
Android Activity, Intent, IntentFilter

jean-michel Douin, douin au cnam point fr
version : 10 Février 2020

Notes de cours

Sommaire

- **Activity**
 - Présentation, cycle de vie
- **Intent, intentFilter**
 - Démarrage d'une autre activité
 - Au sein de la même application
 - Depuis une autre application,
 - Passage de paramètres
 - Un résultat produit par l'activité est attendu
 - Attente du résultat
- **Communiquer entre deux activités**
 - Plusieurs façons
- **Permission**
- **Génie logiciel, conception**
 - Approche par composants Logiciels induite, comme méthode de conception ?
- **Annexes**

Intergiciel : fonctionnalités attendues

- **Devenir une activité**
 - Quel « protocole » doit-on respecter ?
 - Quelles sont les classes à hériter ?
 - Quelles sont méthodes à redéfinir ?, quel cycle de vie ? Est-il imposé ?
- **Communiquer**
 - Entre deux activités, entre deux applications, entre deux processus,
 - Attente d'un résultat d'une activité ?
- **Sélection de la « bonne » application**
 - Une activité se met à la disposition des autres ... comment ?
 - Cette activité devient un fournisseur,
 - Maintenance, déploiement,
 - Substitution d'une application par une autre
 - ...

Bibliographie utilisée

CCY114/SMB116 : Certificat de compétences : intégrateur d'applications mobiles Cnam

<http://developer.android.com/resources/index.html>

Le cours de Victor Matos

<http://grail.cba.csuohio.edu/~matos/notes/cis-493/Android-Syllabus.pdf>

<http://www.vogella.com/android.html>

http://www.cs.unibo.it/projects/android/slides/android_intents.pdf

<http://www.cs.unibo.it/projects/android/index.html>

Plusieurs livres

Android A Programmers Guide - McGraw Hill

Professional Android Application Development – Wrox

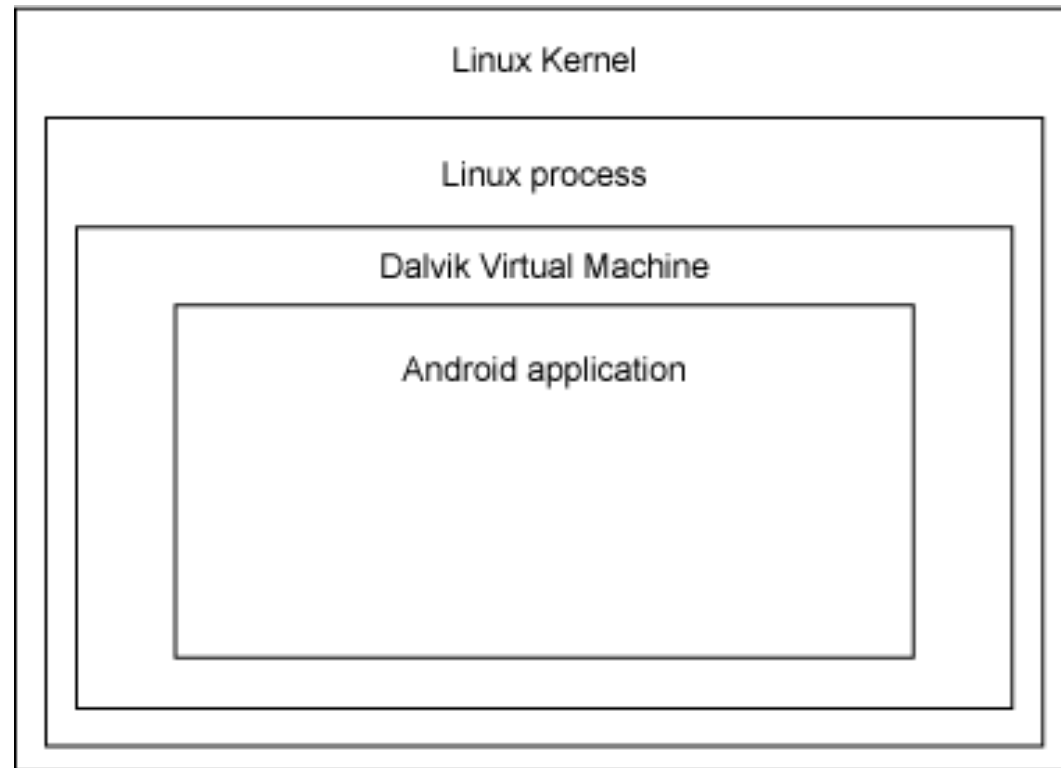
Le livre de Mark Murphy - Pearson

Présentation

- Une application peut être constituée de plusieurs écrans,
- A chaque écran lui correspond une activité,
- Une activité hérite et redéfinit certaines méthodes,
 - onCreate, -> ..., onPause, -> ..., onStop, ->...onDestroy, -> ...
- Android se charge des appels de ces méthodes,
 - Un état de l'activité est donc induit par l'exécution de ces méthodes
- Android impose un cycle de vie, un état de l'activité.
 - Inversion de contrôle
 - (cf. M.Fowler) <http://martinfowler.com/articles/injection.html>

Architecture, vocabulaire

- Source : <http://developer.android.com/guide/topics/fundamentals.html>

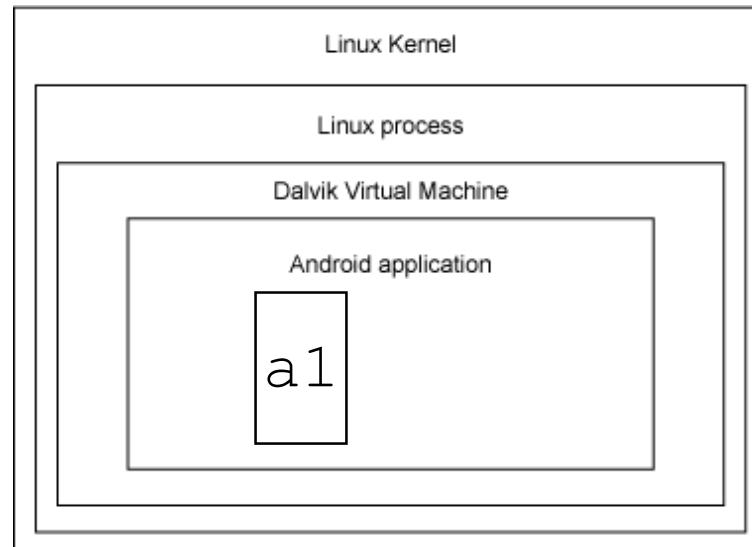


- **Application**
 - Activity
 - Service
 - ContentProvider
 - BroadcastReceiver

Une architecture possible

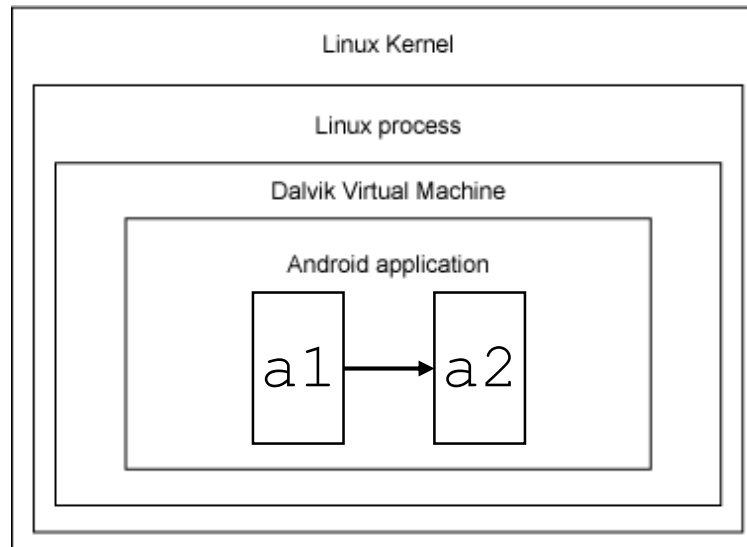
Un processus, une DVM, une application, une activité

`a1` est une activité



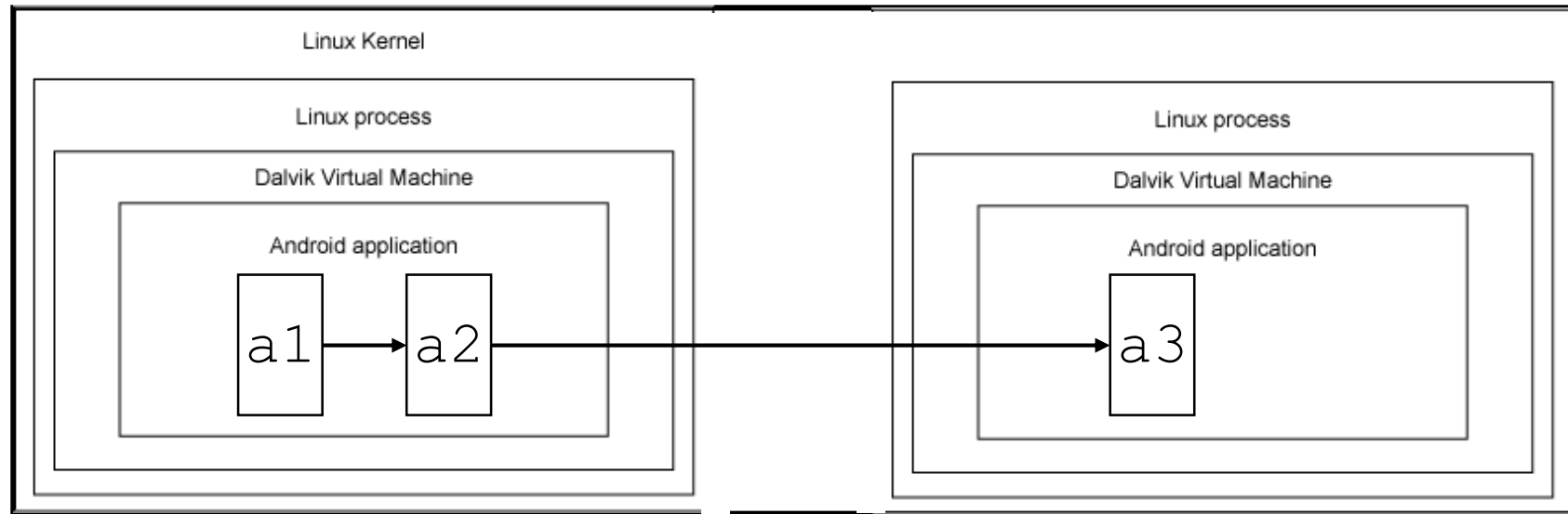
Architecture possible, suite

- Un processus, Une application, une DVM
- Deux activités,
 - Une activité peut en déclencher une autre



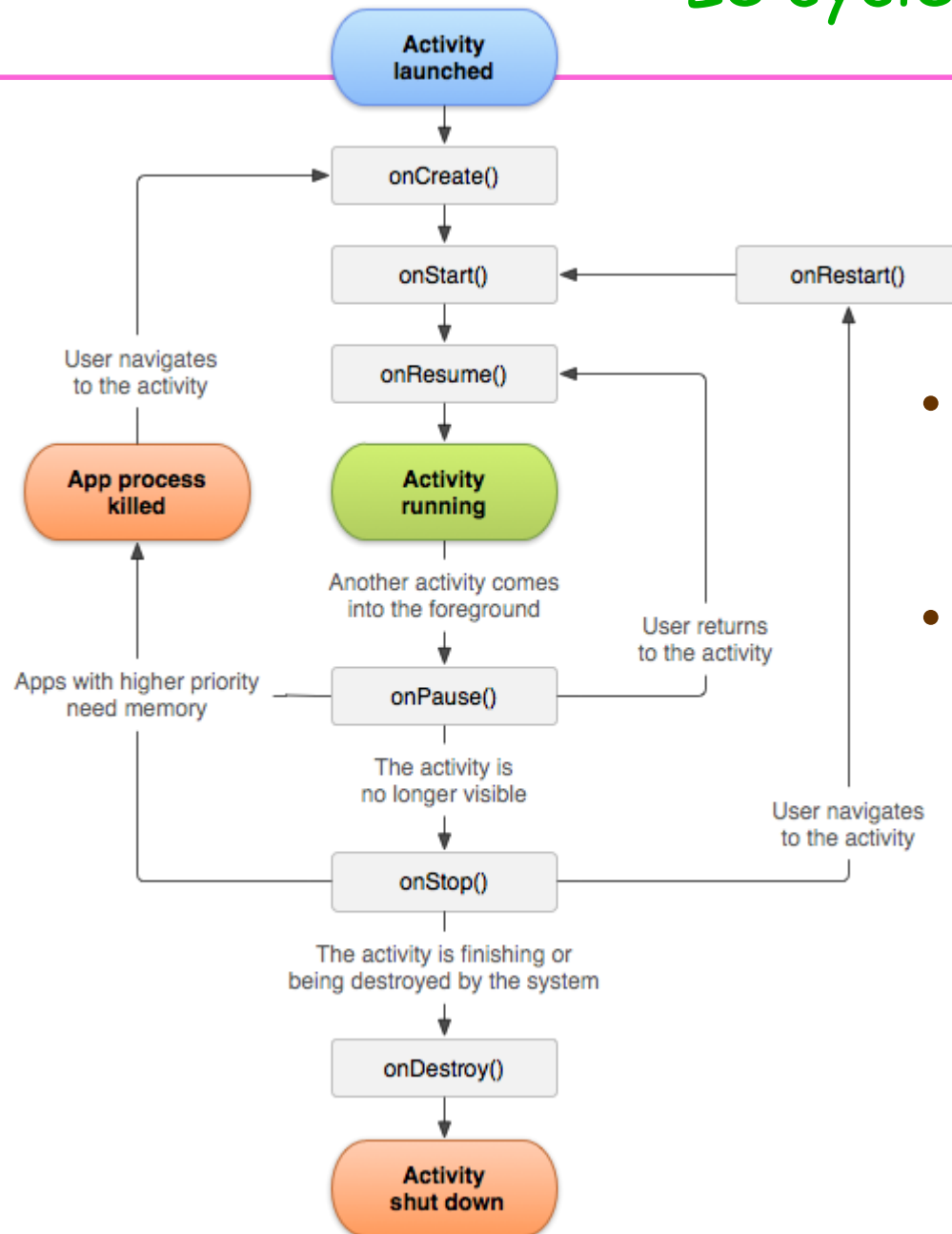
Architectures possibles fin...

