Т БАНК

Как подружить мобильное приложение с JS и прокачать свой SDUI

Вакула Максим | iOS разработчик | Т-Банк
Валиев Тимур | Android разработчик | Т-Банк

Тимур Валиев

Android разработчик
t.valiev@tbank.ru
@teemch

Максим Вакула

iOS разработчик
m.n.vakula@tbank.ru
@VakulaMaksim

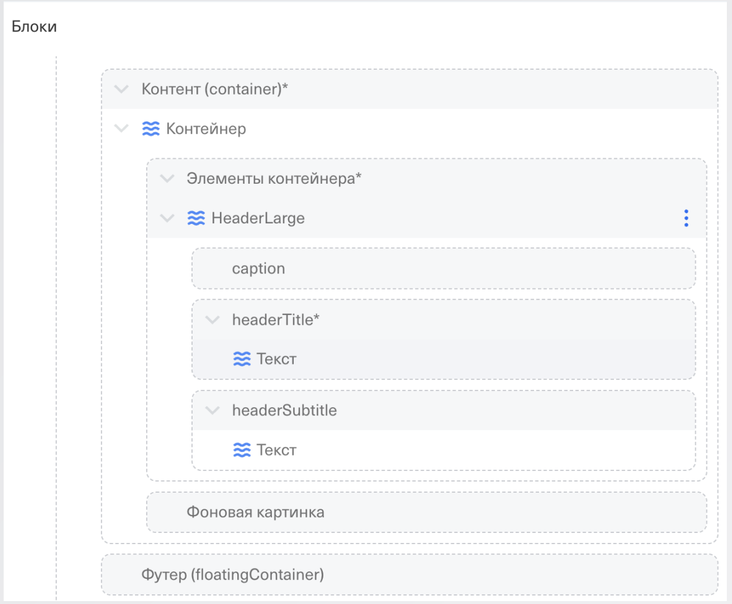
О чем мы вам расскажем

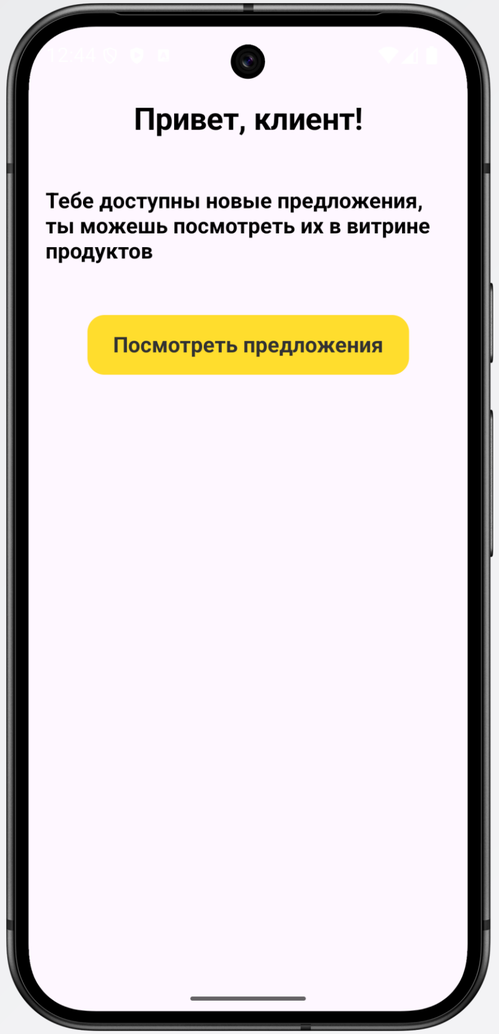
- Какое SDUI решение у нас уже было, и что мы хотели дальше
- Почему выбрали Zipline и как он работает
- Архитектура наших безрелизных фичей
- Проблемы, с которыми мы столкнулись

Путь к Kotlin/JS

У нас был SDUI движок

- Low Code решение
- Админка для верстки
- Статические экраны





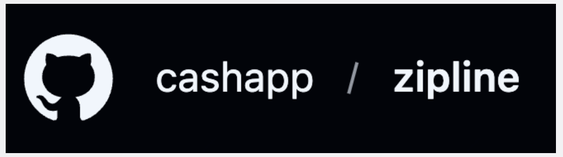
Что мы хотели

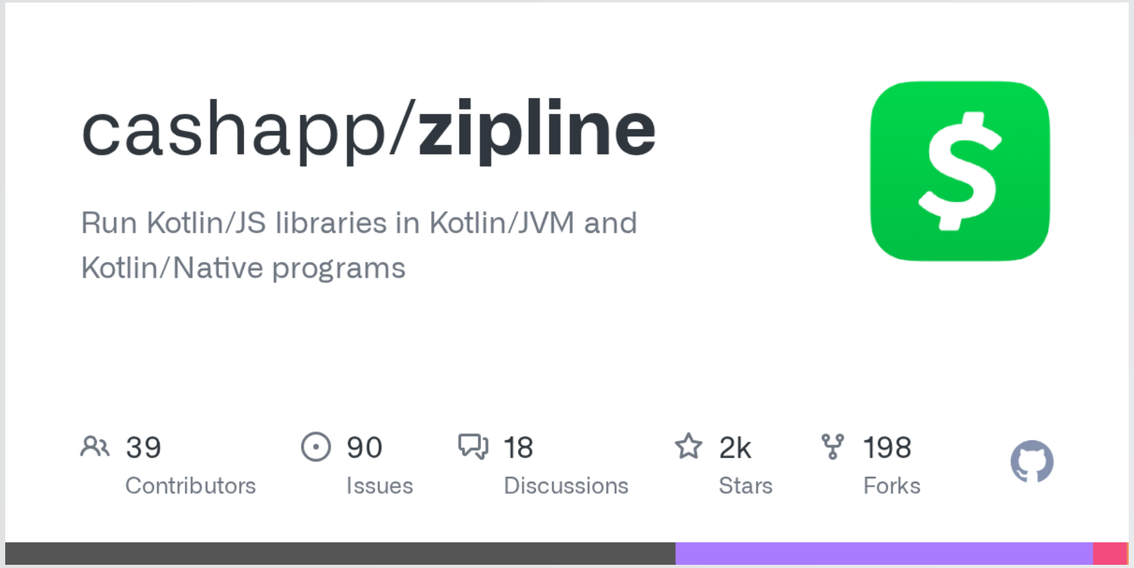
- Безрелизно реализовывать любую бизнес-логику
- Полностью нативный UI и UX
- Чтобы мобильщики могли делать все сами, без Web и backend разработчиков

SDUI + JS +

Как это реализовать?

- Скомпилировать Koltin в JS и опубликовать
- Загрузить исполняемый JS в мобильное приложение и запустить
- Подружить JS код с нативным

SDUI +  +  = 



- Готовый фреймворк для компиляции и публикации Kotlin/JS кода
- Позволяет легко строить протокол общения JS с нативом
- Инкапсулирует встроенный JS runtime и логику его запуска

* Источник: https://github.com/cashapp/zipline

JS Runtime

Запуск JS из мобильного приложения

1. Получить инстанс движка

```
val jsRuntime = JsRuntime()
```

Запуск JS из мобильного приложения

1. Получить инстанс движка
2. Загрузить JS код в движок

```
val jsRuntime = JsRuntime()


val jsCode = """
    function multiplyByTwo(number) {
        return number * 2;
    }
"""
jsRuntime.evaluate(jsCode)
```

Запуск JS из мобильного приложения

1. Получить инстанс движка
2. Загрузить JS код в движок
3. Сделать обращение к JS и получить результат

```
val jsRuntime = JsRuntime()


val jsCode = """
    function multiplyByTwo(number) {
        return number * 2;
    }
"""
jsRuntime.evaluate(jsCode)


val result = jsRuntime.evaluate("multiplyByTwo(21);")

println ("Результат: $result") // Результат: 42
```

Что может быть JS рантаймом?

WebView

Создаем на экране невидимую WebView
и загружаем в нее JS код

- ➕ Не увеличиваем размер приложения
- ➕ Отличная производительность
- ➖ Нужно костылить загрузку локальных JS файлов
- ➖ Ограничения протокола общения с WebView
- ➖ Нужно учитывать различия платформ и реализаций WebView

Бандлим свой JS движок в приложении

Используем готовый JS движок, отделенный от UI
движка, и поставляем его вместе с приложением

- ➕ Движок полностью под нашим контролем
- ➕ Прямолинейная логика загрузки кода и обращения к JS коду
- ➖ Увеличиваем размер приложения

JS runtime в zipline:

bellard/**quickjs**

Public repository of the QuickJS Javascript Engine.

17 Contributors

19 Used by

11k Stars

1k Forks

Источник: https://github.com/bellard/quickjs

Почему QuickJS?

Мало весит

Бинарь движка весит всего 360кб

Предкомпиляция

Позволяет нам заранее компилить JS в байткод, чтобы не тратить на это время в рантайме

Сильно урезан

По сравнению с полноценными движками, имеет урезанную стандартную библиотеку и менее производителен

QuickJs API

```
expect class QuickJs : AutoCloseable {

  ...

}
```

QuickJs API

```
expect class QuickJs : AutoCloseable {

  companion object {
    fun create(): QuickJs
  }

  ...









}
```

QuickJs API

```
expect class QuickJs : AutoCloseable {

  companion object {
    fun create(): QuickJs
  }

  /** Скомпилировать JS в байткод */
  fun compile(sourceCode: String): ByteArray

  ...




}
```

QuickJs API

```
expect class QuickJs : AutoCloseable {

  companion object {
    fun create(): QuickJs
  }

  /** Скомпилировать JS в байткод */
  fun compile(sourceCode: String): ByteArray

  /** Исполнить JS байткод */
  fun execute(bytecode: ByteArray): Any?

  ...



}
```

QuickJs API

```
expect class QuickJs : AutoCloseable {

  companion object {
    fun create(): QuickJs
  }

  /** Скомпилировать JS в байткод */
  fun compile(sourceCode: String): ByteArray

  /** Исполнить JS байткод */
  fun execute(bytecode: ByteArray): Any?

  /** Исполнить JS код  */
  fun evaluate(script: String): Any?

  ...
}
```

