DOTNEXT

An in-depth look at the new features in C# 8.0 and .NET Core 3.0

Raffaele Rialdi - Senior Software Architect

@raffaeler
raffaeler@vevy.com

Who am I?

- Raffaele Rialdi, Senior Software Architect in Vevy Europe – Italy
    - @raffaeler also known as "Raf"
- Consultant in many industries
    - Manufacturing, racing, healthcare, financial, …
- Speaker and Trainer around the globe (development and security)
    - Italy, Romania, Bulgaria, Russia (Moscow, St Petersburg and Novosibirsk), USA, …
- And proud member of the great Microsoft MVP family since 2003

Agenda

- A modern approach to application development in .NET Core
- C# 8 interesting features
  - readonly members in structs
  - static local functions
  - default interface members
- New publishing options for .NET Core 3
- Load Contexts
- Diagnostic tools

Not covering any topic that is available in other #dotnext sessions

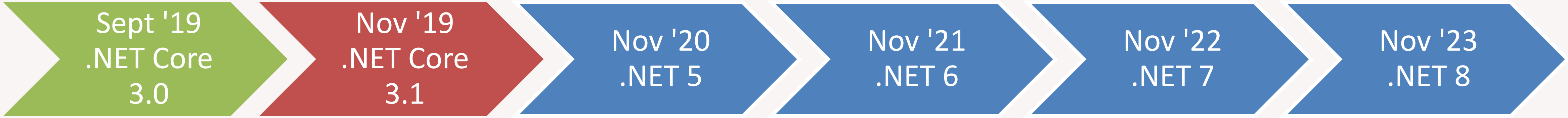
But we can talk about them in the discussion zone!

.NET (Framework) is dead, long live .NET (Core)

- Version 4.8 closes the .NET Framework evolution
  - No worries, it will be supported for a very long time
  - No C# 8 or netstandard2.1 for .NET Framework

- The future is .NET 5, the next major release of .NET Core
  - No more "Core" naming, migration is easier than ever

- The roadmap is predictable, one major every year



Sept '19
.NET Core
3.0

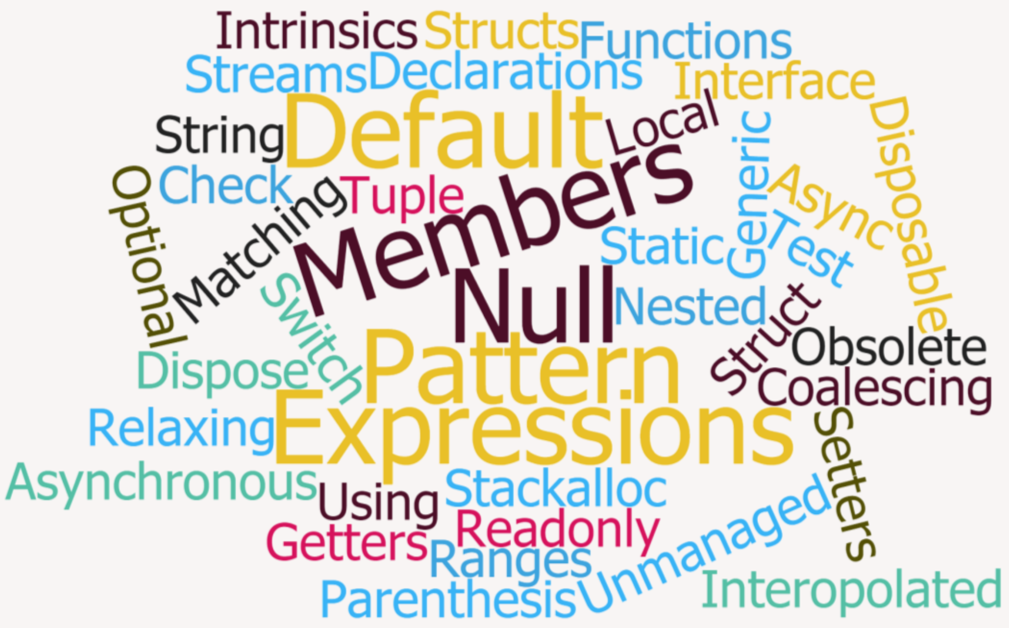
Nov '19
.NET Core
3.1

Nov '20
.NET 5

Nov '21
.NET 6

Nov '22
.NET 7

Nov '23
.NET 8



C# 8.0 readonly members in structs

~20 nuove features

Readonly struct members

- Ability to mark a member as readonly
  - The compiler will enforce immutability on its instance (not on parameters)
  - Auto property getters are implicitly marked as readonly

- When should we use it?
  - To express the readonly intent … better usability and maintenance
  - To help the compiler apply optimizations

- What happens if I try to modify the instance state from a readonly member?
  - Error CS1604, if you try to modify any field
  - Warning CS8656 (perf hit), if accessing a non-readonly explicit property getter
    - *Call to non-readonly member '…' from a 'readonly' member results in an implicit copy of 'this'*

Help the compiler!

