MVI

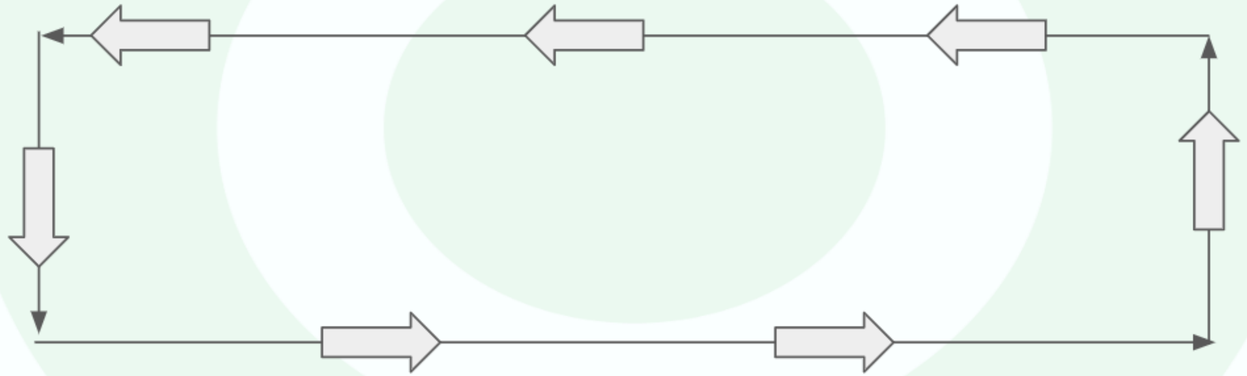
Model View Intent

Note (Intent = User Intention)

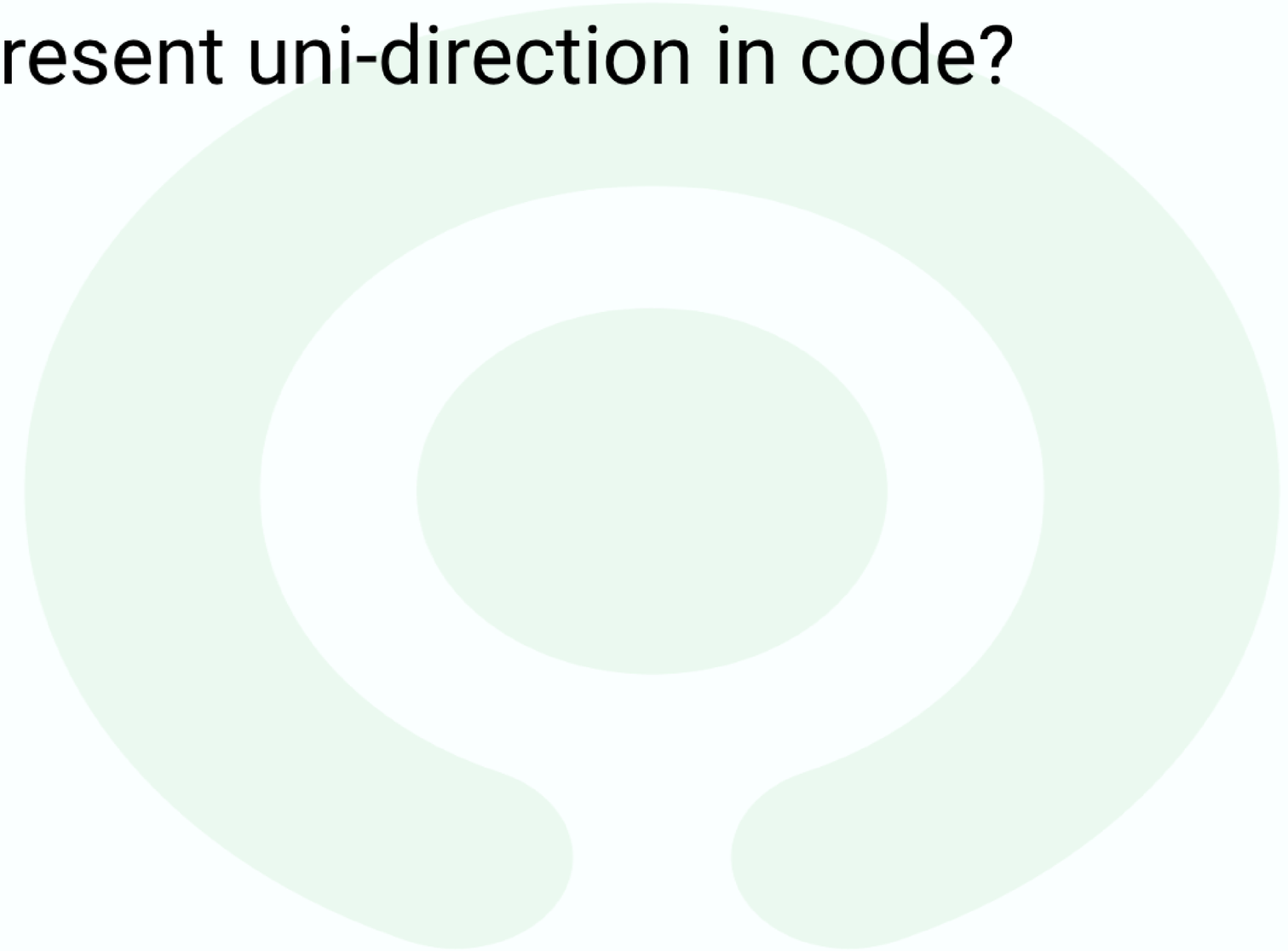
Goal of this talk

- What is MVI
- Why MVI
- Share knowledge in a way, so our team able to reduce learning curve time from two years to one week.

MVI is a unidirectional pattern (like Redux, cycle.js).



How represent uni-direction in code?



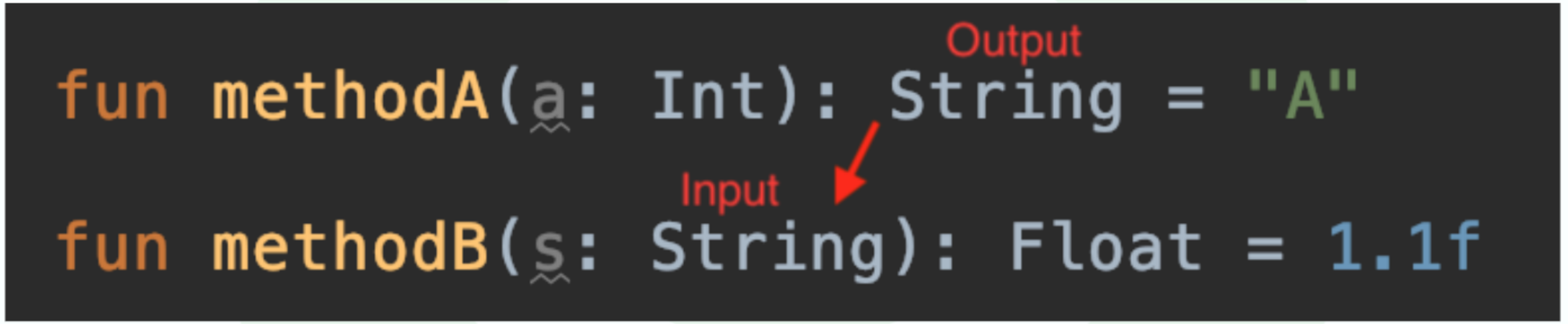
How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"
```

How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f
```

How represent uni-direction in code?



How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f

fun methodC(f: Float): Int = 1
```

How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f

fun methodC(f: Float): Int = 1

methodA( a: 3)
```

How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f

fun methodC(f: Float): Int = 1

methodB(methodA( a: 3))
```

How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f

fun methodC(f: Float): Int = 1

methodA(methodC(methodB(methodA( a: 3))))
```

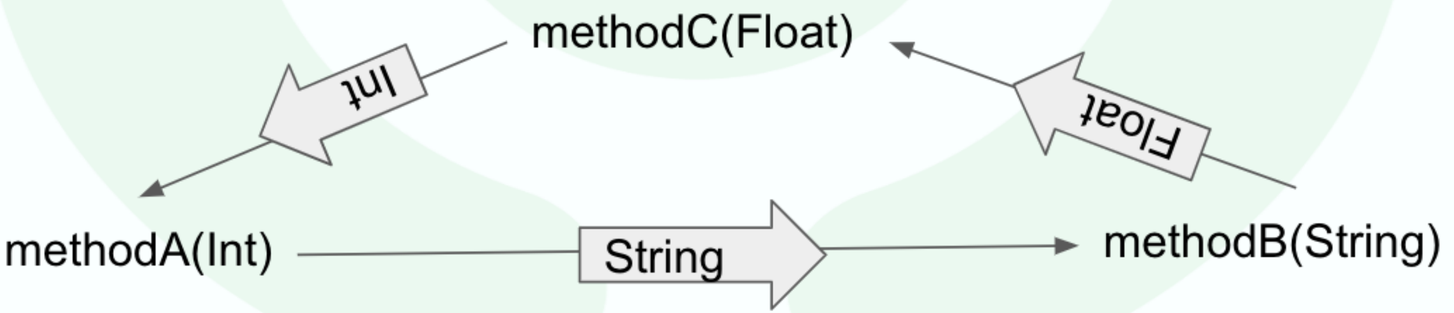
How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f

fun methodC(f: Float): Int = 1

methodB(methodA(methodC(methodB(methodA( a: 3)))))
```

Map code into drawing?

```
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f

fun methodC(f: Float): Int = 1

methodB(methodA(methodC(methodB(methodA( a: 3)))))
```

readability 😱



Tip to make our previous code readable using Kotlin

```kotlin
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f

fun methodC(f: Float): Int = 1

methodA( a: 3)
    .let(::methodB)
    .let(::methodC)
    .let(::methodA)
    .let(::methodB)
    .let(::methodC)
```

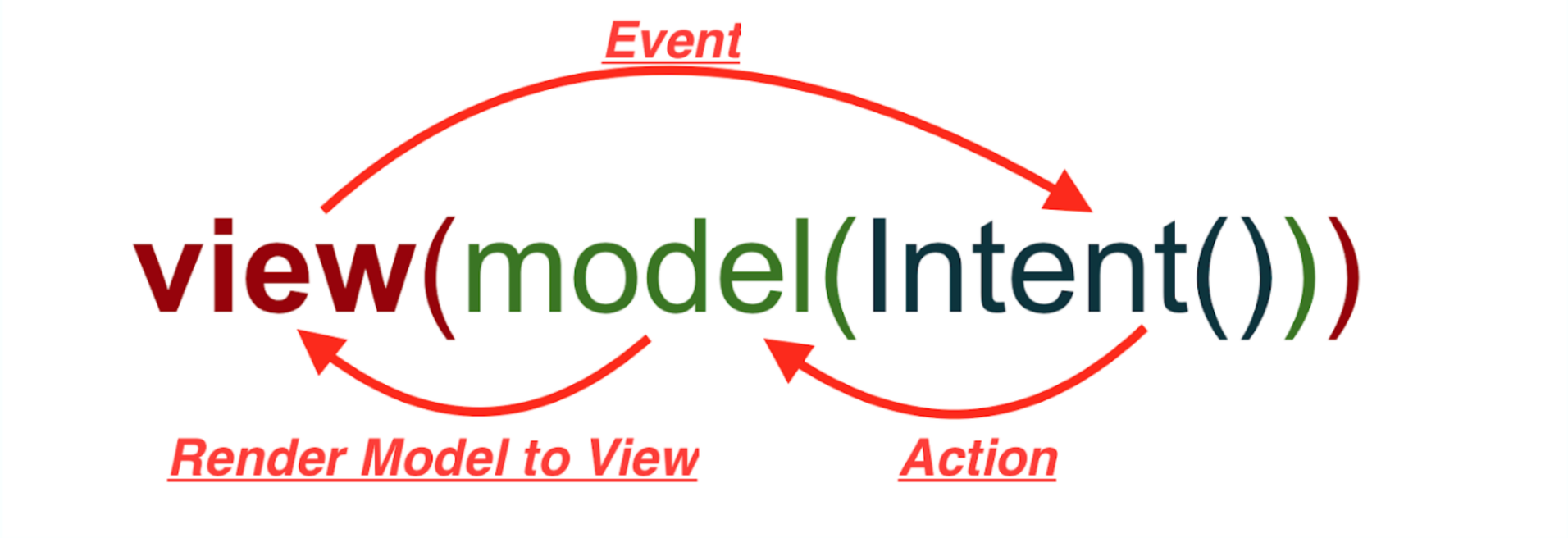
```kotlin
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f

