



SYSTÈME D'EXPLOITATION MOBILE

Dr. Amel OURAHMOUNE
aourahmoune@usthb.dz
2020-2021

RÉFÉRENCES

- <https://developer.android.com/>
- Michael J. Jipping. Smartphone Operating System Concepts with Symbian OS: A Tutorial Guide. ISBN: 978-0-470-03449-1. Wiley, 2007
- Arash Habibi Lashkari, Mohammadreza Moradhaseli. Mobile Operating Systems and Programming: Mobile Communications. 2011
- Jonathan Levin. Android Internals: A Confectioner's Cookbook. 2014
- Karim Yaghmour. Embedded Android: Porting, Extending, and Customizing. 2013
- Earlence Fernandes. Instant Android Systems Development How-to. 2013
- Joshua J. Drake, Zach Lanier, Collin Mulliner, Pau Oliva Fora, Stephen A. Ridley, Georg Wicherski. Android Hacker's Handbook. 2014

CHAPITRE 1 : SYSTÈME EMBARQUÉ

Un système embarqué: c'est une combinaison de matériels et logiciels permettant de remplir une ou plusieurs fonctions spécifiques avec des contraintes plus ou moins sévères tel que la consommation, la température, la taille, les performances...[Patrice Kadionik, 2004]

Un système embarqué est susceptible d'être utilisé dans un environnement matériel de faibles performances (si l'on compare au PC de bureau d'aujourd'hui). Si l'ajout de quelques Mo de mémoire sur un PC de bureau n'a pas une grosse influence sur le budget d'un utilisateur, le gain de quelques Mo sur un produit de grande consommation (téléphone, équipement auto mobile, organisateur personnel) a une énorme influence sur le coût final. [Pierre Ficheux, 2003]

CHAPITRE 1 : SYSTÈME EMBARQUÉ

c'est un système électronique et informatique autonome qui est dédié à une tâche particulière et contenue dans un système englobant.

- Matériel et application intimement liés
- Radio/réveil
- Machine à café
- Télévision / télécommande
- Moyen de transport
- Téléphone portable

CHAPITRE 1 : SYSTÈME EMBARQUÉ

Un système embarqué a :

- des ressources limitées
- Système principalement numérique
- Le moins cher possible
- Une puissance de calcul limitée
- Pas de consommation d'énergie inutile
- Exécution de logiciel dédié aux fonctionnalités spéciales
- Une capacité de communication limitée
- Ne possède pas toujours de système de fichiers

CHAPITRE 1 : SYSTÈME EMBARQUÉ

Faible coût:

- Solution optimale entre le prix et la performance
- À la portée de toute personne
- Par conséquent, les ressources utilisées sont minimales
- Un système embarqué n'a que peu de mémoire

Faible consommation:

- Utilisation d'une batterie d'emmagasins d'énergie
- Gérer la consommation pour rester autonome le plus possible
- Pas de consommation excessive, moins de prix et des batteries de faible capacités

CHAPITRE 1 : SYSTÈME EMBARQUÉ

Faible encombrement et faible poids:

- Minimiser la taille et le poids pour un système embarqué.
- Les composants électroniques (analogique et/ou numérique) doivent cohabiter sur une faible surface.

Fonctionnement en temps réel:

- Les applications embarquées doivent répondre rapidement aux événements internes ou externes.
- Nécessaire dans les applications de système de contrôles
- Le résultat peut être néfaste si le système ne réagit pas à l'immédiat à un événement du système

CHAPITRE 1 : SYSTÈME EMBARQUÉ

Un système embarqué est soumis à des nombreux contraintes d'environnement

Il doit s'adapter et fonctionner avec eux.

Exemple:

- La température
- L'humidité
- Les vibrations
- Les chocs
- Les variations d'alimentation, les interférences RF, les radiations... etc

CHAPITRE 1 : SYSTÈME EMBARQUÉ

L'EMBARQUÉ EN QUELQUES CHIFFRES

En 1999, il a été vendu (dans le domaine de l'embarqué):

1,3 milliards de processeurs 4 bits

1,4 milliards de processeurs 8 bits

375 millions de processeurs 16 bits

127 millions de processeurs 32 bits

3,2 millions de processeurs 64 bits.

CHAPITRE 1 : SYSTÈME EMBARQUÉ

Il a été vendu 108 millions de processeurs pour le marché du PC

En 2004:

14 milliards de processeurs pour l'embarqué (microprocesseur, microcontrôleur, DSP, etc.)

260 millions de processeurs PC.

Moins de 2% (5%) des processeurs vendus sont pour les PC, 98% (95%) pour l'embarqué

Prix moyen d'un processeur 6\$ (2004) alors qu'un processeur PC coûte 300\$.

CHAPITRE 1 : SYSTÈME EMBARQUÉ

ARCHITECTURE D'UN SYSTÈME EMBARQUÉ

Équipements permanents:

CPU: microprocesseur (s) ou des microcontrôleurs

RAM: mémoire centrale

Équipements supplémentaires:

Mémoire de masse:

Disque dur (exp; microdrive 2,5-3,5 inches)

Mémoire flash (exp; FlashDisk, DiskOnChip, SD Card,...)

Utilisation de ROM (exp; Disque virtuel CD, DVD)

Disque à distance (exp; NFS, TFTP)

CHAPITRE 1 : SYSTÈME EMBARQUÉ

Entrées:

Les capteurs/convertisseurs (pression, audio, température,..)

Le clavier, boutons poussoirs ou télécommandes (infrarouge, Bluetooth, radio,..)

Les lecteurs de codes barres

Sorties:

Les écrans et afficheurs LCD

Système d'alarme ou synthèse vocale

Imprimante en tous genres comme papier, étiquette, photos, ..)

IHM:

Communication entre l'humain et la machine

Exp; écran avec les dispositifs « touchScreen»

CHAPITRE 1 : SYSTÈME EMBARQUÉ

Les domaines d'application:

Transport; automobile, aéronautique

Militaire; missile

Astronautique; fusée, satellite artificiel

Électroménager; télévision, four ou micro-ondes

Télécommunication; téléphonie, routeur, pare-feu

Impression; imprimante multifonctions, photocopieur

Informatique; disque dur, lecteur de CD

Équipement médical

Multimédia; console de jeux vidéo

Guichet automatique bancaire (GAB)

métrologie

CHAPITRE 1 : SYSTÈME EMBARQUÉ

DISPOSITIFS MOBILES

Un appareil informatique portable possédant souvent un écran et une interface d'entrée/sortie avec des dispositifs d'interaction nécessaire ou accessoires

Classification des dispositifs mobiles suivant leurs caractéristiques:

Laptop

Tablet PC

PDA (PersonalDigital Assistant)

Téléphone portable

Smartphone

Autres dispositifs; baladeur multimédia personnels (MP3, MP4, ...), consoles de jeux portables

SYSTÈME D'EXPLOITATION MOBILE

NÉCESSITÉ D'UN SYSTÈME D'EXPLOITATION

Les solutions embarquées utilisent des composants Soft conjointement avec le Hard

Par analogie aux ordinateurs, ces composants logiciels devront tournés sur un système d'exploitation.

un système d'exploitation embarqué n'a pas toutes les fonctionnalités et les caractéristiques qu'un un système d'exploitation pour ordinateur.

SYSTÈME D'EXPLOITATION MOBILE

Les principaux systèmes d'exploitation mobiles :

Android

Bada

BlackBerryOS

iOS

OpenMoko

PalmOS

HP webOS

SymbianOS

Windows CE

Windows Mobile

Windows Phone 7

SYSTÈME D'EXPLOITATION MOBILE

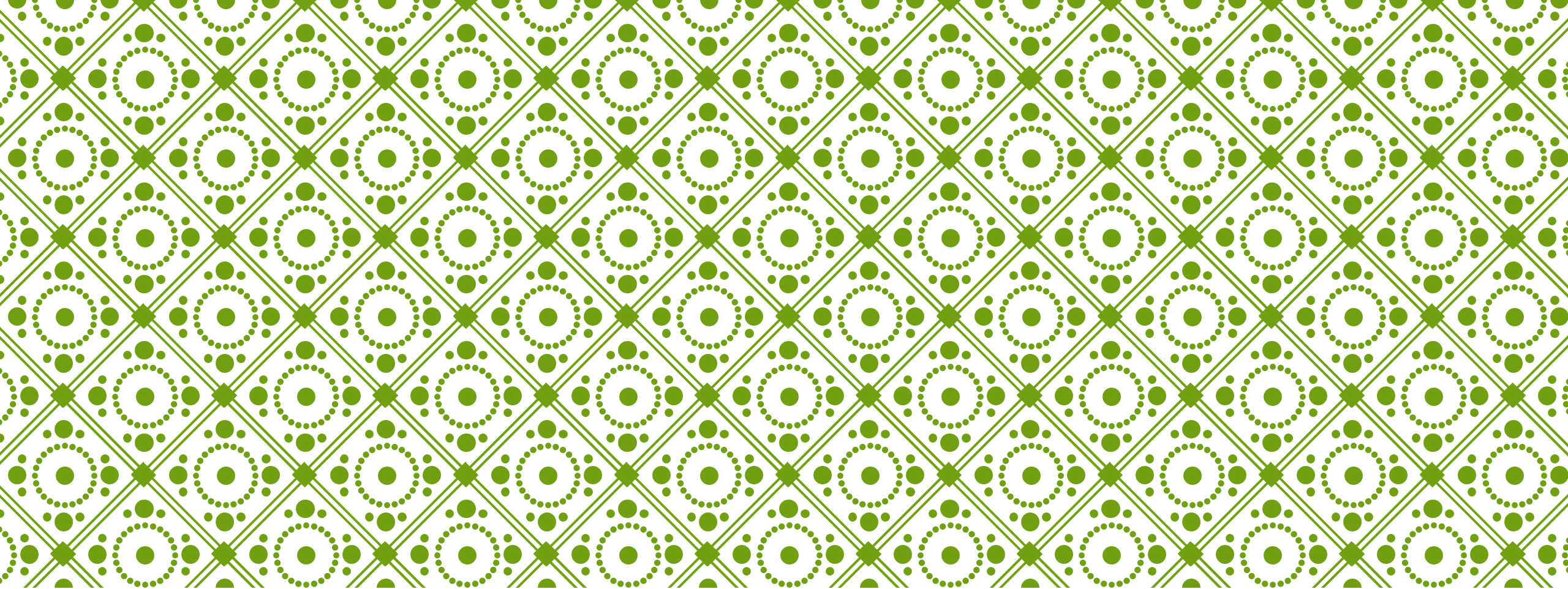
Un système d'exploitation mobile est un ensemble de programmes responsable de la gestion des opérations de:

- contrôle,
- coordination,
- utilisation du matériel
- partage des ressources d'un dispositif entre divers programmes tournant sur ce dispositif
- Les système d'exploitation mobile se diffèrent en fonction des fonctionnalités qu'ils soutiennent.

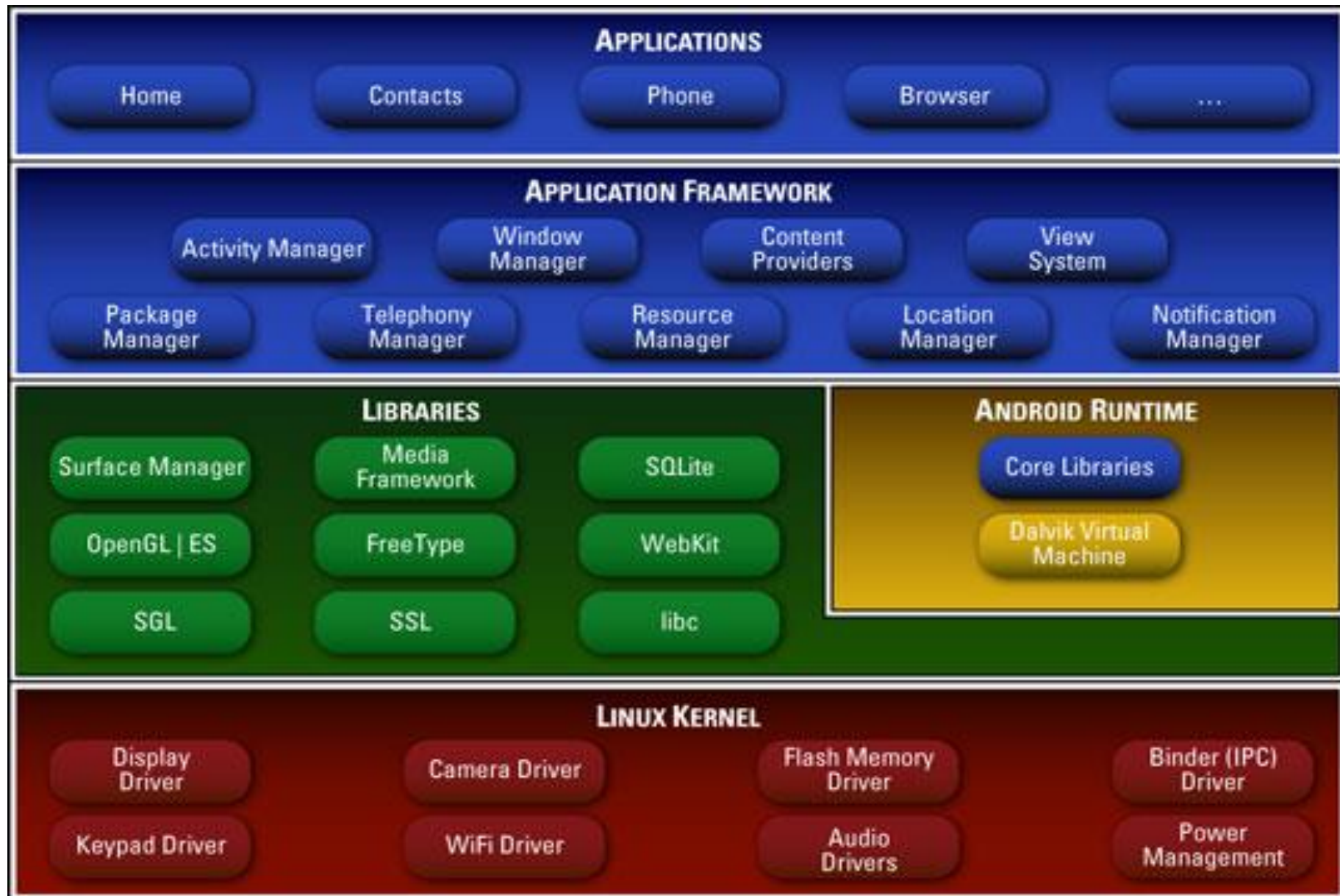
SYSTÈME D'EXPLOITATION MOBILE

Un système d'exploitation mobile regroupe un ensemble des fonctionnalités;

- La gestion de la mémoire
 - La gestion des microprocesseurs et l'ordonnancement
 - La gestion de système de fichiers
 - La gestion des I/O
 - La gestion de sécurité
 - La gestion de fonctionnalité multimédia
- etc



SYSTÈME ANDROID



APPLICATION

Android est livré avec un ensemble d'applications de base

Client email, programme SMS, calendrier, maps, navigateur, contacts, etc.

Les applications sont écrites en Java

APPLICATION FRAMEWORK

Un ensemble de services qui forment collectivement l'environnement dans lequel les applications Android s'exécutent et sont gérées

Content Providers : Permettre aux applications d'accéder aux données d'autres applications (Contacts) - de partager leurs propres données,

Activity Manager – gère le cycle de vie de l'application

Location Manager – fournit l'accès aux services de localisation (GPS, AGPS...)

Package Manager – maintient les informations sur les applications disponibles sur l'appareil

Notification Manager – permet aux applications d'afficher des alertes et notifications personnalisées

Resource Manager – fournit l'accès aux ressources (non-code embedded resources) telle que les chaînes de caractères, les graphiques.

Telephony Manager – fournit l'accès aux services de téléphonie (Appels vocaux) ainsi que certaines informations de l'abonné (Numéros de Tel.)

Window Manager – effectue la gestion des fenêtres

View System Un ensemble riche et extensible de vues utilisées pour créer les interfaces des applications : listes, grilles, boutons, zones de texte, etc.