QuickJs API

```
expect class QuickJs : AutoCloseable {

  ...

  /** Прокинуть канал для отправки событий внутрь JS */
  internal fun initOutboundChannel(outboundChannel: CallChannel)

  /** Канал для получения событий из JS */
  internal fun getInboundChannel(): CallChannel
}
```

Превращаем Kotlin в JS байткод

Kotlin ➔ JS

build.gradle.kts

```
plugins {
  alias(libs.plugins.kotlinMultiplatform)
}

...
```

Kotlin ➔ JS

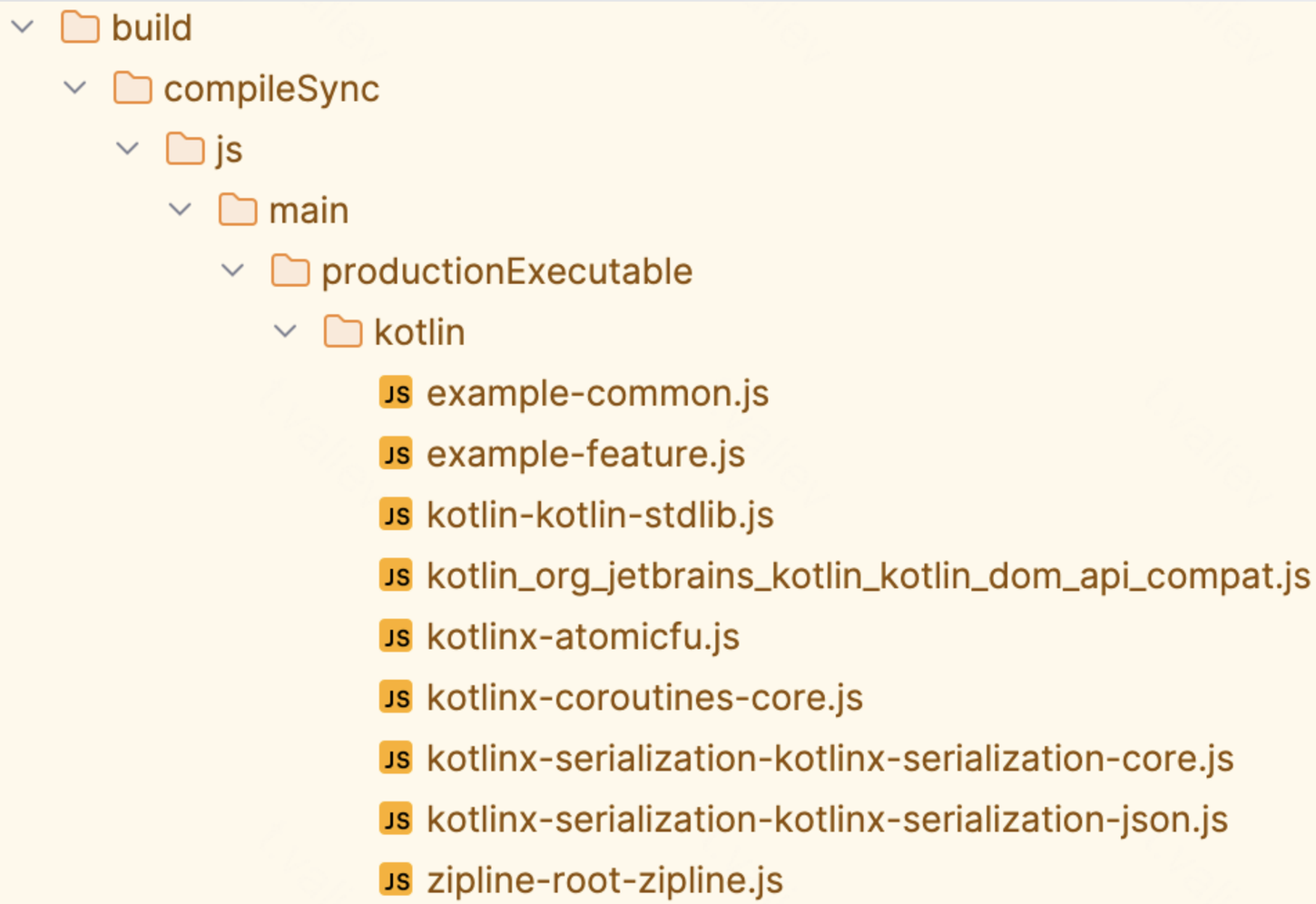
build.gradle.kts

```
plugins {
  alias(libs.plugins.kotlinMultiplatform)
}

kotlin {
  js(IR) {
    browser()
    binaries.executable()
  }
}
```

Kotlin ➔ JS

```
> gradle compileProductionExecutableKotlinJs
```



JS → bytecode

QuickJs.compile(

JS → bytecode

```
> gradle compileProductionExecutableKotlinJsZipline
```

Manifest фичи

Список исполняемых файлов

mainfest.zipline.json

```
{
  "modules": {
    "./kotlin-kotlin-stdlib.js": { ... },
    "./kotlinx-coroutines-core.js": { ... },
    "./kotlinx-serialization-json.js": { ... },
    "./zipline-root-zipline.js": { ... },
    "./feature-increment-counter.js": { ... }
  },
  ...
}
```

Manifest фичи

Список исполняемых файлов

Hash исполняемого файла

mainfest.zipline.json

```
{
  "modules": {
    "./kotlin-kotlin-stdlib.js": { ... },
    "./kotlinx-coroutines-core.js": { ... },
    "./kotlinx-serialization-json.js": { ... },
    "./zipline-root-zipline.js": { ... },
    "./feature-increment-counter.js": {
      "url": "feature-increment-counter.zipline",
      "sha256": "75f6d945cfeccc...",
      ...
    }
  },
  ...

}
```

Manifest фичи

Список исполняемых файлов

Hash исполняемого файла

Список зависимостей файла

mainfest.zipline.json

```
{
  "modules": {
    "./kotlin-kotlin-stdlib.js": { ... },
    "./kotlinx-coroutines-core.js": { ... },
    "./kotlinx-serialization-json.js": { ... },
    "./zipline-root-zipline.js": { ... },
    "./feature-increment-counter.js": {
      "url": "feature-increment-counter.zipline",
      "sha256": "75f6d945cfeccc...",
      "dependsOnIds": [
        "./kotlin-kotlin-stdlib.js",
        "./kotlinx-serialization-kotlinx-serialization-json.js",
        "./zipline-root-zipline.js"
      ]
    }
  },
  ...




}
```

Manifest фичи

Список исполняемых файлов

Hash исполняемого файла

Список зависимостей файла

Указатель на main функцию

mainfest.zipline.json

```
{
  "modules": {
    "./kotlin-kotlin-stdlib.js": { ... },
    "./kotlinx-coroutines-core.js": { ... },
    "./kotlinx-serialization-json.js": { ... },
    "./zipline-root-zipline.js": { ... },
    "./feature-increment-counter.js": {
      "url": "feature-increment-counter.zipline",
      "sha256": "75f6d945cfeccc...",
      "dependsOnIds": [
        "./kotlin-kotlin-stdlib.js",
        "./kotlinx-serialization-kotlinx-serialization-json.js",
        "./zipline-root-zipline.js"
      ]
    }
  },
  "mainModuleId": "./feature-increment-counter.js",
  "mainFunction": "ru.tbank.feature.incrementcounter.main",
  ...



}
```

Manifest фичи

Список исполняемых файлов

Hash исполняемого файла

Список зависимостей файла

Указатель на main функцию

Подписи манифеста

mainfest.zipline.json

```
{
  "modules": {
    "./kotlin-kotlin-stdlib.js": { ... },
    "./kotlinx-coroutines-core.js": { ... },
    "./kotlinx-serialization-json.js": { ... },
    "./zipline-root-zipline.js": { ... },
    "./feature-increment-counter.js": {
      "url": "feature-increment-counter.zipline",
      "sha256": "75f6d945cfeccc...",
      "dependsOnIds": [
        "./kotlin-kotlin-stdlib.js",
        "./kotlinx-serialization-kotlinx-serialization-json.js",
        "./zipline-root-zipline.js"
      ]
    }
  },
  "mainModuleId": "./feature-increment-counter.js",
  "mainFunction": "ru.tbank.feature.incrementcounter.main",
  "unsigned": {
    "signatures": {
      "key1": "99b5a6f6875d72ab9bb86b5b..."
    }
  }
}
```

Загрузка кода

mainfest.zipline.json

```
val zipline = ziplineLoader.loadOnce(
  manifestUrl = "https://cdn.example.com/manifest.zipline.json",
  ...
)
```

Дружим натив с JS-ом

Что такое ZiplineService?

ExampleService.kt

```
interface ExampleService: ZiplineService {

    fun sendRequest(request: NetworkRequest): NetworkResult
}
```

Что такое ZiplineService?

ExampleService.kt

```kotlin
interface ExampleService: ZiplineService {

    fun sendRequest(request: NetworkRequest): NetworkResult

    suspend fun sendRequest(request: NetworkRequest): NetworkResult
}
```

Что такое ZiplineService?

ExampleService.kt

```kotlin
interface ExampleService: ZiplineService {

    fun sendRequest(request: NetworkRequest): NetworkResult

    suspend fun sendRequest(request: NetworkRequest): NetworkResult

    fun subscribe(request: NetworkRequest): Flow<NetworkResult>
}
```

Что такое ZiplineService?

