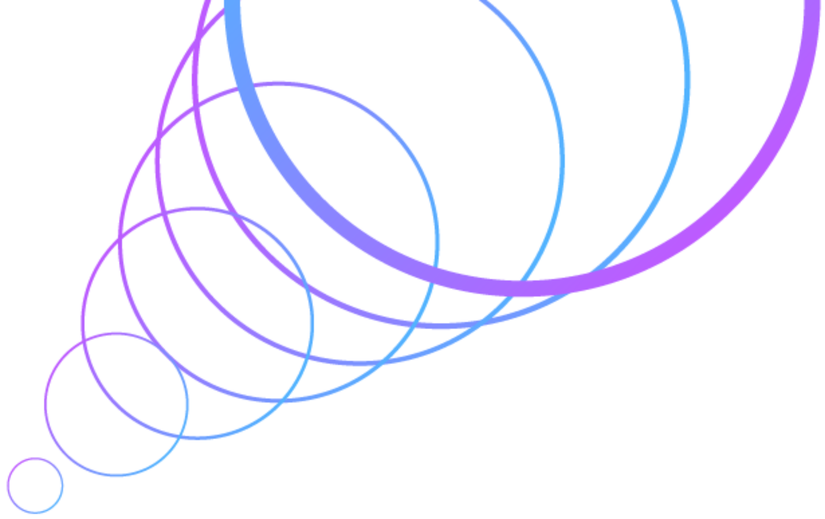
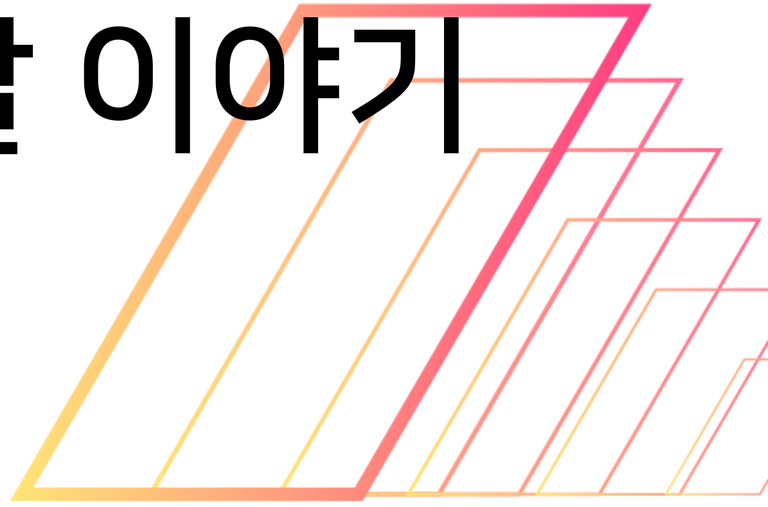




