KMP на Аврора ОС - проблемы и пути их решения

- Супрун Денис Алексеевич, 36 лет, индивидуальный предприниматель
- окончил физический факультет РГУ (ЮФУ), степень магистра по специальности Цифровая Обработка Сигналов
- стаж разработки под Android и iOS более 10 лет
- последние 5 лет mobile tech lead в Liloo Labs Inc.

План

1. Краткий обзор KMP и подходов к созданию общего кода
2. Kotlin JS на Аврора
3. Kotlin Native на Аврора
4. Kotlin JVM на Аврора с помощью Java to Native
5. Проблемы с портированием Compose на Аврора
6. Итоги

План

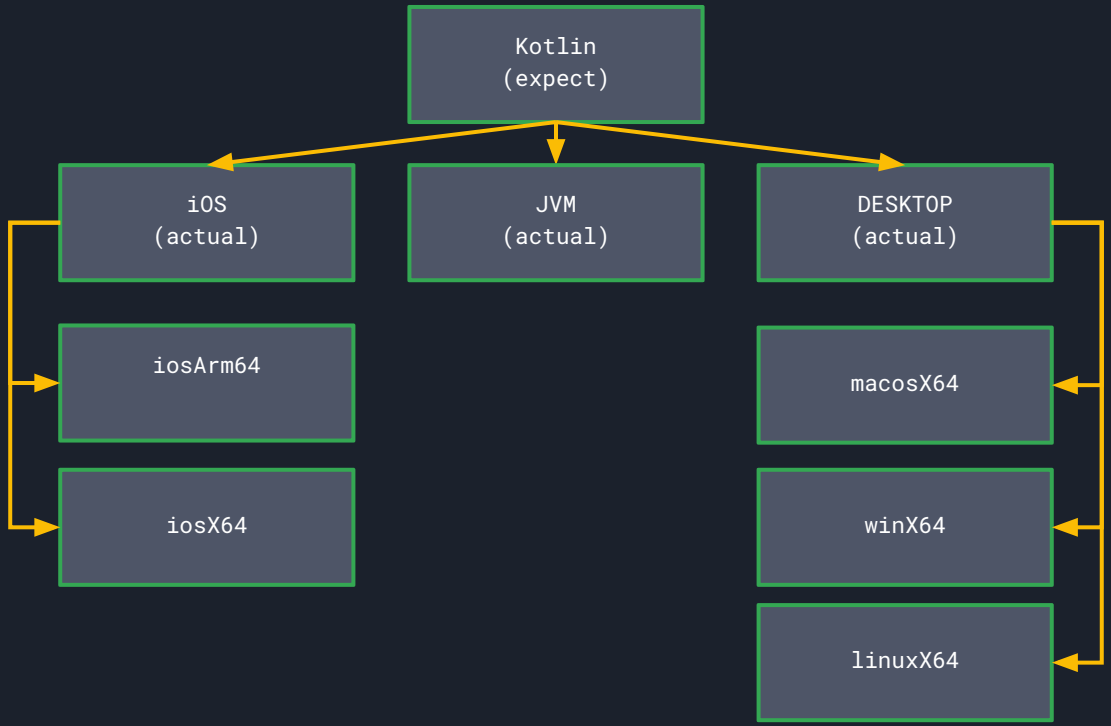
1. Краткий обзор KMP и подходов к созданию общего кода
2. Kotlin JS на Аврора
3. Kotlin Native на Аврора
4. Kotlin JVM на Аврора с помощью Java to Native
5. Проблемы с портированием Compose на Аврора
6. Итоги

Краткий обзор KMP

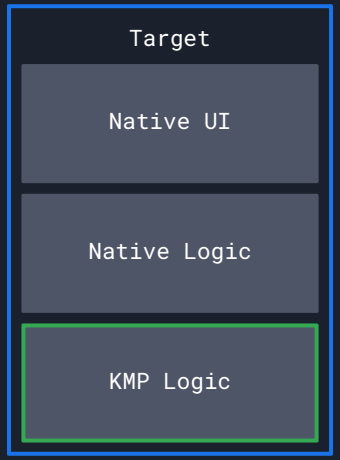
- Примитивные типы данных
- Модели и их сериализация
- Асинхронные операции с помощью корутин
- Сетевое взаимодействие с помощью ktor
- Хранение данных с помощью SQLDelight (но тут с нюансами)
- UI с помощью Compose (тут ещё больше нюансов)

Краткий обзор KMP

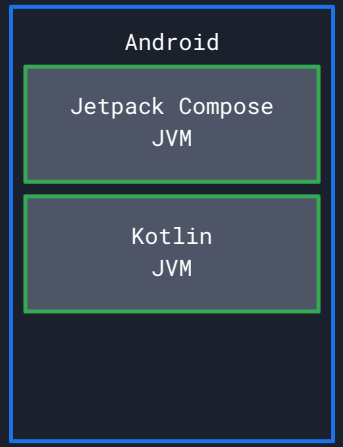
Краткий обзор KMP



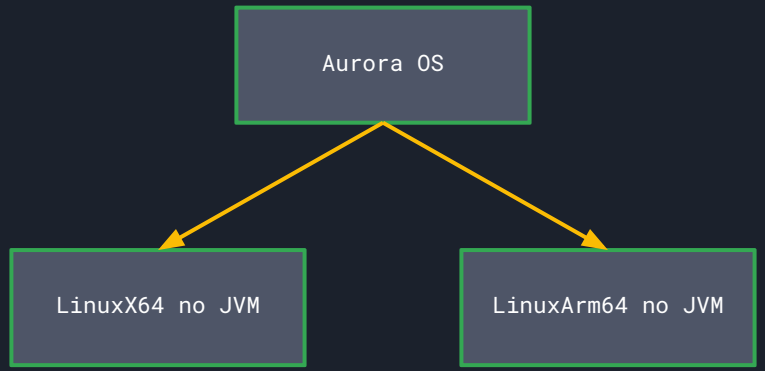
Краткий обзор KMP



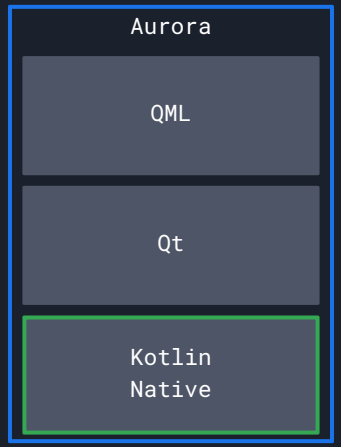
Краткий обзор KMP

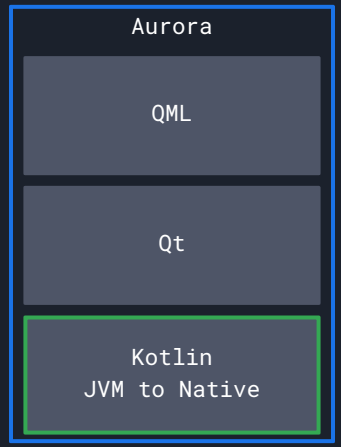


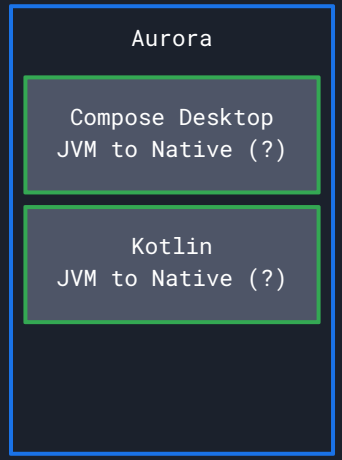
Краткий обзор KMP



Краткий обзор KMP







Краткий обзор KMP

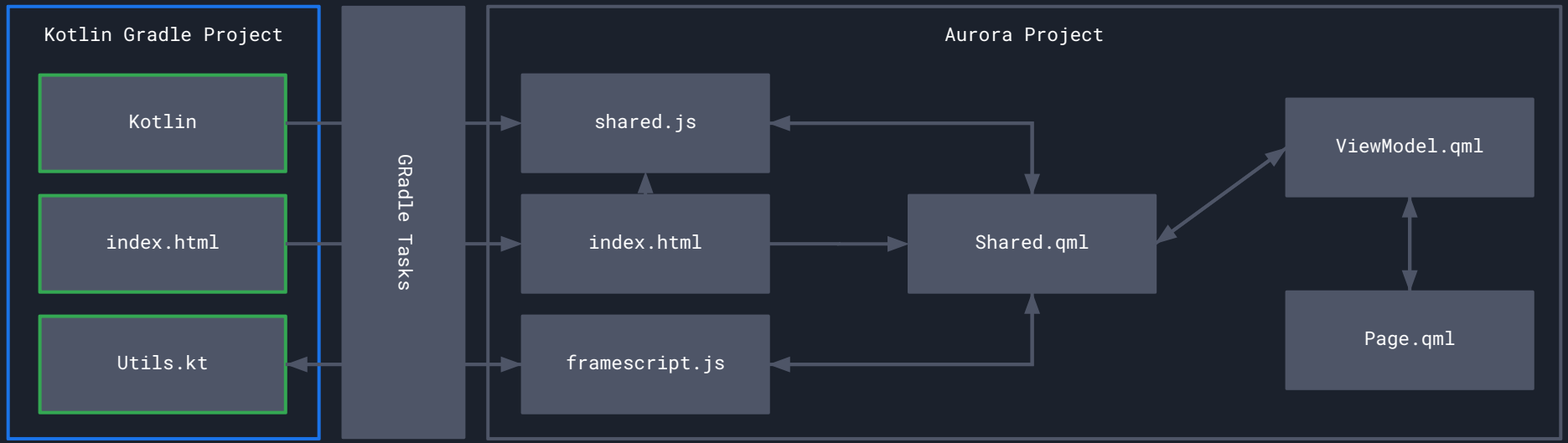
План

1. Краткий обзор KMP и подходов к созданию общего кода
2. Kotlin JS на Аврора
3. Kotlin Native на Аврора
4. Kotlin JVM на Аврора с помощью Java to Native
5. Проблемы с портированием Compose на Аврора
6. Итоги

