UFADEVCONF

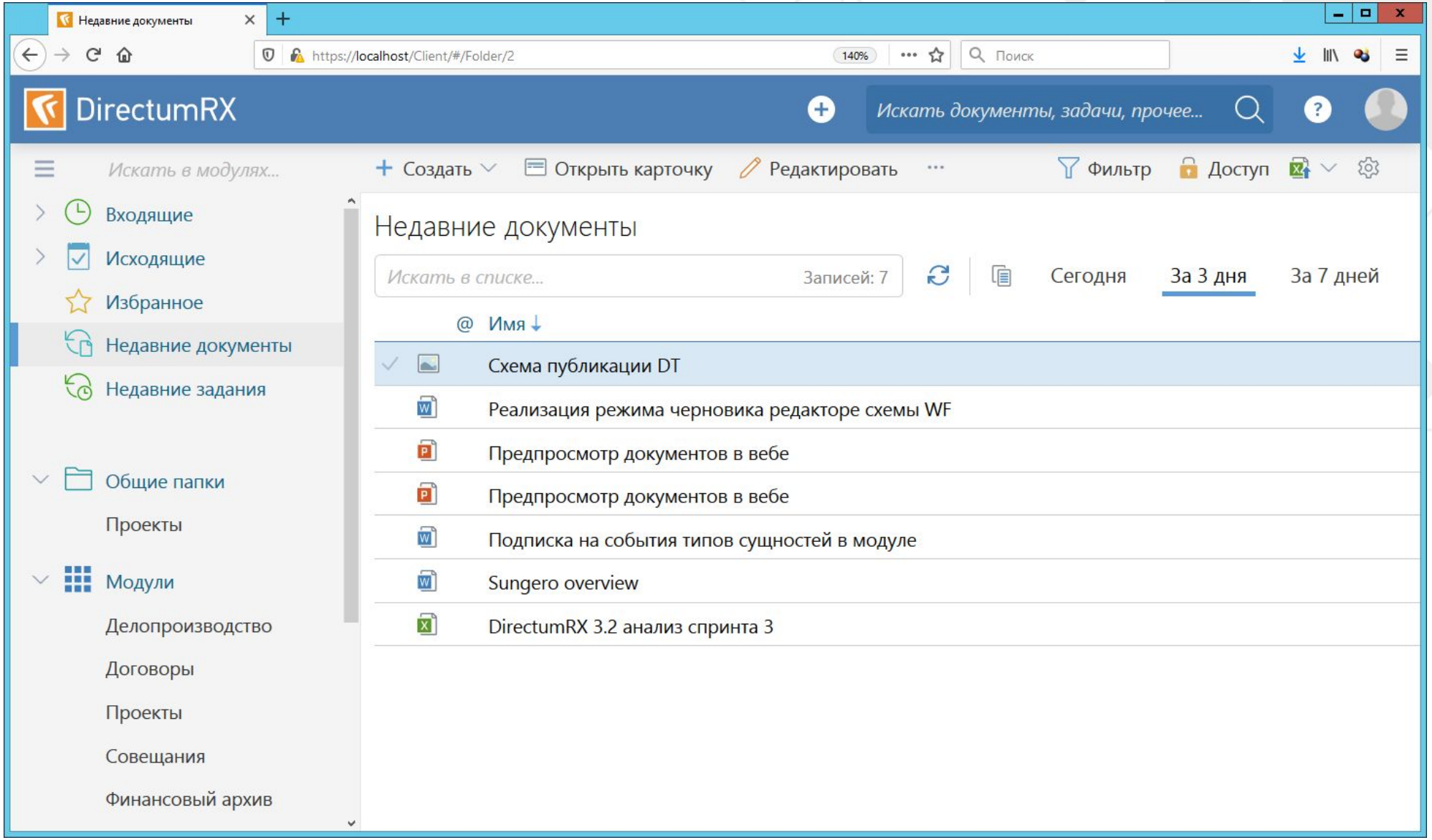
.NET code hot reload

Асадуллин Тимур
Directum, Уфа

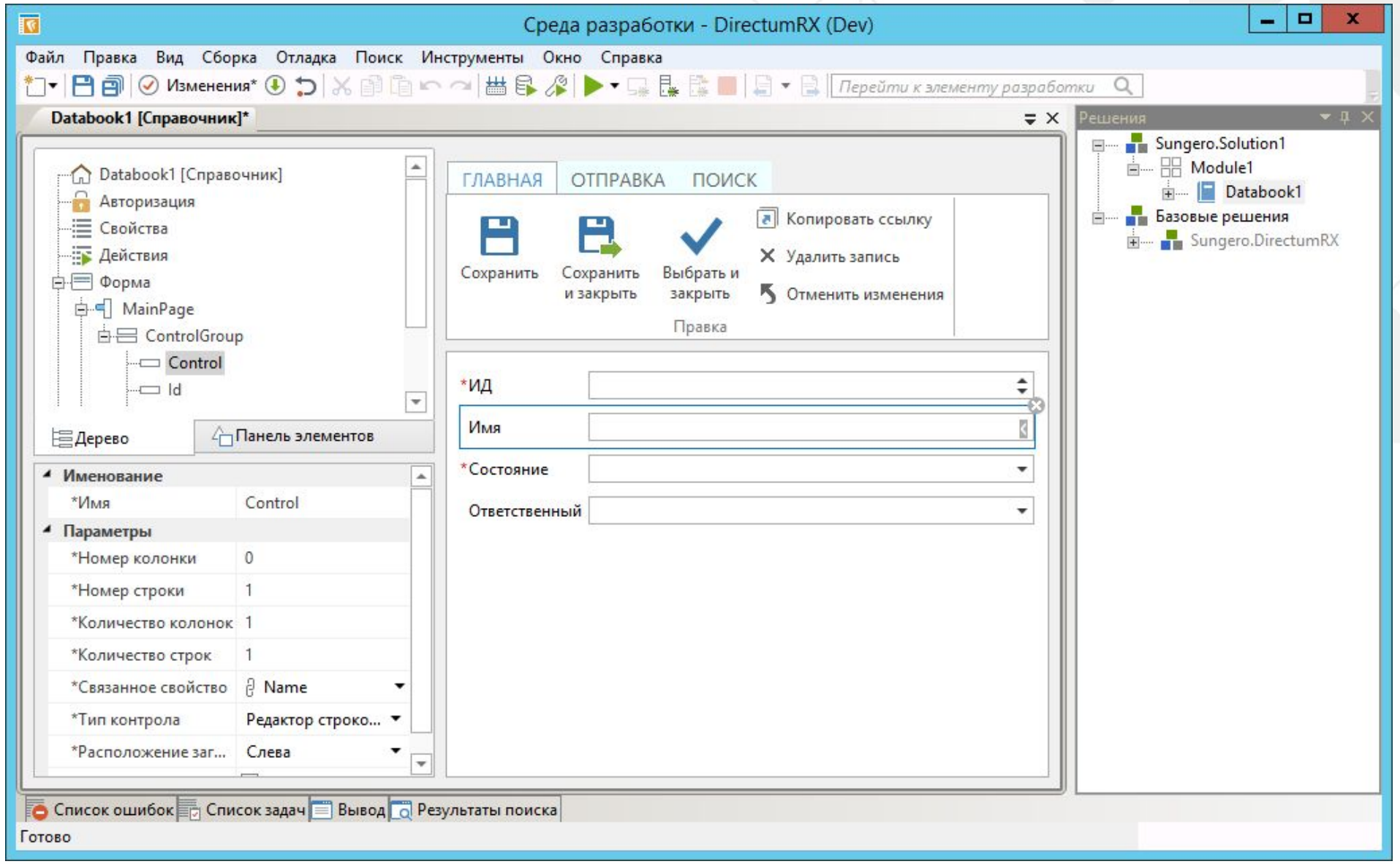
Что за «Code hot reload»?

- Обновление кода приложения без перезапуска этого приложения.
- .NET Framework (4.6 +)
- Надеюсь такие слова вас не пугают:
  CLR, MSIL, Assembly, указатель.

