


Source Generators v2.0

инкрементальные генераторы

Andrey Dyatlov
Software engineer at JetBrains
Working on ReSharper / Rider

 @a_tessenr

 TessenR



Что такое генераторы?

- Ваш тип является частью процесса компиляции
 - Полный доступ к модели кода
 - Возможность добавить новый код в проект



Предыдущий доклад о генераторах
bit.ly/3AjApHD

Что такое генераторы?

- Ваш тип является частью процесса компиляции
 - Полный доступ к модели кода
 - Возможность добавить новый код в проект
- **Сценарии использования**
 - Автоматизация создания шаблонного кода
 - `INotifyPropertyChanged`, `IEquatable`, `ToString`...
 - Оптимизация рефлексии в рантайме
 - Сериализация, контейнеры зависимостей...



Предыдущий доклад о генераторах
bit.ly/3AjApHD

Что такое генераторы?

- Ваш тип является частью процесса компиляции
 - Полный доступ к модели кода
 - Возможность добавить новый код в проект
- Сценарии использования
 - Автоматизация создания шаблонного кода
 - `INotifyPropertyChanged`, `IEquatable`, `ToString`...
 - Оптимизация рефлексии в рантайме
 - Сериализация, контейнеры зависимостей...
- Интеграция с IDE
 - Нет необходимости перекомпиляции проекта \ запуска скриптов



Предыдущий доклад о генераторах
bit.ly/3AjApHD

В докладе

- Влияние генераторов на работу IDE
- Что такое инкрементальные генераторы и зачем они нужны?
- Как разработчик генератора может облегчить жизнь пользователям?
- Как обновить генератор на новый интерфейс?
- В чем отличия старых и новых генераторов?
- Кому в первую очередь стоит мигрировать свой генератор?

Генераторы в IDE

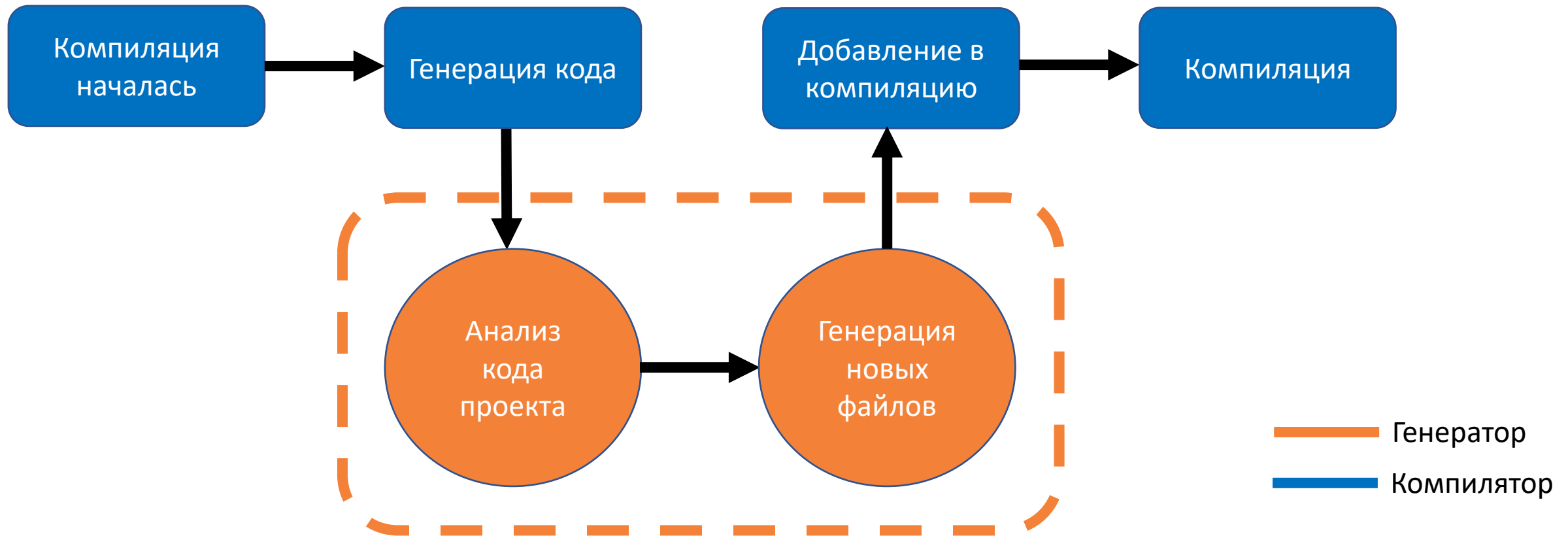
- Основная область применения – большие проекты
- Результат работы мгновенно доступен в IDE
 - Добавили атрибут – в типе появился новый элемент
 - Ничего не нужно делать чтобы исчезли ошибки компиляции
 - Мгновенно доступен в автодополнении

Генераторы в IDE

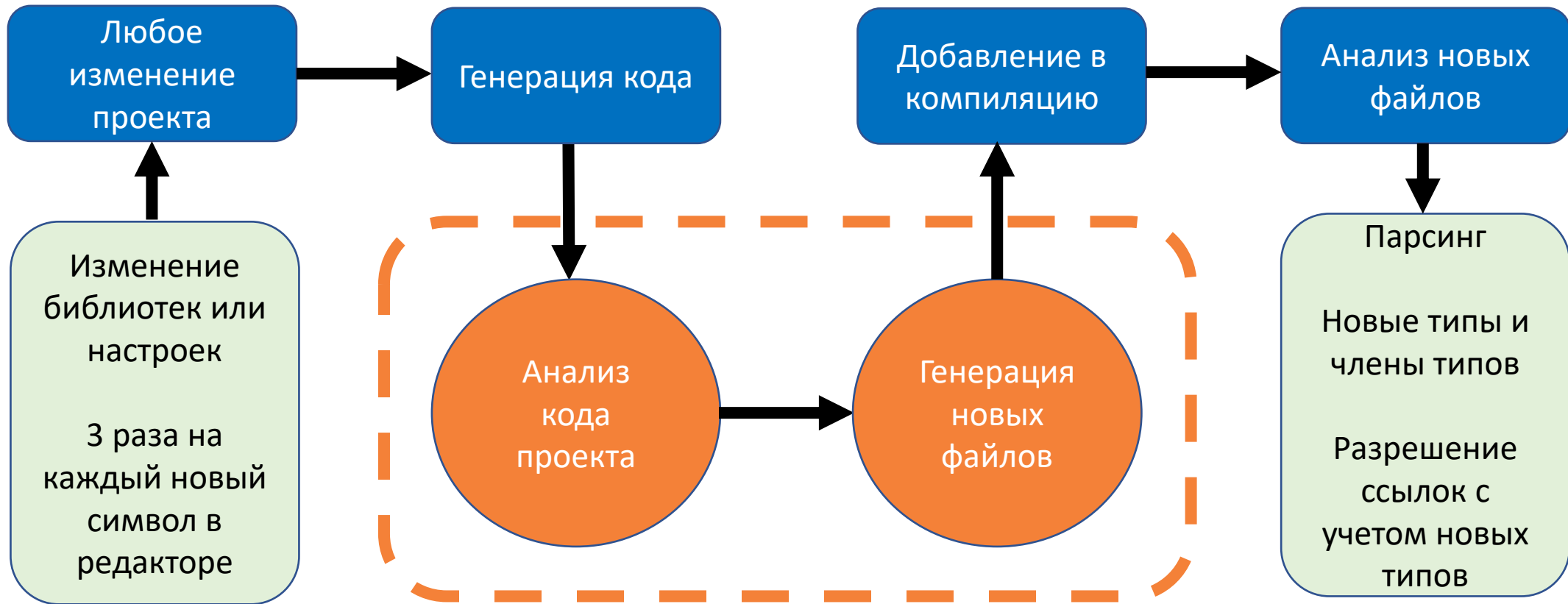
- Основная область применения – большие проекты
- Результат работы мгновенно доступен в IDE
 - Добавили атрибут – в типе появился новый элемент
 - Ничего не нужно делать чтобы исчезли ошибки компиляции
 - Мгновенно доступен в автодополнении
- А за счет чего это достигается?

```
public interface ISourceGenerator {  
    void Initialize(GeneratorInitializationContext context);  
    void Execute(GeneratorExecutionContext context);  
}
```

Как работают генераторы



Генераторы в IDE



Эмулируем большой проект

- 3 генератора
 - AutoNotifyGenerator
 - JsonSrcGen
 - ThisAssembly
- ~1500 классов
 - ~100 с атрибутами [AutoNotify]
 - ~100 с атрибутами [Json]
- 1 файл с ресурсами
 - ~500 констант

Visual Studio interface showing a C# code editor with the following code:

```
4 {
5 static void Main(string[] args)
6 {
7     var converter = new JsonSrcGen.JsonConverter();
8     var json = converter.ToJson(new JsonSerializerizableValue
9     {
10         Property = "my string value",
11         Value = null,
12         IntValue = 10
13     });
14     Console.WriteLine(new string(json));
15     var deserialized = new JsonSerializerizableValue();
16     converter.FromJson(deserialized, json);
17     Console.WriteLine("Property = " + deserialized.Property);
18     Console.WriteLine("Value = {0}", deserialized.Value?.ToString() ?? "null");
19     Console.WriteLine("IntValue = " + deserialized.IntValue);
20
21     while (true)
22     {
23         var text = Console.ReadLine();
24         if (text == "exit")
25             break;
26
27         Console.WriteLine("Hello " + text + "!");
28     }
29 }
30 }
31 }
32
33 [JsonSrcGen.Json]
34 [JsonSrcGen.JsonIgnoreNull]
35 public class JsonSerializerizableValue
36 {
37     public string Property { get; set; }
38     public int? Value { get; set; }
```

The Solution Explorer on the right shows the project structure:

- IncrementalDemo
 - Dependencies
 - LargeProjectFolder
 - AutoNotifyTypes.cs
 - LotsOfTypes.cs
 - Resources
 - IncrementalDemo.csproj.DotSettings
 - Resources1.resx

The Toolbox on the right shows the General tab with the following items:

- Pointer
- JsonSrcGen

At the bottom of the code editor, the status bar shows: Ln: 21 Ch: 7 SPC CRLF

CsWin32 analyzer slows Visual Studio 2019 editor down to a crawl #244

New issue

Open BasTossings opened this issue on Apr 17 · 9 comments



BasTossings commented on Apr 17 · edited

Actual behavior

As soon as I add a reference to `Microsoft.Windows.CsWin32` to my project, the Visual Studio 2019 editor slows down to a crawl to the point of barely being usable anymore. The editor lags behind as I type, autocomplete takes several seconds (3~10) to pop up. Code color coding and error underlining is delayed by the same amount.

The moment I remove the reference and hit save, it's back to normal again and editing, autocomplete and code coloring and error underlining is again instantaneous.

Expected behavior

I'd expect the code generator/analyzer to perform better. I see no reason it should be so slow.

Assignees

No one assigned

Labels

bug

Projects

None yet



Milestone

No milestone





bit.ly/3oKldji

ThisAssembly.StringsGenerator

- ▾ **100.00%** Execute • 51,088 ms • ThisAssembly.**StringsGenerator**.Execute(GeneratorExecutionContext)
 -  **60.80%** NormalizeWhitespace • 31,062 ms • Microsoft.CodeAnalysis.**SyntaxNodeExtensions**.NormalizeWhitespace(TNode, String, String, Boolean)
 -  **24.18%** ParseCompilationUnit • 12,354 ms • Microsoft.CodeAnalysis.CSharp.**SyntaxFactory**.ParseCompilationUnit(String, Int32, CSharpParseOptions)
 - **9.61%** Render • 4,910 ms • Scriban.**Template**.Render(Object, MemberRenamerDelegate, MemberFilterDelegate)
 - ▾ **2.70%** GetText • 1,378 ms • Microsoft.CodeAnalysis.**SyntaxNode**.GetText(Encoding, SourceHashAlgorithm)
 - **1.71%** Load • 874 ms • **ResourceFile**.Load(String, String)
 - ▾ **0.81%** ToString • 415 ms • Microsoft.CodeAnalysis.Text.**StringBuilderText**.ToString(TextSpan)
 - **0.18%** Parse • 94 ms • Scriban.**Template**.Parse(String, String, Nullable, Nullable)

ThisAssembly.StringsGenerator

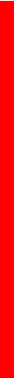
▾	100.00%	Execute	• 51,088 ms	• ThisAssembly. StringsGenerator .Execute(GeneratorExecutionContext)	
▸		60.80%	NormalizeWhitespace	• 31,062 ms	• Microsoft.CodeAnalysis. SyntaxNodeExtensions .NormalizeWhitespace(TNode, String, String, Boolean)
▸		24.18%	ParseCompilationUnit	• 12,354 ms	• Microsoft.CodeAnalysis.CSharp. SyntaxFactory .ParseCompilationUnit(String, Int32, CSharpParseOptions)
▸		9.61%	Render	• 4,910 ms	• Scriban. Template .Render(Object, MemberRenamerDelegate, MemberFilterDelegate)
▾		2.70%	GetText	• 1,378 ms	• Microsoft.CodeAnalysis. SyntaxNode .GetText(Encoding, SourceHashAlgorithm)
▸		1.71%	Load	• 874 ms	• ResourceFile .Load(String, String)
▾		0.81%	ToString	• 415 ms	• Microsoft.CodeAnalysis.Text. StringBuilderText .ToString(TextSpan)
▸		0.18%	Parse	• 94 ms	• Scriban. Template .Parse(String, String, Nullable, Nullable)

- 15% генерация кода
 - 2% чтение файла ресурсов
 - 10% обработка шаблона
- 85% времени на форматирование
 - 25% парсинг кода для форматирования
 - 60% вставка индента

JsonSrcGen – 100 [Json] типов из 1500

▲ 100.00%	Execute	• 7,175 ms	• JsonSrcGen.JsonGenerator.Execute(GeneratorExecutionContext)
▲ 100.00%	Generate	• 7,175 ms	• JsonSrcGen.JsonGenerator.Generate(GeneratorExecutionContext)
▲ 49.26%	GenerateFromResource	• 3,534 ms	• JsonSrcGen.JsonGenerator.GenerateFromResource(String, GeneratorExecutionContext, Compilation, String)
▼ 36.31%	ParseText	• 2,605 ms	• Microsoft.CodeAnalysis.CSharp.CSharpSyntaxTree.ParseText(SourceText, CSharpParseOptions, String, CancellationToken)
▼ 7.58%	ToString	• 544 ms	• System.Text.StringBuilder.ToString
▼ 4.80%	ReadToEnd	• 344 ms	• System.IO.StreamReader.ReadToEnd
▼ 0.57%	AddSyntaxTrees	• 41 ms	• Microsoft.CodeAnalysis.CSharp.CSharpCompilation.AddSyntaxTrees(IEnumerable)
▲ 26.75%	GetJsonClassInfo	• 1,920 ms	• JsonSrcGen.JsonGenerator.GetJsonClassInfo(List, Compilation)
▶ 7.94%	GetMembers	• 570 ms	• Microsoft.CodeAnalysis.CSharp.Symbols.PublicModel.NamespaceOrTypeSymbol.GetMembers
▶ 5.96%	GetType	• 427 ms	• JsonSrcGen.JsonGenerator.GetType(ISymbol, SemanticModel)
▶ 5.76%	HasJsonClassAttribute	• 413 ms	• JsonSrcGen.JsonGenerator.HasJsonClassAttribute(ISymbol)
▼ 2.14%	GetDeclaredSymbol	• 153 ms	• Microsoft.CodeAnalysis.CSharp.CSharpExtensions.GetDeclaredSymbol(SemanticModel, BaseTypeDeclarationSyntax, CancellationToken)
▼ 2.08%	GetSemanticModel	• 149 ms	• Microsoft.CodeAnalysis.CSharp.CSharpCompilation.GetSemanticModel(SyntaxTree, Boolean)
▼ 1.24%	GetAttributes	• 89 ms	• Microsoft.CodeAnalysis.CSharp.Symbols.PublicModel.Symbol.GetAttributes
▶ 0.92%	Any	• 66 ms	• System.Linq.Enumerable.Any(IEnumerable, Func)
▶ 0.49%	Kind	• 35 ms	• Microsoft.CodeAnalysis.CSharp.CSharpSyntaxNode.Kind
▼ 0.24%	get_Modifiers	• 17 ms	• Microsoft.CodeAnalysis.CSharp.Syntax.ClassDeclarationSyntax.get_Modifiers
▶ 9.11%	Generate	• 653 ms	• JsonSrcGen.FromJsonGenerator.Generate(JsonClass, CodeBuilder)
▶ 6.38%	GenerateUtf8	• 458 ms	• JsonSrcGen.ToJsonGenerator.GenerateUtf8(JsonClass, CodeBuilder)
▶ 4.08%	GenerateUtf8	• 293 ms	• JsonSrcGen.FromJsonGenerator.GenerateUtf8(JsonClass, CodeBuilder)
▶ 1.87%	GetCustomTypeConverters	• 134 ms	• JsonSrcGen.JsonGenerator.GetCustomTypeConverters(List, Compilation, Utf8Literals)
▶ 1.61%	Generate	• 116 ms	• JsonSrcGen.ToJsonGenerator.Generate(JsonClass, CodeBuilder)
▶ 0.32%	GetListAttributesInfo	• 23 ms	• JsonSrcGen.JsonGenerator.GetListAttributesInfo(List, Compilation)
▶ 0.17%	GetGenerationOutputFolder	• 12 ms	• JsonSrcGen.JsonGenerator.GetGenerationOutputFolder(List, Compilation)
▶ 0.14%	GetArrayAttributesInfo	• 10 ms	• JsonSrcGen.JsonGenerator.GetArrayAttributesInfo(List, Compilation)
▶ 0.14%	GenerateMatches	• 10 ms	• JsonSrcGen.Utf8Literals.GenerateMatches(CodeBuilder)