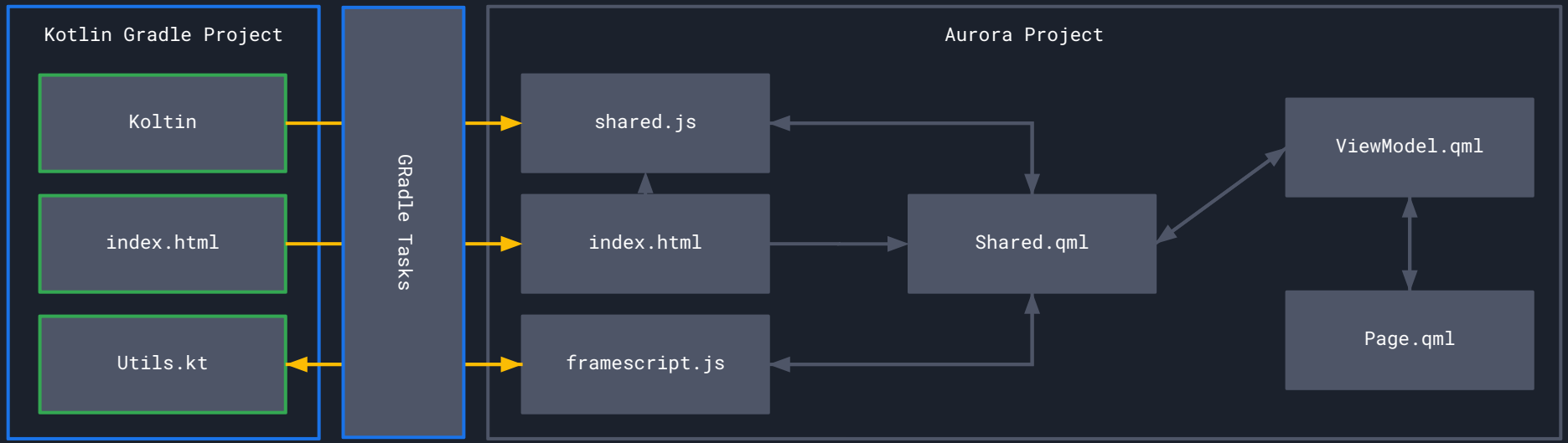
Kotlin JS на Аврора

- Kotlin траспилируется в JavaScript
- JavaScript загружается в WebView
- Из QML вызываем JavaScript методы с помощью **runJavaScript**
- Получаем синхронные результаты в **runJavaScript** колбеке
- Получаем асинхронные результаты в **onRecvAsyncMessage**

