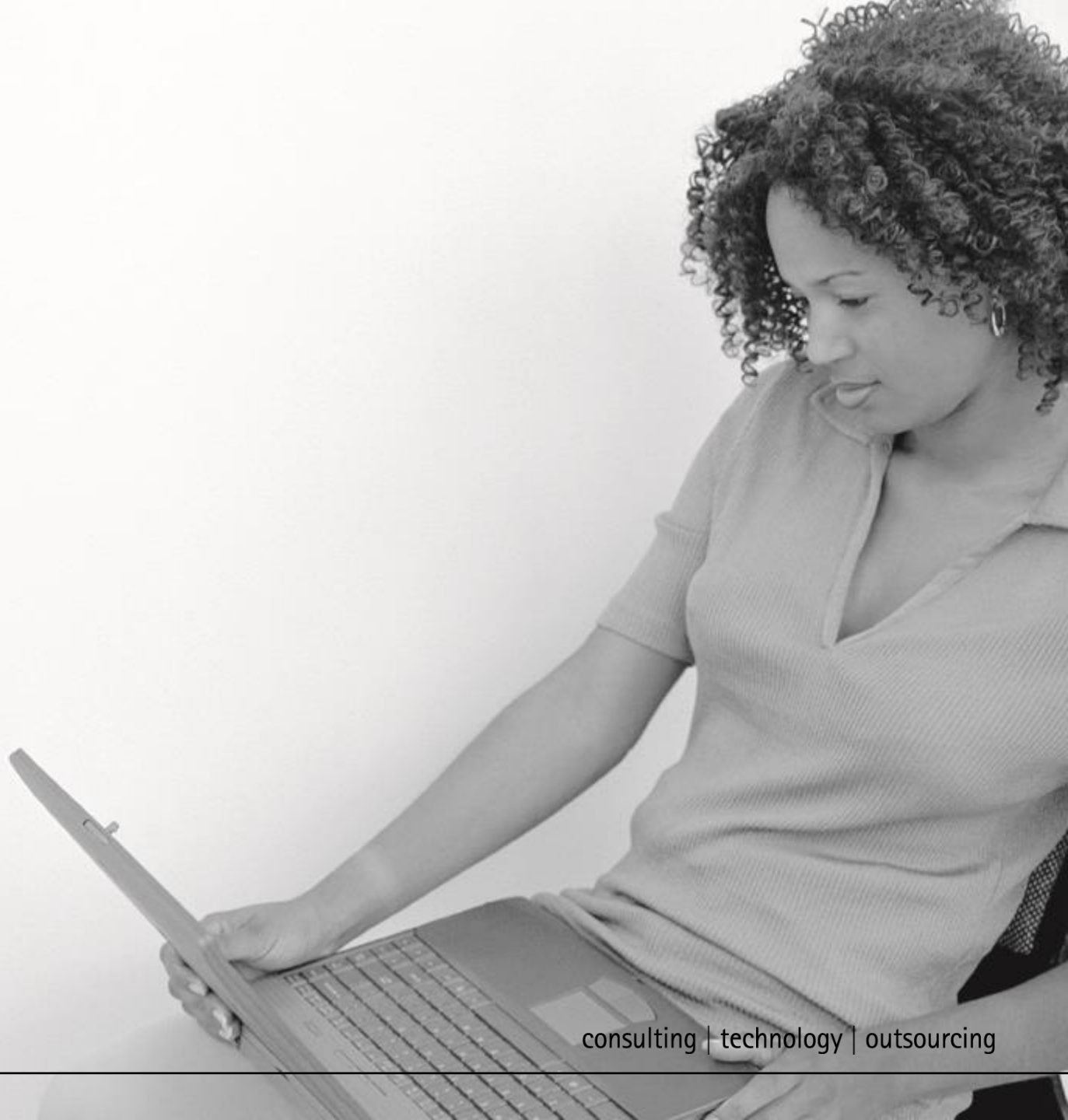
Xamarin

Parameswari.E



Goals

- Introduction to Xamarin
  - Xamarin vs. Hybrid Framework vs. Native Framework
  - Xamarin Development IDE – Visual Studio and Xamarin Studio
  - Xamarin Architecture
  - Introduction to Mono

Goals

- Xamarin Development
  - Xamarin Cross platform solutions
  - Xamarin Family
  - Xamarin Development Approaches
  - Xamarin Advantages
  - Xamarin Disadvantages

Goals

- Setup the Development Environment on Windows
  - Setting up Xamarin on Windows
  - Configuring Visual Studio and SDK for Android and Windows UWP
  - Configuring Visual Studio Emulator for Android and UWP
  - Verify your Xamarin environment on Windows

Goals

- Setup the Development Environment on Mac
  - Setting up Xamarin on Mac
  - Configuring XCode, Visual Studio and Android SDK
  - Verify your Xamarin environment

Goals

- Xamarin Project Types
  - Xamarin Shared Projects
  - Xamarin Portable Class Libraries

Goals

- Xamarin Forms
  - Introduction to Xamarin Forms
  - Xamarin Forms Architecture
  - Xamarin Forms UI
  - Xamarin Forms UI Rendering Process
  - Xamarin vs. Xamarin Forms
  - PCL or Shared Project
  - Advantages of PCL over Shared Project

Goals

- Xamarin Forms Fundamentals
  - Pages
  - Views
  - Layouts
  - Cells
- Xamarin Forms Views/Controls and Views Alignment
  - Xamarin Forms Views/Controls
  - Comparing Xamarin Forms Controls with Native
  - Views Alignment

Goals

- Xamarin Forms App Life Cycle
  - Xamarin Forms Application Methods
  - Comparing Android and iOS and Windows App Life Cycle

- Layouts
  - Xamarin Forms Layouts
  - Layout Options
  - Stack Layout
  - Grid Layout
  - Relative Layout
  - Relative Layout - Constraint Expression Properties

Applications

The Appification of Modern Business

CUSTOMER APPS

Customer Service
Product Info
My Account
Purchases
Loyalty
Orders
...

SUPPLIER APPS

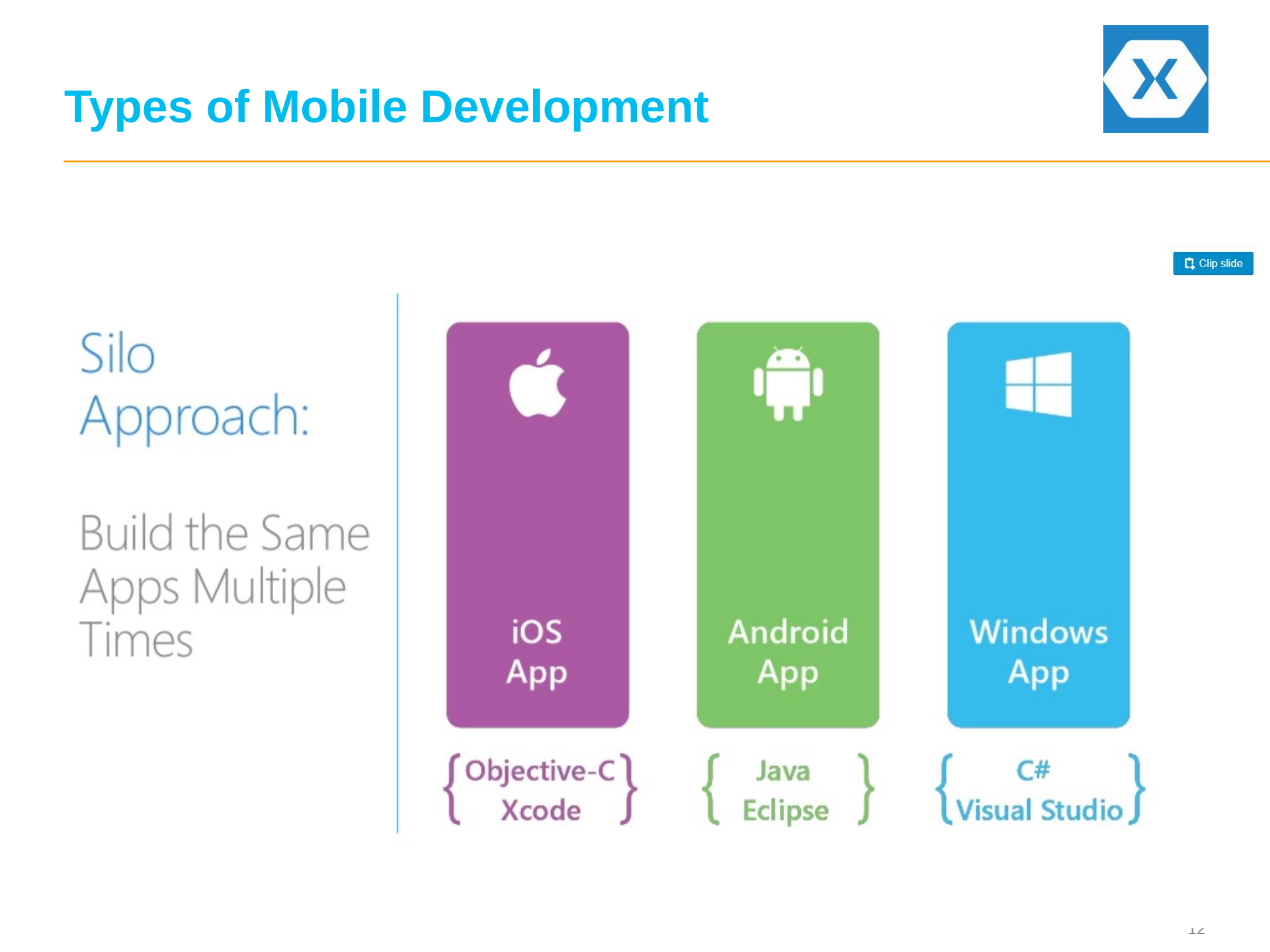
Order Management
Procurement
Dashboards
Inventory
Resellers
ERP
...

EMPLOYEE APPS

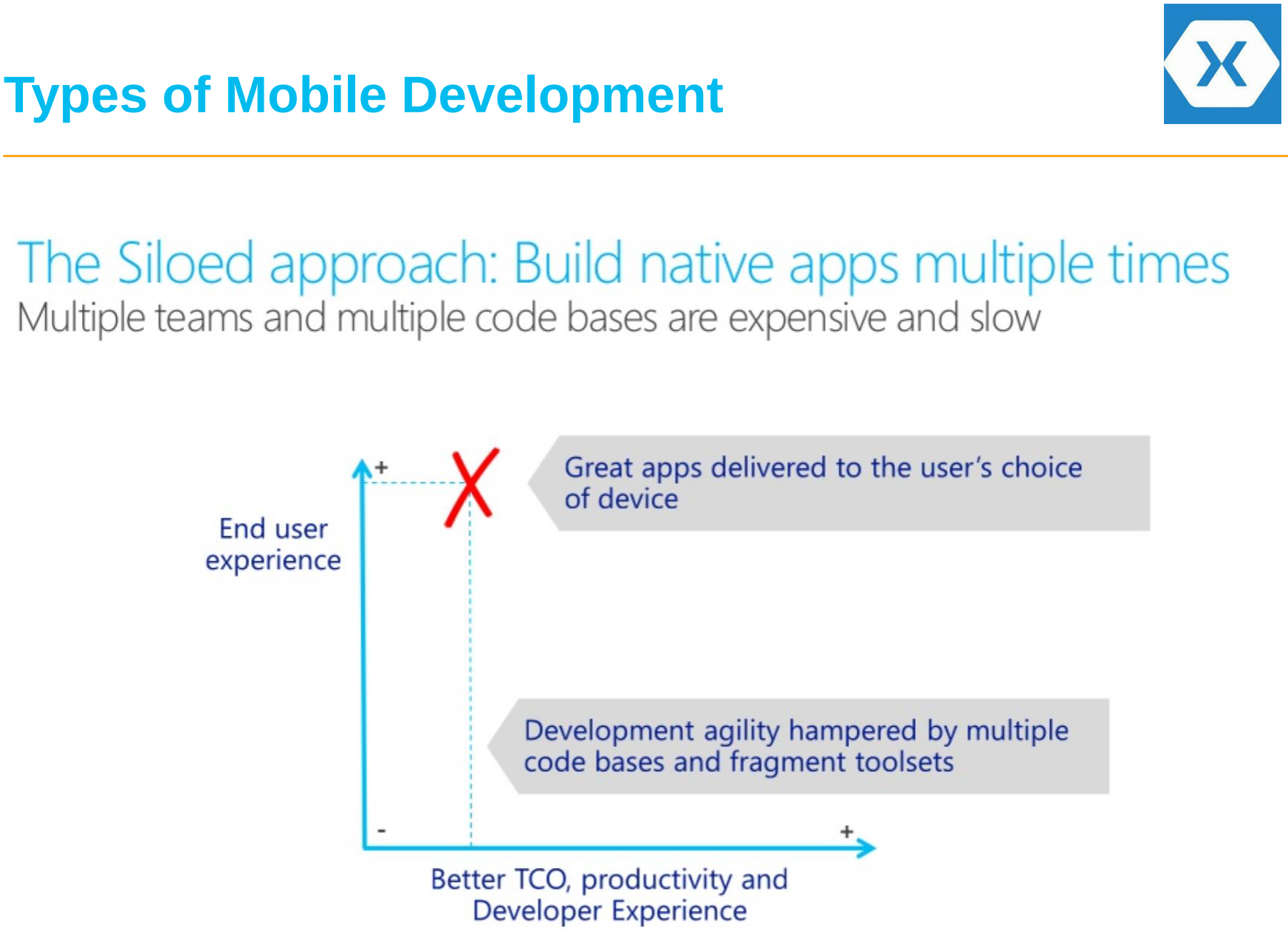
Health Services
Onboarding
Recruiting
Benefits
Payroll
Travel
...

Types of Mobile Development

Types of Mobile Development



Types of Mobile Development

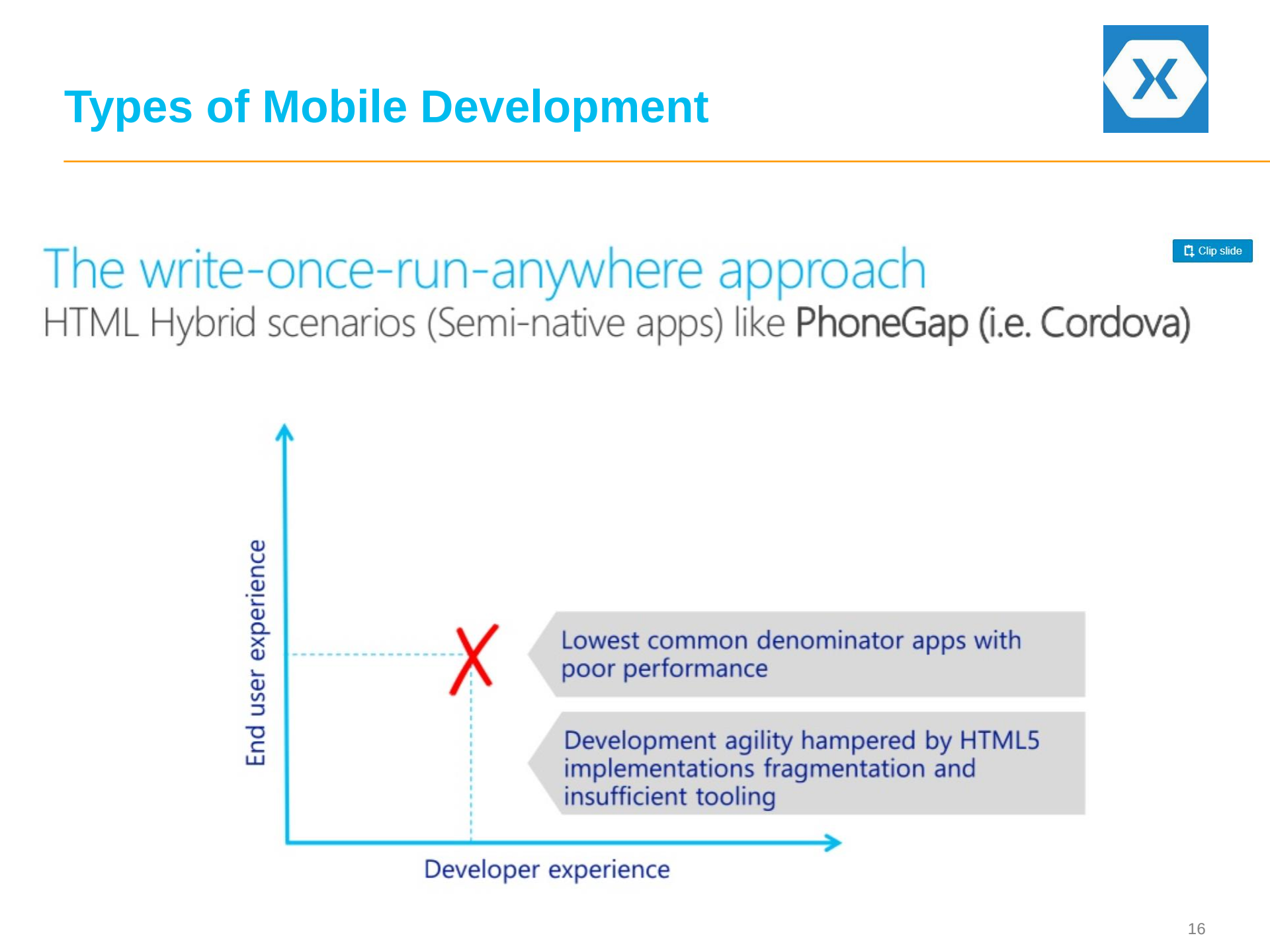


Types of Mobile Development

- Silo Approach Pros and Cons
- Pros
  - Great Application development can be built/delivered to the user's platform
- Cons
  - Multiple Teams
  - Multiple Code Base
  - Very Expensive

Types of Mobile Development

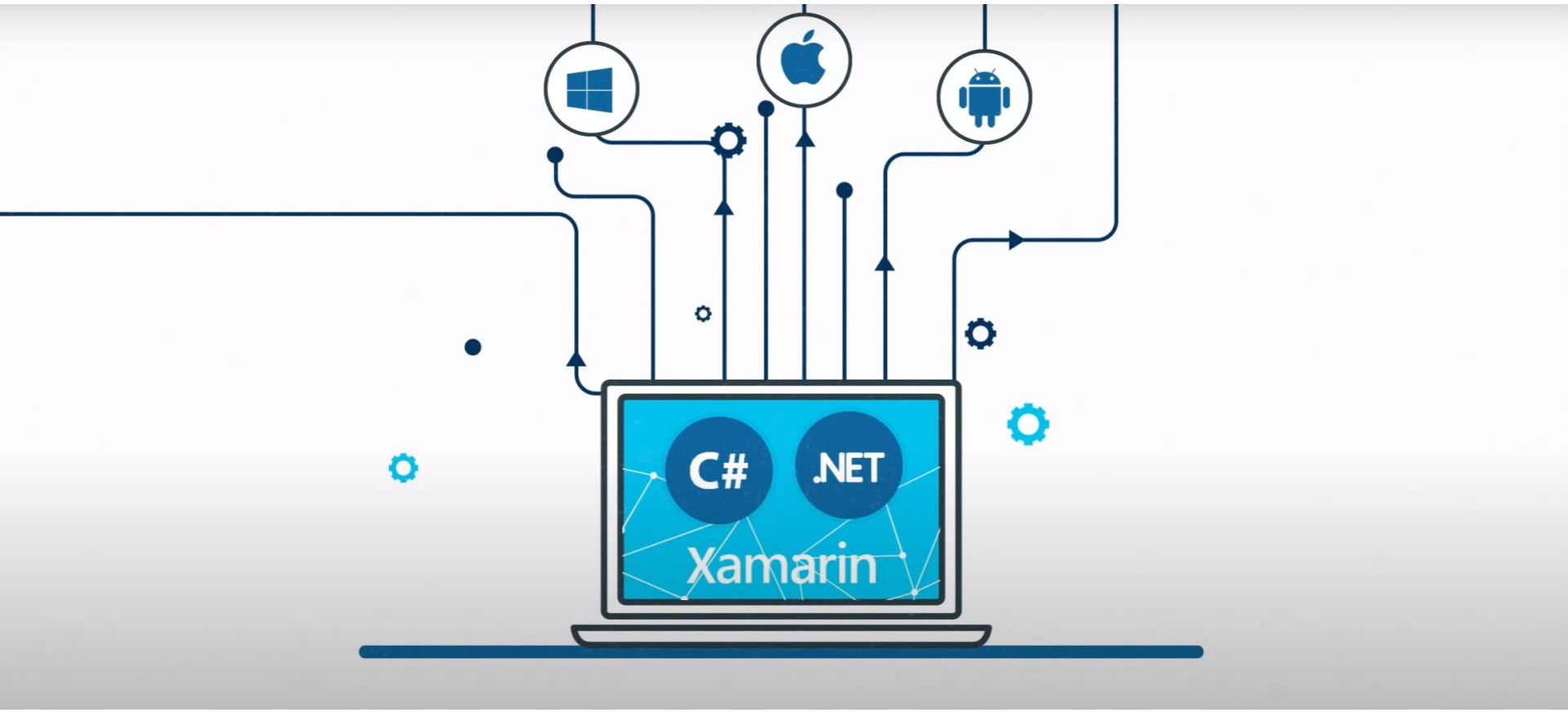
Types of Mobile Development



Types of Mobile Development

- Black Box Approach Pros and Cons
- Pros
  - Easy Development Process
  - Learning curve will be easy
- Cons
  - Performance will be very low
  - Not suitable for enterprise applications

