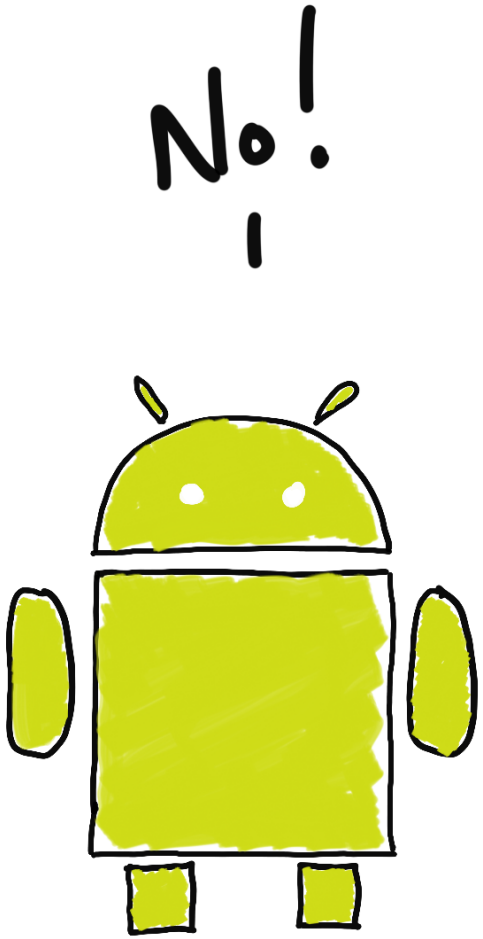
Deep Into the Woods with Xamarin.Forms

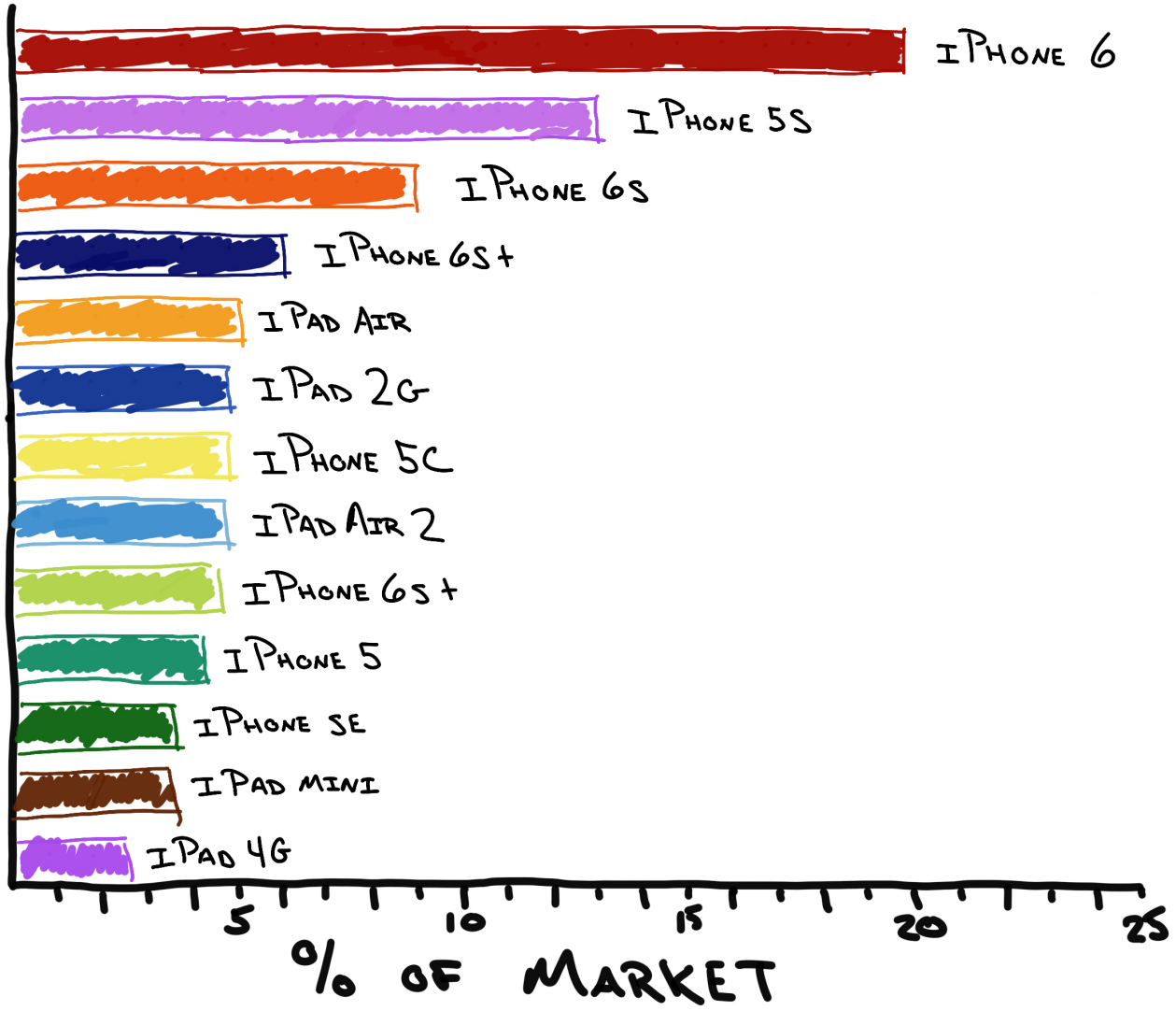
MATTHEW SOUCOUP
Principal
@codemillmatt codemilltech.com

http://bit.ly/forms-woods

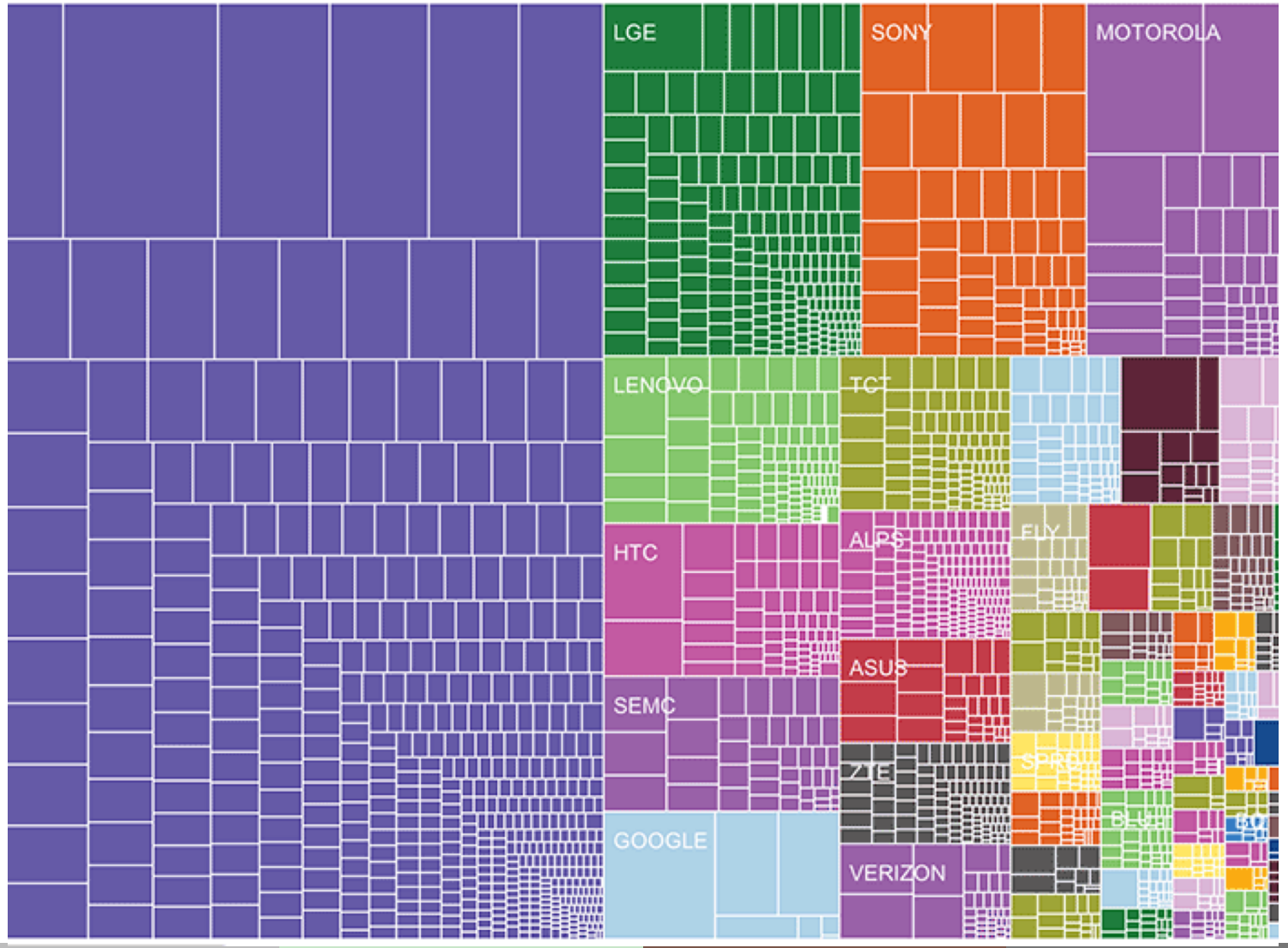
Mobile apps are difficult…



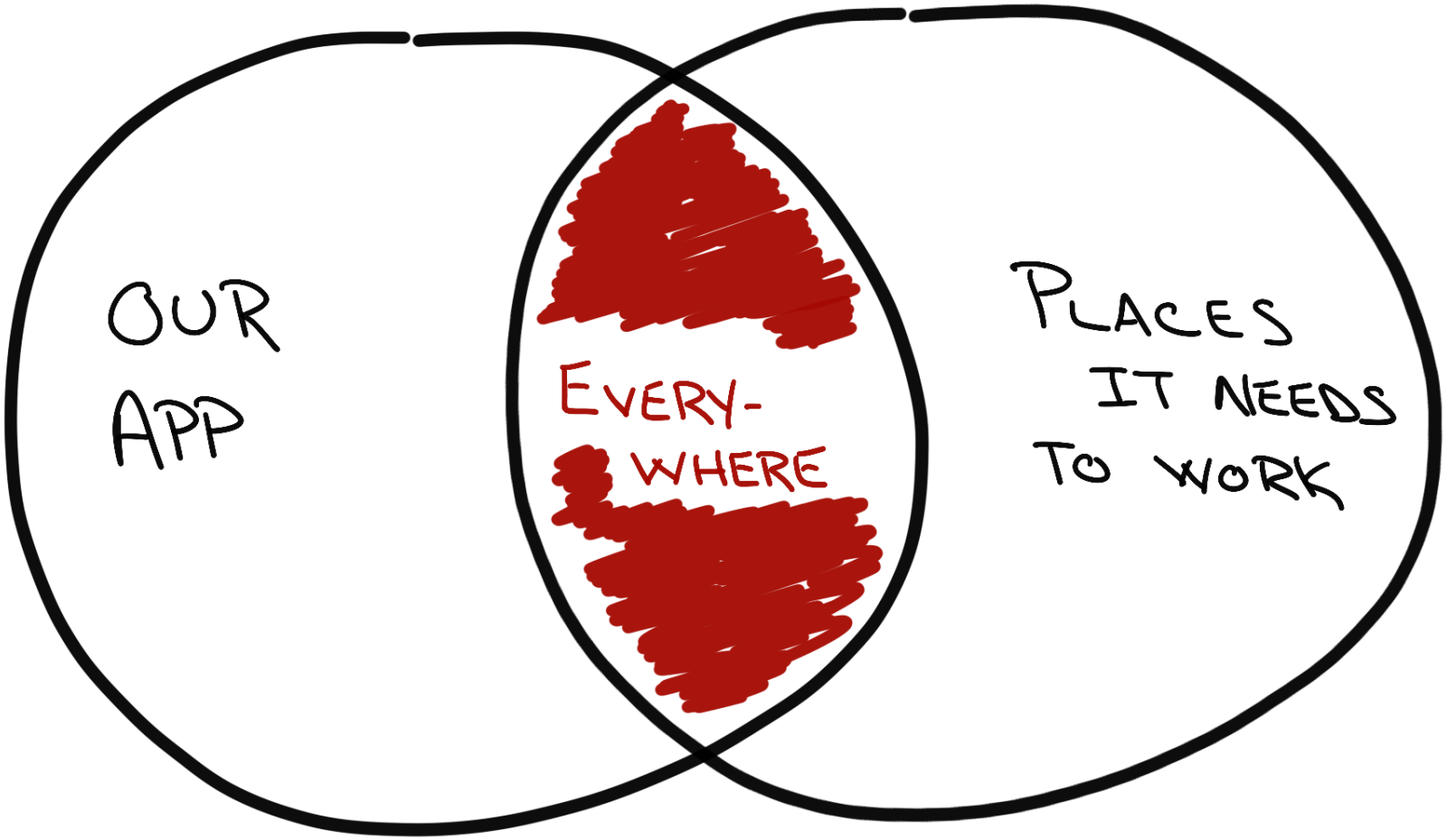
… logic, screens and hardware …



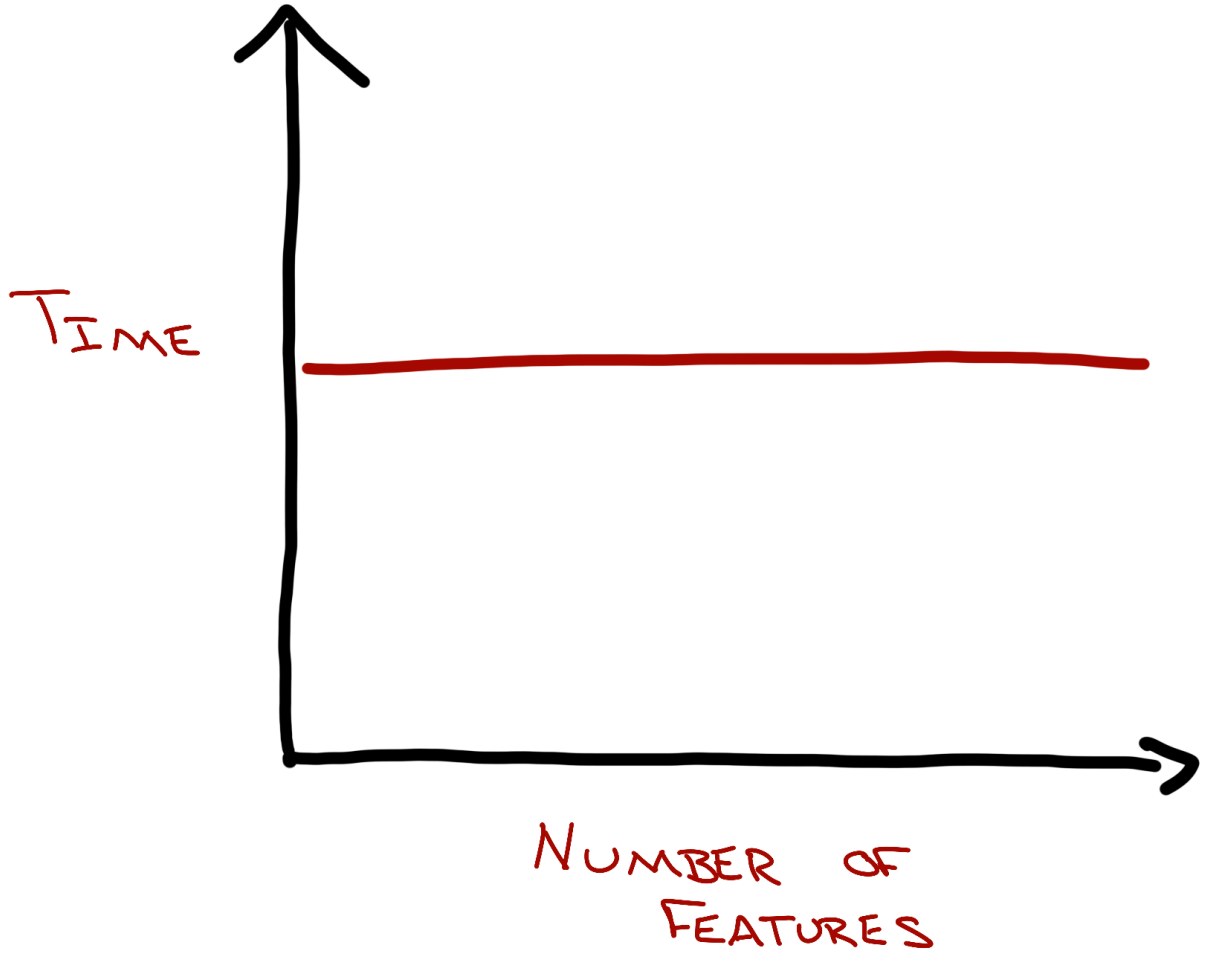
… and more screens and hardware!



