Kotlin 1.4 Online Event

Diving into Kotlin Multiplatform

Dimitry Savvinov

@dsavvinov

October 14, 2020

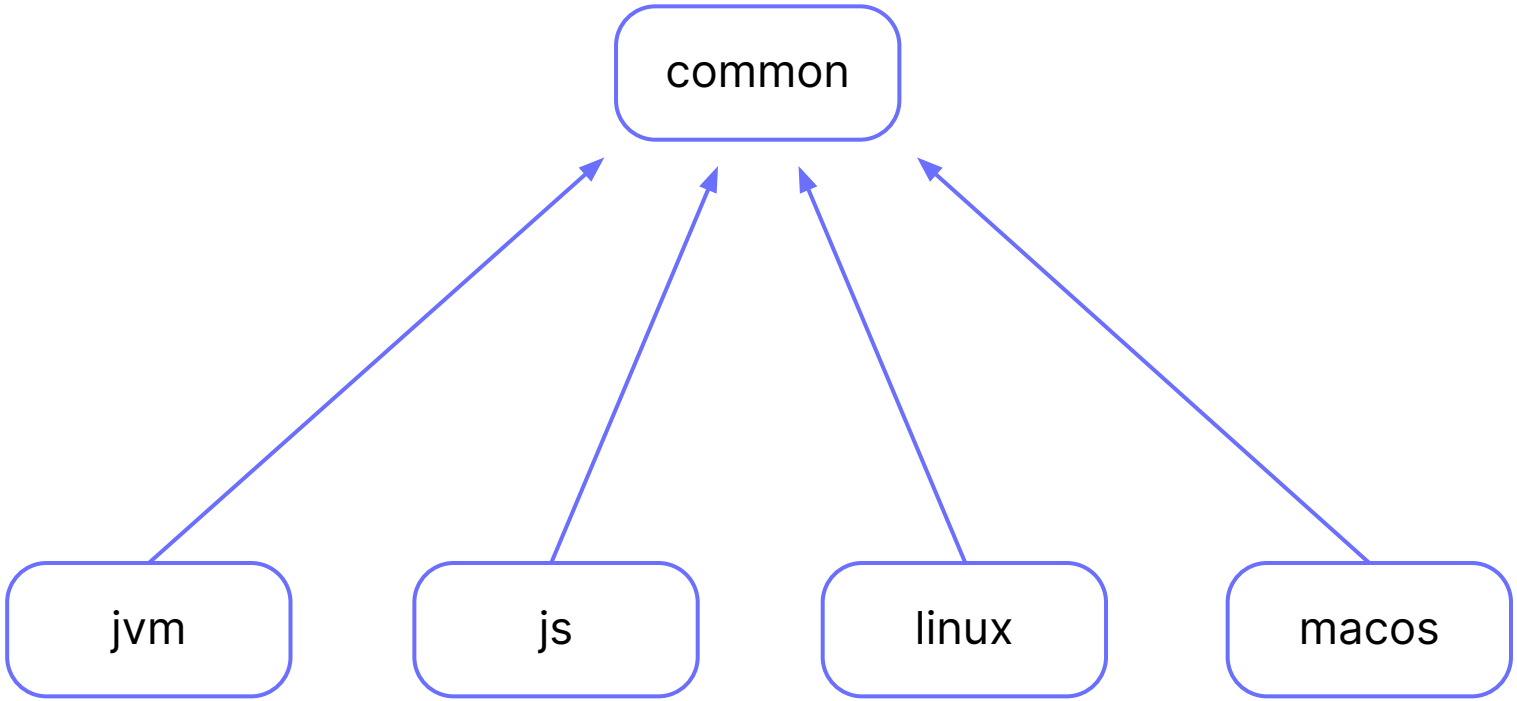
Disclaimer

A lot of internals ahead:

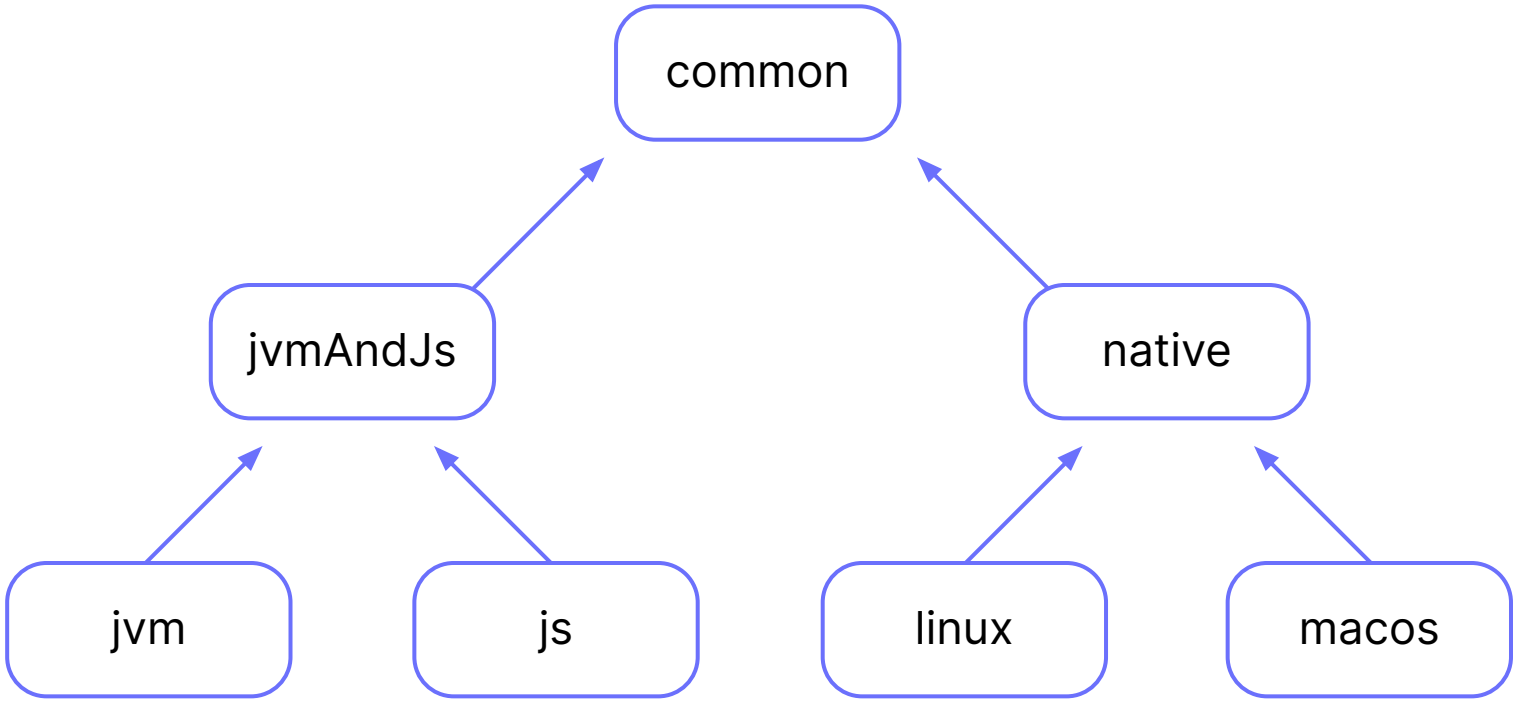
- Information is this talk might become outdated in the future
- The actual documentation is the only source of truth
- Some parts are totally internal (will be marked so explicitly)

