

# MVVM - Android architecture insight

Mecea Mihai

**ThoughtWorks®**

© 2021 ThoughtWorks

# Activity class

Everything goes in-here

```
4829
4830
4831
4832
4833
4834
```



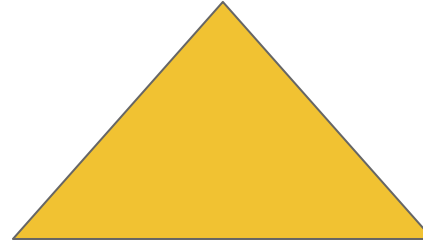
not

# Duplicates

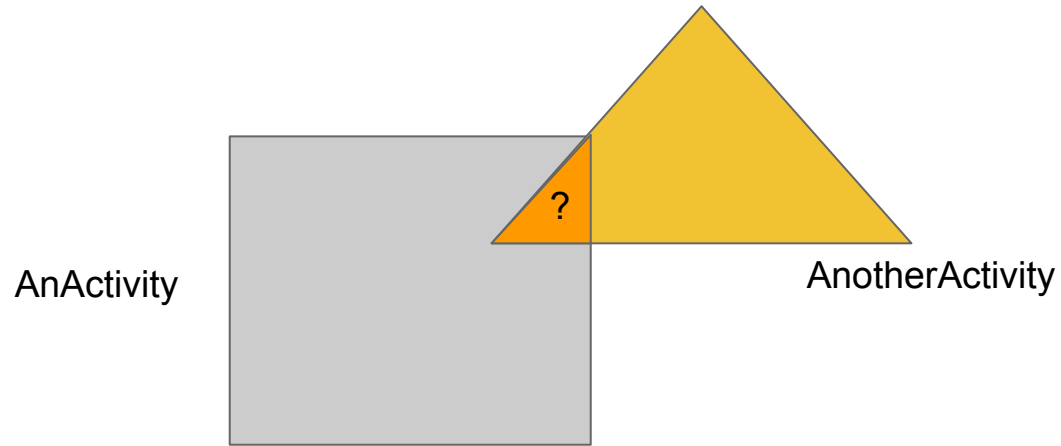
AnActivity



AnotherActivity



# Duplicates





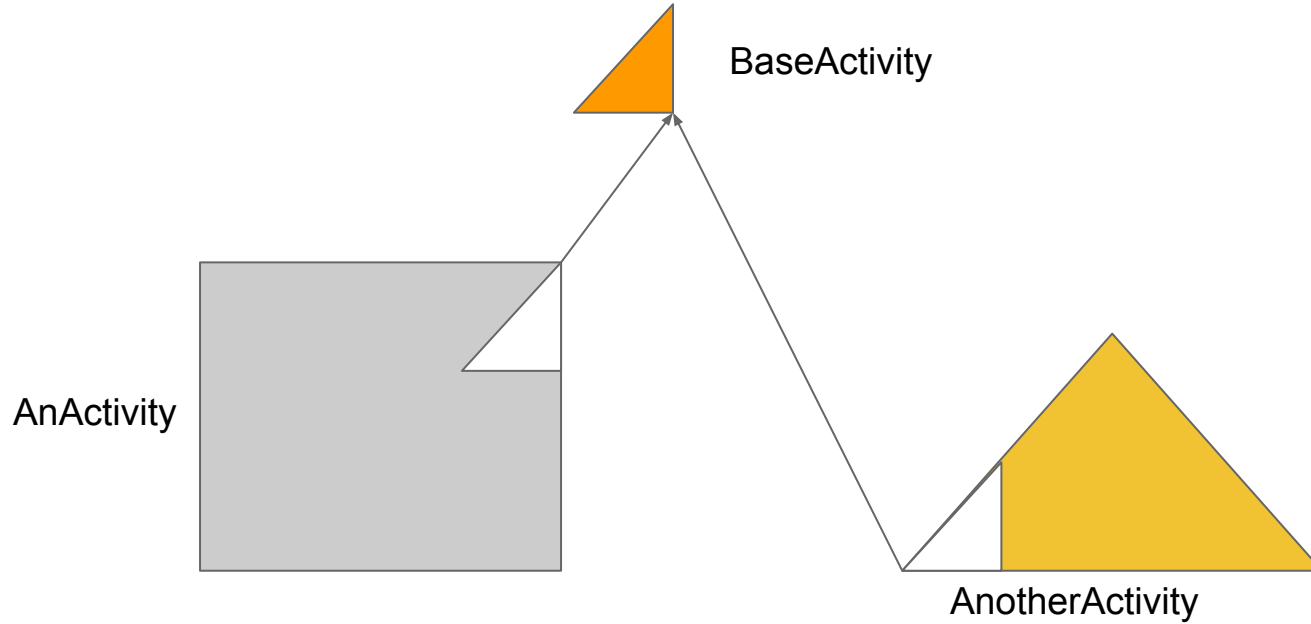
“Code duplicates are like, the worst!”

# Inheritance FTW

BaseActivity



# Duplicates



# Duplicates



BaseActivity

But what's actually in it?



# Duplicates



BaseActivity

- Is this code boilerplate?
- Is it a feature?
- Is it the actual feature or it just looks the same?

# The two look-alike adapters case study

Feature A

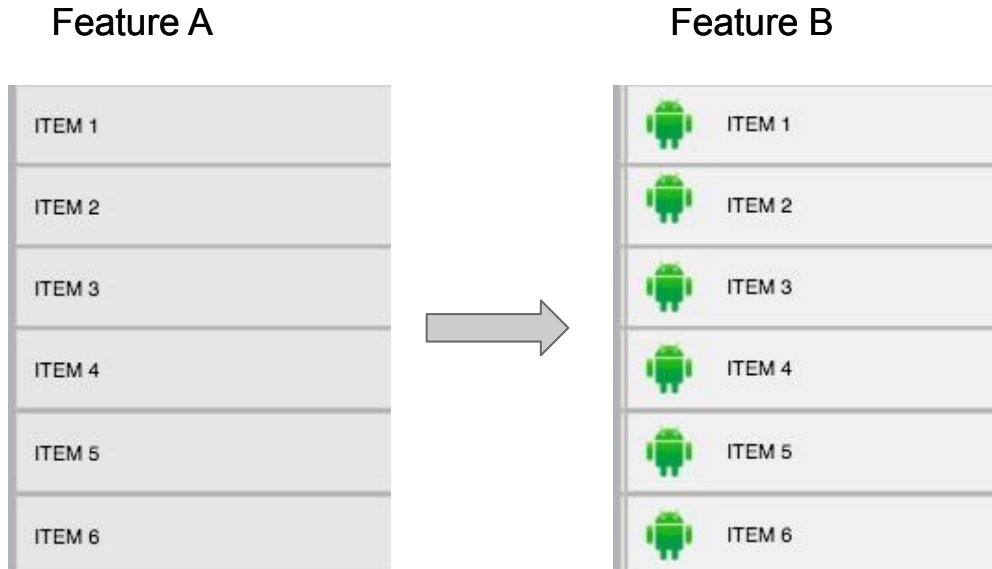
ITEM 1
ITEM 2
ITEM 3
ITEM 4
ITEM 5
ITEM 6