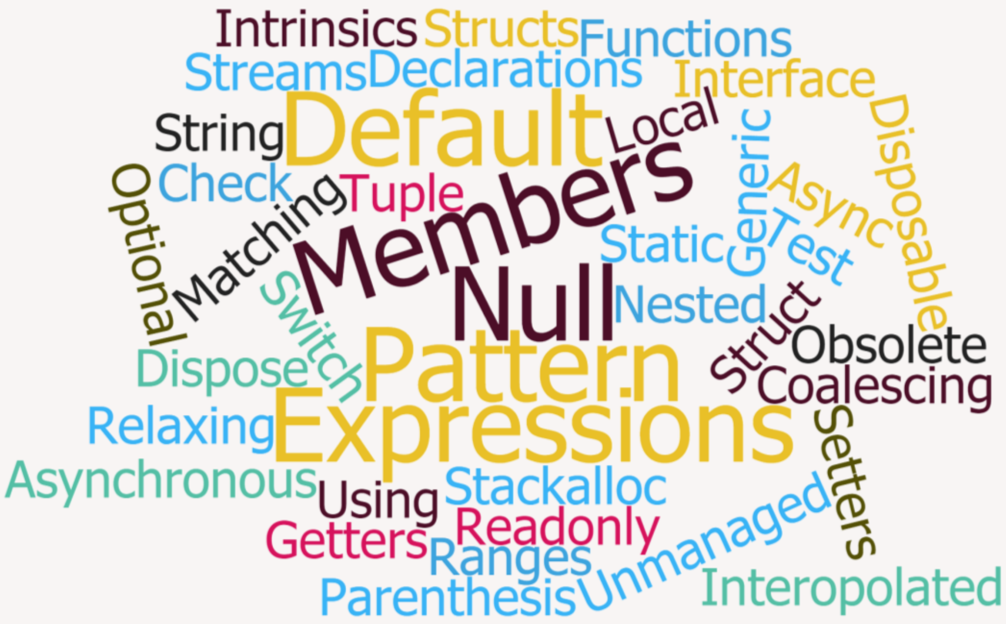
```
public struct Vector
{
    public float GetLength() => ...
    public readonly float GetLengthReadonly() => ...
}
```

```
public static float Bad(in Vector vector)
{
    return vector.GetLength();
}
```

This will cause a local copy of vector

```
public static float Good(in Vector vector)
{
    return vector.GetLengthReadonly();
}
```

"`in`" means "passed by reference, but the reference is readonly"

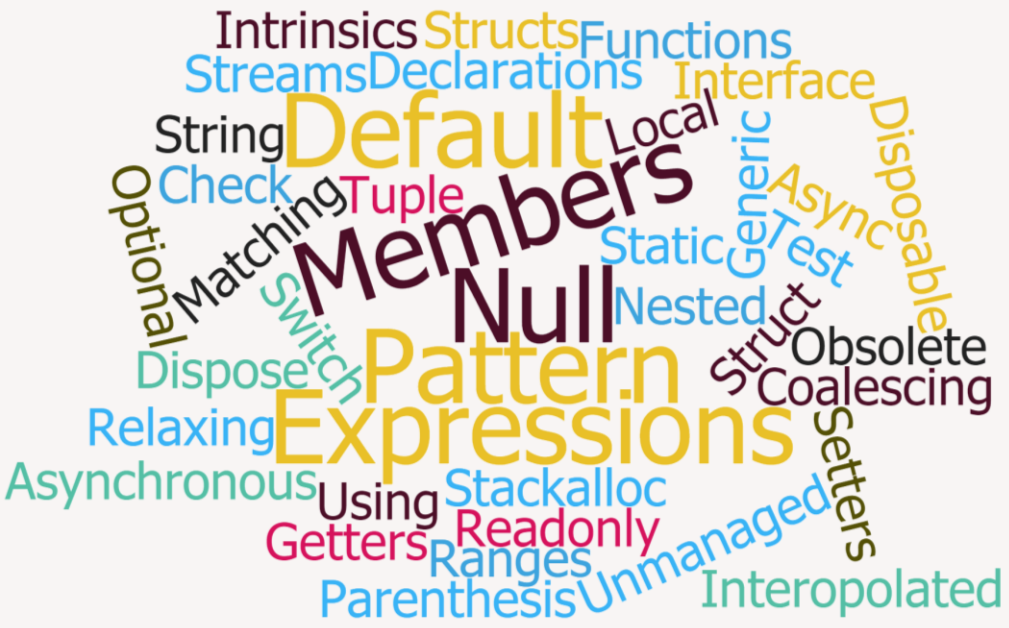


C# 8.0 static local functions

Static local functions

```
private async Task Scale(Point[] vector, int factor)
{
    await Task.Delay(1);
    for (int i = 0; i < vector.Length; i++)
    {
        GetRef(vector, i).X *= factor;
        GetRef(vector, i).Y *= factor;
    }

    static ref Point GetRef(Point[] vector, int index)
    {
        var span = vector.AsSpan();
        return ref span[index];
    }
}
```



C# 8.0 default interface members

Default interface members

- Interfaces can now contain:
  - Bodies on any interface declaration members
  - Static members (including constructors and nested types)
  - Visibility and 'partial' modifiers

- Can not contain
  - Instance constructors, fields or auto-properties (must stay stateless)

- Derived types cannot call base member bodies
  - Proposed syntax for C# 9: base(InterfaceType).Method()

Default interface members: why?

1. Versioning

   how difficult can be adding an interface member?

