Бескомпромиссный SDUI

Кирилл Володин

Руководитель отдела

Александр Евтухов

Архитектор

План доклада

- Появившийся вызов
- Исследование альтернатив
- Верхнеуровневая архитектура
- Релизы фичей
- К чему надо быть готовым….

План доклада

- **Появившийся вызов**
- Исследование альтернатив
- Верхнеуровневая архитектура
- Релизы фичей
- К чему надо быть готовым....

Появившийся вызов

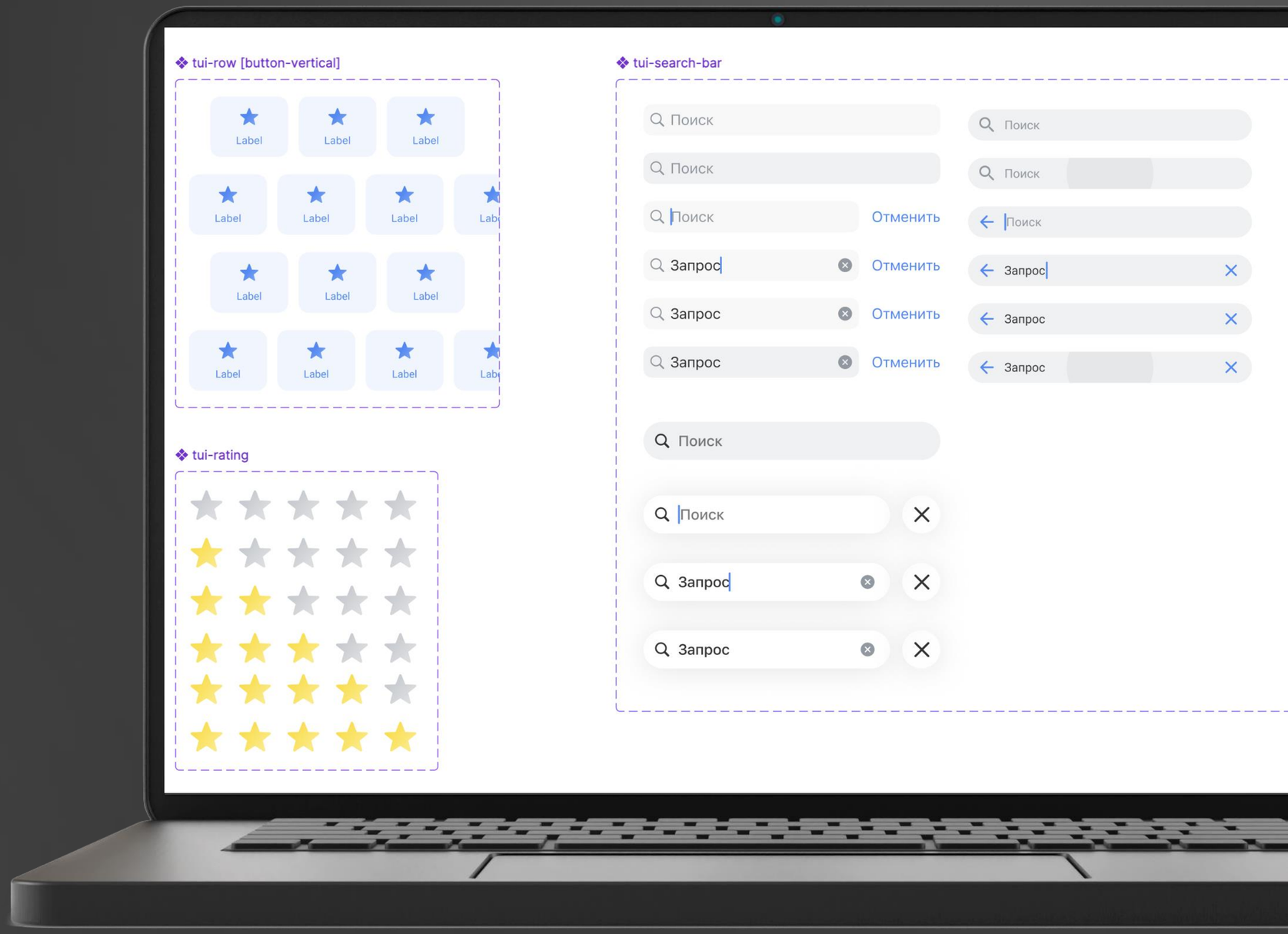
Ставка на SDUI

Разработчики Т-Банк

Gif: tenor.com
Фото: meme-arsenal

Требования и ограничения

Использование нашей дизайн системы



Бизнес логика

- Запросы в сеть, использование кеша, фича тоглов
- Использование геопозиции, камеры, локализации
- Многопоточность

```
val User.formattedName: String
    get() {
        return if (lastName != null) {
            if (firstName != null) {
                "$firstName $lastName"
            } else {
                lastName ?: "Unknown"
            }
        } else {
            firstName ?: "Unknown"
        }
    }

object Repository {

    private val _users = mutableListOf(User("Jane", ""), User("Joh
    val users: List<User>
        get() = _users

    val formattedUserNames: List<String>
        get() = _users.map { user -> user.formattedName }
}
```

Отсутствие лишних лоадеров

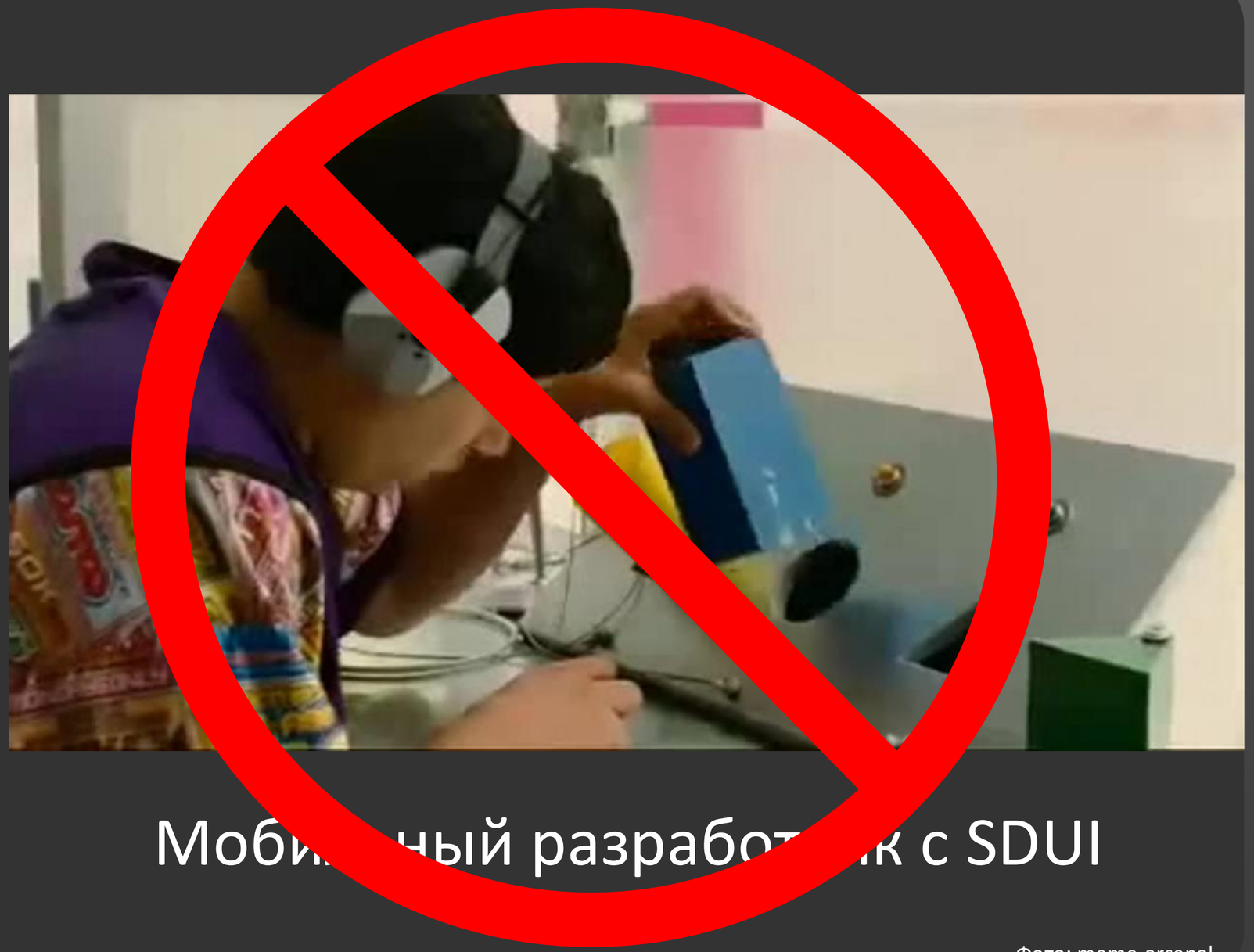
Фото: meme-arsenal

Нативный опыт разработки

Мобильный разработчик с SDUI

Фото: meme-arsenal

Нативный опыт разработки



Мобильный разработчик с SDUI

Фото: meme-arsenal

Фото: ai-generated

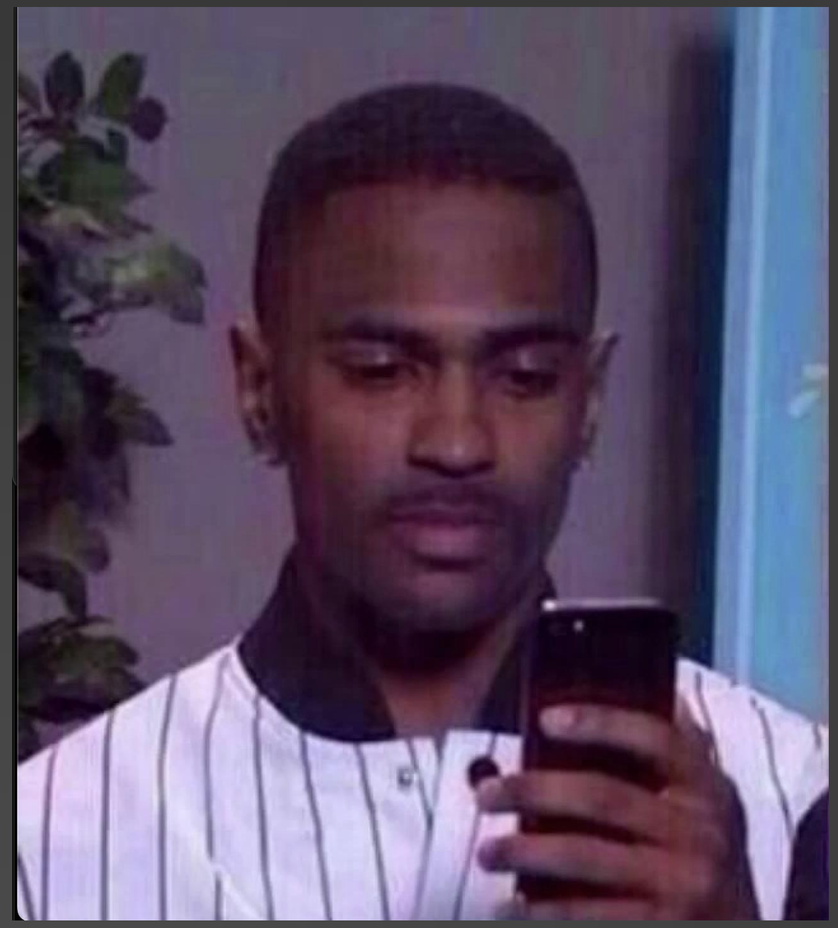
Кросс платформа

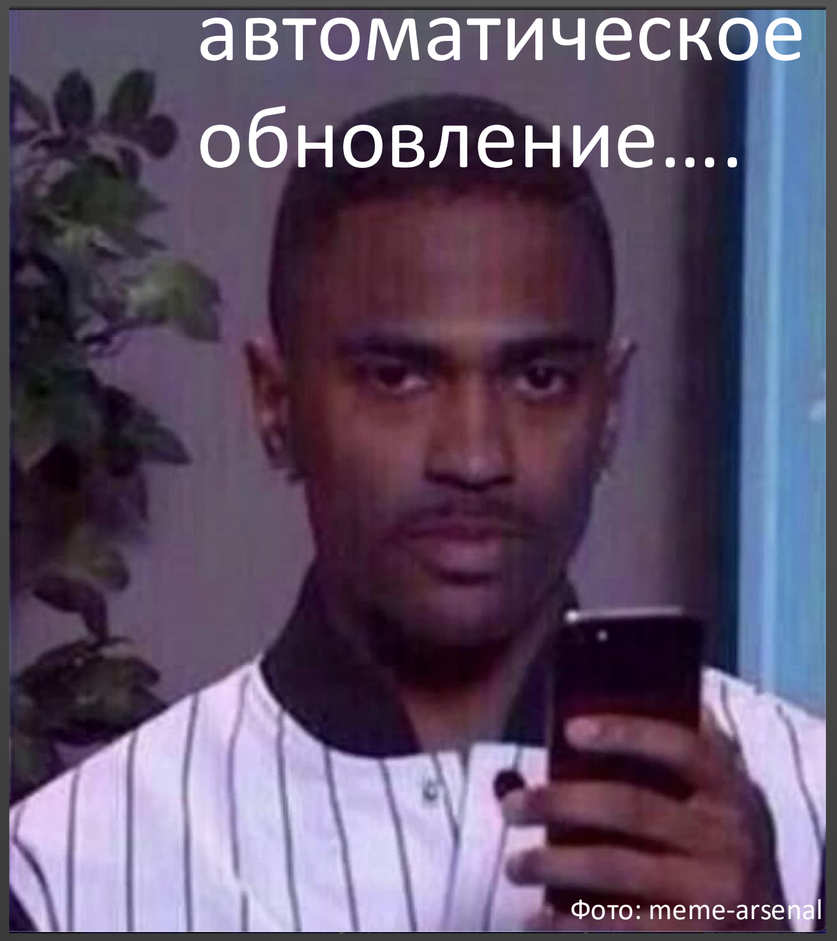
Фото: bitrock.it

Быстрая миграция существующих фичей

Gif: tenor.com

Катить изменения в старые релизы





Gif: tenor.com

Ага поверили, ну и бредятина….

План доклада

- Появившийся вызов
- **Исследование альтернатив**
- Верхнеуровневая архитектура
- Релизы фичей
- К чему надо быть готовым....

Изучение существующих решений и подходов

Критерий
Переиспользовать дизайн систему
Бизнес логика на любой случай жизни
Нет лишних состояний загрузок
Mobile DevEx (Kotlin или Swift)
Кроссплатформа
Дешевая миграция фич
Катиться на старые годовалые версии

DivKit

Критерий	DivKit
Переиспользовать дизайн систему	+−
Бизнес логика на любой случай жизни	−
Нет лишних состояний загрузок	+
Mobile DevEx (Kotlin или Swift)	+−
Кроссплатформа	+
Дешевая миграция фич	−
Катиться на старые годовалые версии	+

Fusion/Beduin и подобные решения

Критерий	DivKit	Fusion/Beduin
Переиспользовать дизайн систему	+−	+
Бизнес логика на любой случай жизни	−	−
Нет лишних состояний загрузок	+	+
Mobile DevEx (Kotlin или Swift)	+−	+−
Кроссплатформа	+	+
Дешевая миграция фич	−	−
Катиться на старые годовалые версии	+	+

Решения опирающиеся на бэкенд

Критерий	DivKit	Fusion/Beduin	Backend based
Переиспользовать дизайн систему	+−	+	+
Бизнес логика на любой случай жизни	−	−	+−
Нет лишних состояний загрузок	+	+	−
Mobile DevEx (Kotlin или Swift)	+−	+−	−
Кроссплатформа	+	+	+
Дешевая миграция фич	−	−	−
Катиться на старые годовалые версии	+	+	+

React Native

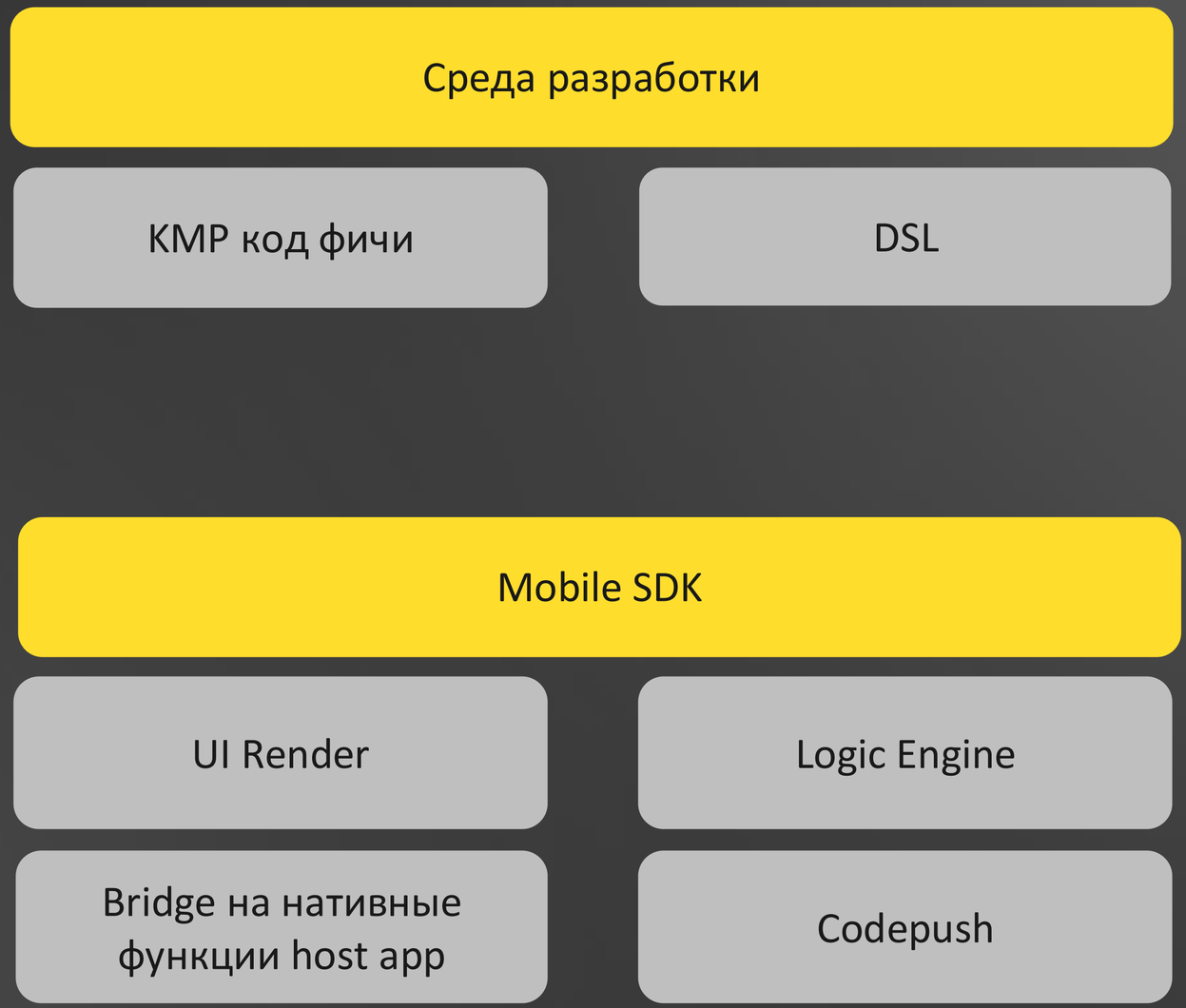
Критерий	DivKit	Fusion/Beduin	Backend based	React Native
Переиспользовать дизайн систему	+−	+	+	+−
Бизнес логика на любой случай жизни	−	−	+−	+
Нет лишних состояний загрузок	+	+	−	+
Mobile DevEx (Kotlin или Swift)	+−	+−	−	−
Кроссплатформа	+	+	+	+
Дешевая миграция фич	−	−	−	−
Катиться на старые годовалые версии	+	+	+	+−

Делаем свою вундервафлю!

