XAMARIN DEVELOPER SUMMIT

Houston, TX

2019

#XamDevSummit

Your Xamarin application is probably leaking memory and you don't know it

CTO & Owner
Mahiz
Microsoft MVP
Developer Technologies
dan@mahiz.it

Dan Ardelean

@danardelean

Mahiz

- Headquarters: **Parma**, Italy
- Local Office: **Cochabamba**, Bolivia & **Bucharest**, Romania

Mahiz

SOFTWARE DEVELOPMENT

CONSULTING

TRAINING

How does Xamarin works?

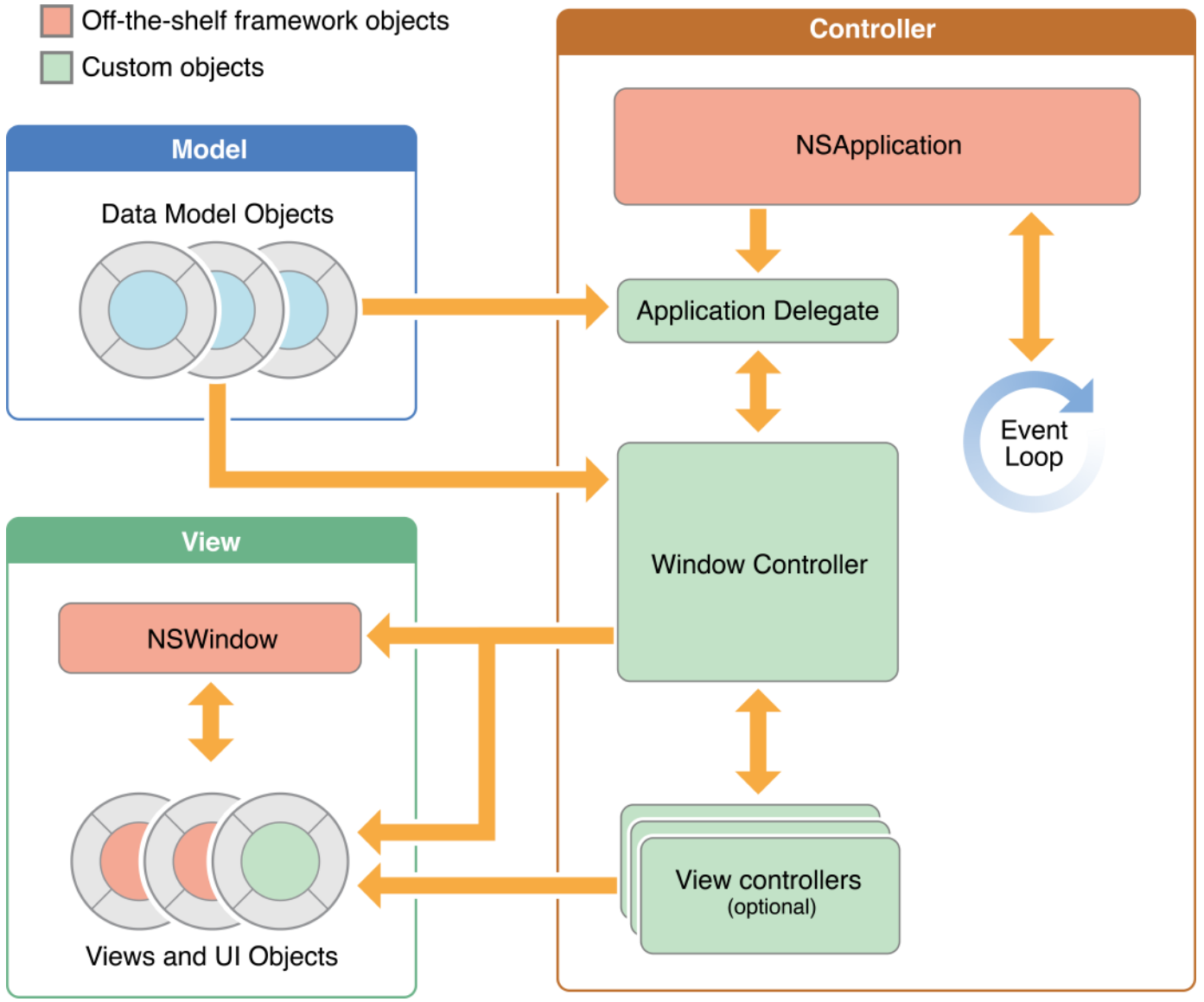
- Download and install Visual Studio
- Write code in C#/F#
- Press F5
- Wait …
- Wait …

