ME GIULIO

- iOS and Android Developer
- Full-time "Rheinfabrikant"

MVVM

MODEL VIEW VIEWMODEL

MVC

(MASSIVE) VIEW CONTROLLER

PROBLEMS!

- Large Fragments/Activities
  - Mixed concerns
  - Hard to test

SOLUTIONS?

MODEL VIEW PRESENTER

- Separates data/presentation logic from view
  - Nice open source project: Mosby

WWW.HANNESDORFMANN.COM/ANDROID/MOSBY/

```
public class CountriesPresenter extends MvpBasePresenter<CountriesView> {

}
```

```
public class CountriesPresenter extends MvpBasePresenter<CountriesView> {

  @Override
  public void loadCountries(final boolean pullToRefresh) {
    getView().showLoading(pullToRefresh);









  }
}
```

```
public class CountriesPresenter extends MvpBasePresenter<CountriesView> {

  @Override
  public void loadCountries(final boolean pullToRefresh) {
    getView().showLoading(pullToRefresh);

    new CountriesAsyncLoader(
        new CountriesAsyncLoader.CountriesLoaderListener() {
          @Override public void onSuccess(List<Country> countries) {




          }

          @Override public void onError(Exception e) {
          }
        }).execute();
  }
}
```

```
public class CountriesPresenter extends MvpBasePresenter<CountriesView> {

  @Override
  public void loadCountries(final boolean pullToRefresh) {
    getView().showLoading(pullToRefresh);

    new CountriesAsyncLoader(
        new CountriesAsyncLoader.CountriesLoaderListener() {
          @Override public void onSuccess(List<Country> countries) {
            if (isViewAttached()) {
              getView().setData(countries);
              getView().showContent();
            }
          }

          @Override public void onError(Exception e) {
          }
        }).execute();
  }
}
```

PROBLEMS!

- The presenter knows the type of the view and has a strong reference to it!
- The presenter contains view logic e.g. checking whether the view is visible etc.

