

**HE<sup>VD</sup>  
IG**

**HAUTE ÉCOLE  
D'INGÉNIERIE  
ET DE GESTION  
DU CANTON  
DE VAUD**



# Développement mobile avancé

Native Development Kit

Fabien Dutoit

# Native Development Kit – NDK



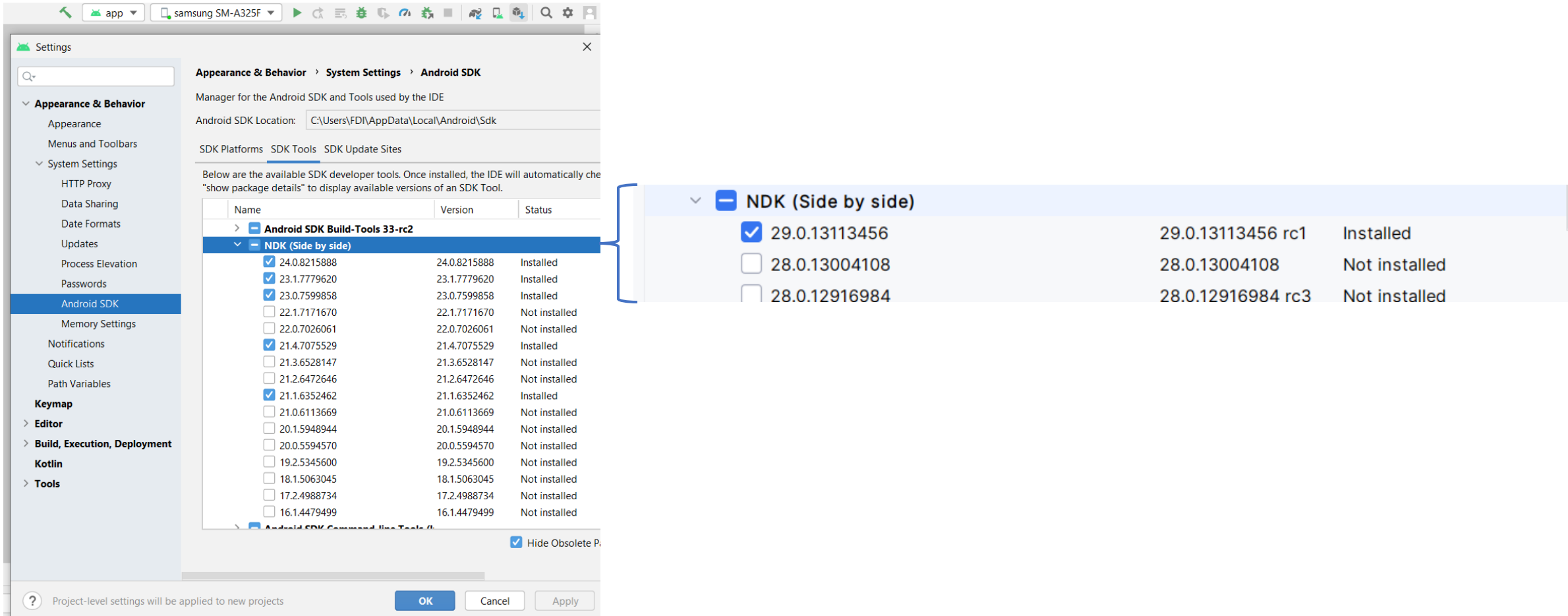
# Native Development Kit – NDK

- Le *NDK* est une suite d'outils destinée à permettre l'utilisation de code C et C++ sur *Android*
- Le code est cross-compilé sous la forme de fichiers .so (dynamically linked shared libraries), pour différentes architectures de processeurs
- L'utilisation du NDK **n'est pas** une alternative au *SDK Kotlin / Java* !  
Son utilisation est réservée à certains cas bien particuliers :
  - Nécessité d'avoir des performances maximales pour des applications intenses en calculs, par exemple le traitement d'images, la compression vidéo, un moteur de jeu vidéo, les réseaux de neurones, etc...
  - L'utilisation de code ou de librairies uniquement disponibles en C ou C++



# Native Development Kit – Installation

- L'installation du *NDK* est gérée par *Android Studio* :



The screenshot shows the 'Android SDK' settings in Android Studio. The 'NDK (Side by side)' section is expanded, showing a list of NDK versions. A blue bracket highlights the 'NDK (Side by side)' section, and a callout box shows a detailed view of the NDK versions.

Version	Name	Status
29.0.13113456	29.0.13113456 rc1	Installed
28.0.13004108	28.0.13004108	Not installed
28.0.12916984	28.0.12916984 rc3	Not installed

# Native Development Kit – Contenu

- Le *NDK* utilise la collection d'outils *LLVM* pour build le code C/C++
- Cela inclut principalement :
  - *CLANG* pour la compilation
  - *LLD* pour le linkage
- Des librairies :
  - la *STL* par la librairie *LLVM's libc++*
    - Supporte C++14 (défaut), C++17, et partiellement C++20
    - Disponible sous la forme d'une librairie partagée (défaut) et d'une librairie statique
  - Plusieurs API *Android* spécifiques, quelques exemples :
    - `#include <android/log.h>`
    - `#include <android/sensor.h>`
    - `#include <android/asset_manager.h>`
    - `#include <android/storage_manager.h>`
    - `#include <android/permission_manager.h> //API 31`

# Native Development Kit – Contenu

- *Android* supporte plusieurs architectures de processeurs, le *NDK* va devoir compiler le code natif pour chacune de ces architectures :

Name	arch	ABI	triple
32-bit ARMv7	arm	armeabi-v7a	arm-linux-androideabi
64-bit ARMv8	aarch64	aarch64-v8a	aarch64-linux-android
32-bit Intel	x86	x86	i686-linux-android
64-bit Intel	x86_64	x86_64	x86_64-linux-android



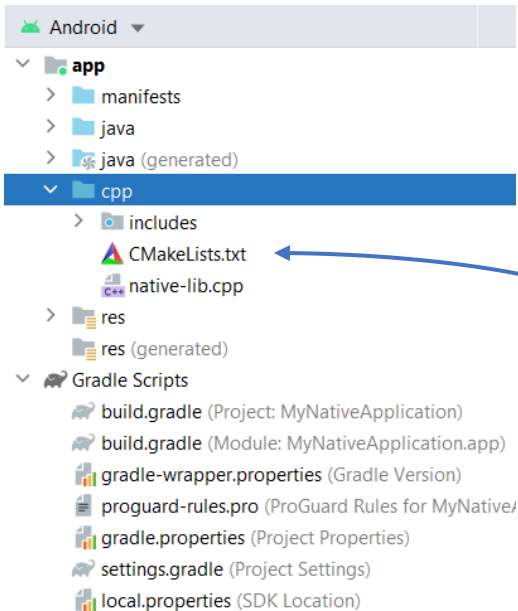
# Native Development Kit – Intégration

- L'utilisation du *NDK* peut se faire selon 2 approches :
  - *JNI (Java Native Interface)* : le code *Kotlin / Java* peut appeler des méthodes natives
  - L'utilisation de la *native-activity* : l'activité est entièrement codée en *C/C++* et utilise *OpenGL ES* pour le rendu de l'*UI*. Cette approche est principalement destinée à la réalisation de jeux vidéo
- L'intégration du code natif est réalisée au travers d'un «*makefile Android*» qui va lister les fichiers sources et configurer les options de compilation et de linkage
- L'utilisation de *CMake* est une alternative encouragée (disponible dans les outils du *SDK*), celle-ci est préférable pour du code utilisé sur plusieurs plateformes (*Android, iOS, Windows, Linux, Mac*)

▼	<input checked="" type="checkbox"/> CMake		
	<input checked="" type="checkbox"/> 3.31.6	3.31.6	Installed
	<input type="checkbox"/> 3.31.5	3.31.5	Not installed
	<input type="checkbox"/> 3.31.4	3.31.4	Not installed

# Native Development Kit – Intégration – Exemple JNI

Création d'un dossier  
*cpp* dans *src/main* :



Celui-ci va contenir  
notre fichier *CMake* et  
les sources natives  
(.h, .c, .cpp, etc.)