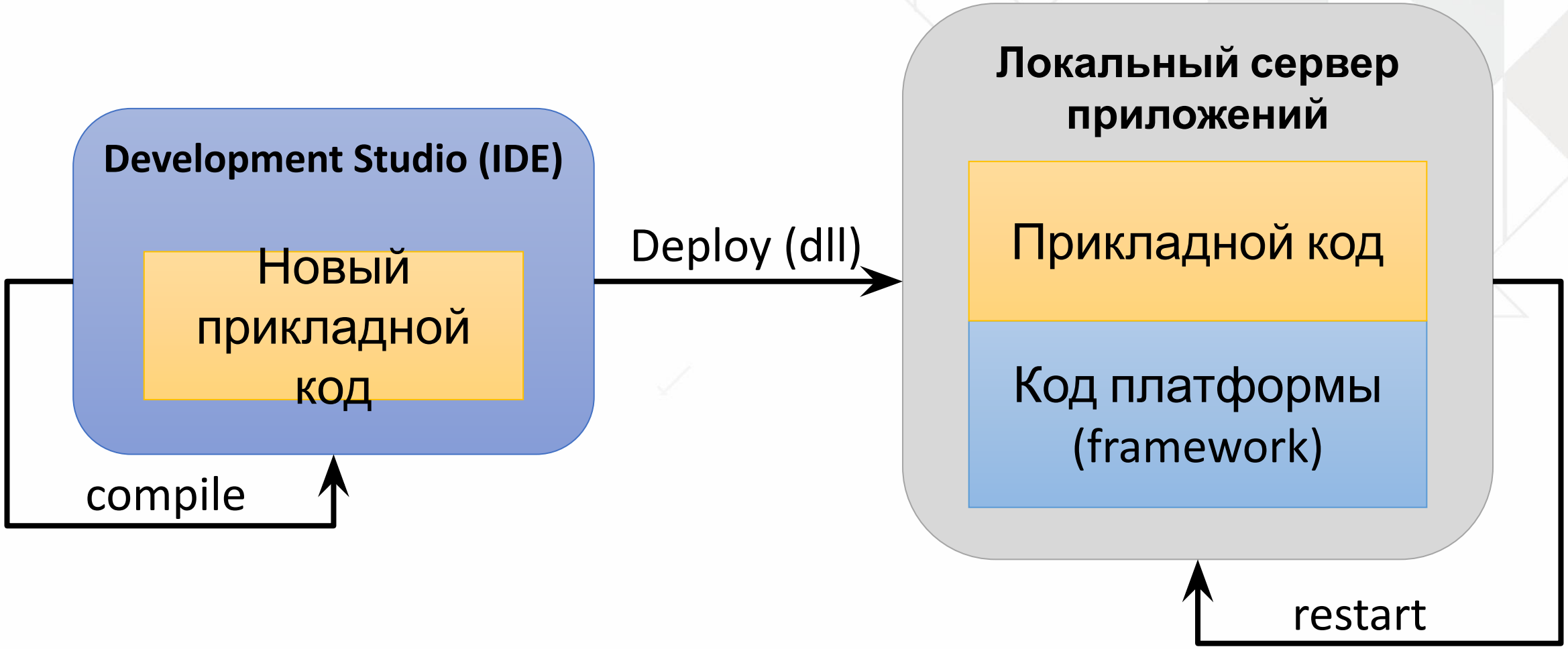
DirectumRX – ECM-система



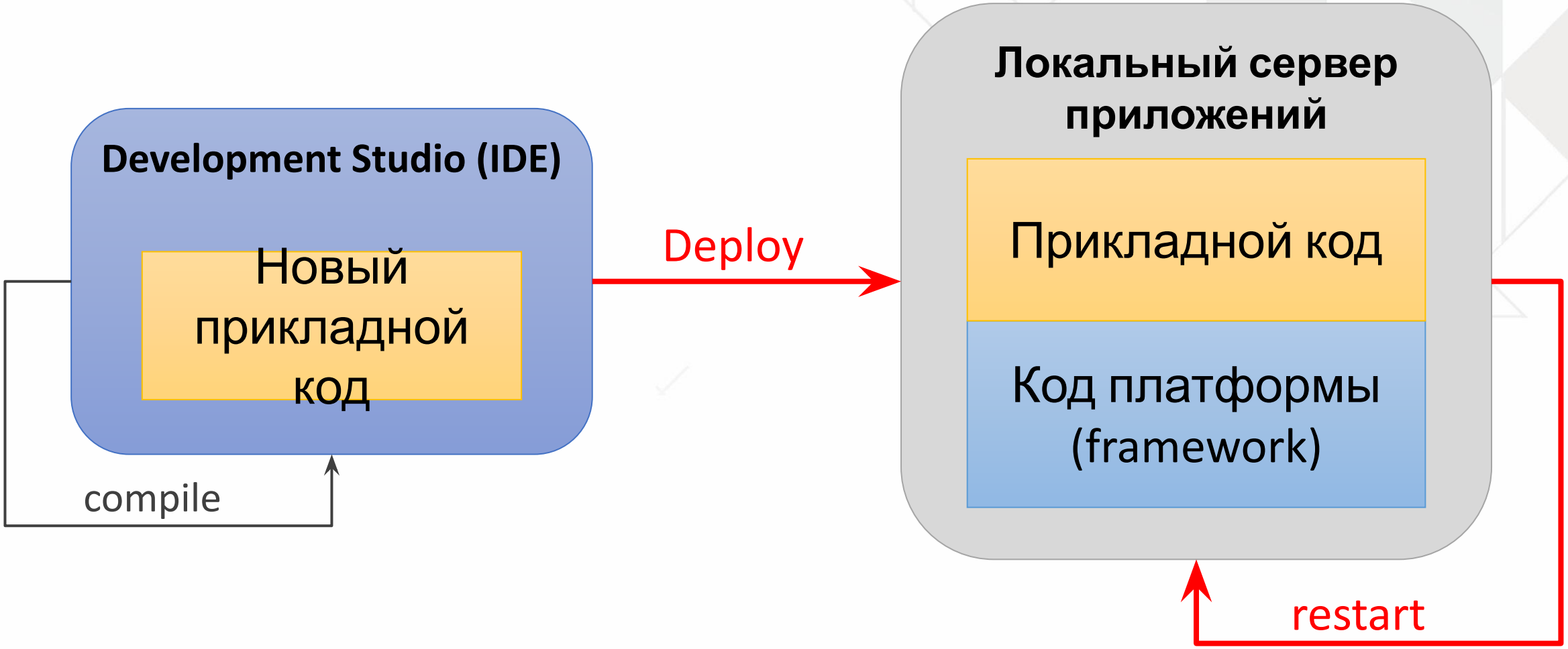
DirectumRX Development Studio



Прикладная разработка DirectumRX



Зачем нам HotReload

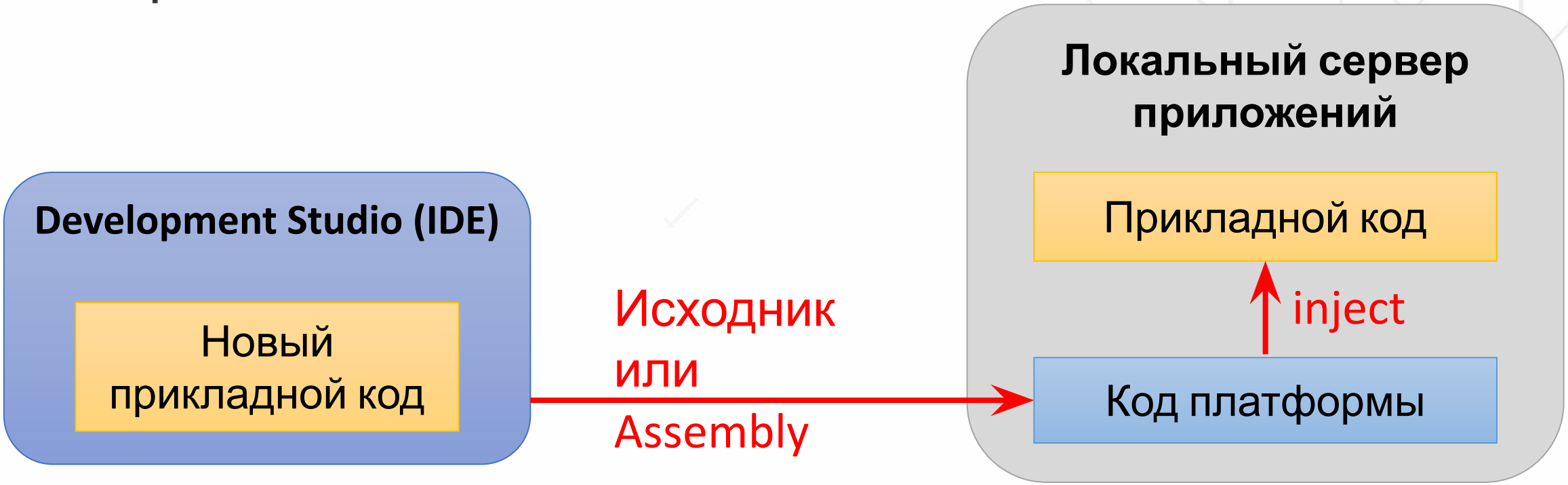


Дополнительные ограничения и пожелания

- Если поменялась структура БД, одним HotReload не обойтись.
- HotReload и отладка в Dev Studio (breakpoints, watch).
- Подходит и для серверного, и для клиентского кода
(у нас есть Desktop-клиент).

Основная идея

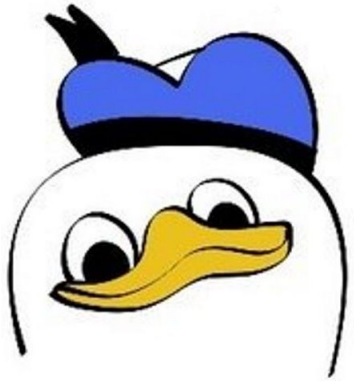
- Избавиться от перезапуска сервера.
- Встраивать код напрямую в работающее приложение.



