Kotlin IR

Прошлое, Настоящее и Будущее

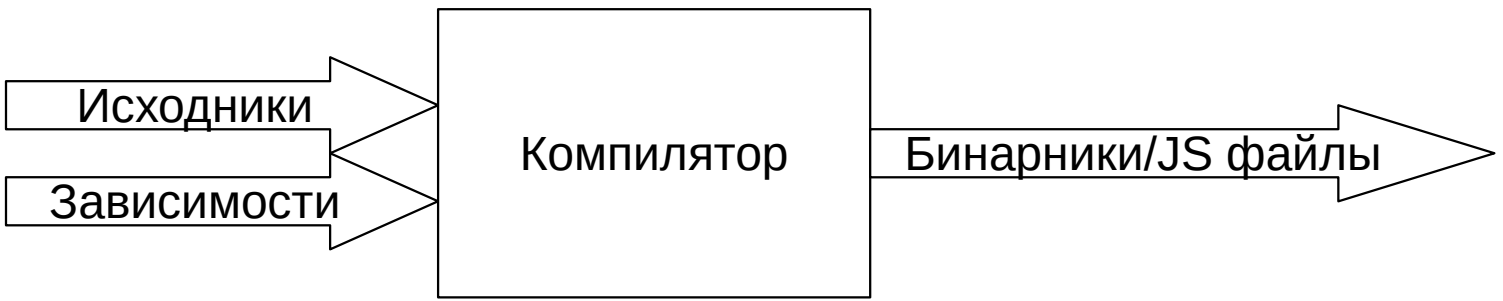
О себе

- Ильмир Усманов
- Kotlin JVM @ JetBrains
- Coroutines, Inline Classes

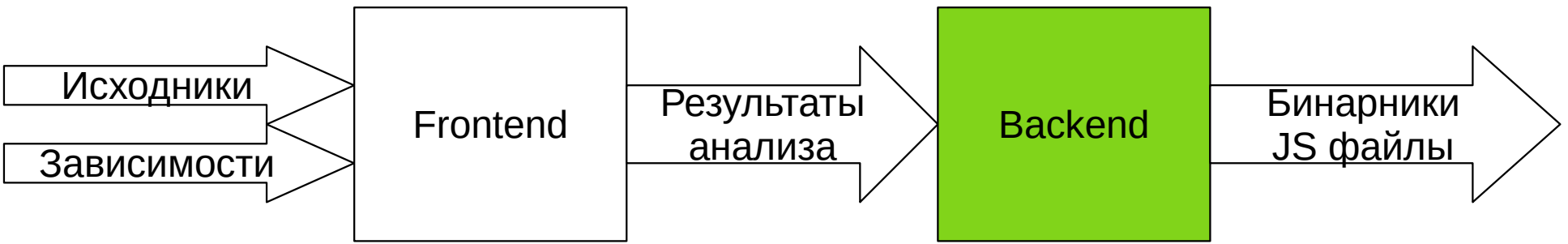
План

- Мультиплатформа – три бекенда, много боли
- Восход IR
- IR Плагины
- Планы

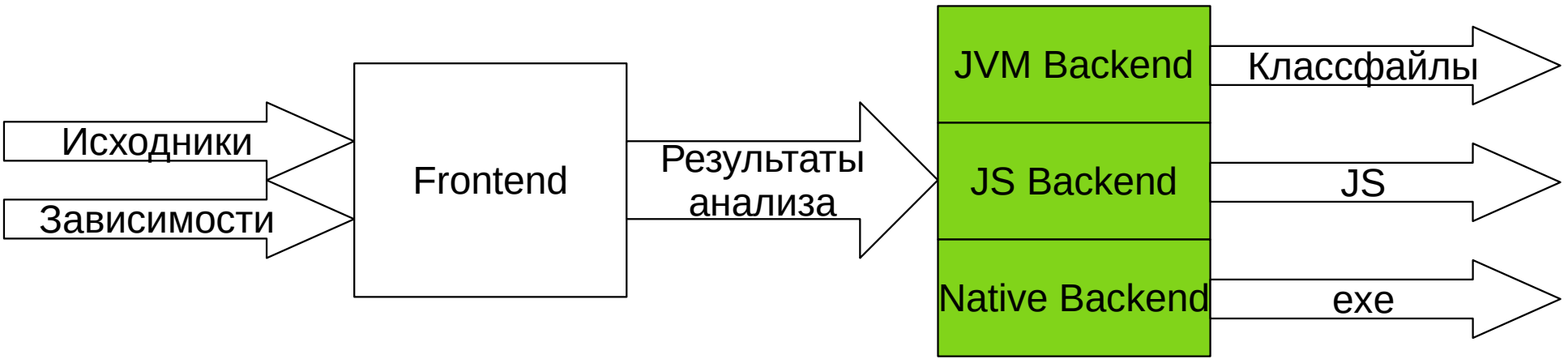
Структура Компилятора



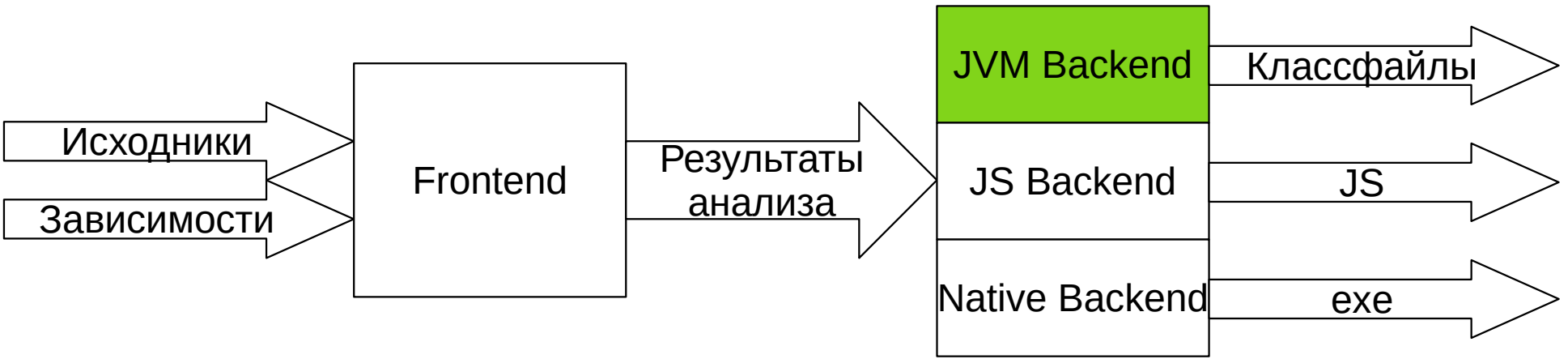
Структура Компилятора



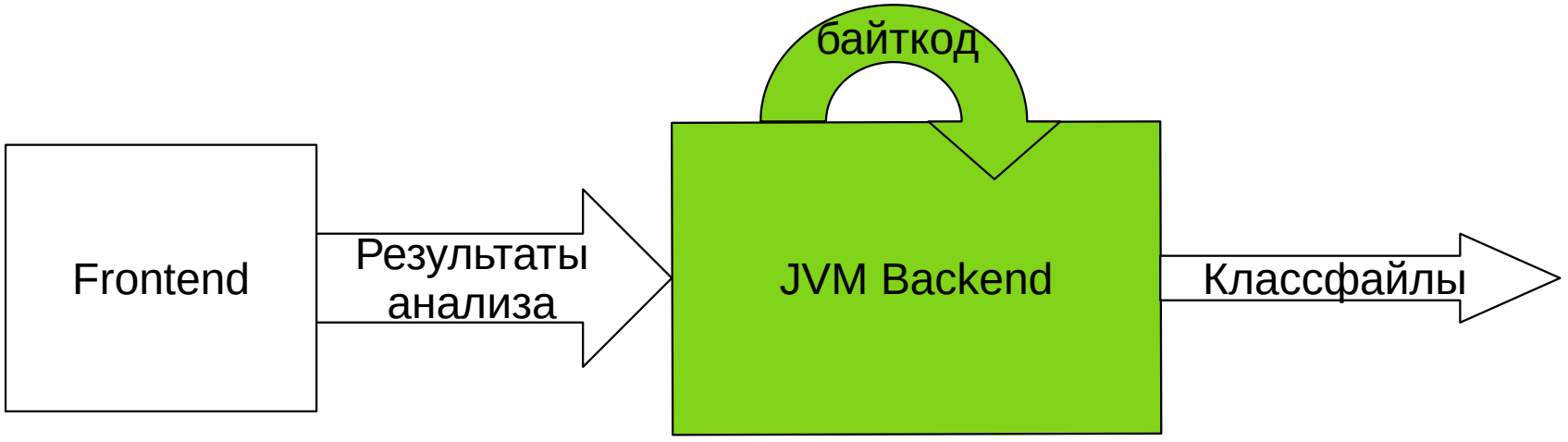
Структура Компилятора



Структура Компилятора



JVM BE


байткод
Frontend
Результаты анализа
JVM Backend
Классфайлы

MPP

- Три бекенда

MPP

- Три бекенда
  - Разные промежуточные представления

MPP

- Промежуточное представление
  - JVM (старый бекенд) – байткод

MPP

- Промежуточное представление
  - JVM (старый бекенд) – байткод
  - JS (старый бекенд) – JS AST

MPP

- Промежуточное представление
  - JVM (старый бекенд) – байткод
  - JS (старый бекенд) – JS AST
  - Native – LLVM bitcode?