2. Interoperability with other languages supporting it

   Swift and Java

3. Traits-based programming

   Composing behavior of an object reusing units of code

   Very popular in C++, used also by Java and Swift

Versioning

Assembly 1, version 1

```
interface I1          interface I2 : I1
{                     {
   int M1() => 1;
}                     }
```

Assembly 1, version 2

```
interface I1          interface I2 : I1
{                     {
   int M1() => 1;        int I1.M1() => 2;
}                     }
```

Assembly 2, version 1 (NOT recompiled)

```
class X : I2          void Print(I2 i2)
{                     {
  // ...                WriteLine(i2.M1());
}                     }
```

At runtime …

```
var x = new X();
Print(x);

1
```

At runtime …

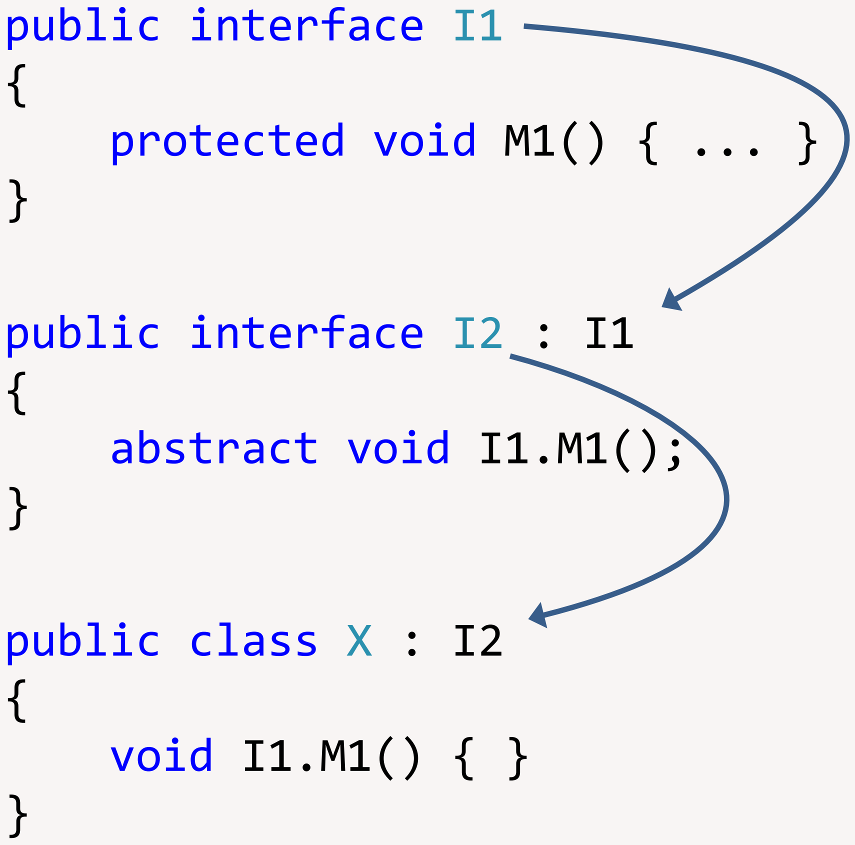
```
var x = new X();
Print(x);

2
```

Interface reabstraction

- Reabstraction is allowed

```
public abstract class Y : I1
{
    public abstract void M1();
}
```



```
public interface I1
{
    protected void M1() { ... }
}

public interface I2 : I1
{
    abstract void I1.M1();
}

public class X : I2
{
    void I1.M1() { }
}
```

Introducing 'Traits Composition'

- The "Language Transliteration" case
  - Plugging in new language transliterations, version after version
  - Defining an interface with all the possible permutations is not realistic
- Using static helper classes?
  - Difficult to take decisions at runtime based on their availability
- Defining many separate interfaces?
  - Not easy to predict the members shape
- Traits to the rescue!
  - Reusable, **stateless** computational units, made of a set of methods and/or properties
  - Each C# 8 interface may define a scope and a set of members
  - Members can be overriden (re-defined) by another interface or class

.NET Core 3 Publishing

New publishing options

- Framework Dependent Deployment is the new default
  - The executable host is now created by default

- Self-contained deployment (SCD) is optional
  - option   --self-contained = true

Single File Publishing: «PublishSingleFile»

- Compact the entire application in a single file
  - Everything but static web files and configurations files
- By default triggers "self-contained" but it can be turned off

```
dotnet publish -r win-x64 -o folder -p:PublishSingleFile=true
                  --self-contained=false
```

- Can be (optionally) specified in the csproj

```
<PublishSingleFile>true</PublishSingleFile>
<RuntimeIdentifier>win-x64</RuntimeIdentifier>
```

App type (Release)	Simple compile	Self contained=false	Self contained=true
Console	166Kb	166K	67Mb
MVC Web App	314Kb	4Mb	88MB

IL Trimming: «PublishTrimmed»

- Feature inherited from the Mono Project linker
- Goal: removing all the unused IL code
  - Requires --self-contained = true
- Nasty reflection code requires instructing the linker
  - TrimmerRootAssembly to include the specified assembly (or type)
  - TrimmerRootDescription to use an xml hint file

App type: Release and self-contained	Not trimmed	Trimmed
Console	67Mb	26Mb
MVC Web App	88MB	48MB

AOT Compilation: «PublishReadyToRun»