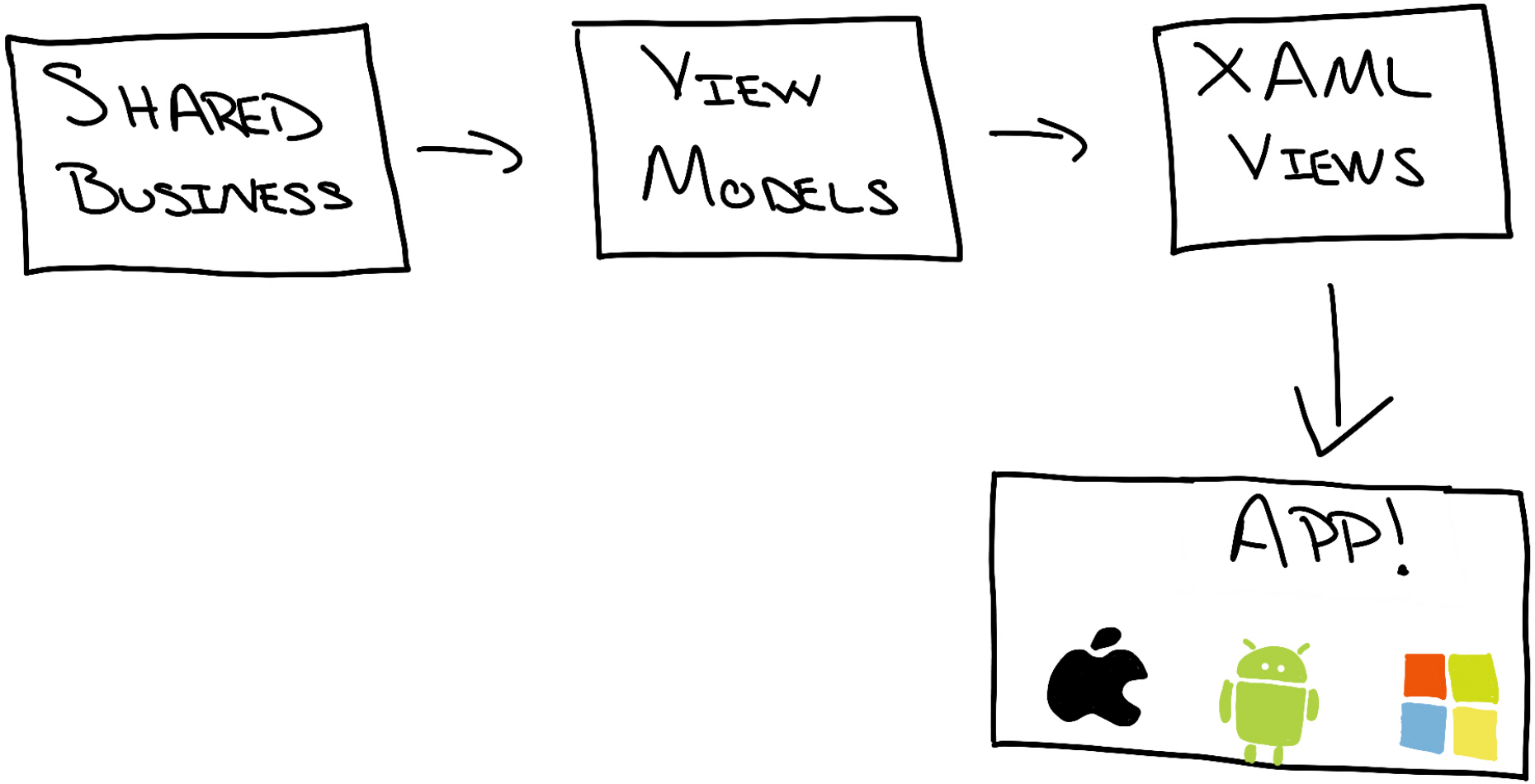
Our app needs to work everywhere



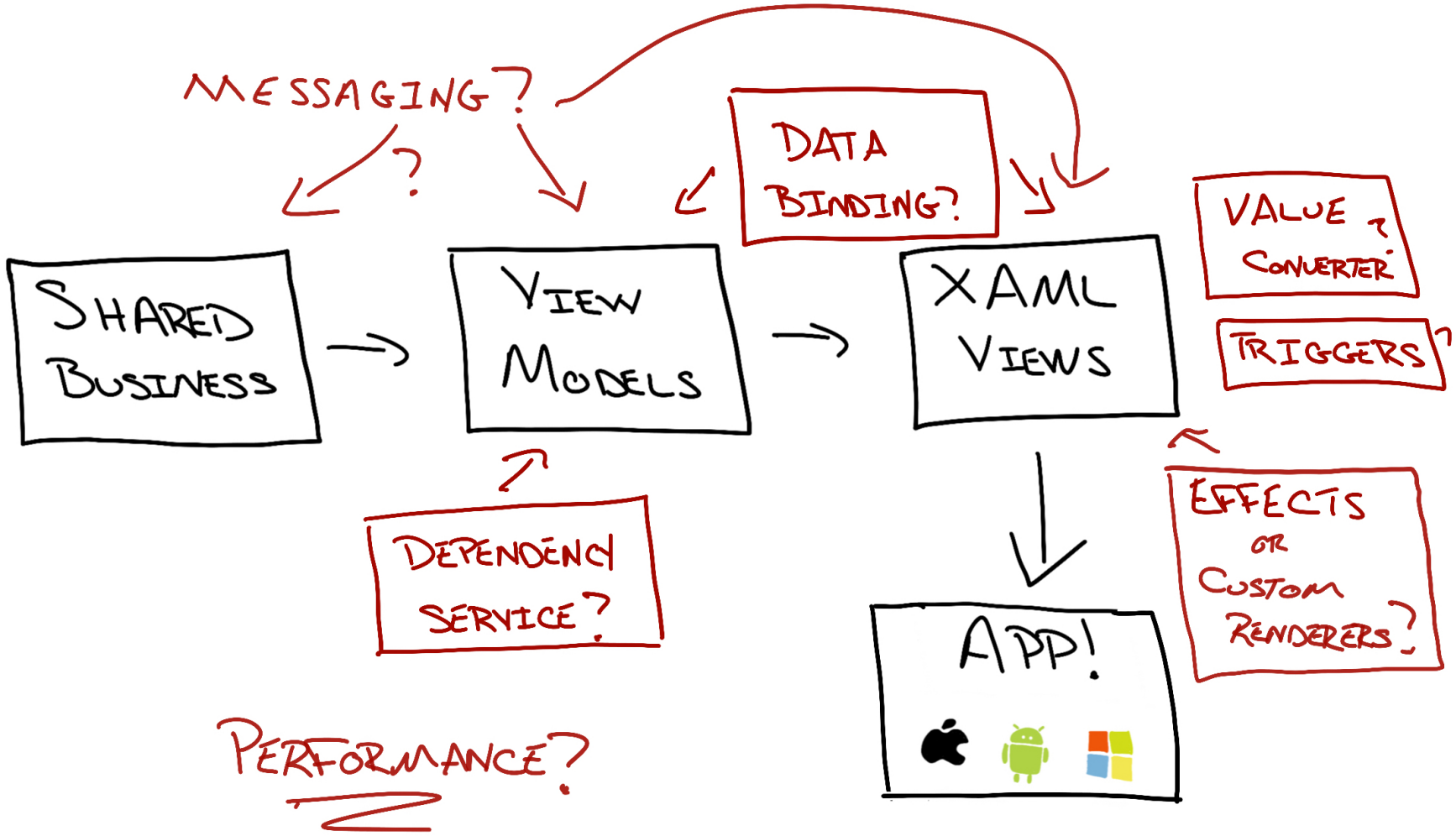
Development time needs to be short



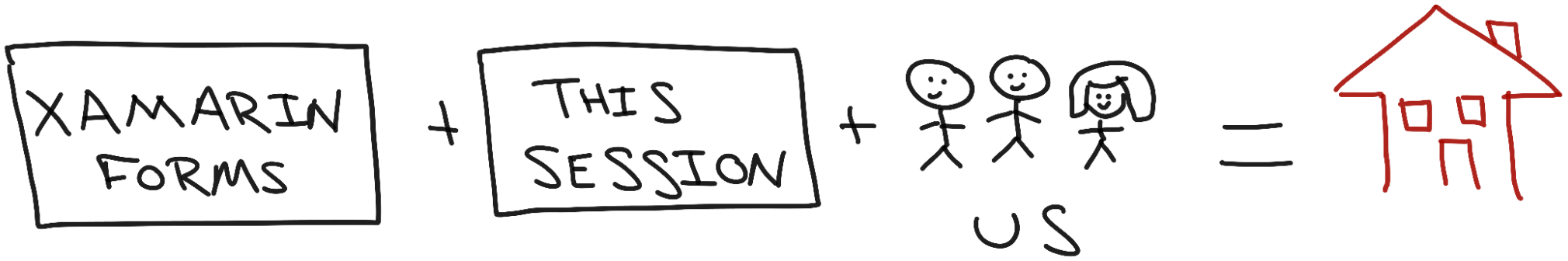
Xamarin.Forms to the rescue!



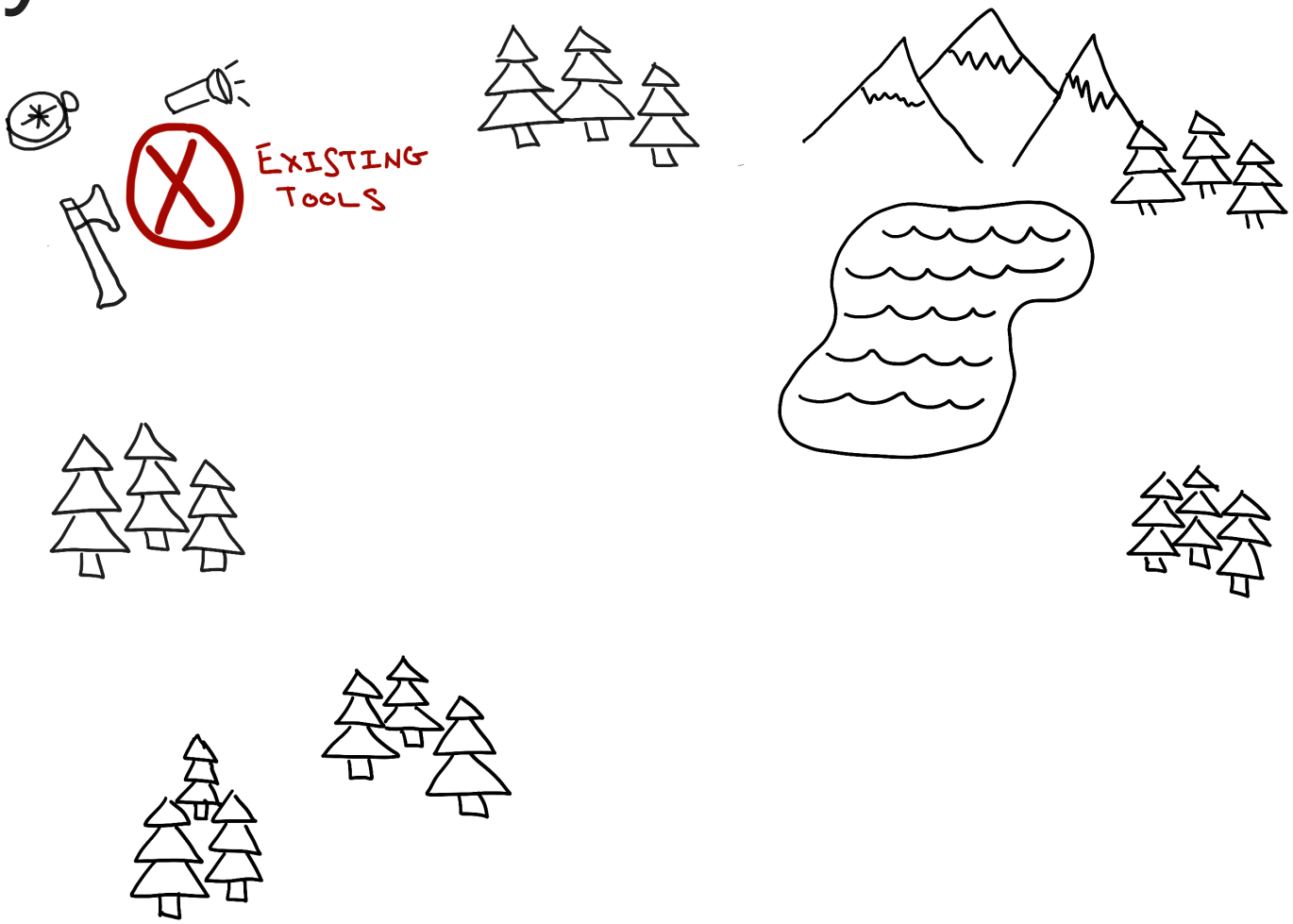
… but it's easy to get lost …



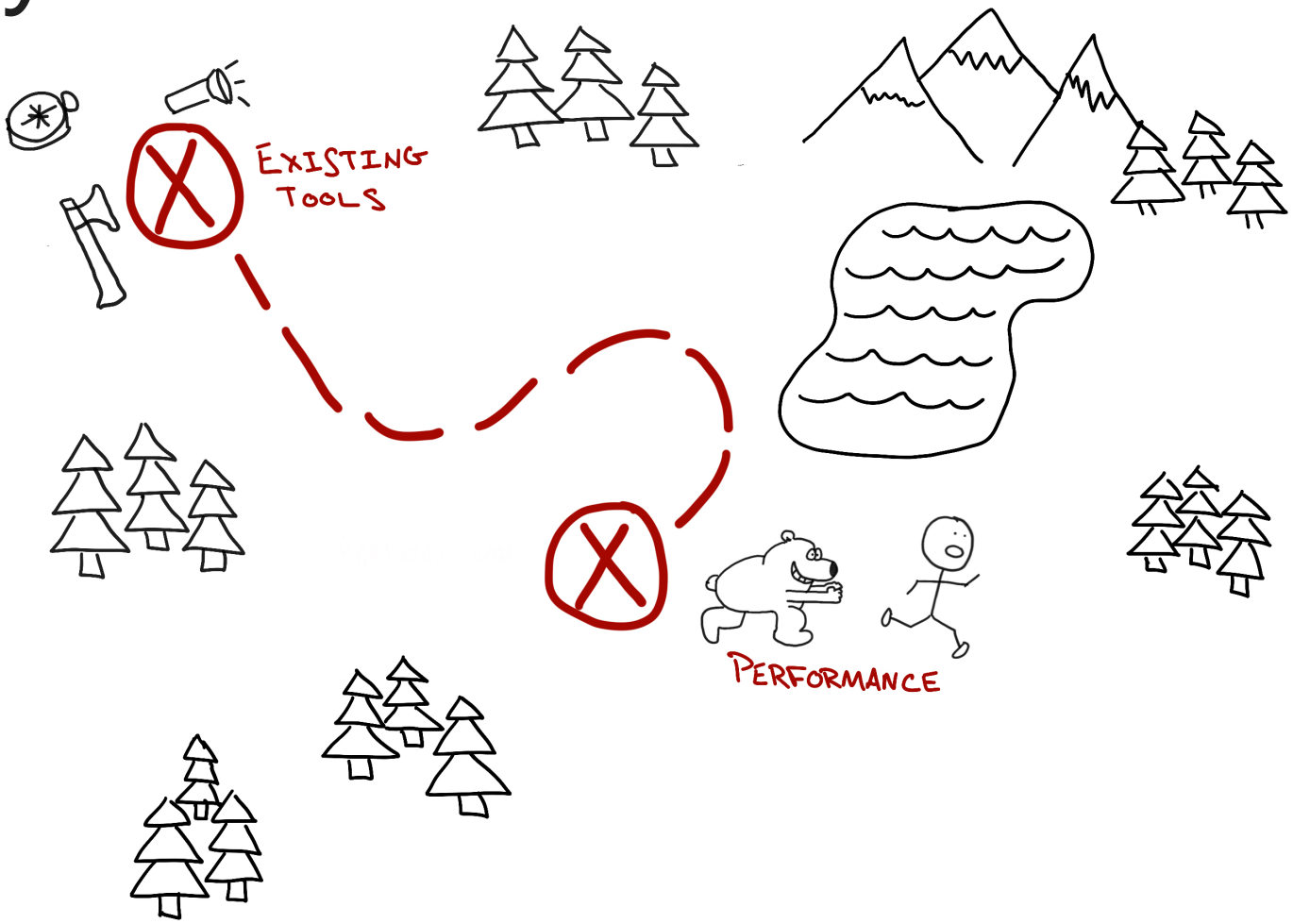
Get out of the woods!



