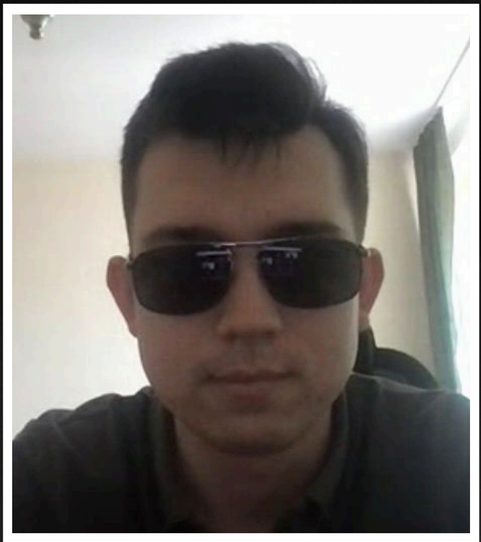
Android navigation at scale

Макушев Александр

Митропольский Александр



Александр Макушев

@batonec

Техлид платформенной команды
СберИнвестор

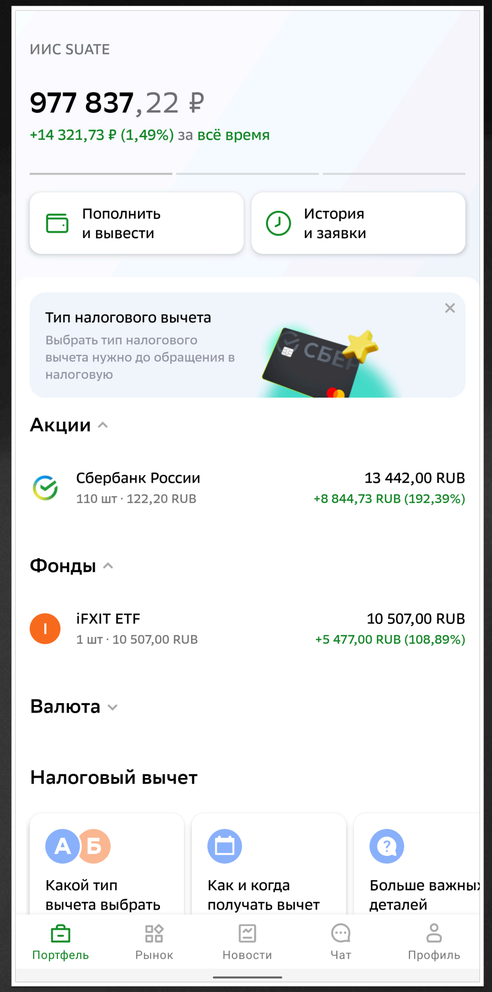
Александр Митропольский

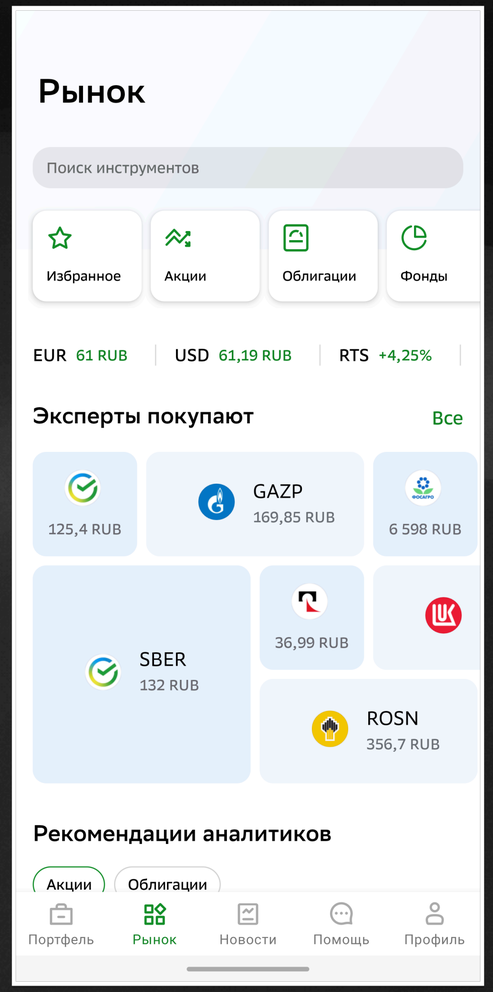
@a_mitr

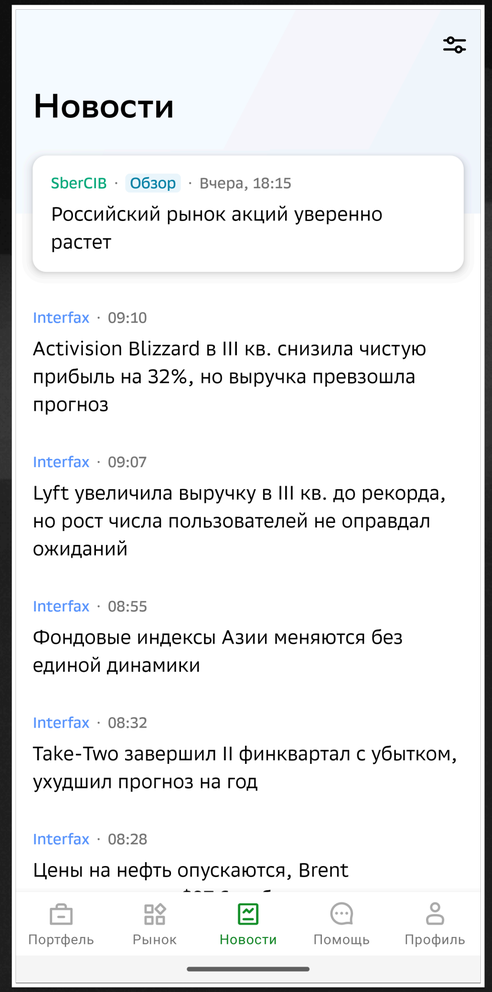
amitropolsky

Полгода придумывал название библиотеки

СберИнвестор







Дисклеймер

Проблемы Jetpack Navigation

Что хотим

Обзор Alcubierre

Deeplinks

Итог

Дисклеймер

Проблемы Jetpack Navigation

Что хотим

Обзор Alcubierre

Deeplinks

Итог

Дальше будем говорить про фрагменты.

Compose планируется как дальшейшее развитие библиотеки.

Дисклеймер

Проблемы Jetpack Navigation

Что хотим

Обзор Alcubierre

Deeplinks

Итог

Что не так

- Единственный способ навигации между модулями - диплинки.
  Не typesafe, только примитивные типы.
- Диплинки работают не оптимально.
- Нет conditional навигации.
- Игнорирование navigate после onStop.
- IllegalArgumentException: Navigation action/destionation cannot be found from the current destination.
- Нет возможности перехватить вызов навигации.
- Сложно управлять бэкстеком.
- God модуль для навигации.

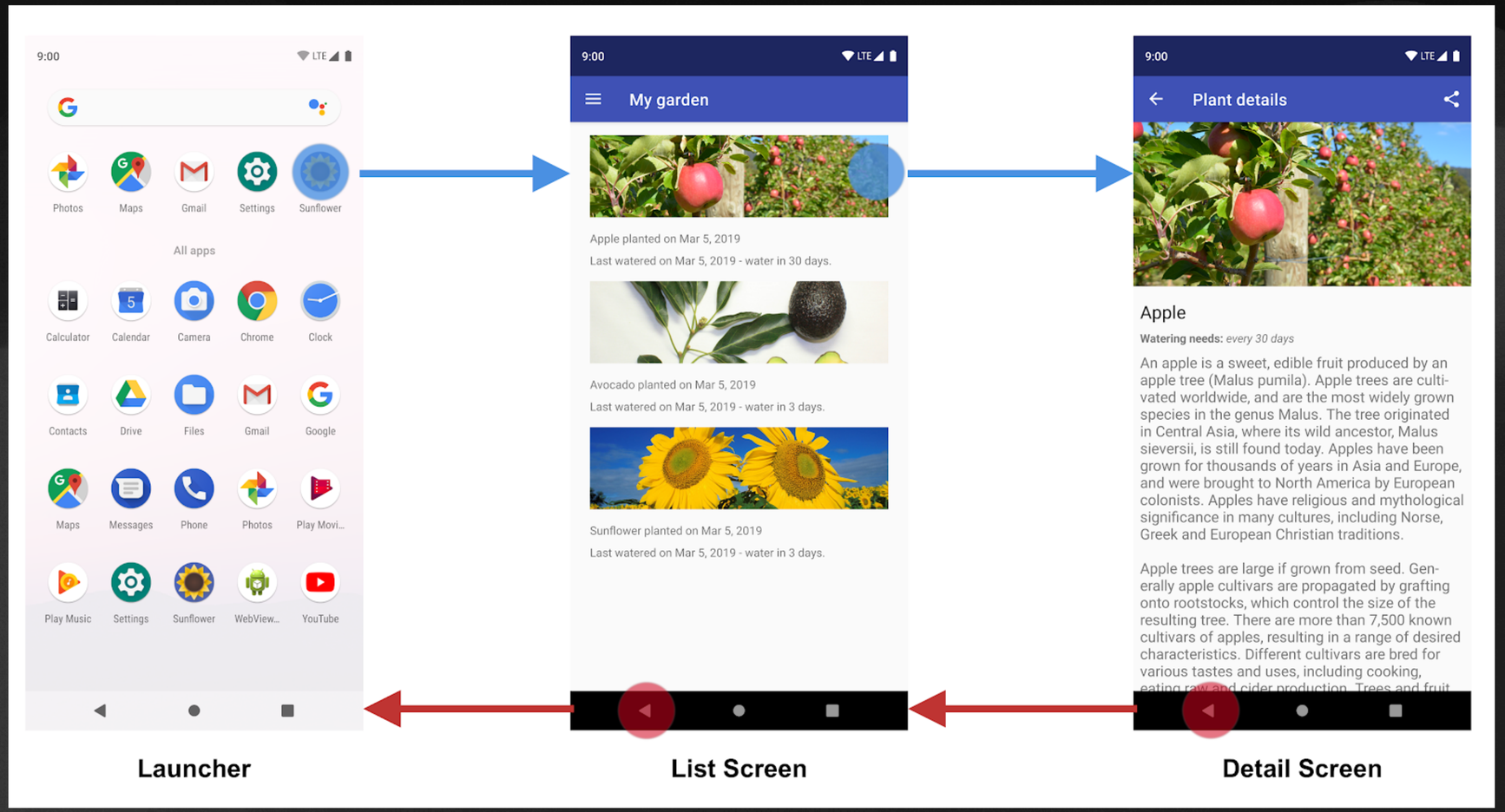
Единственный способ навигации между модулями - диплинки.
Не typesafe, только примитивные типы.

Диплинки работают не оптимально

```
<navigation
    android:id="@+id/second_graph"
    app:startDestination="@id/main_fragment">

    <fragment
        android:id="@+id/main_fragment"
        android:name="com.example.MainFragment">

        <deeplink app:uri="myapp://screen/{name}" />
    </fragment>

    <fragment
        android:id="@+id/secondary_fragment"
        android:name="com.example.SecondaryFragment">

        <deeplink app:uri="myapp://screen/{name}" />
     </fragment>
</navigation>
```

Нет conditional навигации

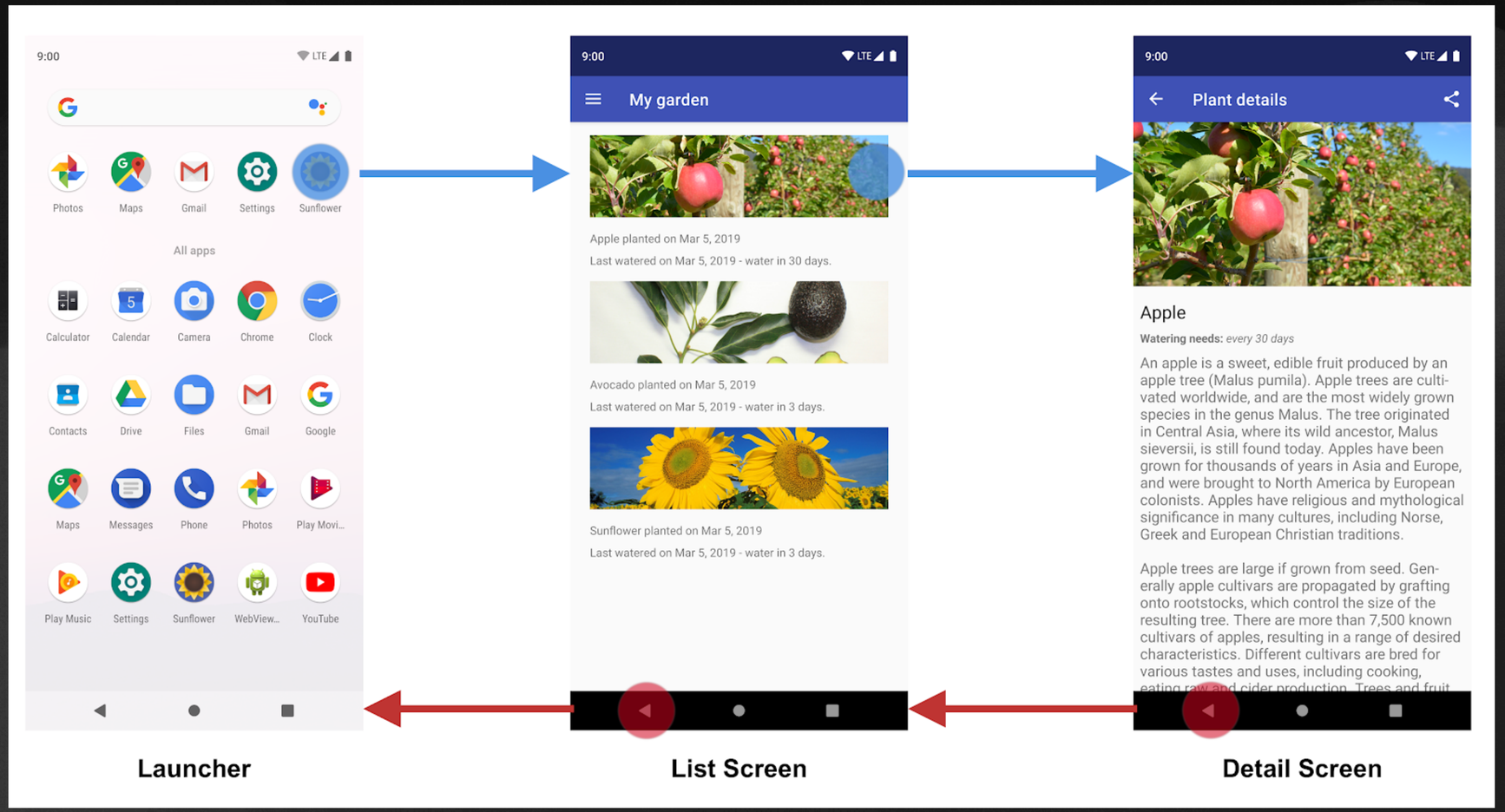
Fixed start destination



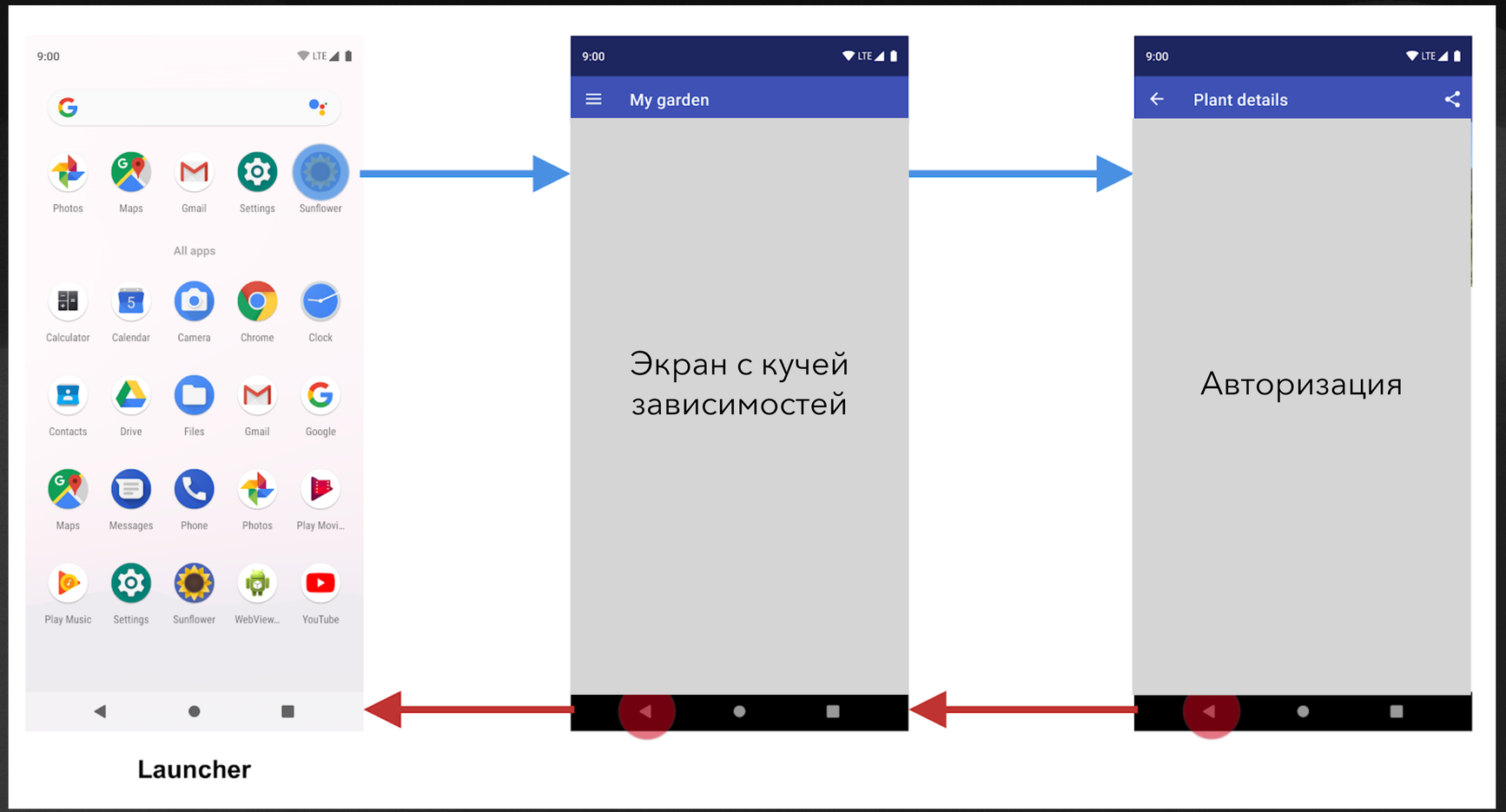
An app might have a one-time setup or series of login screens.
These conditional screens should not be considered start destinations because users see these screens only in certain cases.

developer.android.com

Fixed start destination



Fixed start destination



Игнорирование navigate после onStop

FragmentNavigator::navigate

```
if (mFragmentManager.isStateSaved()) {
    Log.i(TAG, "Ignoring navigate() call: FragmentManager has already"
            + " saved its state");
    return null;
}
```

IllegalArgumentException:
Navigation action/destionation cannot be found from the current destination.

```
<fragment
    android:id="@+id/first_fragment"
    android:name="com.example.FirstFragment">
  <action
    android:id="@+id/action_to_second"
    app:destination="@id/account_fragment" />
</fragment>

<fragment
    id="@+id/second_fragment"
    name="com.example.SecondFragment">
  <argument
    android:name="id"
    app:argType="string" />
</fragment>
```

```xml
<fragment
    android:id="@+id/first_fragment"
    android:name="com.example.FirstFragment">
  <action
    android:id="@+id/action_to_second"
    app:destination="@id/account_fragment" />
</fragment>

<fragment
    id="@+id/second_fragment"
    name="com.example.SecondFragment">
  <argument
    android:name="id"
    app:argType="string" />
</fragment>
```

```
button1.setOnClickListener { findNavController().navigate(actionToSecond("1")) }
```

Нет возможности перехватить вызов навигации

Нет возможности перехватить вызов навигации

myapp://screen/some_disabled_feature

Сложно управлять бэкстеком

Нельзя открыть цепочку экранов

God модуль для навигации

Дисклеймер

Проблемы Jetpack Navigation

Что хотим

Обзор Alcubierre

Deeplinks

Итог

Что хотим

- Избавиться от костылей
- Type safe навигация между модулями
- Возможность навигации в фоне
- Возможность перехватывать события навигации
- Контроль над бекстэком
- Диплинки
- Желательно не в одном модуле
- Максимально использовать готовые api: fragment result, (save/restore)Backstack

Дисклеймер

Проблемы Jetpack Navigation

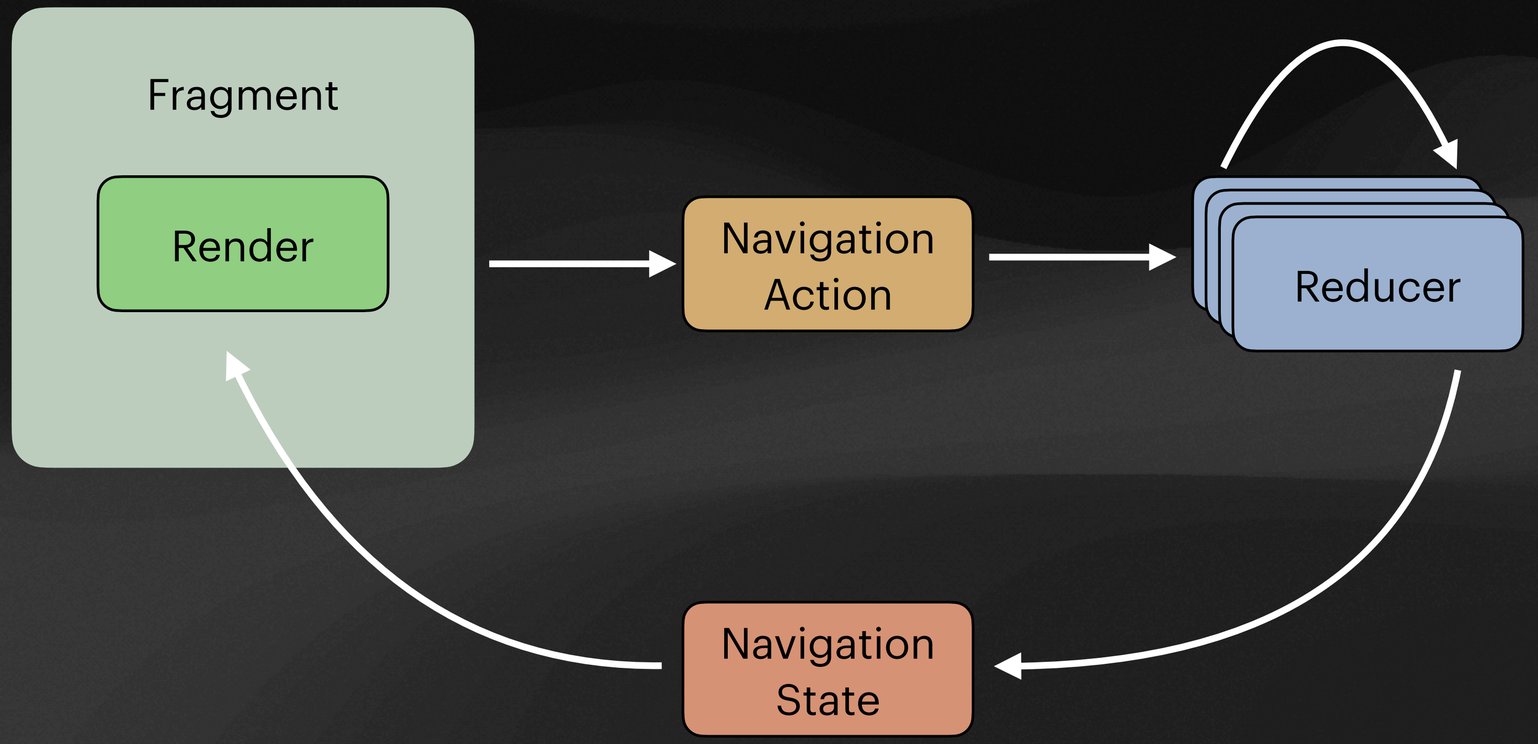
Что хотим

Обзор Alcubierre

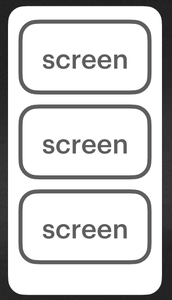
Deeplinks

Итог

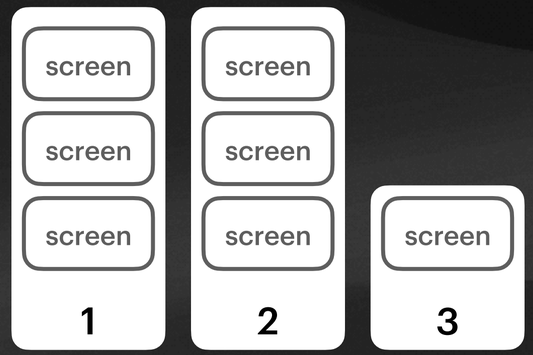
https://github.com/terrakok/Modo


Fragment
Render
Navigation Action
Reducer
Navigation State

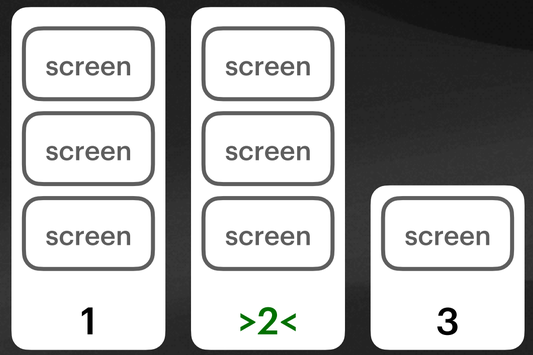
NavigationState



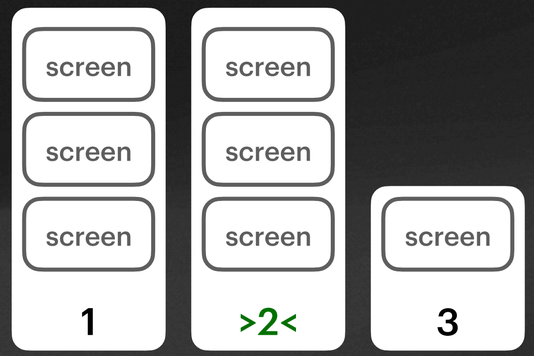
NavigationState



NavigationState

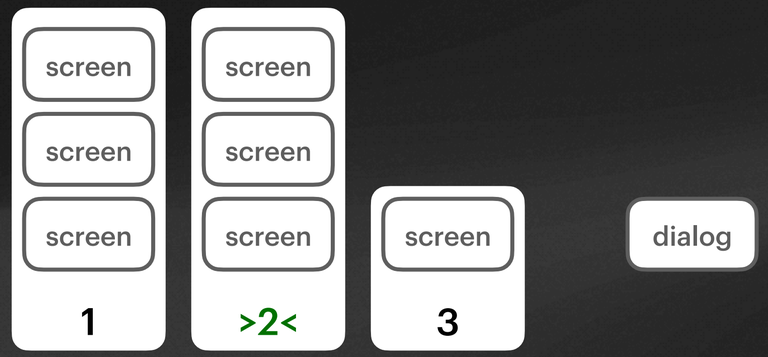


NavigationState



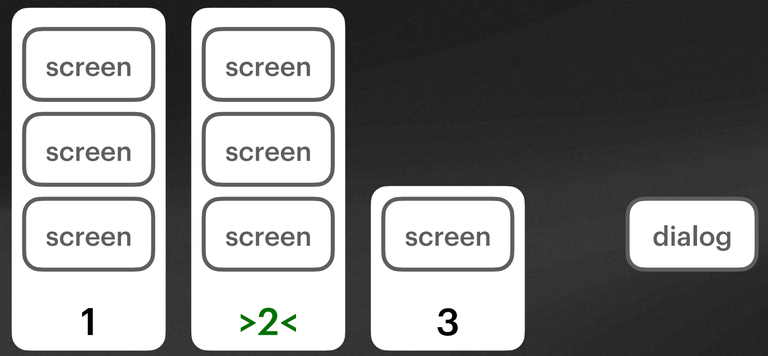

```
class RootNavigationState(
    val stacks: Map<Int, List<Screen>>,
    val currentStackId: Int
)
```