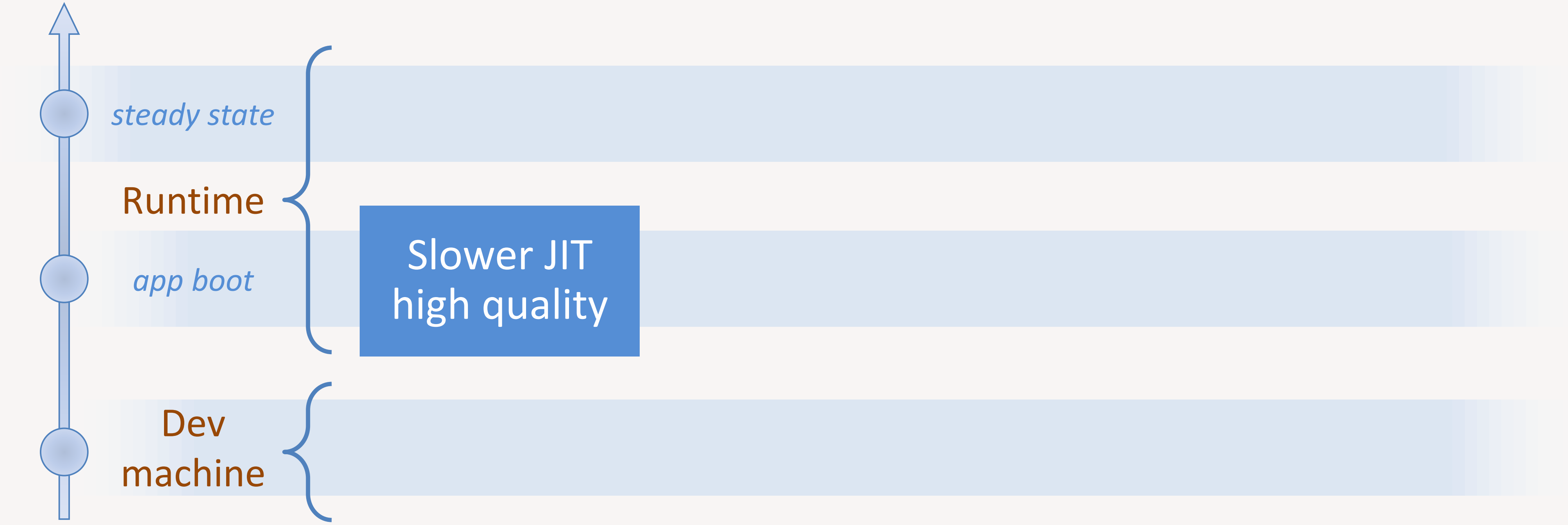
- Ahead Of Time compilation generates native CPU assembly code
  - Similar to NGen, but it is done at compile time, on your (dev) machine
  - Some assemblies can be excluded to reduce the deploy size

```
<PublishReadyToRunExclude
   Include="asm.dll">
```

- Advantages
  - Reduces to almost-zero the bootstrap JIT compilation time
  - Extremely useful for Azure Functions, AWS Lambdas and IoT devices
- Problem:
  - AOT compilation is not able to optimize for a specific CPU
  - Produces less efficient code compared to the JIT/NGen
    - cross-module dependencies cannot be inlined
    - ngen generate absolute addresses that are fragile while AOT computes them

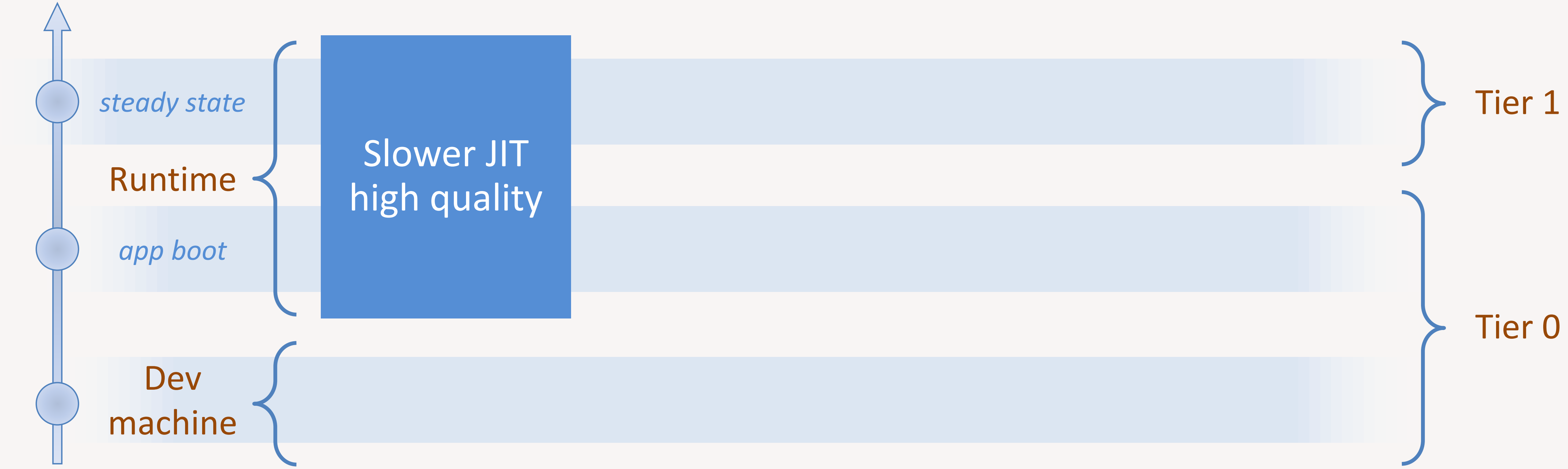
Repeating the JIT compilation: «TieredCompilation»

- When TieredCompilation is off (default is on)
  - The JIT Compiles high quality code, but it takes some time
  - This is also the behaviour of previous versions of the runtime



