


MVVM in Xamarin.Forms

Adam Kemp
Principal Software Architect
National Instruments



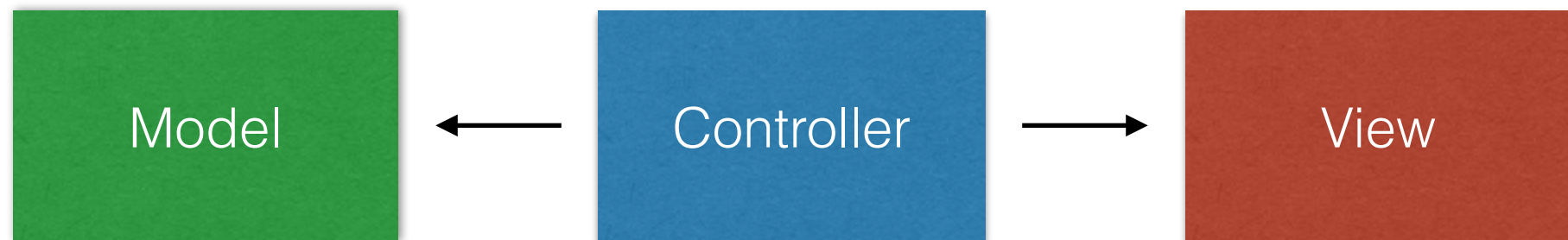
 @TheRealAdamKemp
blog.adamkemp.com

 github.com/TheRealAdamKemp

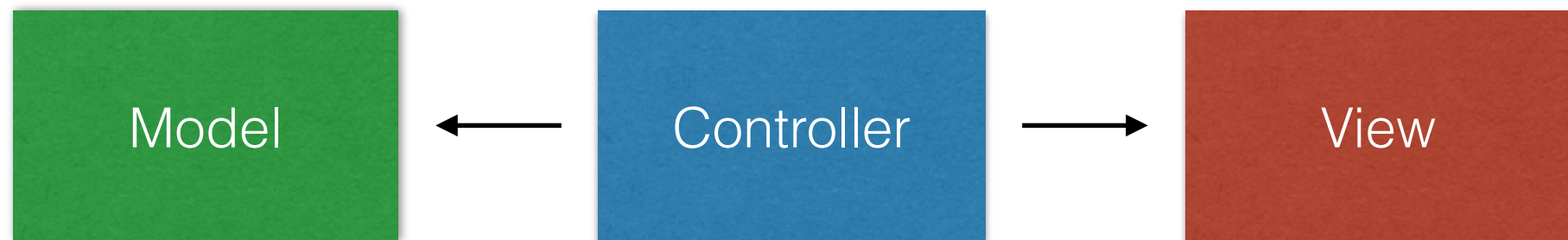
MVC

- **Model:** Data and business logic layer
- **View:** UI layer
- **Controller:** Coordinates between View and Model