1.4 MPP:
The story of one feature

1.3: Two levels of hierarchy

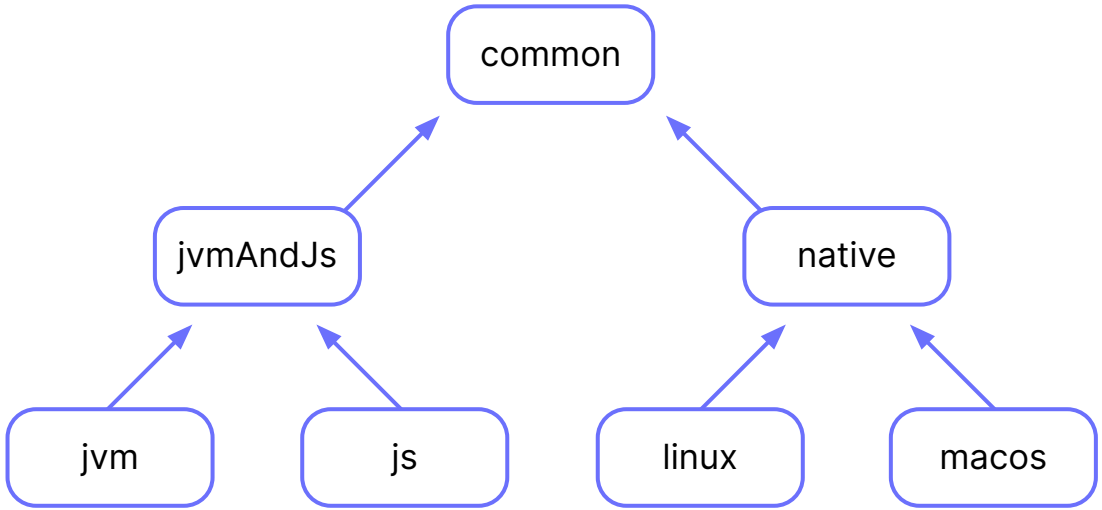


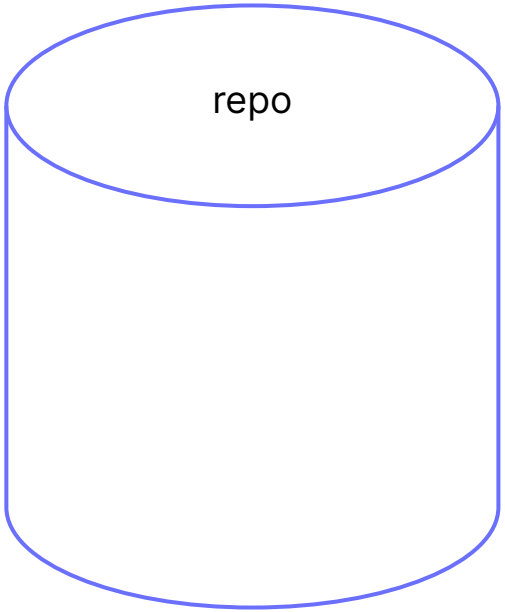
1.4: Multiple levels of hierarchy



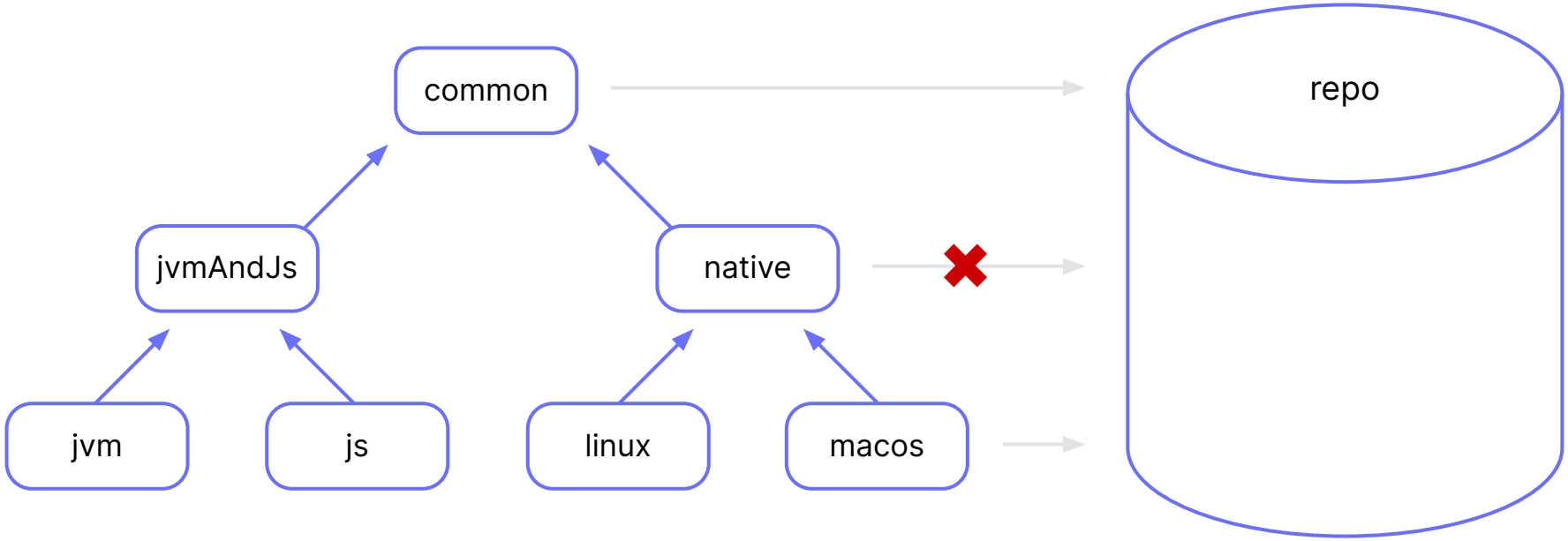
Multiplatform Publishing

Publishing



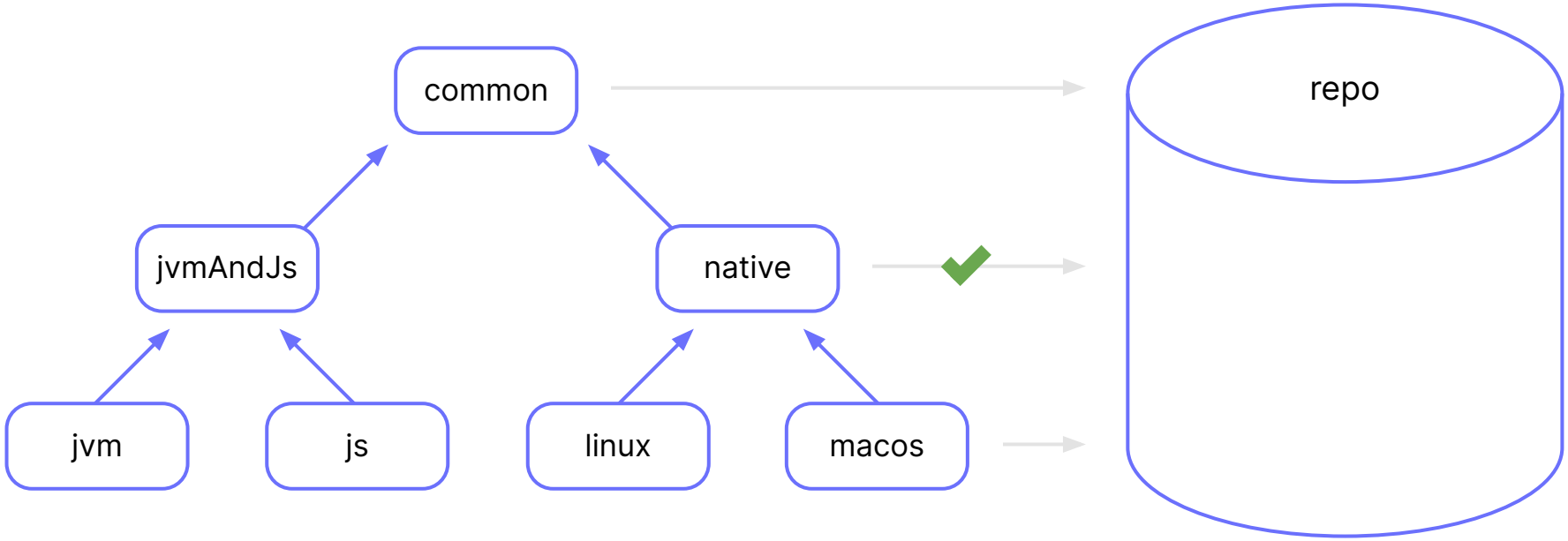


Publishing



In 1.3: intermediate source sets are not published

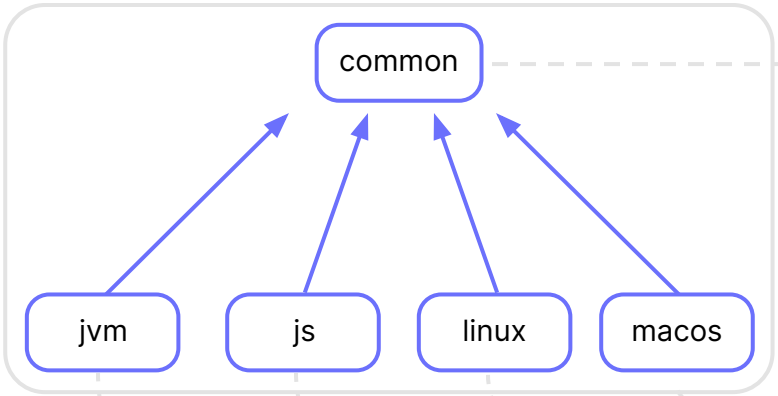
Publishing



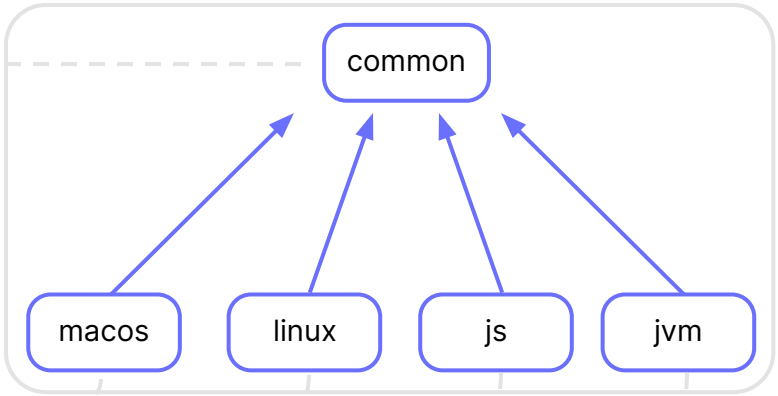
In 1.4: all source sets are published

Why do we need intermediate source sets to be published?

Your project

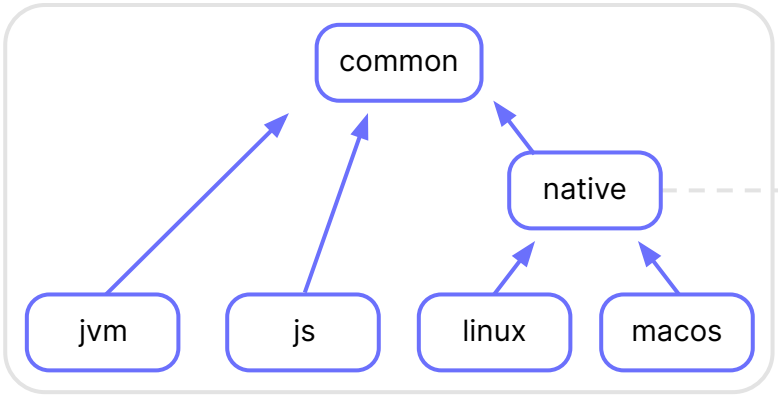


Library you depend on

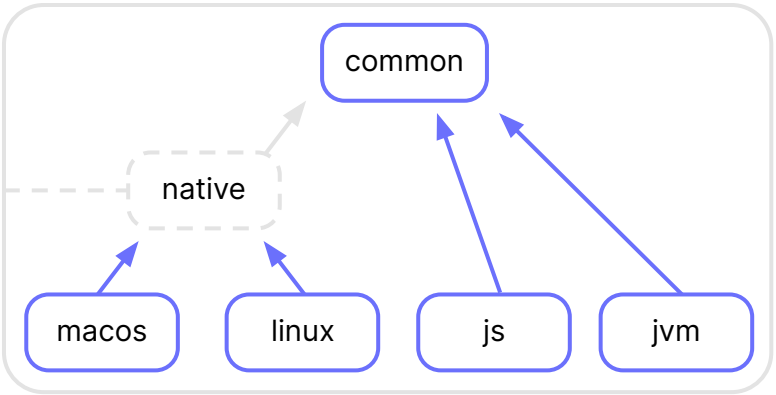


Why do we need intermediate source sets to be published?