NATIVE

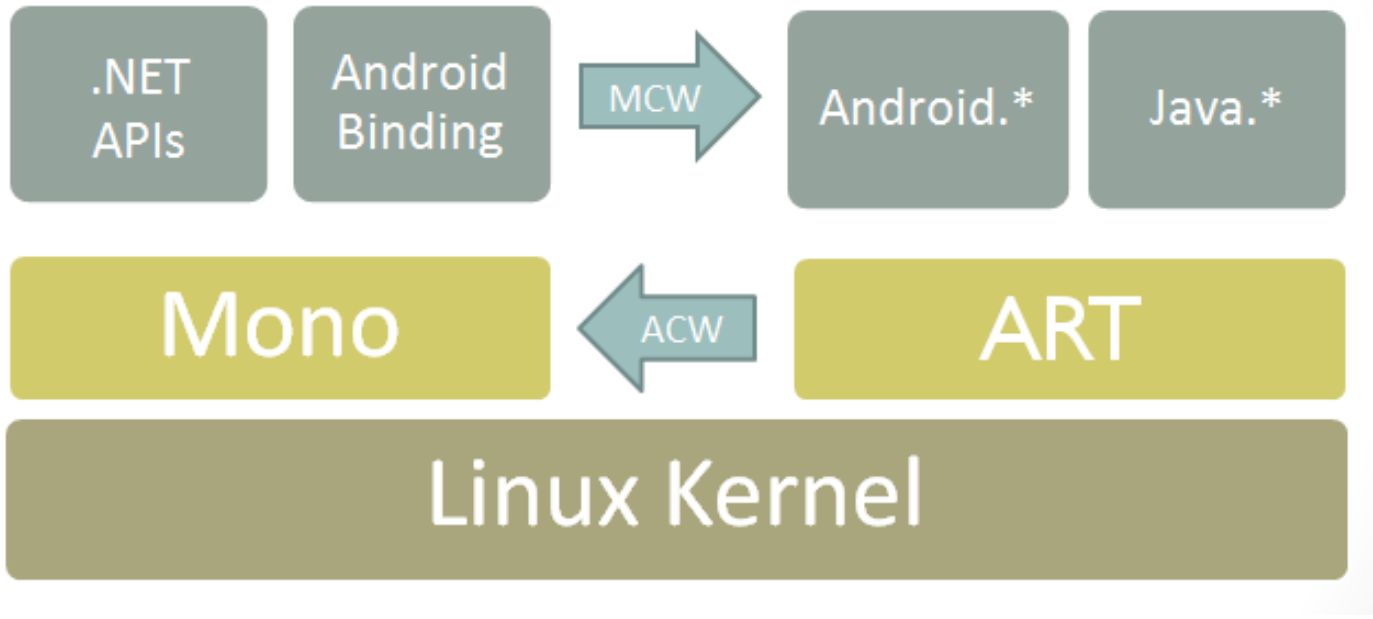
iOS & Android applications


WITH GREAT POWER
COMES GREAT RESPONSIBILITY
memegenerator.net

Xamarin.iOS

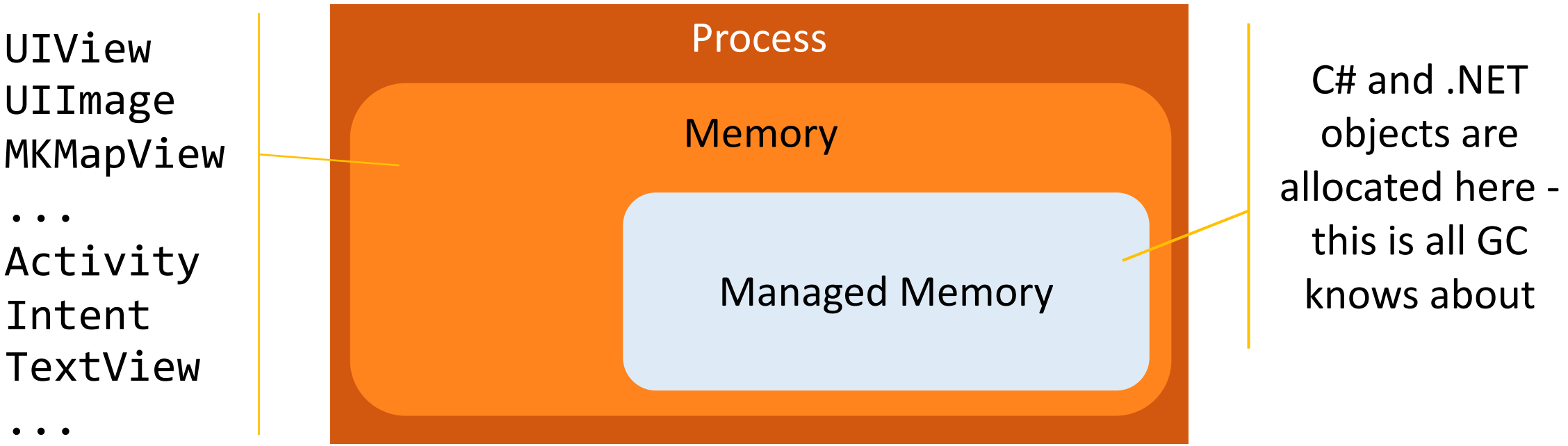

Off-the-shelf framework objects
Custom objects
Model
Data Model Objects
Controller
NSApplication
Application Delegate
