USETECH

Kotlin-Swift-Wasm. Мы пойдем другим путем

Анна Жаркова
Lead Mobile developer

Обо мне



- В мобильной разработке с 2013
- Ведущий мобильный разработчик в Usetech
- Нативная разработка под iOS и Android (Swift/Objective-C, Kotlin/Java) кросс-платформа (Xamarin, Kotlin multiplatform)
- Ментор, управляю командой направления
- Спикер на конференциях AppsConf, Mobius, TechTrain, DroidCon (2022)
- Преподаватель в Otus (iOS Pro и базовый)
- Автор статей по мобильной разработке (SwiftUI, iOS, KMP)
- Автор книги «Kotlin Multiplatform на практике» (2 издание в работе)

Обсудим:

- Web + Mobile, как тренд
- Swift-Wasm, концепция, что под капотом
- Смотрим на примере приложения iOS (PWA), архитектура
- Ограничения и возможности. Работаем с памятью и многопоточностью
- Работаем с UI
- Kotlin-Wasm. Подходы к разработке PWA
- Flutter Web

Тенденции грядущего не-ИИ

- Swift везде и для всего
- Web + Mobile



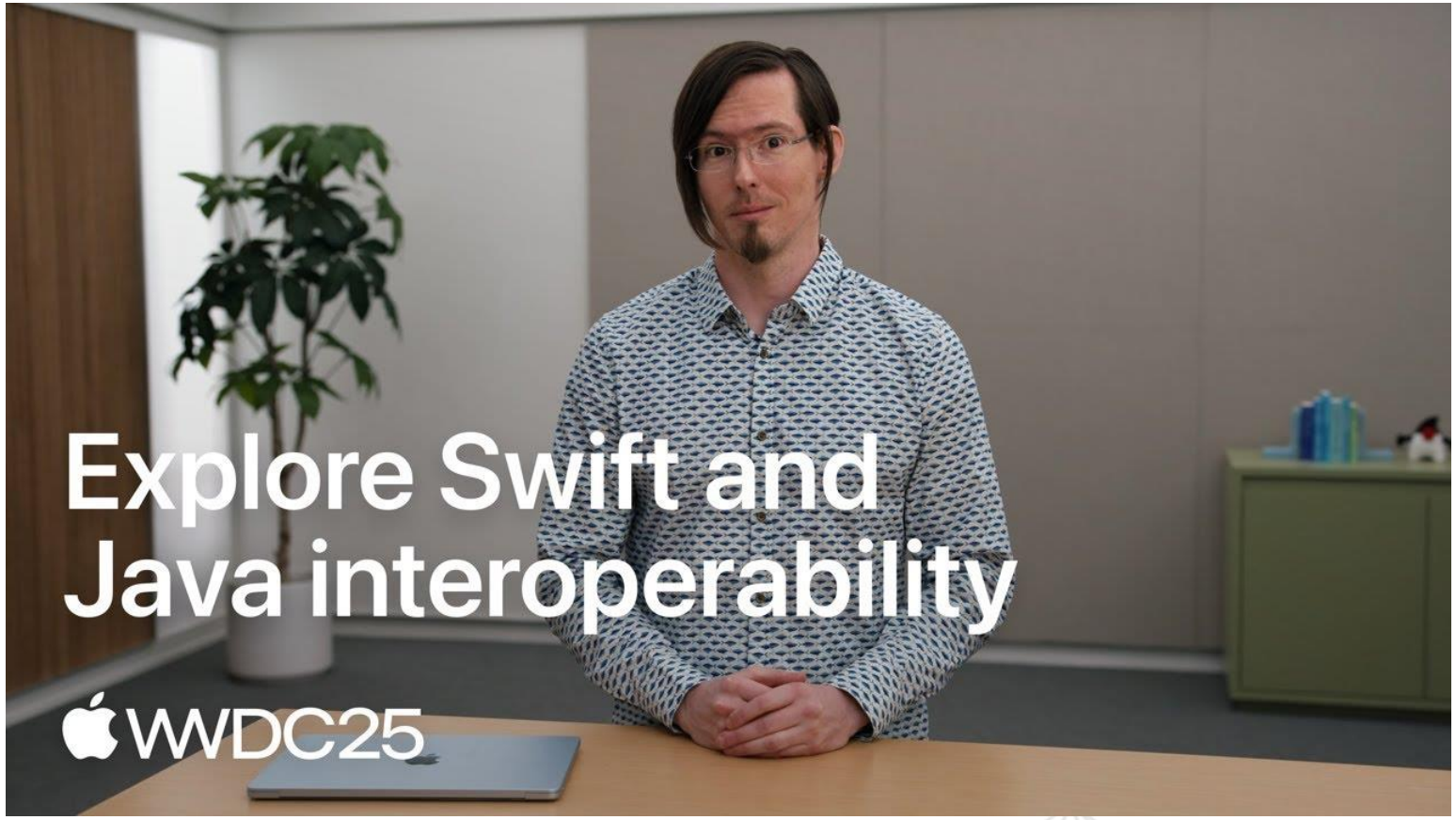


Swift везде и для всего



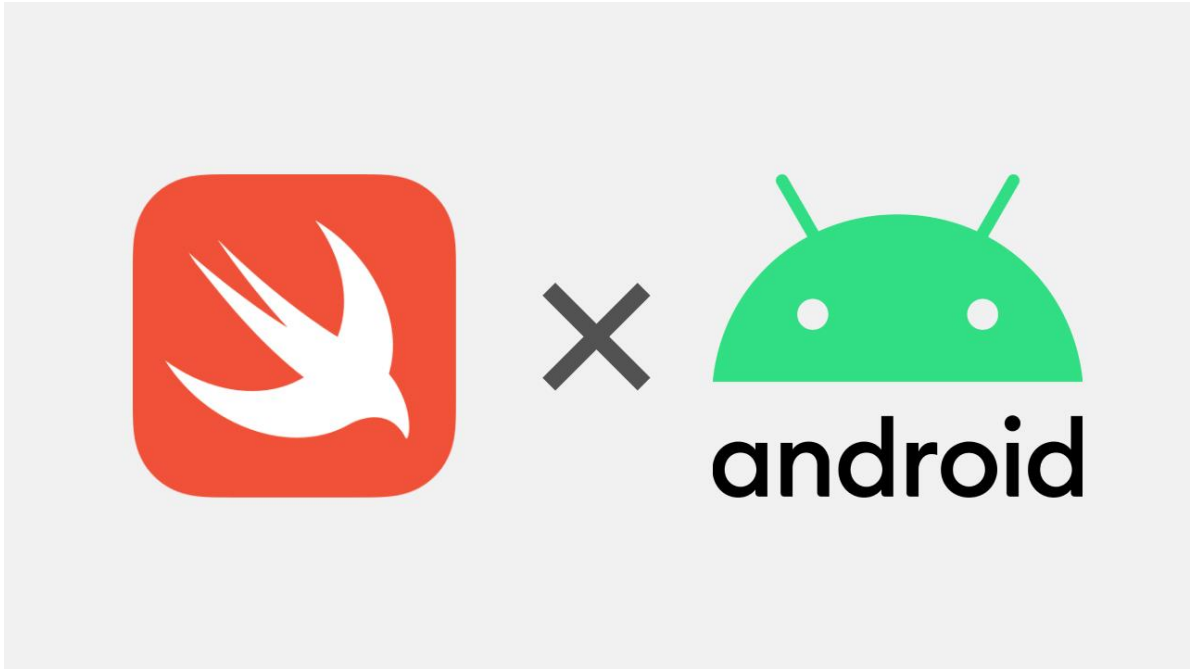
Swift-Java Interoperability. WWDC 2025

https://github.com/swiftlang/swift-java



Swift 6.3 Официальный релиз SDK для Android

https://www.swift.org/documentation/articles/swift-sdk-for-android-getting-started.html



Swift -> Wasm (WebAssembly)



(разработка не менее 6 лет)



Swift -> Wasm. WWDC 2025-2026

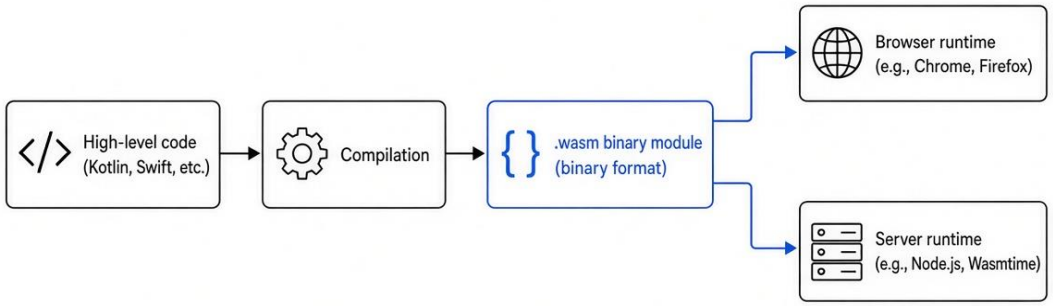
Официальный SDK

- встроенный WasmKit (среда выполнения)
- нативная поддержка WASI потоков
- официальный интероп с JS через BridgeJs, JavaScriptKit и т.п



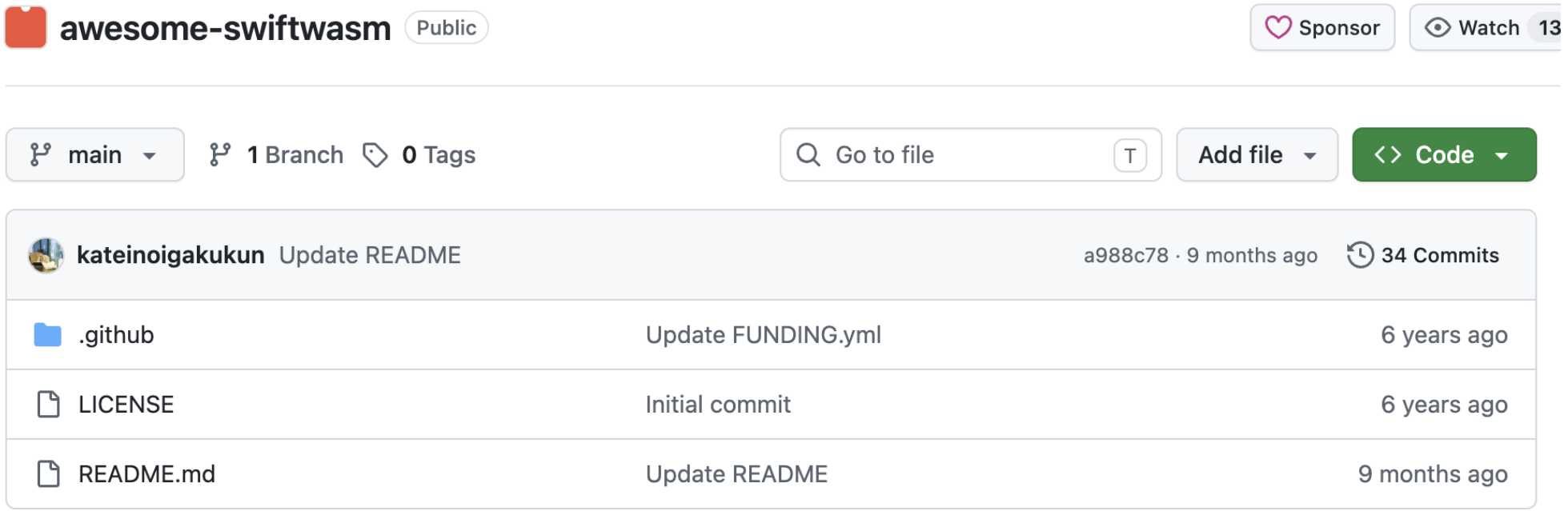
Swift -> Wasm. Что такое Wasm

WebAssembly (Wasm) — это низкоуровневый бинарный формат инструкций, предназначенный для выполнения в браузере и на сервере с производительностью, близкой к нативной.



Swift -> Wasm. WWDC 2025-2026

Разработка не менее 6 лет
SwiftWasm community



Swift -> Wasm. Agent Skill

https://github.com/SwiftWasm/Swift-Wasm-Agent-Skill

SwiftWasm Agent Skills

A collection of Agent Skills designed to help AI coding agents build and maintain Swift applications targeting WebAssembly.

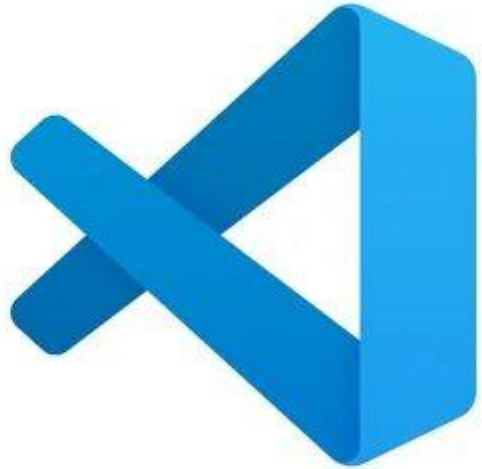
What are Agent Skills?

Agent Skills are specialized modules containing instructions, scripts, and documentation that give AI agents (like Claude, Gemini, or Codex) the domain expertise needed to perform complex tasks.

This repository is compatible with major agent tools, leveraging standardized formats to ensure your agent has the right context for SwiftWasm development.

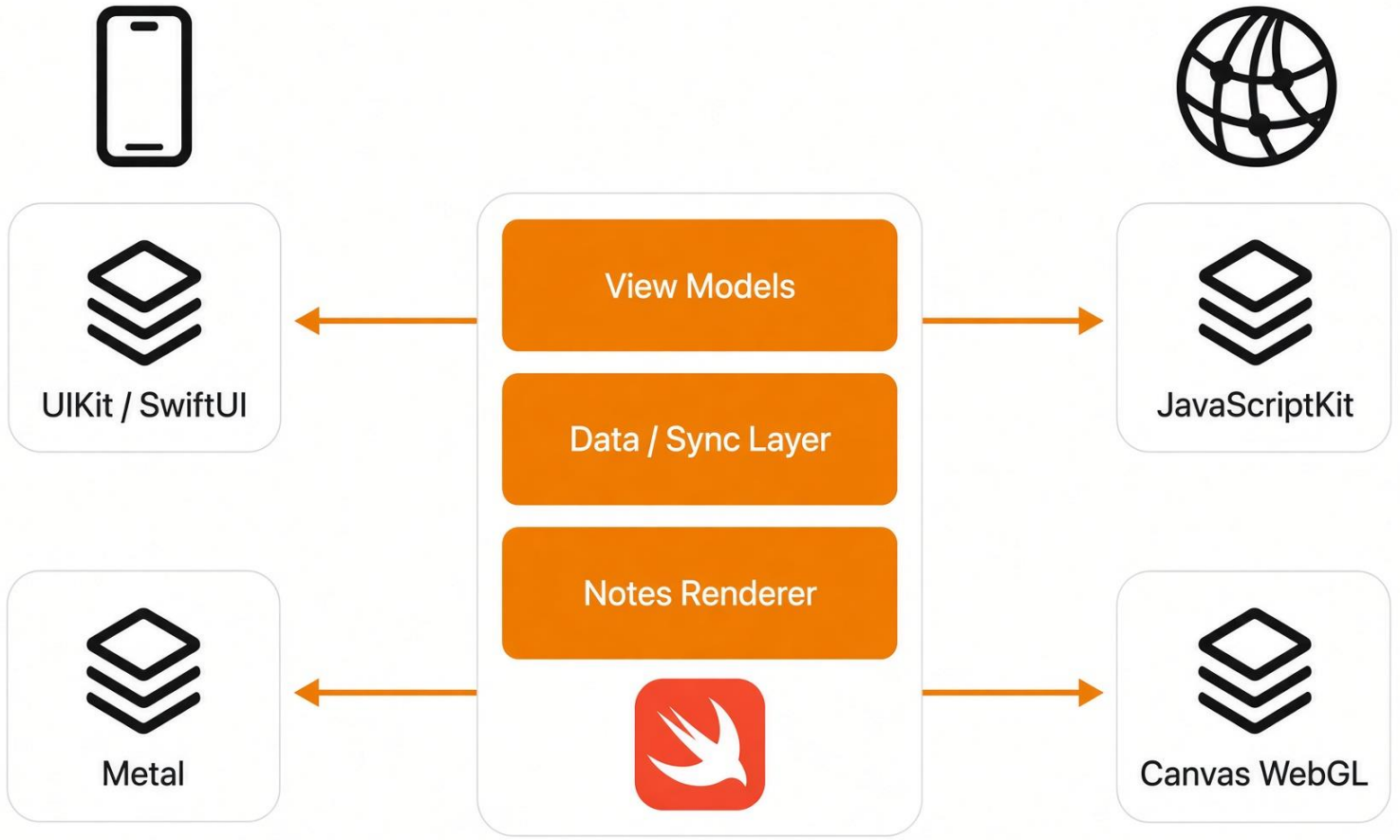
SwiftSDK. WebAssembly

- WasmKit – среда выполнения на Swift
- JavaScriptKit – для взаимодействия и Interop Swift-JavaScript

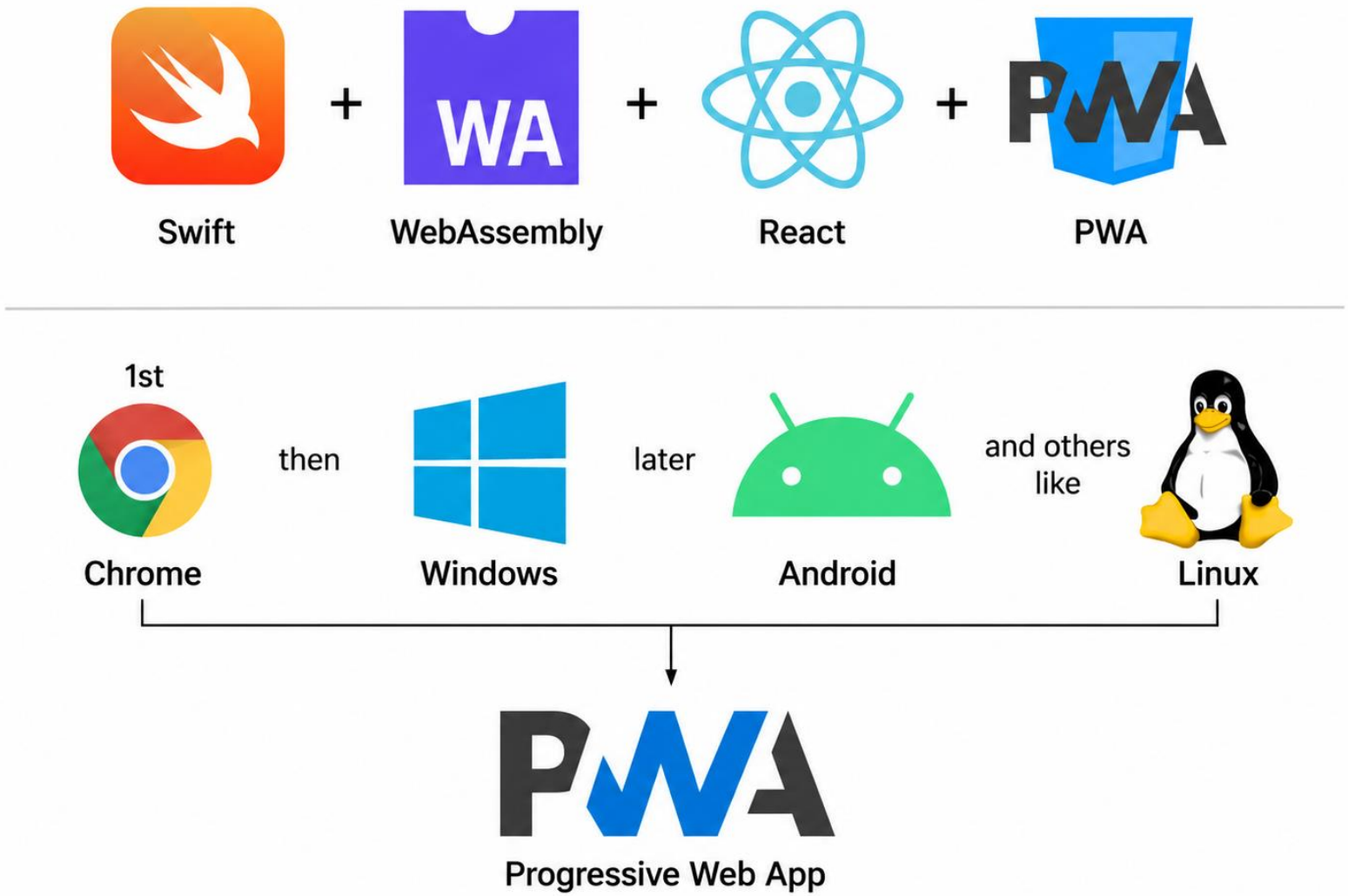


Кейс от Apple. GoodNotes

Миграция приложения



Кейс от Apple. GoodNotes



Кейс от Apple. GoodNotes. Польза от SwiftWasm

- Повторное использование более 100 тысяч строк кода на Swift.
- Поддержка разработки основного продукта + одновременно кроссплатформа
- Быстрый выход на платформу с использованием итеративного процесса разработки
- Нет дублирования бизнес-логики
- Поддержка использования возможностей платформы и улучшения производительности