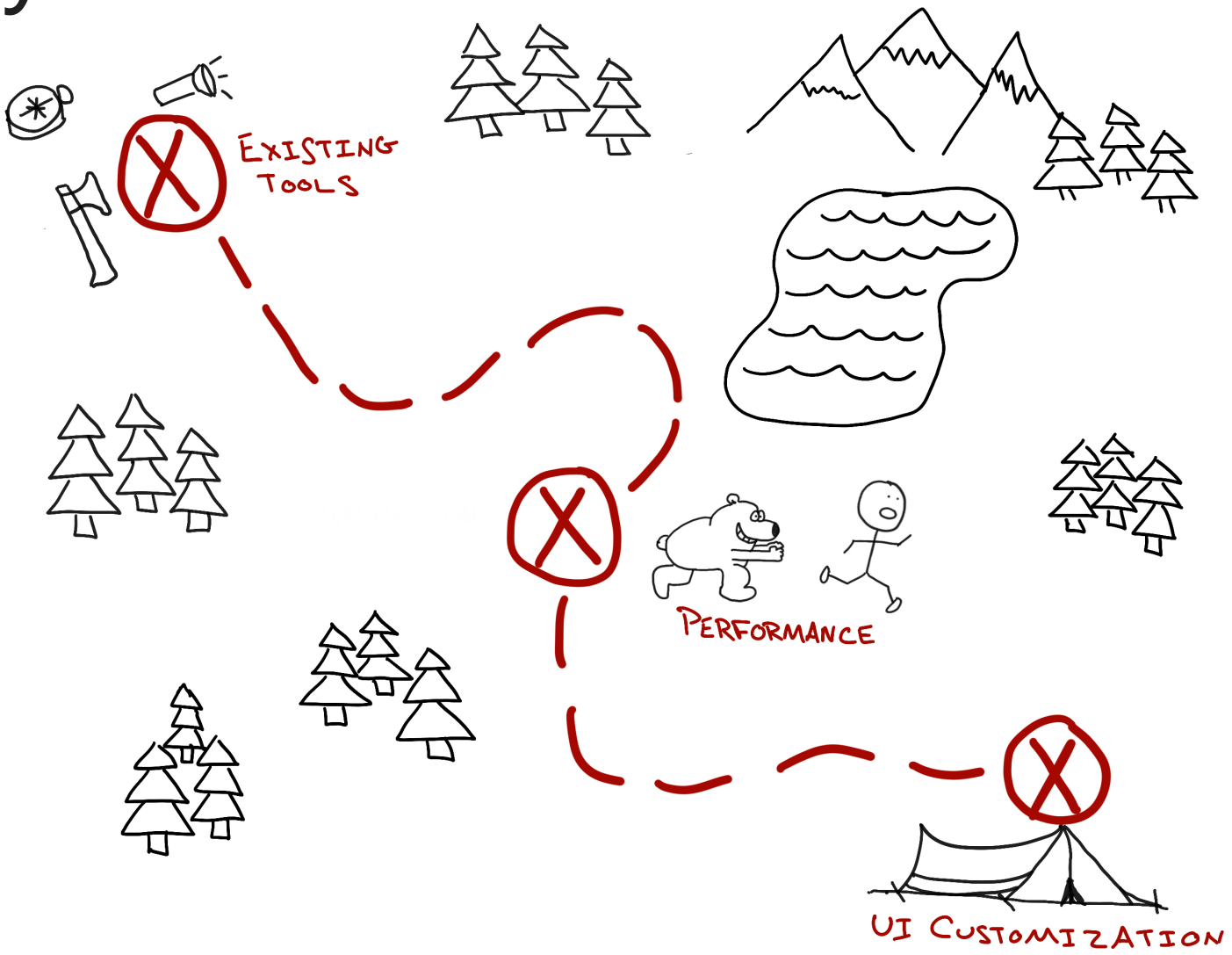
The way out…



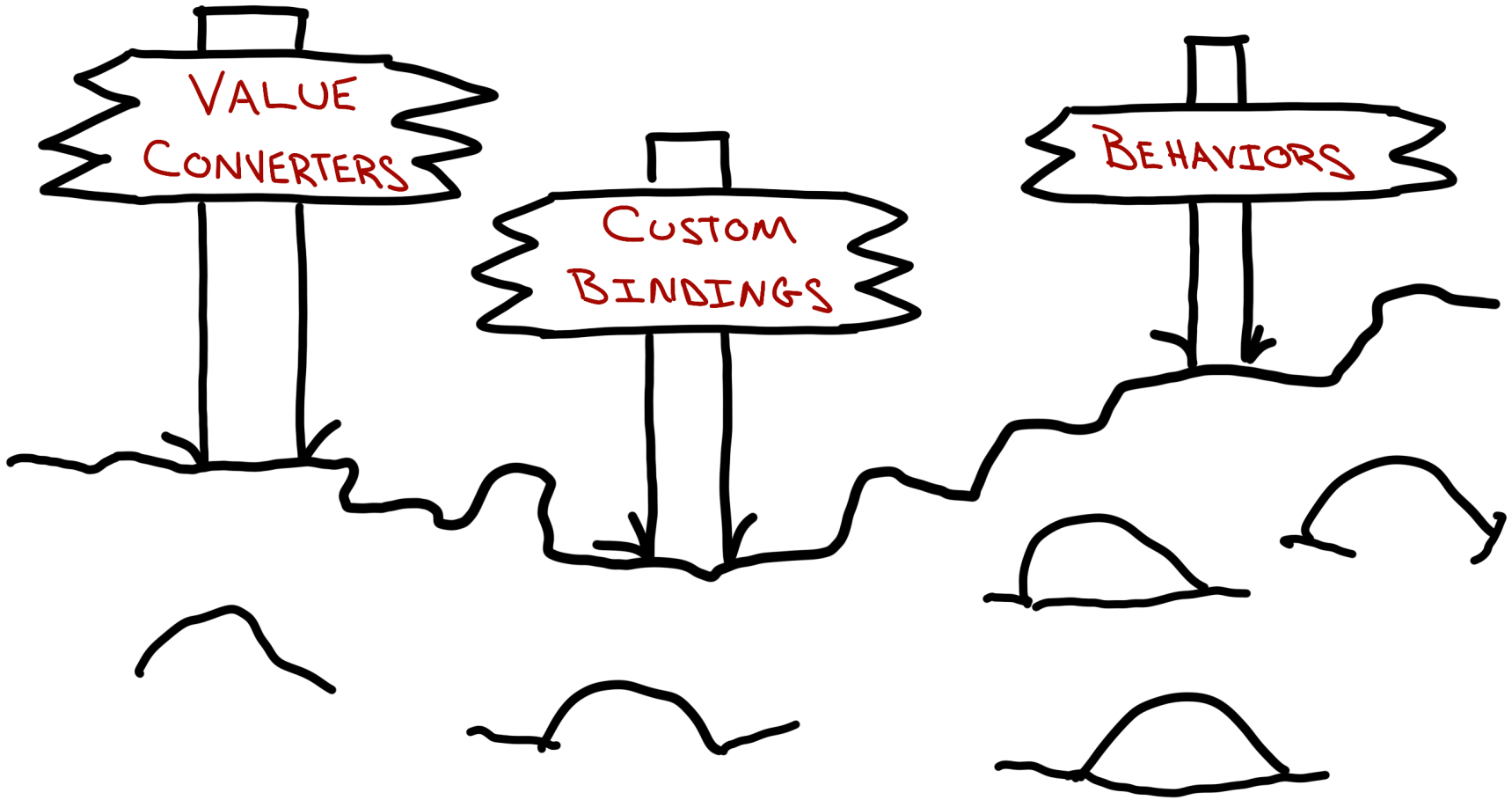
The way out…



The way out…

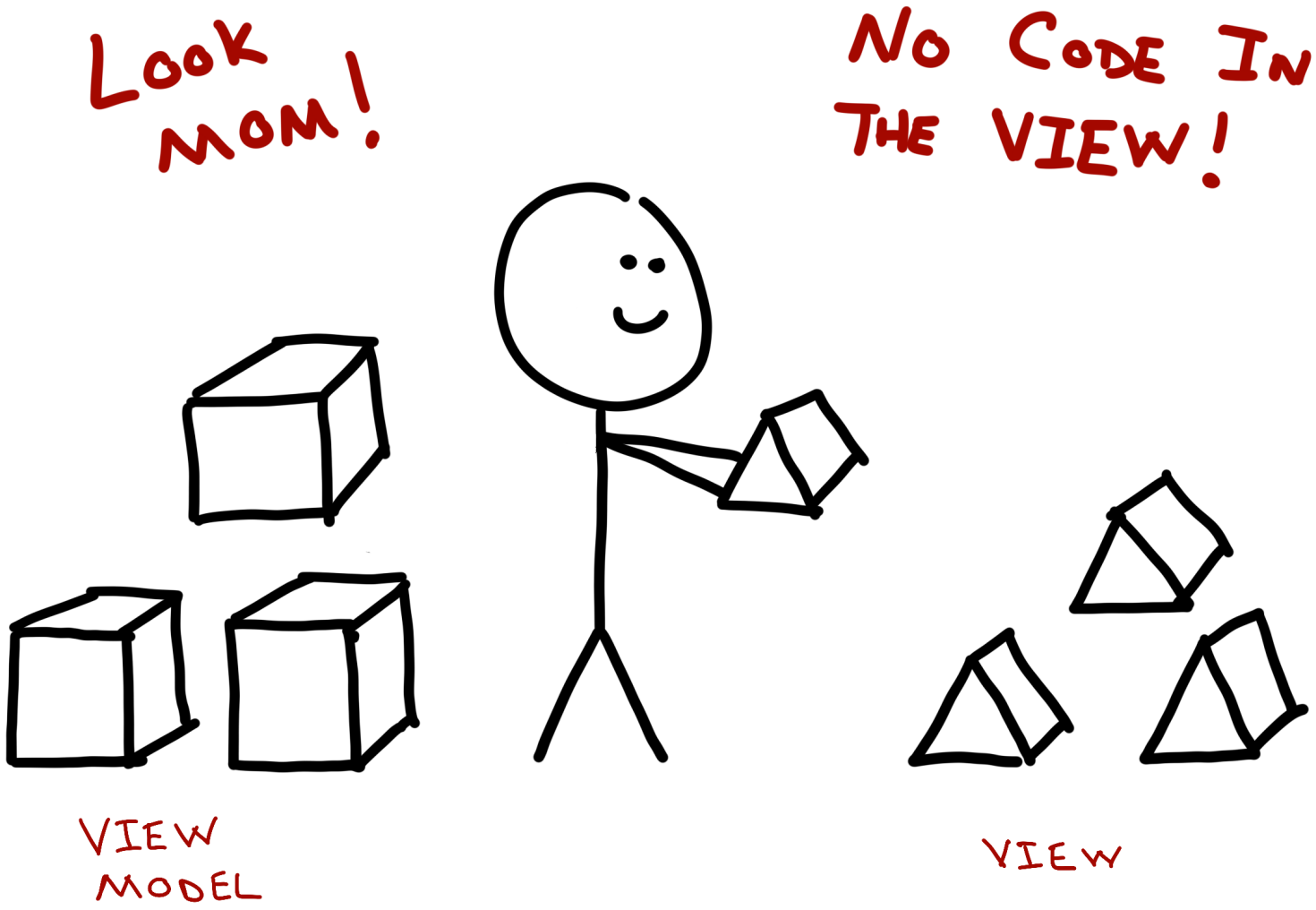


Use what's there!




GOT
MVM
?

Value Converters



```
public interface IValueConverter
{
    object Convert(object value, Type targetType, object parameter,
                   CultureInfo culture);

    object ConvertBack(object value, Type targetType, object parameter,
                       CultureInfo culture);
}
```

- Convert
  - Input value transformed
- ConvertBack
  - Transformed value back
  - Often not implemented

```
public class BoolToColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
                          CultureInfo culture)
    {
        if (value == null || value.GetType() != typeof(bool))
            return Color.Gray;

        if ((bool)value)
            return Color.Green;

        return Color.Gray;

    }

    public object ConvertBack(object value, Type targetType, object parameter,
                              CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Referencing in XAML

Reference converter in page

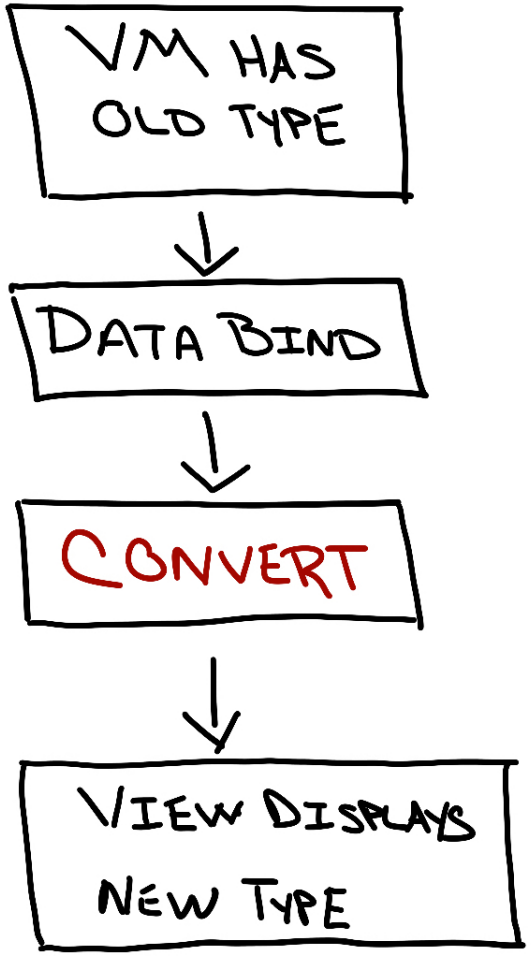
```
<ContentPage.Resources>
  <ResourceDictionary>
    <local:BoolToColorConverter x:Key="colorConverter" />
  </ResourceDictionary>
</ContentPage.Resources>
```

Adding to control

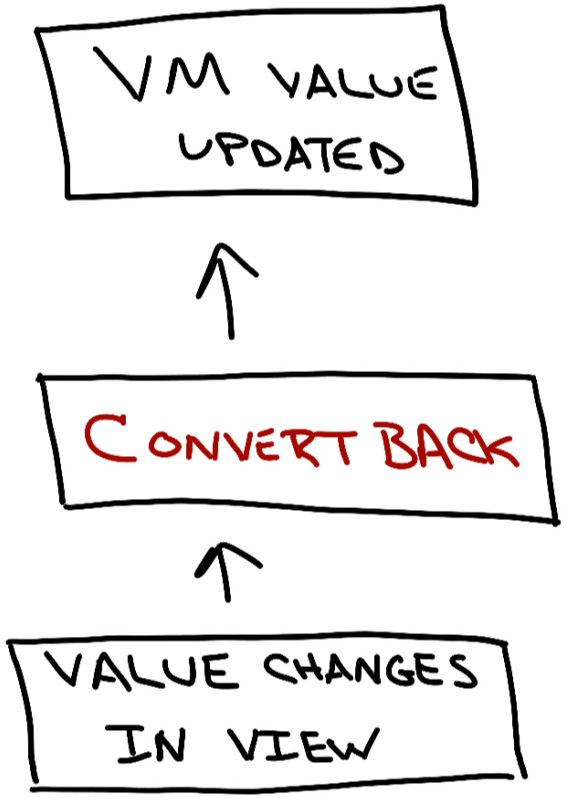
```
<Label Text=”Cool Session" TextColor=
    "{Binding Attended, Converter={StaticResource colorConverter}}" />
```


DEMO

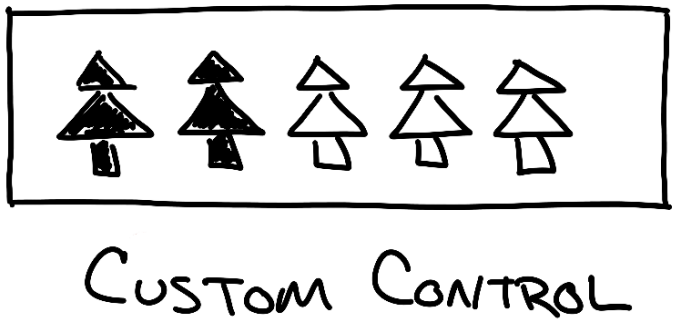
Value Converters - Flow



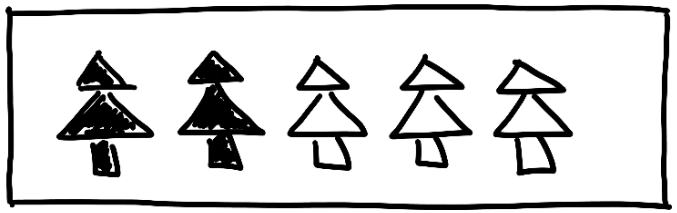
Value Converters - Flow



Custom Bindings



Custom Bindings

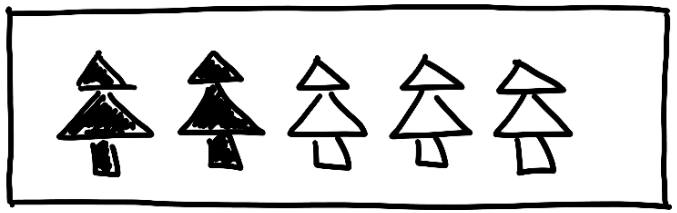


Custom Control

+

Bind # of Highlighted Images to View Model

Custom Bindings



Custom Control

\+

Bind # of Highlighted Images to View Model

=

Custom Bindings!

Custom Bindings

```
public static BindableProperty Create (
    string propertyName,
    Type returnType,
    Type declaringType,
    object defaultValue,
    BindingMode defaultBindingMode = BindingMode.OneWay,
    BindableProperty.ValidateValueDelegate validateValue = null,
    BindableProperty.BindingPropertyChangedDelegate propertyChanged = null,
    BindableProperty.BindingPropertyChangingDelegate propertyChanging = null,
    BindableProperty.CoerceValueDelegate coerceValue = null,
    BindableProperty.CreateDefaultValueDelegate defaultValueCreator = null);
```

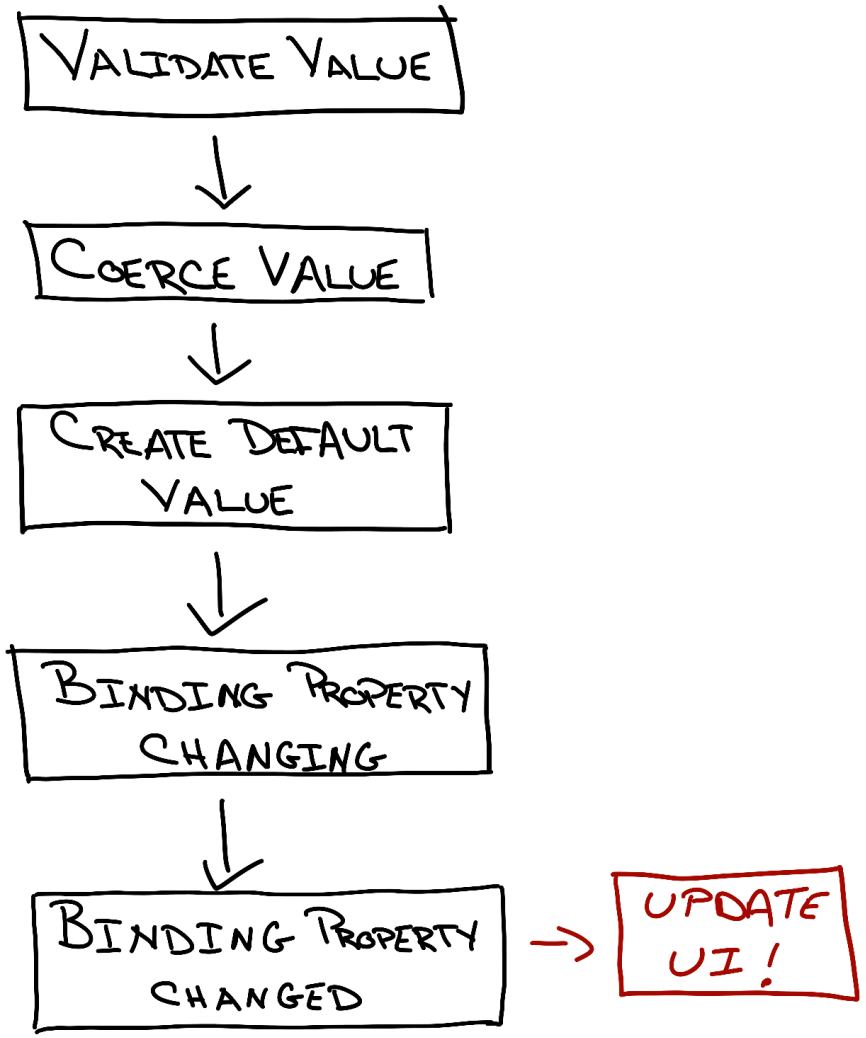
Creating a Binding

