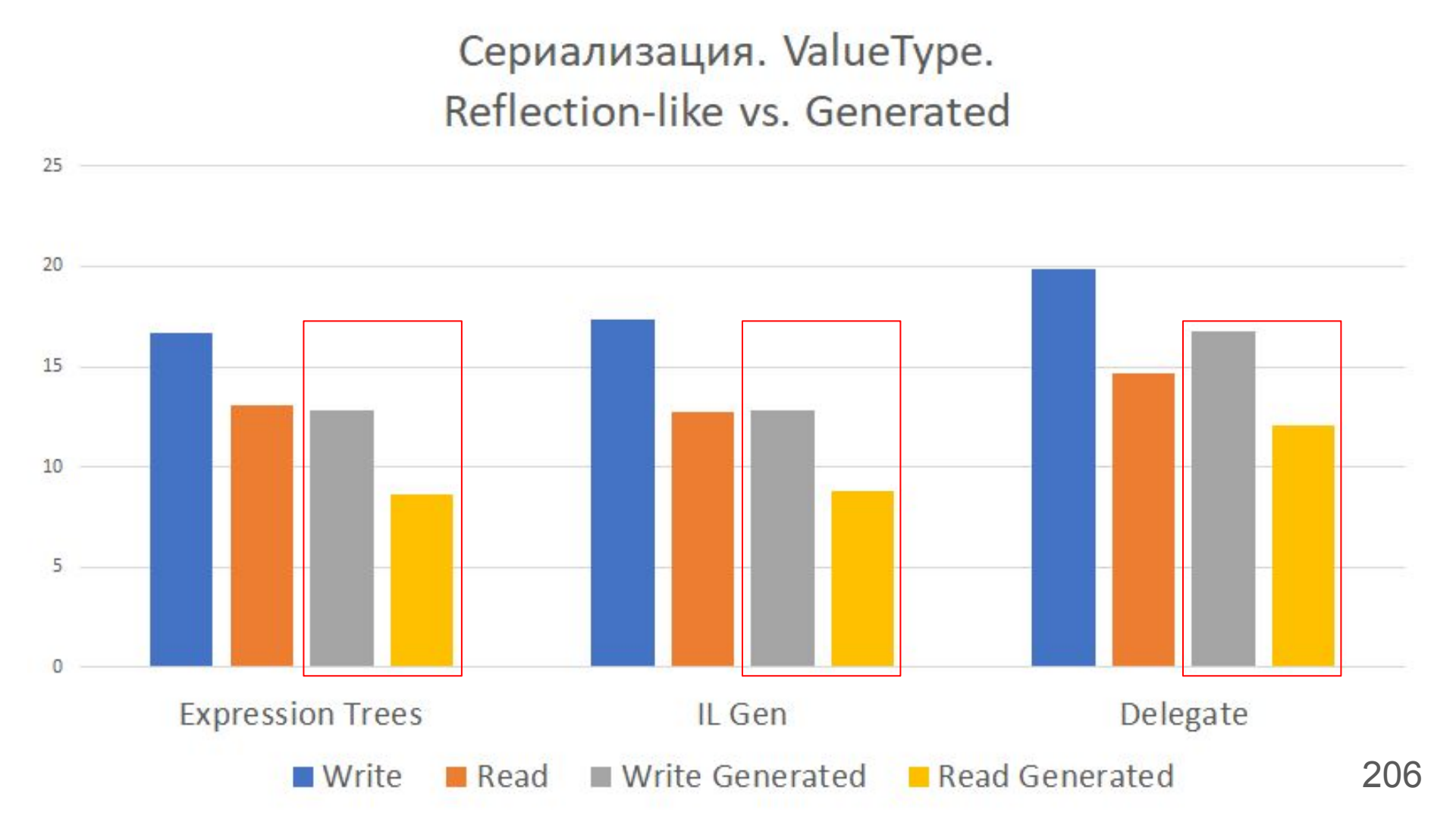
Сериализация

```
[Benchmark, BenchmarkCategory(WriteCategory)]
public void WriteViaILGen()
{
    _stream.Position = 0;
    _ilGenWrite(_test, _writer);
}

[Benchmark, BenchmarkCategory(ReadCategory)]
public void ReadViaIlGen()
{
    _stream.Position = 0;
    _ilGenRead(_test, _reader);
}
```


Сериализация. String property
250
200
150
100
50
0
Property
Expression Trees
IL Gen
Delegate
FastMember
Reflection
Write
Read


Сериализация. String property
Reflection-like vs. Generated
70
60
50
40
30
20
10
0
Expression Trees
IL Gen
Delegate
Write
Read
Write Generated
Read Generated


