ARCHETYPE
MODERN ANDROID ARCHITECTURE

STEPAN GONCHAROV / DENIS NEKLIUDOV

90seconds.tv

14000+ VIDEOS
1200+ BRANDS
92+ COUNTRIES


pikabu.ru

```kotlin
data class RegisterViewModelStateImpl(
        override val email: ObservableString = ObservableString(""),
        override val password: ObservableString = ObservableString("")
) : RegisterViewModelState, AutoParcelable

class RegisterViewModel(
        naviComponent: NaviComponent,
        state: RegisterViewModelState
) : ViewModel by ViewModelImpl(naviComponent, state), RegisterViewModelState by state {

    val isEmailValid = ObservableBoolean()
    val isPasswordValid = ObservableBoolean()
    val flags = listOf(isEmailValid, isPasswordValid).map { it.observe() }
    val isFormValid = ObservableBoolean()

    init {
        email.observe().setTo(isEmailValid, ::validateEmail).bindSubscribe()
        password.observe().setTo(isPasswordValid, ::validatePassword).bindSubscribe()
        combineLatest(flags) { it }
                .setTo(isFormValid) { it.all { it == true } }.bindSubscribe()
    }

    override fun args(): Args = argsOf {
        user { CreateUserModel(email.get(), password.get()) }
    }
}
```

OVERVIEW

FAQ

- - Is it clean?
  - Clean enough!
- - Why not just use clean?
  - Because someone used it already!

HISTORY FACTS

- All started in 2008
- Build for 90 Seconds app
- Stack + requirements = architecture
- 5 iterations
- RxDatabindings

COMMON PROBLEMS SOLVED

- Navigation
- Dialogs
- Network requests
- Restore state
- Testability
- Services
- Lifecycle
- Activity result
- and more

COMMON PATTERN

Observer

REFERENCE IMPLEMENTATION STACK

- Kotlin
- RxJava2
- Navi
- DataBindings

WHY KOTLIN?

KOTLIN

Archetype approach rely on Kotlin language features

Expressive

Rich but tiny stdlib

Extensions

Inlining

Delegation

Type aliases

Null-safe

KOTLIN INJECTION

```kotlin
// No lateinit var
val projectsRepo by lazyInject { projectsBaseRepo() }
```

```kotlin
// Injector == AppComponent(Dagger)
fun <T: Any> lazyInject(block: Injector.() -> T)
        = lazy { Injector().block() }
```

KOTLIN EXTENSION FUNCTIONS + DEFAULT PARAMS

```kotlin
// Typealias could be substituted for tests
typealias Args = Bundle

fun Args.episodeId() = getLong(EPISODE_ID)

fun Args.episodeId(block: () -> Long) = apply {
        putLong(EPISODE_ID, block())
}
```

