

Cours Android 2018/2019

Patrice raby <patrice.raby@corellis.eu>

- Souvent présenté comme l'alternative de Google à l'iPhone
- Système d'exploitation pour terminaux mobiles
- Basé sur Linux
- Open Source (licence Apache)



- Framework applicatif avec réutilisation et remplacement possible des composants
- DVM : Dalvik Virtual Machine (machine virtuelle optimisée pour les périphériques mobiles) remplacée par ART (Android Runtime) depuis la version 5 (Lollipop)
- Navigateur intégré basé sur le moteur WebKit (OpenSource)
- Librairie 2D dédiée
- Gestion de la 3D basée sur une implémentation d'OpenGL ES 1.0 (avec support de l'accélération matérielle)
- Base de données SQLite
- Gestion des écrans tactiles et du Multitouch

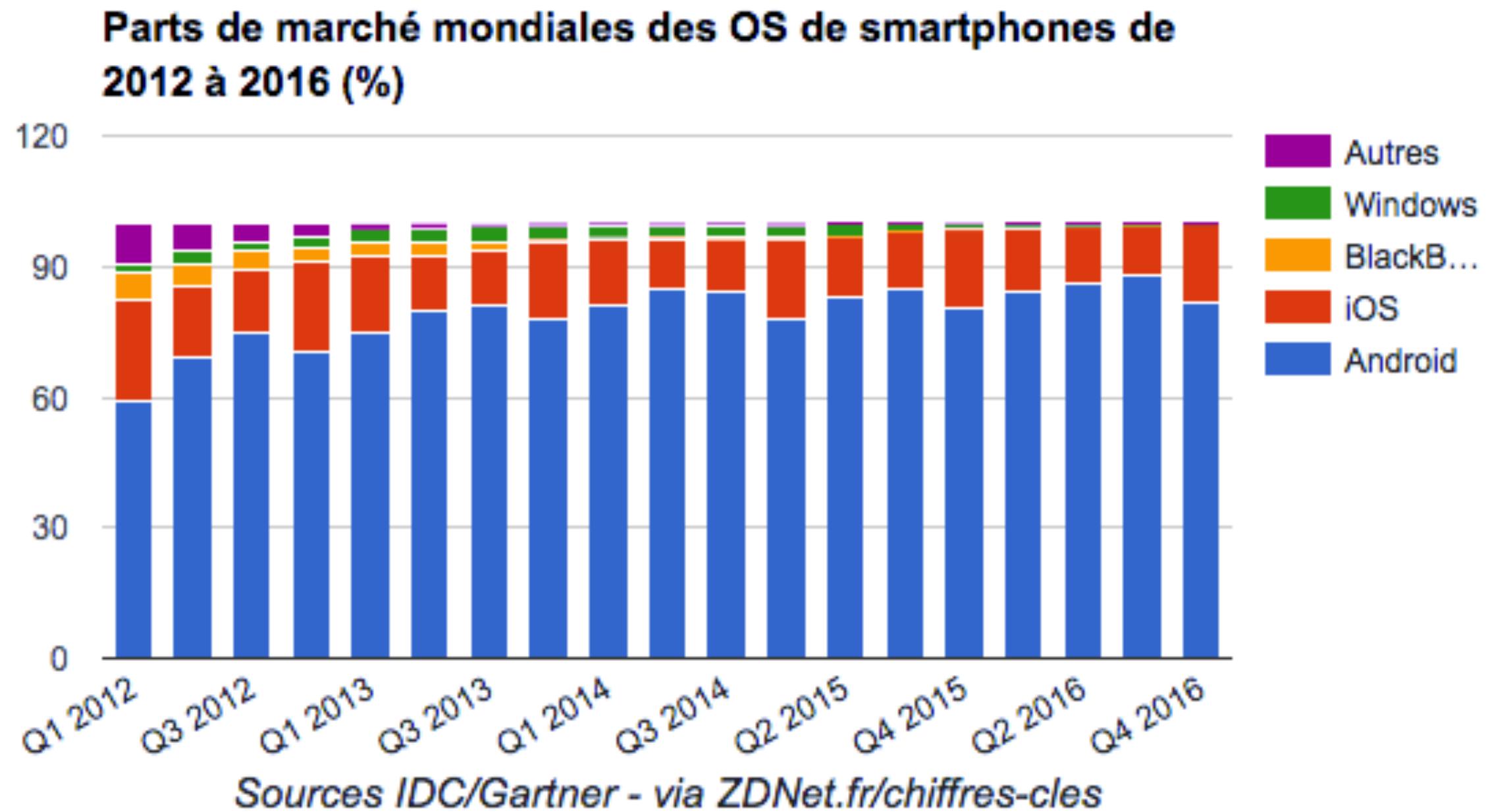


- Multimédia : support de la plupart des formats classiques d'images, de vidéos et audio (MPEG4, H. 264, MP3, AAC, AMR, JPG, PNG, GIF)
- Téléphonie GSM (selon hardware)
- Bluetooth, EDGE, 3G et WiFi (selon hardware)
- Caméra, GPS, compas et accéléromètre (selon hardware)
- Environnement de développement riche incluant :
 - Un émulateur (avec une interface de contrôle)
 - Des outils de débogage
 - Outils de profilage mémoire et performance



- Android et iOS sont aujourd’hui les deux seules alternatives crédibles sur le marché des smartphones. Au quatrième trimestre de l’année 2016, Android et iOS ont compté pour 99,6 % des smartphones vendus (81,7 % pour Android et 17,9 % pour Apple)
- Les autres :
 - Windows
 - Blackberry OS : plutôt dédié entreprise, se démocratisant, mais en grosse difficulté.

Généralités :: Parts de marché mondiales %





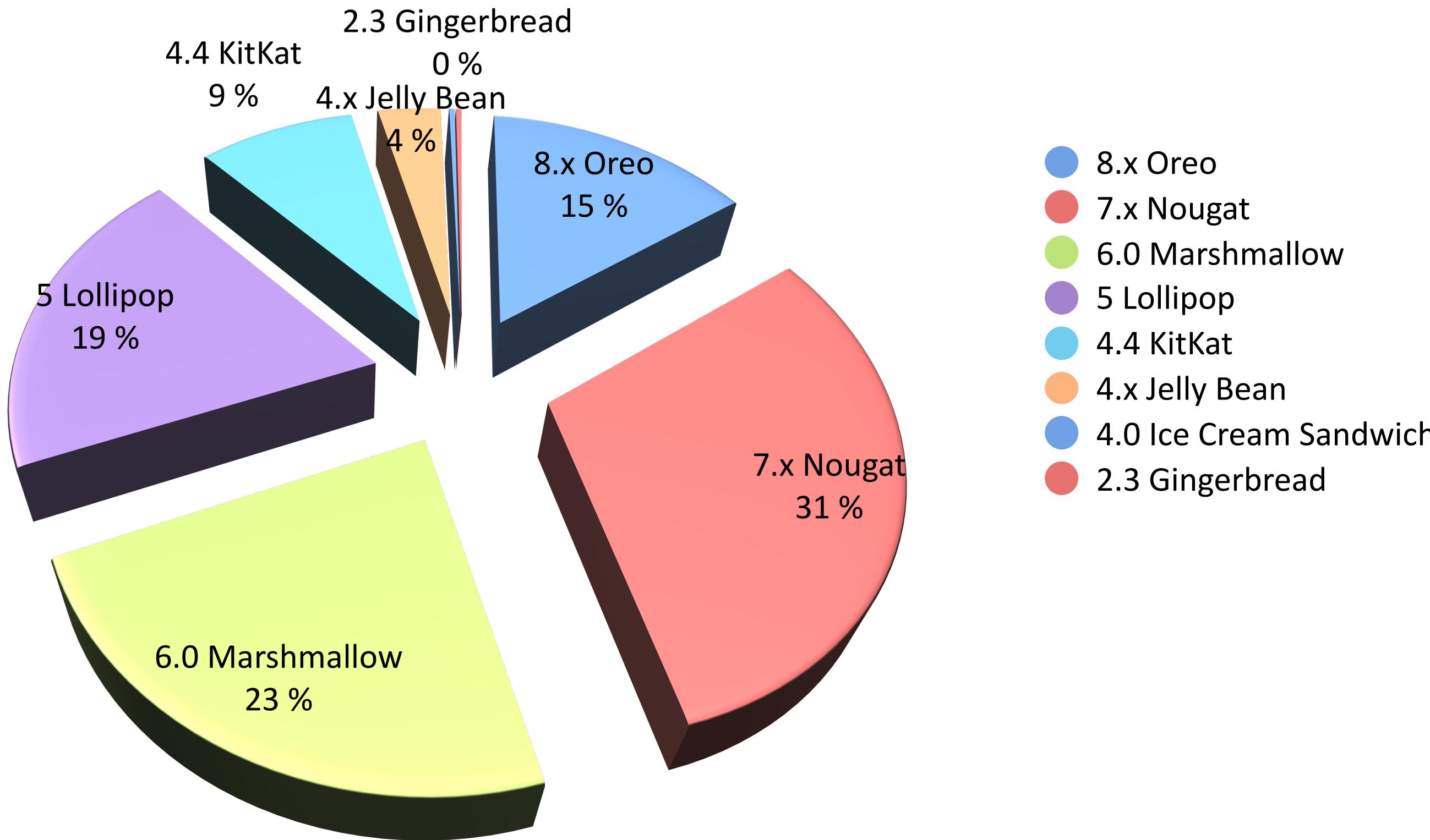
2005	Google achète Android Inc, Le travail sur Davilk commence
2007	Open handset Alliance annonce la sortie prochaine du Software Development Kit.
2008 – 2010	Android devient la plateforme mobile principale
2011 - 2013	Consoles de jeux vidéo, Tablettes, téléviseurs, autoradios, ...
2014 -2017	Wearable devices: Google Glasses, smartwatches, ...
Futur?	Objets connectés

Généralités :: Les versions

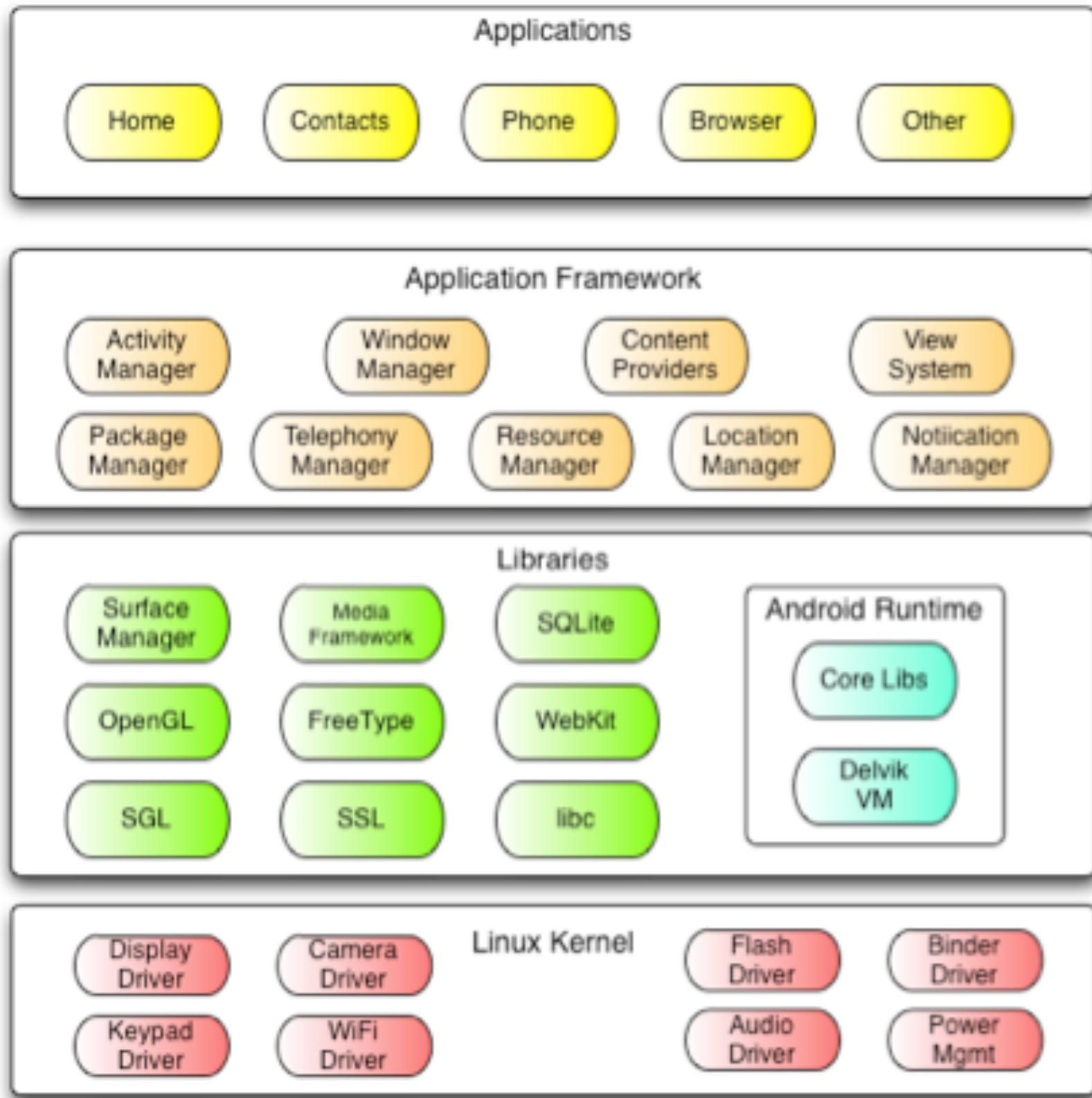


Version	API Level	Nickname
Android 1.0 (09/2008)	1	Android 1.0
Android 1.1 (02/2009)	2	Android 1.1
Android 1.5 (04/2009)	3	Cupcake
Android 1.6 (09/2009)	4	Donut
Android 2.0 (10/2009)	5	Eclair
Android 2.01 (12/2009)	6	Eclair
Android 2.1 (01/2010)	7	Eclair
Android 2.2 (05/2010)	8	Froyo
Android 2.3 – 2.3.2 (12/2010)	9	Gingerbread
Android 2.3.3 -2.3.7 (02/2011)	10	Gingerbread
Android 3.x (02/2011)	11, 12, 13	Honeycomb
Android 4.0 – 4.0.4 (10/2011)	14,15	Ice Cream Sandwich
Android 4.1-4.3 (07/2012)	16, 17, 18	Jelly Bean
Android 4.4 (09/2013)	19, 20	KitKat, KitKat with wearable extensions
Android 5.0 (11/2014), 5.1	21, 22	Lollipop
Android 6.0 (3ème trimestre 2015)	23	MarshMallow
Android 7.x (Août 2016)	24, 25	Nougat
Android 8.0.x – 8.1.x (Août 2017)	26	Oreo

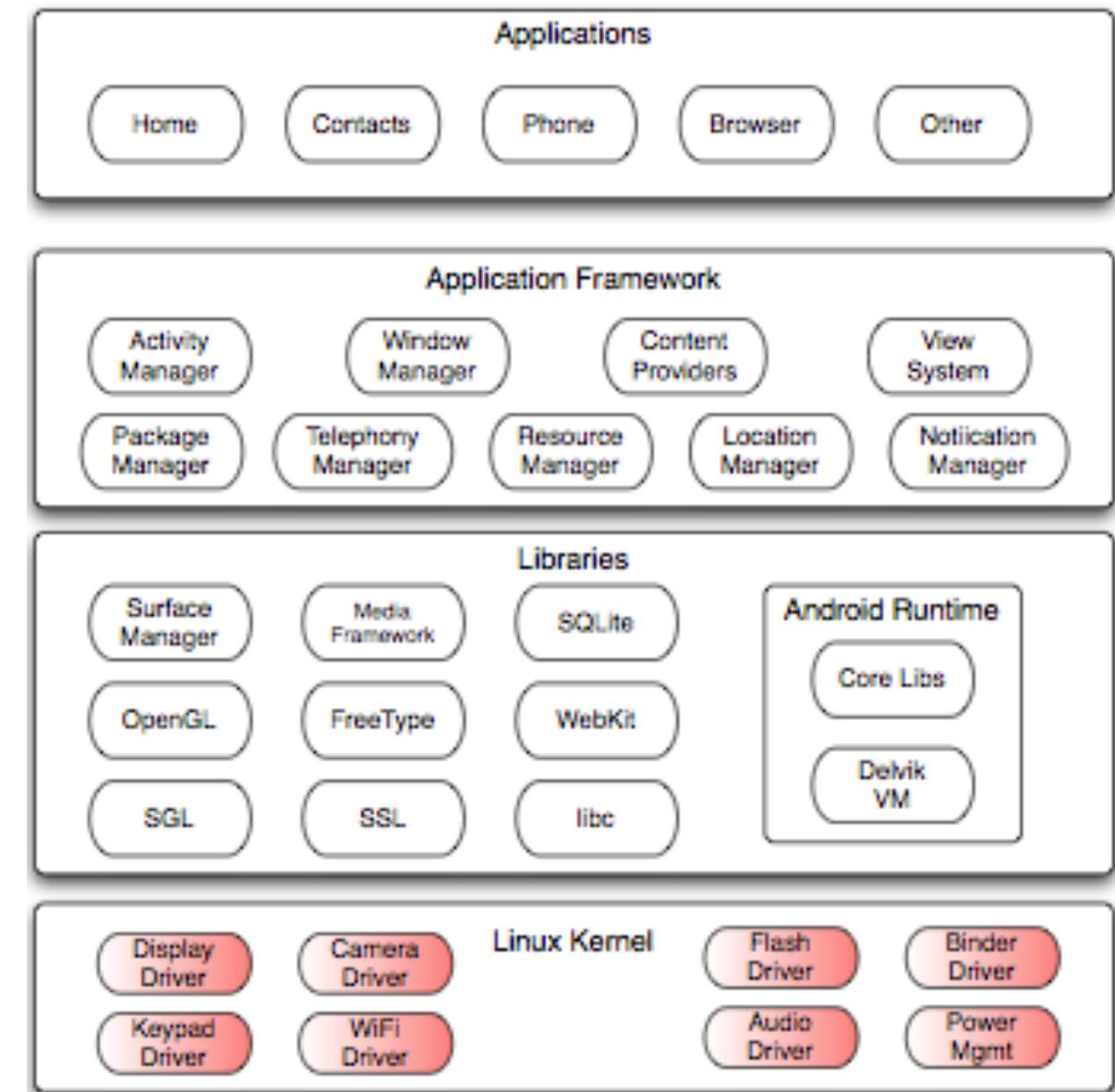
Généralités :: Distribution des versions



Généralités :: Architecture logicielle

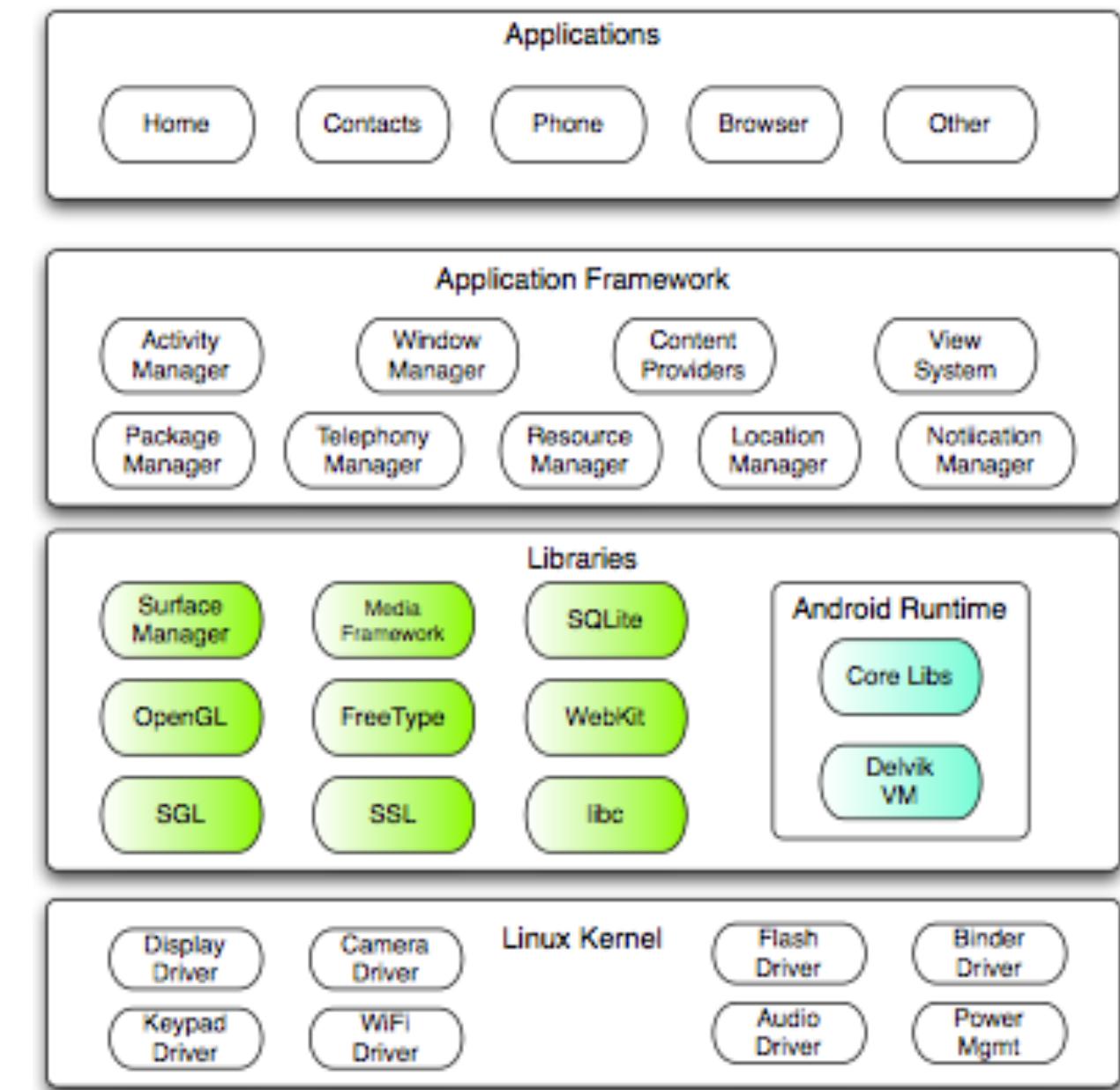


- Android fonctionne sur une base Linux
- Linux gère:
 - Couche d'abstraction matérielles
 - Gestion de la mémoire
 - Gestion des processus
 - La couche Réseau
- L'utilisateur n'accède jamais au système Linux
- La commande “ADB Shell” ouvre un shell Linux.

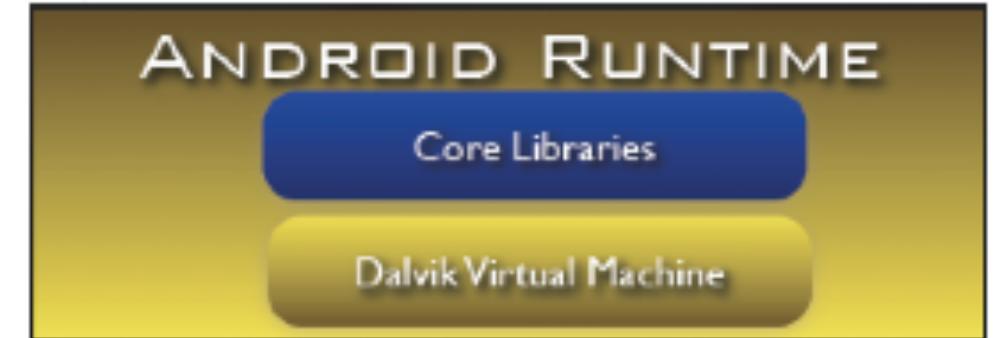


Généralités :: Les bibliothèques natives

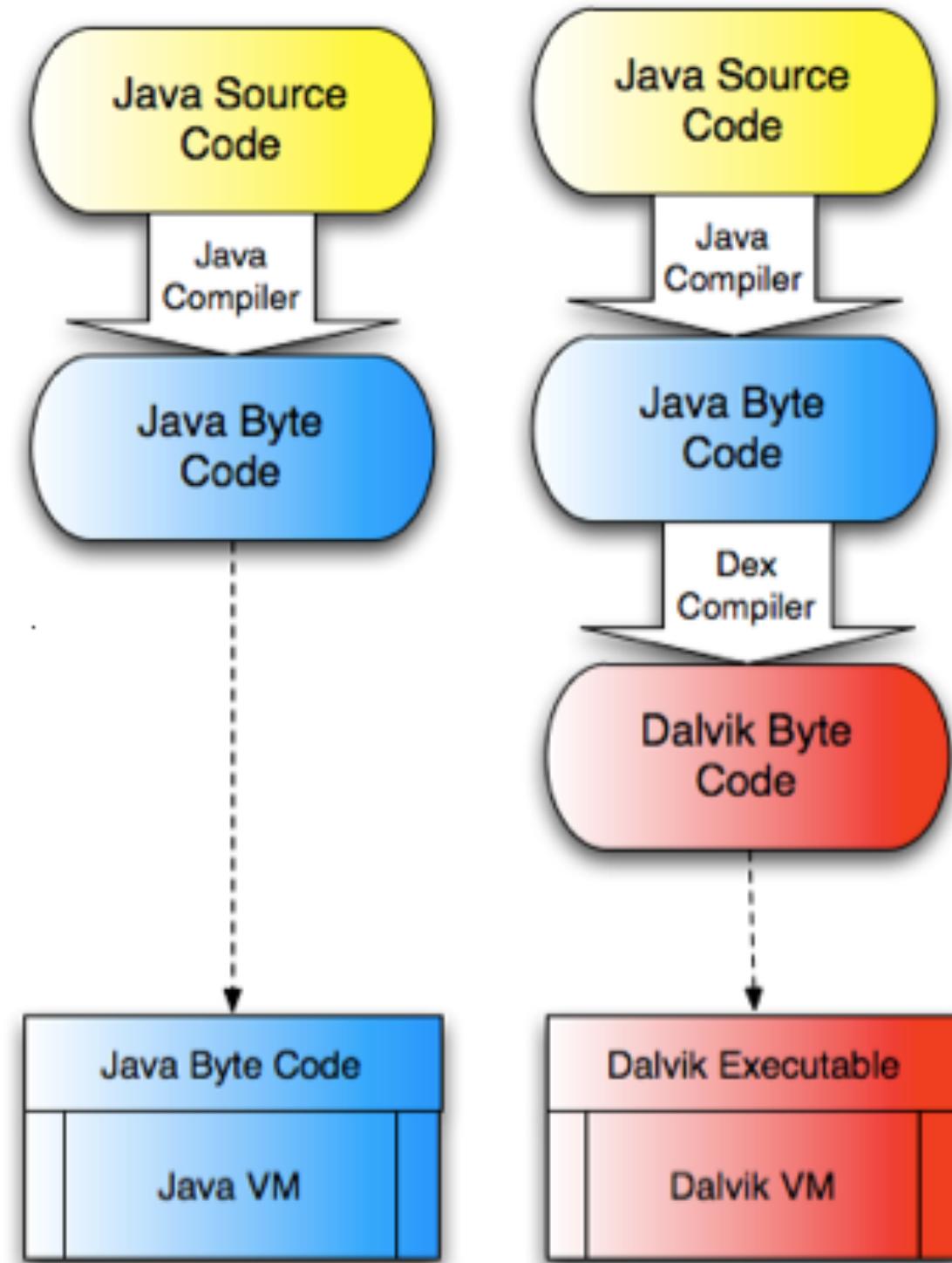
- **Bionic** librairie C optimisée pour Android
- **Webkit** librairie pour le rendu HTML
- **OpenGL** pour les graphiques
- **Codecs Média** qui supportent la majorité des codecs vidéos et Audio du marché
- Base de données **SQLLite**
- Et bien d'autres...



