Topic for discussion:

KMM - Kotlin Multiplatform Mobile basics

Presentation by Rohan Pambhar
(Android Engineer)

Basics of KMM

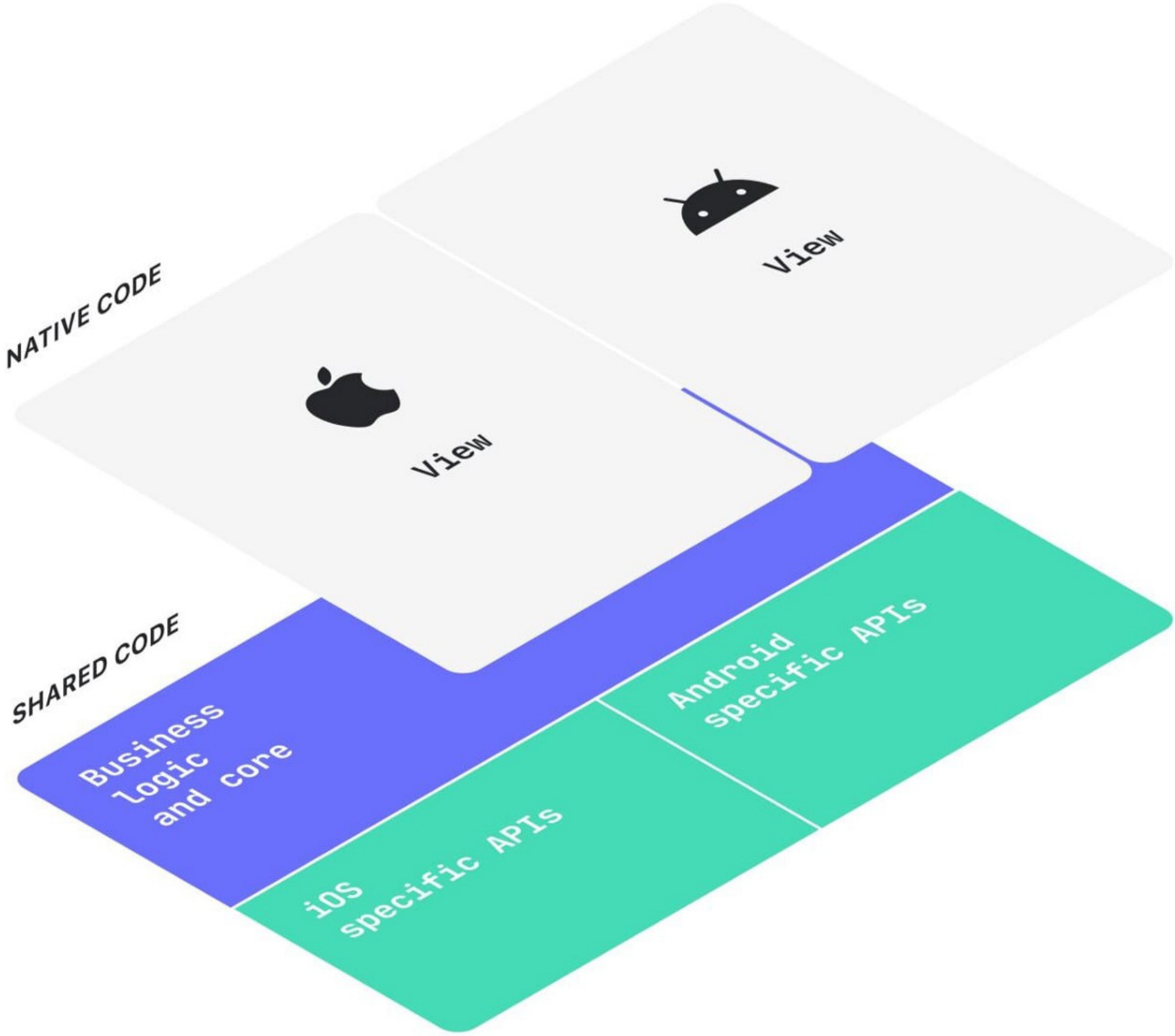
- Kotlin Multiplatform Mobile (KMM) is a framework that allows you to build an application that targets Android and iOS mobile platforms. More specifically, the UI layer *(in Clean Architecture terms*) is the only layer that needs to be developed natively while allowing most of the rest of the architectural layers to be shared.