JsonSrcGen – 100 [Json] типов из 1500



▲	100.00%	Execute	• 7,175 ms	•	JsonSrcGen.JsonGenerator.Execute(GeneratorExecutionContext)		
▲	100.00%	Generate	• 7,175 ms	•	JsonSrcGen.JsonGenerator.Generate(GeneratorExecutionContext)		
▢	▲	49.26%	GenerateFromResource	• 3,534 ms	•	JsonSrcGen.JsonGenerator.GenerateFromResource(String, GeneratorExecutionContext, Compilation, String)	
		▼	36.31%	ParseText	• 2,605 ms	•	Microsoft.CodeAnalysis.CSharp.CSharpSyntaxTree.ParseText(SourceText, CSharpParseOptions, String, CancellationToken)
		▼	7.58%	ToString	• 544 ms	•	System.Text.StringBuilder.ToString
		▼	4.80%	ReadToEnd	• 344 ms	•	System.IO.StreamReader.ReadToEnd
		▼	0.57%	AddSyntaxTrees	• 41 ms	•	Microsoft.CodeAnalysis.CSharp.CSharpCompilation.AddSyntaxTrees(IEnumerable)
▢	▲	26.75%	GetJsonClassInfo	• 1,920 ms	•	JsonSrcGen.JsonGenerator.GetJsonClassInfo(List, Compilation)	
		▶	7.94%	GetMembers	• 570 ms	•	Microsoft.CodeAnalysis.CSharp.Symbols.PublicModel.NamespaceOrTypeSymbol.GetMembers
		▶	5.96%	GetType	• 427 ms	•	JsonSrcGen.JsonGenerator.GetType(ISymbol, SemanticModel)
		▶	5.76%	HasJsonClassAttribute	• 413 ms	•	JsonSrcGen.JsonGenerator.HasJsonClassAttribute(ISymbol)
		▼	2.14%	GetDeclaredSymbol	• 153 ms	•	Microsoft.CodeAnalysis.CSharp.CSharpExtensions.GetDeclaredSymbol(SemanticModel, BaseTypeDeclarationSyntax, CancellationToken)
		▼	2.08%	GetSemanticModel	• 149 ms	•	Microsoft.CodeAnalysis.CSharp.CSharpCompilation.GetSemanticModel(SyntaxTree, Boolean)
		▼	1.24%	GetAttributes	• 89 ms	•	Microsoft.CodeAnalysis.CSharp.Symbols.PublicModel.Symbol.GetAttributes
		▶	0.92%	Any	• 66 ms	•	System.Linq.Enumerable.Any(IEnumerable, Func)
			0.49%	Kind	• 35 ms	•	Microsoft.CodeAnalysis.CSharp.CSharpSyntaxNode.Kind
		▼	0.24%	get_Modifiers	• 17 ms	•	Microsoft.CodeAnalysis.CSharp.Syntax.ClassDeclarationSyntax.get_Modifiers
		▶	9.11%	Generate	• 653 ms	•	JsonSrcGen.FromJsonGenerator.Generate(JsonClass, CodeBuilder)
		▶	6.38%	GenerateUtf8	• 458 ms	•	JsonSrcGen.ToJsonGenerator.GenerateUtf8(JsonClass, CodeBuilder)
		▶	4.08%	GenerateUtf8	• 293 ms	•	JsonSrcGen.FromJsonGenerator.GenerateUtf8(JsonClass, CodeBuilder)
		▶	1.87%	GetCustomTypeConverters	• 134 ms	•	JsonSrcGen.JsonGenerator.GetCustomTypeConverters(List, Compilation, Utf8Literals)
		▶	1.61%	Generate	• 116 ms	•	JsonSrcGen.ToJsonGenerator.Generate(JsonClass, CodeBuilder)
		▶	0.32%	GetListAttributesInfo	• 23 ms	•	JsonSrcGen.JsonGenerator.GetListAttributesInfo(List, Compilation)
			0.17%	GetGenerationOutputFolder	• 12 ms	•	JsonSrcGen.JsonGenerator.GetGenerationOutputFolder(List, Compilation)
		▶	0.14%	GetArrayAttributesInfo	• 10 ms	•	JsonSrcGen.JsonGenerator.GetArrayAttributesInfo(List, Compilation)
		▶	0.14%	GenerateMatches	• 10 ms	•	JsonSrcGen.Utf8Literals.GenerateMatches(CodeBuilder)

JsonSrcGen – 100 [Json] типов из 1500

▲	100.00%	Execute	• 7,175 ms	•	JsonSrcGen.JsonGenerator.Execute(GeneratorExecutionContext)	
▲	100.00%	Generate	• 7,175 ms	•	JsonSrcGen.JsonGenerator.Generate(GeneratorExecutionContext)	
▢	▲	49.26%	GenerateFromResource	• 3,534 ms	•	JsonSrcGen.JsonGenerator.GenerateFromResource(String, GeneratorExecutionContext, Compilation, String)
	▼	36.31%	ParseText	• 2,605 ms	•	Microsoft.CodeAnalysis.CSharp.CSharpSyntaxTree.ParseText(SourceText, CSharpParseOptions, String, CancellationToken)
	▼	7.58%	ToString	• 544 ms	•	System.Text.StringBuilder.ToString
	▼	4.80%	ReadToEnd	• 344 ms	•	System.IO.StreamReader.ReadToEnd
	▼	0.57%	AddSyntaxTrees	• 41 ms	•	Microsoft.CodeAnalysis.CSharp.CSharpCompilation.AddSyntaxTrees(IEnumerable)
▢	▲	26.75%	GetJsonClassInfo	• 1,920 ms	•	JsonSrcGen.JsonGenerator.GetJsonClassInfo(List, Compilation)
	▶	7.94%	GetMembers	• 570 ms	•	Microsoft.CodeAnalysis.CSharp.Symbols.PublicModel.NamespaceOrTypeSymbol.GetMembers
	▶	5.96%	GetType	• 427 ms	•	JsonSrcGen.JsonGenerator.GetType(ISymbol, SemanticModel)
	▶	5.76%	HasJsonClassAttribute	• 413 ms	•	JsonSrcGen.JsonGenerator.HasJsonClassAttribute(ISymbol)
	▼	2.14%	GetDeclaredSymbol	• 153 ms	•	Microsoft.CodeAnalysis.CSharp.CSharpExtensions.GetDeclaredSymbol(SemanticModel, BaseTypeDeclarationSyntax, CancellationToken)
	▼	2.08%	GetSemanticModel	• 149 ms	•	Microsoft.CodeAnalysis.CSharp.CSharpCompilation.GetSemanticModel(SyntaxTree, Boolean)
	▼	1.24%	GetAttributes	• 89 ms	•	Microsoft.CodeAnalysis.CSharp.Symbols.PublicModel.Symbol.GetAttributes
	▶	0.92%	Any	• 66 ms	•	System.Linq.Enumerable.Any(IEnumerable, Func)
		0.49%	Kind	• 35 ms	•	Microsoft.CodeAnalysis.CSharp.CSharpSyntaxNode.Kind
	▼	0.24%	get_Modifiers	• 17 ms	•	Microsoft.CodeAnalysis.CSharp.Syntax.ClassDeclarationSyntax.get_Modifiers
	▶	9.11%	Generate	• 653 ms	•	JsonSrcGen.FromJsonGenerator.Generate(JsonClass, CodeBuilder)
	▶	6.38%	GenerateUtf8	• 458 ms	•	JsonSrcGen.ToJsonGenerator.GenerateUtf8(JsonClass, CodeBuilder)
	▶	4.08%	GenerateUtf8	• 293 ms	•	JsonSrcGen.FromJsonGenerator.GenerateUtf8(JsonClass, CodeBuilder)
	▶	1.87%	GetCustomTypeConverters	• 134 ms	•	JsonSrcGen.JsonGenerator.GetCustomTypeConverters(List, Compilation, Utf8Literals)
	▶	1.61%	Generate	• 116 ms	•	JsonSrcGen.ToJsonGenerator.Generate(JsonClass, CodeBuilder)
	▶	0.32%	GetListAttributesInfo	• 23 ms	•	JsonSrcGen.JsonGenerator.GetListAttributesInfo(List, Compilation)
		0.17%	GetGenerationOutputFolder	• 12 ms	•	JsonSrcGen.JsonGenerator.GetGenerationOutputFolder(List, Compilation)
	▶	0.14%	GetArrayAttributesInfo	• 10 ms	•	JsonSrcGen.JsonGenerator.GetArrayAttributesInfo(List, Compilation)
	▶	0.14%	GenerateMatches	• 10 ms	•	JsonSrcGen.Utf8Literals.GenerateMatches(CodeBuilder)

Какие выводы мы можем сделать?

- Синтаксическая модель – можно считать мгновенной
- Семантическая модель – ленивая и относительно быстрая
 - Используется не только генераторами
 - Анализы
 - Код компишен
 - Является основой компилятора и постоянно оптимизируется
- Парсить код – медленно!

An iceberg floating in the ocean. The tip of the iceberg is above the water surface, and the much larger part is submerged below. The sky is blue with light clouds, and the water is a deep blue.

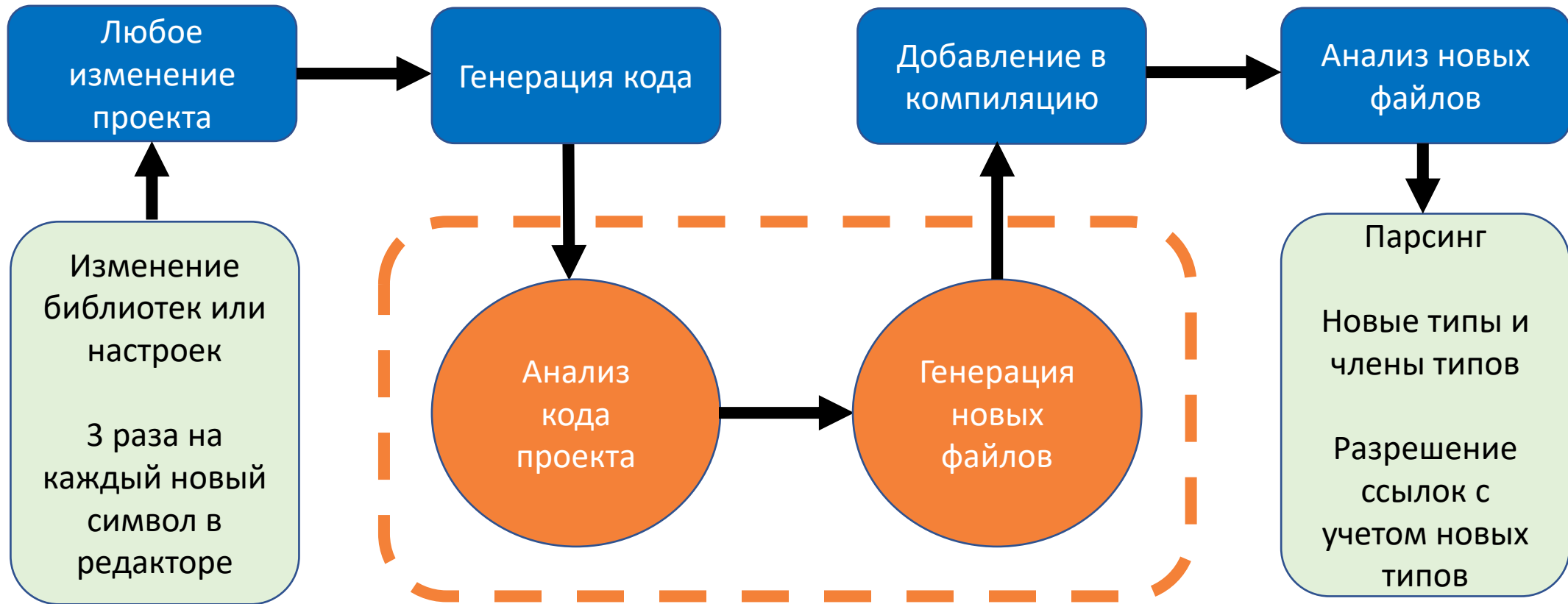
Запуск генератора

Парсинг

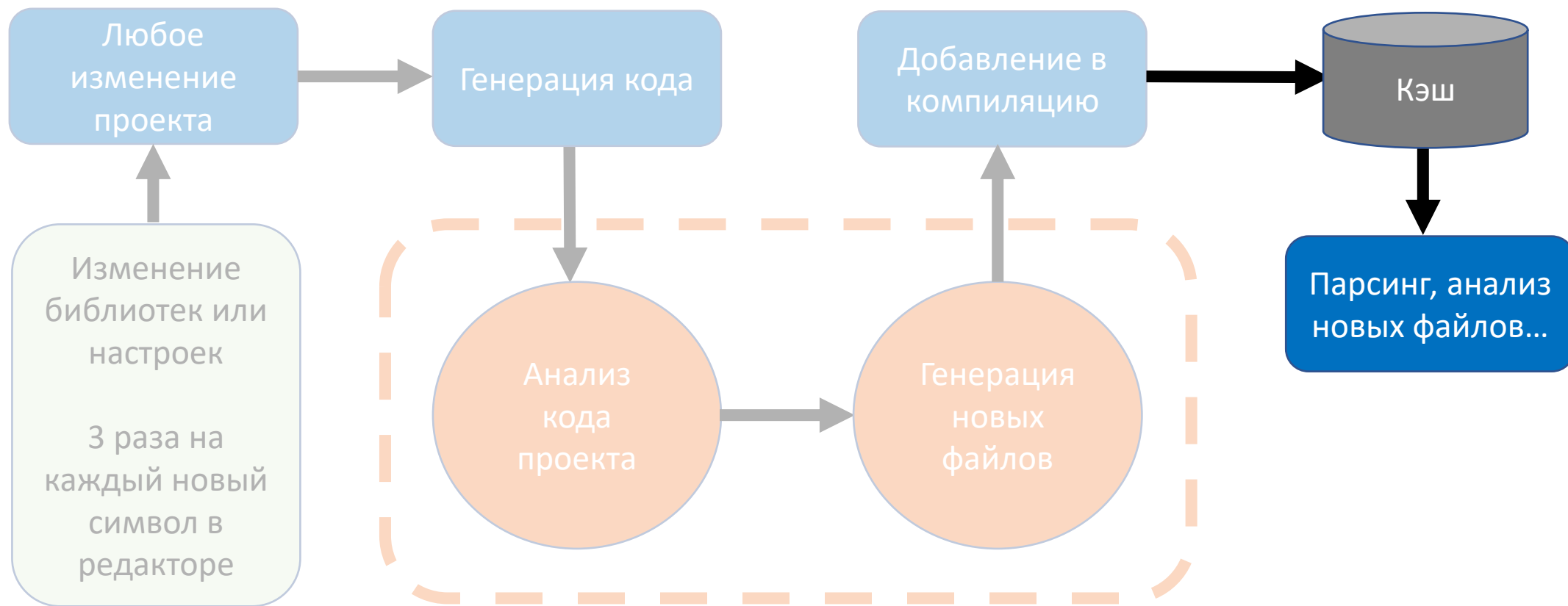
Анализ нового кода

Изменение семантики

Генераторы в IDE



Генераторы в IDE



bit.ly/3AkbTG5

From VS 16.10 Preview2+

Для использования кэша генератор
должен создавать те же файлы

```
static class UniqueNumberGenerator {  
    static int _number = 0;  
  
    public static int UniqueNumber => _number++;  
}
```

Для использования кэша генератор
должен создавать те же файлы

```
static class UniqueNumberGenerator {  
    static int _number = 0;  
  
    public static int UniqueNumber => _number++;  
}
```

```
var builderFieldName = $"listBuilder{UniqueNumberGenerator.UniqueNumber}";
```

```
codeBuilder.AppendLine(indentLevel, $"var {builderFieldName} = {...};");  
codeBuilder.AppendLine(indentLevel, $"if({builderFieldName} == null)");
```

Для использования кэша генератор должен создавать те же файлы

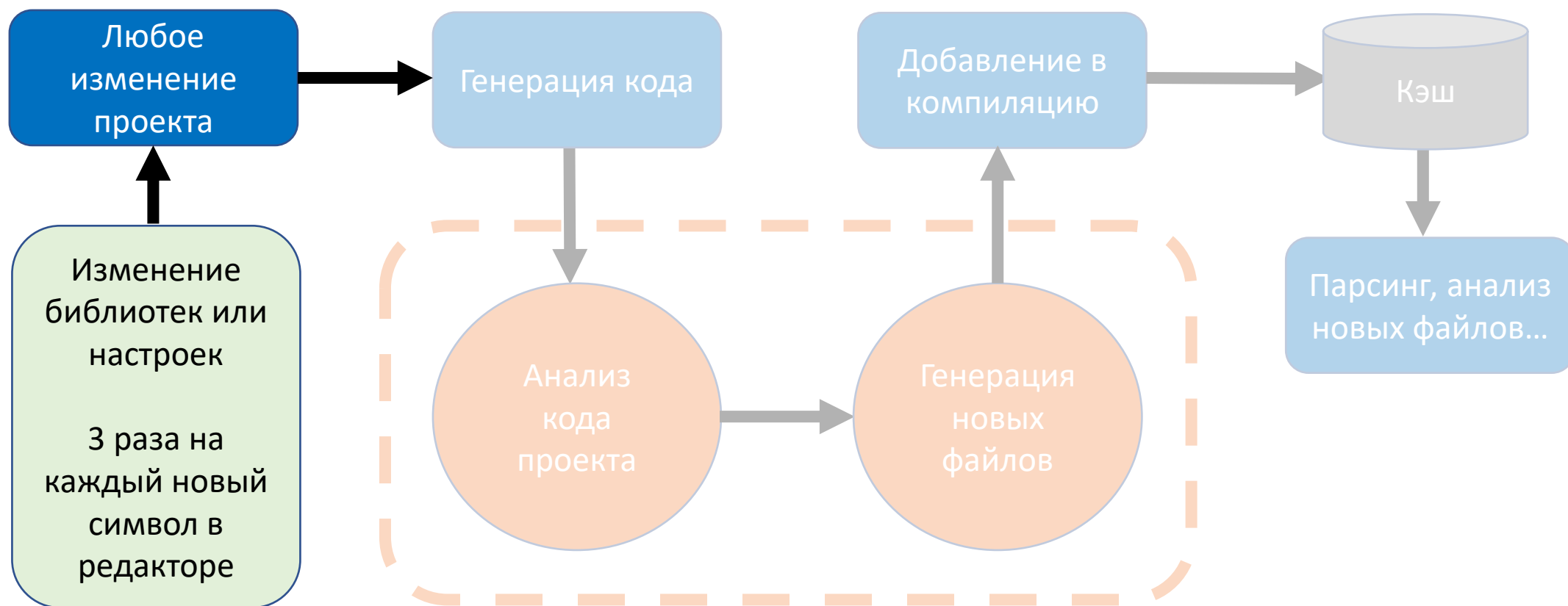
```
Miscellaneous Files | JsonSrcGen.JsonConverter | FromJson(Launch?[]? value, Re...
}
return builder.AsSpan();
}
public JsonSrcGen.RealJsonTests.SpaceX.Launch?[]? FromJson(JsonSrcGen
{
    var listBuilder32697 = _listBuilder32696;
    if(listBuilder32697 == null)
    {
        listBuilder32697 = new List<JsonSrcGen.RealJsonTests.SpaceX.
        _listBuilder32696 = listBuilder32697;
    }
    listBuilder32697.Clear();
    json = json.SkipWhitespaceTo('[', 'n', out char found32698);
    if(found32698 == 'n')
    {
        json = json.Slice(3);
        listBuilder32697 = null;
    }
    else
    {
        listBuilder32697.Clear();
        while(true)
        {
            if(json[0] == '[')
            {
                json = json.Slice(1);
                break;
            }
            json = json.SkinWhitespace();
        }
    }
}

Miscellaneous Files | JsonSrcGen.JsonConverter | FromJson(Launch?[]? value, Re...
}
return builder.AsSpan();
}
public JsonSrcGen.RealJsonTests.SpaceX.Launch?[]? FromJson(JsonSrcGen
{
    var listBuilder47405 = _listBuilder47404;
    if(listBuilder47405 == null)
    {
        listBuilder47405 = new List<JsonSrcGen.RealJsonTests.SpaceX.
        _listBuilder47404 = listBuilder47405;
    }
    listBuilder47405.Clear();
    json = json.SkipWhitespaceTo('[', 'n', out char found47406);
    if(found47406 == 'n')
    {
        json = json.Slice(3);
        listBuilder47405 = null;
    }
    else
    {
        listBuilder47405.Clear();
        while(true)
        {
            if(json[0] == '[')
            {
                json = json.Slice(1);
                break;
            }
            json = json.SkinWhitespace();
        }
    }
}
```


Компилятору нужно ваше содействие!

