.NET STANDARD:
THE ROUTE TO PLATFORM INDEPENDENCE

Kathleen Dollard

Microsoft

Microsoft is moving to an Open Source and cross platform world, and they've created .NET Standard as a super-highway to get you there.

With the .NET Standard 2.0 release, the specification includes most of the API's you depend on in the .NET full framework (.NET 4.n). Because .NET Standard, .NET Core, Xamarin and UWP are all embracing this standard, your class libraries can easily be cross-platform.

In this session, you'll learn more about the goals of .NET Standard and how it differs from PCLs (Portable Class Libraries).

You'll also see how to use the .NET Portability Analyzer to find any changes your app needs.

To show that .NET Standard is not actual magic, you'll learn a little about how redirects support binary compatibility.

And, you'll learn what happens when an API just doesn't make sense on a particular platform and other potential pitfalls.

You'll leave this session understanding why you care about .NET Standard and the path to move your applications to it.

- .NET Core
- .NET Framework
- .NET Full Framework
- .NET Standard
- .NET 5
- TFM
- WTF

- .NET Core
- .NET Framework
- .NET Full Framework
- .NET Standard
- .NET 5
- TFM

- .NET Core
- .NET Framework
- .NET Full Framework
- .NET Standard
- ~~.NET 5~~
- TFM

Questions?

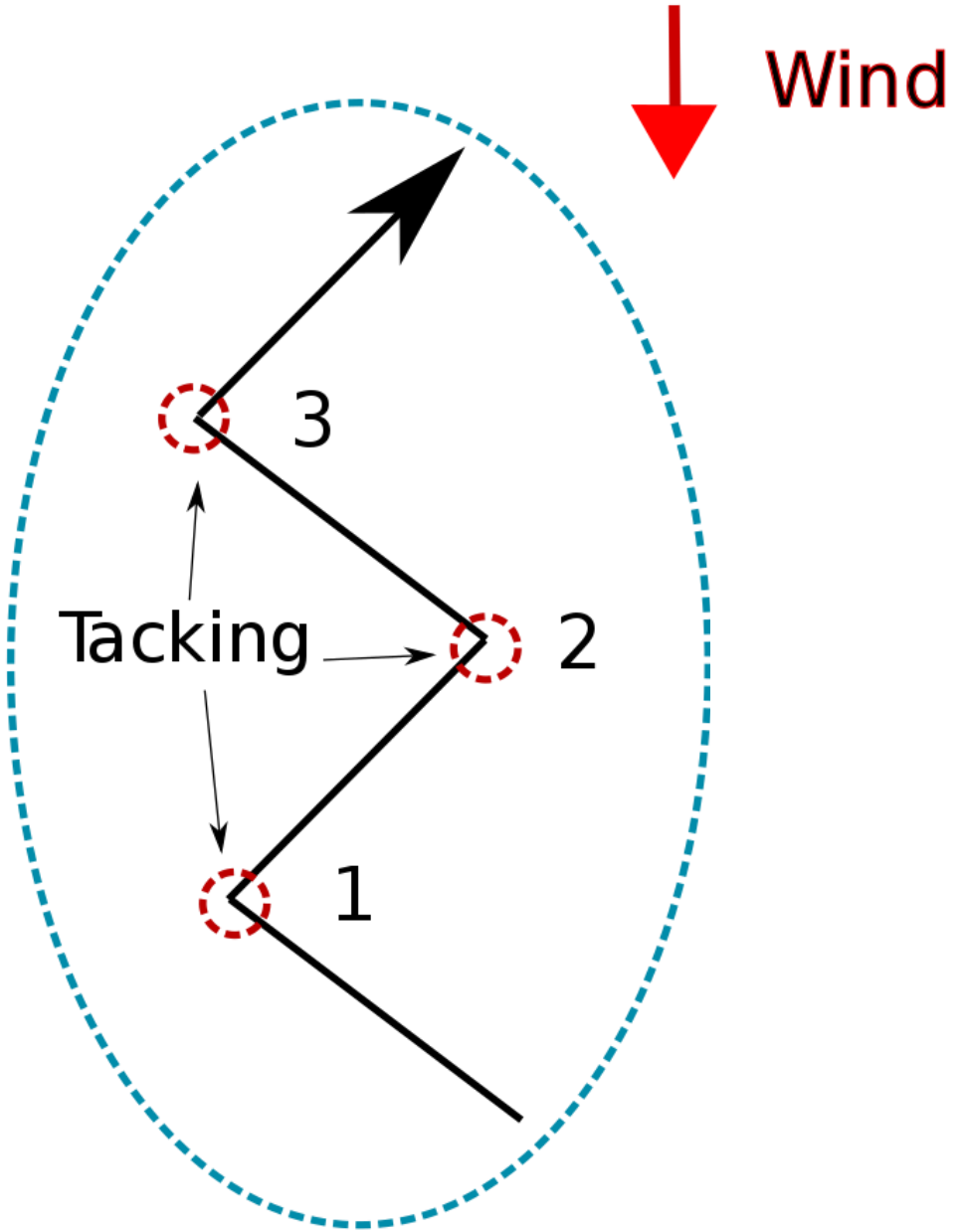
Benefits of .NET Core

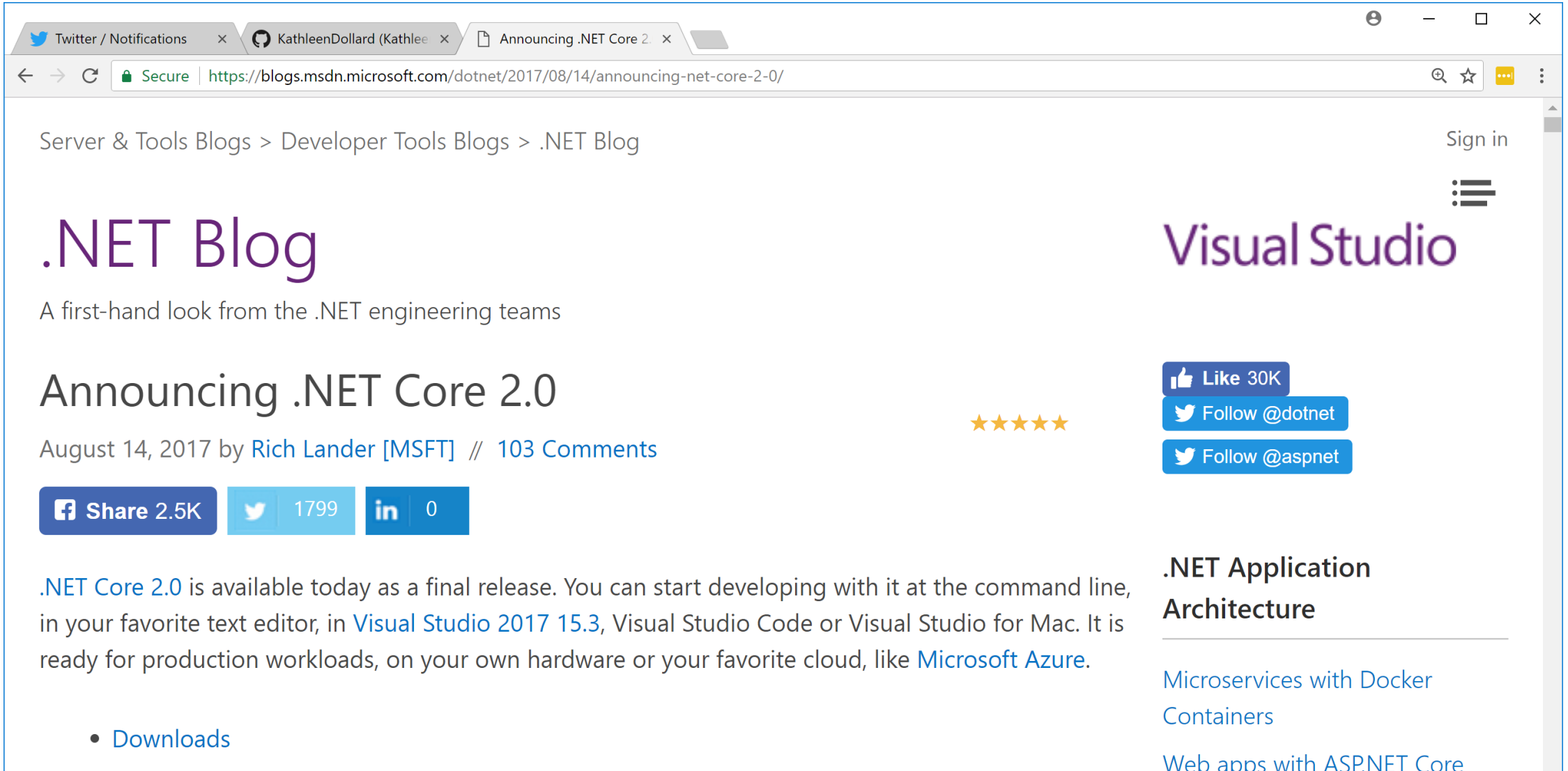
- Smaller and faster runtime
- Cross platform
- Side by side runtime installation
- Lighter development platform
- New stuff is on core, or there first

Benefits of .NET Full Framework

- Familiar
- Lots of apps written in it
- Supports WebForms, WinForms, WPF, WCF
- .NET Core changes made my boss nervous

One year ago,
looking into the .NET future…


Wind
3
Tacking
2
1


Twitter / Notifications
KathleenDollard (Kathlee
Announcing .NET Core 2.
Secure
https://blogs.msdn.microsoft.com/dotnet/2017/08/14/announcing-net-core-2-0/
Server & Tools Blogs > Developer Tools Blogs > .NET Blog
Sign in
.NET Blog
A first-hand look from the .NET engineering teams
Visual Studio
Announcing .NET Core 2.0
August 14, 2017 by Rich Lander [MSFT] // 103 Comments
Like 30K
Follow @dotnet
Follow @aspnet
Share 2.5K
1799
0
.NET Core 2.0 is available today as a final release. You can start developing with it at the command line, in your favorite text editor, in Visual Studio 2017 15.3, Visual Studio Code or Visual Studio for Mac. It is ready for production workloads, on your own hardware or your favorite cloud, like Microsoft Azure.
Downloads
.NET Application Architecture
Microservices with Docker Containers
Web apps with ASP.NET Core

- .NET Full Framework
  - The .NET framework we've been using for years and love
  - .NET 4.5.2…NET 4.7.1 (Fall Creator's Update)
- .NET Core
  - A new cross platform .NET
  - Doesn't do Windows stuff
  - .NET Core 2.0 with .NET Core 2.1 in preview
- .NET Standard
  - A ***standard*** for .NET
  - Definition for what makes something able to be .NET
  - .NET Standard 2.0, no 2.1 preview