LIBRERIES

Recueillir une multitude de bibliothèques spécifiques à Android et des bibliothèques tierces

Fonctionnalités de niveau-bas du système

Légères et chargées dans chaque processus, rapide

- **Bionic libc** est une bibliothèque C/C++:
- **SurfaceManager** est utilisée pour dessiner des surfaces sur l'écran, etc.
- Blink, SQLite, OpenSSL venant du monde du logiciel libre

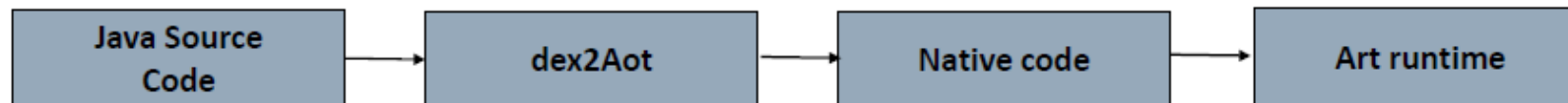
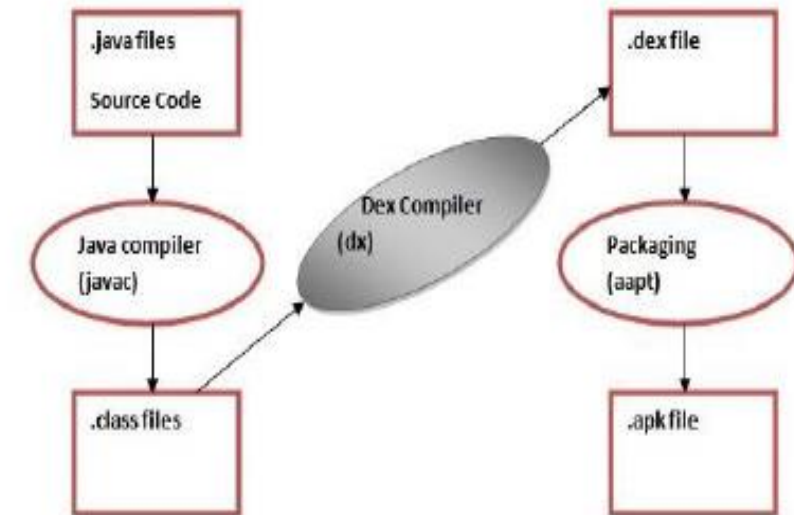
ANDROID RUNTIME

Gère l' exécution des applications Android

Core libraries – Biblio spécifiques à la MV Dalvik/Biblio d'interopérabilité Java/Biblio Android

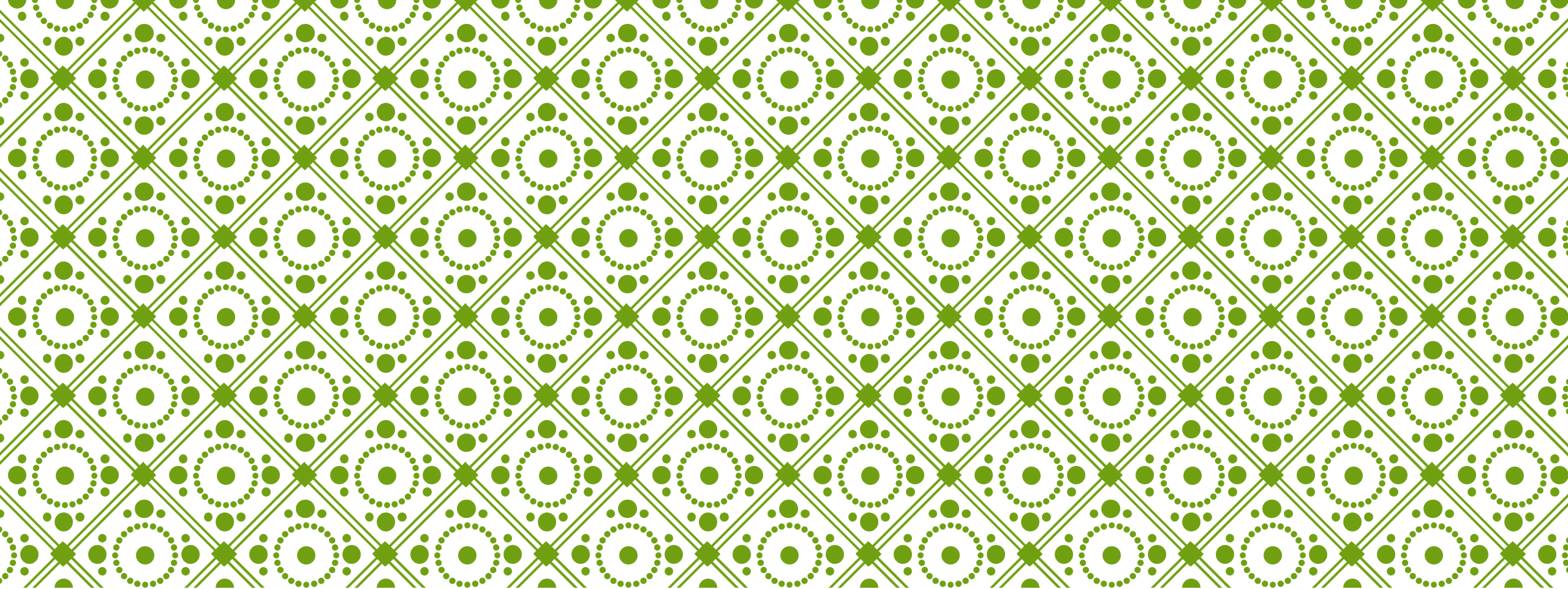
Android Runtime Prédécesseur Dalvik

- l'environnement d'exécution
 - basé sur architecture de registre
 - Exécute le format Dalvik Executable (.dex) .java ---.class-
 - ART Ahead-Of-Time (AOT) compiled runtime

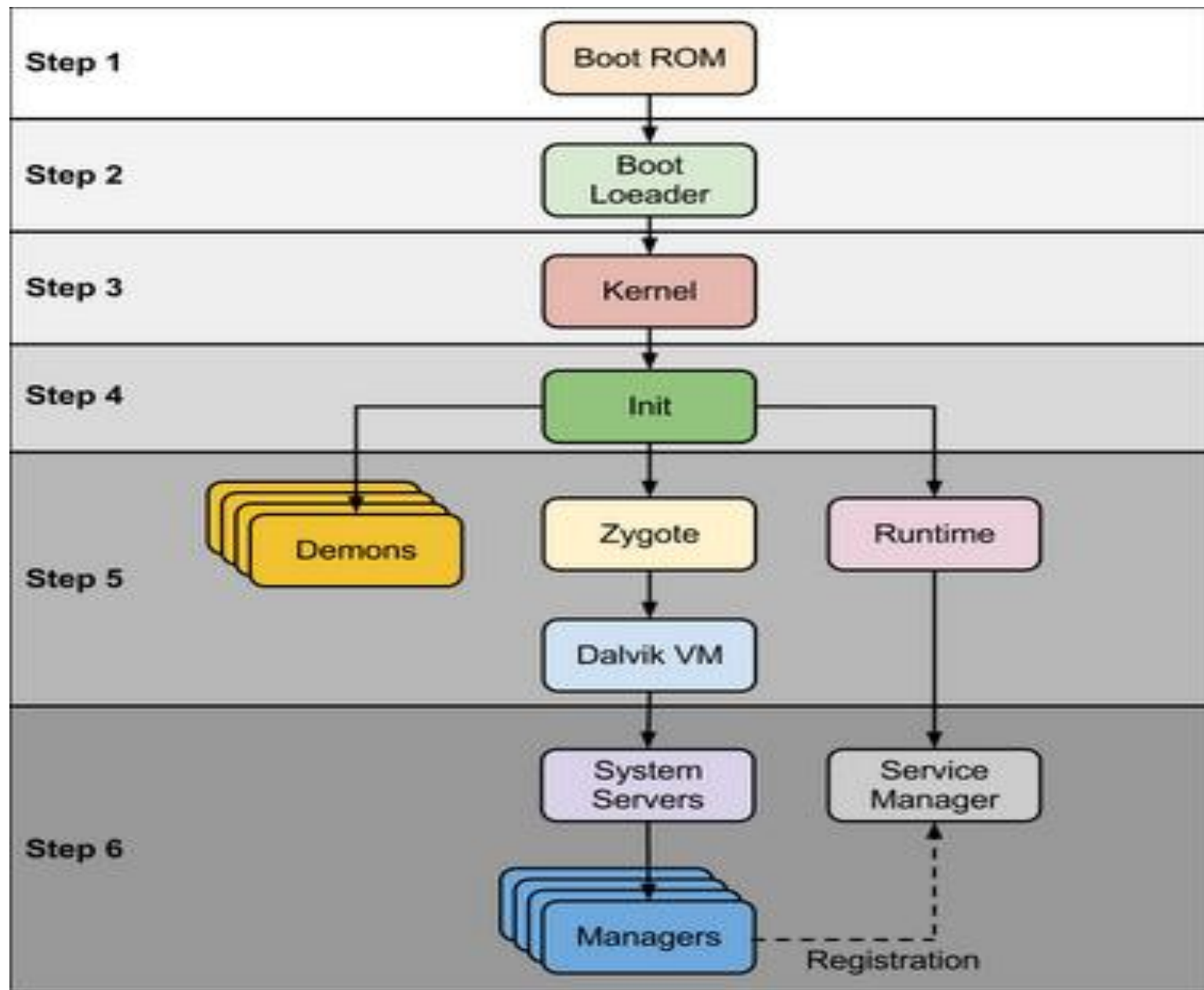


NOYAU LINUX

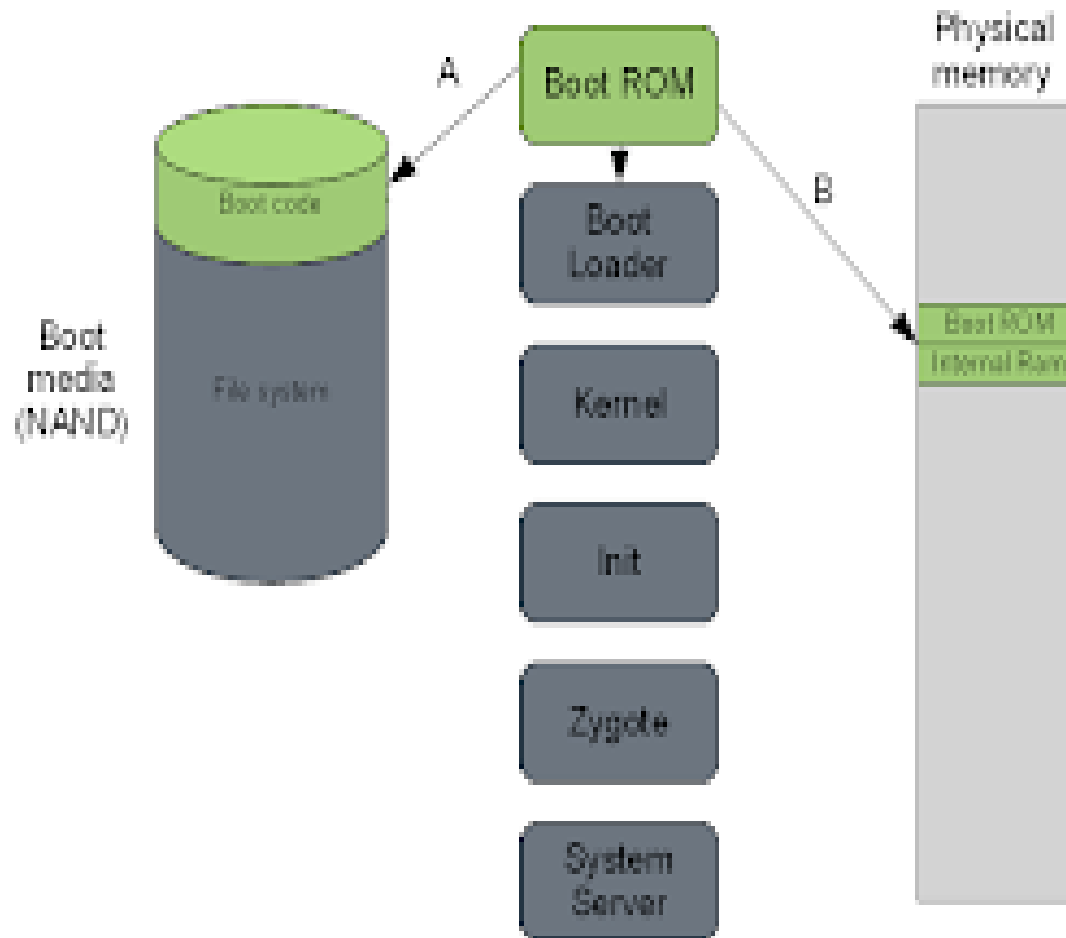
- Basé sur le noyau Linux
- Interagit avec le matériel, contient les pilotes des périphériques camera, audio, clavier, affichage, ...
- Gestion de processus, Gestion de mémoire, Gestion de réseau, ...
- Des spécifications supplémentaires propres aux plateformes embarquées mobiles (wake locks, Binder IPC driver, gestion de l'énergie)
- Binder IPC: communication inter processus :
 - Chaque App est lancé dans un processus particulier
 - Le binder assure le partage des ressources basé sur le partage de mémoire
 - Gère les problèmes de concurrence et de synchronisation



PROCESSUS DE BOOT



BOOT ROM



BOOT ROM

- Le processus d'exécution démarre le « **Boot Rom code** »
- Le boot ROM code détecte le NAND (la mémoire interne du téléphone) en utilisant un registre système qui est lié à des spécificités physiques de l'ASIC. Cela permet de déterminer où trouver le "first stage" du bootloader.
- le boot ROM charge le first stage du boot loader dans la RAM interne. le boot ROM code va passer la main et son exécution continue dans le bootloader.

BOOTLOADER

(A) Détecter et mettre en place la mémoire vive (RAM) externe

(B) Le first stage va charger le « main boot loader » et le placer dans la RAM externe.

(C) Le second stage du boot loader est lancé.

- initialiser des fichiers systèmes, de la mémoire additionnel, des pilotes réseaux et d'autres choses.

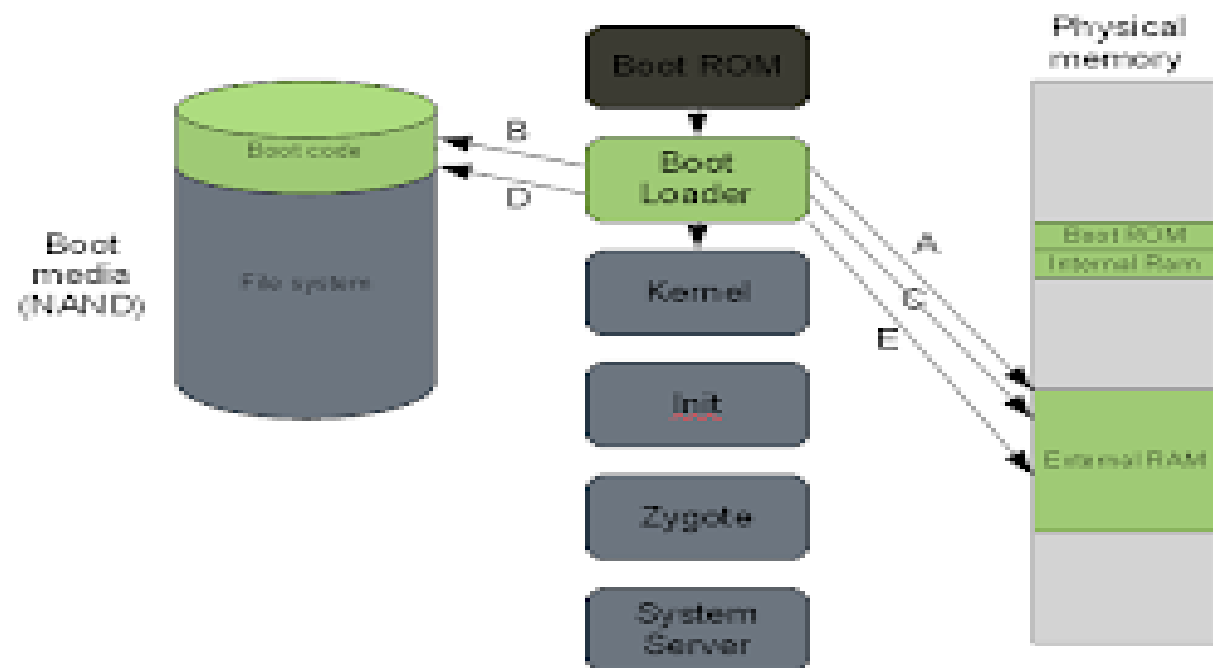
- lancer le code pour le modem CPU et initialiser des protections bas-niveau de la mémoire (low level memory protections) et des options de sécurité.