- Генератор чаще всего будет создавать одно и то же
 - Обычно генератор зависит от иерархии типов \ атрибутов
 - Большая часть запусков генератора – набор кода внутри метода
- Генератор должен быть детерминированным
- Избавьтесь от любых случайных данных
 - `random.Next()`;
 - `Guid.NewGuid()`;
- Избавьтесь от статических переменных

Генераторы в IDE

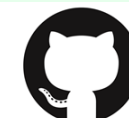


Надо ли перезапускать генератор?...

- ThisAssembly
 - Добавление ресурсов\констант по подключенным файлам и свойствам
- Svg to C#
 - Создание C#-представления .svg изображений по ссылкам из .csproj
- CsWin32
 - Импорт Win32 сигнатур по списку из .txt файла
- RoslynSyntaxGenerator
 - Создание классов по xml-описанию

Можно самим проверять входные данные

```
74 + // Get the current input checksum, which will either be used for verifying the current cache or updating it
75 + // with the new results.
76 + var currentChecksum = syntaxXmlText.GetChecksum();
77 +
78 + // Read the current cached result once to avoid race conditions
79 + if (s_cachedResult is { } cachedResult
80 +     && cachedResult.Item1.SequenceEqual(currentChecksum))
81 + {
82 +     foreach (var (hintName, sourceText) in cachedResult.Item2)
83 +     {
84 +         context.AddSource(hintName, sourceText);
85 +     }
86 +
87 +     return;
88 + }
89 +
```



Почему компилятор сам не закеширует результат?



Почему компилятор сам не закеширует результат?



- Один объект со всеми данными
- Нет возможности проверить к чему вы обращались
- Чтобы гарантировать корректность надо перезапускать на любое изменение

Можно самим проверять входные данные

- В каждом генераторе нужно писать свое кэширование
 - Автор забыл – пользователи страдают
- Что если файлов много?
 - Сотни обращений к файлам на каждый символ в редакторе?
- Никаких гарантий
 - Компилятор не гарантирует переиспользование инстанса генератора
 - Реализация может поменяться

IIncrementalGenerator

```
public interface IIncrementalGenerator
{
    void Initialize(IncrementalGeneratorInitializationContext context);
}
```

- Полностью новый интерфейс генератора
- Вы можете определить схему работы
- Вы можете указать от чего зависит результат работы генератора

IncrementalGenerator

```
public partial readonly struct IncrementalGeneratorInitializationContext
{
    SyntaxValueProvider SyntaxProvider;

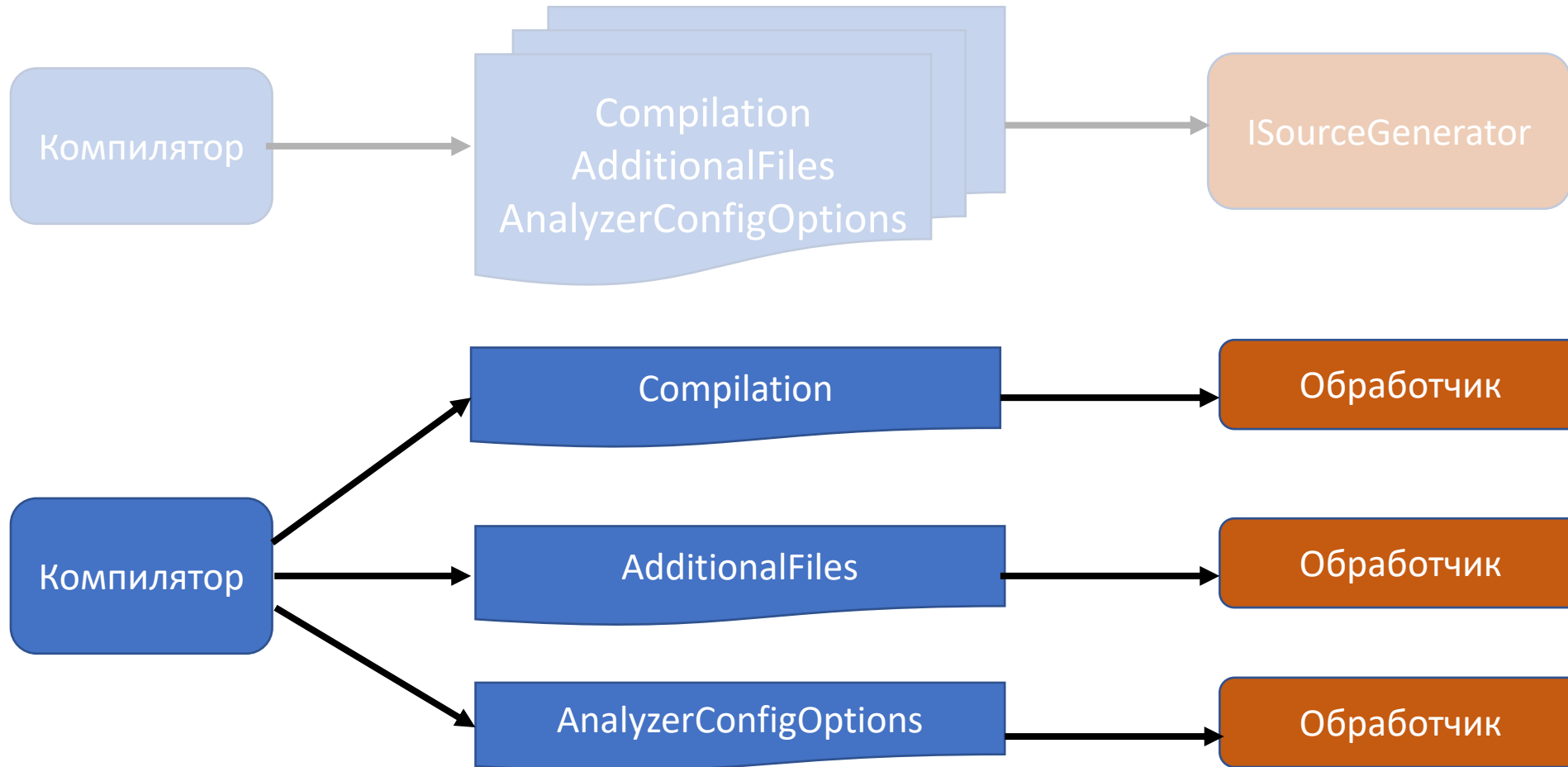
    IncrementalValueProvider<Compilation> CompilationProvider;
    IncrementalValueProvider<ParseOptions> ParseOptionsProvider;
    IncrementalValuesProvider<AdditionalText> AdditionalTextsProvider;
    IncrementalValueProvider<AnalyzerConfigOptionsProvider> AnalyzerConfigOptionsProvider;

    void RegisterSourceOutput<TSource>(
        IncrementalValueProvider<TSource> source,
        Action<SourceProductionContext, TSource> action);
}
```

ISourceGenerator



IncrementalGenerator



Как новый интерфейс нам поможет

```
public void ISourceGenerator.Execute(GeneratorExecutionContext context)
{
    var files = context.AdditionalFiles
        .Where(at => at.Path.EndsWith(".svg"));
    foreach (var file in files)
    {
        var newSource = GenerateCode(file.Path, file.GetText());
        var name = GetName(file);
        context.AddSource(name, newSource);
    }
}
```

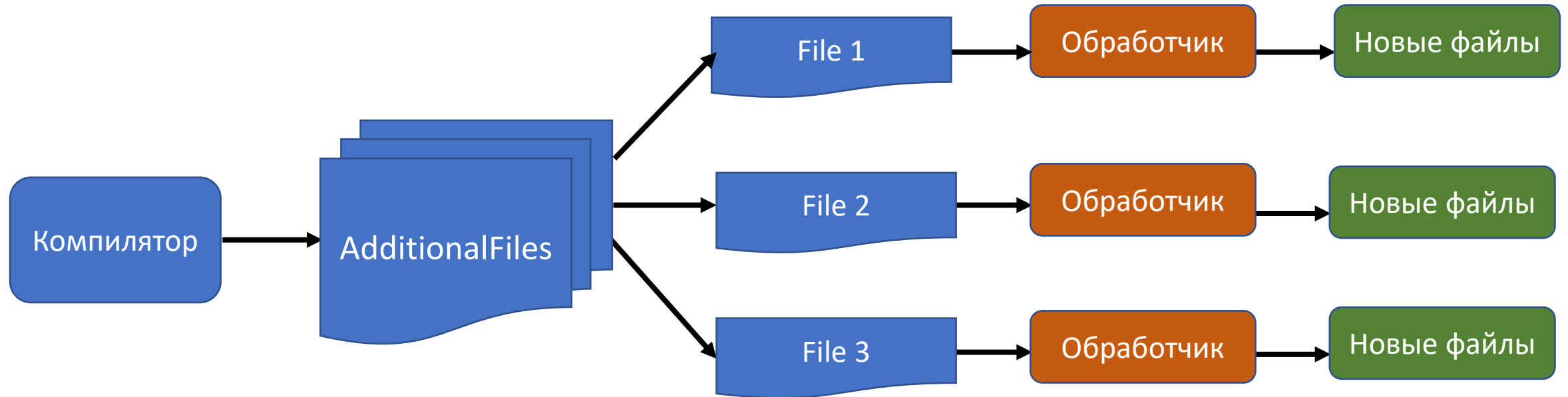
Как новый интерфейс нам поможет

```
public void ISourceGenerator.Execute(GeneratorExecutionContext context)
{
    var files = context.AdditionalFiles
        .Where(at => at.Path.EndsWith(".svg"));
    foreach (var file in files)
    {
        var newSource = GenerateCode(file.Path, file.GetText());
        var name = GetName(file);
        context.AddSource(name, newSource);
    }
}
```

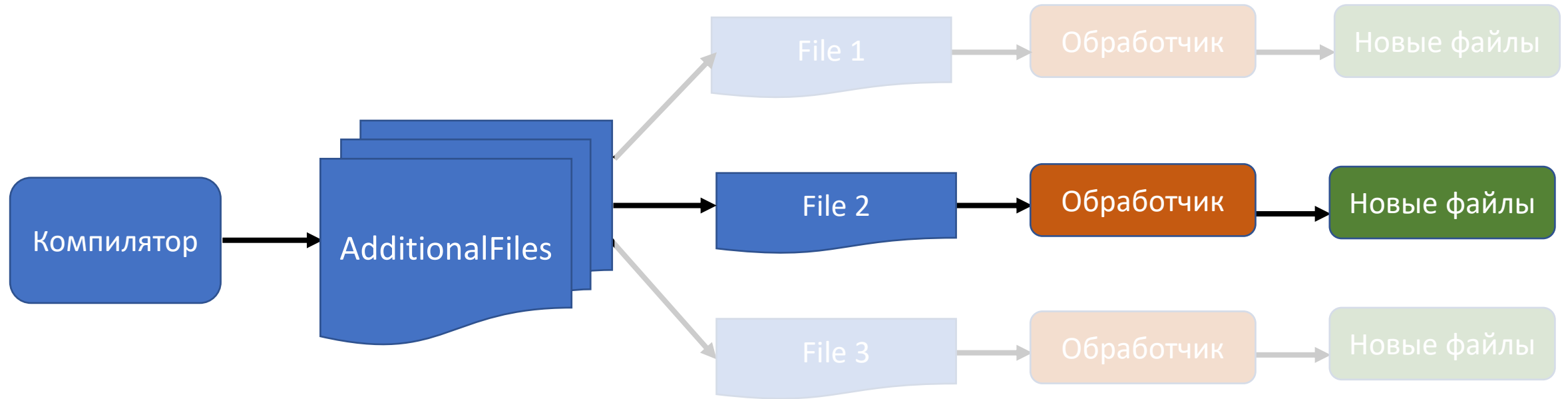
Как новый интерфейс нам поможет

```
public void IIncrementalGenerator.Initialize(  
    IncrementalGeneratorInitializationContext context)  
{  
    var filesProvider = context.AdditionalTextsProvider  
        .Where(at => at.Path.EndsWith(".svg"));  
    context.RegisterSourceOutput(filesProvider,  
        (context, file) =>  
        {  
            var newSource = GenerateCode(file.Path, file.GetText());  
            var name = GetName(file);  
            context.AddSource(name, newSource);  
        });  
}
```

Как работает новый генератор?



Как работает новый генератор?



А что если надо объединить данные из
нескольких провайдеров?

А что если надо объединить данные из нескольких провайдеров?

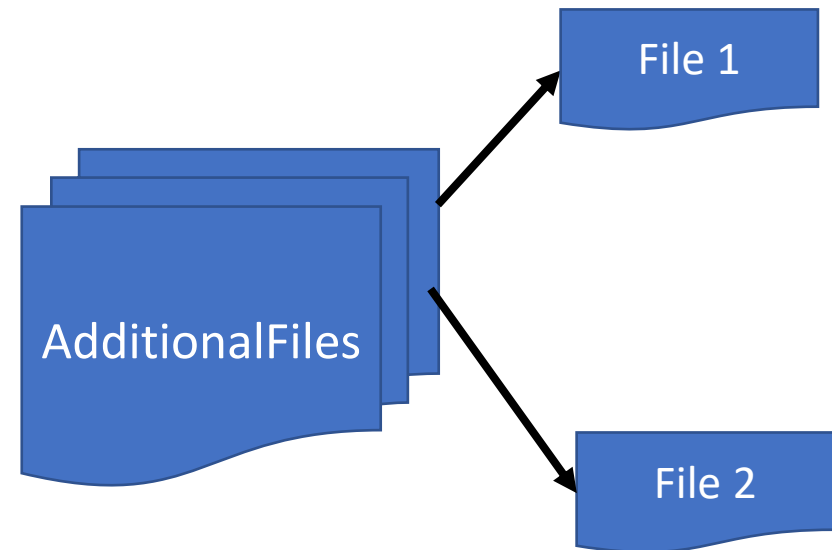
```
public void ISourceGenerator.Execute(GeneratorExecutionContext context)
{
    context.AnalyzerConfigOptions.GlobalOptions.TryGetValue(
        "build_property.NamespaceName", out var globalNamespaceName);

    var files = context.AdditionalFiles.Where(at => at.Path.EndsWith(".svg"));
    foreach (var file in files)
    {
        var newSource = GenerateCode(file.Path, file.GetText(), globalNamespaceName);
        var name = GetName(file);
        context.AddSource(name, newSource);
    }
}
```

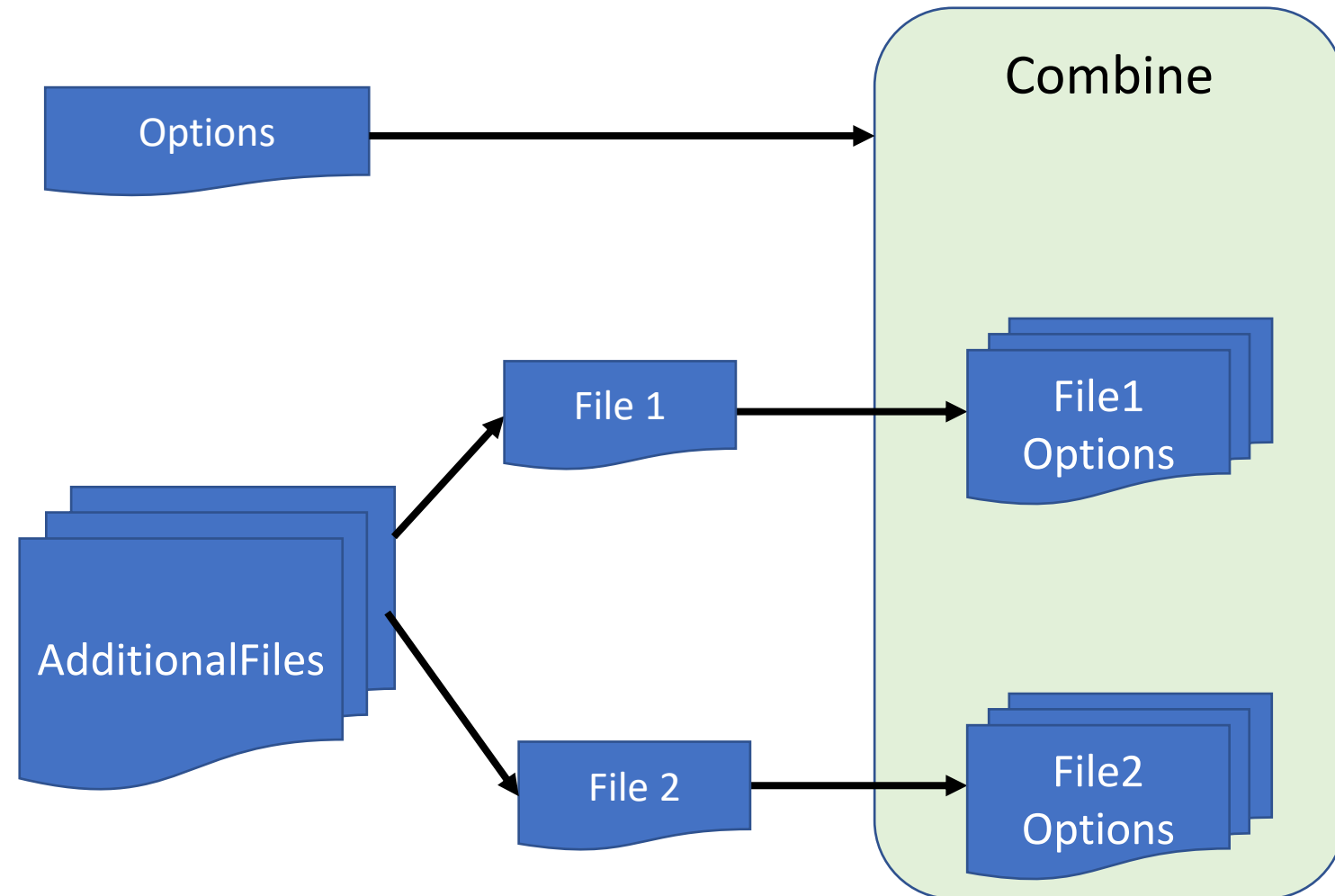
Несколько провайдеров данных

```
public void IIncrementalGenerator.Initialize(
    IncrementalGeneratorInitializationContext context)
{
    var filesProvider = context.AdditionalTextsProvider
        .Where(at => at.Path.EndsWith(".svg"))
        .Combine(context.AnalyzerConfigOptionsProvider);
    context.RegisterSourceOutput(filesProvider,
        (context, data) =>
        {
            var (file, options) = data;
            options.GlobalOptions.TryGetValue(
                "build_property.NamespaceName", out var globalNamespaceName);
            // ...
        });
};
```

