Présentation d'Android

le cnam

Android

- Système d'exploitation Open Source (licence Apache) ciblé **principalement** sur la téléphonie mobile et les tablettes tactiles
- Initialement développé par la société éponyme rachetée par Google en 2005
- premier SDK publié en nov. 2007 et création de l'OHA (Open Handset Alliance)

  http://developer.android.com/index.html

Les versions

1.0		1 nov. 2007	version développeurs, distribuée avec le SDK avant la sortie du premier téléphone Android
1.1	Petit Four	1 oct. 2008	version incluse dans le premier téléphone, le HTC G1/Dream
1.5	Cupcake	1 avr. 2009	nouvelles fonctionnalités et mises à jour de l'interface graphique
1.6	Donut	1 sept. 2009	nouvelles fonctionnalités et mises à jour de l'interface graphique
2.0	Eclair	1 oct. 2009	nouvelles fonctionnalités et mises à jour de l'interface graphique
2.1	Eclair	1 janv. 2010	correction du trop grand nombre de bugs de la 2.0
2.2	Froyo	1 mai 2010	vitesse améliorée (JIT), nouvelles fonctionnalités et mises à jour de l'interface graphique
2.3	Gingerbread	1 déc. 2010	dernière version dédiée uniquement aux smartphones. Cette version est parfois utilisée sur de petites tablettes.
3.0	Honeycomb	1 févr. 2011	réservé aux tablettes tactiles et aux téléviseurs connectés12, cette mise à jour comprend de nombreux changements dans l'interface
4.0	Ice Cream Sandwich	1 oct. 2011	cette nouvelle version, fortement inspirée d'Honeycomb, unifiée pour smartphones, tablettes et Google TV apporte de nombreux changements
4.1	Jelly Bean	1 juil. 2012	ajoute un système de notification améliorée, Google Now un système concurrent de Siri et le Project Butter qui augmente la fluidité d'Android;
4.2	Jelly Bean	1 nov. 2012	nouvelle interface de l'appareil photo et l'introduction de Photosphère permettant une prise des photos à 360° type Street View, d'un système multi-compte uniquement sur tablette, de Type Gesture permettant d'écrire avec le clavier rien qu'en glissant le doigt et d'améliorations de Google Now…

Structure du système Android

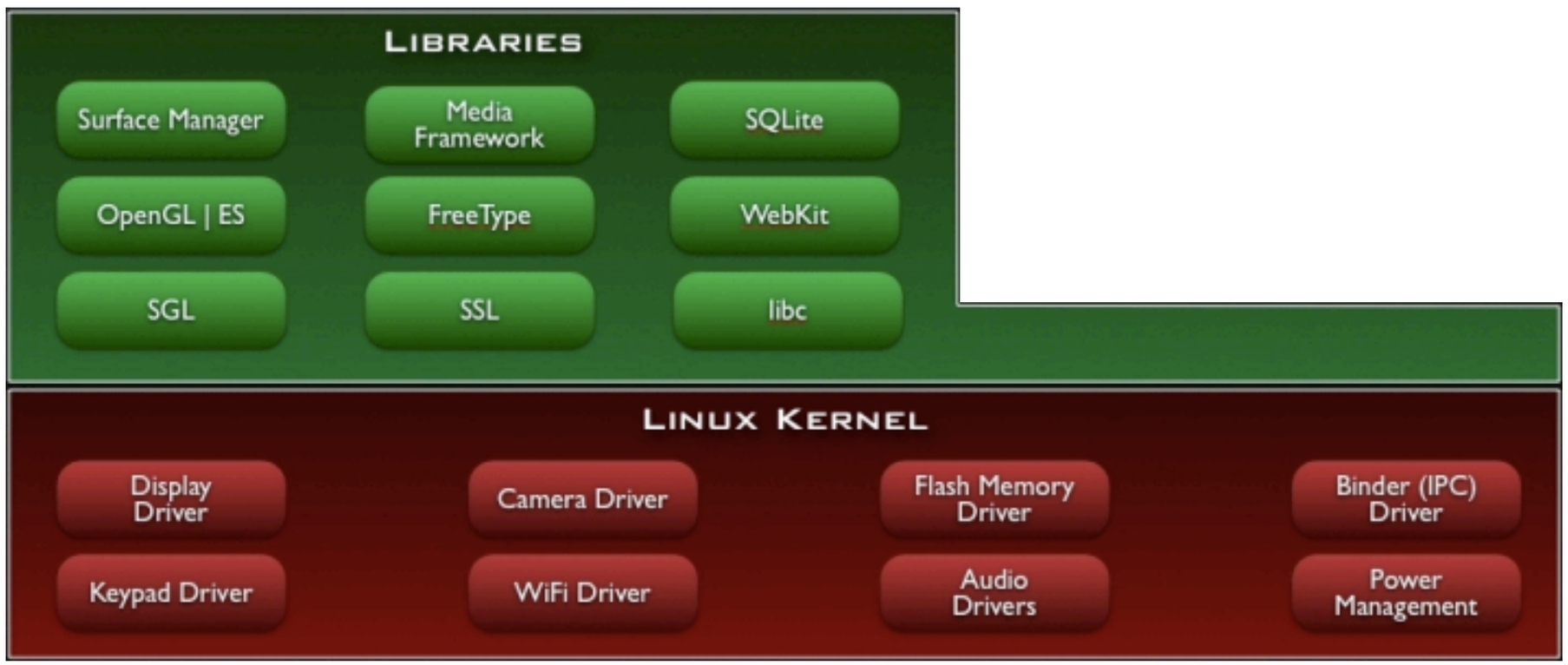
Structure du système Android

Noyau modifié écrit essentiellement en C et en assembleur :

- ajout d'une couche d'abstraction des périphériques
- ajout d'un mécanisme d'IPC spécifique : les Binders
- la mémoire partagée (ashmem), le logcat, l'oom accounting…

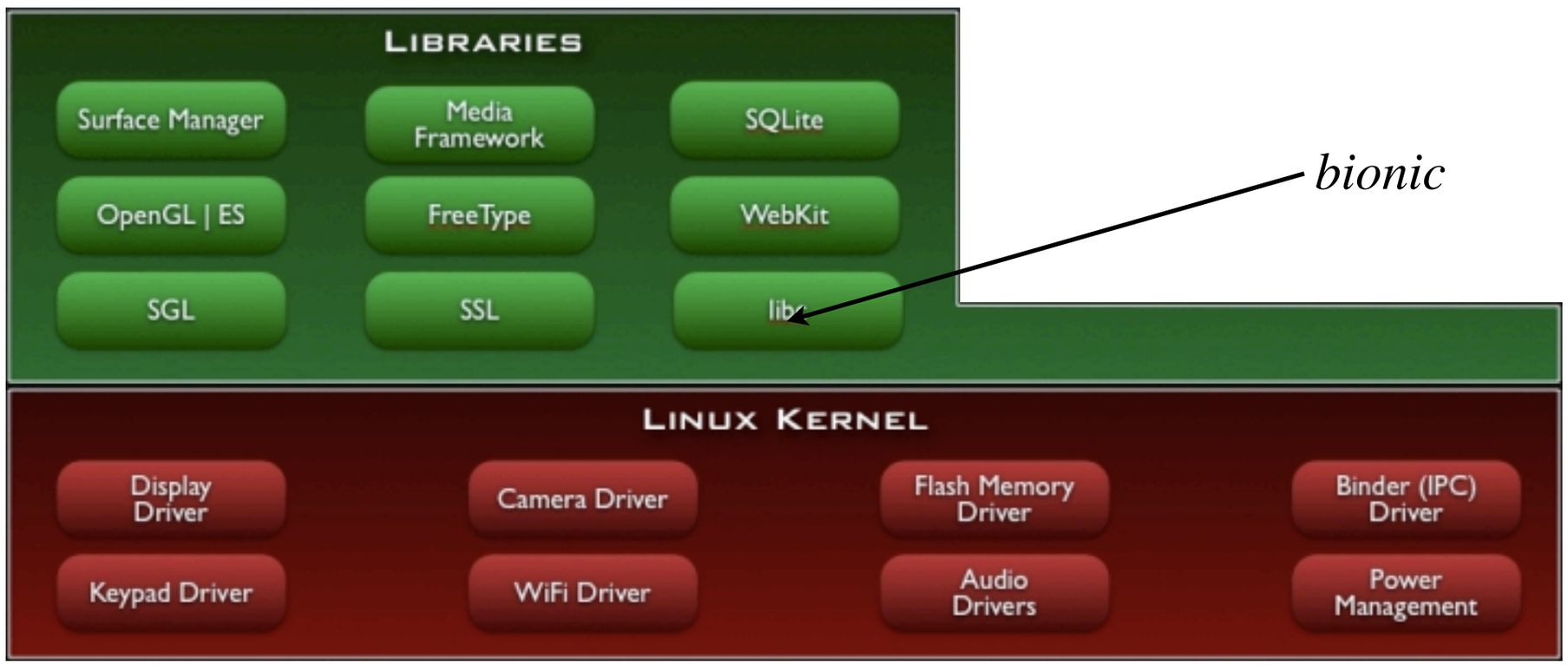
Structure du système Android

Ensemble de librairies écrites en C/C++, offrant les principaux services du système d'exploitation, ces librairies sont : soit des adaptations (parfois conséquentes) issus du système GNU/Linux soit des créations spécifiques