```
public class TreeRating : ContentView
{
    public int NumOfTrees
    {
        get { return (int)GetValue(NumOfTreesProperty); }
        set { SetValue(NumOfTreesProperty, value); }
    }

    public static readonly BindableProperty NumOfTreesProperty =
            BindableProperty.Create(
                nameof(NumOfTrees),
                typeof(int),
                typeof(TreeRating),
                1,
                BindingMode.TwoWay);
}
```

Other Parameters (all delegates)

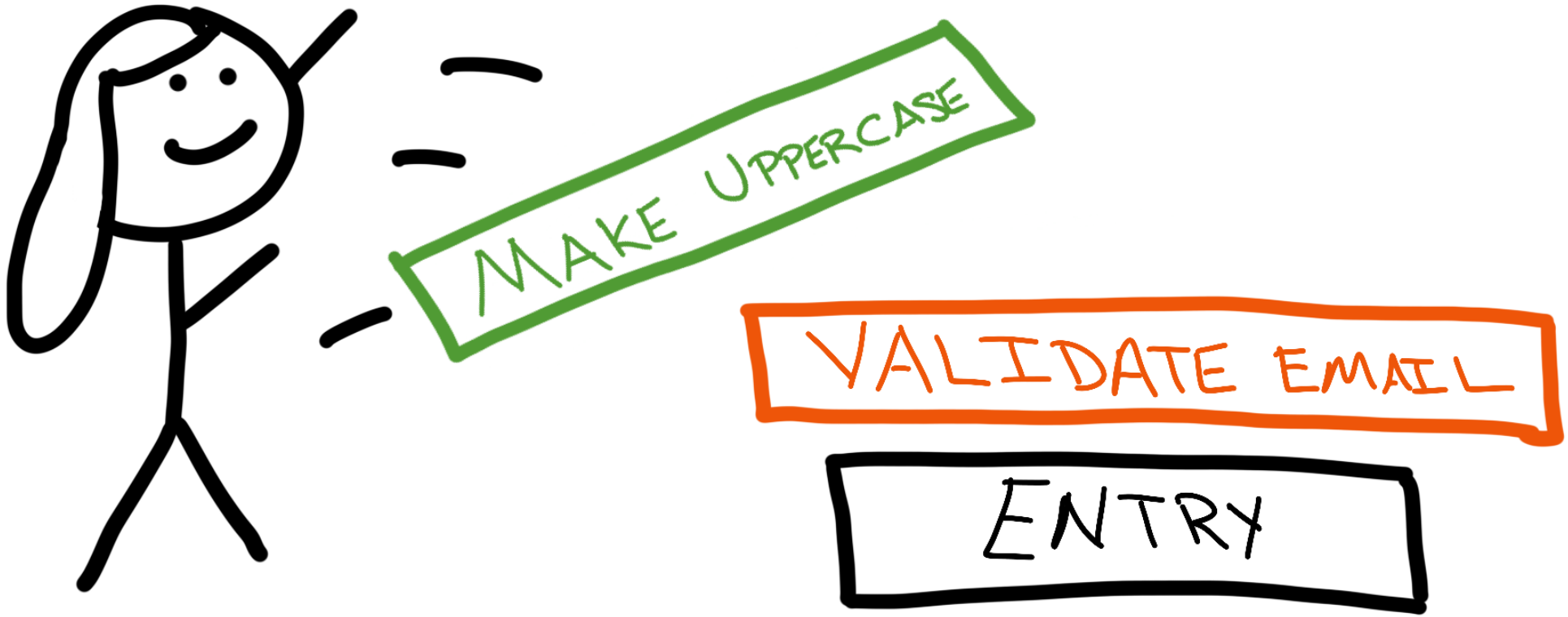
- validateValue
  - Performs validation on new value
- propertyChanged
  - Good place to update the UI
- propertyChanging
  - Fired before property changes
- coerceValue
  - Change value before it gets set
- defaultValueCreator
  - Function to create default value

Custom Bindings - Flow




DEMO

Behaviors



```
public abstract class Behavior<T> : Behavior where T : BindableObject
{
    protected virtual void OnAttachedTo(T bindable);

    protected virtual void OnDetachingFrom(T bindable);
}
```

Behavior<T>

- OnAttachedTo
- OnDetachedFrom

```
protected override void OnAttachedTo(Entry bindable)
{
    bindable.TextChanged += HandleTextChanged;

    base.OnAttachedTo(bindable);
}
```

OnAttachedTo

- Setup overall behavior
- Handle event that triggers behavior

```
protected override void OnDetachingFrom(Entry bindable)
{
    bindable.TextChanged -= HandleTextChanged;

    base.OnDetachingFrom(bindable);
}
```

OnDetachingFrom

- Release memory and cleanup

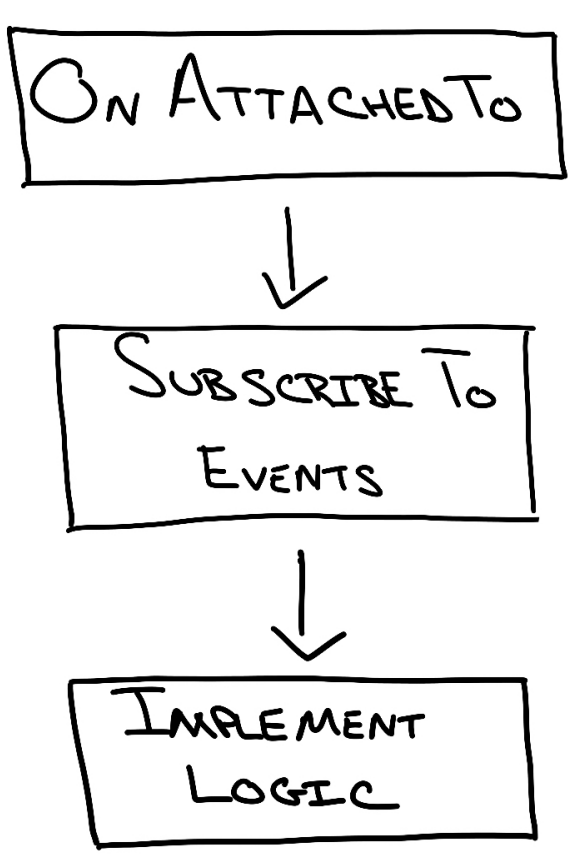
```
public class VisualElement : Element, ...
{
    public IList<Behavior> Behaviors
    {
        get;
    }
    ...
}
```

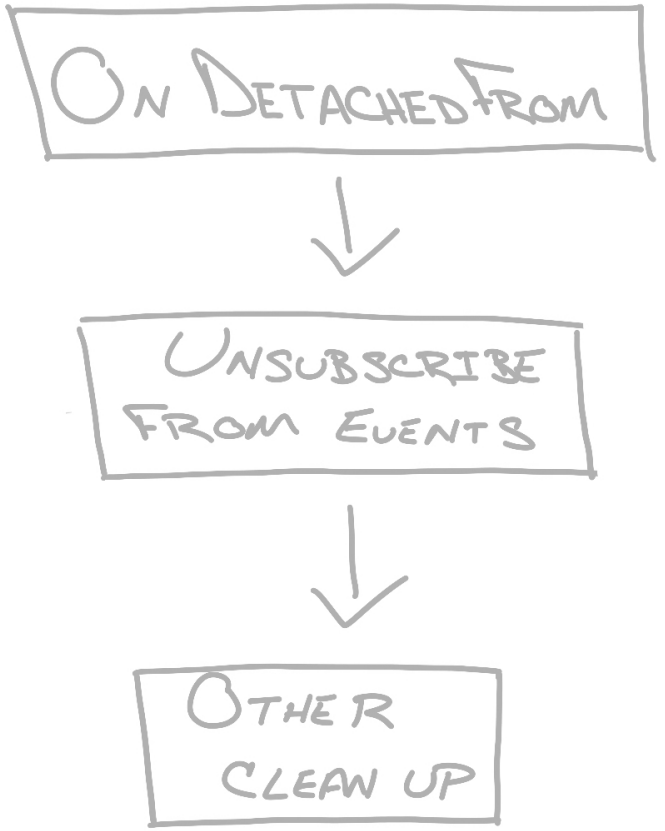
Adding a Behavior to a Control

- Every visual element has a Behaviors list

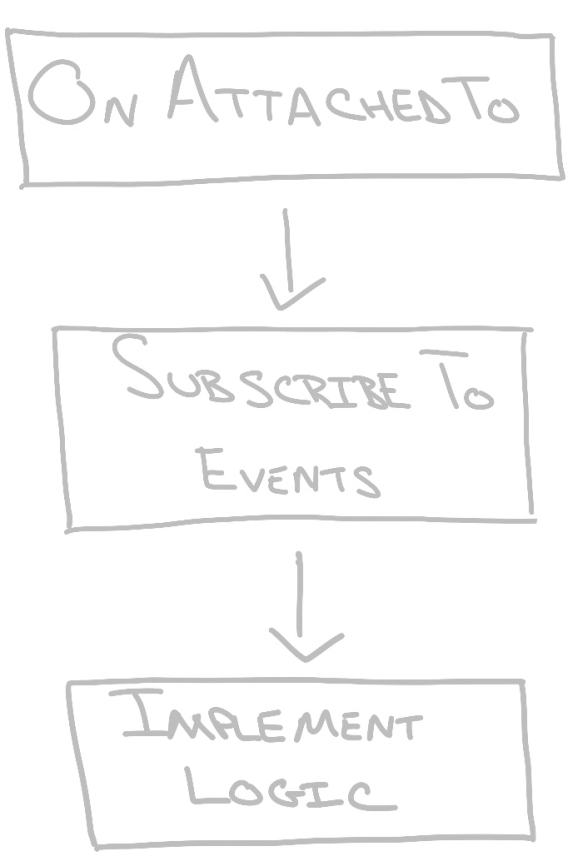

DEMO

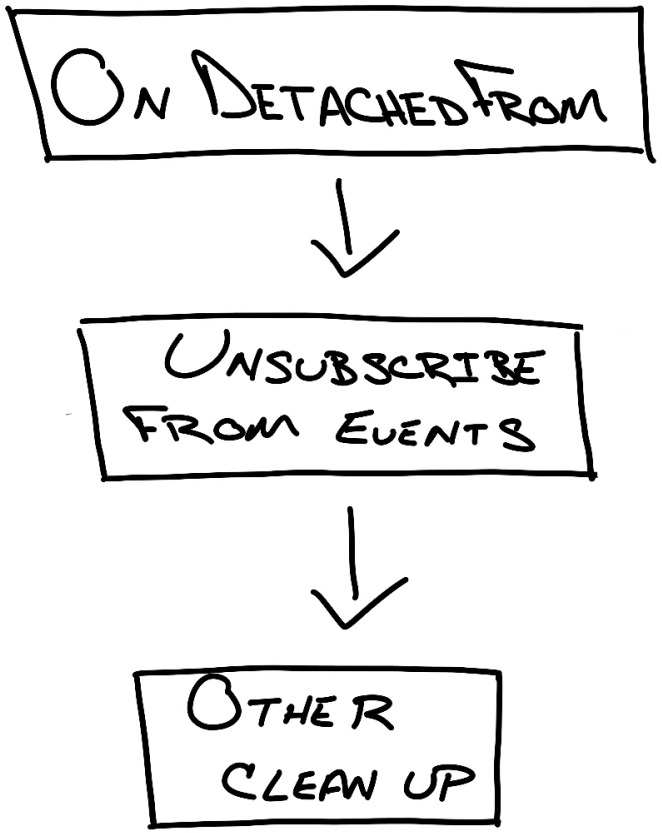
Behaviors - Flow





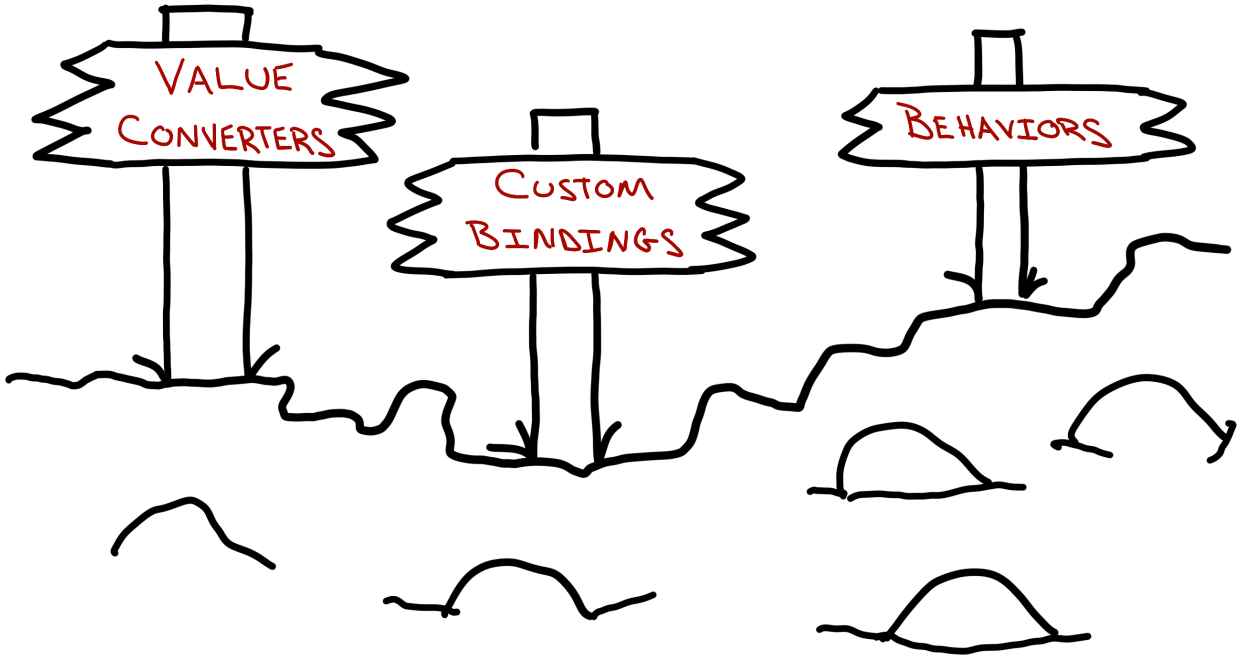
Behaviors - Flow



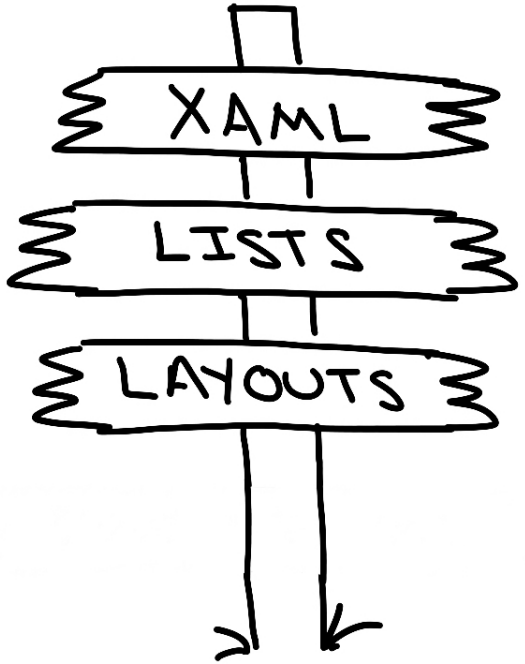


Clean Up the View - Use what's there!

- Value converters
- Custom bindings
- Behaviors



Performance



PERFORMANCE

XAML Compilation

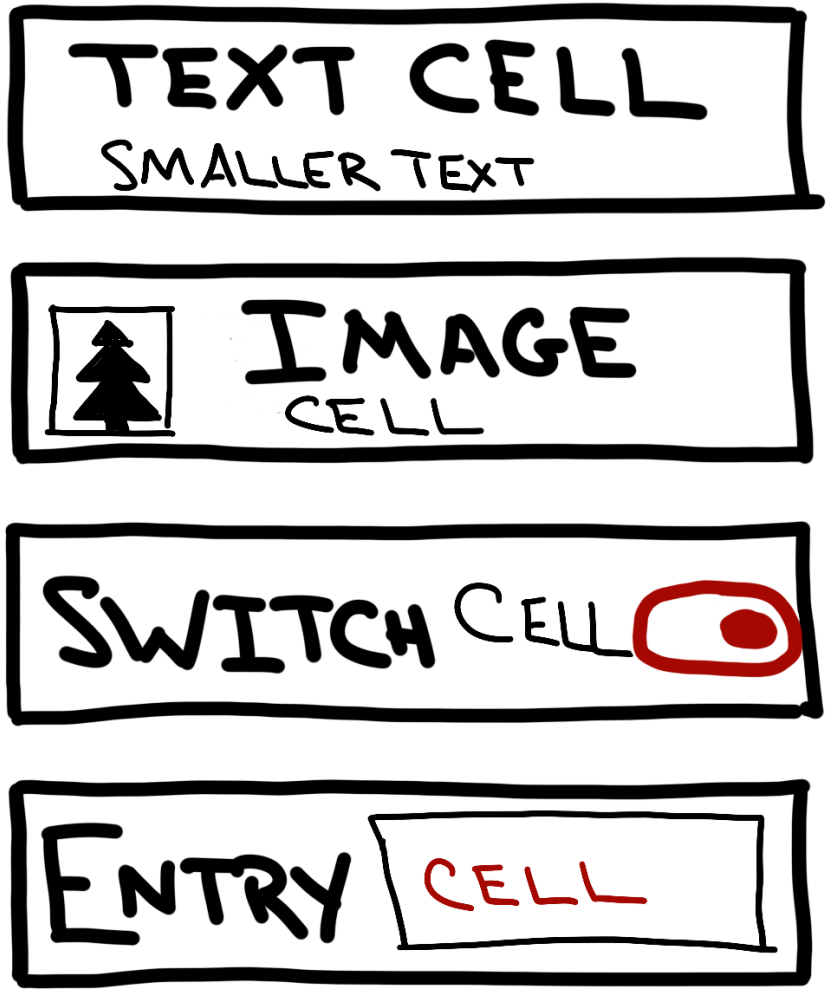
XAML Compilation