Feature B

ITEM 1
ITEM 2
ITEM 3
ITEM 4
ITEM 5
ITEM 6

```
FeatureAdapter {  
    + text  
}
```

# The two look-alike adapters case study



# The two look-alike adapters case study

## The power of “copy/paste”

```
AFeatureAdapter {  
    + text  
}
```

```
BFeatureAdapter {  
    + text  
    + image  
}
```

## The power of “if”

```
THEAdapter {  
    + text  
    if (this is the one with a picture) {  
        + image  
    }  
}
```

# The two look-alike adapters case study

## The power of “copy/paste”

```
AFeatureAdapter {  
    + text  
}
```

```
BFeatureAdapter {  
    + text  
    + image  
}
```

## The power of “if”

```
THEAdapter {  
    + text  
    if (this is the one with a picture) {  
        + image  
    }  
}
```

5 years later?



“Prefer composition over inheritance”

...



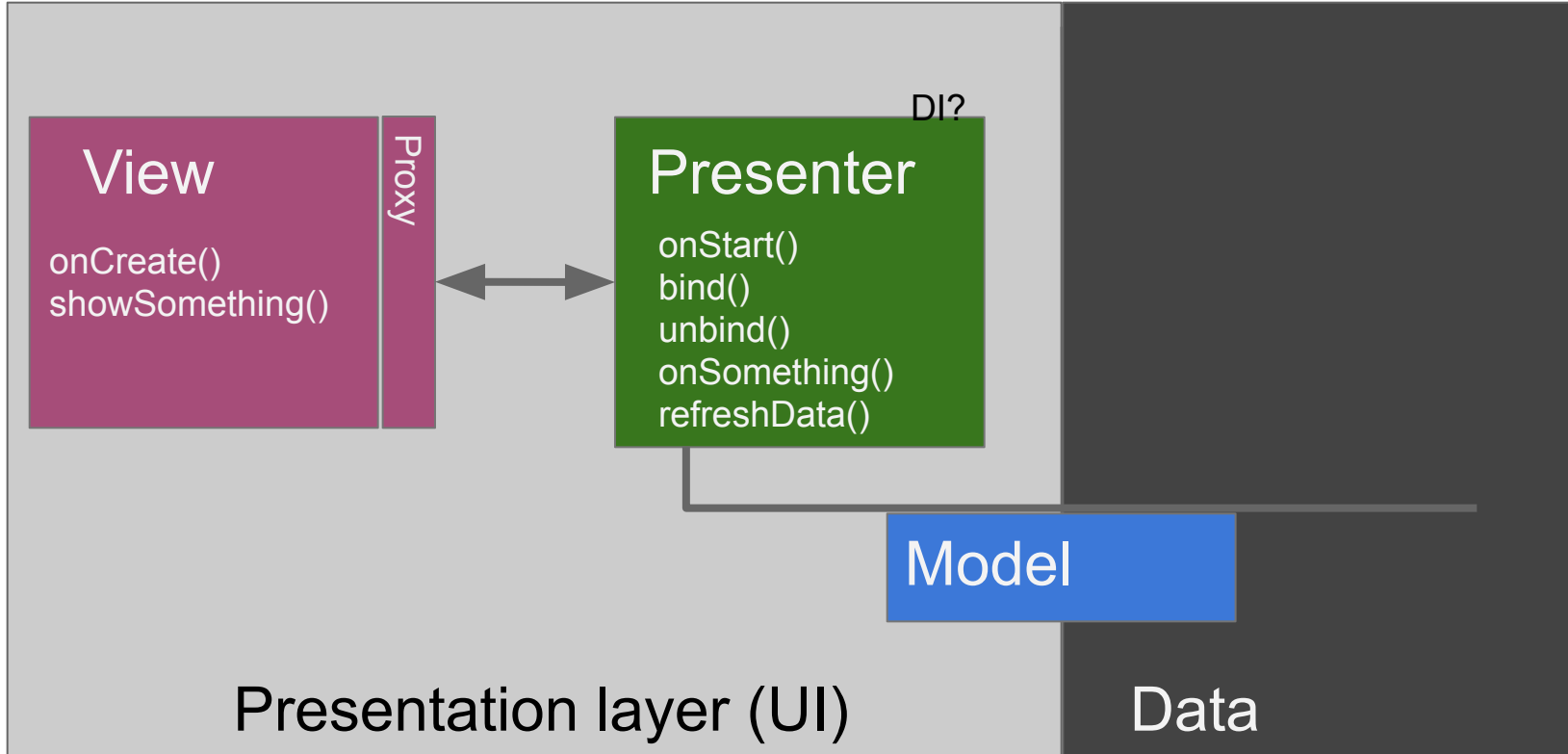