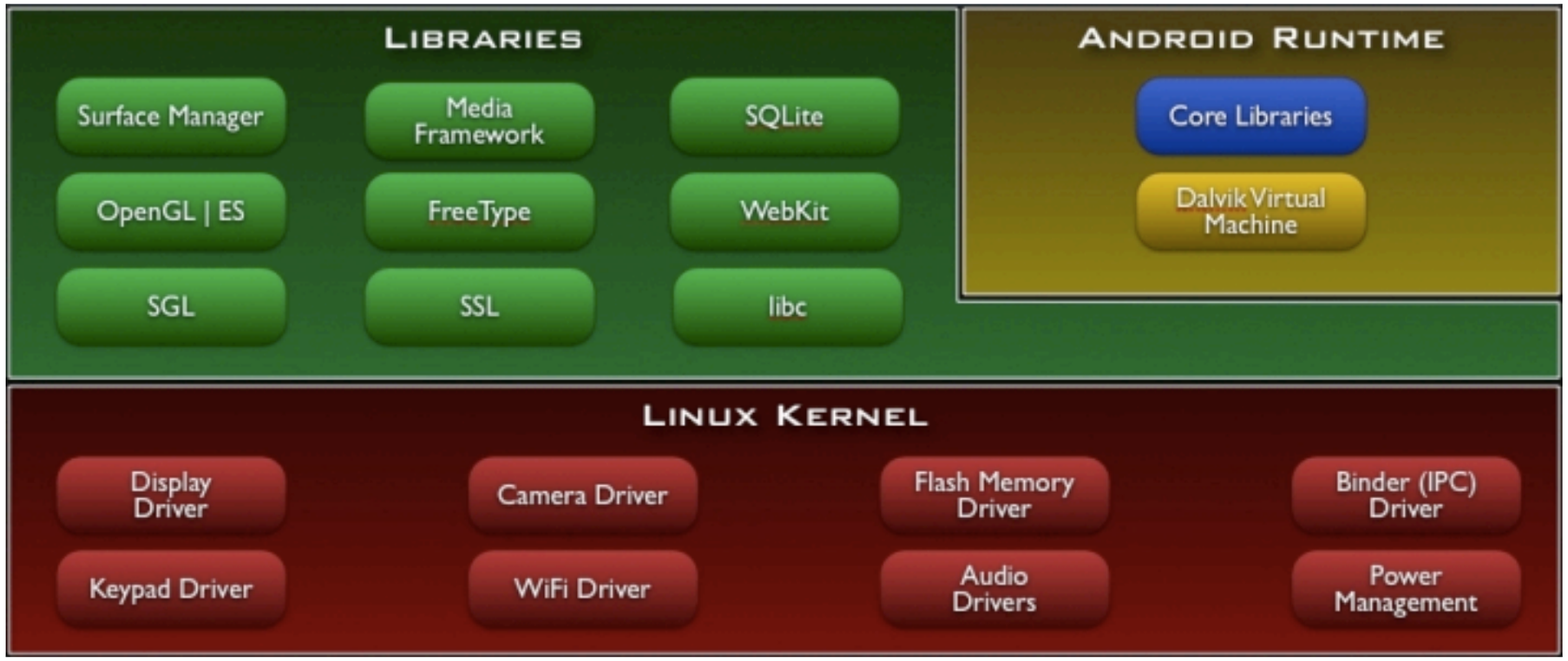
Structure du système Android

Ensemble de librairies écrites en C/C++, offrant les principaux services du système d'exploitation, ces librairies sont : soit des adaptations (parfois conséquentes) issus du système GNU/Linux soit des créations spécifiques



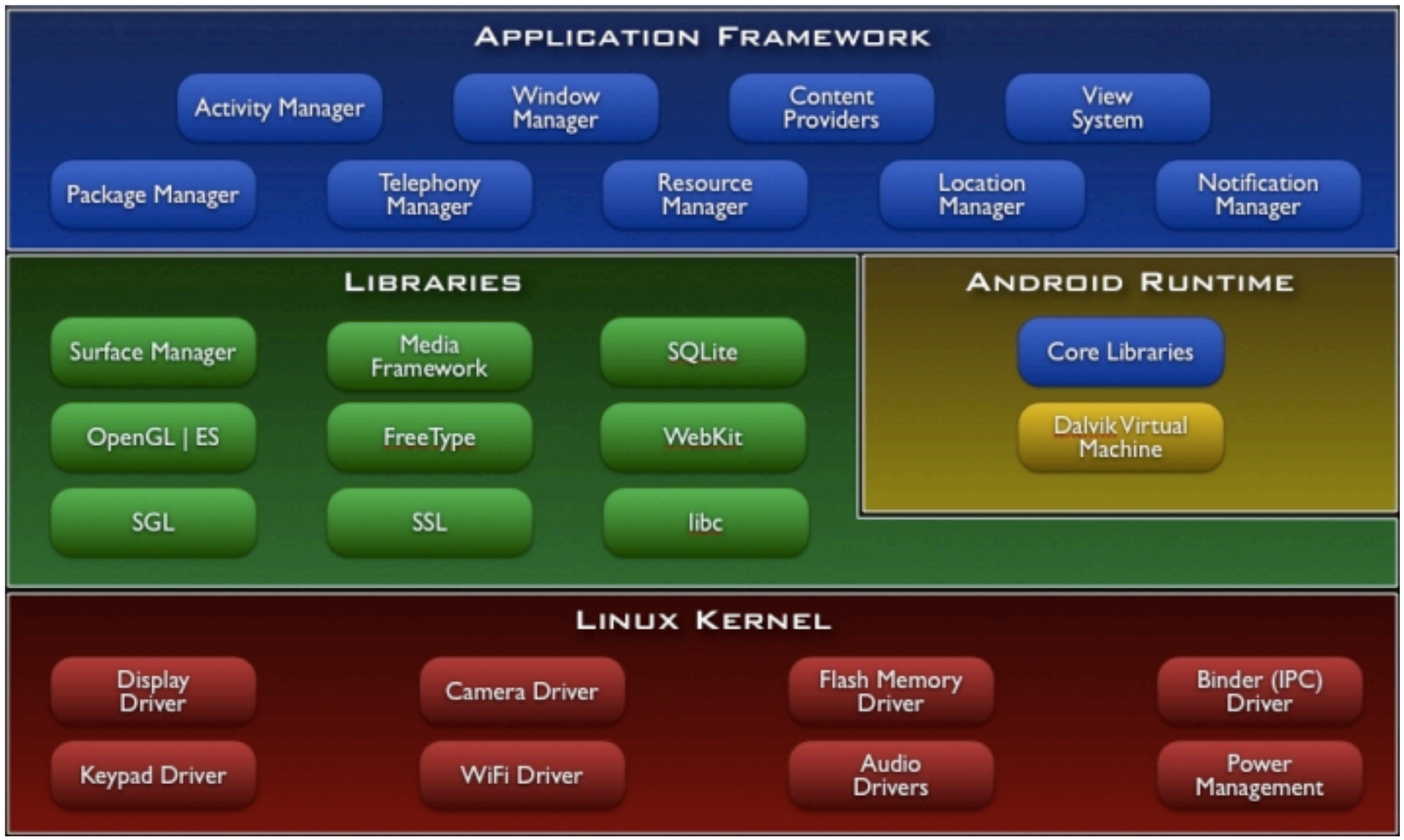
Structure du système Android

Le Runtime Android écrit en C/C++ et Java, apporte le support Java (non officiel) à travers la Dalvik VM et les librairies implantant les APIs Android (Core Libraries)