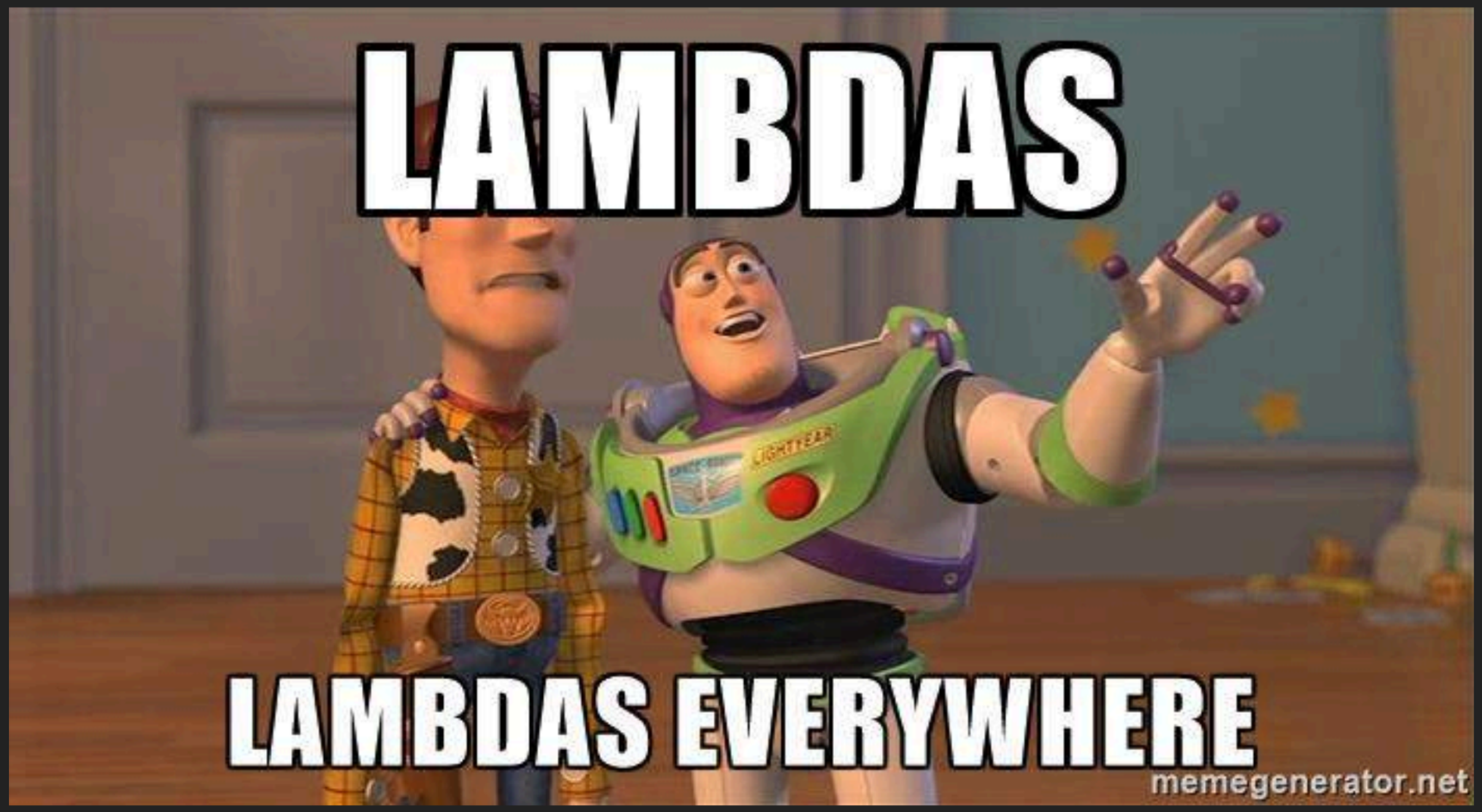
KOTLIN RX EXTENSION FUNCTIONS

```
fun <T : Any> Observable<T>.bindSubscribe(
        scheduler: Scheduler = io(),
        onNext: (T) -> Unit = {},
        onError: (Throwable) -> Unit = {toaster.showToast( “error” )},
        onComplete: () -> Unit = {}
) = subscribeOn(scheduler)
    .subscribe(onNext, onError, onComplete)
    .bind()


episodesComponent.observeEpisodes()
                 .setTo(episodes)
                 .bindSubscribe()
```

FIGHT INHERITANCE WITH CLASS DELEGATION

```
interface EpisodesModelRepo : KeyValueRepo<Long, EpisodesModel> {
    fun pull() = EpisodesRequest().operation()
}

class InMemoryEpisodesRepo : EpisodesModelRepo,
    KeyValueRepo<...> by InMemoryKeyValueRepo<Long, EpisodesModel>()
```

KOTLIN LAMBDAS



WHY RXJAVA?

RX DATA TRANSFORMATION

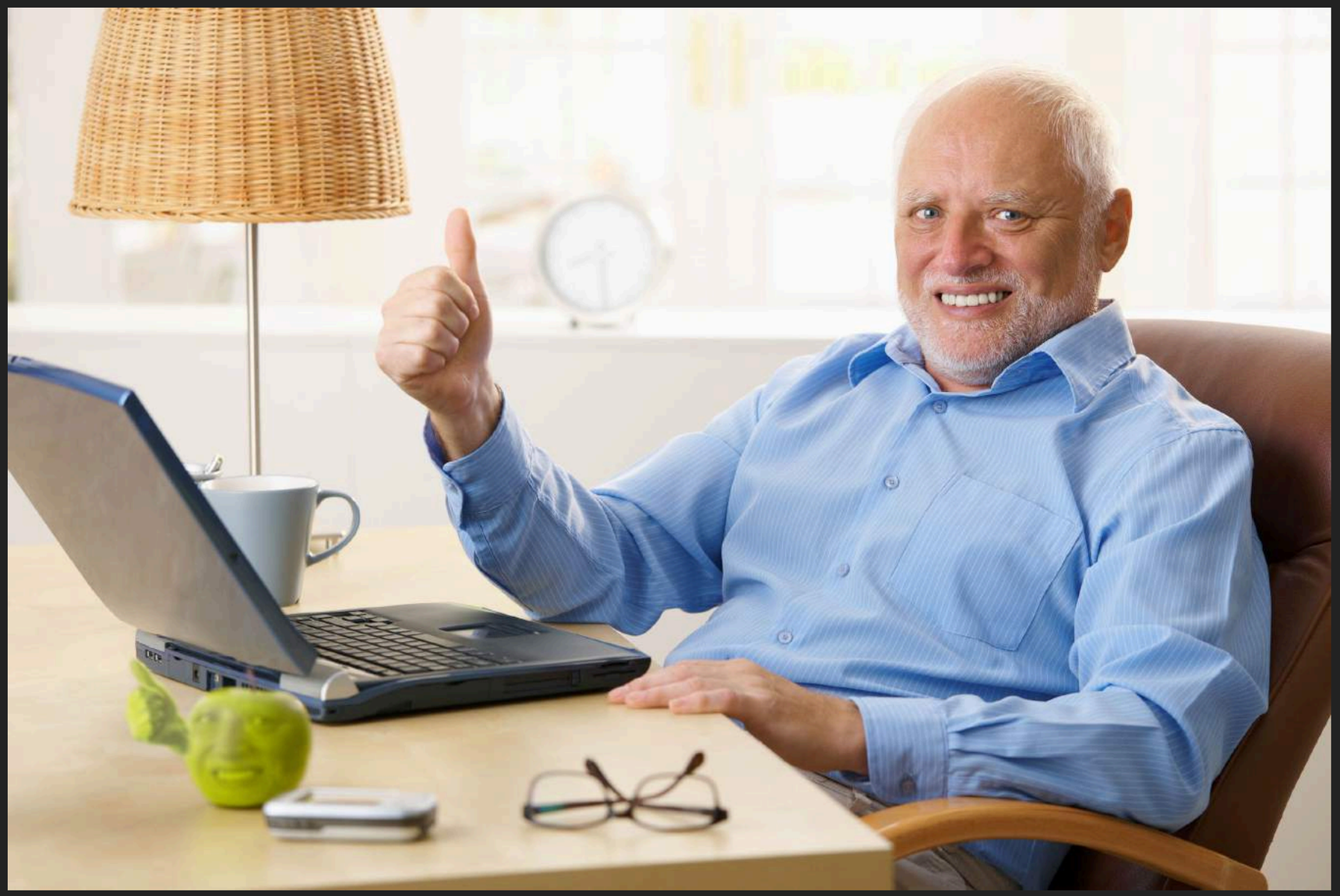
```
fun operation(): Single<List<EpisodesModel>> = api.feed()
        .map { it.channel.item }
        .map {
            it.map { TitleDescWrapper(it.title, it.description) }
        }
```