Сериализация. ValueType.
180
160
140
120
100
80
60
40
20
0
Property
Expression Trees
IL Gen
Delegate
FastMember
Reflection
Write
Read


Сериализация. ValueType.
Reflection-like vs. Generated
25
20
15
10
5
0
Expression Trees
IL Gen
Delegate
Write
Read
Write Generated
Read Generated

ValueType. Reflection-like

Method	Mean	Ratio	Allocated
WriteViaProperty	12.894 ns	1.00	-
WriteViaExpressionTreesReflection	17.549 ns	1.36	24 B
WriteViaILGenReflection	17.821 ns	1.38	24 B
WriteViaDelegateReflection	20.794 ns	1.61	24 B
WriteViaFastMember	46.593 ns	3.61	24 B
WriteViaReflection	122.805 ns	9.52	24 B

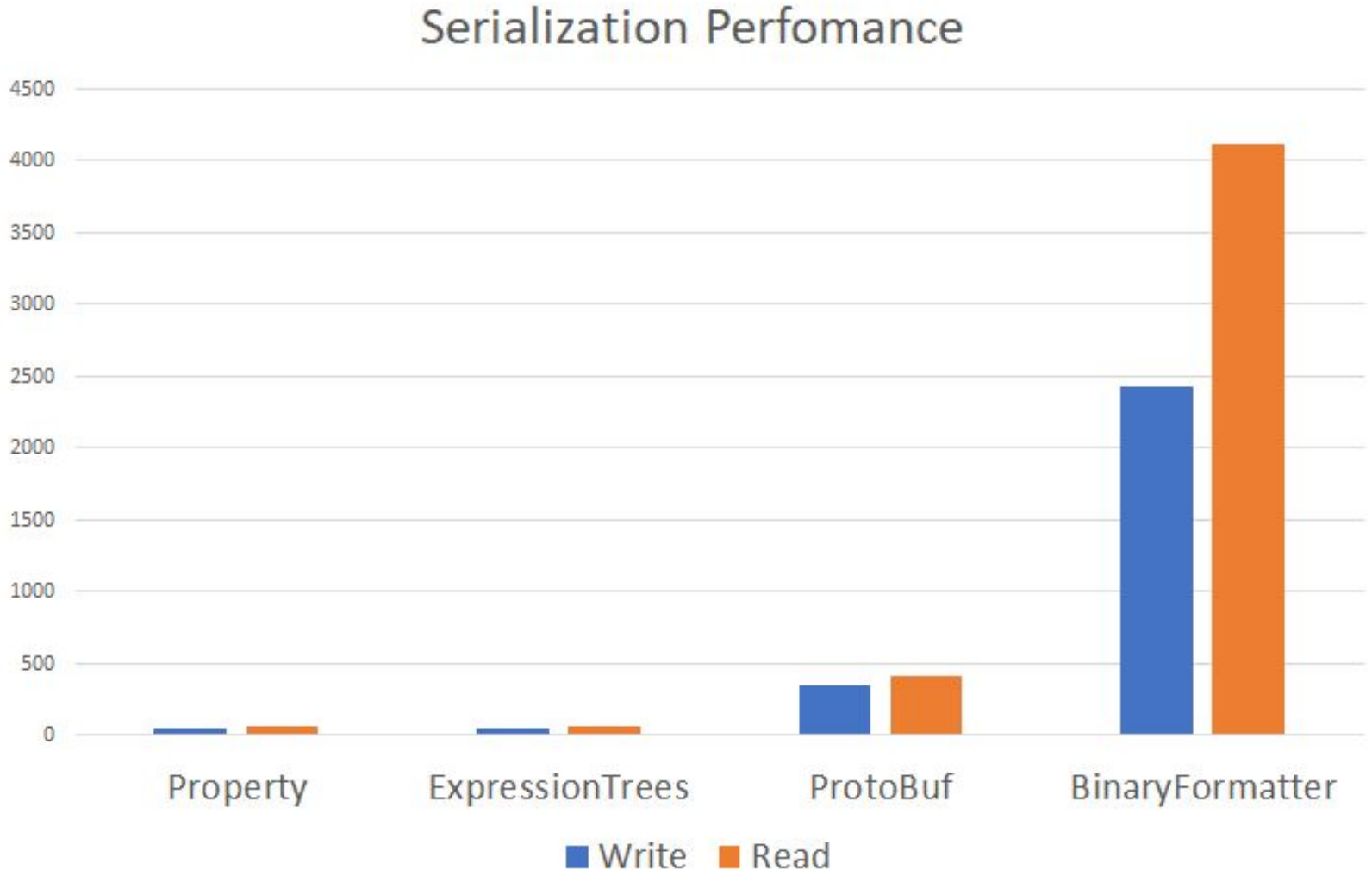
ValueType. Generated

Method	Mean	Ratio	Allocated
WriteViaProperty	12.894 ns	1.00	-
WriteViaExpressionTreesGenerated	13.575 ns	1.05	-
WriteViaILGenGenerated	13.670 ns	1.06	-
WriteViaDelegateGenerated	17.843 ns	1.38	-

```
var test = new Test
{
    IntProperty =  42,
    StringProperty = "Hello, world!"
};
```

```
var test = new Test
{
    IntProperty =  42,
    StringProperty = "Hello, world!"
};
```