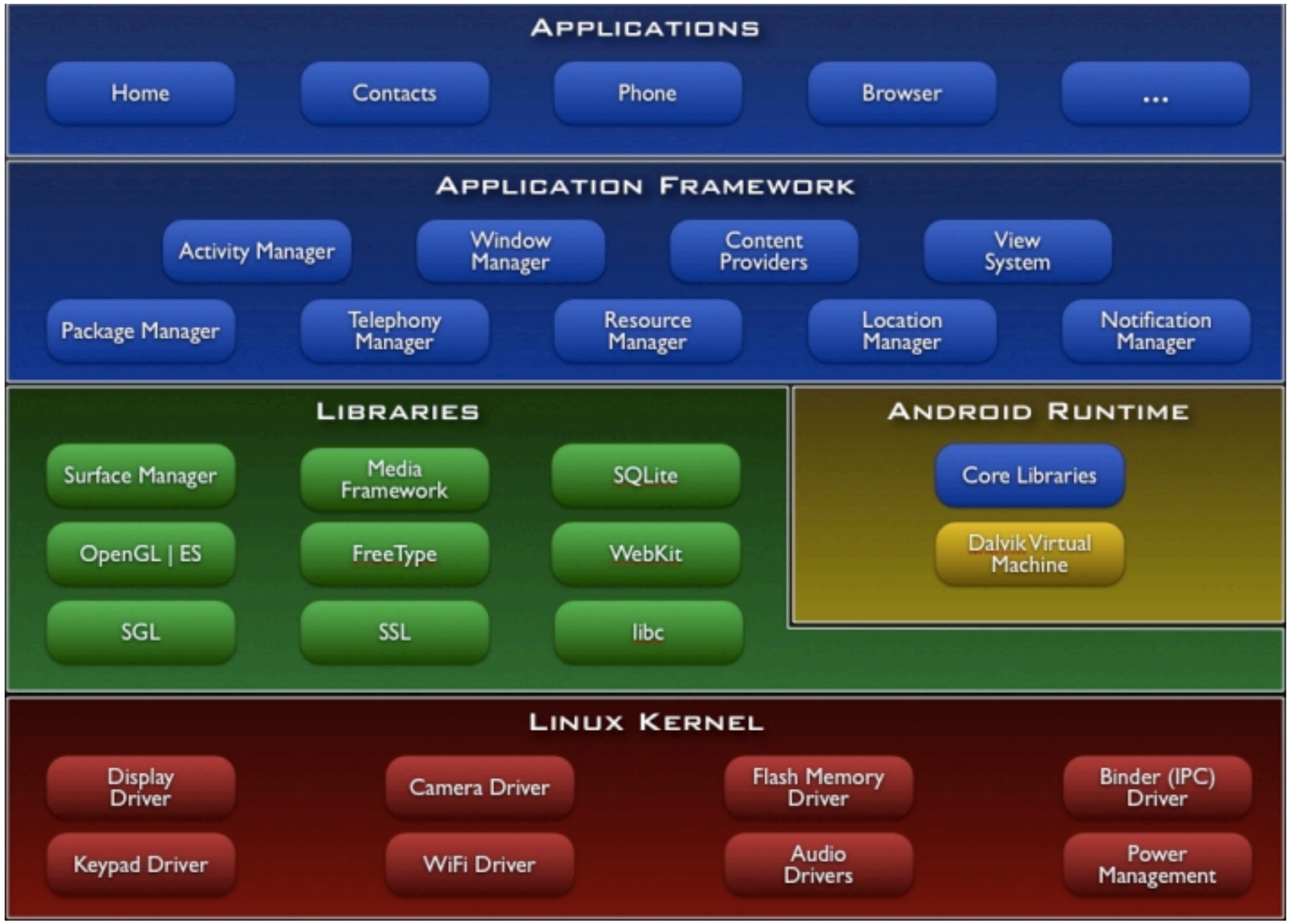


Structure du système Android

Les composants applicatifs implantant les services de haut niveau, écrit en Java (parfois aussi en utilisant le NDK)



Structure du système Android



Structure du système Android

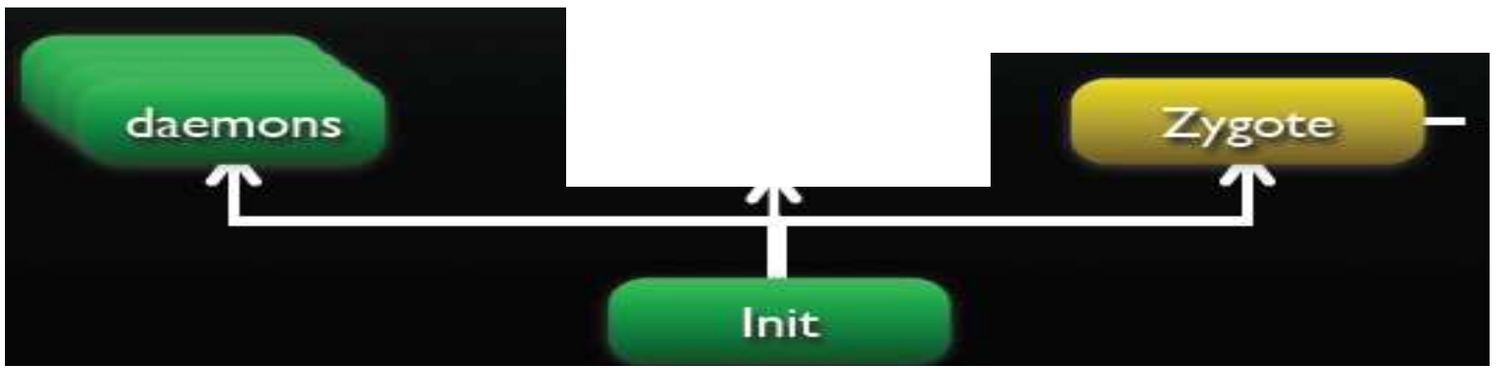
- Système de fichiers constitués de 3 points de montages :
  - le `/` monté (rootfs) ramdisk.img
  - le `/system` (ext2, ext3, ext4, yaffs2, rfs…) system.img
  - le `/data` (ext2, ext3, ext4, yaffs2, rfs…) userdata-qemu.img
- D'autres sont optionnels `/cache`, `/mnt/sdcard`, `/system/mt`…
- Le `/etc` est un lien symbolique vers `/system/etc`