- .NET Full Framework *implementation*
  - The .NET framework we've been using for years and love
  - .NET 4.5.2…NET 4.7.1 (Fall Creator's Update)
- .NET Core *implementation*
  - A new cross platform .NET that doesn't do Windows stuff
  - .NET Core 2.0 with .NET Core 2.1 in preview
- .NET Standard *standard*
  - A ***standard*** for .NET
  - Definition for what makes something able to be .NET
  - .NET Standard 2.0, no 2.1 preview

- .NET Full Framework *implementation*
  - The .NET framework we've been using for years and love
  - .NET 4.5.2…NET 4.7.1 (Fall Creator's Update)
- .NET Core *implementation*
  - A new cross platform .NET that doesn't do Windows stuff
  - .NET Core 2.0 with .NET Core 2.1 in preview
- .NET Standard *standard or specification*
  - A ***standard*** for .NET
  - Definition for what makes something able to be .NET
  - .NET Standard 2.0, no 2.1 preview

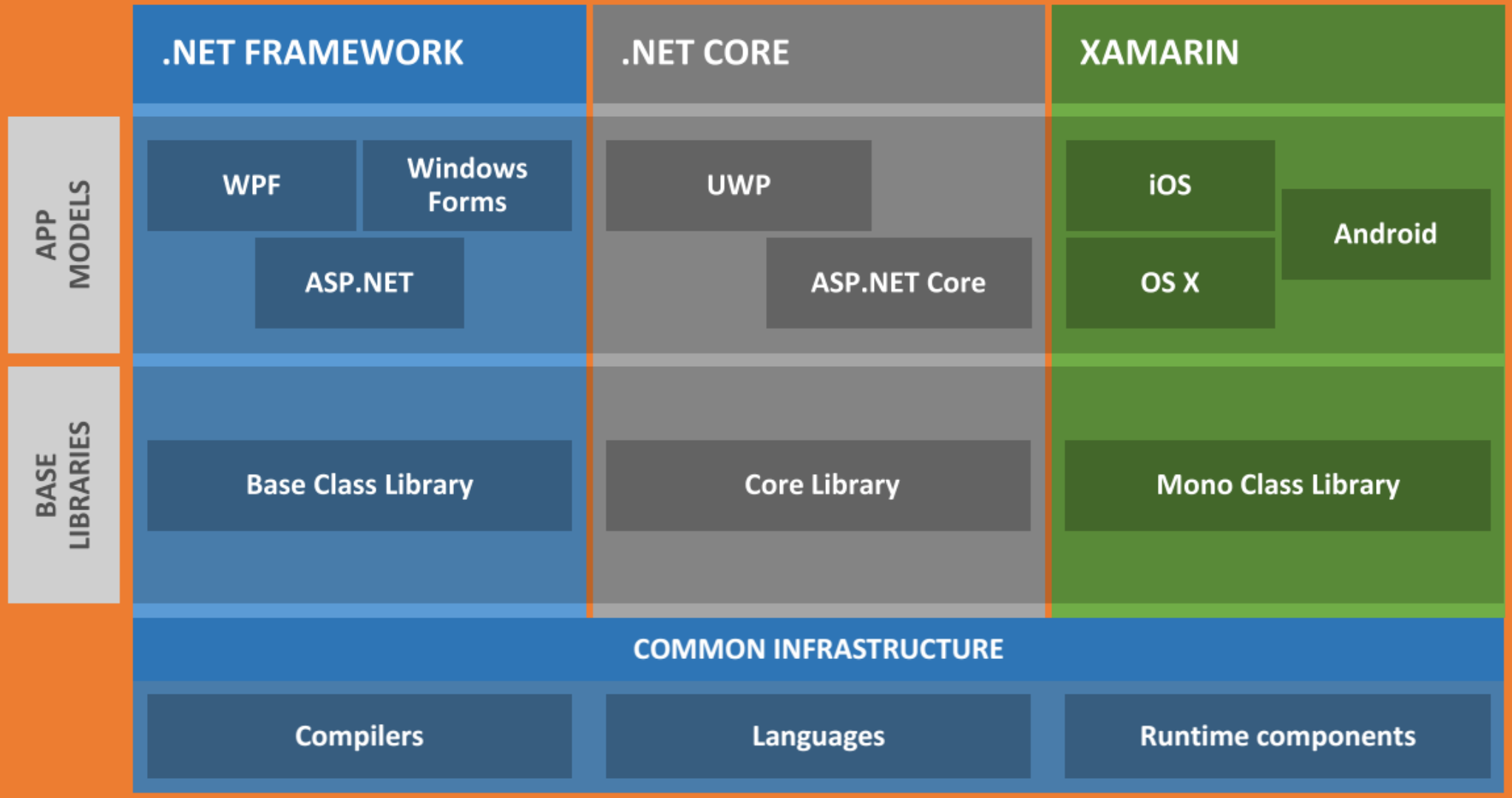
Other implementations

	Available API Set							
.NET Standard 2.0+ ▾	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core							1.0	2.0
.NET Framework		4.5	4.5.1	4.6				4.6.1
Mono							4.6	5.4
Xamarin.iOS							10.0	10.14
Xamarin.Android							7.0	8.0
Universal Windows Platform					10.0			10.0.16299
Windows		8.0	8.1					
Windows Phone			8.1					
Windows Phone Silverlight	8.0							