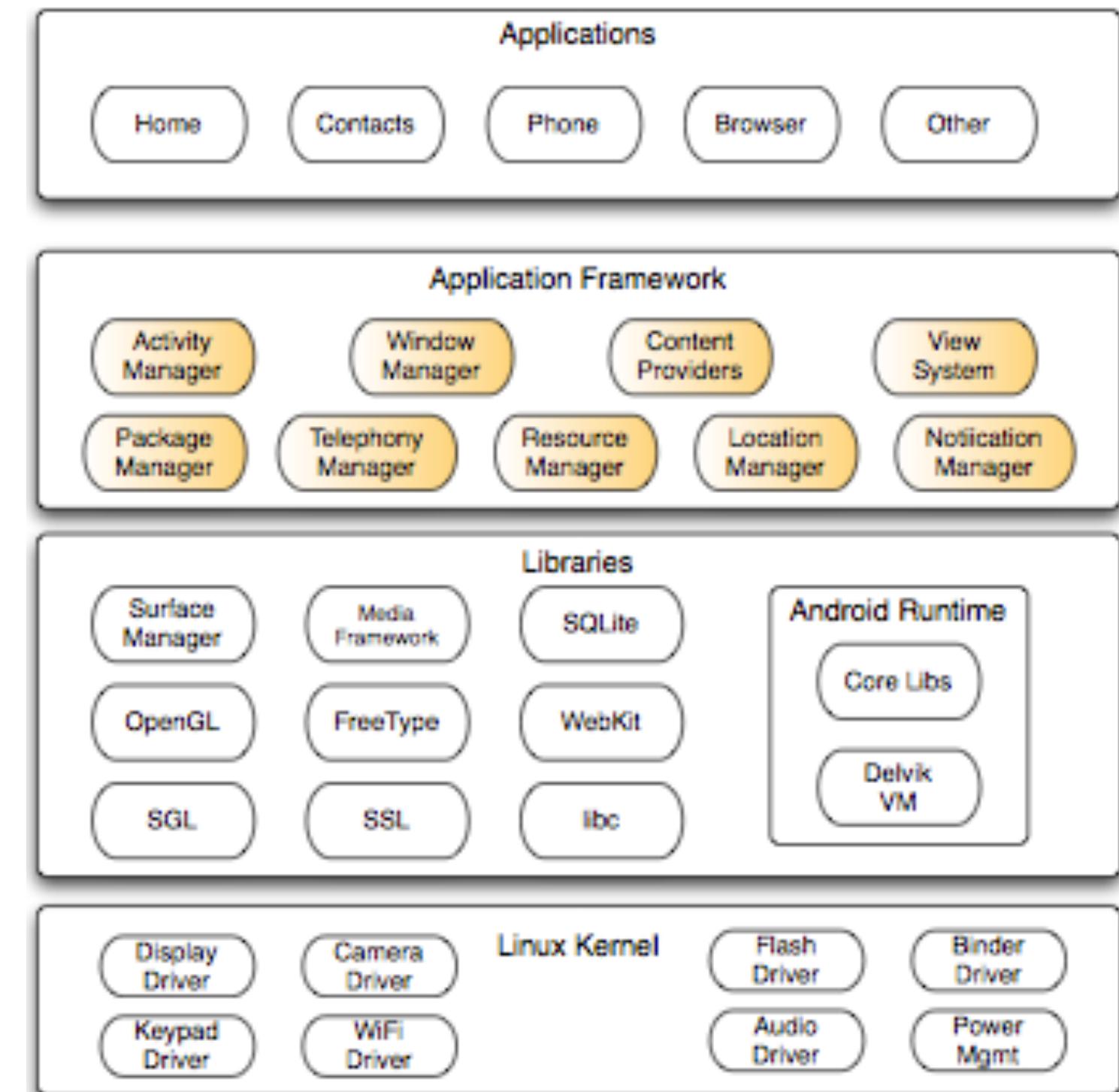
- La **VM Dalvik** est une implémentation Android de la VM Java
- **Optimisée** pour les terminaux mobile:
 - Consommation batterie
 - Capacités CPU
- Principale différences avec la VM
 - Architecture de registre vs Architecture de pile
 - Dalvik exécute des fichiers .dex
 - Implémentation plus efficace et compacte
 - Librairies différentes de celles du SDK



- Android Java = Java SE – AWT/Swing + Android API
- ART remplace Dalvik dans Lollipop (v5.x) et suivant
- ART (Compilation à l'installation – fichiers elf) VS Dalvik (JIT – fichiers dex)



- Ensemble de services « riche » contenu dans une api Java Intuitive
- Permet le développement d'applications plus facilement
 - Localisation, web, téléphonie, WIFI, bluetooth, notifications, médias, caméras, etc...



Introduction à Android Studio



Android
Studio

Powered by IntelliJ Platform

- Qu'est-ce que Android Studio?
 - IDE Spécialement développé pour Android
 - Remplaçant d'Eclipse avec le Plugin ADT
 - Contient tous les outils du SDK Android pour développer une application Android (tests, debug, profiling, ...)
 - Basé sur le moteur IntelliJ IDEA
 - <https://developer.android.com/studio/index.html>

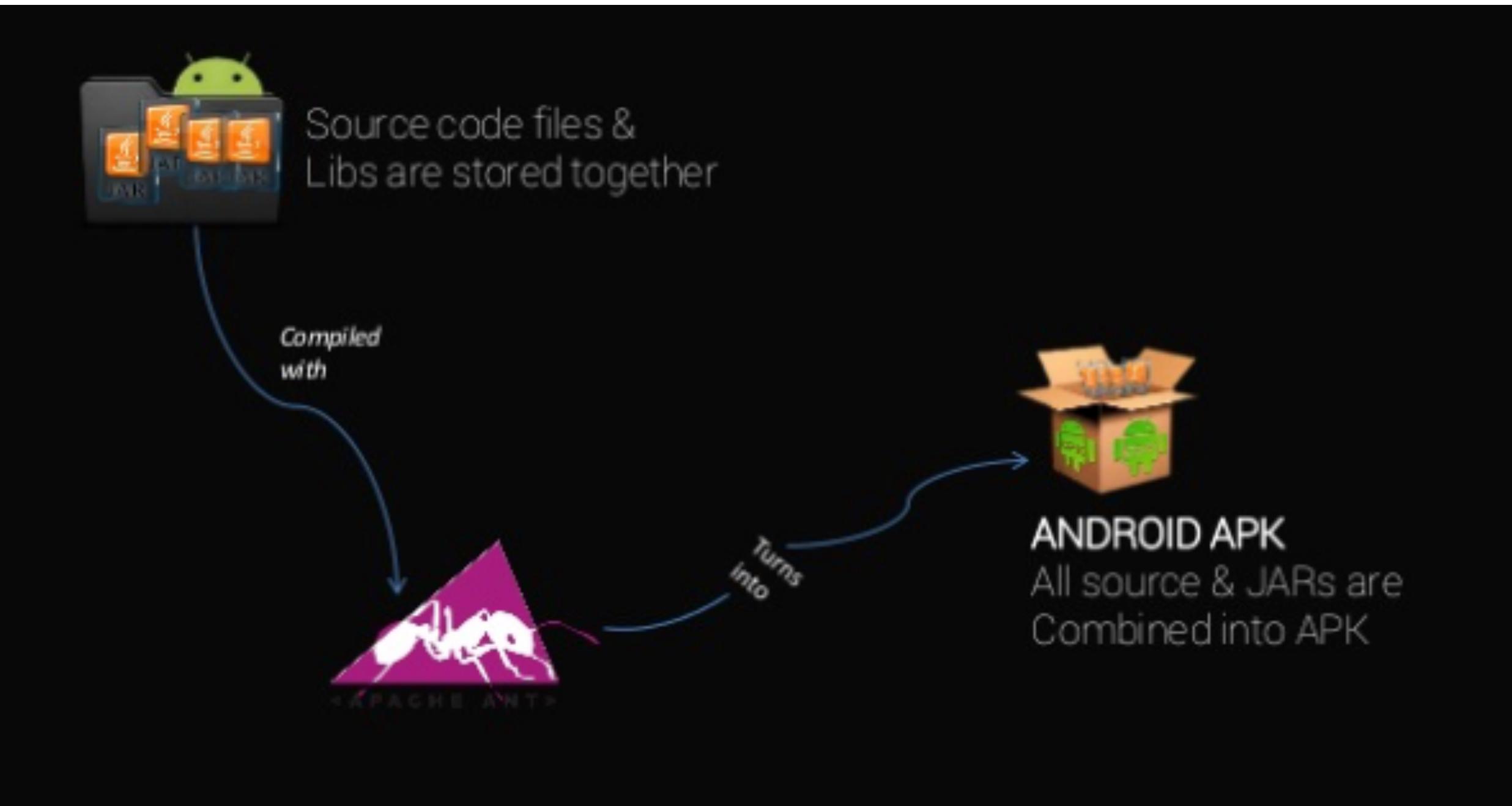


- Fonctionnalités?
 - Editeur code (refactoring, ...)
 - Layout editor « Riche »
 - Moteur de compilation Gradle
 - Compatible Maven
 - Template base Wizards
 - Lint tool analysis (analyse de code)
 - Accès facilité aux services Google (ex: GCM)

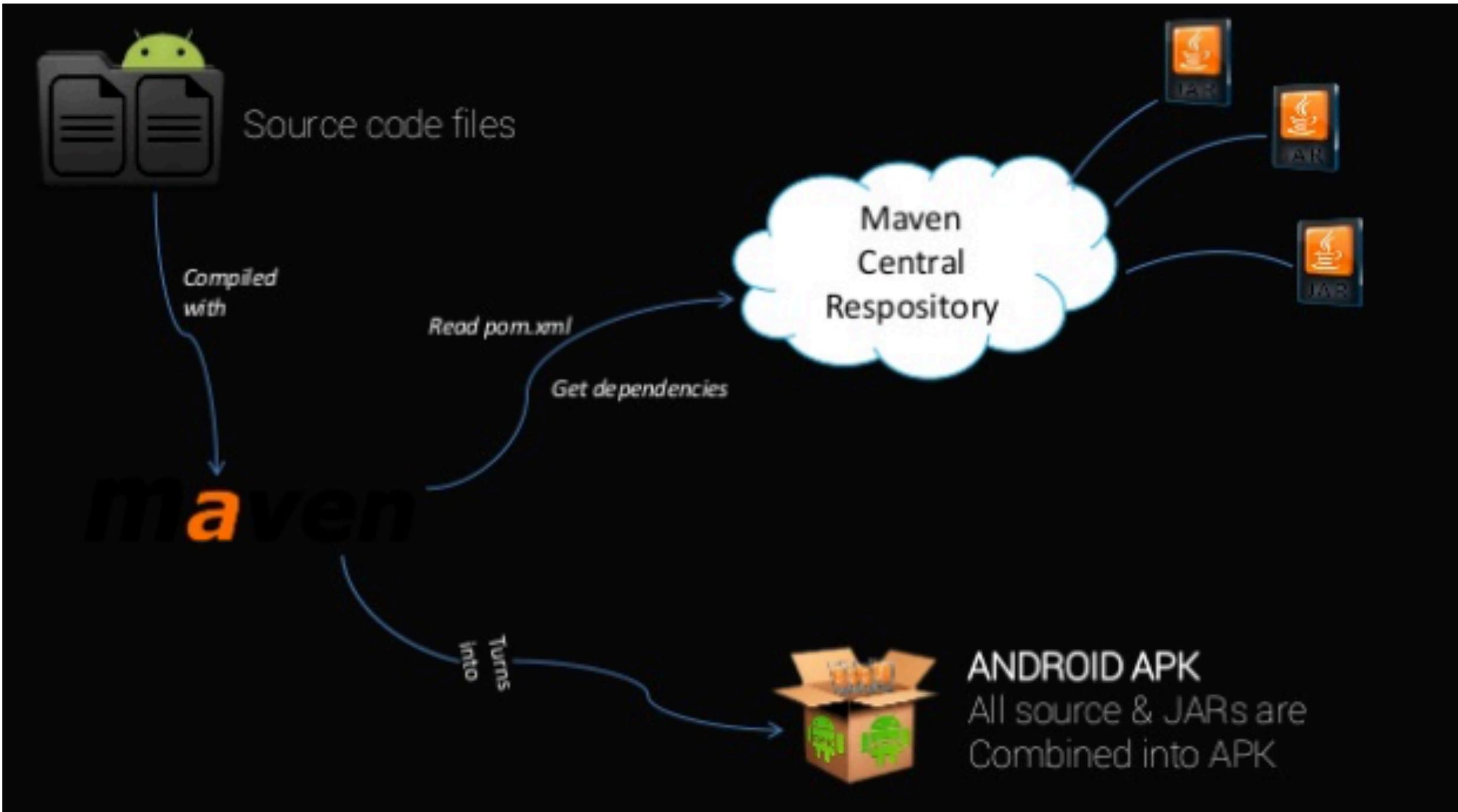


- Gradle est un moteur avancé de compilation basé sur Groovy
- Open source
- Combine les avantages de « ANT » et « MAVEN »
- Fonctionnalités:
 - Gestion des dépendances
 - Un moteur de compilation

