DataBindingで実現する MVVM Architecture

About me

- Kenji Abe
- MEDIROM Inc (前: Re.Ra.Ku)
- twitter/STAR_ZERO
- github/STAR-ZERO

概要

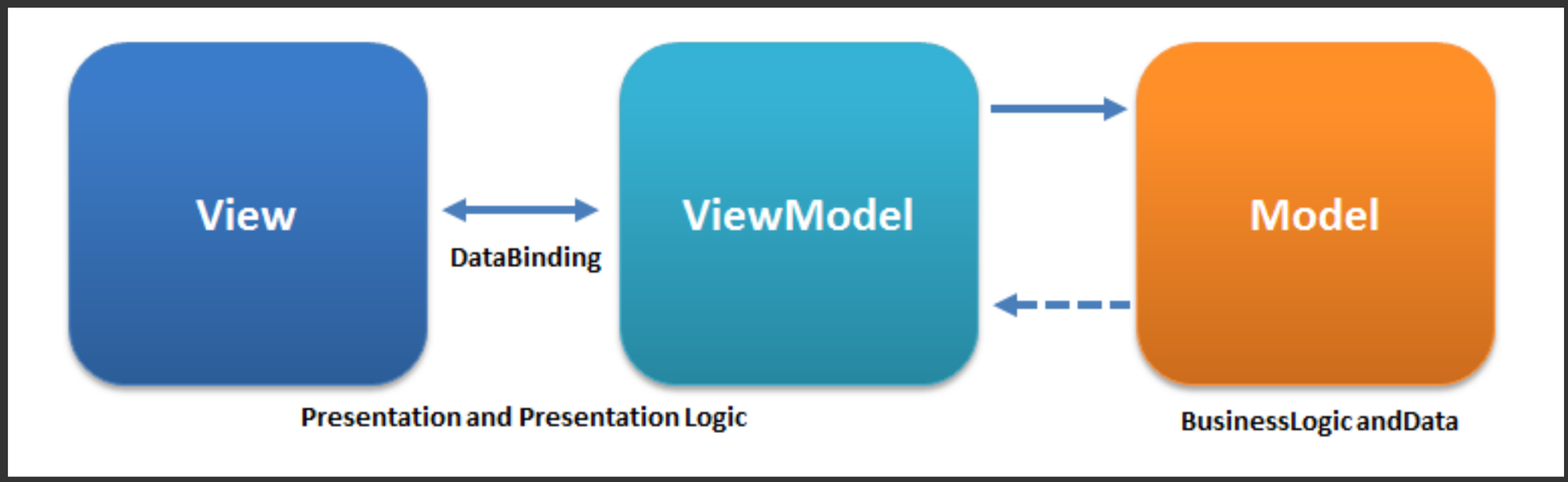
- AndroidでMVVMを実現するための実装方法
- 主にModel-View-ViewModelがそれぞれ何をやるのか話

- スライド内で出てくるコード
  - RxJava1
  - Retrolambda

MVVM

MVVM

- 元となったのはMicrosoftのWPFとSilverlight
- 目的はプレゼンテーションとドメインを分離すること
  - PresentationDomainSeparation
    - https://martinfowler.com/bliki/PresentationDomainSeparation.html
- テストしやすく、変更に強い
- DataBindingを使えば良いわけではない



https://ja.wikipedia.org/wiki/Model_View_ViewModel

MVVM参考

- The MVVM Pattern
  - https://msdn.microsoft.com/ja-jp/library/hh848246.aspx

- MVVMパターンの常識 --- 「M」「V」「VM」の役割とは？ ー @IT
  - http://www.atmarkit.co.jp/fdotnet/chushin/greatblogentry_02/greatblogentry_02_01.html

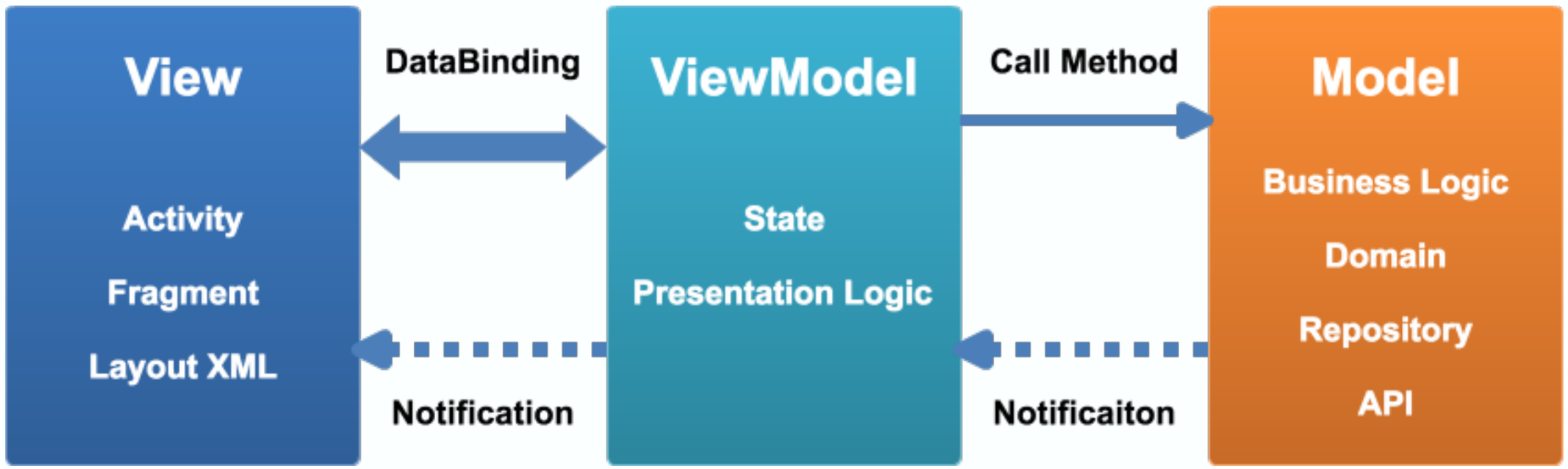
- GUIアーキテクチャパターンの基礎からMVVMパターンへ
  - https://www.slideboom.com/presentations/591514/GUI%E3%82%A2%E3%83%BC%E3%82%AD%E3%83%86%E3%82%AF%E3%83%81%E3%83%A3