DEVIEW 2021

Kotlin Multiplatform Mobile을 활용한 데마에칸 드라이버앱 개발 이야기

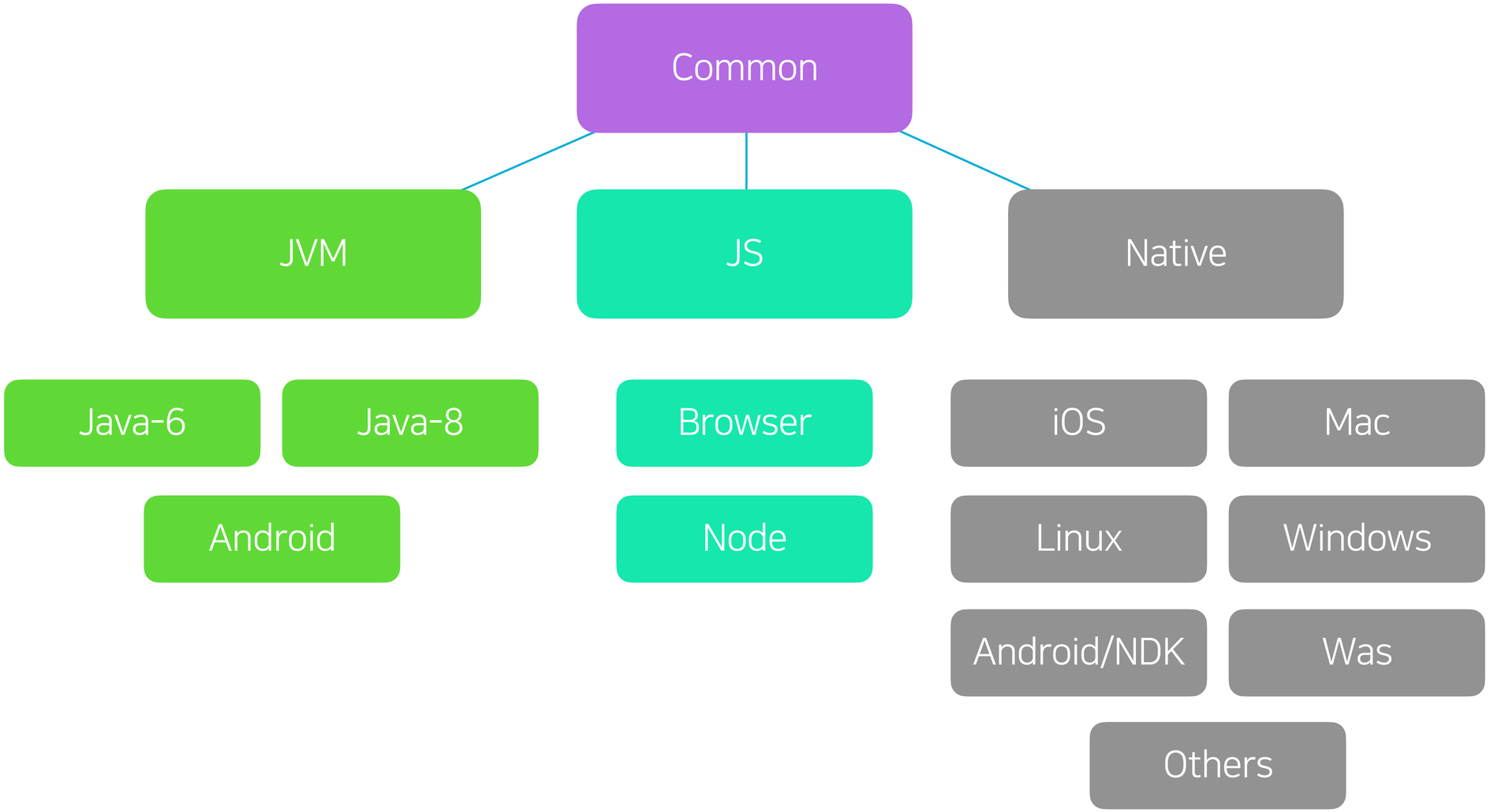
김광현 LINE PLUS

CONTENTS

1. KMM 소개

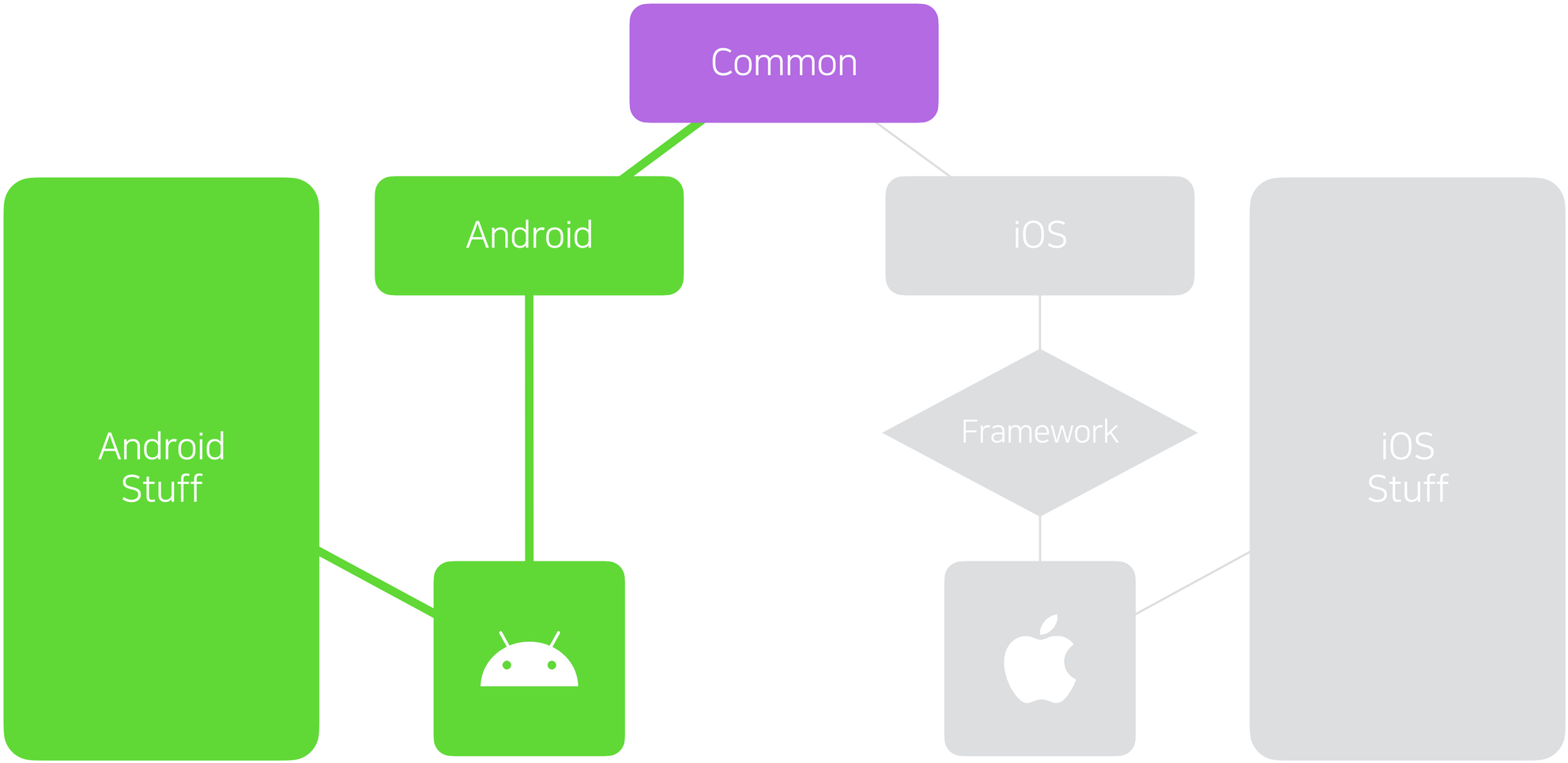
Kotlin Multiplatform

JVM, JS, Native 등 다양한 플랫폼에서 동일한 비지니스로직을 사용할 수 있도록 도와줌

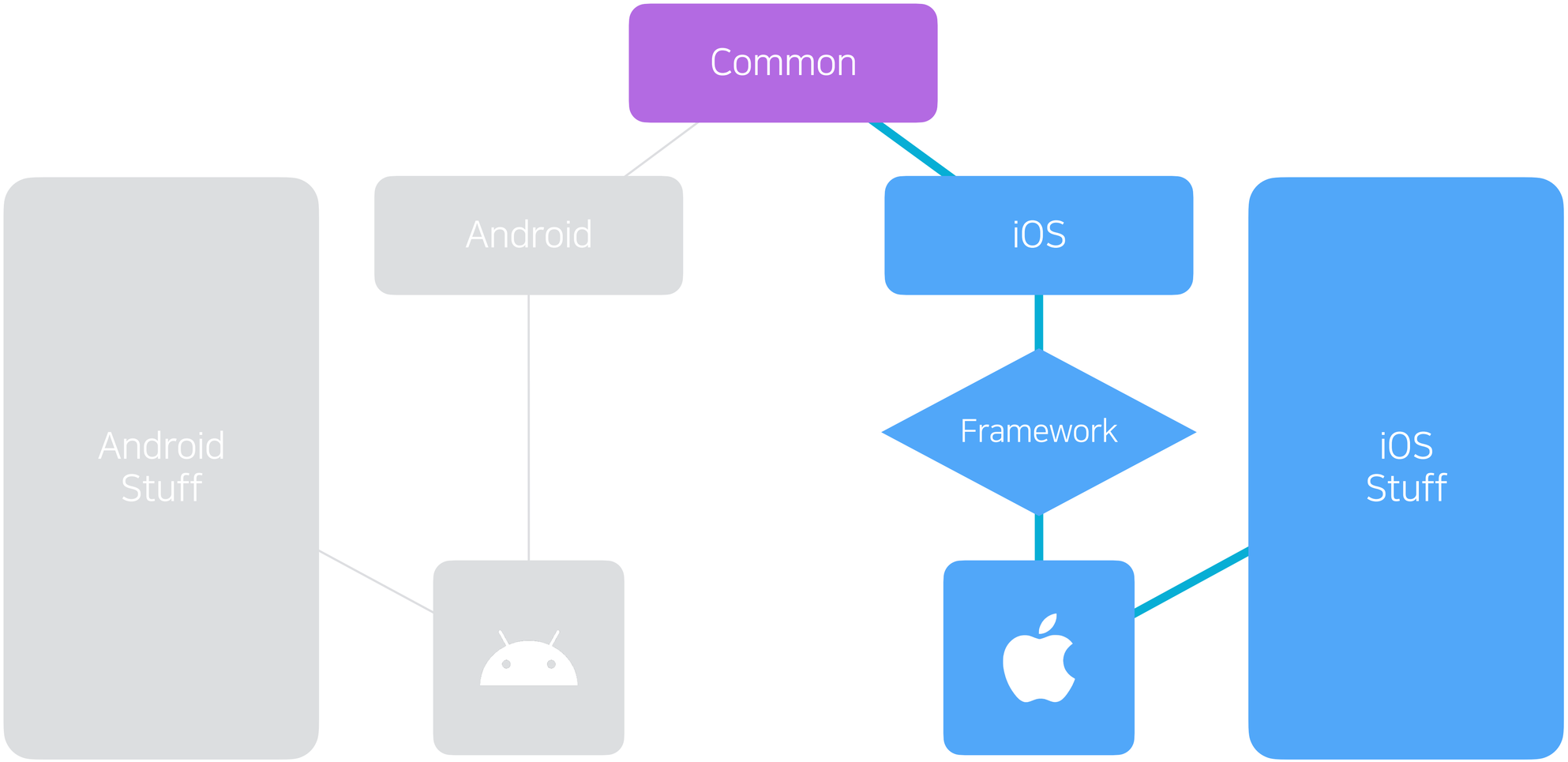


Kotlin Multiplatform Mobile

DEVIEW 2021

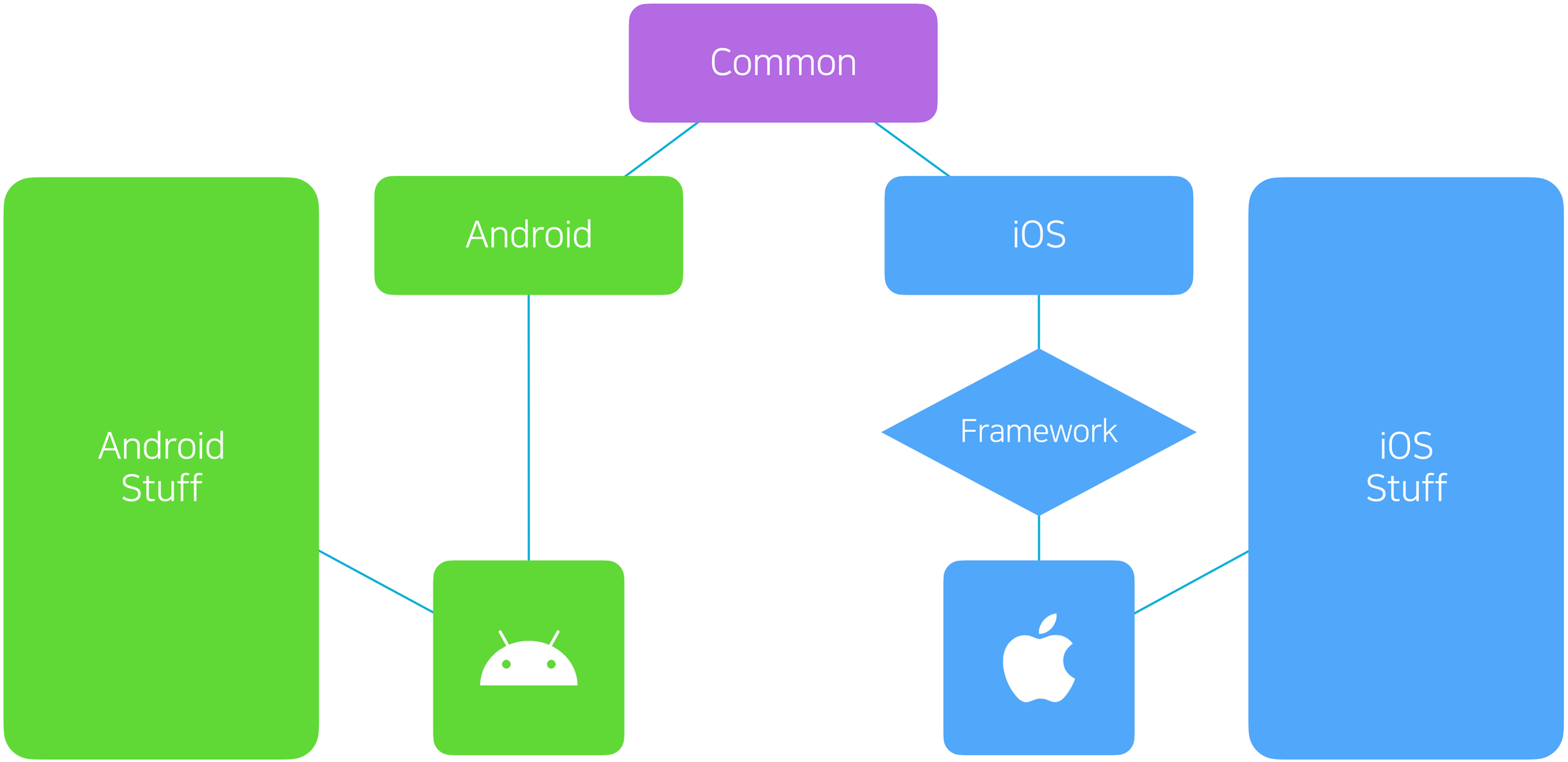


Kotlin Multiplatform Mobile

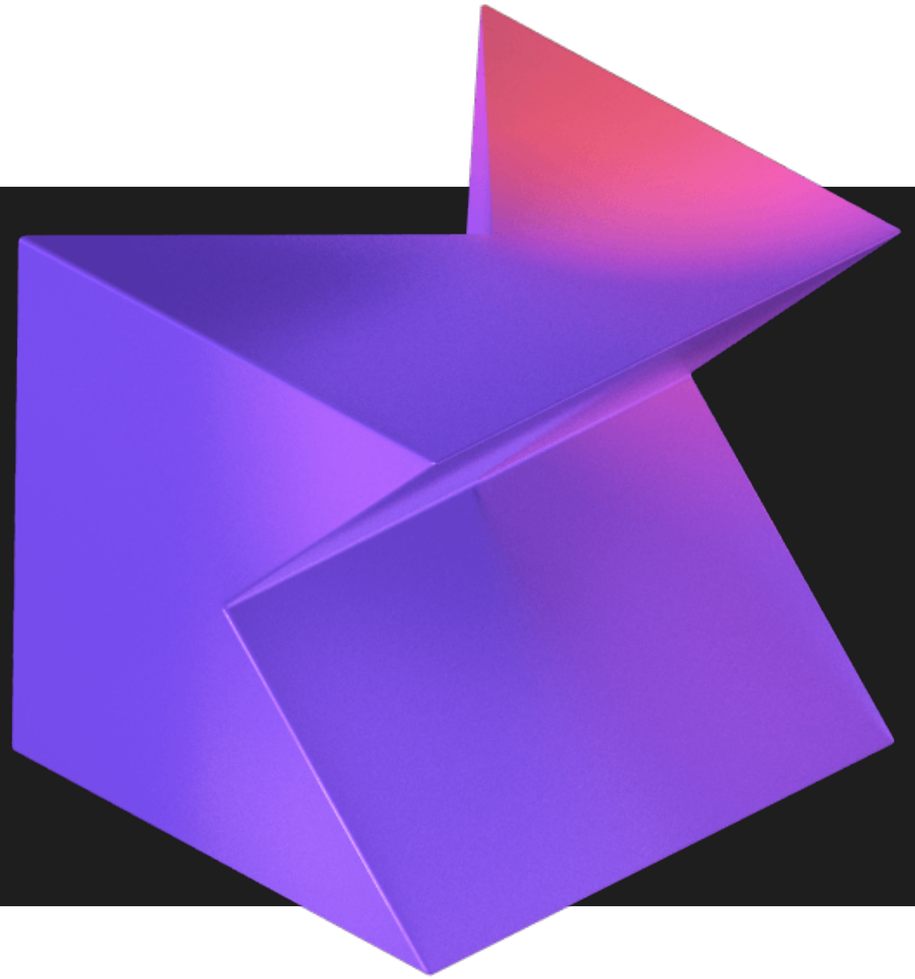


Kotlin Multiplatform Mobile

Android와 iOS용 모바일 애플리케이션 개발을 지원하기 위한 SDK

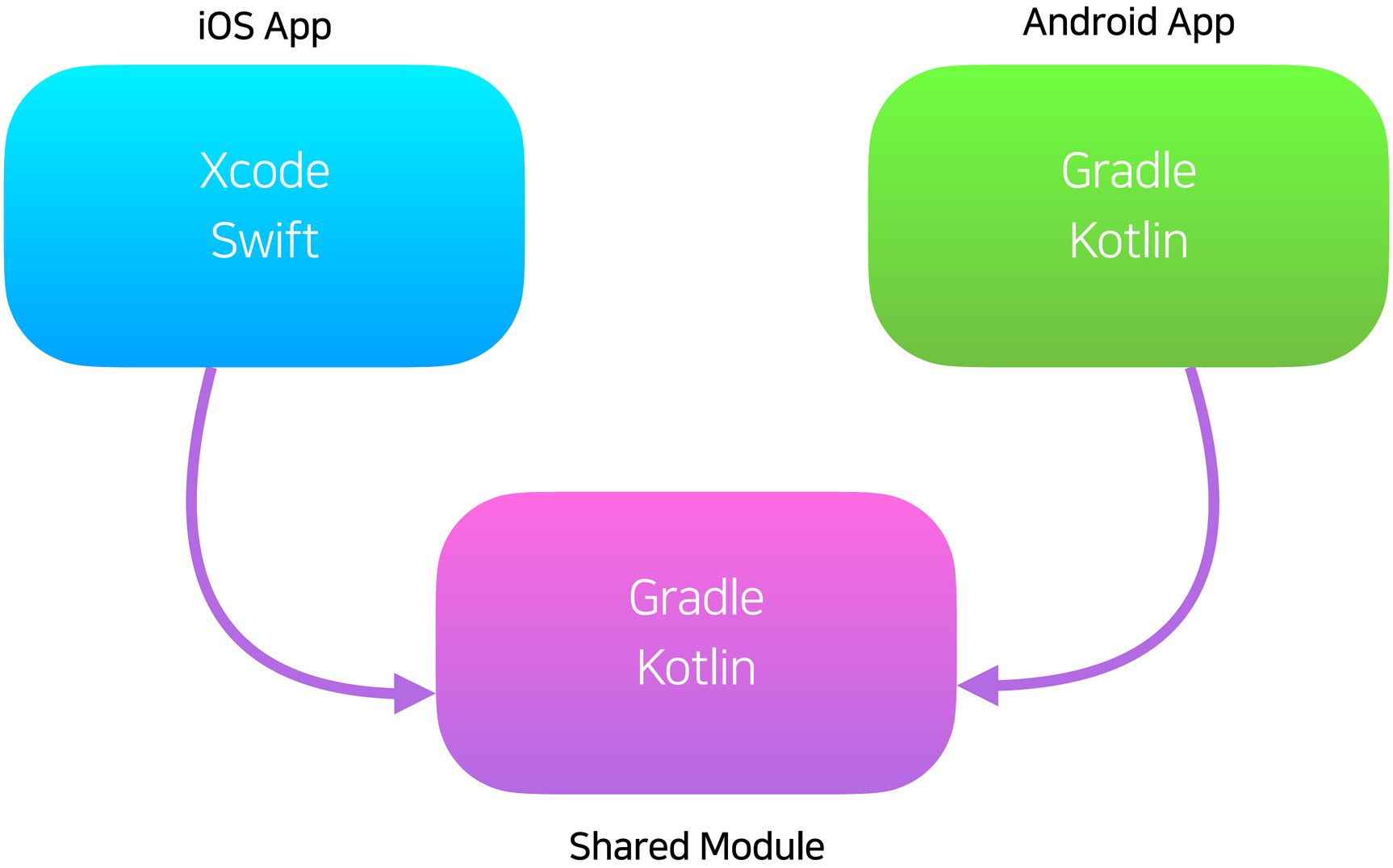


Kotlin으로 iOS와 Android 앱의 비즈니스 로직을 한 번만 작성하세요.



https://kotlinlang.org/lp/mobile/

KMM 앱 구조와 에코시스템



플랫폼별 APIs 연결

Kotlin 기본 기능 외 플랫폼 특화 기능을 구현

Business logic and core

expect fun platformStuff()

iOS specific APIs

actual fun platformStuff() {
- AVFoundation
- Core ML
- LocalAuthentification
- Accounts
- CloudKit
- and lots of community libraries