Mise en place du système

Mise en place du système

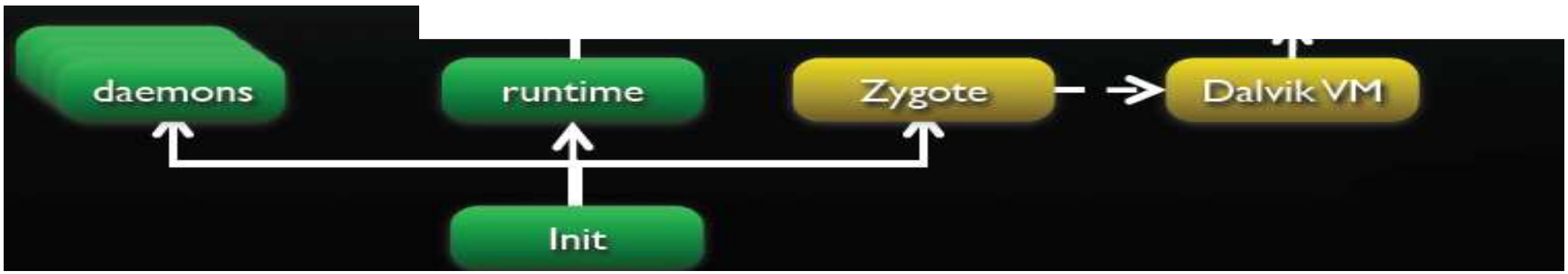
Mise en place du système

Mise en place du système

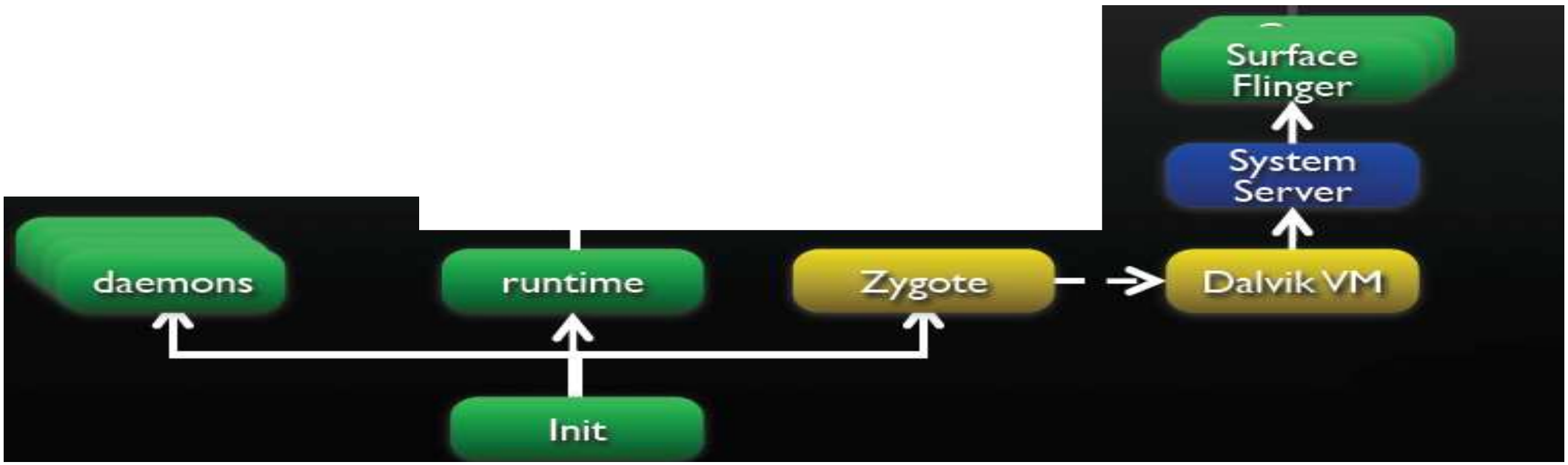


Mise en place du système

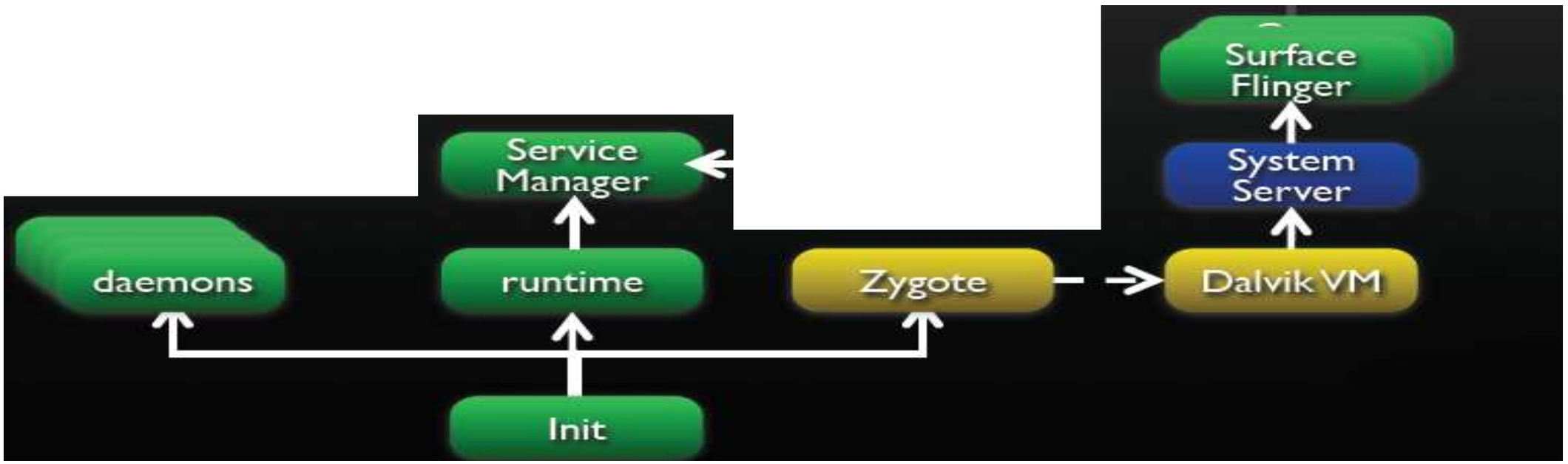
Mise en place du système



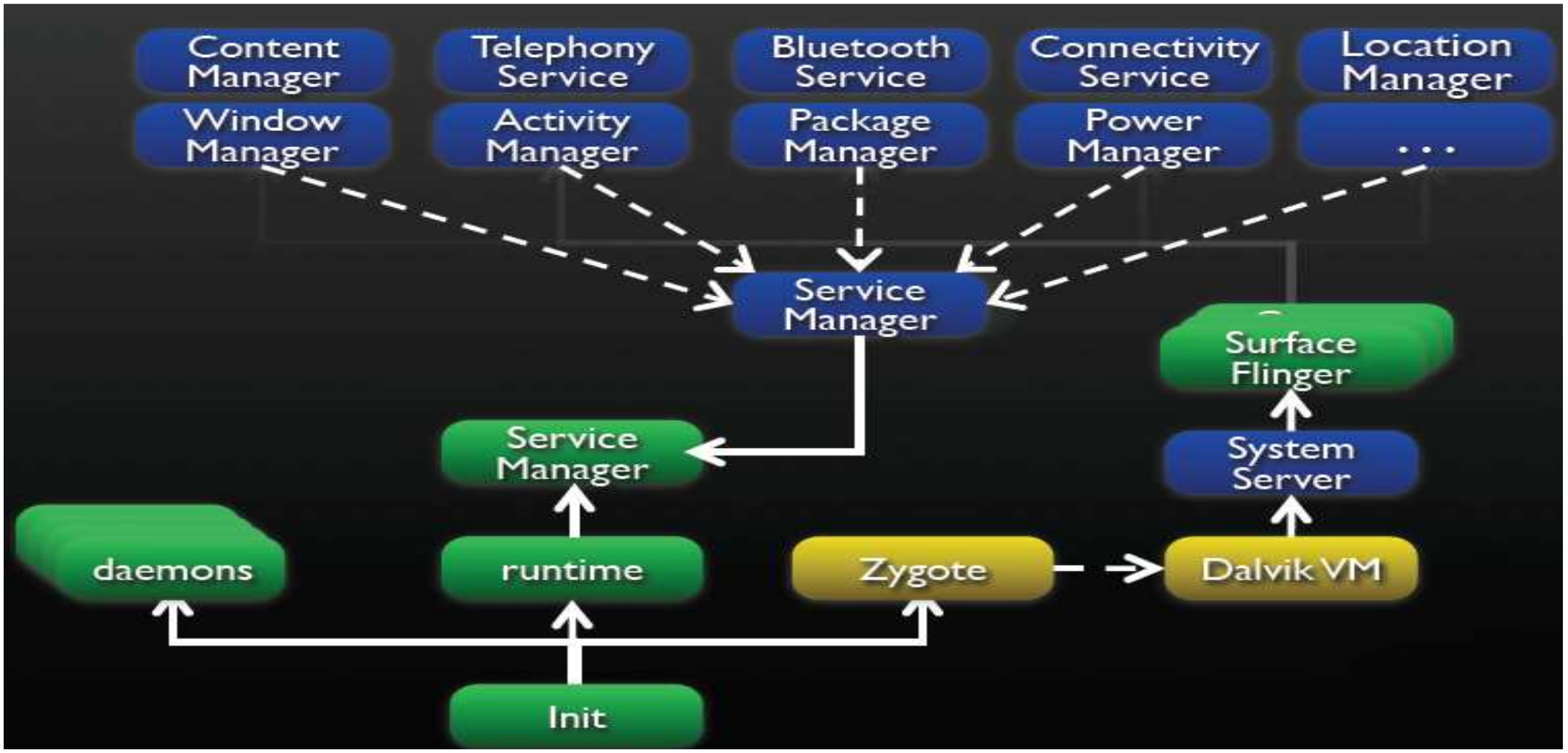
Mise en place du système



Mise en place du système



Mise en place du système



Dalvik VM

- Il ne s'agit pas d'une JVM, elle exécute son propre byte-code
- Les fichiers .class font place à un fichier .dex
- architecture à registres par opposition à l'architecture à pile de la JVM
- Une VM par processus et donc par application
- Les VMs sont clonées et non crées (processus zygote)

Dalvik VM

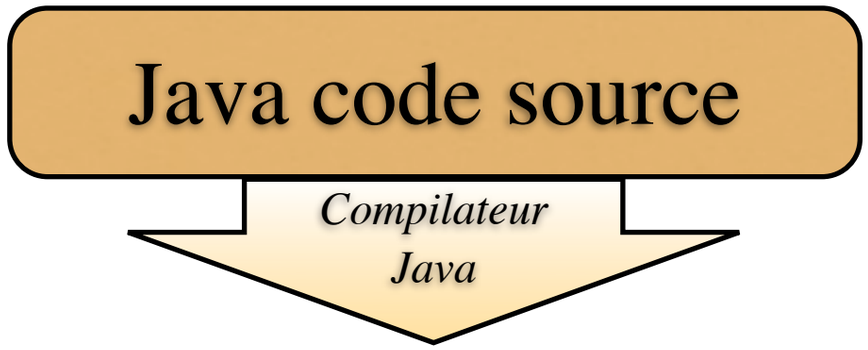
- Il ne s'agit pas d'une JVM, elle exécute son propre byte-code
- Les fichiers .class font place à un fichier .dex

Java code source

- architecture à registres par opposition à l'architecture à pile de la JVM
- Une VM par processus et donc par application
- Les VMs sont clonées et non crées (processus zygote)

Dalvik VM

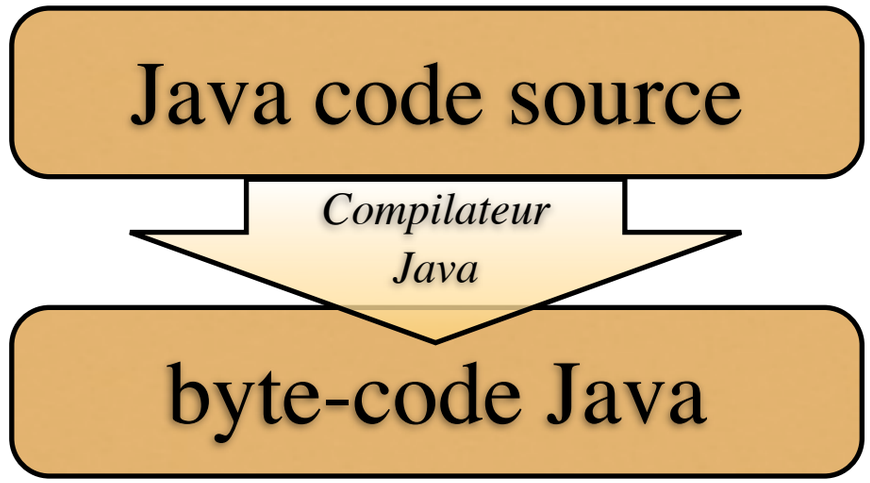
- Il ne s'agit pas d'une JVM, elle exécute son propre byte-code
- Les fichiers .class font place à un fichier .dex



- architecture à registres par opposition à l'architecture à pile de la JVM
- Une VM par processus et donc par application
- Les VMs sont clonées et non crées (processus zygote)