ExampleService.kt

```
interface ExampleService: ZiplineService {

    fun sendRequest(request: NetworkRequest): NetworkResult

    suspend fun sendRequest(request: NetworkRequest): NetworkResult

    fun subscribe(request: NetworkRequest): Flow<NetworkResult>
}


@Serializable
class NetworkRequest(...)

@Serializable
class NetworkResult(...)
```

Что такое ZiplineService?

```
// Реализуется в нативе, используется в JS
interface NetworkService: ZiplineService {

    suspend fun send(request: NetworkRequest): NetworkResult
}
```

Что такое ZiplineService?

```
// Реализуется в нативе, используется в JS
interface NetworkService: ZiplineService {

    suspend fun send(request: NetworkRequest): NetworkResult
}

// Реализуется в JS, используется в нативе
interface ScreenContentProvider: ZiplineService {

    suspend fun provideContent(uiID: String): Result<ScreenContent>
}
```

Как использовать сервисы

1. Биндим реализацию сервиса со стороны натива

ExampleScreen.android.kt

```
val zipline = ziplineLoader.load(...)

val serviceImpl = ExampleServiceImpl()
```

Как использовать сервисы

1. Биндим реализацию сервиса со стороны натива

ExampleScreen.android.kt

```
val zipline = ziplineLoader.load(...)

val serviceImpl = ExampleServiceImpl()
zipline.bind<ExampleService>("exampleService", serviceImpl)
```

Как использовать сервисы

1. Биндим реализацию сервиса со стороны натива

ExampleScreen.android.kt

```
val zipline = ziplineLoader.load(...)

val serviceImpl = ExampleServiceImpl()
zipline.bind<ExampleService>("exampleService", serviceImpl)
```

2. Получаем со стороны JS реализацию того же интерфейса, но с бриджингом в натив

main.js.kt

```
fun main() {
  val zipline = Zipline.get()
  val service = zipline.take<ExampleService>("exampleService")

  ...
}
```

Как использовать сервисы

1. Биндим реализацию сервиса со стороны натива

ExampleScreen.android.kt

```
val zipline = ziplineLoader.load(...)

val serviceImpl = ExampleServiceImpl()
zipline.bind<ExampleService>("exampleService", serviceImpl)

...

val uiService = zipline.take<UIService>("uiService")
```

2. Получаем со стороны JS реализацию того же интерфейса, но с бриджингом в натив

3. Можно сделать и наоборот: bind в JS и take в нативе

main.js.kt

```
fun main() {
  val zipline = Zipline.get()
  val service = zipline.take<ExampleService>("exampleService")

  ...

  zipline.bind<UIService>("uiService", UiServiceImpl(service))
}
```

Как работает сервис под капотом

Пример сервиса

ExampleService.kt

```
interface EchoService : ZiplineService {

  fun echo(request: EchoRequest): EchoResponse
}
```

Как получить реализацию сервиса

Moblie App

interface **EchoService**

val service: EchoService = EchoServiceImpl()

JS Runtime

interface **EchoService**

class **EchoServiceImpl**

Как получить реализацию сервиса

Moblie App

interface **EchoService**

(generated)
class **OutboundService**
: EchoService

val service = zipline.take<EchoService>()

JSON

JS Runtime

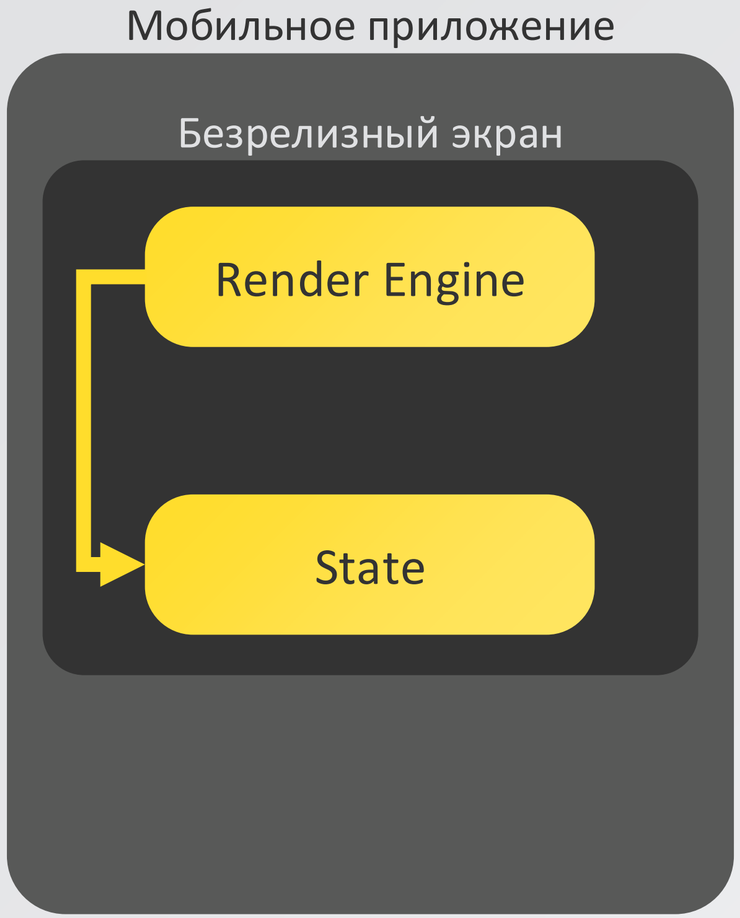
interface **EchoService**

class **EchoServiceImpl**

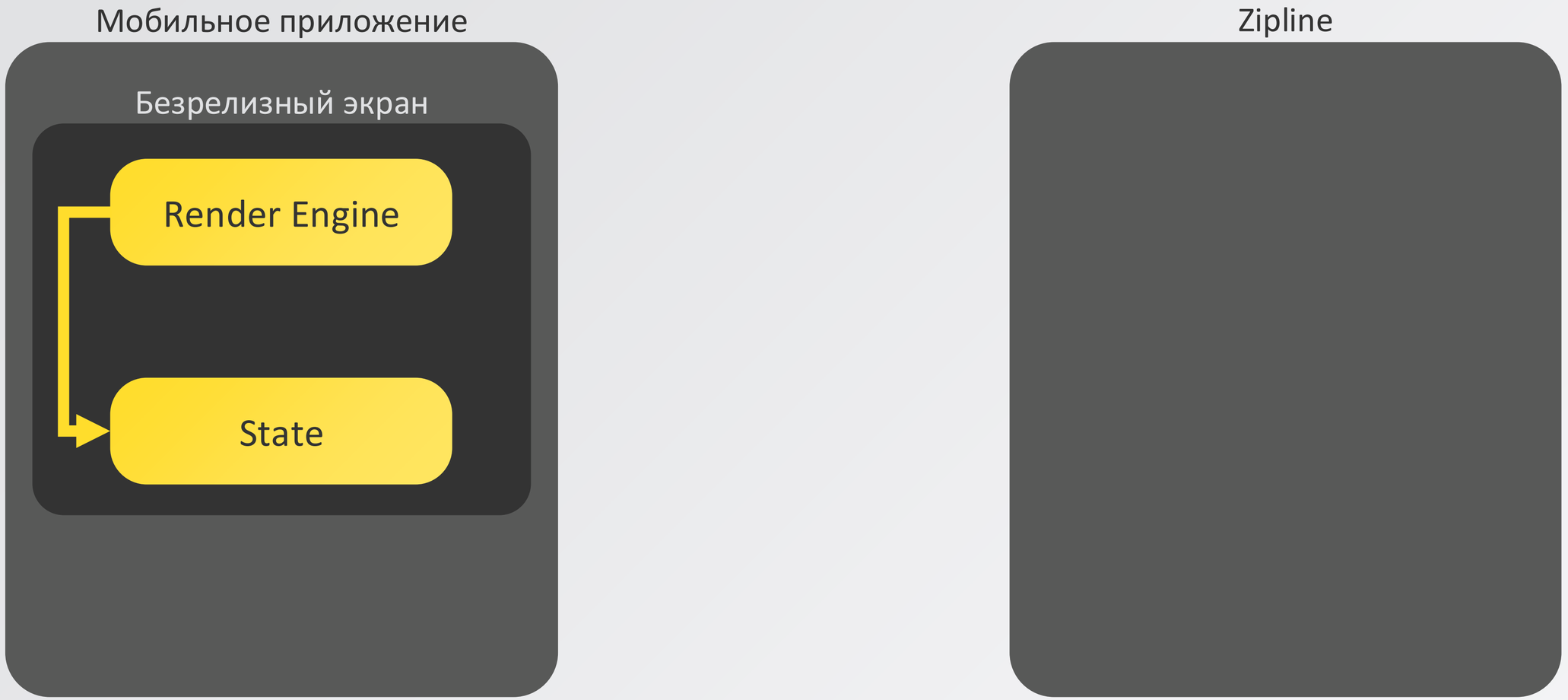
zipline.bind<EchoService>(EchoServiceImpl())