- The shared architecture is provided to each native app as a compiled package.

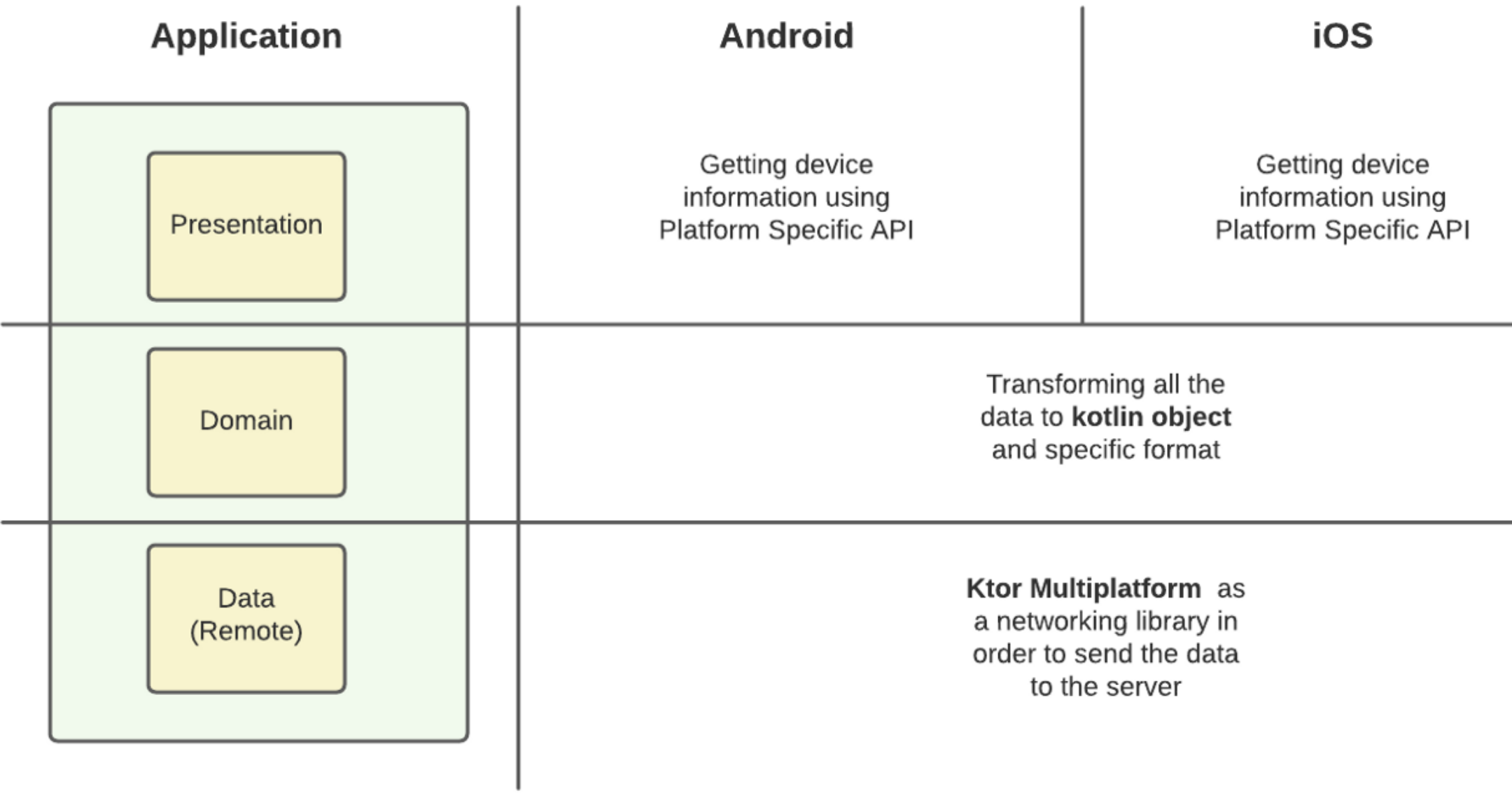
- This package is then accessed within the native app code.

- KMM has the potential to be the foundation for several platforms, including Android, JavaScript, Java, Kotlin, and Swift iOS codebases.

Below is the image that clarifies what KMM looks like.



This is how your mobile application architecture looks like when you use KMM.



Setup environment to use KMM

You should install the latest stable versions for compatibility and better performance.

