Xpirit

Breaking through the abstractions

Xamarin hands-on-labs

Xamarin.Forms elements

❖ Xamarin.Forms allows you to define your UI using a set of elements that are common across all platforms

Button is available everywhere →

```
public class Button : Element
{
    public Color  BorderColor  { get; set; }
    public int    BorderRadius { get; set; }
    public double BorderWidth  { get; set; }
    public string Text         { get; set; }
    public Color  TextColor    { get; set; }
    ...
}
```

Xamarin.Forms elements are models

- Elements provide a *representation* of the UI we want to create and display

Properties let you customize runtime visuals and behavior

```
public class Button : Element
{
    public Color  BorderColor  { get; set; }
    public int    BorderRadius { get; set; }
    public double BorderWidth  { get; set; }
    public string Text         { get; set; }
    public Color  TextColor    { get; set; }
    ...
}
```

Customizing elements

- Changing the properties of Xamarin.Forms elements allows for limited customization – which may or may not be sufficient for your needs

Can change most colors

Can adjust position + width/height

Can add background images into views

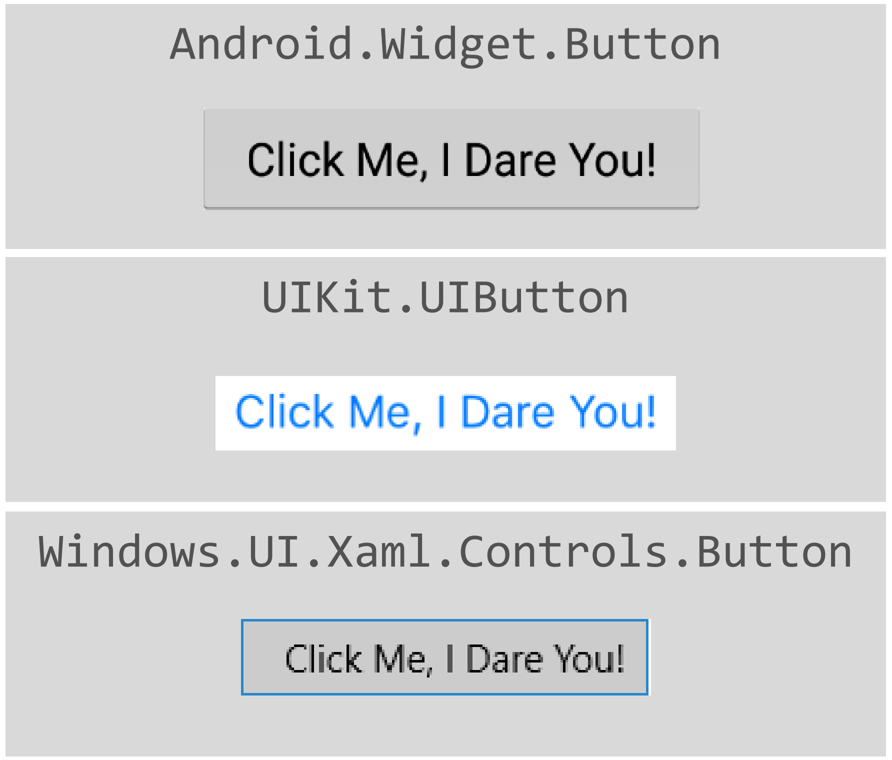
Can control fonts

From Element to Visual

- At runtime, a platform-specific control is created to visualize each Xamarin.Forms element

```
public class Button : Element
{
    public Color  BorderColor  { get; set; }
    public int    BorderRadius { get; set; }
    public double BorderWidth  { get; set; }
    public string Text         { get; set; }
    public Color  TextColor    { get; set; }
    ...
}
```