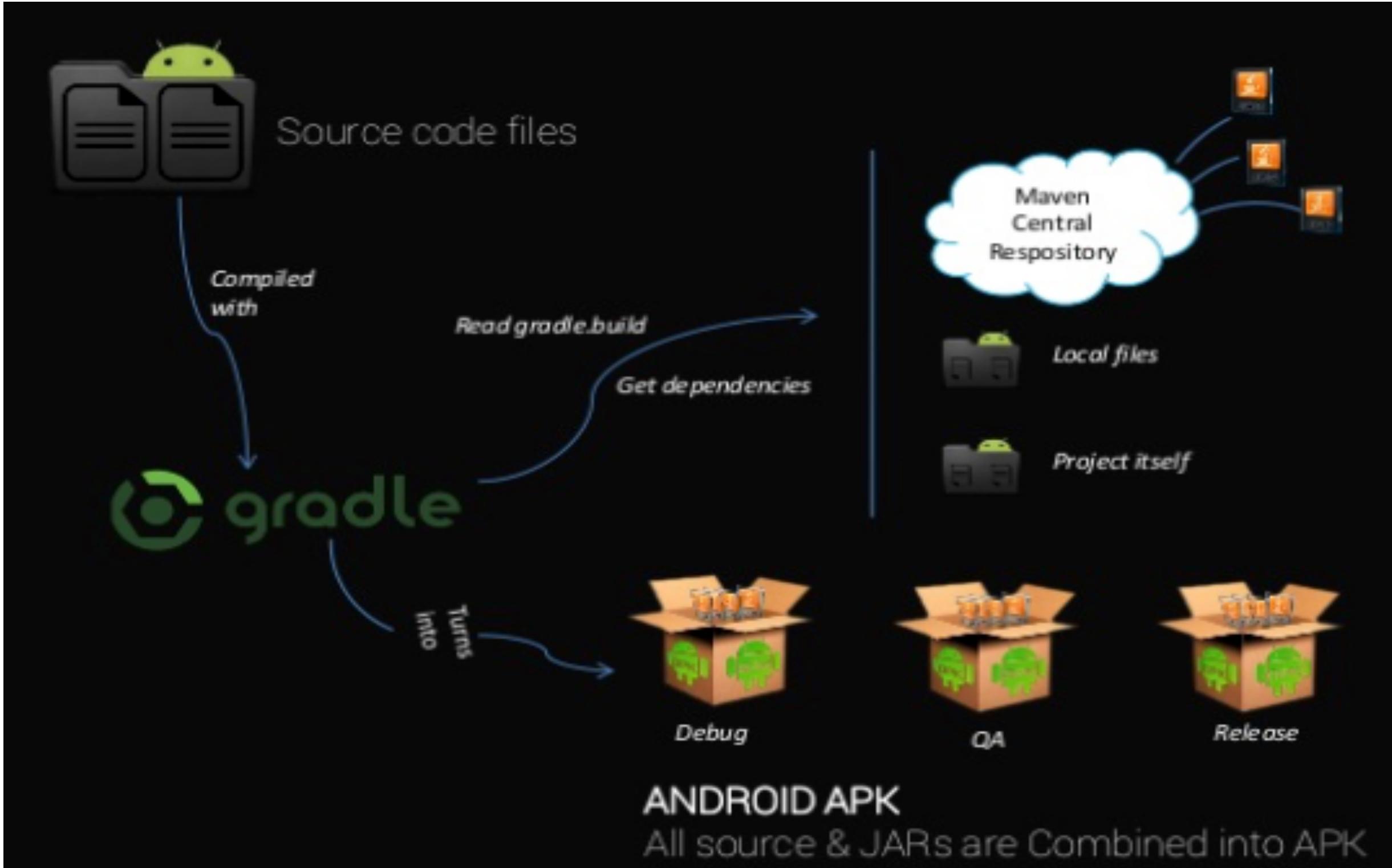
- Compilation avec ANT



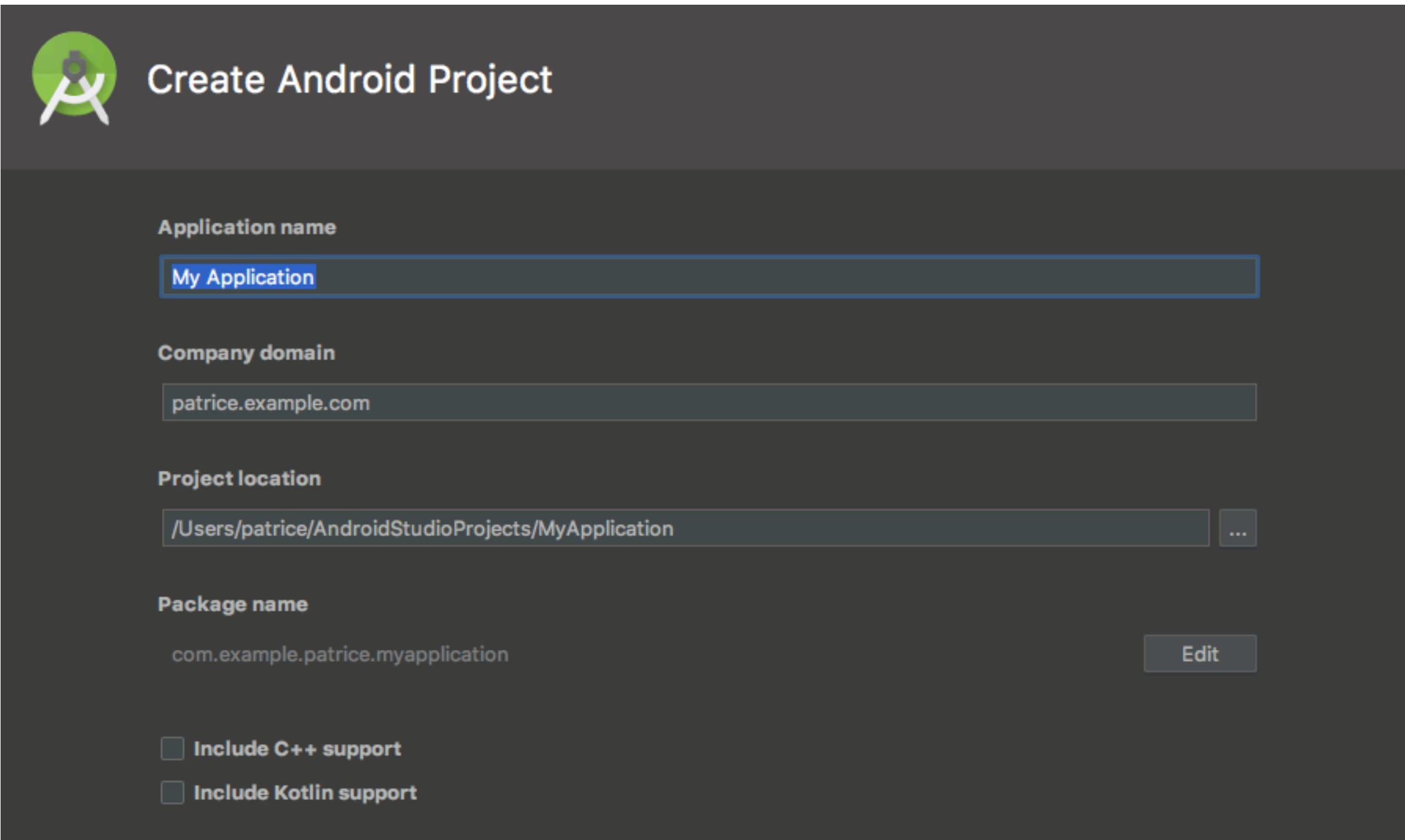
- Compilation avec MAVEN



- Compilation avec GRADLE

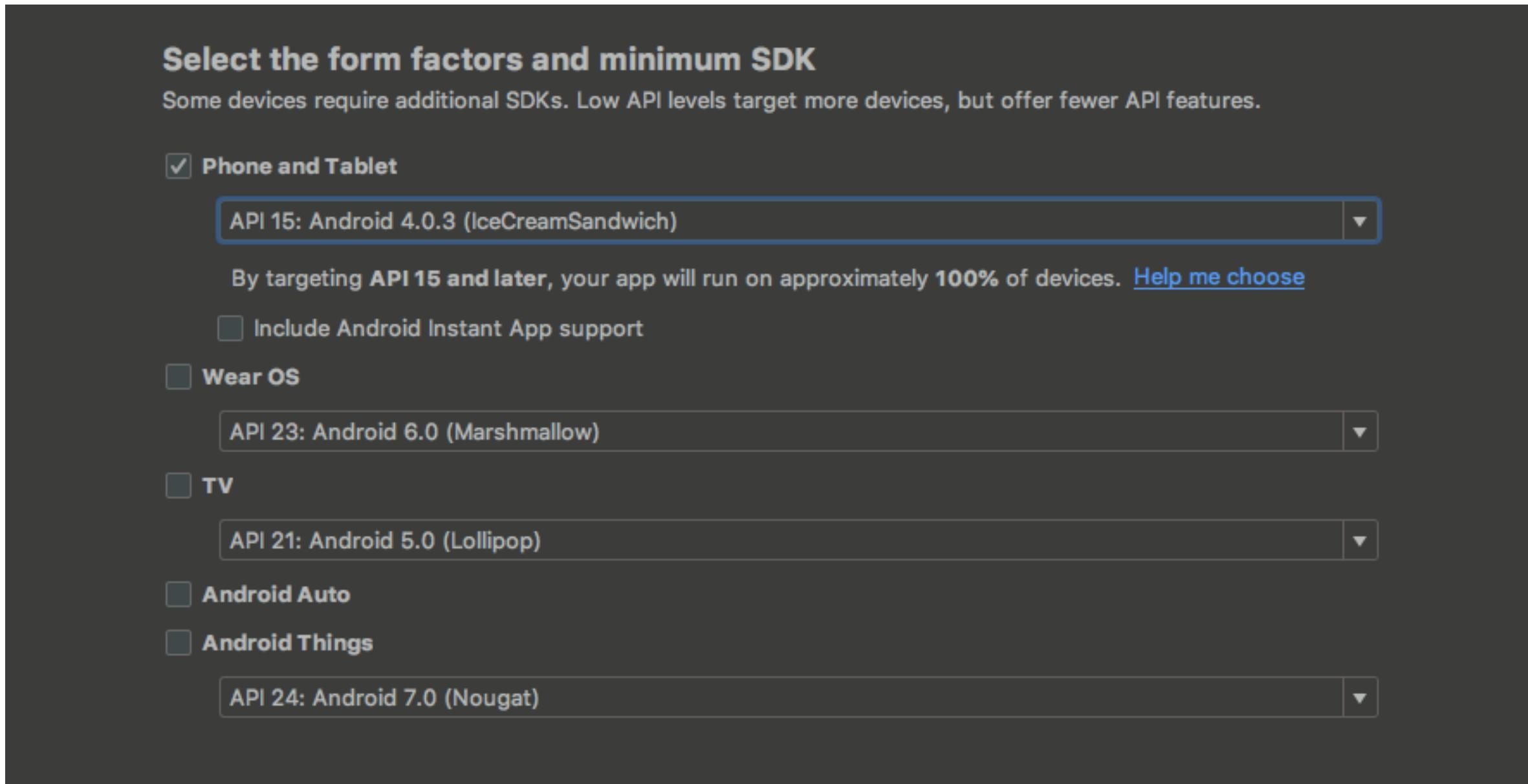


- Créer un nouveau projet android



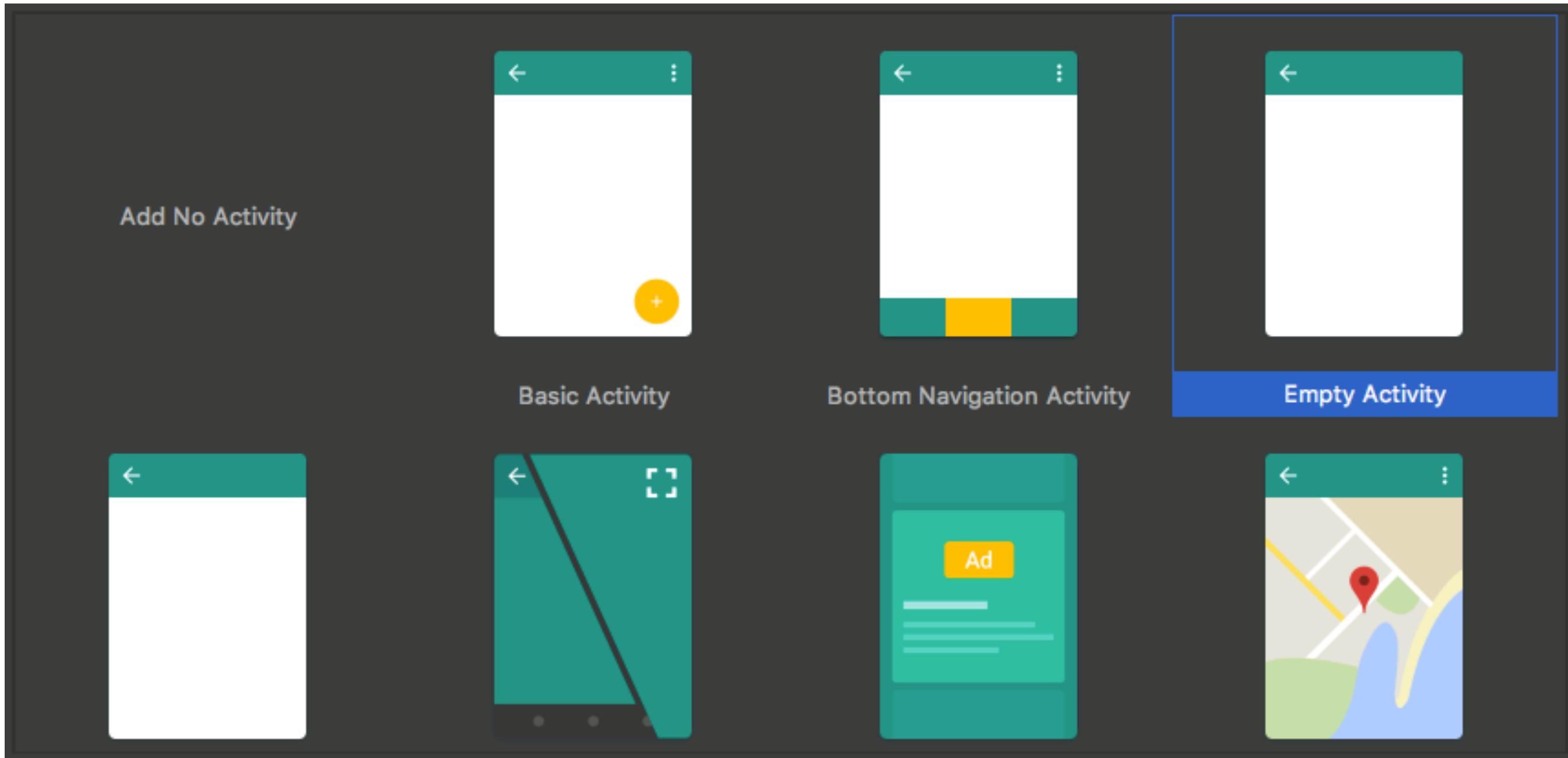


- Créer un nouveau projet android



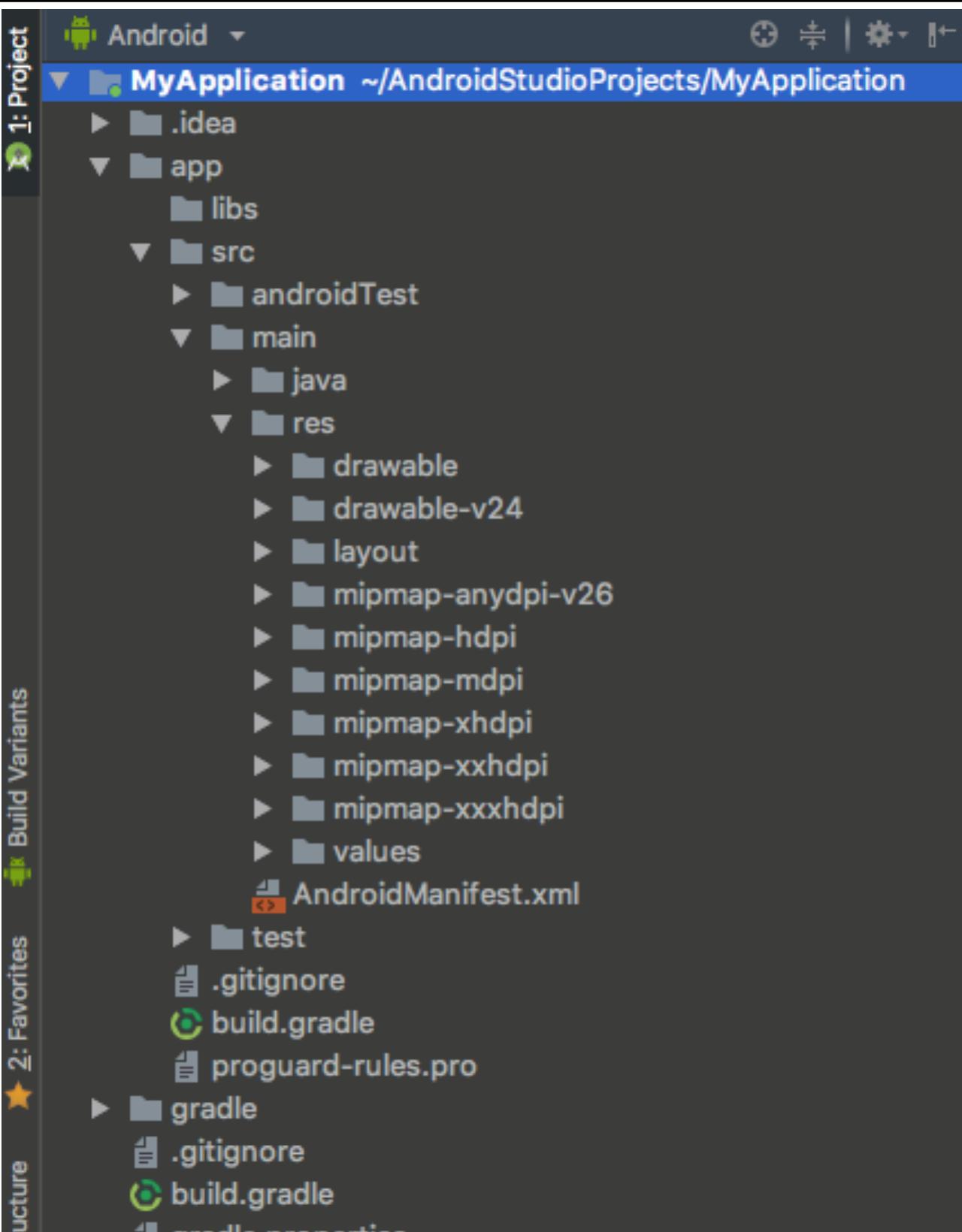


- Créer un nouveau projet android



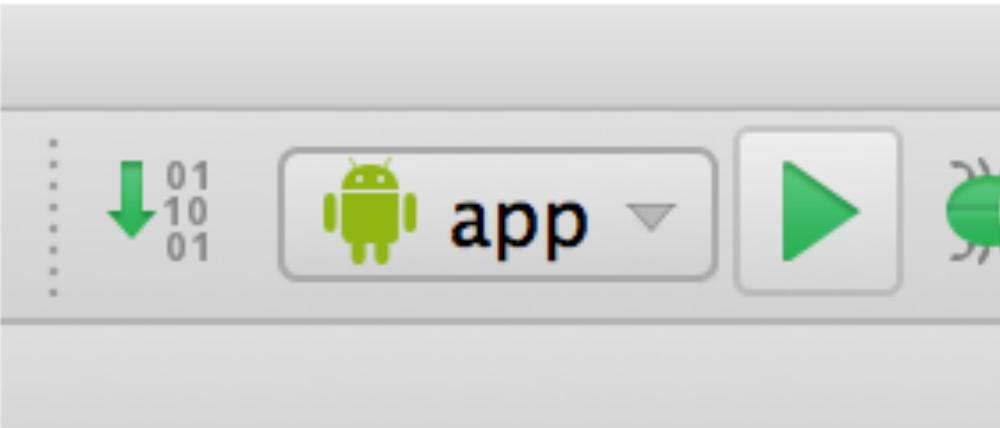


- Organisation des dossiers:
 - App: application
 - libs: librairies
 - Src/main
 - java: sources
 - res: ressources
 - Drawable: images
 - Layout: layout
 - Values: constantes
 - AndroidManifest.xml
 - build.gradle (module)
 - build.gradle (projet)





Exécuter





- **Settings.gradle (Racine):**
 - Liste des modules

```
include ':app', ':secondmodule'
```

- **Fichier build.gradle (Projet)**
 - Informations partagées entre tous les modules
 - Ex: version de gradle utilisée, repository maven, variables partagée entre tous les sous-modules....



- Fichier build.gradle (Module)
 - Les propriétés Android de l'application, anciennement présentes dans le Manifest :
 - **compileSdkVersion** le numéro de version d'android sdk utilisée pour compiler le projet
 - **buildToolsVersion** le nom complet de la version d'android sdk utilisée pour compiler le projet
 - **applicationId** l'identifiant unique de l'application (son nom de package complet)
 - **minSdkVersion** la version minimum d'android supportée
 - **targetSdkVersion** la version d'android pour laquelle l'application a été compilée
 - **versionCode** le numéro de version
 - **versionName** le nom complet de la version



- Dépendance avec un module

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:21.0.3'  
    compile project(':secondmodule')  
}
```

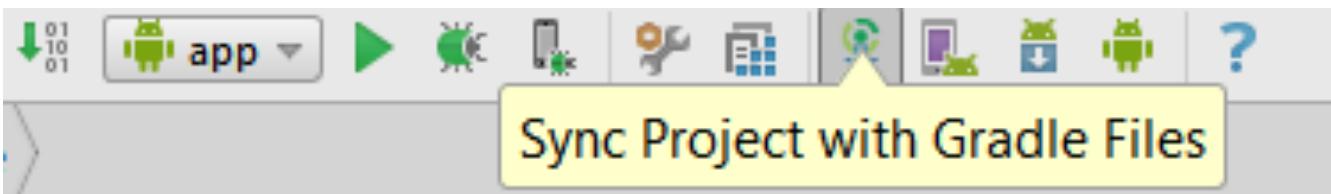
- Librairie Maven

ex: <https://github.com/square/picasso>

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:21.0.3'  
    compile 'com.squareup.picasso:picasso:2.5.0'  
}
```



- Astuce
 - Si vous mettez à jour le fichier Gradle, il est recommandé de cliquer sur le bouton « Sync Project with Gradle Files; »





- Permet de debugger et tracer les messages (system, user, ...)
- Affiche les messages en temps réel
- Ligne de commande: adb logcat
- IDE (Android Monitor)
 - Pour afficher (fenêtre principale)
 - **View > Tool Windows > Android Monitor.**



Introduction à Android Studio :: logcat



The screenshot shows the Android Studio Logcat interface. At the top, it displays the device/emulator configuration: "Emulator Nexus_5X_API_23 Android 6.0, API 23". Below this, the application name is set to "com.example.android.displayingbitmaps (2496)". The log level is set to "Verbose". A search bar and a "Regex" checkbox are also present. A button labeled "Show only selected application" is visible. On the far left, there is a vertical toolbar with various icons for clearing, filtering, and navigating through the log.

```
0(0B) LOS objects, 12% free, 27MB/31MB, paused 427us total 24.480ms
04-25 06:52:57.605 2496-2496/com.example.android.displayingbitmaps I/art: Starting a blocking GC Explicit
04-25 06:52:57.617 2496-2496/com.example.android.displayingbitmaps I/art: Explicit concurrent mark sweep GC freed 3(96B) AllocSpace objects,
0(0B) LOS objects, 12% free, 27MB/31MB, paused 352us total 11.588ms
04-25 06:52:57.626 2496-2496/com.example.android.displayingbitmaps I/art: Starting a blocking GC Explicit
04-25 06:52:57.635 2496-2496/com.example.android.displayingbitmaps I/art: Explicit concurrent mark sweep GC freed 3(96B) AllocSpace objects,
0(0B) LOS objects, 12% free, 27MB/31MB, paused 382us total 8.195ms
04-25 06:52:57.644 2496-2496/com.example.android.displayingbitmaps E/StrictMode: class com.example.android.displayingbitmaps.ui
    .ImageGridActivity; instances=2; limit=1
                                                android.os.StrictMode$InstanceCountViolation: class com.example
    .android.displayingbitmaps.ui.ImageGridActivity; instances=2; limit=1
                                                at android.os.StrictMode.setClassInstanceLimit(StrictMode.java:1)
04-25 06:53:42.232 2496-2496/com.example.android.displayingbitmaps I/art: Starting a blocking GC Explicit
04-25 06:53:42.248 2496-2496/com.example.android.displayingbitmaps I/art: Explicit concurrent mark sweep GC freed 270(16KB) AllocSpace objects,
0(0B) LOS objects, 12% free, 27MB/31MB, paused 384us total 15.932ms
04-25 06:53:42.252 2496-2496/com.example.android.displayingbitmaps I/art: Starting a blocking GC Explicit
```

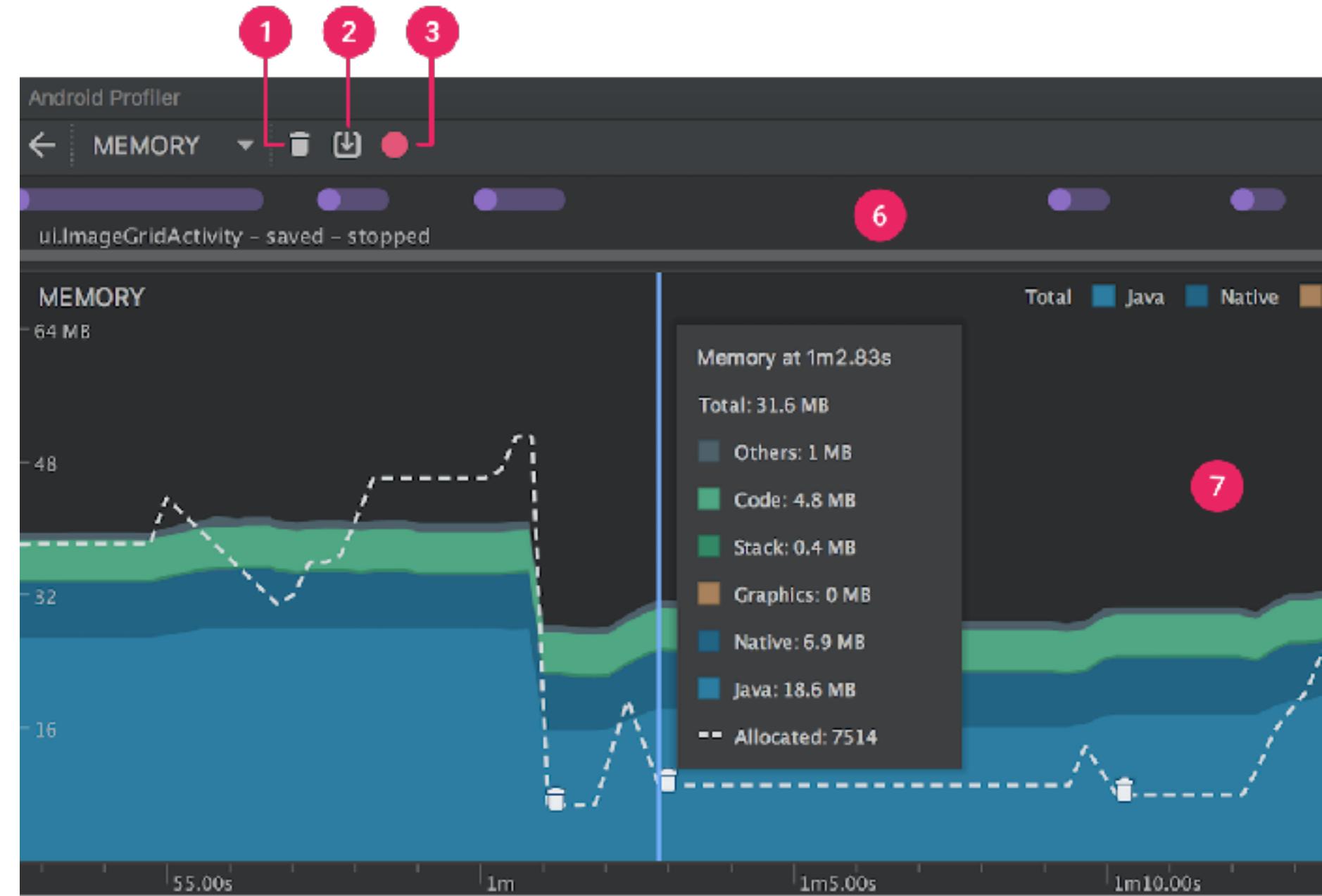


Un log message est composé d'un TAG et d'une priorité

- Tag: « String » Nom de l'activité, classe courante qui a envoyé le message
- Priorité:
 - V → Verbose (lowest priority)
 - D → Debug
 - I → Info
 - W → Warning
 - E → Error
 - A → Assert
- Ex: Log.d(tag, message);

Mesure la performance d'une App:

- Activité CPU
- Allocations Mémoire
- Traffic réseau
- Consommation énergétique





- Permet d'analyser le contenu d'un fichier APK
- Accessible en ligne de commande `apk analyzer`
- Comprendre la composition du fichier DEX
- Visualiser la version finale des fichiers de l'APK
- Comparaison entre 2 versions d'un APK

The screenshot shows the APK Analyzer interface with the following data:

File	Raw File Size	Download Size	% of Total Download
classes.dex	1,1 MB	1,1 MB	72,6%
res	345,2 KB	336,1 KB	21,9%
resources.arsc	219,7 KB	50,3 KB	3,3%
META-INF	33,1 KB	33,1 KB	2,2%
AndroidManifest.xml	1 KB	1 KB	0,1%



- ADB: Android Debug Bridge:
 - Communication avec l'émulateur
 - Communication avec les devices Android
 - Intégré avec l'IDE
 - Utilisable en ligne de commande
 - ADB est composé 3 éléments:
 - Client qui envoie les commandes (poste de développement)
 - Daemon (adb) qui tourne sur le device
 - Serveur qui gère la communication entre le client et le daemon.



- Activer ADB sur votre mobile
 - Il faut autoriser le debuggage USB (system settings > Developper Option)
 - Pour les Versions > Android 4.2: (Settings > About phone > “tap” plusieurs fois sur Build number → Developper Option devrait apparaître.
 - Sur certains Device l’écran Developper Options se trouve ailleurs ou est nommé différemment → se reporter à la documentation du téléphone.



- Lancer ADB
 - > adb start-server
- Liste des terminaux
 - > adb devices

```
# Lists all devices
adb devices
#Result
List of devices attached
emulator-5554 attached
emulator-5555 attached
# Issue a command to a specific device
adb -s emulator-5554 shell
```



- Arrêter ADB
 - > adb kill-server
- Copie de fichiers
 - > adb pull / adb push

```
// assume the gesture file exists on your Android device
adb pull /sdcard/gestures ~/test
// now copy it back
adb push ~/test/gesture /sdcard/gestures2
```

- Désinstaller une application

```
adb uninstall <packagename>
```



- Installer une application
 - > adb install [fichier apk]
- Shell
 - > adb shell
 - permet d'exécuter les commandes de base Linux: ls, cd, mkdir, rm, ...
- Récupérer des informations système
 - adb dumpsys (mémoire utilisée, applications, ...)
 - Information d'une application en particulier

```
adb shell dumpsys meminfo <package.name>
```

- Exécuter un screenshot
 - > adb shell screencap /sdcard/screen.png
 - > adb pull /sdcard/screen.png

Les applications

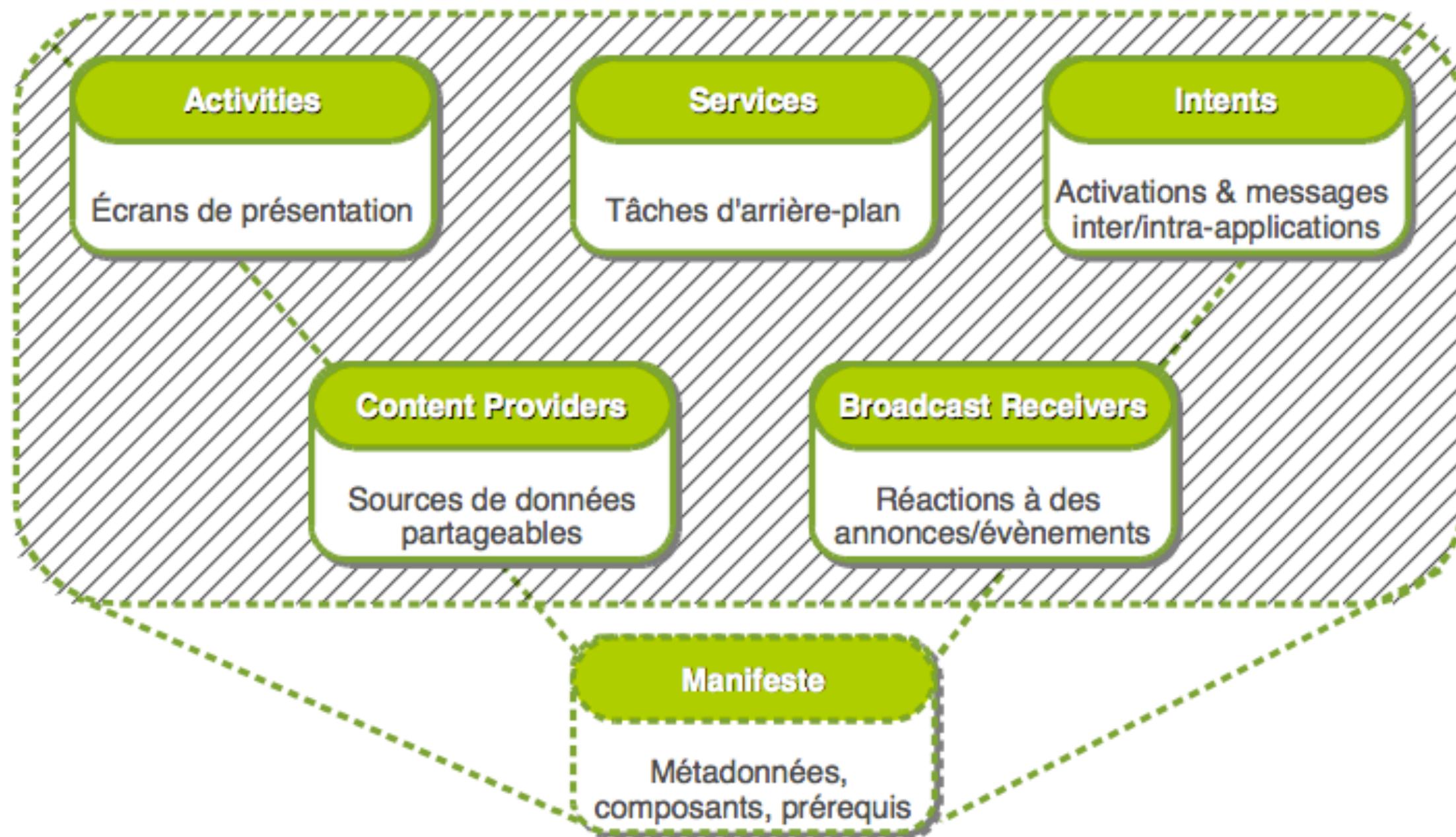


- 2 parties :
 - Les activités : des fenêtres interactives
 - Les services : tâches de fond.
- Les applications tournent dans leurs “SandBoxes”
- Communications entre applications : Les “intent”
 - Intent = intention : formule une demande
- Plusieurs composants peuvent répondre à un “intent”.