План доклада

- Появившийся вызов
- Исследование альтернатив
- **Верхнеуровневая архитектура**
- Релизы фичей
- К чему надо быть готовым....

Экосистема Mystic



Экосистема Mystic
Дополнительные компоненты

Среда разработки

KSP генератор для DSL

KMP код фичи

DSL

Gradle Plugin-ы для сборки релизов

Библиотека контрактов

Инфраструктура для поставки

Backend для управления in-out контрактами экранов

Mobile SDK

Админка для ведения контрактов

Custom Bridge Контракты

UI Render

Logic Engine

Админка для LowCode сборки экранов

Bridge на нативные функции host app

Codepush

Компоненты Render Engine

Контейнеры

Позволяют располагать элементы друг относительно друга.

Элементы дизайн системы

Готовые элементы дизайн системы подключаются в SDUI для получения максимально native-like experience

State Bindings

Для привязки параметров UI элементов к состоянию экрана. Позволяют делать преобразования данных.

Слой низко-уровневых атомов и атрибутов для верстки

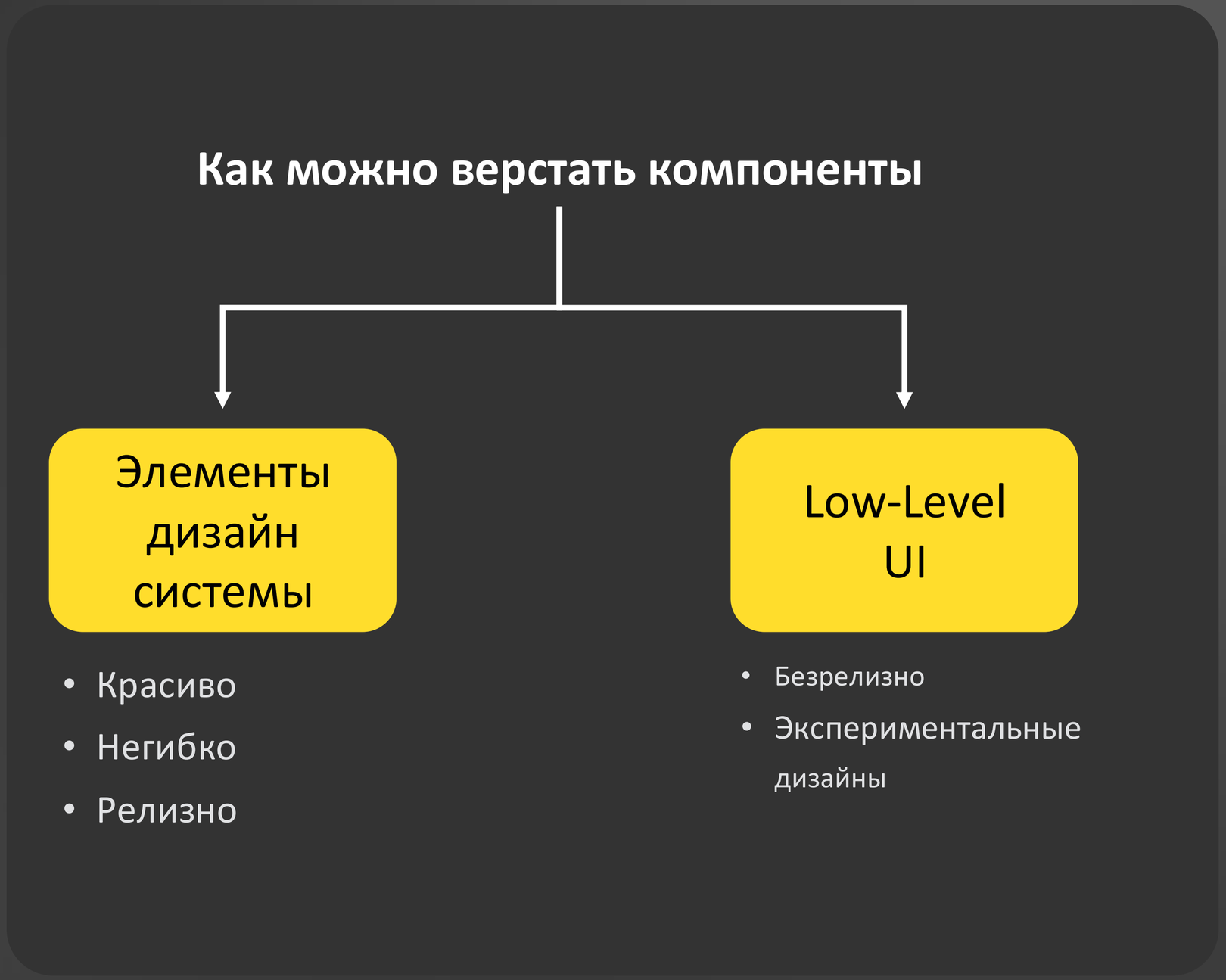
Для экспериментальных UI элемементов или дизайнов отклоняющихся от дизайн системы.

Сравнение подходов к построению SDUI

Критерий	SDUI на базе дизайн системы	SDUI на базе низко-уровневых примитивов
Переиспользовать дизайн систему	+	−
Сложные анимации \ acessability	+	−
Скорость разработки SDUI решения	+	−
Стоимость поддержки	+	−
Возможно безрелизно добавить \ изменить компонент	−	+
Необходимость ждать раскатки нового компонента на всех пользователей	−	+

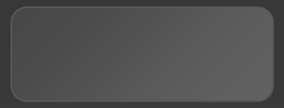
Render Engine

Наш SDUI берет лучшее от подходов с использованием компонентов дизайн системы и версткой из атомов



Low – Level - UI

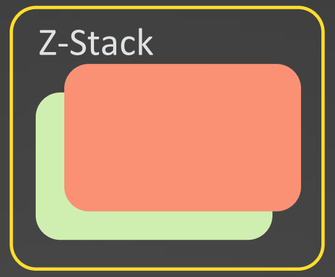
Rectangle



Shadow

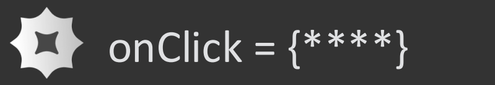
Кнопка Компонент Text

Вызов Action по нажатию



Контейнер Z-Stack

Собираем новую кнопку из элементов LLUI



Новый элемент дизайн системы

Low – Level - UI

Разработчику не нужно делать отдельных версий экрана для разных версий SDK

Код разработчика

```
VStack {
    YellowButton(***)
}
```

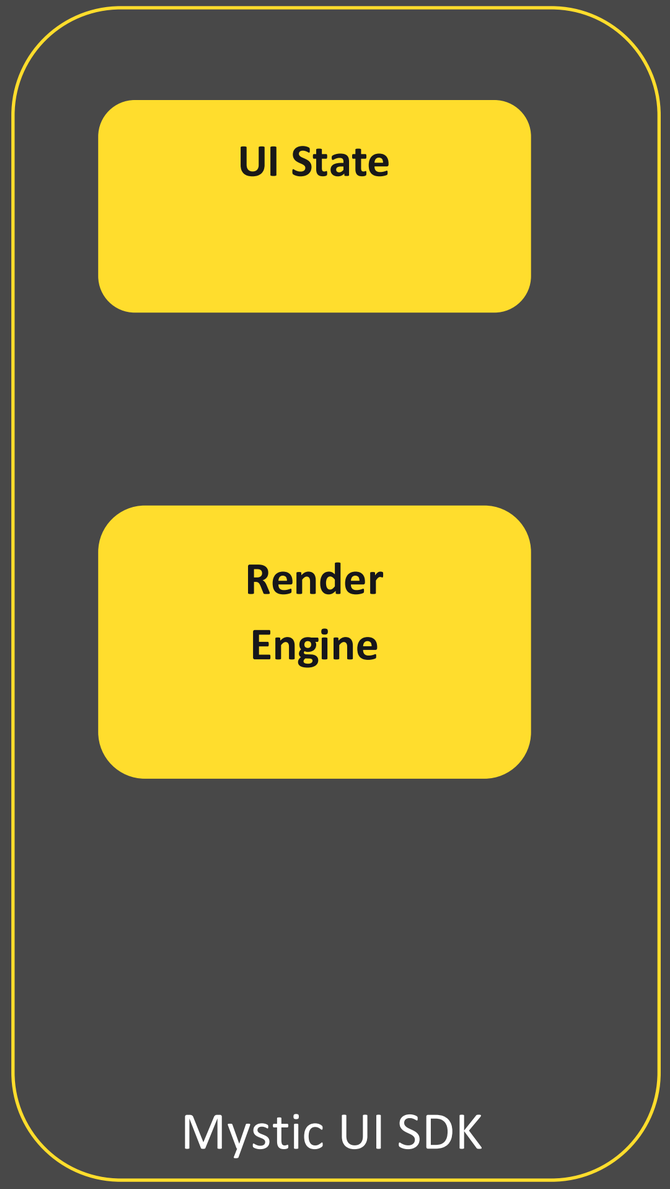
Кнопка

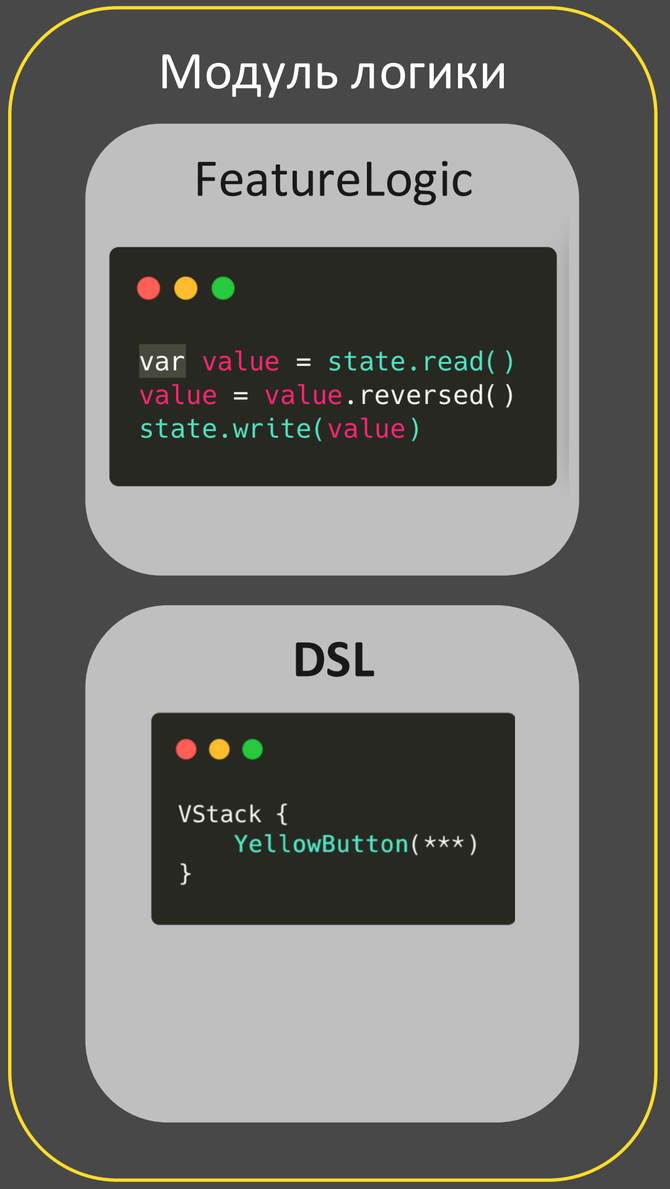
Старая версия МП – бекпорт на LLUI

Кнопка

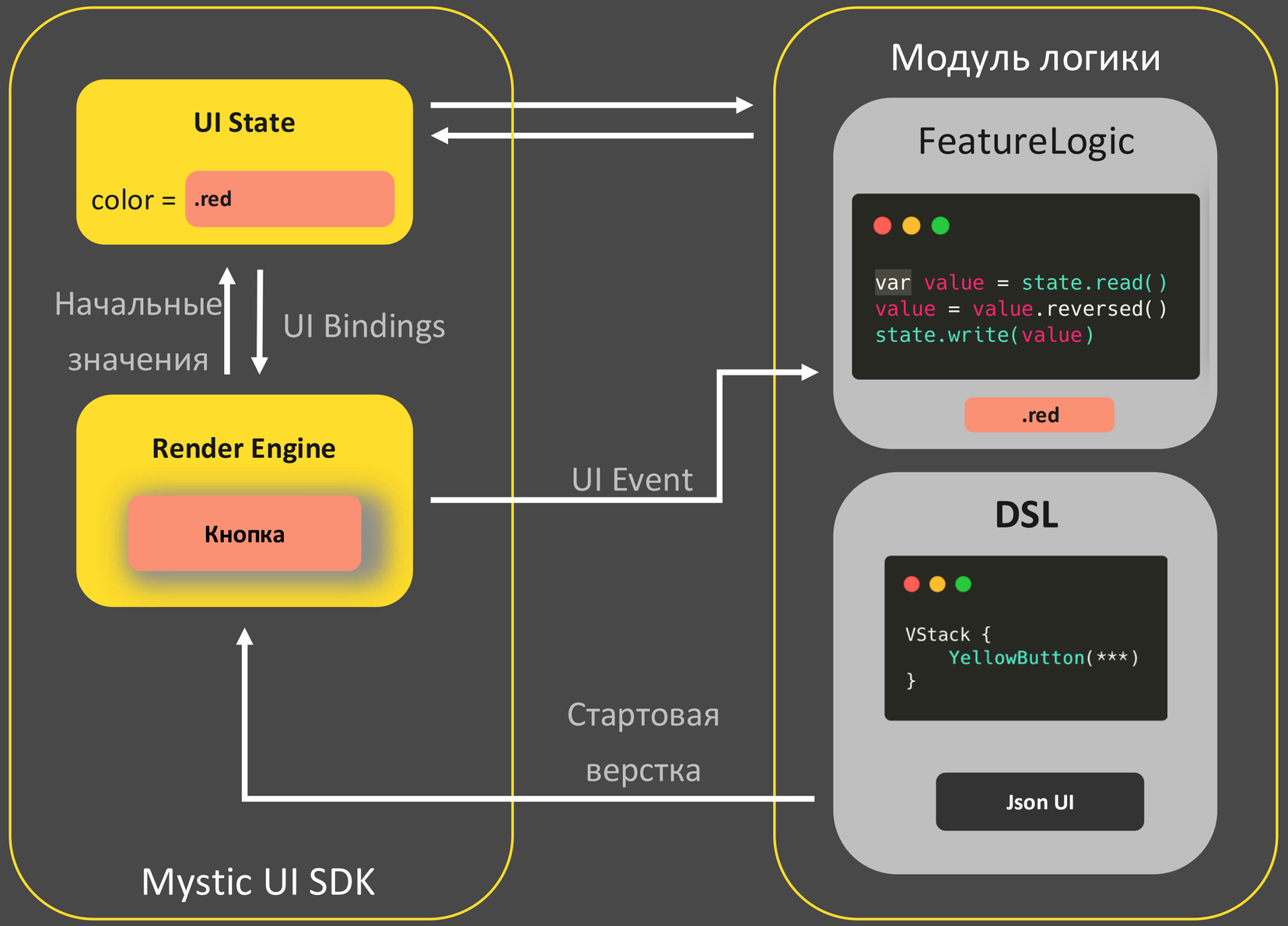
Новая версия МП – элемент дизайн системы