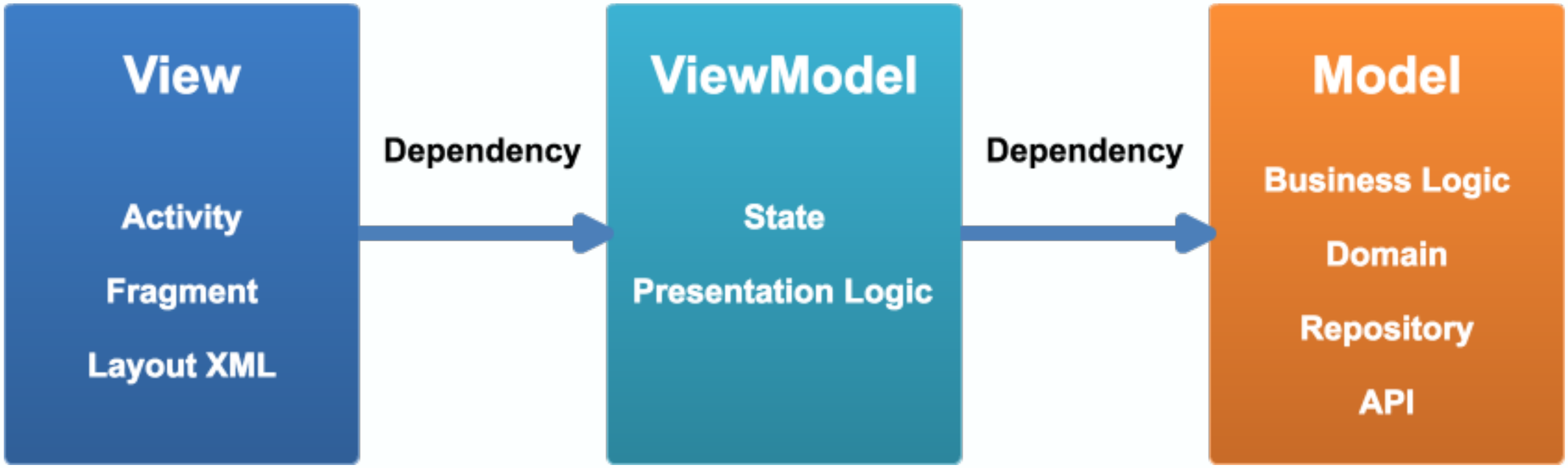
- MVVMのModelにまつわる誤解 - the sea of fertility
  - http://ugaya40.hateblo.jp/entry/model-mistake

Android で MVVM

AndroidでMVVM

- DataBindingによって実現可能に
- RxJavaやEventBus等のサポートが必要


View
Activity
Fragment
Layout XML
DataBinding
Notification
ViewModel
State
Presentation Logic
Call Method
Notificaiton
Model
Business Logic
Domain
Repository
API


View
Activity
Fragment
Layout XML
Dependency
ViewModel
State
Presentation Logic
Dependency
Model
Business Logic
Domain
Repository
API

View

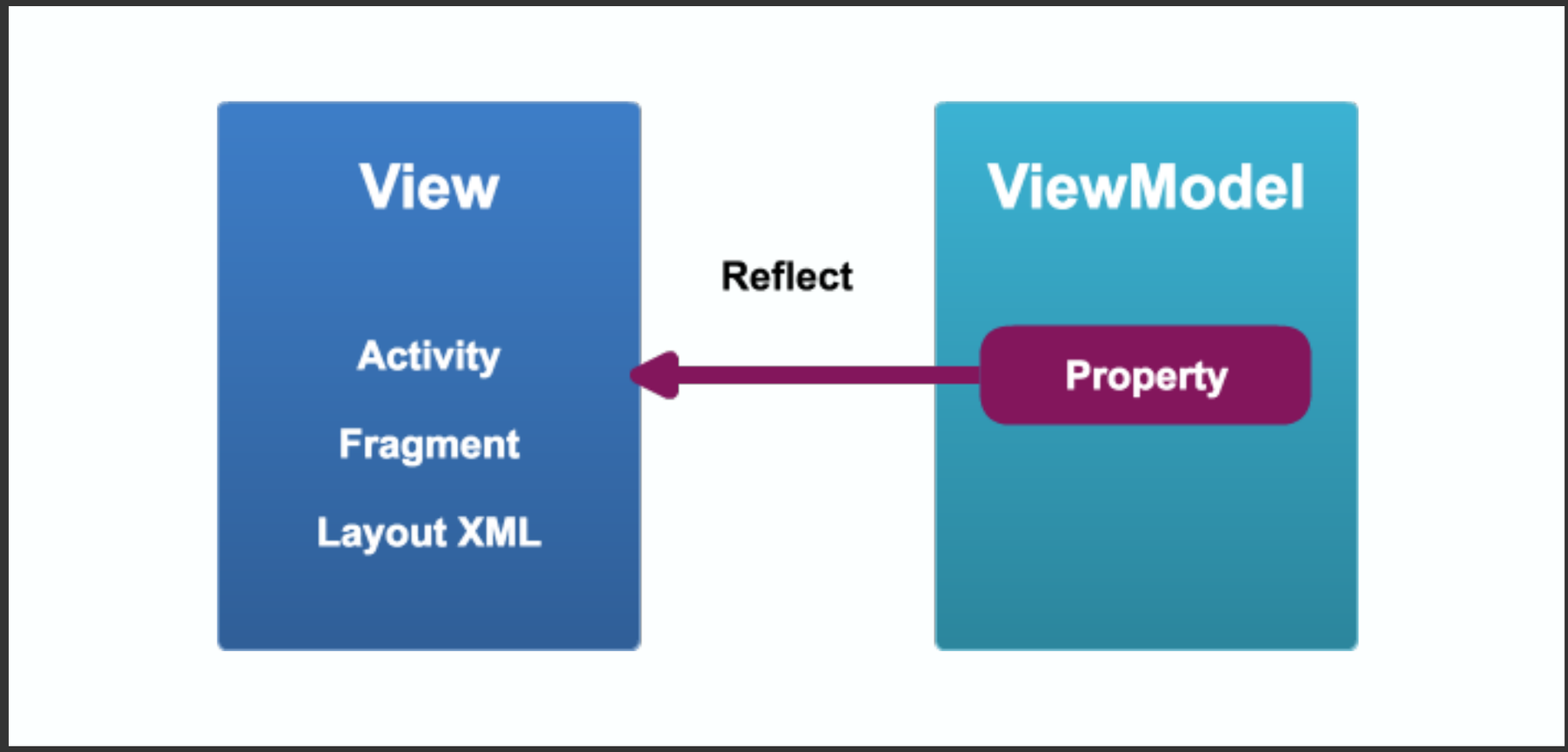
View

- ViewModelの状態を反映
- Viewの入力をViewModelへ伝える
- ViewModelにイベント処理を委譲

View

- ViewModelの状態を反映
- Viewの入力をViewModelへ伝える
- ViewModelにイベント処理を委譲

ViewModelの状態を反映



- ViewModelで公開されてる状態をDataBindingを利用して表示する

ViewModelの状態を反映

```
// ViewModel
public class ViewModel {

    public final ObservableField<String> title =
            new ObservableField<>();

}
```

ViewModelの状態を反映

```
<!-- レイアウトXML -->
<TextView
    android:id="@+id/text_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    android:text="@{viewModel.title}" />
```

ViewModelの状態を反映

- 標準のDataBindingでは難しいもの
- 例えば、画像URLからPicassoやGlideを使って画像を表示する場合など
- DataBindingのカスタムセッターを作る