- Dernière couche sur Android
- Plusieurs sont intégrées dans le système :
 - Ecran "Home"
 - Gestion des Emails
 - Gestion des SMS/MMS
 - Gestion de la téléphonie
 - Google Maps...
 - Application supplémentaires installables
 - Toutes les applications sont écrites via le même SDK !

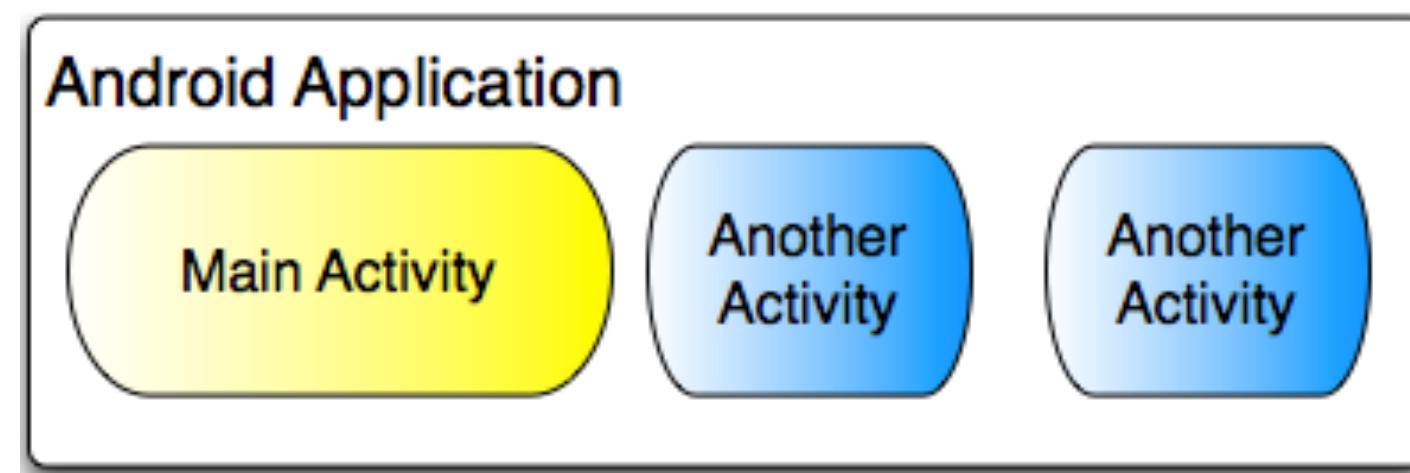
Les applications :: Éléments Fondamentaux





- Une activité ("Activity") = une IHM pour une action utilisateur précise, par ex:
 - Liste d'éléments parmi lesquels l'utilisateur peut choisir
 - Affichage d'une image avec un titre
 - Affichage d'un calendrier pour choisir une date
- Exemple d'une application de SMS :
 - Une activité pour choisir un contact
 - Une autre pour écrire le message
 - Une autre pour afficher un historique d'échanges.
- Chaque activité est indépendante des autres
- Une activité doit hériter de la classe : android.app.Activity

- Une application est donc un ensemble d'activités
- On doit définir quelle est la première activité à exécuter lors du lancement de l'application
- Pour naviguer dans l'application chaque activité doit elle-même lancer l'activité suivante.



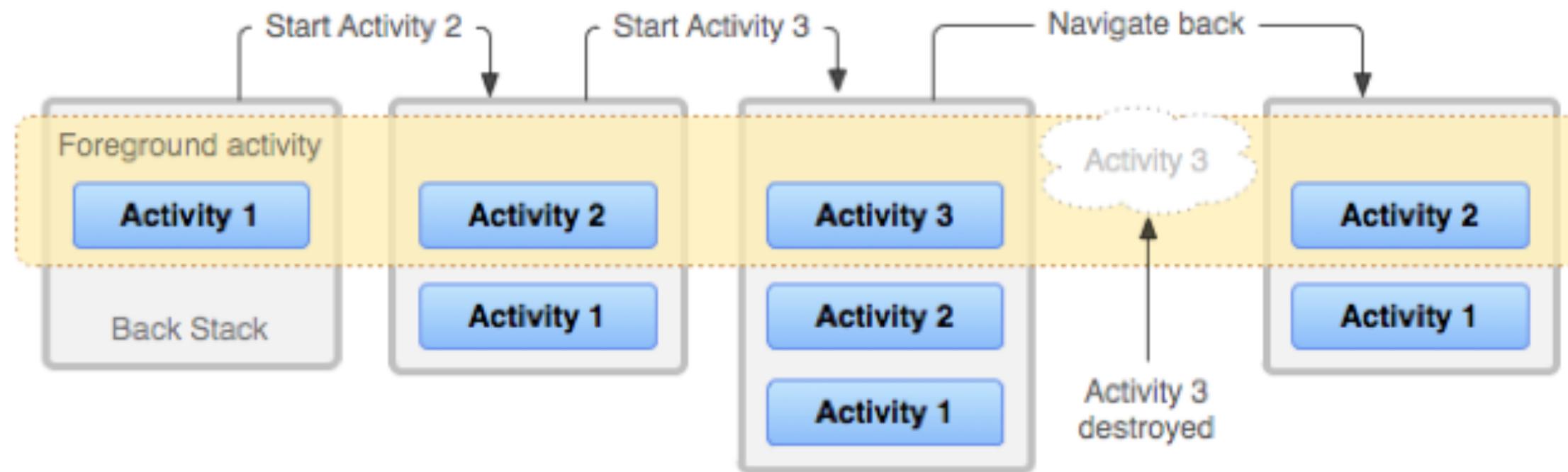


- Chaque activité est assignée à une fenêtre
 - Plein écran
 - Fenêtre flottante
- Une activité peut aussi posséder des sous fenêtres
 - Pop-up ...



- Pour pouvoir être lancée, toute activité doit être préalablement déclarée dans le “Manifest”
 - Une activité est désignée comme activité initiale de l'application
 - Ceci est indiqué dans le fichier manifest
 - Lancer une activité
 - Méthode startActivity(...)
 - Lancer une activité en vue d'obtenir un résultat en retour
 - Méthode startActivityForResult(...)

- Les activités sont empilées / dépilerées
 - Empilée quand une activité démarre
 - Dépilerée (i.e. détruite) quand on presse le bouton 'BACK'



- Une pression sur le bouton 'HOME' ne dépile pas l'activité.
 - Elle passe simplement en arrière plan



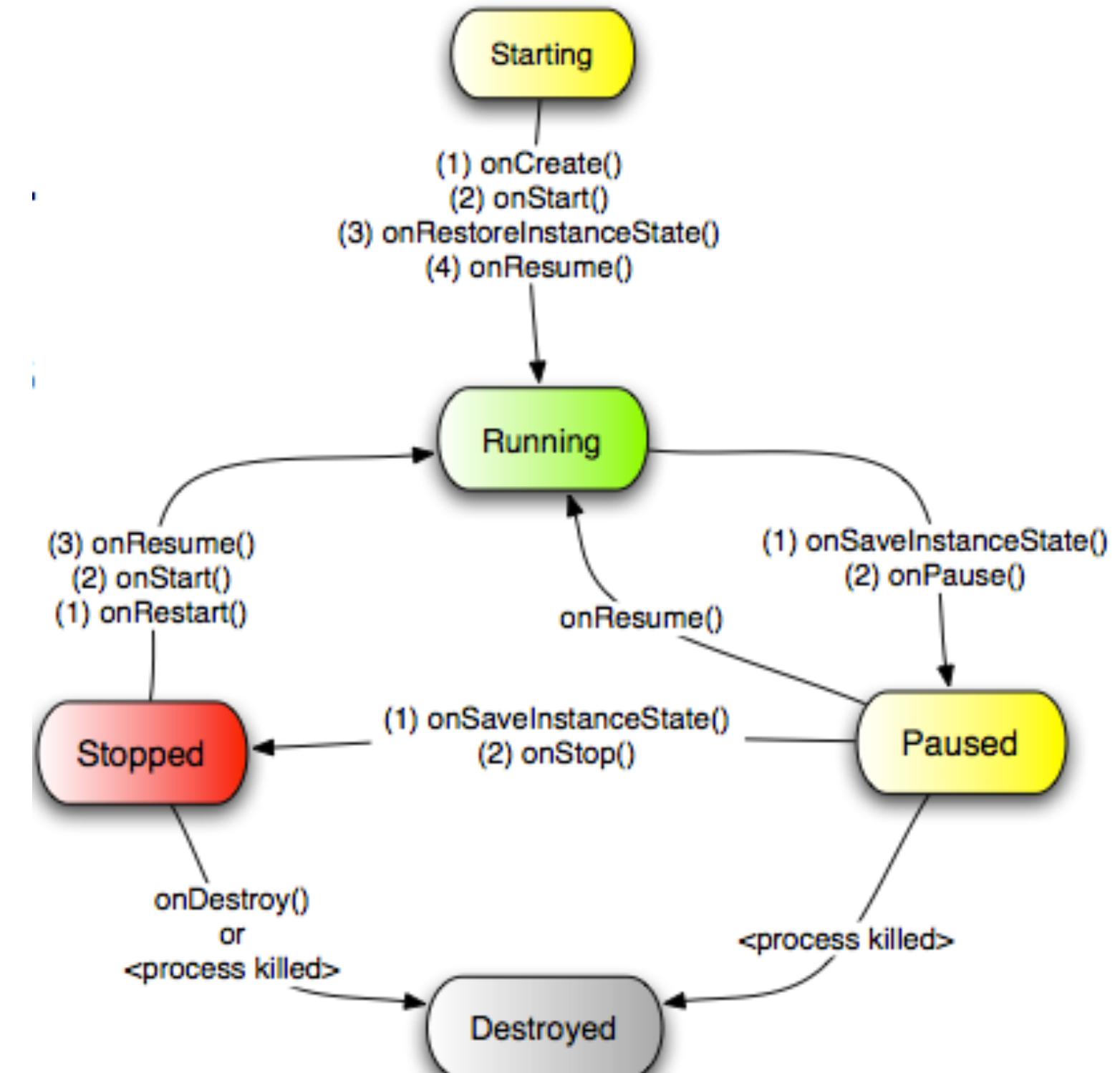
- Une activité possède trois états :
 - Active (running) : Quand l’activité est au premier plan et reçoit les actions utilisateur.
 - Paused : Quand elle est toujours visible mais n’a pas le focus (autre activité transparente par dessus ou activité ne prenant pas tout l’écran)
 - Toujours vivante
 - Mais peut être tuée en cas de ressources très limitées
 - Stopped : Quand elle n’est plus visible
 - Toujours vivante
 - Mais sera tuée dès que des ressources seront nécessaires.



- Le système tue les activités en état “stopped” (ou “paused”) de deux manières :
 - En appelant la méthode `finish()`
 - En tuant le processus tout simplement
- Quand l’activité sera à nouveau demandée :
 - Doit être complètement reconstruite
 - Doit Potentiellement recharger son dernier état

- Une activité est notifiée de ses changements d'état par l'appel à ses méthodes :

- void onCreate(Bundle savedInstanceState)
- void onStart()
- void onRestart()
- void onResume()
- void onPause()
- void onStop()
- void onDestroy()



- Afin de sauvegarder le contexte, le système appelle "onSaveInstanceState()" avant de rendre l'application potentiellement tuable (paused...)
- Cet appel fournit un bundle "clé/valeurs" pour que le développeur puisse sauvegarder l'état
- Au prochain appel de "onCreate()" ce bundle sera fourni
- Il est également fourni via un appel à "onRestoreInstanceState()"
- L'appel à la sauvegarde n'est fait qu'en cas de risque de terminaison de l'activité par le système et non si cela vient d'une action utilisateur (back)
-

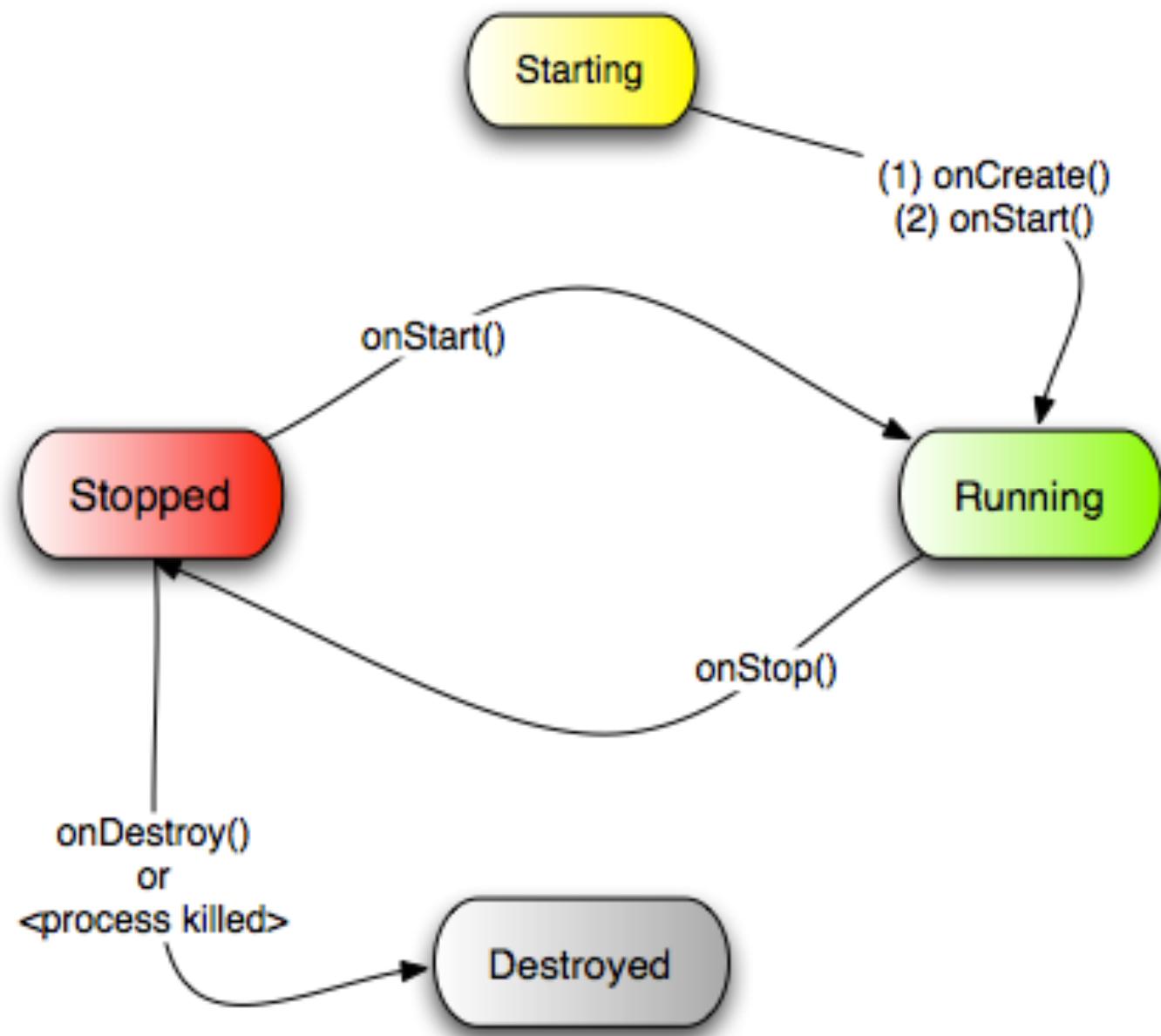


- Un service ne possède pas d'interface, aucune interaction utilisateur
- On les utilise pour des tâches répétitives ou longues en background
 - Lecture de musique
 - Collecte de données
 - Suivi GPS
 - Vérification de mise à jour
 - ...
- Attention, les services s'exécutent sur le Thread principal de l'application: UI Thread
- Ne réagissent pas au changement d'activité



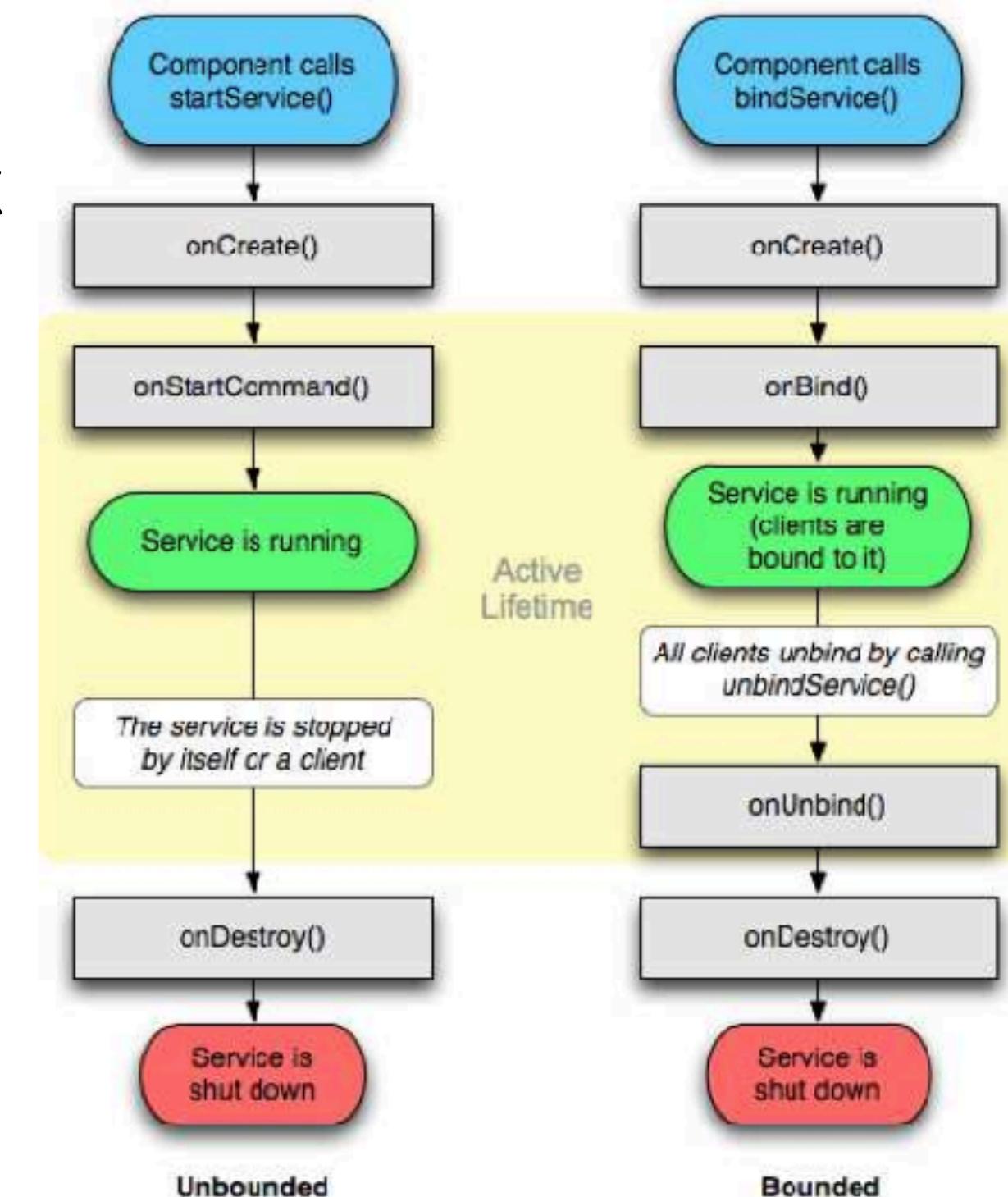
- Mode *Unbounded*
 - Un composant démarre et arrête le service.
 - Un service “Unbound” s’exécute en background, même si le composant qui a initié le service est détruit.
- Opérations
 - startService(...)
 - stopService(...)
- Mode *Bounded*
 - Le service est attaché à un composant. Ils établissent une connexion permanente afin d’interagir avec un service par le biais d’une interface
 - Un service “Bound” s’exécute tant que le service et le composant sont liés.
 - Opérations
 - bindService(...)
 - unbindService(...)

- Cycle de vie



5 méthodes de callback

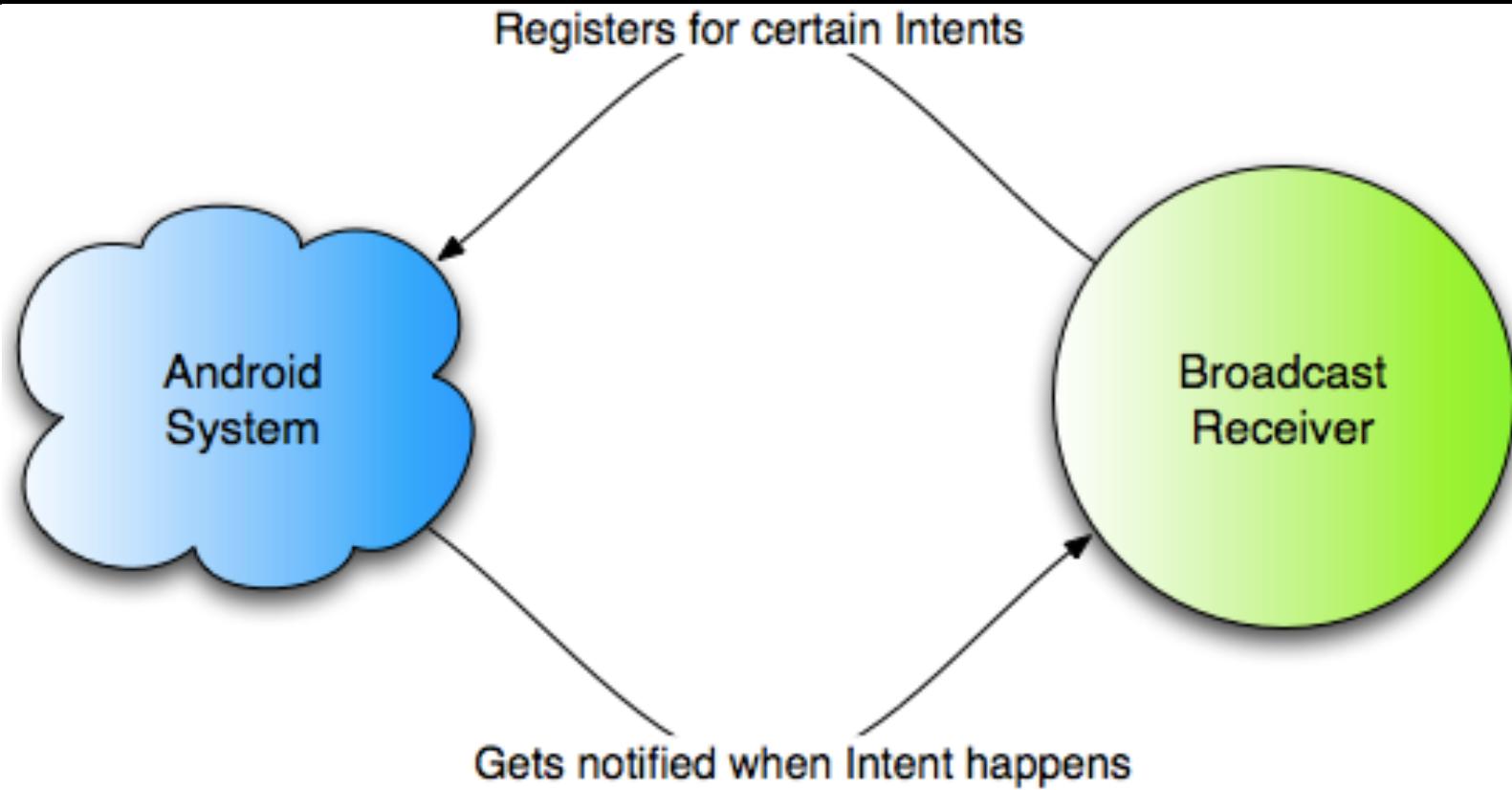
- `onCreate()`
- `onStartCommand()`
- `onDestroy()`
- `onBind()`
- `onUnbind()`





- Étendent la classe "Service"
- Surchagent les méthodes onBind, onCreate, onStartCommand, onDestroy
- Les services doivent être déclarés dans le Manifest
- Pour lancer un service:
 - Intent intent = new Intent(this, UpdaterService.class);
startService(intent);
- Pour arrêter un service:
 - Intent intent = new Intent(this, UpdaterService.class);
stopService(intent);
- On peut voir les services actifs **Manage application > running** et en entrant dans l'application, on peut tuer un service

Les applications :: Broadcast Receiver

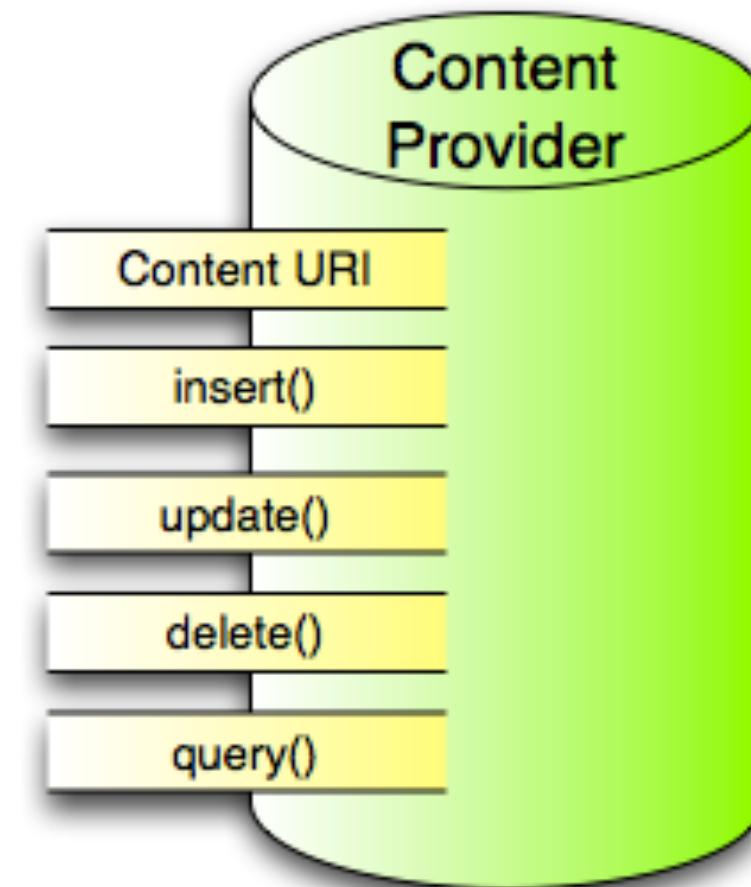


- Les broadcast receiver sont :
 - Des éléments inactifs qui attendent un évènement
 - Il y a différents évènements système :
 - Batterie faible
 - Changement de langue du système
 - L'utilisateur a pris une photo
 - ...
- Il est possible de définir ses propres évènements
- < 10 secondes
- Ils héritent de la classe `android.content.BroadcastReceiver`



- Une application peut contenir plusieurs receivers : un par événement important
- Les receivers n'ont évidemment pas d'interface. Ils peuvent lancer des activités en cas de besoin
- Un receiver s'abonne/désabonne via le fichier manifest ou programmatiquement
- Ils peuvent également utiliser le NotificationManager pour signaler quelque chose à l'utilisateur (préférable)
 - Icône, vibration, alerte sonore, clignotement diode...