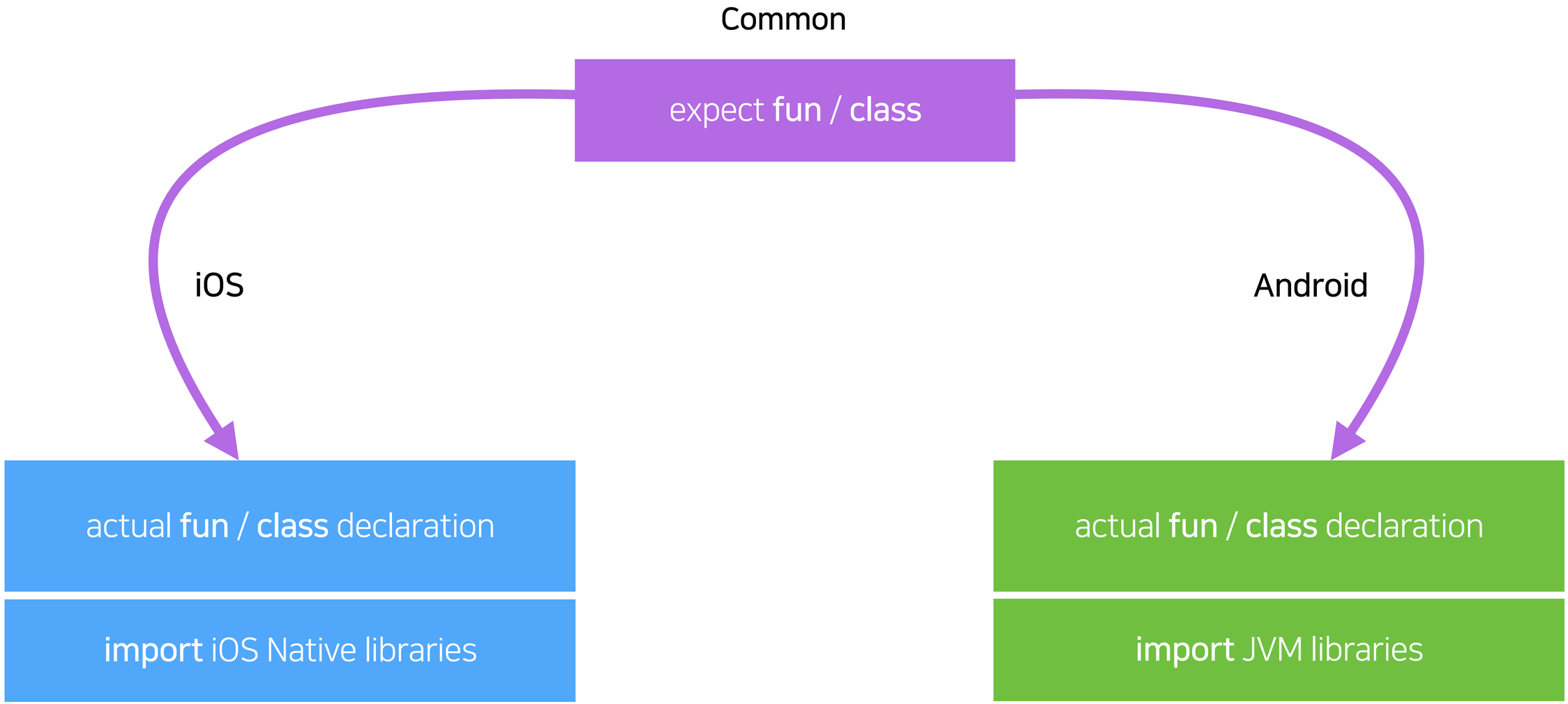
}

Android specific APIs

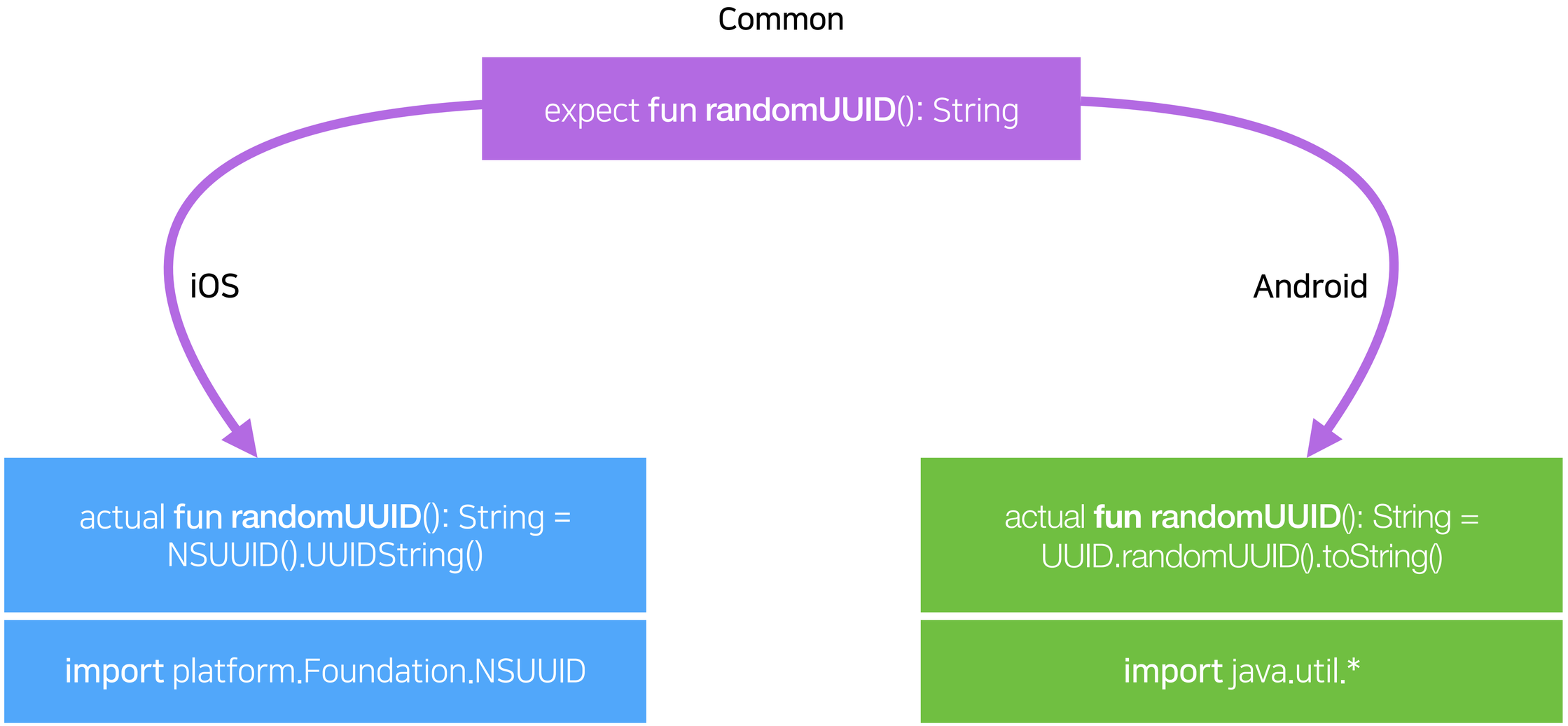
actual fun platformStuff() {
- CameraX
- MLKit
- Biometric
- Room
- Play Services
- and lots of community libraries

}

expect and actual keyword



Example


Common
expect fun randomUUID(): String
iOS
Android
actual fun randomUUID(): String =
NSUUID().UUIDString()
import platform.Foundation.NSUUID
actual fun randomUUID(): String =
UUID.randomUUID().toString()
import java.util.*

Code level

DEVIEW 2021

Common

```
expect fun randomUUID(): String
```

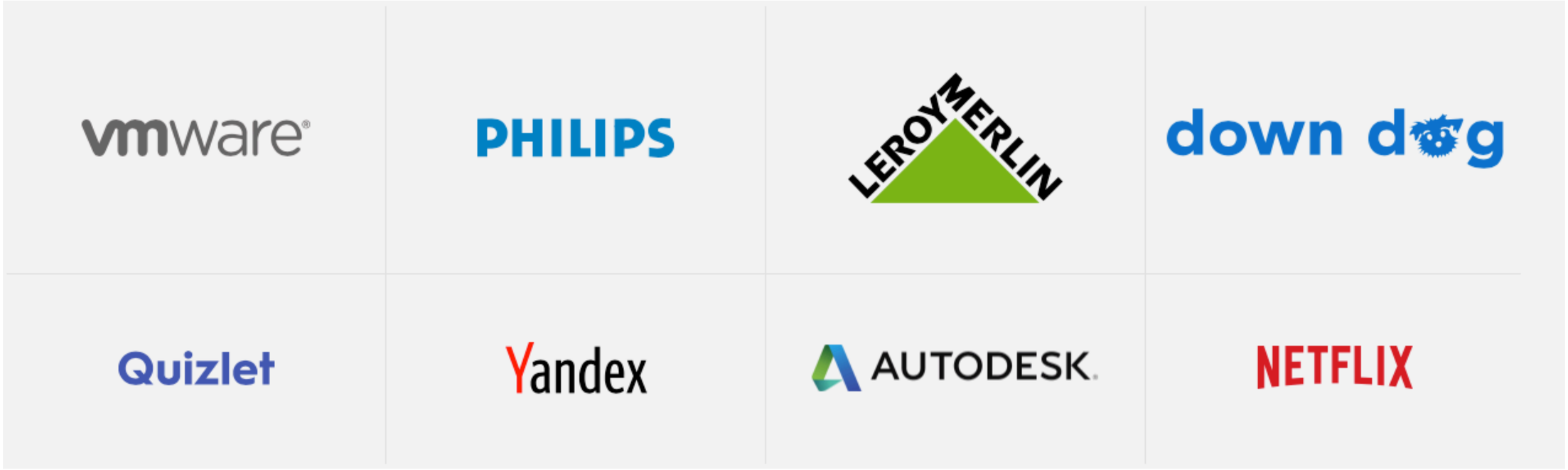
iOS

```
actual fun randomUUID() = NSUUID().UUIDString()
```

Android

```
actual fun randomUUID() = UUID.randomUUID().toString()
```

누가 KMM을 사용하고 있는가



누가 KMM을 사용하고 있는가

DEVIEW 2021

vmware
PHILIPS
LEROY MERLIN
down dog
Quizlet
Yandex
AUTODESK
NETFLIX

LINE
NAVER
Demaecan

2.데마에칸 드라이버앱 소개

데마에칸?

- 한국에 배민과 쿠팡이츠가 있다면 일본에는 데마에칸과 우버이츠가
- 아직 배달 시장이 한국 만큼 성장하진 않았지만 지속적 성장
- 2020년 네이버 라인, 일본판 배달의 민족 '데마에칸' 인수
- 네이버.라인 개발 조직에서 기술적 선진화에 적극적으로 참여 중

LINE

우리가 KMM을 선택한 이유

- 프로젝트 로드맵으로 본 기존 시스템의 한계
- 네이티브 개발자로 구성된 우리팀
- 크로스플랫폼의 장점을 차용하고 싶은 니즈
- 네이티브 플랫폼만의 장점
- 개발 범위가 작아 프로덕션 레벨에 새로운 기술을 도입하기에 수월할 것이라 판단

프로젝트를 통해 얻은 것

- 동일한 비지니스 로직 사용을 통한 생산성 향상 및 버그 발생율 감소
- 동일한 Task 관리로 프로젝트 관리 효율 증대
- 단일 코드베이스로 모든 팀원들의 서비스 이해도 증대
- Gradle 스크립트 등 공유 자원을 통한 생산성 향상
- 보다 긴밀한 협업으로 우리는 하나 👨👩👧
- 강력한 책임감 💁 (공용 모듈 개발 담당)

프로젝트에서 어려웠던 점

- iOS 개발자의 Kotlin 학습 및 트러블 슈팅
- 공용 모듈 개발에 병목 발생
  - 코드베이스 전반에 대한 이해
  - 변경으로 인한 사이드 이펙트
  - 양 플랫폼 테스트의 부담
- 코드 리뷰 병목
- 공용 비지니스 로직에 대한 인식 동기화