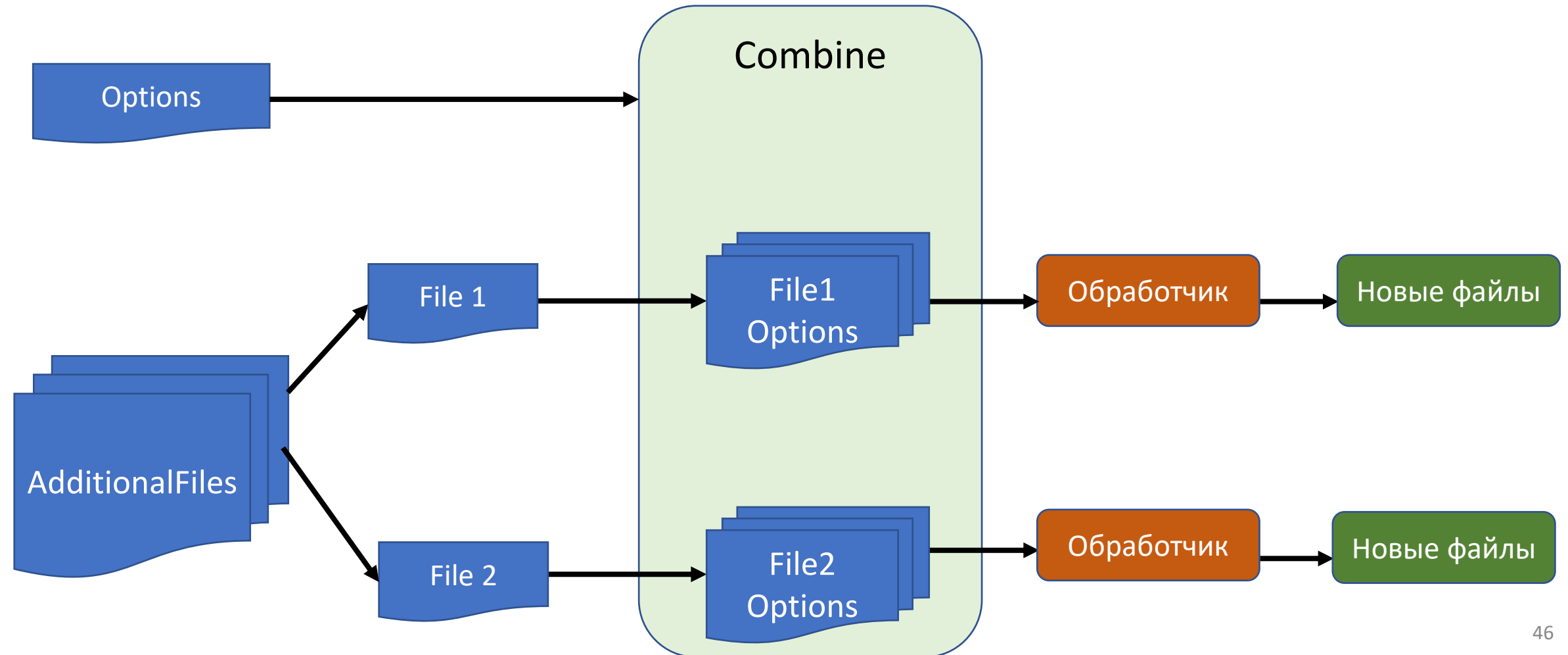
Как тогда выглядит генерация?



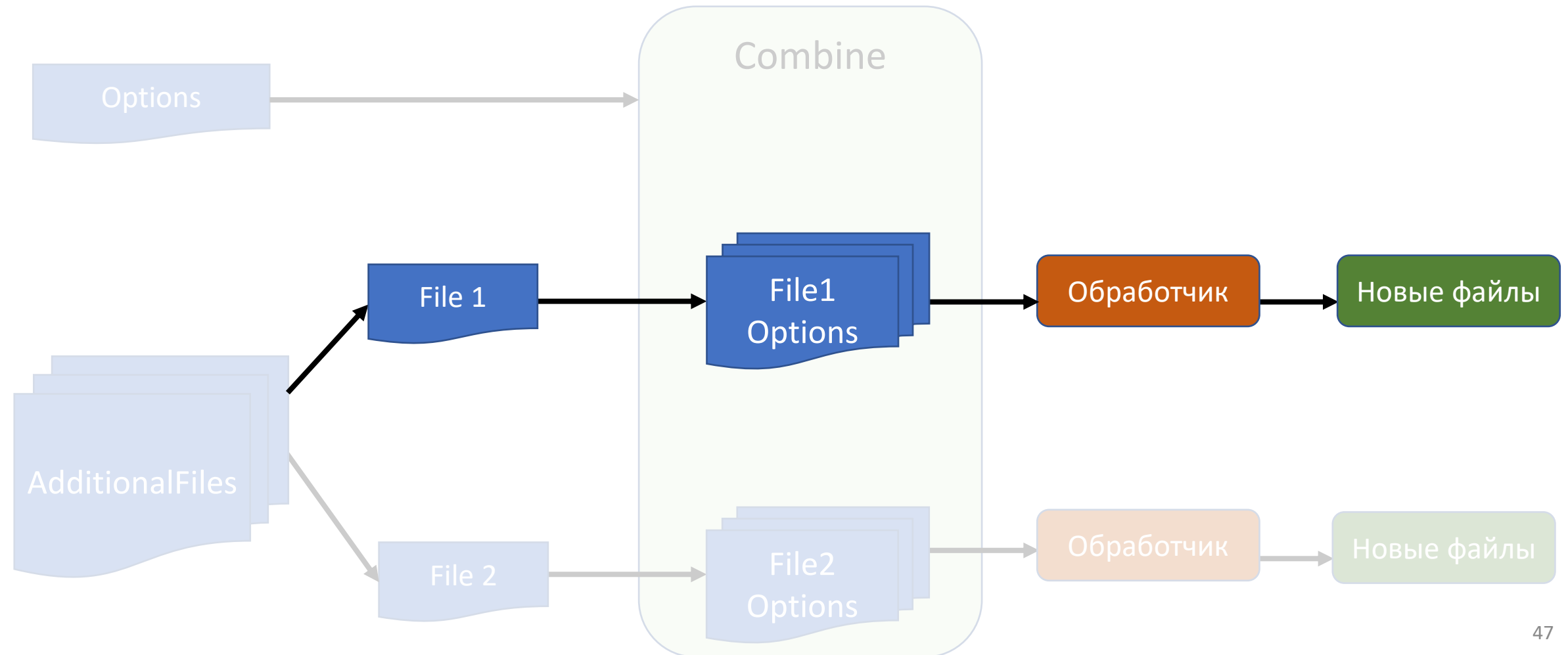
Как тогда выглядит генерация?



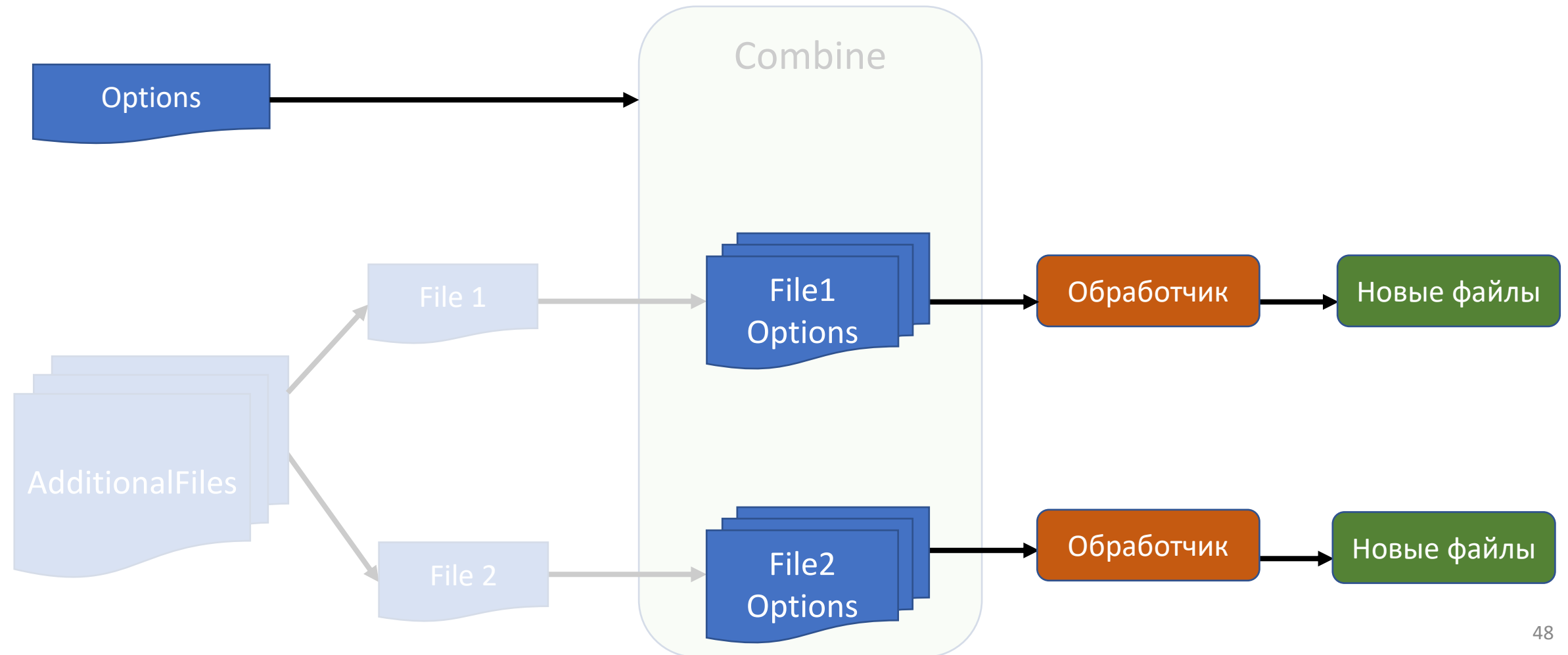
Как тогда выглядит генерация?



Как тогда выглядит генерация?



Как тогда выглядит генерация?



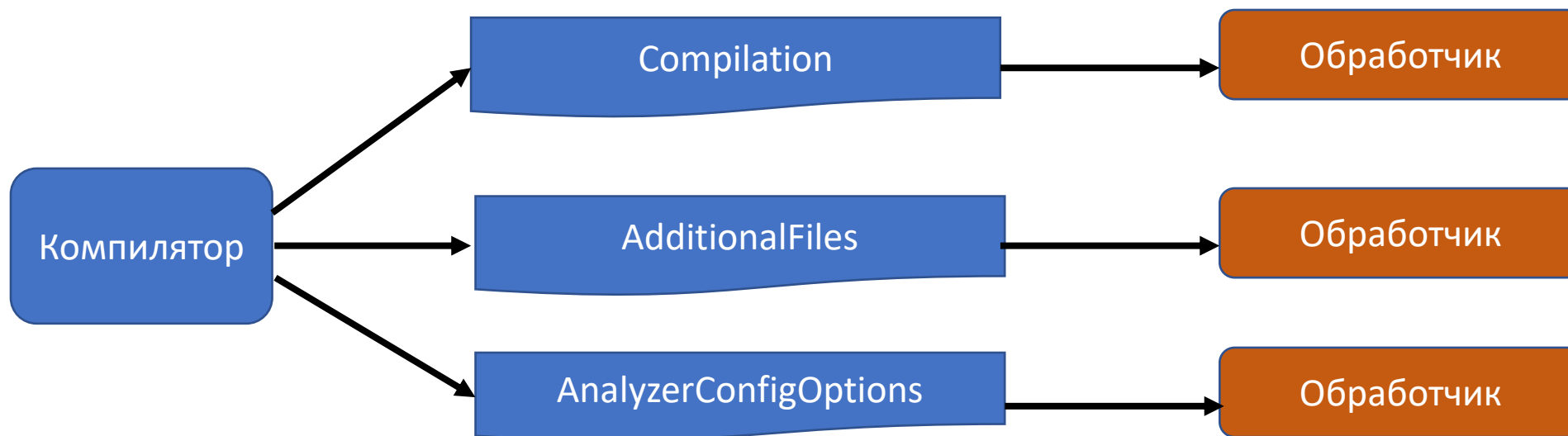
Если генератор не зависит от кода

- Переведите его на новый интерфейс
 - Огромный прирост производительности для пользователей генератора
- Генератор будет запускаться очень редко
 - Открытие проекта
 - Компиляция
 - Изменение нужных ему файлов
- Если вы отказались от генератора для «одноразовых» задач – попробуйте снова
 - T4-like схема работы

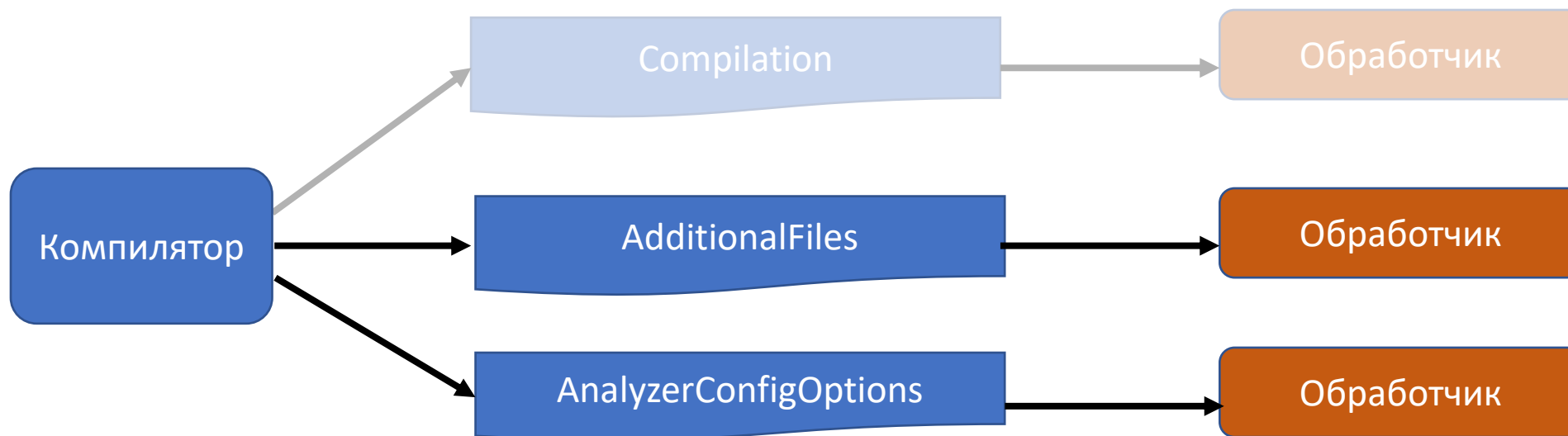
Оптимизация за счет сокращения зависимостей



Оптимизация за счет сокращения зависимостей



Оптимизация за счет сокращения зависимостей



А что если зависимости от Compilation не избежать?...

- CsWin32 generator
 - NativeMethods.txt

GetTickCount
IEnumDebugPropertyInfo
CreateFile
DISPLAYCONFIG_VIDEO_SIGNAL_INFO

А что если зависимости от Compilation не избежать?...

- CsWin32 generator

- NativeMethods.txt

```
GetTickCount  
IEnumDebugPropertyInfo  
CreateFile  
DISPLAYCONFIG_VIDEO_SIGNAL_INFO
```

- `Compilation.Options.AllowUnsafe`

- `Compilation.GetTypeByMetadataName(
 typeof(MemoryMarshal).FullName)
 ?.GetMembers("CreateSpan");`

Селекторы спешат на помощь!

```
var unsafeProvider = context.CompilationProvider.Select(  
    (comp, cancellationToken) => comp.Options.AllowUnsafe);
```

```
var createSpanProvider = context.CompilationProvider.Select(  
    (comp, cancellationToken) =>  
        comp.GetTypeByMetadataName(typeof(MemoryMarshal).FullName)  
            ?.GetMembers("CreateSpan"));
```

```
var compilationInfoProvider = unsafeProvider.Combine(createSpanProvider)  
    .Combine(...)  
    .Combine(...)
```