# MVP

## Splitting the Activity class

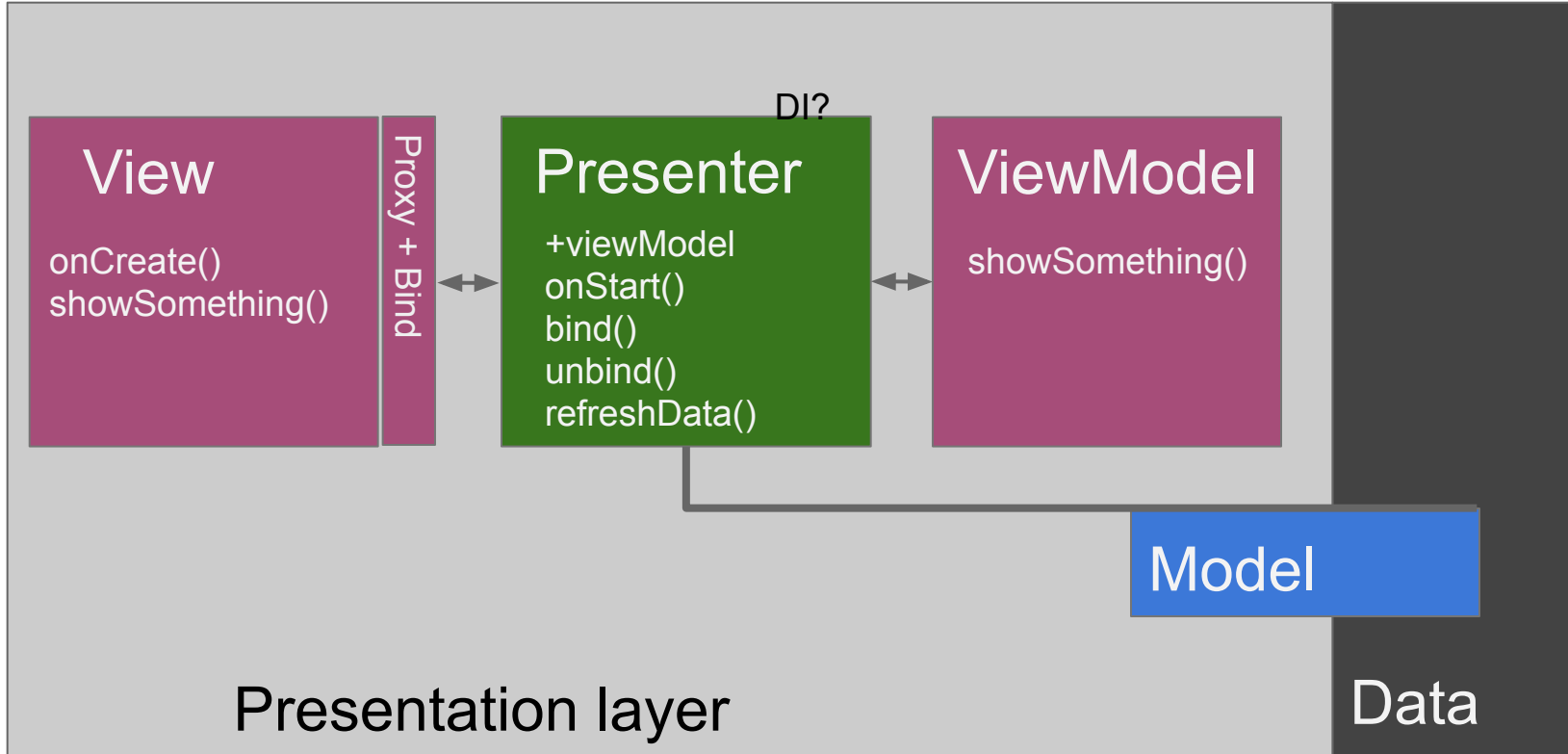




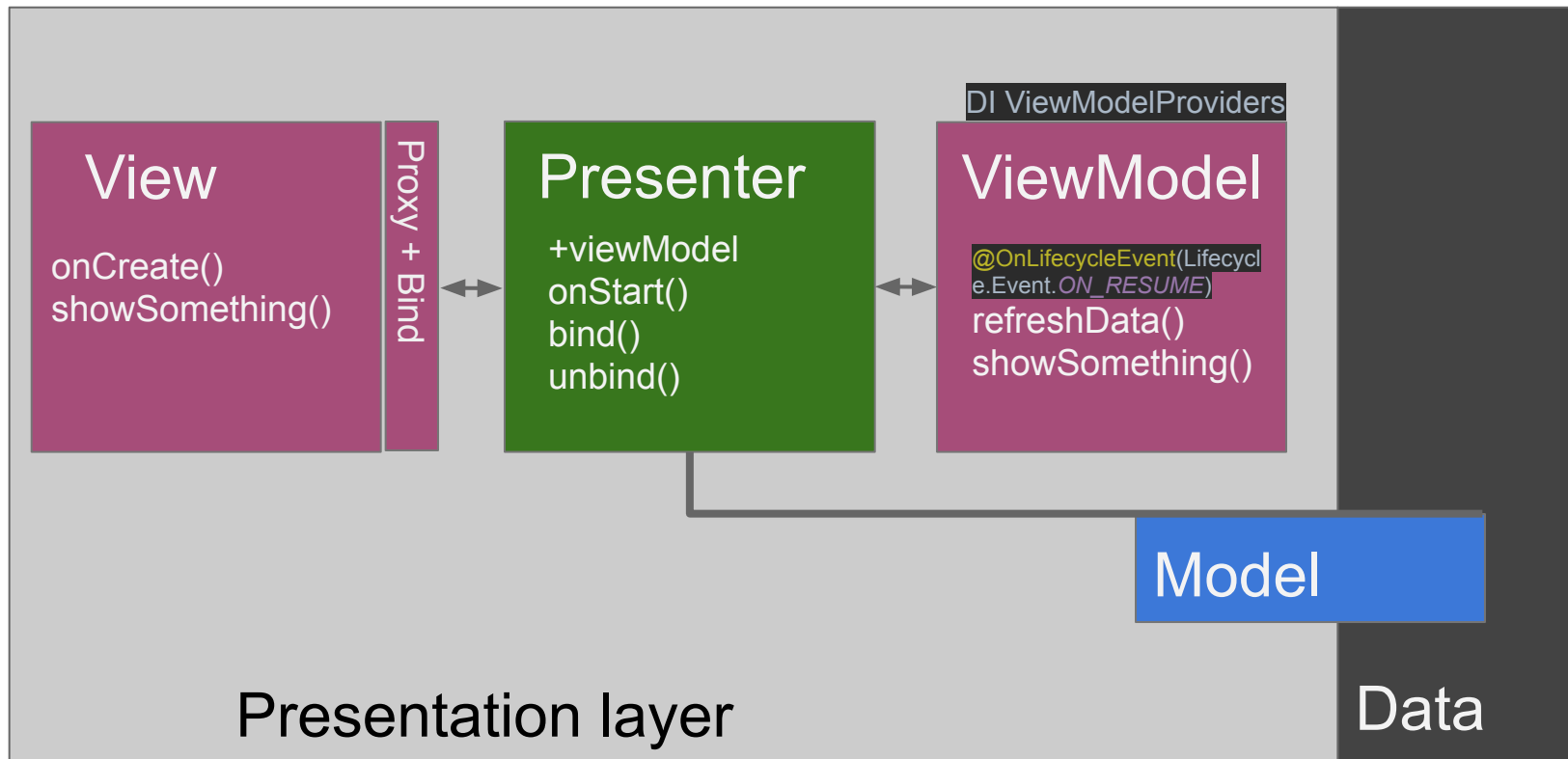
# MVP



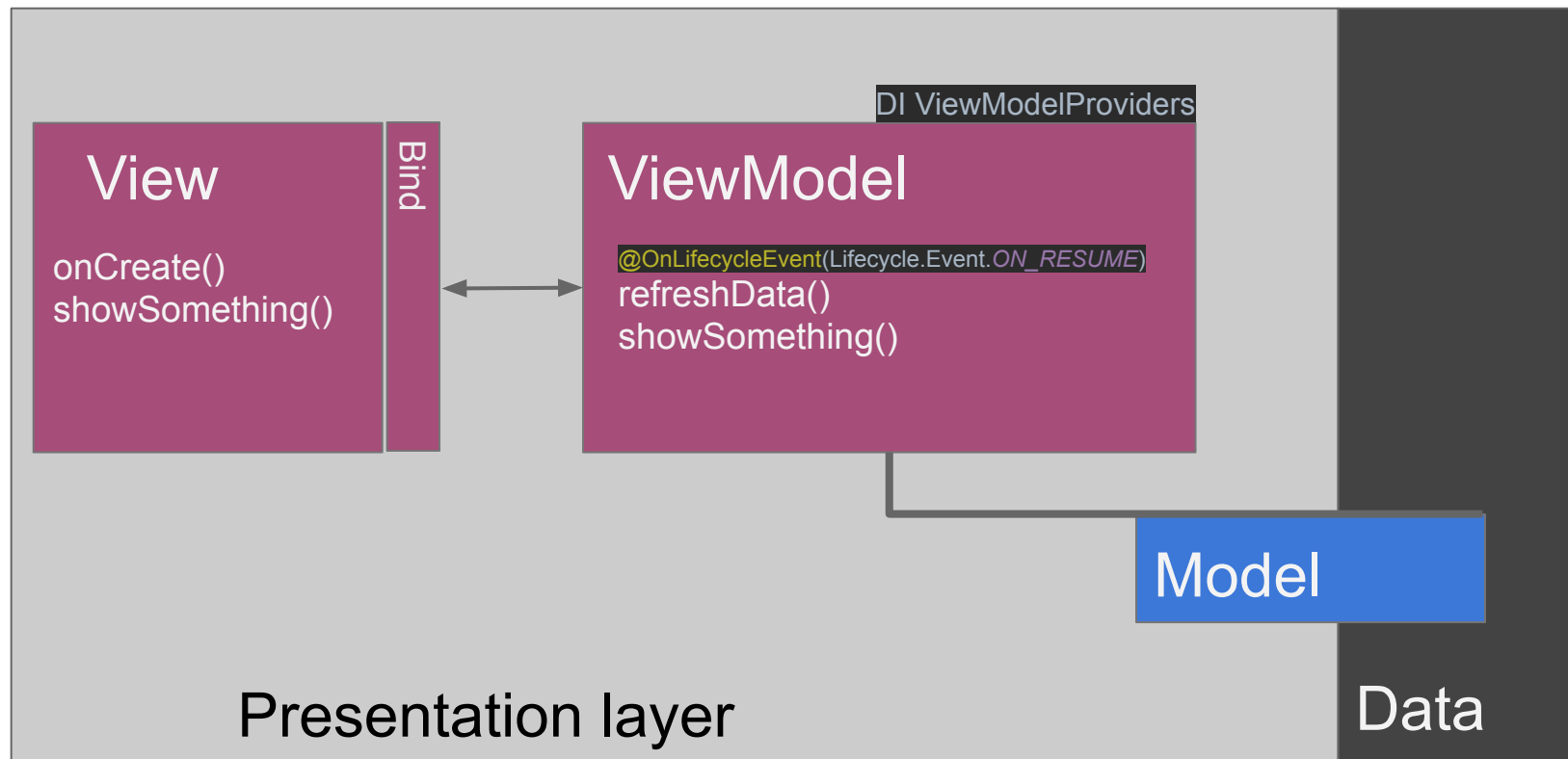
# MVVM



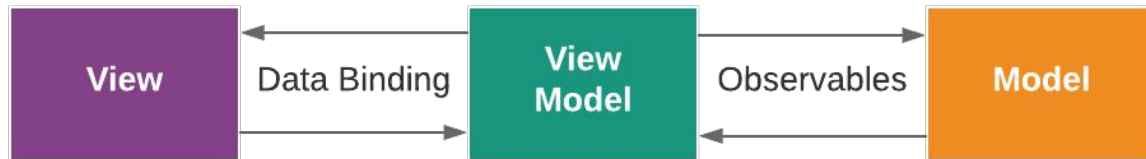
# MVVMP



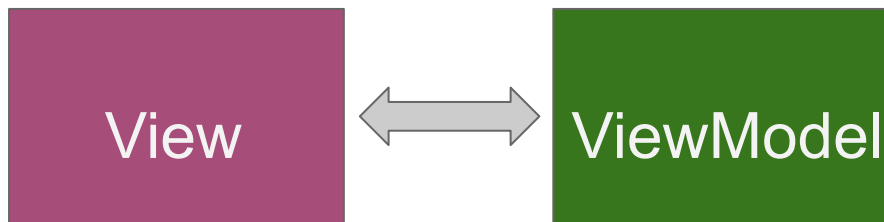
# MVVM



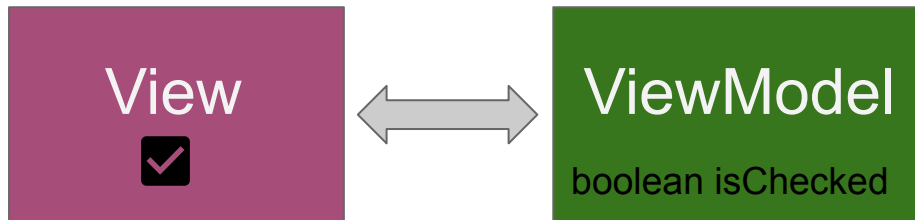
# What is MVVM?



Allows you to **bind UI components** in your layout to **data**.