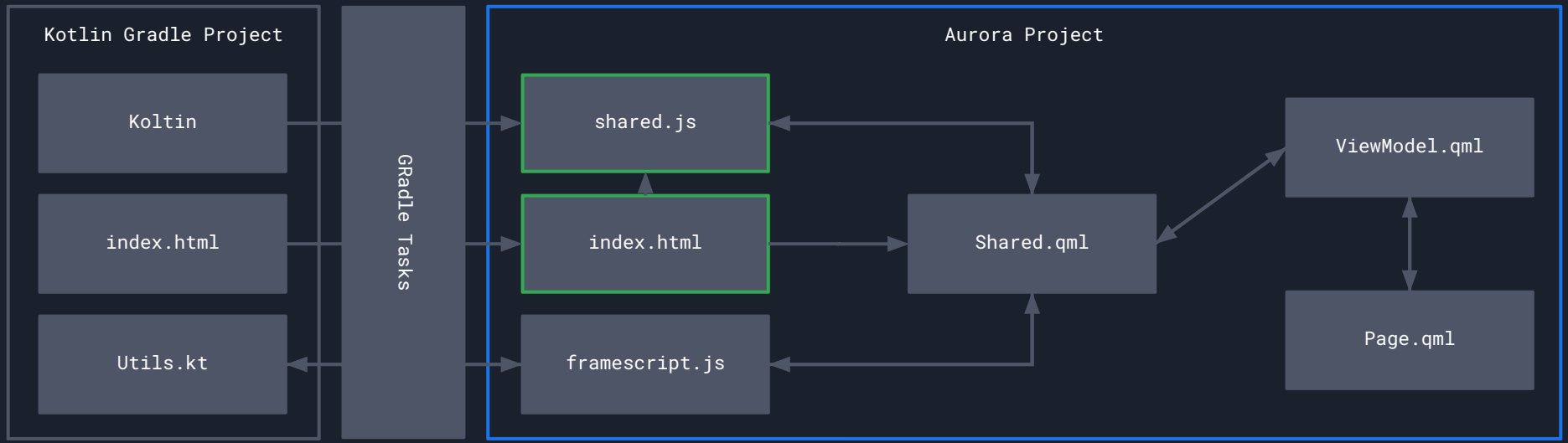
Kotlin JS на Аврора



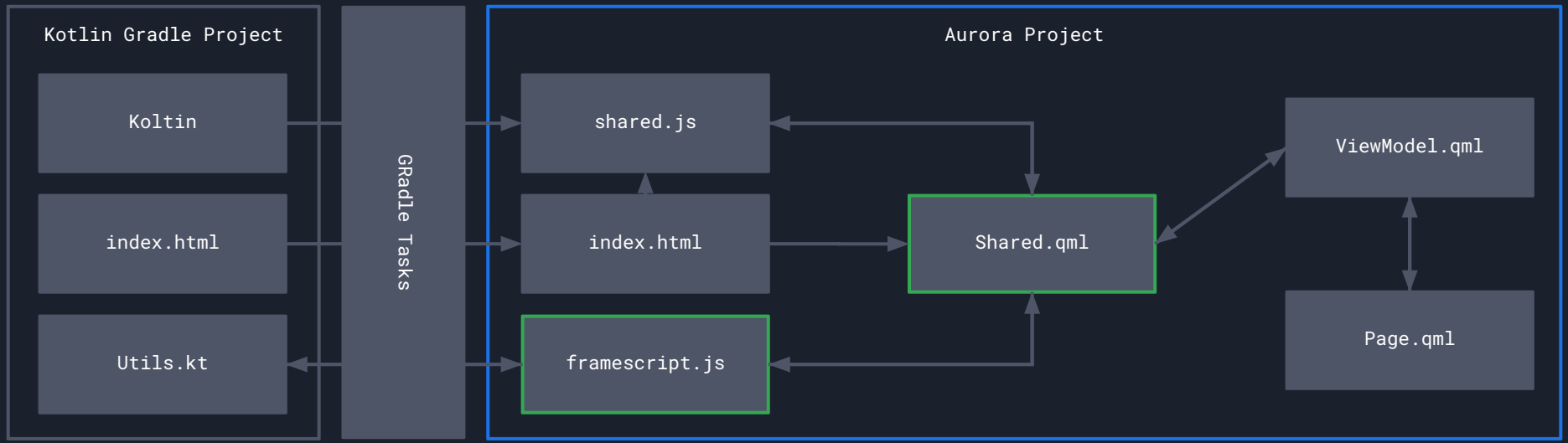
Kotlin JS на Аврора



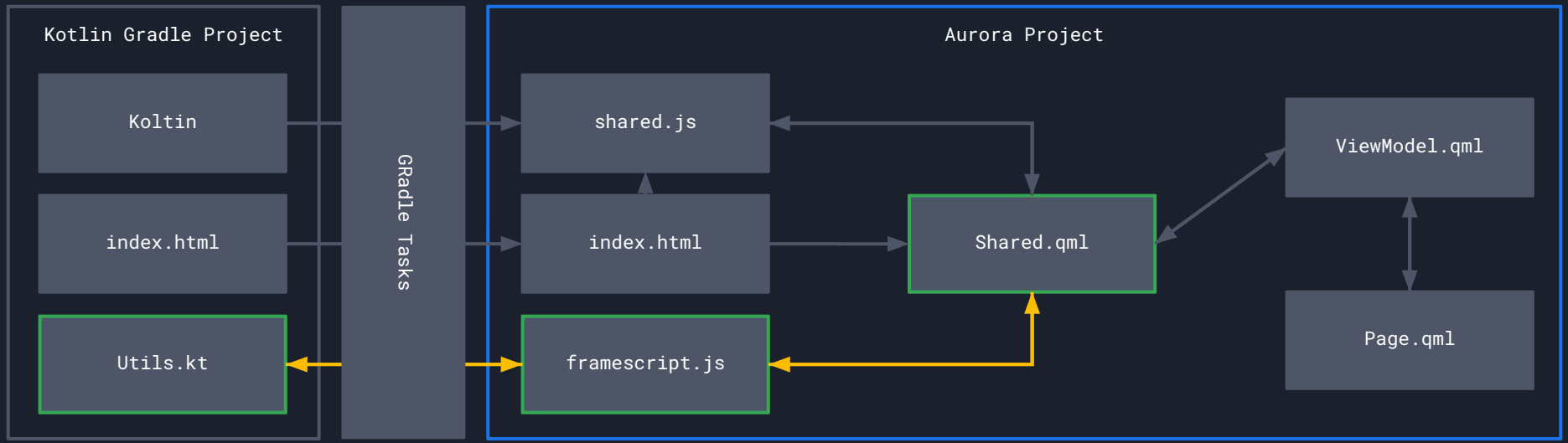
Kotlin JS на Аврора



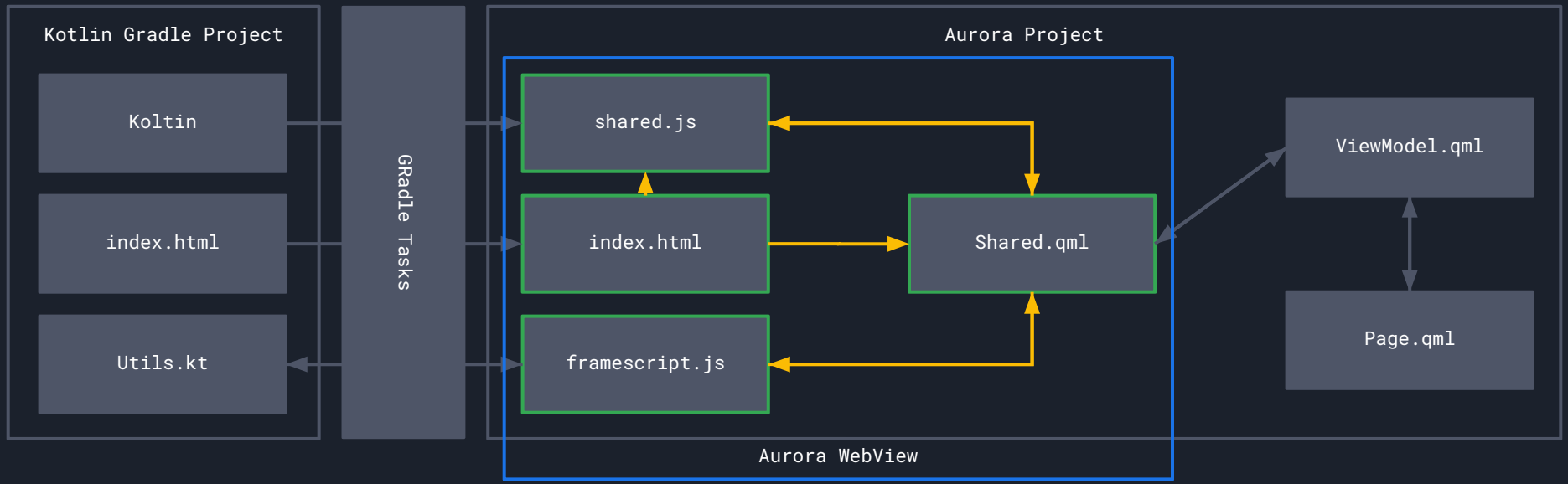
Kotlin JS на Аврора



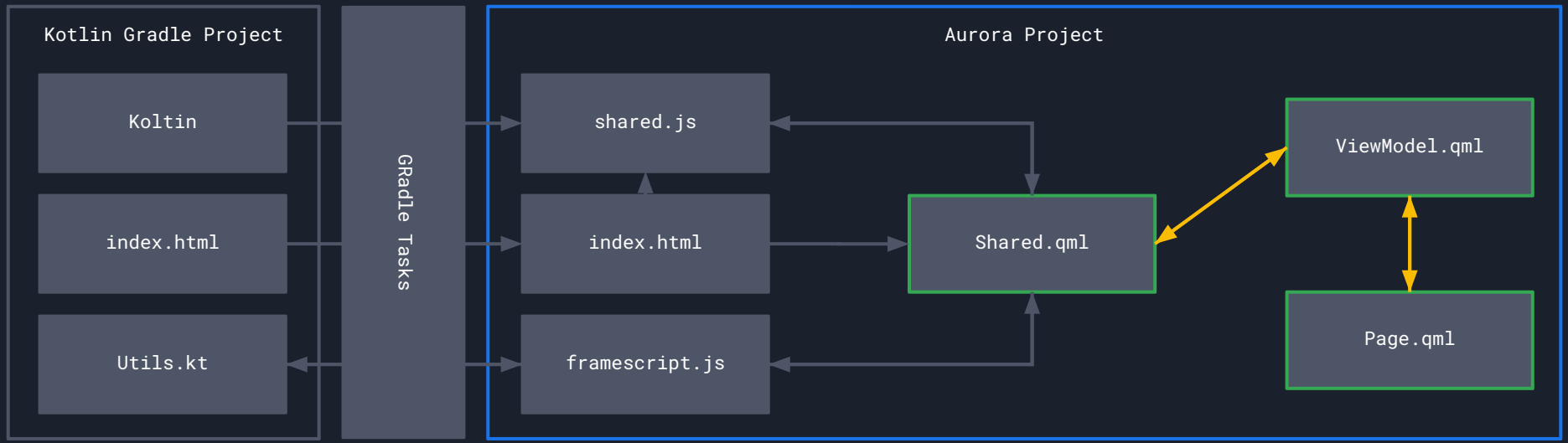
Kotlin JS на Аврора



Kotlin JS на Аврора



Kotlin JS на Аврора



Kotlin JS на Аврора

```
js(IR) {
    moduleName = "shared"
    version = "0.0.1"
    browser()
    binaries.library()
}

sourceSets {
    ...
    val commonMain by getting {
        dependencies {
            implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:$version_coroutines")
            implementation("io.ktor:ktor-client-core:$version_ktor")
            implementation("io.ktor:ktor-serialization-kotlinx-json:$version_ktor")
            implementation("io.ktor:ktor-client-content-negotiation:$version_ktor")
```

Kotlin JS на Аврора

```
sourceSets {
    ...
    val commonMain by getting {
        dependencies {
            implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:$version_coroutines")
            implementation("io.ktor:ktor-client-core:$version_ktor")
            implementation("io.ktor:ktor-serialization-kotlinx-json:$version_ktor")
            implementation("io.ktor:ktor-client-content-negotiation:$version_ktor")
            implementation("org.jetbrains.kotlinx:kotlinx-datetime:$version_datetime")
        }
    }
    ...
    val jsMain by getting {
        dependencies {
            implementation(npm("uuid", "9.0.0"))
```

Kotlin JS на Аврора

```
            implementation("org.jetbrains.kotlinx:kotlinx-datetime:$version_datetime")
        }
    }
    ...
    val jsMain by getting {
        dependencies {
            implementation(npm("uuid", "9.0.0"))
            implementation("io.ktor:ktor-client-js:$version_ktor")

            implementation("app.cash.sqldelight:web-worker-driver:$version_sqldelight")
            implementation(npm("@cashapp/sqldelight-sqljs-worker", version_sqldelight))
            implementation(npm("sql.js", "1.8.0"))
            implementation(devNpm("copy-webpack-plugin", "11.0.0"))
        }
    }
}
```

Kotlin JS на Аврора

```
tasks.register("jsBuildForAurora") {
    dependsOn("jsBrowserProductionWebpack")
    doLast {
        copy {
            from(layout.buildDirectory.dir("dist/js/productionExecutable"))
            into("${rootProject.rootDir}/aurora/KmpWaysToAurora/qml/kmp")
        }
    }
}
```

Kotlin JS на Аврора

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <script type="text/javascript" src="shared.js"></script>
    <title>KMP</title>
</head>
<body onload="shared.com.den3000.kmpwaystoaurora.shared.init()"></body>
</html>
```

Kotlin JS на Аврора

```
Item { // Shared.qml
    ...
    WebView {
        id: webview
        height: parent.height
        width: parent.width
        privateMode: true
        url: Qt.resolvedUrl("index.html")
        visible: false

        onViewInitialized: {
            webview.loadFrameScript(Qt.resolvedUrl("framescript.js"))
            webview.addMessageListener("webview:action")
        }
    }
    ...
}
```

Kotlin JS на Аврора

```
onRecvAsyncMessage: {
    switch (message) {
    case "webview:action":
        try {
            if (data.caller === 'Init') {
                root.completed()
            } else if (root._listeners[data.caller] !== undefined) {
                if (data.response !== undefined) {
                    root._listeners[data.caller][0](data.response)
                } else {
                    root._listeners[data.caller][1](data.error)
                }
            } else { ... }
        } catch (e) { ... }
        break
    }
}
```

Kotlin JS на Аврора

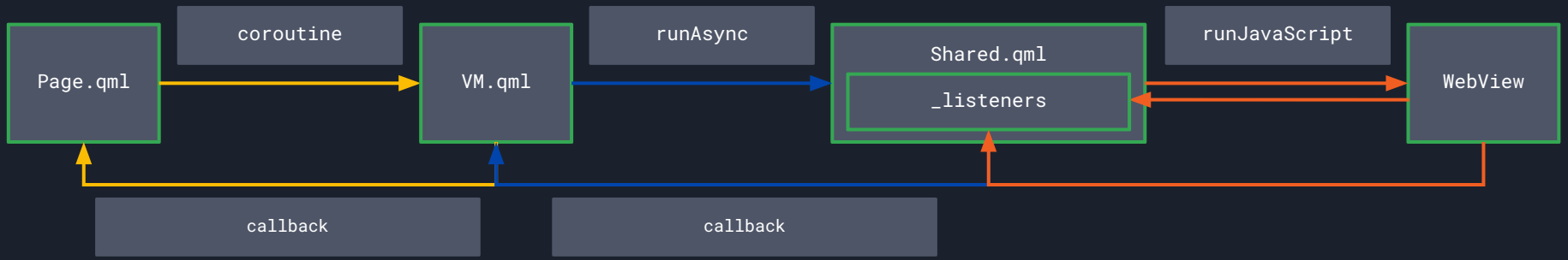
```
// framescript.js
addEventListener("DOMContentLoaded", function (e1) {
    e1.originalTarget.addEventListener(
        "framescript:log",
        function (e2) {
            sendAsyncMessage("webview:action", e2.detail)
        }
    );
});
```

Kotlin JS на Аврора

