Kotlin/Native. Final step on the way to multiplatform projects on Kotlin

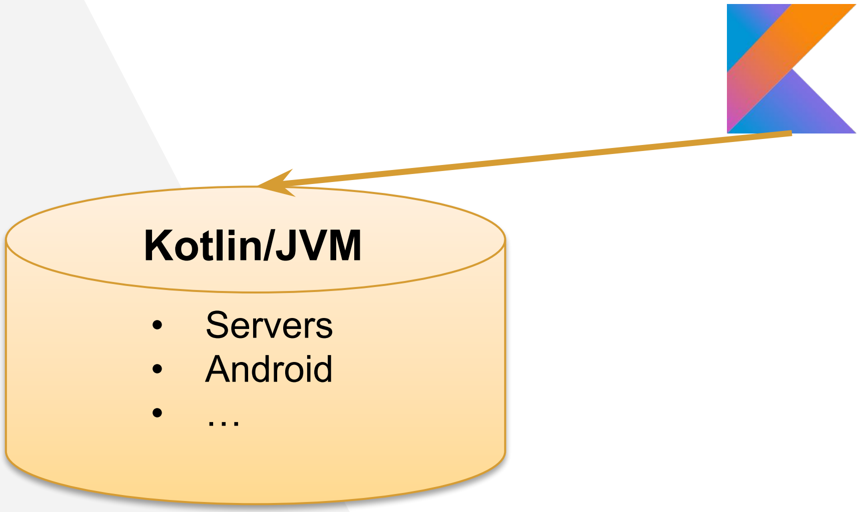
Elena Lepilkina, JetBrains

Kotlin

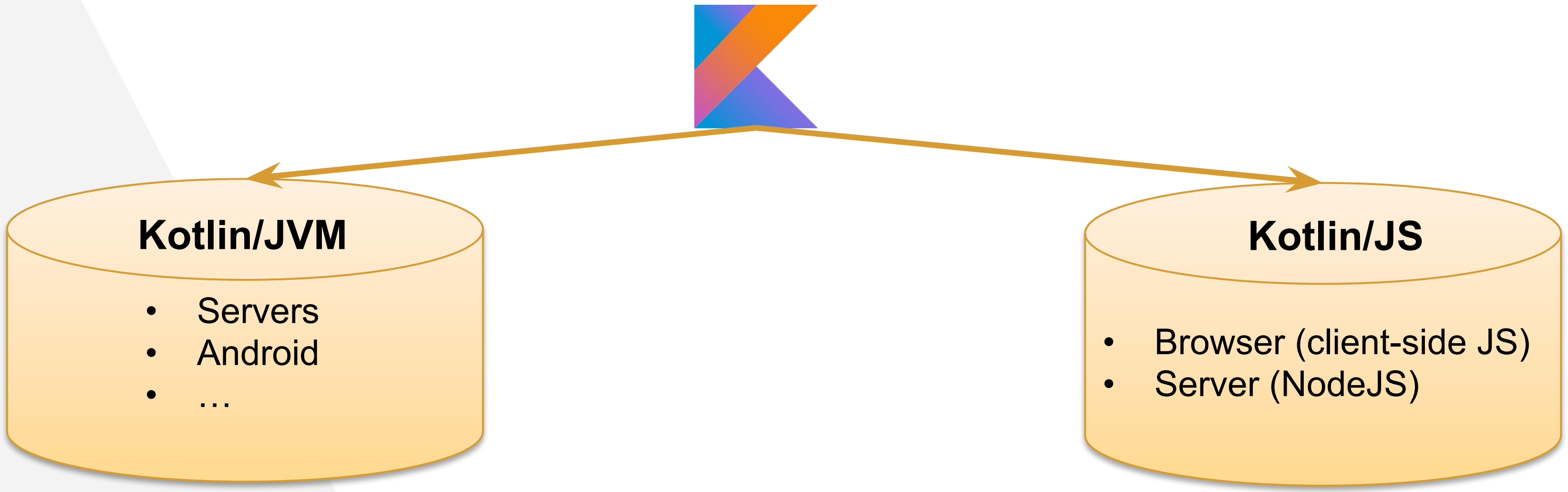
- Cross-platform, statically typed, general-purpose programming language with type inference. Kotlin started as a language to target the JVM.