Xamarin Approach



Types of Mobile Development

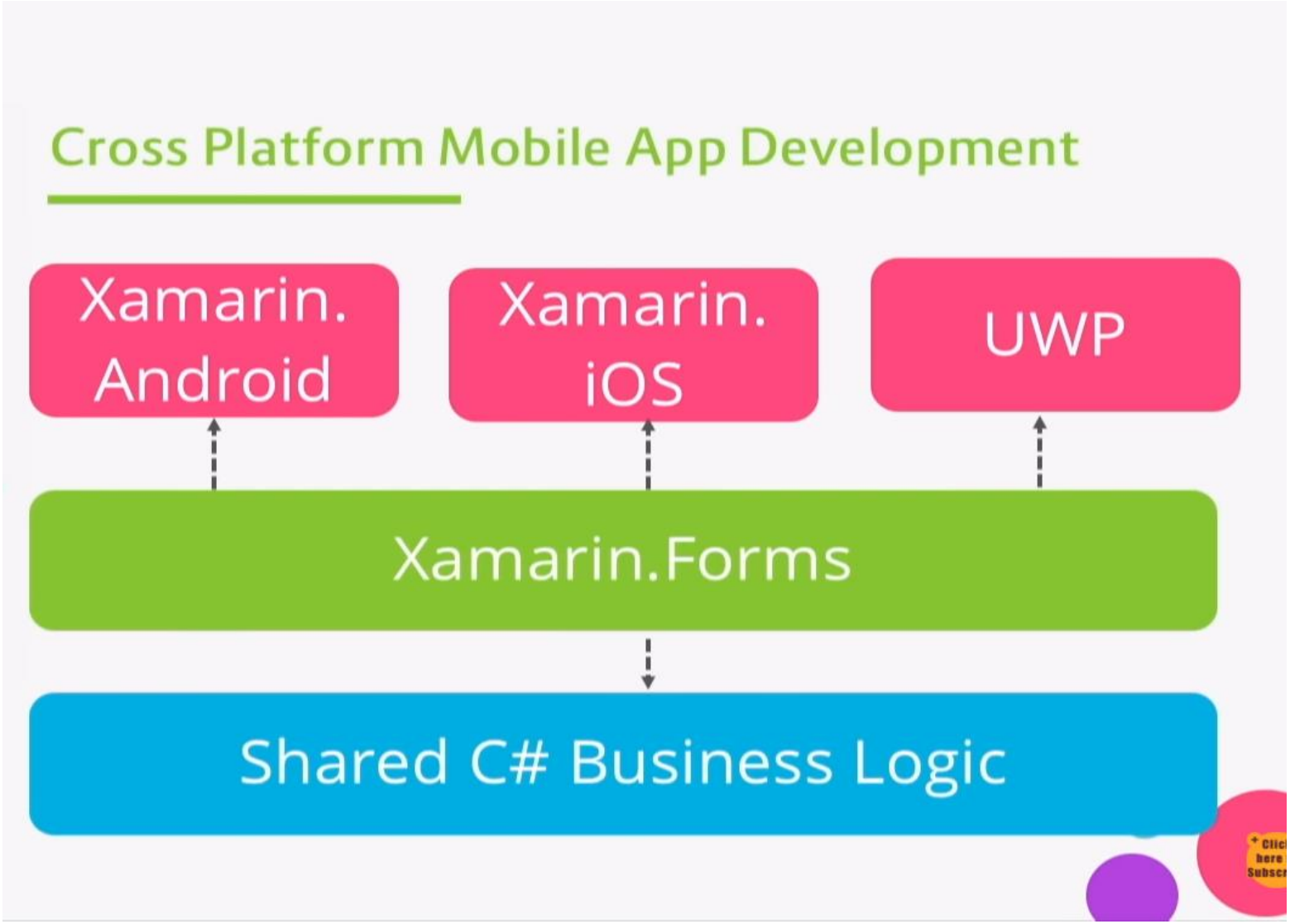
Xamarin Approach

Types of Mobile Development

Types of Mobile Development

- Xamarin Approach Pros and Cons
- Pros
  - Great Application can be built
  - Learning curve will be reduced
- Cons
  - Need to have some basic on native platform
  - UI has to be created separately for every platform

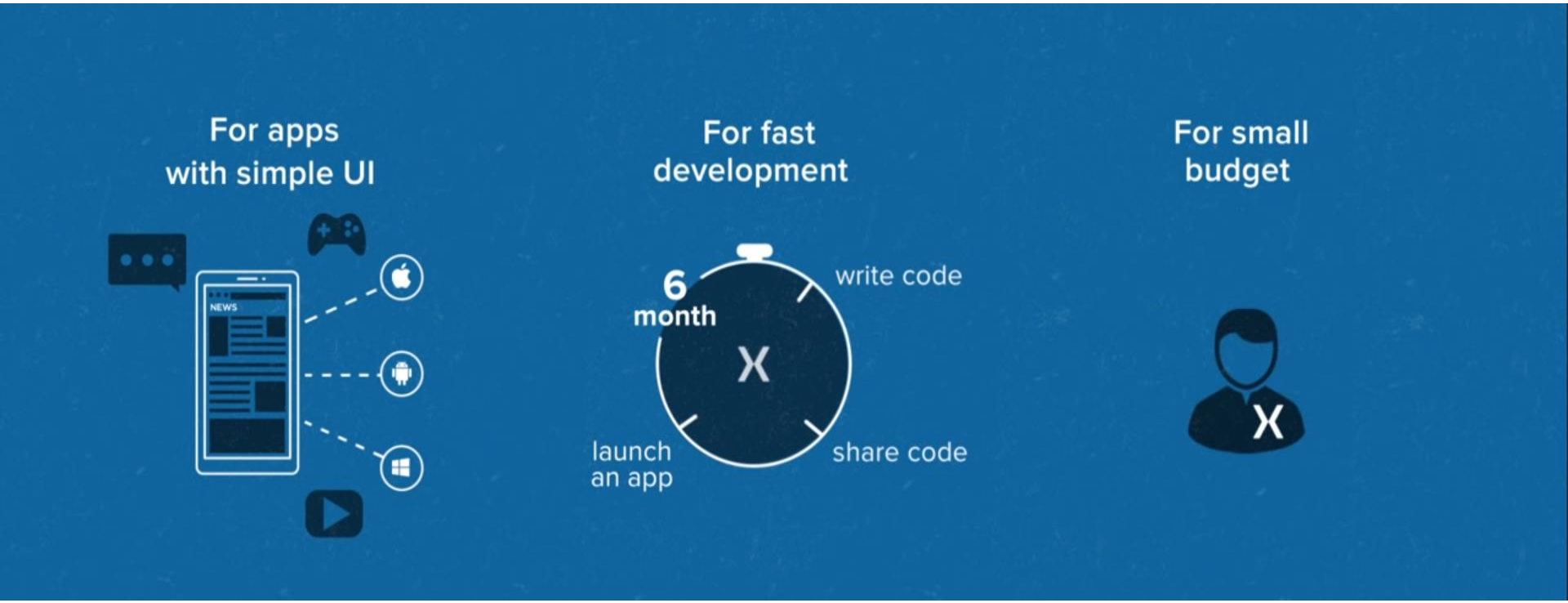
Types of Mobile Development



Types of Mobile Development

- Xamarin Approach Pros and Cons
- Pros
  - Business Logic and UI will be shared
  - Single codebase
  - Single Team
  - Development time will be faster
  - Native Apps will be developed
- Cons
  - Limited in capabilities
  - Performance will be slightly slow

When you should consider Xamarin



Introduction to Xamarin

- Xamarin is a software company which is originated in 2011.
- Xamarin was acquired by Microsoft in 2016.
- Xamarin provides the developers with the whole range of tools which can be used for the development of the cross-platform mobile application.
- It is a framework to develop a cross platform mobile application using C#.
- There are various frameworks which offer the cross-platform app development.
- It uses HTML and JavaScript. Using these frameworks, we can develop apps like a website for mobile app using JS libraries, the website is packed in a container which gives the feel of a native app.

Introduction to Xamarin

- Xamarin is different because it offers a single language C# and runtime, which works on three mobile platforms (Android, iOS, and Windows).
- Through Xamarin, we develop a mobile app whose look and feel is completely native.
- In Xamarin, we write one C# codebase which has access to all the features which is available for native SDK.

Xamarin Development IDE – Installation Of Xamarin

- Install Visual Studio
- Step 1: Make sure our computer is ready for Visual Studio
- To download the Visual Studio, we should focus on the system requirement to install the Visual Studio in our system.
- Here is the product support minimum system requirement is:
  - Visual Studio Enterprise 2019
  - Visual Studio Team Foundation Server Office Integration 2019
  - Visual Studio Professional 2019
  - Visual Studio Community 2019

Xamarin Development IDE – Installation Of Xamarin

Supported Operating System	Visual Studio 2019 install and run on the following operating system(64 bit recommended). ◦ Windows 10 Version 1703 or higher: Education, Home, Professional, and Enterprise Windows 8.1: Core, Professional, and Enterprise Windows Server 2012 R2: Essential, Standard and DataCenter Windows 7 SP1 (With Latest Windows Update): Home Premium, Professional, Enterprise, Ultimate.
Hardware	◦ 1.8 GHz or Faster Processor. Quad-Core or better recommended ◦ 2GB of RAM; 8GB of RAM recommended (2.5 GB minimum if running on a virtual machine). ◦ Hard Disk Space: Minimum of 800 MB up to 210 GB of available space, depending on features installed; typical installation required ◦ Hard Disk Speed: to improve performance, install Visual Studio and Windows on a solid-state drive (SSD). ◦ Video Card supports a minimum display resolution of 720p. Visual Studio will work best at a resolution of WXGA (1366 BY 768) or higher.
Supported Language	Visual Studio is available in English, Chinese (Simplified), Chinese (Traditional), Czech, French, German, Italian, Japanese, Korean, Polish, Portuguese (Brazil), Russian, Spanish, and Turkish. We can select the language of visual Studio during installation.

Xamarin Development IDE – Installation Of Xamarin

Additional Requirement	The administrator is required to install the Visual Studio. .Net Framework 4.5 is required to install Visual Studio. Visual Studio requires .NET Framework 4.7.2 which will installed during installation. .NET Core has specific Windows Prerequisite for windows 8.1. Windows 10 Enterprise LTSC edition, Windows 10 Edition is not supported for development. We may use Visual Studio 2017 to build apps that run on windows 10 LTSC and Windows 10S. Internet Explorer 11 or edge is required for internet-related scenarios. Some features might not work unless these or later version is installed. **For Emulator Support**, Windows 8.1 Pro or Enterprise(x64) editions are required. A processor which supports Client Hyper-V and Second Level Address Translation (SLAT) is also required. Universal Windows app development, including, editing, designing, and debugging, is required. **Xamarin.Android requires the 64-bit edition of Windows and the 64-bit java development kit (JDK).** Power shell 3.0 or higher is required on Windows 7 to install the mobile development with C++, JavaScript or .Net workloads.

Xamarin Environments

Clip slide

Xamarin for Visual Studio

Xamarin.iOS

Xamarin Studio

Xamarin.Android

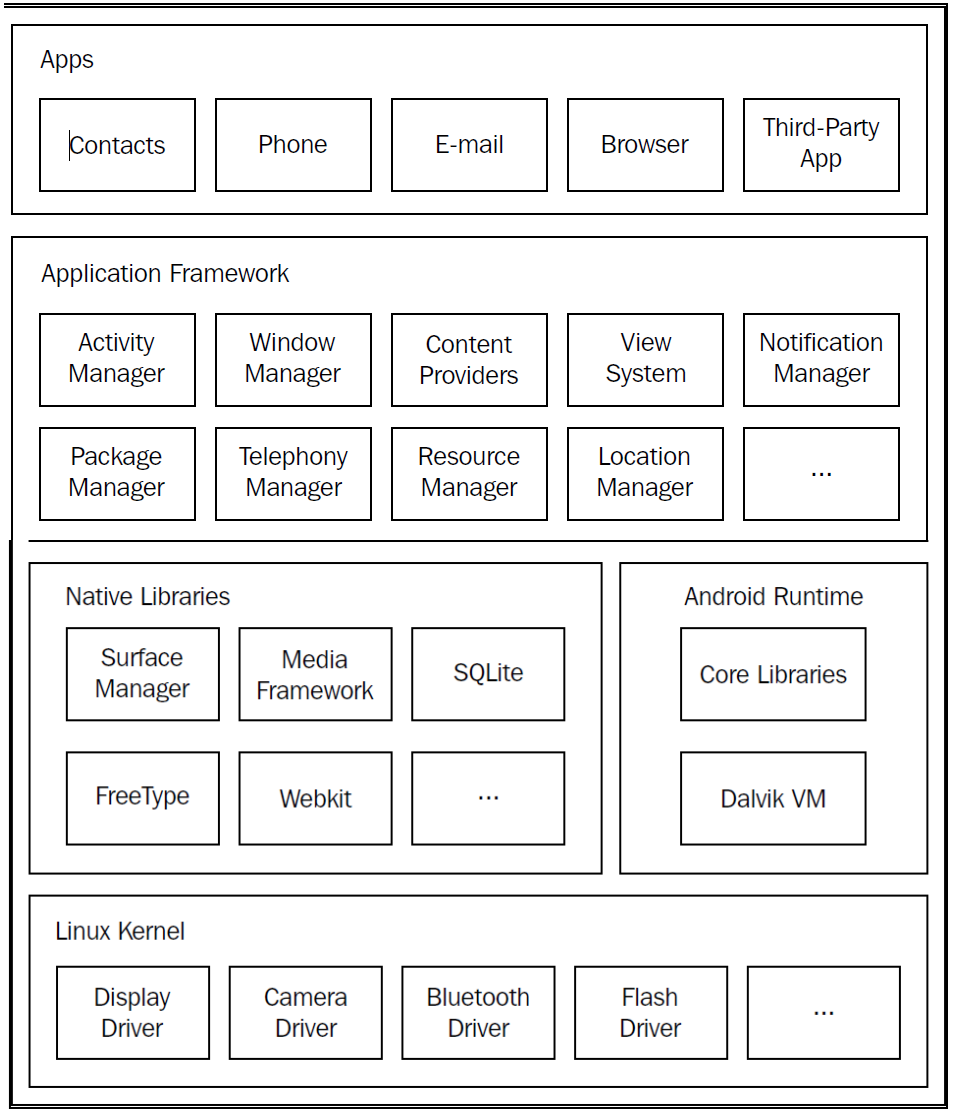
Xamarin Test Cloud

Xamarin.Mac

Component Store

.NET Mobility Scanner

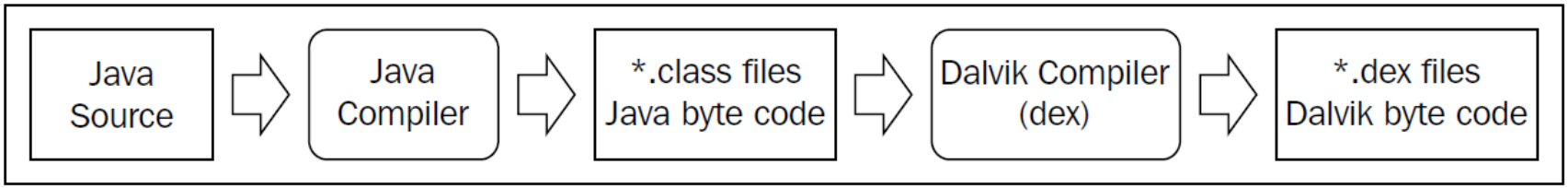
Anatomy of Android



The Android runtime

- Android apps run within the Dalvik Virtual Machine (Dalvik VM), which is similar to a Java VM but has been optimized for devices with limited memory and processing capacity.
- Android apps are initially compiled to the Java byte code using the Java compiler, but they have an additional compilation step that transforms the Java byte code to the Dalvik byte code, suitable to run within the Dalvik VM.

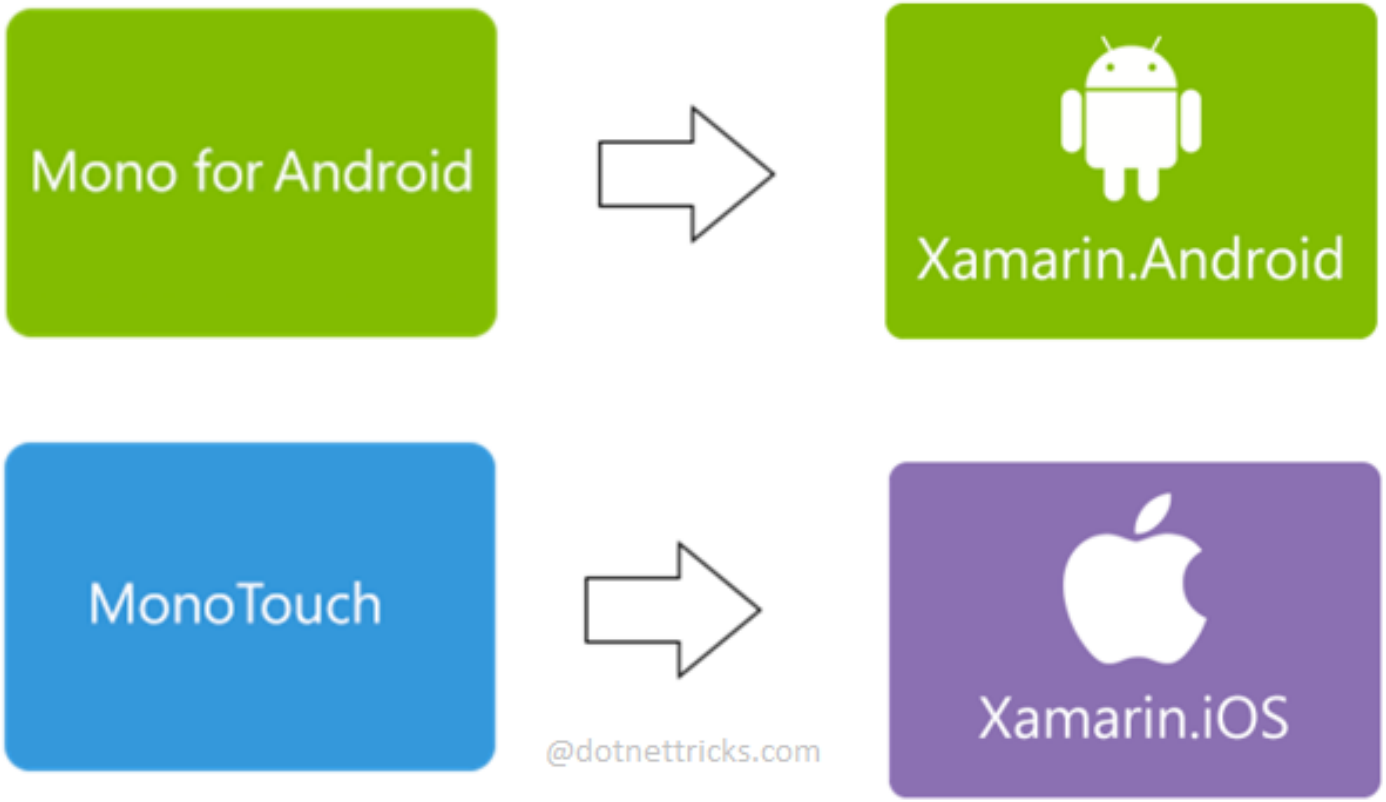
The Android runtime



What is Mono?

- Mono is a free open-source .NET framework developed by Novell and owned by Xamarin a company of Microsoft.
- First time it was introduced in June 30, 2004. The purpose of Mono project is to run .NET based applications on cross-platform including Linux, Android, mac etc.

What is Mono?


Mono for Android
Xamarin.Android
MonoTouch
Xamarin.iOS
@dotnettricks.com

What is Mono?

- Mono is an open source, cross-platform implementation of a C# complier, and a Common Language Runtime (CLR) that is binary compatible with Microsoft .NET.
- The Mono CLR has been ported to many platforms, including Android, most Linux distributions, BSD, OS X, Windows, Solaris, and even some game consoles such as Wii and Xbox 360.
- In addition, Mono provides a static compiler that allows apps to be compiled for environments such as iOS and PS3.