```
using Xamarin.Forms.Xaml;

[assembly: XamlCompilation(XamlCompilationOptions.Compile)]
namespace FormsTalk
{
   …
```

```
[XamlCompilation(XamlCompilationOptions.Compile)]
public class Woods
{
   …
```

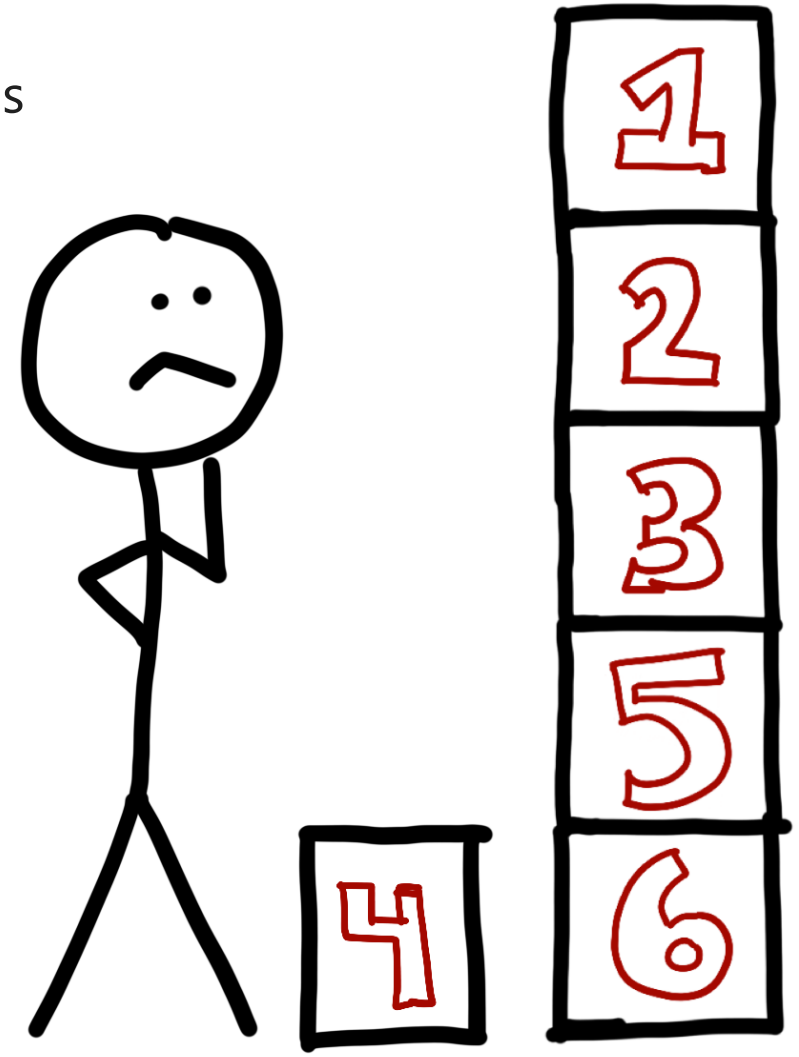
List Performance

Use Built-In Cells

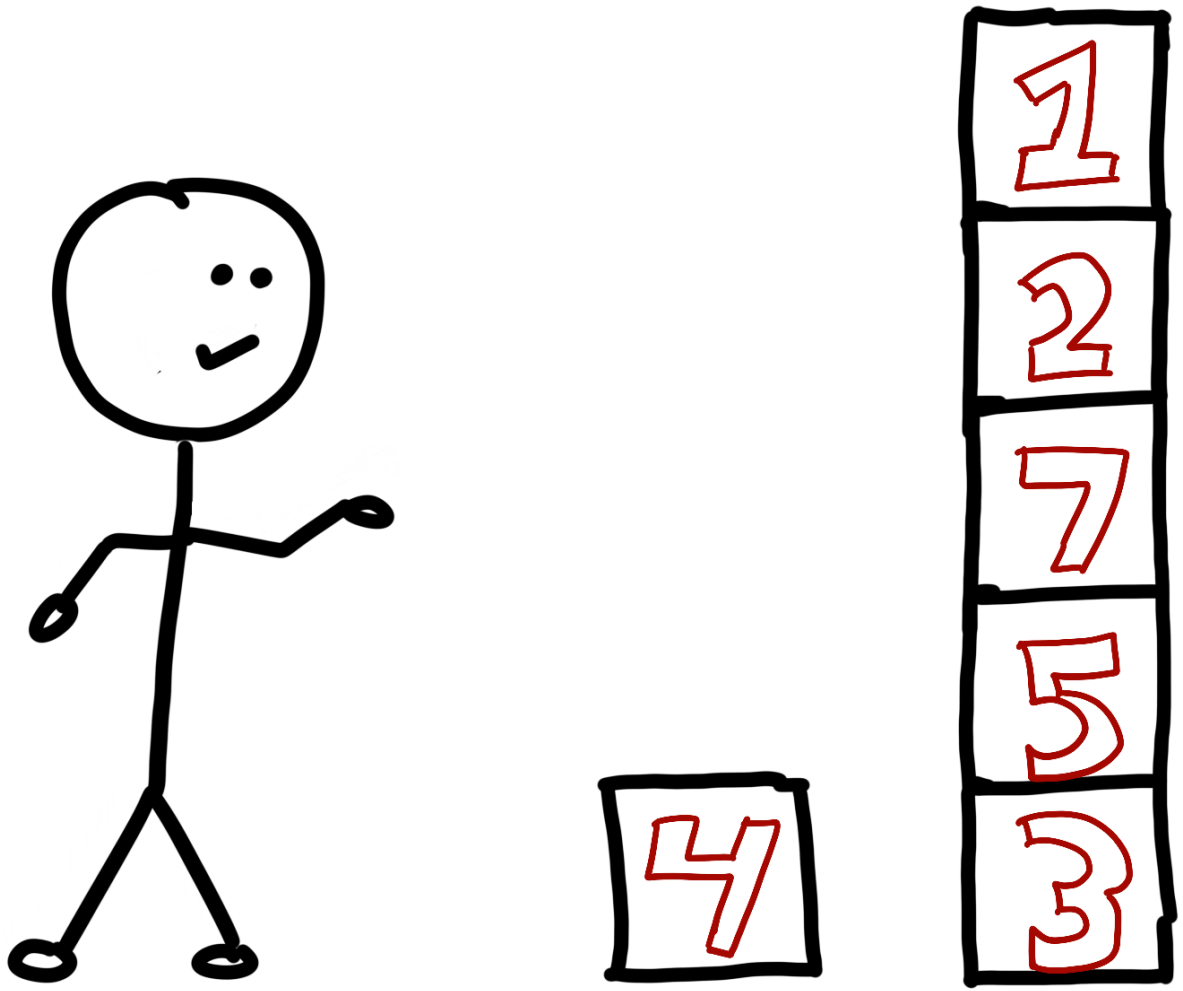


Don’t bind to IEnumerable<T>

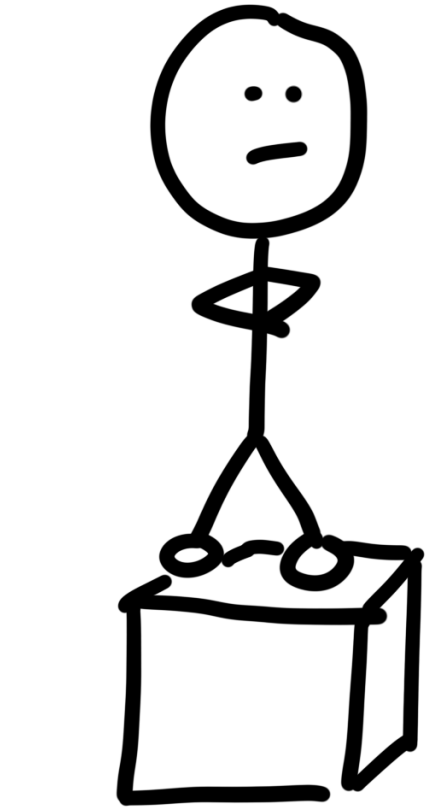
It doesn’t have random access



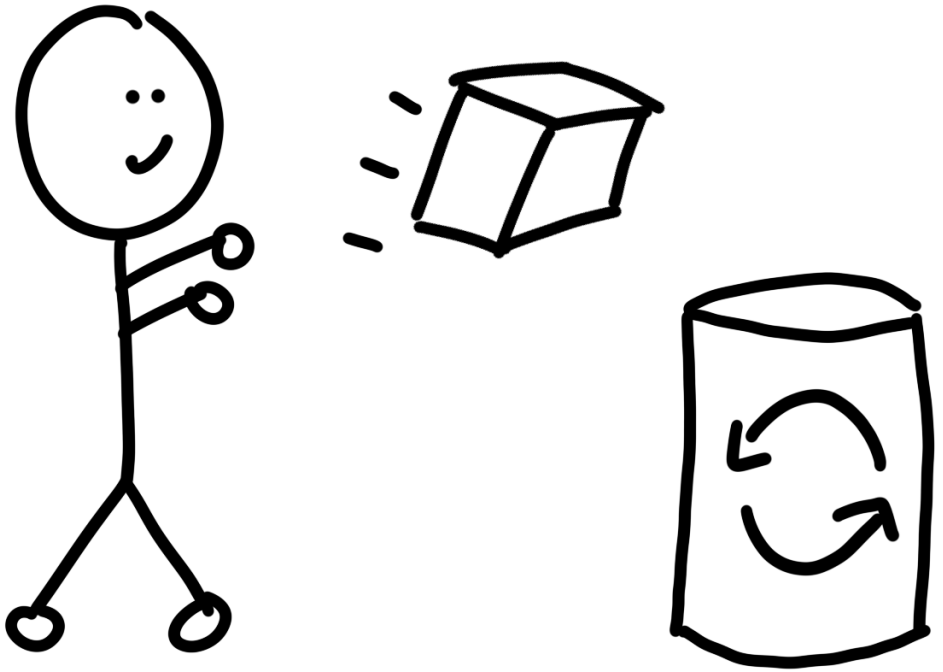
Bind to IList<T>



Caching Strategy



RETAIN



RECYCLE

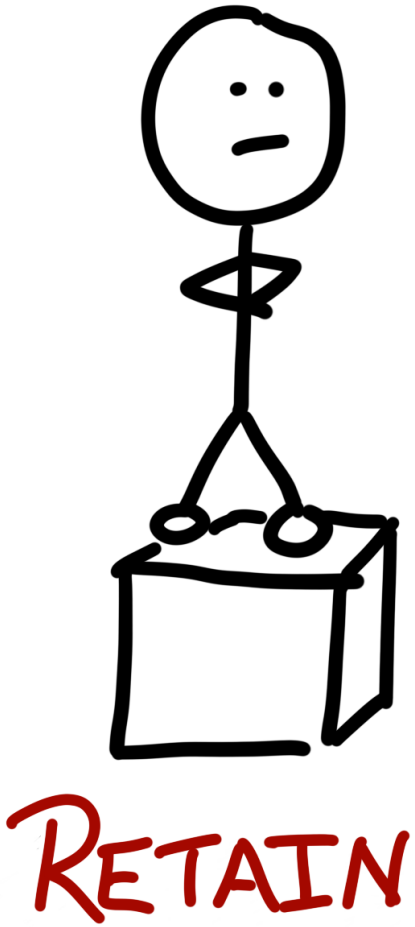
RetainElement

Use when:

- Cell has large number of bindings
- Cell template changes a lot

Remember:

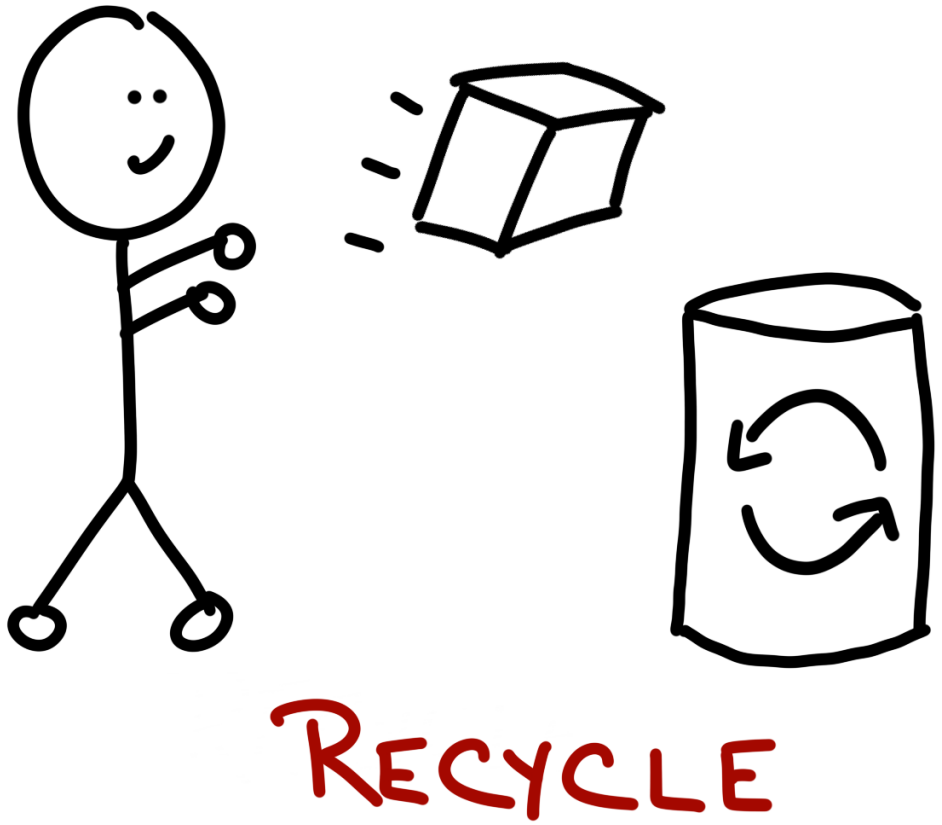
- Any cell initialization code will run for each cell!
- Default behavior



RecycleElement

Use when:

- Cell has small number of bindings
- Cell's binding context defines ALL of the data
- All cell's look the same

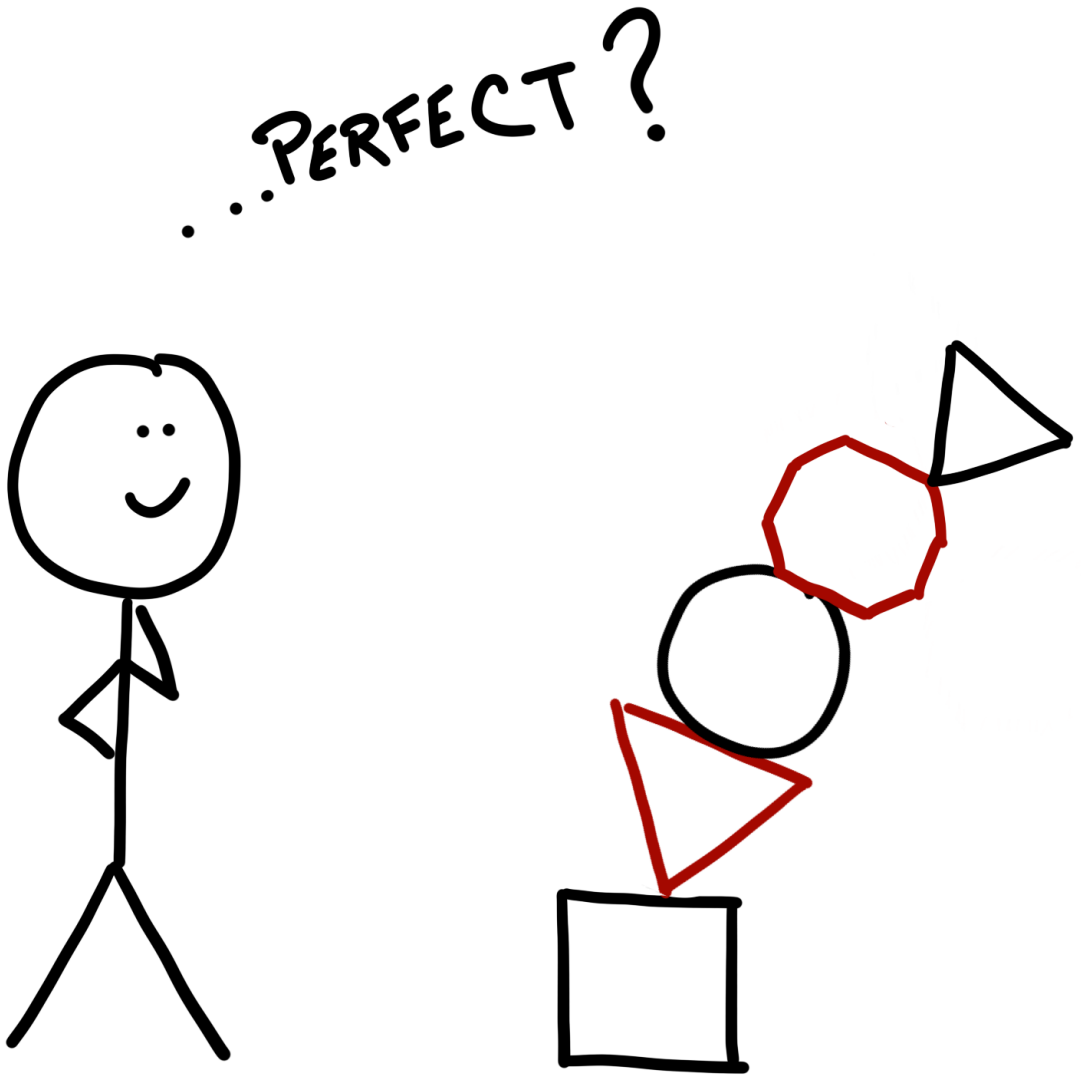


Setting Caching Strategy

