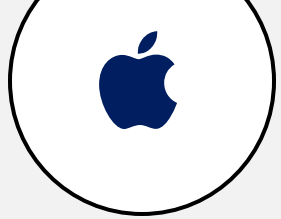
Inside Mobile Frameworks

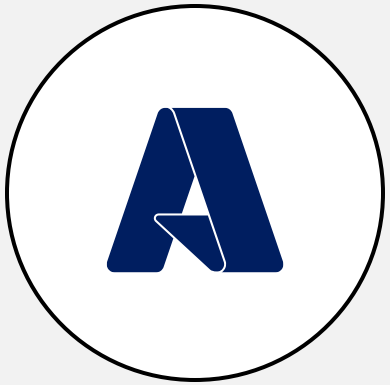
How do they work?

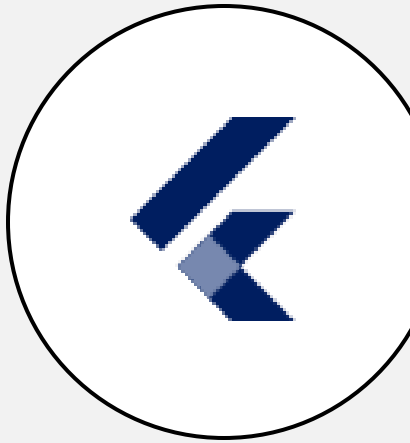
ConFoo February 2025

Mathieu Fillion

12 yearsat nventive







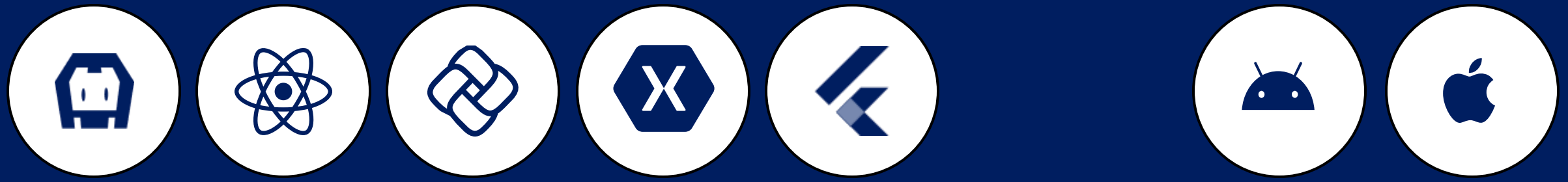
Windows Phone

Mobile / Xamarin / .NET / Flutter

Azure

I always had the user at the center of my attention and loved collaborating with designers.

Multiplatform frameworks.



Categories

Cordova

Ionic

- Web rendering
- Near 100% shared code

Xamarin.classic

Kotlin Multiplatform

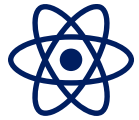
- Native rendering
- Shared business logic
- Platform specific code for the view

Flutter

Xamarin.Forms

.NET MAUI

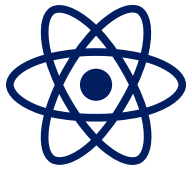
Uno Platform



React Native

- Native rendering*
- Near 100% shared code

Multiplatform



React Native

- Typescript
- Developed by Meta
- Released in 2015

Xamarin.Forms

.NET

Uno Platform

- C#, .NET and XAML(optional)
- Xamarin.iOS/Android released in 2012
- Now rebranded .NET MAUI
- Uno Platform is built right here in Montreal. Open-Sourced in 2018.

Flutter

- Dart language
- Developed by Google
- Released in 2017

Before we move on…

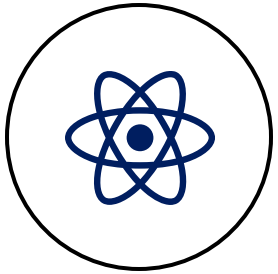
They are all complete and mature solutions.

They are all excellent choices for your next project.

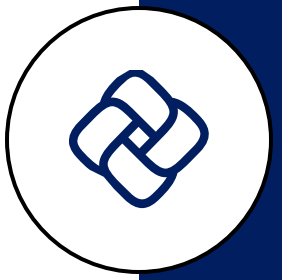
Rendering

How apps are drawn

Drawing native components



Leverage iOS SDK (UIKit) and Android.Views to render UI Elements



Drawing pixels

Use their own rendering engines (Skia or *Impeller*) to draw widgets.

Drawing native components

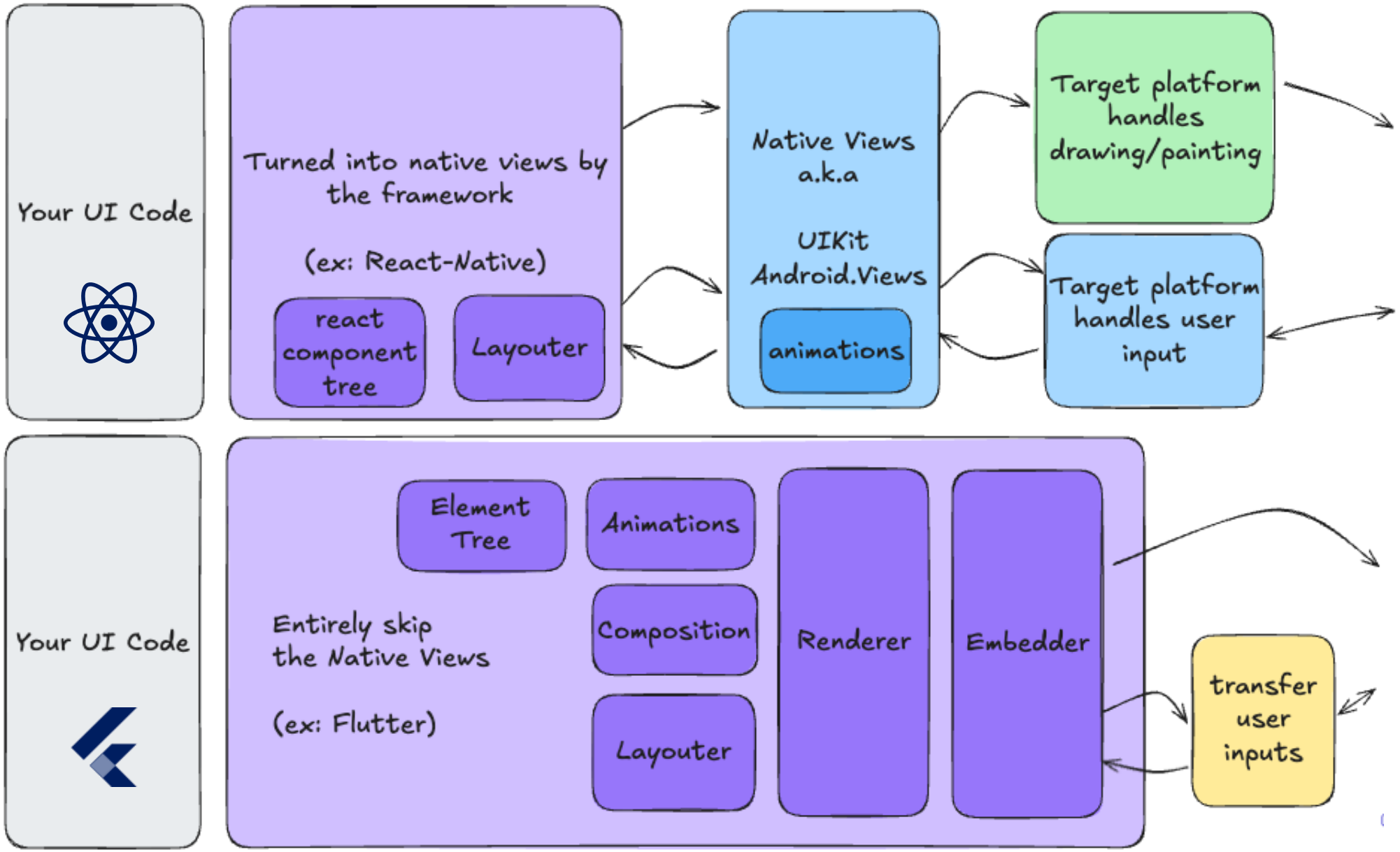
```
const Item = ({title}: ItemProps) => (
  <View style={styles.item}>
    <Text style={styles.title}>{title}</Text>
  </View>
);

const App = () => (
  <SafeAreaProvider>
    <SafeAreaView style={styles.container} edges={['top']}>
      <VirtualizedList
        initialNumToRender={4}
        renderItem={({item}) => <Item title={item.title} />}
        keyExtractor={item => item.id}
        getItemCount={getItemCount}
        getItem={getItem}
      />
    </SafeAreaView>
  </SafeAreaProvider>
);
```