RX DETACH

```
interface CompositeDisposableComponent {
    val composite: CompositeDisposable

    fun Disposable.bind() = composite.add(this)

    fun resetCompositeDisposable() {
        composite.clear()
    }
}
```

RX+DATA BINDING THREADS

Before all view updates on UI thread



RX+DATA BINDING THREADS

Now we change values of observing fields from any thread

WHY DATABINDINGS?

DATA BINDINGS BENEFITS

- No more findViewById()
- Clean Activity, Fragment and View classes
- State handling
- Observing changes, not pushing to UI

DATA BINDING LAYOUT HEADER

```
<layout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:bind="http://schemas.android.com/apk/res-auto"
    >

<data>

    <variable
        name="vm"
        type="com.stepango.archetype.player.ui.episodes.EpisodesViewModel"
        />

</data>
```

DATA BINDING LAYOUT HEADER

```
<variable
      name="vm"
      type="com.stepango.archetype.player.ui
                        .episodes.EpisodesViewModel"
/>
```

DATA BINDING LAYOUT

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >

    <include
        layout="@layout/include_app_bar"
        bind:title="@{@string/episodes_title}"
        />

    <android.support.v7.widget.RecyclerView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        bind:episodesAdapter="@{vm.episodes}"
        bind:setHasFixedSize="@{true}"
        bind:useDefaults="@{true}"
        />

</LinearLayout>
```

DATA BINDING LAYOUT

```
<include
    layout="@layout/include_app_bar"
bind:title="@{@string/episodes_title}"
/>
```

DATA BINDING LAYOUT

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >

    <include
        layout="@layout/include_app_bar"
        bind:title="@{@string/episodes_title}"
        />

    <android.support.v7.widget.RecyclerView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        bind:episodesAdapter="@{vm.episodes}"
        bind:setHasFixedSize="@{true}"
        bind:useDefaults="@{true}"
        />

</LinearLayout>
```

DATA BINDING LAYOUT

```
<android.support.v7.widget.RecyclerView
    android:layout_width="match_parent"
android:layout_height="match_parent"
bind:episodesAdapter="@{vm.episodes}"
bind:setHasFixedSize="@{true}"
bind:useDefaults="@{true}"
/>
```

DATA BINDING VIEWMODEL

```
class EpisodesViewModel(
        naviComponent: NaviComponent
) : ViewModel by ViewModelImpl(naviComponent) {

    val episodesComponent by lazyInject { episodesComponent() }

    val episodes = ObservableField<List<EpisodesWrapper>>(listOf())

    init {
        refreshItems()
        episodesComponent.observeEpisodes()
                .setTo(episodes)
                .bindSubscribe()
    }

     fun refreshItems() {
        episodesComponent.updateEpisodes()
                .bindSubscribe(
                        onError = { toaster.showError(it, R.string.episodes_error_loading) }
                )
    }
}
```