Архитектура безрелизных фичей

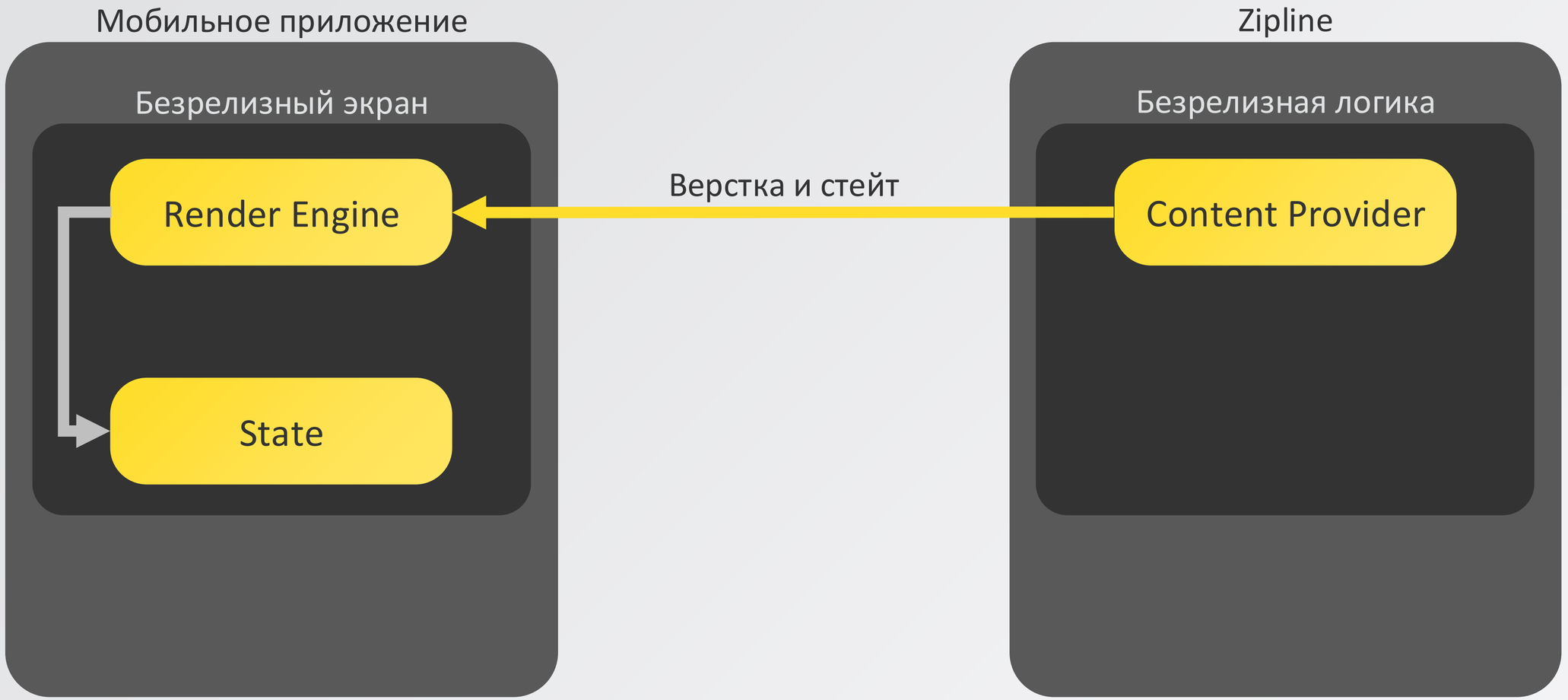
Архитектура. SDUI



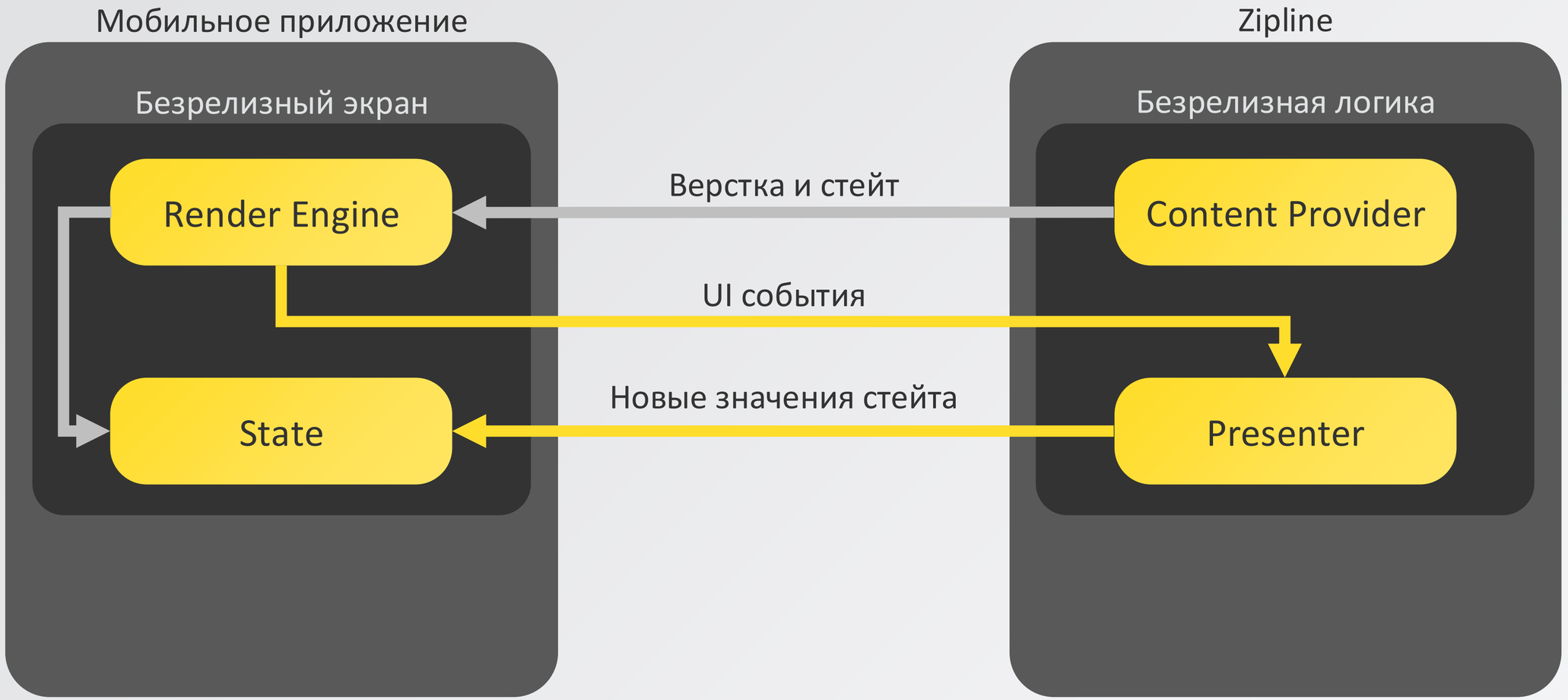
Архитектура. SDUI



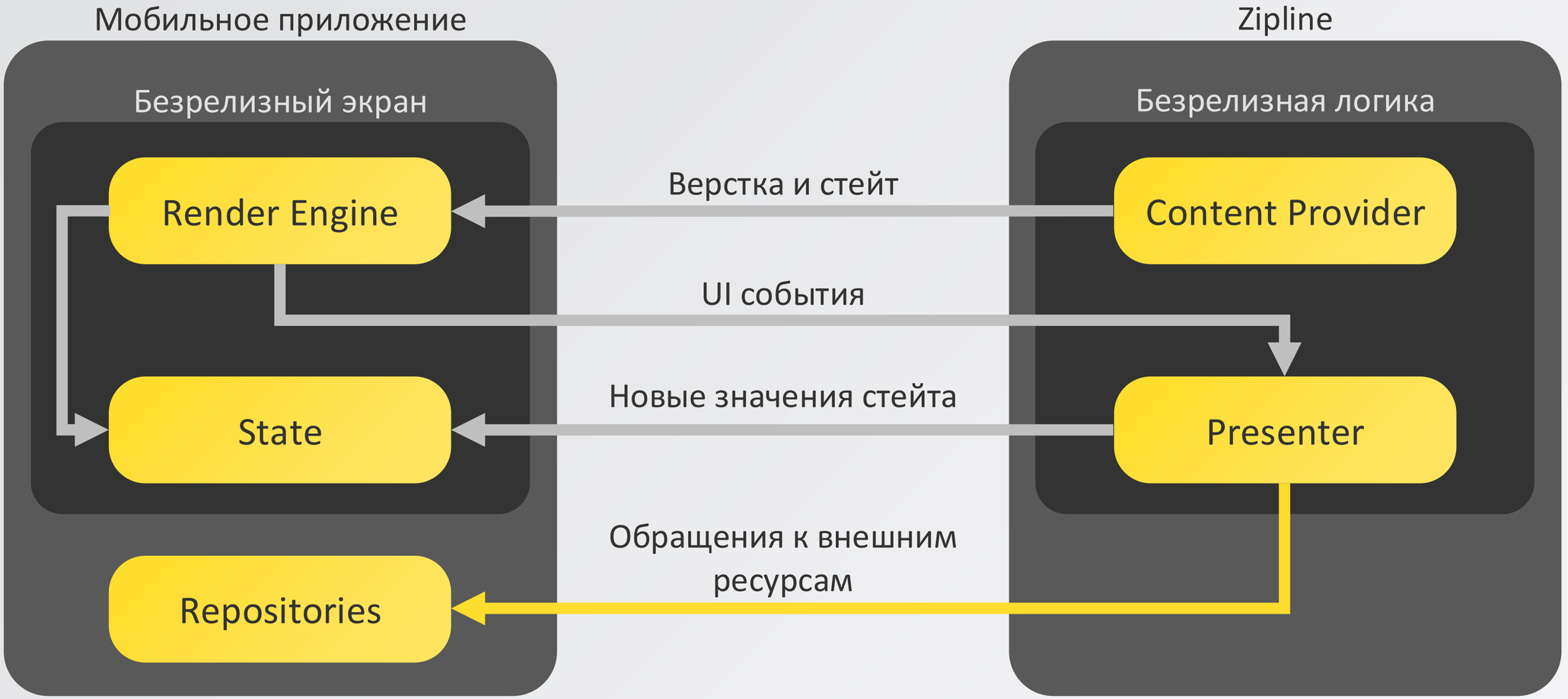
Архитектура. Отрисовка



Архитектура. Обработка событий



Архитектура. Внешние вызовы



Проблемы, о которых не догадывались

QuickJS – сильно урезан. Однопоточное исполнение

codeSampleJvm.kt

```kotlin
suspend fun processListInParallel(
    list: List<Int>,
    chunkSize: Int
): List<Int> {
    val chunks = list.chunked(chunkSize)

    val precessedChunks = chunks.map { chunk ->
        async(Dispatchers.Default) {
            chunk.map { it * 2 }
        }
    }.awaitAll()

    val result = precessedResult.flatten()

    return result // ~5 ms
}
```

QuickJS – сильно урезан. Однопоточное исполнение

codeSampleJs.kt

```kotlin
suspend fun processListInParallel(
    list: List<Int>,
    chunkSize: Int
): List<Int> {
    val chunks = list.chunked(chunkSize)

    val precessedChunks = chunks.map { chunk ->
        async(Dispatchers.Default) {
            chunk.map { it * 2 }
        }
    }.awaitAll()

    val result = precessedResult.flatten()

    return result // ~25 ms
}
```