(D) Lancer le kernel Linux. depuis le boot media (ou depuis d'autres sources dépendant de la configuration du système) et le placer dans la RAM.

- Il place d'autres paramètres de démarrage dans la mémoire.

(E) Une fois que le bootloader a terminé, il va passer la main au noyau Linux, habituellement en effectuant quelques opérations de décompression. Le kernel assure alors la responsabilité du système

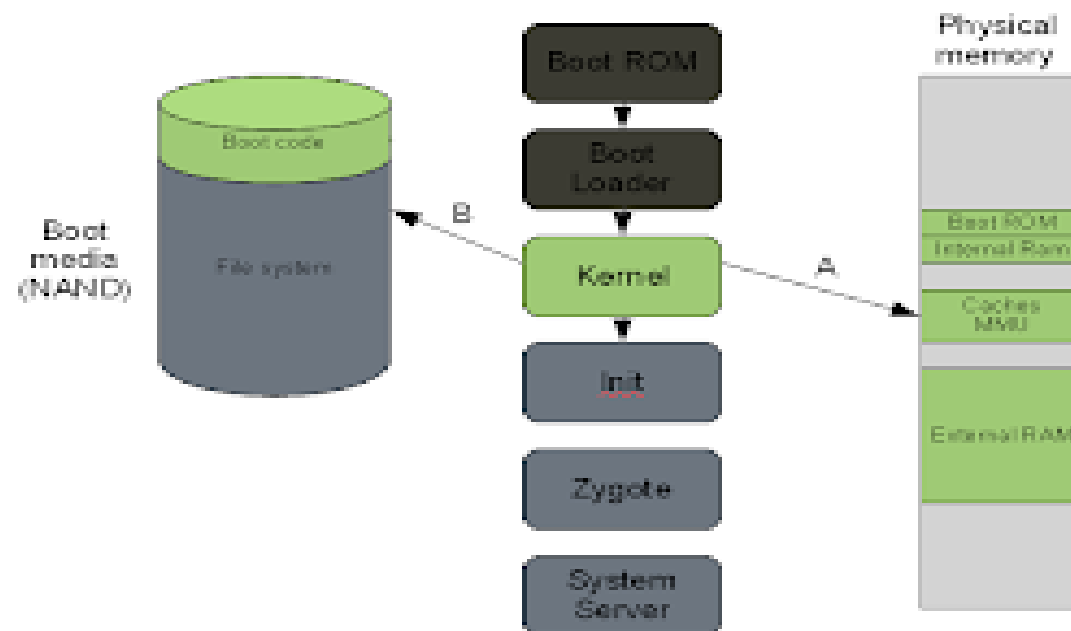
BOOTLOADER



BOOTLOADER

- initialise les contrôleurs d'interruptions, les protections de mémoire, le cache et le scheduling.
- le système utilise la mémoire virtuelle et de lance des processus utilisateur.
- Le noyau lance comme étant le tout premier processus utilisateur le « **init process** »
(`/system/core/init` dans le code source d'android, une fois compilé il se trouve par exemple à la racine du NAND pour le nexus 4)

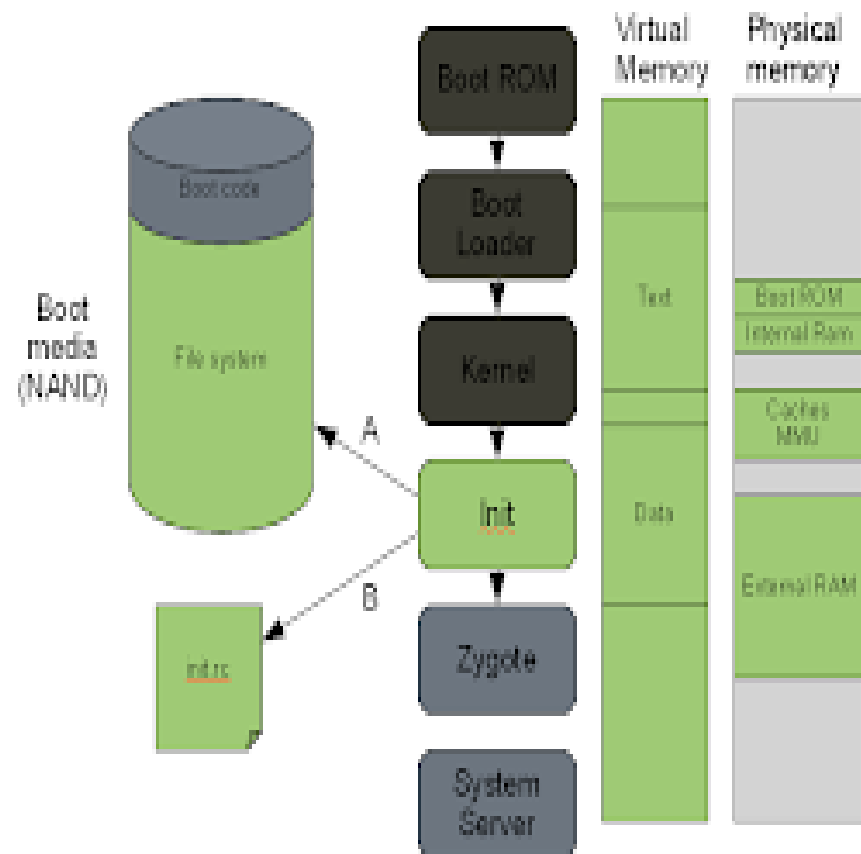
BOOTLOADER



INIT PROCESS

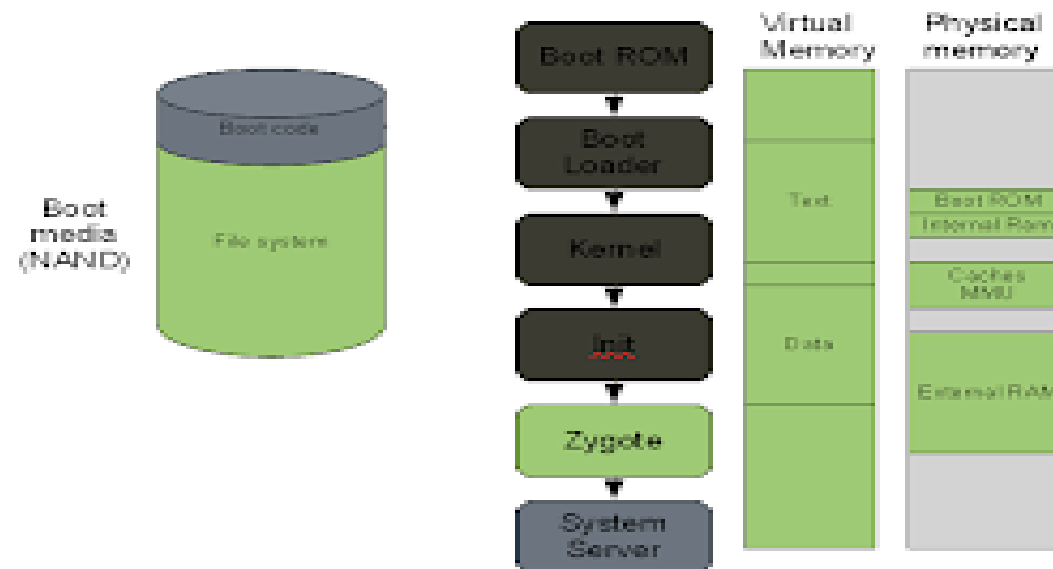
- L'init process dans Android execute un fichier appelé **init.rc**. script qui décrit les services du système, le fil system et les autres paramètres qui ont besoin d'être lancés. Le script init.rc est placé à la racine
- Le processus init execute le script d'initialisation et lancer les processus de service système.

INIT PROCESS



ZYGOTE

Le Zygote est lancé par le processus init et démarre l'exécution la machine virtuelle de Dalvik.



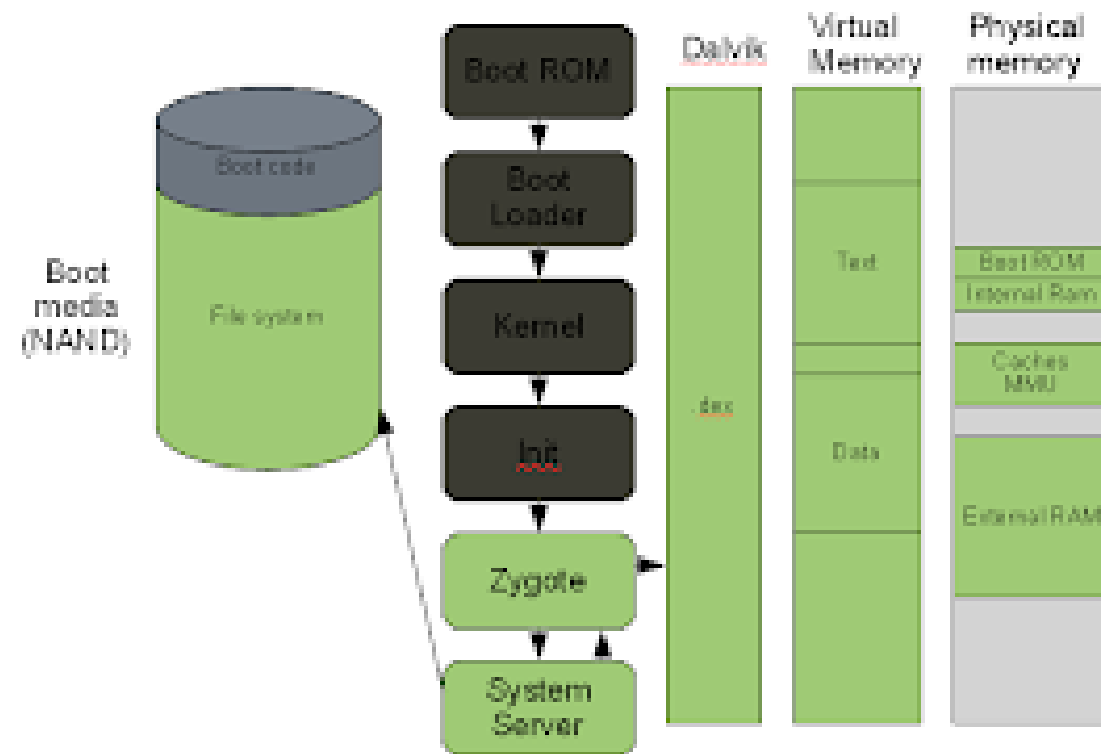
SYSTEM SERVER

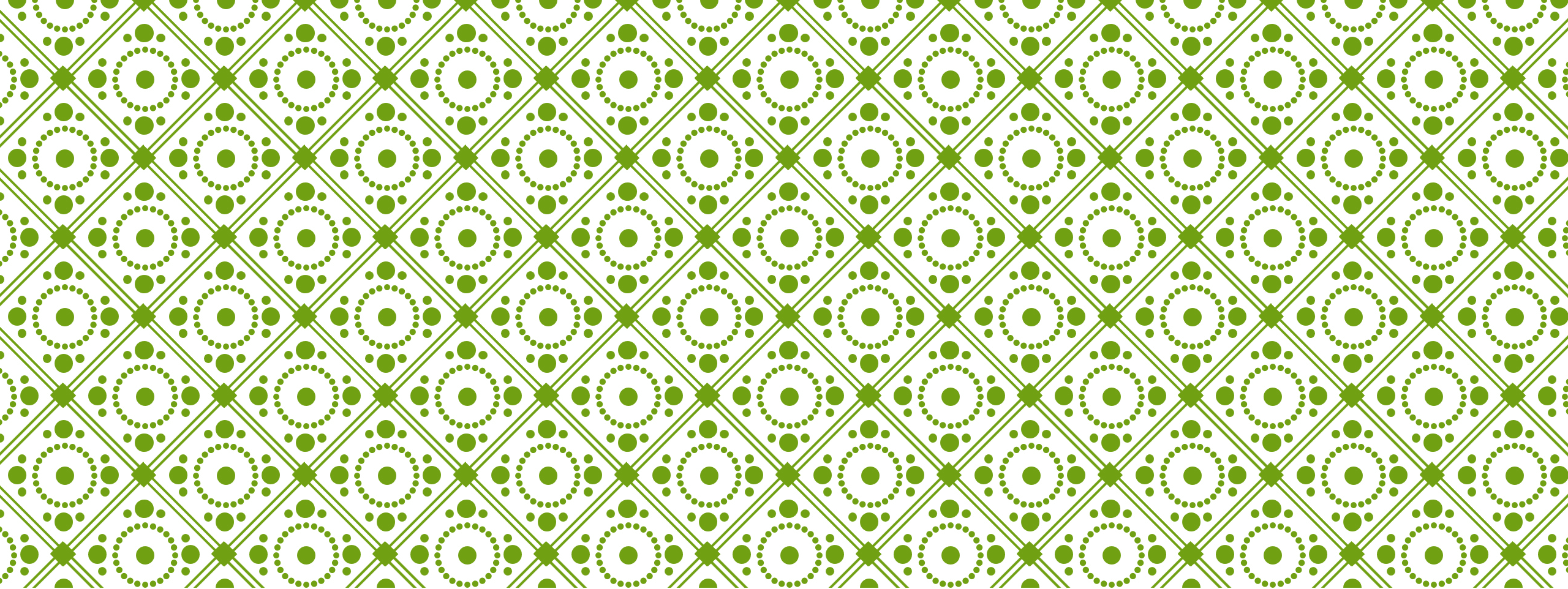
Le system server est le premier composant java qui va s'exécuter sur le système.

Il démarre tous les services Android comme le **telephony manager** et le **bluetooth**.

Le démarrage de chaque service est inscrit directement dans le run method du system server.

SYSTEM SERVER





APPLICATION ANDROID

APPLICATION

Application Android peut être composée des éléments suivants:

- Activités(**Activity**): c'est une partie de l'application présentant une interface à l'utilisateur;
- Services(**Service**):c'est une tâche de fond sans interface associée;
- Fournisseurs de contenus (**android.content.ContentProvider**) permettent le partage d'informations au sein ou entre applications;
- Récepteurs d'*Intents* (**android.content.BroadcastReceiver**): permet de déclarer être capable de répondre à des *Intents*;

ACTIVITÉ

Applications peuvent inclure plusieurs activités

Exemple :