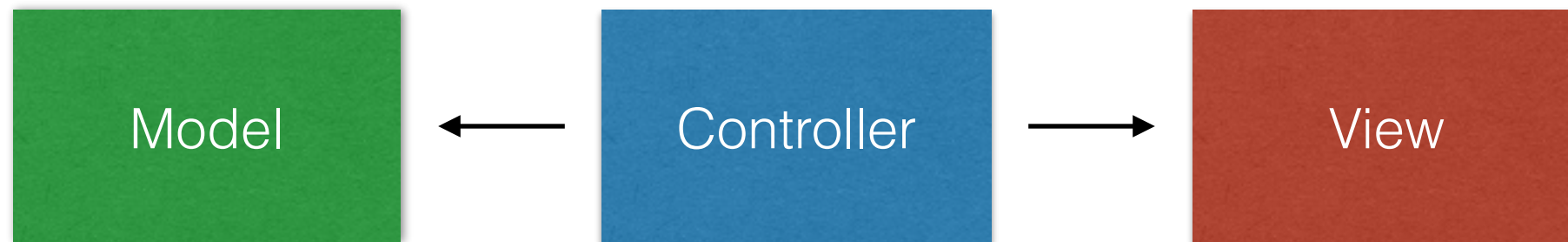
MVC



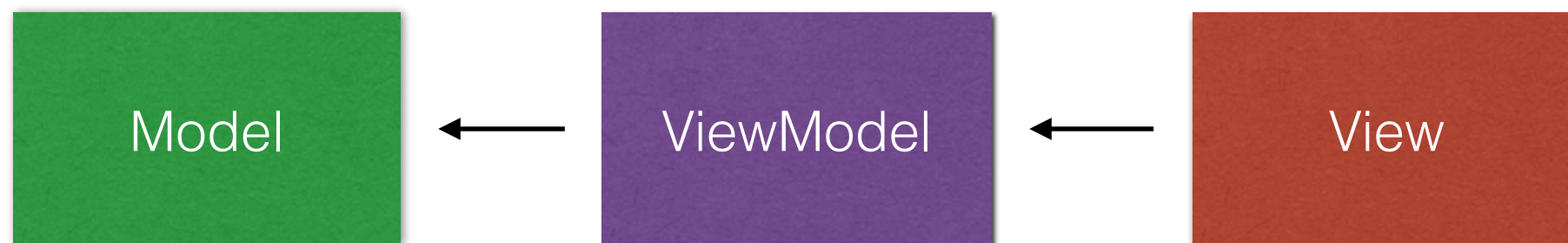
MVC



MVC



MVVM

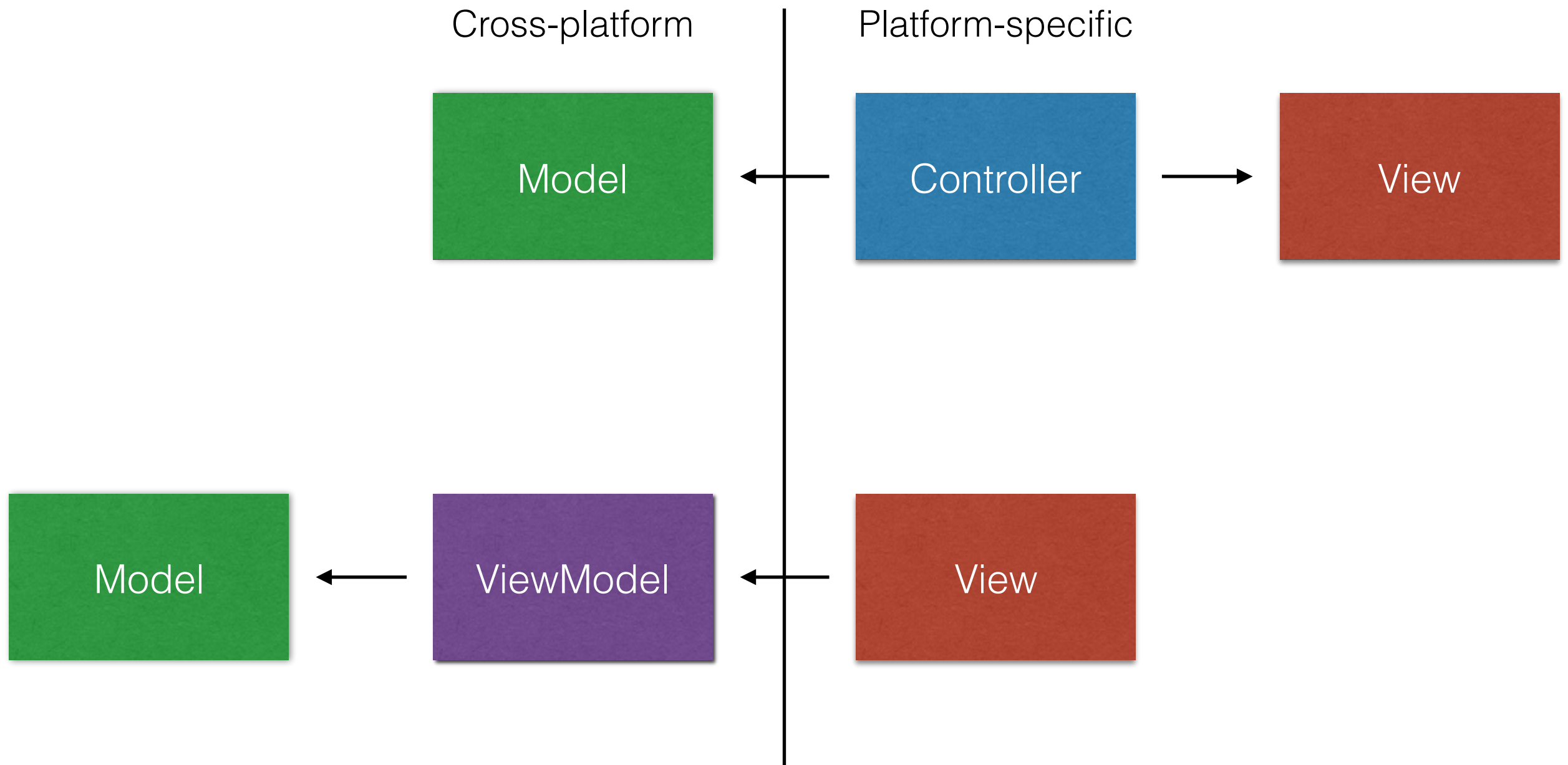


MVVM

- **Model:** Data and business logic layer
- **View:** UI layer
- **View Model:** State and logic for UI layer

Benefits of MVVM

- **More** of your code is portable



Benefits of MVVM

- **Most** of your code is unit testable
- Very little code requires a real UI
- View models are just data and logic