Event Loop
Window Controller
View
NSWindow
Views and UI Objects
View controllers
(optional)

Xamarin.Android



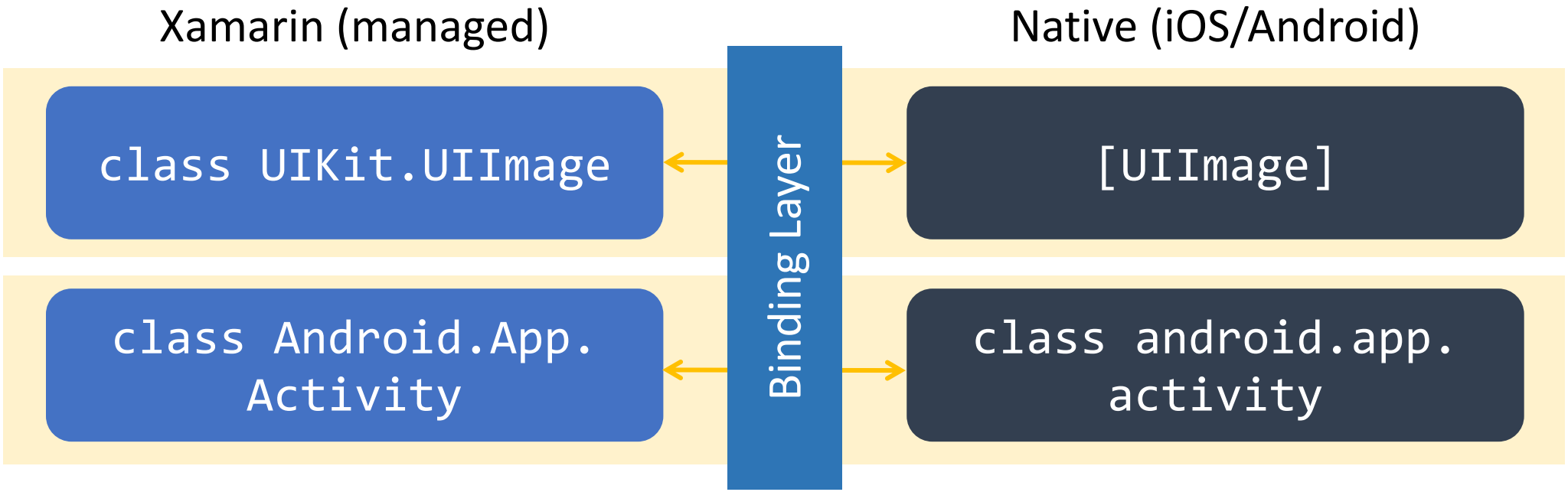
Defining memory

- Your application works with **two types of memory**, both allocated from the same process space; apps must be concerned about both types