Решения в лоб

- Reflection – LoadAssembly
  (Shadow Copy Assemblies)
- Managed Extensibility Framework (MEF, VS-MEF)
- Mono.Cecil

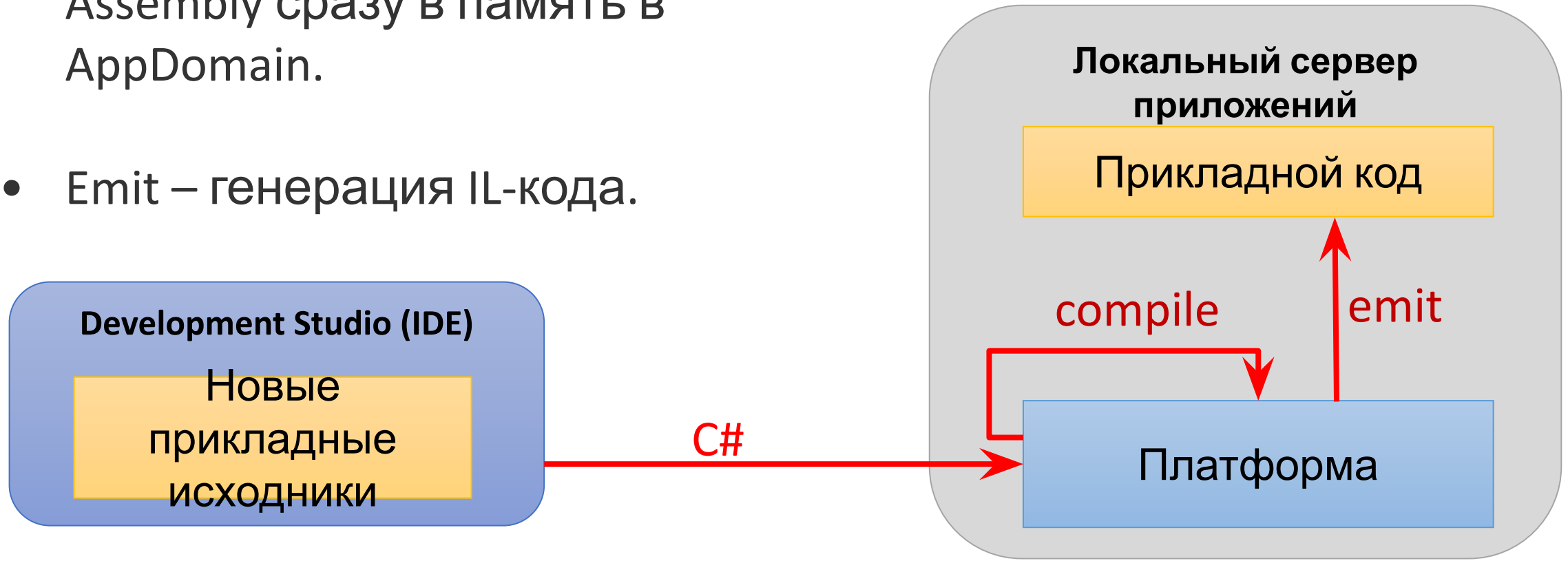
Решения в лоб. Проблемы



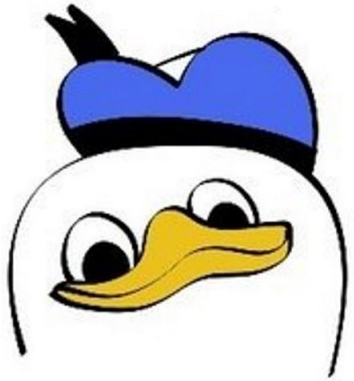
- Не повлияем на уже существующие в памяти объекты.
- Новые объекты, создаваемые через **new** будут ссылаться на старые типы.
- В случае с MEF прикладной код придётся переписать особым образом (атрибуты, интерфейсы).

Emit

- CodeDom – компиляция C# в Assembly сразу в память в AppDomain.
- Emit – генерация IL-кода.



Emit. Проблемы



- Emit работает только с DynamicAssembly.
  А при компиляции CodeDom’а мы получаем обычную Assembly.

- Чтобы «завести» динамический класс,
  нужно вызвать у него CreateType().
  Это блокирует его дальнейшие модификации.

Edit and Continue

- Встроенный в Visual Studio хитрый механизм, генерирующий некоторые дельты.
- Общедоступного API нет.
- Даже в самой VS механизм не работает в ряде случаев.

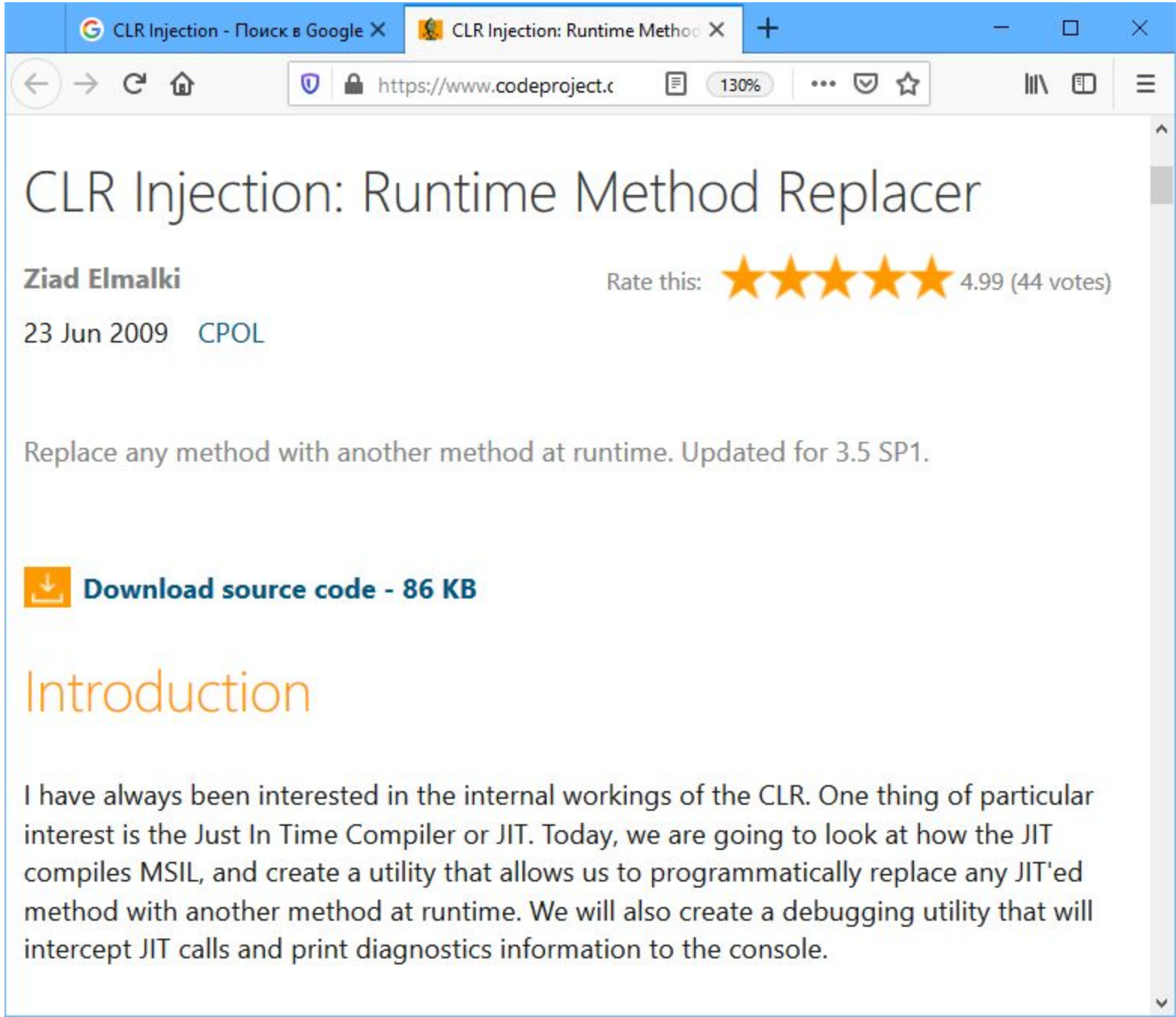

Google


stack
overflow


CODE
PROJECT
For those who code


GitHub

Method Inject v1



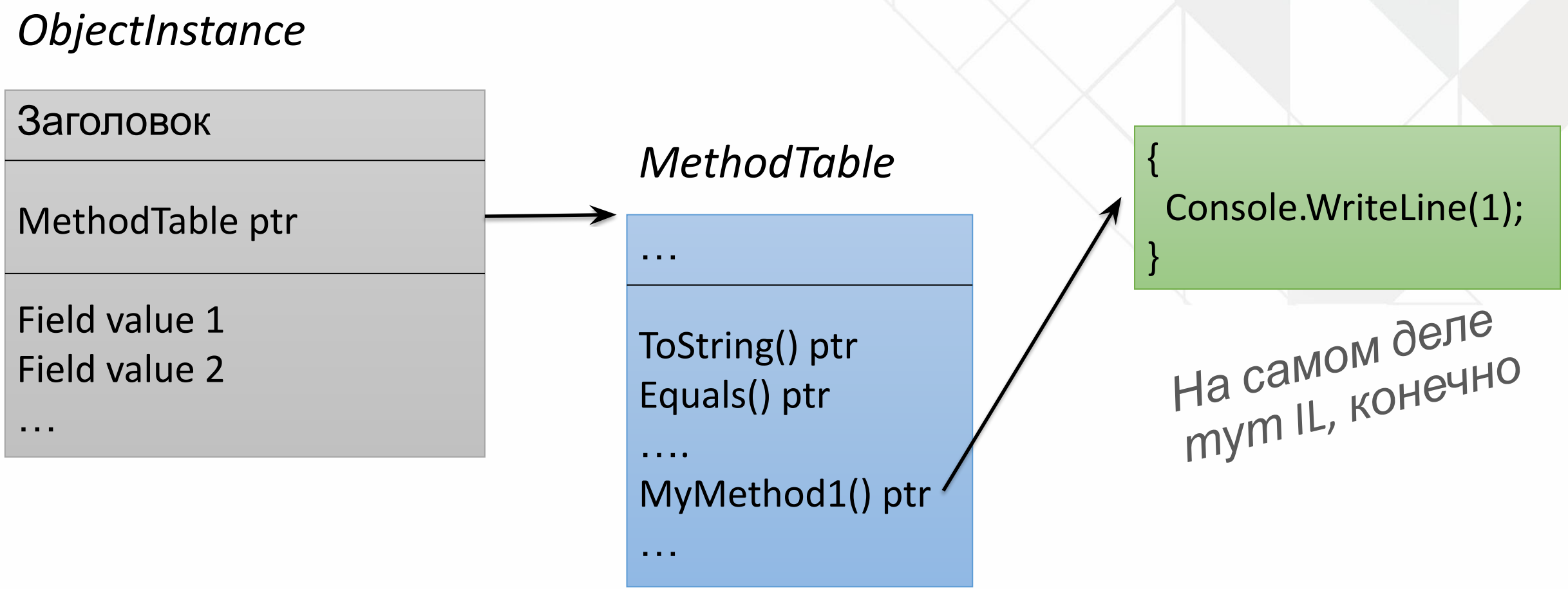
Method Inject v1

Суть – замена указателя на метод.