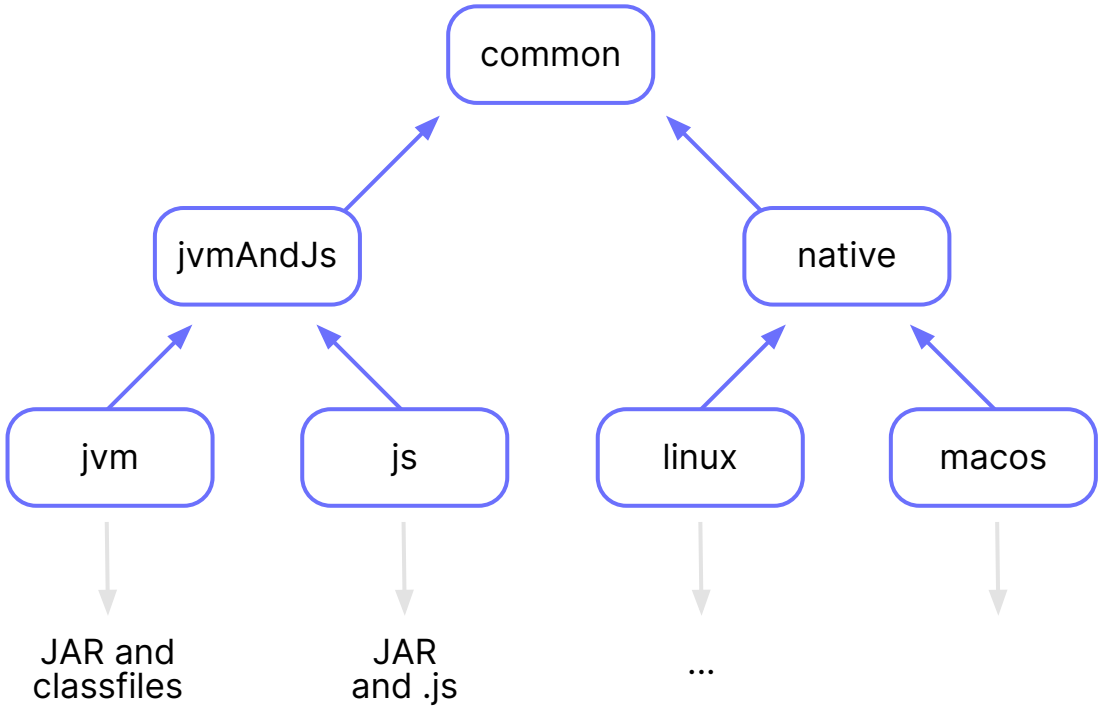
Your project



Library you depend on

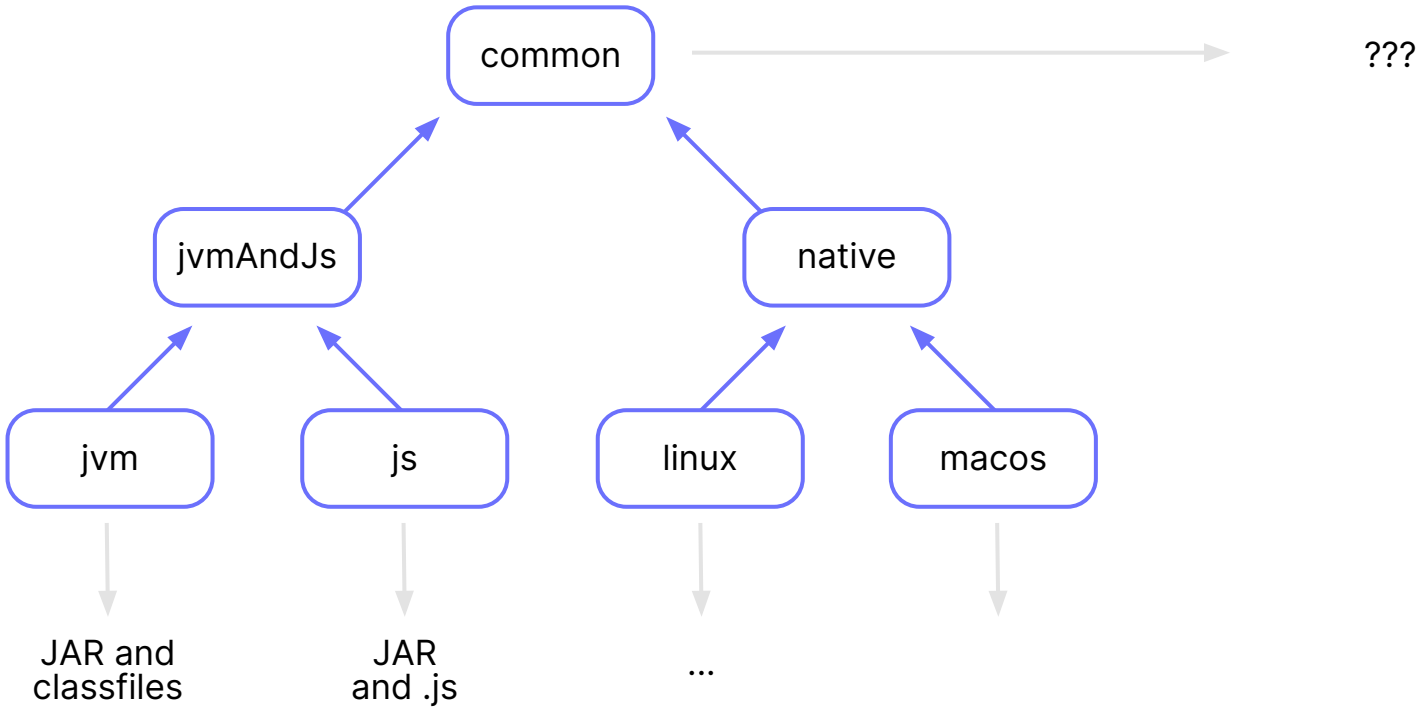


Kotlin Publishing Model in a nutshell

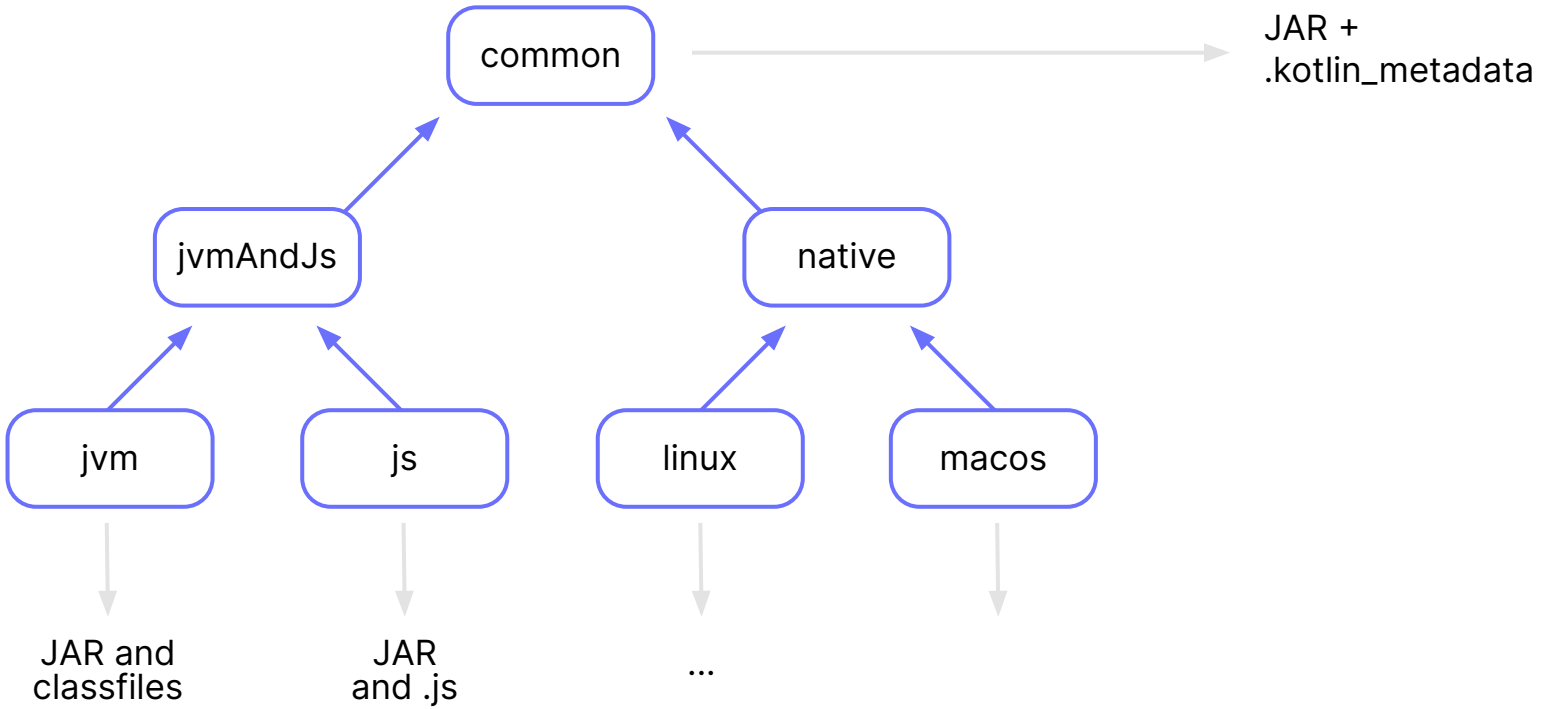


Kotlin tries to reuse as much of the platform ecosystem as possible

Kotlin Publishing Model in a nutshell



Kotlin Publishing Model in a nutshell



Experiments section: publishing in 1.3

How I can play with a published library?

1. build.gradle

```
plugins {
    id 'maven-publish'
}

group = '<group.id>'

version = '<my-version>'
```

settings.gradle

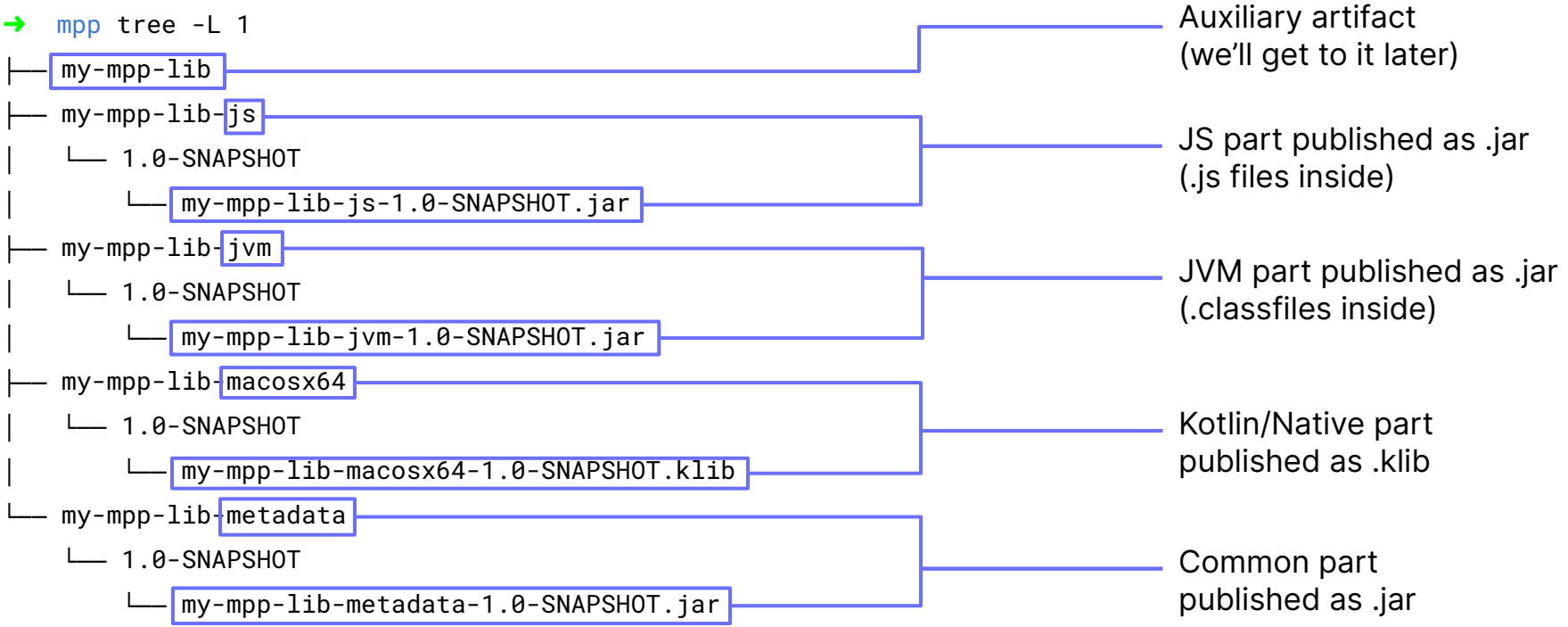
```
rootProject.name = '<project-name>'
```

2. `$ ./gradlew publishToMavenLocal`

3. `$ cd ~/.m2/repository/<group>/<id>/<project-name>`

Example: MPP Library in 1.3

```
➜  mpp pwd
/Users/dmitry.savvinov/.m2/repository/hello/mpp
➜  mpp tree -L 1
├── my-mpp-lib
├── my-mpp-lib-js
│   └── 1.0-SNAPSHOT
│       └── my-mpp-lib-js-1.0-SNAPSHOT.jar
├── my-mpp-lib-jvm
│   └── 1.0-SNAPSHOT
│       └── my-mpp-lib-jvm-1.0-SNAPSHOT.jar
├── my-mpp-lib-macosx64
│   └── 1.0-SNAPSHOT
│       └── my-mpp-lib-macosx64-1.0-SNAPSHOT.klib
└── my-mpp-lib-metadata
    └── 1.0-SNAPSHOT
        └── my-mpp-lib-metadata-1.0-SNAPSHOT.jar
```

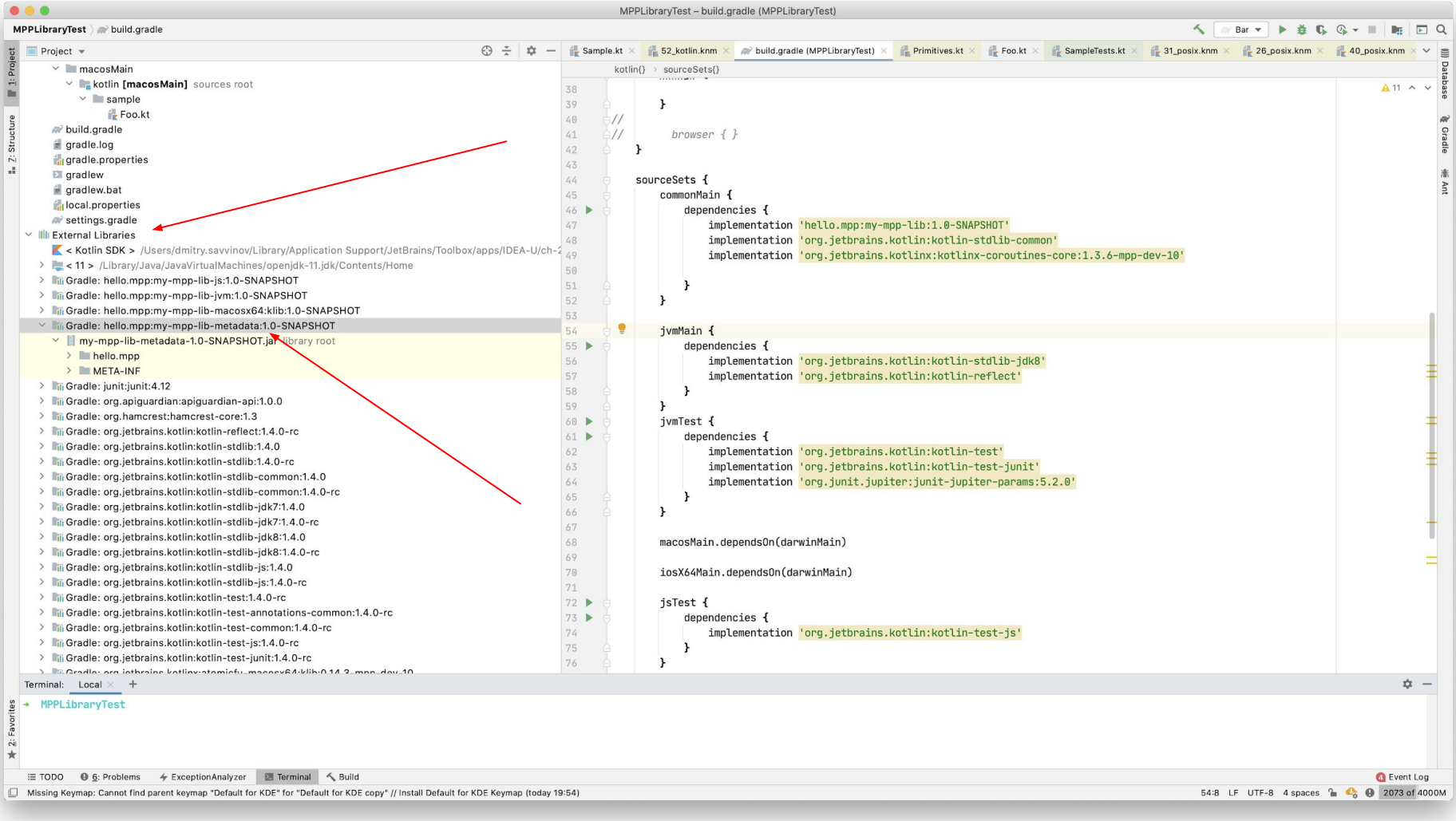


How can I peek into .kotlin_metadata?