Жизненный цикл экрана

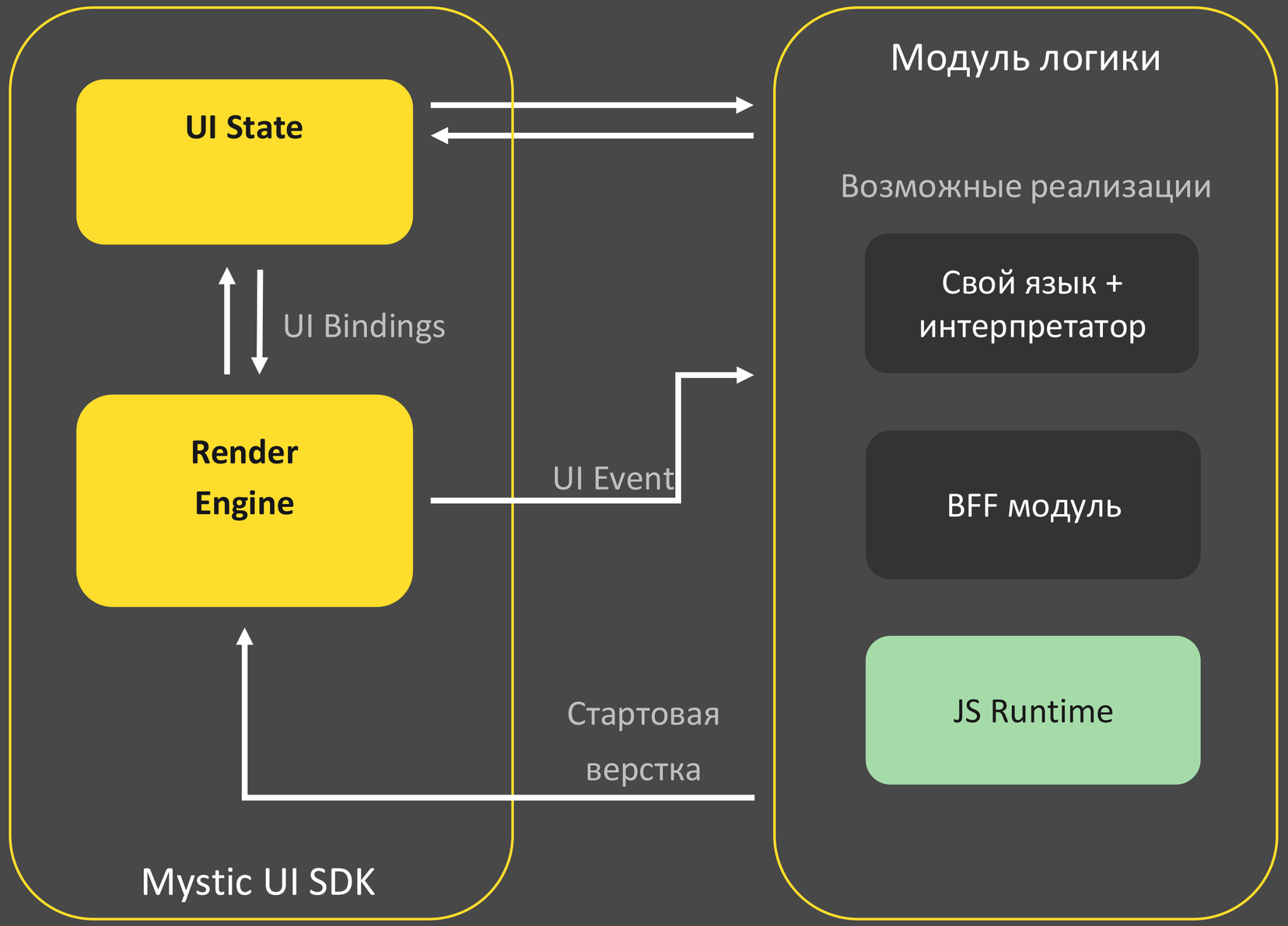




Жизненный цикл экрана



Жизненный цикл экрана

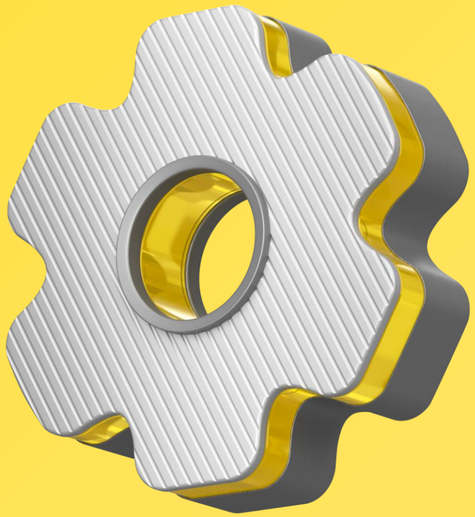


Ну что? Пишем на JS?

Gif: tenor.com

KotlinJS

Технология компиляции Kotlin в JS от компании JetBrains



Native-like dev exp

Kotlin привычен для разработчиков. iOS комьюнити легко переучивается.

Портирование фрагментов МБ

За счет Kotlin можно портировать большие фрагменты кода из основных приложений, особенно если они написаны под KMP.

Позволяет писать очень сложную логику

Наличие корутин и большого кол-ва библиотек позволяет писать логику любой сложности. Нет проблемы нереализованности каких-то компонентов.

Готовый тулинг для сборки и запуска

Готовые инструменты для сборки, отладки, минификации кода. Готовые среды запуска.

Приходите на доклад завтра

В 11-00 Тимур и Максим все расскажут

Доклад

Как подружить мобильное приложение с JS и прокачать свой SDUI

Расскажем, как в Server Driven UI добавить безрелизную бизнес-логику с помощью Kotlin Multiplatform.

Максим Вакула
Т-Банк

Тимур Валиев
Т-Банк

Architecture & Infrastructure | UX/UI & Animations

Logic Engine

Что может использовать разработчик при разработке экрана

KotlinJS

Основная бизнес логика экрана

SDUI Script

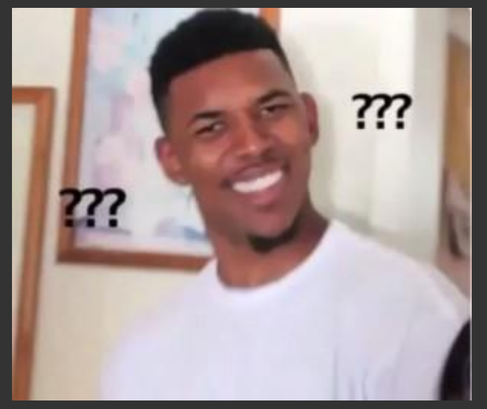
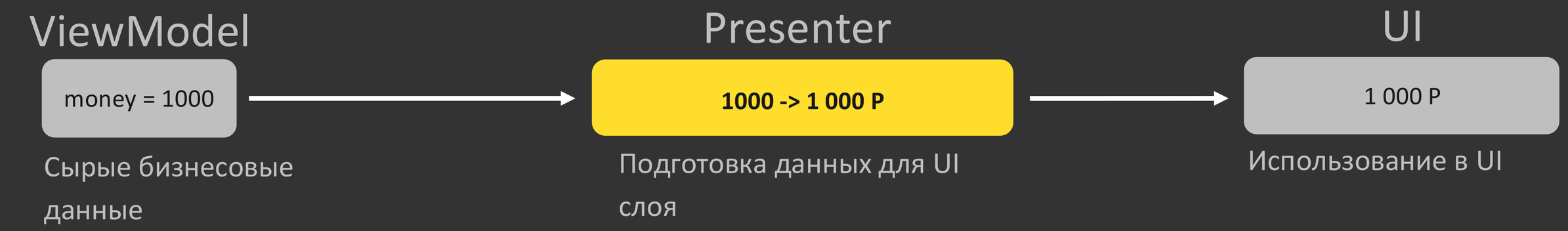
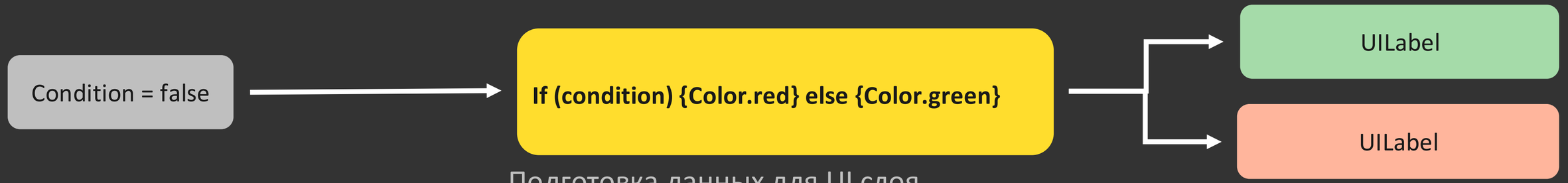


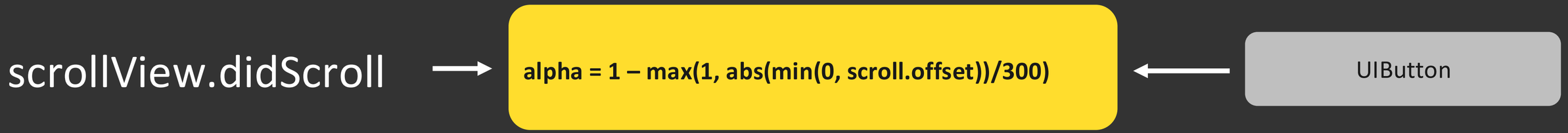
Фото: meme-arsenal

Презентационная логика



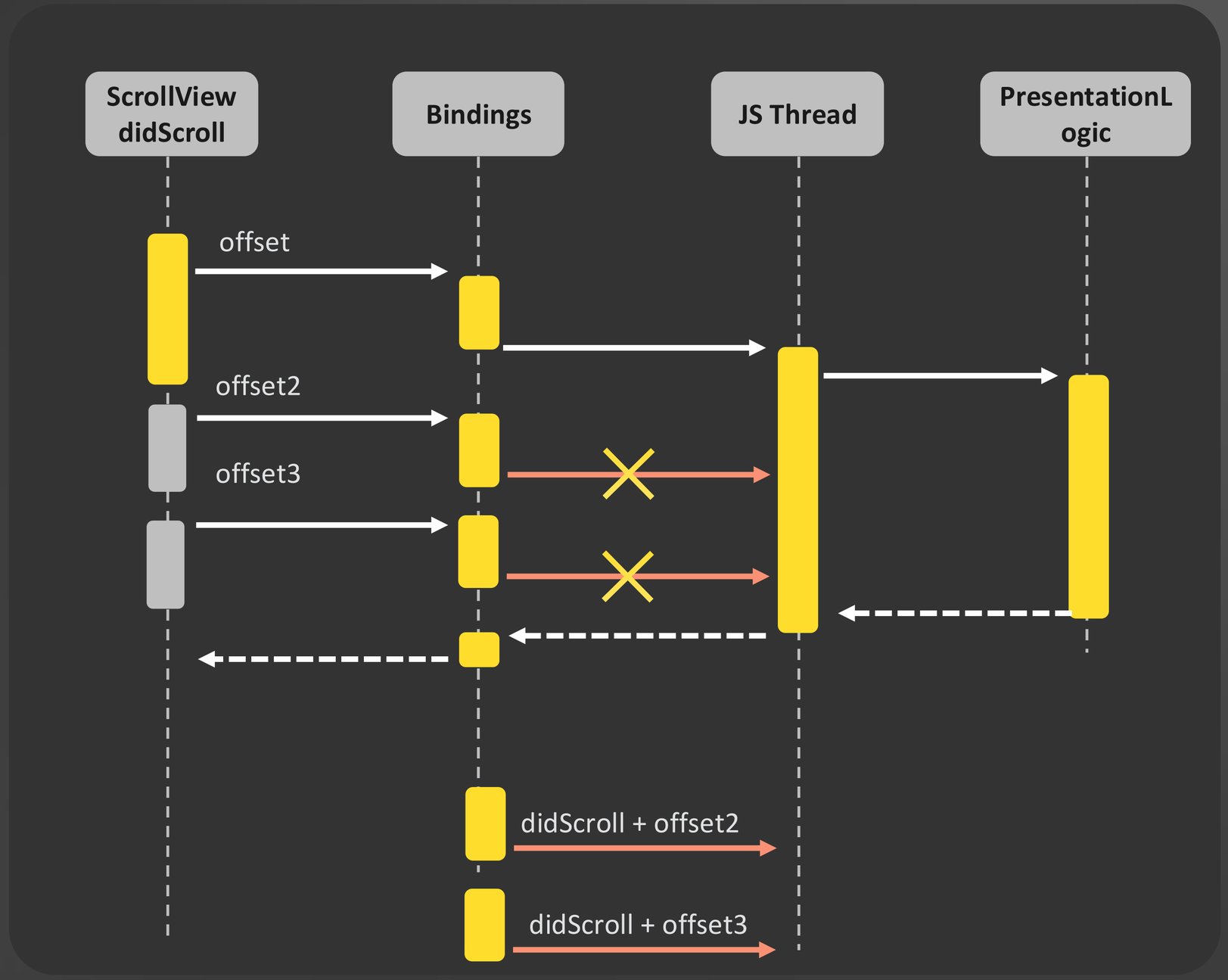


UI логика

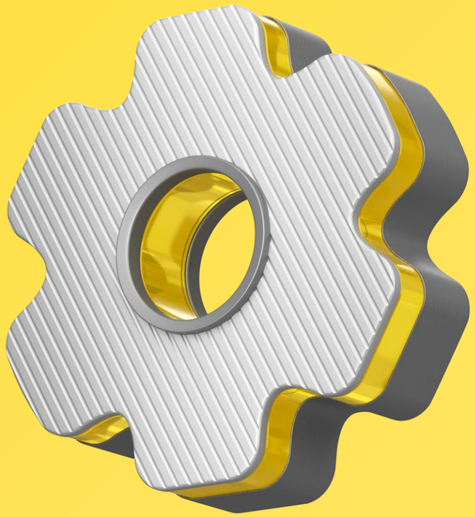


Перегрузка бриджа

Если перегрузить бридж событиями то возникнет эффект glitch на экране



SDUI Script



Скорость работы

Работает быстрее JS позволяя реализовывать сложные события анимаций и обработки скролла

Сериализуем

Можно вернуть вместе с верстской в виде JSON. Используется в LowCode решении.

Базовый набор операторов

Покрывает все базовые вычисления, которые нужны для связывания UI со State

Простые ветвления логики

Возможность реализовать часть презентационной логики в нативе снижая нагрузку на бридж.

Logic Engine

Что может использовать разработчик при разработке экрана

KotlinJS

Основная логика экрана

- сложная
- native-like
- медленная

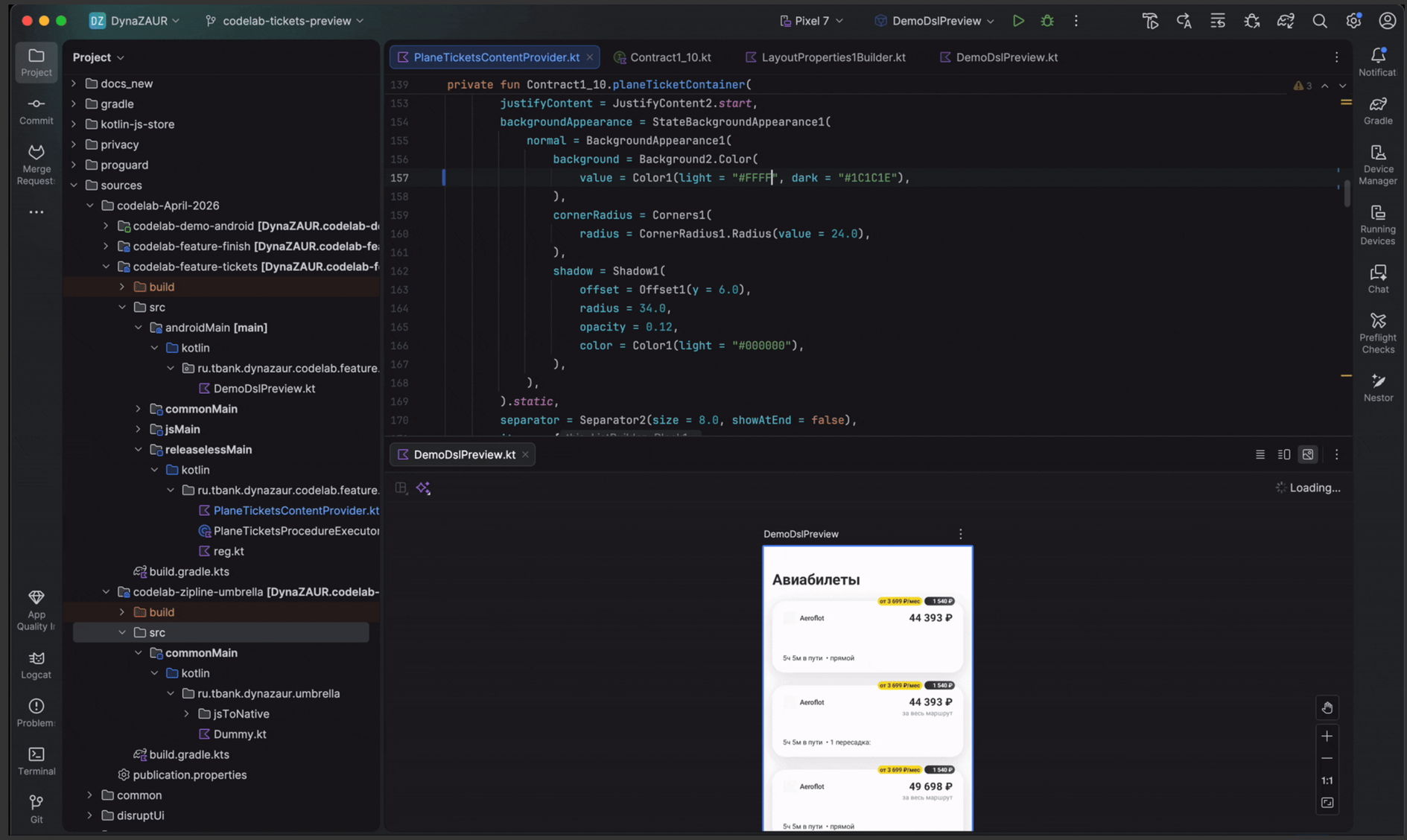
SDUI Script

Логика UI слоя

- сложные биндинги
- простые преобразования
- анимации\события скролла

SDUI проект

- Привычный KMP проект
- Не JSON Based разработка



DSL

- Управление контейнерами и элементами

```
private fun Contract1_10.headerContainer(
    screenData: PlaneTicketsRecyclerItemDslStateAdapter.Header,
    screenApi: PlaneTicketsScreenApiAdapter
): Stack1 {
    return stack1(
        layoutProperties = layoutProperties1(
            width = Size1.matchParent,
            height = Size1.wrapContent,
            margins = Inset1(
                horizontal = 16.0,
                bottom = 8.0
            )
        ),
        direction = Direction1.column,
        alignItems = Alignment1.start,
        justifyContent = JustifyContent2.start,
        items = { this: ListBuilder<Block1>
            +headerText(screenData.text)
            +filterText(screenData, screenApi)
        }
    )
}
```

DSL

- Управление контейнерами и элементами
- Настройка атрибутов

```
private fun Contract1_10.headerContainer(
    screenData: PlaneTicketsRecyclerItemDslStateAdapter.Header,
    screenApi: PlaneTicketsScreenApiAdapter
): Stack1 {
    return stack1(
        layoutProperties = layoutProperties1(
            width = Size1.matchParent,
            height = Size1.wrapContent,
            margins = Inset1(
                horizontal = 16.0,
                bottom = 8.0
            )
        ),
        direction = Direction1.column,
        alignItems = Alignment1.start,
        justifyContent = JustifyContent2.start,
        items = { this: ListBuilder<Block1>
            +headerText(screenData.text)
            +filterText(screenData, screenApi)
        }
    )
}
```