- Les content provider permettent de rendre une partie des données d'une application accessible aux autres applications
 - Seul moyen pour un partage de données entre applications
- Les données sont donc exposées par une classe héritant de android.content.ContentProvider
 - Méthodes query(), Insert(), Update(), delete()...
- Les données sont accessibles grâce à l'URI dont le schème dédié est 'content'.
 - Exemple: *content://contacts/people/125*

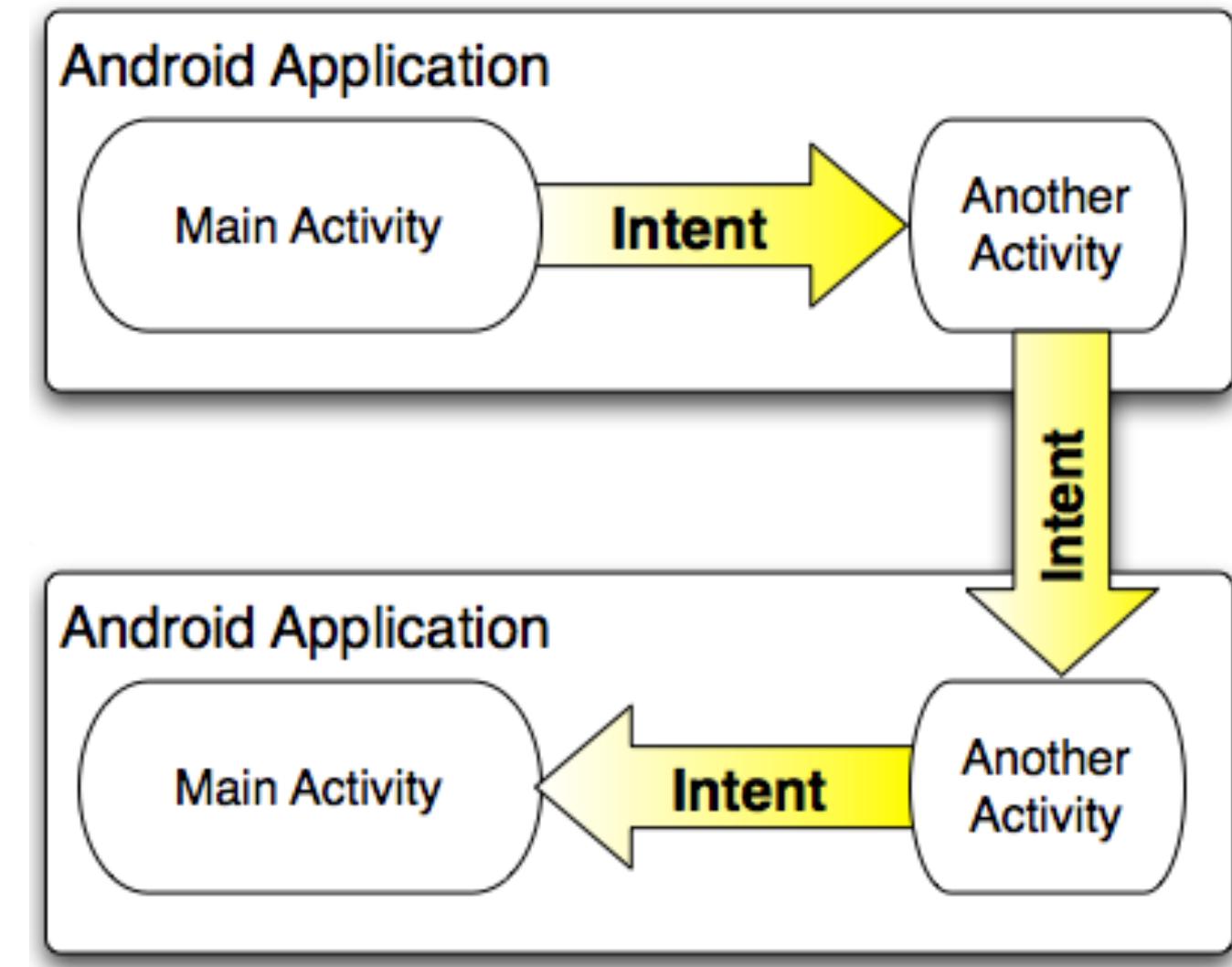




- Les autres applications n'accèdent pas directement à la classe du ContentProvider
- Utilisation d'un ContentResolver qui va rediriger les requêtes vers le provider voulu
- Si l'on tente d'accéder à une ressource d'une application n'étant pas en cours d'exécution le système Android se charge de la lancer avant.

Les applications :: Intent

- Les content providers sont activés par une requête d'un content resolver
- Mais les 3 autres systèmes (Activity, Service, BroadCast Receiver) sont activés par des messages asynchrone appelés "Intent" ou Intention
- Un intent dérive de android.content.Intent
- Un intent est constitué de:
 - une action à réaliser,
 - Donnée sur laquelle réaliser l'action sous forme d'URI ou type MIME
 - Paramètre optionnels (EXTRA)





- Pour les activités et les services il nomme l’action désirée et précise l’URI des données sur lesquelles agir.
 - Afficher / image
 - Editer / texte
 - ...
- Pour les broadcast receivers il se contente de nommer l’action à annoncer
 - Batterie faible



- Les Intents et les activités :
 - Lancement en passant un Intent en paramètre à une des méthodes suivantes :
 - Context.startActivity()
 - Activity.startActivityForResult()
 - L’activity peut accéder à celui ci avec :
 - getIntent()
 - Si le système doit envoyer des nouveaux intents :
 - Appel de onNewIntent() sur l’activité
 - En cas de résultat attendu
 - Appel de onActivityResult() sur l’activité appelante



- Les Intents et les services :
 - Lancement en passant un Intent en paramètre à la méthode suivante :
 - Context.startService()
 - Le système appellera ensuite la méthode onStart() en précisant cet Intent en paramètre
 - Connexion en passant un Intent en paramètre à la méthode suivante :
 - Context.bindService()
 - Le Système appellera ensuite la méthode onBind() en précisant cet Intent en paramètre



- Les Intents et les Broadcast receiver :
 - Une application voulant envoyer un évènement va utiliser une des méthodes suivantes :
 - Context.sendBroadcast()
 - Context.sendOrderedBroadcast()
 - Context.sendStickyBroadcast()
 - Le système va alors appeler la méthode onReceive() sur tous les broadcast receivers intéressés en passant en paramètre l'Intent.



- **Les Intents Explicites:**
 - Spécifie la classe du service / Activity / Broadcast Receiver que l'on souhaite lancer
 - Le composant à exécuté est connu à l'avance
- **Les Intents Implicites:**
 - On ne connaît pas le composant à invoquer (autre package, ...)
 - Dans ce cas on accède au composant en spécifiant l'ACTION
 - C'est le système qui va se charger de trouver le composant adéquat, en comparant le contenu de l'Intent à l'intent filter déclaré dans le Manifest.
 - Si l'Intent « matche » l'intent filter, le système lance le composant et lui délivre l'Intent.
 - Si plusieurs Intent filter correspondent à l'objet le système propose une boîte de dialogue afin que l'utilisateur choisisse l'application à utiliser.



- Comment créer un Intent Explicite?
 - Un Intent Explicite transporte l’information dont Android a besoin pour déterminer quel composant démarrer:
 - Nom du composant à activer
 - « Extra Data » qui seront transmises au composant

The screenshot shows a code editor window titled "Android Explicit Intent Example". The code is written in Java and demonstrates how to start a service using an explicit intent. The code consists of five numbered lines:

```
1 // Executed in an Activity, so 'this' is the Context
2 // The fileUrl is a string URL, such as "http://www.slidenerd.com/logo.png"
3 Intent downloadIntent = new Intent(this, DownloadImageService.class);
4 downloadIntent.setData(Uri.parse(fileUrl));
5 startService(downloadIntent);
```



- Comment créer un Intent Implicit?
- Si on ne spécifie pas le composant à démarrer, l’Intent est dit « Implicite ».
- Un Intent Implicite est composé de 4 parties:
 - ACTION
 - DATA
 - CATEGORY
 - EXTRAS



- ACTION
 - Chaîne de caractère qui spécifie l'action que l'on souhaite exécuter.
 - Android dispose d'un certain nombre d'actions au travers de son API, mais il est possible de définir ses propres actions.
 - Actions prédéfinies: ACTION_VIEW, ACTION_CALL, ...
 - Custom action: ex: eu.corellis.myapp.DOWNLOAD



- **Category**
 - Une chaîne de caractère précisant quel type de composant peut gérer l'intent.
 - Plusieurs catégories peuvent être précisées.
 - Exemples :
 - CATEGORY BROWSABLE : Le contenu peut être Affiché dans le navigateur
 - CATEGORY HOME : L'activité est de type Home
 - CATEGORY LAUNCHER : par ex l'activité principale d'une application appartient à cette catégorie, pour indiquer qu'elle apparaît dans le menu du launcher
 - CATEGORY PREFERENCE : l'activité est un panneau de préférences



- DATA
 - Un objet URI et/ou le type MIME des données
 - L'action est suffisante dans la plupart des cas, mais dans le cas de l'action « ACTION_EDIT », les données contiennent l'URI du document à éditer.
 - setData() : pour « setter » seulement l'URI.
 - setType() : pour « setter » seulement le type MIME
 - **Toujours utiliser setDataAndType()** pour « setter » les 2 types, setData() et setType() écrasent l'autre paramètre.



- **EXTRAS**
 - Les paramètres additionnels sont envoyés à l’Intent sous la forme de clé/valeur.
 - Certaines actions requièrent des paramètres « EXTRAS ».
 - Il est possible d’utiliser putExtra() en spécifiant la clé – valeur ou d’utiliser un Bundle qui contient l’ensemble des clés – valeur avec la méthode putExtras().
 - Par exemple, il est possible d’envoyer un email avec ACTION_SEND en remplissant le sujet avec la clé EXTRA_SUBJECT, le to avec EXTRA_EMAIL, ...



- Quelques exemples :
 - ACTION_VIEW content://contacts/people/1 – Affiche les informations sur le contact 1
 - ACTION_CALL content://contacts/people/1 – Affiche le mode d'appel rempli avec les informations du contact 1
 - ACTION_VIEW tel:123 – Affiche le mode d'appel rempli avec "123". (ACTION VIEW s'adapte donc au contenu)
 - ACTION_CALL tel:123 – Idem
 - ACTION_EDIT content://contacts/people/1 – Permet de modifier les informations du contact 1
 - ACTION_VIEW content://contacts/people/ – Affiche la liste des contacts



- Une application peut se déclarer auprès du système comme récepteur possible d'un Intent. On utilise un filtre (Intent Filter) ou récepteur d'intention -> ils informent le système à quel « intent » les composants peuvent réagir.
- Un composant peut avoir plusieurs filtres
- Chaque composant peut définir ses filtres d'intention au niveau du manifest ou dynamiquement dans le code:
 - ACTION : Quelles actions sont supportées ?
 - DATA : Pour des données de quelle nature?
 - CATEGORY : Dans quelles circonstances ?
- Permet au bus de savoir si le message d'activation (i.e. l'intent) doit être délivré au composant ou non
- **Attention ! Un filtre dédié à chaque Intention**

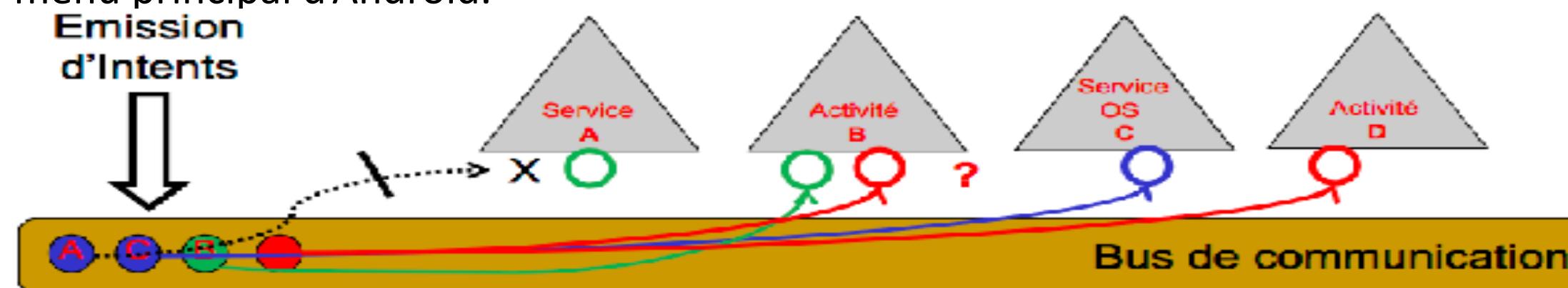
Les applications :: Intent - Filtres d'intention

- Exemple de filtre:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    http://schemas.android.com/apk/res/android>  
    <application>  
        <activity android:name=".Now" android:label="Now">  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>  
    </application>  
</manifest>
```

L'élément intent-filter sous l'élément activity annonce que :

- Cette activité est l'activité principale de cette application.
- Elle appartient à la catégorie LAUNCHER, ce qui signifie qu'elle aura une icône dans le menu principal d'Android.



Les applications :: Manifest



- Fichier XML
- Précise l'architecture de l'application
- Chaque application doit avoir un fichier AndroidManifest.xml à la racine du projet

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.marakana"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".HelloAndroid"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="5" />
</manifest>
```





- Déclaration des droits dans AndroidManifest.xml
- Spécifier les ressources utilisées
 - Informer l'utilisateur sur l'application
 - ✓ <application> ... </application>
 - Sécuriser le droit d'accès aux ressources
 - ✓ <uses-permission android:name="" />
 - Utiliser ce qui est nécessaire
 - ✓ <uses-feature android:name="" android:required="" />
 - Ainsi que d'autres spécifications
 - ✓ <use-library>, <use-sdk>, ...



- Contenu :
 - Précise le nom du package java utilisant l'application.
Cela sert d'identifiant unique !
 - Il décrit les composants de l'application:
 - Liste des activités, services, broadcast receivers
 - Précise les classes qui les implémentent
 - Précise leurs capacités (à quels intents ils réagissent)
 - Ceci permet au système de savoir comment lancer chaque partie de l'application afin de satisfaire au principe de réutilisabilité.



- Contenu (suite) :
 - Définit les permissions de l’application:
 - Droit de passer des appels
 - Droit d'accéder à Internet
 - Droit d'accéder au GPS ...
 - Précise la version d’Android minimum nécessaire
 - Déclare les librairies utilisées
 - Déclare des outils d’Instrumentation (uniquement pour le développement)



- Conventions :
 - Seuls deux éléments sont obligatoires
 - < manifest > : contient le package, la version... Englobe tout le fichier
 - < application > : décrit l'application et contiendra la liste de ses composants.
 - Les données sont passées en tant qu'attribut et non en tant que contenu
 - Tous les attributs commencent par "android:" (sauf quelques un dans < manifest >)



- Les ressources
 - Au lieu de contenir les données en tant que tel le fichier manifest peut faire appel à des ressources
 - < activityandroid : icon = "@drawable/smallPic"... >
 - Ces ressources sont définies dans le répertoire « res » de l'application.



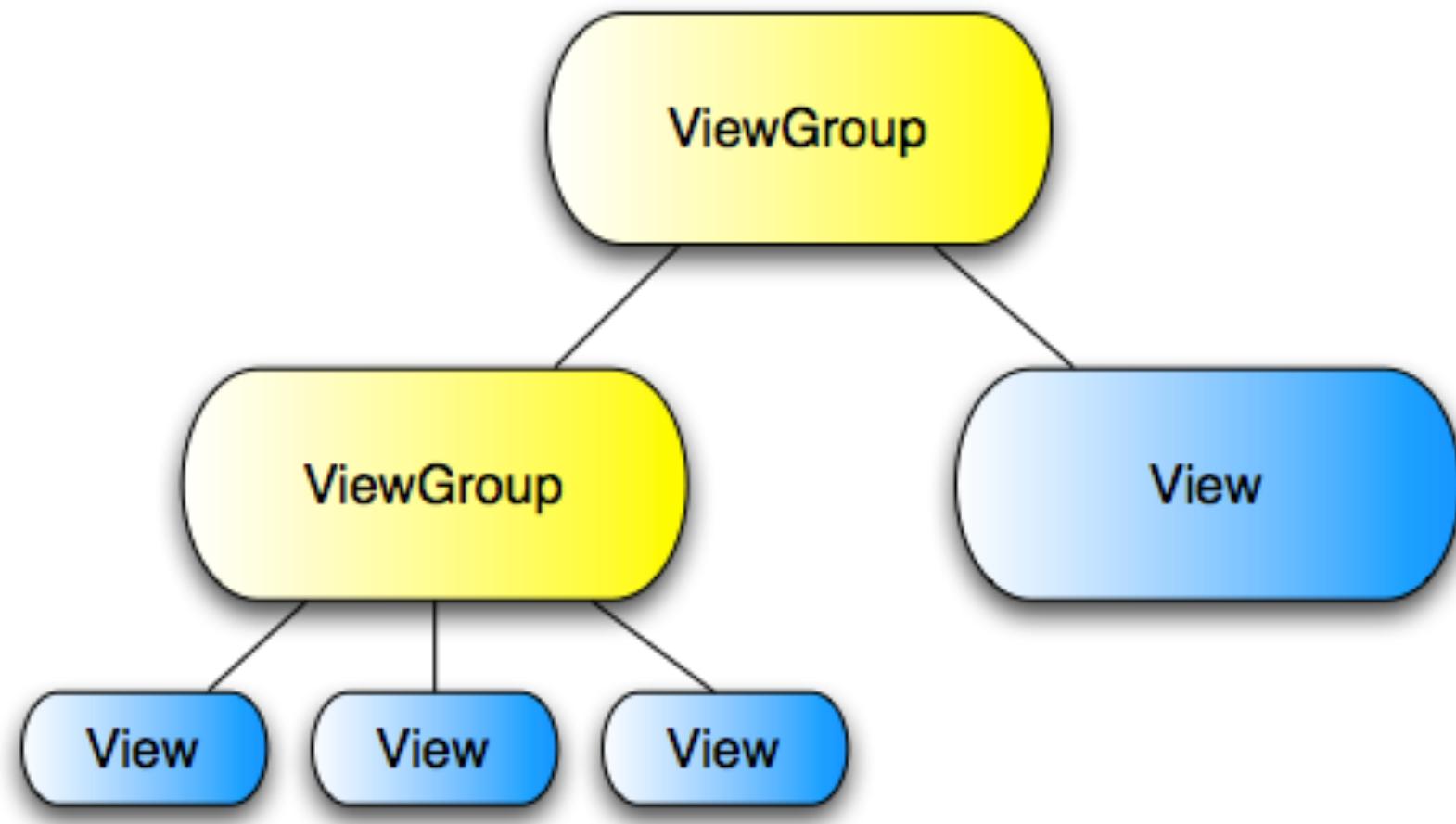
- Permissions
 - Une application ne peut utiliser certaines fonctionnalités à moins que cela ne soit précisé dans le fichier Manifest
 - Il faut donc préciser les permissions nécessaires grâce à :
 - <uses permission>
 - Il existe des permission standard :
 - android.permission.CALL EMERGENCY NUMBERS
 - android.permission.READ OWNER DATA
 - android.permission.SET WALLPAPER
 - android.permission.DEVICE POWER
 - Il est possible de définir ses propres permissions

Interface graphique utilisateur (GUI)



- Une API Java riche
 - Des layouts et des widgets (appelés “Vues”)
- Programmation déclarative XML
 - Sépare la vue du code métier
- Fonctionnalité de personnalisation
 - Hériter et redéfinir un widget de base
 - Combiner des widgets existants
- Rendu 2D/3D
 - OpenGL, Renderscript