1. Publish it into a maven local as described in the previous steps
2. Create a stub-project and add a dependency from it on a published library: build.gradle

```
repositories {
    ...
    mavenLocal()
}
...
sourceSets {
    commonMain {
        dependencies {
            implementation '<group>.<id>:<project-name>:<version>'
        }
    }
    ...
}
```

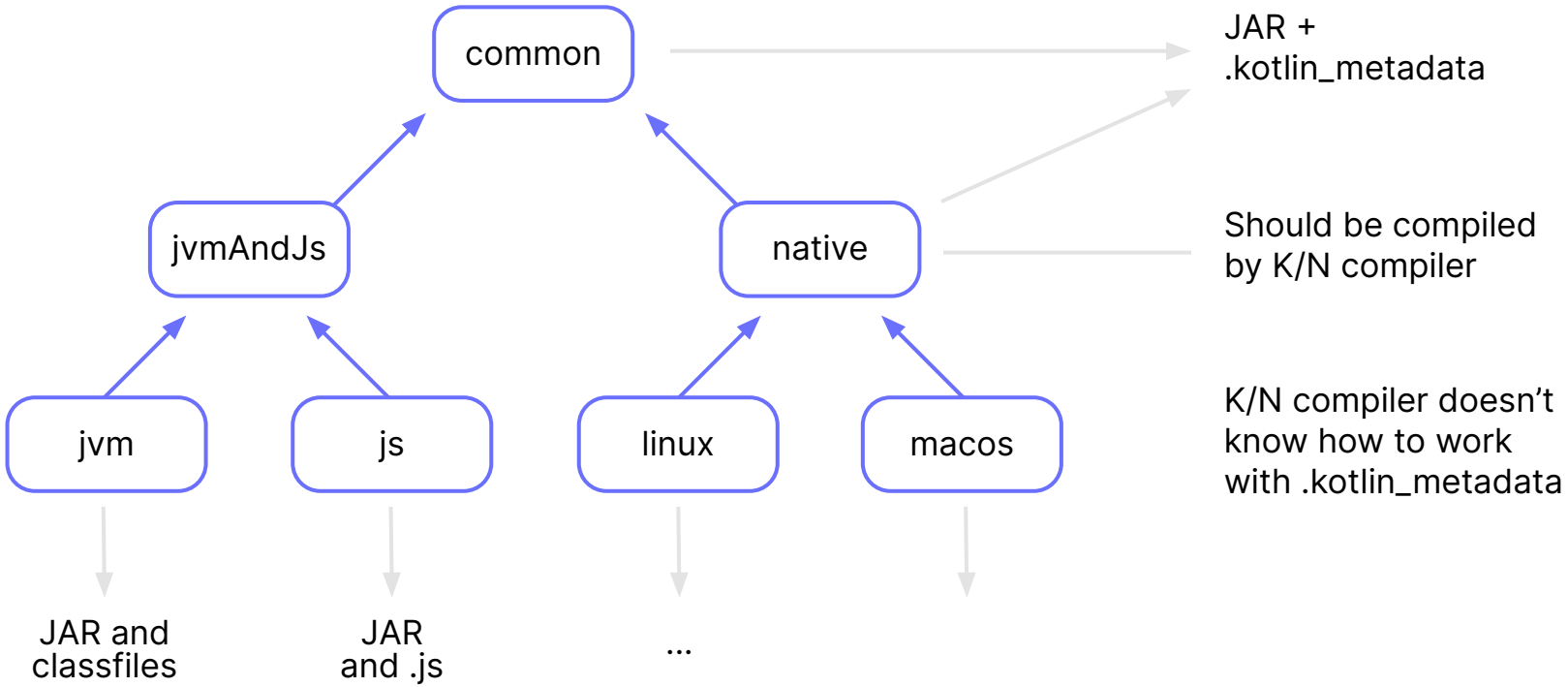
3. Open in the IDE and see “Externals Libraries” in the “Project Structure”


MPPLibraryTest – build.gradle (MPPLibraryTest)
MPPLibraryTest
build.gradle
Project
macosMain
kotlin [macosMain] sources root
sample
Foo.kt
build.gradle
gradle.log
gradle.properties
gradlew
gradlew.bat
local.properties
settings.gradle
External Libraries
< Kotlin SDK > /Users/dmitry.savvinov/Library/Application Support/JetBrains/Toolbox/apps/IDEA-U/ch-2
< 11 > /Library/Java/JavaVirtualMachines/openjdk-11.jdk/Contents/Home
Gradle: hello.mpp:my-mpp-lib-js:1.0-SNAPSHOT
Gradle: hello.mpp:my-mpp-lib-jvm:1.0-SNAPSHOT
Gradle: hello.mpp:my-mpp-lib-macosx64:klib:1.0-SNAPSHOT
Gradle: hello.mpp:my-mpp-lib-metadata:1.0-SNAPSHOT
my-mpp-lib-metadata-1.0-SNAPSHOT.jar library root
hello.mpp
META-INF
Gradle: junit:junit:4.12
Gradle: org.apiguardian:apiguardian-api:1.0.0
Gradle: org.hamcrest:hamcrest-core:1.3
Gradle: org.jetbrains.kotlin:kotlin-reflect:1.4.0-rc
Gradle: org.jetbrains.kotlin:kotlin-stdlib:1.4.0
Gradle: org.jetbrains.kotlin:kotlin-stdlib:1.4.0-rc
Gradle: org.jetbrains.kotlin:kotlin-stdlib-common:1.4.0
Gradle: org.jetbrains.kotlin:kotlin-stdlib-common:1.4.0-rc
Gradle: org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.4.0
Gradle: org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.4.0-rc
Gradle: org.jetbrains.kotlin:kotlin-stdlib-jdk8:1.4.0
Gradle: org.jetbrains.kotlin:kotlin-stdlib-jdk8:1.4.0-rc
Gradle: org.jetbrains.kotlin:kotlin-stdlib-js:1.4.0
Gradle: org.jetbrains.kotlin:kotlin-stdlib-js:1.4.0-rc
Gradle: org.jetbrains.kotlin:kotlin-test:1.4.0-rc
Gradle: org.jetbrains.kotlin:kotlin-test-annotations-common:1.4.0-rc
Gradle: org.jetbrains.kotlin:kotlin-test-common:1.4.0-rc
Gradle: org.jetbrains.kotlin:kotlin-test-js:1.4.0-rc
Gradle: org.jetbrains.kotlin:kotlin-test-junit:1.4.0-rc
Sample.kt
52_kotlin.knm
build.gradle (MPPLibraryTest)
Primitives.kt
Foo.kt
SampleTests.kt
31_posix.knm
26_posix.knm
40_posix.knm
kotlin{} › sourceSets{}
}
//
// browser { }
}
sourceSets {
commonMain {
dependencies {
implementation 'hello.mpp:my-mpp-lib:1.0-SNAPSHOT'
implementation 'org.jetbrains.kotlin:kotlin-stdlib-common'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.6-mpp-dev-10'
}
}
jvmMain {
dependencies {
implementation 'org.jetbrains.kotlin:kotlin-stdlib-jdk8'
implementation 'org.jetbrains.kotlin:kotlin-reflect'
}
}
jvmTest {
dependencies {
implementation 'org.jetbrains.kotlin:kotlin-test'
implementation 'org.jetbrains.kotlin:kotlin-test-junit'
implementation 'org.junit.jupiter:junit-jupiter-params:5.2.0'
}
}
macosMain.dependsOn(darwinMain)
iosX64Main.dependsOn(darwinMain)
jsTest {
dependencies {
implementation 'org.jetbrains.kotlin:kotlin-test-js'
}
}
Terminal: Local
MPPLibraryTest
TODO
6: Problems
ExceptionAnalyzer
Terminal
Build
Event Log
Missing Keymap: Cannot find parent keymap "Default for KDE" for "Default for KDE copy" // Install Default for KDE Keymap (today 19:54)
54:8 LF UTF-8 4 spaces
2073 of 4000M

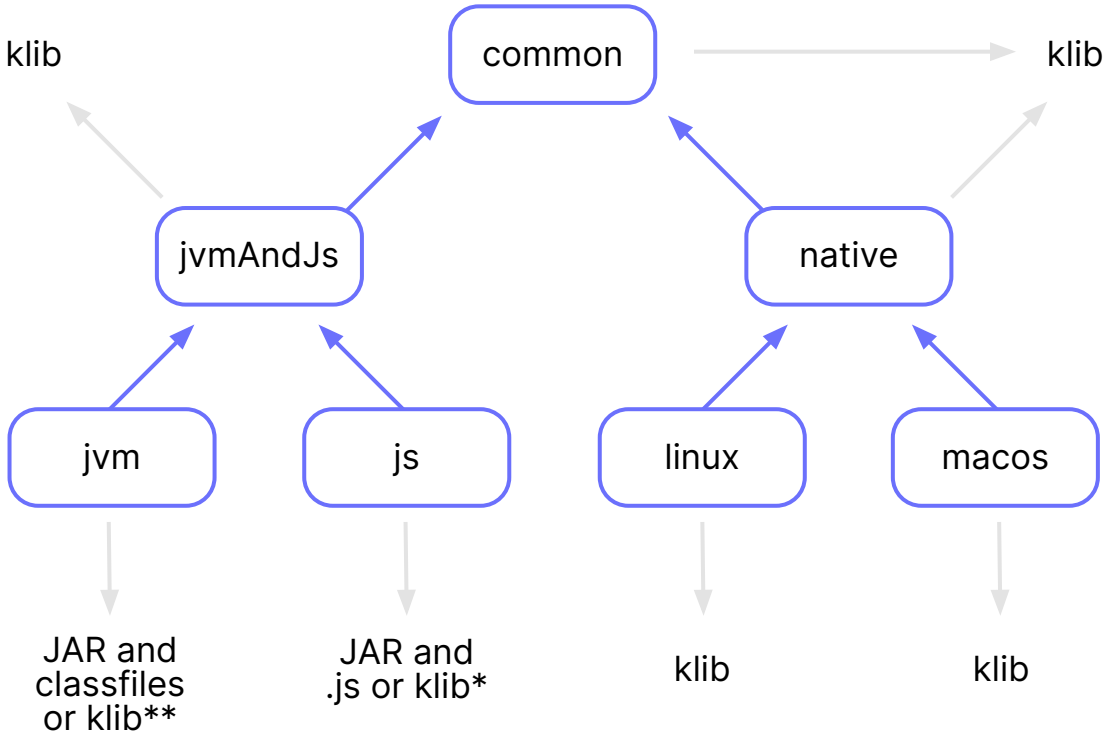
You can even browse the common part of stdlib!

```
// IntelliJ API Decompiler stub source generated from a class file
// Implementation of methods is not available

package kotlin

public final class String public constructor() : kotlin.Comparable<kotlin.String>, kotlin.CharSequence {
   public companion object {
   }

   public open val length: kotlin.Int /* compiled code */

   public open operator fun compareTo(other: kotlin.String): kotlin.Int { /* compiled code */ }

   public open operator fun equals(other: kotlin.Any?): kotlin.Boolean { /* compiled code */ }

   public open operator fun get(index: kotlin.Int): kotlin.Char { /* compiled code */ }

   public open fun hashCode(): kotlin.Int { /* compiled code */ }

 ...
```