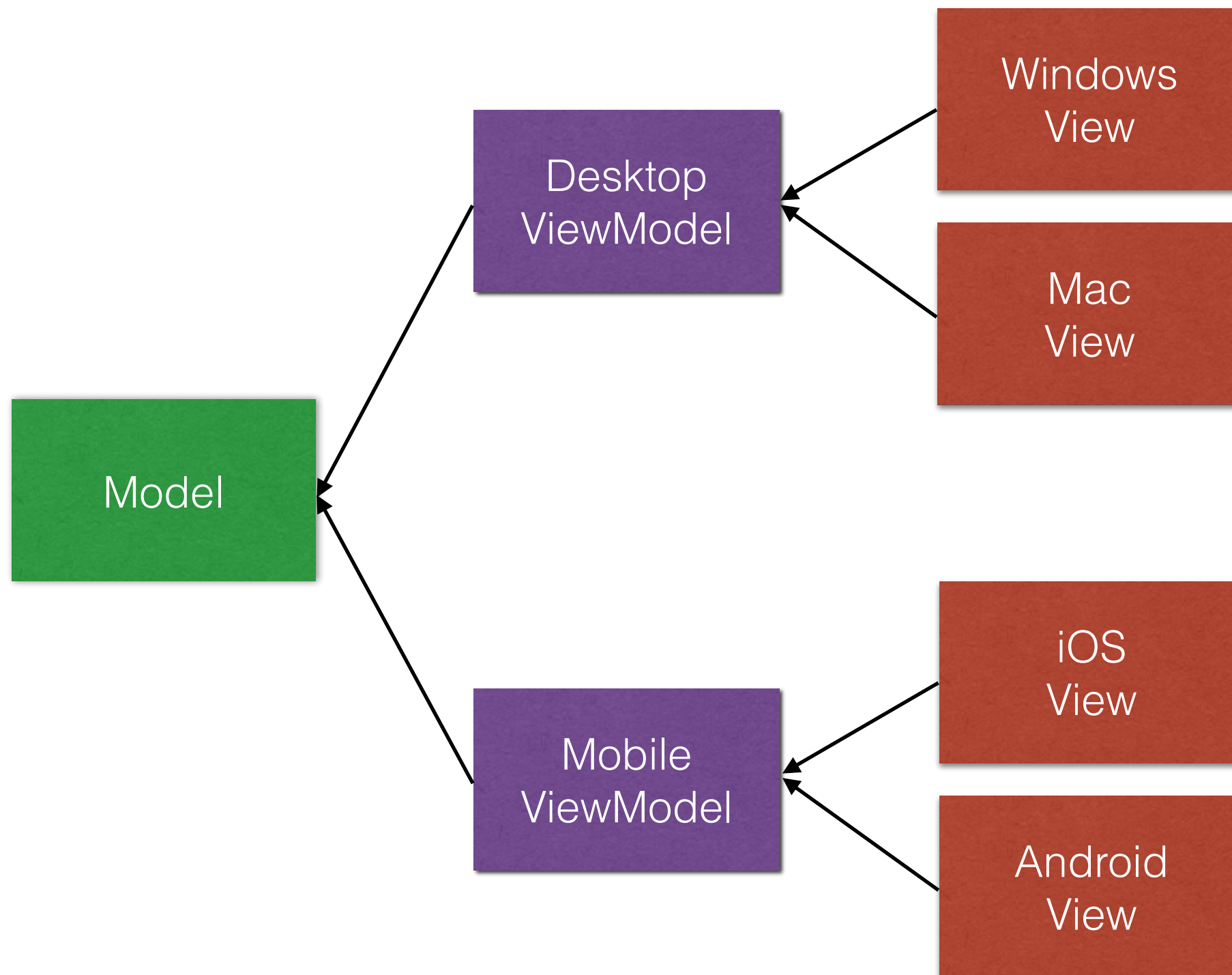
MVVM Roles

- **Model:** Data and business logic layer
- **View Model:** State and logic for UI layer
- **View:** UI layer