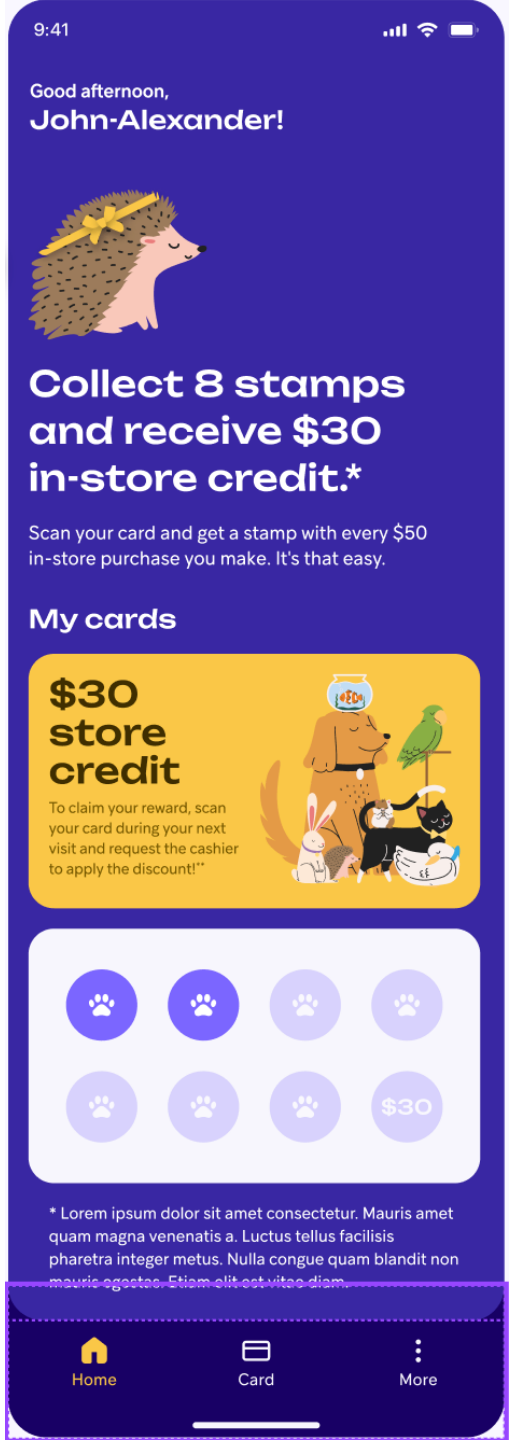
Android

```
<LinearLayout>
  <TextView/>
</LinearLayout>
```

```
<Layout>
  <RecyclerView/>
</Layout>
```

Drawing native components





Drawing native components

Architecture **challenges** 💪

- Make platforms look the same
- Find a common denominator for all platforms.

Architecture **benefits** ✅

- Delegate a lot of fine details
  - Accessibility voice-over
  - Text selection, copy-paste.
- Immediately get the new look-n-feel when iOS or Android release new versions.

Drawing pixels

Architecture **challenges** 💪

- Embedding Truly native views like Maps (now solved)
- Keep up with platform-specific stylings
- Re-implement features the target platforms already have.

Architecture **benefits** ✅

- Pixel-perfect apps on all platforms
- Animation engine is built-in.

Native APIs

Accessing platform-specifics

What is a native API ?

→ Part of the native Android SDK and iOS SDK.

→ Access their unique features.

- Location services (GPS)
- Bluetooth
- FaceID / Biometry
- Push notifications
- Accelerometers / gyroscope
- Haptics
- …

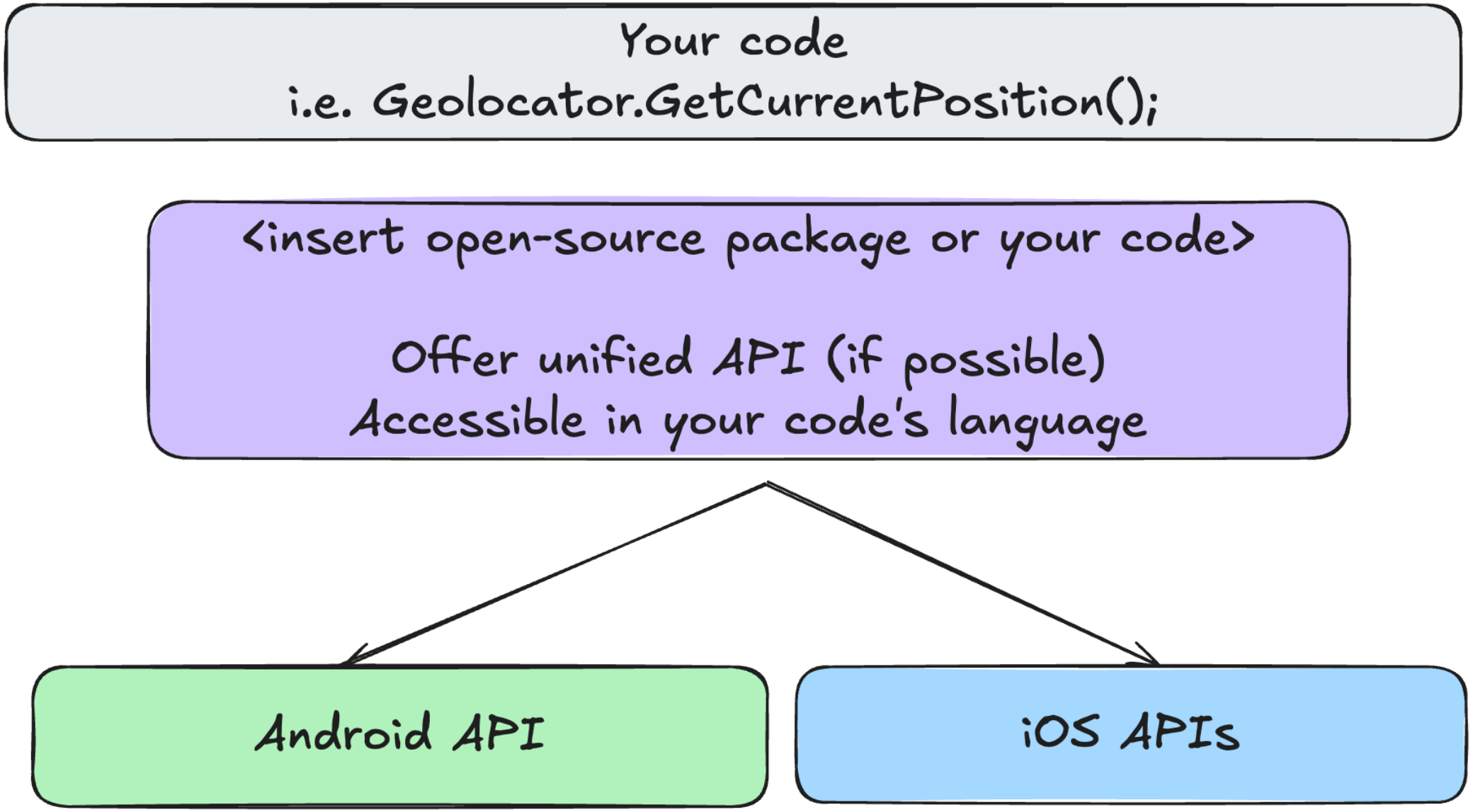
→ Create and render native views like text inputs, buttons, etc.

→ iOS SDKs and Android SDKs make those available in their language only.

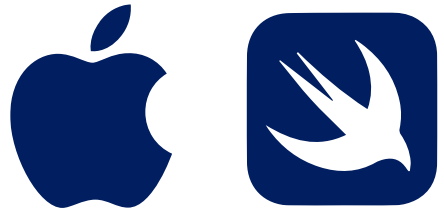
→ iOS – Swift or Obj-C

→ Android – Kotlin or Java

Native API in a multiplatform app

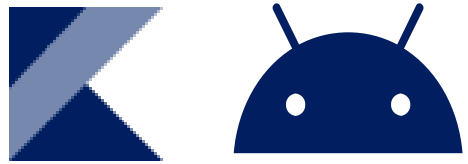


Retrieving the user's last known position



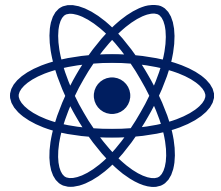
```swift
// Ask for Authorisation from the User.
self.locationManager.requestAlwaysAuthorization()

// For use in foreground
self.locationManager.requestWhenInUseAuthorization()

if CLLocationManager.locationServicesEnabled() {
    locationManager.delegate = self
    locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters
    locationManager.startUpdatingLocation()
}

// Delegate method from the CLLocationManagerDelegate protocol.
// triggered whenever a new location update is received by the location manager.
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    let lastLocation = locations.last! as CLLocation
    let recentLocations: CLLocationCoordinate2D = manager.location?.coordinate else { return }
    print("locations = \(locValue.latitude) \(locValue.longitude)")
}
```

Retrieving the user's last known position



```
    // Location Provider from Google Play Services
    fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)

    // Check for location permission before calling
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION)!= PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, arrayOf(Manifest.permission.ACCESS_FINE_LOCATION), REQUEST_LOCATION_PERMISSION)
    } else {
        getDeviceLocation()
    }
}

@SuppressLint("MissingPermission") // This assumes you have location permission by now
private fun getDeviceLocation(){
    fusedLocationClient.lastLocation
        .addOnSuccessListener { location -> Log.d( tag: "LOCATION",  msg: "Got my location!") /* Do Something */ }
        .addOnFailureListener { e -> Log.e( tag: "LOCATION",  msg: "Error getting location", e) }
}

override fun onRequestPermissionsResult(requestCode: Int,permissions: Array<out String>,grantResults: IntArray) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (requestCode == REQUEST_LOCATION_PERMISSION && grantResults.isNotEmpty() &&
        grantResults[0] == PackageManager.PERMISSION_GRANTED) {
        getDeviceLocation() // Now that we have permission, get the location
    }
}
```