fun methodC(f: Float): Int = 1

methodB(methodA(methodC(methodB(methodA( a: 3)))))
```

Until now, we know what uni-direction is and how to achieve uni-direction in code.

Back to MVI

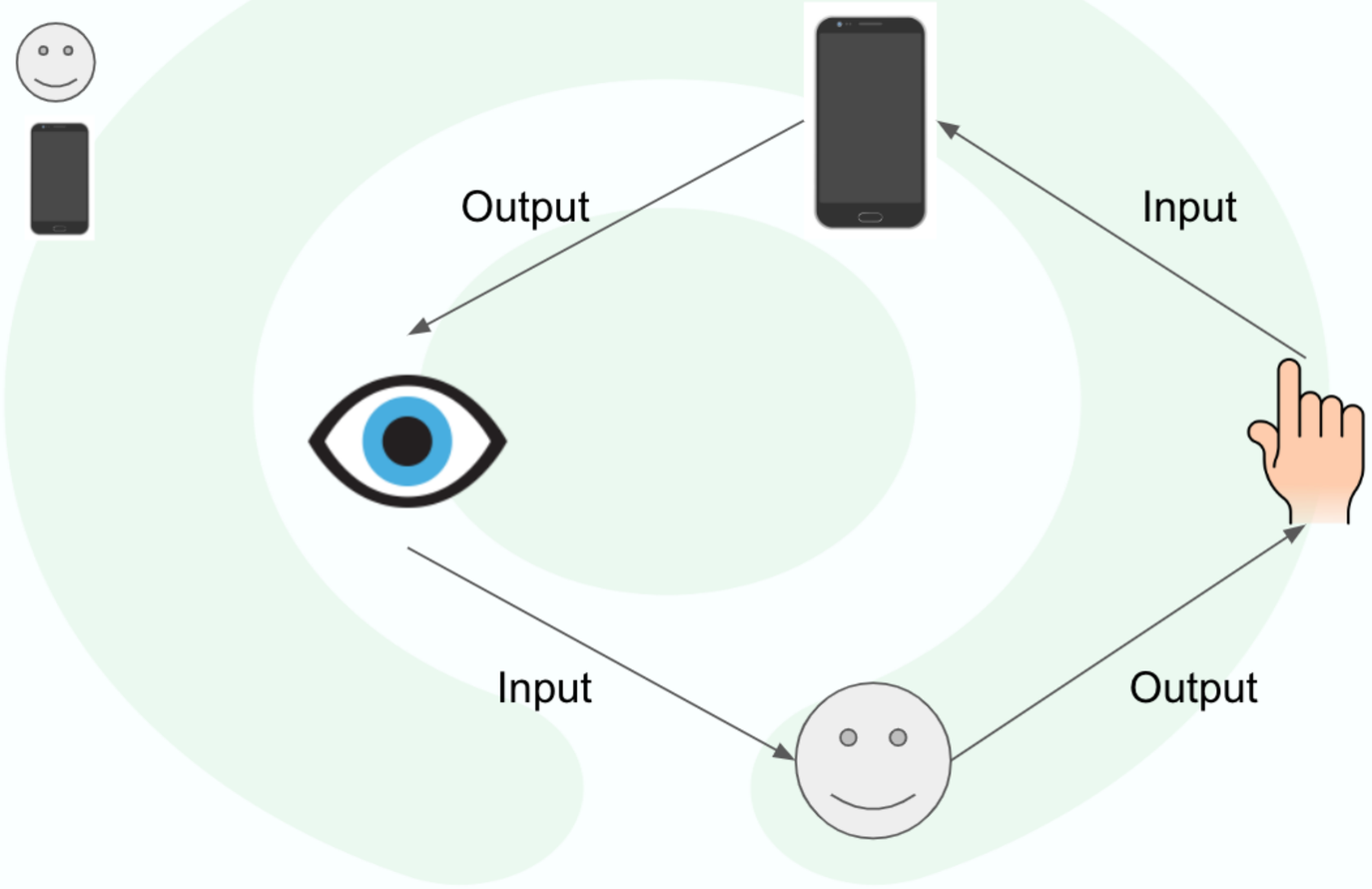
MVI implementation be like VIM(View Intent Model)

view -> intent -> model

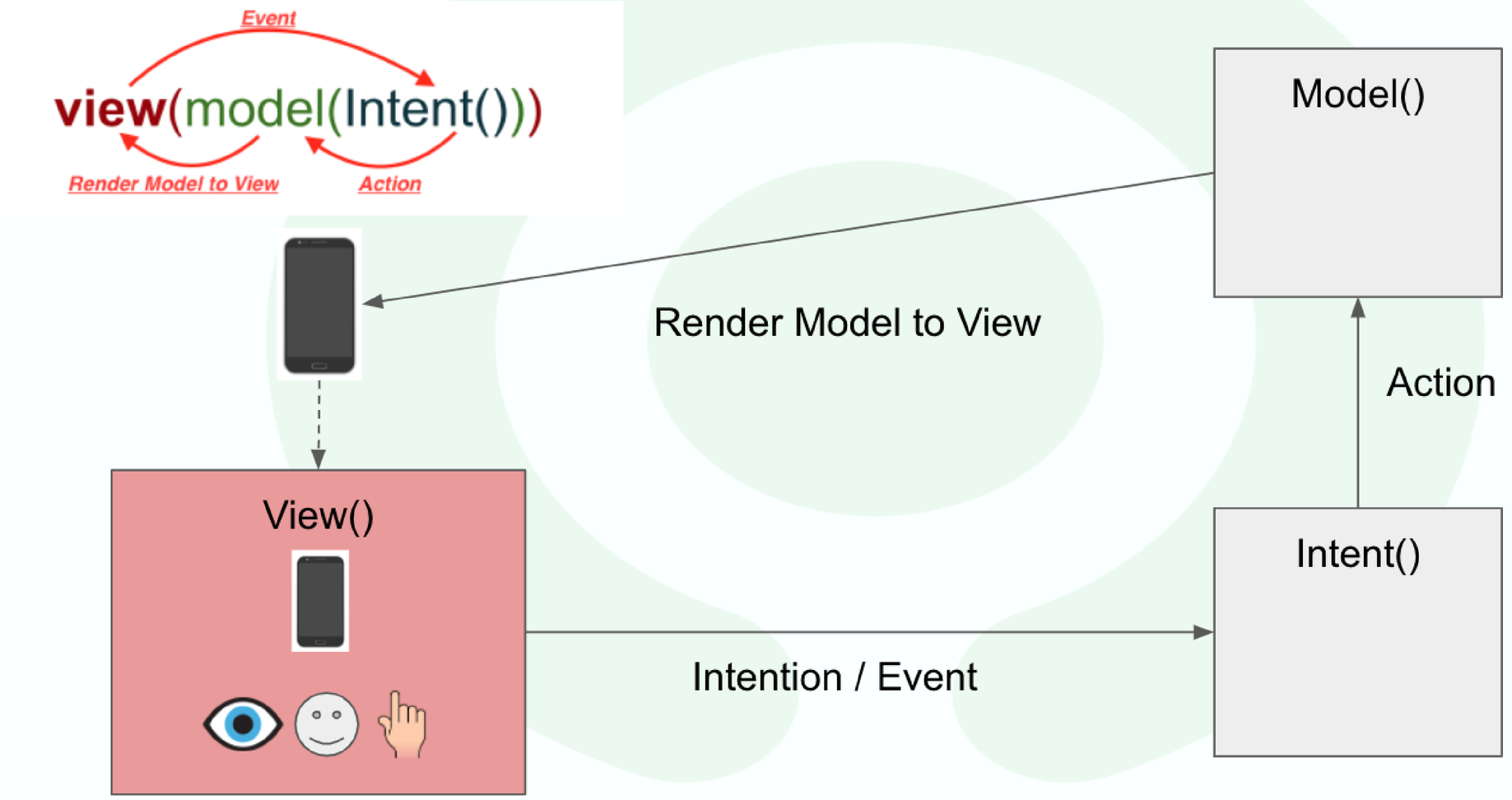


How app and user interact?

1. User
2. Device



Mapping User Interaction with MVI.



Recap

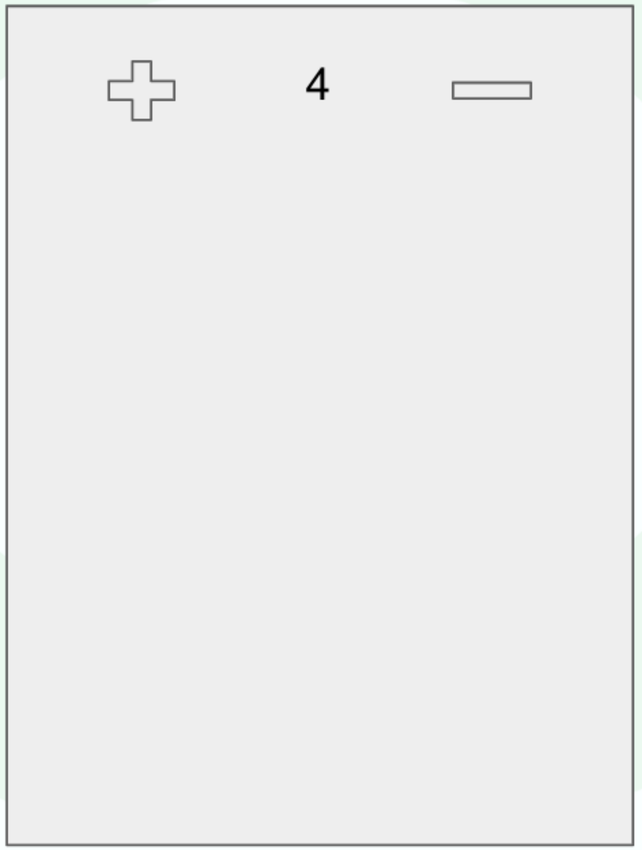
- Unidirectional
- View
- User and it's interaction
- Intent
- Action
- Model

It's time to see this pattern in code

Language and libraries as Prerequisite (All are optional)

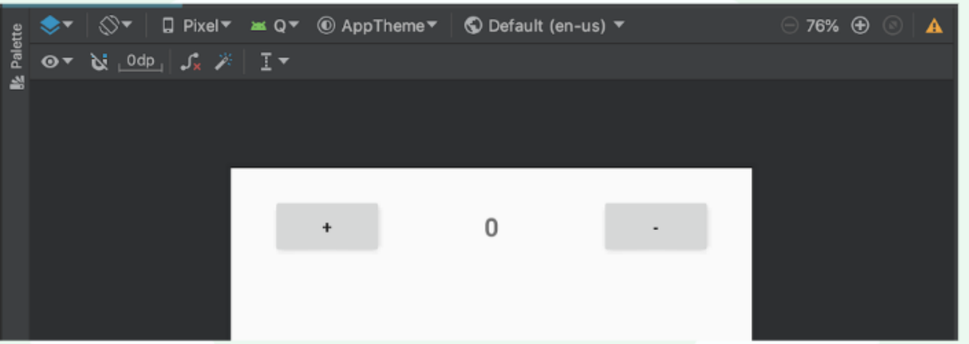
- Kotlin Language (Sealed classes, data classes, when exhaustive)
- RxJava/Kotlin (map, merge, scan)
- Rxbinding

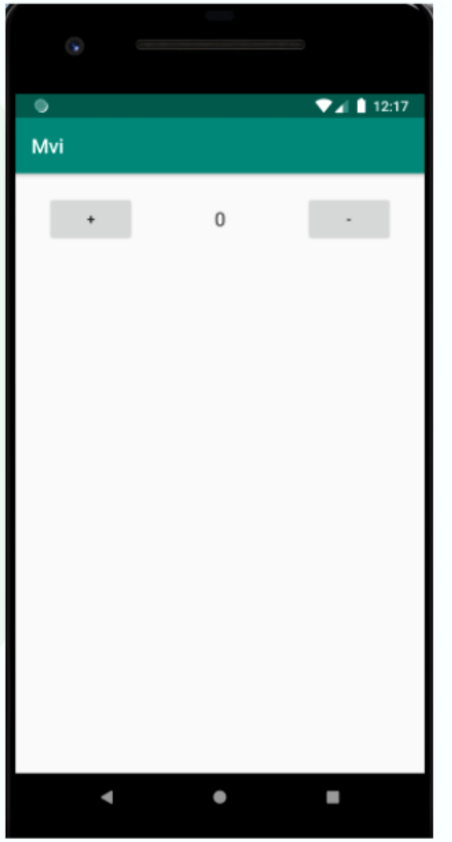
Easy Scenario



Android Ceremonies

```
package com.uwantolearn.mvi.mvi_presentation

import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import com.uwantolearn.mvi.R

class HomeActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home)
    }
}
```





Implementation Strategy

There are a lot of strategies for implementation of this Pattern. In our case, we will go with the simplest one. We will do prepare independent concepts, and once they are ready, we will write glue code in between these concepts.

User Intentions or Intent

```
sealed class HomeIntent{
    object IncrementIntent : HomeIntent()
    object DecrementIntent : HomeIntent()
}
```

Code how user will generate these intentions.

```
incrementButton.clicks().map { HomeIntent.IncrementIntent },
decrementButton.clicks().map { HomeIntent.DecrementIntent }
```

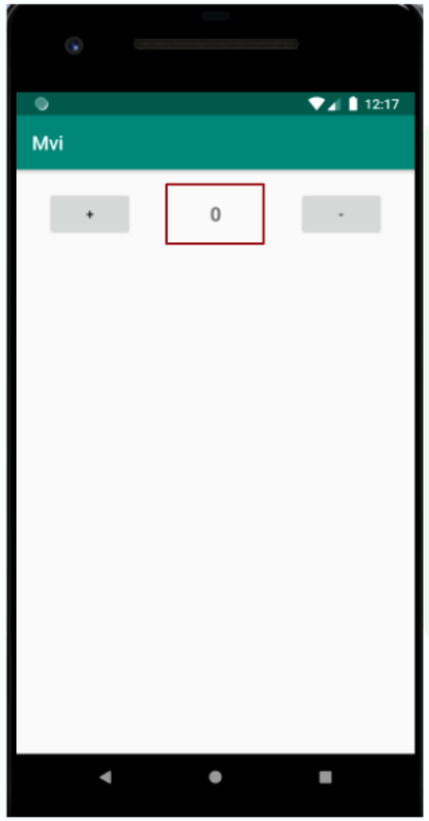
Merge All Intents

```
Observable.merge(
    listOf<Observable<HomeIntent>>(
        incrementButton.clicks().map { HomeIntent.IncrementIntent },
        decrementButton.clicks().map { HomeIntent.DecrementIntent }
    )
)
```

View Will Provide all intents using getIntents()

```
fun getIntents(): Observable<HomeIntent> =
    Observable.merge(
        listOf<Observable<HomeIntent>>(
            incrementButton.clicks().map { HomeIntent.IncrementIntent },
            decrementButton.clicks().map { HomeIntent.DecrementIntent }
        )
    )
```

ViewState (Model will be as view state)




```
data class HomeViewState(val counter: Int)
```

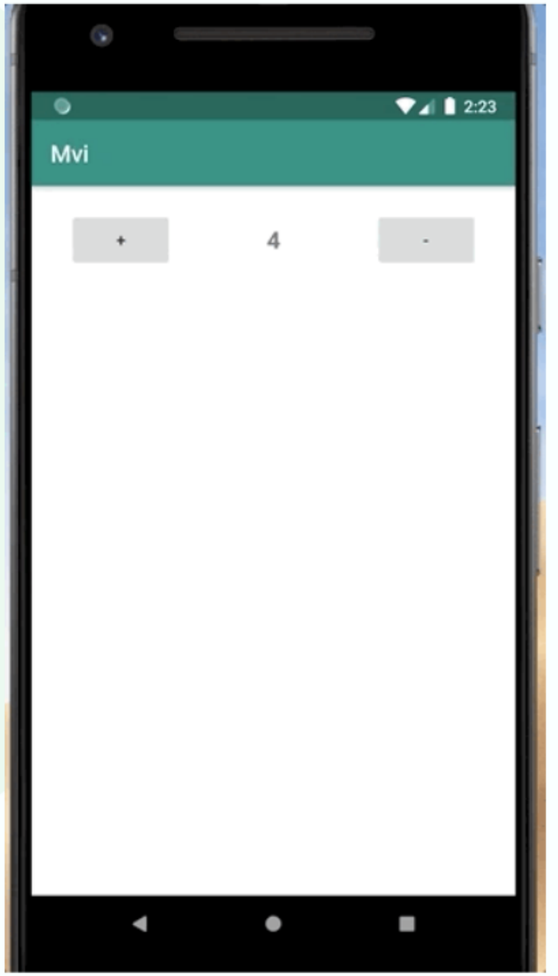
Intent Processing

```
private fun reduce(previousState: HomeViewState, intent: HomeIntent): HomeViewState =
    when (intent) {
        HomeIntent.IncrementIntent -> previousState.copy(counter = previousState.counter + 1)
        HomeIntent.DecrementIntent -> previousState.copy(counter = previousState.counter - 1)
    }
```

```
.scan(HomeViewState( counter: 0), ::reduce)
```

Glue code between Model View Intent

```kotlin
package com.uwantolearn.mvi.mvi_presentation

import ...

class HomeActivity : AppCompatActivity() {

    private val compositeDisposable = CompositeDisposable()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home)

        getIntents()
            .scan(HomeViewState( counter: 0), ::reduce)
            .subscribe(::render)
            .let(compositeDisposable::add)
    }

    private fun getIntents(): Observable<HomeIntent> =
        Observable.merge(
            listOf<Observable<HomeIntent>>(
                incrementButton.clicks().map { HomeIntent.IncrementIntent },
                decrementButton.clicks().map { HomeIntent.DecrementIntent }
            )
        )

    private fun reduce(previousState: HomeViewState, intent: HomeIntent): HomeViewState =
        when (intent) {
            HomeIntent.IncrementIntent -> previousState.copy(counter = previousState.counter + 1)
            HomeIntent.DecrementIntent -> previousState.copy(counter = previousState.counter - 1)
        }


    private fun render(state: HomeViewState) {
        state.counter.toString()
            .let(counterTextView::setText)
    }

    override fun onDestroy() {
        compositeDisposable.clear()
        super.onDestroy()
    }


}

sealed class HomeIntent {
    object IncrementIntent : HomeIntent()
    object DecrementIntent : HomeIntent()
}

data class HomeViewState(val counter: Int)
```



Until Now

- We know what is Unidirectional flow and how we can achieve this in code.
- View, Model (as ViewState), Intents, Intent Processing (scan, reduce) are clear

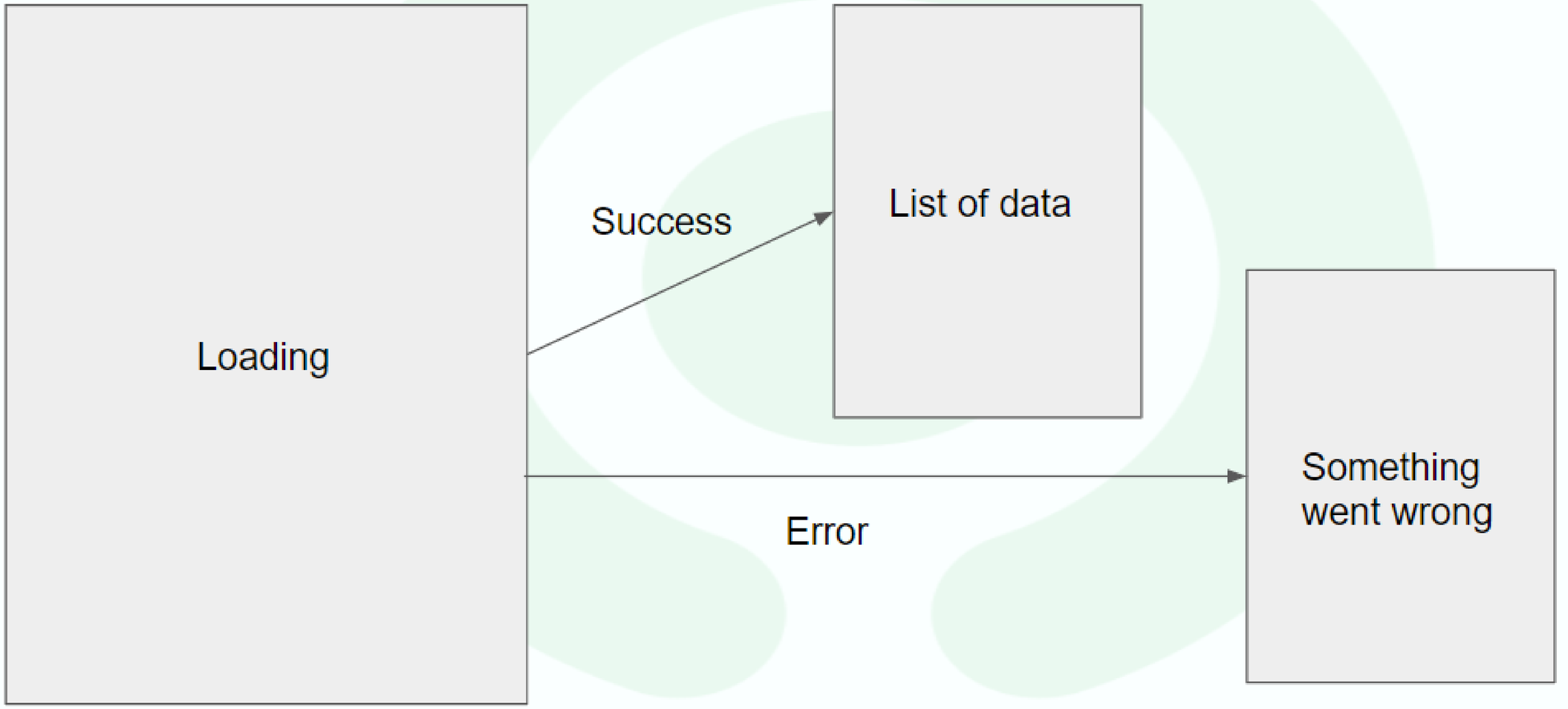
Code: https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-easy-implementation

Any confusions or questions?

Issues or Problems Until Now

- ViewState, Intents, processing are coupled
- Everything is Synchronized (No Async calls)
- Config changes are not handle (Will lose data on rotation)
- Android Process Death not handle

Medium Scenario



New Intentions