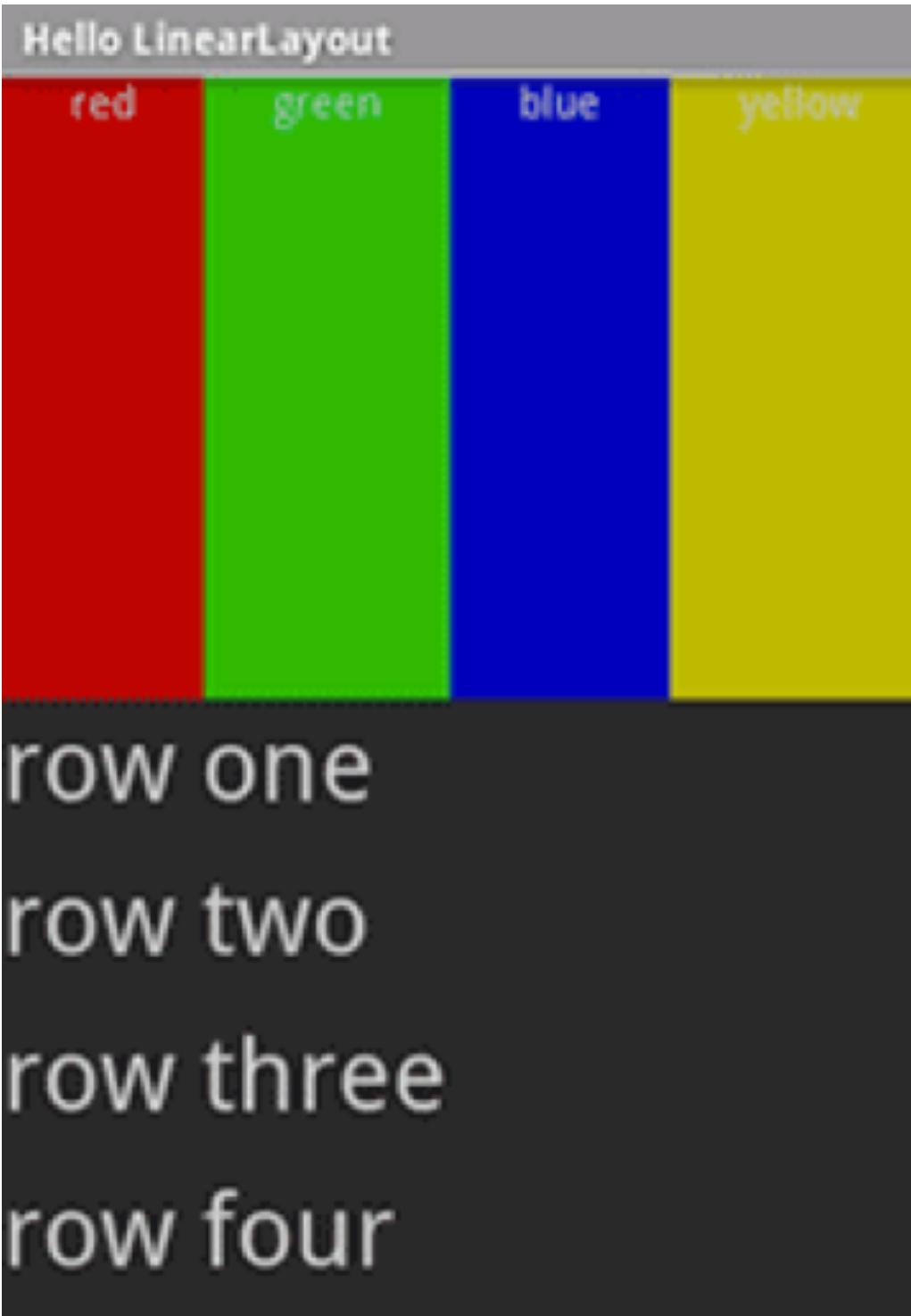
- Les vues et les layouts



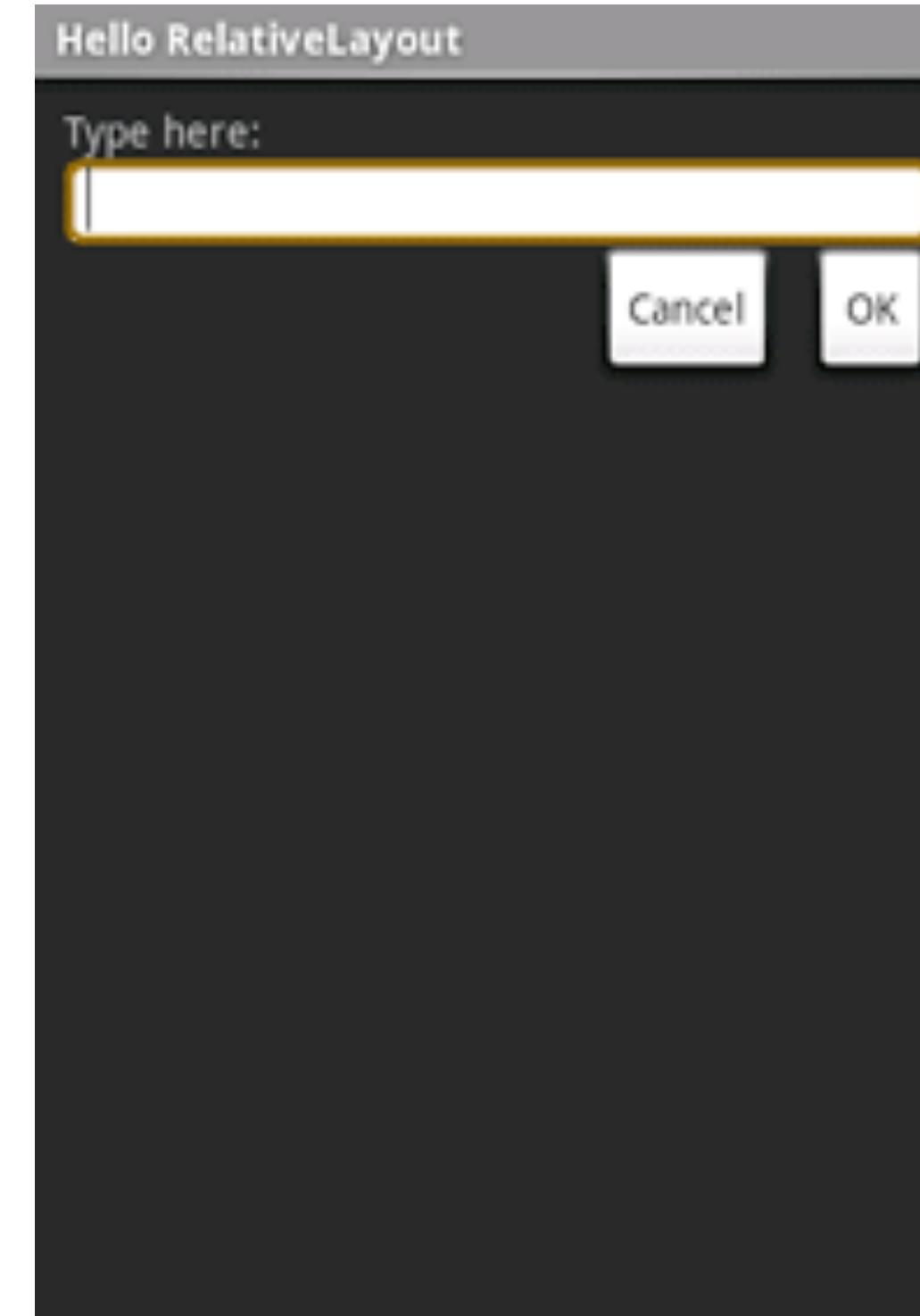


- Sous Android, la notion de mise en page est reliée à la notion de « Layout ». Cette dernière représente l’agencement des différents éléments graphiques dans votre interface.
- Voici quelques exemples de layouts :
 - LinearLayout
 - FrameLayout
 - RelativeLayout
 - TableLayout
 - GridView
 - TabHost

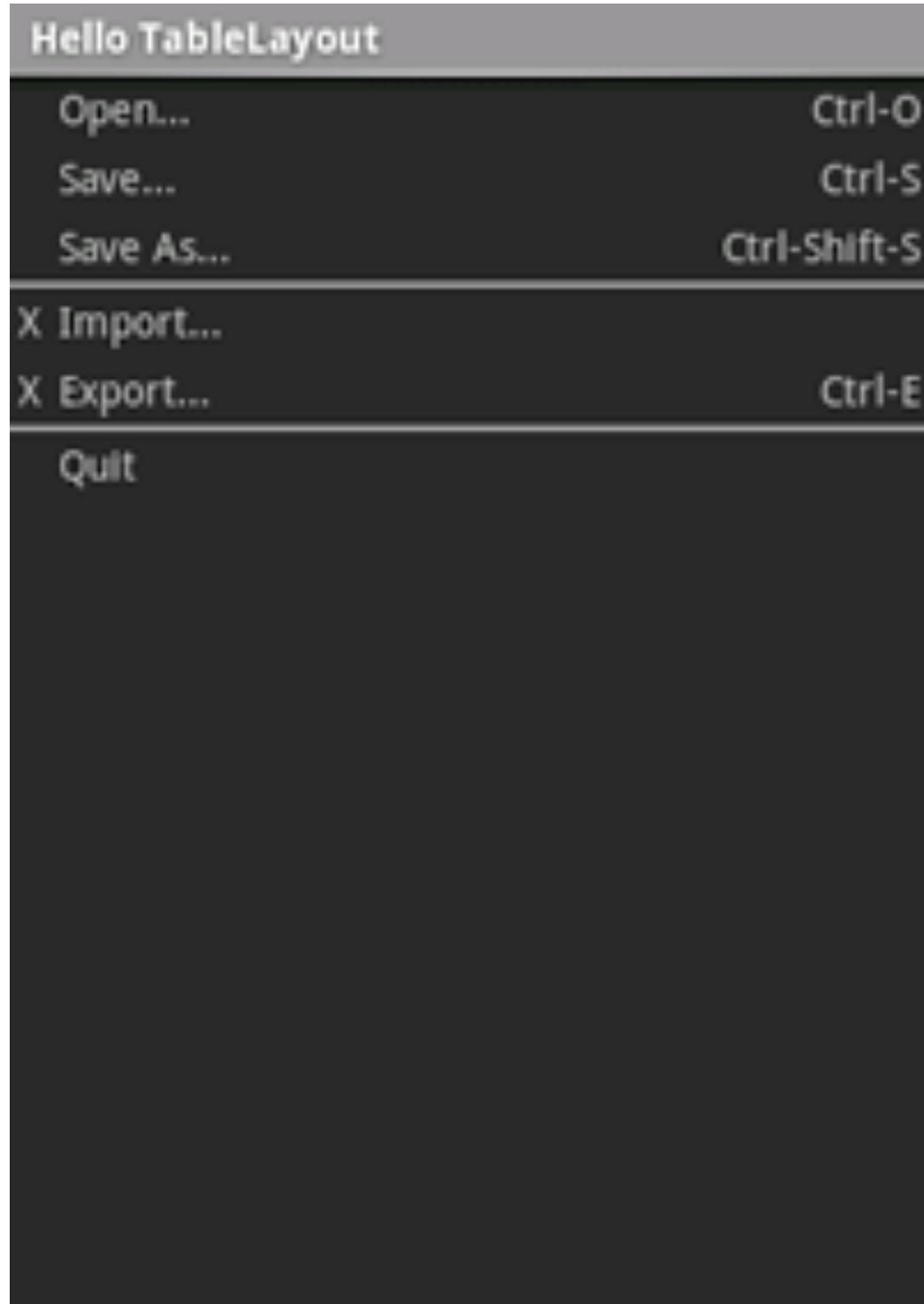
- **LinearLayout**



- **RelativeLayout**



- **TableLayout**



- **FrameLayout**

- Ne contient qu'un seul élément (si on en met plusieurs ils se superposent)

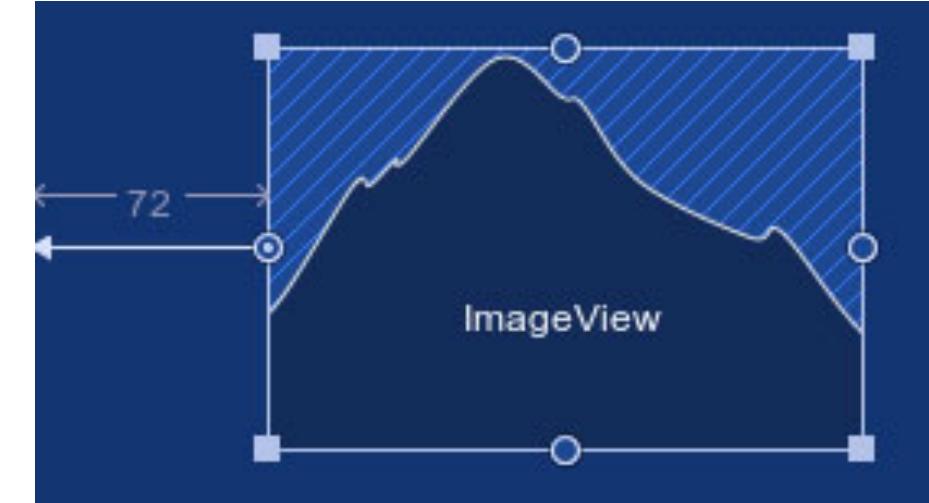
- **AbsoluteLayout**

- Pour placer les éléments par positions fixes

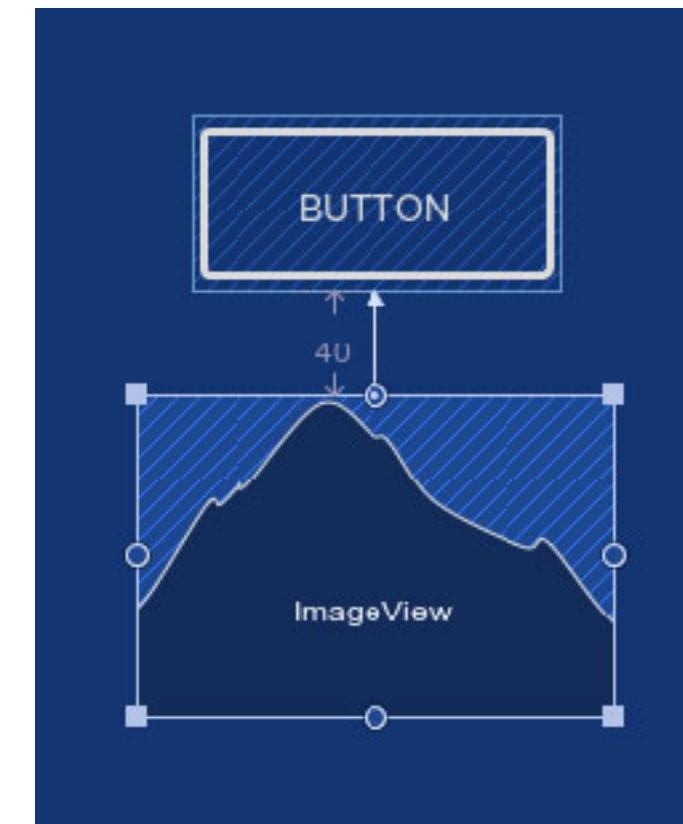
ConstraintLayout

- Min sdk 23
- Défini des contraintes pour chaque vue / composant enfant par rapport aux autres vues présentes
- Semblable à RelativeLayout

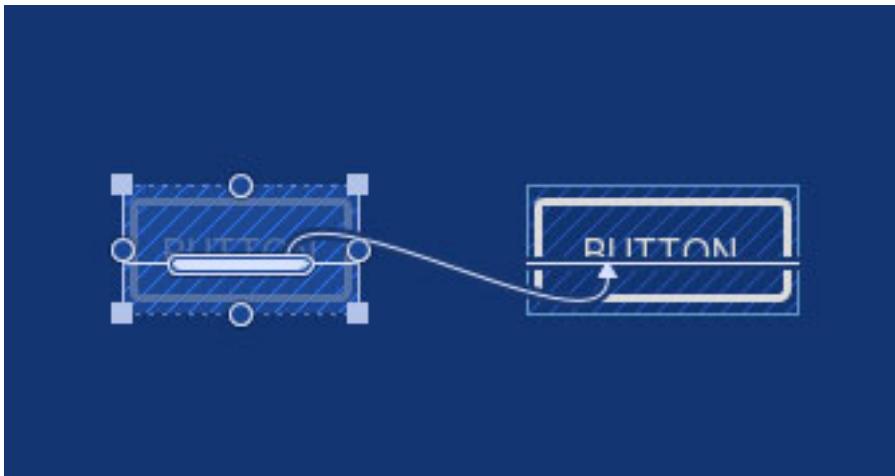
Défini une contrainte par rapport à son parent



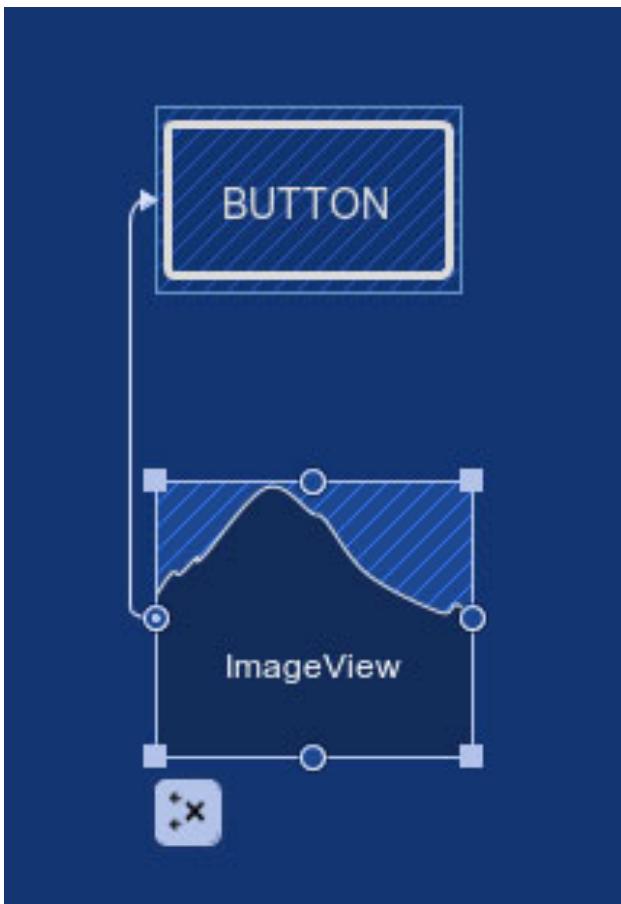
Contrainte par rapport à une autre vue



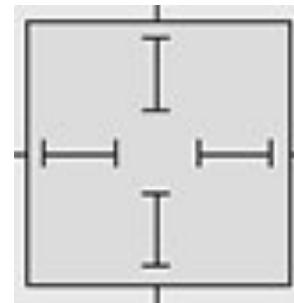
Contrainte d'alignement baseline



Contrainte alignement horizontal



Gestion des contraintes



☰ Fixed mode: vous devez spécifier une taille

☰ Wrap content: La vue gère la taille

☰ Any size: C'est l'équivalent de match_parent
Dans ce cas la vue va utiliser tout l'espace disponible



- **Orientation**
 - Sens de placement des vues dans un conteneur
 - android:orientation = vertical | horizontal
- **Taille**
 - Surface prise par la vue
 - android:layout_width/android:layout_height = <??>px | fill_parent | wrap_content
- **Gravité**
 - Alignement d'une vue dans son conteneur
 - android:layout_gravity = left | center_horizontal | top | bottom | right | ...



- **Poids**
 - Taux d'espace libre affectés à chaque widgets
 - `android:layout_weight = ?` (0 par défaut)
- **Espacement (intra)**
 - Espacement entre un contenu et les bords de sa vue
 - `android:padding? = top | left | right | bottom`
- **Espacement (inter)**
 - Espacement entre les vues
 - `android:layout_margin? = ??px`



- Utiliser des ressources depuis le code
 - La plupart des éléments de l'API sont prévus pour accepter des ressources Android en paramètre (int ressource)
 - `obj.setColor(R.color.rose_bonbon);`
 - `obj.setColor(android.R.color.darker_gray);`
- Référencer des ressources depuis d'autres ressources
 - `attribute="@[packageName]:resourcetype/resourceldent"`
 - `<EditText android:textColor="@color/rose_bonbon"/>`
 - `<EditText android:textColor="@android:color/darker_gray"/>`

- Il est nécessaire d'instancier les vues (i.e. obtenir des objets) pour pouvoir les manipuler
- Récupérer les widgets depuis le code
 - Grâce aux identifiants affectés à chaque vue
 - `Button myButton = (Button) findViewById(R.id.my_button);`
- Récupérer toute une vue depuis le code
 - Déserialiser (inflate) un fichier XML décrivant un layout ou un menu
 - `View my_view = LayoutInflater.inflate(R.layout.main, null);`
 - `MenuInflater.inflate(R.menu.control, my_menu);`

- Qu'est ce qu'un Listener?

- Un listener se traduit par un « écouteur ».
- Écoute une source jusqu'à ce qu'elle soit utilisée.
- Pour qu'une action soit réalisée, il faut qu'un émetteur envoie un signal dans le canal d'écoute pour que le Listener se déclenche.

- Principe des écouteurs

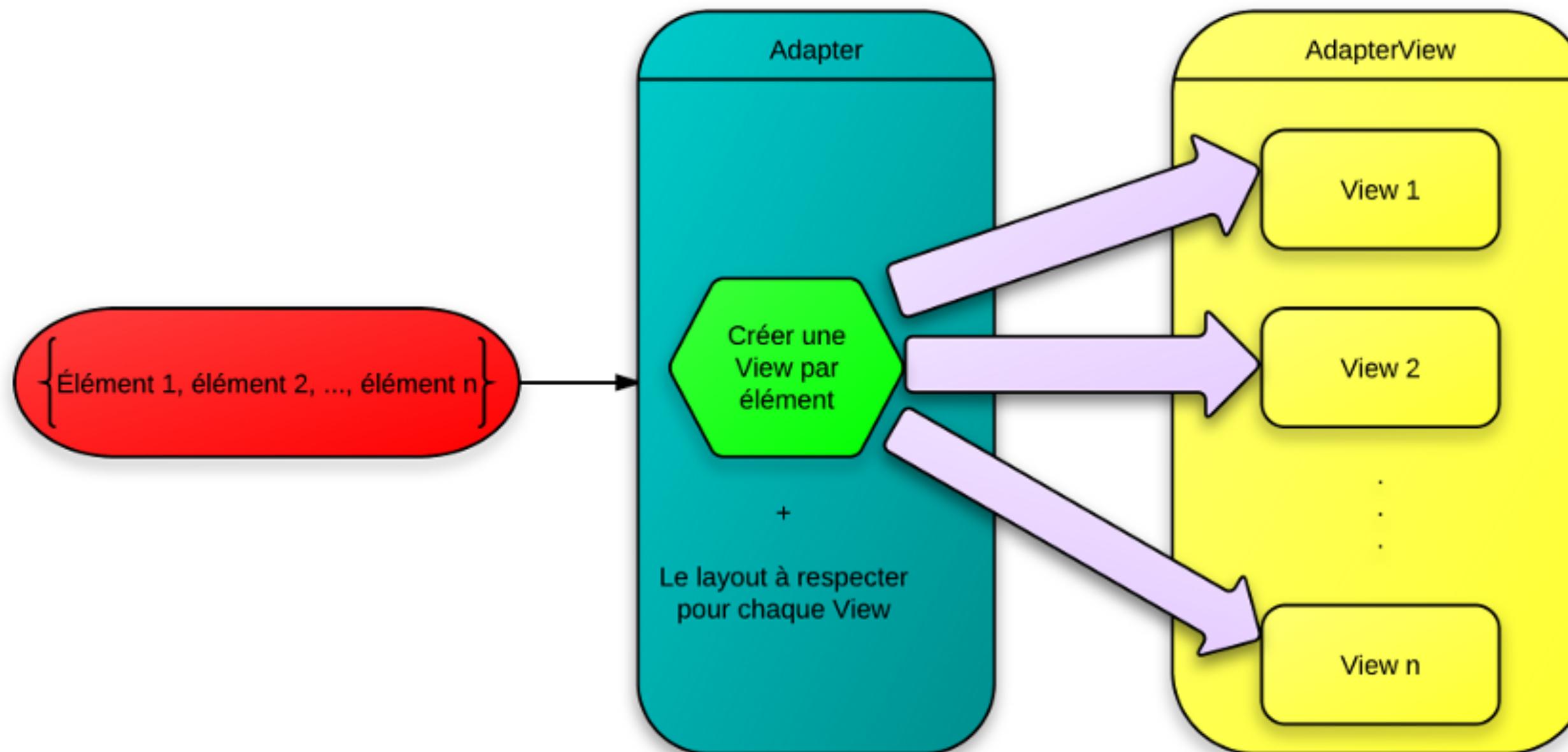
- Fournir une implémentation respectant un contrat (interface) afin de réagir à des types d'évènements particuliers :
 - Gestion du KeyPad (tap, trackball)
 - OnClickListener, OnLongClickListener
 - onClick(View), onLongClick(View)
 - OnKeyListener
 - onKeyUp(KeyEvent), onKeyDown(KeyEvent)
 - Gestion du TouchScreen (pression, gesture)
 - OnTouchListener
 - onTouchEvent(MotionEvent)

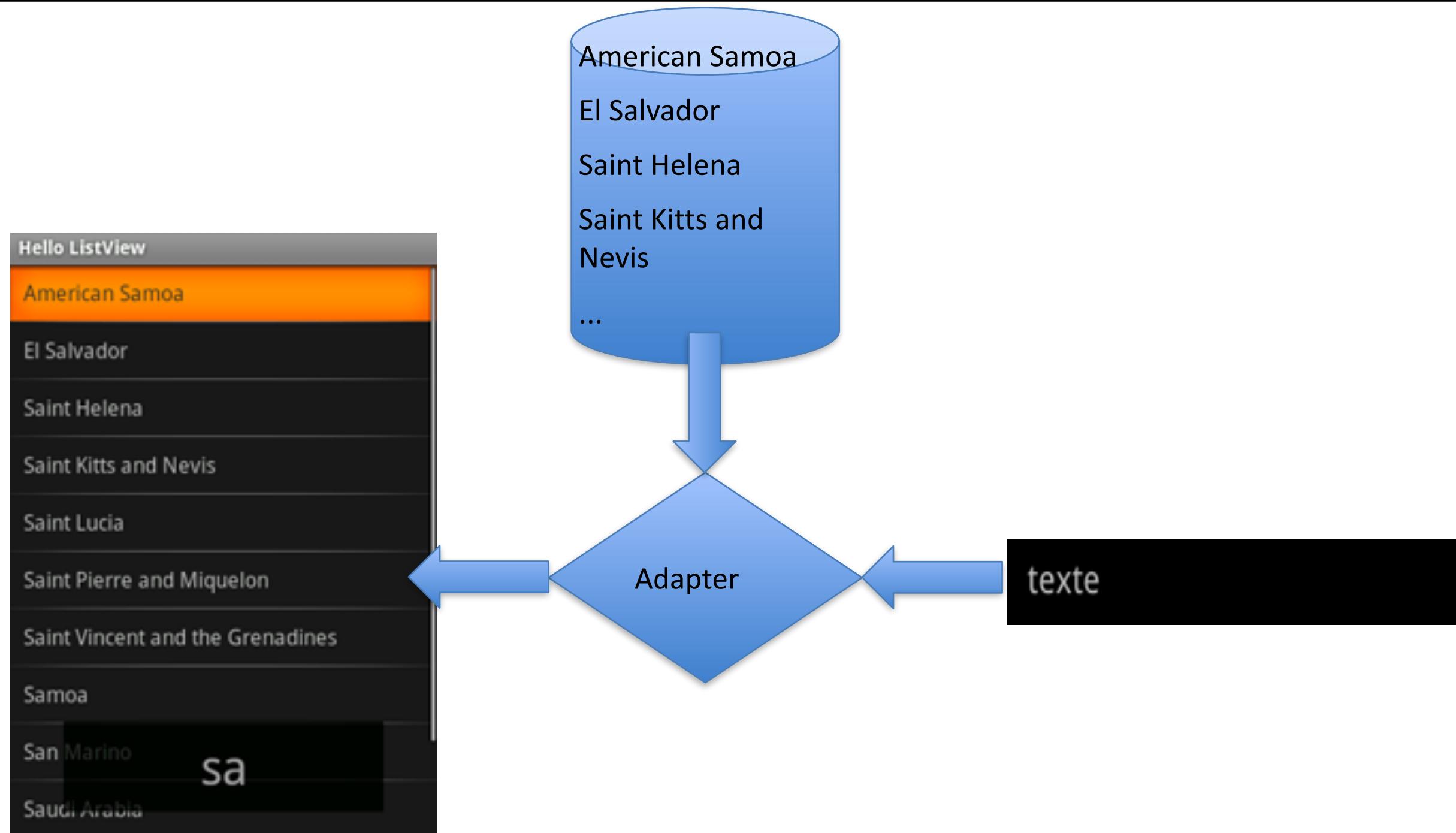


- Le « Toast » (le plus simple): permet d'afficher de petits messages rapides à l'utilisateur. Moins intrusif qu'une popup, mais réservé à des petites notifications.
- « Dialog »: *AlertDialog, ProgressDialog, DatePickerDialog ou TimePickerDialog*.
- « Personnaliser un Dialog »: Utilisation de la classe `LayoutInflater` qui va nous permettre de générer une « view » à partir d'un fichier xml et de l'affecter au « Dialog » grâce à la méthode `setView()`.
- Les « FragmentDialog »: utilisation recommandée par Google.