On doit faire référence au fichier *CMake* dans  
le fichier *gradle* du module *app* :

```
android {
    // [...]
    externalNativeBuild {
        cmake {
            path file('src/main/cpp/CMakeLists.txt')
            version "3.31.6"
        }
    }
}
```

L'IDE va parser le fichier *CMake* et afficher les  
sources référencées dans la structure du projet

# Native Development Kit – Intégration – Exemple JNI

*# For more information about using CMake with Android Studio, read the documentation: <https://d.android.com/studio/projects/add-native-code.html>*

*# Sets the minimum version of CMake required to build the native library.*

```
cmake_minimum_required(VERSION 3.22.1)
```

```
set(CMAKE_CXX_STANDARD 20)
```

```
set(CMAKE_CXX_FLAGS_RELEASE "${CMAKE_CXX_FLAGS_RELEASE} -O3")
```

Options de compilation

*# Creates and names a library, sets it as either STATIC or SHARED, and provides the relative paths to its source code.*

*# You can define multiple libraries, and CMake builds them for you. Gradle automatically packages shared libraries with your APK.*

```
add_library( # Sets the name of the library.
```

```
native-lib
```

```
  # Sets the library as a shared library.
```

```
  SHARED
```

```
  # Provides a relative path to your source file(s).
```

```
  native-lib.cpp)
```

Nom de notre librairie partagée

Liste des fichiers sources à compiler, ici un seul

*# Searches for a specified prebuilt library and stores the path as a variable. Because CMake includes system libraries in the search path by*

*# default, you only need to specify the name of the public NDK library you want to add. CMake verifies that the library exists before completing its build.*

```
find_library( # Sets the name of the path variable.
```

```
log-lib
```

```
  # Specifies the name of the NDK library that you want CMake to locate.
```

```
log)
```

On veut utiliser la librairie log du NDK

*# Specifies libraries CMake should link to your target library. You can link multiple libraries, such as libraries you define in this*

*# build script, prebuilt third-party libraries, or system libraries.*

```
target_link_libraries( # Specifies the target library.
```

```
native-lib
```

```
  # Links the target library to the log library included in the NDK.
```

```
  ${log-lib})
```

On spécifie comment linker les différents composants pour notre librairie

# Native Development Kit – Intégration – Exemple JNI

Dans le code *Kotlin*, on va définir une fonction **external**

Par exemple dans notre *Activité* :

```
private external fun nativeSum(v1: Int, v2: Int) : Int
```

Cette fonction n'a pas de corps, il s'agit juste d'une signature, elle sera implémentée dans le code natif

L'IDE vérifie qu'il trouve son implémentation, si ce n'est pas le cas il proposera de créer automatiquement son squelette dans les sources C++

Dans le code *natif* (native-lib.cpp) :

```
#include <jni.h>

extern "C"
JNIEXPORT jint JNICALL
Java_ch_heigvd_iict_dma_mynativeapplication_MainActivity_nativeSum(
    JNIEnv *env,
    jobject thiz,
    jint v1,
    jint v2) {
    return v1 + v2;
}
```

Type de retour

Environnement *JNI*

Objet *Java* déclarant la méthode native, ici l'*Activité*

Paramètres de la méthode

# Native Development Kit – Intégration – Exemple JNI

```
class MainActivity : AppCompatActivity() {
```

```
    companion object {
```

```
        init {
```

```
            System.loadLibrary("native-lib")
```

```
        }
```

```
    }
```

```
    private external fun nativeSum(v1: Int, v2: Int) : Int
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        // [...]
```

```
        val un = 1
```

```
        val deux = 2
```

```
        val resultat = nativeSum(un, deux)
```

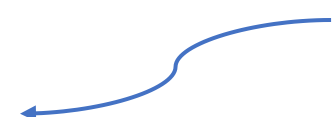
```
    }
```

```
}
```

La librairie native doit être chargée



La méthode native peut être appelée depuis *Kotlin*  
En général on évitera de le faire depuis le *UI-Thread*



# Native Development Kit – JNI – Types primitifs

- Nous avons vu dans le précédent exemple que les paramètres de la fonction externe *Kotlin*, des `Int`, sont devenus des `jint` dans son implémentation en C++
- Le *NDK* propose un ensemble de types équivalents aux types primitifs *Java*, qui correspondent eux-mêmes à des types *Kotlin*

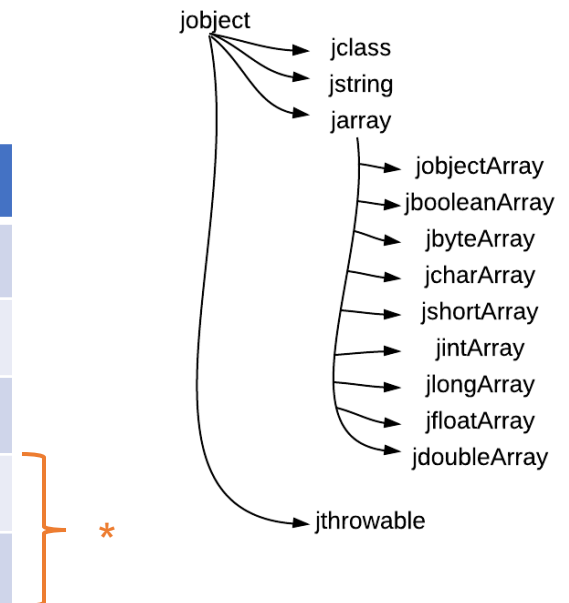
Java	JNI	Description	Kotlin
boolean	jboolean	1 byte, unsigned	Boolean
int	jint	4 bytes, signed	Int
long	jlong	8 bytes, signed	Long
short	jshort	2 bytes, signed	Short
byte	jbyte	1 byte, signed	Byte
char	jchar	2 bytes, unsigned	Char
float	jfloat	4 bytes	Float
double	jdouble	8 bytes	Double
void	void	N/A	Unit



# Native Development Kit – JNI – Types avancés

- La gestion des types «complexes», soit des objets *Kotlin / Java*, va demander quelques précautions, on souhaitera la plupart du temps les convertir vers des types C++
- Quelques exemples :

Java	JNI	Kotlin
Object	jobject	Any
String	jstring	String
Class	jclass	Class<>
int[]	jintArray	IntArray
double[]	jdoubleArray	DoubleArray



- \* En *Kotlin* pour réaliser un tableau d'entiers, nous utiliserons en général le type `Array<Int>`, celui-ci correspond en *Java* à `Integer[]` (tableau d'objets). Il existe un autre type, le `IntArray` qui correspond à `int[]` (tableau primitif)

# Native Development Kit – JNI – Types avancés

- La conversion vers des types «standards» C++, quelques exemples :
  - Un tableau d'entier (jintArray ↔ std::vector<int>)
    - Seuls les conteneurs doivent être convertis, pas le contenu (jint ≡ int)

*// conversion du tableau du type jintArray en vector*

```
inline std::vector<int> convert(JNIEnv* env, const jintArray &values) {  
    jsize arrayLength = env->GetArrayLength(values);  
    jint* arrayContent = env->GetIntArrayElements(values, nullptr);  
  
    std::vector<int> v(arrayLength);  
    for(int i = 0; i < arrayLength; ++i)  
        v[i] = arrayContent[i];  
  
    return v;  
}
```

*// conversion du vector vers un tableau du type jintArray*

```
jintArray convert(JNIEnv* env, const std::vector<int> &values) {  
    jintArray arr = env->NewIntArray(values.size());  
    env->SetIntArrayRegion(arr, 0, values.size(), (jint*) &values[0]);  
    return arr;  
}
```

# Native Development Kit – JNI – Types avancés

- La conversion vers des types «standards» C++, quelques exemples :
  - Une chaîne de caractères (jstring ↔ std::string), un peu plus compliqué...
    - En *Kotlin / Java*, les *Strings* sont codées en *UTF-16*, en C++ elles le sont en *UTF-8*

[//https://stackoverflow.com/a/41820336](https://stackoverflow.com/a/41820336)

```
std::string jstring2string(JNIEnv* env, jstring &jStr) {
    if (!jStr) return "";
```

```
    const jclass stringClass = env->GetObjectClass(jStr);
    const jmethodID getBytes = env->GetMethodID(stringClass, "getBytes", "(Ljava/lang/String;)[B");
    const jbyteArray stringJbytes = (jbyteArray) env->CallObjectMethod(jStr, getBytes, env->NewStringUTF("UTF-8"));
```

```
    size_t length = (size_t) env->GetArrayLength(stringJbytes);
    jbyte* pBytes = env->GetByteArrayElements(stringJbytes, NULL);
    std::string ret = std::string((char *)pBytes, length);
```

```
    env->ReleaseByteArrayElements(stringJbytes, pBytes, JNI_ABORT);
    env->DeleteLocalRef(stringJbytes);
    env->DeleteLocalRef(stringClass);
```

```
    return ret;
```

```
}
```

On appelle la méthode *getBytes(String charsetName)* sur la string *Java* par réflexion, depuis le C++