Xamarin Architecture

Xamarin Family

Xamarin.Android

Native mobile apps for Android, Android Wear, and Android TV

Xamarin.iOS & Xamarin.Mac

Native mobile apps for iOS, watchOS, tvOS, and OS X

Xamarin Forms

Native UIs for iOS, Android & Windows from a single, shared codebase

Development SDKs

Xamarin Apps Development IDEs

Xamarin Studio

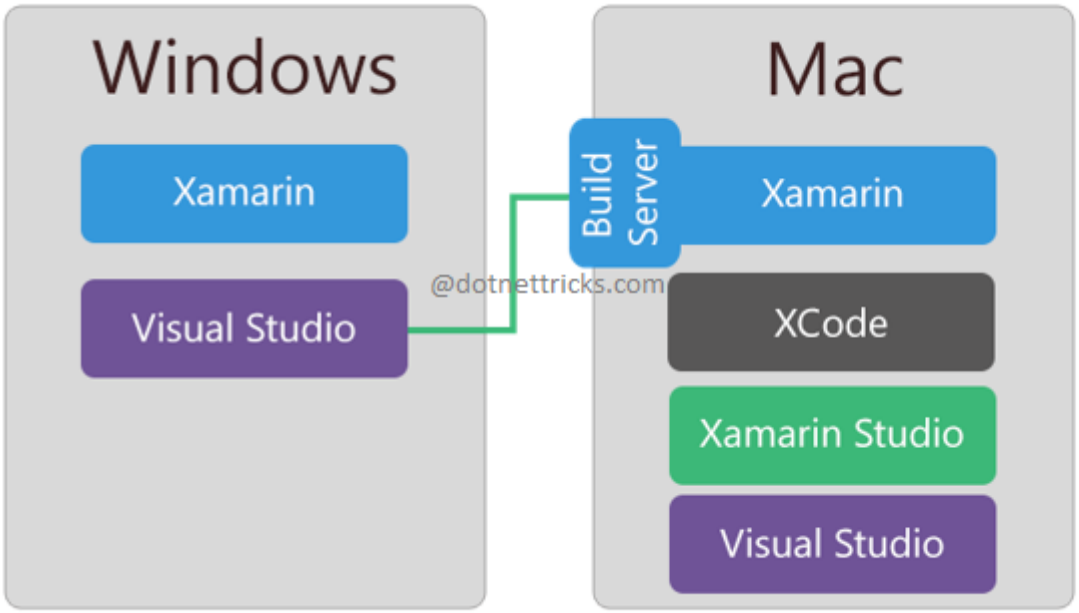
Visual Studio for Mac

Visual Studio for Windows

Development with Mac Machine

Development with Windows Machine

Development with Windows and Mac Machine

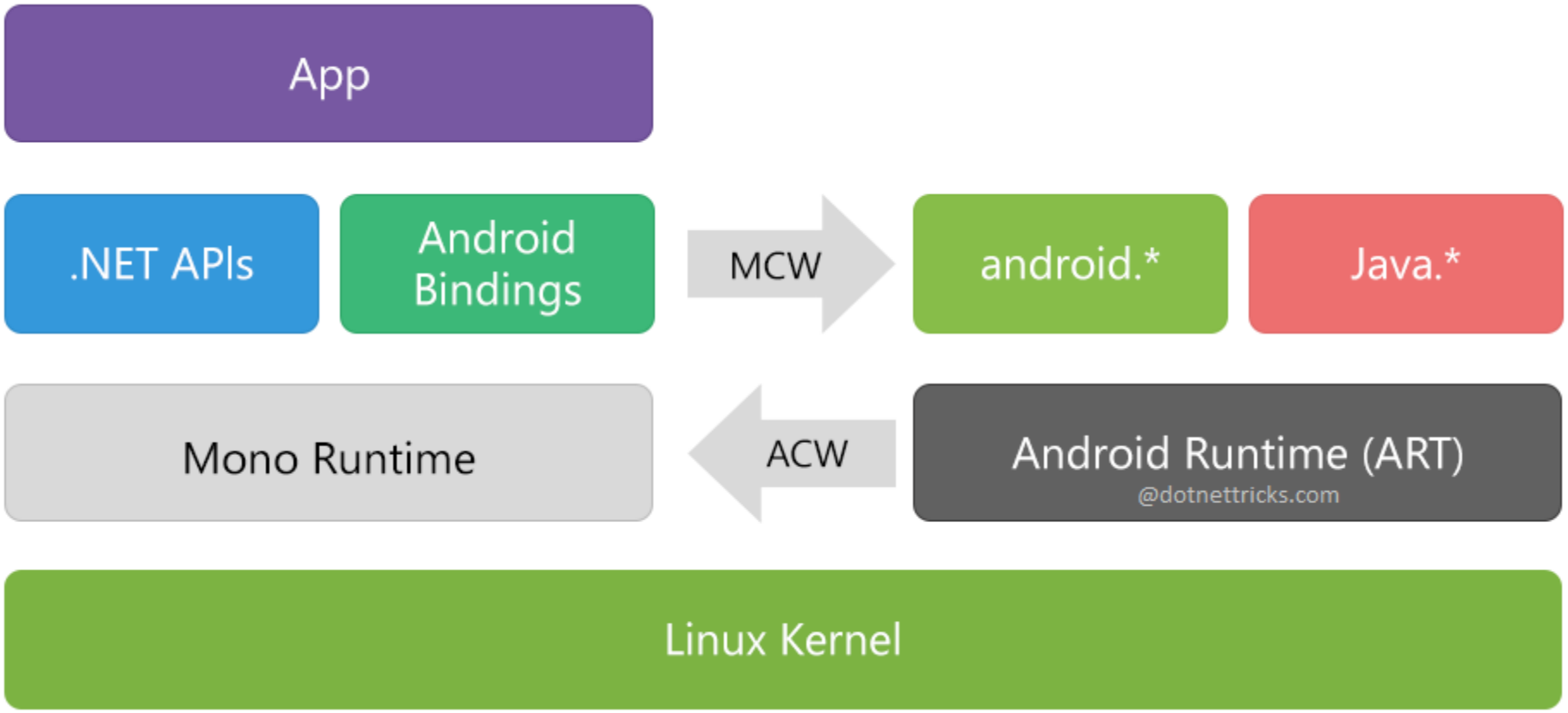


Android

iOS

UWP

Xamarin Architecture



C# packages

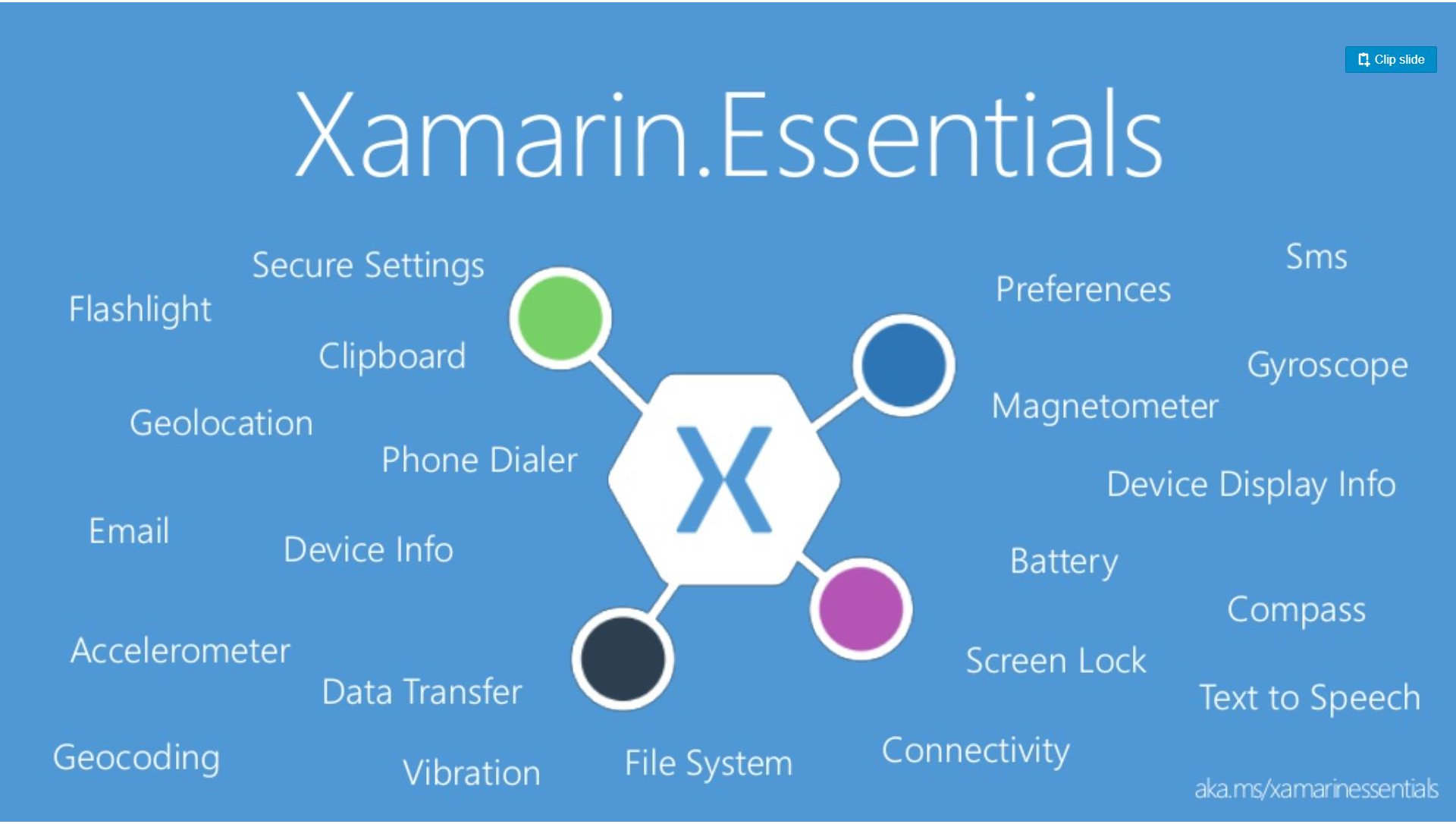
Xamarin.Android

Clip slide

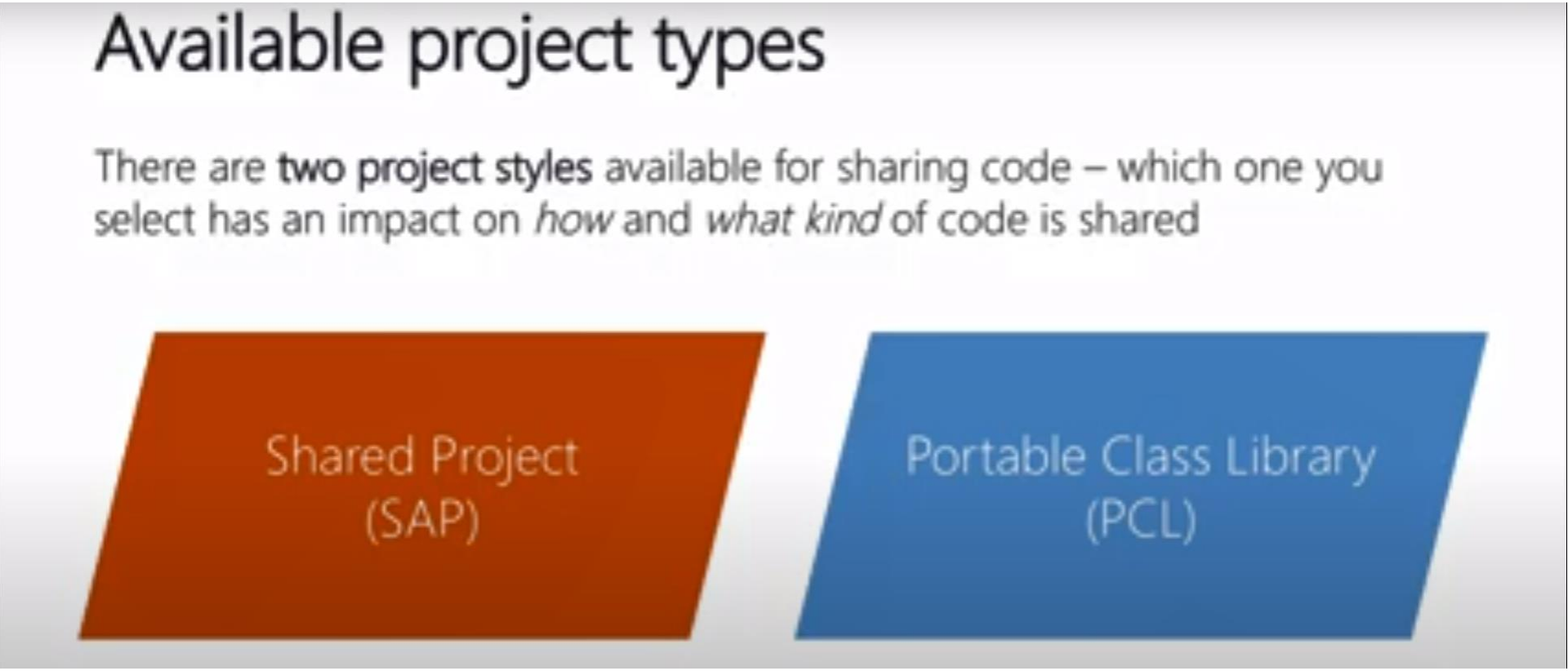
Fingerprint	Bluetooth	Picture-in-Picture	Geolocation	NFC
System.Net	System	System.IO	System.Linq	System.Xml
System.Data	System.Windows	System.Numerics	System.Core	System.ServiceModel

C#

Xamarin Essentials



Xamarin Project Types

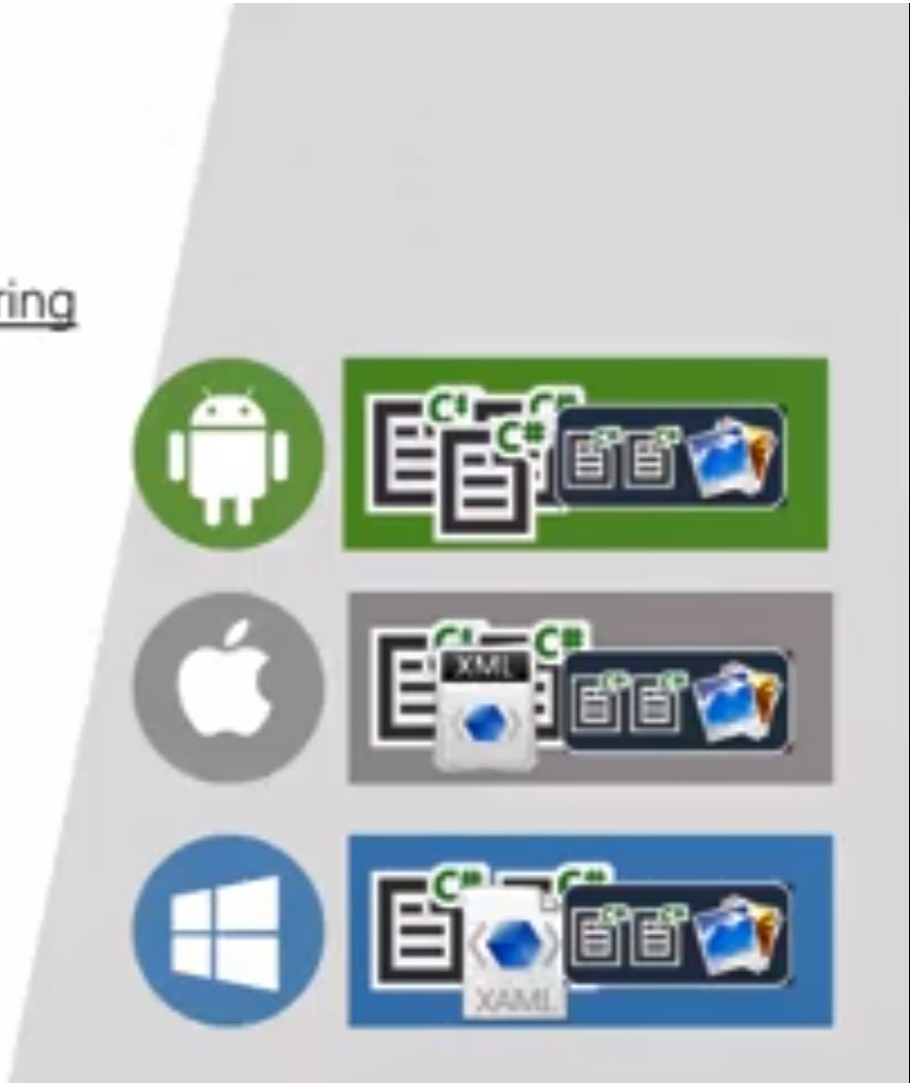


Xamarin Project Types

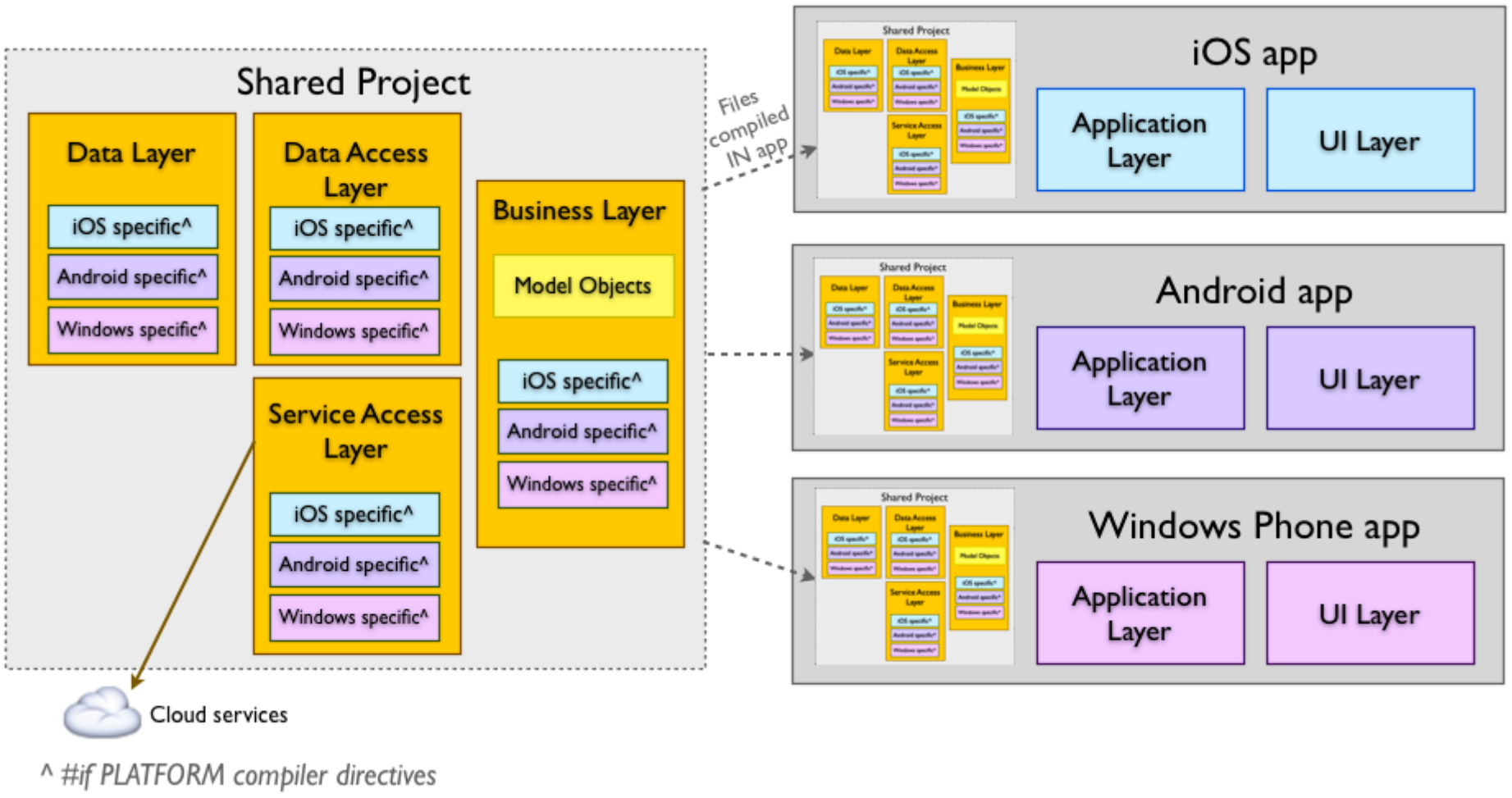
Shared Projects

Shared Projects enable project-level sharing of **source + assets**

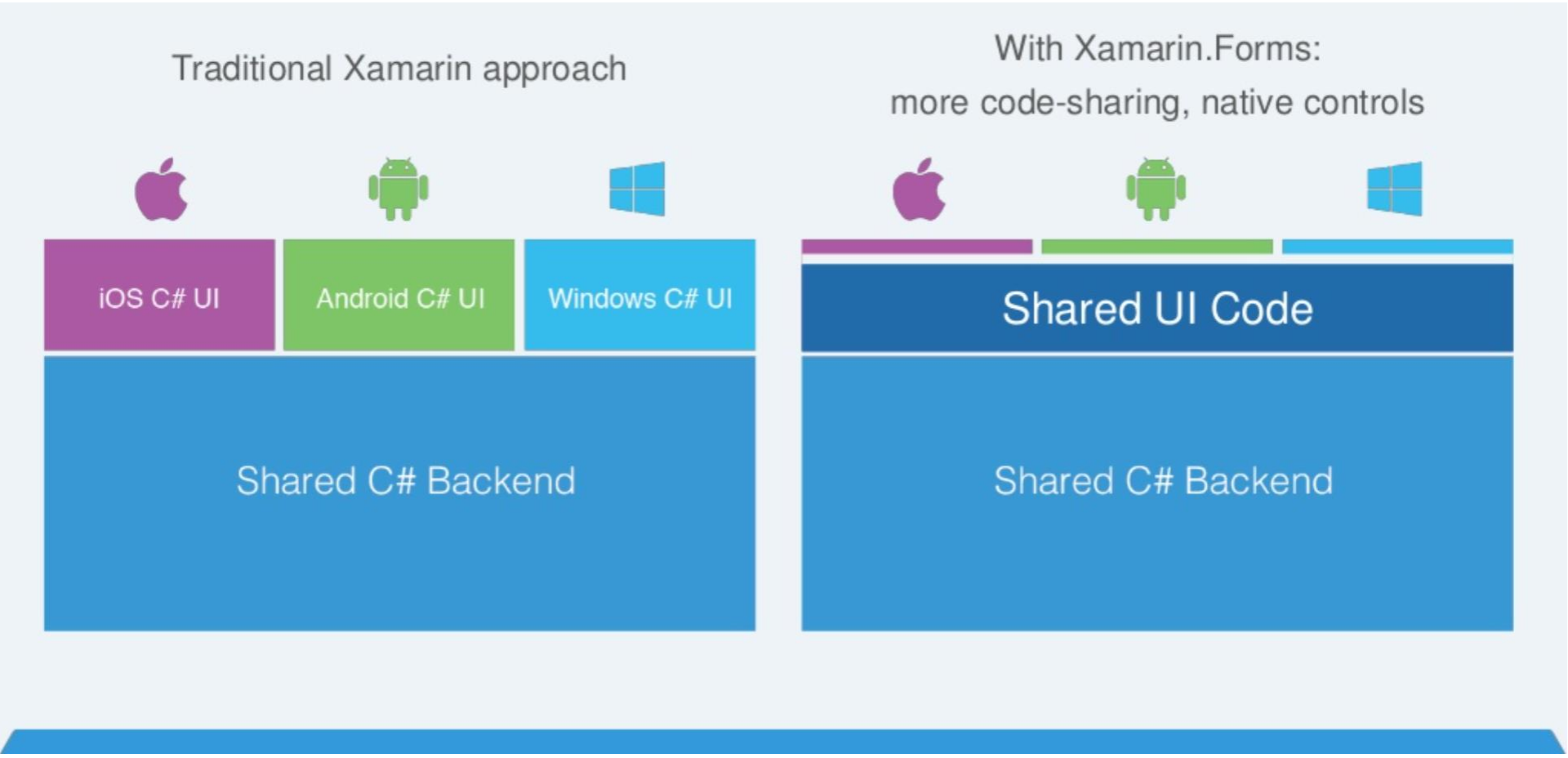
- ✔ single copy of source file
- ✔ compiled uniquely into project
- ✔ Normal refactoring + navigation works



Shared Projects code sharing



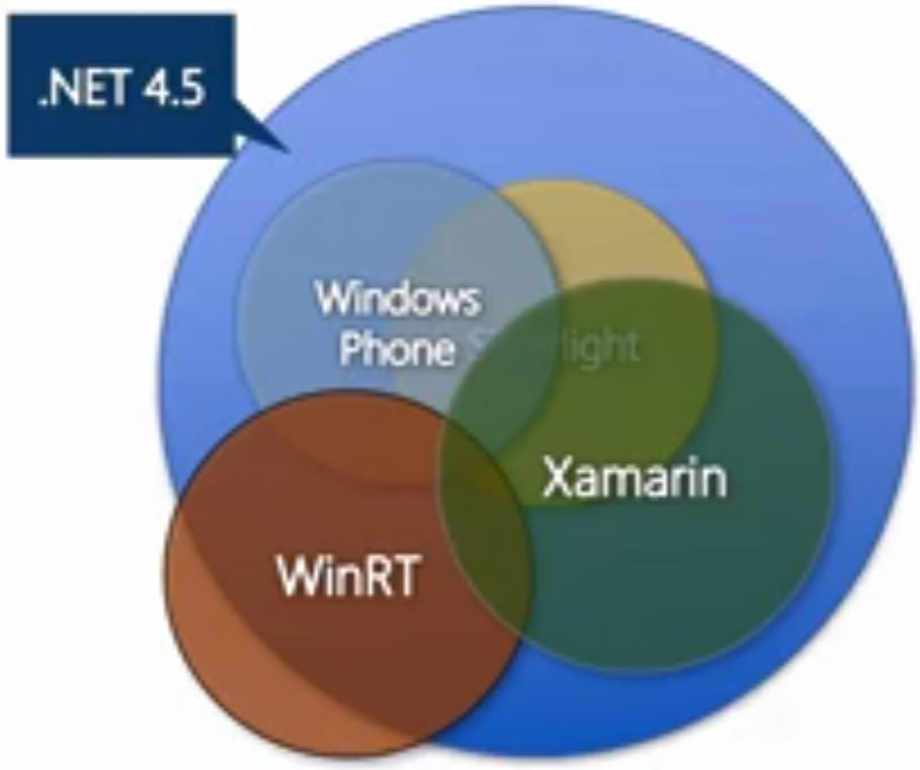
Shared Projects code sharing



Portable Class Libraries (PCL)

- PCL projects target specific profiles that support a known set of BCL classes/features.
- When you create an Application Project or a Library Project, the resulting DLL is restricted to working on the specific platform it is created for.
- This prevents you from writing an assembly for a Windows app, and then re-using it on Xamarin.iOS and Xamarin.Android

Portable Class Libraries (PCL)



Shared as a binary library (.dll) – client does not need source code

You select the platforms the library will be used on – this decides the **profile**

The available combinations are controlled by the profiles Microsoft has defined

The *more platforms* supported, the *less APIs* you will be able to use

Portable Class Libraries (PCL)

Complex requirements can be described by an abstraction that is defined in the PCL