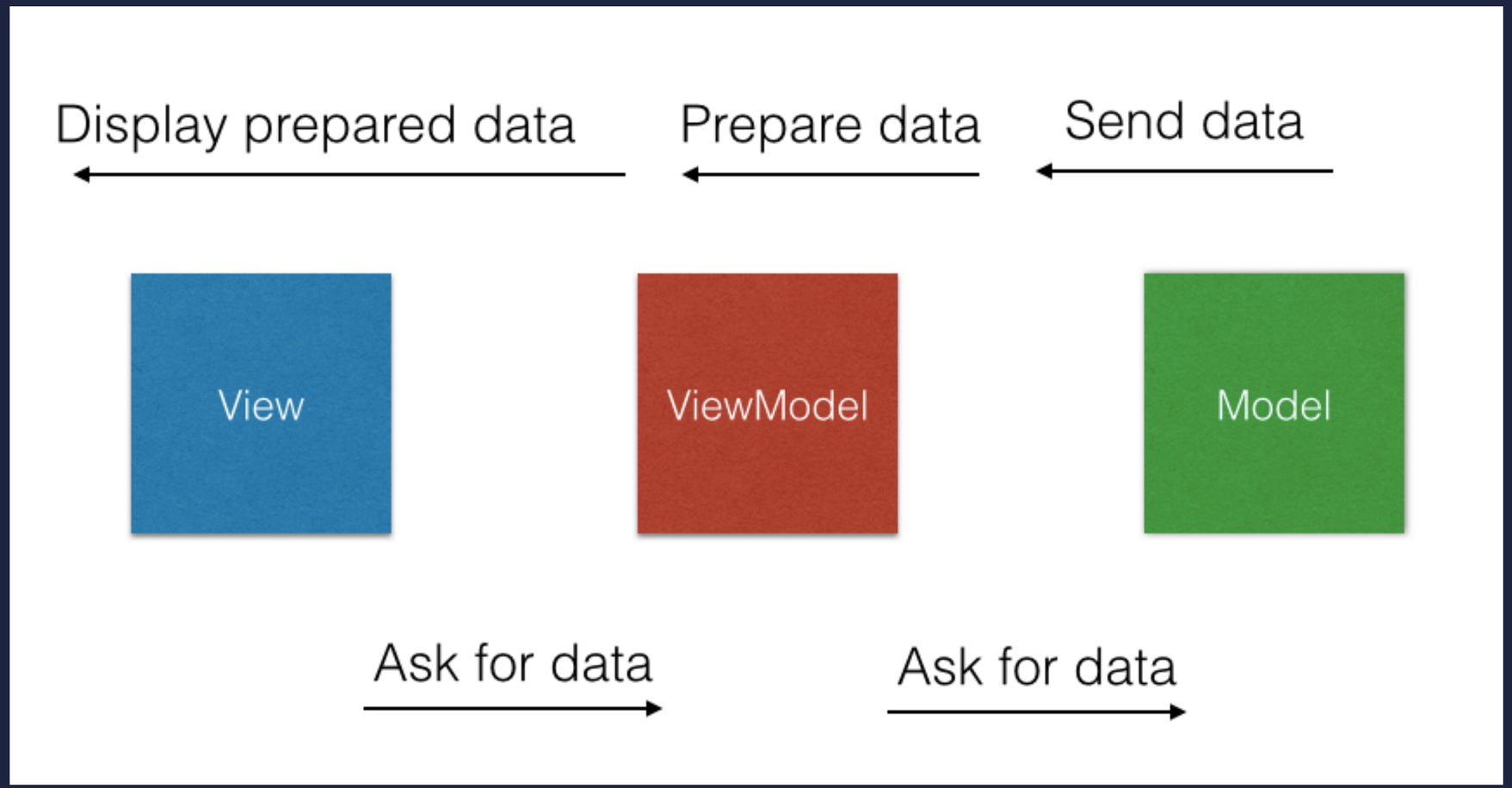
LEADS TO PROBLEMS WITH THE ANDROID VIEW LIFECYCLE

MVVM

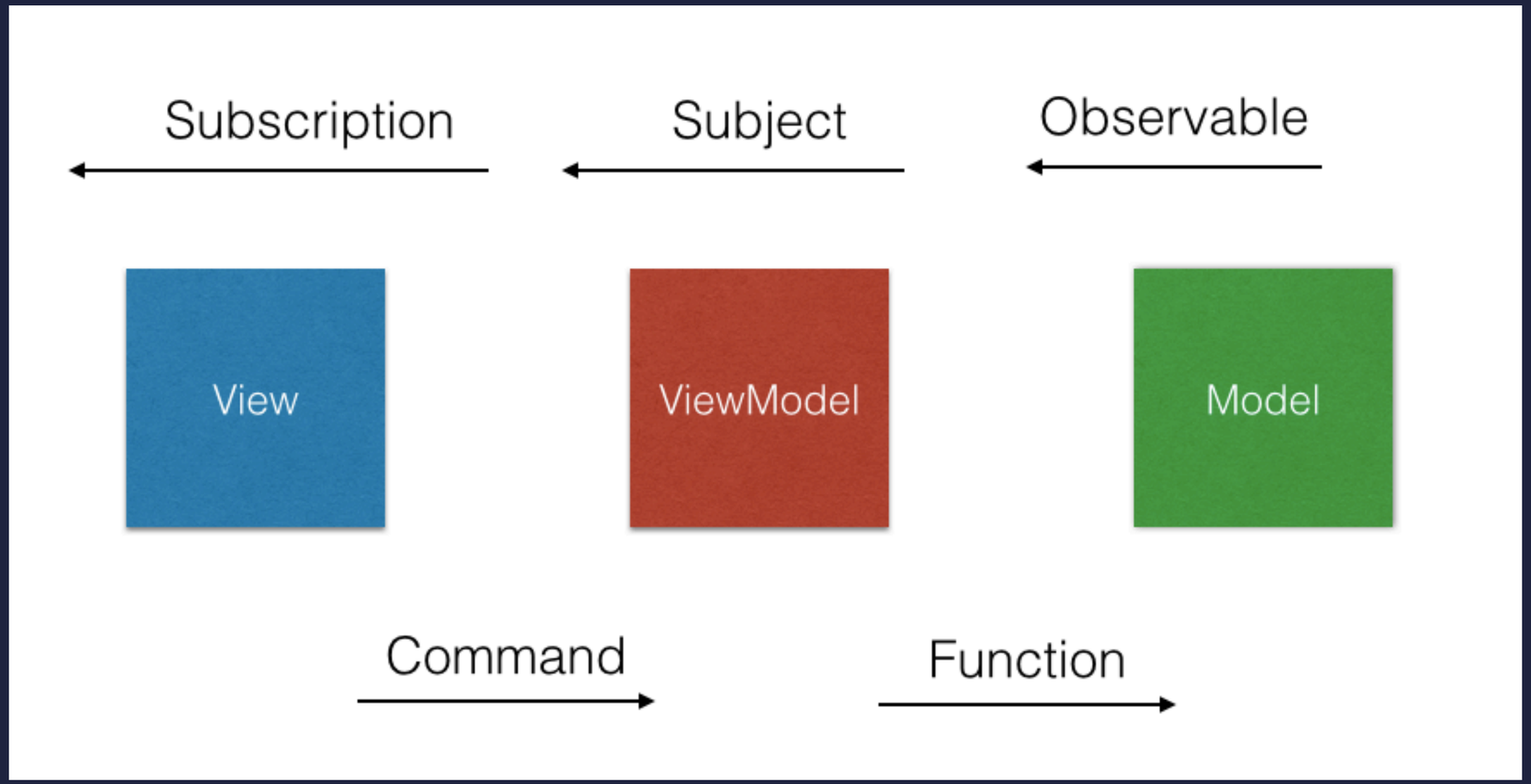
MVVM

- Also separates data/presentation logic from view
- Similar to MVP but: *The ViewModel has no reference to the view*

THE VIEW HAS NO KNOWLEDGE ABOUT THE MODEL!



COMMUNICATION BETWEEN COMPONENTS



DEMO

VIEWMODELS

- SearchViewModel
- SearchResultViewModel
- DetailsViewModel

SEARCH RESULT VIEWMODEL

SUBJECTS - VIEWMODEL

```
public class SearchResultViewModel {

    // Emits the text that should be displayed
    public final BehaviorSubject<Spanned> textSubject = BehaviorSubject.create();

    // Emits the correct intent to open the details activity.
    public final PublishSubject<Intent> onOpenDetailsSubject = PublishSubject.create();

    // ...
}
```

COMMANDS - VIEWMODEL

```
public class SearchResultViewModel {

    // ...

    // Send item to this command to trigger changes to the text subject.
    public final PublishSubject<SearchResult> setSearchResultCommand = PublishSubject.create();

    // Send null to this command to request a details intent.
    public final PublishSubject<Void> openDetailsCommand = PublishSubject.create();

    // ...
}
```

BINDINGS #1 - VIEWMODEL

```
public SearchResultViewModel(Context context) {
    super()

    // Text
    setSearchResultCommand
        .map(searchResult -> {
            return buildHTML(searchResult); // e.g. Frozen (2012)
        })
        .subscribe(html -> {
            textSubject.onNext(html);
        });

     //...
}
```