Retrieving the user's last known position



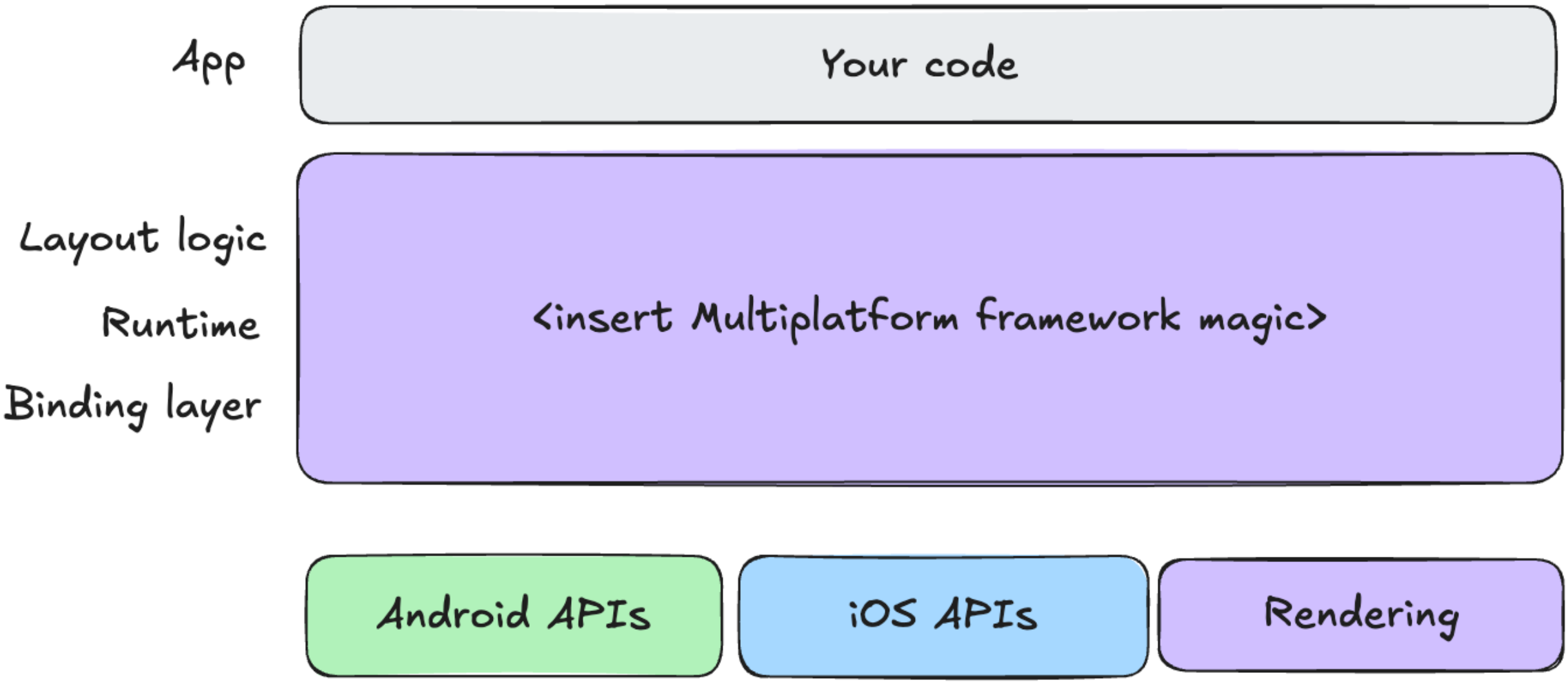
```
// Request location permissions
let { status } = await Location.requestForegroundPermissionsAsync()
if (status !== 'granted') {
  setErrorMsg('Permission to access location was denied')
  return
}

// Try to fetch the last known location first
let lastKnown = await Location.getLastKnownPositionAsync({})
if (lastKnown) {
  // I have my location!
} else {
  // If there's no last known location, fetch the current location
  let current = await Location.getCurrentPositionAsync({})
  // I have my location!
}
```

Same concept with Flutter or .NET

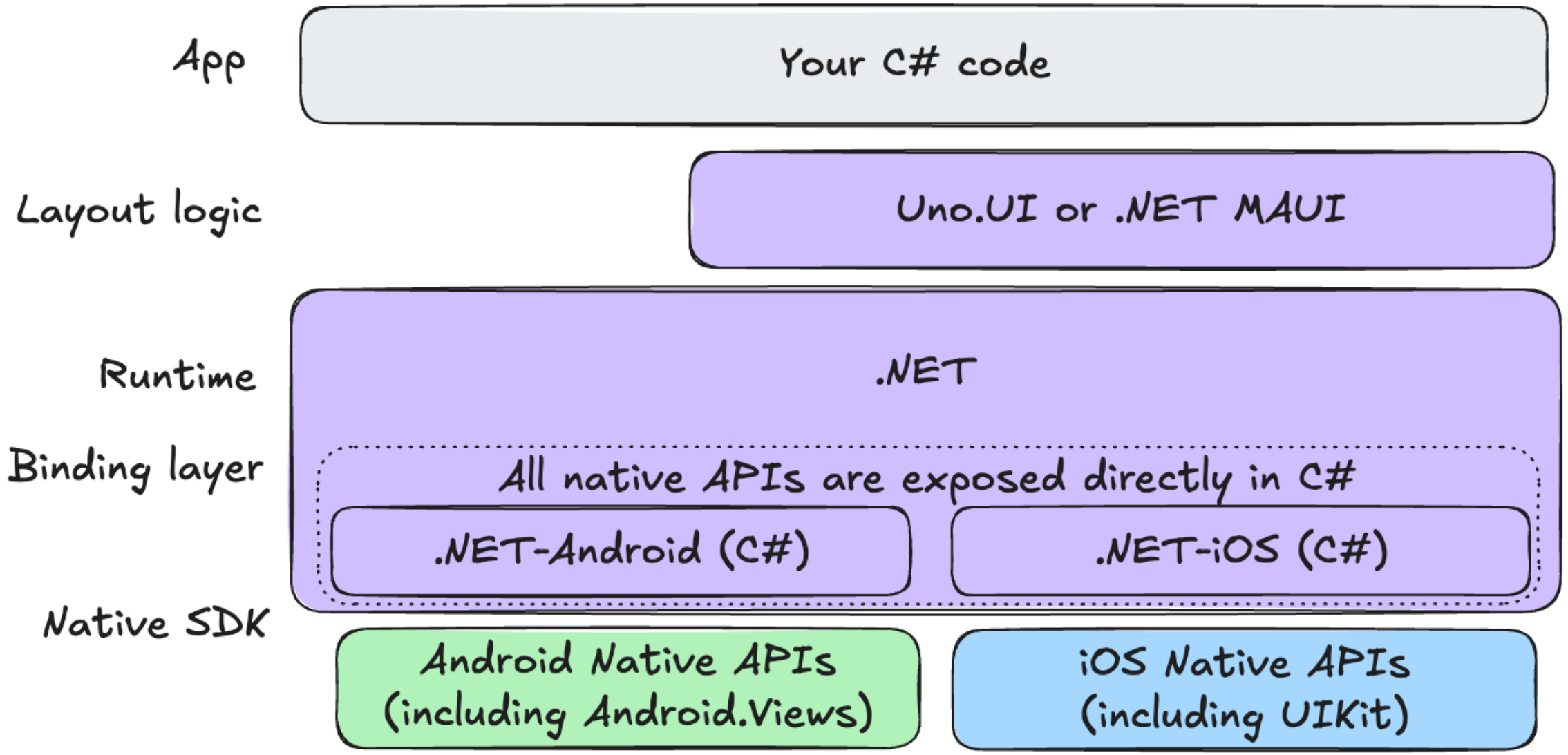
High-level Architecture

How does shared code become 2 native apps?