```
sealed class HomeIntent {
    object IncrementIntent : HomeIntent()
    object DecrementIntent : HomeIntent()
    object LoadDataIntent : HomeIntent() // New Intent which we need in new requirement
}
```

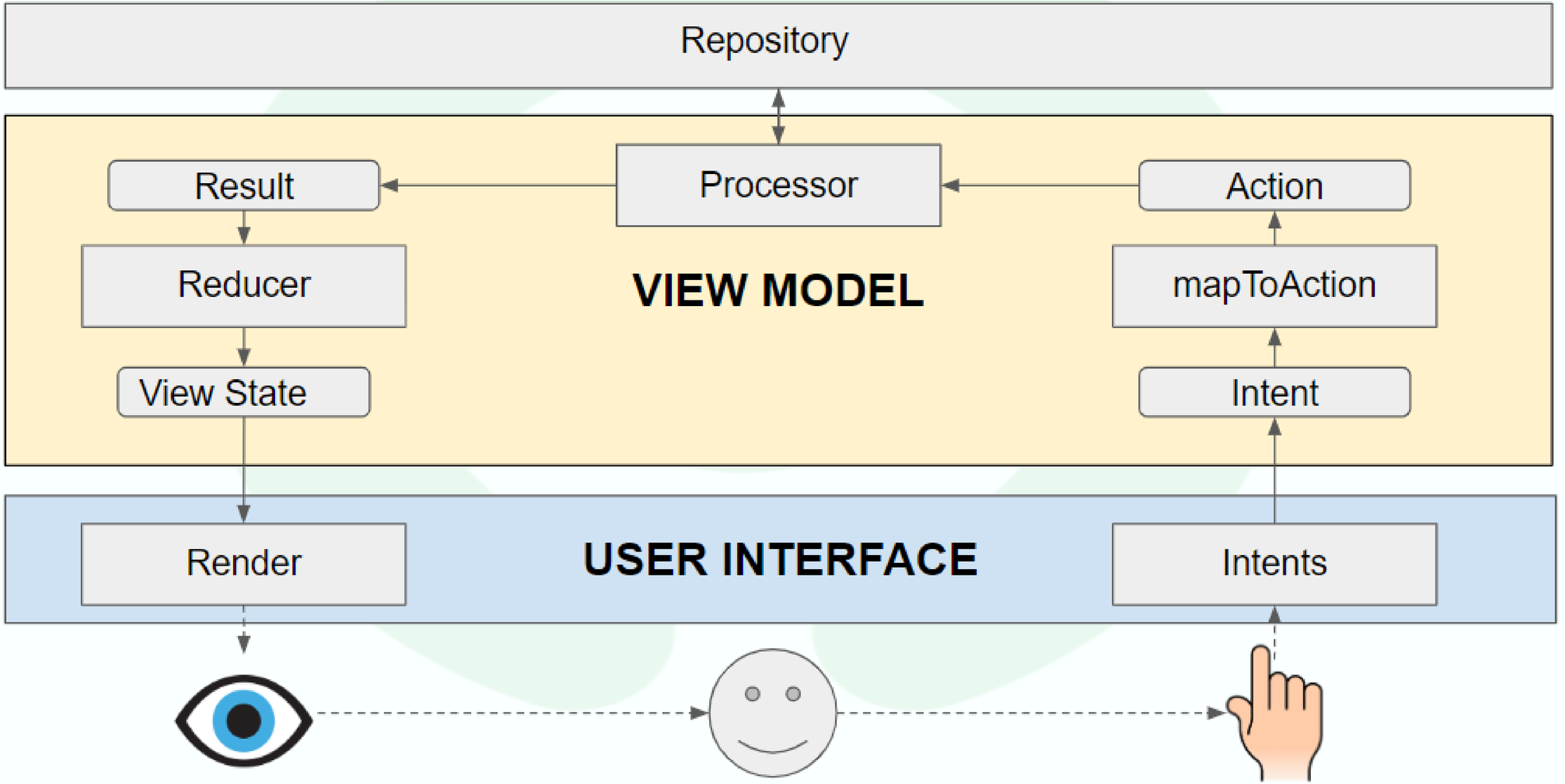
ViewState

```
data class HomeViewState(
    val counter: Int,
    val inProgress: Boolean,
    val data: List<String>,
    val isError: Boolean
)
```

Here, I change ViewState data class to sealed class. We can use both.

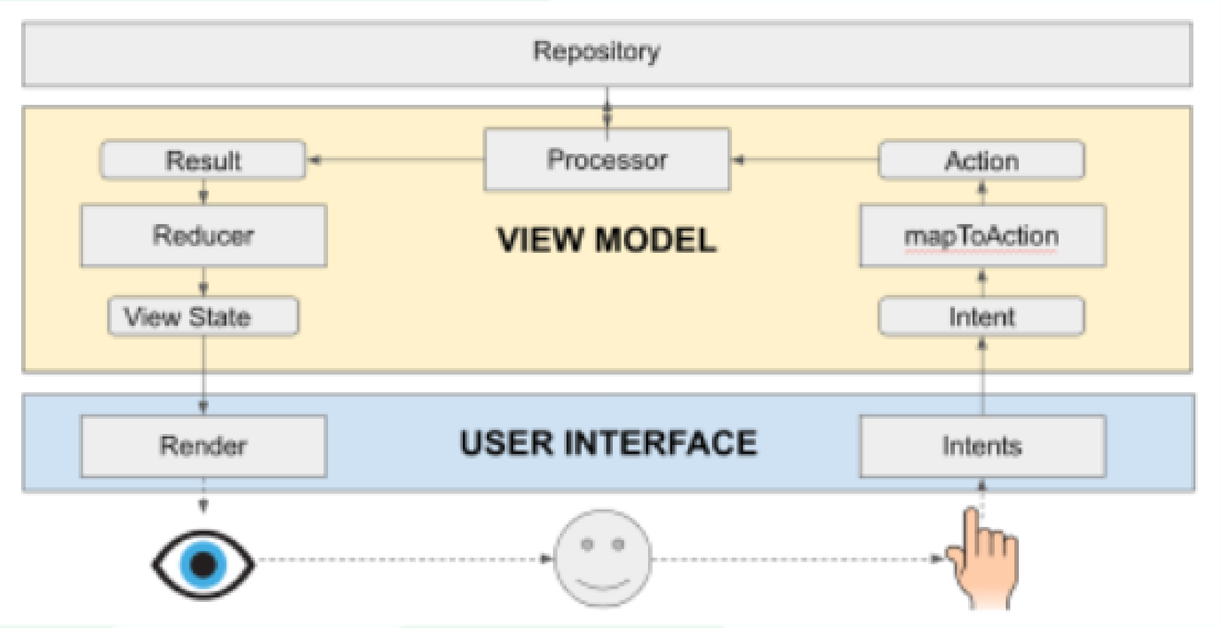
```
sealed class HomeViewState {
    object ProgressViewState : HomeViewState()
    object FailureViewState : HomeViewState()
    data class DataViewState(val list: List<String>) : HomeViewState()
    data class CounterViewState(val counter: Int) : HomeViewState()
}
```

Fix of ViewState, Intents and Processing Coupling



User Interface

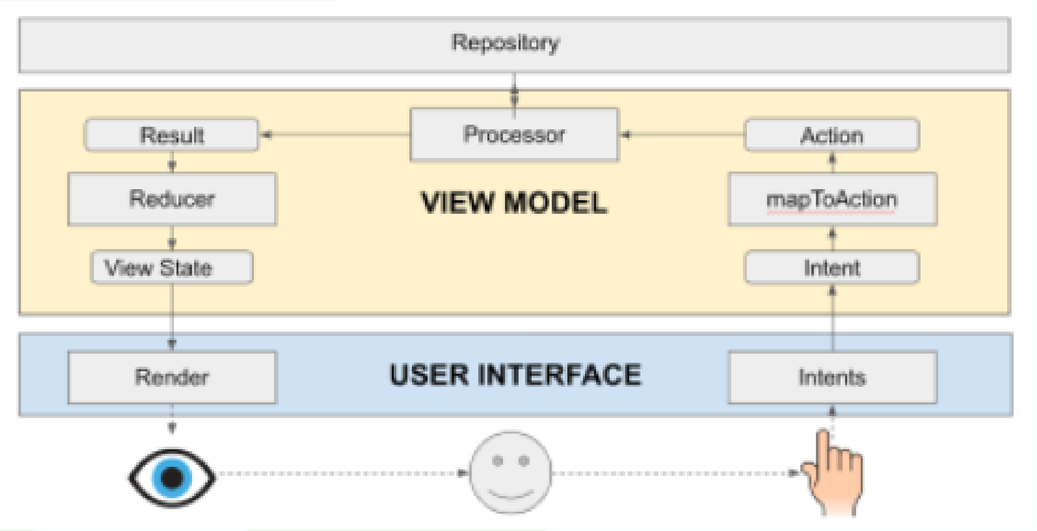
```kotlin
package com.example.mvi.mvi_presentation

import ...

class HomeActivity : AppCompatActivity() {

    private val compositeDisposable = CompositeDisposable()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home)
    }

    override fun onDestroy() {
        compositeDisposable.clear()
        super.onDestroy()
    }

    fun intents(): Observable<HomeIntent> =
        Observable.merge(
            listOf<Observable<HomeIntent>>(
                Observable.just(HomeIntent.LoadDataIntent)
            )
        )

    fun render(state: HomeViewState): Unit = when (state) {
        HomeViewState.ProgressViewState -> renderLoadingState()
        HomeViewState.FailureViewState -> renderFailureState()
        is HomeViewState.DataViewState -> renderDataState(state.list)
    }

    private fun renderLoadingState() {
        dataTextView.visibility = View.GONE
        progressBar.visibility = View.VISIBLE
    }

    private fun renderFailureState() {
        progressBar.visibility = View.GONE

        dataTextView.visibility = View.VISIBLE
        dataTextView.text = getString(R.string.something_went_wrong)
    }

    private fun renderDataState(data: List<String>) {
        progressBar.visibility = View.GONE

        dataTextView.visibility = View.VISIBLE
        data.reduce { acc, s -> "$acc, $s" }
            .let(dataTextView::setText)
    }
}

sealed class HomeIntent {
    object LoadDataIntent : HomeIntent()
}

sealed class HomeViewState {
    object ProgressViewState : HomeViewState()
    object FailureViewState : HomeViewState()
    data class DataViewState(val list: List<String>) : HomeViewState()
}
```



Repository

MVIPresentation.kt | MVIPresentationRepo.kt

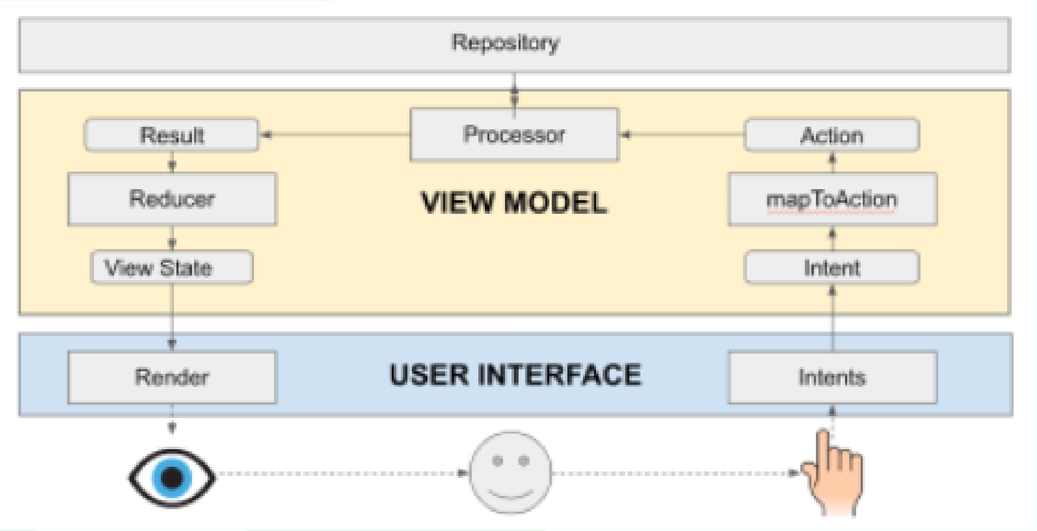
```kotlin
package com.uwantolearn.mvi.mvi_presentation

import ...

interface MVIPresentationRepo {
    fun loadData(): Observable<List<String>>
}

class MVIPresentationRepoImpl : MVIPresentationRepo {

    override fun loadData(): Observable<List<String>> = Observable
        .timer( delay: 5, TimeUnit.SECONDS, Schedulers.io())
        .map { it: Long
            listOf(
                "GoJek Indonesia",
                "GoJekSingapore",
                "GoJekIndia",
                "GoJekThailand",
                "GoJekPhilippines",
                "GoJekVietnam"
            )
        }

}
```



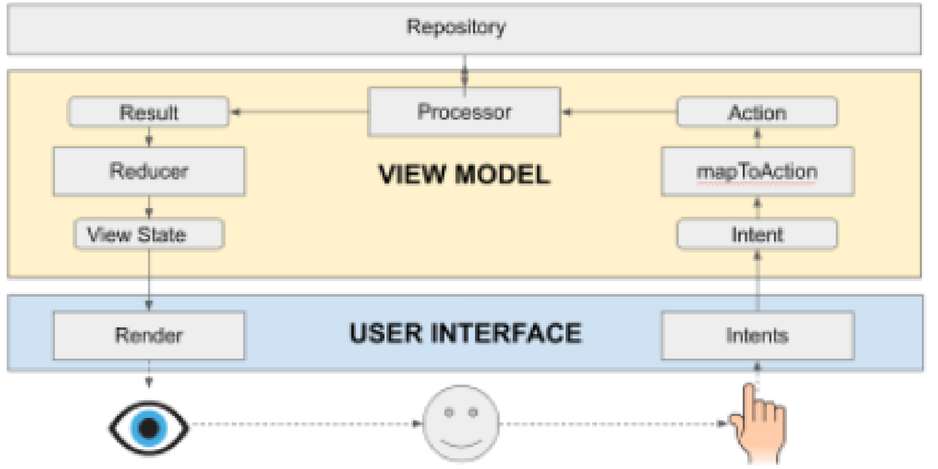
Repository

MVIPresentation.kt | MVIPresentationRepo.kt

```kotlin
package com.uwantolearn.mvi.mvi_presentation

import ...

interface MVIPresentationRepo {
    fun loadData(): Observable<List<String>>
}

class MVIPresentationRepoImpl : MVIPresentationRepo {

    override fun loadData(): Observable<List<String>> = Observable
        .timer( delay: 5, TimeUnit.SECONDS, Schedulers.io())
        .map { it: Long
            listOf(
                "GoJek Indonesia",
                "GoJekSingapore",
                "GoJekIndia",
                "GoJekThailand",
                "GoJekPhilippines",
                "GoJekVietnam"
            )
        }

}
```



ViewModel

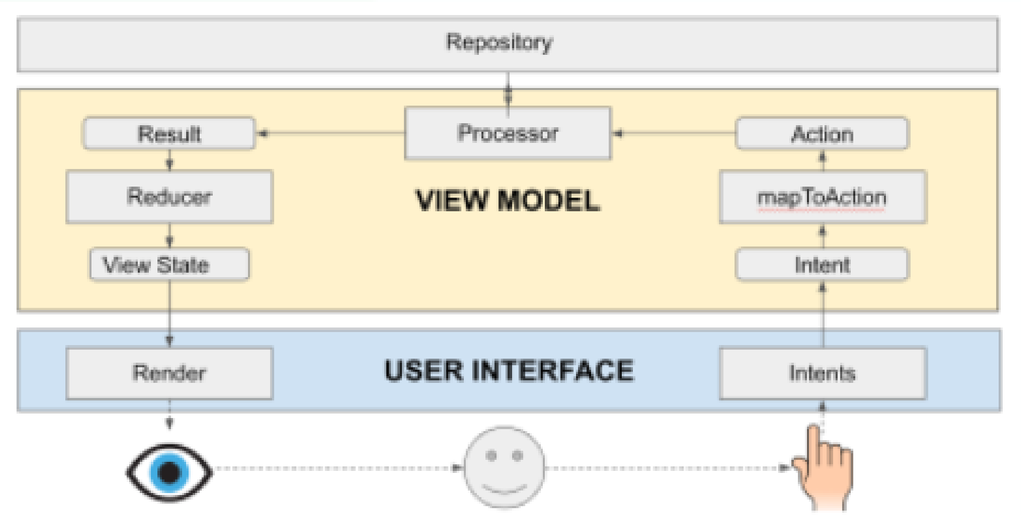



```
sealed class HomeActivityAction {
    object LoadDataAction : HomeActivityAction()
}
```

```
private fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {
    HomeIntent.LoadDataIntent -> HomeActivityAction.LoadDataAction
}
```

ViewModel