- Les adaptateurs sont des classes qui lient des données aux vues de l'UI
 - Les vues concernées étendent android.widget.AdapterView
- Responsables de la création des vues « enfants »
- Classes d'adaptateurs
 - Héritent de android.widget.BaseAdapter
 - Adapters natifs: SimpleAdapter, ArrayAdapter<?> (sert à récupérer des données stockées dans une collection)
 - Exploite par défaut la valeur de la méthode `toString()` des objets de la liste
 - CursorAdapter : sert à récupérer des données stockées dans une base de données relationnelle (SQLite)
 - Vous pouvez étendre ces classes de base pour gérer finement vos items (conseillé)





ArrayAdapter

- Utilise les types génériques pour lier une View Adapter à un tableau d'objet de la classe spécifiée
- Utilise la valeur `toString` du tableau pour remplir la `textView`
- Exemple d'appel :

```
ArrayAdapter<String> aa = new ArrayAdapter<String>(this, R.layout.list_item,  
COUNTRIES);  
  
myListView.setAdapter(aa);
```

SimpleAdapter

- Adapter simple pour lier des données à des vues

```
SimpleAdapter mSchedule = new SimpleAdapter (this.getBaseContext(), listItem,  
R.layout.layoutitem, new String[] {"img", "title", "description"}, new int[] {R.id.img,  
R.id.title, R.id.description});
```

- Contexte de la vue
- tableau d'objets
- Layout pour chaque item
- Les clefs de la hashmap (par exemple)
- Les ID des ressources correspondantes dans le layout

Custom Adapter

- Il est possible de définir ses propres adapters
- Étendre la classe BaseAdapter
- Surcharger les méthodes:
 - `getView`: génère la vue pour un objet
 - `getCount`: renvoie le nombre d'objets
 - `getItem`: renvoie l'item à la position
 - `getItemId`: retourne la position de l'item

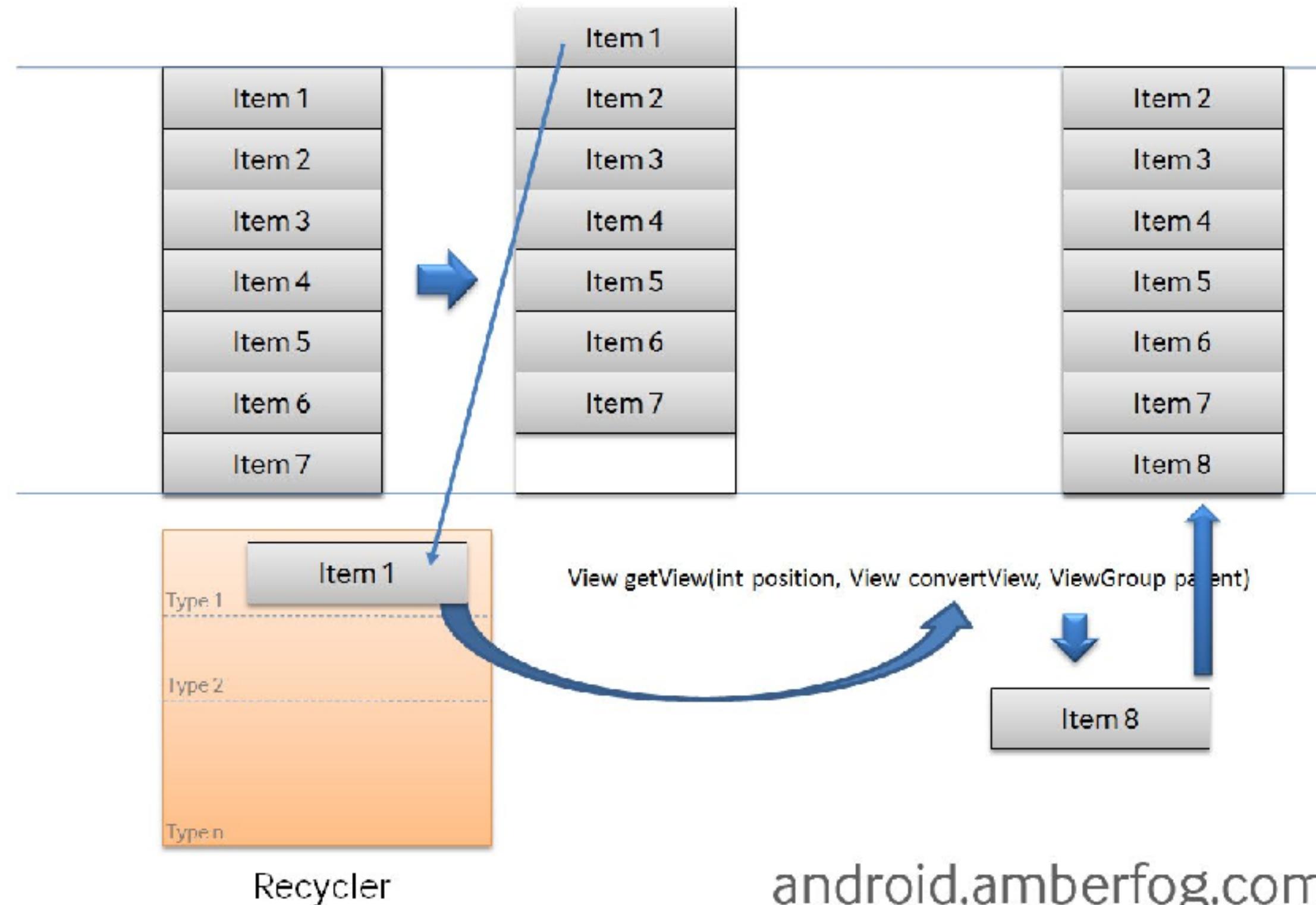


Problème sur les jeux de données importants:

- Mémoire
- Performance

Solution

- « populate on demand »
- Recycler les vues pour réduire la création d'objets



android.amberfog.com

GetView :: méthode non optimisée

```
1 public View getView(int position, View convertView, ViewGroup parent) {  
2     View item = mInflater.inflate(R.layout.list_item_icon_text, null);  
  
3     ((TextView) item.findViewById(R.id.text)).setText(DATA[position]);  
4     ((ImageView) item.findViewById(R.id.icon)).setImageBitmap(  
5             (position & 1) == 1 ? mIcon1 : mIcon2);  
  
6     return item;  
7 }
```

GetView :: La bonne méthode

```
1 public View getView(int position, View convertView, ViewGroup parent) {  
2     if (convertView == null) {  
3         convertView = mInflater.inflate(R.layout.item, parent, false);  
4     }  
  
5     ((TextView) convertView.findViewById(R.id.text)).setText(DATA[position]);  
6     ((ImageView) convertView.findViewById(R.id.icon)).setImageBitmap(  
7             (position & 1) == 1 ? mIcon1 : mIcon2);  
  
8     return convertView;  
9 }
```

```
static class ViewHolder {  
    TextView text;  
    ImageView icon;  
}
```

GetView :: La plus rapide

```
1 public View getView(int position, View convertView, ViewGroup parent) {  
2     ViewHolder holder;  
3  
4     if (convertView == null) {  
5         convertView = mInflater.inflate(R.layout.list_item_icon_text,  
6                                         parent, false);  
7         holder = new ViewHolder();  
8         holder.text = (TextView) convertView.findViewById(R.id.text);  
9         holder.icon = (ImageView) convertView.findViewById(R.id.icon);  
10  
11         convertView.setTag(holder);  
12     } else {  
13         holder = (ViewHolder) convertView.getTag();  
14     }  
15  
16     holder.text.setText(DATA[position]);  
17     holder.icon.setImageBitmap((position & 1) == 1 ? mIcon1 : mIcon2);  
18  
19     return convertView;  
20 }
```

Header & Footer

- `ListView.addHeaderView()`
- `ListView.addFooterView()`
- Ces méthodes doivent être appelées avant `setAdapter()`

List selector

- « highlight » l'item sélectionné
- Non visible en « touch mode » (ex: un téléphone avec un trackball)
 - Il n'y a aucune sélection dans ce mode
- Visible derrière les items de la liste
 - `android:drawSelectorOnTop="true"`



```
1 <selector>
2
3     <item android:state_window_focused="false"
4         android:drawable="@color/transparent" />
5
6     <item android:state_focused="true" android:state_enabled="false"
7         android:state_pressed="true"
8         android:drawable="@drawable/list_selector_background_disabled" />
9
10    <item android:state_focused="true" android:state_enabled="false"
11        android:drawable="@drawable/list_selector_background_disabled" />
12
13    <item android:state_focused="true" android:state_pressed="true"
14        android:drawable="@drawable/list_selector_background_transition" />
15    <item android:state_focused="false" android:state_pressed="true"
16        android:drawable="@drawable/list_selector_background_transition" />
17
18    <item android:state_focused="true"
19        android:drawable="@drawable/list_selector_background_focus" />
20
21 </selector>
```

*Il suffit ensuite de définir un background dans le fichier xml décrivant une cellule :
android:background="@drawable/listview_selector"*



- **android:transcriptMode**
 - Comportement de la liste quand le contenu change
 - “disabled” -> ne scrolle pas
 - “normal” -> scrolle vers le bas si le dernier élément est visible
 - “alwaysScroll” -> scrolle toujours vers le bas
- **android:stackFromBottom**
 - Empile les éléments dans l’ordre inverse
 - Commence avec le dernier item de l’adapter
 - Utile pour les tchats
 - Messaging, Talk, IRC, etc.



- **Socket**
 - Classes Socket, ServerSocket, ...
 - ex: Websocket
- **HTTP**
 - Classes HttpClient, HttpResponse, NameValuePair
 - "HttpClient was deprecated in API Level 22 and removed in API Level 23. You have to use URLConnection."*
 - Les 3 usages CRUD du protocole HTTP
 - Function oriented: GET http://<url>/customerDelete?id=1
 - Service oriented: GET http://<url>/customer?cmd=del&id=1
 - REST: DELETE http://<url>/customer/1/
- **SOAP**
 - Classes SoapObject, SoapSerializationEnvelope, ...



- Les webservices de type REST (Representational State Transfer) permettent d'exposer des fonctionnalités comme un ensemble de ressources (URI) accessibles par la syntaxe et la sémantique du protocole HTTP.
- Format des données renvoyées:
 - XML
 - Parsing de ressources au format XML
 - Voir API javax.xml.parsers
 - Approche hiérarchique (DOM) : org.w3c.dom
 - Approche évènementielle (SAX) : org.xml.sax
 - JSON
 - JSON est un format léger d'échange de données
 - Il est facile à écrire et lire, facilement analysable
 - C'est un format totalement indépendant de tout langage. Ce qui en fait un langage d'échange de données idéal.
 - Il se base sur deux structures :
 - Une collection de couple "Nom / Valeur"
 - Une liste de valeurs ordonnées

- Il existe plusieurs librairies JSON
 - Sojo
 - JSON.org
 - FlexJSON
 - Jackson
 - Gson
 - Et bien d'autres....

Threading

- Gérer les traitements qui ralentissent l'UI et donc qui dégradent l'expérience utilisateur
 - Éviter une « Application Not Responding » (ANR)
 - Fermeture forcée au bout de n secondes
- Deux véritables cas d'école
 - Communication réseau
 - Une NetworkOnMainThreadException est levée depuis Android 3.x
 - Manipulation SQLite
- Pensez à faire patienter l'utilisateur
 - barre de progression, Splashscreen au démarrage...

Thread Principal: UIThread

- Tous les composants d'une application démarrent dans le **thread principal UIThread**
 - Gère l'affichage et les interactions utilisateur
 - Les traitements consommateur et lents bloqueront les autres composants (
- Nécessité d'exécuter ces traitements en tâches de fond
 - À l'aide de tâches asynchrones
 - À l'aide de vos propres Threads enfants
- **Synchronisation avec l'interface graphique**
 - Seul le UIThread est autorisé à modifier les vues

- Les Threads

- Exemple:
- Peuvent être mis en pause ou interrompus

```
Thread background=new Thread(new Runnable() {  
    public void run() {  
        try {  
            //code  
        }  
        catch (Throwable t) {  
            // Termine le thread en arrière-plan  
        }  
    }  
});  
  
background.start();
```

- Pour synchroniser le Thread enfant avec l'UI
 - Directement depuis une activité
 - runOnUiThread(new Runnable() {...});
 - Dans les autres cas
 - Instancier un handler dans le UIThread: h=new handler();
 - Ajouter des runnables à la file d'attente depuis le thread enfant
 - h.post (new Runnable() {...});
 - h.postDelayed(new Runnable() { ...},200); //délai de 200ms

Tâche Asynchrone

- Tâche qui étend la classe `android.os.AsyncTask<?, ?, ?>`
- Fournit des gestionnaires d"événements déjà synchronisés avec le `UIThread` pour mettre à jour les vues
 - **doInBackground**: code à exécuter en background, pas d'interaction avec l'UI
 - **onProgressUpdate**: appelé pendant le traitement pour afficher la progression par exemple.
 - **onPostExecute** : appelé une fois le traitement terminé
 - **onPreExecute** : appelée avant le traitement



```
// private class LoadItemsTask extends AsyncTask<URL, Void, JSONObject> {

    protected JSONObject doInBackground(URL... params) {
        HttpURLConnection conn = (HttpURLConnection) params[0].openConnection();
        InputStream input = conn.getInputStream();
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        copy(input, baos);
        JSONObject jsonRoot = new JSONObject(baos.toString());
        return jsonRoot;
    }
}
```

```
// private class LoadItemsTask extends AsyncTask<URL, Void, JSONObject> {

    protected void onPostExecute(JSONObject jsonRoot) {
        List<Items> items = parseJson(jsonRoot);
        appendItemsToList(item);
        notifyDataSetChanged();
    }
}
```

Java

```
// private class LoadImageTask extends AsyncTask<URL, Void, Bitmap> {  
  
    public LoadImageTask(ImageView imageView) {  
        mImageView = imageView;  
    }  
  
    protected Bitmap doInBackground(URL... params) {  
        HttpURLConnection conn = (HttpURLConnection) params[0].openConnection();  
        InputStream input = conn.getInputStream();  
        return BitmapFactory.decodeStream(input);  
    }  
}
```

Java

```
// private class LoadImageTask extends AsyncTask<URL, Void, Bitmap> {  
  
    protected void onPostExecute(Bitmap result) {  
        mImageView.setImageBitmap(result);  
    }  
}
```

```
new LoadImageTask(holder.imageView).execute(  
    new URL(BASE_URL + item.imageUrl));
```

- Plusieurs problèmes
 - AsyncTask et les rotations: lors d'une rotation, l'Activité est détruite, puis recréée => du coup les données sont rechargées.
 - AsyncTask et cycle de vie: AsyncTask continue de s'exécuter même si l'Activité ou l'application ont été détruites -> il faut tuer explicitement la tâche: `AsyncTask.cancel()`.
 - Stopper une tâche: La tâche n'est pas tuée immédiatement, elle passe dans un état « `cancelled` », mais du coup il faut gérer cet état pendant le traitement de longues opérations.
 - Limitations sur le nombre de tâches lancées, un trop grand nombre de tâches lancées, fait « `crasher` » une application
 - Les AsyncTasks s'exécutent en série et non en parallèle

- Librairie Volley
 - Bibliothèque permettant de construire facilement des applications réseaux très performantes sur Android.
 - Volley fonctionne avec un mécanisme de thread pool configurable
 - Il supporte la priorisation des requêtes
 - Les résultats des requêtes sont automatiquement mis en cache disque et mémoire
 - On peut très facilement annuler des requêtes en cours, afin d'éviter qu'elles ne soient exécutées pour rien
 - Supporte différents formats: images, json, texte brut,...

```
git clone https://android.googlesource.com/platform/frameworks/volley
```

- Implémenter Volley
 - Intégrer les sources directement dans le projet ou en liant la librairie.
 - Dans la main Activity, il faut déclarer un objet RequestQueue.

```
1 | private RequestQueue mRequestQueue;  
2 | :  
3 | :  
4 | :  
5 | mRequestQueue = Volley.newRequestQueue(this);
```

- Création d'un l'objet JsonObjectRequest en lui passant divers paramètres comme : méthode http, listeners en cas de succès, d'erreur

```
01 JsonObjectRequest jr = new
02     JsonObjectRequest(Request.Method.GET,url,null,new
03         Response.Listener<JSONObject>() {
04             @Override
05             public void onResponse(JSONObject response) {
06                 Log.i(TAG,response.toString());
07                 parseJSON(response);
08                 va.notifyDataSetChanged();
09                 pd.dismiss();
10             }
11         },new Response.ErrorListener() {
12             @Override
13             public void onErrorResponse(VolleyError error) {
14                 Log.i(TAG,error.getMessage());
15             }
16     });
17 }
```

- Ajout de l'objet jsonObjectRequest à la RequestQueue:
mRequestQueue.add(jr);



- Nouveauté depuis Android 3.0
 - Prévus pour les écrans plus larges, comme les tablettes
- Principe de base
 - Fragmentation de l'espace d'affichage en différentes zones, chargeables indépendamment, et réutilisables
 - Même idée que les "Frames" en HTML
- Un fragment est une classe qui étend
 - android.app.Fragment
- Les fragments sont ensuite attachés/détachés à une activité hôte