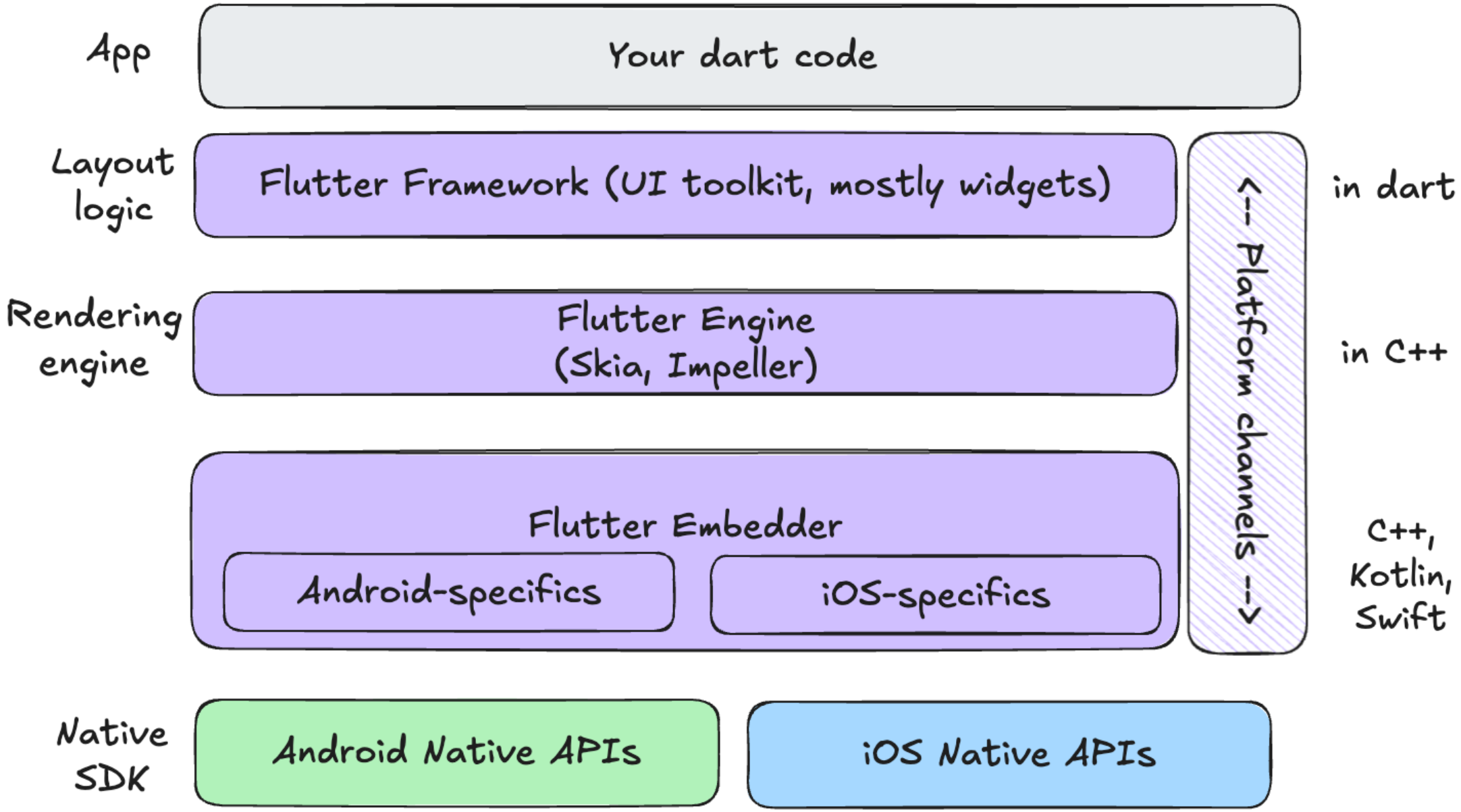
Common pieces



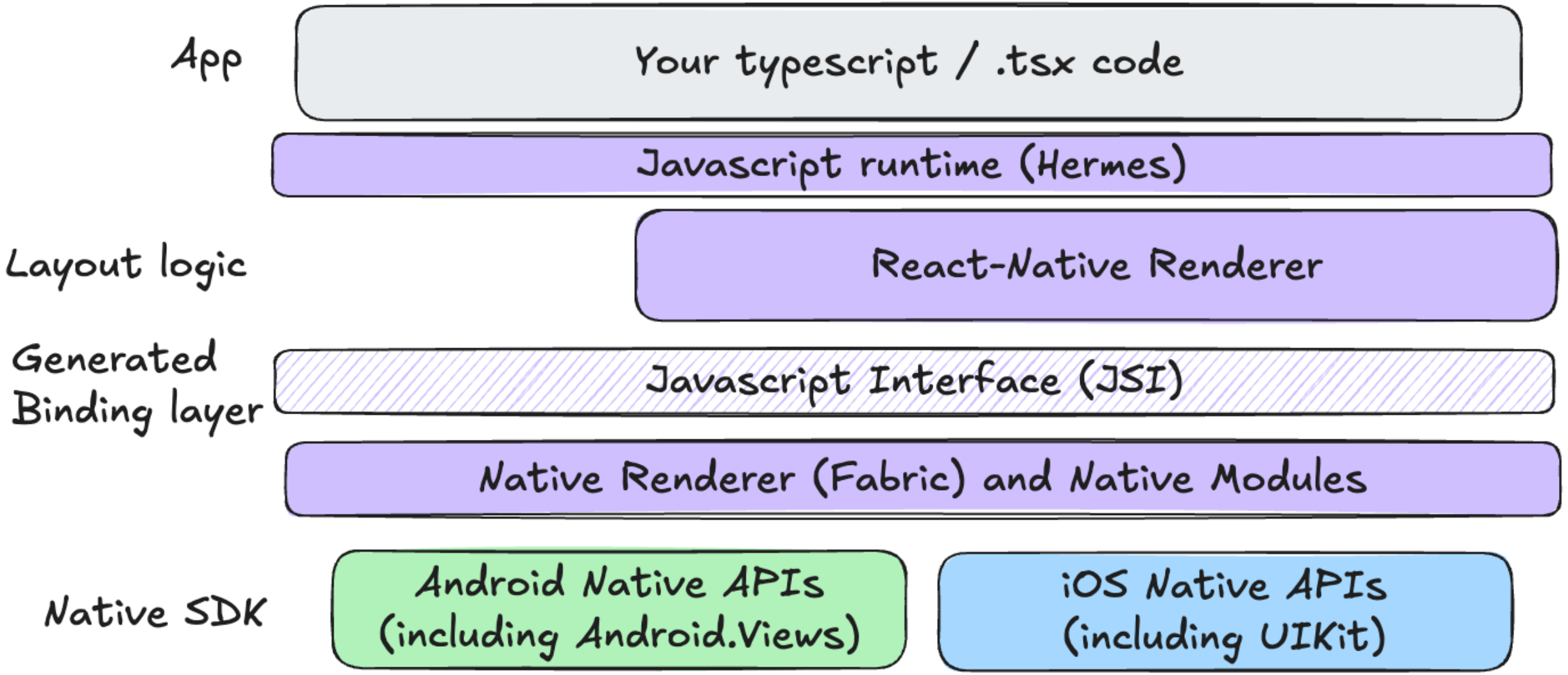
Uno Platform / .NET



Flutter


App
Your dart code
Layout logic
Flutter Framework (UI toolkit, mostly widgets)
in dart
Rendering engine
Flutter Engine
(Skia, Impeller)
in C++
Flutter Embedder
Android-specifics
iOS-specifics
<-- Platform channels -->
C++, Kotlin, Swift
Native SDK
Android Native APIs
iOS Native APIs

React-Native (new architecture since fall 2024)



High-level architecture

Packaging it all together

Not so different than purely native apps

What is native code?

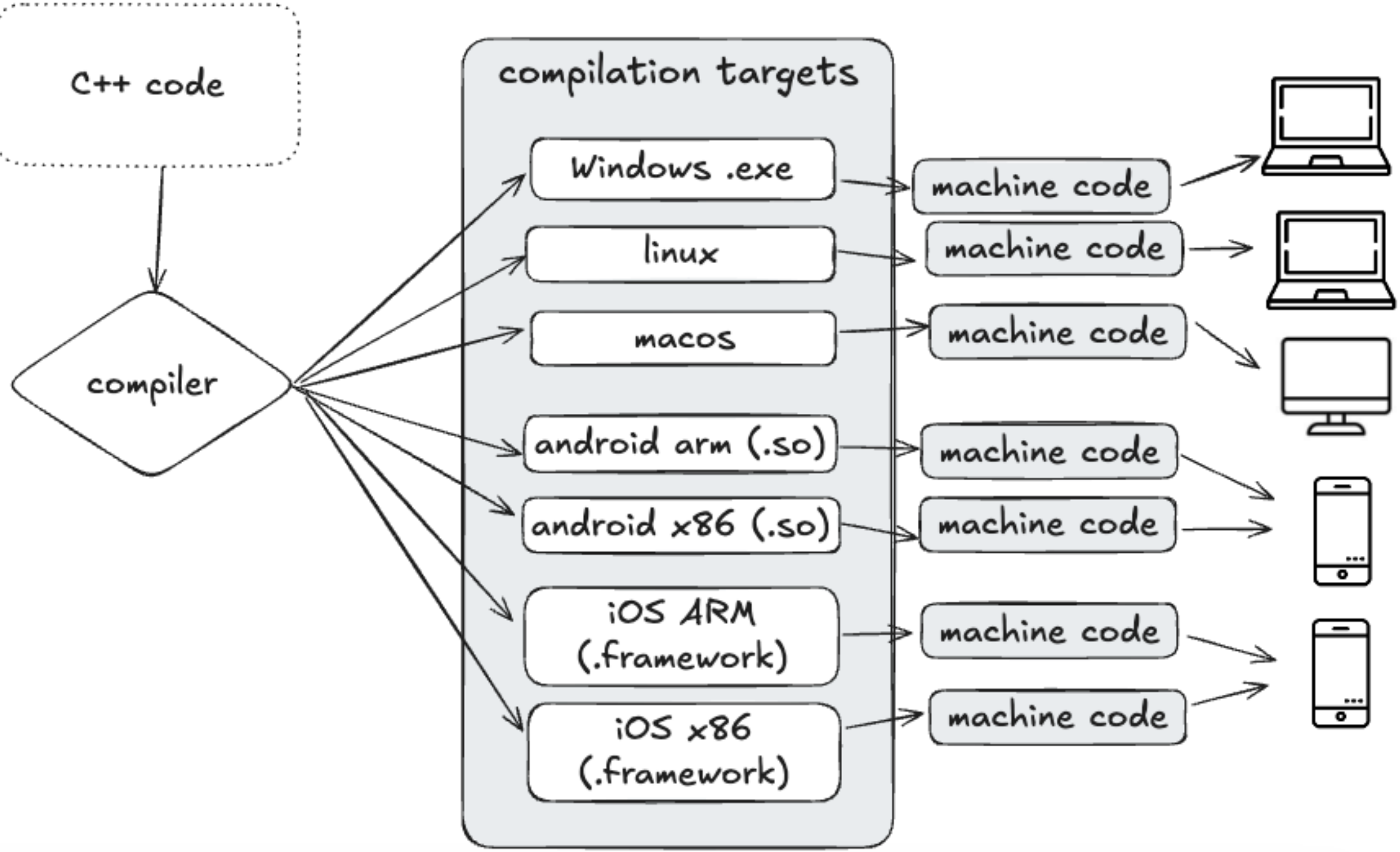
→ Code compiled to run directly on a specific platform or OS.