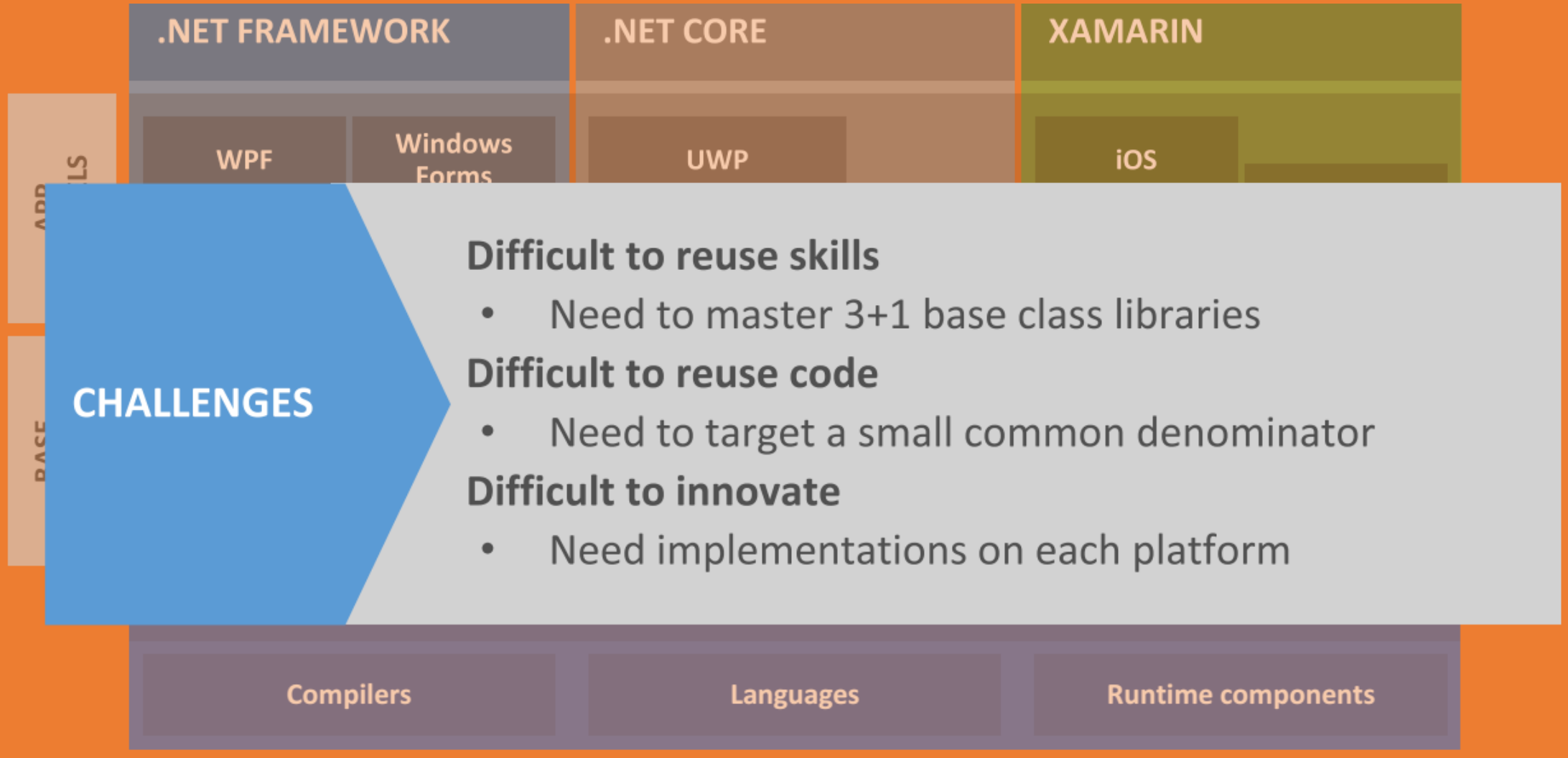
.NET Standard

www.aka.ms/immo

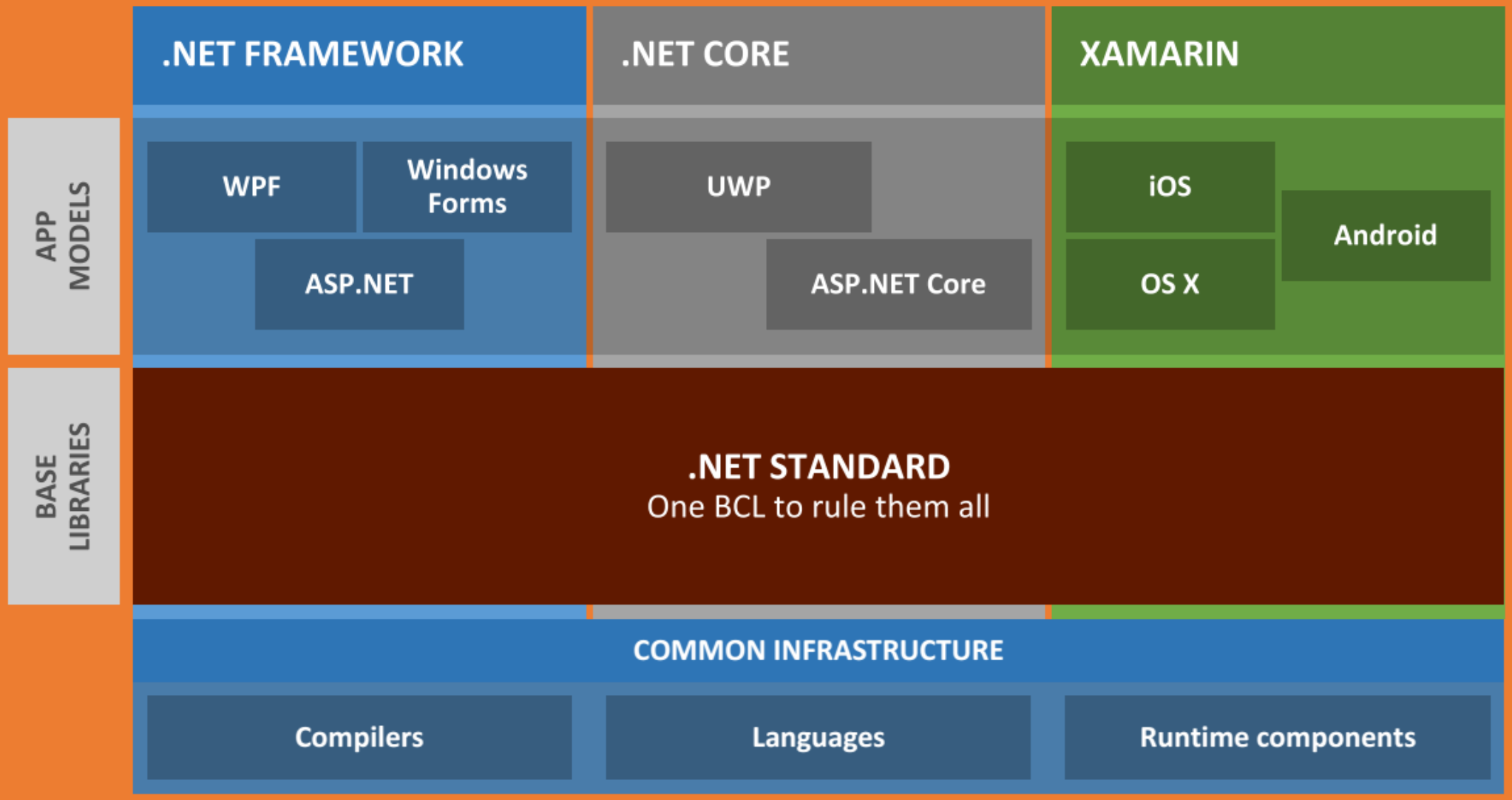
Without .NET Standard



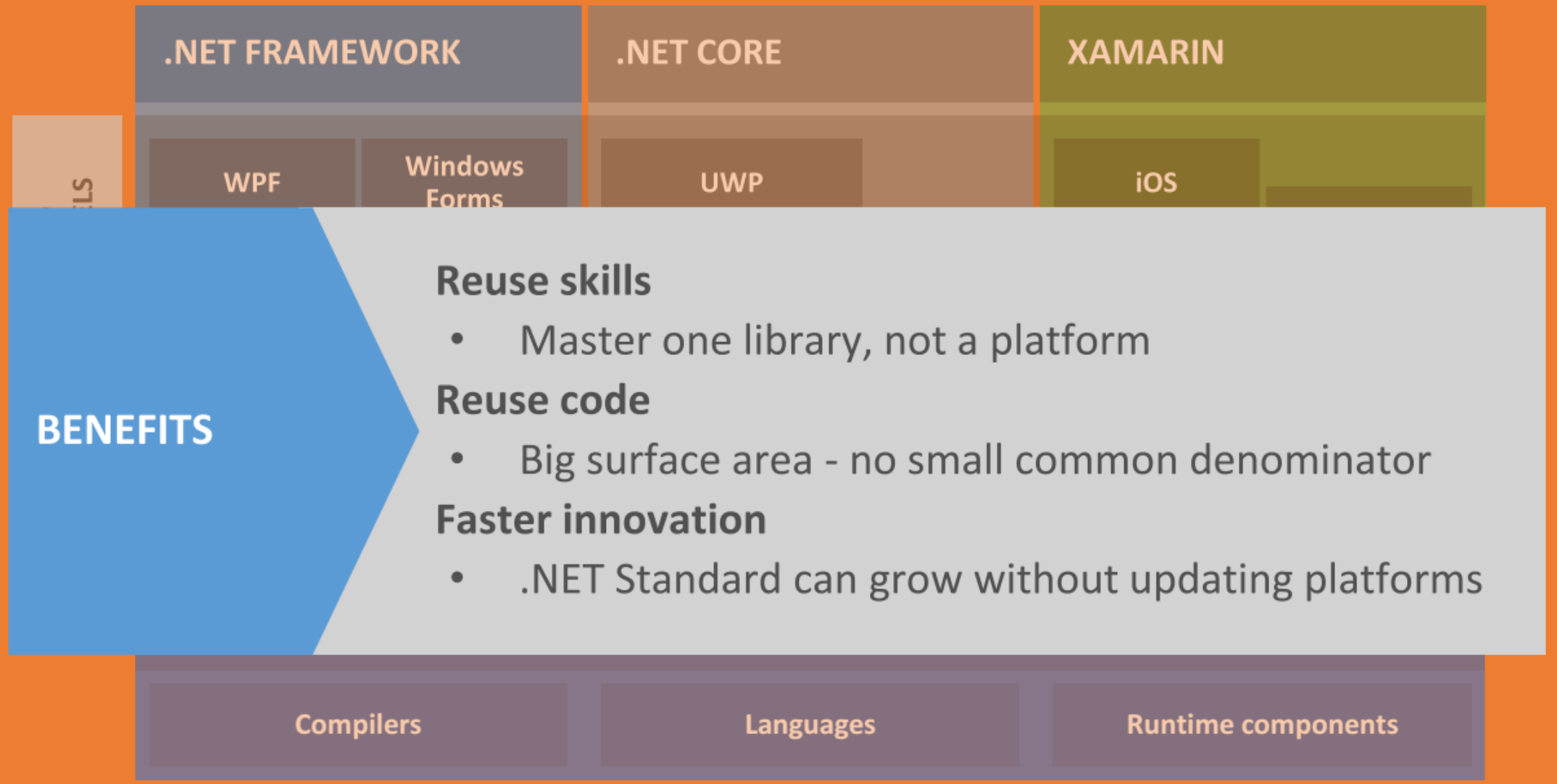
Without .NET Standard



With .NET Standard

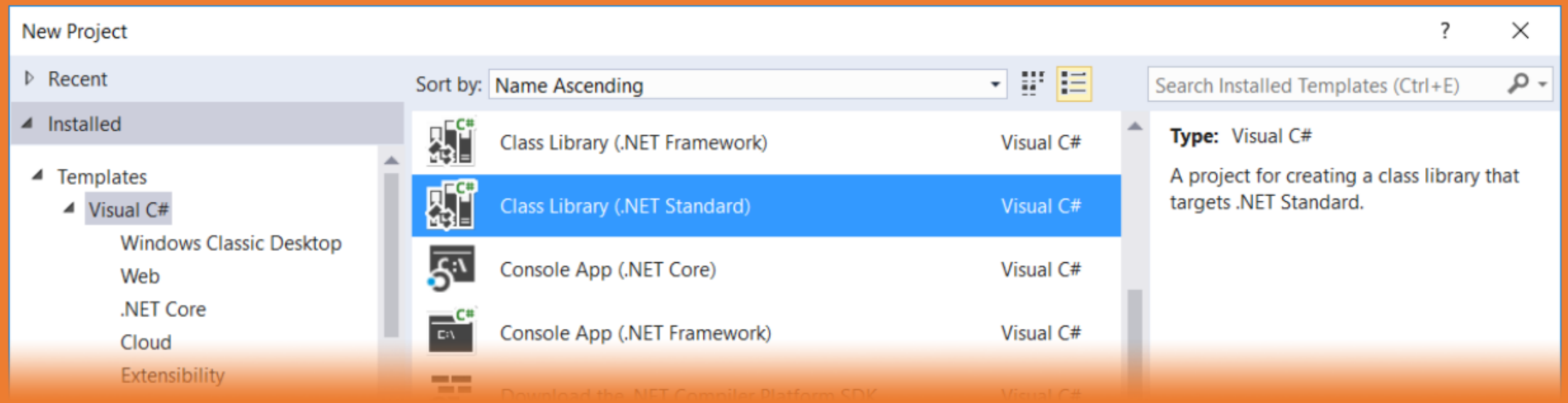


With .NET Standard



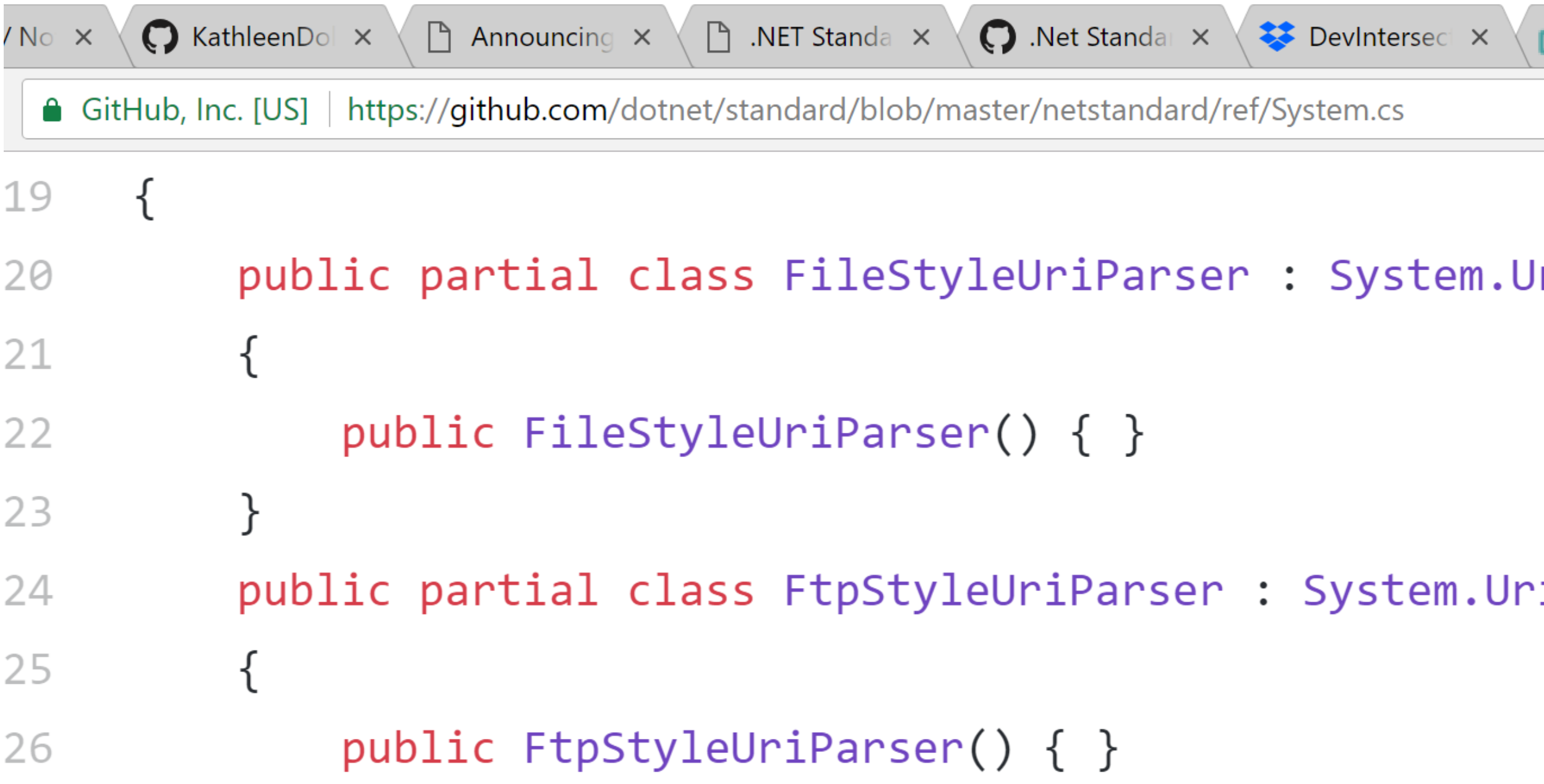
What is .NET Standard

- It is a specification
- It represents a set of APIs all .NET Standard platforms implement
- If you're class libraries just use Standard, they conform to it
  - .NET Core supplies a host letting you can .NET Standard libraries



So .NET Standard
is a standard,
a document
written in code

```
19  {
20      public partial class FileStyleUriParser : System.UriParser
21      {
22          public FileStyleUriParser() { }
23      }
24      public partial class FtpStyleUriParser : System.UriParser
25      {
26          public FtpStyleUriParser() { }
27      }
28      public partial class GenericUriParser : System.UriParser
29      {
30          public GenericUriParser(System.GenericUriParserOptions options) { }
31      }
32      [System.FlagsAttribute]
33      public enum GenericUriParserOptions
34      {
35          AllowEmptyAuthority = 2,
36          Default = 0,
37          DontCompressPath = 128,
38          DontConvertPathBackslashes = 64,
39          DontUnescapePathDotsAndSlashes = 256,
40          GenericAuthority = 1,
41          Idn = 512
```


/ No
KathleenDo
Announcing
.NET Standa
.Net Standa
DevIntersec
GitHub, Inc. [US]
https://github.com/dotnet/standard/blob/master/netstandard/ref/System.cs
19 {
20 public partial class FileStyleUriParser : System.U
21 {
22 public FileStyleUriParser() { }
23 }
24 public partial class FtpStyleUriParser : System.Ur
25 {