- Install necessary tools.
  - Android Studio
    - External JDK
    - KMM plugin
    - Kotlin plugin

- If you are targeting an iOS application, then install
  - XCode
    - Cocoapods

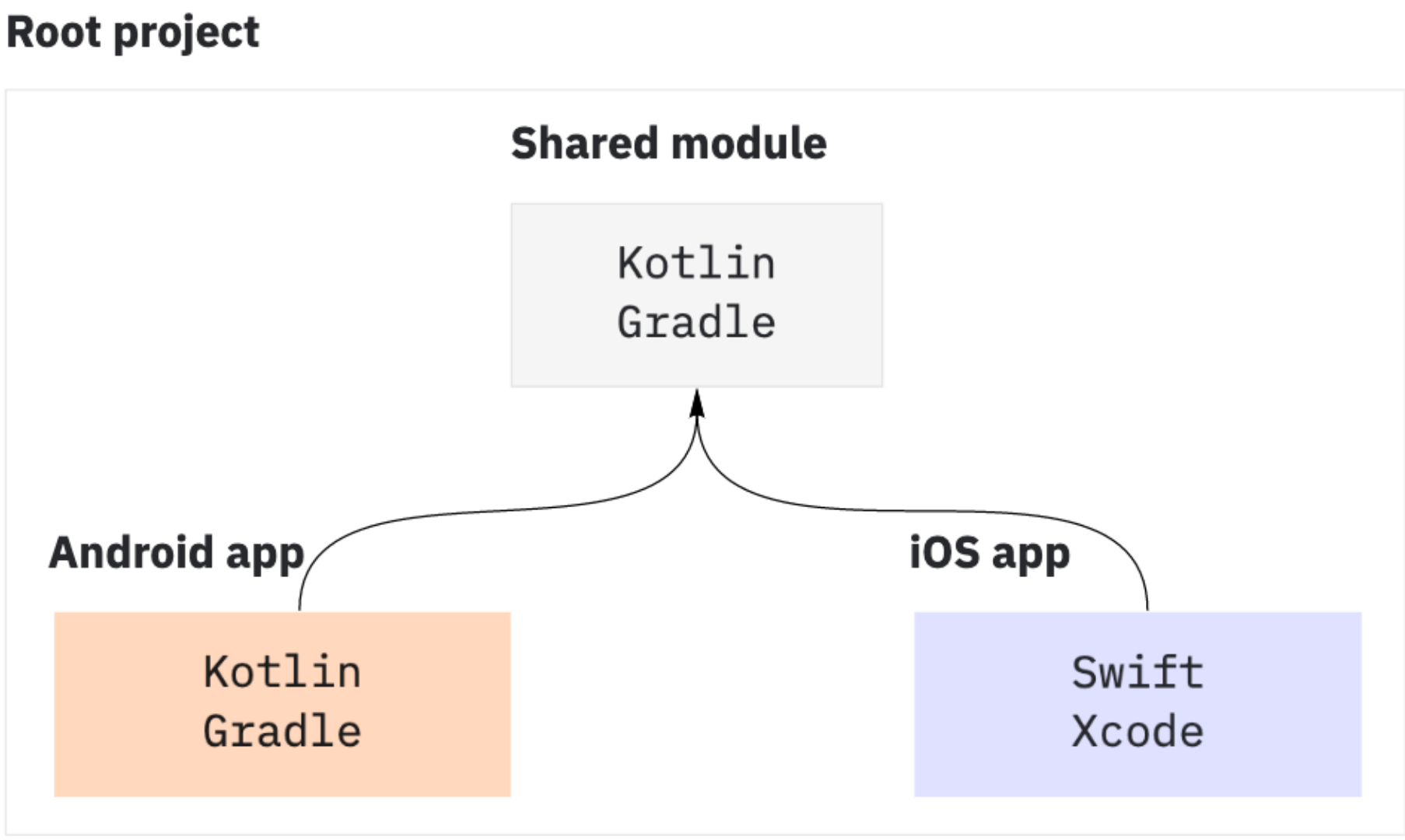
To make sure everything works as expected,
install and run the **KDoctor** tool:

KDoctor works on macOS only.