# NDK – JNI – Exemple d'application 1

- Code natif pour le tri d'un tableau d'entiers

```
#include <jni.h>
#include <android/log.h>
#include <algorithm>
#include <vector>

// conversion d'un tableau du type jintArray en vector
std::vector<int> convert(JNIEnv *env, const jintArray &values)
// conversion d'un vector en un tableau du type jintArray
jintArray convert(JNIEnv *env, const std::vector<int> &values)

extern "C"
JNIEXPORT jintArray JNICALL
Java_ch_heigvd_iict_dma_mynativeapplication_MainActivity_nativeSort(JNIEnv* env, jobject thiz, jintArray values) {
    // affichage dans le logcat
    __android_log_print(ANDROID_LOG_INFO, "NativeLib", "native sort called\n");

    std::vector<int> v = convert(env, values);
    sort(v.begin(), v.end());
    return convert(env, v);
}
```

# NDK – JNI – Exemple d'application 1

- Codes pour lancer le tri en *Kotlin* et en natif dans des Coroutines

```
private suspend fun sortAndDisplayRandomValuesKotlin(numberOfValues: Int) = withContext(Dispatchers.Default) {
    val unsorted = generatedRandomValues(numberOfValues) //retourne un IntArray de n entiers aléatoires
    val timeInNano = measureNanoTime {
        val sorted = unsorted.sorted()
        values.postValue(sorted)
    }
    Log.d("MainActivity", "sortAndDisplayRandomValuesKotlin($numberOfValues): ${timeInNano/1000000}")
}
```

Utilisation de la méthode de tri de *Kotlin* sur un IntArray

```
private suspend fun sortAndDisplayRandomValuesNative(numberOfValues: Int) = withContext(Dispatchers.Default) {
    val unsorted = generatedRandomValues(numberOfValues) //retourne un IntArray de n entiers aléatoires
    val timeInNano = measureNanoTime {
        val sorted = nativeSort(unsorted)
        values.postValue(sorted.asList())
    }
    Log.d("MainActivity", "sortAndDisplayRandomValuesNative($numberOfValues): ${timeInNano/1000000}")
}
```

Appel du code natif C++ pour le tri


# NDK – JNI – Exemple d'application 1

- Test des deux méthodes de tri, sur 1'000'000 d'entiers aléatoires :

```
lifecycleScope.launch(Dispatchers.Default) {
    val n = 1000000
    sortAndDisplayRandomValuesKotlin(n)
    delay(2000)
    sortAndDisplayRandomValuesNative(n)
}
```


- Exécution sur un *Samsung Galaxy A32*

D/MainActivity: sortAndDisplayRandomValuesKotlin(1000000): 1467  
I/NativeLib: native sort called  
D/MainActivity: sortAndDisplayRandomValuesNative(1000000): 431



÷ 3.5

D/MainActivity: sortAndDisplayRandomValuesKotlin(1000000): 1388  
I/NativeLib: native sort called  
D/MainActivity: sortAndDisplayRandomValuesNative(1000000): 99



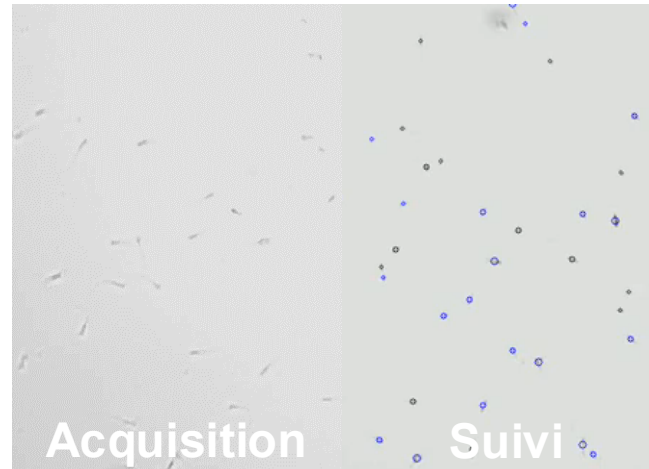
÷ 14

← Optimisation -03



## NDK – Exemple d'application 2

- Système d'analyse portable de la fertilité pour des applications principalement vétérinaires
- Une application *Android* accompagne un dispositif permettant l'acquisition vidéo d'échantillons de semence
- L'application comporte 3 modules natifs (*NDK*) :
  - Le «pilote» *USB* pour la caméra du dispositif
  - L'intégration de *FFMPEG* pour la compression / décompression des vidéos
  - Les algorithmes d'analyse vidéos des échantillons, utilisation de la librairie *OpenCV*. Les algorithmes sont utilisables également sur *Mac*, *Windows*, et *iOS*



# Native Development Kit – Optimisation

- Attention au niveau d'optimisation, -Oz est conseillé (taille + vitesse)
- L'*APK* va contenir 4 copies de toutes les librairies, cela peut prendre **beaucoup** de place...

C:\Users\fabie\Desktop\app-release-0.3.2.apk\		
Name	Size	Packed Size
assets	92 327	29 732
kotlin	24 359	9 268
lib	242 547 440	242 547 440
META-INF	350	442
res	1 498 955	776 179
AndroidManifest.xml	5 644	1 671
classes.dex	7 819 856	3 180 207
classes2.dex	4 347 424	1 726 209
resources.arsc	1 186 608	1 186 608

Name	Size	Packed Size
arm64-v8a	45 424 488	45 424 488
armeabi-v7a	39 807 624	39 807 624
x86	68 761 488	68 761 488
x86_64	88 553 840	88 553 840

Name	Size	Packed Size
libavcodec.so	22 246 576	22 246 576
libavdevice.so	64 200	64 200
libavfilter.so	4 027 208	4 027 208
libavformat.so	5 270 504	5 270 504
libavutil.so	404 504	404 504
libc++_shared.so	989 800	989 800
libjpeg-turbo1500.so	526 704	526 704
libmobilefmpeg.so	428 904	428 904
libmobilefmpeg_abidetec.so	30 992	30 992
libnative-lib.so	1 161 736	1 161 736
libopencv_java4.so	52 187 432	52 187 432
libswresample.so	334 824	334 824
libswscale.so	485 376	485 376
libusb100.so	108 752	108 752
libuvc.so	108 992	108 992
libUVCCamera.so	177 336	177 336

**HE<sup>VD</sup>  
IG**

**HAUTE ÉCOLE  
D'INGÉNIERIE  
ET DE GESTION  
DU CANTON  
DE VAUD**



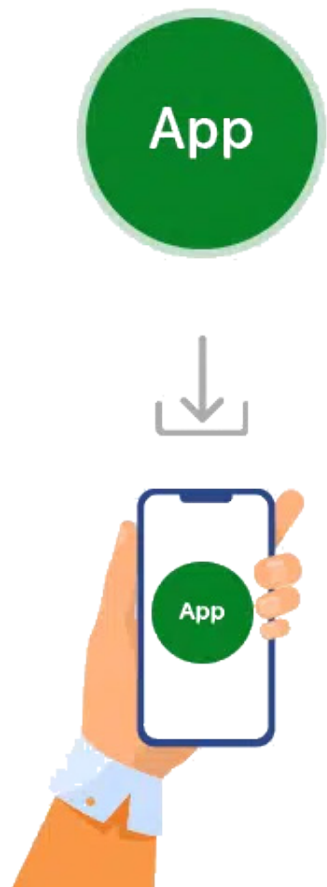


# Développement mobile avancé

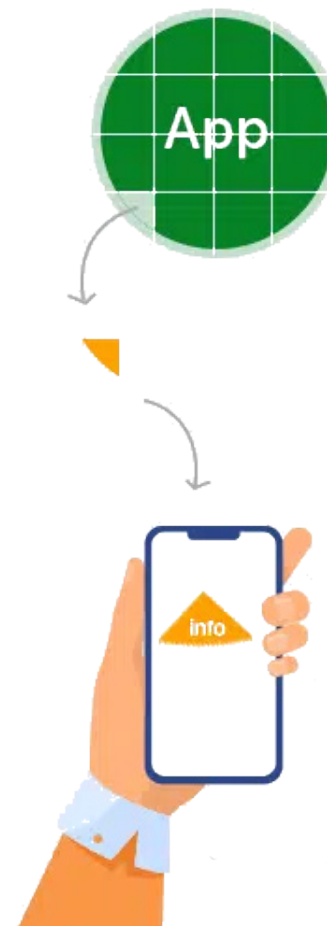
Approches alternatives au développement natif  
*Instant app et cross-plateformes*

Fabien Dutoit

# Instant App et App Clips



Installation traditionnelle  
d'une app depuis un store



Le module utile est chargé à l'utilisation et  
disparaîtra lorsque l'utilisateur a terminé



# *Instant App et App Clips*

- Il existe de nombreuses situations dans lesquelles on a besoin d'une app pour accéder à un service :
  - Louer un vélo
  - Payer une place de parking
  - Passer commande dans un restaurant
  - Configurer un appareil connecté, etc.
- Il est probable que l'utilisation de ce service soit ponctuelle et que l'app, une fois installée, ne soit jamais ouverte à nouveau...
  - 24% des apps installées ne sont ouvertes qu'une seule fois
  - Raisons principales à la désinstallation d'une app :
    - 39.9% Pas utilisée
    - 18.7% Besoin de récupérer de l'espace de stockage