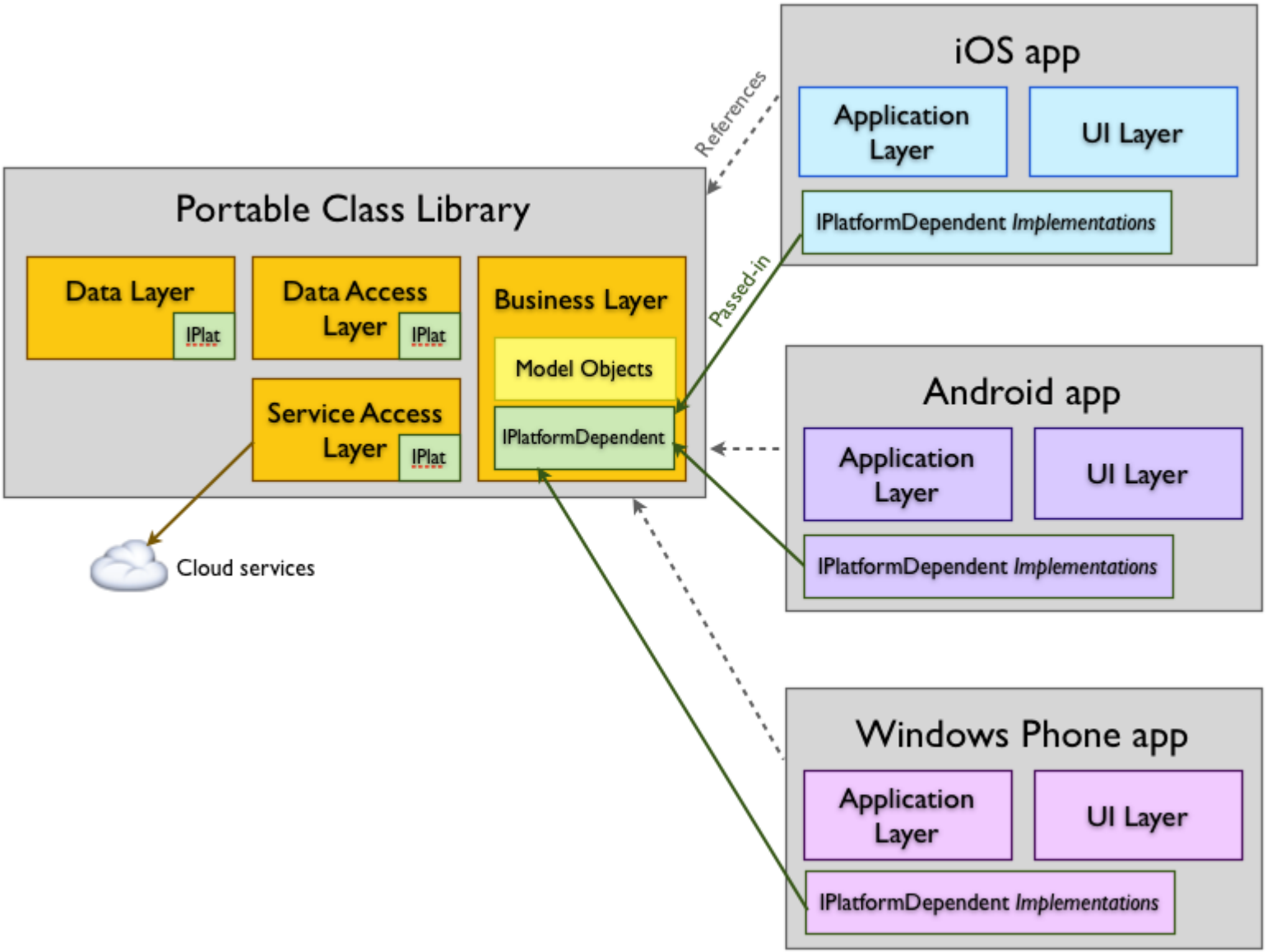
```
public interface IDialer
{
    bool MakeCall(string number);
}
```

PCL

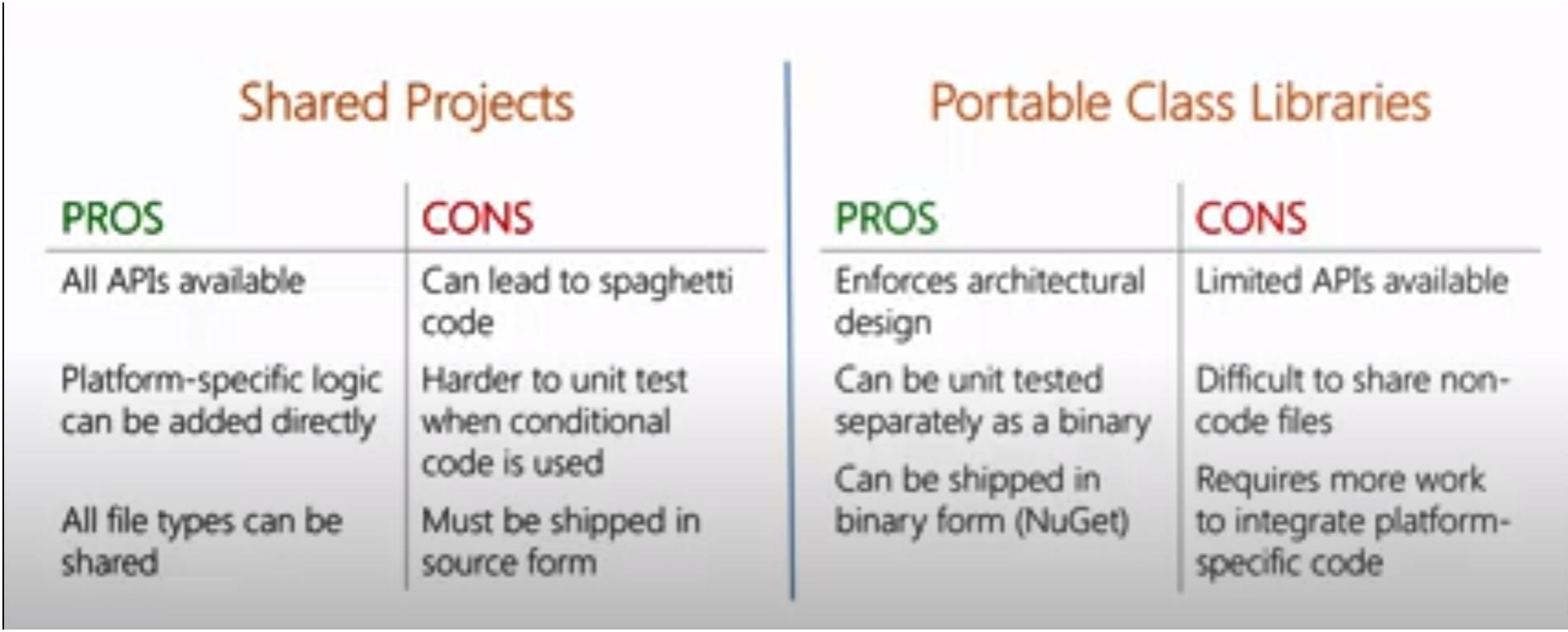
Shared code defines **IDialer** interface to **represent required functionality** – this is what the PCL uses to get to the API

Platform projects **implement the shared dialer interface** using the platform-specific APIs

Portable Class Libraries (PCL)



Portable Class Libraries (PCL)



Shared Projects

PROS	CONS
All APIs available	Can lead to spaghetti code
Platform-specific logic can be added directly	Harder to unit test when conditional code is used
All file types can be shared	Must be shipped in source form

Portable Class Libraries

PROS	CONS
Enforces architectural design	Limited APIs available
Can be unit tested separately as a binary	Difficult to share non-code files
Can be shipped in binary form (NuGet)	Requires more work to integrate platform-specific code

Xamarin Forms

- Xamarin.Forms is an open-source, cross-platform framework acquired by Microsoft for building Android, iOS, and windows app with .NET from a single shared codebase.
- We use Xamarin. Forms built-in Pages, layouts, and controls to build and design mobile apps from a single API that is highly extensible.
- Subclass any control to customize the behavior or to define our controls, layouts, pages, and cells to make our apps pixel perfect.
- One of the greatest benefits of Xamarin.Forms is that it gives us the ability to develop native mobile apps for several platforms simultaneously

Xamarin.forms

Xamarin.Forms is a cross-platform UI framework to create mobile apps for:

- Android 4.0+
- iOS 6.1+
- Windows Phone 8.x (SL)
- Windows Phone 8.1 (RT)
- Windows 10 (UWP)

Xamarin.forms

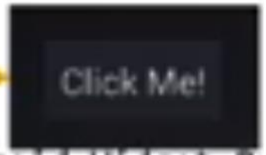
Xamarin.Forms renders native controls

Platform defines a *renderer* for each view that creates a native representation of the UI

UI uses a Xamarin.Forms Button

```
Button button = new Button {
    Text = "Click Me!"
};
```

Platform Renderer takes view and turns it into platform-specific control

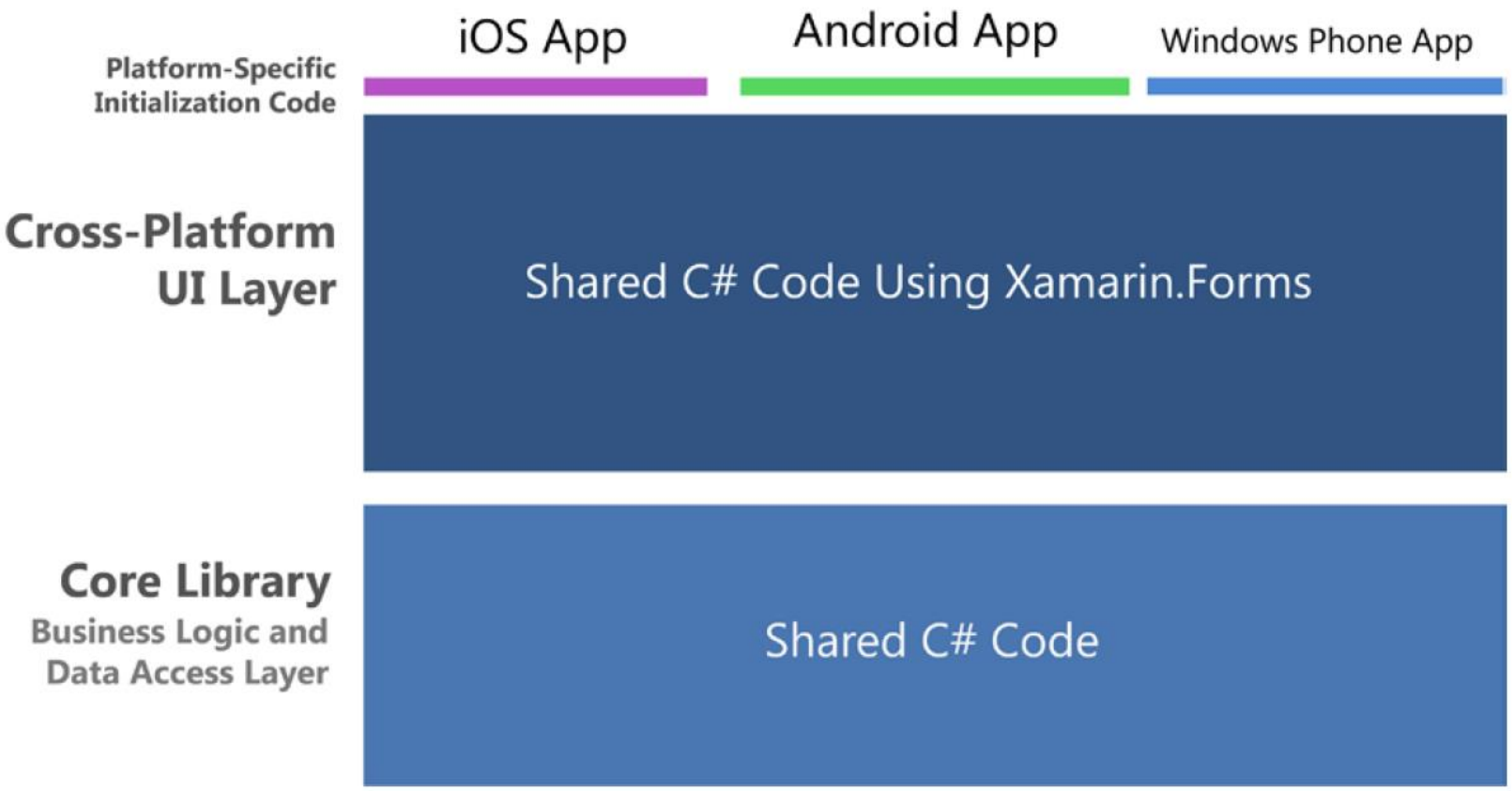


Android.Widget.Button

UIButton

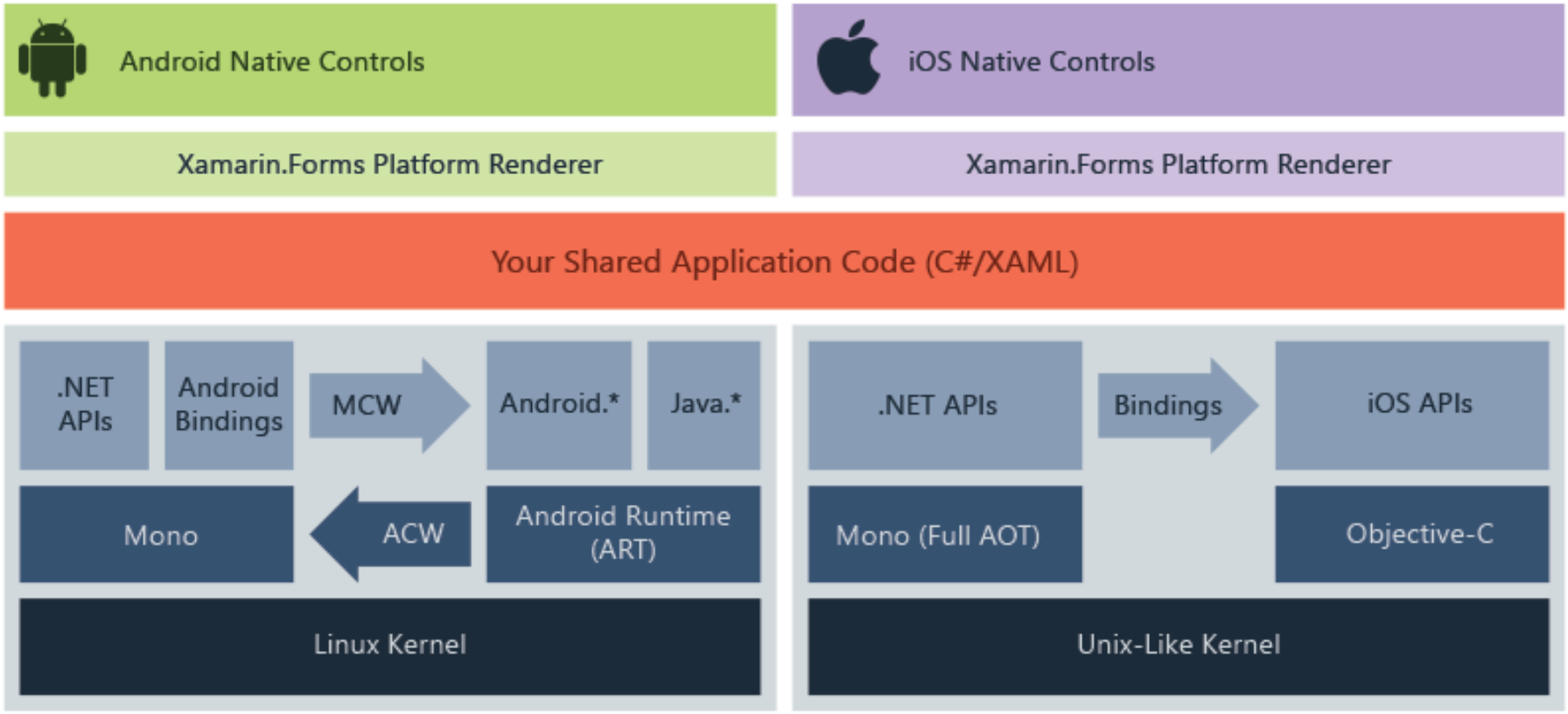
System.Windows.Button

Xamarin Forms Architecture



Xamarin.Forms solution architecture: One app for three platforms

Xamarin Forms Architecture



Xamarin.Forms solution architecture: One app for three platforms

Choosing Xamarin.Forms or a Platform-Specific UI

- Use Xamarin.Forms for the following:
  - Learning Xamarin: If we're new to mobile development using C# then Xamarin. Forms is a great way to get started!
  - Cross-platform scaffolding: When building cross-platform apps, Xamarin.Forms is useful to build out the scaffolding of your app, as a rapid-application development toolset.
  - Basic Business Apps: Xamarin.Forms does these things well: basic data display, navigation, and data entry. This is a good fit for many business apps.
  - Basic design: Xamarin.Forms provides controls with baseline design features, facilitating basic visual formatting.

Choosing Xamarin.Forms or a Platform-Specific UI

- Use Xamarin.Forms for the following:
  - Simple cross-platform screens: Xamarin.Forms is great for creating fully functional basic screens. For more complex screens, leverage Xamarin.Forms custom renderers for platform-specific details.

Choosing Xamarin.Forms or a Platform-Specific UI

- Use a platform-specific UI (Xamarin.iOS or Xamarin.Android) for:
  - Complex screens: When an entire screen (or an entire app) requires a nuanced and complex design and UI approach, and Xamarin.Forms isn't quite up to the task,go with a platform-specific UI using Xamarin.Android and Xamarin.iOS.
  - Consumer Apps: Platform-specific UI has everything a developer needs to create a consumer app with complex visual design, nuanced gesture sensitivity, and high-end graphics and animation.

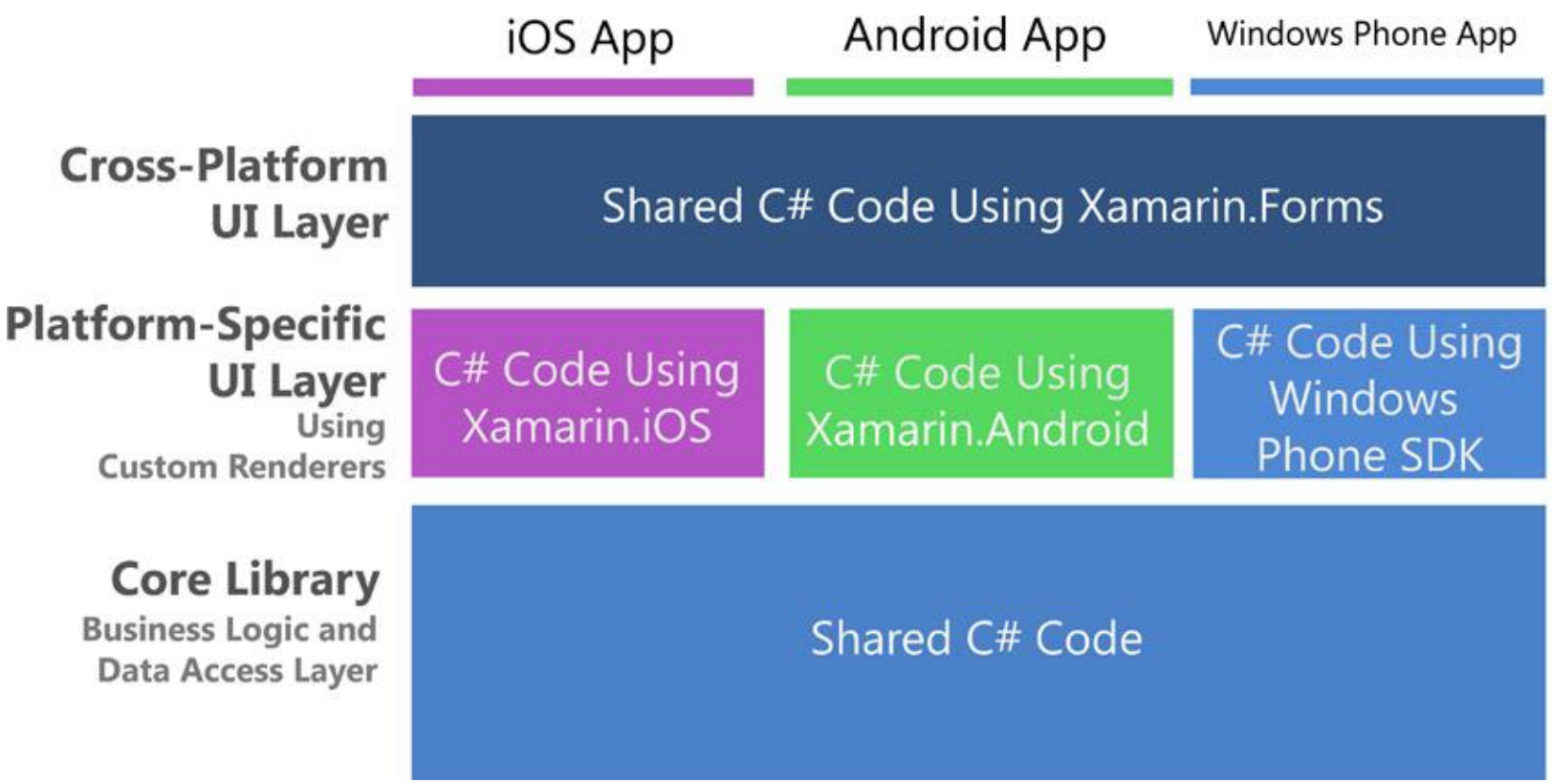
Choosing Xamarin.Forms or a Platform-Specific UI

- Use a platform-specific UI (Xamarin.iOS or Xamarin.Android) for:
  - High-design: This approach provides complete native UI APIs with low-level access to design properties on each control, allowing for a high visual standard of design. Native animation and graphics are also available with this approach.
  - Single-platform apps: If we're building for only one platform, and a cross-platform approach for your app is not important in the foreseeable future (a rare case even if you're starting with one platform), consider using a platform-specific UI.
  - Unsupported platforms: Mac OS X, Windows Store, and WinRT apps are not currently supported by Xamarin.Forms at this time.

Use Both Approaches with Custom Renderers

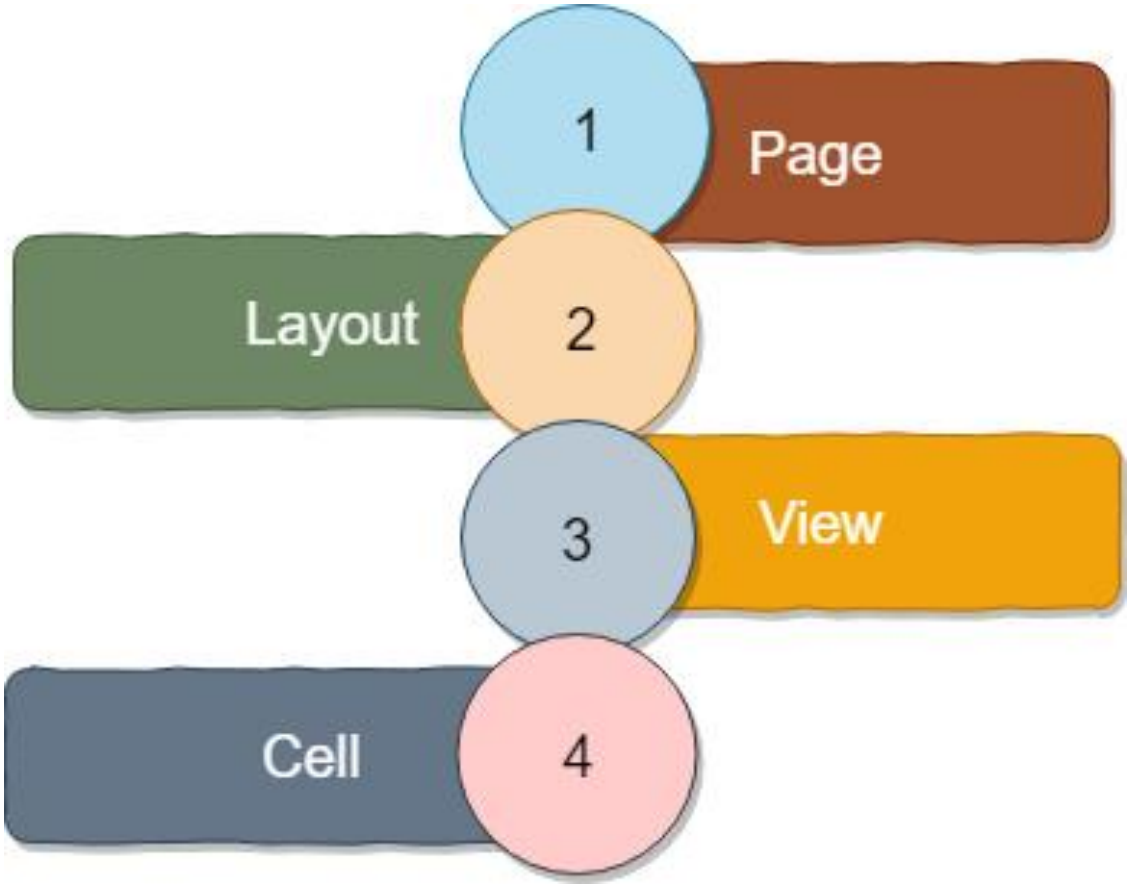
- Custom Renderers provide access to the lower-level, platform-specific, screen-rendering classes called Renderers, which use platform-specific controls to create all Xamarin.Forms screens.
- Any Xamarin. Forms screen can be broken into platform-specific screens and classes using this approach.
- This means we can write a Xamarin.Forms page or app, and customize it by platform whenever necessary.