World of modern Kotlin

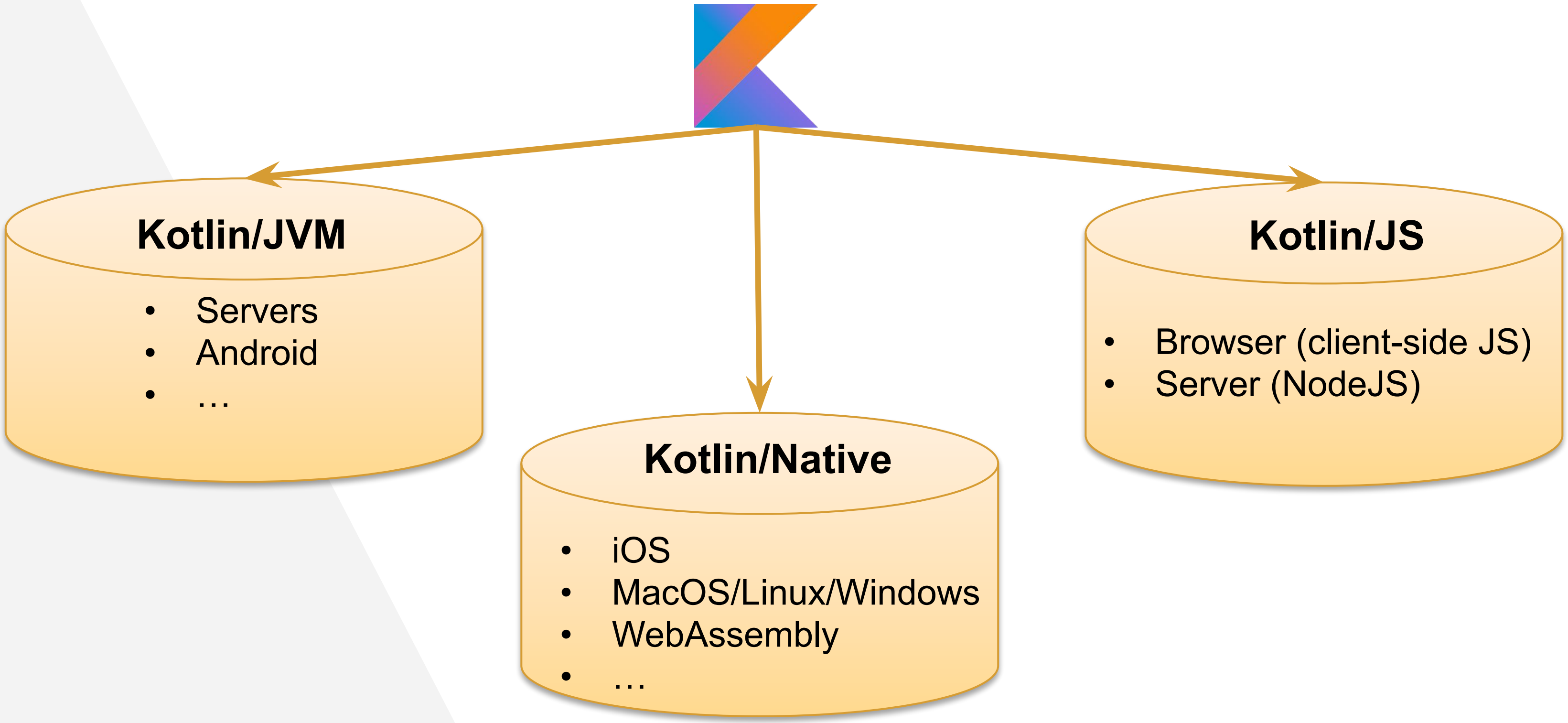
World of modern Kotlin



World of modern Kotlin



World of modern Kotlin

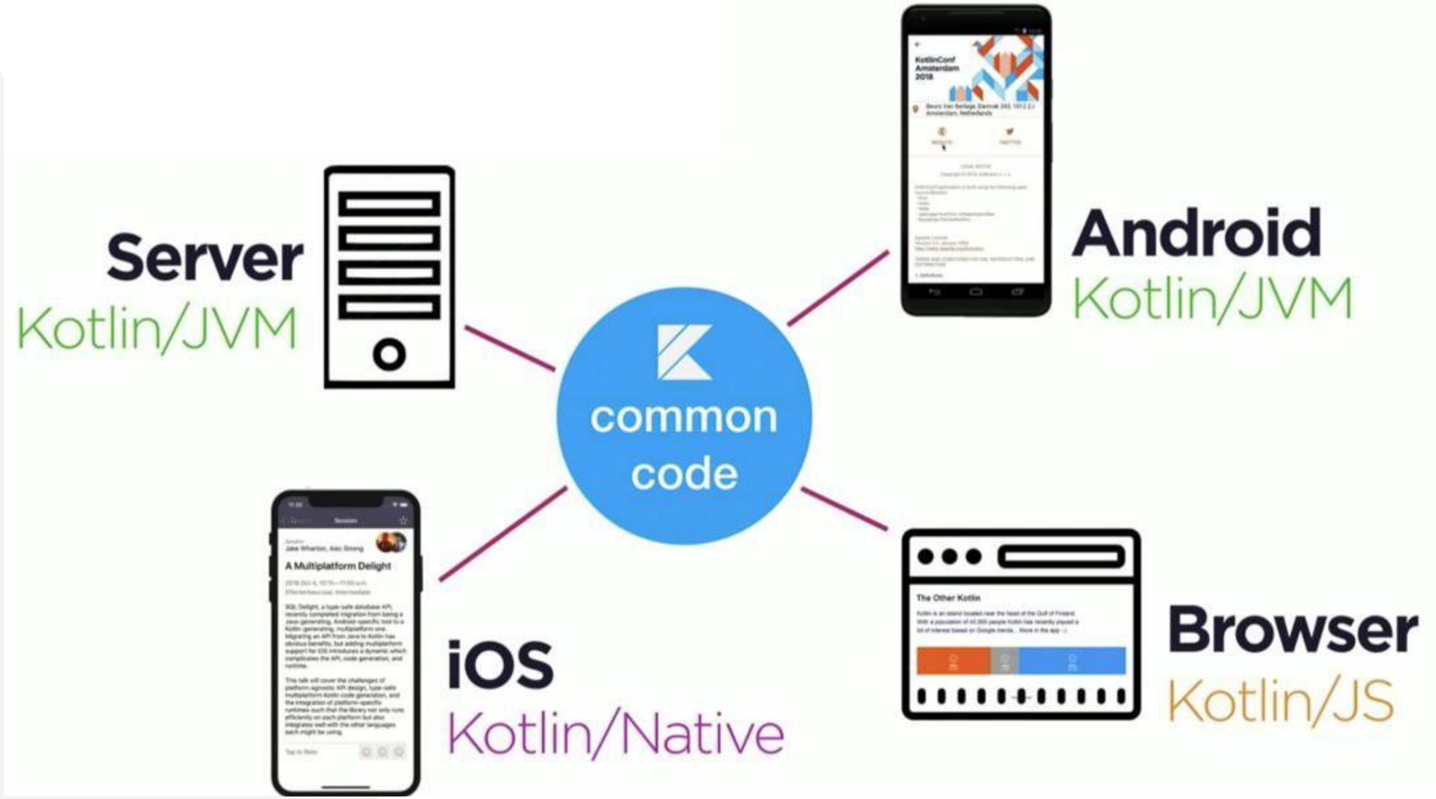


Programmers working with high-level languages achieve better productivity and quality than those working with lower-level languages.

Good code is its own best documentation.