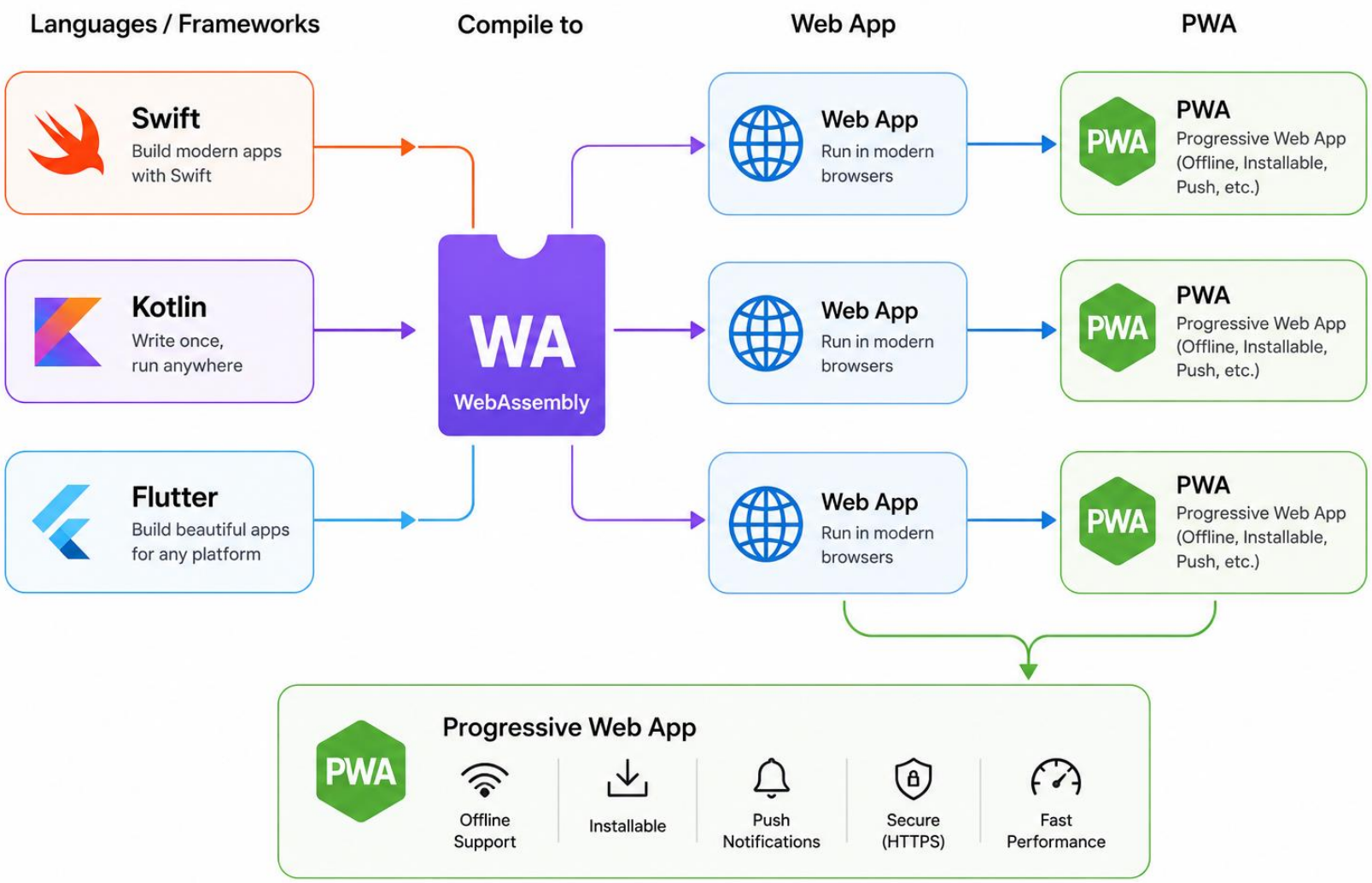
SwiftWasm. Для чего же

- PWA на основе iOS Swift кода и поддерживаемой бизнес-логики
- Удобный синтаксис для библиотек для Web
- Адаптация библиотек (например, криптография, анимация, вычисления и т.п)
- Логика и механика игр

SwiftWasm. Для чего же

- **PWA на основе iOS Swift кода и поддерживаемой бизнес-логики**
- Удобный синтаксис для библиотек для Web
- Адаптация библиотек (например, криптография, анимация, вычисления и т.п)
- Логика и механика игр

Web + Mobile как общий тренд



Web + Mobile как общий тренд

ДОКЛАД 13.05 / 16:15 – 17:00 (UTC+7) Platforms

PWA вместо App Store: опыт замены нативного iOS-приложения и технические ограничения

Спикеры



Константин Шкурко
РСХБ.Цифра (РСХБ-Автоматизация)

Приглашенные эксперты



Юрий Дубовой
Fractalog

Web + Mobile как общий тренд

SwiftWasm – это не просто PWA

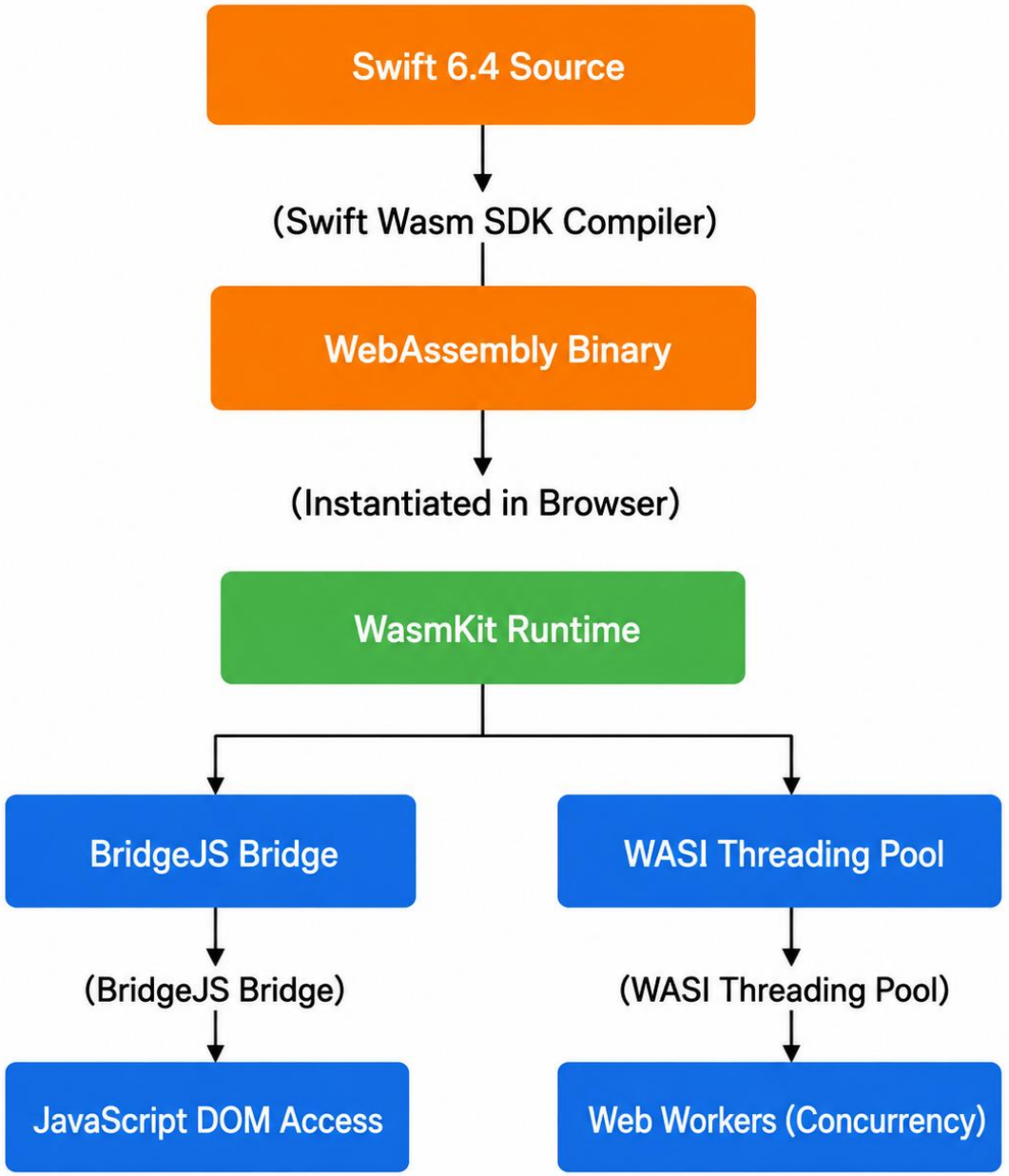
Это PWA с кодом Swift -> решение вопросов с релизами iOS (и даже без SDUI)



Swift-> Wasm,
как работает внутри

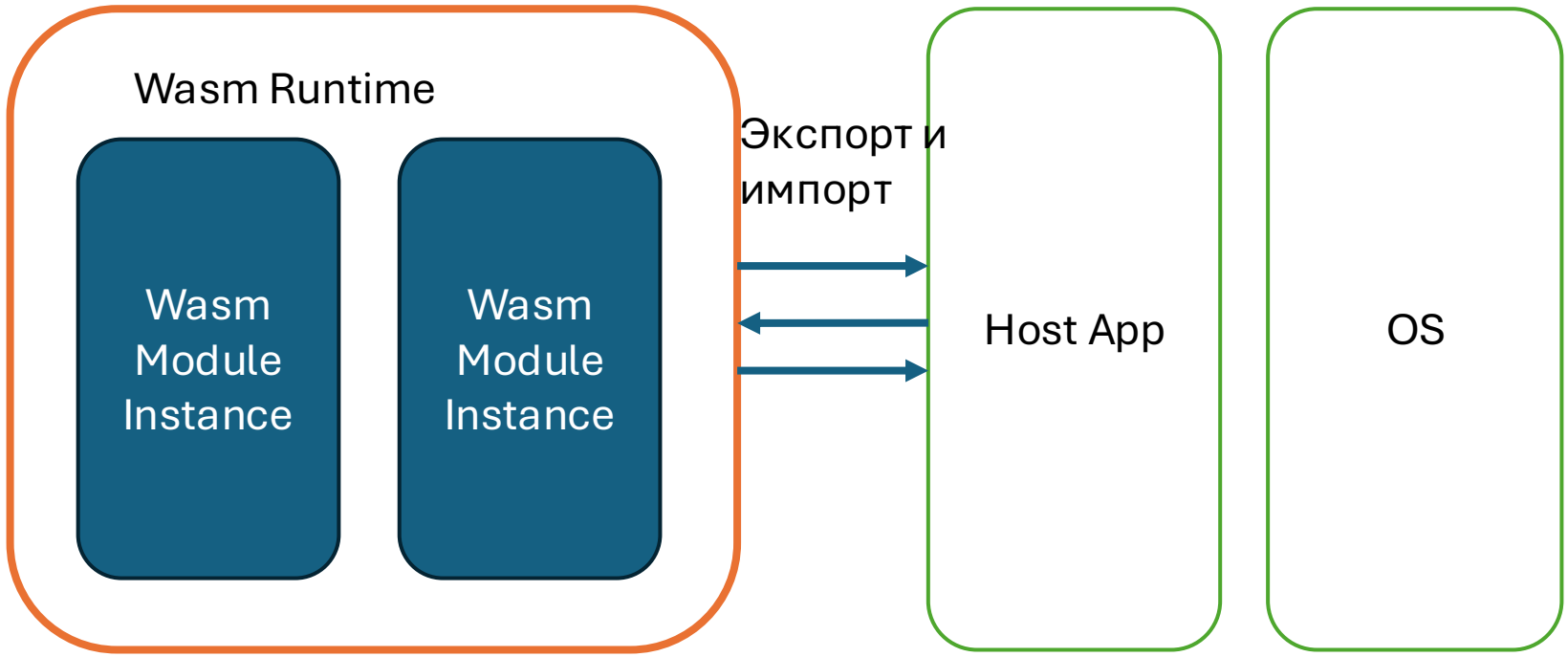
Swift -> Wasm. Что такое Wasm

Архитектура решения



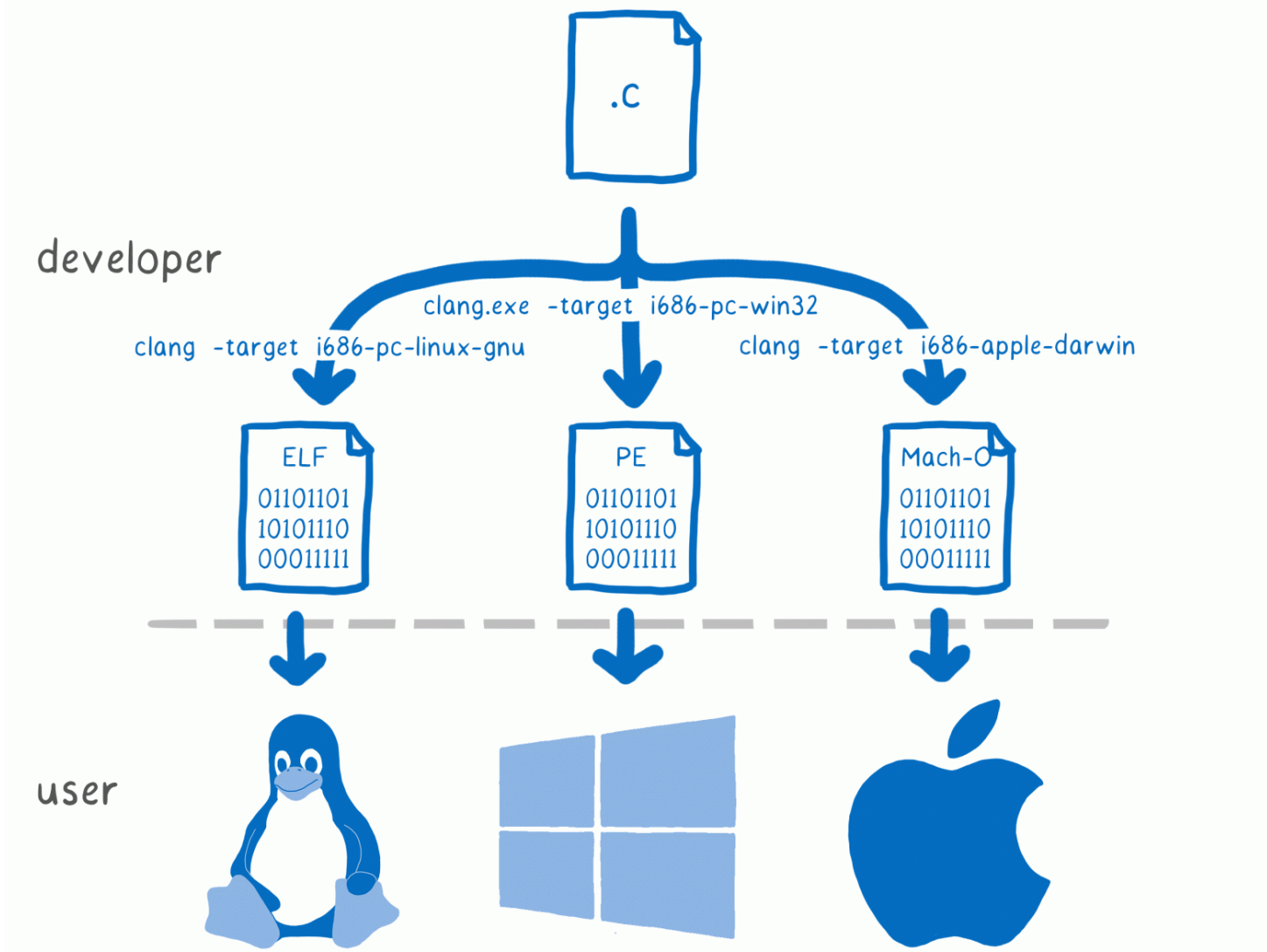
Swift -> Wasm

Модуль – наименьшая единица



Swift -> Wasm. WASI

WASI: это системный интерфейс для платформы WebAssembly для взаимодействия со средой хоста



Swift -> Wasm

- Wasm — это стековая машина, оперирующая числами и непрозрачными ссылками
- Линейная память используется для низкоуровневых операций с указателями и кучи
- Текстовый формат WebAssembly (.wat) используется для отладки
- Двоичные модули WebAssembly (.wasm) используются для распространения