Dalvik VM

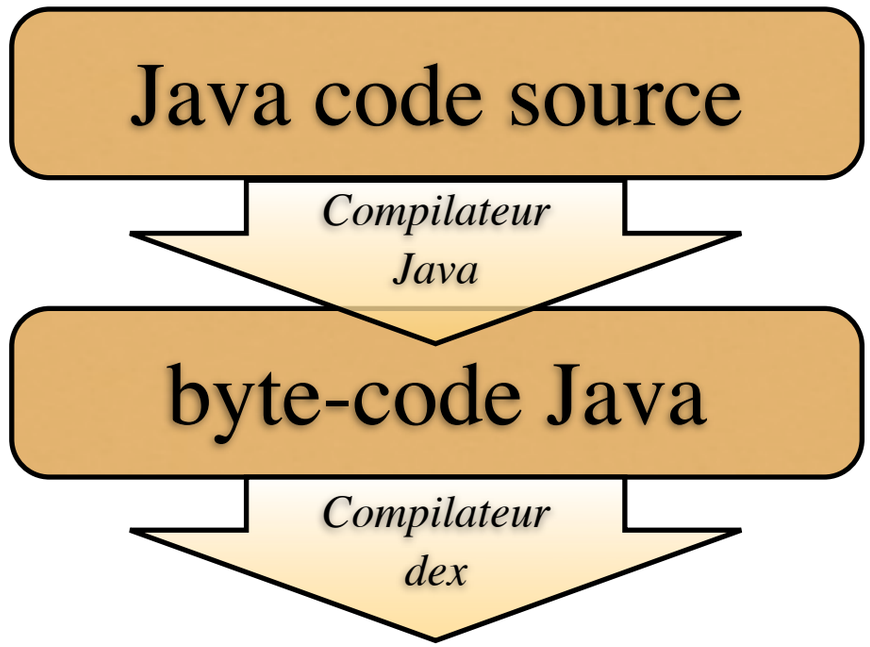
- Il ne s'agit pas d'une JVM, elle exécute son propre byte-code
- Les fichiers .class font place à un fichier .dex



- architecture à registres par opposition à l'architecture à pile de la JVM
- Une VM par processus et donc par application
- Les VMs sont clonées et non crées (processus zygote)

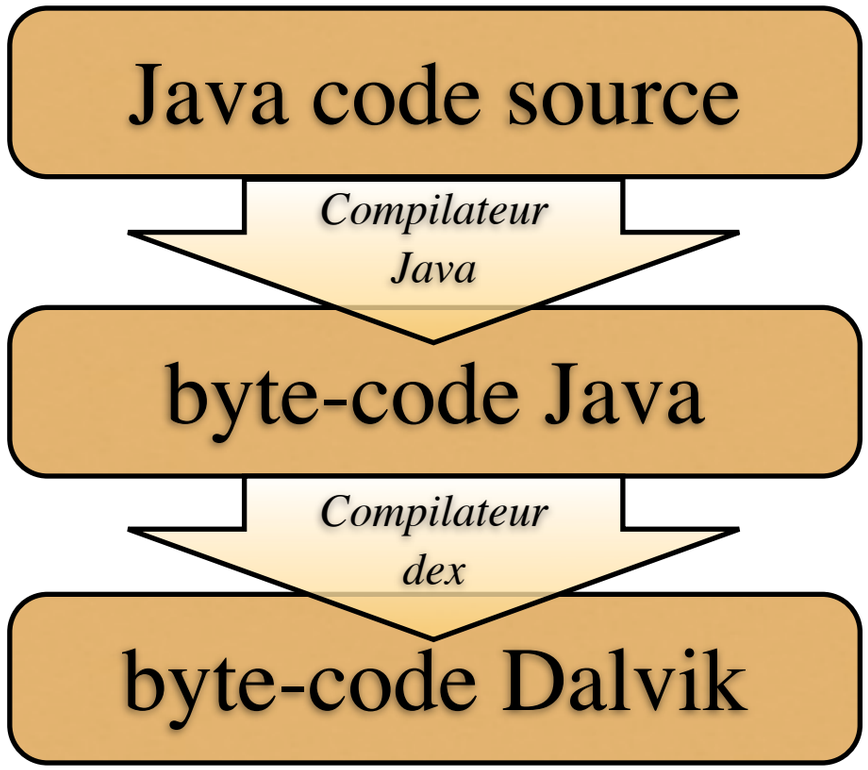
Dalvik VM

- Il ne s'agit pas d'une JVM, elle exécute son propre byte-code
- Les fichiers .class font place à un fichier .dex
- architecture à registres par opposition à l'architecture à pile de la JVM
- Une VM par processus et donc par application
- Les VMs sont clonées et non crées (processus zygote)



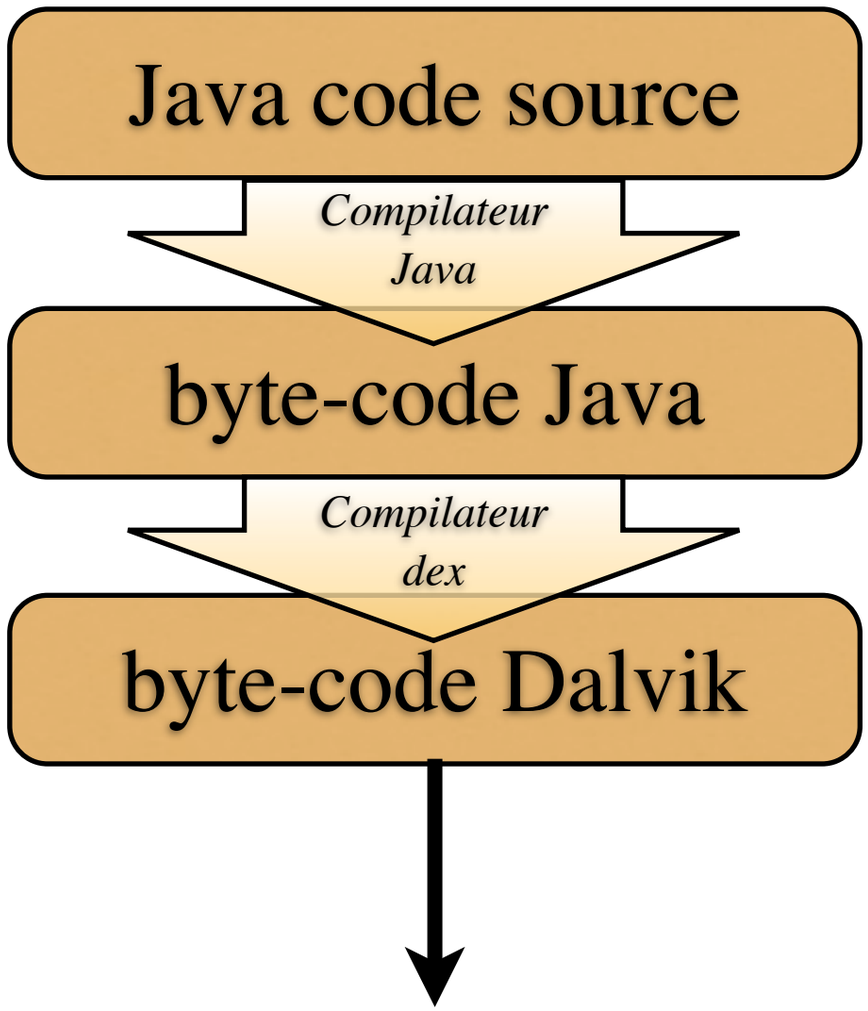
Dalvik VM

- Il ne s'agit pas d'une JVM, elle exécute son propre byte-code
- Les fichiers .class font place à un fichier .dex
- architecture à registres par opposition à l'architecture à pile de la JVM
- Une VM par processus et donc par application
- Les VMs sont clonées et non crées (processus zygote)