— Steve McConnell "Code Complete"

Multiplatform concept



Best practices of programming are easy on Kotlin

Effective reusing code

Common part of project + Kotlin libraries + interoperability with other languages

Abstractions

Clear abstractions in right places

Managing complexity

Separating on subsystems by main responsibility

Central control points

In common code

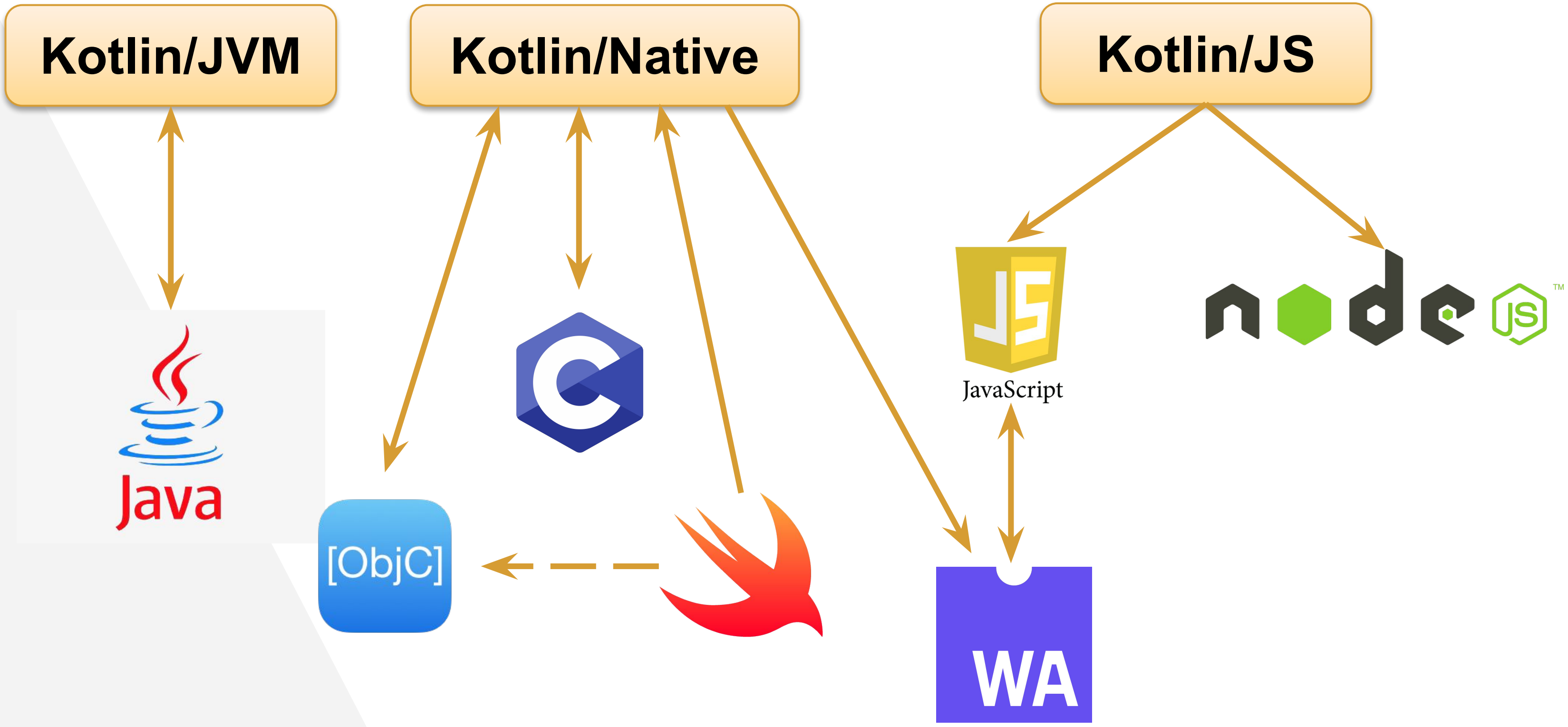
Iterative developing

Adding targets one by one + interoperability

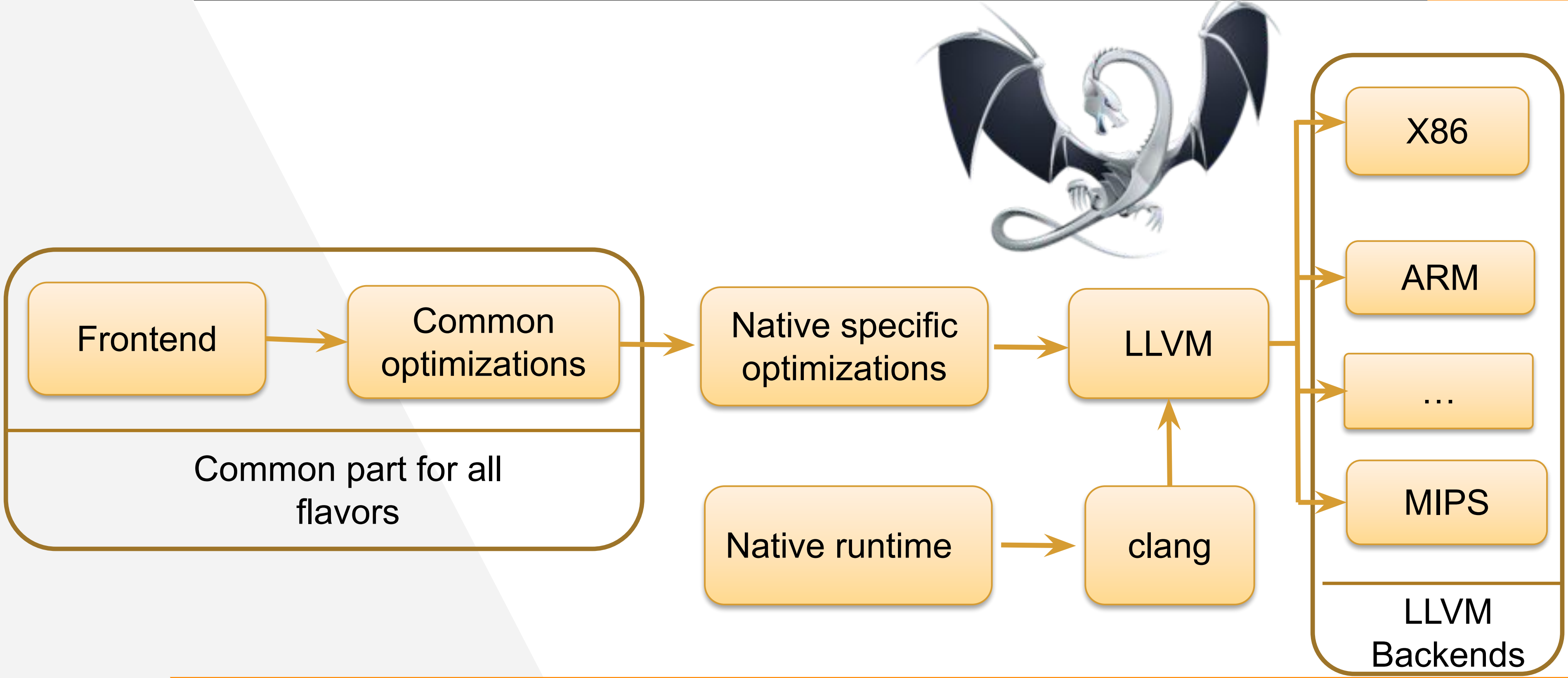
High-level and low-level programming

Access to targets API straight way

Interoperability



Kotlin/Native Compiler



How to make JVM and Native worlds become friends

Kotlin platform libraries

- platfrom.posix
- platform.linux
- platform.windows
- platform.osx
- etc.

Kotlin stdlib

- kotlin.collections
- kotlin.math
- etc.

Kotlin MPP libraries

- kotlinx.serialization
- kotlinx.coroutines
- kotlinx-io

How to make JVM and Native worlds become friends

C world

- pointer
- struct
- lvalue
- rvalue
- macros
- etc.

JVM world

Kotlin/Native approach

Equality of similar features

- C enum = Kotlin enum || int
- C struct = Kotlin class
- C typedef = Kotlin typealias

Extensions for strings

- kotlinString.cstr
- CPointer<ByteVar>.toKString()

Abstractions for absent definitions

- CPointer<T>
- CVariable
- CValues
- CValuesRef
- StableRef
- staticCFunction

Automatic memory management

- Kotlin/Native has GC reference-counter based algorithm in runtime
- It works with Objective C ARC
- Cycle collector based on the trial deletion

Manual memory management

- fun <reified T : CVariable> alloc(): T
- usePinned
- nativeHeap (needs free!)
- memScoped

```
val fileSize = memScoped {
    val statBuf = alloc<stat>()
    val error = stat("/", statBuf.ptr)
    statBuf.st_size
}
```