NavigationState




```
class RootNavigationState(
    val dialog: Dialog?,
    val stacks: Map<Int, List<Screen>>,
    val currentStackId: Int
)
```

NavigationState




```
class RootNavigationState(
    val dialog: Dialog?,
    val stacks: Map<Int, List<Screen>>,
    val currentStackId: Int
) : Parcelable
```

Screen, Dialog

```
interface Screen : Parcelable {
    val screenId: String
}
```

Screen, Dialog

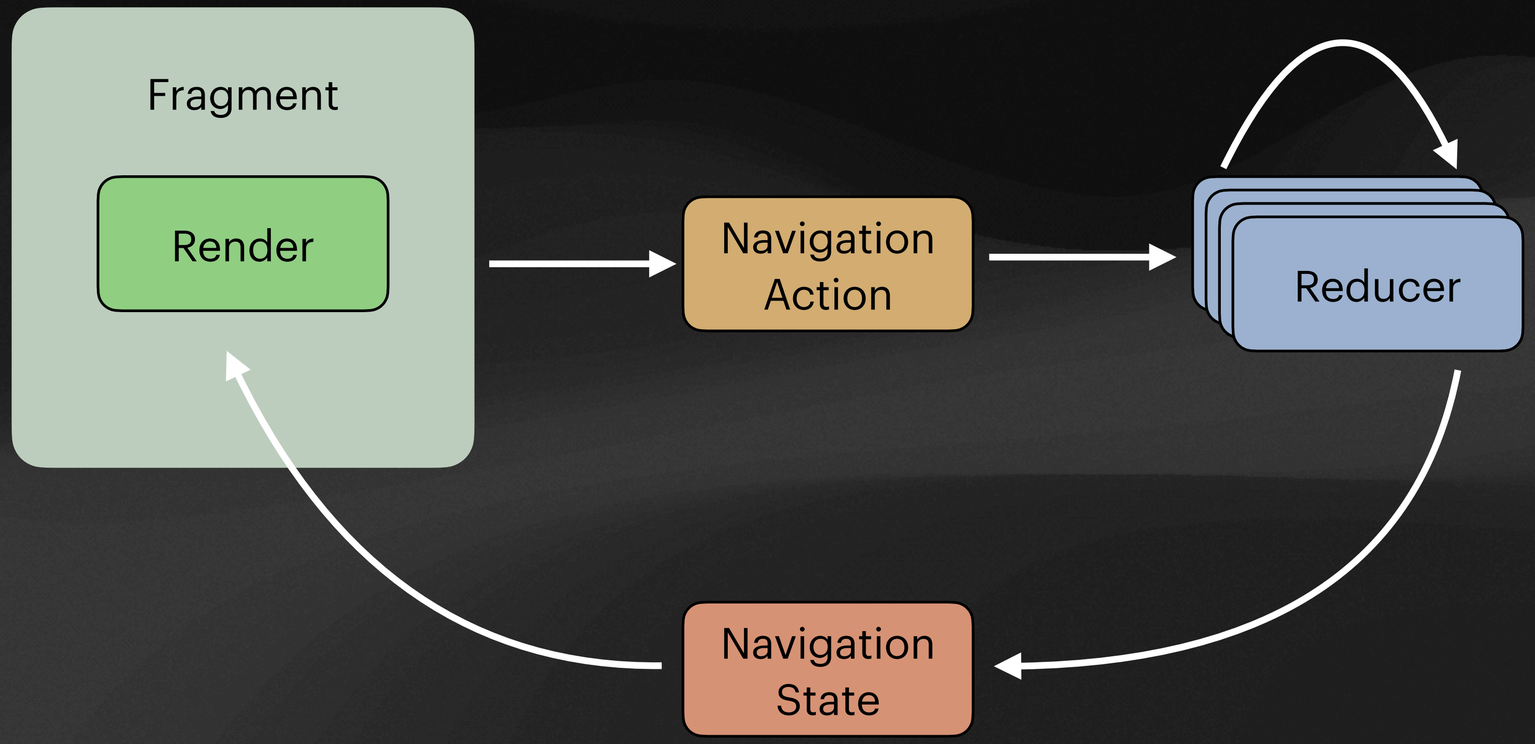
```
abstract class FragmentScreen(
    val fragmentName: String,
    val replace: Boolean = true
) : Screen {

    constructor(
        fragmentClass: KClass<out Fragment>,
        replace: Boolean = true
    ) : this(fragmentClass.java.name, replace)

    override val screenId by lazy(NONE) { fragmentName + hashCode() }

    override fun toString() = "[$screenId]"
}
```

Screen, Dialog

```
@Parcelize
data class PortfolioScreen(
    val someId: Long,
    val someArgument: String,
    val moreArgument: Parcelable
) : FragmentScreen(PortfolioFragment::class)
// или FragmentScreen(“some.package.PortfolioFragment”)
```

Screen, Dialog

```
interface FragmentCreator {

    fun create(): Fragment
}
```


Fragment
Render
Navigation Action
Reducer
Navigation State

Render

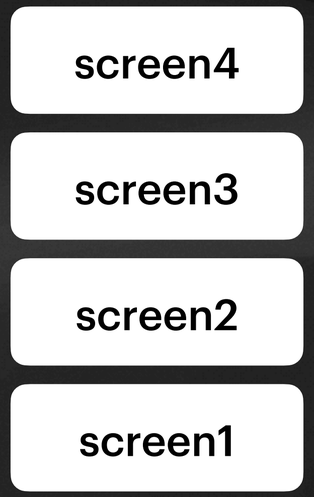
```
var currentState: State

fun render(newState: State) {
    val diff = diff(currentState, newState)
    doRender(diff)
    currentState = newState
}
```

Render

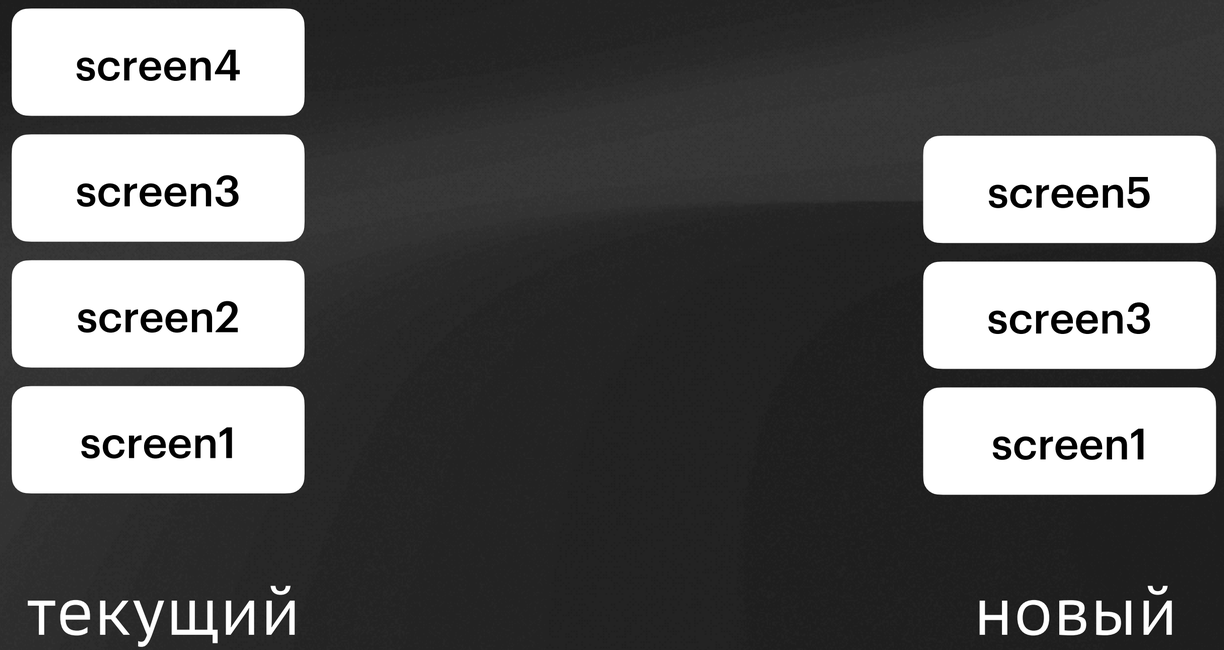
StackRender - управляет одним стеком

RootRender - управляет сменой стеков и StackRender’ами

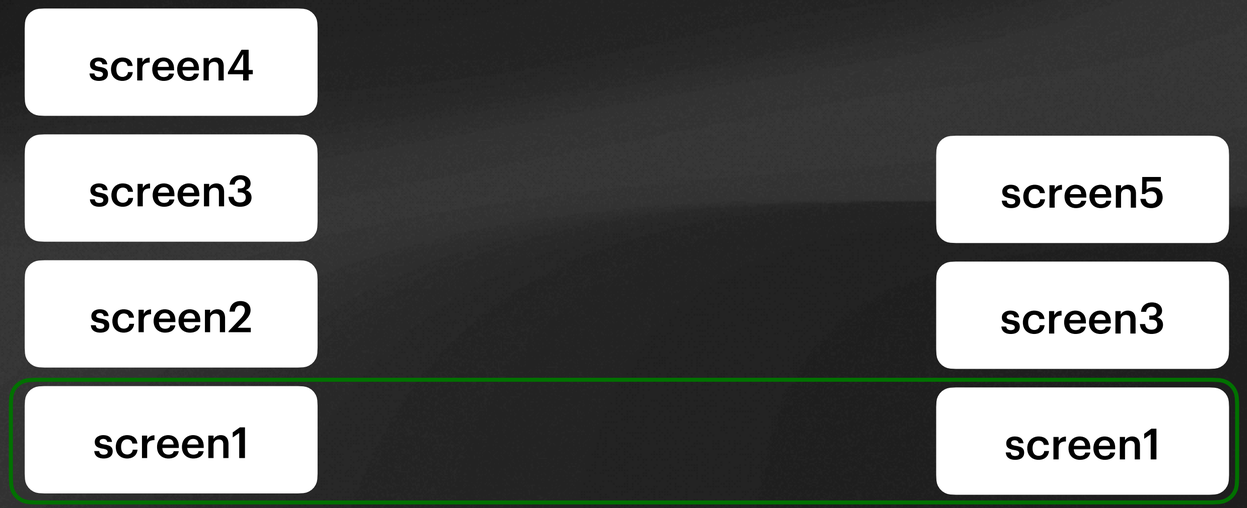
StackRender



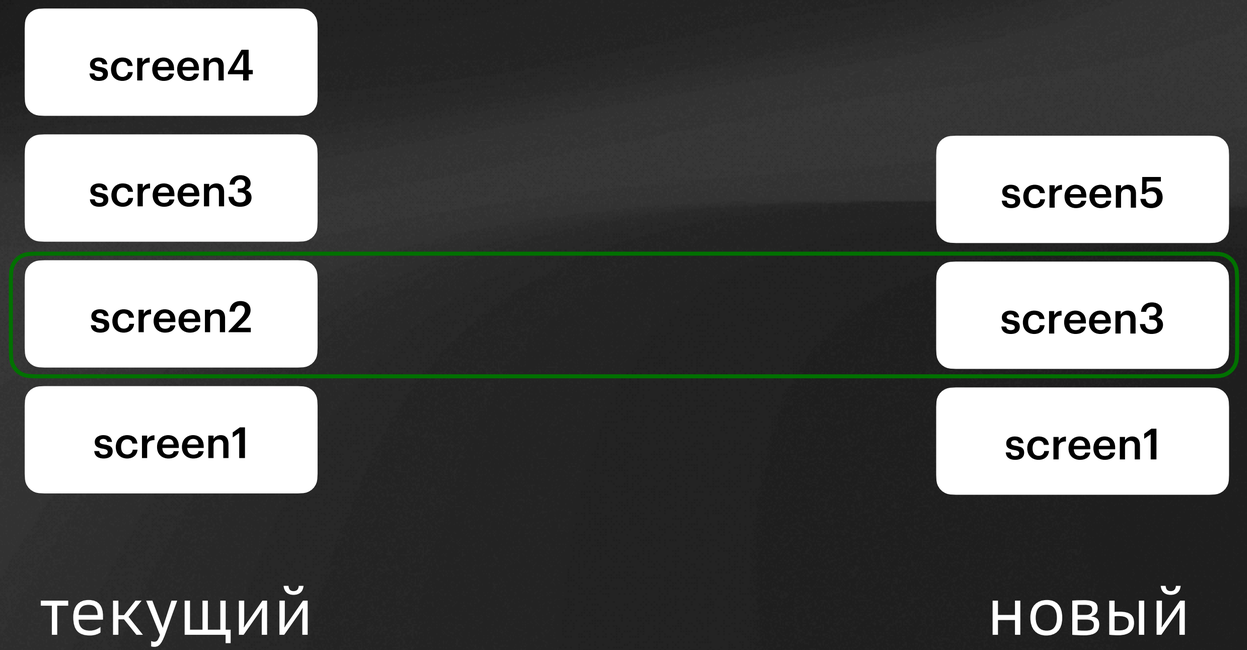
StackRender



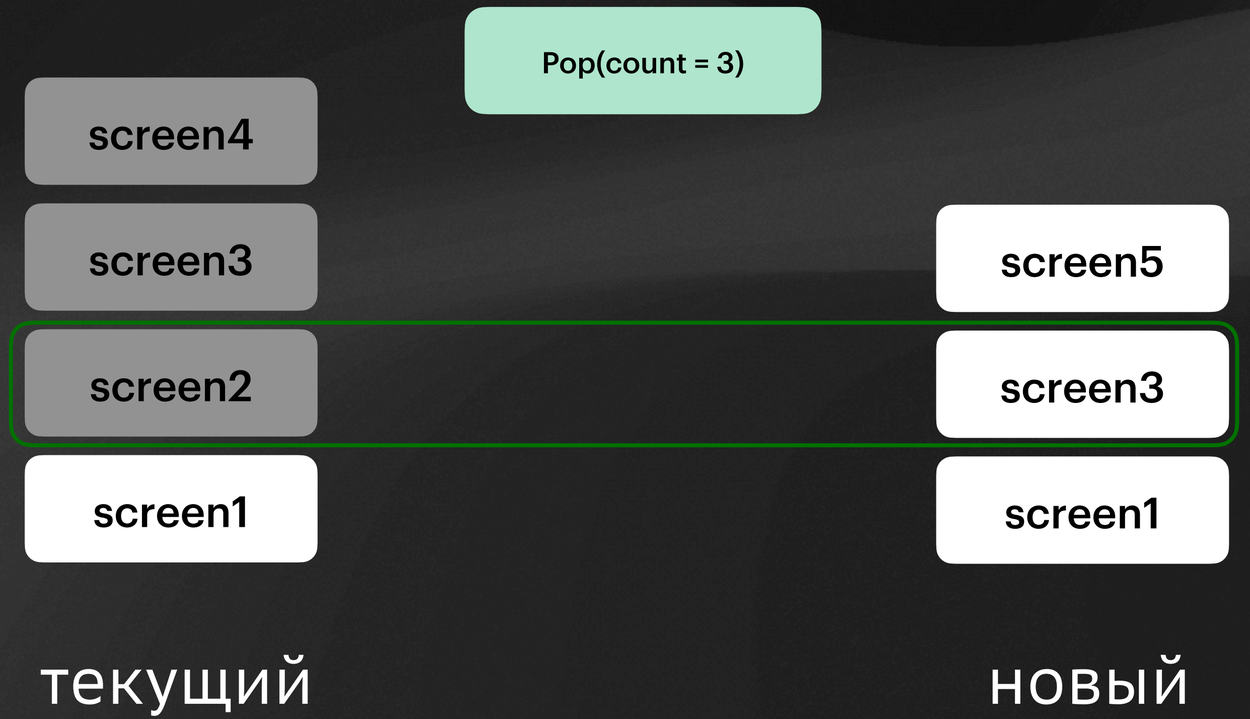
StackRender



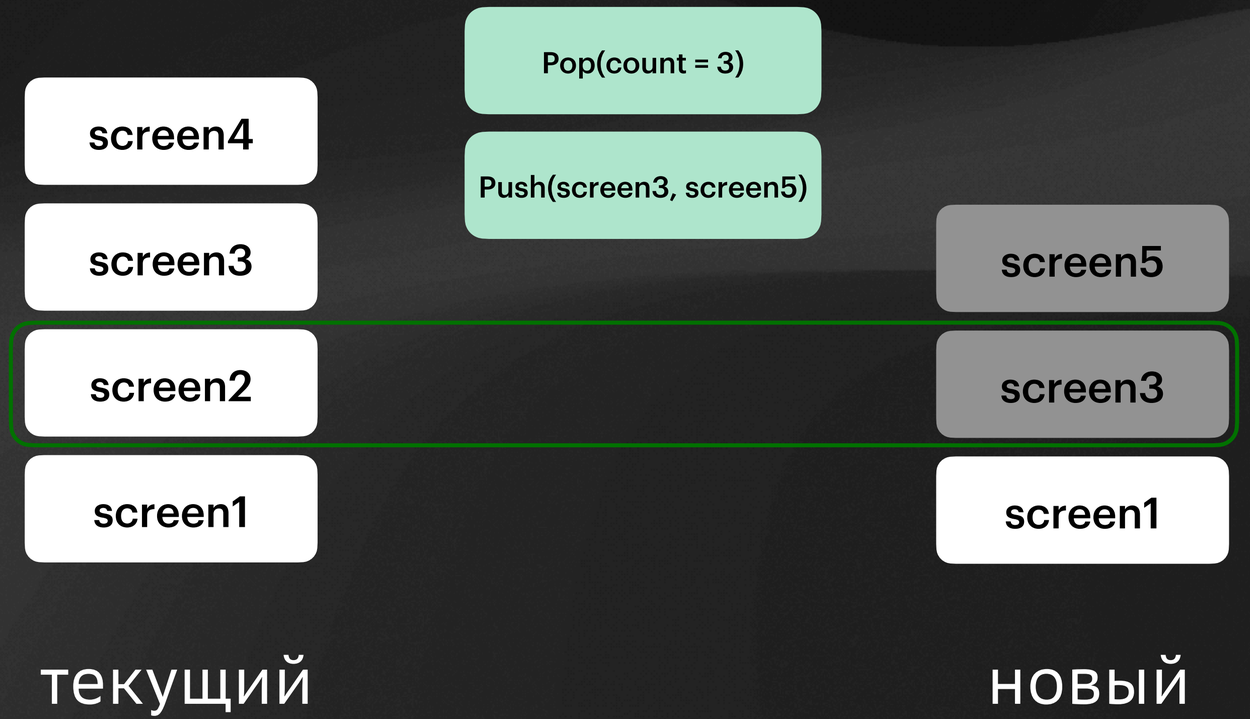
StackRender



StackRender



StackRender



StackRender
Push

```
fun push(screens: List<FragmentScreen>) {
    screens.forEach { screen →
        fragmentManager.commit {
            setReorderingAllowed(true)
            val fragment = createFragment(screen)
            transactionModifier.modify(this, screen, fragment)
            if (screen.replace) {
                replace(containerId, fragment, screen.screenId)
            } else {
                add(containerId, fragment, screen.screenId)
            }
            addToBackStack(screen.screenId)
        }
    }
}
```

StackRender
Push

```
fun push(screens: List<FragmentScreen>) {
    screens.forEach { screen →
        fragmentManager.commit {
            setReorderingAllowed(true)
            val fragment = createFragment(screen)
            transactionModifier.modify(this, screen, fragment)
            if (screen.replace) {
                replace(containerId, fragment, screen.screenId)
            } else {
                add(containerId, fragment, screen.screenId)
            }
            addToBackStack(screen.screenId)
        }
    }
}
```

StackRender
Push

```
fun push(screens: List<FragmentScreen>) {
    screens.forEach { screen →
        fragmentManager.commit {
            setReorderingAllowed(true)
            val fragment = createFragment(screen)
            transactionModifier.modify(this, screen, fragment)
            if (screen.replace) {
                replace(containerId, fragment, screen.screenId)
            } else {
                add(containerId, fragment, screen.screenId)
            }
            addToBackStack(screen.screenId)
        }
    }
}
```

StackRender
Push

```
fun push(screens: List<FragmentScreen>) {
    screens.forEach { screen →
        fragmentManager.commit {
            setReorderingAllowed(true)
            val fragment = createFragment(screen)
```

```
fun createFragment(screen: FragmentScreen): Fragment =
    if (screen is FragmentCreator) {
        screen.create()
    } else {
        fragmentManager.fragmentFactory
            .instantiate(classLoader, screen.fragmentName)
            .withScreenData(screen)
    }
```

StackRender
Push

```
fun push(screens: List<FragmentScreen>) {
    screens.forEach { screen →
        fragmentManager.commit {
            setReorderingAllowed(true)
            val fragment = createFragment(screen)
```

```
fun createFragment(screen: FragmentScreen): Fragment =
    if (screen is FragmentCreator) {
        screen.create()
    } else {
        fragmentManager.fragmentFactory
            .instantiate(classLoader, screen.fragmentName)
            .withScreenData(screen)
    }
```

StackRender
Push

```
fun push(screens: List<FragmentScreen>) {
    screens.forEach { screen →
        fragmentManager.commit {
            setReorderingAllowed(true)
            val fragment = createFragment(screen)
            transactionModifier.modify(this, screen, fragment)
            if (screen.replace) {
                replace(containerId, fragment, screen.screenId)
            } else {
                add(containerId, fragment, screen.screenId)
            }
            addToBackStack(screen.screenId)
        }
    }
}
```

StackRender
Push

```
fun push(screens: List<FragmentScreen>) {
    screens.forEach { screen →
        fragmentManager.commit {
            setReorderingAllowed(true)
            val fragment = createFragment(screen)
            transactionModifier.modify(this, screen, fragment)
            if (screen.replace) {
                replace(containerId, fragment, screen.screenId)
            } else {
                add(containerId, fragment, screen.screenId)
            }
            addToBackStack(screen.screenId)
        }
    }
}
```

StackRender

Pop

```
fun pop(count: Int) {
    val entryIndex = fragmentManager.backStackEntryCount - count
    if (entryIndex !in 0 until fragmentManager.backStackEntryCount) return
    val entryName = fragmentManager.getBackStackEntryAt(entryIndex).name
    fragmentManager.popBackStack(entryName, POP_BACK_STACK_INCLUSIVE)
}
```

StackRender

Pop

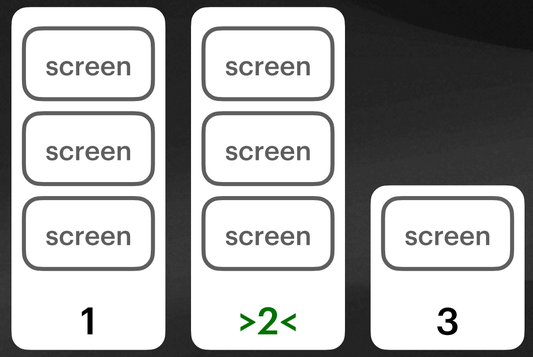
```
fun pop(count: Int) {
    val entryIndex = fragmentManager.backStackEntryCount - count
    if (entryIndex !in 0 until fragmentManager.backStackEntryCount) return
    val entryName = fragmentManager.getBackStackEntryAt(entryIndex).name
    fragmentManager.popBackStack(entryName, POP_BACK_STACK_INCLUSIVE)
}
```

StackRender

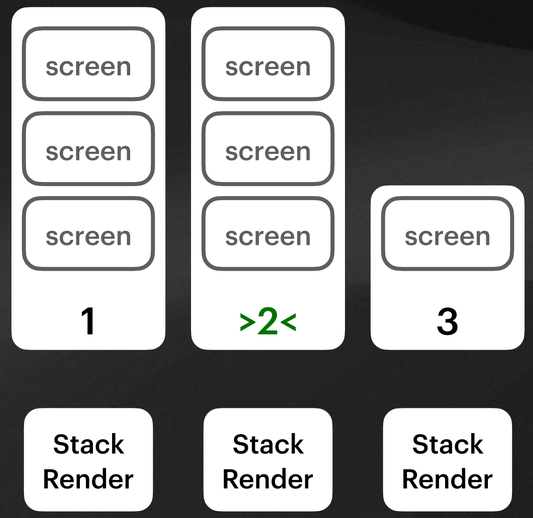
Pop

```
fun pop(count: Int) {
    val entryIndex = fragmentManager.backStackEntryCount - count
    if (entryIndex !in 0 until fragmentManager.backStackEntryCount) return
    val entryName = fragmentManager.getBackStackEntryAt(entryIndex).name
    fragmentManager.popBackStack(entryName, POP_BACK_STACK_INCLUSIVE)
}
```

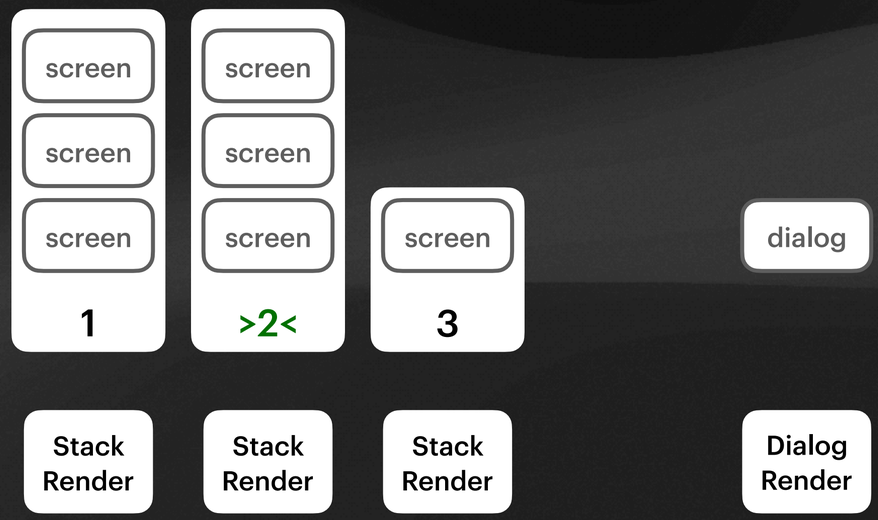
RootRender



RootRender

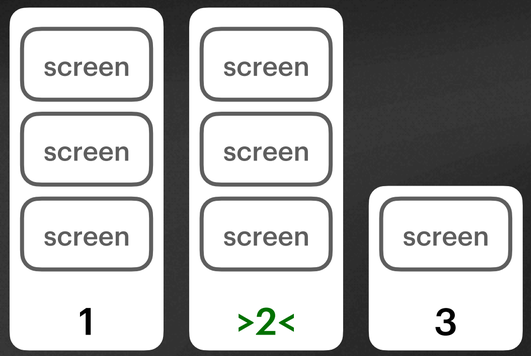

screen
screen
screen
1
screen
screen
screen
>2<
screen
3
Stack Render
Stack Render
Stack Render