Xamarin Forms Architecture

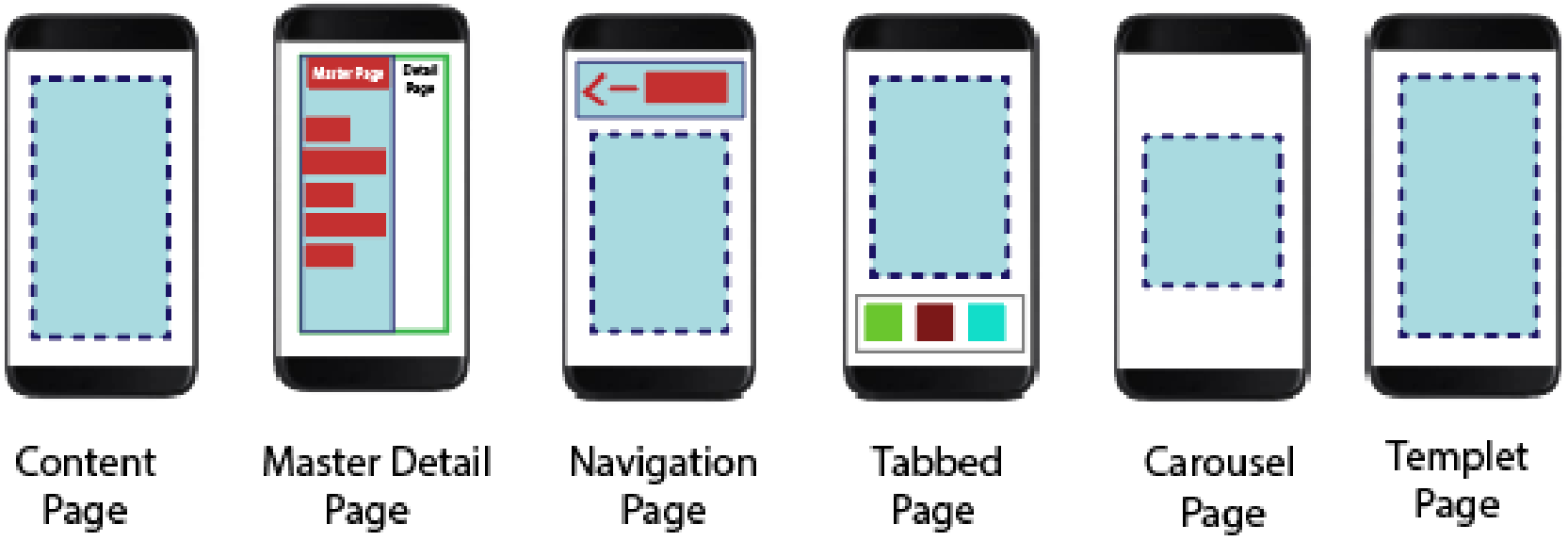


Xamarin.Forms architecture with custom renderers

Xamarin.forms UI



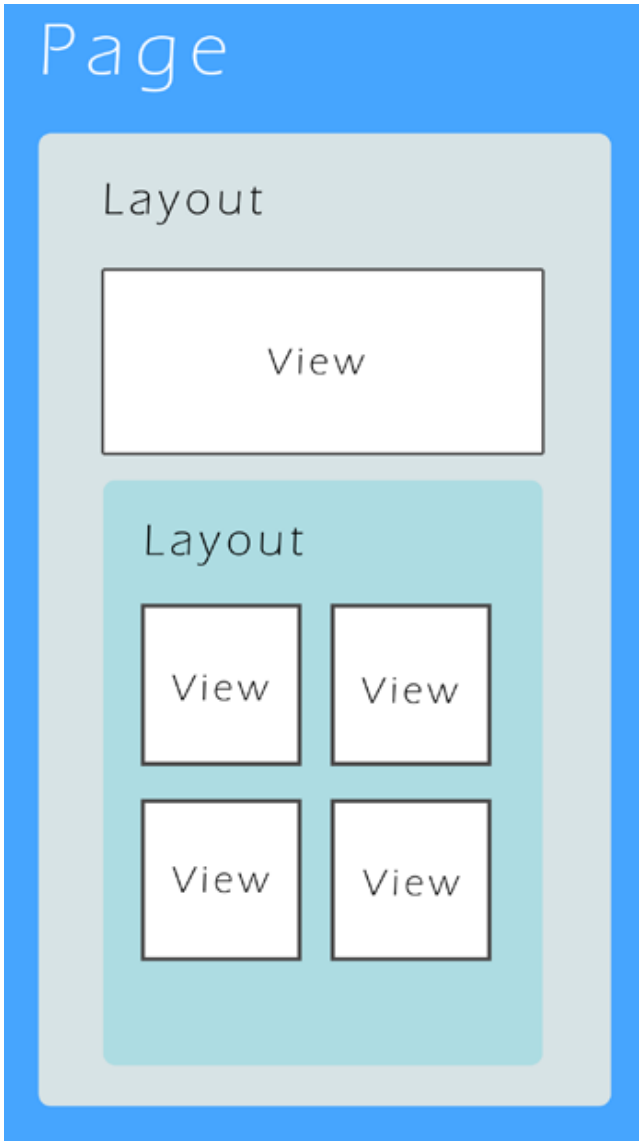
Xamarin.forms UI



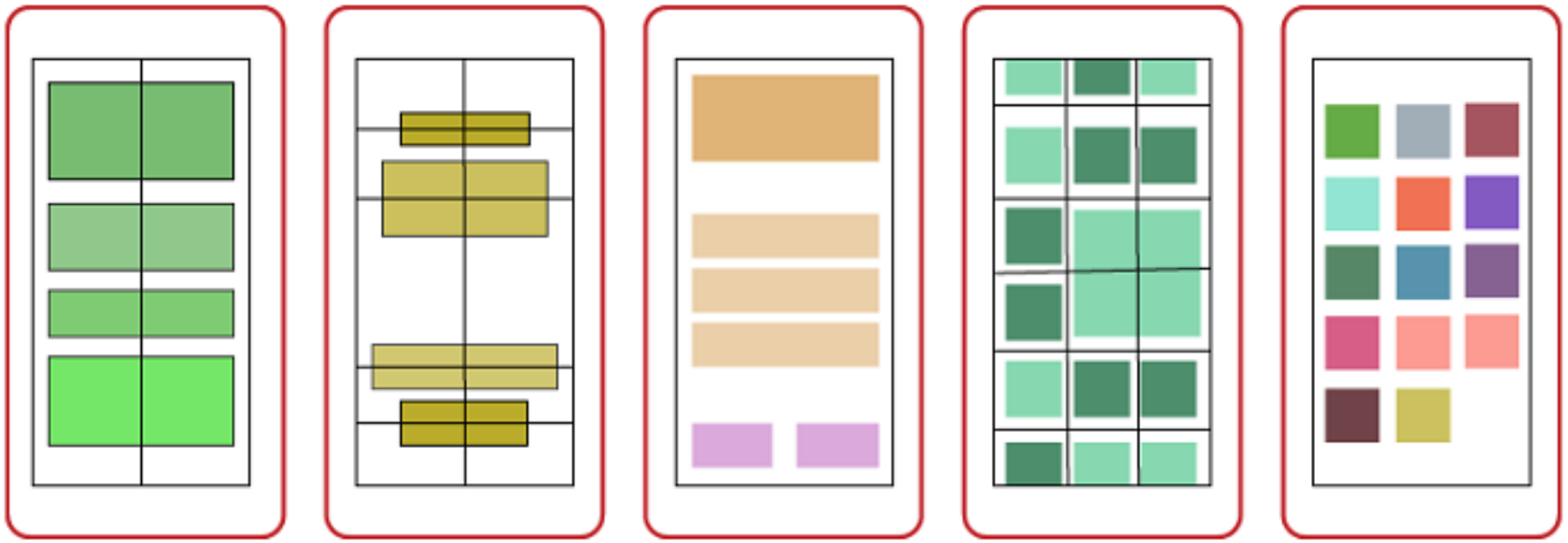
Xamarin.forms UI

Pages	Description
ContentPage	Content Page contains a single view.
MasterDetailPage	MasterDetailPage has two panes for the page. Master Page contains the Menu and detail page contains the content.
NavigationPage	NavigationPage contains the navigation bar. In NavigationPage, we kept the page on a stack and can jump from one page to another. The navigation bar can have navigation buttons as well as the title.
TabbedPage	TabbedPage is a container page. The tabbed page acts as a container which holds the content page associated with each tab.
CarousalPage	The page that allows the sweeping across to show other views.

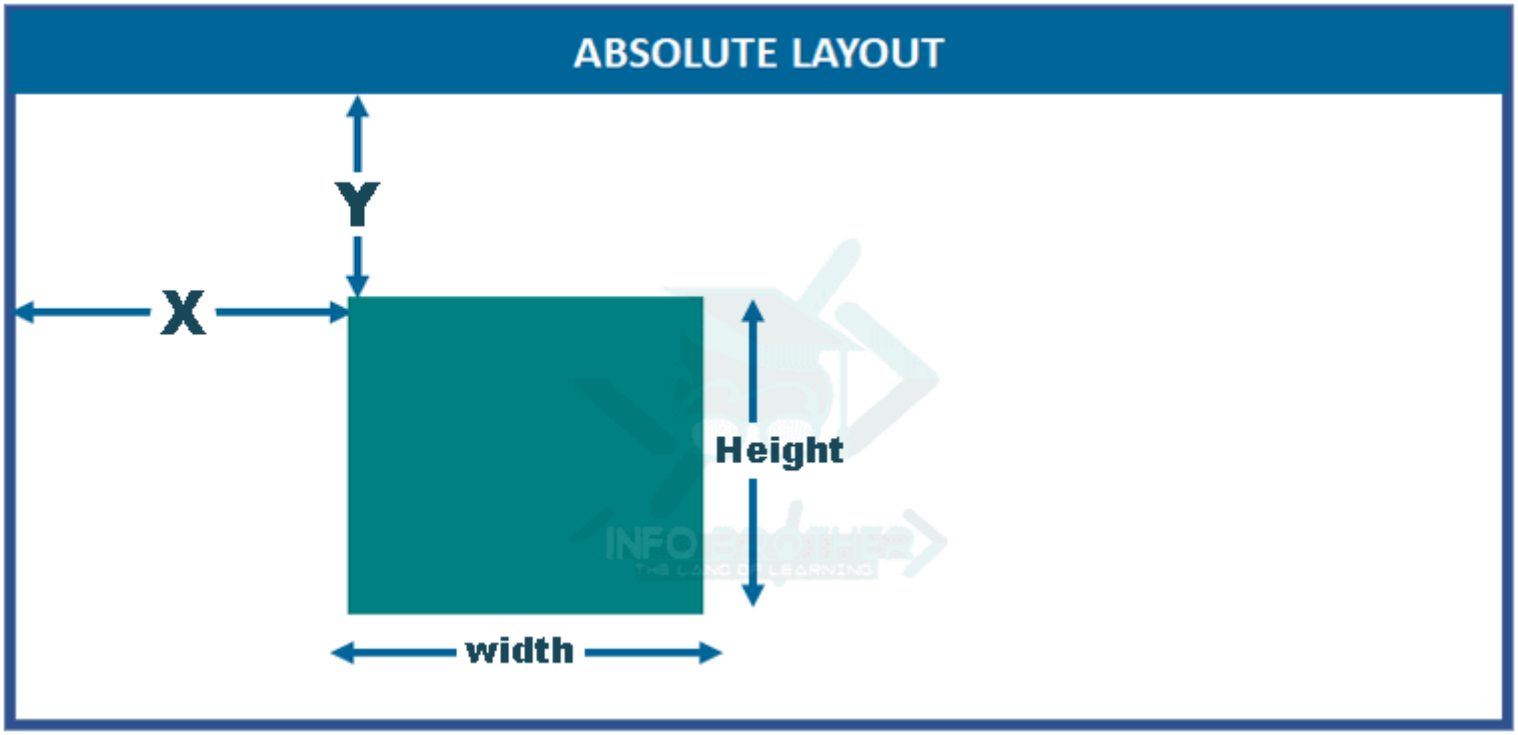
Xamarin.forms UI



Xamarin.forms UI



Absolute Layout



Absolute Layout

DECLARATION:

The Absolute layout could be used to positioned the elements anywhere in the layout by using the following syntax:

```
<AbsoluteLayout>
    <Children
        AbsoluteLayout.LayoutBounds="x, y, width, height"
        AbsoluteLayout.LayoutFlags="Set Flags"
    />
</AbsoluteLayout>
```

In Absolute layout, Views take five things to set its position. Distance from left side of Parent (X), distance from top of the Parent (Y), height & width of view and The Layout Flag. In order to set any Child Anywhere in the layout, we need to know about two Important Things -

- Layout Bounds:
- Layout Flags:

Absolute Layout

LAYOUT BOUNDS:

In Layout bounds, we set the Height and width of the view, and distance from the top (y-coordinate) and distance from the left side of view (X-coordinate). In Layout, the bounds values are set as a comma-separated list values in that order - X, Y, Width, Height - Using `LayoutBounds` property as shown in the following XML code -

```
AbsoluteLayout.LayoutBounds="x, y, width, height"
```

- **X:** X is the Horizontal position of the View - Distance from the left side:
- **Y:** Y is the Vertical position of the View - Distance from top:
- **Width:** Width of the View.
- **Height:** Height of the View.

We can set the above Layout Bounds value using two ways:

- Absolute Value
- Proportional Value

Absolute Layout

LAYOUT FLAGS:

The Layout Flags specifies how Layout bounds values will be interpreted. In XAML, bounds and flags are set as a part of the definition of views within the layout using the following syntax.

```
<Button
     AbsoluteLayout.LayoutBounds="x, y, width, height"
     AbsoluteLayout.LayoutFlags="Set Flags"
/>
```

Same Like Layout Bounds, The Layout Flags are also specified in the Declaration of views in the layout using the `LayoutFlags` Property. The Flags can be combined in XAML using a comma-separated list. Following are the Predefined Options of Layout Flags.

Absolute Layout

Absolute Value

Absolute Values Explicitly define where views should be Positioned within the layout. Absolute values will fix the position of views that will not be able to change its position dynamically for other screen sizes. Using Absolute Values for Positioning can be dangerous, when the size of the layout is Unknown. If we fix our view in center of screen at one size using Absolute values, it could be offset on any other device screen at different size. Therefore, it is important to test our app across the various screen sizes of our supported devices.

In Order to use Absolute values, We set layout bounds using x and y coordinates and explicit sizes, then set Layout flags to none. as shown in the given XAML Code -

```
<Button
     AbsoluteLayout.LayoutBounds="85, 185, 200, 200"
/>
```

NOTE: If there is no Layout flags are specified, None will be default flags. so no need to Set the Layout flags to None Using (AbsoluteLayout.LayoutFlags="None".

Absolute Layout

```
<?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:local="clr-namespace:Layout"
             x:Class="Layout.MainPage">
    <ContentPage.Content>
        <AbsoluteLayout>
            <Button
                BackgroundColor="Teal"
                Text="Hi, I am Centered, am i?"
                AbsoluteLayout.LayoutBounds="85, 185, 200, 200"
            />
        </AbsoluteLayout>
    </ContentPage.Content>
</ContentPage>
```

Absolute Layout

Proportional Value:

The Proportional Values define a relationship between a Layout and a View. Unlike Absolute values, Proportional values will not fix the position of views. the views can change dynamically its position and its size. The Proportional values are expressed as double with the values between 0 and 1. the 0(zero) mean 0%, and 1 mean 100%. In simple, the layout set its views value in percentage using proportional value. If we set the view at the top left corner by assigning the value 0(zero) to both x and y, it will be same for all devices.

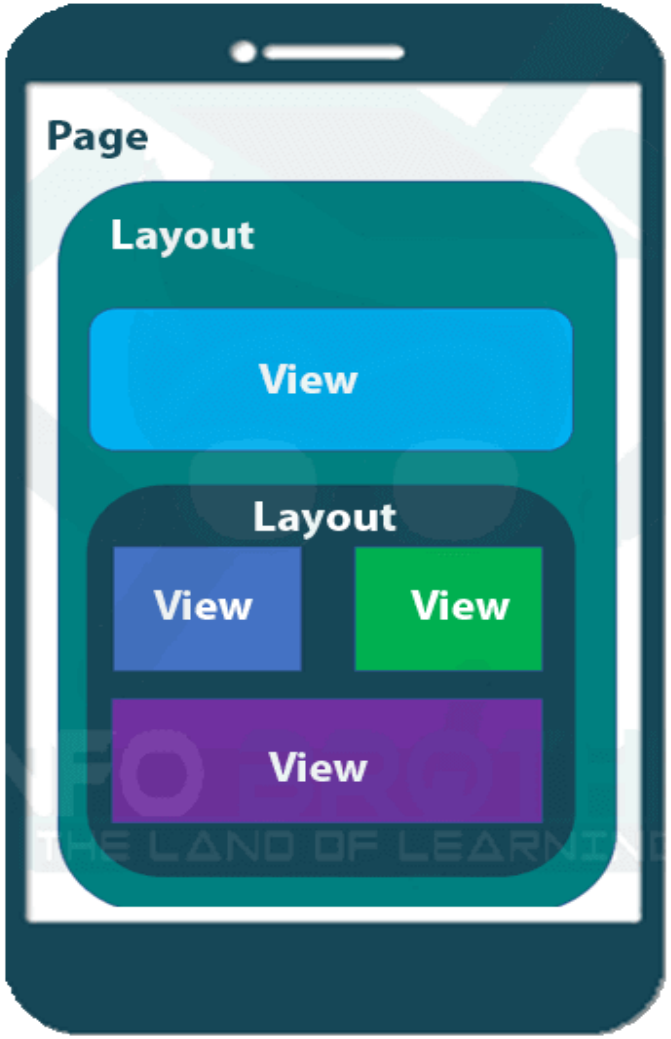
In Order to use Proportional values, we set layout bounds using x and y coordinates and proportional sizes, then set Layout flags to All. as shown in the given XAML Code -

```
<Button
    AbsoluteLayout.LayoutBounds=".50, .50, .50, .30"
    AbsoluteLayout.LayoutFlags="All"
/>
```

Absolute Layout

Option	Description:
None	Default Value - Interprets all values as absolute.
All	Interprets All Values as Proportional.
WidthProportional	Interprets Only the Width value as proportional with all other values absolute.
HeightProportional	Interprets Only the Height value as proportional with all other values absolute.
XProportional	Interprets Only The X value as proportional, with all other values absolute.
YPorportional	Interprets Only The Y value as Proportional, with all other values absolute.
PositionProportional	Interprets the X and Y values as proportional, while the size values are interpreted as absolute.
SizeProportional	Interprets the Width and Height values as proportional, while the size values are interpreted as absolute.

Views



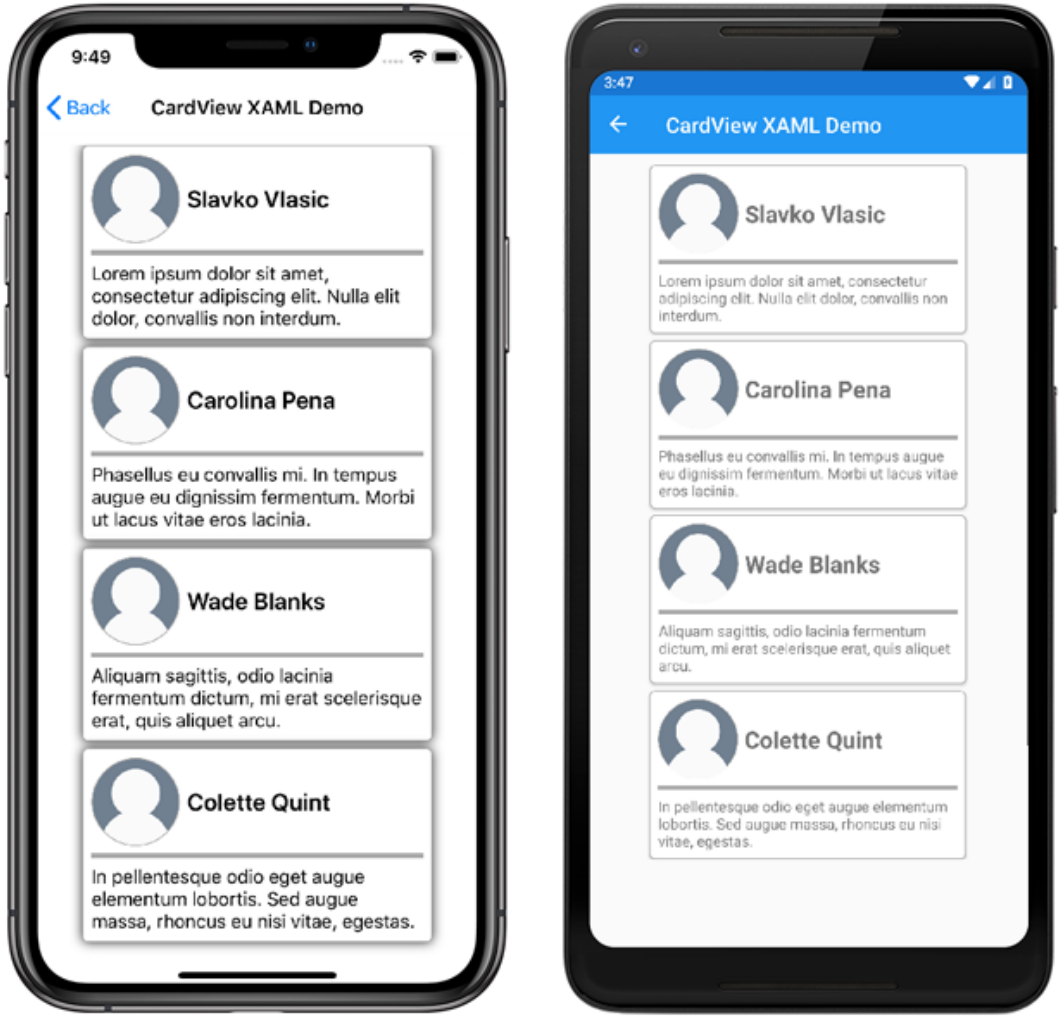
Views

- Box View
- Text View
- Surface View
- Texture View
- Image View
- Card View
- Grid View
- Scroll View
- List View
- Search View
- Web View
- Stack View
- Table View

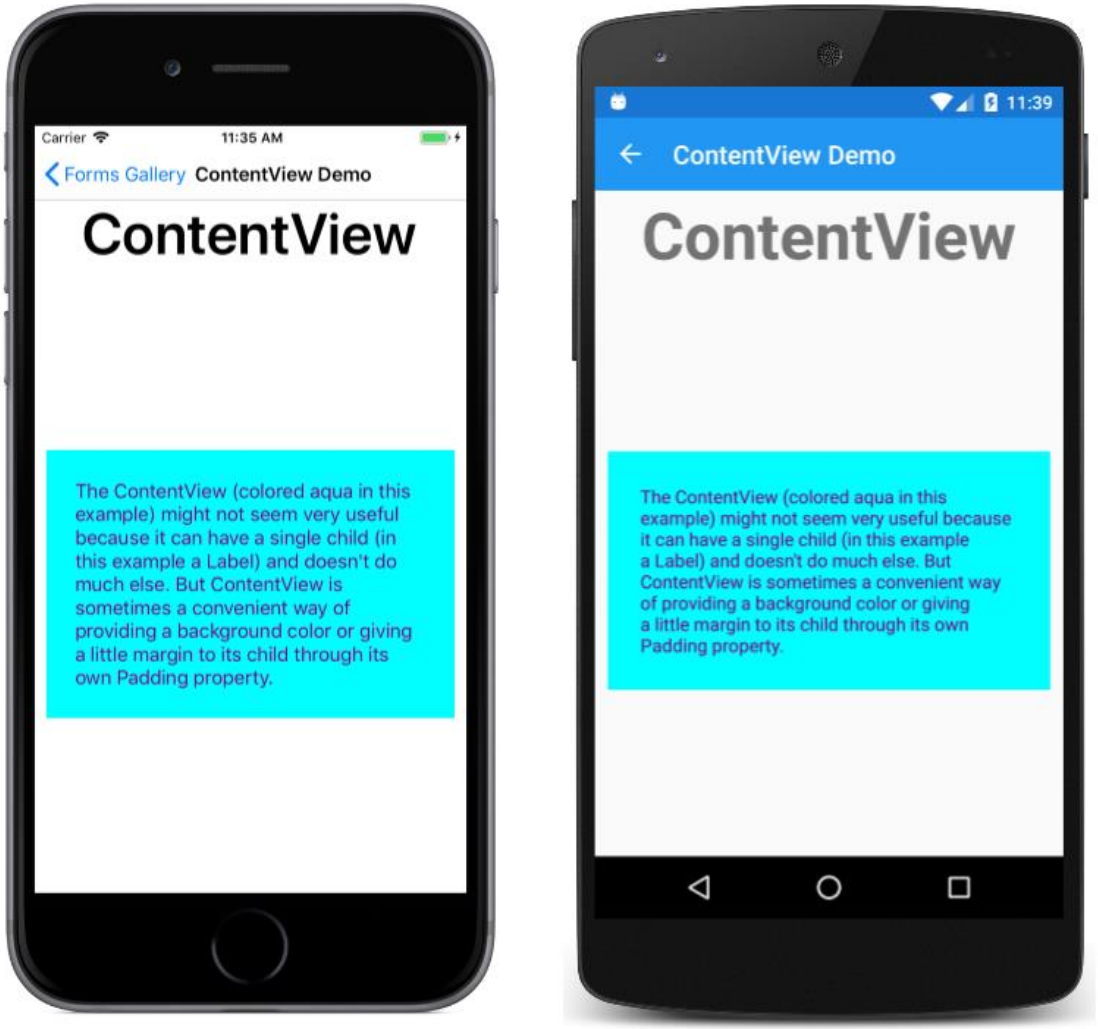
Content View

- Content View contains a single child that is set with the Content property.
- The Content property can be set to any View derivative, including other Layout derivatives.
- Content View is mostly used as a structural element and serves as a base class to Frame.