26 public FtpStyleUriParser() { }

Wat!
BINARY
compat

```
19  {
20      public partial class FileStyleUriParser : System.UriParser
21      {
22          public FileStyleUriParser() { }
23      }
24      public partial class FtpStyleUriParser : System.UriParser
25      {
26          public FtpStyleUriParser() { }
27      }
28      public partial class GenericUriParser : System.UriParser
29      {
30          public GenericUriParser(System.GenericUriParserOptions options) { }
31      }
32      [System.FlagsAttribute]
33      public enum GenericUriParserOptions
34      {
35          AllowEmptyAuthority = 2,
36          Default = 0,
37          DontCompressPath = 128,
38          DontConvertPathBackslashes = 64,
39          DontUnescapePathDotsAndSlashes = 256,
40          GenericAuthority = 1,
41          Idn = 512
```

How does .NET Standard work?

.NET Standard is represented by

- The NuGet package **NetStandard.Library** which contains
- The reference assembly **netstandard.dll**

At build time

- .NET Standard bridges references to existing .NET Framework and PCL assemblies via type forwarding

At runtime

- Each platform provides an implementation for netstandard.dll that type forwards to its implementation

I thought Portable Class Libraries were supposed to do handle platforms?

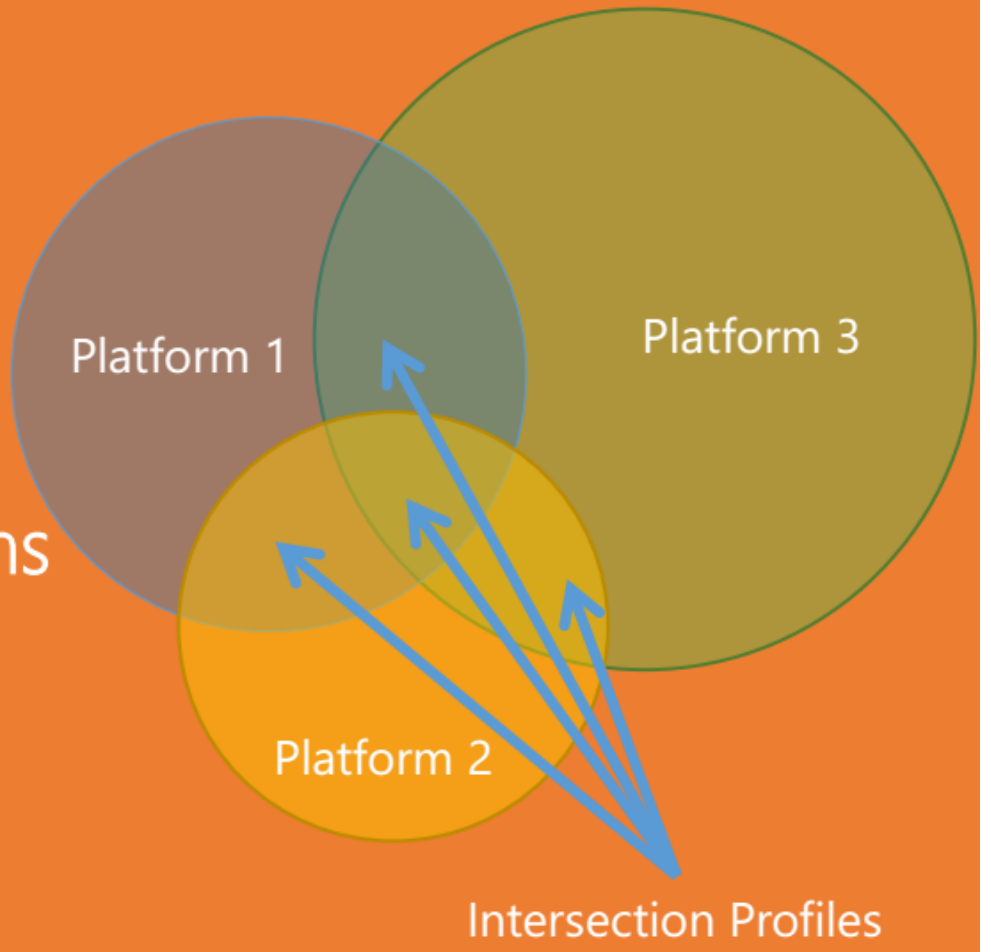
Difference to Portable Class Libraries (PCL)

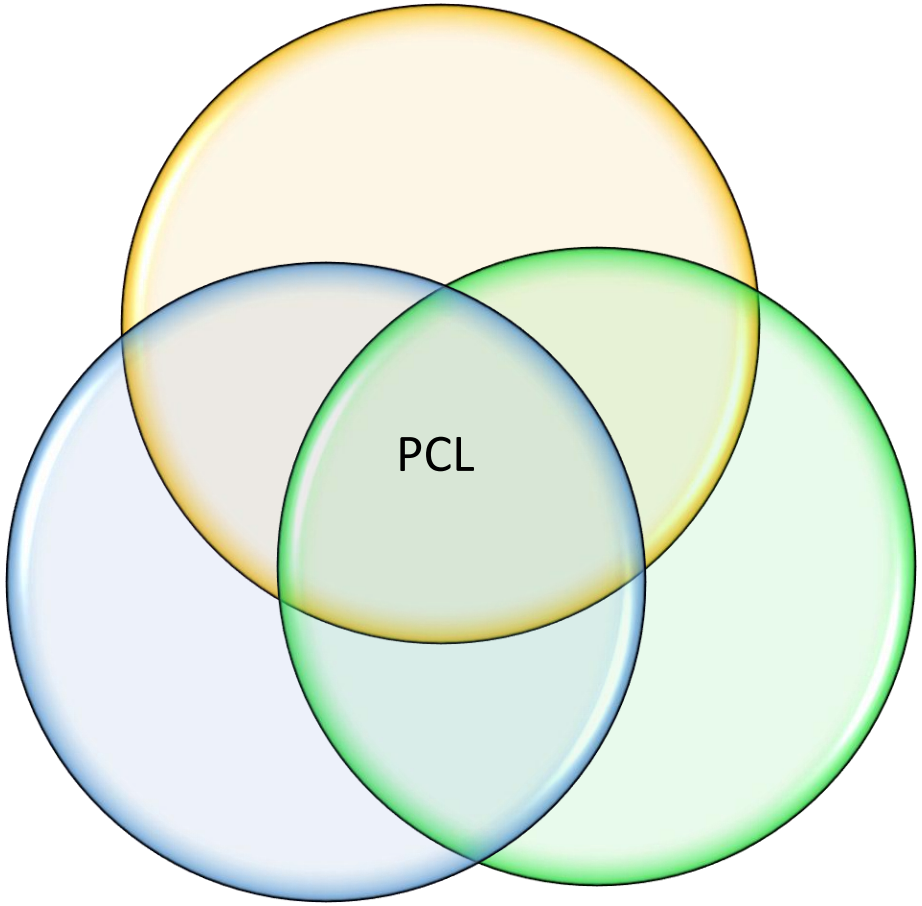
PCLs were an **after thought**, i.e. each **platform could decide** which APIs to includes

- No systematic approach to versioning
- Computed intersection profiles

Each PCLs is targeting a **specific set of platforms**

- Not compatible with newer platforms
- Hard to understand compatibility relationships