DSL

- Управление контейнерами и элементами
- Настройка атрибутов
- Биндинг к стейту

```
11 Usages
@Serializable
data class Header(
    override val myItemId: String,
    val text: String,
    val isFiltered: Boolean
) : PlaneTicketsRecyclerItem
```

```
private fun Contract1_10.headerContainer(
    screenData: PlaneTicketsRecyclerItemDslStateAdapter.Header,
    screenApi: PlaneTicketsScreenApiAdapter
): Stack1 {
    return stack1(
        layoutProperties = layoutProperties1(
            width = Size1.matchParent,
            height = Size1.wrapContent,
            margins = Inset1(
                horizontal = 16.0,
                bottom = 8.0
            )
        ),
        direction = Direction1.column,
        alignItems = Alignment1.start,
        justifyContent = JustifyContent2.start,
        items = { this: ListBuilder<Block1>
            +headerText(screenData.text)
            +filterText(screenData, screenApi)
        }
    )
}
```

DSL

- Строгая типизация вызова бизнес логики

```
7 Usages  2 Implementations
@DslScreenApi(mode = ScreenApiMode.CALLBACK)
interface PlaneTicketsScreenApi {

    3 Usages  1 Implementation
    fun filterTicket()

    3 Usages  1 Implementation
    fun navigate()
}
```

```
private fun Contract1_10.filterText(
    screenApi: PlaneTicketsScreenApiAdapter
): Stack1 {
    return stack1(
        layoutProperties = layoutProperties1(
            width = Size1.wrapContent,
            height = Size1.wrapContent,
        ),
        padding = Inset1(horizontal = 8.0, vertical = 4.0),
        direction = Direction1.column,
        alignItems = Alignment1.start,
        justifyContent = JustifyContent2.start,
        onClick = screenApi::filterTicket,
        backgroundAppearance = ifLet(
            condition = screenData.isFiltered,
            ifFalse = StateBackgroundAppearance1(
                normal = BackgroundAppearance1(
                    background = Background2.Color(
                        value = Color1(light = "#F7F8F8", dark = "#F7F8F8"),
                    ),
                    cornerRadius = Corners1(
                        radius = CornerRadius1.Radius(value = 24.0),
                    ),
```

DSL

- Строгая типизация вызова бизнес логики
- Обработка на KMP

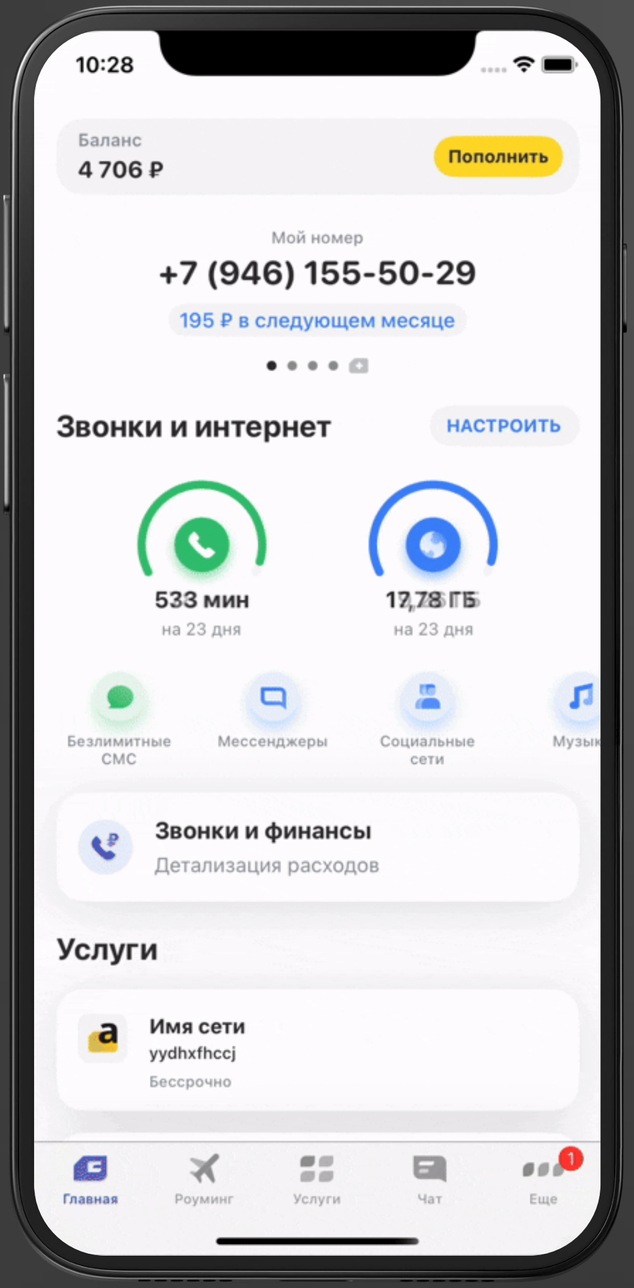
```
7 Usages  2 Implementations
@DslScreenApi(mode = ScreenApiMode.CALLBACK)
interface PlaneTicketsScreenApi {

    3 Usages  1 Implementation
    fun filterTicket()

    3 Usages  1 Implementation
    fun navigate()
}
```

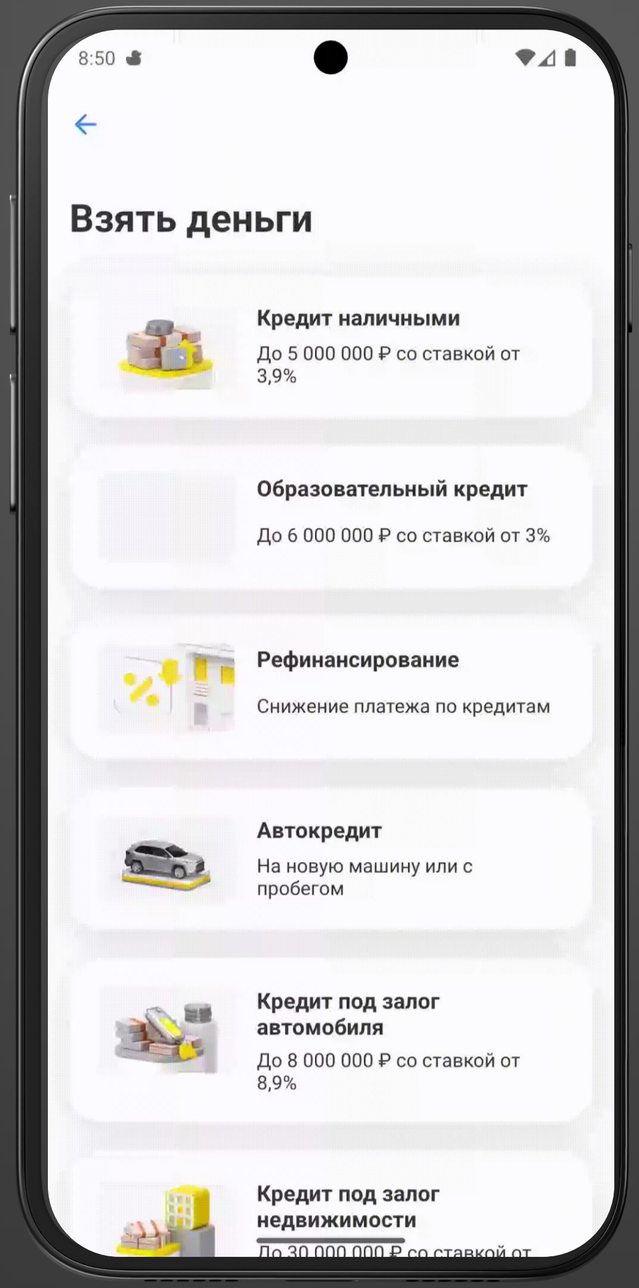
```
internal class PlaneTicketsProcedureExecutorImpl(
    override val dataStateConnection: Lazy<DynaNativeBridgeDataStateConnection>,
) : PlaneTicketsProcedureExecutor() {

    2 Usages
    private val screenState by lazy { PlaneTicketsDataStateDslStateAdapter() }

    override fun filterTicket() {
        val currentItems = screenState.items.readValue()
        val headerItem = currentItems.filterIsInstance<PlaneTicketsRecyclerItem.Header>().first()
        val currentFilterState = headerItem.isFiltered
        val newFilterState = !currentFilterState
        val newTicketsItems = if (newFilterState) {
            recyclerBlockTickets().filter { it.airlineName == "Aeroflot" }
        } else {
            recyclerBlockTickets()
        }
        val newHeaderItem = headerItem.copy(isFiltered = newFilterState)

        screenState.items.setValue(listOf(newHeaderItem) + newTicketsItems)
    }
```

Примеры фичей на базе движка

- Настройка тарифа
  6к+ строк кода

- Предложение кредита
  1к+ строк кода



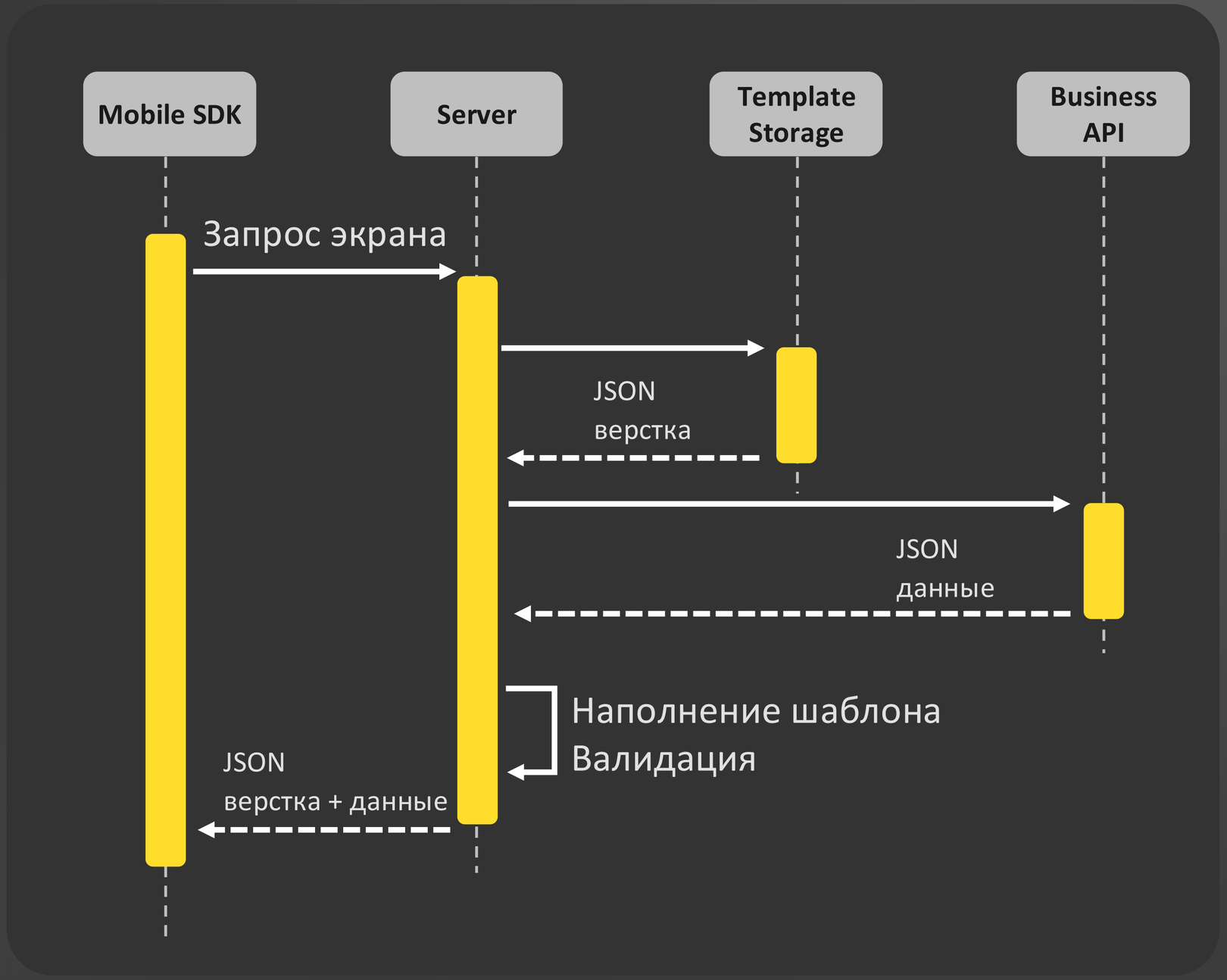


План доклада

- Появившийся вызов
- Исследование альтернатив
- Верхнеуровневая архитектура
- **Релизы фичей**
- К чему надо быть готовым….

Типичная схема SDUI поставки

Весь экран готовится и обогащается данными на беке и поступает в форме JSON на устройство



Native
Look And Feel

Выглядеть нативно

Работать быстрее Web

Старт экрана быстрее Web

Долгие загрузки более 3х секунд дают значительные падения конверсий

Первый старт без скачивания

Фича на SDUI должна быть доступна сразу после скачивания приложения (исключая бизнесовые данные)

Offline режим

Если у пользователя нет интернета – важно уметь показать закешированное состояние приложения.