Wasm: представление модуля

add.wat

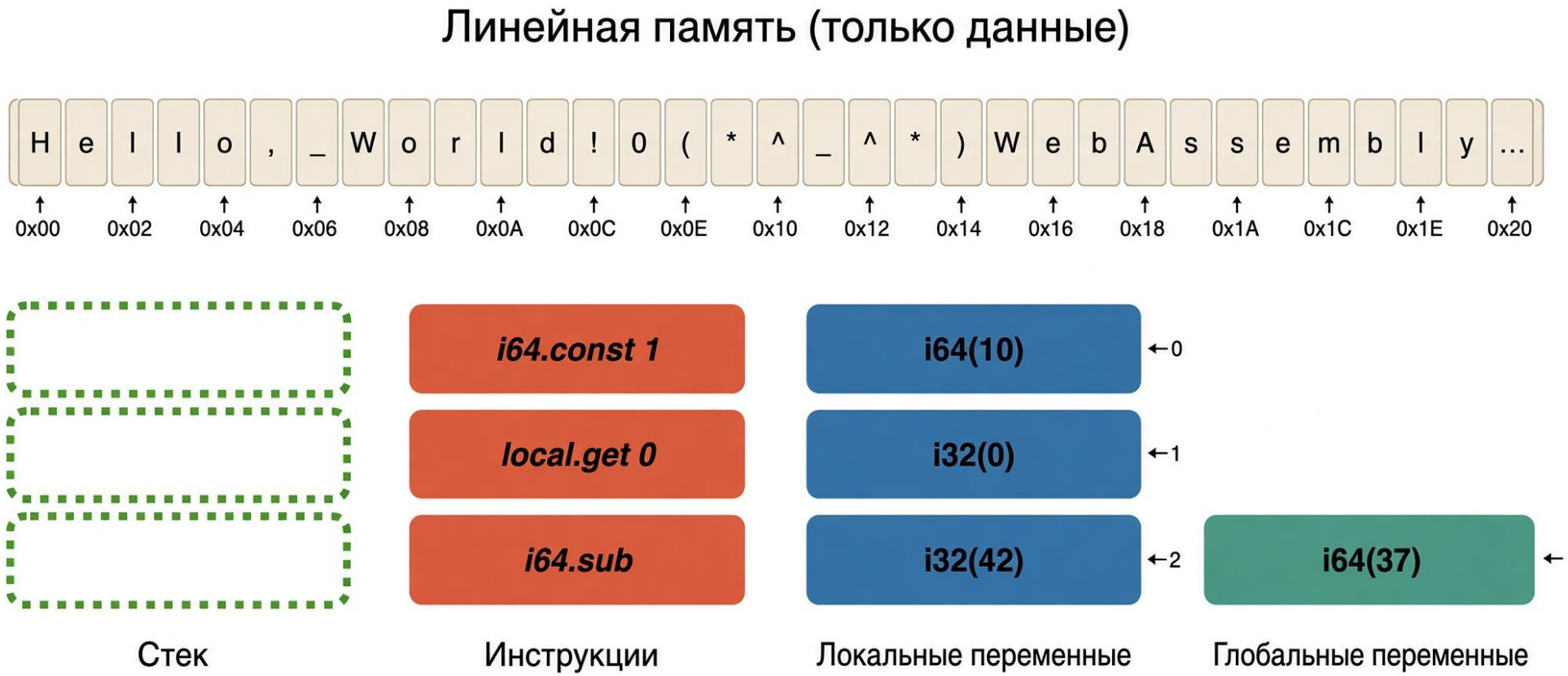
```
;; add.wat — текстовый формат (WAT)
(module
  (func $add (export "add")
    (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add)
)
```

.wasm — компактный бинарь для доставки и исполнения

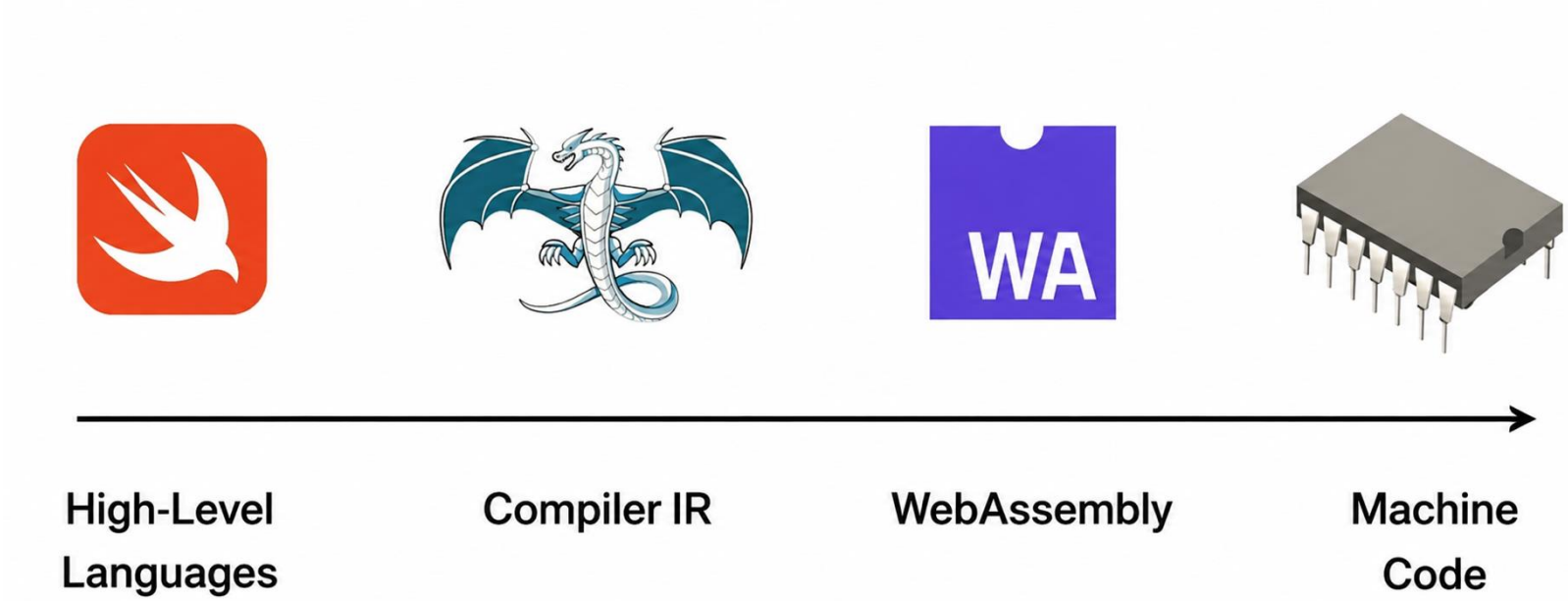
.wat — текстовое представление того же модуля: для отладки, чтения и спецификации

Модуль валидируется при загрузке: типы, границы памяти и сигнатуры проверяются до исполнения.

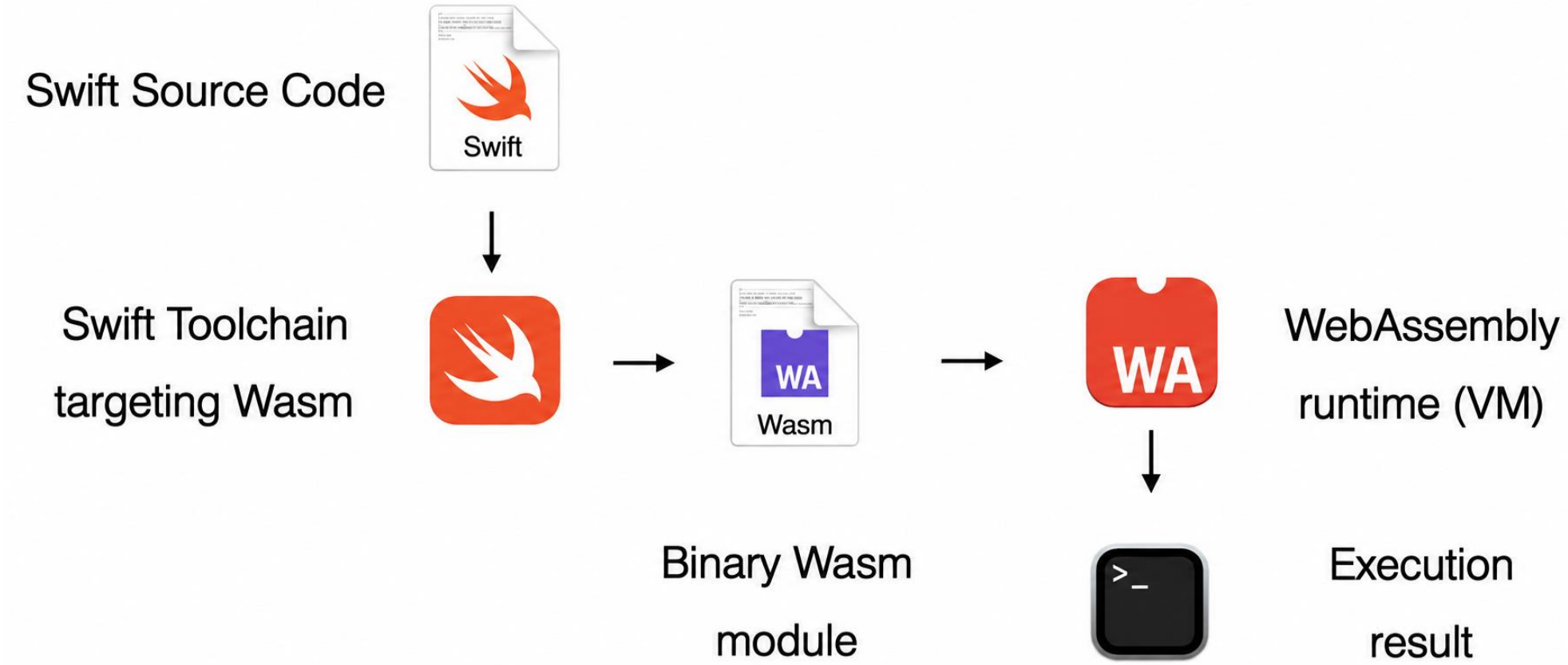
Компиляция WebAssembly



Компиляция Swift -> Wasm



Компиляция Swift -> Wasm



Wasm GC

До: Runtime GC языка среды

· собственный сборщик мусора внутри бинарника

· два GC рядом· мегабайты лишнего веса

· сложный интероп объектов

WasmGC:Объекты живут в куче движка

· структуры и массивы — типы WasmGC

· сборка мусора — задача браузера

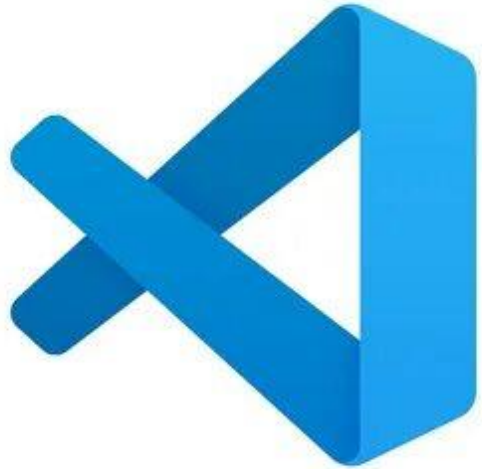
· компактные бинари

· естественный обмен объектами с JS

Swift-> Wasm,
подключаем и настраиваем

Swift WebAssembly. Установка и настройка

- Swift 6.3.3 (wasm и wasm-embedded)
- Xcode по умолчанию
- VSCode для работы



Swift WebAssembly. Установка и настройка

- https://www.swift.org/documentation/articles/wasm-getting-started.html

```
swiftly install 6.3.3

swiftly use 6.3.3

swift sdk install https://download.swift.org/swift-6.3.3-release/wasm-sdk/swift-6.3.3-RELEASE/swift-6.3.3-RELEASE_wasm.artifactbundle.tar.gz --checksum cabfa08b73bb8ac783927ecd15fa386e99d0c139c5f232445067bcf58379cae7
```

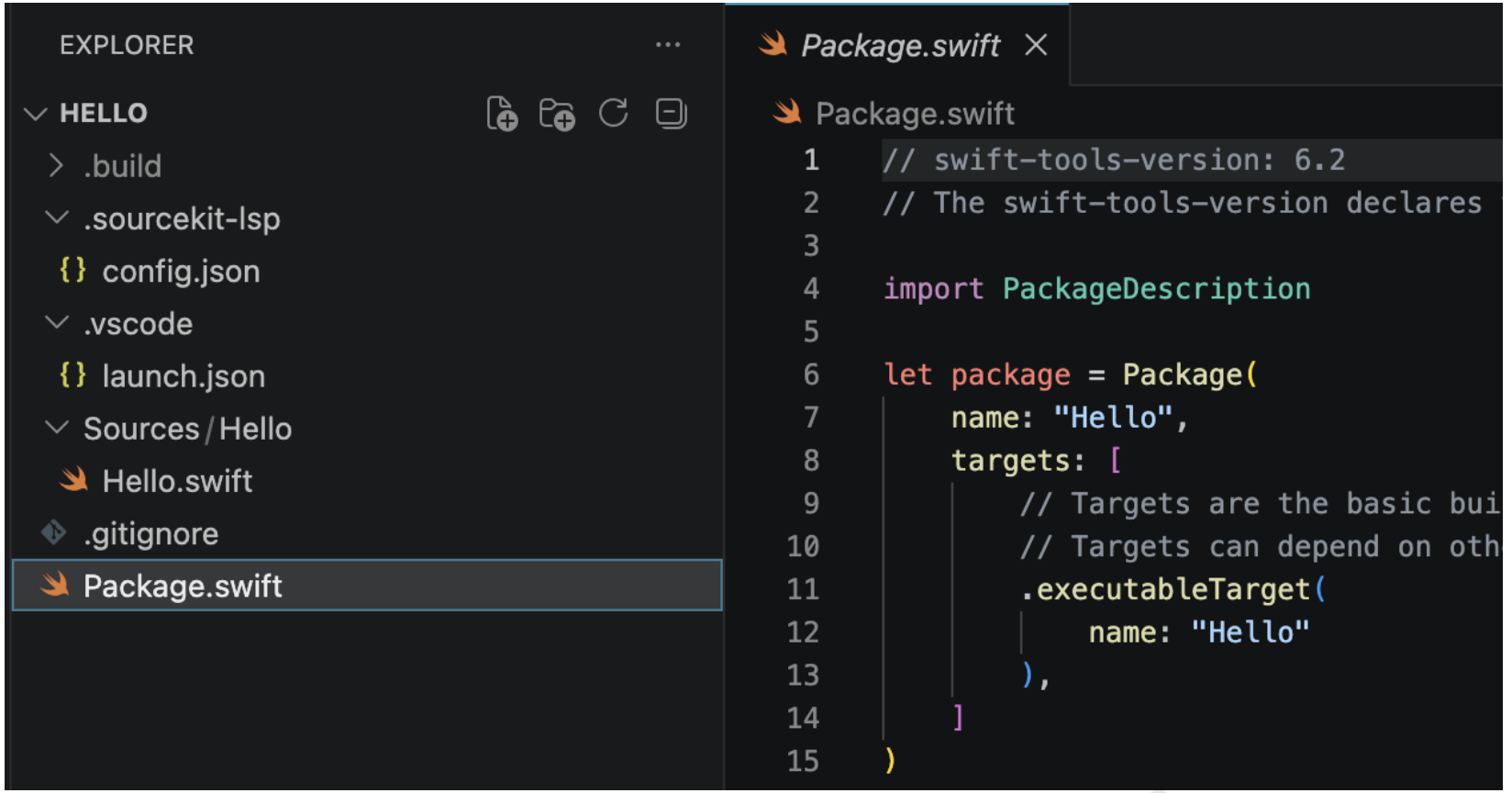
Swift WebAssembly. Установка и настройка

Тренируемся на базовом проекте

```
mkdir Hello
cd Hello
swift package init --type
executable
```

Swift WebAssembly. Установка и настройка

SPM



Swift WebAssembly. Установка и настройка

Sources/Hello/Hello.swift

```swift
@main
struct wasi_test {
    static func main() {
#if os(WASI)
        print("Hello from WASI!")
#else
        print("Hello from the host system!")
#endif
    }
}
```

Swift WebAssembly. Установка и настройка

Запуск

```
swift build --swift-sdk swift-6.3.3-RELEASE_wasm

swift run --swift-sdk swift-6.3.3-RELEASE_wasm
```

Swift WebAssembly. Установка и настройка

Результат

```
annazharkova@MacBook-Pro-Anna Hello % swift build --swift-sdk swift-6.3.3-
RELEASE_wasm
Building for debugging...
[8/8] Linking Hello.wasm
Build complete! (1.47s)
annazharkova@MacBook-Pro-Anna Hello % swift run --swift-sdk swift-6.3.3-RELEASE_wasm
[1/1] Planning build
Building for debugging...
[1/1] Write swift-version--7B1FAEB463ACD8E8.txt
Build of product 'Hello' complete! (0.07s)
Hello, world!
annazharkova@MacBook-Pro-Anna Hello %
```

Swift WebAssembly. Установка и настройка

Для поддержки VSCode

```
{
    "swiftPM": {
        "swiftSDK": "swift-6.4.1-RELEASE_wasm"
    }
}
```

Swift WebAssembly. Swift -> JS

BridgeJs

```
import JavaScriptKit

@JS class Greeter {
    @JS var name: String

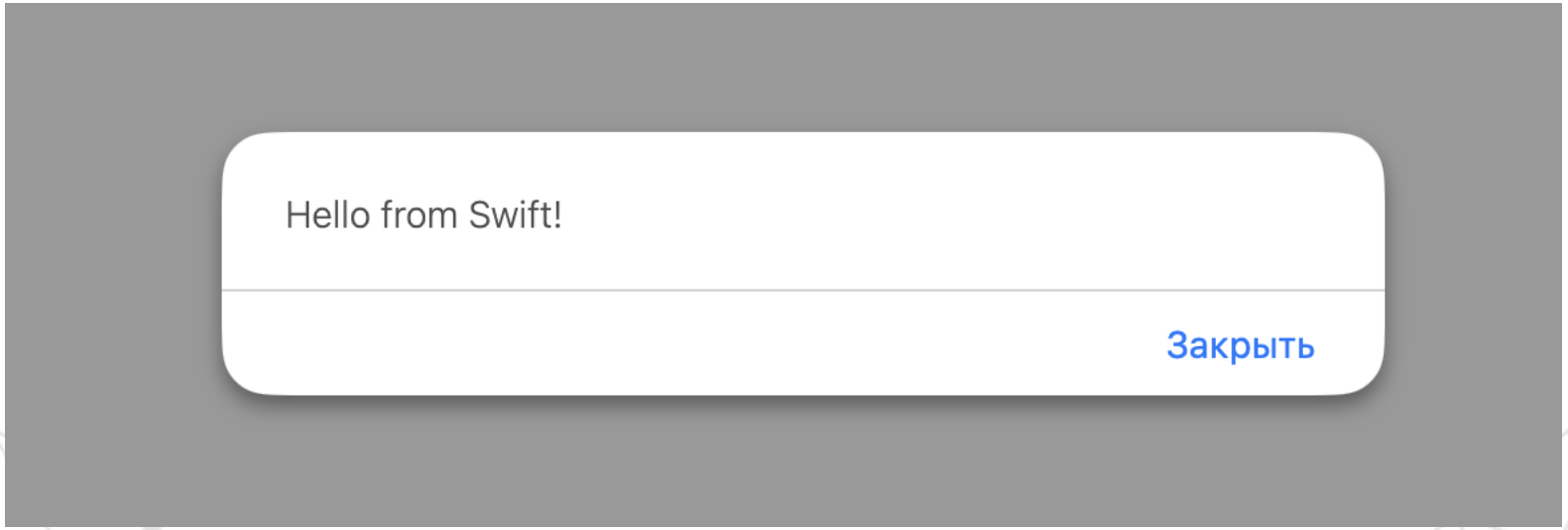
    @JS init(name: String) {
        self.name = name
    }

    @JS func greet() -> String {
        "Hello, \(name)!"
    }
}
```

Swift WebAssembly. Простой пример с UI

- JavaScriptKit

Click me!