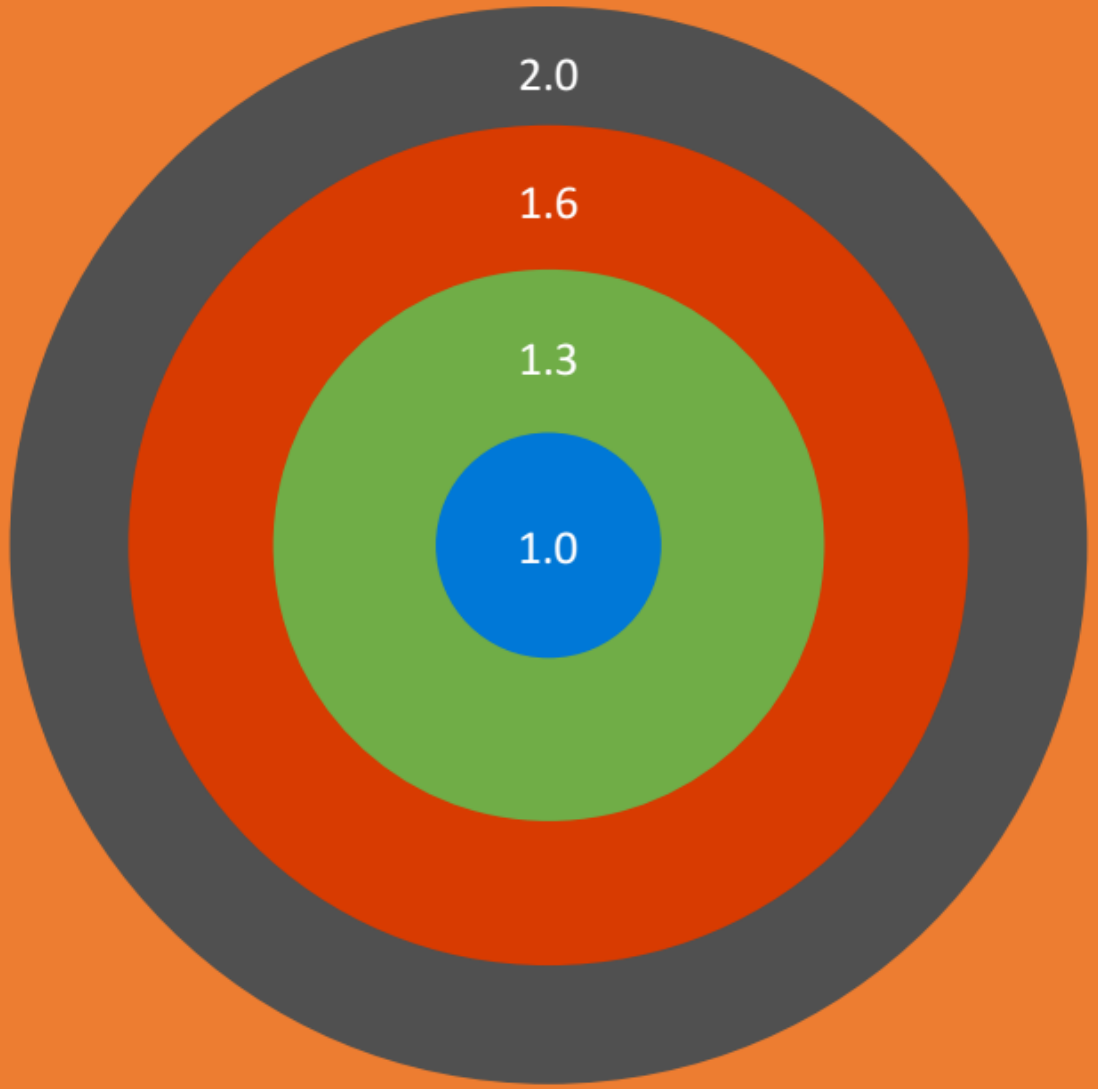


PCL is driven by other libraries

.NET Standard drives

And now we are going to chat about versions

How does versioning work in .NET Standard?



Higher versions incorporate all APIs from previous versions.

- Projects targeting version *X.Y* can reference libraries & projects targeting any version between 1.0 and X.Y

Concrete .NET platforms implement a specific version of .NET Standard

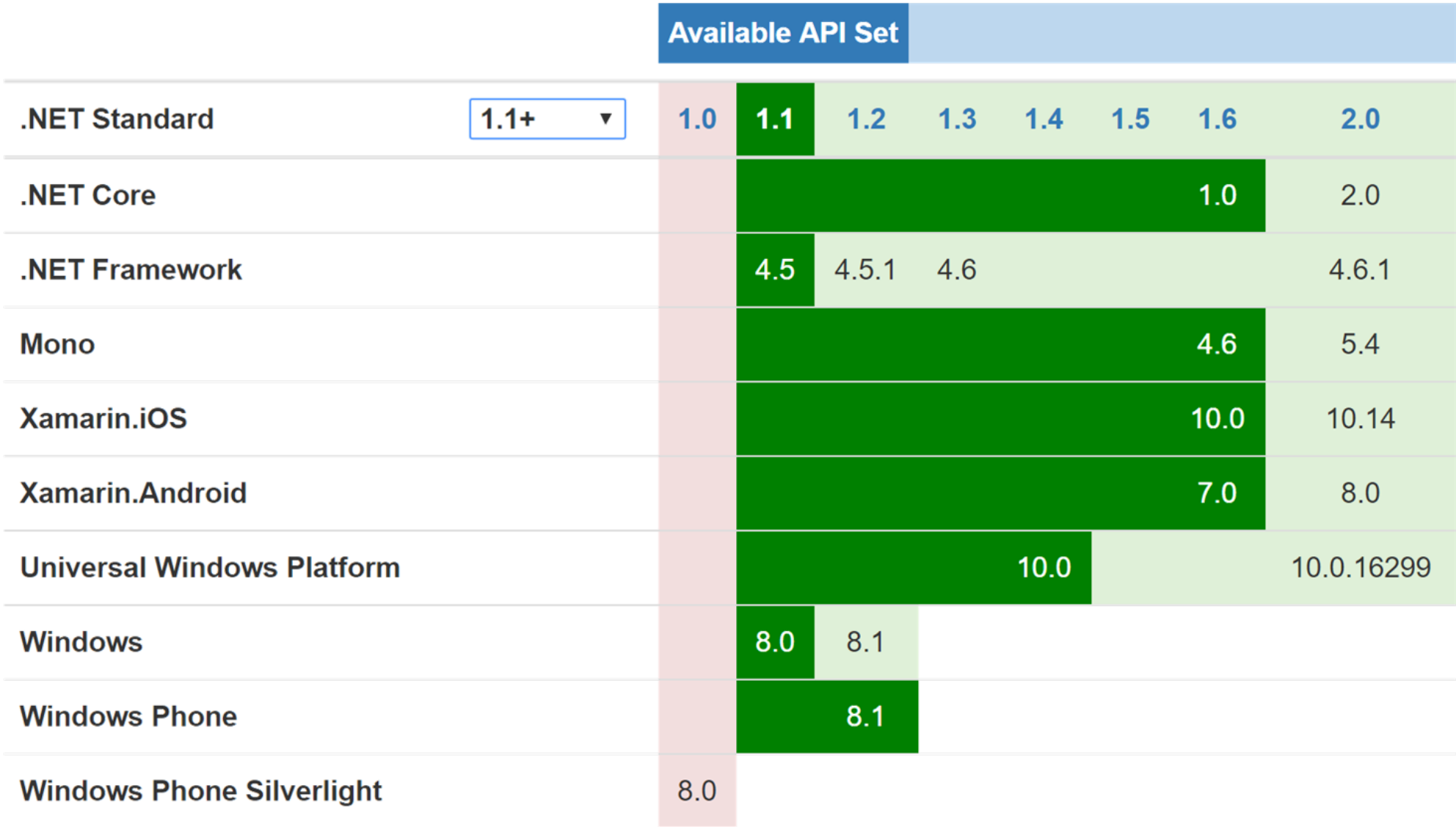
- From that platform you can reference libraries up to that version

	Available API Set							
.NET Standard 2.0+ ▼	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core							1.0	2.0
.NET Framework		4.5	4.5.1	4.6				4.6.1
Mono							4.6	5.4
Xamarin.iOS							10.0	10.14
Xamarin.Android							7.0	8.0
Universal Windows Platform					10.0			10.0.16299
Windows		8.0	8.1					
Windows Phone			8.1					
Windows Phone Silverlight	8.0							

Deciding on a Version

- Create a standard library to work with an existing app targeting 4.5.0
  - Find .NET 4.5.1 in table-that's the version of .NET Standard you need to use
- Create a green-field app with .NET Standard architecture
  - Use .NET Standard 2.0
- Create a library for broadest possible reach
  - Use the lowest version of .NET Standard that has the features you require

	Available API Set							
.NET Standard 2.0+ ▼	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core							1.0	2.0
.NET Framework		4.5	4.5.1	4.6				4.6.1
Mono							4.6	5.4
Xamarin.iOS							10.0	10.14
Xamarin.Android							7.0	8.0
Universal Windows Platform					10.0			10.0.16299
Windows		8.0	8.1					
Windows Phone			8.1					
Windows Phone Silverlight	8.0							


Available API Set
.NET Standard
1.1+
1.0
1.1
1.2
1.3
1.4
1.5
1.6
2.0
.NET Core
1.0
2.0
.NET Framework
4.5
4.5.1
4.6
4.6.1
Mono
4.6
5.4
Xamarin.iOS
10.0
10.14
Xamarin.Android
7.0
8.0
Universal Windows Platform
10.0
10.0.16299
Windows
8.0
8.1
Windows Phone
8.1
Windows Phone Silverlight
8.0

The Version Table

To get the static version table

- https://github.com/dotnet/standard/blob/master/docs/versions.md

To get the dynamic version table

- http://immo.landwerth.net/netstandard-versions/#

What about the next version of Standard?