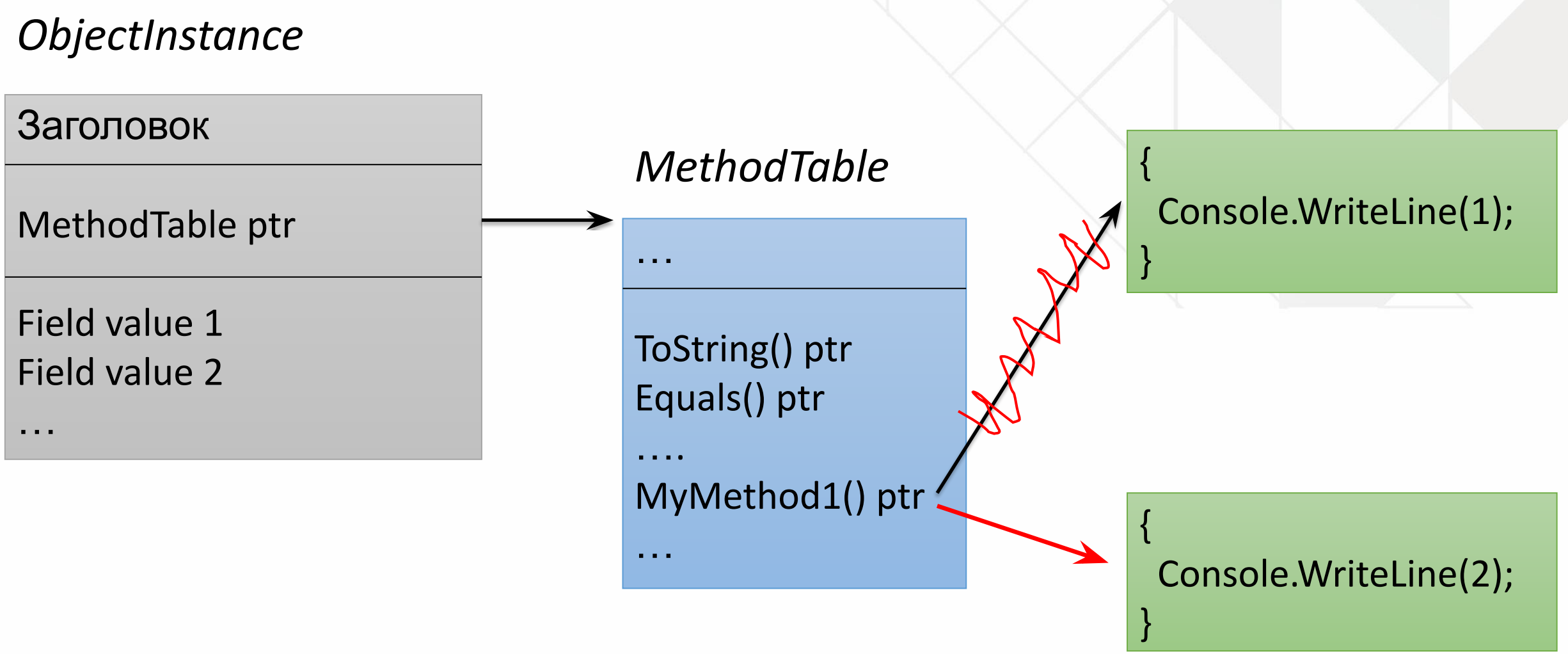
```
MethodInfo methodToReplace = ... ;
MethodInfo methodToInject = ... ;


unsafe
{
    long* target = (long*) methodToReplace.MethodHandle.Value.ToPointer();
    long* inject = (long*) methodToInject.MethodHandle.Value.ToPointer();
    *target = *inject;
}
```

В реальности чуток сложнее, потому что надо учесть x86/x64, Debug/Release.