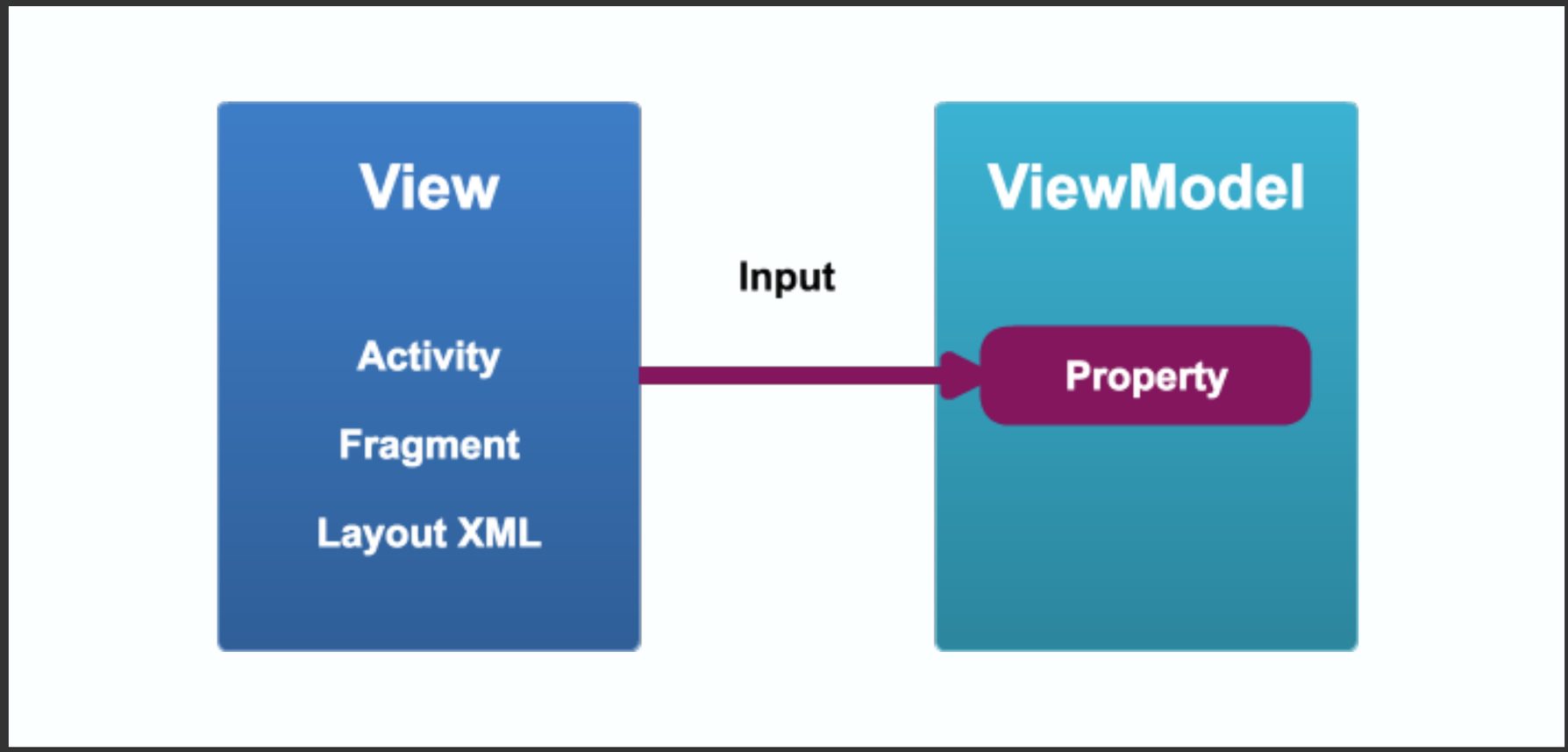
ViewModelの状態を反映

```
// カスタムセッター定義
public class ImageViewBinding {

    @BindingAdapter("imageFromURL")
    public static void loadImage(ImageView view, String url) {
        Glide.with(view.getContext()).load(url).into(view);
    }
}

<!-- レイアウトXML -->
<ImageView
    android:layout_width="match_parent"
    android:layout_height= “match_parent"

    app:imageFromURL="@{viewModel.imageURL}" />
```

View

- ViewModelの状態を反映
- Viewの入力をViewModelへ伝える
- ViewModelにイベント処理を委譲

Viewの入力をViewModelへ伝える



- Viewの入力をDataBindingを使用して、ViewModelの状態へ反映する
- 双方向バインディング

Viewの入力をViewModelへ伝える

```
<!-- レイアウトXML -->
<EditText
    android:id="@+id/edit_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"

    android:text="@={viewModel.title}" />
```

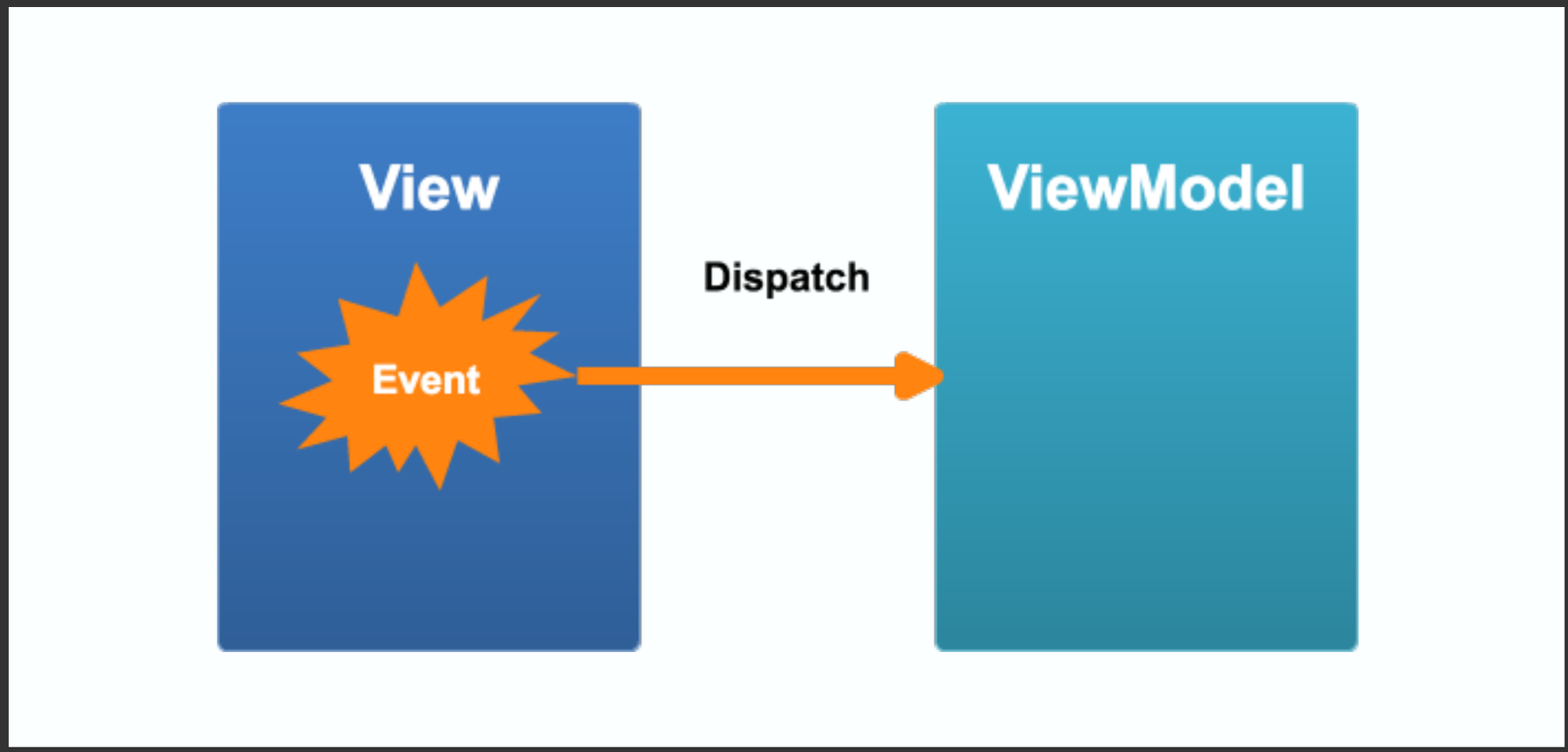
Viewの入力をViewModelへ伝える

```
// ViewModel
public class ViewModel {

    public final ObservableField<String> title =
            new ObservableField<>();

}
```