- x86 (32bit)
- x86_64
- ARMv7 (32bit)
- ARM64
- Apple Silicon (ARMv8.5)
- …

Examples:

C/C++
Rust
Go
Objective-C
Swift

And more.


C++ code
compiler
compilation targets
Windows .exe
linux
macos
android arm (.so)
android x86 (.so)
iOS ARM (.framework)
iOS x86 (.framework)
machine code
machine code
machine code
machine code
machine code
machine code
machine code

JVM, Dart VM, .NET Runtime. What are they?

→ Virtual machines built specifically for the OS/Platform.

→ JVM understands java and transforms it into machine code calls.

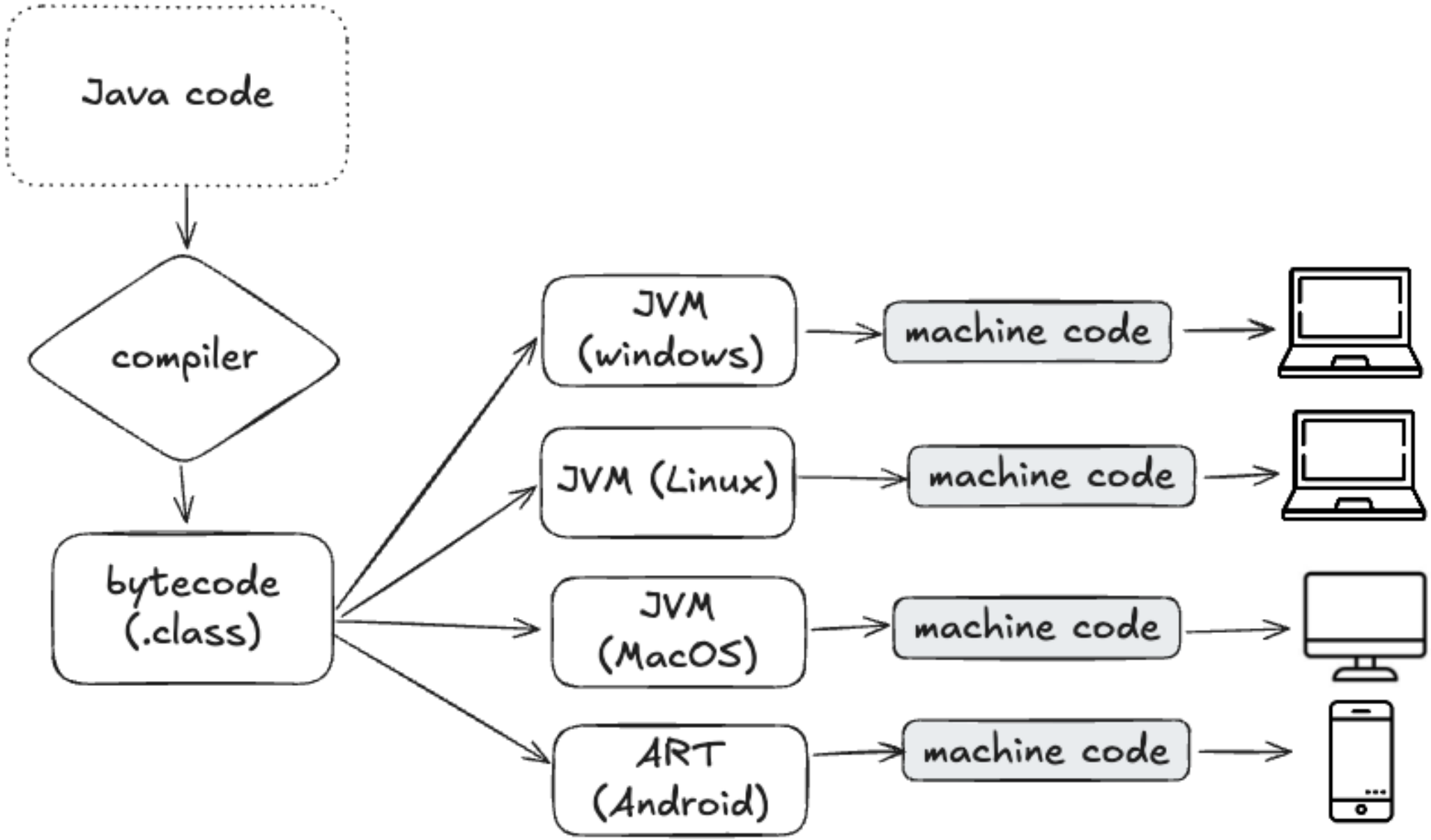
→ There are different JVMs for each OS/Platform.

Generally include:

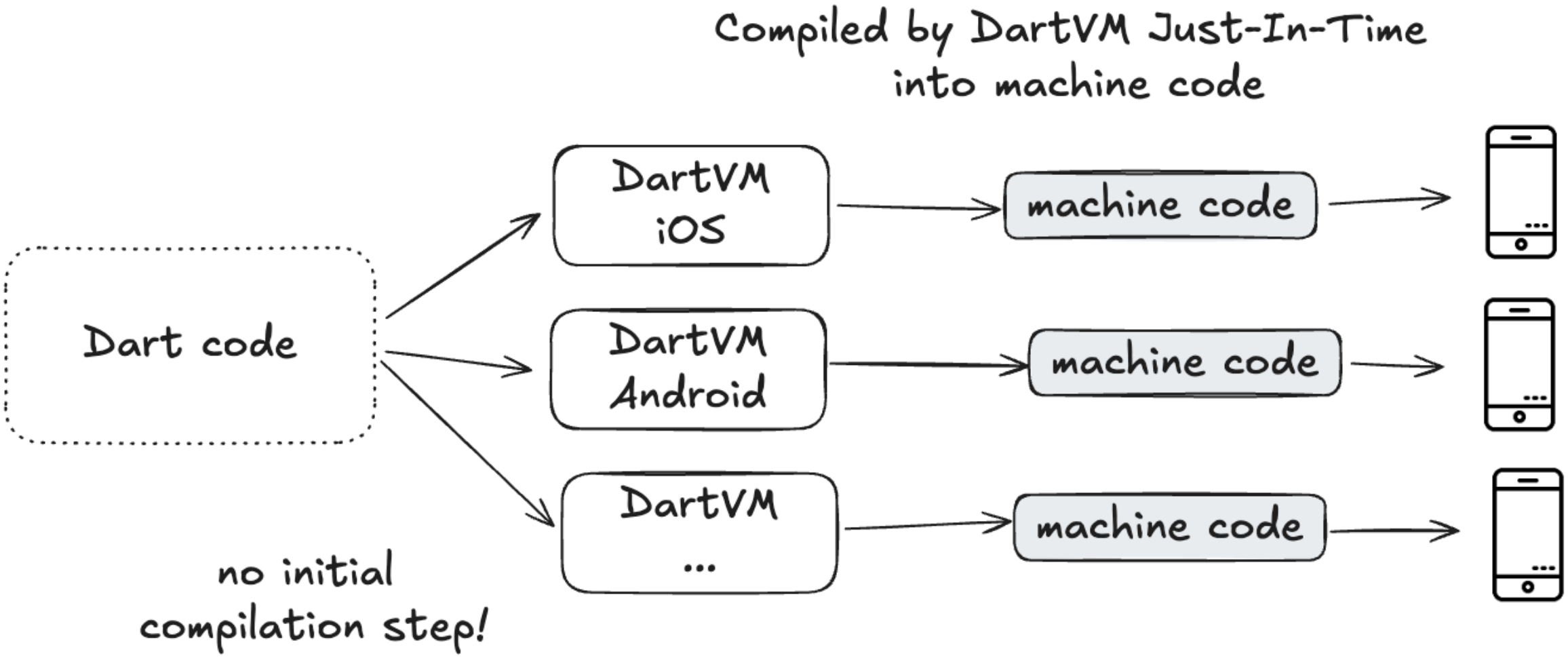
Runtime systems like Garbage Collection

Development experience components like debugging and hot-reload.