2 processus, 2 applications, 2 DVM, 3 activités



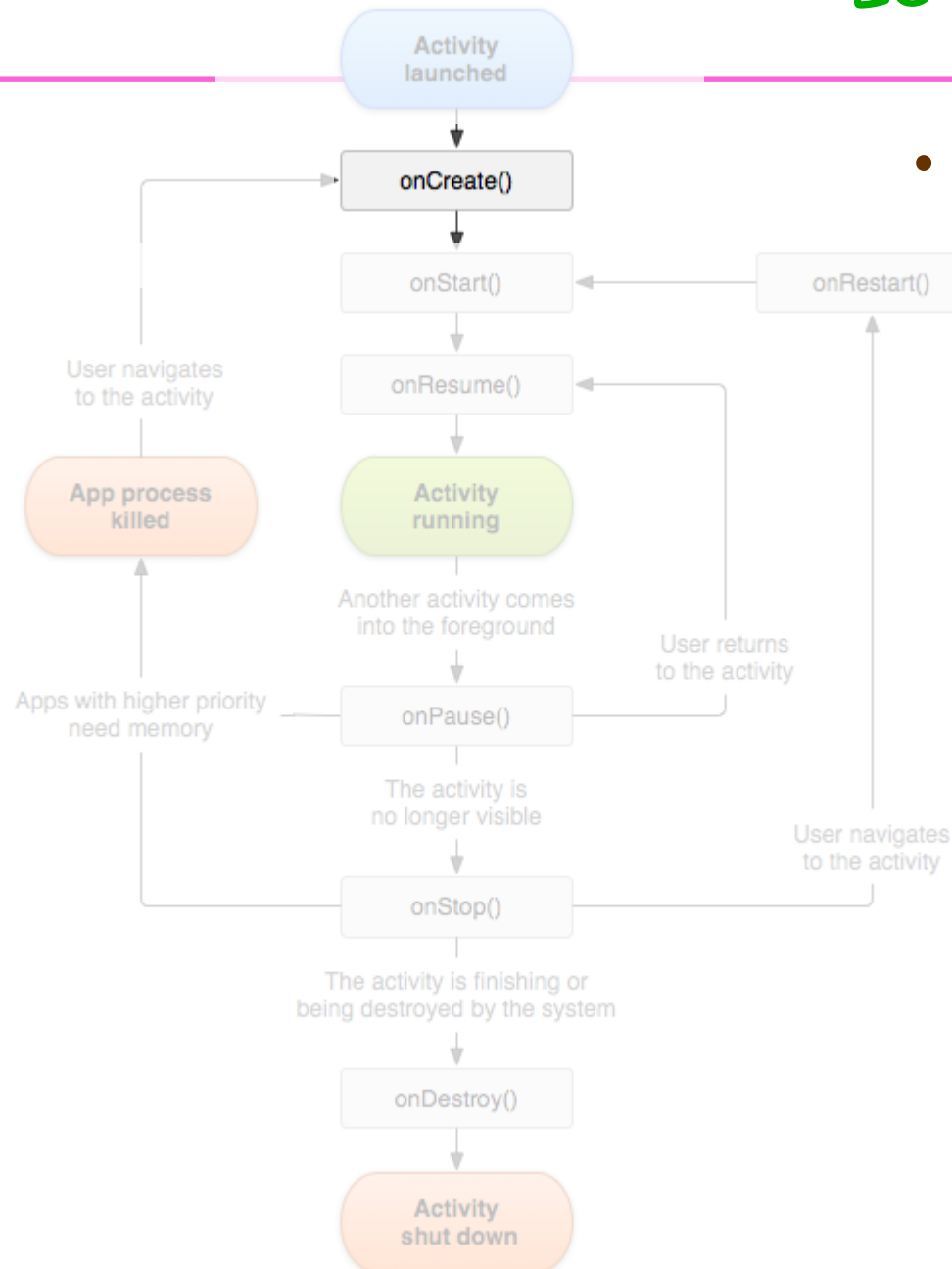
Une autre architecture possible :
2 processus, 2 DVM, Une application

Le cycle de vie d'une activité



- Séquencement imposé
 - Cf. M. Fowler, inversion de contrôle
- Android a le contrôle
 - Impose le séquencement

Le cycle de vie onCreate

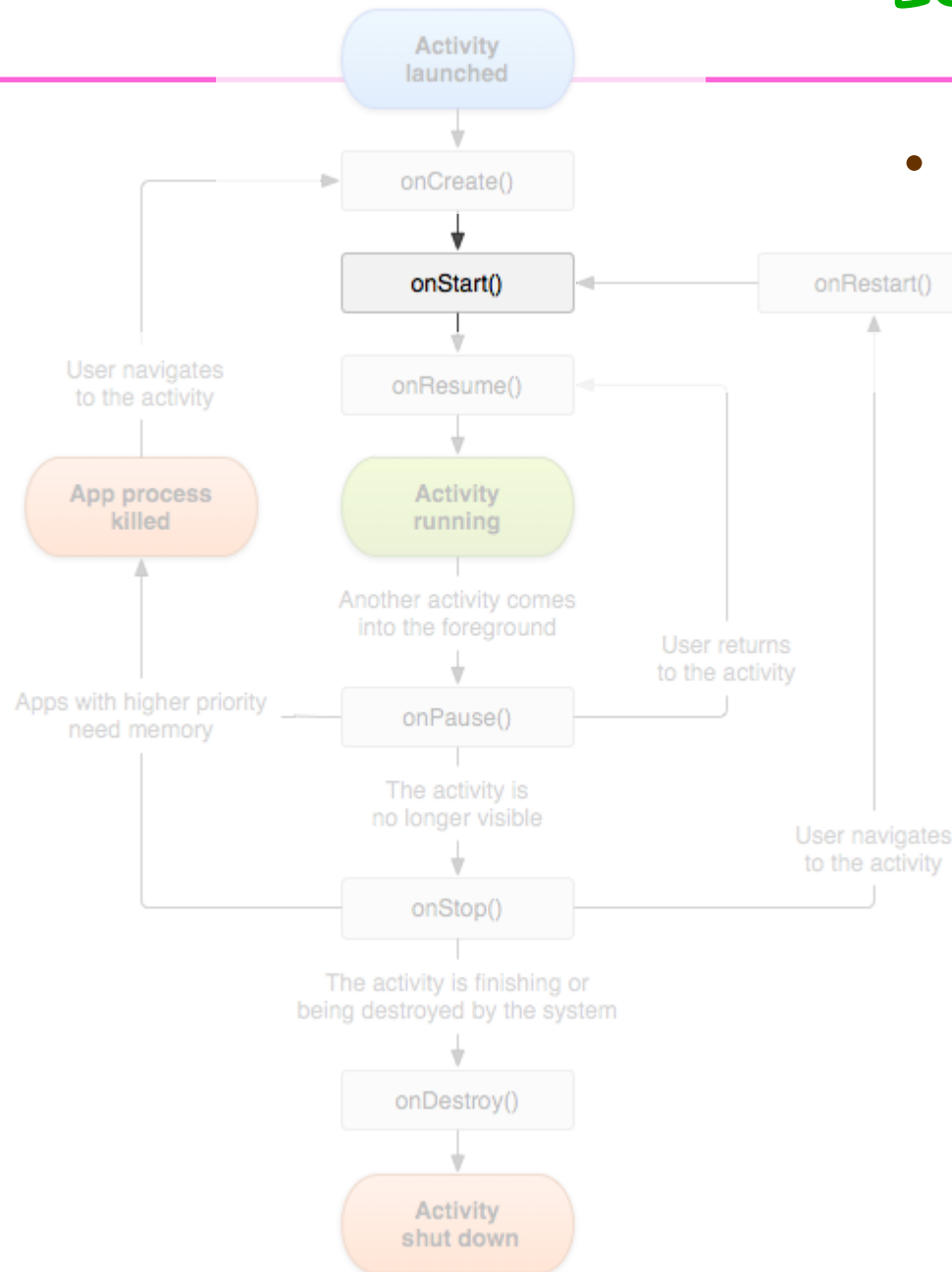


- onCreate

- Est appelée à la création de l'activité
- Contient les opérations d'initialisation

- Un *Bundle(Map)* est en paramètre
- Si *onCreate* se termine avec succès, alors *onStart* est appelée

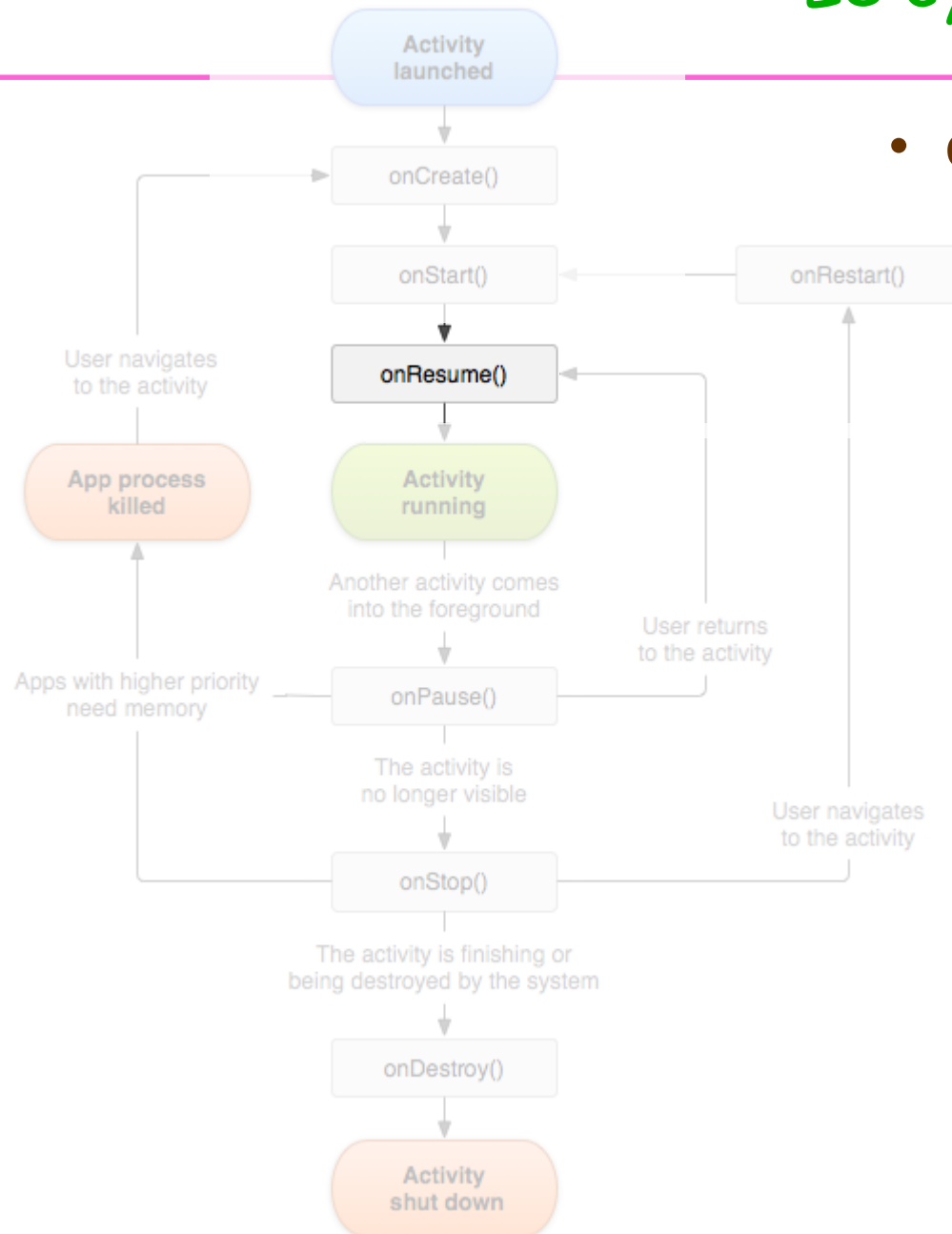
Le cycle de vie onStart



- onStart

- Est exécutée à la fin de *onCreate*
 - Nouvelle activité
- Est exécutée à la fin de *Restart*
 - Activité existante préalablement stoppée
- Est appelée « juste avant » que l'activité soit visible à l'utilisateur

Le cycle de vie onResume



- onResume

- Est appelée quand l'activité est prête à interagir avec l'utilisateur

- Est également appelée quand l'activité reprend son exécution, venant de l'arrière plan

- Si *onResume* se termine avec succès alors l'activité est en **phase d'exécution**

- *Soit la prise en compte des évènements de l'utilisateur, l'activité possède l'écran et le processus qui lui est associé*

Le cycle de vie onPause



- onPause

- Est appelée quand une autre activité passe au premier plan ou quand la touche retour est appuyée
- Doit libérer les ressources capteurs, ...
- **Est arrêtée** en fin de onPause,
- Android peut décider de l'arrêt par **destruction de cette activité** à tout moment
- La persistance de l'activité peut (doit) être assurée: **onSaveInstanceState**

Le cycle de vie onStop



- **onStop**

- L'activité n'est plus visible à l'utilisateur

- Est appelée quand

- Une autre activité est passée au premier plan. Cf.onResume
 - L'activité s'apprête à être détruite,

- *Android peut décider de l'arrêt par destruction de cette activité à tout moment*

Le cycle de vie onRestart



- **onRestart**

- Est appelée quand l'activité a été préalablement stoppée
- L'utilisateur a de nouveau sélectionné cette activité

Le cycle de vie onDestroy

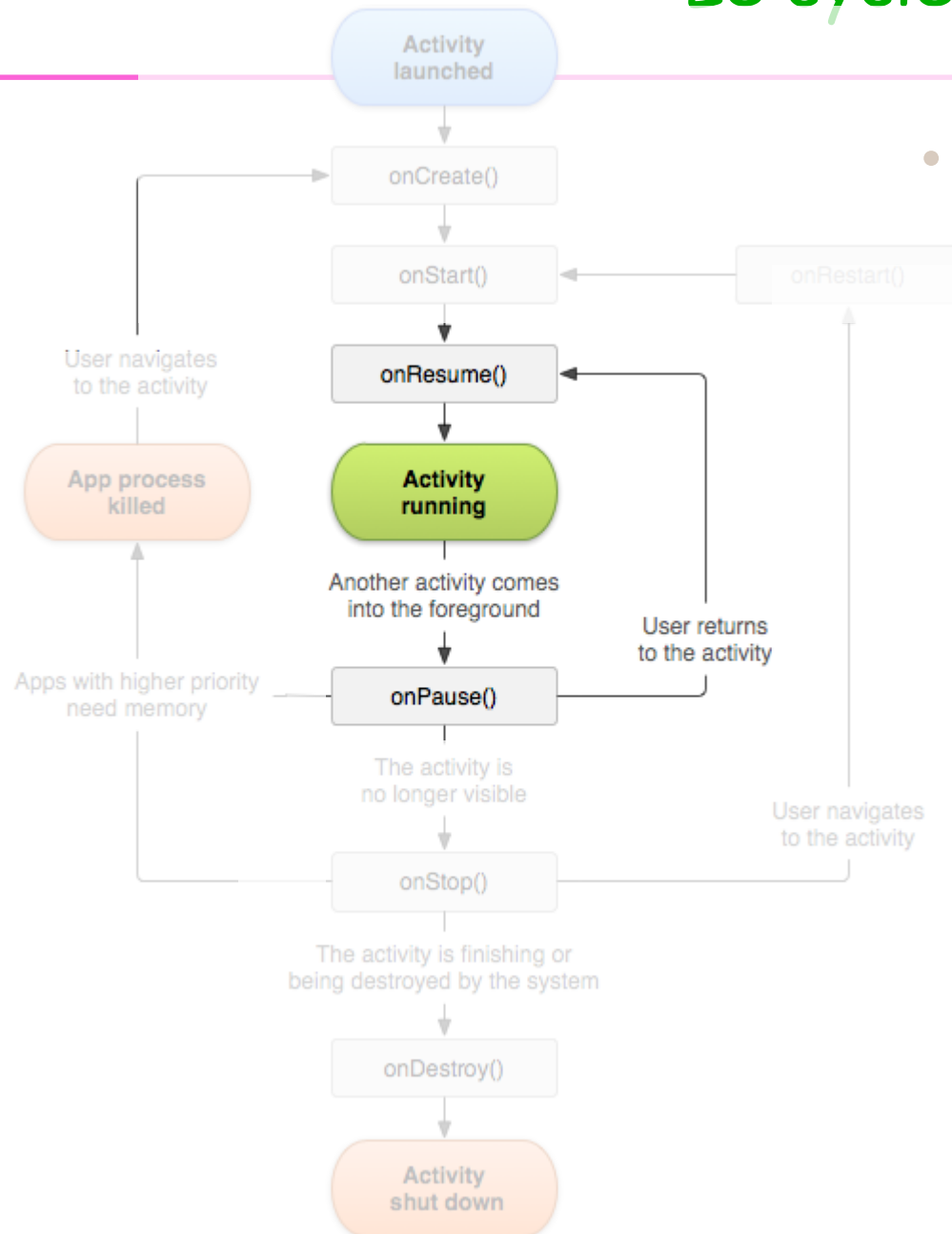


- **onDestroy**

- L'activité va être détruite

Le système a besoin de place
ou bien la méthode **finish** a été appelée

Le cycle de vie d'une activité



- Un scénario :