Une application de messagerie électronique

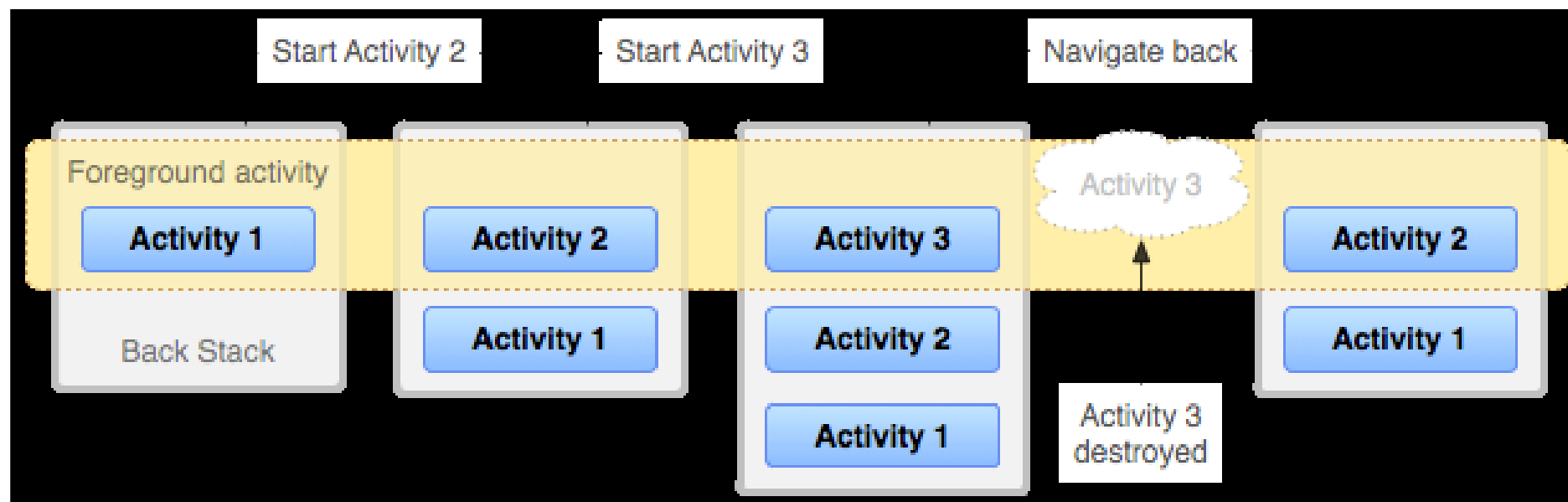
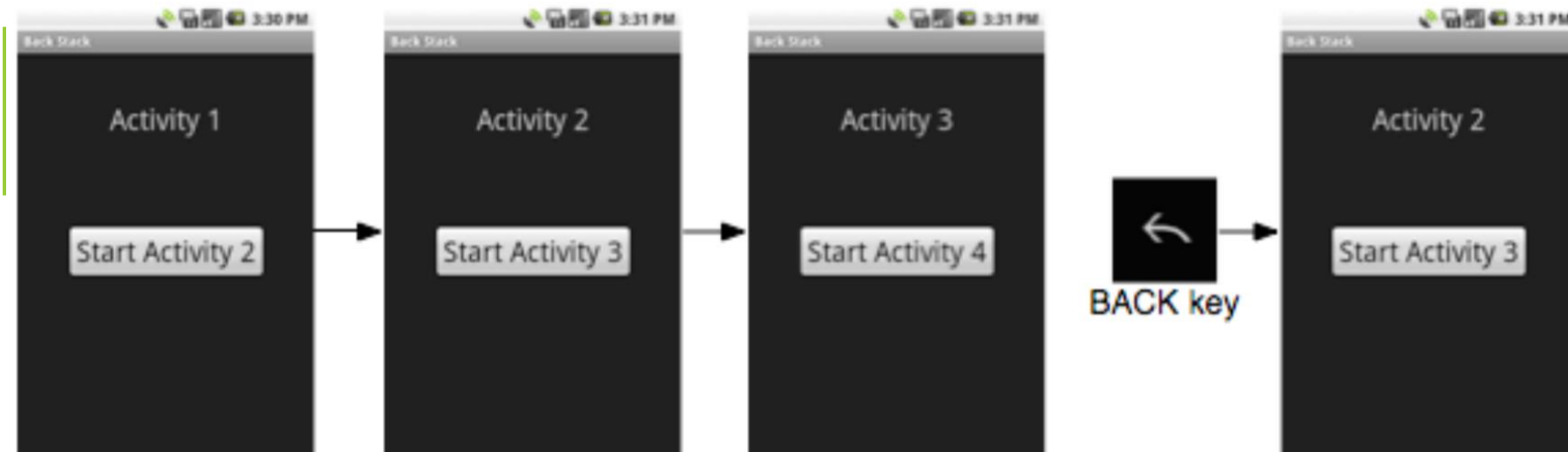
Une activité listant les emails reçus

Une activité pour rédiger un nouvel email

Une activité pour lire un email

TÂCHES

- Une tâche est un ensemble d'activité
- Activité pas nécessairement fournie par une seule application
- Android gère une pile de retour d'activité (Activity backstack)
- Lancer une activité -- Activité empiler dans la pile
- Presser le bouton de retour arrière --- Dépiler l'activité courante de la pile



TACHE AVANT PLAN / ARRIÈRE PLAN



ETATS D'ACTIVITÉ

➤ Non lancée – non encore créée

➤ Active

Resumed/Running – visible, has focus

Paused – visible, does not have focus, peut être terminée

Stopped – non visible, does not have focus, peut être terminée

➤ Terminée

ETATS D'ACTIVITÉ

Android communique les changements d'état à l'application en appelant des méthodes spécifiques (callbacks) du cycle de vie de l'activité

`protected void onCreate()`

`protected void onStart()`

`protected void onResume()`

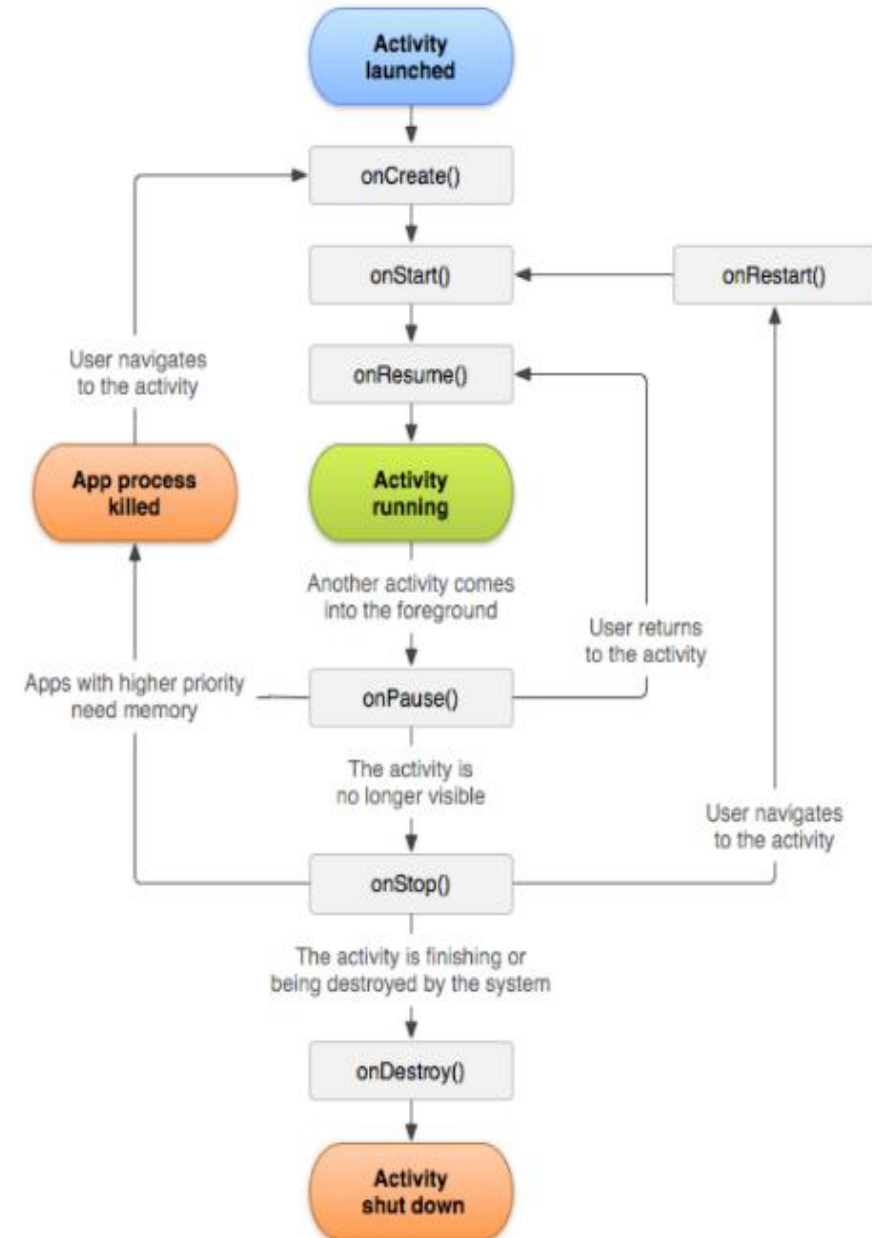
`protected void onPause()`

`protected void onRestart()`

`protected void onStop()`

`protected void onDestroy()`

ACTIVITÉ— CYCLE DE VIE



ETATS D'ACTIVITÉ

onCreate()

- Appelée quand l'activité est créée pour la première fois
- Initialise l'état global
- Appelle **super.onCreate()**
- Instancie et configure les vues de l'interface utilisateur
- Initialise de content view de l'activité

onRestart ()

- Appelée si l'activité a été arrêtée et elle est sur le point de redémarrer
- Actions possibles
- Lire l'état sauvegardé

ETATS D'ACTIVITÉ

onStart ()

- L'activité est sur le point de devenir visible
- Actions possibles
- Réinitialiser l'application

onPause ()

- Sur le point de basculer vers une autre application
- Actions possibles
- Arrêter les comportements foreground-only

ETATS D'ACTIVITÉ

`onStop()`

- L'activité n'est plus visible à l'utilisateur mais peut être redémarrer plus tard
- Sauvegarder l'état

`onDestroy()`

- L'activité est sur le point d'être détruite
- Sauvegarder l'état persistant

ETATS D'ACTIVITÉ

Les apps ont tendance à être tuées et redémarrées souvent

- Sauvegarde de l'état interne quand l'App est tuée
- Rechargement de l'état sauvegardé quand l'App est redémarrée

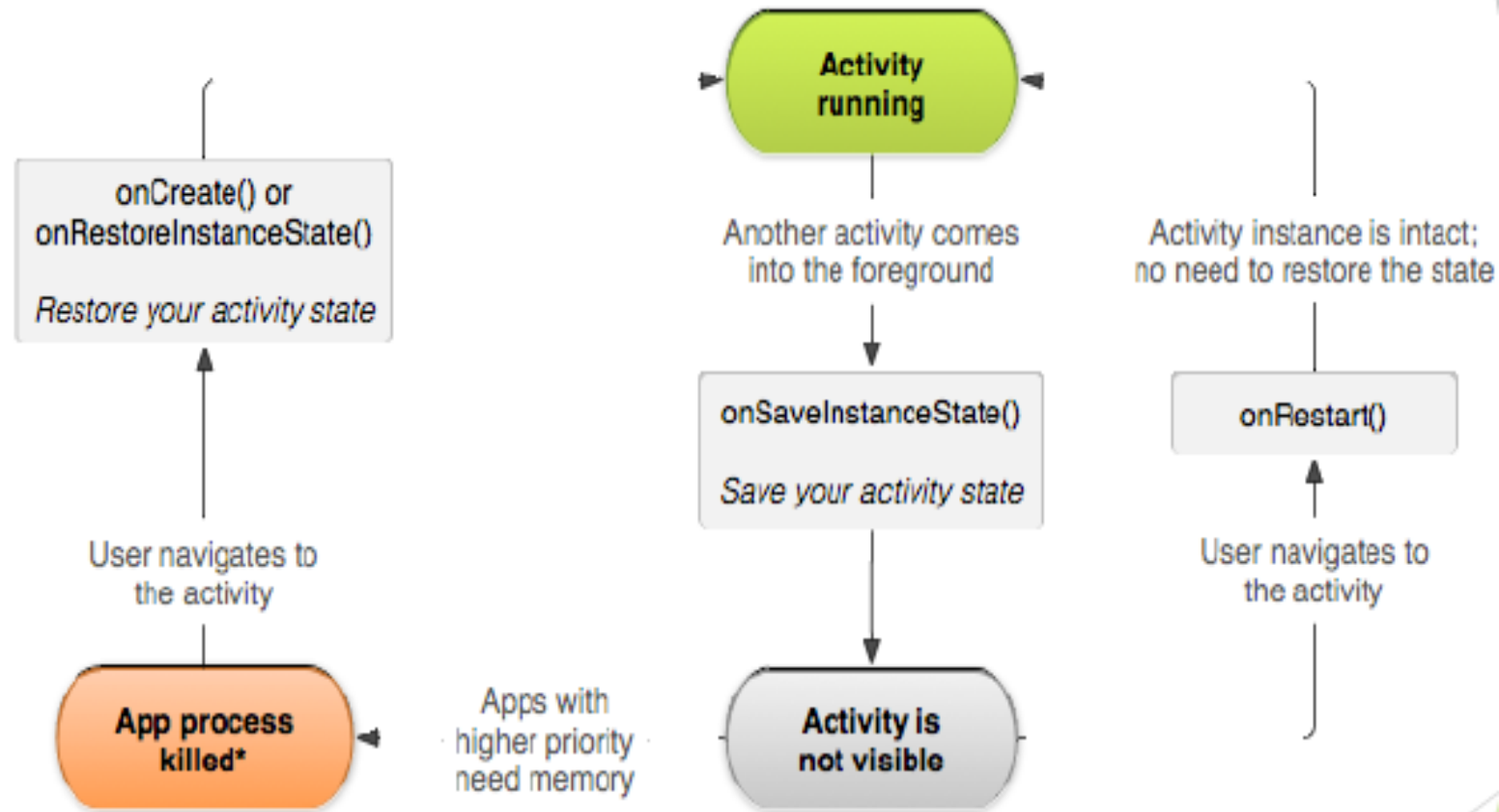
A travers des callbacks

- **onSaveInstanceState** ; avant de tuer l'App
- **onRestoreInstanceState** ; au redémarrage de l'App
- Appelés également lors de la rotation de l'appareil

Activité tuée et redémarrée pour charger les nouvelles ressources

- Non appelés en cas de terminaison volontaire de l'App par l'utilisateur

SAUVEGARDER L'ETAT D'ACTIVITÉ



*Activity instance is destroyed, but the state from `onSaveInstanceState()` is saved

SAUVEGARDER L'ETAT D'ACTIVITÉ

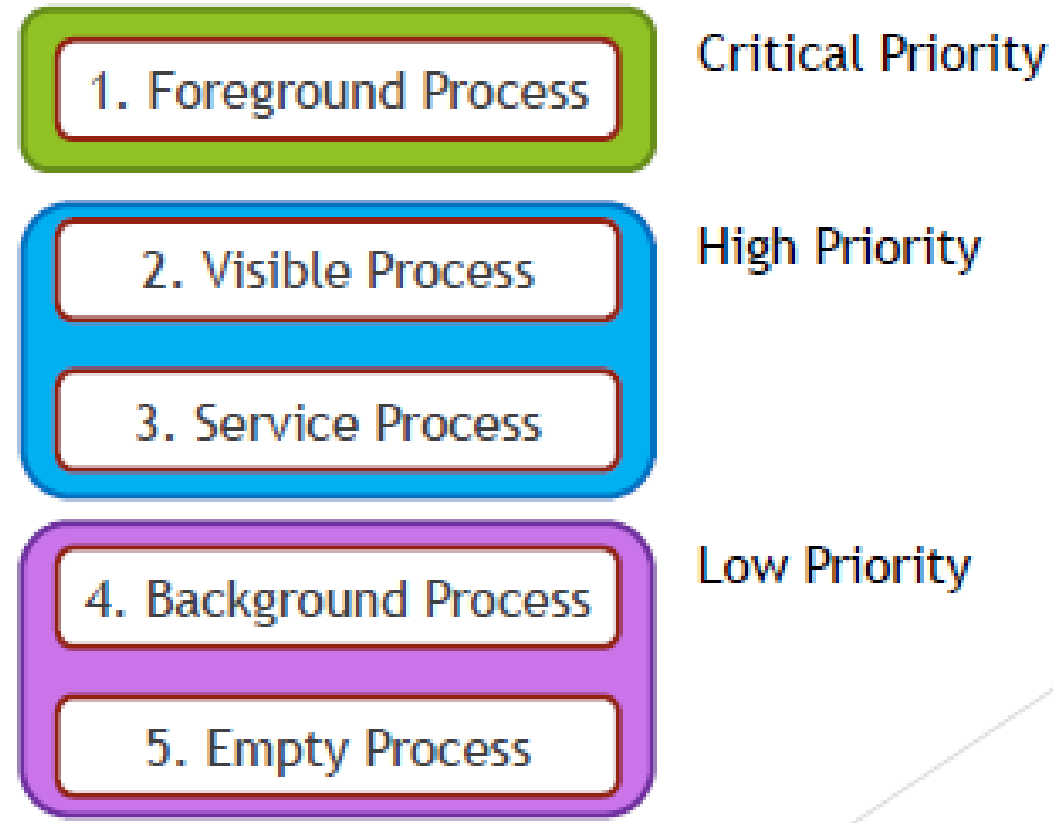
Hiérarchie de Priorité

Linux récupère la mémoire en supprimant les processus de faible priorité

Il y a cinq (05) niveaux de priorité

Android classe un processus dans le plus haut niveau qu'il peut lui affecter

SAUVEGARDER L'ETAT D'ACTIVITÉ



ACTIVITÉ

Processus en avant plan (Foreground process)

Un processus qui est nécessaire pour ce que l'utilisateur est entrain de faire

Processus visible (Visible process)

Un processus qui n'interagit pas avec l'utilisateur mais il peut affecter ce que l'utilisateur voit à l'écran

Processus service (Service process)

Un processus qui exécute un service

Lancé avec la méthode **startService()**

INTENT

- Un Intent est une sous classe de **android.content.Intent**,
- Le moyen de lier 2 composants
- Un événement(intent) est une "intention« à faire quelque chose à un composant Android.
- les Intents pour communiquer entre Activités, Services.
- Attributs d'un Intent