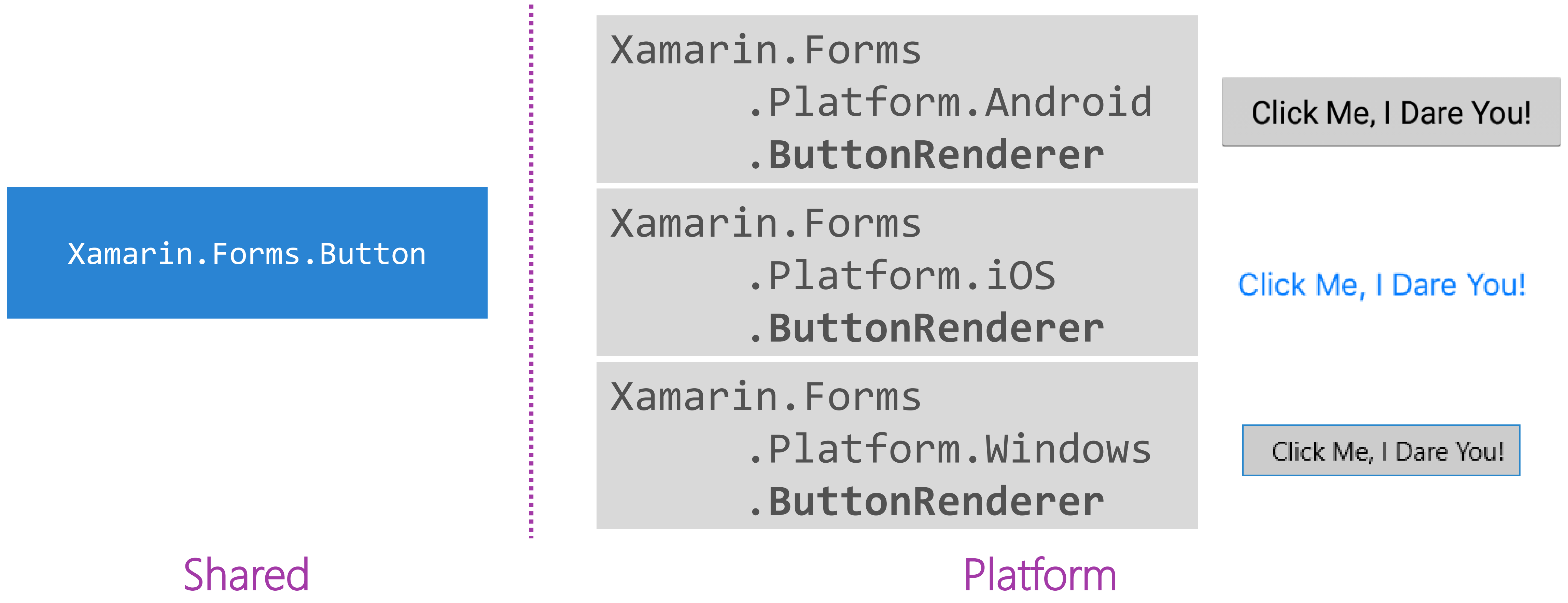
Shared



Platform

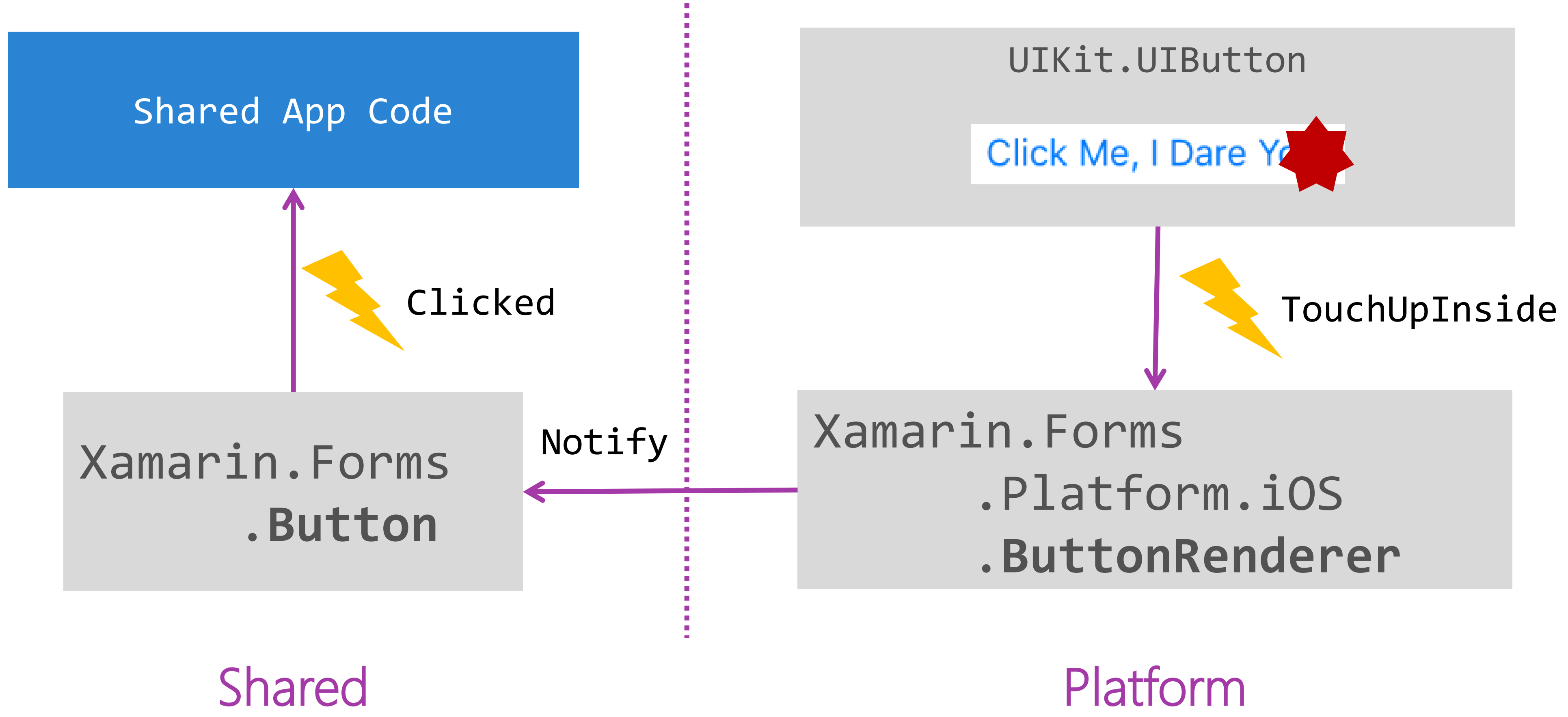
Platform renderers

- The *platform renderer* is the code that translates Xamarin.Forms elements to a platform native control

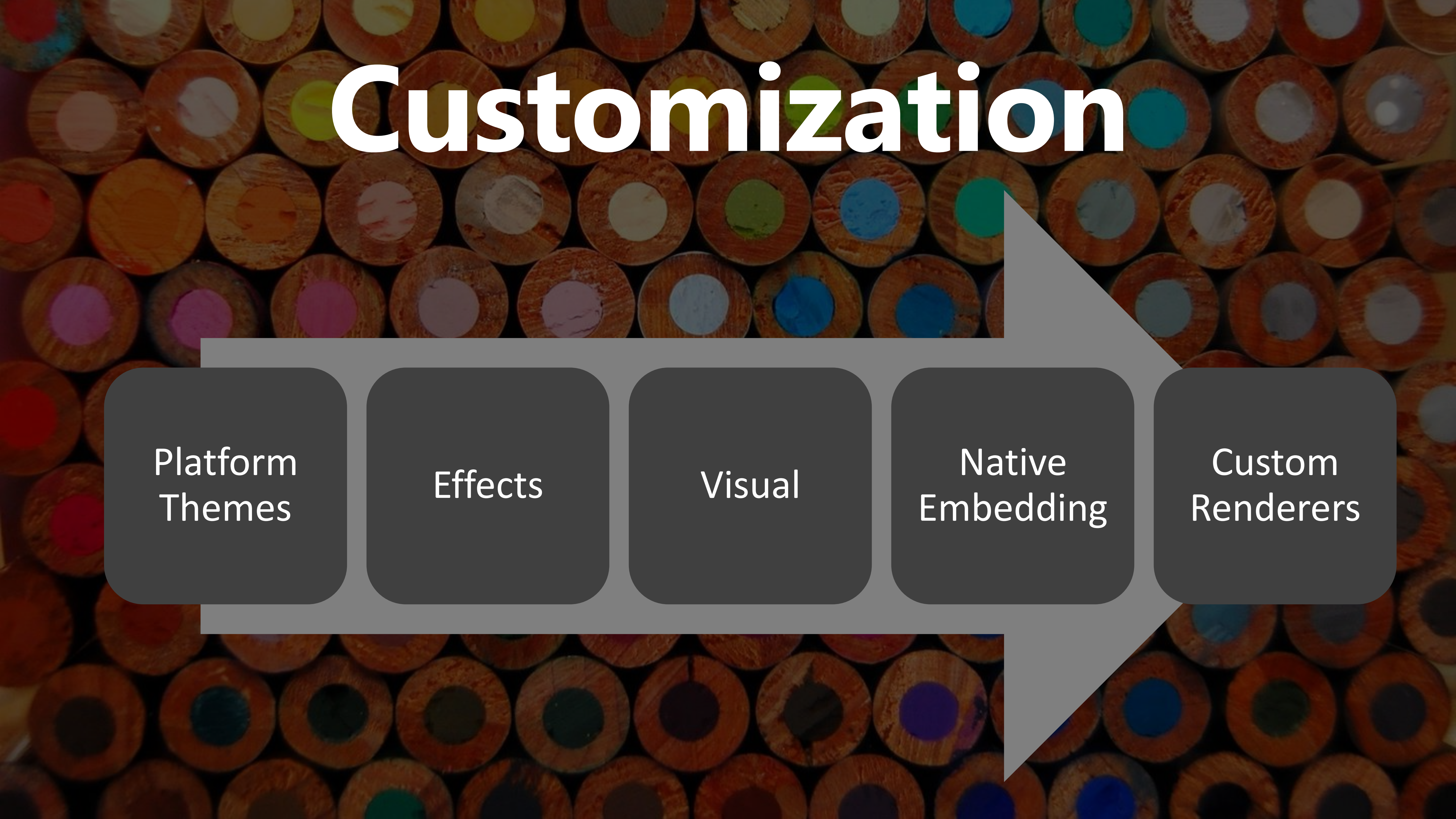

Xamarin.Forms.Button
Xamarin.Forms .Platform.Android .ButtonRenderer
Click Me, I Dare You!
Xamarin.Forms .Platform.iOS .ButtonRenderer
Click Me, I Dare You!
Xamarin.Forms .Platform.Windows .ButtonRenderer
Click Me, I Dare You!
Shared
Platform

From Visual to Element

❖ The renderer is responsible for **watching** the native control notifications and **forwarding** them to the Xamarin.Forms element



Customization



- Platform Themes
- Effects
- Visual
- Native Embedding
- Custom Renderers

Platform Themes

- Each platform has an API you can use to control the native visual appearance of your app

`UIAppearance API`

`android:theme`

`Style + ControlTemplate`

Style + ControlTemplate

- Each Windows XAML control has a default style and control template – these can be modified to customize appearance and behavior