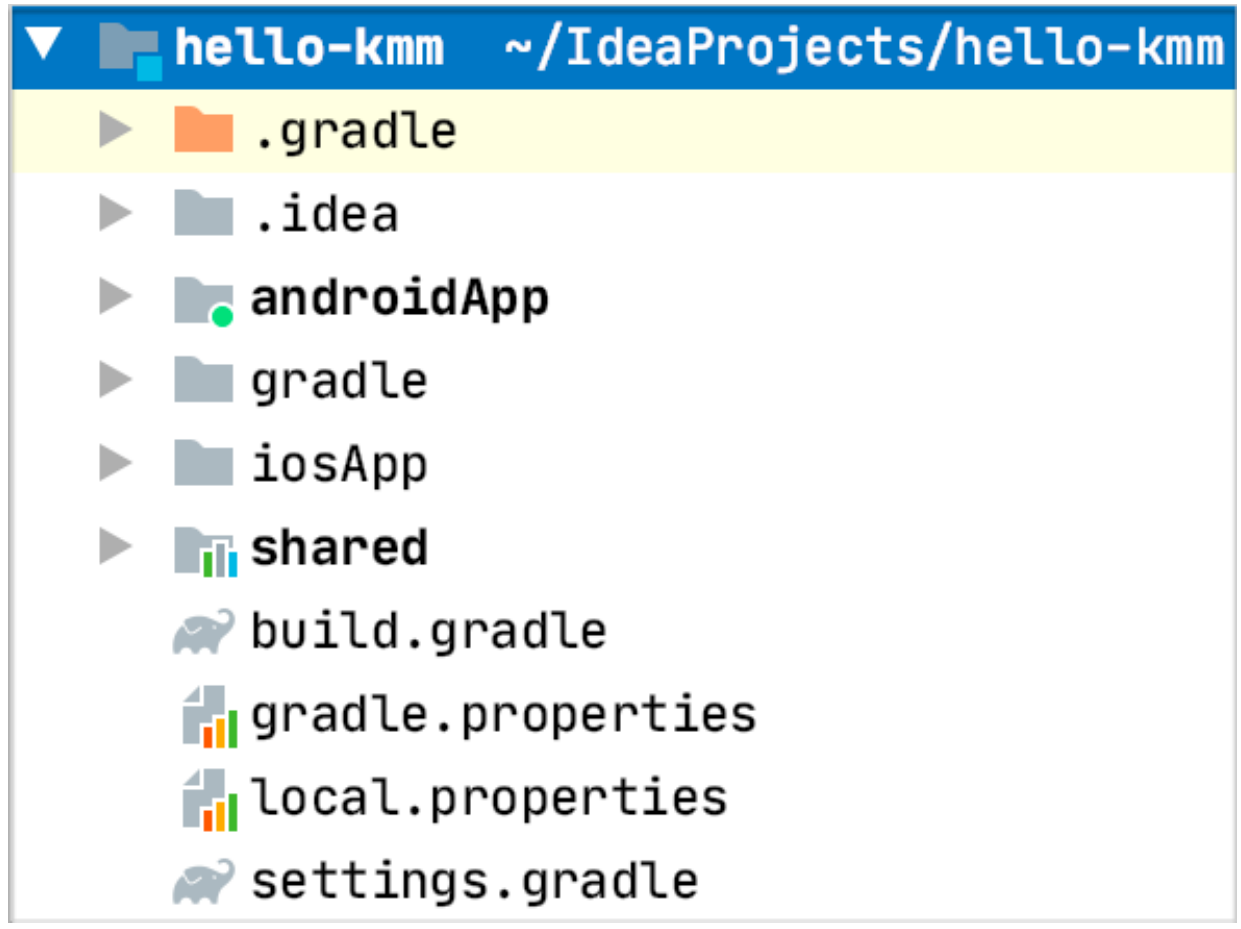
1. In the Android Studio terminal or your command-line tool, run the following command to install the tool using Homebrew: ***brew install doctor***
2. If you don't have Homebrew yet, install it or see the KDoctor README for other ways to install it.
3. After the installation is completed, call KDoctor in the console: ***kdoctor***
4. If KDoctor diagnoses any problems while checking your environment, review the output for issues and possible solutions

Understand the KMM project structure

This is how your root project looks like -



This is a basic structure of a cross-platform mobile project:



- The iOS application is produced from an Xcode project. It’s stored in a separate directory within the root project.

- Xcode uses its own build system; thus, the iOS application project isn’t connected with other parts of the Multiplatform Mobile project via Gradle.

- Instead, it uses the shared module as an external artifact – framework.


it Agenturen