```
<ListView CachingStrategy="RecycleElement">
    ...
</ListView>
```

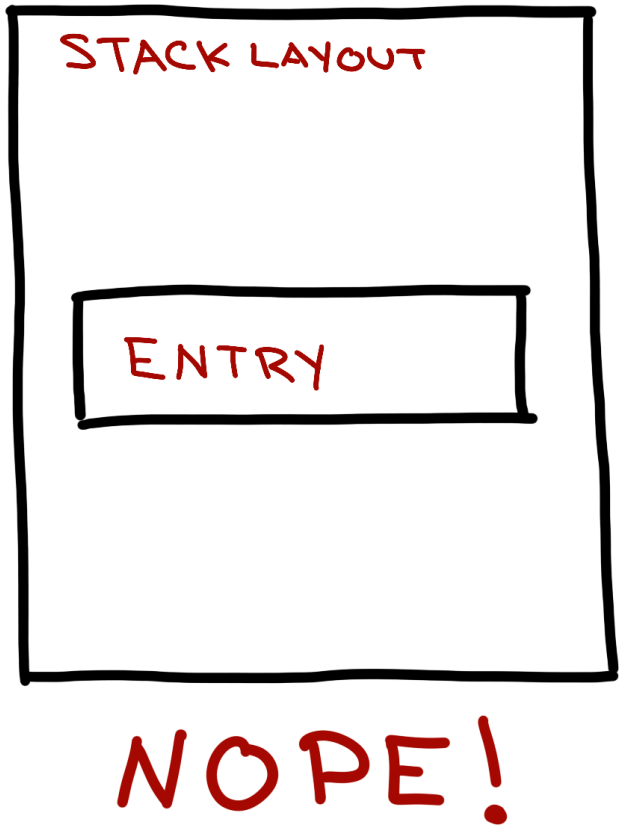
```
var listView = new ListView(ListViewCachingStrategy.RecycleElement);
```

Layout Performance



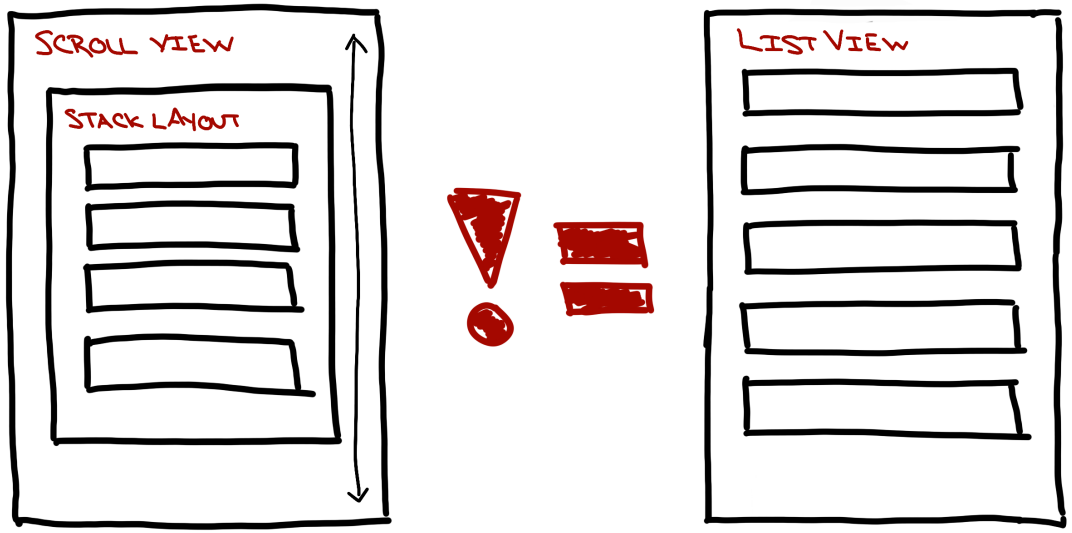
Stack Layout Performance

- Make sure to have more than one element

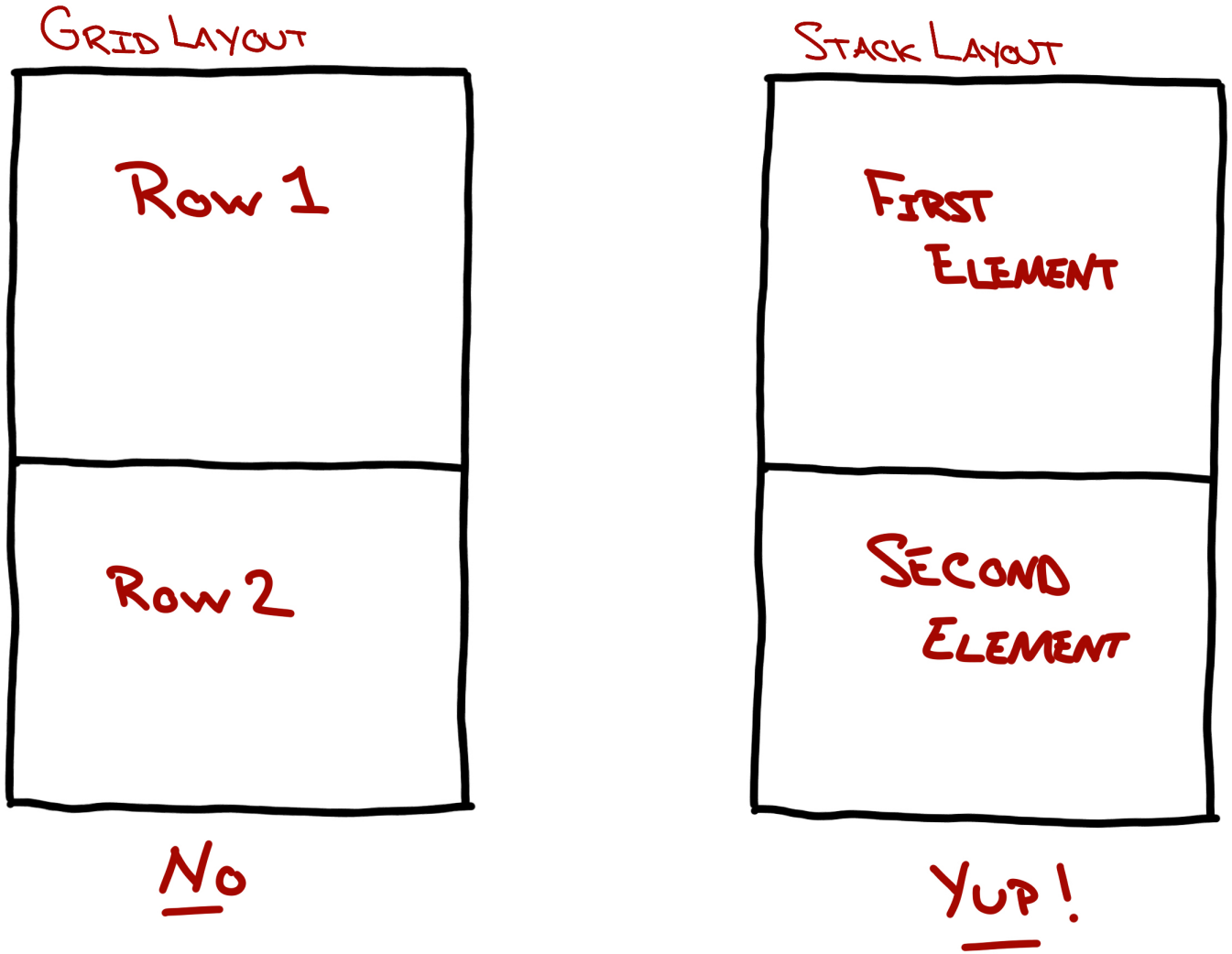


Stack Layout Performance

- Use list views instead of stack layouts inside scroll views
- (And no list views inside of scroll views)

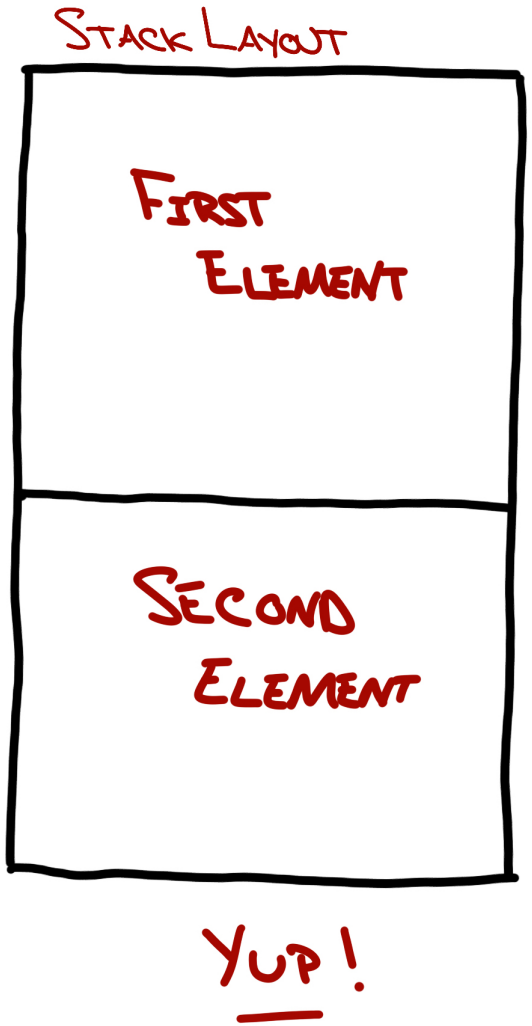


Stack Layout vs. Grid



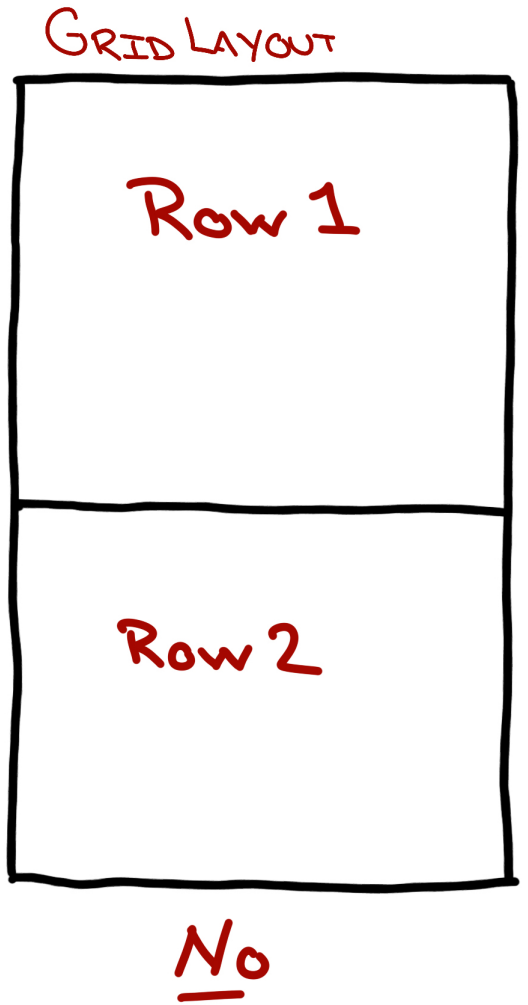
Stack Layout vs. Grid

Don’t reproduce an easy stack in a grid

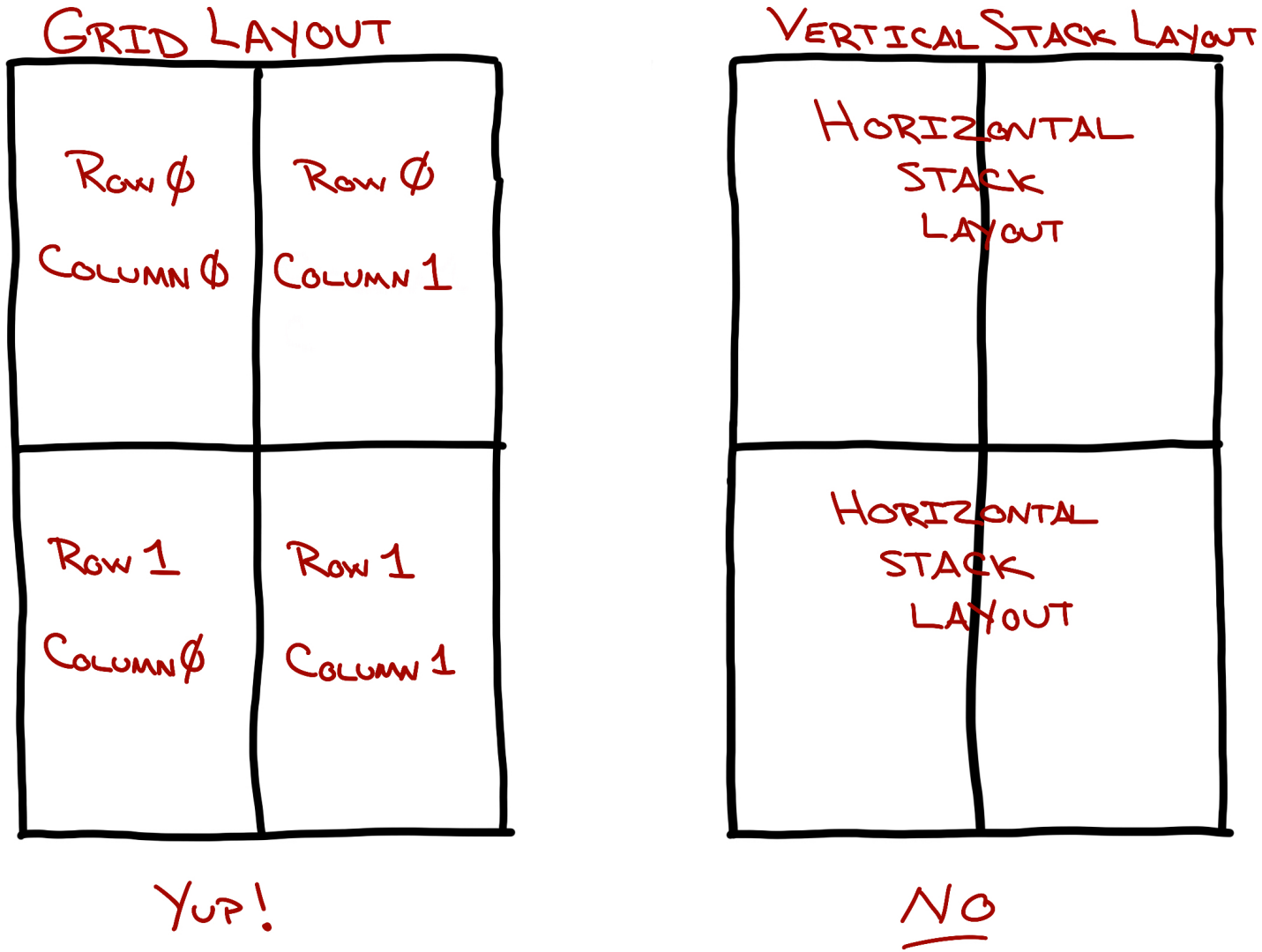


Stack Layout vs. Grid

Don’t reproduce an easy stack in a grid

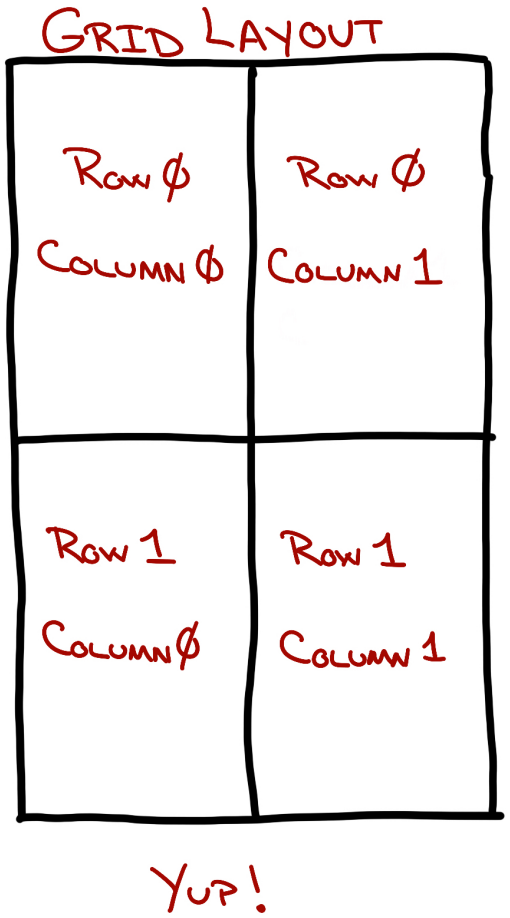


Stack Layout vs. Grid



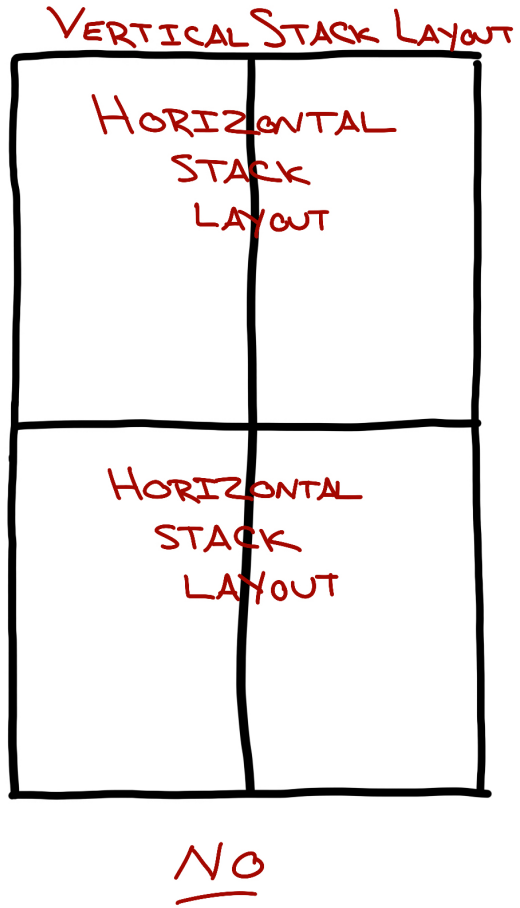
Stack Layout vs. Grid

Don’t reproduce an easy grid in a stack



Stack Layout vs. Grid

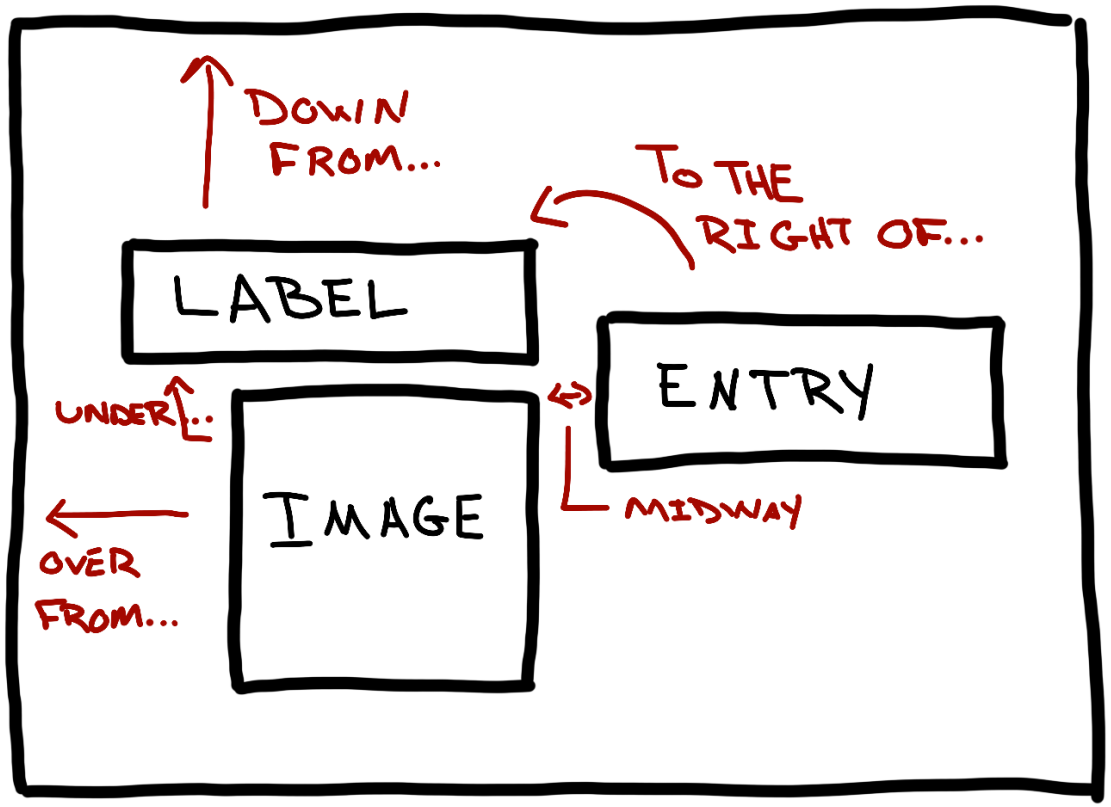
Don’t reproduce an easy stack in a grid