```
// Utils.kt
private fun <T> _sendEvent(
    caller: String,
    response: T?,
    error: String?,
) {
    document.dispatchEvent(
        CustomEvent(
            type = "framescript:log",
            eventInitDict = CustomEventInit(
               detail = js("{'caller': caller, 'response': response, 'error': error}")
            ),
        )
    )
}
```

Kotlin JS на Аврора

Kotlin JS на Аврора



Kotlin JS на Аврора

```kotlin
// Shared.kt
suspend fun triggerCoroutine(delayInMs: Long): String {
    delay(delayInMs)
    return "Coroutine finished"
}
@OptIn(ExperimentalJsExport::class)
@JsExport
fun coroutineJS(): String {
    val scope = CoroutineScope(getExecutionContext())
    val caller = Uuid.v4()
    scope.async {
        triggerCoroutine(4000)
    }.asPromise().then {
        sendEventResponse(caller, response = it)
    }
    return caller
}
```

Kotlin JS на Аврора

```kotlin
// Shared.kt
suspend fun triggerCoroutine(delayInMs: Long): String {
    delay(delayInMs)
    return "Coroutine finished"
}
@OptIn(ExperimentalJsExport::class)
@JsExport
fun coroutineJS(): String {
    val scope = CoroutineScope(getExecutionContext())
    val caller = Uuid.v4()
    scope.async {
        triggerCoroutine(4000)
    }.asPromise().then {
        sendEventResponse(caller, response = it)
    }
    return caller
}
```

Kotlin JS на Аврора

```
// Shared.kt
suspend fun triggerCoroutine(delayInMs: Long): String {
    delay(delayInMs)
    return "Coroutine finished"
}
@OptIn(ExperimentalJsExport::class)
@JsExport
fun coroutineJS(): String {
    val scope = CoroutineScope(getExecutionContext())
    val caller = Uuid.v4()
    scope.async {
        triggerCoroutine(4000)
    }.asPromise().then {
        sendEventResponse(caller, response = it)
    }
    return caller
}
```

Kotlin JS на Аврора

```
// Page.qml
Button {
    text: qsTr("Coroutines")
    onClicked: {
        strText = "Coroutine started"
        vm.coroutine(function(result){
            strText = result
        })
    }
}
```

Kotlin JS на Аврора

```
// Page.qml
Button {
    text: qsTr("Coroutines")
    onClicked: {
        strText = "Coroutine started"
        vm.coroutine(function(result){
            strText = result
        })
    }
}
```

Kotlin JS на Аврора

```
// VM.qml
KMP.Shared {
    id: libKMPShared
    onCompleted: { console.log("KMP LOADED") }
}

...

function coroutine(callback) {
    libKMPShared.runAsync(
        "shared.com.den3000.kmpwaystoaurora.shared.coroutineJS()",
        callback,
        function(error) { console.log(error) }
    );
}
```

Kotlin JS на Аврора

```
// VM.qml
KMP.Shared {
    id: libKMPShared
    onCompleted: { console.log("KMP LOADED") }
}

...

function coroutine(callback) {
    libKMPShared.runAsync(
        "shared.com.den3000.kmpwaystoaurora.shared.coroutineJS()",
        callback,
        function(error) { console.log(error) }
    );
}
```

Kotlin JS на Аврора

```
// VM.qml
KMP.Shared {
    id: libKMPShared
    onCompleted: { console.log("KMP LOADED") }
}

...

function coroutine(callback) {
    libKMPShared.runAsync(
        "shared.com.den3000.kmpwaystoaurora.shared.coroutineJS()",
        callback,
        function(error) { console.log(error) }
    );
}
```

Kotlin JS на Аврора

```
// Shared.qml
function runAsync(method, result, error) {
    if (method.indexOf("return") === -1) {
        webview.runJavaScript("return " + method, function(data) {
            root._listeners[data] = [result, error]
        }, error);
    } else {
        webview.runJavaScript(method, result, error);
    }
}
```

Kotlin JS на Аврора

```
// Shared.qml
function runAsync(method, result, error) {
    if (method.indexOf("return") === -1) {
        webview.runJavaScript("return " + method, function(data) {
            root._listeners[data] = [result, error]
        }, error);
    } else {
        webview.runJavaScript(method, result, error);
    }
}
```

Kotlin JS на Аврора

```
// Shared.qml
onRecvAsyncMessage: {
    switch (message) {
    case "webview:action":
        try {
            if (data.caller === 'Init') {
                root.completed()
            } else if (root._listeners[data.caller] !== undefined) {
                if (data.response !== undefined) {
                    root._listeners[data.caller][0](data.response)
                } else {
                    root._listeners[data.caller][1](data.error)
                }
            } else { ... }
        } catch (e) { ... }
        break
```

Kotlin JS на Аврора: плюсы

- Быстродействие
- Многие библиотеки которые доступны для KJS тут тоже оказываются доступны: корутины, сериализация, ktor
- Сравнительная простота
- Уже работает на любой версии Аврора ОС

Kotlin JS на Аврора: минусы

- Специфика работы с JS и вэбом в целом
- Неочевидные баги при использовании runJavaScript
- Некоторые библиотеки будут не доступны, в силу ограниченности возможностей Aurora WebView, например SQLDelight

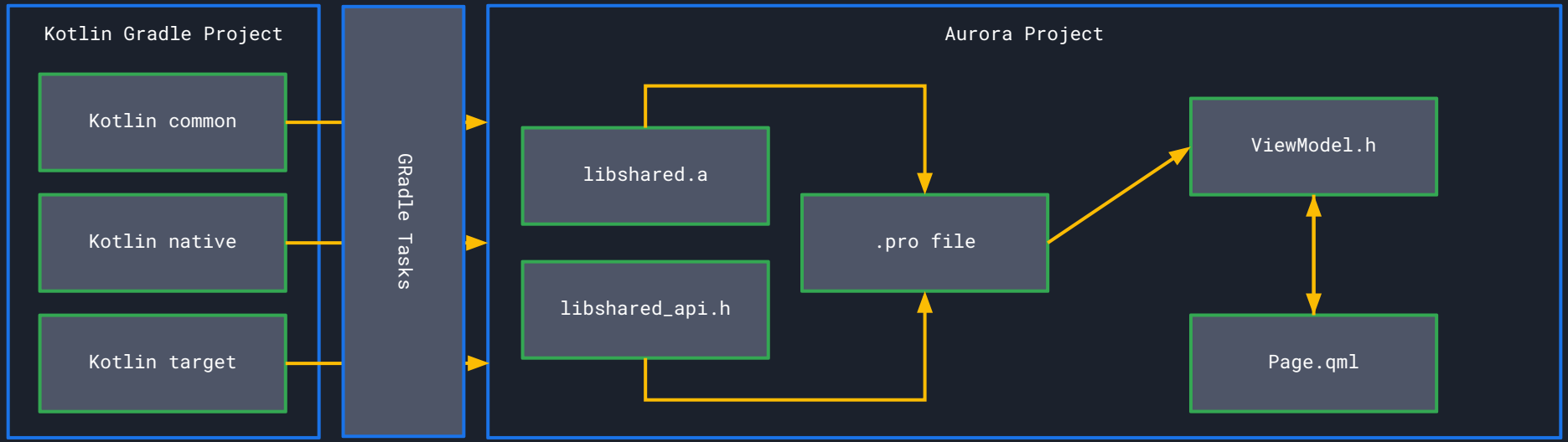
План

1. Краткий обзор KMP и подходов к созданию общего кода
2. Kotlin JS на Аврора
3. Kotlin Native на Аврора
4. Kotlin JVM на Аврора с помощью Java to Native
5. Проблемы с портированием Compose на Аврора
6. Итоги

Kotlin Native на Аврора

- Компилируем Kotlin Native код в статическую C библиотеку и соответствующий набор хэдеров
- На стороне Аврора ОС пишем ViewModel на C++
- Импортируем библиотеку
- Вызываем методы

Kotlin JS на Аврора



Kotlin Native на Аврора

```
kotlin {
    ...
    mingwX64("nativeWinX64") {
        binaries {
            staticLib {  }
        }
    }
    linuxX64("nativeLinuxX64") {
        binaries {
            staticLib {  }
        }
    }
    linuxArm64("nativeLinuxArm64") {
        binaries {
            staticLib {  }
        }
    }
```

Kotlin Native на Аврора

```
sourceSets {
        val commonMain by getting {
            dependencies {
                implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:$version_coroutines")
                implementation("io.ktor:ktor-client-core:$version_ktor")
                implementation("io.ktor:ktor-serialization-kotlinx-json:$version_ktor")
                implementation("io.ktor:ktor-client-content-negotiation:$version_ktor")
                implementation("org.jetbrains.kotlinx:kotlinx-datetime:$version_datetime")
            }
        }
        ...
        val nativeWinX64Main by getting {
            dependencies {
                implementation("io.ktor:ktor-client-winhttp:$version_ktor")
                implementation("app.cash.sqldelight:native-driver:$version_sqldelight")
            }
        }
```

Kotlin Native на Аврора