RootRender

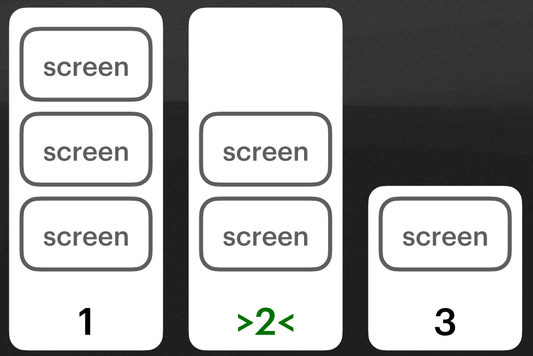

screen
screen
screen
1
screen
screen
screen
>2<
screen
3
dialog
Stack Render
Stack Render
Stack Render
Dialog Render

RootRender

Стек не поменялся



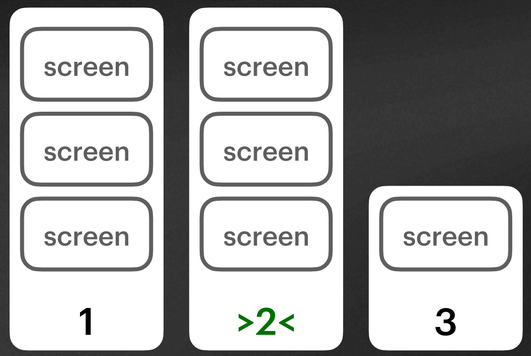
текущий



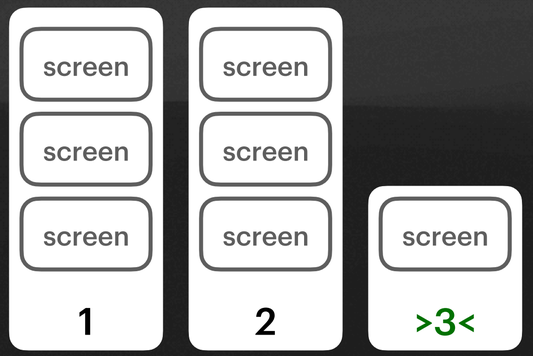
новый

RootRender

Стек поменялся



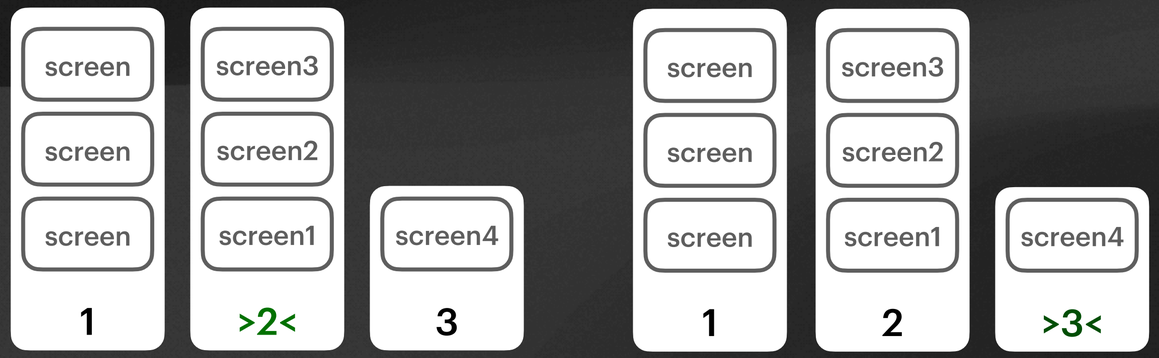
текущий



новый

RootRender

Стек поменялся



текущий

новый

FragmentManager
BackStack

RootRender

Стек поменялся

FragmentManager
BackStack

RootRender

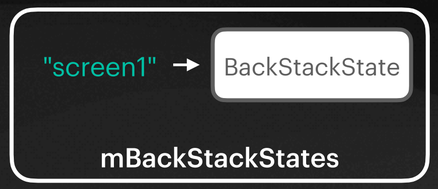
Стек поменялся

FragmentManager
BackStack

```
saveBackStack(
  "screen1"
)
```

RootRender

Стек поменялся




```
saveBackStack(
  "screen1"
)
```

FragmentManager
BackStack

FragmentManager
BackStack

RootRender
Стек поменялся

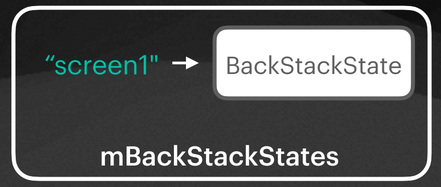

```
saveBackStack(
  "screen1"
)
```

FragmentManager
BackStack

FragmentManager
BackStack

RootRender

Стек поменялся

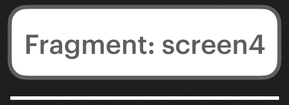


FragmentManager
BackStack

```
saveBackStack(
  "screen1"
)
```

FragmentManager
BackStack

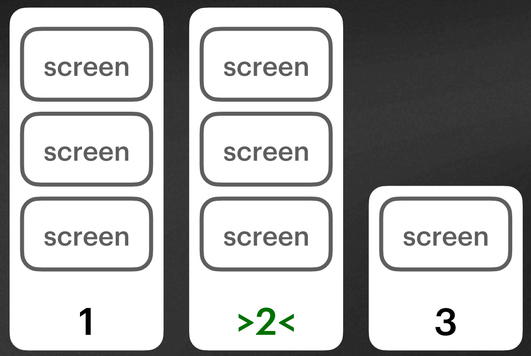
```
restoreBackStack(
  "screen4"
)
```



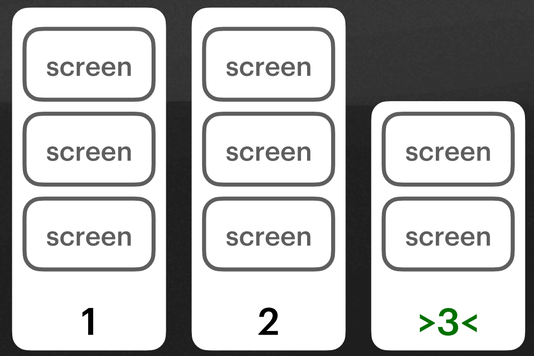
FragmentManager
BackStack

RootRender

Стек поменялся

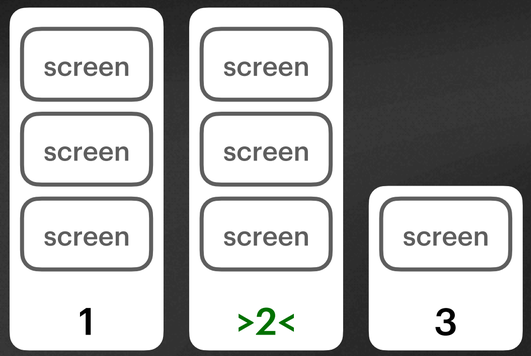


текущий

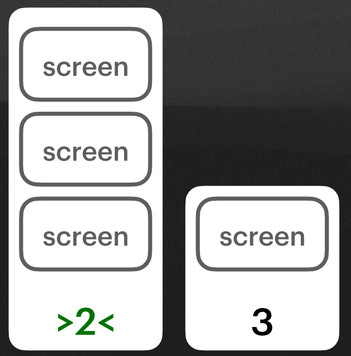


новый

RootRender

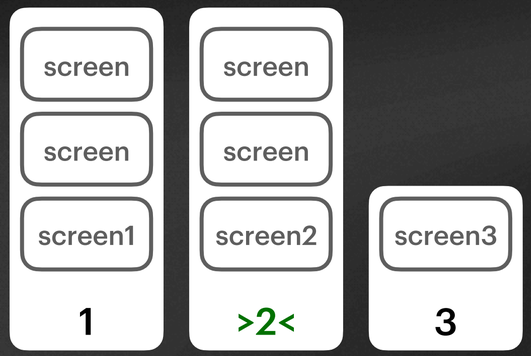


текущий

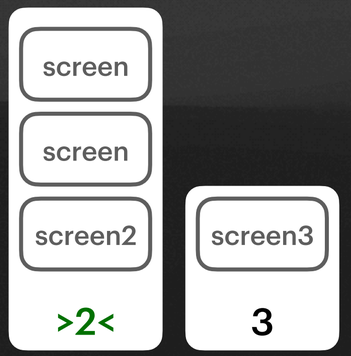


новый

RootRender



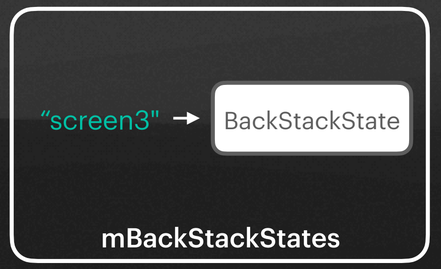
текущий



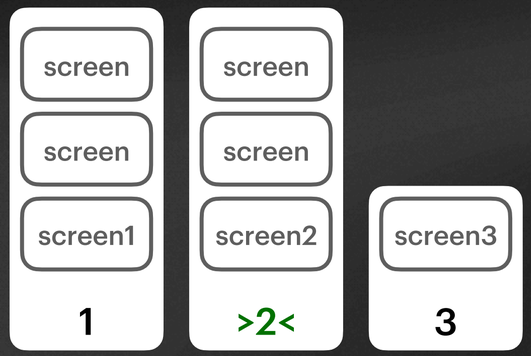
новый

RootRender

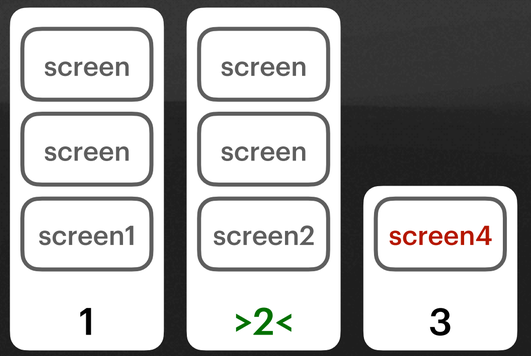

```
clearBackStack(
  "screen1"
)
```



RootRender

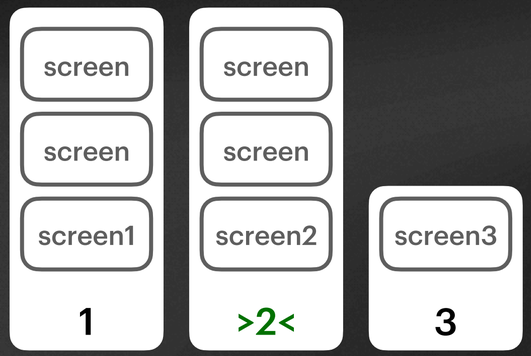


текущий

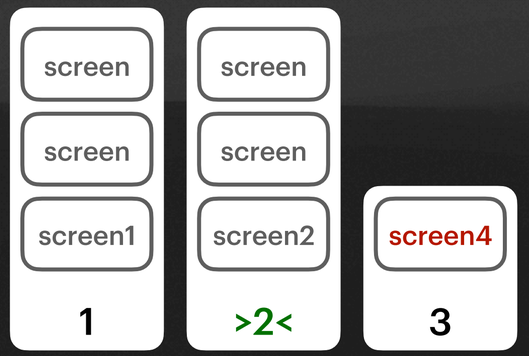


новый

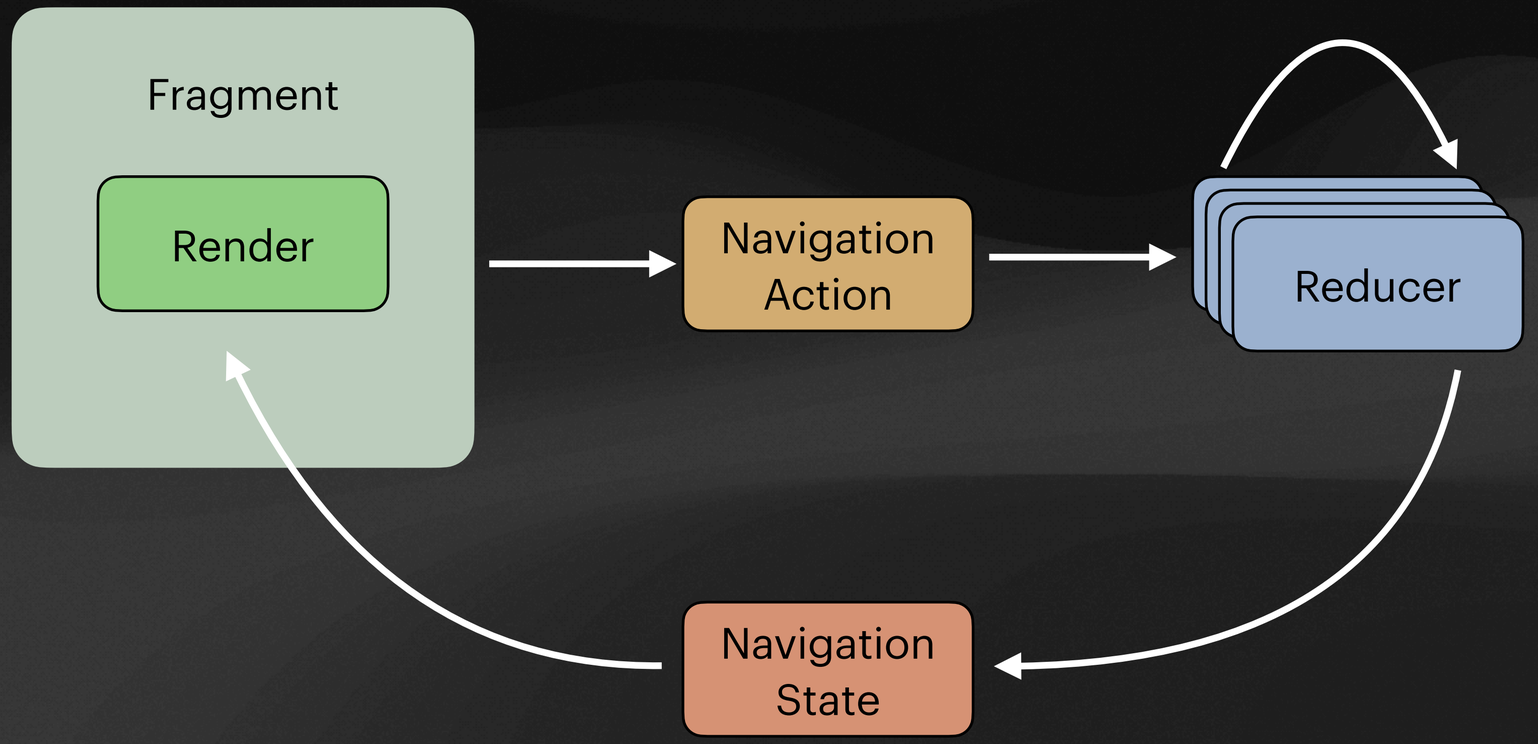
RootRender



текущий



новый

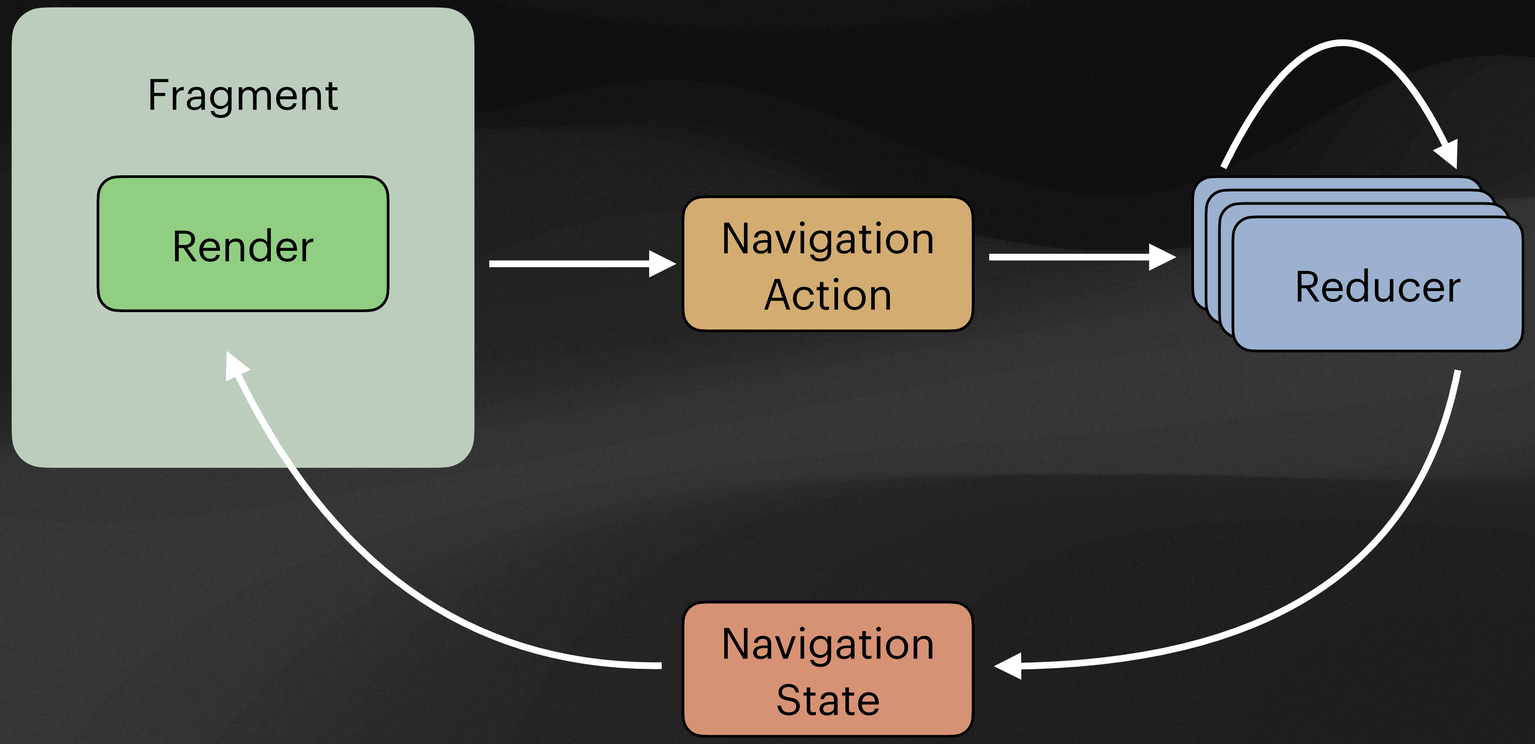

Fragment
Render
Navigation Action
Reducer
Navigation State

NavigationAction

```
class Forward(val screens: List<Screen>) : NavigationAction
class Replace(val screens: List<Screen>) : NavigationAction
class BackTo(val screenId: String) : NavigationAction
object Back : NavigationAction

class NewStack(val stackId: Int, val screens: List<Screen>) : NavigationAction
class SelectStack(val stackId: Int) : NavigationAction
```

NavigationAction

```
class Forward(val screens: List<Screen>) : NavigationAction
class Replace(val screens: List<Screen>) : NavigationAction
class BackTo(val screenId: String) : NavigationAction
object Back : NavigationAction

class NewStack(val stackId: Int, val screens: List<Screen>) : NavigationAction
class SelectStack(val stackId: Int) : NavigationAction

object Shuffle : NavigationAction
```

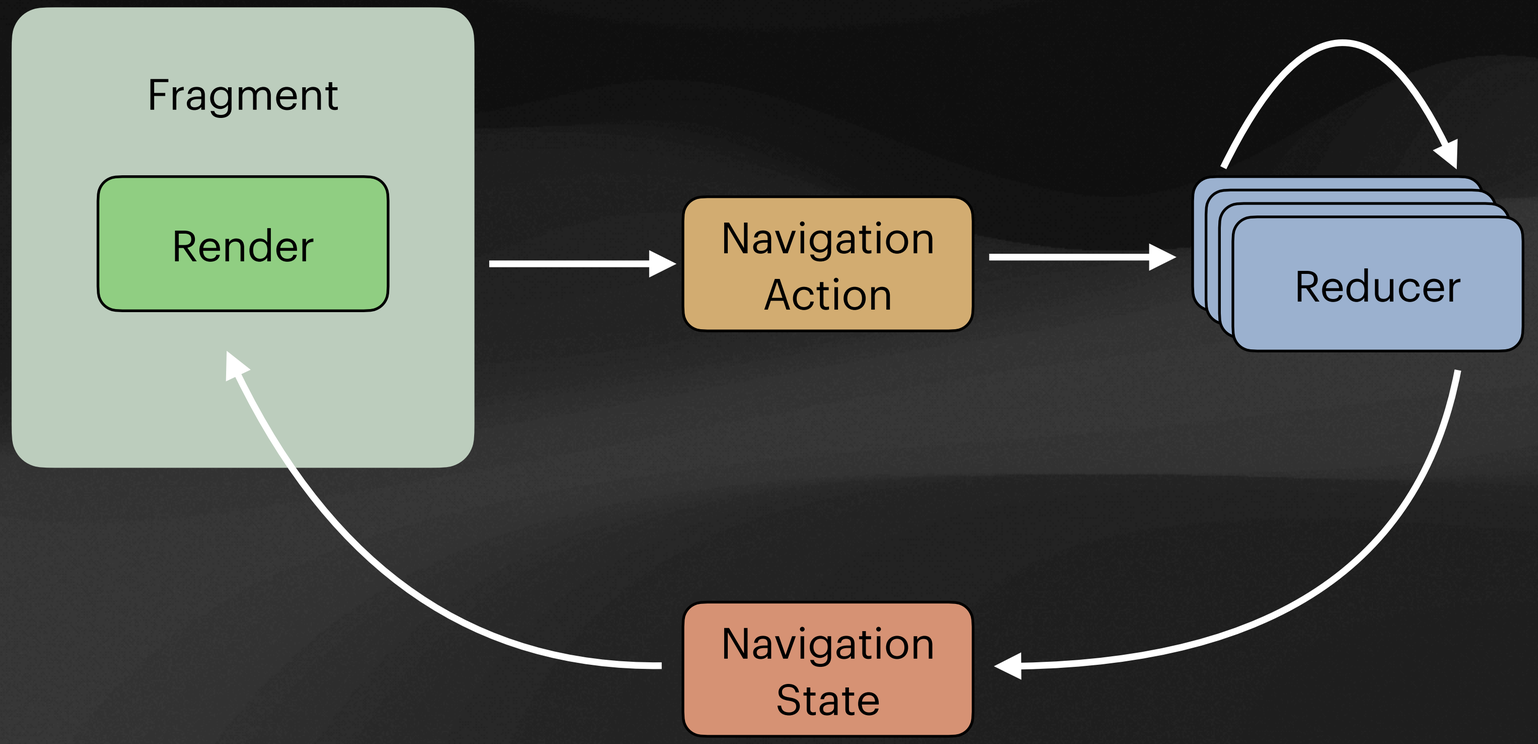

Fragment
Render
Navigation Action
Reducer
Navigation State

Reducer

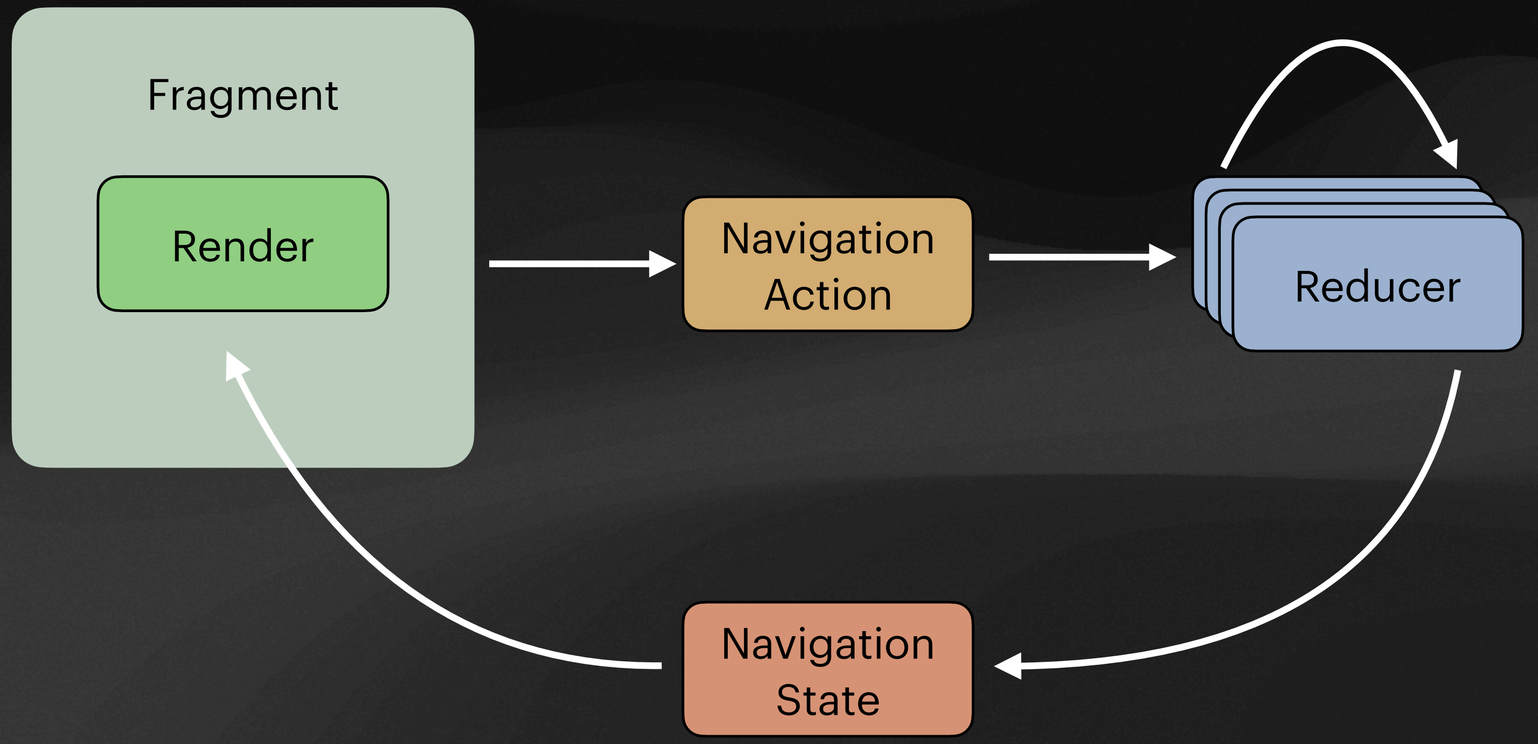
```
fun reduce(currentState: State, action: NavigationAction): State
```

Reducer

```
class StackChangedListenerReducer(
    private val delegate: NavigationReducer<RootNavigationState>,
    private val onStackChanged: (from: Int, to: Int) → Unit
): NavigationReducer<RootNavigationState> {

    override fun reduce(
        state: RootNavigationState,
        action: NavigationAction
    ): RootNavigationState {
        val newState = delegate.reduce(state, action)
        if (state.currentStackId ≠ newState.currentStackId) {
            onStackChanged(state.currentStackId, newState.currentStackId)
        }
        return newState
    }
}
```


Fragment
Render
Navigation Action
Reducer
Navigation State

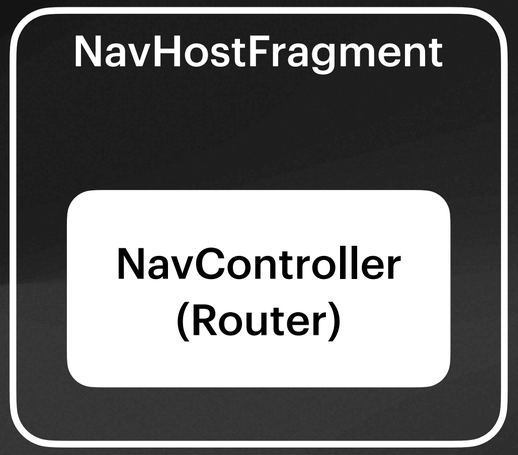
Где точка входа?