Wasm vs JS

Wasm не заменяет JavaScript: у модуля нет прямого доступа к DOM, сети и Web API.

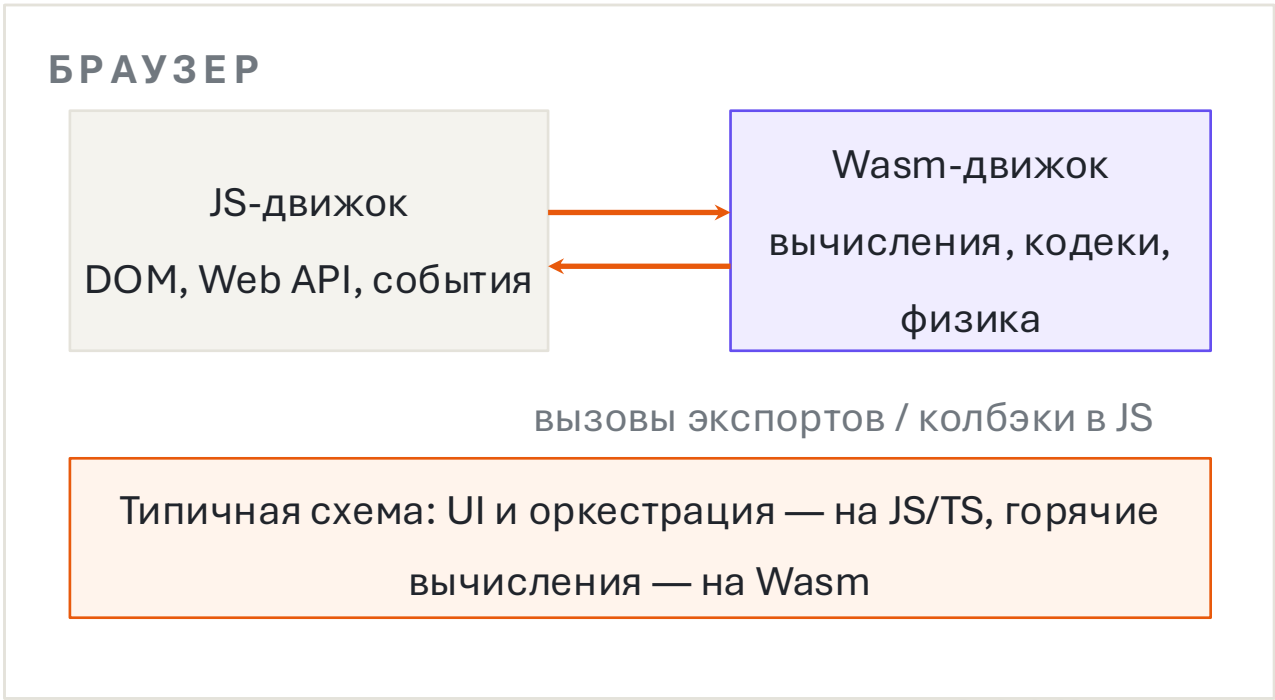
JS:

DOM и все Web API

Загрузка и инстанцирование модуля

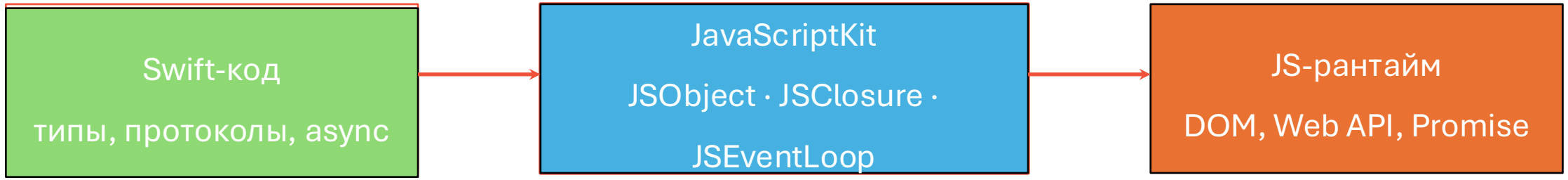
Маршалинг данных

Wasm - CPU-bound код



JavaScriptKit

JavaScriptKit — типобезопасный мост: JS-объекты становятся Swift-значениями.



JSObject	динамический доступ к любому JS-объекту: свойства, методы, конструкторы
JSClosure	замыкания Swift как JS-колбэки
JavaScriptEventLoop	обмен буферами между линейной памятью и JS без лишних копий
TypedArray и мосты	исполнение Swift-задач поверх event loop браузера: async/await

WasmKit

WasmKit — интерпретатор WebAssembly, написанный на чистом Swift

- Интерпретатор, не JIT
- WASI из коробки
- Встраивание Wasm-модуля через SwiftPM
- swift run для запуска WasmKit

Swift WebAssembly. Простой пример с UI

- JavaScriptKit

```
import PackageDescription

let package = Package(
    name: "Hello",
    dependencies: [
        .package(url: "https://github.com/swiftwasm/JavaScriptKit.git", exact: "0.57.0"),
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module or a test suite.
        // Targets can depend on other targets in this package and products from dependencies.
        .executableTarget(
            name: "Hello",
            dependencies: [
                .product(name: "JavaScriptKit", package: "JavaScriptKit"),
            ]
        ),
        .testTarget(
            name: "HelloTests",
            dependencies: ["Hello"]
        ),
    ],
    swiftLanguageModes: [.v6]
)
```

USETECH

Swift WebAssembly. Простой пример с UI

- JSObject
- Выводим алерт по кнопке

```swift
import JavaScriptKit

// Access the global JavaScript object
let global = JSObject.global

// Get a reference to the document
let document = global.document

// Create a new HTML element
var button = document.createElement("button")
button.innerText = "Click me!"

// Add click handler using the recommended JavaScriptKit pattern
button.onclick = .object(JSClosure { _ in
    _ = global.alert!("Hello from Swift!")
    return .undefined
})

// Append to body
_ = document.body.appendChild(button)
```

USETECH

Swift WebAssembly. Простой пример с UI

- Входная точка js из PackageToJs вывода

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Hello SwiftWasm</title>
</head>
<body>
  <script type="module">
    import { init } from "./.build/plugins/PackageToJS/outputs/Package/index.js";
    init();
  </script>
</body>
</html>
```

Swift WebAssembly. Простой пример с UI

- Входная точка js из PackageToJs вывода

```
// @ts-check
import { instantiate } from './instantiate.js';
import { defaultBrowserSetup } from './platforms/browser.js';

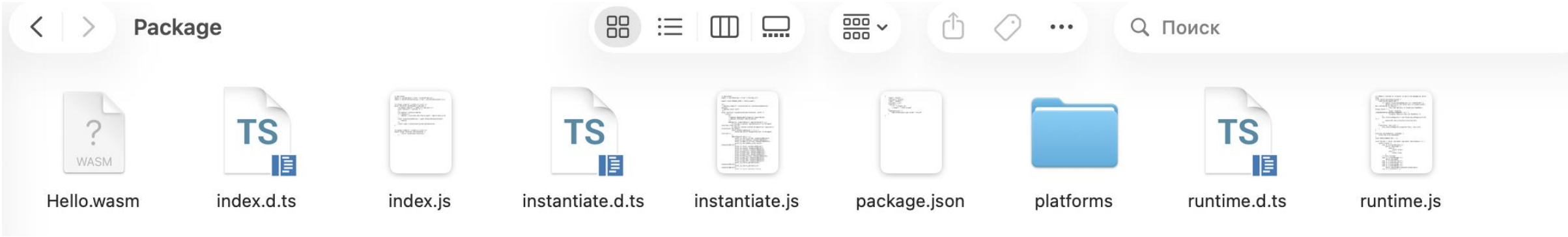

/** @type {import('./index.d').init} */
async function initBrowser(_options) {
    /** @type {import('./index.d').Options} */
    const options = _options || {
    };
    let module = options.module;
    if (!module) {
        module = fetch(new URL("Hello.wasm", import.meta.url))
    }
    const instantiateOptions = await defaultBrowserSetup({
        module,
    })
    return await instantiate(instantiateOptions);
}


/** @type {import('./index.d').init} */
export async function init(options) {
        return initBrowser(options);
    }
```

USETECH

Swift WebAssembly. Простой пример с UI

- Входная точка js из PackageToJs вывода



Swift WebAssembly. Общий модуль

Базовая структура
общего модуля



Swift WebAssembly. Общий модуль

Общий модуль кода для
iOS и Wasm

```
// Package.swift
let package = Package(
    name: "SharedLogic",
    platforms: [
        .iOS(.v14),
        .macOS(.v11)
    ],
    products: [
        .library(name: "SharedLogic", targets:
["SharedLogic"]),
    ],
    targets: [
        .target(name: "SharedLogic"),
        .testTarget(name: "SharedLogicTests", dependencies:
["SharedLogic"]),
    ]
)
```

Swift WebAssembly. Общий модуль

- JavaScriptKit
- JavaScriptEventLoop
- BridgeJs

```
// swift-tools-version: 6.3

import PackageDescription

let package = Package(
    name: "swift-wasm-news",
    platforms: [
        .macOS(.v14)
    ],
    products: [
        .executable(
            name: "NewsWasmClient", targets: ["NewsWasmClient"]
        )
    ],
    dependencies: [
        .package(
            url: "https://github.com/swiftwasm/JavaScriptKit.git",
            exact: "0.53.0"
        )
    ],
```

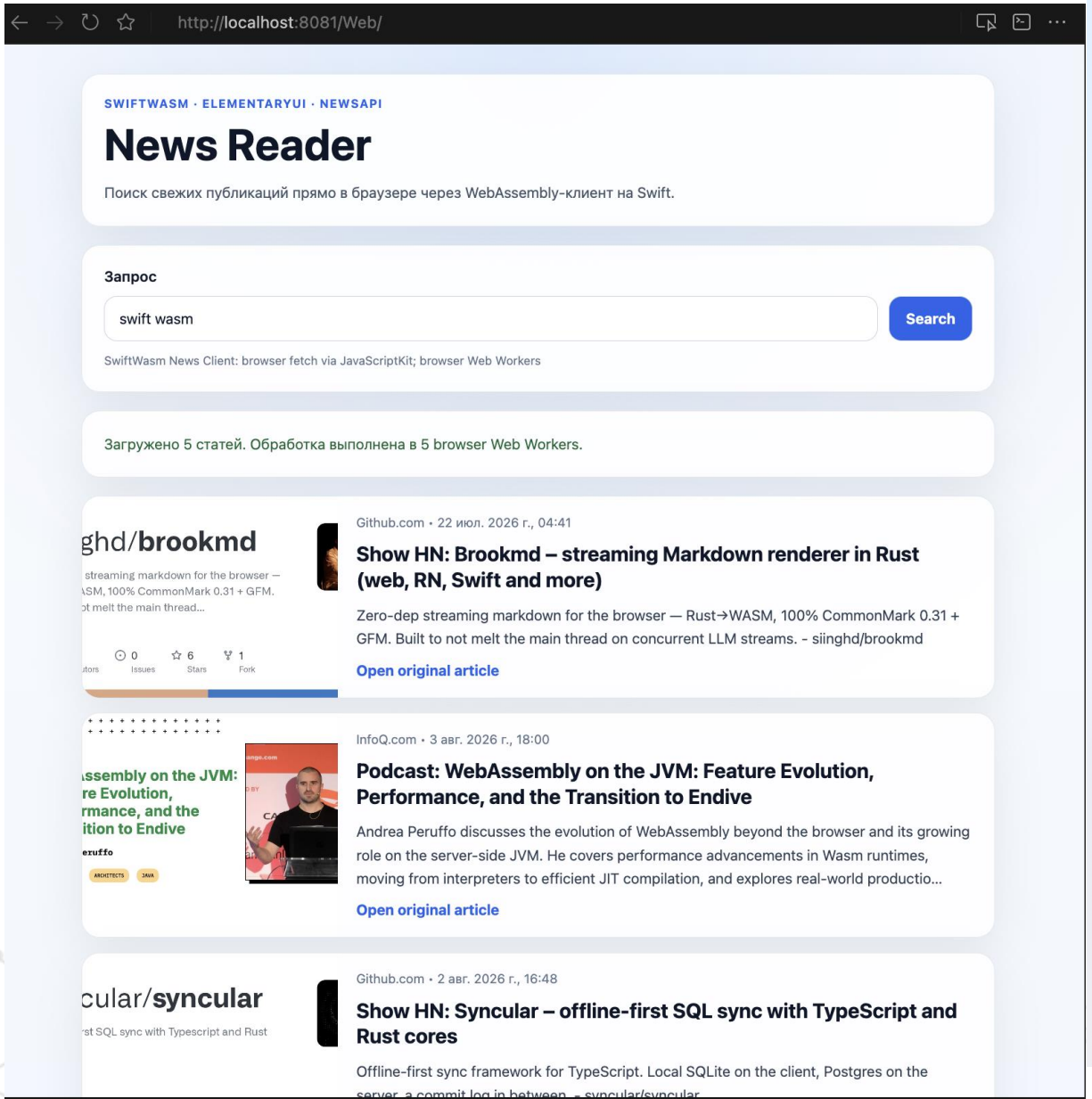
Swift WebAssembly. Общий модуль

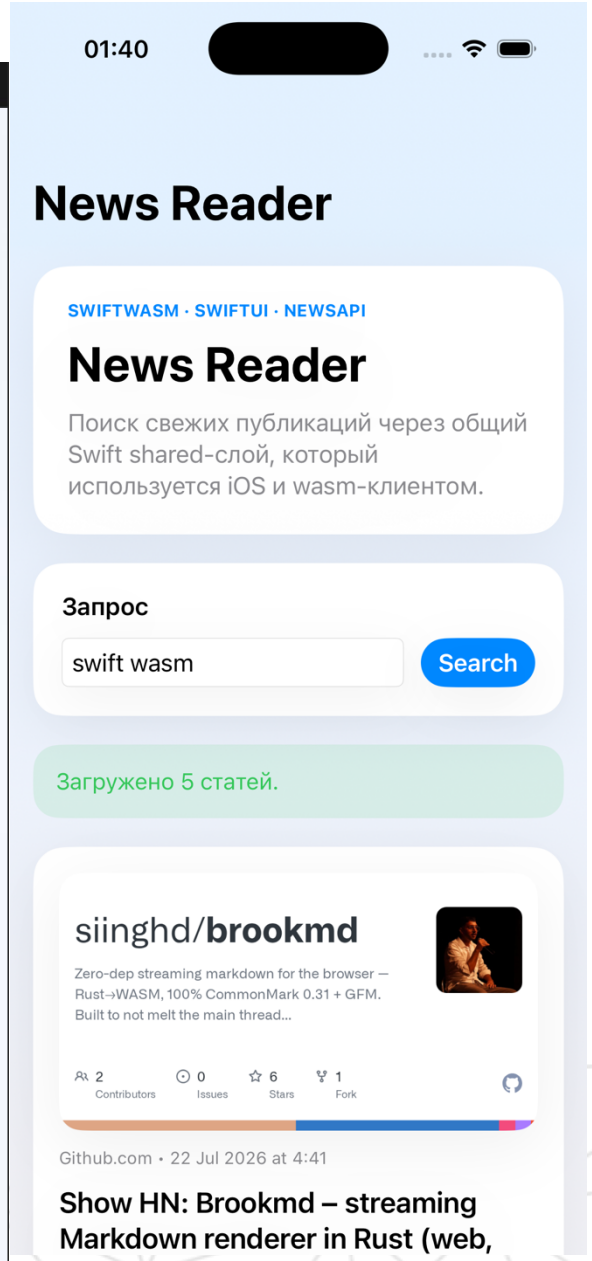
- JavaScriptKit
- JavaScriptEventLoop
- BridgeJs