Serializer	Size in bytes
Google.Protobuf	17
Custom(Property)	18
Custom(Expression Trees)	18
BinaryFormatter	236


Serialization Perfomance
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Property
ExpressionTrees
ProtoBuf
BinaryFormatter
Write
Read

Method	Categories	Mean	Ratio	Allocated
Property	Write	53.53 ns	1.00	-
ExpressionTrees	Write	55.58 ns	1.04	-
Proto	Write	352.30 ns	6.58	4168 B
BinaryFormatter	Write	2,422.71 ns	45.21	3488 B
Property	Read	63.32 ns	1.00	40 B
ExpressionTrees	Read	64.79 ns	1.02	40 B
Proto	Read	415.83 ns	6.57	4280 B
BinaryFormatter	Read	4,117.29 ns	65.02	4800 B

Method	Categories	Mean	Ratio	Allocated
Property	Write	53.53 ns	1.00	-
ExpressionTrees	Write	55.58 ns	1.04	-
Proto	Write	352.30 ns	6.58	4168 B
BinaryFormatter	Write	2,422.71 ns	45.21	3488 B
Property	Read	63.32 ns	1.00	40 B
ExpressionTrees	Read	64.79 ns	1.02	40 B
Proto	Read	415.83 ns	6.57	4280 B
BinaryFormatter	Read	4,117.29 ns	65.02	4800 B

Method	Categories	Mean	Ratio	Allocated
Property	Write	53.53 ns	1.00	-
ExpressionTrees	Write	55.58 ns	1.04	-
Proto	Write	352.30 ns	6.58	4168 B
BinaryFormatter	Write	2,422.71 ns	45.21	3488 B
Property	Read	63.32 ns	1.00	40 B
ExpressionTrees	Read	64.79 ns	1.02	40 B
Proto	Read	415.83 ns	6.57	4280 B
BinaryFormatter	Read	4,117.29 ns	65.02	4800 B

Ни рефлексией единой

- BinaryFormatter
- Newtonsoft.Json
- System.Text.Json
- Google.Protobuf
- AutoMapper

Ни рефлексией единой

- BinaryFormatter: Reflection
- Newtonsoft.Json
- System.Text.Json
- Google.Protobuf
- AutoMapper

Ни рефлексией единой

- BinaryFormatter: Reflection ¯_(ツ)_/¯
- Newtonsoft.Json
- System.Text.Json
- Google.Protobuf
- AutoMapper

Ни рефлексией единой

- BinaryFormatter: Reflection
- Newtonsoft.Json: IL Emit
- System.Text.Json
- Google.Protobuf
- AutoMapper

Ни рефлексией единой

- BinaryFormatter: Reflection
- Newtonsoft.Json: IL Emit
- System.Text.Json: IL Emit
- Google.Protobuf
- AutoMapper

Ни рефлексией единой

- BinaryFormatter: Reflection
- Newtonsoft.Json: IL Emit
- System.Text.Json: IL Emit
- Google.Protobuf: Property
- AutoMapper

Ни рефлексией единой

- BinaryFormatter: Reflection
- Newtonsoft.Json: IL Emit
- System.Text.Json: IL Emit
- Google.Protobuf: Property (Генерация C# - кода)
- AutoMapper

Ни рефлексией единой

- BinaryFormatter: Reflection
- Newtonsoft.Json: IL Emit
- System.Text.Json: IL Emit
- Google.Protobuf: Property (Генерация C# - кода)
- AutoMapper: Expression Trees

Осталось за скобками

- Безопасность
- Инстанцирование объектов (new, CreateInstance)
- Парсинг данных
- Схема и версионирование данных
- Генерация C# - кода

Ни рефлексией единой

- Reflection - медленно
- Есть достойные альтернативы:
  - CreateDelegate / IL Emit / Expression Trees
- Доступ к данным еще не всё
- Runtime генерация кода для “сложных” случаев


ГЕНЕРИРОВАТЬ
IL В RUNTIME
REFLECTION
imgflip.com

Константин Рудниченко
технический директор Cadwise

rudnichenko.k@gmail.com

https://t.me/krudnichenko

https://twitter.com/KRudnichenko

http://cadwise.ru