Схема работы новых генераторов

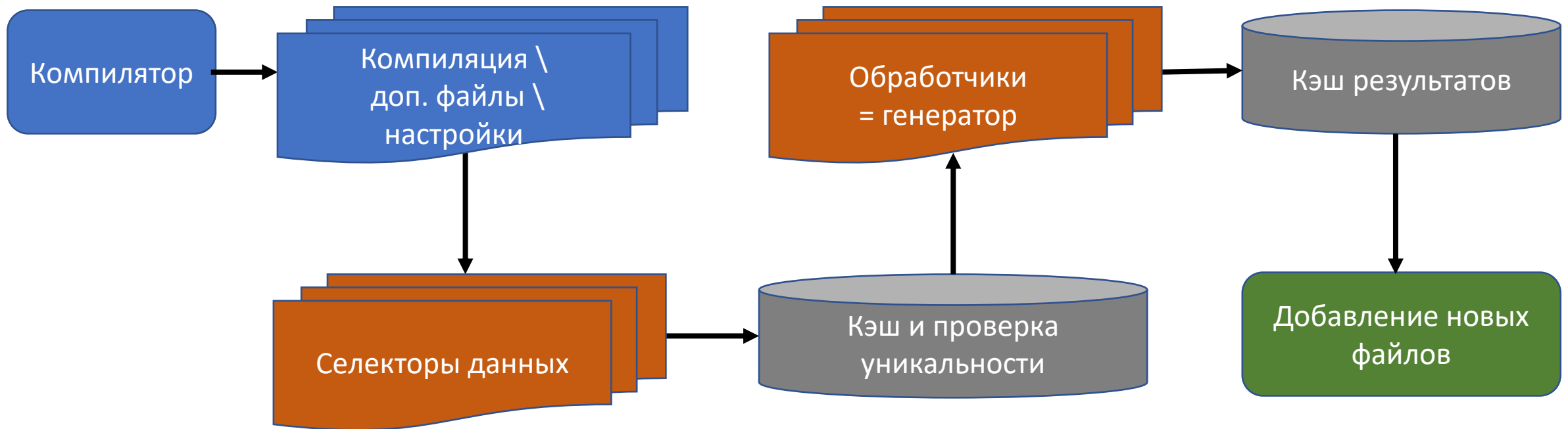
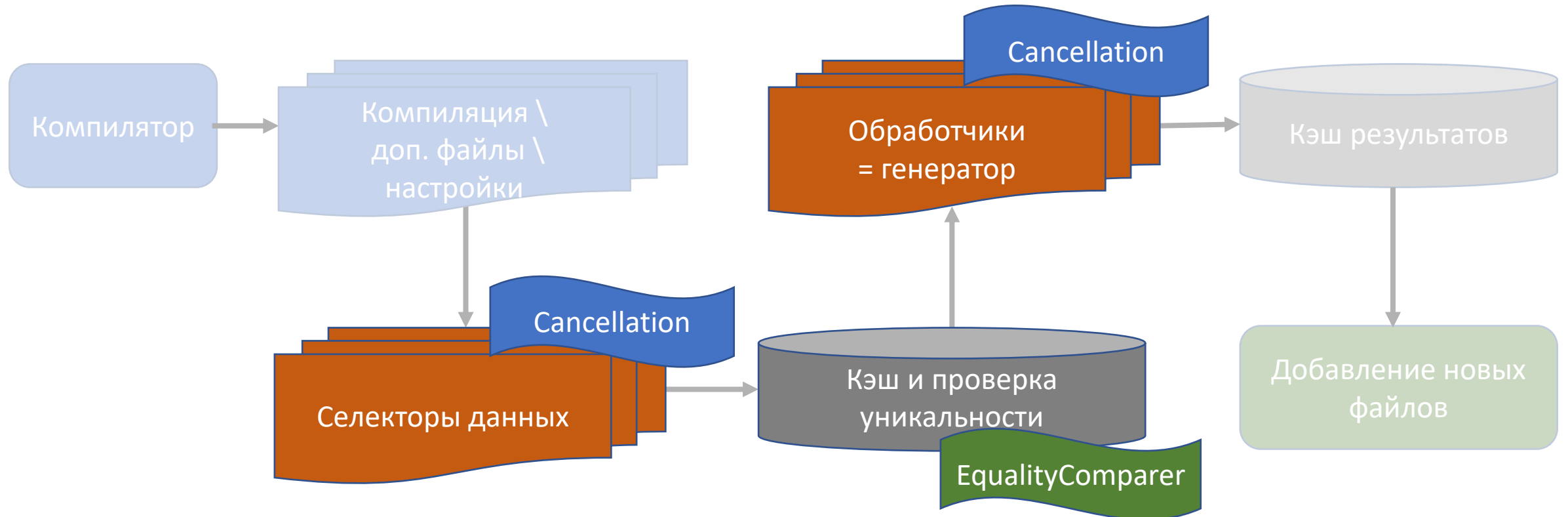


Схема работы новых генераторов



Если вашему генератору нужна информация о пользовательском проекте

- Определите какая информация вам нужна
- Создайте селекторы данных из объекта `Compilation`
 - Разрешен ли `Unsafe` код
 - Целевой фреймворк
 - Наличие нужных типов
 - И т.д.
- Подпишитесь на эти данные вместо всего объекта `Compilation`
- Перезапускаться будут только селекторы

Что если все таки нужен синтаксис?

Что если все таки нужен синтаксис?

- Реализации интерфейсов\методов
 - INotifyPropertyChanged, IEquatable, ToString, [DebuggerDisplay], ...
- Генерация сериализации без рефлексии
 - JsonSrcGen, .Net6 System.Text.Json, ...
- Контейнеры зависимостей
 - StrongInject, Jab, ...
- Генераторам нужны поля, свойства, интерфейсы, атрибуты типа

Как мы можем ускорить сам генератор?

- Но мы не хотим запускать генератор на каждую букву в
 - `Console.WriteLine("Hello world!");`
- Нет смысла запускать генератор для всех типов если поменялся один – генерация независимого кода
 - `INotifyPropertyChanged`
 - `IEquatable`
 - `Logging`
 - `ToString`

Как теперь работать с синтаксисом?

```
public partial readonly struct IncrementalGeneratorInitializationContext
{
    SyntaxValueProvider SyntaxProvider;

    IncrementalValueProvider<Compilation> CompilationProvider;
    IncrementalValueProvider<ParseOptions> ParseOptionsProvider;
    IncrementalValuesProvider<AdditionalText> AdditionalTextsProvider;
    IncrementalValueProvider<AnalyzerConfigOptionsProvider> AnalyzerConfigOptionsProvider;

    void RegisterSourceOutput<TSource>(
        IncrementalValueProvider<TSource> source,
        Action<SourceProductionContext, TSource> action);
}
```

Как теперь работать с синтаксисом?

- Создание провайдера синтаксиса в 2 шага
 - Фильтрация интересующих элементов
 - Селектор с доступом к семантике
- `SyntaxProvider` – не только про синтаксис
 - Вы будете получать вызовы при изменениях в других файлах
 - Они могут изменить семантическую модель

Зачем нужна семантическая модель?

```
public class AutoNotifyAttribute : Attribute { }
```

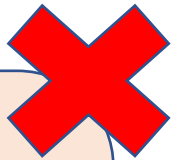
```
[AutoNotify] public class MyTargetType { }
```


Зачем нужна семантическая модель?

```
public class AutoNotifyAttribute : Attribute { }
```

```
[AutoNotify] public class MyTargetType { }
```

```
syntaxNode is TypeDeclarationSyntax typeDeclarationSyntax  
&& typeDeclarationSyntax.AttributeLists  
    .SelectMany(x => x.Attributes)  
    .Select(x => x.Name)  
    .OfType<SimpleNameSyntax>()  
    .Any(x => x.Identifier.Text == "AutoNotify")
```



Зачем нужна семантическая модель?

```
public class AutoNotifyAttribute : Attribute { }
```

```
[AutoNotify] public class MyTargetType { }
```

```
[Generator.AutoNotify] public class MyTargetType { }
```

```
[AutoNotifyAttribute] public class MyTargetType { }
```

Зачем нужна семантическая модель?

```
public class AutoNotifyAttribute : Attribute { }
```

```
[AutoNotify] public class MyTargetType { }
```

```
[Generator.AutoNotify] public class MyTargetType { }
```

```
[AutoNotifyAttribute] public class MyTargetType { }
```

```
using GenerateNotification = Generator.AutoNotifyAttribute;
```

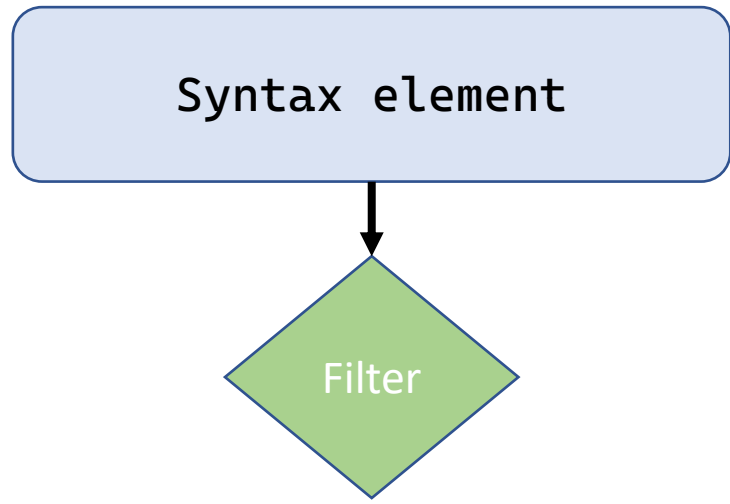
```
[GenerateNotification] public class MyTargetType { }
```

Новый интерфейс объединяет синтаксис и семантику

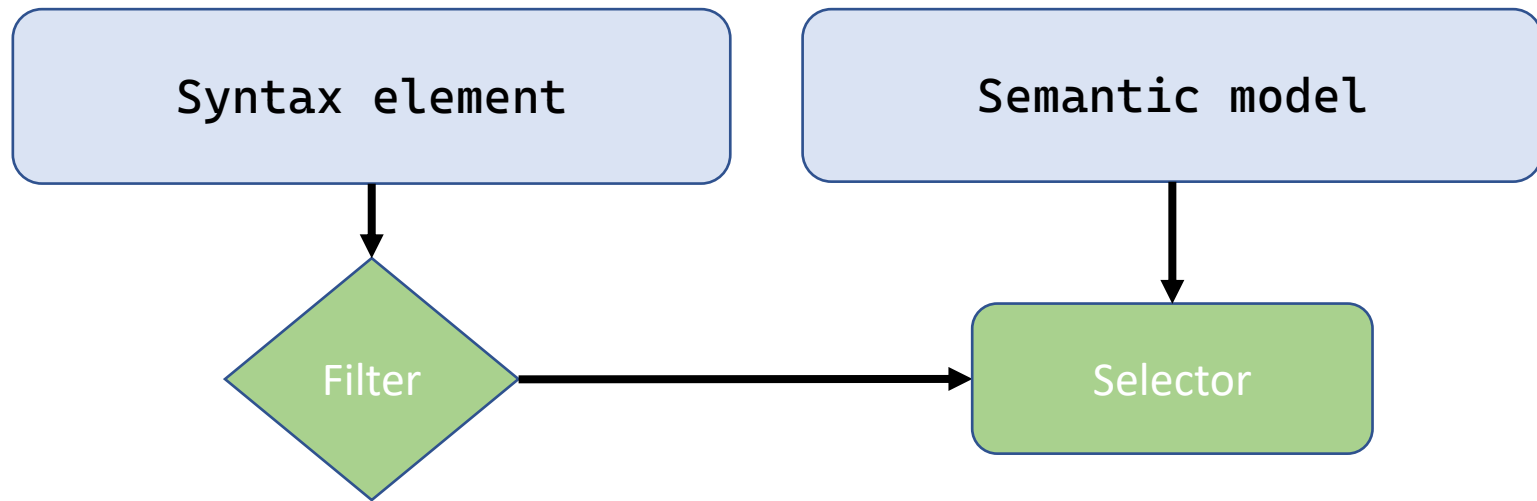
```
var syntaxProvider = context.SyntaxProvider.CreateSyntaxProvider(  
    predicate: (node, cancellationToken) => node is TypeDeclarationSyntax,  
  
    .Where(x => x != null);  
  
context.RegisterSourceOutput(syntaxProvider, (context, type) => ...);
```

Новый интерфейс объединяет синтаксис и семантику

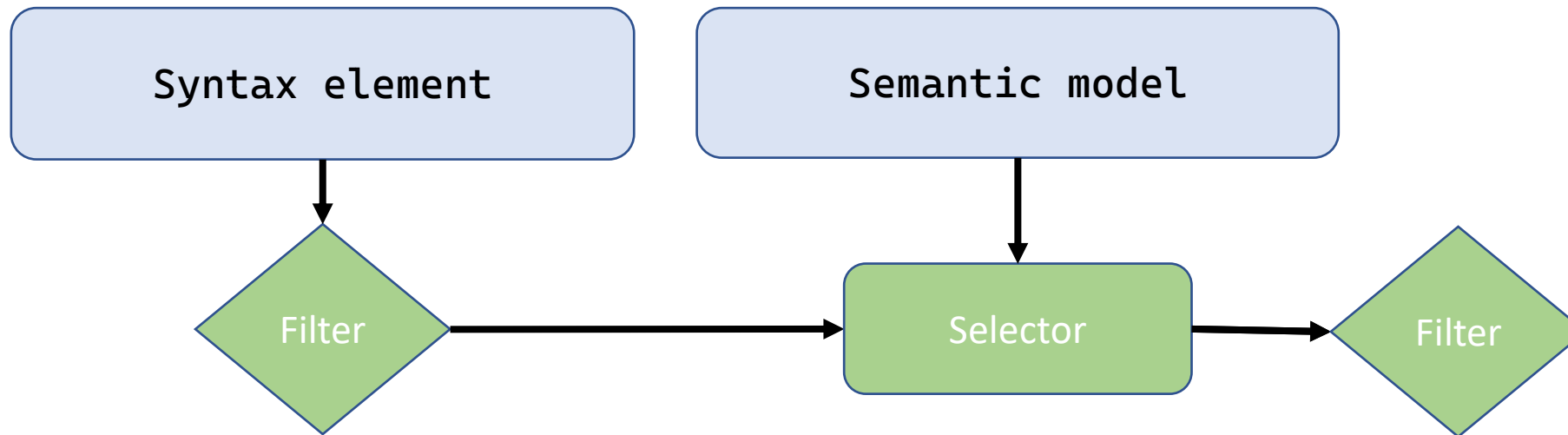
```
var syntaxProvider = context.SyntaxProvider.CreateSyntaxProvider(  
    predicate: (node, cancellationToken) => node is TypeDeclarationSyntax,  
    transform: (context, cancellationToken) =>  
    {  
        var node = context.Node;  
        var semanticModel = context.SemanticModel;  
        var type = (ITypeSymbol) semanticModel.GetDeclaredSymbol(node);  
        return HasRequiredAttributes(type, semanticModel) ? type : null;  
    })  
    .Where(x => x != null);  
  
context.RegisterSourceOutput(syntaxProvider, (context, type) => ...);
```



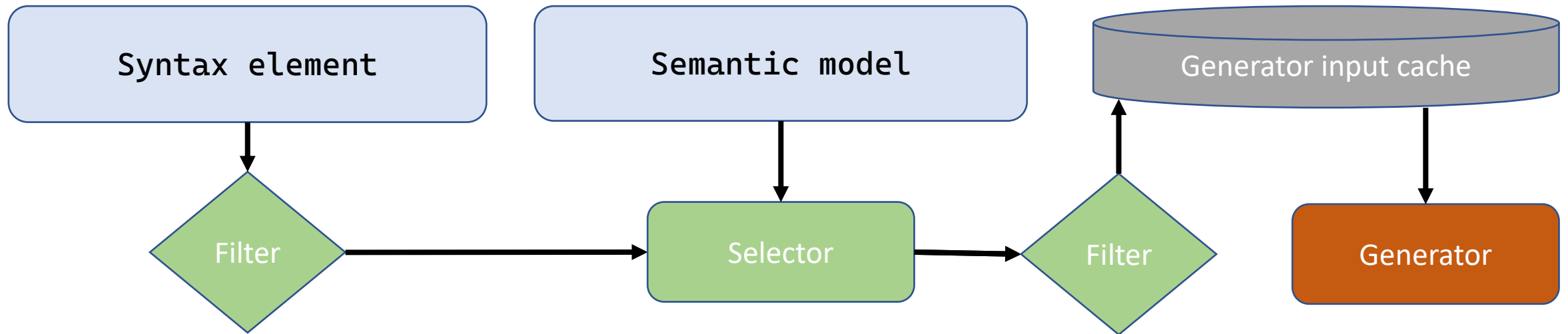
Filter:
x => x is TypeDeclarationSyntax



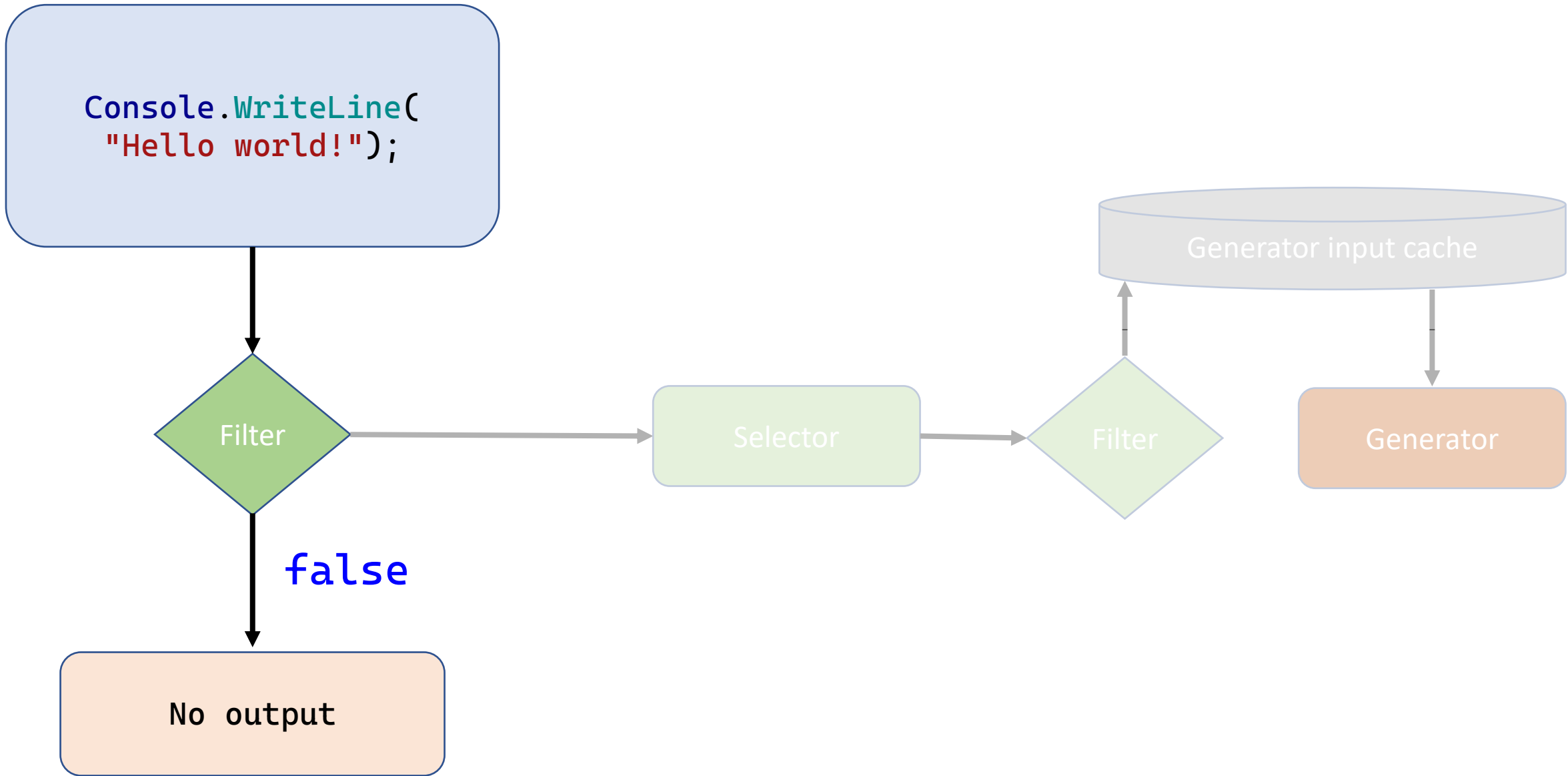
```
Selector:  
    var semanticModel = ctx.SemanticModel;  
    var type = semanticModel.GetDeclaredSymbol(ctx.Node);  
    return HasRequiredAttribute(type, semanticModel)  
        ? type  
        : null;
```