# *Instant App et App Clips*

- Depuis octobre 2017, *Android* met à disposition les *Instant Apps*
- *Apple* a sorti les *App Clips* en septembre 2020 avec *iOS 14*
- Il s'agit d'applications (ou de sous-parties d'applications) natives ne nécessitant pas d'installation préalable
  - Elles se téléchargent et s'ouvrent automatiquement (clic sur un lien, lecture d'un code-barres ou d'un tag *NFC*)
  - Lorsque la fonctionnalité a été utilisée, l'app se désinstalle
- Etant natives, elles peuvent offrir des fonctionnalités assez avancées
  - L'objectif est de permettre à l'utilisateur d'effectuer sa tâche en quelques secondes
- Toutefois, sur les deux plateformes, ces apps «streamées» tournent dans une sandbox avec des restrictions plus importantes qu'une application «normale»

# *Instant App et App Clips*

- Une autre utilisation possible des *Instant Apps* et des *App Clips* est de permettre aux potentiels utilisateurs de tester notre app avant de l'installer
- Sur *Android* :
  - Une app peut contenir plusieurs points d'entrées *Instant App*
    - *Utilisation du deep linking*
  - Chaque *Instant App* a une taille maximale de 15 Mo
  - Depuis une *Instant App*, on peut proposer à l'utilisateur d'installer l'app complète. Dans ce cas les données créées dans la sandbox seront transférées à l'app installée
- Jusqu'à fin 2020, *Android* étant seul à proposer cette approche, très peu d'applications ont fait l'effort de l'intégrer (nécessite de réorganiser l'app en modules distincts). Principalement des versions d'évaluation de jeux mobiles
- Depuis 2022, il semble que l'utilisation d'*App Clips* et *Instant App* commence à se développer un peu, malgré la pandémie. A voir pour la suite...

# Cross-plateformes

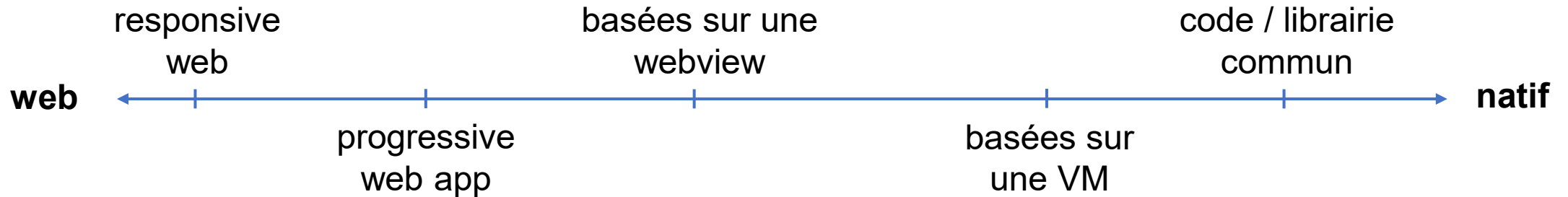


# Cross-plateformes

- Le développement sur les deux principales plateformes mobiles (*iOS* et *Android*) passe par des outils totalement différents :
  - IDE *Android Studio* ou *Xcode*
  - Langage *Kotlin/Java* ou *Swift/Objective-C*
  - SDK *Android SDK* ou *iOS SDK*
- Le fonctionnement des deux plateformes, ainsi que leurs «philosophies» sont très différents (ouverture, personnalisation, design, navigation, animations, etc.)
- Pour développer une app native sur *Android* et sur *iOS* il sera nécessaire d'avoir deux équipes distinctes développant et maintenant deux fois la même application, en parallèle. Cela représente des coûts substantiels...
  - Il existe donc, depuis très longtemps, un fort intérêt pour le développement cross-plateformes : un seul code compatible avec plusieurs plateformes

# Cross-plateformes – Différentes approches

- Il existe une multitude d'approches et de technologies permettant de réaliser des applications mobiles cross-plateformes :



- Les premières approches cross-plateformes pour du développement mobile étaient très proches du web. La raison principale était pour que les développeurs existants, formés et habitués aux technologies web, puissent travailler directement sur des solutions mobiles
- Depuis, les approches cross-plateformes se rapprochent de plus en plus du natif afin de pouvoir bénéficier de meilleures performances, d'une meilleure expérience utilisateur et d'avoir accès à un plus large choix de fonctionnalités spécifiques au mobile



# Cross-plateformes – *Applications web (web app)*

- *Android* et *iOS* disposent de navigateurs web récents et aujourd'hui relativement homogènes dans l'interprétation de *HTML5*, *CSS* et *JavaScript*
- Il existe un grand nombre de frameworks permettant la réalisation d'interface responsives, adaptées aux écrans de smartphones
- Les principales librairies *JavaScript* sont disponibles dans tous les navigateurs mobiles, mais pas forcément les plus avancées...

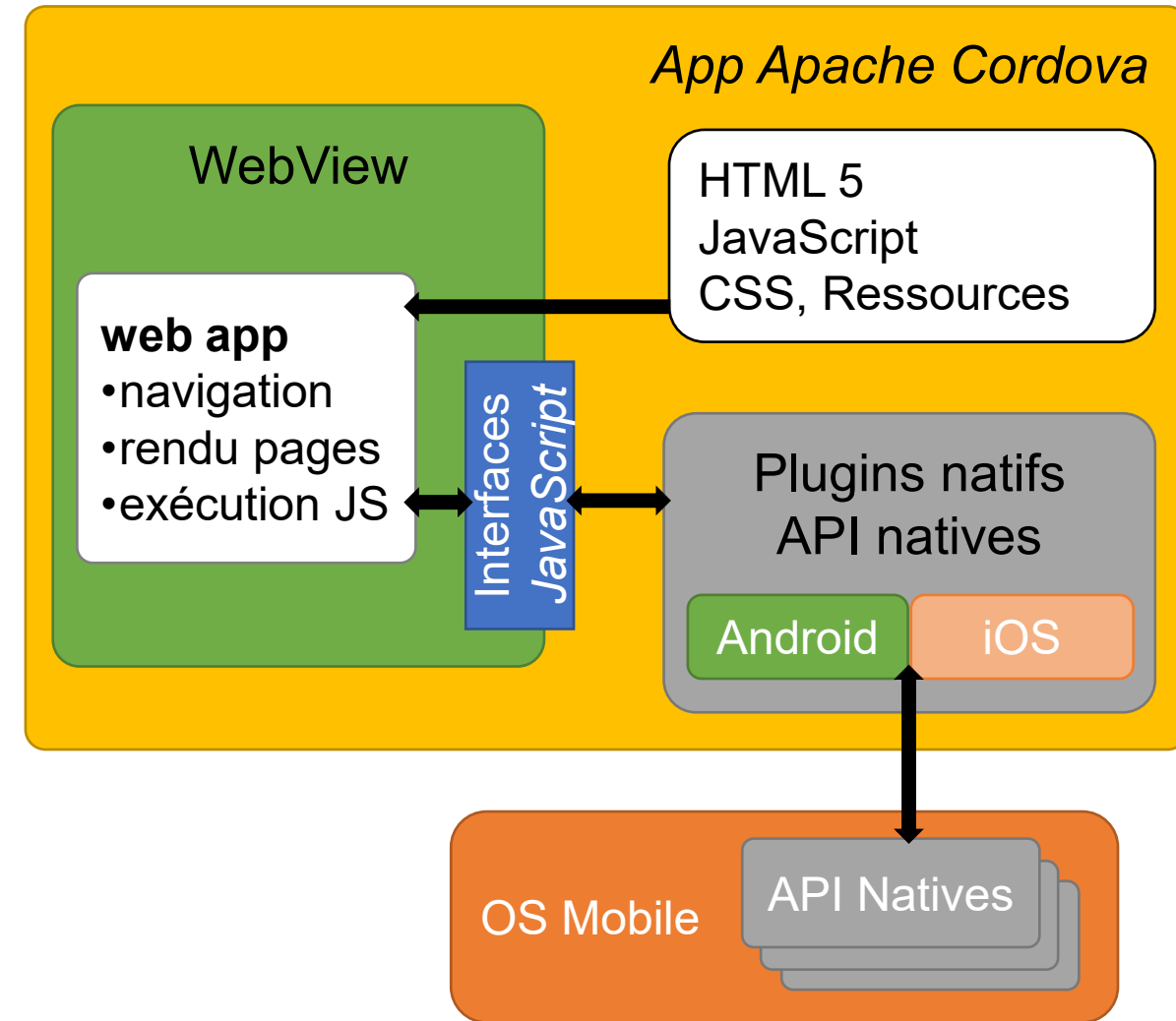
Librairie	Chrome <i>Android</i>	Firefox <i>Android</i>	WebView <i>Android</i>	Safari <i>iOS</i>	WebView <i>iOS</i>
<i>Storage</i>	✓	✓	✓	✓	✓
<i>Bluetooth</i>	✓	✗	✗	✗	✗
<i>Geolocation</i>	✓	✓	✓	✓	✓
<i>Sensor</i>	✓	✗	✓	✗	✗
<i>NDEFReader</i>	✓	✗	✓	✗	✗