Grids

- Use for layering
- Be aware of ColumnSpacing and RowSpacing as opposed to padding
- Prefer * sized columns and rows instead of auto sized

Relative Layout – Don’t!



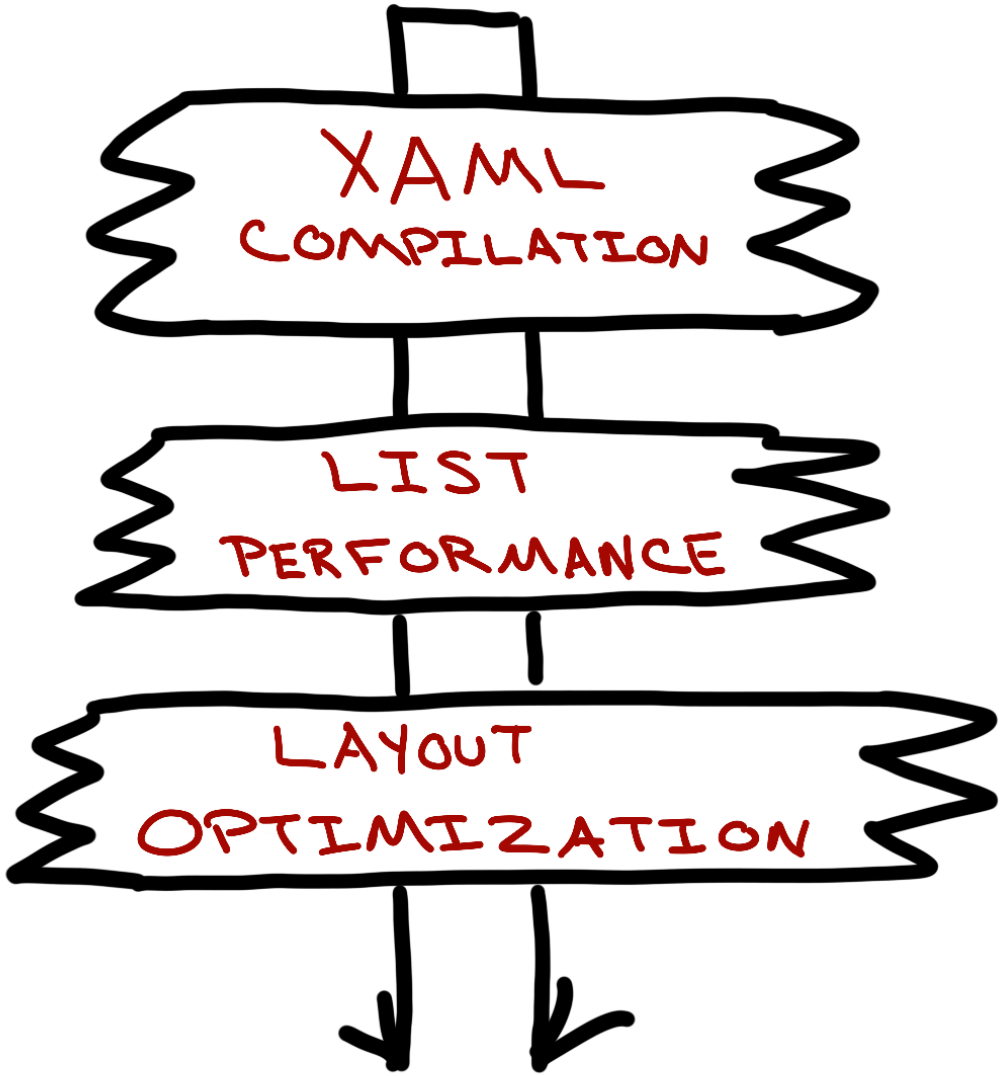
DON’T USE

More Performance…

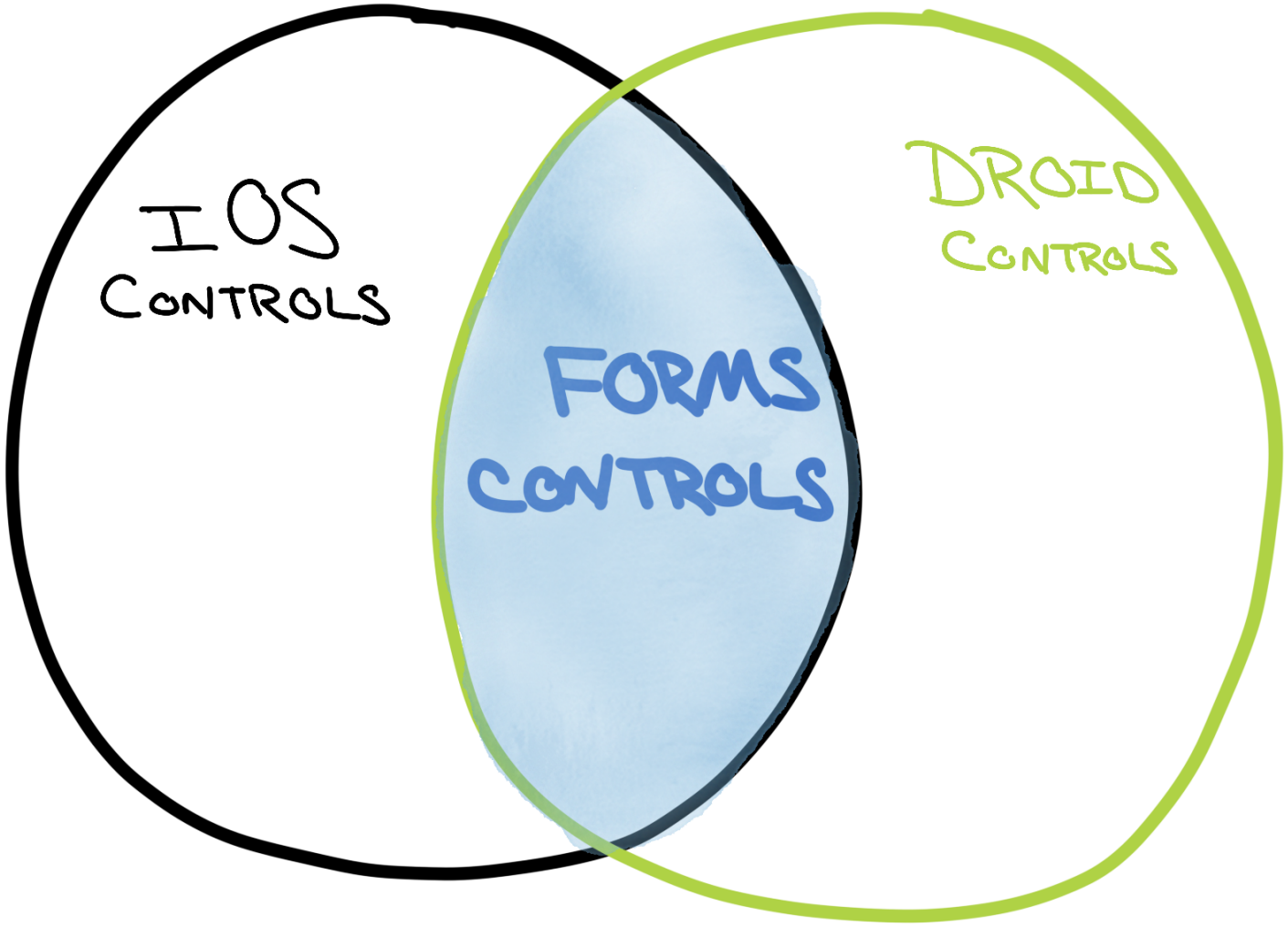
- Use margin over padding
- LayoutOptions.Fill default and doesn’t need to change
- Don’t use CarouselPage
  - CarouselView within ContentPage
- Don’t use MessagingCenter
- Try to use a single label (text sizing hard!)
- Avoid transparency
- Use Android AppCompat

Performance!

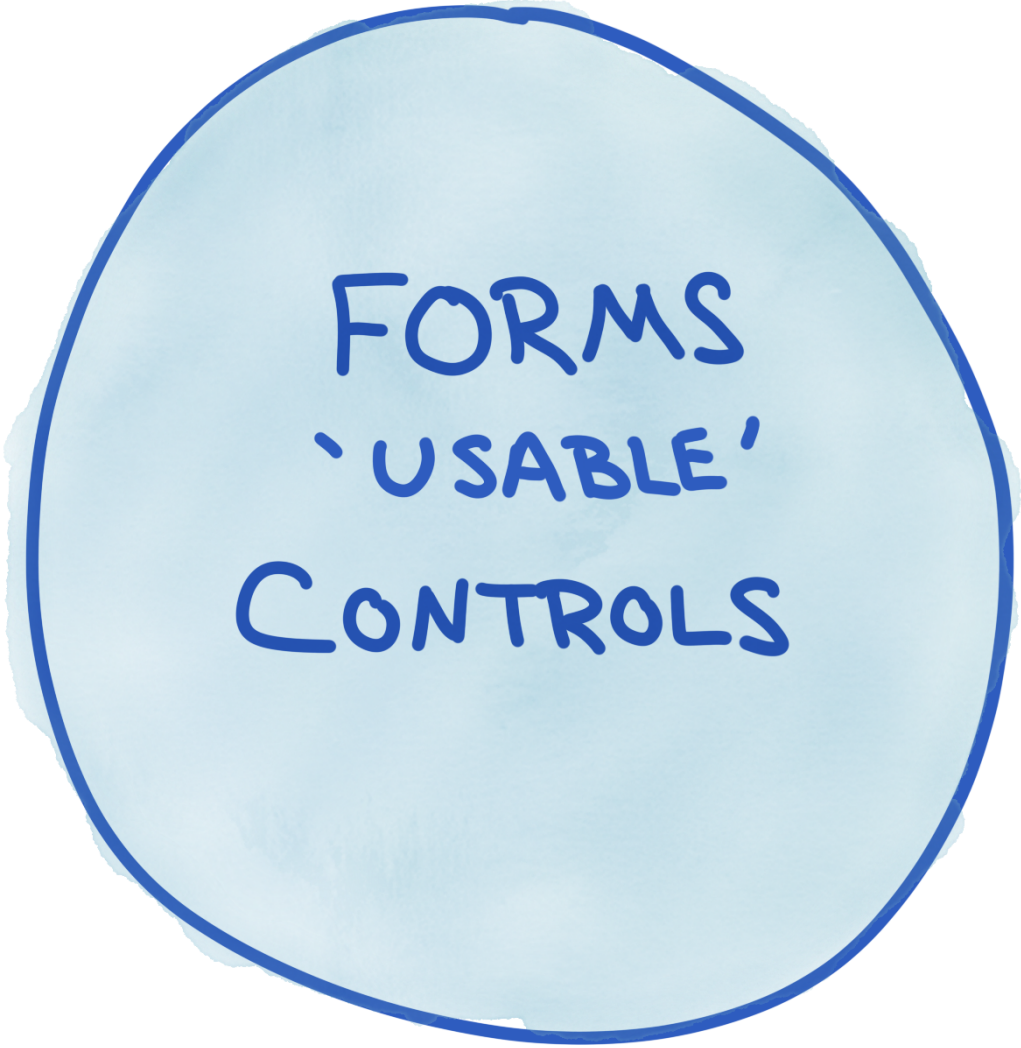
- Compile XAML
- Optimize lists
- Layout tips
- General performance
- http://bit.ly/forms-perf



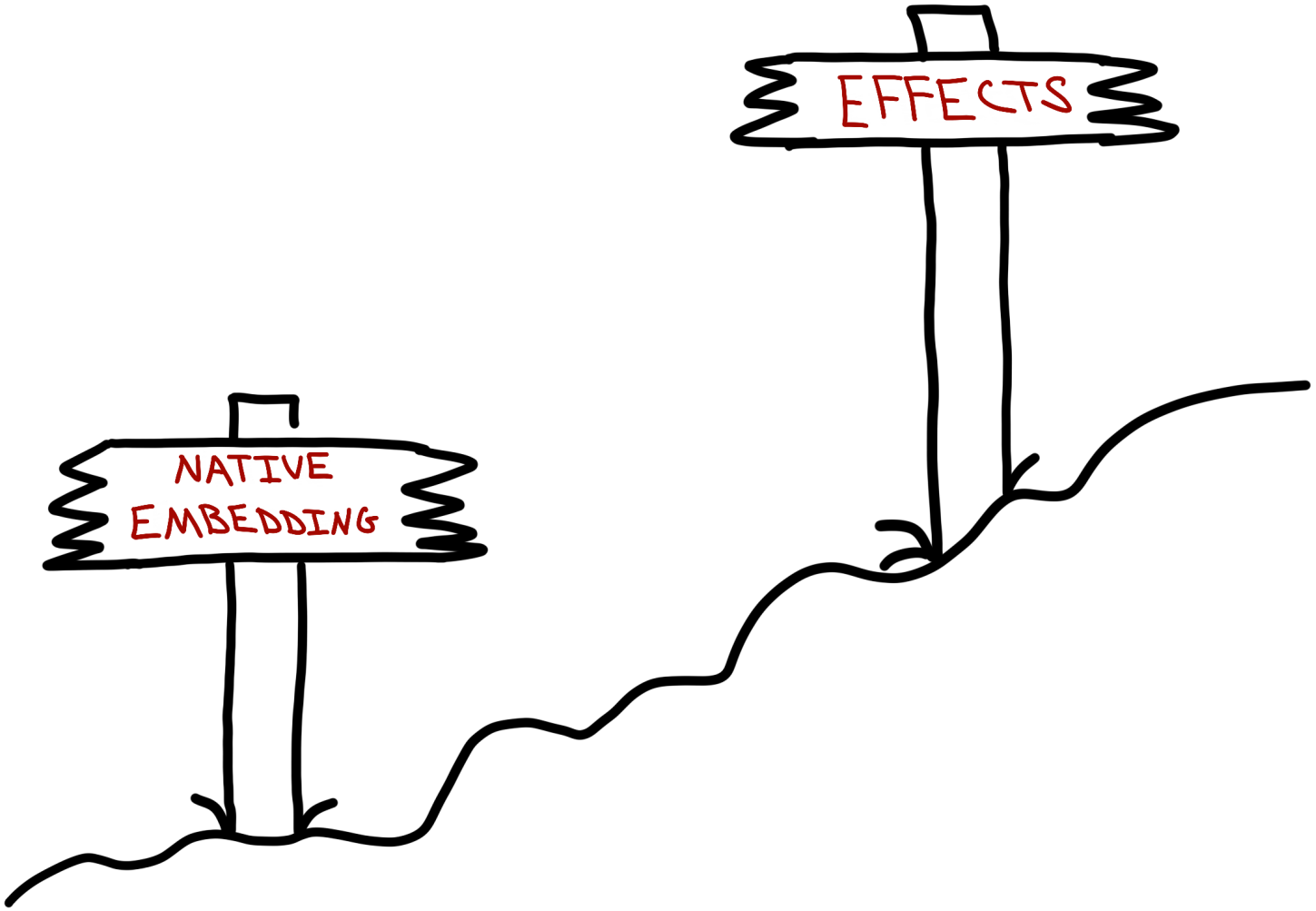
UI Customization



UI Customization



Platform Specific UI Customizations



Native Embedding

- Add platform-specific controls inline
- Any element that allows content or has Children
- Words best with Shared Projects
- Compiler directives
  - OR – creative use of partial classes

Native Embedding