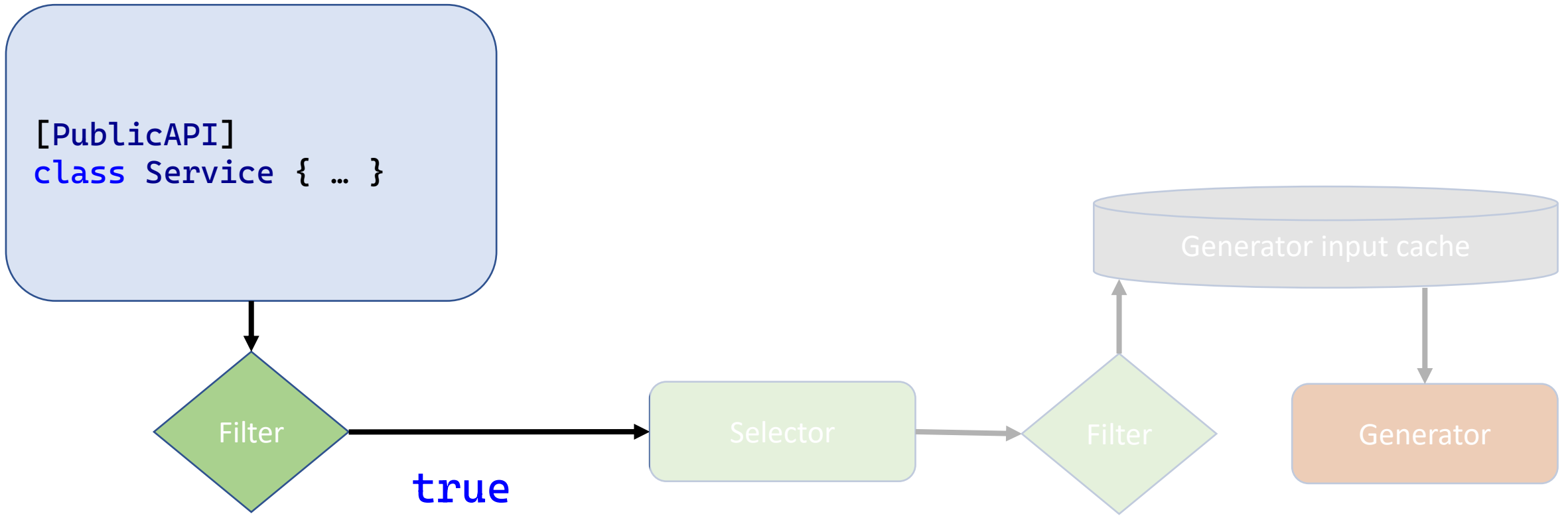


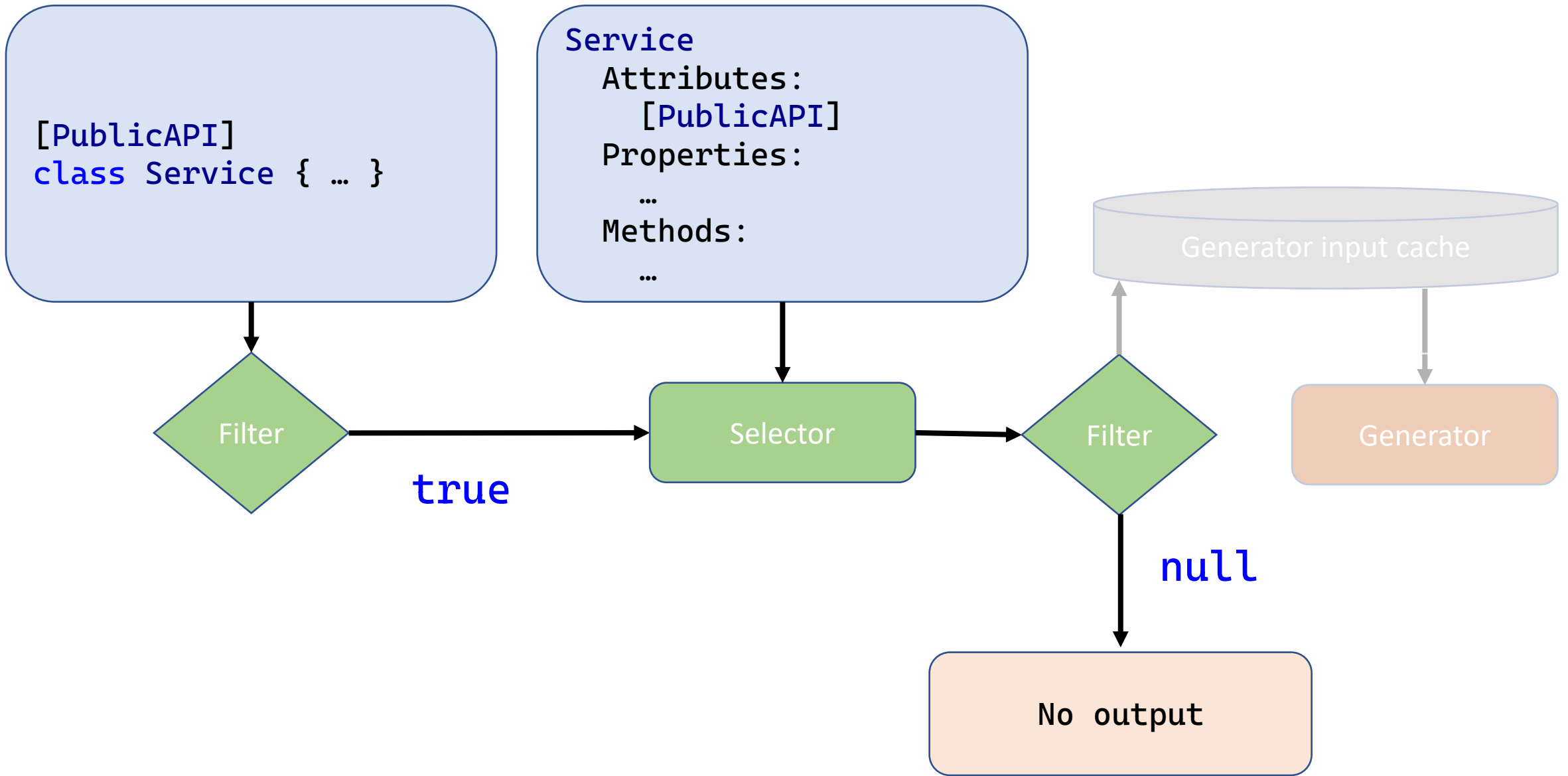
```
Filter:  
    .Where(x => x != null);
```

```
Generator:  
  (ctx, type) => ctx.AddSource(  
    $"{type.Name}.Notify.cs",  
    GeneratePropertyChanged(type))
```







```
[AutoNotify]
```

```
class Person
```

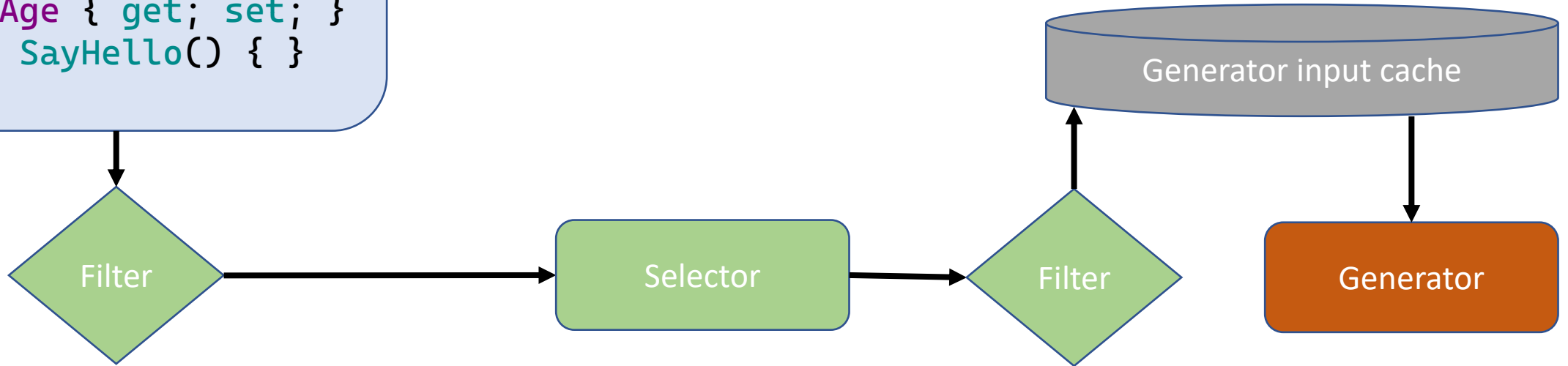
```
{
```

```
    string name;
```

```
    int Age { get; set; }
```

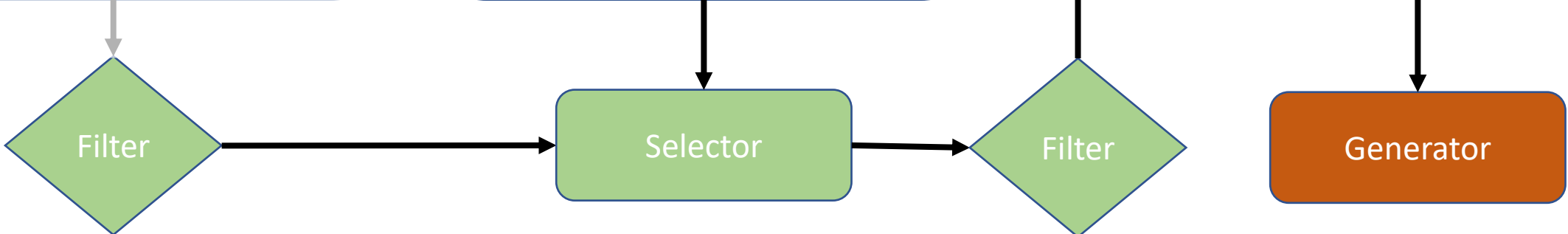
```
    void SayHello() { }
```

```
}
```



```
[AutoNotify]
class Person
{
  string name;
  int Age { get; set; }
  void SayHello() { }
}
```

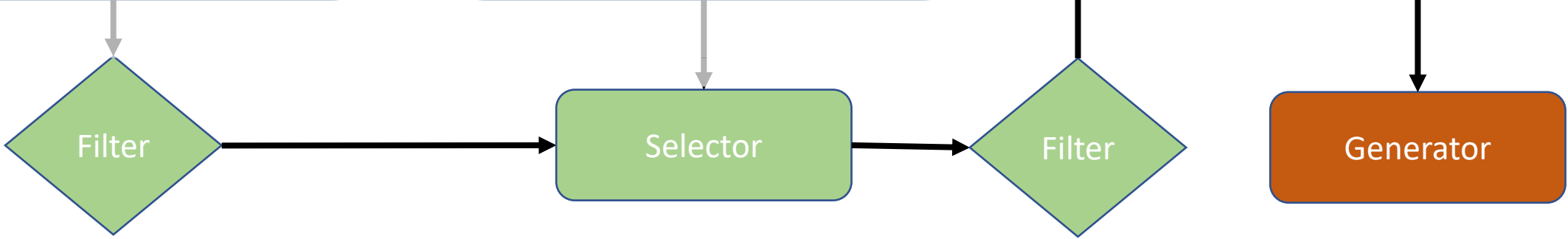
```
Person
Fields:
  string name;
Properties:
  int Age {get;set;}
Methods:
  SayHello();
```

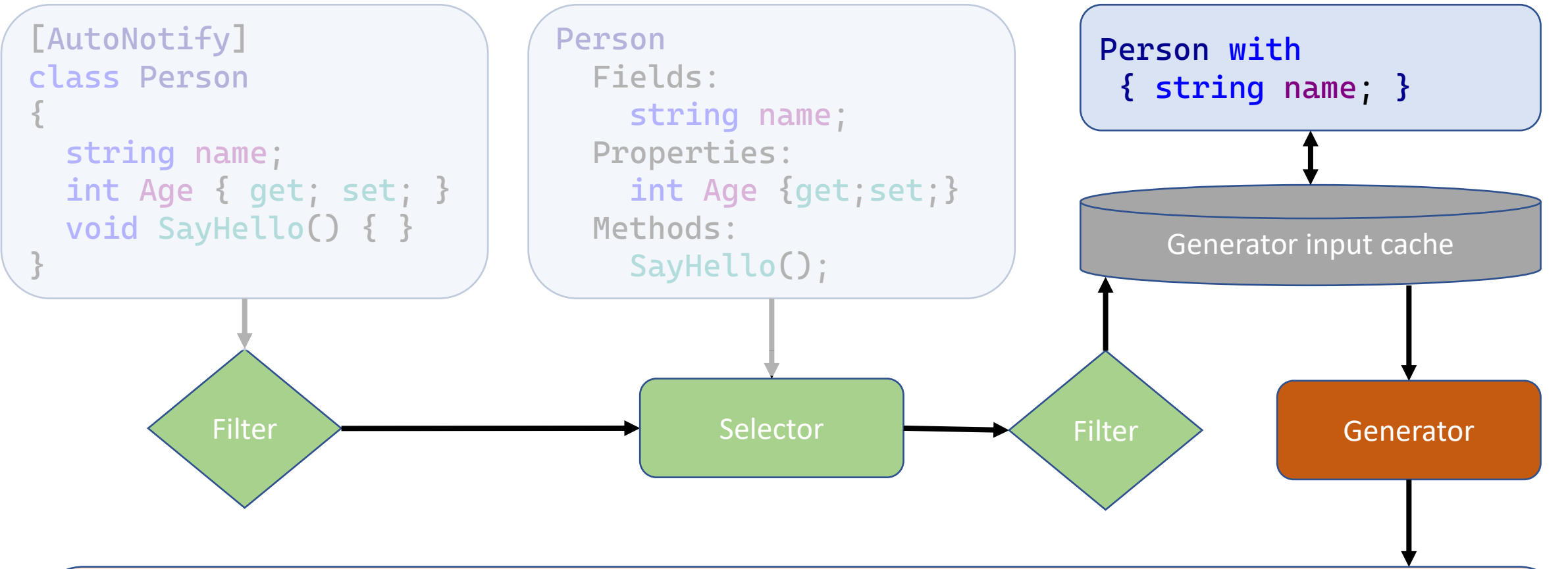


```
[AutoNotify]
class Person
{
  string name;
  int Age { get; set; }
  void SayHello() { }
}
```

```
Person
Fields:
  string name;
Properties:
  int Age {get;set;}
Methods:
  SayHello();
```

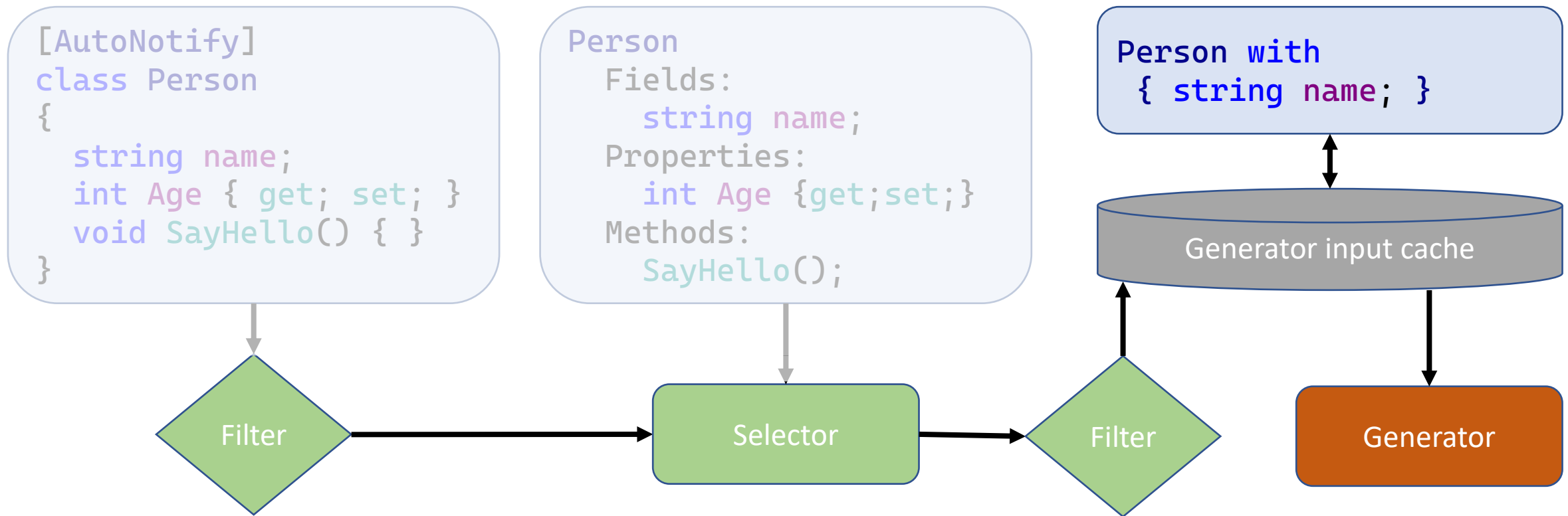
```
Person with
{ string name; }
```





```

partial class Person : INotifyPropertyChanged {
  public string Name {
    get => name;
    set { name = value; PropertyChanged?.Invoke(this, nameof(Name)); }
  }
  public event PropertyChangedEventHandler PropertyChanged;
}
  
```

- Генератор не будет перезапущен на изменения в других типах
- Фильтрация и селектор могут быть перезапущены
- Обращения к данным компилятора – обычно быстрые (5-10%)
 - В крайнем случае есть `CancellationToken`

Проблема перехода к семантике

- Подписка на синтаксис
 - Семантическая модель – дополнительные данные

Проблема перехода к семантике

- Подписка на синтаксис
 - Семантическая модель – дополнительные данные
- Несколько partial-частей = несколько нотификаций
 - Если вы перешли к типу вы обработаете его несколько раз

```
[AutoNotify] public partial class Person { ... }
```