하지만

모두 극복할 만한 어려움

= 성장 

2021년 7월 데마에칸 드라이버앱 론칭

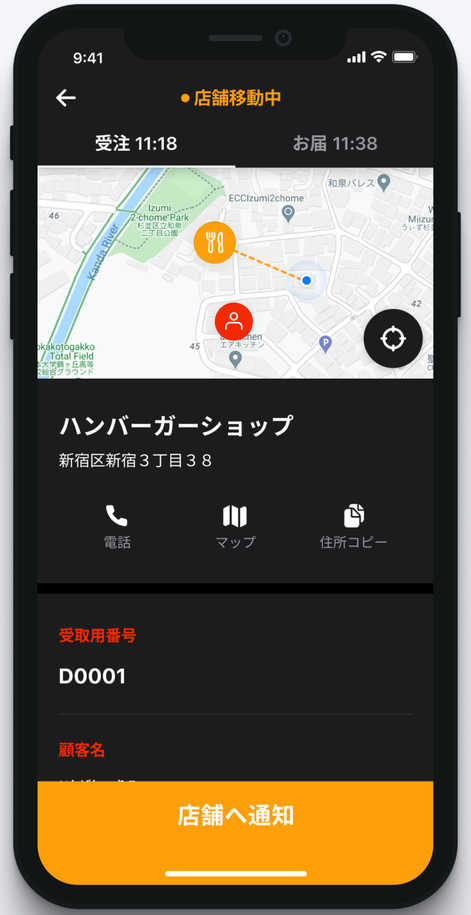
DEVIEW 2021

希望する配送手段を自由に選択

配達する注文を一覧で確認

空いた時間に1件だけ配達も可能

お店とお届け先の位置をマップで表示



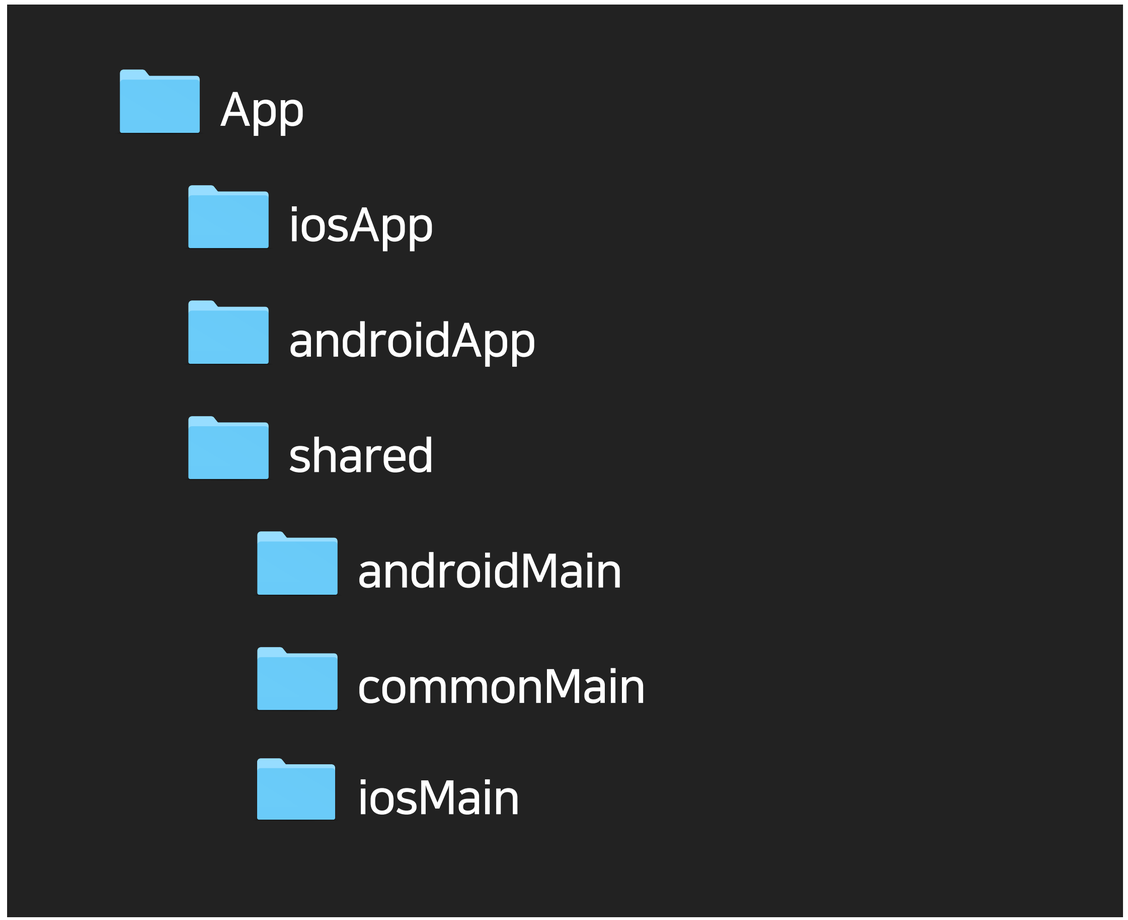
3. KMM 프로젝트 설계 공유

프로젝트 구조와 Git 레파지토리 구성

Project Structure

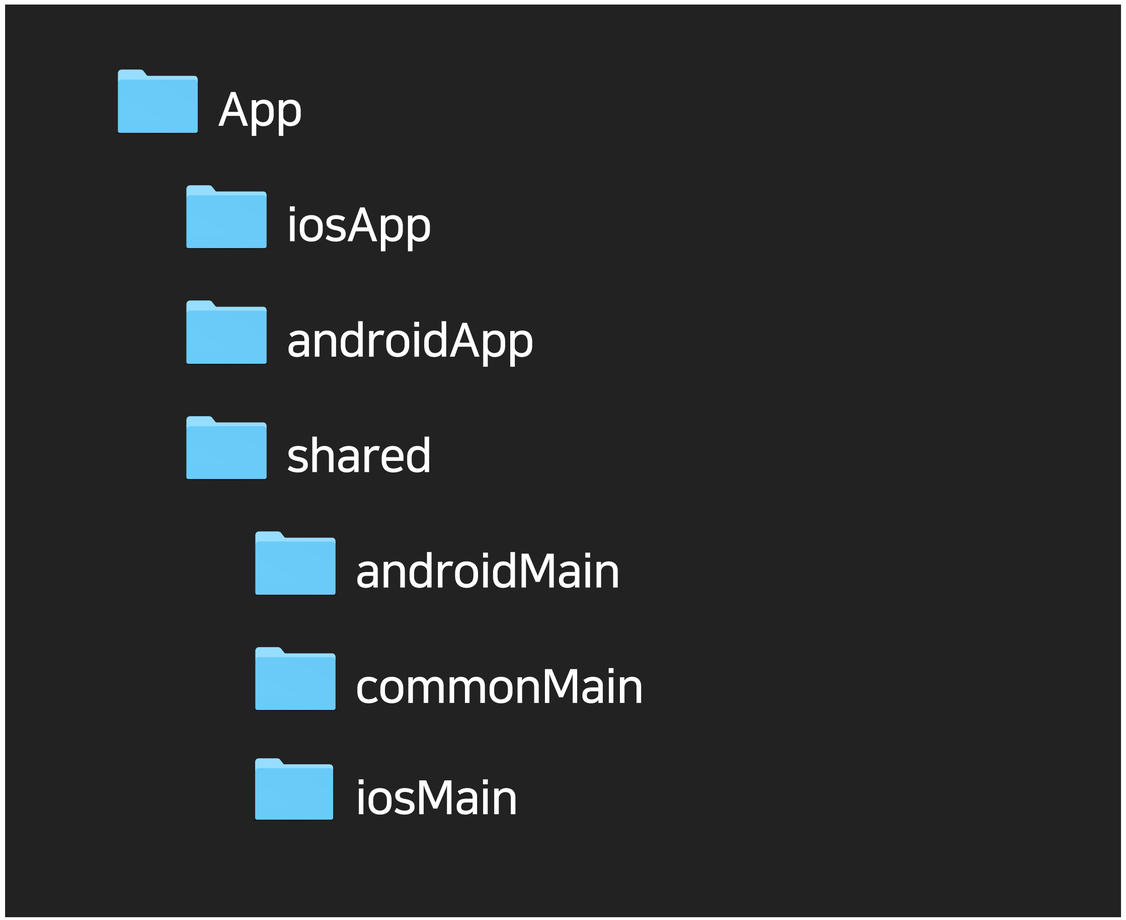
DEVIEW 2021

General

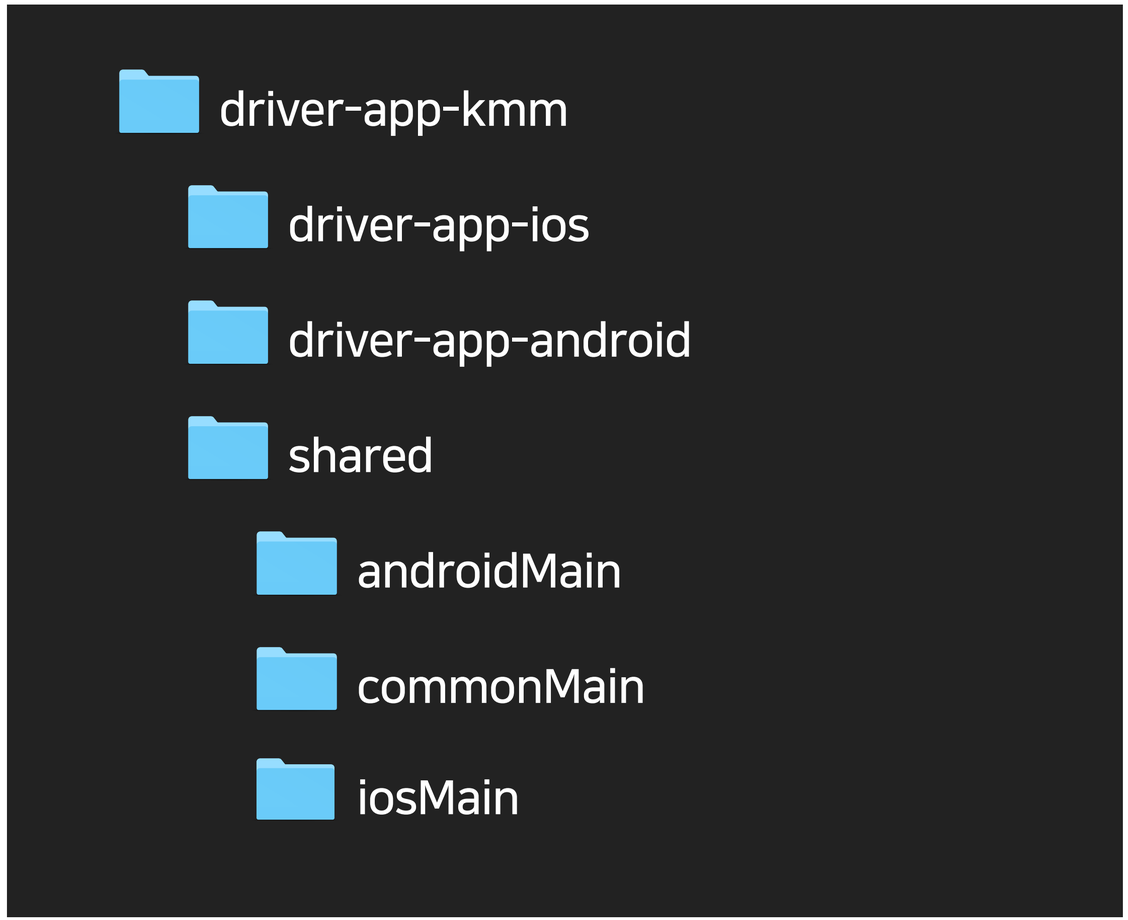


Project Structure

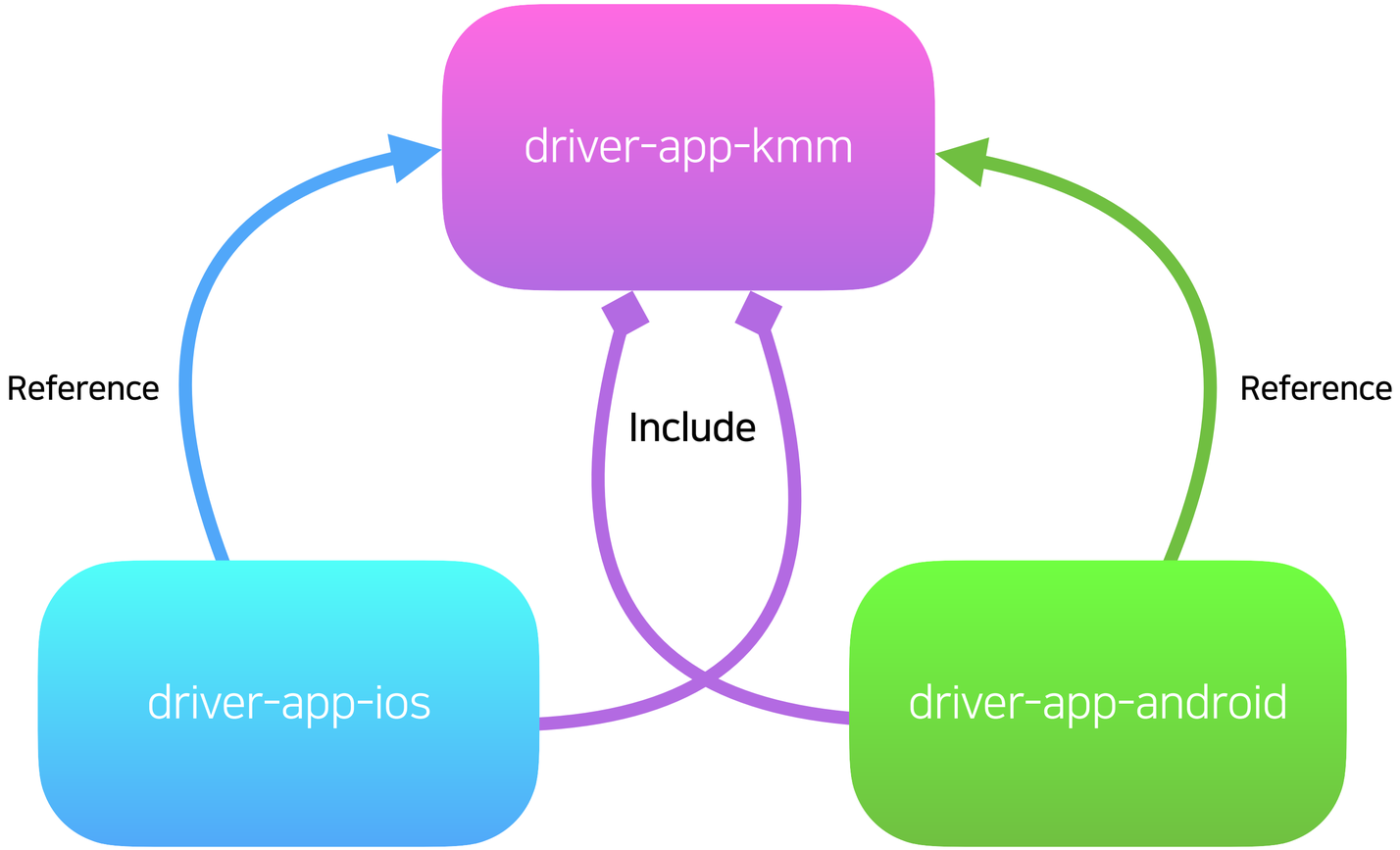
General

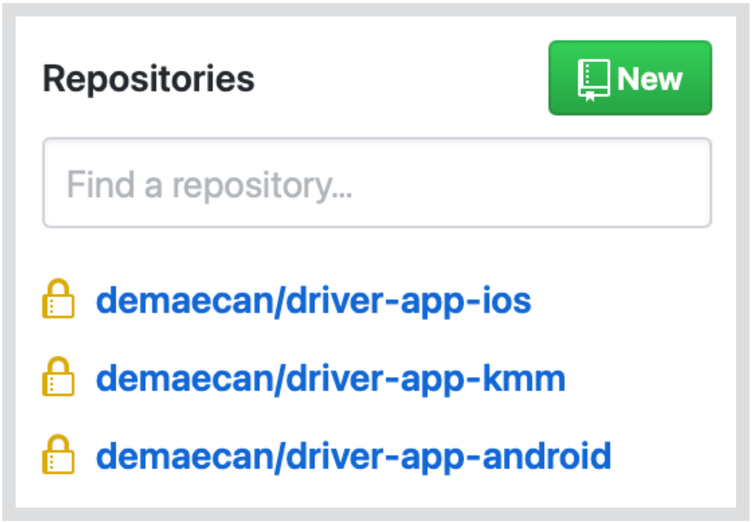


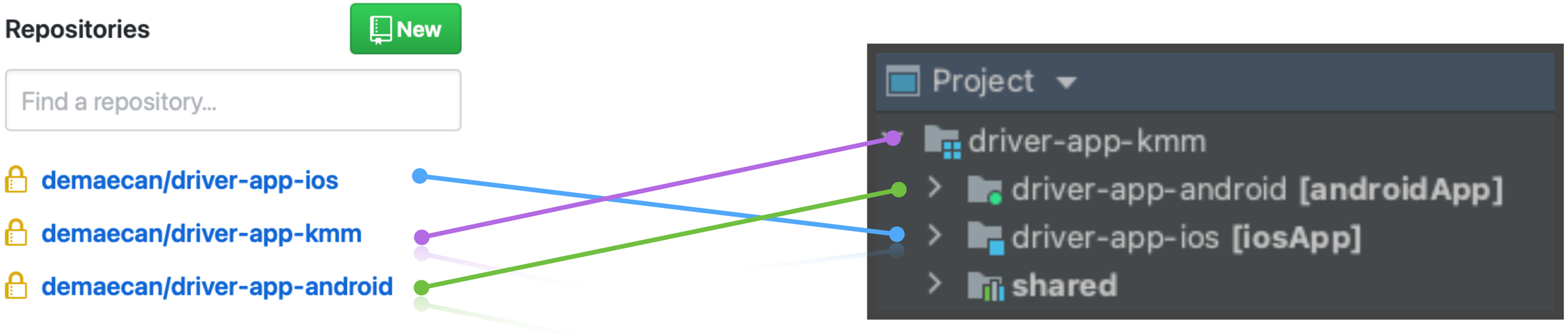
Driver App



Git Repository







settings.gradle.kts

```
rootProject.name = "driver-app-kmm"

include(":androidApp")
include(":iosApp")
include(":shared")

project(":iosApp").projectDir = file("../driver-app-ios")
project(":androidApp").projectDir = file("../driver-app-android")
```

무엇을 단일 코드베이스로 구성할 수 있을까

기능별 (Capability)

- 네트워킹
- 캐시 (Disk or Memory)
- 위치서비스
- 푸시 알림 (FCM & APNs)
- 애널리틱스 (Firebase & Others)

특징별 (Features)

- 유저 인증
- 디바이스토큰 관리
- API 통합
- Data 모델링
- 마이그레이션
- 앱 버전 관리
- 앱 초기화 및 라이프사이클에 따른 처리
- 로깅 및 에러 핸들링

😎 생각보다

많은 부분을 단일 코드베이스로
구성할 수 있다.