```
sealed class HomeActivityResult {
    data class DataResult(val data: List<String>) : HomeActivityResult()
    object FailureResult : HomeActivityResult()
    object LoadingResult : HomeActivityResult()
}
```

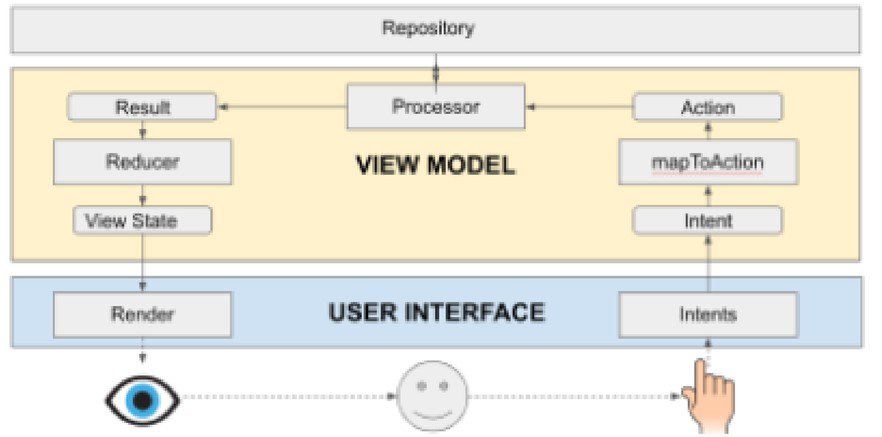



```
private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
    when (result) {
        is HomeActivityResult.DataResult -> HomeViewState.DataViewState(result.data)
        HomeActivityResult.FailureResult -> HomeViewState.FailureViewState
        HomeActivityResult.LoadingResult -> HomeViewState.ProgressViewState
    }
```

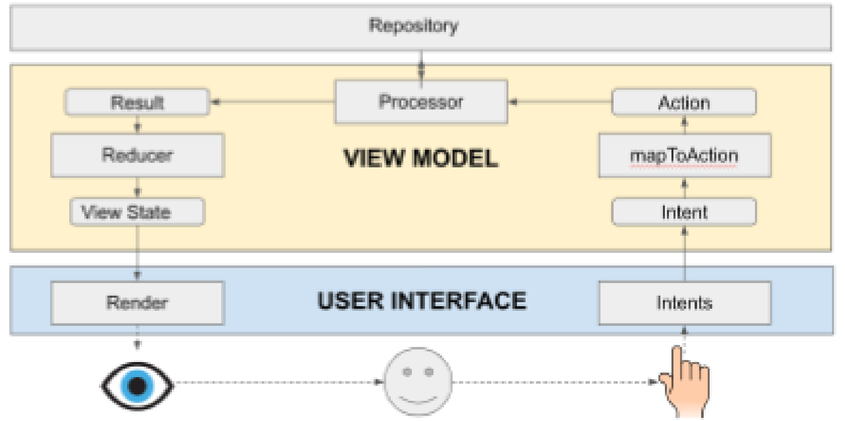
Until Now View Model

MVIPresentation.kt | MVIPresentationViewModel.kt

```kotlin
package com.uwantolearn.mvi.mvi_presentation

import io.reactivex.ObservableTransformer

class MVIPresentationViewModel(val repo: MVIPresentationRepo) {

    private fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {
        HomeIntent.LoadDataIntent -> HomeActivityAction.LoadDataAction
    }

    private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
        when (result) {
            is HomeActivityResult.DataResult -> HomeViewState.DataViewState(result.data)
            HomeActivityResult.FailureResult -> HomeViewState.FailureViewState
            HomeActivityResult.LoadingResult -> HomeViewState.ProgressViewState
        }
}

sealed class HomeActivityAction {
    object LoadDataAction : HomeActivityAction()
}

sealed class HomeActivityResult {
    data class DataResult(val data: List<String>) : HomeActivityResult()
    object FailureResult : HomeActivityResult()
    object LoadingResult : HomeActivityResult()
}
```



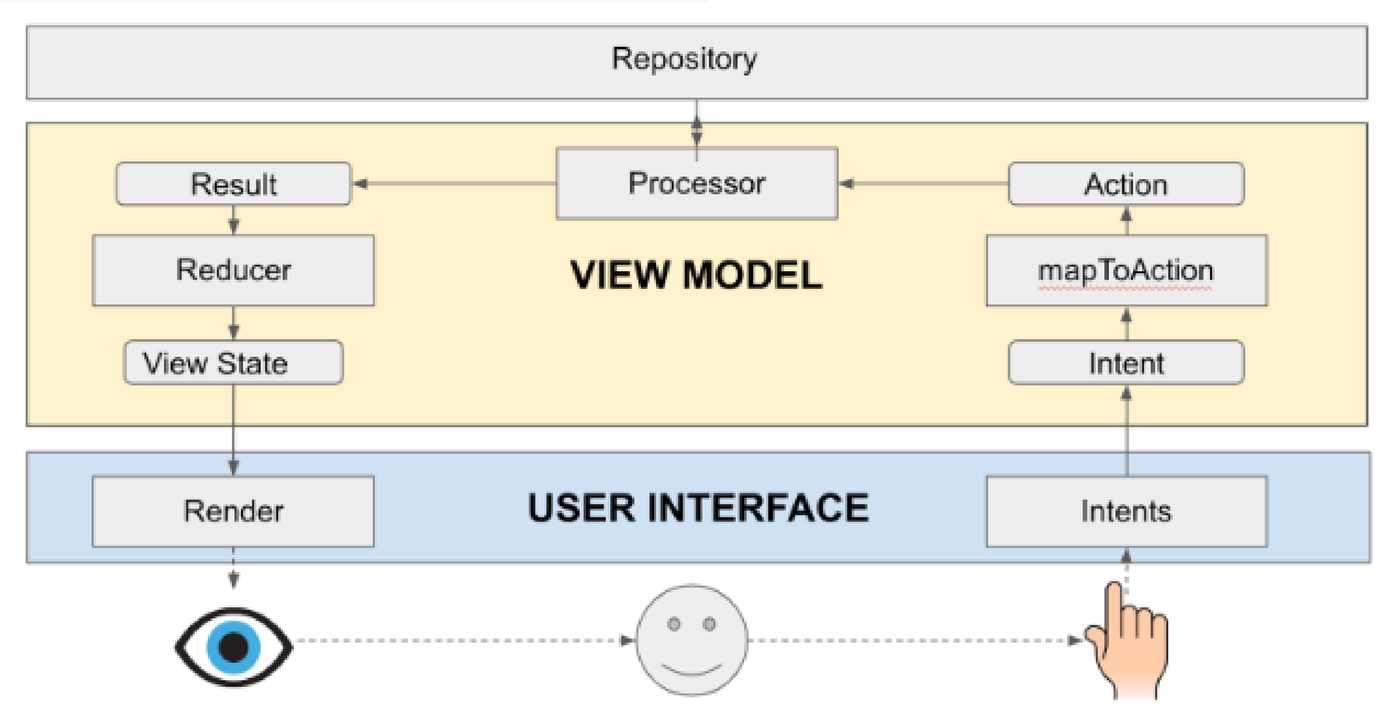
ViewModel




```
class MVIPresentationActionProcessor(private val repo: MVIPresentationRepo) {

    private val loadDataActionProcessor =
        ObservableTransformer<HomeActivityAction.LoadDataAction, HomeActivityResult> { action ->
            action.flatMap { it: HomeActivityAction.LoadDataAction
                repo.loadData()
                    .map(HomeActivityResult::DataResult)
                    .cast(HomeActivityResult::class.java)
                    .onErrorReturn { HomeActivityResult.FailureResult }
                    .startWith(HomeActivityResult.LoadingResult)
            }
        }

    val actionProcessor = ObservableTransformer<HomeActivityAction, HomeActivityResult> { action ->
        action.publish { actionSource ->
            actionSource.ofType(HomeActivityAction.LoadDataAction::class.java)
                .compose(loadDataActionProcessor)
        }
    }
}
```

Complete View Model

```
package com.iwantolearn.mvi.mvi_presentation

import ...

class MviPresentationViewModel(repo: MviPresentationRepo) {

    private val actionProcessor = MviPresentationActionProcessor(repo)

    fun bind(intents: Observable<HomeIntent>): Observable<HomeViewState> =
        intents
            .map(::mapToActions)
            .compose(actionProcessor.processActions)
            .scan(HomeViewState.ProgressViewState, ::reduce)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())

    private fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {
        HomeIntent.LoadDataIntent -> HomeActivityAction.LoadDataAction
    }

    private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
        when (result) {
            is HomeActivityResult.DataResult -> HomeViewState.DataViewState(result.data)
            HomeActivityResult.FailureResult -> HomeViewState.FailureViewState
            HomeActivityResult.LoadingResult -> HomeViewState.ProgressViewState
        }
}

sealed class HomeActivityAction {
    object LoadDataAction : HomeActivityAction()
}

sealed class HomeActivityResult {
    data class DataResult(val data: List<String>) : HomeActivityResult()
    object FailureResult : HomeActivityResult()
    object LoadingResult : HomeActivityResult()
}

class MviPresentationActionProcessor(private val repo: MviPresentationRepo) {

    private val loadDataActionProcessor =
        ObservableTransformer<HomeActivityAction.LoadDataAction, HomeActivityResult> { action ->
            action.flatMap { it: HomeActivityAction.LoadDataAction
                repo.loadData()
                    .map(HomeActivityResult::DataResult)
                    .cast(HomeActivityResult::class.java)
                    .onErrorReturn { HomeActivityResult.FailureResult }
                    .startWith(HomeActivityResult.LoadingResult)
            }
        }

    val processActions = ObservableTransformer<HomeActivityAction, HomeActivityResult> { action ->
        action.publish { actionSource ->
            actionSource.ofType(HomeActivityAction.LoadDataAction::class.java)
                .compose(loadDataActionProcessor)
        }
    }
}
```



Glue Code

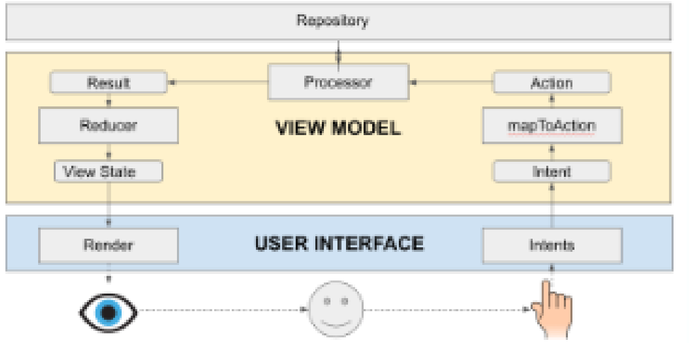
```kotlin
package com.uwantolearn.mvi.mvi_presentation

import ...

class HomeActivity : AppCompatActivity() {

    private val compositeDisposable = CompositeDisposable()
    private val viewModel: MVIPresentationViewModel =
        MVIPresentationViewModel(MVIPresentationRepoImpl())

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home)
        intents()
            .let(viewModel::bind)
            .subscribe(::render)
            .let(compositeDisposable::add)
    }

    override fun onDestroy() {
        compositeDisposable.clear()
        super.onDestroy()
    }

    private fun intents(): Observable<HomeIntent> =
        Observable.merge(
            listOf<Observable<HomeIntent>>(
                Observable.just(HomeIntent.LoadDataIntent)))

    private fun render(state: HomeViewState): Unit = when (state) {
        HomeViewState.ProgressViewState -> renderLoadingState()
        HomeViewState.FailureViewState -> renderFailureState()
        is HomeViewState.DataViewState -> renderDataState(state.list)
    }

    private fun renderLoadingState() {
        dataTextView.visibility = View.GONE
        progressBar.visibility = View.VISIBLE
    }

    private fun renderFailureState() {
        progressBar.visibility = View.GONE
        dataTextView.visibility = View.VISIBLE
        dataTextView.text = "Something went wrong!"
    }

    private fun renderDataState(data: List<String>) {
        progressBar.visibility = View.GONE
        dataTextView.visibility = View.VISIBLE
        data.reduce { acc, s -> "$acc\n$s" }.let(dataTextView::setText)
    }
}

sealed class HomeIntent {
    object LoadDataIntent : HomeIntent()
}

sealed class HomeViewState {
    object ProgressViewState : HomeViewState()
    object FailureViewState : HomeViewState()
    data class DataViewState(val list: List<String>) : HomeViewState()
}
```

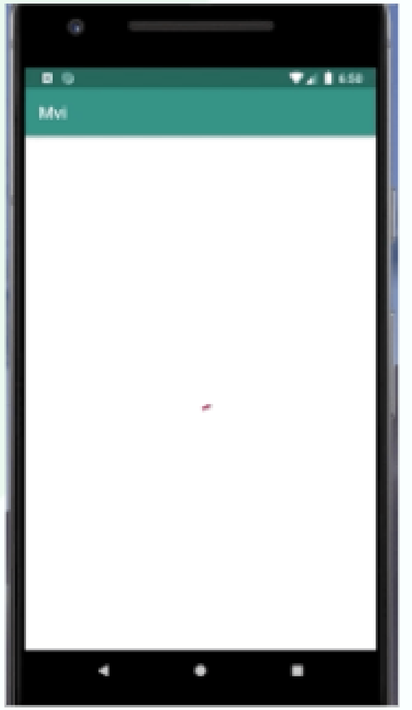
HomeActivity › renderDataState() › data.reduce(...)

```kotlin
class MVIPresentationViewModel(repo: MVIPresentationRepo) {

    private val actionProcessor = MVIPresentationActionProcessor(repo)

    fun bind(intents: Observable<HomeIntent>): Observable<HomeViewState> =
        intents
            .map(::mapToActions)
            .compose(actionProcessor.processActions)
            .scan(HomeViewState.ProgressViewState, ::reduce)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())

    private fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {
        HomeIntent.LoadDataIntent -> HomeActivityAction.LoadDataAction
    }

    private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
        when (result) {
            is HomeActivityResult.DataResult -> HomeViewState.DataViewState(result.data)
            HomeActivityResult.FailureResult -> HomeViewState.FailureViewState
            HomeActivityResult.LoadingResult -> HomeViewState.ProgressViewState
        }
}

sealed class HomeActivityAction {
    object LoadDataAction : HomeActivityAction()
}

sealed class HomeActivityResult {
    data class DataResult(val data: List<String>) : HomeActivityResult()
    object FailureResult : HomeActivityResult()
    object LoadingResult : HomeActivityResult()
}

class MVIPresentationActionProcessor(private val repo: MVIPresentationRepo) {

    private val loadDataActionProcessor =
        ObservableTransformer<HomeActivityAction.LoadDataAction, HomeActivityResult> { action ->
            action.flatMap {...}
        }

    val processActions = ObservableTransformer<HomeActivityAction, HomeActivityResult> { action ->
        action.publish { actionSource ->
            actionSource.ofType(HomeActivityAction.LoadDataAction::class.java)
                .compose(loadDataActionProcessor)
        }
    }
}
```

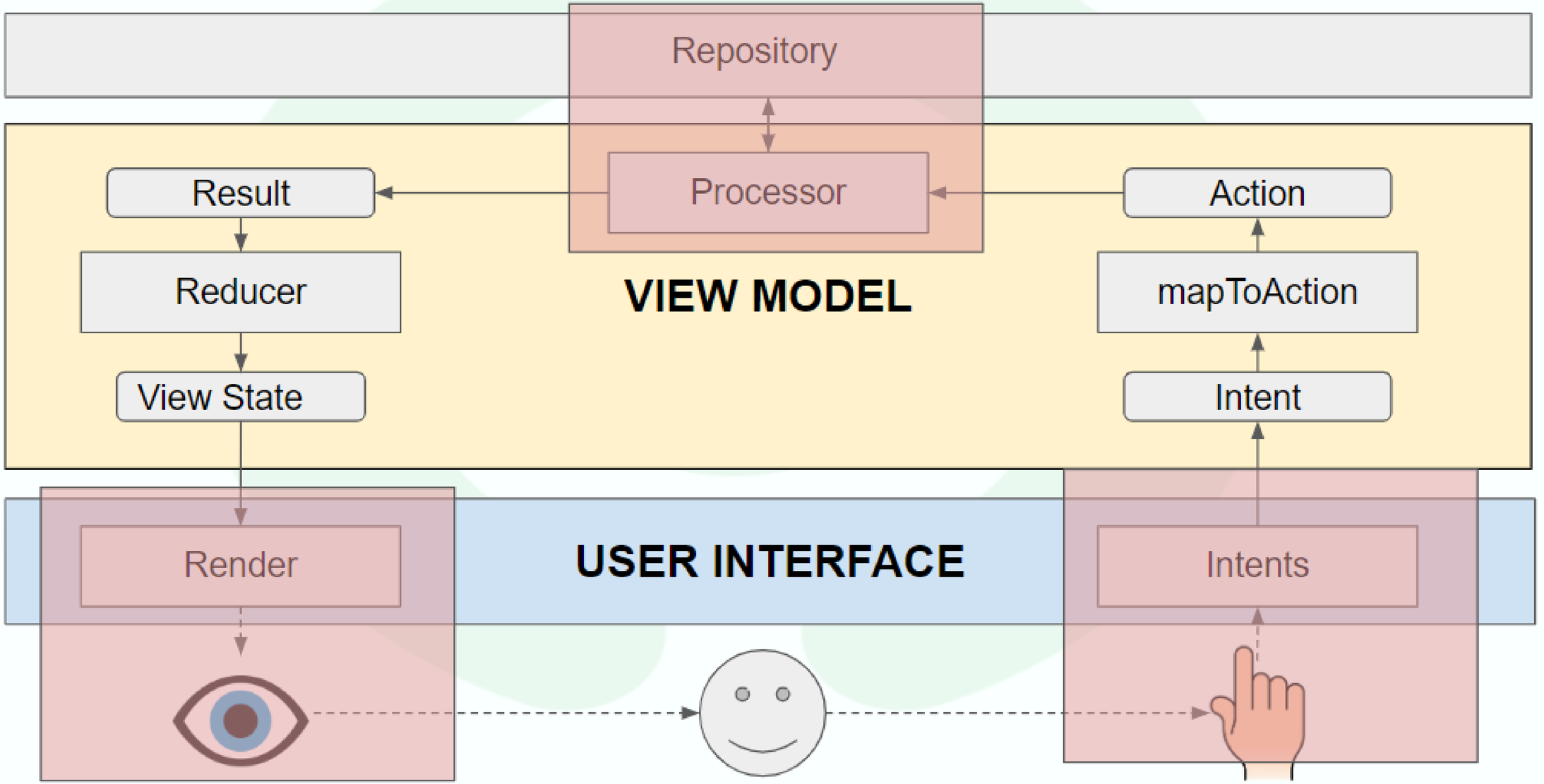
MVIPresentationViewModel › bind()