Как хранятся описания классов в .NET

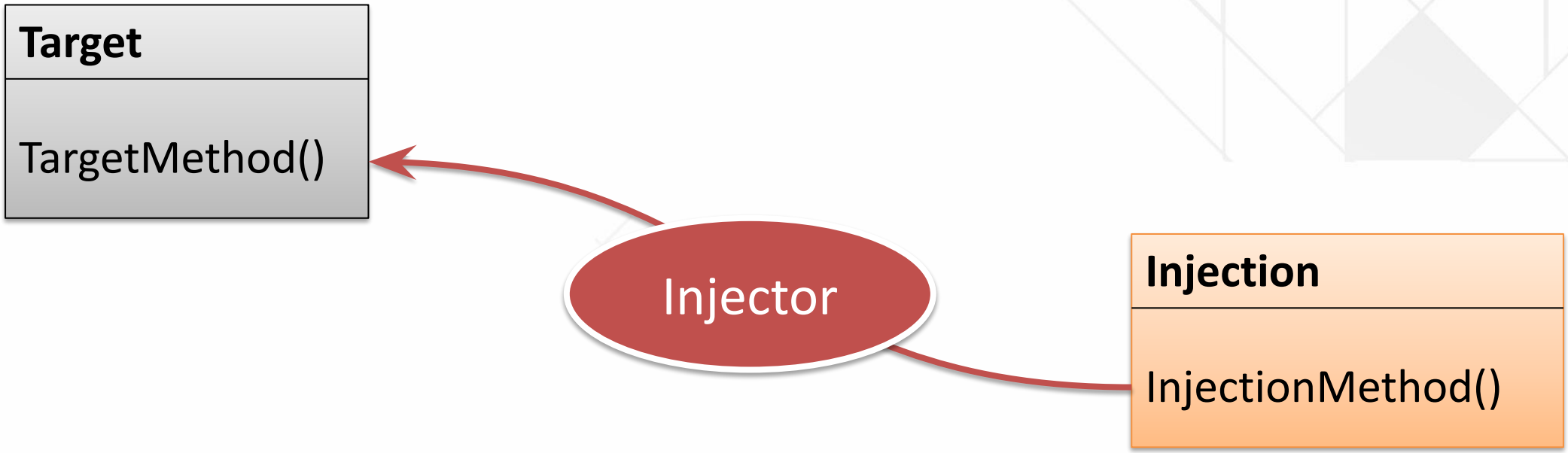


Method Inject v1. Суть



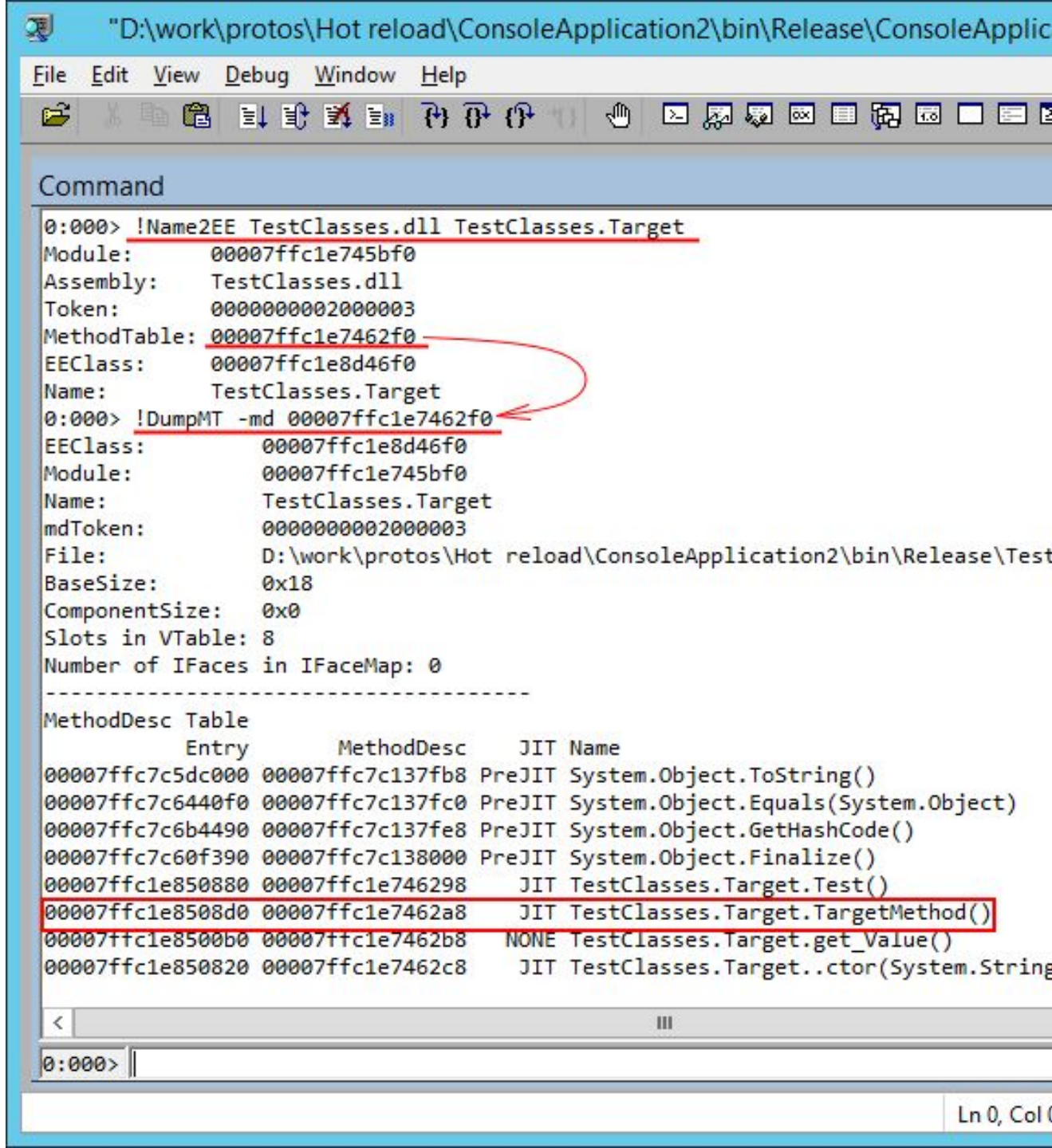
Method Inject v1. Проверяем на практике

- Есть два класса…



Method Inject v1. Проверяем на практике

WinDbg + SOS →



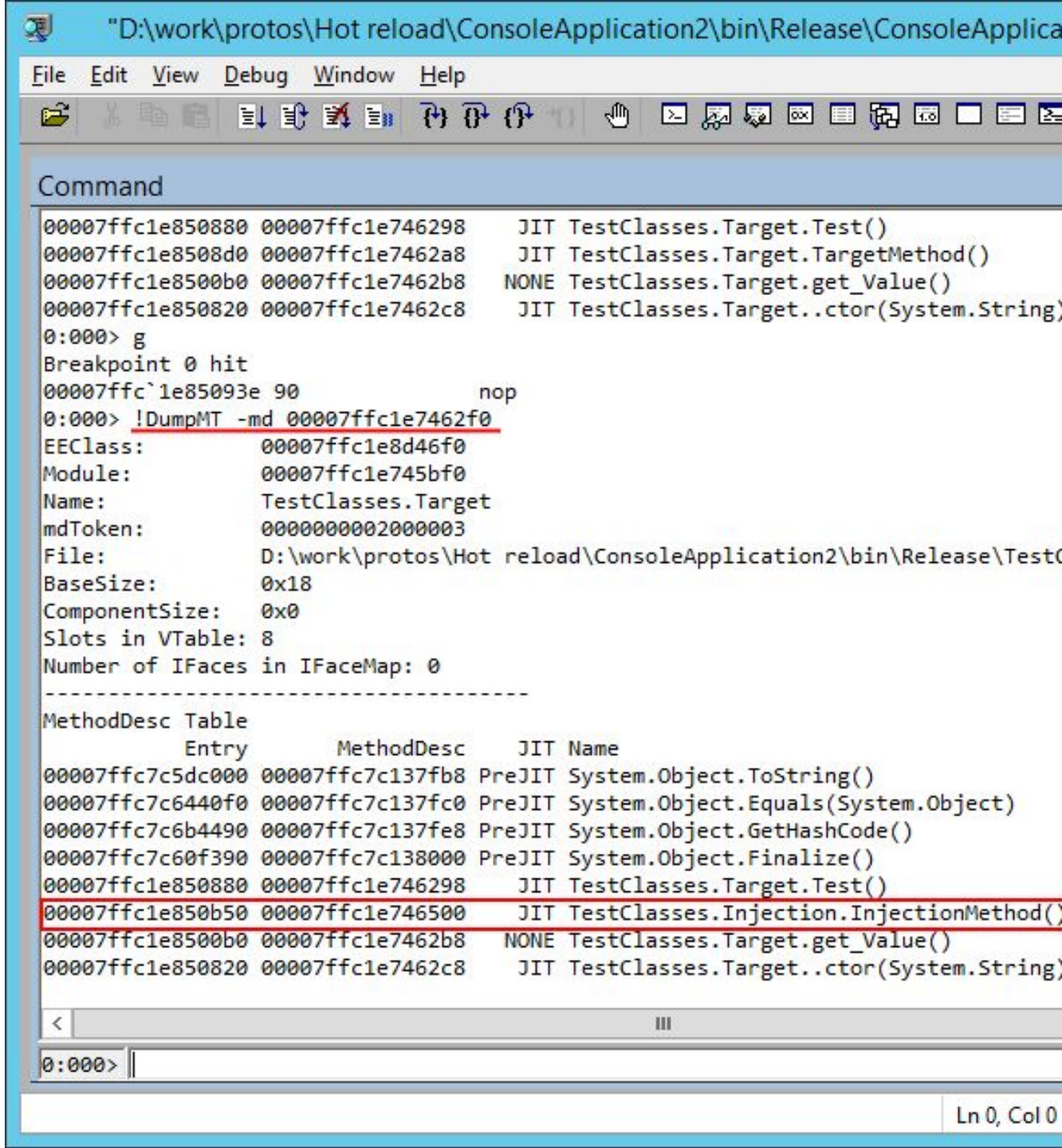
Исходный MethodTable

TestClasses.Target.Test()
TestClasses.Target.TargetMethod()
TestClasses.Target.get_Value()
TestClasses.Target..ctor (System.String)

Method Inject v1. Проверяем на практике

MethodTable после Inject

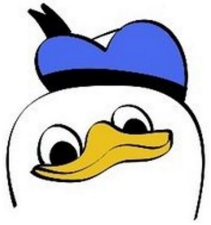
TestClasses.Target.Test()
TestClasses.Injection.InjectionMethod()
TestClasses.Target.get_Value()
TestClasses.Target..ctor (System.String)



Method Inject v1. Дополнительные работы

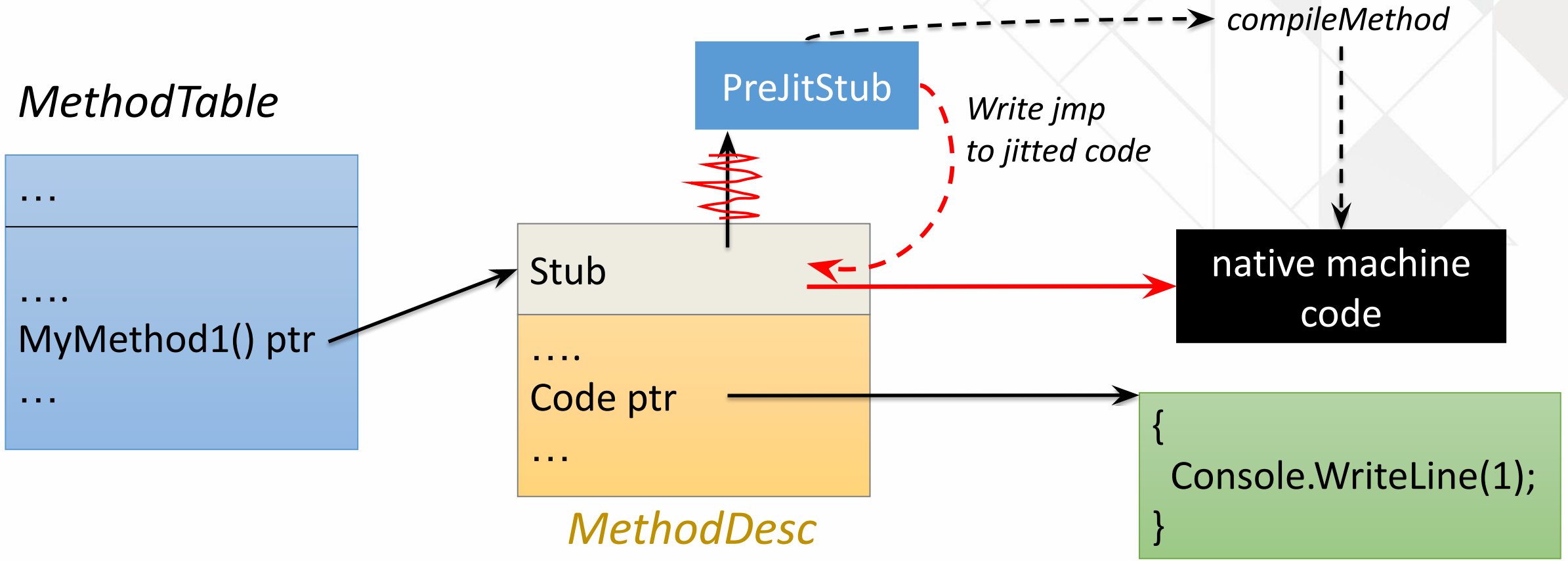
- Сравнить прикладные исходники и найти изменившиеся методы.
- Передать исходники методов на сервер.
- Создать новый класс, засунуть в него методы и скомпилировать в память.
- Найти старый метод и сделать MethodInject на новый.

Method Inject v1. ПРОБЛЕМА!