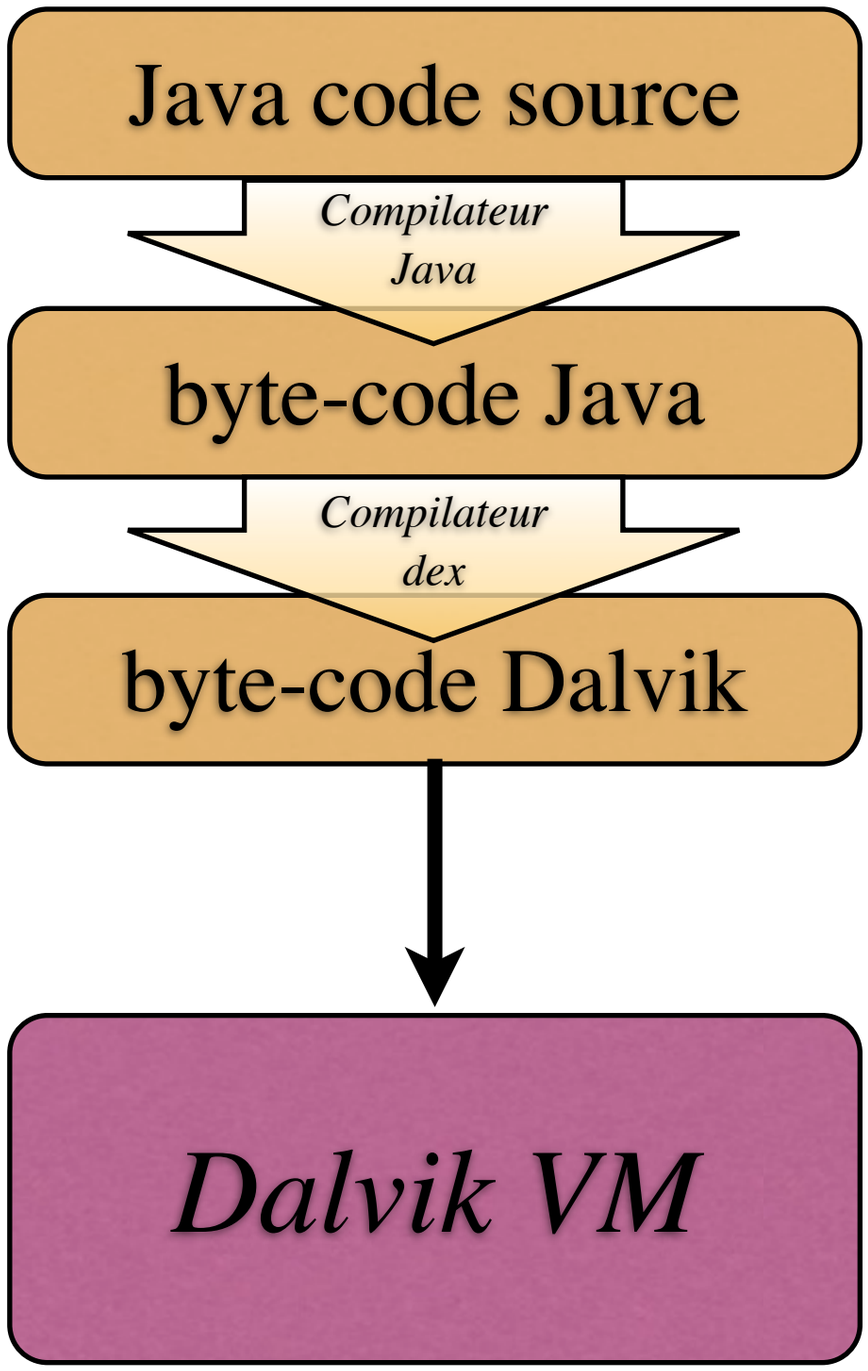
Dalvik VM

- Il ne s'agit pas d'une JVM, elle exécute son propre byte-code
- Les fichiers .class font place à un fichier .dex
- architecture à registres par opposition à l'architecture à pile de la JVM
- Une VM par processus et donc par application
- Les VMs sont clonées et non crées (processus zygote)



Dalvik VM

- Il ne s'agit pas d'une JVM, elle exécute son propre byte-code
- Les fichiers .class font place à un fichier .dex
- architecture à registres par opposition à l'architecture à pile de la JVM
- Une VM par processus et donc par application
- Les VMs sont clonées et non crées (processus zygote)



Applications Android

- Une application Android se concrétise sous la forme d'un fichier archive dont le nom possède l'extension .apk. Cette archive contient :
  - un fichier manifest décrivant le contenu de l'archive, la nature de l'application et ses points d'entrée
  - le code exécutable contenu dans un fichier .dex (éventuellement pré-lié pour améliorer les performance : .odex)
  - des fichiers de ressources contenant des données manipulées par l'application : image, son, description d'interface graphique, fichiers de localisation…

Applications Android


```
./AndroidManifest.xml
./classes.dex
./lib/armeabi/libtoto.so
./lib/x86/libtoto.so
./META-INF/CERT.RSA
./META-INF/CERT.SF
./META-INF/MANIFEST.MF
./res/anim/toto.xml
./res/layout/main.xml
./res/layout-land/main.xml
./res/layout-large/main.xml
./res/drawable-ldpi/icon.png
./res/drawable-mdpi/icon.png
./res/drawable-hdpi/icon.png
./res/drawable-xhdpi/icon.png
./res/menu/totoMenu.xml
./res/values/strings.xml
./res/values-fr/strings.xml
./res/values-es/strings.xml
```

Applications Android

- Une application installée dans un terminal Android, dispose d'un identifiant unique (user id), qui isolera l'application des autres sauf si on accorde des permissions explicitement (sandboxing utilisateur)
- Une application s'exécute dans un processus unique, lancé dès qu'un des points d'entré de l'application est sollicité
- Chaque application possède sa propre Dalvik VM
- Des exceptions peuvent-être consenties, quand plusieurs applications possèdent le même certificat (même signature => possibilité même uid ou même DVM)

Applications Android

- Le répertoire de travail : /data/data/packageName a la structure suivante :
  - ./databases contient les fichiers stockants les bases SQLite manipulées par l'application
  - ./lib contient les librairies natives utilisées par l'application sur l'architecture courante
  - ./files contient des fichiers manipulés par l'application
  - ./shared_prefs stockage particulier des données de praramétrisation de l'application
  - d'autres répertoires peuvent être présents cache, app_widgets…

Applications Android

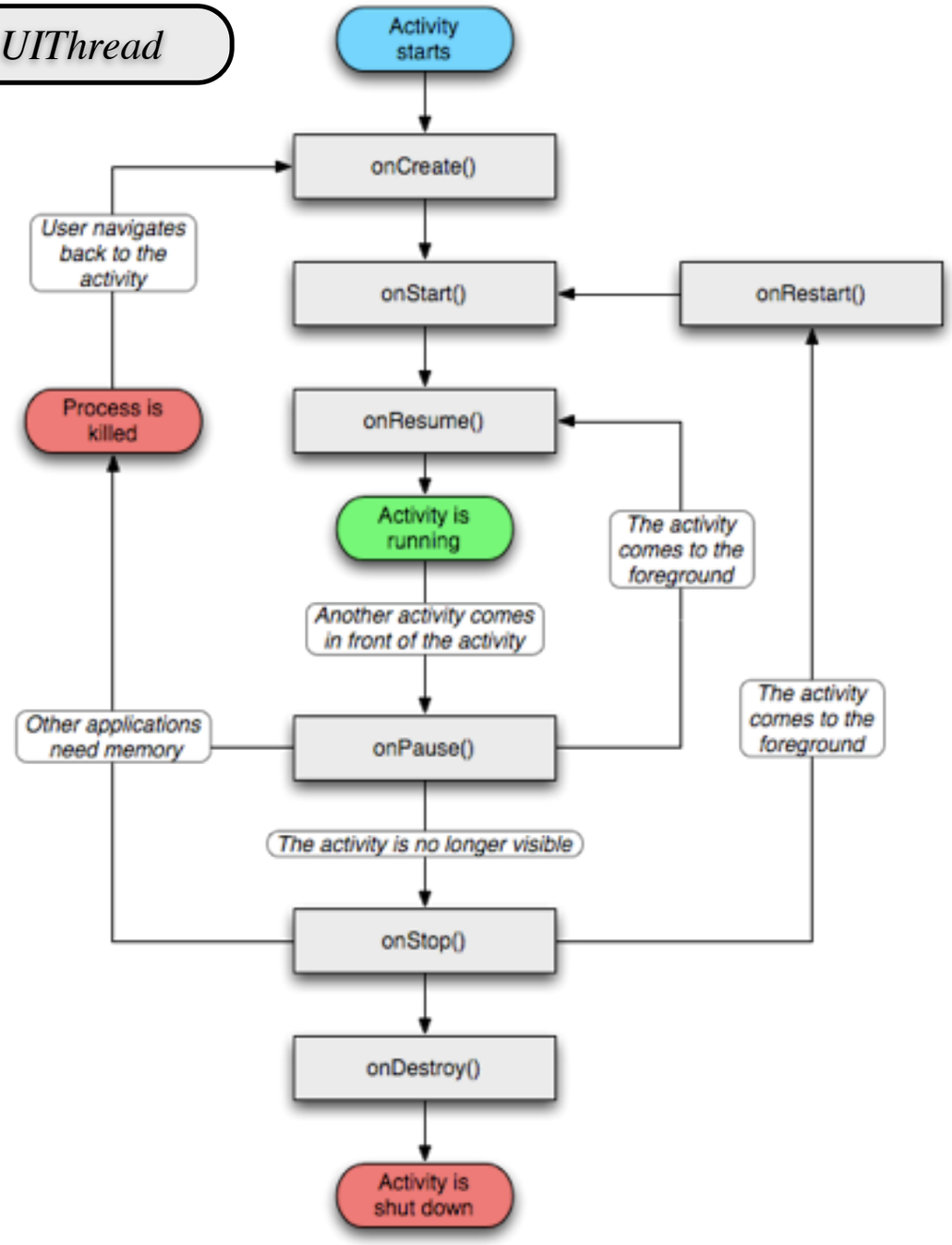
- Android propose un modèle à composants applicatifs, offrant plusieurs points d'entrées dans l'application. On distingue 4 types de composants :
  - Les activités (Activity)
  - Les services (Service)
  - Les fournisseurs de contenu (Content Provider)
  - Les traitants d'événements diffusés (Broadcast Reciever)