Где точка входа?

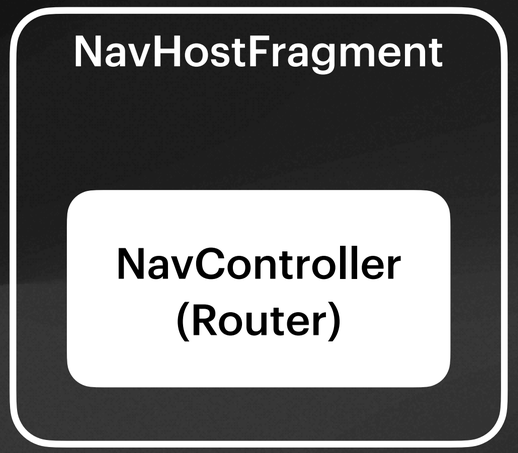
NavController
(Router)

Где точка входа?

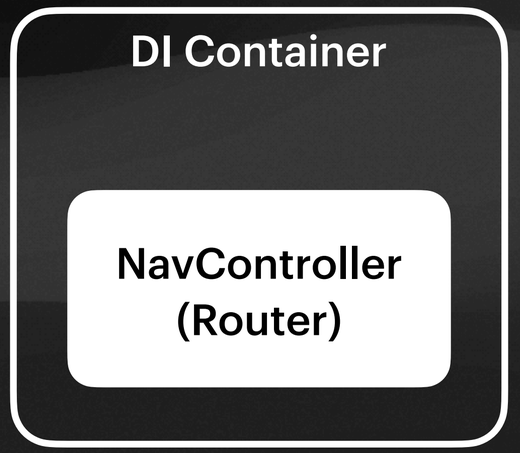



```
Fragment.findNavController()
View.findNavController()
```

Где точка входа?




```
Fragment.findNavController()
View.findNavController()
```

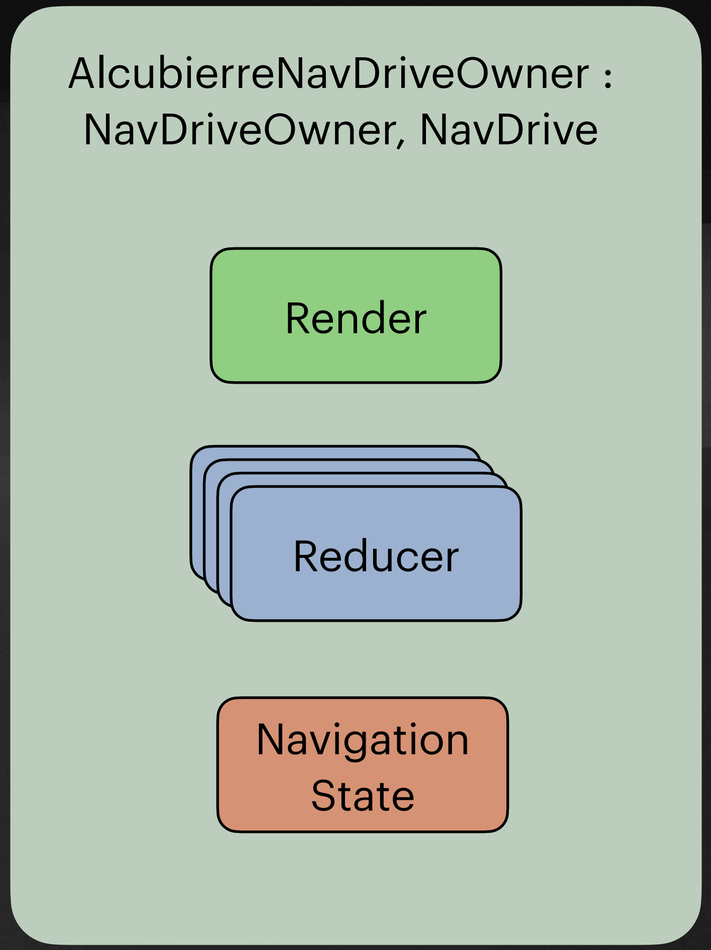



```
@Inject navController: NavController
```

Где точка входа?

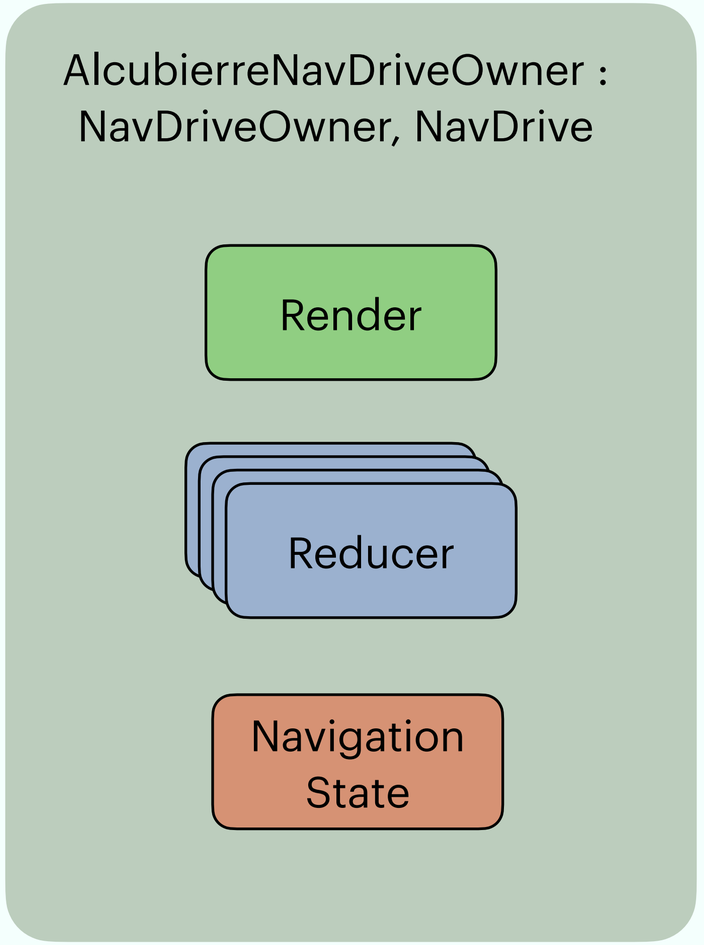
```
interface NavDrive {

    val state: RootNavigationState
    fun dispatch(action: NavigationAction)
}

interface NavDriveOwner : NavDrive {

    fun initialize(...)

    fun onResume()
    fun onPause()
    fun saveState(outState: Bundle)
}
```

Где точка входа?



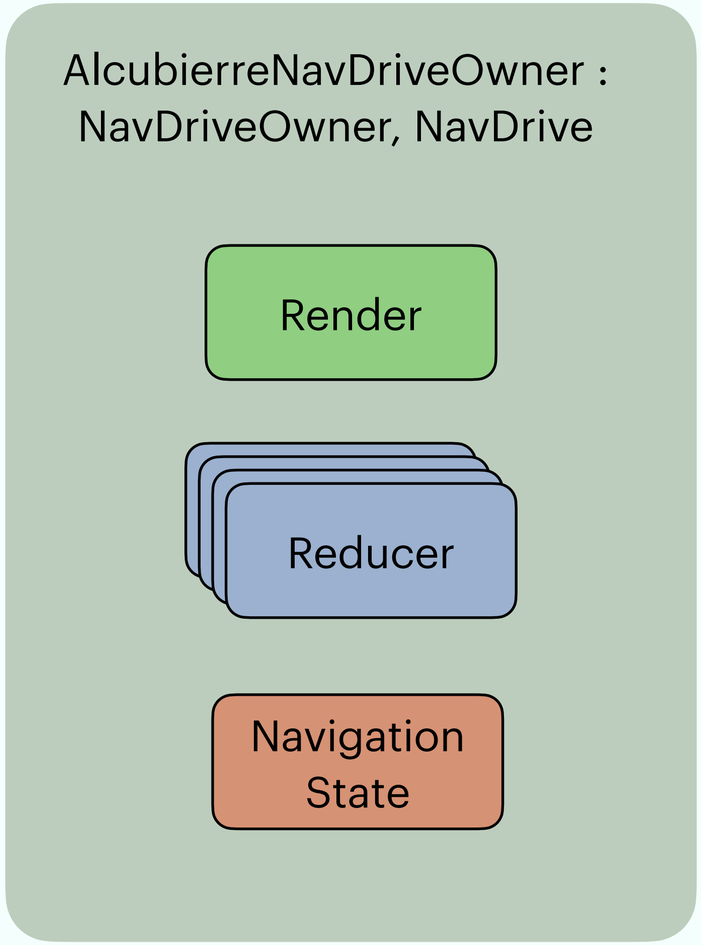
Где точка входа?

AlcubierreNavDriveFragment :
NavDriveOwner, NavDrive



Где точка входа?

AlcubierreNavDriveFragment :
NavDriveOwner, NavDrive




```
Fragment.findNavDrive()
View.findNavDrive()
```

Расширения

```
fun NavDrive.forward(vararg screens: Screen) = dispatch(Forward(screens.toList()))
fun NavDrive.replace(vararg screens: Screen) = dispatch(Replace(screens.toList()))
fun NavDrive.backTo(screenId: String) = dispatch(BackTo(screenId))
fun NavDrive.backTo(screen: Screen) = dispatch(BackTo(screen.id))
fun NavDrive.back() = dispatch(Back)
//and so on…
```

Почему не Modo?

```
// Alcubierre
class RootNavigationState(
    val dialog: Dialog?,
    val stacks: Map<Int, List<Screen>>,
    val currentStackId: Int
)
```

```
// Modo
class NavigationState(
    val chain: List<Screen>
)
```

Почему не Modo?

```kotlin
// Alcubierre
class RootNavigationState(
    val dialog: Dialog?,
    val stacks: Map<Int, List<Screen>>,
    val currentStackId: Int
)
```

```kotlin
// Modo
class NavigationState(
    val chain: List<Screen>
)
```

Мультибекстек только в корне:

- Более простые reducer'ы
- Явный доступ к мультибекстеку
- Табы не связаны с библиотекой
- Работает Fragment Result API

Почему не Modo?

```
// Alcubierre
fm.saveBackStack("screen")
fm.restoreBackStack("screen")
```

```
// Modo
fm.commit { show(stackContainerFragment) }
fm.commit { hide(stackContainerFragment) }
```

Новое API FragmentManager:

- Работает Fragment Result API
- Полноценный ЖЦ фрагмента

Почему не Modo?

- Диалоги
- Многомодульность (создание фрагментов через FragmentFactory)
- Аналог SafeArgs (объект Screen в аргументах фрагмента)
- Диплинки
- И другие небольшие доработки

Дисклеймер

Проблемы Jetpack Navigation

Что хотим

Обзор Alcubierre

Deeplinks

Итог

Схема процесса

Схема процесса

```
class FeatureScreen(val id: Int): FragmentScreen(FeatureFragment::class)
```

Схема процесса

```
@Deeplink("scheme://feature/{id}")
class FeatureScreen(val id: Int): FragmentScreen(FeatureFragment::class)
```