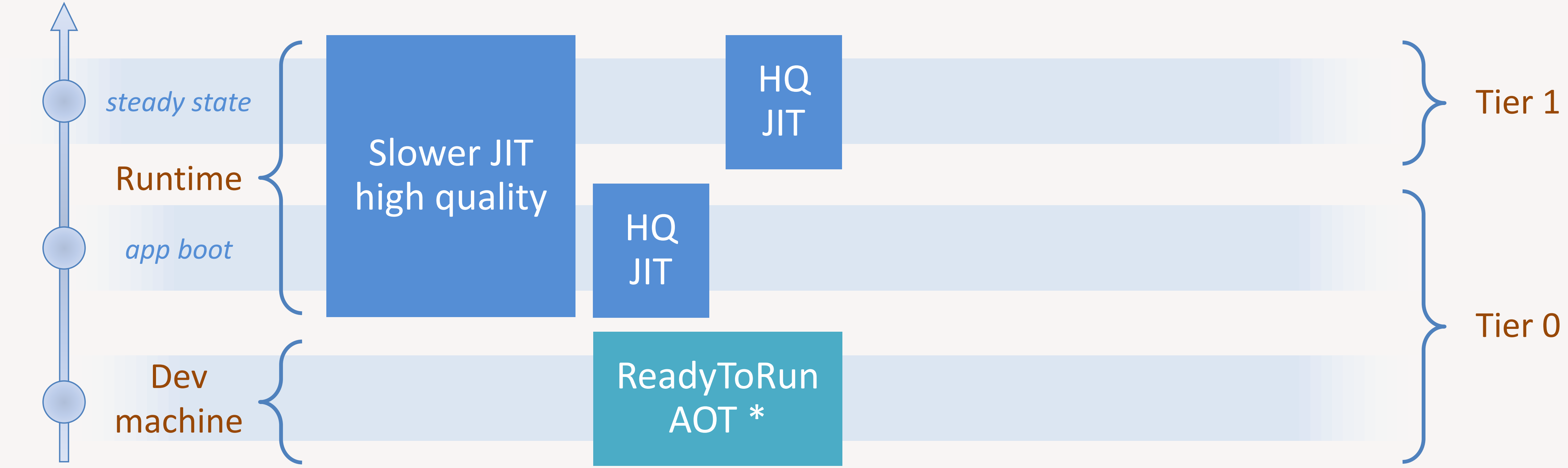
Repeating the JIT compilation: «TieredCompilation»

- TieredCompilation enables a "Tier 1" compilation level
  - In Tier 1, compilation quality and performance are the same we already know



Repeating the JIT compilation: «TieredCompilation»

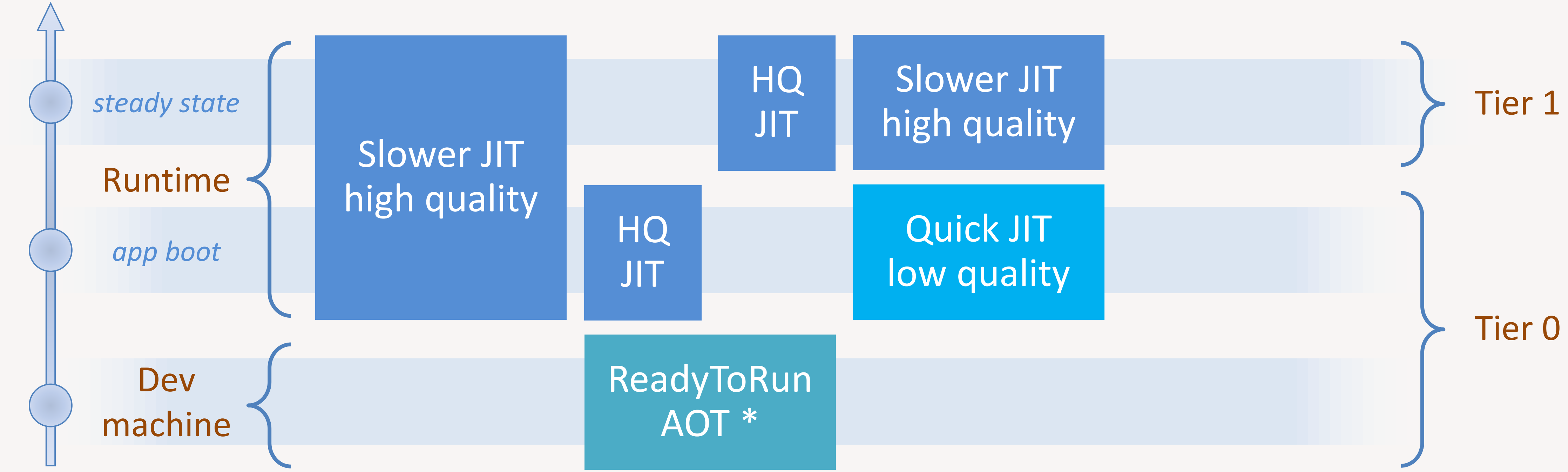
- AOT cannot compile everything (JIT is still needed)
  - All the "hot" paths are recompiled to high-quality code
  - Only the AOT generated code is a candidate to be recompiled



* AOT is an opt-in feature, disabled by default

Repeating the JIT compilation: «TieredCompilation»