- It’s a reference assembly, it only evolves on new features
  - Don’t expect to see patches
- Platforms must adapt for a new .NET Standard to be meaningful
  - Imagine a new column in the table
  - Initially empty
  - As each platform adapts, it gets an entry
- Don’t worry about seeing another massive change soon
  - Platforms and Standard now largely align

.NET Core and .NET Standard

.NET Core is an implementation of the .NET Standard

They are **fully separated**, e.g. different GitHub repositories

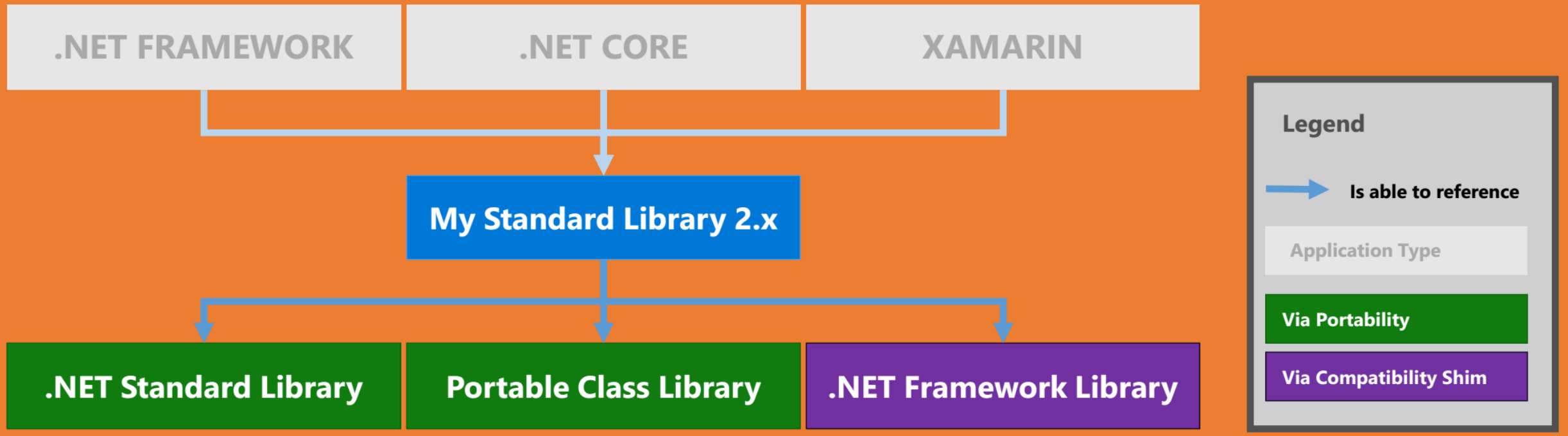
.NET Standard updates are coordinated across all .NET implementers

- There is a .NET Standard review board

.NET Core can be updated independently

- Used by us to experiment and accelerate innovation

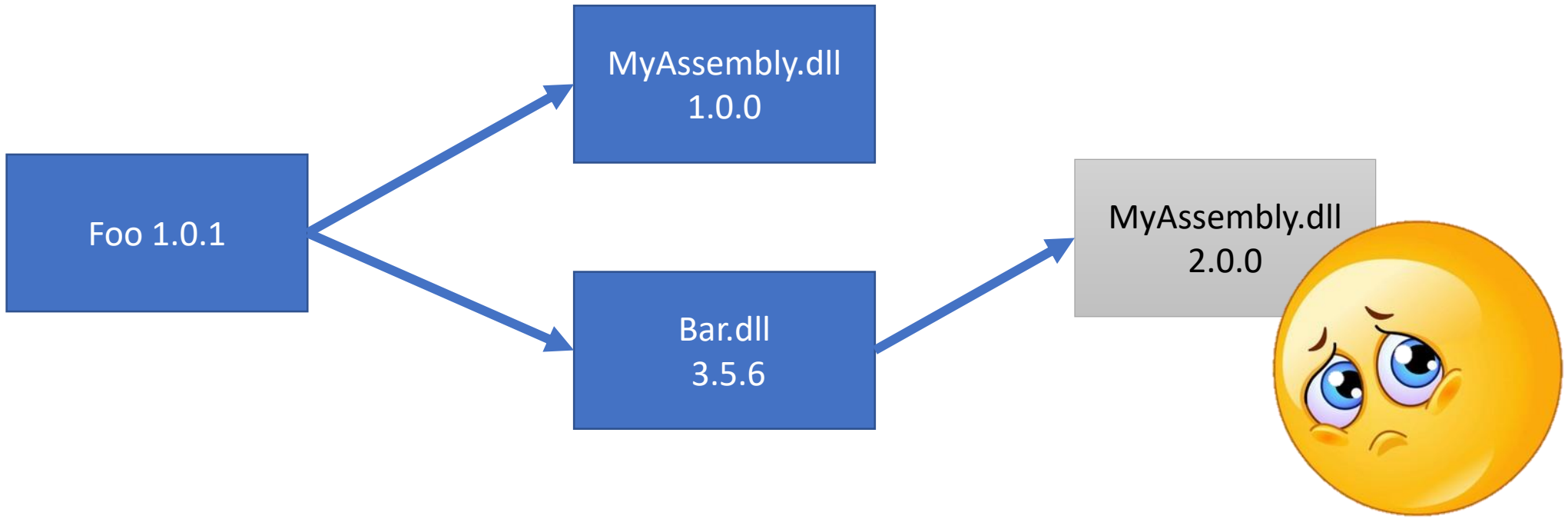
What can you reference from .NET Standard?



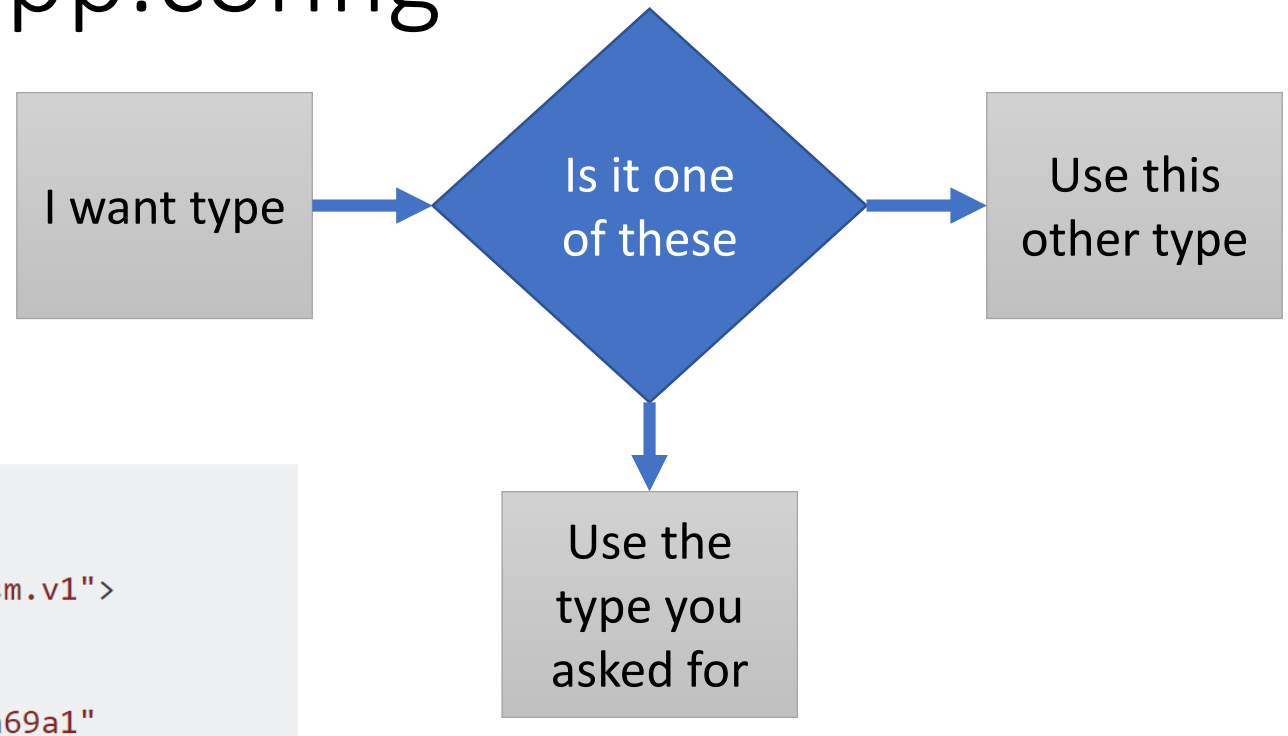
How did they do that!?!

- Binding redirects
- Type forwarding
- Type unification

Binding Redirects – The Problem



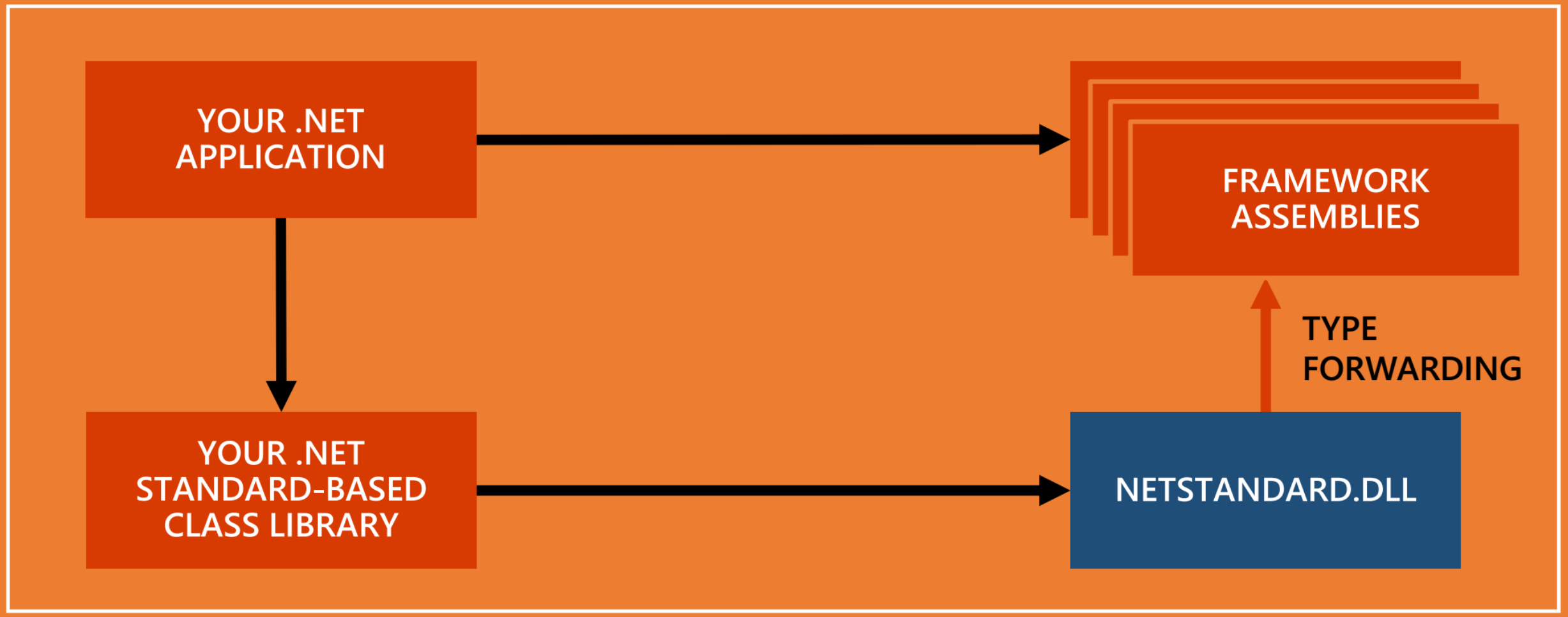
Binding Redirects in app.config




```
<configuration>
   <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
         <dependentAssembly>
            <assemblyIdentity name="myAssembly"
                              publicKeyToken="32ab4ba45e0a69a1"
                              culture="neutral" />
            <bindingRedirect oldVersion="1.0.0.0"
                             newVersion="2.0.0.0"/>
         </dependentAssembly>
      </assemblyBinding>
   </runtime>
</configuration>
```