```
    targets: [
        .executableTarget(
            name: "NewsWasmClient",
            dependencies: [
                .product(name: "JavaScriptKit", package: "JavaScriptKit"),
                .product(name: "JavaScriptEventLoop", package: "JavaScriptKit"),
            ],
            linkerSettings: [
                .unsafeFlags(
                    ["-lwasi-emulated-signal", "-lwasi-emulated-mman"],
                    .when(platforms: [.wasi])
                )
            ],
            plugins: [
                .plugin(name: "BridgeJS", package: "JavaScriptKit")
            ]
        )
    ],
    swiftLanguageModes: [.v6]
)
```

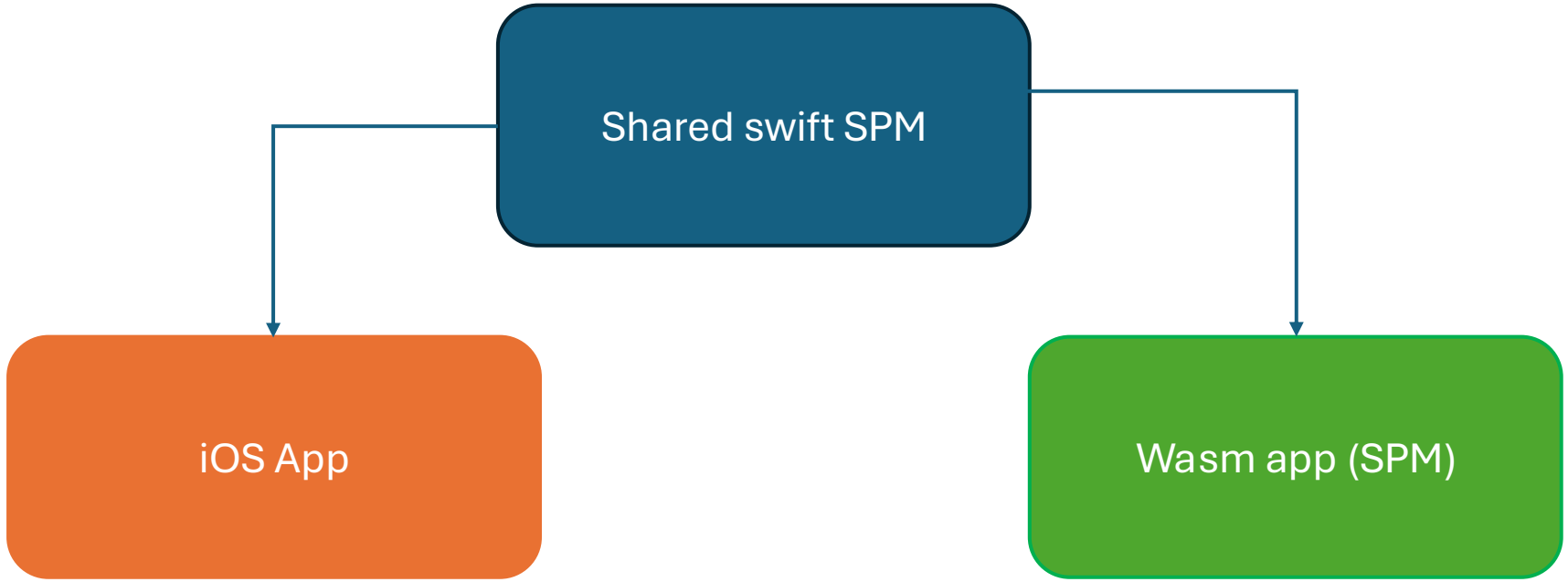
Swift-> Wasm,
разбираем на примере

Приложение News для Wasm и iOS

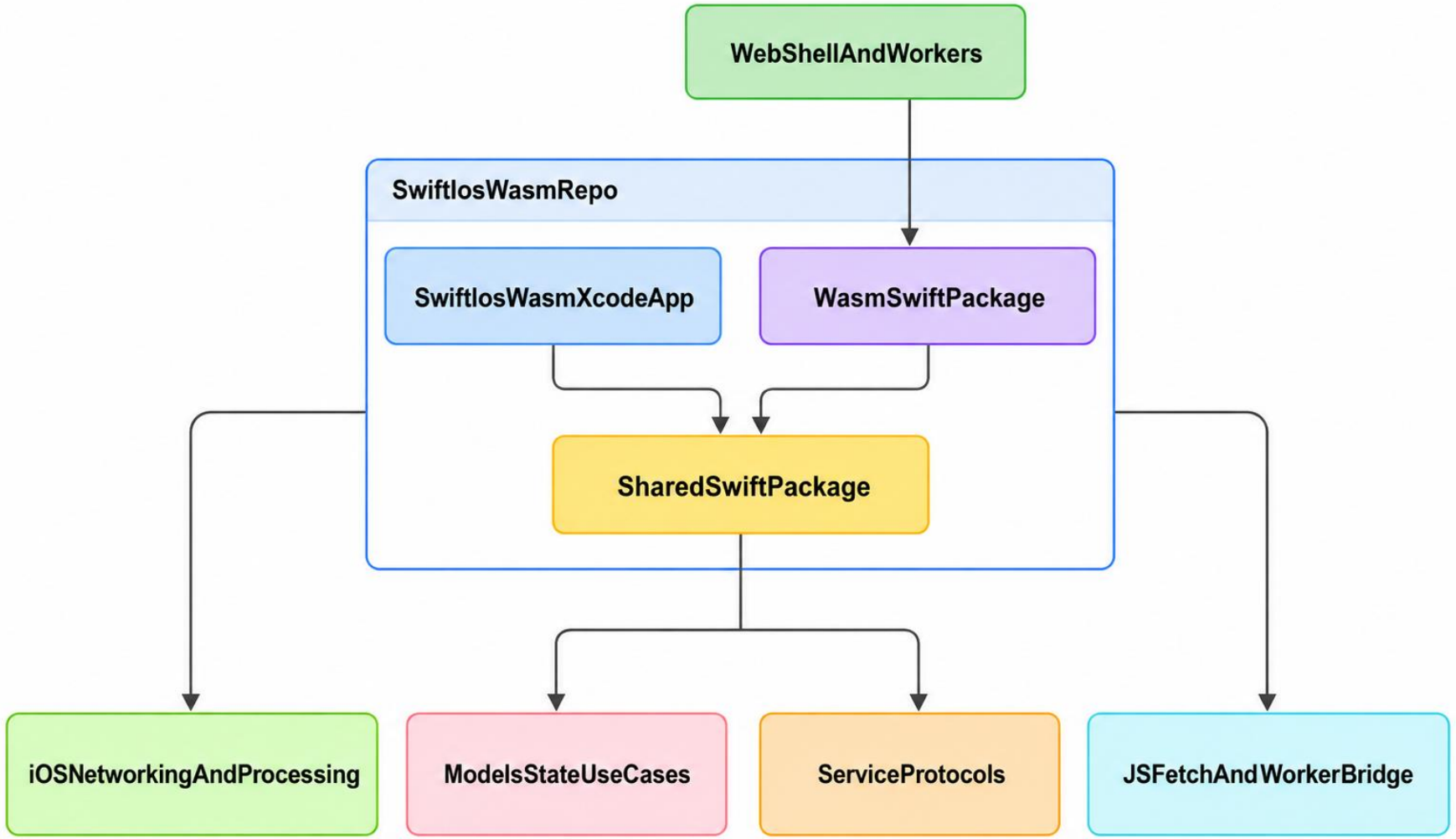




Приложение News для Wasm и iOS



Приложение News для Wasm и iOS



Приложение News для Wasm и iOS. Shared

Общий код

- Модели данных
- Контракты сетевых ответов
- Утилиты
- Мапперы, декодеры

- Многопоточность
- Работа с сетью



Приложение News для Wasm и iOS. Shared

Общий код

- Модели данных
- Контракты сетевых ответов
- Утилиты
- Мапперы, декодеры

~~- Многопоточность~~
~~- Работа с сетью~~

Приложение News для Wasm и iOS. Ограничения

https://book.SwiftWasm.org/

The Foundation core library is available in SwiftWasm, but in a limited capacity. The main reason is that the Dispatch core library is unavailable due to the lack of standardized multi-threading support in WebAssembly and SwiftWasm itself. Many Foundation APIs rely on the presence of Dispatch under the hood, specifically threading helpers. A few other types are unavailable in browsers or aren't standardized in WASI hosts, such as support for sockets and low-level networking, and they had to be disabled. These types are therefore absent in SwiftWasm Foundation:

Приложение News для Wasm и iOS. Ограничения

https://book.SwiftWasm.org/

The Foundation core library is available in SwiftWasm, but in a limited capacity. The main reason is that the Dispatch core library is unavailable due to the lack of standardized multi-threading support in WebAssembly and SwiftWasm itself. Many Foundation APIs rely on the presence of Dispatch under the hood, specifically threading helpers. A few other types are unavailable in browsers or aren't standardized in WASI hosts, such as support for sockets and low-level networking, and they had to be disabled. These types are therefore absent in SwiftWasm Foundation:

Есть ограничения по поддержки сети и многопоточности

Приложение News для Wasm и iOS. Ограничения

https://book.SwiftWasm.org/

Type or module	Status
`FoundationNetworking`	❌ Unavailable
`FileManager`	✅ Available after 6.0
`Host`	✅ Partially available after 6.0
`Notification`	✅ Available after 6.0
`NotificationQueue`	❌ Unavailable
`NSKeyedArchiver`	✅ Available after 6.0
`NSKeyedArchiverHelpers`	✅ Available after 6.0
`NSKeyedCoderOldStyleArray`	✅ Available after 6.0
`NSKeyedUnarchiver`	✅ Available after 6.0
`NSNotification`	✅ Available after 6.0
`NSSpecialValue`	✅ Available after 6.0
`Port`	✅ Available after 6.0
`PortMessage`	✅ Available after 6.0
`Process`	❌ Unavailable
`ProcessInfo`	✅ Partially available after 5.7
`PropertyListEncoder`	✅ Available after 6.0
`RunLoop`	❌ Unavailable
`Stream`	✅ Partially available after 6.0
`SocketPort`	❌ Unavailable
`Thread`	✅ Available after 6.4
`Timer`	❌ Unavailable
`UserDefaults`	✅ Available after 6.0

Swift-> Wasm, многопоточность

SwiftWasm. Многопоточность

Limitations

WebAssembly and WASI provide a constrained environment, which currently does not directly support multi-threading. Thus, you should not expect these APIs to work when importing `WASILibc`. Please be aware of these limitations when porting your code, which also has an impact on what can be supported in the Swift Foundation at the moment.

- Wasm не поддерживают многопоточность напрямую

SwiftWasm. Многопоточность

While the WebAssembly spec defines atomic operations, it does not define a way to create threads. This means that WebAssembly modules can't create threads themselves, and the host environment must provide a way to create threads and run WebAssembly modules on them.

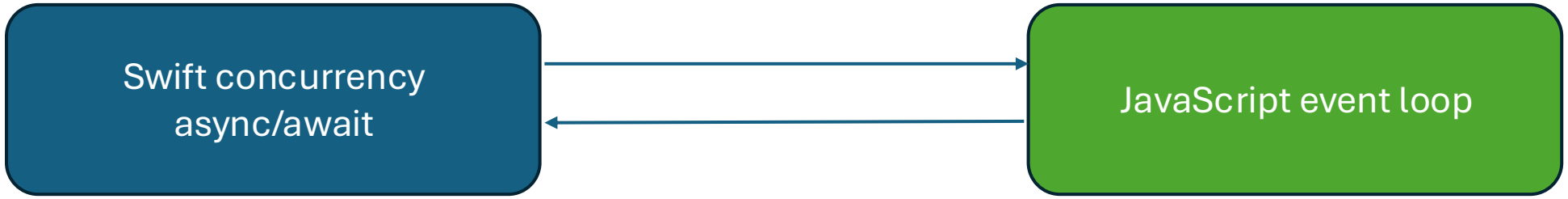
- Wasm модули не создают потоки сами
- SwiftWasm-код исполняется внутри браузера как WebAssembly-модуль. Хост предоставляет среду для создания потока и запуска на нем модуля

SwiftWasm. Многопоточность

- async/await Swift
- JavaScript Promise
- Web Workers.

SwiftWasm. Многопоточность

JavaScriptEventLoop.installGlobalExecutor()



SwiftWasm. Многопоточность

Обычная Swift Structured concurrency для работы вне UI

```swift
// USAGE
// $ swiftc -target wasm32-unknown-wasi -parse-as-library main.swift -o main.wasm
// $ wasmtime main.wasm
@main
struct Main {
    static func main() async throws {
        print("Sleeping for 1 second... 😴")
        try await Task.sleep(nanoseconds: 1_000_000_000)
        print("Wake up! 😄")
    }
}
```

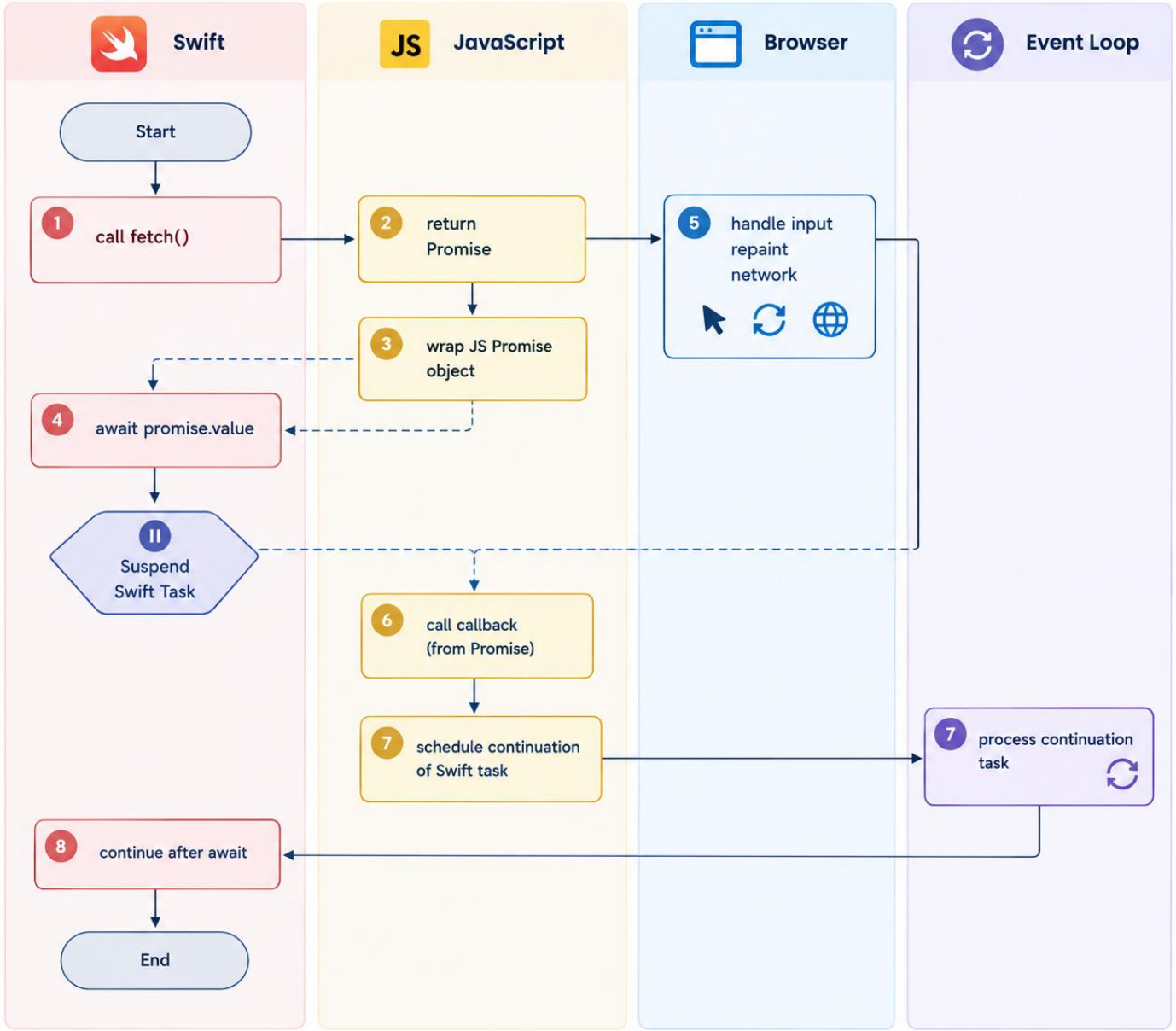
SwiftWasm. Многопоточность

JavaScript Promise – под капотом веб-воркеры

```
guard let promise = JSPromise(fetch(url, options).object!) else {
    throw NewsAPIError.network("Fetch не вернул Promise.")
}

let response = try await promise.value
```

SwiftWasm. Многопоточность



SwiftWasm. Многопоточность

Обработка в общей (или не общей) ViewModel

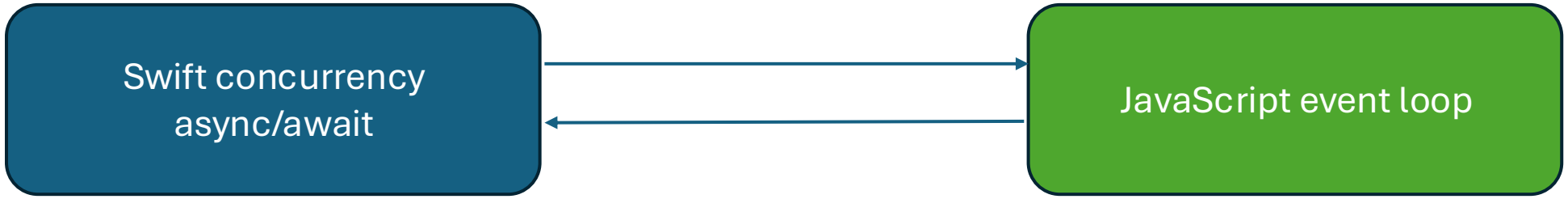
```
Task { @Sendable in
    do {
        let articles = try await client.fetchArticles(query: currentQuery)
        let output = try await pipeline.process(
            articles: articles,
            query: currentQuery
        )

        self.completeLoad(...)
    } catch {
        self.completeLoad(...)
    }
}
```

SwiftWasm. Многопоточность. TaskExecutor

deprecated

JavaScriptEventLoop.installGlobalExecutor()



WebWorkerTaskExecutor

SwiftWasm. Многопоточность. TaskExecutor

TaskExecutor для упрощения работы с многопоточностью

Add `WebWorkerTaskExecutor` #256

Merged **kateinoigakukun** merged 12 commits into `main` from `yt/web-worker-executor` on Jul 7, 2024

Conversation 3 | Commits 12 | Checks 0 | Files changed 16



kateinoigakukun commented on Jul 6, 2024 • edited — Member

WebWorkerTaskExecutor is an implementation of `TaskExecutor` protocol, which is introduced by SE-0417 since Swift 6.0. This task executor runs tasks on Worker threads, which is useful for offloading computationally expensive tasks from the main thread.

The `WebWorkerTaskExecutor` is designed to work with Web Workers API and Node.js's `worker_threads` module.

This depends on swiftlang/swift#75008

Example

Try it out: https://swiftwasm-threading-example.vercel.app/

SwiftWasm.Многопоточность. Поддержка своих акторов

Акторы https://github.com/SwiftWasm/WebWorkerKit

```swift
import WebWorkerKit

public struct SomeWorkResult: Codable, Sendable { // Codable, Sendable is important
    init(_ intermediateResults: Whatever) {...}
    // ...
}

distributed actor MyDistributedActorWorker: WebWorker {
     static let scriptPath: String? = nil

    static let isModule = false

    public distributed func doWork() async throws -> SomeWorkResult {
        let intermediateResults = try await calculateIntermediateResults() // this happens inside the web worker
        return SomeSendableWorkType(intermediateResults) // returned to "main" JS context
    }
}
```

SwiftWasm. Многопоточность. TaskExecutor

TaskExecutor на практике для передачи между потоками

```
// Create and transfer an object to a worker thread
let buffer = JSObject.global.ArrayBuffer.object!.new(1024).object!
let transferring = JSSending.transfer(buffer)


let task = Task(executorPreference: executor) {
    // Receive the transferred buffer in the worker
    let workerBuffer = try await transferring.receive()
    // Use the buffer in the worker thread
}
```

SwiftWasm. Многопоточность

TaskExecutor на практике

```swift
// Create an executor with 4 worker threads
let executor = try await WebWorkerTaskExecutor(numberOfThreads: 4)
defer { executor.terminate() }

// Execute a task on a worker thread
let task = Task(executorPreference: executor) {
    // This runs on a worker thread
    return performHeavyComputation()
}
let result = await task.value

// Run a block on a worker thread
await withTaskExecutorPreference(executor) {
    // This entire block runs on a worker thread
    performHeavyComputation()
}

// Execute multiple tasks in parallel
await withTaskGroup(of: Int.self) { group in
    for i in 0..<10 {
        group.addTask(executorPreference: executor) {
            // Each task runs on a worker thread
            return fibonacci(i)
        }
    }

    for await result in group {
        // Process results as they complete
    }
}
```

USETECH

SwiftWasm. Многопоточность. Поддержка своих акторов

Исправление работы MainActor -> без run идет на фоновом потоке, а не главном

```
func run(executor: WebWorkerTaskExecutor) async {
  await withTaskExecutorPreference(executor) {
    // This block runs on the Web Worker thread.
    await MainActor.run {
        // This block should run on the main thread, but it runs on
        // the Web Worker thread.
    }
  }
  // Back to the main thread.
}
```

Swift-> Wasm,
решаем работу с сетью

Приложение News для Wasm и iOS. Wasm для сети

Общий протокол для реализации сетевых клиентов

```
public protocol NewsFetching: Sendable {
    func fetchArticles(query: String) async throws -> [Article]
}
```

Приложение News для Wasm и iOS. Wasm для сети

```
// В браузере нельзя использовать `URLSession`, поэтому запрос выполняется
// через browser API `fetch`, доступный из Swift через `JavaScriptKit`:

private let fetch = JSObject.global.fetch.function!
```

Приложение News для Wasm и iOS. Wasm для сети

```
// Отмена запроса через контроллер и таймаут

let controller = JSObject.global.AbortController.function?.new()
options.signal = controller?.signal


private let timeoutMilliseconds = 12_000

controller.abort()
```

Swift-> Wasm,
делаем удобный UI

Выбираем свой UI

Shared

Html

TokomakUI/
ElementaryUI



WebApp

Tokamak UI

https://github.com/TokamakUI/Tokamak

ARCHIVED



Tokamak UI

UI в стиле SwiftUI