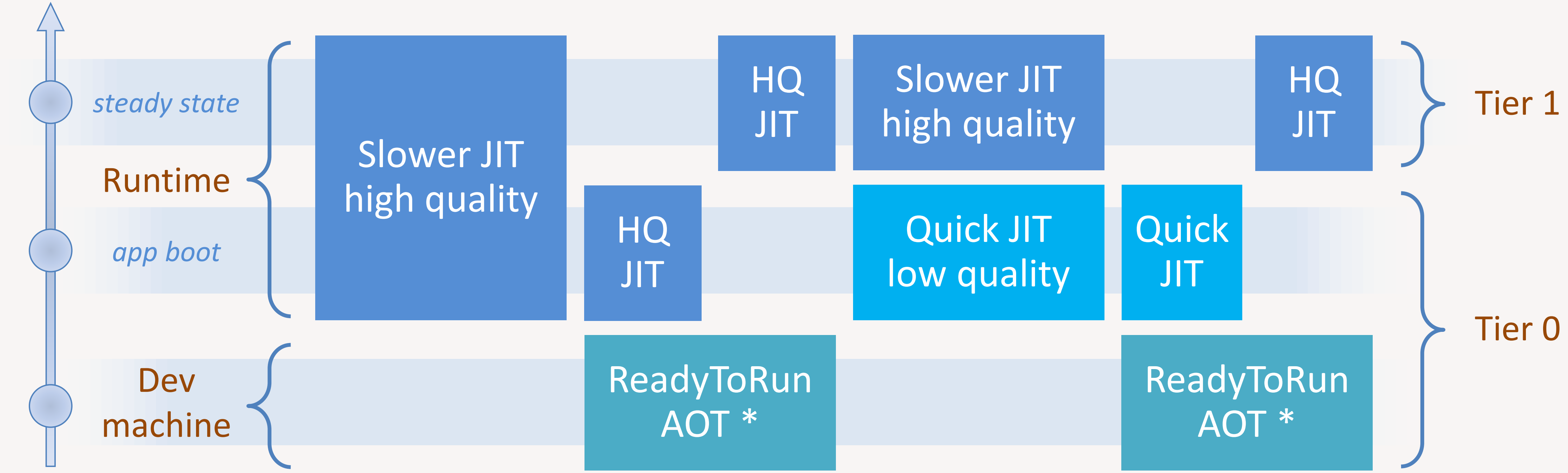
- TieredCompilationQuickJit improves the startup time
  - All the "hot" paths are recompiled to high-quality code



* AOT is an opt-in feature, disabled by default

Repeating the JIT compilation: «TieredCompilation»

- Non AOT code will be JITted initially with the Quick JIT
  - All the "hot" paths are recompiled to high-quality code



* AOT is an opt-in feature, disabled by default

Tiered compilation notes

- The compilation is repeated only if the path is 'hot'
  - A call is hot when its counter reaches 30 times after the initial app boot
  - Re-compilation is queued on a background thread

- AOT: how can I know which code will need the jitter?
  - Perfview
  - R2RDump to analyze the precompiled executable

- In the future it may leverage PGO
  - Profile-Guided Optimization

Load Contexts

System.Runtime.Loader.AssemblyLoadContext

- With .NET Core, there is only a single AppDomain
- In .NET Core 3, Contexts were introduced to replace AppDomains
  - They are **not** a security boundary
  - Load contexts are named, there is no "Current" as it was for AppDomains
  - Almost zero-cost in accessing code in a different context

- Main use cases:
  - Ability to unload addons/plugins from the AppDomain
  - Controlling the resolution (probing) of addons assemblies and native dlls
  - Isolating and using different versions of the same addons

Unloading contexts

- Basic usage

```
var newContext = new AssemblyLoadContext(name: "MyContext",  isCollectible: true);
newContext.LoadFromAssemblyPath(FullAddonFilename);
// ... doing something with the assembly
newContext.Unload();
```

- Typical usage
  - Derive from AssemblyLoadContext
  - Use AssemblyDependencyResolver to resolve the paths
    1. uses the .deps.json file of the main addon, if available
    2. probes subfolders normally used for localization purposes
  - Override Load method to return the assembly or null to skip it

Going deeper on contexts

- The main reflection behavior has not changed
- These calls always creates a separate load context:
  - Assembly.Load(byte[]), Assembly.LoadFrom(filename)

- Unloading from the AppDomain (from memory) **is not deterministic**
  - The GC can be forced to accelerate unloading, but there is no event advising
    - Example: the **TypeDescriptor** private cache prevents contexts using it to be unloaded
      - Newtonsoft.Jsoft is one of the libraries using TypeDescriptor and demonstrating the problem
  - The Unloading event fires on Unload request, not when memory is freed

- There is no "current context" concept
  - AssemblyLoadContext.GetContext(Assembly) is a good alternative

Making existing code use the desired context

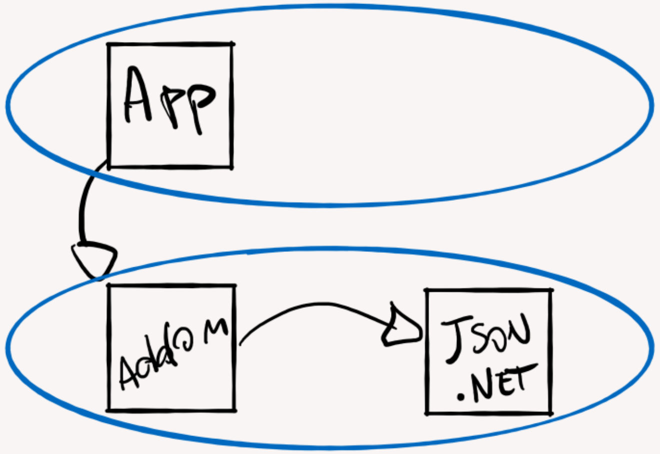
- Code using reflection APIs can be diverted to load the assemblies into the desired LoadContext
  - Assembly.Load(assemblyName), Assembly.LoadWithPartialName(…)
  - CreateInstance(assemblyName, …)
  - Type.GetType and Assembly.GetType using assembly qualified names

```
using (addonContext.EnterContextualReflection())
{
    addonAssembly = Assembly.Load(addonAssemblyName);
}
```