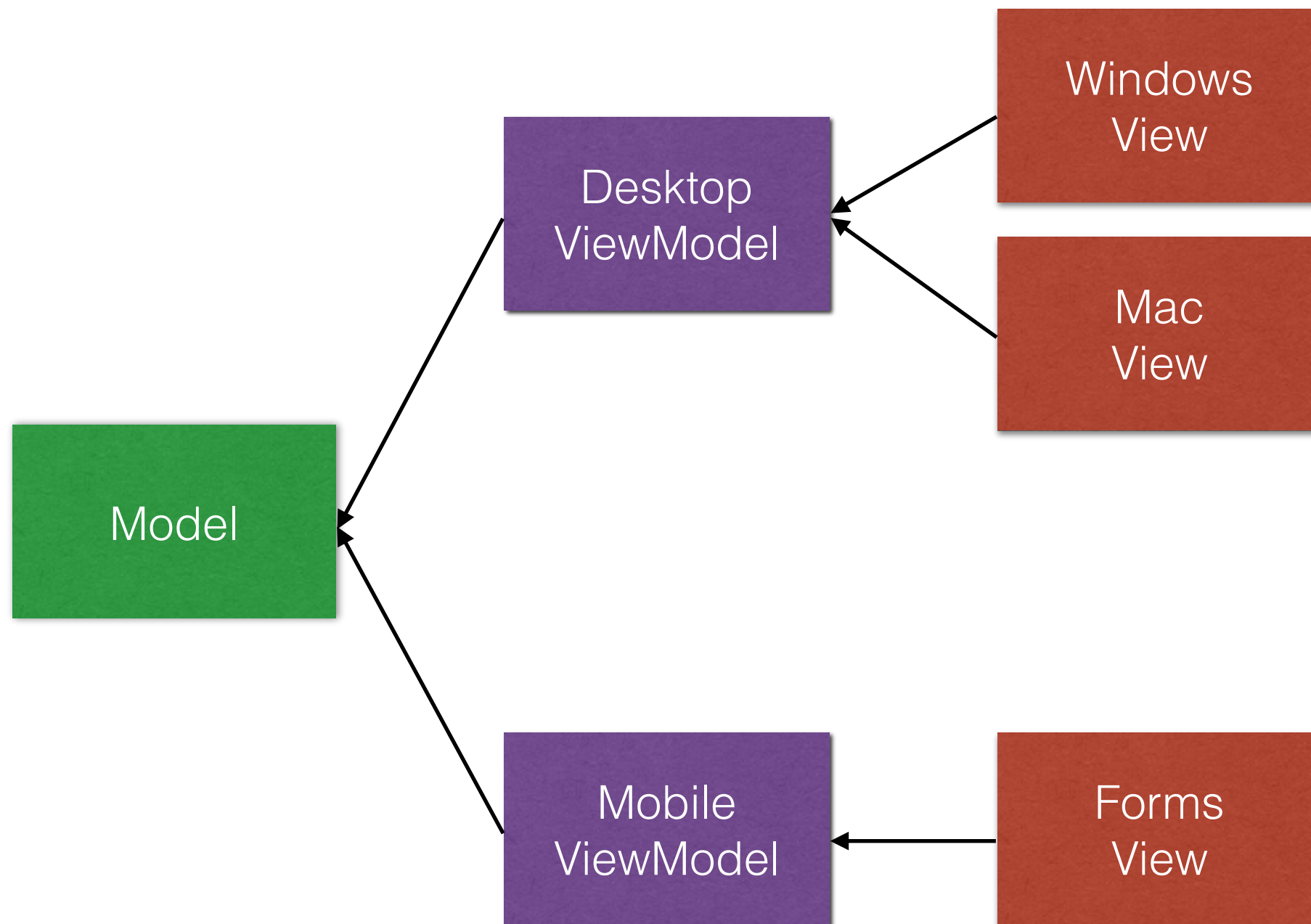
MVVM Roles

- **Model:** Application-specific data and logic
- **View Model:** UX-specific data and logic
- **View:** Platform-specific UI

MVVM Roles



MVVM Roles



MVVM Roles - Model

- Most portable code
- Should **not** reference the view or any view types

MVVM Roles - Model

```
public class TResult  
{  
    public bool Passed { get; set; }  
  
    public Color ResultColor { get { return Passed ? Color.Green : Color.Red; } }  
}
```



No!

MVVM Roles - View Model

- View-related state and behavior/logic
- Translates model data into form appropriate for the view
- Translates user input into form appropriate for the model

MVVM Roles - View Model

```
public class TResult
{
    public bool Passed { get; set; }
}

public class TResultViewModel
{
    public TResult TResult { get; set; }

    public Color ResultColor { get { return TResult.Passed ? Color.Green : Color.Red; } }
}
```

*You can also use value converters for this.

MVVM Roles - View

- Presentation layer
- Minimal code
- Binds to view model
- Often simple enough to be written declaratively using XAML

MVVM

Key Concepts

INotifyPropertyChanged

- An interface for notifying other objects about changes to properties
- The foundation of the binding system
- One event: PropertyChanged
 - Event args give the name of the property that changed
- **All View Models (and most Models) should implement this**

INotifyPropertyChanged Demo

Binding

- Synchronizes a property of one object with a property of another object
- Can be one-way or two-way
- Typically used to bind properties of a view to properties of a view model

BindingContext

- A property of a BindableObject (base class for all views)
- The object which has the properties you are binding to (usually a view model)
- Inherited from parent object/view
- Equivalent to “DataContext” in Microsoft’s MVVM frameworks

Binding Demo

Commands

- Bindable event handlers
- Delegate events to view model to share more code

Commands Demo

BindableProperty

- Allows a property to be bound
- Requirements:
 - Expose a **public, static field**
 - Named **FooProperty**
 - Of type **BindableProperty**
- Create using BindableProperty.Create* method

BindableProperty Demo

ObservableCollection

- A list with an event for notifying when the list changes
- Used by ListView and other collection views
- Allows the view to reflect changes to the collection automatically
- Use this any time you bind to a collection that might change

ObservableCollection Demo

DataTemplate

- A template for inflating a collection of views
- Used to allow code outside a view to define the contents of some piece of that view
- Example: ListView's ItemTemplate property

DataTemplate Demo

Questions?