```
    val nativeLinuxX64Main by getting {
        dependencies {
            implementation("io.ktor:ktor-client-curl:$version_ktor")
            implementation("app.cash.sqldelight:native-driver:$version_sqldelight")
        }
    }

    val nativeLinuxArm64Main by getting  {
        dependencies {
            // Not yet available
        }
    }
}
```

Kotlin Native на Аврора

```
...
tasks.register<Copy>(tnCopyAndLinkReleaseLinuxArm64) {
    dependsOn(tasks.getByName(tnLinkReleaseLinuxArm64))
    copy("nativeLinuxArm64", "releaseStatic", "aarch64", "release")
}

tasks.register("linkAndCopySharedForAllTargets") {
    dependsOn(
        tasks.getByName(tnCopyAndLinkDebugLinuxX64),
        tasks.getByName(tnCopyAndLinkDebugLinuxArm64),
        tasks.getByName(tnCopyAndLinkReleaseLinuxX64),
        tasks.getByName(tnCopyAndLinkReleaseLinuxArm64)
    )
}
```

Kotlin Native на Аврора

```
contains(QMAKE_HOST.arch, armv7l):{ error("Unsupported architecture") }
contains(QMAKE_HOST.arch, x86_64):{ SHARED_LIB_ARCH_TYPE_PATH=lib_shared/x86_64 }
contains(QMAKE_HOST.arch, aarch64):{ SHARED_LIB_ARCH_TYPE_PATH=lib_shared/aarch64 }
CONFIG(debug, debug|release):{ SHARED_LIB_BUILD_TYPE_PATH=debug }
CONFIG(release, debug|release):{ SHARED_LIB_BUILD_TYPE_PATH=release }

# path to libshared.a
SHARED_LIB_PATH=$${SHARED_LIB_ARCH_TYPE_PATH}/$${SHARED_LIB_BUILD_TYPE_PATH}
```

Kotlin Native на Аврора

```
...
HEADERS += \
    $${SHARED_LIB_PATH}/libshared_api.h \
...
unix:!macx: LIBS += -L$$PWD/$${SHARED_LIB_PATH} -lshared -lcurl -lsqlite3 \

INCLUDEPATH += $$PWD/$${SHARED_LIB_PATH} \

DEPENDPATH += $$PWD/$${SHARED_LIB_PATH} \

unix:!macx: PRE_TARGETDEPS += $$PWD/$${SHARED_LIB_PATH}/libshared.a
```

Kotlin Native на Аврора

```kotlin
actual fun platform() = "Shared Linux X64"

@Serializable
data class DataClass(
    val int: Int,
    val string: String,
)

actual fun getDataClass(): DataClass {
    return DataClass(
        int = 10,
        string = "some string"
    )
}
```

Kotlin Native на Аврора

```
fun triggerLambda(callback: () -> Unit) {
    callback()
}

actual fun serializeToString(dc: DataClass): String {
    return Json.encodeToString(dc)
}

actual fun deserializeFromString(str: String): DataClass {
    return Json.decodeFromString(str)
}
```

Kotlin Native на Аврора

```
auto klib = libshared_symbols()->kotlin.root.com.den3000.kmpwaystoaurora.shared;

auto ktText = klib.platform();
auto ktDataClass1 = klib.getDataClass();
auto ktDataClass1Str = klib.DataClass.toString(ktDataClass1);

auto ktDataClass2 = klib.DataClass.DataClass(
            2,
            "some aurora string"
            );
auto ktDataClass2Int = klib.DataClass.get_int(ktDataClass2);
auto ktDataClass2Str = klib.DataClass.get_string(ktDataClass2);

auto dc = klib.getDataClass();
auto str = klib.serializeToString(dc);
dc = klib.deserializeFromString(str);
updateText(klib.DataClass.toString(dc));
```

Kotlin Native на Аврора

```
auto klib = libshared_symbols()->kotlin.root.com.den3000.kmpwaystoaurora.shared;

auto ktText = klib.platform();
auto ktDataClass1 = klib.getDataClass();
auto ktDataClass1Str = klib.DataClass.toString(ktDataClass1);

auto ktDataClass2 = klib.DataClass.DataClass(
            2,
            "some aurora string"
            );
auto ktDataClass2Int = klib.DataClass.get_int(ktDataClass2);
auto ktDataClass2Str = klib.DataClass.get_string(ktDataClass2);

auto dc = klib.getDataClass();
auto str = klib.serializeToString(dc);
dc = klib.deserializeFromString(str);
updateText(klib.DataClass.toString(dc));
```

Kotlin Native на Аврора

```
auto klib = libshared_symbols()->kotlin.root.com.den3000.kmpwaystoaurora.shared;

auto ktText = klib.platform();
auto ktDataClass1 = klib.getDataClass();
auto ktDataClass1Str = klib.DataClass.toString(ktDataClass1);

auto ktDataClass2 = klib.DataClass.DataClass(
            2,
            "some aurora string"
            );
auto ktDataClass2Int = klib.DataClass.get_int(ktDataClass2);
auto ktDataClass2Str = klib.DataClass.get_string(ktDataClass2);

auto dc = klib.getDataClass();
auto str = klib.serializeToString(dc);
dc = klib.deserializeFromString(str);
updateText(klib.DataClass.toString(dc));
```

Kotlin Native на Аврора

```
auto klib = libshared_symbols()->kotlin.root.com.den3000.kmpwaystoaurora.shared;

auto ktText = klib.platform();
auto ktDataClass1 = klib.getDataClass();
auto ktDataClass1Str = klib.DataClass.toString(ktDataClass1);

auto ktDataClass2 = klib.DataClass.DataClass(
            2,
            "some aurora string"
            );
auto ktDataClass2Int = klib.DataClass.get_int(ktDataClass2);
auto ktDataClass2Str = klib.DataClass.get_string(ktDataClass2);

auto dc = klib.getDataClass();
auto str = klib.serializeToString(dc);
dc = klib.deserializeFromString(str);
updateText(klib.DataClass.toString(dc));
```

Kotlin Native на Аврора

```
auto klib = libshared_symbols()->kotlin.root.com.den3000.kmpwaystoaurora.shared;

auto ktText = klib.platform();
auto ktDataClass1 = klib.getDataClass();
auto ktDataClass1Str = klib.DataClass.toString(ktDataClass1);

auto ktDataClass2 = klib.DataClass.DataClass(
            2,
            "some aurora string"
            );
auto ktDataClass2Int = klib.DataClass.get_int(ktDataClass2);
auto ktDataClass2Str = klib.DataClass.get_string(ktDataClass2);

auto dc = klib.getDataClass();
auto str = klib.serializeToString(dc);
dc = klib.deserializeFromString(str);
updateText(klib.DataClass.toString(dc));
```

Kotlin Native на Аврора

```kotlin
// kotlin
fun triggerLambda(callback: () -> Unit) {
    callback()
}
```

```c
// c
void (*triggerLambda)(libshared_kref_kotlin_Function0 callback);

typedef struct {
  libshared_KNativePtr pinned;
} libshared_kref_kotlin_Function0;
```

Kotlin Native на Аврора

```kotlin
// kotlin
@OptIn(ExperimentalForeignApi::class)
fun triggerLambdaCfptr(
    cfptr: CPointer<CFunction<(COpaquePointer) -> Unit>>,
    data: COpaquePointer
) {
    triggerLambda {
        cfptr.invoke(data)
    }
}

// c
void (*triggerLambdaCfptr)(void* cfptr, void* data);
```

Kotlin Native на Аврора

```
struct res {
    ...
} result {...};
auto noCapture = [](void * data) {
    auto pResult = reinterpret_cast<res *>(data);
    ...
};
typedef void(*NormalFuncType)(void * );
NormalFuncType noCaptureLambdaPtr = noCapture;
klib.triggerLambdaCfptr((libshared_KNativePtr)noCaptureLambdaPtr, &result);
```

Kotlin Native на Аврора

```
struct res {
    ...
} result {...};
auto noCapture = [](void * data) {
    auto pResult = reinterpret_cast<res *>(data);
    ...
};
typedef void(*NormalFuncType)(void * );
NormalFuncType noCaptureLambdaPtr = noCapture;
klib.triggerLambdaCfptr((libshared_KNativePtr)noCaptureLambdaPtr, &result);
```

Kotlin Native на Аврора

```
struct res {
    ...
} result {...};
auto noCapture = [](void * data) {
    auto pResult = reinterpret_cast<res *>(data);
    ...
};
typedef void(*NormalFuncType)(void * );
NormalFuncType noCaptureLambdaPtr = noCapture;
klib.triggerLambdaCfptr((libshared_KNativePtr)noCaptureLambdaPtr, &result);
```

Kotlin Native на Аврора

```
struct res {
    ...
} result {...};
auto noCapture = [](void * data) {
    auto pResult = reinterpret_cast<res *>(data);
    ...
};
typedef void(*NormalFuncType)(void * );
NormalFuncType noCaptureLambdaPtr = noCapture;
klib.triggerLambdaCfptr((libshared_KNativePtr)noCaptureLambdaPtr, &result);
```

Kotlin Native на Аврора

```
suspend fun getKtorIoWelcomePageAsText() : String {
    val client = getHttpRequestClient() ?: return "NO HttpClient AVAILABLE"
    val response = client.get("https://ktor.io/docs/welcome.html")
    val responseText = response.bodyAsText()
    client.close()
    return responseText
}
```

Kotlin Native на Аврора