- Une activité affiche une fenêtre de dialogue

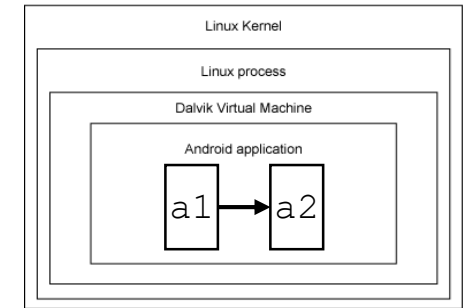
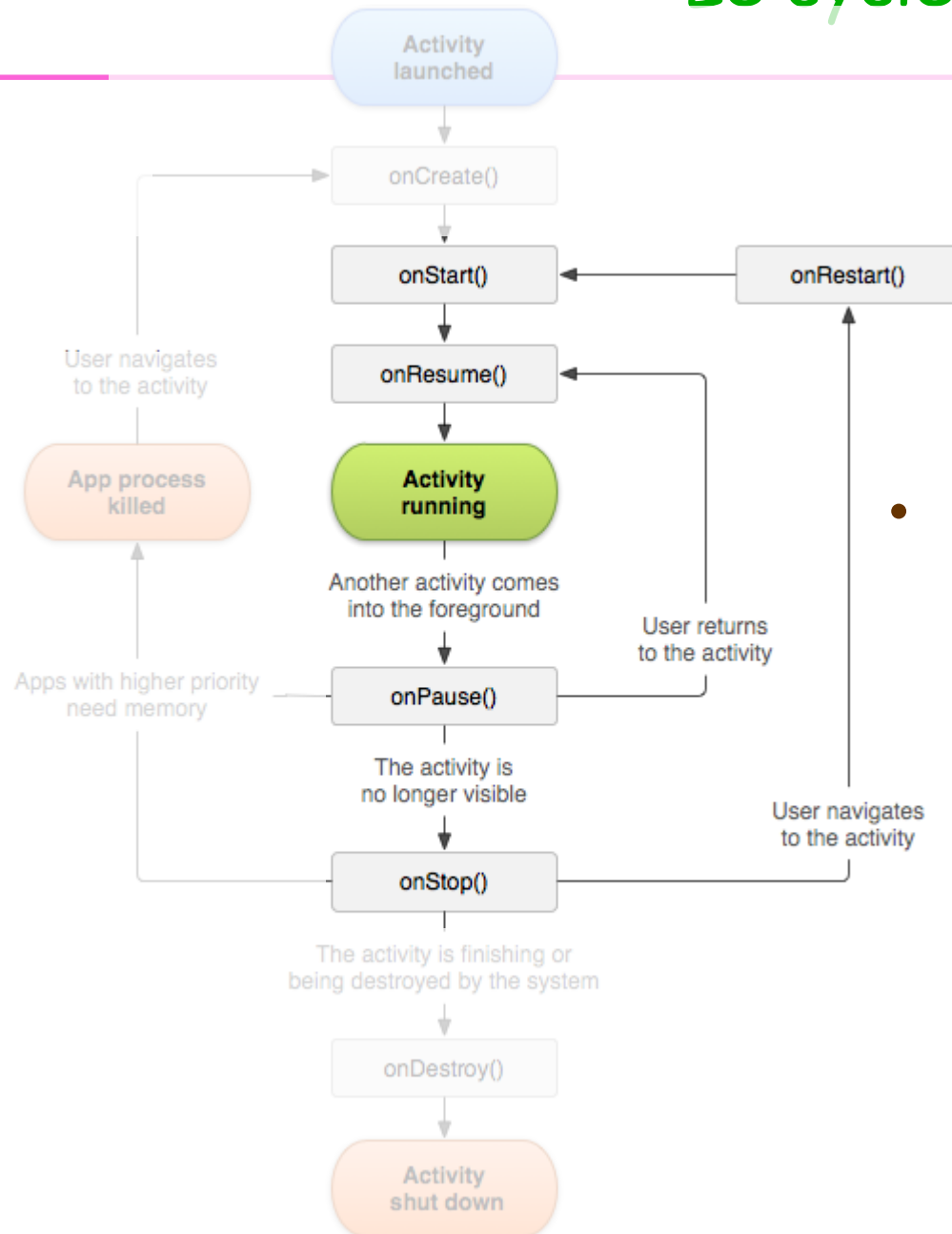
- L'activité est au second plan,

- Notée semi –visible dans la documentation
 - L'écran est flouté au second plan

- La fenêtre de dialogue est au 1er plan
 - **onPause (de l'activité)**
 - La fenêtre de dialogue disparaît
 - **onResume**

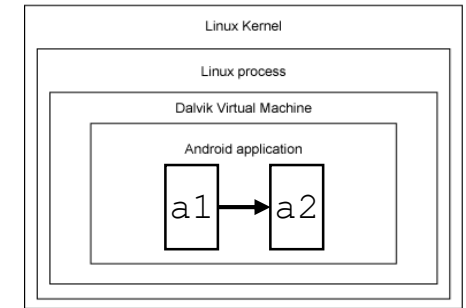
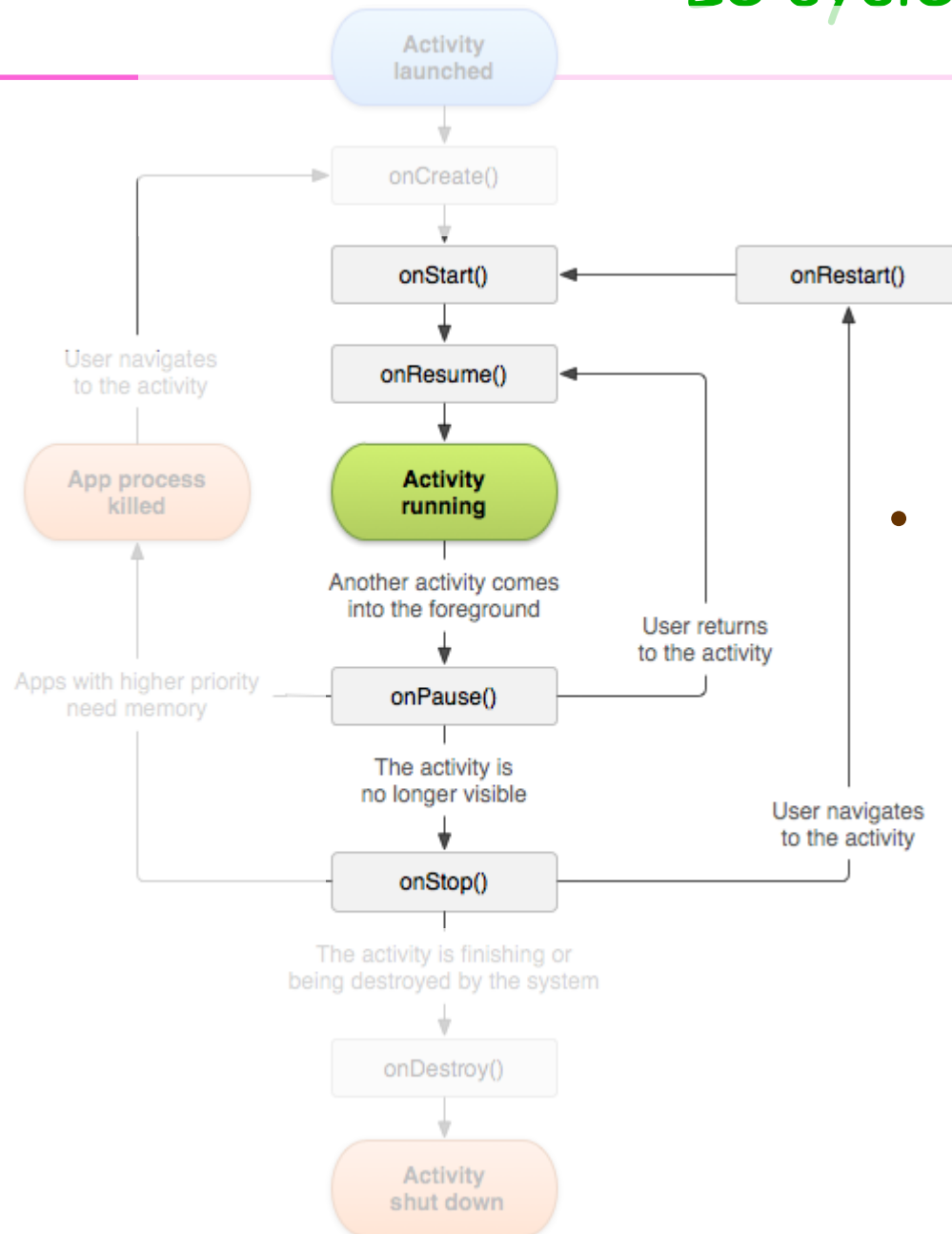
- **Phase d'exécution**

Le cycle de vie d'une activité



- Un autre scénario :
 - **a1** déclenche **a2**
 - L'activité a1 se met en pause
 - **onPause a1**
 - L'activité a2 « démarre »
 - Cf. son cycle de vie
 - 1. **onCreate a2**
 - 2. **onStart a2**
 - 3. **onResume a2**
 - 4. **onStop a1**,
(a1 n'est plus visible)
 - 5. **a2 poursuit son exécution**

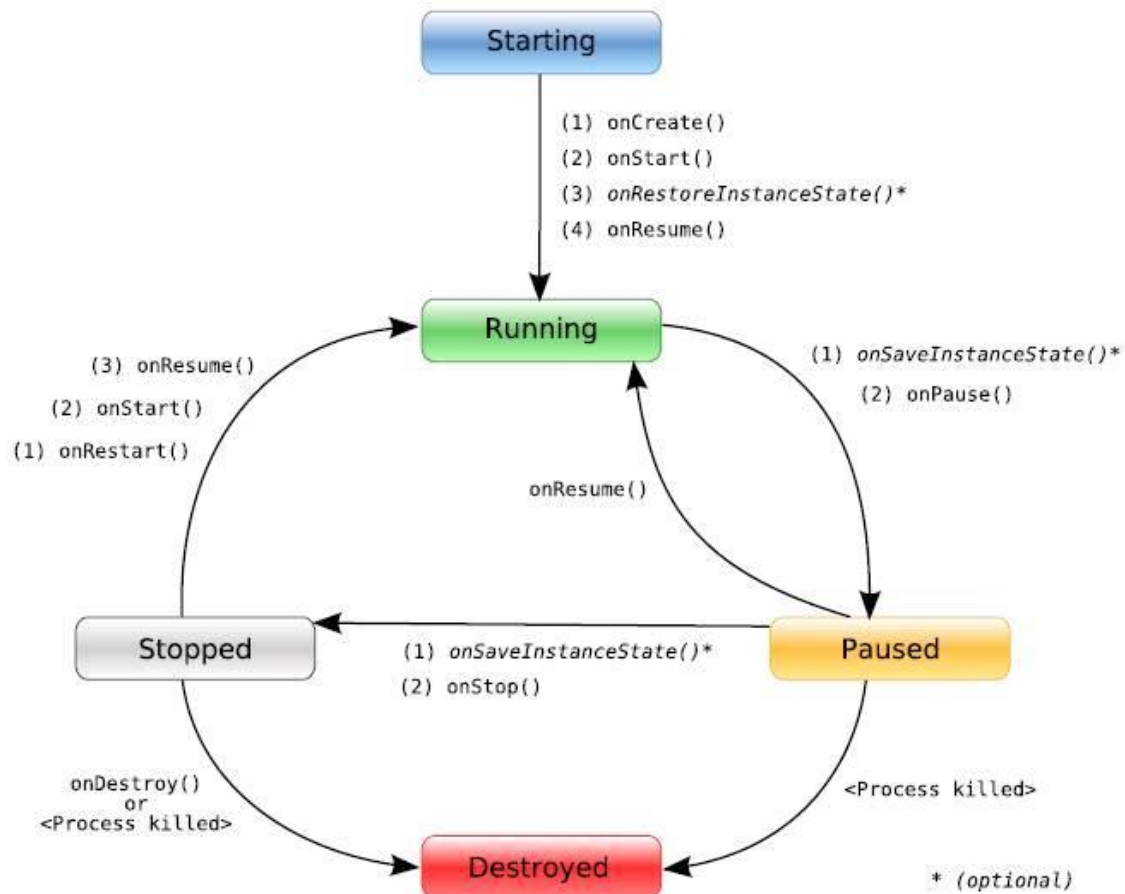
Le cycle de vie d'une activité



- scénario suite:

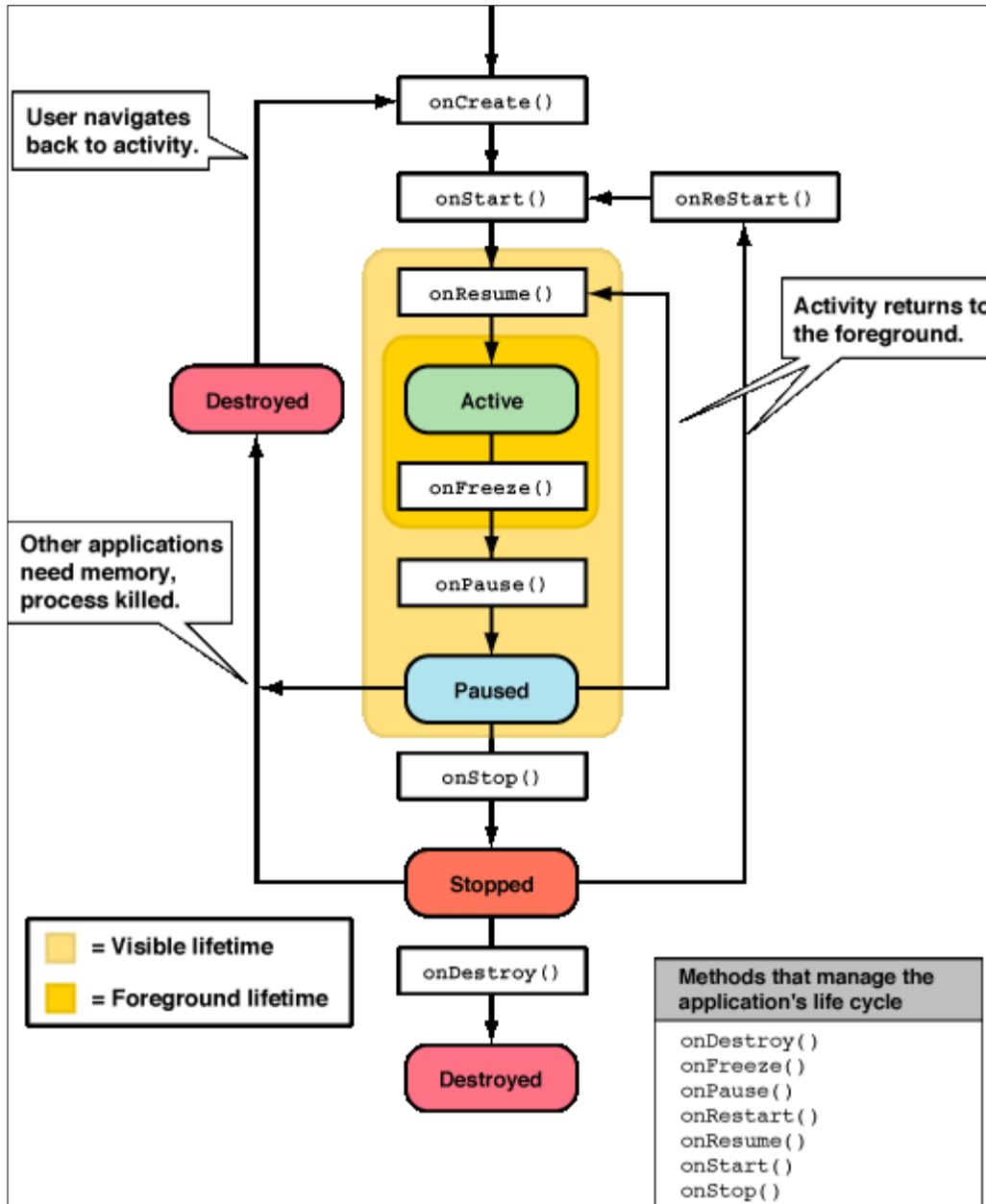
- **a2 se termine par un appel de finish**
- L'activité a2 se met en pause
 - **onPause a2**
- L'activité a1 « revient »
 1. **onRestart a1**
 2. **onResume a1**
 3. **onStop a2,**
(a2 n'est plus visible)
 4. **onDestroy**
 5. **a1 reprend son exécution**

États d'une Activité



- <http://inandroid.in/archives/tag/activity-lifecycle>

Cycle de vie + états...