MPP

- Промежуточное представление
  - JVM (старый бекенд) – байткод
  - JS (старый бекенд) – JS AST
  - Native – IR -> LLVM bitcode

Корутины

```
suspend fun returnsInt() = 1

suspend fun test() {
    println(returnsInt())
}
```

Корутины

```
suspend fun returnsInt() = 1

suspend fun test() {
    println(returnsInt())
}
```

Корутины

```
suspend fun returnsInt() = 1

suspend fun test() {
    println(returnsInt())
}
```

```
fun returnsInt($continuation: Continuation<Unit>) = 1

fun test($continuation: Continuation<Unit>): Any? {
    if ($continuation is test$Continuation) {
        if ($continuation.label.sign_bit == 1) {
            $continuation.label.sign_bit = 0
        }
    } else {
        $continuation = test$Continuation($continuation)
    }
    var $result: Any? = $continuation.result
    while(true) {
        when ($continuation.label) {
            0 -> {
                $continuation.label = 1
                $result = returnsInt($continuation)
                if ($result === COROUTINE_SUSPENDED) return COROUTINE_SUSPENDED
            }
            1 -> {
                $continuation.label = 2
                $result = println($result)
                if ($result === COROUTINE_SUSPENDED) return COROUTINE_SUSPENDED
                return Unit
            }
            else -> error("unreachable")
        }
    }
}
```

MPP: Корутины

```
suspend fun returnsInt() = 1

suspend fun test() {
    println(returnsInt())
}
```

MPP: Корутины

```
suspend fun returnsInt() = 1

suspend fun test() {
    println(returnsInt())
}
```


```
test (Lkotlin/coroutines/Continuation;)Ljava/lang/Object;:
  @Lorg/jetbrains/annotations/Nullable;() // invisible
    // annotable parameter count: 1 (visible)
    // annotable parameter count: 1 (invisible)
    @Lorg/jetbrains/annotations/NotNull;() // invisible, parameter 0
   L0
   L1
   L2
    LINENUMBER 7 L2
    INVOKESTATIC kotlin/jvm/internal/InlineMarker.beforeInlineCall ()V
    ALOAD 0
    ICONST_0
    INVOKESTATIC kotlin/jvm/internal/InlineMarker.mark (I)V
    INVOKESTATIC TestKt.returnsInt (Lkotlin/coroutines/Continuation;)Ljava/lang/Object;
    ICONST_1
    INVOKESTATIC kotlin/jvm/internal/InlineMarker.mark (I)V
    INVOKESTATIC kotlin/jvm/internal/InlineMarker.afterInlineCall ()V
    CHECKCAST java/lang/Number
    INVOKEVIRTUAL java/lang/Number.intValue ()I
    ISTORE 1
    NOP
   L3
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    ILOAD 1
    INVOKEVIRTUAL java/io/PrintStream.println (I)V
   L4
    NOP
    GOTO L5
   L6
   L7
   L5
   L8
    LINENUMBER 8 L8
    GETSTATIC kotlin/Unit.INSTANCE : Lkotlin/Unit;
    ARETURN
   L9
    MAXSTACK = 2
    MAXLOCALS = 2
```

MPP: Корутины

```
suspend fun returnsInt() = 1
```

```
suspend fun test() {
    println(returnsInt())
}
```

```
function test(continuation) {
  returnsInt($this$);
  println($this$.$$coroutineResult$$);
}
```

MPP: Корутины

```
suspend fun returnsInt() = 1

suspend fun test() {
    println(returnsInt())
}
```

```
FUN name:test visibility:public modality:FINAL <> () returnType:kotlin.Unit [suspend]
  BLOCK_BODY
    CALL 'public final fun println (message: kotlin.Int): kotlin.Unit [inline]
          declared in kotlin.io.ConsoleKt' type=kotlin.Unit origin=null
      message: CALL 'public final fun returnsInt (): kotlin.Int [suspend]
                    declared in <root>.TestKt' type=kotlin.Int origin=null
```

MPP: Корутины

- Генерация стейт-машины
  - JVM (старый бекенд) – генерация стейт-машины из байткода
  - JS (старый бекенд) – генерация стейт-машины из JS AST
  - Native – генерация стейт-машины из IR

MPP: Корутины

- Оптимизации
  - JVM (старый бекенд) – на байткоде
  - JS (старый бекенд) – на JS AST
  - Native – на IR

MPP

- Три бекенда

MPP

- Три бекенда
  - Утраивание работы

Вопросы?

?

IR

- Промежуточное представление
  - JVM (старый бекенд) – байткод
  - JS (старый бекенд) – JS AST
  - Native – IR -> LLVM bitcode

IR

- Промежуточное представление
  - JVM (старый бекенд) – байткод
  - JS (старый бекенд) – JS AST
  - Native – IR -> LLVM bitcode

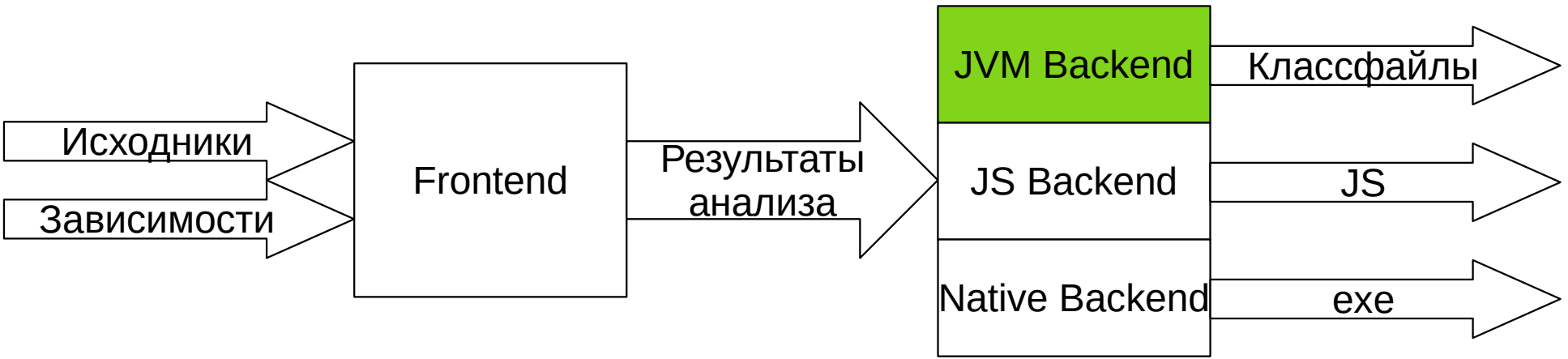
IR

- Промежуточное представление
  - JVM – **IR** & байткод -> байткод
  - JS – **IR** -> JS AST
  - Native – **IR** -> LLVM bitcode

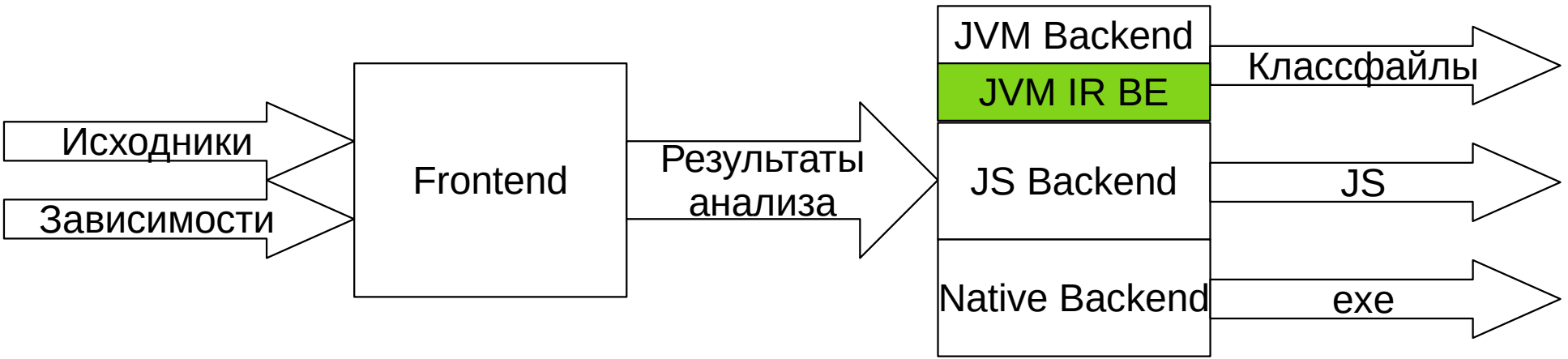
IR

- Промежуточное представление
  - JVM – **IR** & байткод -> байткод
  - JS – **IR** -> JS AST
  - Native – **IR** -> LLVM bitcode

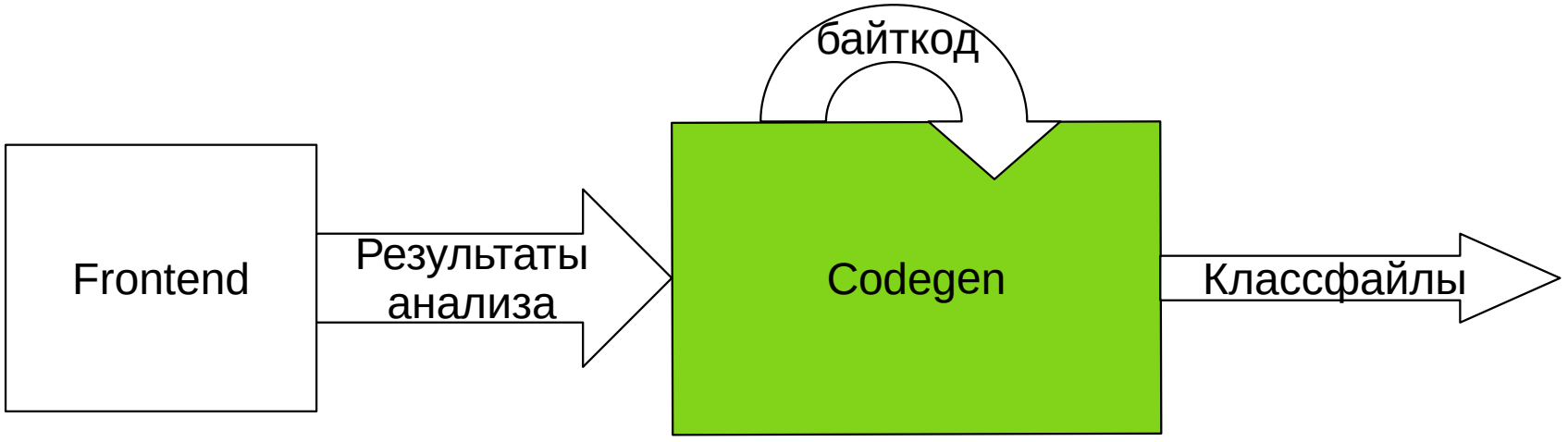
JVM IR


Исходники
Зависимости
Frontend
Результаты анализа
JVM Backend
JS Backend
Native Backend
Классфайлы
JS
exe