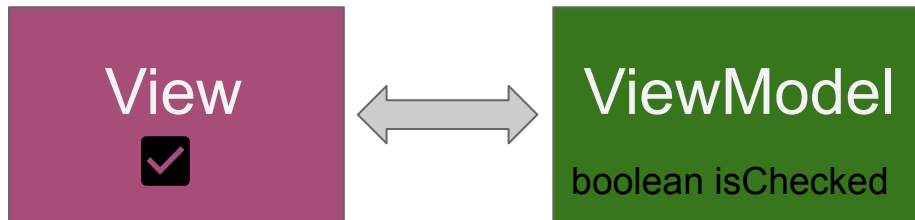
# V <-> VM



View and ViewModel mirror each other:

- If a Checkbox view is checked by the user
  - then the ViewModel will have `boolean isChecked == true`
- If in coding we set the `isChecked = true`
  - Then the Checkbox view is shown checked

$V \leftrightarrow VM$



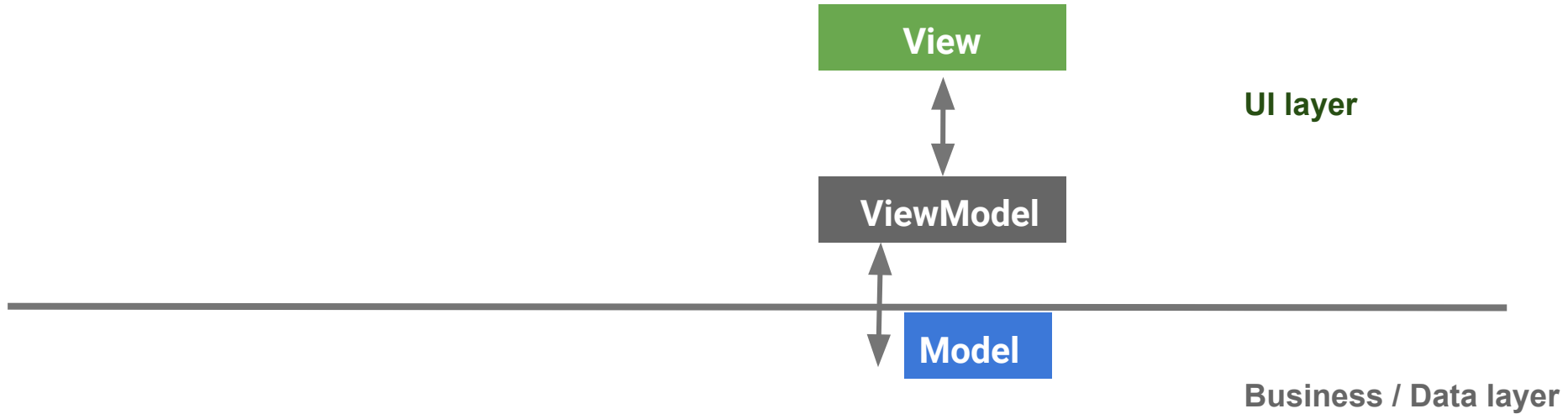
$V \leftrightarrow VM$  two way binding

$V \rightarrow VM$  one way binding

$V \leftarrow VM$  one way binding



# MVVM overview



# MVVM overview

How bad could it go?



# MVVM

```
public class SomeViewModel extends ViewModel {  
    ...  
    void onPressed(int resourceButtonPressed) {  
        User user = User.getCurrentUser();  
        String token = Preferences.getInstance(getContext()).getString(Preferences.USER_TOKEN, "");  
        RetrofitApiService.doRestCallForUser(getContext(), token, new Callback<Map<String, Object>>() {  
            @Override  
            public void onResponse(Call<Map<String, Object>> call, Response<Map<String, Object>>  
response) {  
                if (response.code() == 200) {  
                    Util.logDebugMessage(RetrofitApiService.class.getSimpleName(),  
response.body().toString());  
                } else {  
                    prefs.writeSomething...  
                }  
            }  
        })  
    }  
    ...  
}
```

# MVVM

## Spaghetti integration





# What's wrong with spaghetti?

# Documentability

How you gonna document that?

- and then wrote a callback for request
- and then i saw it had a method called onResponse
- and then i checked for 200
- and then...

If it's not documentable, it's probably not the best design

# Testability

- Don't have to actually test it
- Mocking comes natural when units are small and well thought

if it's not testable, it's probably not the best design

# Complexity

if it's not stupid simple, it's probably not the best design



# MVVM

```
public class SomeViewModel extends ViewModel {  
    ...  
    void onPressed(int resourceButtonPressed) {  
        User user = User.getCurrentUser();  
        String token = Preferences.getInstance(getContext()).getString(Preferences.USER_TOKEN, "");  
        RetrofitApiService.doRestCallForUser(getContext(), token, new Callback<Map<String, Object>>() {  
            @Override  
            public void onResponse(Call<Map<String, Object>> call, Response<Map<String, Object>>  
response) {  
                if (response.code() == 200) {  
                    Util.logDebugMessage(RetrofitApiService.class.getSimpleName(),  
response.body().toString());  
                } else {  
                    prefs.writeSomething...  
                }  
            }  
        })  
    }  
    ...  
}
```

# Complexity



**Richard Branson** ✓

@richardbranson

Follow

Complexity is your enemy. Any fool can make something complicated. It is hard to make something simple [virg.in/cye](https://www.virginia.edu/cye)