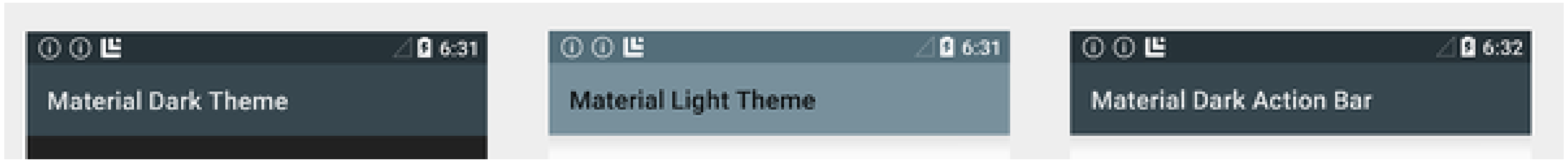
```xml
<?xml version="1.0" encoding="UTF-8" ?>
<Application.Resources>
   <Style TargetType="TextBlock">
      <Setter Property="Foreground" Value="Yellow" />
      <Setter Property="FontFamily" Value="Verdana" />
      <Setter Property="FontSize" Value="96" />
   </Style>
</Application.Resources>
```

Hello World

Native Windows **Style**s will affect controls created by the Xamarin.Forms renderer

Themes

❖ Android Themes determine the look and feel of views and activities; there are built in themes and you can create custom themes

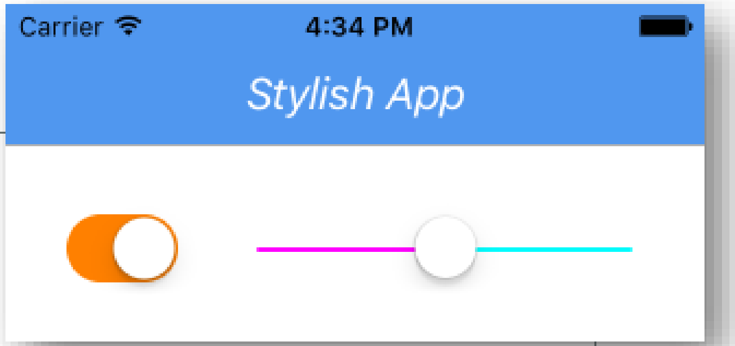


```
[Activity (Label = "DroidThemes",
           Theme = "@android:style/Theme.Material.Light",
           MainLauncher = true, Icon = "@mipmap/icon")]
public class MainActivity : Activity
{
    ...
}
```

Appearance API

❖ The iOS Appearance API lets you define visual settings at a class level that apply to all instances of that type

```
public override bool FinishedLaunching(...)
{
      UISwitch.Appearance.OnTintColor = UIColor.Orange;
      UISlider.Appearance.MinimumTrackTintColor = UIColor.Magenta;
      UISlider.Appearance.MaximumTrackTintColor = UIColor.Cyan;

      UINavigationBar.Appearance.BarTintColor = UIColor.FromRGB(51, 134, 238);
      UINavigationBar.Appearance.SetTitleTextAttributes(new UITextAttributes()
      { TextColor = UIColor.White, Font = UIFont.ItalicSystemFontOfSize(20)});
}
```



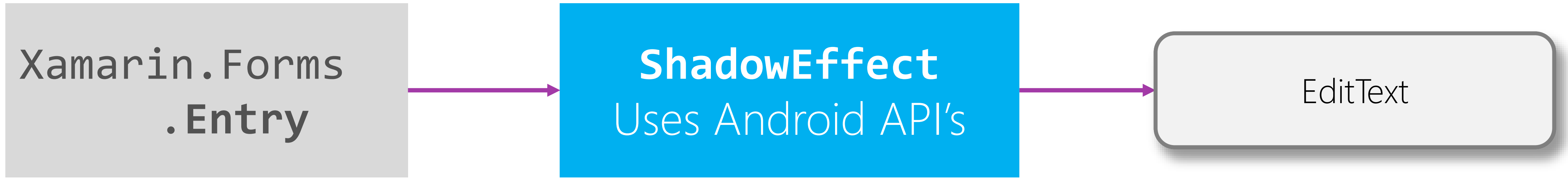
Effects

- ❖ The **Effects API** lets your code *tweak* the visual appearance and behavior of the native controls generated by the renderer

- Change properties not exposed by X.F.
- Access platform features (e.g. shadows)
- Handle native control notifications
- Add or remove visual children

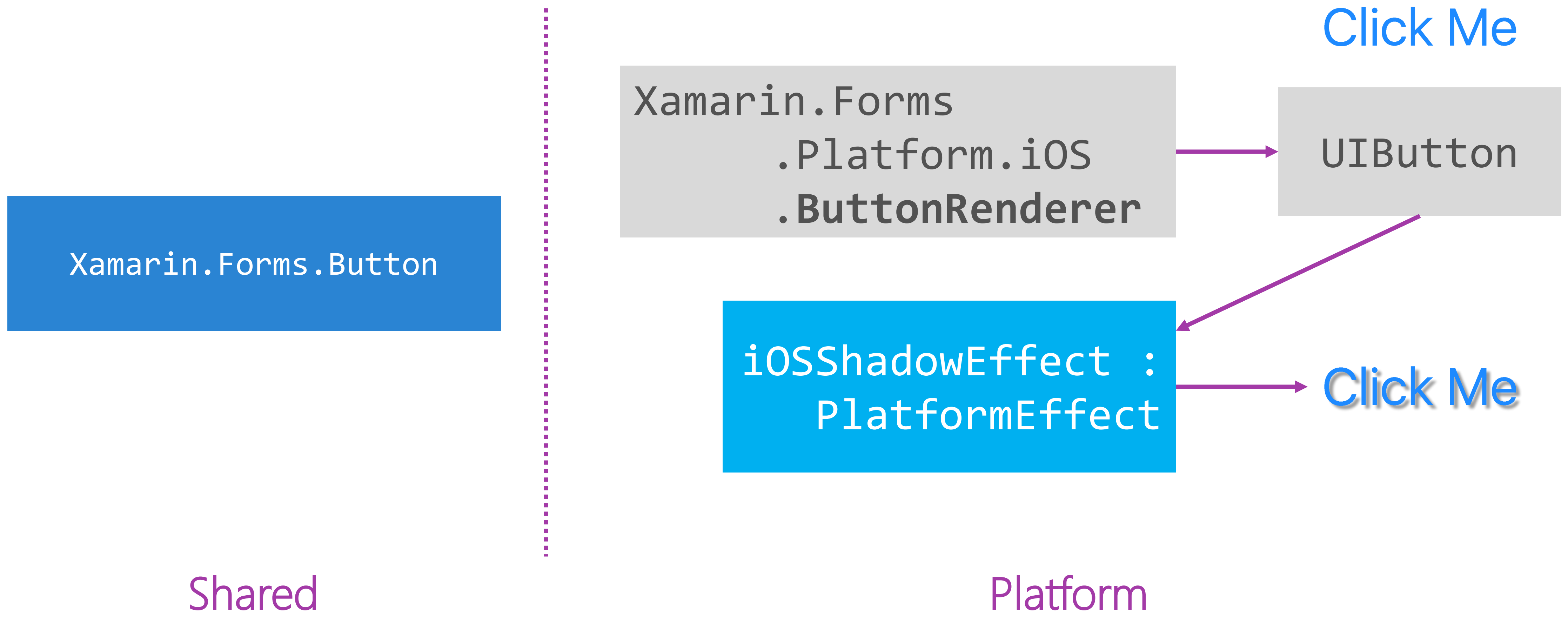
What is an Effect?