Схема процесса

Многомодульность

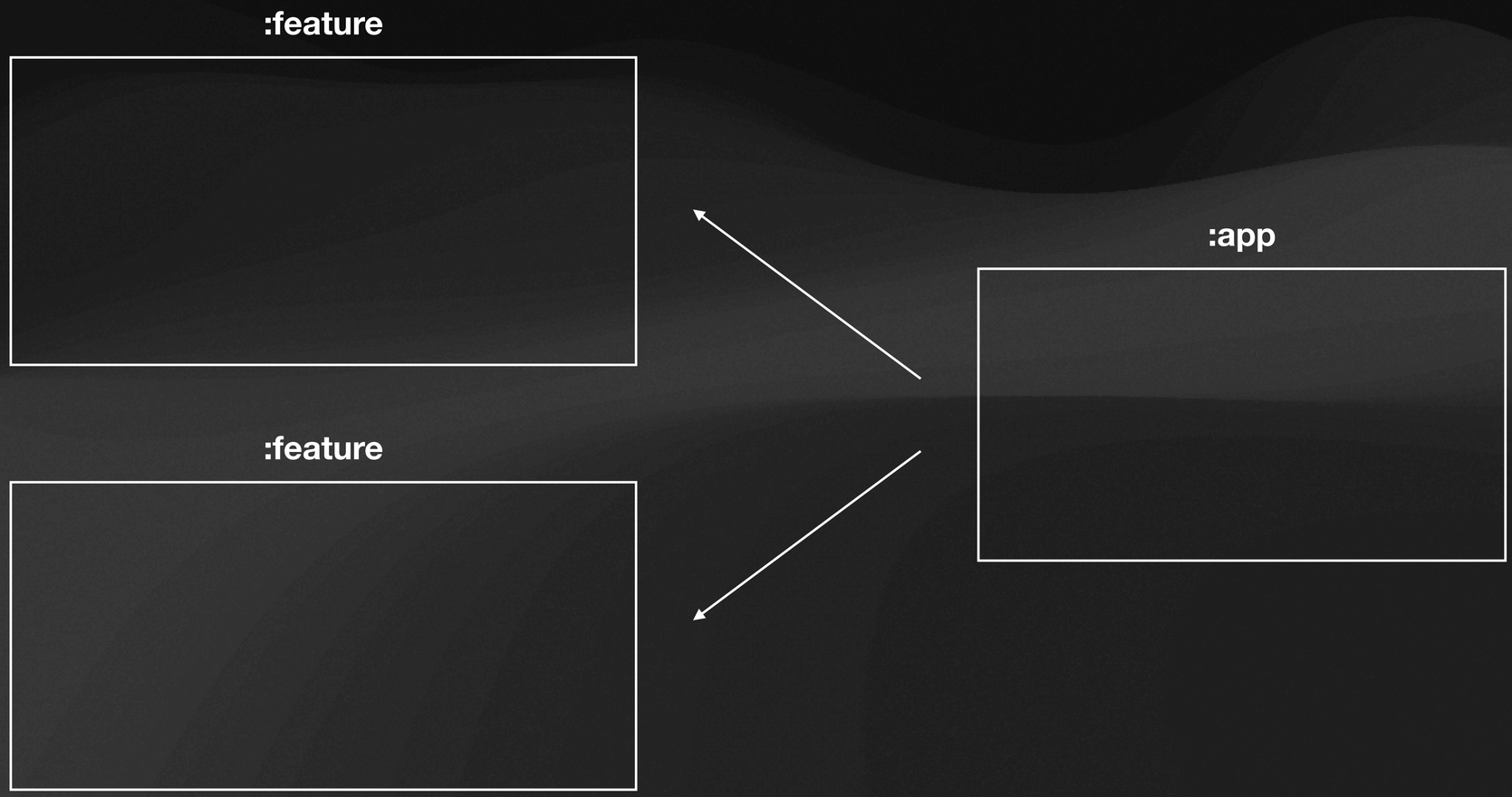


Схема процесса
Многомодульность

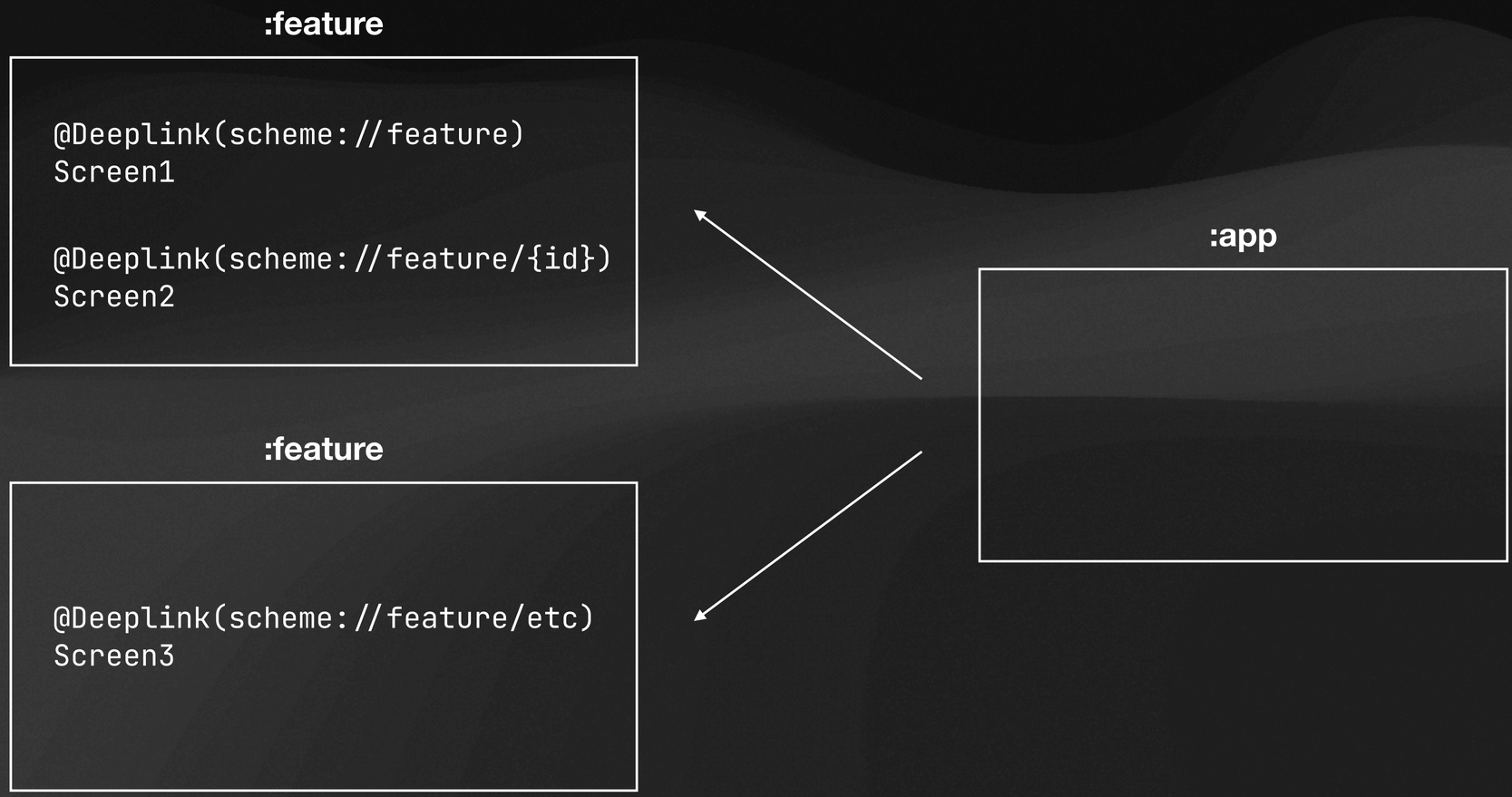


Схема процесса
Многомодульность

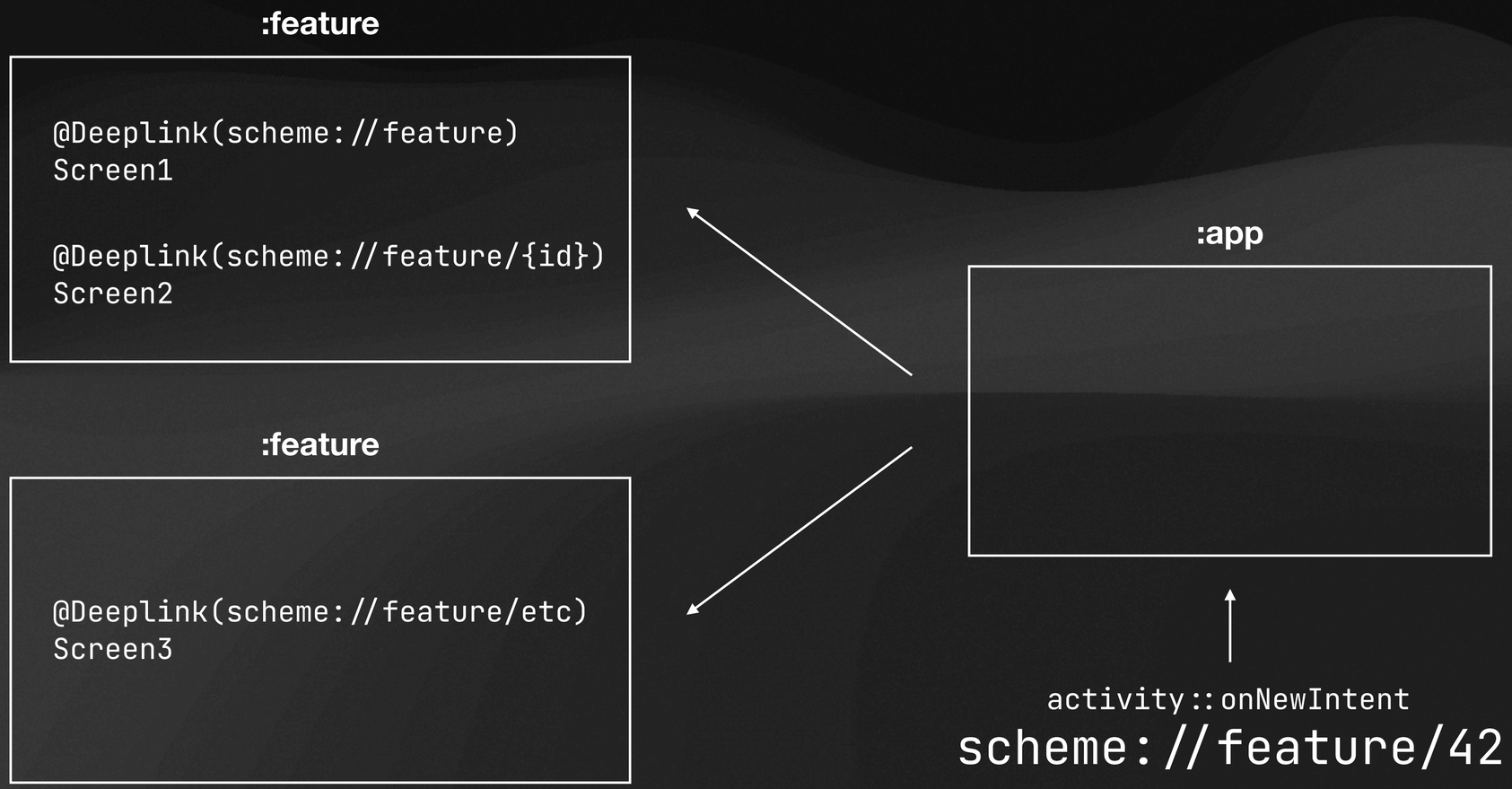


Схема процесса
Многомодульность

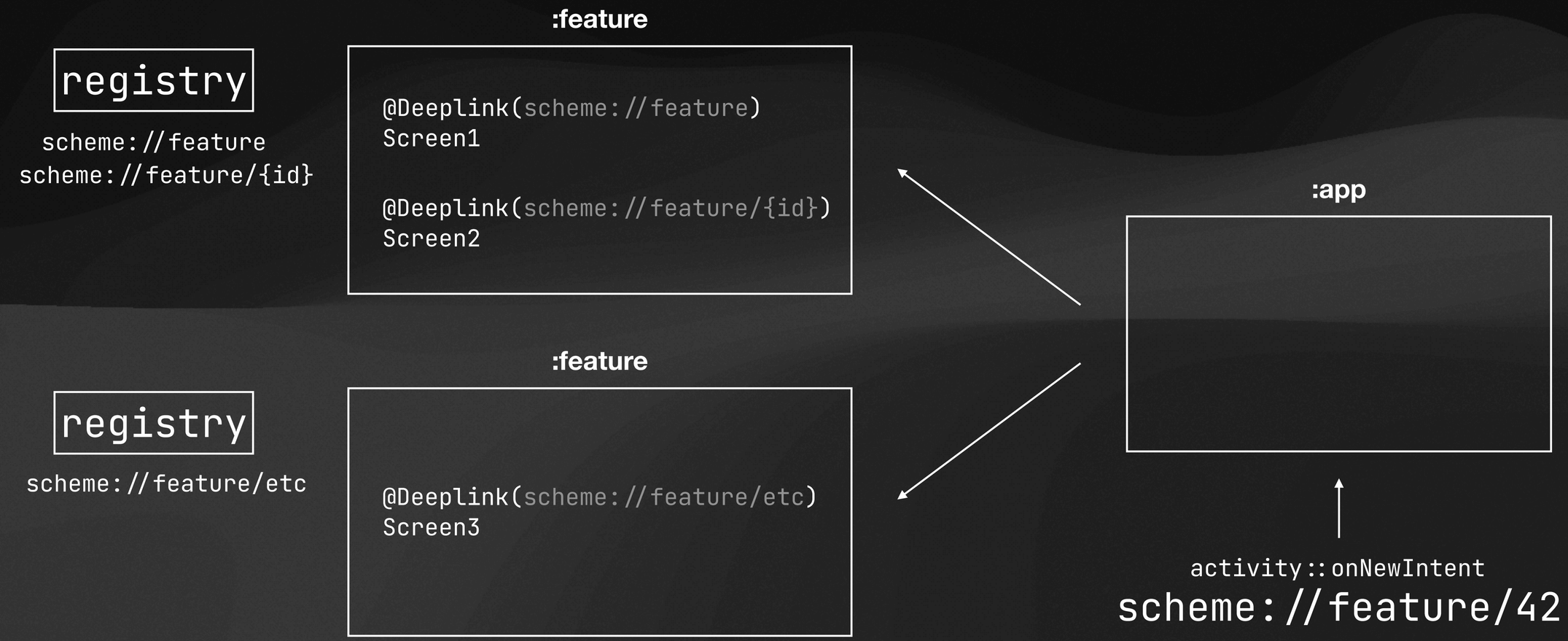


Схема процесса
Многомодульность

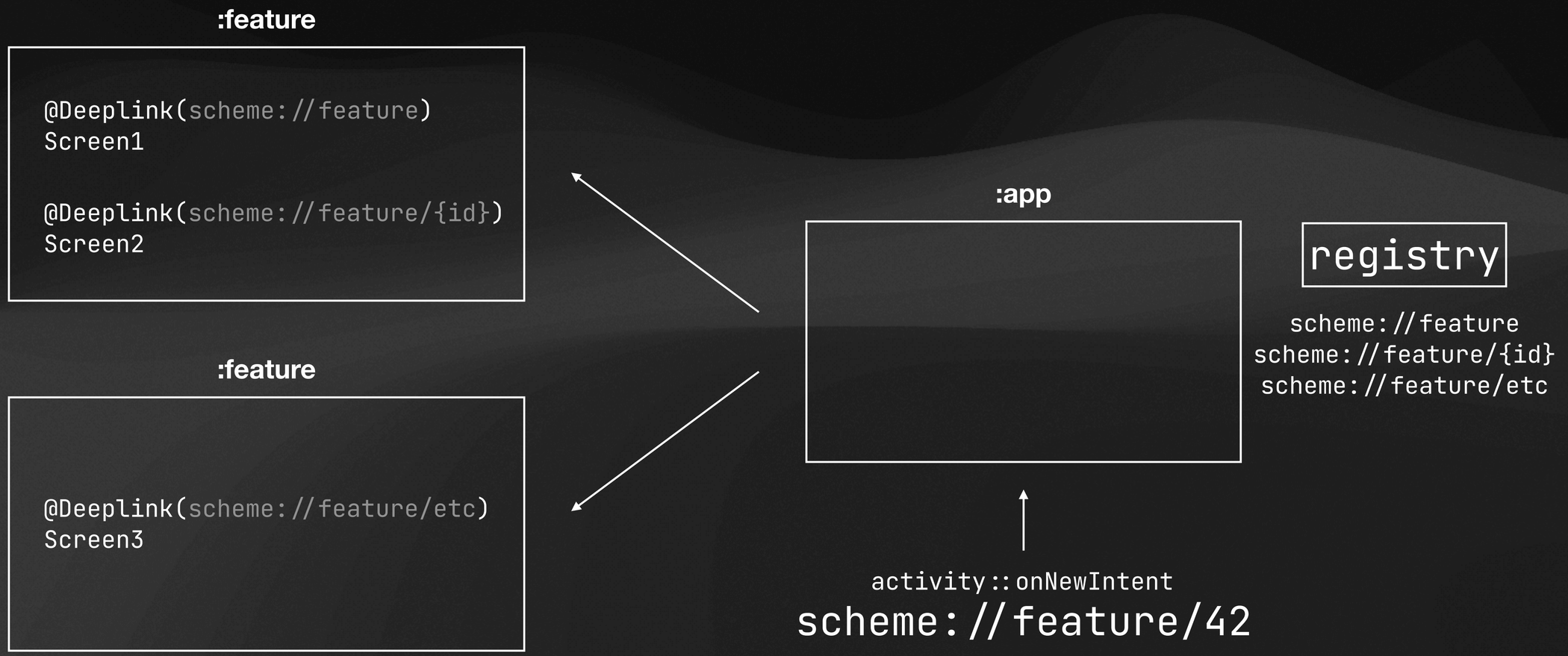


Схема процесса
Многомодульность

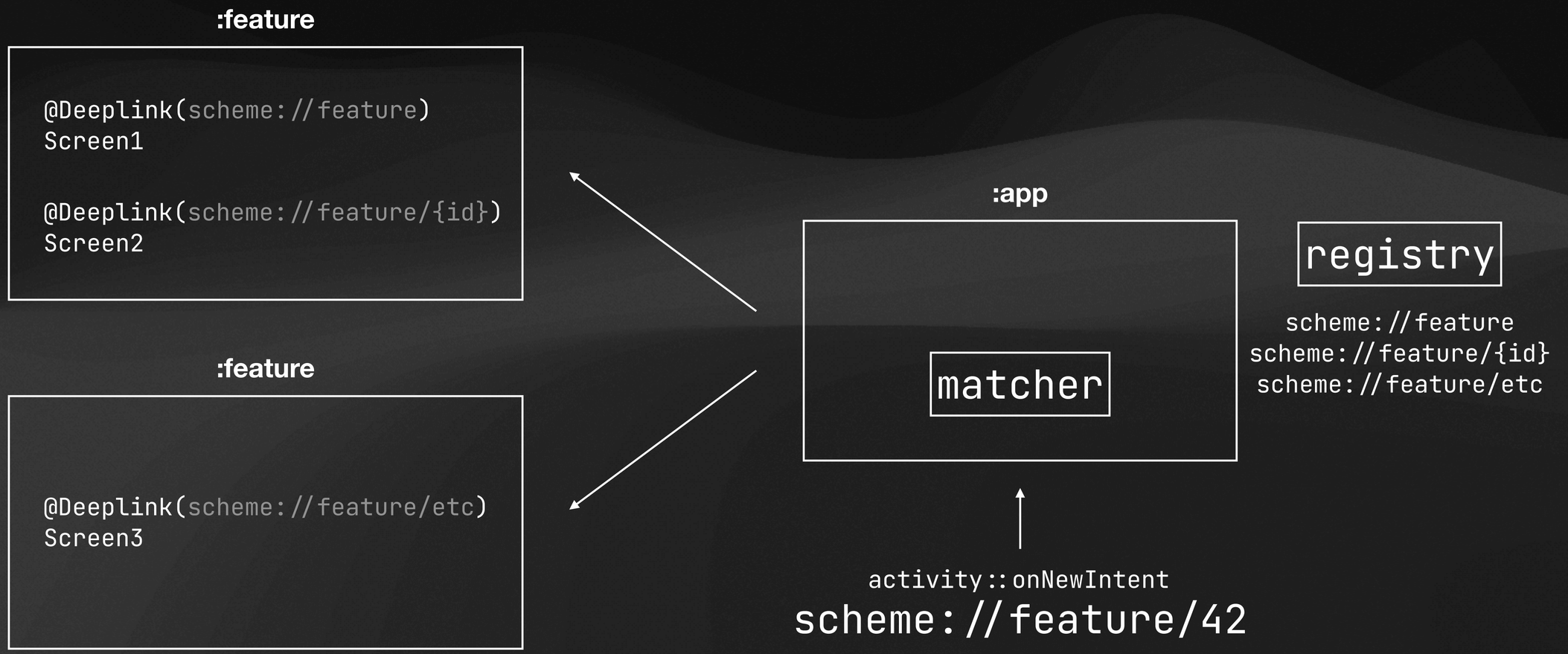


Схема процесса
Многомодульность

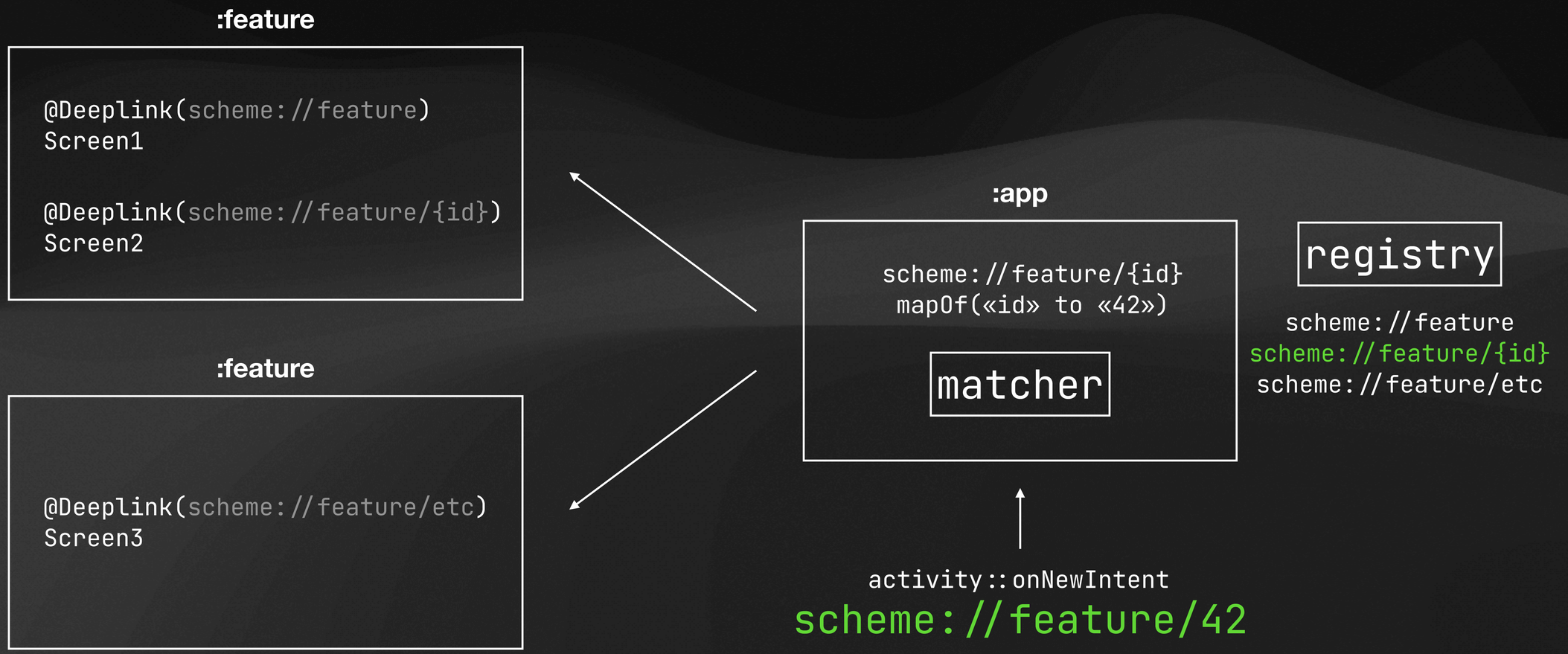


Схема процесса
Многомодульность

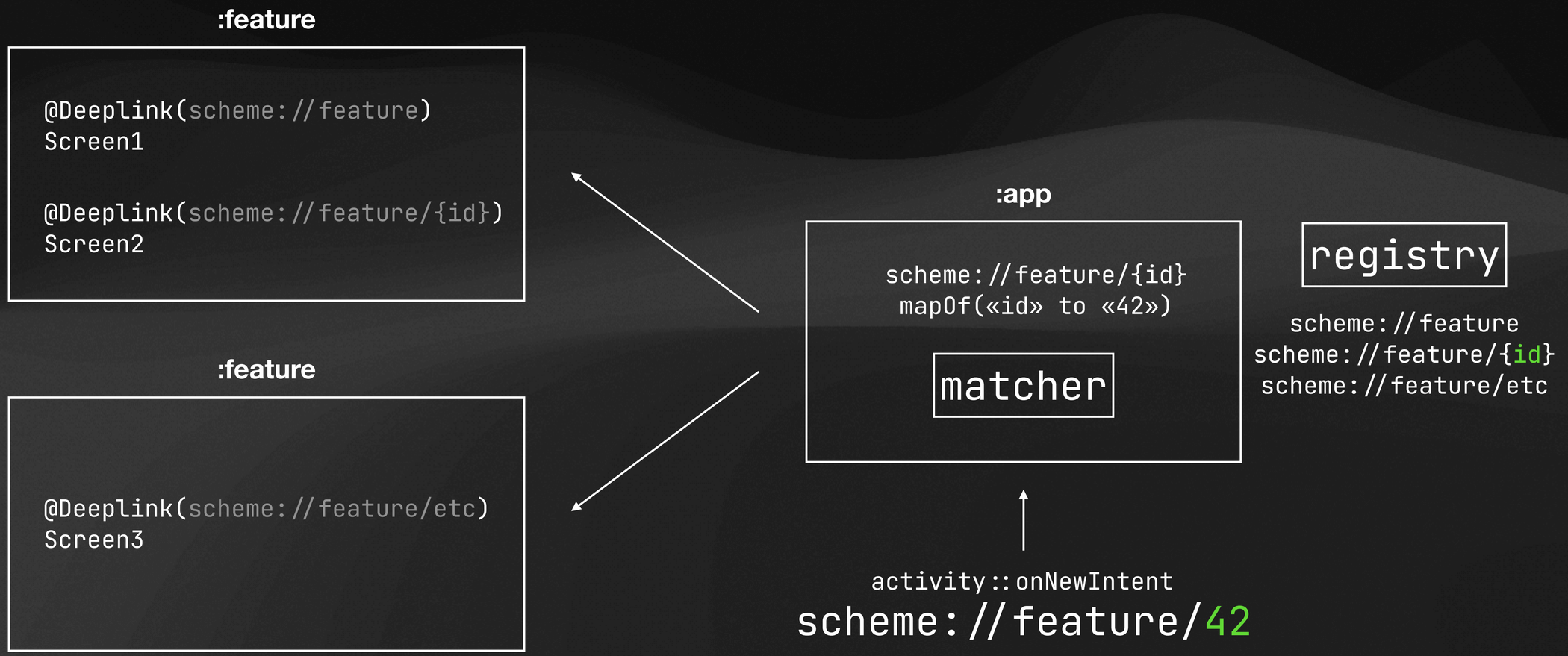


Схема процесса
Многомодульность

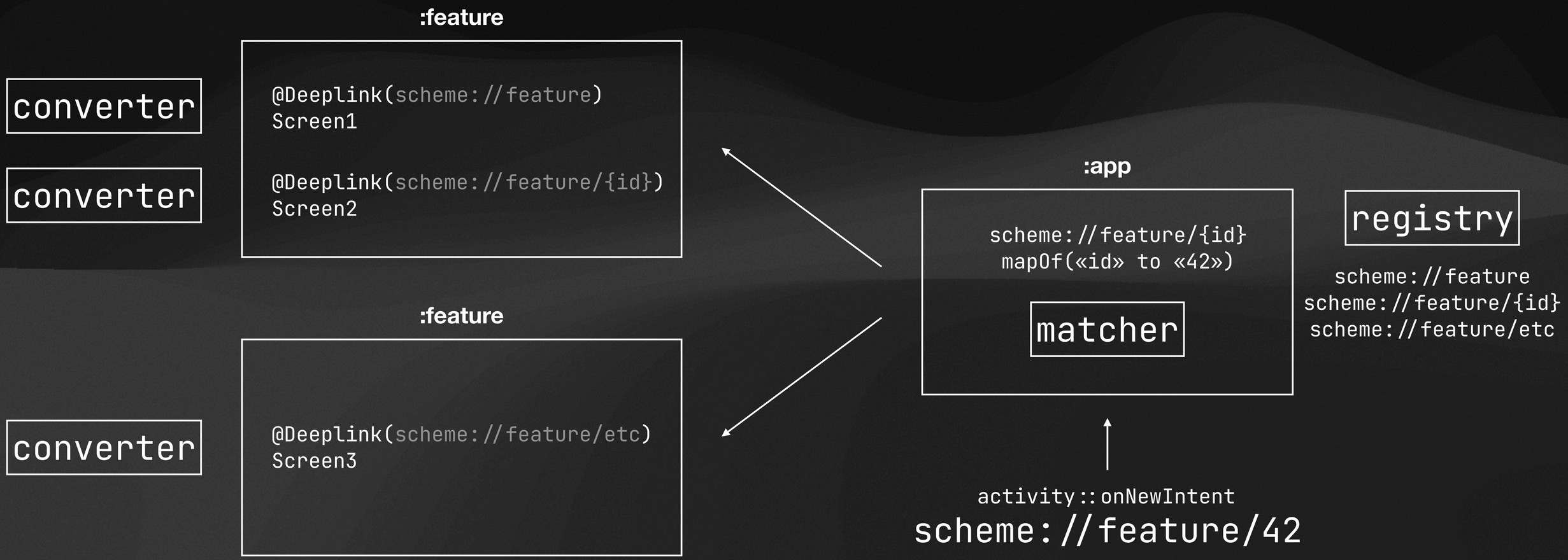


Схема процесса
Многомодульность

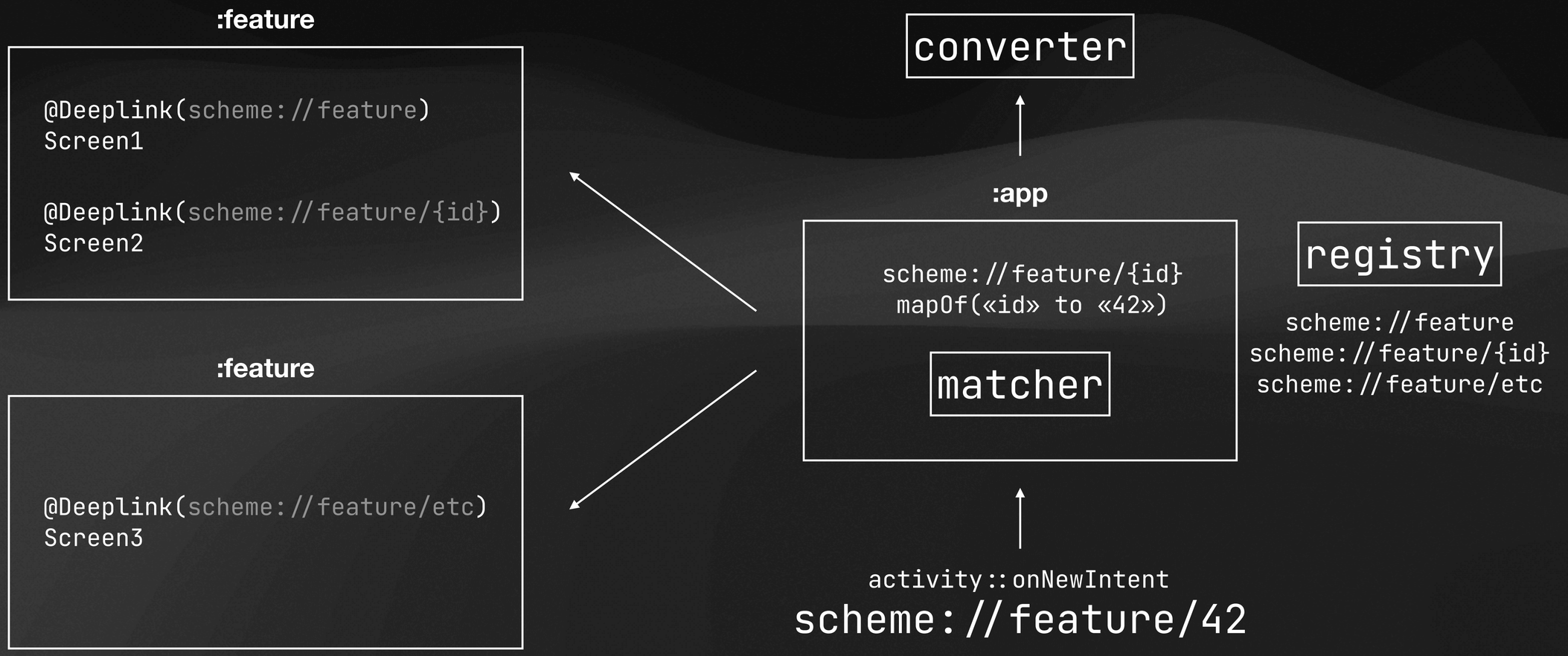


Схема процесса
Многомодульность

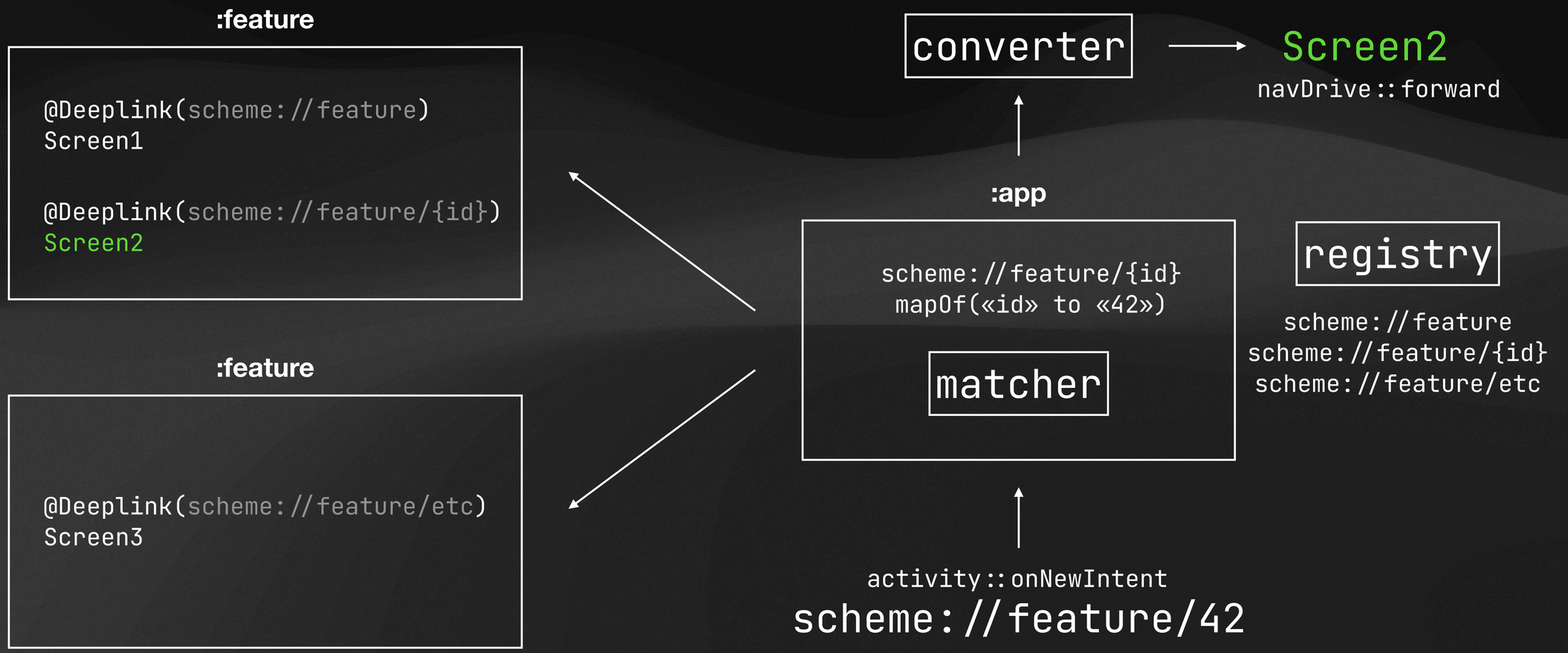


Схема процесса
Многомодульность

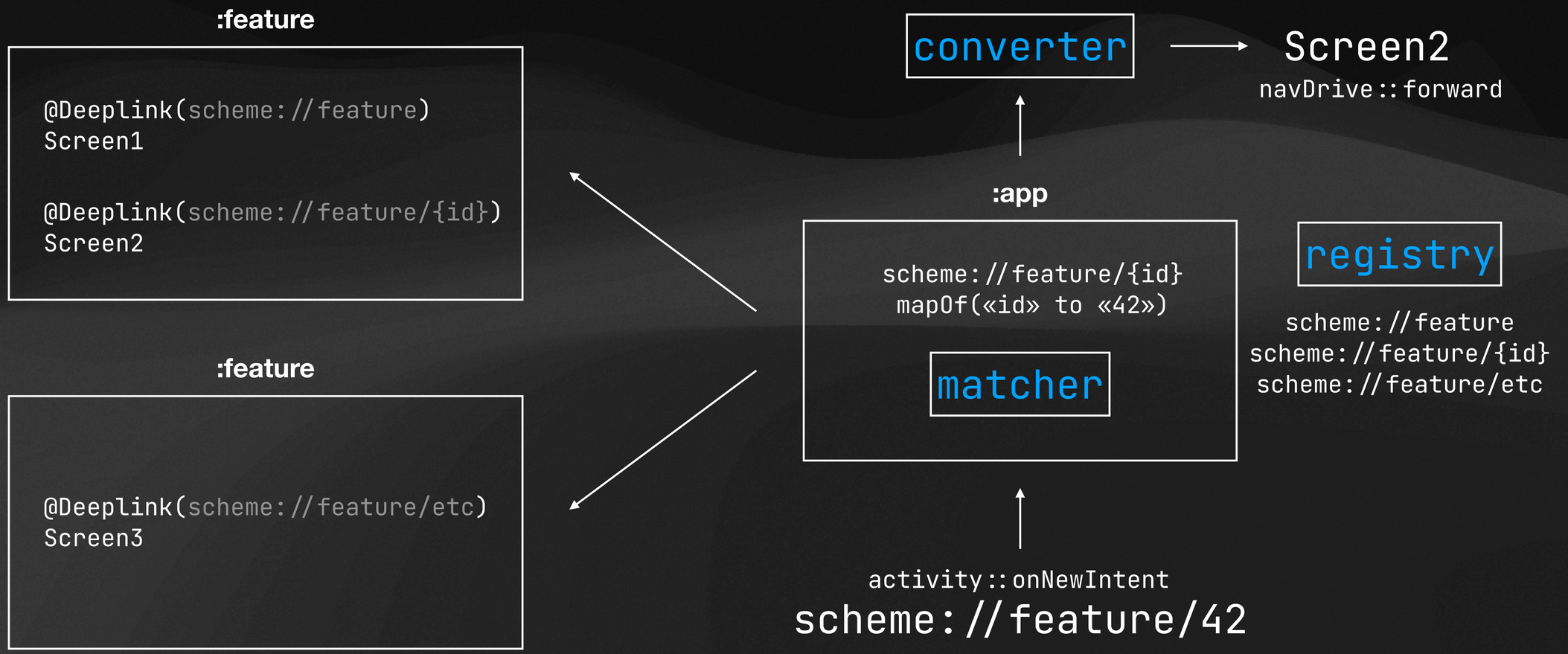


Схема процесса

```
val registry: DeeplinkRegistry
val matcher = DeeplinkMatcher(registry.deeplinkPatterns)
```

MainActivity::onNewIntent

Схема процесса

```
val registry: DeeplinkRegistry
val matcher = DeeplinkMatcher(registry.deeplinkPatterns)

// scheme://feature/{id}
// mapOf("id" to "42")
val (pattern, placeholders) = matcher.match("scheme://feature/42")
```

MainActivity::onNewIntent

Схема процесса

```
val registry: DeeplinkRegistry
val matcher = DeeplinkMatcher(registry.deeplinkPatterns)

// scheme://feature/{id}
// mapOf("id" to "42")
val (pattern, placeholders) = matcher.match("scheme://feature/42")
val converter = registry.converters[pattern]
```

MainActivity::onNewIntent

Схема процесса

```
val registry: DeeplinkRegistry
val matcher = DeeplinkMatcher(registry.deeplinkPatterns)

// scheme://feature/{id}
// mapOf("id" to "42")
val (pattern, placeholders) = matcher.match("scheme://feature/42")
val converter = registry.converters[pattern]

// Screen(id = 42)
val screen = converter.convert(placeholders)
findNavDrive().forward(screen)
```