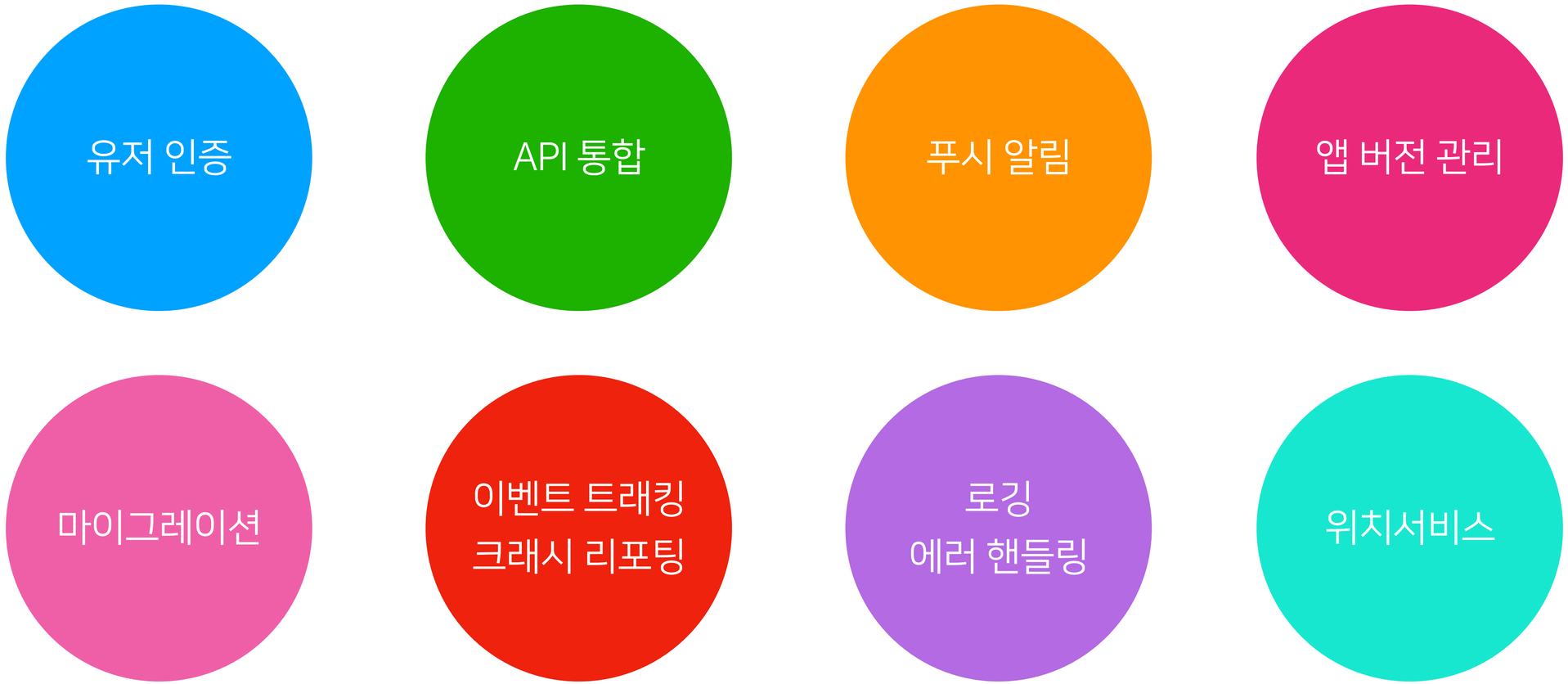
어떻게 단일 코드베이스를 구성할 것인가

단일 코드베이스 구성 목표

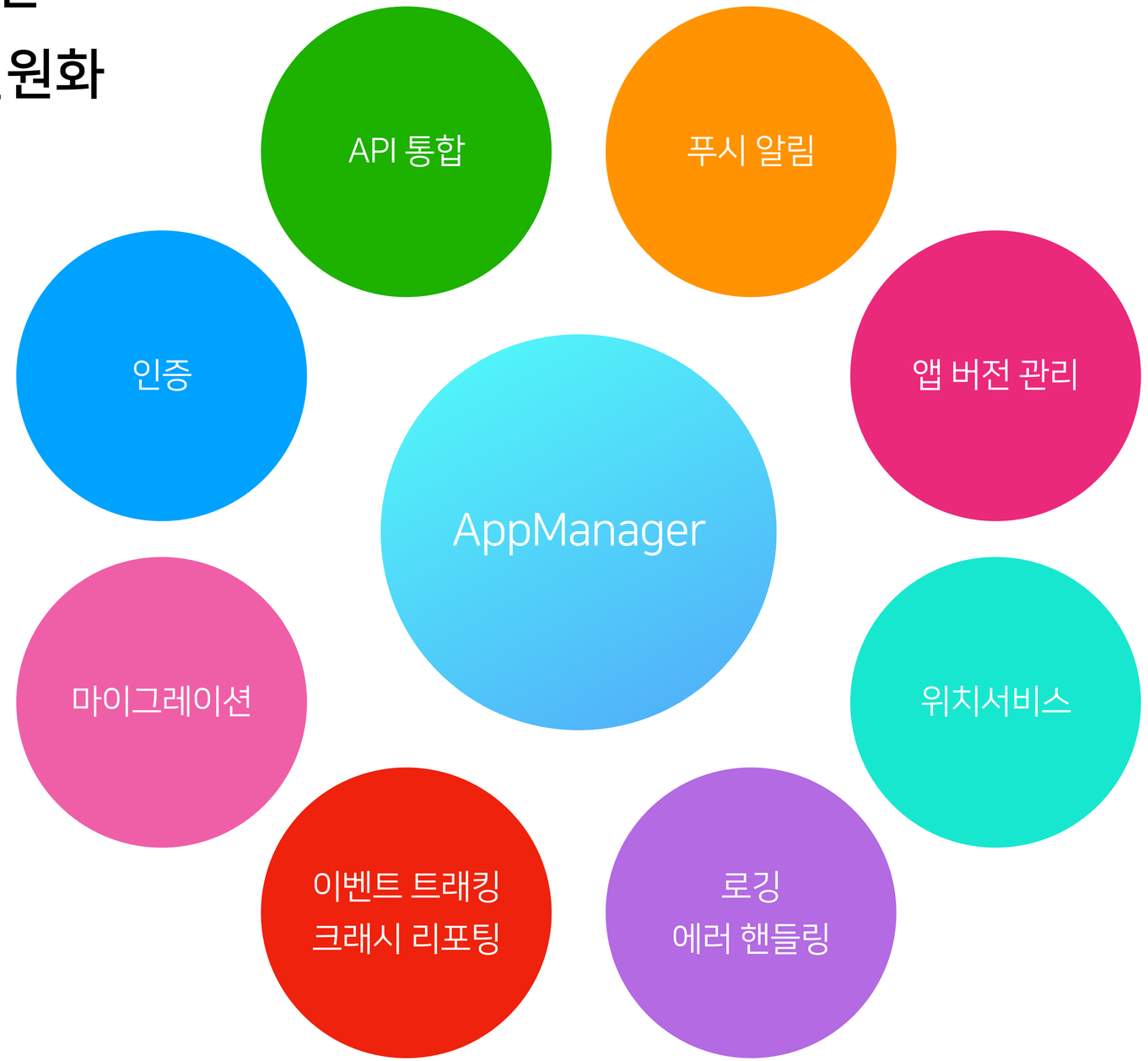
- 플랫폼 의존성 없는 기능의 공통화
- 앱 설정 및 관리의 일원화
- 비지니스 로직 공용화
- KMM 라이브러리를 통한 모듈화
- UI 구성 및 반응형 개발에 플랫폼 자유도 제공

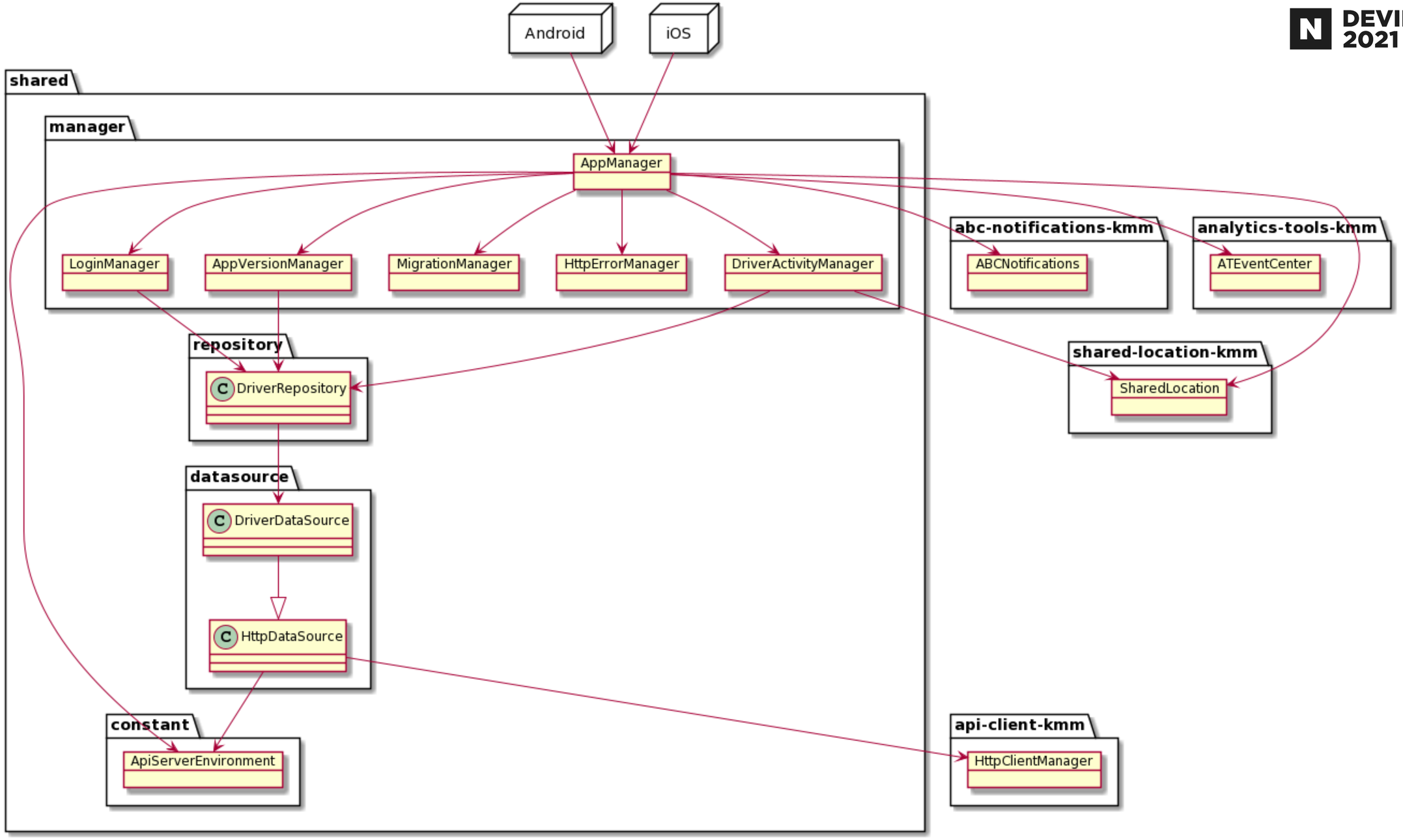
플랫폼 의존성 없는 기능의 공통화

보통의 앱이 갖춰야할 일반적인 기능들을 나열



AppManager를 통한
앱 설정 및 관리의 일원화




Android
iOS
shared
manager
AppManager
LoginManager
AppVersionManager
MigrationManager
HttpErrorManager
DriverActivityManager
abc-notifications-kmm
ABCNotifications
analytics-tools-kmm
ATEventCenter
repository
DriverRepository
shared-location-kmm
SharedLocation
datasource
DriverDataSource
HttpDataSource
constant
ApiServerEnvironment
api-client-kmm
HttpClientManager

AppManager (Kotlin DSL)

DEVIEW 2021

```
AppManager.initialize {
    defaultEnvironment()
    logging { }
    notifications { }
    loginManager { }
    location { }
    versionManager { }
    eventCenter { }
    driverActivityManager { }
    httpErrorManager { }
}
```

Shared

DEVIEW 2021

```
class AppManager {

   @ThreadLocal
   companion object {

      fun initialize(block: Companion.() -> Unit) {
         apply(block)

         loginManager {
            onLogin(this) {
               ABCNotifications.beginListening()
            }
            onLogout(this) {
               ABCNotifications.unregister()
               DriverActivityManager.endLocationDataUpdating()
            }
         }
         notifications {
            onNewToken {
               ExpoPushTokenManager.updatePushToken()
            }
         }

         AppVersionManager.checkVersion()
      }
   }
}
```

iOS

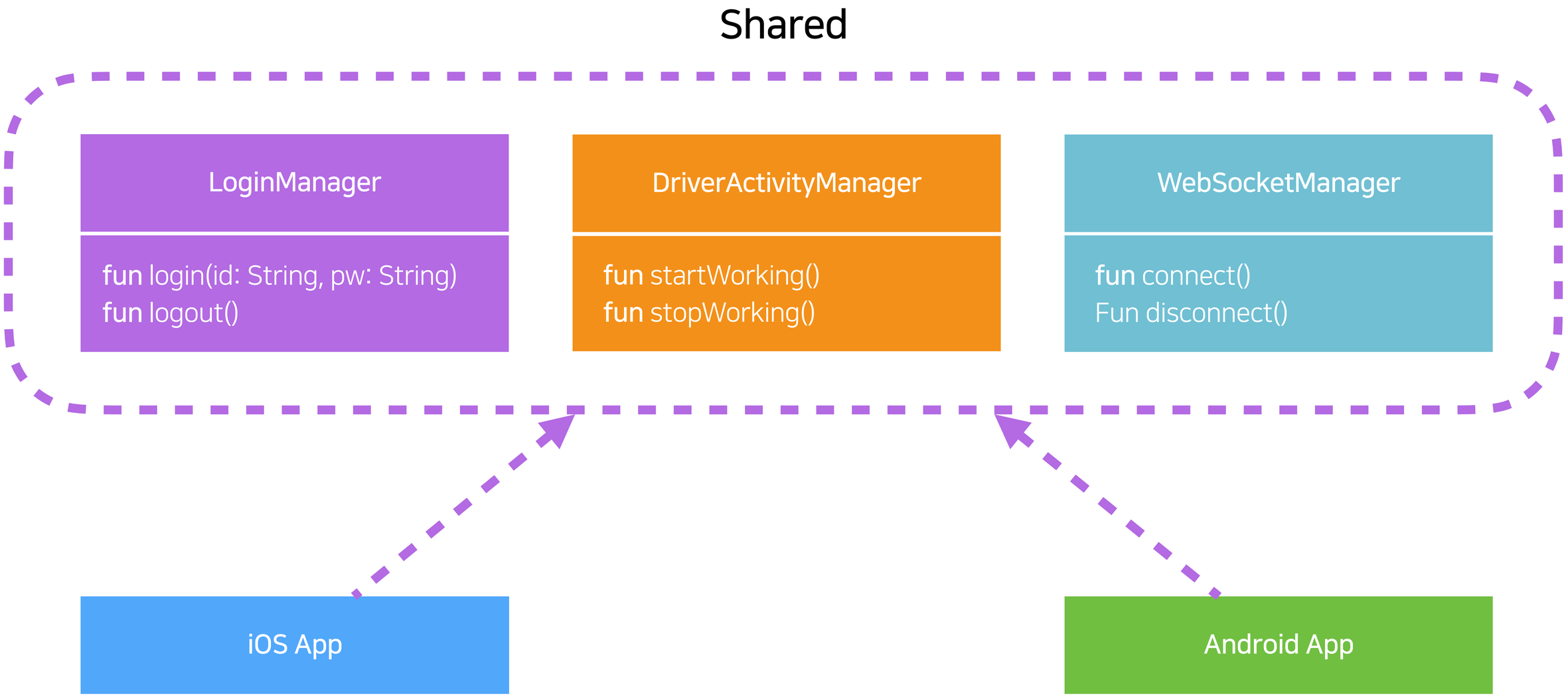
```
AppManager.default.initialize { [unowned self] in

  $0.defaultEnvironment(value: .real)

  $0.logging {
    $0.logger = LoggerCompanion().DEFAULT
    $0.level = .info
  }

  $0.loginManager {
    $0.onLogin(target: self) {
      router.trigger(.slidemenu)
    }
    $0.onLogout(target: self) {
      router.trigger(.login)
    }
  }

  $0.notifications {
    $0.onMessageReceived {
      PushNotificationProxy.default.process($0)
    }
  }
}
```