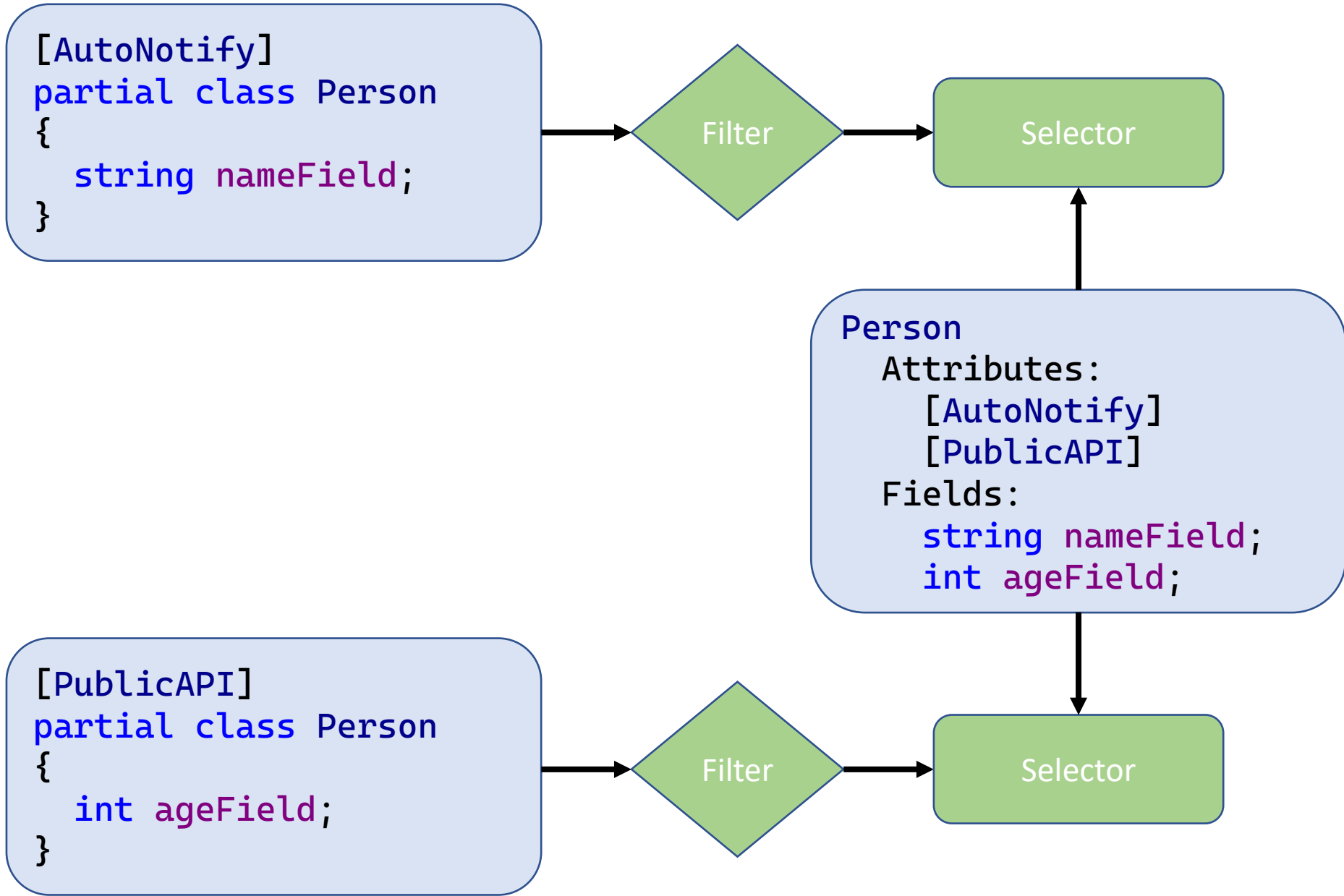
```
[PublicAPI] public partial class Person { ... }
```

```
[AutoNotify]
partial class Person
{
    string nameField;
}
```

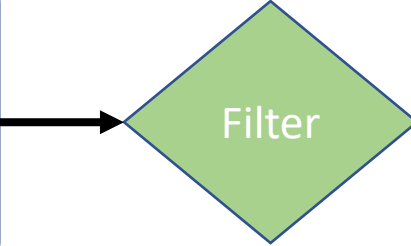


```
[PublicAPI]
partial class Person
{
    int ageField;
}
```



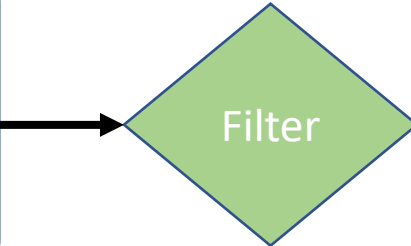


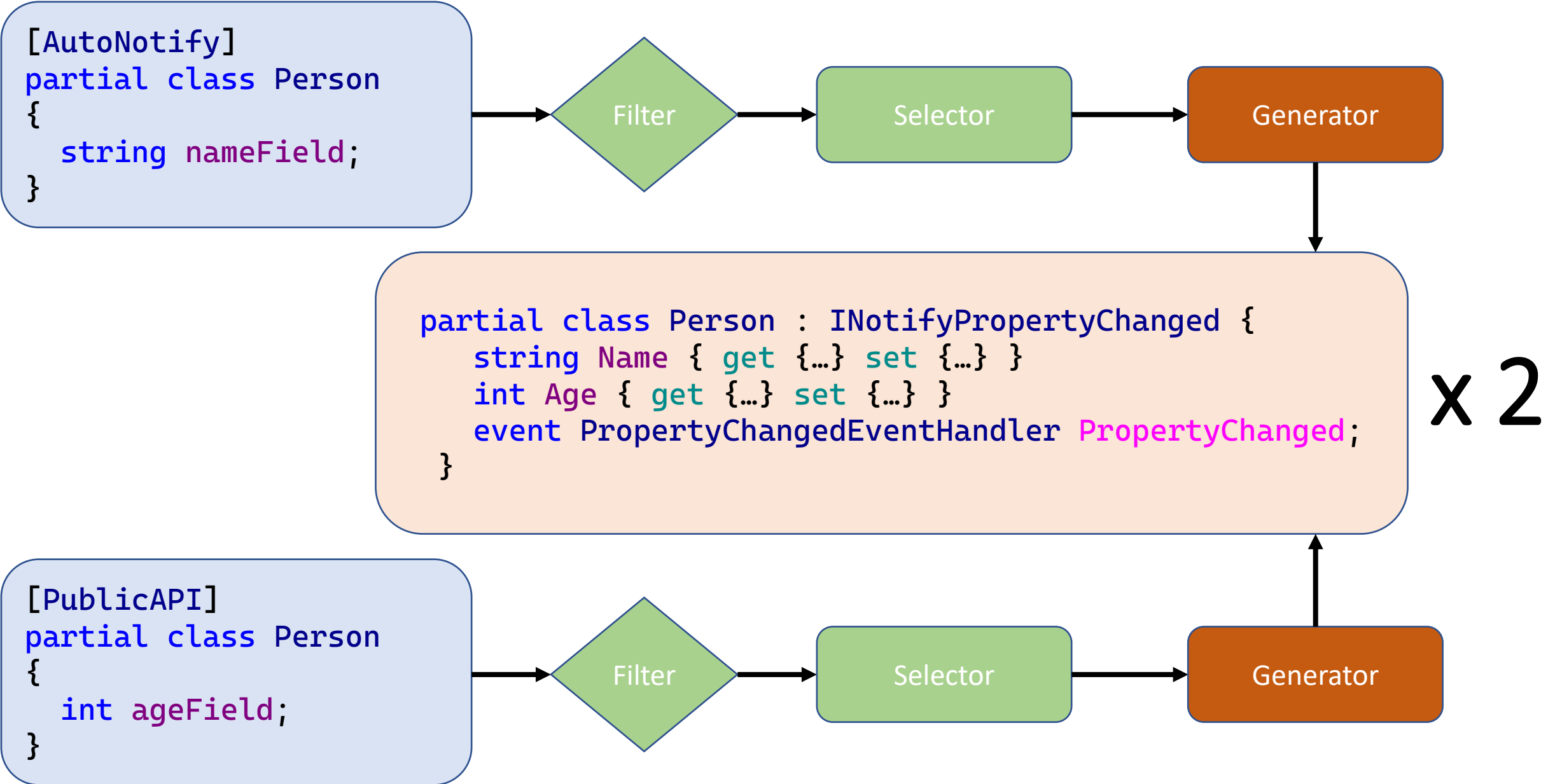
```
[AutoNotify]
partial class Person
{
    string nameField;
}
```



```
Selector:
    var semanticModel = ctx.SemanticModel;
    var type = semanticModel.GetDeclaredSymbol(ctx.Node);
    return HasRequiredAttribute(type, semanticModel)
        ? type
        : null;
```

```
[PublicAPI]
partial class Person
{
    int ageField;
}
```





Как тогда обрабатывать, например, все поля в классе?

- Сохраните синтаксический элемент в провайдере данных
- Обработайте только элементы объявленные в этой декларации
- Для нескольких partial-частей создайте несколько partial-классов
- В каждой части – только то что относится к исходной декларации

Сохраняем синтаксический элемент

```
var serviceProvider = context.SyntaxProvider.CreateSyntaxProvider(  
    predicate: (node, cancellationToken) => node is TypeDeclarationSyntax,  
    transform: (context, cancellationToken) =>  
    {  
        var node = context.Node;  
        var semanticModel = context.SemanticModel;  
        var type = (ITypeSymbol) semanticModel.GetDeclaredSymbol(node);  
        return HasRequiredAttributes(type, semanticModel) ? type : null;  
    })  
.Where(x => x != null);
```

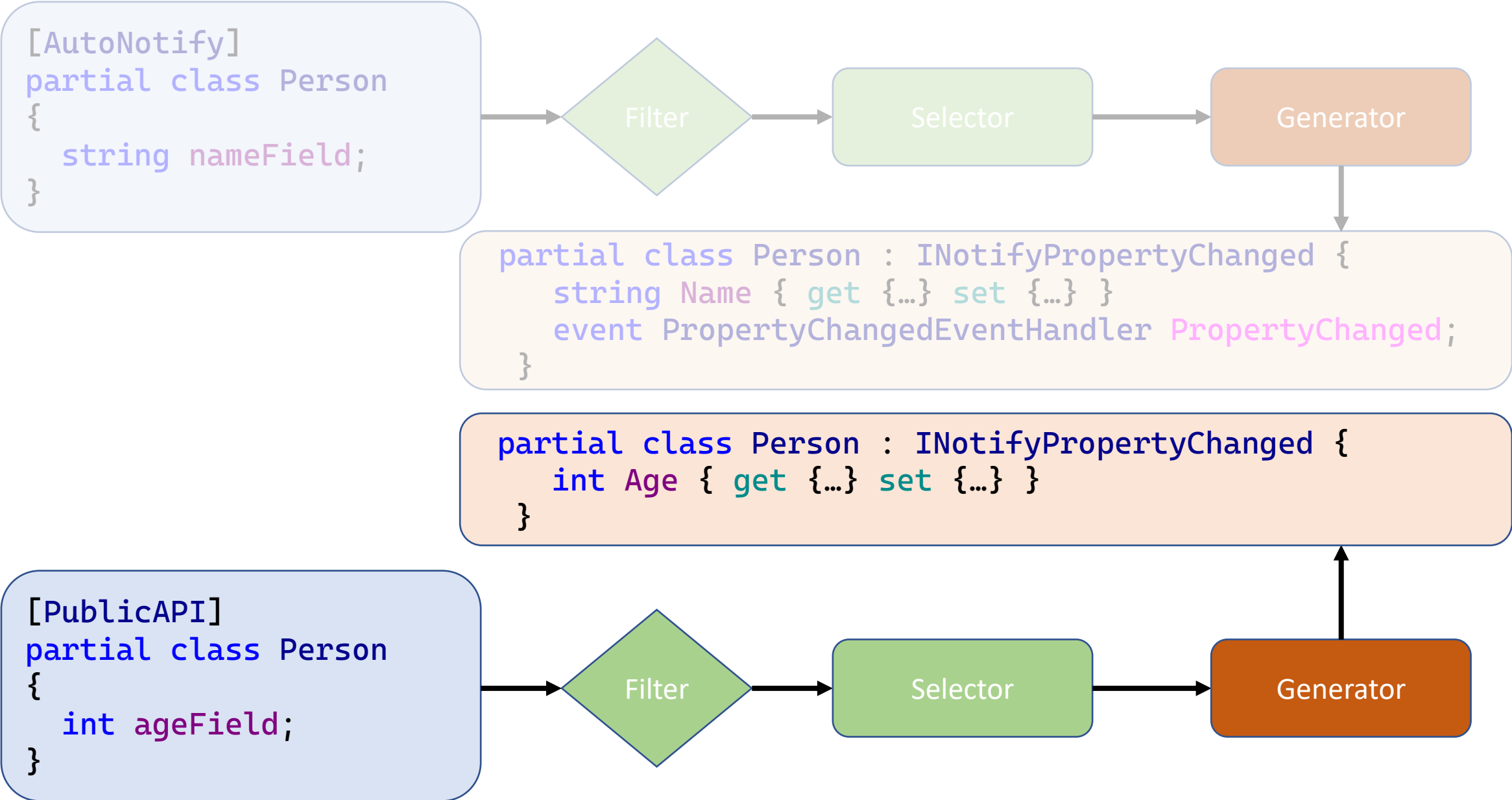
Сохраняем синтаксический элемент

```
var serviceProvider = context.SyntaxProvider.CreateSyntaxProvider(
    predicate: (node, cancellationToken) => node is TypeDeclarationSyntax,
    transform: (context, cancellationToken) =>
    {
        var node = context.Node;
        var semanticModel = context.SemanticModel;
        var type = (ITypeSymbol) semanticModel.GetDeclaredSymbol(node);
        return HasRequiredAttributes(type, semanticModel) ? type : null;
        return HasRequiredAttributes(type, semanticModel) ? node : null;
    })
    .Where(x => x != null);
```

```
[AutoNotify]
partial class Person
{
    string nameField;
}
```



```
partial class Person : INotifyPropertyChanged {
    string Name { get {...} set {...} }
    event PropertyChangedEventHandler PropertyChanged;
}
```



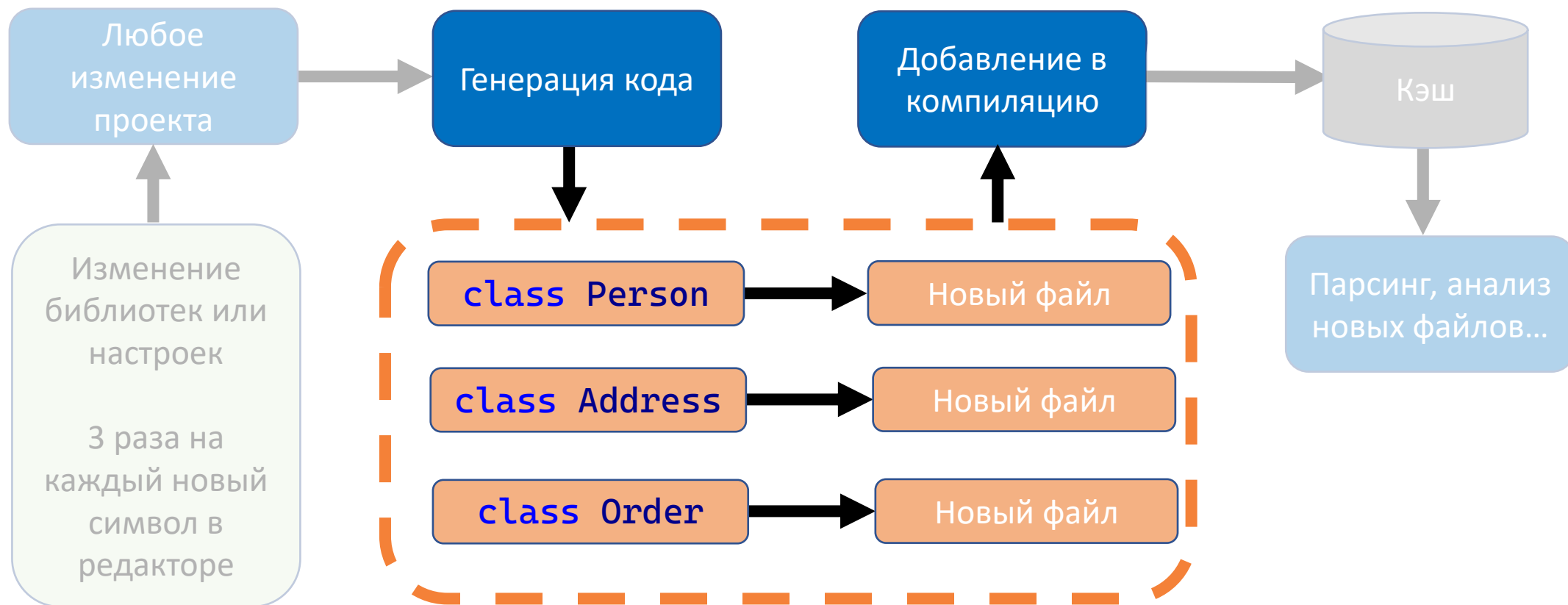
Преимущества новой ПОДПИСКИ НА СИНТАКСИС

- Кэширование исходных данных для генератора
 - Генератор перезапускается только если изменился нужный ему тип
 - Нет перезапусков генератора на набор стороннего кода
 - 60+ запусков генератора -> 0
 - `Console.WriteLine("Hello world!");`
 - 60+ запусков фильтра и селектора

Преимущества новой ПОДПИСКИ НА СИНТАКСИС

- Кэширование исходных данных для генератора
 - Генератор перезапускается только если изменился нужный ему тип
 - Нет перезапусков генератора на набор стороннего кода
 - 60+ запусков генератора -> 0
 - `Console.WriteLine("Hello world!");`
 - 60+ запусков фильтра и селектора
- Гранулярный запуск per type
 - Генератор обрабатывает один тип за раз
 - Если изменился один тип запустится только его обработчик
 - Возможность параллельного запуска
 - Поддержка прерываний на уровне схемы работы

Генераторы в IDE

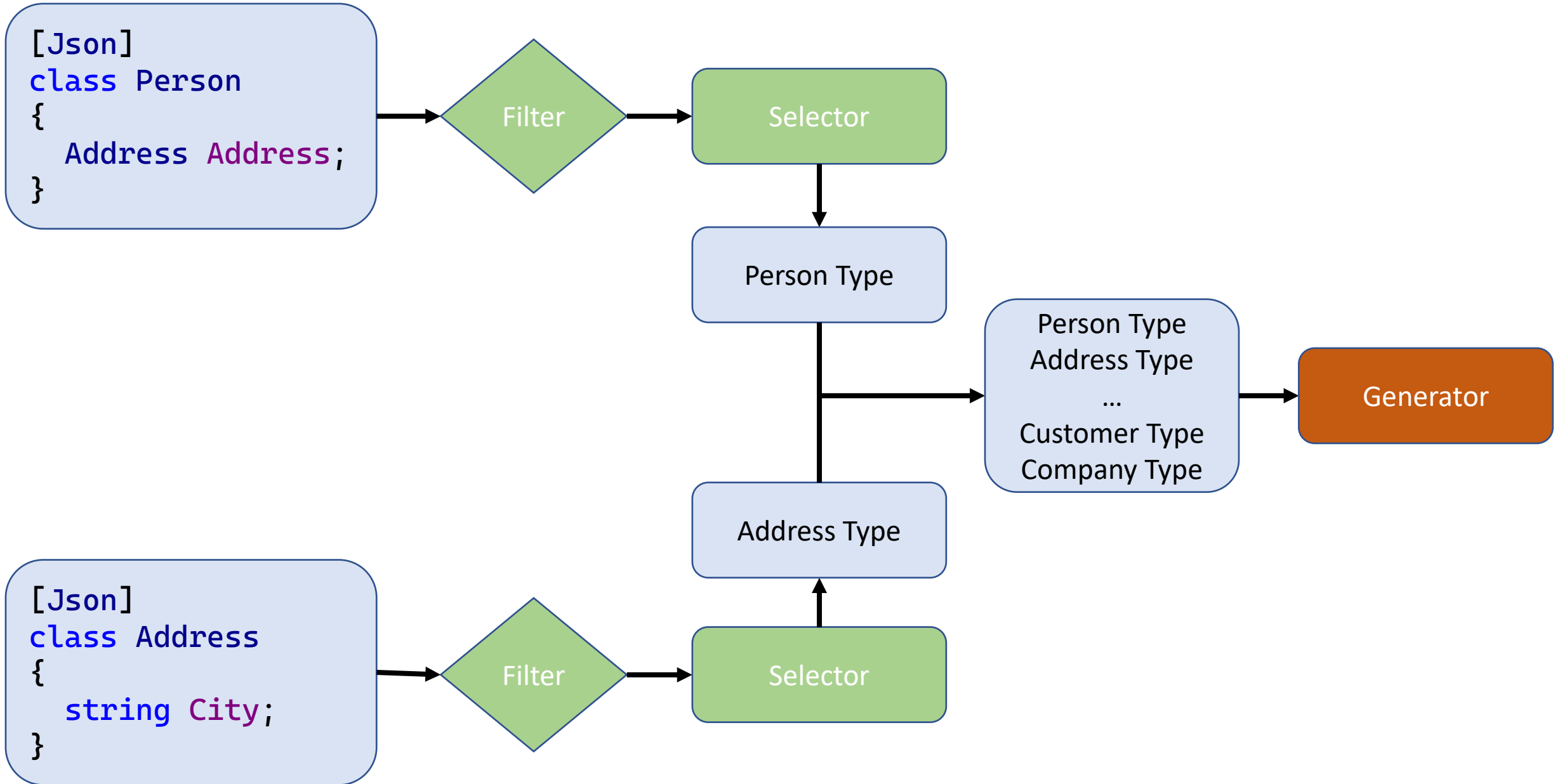


Что делать если нужны все типы в проекте?