Objective C/Swift Interoperability

Kotlin	Swift	Objective-C
class	class	@interface
Extension	Extension	Category member
companion member <-	Class method or property	Class method or property
MutableList	NSMutableArray	NSMutableArray
...	...	...
Function type	Function type	Block pointer type

https://github.com/JetBrains/kotlin-native/blob/master/OBJC_INTEROP.md

Usage of cinterop

def file for C interop

```
headers = curl/curl.h
headerFilter = curl/*
linkerOpts.osx = -L/opt/local/lib
-L/usr/local/opt/curl/lib -lcurl
linkerOpts.linux = -L/usr/lib64
-L/usr/lib/x86_64-linux-gnu -lcurl
linkerOpts.mingw = -lcurl


---


/* C code */
```

def file for Objective C interop

```
package =
org.jetbrains.complexNumbers
language = Objective-C
```

Multiplatform projects. Expect/actual

```
expect class Logger {
    fun log(message: String)
}
```

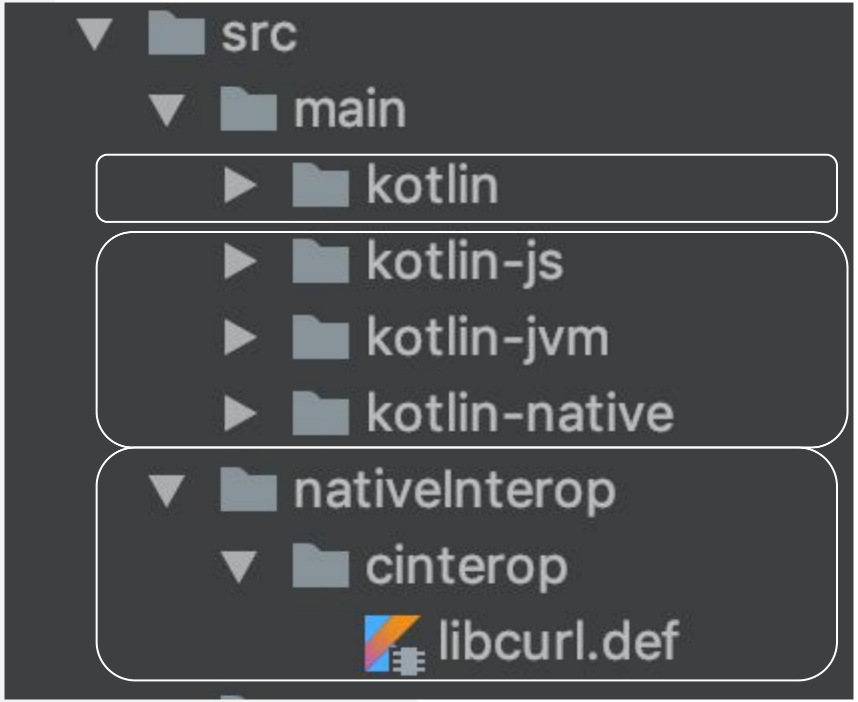
```
import platform.Foundation.*

actual class Logger() {
    fun log(message: String) {
        NSLog(message)
    }
}
```

```
import android.util.Log

actual class Logger {
    fun log(message: String) {
        Log.i("Tag", message)
    }
}
```

Multiplatform project. Example. Common part.



Common code

Target specific code

Def files for Kotlin/Native interop

Multiplatform project. Example. Common part.

```kotlin
// Implemented common part.
class FieldChange<T>(val field: String, val previous: T, val current: T) {
    ...
}

// Declaring target specific things.
expect fun Double.format(decimalNumber: Int = 4): String
expect fun writeToFile(fileName: String, text: String)
expect fun readFile(fileName: String): String
expect class ComplexNumber
```

Multiplatform project. Example. JVM target.

```
actual fun readFile(fileName: String): String {
    val inputStream = File(fileName).inputStream()
    val inputString = inputStream.bufferedReader().use { it.readText() }
    return inputString
}

actual fun Double.format(decimalNumber: Int): String = "%.${decimalNumber}f".format(this)

actual fun writeToFile(fileName: String, text: String) {
    File(fileName).printWriter().use { out -> out.println(text) }
}

actual class ComplexNumber { ...}
```