- An *effect* is a platform-specific class that uses the native APIs to change the appearance and behavior of the native control that underlies a Xamarin.Forms Element



Xamarin.Forms Effects API

❖ The Effects API allows you to interact with and change properties on the controls created by the native renderers



Xamarin.Forms.Button
Shared
Xamarin.Forms .Platform.iOS .ButtonRenderer
UIButton
Click Me
iOSShadowEffect : PlatformEffect
Click Me
Platform

One effect per platform

- The author of an effect implements one class for each platform they choose to support

```
public class ShadowEffect : RoutingEffect
{
    ...
}
```

```
public class AndroidShadowEffect :  ...
{
    ...
}
```

```
public class iOSShadowEffect :  ...
{
    ...
}
```

```
public class UWPShadowEffect :  ...
{
    ...
}
```

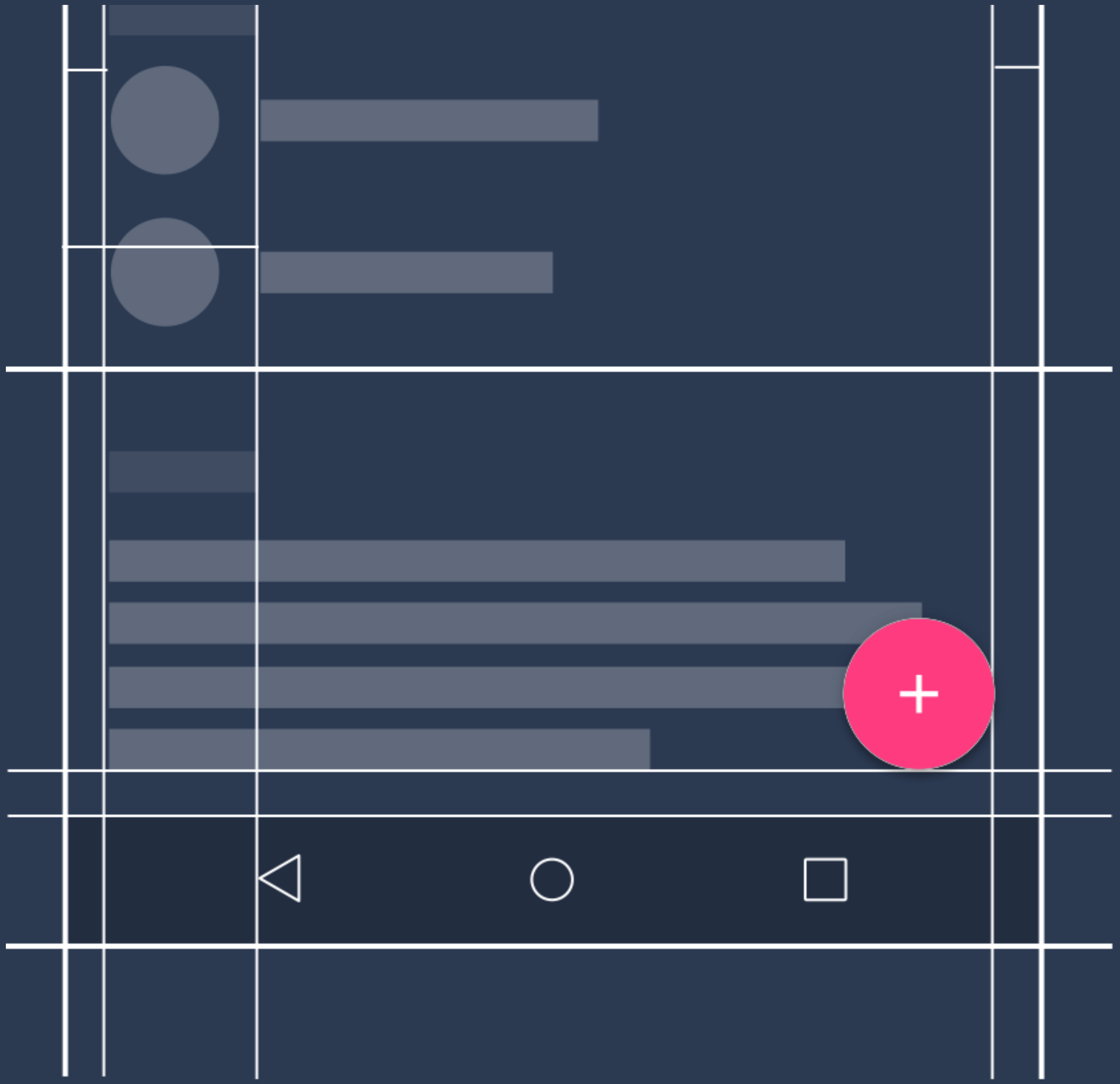
Shared

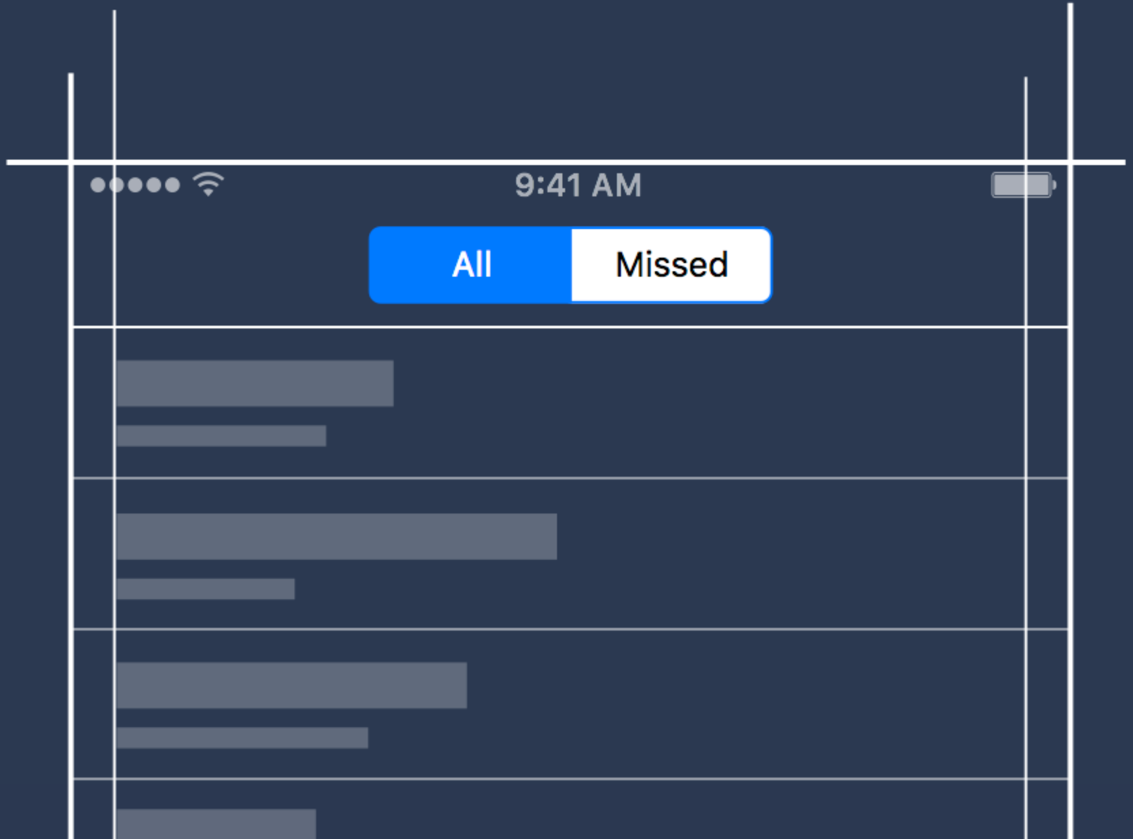
Platform

Effect implementation

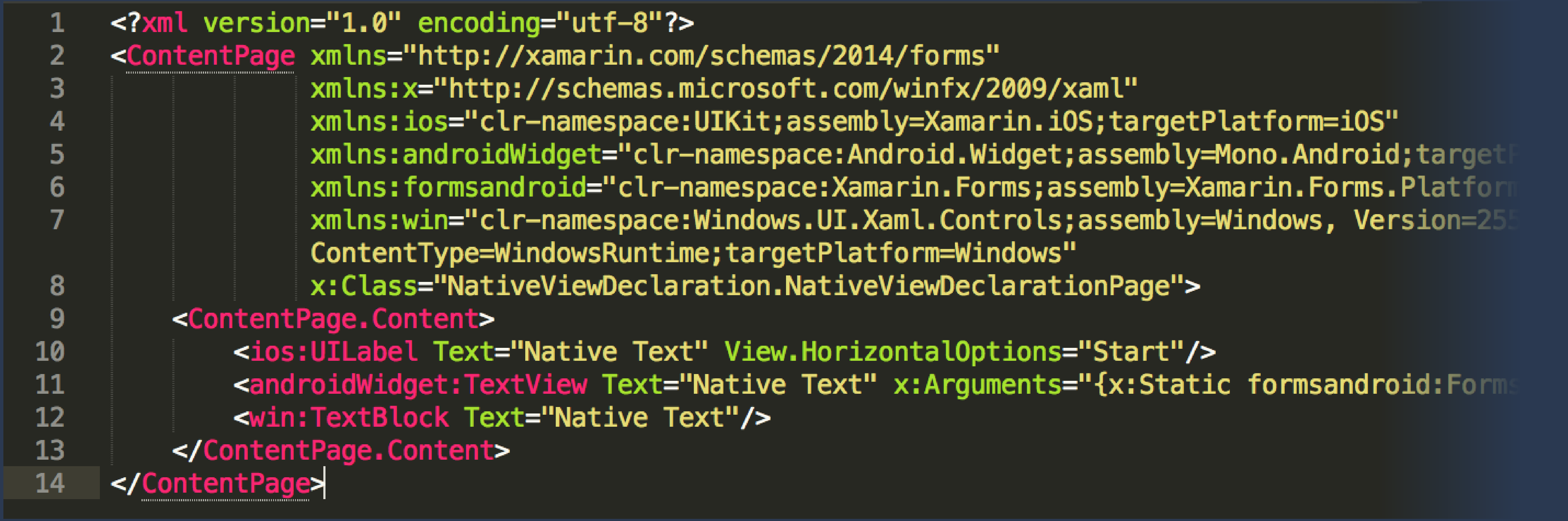
```
using System.ComponentModel;
using EffectsTest.iOS;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;

namespace EffectsTest.iOS
{
    [assembly:ResolutionGroupName("MyCompany")] // avoid naming conflicts
    [assembly:ExportEffect(typeof(MyEffect), nameof(MyEffect))]
    public class MyEffect: PlatformEffect
    {
        protected override OnAttached()
        {
        }
        protected override OnDetached()
        {
        }