Objects in memory

- Some objects live in both the managed and the native world and must be treated properly to ensure that they are freed when the app is finished with them … **but not before**!



Peer object types

- Xamarin.iOS and Xamarin.Android supports *two* types of peer objects:

Framework Peers

User Peers

What is a Framework Peer?

- Peers always call the native object to get or set the state

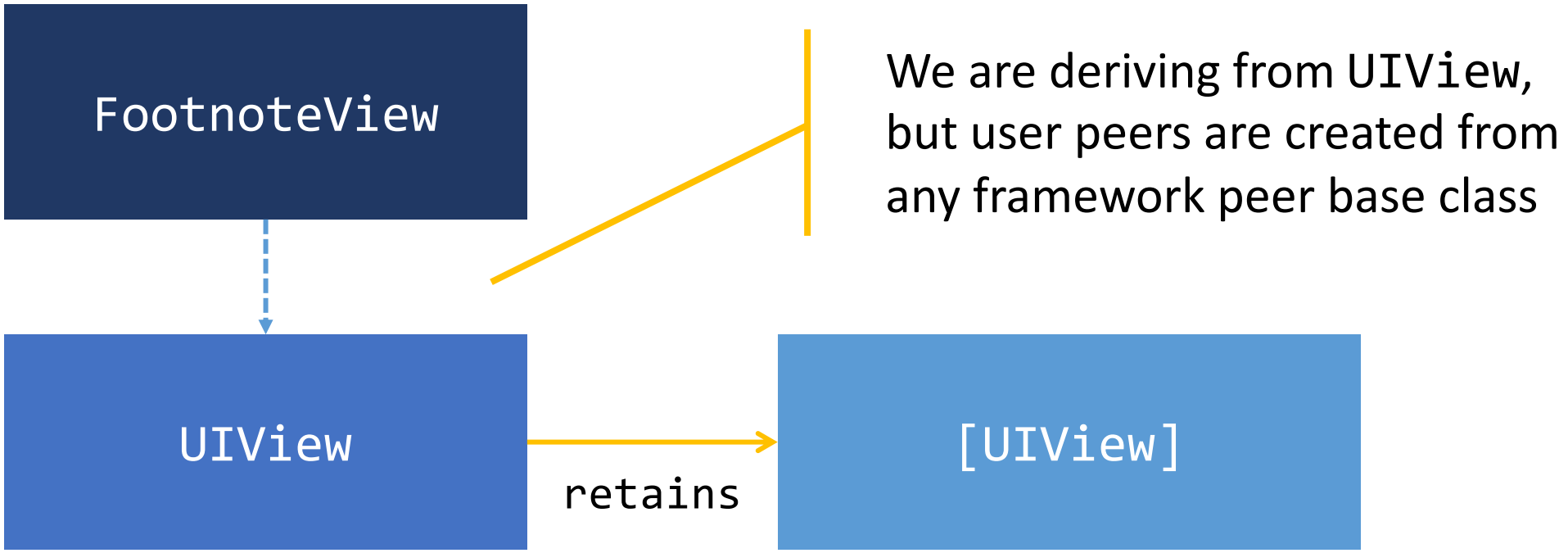
```
UIButton button = new UIButton();
button.SetTitle("Click Me", UIControlState.Normal);
if (button.CurrentTitle == "Click Me") { ... }
```

C#


```
[button setTitle:@"Click Me" forState:UIControlStateNormal]
button.currentTitle
```