1.4: Publishing intermediate source sets



1.4: Publishing everything into klibs



- Common denominator across all backends
- Can be easily evolved further

* experimental, under explicit opt-in

** in the future

Experiments section: publishing in 1.4

Publish an MPP library in a new format

1. build.gradle

```
plugins {
    id 'maven-publish'
}
group = '<group.id>'
version = '<my-version>'
```

settings.gradle

```
rootProject.name = '<project-name>'

kotlin.mpp.enableGranularSourceSetsMetadata=true
```

2. `$ ./gradlew publishToMavenLocal`

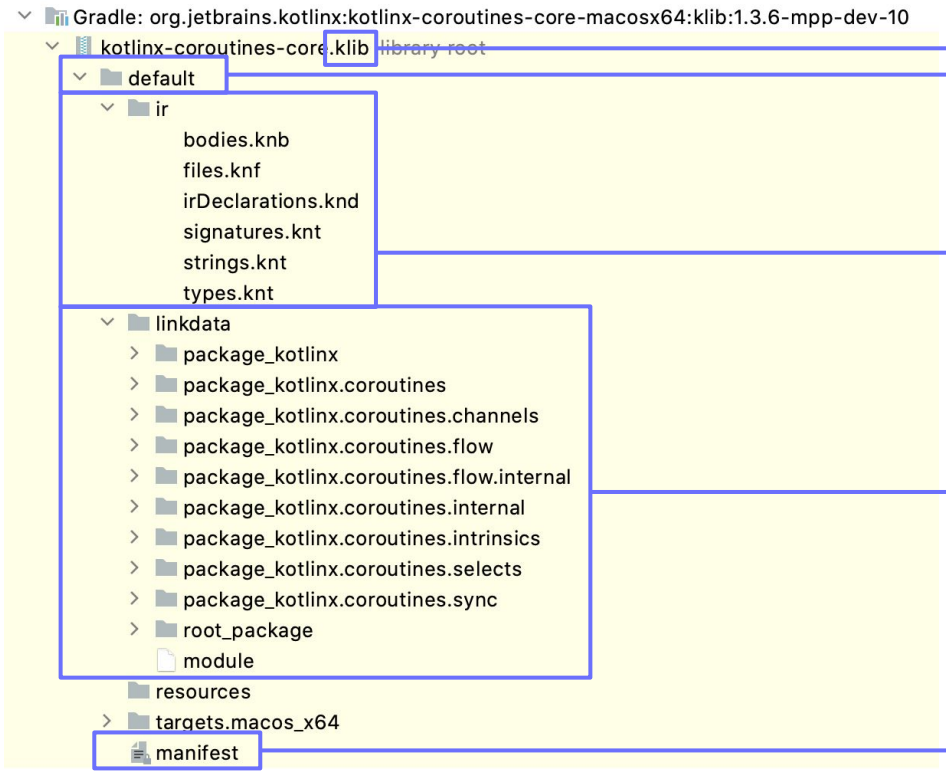
3. `$ cd ~/.m2/repository/<group>/<id>/<project-name>`

Example: MPP Library in 1.4

```
➜  mpp pwd
/Users/dmitry.savvinov/.m2/repository/hello/mpp
➜  mpp tree -L 1
├── my-mpp-lib
├── my-mpp-lib-js
│   └── 1.0-SNAPSHOT
│       └── my-mpp-lib-js-1.0-SNAPSHOT.jar
├── my-mpp-lib-jvm
│   └── 1.0-SNAPSHOT
│       └── my-mpp-lib-jvm-1.0-SNAPSHOT.jar
├── my-mpp-lib-macosx64
│   └── 1.0-SNAPSHOT
│       └── my-mpp-lib-macosx64-1.0-SNAPSHOT.klib
└── my-mpp-lib-metadata
    └── 1.0-SNAPSHOT
        └── my-mpp-lib-metadata-1.0-SNAPSHOT.klib
```

Common part published as .klib

A peek into klibs



.klib extension
packaged as .zip

Reserved multiple
variants for better
compatibility in the future

Bodies are stored in an
internal format called IR

.kotlin_metadata
is still present for
the sake of tooling

Key-value manifest
for internal purposes

Dependencies management

Dependencies management

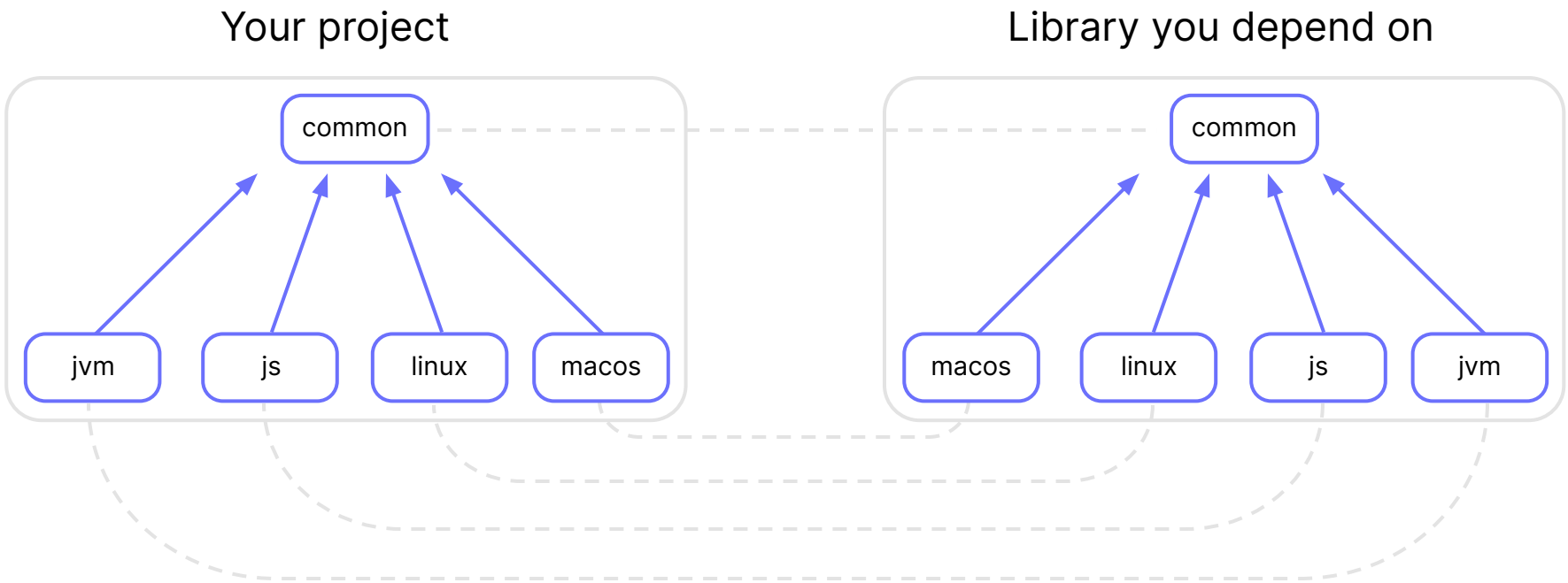
1.3

```
sourceSets.commonMain.dependencies {
    implementation 'my-library-common'
}

sourceSets.jvmMain.dependencies {
    implementation 'my-library-jvm'
}

sourceSets.jsMain.dependencies {
    implementation 'my-library-js'
}
```

1.4

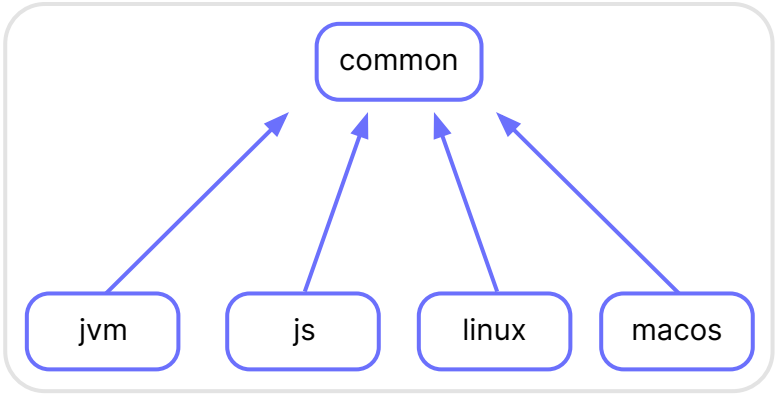
```
sourceSets.commonMain.dependencies {
    implementation 'my-library'
}
```

Simple case

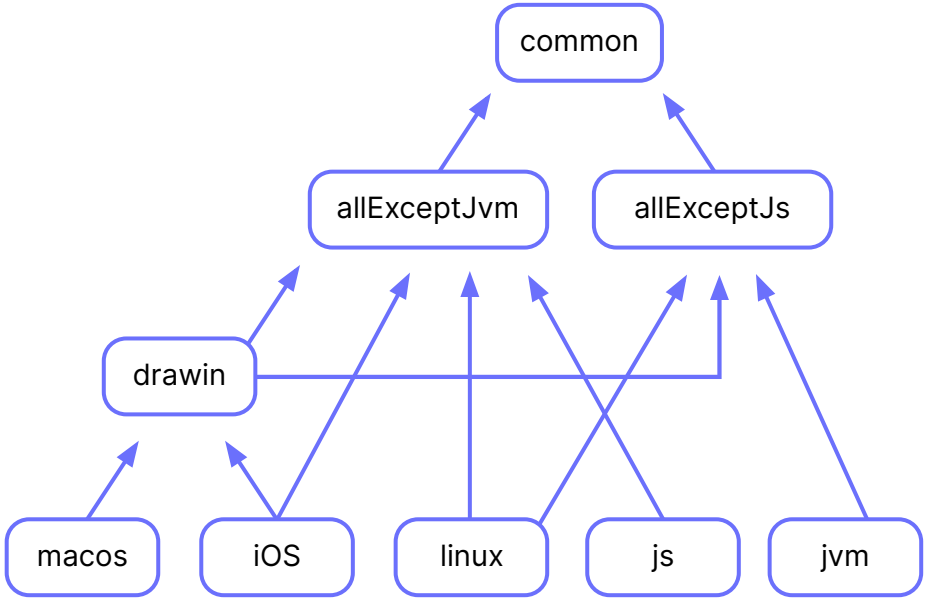

Your project
common
jvm
js
linux
macos
Library you depend on
common
macos
linux
js
jvm