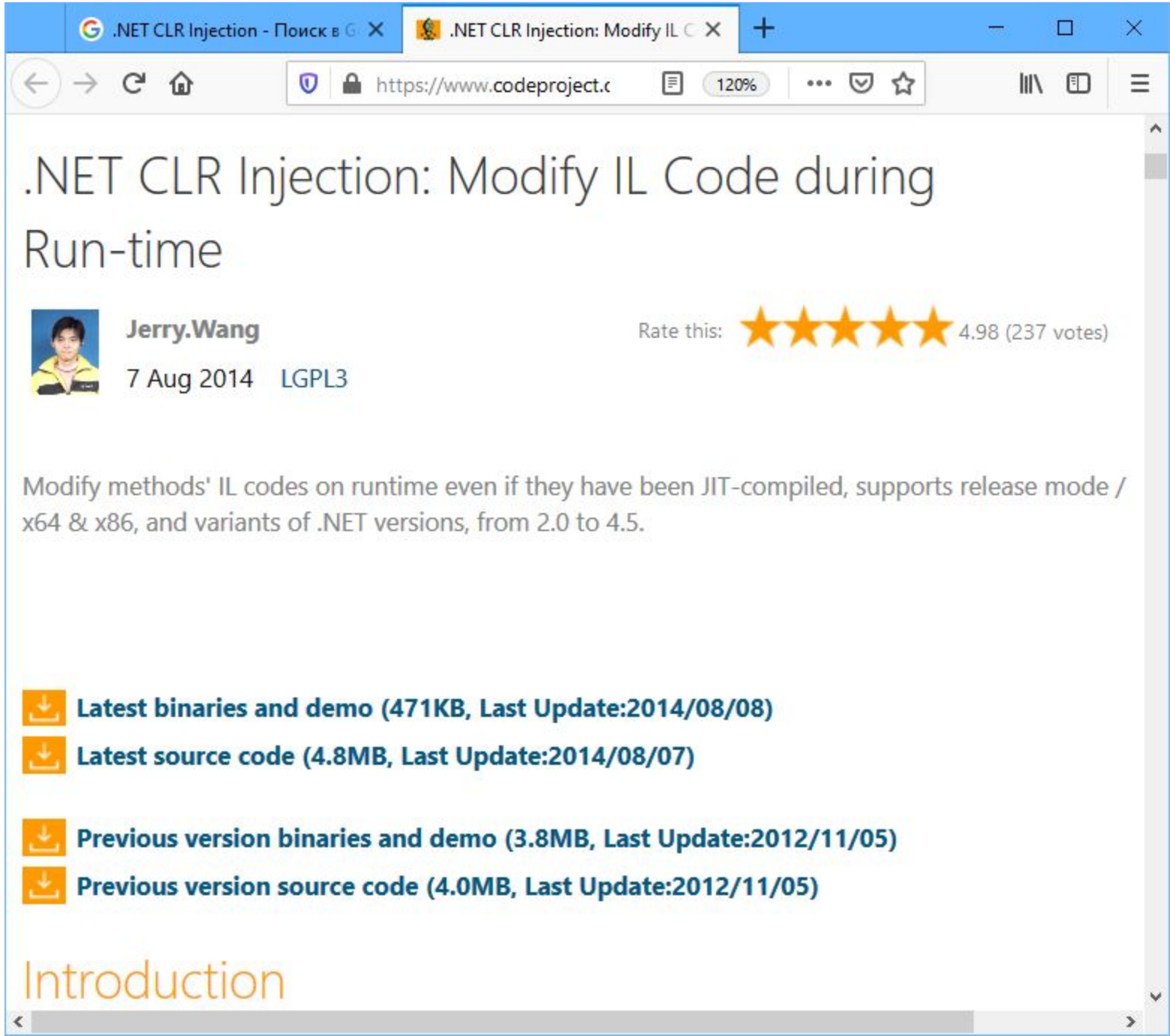


Прекрасно работает
пока мы не вызвали подменяемый метод.

Method Inject v1. ПРОБЛЕМА!



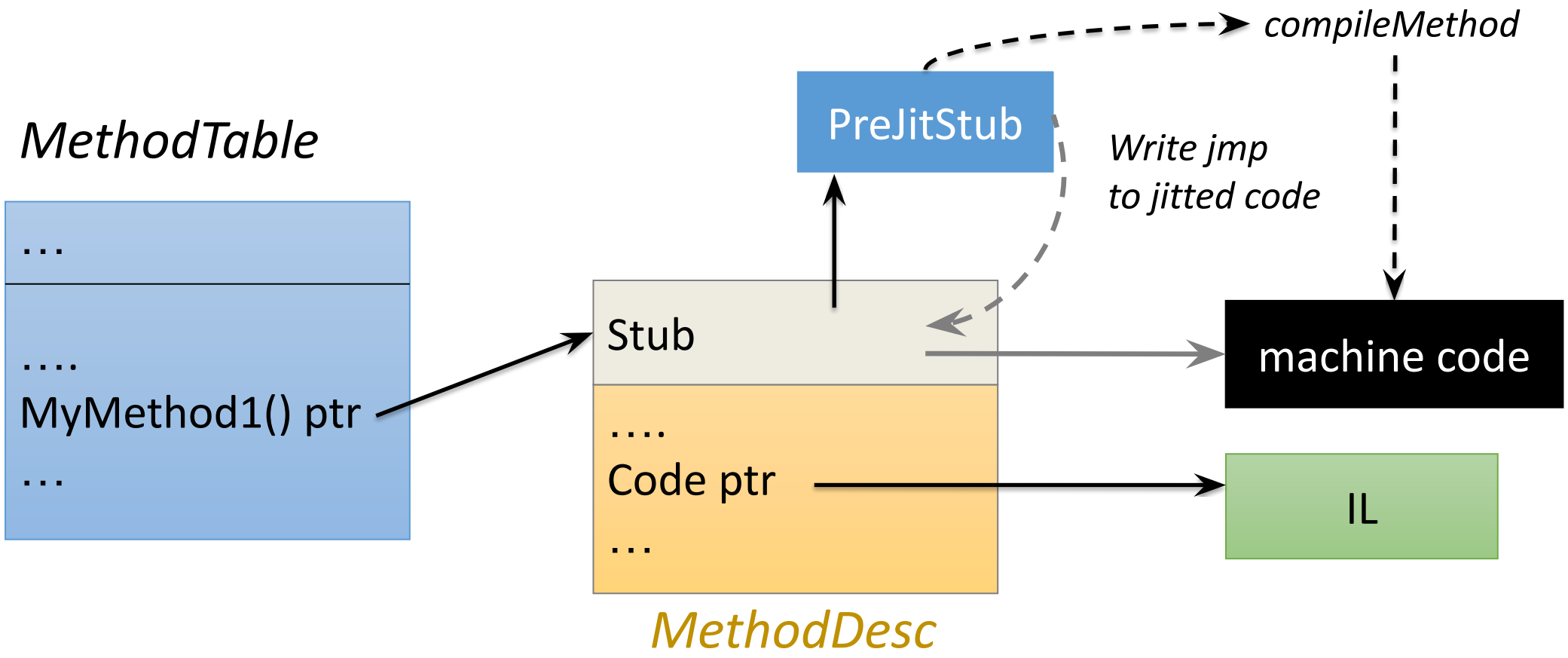
Method Inject v2



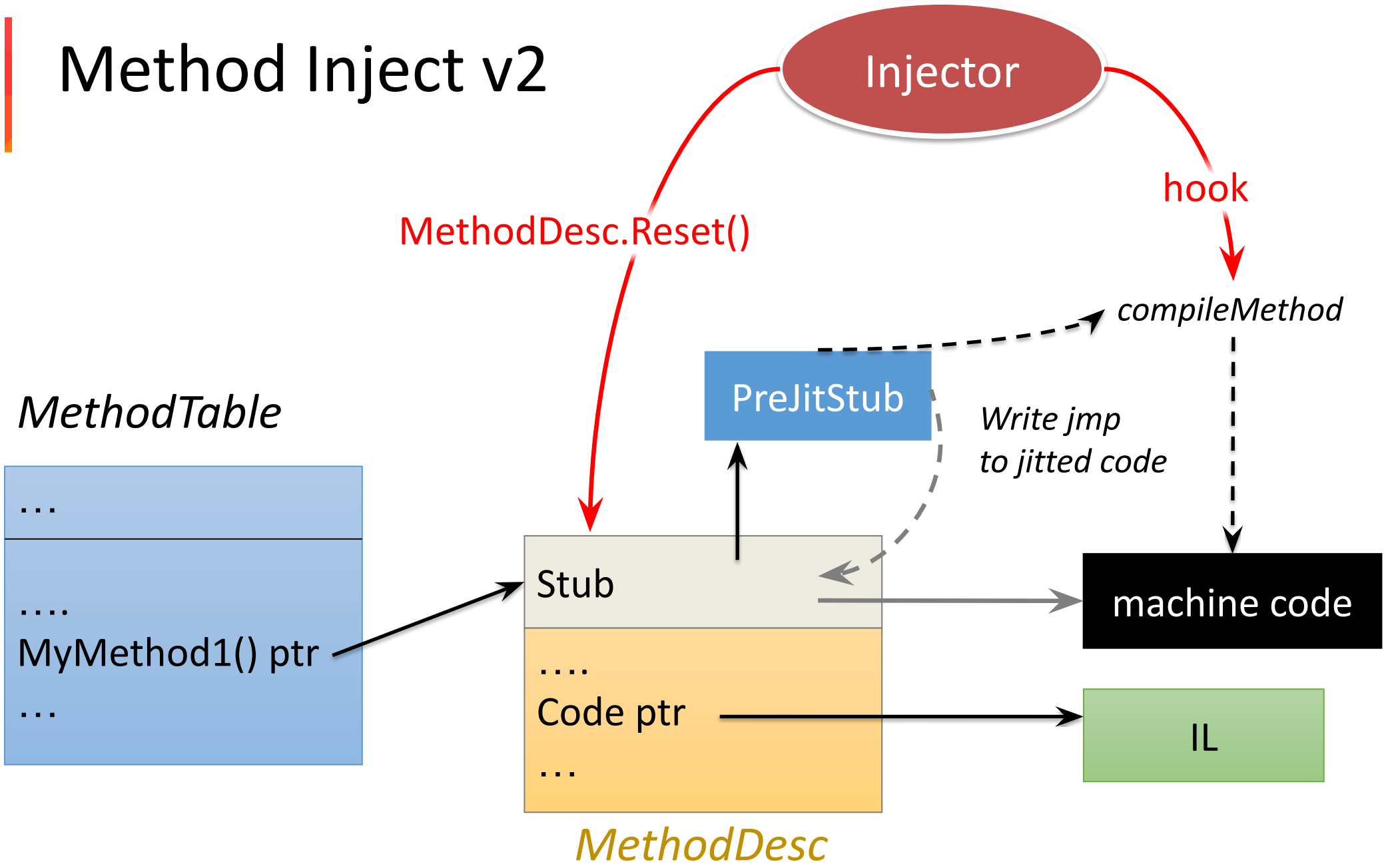
Method Inject v2

```
public static class InjectionHelper
{
  ...

  public static void Initialize()

  public static void UpdateILCodes(MethodInfo method, byte[] ilCodes)

  ...
}
```