Фото: meme-arsenal

SDUI CodePush

Схема поставки артефактов

CDN

Хранилище файлов для релиза

Мобильный SDK

Local Cache

Локальный кеш приложения

Embeded Files

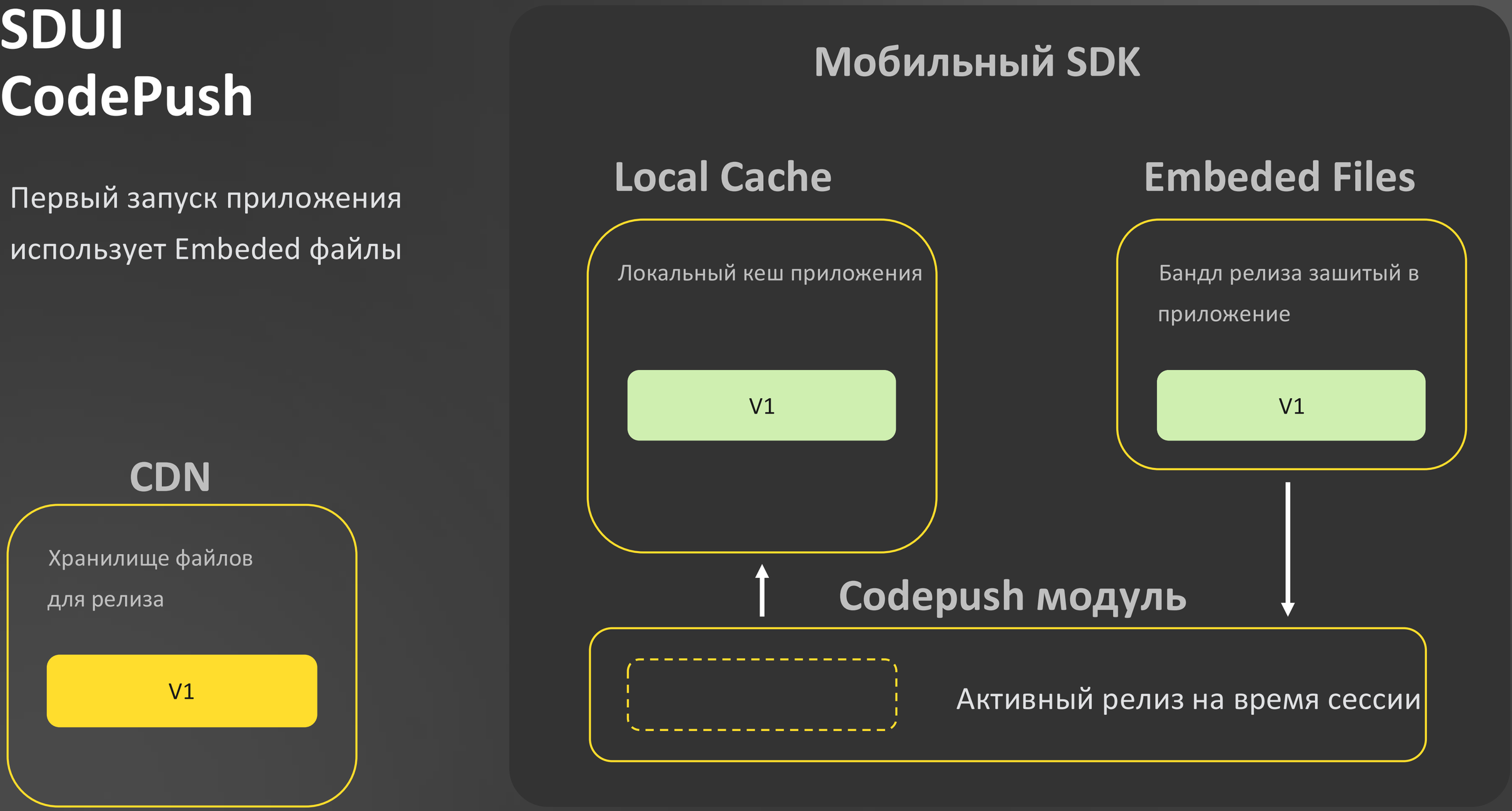
Бандл релиза зашитый в приложение

Codepush модуль

Активный релиз на время сессии

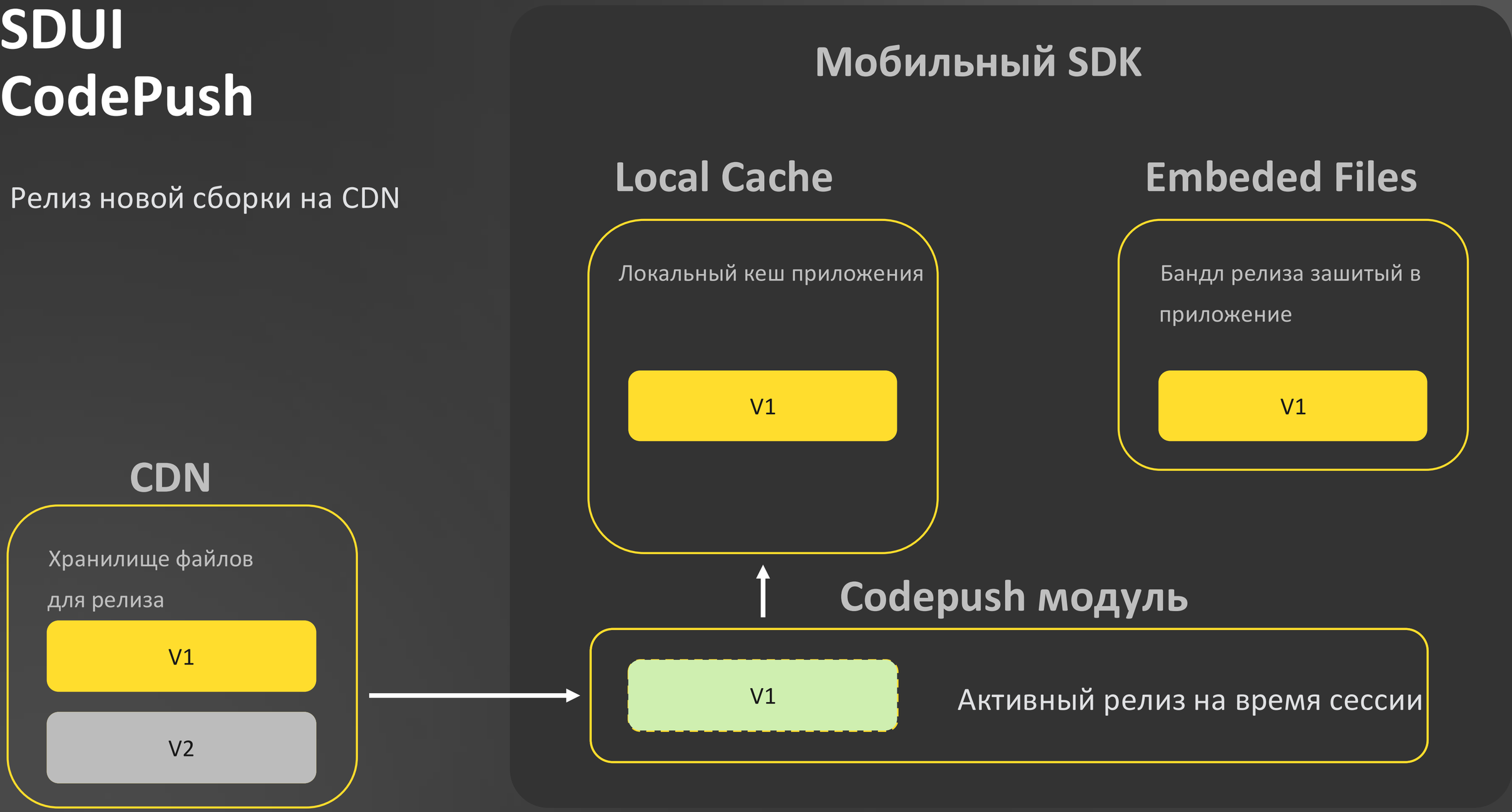
SDUI CodePush

Первый запуск приложения использует Embeded файлы



SDUI CodePush

Релиз новой сборки на CDN



SDUI CodePush

Релиз новой сборки на CDN

CDN

Хранилище файлов для релиза

V1

V2

Мобильный SDK

Local Cache

Локальный кеш приложения

V1

V2

Embeded Files

Бандл релиза зашитый в приложение

V1

Codepush модуль

V1

Активный релиз на время сессии

SDUI CodePush

Релиз новой сборки на CDN

CDN

Хранилище файлов для релиза

V1

V2

Мобильный SDK

Local Cache

Локальный кеш приложения

V1

V2

Embeded Files

Бандл релиза зашитый в приложение

V1

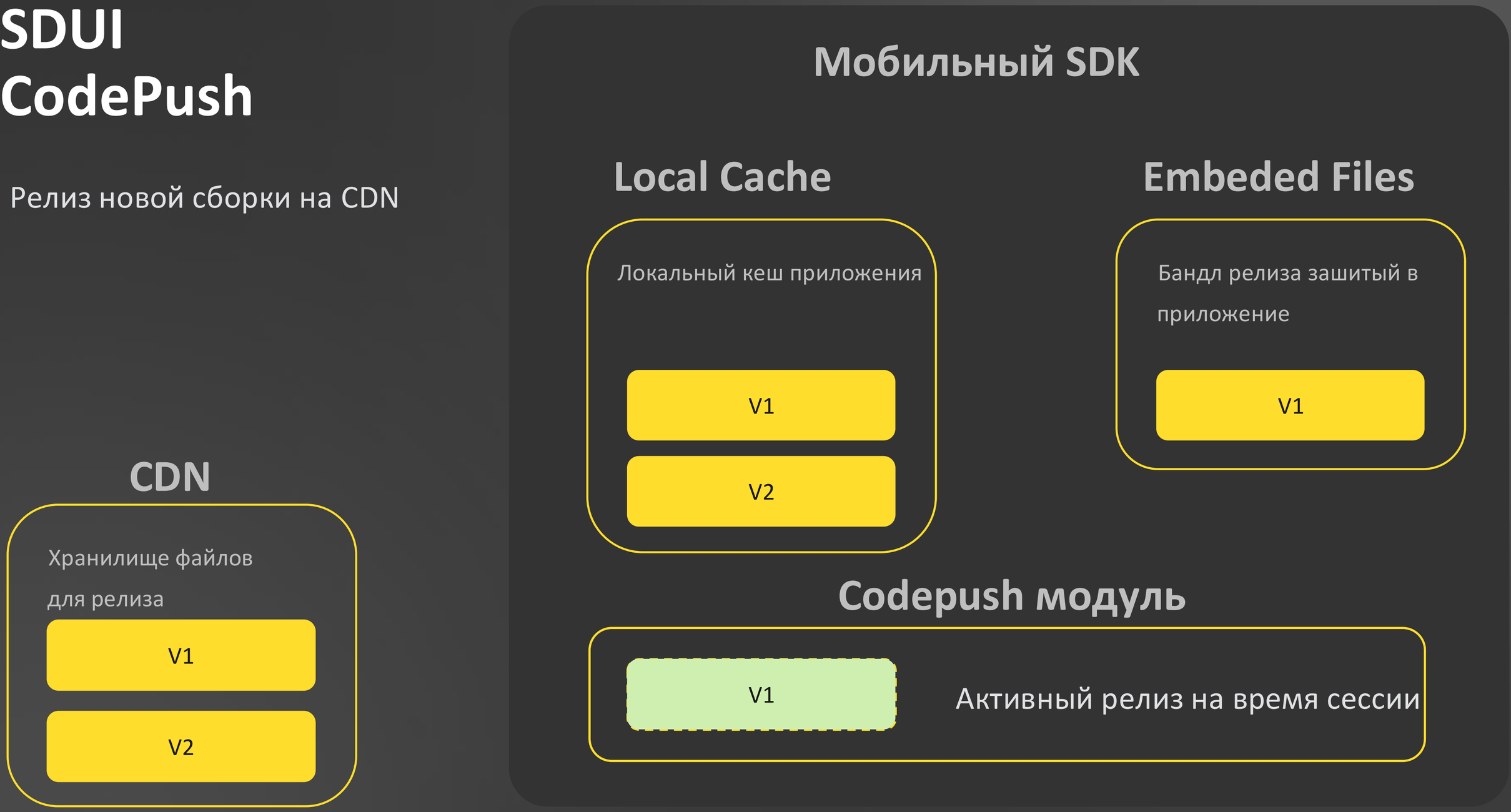
Codepush модуль

V1

Активный релиз на время сессии

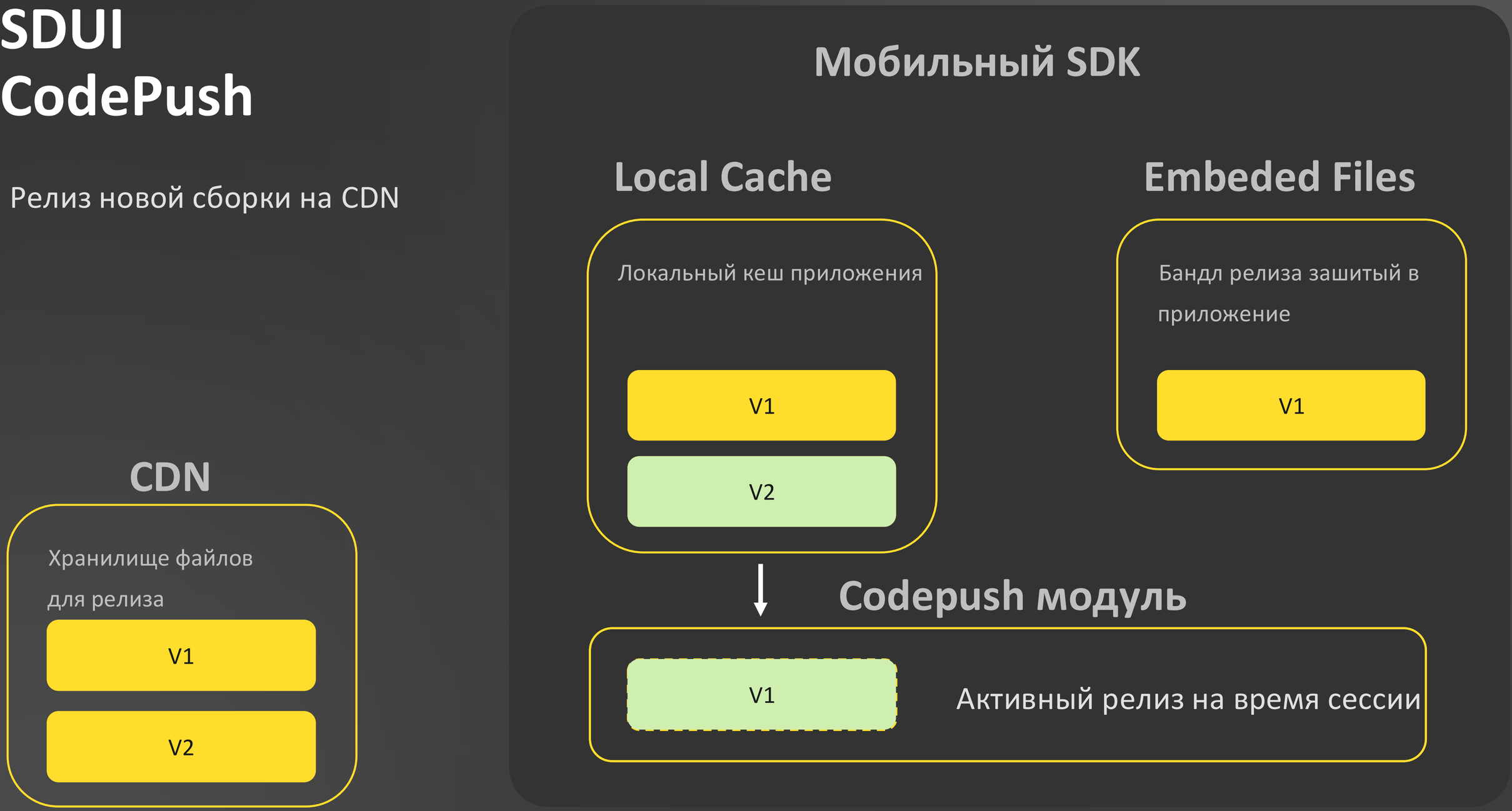
SDUI CodePush

Релиз новой сборки на CDN



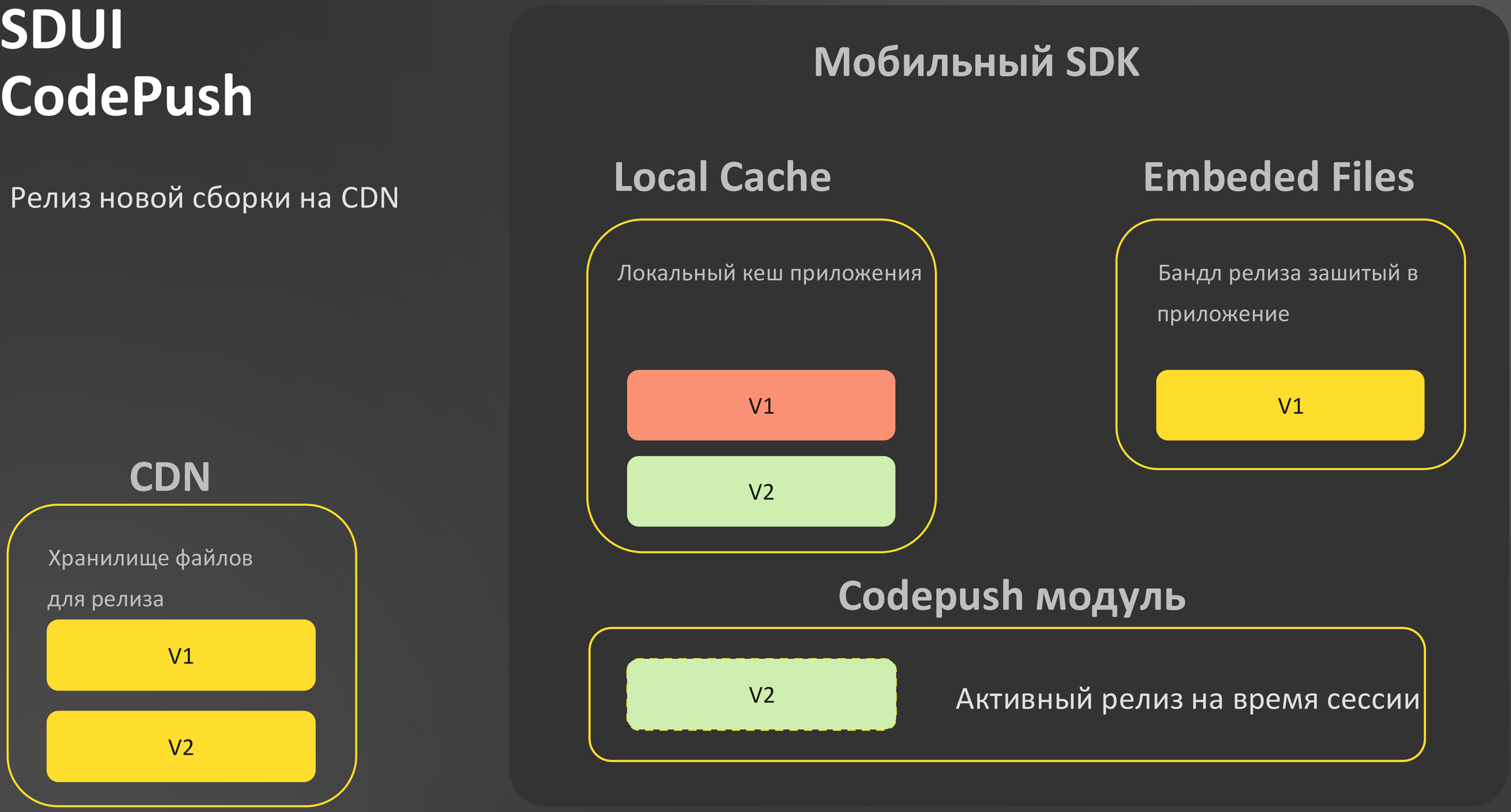
SDUI CodePush

Релиз новой сборки на CDN



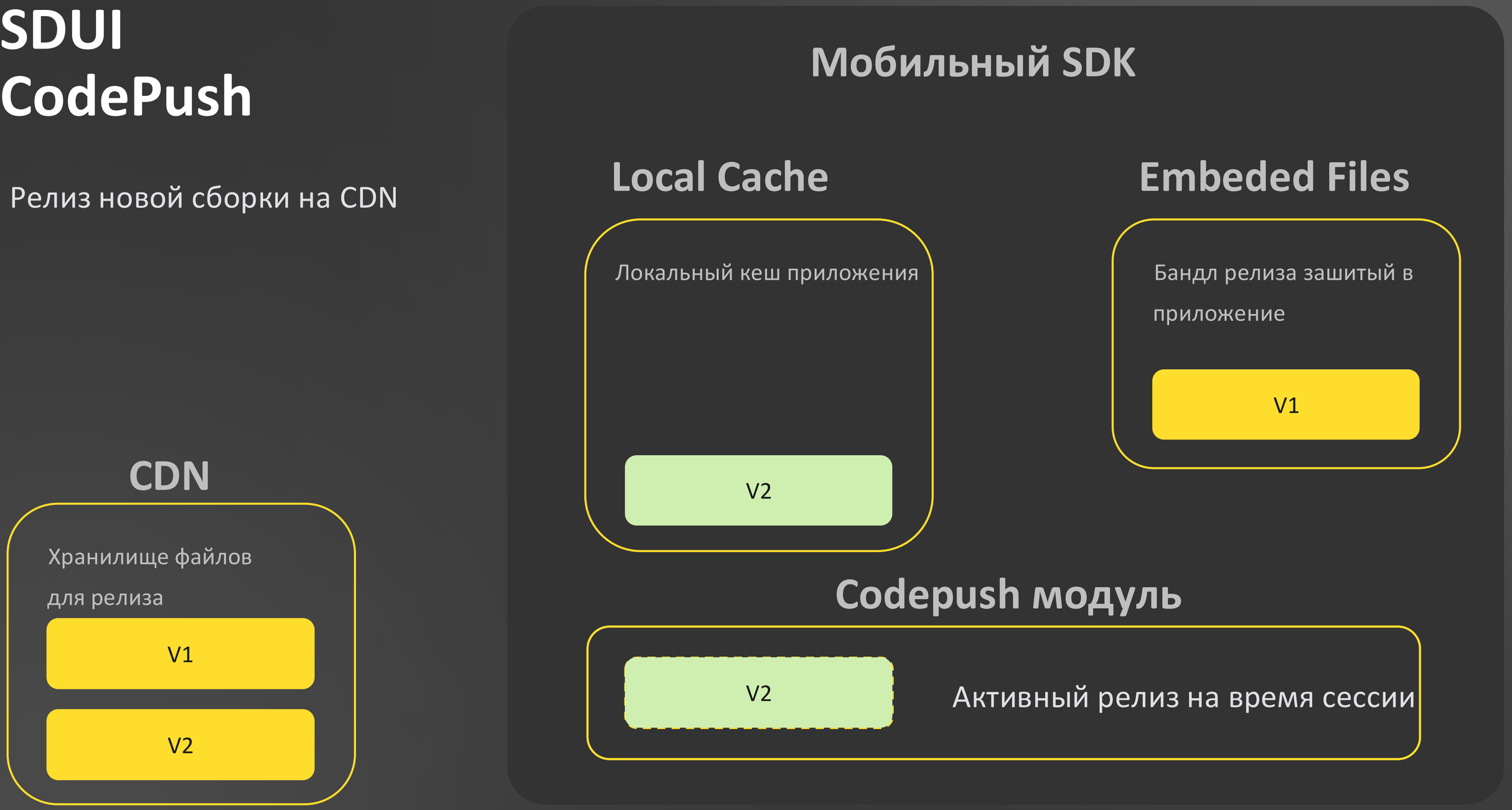
SDUI CodePush

Релиз новой сборки на CDN



SDUI CodePush

Релиз новой сборки на CDN

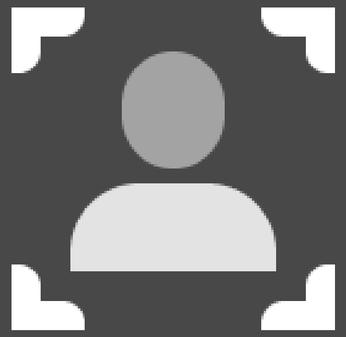


Поведенческая совместимость

В приложении много связанных друг с другом экранов, за совместимостью которых необходимо следить