Android

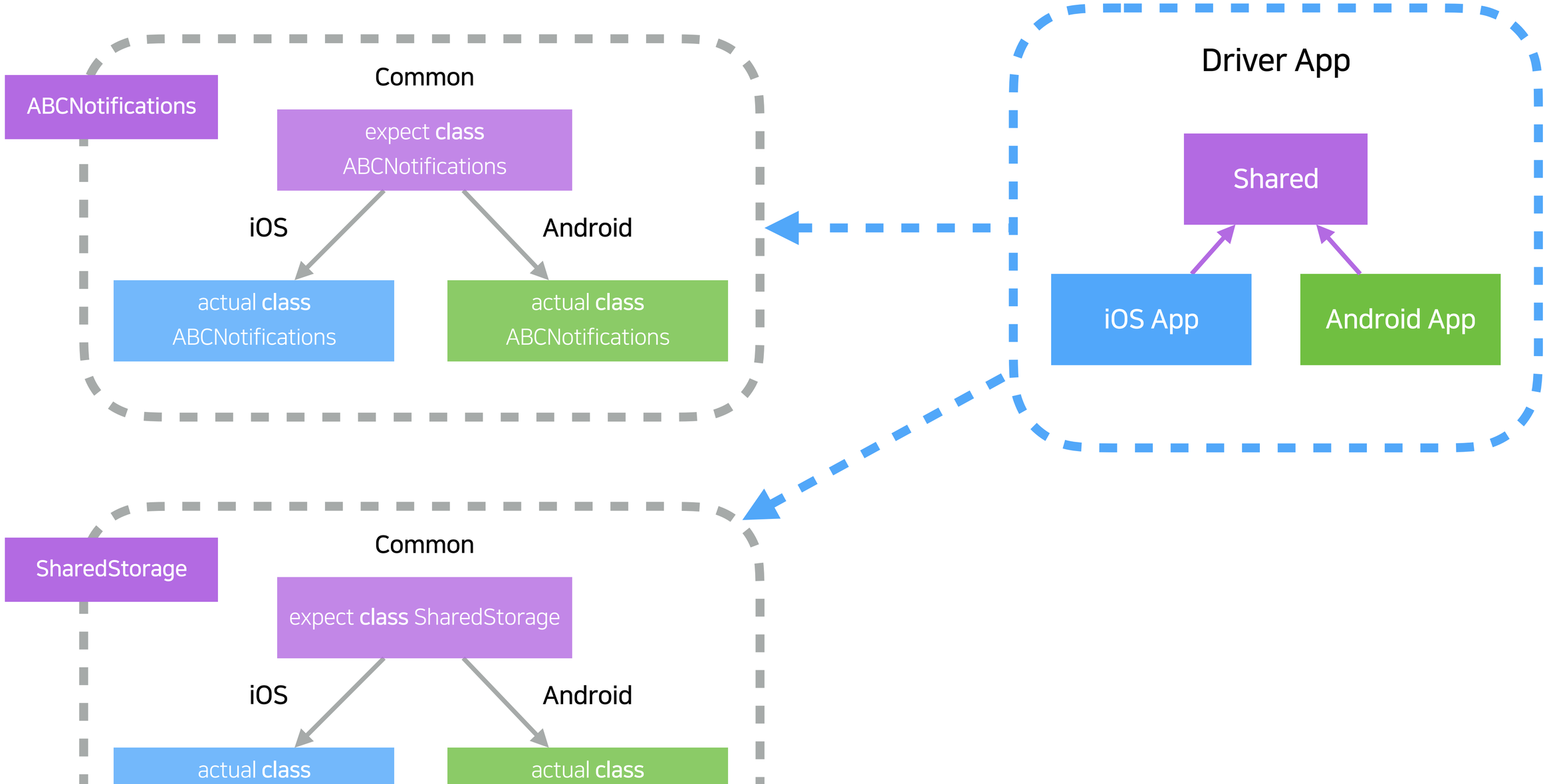
DEVIEW 2021

```
AppManager.initialize {

  defaultEnvironment(ApiServerEnvironment.REAL)

  logging {
    logger = Logger.DEFAULT
    level = LogLevel.INFO
  }

  loginManager {
    onLogin(TAG) {
      DeStateModel.session.refreshSession()
    }
    onLogout(TAG) {
      DeStateModel.session.refreshSession()
    }
  }

  notifications {
    onMessageReceived {
      PushNotificationProxy.process(it)
    }
  }
}
```

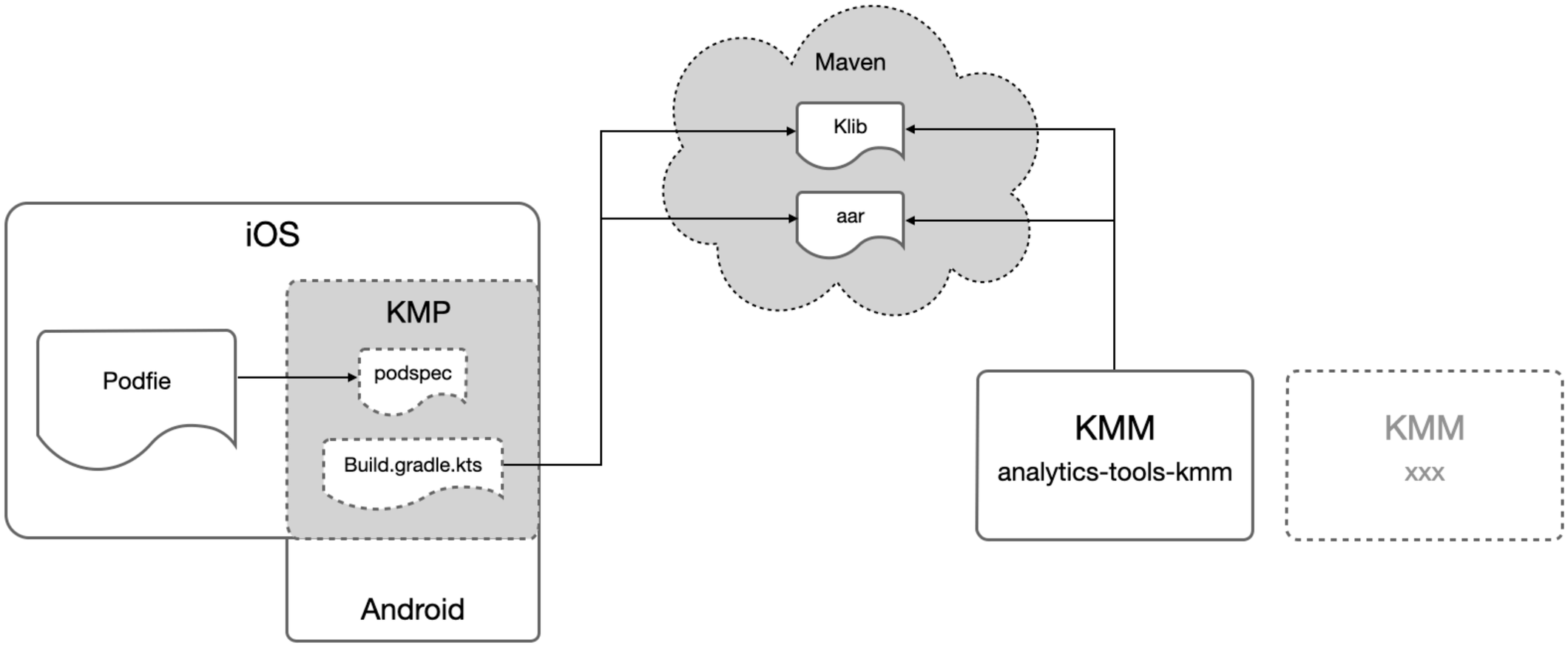
비즈니스 로직 공용화


Shared
LoginManager
fun login(id: String, pw: String)
fun logout()
DriverActivityManager
fun startWorking()
fun stopWorking()
WebSocketManager
fun connect()
Fun disconnect()
iOS App
Android App

KMM 라이브러리를 통한 모듈화



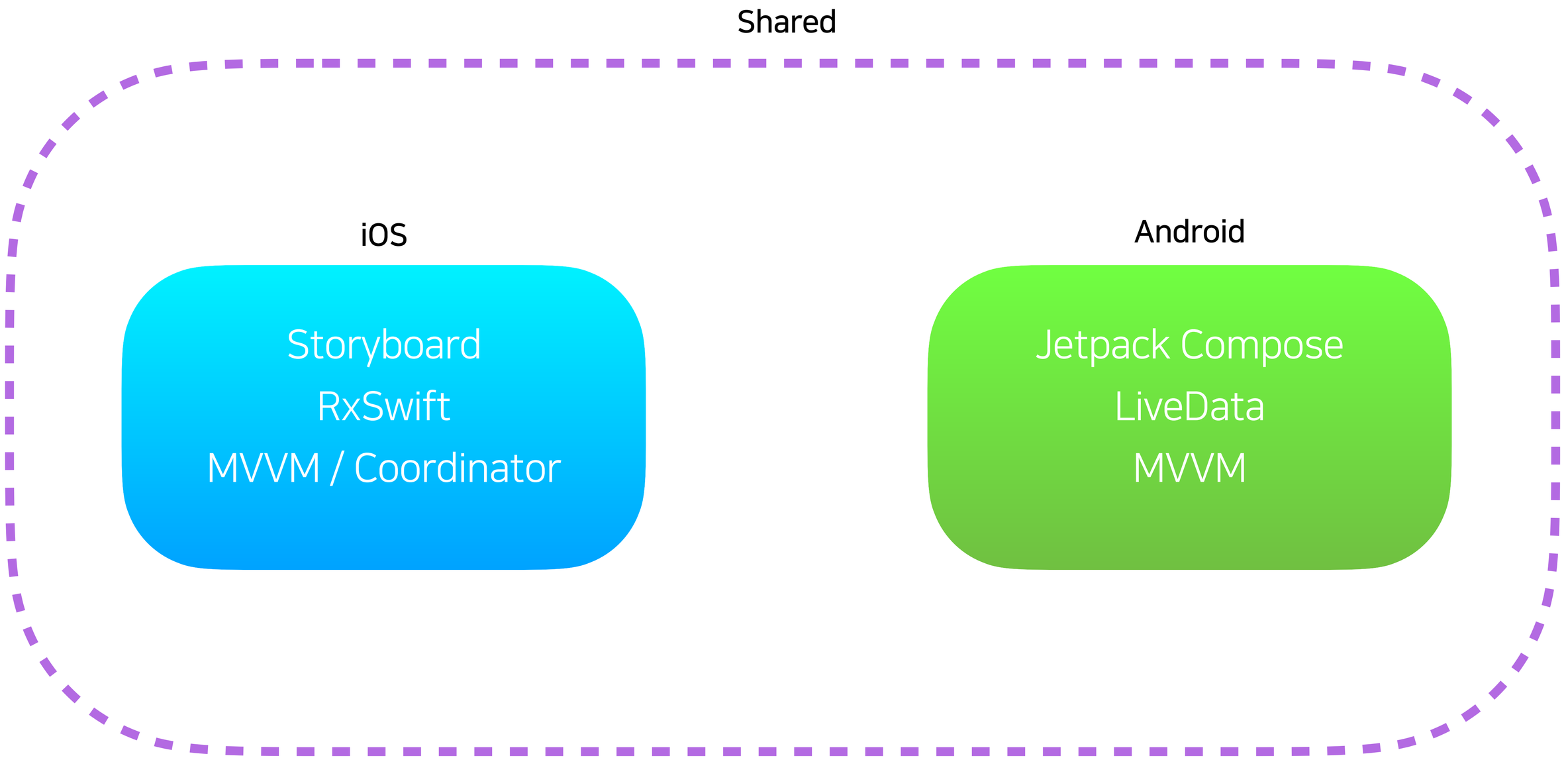
KMM 라이브러리 구성도



KMM 라이브러리 사용 장점

- 공용 인터페이스를 통해 어디서든 사용 가능
- 플랫폼 분기에 신경쓰지 않고 구현 가능
- 재사용성
- 기능별 역할 분담을 통해 도메인 개발 스코프를 줄임

UI 구성 및 반응형 개발에 플랫폼 자유도 제공



4. iOS 개발자를 위한 트러블 슈팅 경험 공유

Mutable Property in Global Object

Problem

DEVIEW 2021

```
object TestObject {

    var isInitialized = false

    fun initialize() {
        isInitialized = true
    }
}
```

Run on Android

DEVIEW 2021

```
object TestObject {

    var isInitialized = false

    fun initialize() {
        isInitialized = true
    }
}
```

😀 Work fine

Run on iOS

DEVIEW 2021

```
 0x10c101d21 <+145>: jmp    0x10c101d23               ; <+147> at <compiler-generated>
    0x10c101d23 <+147>: movq   -0x60(%rbp), %rdi
    0x10c101d27 <+151>: callq  0x10be524e0                ;
kfun:com.linecorp.abc.demaecan.driver.manager.TestObject#initialize(){} at AppManager.kt
    0x10c101d2c <+156>: jmp    0x10c101d2e                ; <+158> at <compiler-generated>:1
    0x10c101d2e <+158>: jmp    0x10c101d5c                ; <+204> at <compiler-generated>
    0x10c101d30 <+160>: movq   %rax, %rdi
    0x10c101d33 <+163>: movq   %rdx, -0x68(%rbp)
    0x10c101d37 <+167>: callq  0x10caa06ca                ; symbol stub for:
__cxa_begin_catch
    0x10c101d3c <+172>: movq   (%rax), %rax
    0x10c101d3f <+175>: movq   -0x58(%rbp), %rcx
    0x10c101d43 <+179>: movq   %rax, (%rcx)
    0x10c101d46 <+182>: movq   %rax, -0x70(%rbp)
    0x10c101d4a <+186>: callq  0x10caa06d6                ; symbol stub for: __cxa_end_catch
    0x10c101d4f <+191>: movq   -0x70(%rbp), %rdi
    0x10c101d53 <+195>: callq  0x10c439d60                ;
Kotlin_ObjCExport_trapOnUndeclaredException
```

😡 What the $%)*@&🙈

Solution

DEVIEW 2021

```
@ThreadLocal
object TestObject {

    var isInitialized = false

    fun initialize() {
        isInitialized = true
    }
}
```

Kotlin Native Tip

- Global Object 인스턴스들은 기본적으로 frozen 상태입니다.
  즉, 모든 스레드가 액세스할 수 있지만 변경할 수 없습니다.