```
import JavaScriptKit
import TokamakDOM
import TokamakStaticHTML

struct NewsAppView: View {
    @ObservedObject private var model: NewsViewModel

    init(model: NewsViewModel = NewsViewModel()) {
        _model = ObservedObject(wrappedValue: model)
    }

    var body: some View {
        DynamicHTML("main", ["class": "shell"]) {
            HeroView()
            SearchControlsView(model: model)
            StatusView(status: model.status)
            ArticleListView(articles:
model.articles)
        }.onAppear {
            model.loadInitialNewsIfPossible()
        }
    }
}
```

Tokamak UI

UI в стиле SwiftUI

Свой SwiftUI на SwiftWasm

```swift
public protocol View {
  associatedtype Body: View

  @ViewBuilder
  var body: Self.Body { get }

  func _visitChildren<V: ViewVisitor>(_ visitor: V)

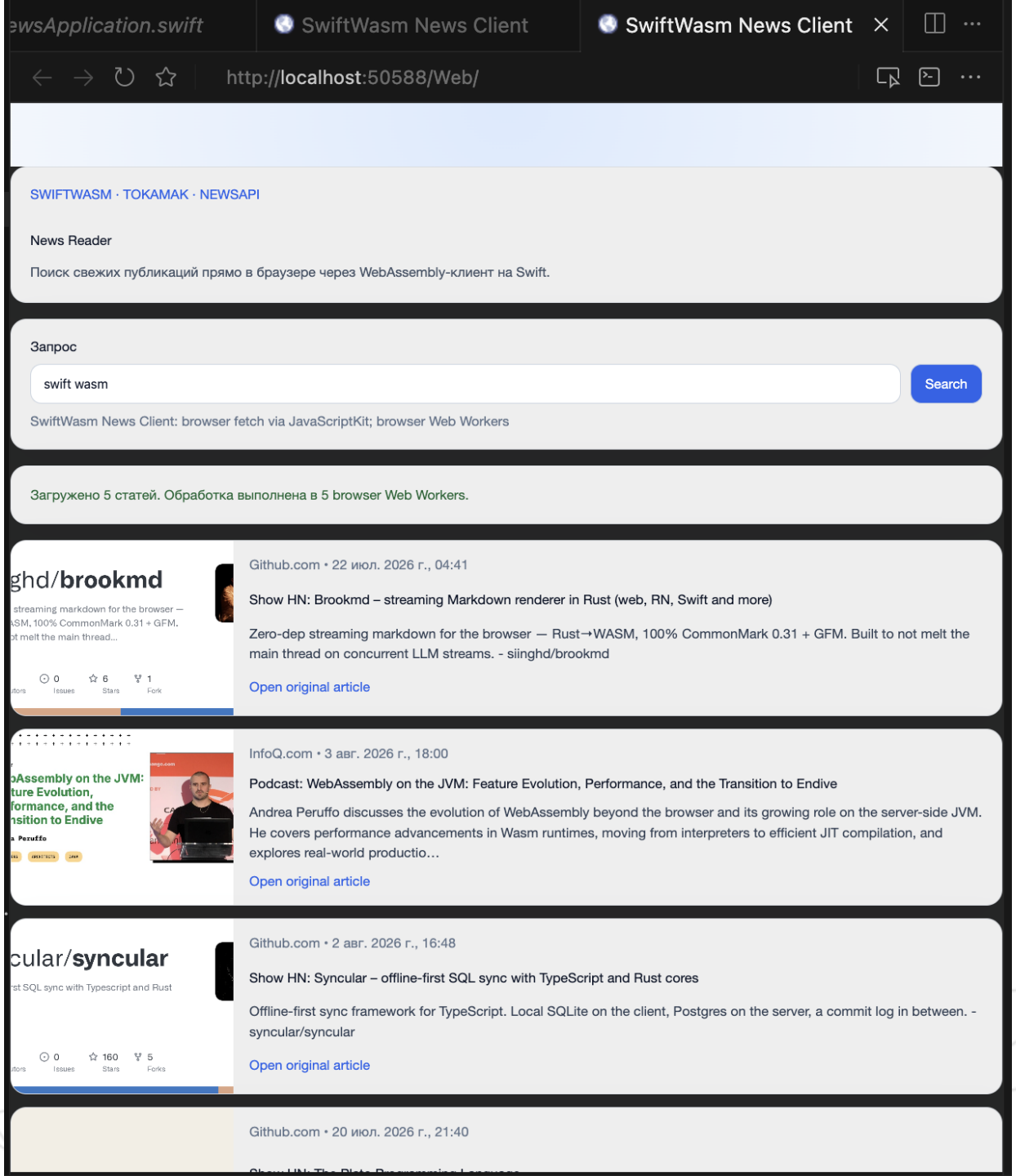
  static func _makeView(_ inputs: ViewInputs<Self>) ->
ViewOutputs
}

public extension Never {
  @_spi(TokamakCore)
  var body: Never {
    fatalError()
  }
}
```

Tokamak UI. Результат

UI в стиле SwiftUI

Свой SwiftUI на SwiftWasm



USETECH

OpenCombineJS. TokamakCore

UI в стиле SwiftUI

Логика в стиле Combine

https://github.com/SwiftWasm/OpenCombineJS

```
import Foundation
import JavaScriptKit
import TokamakDOM


final class NewsViewModel: ObservableObject, @unchecked Sendable {
    private let configuration = AppConfiguration.load()
    private lazy var client = NewsAPIClient(configuration: configuration)


    @Published var query: String
    @Published var articles: [Article] = []

    init() {...}

    var exportedAPIInfo: String {
        newsWasmClientInfo()
    }

    func loadInitialNewsIfPossible() {
        guard configuration.hasValidAPIKey, articles.isEmpty, !isLoading else
{           return
        }
        loadNews()
    }
  //..
}
```

USETECH

Combine Wasm + Tokamak Core

UI в стиле SwiftUI

Все доступно через Foundation

```
@available(macOS 12, iOS 15, tvOS 15, watchOS 8, *)
extension ObjectiveCConvertibleAttributedStringKey where Self.ObjectiveCValue == NSString,

    public static func objectiveCValue(for value: Self.Value) throws -> Self.ObjectiveCVal

    public static func value(for object: Self.ObjectiveCValue) throws -> Self.Value
}

@available(macOS 10.15, iOS 13.0, tvOS 13.0, watchOS 6.0, *)
public typealias ObservableObject = ObservableObject

/// Describes an error in the POSIX
@available(macOS 10.10, iOS 8.0, wat
public struct POSIXError {

    /// The domain of the error.
    public static var errorDomain: S

    /// The hash value.
    ///
    /// Hash values are not guarante
    /// your program. Do not save ha
    ///
    /// - Important: `hashValue` is
```

```
protocol ObservableObject : AnyObject

A type of object with a publisher that emits before the object has ch

By default an ObservableObject synthesizes an ObservableOb
the changed value before any of its @Published properties chang

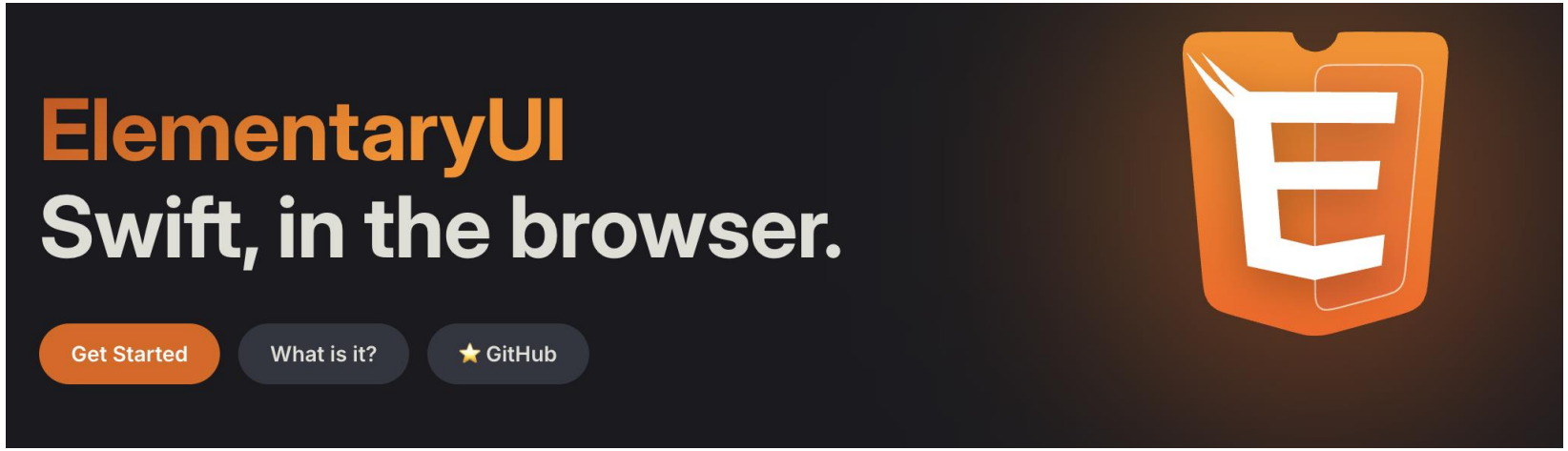
class Contact: ObservableObject {
    @Published var name: String
    @Published var age: Int

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }
```

ElementaryUI

UI в стиле SwiftUI

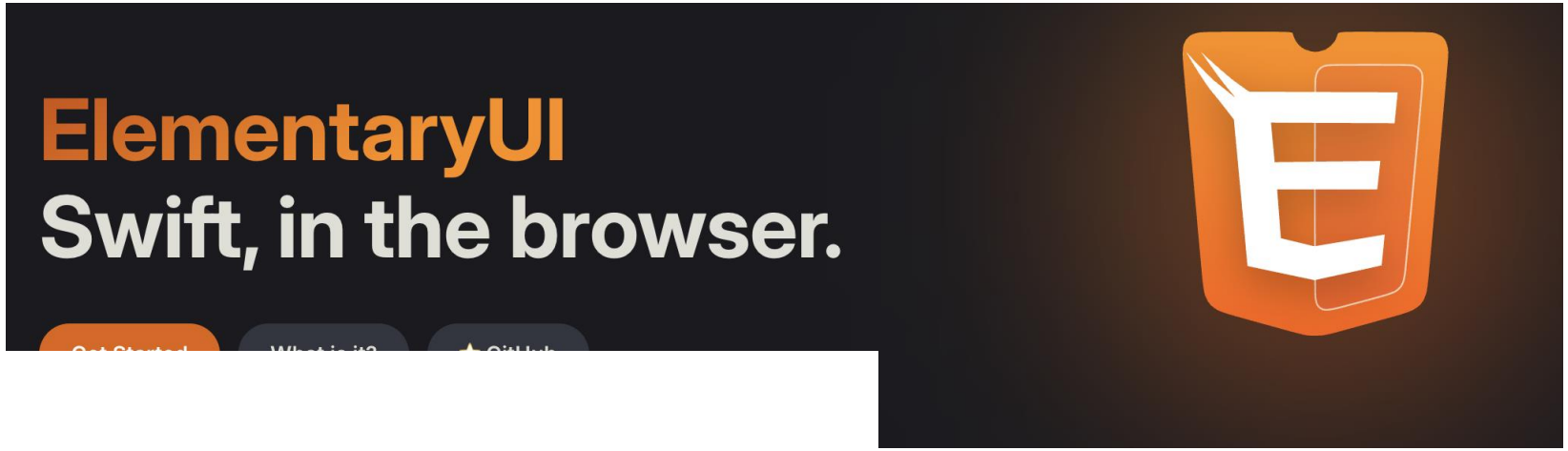
https://elementary.codes/



ElementaryUI

UI в стиле SwiftUI

https://elementary.codes/



ElementaryUI

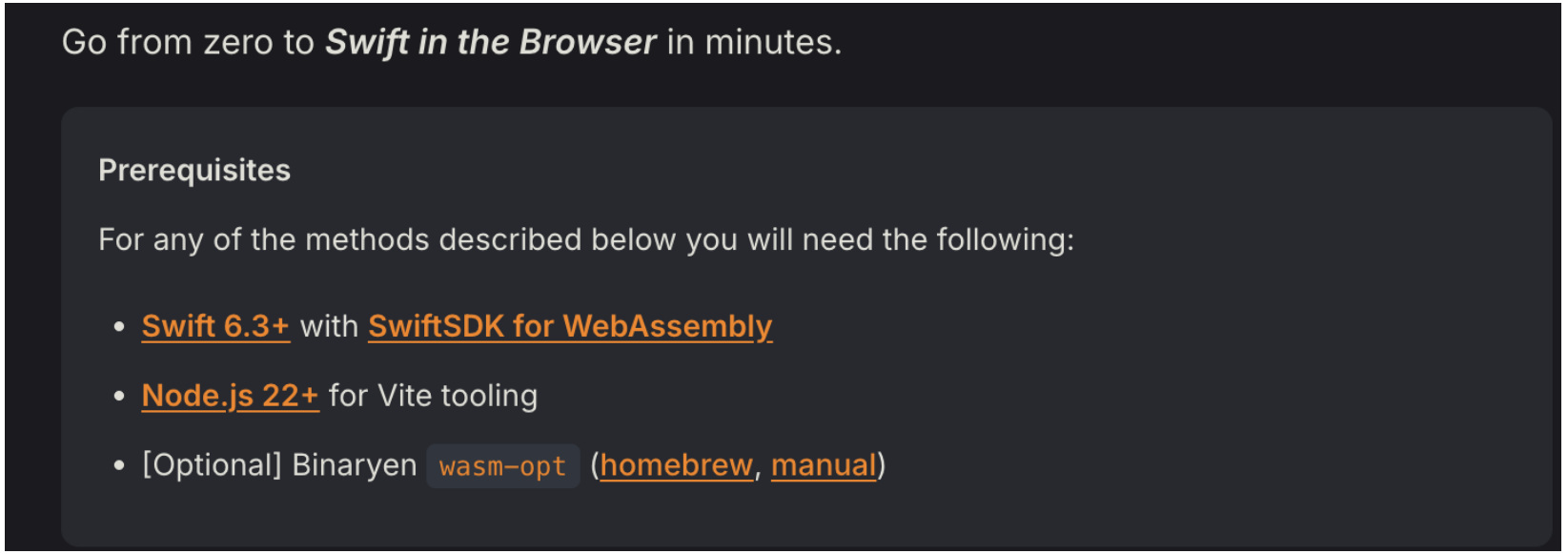
release v0.4.7 | Platforms macOS | Swift 6.3 | 6.1 | license Apache-2.0

> **Caution**
>
> ElementaryUI is currently in its alpha phase. This means it is still under active development and may undergo significant changes. Users should expect potential breaking changes and are encouraged to provide feedback to help improve the package. Contributions are welcome, and developers are invited to participate in the development process by submitting issues and pull requests.

ElementaryUI

UI в стиле SwiftUI

https://elementary.codes/



ElementaryUI

UI в стиле SwiftUI

https://github.com/buildexperience/ElementaryUI

```
import ElementaryUI
import JavaScriptKit

@View
struct NewsAppView {
    @State var model = NewsViewModel()

    var body: some View {
        main(.class("shell")) {
            HeroView()
            SearchControlsView(
                query: #Binding(model.query),
                isLoading: model.isLoading,
                exportedAPIInfo:
model.exportedAPIInfo,
                onSearch: { model.loadNews() }
            )
            StatusView(status: model.status)
            ArticleListView(articles: model.articles)
        }
        .onAppear {
            model.loadInitialNewsIfPossible()
        }
    }
}
```

USE TECH

ElementaryUI

@View

Макросы

ElementaryUI / View()

Macro

View()

Marks a struct as a view, enabling it to be used in the view hierarchy.

```
@attached(extension, conformances: __FunctionView, View, SVGView, __View
Equatable, names: named(__initializeState), named(__restoreState), named(__ap
macro View()
```

Macros.swift

ElementaryUI.Reactive

Макрос для ViewModel

```
import JavaScriptKit
import Reactivity

@Reactive
final class NewsViewModel: @unchecked Sendable {
    private let configuration: AppConfiguration
    private let client: NewsAPIClient
    private let pipeline = ArticleProcessingPipeline()

    var query: String
    var articles: [Article] = []

    func loadInitialNewsIfPossible() {
        guard configuration.hasValidAPIKey, articles.isEmpty, !isLoading else
{           return
        }
        loadNews()
    }
}
```

ElementaryUI.Reactive

Макрос для ViewModel

Macro

Reactive()

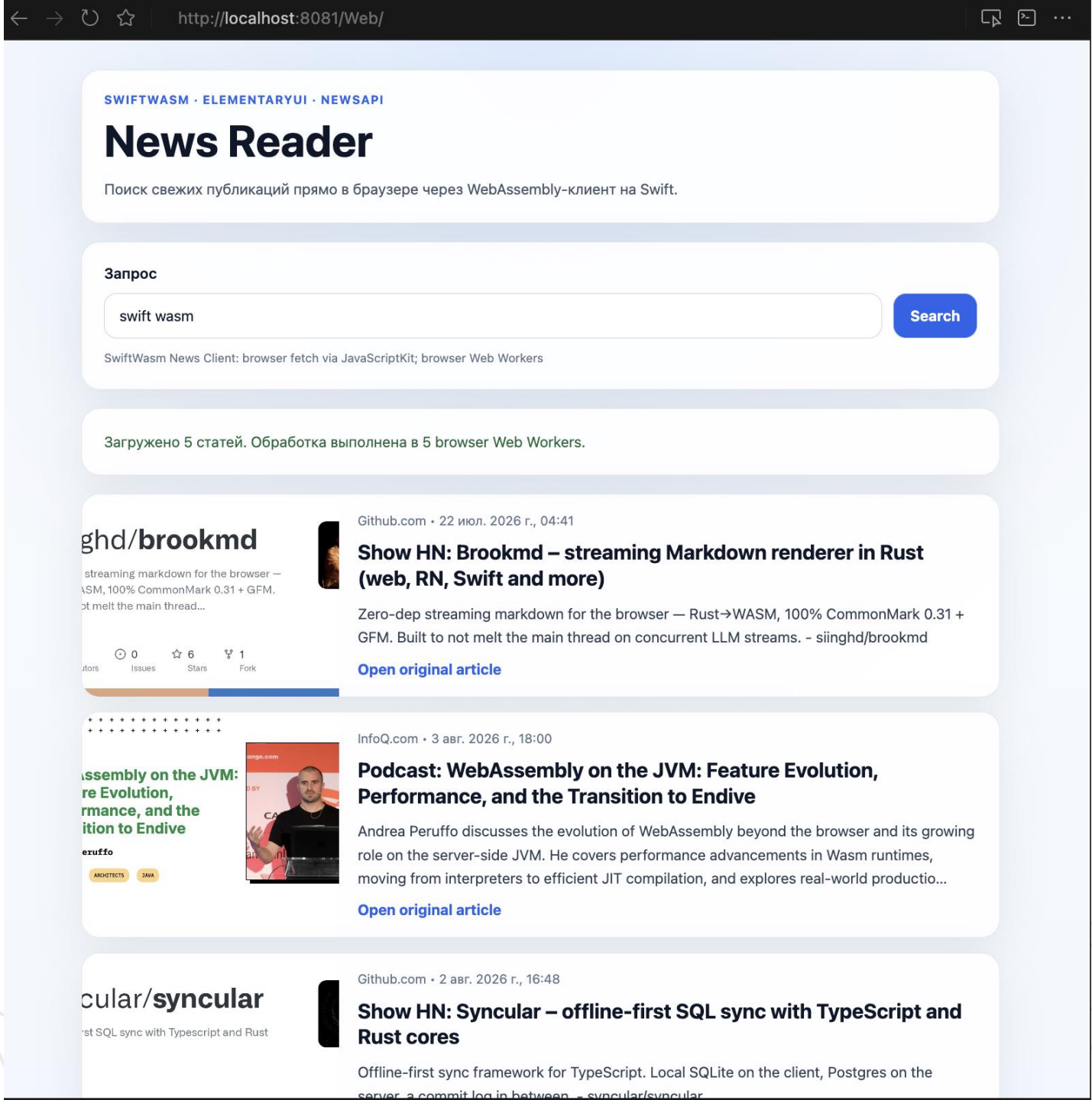
Marks a class as reactive, enabling automatic dependency tracking for its properties.