Фича из 2х экранов

Экран 1



Выберите фото

Экран 2

Введите ФИО

****

****

Поведенческая совместимость

Состояние CDN

https://cdn.ru/**SDUI**

V1

V2

Экран 1

Выберите фото

Экран 2

Введите ФИО

****

****

Экран 1

Выберите фото

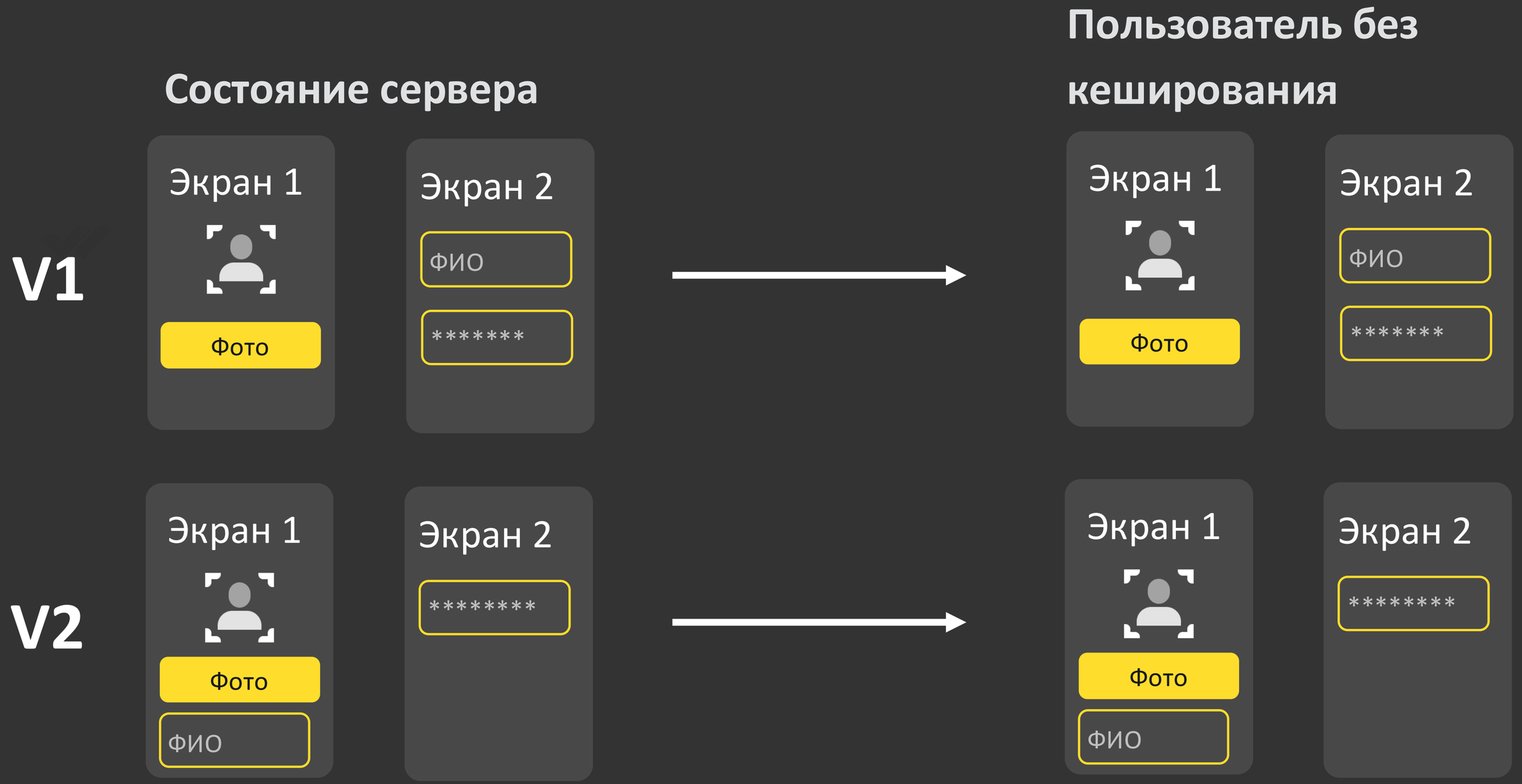
Введите ФИО

Экран 2

****

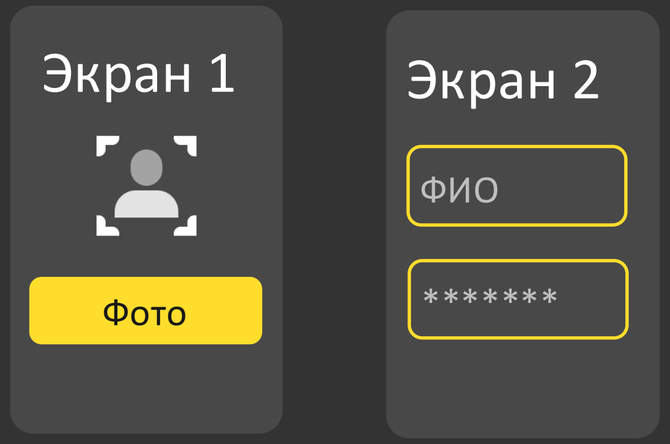
****

Поведенческая совместимость

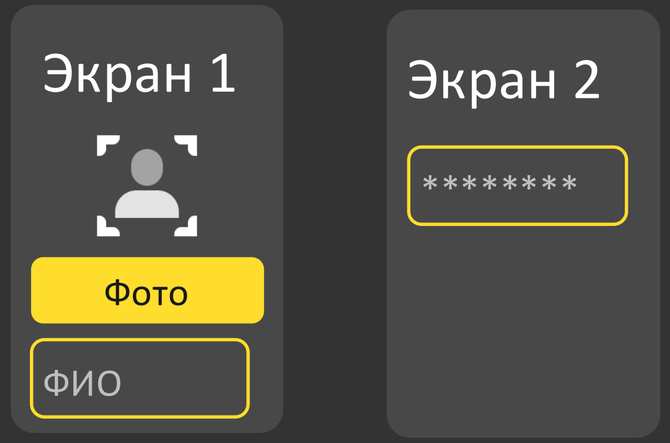


Возможные состояния кеша

V1



V2



Приоритеты данных для Codepush

1. Local cache
2. Embeded файлы
3. OnDemand загрузка из сети

Возможные состояния кеша

Состояние сервера

Что видит клиент

V1

Экран 1

Фото

Экран 2

ФИО

****

V2

Экран 1

Фото

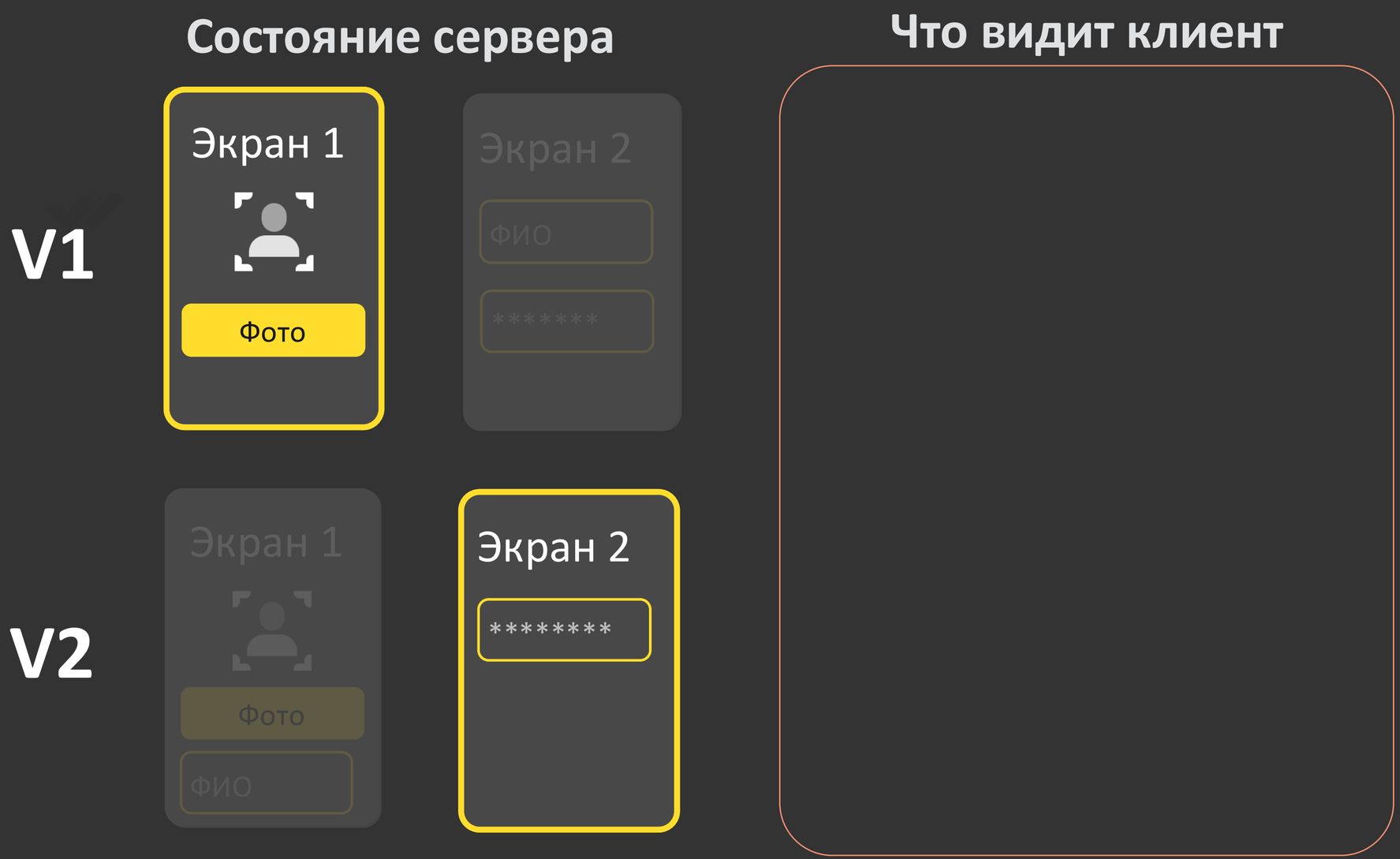
ФИО

Экран 2

****

Первый экран не успел обновиться

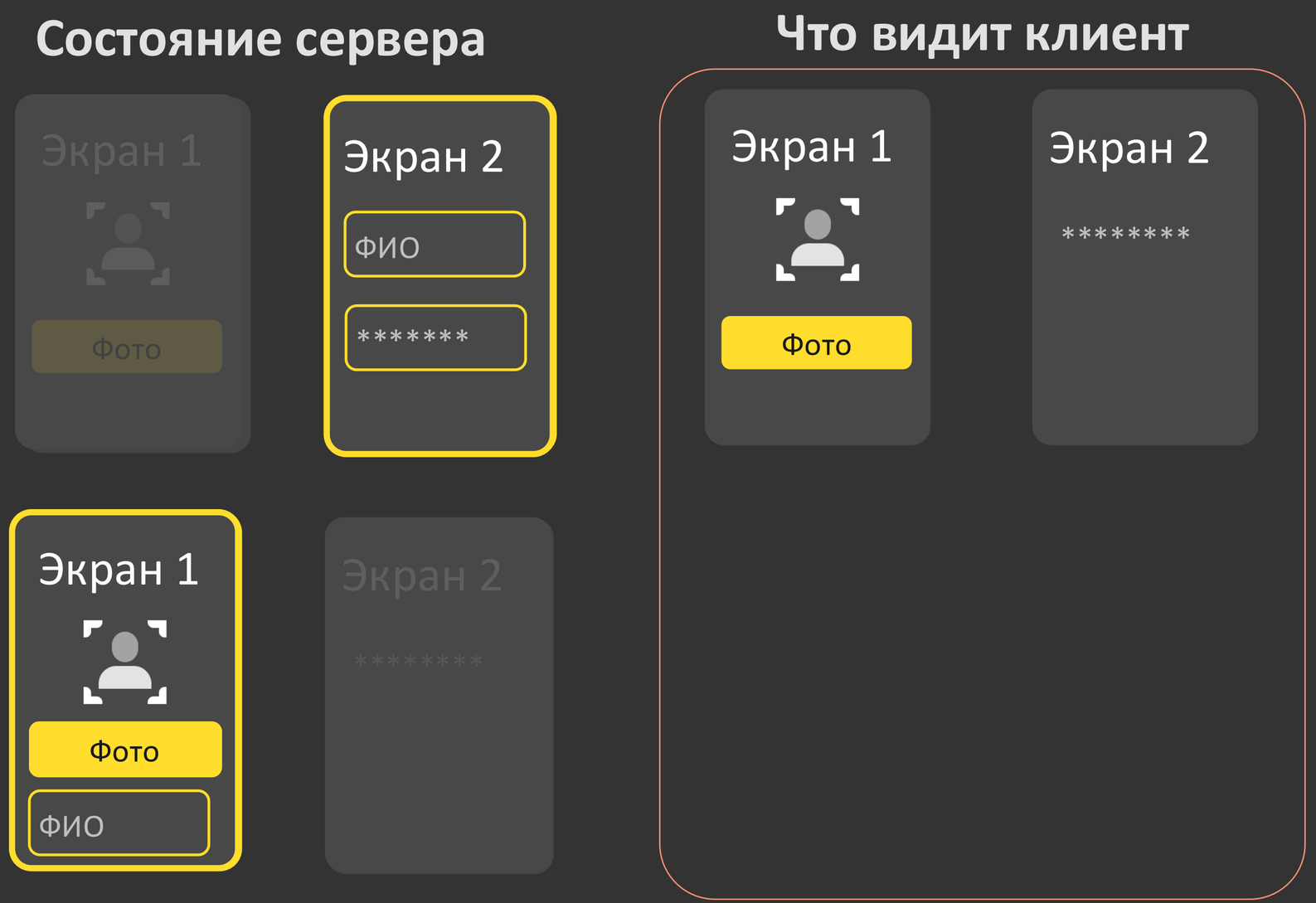
Возможные состояния кеша



Первый экран не успел обновиться

Поля ФИО нет

Возможные состояния кеша



Первый экран не успел обновиться
Поля ФИО нет

Второй экран не успел обновиться
2 поля ФИО

Поставка фичей бандлами

Регресс

Возможен регресс

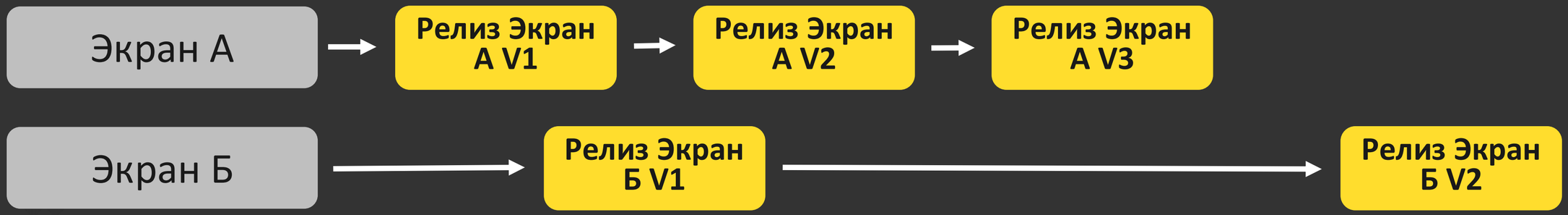
Совместимость

Поведенческая совместимость, меньше требований к контракту данных экрана

Вес артефактов

Все фичи используют одни версии либ, релиз весит меньше

Независимая поставка фичей



Нет регресса

Нельзя провести регресс, так как все релизятся в разные моменты времени

Поведенческая несовместимость

Выше требования к in-out контрактам экрана

Больший вес артефактов

Например у каждой фичи будет свой уникальный kotlinstd.js который будет дублироваться в кеше

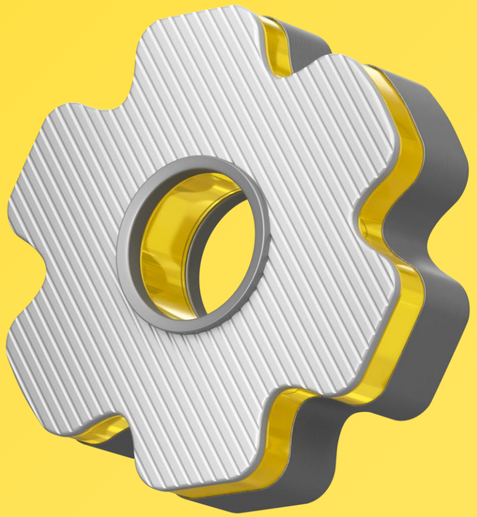
План доклада

- Появившийся вызов
- Исследование альтернатив
- Верхнеуровневая архитектура
- Релизы фичей
- **К чему надо быть готовым….**

А куда мы можем зарелизиться

Контракты

Важная часть любой SDUI системы – различные контракты и их обратная совместимость



Контракты UI элементов

Контракты функциональных возможностей

Контракты Bridges

Контракты SDUI Script

Проблема определения доступности

В системе постоянно появляются новые элементы.