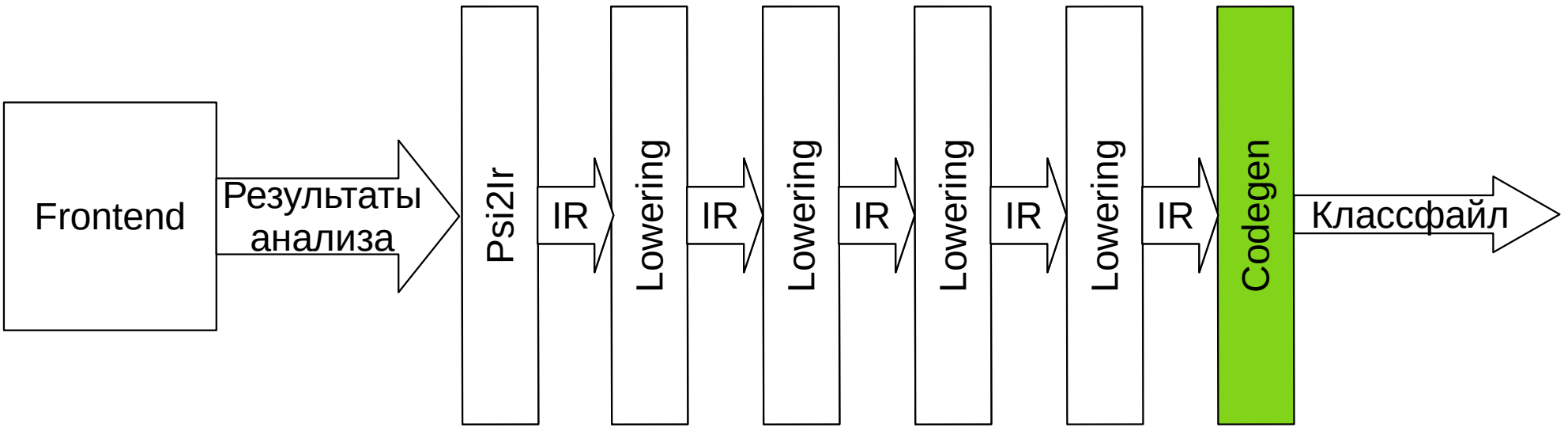
JVM IR


Исходники
Зависимости
Frontend
Результаты анализа
JVM Backend
JVM IR BE
JS Backend
Native Backend
Классфайлы
JS
exe

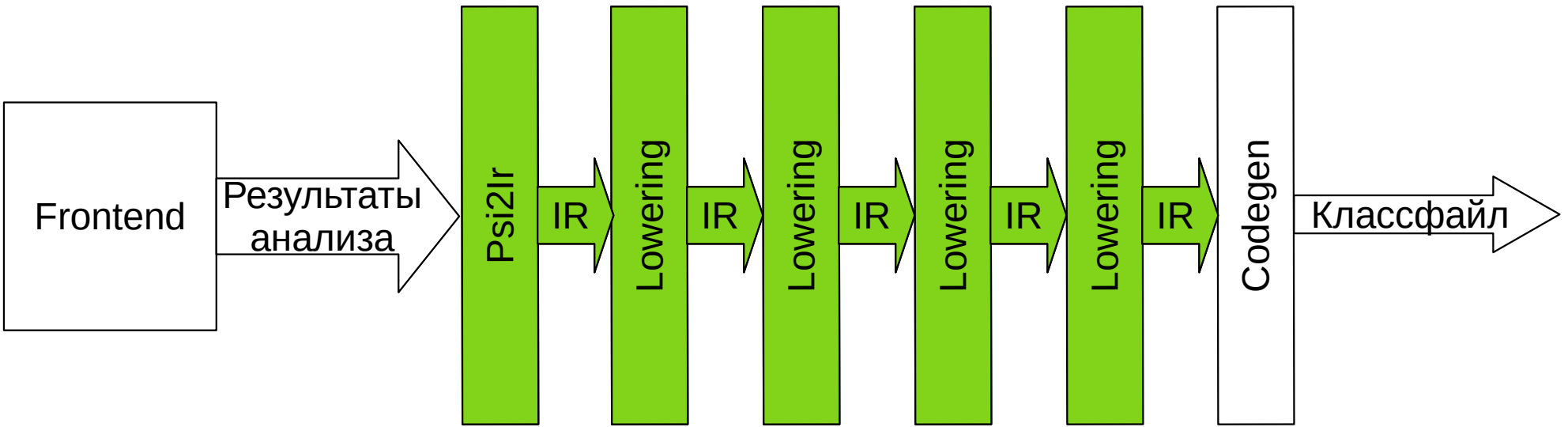
JVM BE



JVM IR BE


Frontend
Результаты анализа
Psi2Ir
IR
Lowering
IR
Lowering
IR
Lowering
IR
Lowering
IR
Codegen
Классфайл

JVM IR BE


Frontend
Результаты анализа
Psi2Ir
IR
Lowering
IR
Lowering
IR
Lowering
IR
Lowering
IR
Codegen
Классфайл

JVM_IR: Корутины

```
suspend fun returnsInt() = 1

suspend fun test() {
    println(returnsInt())
}
```

JVM_IR: Корутины

```
suspend fun returnsInt() = 1

suspend fun test() {
    println(returnsInt())
}
```

```
FUN name:test visibility:public modality:FINAL <> ()
returnType:kotlin.Unit [suspend]
  BLOCK_BODY
    CALL 'public final fun println (message: kotlin.Int):
       kotlin.Unit [inline]
          declared in kotlin.io.ConsoleKt' type=kotlin.Unit
               origin=null
      message: CALL 'public final fun returnsInt (): kotlin.Int
                                                  [suspend]
                    declared in <root>.TestKt' type=kotlin.Int
                             origin=null
```

JVM_IR: Корутины

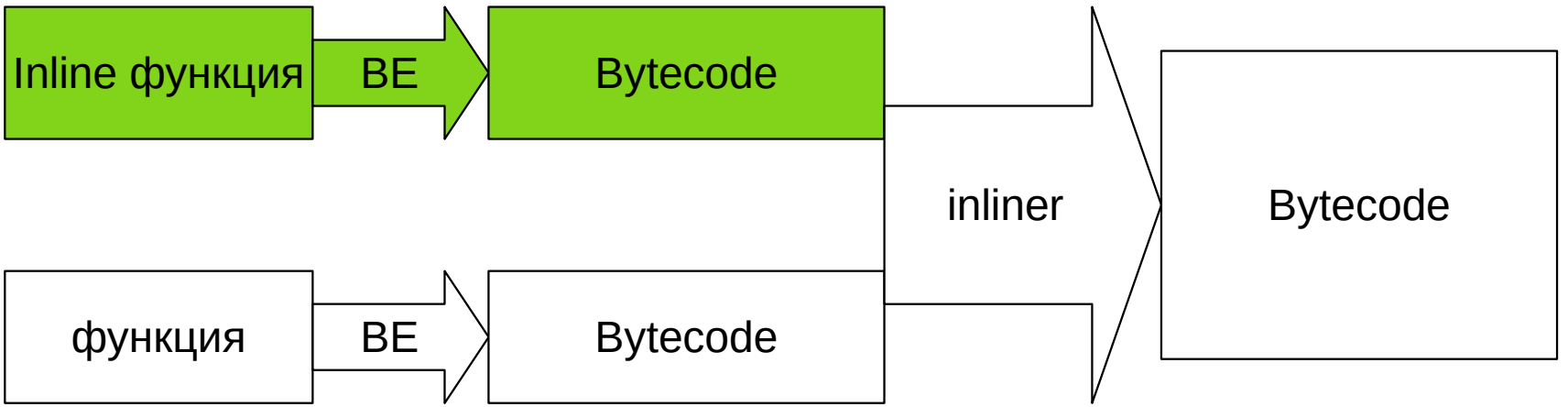
```
suspend fun returnsInt() = 1

suspend fun test() {
    println(returnsInt())
}
```

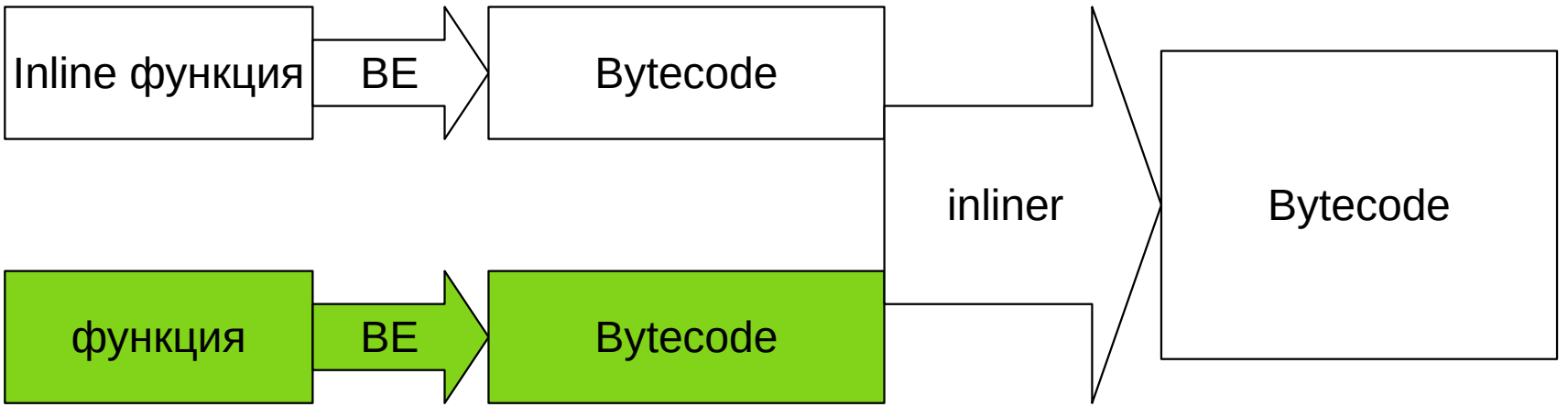
```
FUN name:test visibility:public modality:FINAL <> ()
returnType:kotlin.Unit [suspend]
  BLOCK_BODY
    CALL 'public final fun println (message: kotlin.Int):
       kotlin.Unit [inline]
          declared in kotlin.io.ConsoleKt' type=kotlin.Unit
               origin=null
      message: CALL 'public final fun returnsInt (): kotlin.Int
                                                  [suspend]
                  declared in <root>.TestKt' type=kotlin.Int
                           origin=null
```

```
test (Lkotlin/coroutines/Continuation;)Ljava/lang/Object;:
  @Lorg/jetbrains/annotations/Nullable;() // invisible
    // annotable parameter count: 1 (visible)
    // annotable parameter count: 1 (invisible)
    @Lorg/jetbrains/annotations/NotNull;() // invisible, parameter 0
   L0
   L1
   L2
    LINENUMBER 7 L2
    INVOKESTATIC kotlin/jvm/internal/InlineMarker.beforeInlineCall ()V
    ALOAD 0
    ICONST_0
    INVOKESTATIC kotlin/jvm/internal/InlineMarker.mark (I)V
    INVOKESTATIC TestKt.returnsInt (Lkotlin/coroutines/Continuation;)Ljava/lang/Object;
    ICONST_1
    INVOKESTATIC kotlin/jvm/internal/InlineMarker.mark (I)V
    INVOKESTATIC kotlin/jvm/internal/InlineMarker.afterInlineCall ()V
    CHECKCAST java/lang/Number
    INVOKEVIRTUAL java/lang/Number.intValue ()I
    ISTORE 1
    NOP
   L3
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    ILOAD 1
    INVOKEVIRTUAL java/io/PrintStream.println (I)V
   L4
    NOP
    GOTO L5
   L6
   L7
   L5
   L8
    LINENUMBER 8 L8
    GETSTATIC kotlin/Unit.INSTANCE : Lkotlin/Unit;
    ARETURN
   L9
    MAXSTACK = 2
    MAXLOCALS = 2
```