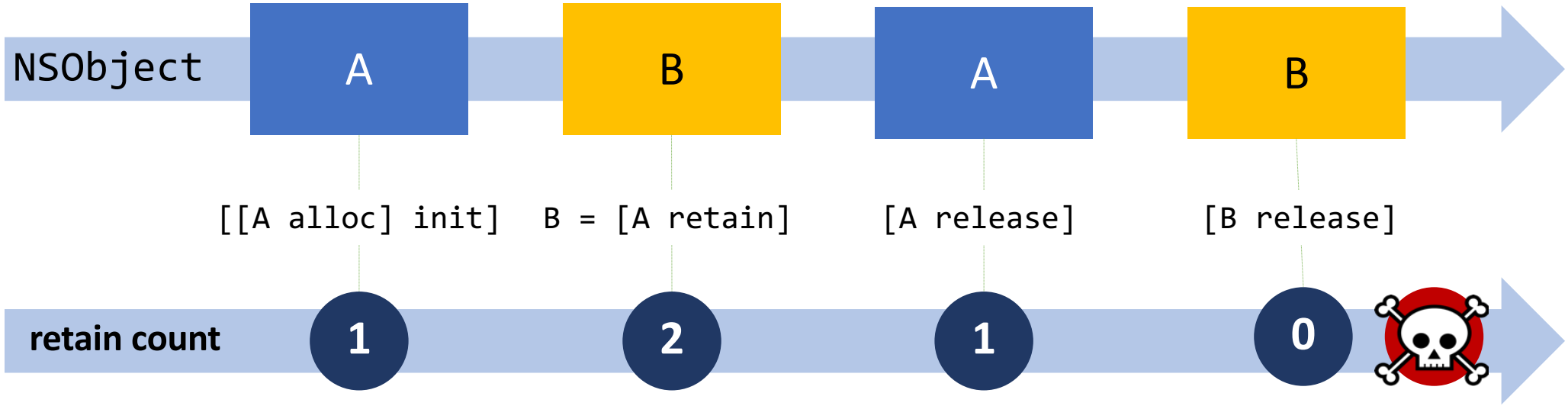
What is a User Peer?

- User peers (sometimes called derived objects) are *custom* managed types which derive from a built-in iOS or Android wrapper



Memory Management in iOS

- iOS uses *reference counting* to manage memory (manual or automatic)



GC Process Xamarin.iOS

- Peer object increment the retain count; it is released when the managed peer is **disposed** or **finalized**

```
var b = new UIButton(...)
```

```
[[UIButton alloc] init]
```