Not So Simple case

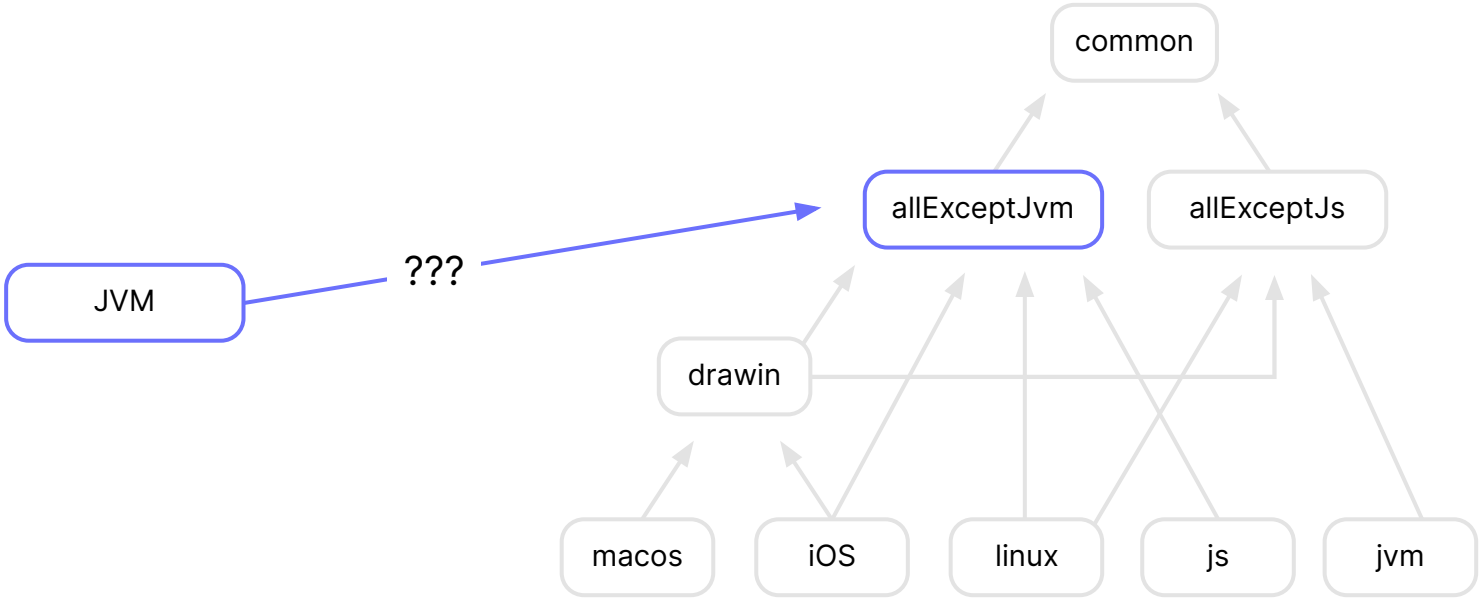
Your project



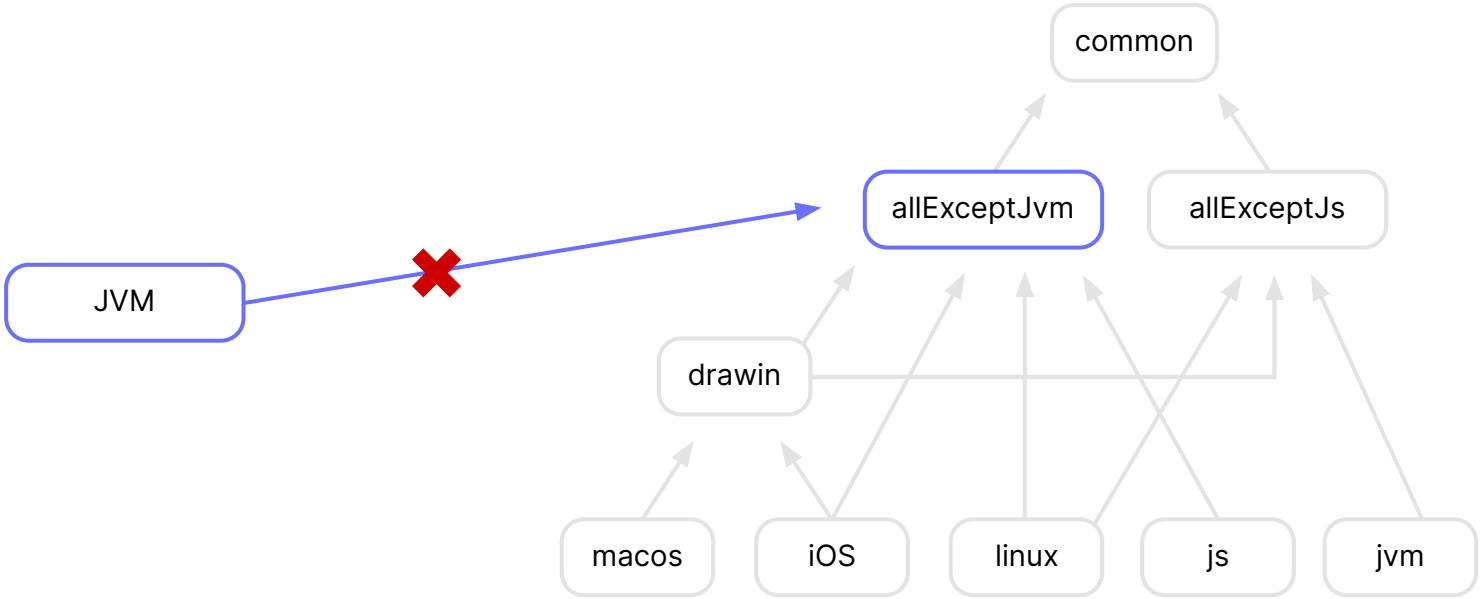
Library you depend on



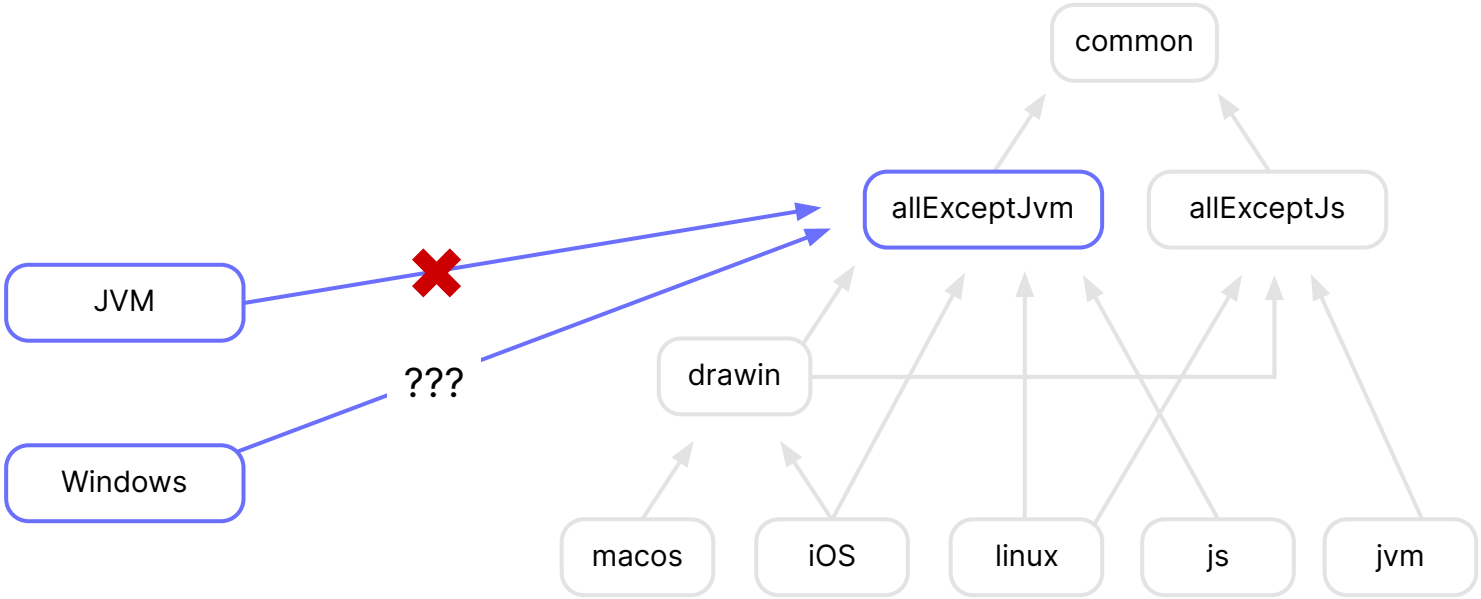
What are we even supposed to do?



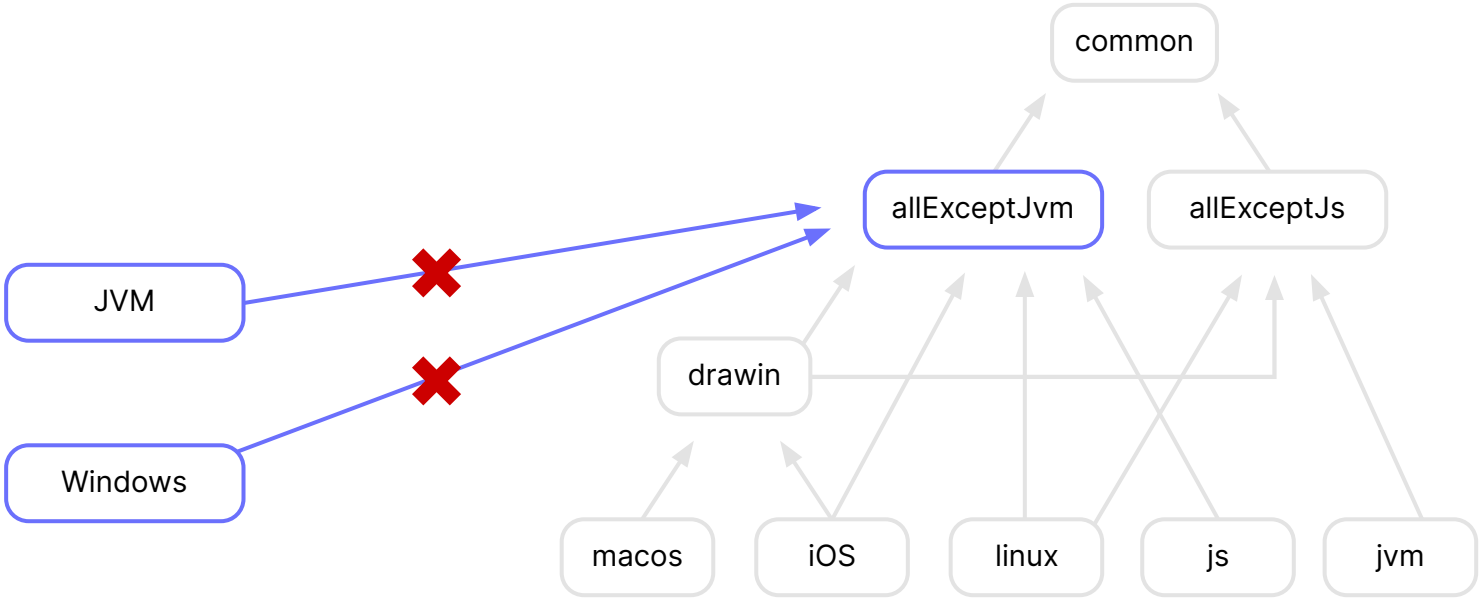
What are we even supposed to do?



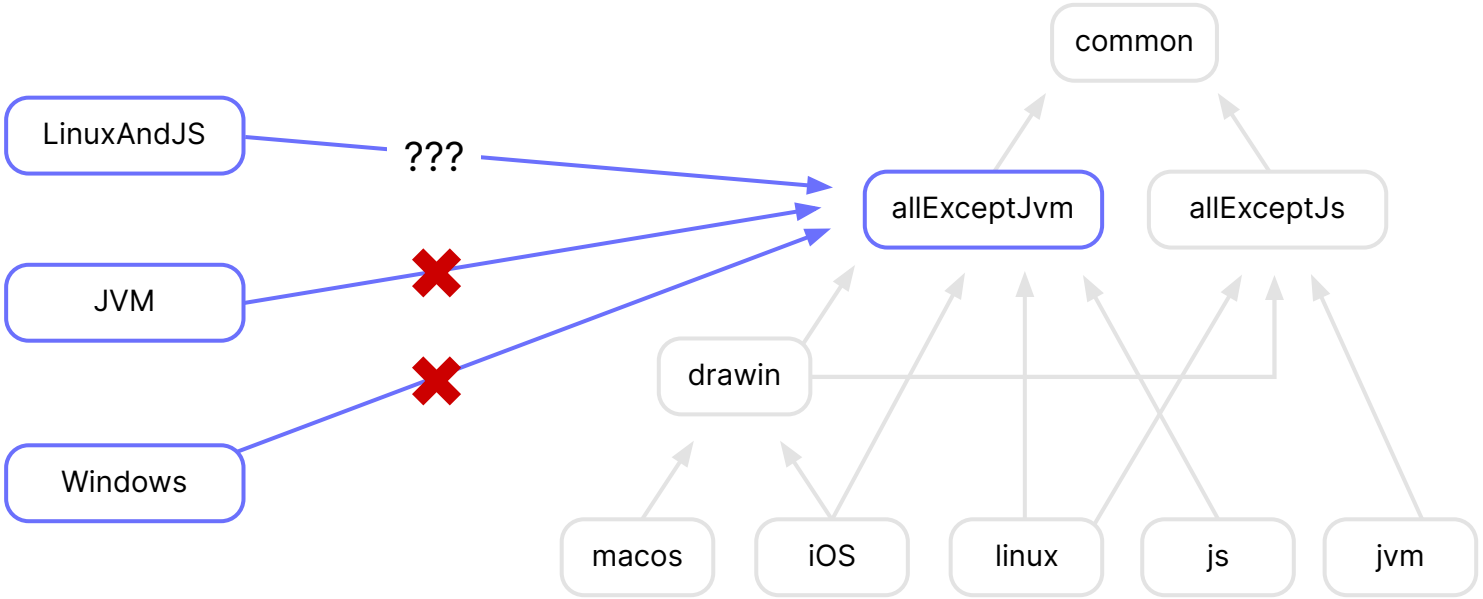
What are we even supposed to do?