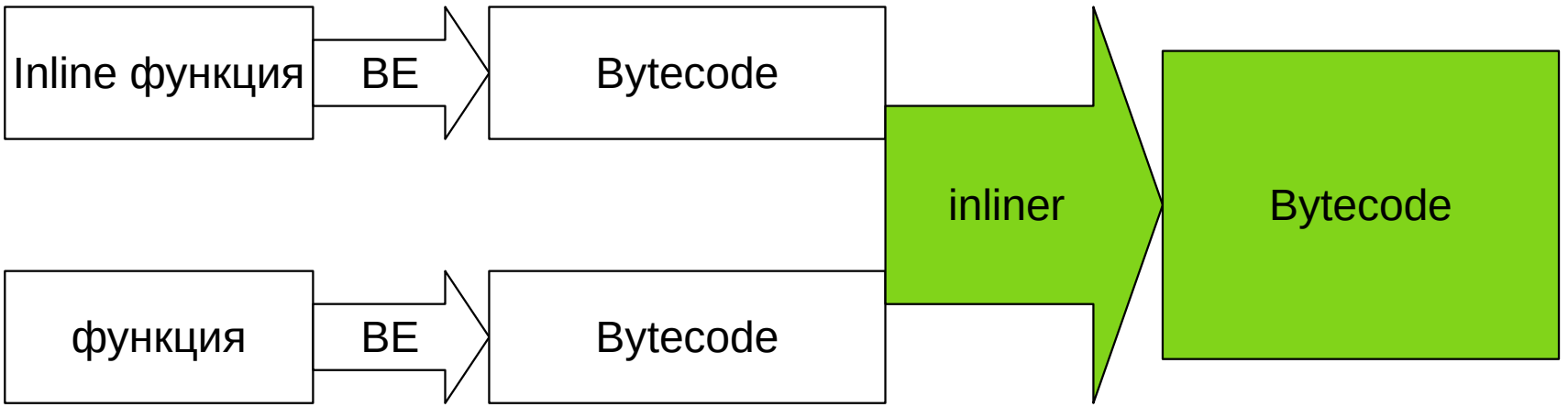
JVM_IR: Inline



JVM_IR: Inline



JVM_IR: Inline



JVM_IR: Результаты

- Alpha в 1.4

JVM_IR: Результаты

- Alpha в 1.4
  - + Compose

JVM_IR: Результаты

- Alpha в 1.4
  - + Compose
- Релиз в 1.5

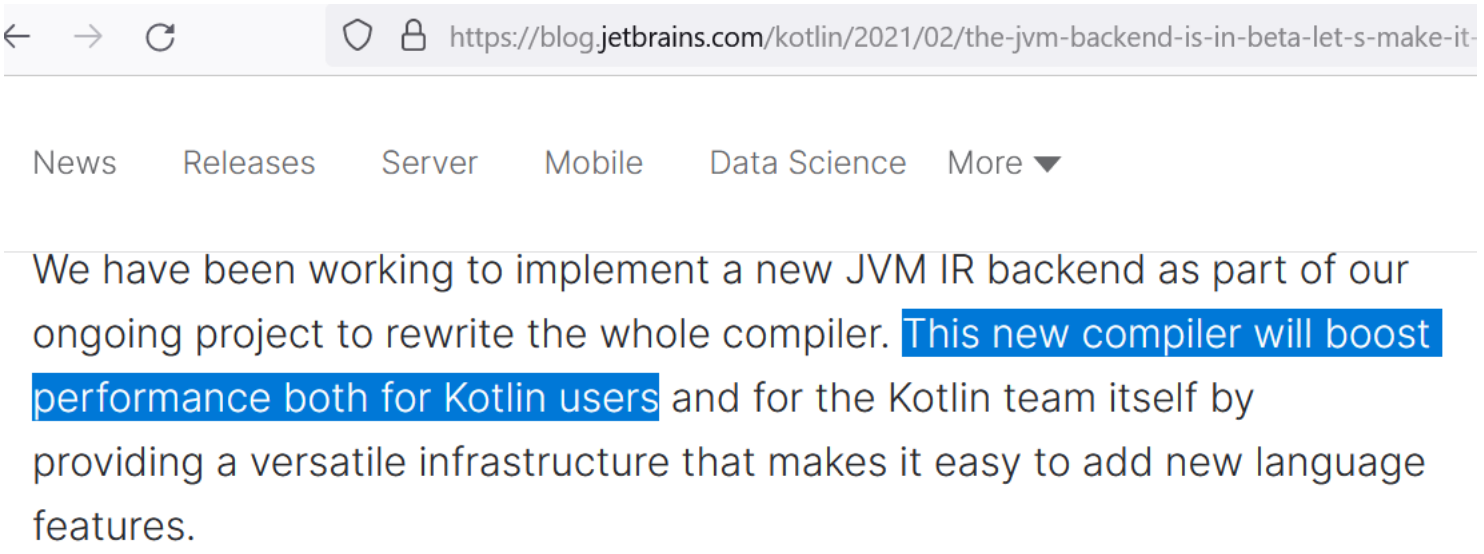
JVM_IR: Результаты

- Alpha в 1.4
  - + Compose
- Релиз в 1.5
  - ~ 250 пофикшенных багов!

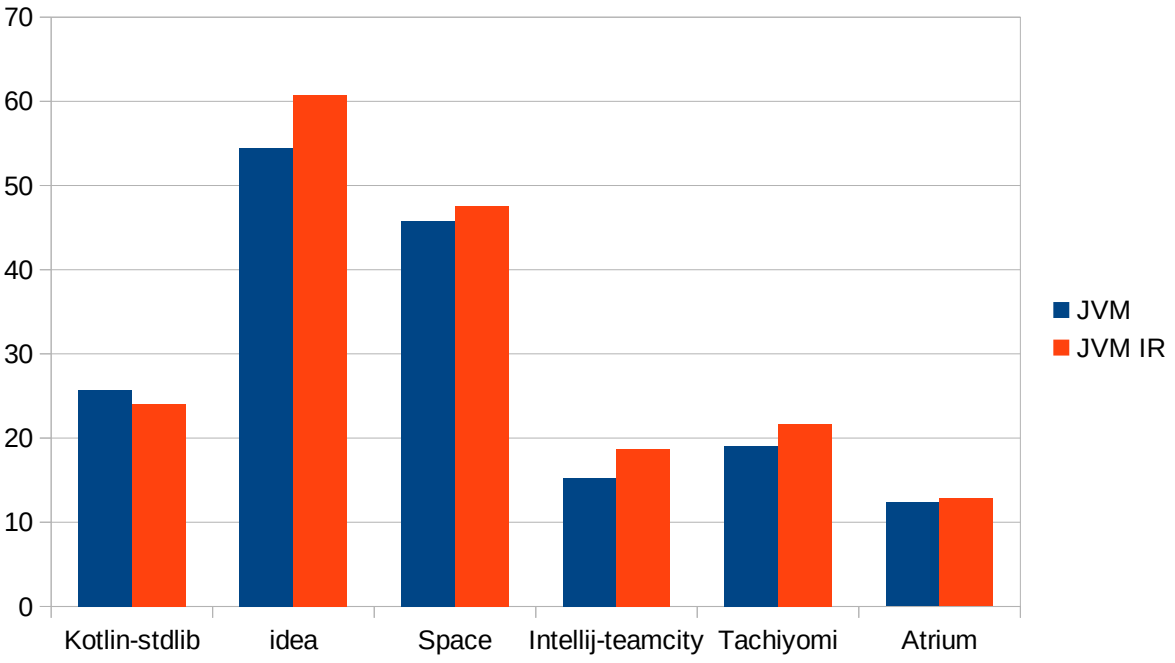
JVM_IR: Результаты

- Alpha в 1.4
  - + Compose
- Релиз в 1.5
  - ~ 250 пофикшенных багов!
  - Сравнительно мало регрессий