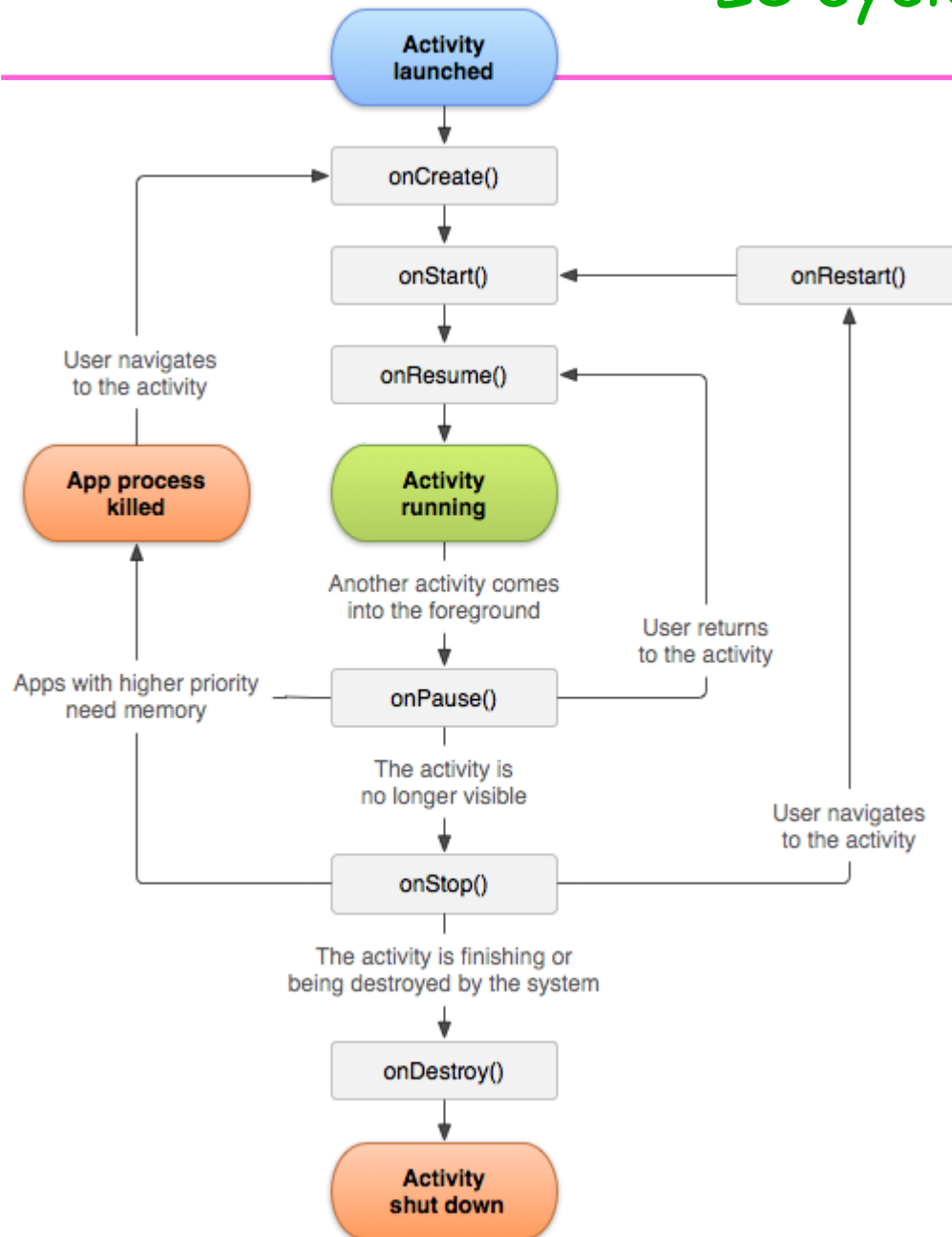


- Visible
- Semi-visible
- Stoppée

Note:

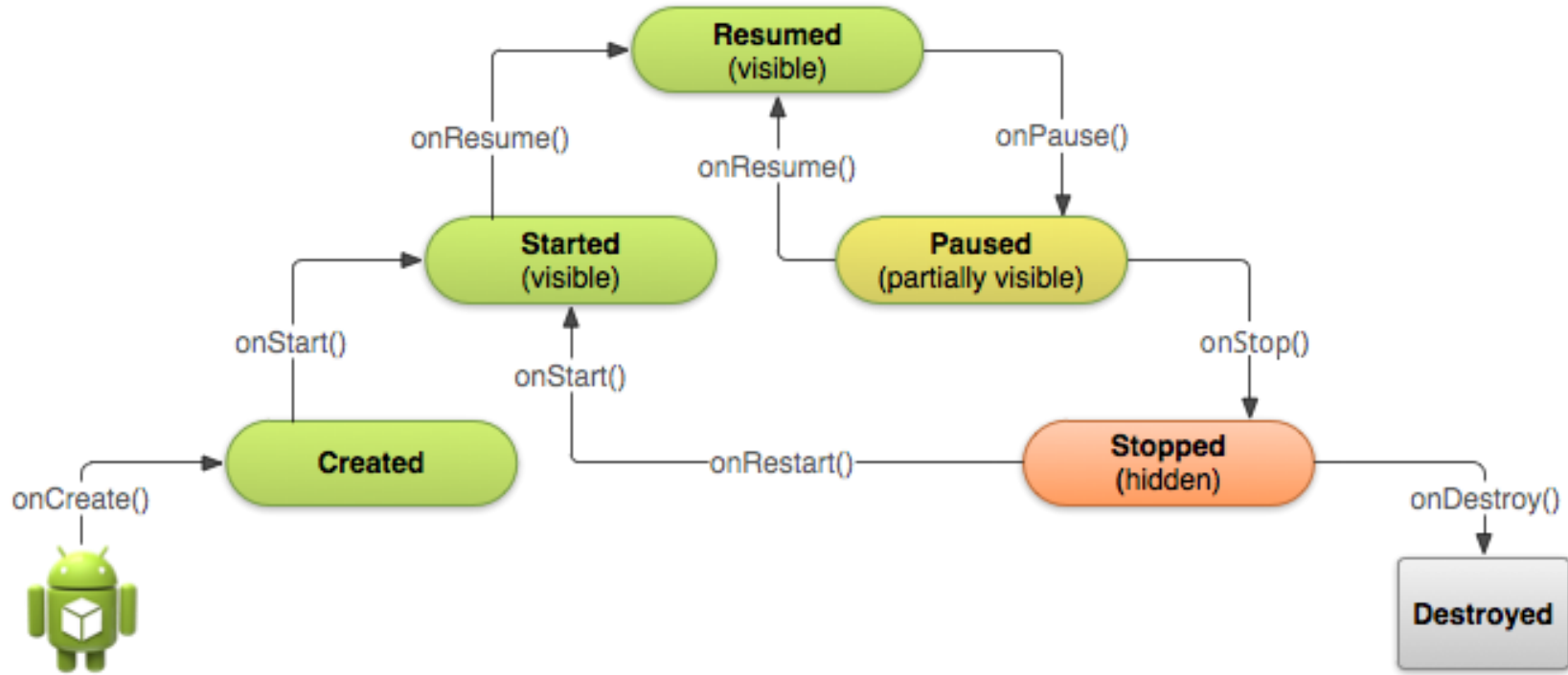
Attention **onFreeze** (Android 0.9) a été depuis renommée par **onSaveInstanceState** (la bien nommée)

Le cycle de vie d'une activité



- En résumé, une seule activité, un écran, un seul *Thread* (*l'UIThread*)
...
- Android gère le cycle de vie de vos activités

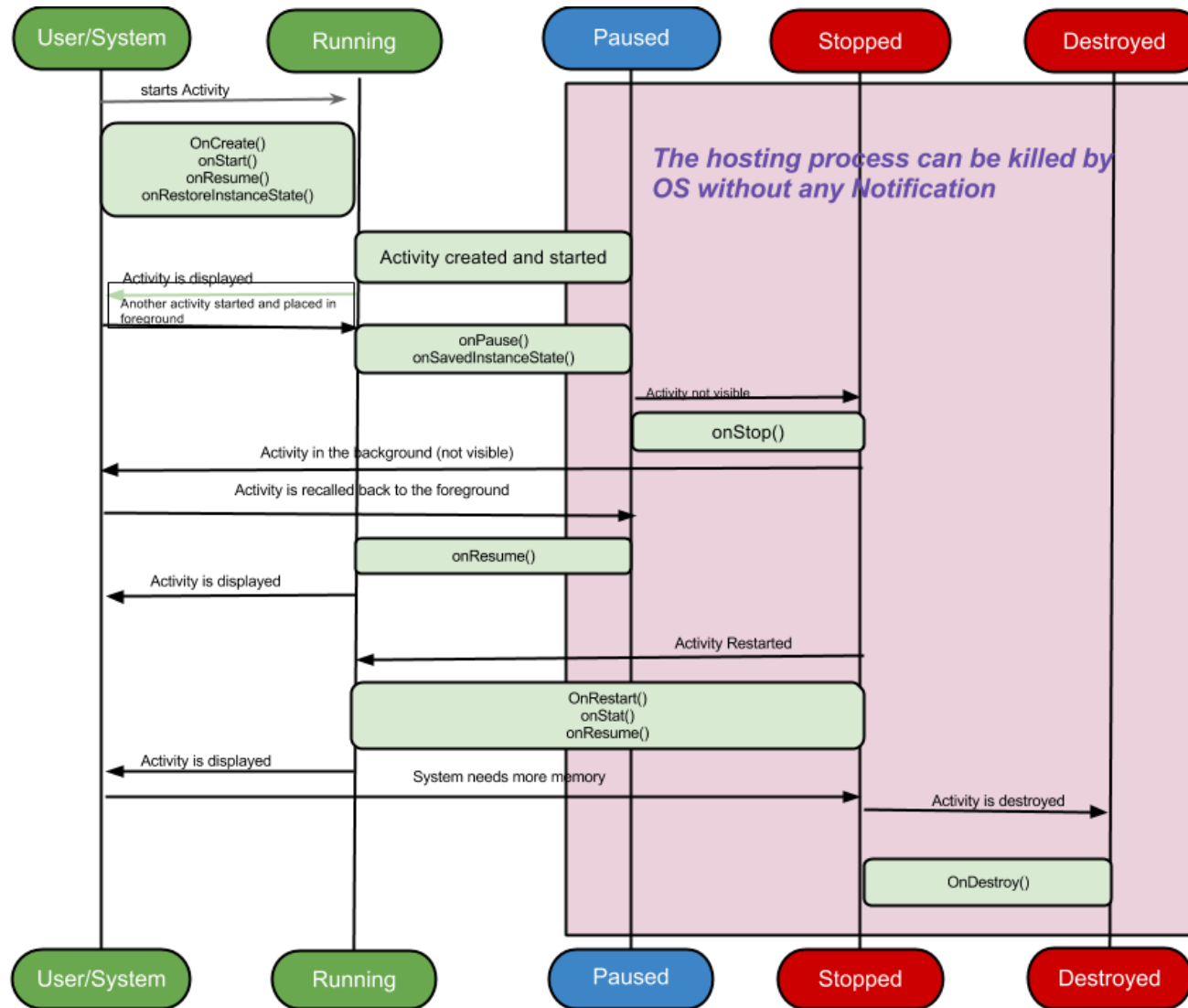
Avec une pyramide, doc développeur



<http://developer.android.com/training/basics/activity-lifecycle/starting.html#lifecycle-states>

Pour en savoir plus : une démo est à télécharger à cette URL

En diagramme de séquences



- <https://thamilandroid.wordpress.com/2013/04/29/android-life-cycle/>

Sommaire

- **Activity**
 - Présentation, cycle de vie
- **Intent, intentFilter**
 - Démarrage d'une autre activité
 - Au sein de la même application
 - Depuis une autre application,
 - Passage de paramètres
 - Un résultat produit par l'activité est attendu
 - Attente du résultat
- **Communiquer entre deux activités**
 - Plusieurs façons
- **Permission**
- **Génie logiciel, conception**
 - Approche par composants Logiciels induite, comme méthode de conception ?
- **Annexes**

Activité

- Créer une activité c'est

- Hériter de la classe **android.app.Activity**;

- Redéfinir certaines méthodes

- **onCreate**

- super.onCreate *cumul de comportement (obligatoire)*

- initialisation *des variables d'instance*

- setContentView *affectation de la vue*

- **onStart,**

- super.onStart *cumul du comportement*

- **onResume ...**

- super.onResume *cumul du comportement*

onCreate, onPause, ... sont appelées des « callback »

<http://developer.android.com/training/basics/activity-lifecycle/starting.html#lifecycle-states>

Activity les bases, HelloWorldActivity.java

- Une simple Activité, HelloWorld (*encore*)

```
import android.app.Activity;

public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

onCreate

- super.onCreate *cumul de comportement (obligatoire)*
- setContentView *affectation de la vue*

Description de l'activité pour Android

- **AndroidManifest.xml** pour HelloWorldActivity
 - Un langage destiné au système Android
 - **Précise**
 - Le nom de l'application, son icône,...
 - Le nom de l'activité,
 - Le nom du paquetage,
 - Les permissions,
 - Les autres activités de cette application,
 - Les autres services,
 - ...

AndroidManifest.xml

```
<application android:icon="@drawable/icon" android:label="@string/app_name" >

    ...
    <activity android:name=".HelloWorldActivity"
        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>
```

- Le nom de l'application, son icône,...
 - Le nom de l'activité,
- Comment le système déclenche t-il cette activité ?
- <intentFilter ?
 - <action ?
 - <category ?

AndroidManifest.xml

```
<application android:icon="@drawable/icon" android:label="@string/app_name" >

    <activity android:name=".HelloWorldActivity"
        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>
```

- **<intent-filter>** comme **IntentFilter**

A destination du système, un mécanisme de sélection de la « bonne » activité

Abonnement, souscription à ce type d'évènement

- Une *Intention* a été générée par le système

Intent : un message

- **Explicite**

- `Intent intent = new Intent(.....);`
- **La cible est précisée**
 - Dans le constructeur
 - `new Intent(this, ActivityClassName.class)`
 - Par un appel de **`setClassName`**
 - `i.setClassName(Context, ActivityClassName)`
 - Ou bien de **`setComponentName`**
 - `i.setComponentName(package, package+.ActivityClassName)`

- **Implicite**

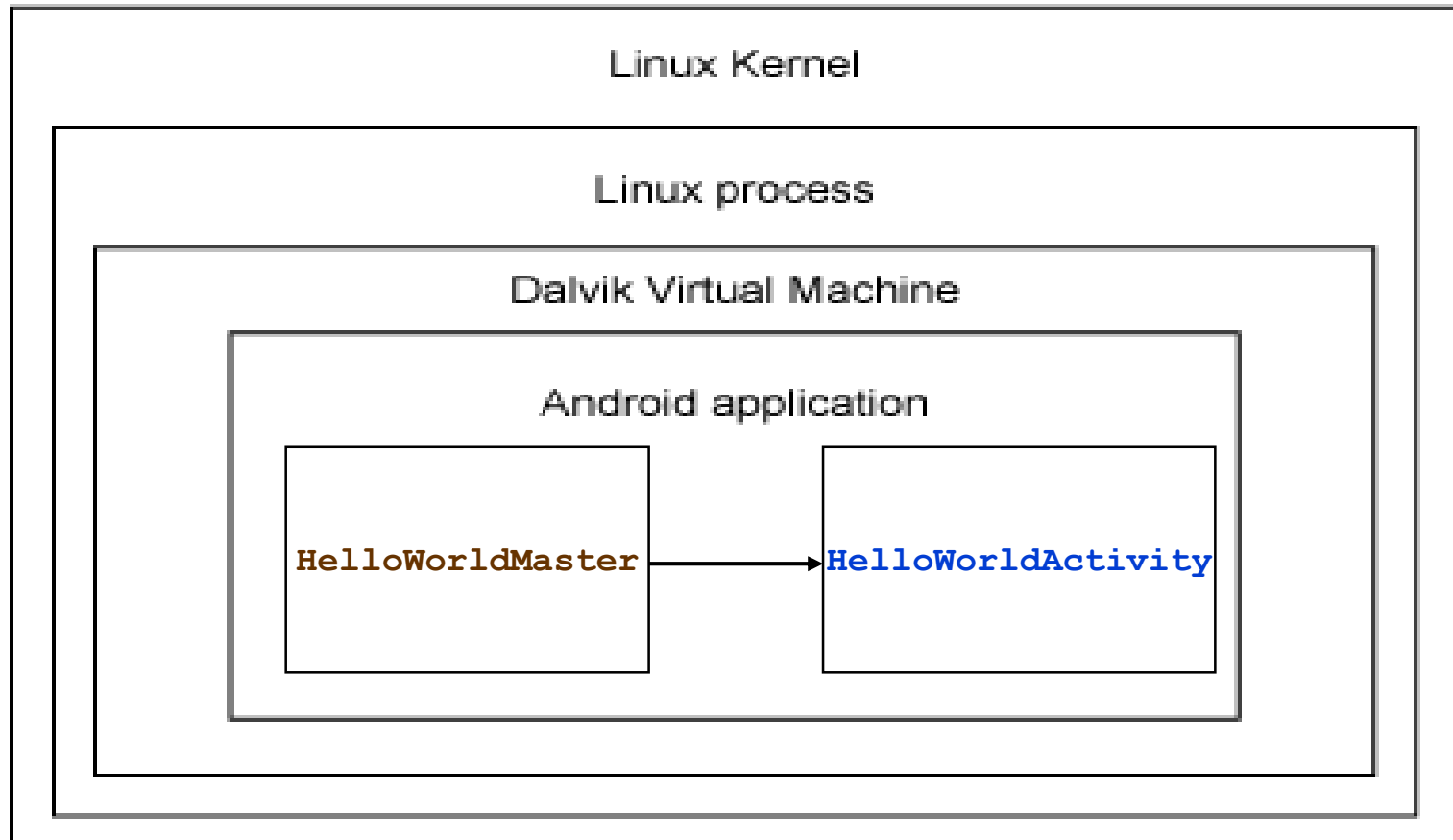
- **La cible ou les cibles dépendent du filtre**
- **Une résolution est effectuée**
 - **Sélection des activités répondant aux critères**
 - **Critères comme**
 - **Le nom d'une action (ex: `Intent.ACTION_DIAL`)**
 - **Une catégorie d'action (ex: `android.intent.category.DEFAULT`)**
 - **Une donnée (ex: `http://www.google.com`)**

Sommaire

- *Activity*
 - Présentation, cycle de vie
- **Intent, IntentFilter**
 - Démarrage d'une autre activité
 - Au sein de la même application
 - Depuis une autre application,
 - Passage de paramètres
 - Un résultat produit par l'activité sélectionnée
 - Attente du résultat,
- Communiquer entre deux activités
 - Plusieurs façons
- Permission
- Génie logiciel, conception
 - Approche par composants Logiciels induite, comme méthode de conception ?
- Annexes

Une DVM, une application deux activités

- **HelloWorldMaster** déclenche **HelloWorldActivity**
Au sein de la même application



Intent Explicite, constructeur

- Au sein de la même application, dans le constructeur