        protected override OnElementPropertyChanged(PropertyChangedEventArgs args)
        {
        }
    }
}
```

Native Embedding





Native Embedding

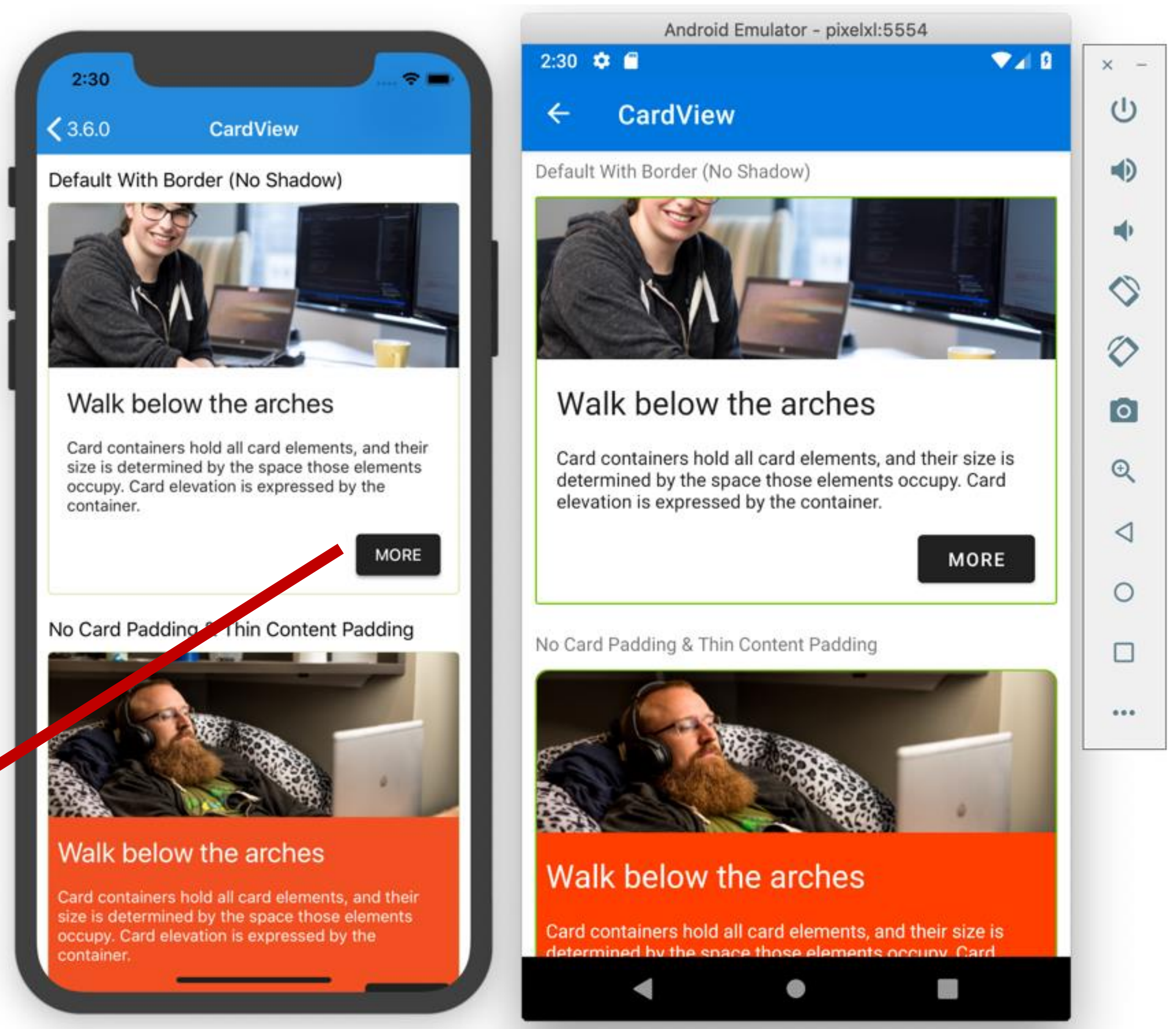


```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:ios="clr-namespace:UIKit;assembly=Xamarin.iOS;targetPlatform=iOS"
             xmlns:androidWidget="clr-namespace:Android.Widget;assembly=Mono.Android;target
             xmlns:formsandroid="clr-namespace:Xamarin.Forms;assembly=Xamarin.Forms.Platfor
             xmlns:win="clr-namespace:Windows.UI.Xaml.Controls;assembly=Windows, Version=25
             ContentType=WindowsRuntime;targetPlatform=Windows"
             x:Class="NativeViewDeclaration.NativeViewDeclarationPage">
    <ContentPage.Content>
        <ios:UILabel Text="Native Text" View.HorizontalOptions="Start"/>
        <androidWidget:TextView Text="Native Text" x:Arguments="{x:Static formsandroid:Forms
        <win:TextBlock Text="Native Text"/>
    </ContentPage.Content>
</ContentPage>
```

Visual: pre-made set of custom renderers

- Nuget: Xamarin.Forms.Visual.Material
- Requires target Android 9.0 (Pie)
- Leverages bindings to official Google Material Components for iOS

```
Xamarin.Forms
        .Visual.Material.iOS
        .MaterialButtonRenderer
```



Default With Border (No Shadow)

Walk below the arches

Card containers hold all card elements, and their size is determined by the space those elements occupy. Card elevation is expressed by the container.