```kotlin
package com.uwantolearn.mvi.mvi_presentation

import ...

interface MVIPresentationRepo {
    fun loadData(): Observable<List<String>>
}

class MVIPresentationRepoImpl : MVIPresentationRepo {...}
```





Side Effects



Until Now

- We know what is Unidirectional flow and how we can achieve this in code.
- View, Model (as ViewState) , Intents, Intent Processing (scan, reduce) are clear
- Action Processor, Actions and Results
- How to manage Async Code

Code:
https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-medium-implementation-pre-refresh-intent

Any confusions or questions?

Live Demo So we can see some benefits

- Add a Refresh Intent with Exception Mock
- Add a Button to get Random Number

Code:
https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-medium-implementation-complete

Difficult Scenario

- Manage Config Changes (For now if your app is only portrait mode so this is optional step)
- Refactor to a small framework

Config Changes by using AndroidViewModel

```
class MVIPresentationViewModel(repo: MVIPresentationRepo) : ViewModel() {
```

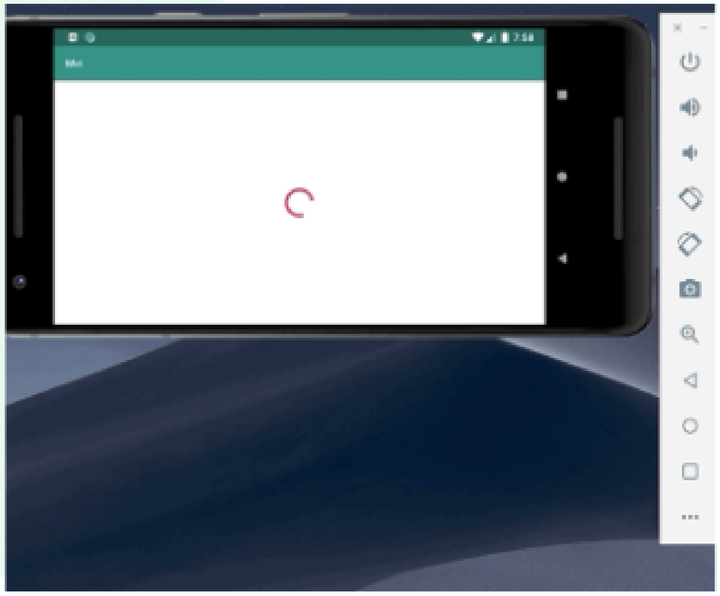
```
class MVIPresentationViewModelFactory : ViewModelProvider.Factory {
    override fun <T : ViewModel?> create(modelClass: Class<T>): T = MVIPresentationViewModel(
        MVIPresentationRepoImpl()
    ) as T
}

class HomeActivity : AppCompatActivity() {

    private val compositeDisposable = CompositeDisposable()
    private lateinit var viewModel: MVIPresentationViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home)
        viewModel =
            ViewModelProviders.of(
                activity: this,
                MVIPresentationViewModelFactory()
            )[MVIPresentationViewModel::class.java]

        intents()
            .let(viewModel::bind)
            .subscribe(::render) { e -> Log.d(this.javaClass.simpleName,  msg: "Error" + e.message) }
            .let(compositeDisposable::add)
    }
```

Is ViewModel Survive Config Changes?

```
setContentView(R.layout.activity_home)
viewModel =  viewModel: MVIPresentationViewModel@5356
    ViewModelProviders.of(
        activity: this,
        MVIPresentationViewModelFactory()
    )[MVIPresentationViewModel::class.java]
```

After Rotation

```
setContentView(R.layout.activity_home)
viewModel =  viewModel: MVIPresentationViewModel@5356
    ViewModelProviders.of(
        activity: this,
        MVIPresentationViewModelFactory()
    )[MVIPresentationViewModel::class.java]
```



What Happened with data :(

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_home)
    viewModel =
        ViewModelProviders.of(
            activity: this,
            MVIPresentationViewModelFactory()
        )[MVIPresentationViewModel::class.java]

    intents()
        .let(viewModel::bind)
        .subscribe(::render) { e -> Log.d(this.javaClass.simpleName, msg: "Error" + e.message) }
        .let(compositeDisposable::add)
}

override fun onDestroy() {
    compositeDisposable.clear()
    super.onDestroy()
}
```

We lose our stream.

Make ViewModel Stream independent from lifecycle.

```
class MVIPresentationViewModel(repo: MVIPresentationRepo) : ViewModel() {

    private val actionProcessor = MVIPresentationActionProcessor(repo)
    private val intentsSubject = PublishSubject.create<HomeIntent>()
    private val states = PublishSubject.create<HomeViewState>()

    init {
        intentsSubject
            .map(::mapToActions)
            .compose(actionProcessor.processActions)
            .scan(HomeViewState.ProgressViewState, ::reduce)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(states)
    }

    fun processIntents(intents: Observable<HomeIntent>): Disposable = intents
        .subscribe(intentsSubject::onNext)

    fun state(): Observable<HomeViewState> = states.hide()

    fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {
        HomeIntent.LoadDataIntent, HomeIntent.RefreshIntent -> HomeActivityAction.LoadDataAction
        HomeIntent.GetRandomNumberIntent -> HomeActivityAction.GetRandomNumberAction
    }

    private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
        when (result) {
            is HomeActivityResult.DataResult -> HomeViewState.DataViewState(result.data)
            HomeActivityResult.FailureResult -> HomeViewState.FailureViewState
            HomeActivityResult.LoadingResult -> HomeViewState.ProgressViewState
            is HomeActivityResult.RandomNumber -> HomeViewState.RandomNumberState(result.randomNumber)
        }
}
```

Diagram Changes

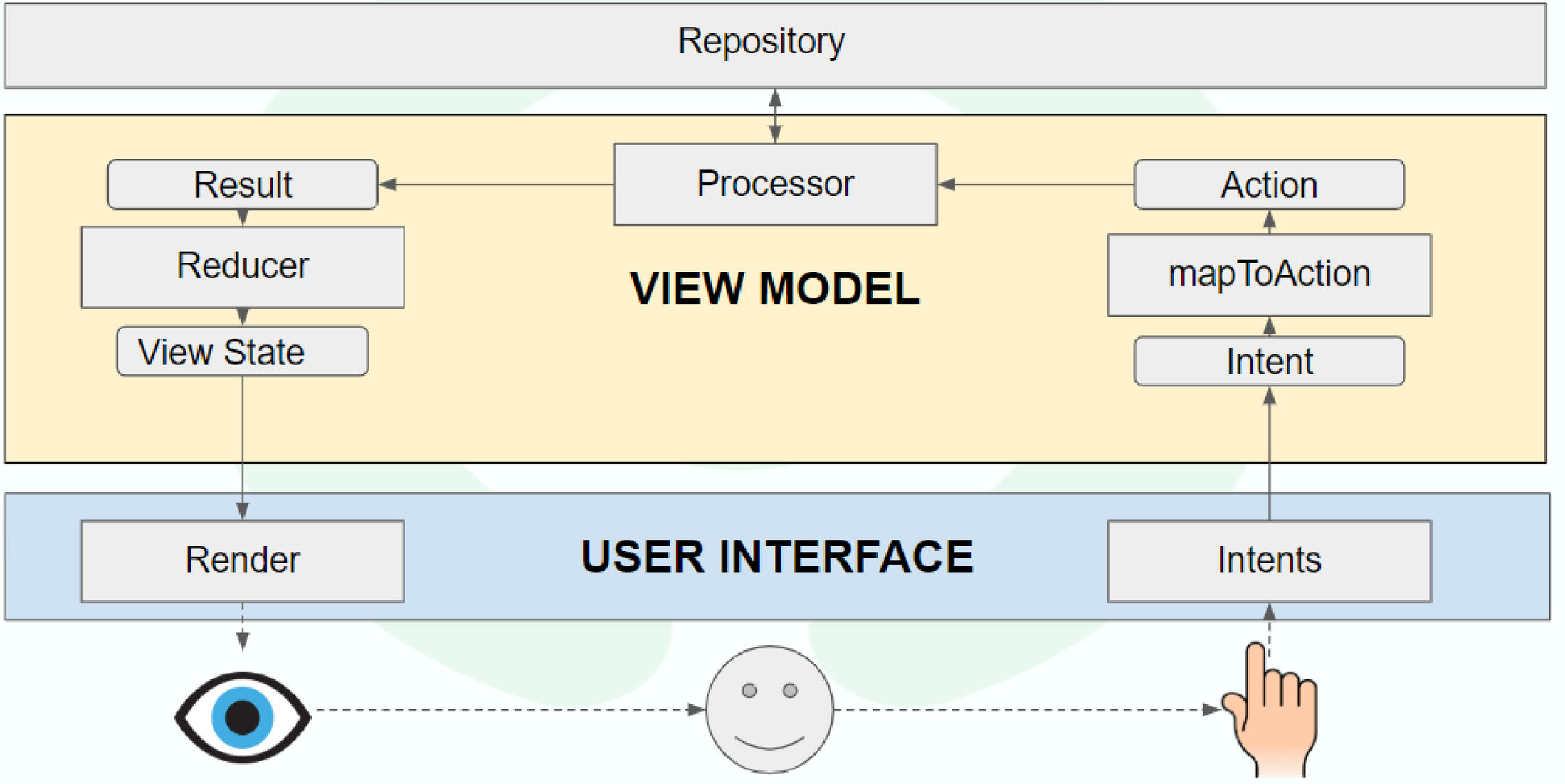


Diagram Changes

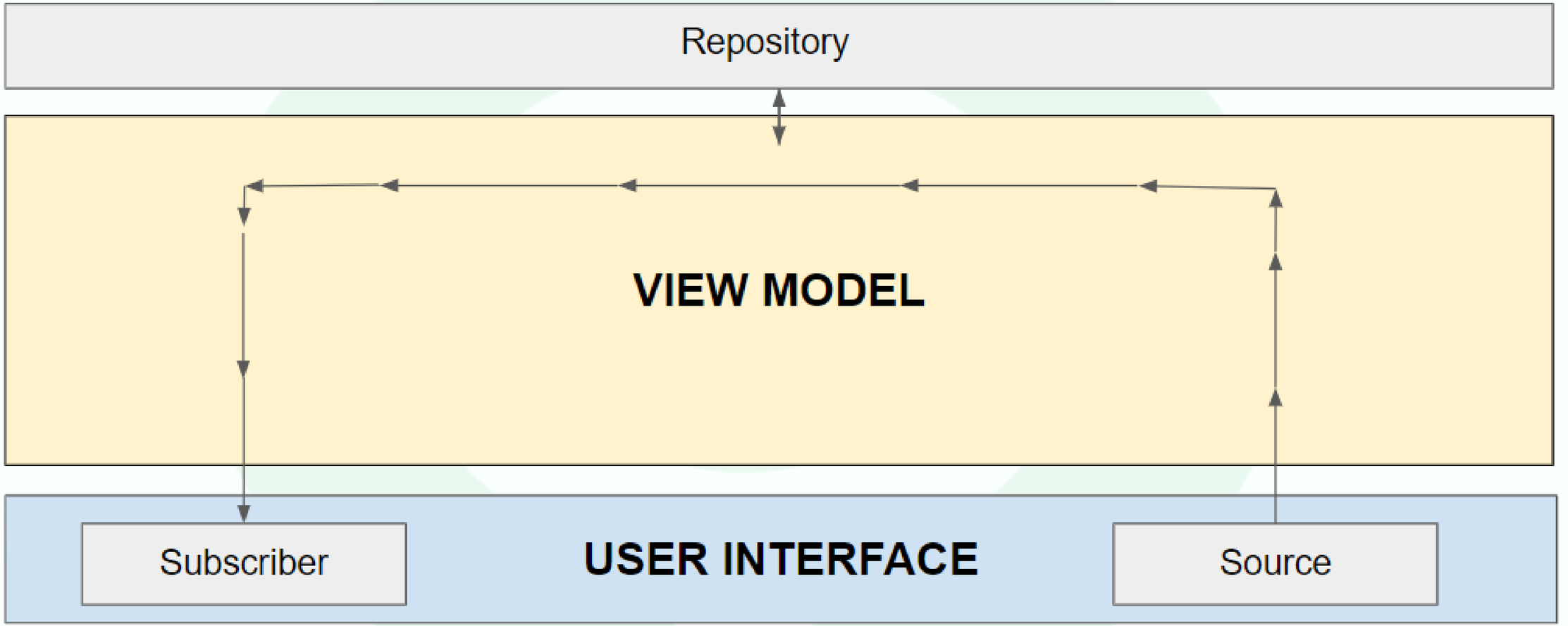


Diagram Changes

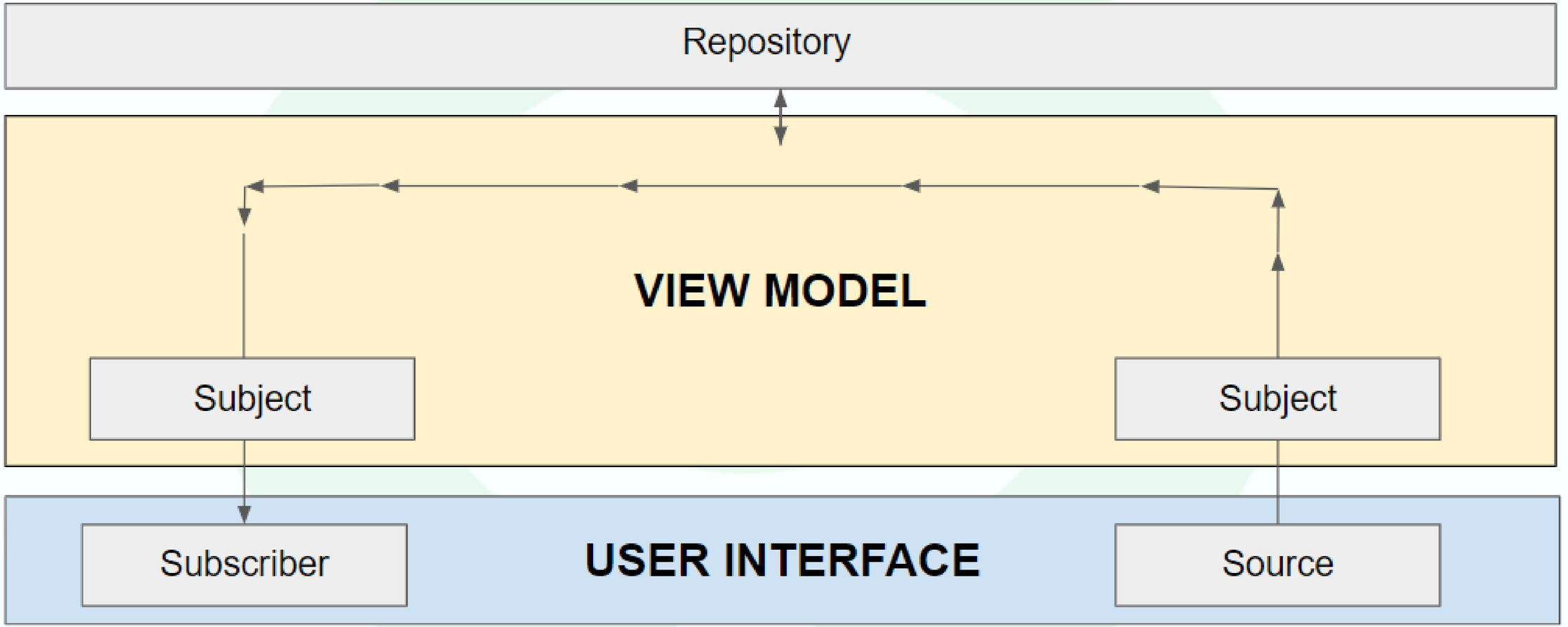


Diagram Changes (Config Change)