MainActivity::onNewIntent

Схема процесса

```
val registry: DeeplinkRegistry
val matcher = DeeplinkMatcher(registry.deeplinkPatterns)

// scheme://feature/{id}
// mapOf("id" to "42")
val (pattern, placeholders) = matcher.match("scheme://feature/42")
val converter = registry.converters[pattern]

// Screen(id = 42)
val screen = converter.convert(placeholders)
findNavDrive().forward(screen)
```

`MainActivity::onNewIntent`

Кодогенерация

Registry

```
@Deeplink("scheme://feature/{id}")
class Screen1(val id: Int)
```

```
@Deeplink("scheme://feature")
object Screen2
```

Кодогенерация

Registry

```
@Deeplink("scheme://feature/{id}")
class Screen1(val id: Int)
```

```
@Deeplink("scheme://feature")
object Screen2
```

```
class FeatureDeeplinkRegistry {

    val converters = mapOf(
        "scheme://feature/{id}" to Screen1Converter()
        "scheme://feature" to Screen2Converter()
    )

    val deeplinkPatterns = converters.keys
}
```

build/generated/FeatureDeeplinkRegistry.kt

Кодогенерация
Converter

```
class Screen(val id: Int)
```

Кодогенерация
Converter

```
class Screen(val id: Int)

class ScreenConverter {
                    ("id" to "42")
    fun convert(placeholders: Map<String, String>): Screen {


    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int)

class ScreenConverter {
                    ("id" to "42")
    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]
        return Screen(id = id)
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int)

class ScreenConverter {
                    ("id" to "42")
    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]
        return Screen(id = id)*
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int)

class ScreenConverter {
                    ("id" to "42")
    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]?.toInt()
        return Screen(id = id)
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int)

class ScreenConverter {
                    ("id" to "42")
    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]?.toInt()
        return Screen(id = id)*
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int)

class ScreenConverter {
                    ("id" to "42")
    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]!!.toInt()
        return Screen(id = id)
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42)

class ScreenConverter {
                    ("id" to "42")
    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]!!.toInt()
        return Screen(id = id)
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42)

class ScreenConverter {
                        emptyMap()
    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]!!.toInt()
        return Screen(id = id)*
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42)

class ScreenConverter {

    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]?.toInt()
        return Screen(id ?: 42)
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42)

class ScreenConverter {

    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]?.toInt()
        return Screen(id ?: 42)*
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42)

class ScreenConverter {

    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]?.toInt()
        return if (id ≠ null) Screen(id = id) else Screen()
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42)

class ScreenConverter {

    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]?.toInt()
        return if (id ≠ null) Screen(id = id) else Screen()
    }
}
```

Screen(id) Screen(42)

build/generated/ScreenConverter.kt

Кодогенерация

Converter

```
class Screen(val id: Int = 42) { constructor() : this(id = -1) }

class ScreenConverter {

    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]?.toInt()
        return if (id ≠ null) Screen(id = id) else Screen()
    }
}
```

Screen(id)

Screen(42)

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42) { constructor() : this(id = -1) }

class ScreenConverter {

    fun convert(placeholders: Map<String, String>): Screen {
        val id = placeholders["id"]?.toInt()
        return if (id ≠ null) Screen(id = id) else Screen()*
    }                                  Screen(id)          Screen(-1)
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42)
```

Кодогенерация
Converter

```
fun main() = Screen()

class Screen(val id: Int = 42)
```

Кодогенерация
Converter

```
fun main() = Screen() // Screen(0, 1, (DefaultConstructorMarker) null)

class Screen(val id: Int = 42)
```

Кодогенерация
Converter

```
fun main() = Screen() // Screen(0, 1, (DefaultConstructorMarker) null)

class Screen(val id: Int = 42) {

    // syntetic
    constructor (id: Int, mask: Int, marker: DefaultConstructorMarker) {


    }
}
```

Кодогенерация
Converter

```
fun main() = Screen() // Screen(0, 1, (DefaultConstructorMarker) null)

class Screen(val id: Int = 42) {

    // syntetic                                  kotlin.jvm.internal.DefaultConstructorMarker
    constructor (id: Int, mask: Int, marker: DefaultConstructorMarker) {


    }
}
```

Кодогенерация
Converter

```
fun main() = Screen() // Screen(0, 1, (DefaultConstructorMarker) null)

class Screen(val id: Int = 42) {

    // syntetic     0                    kotlin.jvm.internal.DefaultConstructorMarker
    constructor (id: Int, mask: Int, marker: DefaultConstructorMarker) {


    }
}
```

Кодогенерация
Converter

```
fun main() = Screen() // Screen(0, 1, (DefaultConstructorMarker) null)

class Screen(val id: Int = 42) {

    // syntetic      0          00000001      kotlin.jvm.internal.DefaultConstructorMarker
    constructor (id: Int, mask: Int, marker: DefaultConstructorMarker) {


    }
}
```

Кодогенерация
Converter

```
fun main() = Screen() // Screen(0, 1, (DefaultConstructorMarker) null)

class Screen(val id: Int = 42) {

    // syntetic     0         00000001     kotlin.jvm.internal.DefaultConstructorMarker
    constructor (id: Int, mask: Int, marker: DefaultConstructorMarker) {
        if (mask & 1 ≠ 0) id = 42
        this(id)
    }
}
```

Кодогенерация
Converter

```
fun main() = Screen() // Screen(0, 1, (DefaultConstructorMarker) null)

class Screen(val id: Int = 42) {

    // syntetic     0         00000001     kotlin.jvm.internal.DefaultConstructorMarker
    constructor (id: Int, mask: Int, marker: DefaultConstructorMarker) {
        if (mask & 1 != 0) id = 42
        this(id)
    }
}

val marker = Class.forName("kotlin.jvm.internal.DefaultConstructorMarker")
val ctor = Screen::javaClass.getDeclaredConstructor(Int, Int, marker)
```

Кодогенерация
Converter

```
class Screen(val id: Int = 42)

class ScreenConverter {

    fun convert(placeholders: Map<String, String>): Screen {
        var mask = -1 // 1111111




    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42)

class ScreenConverter {

    fun convert(placeholders: Map<String, String>): Screen {
        var mask = -1 // 1111111

        val id = placeholders["id"]?.toInt()
        if (id ≠= null) mask = mask and 0xfffffffe // 1111110


    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42)

class ScreenConverter {

    fun convert(placeholders: Map<String, String>): Screen {
        var mask = -1 // 1111111

        val id = placeholders["id"]?.toInt()
        if (id ≠= null) mask = mask and 0xfffffffe // 1111110

        return ctor.newInstance(id, mask, null)
    }
}
```

build/generated/ScreenConverter.kt

Кодогенерация
Converter

```
class Screen(val id: Int = 42)

class ScreenConverter {

    fun convert(placeholders: Map<String, String>): Screen {
        var mask = -1 // 1111111

        val id = placeholders["id"]?.toInt()
        if (id ≠= null) mask = mask and 0xfffffffe // 1111110

        return if (mask == 0xfffffffe) Screen(id = id!!)
        else ctor.newInstance(id, mask, null)
    }
}
```

build/generated/ScreenConverter.kt

Deeplink matching - Jetpack navigation

Deeplink matching - Jetpack navigation

```
// main_graph.xml
<navigation>

    <fragment>
        <deeplink uri = "..." />
    </fragment>

    <include graph = "..." />

</navigation>
```

Deeplink matching - Jetpack navigation

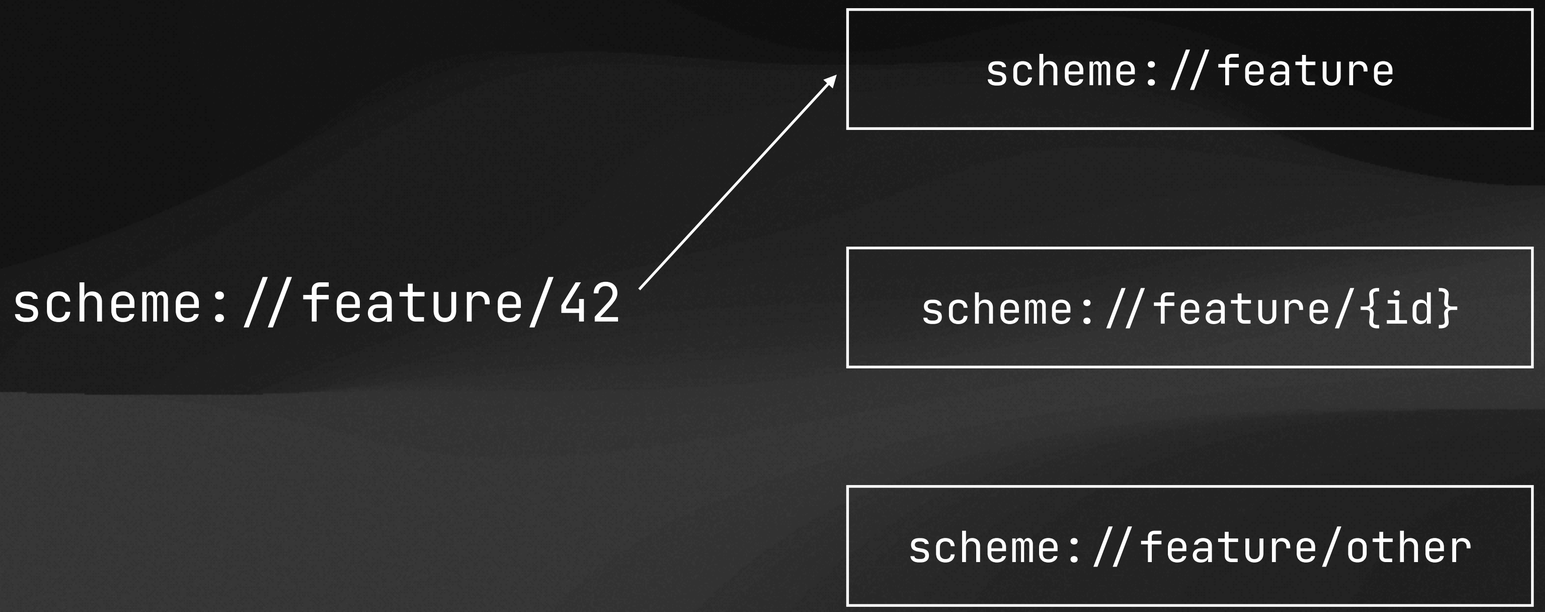
`scheme://feature`

`scheme://feature/42`

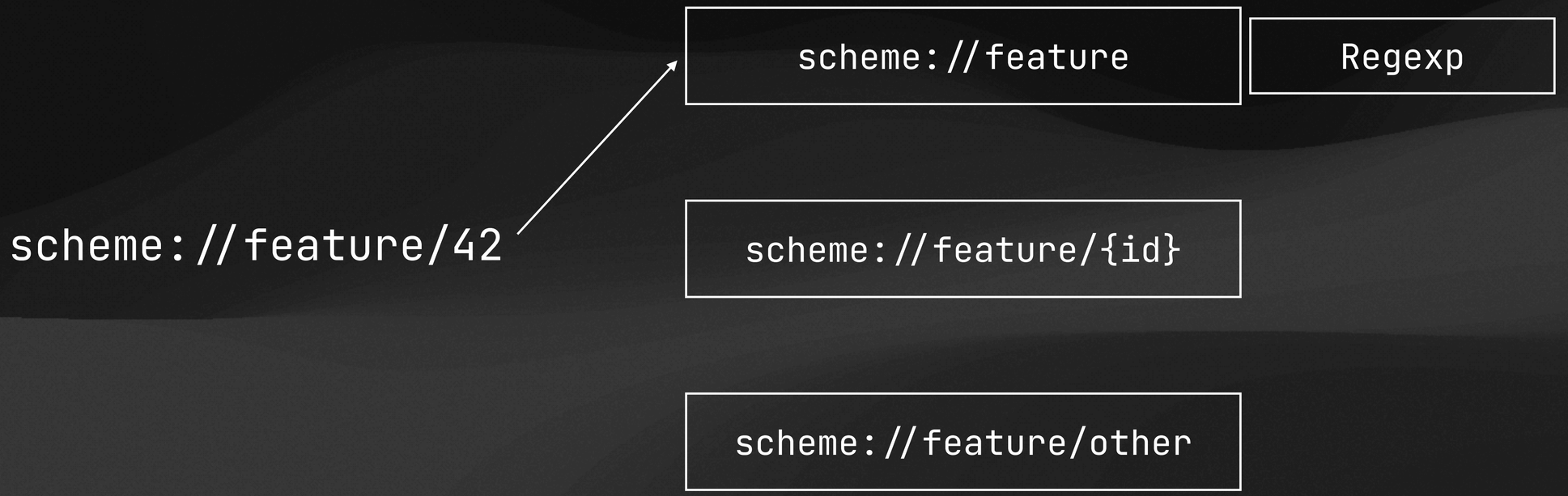
`scheme://feature/{id}`

`scheme://feature/other`

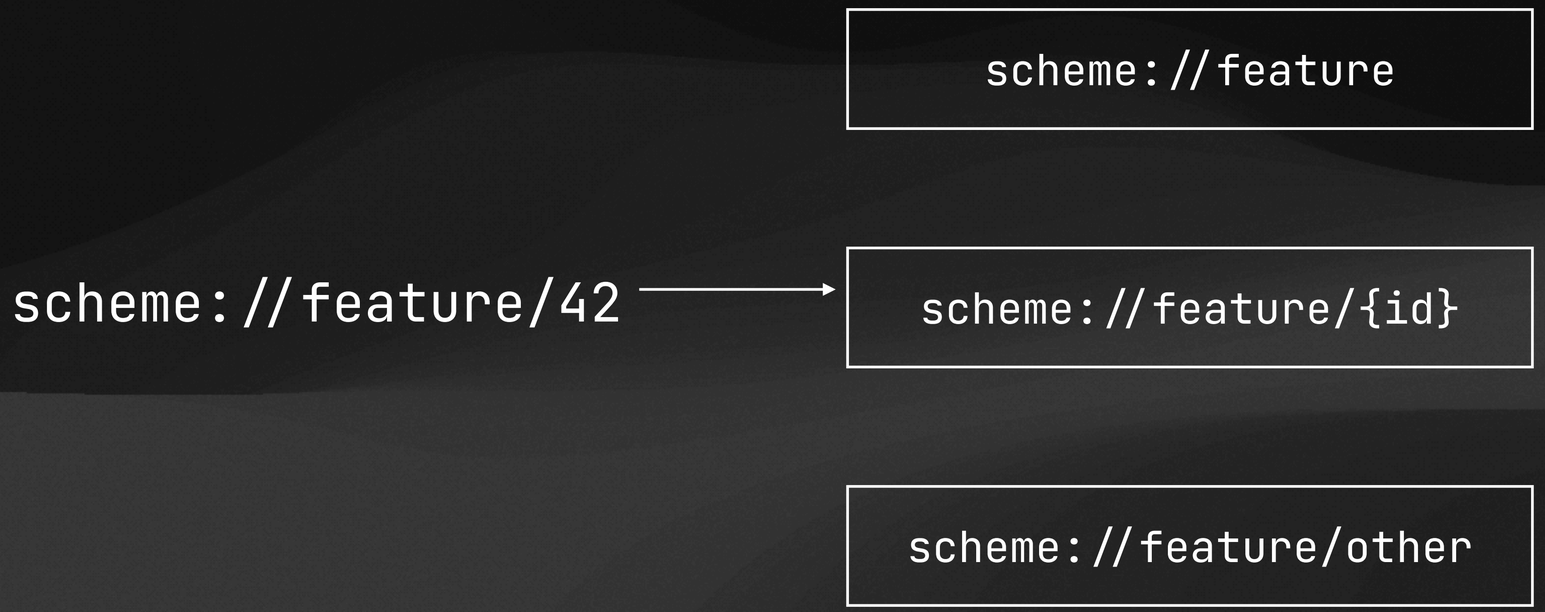
Deeplink matching - Jetpack navigation



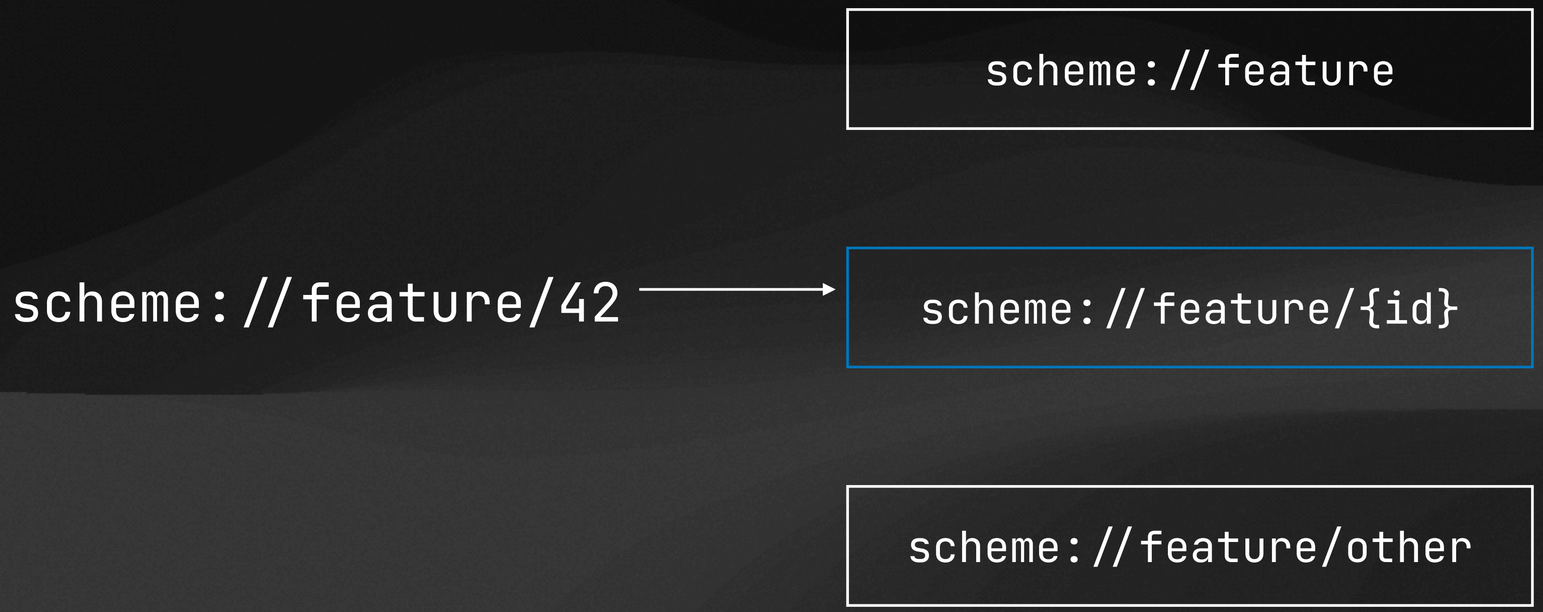
Deeplink matching - Jetpack navigation



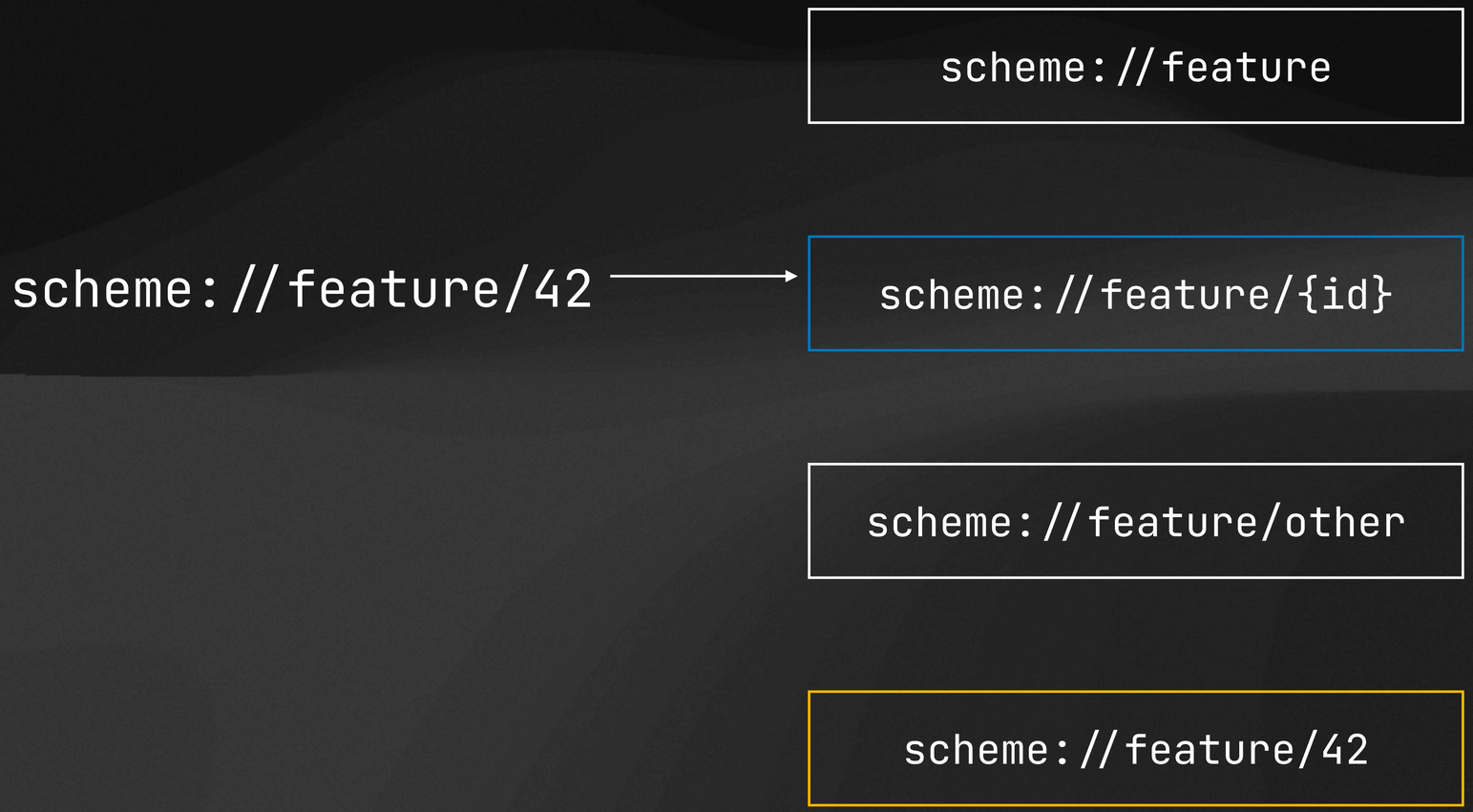
Deeplink matching - Jetpack navigation



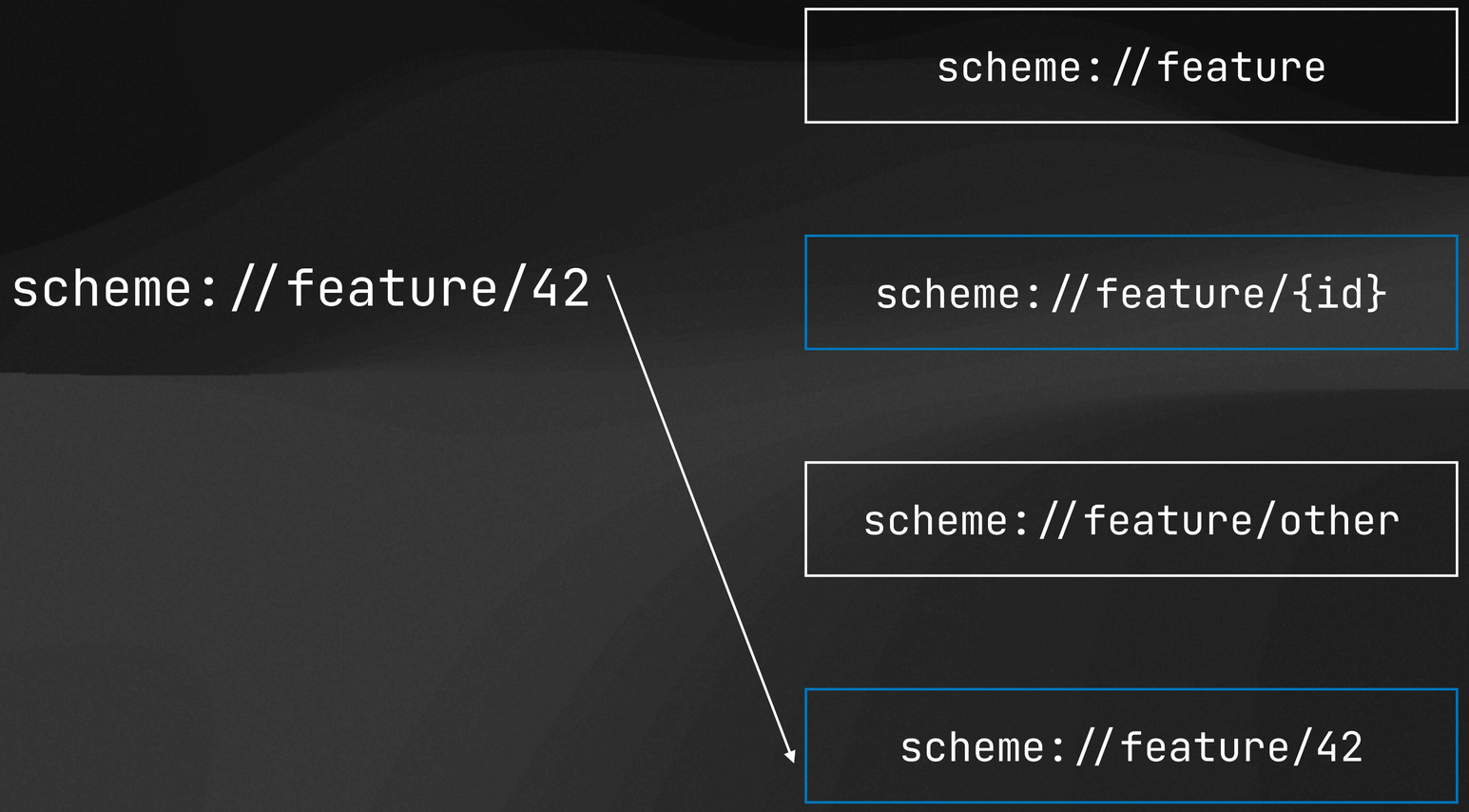
Deeplink matching - Jetpack navigation



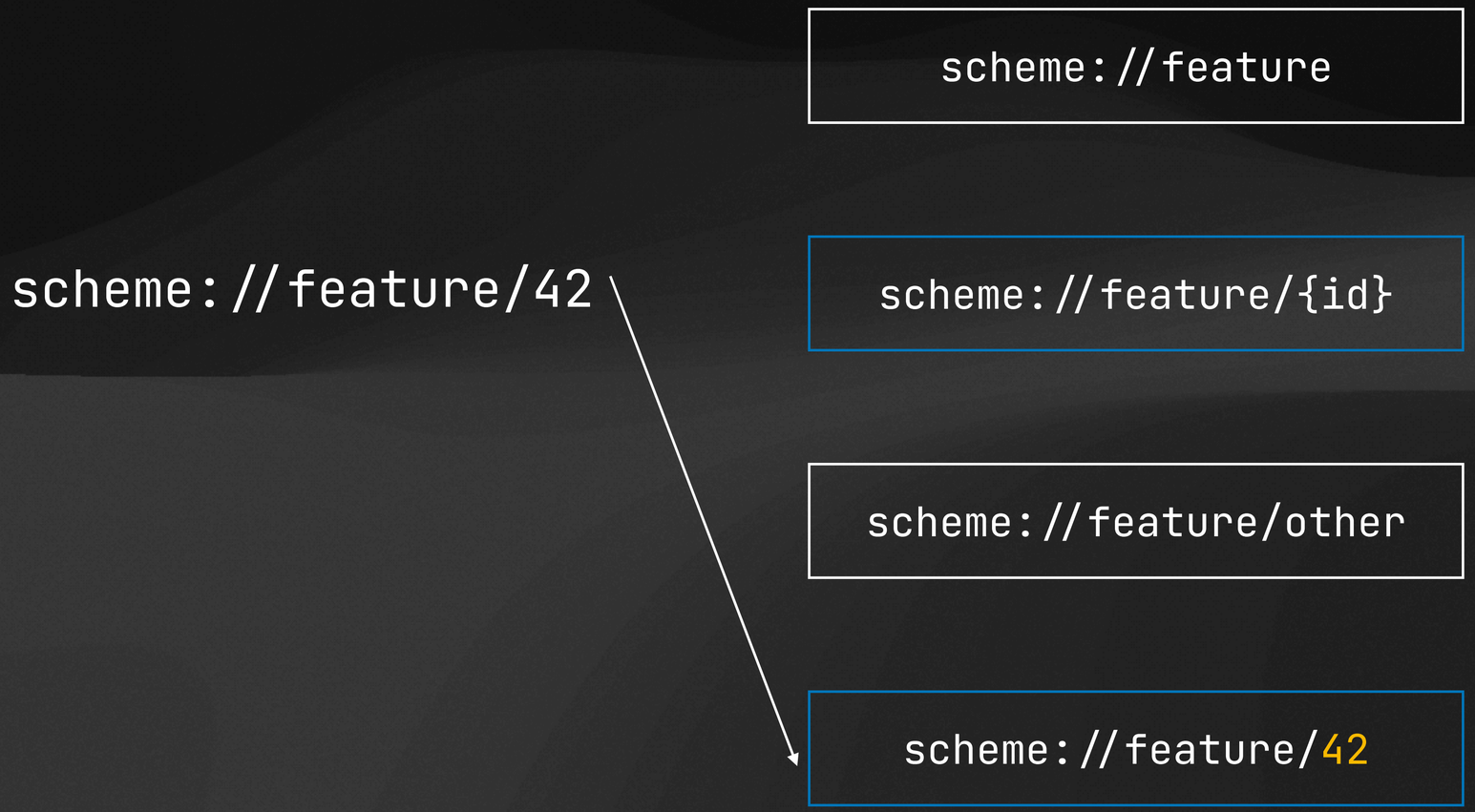
Deeplink matching - Jetpack navigation



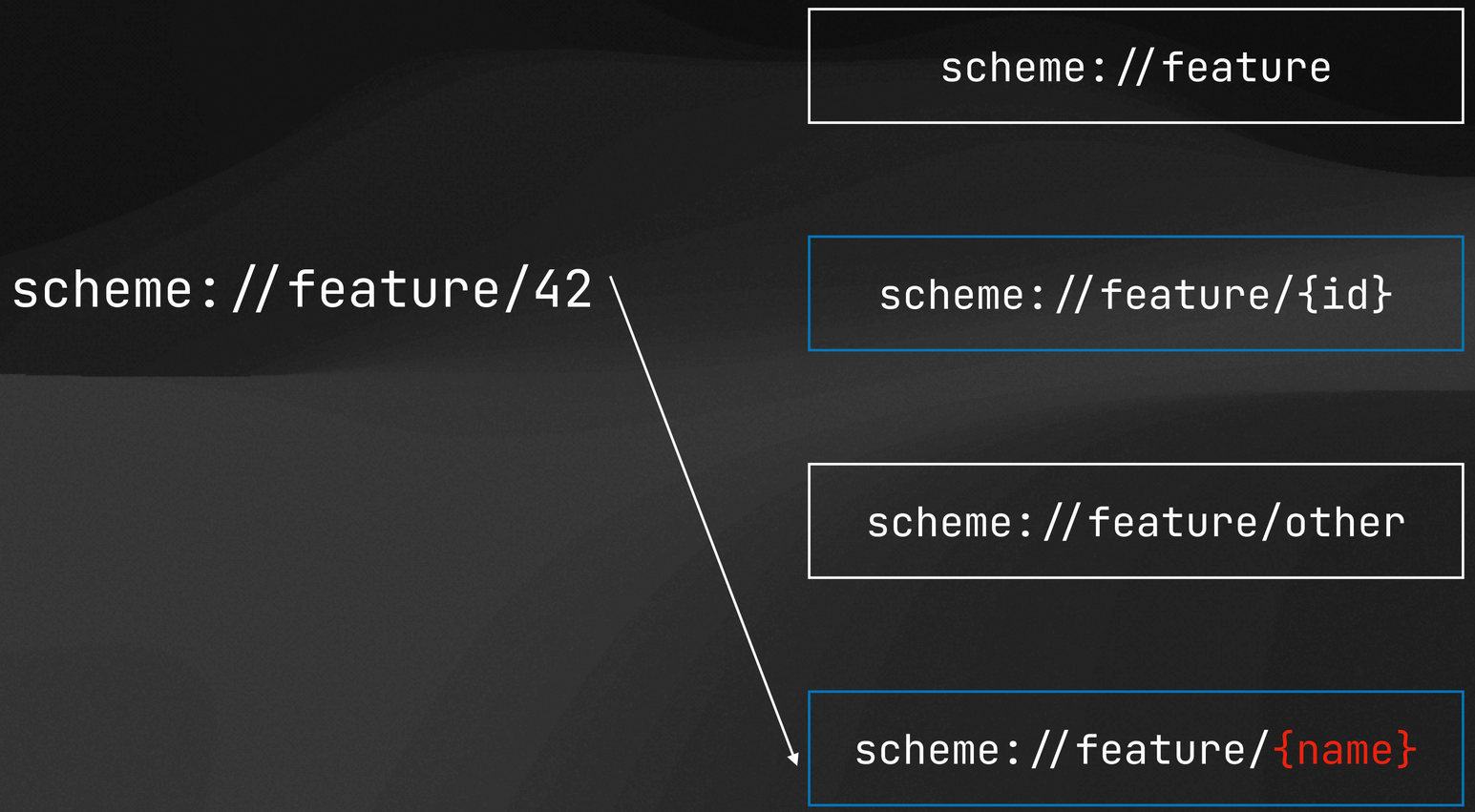
Deeplink matching - Jetpack navigation



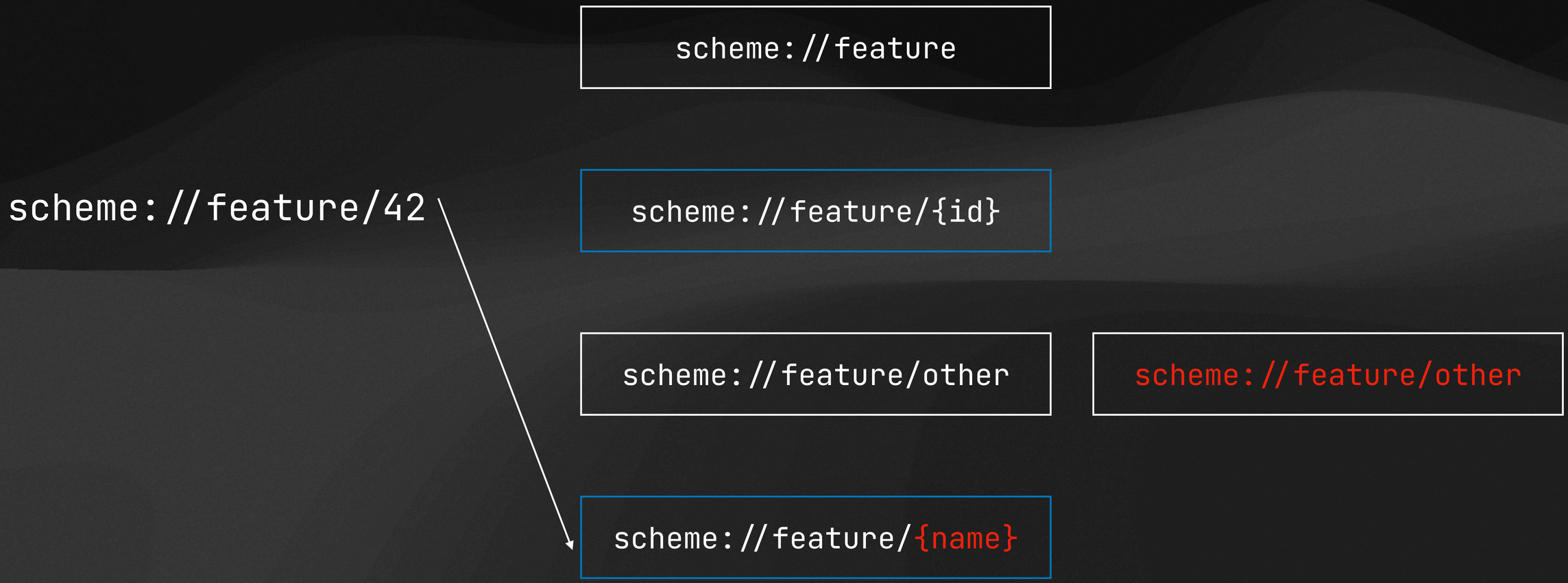
Deeplink matching - Jetpack navigation



Deeplink matching - Jetpack navigation



Deeplink matching - Jetpack navigation



Deeplink matching - Jetpack navigation