- @ThreadLocal을 사용해 각 스레드에 변경 가능한 복사본을 제공할 수 있습니다.

DEVIEW 2021

Concurrency Issue of @ThreadLocal

Problem


```
@ThreadLocal
object Counter {

    var count = 0

    fun increase() {
        count++
        println("count -> $count on ${Thread.currentThread()}")
    }
}
```

Run on Android

DEVIEW 2021

```
runBlocking {
    Counter.increase()

    launch(Dispatchers.Default) {
        Counter.increase()
    }
}
```

Run on Android


```
runBlocking {
    Counter.increase()

    launch(Dispatchers.Default) {
        Counter.increase()
    }
}
```

😀

```
I/System.out: count -> 1 on main
I/System.out: count -> 2 on DefaultDispatcher-worker-1
```

Run on iOS

DEVIEW 2021

```
let globalQueue = DispatchQueue.global(qos: .userInteractive)

Counter().increase()

globalQueue.async {
   Counter().increase()
}
```

Run on iOS

DEVIEW 2021

```
let globalQueue = DispatchQueue.global(qos: .userInteractive)

Counter().increase()

globalQueue.async {
   Counter().increase()
}
```

```
count -> 1 on <NSThread: 0x600003848740>{number = 1, name = main}
count -> 1 on <NSThread: 0x600003808400>{number = 5, name = (null)}
```

Solution


```
object Counter {

    private var countReference = NativeAtomicReference(0)

    fun increase() {
        countReference.value++
        println("count -> ${countReference.value}")
    }
}
```

Kotlin Native Tip

- Global Object 인스턴스들은 기본적으로 frozen 상태입니다.
  즉, 모든 스레드가 액세스할 수 있지만 변경할 수 없습니다.

- @ThreadLocal을 사용해 각 스레드에 변경 가능한 복사본을 제공할 수 있습니다.

GCD Issue in Kotlin Native

Problem

DEVIEW 2021

```
val queue = dispatch_get_global_queue(
    DISPATCH_QUEUE_PRIORITY_BACKGROUND.toLong(), 0
)

dispatch_async(queue, {
    println("block invoked")
})
```

Run on iOS

DEVIEW 2021

```
Thread 4 Queue : com.apple.root.background-qos (concurrent)
#0  0x00007fff60309946 in __pthread_kill ()
#1  0x00007fff60343615 in pthread_kill ()
#2  0x000000011741dcb5 in abort ()
#3  0x000000010e09e769 in konan::abort() ()
#4  0x000000010e09ec3c in (anonymous namespace)::TerminateHandler::kotlinHandler()::'lambda'()::operator()()
const ()
#5  0x000000010e09eb96 in void (anonymous namespace)::$_0::operator()<(anonymous
namespace)::TerminateHandler::kotlinHandler()::'lambda'()>((anonymous
namespace)::TerminateHandler::kotlinHandler()::'lambda'()) ()
#6  0x000000010e09eb69 in (anonymous namespace)::TerminateHandler::kotlinHandler() ()
#7  0x00000001161f2aa7 in std::__terminate(void (*)()) ()
#8  0x00000001161f2a49 in std::terminate() ()
#9  0x000000010e0be016 in (anonymous namespace)::terminateWithIllegalSharingException(ObjHeader*) ()
#10 0x000000010e0c8dca in ObjHeader* KRefSharedHolder::ref<(ErrorPolicy)3>() const ()
#11 0x000000010de87b5a in Kotlin_Interop_unwrapKotlinObjectHolder(objc_object*) ()
#12 0x000000010dbbf3bc in _6472697665722d6170702d6b6d6d3a736861726564_knbridge19 at /Users/user/Workspace/
line/demaecan/driver-app-kmm/shared/src/iosMain/kotlin/com/linecorp/abc/demaecan/driver/native/
PlatformFeatures.kt:37
#13 0x0000000117259578 in _dispatch_call_block_and_release ()
#14 0x000000011725a74e in _dispatch_client_callout ()
#15 0x000000011726c38c in _dispatch_root_queue_drain ()
#16 0x000000011726cb0f in _dispatch_worker_thread2 ()
#17 0x00007fff6034047a in _pthread_wqthread ()
#18 0x00007fff6033f493 in start_wqthread ()
```

😡 What the $%)*@&🙈

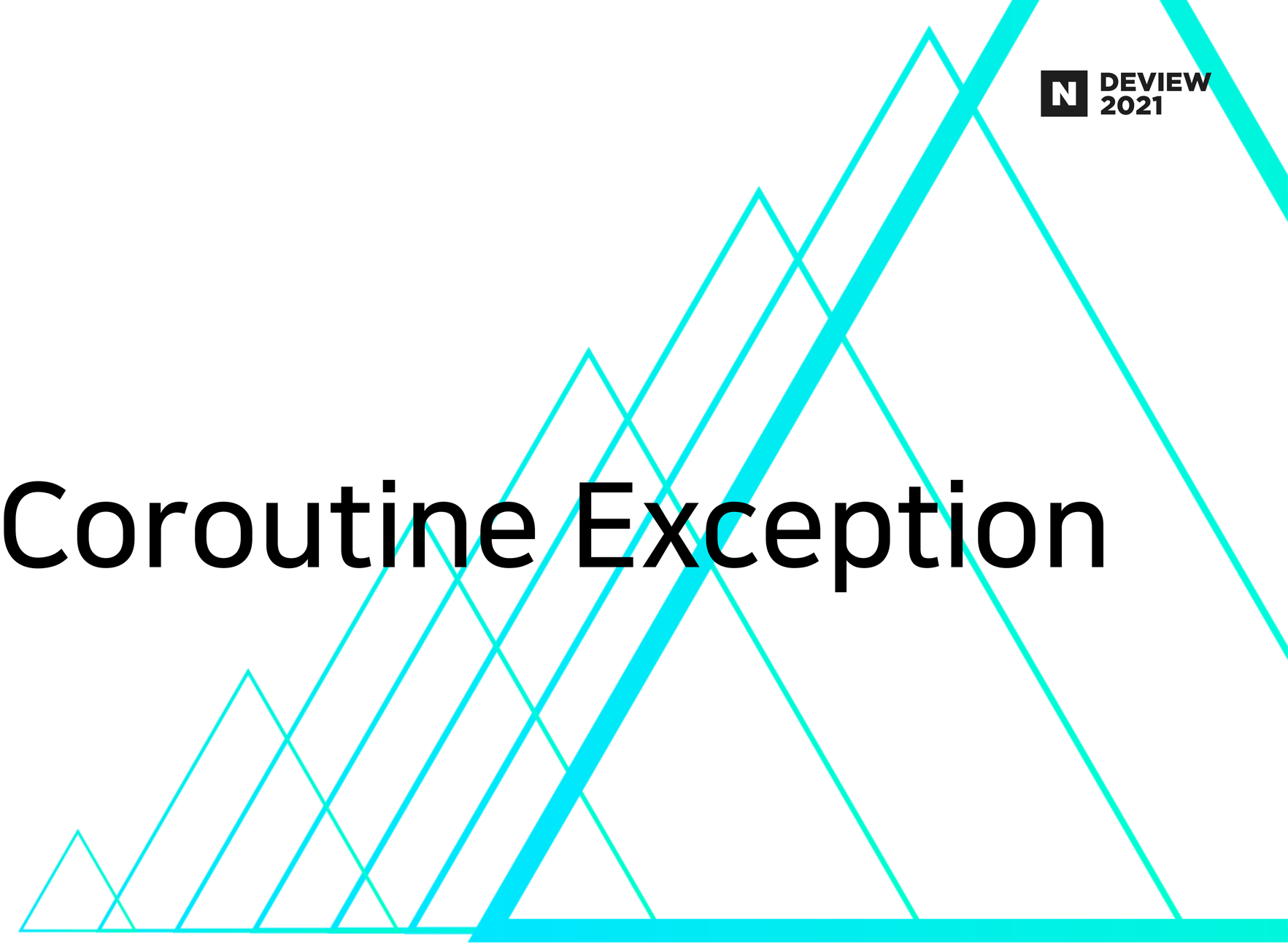
Solution


```
val queue = dispatch_get_global_queue(
    DISPATCH_QUEUE_PRIORITY_BACKGROUND.toLong(), 0
)

dispatch_async(queue, withFreezing {
    println("block invoked")
})

fun <T> withFreezing(block: T): T {
    block.freeze()
    return block
}
```

Kotlin Native Tip

- Thread 간 데이터 공유를 위해 Kotlin Object들을 Mutating 하지 못하도록 Freezing 하는데, GCD 블록이 정상적으로 Freezing 되지 않아 Crash가 발생한다. (버근가?)

- Kotlin Native는 모든 자료형에 사용할 수 있는 freezing 관련 함수들을 제공하고 있다.



Handle Coroutine Exception

Problem


```
class NumberProvider: CoroutineScope {

    override val coroutineContext: CoroutineContext by lazy {
        CoroutineScope(Job() + Dispatchers.Main).coroutineContext
    }

    suspend fun getOne(): Int {
        delay(1000)
        return 1
    }

    fun cancel() {
        coroutineContext.cancel()
    }
}
```

Run on iOS

DEVIEW 2021

```
let provider = NumberProvider()

provider.getOne { value, error in
   print("value, error -> ", value, error)
}

provider.cancel()
```

Run on iOS

DEVIEW 2021

```
0x10334e51e <+622>:  jmp    0x10334e520               ; <+624> at ObjCExportCoroutines.kt:67:8557
    0x10334e520 <+624>:  leaq   0xd047f1(%rip), %rax      ; theUnitInstance
    0x10334e527 <+631>:  movq   %rax, -0x138(%rbp)
    0x10334e52e <+638>:  jmp    0x10334e5d2               ; <+802> at Preconditions.kt
    0x10334e533 <+643>:  movq   -0xc8(%rbp), %rax
    0x10334e53a <+650>:  movq   (%rax), %rcx
    0x10334e53d <+653>:  movq   -0xf8(%rbp), %rdx
    0x10334e544 <+660>:  movq   (%rdx), %rsi
    0x10334e547 <+663>:  movq   0x20(%rcx), %rdi
    0x10334e54b <+667>:  movq   0x28(%rcx), %rdx
    0x10334e54f <+671>:  callq  0x1034b8610               ; Kotlin_ObjCExport_runCompletionFailure
->  0x10334e554 <+676>:  jmp    0x10334e556               ; <+678> [inlined] <anonymous> + 28 at Result.kt:229
    0x10334e556 <+678>:  jmp    0x10334e558               ; <+680> [inlined] <anonymous> + 30 at Result.kt:229
    0x10334e558 <+680>:  jmp    0x10334e51c               ; <+620> [inlined] fold + 21 at ObjCExportCoroutines.kt:18
    0x10334e55a <+682>:  leaq   0xd4aa2f(%rip), %rax      ; __unnamed_411
    0x10334e561 <+689>:  movq   -0xd8(%rbp), %rcx
    0x10334e568 <+696>:  movq   %rax, (%rcx)
    0x10334e56b <+699>:  movq   (%rcx), %rax
    0x10334e56e <+702>:  movq   (%rax), %rdx
    0x10334e571 <+705>:  andq   $-0x4, %rdx
    0x10334e575 <+709>:  movq   (%rdx), %rdx
    0x10334e578 <+712>:  movq   0x90(%rdx), %rdx
    0x10334e57f <+719>:  movq   %rax, %rdi
    0x10334e582 <+722>:  movq   -0xe0(%rbp), %rsi
    0x10334e589 <+729>:  callq  *%rdx
    0x10334e58b <+731>:  movq   %rax, -0x140(%rbp)
```

😡 What the $%)*@&🙈

Solution

DEVIEW 2021

```
class NumberProvider: CoroutineScope {

    override val coroutineContext: CoroutineContext by lazy {
        CoroutineScope(Job() + Dispatchers.Main).coroutineContext
    }

    @Throws(Throwable::class)
    suspend fun getOne(): Int {
        delay(100)
        return 1
    }

    fun cancel() {
        coroutineContext.cancel()
    }
}
```

Run on iOS

DEVIEW 2021

```
let provider = NumberProvider()

provider.getOne { value, error in
    print("value, error -> ", value, error)
}

provider.cancel()
```

```
value, error ->  nil Optional(Error Domain=KotlinException Code=0 "There is no
event loop. Use runBlocking { ... } to start one."
UserInfo={NSLocalizedDescription=There is no event loop. Use runBlocking { ...
} to start one., KotlinException=kotlin.IllegalStateException: There is no
event loop. Use runBlocking { ... } to start one., KotlinExceptionOrigin=})
```

Kotlin Native Tip

- Kotlin Suspend Function은 Swift Code Generating을 통해 Result와 Error를 인자로 받는 Callback Function으로 변경된다.