Method Inject v2



Method Inject v2



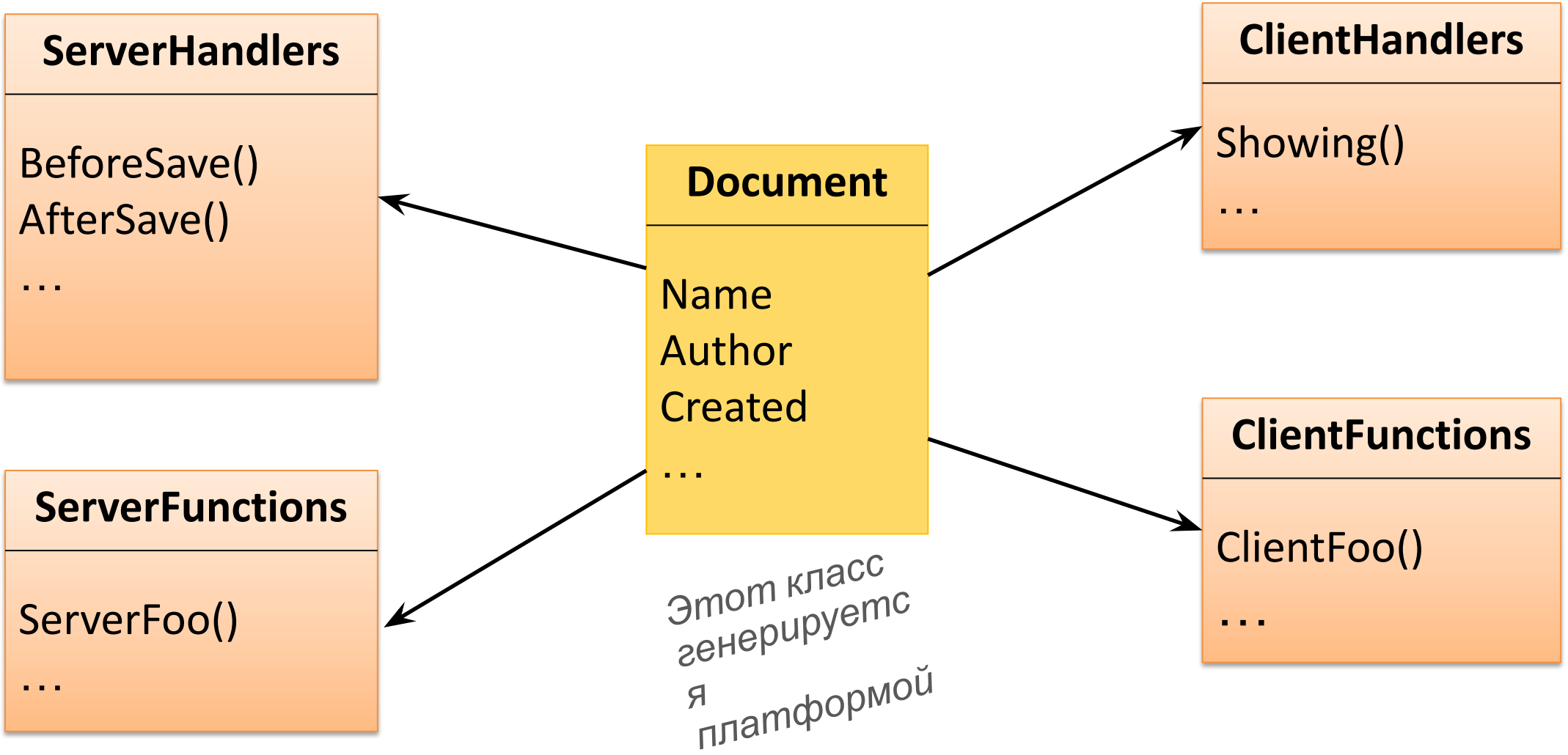
Method Inject v2. Проблемки

- Нормально заработало только на .NET Framework 2.0 - 4.0
- Inline маленьких методов.
- Лезем внутрь CLR.
  Слишком рискованно для продакшена.

Давайте вспомним, с чего мы начали

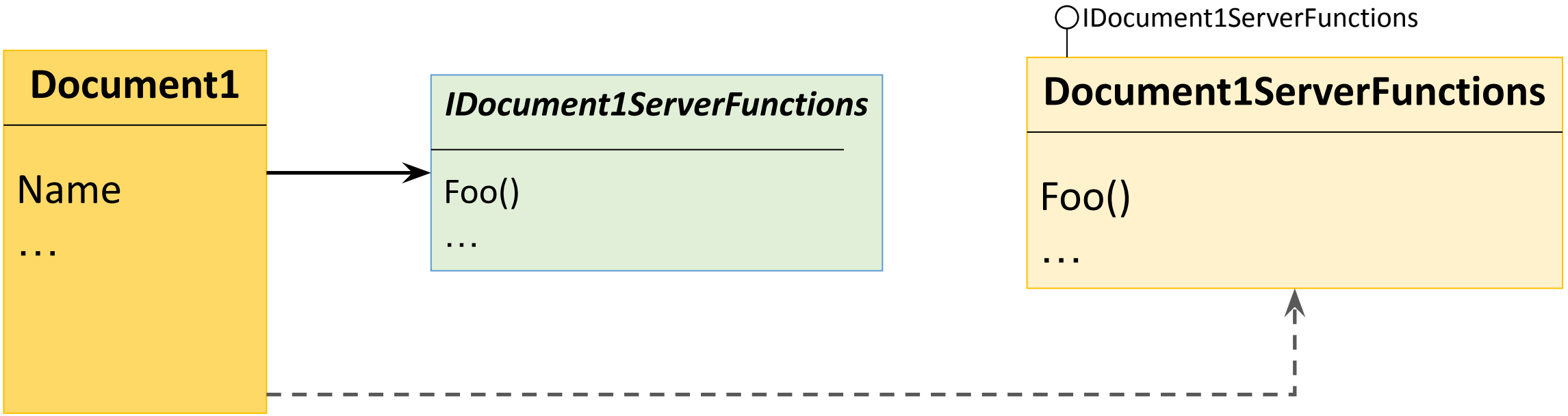
- Есть платформа.
- Она исполняет прикладной код.
- Мы хотим подменять прикладной код.

Что за прикладной код?