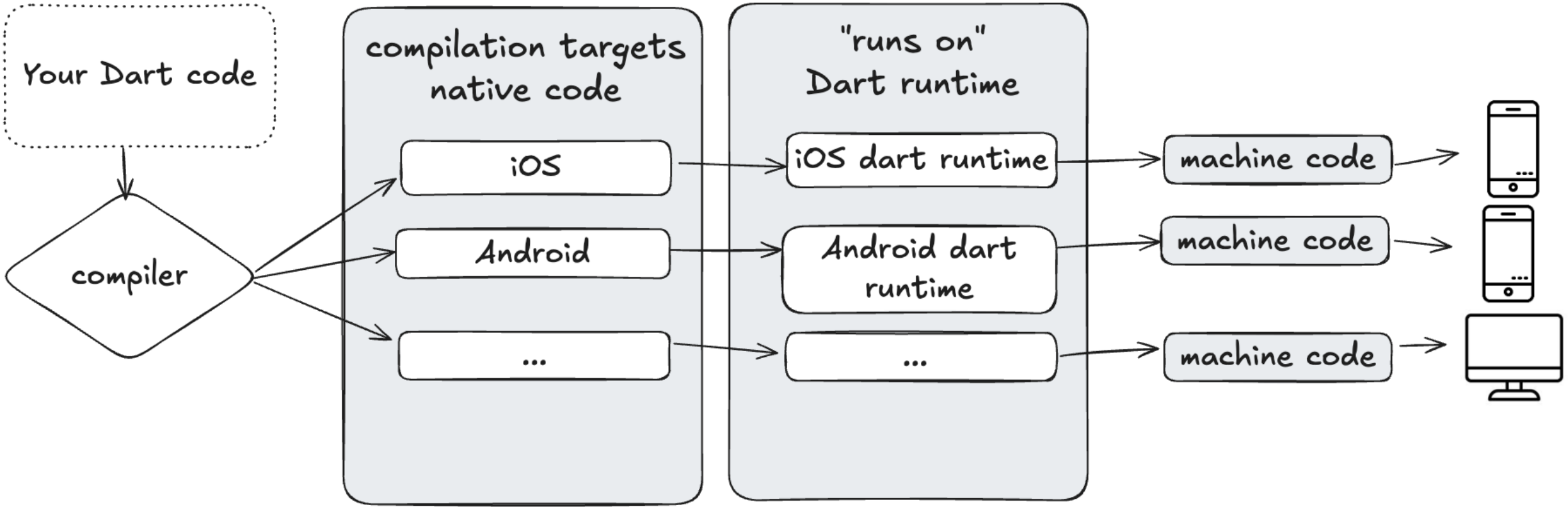
Just-in-Time (JIT) and Ahead-of-Time (AOT) compilers.


Java code
compiler
bytecode
(.class)
JVM
(windows)
JVM (Linux)
JVM
(MacOS)
ART
(Android)
machine code
machine code
machine code
machine code

Dart (Just-in-Time compilation)

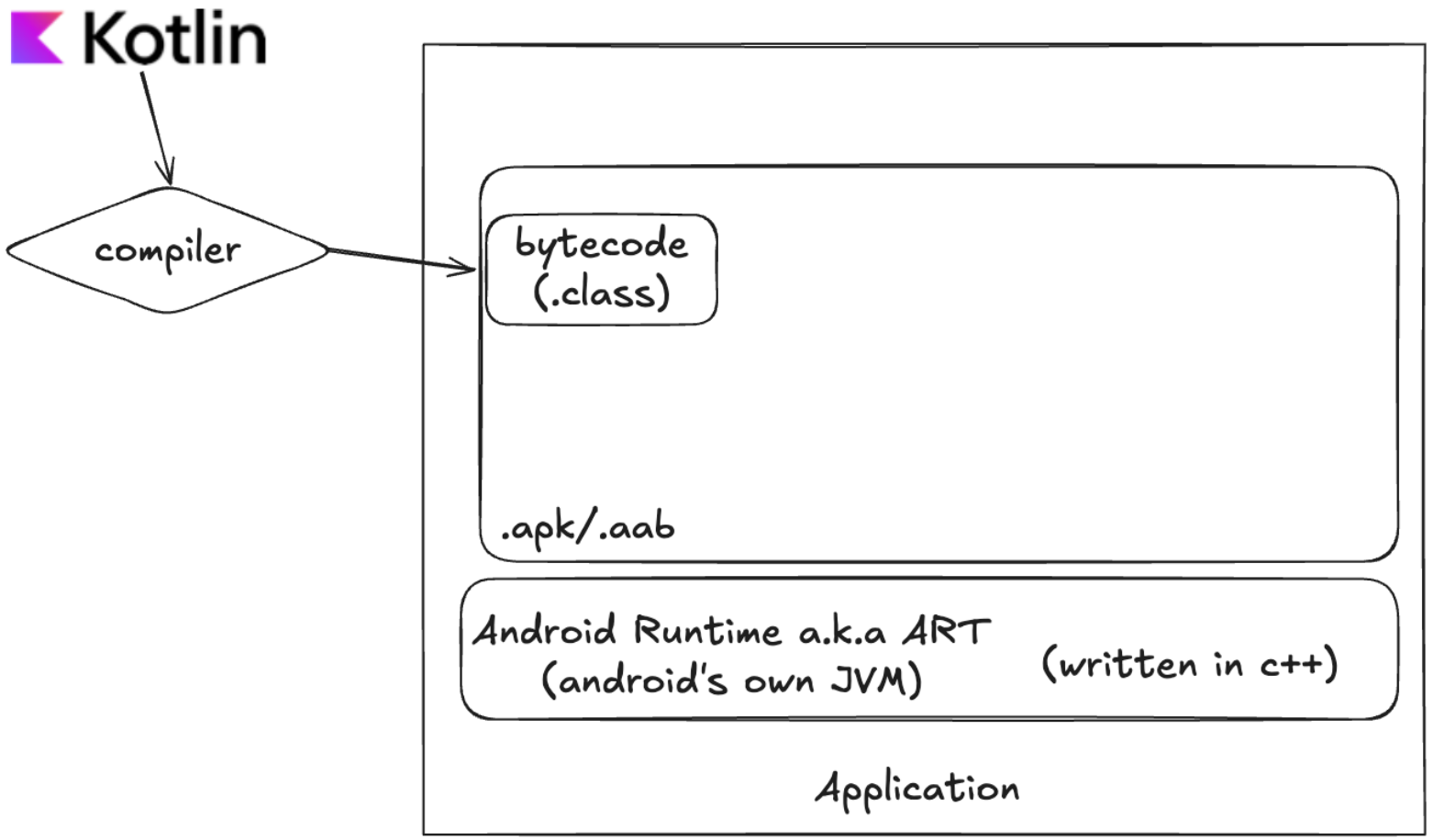


Dart (AOT compilation)