```
@OptIn(ExperimentalForeignApi::class)
fun getKtorIoWelcomePageAsTextCfptr(
    cfptr: CPointer<CFunction<(COpaquePointer, CValuesRef<ByteVar>) -> Unit>>,
    data: COpaquePointer
) {
    val scope = CoroutineScope(getExecutionContext())
    scope.launch {
        val result = getKtorIoWelcomePageAsText()
        cfptr.invoke(data, result.cstr)
    }
}
```

Kotlin Native на Аврора

```
auto klib = libshared_symbols()->kotlin.root.com.den3000.kmpwaystoaurora.shared;

auto noCapture = [](void * data, const char * text) {
    auto that = reinterpret_cast<KotlinNativeVM *>(data);
    that->updateText(text);
};
typedef void(*NormalFuncType)(void *, const char *);
NormalFuncType noCaptureLambdaPtr = noCapture;

updateText("Request started");
klib.getKtorIoWelcomePageAsTextCfptr((libshared_KNativePtr)noCaptureLambdaPtr, this);
```

Kotlin Native на Аврора: плюсы

- Простота и удобство, за счёт gradle и кросс-компиляции
- Гибкость в плане экспортируемого кода
- Хорошие перспективы развития

Kotlin Native на Аврора: минусы

- Сейчас в основном доступно только для эмулятора
- Вопросы с быстродействием, будет меняться по мере развития компилятора
- Ограниченность доступных библиотек и зависимостей

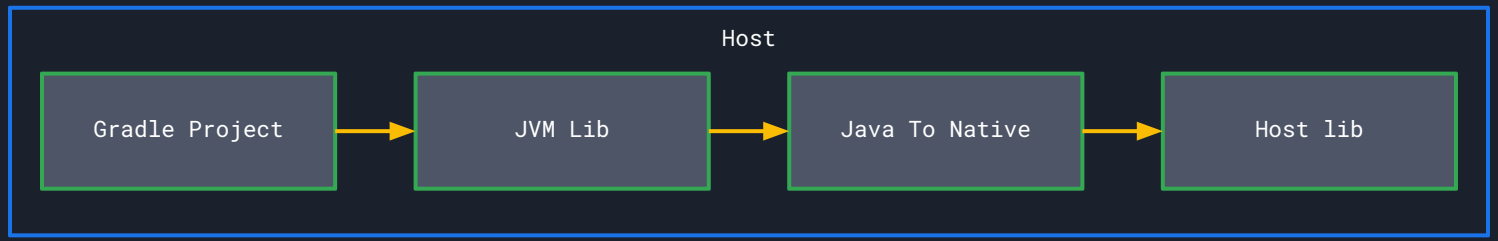
План

1. Краткий обзор KMP и подходов к созданию общего кода
2. Kotlin JS на Аврора
3. Kotlin Native на Аврора
4. Kotlin JVM на Аврора с помощью Java to Native
5. Проблемы с портированием Compose на Аврора
6. Итоги

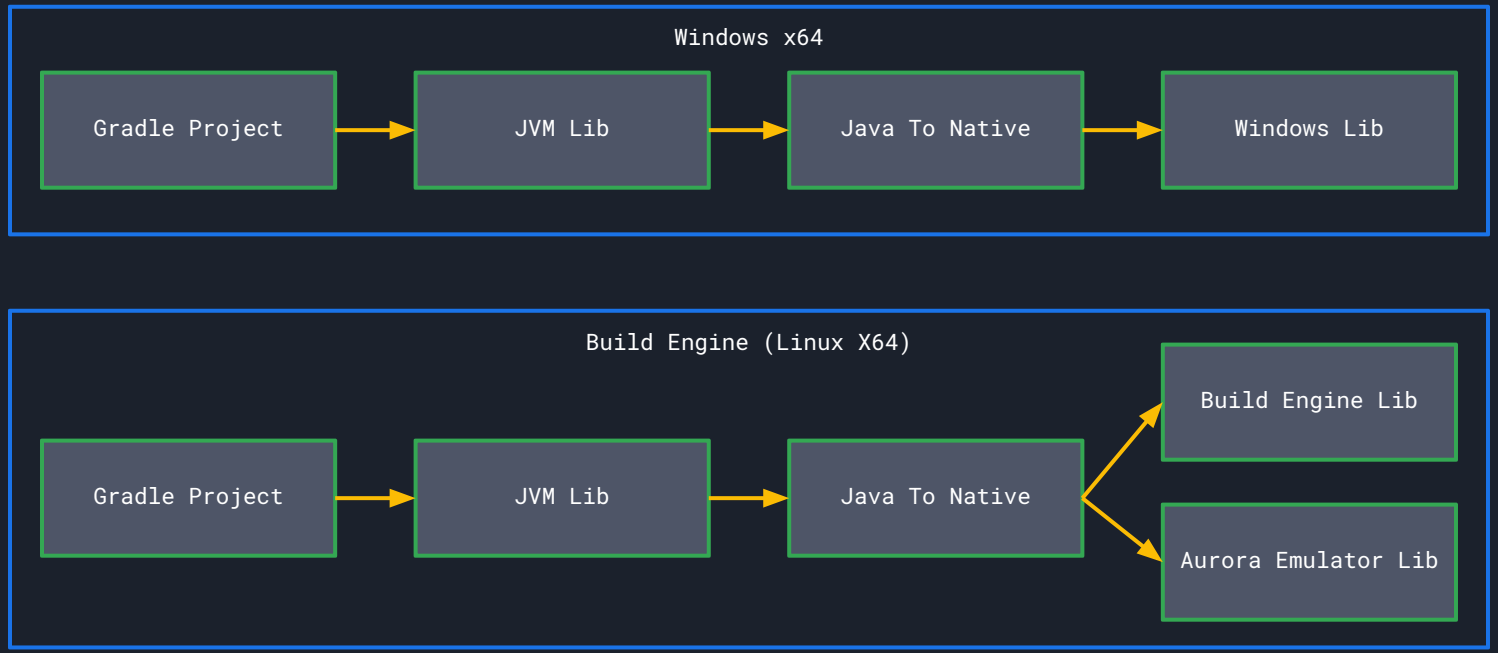
Kotlin JVM на Аврора с помощью Java to Native

- Преобразуем Kotlin JVM код в C библиотеку и соответствующий набор хэдеров
- На стороне Аврора ОС пишем ViewModel на C++
- Импортируем библиотеку
- Вызываем методы

Kotlin JVM на Аврора с помощью Java to Native



Kotlin JVM на Аврора с помощью Java to Native



Kotlin JVM на Аврора с помощью Java to Native

```
plugins {
    ...
    id("org.graalvm.buildtools.native") version "0.10.1"
}

kotlin {
    jvm { withJava() }
    ...
    sourceSets {
        ...
        val jvmMain by getting {
            dependencies {
                implementation("org.graalvm.sdk:graal-sdk:24.0.1")
            }
        }
    }
}
```

Kotlin JVM на Аврора с помощью Java to Native

```
graalvmNative {
    toolchainDetection.set(false)
    binaries{
        named("main") {
            mainClass.set("com.den3000.kmpwaystoaurora.desktop.Jtn")
            buildArgs(
                // build as lib, not executable
                "--shared",
            )
        }
    }

    ...
}
```

Kotlin JVM на Аврора с помощью Java to Native

```
graalvmNative {
    ...
    agent{
        defaultMode.set("standard")

        metadataCopy {
            inputTaskNames.add("run") // Tasks previously executed with the agent attached.
            outputDirectories.add("src/main/resources/META-INF/native-image")
            mergeWithExisting.set(true)
        }
    }
}
```

Kotlin JVM на Аврора с помощью Java to Native

```
DESKTOP_LIB_PATH=lib_desktop/x86_64

desktop_library_install.path = /usr/share/com.den3000.kmpwaystoaurora.KmpWaysToAurora/lib/
desktop_library_install.files = $$PWD/$${DESKTOP_LIB_PATH}/*.so*
desktop_library_install.CONFIG = no_check_exist
INSTALLS += desktop_library_install

unix:!macx: LIBS += ...
    -L$$PWD/$${DESKTOP_LIB_PATH}/ -ldesktop

INCLUDEPATH += ...
    $$PWD/$${DESKTOP_LIB_PATH}/
DEPENDPATH += ...
    $$PWD/$${DESKTOP_LIB_PATH}/
```

Kotlin JVM на Аврора с помощью Java to Native

```
%define __provides_exclude_from ^%{_datadir}/.*$
%define __requires_exclude ^libdesktop.*$
```

Kotlin JVM на Аврора с помощью Java to Native

```
fun runTestTwo(
    driverFactory: DriverFactory,
    started: () -> Unit
): Flow<String> {
    return flow {
        ...
    }
}
```

Kotlin JVM на Аврора с помощью Java to Native

```kotlin
// Jtn.kt
interface TestTwoJtnCallback {
    fun invoke(str: String)
}
fun test2_jtn(callback: TestTwoJtnCallback) {
    CoroutineScope(getExecutionContext()).launch {
        val totalTime = MutableStateFlow<Long>(0)
        val start = MutableStateFlow(getTimeMark())
        val df = DriverFactory()
        runTestTwo(df, started = {
            start.update { getTimeMark() }
        }).collect { text ->
            totalTime.update { getDiffMs(start.value) }
            callback.invoke("Time spent: ${totalTime.value} ms\n" + text.take(40))
        }
    }
}
```

Kotlin JVM на Аврора с помощью Java to Native