Nom - Action - Données (Data) - Catégorie (Category) – Type – Composant – Extras

```
<activity android:name=".AfficheURL" >  
    <intent-filter>  
        <action android:name="android.intent.action.VIEW" />  
        <category android:name="android.intent.category.DEFAULT" />  
        <data android:scheme="http" />  
    </intent-filter>  
</activity>
```


INTENT

Intent explicite

- permet de lancer une activité particulière

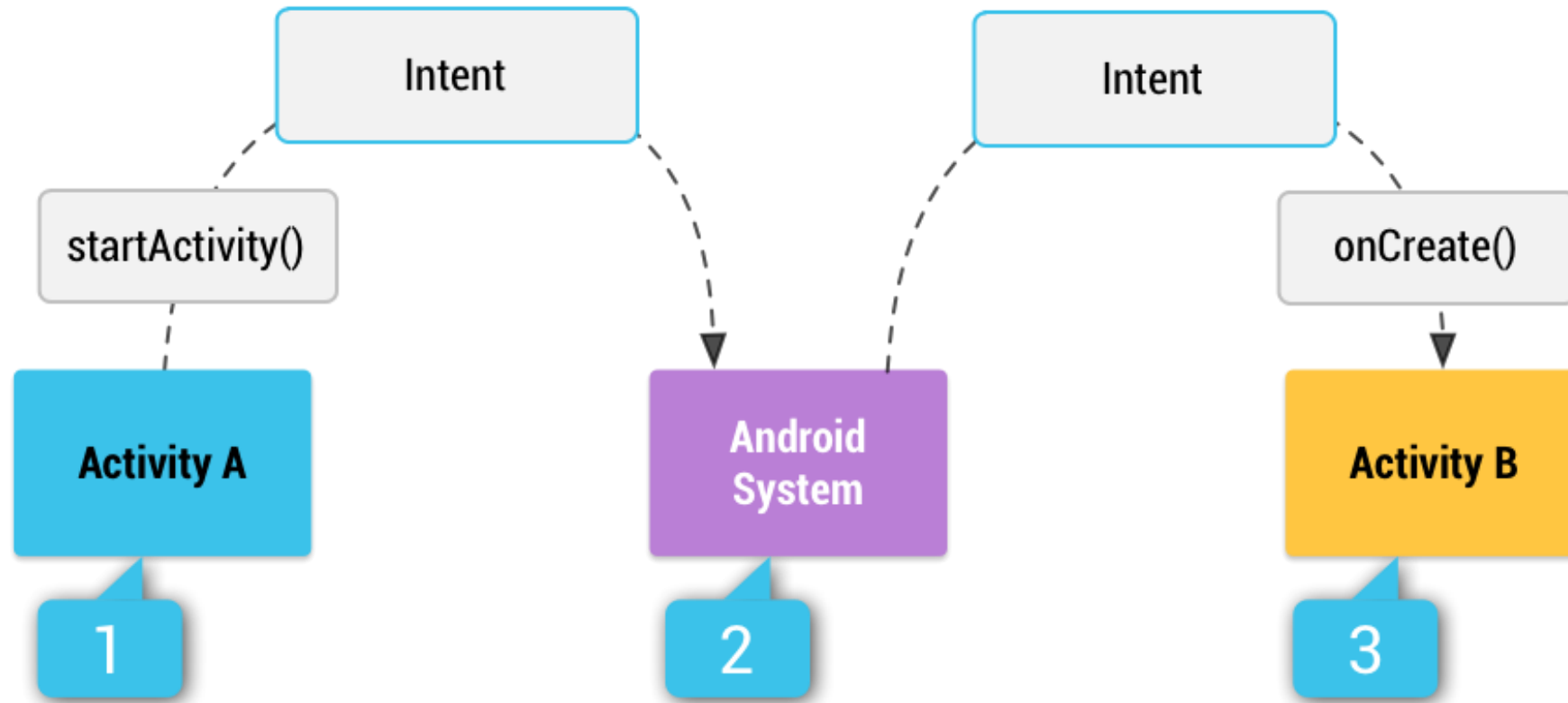
Intent implicite

- Permet de demander au système de réaliser une action particulière sans viser une Activité spécifique
- Lors de l'utilisation, Android cherche parmi les activités qui se sont enregistrées comme capable de gérer cette demande (*manifest*)
- Si plusieurs activités sont trouvées, il est automatiquement demandé à l'utilisateur de choisir ("ouvrir avec")

Exemple : - affichage d'une page web

```
Uri webpage = Uri.parse("http://www.android.com");  
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);  
startActivity(webIntent);
```

INTENT



SERVICE

Une tâche qui s'exécute sans interface

peut prendre une longue période d'exécution

Une activité peut lancer un service ou bien communiquer (Binder IPC) avec un service existant,

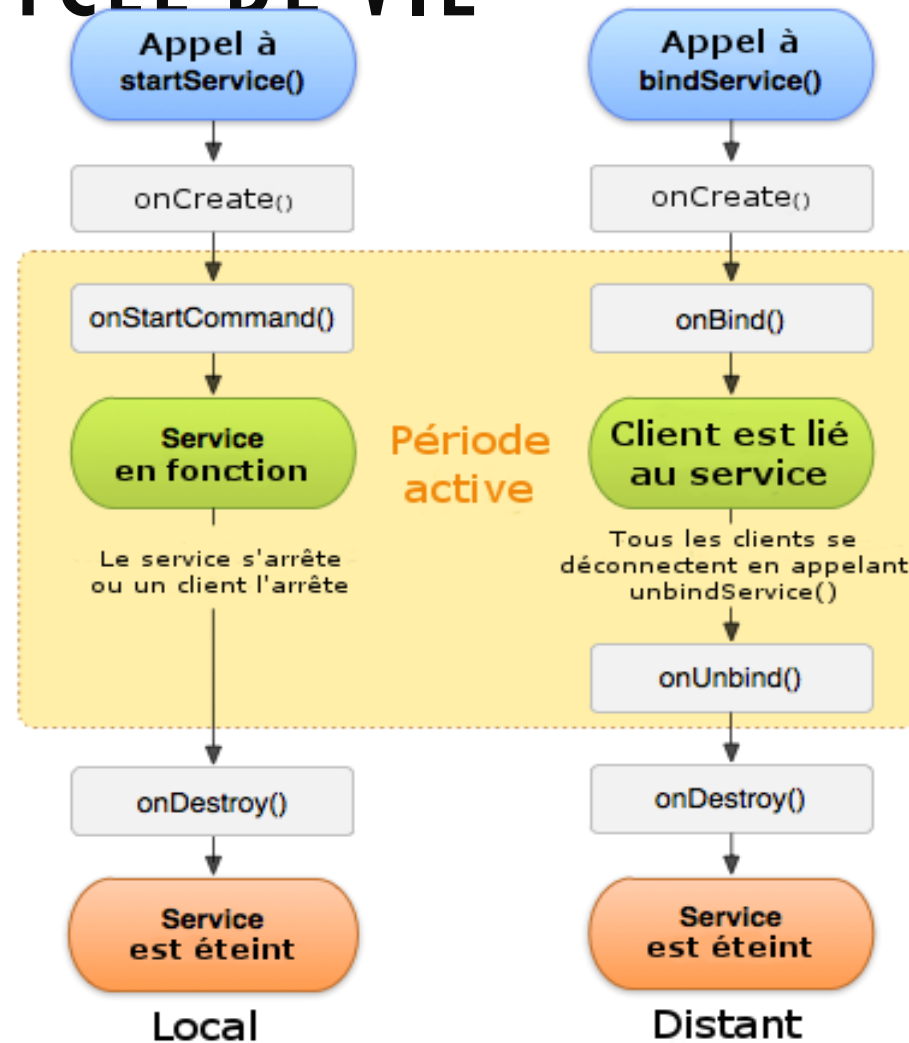
```
Intent intent = new Intent(Activite.this, UnService.class);  
startService(intent);
```

en avant plan ou en arrière plan

Exemple : lecteur de musique

```
<service android:name="MusicService"  
    android:process=":player" >  
    ...  
</service>
```

SERVICE — CYCLE DE VIE



BROADCASTRECEIVERS

- Est une class pour recevoir les Intents et appliquer une réaction
- Ne possède pas d'interface utilisateur
- L'intent est diffusé dans le système
- L'intent est reçu et traité via la méthode **OnReceive()**

Exemple : réception d'un sms

BROADCASTRECEIVERS

BroadcastReceivers s'enregistrent pour recevoir des Intents spécifiques

- enregistrement statique : *AndroidManifest.XML*
- enregistrement dynamique : *Context.registerReceiver()* /

LocalBroadcastManager.registerReceiver()

Quand un composant diffuse un Intent, Android identifie les récepteurs appropriés et délivre l'évènement En appelant ***BroadcastReceiver.onReceive()*** qui traite l'évènement

BROADCASTRECEIVERS

Méthodes de diffusion d'événement sont avec ou sans permission du récepteur

Normale : ordre de traitement non défini

void sendBroadcast(Intent intent)/ void sendBroadcast(Intent intent, String receiverPermission)

Ordonnée : traitement séquentiel dans l'ordre de priorité

void sendOrderedBroadcast(Intent intent, String receiverPermission)

BROADCASTRECEIVERS

OnReceive()

- Est de courte durée
- Si le traitement est lent un service est lancé
- Ne permet pas de lancer une activité, un dialogue, un service
- Le processus a une priorité élevée durant l'exécution de *onReceive()*

CONTENT PROVIDER

Permet le partage de données entre applications :

- Navigateur – bookmarks, historique
- Journal des appels – utilisation du téléphone
- Contacts – données des contacts
- Media – base de données des media
- UserDictionary – base de données pour l'orthographe prédictive

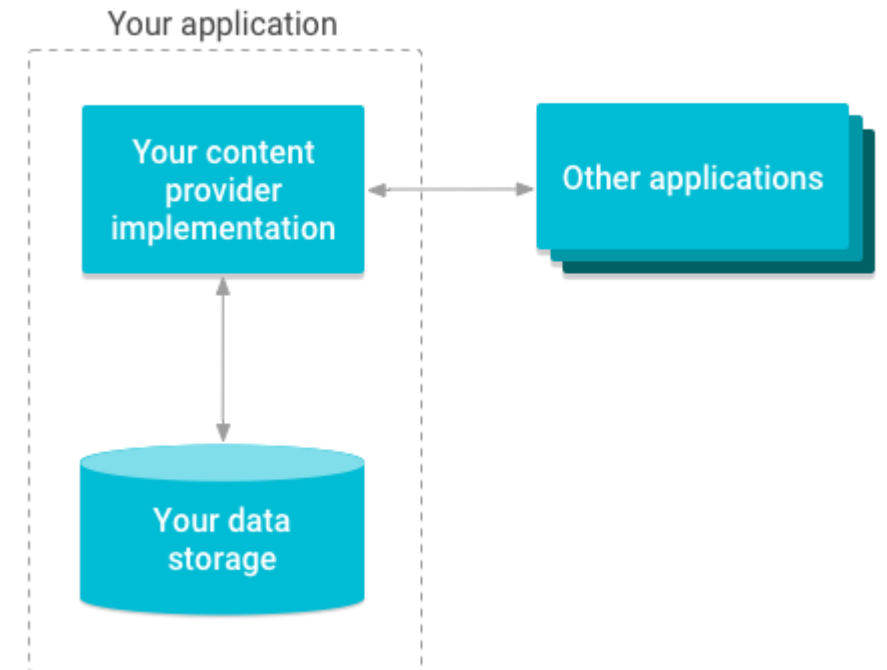
Stocké sous forme de tables de bases de données

ContentProvider identifie les ensembles de données à travers les URIs

Uri pour chercher la base de données des contacts

Exemple :

```
ContactsContract.Contacts.CONTENT_URI =  
"content://com.android.contacts/contacts/"
```



CONTENT PROVIDER

ContentResolver

- Gère les content provider dans les différents application

Exemple : notification de changement

Utiliser ***Context.getContentResolver()*** pour accéder

CONTENT PROVIDER

Pour implémenter un système de stockage pour les données il est nécessaire de :

- Classe ***SQLiteOpenHelper***
- Implémenter une sous-classe **ContentProvider**
- Surcharger les méthodes : **query()**, **insert()**, **update()**, **delete()**, **getType()**, **onCreate()**
- Déclarer le fournisseur de contenu dans le manifest

CONTENT PROVIDER

Utiliser **ContentResolver.query()** pour extraire les données

- Retourne une instance **Cursor** pour consulter les résultats
- Un Cursor est un itérateur un ensemble de résultats Cursor query(

Uri uri, //Uri de ContentProvider

String[] projection, //Colonnes à extraire

String selection, //Motif de sélection SQL

String[] selectionArgs, //Args du motif SQL

String sortOrder //Ordre de tri

)

CONTENT PROVIDER

Le *cursor* fournit l'accès aux résultats de la requête

- boolean moveToFirst()
- boolean moveToNext()
- int getColumnIndex(String columnName)
- String getString(int columnIndex)

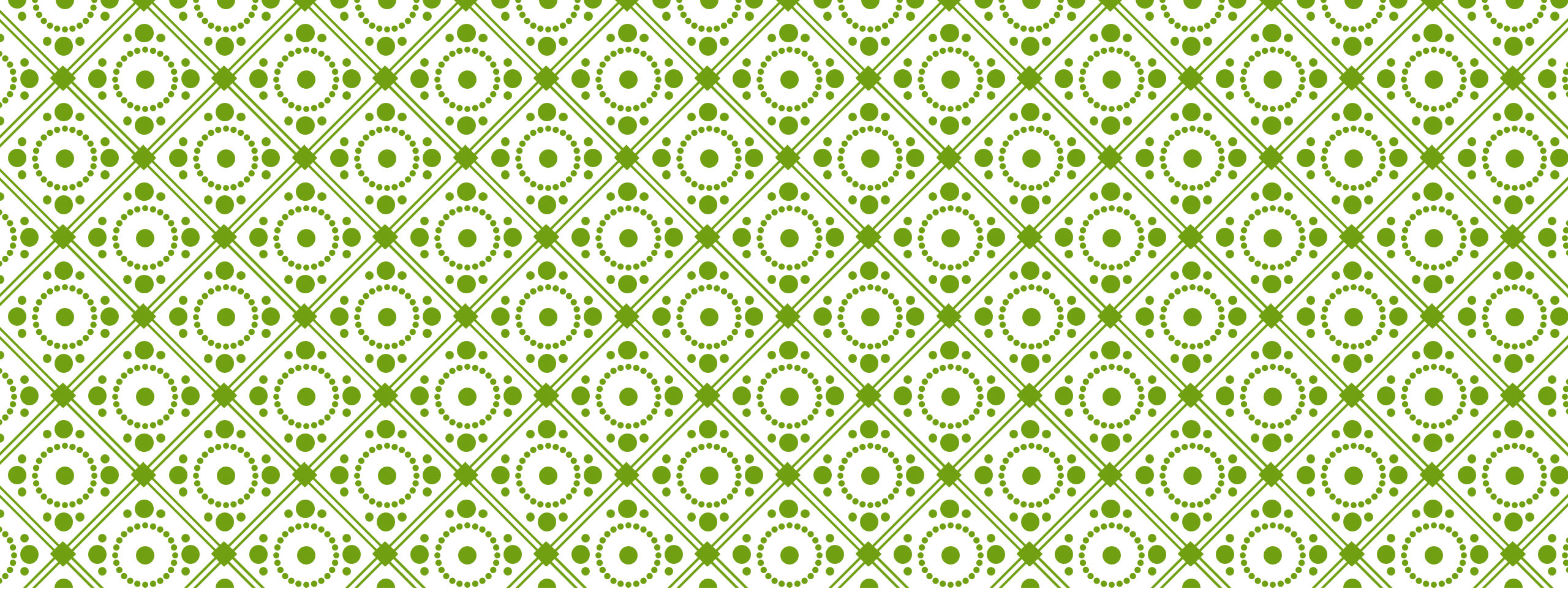
suppression

- Utiliser ***ContentResolver.delete()*** pour supprimer les données

```
public final int delete( Uri url, //Uri du contenu
                        String where, //Motif de sélection SQL
                        String[] selectionArgs //Args du motif SQL )
```

Insertion

```
private void insertContact(String name){ ArrayList<ContentProviderOperation> ops =
    new ArrayList<ContentProviderOperation>();
```



PROCESSUS ET THREAD

RAPPEL UNIX

Processus :

- Par défaut tout composant d'une même application s'exécute dans le même processus, un processus est lancé par **Zygote** identifié par un **PID, PPID, UID, Pile, Heap**.
- il reste possible de modifier le fichier **Manifest.xml** pour une autre spécification dans l'attribut **android:process**.
- un processus est exécuté sur sa VM, chaque app est isolée des autres, sauf si elles partagent le même ID ou le même certificat.
- le système Android peut mettre fin à l'exécution d'un processus si manque de ressource, le processus sacrifié est choisi suivant un ordre de priorité.