View

- ViewModelの状態を反映
- Viewの入力をViewModelへ伝える
- ViewModelにイベント処理を委譲

ViewModelにイベント処理を委譲



- Viewで発生したイベントをViewModelへ処理を委譲
- DataBindingやメソッド呼び出し

ViewModelにイベント処理を委譲

```
<!-- レイアウトXML -->
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="@dimen/fab_margin"
    android:onClick="@{viewModel::onClickDone}"
    android:src="@drawable/ic_done" />
```

ViewModelにイベント処理を委譲

```
// ViewModel
public class ViewModel {

    public void onClickDone(View view) {
        // ...
    }
}
```

ViewModelにイベント処理を委譲

- android:onClick以外のイベントを処理したい場合
- ドキュメントには載っていないがいくつかは定義されてる

https://android.googlesource.com/platform/frameworks/data-binding/+/android-7.1.1_r13/extensions/baseAdapters/src/main/java/android/databinding/adapters

- 例えば、EditTextが変更されるたびにイベントを処理する

ViewModelにイベント処理を委譲

```
<!-- レイアウトXML -->

<!--suppress AndroidUnknownAttribute -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onTextChanged="@{viewModel::onTextChanged}" />
```

ViewModelにイベント処理を委譲

```
// ViewModel
public class ViewModel {

    public void onTextChanged(CharSequence s,
                              int start,
                              int before,
                              int count) {
        // ...
    }

}
```

https://android.googlesource.com/platform/frameworks/data-binding/+/android-7.1.1_r13/extensions/baseAdapters/src/main/java/android/databinding/adapters/TextViewBindingAdapter.java#344

ViewModelにイベント処理を委譲

- どこにも定義されてないイベントを処理したい場合
- setterがあるものはそのまま使える
- 例えば、

SwipeRefreshLayout.setOnRefreshListener

ViewModelにイベント処理を委譲

```
// ViewModel
public class ViewModel {

    public void onRefresh() {
        // ...
    }
}
```

```
<!-- レイアウトXML -->
<android.support.v4.widget.SwipeRefreshLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:onRefreshListener="@{viewModel::onRefresh}">
```

ViewModelにイベント処理を委譲

- DataBidingが使えないパターン
- 例えばメニュー処理
- 直接ViewModelを呼ぶ

ViewModelにイベント処理を委譲

```
// Activity or Fragment
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_action:
            viewModel.someAction();
            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

ViewModelにイベント処理を委譲

```
// ViewModel
public class ViewModel {

    void someAction() {
        // ...
    }
}
```

ViewModel

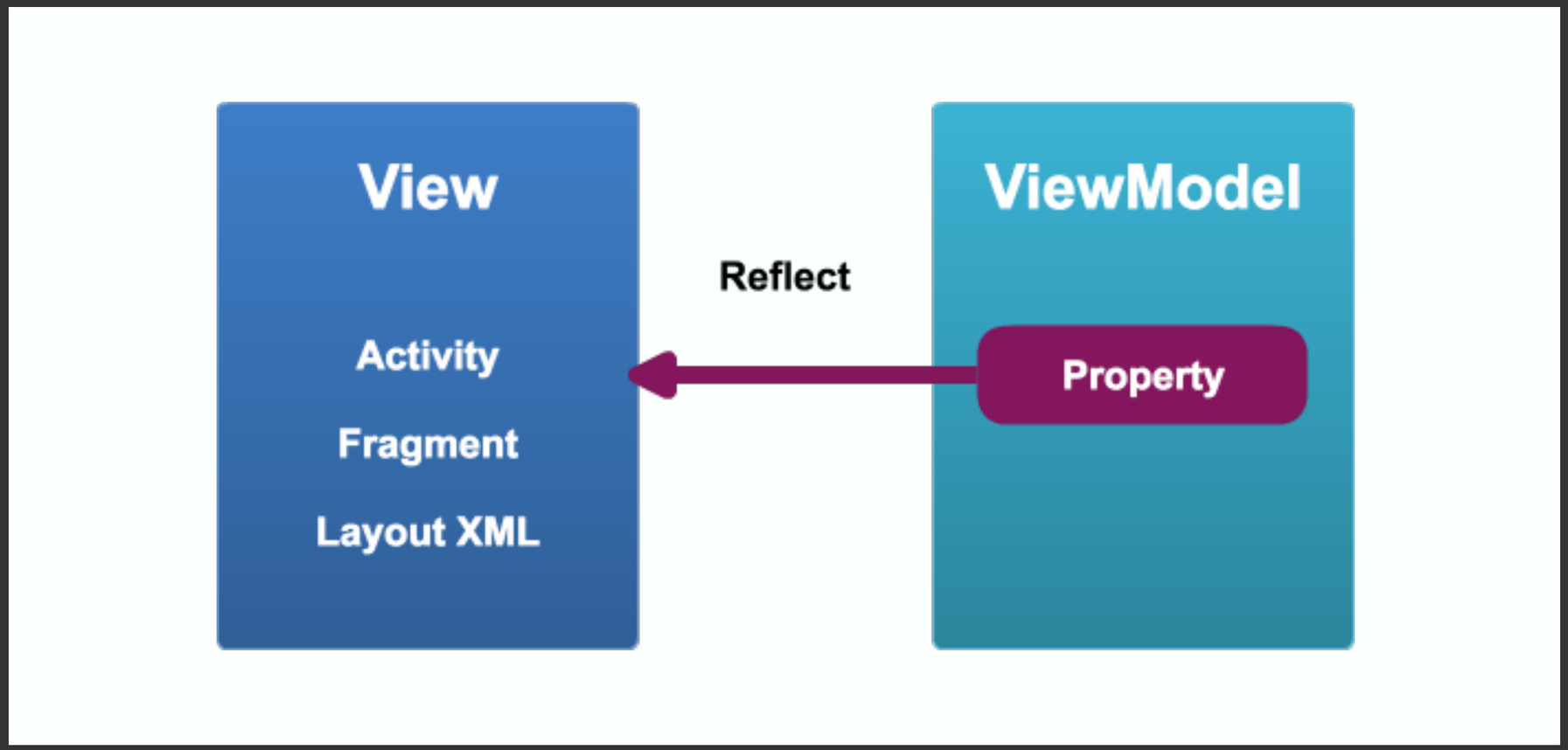
ViewModel

- Viewのための状態保持と公開
- Modelの処理呼び出し
- Viewへの変更通知イベント

ViewModel

- Viewのための状態保持と公開
- Modelの処理呼び出し
- Viewへの変更通知イベント

Viewのための状態保持と公開



- Viewで表示するための状態を保持し、公開する
- ObservableFieldやgetterなど

Viewのための状態保持と公開