QuickJS – сильно урезан. Однопоточное исполнение

eventLoop.kt

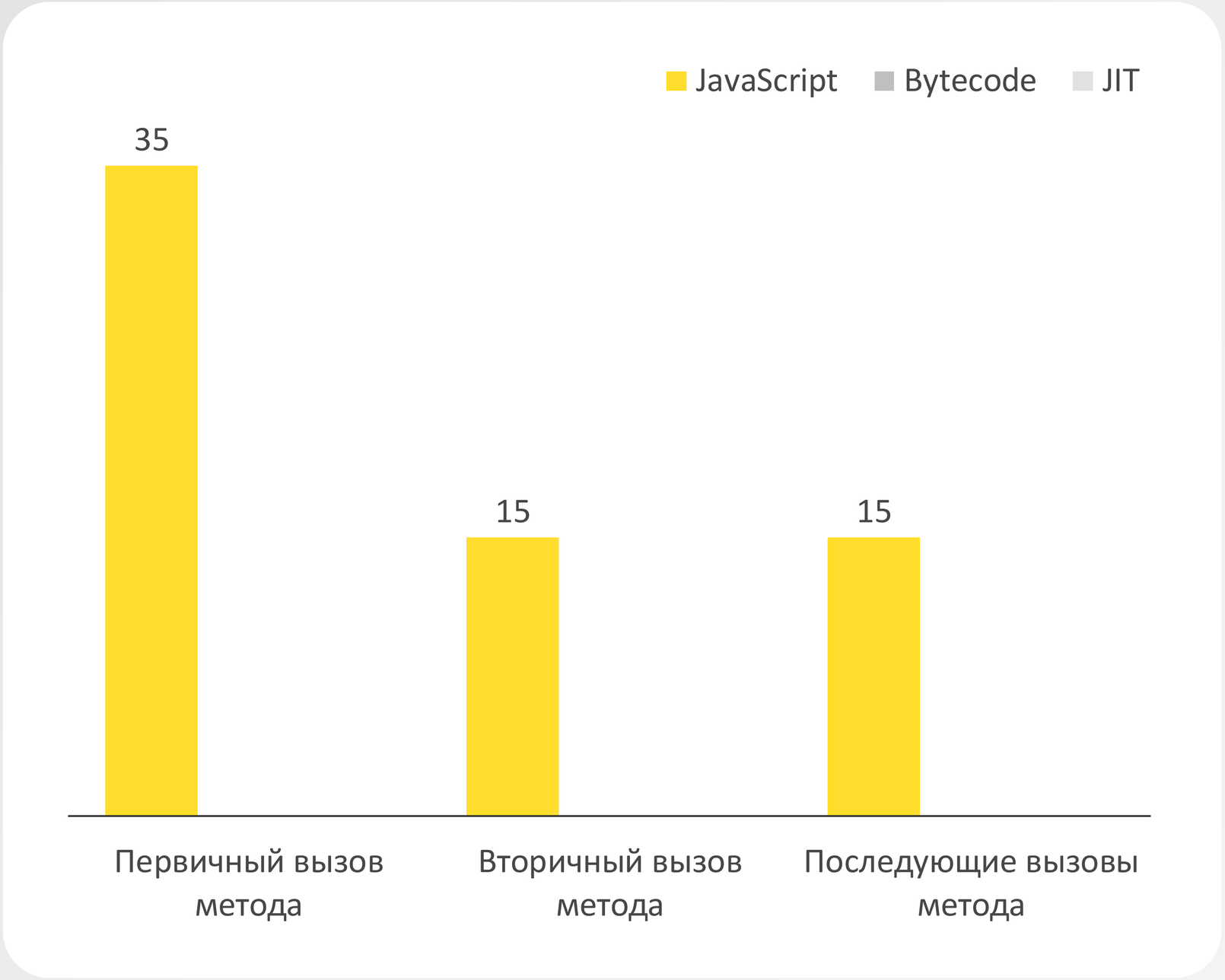
```kotlin
fun blockingTask() {
    val start = System.currentTimeMillis()
    while (System.currentTimeMillis() - start < 5000) { }
    println("Блокирующая задача выполнена")
}

fun main() = runBlocking {
    println("Начало")
    launch {
        delay(1000)
        println("Таймер 1 сек")
    }
    blockingTask()
    println("Конец")
}

/*
 * Начало
 * Блокирующая задача выполнена <-- таймер ждал 5 сек!
 * Таймер 1 сек
 * Конец
 */
```

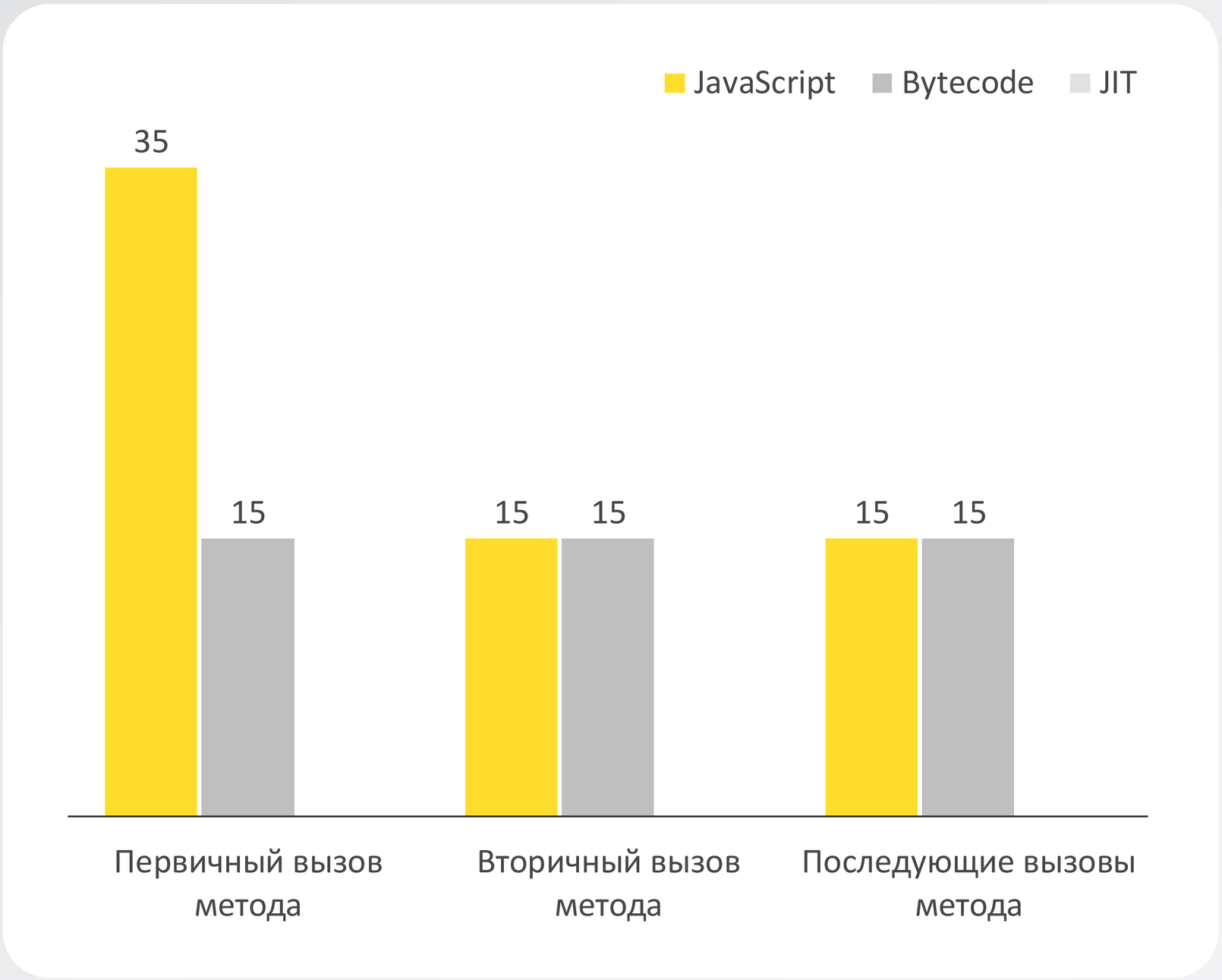
QuickJS – сильно урезан. Интерпретация JavaScript кода

➔ Исходный код



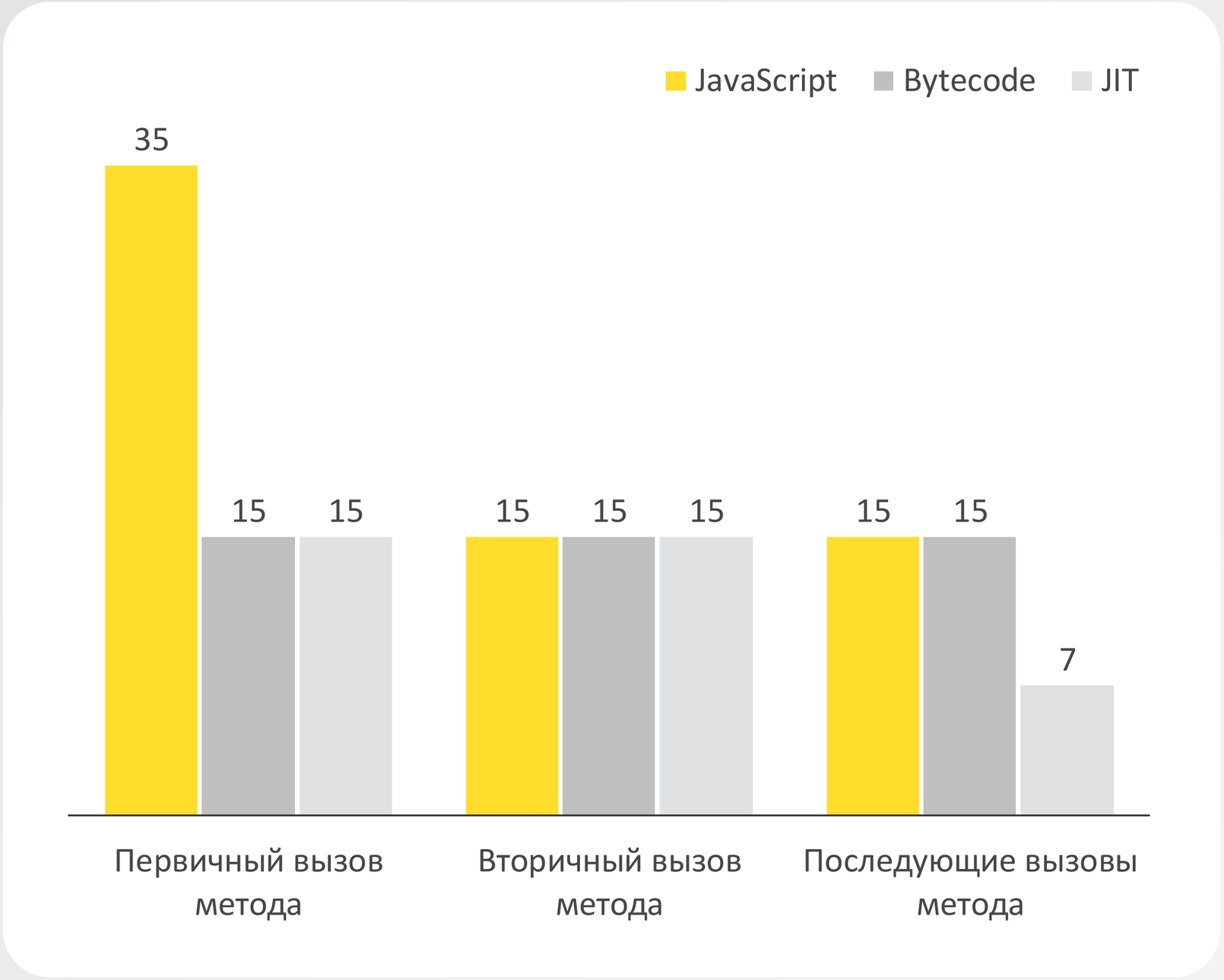
QuickJS – сильно урезан. Интерпретация JavaScript кода