```
// Jtn.java
interface IStringCallback extends CFunctionPointer {
    @InvokeCFunctionPointer
    void invoke(CCharPointer str, Pointer data);
}
public class Jtn {
    ...
    @CEntryPoint(name = "jtn_test2")
    private static void test2(IsolateThread thread, IStringCallback callback, Pointer data) {
        JtnKt.test2_jtn(new TestTwoJtnCallback() {
            @Override
            public void invoke(@NotNull String str) {
                callback.invoke(CTypeConversion.toCString(str).get(), data);
            }
        });
    }
}
```

Kotlin JVM на Аврора с помощью Java to Native

```
graal_isolate_t *isolate = NULL;
graal_isolatethread_t *thread = NULL;

if (graal_create_isolate(NULL, &isolate, &thread) != 0) {
    qDebug() << "initialization error\n";
}

auto noCapture = [](char * text, size_t p) {
    auto vm = reinterpret_cast<KotlinJtnVM *>(p);
    vm->updateText(text);
};
typedef void(*NormalFuncType)(char *, size_t);
NormalFuncType noCaptureLambdaPtr = noCapture;

jtn_test2(thread, (void *) noCaptureLambdaPtr, (size_t)this);

//        graal_tear_down_isolate(thread);
```

Kotlin JVM на Аврора с помощью Java to Native

```
graal_isolate_t *isolate = NULL;
graal_isolatethread_t *thread = NULL;

if (graal_create_isolate(NULL, &isolate, &thread) != 0) {
    qDebug() << "initialization error\n";
}

auto noCapture = [](char * text, size_t p) {
    auto vm = reinterpret_cast<KotlinJtnVM *>(p);
    vm->updateText(text);
};
typedef void(*NormalFuncType)(char *, size_t);
NormalFuncType noCaptureLambdaPtr = noCapture;

jtn_test2(thread, (void *) noCaptureLambdaPtr, (size_t)this);

//        graal_tear_down_isolate(thread);
```

Kotlin JVM на Аврора с помощью Java to Native

```
graal_isolate_t *isolate = NULL;
graal_isolatethread_t *thread = NULL;

if (graal_create_isolate(NULL, &isolate, &thread) != 0) {
    qDebug() << "initialization error\n";
}

auto noCapture = [](char * text, size_t p) {
    auto vm = reinterpret_cast<KotlinJtnVM *>(p);
    vm->updateText(text);
};
typedef void(*NormalFuncType)(char *, size_t);
NormalFuncType noCaptureLambdaPtr = noCapture;

jtn_test2(thread, (void *) noCaptureLambdaPtr, (size_t)this);