DATA BINDING VIEWMODEL

```
class EpisodesViewModel(
        naviComponent: NaviComponent
) : ViewModel by ViewModelImpl(naviComponent) {
```

DATA BINDING VIEWMODEL

```
class EpisodesViewModel(
        naviComponent: NaviComponent
) : ViewModel by ViewModelImpl(naviComponent) {

    val episodesComponent by lazyInject { episodesComponent() }

    val episodes = ObservableField<List<EpisodesWrapper>>(listOf())

    init {
        refreshItems()
        episodesComponent.observeEpisodes()
                .setTo(episodes)
                .bindSubscribe()
    }

     fun refreshItems() {
        episodesComponent.updateEpisodes()
                .bindSubscribe(
                        onError = { toaster.showError(it, R.string.episodes_error_loading) }
                )
    }
}
```

DATA BINDING VIEWMODEL

```
val episodesComponent by lazyInject { episodesComponent() }

val episodes = ObservableField<List<EpisodesWrapper>>(listOf())
```

DATA BINDING VIEWMODEL

```
class EpisodesViewModel(
        naviComponent: NaviComponent
) : ViewModel by ViewModelImpl(naviComponent = naviComponent) {

    val episodesComponent by lazyInject { episodesComponent() }
    val episodes = ObservableField(listOf<EpisodeListItemWrapper>())

    init {
        refresh()
        display()
    }

    fun display() = episodesComponent.observeEpisodes().setTo(episodes).bindSubscribe()

    fun refresh() {
        episodesComponent.updateEpisodes().bindSubscribe(
                onComplete = { hideLoader() },
                onError = { toaster.showError(it, R.string.episodes_error_loading) })
    }
}
```

DATA BINDING VIEWMODEL

```
init {
    refresh()
    display()
}
```

DATA BINDING VIEWMODEL

```
class EpisodesViewModel(
        naviComponent: NaviComponent
) : ViewModel by ViewModelImpl(naviComponent = naviComponent) {

    val episodesComponent by lazyInject { episodesComponent() }
    val episodes = ObservableField(listOf<EpisodeListItemWrapper>())

    init {
        refresh()
        display()
    }

    fun display() = episodesComponent.observeEpisodes().setTo(episodes).bindSubscribe()

    fun refresh() {
        episodesComponent.updateEpisodes().bindSubscribe(
                onComplete = { hideLoader() },
                onError = { toaster.showError(it, R.string.episodes_error_loading) })
    }
}
```

DATA BINDING VIEWMODEL

```
fun refresh() {
    episodesComponent.updateEpisodes().bindSubscribe(
            onComplete = { hideLoader() },
            onError = {
                toaster.showError(it, R.string.episodes_error_loading)
            }
    )
}
```

DATA BINDING WITH LAST ADAPTER

```
@BindingAdapter("episodesAdapter")
fun episodesAdapter(view: RecyclerView, list: List<EpisodesWrapper>) {
    LastAdapter.with(list)
            .map<ItemEpisodeBinding>(R.layout.item_episode)
            .swap(view)
}
```

DATA BINDING ACTIONS

```
<variable
    name="item"
    type="com.stepango.archetype.player.data.wrappers.EpisodesWrapper"
    />
```