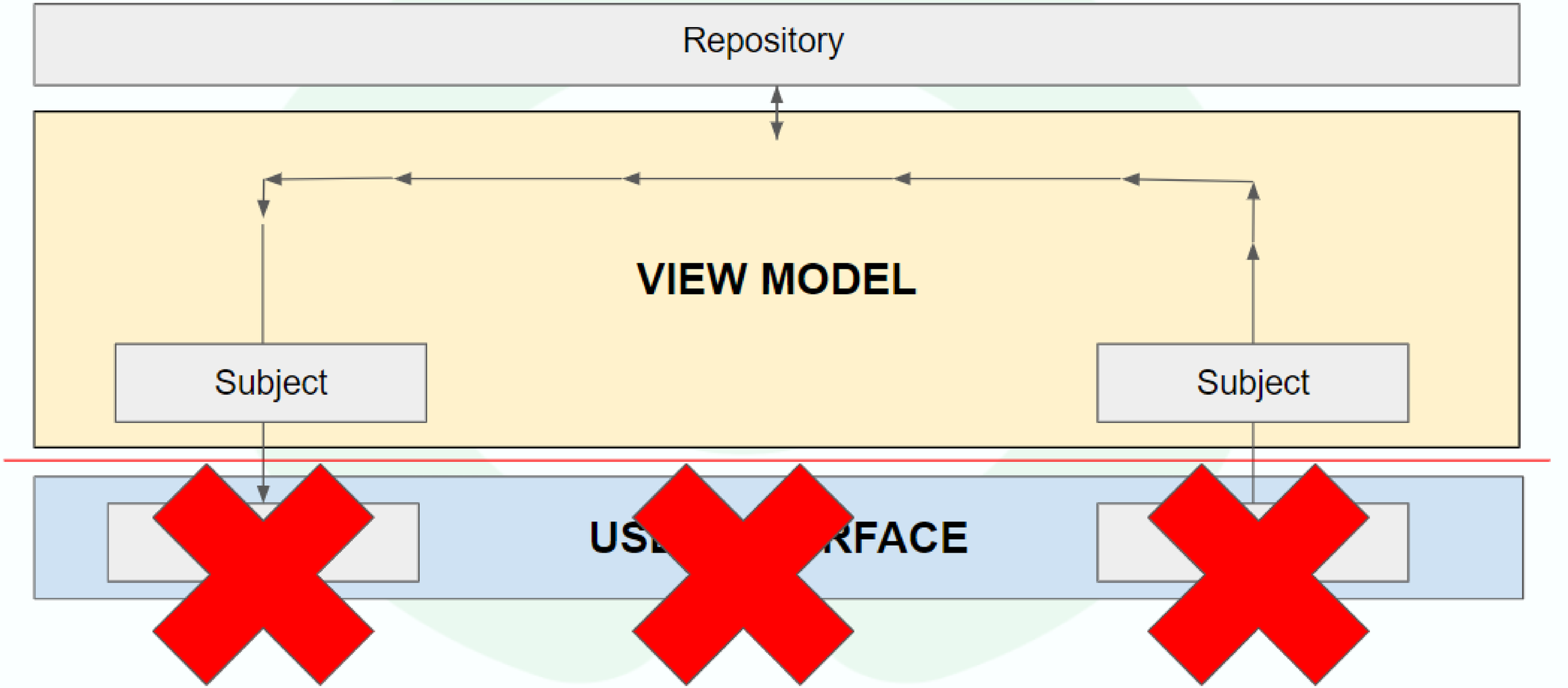


Diagram Changes (Config Change)

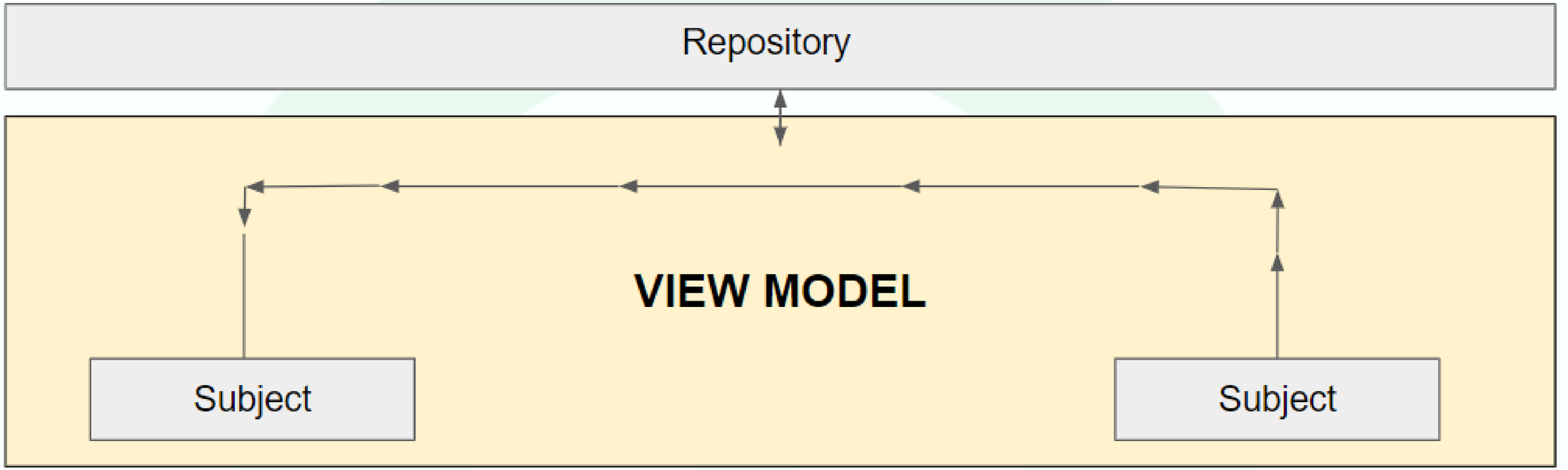
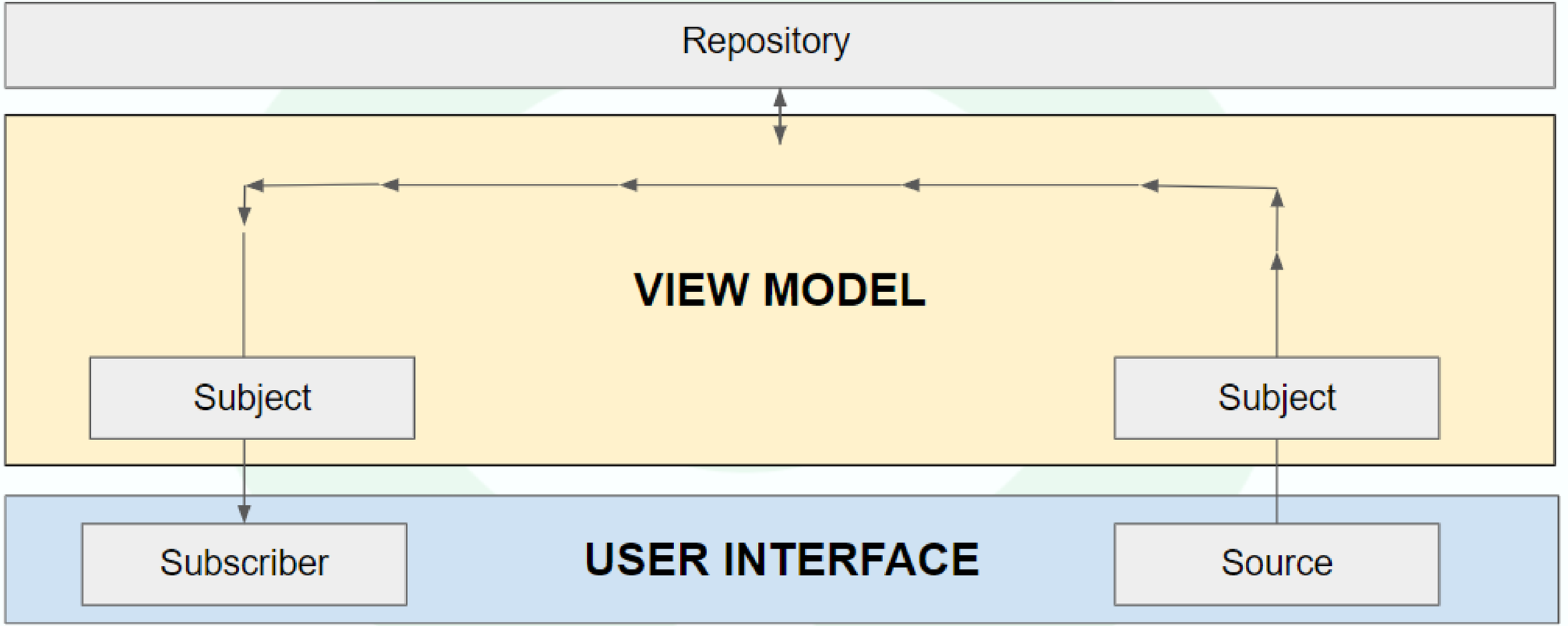
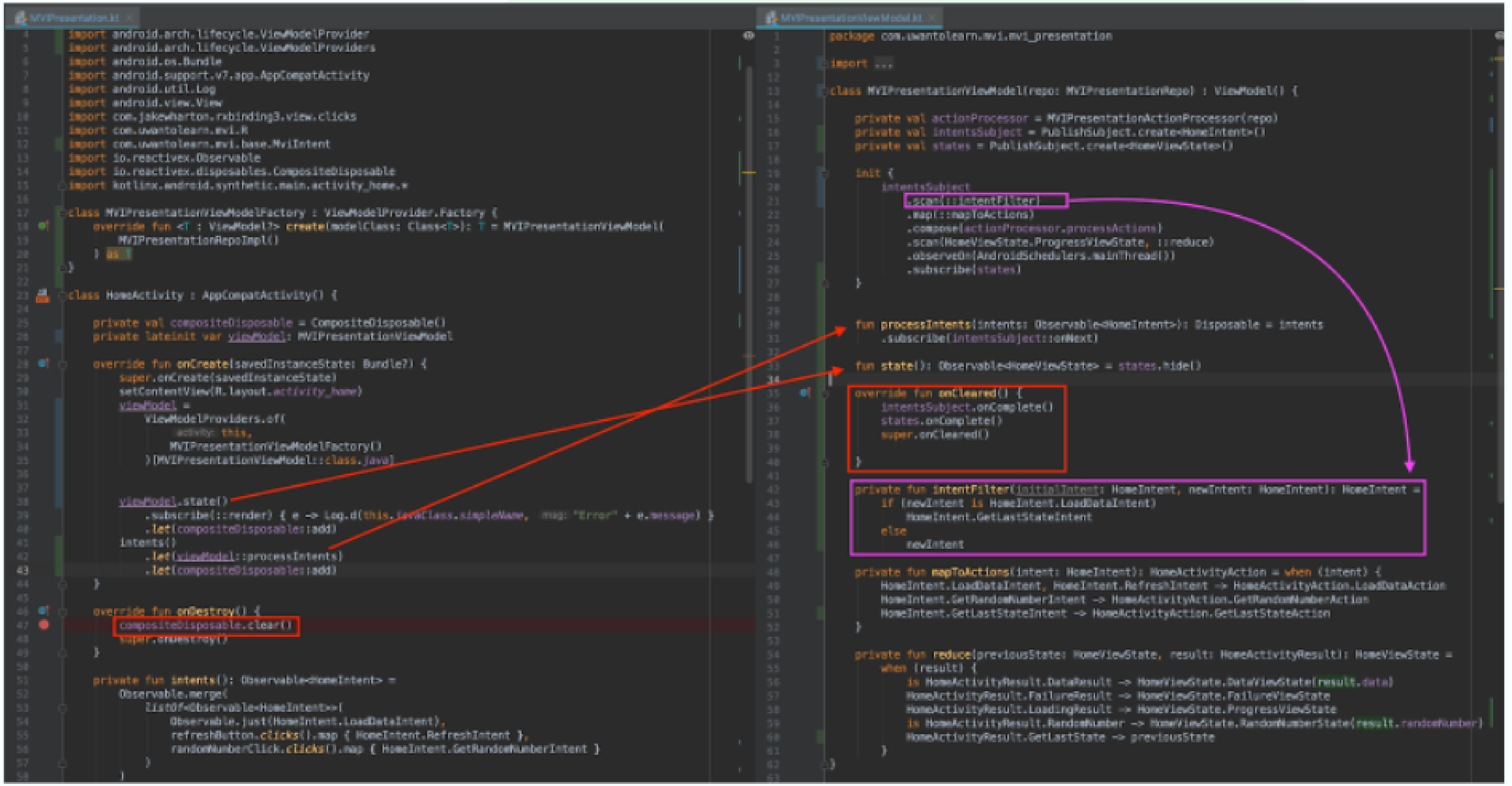
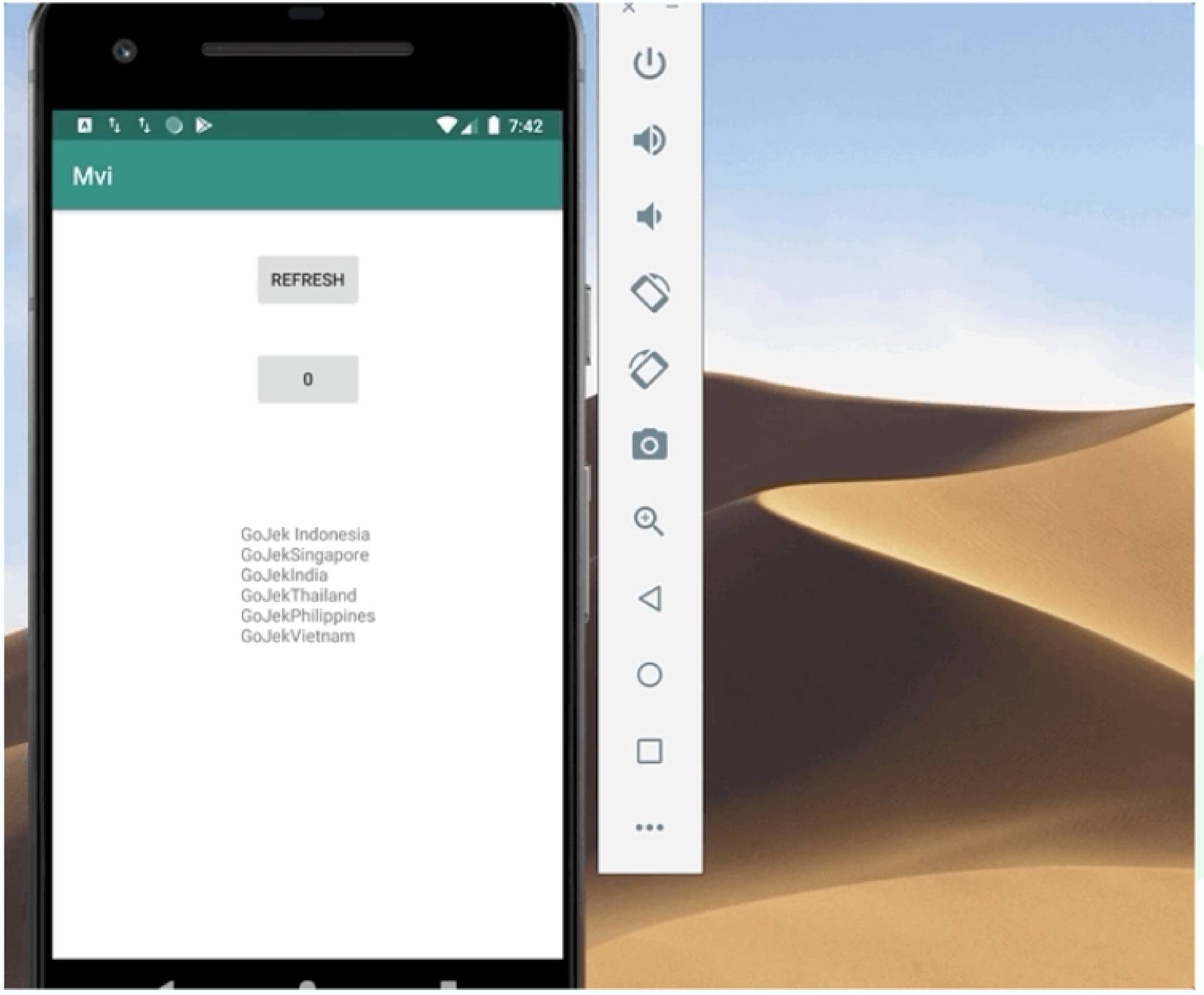