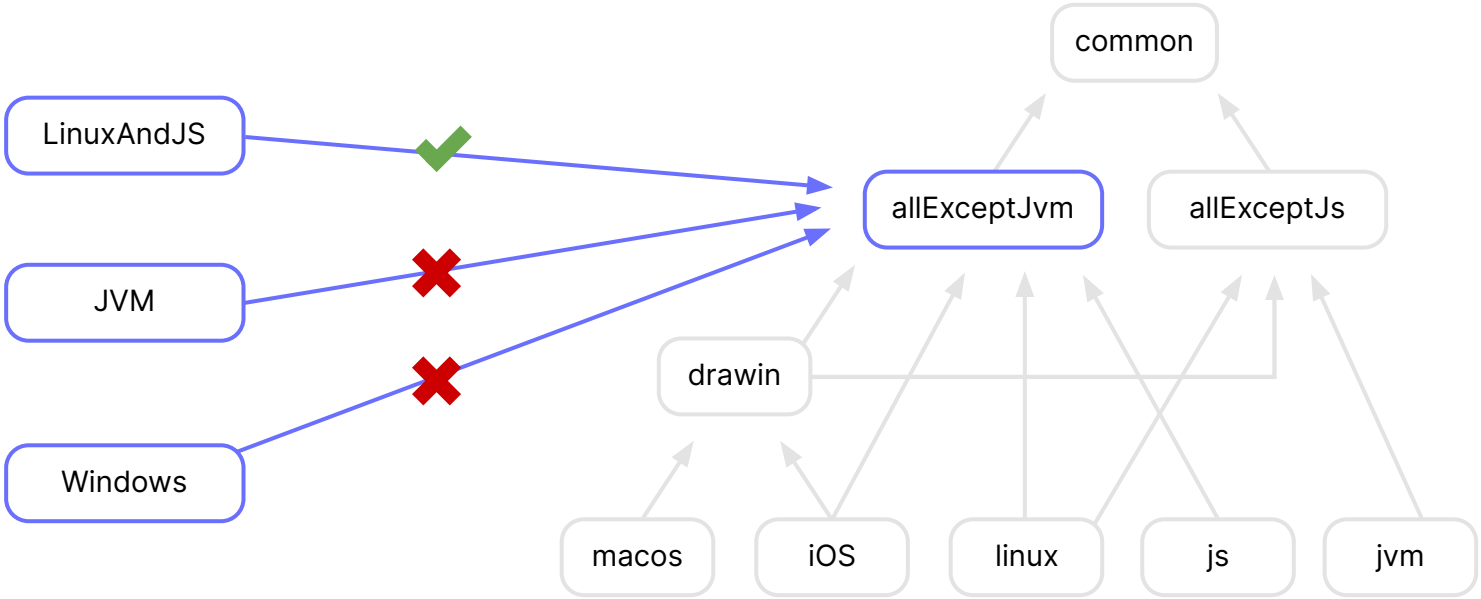
What are we even supposed to do?



What are we even supposed to do?

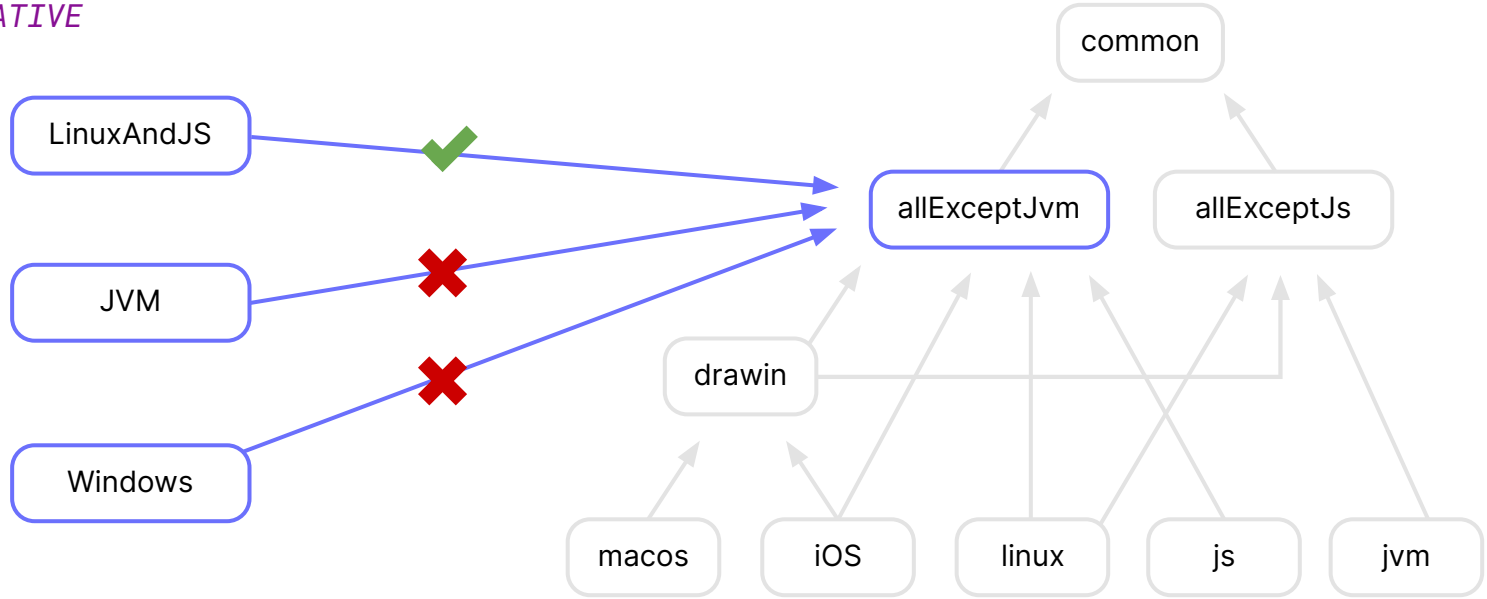


What are we even supposed to do?

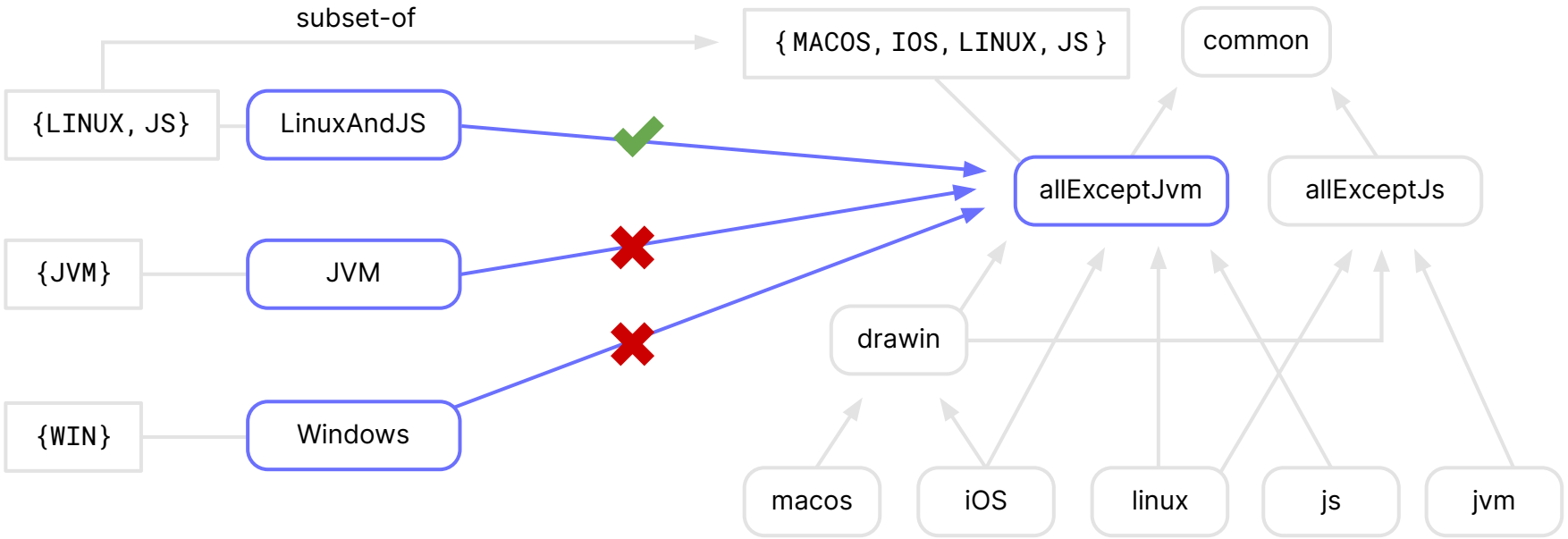


What are we even supposed to do?

```
data class Platform(val components: Set<SimplePlatform>)

enum class SimplePlatform {
    JS, JVM, NATIVE
}
```



What are we even supposed to do?



Experiments section: dependencies

A peek into Gradle Module Metadata

1. build.gradle

```
plugins {
    id 'maven-publish'
}

group = '<group.id>'

version = '<my-version>'
```

settings.gradle

```
rootProject.name = '<project-name>'
```

2. `$ ./gradlew publishToMavenLocal`

3. `$ cd ~/.m2/repository/<group>/<id>/<project-name>`

Example: Gradle Module Metadata in MPP

```
➜  mpp pwd
/Users/dmitry.savvinov/.m2/repository/hello/mpp
➜  mpp tree -L 1
├── my-mpp-lib
│   ├── 1.0-SNAPSHOT
│   │   ├── maven-metadata-local.xml
│   │   ├── my-mpp-lib-1.0-SNAPSHOT.module
│   │   └── my-mpp-lib-1.0-SNAPSHOT.pom
│   └── maven-metadata-local.xml
├── my-mpp-lib-js
├── my-mpp-lib-jvm
├── my-mpp-lib-macosx64
└── my-mpp-lib-metadata
```

Gradle module metadata

Example: Gradle Module Metadata in MPP

```
"variants": [
  {
    "name": "js-api",
    "attributes": {
      "org.gradle.usage": "kotlin-api",
      "org.jetbrains.kotlin.js.compiler": "legacy",
      "org.jetbrains.kotlin.platform.type": "js"
    },
    "available-at": {
      "url": "../../my-mpp-lib-js/1.0-SNAPSHOT/
              my-mpp-lib-js-1.0-SNAPSHOT.module",
      "group": "hello.mpp",
      "module": "my-mpp-lib-js",
      "version": "1.0-SNAPSHOT"
    }
```

"org.jetbrains.kotlin.platform.type": "js" — "Hey, I have a JS-part!"