Thread :

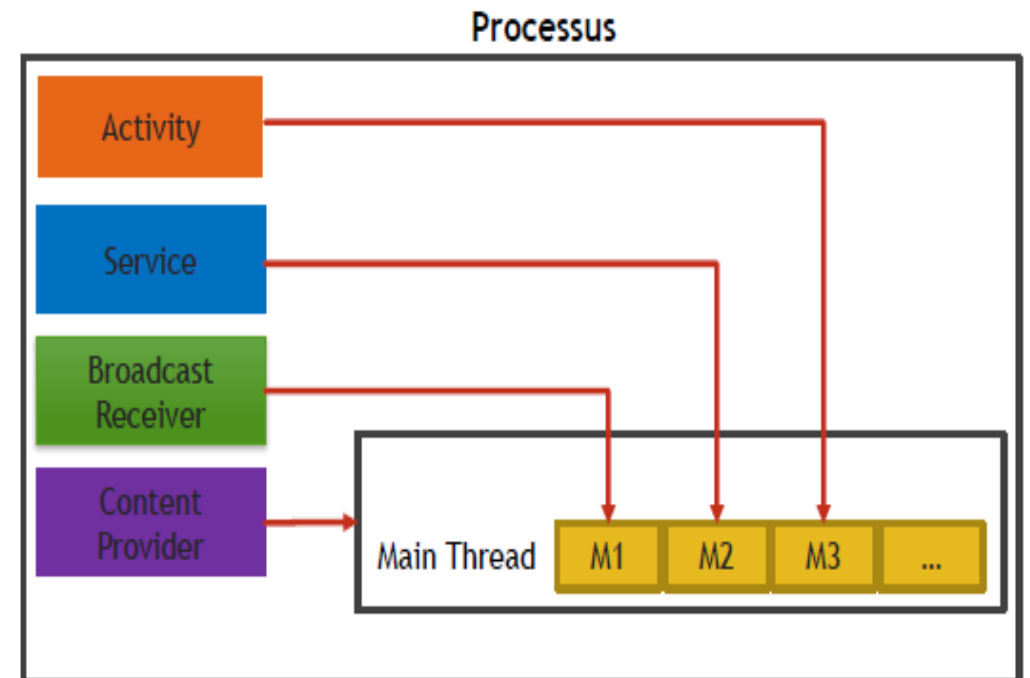
- un processus peut intégrer un ou plusieurs thread, le processus est parent au thread
- définit dans le standard POSIX

MÉCANISME DE THREADING ANDROID

Au lancement de l'application un main thread est lancé, l'activité principale et toutes les activités d'interface s'exécutent dans le main thread souvent appelé ***IU thread***.

Sur le main thread ***IU thread*** s'exécutent

- Les méthodes liées aux événements
- Les méthodes du cycle de vie du service
- Les événements déclenchant un BroadcastReceiver



MÉCANISME DE THREADING ANDROID

Contraintes : temps de réponse (<5secs)

- Le main thread ne doit pas être bloqué
- Eviter d'exécuter des opérations réseaux, de gestion bases données ou toutes opération couteuse dans le main thread

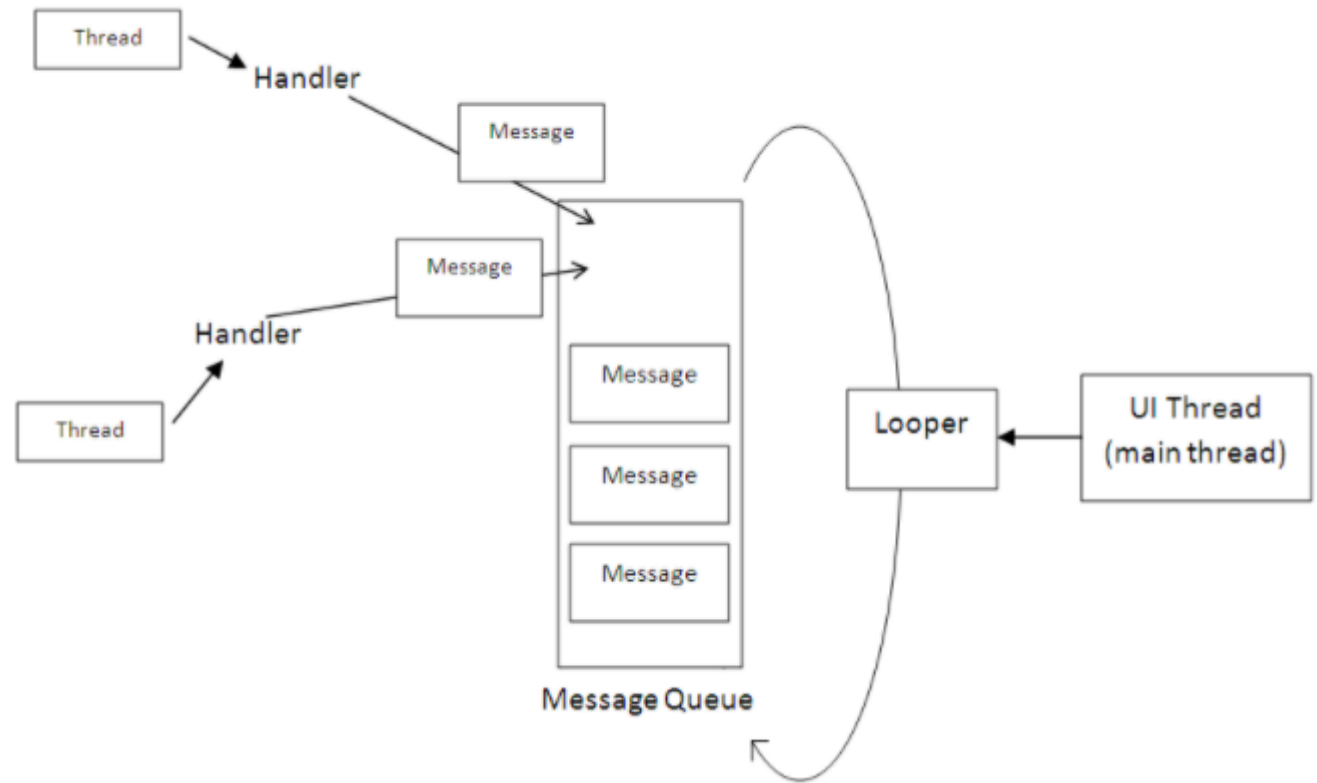
Solution : utilisation des **Worker thread**

Il existe 2 types de threads:

- Thread rattaché à une activité : termine son exécution dès que l'activité est terminée.
- Thread non rattaché à une activité: continue de s'exécuter après que l'activité soit terminée.

MÉCANISME DE THREADING ANDROID

- Pool threads
- Queue de messages (évènement)



MÉCANISME DE THREADING ANDROID

Le UI reste manipulable uniquement par le main thread

Solutions

1. `Activity.runOnUiThread (Runnable)`
2. `View.post(Runnable)`
3. `View.postDelayed (Runnable, long)`

MÉCANISME DE THREADING ANDROID

`Activity.runOnUiThread (Runnable)`

- hérite de la classe `Activity`

spécifie qu'une action doit s'exécuter dans le thread UI. Si le thread actuel est le thread UI, alors l'action est exécutée immédiatement.

Sinon, l'action est ajoutée à la pile des événements du thread UI.

MÉCANISME DE THREADING ANDROID

View.post(Runnable)

Classe View

pour ajouter le **Runnable** à la pile des messages du thread UI. Le retourné vaut `true` s'il a été correctement placé dans la pile des messages.

Runnable est une commande à exécuter

Run () l'exécution du code active

Start () lance l'exécution du thread, la JVM appelle la méthode `run` du thread

MÉCANISME DE THREADING ANDROID

View.postDelayed (Runnable, long)

permet d'ajouter un ***Runnable*** à la pile des messages, mais uniquement après qu'une certaine durée soit écoulée.

EXAMPLE THREAD SAFE

```
public void onClick(View v) {  
    new Thread(new Runnable() {  
        public void run() {  
            // a potentially time consuming task  
            final Bitmap bitmap = processBitMap("image.png");  
            imageView.post(new Runnable() {  
                public void run() {  
                    imageView.setImageBitmap(bitmap);  
                }  
            });  
        }  
    }).start();  
}
```

THREAD SAFE

Thread safe

- cas des méthodes appelées par plusieurs threads doivent être codé Thread Safe

EXE: les méthodes *IBinder*

Handler

- niveau supérieur au thread (encapsulation)
- est attaché à un thread
- tâche à long exécution, les opérations à exécuter sont plus complexes,
- file de messages, les messages sont reçus dans la méthode *callback*

ASYNCK TASK

Asynck task

permet d'effectuer des opérations d'arrière-plan et de publier les résultats dans le thread UI sans avoir à manipuler de threads ou de handlers.

- Niveau d'abstraction supérieur au thread et handler
- Tache de courte durée, le résultat est publié au IUIThread
- héritage de la classe avec les paramètres :
 - **Params** permet de définir le typage des objets sur lesquels on va faire une opération.
 - **Progress** indique le typage des objets qui indiqueront l'avancement de l'opération.
 - **Result** est utilisé pour symboliser le résultat de l'opération.

ASYNCK TASK

Utilise les méthodes :

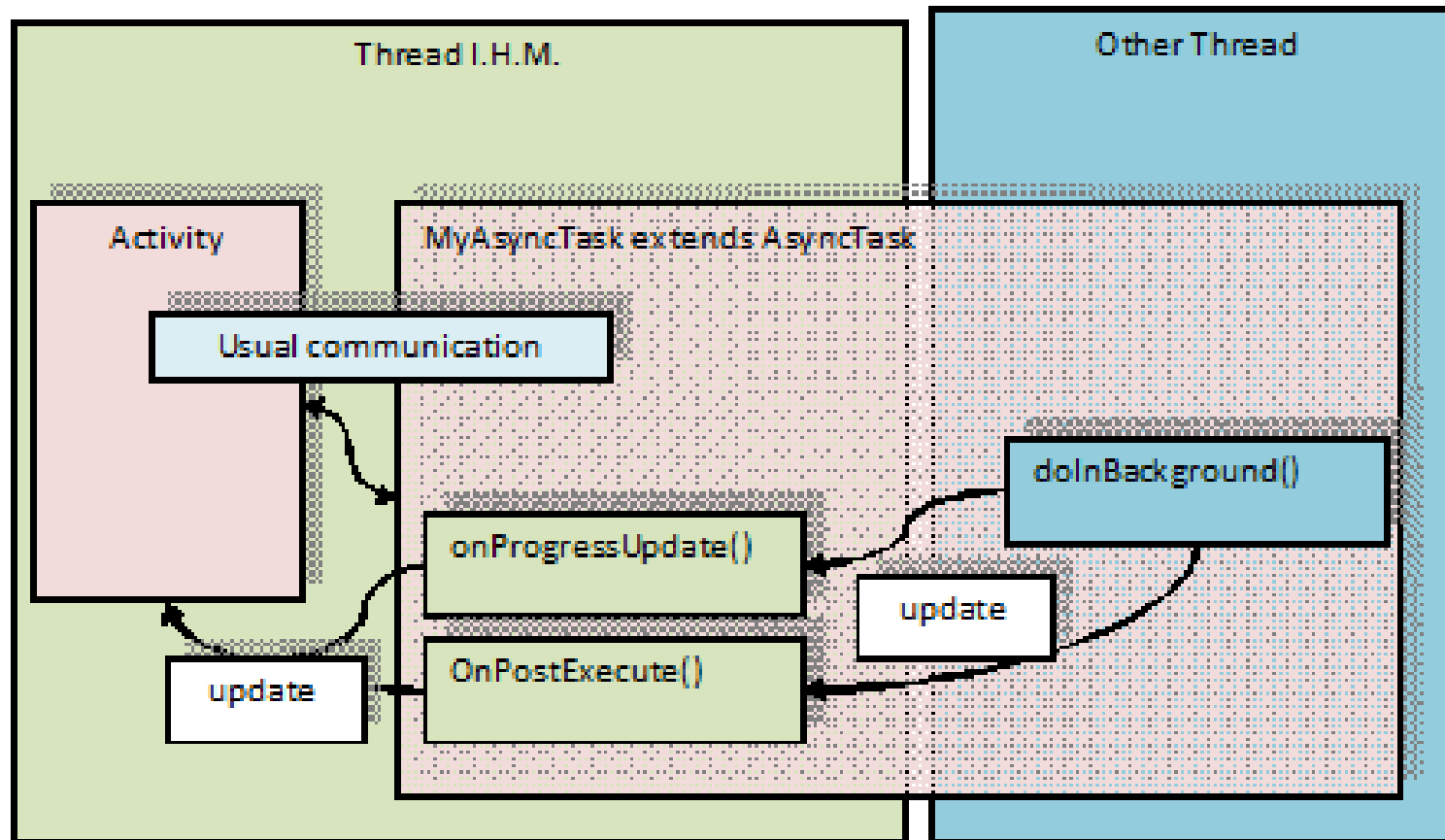
OnPreExecute : appelé sur le UIThread avant que la tâche soit exécutée.

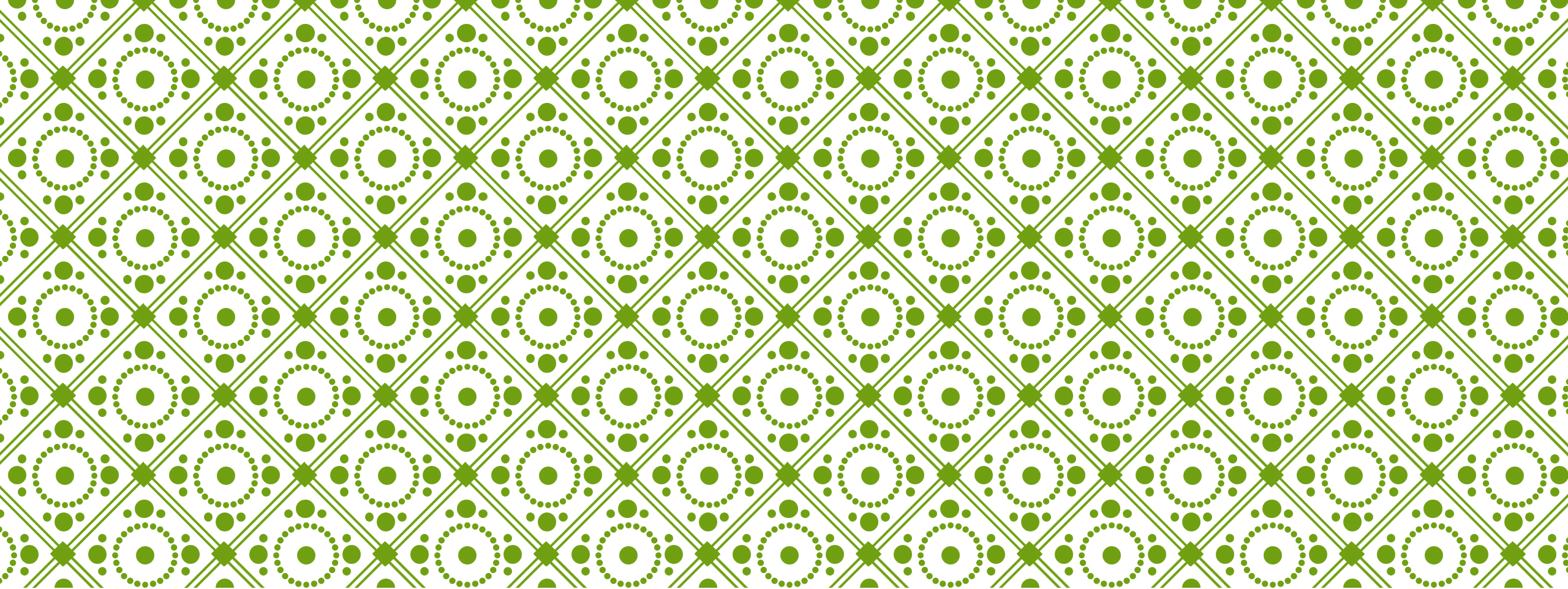
doInBackground : appelé sur le thread en arrière-plan, permet de faire des tache en arrière plan

onProgressUpdate : appelé sur le UIThread, pour publier les mise a jours de progression,

onPostExecute : appelé sur le UIThread après la fin du calcul, permet de transmettre le resultat