1

…

```
b = null;
```

GC Collect

```
[b release]
```

0

or

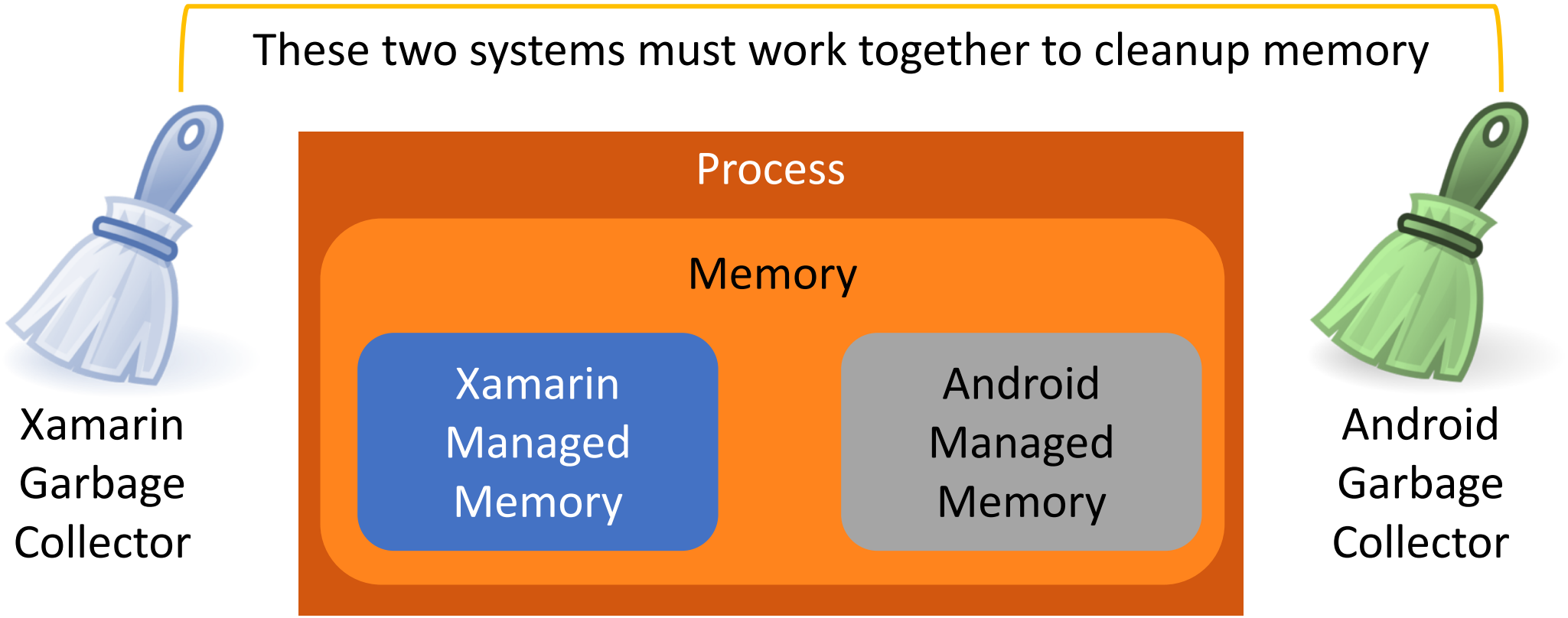
```
b.Dispose();
```

```
[b release]
```