Content View



Content View



Scroll View

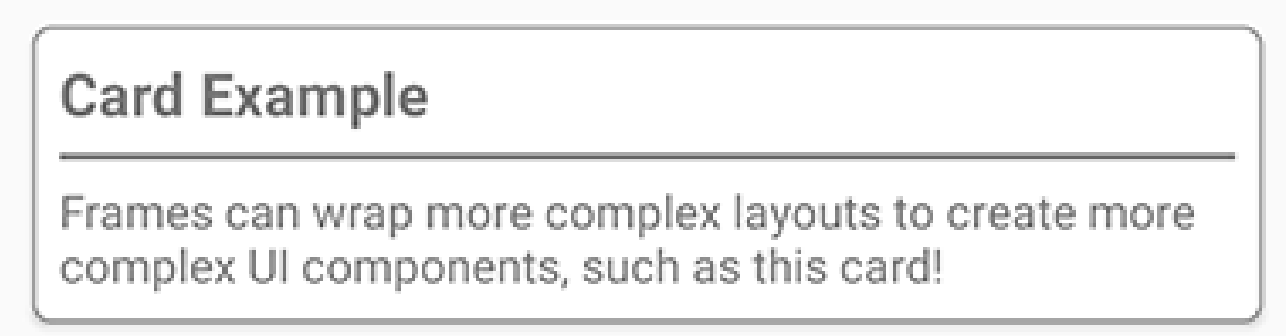
- The Scroll View is used to wrap content that needs to scroll.
- The Scroll View control can only have one child.

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    Padding="10"
    x:Class="Demo.LoginPage">
    <ContentPage.Content>
    <ScrollView>
        <StackLayout>
            <StackLayout VerticalOptions="CenterAndExpand" Spacing="10">
              <Image Source="logo" WidthRequest="100" HeightRequest="100" />
              <Label Text="Login Here!" HorizontalOptions="Center" TextColor="Gray'
              <Entry Placeholder="Username" Keyboard="Email" />
              <Entry Placeholder="Password" IsPassword="true" />
              <StackLayout Orientation="Horizontal">
                    <Button Text="Login" HorizontalOptions="FillAndExpand" Backgrou
                    <Button Text="Register" HorizontalOptions="FillAndExpand" Backg
              </StackLayout>
            </StackLayout>
        </StackLayout>
    </ScrollView>
    </ContentPage.Content>
```

Frame

- The Xamarin.Forms Frame class is a layout used to wrap a view with a border that can be configured with color, shadow, and other options.
- Frames are commonly used to create borders around controls but can be used to create more complex UI.
- The Frame class defines the following properties:
  - BorderColor is a Color value that determines the color of the Frame border.
  - CornerRadius is a float value that determines the rounded radius of the corner.
  - HasShadow is a bool value that determines whether the frame has a drop shadow.

Frame



C# Copy

```
Frame circleImageFrame = new Frame
{
    Margin = 10,
    BorderColor = Color.Black,
    CornerRadius = 50,
    HeightRequest = 60,
    WidthRequest = 60,
    IsClippedToBounds = true,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center,
    Content = new Image
    {
        Source = ImageSource.FromFile("outdoors.jpg"),
        Aspect = Aspect.AspectFill,
        Margin = -20,
        HeightRequest = 100,
        WidthRequest = 100
    }
};
```

FlexLayout

- Flex Layout provides different ways for allocating components in the screen, making alignment, design and spaces organization way easier.
- This layout have the power to give a better proportional size for the components inside, because it arrange elements in a ratio based in screen dimensions and among elements in the screen.
- Keeping your view cleaner and more organized.

FlexLayout

JustifyContent:

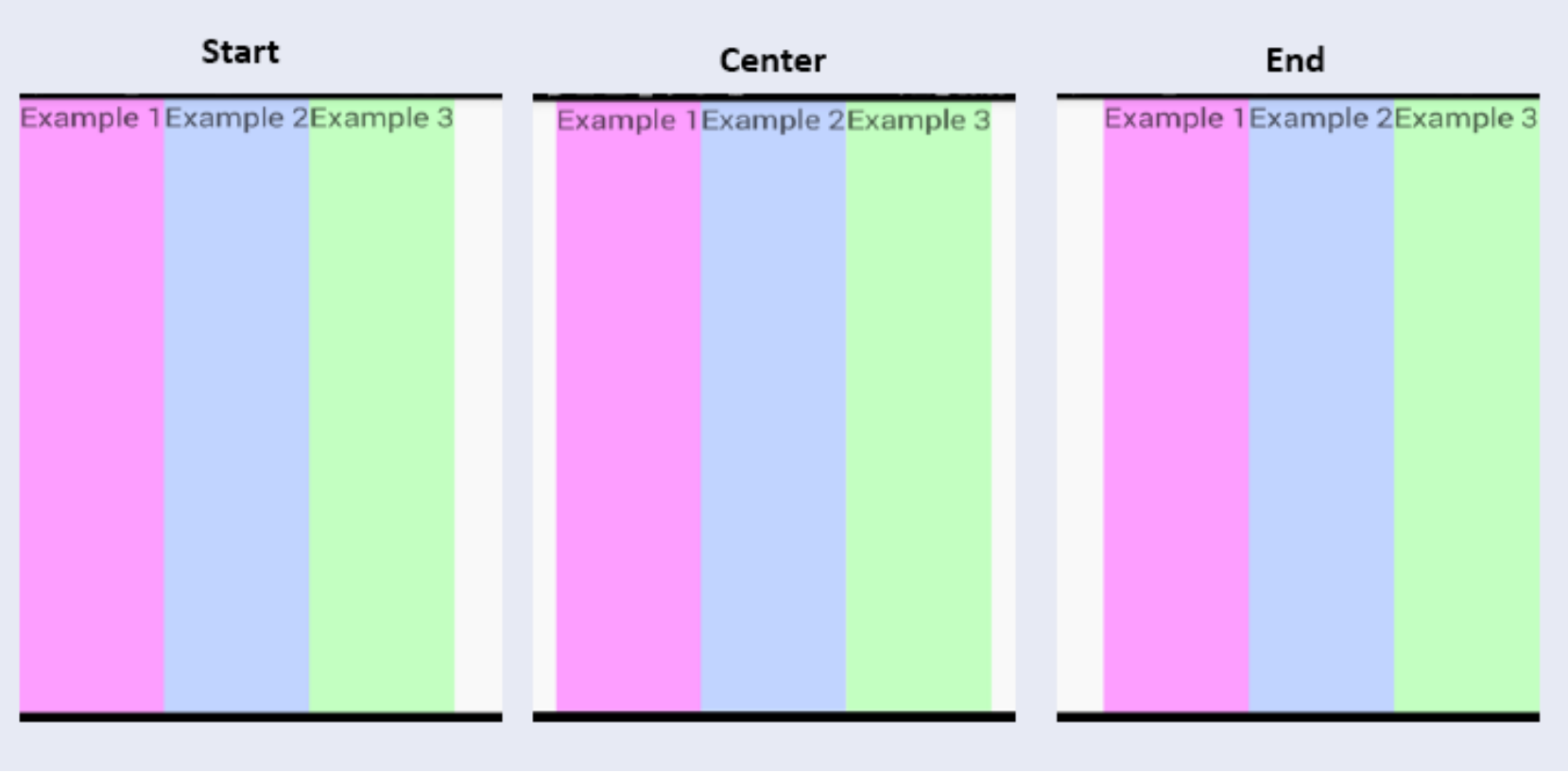
This property define how will be organize the content.

```
<FlexLayout JustifyContent = "Start|Center|SpaceAround|SpaceBetween|SpaceEvenly">
    <Label Text = "Example 1" BackgroundColor = "#ffb3ff" FontSize = "25"/>
    <Label Text = "Example 2" BackgroundColor = "#ccddff" FontSize = "25"/>
    <Label Text = "Example 3" BackgroundColor = "#ccffcc" FontSize = "25"/>
</FlexLayout>
```

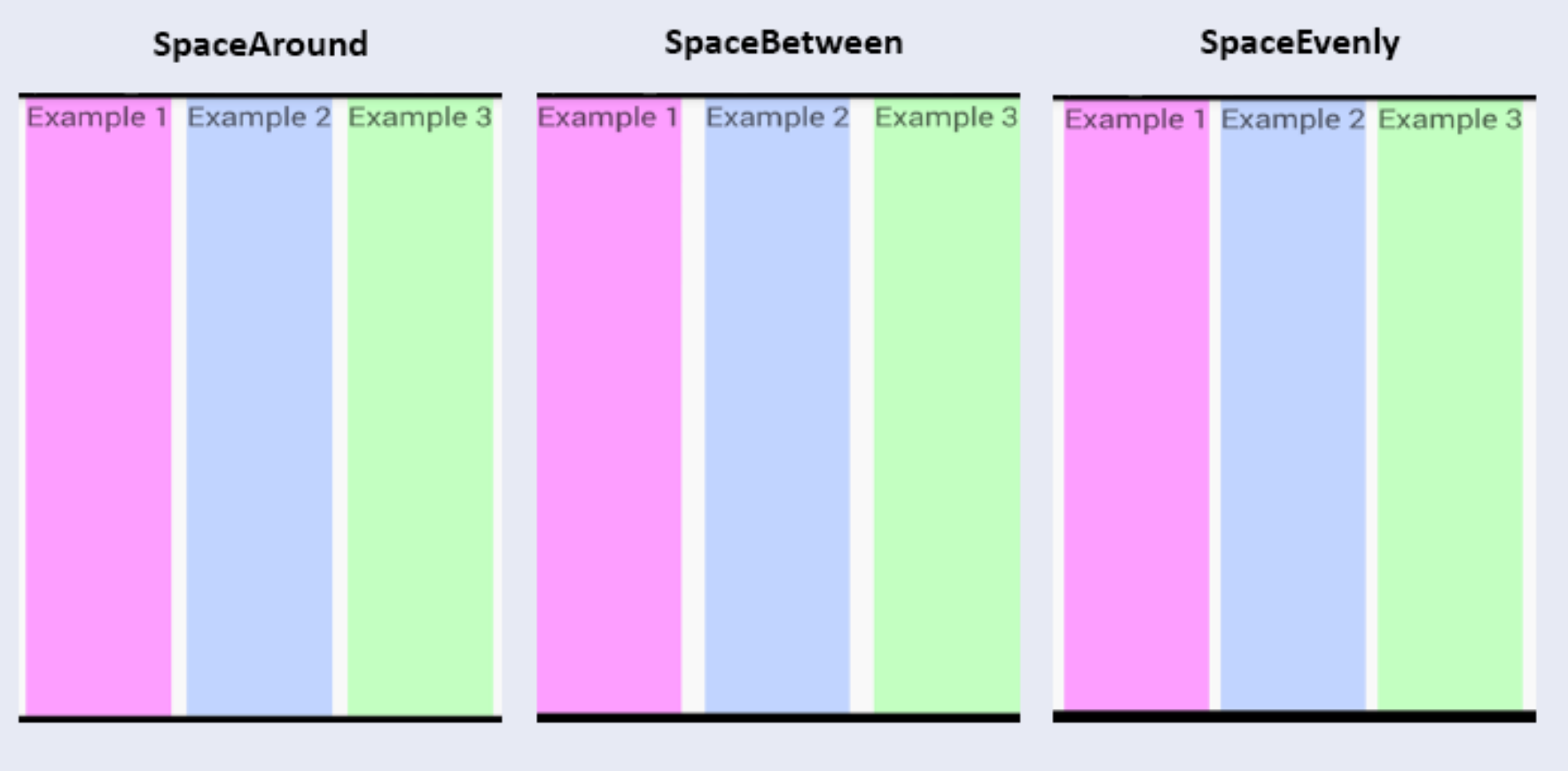
Supported values. Organize your elements as follows:

- **Start:** At begin of the container. **(Default value).**
- **Center:** Center of the container.
- **End:** End of the container.
- **SpaceAround:** Begins with one unit of space for the borders and two units respect to others elements in the container.
- **SpaceBetween:** Gets same space between them.
- **SpaceEvenly:** Gets same spaces between borders and the other elements in the container.

FlexLayout


Start
Example 1
Example 2
Example 3
Center
Example 1
Example 2
Example 3
End
Example 1
Example 2
Example 3

FlexLayout


SpaceAround
Example 1
Example 2
Example 3
SpaceBetween
Example 1
Example 2
Example 3
SpaceEvenly
Example 1
Example 2
Example 3

FlexLayout

AlignItems:

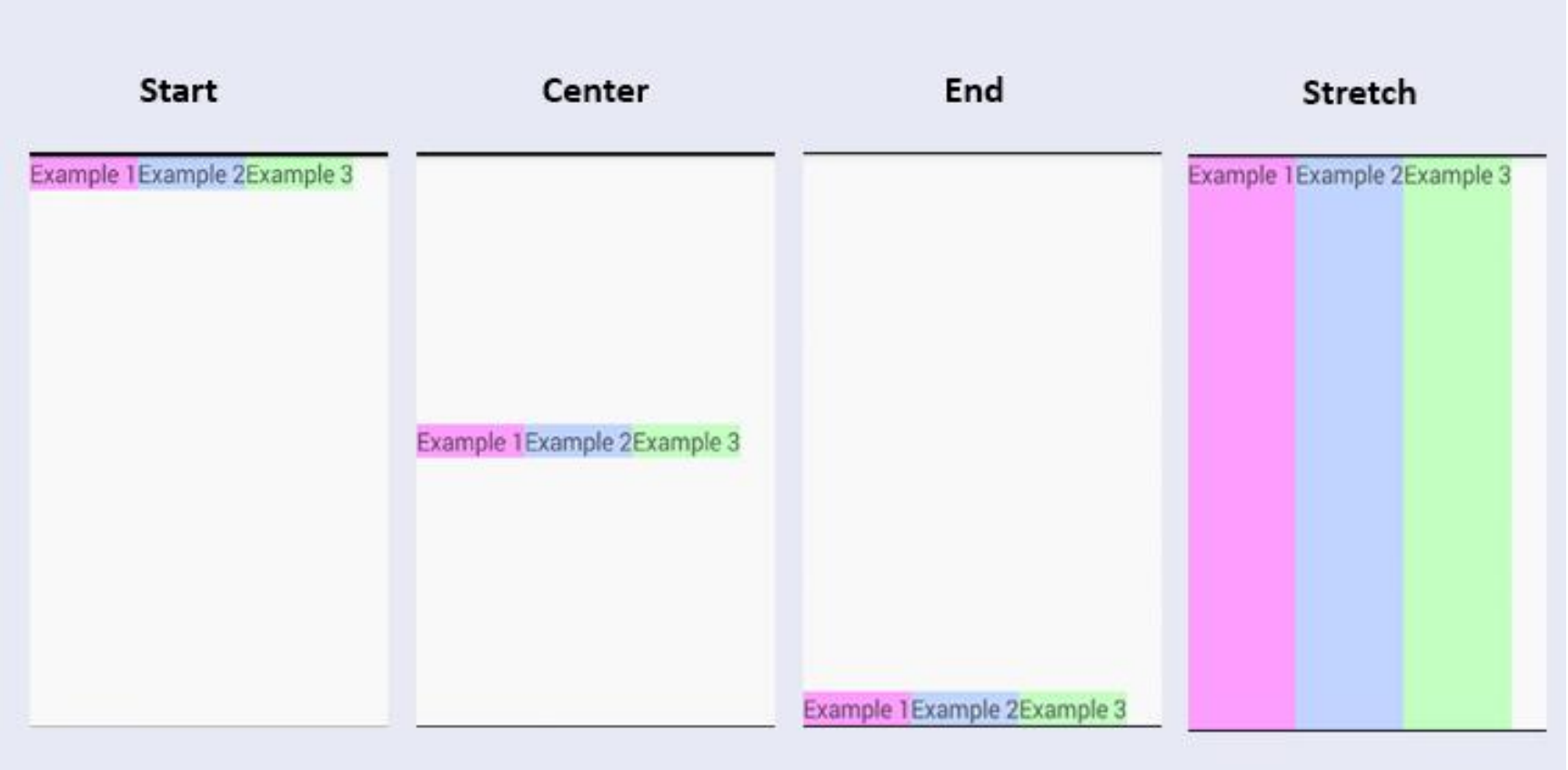
Defines elements behavior alignment in the container.

```
<FlexLayout AlignItems = "Start|Center|End|Stretch">
    <Label Text = "Example 1" BackgroundColor = "#ffb3ff" FontSize = "25"/>
    <Label Text = "Example 2" BackgroundColor = "#ccddff" FontSize = "25"/>
    <Label Text = "Example 3" BackgroundColor = "#ccffcc" FontSize = "25"/>
</FlexLayout>
```

Supported values. Organize your elements as follows:

- **Start:** Your elements at begin of the container just getting a line space.
- **Center:** Puts elements in the center of the container.
- **End:** Elements at the end of the container.
- **Stretch:** Your elements at begin of the container in a horizontal stretched. **(Default value)**

FlexLayout


Start
Center
End
Stretch
Example 1Example 2Example 3
Example 1Example 2Example 3
Example 1Example 2Example 3
Example 1Example 2Example 3

FlexLayout

Direction:

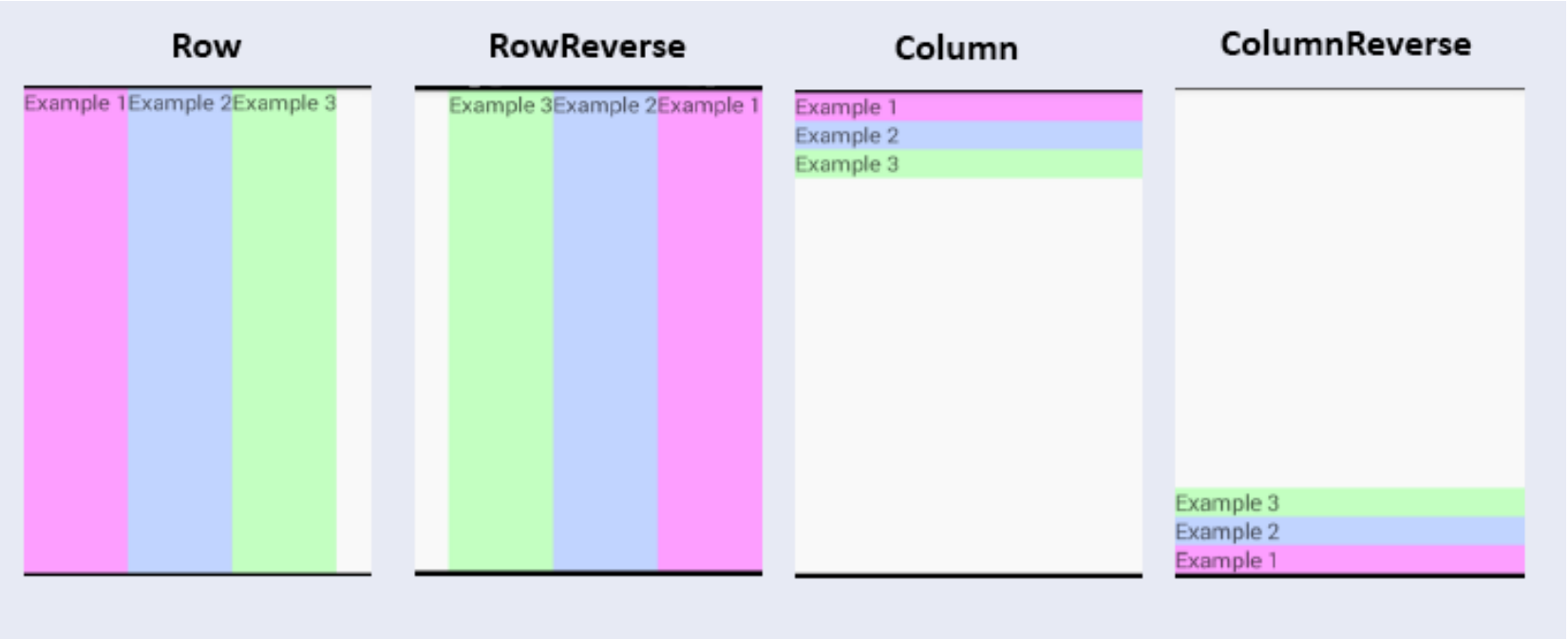
Indicates the direction of the elements in the screen and you can use the values that I show below.