```
public class MasterActivity extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        Intent intent = new Intent(this, HelloWorldActivity.class);  
        startActivity(intent);  
    }  
}
```

AndroidManifest.xml

```
<application android:icon="@drawable/icon" android:label="@string/app_name" >  
  
    <activity android:name="cnam.essais.MasterActivity"  
        android:label="@string/app_name">  
        <intent-filter>  
            <action android:name="android.intent.action.MAIN" />  
            <category android:name="android.intent.category.LAUNCHER" />  
        </intent-filter>  
  
    </activity>  
    <activity android:name="cnam.essais.HelloWorldActivity"></activity>  
</application>
```

Intent Explicite, setClassName

- **Au sein de la même application, tjs MainActivity**

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    Intent intent = new Intent();  
    intent.setClassName(this, "cnam.essais.HelloWorldActivity");  
  
    // ou intent.setClassName(getApplicationContext(),  
    //                               "cnam.essais.HelloWorldActivity");  
    startActivity(intent);  
}
```

ManifestAndroid.xml identique au précédent

Séqencement, diagramme d'état, illustration

- **Par la pratique ...**

Une trace est installée dans chaque méthode,

- **onCreate, onStart ... des deux activités**

```
public void onCreate(Bundle b){  
    super.onCreate(b);  
    Log.i(TAG, " onCreate ");
```

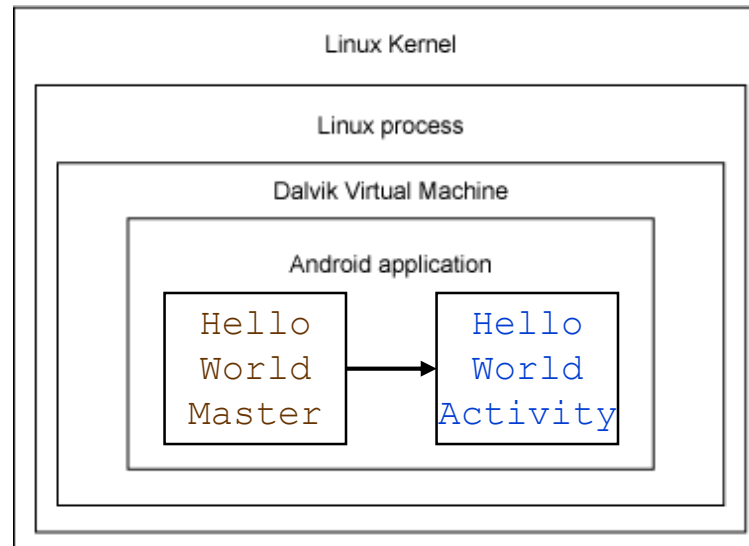
Avec TAG = getClass().getName()

Démonstration

```
HelloWorldMaster      onCreate  
HelloWorldMaster      onStart  
HelloWorldMaster      onResume  
HelloWorldMaster      onPause  
HelloWorldActivity    onCreate  
HelloWorldActivity    onStart  
HelloWorldActivity    onResume  
HelloWorldMaster      onStop  
KeyCharacterMap        No keyboard d  
KeyCharacterMap        Using default  
HelloWorldActivity    onPause  
HelloWorldMaster      onRestart  
HelloWorldMaster      onStart  
HelloWorldMaster      onResume  
HelloWorldActivity    onStop  
HelloWorldActivity    onDestroy  
HelloWorldMaster      onPause  
HelloWorldMaster      onStop  
HelloWorldMaster      onDestroy
```

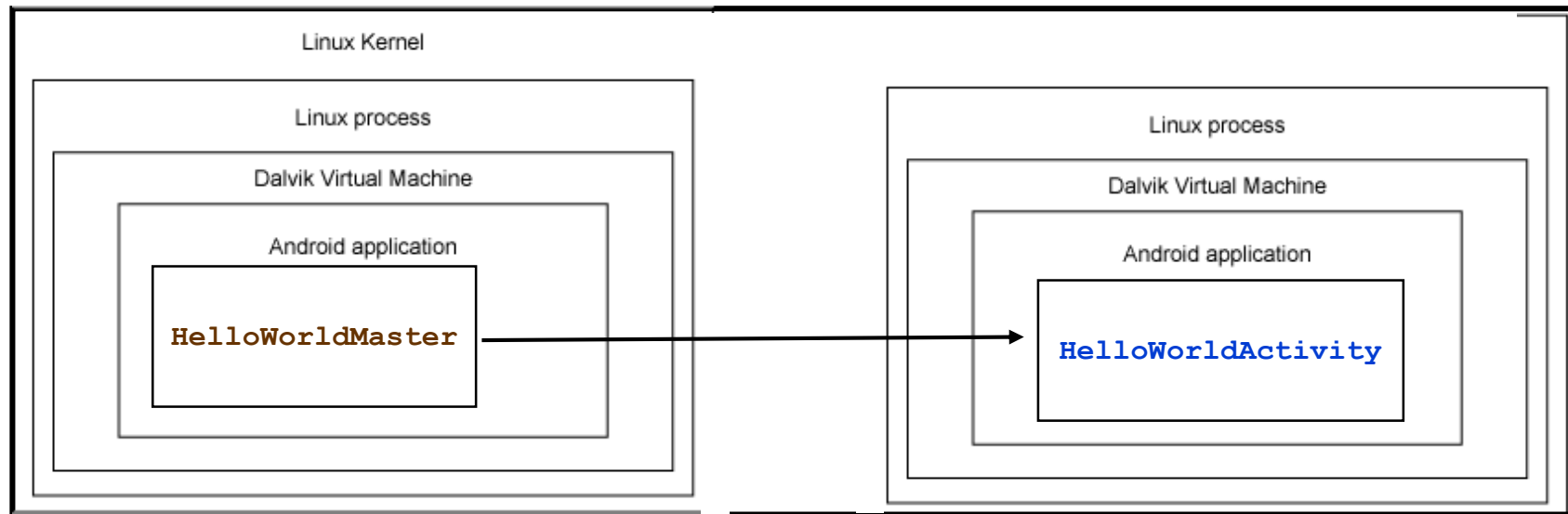
Une DVM, une application deux activités

- **HelloWorldMaster** déclenche HelloWorldActivity
Une application, un processus, une DVM



Une DVM, une application deux activités

- **HelloWorldMaster** déclenche HelloWorldActivity
Une application, deux processus, deux DVM



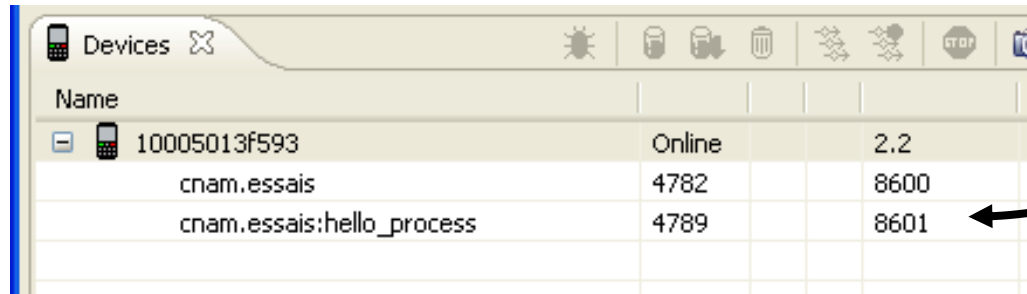
Une application, deux processus, deux DVM

- AndroidManifest.xml

```
<application android:icon="@drawable/icon" android:label="@string/app_name" >

    <activity android:name=".MasterActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>

    </activity>
    <activity android:name=".HelloWorldActivity" android:process=":hello_process"></activity>
</application>
```



Name			
10005013f593	Online		2.2
cnam.essais	4782		8600
cnam.essais:hello_process	4789		8601

Séquencement identique et constaté

- **HelloWorldMaster**

1. onCreate

- startActivity ...

2. onStart

3. onResume

4. onPause

8. onStop

10. onRestart

11. onStart

12. onResume

- **HelloWorldActivity**

5. onCreate

6. onStart

7. onResume

– Touche retour

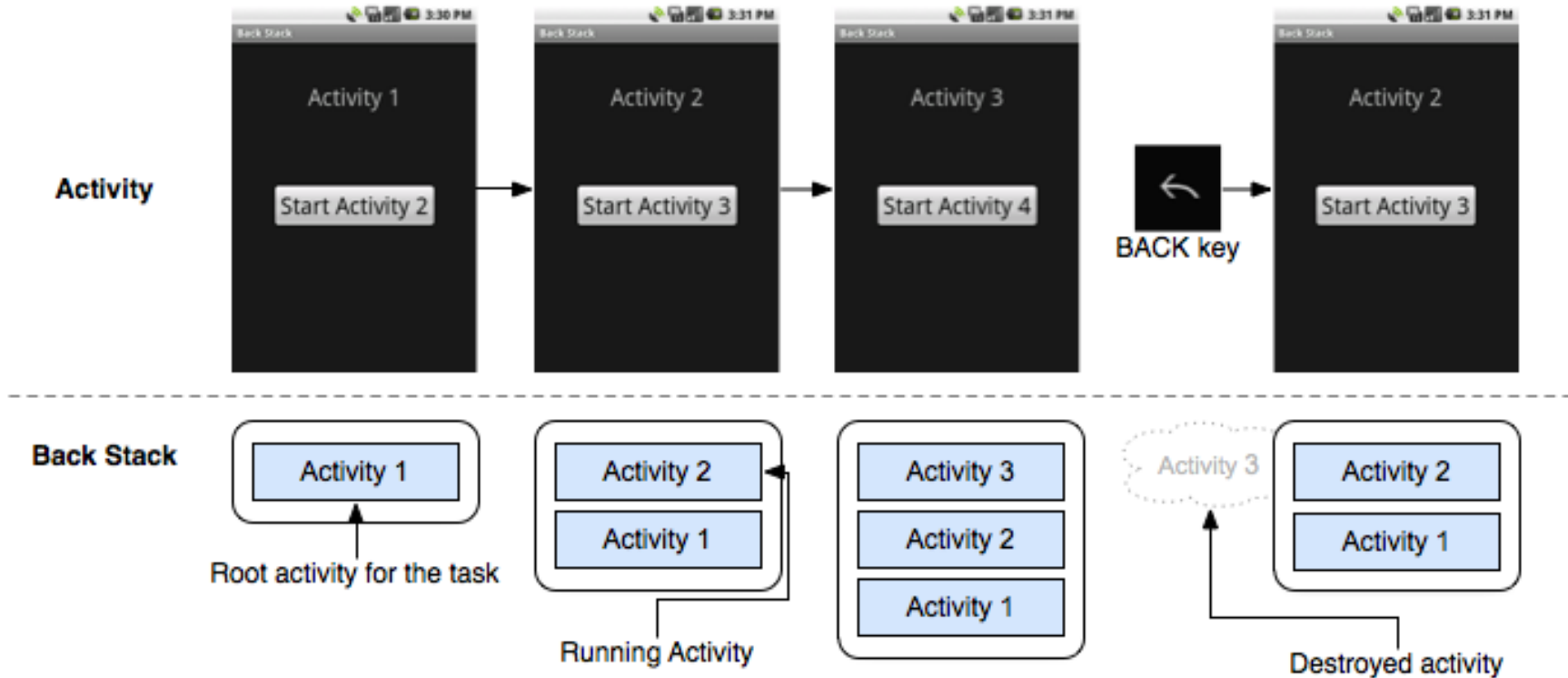
9. onPause

13. onStop

- HelloWorldActivity s'arrête,
- HelloWorldMaster redevient le maître

• À trois activités ?, un appel urgent ? *Une pile des activités*

Une pile des activités est en place



- <http://docs.huihoo.com/android/3.0/guide/topics/fundamentals/tasks-and-back-stack.html>

Intent explicite, setComponentName

- Une autre application,
 - Au sein de laquelle une activité HelloWorldMaster est créée
 - `cnam.master.HelloWorldMaster`
 - Cette activité se charge de déclencher la simple Activité ...
 - `cnam.essais.HelloWorldActivity`

```
public class HelloWorldMasterActivity extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        Intent intent = new Intent();  
        String pkg    = "cnam.essais";  
        String clazz = pkg+ ".HelloWorldActivity";  
        intent.setComponent(new ComponentName(pkg,clazz));  
        startActivity(intent);  
    }  
}
```

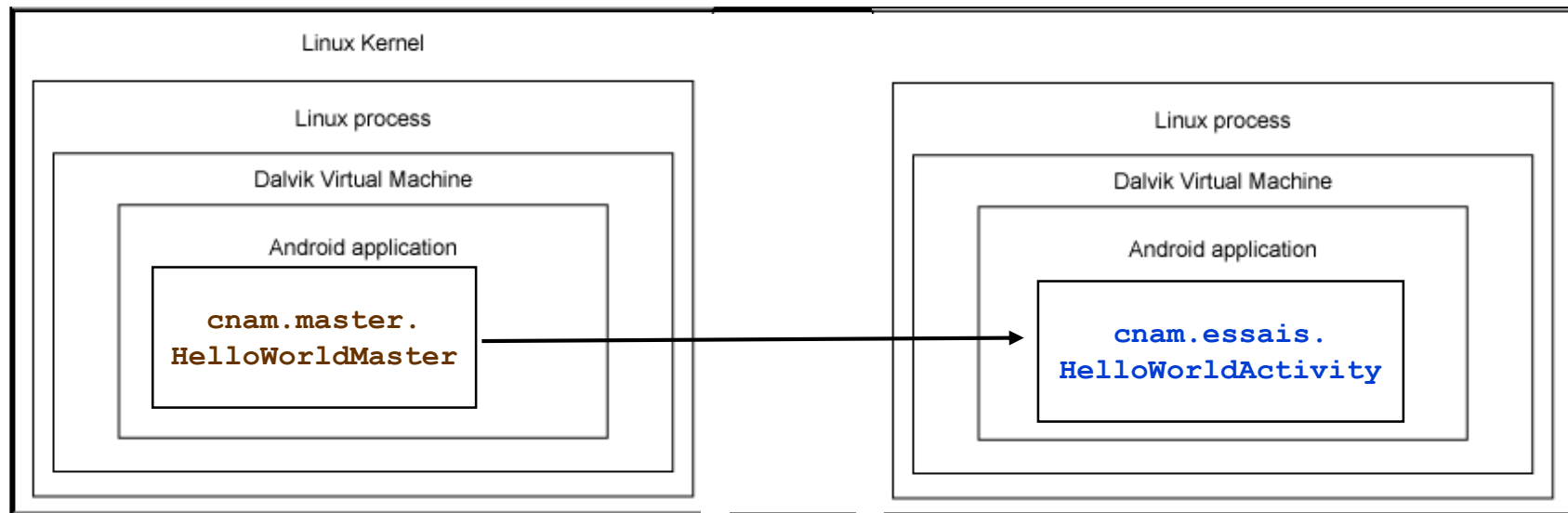
Intent explicite, setComponentName

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
  <activity android:name=".HelloWorldMasterActivity"
    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>
```

- **Une autre application, une autre activité**
 - La balise `<activity>` de l'activité appelée n'est plus présente

Une autre activité d'une autre application

- Une autre activité d'une autre application est spécifiée
 - Par son nom de *package*, son nom de classe
 - Deux applications, deux processus, deux DVM



- Avec
 - `<activity android:exported="true" android:name=".HelloWorldActivity" />`
» Au sein de AndroidManifest.xml

Séquencement identique

- **HelloWorldMaster**

1. onCreate
 - startActivity ...
2. onStart
3. onResume
4. onPause
8. onStop
10. onRestart
11. onStart
12. onResume

- **HelloWorldActivity**

5. onCreate
6. onStart
7. onResume
 - Touche retour
9. onPause

13. onStop

Intent Explicite

- **Démonstration**

- Divers, syntaxe

Note : `setClassName`

Intent	<code>setClassName(Context packageContext, String className)</code> Convenience for calling <code>setComponent (ComponentName)</code> with an explicit class name.
Intent	<code>setClassName(String packageName, String className)</code> Convenience for calling <code>setComponent (ComponentName)</code> with an explicit application package name and class name.

Un résumé

- **Intent explicite**
 - Le nom de la classe est connue
 - `setComponentName`
 - Au sein de la même DVM
 - `AndroidManifest` le précise (`<activity`
 - Dans une autre application
 - Le nom du paquetage identifie l'activité cible
- **La suite : Intent implicite**
 - Abstraction du nom de la classe
 - Notion de clients demandeurs / fournisseurs

Intent implicite

- Le système sélectionne la ou les bonnes activités
 - En fonction de critères
 - Un nom d'action, une donnée ...
- Exemple
 - Renseignons le filtre de l'activité simple **bonjour.ACTION !!!**
 - Enregistrement auprès du système Android avec cette ACTION,

```
<activity android:name=".HelloWorldActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="bonjour.ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

- Cette activité devient éligible à deux types d'*Intention*

Intent implicite, le Maître ...