```
@attached(member, names: named(_$reactivity)) @attached(memberAttribute) @att
Object) macro Reactive()
```

Macros.swift

Результат

Полноценный Swift в браузере



USETECH

Что выбираем из UI

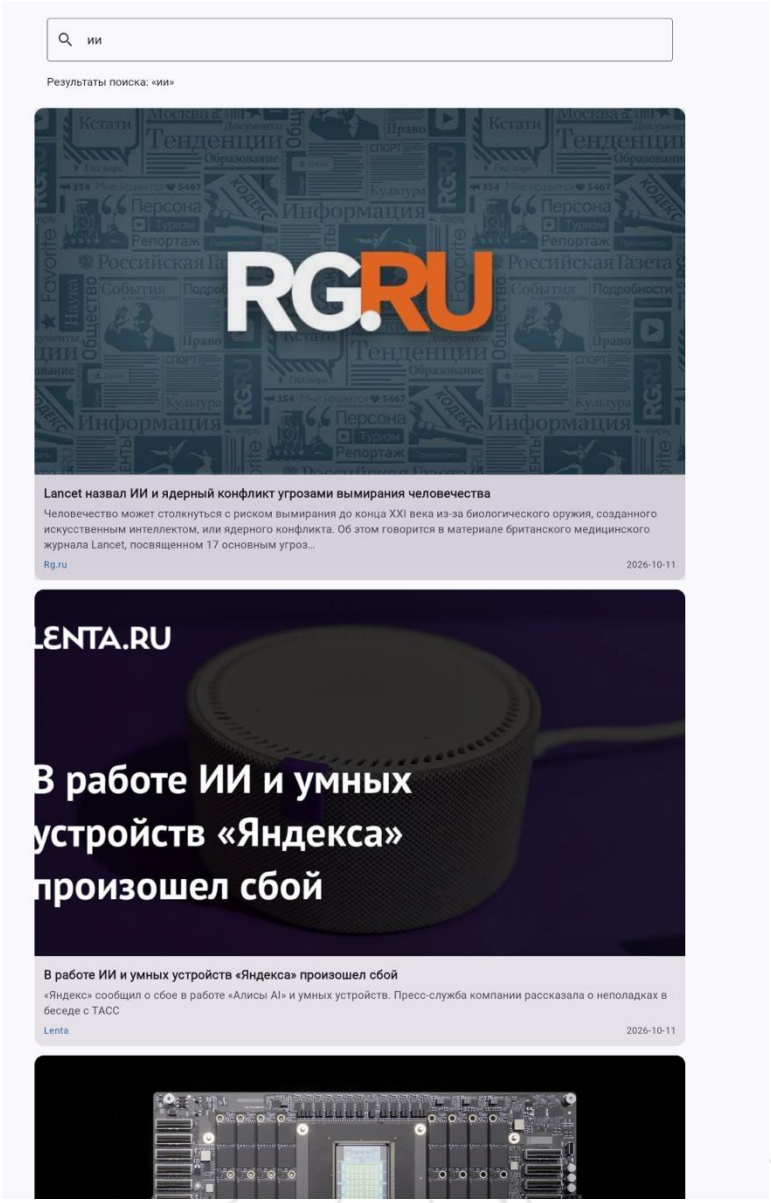
- Tokamak UI (архивированный, но рабочий)
- Elementary UI (еще Alpha)
- Пишем свое
- Html

Swift-> Wasm
vs Kotlin->Wasm

Kotlin/Wasm Beta

- Web target
- Compose for Web UI
- PWA и фронт на Kotlin





Kotlin/Wasm Beta

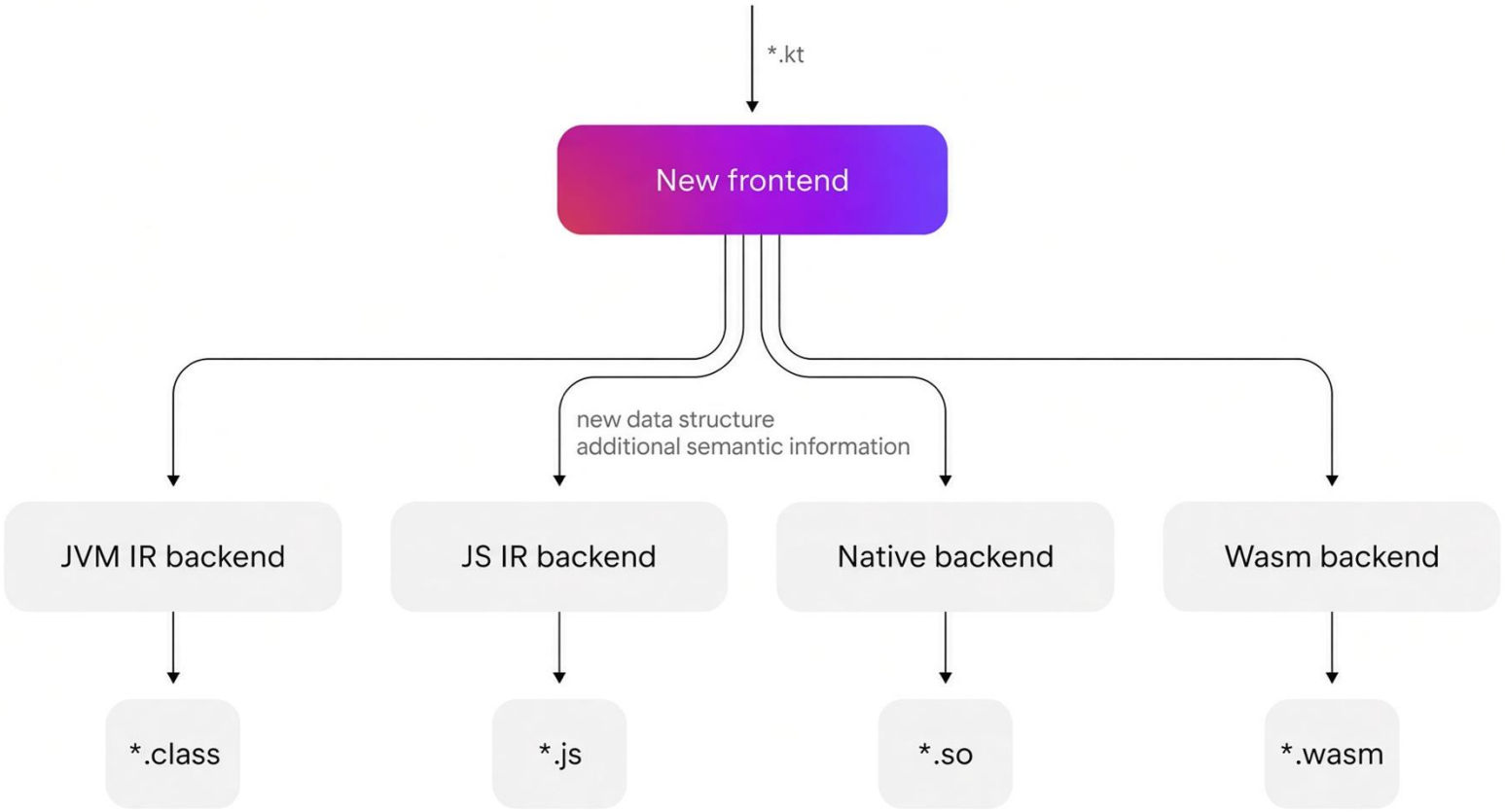
Общий код

- Модели данных
- Контракты сетевых ответов
- Утилиты
- Мапперы, декодеры

- ~~Многопоточность~~
- Работа с сетью

Kotlin/Wasm

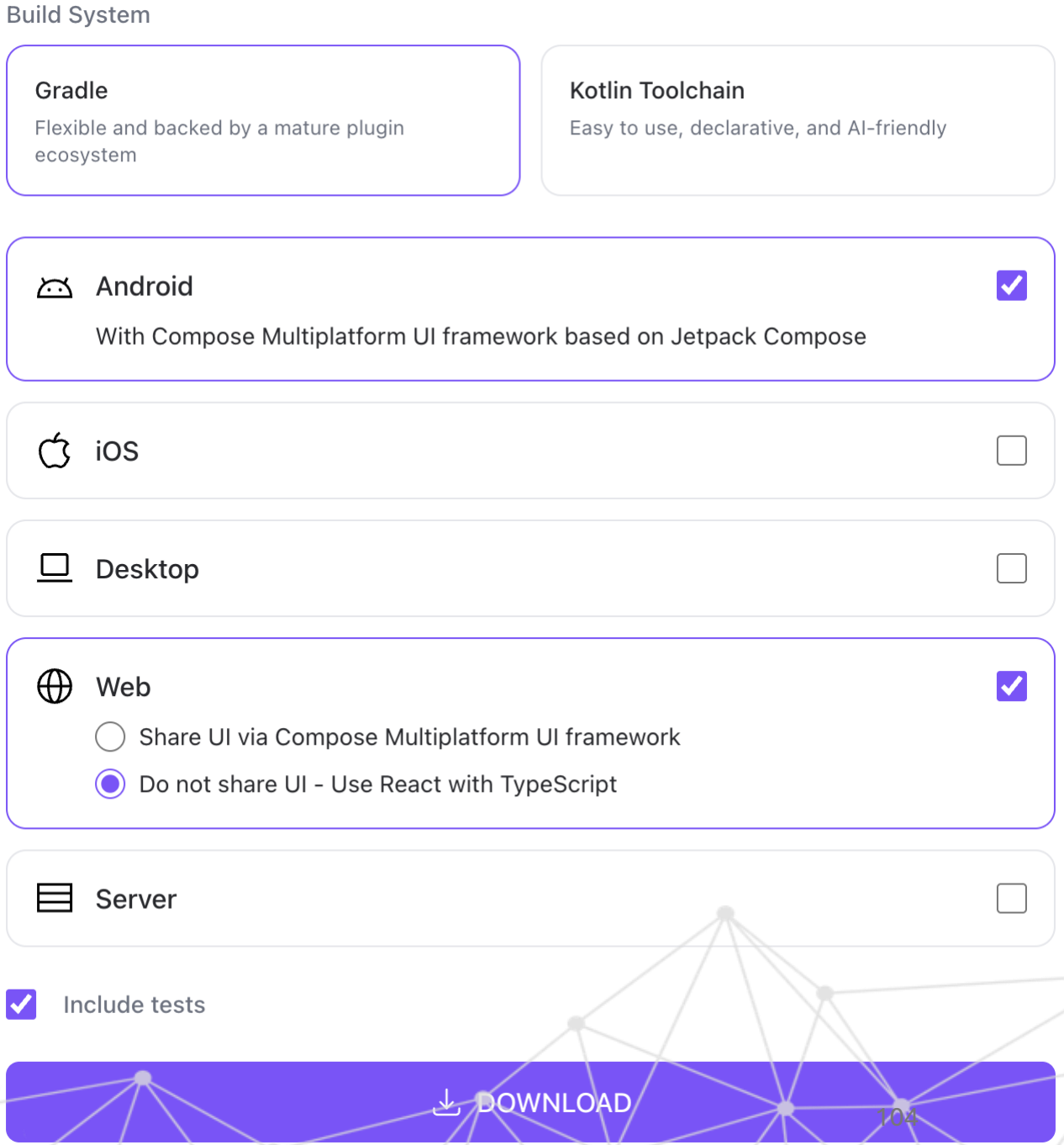
Beta



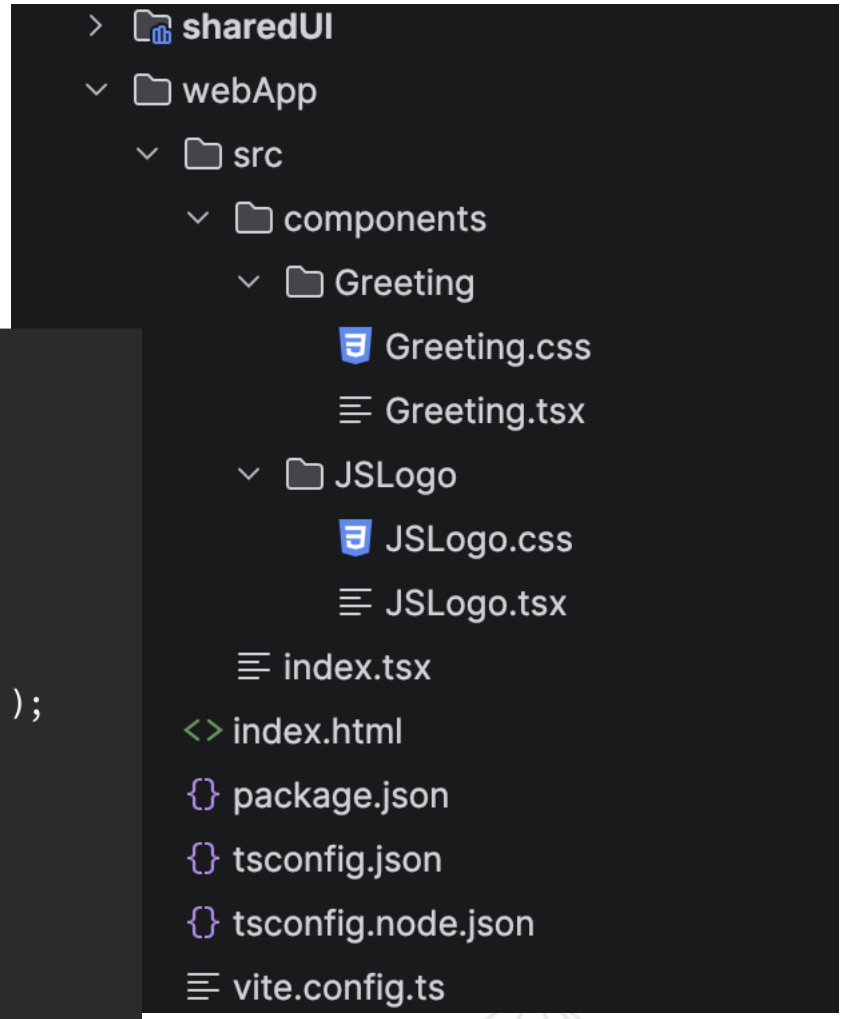
Kotlin Js vs Wasm

JS:

- Нативный UI на JS , нативное приложение на React
- Выборочное подключение логики из Shared-модуля



Kotlin Js vs Wasm




```tsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import { Greeting } from './components/Greeting/Greeting.tsx';

const rootElement = document.getElementById('root');
if (!rootElement) throw new Error('Failed to find the root element');

ReactDOM.createRoot(rootElement).render(
  <React.StrictMode>
    <Greeting />
  </React.StrictMode>
);
```

Kotlin Js vs Wasm

Использование TypeScript

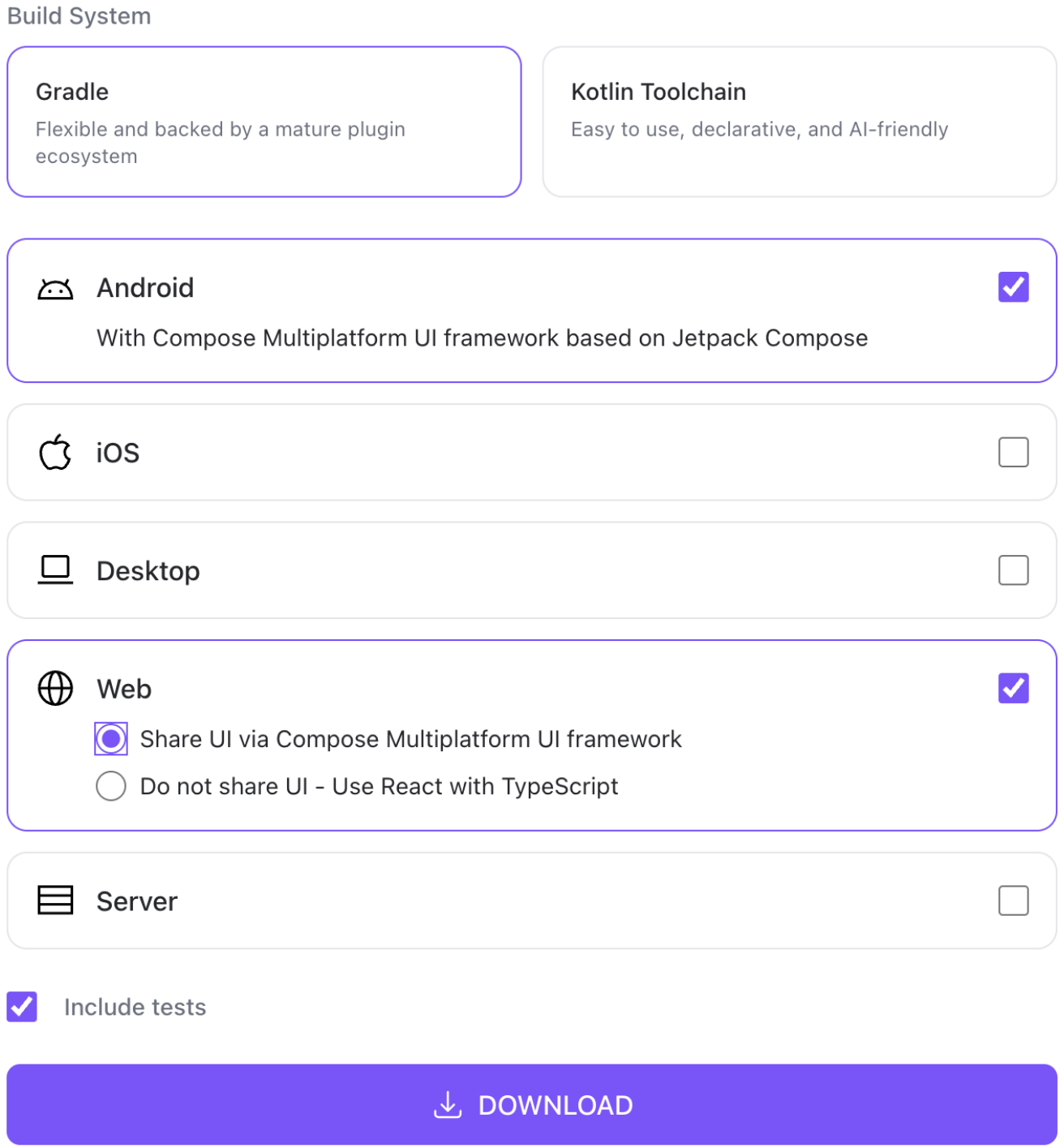
ES – базовый таргет