```
<FlexLayout Direction = "Column|ColumnReverse|Row|RowReverse">
    <Label Text = "Example 1" BackgroundColor = "#ffb3ff" FontSize = "25"/>
    <Label Text = "Example 2" BackgroundColor = "#ccddff" FontSize = "25"/>
    <Label Text = "Example 3" BackgroundColor = "#ccffcc" FontSize = "25"/>
</FlexLayout>
```

Supported values. Organize your elements as follows:

- **Row:** **Left to Right** in a vertical way **(Default value)**
- **RowReverse:** **Right to Left** in a vertical way.
- **Column:** **Top to Bottom** in a horizontal way.
- **ColumnReverse:** **Bottom to Top** in a horizontal way.

FlexLayout


Row
Example 1
Example 2
Example 3
RowReverse
Example 3
Example 2
Example 1
Column
Example 1
Example 2
Example 3
ColumnReverse
Example 3
Example 2
Example 1

FlexLayout

Wrap:

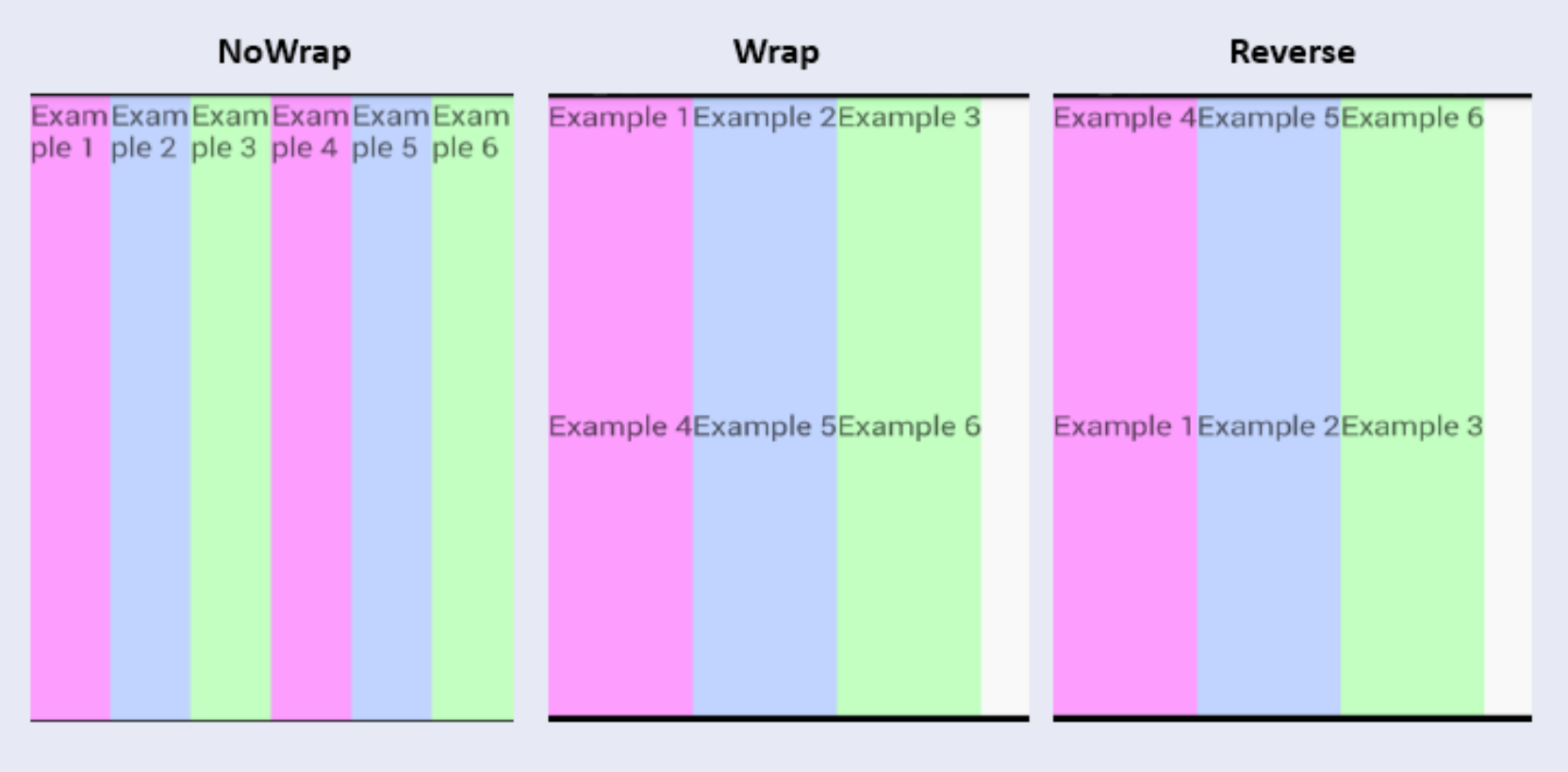
Organizes location of the components. By default, FlexLayout elements put them all in just one line. With this property we can change this behavior.

```
<FlexLayout  Wrap =  "Wrap|NoWrap|Reverse">
     <Label Text = "Example 1" BackgroundColor = "#ffb3ff" FontSize = "25"/>
     <Label Text = "Example 2" BackgroundColor = "#ccddff" FontSize = "25"/>
     <Label Text = "Example 3" BackgroundColor = "#ccffcc" FontSize = "25"/>
     <Label Text = "Example 4" BackgroundColor = "#ffb3ff" FontSize = "25"/>
     <Label Text = "Example 5" BackgroundColor = "#ccddff" FontSize = "25"/>
     <Label Text = "Example 6" BackgroundColor = "#ccffcc" FontSize = "25"/>
</FlexLayout>
```

Supported values. Organize your elements as follows:

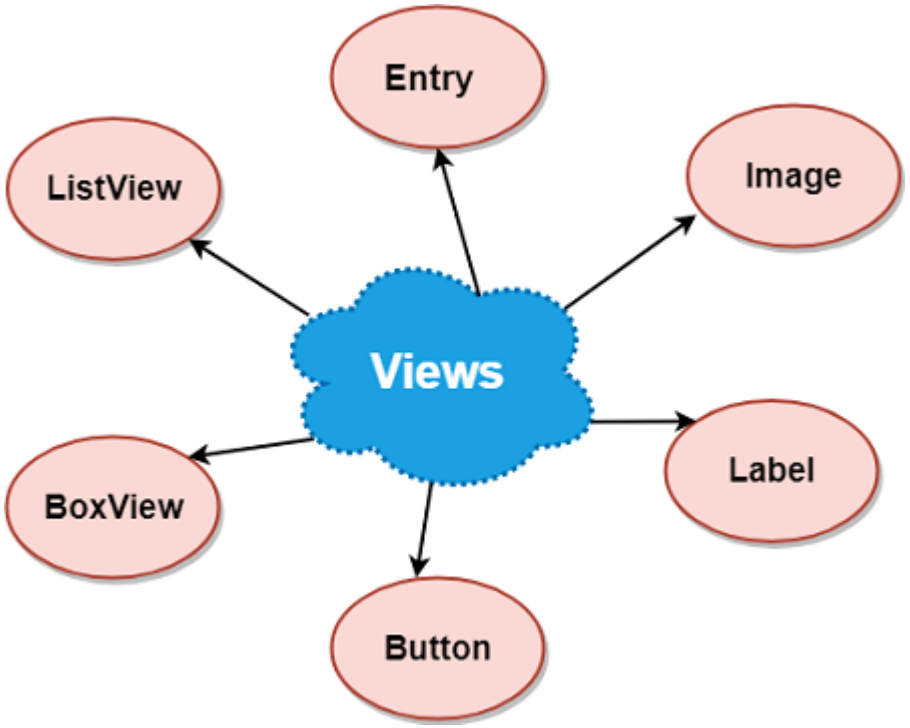
- **NoWrap:** All your Elements in the same line. **(Default value)**
- **Wrap:** Elements in multiple lines from Top to Bottom.
- **Reverse:** Elements in multiple lines from Bottom to Top.

FlexLayout

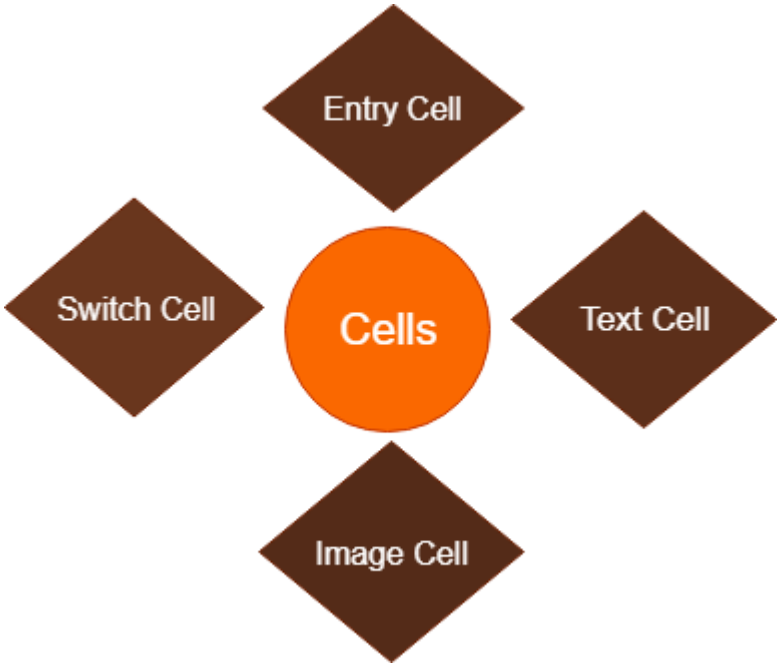

NoWrap
Example 1
Example 2
Example 3
Example 4
Example 5
Example 6
Wrap
Example 1
Example 2
Example 3
Example 4
Example 5
Example 6
Reverse
Example 4
Example 5
Example 6
Example 1
Example 2
Example 3

Xamarin.forms UI

Layouts	Description
StackLayout	In StackLayout, all the child elements are kept in a line. StackLayout is the most used layout.
AbsoluteLayout	A view that positioned the child layout at a specified position using anchors to define the place and size.
RelativeLayout	In RelativeLayout, we position the elements relative to each other using constraint.
Grid	Arrange the multiple views in rows and columns, just like a table.



Xamarin.forms UI



Cells	Description
EntryCell	The cell is containing a label and single-line entry element
SwitchCell	This cell is the same as a switch, but before that, there is a label.
TextCell	The cell contains both primary and secondary field.
ImageCell	Text cell that also contains the image

Xamarin Form Solution

```
namespace FormsExample
{
    public class App : Application
    {
        public App()
        {
            MainPage = new ContentPage
            {
                Content = new StackLayout
                {
                    VerticalOptions = LayoutOptions.Center,
                    Children = {
                        new Label {
                            XAlign = TextAlignment.Center,
                            Text = "Welcome to Xamarin Forms!"
                        }
                    }
                }
            };
        }

        protected override void OnStart()
        {
        }

        protected override void OnSleep()
        {
        }

        protected override void OnResume()
        {
        }
    }
}
```

Application Lifecycle Methods: OnStart, OnSleep, and OnResume

- When the user clicks the Back or Home (or App Switcher) buttons on their device, an app moves into the background. When they reselect the app again, it resumes and moves back into the foreground.
- The starting of an app, the progression of the app from the foreground into a background state then back into the foreground again, until termination, is called the application lifecycle.
- The Application class includes three virtual methods to handle lifecycle events:
  - OnStart – Called when the app is first started
  - OnSleep – Called each time the app is moved into the background
  - OnResume – Called when the app is resumed after being in the background
- OnSleep is also used for normal application termination (not a crash). Any time an app moves into a background state, it must be assumed that it may never return from that state.

Hierarchical Navigation

- The Navigation Page class provides a hierarchical navigation experience where the user is able to navigate through pages, forwards and backwards, as desired.
- The class implements navigation as a last-in, first-out (LIFO) stack of Page objects.

Hierarchical Navigation

To move from one page to another, an application will push a new page onto the navigation stack, where it will become the active page, as shown in the following diagram:

To return back to the previous page, the application will pop the current page from the navigation stack, and the new topmost page becomes the active page, as shown in the following diagram:

Navigation methods are exposed by the Navigation property on any Page derived types. These methods provide the ability to push pages onto the navigation stack, to pop pages from the navigation stack, and to perform stack manipulation.

Performing Navigation

```
public App ()
{
  MainPage = new NavigationPage (new Page1Xaml ());
}
```

C# Copy

```
async void OnNextPageButtonClicked (object sender, EventArgs e)
{
  await Navigation.PushAsync (new Page2Xaml ());
}
```

C# Copy

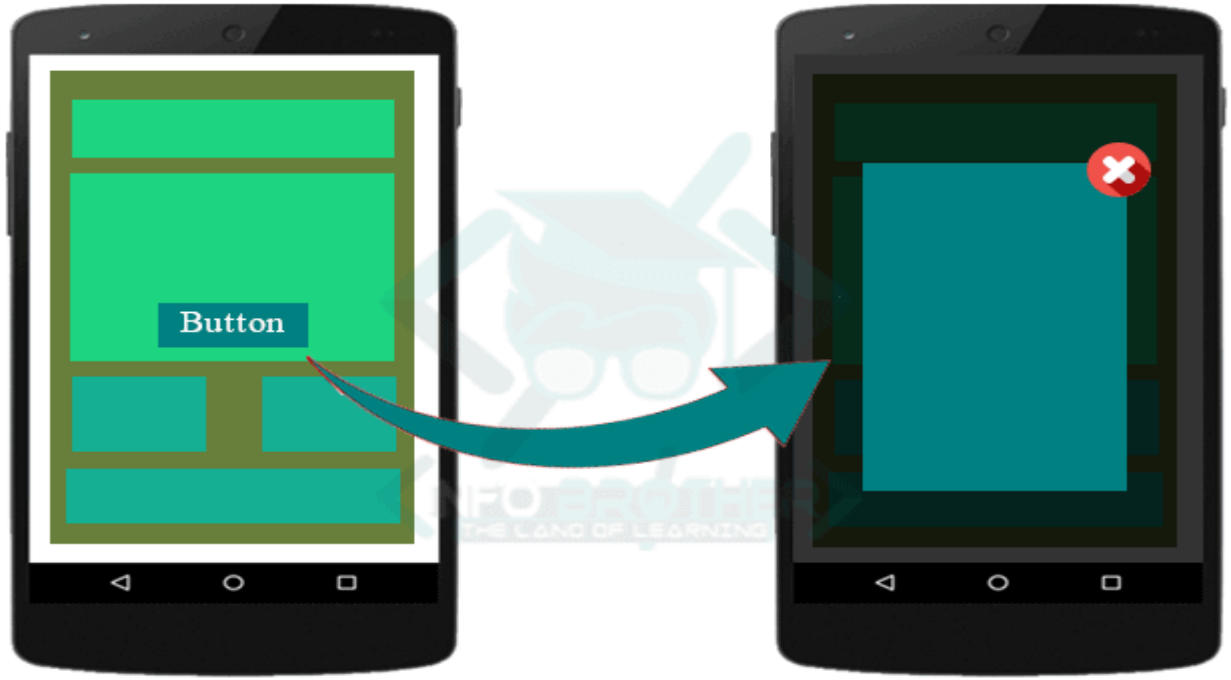
```
async void OnPreviousPageButtonClicked (object sender, EventArgs e)
{
  await Navigation.PopAsync ();
}
```

Performing Navigation

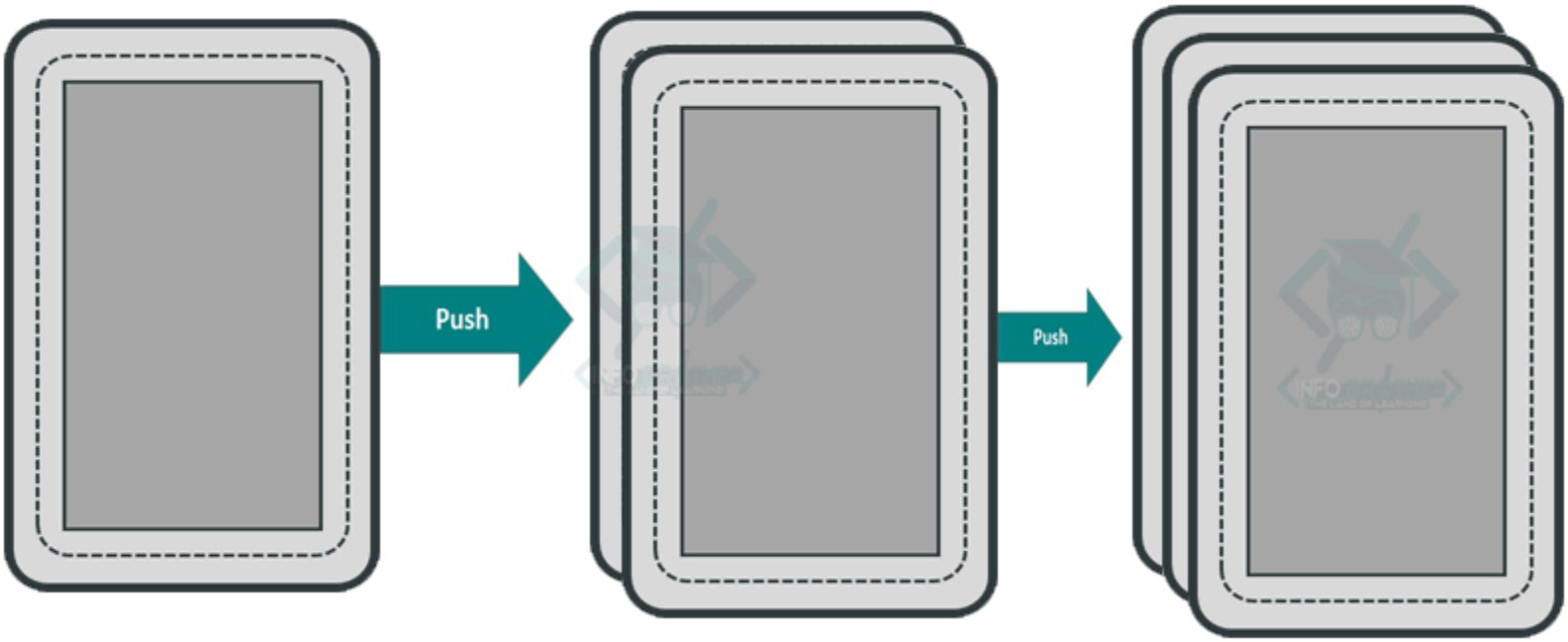
```
async void OnNextPageButtonClicked (object sender, EventArgs e)
{
  // Page appearance not animated
  await Navigation.PushAsync (new Page2Xaml (), false);
}

async void OnPreviousPageButtonClicked (object sender, EventArgs e)
{
  // Page appearance not animated
  await Navigation.PopAsync (false);
}