Les Activités

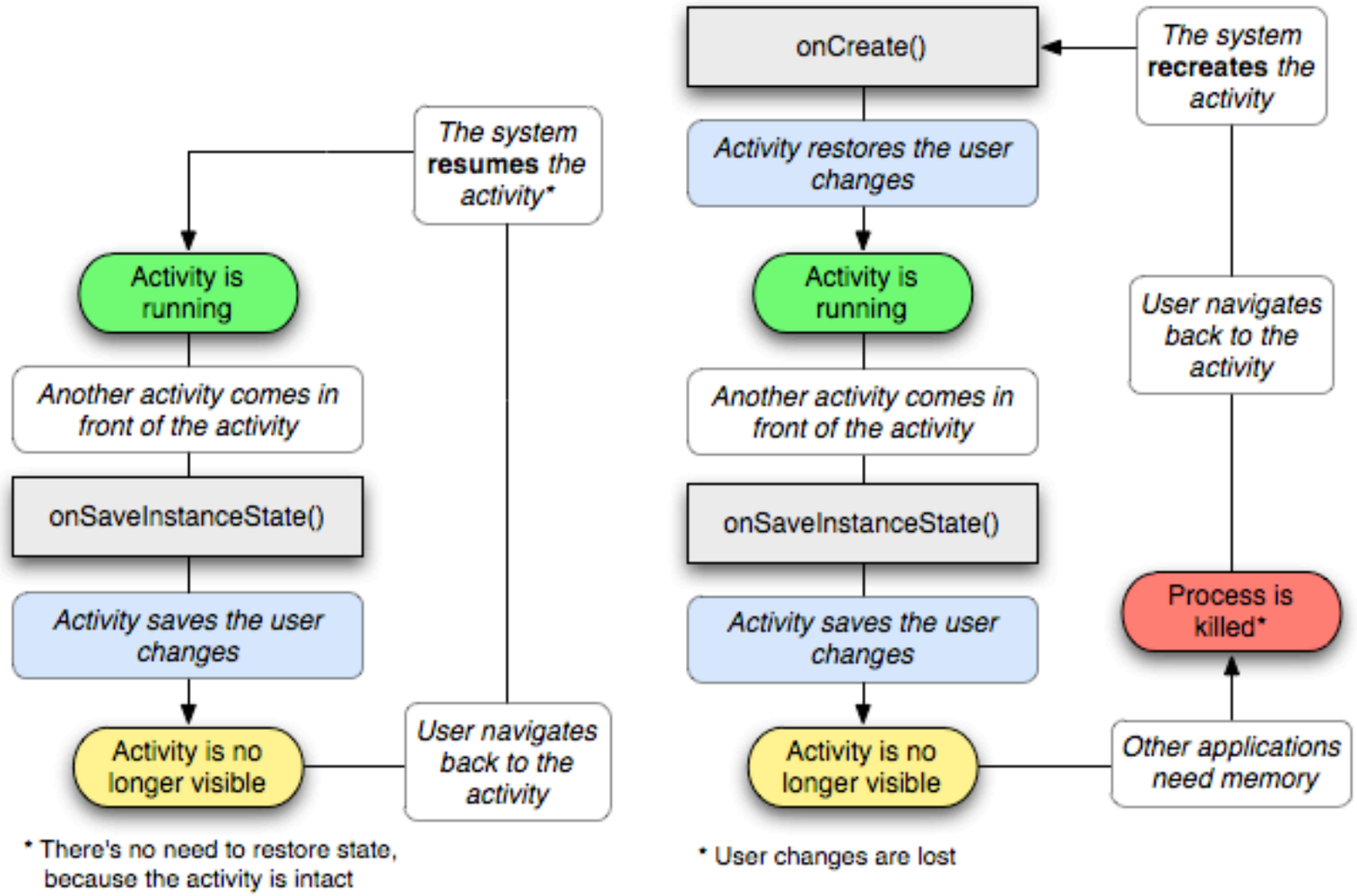
- Le modèle de comportement est celui d'une page Web
- Une activité "ne devrait pas avoir d'état propre" sauvegardé en mémoire (stateless)
- Composant central de l'interface utilisateur d'une application
- Chaque nouvel écran présenté à l'utilisateur est porté par une activité différente (sauf onglets)
- Le système gère la navigation (lancement, touche back…), dialogue avec des tâches gérant l'état ou les données

Cycle de vie d'une activité

StartActivity(Intent) → *UIThread*



État d'une activité



Impact sur la définition des IHM

- Modèle de la thread unique de gestion de l'affichage : UIThread => toutes les opérations attachées à l'activité doivent être brèves sous peine de bloquer la thread principale
- Un timer supervise une activité => arrêt forcé en cas de non réponse de l'activité à l'issu de la période de test
- Toute tâche complexe doit-être déléguée à une thread spécifique ou un autre composant applicatif non graphique afin de préserver la réactivité du système

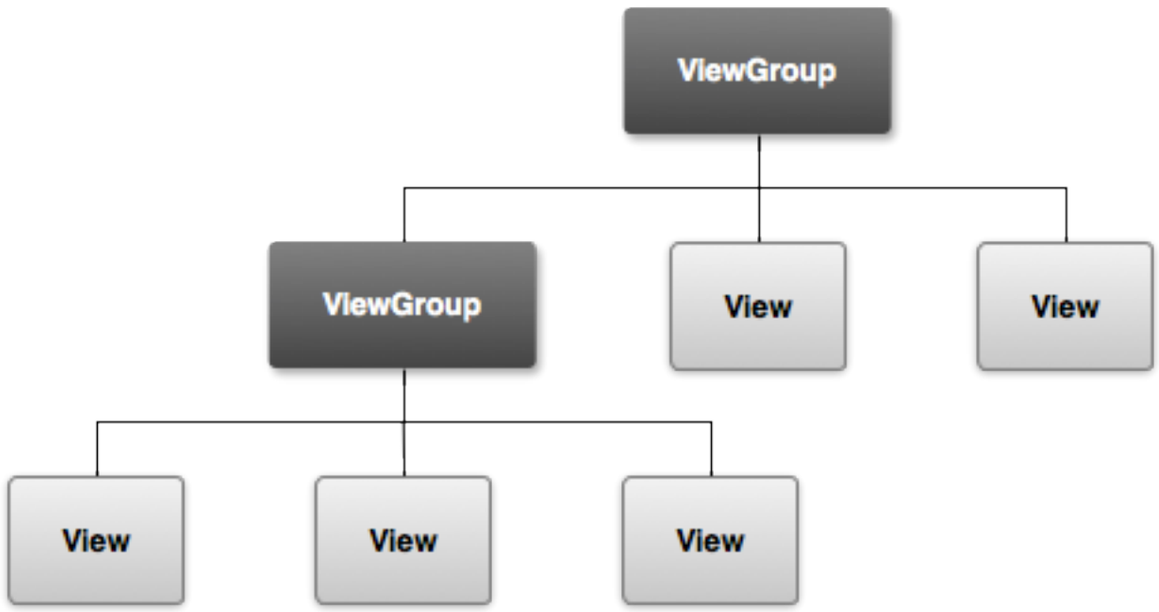
Interface Graphique

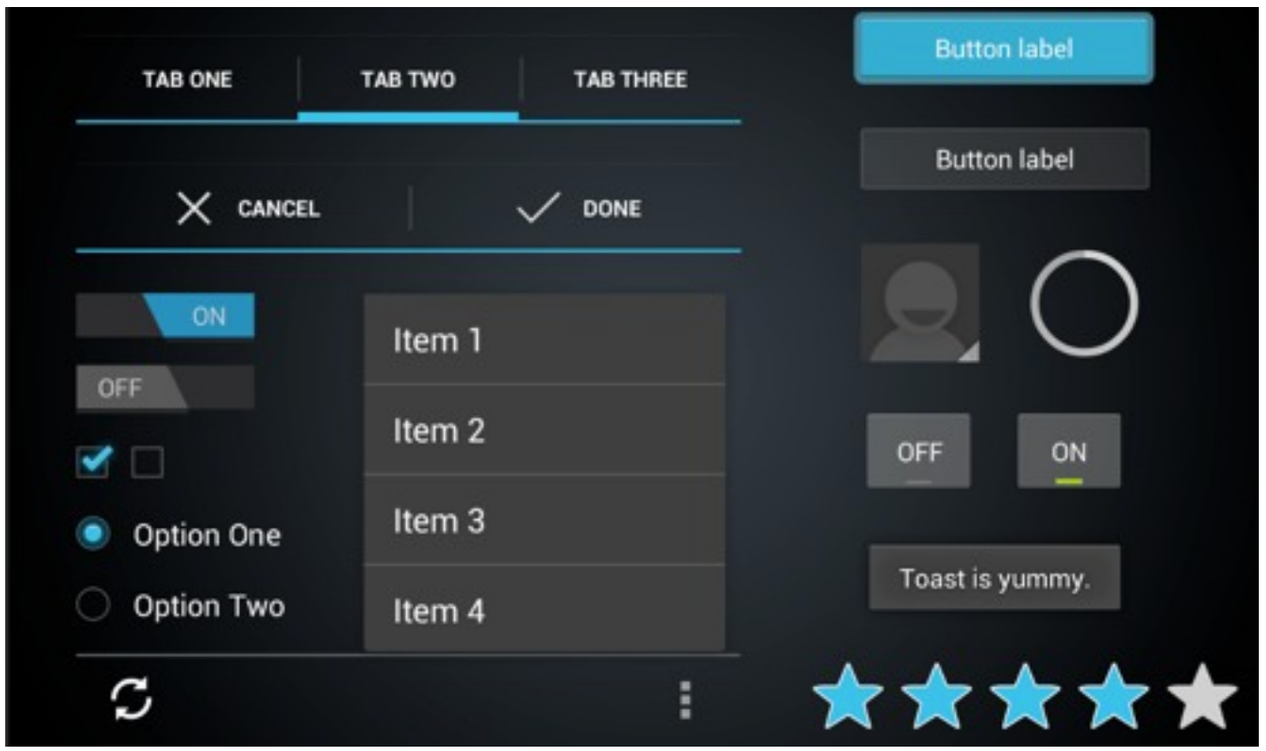
- 2 approches pour spécifier l'interface graphique :
  - approche programmatique classique
  - basée sur une description XML
- Framework : une structure hiérarchique faite de vues et de conteneurs. Extension du modèle avec les Fragments
- Le choix entre les 2 approches varie selon que l'interface soit dynamique ou non. Il est possible de mélanger les 2 approches
- Interaction à travers des EventListeners ou par redéfinition de callbacks