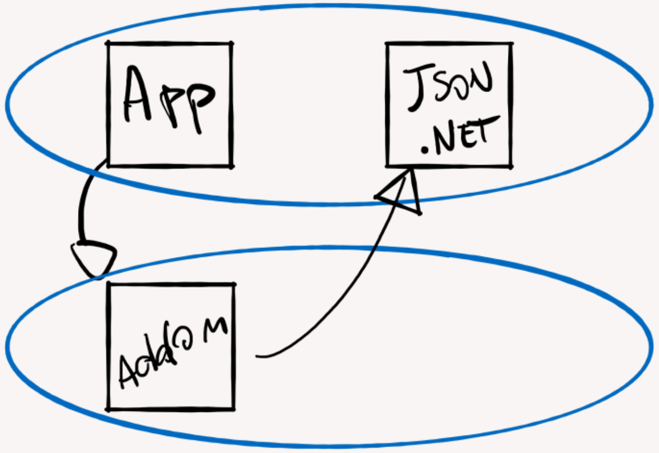
Managing dependencies and binding isolation

- Contexts isolate the assemblies
  - Load the Common assemblies (IAddon type) **only** in the default context
- Dependencies can be loaded where you want:

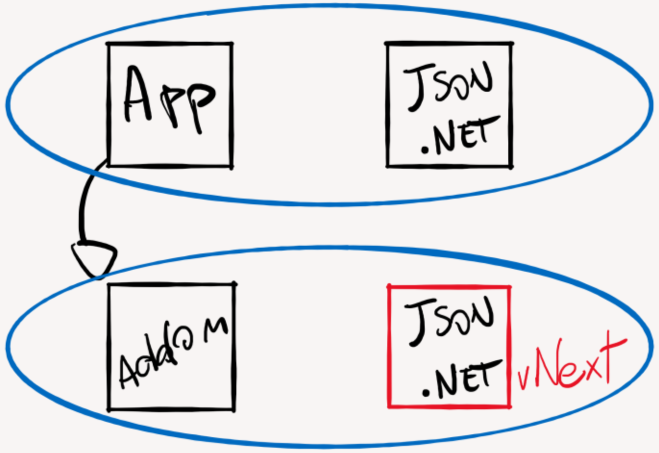
Json dependency
loaded in the addon context



Json dependency
loaded in the default context



Each context
uses its own version



Diagnostic tools:
finding the leaking reference

Diagnosing an unloadable AssemblyLoadContext

- The question is: Who is taking a reference to LoaderAllocator?
  - Walk the stack until you get **the first** instance that lives in the outside the addon Context.
- dumpheap -type LoaderAllocator
- gcroot -all (hex address of LoaderAllocator)
  - Address hold in a register?
    - Registers typically hold local variables in the current method
  - Pinned handle?
    - Static fields are hold by a pinned handle of an object array

rectangles indicate the objects that should have gone away with the context

```
> gcroot -all 0025822f41b20
Thread 2de8:
  00CF94B7E260 00007FFF920067FC System.ConsolePal.ReadKey(Boolean)
[/_/src/System.Console/src/System/ConsolePal.Windows.cs @ 338]
        rbx:  (interior)
            ->  0000025832F31038 System.Object[]          array holding static elements
            ->  0000025822F456F8 AddonLibrary.FileProvider
            ->  0000025822F41B20 System.Reflection.LoaderAllocator

  00CF94B7E380 00007FFEFEA41004 NetCore3.Program.Main(System.String[]) [...\NetCore3\Program.cs @ 57]
        rbp+30: 000000cf94b7e3b0                          rbp is the stack Base Pointer
            ->  0000025822F3BAD0 NetCore3.DemoAddonsBuggy
            ->  0000025822F456F8 AddonLibrary.FileProvider
            ->  0000025822F41B20 System.Reflection.LoaderAllocator

  00CF94B7E380 00007FFEFEA41004 NetCore3.Program.Main(System.String[]) [...\NetCore3\Program.cs @ 57]
        rbp+48: 000000cf94b7e3c8                          rbp is the stack Base Pointer
            ->  0000025822F3BAD0 NetCore3.DemoAddonsBuggy
            ->  0000025822F456F8 AddonLibrary.FileProvider
            ->  0000025822F41B20 System.Reflection.LoaderAllocator

HandleTable:
    00000258212A15F8 (pinned handle)
    -> 0000025832F31038 System.Object[]  array holding static elements
    -> 0000025822F456F8 AddonLibrary.FileProvider
    -> 0000025822F41B20 System.Reflection.LoaderAllocator

Found 4 roots.
```