async void OnRootPageButtonClicked (object sender, EventArgs e)
{
  // Page appearance not animated
  await Navigation.PopToRootAsync (false);
}
```

Modal Page

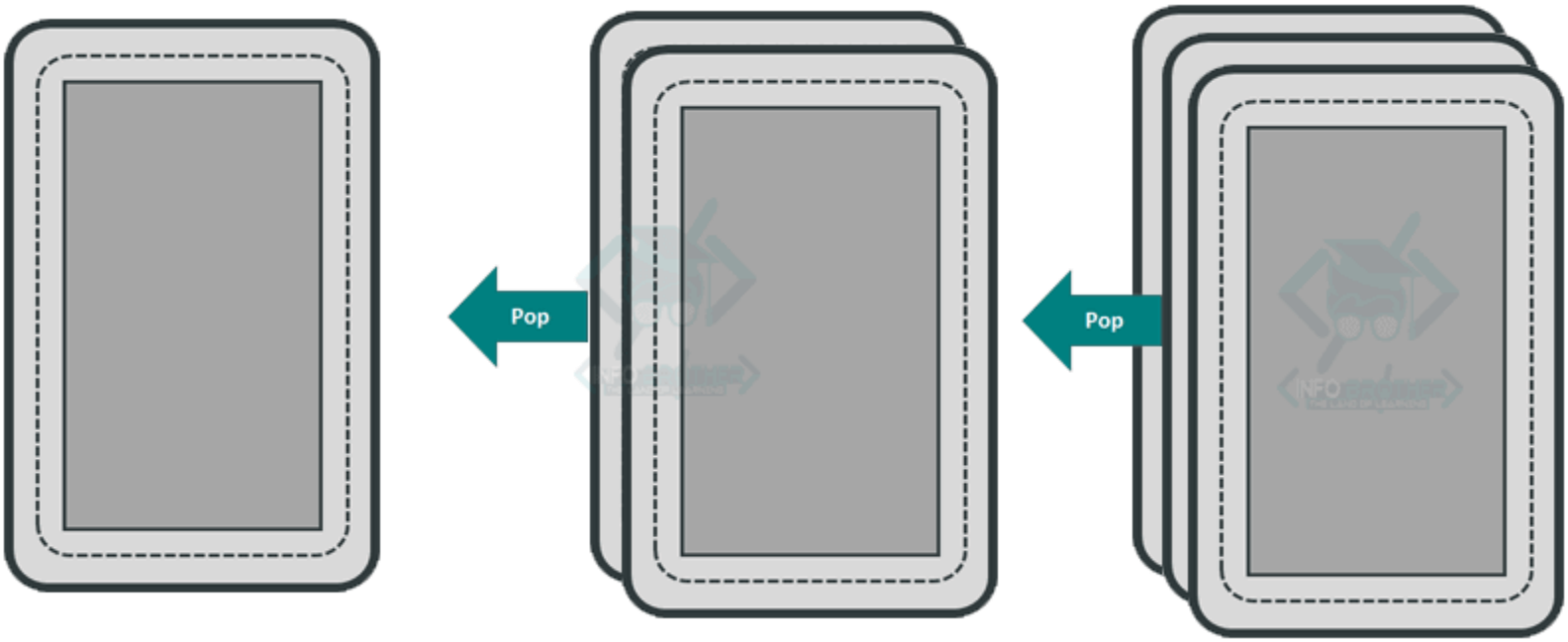
- Modal Page is the child window to a parent window.
- It just works like pop up window.
- It stays in front of parent window till user cancels or completes the self contained task.



Modal Page



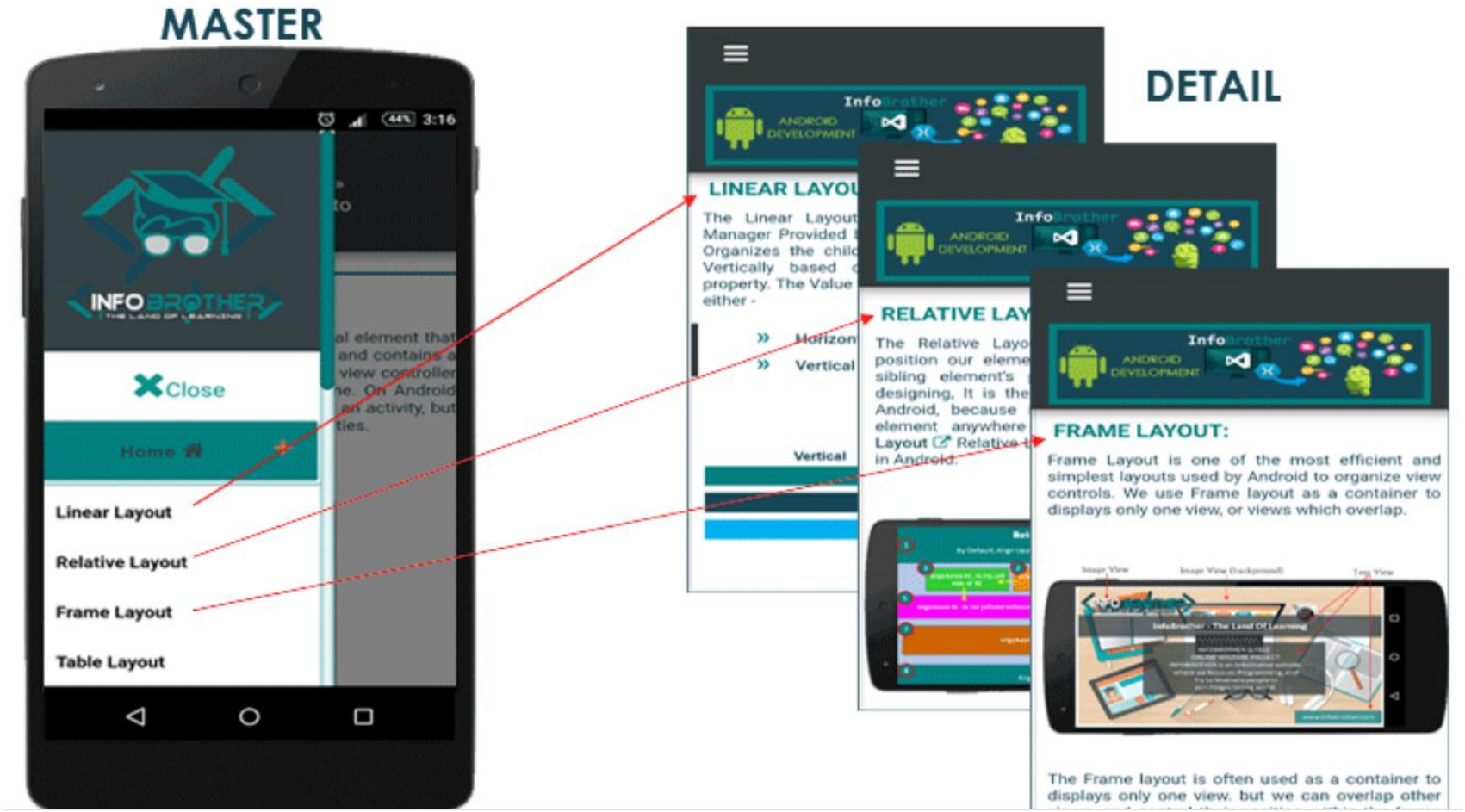
Modal Page



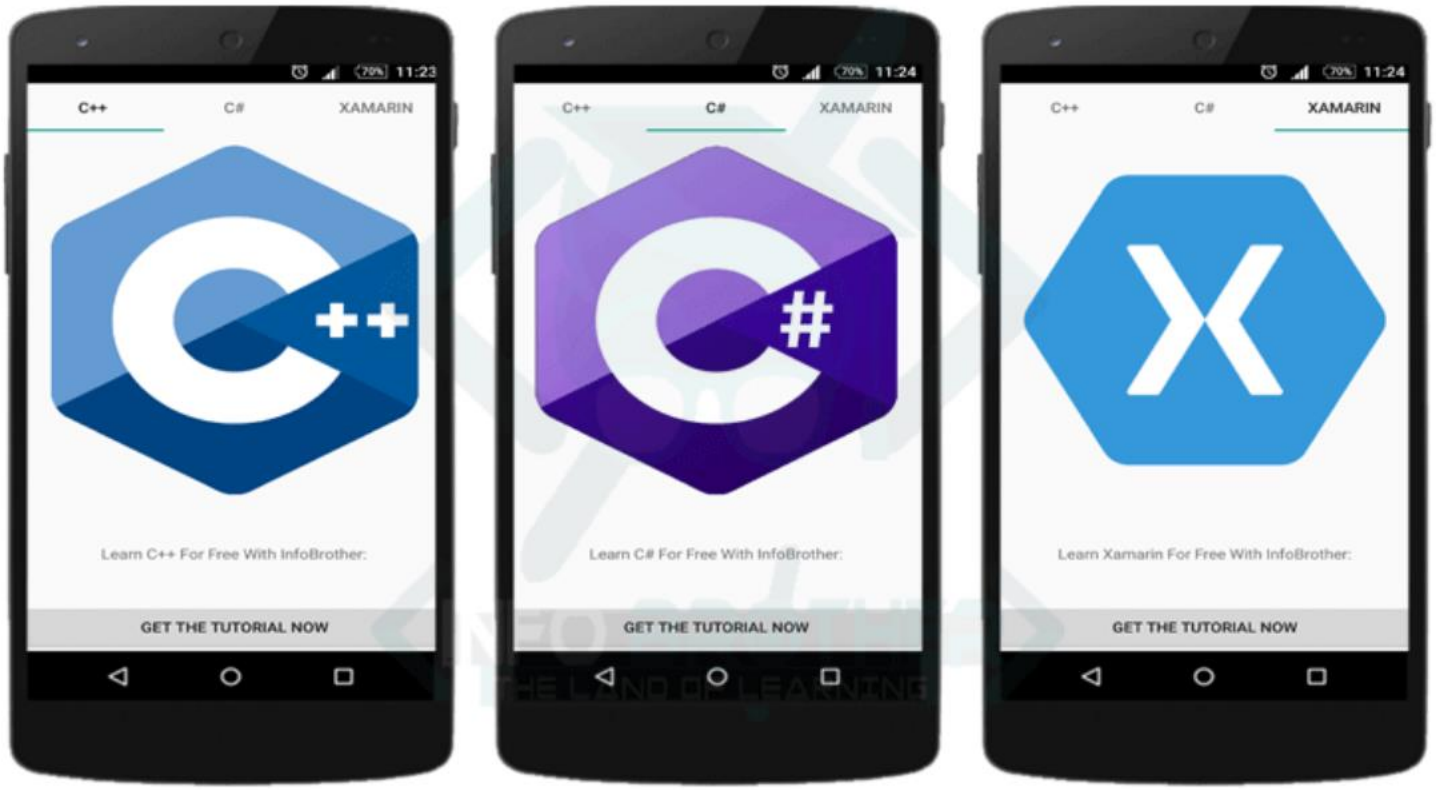
Modal Page


9:38
CLICK ME TO OPEN MODAL PAGE
9:38
Its me, Your Modal Page
CLOSE ME

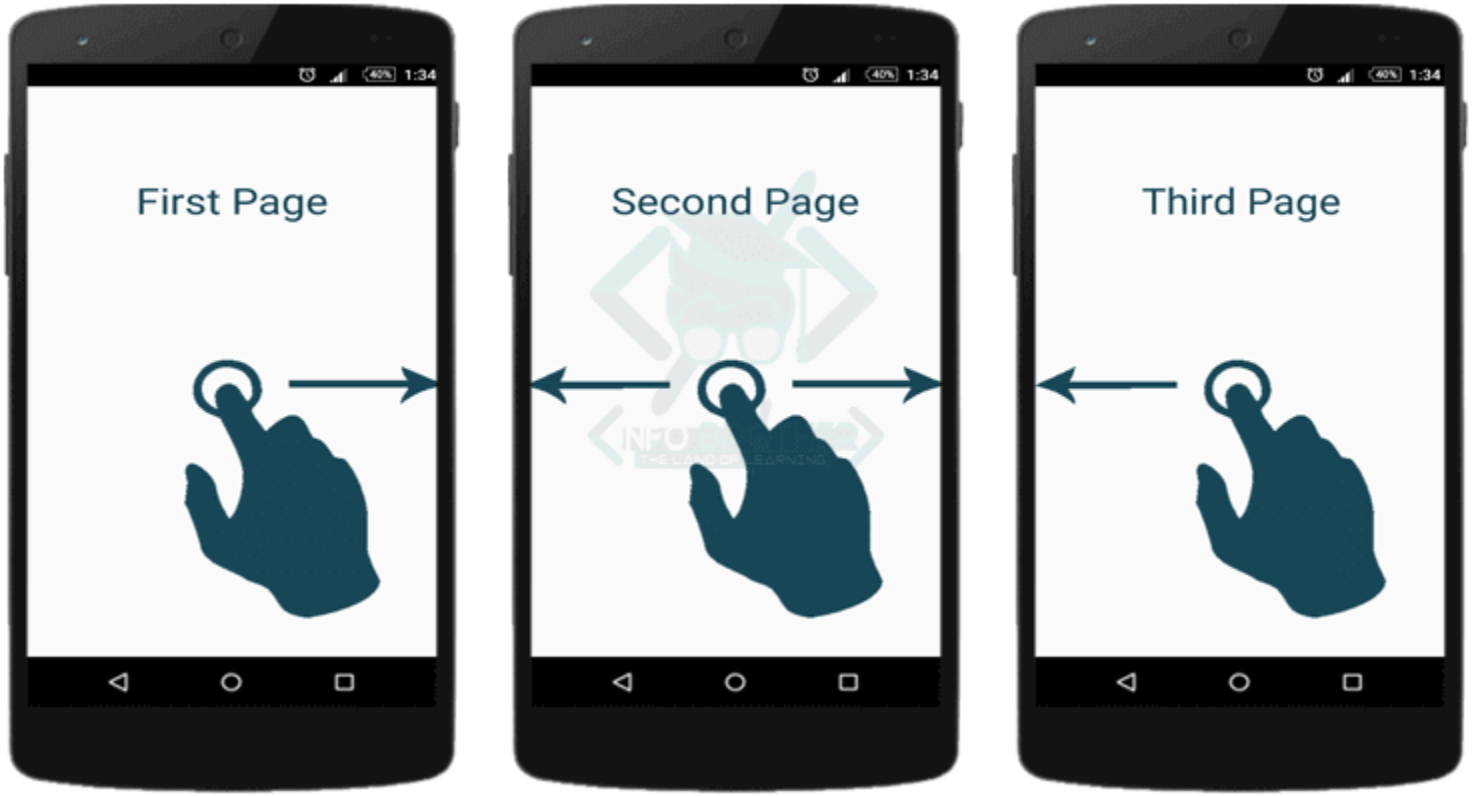
Master Detail page



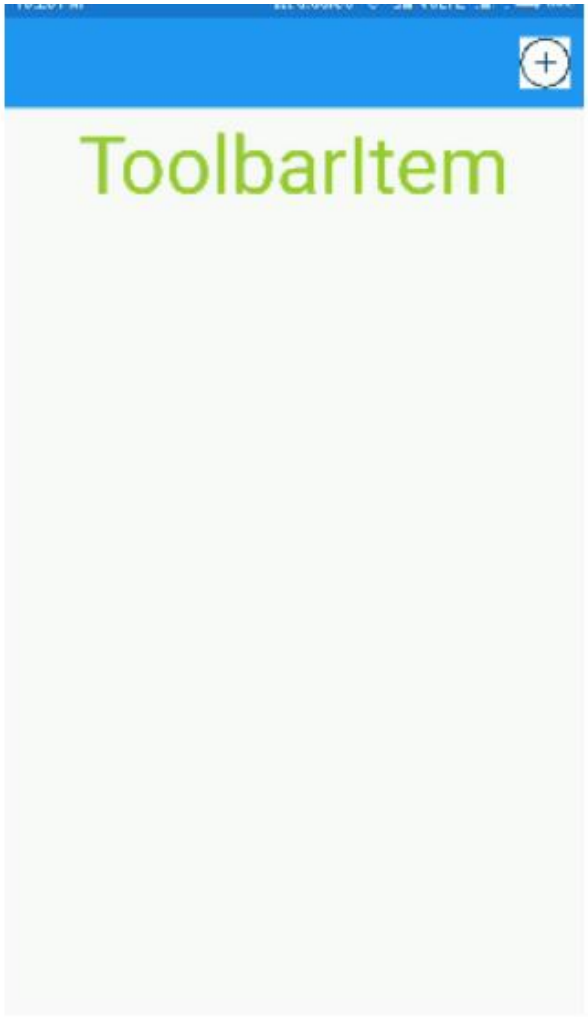
Tabbed Page


11:23
C++
C#
XAMARIN
Learn C++ For Free With InfoBrother:
GET THE TUTORIAL NOW
11:24
C++
C#
XAMARIN
Learn C# For Free With InfoBrother:
GET THE TUTORIAL NOW
11:24
C++
C#
XAMARIN
Learn Xamarin For Free With InfoBrother:
GET THE TUTORIAL NOW

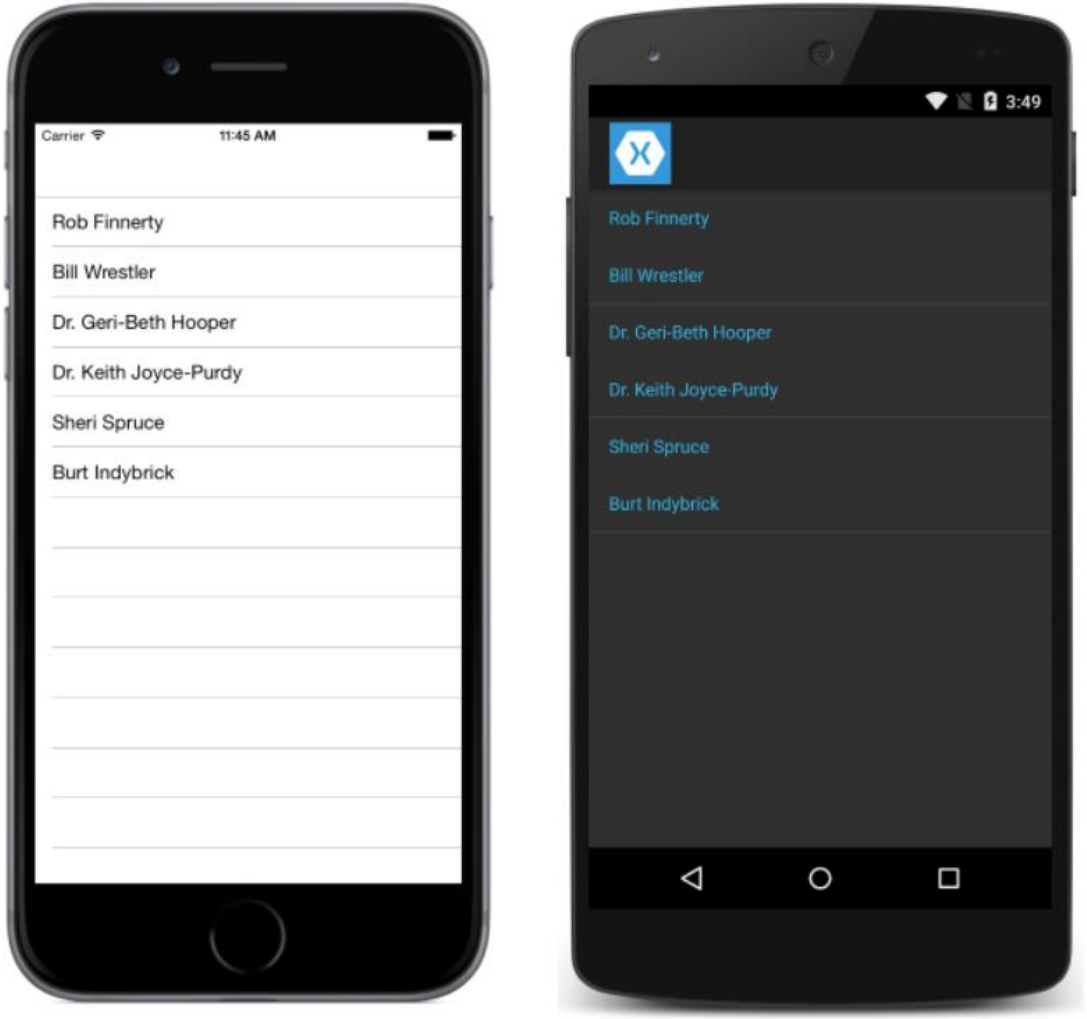
Carousel Page



ToolBar Items



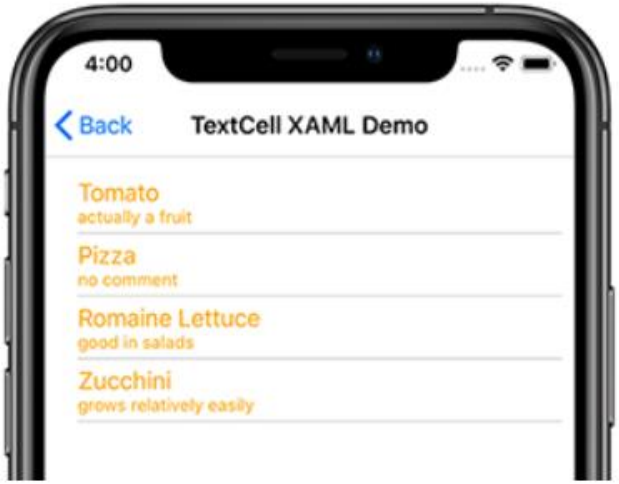
List View

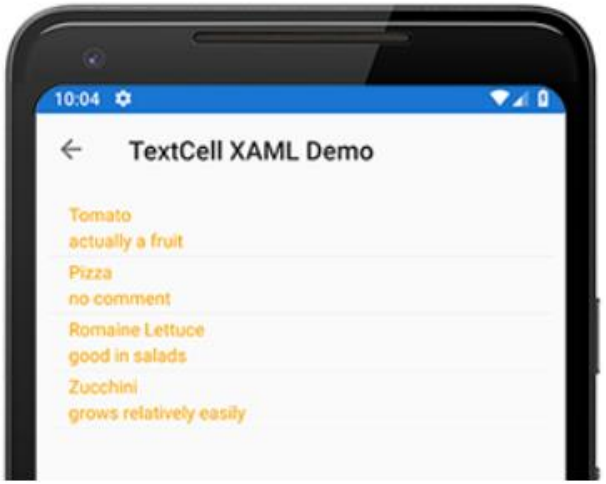

Carrier
11:45 AM
Rob Finnerty
Bill Wrestler
Dr. Geri-Beth Hooper
Dr. Keith Joyce-Purdy
Sheri Spruce
Burt Indybrick
3:49
Rob Finnerty
Bill Wrestler
Dr. Geri-Beth Hooper
Dr. Keith Joyce-Purdy
Sheri Spruce
Burt Indybrick

Customizing ListView Cell Appearance

- Built in Cells
- Xamarin.Forms comes with built-in cells that work for many applications:
  - TextCell controls are used for displaying text with an optional second line for detail text.
  - ImageCell controls are similar to TextCells but include an image to the left of the text.
  - SwitchCell controls are used to present and capture on/off or true/false states.
  - EntryCell controls are used to present text data that the user can edit.

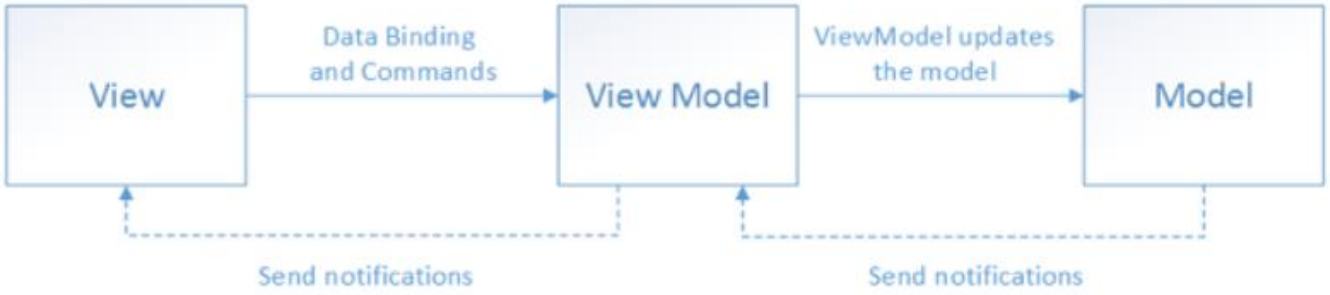
TextCell,Image Cell



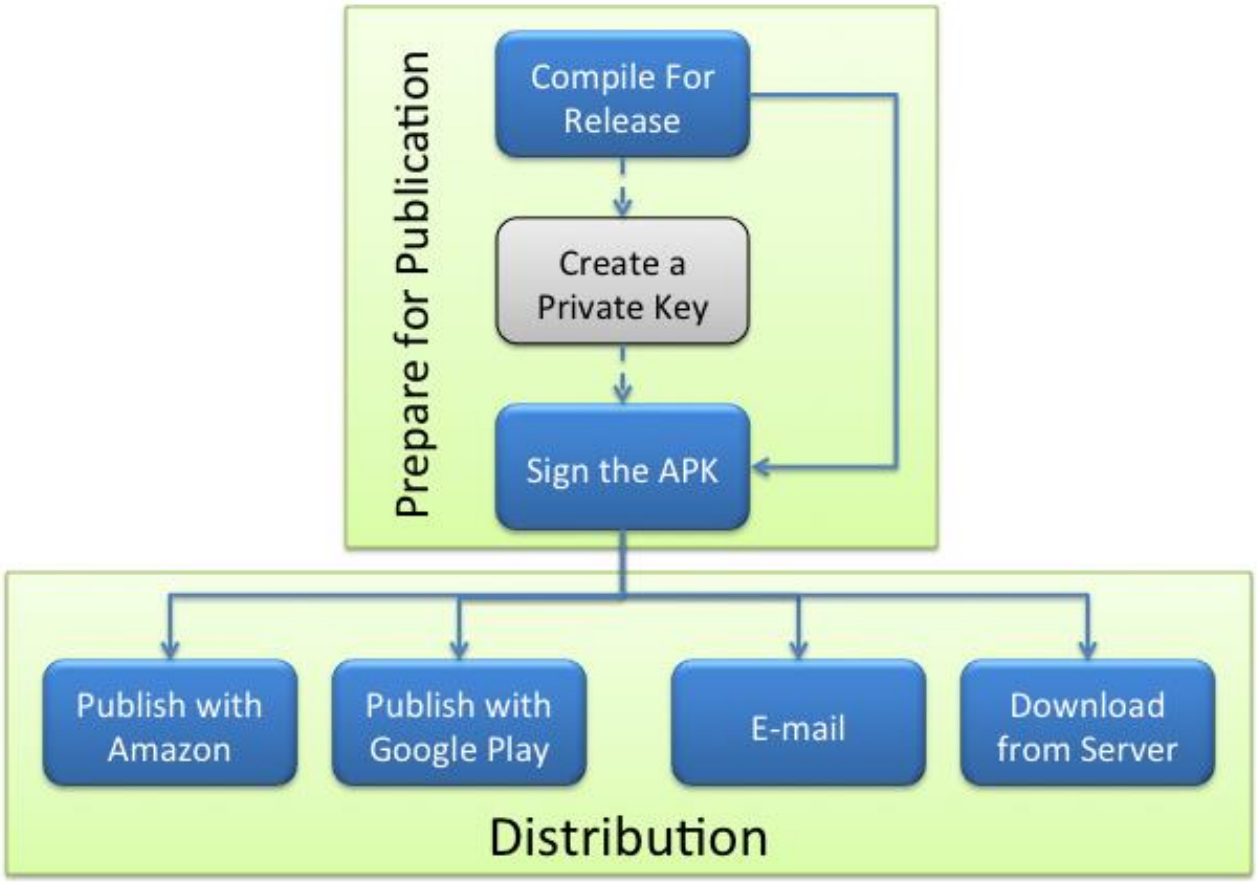


The MVVM Pattern

There are three core components in the MVVM pattern: the model, the view, and the view model. Each serves a distinct purpose. Figure 2-1 shows the relationships between the three components.



Xamarin Publishing Application



Questions

Module Summary

- Xamarin Architecture
- Xamarin Forms
- Data Binding MVVM
- Data Access Sqlite
- Rendering and Publishing Mobile Application