- HelloWorldMaster à la recherche d'une bonne ACTION
 - Quels sont les fournisseurs de `bonjour.ACTION` ?
 - HelloWorldActivity est sélectionné et s'exécute

```
public class HelloWorldMasterActivity extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        Intent intent = new Intent();  
        intent.setAction("bonjour.ACTION");  
        startActivity(intent);  
    }  
}
```

Android reçoit l'*intention*, du client
parcourt ses tables, à la recherche de la bonne activité
Trouve et sélectionne le fournisseur HelloWorldActivity

Ajoutons une seconde Activity

- Avec le même filtre, *bonjour.ACTION*
 - *Dis bonjour à la dame* est une nouvelle activité
 - Le code est identique ... HelloWorld

```
<application android:icon="@drawable/icon" android:label="@string/app_name" >

    <activity android:name=".DisBonjourALaDameActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="bonjour.ACTION" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>

</application>
```

– Cette activité est éligible à un seul type d'*Intention*

Intent implicite, le Maître ...

- **HelloWordMaster à la recherche d'une bonne ACTION**
 - Quels sont les fournisseurs de `bonjour.ACTION` ?

```
public class HelloWorldMasterActivity extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        Intent intent = new Intent();  
        intent.setAction("bonjour.ACTION");  
        startActivity(intent);  
    }  
}
```

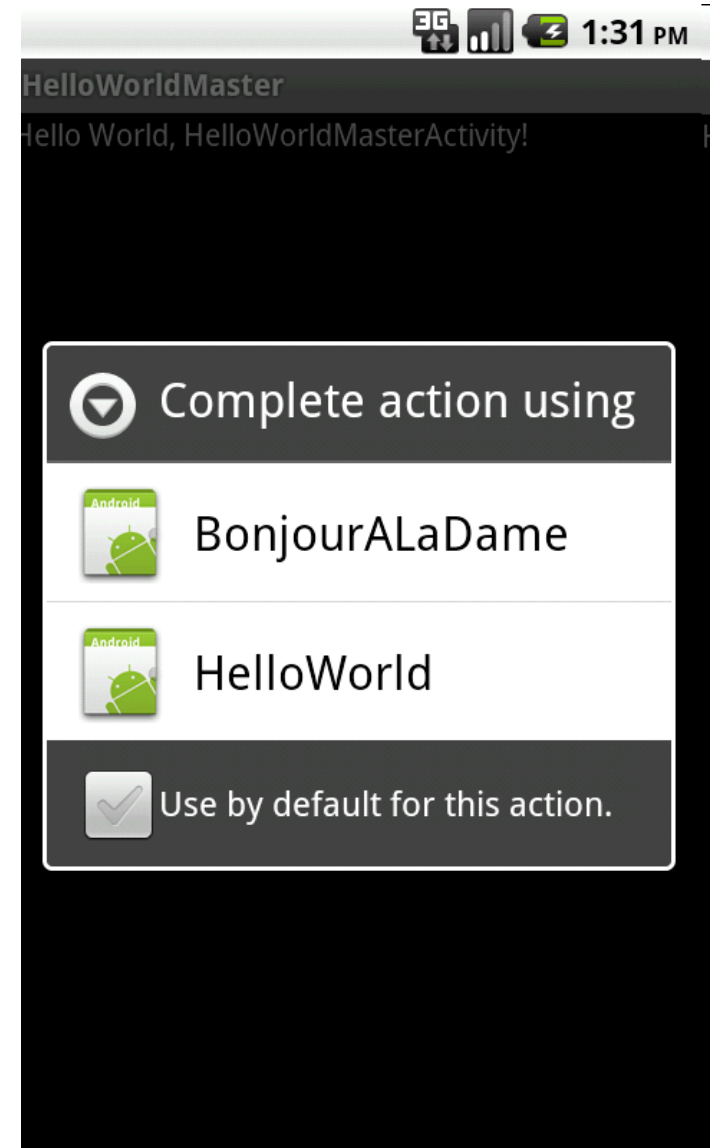
- **Nouvelle exécution de l'application**
- **Nous avons maintenant plusieurs fournisseurs possibles**
 - `HelloWord.Activity`
 - `DisBonjourALaDameActivity`

Le Maître a maintenant le choix

- **Qui voudrait dire Bonjour ?**
 - Quels sont les fournisseurs de ce service ?
- **2 activités sont candidates**

Une fenêtre de dialogue apparaît ...

Décidez vous ...

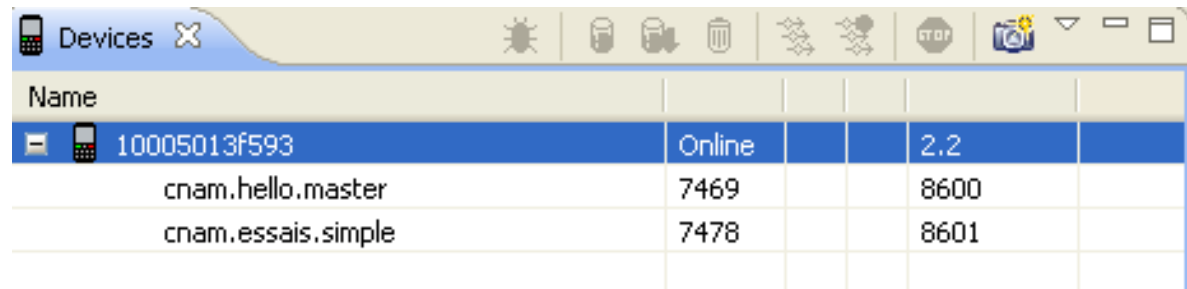


Intent implicite

- **Démonstration**

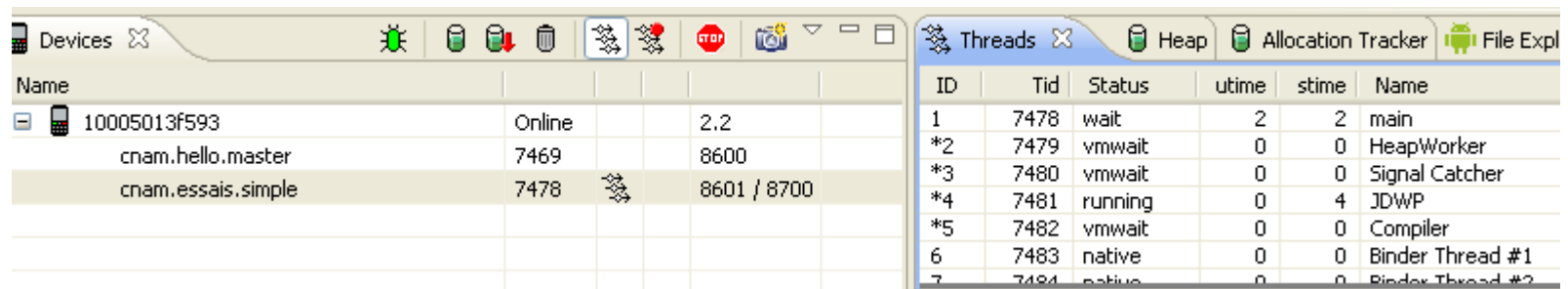
Remarques ... post démonstration

- L'activité sélectionnée engendre un processus
 - Processus avec sa DVM



Name			
10005013f593	Online		2.2
cnam.hello.master	7469		8600
cnam.essais.simple	7478		8601

- Ce processus engendre des *Thread*, au démarrage de la DVM dont l'un d'entre eux (*main*) se charge
 1. De créer l'activité
 2. D'exécuter les méthodes selon le cycle de vie



ID	Tid	Status	utime	stime	Name
1	7478	wait	2	2	main
*2	7479	vmwait	0	0	HeapWorker
*3	7480	vmwait	0	0	Signal Catcher
*4	7481	running	0	4	JDWP
*5	7482	vmwait	0	0	Compiler
6	7483	native	0	0	Binder Thread #1
7	7484	native	0	0	Binder Thread #2

Intent implicite, autres critères

- Envoyer un SMS : *HelloWorld*

```
Intent intent = new Intent(Intent.ACTION_VIEW);

intent.setData(Uri.parse("sms:"));

intent.putExtra("sms_body", "hello world");
startActivity(intent);
```

- Ouvrir un fichier : *hello_world.mp3*

```
Intent intent = new Intent(Intent.ACTION_VIEW);
File hello = new File("/sdcard/hello.mp3");

intent.setDataAndType(Uri.fromFile(hello), "audio/mp3");

startActivity(intent);
```


Intent implicite, autres critères

- **Téléphoner**

```
String url = "tel:43556";
```

```
Intent intent = new Intent( Intent.ACTION_CALL, Uri.parse(url) );  
startActivity(intent);
```

- **Ouvrir un navigateur**

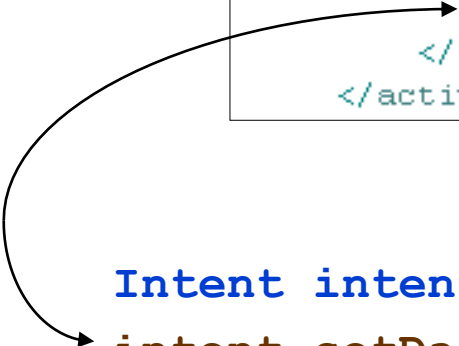
```
Intent intent = new Intent(Intent.ACTION_VIEW);
```

```
intent.setData(Uri.parse("http://www.hello.com"));
```

```
startActivity(intent);
```

Le scheme hello:// devient un critère

```
<activity android:name=".HelloWorldActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="bonjour.ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="hello" />
    </intent-filter>
</activity>
```



```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("hello://www.world.com"));
startActivity(intent);
```

Composants logiciels, enfin ?

Serait-ce une « nouvelle » approche de conception des logiciels ?

Une application est constituée de composants logiciels,

Ces composants sont indépendants, interchangeables, ont leur contexte d'exécution, ... Déploiement, maintenance, ...

Ces composants sont sélectionnés à l'exécution,

Peut-on avoir une liste des applications candidates ?

Comment obtient-on les fonctionnalités de chacune ?

cf. les travaux sur service requis \subseteq service fourni ?

Mécanisme de résolution

- Quelles sont les activités sensibles à cette *Intention* ?
 - Calcul de l'éligibilité des activités, (une notification conditionnée)
 - La sélection de l'activité est définie, par un IntentFilter

```
<application android:icon="@drawable/icon" android:label="@string/app_name" >

    <activity android:name=".DisBonjourALaDameActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="bonjour.ACTION" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>

</application>
```

- Sur cet exemple la résolution s'effectue en 2 étapes :
 - Test du champ *action* (pour l'activité)
 - Test du champ *category* (pour Android)
 - Si les deux réussissent alors l'activité est sélectionnée

Résolution phase 1

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING" />
    . . .
</intent-filter>
```

- Le test réussit si le champ *action* de l'Intent est inclus dans la liste
 - <http://developer.android.com/guide/components/intents-filters.html>
- phase 1 : test du champ *action*

ACTION prédéfinies

Constant	Target component	Action
ACTION_CALL	activity	Initiate a phone call.
ACTION_EDIT	activity	Display data for the user to edit.
ACTION_MAIN	activity	Start up as the initial activity of a task, with no data input and no returned output.
ACTION_SYNC	activity	Synchronize data on a server with data on the mobile device.

- <http://developer.android.com/reference/android/content/Intent.html>
- Action définies par le programmeur, une règle d'écriture
 - Le nom du paquetage + le nom de l'action

Résolution phase 2

```
<intent-filter . . . >
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    . . .
</intent-filter>
```

- Le test réussit si le champ *category* de l'Intent est inclus dans la liste
 - <http://developer.android.com/guide/components/intents-filters.html>
- phase 2 : test du champ *category*

CATEGORY prédéfinies

Constant	Meaning
CATEGORY_BROWSABLE	The target activity can be safely invoked by the browser to display data referenced by a link – for example, an image or an e-mail message.
CATEGORY_GADGET	The activity can be embedded inside of another activity that hosts gadgets.
CATEGORY_HOME	The activity displays the home screen, the first screen the user sees when the device is turned on or when the <i>Home</i> button is pressed.
CATEGORY_LAUNCHER	The activity can be the initial activity of a task and is listed in the top-level application launcher.
CATEGORY_PREFERENCE	The target activity is a preference panel.

- <http://developer.android.com/reference/android/content/Intent.html>
- **Le programmeur peut ajouter des catégories à l'Intent**
 - **addCategory()**
 - Par exemple pour demander au système toutes les applications candidates ... candidates à cet *Intention*

Résolution phase 3

```
<intent-filter . . . >
    <data android:mimeType="video/mpeg" android:scheme="http" . . . />
    <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
    . . .
</intent-filter>
```

- Le test réussit si le champ *data* de l'Intent appartient à cette liste
 - <http://developer.android.com/guide/components/intents-filters.html>
- Phase 3 : test du champ *data*
 - Le programmeur peut ajouter des *data* à l'Intent
 - **setData() setDataAndType**

Toutes les applications éligibles, précaution

Obtenir la liste de toutes les activités éligibles à une *intention* ?

Toutes non, seules celles capables de dire Bonjour !