ASYNCK TASK





GESTION DE PROCESSUS

CFS COMPLETELY FAIRE SCHEDULING

- L'exécution des threads est géré par le Scheduleur du noyau Linux
- L'unité fondamentale est un thread et non pas un processus
- Le Scheduleur désigne le thread à exécuter et le temps d'exécution alloué,
- CSF est un algorithme de Scheduling
 - maximise le taux d'utilisation de l'UC ainsi que l'interactivité
 - Pour les processus de même priorité normale SCHED_NORMAL,
 - ne préempte pas les processus batch SCHED_BATCH
 - mise en veille SCHED_IDLE

CFS COMPLETELY FAIRE SCHEDULING

Temps partagé

- quantum

- préemptive

- priorité (nice value)

N'utilise pas une file d'attente mais un arbre bicolore

CFS COMPLETELY FAIRE SCHEDULING

Le nice :

- entre -20 et 19 (la valeur la plus faible désigne une plus haute priorité)
- induit le temps de calcul alloué a chaque processus

Le poids :

- pré calculé: **$\text{Weight}(\text{nice}) \approx 1024/1.25^{(\text{nice})}$**

CFS COMPLETELY FAIRE SCHEDULING

Le Quantum

$$\frac{TL * W_i}{\sum_{j=1}^n W_j}$$

TL : Latence ciblée (période de temps ou les runnable sont exécuté au moins une fois)

Wi : le poids de la tâche i

n : le nombre de tâches dans l'état runnable

CFS COMPLETELY FAIRE SCHEDULING

Le Vruntime

- calculé au scheduler tick
- Virtual run time ne peut être égale au réel run time
- le processus dont le Vruntime est le plus petit est élu à l'exécution
- le vruntime est incrémenté par : $\frac{t * W_0}{W_i}$ sachant que :

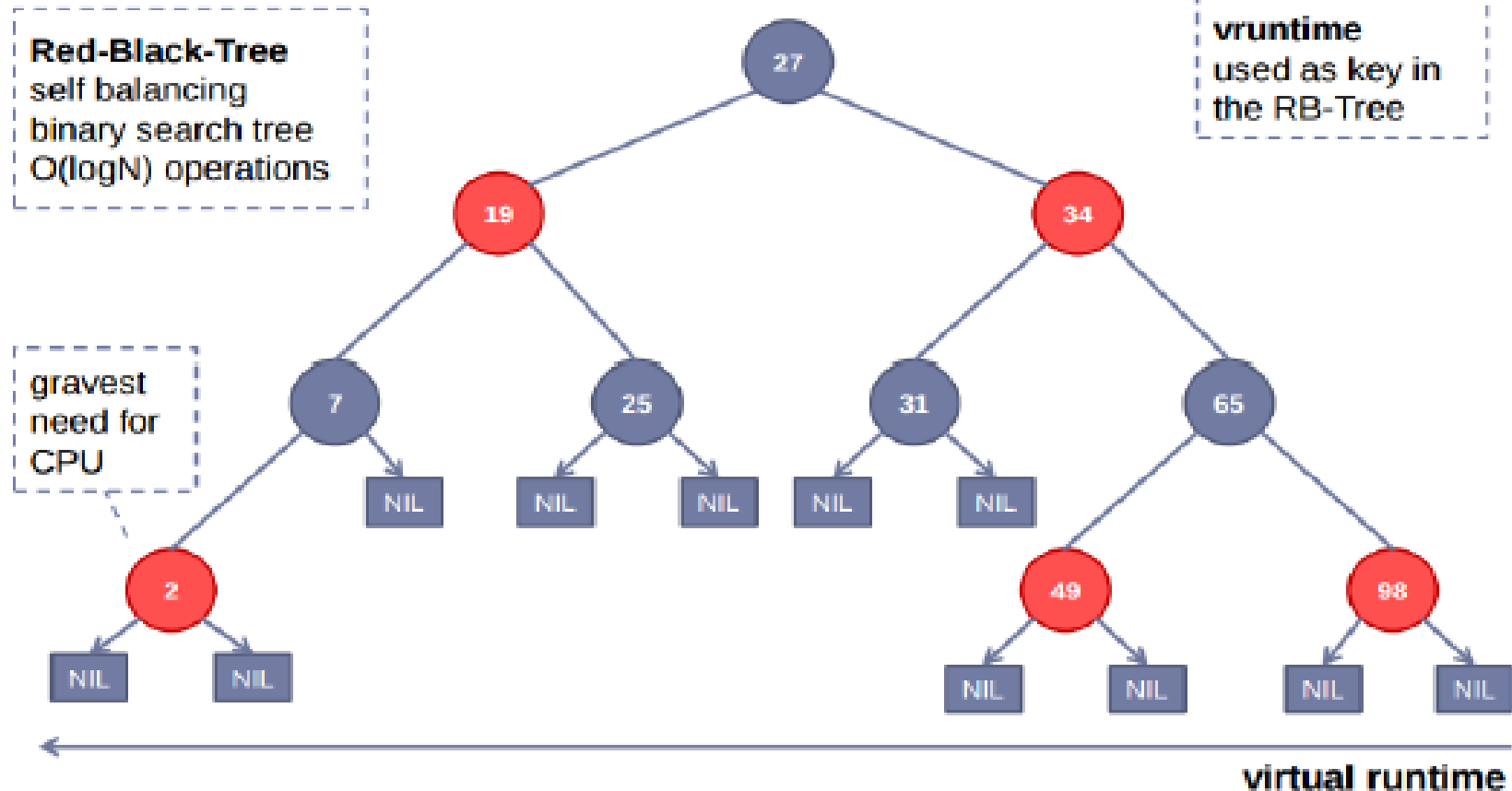
t: temps d'exécution dans l'UC

Wo: le poids avec nice 0

Wi: le poids avec nice i

CFS COMPLETELY FAIRE SCHEDULING

L'arbre bicolore

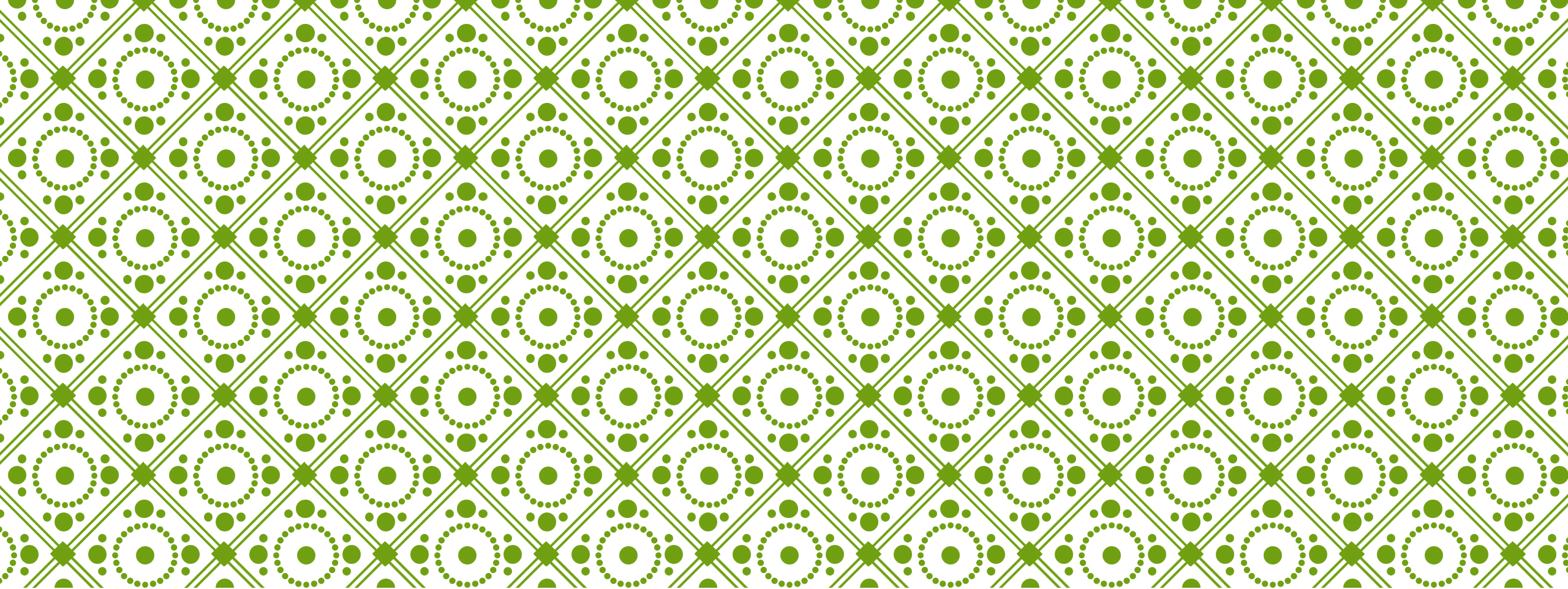


CFS COMPLETELY FAIRE SCHEDULING

Le vruntime est utilisé comme clé

Le sous arbre droit contient les clé plus grande que la valeur du nœud

Le sous arbre gauche contient les clés plus petite que la valeur du nœud



PERMISSION

- 
- Les permissions servent à protéger la confidentialité d'un utilisateur d'un appareil Android.

Exemple :

- lecture ou l'écriture des données privées de l'utilisateur (telles que les contacts ou les courriels),
 - la lecture ou l'écriture des fichiers d'une autre application, la création d'un accès réseau, le maintien de l'appareil en veille, etc.
- Si l'action nécessite l'abrobation de l'utilisateur elle est envoyée a ce dernier

APPROBATION DE LA PERMISSION

- Une application doit publier les autorisations requises en incluant des bases dans le manifeste de l'application.

Si une application doit envoyer des messages SMS, elle doit inclure cette permission le fichier manifeste

```
<permission android:name="android.permission.SEND_SMS"
```

```
...
```

```
/>
```

LA PERMISSION

➤ Composants

Les composants individuels peuvent définir leurs propres permissions

- annulent les permissions niveau application

➤ Activité

Spécifie si le composants peut lancer l'activité associée lors de :

- **startActivity()** - **startActivityForResult()**

Génère **SecurityException** en cas d'échec

➤ ContentProviders

Spécifie si le composant peut lire et écrire les données dans un ContentProvider

LA PERMISSION

➤ Service

Spécifie si le composant peut lancer ou se lier au service associé lors :

- **Context.startService()** - **Context.stopService()** - **Context.bindService()**

Génère **SecurityException** en cas d'échec

➤ Broadcast receiver

Spécifie si le composant peut envoyer des broadcasts au BroadcastReceiver lors :

- **Context.sendBroadcast()**

Ne génère pas de **SecurityException** en cas d'échec

MÉTHODES

Est-ce qu'un processus appelant peut accéder à un service ?

Context.checkCallingPermission()

Est-ce qu'un processus donné a des permissions spécifiques ?

Context.checkPermission(String permission, int pid, int uid)

Est-ce qu'une application donnée a des permissions spécifiques ?

PackageManager.checkPermission(String permName, String pkgName)

NIVEAUX DES PERMISSIONS

Permission normale

représente un risque très faible pour la confidentialité de l'utilisateur ou le fonctionnement d'autres applications.

- attribuée automatiquement par le système

Exp: autorisation d'accès à internet

Permission signature

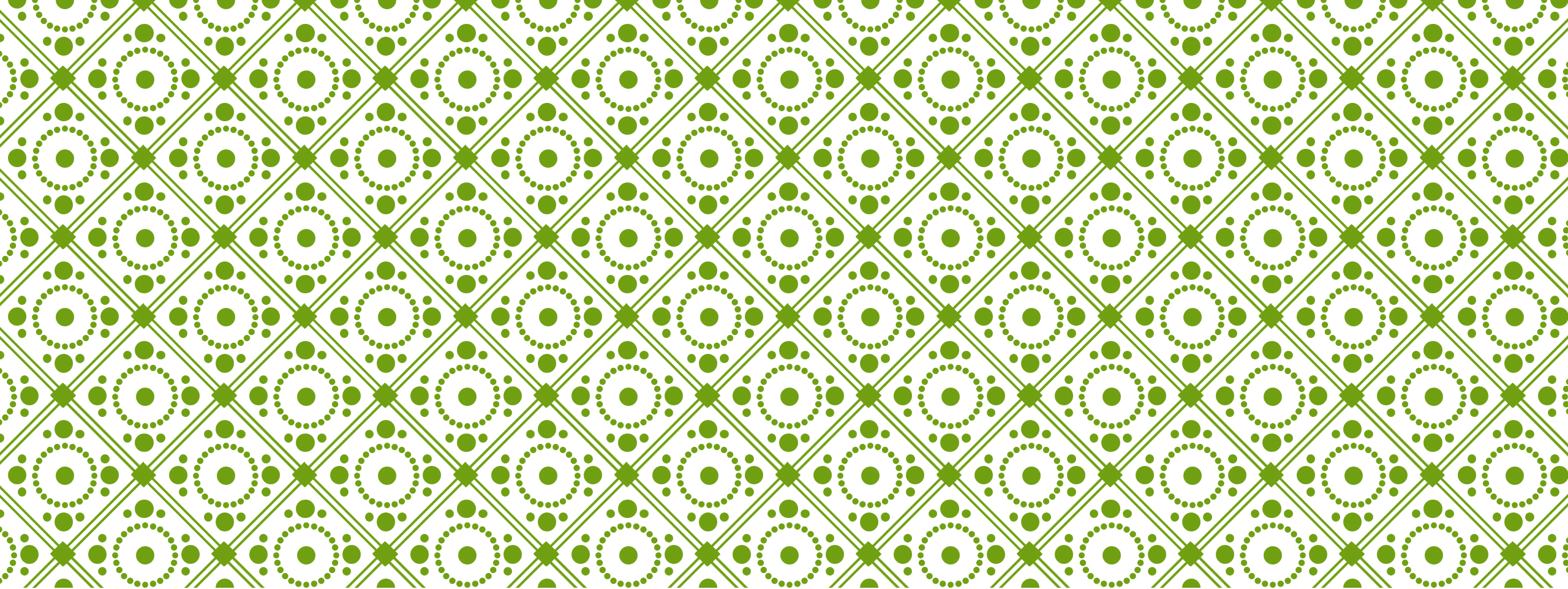
Le système accorde ces permissions à l'application au moment de l'installation, mais uniquement lorsque l'application qui tente de les utiliser est signée par le même certificat que l'application qui définit ces permissions.

NIVEAUX DES PERMISSIONS

Permission dangereuse

pour obtenir des données ou des ressources impliquant les informations privées de l'utilisateur ou susceptibles d'affecter les données stockées de l'utilisateur ou le fonctionnement d'autres applications.

Exemple, la possibilité de lire les contacts de l'utilisateur.