- ➜ Исходный код
- ➜ Bytecode



QuickJS – сильно урезан. Интерпретация JavaScript кода

- Исходный код
- Bytecode
- JIT компиляция



QuickJS – сильно урезан. Отладка кода

Debug Adapter Protocol

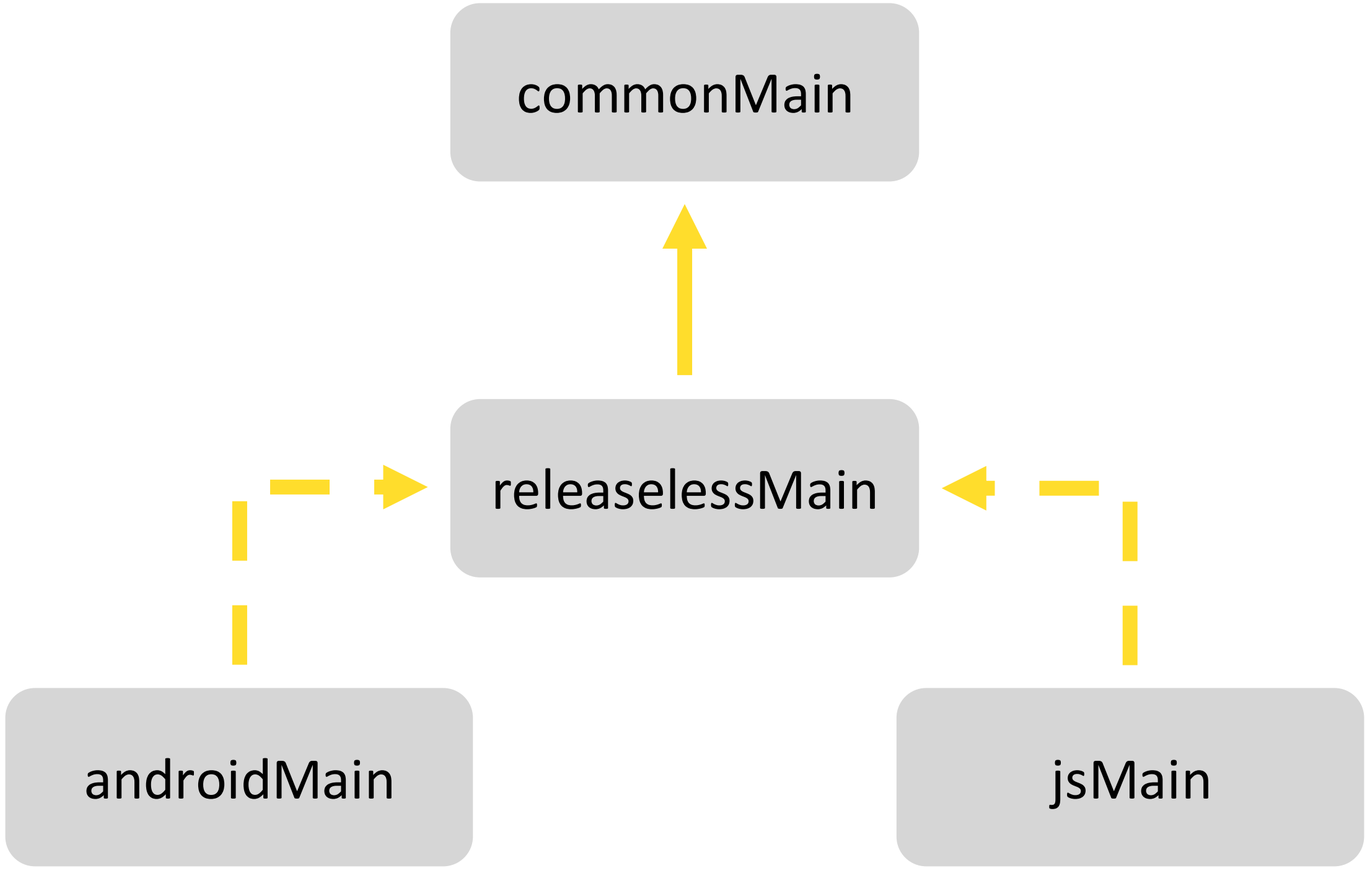
QuickJS Debug for VS Code

QuickJS: The Next Generation

* Источник: https://microsoft.github.io/debug-adapter-protocol/
* * Источник: https://github.com/quickjs-ng/quickjs
* * * Источник: https://github.com/koush/vscode-quickjs-debug

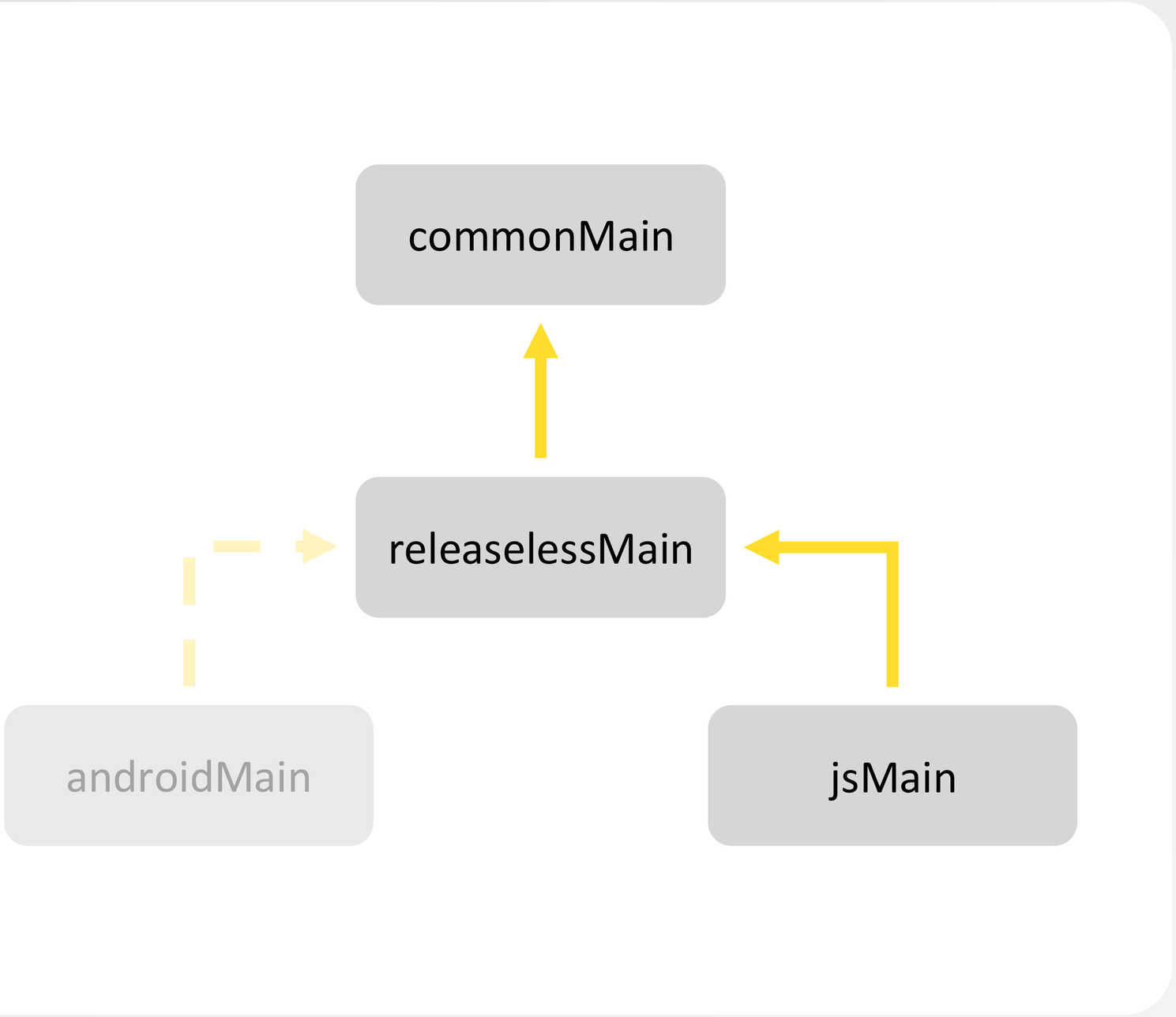
QuickJS – сильно урезан. Отладка кода. Решение