```
// ViewModel
public class ViewModel {

    public final ObservableField<String> title =
            new ObservableField<>();

}
```

Viewのための状態保持と公開

```
// ViewModel
public class ViewModel extends BaseObservable {

    private String title;

    @Bindable
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
        notifyPropertyChanged(BR.title);
    }
}
```

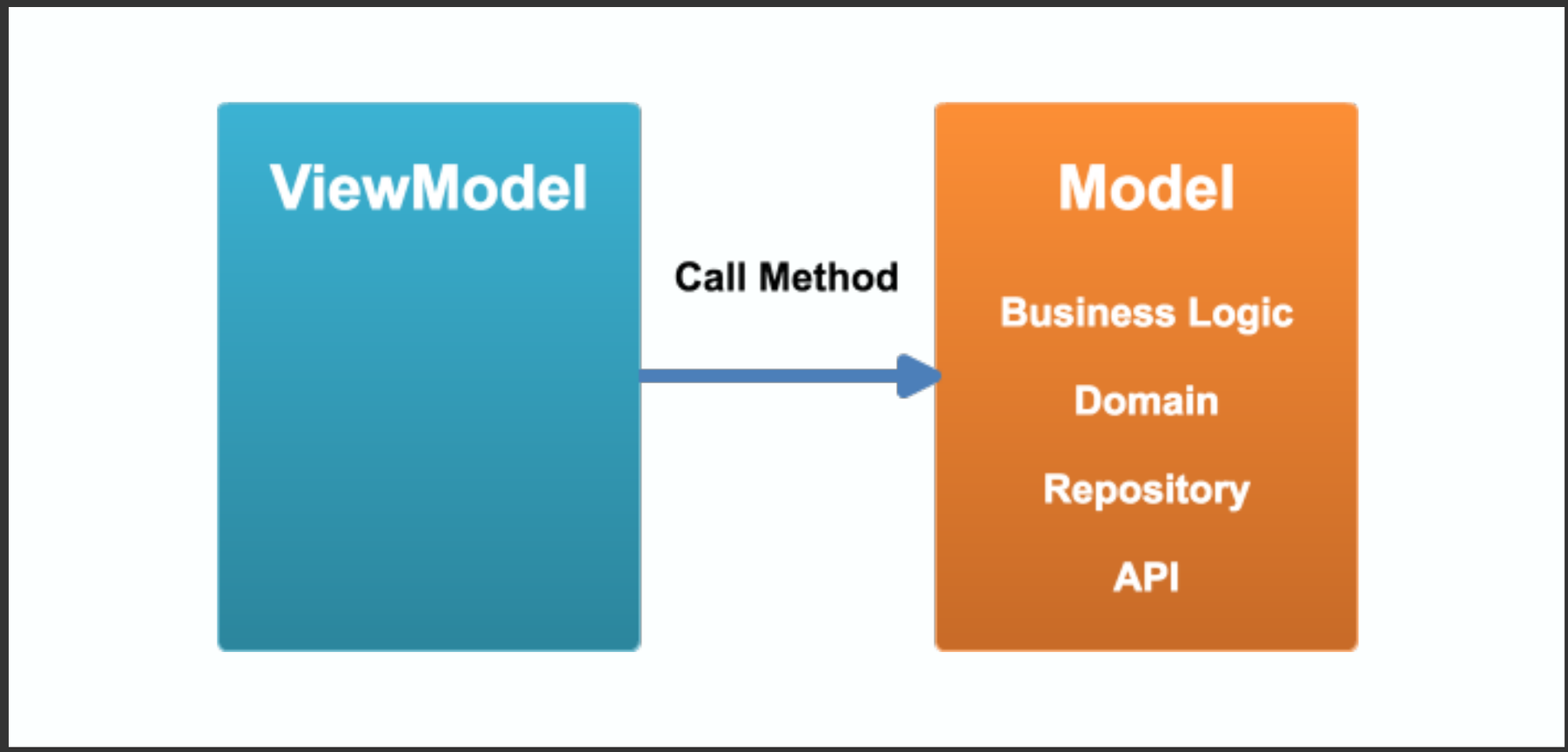
Viewのための状態保持と公開

```
<!-- レイアウトXML -->
<TextView
    android:id="@+id/text_title"
    android:layout_width="wrap_content"
    android:layout_height= “wrap_content"
    android:text="@{viewModel.title}" />
```

ViewModel

- Viewのための状態保持と公開
- Modelの処理呼び出し
- Viewへの変更通知イベント

Modelの処理呼び出し



- Viewからのイベントなどを受けてModelの処理を呼び出す

Modelの処理呼び出し

- MVVMのModelにまつわる誤解
  - http://ugaya40.hateblo.jp/entry/model-mistake
- ModelについてViewModelが行うことは、イベントに対する反応と**戻り値のないメソッド**の呼び出ししかない事

ViewModelに対するModelのインターフェース

それを踏まえて考えれば、ViewModelに公開するModelのインタフェースは以下の二つしかありません。

- Modelのステートの公開とその変更通知
- **Modelの操作のための戻り値のないメソッド**

Modelの処理呼び出し