//        graal_tear_down_isolate(thread);
```

Kotlin JVM на Аврора с помощью Java to Native: плюсы

- Уже сейчас работает на основных таргетах (эмулятор и девайсы с aarm64)
- Доступность многих библиотек JVM
- Производительность

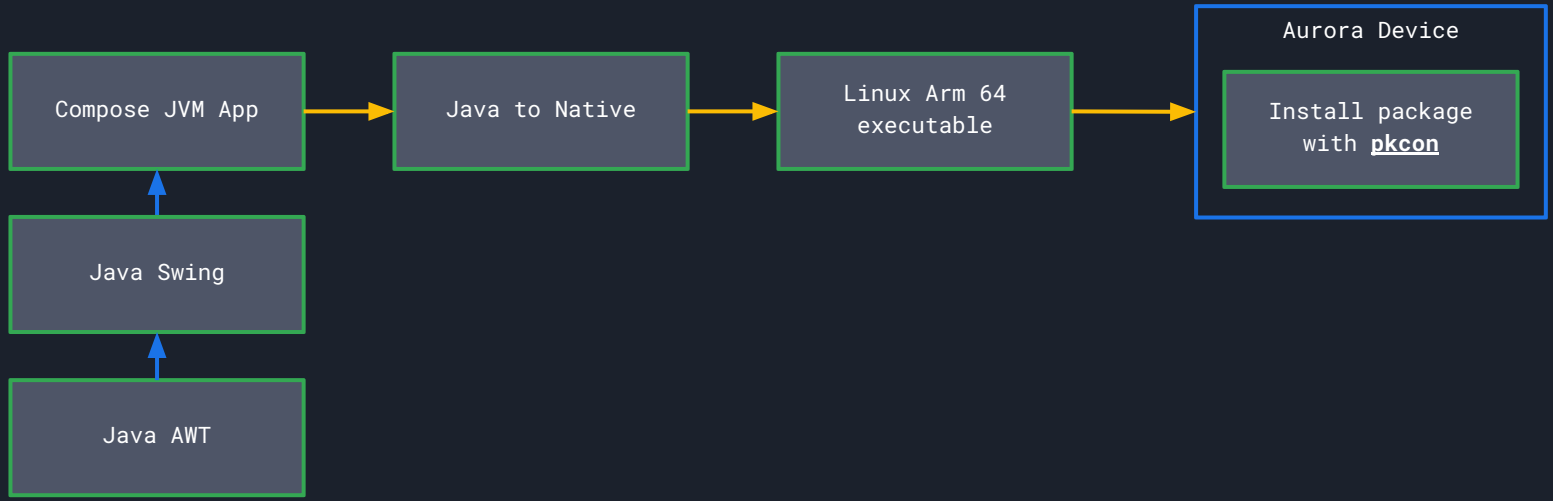
Kotlin JVM на Аврора с помощью Java to Native: минусы

- Сложности сборки проекта в связи с отсутствием кросскомпиляции
- Необходимость писать Java врапперы

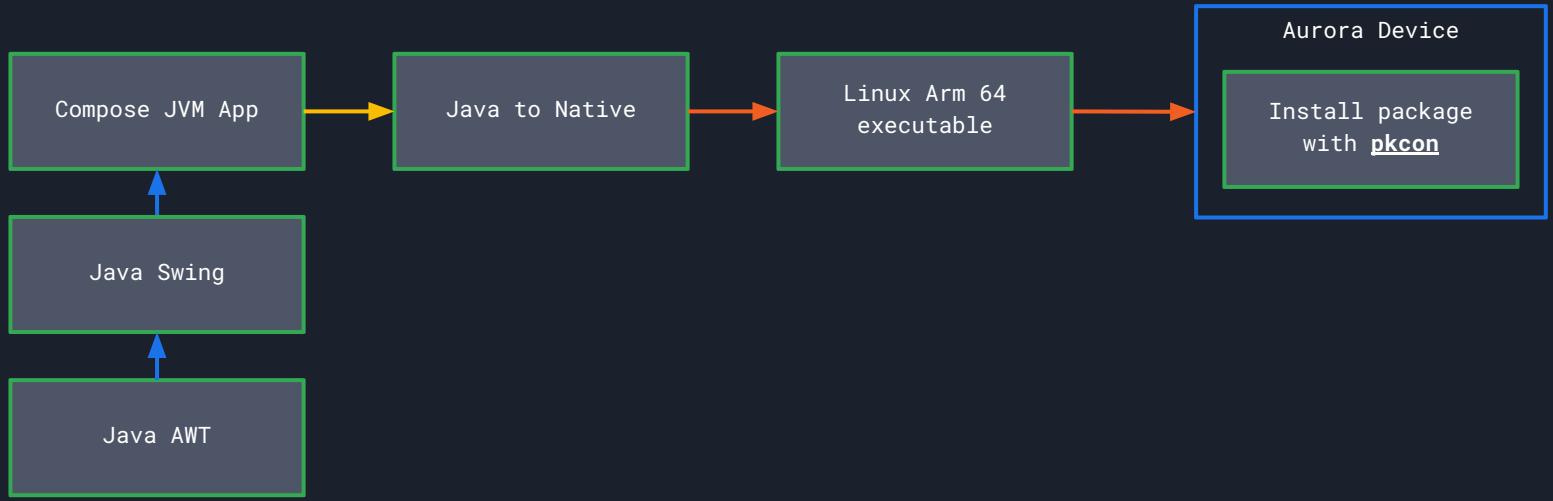
План

1. Краткий обзор KMP и подходов к созданию общего кода
2. Kotlin JS на Аврора
3. Kotlin Native на Аврора
4. Kotlin JVM на Аврора с помощью Java to Native
5. Проблемы с портированием Compose на Аврора
6. Итоги

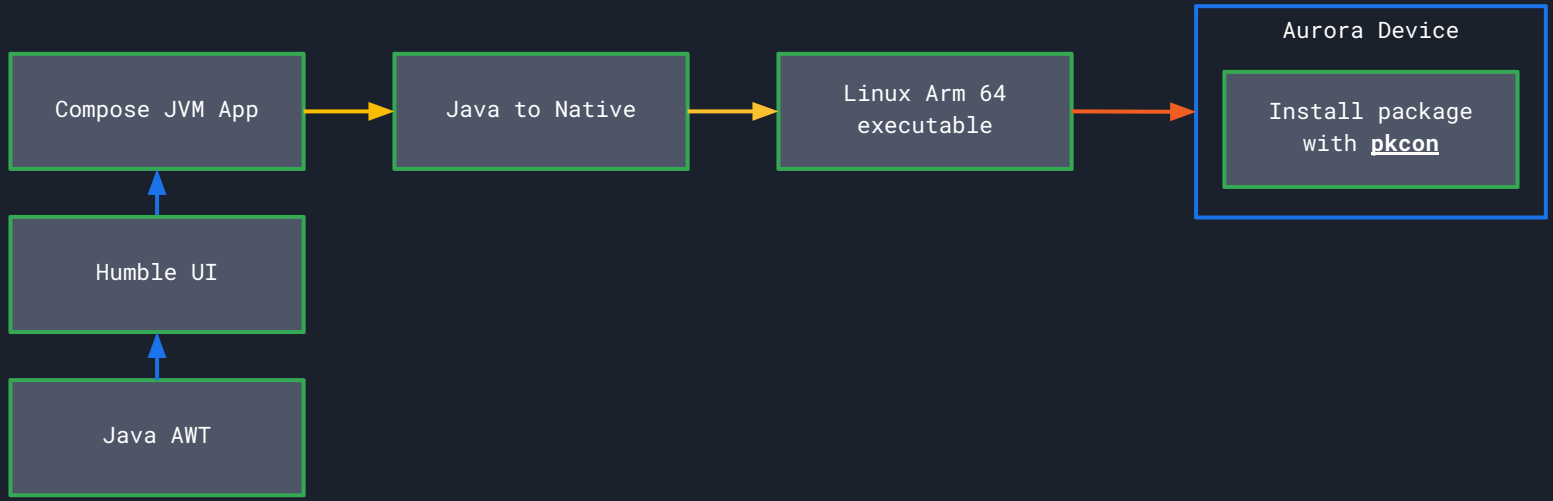
Проблемы с портированием Compose на Аврора



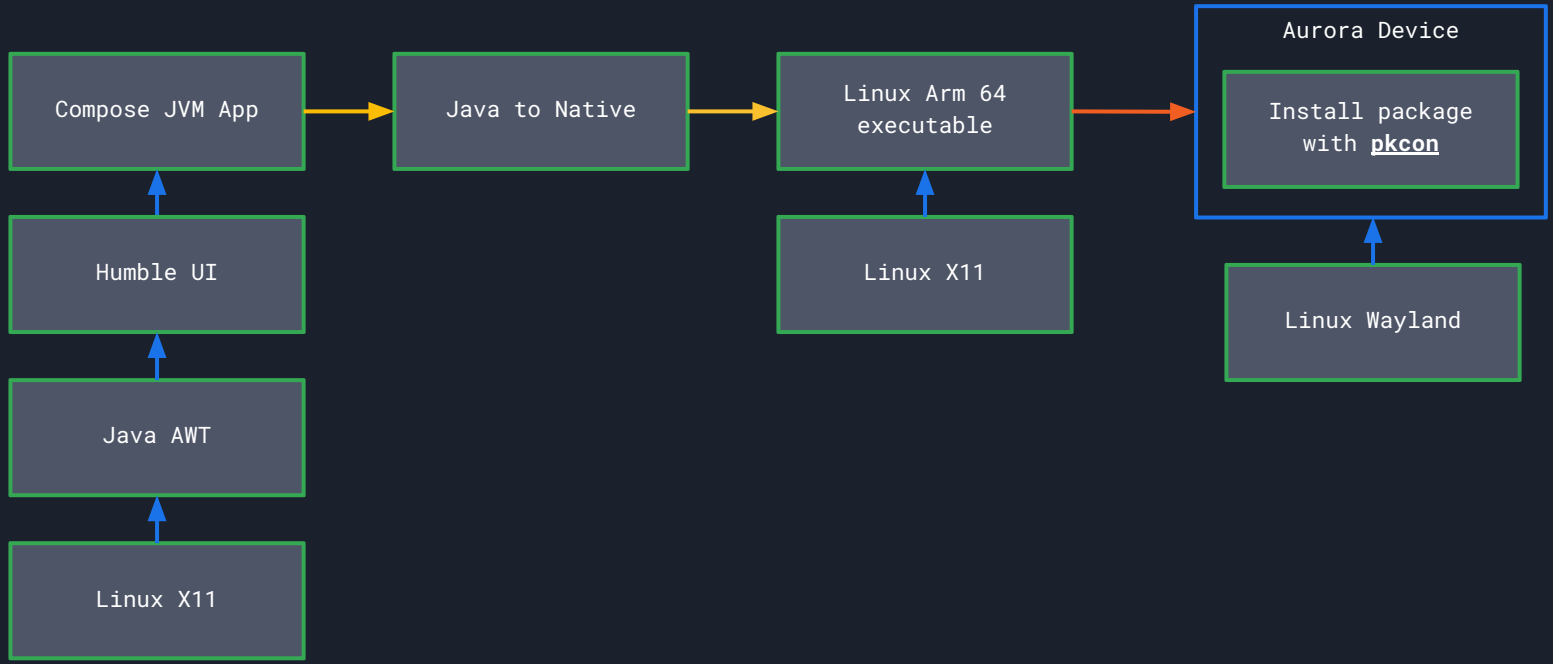
Проблемы с портированием Compose на Аврора



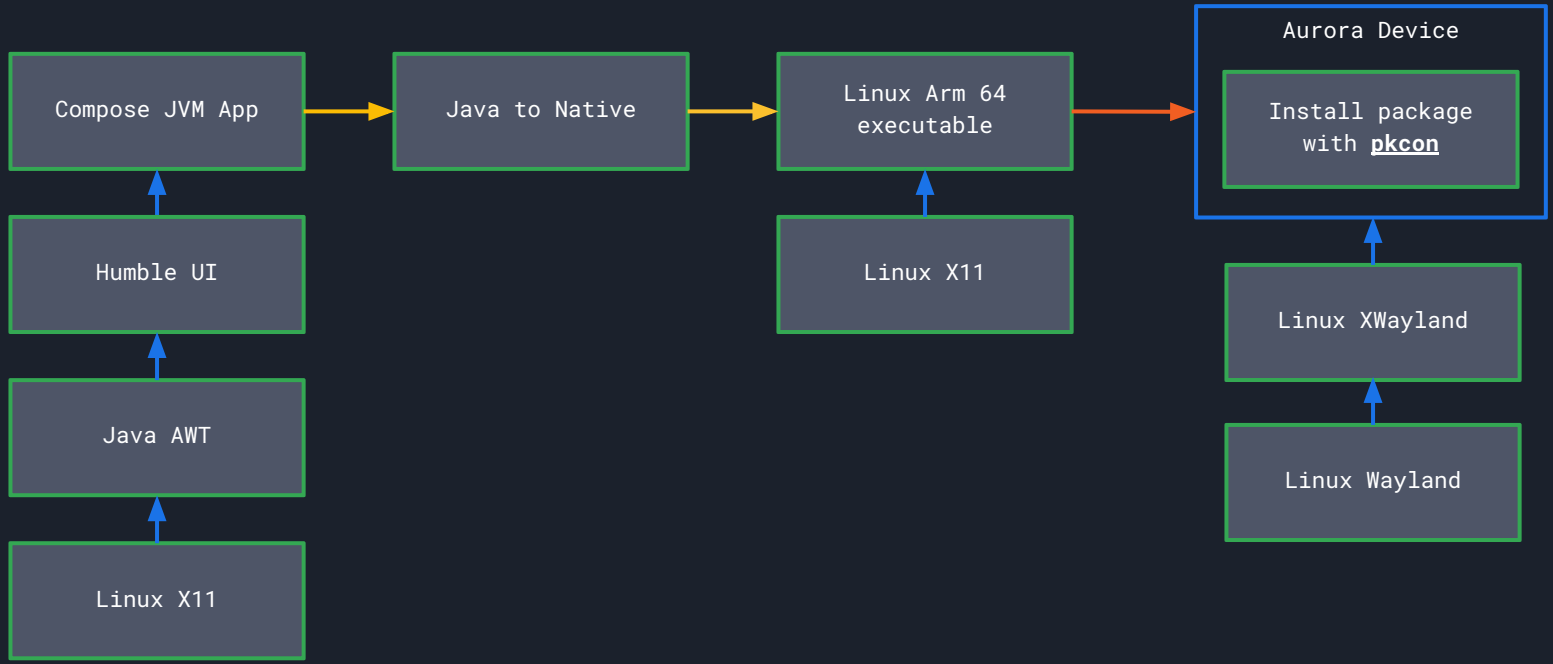
Проблемы с портированием Compose на Аврора



Проблемы с портированием Compose на Аврора



Проблемы с портированием Compose на Аврора



План

1. Краткий обзор KMP и подходов к созданию общего кода
2. Kotlin JS на Аврора
3. Kotlin Native на Аврора
4. Kotlin JVM на Аврора с помощью Java to Native
5. Проблемы с портированием Compose на Аврора
6. Итоги

Итоги

- Используйте KJS + QML, если ваше приложение по сути тонкий клиент к бэкенду, вам не нужно локальное хранение данных, и у вас есть люди которые разбираются в Web разработке

Итоги

- Если ваше приложение полагается на большое количество зависимостей, которые вряд ли получат поддержку KMP в ближайшем будущем, в команде есть джавист, а релизить приложение под Аврора ОС надо было ещё вчера - то использование Graal VM ваш выбор

Итоги

- Если вы любите KMP, и у вас уже есть приложение которое построено на Coroutines / Ktor / SQLDelight (Room) и т.п. подобное, то есть что уже прям поддерживает KMP, или вы только пишите новое приложение, то стоит попробовать уже добавить и target для Аврора ОС - интересный опыт гарантирован

Спасибо за внимание

1. Скачать Aurora IDE
   https://developer.auroraos.ru/#tree
2. Исходный код KmpWaysToAurora
   https://github.com/den3000/KmpWaysToAurora
3. ТГ https://t.me/glavkod