```
scheme://feature?from={from}
```

Deeplink matching - Jetpack navigation

Производительность

Шаблонов с плейсхолдером в query параметре(шт.)/match	Jetpack
40	**~1**
70	**~2**
100	**~3**
200	**~6**

Составление дерева

Составление дерева

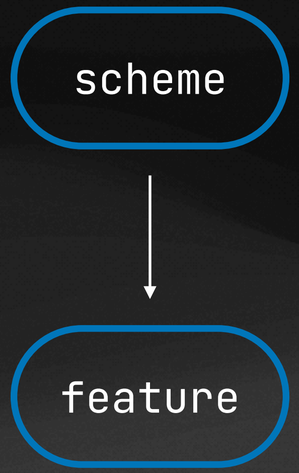
```
scheme://feature/42
scheme://feature/{id}
scheme://feature/{greedy}
```

Составление дерева

```
scheme://feature/42
scheme://feature/{id}
scheme://feature/{greedy}
```

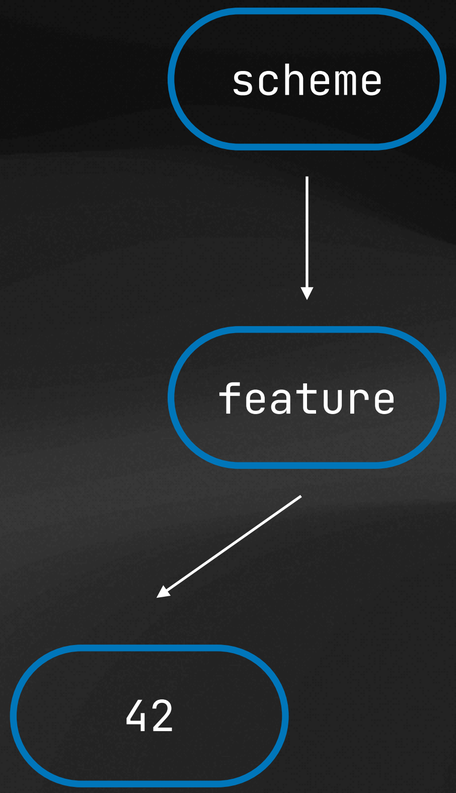
Составление дерева

```
scheme://feature/42
scheme://feature/{id}
scheme://feature/{greedy}
```



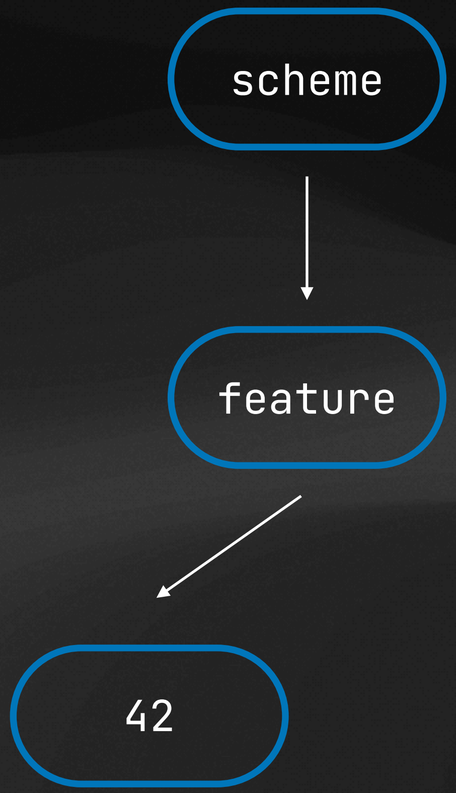
Составление дерева

```
scheme://feature/42
scheme://feature/{id}
scheme://feature/{greedy}
```



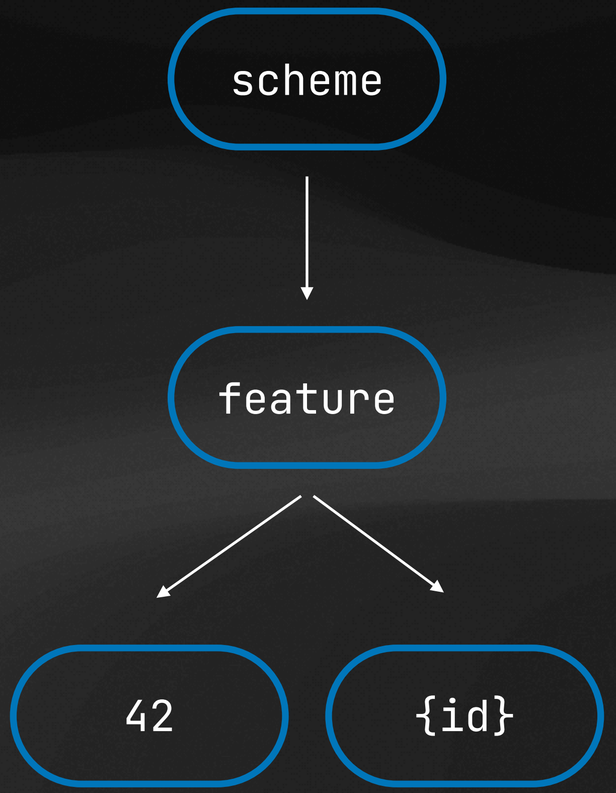
Составление дерева

```
scheme://feature/42
scheme://feature/{id}
scheme://feature/{greedy}
```



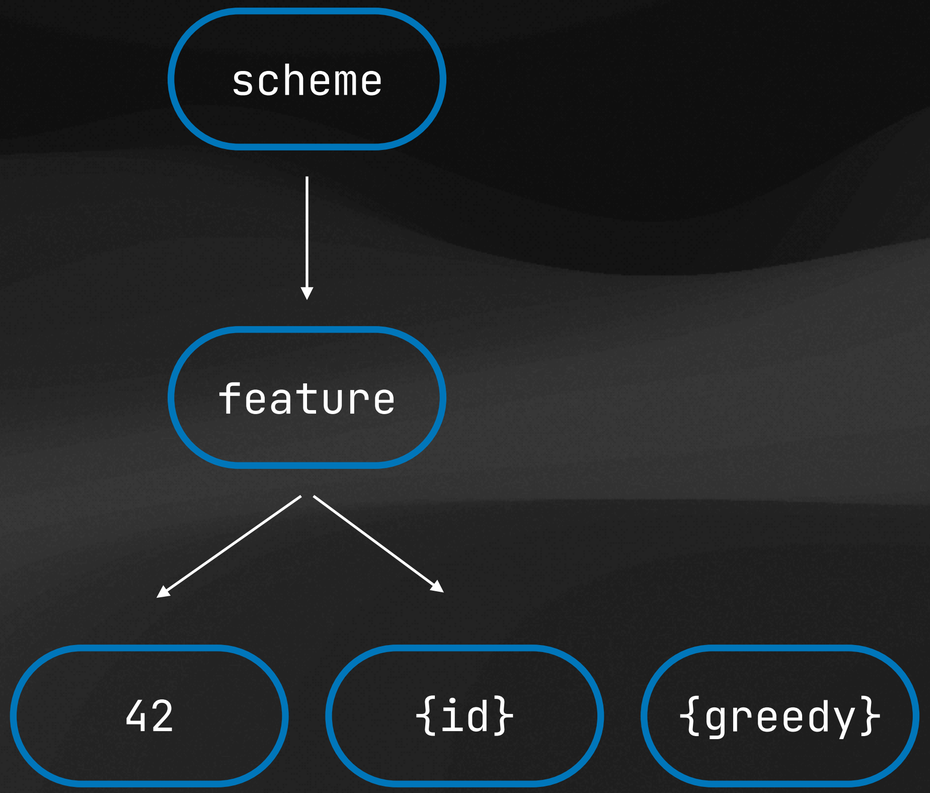
Составление дерева

```
scheme://feature/42
scheme://feature/{id}
scheme://feature/{greedy}
```



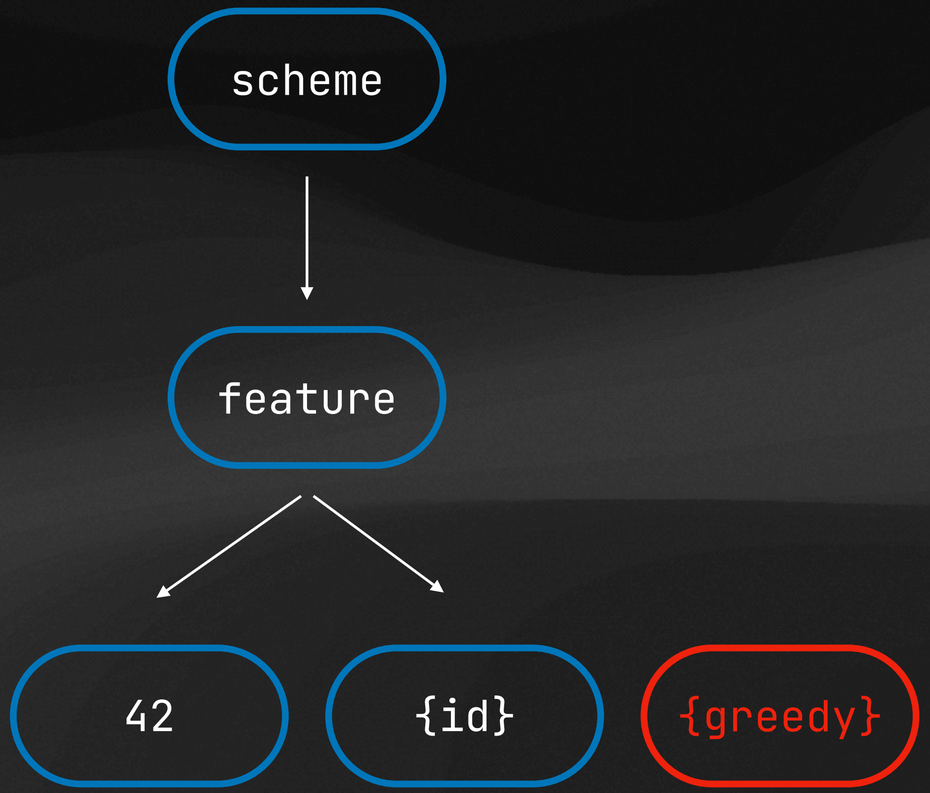
Составление дерева

```
scheme://feature/42
scheme://feature/{id}
scheme://feature/{greedy}
```



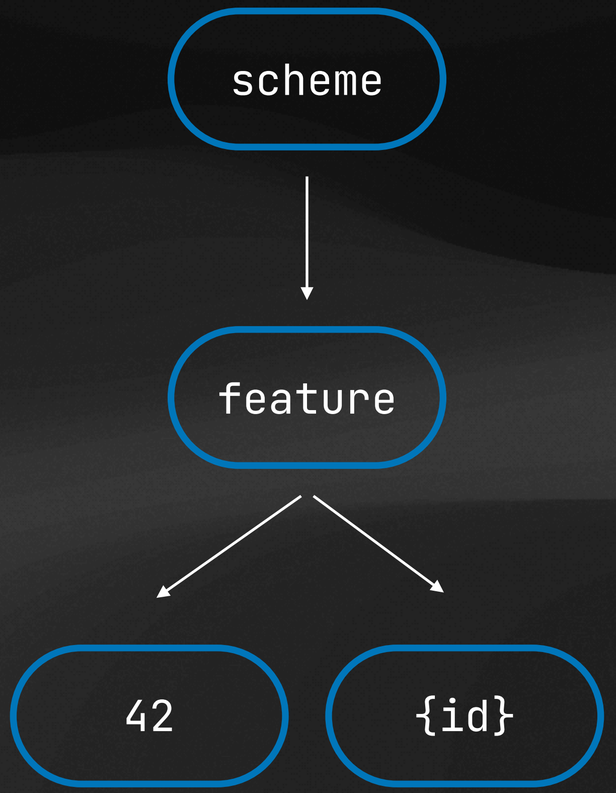
Составление дерева

```
scheme://feature/42
scheme://feature/{id}
scheme://feature/{greedy}
```



Составление дерева

```
scheme://feature/42
scheme://feature/{id}
```

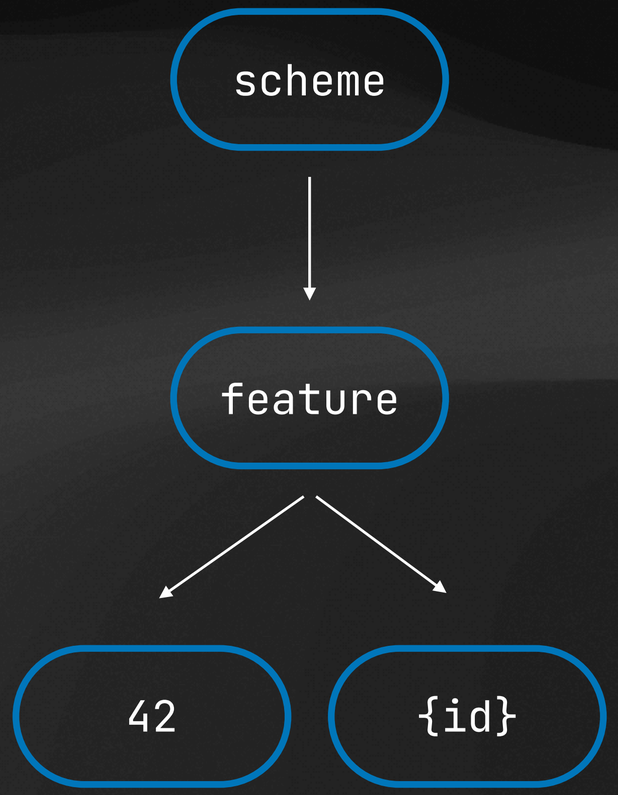


Сопоставление в дереве

Сопоставление в дереве

Шаблоны

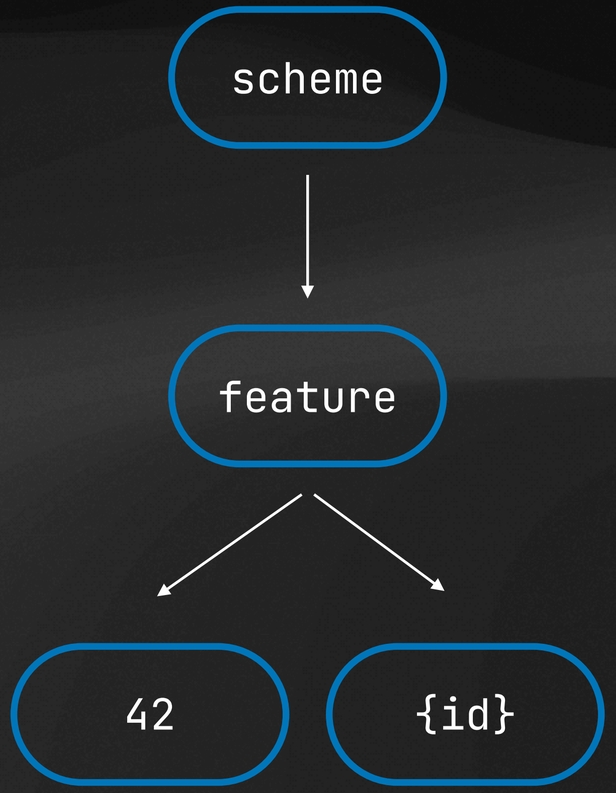
```
scheme://feature/42
scheme://feature/{id}
```



Сопоставление в дереве

Шаблоны
`scheme://feature/42`
`scheme://feature/{id}`

Диплинк
`scheme://feature`



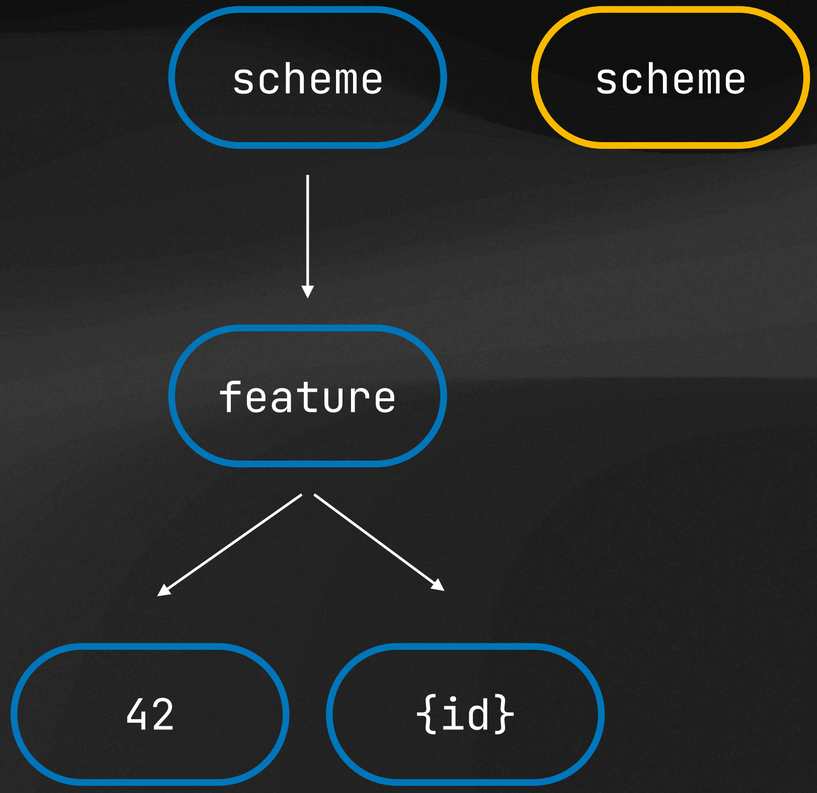
Сопоставление в дереве

Шаблоны