No Card Padding & Thin Content Padding

Walk below the arches

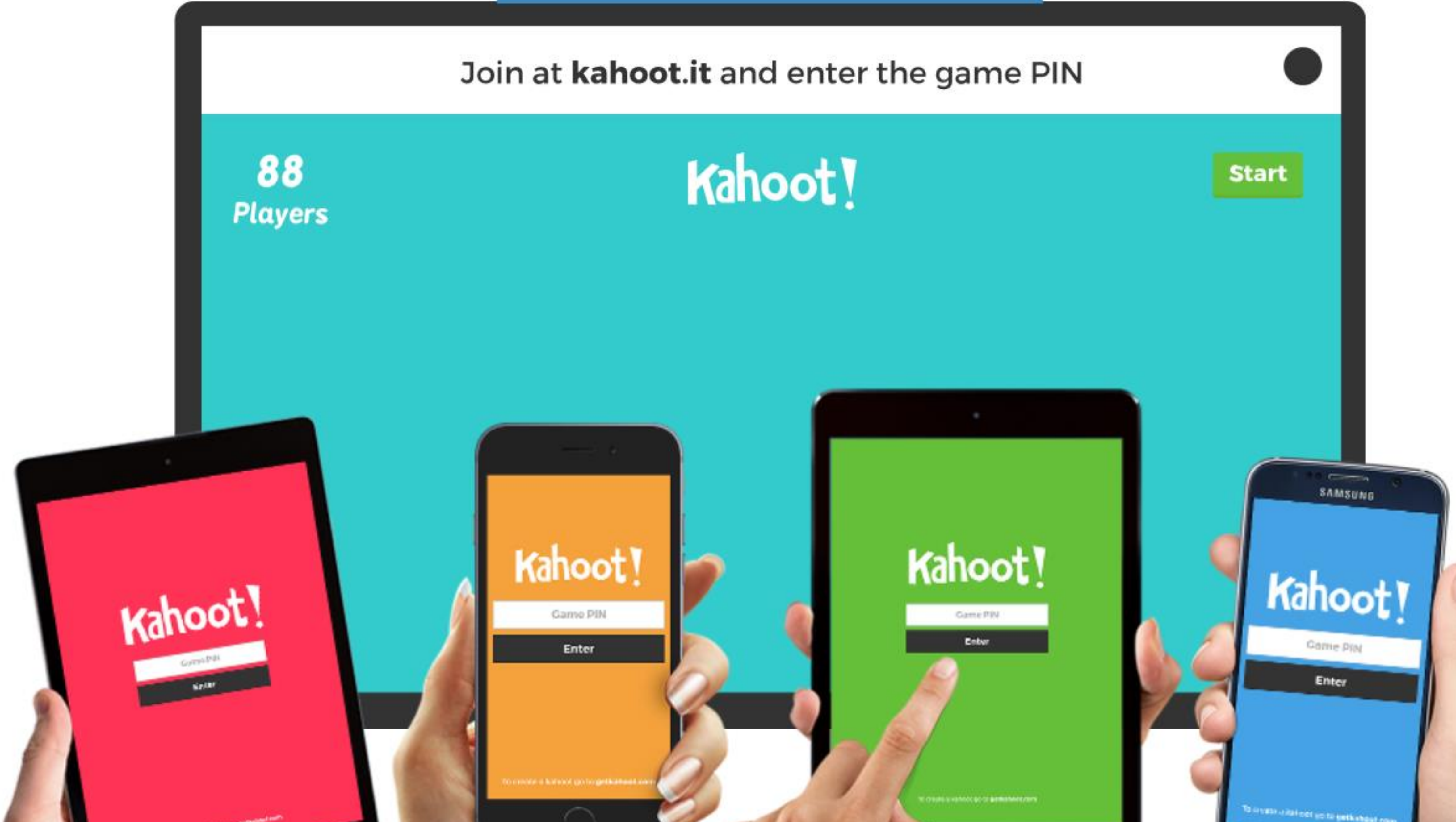
Card containers hold all card elements, and their size is determined by the space those elements occupy. Card elevation is expressed by the container.