```
PackageManager pm = getPackageManager();  
Intent intent = new Intent();  
intent.setAction("bonjour.ACTION");  
  
List<ResolveInfo> l = pm.queryIntentActivities(intent, 0);  
boolean intentSafe = l.size() > 0;  
if(intentSafe)  
    startActivity(intent)
```

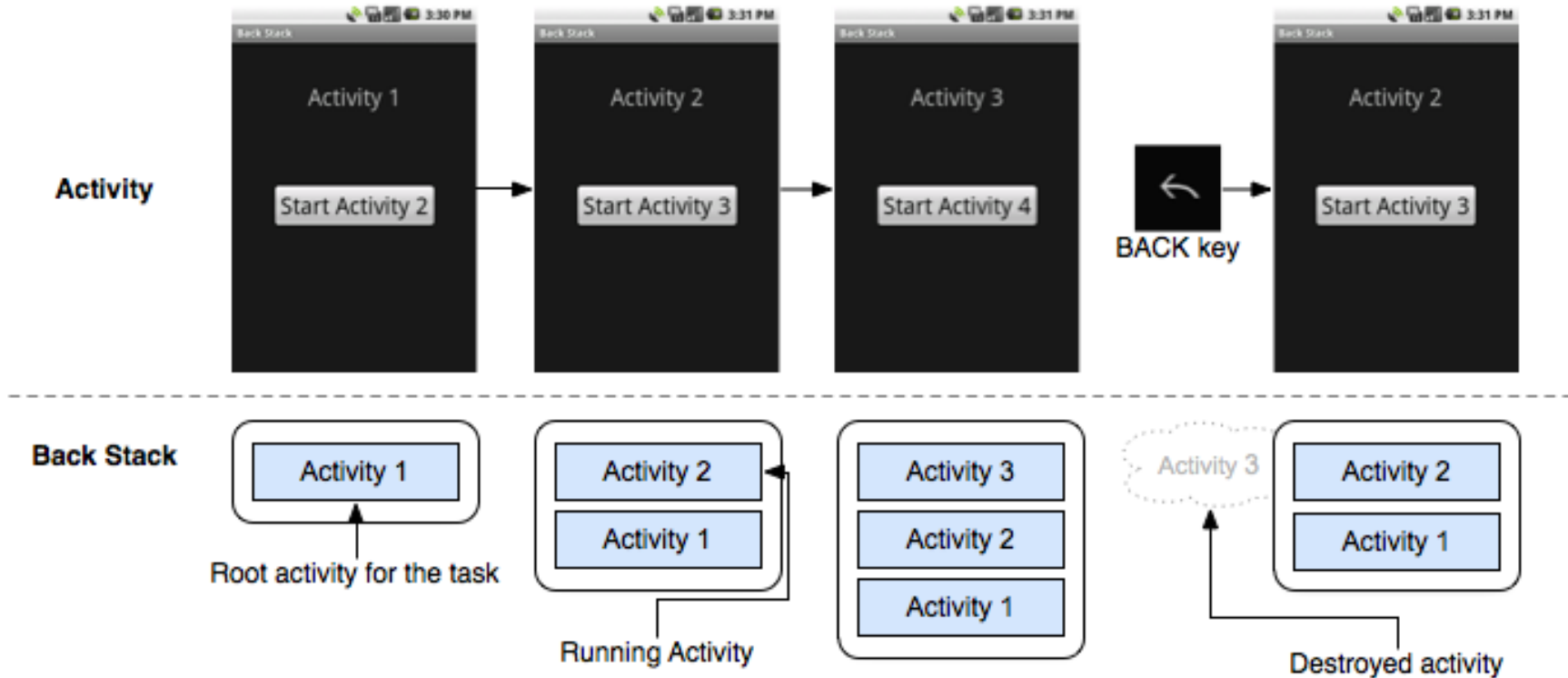
Attention émettre une intention sans qu'il y ait d'applications éligibles lève une exception

<http://developer.android.com/training/basics/intents/sending.html>

Intent, la suite

- **Un résumé**
 - Intent explicite, implicite, processus
- **Un rappel sur la pile des activités**
- **Architecture : Le pattern Publish-Subscribe**
- **Passage de paramètres, via l'intent**
 - (ce n'est pas la seule façon...)
- **Retour de résultats**
 - Comment ?
 - Redéfinition de `startActivityForResult`

Rappel : Une pile des activités est en place



- **Pouvons nous modifier le contenu de cette Pile ?**

- startActivity et les FLAG_ACTIVITY

- Méthode setFlags ou addFlags d'une Intent

- Intent intent = new Intent(ctxt, Activity2.class);

- Intent.setFlags(Intent.FLAG_ACTIVITY...

Pile des activités... cf FLAG_ACTIVITY_...

- A,B,C,D sont des activités,

- A démarre B (A -> B, par startActivity)

- A -> B -> C -> D

- A la fin de D Nous voudrions directement revenir en A

- Depuis D

- ```
Intent intent = new Intent(this, A.class);
Intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
```

- A -> B -> A -> B -> A ...

- Par défaut : création d'une nouvelle activité à chaque startActivity

- La même instance est conservée

- Depuis A

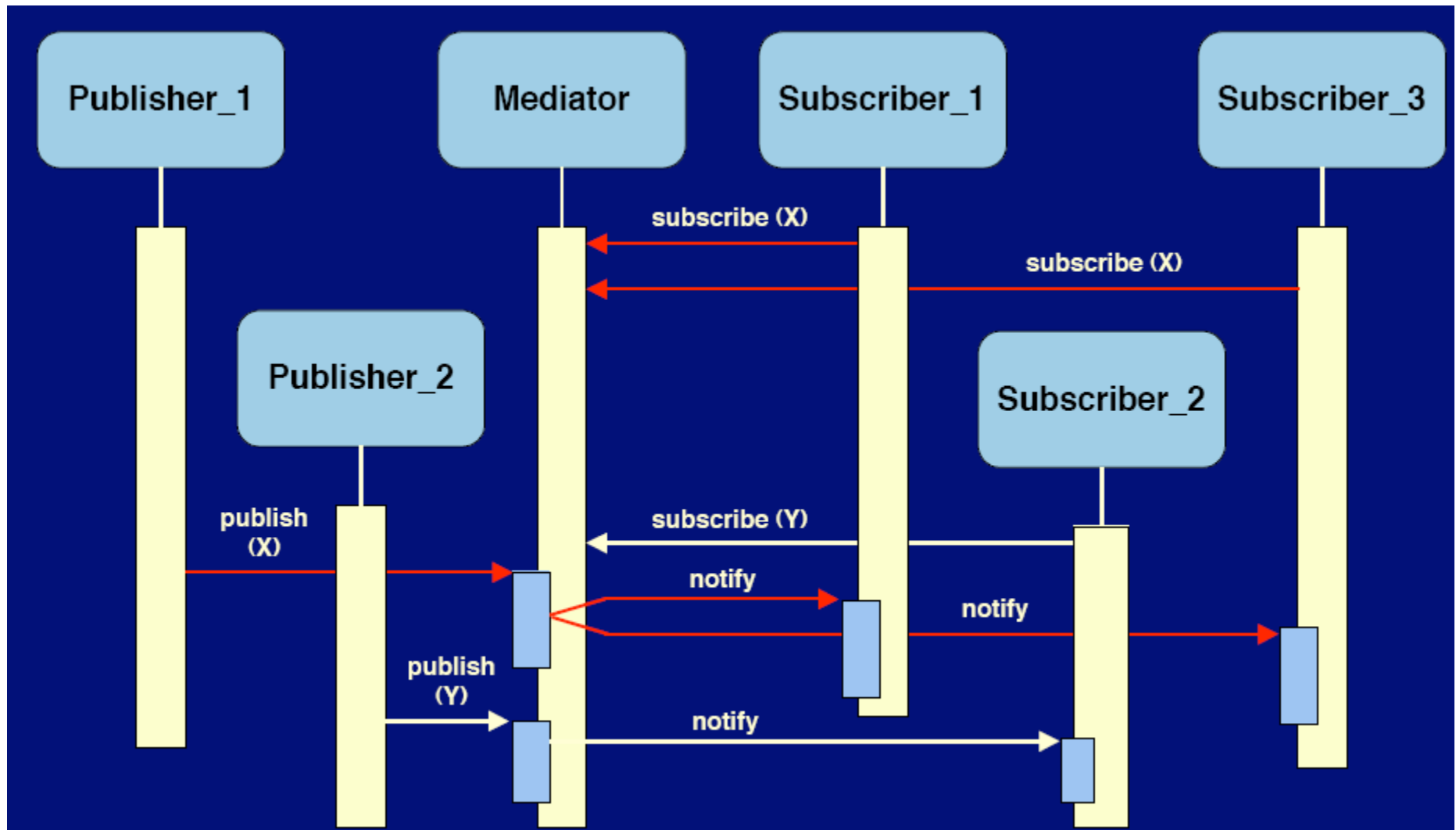
- ```
Intent intent = new Intent(this, B.class);  
Intent.addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);  
startActivity(intent);
```

- Depuis B idem

Architecture

- **Nous avons :**
 - Des souscripteurs de certaines intentions,
 - Des publieurs d'intentions,
 - Un mécanisme de résolution de la bonne activité,
 - Le système Android se charge de tout.
- **Alors serait-ce le patron Publish-Subscribe ?**

Publish-Subscribe

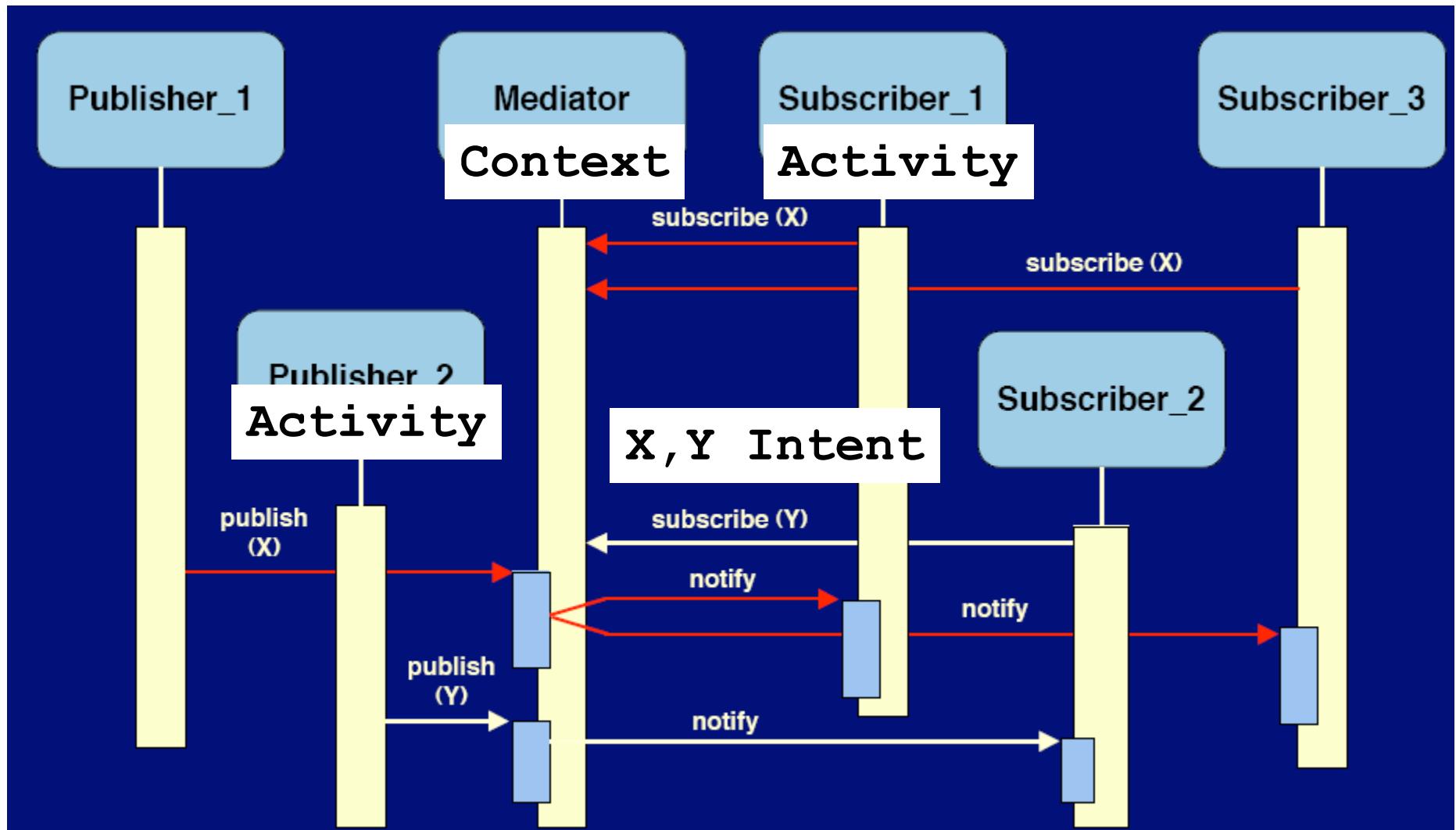


- source: <http://www2.lifl.fr/icar/Chapters/Intro/intro.html>

Publish-subscribe / pull-push

- Un forum de discussion ...
 - **Enregistrement** d'un « client » à un sujet de discussion,
 - Un des « clients » décide de poster un message,
 - Les utilisateurs à leur initiative vont chercher l'information,
 - ***Publish-subscribe, mode pull***
- Les listes de diffusion, logiciels de causerie,
 - **Abonnement** d'un « client » à une liste de diffusion,
 - Un des « clients » décide de poster un message,
 - Tous les abonnés reçoivent ce message,
 - Les abonnés peuvent installer un **filtre** sur les contenus des messages,
 - ***Publish-subscribe, mode push***

Publish-Subscribe/Intent & Context



- Activity, Service, Receiver même principe ...

Les classes au sein de ce patron

- **Basé sur les**
 - Intent (**Topic**),
 - Context (**Mediator**),
 - Activity (**Subscriber**, **Publisher**).
- **Publisher**
 - Intent i = new ...
 - startActivity(context, i);
- **Subscriber**
 - IntentFilter
- **Mediator**
 - Context
 - La notification utilise un mécanisme de résolution
 - En fonction des filtres installés par les souscripteurs

Sommaire-suite

- **Activity**
 - Présentation, cycle de vie
- **Intent, intentFilter**
 - Démarrage d'une autre activité
 - Au sein de la même application
 - Depuis une autre application,
 - Passage de paramètres
 - Un résultat produit par l'activité est attendu
 - Attente du résultat,
- Communiquer entre deux activités
 - Plusieurs façons
- Permission
- Génie logiciel, conception
 - Approche par composants Logiciels induite, comme méthode de conception ?
- Annexes

L'intent peut contenir des paramètres

- **Les extras, un *Bundle*, une *Map* !**

Une table de couples <clé, valeur>, la clé est de type String

```
Intent i = new Intent();  
// i.setAction...  
i.putExtra("fichier", "hello.mp3");  
i.putExtra("compteur", 2);
```

Intent	<code>putExtra (String name, double[] value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, int value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, CharSequence value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, char value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, Bundle value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, Parcelable[] value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, Serializable value)</code> Add extended data to the intent.

Des paramètres à l'intention de

L'activité lit les paramètres transmis

- **Les extras, un *Bundle*, une *Map* !**

Une table de couples <clé, valeur>, la clé est de type *String*

```
Intent i = getIntent();  
String f = i.getStringExtra("fichier");
```

double	<code>getDoubleExtra (String name, double defaultValue)</code> Retrieve extended data from the intent.
Bundle	<code>getExtras ()</code> Retrieves a map of extended data from the intent.
int	<code>getFlags ()</code> Retrieve any special flags associated with this intent.
float[]	<code>getFloatArrayExtra (String name)</code> Retrieve extended data from the intent.
float	<code>getFloatExtra (String name, float defaultValue)</code> Retrieve extended data from the intent.
int[]	<code>getIntArrayExtra (String name)</code> Retrieve extended data from the intent.
int	<code>getIntExtra (String name, int defaultValue)</code> Retrieve extended data from the intent.

Des paramètres reçus par l'activité sélectionnée

putExtra,

getExtras

Intent	<code>putExtra (String name, Bundle value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, Parcelable[] value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, Serializable value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, int[] value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, float value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, byte[] value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, long[] value)</code> Add extended data to the intent.
Intent	<code>putExtra (String name, Parcelable value)</code> Add extended data to the intent.

Bundle	<code>getExtras ()</code> Retrieves a map of extended data from the intent.
int	<code>getFlags ()</code> Retrieve any special flags associated with this intent.
float[]	<code>getFloatArrayExtra (String name)</code> Retrieve extended data from the intent.
float	<code>getFloatExtra (String name, float defaultValue)</code> Retrieve extended data from the intent.
int[]	<code>getIntArrayExtra (String name)</code> Retrieve extended data from the intent.
int	<code>getIntExtra (String name, int defaultValue)</code> Retrieve extended data from the intent.
<code>ArrayList<Integer></code>	<code>getIntegerArrayListExtra (String name)</code> Retrieve extended data from the intent.

<http://developer.android.com/reference/android/content/Intent.html>

Types simples : c'est prévu

Objets métiers : ils **implémenteront** Parcelable

L'activité transmet une instance complète : Parcelable

- La classe des instances transmises implémente Parcelable
- Parcelable comme Serializable
 - **Sauf** que tout est à la charge du programmeur
 - Chaque champ doit être écrit et restitué
- Exemple :
 - Auditeur est une classe qui implémente Parcelable
 - Les noms de méthodes sont imposées via l'interface
- Envoi depuis l'activité a1

```
Auditeur unAuditeur = new Auditeur("alfred");  
intent.putExtra("auditeur", unAuditeur);  
startActivity(intent);
```

- Appel
 - du constructeur
 - Lors de la transmission
 - writeToParcel

```
public class Auditeur implements Parcelable {  
    private String nom;  
    private long id;  
  
    public Auditeur(String nom) {  
        this.nom = nom;  
        this.id = globalId++;  
    }  
  
    @Override  
    public int describeContents() {  
        return 0;  
    }  
  
    @Override  
    public void writeToParcel(Parcel dest, int flags) {  
        dest.writeString(this.nom);  
        dest.writeLong(this.id);  
    }  
}
```

L'activité reçoit une instance Parcelable

- **Restitution :**
 - **Classe Auditeur suite**

```
public static final Parcelable.Creator<Auditeur> CREATOR
    = new Parcelable.Creator<Auditeur>() {
    public Auditeur createFromParcel(Parcel in) {
        return new Auditeur(in);
    }