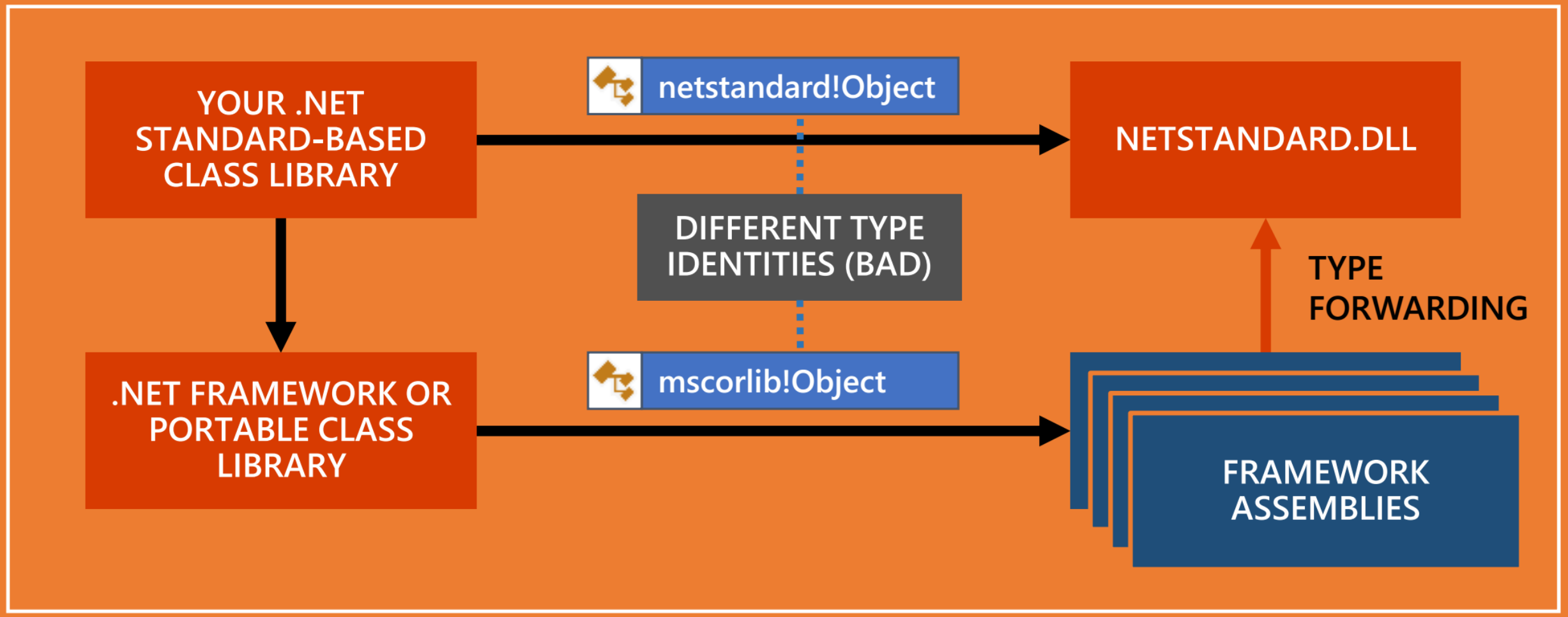
https://stackoverflow.com/questions/468825/binding-redirects

.NET Standard under the hood



This happens when you load .NET Standard-based library

.NET Standard under the hood



This happens when you build a .NET Standard-based Library

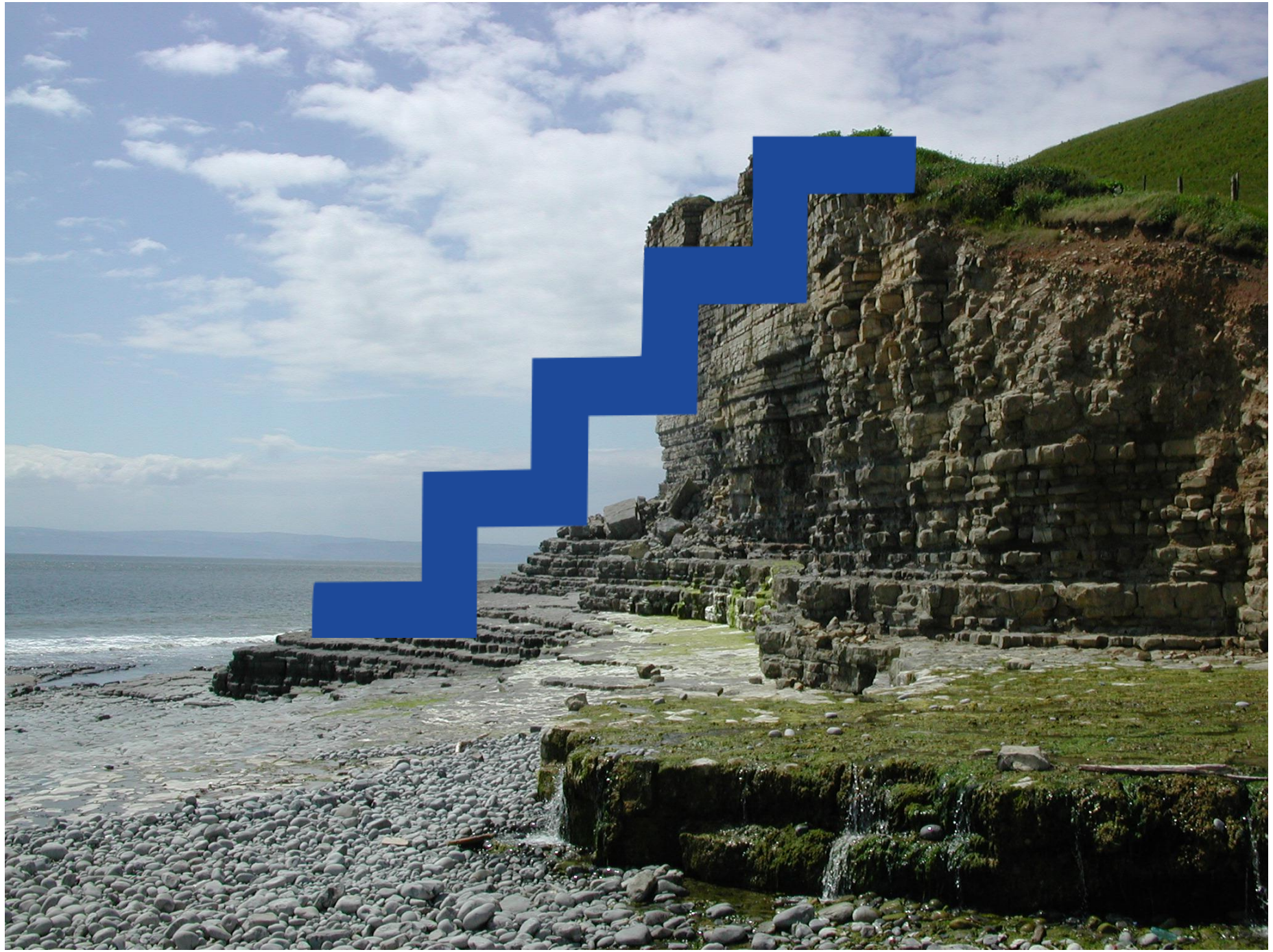
And what about me?

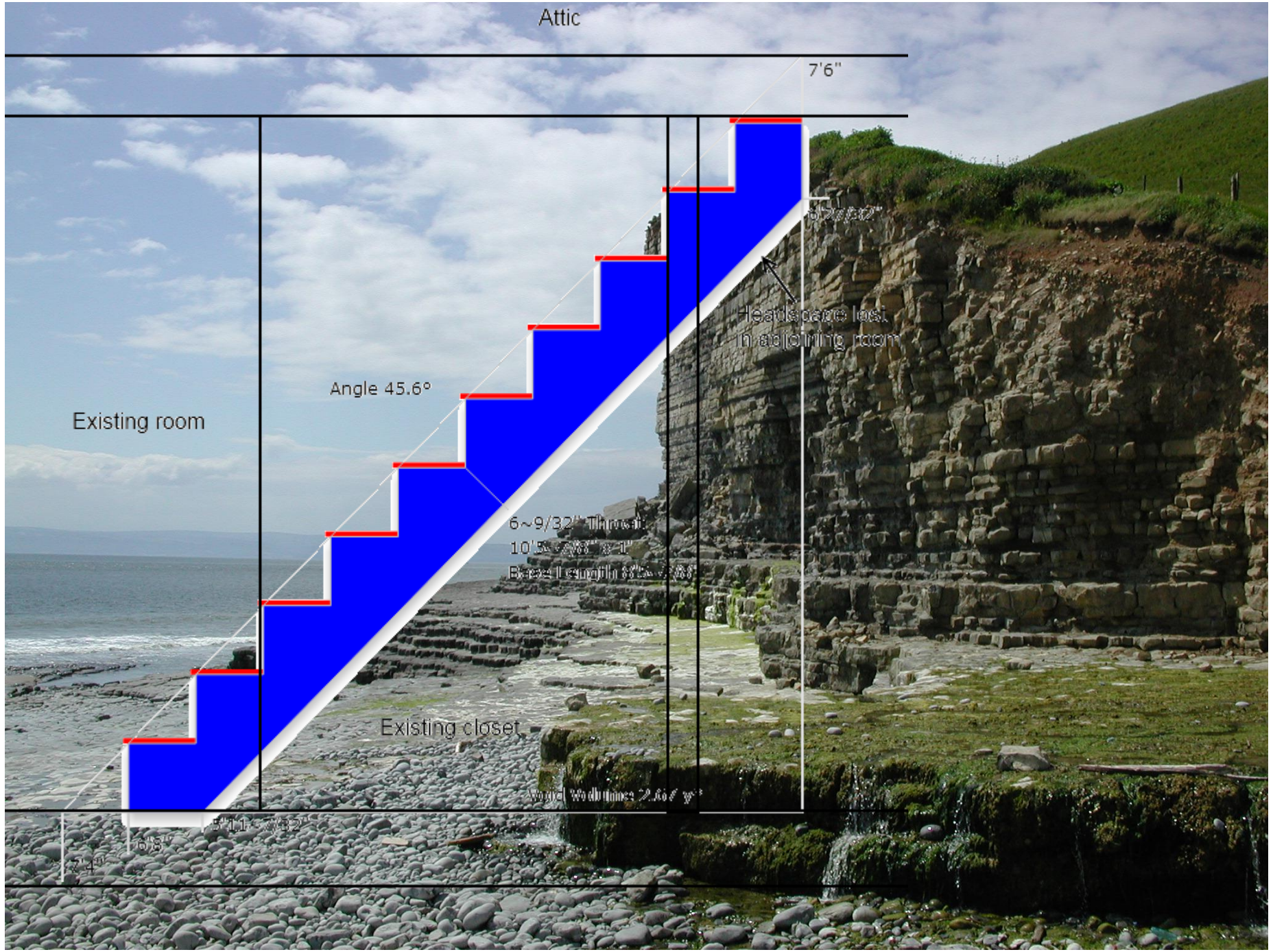
Figuring Out Compatibility

Demo

- apisof.net
- .NET PortabilityAnalyzer
  - VS extension and EXE for command – in Gallery
- API Analyzer
  - Analyzer downloaded via NuGet – Microsoft.DotNet.Analyzers.Compatibility

So, whatcha’ going to do?