- Например, JsonSrcGen - сериализатор
 - Нужно знать как сериализовать не только текущий тип но и его поля
 - Поля могут быть других типов и иметь собственную логику сериализации
- Если провайдер возвращает несколько объектов
 - `SyntaxProvider`
 - `AdditionalTextsProvider`
 - Вы можете вызвать `Collect()` и работать со всеми результатами вместе

Объединение нескольких элементов

```
var syntaxProvider = context.SyntaxProvider.CreateSyntaxProvider(  
    predicate: (node, cancellationToken) => { ... },  
    transform: (context, cancellationToken) => { ... })  
.Collect()  
.WithComparer(  
    (IEqualityComparer<ImmutableArray<ITypeSymbol>>) comparer);
```



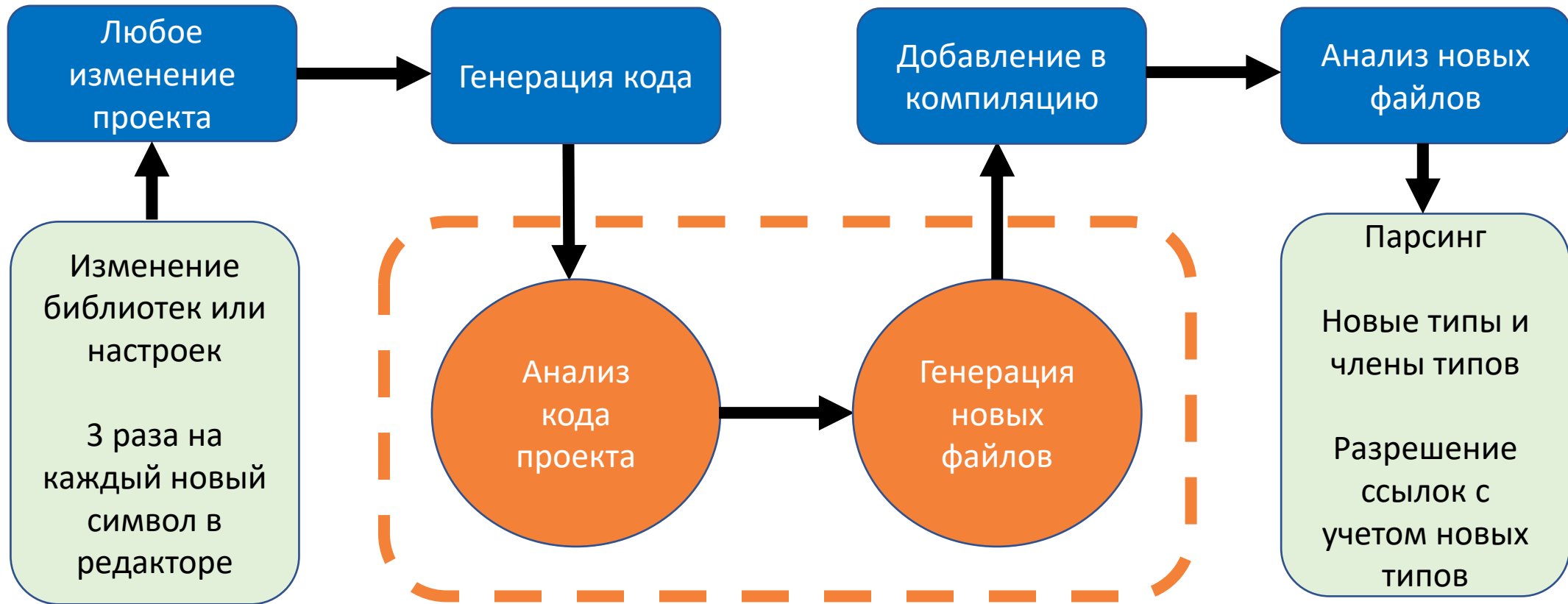
Если вам нужны все типы в проекте

- Соберите их при помощи вызова `.Collect()`
- Передайте компаратор!
 - Скорее всего вас интересует наличие элементов, а не их порядок
- При изменении любого типа будет перезапущен весь генератор

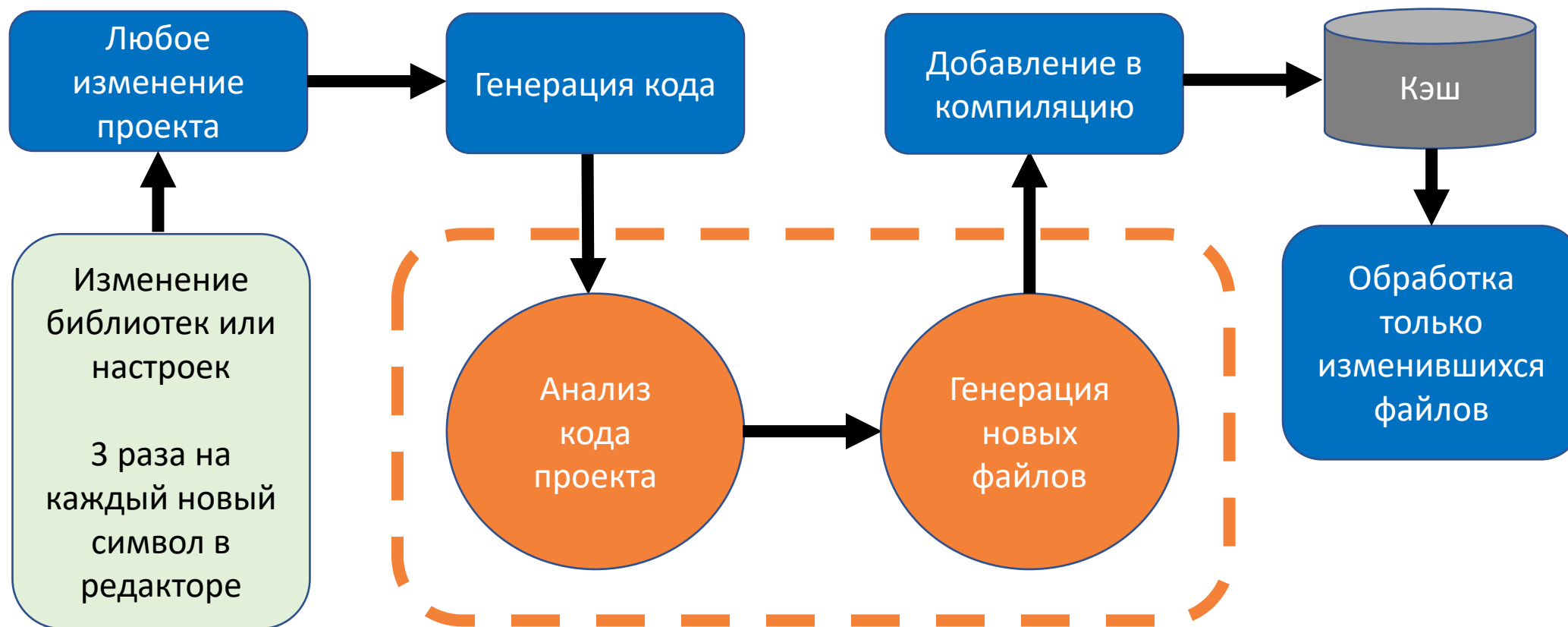
Если вам нужны все типы в проекте

- Соберите их при помощи вызова `.Collect()`
- Передайте компаратор!
 - Скорее всего вас интересует наличие элементов, а не их порядок
- При изменении любого типа будет перезапущен весь генератор
- Генератор станет быстрее
 - Нет перезапуска если изменилось что то стороннее – строчка в методе
 - Компилятор решает что нужно перезапустить
 - Любые его оптимизации не потребуют изменений генератора

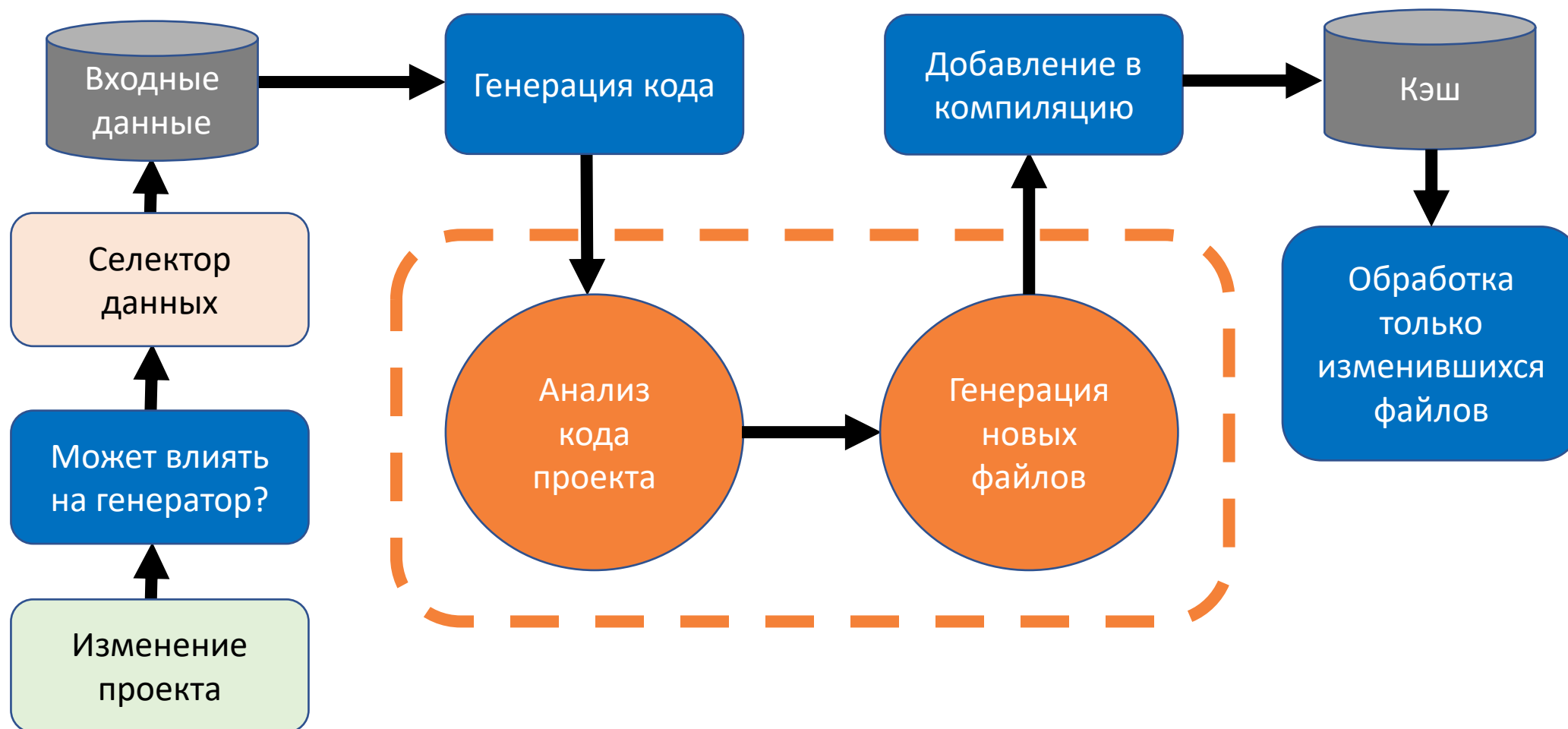
Генераторы в IDE



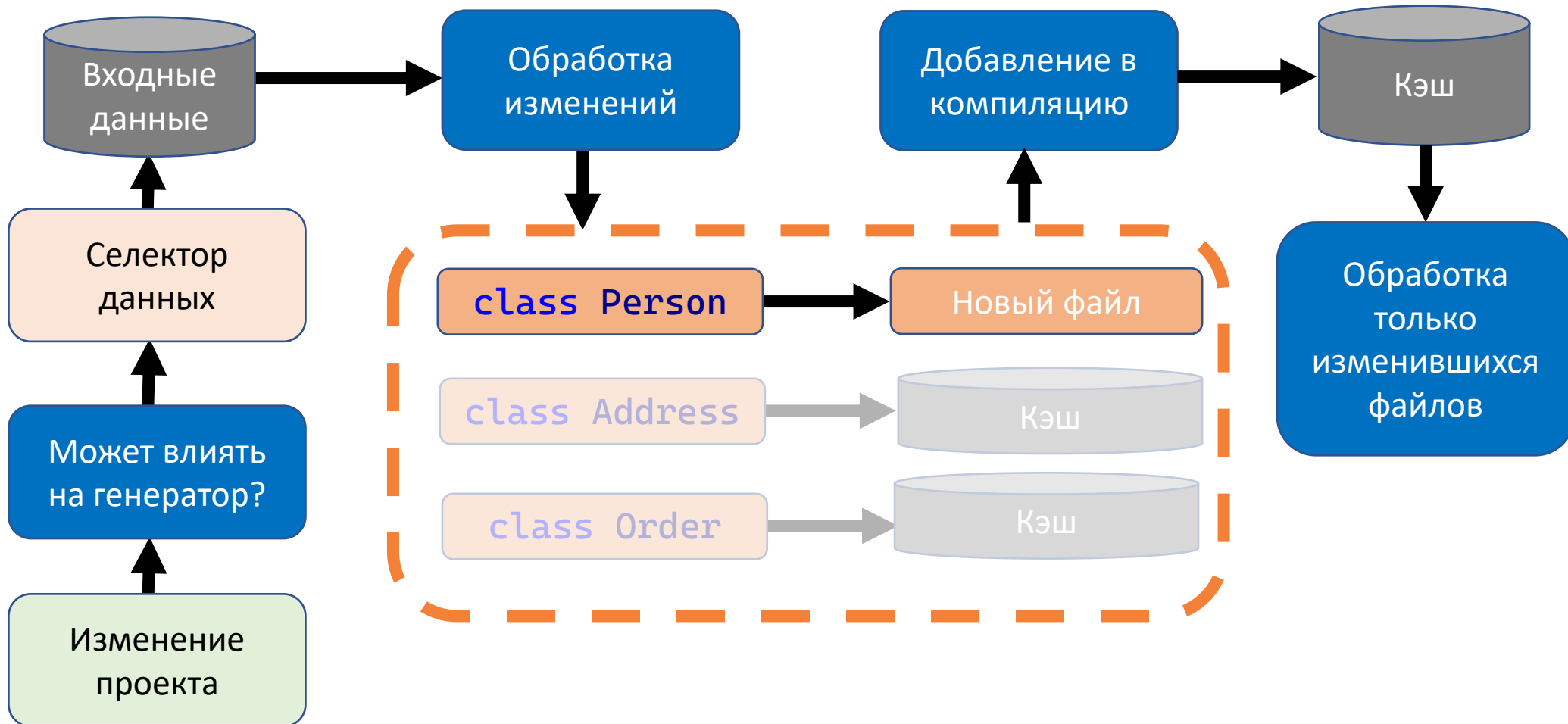
Генераторы в IDE



Генераторы в IDE



Генераторы в IDE



А что станет со старыми генераторами?

- Ничего
- Компилятор предоставит адаптер для старого интерфейса
- Адаптер объединяет все входные данные
- Старый генератор эквивалентен подписке на все источники данных сразу без фильтрации

```
goto demo;
```

Насколько новые генераторы быстрее?

- Визуальные задержки набора кода исчезли
- ThisAssembly
 - Полностью исчезли запуски во время набора кода
- JsonSrcGen
 - Единый метод Execute разбит на отдельные лямбды
 - Может исполняться многопоточно
 - Автоматическая поддержка прерываний
- Неизмеряемая часть нагрузки на компилятор
 - Парсинг новых файлов и их обработка случаются реже

А кофе когда пить??

[Generator]

```
public class CoffeeBreakGenerator : IIncrementalGenerator
{
    public void Initialize(IncrementalGeneratorInitializationContext context)
        => context.RegisterSourceOutput(
            context.ParseOptionsProvider.Select((x, _) => x.Language),
            (_, _) => Thread.Sleep(600_000));
}
```





xkcd.com/303

Если у вас есть генератор

- Перейдите на новый интерфейс
- Выигрыш производительности зависит от сценария
 - Генератор зависит только от настроек \ доп. файлов
 - Ваш генератор зависит от информации о целевом проекте
 - Ваш генератор зависит от синтаксиса
 - Ваш генератор зависит от всех типов в проекте
- Если есть статические зависимости
 - Используйте `RegisterPostInitializationOutput`
 - Поддерживается в старом интерфейсе

Если вы отказались от генераторов

- Из-за производительности
 - Перейдите на VS2022
 - Поддержите новый интерфейс
 - Попросите автора поддержать новый интерфейс
 - Можно предоставить разные реализации для «старого» и «нового» компиляторов – см. `System.Text.Json.SourceGeneration`
 -  bit.ly/3ABcrrb (Roslyn 4.0)
 -  bit.ly/2YI1HKU (Roslyn 3.11)
- Из-за безопасности
 - `EmitCompilerGeneratedFiles`

Спасибо за внимание



tessenr@gmail.com



github.com/TessenR



twitter.com/a_tessenr