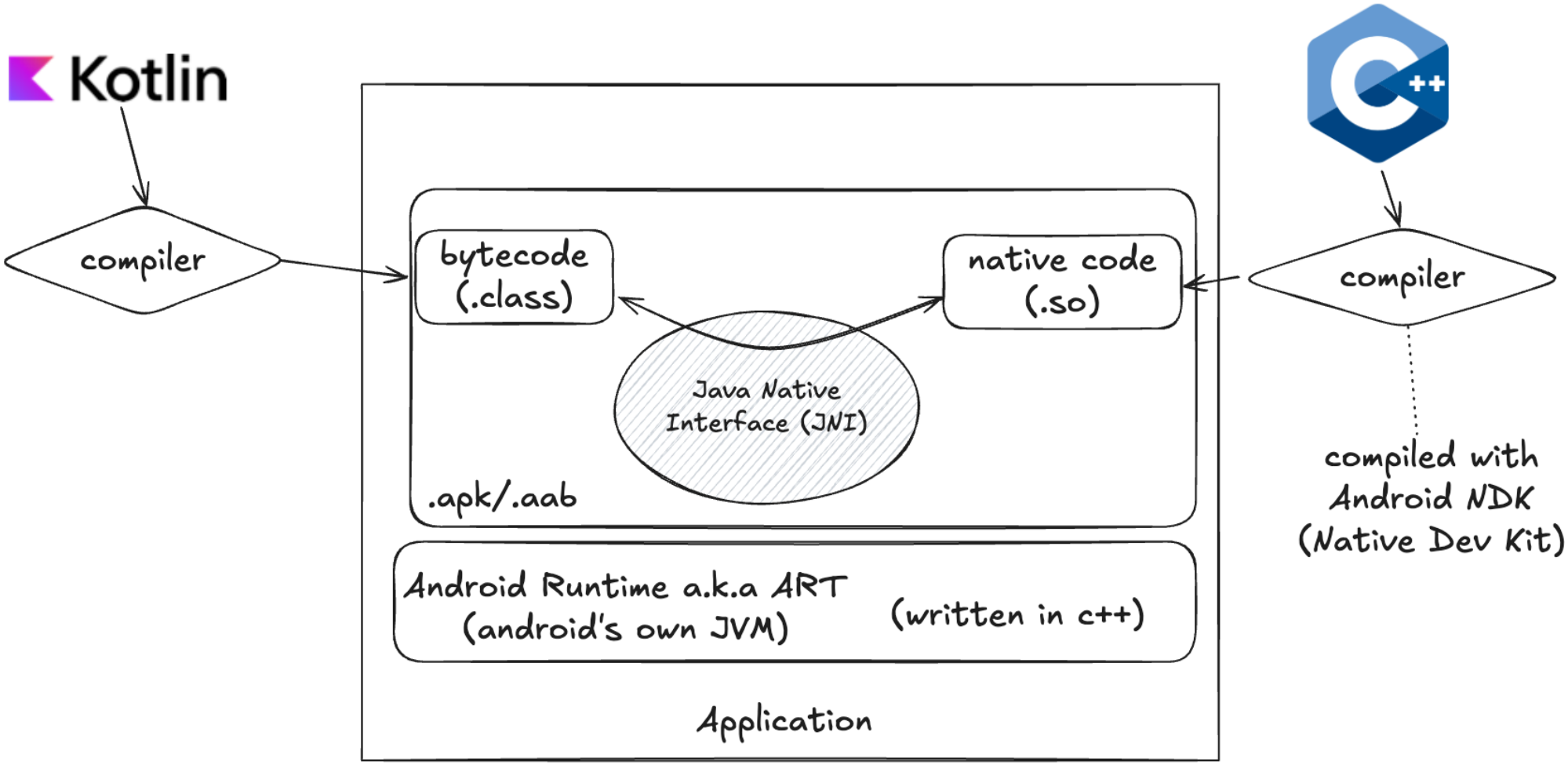


.NET is very similar. Native code is included in .dll files and runs on the .NET Runtime on each platform