GESTION DE LA MÉMOIRE

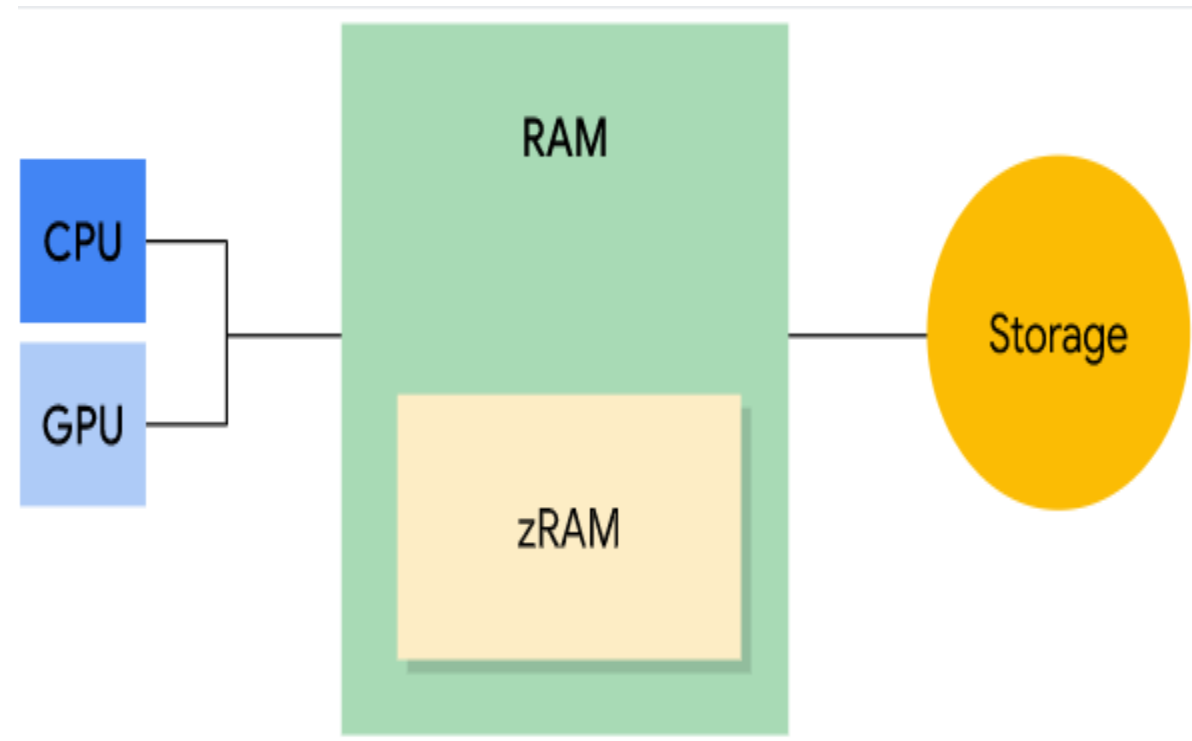
RAM : mémoire vive

zRAM: partition de RAM pour

Swaping, données compressé

Storage : données permanent

Contacts, bibliothèques



RAM

- Structurée en page mémoire
- La taille d'une page est de **4KB**
- Une page est soit libre soit utilisée

RAM

➤ **cached** : exemple : du code d'application, fichier

- **privé**: appartient à un seule processus
 - **original**: copie de l'element stocké – peut etre libérée par *kswapd*
 - **modifié**: copie modifié de l'element stocké – peut etre zipée en zRam par *kswapd*
- **partagé** : appartient a plus d'un processus
 - **original**: copie de l'element stocké – peut etre libéré par *kswapd*
 - **modifié**: copie modifié de l'element stocké – peut enregistré en storage par *kswap* ou explicitement par *msync()* ou *munmap()*

➤ **anonyme** : n'est pas associé à un fichier enregistré allouée par *mmap()/MAP_ANONYMOUS* flag

- **modifié**: peut etre compressée en zRam par *kswapd*

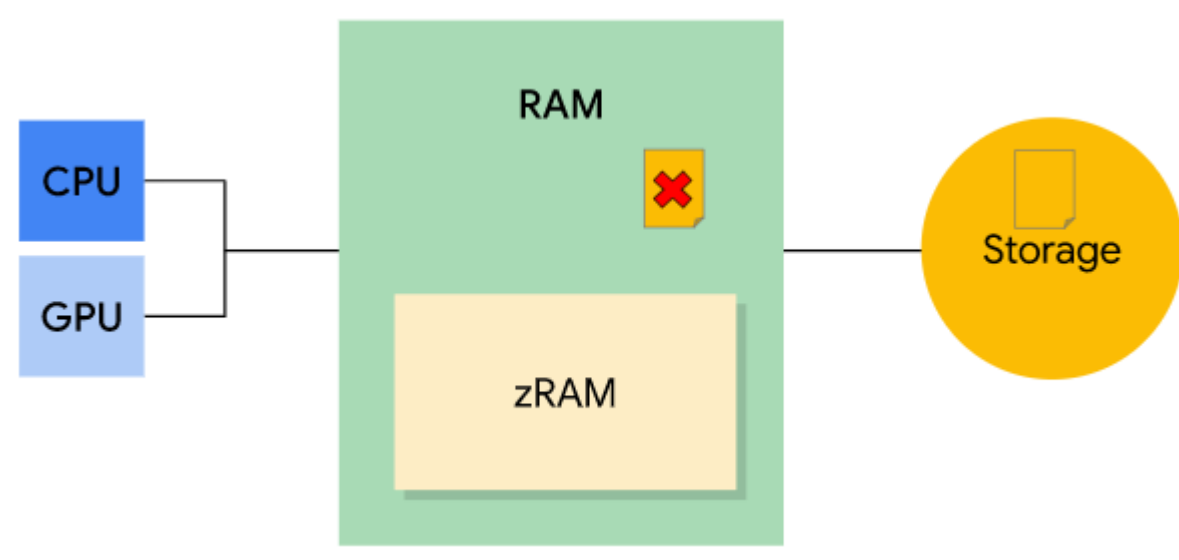
DÉFAUT DE PAGE

Kswapd : kernel swap daemon

- un processus du noyau linux
- devient actif dans le cas de mémoire Libre passe en dessous d'un seuil minimal.
- quand le seuil maximal de mémoire libre est atteint Le processus s'arrete

KSWAPD

- le kswapd libère les pages qui sont enregistrées en storage et qui n'ont pas été modifiées
- si les pages non modifiées sont demandé, le système les copie du storage en RAM,



KSWAPD

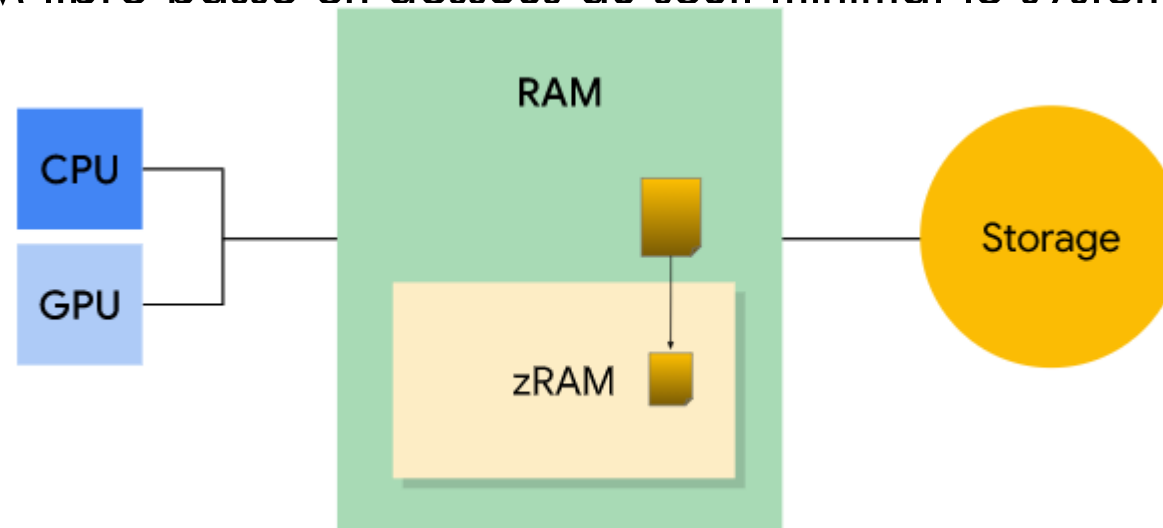
Les pages du cache et anonyme utilisés libérés de la RAM et sont compressés en zRam,

-si un processus les redemande alors elles sont décompressées et reviennent en RAM

-si le processus associé est terminé, les pages sont alors supprimées de zRAM

Dans le cas où la RAM libre passe en dessous du seuil minimal le système fait appel à





















LMK



LOW MEMORY KILLER LMK

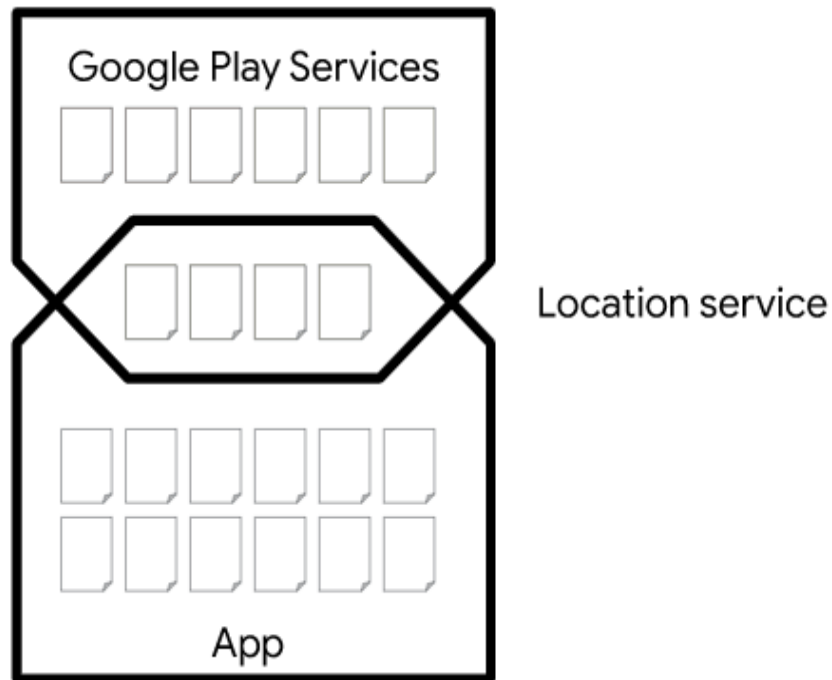
- Dans le cas où le kswapd ne libère pas assez de mémoire, le système utilise *onTrimMemory()* pour demander à l'application de réduire son allocation
- Si ce n'est pas suffisant, le système commence à terminer les processus à l'aide du LMK
- pour élire le processus à terminer, un score *oom_adj_score* est utilisé pour donner la priorité aux processus en cours d'exécution,
- les processus en arrière plan sont terminés en premier et les processus système en dernier

LMK ORDRE DE PRIORITÉ

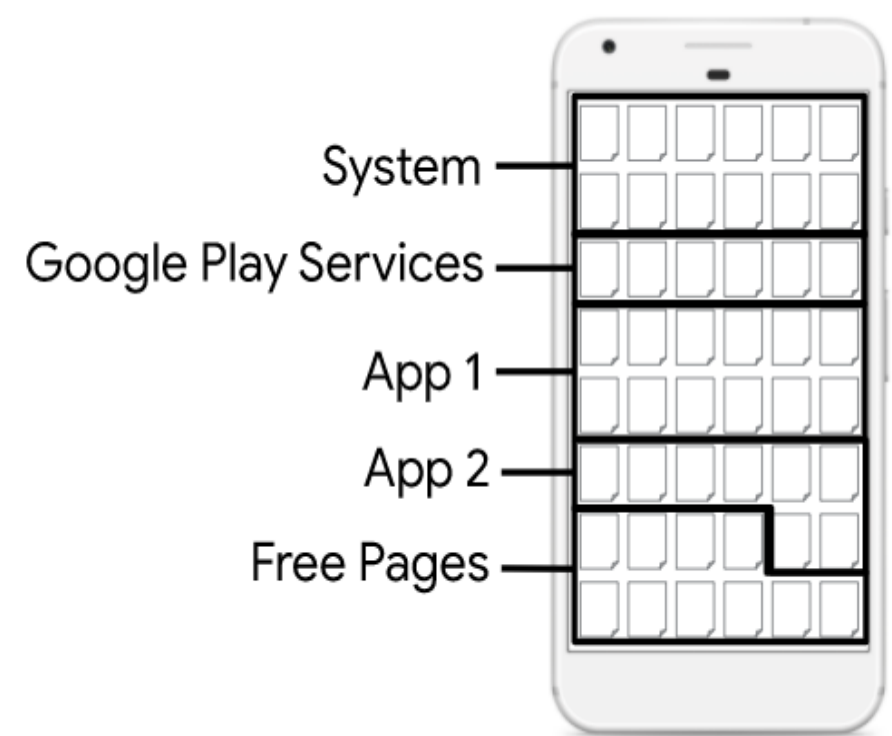
Background apps	    
Previous app	
Home app	
Services	  
Perceptible apps	  
Foreground app	
Persistent	     
System	<code>system_server</code>
Native	<code>init kswapd netd logd adbd installd</code>

CALCUL ESPACE OCCUPÉ PAR APPLICATION

Allocation avec pages partagées



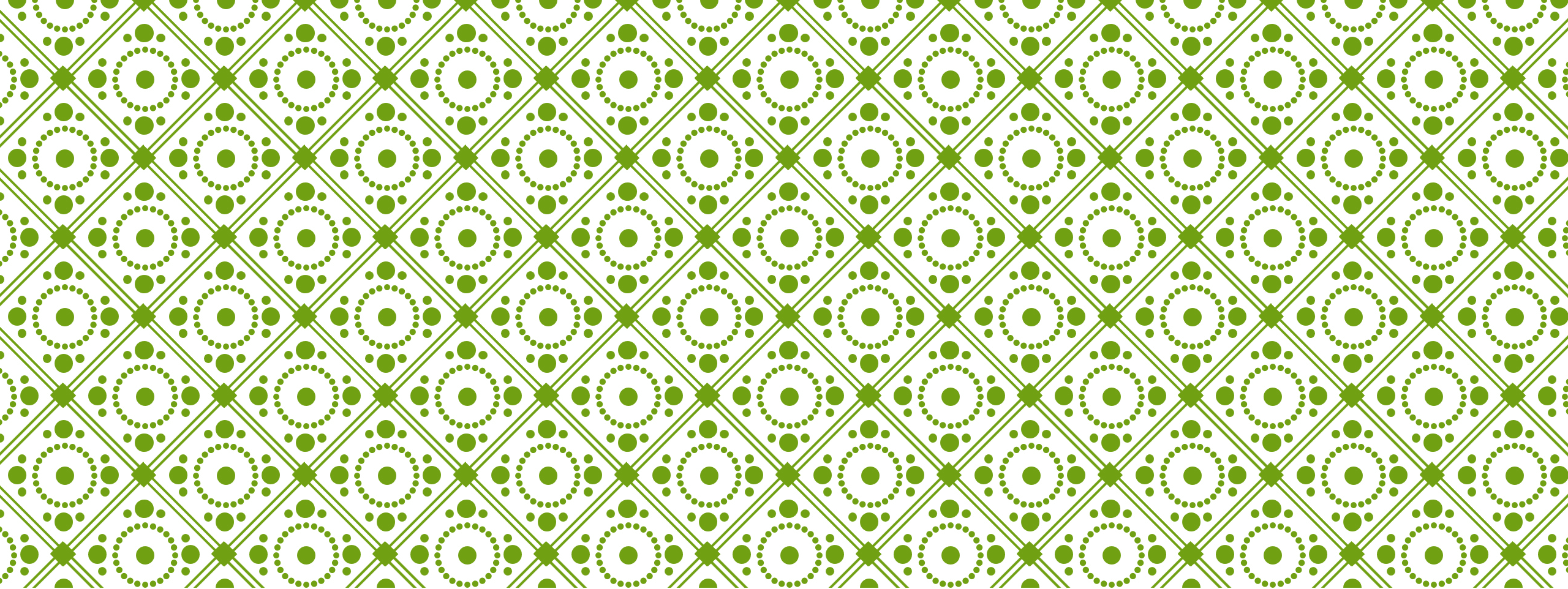
Allocation sans partage



CALCUL ESPACE OCCUPÉ PAR APPLICATION

Pour calculer l'espace occupé le système utilise les métriques suivantes :

- Resident Set Size (RSS): le nombre de pages partagées et non partagées
- Proportional Set Size (PSS): le nombre de pages non partagé et une distribution des pages partagées
- Unique Set Size (USS): le nombre de pages non partagées,



GESTION DE L'ENERGIE

GESTION DE L'ENERGIE

Le Android 9 (API level 28) a introduit de nouvelles fonctionnalités :

- App standby buckets : le système limite l'accès à l'application aux ressources suivant son utilisation par l'utilisateur
- Battery saver improvements : quand la batterie, applique les restrictions à toutes les applications,

APP STANDBY BUCKETS

Le système catégorise les applications en fonction de leurs utilisation fréquente et ressentie en 5 niveaux :

1. **Active:** l'application est en cours d'exécution (a lancé une activité, un service d'avant plan,,,,)

Aucune restriction est appliquée

2. **working** set: l'application est utilisée mais pas active exp. facebook

Restriction moyenne est appliquée

3. **frequent** : l'application est utilisée frequemment exp. Sport app

Restriction forte

APP STANDBY BUCKETS

4. **Rare:** application utilisé rarement exp hotel app

Restriction stricte

5. **Never** : application installée mais jamais utilisées

Restriction sévère

BATTERY SAVER IMPROVEMENTS

- Le system met les apps en standby mode plus aggresssivement et n'attend pas que app soit en veille.
- Background apps ont une execution limités, quell que soit leurs API level.
- Location services peut etre désactivé quand l'écran est verouillié.
- Background apps n'ont pas accès au réseau .