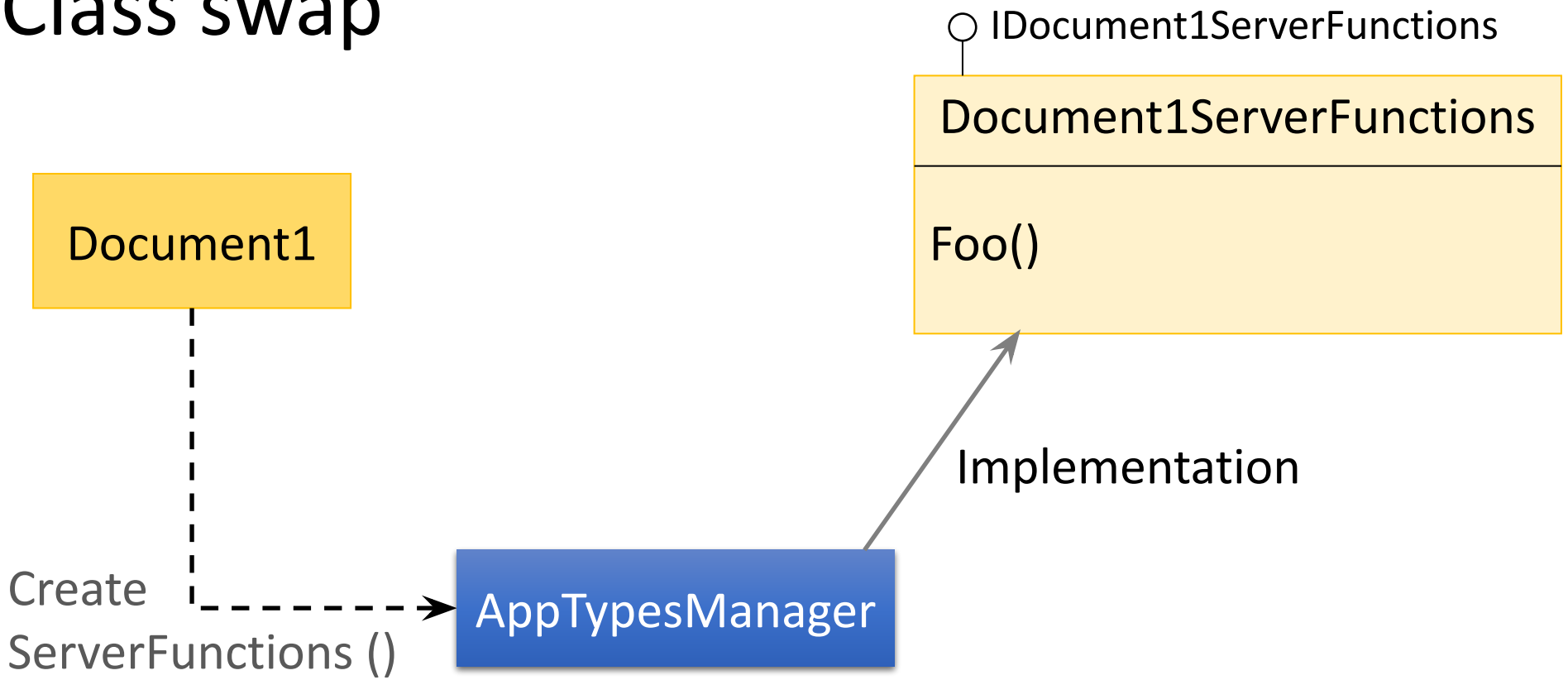


Class swap

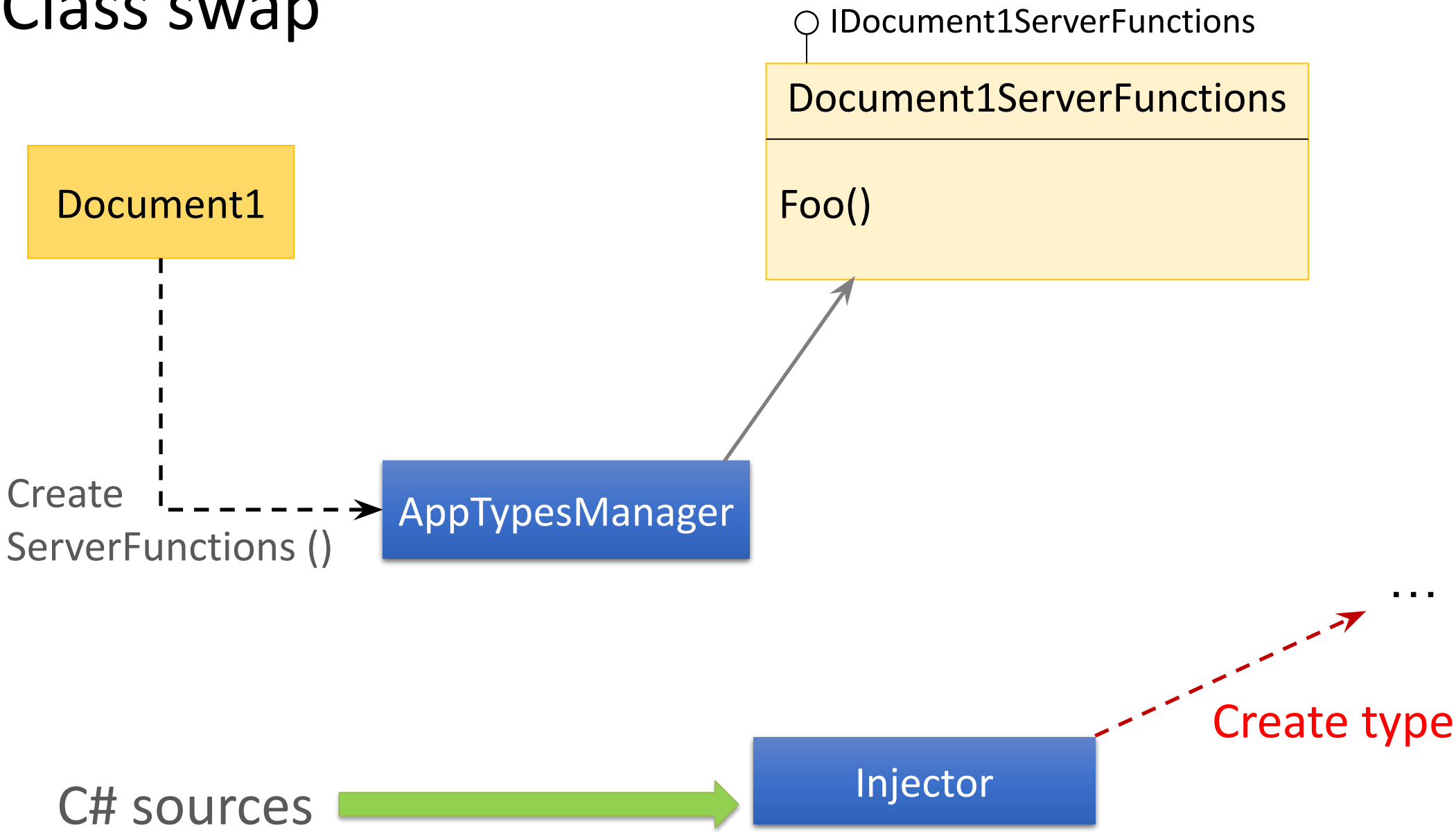
- Платформенный код создаёт прикладные объекты (Dependency Injection).
- Не возимся с методами – заменяем сразу весь объект.



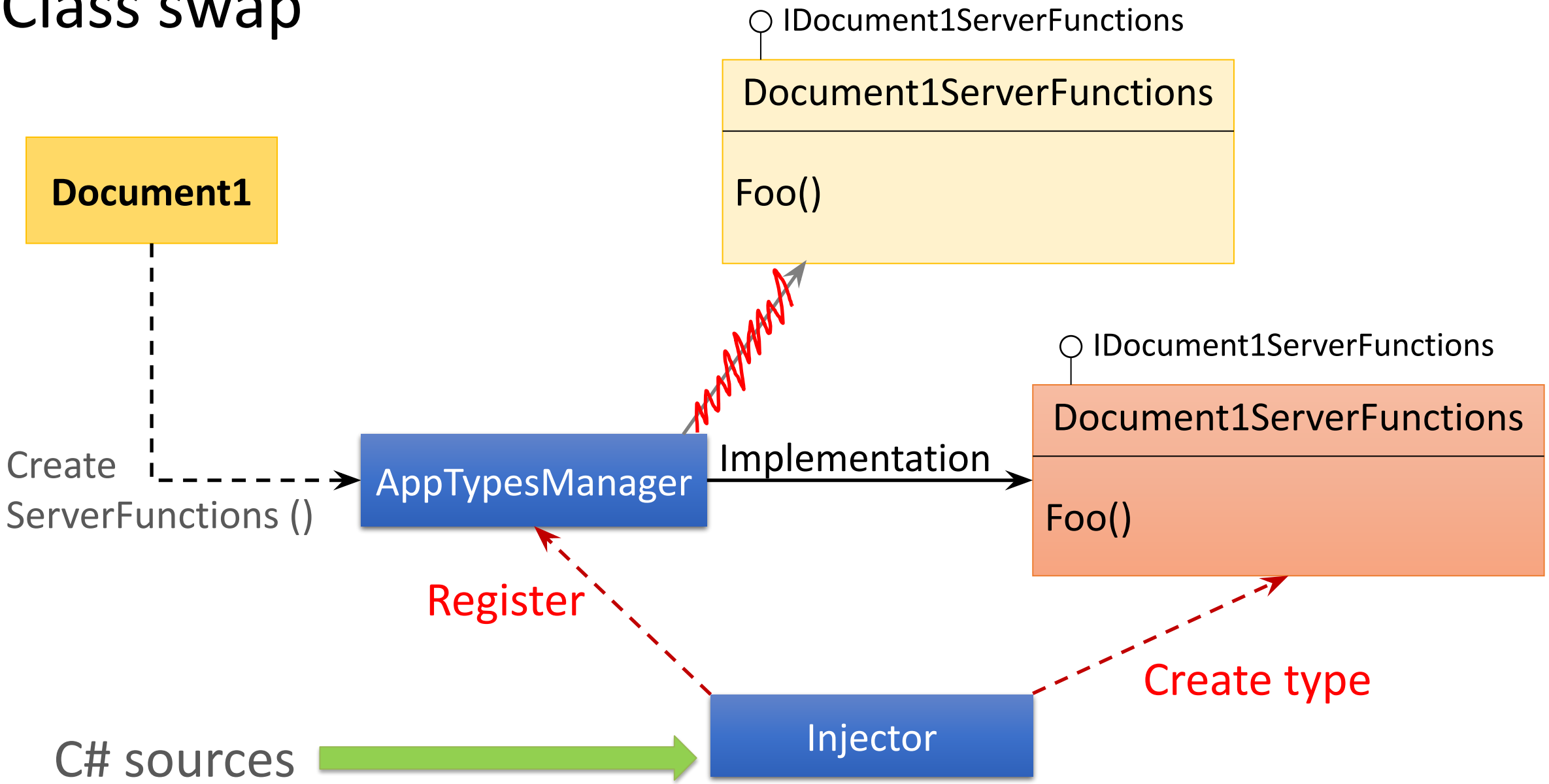
Class swap


IDocument1ServerFunctions
Document1ServerFunctions
Foo()
Document1
Implementation
Create
ServerFunctions ()
AppTypesManager

Class swap


IDocument1ServerFunctions
Document1ServerFunctions
Foo()
Document1
Create
ServerFunctions ()
AppTypesManager
...
Create type
C# sources
Injector

Class swap


IDocument1ServerFunctions
Document1ServerFunctions
Foo()
Document1
IDocument1ServerFunctions
Document1ServerFunctions
Foo()
Create
ServerFunctions ()
AppTypesManager
Implementation
Register
Create type
Injector
C# sources

Заключение

- В продакшен это не пошло :(
  Деплой мы ускорили иначе – оптимизации.
- Исследования – круто.
- Погружение в детали инструмента – интересно.

()

- .net core
- JIT disable
- Custom .NET CLR

THE END

Асадуллин Тимур
Directum, Уфа
timonik@bk.ru

Ссылочки

- MethodInject v1
  https://www.codeproject.com/Articles/37549/CLR-Injection-Runtime-Method-Replacer
- MethodInject v2
  https://www.codeproject.com/Articles/463508/NET-CLR-Injection-Modify-IL-Code-during-Run-time
- EasyHook
  https://easyhook.github.io
- Отладка WinDbg + SOS
  https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugging-managed-code
- .net core
  «How to use and debug assembly unloadability in .NET Core»
  https://docs.microsoft.com/ru-ru/dotnet/standard/assembly/unloadability