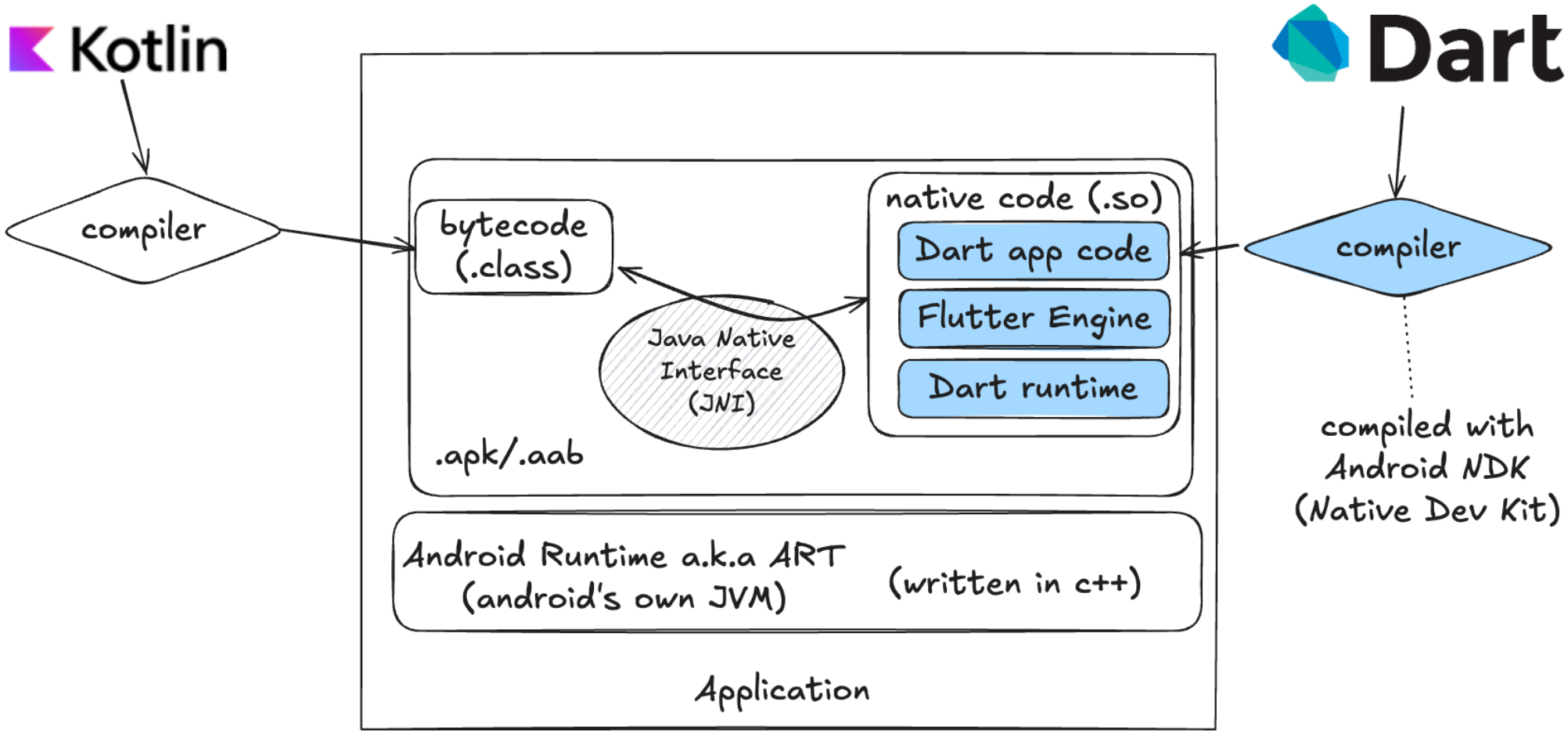
How a native Android app is packaged



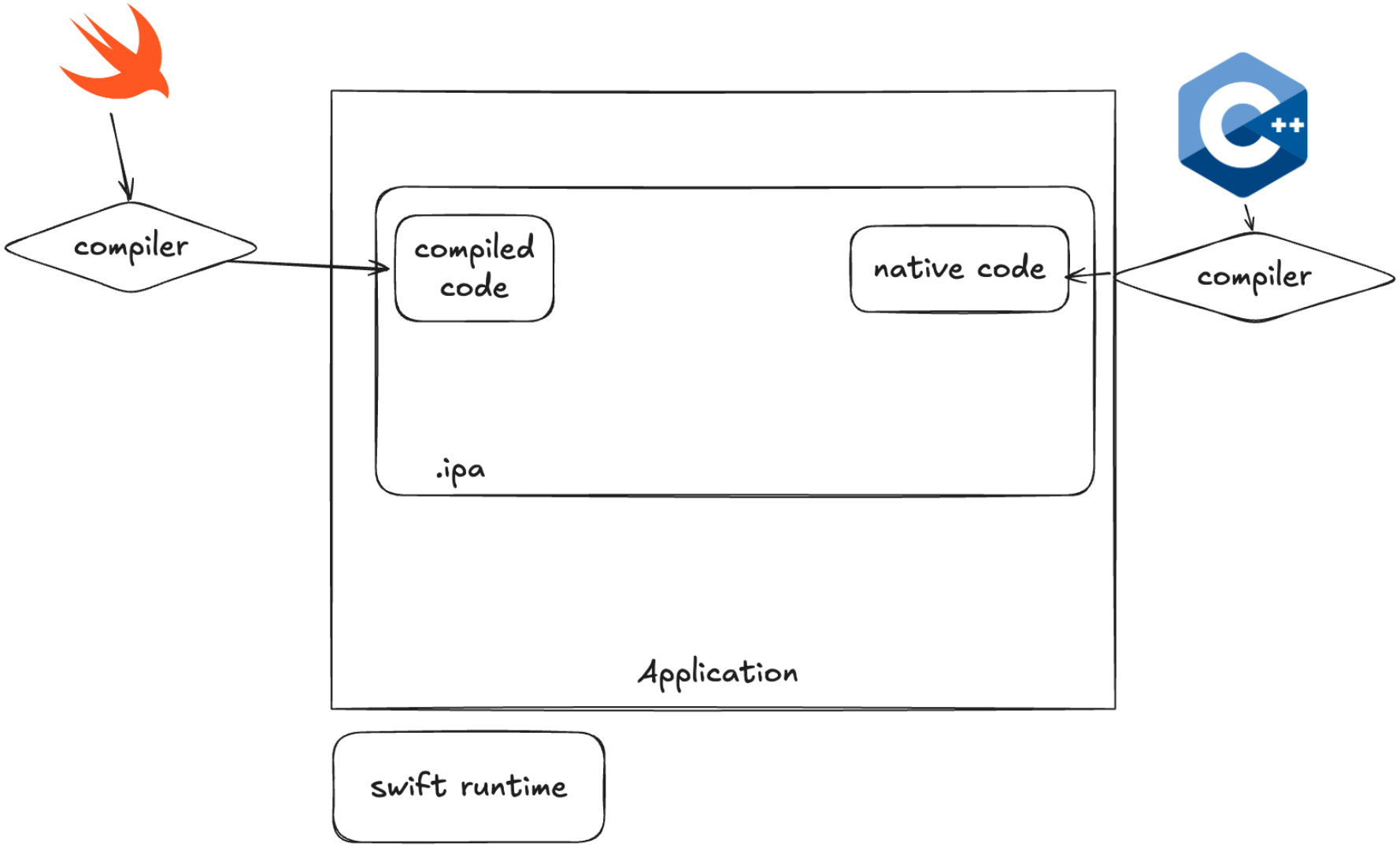
How a native Android app is packaged



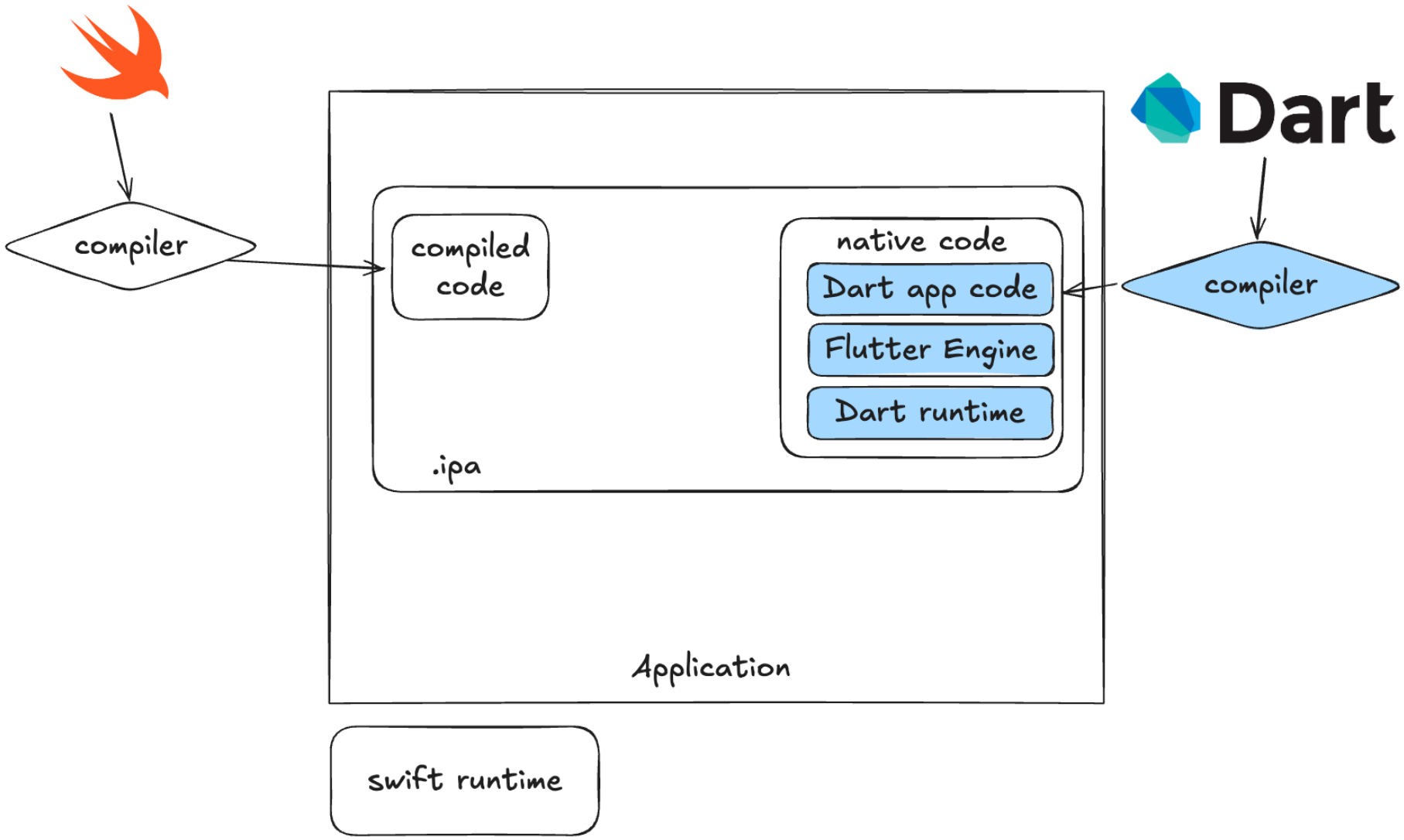
How a Flutter Android app is packaged



How a native iOS app is packaged



How a Flutter iOS app is packaged

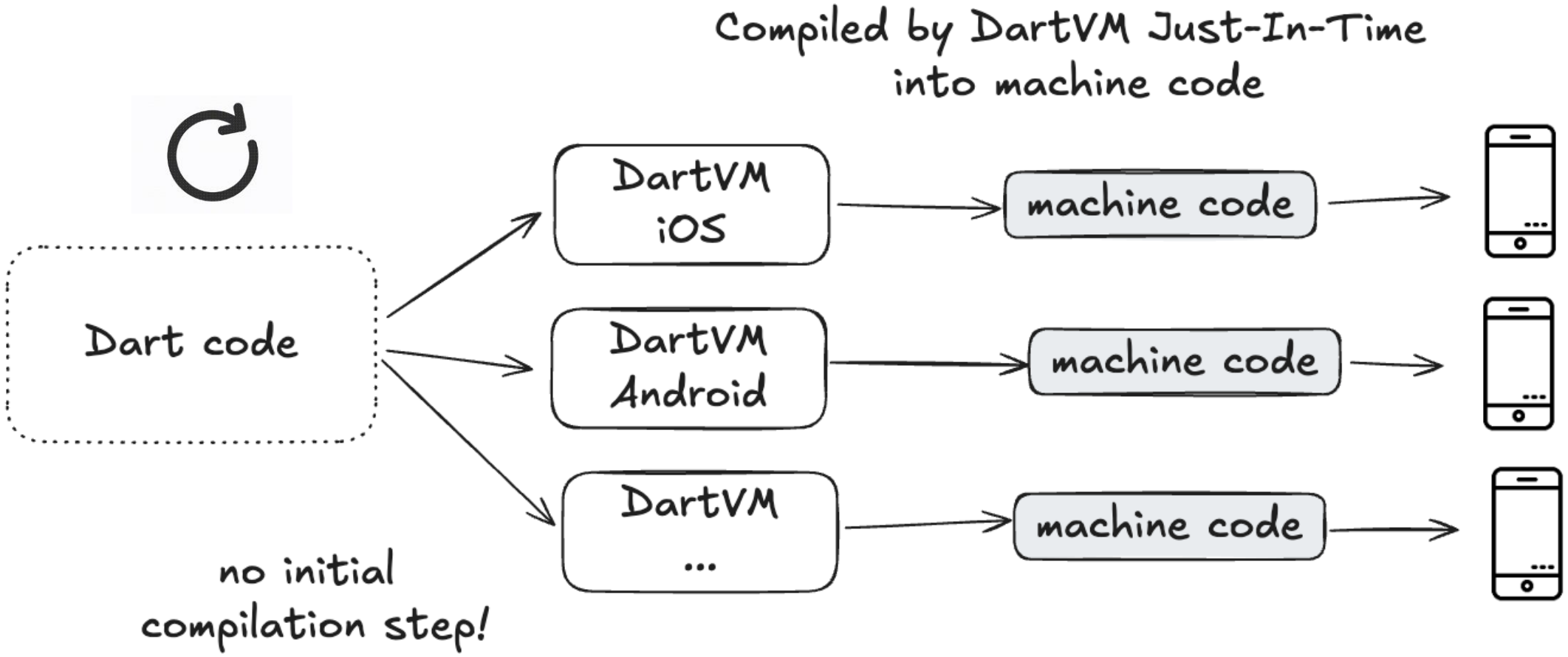


Dev Experience

Hot reload

Hot reload

Takes advantage from the VMs / runtimes ability to dynamically load code with Just-in-time compilation (JIT)



Programming language challenges

Threading

Threading

→ Dart and Typescript are single-threaded languages

→ Dart offers “isolates”, which is almost like another process. Memory isn’t shared, data needs to be serialized.

→ If parallelism is absolutely required, you can leverage the feature with a native module. (and code that part in Swift+Kotlin)

Not a big deal for most apps.

Unique features

Code push

→ Ability to update your app over-the-air bypassing the app stores

→ React-Native via Expo EAS (Expo Application Services) ($) or self-hosted

→ Flutter via Shorebird ($).

Some limitations on what code you can update. Generally, access to native APIs can’t change.

Typically initiated on app launch

Desktop and Web

- → .NET (Uno Platform), Flutter and React-Native all offer desktop and web support.
- → Same architecture pattern as iOS and Android.
- → Flutter and .NET can run as “native code” in the browser through WebAssembly.

How to choose?

Factors to consider

How to choose?

→ Try them for at least a few days with your team and see how you appreciate the developer experience.

→ What is your current team’s ecosystem?

→ Are the products/libraries you want to include offering a SDK for the mobile framework you want to pick?

- You can create the ”bindings” yourself, but it can be an adventure depending on the library

→ Do you need heavy parallelism / multi-threading?

Ask me two questions

Session feedback

Merci!
Thank you!

Some sources

- → https://reactnative.dev/docs/turbo-native-modules-introduction
- → https://reactnative.dev/blog/2024/10/23/the-new-architecture-is-here
- → React Conf keynote day2 https://www.youtube.com/watch?v=Q5SMmKb7qVI
- → DartVM : https://mrale.ph/dartvm/
- → Flutter: https://docs.flutter.dev/resources/architectural-overview