Все они распределены по разным версиям SDK.

```
1    if (Button::class.existis) {
2        Button("text")
3    }
4
5    if (Text::class.existis) {
6        Text("text")
7    }
```

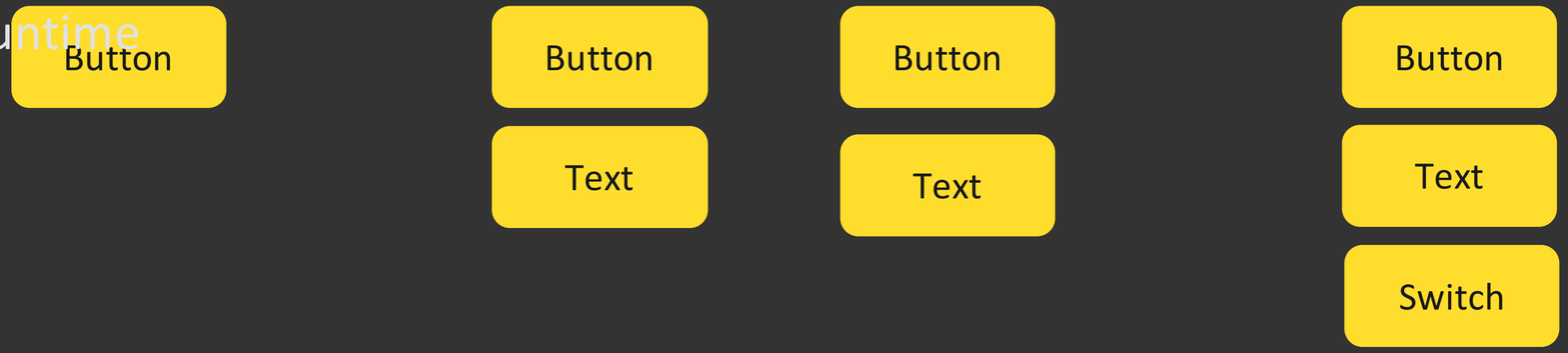
Как определить где запустится экран

Версии приложения

Элементы доступные в SDUI

Экран на SDUI

Runtime

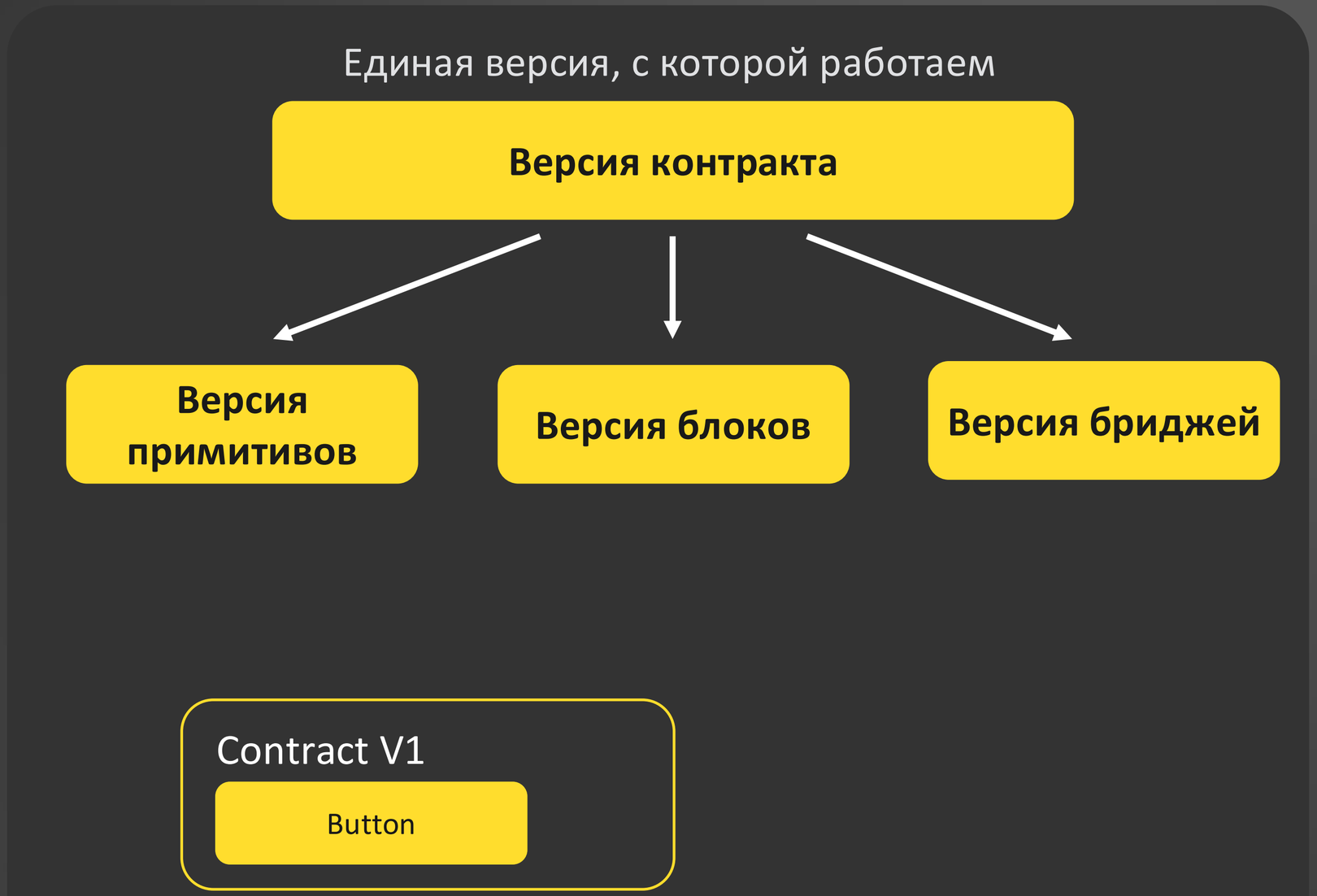


Общая версия контрактов

Единая версия всех контрактов

Общая версия контрактов

Единая версия всех контрактов



Общая версия контрактов

Единая версия всех контрактов

Общая версия контрактов

Единая версия всех контрактов

Общая версия контрактов

Contract V2

minSDK <> min iOS в обычном проекте

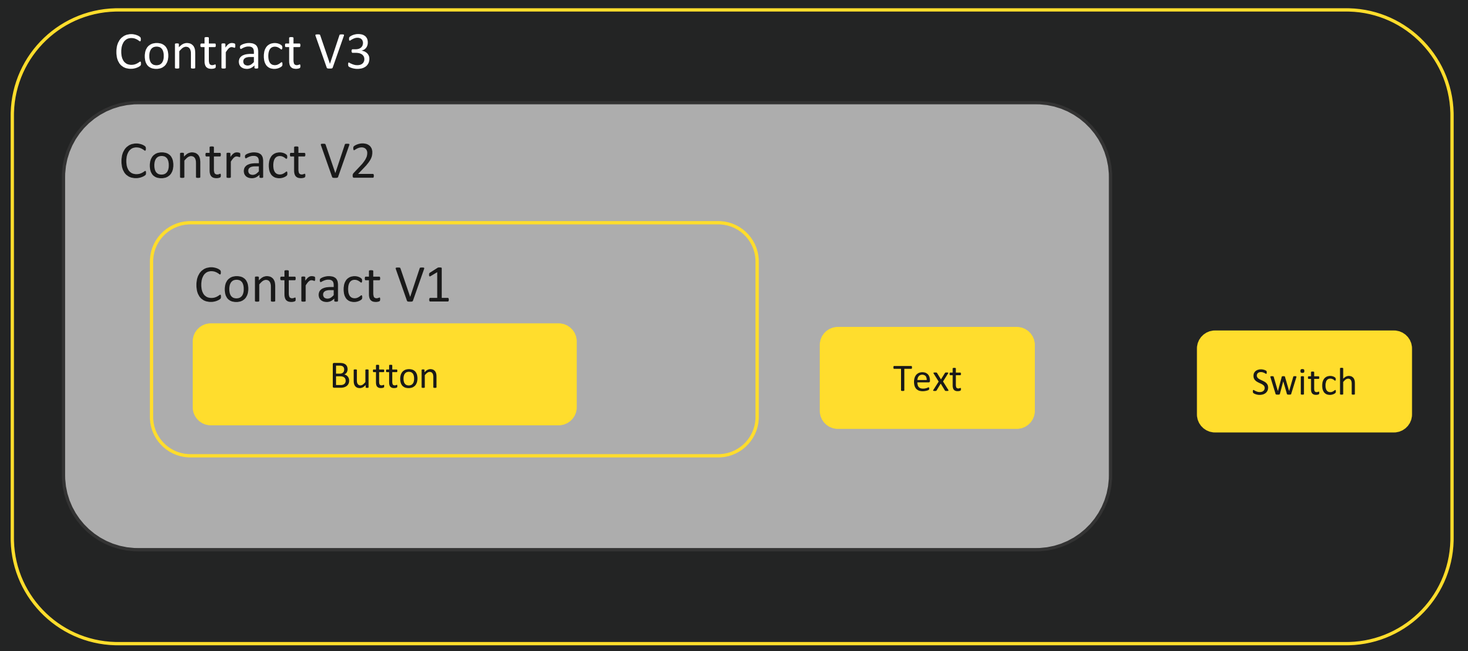
Можно безопасно использовать

Button

Text

Можно использовать только после проверки

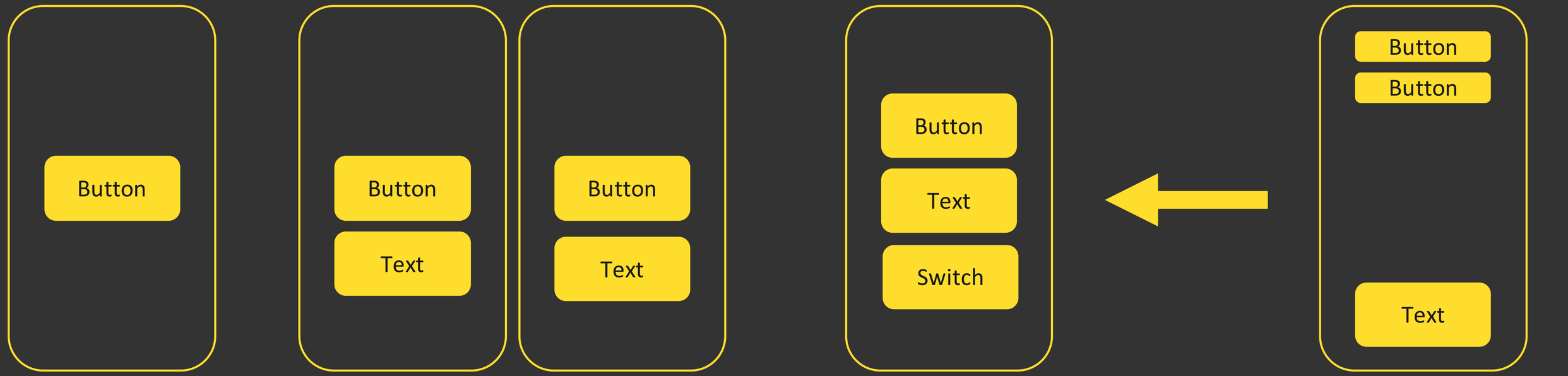
Switch



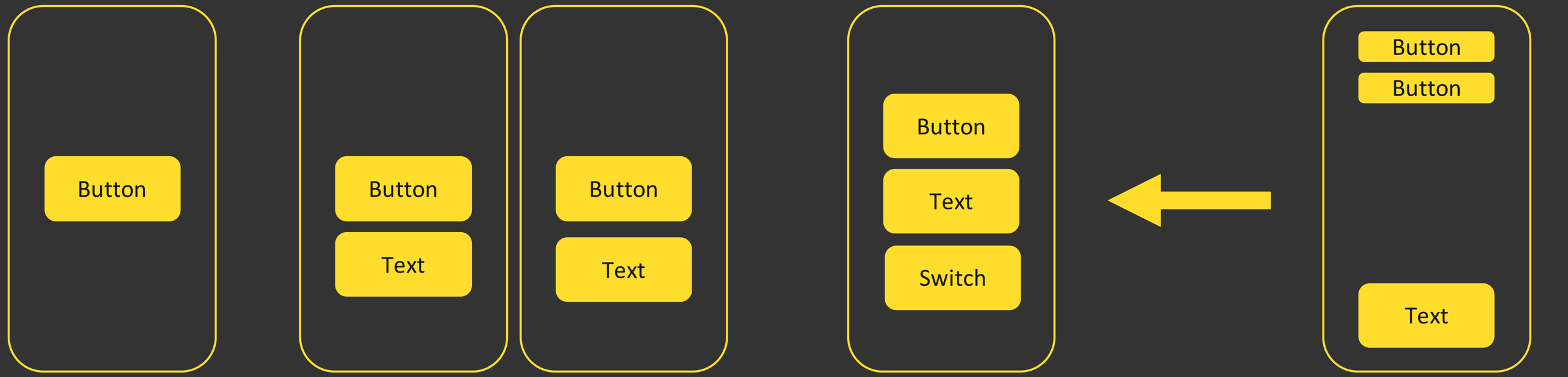
Общая версия контрактов

```
List {
    Section {
    }
    .modify { content in
        if #available(iOS 26, *) {
            content
                .sectionIndexLabel(Text("..."))
        } else {
            content
        }
    }
}
```

Определение безрелизного окна



Определение безрелизного окна



Код
не обновляется уже

Окно версий для поставки решения

Это еще не все нюансы!

Нюансы

Унификация UI в SDUI

01

02

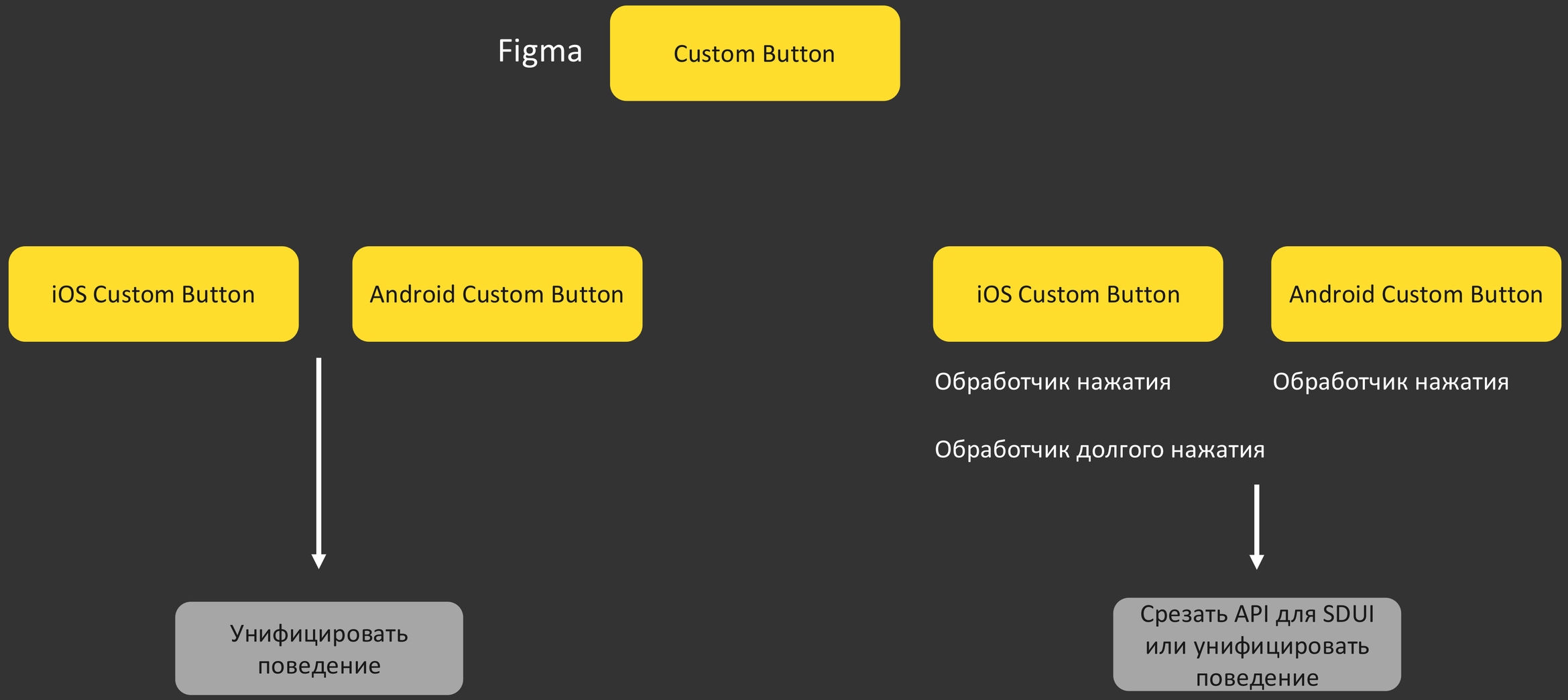
03

04

05

06

Унификация UI в SDUI



Нюансы

Унификация UI в SDUI

01

Все возможности нативной разработки (локализация, фича тоглы, персистенс) дорого тянуть в SDUI