```
// ViewModel
public class ViewModel {

    public void onClickAction(View view) {
        model.someOperation();
    }

}
```

Modelの処理呼び出し

- 処理結果などはどう受け取るのか？
- Modelのとこで解説します

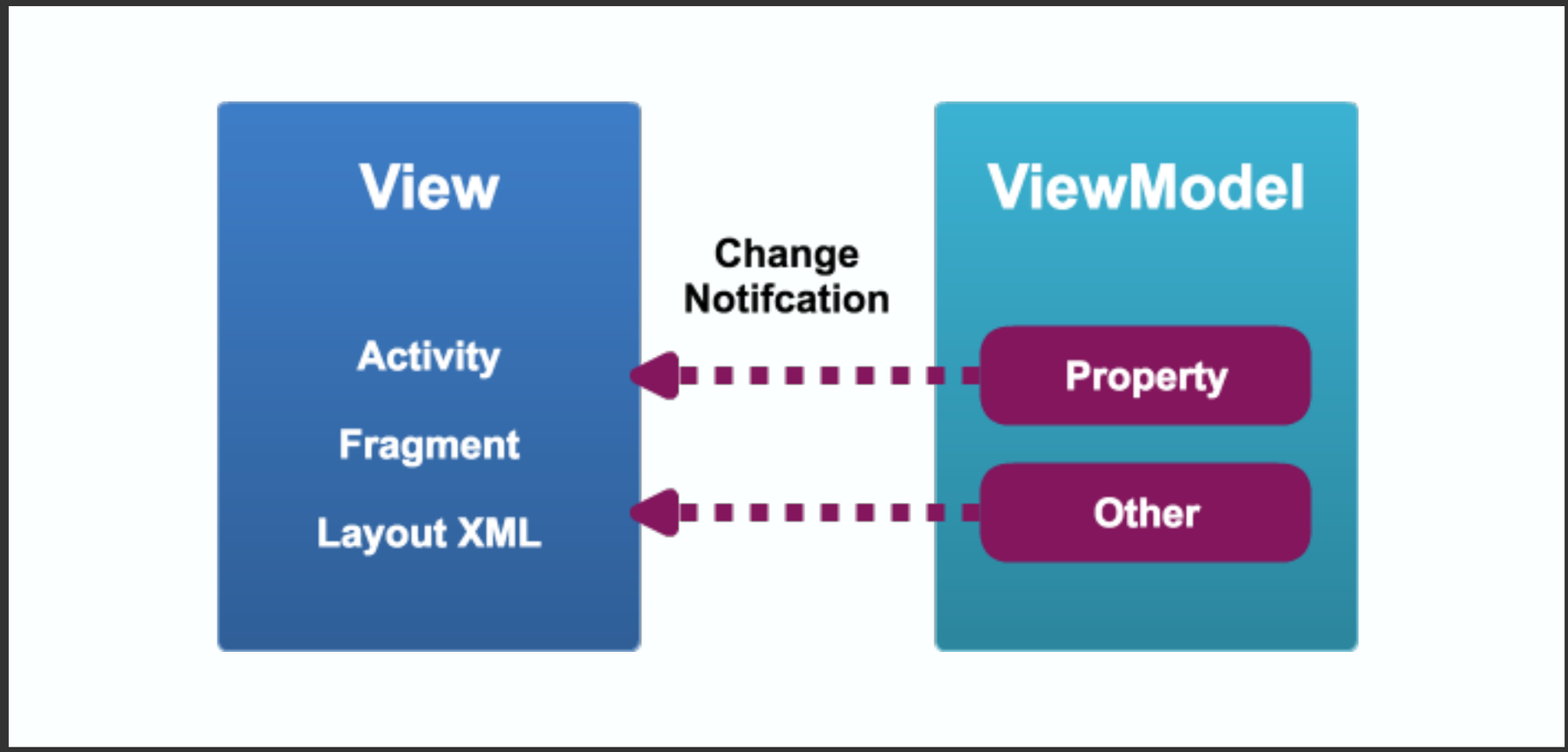
ViewModel

- Viewのための状態保持と公開
- Modelの処理呼び出し
- Viewへの変更通知イベント

Viewへの変更通知イベント

- Viewを変更するためにViewModelの状態を変更させて、それをViewへ伝える必要がある

Viewへの変更通知イベント



- ViewModelの変更をViewへ伝える
- ObservableFieldやBaseObservable
- RxJavaやEventBus

Viewへの変更通知イベント

- DataBindingを使う場合

Viewへの変更通知イベント

```
// ViewModel
public class ViewModel {

    public final ObservableField<String> title =
            new ObservableField<>();

    public void something(String result) {
        // 変更通知
        title.set(result);
    }
}
```

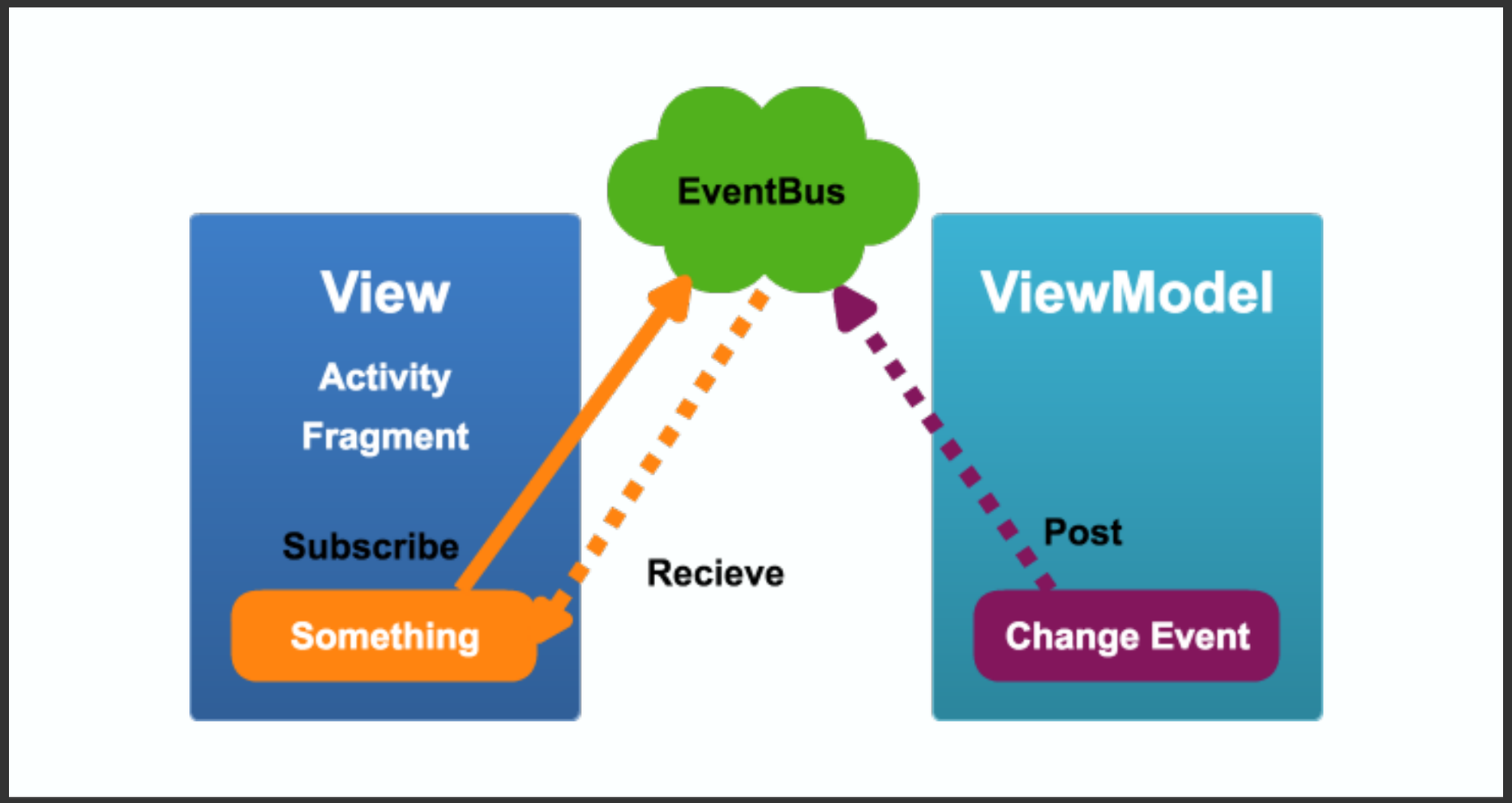
Viewへの変更通知イベント