Attic
7'6"
Angle 45.6°
Existing room
Existing closet

Every step adds value

Organize Code

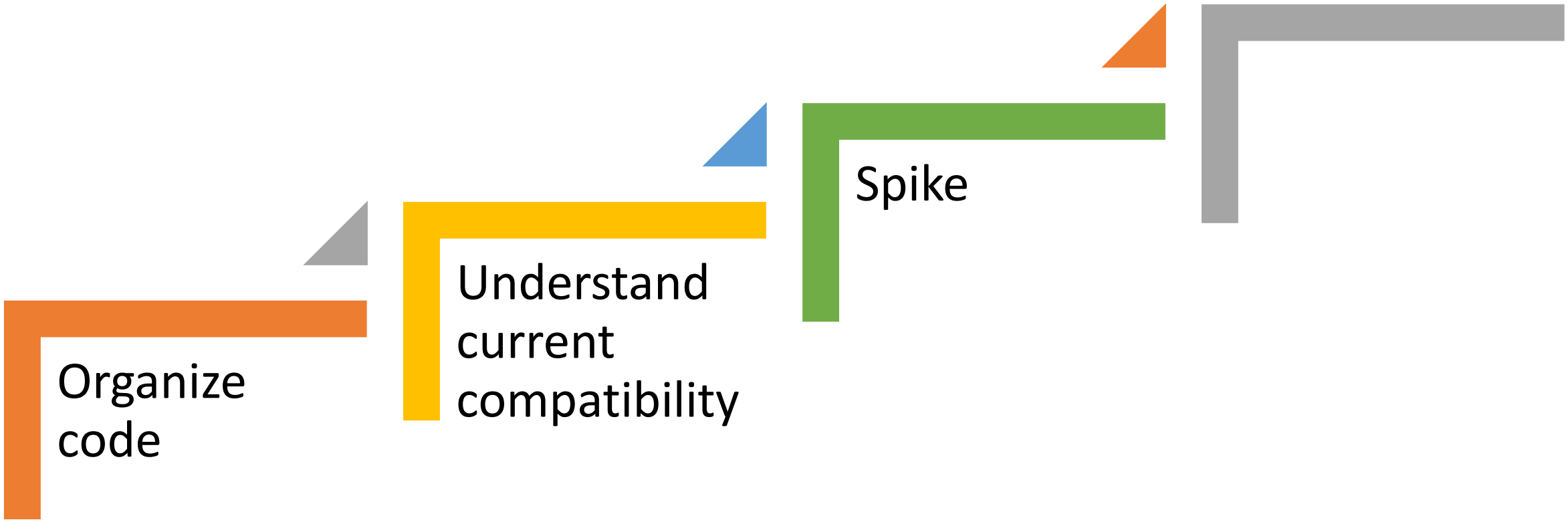
- Isolate platform specific code
  - UI and other code
- Consider isolating code by technology
  - Service code separate from calcuatoins, etc
  - Touch tech specific via functionally designed interfaces
- Consider layers

Every step adds value

Understand Compatibility

- Use tools to know you align with other platforms
- Run these tools regularly to catch issues early

Every step adds value

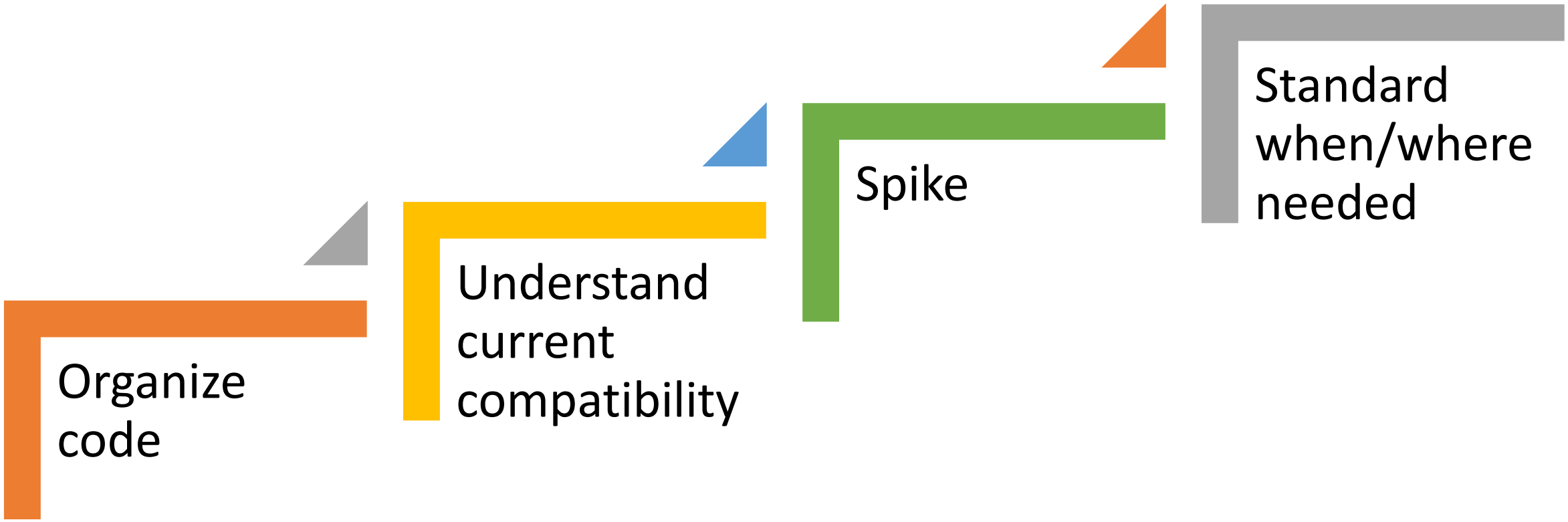


Spike with Dev Tools and Production

- .NET Full Framework had more challenging work
  - If practical, go to 4.7.1 or higher
  - Consider multi-targeting
- Ensure perf with your tools remains acceptable
- *Do not jump off the high dive until you know there is water in the pool(*)*

(*) and the water is warm and preferably margaritas near by

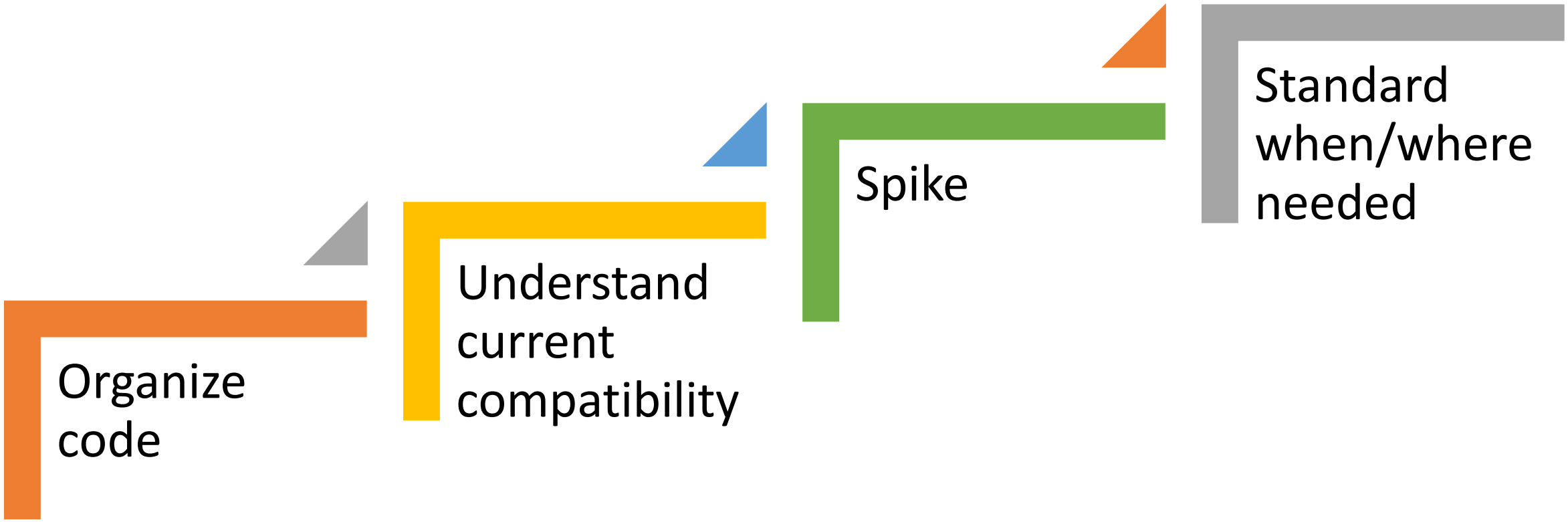
Every step adds value



Standard When/Where Needed

- Standard is awesome!
  - Allows you to write for cross platform
  - Maintains a broad and effective API surface (2.0)
  - Has a broad reach (1.n)
  - Defines a common future
- Your projects can have multiple targets
  - Compile for .NET Standard for most platforms
  - Also, compile for .NET Full Framework if that makes sense

Every step adds value



You got this!

Questions?

.NET Standard

kdollard@microsoft.com

@kathleendollard

References

Immo Landwerth on You Tube: aka.ms/immo