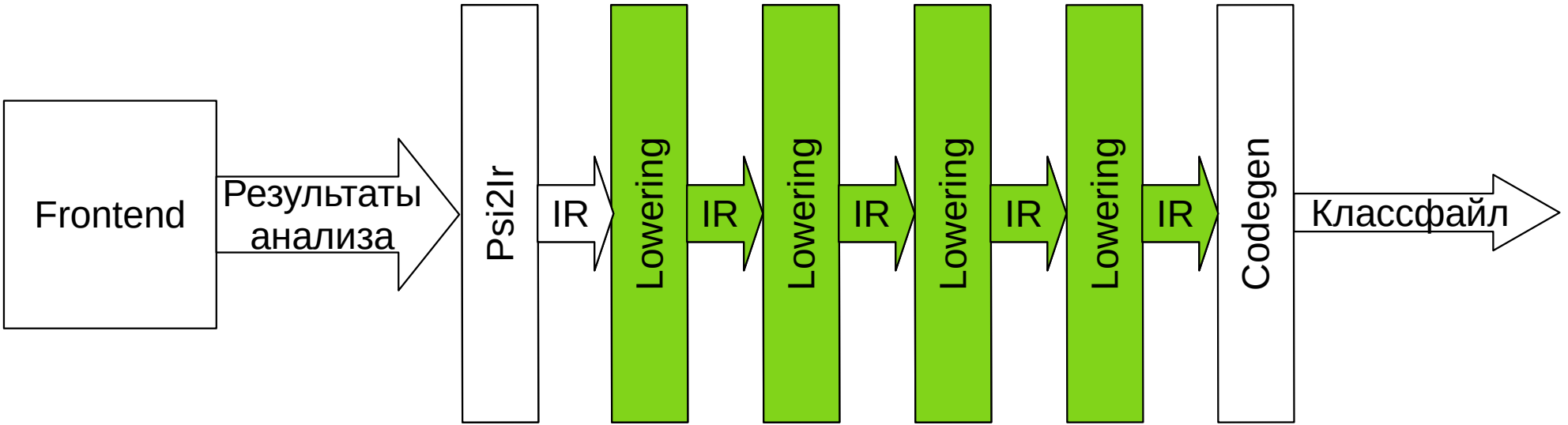
JVM_IR: Результаты



JVM_IR: Результаты



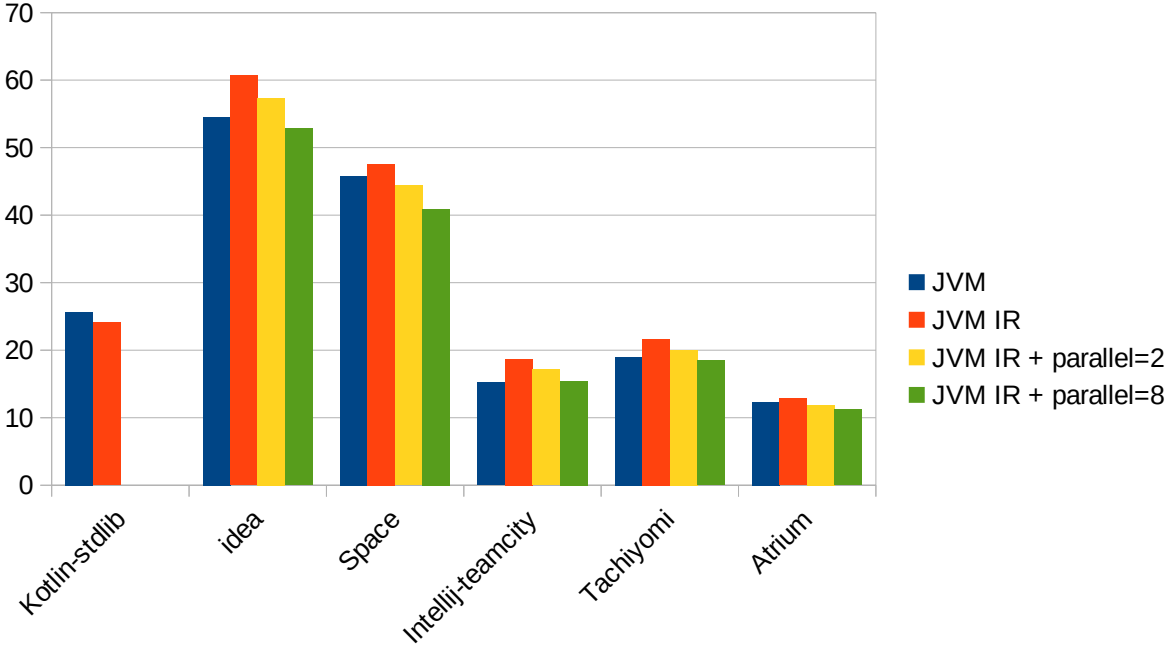
JVM_IR: Результаты



JVM_IR: Результаты

- IR проще параллелить

JVM_IR: Результаты



JVM_IR: Результаты

- IR проще параллелить
  - Флаг `-Xbackend-threads=<N>`, где N=0 по числу ядер

JVM_IR: Результаты

- IR проще параллелить
  - Флаг `-Xbackend-threads=<N>`, где N=0 по числу ядер
    - С 1.6.0

Вопросы?

?

JVM_IR: Плагины

- Для JVM_IR проще писать плагины

JVM_IR: Плагины

- Для JVM_IR проще писать плагины
  - И их уже начали писать

JVM_IR: Плагины

- kotlin-power-assert

https://github.com/bnorm/kotlin-power-assert

JVM_IR: Плагины

A custom assertion message can be provided:

```
val hello = "Hello"
assert(hello.length == "World".substring(1, 4).length) { "Incorrect length" }
```

But this just replaces the message:

```
java.lang.AssertionError: Incorrect length
        at <stacktrace>
```

With `kotlin-power-assert` included, the error message for the previous example will be transformed:

```
java.lang.AssertionError: Incorrect length
assert(hello.length == "World".substring(1, 4).length)
       |     |      |         |              |
       |     |      |         |              3
       |     |      |         orl
       |     |      false
       |     5
       Hello
        at <stacktrace>
```

https://github.com/bnorm/kotlin-power-assert

JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose

https://android.googlesource.com/platform/frameworks/support/+/refs/heads/androidx-main/compose

JVM_IR: Плагины

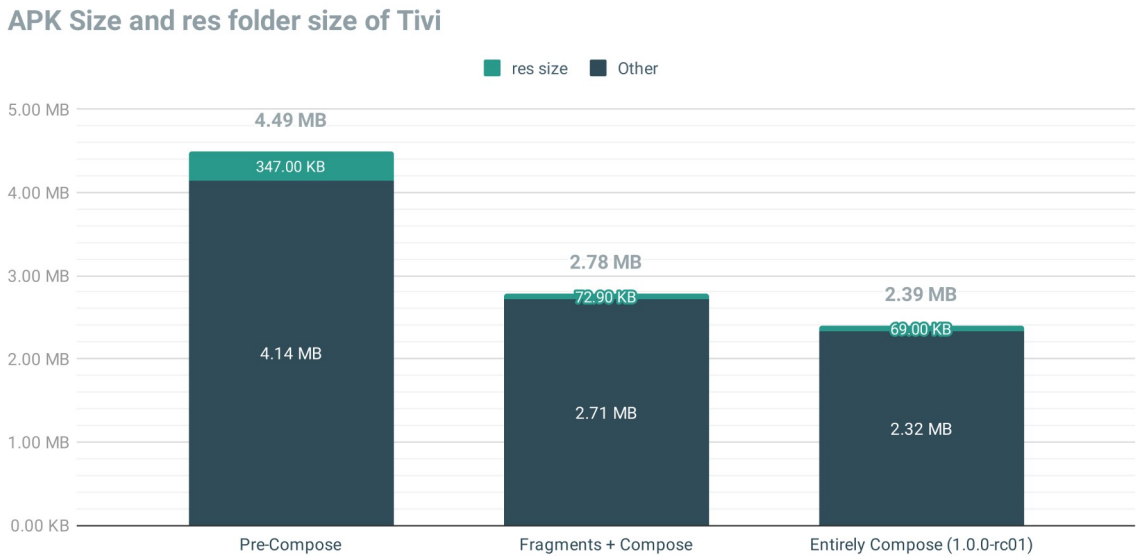
- kotlin-power-assert
- JetPack Compose

```
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
                    text = "Jetpack Compose",
                    style = MaterialTheme.typography.v2,
                )
            }
        }
    }
}
```

https://android.googlesource.com/platform/frameworks/support/+/refs/heads/androidx-main/compose

JVM_IR: Плагины

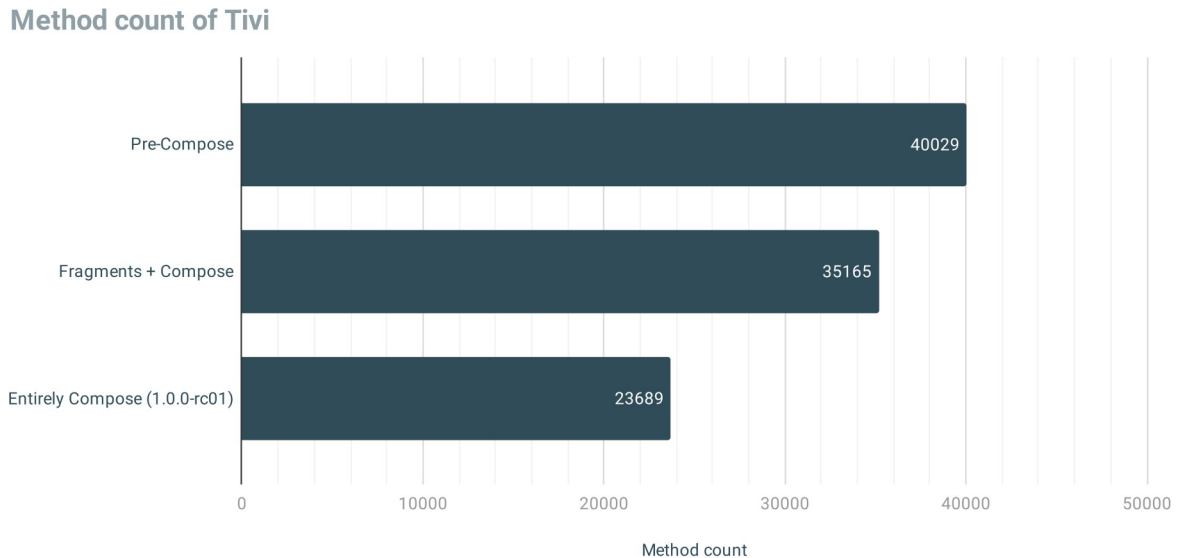
- kotlin-power-assert
- JetPack Compose



https://medium.com/androiddevelopers/jetpack-compose-before-and-after-8b43ba0b7d4f

JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose



https://medium.com/androiddevelopers/jetpack-compose-before-and-after-8b43ba0b7d4f

JVM_IR: Плагины

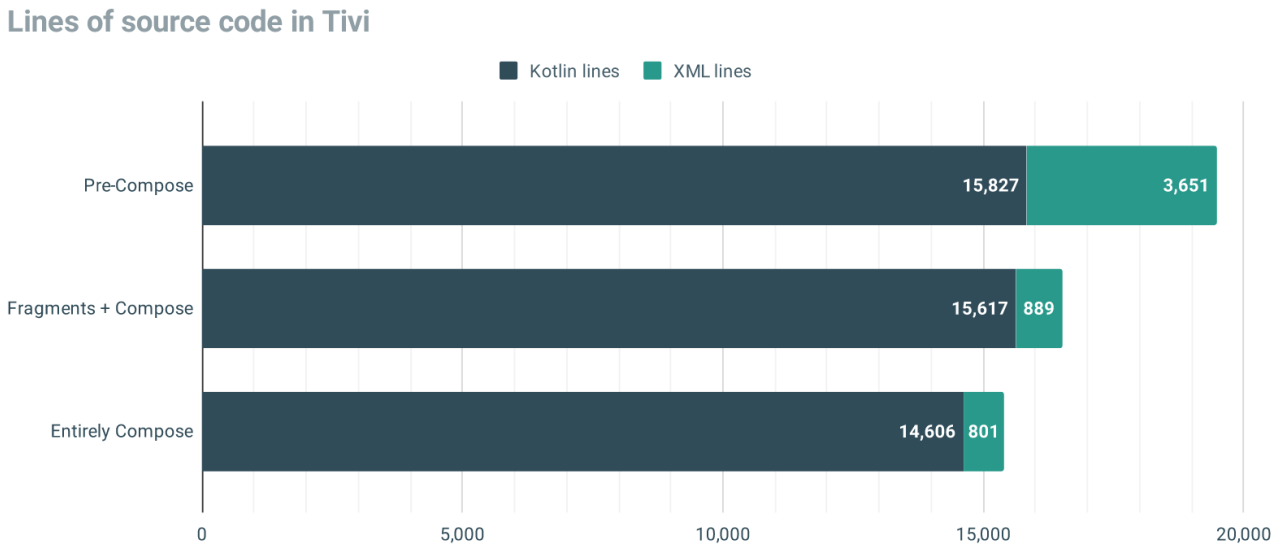
- kotlin-power-assert
- JetPack Compose

> We see a 41% reduction in APK size, and 17% reduction in method count when using Compose

https://medium.com/androiddevelopers/jetpack-compose-before-and-after-8b43ba0b7d4f