Diagram Changes



After New Code Changes

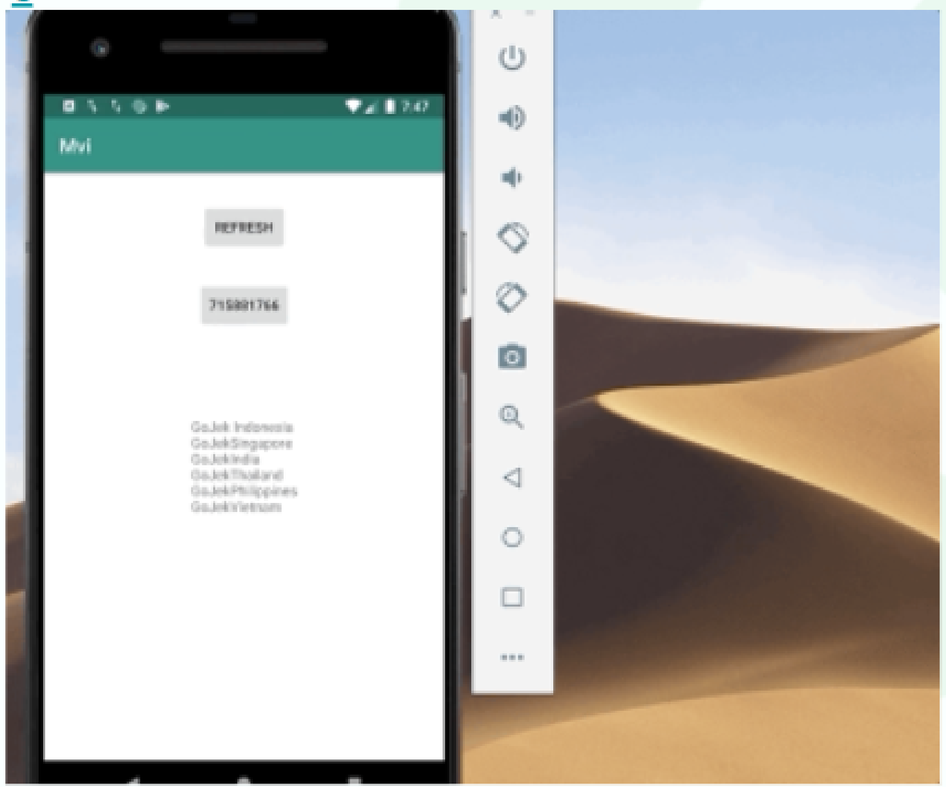



7:42
Mvi
REFRESH
0
GoJek Indonesia
GoJekSingapore
GoJekIndia
GoJekThailand
GoJekPhilippines
GoJekVietnam

Until Now

- Config Changes are working but with one issue. :(

Code:
https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-difficult-implementation-complete-with-one-issue

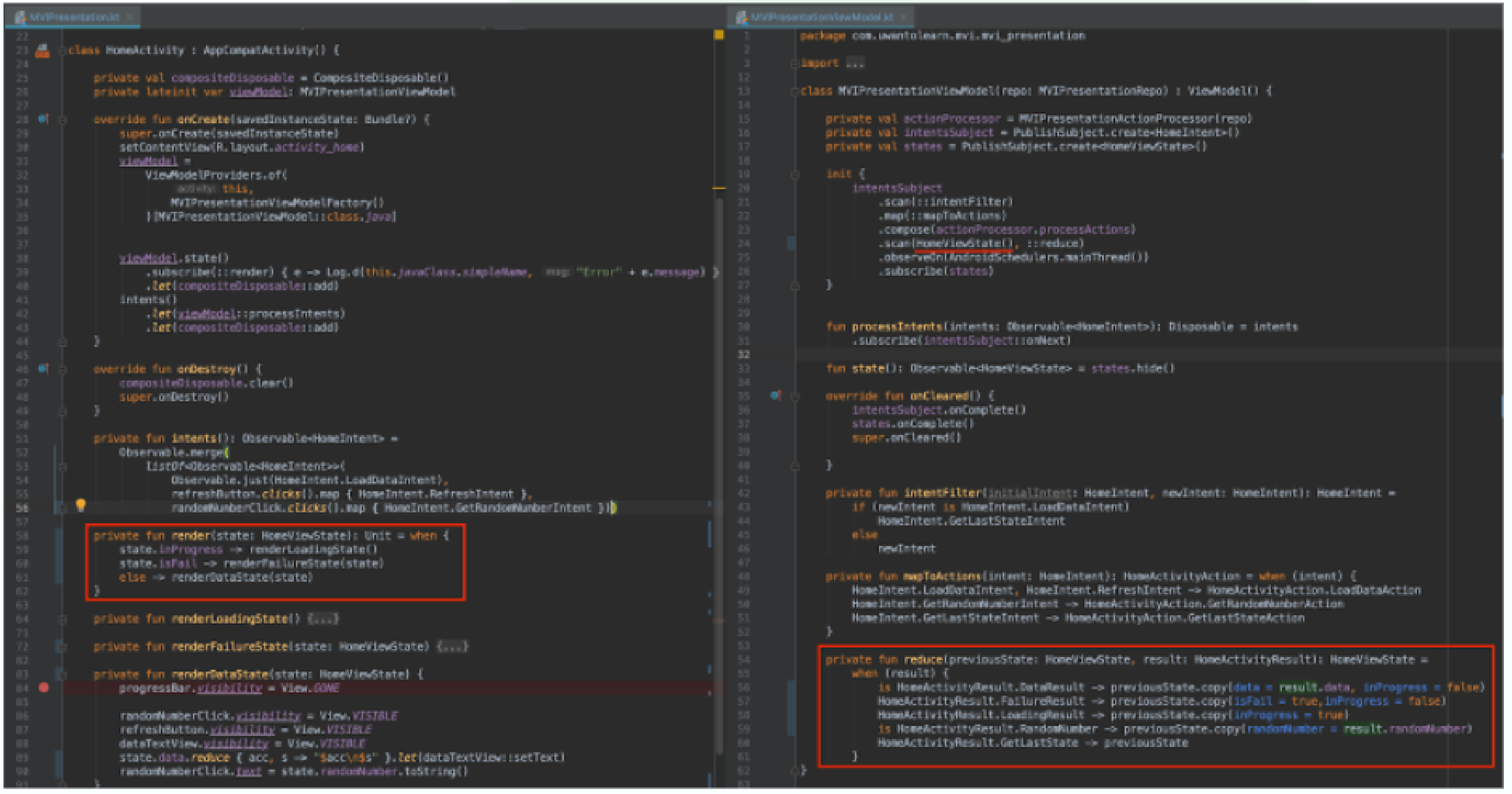


Simple solution

```
sealed class HomeViewState {
    object ProgressViewState : HomeViewState()
    object FailureViewState : HomeViewState()
    data class DataViewState(val list: List<String>) : HomeViewState()
    data class RandomNumberState(val randomNumber: Int) : HomeViewState()
}
```

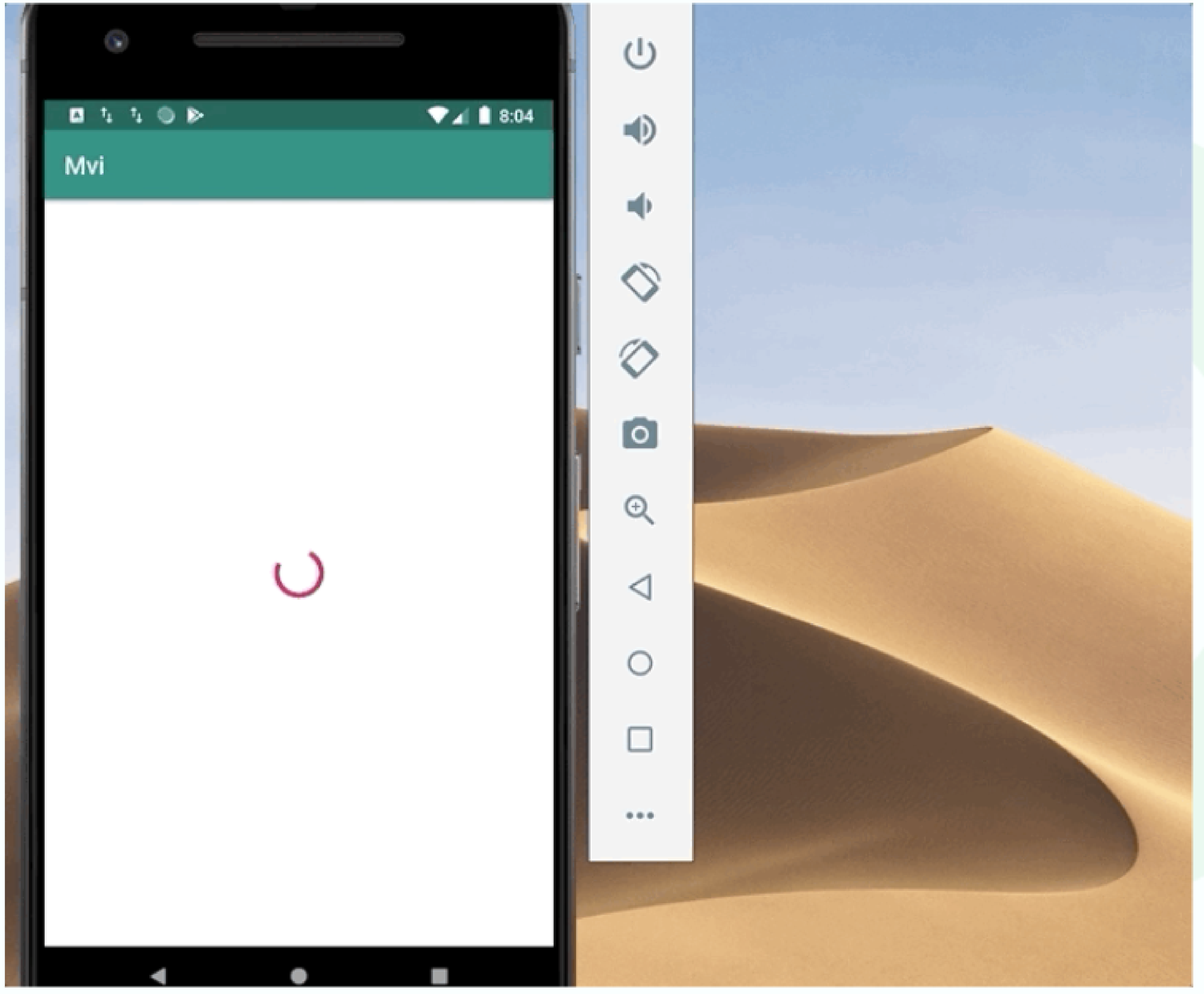
```
data class HomeViewState(
    val inProgress: Boolean,
    val isFail: Boolean,
    val data: List<String>,
    val randomNumber: Int
)
```

HomeViewState as data class



```
class HomeActivity : AppCompatActivity() {

    private val compositeDisposable = CompositeDisposable()
    private lateinit var viewModel: MVIPresentationViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home)
        viewModel =
            ViewModelProviders.of(
                activity: this,
                MVIPresentationViewModelFactory()
            ).get(MVIPresentationViewModel::class.java)


        viewModel.state()
            .subscribe(::render) { e -> Log.d(this.javaClass.simpleName,  msg: "Error" + e.message) }
            .let(compositeDisposable::add)
        intents()
            .let(viewModel::processIntents)
            .let(compositeDisposable::add)
    }

    override fun onDestroy() {
        compositeDisposable.clear()
        super.onDestroy()
    }

    private fun intents(): Observable<HomeIntent> =
        Observable.merge(
            listOf<Observable<HomeIntent>>(
                Observable.just(HomeIntent.LoadDataIntent),
                refreshButton.clicks().map { HomeIntent.RefreshIntent },
                randomNumberClick.clicks().map { HomeIntent.GetRandomNumberIntent })

    private fun render(state: HomeViewState): Unit = when {
        state.inProgress -> renderLoadingState()
        state.isFail -> renderFailureState(state)
        else -> renderDataState(state)
    }

    private fun renderLoadingState() {...}

    private fun renderFailureState(state: HomeViewState) {...}

    private fun renderDataState(state: HomeViewState) {
        progressBar.visibility = View.GONE

        randomNumberClick.visibility = View.VISIBLE
        refreshButton.visibility = View.VISIBLE
        dataTextView.visibility = View.VISIBLE
        state.data.reduce { acc, s -> "$acc\n$s" }.let(dataTextView::setText)
        randomNumberClick.text = state.randomNumber.toString()
    }
```

```
package com.uwantolearn.mvi.mvi_presentation

import ...

class MVIPresentationViewModel(repo: MVIPresentationRepo) : ViewModel() {

    private val actionProcessor = MVIPresentationActionProcessor(repo)
    private val intentsSubject = PublishSubject.create<HomeIntent>()
    private val states = PublishSubject.create<HomeViewState>()

    init {
        intentsSubject
            .scan(::intentFilter)
            .map(::mapToActions)
            .compose(actionProcessor.processActions)
            .scan(HomeViewState(), ::reduce)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(states)
    }


    fun processIntents(intents: Observable<HomeIntent>): Disposable = intents
        .subscribe(intentsSubject::onNext)

    fun state(): Observable<HomeViewState> = states.hide()

    override fun onCleared() {
        intentsSubject.onComplete()
        states.onComplete()
        super.onCleared()

    }

    private fun intentFilter(initialIntent: HomeIntent, newIntent: HomeIntent): HomeIntent =
        if (newIntent is HomeIntent.LoadDataIntent)
            HomeIntent.GetLastStateIntent
        else
            newIntent

    private fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {
        HomeIntent.LoadDataIntent, HomeIntent.RefreshIntent -> HomeActivityAction.LoadDataAction
        HomeIntent.GetRandomNumberIntent -> HomeActivityAction.GetRandomNumberAction
        HomeIntent.GetLastStateIntent -> HomeActivityAction.GetLastStateAction
    }

    private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
        when (result) {
            is HomeActivityResult.DataResult -> previousState.copy(data = result.data, inProgress = false)
            HomeActivityResult.FailureResult -> previousState.copy(isFail = true,inProgress = false)
            HomeActivityResult.LoadingResult -> previousState.copy(inProgress = true)
            is HomeActivityResult.RandomNumber -> previousState.copy(randomNumber = result.randomNumber)
            HomeActivityResult.GetLastState -> previousState
        }
}
```

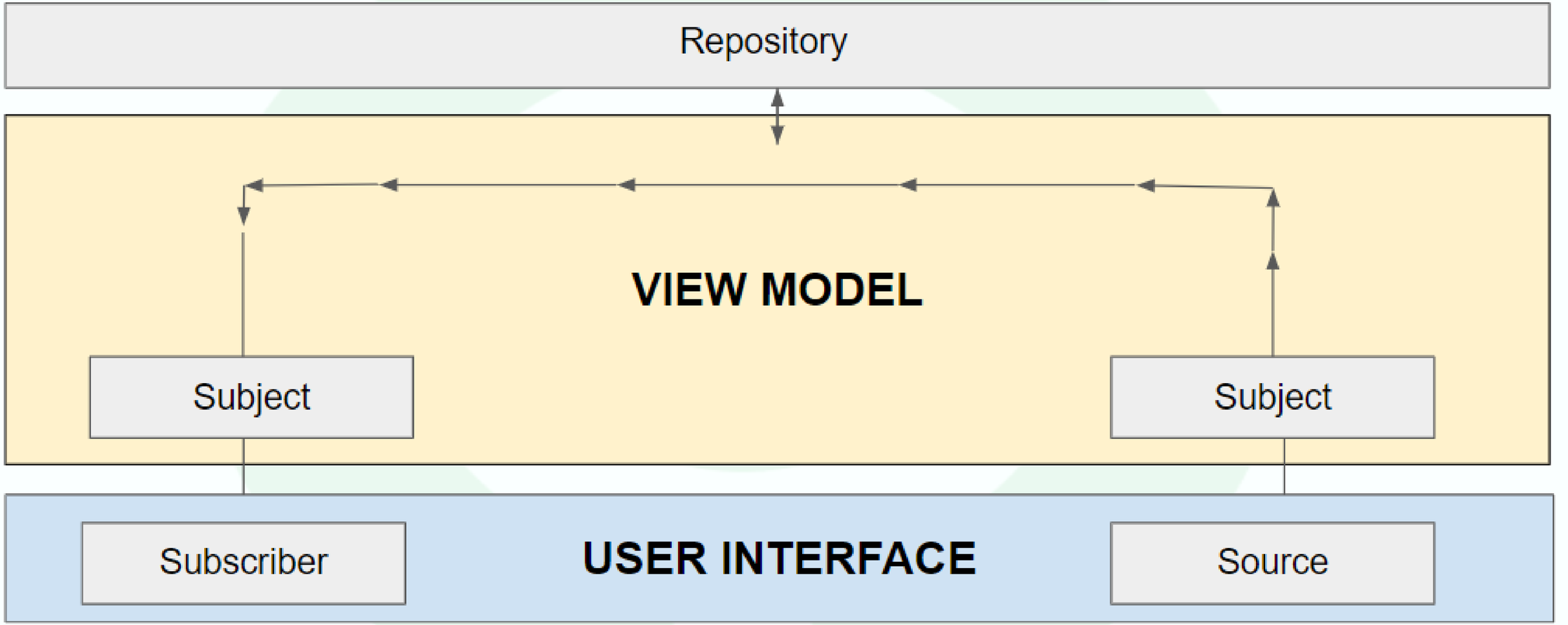

8:04
Mvi

Until Now

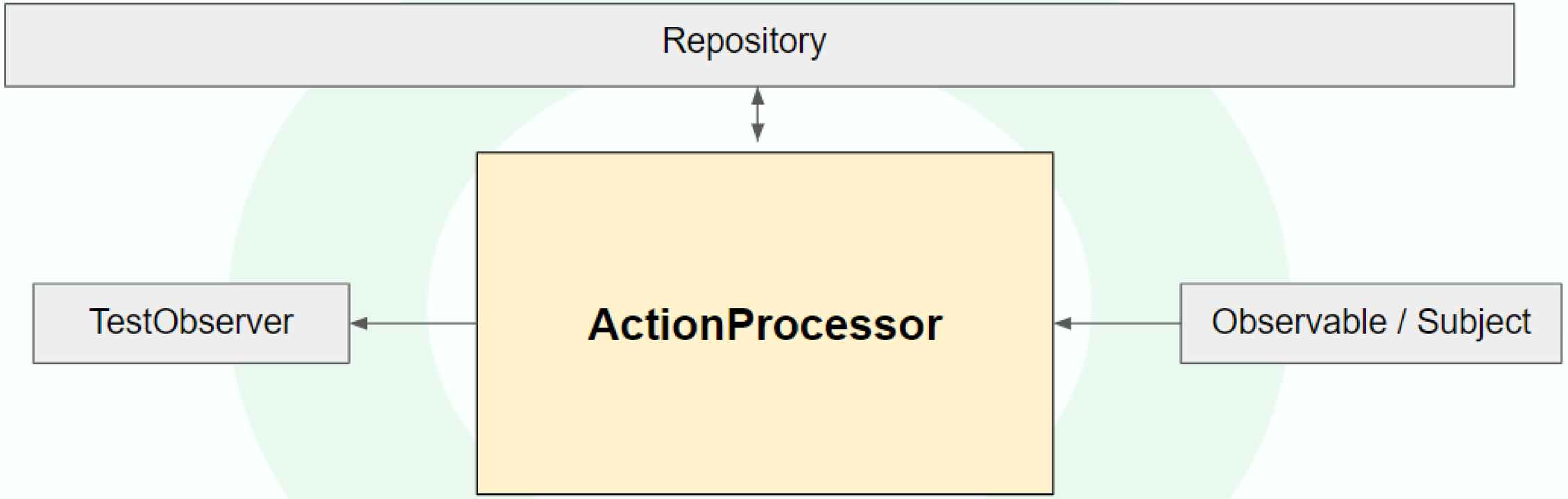
- We achieved everything except proper framework api