```
// ViewModel
public class ViewModel extends BaseObservable {
    private String title;
    @Bindable
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;

        // 変更通知
        notifyPropertyChanged(BR.title);
    }
}
```

Viewへの変更通知イベント

- DataBidingが使えないパターン
- ダイアログ表示や画面遷移
- EventBus or RxJava で対応

Viewへの変更通知イベント



EventBus

https://github.com/greenrobot/EventBus

Viewへの変更通知イベント

```
// EventClass
public class ShowDialogEvent {
    private final String message;

    public ShowDialogEvent(String message) {
        this.message = message;
    }
}
```

Viewへの変更通知イベント

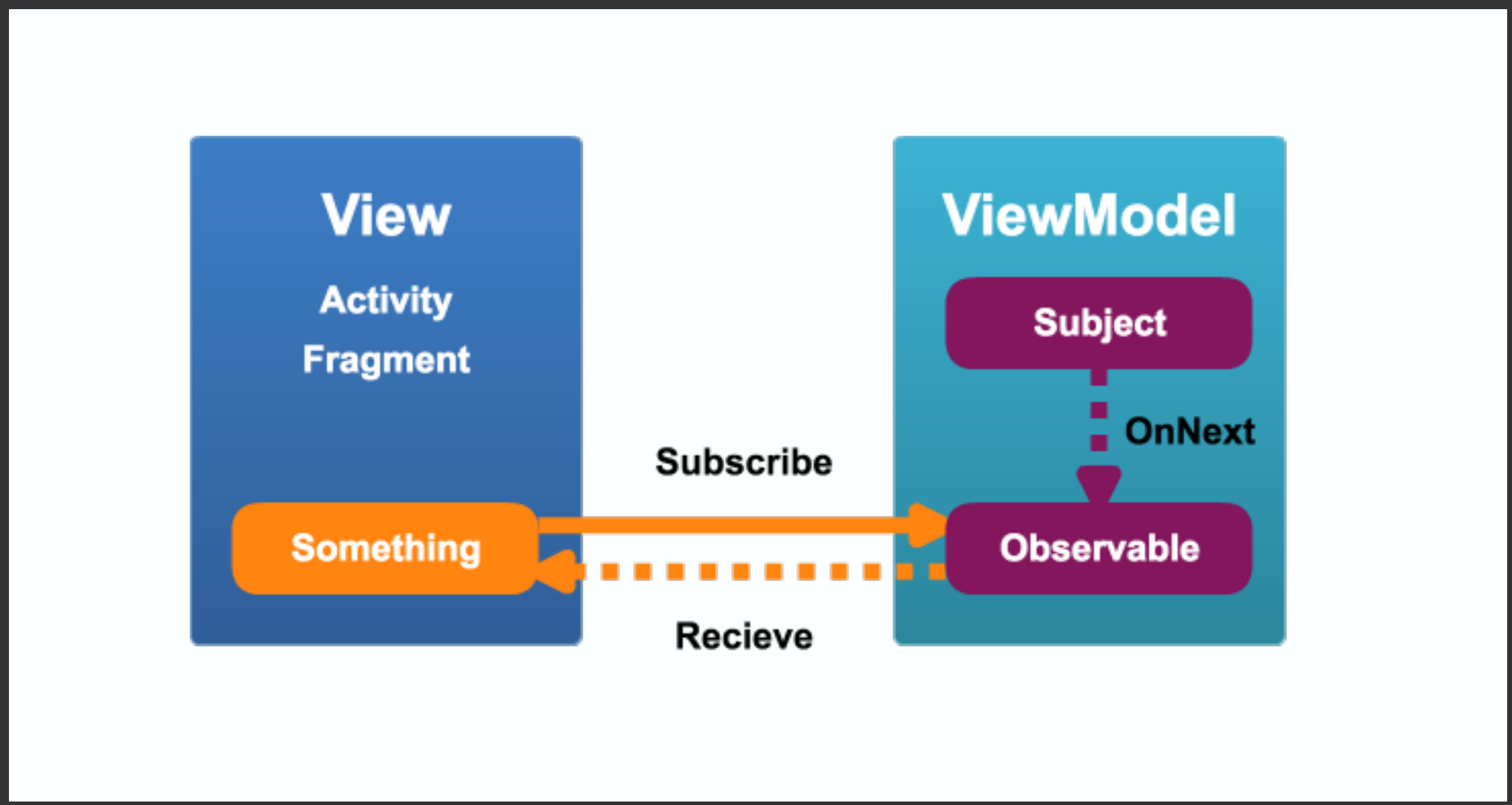
```
// ViewModel
public class ViewModel {
    public void something() {
        EventBus.getDefault().post(
                new ShowDialogEvent("message")
        );
    }
}
```

Viewへの変更通知イベント

```
// View
@Override
protected void onStart() {
    super.onStart();
    EventBus.getDefault().register(this);
}
```

```
@Subscribe(threadMode = ThreadMode.MAIN)
public void showDialog(ShowDialogEvent event) {

    // ダイアログを表示する

}
```

Viewへの変更通知イベント



RxJava

https://github.com/ReactiveX/RxJava

Viewへの変更通知イベント

```
// ViewModel
public class ViewModel {

    private final PublishSubject<String> showDialogSubject
        = PublishSubject.create();

    final Observable<String> showDialog
        = showDialogSubject.asObservable();

    public void something() {
        // 通知イベントを発行
        showDialogSubject.onNext("message");
    }
}
```

Viewへの変更通知イベント

```
// View
private void subscribe() {

    viewModel.showDialog.subscribe(message -> {

        // ダイアログを表示する

    });

}
```

補足: RxJava - Subject

- Observerにもなるし、Observableにもなる
- ObserverとしてonNextなどを呼べる
- Observableとしてsubscribeできる
- これを通知に利用する
- よく使うのはPublishSubject
- 説明難しいので実際に試すと早いです

補足: RxJava - asObservable

- Subjectをそのまま公開しない
- そのまま公開してしまうと、他の箇所からもonNextを呼べてしまう
- Observableと公開する時もasObservableを使う