```
#if __ANDROID__
    var cdc = new CustomDroidControl(Forms.Context);

    stack.Children.Add(cdc);

#elif __IOS__
    var segmentControl = new UISegmentedControl();

    stack.Children.Add(segmentControl);

#endif
```

XAML Native Embedding!

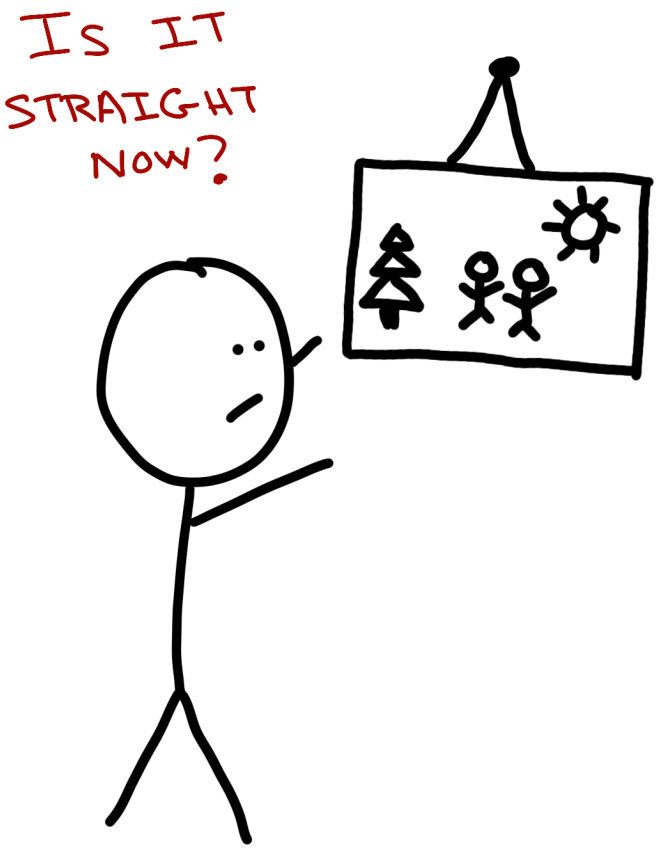
- Introduced in Forms 2.3.3 (pre-release)
- Add native views directly to XAML

```
xmlns:ios="clr-namespace:UIKit;assembly=Xamarin.iOS;targetPlatform=iOS”

...

<ios:UILabel Text="Native Text" View.HorizontalOptions="Start"/>
```

- Add XML namespaces
- Reference the controls

Effects

- Allow native controls to be customized
- Great for small changes
- Recommended for when changing *properties* will achieve desired result
- Use custom renderers when need to change behavior



Platform Effects

Basics

- Implemented in platform code
  - Has reference to platform and Xamarin.Forms specifics
- Consumed in shared code
- ResolutionGroupName
- ExportEffect attribute

Effects' Properties

- Container
  - Native renderer / parent
  - UIView or ViewGroup
- Control
  - Native control
  - UIView or View
- Element
  - Xamarin.Forms control
  - Element

```
protected override void OnAttached()
{
    if (Element is Label == false)
        return;

    var nativeLabel = Control as UILabel;

    oldShadowOffset = nativeLabel.ShadowOffset;

    nativeLabel.ShadowOffset = new CGSize(2, 3);
}
```

OnAttached

- Called when added to control
- Adjusts visual properties
- Handle events

```
protected override void OnDetached()
{
    if (oldShadowOffset != null)
    {
        var nativeLabel = Control as UILabel;
        nativeLabel.ShadowOffset = oldShadowOffset;
    }
}
```

OnDetached

- Called when effect being removed
- Reverse visual changes
- Unsubscribe from events

Adding Effects

```
<Label Text="Effects change the UI.">
    <Label.Effects>
        <ef:LabelFontEffect />
    </Label.Effects>
</Label>
```

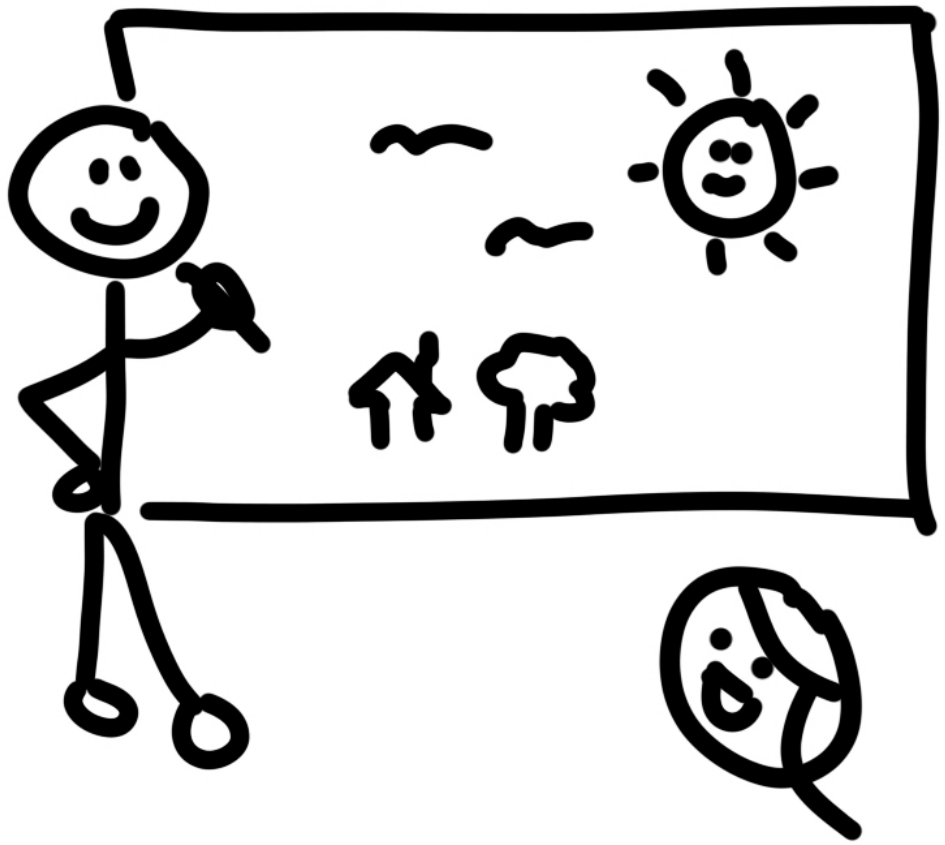
```
labelWelcome.Effects.Add(fontEffect);
```

**** Must ”resolve” effect first!**

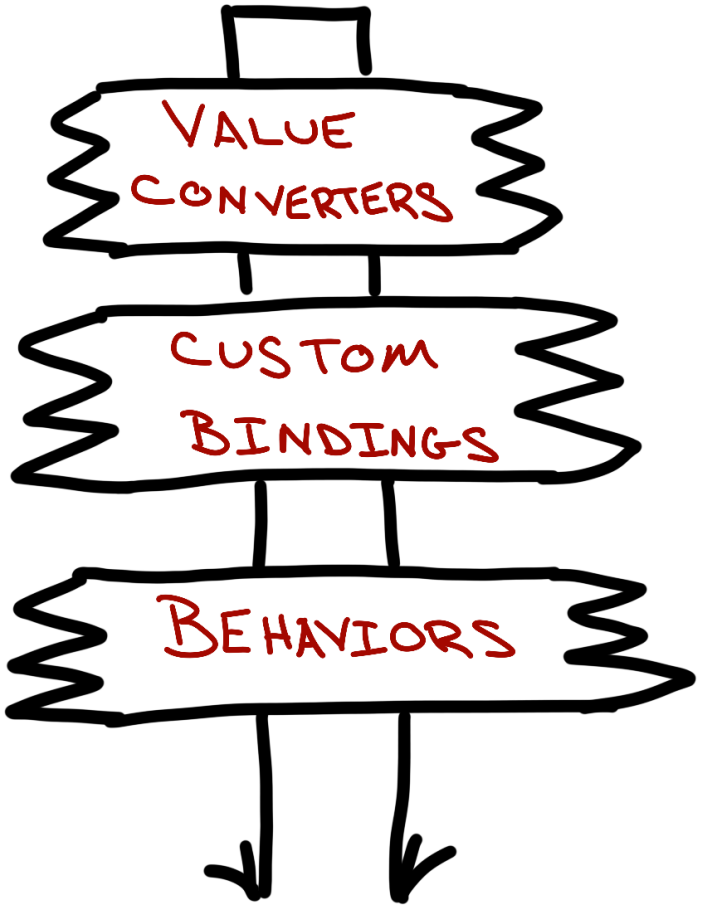

DEMO

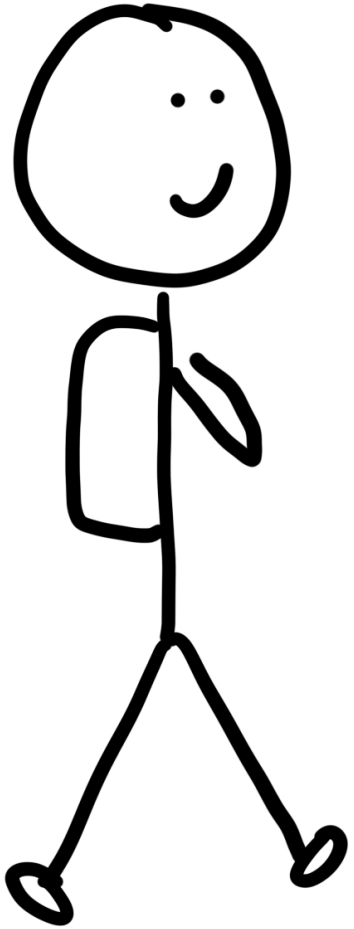
UI Customizations

- Native embedding
- Platform effects

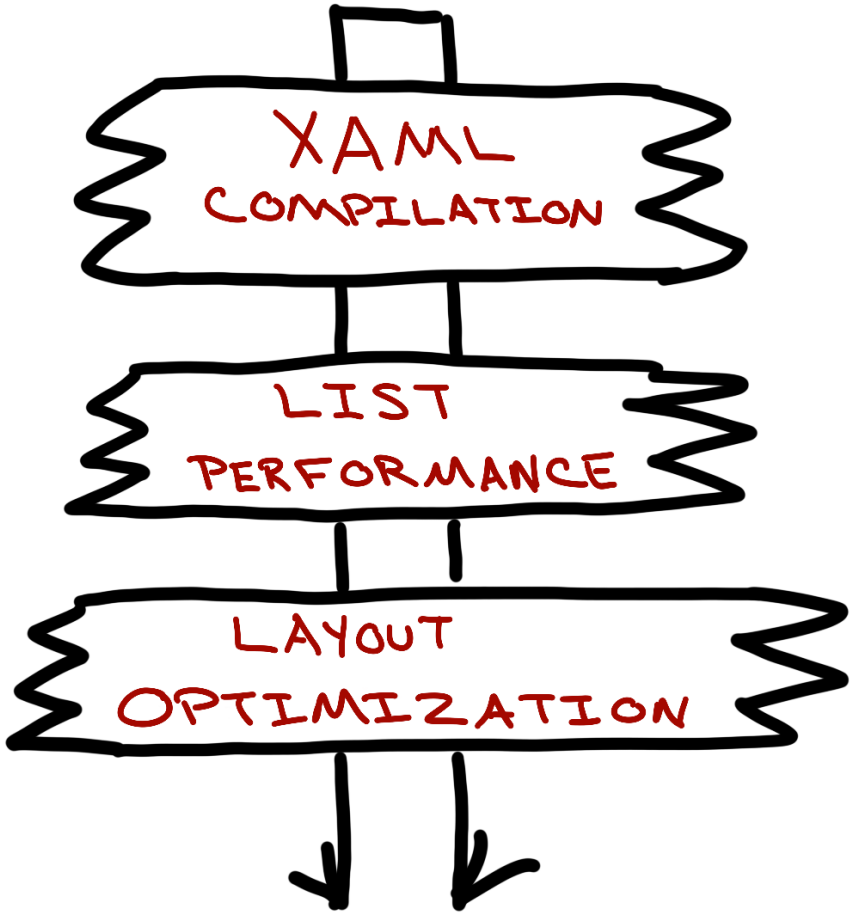


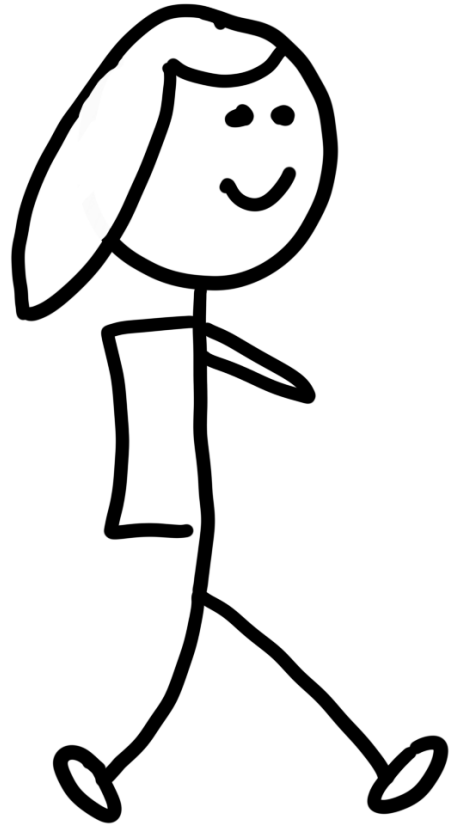
Out of the woods … use what we have!



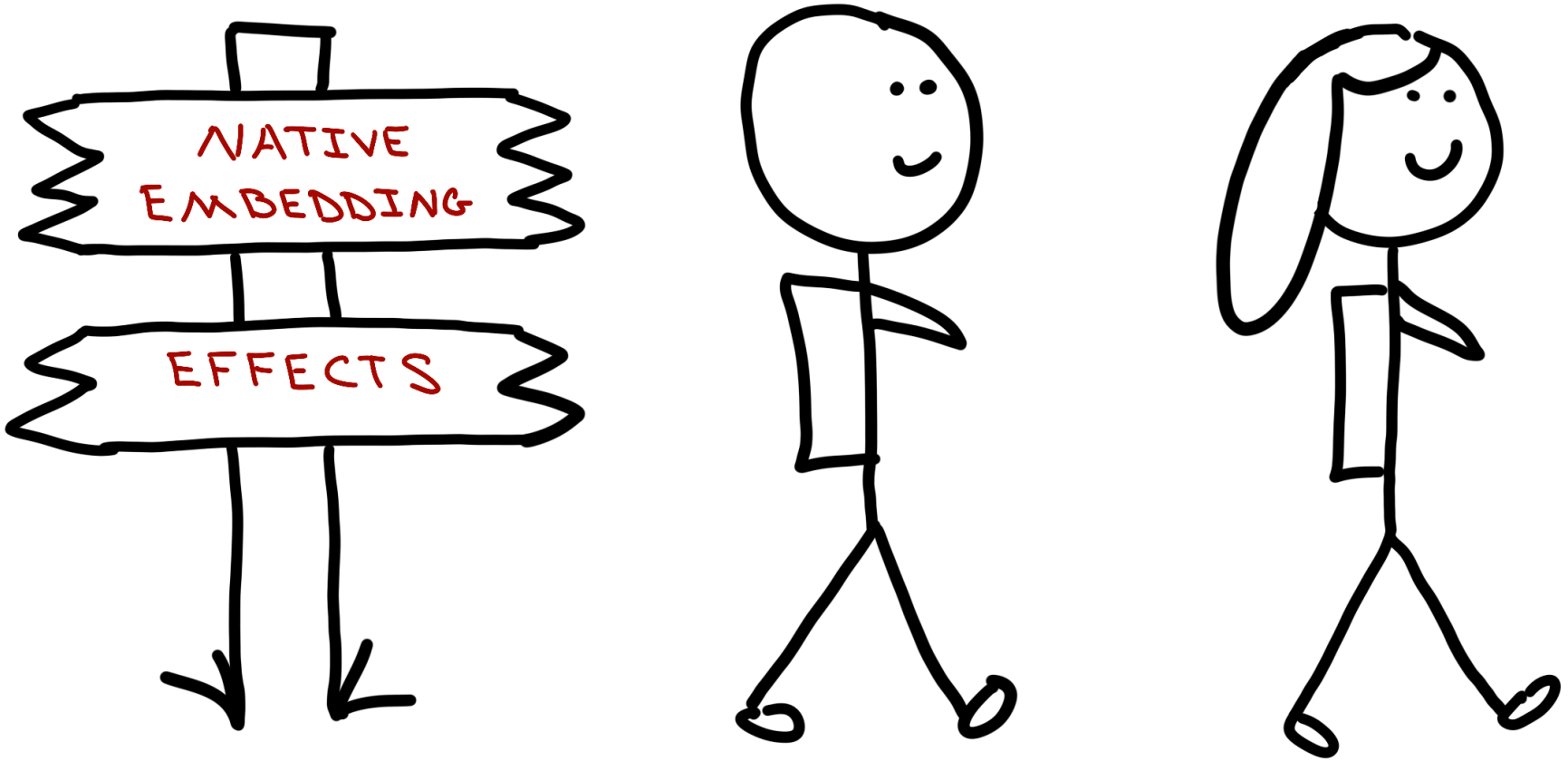


Out of the woods … performance!





Out of the woods … UI customization!



Deep Into the Woods with Xamarin.Forms

MATTHEW SOUCOUP
Principal
@codemillmatt codemilltech.com

http://bit.ly/forms-woods