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    bind:performAction="@{R.id.action_show_episode}"
    bind:actionData="@{item::args}"
    >
```

DATA BINDING ACTIONS

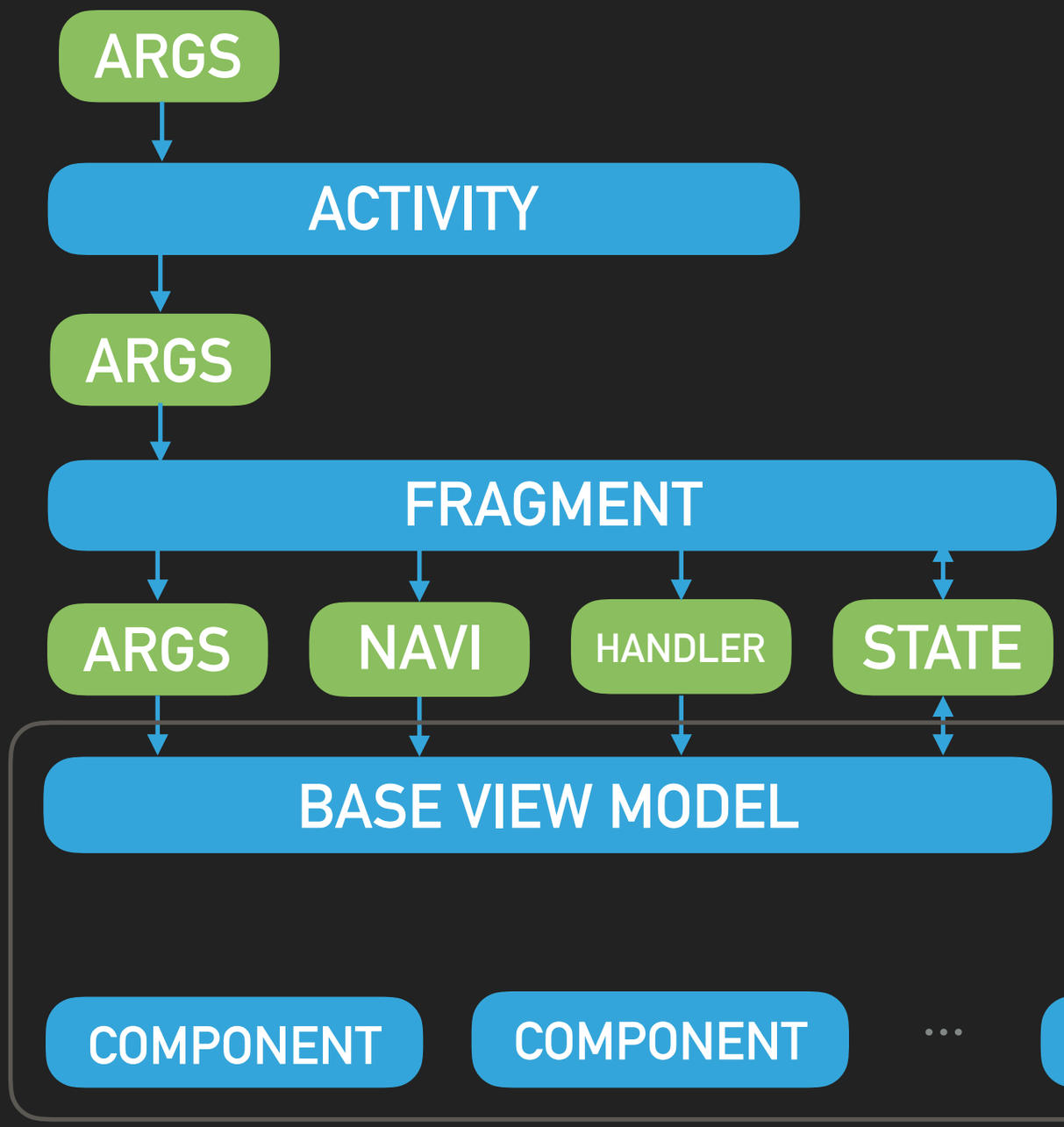
```
bind:performAction="@{R.id.action_show_episode}"
bind:actionData="@{item::args}"
```

BUILDING BLOCKS

BLOCKS

- View(DataBinding layout)
- Args
- Repo
- Pure/extension functions
- Component
- Action
- ViewModel + State

} Decoupled business logic

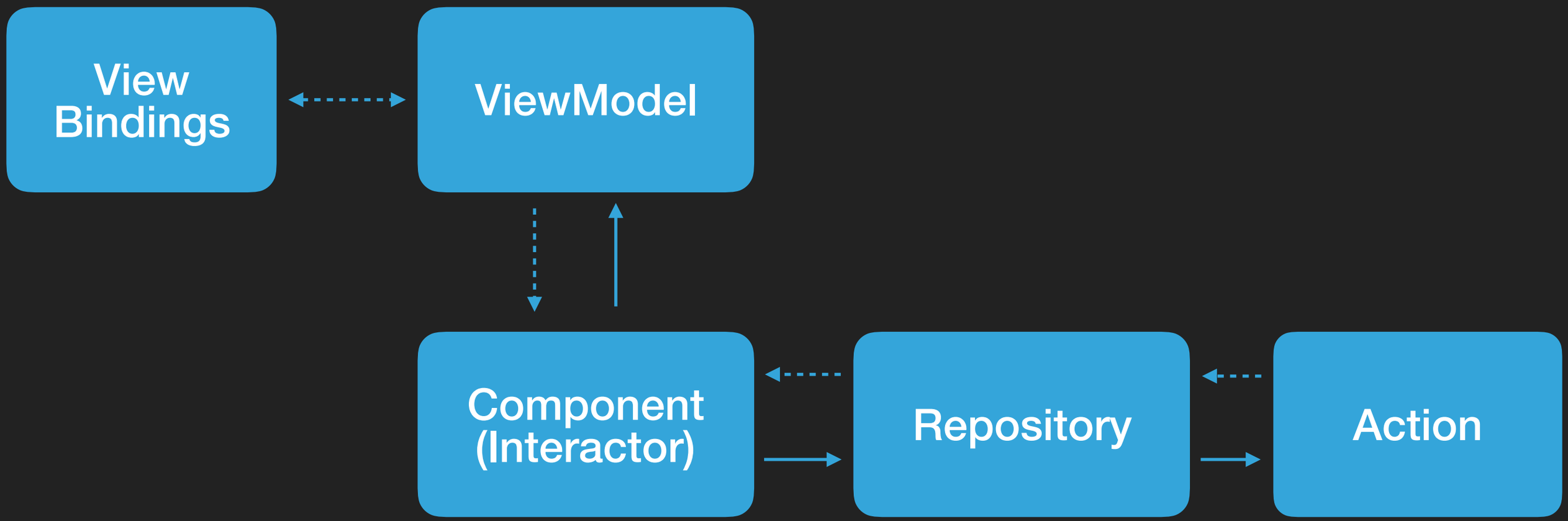


NAVI - Lifecycle callbacks

ARGS - Universal serialisable arguments

Fragment could pass no data to VM

BASIC FLOW


View
Bindings
ViewModel
Component
(Interactor)
Repository
Action

VIEWMODEL

BASE VIEW MODEL