Fields investigation

```
> dumpobj 0000025822F3BAD0
Name:        NetCore3.DemoAddonsBuggy
MethodTable: 00007ffefeb01ef0
EEClass:     00007ffefeafd480
Size:        24(0x18) bytes
File:        H:\...\NetCore3\bin\Debug\netcoreapp3.0\NetCore3.dll
Fields:
           MT    Field Offset               Type VT     Attr           Value Name
07ffefeb52880 400000e      8        Common.IAddon  0 instance 000025822f456f8 <Addon>k__BackingField
07ffefeb52178 400000d     10 NetCore3.AddonInfo  0   static 000025822f40868 _addonInfo
07ffefeb52880 400000f     18        Common.IAddon  0   static 000025822f456f8 <Addon2>k__BackingField
```

dumpmt (address)

dumpclass (address)

property names

The nasty case of TypeConverter used by Json.NET

```
> gcroot -all 01a5a0edeb98
HandleTable:
    000001A59F4715D0 (pinned handle)
    -> 000001A5B0ED5CD8 System.Object[]
    -> 000001A5A0EE91D0 System.Collections.Hashtable
    -> 000001A5A0EE9730 System.Collections.Hashtable+bucket[]
    -> 000001A5A0EDFA68 System.RuntimeType
    -> 000001A5A0EDEB98 System.Reflection.LoaderAllocator
Found 1 roots.
```

The problem is a static reference to an Hashtable

dumpobj ➔ AddonLibrary.FileProvider

??? ➔ manual search!

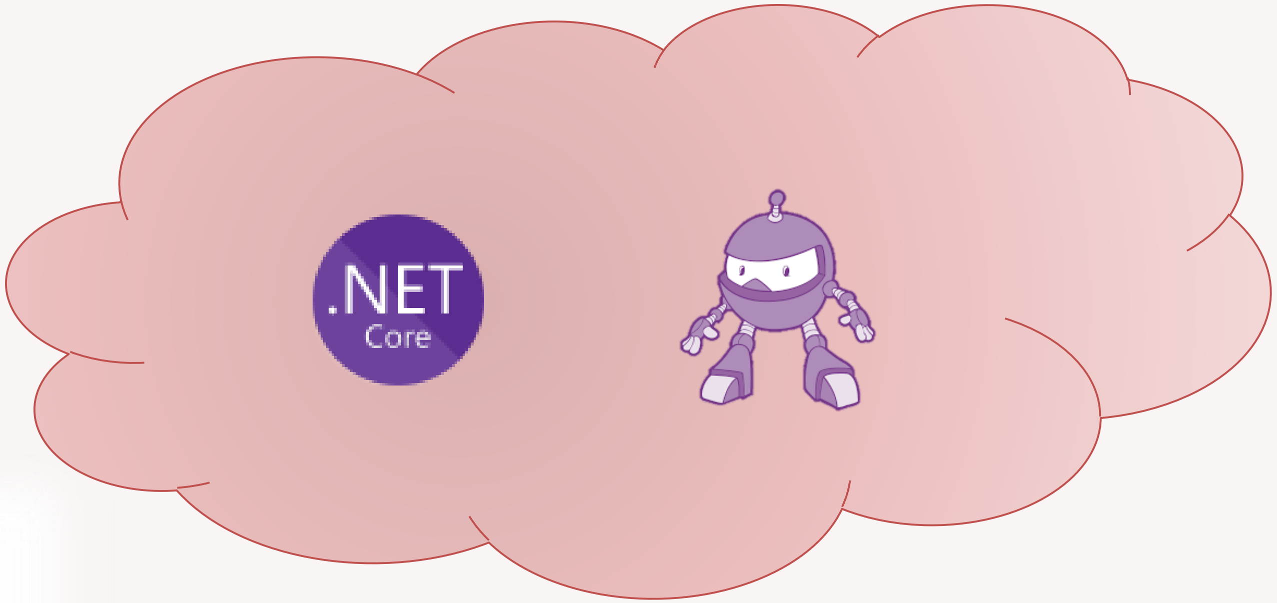
```
> dumpobj 000001a5a0ee8828
Name:         System.ComponentModel.ReflectTypeDescriptionProvider
MT              Field   Offset                 Type VT     Attr            Value Name
7ffefd30a010  40000d0        8 ...scriptionProvider  0 instance 0000000000000000 _parent
000000000000  40000d1       10 ...tomTypeDescriptor  0 instance 0000000000000000 _emptyDescriptor
7ffefd30b998  400008d       18 ...ections.Hashtable  0 instance 000001a5a0ee89f8 _typeData
7ffefd0dc620  400008e       c0        System.Type[]  0   static 000001a5a0ee8908 s_typeConstructor
7ffefd30b998  400008f       c8 ...ections.Hashtable  0   static 0000000000000000 s_editorTables
7ffefd30b998  4000090       d0 ...ections.Hashtable  0   static 000001a5a0ee91d0 s_intrinsicTypeConverters
```

SOS cheat sheet

dotnet-dump = SOS made easy

- Install dotnet-dump, dump the process and analyze it:
  - dotnet tool install -g dotnet-dump
  - dotnet-dump collect -p <pid>
  - dotnet-dump analyze filename.dmp
- List MT/metadata types for live objects dumpheap -stat
- Search by partial type name dumpheap -type Assem
- Getting more details
  - List all the objects of a given MT dumpheap -mt <hex returned from dumpheap -stat>
  - Details about the given instance dumpobj <address>
  - Details about the given EEClass dumpclass <eeclass>
  - Details about the given MT dumpmt <metadata table>
- Assemblies
  - List all the assemblies in memory: dumpdomain
  - Details about the given assembly dumpassembly <address>

Questions @ booth 1



Thank you!

@raffaeler

raffaeler@vevy.com