# Cross-plateformes – *Progressive Web App (PWA)*

- Il s'agit principalement d'applications web accessibles depuis le navigateur du smartphone à une *url* spécifique et pouvant être «installées»
- Une *web app* peut être qualifiée de *PWA* lorsqu'elle satisfait le critère d'installation, c.-à-d. qu'elle peut fonctionner en l'absence de réseau et qu'un raccourci peut être installé sur le launcher, soit, au minimum :
  - Être accompagnée d'un *Manifest* (fichier *json* décrivant l'app), et
  - Utiliser un *service worker* (proxy permettant la mise en cache des fichiers de la *PWA*)
- Sur le *iOS* le support des *PWA* n'est pas total, mais cela s'améliore avec les versions successives :
  - Site d'un développeur tenant à jour un tableau précisant le support des *PWA* sur *iOS* :  
<https://firt.dev/notes/pwa-ios/>

# Cross-plateformes – *Apache Cordova*

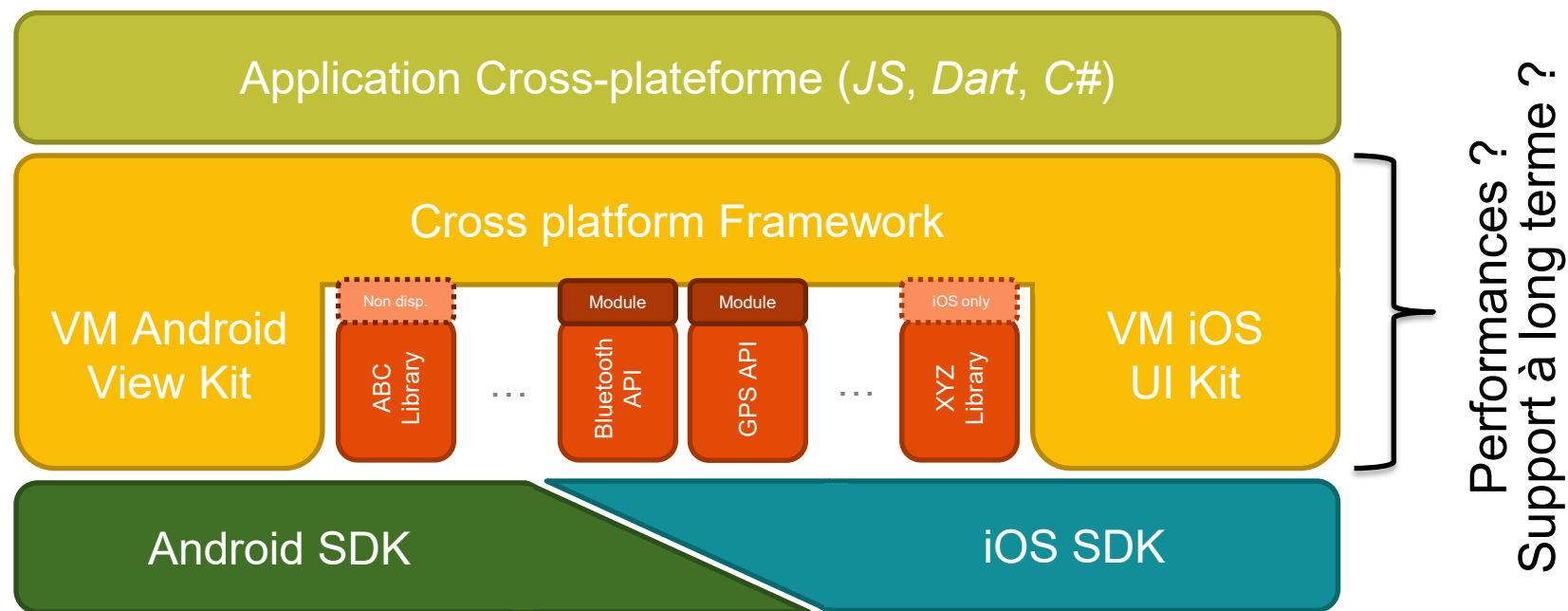
- Développé initialement par *Adobe* sous le nom de *PhoneGap*
- *Apache Cordova* permet la réalisation d'applications mobiles hybrides (*HTML5* / natif)
  - Une *web app* empaquetée dans une application mobile native, avec une *WebView* exécutant l'app en local
- Mise à disposition de librairies *JavaScript* spécifiques pour interagir avec certaines fonctionnalités natives des OS mobiles
- *Apache Cordova* sert de base à plusieurs autres solutions cross-plateformes, telle que *Ionic* qui ajoute le support de frameworks web avancés tels que : *Angular*, *React* ou *Vue*



# Cross-plateformes – Solutions basées sur une VM

- Il existe plusieurs solutions basées sur différents langages de programmation (*JavaScript*, *Dart*, *C#*, etc.)
- Ces différentes solutions peuvent avoir des architectures relativement différentes mais elles reposent principalement sur :
  - une **machine virtuelle** permettant d'exécuter du code en *Dart*, en *JavaScript*, en *C#*, etc.
  - un **moteur de rendu graphique** permettant d'afficher l'app sur les différentes plateformes
  - une collection de **modules natifs** permettant d'appeler des *API* propres aux diff. plateformes
- L'application développée pourra ainsi être buildée pour *Android* et pour *iOS*
  - mais potentiellement aussi sur d'autres plateformes (*Windows*, *MacOS*, *Linux*, web, etc.)

# Cross-plateformes – Solutions basées sur une VM

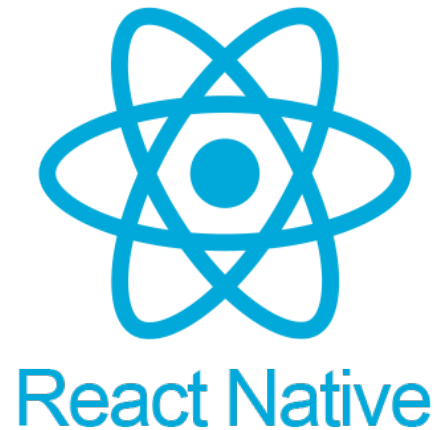


# Cross-plateformes – Solutions basées sur une VM

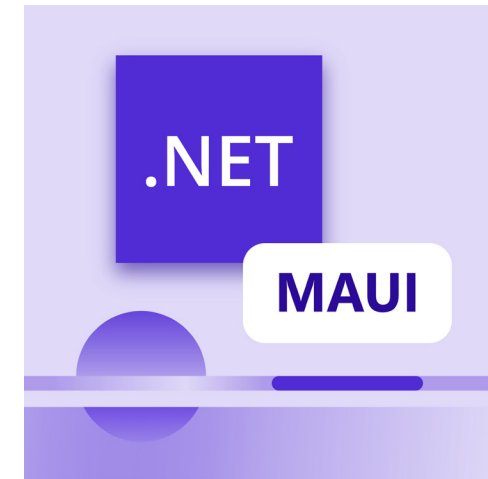
- Les trois principales solutions rentrant dans cette catégorie sont :