*What's the right architecture?*

*Mobile wise or even Android wise*



*What's the minimal architecture?*



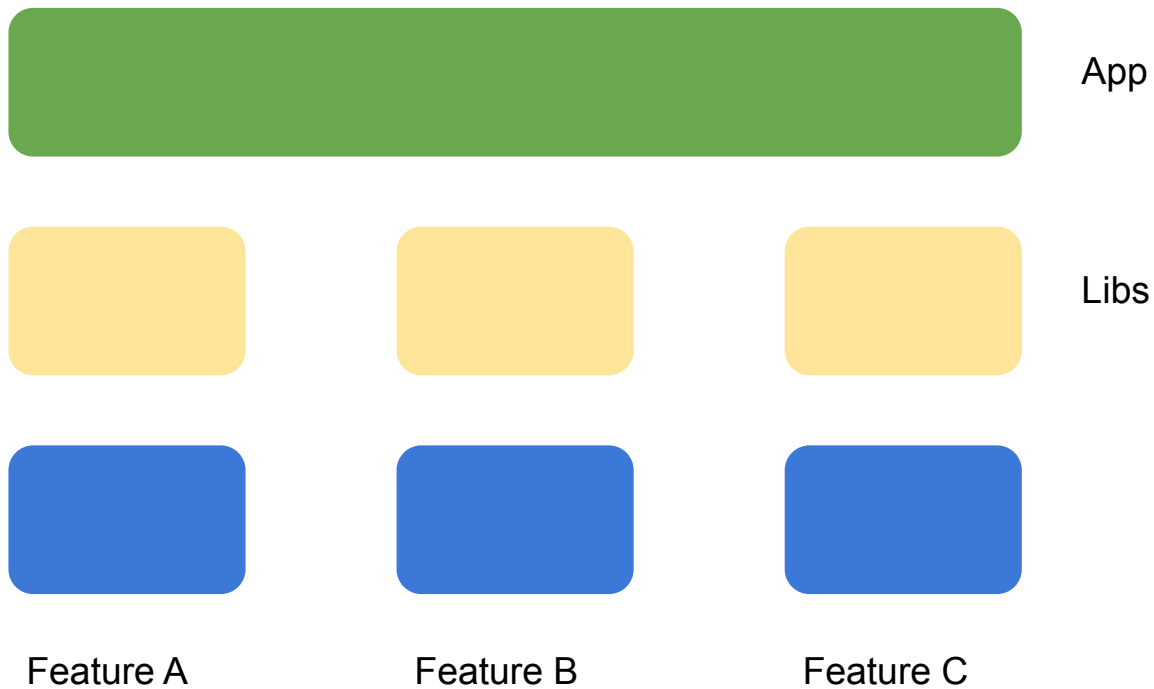
Presentation layer



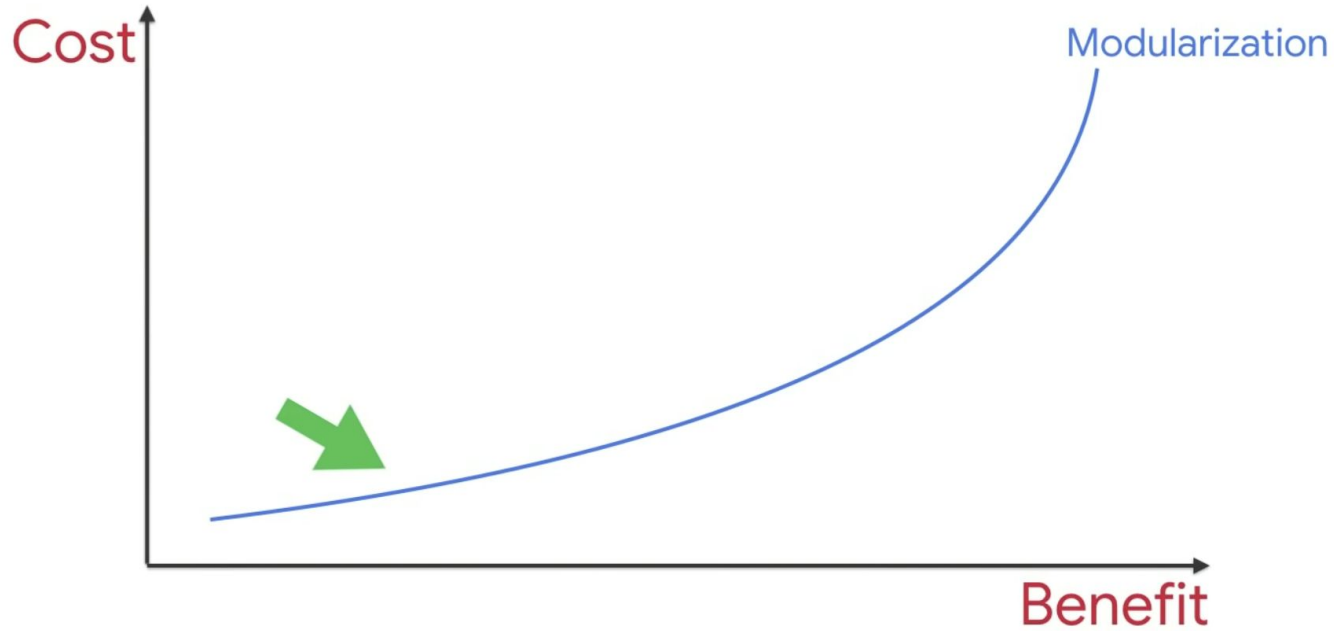
Business layer



Data layer

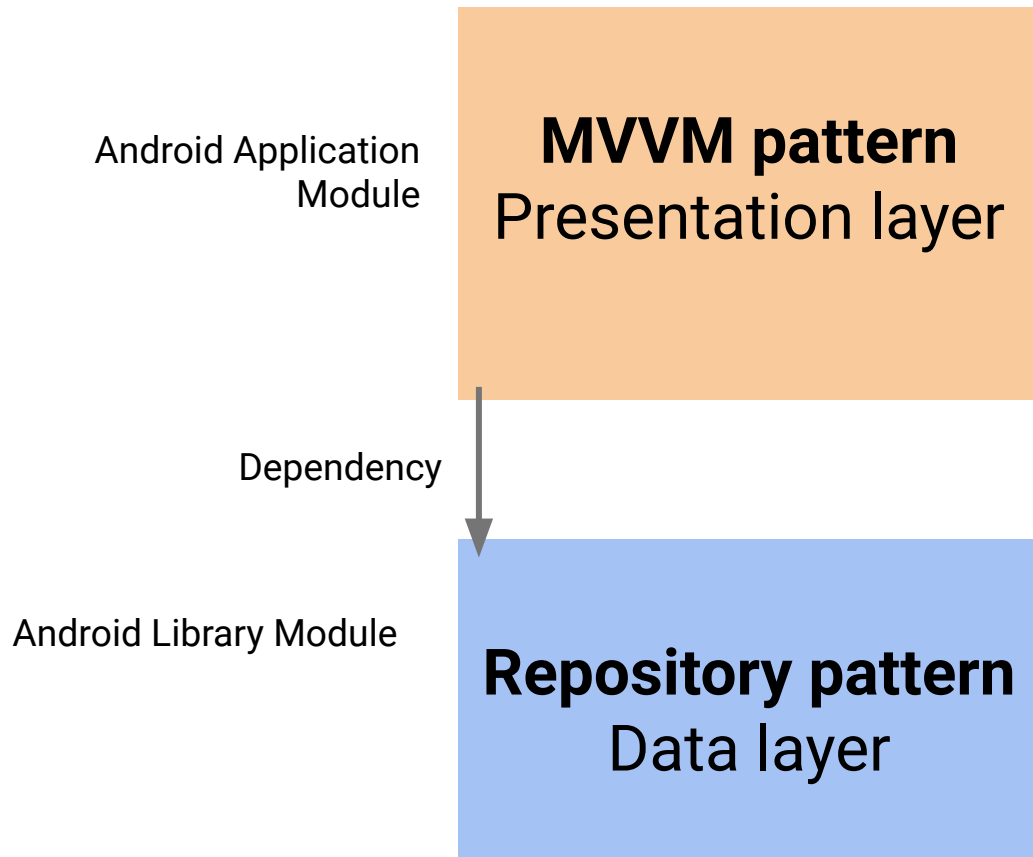


# Modularization



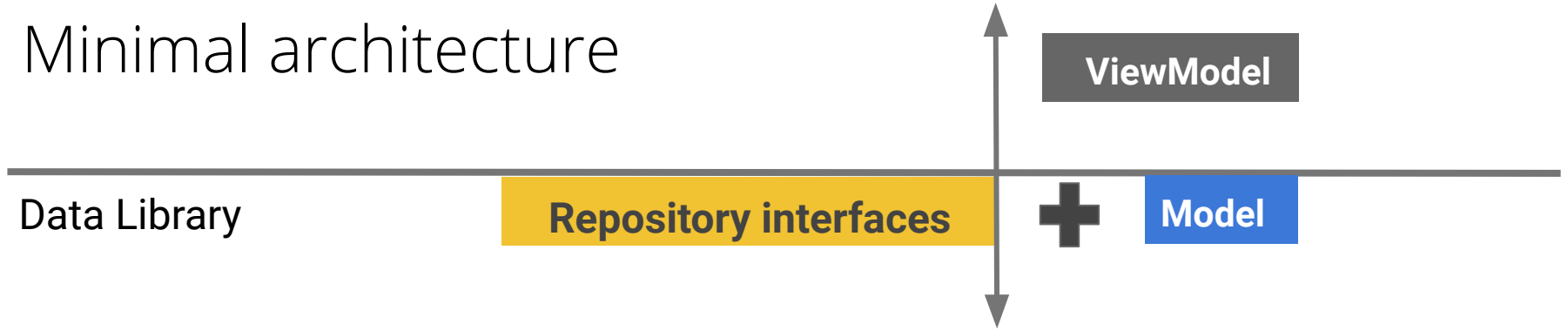
[Google I/O '19: Modular Android App](#)