```
class ViewModelImpl(
        naviComponent: NaviComponent,
        event: Event<*> = Event.DETACH,
        inline val args: Args = argsOf(),
        inline val state: Parcelable = object : AutoParcelable {}
) :
        ViewModel,
        NaviComponent by naviComponent,
        CompositeDisposableComponent by CompositeDisposableComponentImpl()
{

    override fun args(): Args = args

    init {
        observe(event).bindSubscribe(onNext = { resetCompositeDisposable() })
        observe(Event.SAVE_INSTANCE_STATE).bindSubscribe(onNext = { it.putState(state) })
    }

    fun <T : Any> NaviComponent.observe(event: Event<T>) = RxNavi.observe(this, event)
}
```

STATE RESTORING

```
data class RegisterViewModelState {
        override val email: ObservableStrin
        override val password: ObservableString
}

data class RegisterViewModelStateImpl(
        override val email: ObservableString = ObservableString(""),
        override val password: ObservableString = ObservableString("")
) : RegisterViewModelState, AutoParcelable

class RegisterViewModel(
        naviComponent: NaviComponent,
        state: RegisterViewModelState
) :
        ViewModel by ViewModelImpl(naviComponent, state),
        RegisterViewModelState by state
```

STATE RESTORING

```
class LoginFragment : BaseFragment<LoginBinding>() {
    override fun initBinding(binding: LoginBinding, state: Bundle?) {

        val state = state.extract { LoginViewModelStateImpl() }

        binding.vm = LoginViewModel(this, state)
    }
}
```

STATE RESTORING

```
class EpisodesViewModel(
        naviComponent: NaviComponent,
        state: EpisodesState
) :
        ViewModel by ViewModelImpl(naviComponent, state),
        EpisodesState by state { ... }
```

REPOSITORY

FLOW

Notify about changes

API DATA RETRIEVAL

```
class EpisodesComponentImpl : EpisodesComponent {

    val episodesRepo by lazyInject { episodesRepo() }

    override fun updateEpisodes(): Single<List<EpisodesModel>>
            = episodesRepo.pull()
}
```

API DATA RETRIEVAL

```
interface EpisodesModelRepo : KeyValueRepo<Long, EpisodesModel> {

    val actionProducer: ActionProducer<ApiAction>
    val apiService: Api

    fun pull(): Completable = actionProducer
            .createAction(R.id.action_get_episodes)
            .invoke(apiService, argsOf())
}
```

API DATA RETRIEVAL

```
class GetEpisodesAction : ApiAction {

    val episodesRepo by lazyInject { episodesRepo() }

    override fun invoke(context: Api, args: Args): Completable = context.feed()
                .map { it.channel.item.map(::transformResponse) }
                .flatMap { episodesRepo.save(it.associateBy({ it.id }) { it }) }
                .subscribeOn(io())
                .toCompletable()