JVM_IR: Плагины

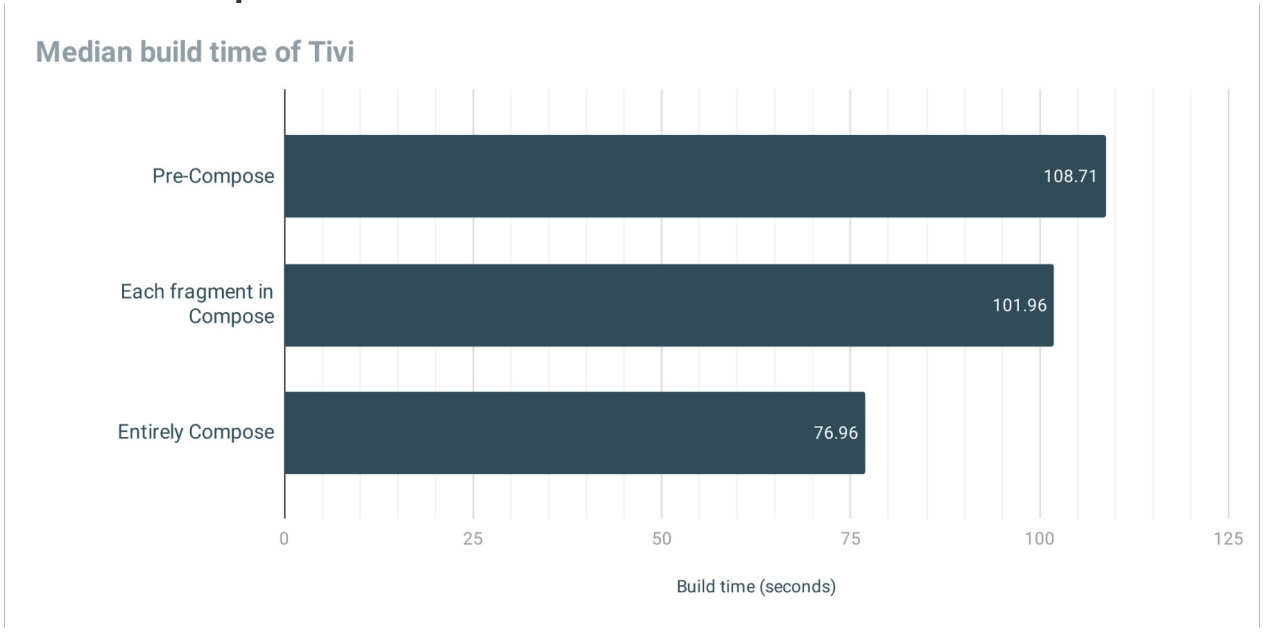
- kotlin-power-assert
- JetPack Compose



https://medium.com/androiddevelopers/jetpack-compose-before-and-after-8b43ba0b7d4f

JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose



https://medium.com/androiddevelopers/jetpack-compose-before-and-after-8b43ba0b7d4f

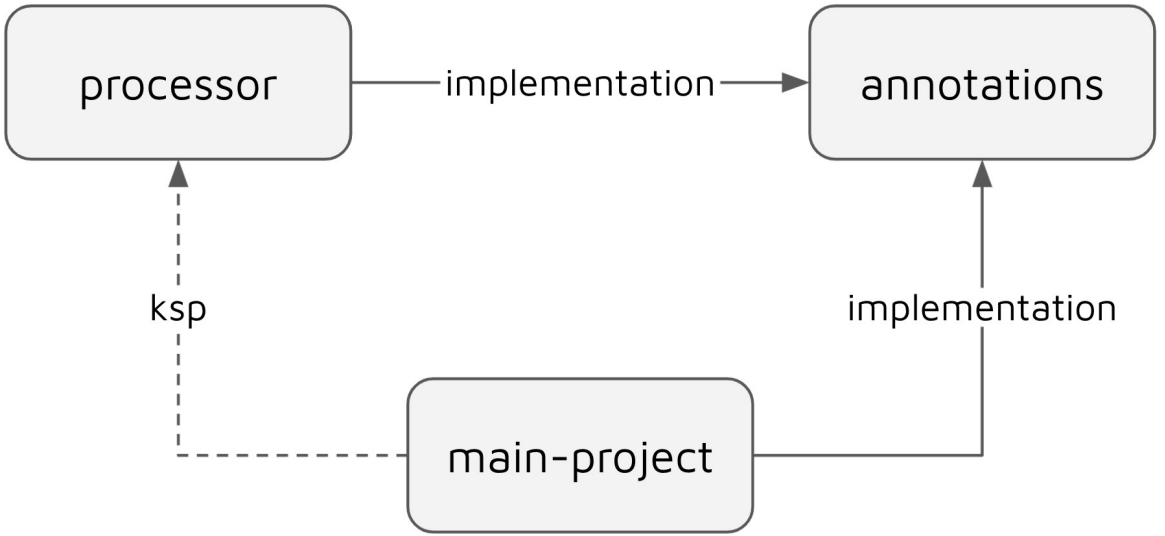
JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose
- KSP

https://github.com/google/ksp

JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose
- KSP



https://github.com/google/ksp

JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose
- KSP

```
plugins {
  ...
  id("com.google.devtools.ksp")
}

kapt(project(":test-processor"))
ksp(project(":test-processor"))
```

This is the goal of KSP: most Android app developers don't need to worry about its internals; other than this one line change, a library that supports KSP looks just like a normal annotation processor, only it's up to 2x faster. That said, using KAPT and KSP in the same module will likely slow down your build initially, so during this alpha period, it is best to use KSP and KAPT in separate modules.

https://github.com/google/ksp

JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose
- KSP
- reflekt

https://github.com/JetBrains-Research/reflekt

JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose
- KSP
- reflekt
  - Для поддержки GraalVM (AOT компиляция)

https://github.com/JetBrains-Research/reflekt

JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose
- KSP
- reflekt
  - Для поддержки GraalVM (AOT компиляция)
  - Быстрый старт приложений

https://github.com/JetBrains-Research/reflekt

JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose
- KSP
- reflekt
- arrow-meta

https://github.com/arrow-kt/arrow-meta

JVM_IR: Плагины

The Hello World plugin auto implements the `helloWorld` function by rewriting the Kotlin AST before the compiler proceeds.

```
val Meta.helloWorld: CliPlugin get() =
  "Hello World" {
    meta(
      namedFunction(this, { name == "helloWorld" }) { c ->  // <— namedFunction(...) {...}
        Transform.replace(
          replacing = c,
          newDeclaration = """|fun helloWorld(): Unit =
                              |  println("Hello ΛRROW Meta!")
                              |""".function.syntheticScope
        )
      }
    )
  }
```

For any user code whose function name is `helloWorld`, our compiler plugin will replace the matching function for a function that returns Unit and prints our message.

```
-fun helloWorld(): Unit = TODO()
+fun helloWorld(): Unit =
+  println("Hello ΛRROW Meta!")
```

https://github.com/arrow-kt/arrow-meta

JVM_IR: Плагины

- kotlin-power-assert
- JetPack Compose
- KSP
- reflekt
- arrow-meta
- Место для вашего плагина

JVM_IR: Плагины

Writing Your Second Kotlin Compiler Plugin, Part 1 — Project Setup

Brian Norman Nov 21, 2020 · 4 min read

Photo by Ben Stern on Unsplash

https://bnorm.medium.com/writing-your-second-kotlin-compiler-plugin-part-1-project-setup-7b05c7d93f6c

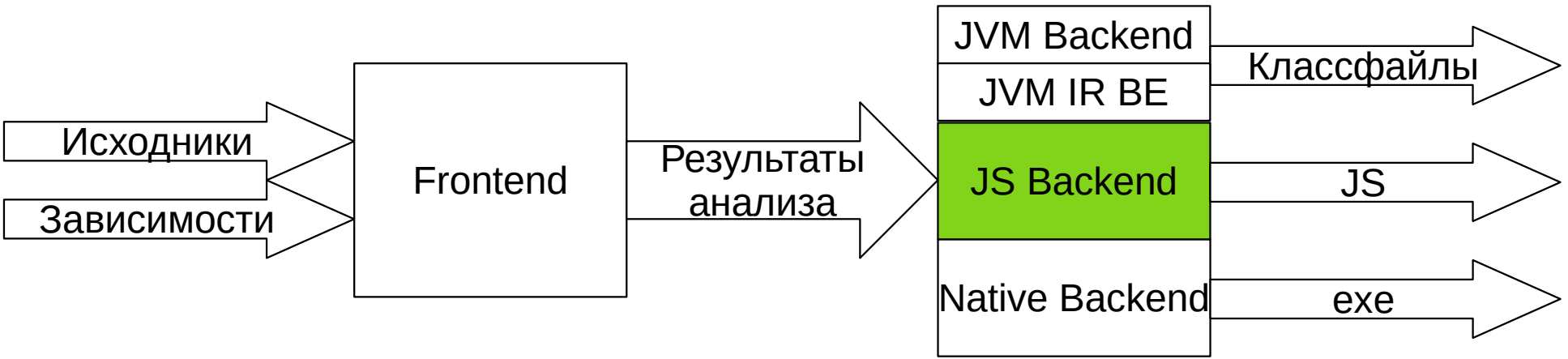
JS_IR

Kotlin/JS

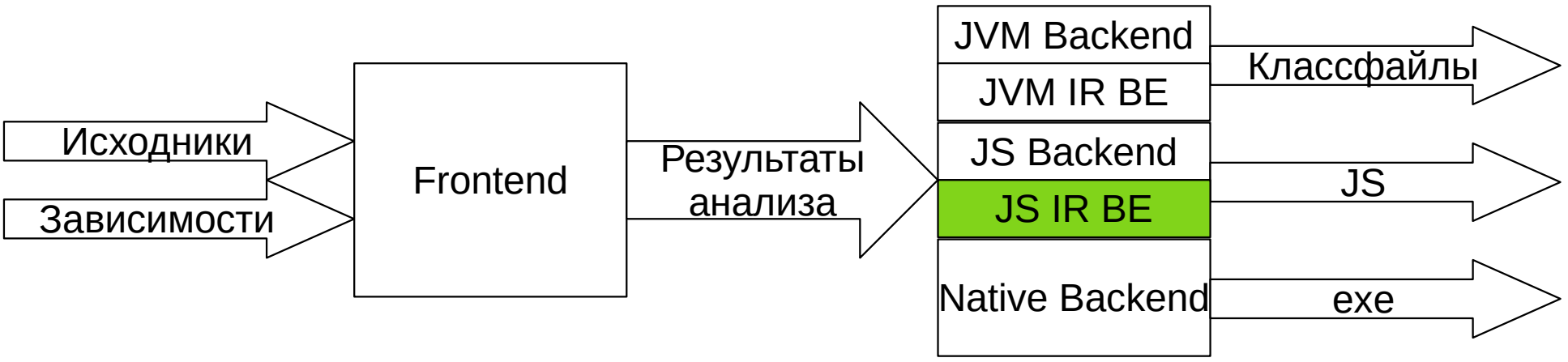
Kotlin 1.5.30 provides the following improvements for Kotlin/JS:

- JS IR compiler backend reaches Beta. To simplify migration to the new backend, you can use the migration guide and the new Kotlin/JS Inspection Pack IDE plugin, which guides you through the process of making the necessary changes directly in IntelliJ IDEA.
- A better debugging experience for applications with the Kotlin/JS IR backend, thanks to JavaScript source map generation. Now you can benefit from support for breakpoints, stepping, and readable stack traces with proper source references in any JavaScript debugger. Learn more about debugging Kotlin/JS applications.

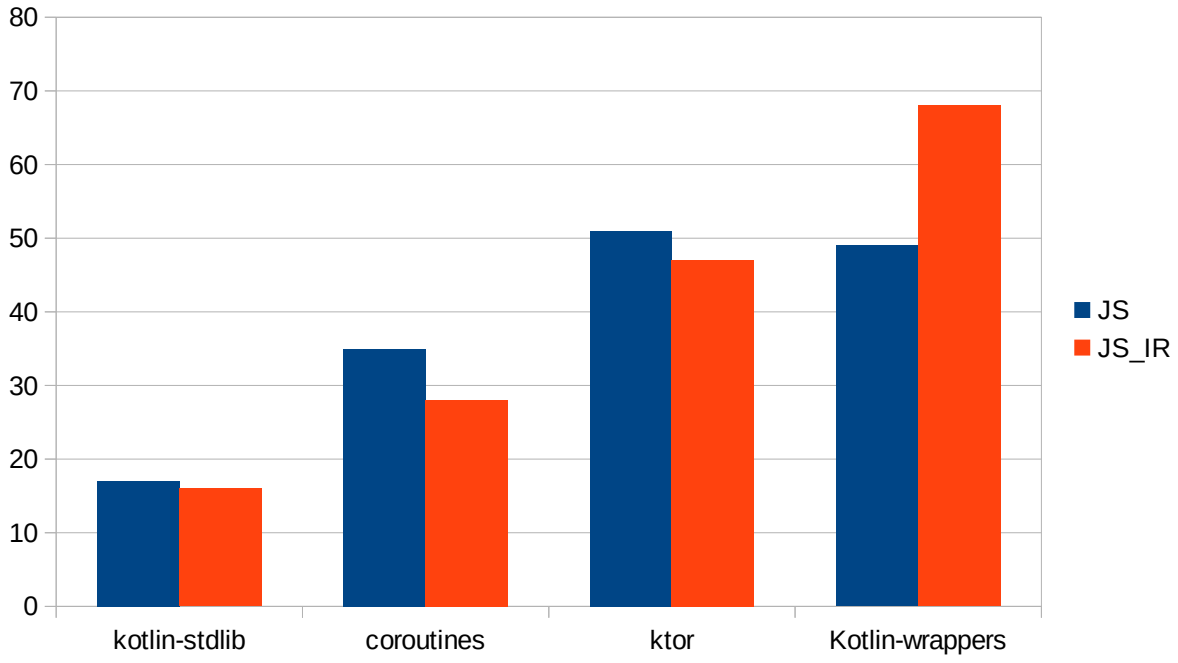
JS IR


Исходники
Зависимости
Frontend
Результаты анализа
JVM Backend
JVM IR BE
JS Backend
Native Backend
Классфайлы
JS
exe

JS IR


Исходники
Зависимости
Frontend
Результаты анализа
JVM Backend
JVM IR BE
JS Backend
JS IR BE
Native Backend
Классфайлы
JS
exe

JS_IR: Performance



Вопросы?

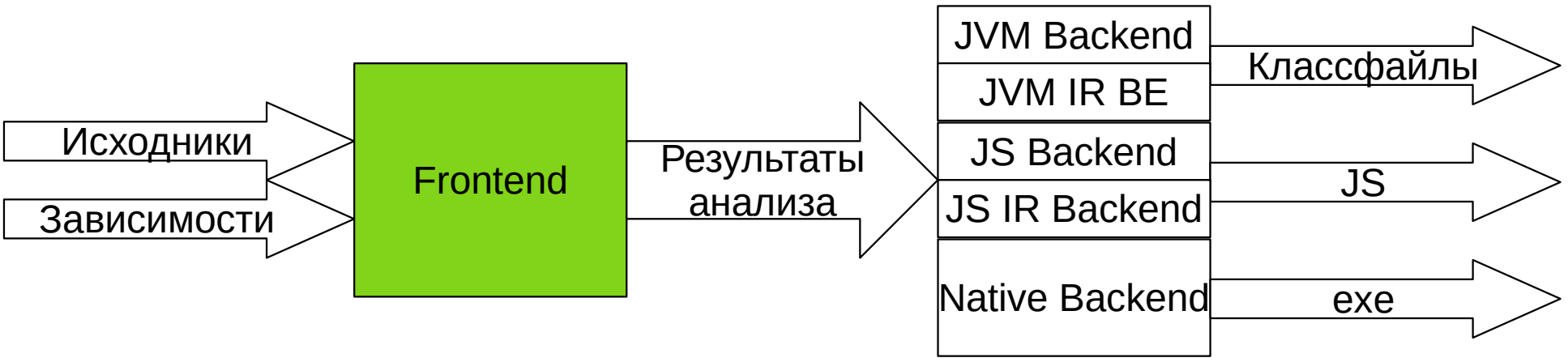
?

Планы

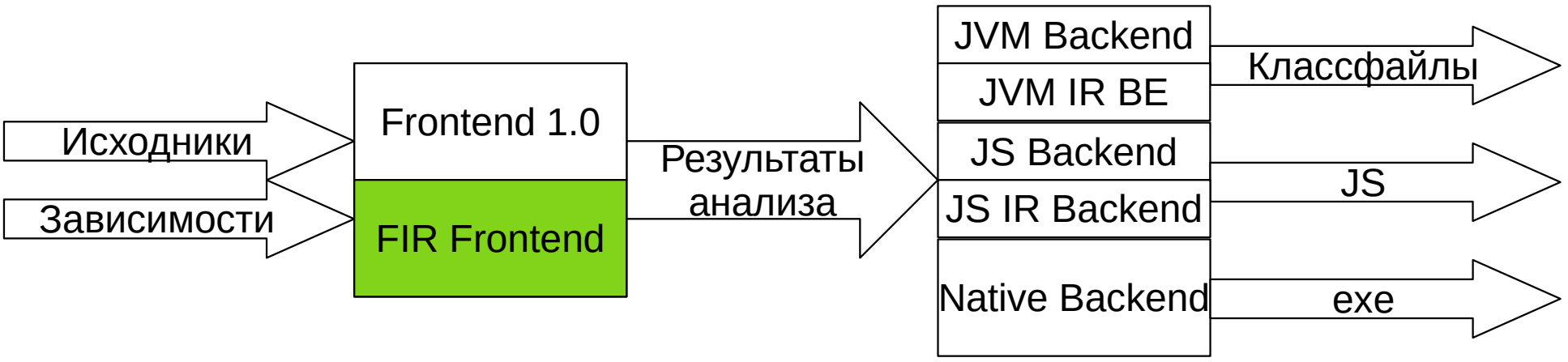
Планы

- FIR

FIR

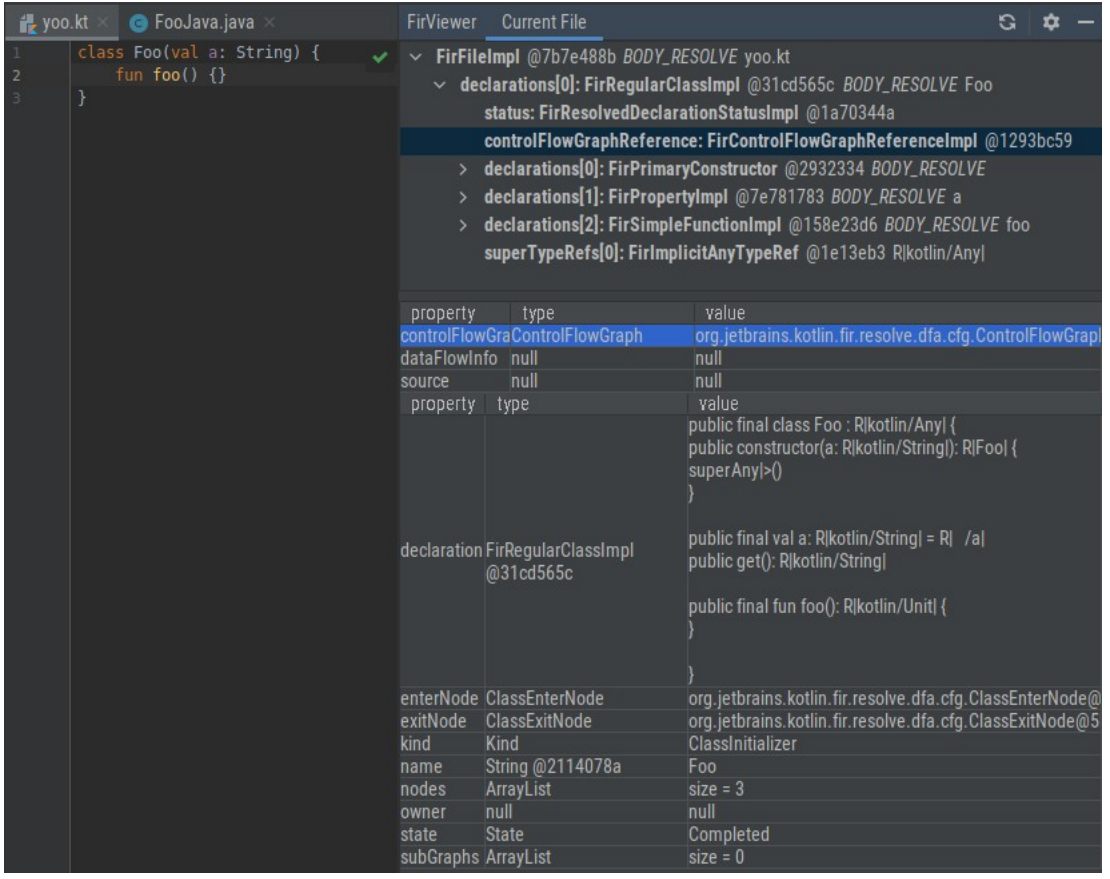

Исходники
Зависимости
Frontend
Результаты анализа
JVM Backend
JVM IR BE
JS Backend
JS IR Backend
Native Backend
Классфайлы
JS
exe