```
kotlin {
    js {
        outputModuleName = "sharedLogic"
        browser()
        binaries.library()
        generateTypeScriptDefinitions()
        compilerOptions {
            target = "es2015"
            optIn.add("kotlin.js.ExperimentalJsExport")
        }
    }
```

Kotlin Js vs Wasm

Wasm:

- Общий UI на Compose Web
- Общая логика



USETECH

Kotlin Js vs Wasm

Настраиваются WasmJs и Js

```
// dev-server c hot-reload
./gradlew wasmJsBrowserDevelopmentRun
```

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

    @OptIn(ExperimentalWasmDsl::class)
    wasmJs {
        browser()
        binaries.executable()
    }
}
```

Kotlin Js vs Wasm

```kotlin
@OptIn(ExperimentalComposeUiApi::class)
fun main() {
    ComposeViewport {
        App()
    }
}
```

```kotlin
@ExperimentalComposeUiApi
fun ComposeViewport(
    viewportContainerId: String? = null,
    configure: ComposeViewportConfiguration.() -> Unit = {},
    content: @Composable () -> Unit = { }
) {
    onDomReady {
        val providedContainer = if (viewportContainerId != null) {
            document.getElementById(viewportContainerId)
          ?: error("failed to find element by viewportContainerId: '$viewportContainerId'")
        } else {
            document.body ?: error("failed to find <body> element")
        }

        ComposeViewport(providedContainer, configure, content)
    }
}
```

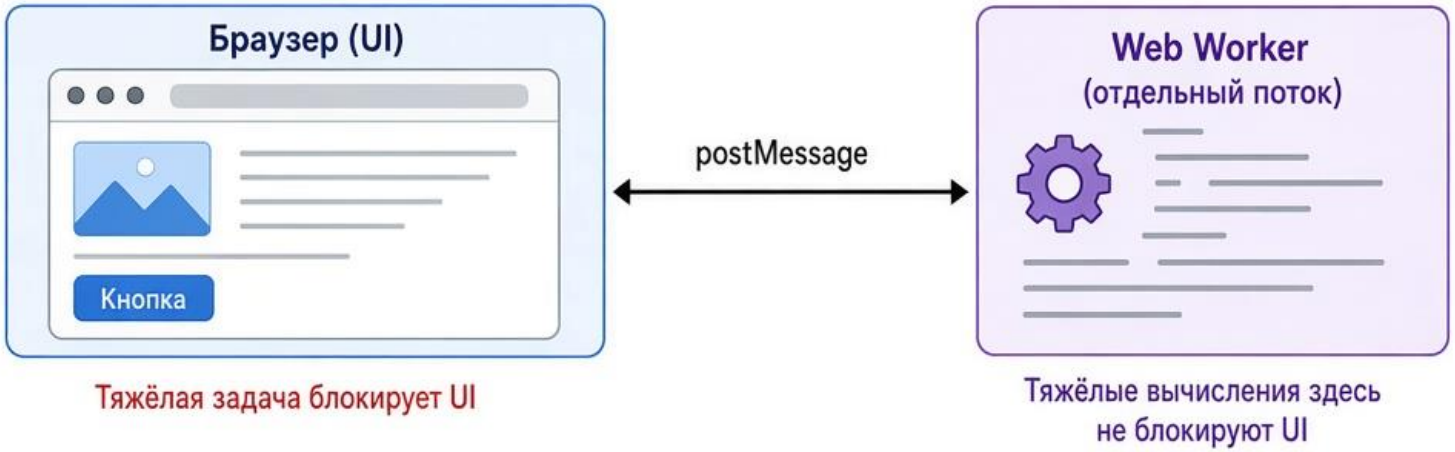
Kotlin Js vs Wasm. Многопоточность



Важно:

Dispatchers.Default и Dispatchers.IO в Wasm не создают фоновые потоки и работают как Dispatchers.Main.

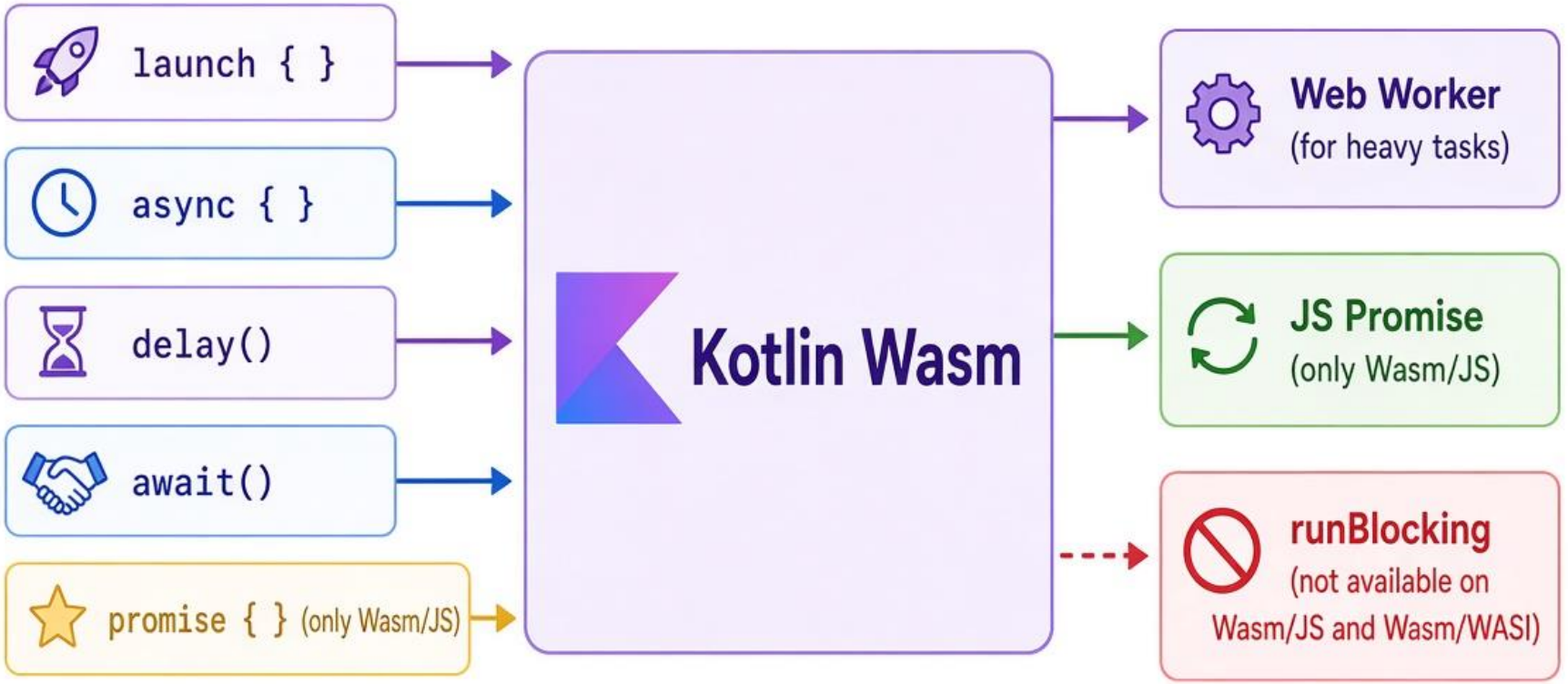
Для тяжёлых вычислений нужен Web Worker (автоматического диспетчера нет).



Тяжёлая задача блокирует UI

Тяжёлые вычисления здесь не блокируют UI

Kotlin Js vs Wasm. Многопоточность



- Wasm is **single-threaded event loop** environment;
- for real multithreading and heavy tasks need **Web Workers**;
- coroutine dispatchers only schedule on the **main thread**.

Kotlin Js vs Wasm. Общая логика в нативном приложении

Помечаем функционал через @JsExport

```
@JsExport
class NewsFeed {
    private val repository: NewsRepository = NewsRepositoryImpl(NewsApiService(createHttpClient()))


    suspend fun topHeadlines(category: String? = null): Array<Article> =
        repository.getTopHeadlines(category).getOrThrow().toTypedArray()

}
```

Kotlin Js vs Wasm. Общая логика в нативном приложении

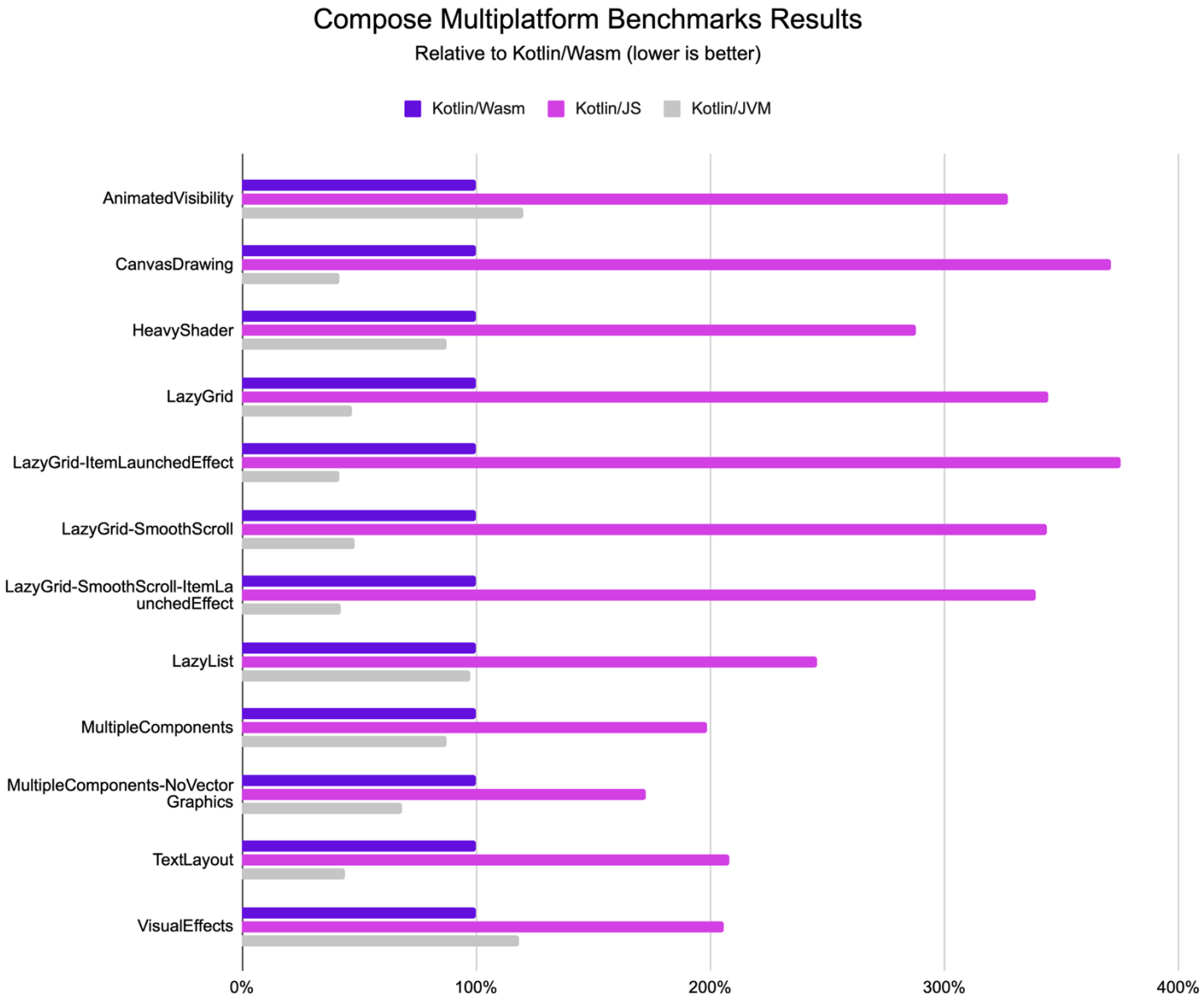
Внедряем в index.tsx

```tsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import { NewsApp } from './components/News/NewsApp.tsx';

const rootElement = document.getElementById('root');
if (!rootElement) throw new Error('Failed to find the root element');

ReactDOM.createRoot(rootElement).render(
  <React.StrictMode>
    <NewsApp />
  </React.StrictMode>,
);
```

Kotlin/Wasm + Compose



Kotlin/Wasm лучше Kotlin Js

Kotlin Interop JS

Kotlin	JavaScript
Byte, Short, Int, Char, UByte, UShort, U Int,	Number
Float, Double,	Number
Long, ULong,	BigInt
Boolean,	Boolean
String,	String
Unit in return position	undefined
Function type, for example (String) -> Int	Function
JsAny and subtypes	Any JavaScript value
JsReference	Opaque reference to Kotlin object
Other types	Not supported

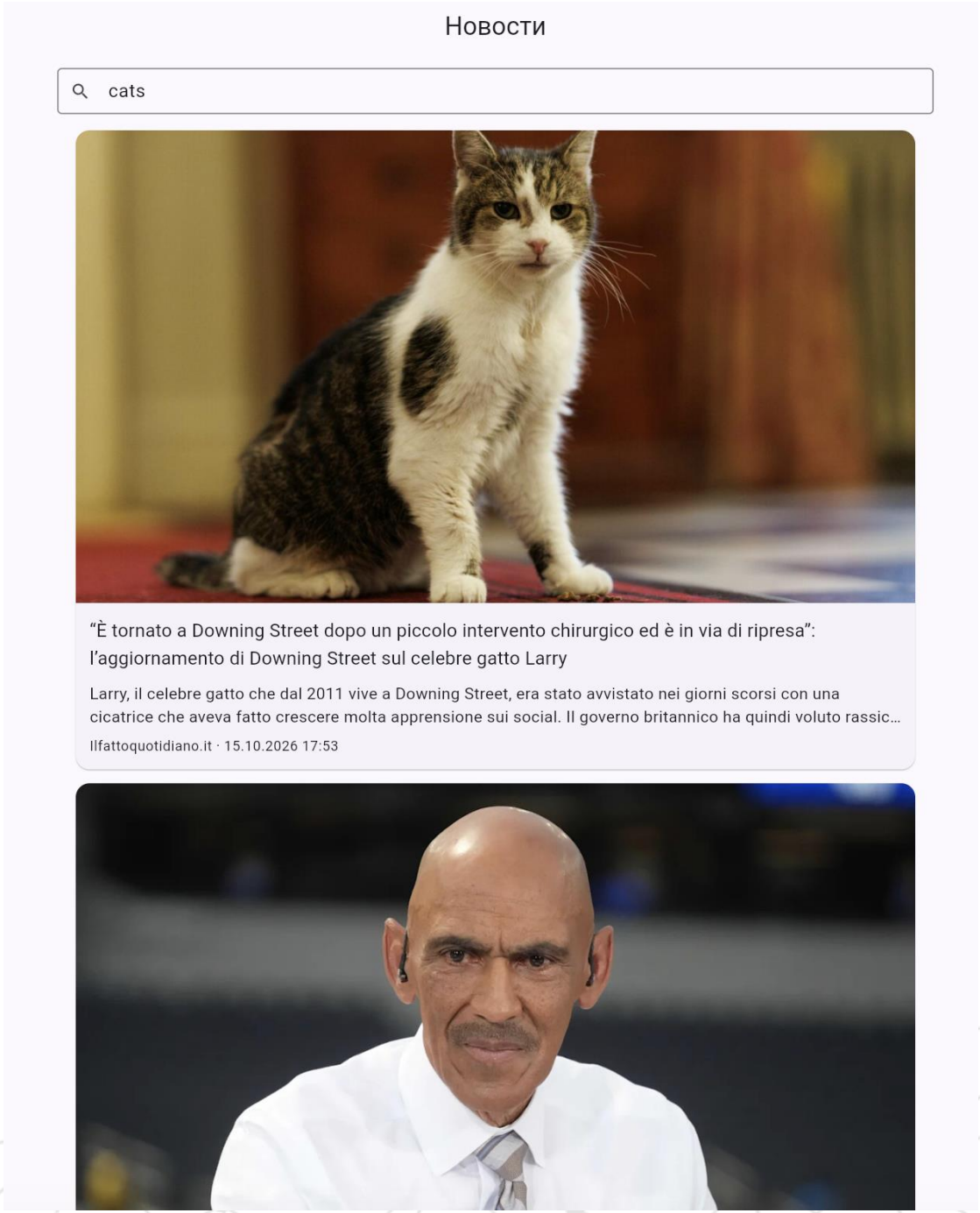
Kotlin/Wasm. Что ожидаем

- Проработку нативной поддержки многопоточности (shared-everything threads)
- Стабилизацию работы с Compose Multiplatform
- Поддержку современного JavaScript
- Расширение возможности экспорта между JavaScript и Kotlin

Flutter Web (бонус)

Flutter WebAssembly

- Stable с Flutter 3.22
- май 2024, анонс на Google I/O
- dart2wasm + WasmGC



USETECH

Flutter WebAssembly. Js -> Wasm

dart:js_interop

```dart
// dart:html legacy
// БЫЛО — не компилируется в Wasm
import 'dart:html' as html;

final el = html.document
    .createElement('a');

// package:web
// СТАЛО — работает и в JS, и в Wasm
import 'package:web/web.dart' as web;
import 'dart:js_interop';

final el = web.document
    .createElement('a')
    as web.HTMLAnchorElement;
```

Flutter WebAssembly. Производительность



Flutter WebAssembly. Dart2Wasm



Контраст с CanvasKit: раньше в Wasm была только Skia, логика оставалась на JS.
Теперь типизированный Wasm выполняет весь путь кода.

Сравним всех

	Swift (офиц. SDK)	Kotlin/Wasm	Flutter (dart2wasm)
Статус	официальный, Swift 6.2+	Beta (2025)	Stable (3.22+, 2024)
Память	ARC поверх линейной памяти	WasmGC	WasmGC
UI в браузере	Tokamak, ExperimentalUI	Compose for Web (Beta)	Flutter (stable)
Интероп с JS	JavaScriptKit: динамический	external/@JsFun: строгий	dart:js_interop: статический
Сервер (WASI)	основной сценарий	wasmWasi — Alpha	не поддерживается

Kotlin/Wasm. Когда стоит

- Есть готовы KMP-проект: веб-таргет добавляется быстро
- ·UI тяжёлый и интерактивный: data-viz, дашборды, редакторы
- команда знает Kotlin и Compose
- ·Beta-статус приемлем (Stable — близко)

Kotlin/Wasm. Когда не стоит

- Чисто веб проект без поддержки мобильных таргетов или десктоп: Kotlin/JS или TS проще

SwiftWasm. Когда стоит

- общая бизнес-логика с iOS/macOS-кодовой базой
- встраиваемые модули и плагины
- CLI-утилиты на WASI
- исполнение wasm внутри Swift-приложений (WasmKit)

SwiftWasm. Когда не стоит

- Больше веб-кода, чем Swift
- Нужен полноценный UI фреймворк/ сложный UI
- Важен маленький размер бандла



Спасибо за внимание!

@anioutkajarkova

azharkova
prettygeeknotes