}
```

OBSERVING CHANGES OF CACHE

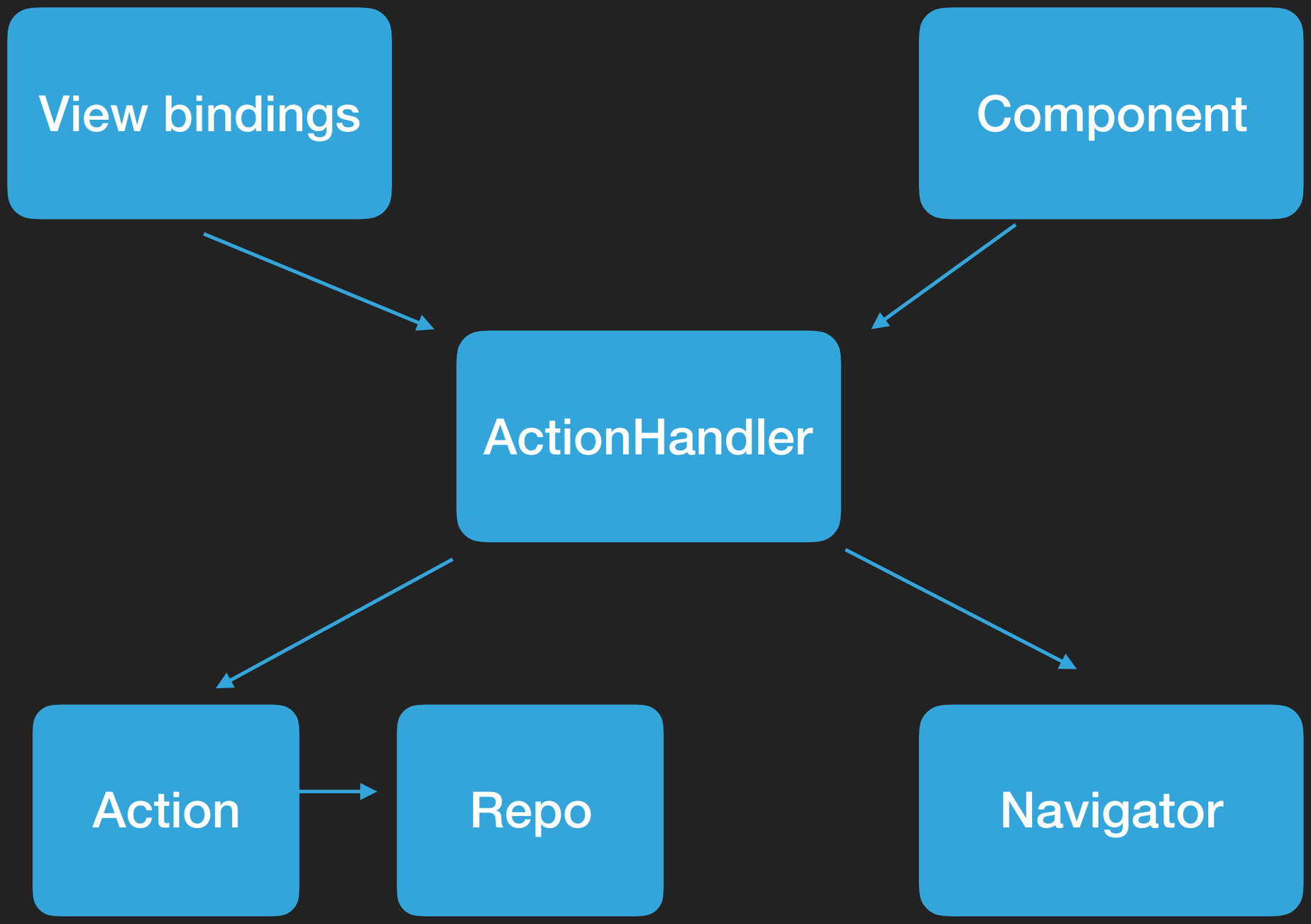
```
class EpisodesComponentImpl : EpisodesComponent {

    private val episodesRepo by lazyInject { episodesRepo() }

    override fun observeEpisodes(): Observable<List<EpisodesWrapper>>
                // observeWindow(offset, limit)
                = episodesRepo.observeAll()
                .map { it.map(::EpisodesWrapper) }
}
```

ACTIONS

ACTIONS

- ID based
- Isolated(allowed to access platform specific code)
- Replaceable
- Async
- Serialisable
- Call declared in xml
- Call for code

BASIC FLOW


View bindings
Component
ActionHandler
Action
Repo
Navigator

ACTION IS SIMPLE

```
interface Action<in T> {
    fun isDisposable() = true
    fun invoke(context: T, args: Args): Completable
}

interface ActionProducer<out T : Action<*>> {
    fun createAction(actionId: Int): T
}
```

ACTION HANDLERS

```
interface ActionHandler<in T> {
    fun handleAction(context: T, actionId: Int, args: Args = argsOf())
    fun stopActions(): Completable
}

interface BaseActionHandler {
    //val context: T
    fun handleAction(actionId: Int, args: Args = argsOf())
    fun stopActions(): Completable
}
```

SIMPLE ACTION EXAMPLES

```
class IdleAction : Action<Nothing> {
    override fun invoke(context: Nothing, args: Args)
            = Completable.complete()
}
```

```
class RequestUserAction : Action<Nothing> {
    override fun invoke(context: Nothing, args: Args) = injector
        .userRepo()
        .pullUser(args.userId) // Save to repo happens here
        .toCompletable()
}
```

NAVIGATION ACTION EXAMPLE