02

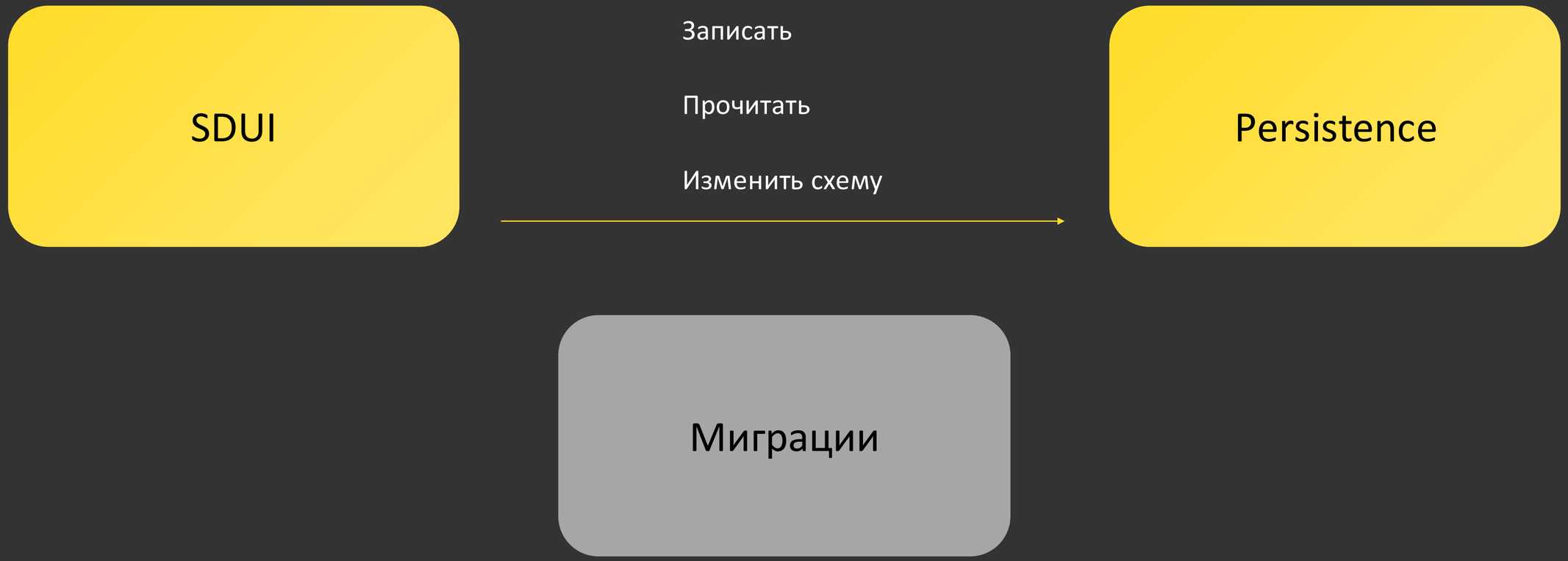
03

04

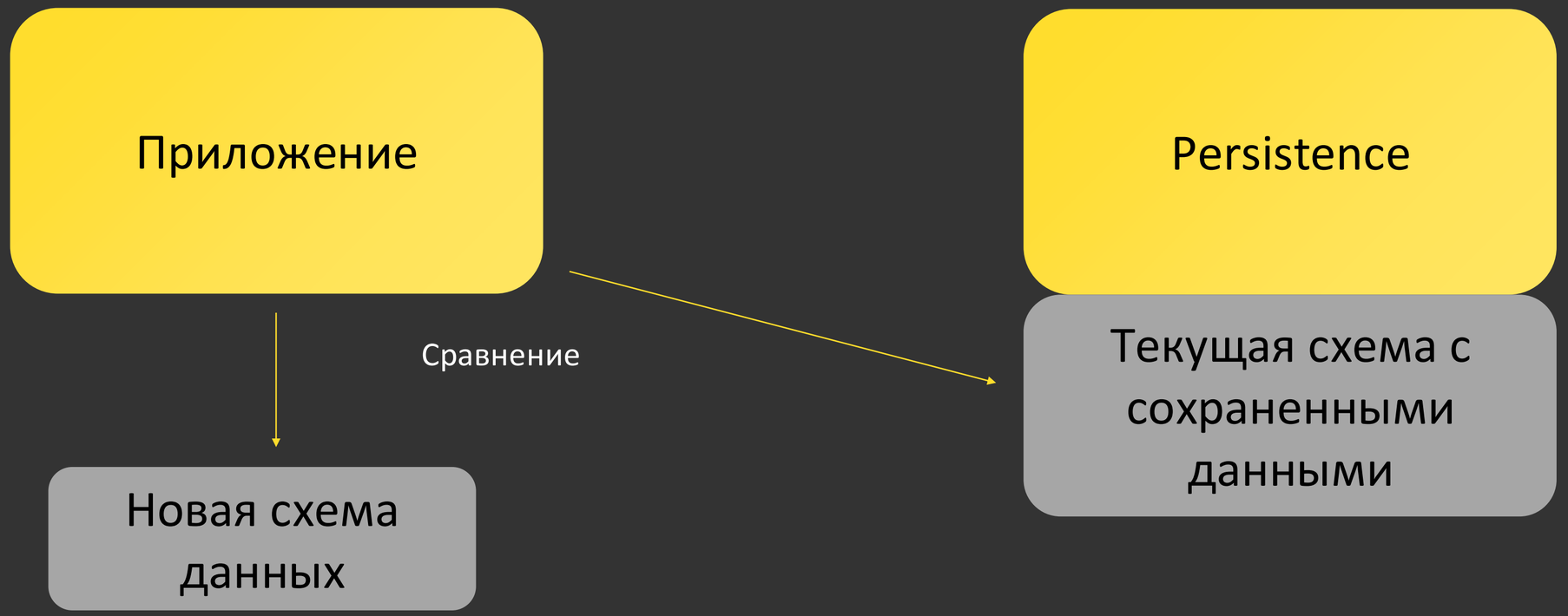
05

06

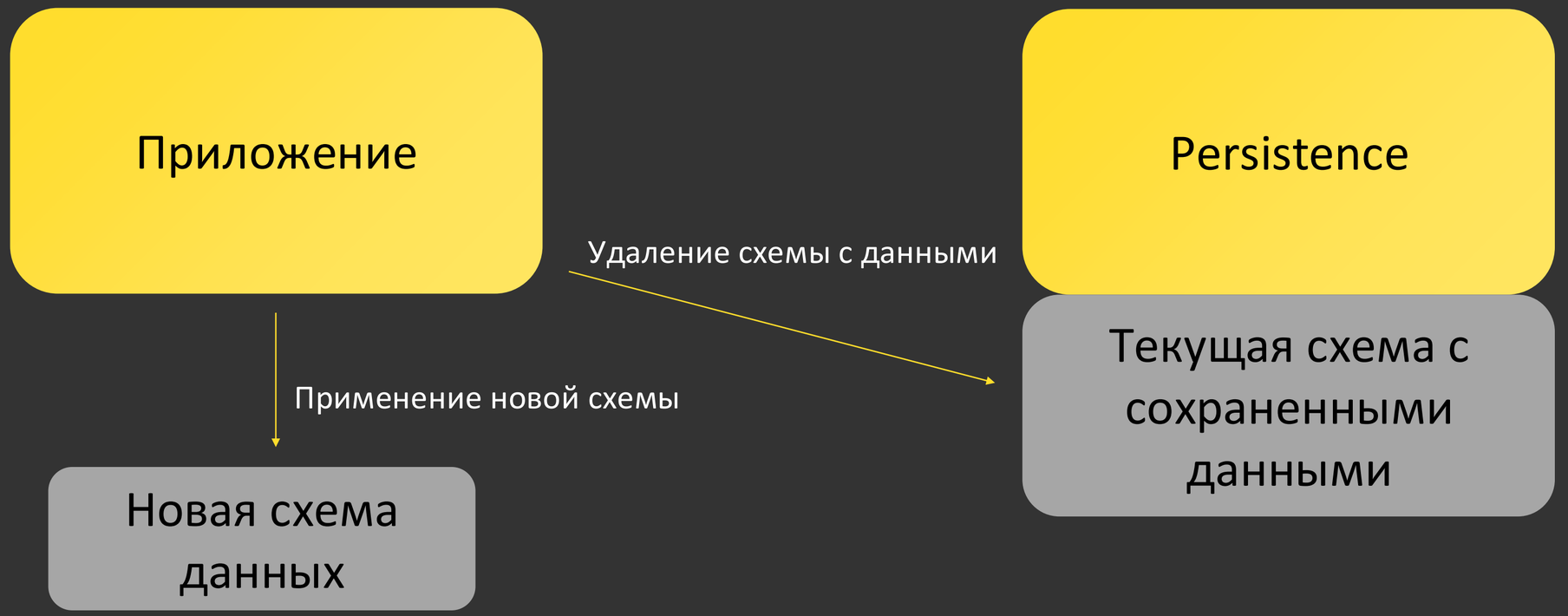
Какой сюрприз готовит персистенс



Как у нас работают миграции для натива

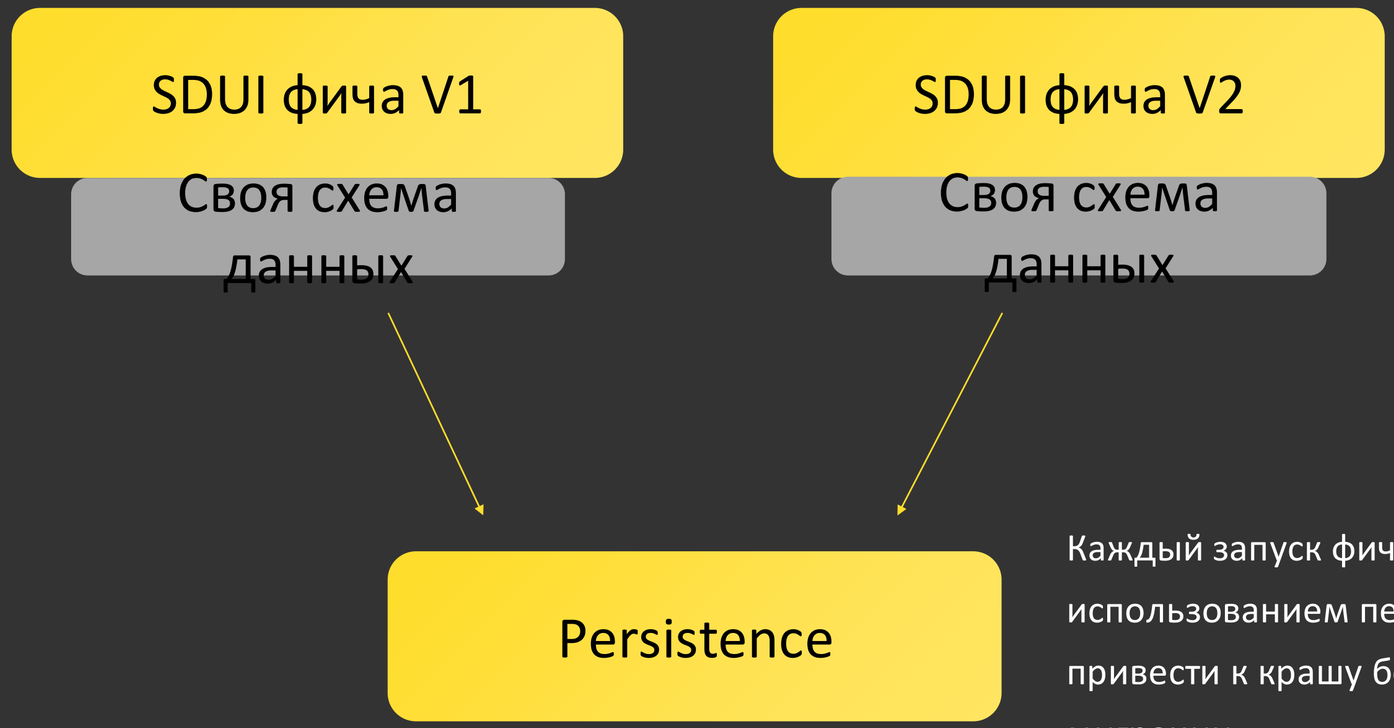


Как у нас работают миграции для натива



И все это работает на старте приложения

Как у нас работают миграции для SDUI



Каждый запуск фичи с использованием персистенса может привести к крашу без механизма миграции

Нюансы

Унификация UI в SDUI

01

Все возможности нативной разработки (локализация, фича тоглы, персистенс) дорого тянуть в SDUI

02

Кроссплатформенное API – это большой вызов

03

Разработка начинает включать в себе также бекенд паттерны

04

05

06

Нюансы

Унификация UI в SDUI

01

Все возможности нативной разработки (локализация, фича тоглы, персистенс) дорого тянуть в SDUI

02

Кроссплатформенное API – это большой вызов

03

Разработка начинает включать в себе также бекенд паттерны

04

Дорого первым заезжающим людям

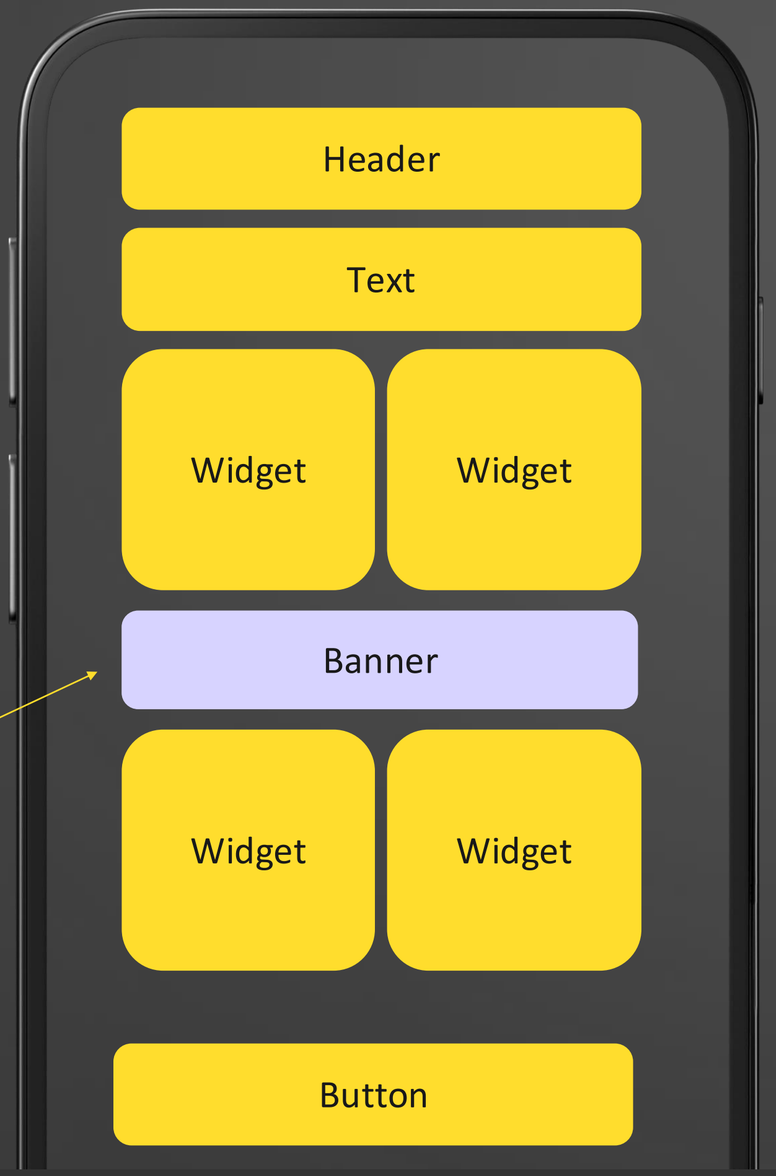
05

06

Пример фичи

Пример фичи

Придется договариваться с другой командой или самим переводить компонент на SDUI



Нюансы

Унификация UI в SDUI

01

Все возможности нативной разработки (локализация, фича тоглы, персистенс) дорого тянуть в SDUI

02

Кроссплатформенное API – это большой вызов

03

Разработка начинает включать в себе также бекенд паттерны

04

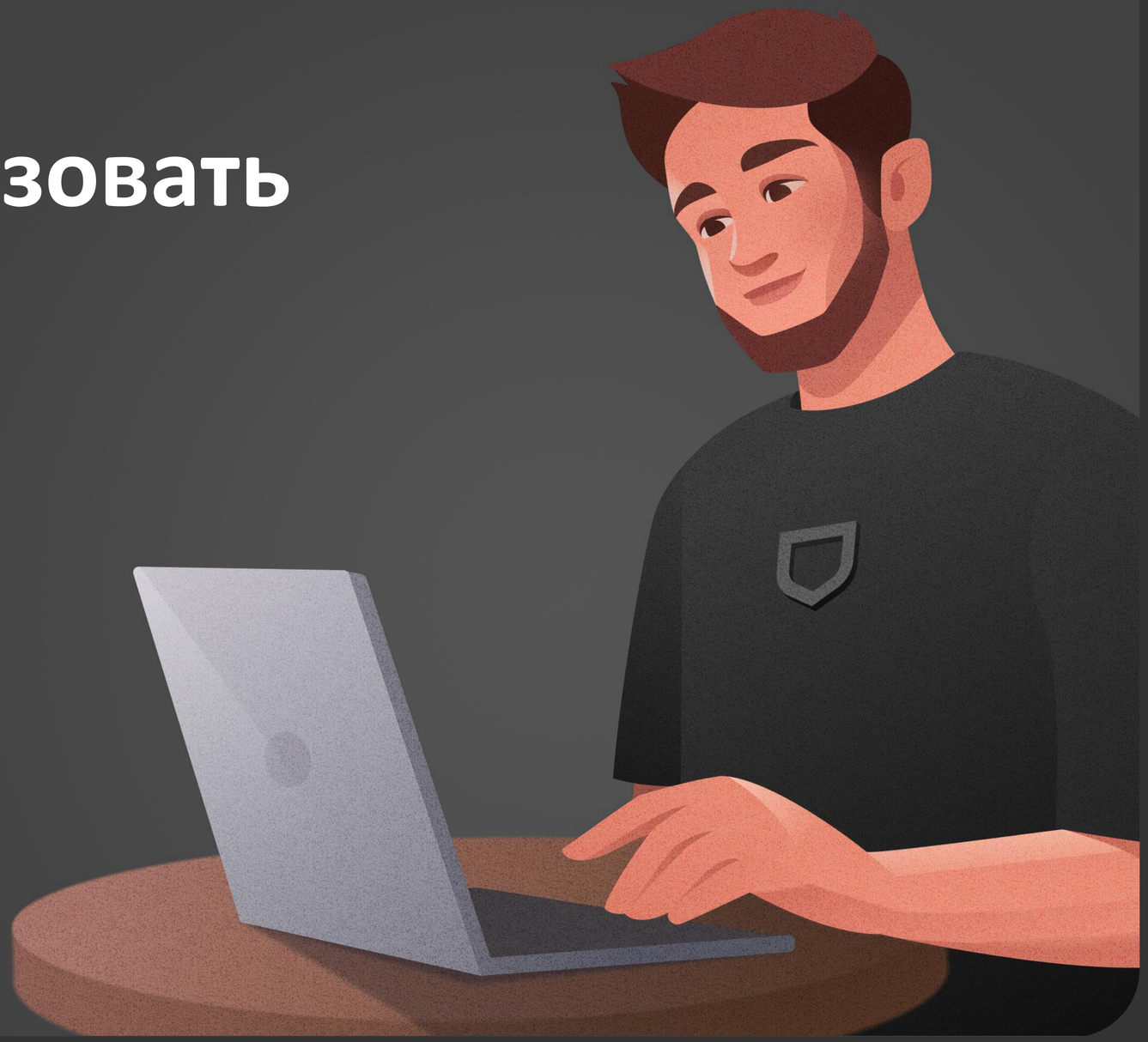
Дорого первым заезжающим людям

05

Надо сначала дождаться обновления SDUI движка у большинства пользователей

06

Что в итоге использовать вам?



Итоги

- Появившийся вызов
- Исследование альтернатив
- Верхнеуровневая архитектура
- Релизы фичей
- К чему надо быть готовым….

БАНК

Спасибо
за внимание!