Applying the Material Visual

- Add Xamarin.Forms.Visual.Material Nuget Package
- Add **`Xamarin.Forms.FormsMaterial.Init();`** to startup
- Use **`Visual`** attribute in XAML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<StackLayout>
    <Button Text="Normal button" />
    <Button Text="Material button" Visual="Material" />
</StackLayout>
```

Ready to join?



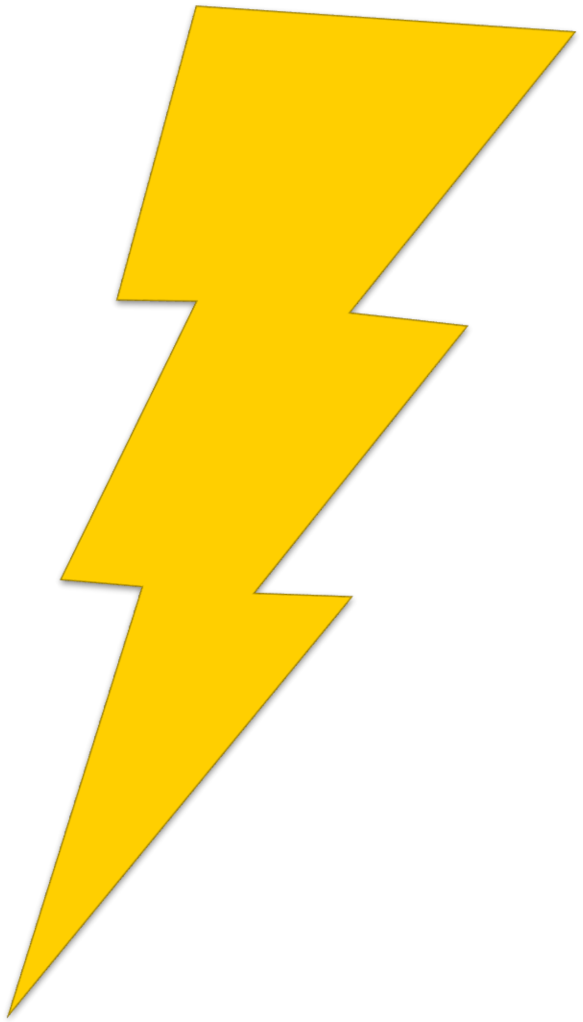
Other useful features in Xamarin.Forms

Recent features

- Visual State Manager
- Performance
- Right-to-Left
- FlexLayout
- Xamarin.Forms Embedding
- Visual

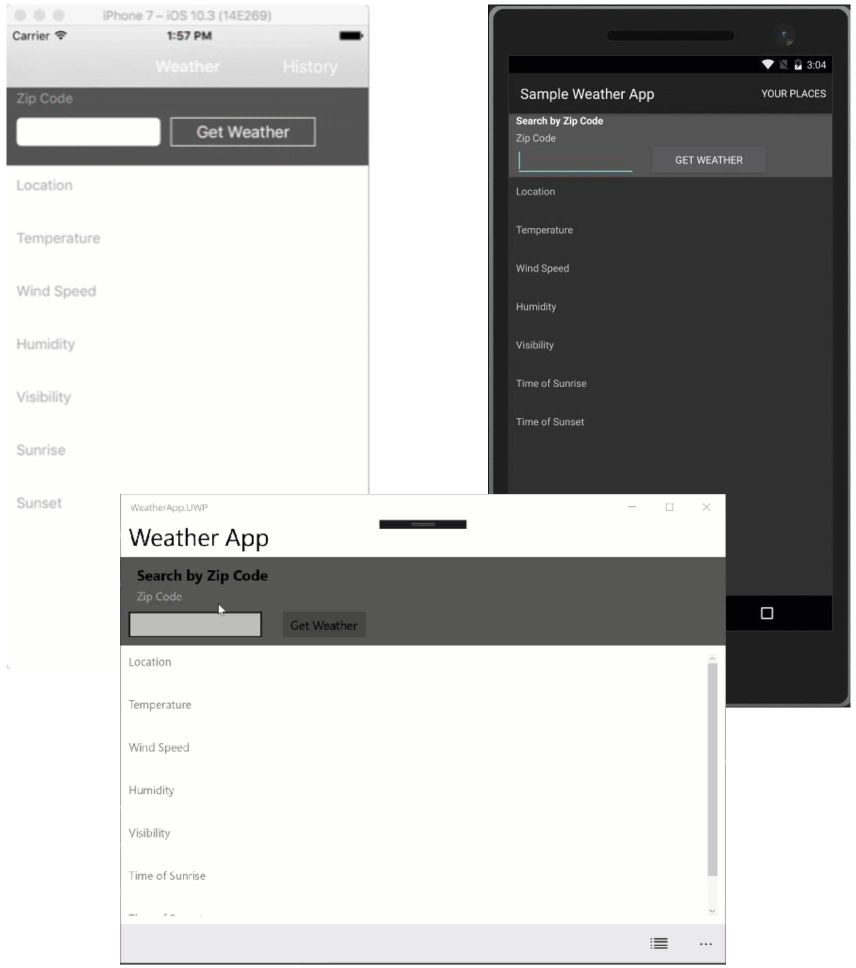
Performance

- .NET Standard 2.0
- Compiled Bindings
- Fast Renderers
- Layout Compression
- Startup Optimizations
- XAMLC improvements



Xamarin.Forms Embedding

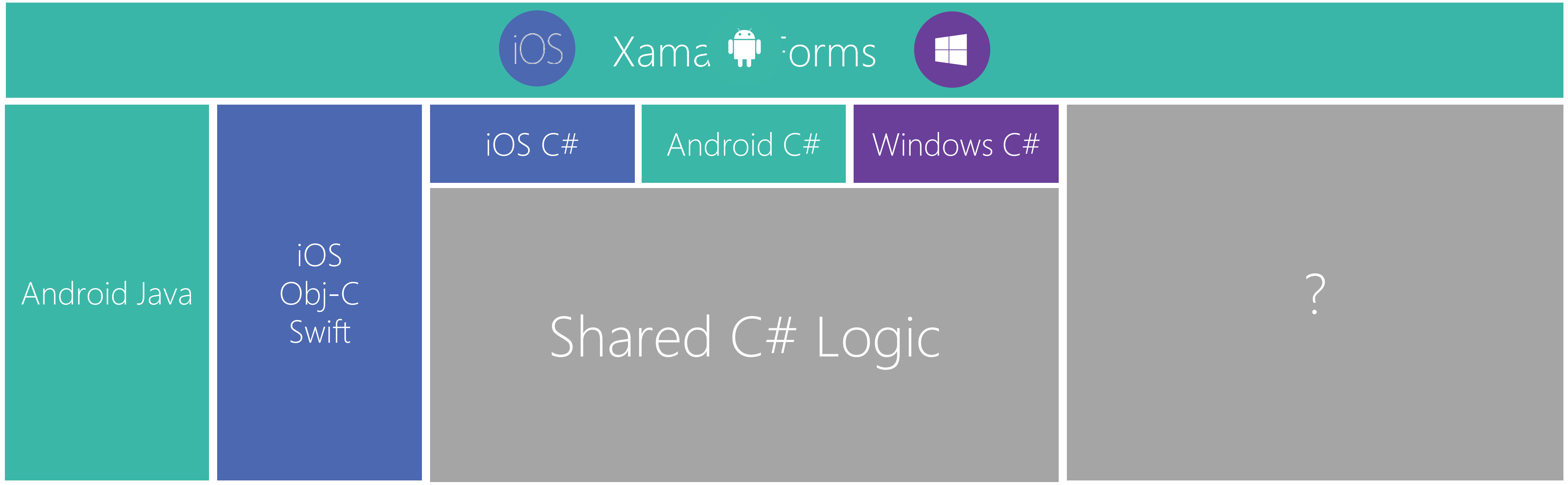
- Easily embed any page into a Xamarin Native Application



```
// Android
Forms.Init(this, null);
var androidFragment = new MyFormsPage().CreateFragment(this);

// iOS
Forms.Init()
var iosViewController = new MyFormsPage().CreateViewController();

// UWP
Forms.Init(e);
var uwpElement = new MyFormsPage().CreateFrameworkElement();
```

Embedding

- Works on ContentPages
- Full support for DependencyService and MessagingCenter

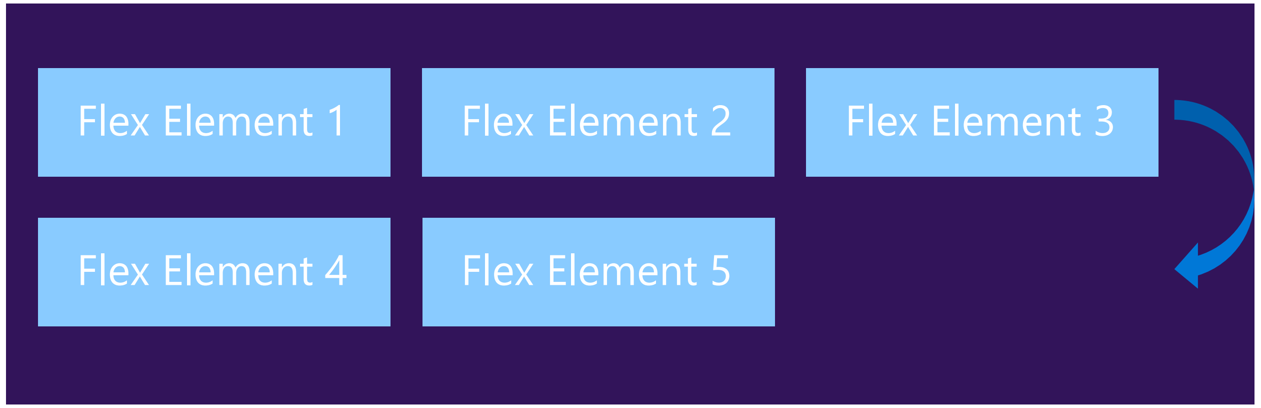


FlexLayout

- A CSS FlexBox inspired layout system
- Used for
  - Flowing items
  - Adaptive layout

FlexLayout Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
         xmlns:x=http://schemas.microsoft.com/winfx/2009/xaml
         x:Class="FormsFlexLayoutDemo.FlexDemoPage">
    <FlexLayout x:Name="flexbox”>
        <Label Text="Flex Element 1" />
        <Label Text="Flex Element 2" />
        <Label Text="Flex Element 3" />
        <Label Text="Flex Element 4" />
        <Label Text="Flex Element 5" />
    </FlexLayout>
</ContentPage>
```