```
private final PublishSubject<String> subject
    = PublishSubject.create();

final Observable<String> observable
    = subject.asObservable();
```

補足: EventBus vs RxJava

- 好きな方を使えばいいと思う
- 個人的にはRxJava
  - retrolambdaがあるといいかも
- EventBusはグローバルになるのでどこから通知が来るのかが分かりにくい
- EventClassが増えすぎていく
- ただし、限定的にEventBus使用
  - RecyclerViewのAdapterからActivityへの通知

  など

Model

Model

- ViewとViewModel以外の処理
- ViewModelへの変更通知イベント

Model

- ViewとViewModel以外の処理
- ViewModelへの変更通知イベント

ViewとViewModel以外の処理

- ドメイン、ビジネスロジック
- DB
- API
- その他色々

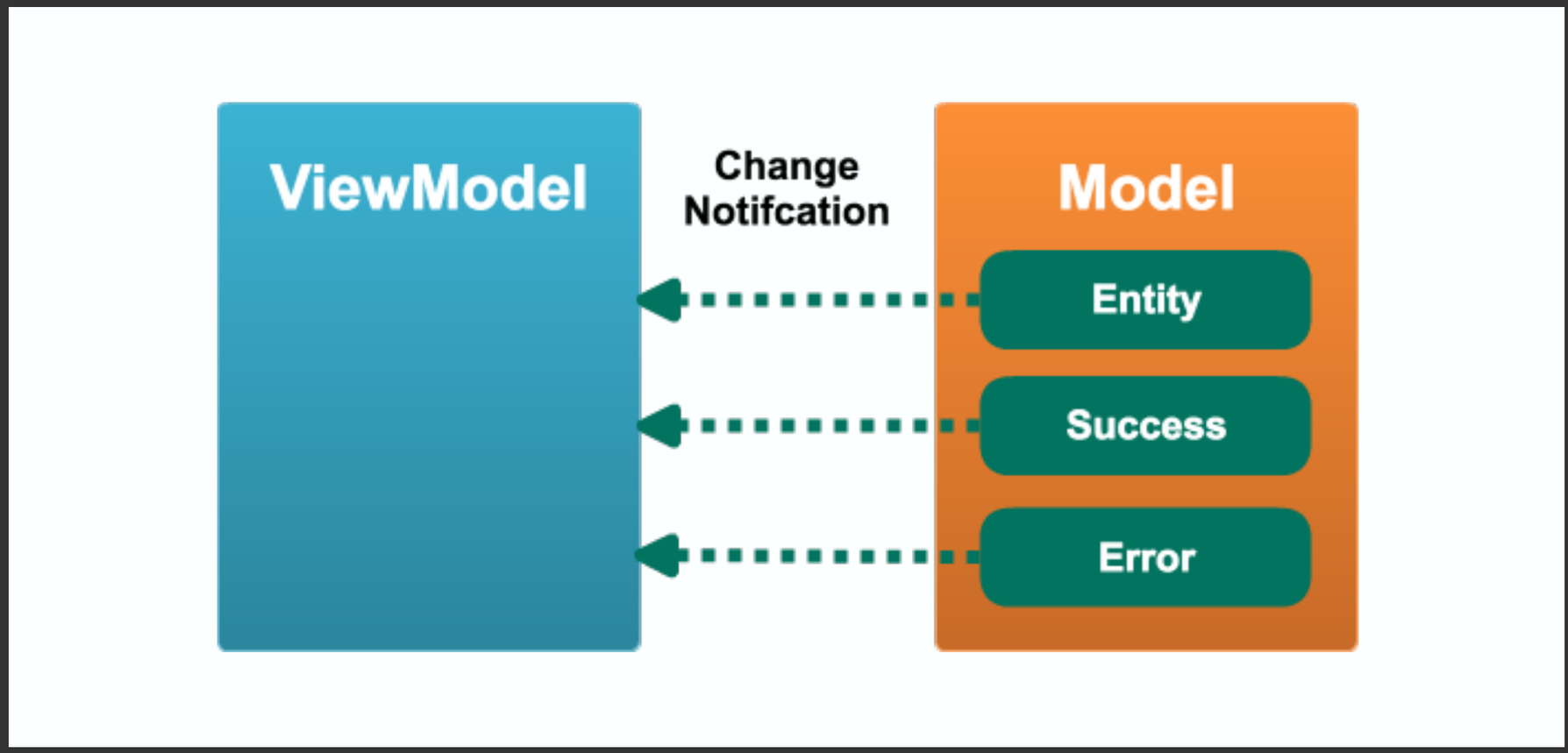
Model

- ViewとViewModel以外の処理
- ViewModelへの変更通知イベント

ViewModelへの変更通知イベント

- ViewModelで話した戻り値のないメソッドを呼んで、結果を受け取る方法
- Modelに対する操作はModelの状態を変更させること
- Modelが変更されたら変更通知イベント発行する
- ViewModelはそのイベントを受け取るだけ
- 例外処理などもModelで処理して通知イベントを発行するだけ
- 必要があればModelでBaseObservableを使用

ViewModelへの変更通知イベント



- Modelからは通知を使ってViewModelへ変更を伝える
- ViewModelは通知を受け取るだけ

ViewModelへの変更通知イベント

・Repositoryから非同期でEntityを取得する

ViewModelへの変更通知イベント

```
// Model

// 成功
private final PublishSubject<Entity> entitySubject
        = PublishSubject.create();
public final Observable<Entity> entity
        = entitySubject.asObservable();

// 失敗
private final PublishSubject<Void> errorSubject
        = PublishSubject.create();
public final Observable<Void> error
        = errorSubject.asObservable();
```

ViewModelへの変更通知イベント

```
repository.get()
        .subscribeOn(Schedulers.newThread())
        .unsubscribeOn(AndroidSchedulers.mainThread())
        .subscribe(new Observer<Entity>() {
            @Override
            public void onCompleted() {
            }

            @Override
            public void onError(Throwable e) {
                // 失敗通知イベント
                errorSubject.onNext(null);
            }

            @Override
            public void onNext(Entity entity) {
                // 成功通知イベント
                entitySubject.onNext(entity);
            }
    });
```

ViewModelへの変更通知イベント

```
// ViewModel

model.entity.subscribe(entity -> {
    // 成功
});

model.error.subscribe(aVoid -> {
    // 失敗
});
```

ViewModelへの変更通知イベント

- EntityをDataBindingでViewとバインドしてる場合
- Entityのプロパティの変更を通知する

ViewModelへの変更通知イベント

```
// ViewModel
public class ViewModel {
    public final ObservableField<Entity> entity =
            new ObservableField<>();
}
```

```
<!-- レイアウトXML -->
<TextView
    android:id="@+id/text_name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@{viewModel.entity.name}" />
```

ViewModelへの変更通知イベント

```
public class Entity extends BaseObservable {
    private String name;

    @Bindable
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;

        // 値がセットされたら通知
        notifyPropertyChanged(BR.name);
    }

    // 何か操作してModelの状態を変更
    public void someOperation() {
        // …
        setName(value);
    }
}
```

まとめ

まとめ

- DataBindingを使うだけではダメ
- プレゼンテーションとドメインの分離をしっかり意識する
- 各レイヤーでやること、レイヤー間の対話の方法を理解する
- EventBusやRxJavaなどを活用する

良いとこ

- 各レイヤーの責務が明確になるので、やることが把握しやすい、変更に強い
- DataBindingによってView側のコードが減り、複雑にならない
- ModelがViewから切り離されてるのでテストしやすい

ツライとこ

- View側のコードが減るけど、それでもライフサイクルとか色々ツライ
- ObservableField等のプロパティが増えていく
- EventBusのEventクラスが増える、register/unregister管理
- RxJavaのSubjectやObservableが増える、subscribe/unsubscribe管理

サンプル

https://github.com/STAR-ZERO/AndroidMVVM

宣伝

- 弊社メンバー募集中です
- Android
- iOS
- Scala
- JavaScript

ありがとうございました