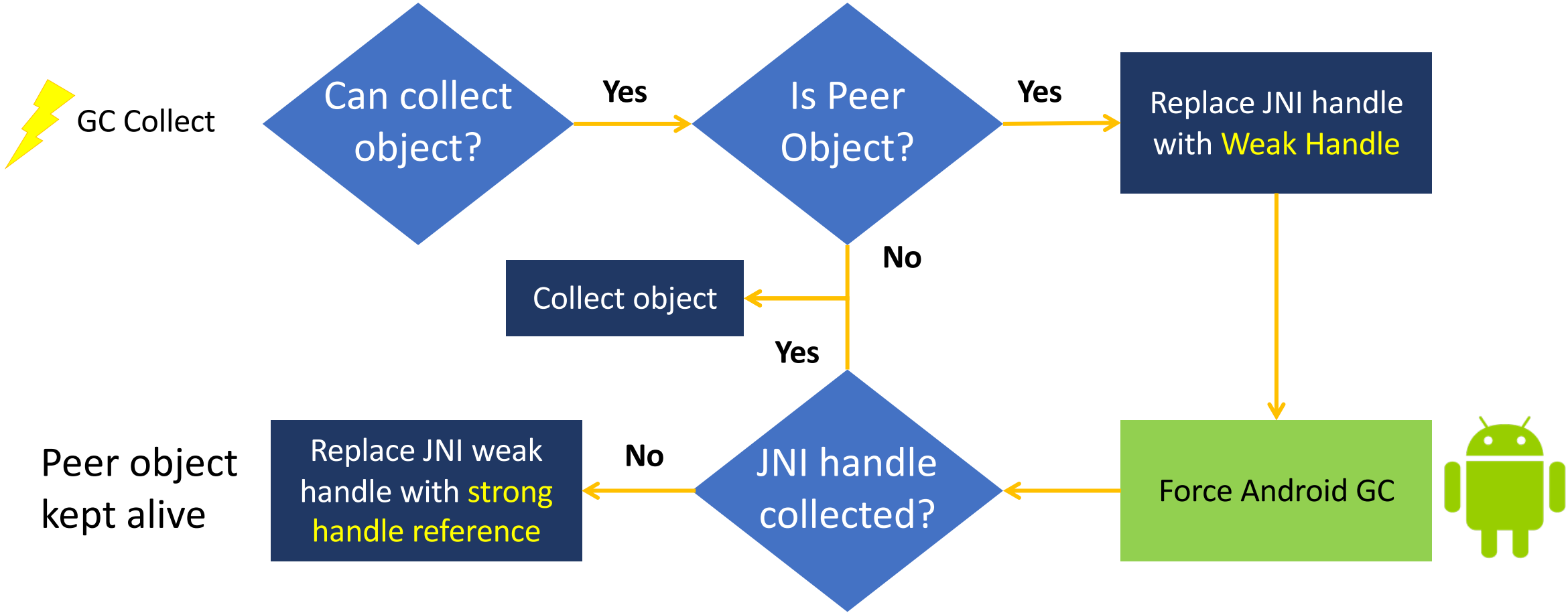
0

Memory Management in Xamarin.Android

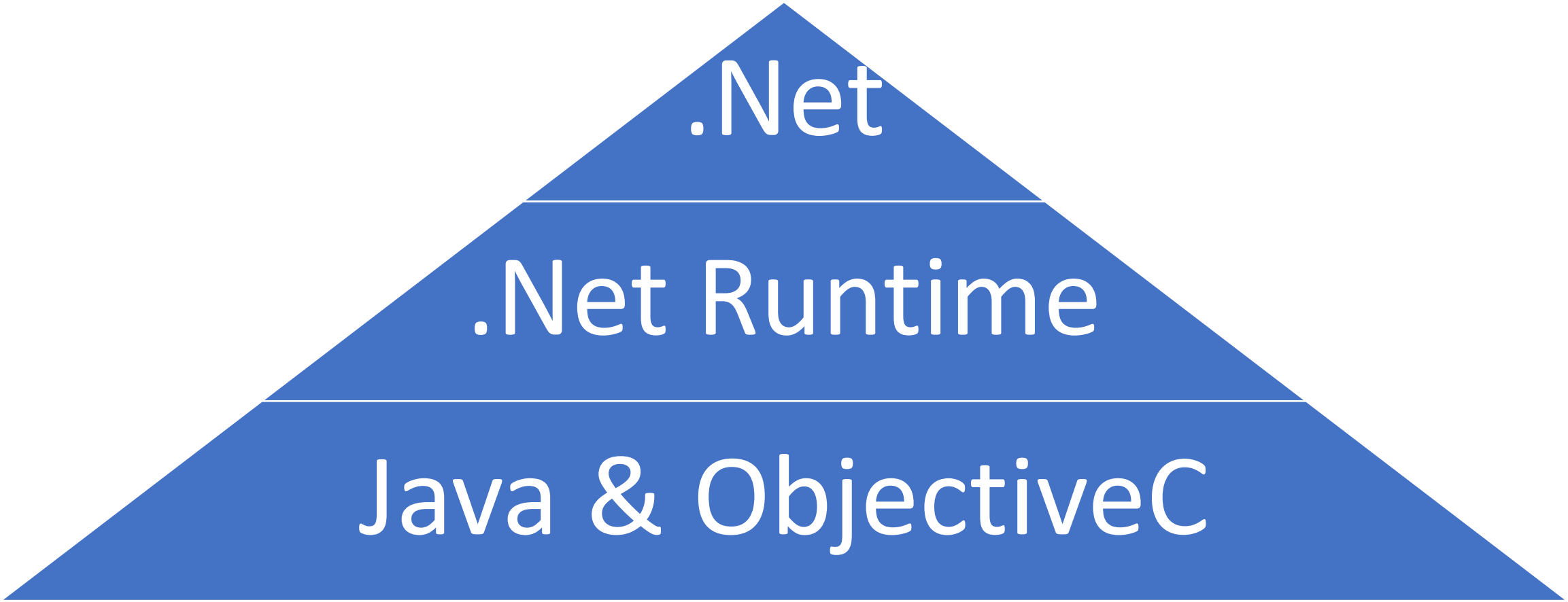
- Android also uses Garbage Collection to clean up resources



GC process Xamarin.Android



Memory Leaks



Monitoring memory allocations

- Several tools you can use to help check the memory allocations in your Xamarin application