➜ Завели собственный Source Set



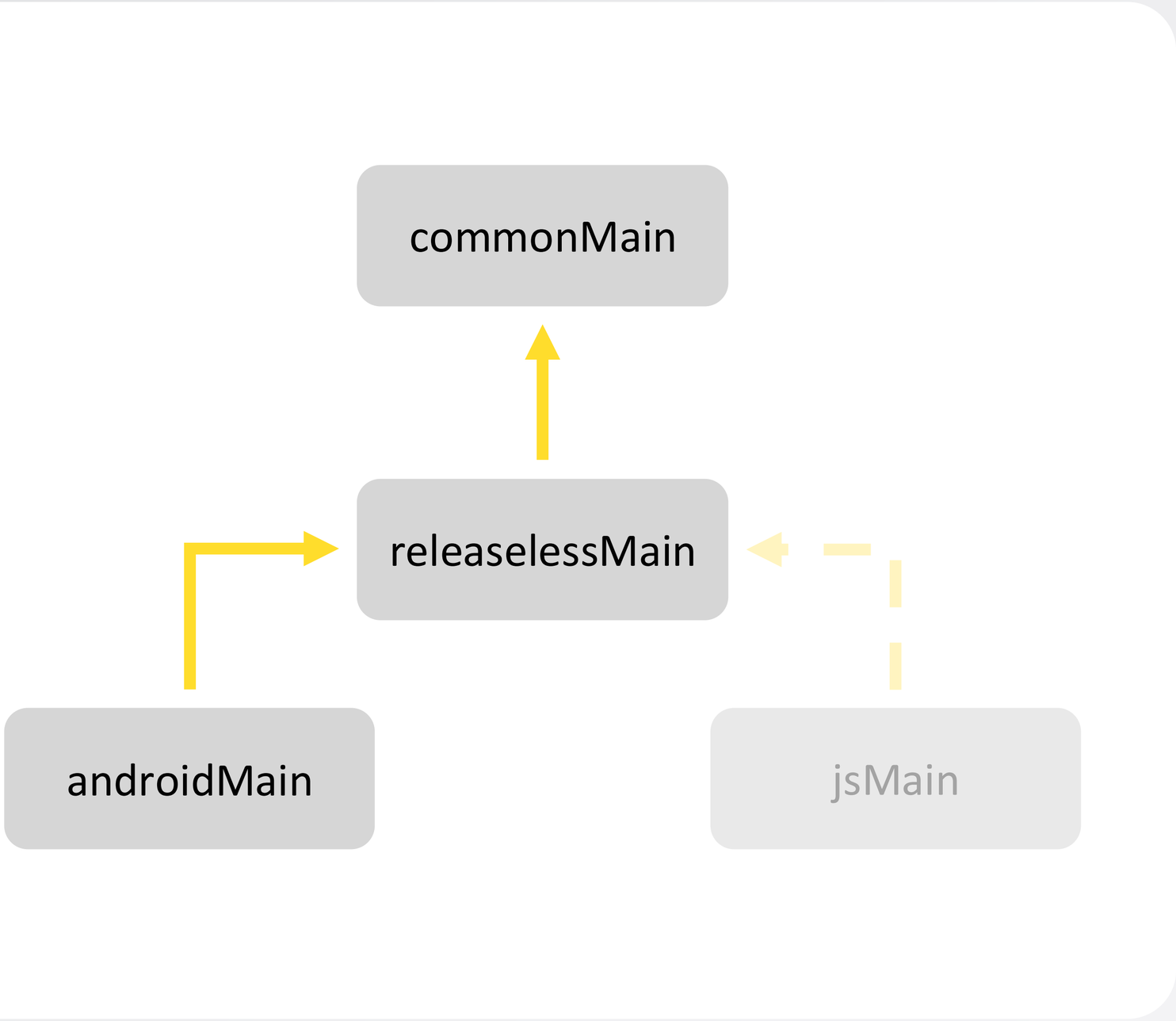
QuickJS – сильно урезан. Отладка кода. Решение

- Завели собственный Source Set
- Подключение к jsMain для работы с QuickJS

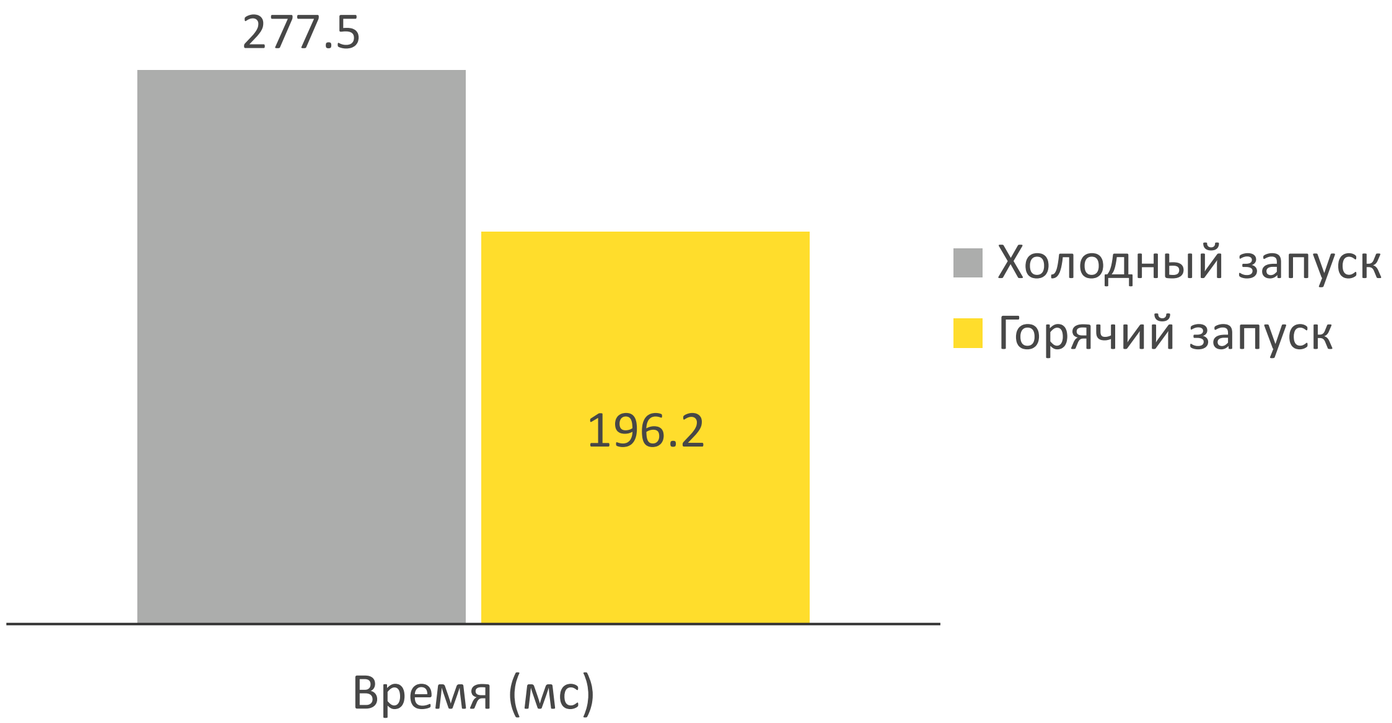


QuickJS – сильно урезан. Отладка кода. Решение

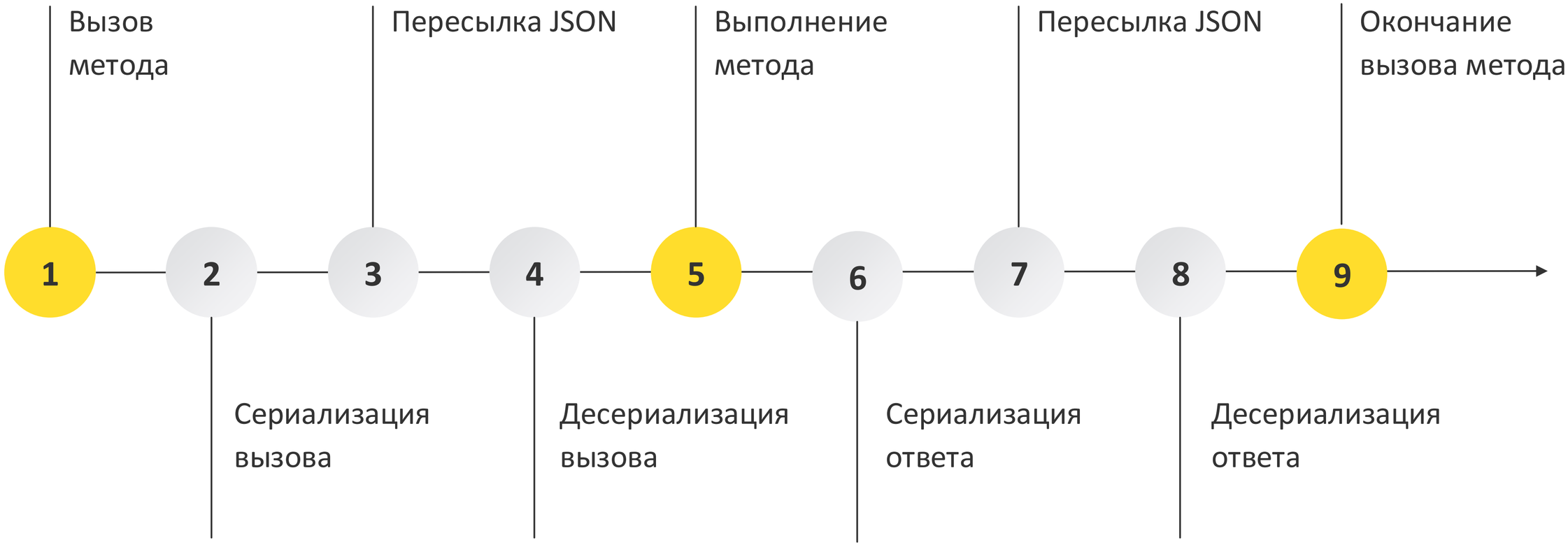
- Завели собственный Source Set
- Подключение к jsMain для работы с QuickJS
- Подключение к androidMain для работы с Android



Производительность. Инициализация QuickJS



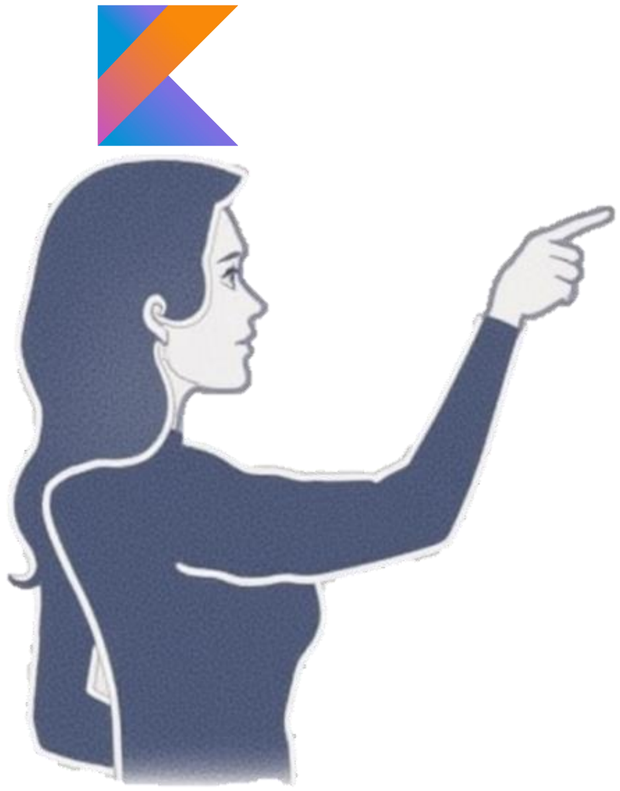
Производительность.
JSON как промежуточный формат