Code: https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-difficult-implementation-complete

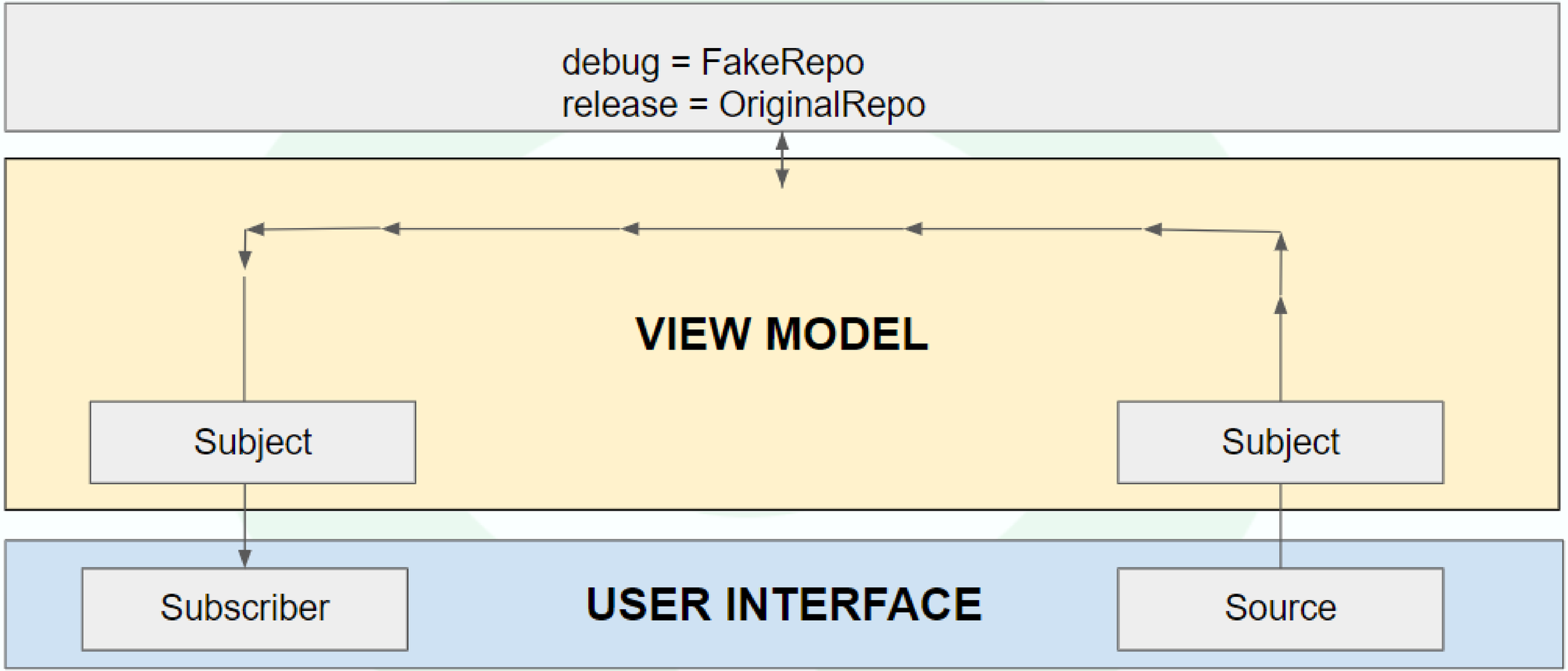
Before Framework API what about testing



```
class MVIPresentationViewModelTest {
    @Mock
    lateinit var repo: MVIPresentationRepo
    lateinit var actionProcessor: MVIPresentationActionProcessor
    lateinit var viewModel: MVIPresentationViewModel
    lateinit var testObserver: TestObserver<HomeViewState>

    @Test
    fun `when LoadDataIntent successful should return data with`() {
        repo.loadData()
            .let(Mockito::`when`)
            .thenReturn(just(mockData))

        HomeIntent.LoadDataIntent
            .let(::just)
            .cast(HomeIntent::class.java)
            .let(viewModel::processIntents)

        testObserver.values()
            .let(::println)
        testObserver.assertValueAt( index: 0, HomeViewState::inProgress)
        testObserver.assertValueAt( index: 1) { !it.inProgress }
        testObserver.assertValueAt( index: 1) { it.data == mockData }
    }

    @Test
    fun `when LoadDataIntent fail should return failure state`() {
        repo.loadData()
            .let(Mockito::`when`)
            .thenReturn(Observable.error(Exception()))

        HomeIntent.LoadDataIntent
            .let(::just)
            .cast(HomeIntent::class.java)
            .let(viewModel::processIntents)

        testObserver.values()
            .let(::println)

        testObserver.assertValueAt( index: 0, HomeViewState::inProgress)
        testObserver.assertValueAt( index: 1) { !it.inProgress }
        testObserver.assertValueAt( index: 1, HomeViewState::isFail)

    }

    @Test
    fun `when RandomNumber click should return value`() {
        HomeIntent.GetRandomNumberIntent
            .let(::just)
            .cast(HomeIntent::class.java)
            .let(viewModel::processIntents)

        testObserver.values()
            .let(::println)

        testObserver.assertValueAt( index: 0) { it.randomNumber != 0 }

    }
```


Repository
TestObserver
ActionProcessor
Observable / Subject

```kotlin
class MVIPresentationActionProcessorTest {

    @Mock
    lateinit var repo: MVIPresentationRepo
    lateinit var actionProcessor: MVIPresentationActionProcessor
    private val mockData = listOf(...)

    companion object {...}

    @Before
    fun setUp() {...}

    @Test
    fun `when LoadDataAction success should return DataResult`() {
        repo.loadData()
            .let(Mockito::`when`)
            .thenReturn(mockData.let(::just))

        val testObserver = HomeActivityAction.LoadDataAction.let(::just)
            .compose(actionProcessor.processActions)
            .test()

        testObserver.values()
            .let(::println)
        testObserver.assertValueAt( index: 0, HomeActivityResult.LoadingResult)
        testObserver.assertValueAt( index: 1) { it == HomeActivityResult.DataResult(mockData) }
    }


    @Test
    fun `when LoadDataAction failed should return FailedResult`() {
        repo.loadData()
            .let(Mockito::`when`)
            .thenReturn(error(Exception()))

        val testObserver = HomeActivityAction.LoadDataAction.let(::just)
            .compose(actionProcessor.processActions)
            .test()

        testObserver.values()
            .let(::println)
        testObserver.assertValueAt( index: 0, HomeActivityResult.LoadingResult)
        testObserver.assertValueAt( index: 1, HomeActivityResult.FailureResult)
    }
}
```


debug = FakeRepo
release = OriginalRepo
VIEW MODEL
Subject
Subject
Subscriber
USER INTERFACE
Source

```kotlin
@RunWith(AndroidJUnit4::class)
class HomeActivityTest {

    @get:Rule
    var homeActivityTestRule = ActivityTestRule(
        HomeActivity::class.java,
        initialTouchMode: true,
        launchActivity: true
    )

    @Test
    fun whenLaunchShouldShowProgressBar() {
        onView(withId(R.id.progressBar))
            .check(matches(isDisplayed()))

        withId(R.id.progressBar)
            .let(::onView)
            .check(matches(isDisplayed()))
    }

    @Test
    fun whenRandomButtonClickShouldChangeButtonValue() {

        withId(R.id.progressBar)
            .let(::onView)
            .check(matches(isDisplayed()))
        Thread.sleep( millis: 4000)

        withId(R.id.randomNumberClick)
            .let(::onView)
            .check(matches(withText( text: "0")))

        withId(R.id.randomNumberClick)
            .let(::onView)
            .perform(ViewActions.click())

        withId(R.id.randomNumberClick)
            .let(::onView)
            .check(matches(not(withText( text: "0"))))

    }

}
```

Framework API

MviIntent.kt

```kotlin
package com.uwantolearn.mvi.base

interface MviIntent

```

MviAction.kt

Close. Alt-click to close others

```kotlin
n.mvi.base

interface MviAction
```

MviResult.kt

```kotlin
package com.uwantolearn.mvi.base

interface MviResult
```

MviViewState.kt

```kotlin
package com.uwantolearn.mvi.base

interface MviViewState
```

MviView.kt

```kotlin
package com.uwantolearn.mvi.base

import io.reactivex.Observable


interface MviView<I : MviIntent, S : MviViewState> {
    fun intents(): Observable<I>
    fun render(state: S)
}
```

MviViewModel.kt

```kotlin
package com.uwantolearn.mvi.base

import ...


interface MviViewModel<I : MviIntent, S : MviViewState> {
    fun processIntents(intents: Observable<I>) : Disposable
    fun state(): Observable<S>
}
```

After applying API's

```
sealed class HomeIntent : MviIntent {
    object LoadDataIntent : HomeIntent()
    object RefreshIntent : HomeIntent()
    object GetRandomNumberIntent : HomeIntent()
    object GetLastStateIntent : HomeIntent()
}
```

```
data class HomeViewState(
    val inProgress: Boolean = true,
    val isFail: Boolean = false,
    val data: List<String> = listOf(),
    val randomNumber: Int = 0
) : MviViewState
```

```
sealed class HomeActivityResult : MviResult {
    data class DataResult(val data: List<String>) : HomeActivityResult()
    data class RandomNumber(val randomNumber: Int) : HomeActivityResult()
    object GetLastState : HomeActivityResult()
    object FailureResult : HomeActivityResult()
    object LoadingResult : HomeActivityResult()
}
```

```
sealed class HomeActivityAction : MviAction {
    object LoadDataAction : HomeActivityAction()
    object GetRandomNumberAction : HomeActivityAction()
    object GetLastStateAction : HomeActivityAction()
}
```

After applying API's

```
class HomeActivity : MviView<HomeIntent, HomeViewState>, AppCompatActivity() {

    private val compositeDisposable = CompositeDisposable()
    private lateinit var viewModel: MVIPresentationViewModel

    override fun onCreate(savedInstanceState: Bundle?) {...}

    override fun onDestroy() {...}

    override fun intents(): Observable<HomeIntent> =
        Observable.merge(
            listOf<Observable<HomeIntent>>(
                Observable.just(HomeIntent.LoadDataIntent),
                refreshButton.clicks().map { HomeIntent.RefreshIntent },
                randomNumberClick.clicks().map { HomeIntent.GetRandomNumberIntent })
        )

    override fun render(state: HomeViewState): Unit = when {
        state.inProgress -> renderLoadingState()
        state.isFail -> renderFailureState(state)
        else -> renderDataState(state)
    }

    private fun renderLoadingState() {...}
    private fun renderFailureState(state: HomeViewState) {...}
    private fun renderDataState(state: HomeViewState) {...}
}
```

```
class MVIPresentationViewModel(repo: MVIPresentationRepo) : ViewModel(),
    MviViewModel<HomeIntent, HomeViewState> {

    private val actionProcessor = MVIPresentationActionProcessor(repo)
    private val intentsSubject = PublishSubject.create<HomeIntent>()
    private val states = PublishSubject.create<HomeViewState>()

    init {...}

    override fun processIntents(intents: Observable<HomeIntent>): Disposable = intents
        .subscribe(intentsSubject::onNext)

    override fun state(): Observable<HomeViewState> = states.hide()

    override fun onCleared() {...}

    private fun intentFilter(initialIntent: HomeIntent, newIntent: HomeIntent): HomeIntent =
        if (newIntent is HomeIntent.LoadDataIntent)
            HomeIntent.GetLastStateIntent
        else
            newIntent

    private fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {
        HomeIntent.LoadDataIntent, HomeIntent.RefreshIntent -> HomeActivityAction.LoadDataAction
        HomeIntent.GetRandomNumberIntent -> HomeActivityAction.GetRandomNumberAction
        HomeIntent.GetLastStateIntent -> HomeActivityAction.GetLastStateAction
    }

    private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
        when (result) {
            is HomeActivityResult.DataResult -> previousState.copy(
                data = result.data,
                inProgress = false
            )
            HomeActivityResult.FailureResult -> previousState.copy(
                isFail = true,
                inProgress = false
            )
            HomeActivityResult.LoadingResult -> previousState.copy(inProgress = true)
            is HomeActivityResult.RandomNumber -> previousState.copy(randomNumber = result.randomNumber)
            HomeActivityResult.GetLastState -> previousState
        }
}
```

Until Now

- Everything is perfect.

Code:
https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-with-framework

Improve API's (For now Activity Support only)

```
abstract class MviBaseView<I : MviIntent, A : MviAction, S : MviViewState, R : MviResult> :
    MviView<I, S>, AppCompatActivity() {

    private val compositeDisposable = CompositeDisposable()
    abstract val viewModel: MviBaseViewModel<I, A, S, R>

    abstract val layoutId: Int

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(layoutId)
    }

    override fun onStart() {
        super.onStart()
        viewModel.state()
            .subscribe(::render) { e ->
                Log.d(
                    this.javaClass.simpleName,
                    msg: "Error" + e.message
                )
            }
            .let(compositeDisposable::add)
        intents()
            .let(viewModel::processIntents)
            .let(compositeDisposable::add)
    }

    override fun onStop() {
        compositeDisposable.clear()
        super.onStop()
    }
}
```

```
abstract class MviBaseViewModel<I : MviIntent, A : MviAction, S : MviViewState, R : MviResult>(
    actionProcessor: MviBaseActionProcessor<A, R>,
    initialState: S
) :
    ViewModel(), MviViewModel<I, S> {

    private val intentsSubject = PublishSubject.create<I>()
    private val states = PublishSubject.create<S>()

    init {
        intentsSubject
            .scan(::intentFilter)
            .map(::mapToActions)
            .compose(actionProcessor.processActions)
            .scan(initialState, ::reduce)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(states)
    }

    override fun processIntents(intents: Observable<I>): Disposable = intents
        .subscribe(intentsSubject::onNext)


    override fun state(): Observable<S> = states.hide()

    override fun onCleared() {
        intentsSubject.onComplete()
        states.onComplete()
        super.onCleared()
    }

    abstract fun intentFilter(initialIntent: I, newIntent: I): I

    abstract fun mapToActions(intent: I): A

    abstract fun reduce(previousState: S, result: R): S

}
```

```
abstract class MviBaseActionProcessor<A : MviAction, R : MviResult> {

    abstract val processActions: ObservableTransformer<A, R>
}
```

```
sealed class OrderIntent : MviIntent
sealed class OrderAction : MviAction
sealed class OrderResult : MviResult
data class OrderState(val orderId: Int = 0) : MviViewState
```

Implementation of a new feature by using framework api

```
class OrderActivity : MviBaseView<OrderIntent, OrderAction, OrderState, OrderResult>() {


    override val viewModel: MviBaseViewModel<OrderIntent, OrderAction, OrderState, OrderResult>
        get() = OrderViewModel()

    override
    val layoutId: Int
        get() = R.layout.activity_home

    override fun intents(): Observable<OrderIntent> {
    }

    override fun render(state: OrderState) {

    }

}
```

```
class OrderActionProcessor : MviBaseActionProcessor<OrderAction, OrderResult>() {

    override val processActions: ObservableTransformer<OrderAction, OrderResult>
        get() = TODO()

}
```

```
class OrderViewModel() : MviBaseViewModel<OrderIntent, OrderAction, OrderState, OrderResult>(
    OrderActionProcessor(),
    OrderState()
) {

    override fun intentFilter(initialIntent: OrderIntent, newIntent: OrderIntent): OrderIntent {
    }

    override fun mapToActions(intent: OrderIntent): OrderAction {
    }

    override fun reduce(previousState: OrderState, result: OrderResult): OrderState {

    }
}
```

Only write the code in given methods. The framework will take care for everything. (Not production-ready but maybe we can use :))

Benefits

- Predictable
- Debuggable (Instead QA give us steps how to fix not how to reproduce)
- Crash reports are more readable
- Onboarding is easy
- No need for fancy frameworks to unit test
- Code reviews are easy
- Design pattern with Kotlin is self-sufficient to take care issues at dev time (Live Demo)

Code:
https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-with-framework-complete-with-tests