Google  
Dart  
License BSD



Meta (Facebook)  
JavaScript  
License MIT

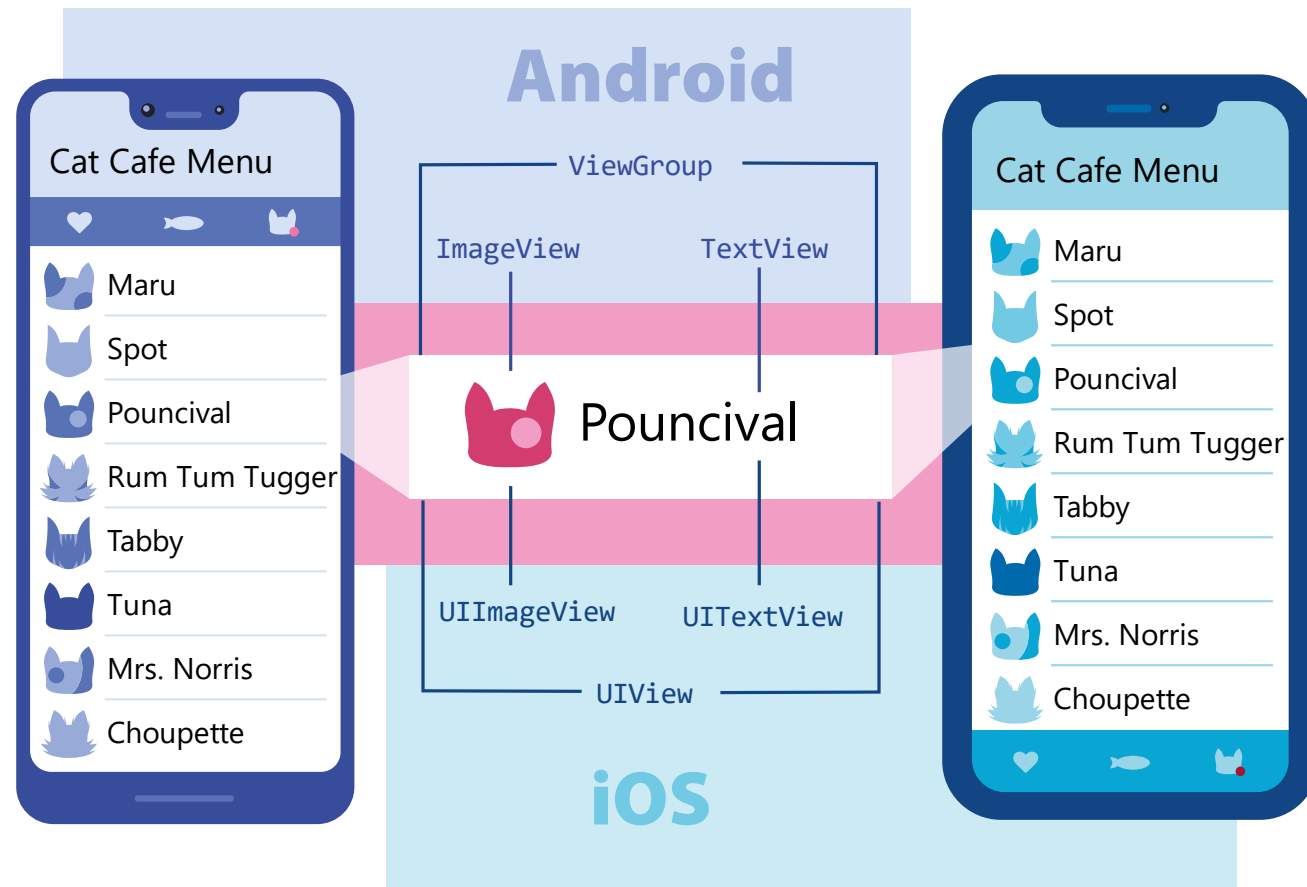


Microsoft  
Univers .NET / C#  
License MIT  
Successeur de XAMARIN

- Les moteurs de jeux vidéo se rapprochent également de cette catégorie, par exemple *Unity* permettant du cross-plateformes en C#

# Cross-plateformes – *React Native*

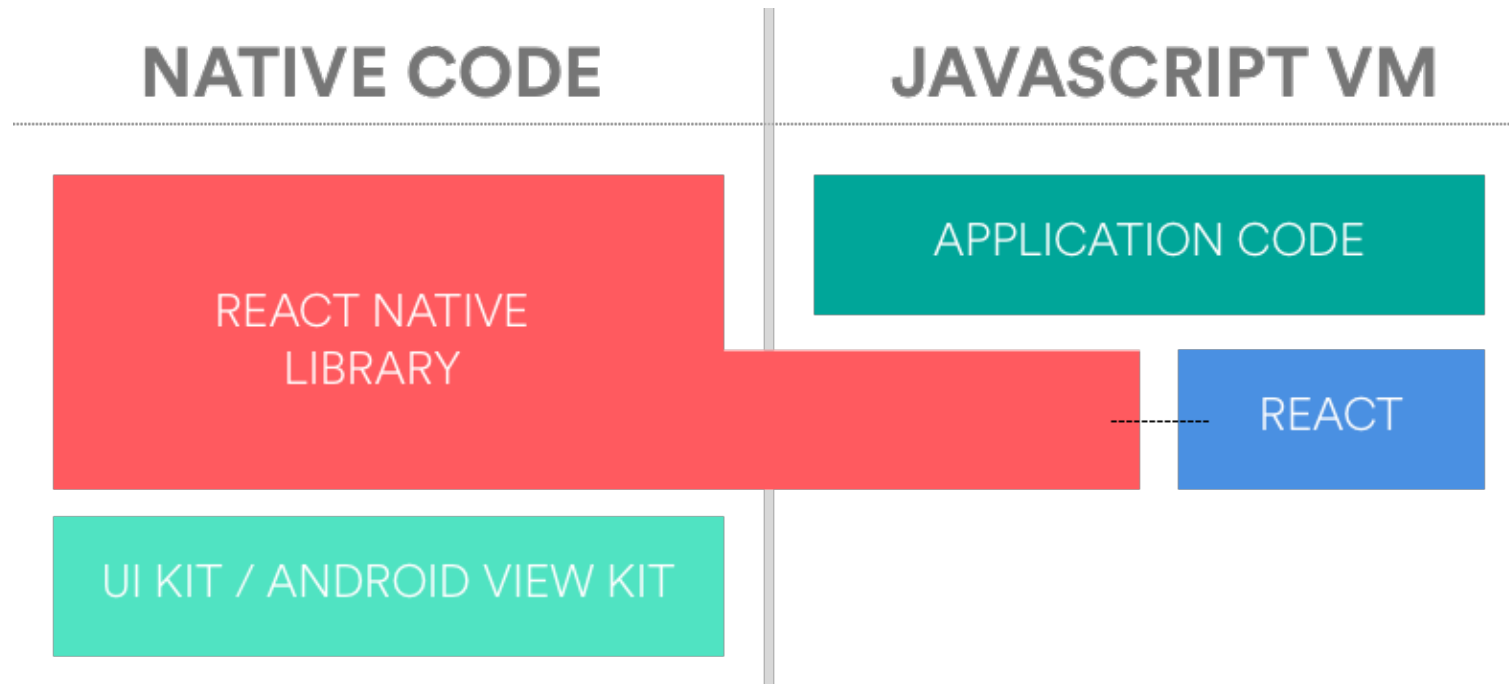
- Le moteur graphique va mapper l'interface décrite en composants natifs :
  - *React Native* tient son nom de l'utilisation de widgets natifs pour l'UI, offrant une interaction fluide avec l'utilisateur





# Cross-plateformes – *React Native*

- Le code *JavaScript* de l'app ne sera pas exécuté nativement, une *VM* (adaptée à la plateforme) l'interprétera au runtime



# Cross-plateformes – *React Native*

- Sous *Android* la *VM* prend la forme d'une *dynamically linked shared libraries* (.so) compilée pour chaque famille de processeurs (*arm*, *intel*, etc.)
- Par exemple si on ouvre un *APK* d'une app *React Native*, on trouve :

i4\_v8a > lib > arm64-v8a

Nom	Modifié le	Type	Taille
libjsc.so	01.01.1981 00:01	Fichier SO	8 603 Ko
libreanimated.so	01.01.1981 00:01	Fichier SO	1 723 Ko
libhermes-inspector.so	01.01.1981 00:01	Fichier SO	1 259 Ko
libfbjni.so	01.01.1981 00:01	Fichier SO	931 Ko
libc++_shared.so	01.01.1981 00:01	Fichier SO	915 Ko
libreactnativejni.so	01.01.1981 00:01	Fichier SO	827 Ko
libnative-image-transcoder.so	01.01.1981 00:01	Fichier SO	466 Ko
libfolly_futures.so	01.01.1981 00:01	Fichier SO	343 Ko

JSC - *JavaScriptCore*  
un moteur *JavaScript*

# Cross-plateformes – *React Native* – *Packages natifs*

- *React Native* et sa communauté proposent des packages permettant d'utiliser les fonctionnalités natives des *SDK* des plateformes mobiles

Par ex. *La Géolocalisation* :

- Mise à disposition d'une interface en *JavaScript* permettant d'uniformiser l'accès à la géolocalisation sur *iOS* et sur *Android*
- Le package repose sur des modules natifs *Objective-C* et *Java* utilisant les API propres aux deux plateformes :

```
@Override
public void onLocationChanged(Location location) {
    synchronized (SingleUpdateRequest.this) {
        if (!mTriggered && isBetterLocation(location, mOldLocation)) {
            mSuccess.invoke(locationToMap(location));
            mHandler.removeCallbacks(mTimeoutRunnable);
            mTriggered = true;
            mLocationManager.removeUpdates(mLocationListener);
        }

        mOldLocation = location;
    }
}
```

```
- (void)locationManager:(CLLocationManager *)manager
    didUpdateLocations:(NSArray<CLLocation *> *)locations
{
    // Create event
    CLLocation *location = locations.lastObject;

    ...
}
```

# Cross-plateformes – *React Native* – OTA

- *React Native* étant basé sur du *JavaScript* interprété, rien n'empêche l'app de télécharger une nouvelle version d'un script sans passer par une mise à jour complète via un store (*Google Play* ou *App Store*)
- Étonnamment *Apple* n'interdit pas cette pratique, mais...