Производительность.
JSON как промежуточный формат. Советы

- Проектировать модели данных
- Писать собственные сериализаторы
- Оптимизировать структуру JSON

Числа в Kotlin и JavaScript



Int
UInt
Short
UShort
Double
Float

Number

Числа в Kotlin и JavaScript

mainJs.kt

```
fun main() {
  val double: Double = 1.0

  println(double is Int)    // ???
  println(double is Float)  // ???
}
```

Числа в Kotlin и JavaScript

mainJs.kt

```
fun main() {
  val double: Double = 1.0

  println(double is Int)    // true
  println(double is Float)  // true
}
```

Числа в Kotlin и JavaScript

mainJs.js

```
function main() {
  var double = 1.0;
  println(typeof double === 'number');
  println(typeof double === 'number');
}
```

Числа в Kotlin и JavaScript. Long

```
Caused by: app.cash.zipline.QuickJsException:
10 can't be deserialized to Long due to a potential precision loss
    at captureStack (js/runtime/coreRuntime.kt:48)
    at SerializationException_init_$Create$_0 (kotlinx-serialization-kotlinx-serialization-core.js)
    at <anonymous> (kotlinx/serialization/json/internal/DynamicDecoders.kt:159)
    ...
```

Числа в Kotlin и JavaScript. Long

Long

$2^{63}-1$

Number

$2^{53}-1$

Числа в Kotlin и JavaScript. Long

kotlinx-serialization-core / kotlinx.serialization.builtins / LongAsStringSerializer

LongAsStringSerializer

```
object LongAsStringSerializer : KSerializer<Long>                    (source)
```

Serializer that encodes and decodes Long as its string representation.

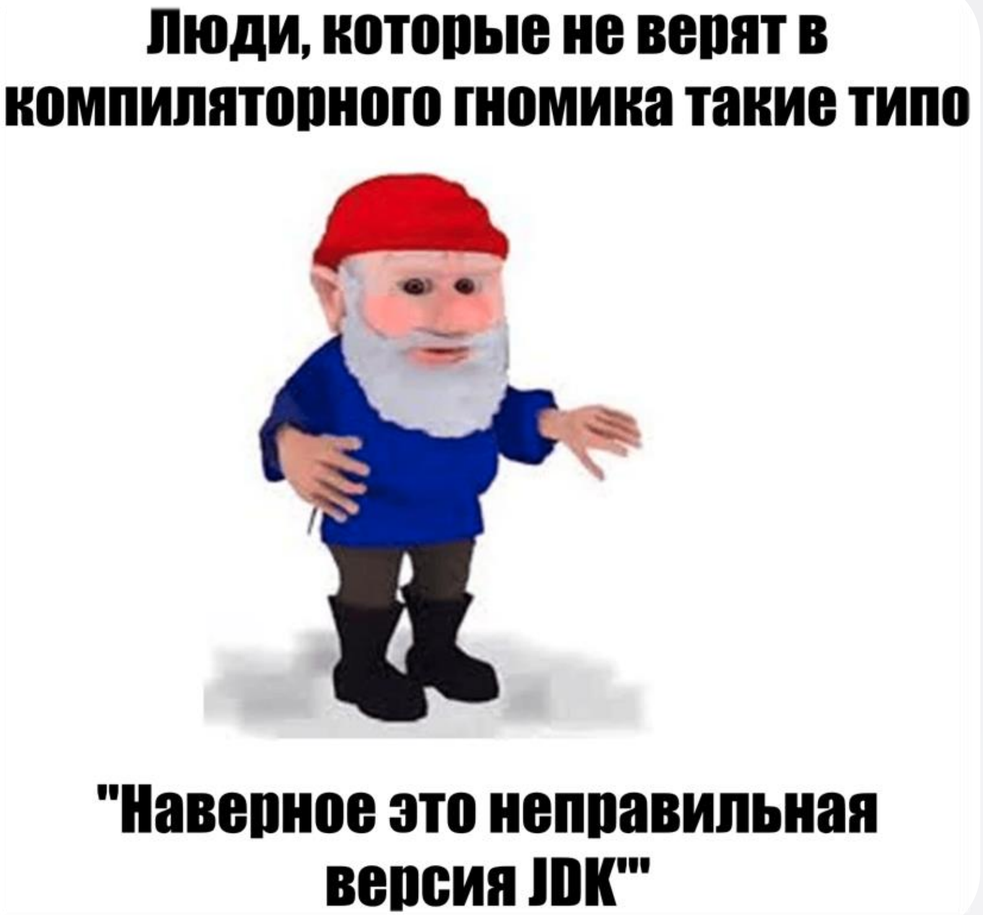
Intended to be used for interoperability with external clients (mainly JavaScript ones), where numbers can't be parsed correctly if they exceed `abs(2^53-1)`.

Members | Members & Extensions

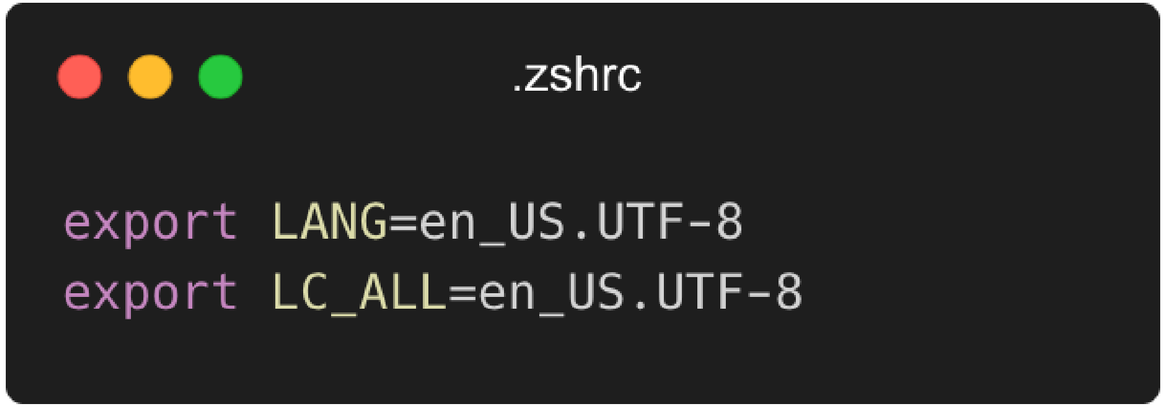
* Источник: https://kotlinlang.org

Компиляция в Bytecode

- Воспроизвелось только у 3 разработчиков
- Зависание при работе с корутинами в JS
- В коде JS разный порядок методов
- Небольшие отличия в байткоде



Компиляция в Bytecode. Решение



KMP в iOS. Размеры

- Kotlin-stdlib
- Базовые kotlin зависимости
- Дубликаты моделей
- Kotlin-swift адаптеры

KMP в iOS. Стирание типов

➜ Протоколы теряют Generic типы

Repository.kt

```kotlin
interface Repository<T> {
  fun fetch(): T
}
```

Repository.swift

```swift
protocol Repository {
    func fetch() -> Any?
}
```

KMP в iOS. Стирание типов

- Протоколы теряют Generic типы
- Классы частично поддерживают Generics

Box.kt

```kotlin
class Box<T: Number>(private val value: T) {
    fun unwrap(): T = value
}
```

Box.swift

```swift
class Box<T: AnyObject> {
    let value: T
    func unwrap() -> T { value }
}
```

KMP в iOS. DevEx

- Имена теряют исходный вид
- Enum всегда требуют default
- Работа с Flow только через Kotlin
- Sealed class не имеют enum семантику
- Краши из-за неверных приведений
- Проблемы с многопоточностью

TOUCHLAB

KMP в iOS. Бинарный файл

- Objective-C добавляет информацию в специальные секции
- Kotlin/Native хранит собственные runtime-метаданные

```
Contents of (__DATA,__objc_classlist) section
0x100008008  _OBJC_CLASS_$_ZiplineTask
0x100008020  _OBJC_CLASS_$_ZiplineDatabase
0x100008038  _OBJC_CLASS_$_ZiplinePreferences

__OBJC_CLASS_RO_$_ZiplineTask:
    superclass _OBJC_CLASS_$_NSObject
          name ZiplineTask
   baseMethods (16 methods)
         0x100004400 initWithTitle:dueDate:priority:
         0x100004420 setTitle:
         0x100004440 title
         0x100004520 setCompleted:
         0x100004540 isCompleted
         0x100004560 isOverdue
         0x100004580 validate
         ivars (7 ivars)
         0x1000045a0 _taskID @"NSString"
         0x1000045c0 _title @"NSString"
         0x1000045e0 _taskDescription @"NSString"
         0x100004600 _dueDate @"NSDate"
         0x100004620 _priority q
         0x100004640 _completed B
         0x100004660 _category @"ZiplineCategory"
```

KMP в iOS. Бинарный файл

- @HidesFromObjC и @HiddenFromObjC
- @ObjCEnum и @ObjCName
- @ExportObjCClass
- ExplicitApiMode

KMP в iOS. Решение

- Написали свой аналог Zipline на Swift
- Генерируем протоколы ZiplineService на Swift

- Контроль кодовой базы
- Честная генерация swift кода
- Уменьшили размер фреймворков
- Технология не оставляет следов в бинаре

Итоги

- Предпосылки
- Zipline и Kotlin/JS
- Интеграция с SDUI
- Проблемы, с которыми столкнулись

Выводы

- KMP это не только про iOS и Android
- Zipline – инструмент, которым можно пользоваться в проде
- У Zipline есть проблемы, но они решаемы
- Из любого SDUI можно сделать полностью безрелизный экран

Вопросы и ответы

Тимур Валиев

Android разработчик
t.valiev@tbank.ru
@teemch

Максим Вакула

iOS разработчик
m.n.vakula@tbank.ru
@VakulaMaksim

Спасибо за внимание