Multiplatform project. Example. Native target.

```
actual typealias ComplexNumber = Complex // Complex - struct from C library

actual fun readFile(fileName: String): String {...}

actual fun Double.format(decimalNumber: Int): String { ... }

actual fun writeToFile(fileName: String, text: String) {
    val file = fopen(fileName, "wt") ?: error("Cannot write file '$fileName'")
    try {
        if (fputs(text, file) == EOF) throw Error("File write error")
    } finally {
        fclose(file)
    }
}
```

Multiplatform project. Example. JS target.

```kotlin
actual typealias ComplexNumber = math.complex // from math.js

actual fun readFile(fileName: String): String {
    error("Reading from local file for JS isn't supported")
}

actual fun Double.format(decimalNumber: Int): String = this.asDynamic().toFixed(decimalNumber)

actual fun writeToFile(fileName: String, text: String) {
    if (fileName != "html")
        error("Writing to local file for JS isn't supported")
    val bodyPart = text.substringAfter("<body>").substringBefore("</body>")
    document.body?.innerHTML = bodyPart
}
```

Multiplatform project. Example. JS specific.

```
// API for interop with JS library Chartist.
external object Chartist {
    class Svg(form: String, parameters: dynamic, chartArea: String)
    val plugins: ChartistPlugins
    val Interpolation: dynamic
    fun Line(query: String, data: dynamic, options: dynamic): dynamic
}


val chart = Chartist.Line("#chart", getChartData(labels, time.values),
            getChartOptions(samples.keys.toTypedArray(), "Time"))
```

Multiplatform project. Example. JS specific.

```
js("$('#inputGroupBuild')").change({
        val newValue = js("$(this).val()")
        if (newValue != parameters["type"]) {
            window.location.href = "http://some-link.com"
        }
    })
```

Multiplatform project. Example. Build script.

```
apply plugin: 'kotlin-multiplatform'


kotlin {
    sourceSets {
        commonMain {
            dependencies { implementation "org.jetbrains.kotlin:kotlin-stdlib-common" }
            kotlin.srcDir 'src/main/kotlin'
        }
        nativeMain {
            dependsOn commonMain
            kotlin.srcDir 'src/main/kotlin-native'
        }
```

Multiplatform project. Example. Build script.

```
...
        jvmMain {
            dependencies {implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8" }
            kotlin.srcDir 'src/main/kotlin-jvm'
        }
        jsMain {
            dependencies { implementation "org.jetbrains.kotlin:kotlin-stdlib-js" }
            kotlin.srcDir 'src/main/kotlin-js'
        }
        linuxMain { dependsOn nativeMain }
        windowsMain { dependsOn nativeMain }
        macosMain {dependsOn nativeMain }
    }
```

Multiplatform project. Example. Build script.

```
jvm() {
    compilations.all {
        tasks[compileKotlinTaskName].kotlinOptions.suppressWarnings = true
    }
}

targetFromPreset(presets.mingwX64, 'windows') {
    compilations.main.cinterops {
        libcurl {
            includeDirs.headerFilterOnly "${getMingwPath()}/include"
        }
    }
}
```

Multiplatform project. Example. Build script.

```
targetFromPreset(presets.linuxX64, 'linux') {
    compilations.main.cinterops {
        libcurl {
            includeDirs.headerFilterOnly '/usr/include', '/usr/include/x86_64-linux-gnu'
        }
    }
}

targetFromPreset(presets.macosX64, 'macos') {
   ...
}

js()
```

Multiplatform project. Example. Build script.

```
    configure([windows, linux, macos]) {
        binaries.all {
            linkTask.enabled = isCurrentHost
        }

        binaries {
            executable('myExecutable', [RELEASE])
        }
    }
}
```

Current status

- Kotlin/Native is in beta phase.
- MPP plugin is changing. It should become better!
- You can influence technology development providing use cases where it can work better.

THANKS!

Any questions?

https://kotlinlang.slack.com/

https://github.com/JetBrains/kotlin-native