2.4.5 Apps distributed via the Mac App Store have some additional requirements to keep in mind:

(iv) They may not download or install standalone apps, kexts, additional code, or resources to add functionality or significantly change the app from what we see during the review process.

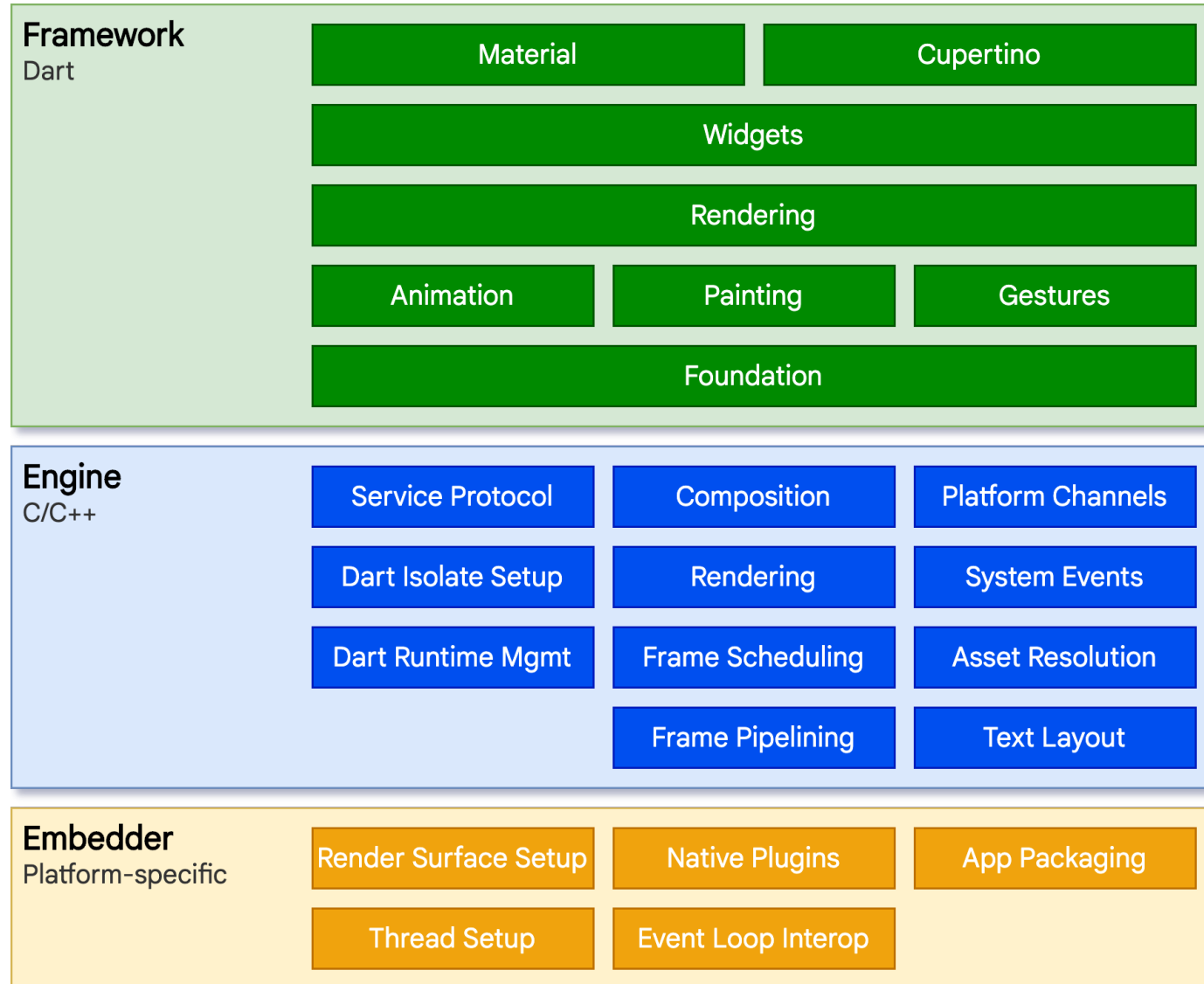
# Cross-plateformes – *Flutter*

- *Flutter* est un kit de développement logiciel cross-plateformes créé par *Google*
- Initialement destiné au développement mobile, il permet aujourd'hui de réaliser des applications pour *Android*, *iOS*, *Linux*, *macOS*, *Windows*, *Fuchsia*, le *web* et également des systèmes embarqués, à partir de sources uniques
- Les composants principaux de *Flutter* sont :
  - Plateforme *Dart*                      Utilisation du langage *Dart*. Utilisation d'une machine virtuelle *Dart* avec compilation *JIT* sur *Desktop*. Sur mobile, compilation *JIT* lors du développement et compilation *AOT* en mode release
  - Foundation Library              Ensembles d'API permettant de construire une app
  - Moteur *Flutter*                    Moteur graphique *Skia* permettant le rendu des app
  - Widgets                              Collection de widgets graphiques, 2 versions : *Material* & *Cupertino* conçus pour ressembler visuellement à des composants natifs
  - Development Tools                Collection d'outils permettant le développement

# Cross-plateformes – *Flutter* – Langage *Dart*

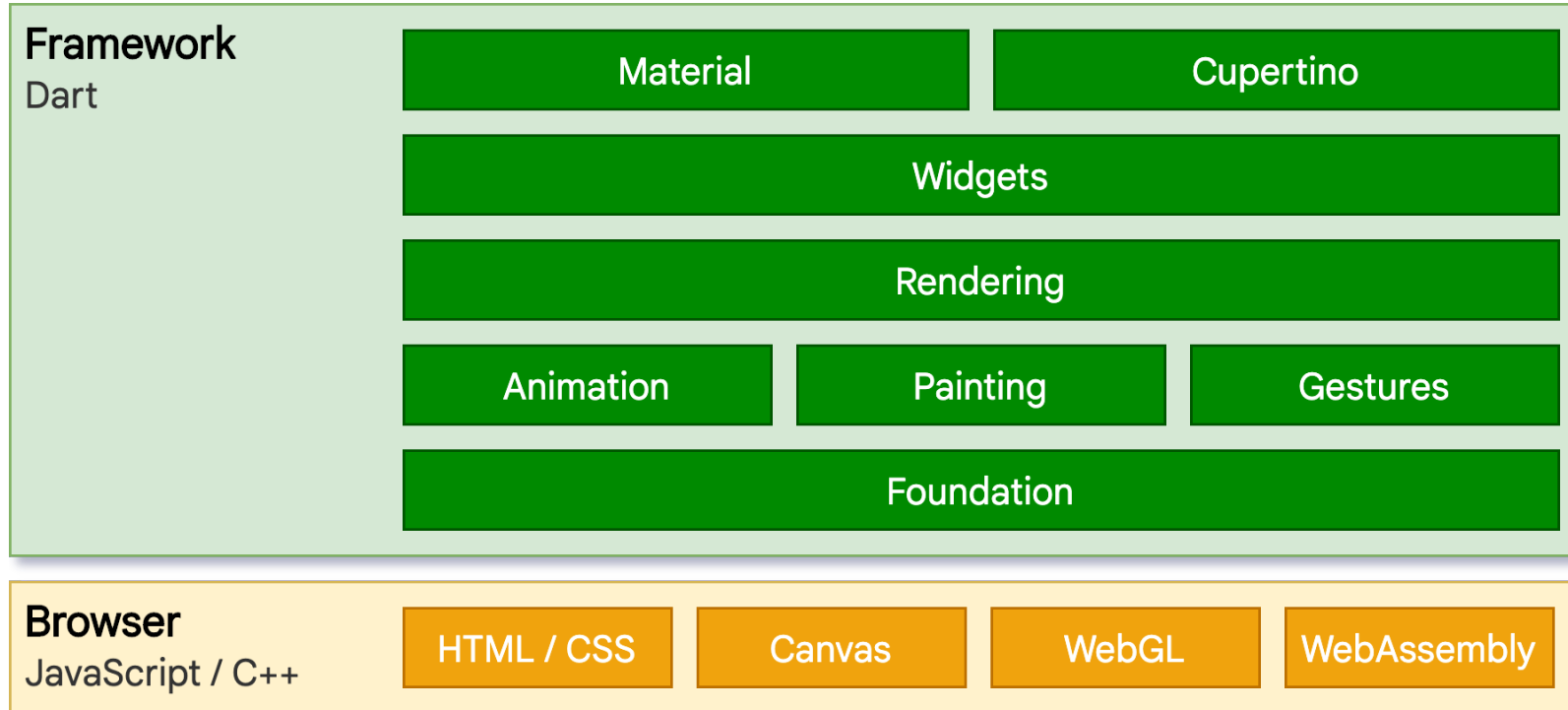
- Langage de programmation développé par *Google* à partir de 2011
- Il est conçu et optimisé pour le développement cross-plateformes
- Il s'agit d'un langage orienté objet avec une syntaxe «à la *Java*», il dispose d'un *garbage collector*
- Il y a 4 façons d'exécuter du code *Dart* :
  - Native                      Les dernières versions de *Dart* peuvent compiler le code vers un exécutable natif indépendant
  - Autonome                  Utilisation d'une *VM Dart*, principalement utilisé pour les applications *Desktop*
  - Compilation AOT      Approche utilisée par *Flutter* pour le déploiement sur *Android* et *iOS*
  - Transpilé en JS        Trans-compilation source-source vers *JavaScript* permettant l'exécution dans un navigateur web