"available-at": { — "Here's where you should look for it"

Native dependencies

Using libraries in shared native code

1.3

nativeMain/main.kt

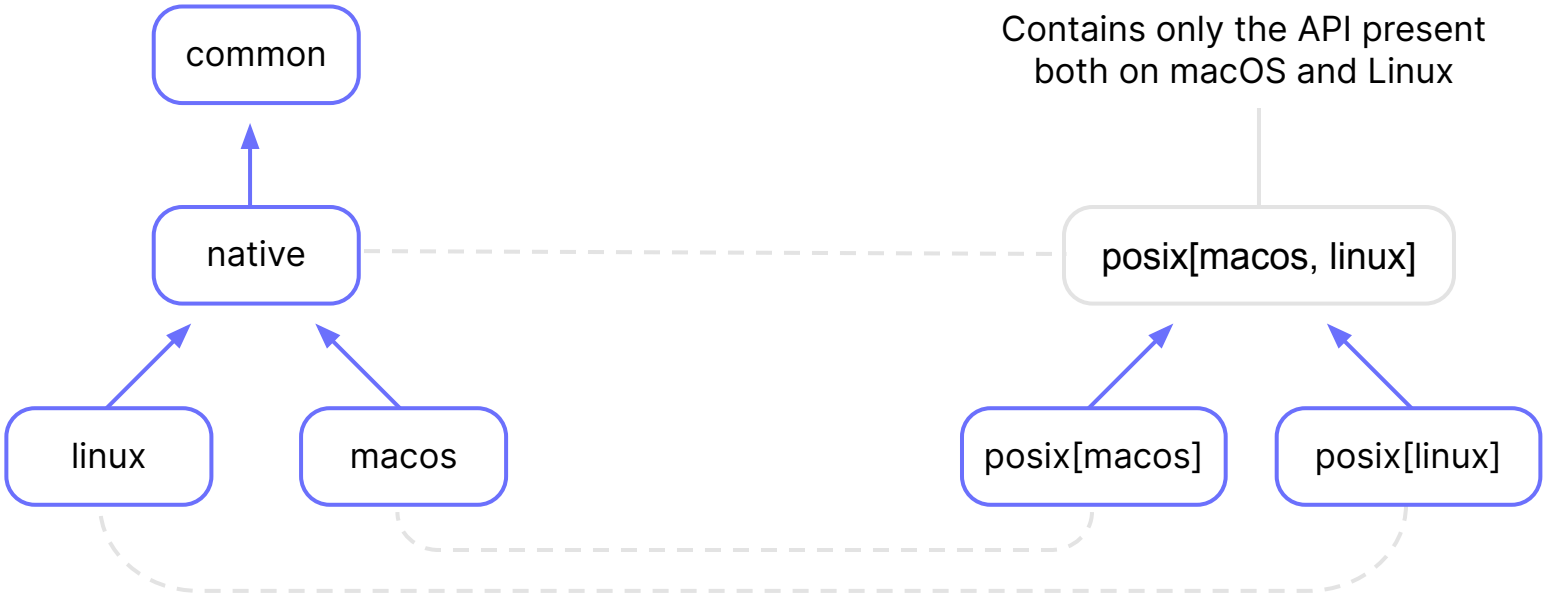
```
import platform.posix.pthread_create

fun main() {
   pthread_create(...)
}
```

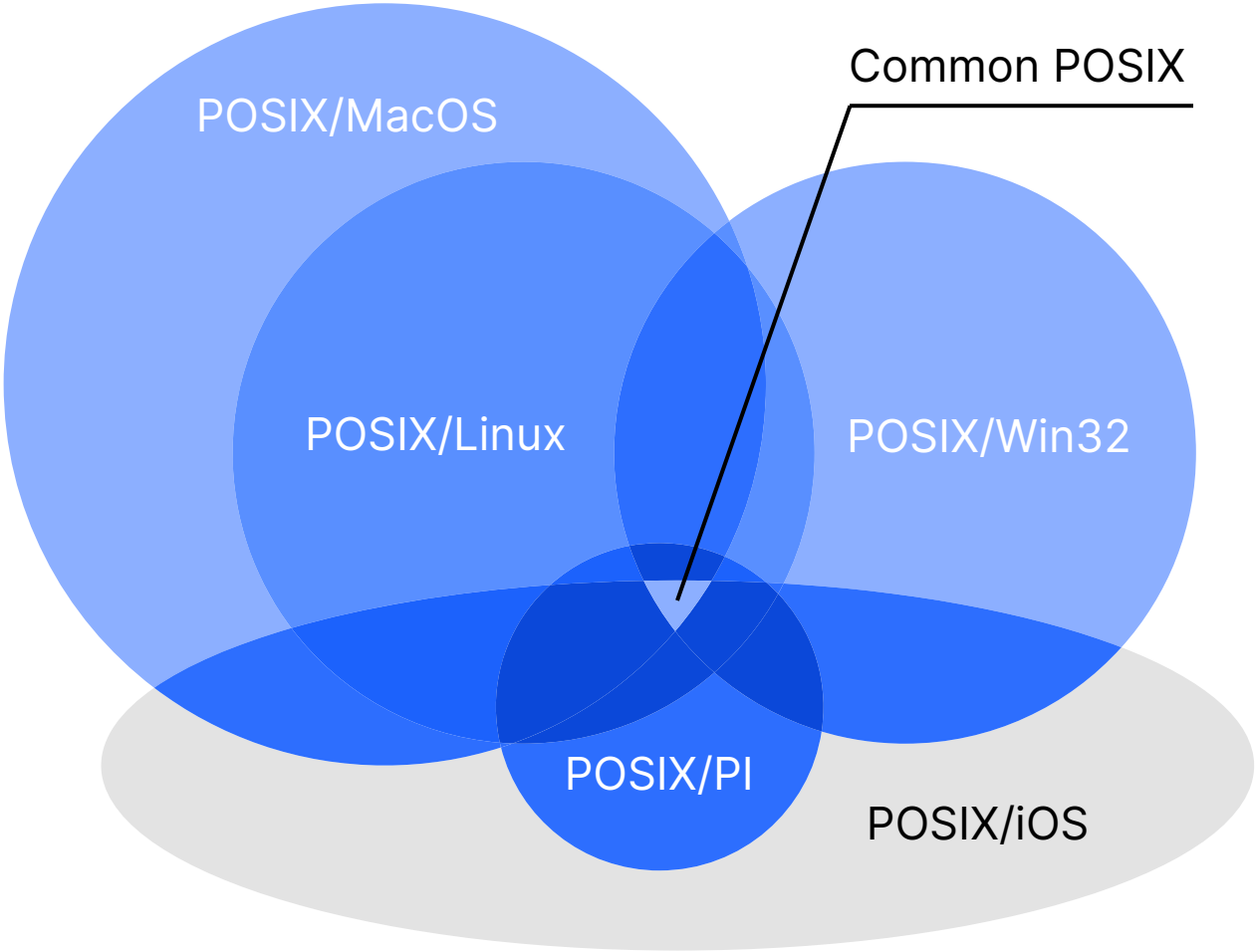
1.4

nativeMain/main.kt

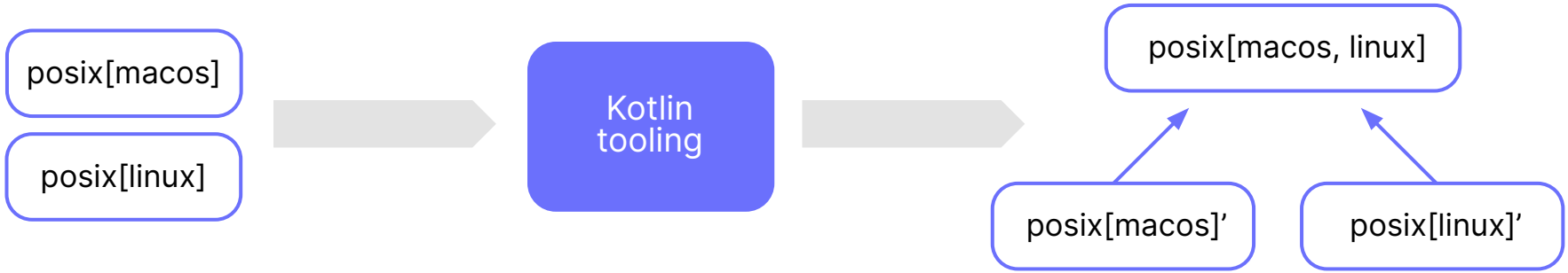
```
import platform.posix.pthread_create

fun main() {
   pthread_create(...)
}
```

Native dependencies are tricky




Common POSIX
POSIX/MacOS
POSIX/Linux
POSIX/Win32
POSIX/PI
POSIX/iOS

Do it automagically



Experiments section: the commonizer

How can I play with the commonizer?

1. Settings.gradle

```
kotlin.native.enableDependencyPropagation=false
```

2. Create a project with native-shared source-set

build.gradle

```
kotlin {
    iosX64()
    iosArm64()
    sourceSets {
        nativeMain.dependsOn(commonMain)

        iosArm64Main.dependsOn(nativeMain)
        iosX64Main.dependsOn(nativeMain)
    }
}
```

3. Either import the project in IntelliJ IDEA or run ./gradlew runCommonizer

How can I play with the commonizer?

4. You'll see something like this:

```
> Task :runCommonizer

Kotlin KLIB commonizer: Please wait while preparing libraries.
[Step 1 of 1] Preparing commonized Kotlin/Native libraries for
targets [ios_arm64], [ios_x64] (258 items)
  * Read lazy (uninitialized) libraries in 144ms
...
```

5. Wait a bit (it might take a few minutes!)

6. Use the IDE

How can I play with the commonizer?

- Kotlin/Native 1.4.0 - posix [ios_arm64(*), ios_x64]
- Kotlin/Native 1.4.0 - posix [ios_arm64, ios_x64(*)]
- Kotlin/Native 1.4.0 - posix [ios_arm64, ios_x64]
- Kotlin/Native 1.4.0 - posix [ios_arm64]
- Kotlin/Native 1.4.0 - posix [ios_x64]

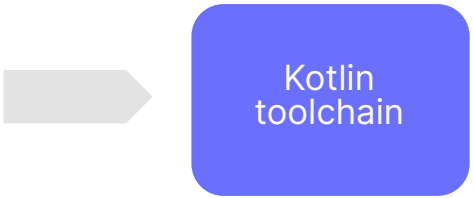
How can I play with the commonizer?

platform1

```
class Foo {
    fun common() { }
    fun platform1() { }
}

val commonVal: Int = 42
```

platform2

```
class Foo {
    fun common() { }
    fun platform2() { }
}

val commonVal: Int = 42
```



common[platform1, platform2]

```
expect class Foo {
    fun common()
}

val commonVal: Int = 42
```

platform1’

```
actual class Foo {
    actual fun common() { }
    fun platform1() { }
}
```

platform2’

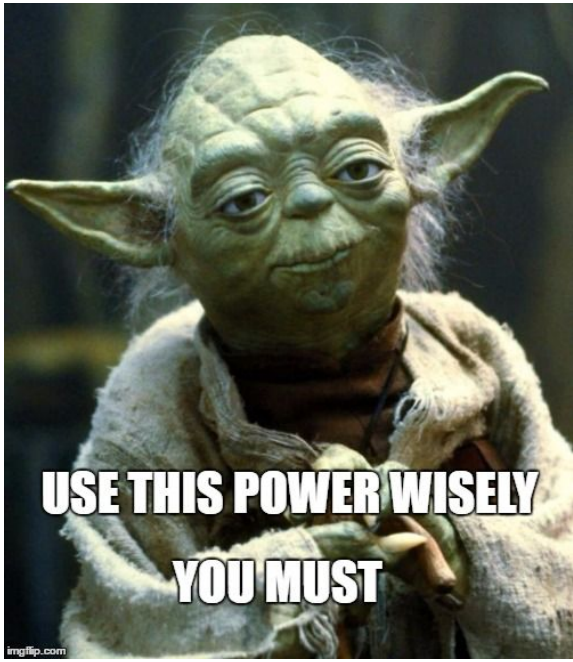
```
actual class Foo {
    actual fun common() { }
    fun platform2() { }
}
```

Why doesn’t the commonizer produce only common code?

Outro

What did we learn today?

- 1.4: publish intermediate source-sets (needs explicit opt-in)
- 1.4: .jar with .kotlin_metadata → .klib with IR
- .klib is a zip file with a strictly defined layout
- .kotlin_metadata is a set of serialized Kotlin declarations
- IR is a ~serialized AST
- Gradle Metadata helps to declare dependencies once
- The Kotlin Platform has a notion of “compatibility” and models it in Gradle Attributes
- The Kotlin toolchain builds an API intersection automagically to enable shared native libraries (needs an explicit opt-in)



Thanks!
Have a nice Kotlin

@dsavvinov