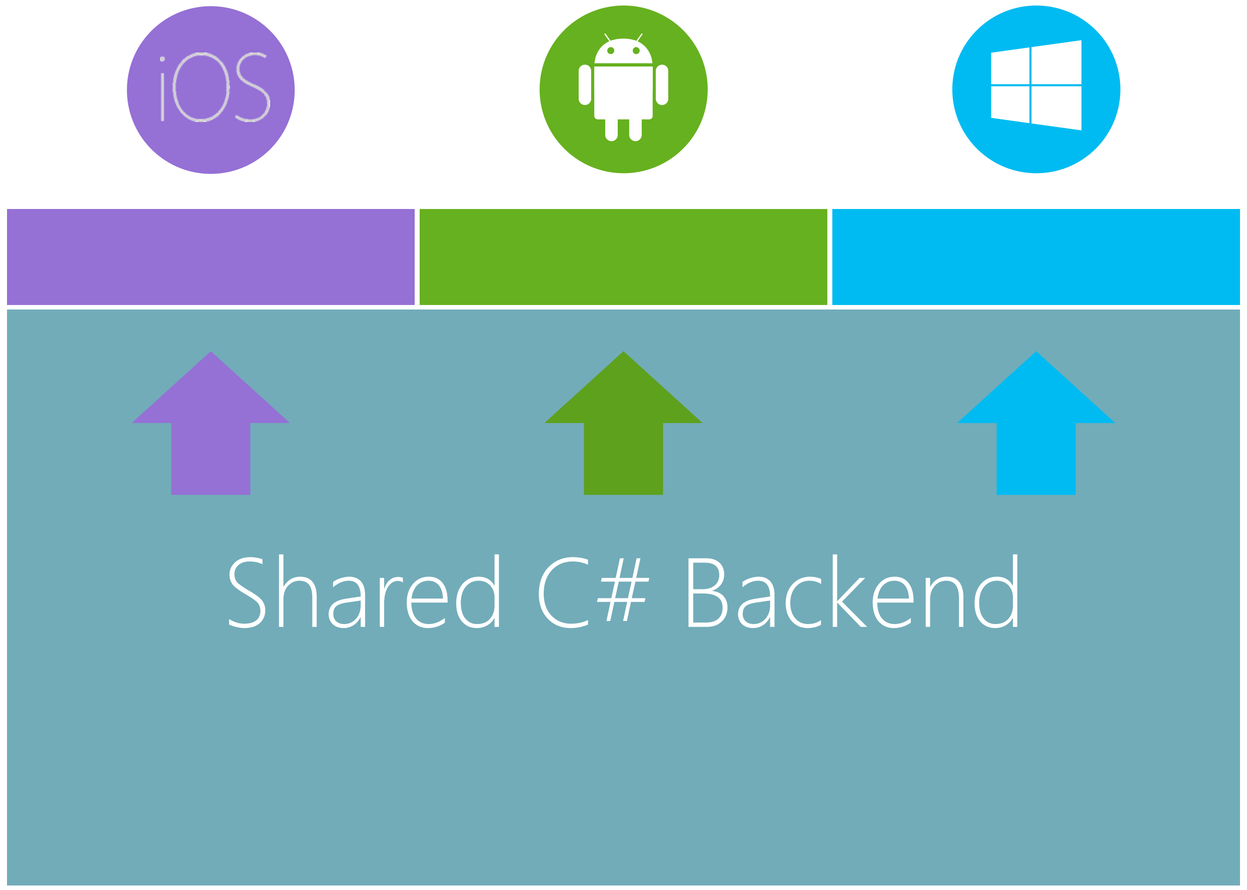
Other platforms

- Samsung Tizen
  - Televisions, Wearables, Mobile
- macOS
- WPF
- Linux: GTK#

Using an effect, platform themes and Visual

Lab - app-quotes

Lab04 – exercises 1 - 4

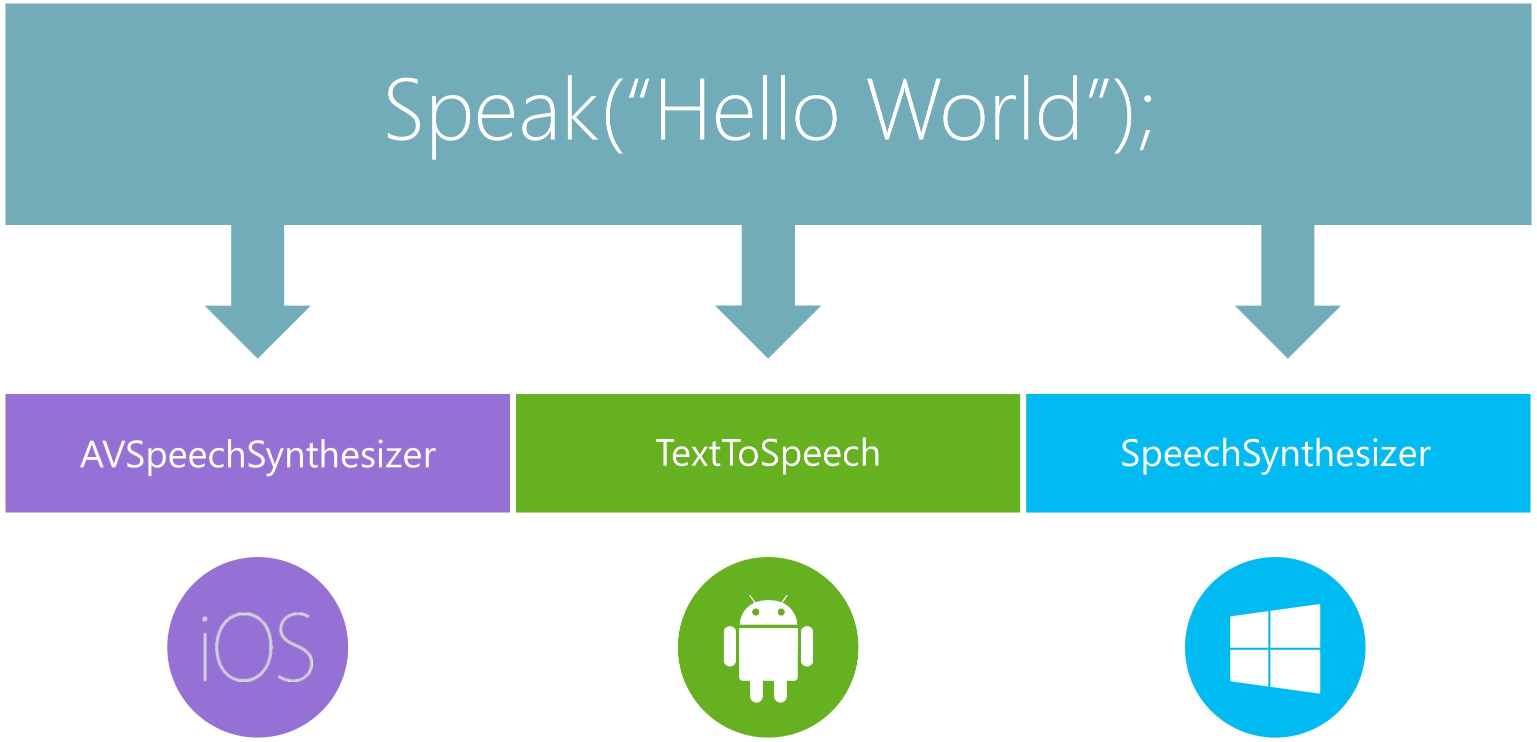

iOS
Shared C# Backend

Platform Specific Code

UI+APIs	UI + APIs	UI + APIs
Battery	Battery	Battery
GPS	GPS	GPS
Lights	Lights	Lights
Notifications	Notifications	Notifications
Settings	Settings	Settings
Text To Speech	Text To Speech	Text To Speech

What if we didn't have to write this code?

What if we could access it from shared code?


Speak("Hello World");
AVSpeechSynthesizer
TextToSpeech
SpeechSynthesizer
iOS

Plugins for Xamarin & Windows

Xamarin.com/plugins

https://docs.microsoft.com/en-us/xamarin/essentials

Common API

Adding Text-To-Speech

Lab – app-quotes

Lab04 – exercises 5 & 6