# Cross-plateformes – *Flutter* – *Mobile/Desktop*





# Cross-plateformes – *Flutter* – web



# Cross-plateformes – *Flutter* – *packages*

- *Flutter* et sa communauté mettent également à disposition des packages permettant l'ajout de fonctionnalités natives, par exemple pour l'accès aux capteurs du smartphone :
  - Une interface commune en *Dart*
  - Des implémentations spécifiques aux différentes plateformes, par exemple les capteurs :

## Android, en Kotlin :

```
magnetometerStreamHandler = StreamHandlerImpl(
    sensorsManager,
    Sensor.TYPE_MAGNETIC_FIELD
)

override fun onListen(arguments: Any?, events: EventSink) {
    sensorEventListener = createSensorEventListener(events)
    sensorManager.registerListener(sensorEventListener, sensor, SensorManager.SENSOR_DELAY_NORMAL)
}

override fun onListen(arguments: Any?, events: EventSink) {
    [...]
    sensorEventListener = createSensorEventListener(events)
    sensorManager.registerListener(sensorEventListener, sensor, samplingPeriod)
    [...]
}
```

## iOS, en Objective-C :

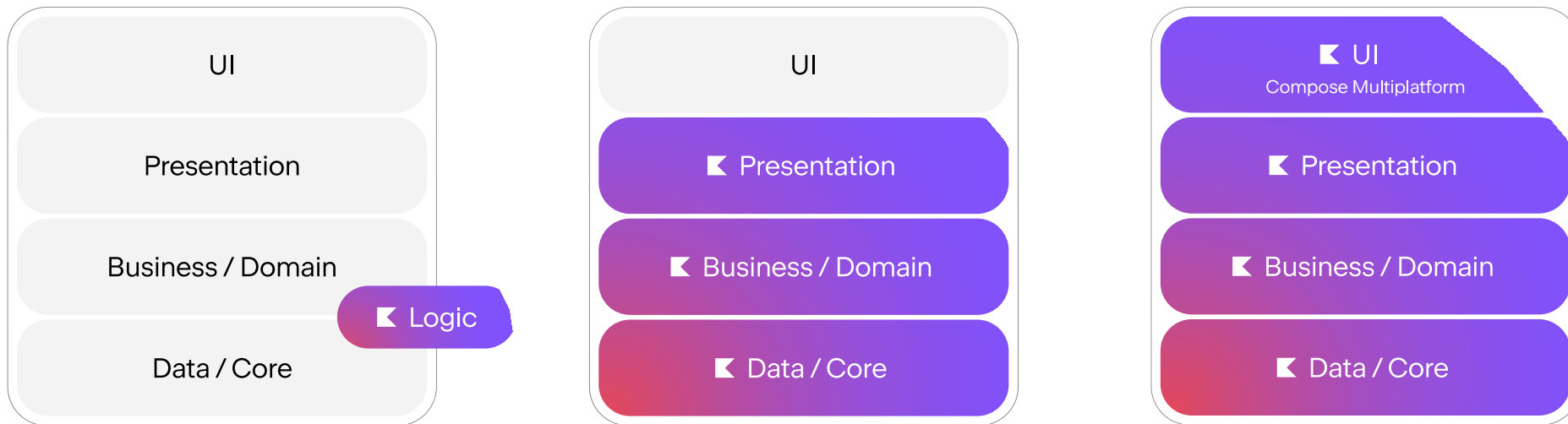
```
_motionManager
startDeviceMotionUpdatesUsingReferenceFrame:
    // https://developer.apple.com/documentation/coremotion/cmattitudereferenceframe?language=objc
    // "Using this reference frame may require device movement to
    // calibrate the magnetometer," which is desired to ensure the
    // DeviceMotion actually has updated, calibrated geomagnetic data.
    CMAccelerationReferenceFrameXMageticNorthZVertical
        toQueue:[[NSOperationQueue alloc]
            init]
        withHandler:^(CMDeviceMotion *motionData,
            NSError *error) {
            // The `magneticField` is a
            // CMMagnetometerField.
            CMMagnetometerField b =
                motionData.magneticField.field;
            if (!_isCleanUp) {
                return;
            }
            sendTriplet(b.x, b.y, b.z, eventSink);
        }
    };
```

Passage au Swift lors  
de l'intégration de la  
*samplingperiod* en 2024

2024

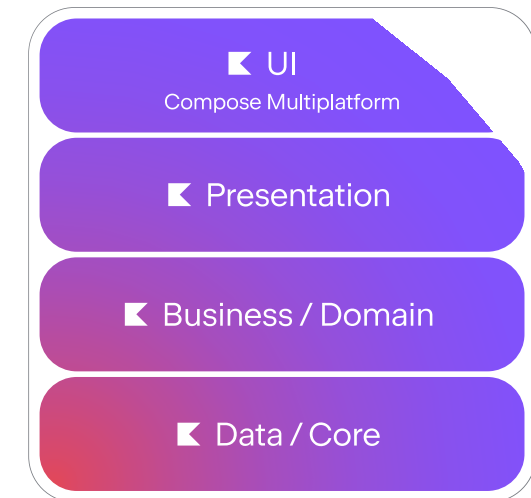
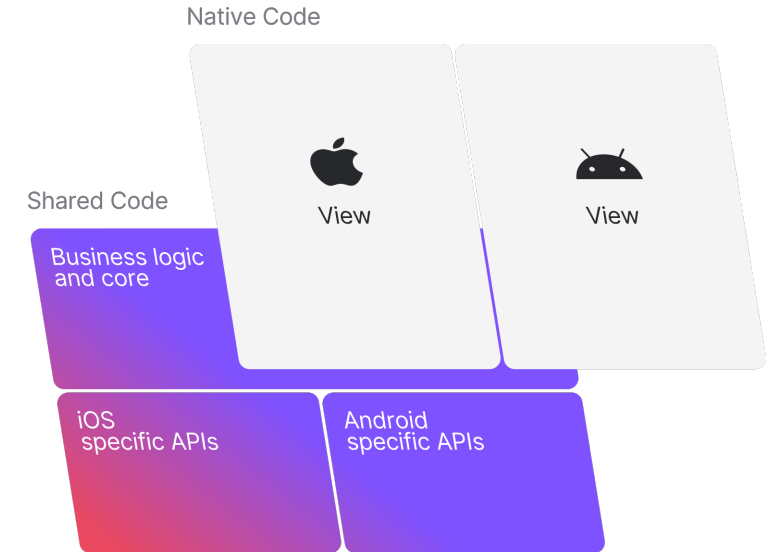
# Cross-plateformes – *Kotlin Multiplatform*

- Nouvelle approche proposée par *JetBrain* pour un développement *cross-plateformes* «à la carte», actuellement en *béta*
- Offre une granularité assez importante sur les fonctionnalités à partager entre plusieurs plateformes : d'un module précis à une app complète en passant par la gestion des données, des logiques métiers et/ou du réseau



# Cross-plateformes – *Kotlin Multiplatform*

- L'approche est de pouvoir mettre en commun une partie du développement, par exemple la gestion des données et les logiques métiers spécifiques sous la forme d'une micro-librairie
- On pourra ensuite intégrer notre micro-librairie dans nos applications *iOS* et *Android* natives
- *Kotlin Multiplatform* est en train d'être complété avec *Compose Multiplatform* qui permettra également de mettre en commun l'*UI* :
  - Support pour *Android*, *Windows*, *MacOS* et *Linux*
  - support en beta pour *iOS*
  - support en alpha pour le *web*



HE<sup>VD</sup>  
IG

HAUTE ÉCOLE  
D'INGÉNIERIE  
ET DE GESTION  
DU CANTON  
DE VAUD