BINDINGS #2 - VIEWMODEL

```
public SearchResultViewModel(Context context) {

 // ...

 // Details
 cachedSample(setSearchResultCommand, openDetailsCommand)
    .map(item -> IntentFactory.newDetailsActivityIntent(context, item))
    .subscribe(intent -> onOpenDetailsSubject.onNext(intent));
}
```

SUBSCRIPTIONS - VIEW

```
public void bind() {

    // Bind text
    bindView(itemView, mViewModel.textSubject)
        .subscribe(text -> mTextView.setText(text));

    // Clicks
    clicks(mCardView)
        .subscribe(_ -> mViewModel.openDetailsCommand.onNext(null));

    // Details
    bindView(itemView, mViewModel.onOpenDetailsSubject)
        .subscribe(intent -> mContext.startActivity(intent));
    }
```

TESTS

```
def "The SearchResultViewModel should build the correct text"() {

    given: "A search results item with a title and a year"
       SearchResult searchResult = new SearchResult("Title", "2015")

    and: "A SearchResultViewModel instance"
       SearchResultViewModel viewModel = new SearchResultViewModel(mContext)

    when: "I send the SearchResult item to view model"
       viewModel.setSearchResultCommand.onNext(searchResult)

    and: "I ask for the text"
        Spanned text = viewModel.textSubject.toBlocking().first()

    then: "It should match my expected text"
        text.toString() == "Title (2015)"
}
```

DEMO

WWW.GITHUB.COM/RHEINFABRIK/ANDROID-MVVM-EXAMPLE

WRAP UP

- Easy to test!
- Clear separation of concerns!

Keep your Views simple!

THANKS

SIDE NOTE - WE ARE HIRING ;)

WWW.RHEINFABRIK.DE/FABRIK/JOBS/

QUESTIONS?

@GILOTM