Spécification de l'IHM en XML

- On récupère la référence d'un objet créé en XML par la méthode **`findViewById()`** de la classe **`Activity`**.
- La classe R est mise à jour automatiquement par le SDK
- De très nombreux widgets sont disponibles : TextView, EditText, CheckBox, Button, ToggleButton, RadioGroup, Spinner, AutoCompleteTextView, DatePicker, TimePicker, ImageView, ImageButton, ProgressBar, RattingBar, Gallery…
- Idem pour les Layout : ListView, FrameLayout, LinearLayout, TableLayout…
- Les menus sont gérés par XML ou par programmation

Spécification de l'IHM en XML





Interaction avec l'utilisateur

- Une gestion spécifique de l'interaction utilisateur "l'objet physique" :
  - écran tactile -> masque l'écran pendant l'interaction
  - multi-points -> gestuelle (rendu mécanique)
  - utilisation et détournement des capteurs : proximité, luminosité, accéléromètres, gyroscopes, magnétomètres…
  - vibreur, diodes multicouleurs, flash led, environnement sonore…
  - GPS, Camera, le NFC…

Les Intents

- Objets (**`android.content.Intent`**) définissant des messages entre activités (intentions). Au centre du mécanisme de navigation entre activités. Un objet Intent comporte principalement :
  - le nom du composant cible (optionnel)
  - la déclaration d'une action à réaliser
  - des données (optionnelles) et/ou leur type MIME sur lesquels l'action doit porter
  - La catégorie de l'Intent pour différencier les composants cibles
  - flags, extras...

Les Intents

- Pas de communication directe, mais à travers des “appels systèmes” (**`Context.startActivity()`**, **`Activity.startActivityForResult()`**...)
- 2 modes d’utilisation :
  - explicite : utilise le nom désignant le composant cible
  - implicite : recherche un composant cible potentiel à partir des autres informations portées par l’Intent
- Recommandé d’utiliser une description précise

Les Intents

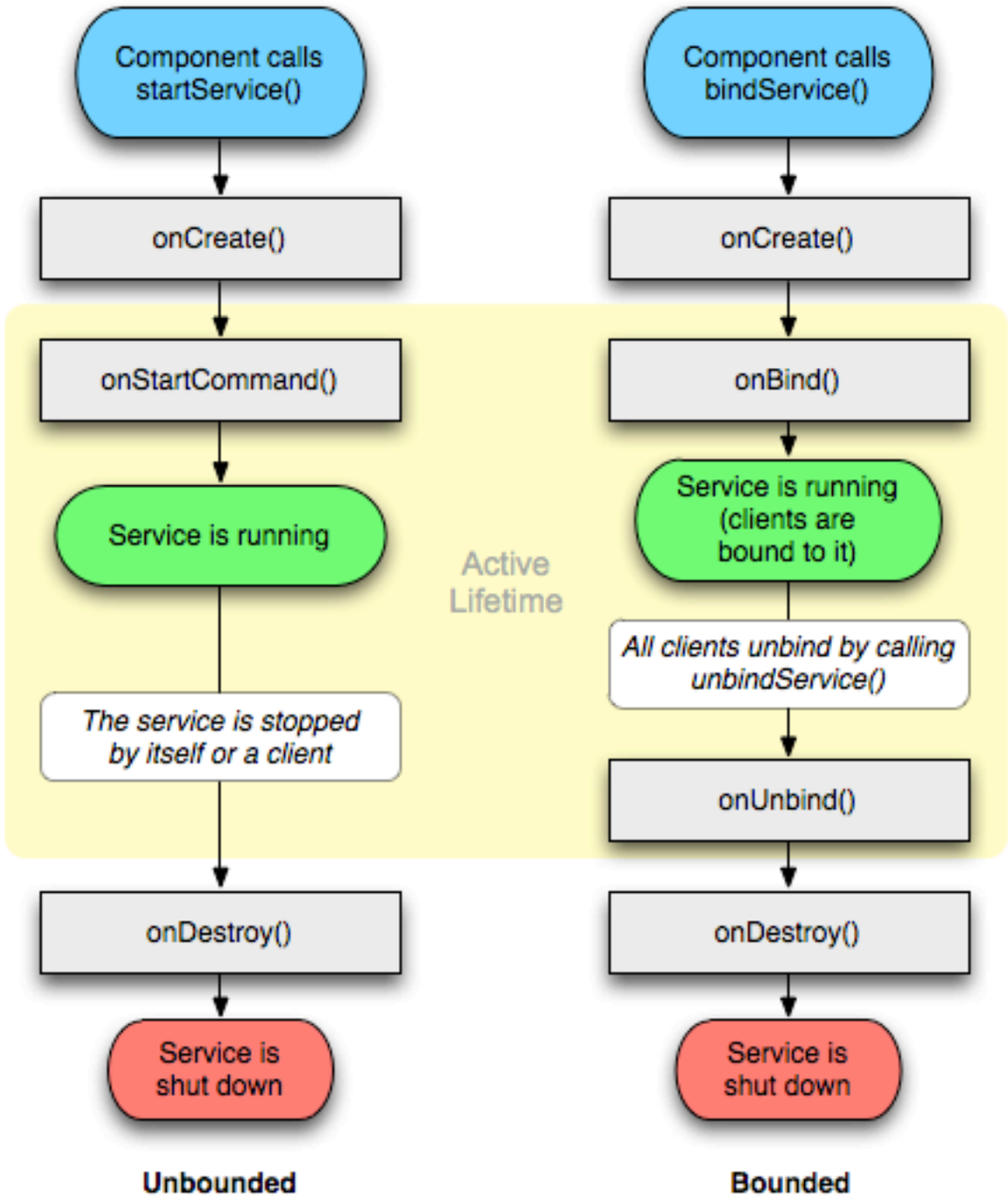
- Mécanisme de résolution du mode implicite :
  - Le runtime Android confronte l'Intent au déclarations "d'Intent-Filters" dans les fichiers Manifest des différentes applications
    - La comparaison commence par résoudre l'action, puis la catégorie, enfin le type de donnée ou une URI
  - Les activités sont associées à des "Intents-Filters"

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="video/mpeg" android:scheme="http" . . . />
    ...
</intent-filter>
```

Services

- Ne comporte pas d'interface graphique
- Perdure même si l'application n'est pas au premier plan
- Expose une interface regroupant des méthodes utilisables "à distance"
- Interrogeable par RPC : Android Interface Definition Language
- génération de souches permettant d'accéder au service
- la liaison au service est réalisé en utilisant un Intent. Une fois la liaison effectuée on appelle directement les méthodes de l'interface du service

Services



Les Content Providers

- Exposent des données mises à disposition par une application sous la forme d'une ou plusieurs tables (la mise en forme est à la charge du Provider)
- Une table définit un type de donnée et chaque ligne une instance de ce type de donnée
- L'accès aux données exportées se fait par l'utilisation d'un ContentResolver
- Accès via URI : content://package_name_provider/*type*/*id*
- requêtes : query, update, insert, delete

Les Content Providers

- La connexion entre ContentProvider et ContentResolver se fait par IPC
- Requêtes sur les données similaires à SQL
- déclaration dans le manifest de l'application : balise <provider>
- Initialisation à la première utilisation onCreate()
- Pas de callback particulière à la libération des ressources -> gérer la sauvegarde des données après chaque requête

Les Broadcast Receivers

- Sous-classe de **`android.content.BroadcastReciever`**
- Implantation d'un bus logiciel à message pour Android
- Les événements sont des Intents, diffusés par des "appels systèmes" (**`Context.sendBroadcast()`**, **`Context.sendOrderedBroadcast()`**)
- Déclaration dans le fichier Manifest (élément receiver avec Intent-Filters + permissions) ou (**`Context.registerReciever()`**)
- Le traitement de l'Intent par redéfinition de **`onRecieve()`**
- Attention : géré par l'UIThread