Instruments

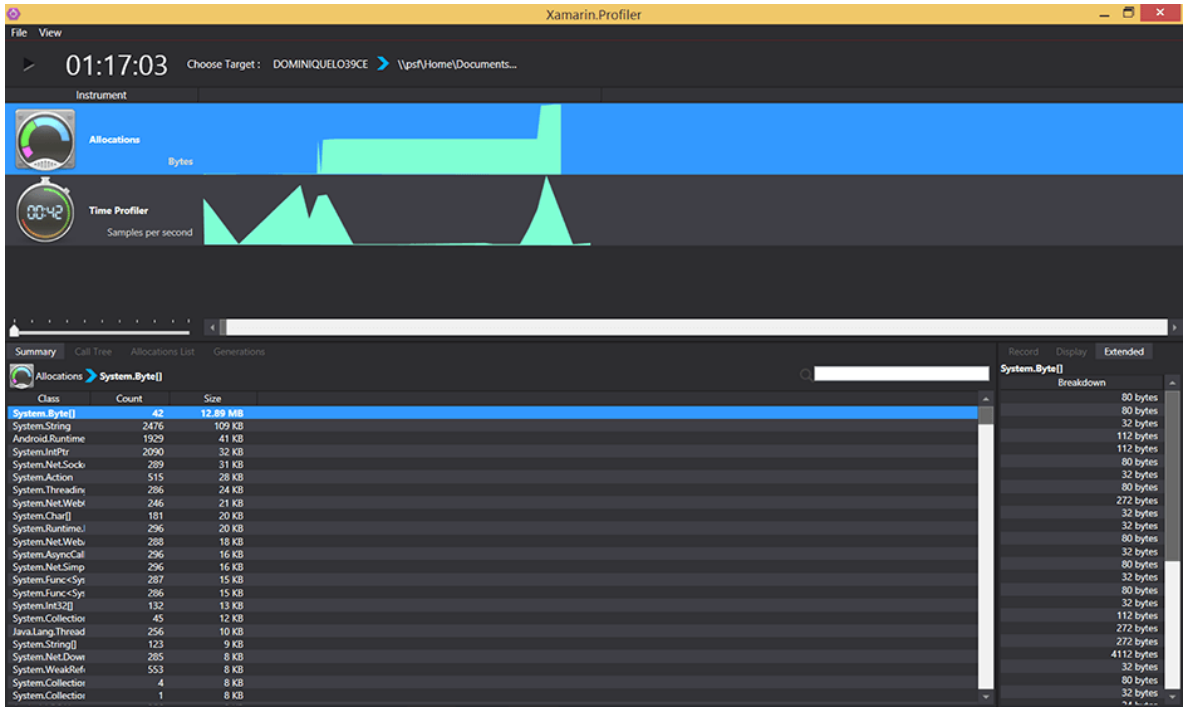
Android Profiler

Xamarin Profiler

Visual Studio Profiling Tools

Xamarin Profiler

- Xamarin Profiler is a managed memory monitor available with an enterprise license that can help identify all the managed objects being allocated and retained in your app



Monitoring memory growth

- GC_MAJOR reports current and previous memory usage for major heap and LOS; watch these values to identify potential leaks

```
GC_MAJOR: (user request) pause 6.53ms, total 6.54ms, bridge 0.00ms
major 1008K/16912K los 64K/0K
```

How much memory is currently in use by the major generation

How much memory *was* in use prior to this collect?

Checking for a leak

- Can use `WeakReference` as a diagnostic tool if you suspect an object is not being collected when you expect it to

```
DataLoader dl = new DataLoader ();
WeakReference wr = new WeakReference (dl);
...
dl = null;
...
GC.Collect ();
if (wr.IsAlive) {
    Debug.WriteLine ("DataLoader still alive");
}
```

DEMOS

Advices

- Prefer full delegate methods over lambdas – it makes it easier to see and understand strong references

- Call `Dispose()` to release native resources immediately (vs. waiting on a GC) when you are finished with a peer wrapper

- Always unsubscribe from events you manually wire up; alternatively, use the Storyboard to connect events which can then be cleaned up automatically

Q & A

Thank You!