```
class OpenSettingsAction : Action<Activity> {
    override fun invoke(context: Activity, args: Args)
        = Completable.fromCallable {
        context.startActivity<SettingsActivity>(args)
    }
}
```

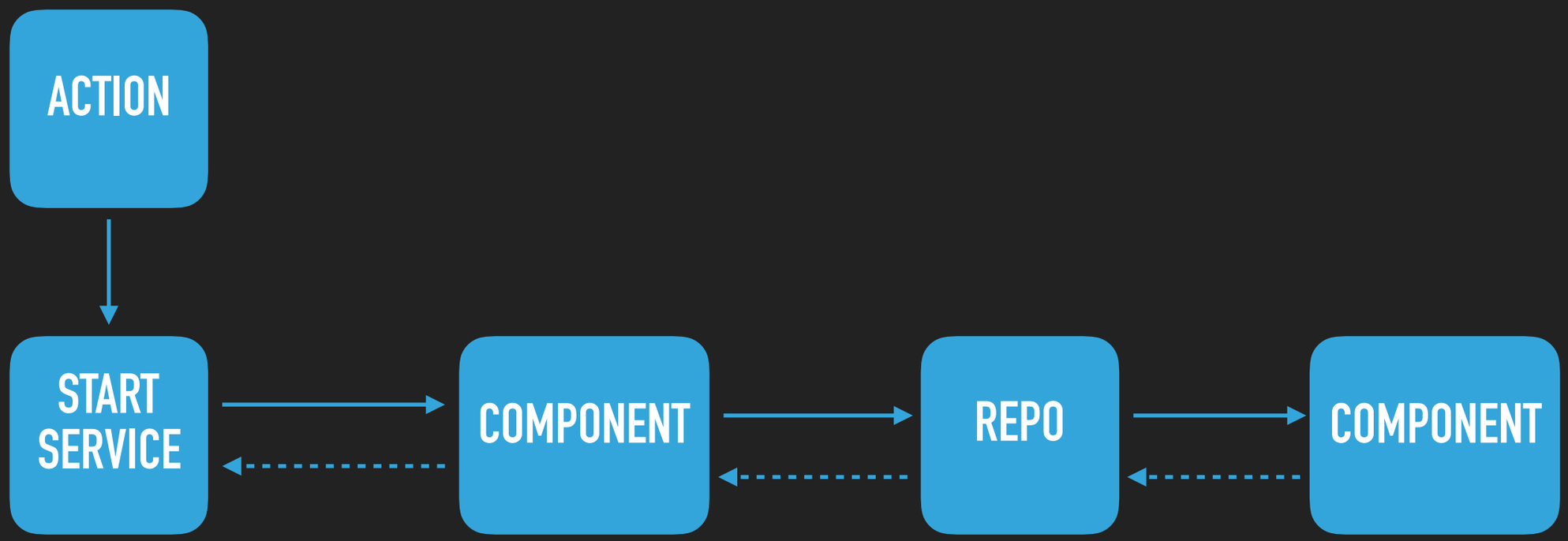
```
class GetImageAction : Action<Activity> {
    override fun invoke(context: Activity, args: Args): Completable
        = Completable.fromCallable {
        ImagePicker.Builder(context)
                .mode(ImagePicker.Mode.CAMERA_AND_GALLERY)
                .enableDebuggingMode(BuildConfig.DEBUG)
                .build()
    }
}
```

SERVICES

SERVICES

- Start Service using Action
- Service logic stored in Components
- Service could inherit multiple components and contain clue logic

WORKING WITH SERVICE



TESTS

TESTABILITY

Unit-testable layers
with no Android SDK dependency

- VIEWMODEL
- Action*
- Repository
- Component* (Interactor)

* Except implementations dependent on platform specific code

COMPONENT TEST

```
@Test fun observeEpisodesNonEmpty() {
    val episodesRepo = mock<EpisodesModelRepo> {
        on { observeAll() } doReturn Observable.just(listOf(mock()))
    }
    val component = EpisodesComponentImpl(episodesRepo)

    component.observeEpisodes()
            .test()
            .assertValueCount(1)

    verify(episodesRepo, times(1)).observeAll()
}
```

VIEW MODEL + LIFECYCLE TEST

```
@Test fun onDetach() {
    val naviComponent = NaviEmitter.createFragmentEmitter()
    val viewModel = ViewModelImpl(naviComponent)
    val test = viewModel.onTerminalEventObserver()
            .test()

    naviComponent.onDetach()

    test.assertValueCount(1)
}
```

NETWORK ACTION TEST

```
@Test fun invoke() {
    val repo = mock<EpisodesModelRepo> {
        on { save(any()) } doReturn Single.just(emptyMap<Long, EpisodesModel>())
    }
    val context = mock<Api> {
        on { feed() } doReturn Single.just(rssStub())
    }
    whenever(injector.episodesRepo()).doReturn(repo)

    GetEpisodesAction().invoke(context, argsOf())
            .test()
            .await()
            .assertComplete()

    verify(repo, times(1)).save(any())
}
```

WHATS NEXT?

- Ask your questions right now!
- Get example https://github.com/stepango/Archetype
- Watch this presentation once again
- Thanks

SOCIAL LINKS

- @stepango
- @nekdenis
- androiddev.apptractor.ru

SOCIAL LINKS

- @stepango
- @nekdenis
- androiddev.apptractor.ru

SOCIAL LINKS

- @stepango
- @nekdenis
- androiddev.apptractor.ru