ListFragment



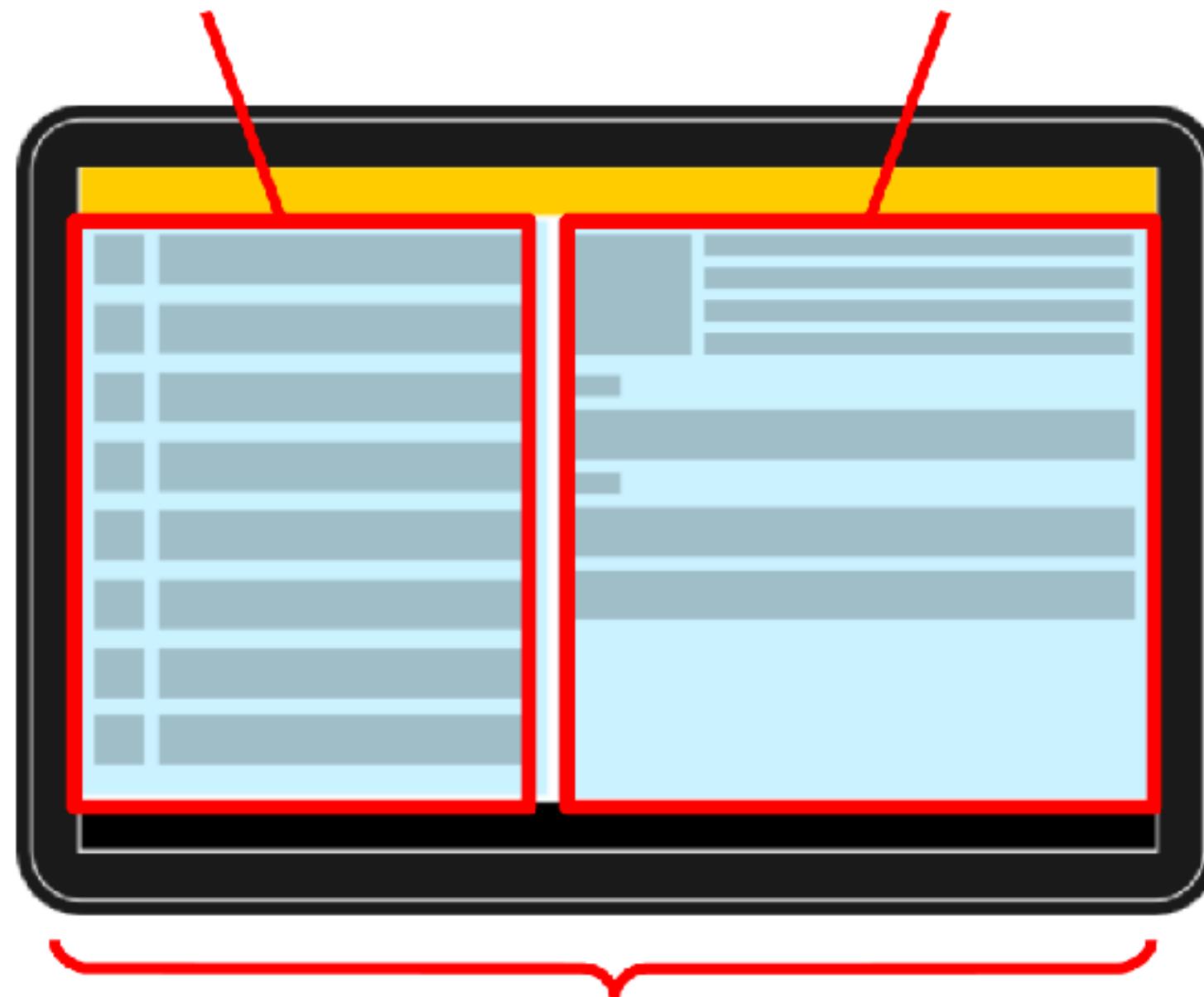
DetailsFragment



MainActivity*

ListFragment

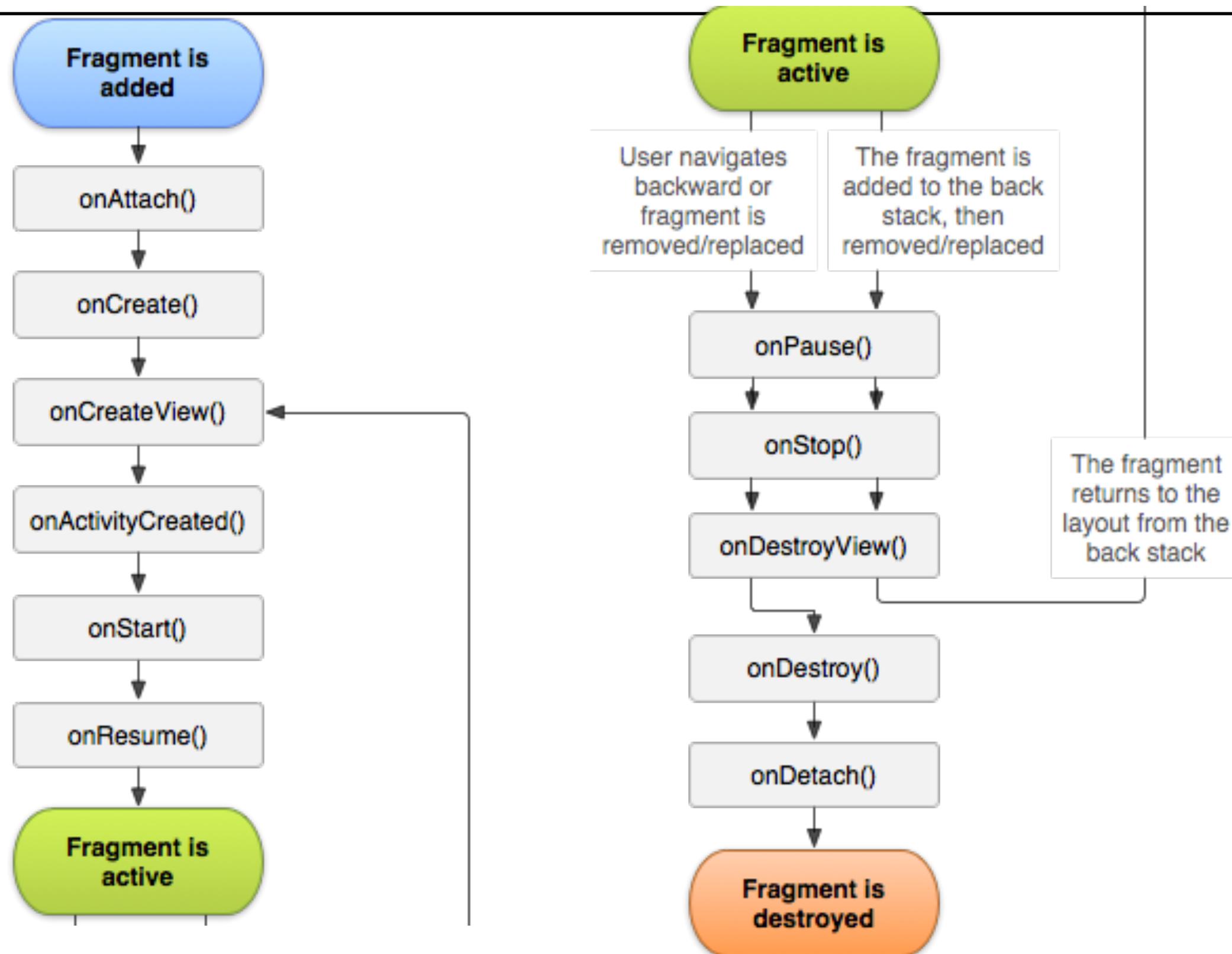
DetailsFragment

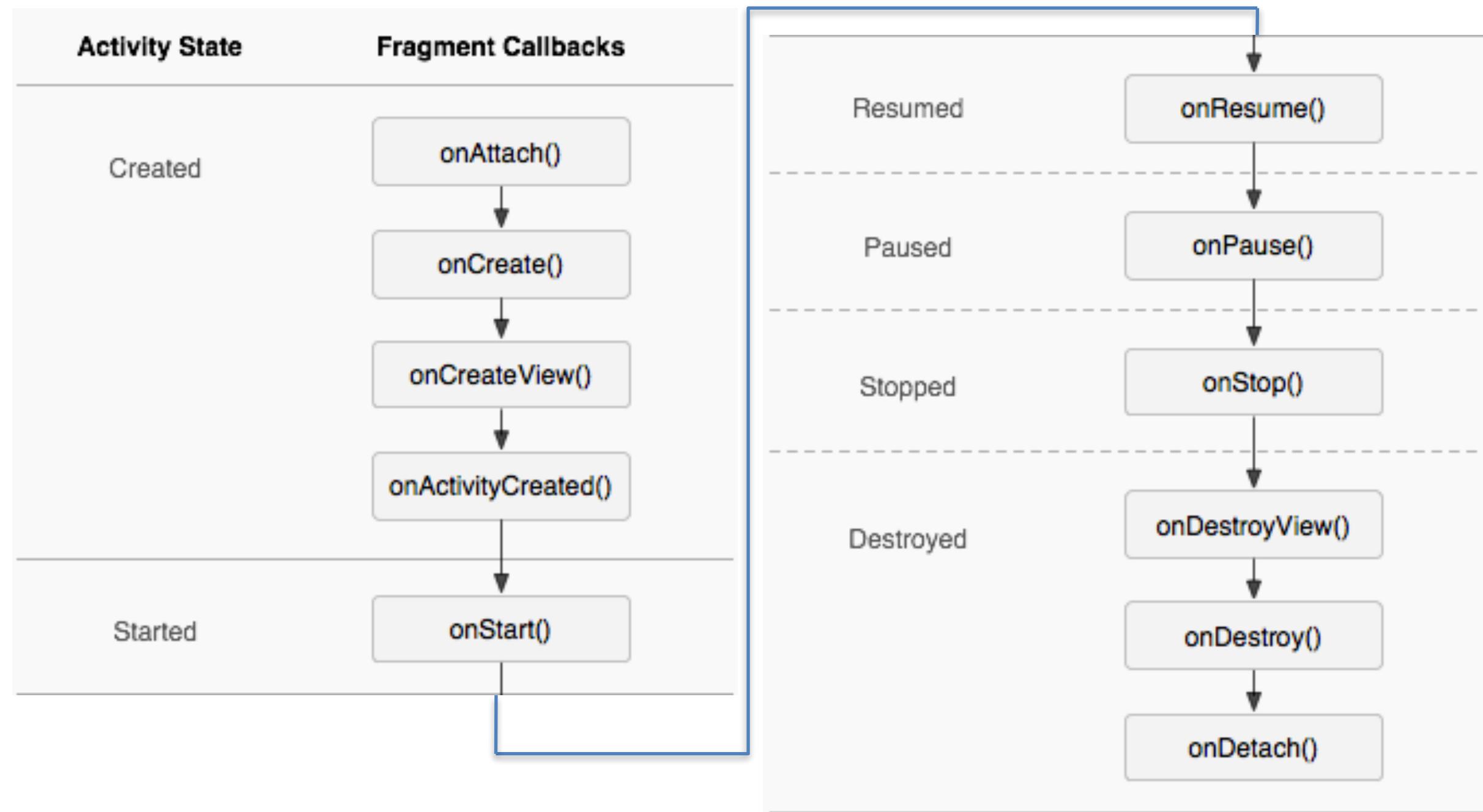


MainActivity



- Avantages:
 - Composants faciles à réutiliser
 - Possible de concevoir des applications « single panel » pour les téléphones et « multi-panel » pour les tablettes.
- Cycle de vie
 - Un fragment a son propre cycle de vie mais il est toujours connecté au cycle de vie de l'activité.





- Généralement un fragment implémentera les méthodes suivantes:
 - `onCreate(Bundle)` pour initialiser les différents composants
 - `onCreateView(LayoutInflater, ViewGroup, Bundle)` appelé lorsque l'interface est créée la première fois. Cette méthode permet de renvoyer la vue associée au fragment à l'activité
 - `onPause()` pour conserver les modifications utilisateurs pour qu'il puisse les retrouver lorsque le fragment sera à nouveau actif.
- Hérite de la classe `Fragment`, `ListFragment`,

GUI :: Fragments :: Ajouter un fragment

- Créer le fragment

MainFragment.java

```
01 public class MainFragment extends Fragment {  
02  
03     TextView title;  
04  
05     @Nullable  
06     @Override  
07     public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
08         return inflater.inflate(R.layout.fragment_main, container, false);  
09     }  
10  
11     @Override  
12     public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {  
13         super.onViewCreated(view, savedInstanceState);  
14         title = (TextView) view.findViewById(R.id.title);  
15  
16         title.setText("Je suis un fragment");  
17     }  
18 }  
19 }
```

layout/fragment_main.xml

```
01 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
02     android:layout_width="match_parent"  
03     android:layout_height="match_parent"  
04     android:orientation="vertical">  
05  
06     <TextView  
07         android:id="@+id/title"  
08         android:layout_width="wrap_content"  
09         android:layout_height="wrap_content"  
10         android:layout_gravity="center"/>  
11  
12 </LinearLayout>
```

GUI :: Fragments :: Ajouter un fragment

- Déclarer le fragment dans le layout

layout/activity_main.xml

```
01 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
02     android:layout_width="match_parent"
03     android:layout_height="match_parent">
04
05     <fragment
06         android:name="com.tutosandroidfrance.fragments.MainFragment"
07         android:layout_width="match_parent"
08         android:layout_height="match_parent">
09
10    </FrameLayout>
```

Il n'y a aucun code à ajouter dans votre activité

MainActivity.java

```
1 public class MainActivity extends AppCompatActivity {
2
3     @Override
4     protected void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
6         setContentView(R.layout.activity_main);
7     }
8
9 }
```

- Ajouter dynamiquement des fragments

layout/activity_main.xml

```
01 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
02     android:layout_width="match_parent"
03     android:layout_height="match_parent">
04
05     <FrameLayout
06         android:id="@+id/fragmentLayout"
07         android:layout_width="match_parent"
08         android:layout_height="match_parent"/>
09
10 </FrameLayout>
```

On utilise maintenant le FragmentManager dans notre Activity

MainActivity.java

```
01 public class MainActivity extends AppCompatActivity {
02
03     @Override
04     protected void onCreate(Bundle savedInstanceState) {
05         super.onCreate(savedInstanceState);
06         setContentView(R.layout.activity_main);
07
08         getSupportFragmentManager().beginTransaction()
09             .replace(R.id.fragmentLayout, new MainFragment())
10             .commit();
11     }
12
13 }
```



- Envoyer des données à notre fragment

```
01 public class MainFragment extends Fragment {  
02  
03     private static final String TITLE = "TITLE";  
04  
05     public static MainFragment newInstance(String title) {  
06         MainFragment fragment = new MainFragment();  
07         Bundle args = new Bundle();  
08         args.putString(TITLE, title);  
09         fragment.setArguments(args);  
10         return fragment;  
11     }  
12  
13     TextView title;  
14  
15     @Nullable  
16     @Override  
17     public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
18         return inflater.inflate(R.layout.fragment_main, container, false);  
19     }  
20  
21     @Override  
22     public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {  
23         super.onViewCreated(view, savedInstanceState);  
24         title = (TextView) view.findViewById(R.id.title);  
25  
26         if (getArguments() != null) {  
27             Bundle args = getArguments();  
28             if (args.containsKey(TITLE))  
29                 title.setText(args.getString(TITLE));  
30         }  
31     }  
32 }
```



- Envoyer des données à notre activité

Créer un interface MainFragmentCallBack

```
1 public class MainFragment extends Fragment {  
2  
3     public interface MainFragmentCallback{  
4         void onTitleClicked();  
5     }  
6  
7     ...  
8 }
```

Implémenter cette interface dans notre activité

```
01 public class MainActivity extends AppCompatActivity implements MainFragment.MainFragmentCallback{  
02  
03     @Override  
04     protected void onCreate(Bundle savedInstanceState) {  
05         super.onCreate(savedInstanceState);  
06         setContentView(R.layout.activity_main);  
07  
08         getSupportFragmentManager().beginTransaction()  
09             .replace(R.id.fragmentLayout, MainFragment.newInstance("Mon nouveau titre"))  
10             .commit();  
11     }  
12  
13     @Override  
14     public void onTitleClicked() {  
15         //votre action, ici ce sera un simple Toast  
16         Toast.makeText(this,"Tu as clické !",Toast.LENGTH_SHORT).show();  
17     }  
18 }
```

GUI :: Fragments :: Ajouter un fragment

- Envoyer des données à notre activité

Récupérer le callback

```
01 public class MainFragment extends Fragment {  
02  
03     public interface MainFragmentCallback {  
04         void onTitleClicked();  
05     }  
06  
07     private static final String TITLE = "TITLE";  
08  
09     public static MainFragment newInstance(String title) {  
10         MainFragment fragment = new MainFragment();  
11         Bundle args = new Bundle();  
12         args.putString(TITLE, title);  
13         fragment.setArguments(args);  
14         return fragment;  
15     }  
16  
17     TextView title;  
18     MainFragmentCallback mainFragmentCallback;  
19  
20     @Nullable  
21     @Override  
22     public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle  
23         return inflater.inflate(R.layout.fragment_main, container, false);  
24     }  
25  
26     @Override  
27     public void onAttach(Activity activity) {  
28         super.onAttach(activity);  
29         if (activity instanceof MainFragmentCallback)  
30             mainFragmentCallback = (MainFragmentCallback) activity;  
31     }  
32 }
```



- Envoyer des données à notre activité

Récupérer le callback

```
32  
33     @Override  
34     public void onDetach() {  
35         super.onDetach();  
36         mainFragmentCallback = null;  
37     }  
38  
39     @Override  
40     public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {  
41         super.onViewCreated(view, savedInstanceState);  
42         title = (TextView) view.findViewById(R.id.title);  
43  
44         Bundle args = getArguments();  
45         if (args != null) {  
46             if (args.containsKey(TITLE))  
47                 title.setText(args.getString(TITLE));  
48         }  
49  
50         title.setOnClickListener(new View.OnClickListener() {  
51             @Override  
52             public void onClick(View v) {  
53                 if(mainFragmentCallback != null)  
54                     mainFragmentCallback.onTitleClicked();  
55             }  
56         });  
57     }  
58  
59 }
```



- Android permet de définir l'apparence des composants Android dans les fichiers de ressources XML.
- Ex: styles.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="text">
        <item name="android:padding">4dip</item>
        <item name="android:textAppearance">?android:attr/textAppearanceLarge</item>
        <item name="android:textColor">#000000</item>
    </style>

    <style name="layout">
        <item name="android:background">#C0C0C0</item>
    </style>
</resources>
```

- Il sera possible d'attribuer le style à des éléments de type texte par `style="@style/text"`



- Thème: un thème est un style appliqué à toute une activité ou application.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <style name="MyTheme" parent="android:Theme.Light">
        <item name="android:windowNoTitle">true</item>
        <item name="android:windowBackground">@color/translucent_red</item>
        <item name="android:listViewStyle">@style/MyListView</item>
    </style>

    <style name="MyListView" parent="@android:style/Widget.ListView">
        <item name="android:listSelector">@drawable/ic_menu_home</item>
    </style>

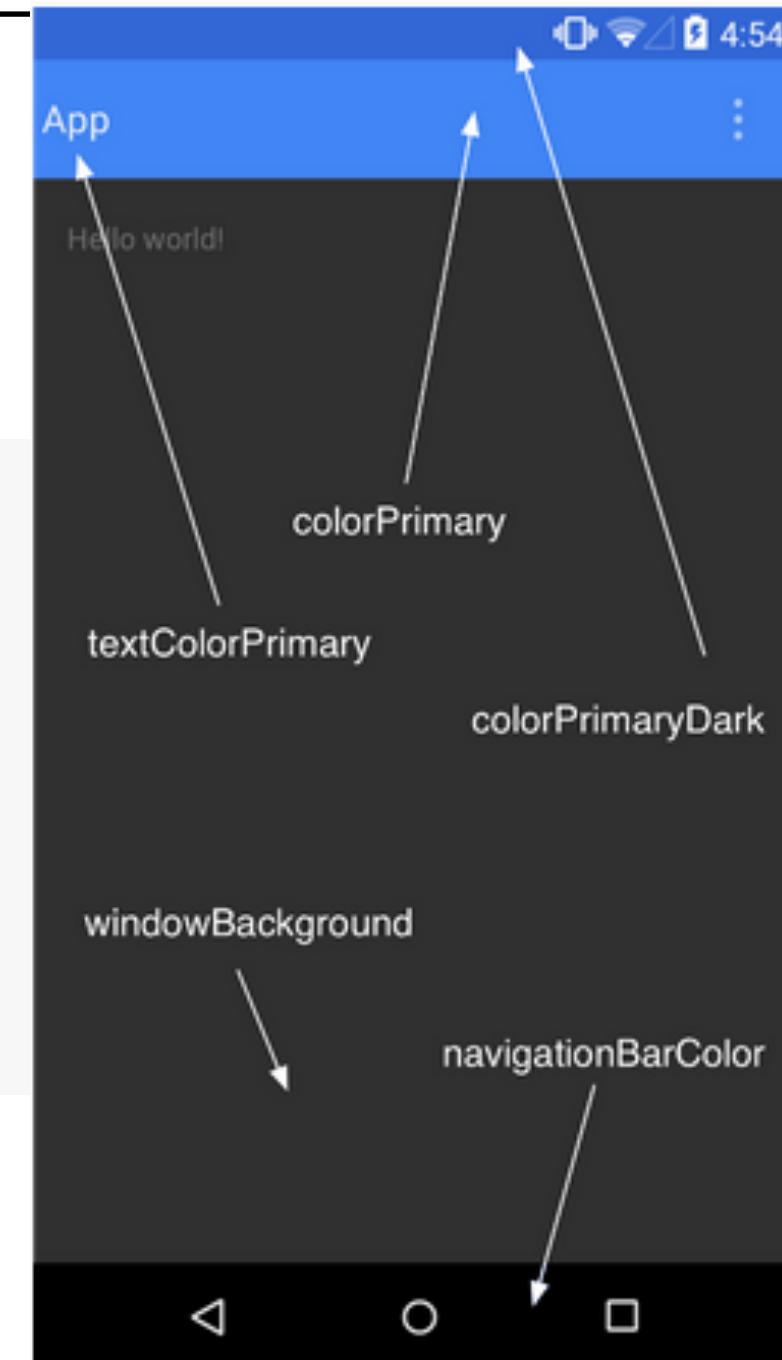
</resources>
```

Il est considéré comme **bonne pratique** d'hériter d'un style de plate-forme et le personnaliser

GUI :: Thèmes et styles

- Ex: Customiser son application

```
<resources>
    <style name="AppTheme" parent="android:Theme.Material">
        <!-- Main theme colors -->
        <!-- your app branding color for the app bar -->
        <item name="android:colorPrimary">@color/primary</item>
        <!-- darker variant for the status bar and contextual app bars -->
        <item name="android:colorPrimaryDark">@color/primary_dark</item>
        <!-- theme UI controls like checkboxes and text fields -->
        <item name="android:colorAccent">@color/accent</item>
    </style>
</resources>
```





- Une notification fournit une indication permettant de prévenir l'utilisateur de certains événements (arrivée d'un SMS, mail , appel en absence, ...)
- Différentes formes
 - LED
 - Son
 - Vibreur
 - Barre de notification (icône)
- Utilisation facilitée par le notification manager
- Exemple

GUI :: Notifications



```
// prepare intent which is triggered if the
// notification is selected

Intent intent = new Intent(this, NotificationReceiver.class);
// use System.currentTimeMillis() to have a unique ID for the pending intent
PendingIntent pIntent = PendingIntent.getActivity(this, (int)
System.currentTimeMillis(), intent, 0);

// build notification
// the addAction re-use the same intent to keep the example short
Notification n = new Notification.Builder(this)
    .setContentTitle("New mail from " + "test@gmail.com")
    .setContentText("Subject")
    .setSmallIcon(R.drawable.icon)
    .setContentIntent(pIntent)
    .setAutoCancel(true)
    .addAction(R.drawable.icon, "Call", pIntent)
    .addAction(R.drawable.icon, "More", pIntent)
    .addAction(R.drawable.icon, "And more", pIntent).build();

NotificationManager notificationManager =
(NotificationManager) getSystemService(NOTIFICATION_SERVICE);

notificationManager.notify(0, n);
```

- Utiliser les images 9 patches
 - Images divisées en neuf zones (dont certaines étirables)
 - Outil Draw 9-patch du répertoire /tool du SDK Android
- Draw9patch.exe produit des fichiers *.9.png





- Prévoir différentes variantes d'une même chaîne
 - /res/values-fr/strings.xml On manipule une seule ressource,
 - /res/values-en/strings.xml
 - /res/values-it/strings.xml
- sans se soucier de la langue (R.strings.hello)
- Le choix sera fait automatiquement en fonction de la configuration du terminal (ex: LOCALE=FR_fr)
- S'applique également aux images car elles peuvent afficher du texte !
 - /res/drawable-fr/splashscreen.png
 - /res/drawable-en/splashscreen.png



- Mécanisme simple et léger
 - Sauvegarde de paires clé/valeur simple
 - SharedPreferences pref = getPreferences(Activity.MODE_PRIVATE)
- Sauvegarder des préférences
 - Récupère un éditeur de préférences :
Editor ed = pref.edit()
 - Stocke les paires :
`ed.putString("teacher", "Olivier Le Goaer");`
`ed.putBoolean("isBrilliant", true);`
 - Valide les modifications :
`ed.commit();`
- Retrouvez des préférences
 - String t = pref.getString("teacher","unknown");

Publier une application



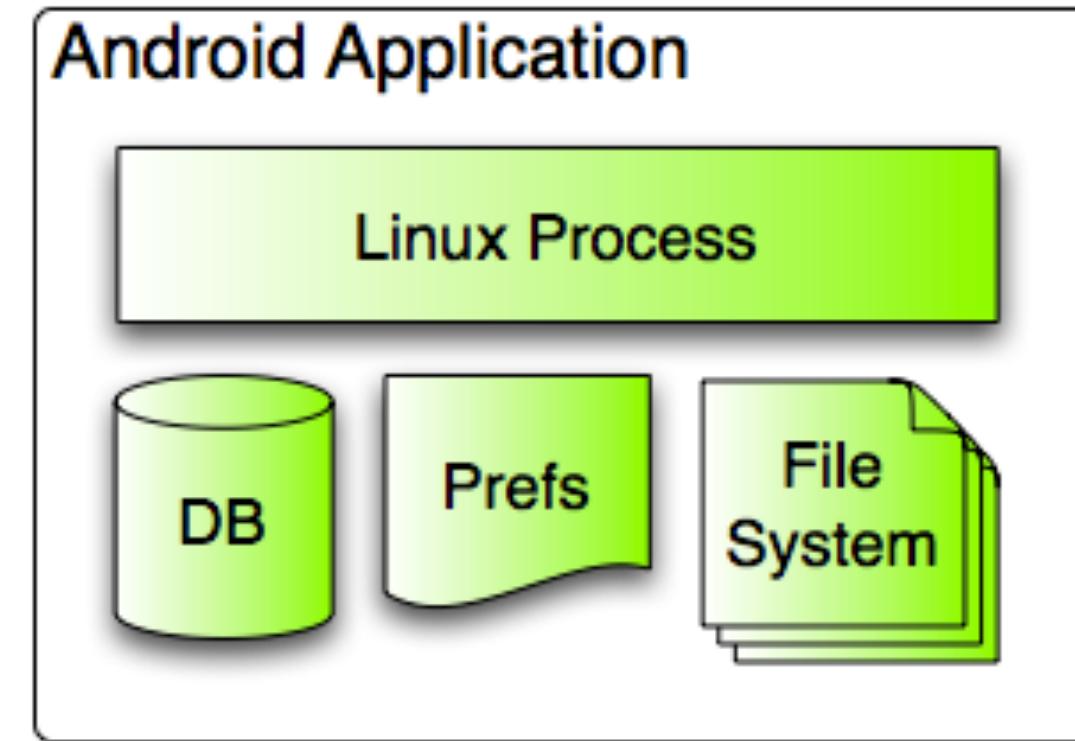
- Toute application doit être signée
 - Signature digitale avec un certificat dont la clé privée est conservée par le(s) développeur(s)
 - Signature avec la clé de débugage par défaut
- Trois étapes :
 1. Obtenir une clé privée (stockée dans un "keystore")
Utilitaire keytool.exe du JDK
 2. Signer l'APK avec la clé privée
 - Procédure : Utilitaire jarsigner.exe du JDK entièrement automatisée
 3. Optimiser l'APK qui vient d'être signé



- 3 solutions s'offrent à vous
 - Choisir Google Play Store (Android Market)
 - Site officiel : <https://play.google.com>
 - 25\$ de frais de dossier pour l'accès au store
 - 70% du prix de vente va aux développeurs (30% à Google)
 - Les autres revenus générés sont reversés via Google Checkout
 - Autopublier sur votre propre site Web
 - Exemple : <http://www.univ-pau.fr/~olegoaer/bankster.apk>
 - Type MIME pour le téléchargement : application/vnd.android.package-archive
 - Choisir un magasin alternatif
 - YAAM (<http://yaam.mobi/>)
 - Bazaar (<http://www.bazaarandroid.com/>)
 - AndroLib (<http://fr.androlib.com/>)

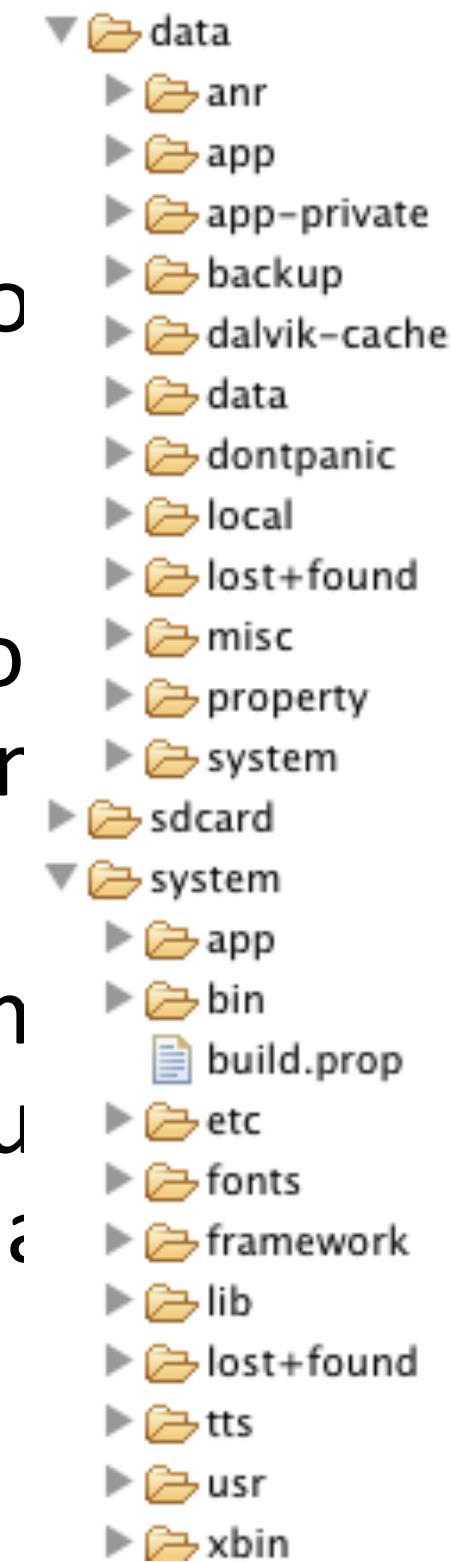
Android et la sécurité

- Chaque application Android s'exécute dans son propre processus Linux
- Chaque application dispose de sa “sandbox”, avec ses propres préférences, sa base de données
- Les autres applications ne peuvent accéder à aucune de ces données à moins qu'elles n'aient été partagées explicitement.





- Le “file system” dispose de 3 points de montée pour le système, un pour les applications, et le reste.
- Chaque application dispose de sa “sandbox”. Personne d’autre ne peut y accéder. La structure trouve dans /data/data/<package name>
- La sdcard est toujours accessible. Tout le monde peut y accéder et c’est un bon endroit pour stocker de gros fichiers comme des vidéos ou de la musique.



- La “Sandbox”
 - Une application est une « île » à elle seule
 - Elle contient un certain nombre d’activités, services, receivers et providers.
 - Elle dispose de son propre système de fichiers, base de données, librairies natives.
 - Aucune application ne peut accéder à n’importe laquelle de ses données, sans demander au préalable la permission.