# Minimal architecture

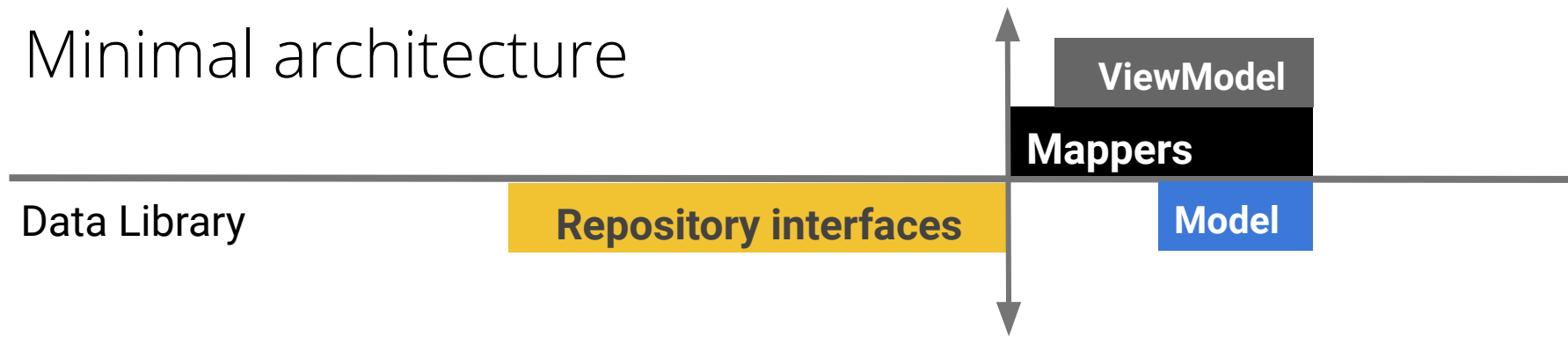




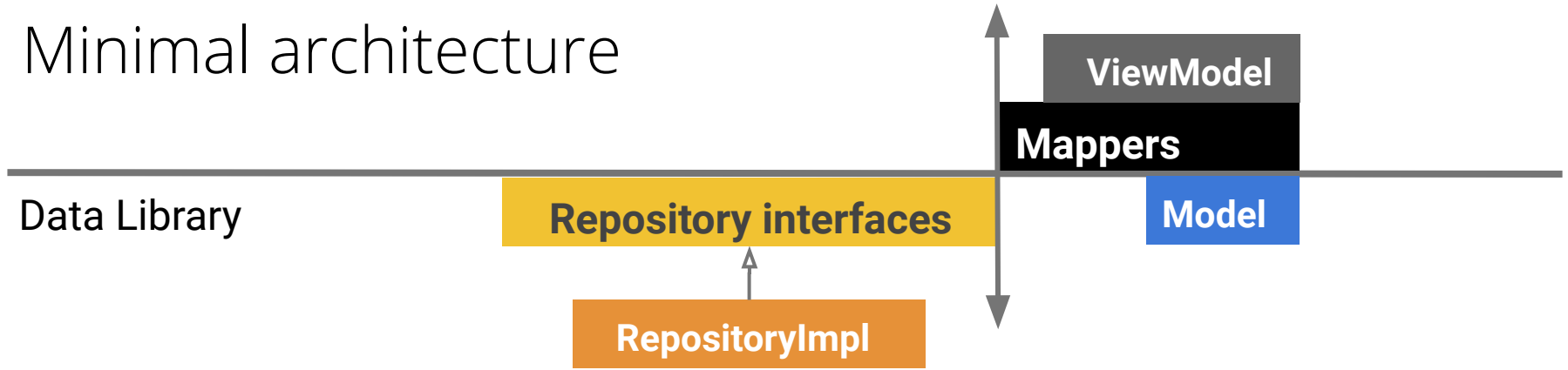
# Minimal architecture



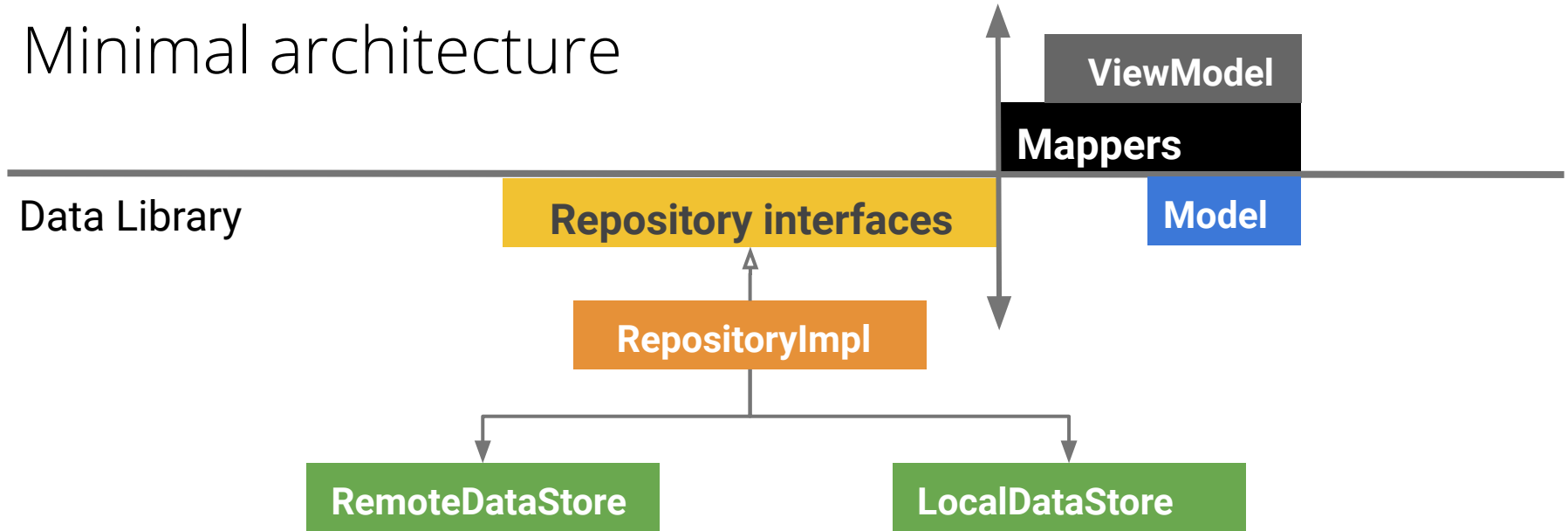
# Minimal architecture



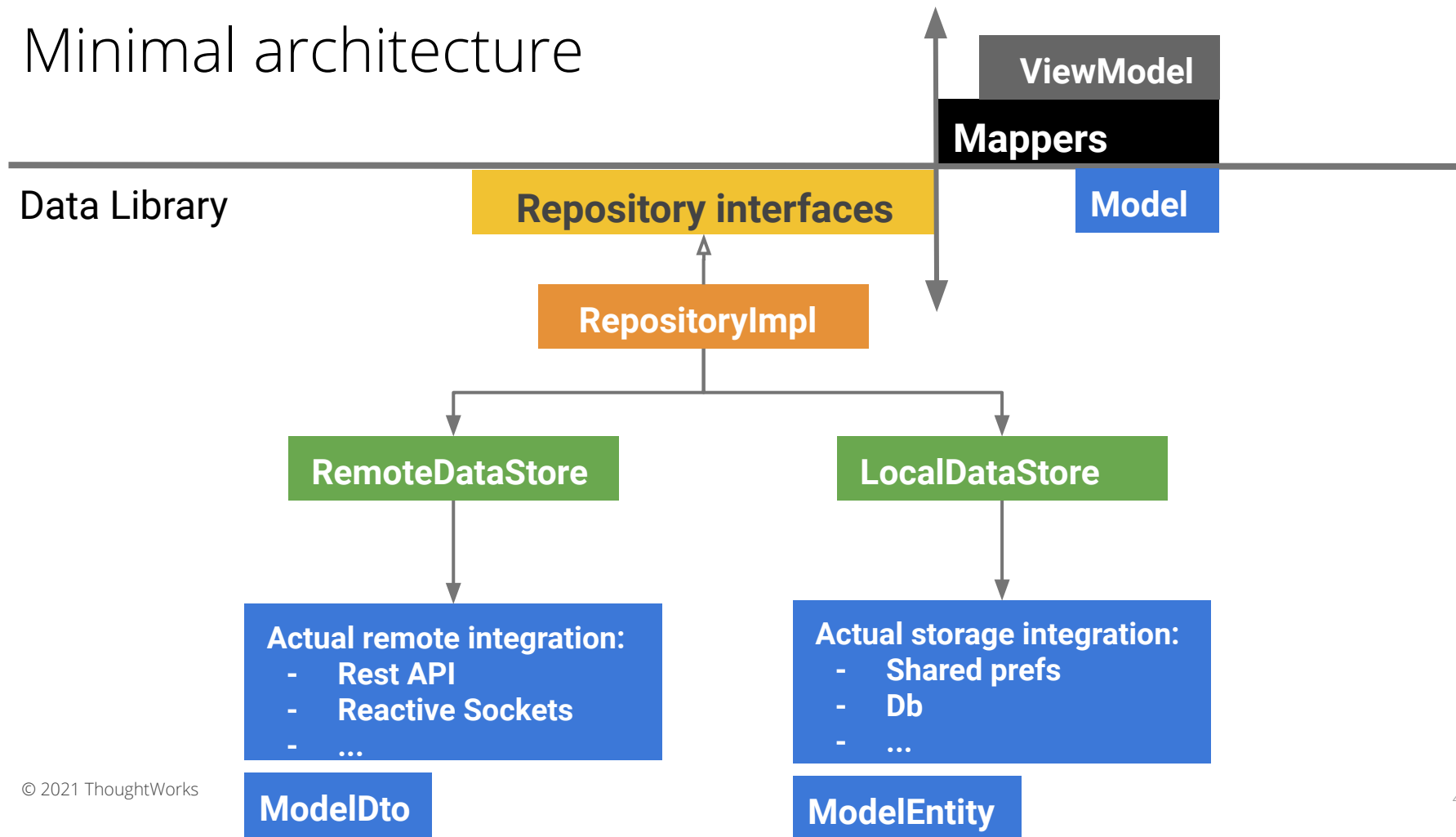
# Minimal architecture



# Minimal architecture



# Minimal architecture



# Minimal architecture

[Architecture sample](#)



# ThoughtWorks®

**MECEA MIHAI**

Android Developer

[mihai.mecea@thoughtworks.com](mailto:mihai.mecea@thoughtworks.com)