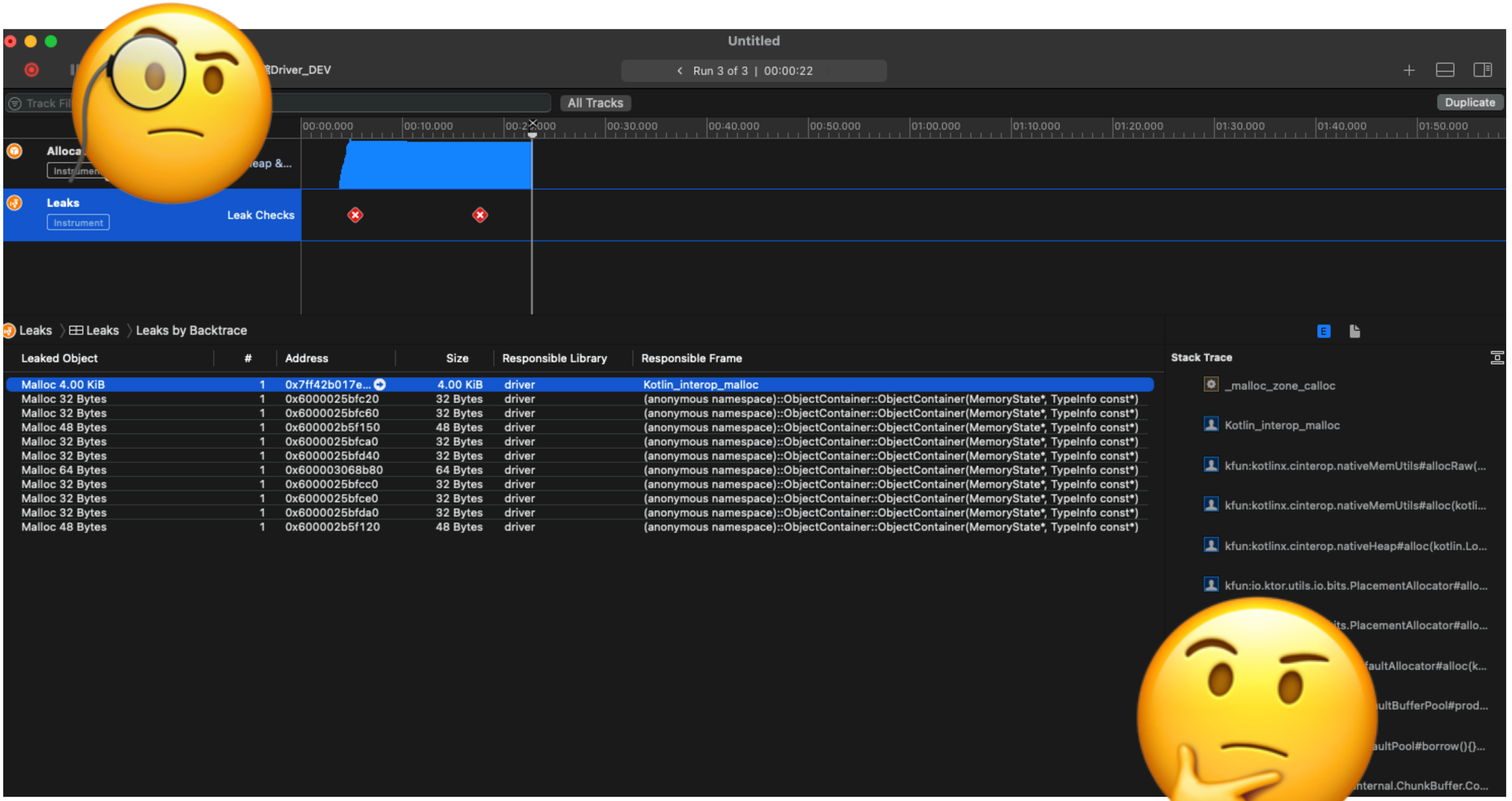
- 이 때 Kotlin에서 발생한 예외가 Error로 전달되길 기대하지만 기대와 달리 예외를 처리하지 못하고 Crash를 발생시킨다.

- Crash를 유발하는 예외를 Try Catch로 방어할 수 있는 Kotlin과 달리 Swift는 그렇지 못하다. @Throws를 사용해 Error 객체로 전달 받을 수 있다.

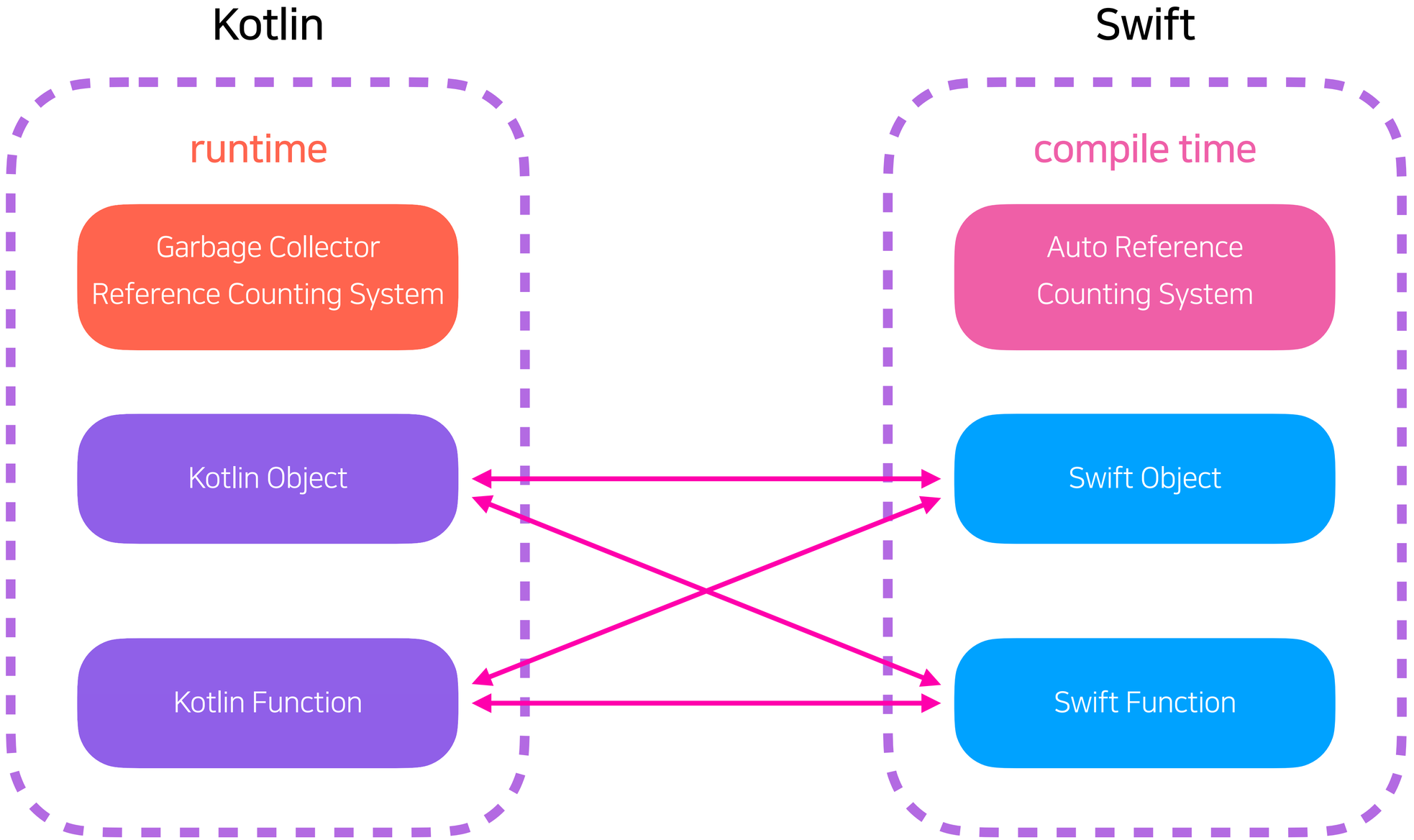
Memory Management System on Kotlin Native iOS

Leaks (Anonymous namespace::ObjectContainer)

DEVIEW 2021



- Reference Counting System
- Cyclic garbage collector
  based on a trial-deletion algorithm


Kotlin
Swift
runtime
compile time
Garbage Collector
Reference Counting System
Auto Reference
Counting System
Kotlin Object
Swift Object
Kotlin Function
Swift Function

Kotlin Native Tip

- iOS App 실행 시 Kotlin Native Runtime이 함께 실행되고 Kotlin Object들이 관리된다.

- 런타임 중 Kotlin과 Swift 간 통신을 위해 지속적으로 객체 및 함수의 컨버팅이 이루어진다.
  이로 인해 짐작하기 힘든 여러 문제들이 발생한다. (Crash, 비정상 Leaks)
  Xcode Instruments에 잡힌 Leaks는 실제 Leaks가 아니다.

- 아직 Kotlin Native Runtime의 메모리 관리 시스템이 완벽하지 않다.
  JetBrains는 완전히 새로운 메모리 관리 시스템을 준비 중이다.

  https://blog.jetbrains.com/kotlin/2021/08/try-the-new-kotlin-native-memory-manager-development-preview/#why-do-we-need-a-new-memory-manager

Kotlin Native Runtime 문제를 해결하기 위한 팁

Shared

```
fun start() {
    ABCNotifications.beginListening()
    AppVersionManager.checkTermsOfServiceUpdated()
    resume()
}
```

iOS

```
AppManager.shared.start()
```

Shared

```
@Throws(Throwable::class)
fun start() {
    ABCNotifications.beginListening()
    AppVersionManager.checkTermsOfServiceUpdated()
    resume()
}
```

iOS

```
do {
    try AppManager.shared.start()
} catch {
    Crashlytics.crashlytics().record(error: error)
}
```

Kotlin Native Tip

- Swift 에서 호출하는 KMM 함수에 @Throws를 적극적으로 사용하라.
- Kotlin Native Runtime 내부에서 발생한 예외에 대해 처리 할 수 없어 Crash가 발생하는데, @Throws를 사용하면 어떤 Kotlin Exception 이던 Swift Exception 으로 컨버팅 해 준다.

5. 라인에서 제작한 KMM 라이브러리 소개

abc-notifications-kmm

Remote Notification Manager for Kotlin Multiplatform Mobile

- APNs와 FCM을 하나의 인터페이스로 제공
- 간단한 권한 설정 제공
- 코드양을 극적으로 줄여줌
- 여러 iOS 버전을 신경쓰지 않도록 해줌
- FCM을 쉽게 iOS에 통합 가능
- KMM Shared에서 공통 인터페이스 사용 가능

abc-shared-storage-kmm

- iOS와 Android의 로컬 저장소를 하나의 인터페이스로 제공
- 일반 저장소 (iOS UserDefaults, Android SharedPreferences) 제공
- 보안 저장소 (iOS Keychain, Android EncryptedPreferences) 제공
- Annotation을 제공해 인터페이스 정의만으로 맞춤 클래스 생성
- KMM Shared에서 공통 인터페이스 사용 가능

abc-location-kmm

- LocationManager와 FusedLocationProviderClient를 하나의 인테페이스로 제공
- 간단한 권한 설정 제공
- 코드양을 극적으로 줄여줌
- KMM Shared에서 공통 인터페이스 사용 가능

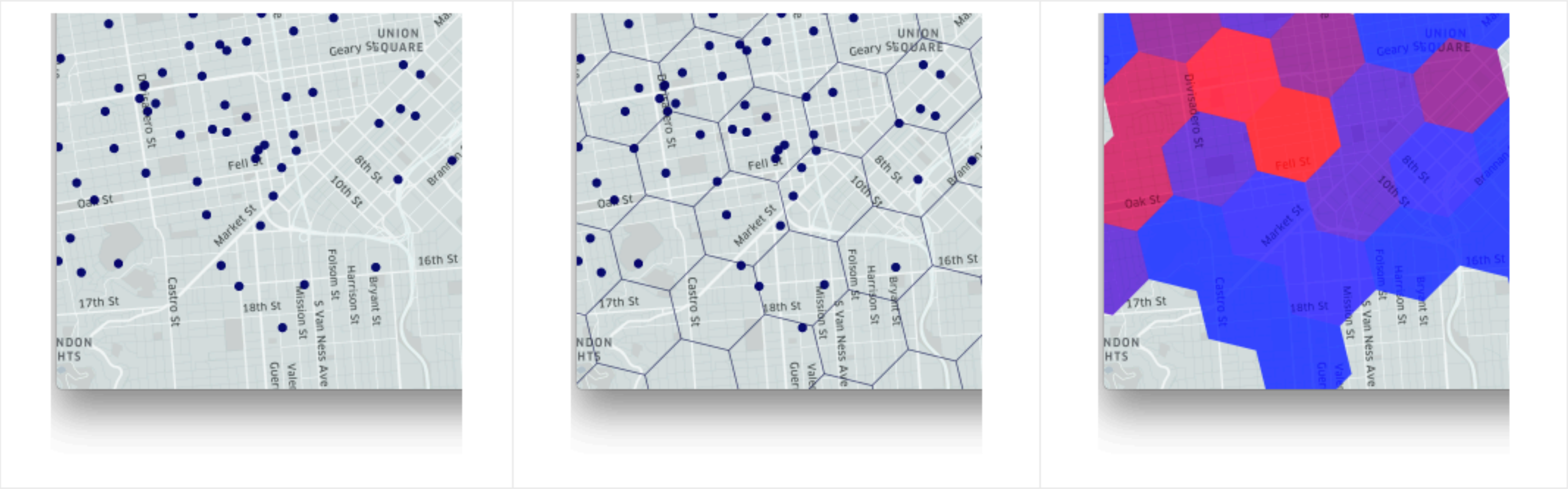
abc-analytics-tools-kmm

- 여러가지 트래킹 툴을 하나의 인터페이스로 통합 제공
- 자동 스크린 뷰 이벤트 전송
- 스크린 이름 매핑 기능
- UI 요소를 이벤트 트리거로 바인딩하여 보일러플레이트 제거
- KMM Shared에서 공통 인터페이스 사용 가능

abc-h3-kmm

A library to convert Uber's H3 geo-index to LatLng vertices for Kotlin Multiplatform Mobile

- Uber H3를 하나의 인터페이스로 제공
- KMM Shared에서 공통 인터페이스 사용 가능



이 모든 라이브러리를 곧 오픈소스로 공개 예정

https://github.com/line

KMM 생태계를 더욱 풍성하게할 여러분들의 참여를 기다립니다.

https://github.com/line

Thank You