```
scheme://feature/42
scheme://feature/{id}
```

Диплинк

```
scheme://feature
```



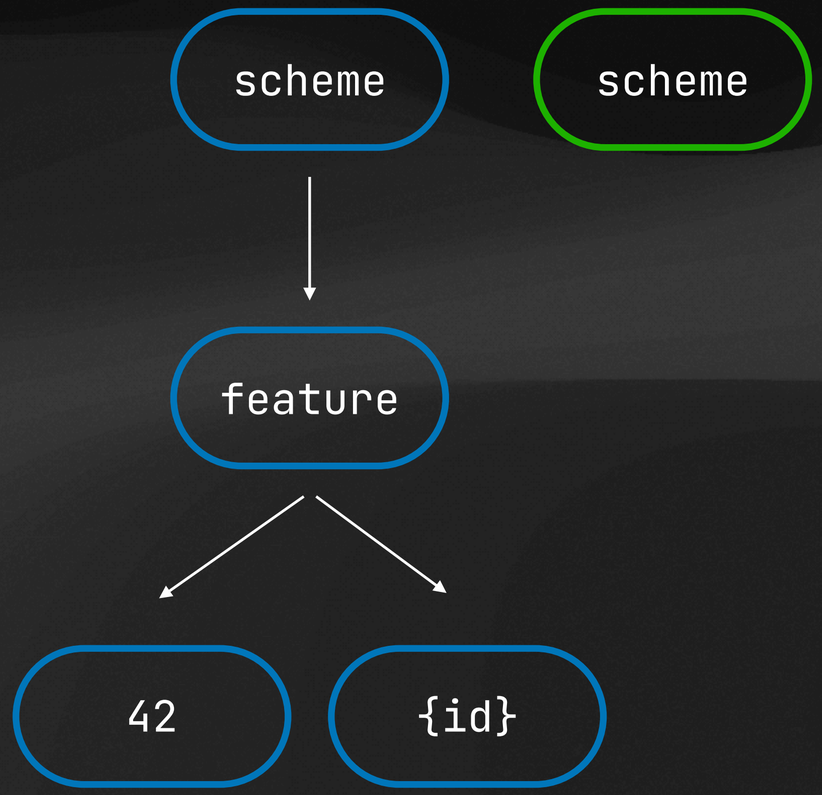
Сопоставление в дереве

Шаблоны

```
scheme://feature/42
scheme://feature/{id}
```

Диплинк

```
scheme://feature
```



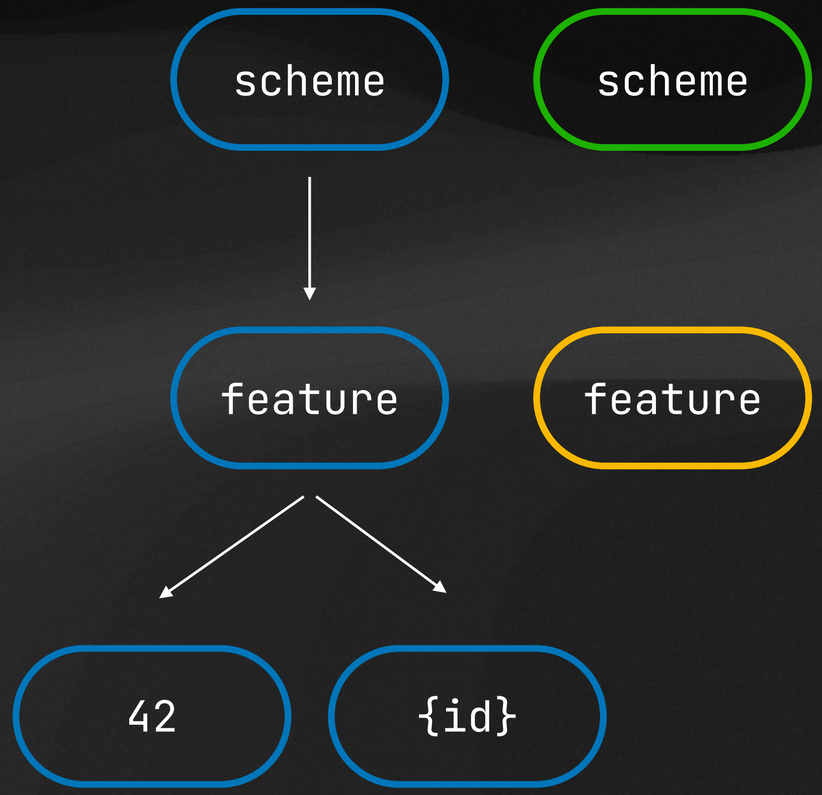
Сопоставление в дереве

Шаблоны

```
scheme://feature/42
scheme://feature/{id}
```

Диплинк

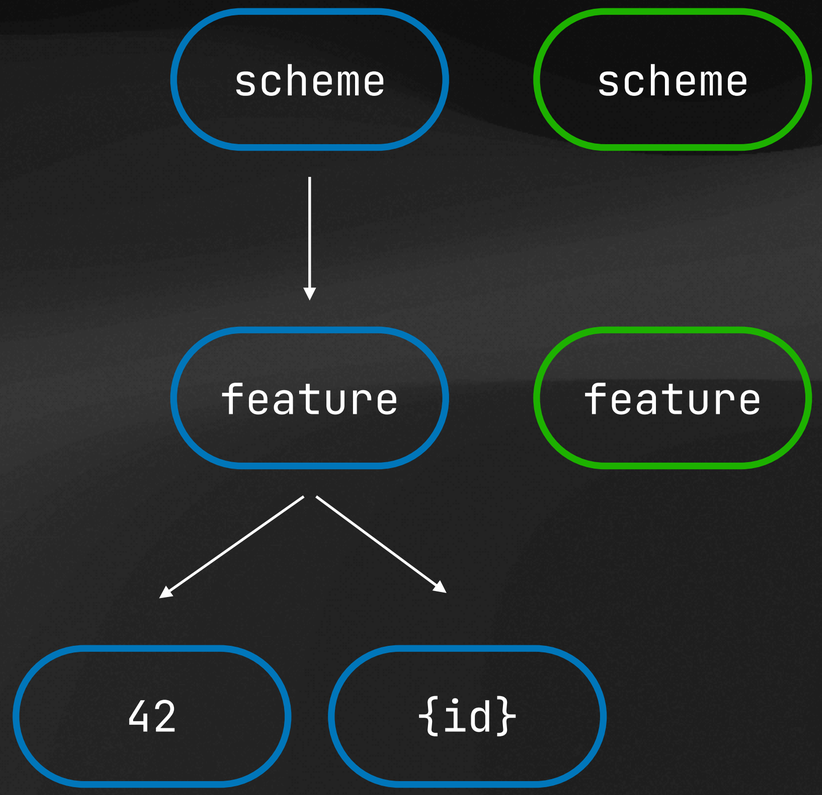
```
scheme://feature
```



Сопоставление в дереве

Шаблоны
scheme://feature/42
scheme://feature/{id}

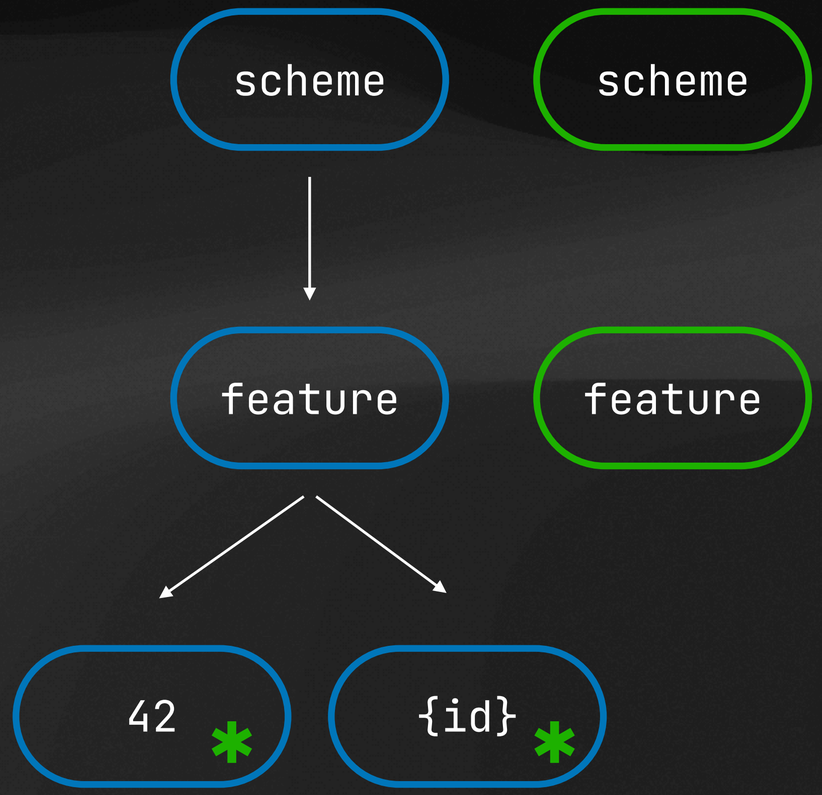
Диплинк
scheme://feature



Сопоставление в дереве

Шаблоны
scheme://feature/42
scheme://feature/{id}

Диплинк
scheme://feature



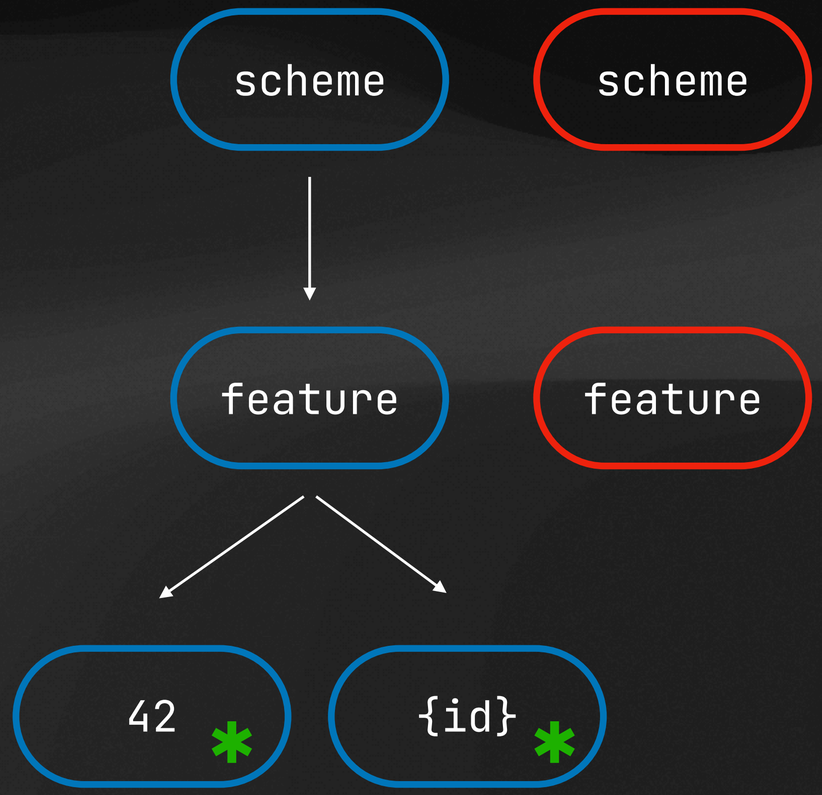
Сопоставление в дереве

Шаблоны

```
scheme://feature/42
scheme://feature/{id}
```

Диплинк

```
scheme://feature
```



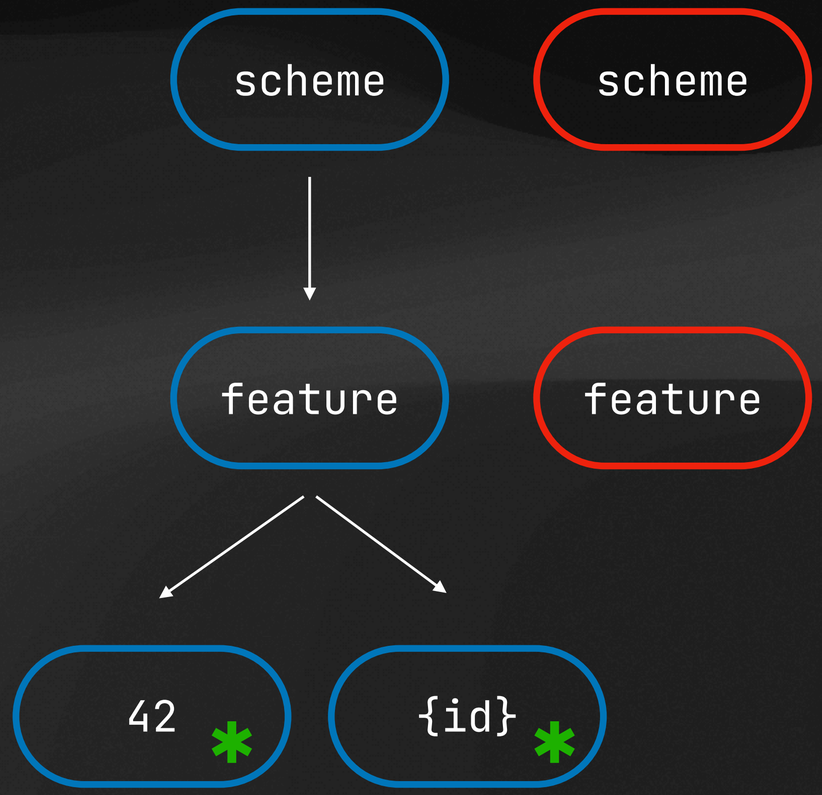
Сопоставление в дереве

Шаблоны

```
scheme://feature
scheme://feature/42
scheme://feature/{id}
```

Диплинк

```
scheme://feature
```



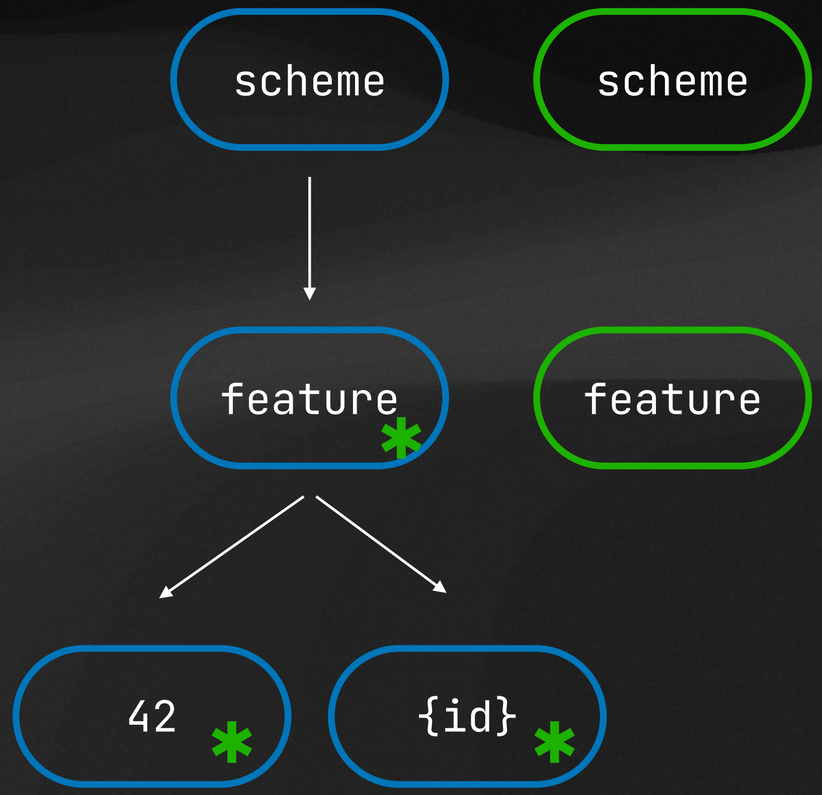
Сопоставление в дереве

Шаблоны

```
scheme://feature
scheme://feature/42
scheme://feature/{id}
```

Диплинк

```
scheme://feature
```



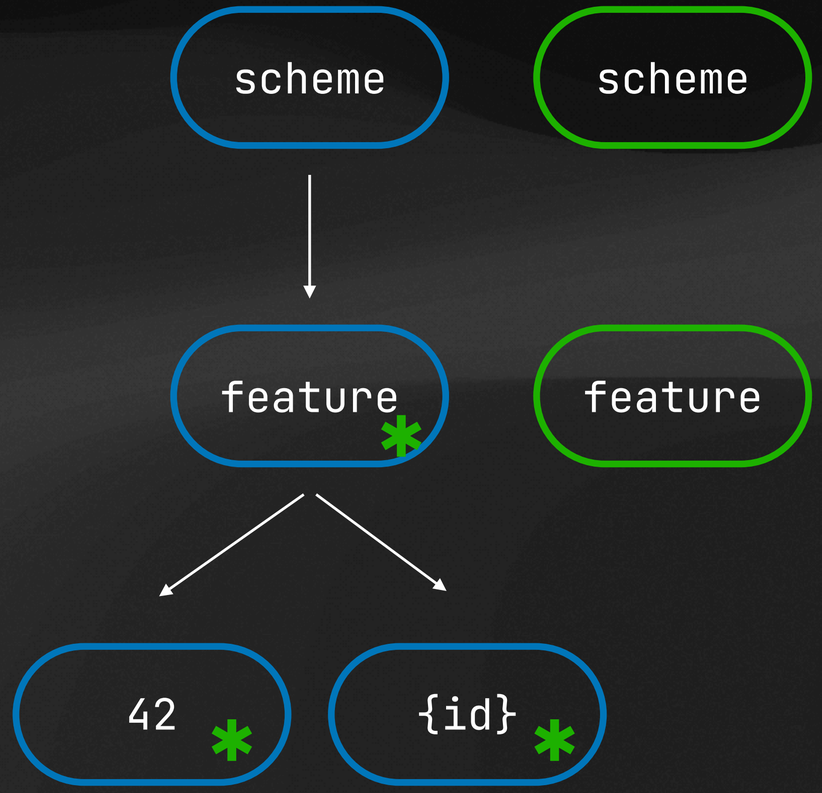
Сопоставление в дереве

Шаблоны

```
scheme://feature
scheme://feature/42
scheme://feature/{id}
```

Диплинк

```
scheme://feature/42
```



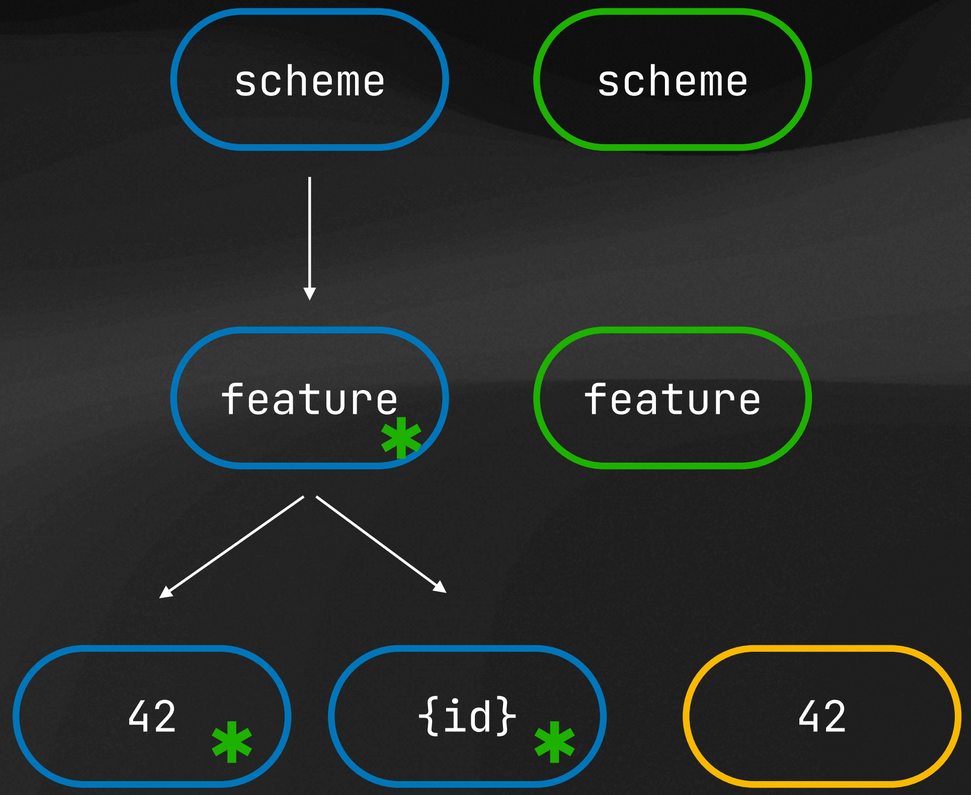
Сопоставление в дереве

Шаблоны

```
scheme://feature
scheme://feature/42
scheme://feature/{id}
```

Диплинк

```
scheme://feature/42
```



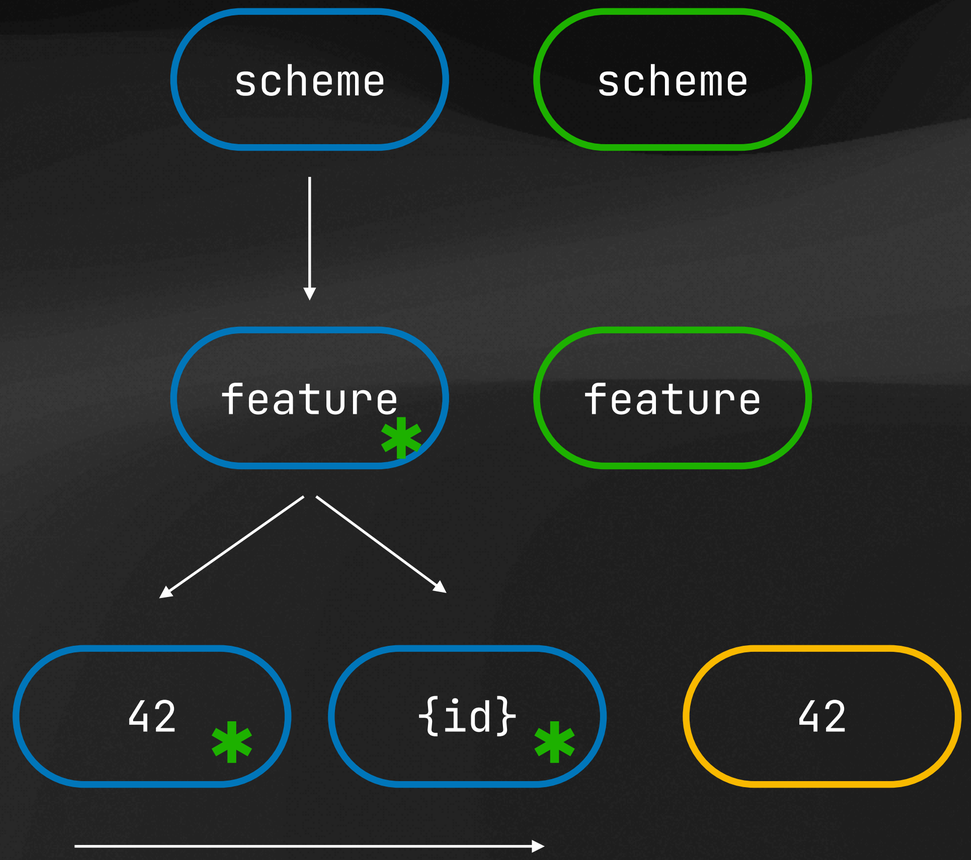
Сопоставление в дереве

Шаблоны

```
scheme://feature
scheme://feature/42
scheme://feature/{id}
```

Диплинк

```
scheme://feature/42
```



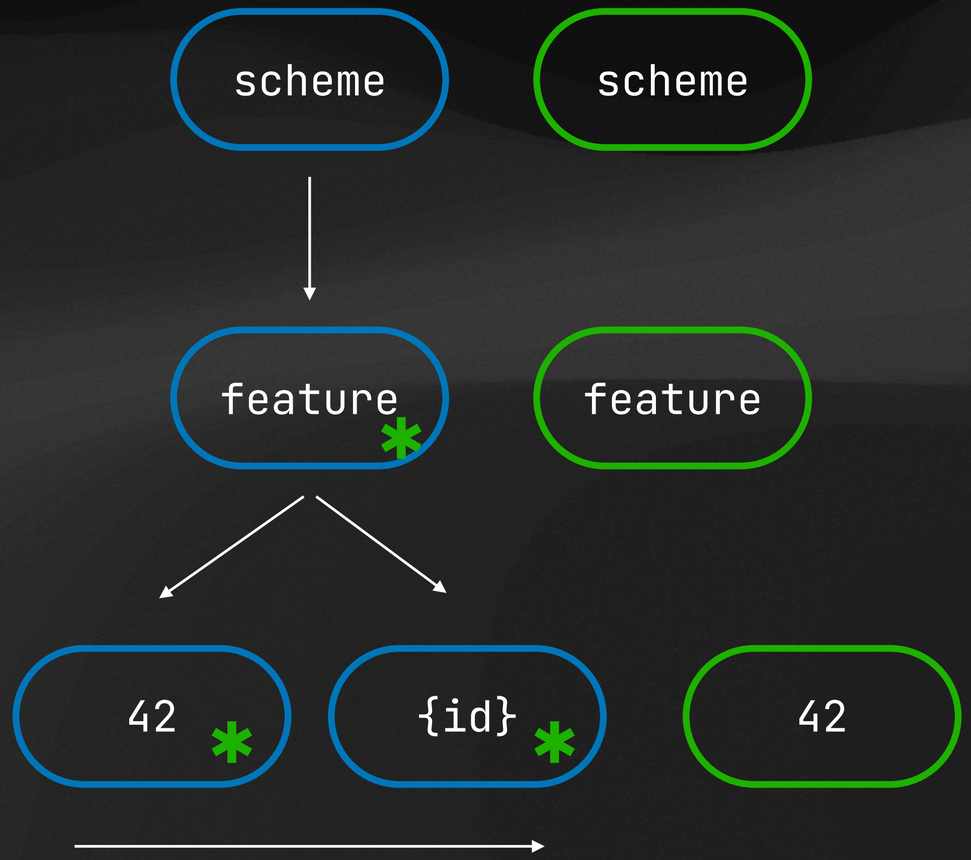
Сопоставление в дереве

Шаблоны

```
scheme://feature
scheme://feature/42
scheme://feature/{id}
```

Диплинк

```
scheme://feature/42
```



Дисклеймер

Проблемы Jetpack Navigation

Что хотим

Обзор Alcubierre

Deeplinks

Итог

Итоги

- Решили наши боли, мигрируем фичи
- Думаем над Compose Render'ом
- Библиотека на GitHub: https://github.com/octa-one/**Alcubierre**

Спасибо!

https://github.com/octa-one/**Alcubierre**

Макушев Александр

Митропольский Александр