FIR


Исходники
Зависимости
Frontend 1.0
FIR Frontend
Результаты анализа
JVM Backend
JVM IR BE
JS Backend
JS IR Backend
Native Backend
Классфайлы
JS
exe

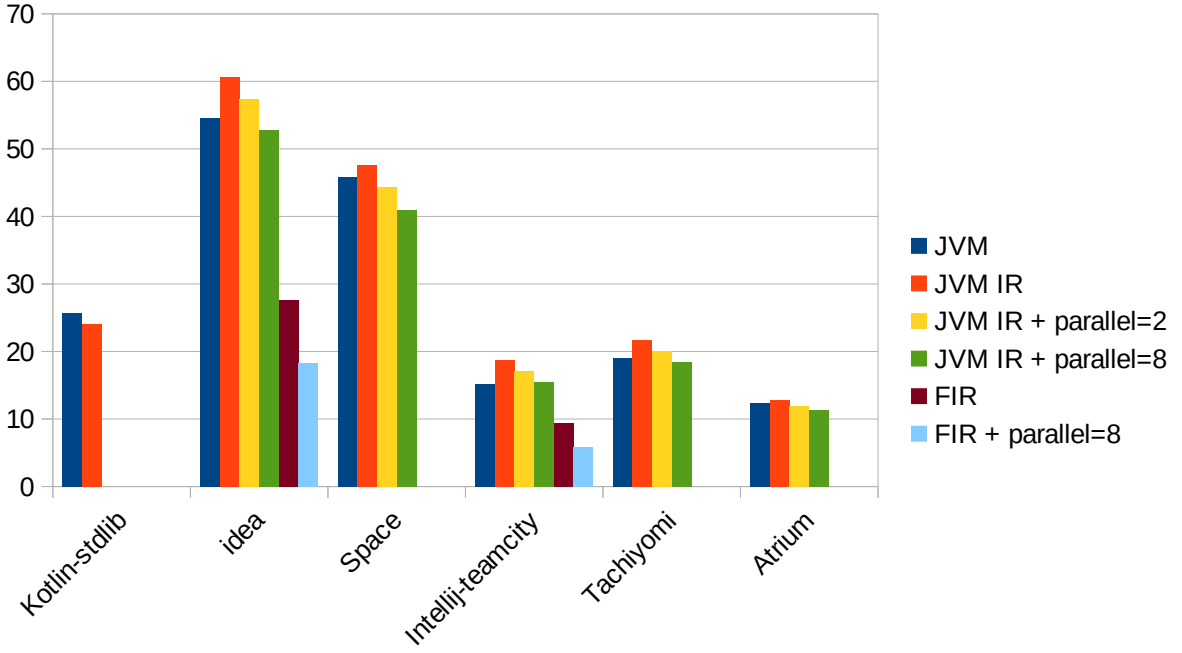
Планы

- FIR


yoo.kt
FooJava.java
class Foo(val a: String) {
fun foo() {}
}
FirViewer
Current File
FirFileImpl @7b7e488b BODY_RESOLVE yoo.kt
declarations[0]: FirRegularClassImpl @31cd565c BODY_RESOLVE Foo
status: FirResolvedDeclarationStatusImpl @1a70344a
controlFlowGraphReference: FirControlFlowGraphReferenceImpl @1293bc59
declarations[0]: FirPrimaryConstructor @2932334 BODY_RESOLVE
declarations[1]: FirPropertyImpl @7e781783 BODY_RESOLVE a
declarations[2]: FirSimpleFunctionImpl @158e23d6 BODY_RESOLVE foo
superTypeRefs[0]: FirImplicitAnyTypeRef @1e13eb3 R|kotlin/Any|
property
type
value
controlFlowGra
ControlFlowGraph
org.jetbrains.kotlin.fir.resolve.dfa.cfg.ControlFlowGrapl
dataFlowInfo
null
null
source
null
null
property
type
value
declaration
FirRegularClassImpl @31cd565c
public final class Foo : R|kotlin/Any| {
public constructor(a: R|kotlin/String|): R|Foo| {
superAny|>()
}
public final val a: R|kotlin/String| = R| /a|
public get(): R|kotlin/String|
public final fun foo(): R|kotlin/Unit| {
}
}
enterNode
ClassEnterNode
org.jetbrains.kotlin.fir.resolve.dfa.cfg.ClassEnterNode@
exitNode
ClassExitNode
org.jetbrains.kotlin.fir.resolve.dfa.cfg.ClassExitNode@5
kind
Kind
ClassInitializer
name
String @2114078a
Foo
nodes
ArrayList
size = 3
owner
null
null
state
State
Completed
subGraphs
ArrayList
size = 0

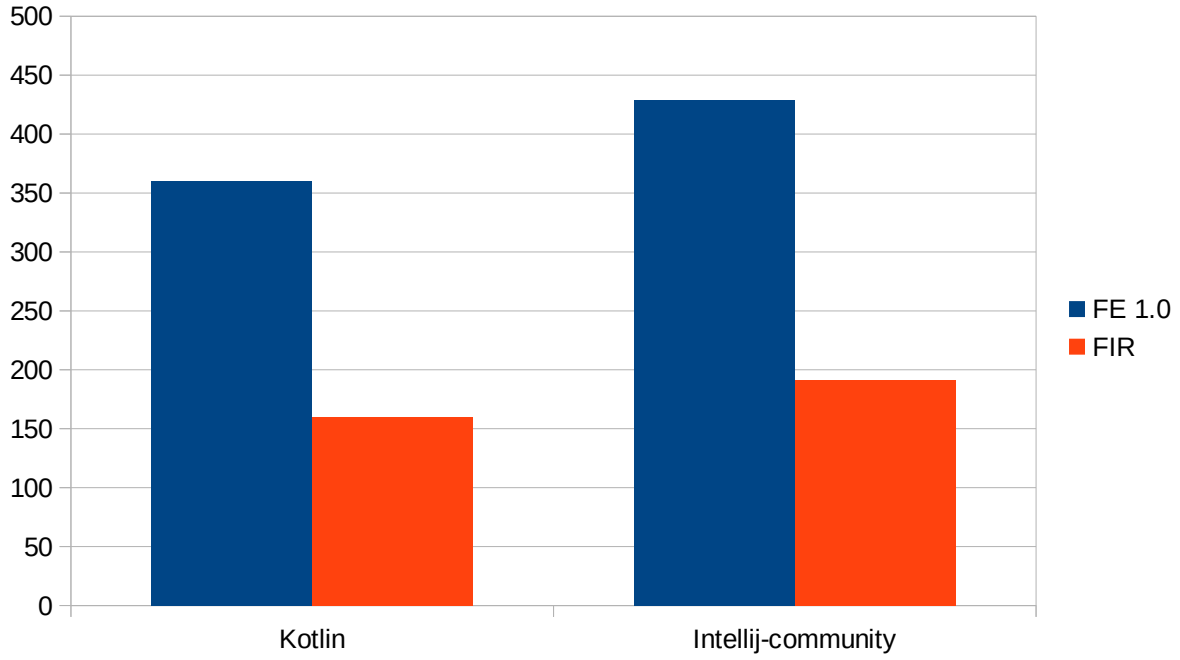
Планы

- FIR


70
60
50
40
30
20
10
0
Kotlin-stdlib
idea
Space
Intellij-teamcity
Tachiyomi
Atrium
JVM
JVM IR
JVM IR + parallel=2
JVM IR + parallel=8
FIR
FIR + parallel=8

Планы

- FIR


500
450
400
350
300
250
200
150
100
50
0
Kotlin
Intellij-community
FE 1.0
FIR

Планы

- FIR
  - Флаг `-Xuse-fir`

Планы

- FIR
  - Как и IR, должен упростить написание плагинов
    - Compiler Plugin API

Планы

- FIR
  - Как и IR, должен упростить написание плагинов
  - Быстрый анализ кода

Планы

- FIR
- IR interpreter

Планы

- FIR
- IR interpreter

★ Introduce constexpr/const modifier/annotation for functions that can be computed in compile-time 106

Introduce C++ style `constexpr` modifier (in Kotlin the better name might be `const` or it might be some `@CompileTime` annotation – to be designed).

In short, functions marked with `constexpr` are verified by compiler to satisfy certain strict rules and they can be used as a part of constant expressions and assigned to `const val` properties with their values evaluated in compile-time. The goals of this change are:

- Documentation, discoverability, and language learning-curve reduction
- Language consistency for novices (especially coming from non-Java world)
- Language extensibility

Планы

- FIR
- IR interpreter
- klib (IR serialization)

Планы

- FIR
- IR interpreter
- klib (IR serialization)
  - Не анализировать код несколько раз

Планы

- FIR
- IR interpreter
- klib (IR serialization)
  - Не анализировать код несколько раз
  - debug/release сборки

Планы

- FIR
- IR interpreter
- klib (IR serialization)
- IR inlining

Планы

- FIR
- IR interpreter
- klib (IR serialization)
- IR inlining
  - Не инлайнить байткод

Планы

- FIR
- IR interpreter
- klib (IR serialization)
- IR inlining
  - Не инлайнить байткод
  - Рабочая* tail-call оптимизация

* во всех случаях, а не только в самых простых

Планы

- FIR
- IR interpreter
- klib (IR serialization)
- IR inlining
- WASM BE

Планы

- FIR
- IR interpreter
- klib (IR serialization)
- IR inlining
- WASM BE
- Expression Trees

Контакты

- TG: @ilmirus
- koltinlang.slack.com: Ilmir Usmanov [JB]

Q&A