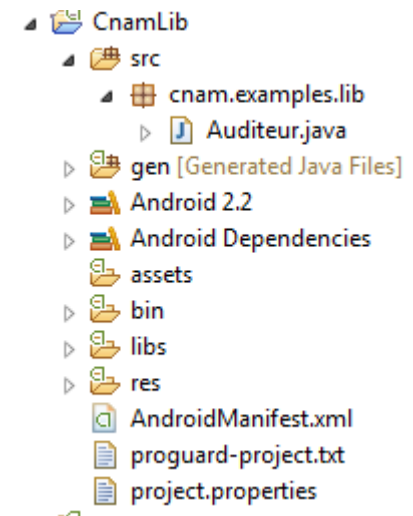
    public Auditeur[] newArray(int size) {
        return new Auditeur[size];
    }
};

private Auditeur(Parcel in) {
    this.nom = in.readString();
    this.id = in.readLong();
}
```

- Les noms de méthodes sont imposées via l'interface
- Un constructeur avec en paramètre l'instance Parcel reçue
 - Déclenché par le mécanisme de restitution
- La classe *Auditeur* se doit d'être dans une librairie
 - (afin d'éviter la recopie multiple des sources)

Une librairie, un .jar

- **Connaissance des classes par les deux activités ?**
 - Toutes ces classes sont dans un projet eclipse : case isLibrary à cocher
 - Une librairie, un .jar à installer dans chaque application cliente
 - CnamLib contient la classe Auditeur ci-dessous



Serializable / Parcelable, discussions

- **Serializable**

- Puissant mais lent, dû à l'usage de l'introspection
- *Comically slow ...*

- **Parcelable**

- Pas d'introspection, tout est défini par l'utilisateur
- *Fanastically quick ...*

- <http://stackoverflow.com/questions/5550670/benefit-of-using-parcelable-instead-of-serializing-object>



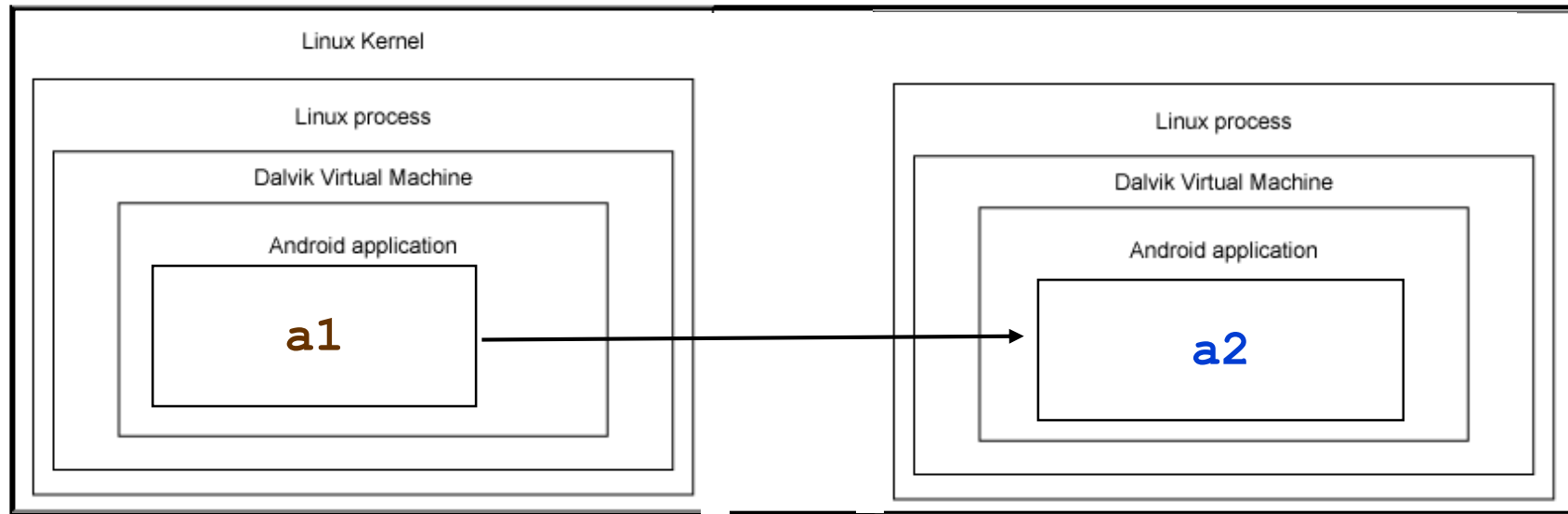
`Serializable` is comically slow on Android. Borderline useless in many cases in fact.

`Parcel` and `Parcelable` are fantastically quick, but its documentation says you must not use it for general-purpose serialization to storage, since the implementation varies with different versions of Android (i.e. an OS update could break an app which relied on it).

- **Benchmark intéressant ...**

- <http://code.google.com/p/thrift-protobuf-compare/wiki/Benchmarking>
 - Java Serializable, Externalizable, JSON ...

Retour de résultats



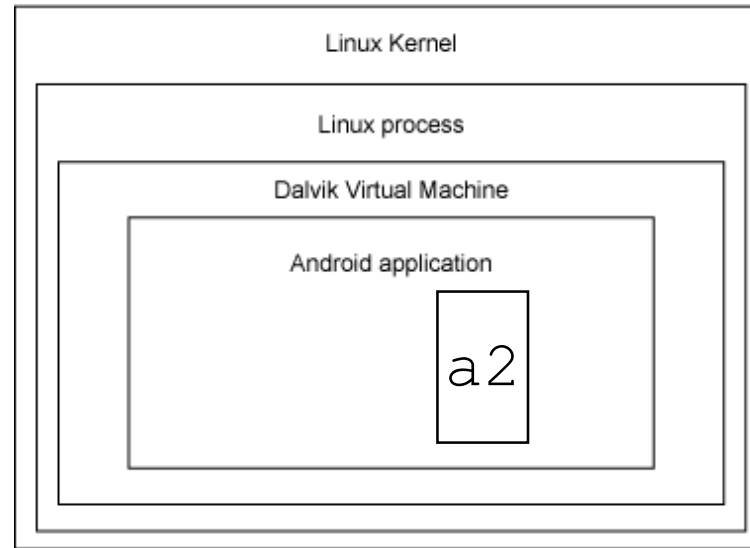
a1) `startActivityForResult(intent, requestCode);`

→

- **Un résultat est attendu**
 - une **intention** comme d'habitude ...
 - un paramètre `requestCode`,
 - un identifiant de l'activité appelante (a1)

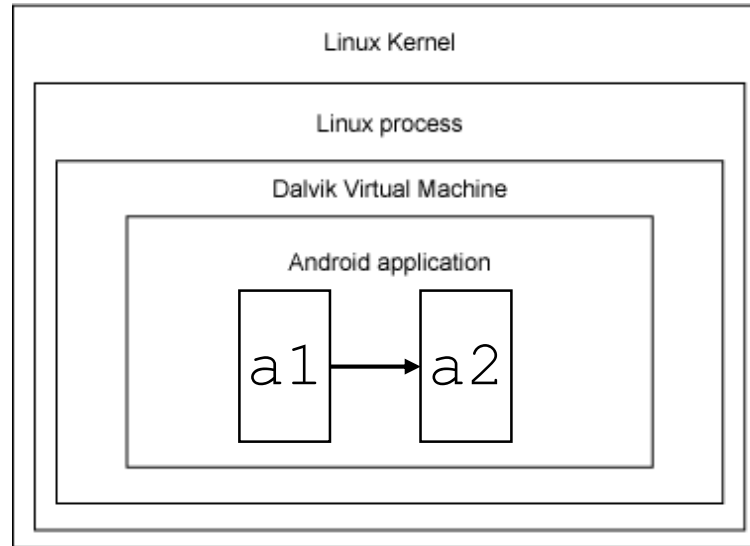
`onActivityResult` est redéfinie dans l'activité appelante (a1)

L'activité a2 est déclenchée, et s'exécute



- **Tout s'est bien passé**
 - `setResult(RESULT_OK);`
- **Tout s'est bien passé et en plus il y a des résultats retournés**
 - `setResult(RESULT_OK, intent);`
- **a2 se termine cf cycle de vie**

a2 se termine



onActivityResult redéfinie dans **a1** est appelée,
cette méthode s'exécutera lorsque **a1** sera au 1er plan

Exemple HelloMaster -> HelloActivity

- **HelloMaster déclenche HelloActivity**
 - Le dialogue s'installe « Bonjour ! » -> « Bonjour, comment allez-vous ? »
 - Reprenons l'exemple des deux activités une application, deux DVM

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent intent = new Intent();
    intent.setClassName(this, "cnam.essais.HelloWorldActivity");

    final int AVEC_UNE_REPONSE = 3;
    startActivityForResult(intent, AVEC_UNE_REPONSE);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode == RESULT_OK && data != null) {
        String reponse = data.getStringExtra("reponse");
        Log.i("HelloWorldMasterActivity", reponse);
    } else {
        Log.i("HelloWorldMasterActivity", "aucune reponse !, des soucis ?");
    }
}
```

HelloActivity répond (aujourd'hui)

```
@Override
public void onResume() {
    super.onResume();
    Intent i = new Intent();
    i.putExtra("reponse", "Bonjour, comment-allez vous ?");
    setResult(RESULT_OK, i);
}
```

- **setResult est appelée**
- **Puis touche retour**

Exemple HelloActivity -> HelloMaster

- **HelloMaster reçoit une réponse**
 - **onActivityResult** est appelée

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent intent = new Intent();
    intent.setClassName(this, "cnam.essais.HelloWorldActivity");

    final int AVEC_UNE_REPONSE = 3;
    startActivityForResult(intent, AVEC_UNE_REPONSE);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode == RESULT_OK && data != null) {
        String reponse = data.getStringExtra("reponse");
        Log.i("HelloWorldMasterActivity", reponse);
    } else {
        Log.i("HelloWorldMasterActivity", "aucune reponse !, des soucis ?");
    }
}
```


Séquencement des deux activités

- **HelloWorldMaster**

1. onCreate

- startActivity ...

2. onStart

3. onResume

4. onPause

- Activity en arrière plan

8. onStop

10. onActivityResult

11. onRestart

12. onStart

13. onResume

- **HelloWorldActivity**

5. onCreate

6. onStart

7. onResume

- Activity visible

- setResult

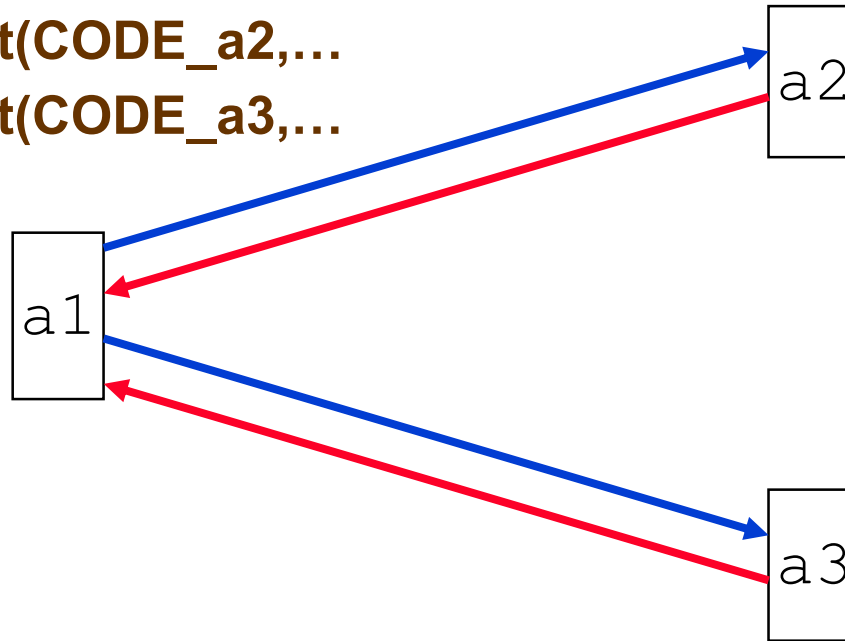
- « Touche retour »

9. onPause

14. onStop

Deux activités fournissent des résultats

startActivityForResult(CODE_a2,...
startActivityForResult(CODE_a3,...



- **a2**
- **setResult**

public void onActivityResult(**CODE**

if (CODE==CODE_a2)

...

else if (CODE==CODE_a3)

...

- **a3**
- **setResult**

onActivityResult, schéma de programme

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
    switch(requestCode){
        case LOGIN_REQUEST_CODE :
            switch(resultCode){
                case RESULT_OK :
                    String result1 = data.getStringExtra("key1");
                    String result2 = data.getStringExtra("key2");
                    Toast.makeText(this, "<" + result1 + ", " + result2 + ">", Toast.
                        break;
                case RESULT_CANCELED :
                    Toast.makeText(this, "canceled", Toast.LENGTH_LONG).show();
                    break;
            }
        case ANOTHER_REQUEST_CODE :
            switch(resultCode){
                case RESULT_OK :
                    // ...
                    break;
                case RESULT_CANCELED :
                    // ...
                    break;
            }
        default: super.onActivityResult(requestCode, resultCode, data);
    }
}
```

- **requestCode** identifie l'activité appelée
 - Sur cet exemple **LOGIN_REQUEST_CODE**
 - i.e. `startActivityForResult(intent, LOGIN_REQUEST_CODE);`
 - Ou `startActivityForResult(intent, ANOTHER_REQUEST_CODE);`

Passage de paramètres, suite

- **Passage de paramètres, via l'intent**

Mais ce n'est pas la seule façon

<http://developer.android.com/guide/faq/framework.html#3>

<http://stackoverflow.com/questions/4878159/android-whats-the-best-way-to-share-data-between-activities>

<http://developer.android.com/guide/practices/performance.html>

- **L'activité déclenchée partage la même application**

- *Une instance d'une classe, pattern Singleton, cf annexe*
- *Une variable de classe et ses méthodes, cf annexe*
- *Une HashMap de WeakReference, la clé est transmise via l'intent cf annexe*
- **Via l'application.**

- **L'activité est sur une autre DVM**

- **Intent,**
- *Messenger (autre support, cf; service)*

- **Avec persistance**

- **Préférences,**
- *Fichiers, (java.io)*
- *contentProviders, SQLite(autre support)*

Paramètres : via l'application

```
import android.app.Application;

public class Global<T> extends Application {
    private List<T> value;

    public synchronized void setValue(List<T> liste){
        value = liste;
    }

    public synchronized List<T> getValue(){
        return value;
    }
}
```

- **AndroidManifest.xml**

```
<application android:icon="@drawable/icon" android:label="@string/app_name"
    android:name="cnam.essais.Global">
```

- **Même Application, même DVM**

- Chaque application Android peut être associée à une instance de **Android.app.Application**
- Un seul point d'entrée pour toutes les activités
 - Peut-être le plus COO voir annexe, cf pattern délégation et façade

Paramètres : via l'Application

- **Ecriture**

```
List<String> l = new ArrayList<String>();  
l.add("autre"); l.add("passage"); l.add("de"); l.add("parametre");  
Global gs = (Global)getApplication();  
gs.setValue(l);
```

- **Lecture**

```
Global gs = (Global)getApplication();  
List<String> l = gs.getValue();  
  
l.add("avec"); l.add("un"); l.add("Application_memeDVM");
```

- **Attention même DVM ...**

Passage de paramètres avec persistance

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    SharedPreferences prefs = this.getSharedPreferences("prefs", MODE_WORLD_READABLE);
    SharedPreferences.Editor prefsEdit = prefs.edit();
    prefsEdit.putString("message", "hello !");
    prefsEdit.commit();

    Intent intent = new Intent();
    intent.setClassName(this, "cnam.essais.HelloWorldActivity");
    startActivityForResult(intent, 33);
}
```

- Un fichier xml est généré,
 - ici accessible et modifiable par tous

```
D:\android-sdk-windows\platform-tools>adb shell
$ cd /data/data/cnam.essais/shared_prefs/
$ cd /data/data/cnam.essais/shared_prefs/
$ su
su
# ls
ls
prefs.xml
# cat prefs.xml
cat prefs.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="reponse">Bonjour, comment-allez vous ?</string>
<string name="message">hello !</string>
</map>
#
```

Persistence, lecture

```
SharedPreferences prefs = this.getSharedPreferences("prefs", MODE_WORLD_READABLE);  
String reponse = prefs.getString("reponse", "aucune reponse !, des soucis ?");
```

- **Lecture du fichier de préférences**

Sommaire

- **Activity**
 - Présentation, cycle de vie
- **Intent, intentFilter**
 - Démarrage d'une autre activité
 - Au sein de la même application
 - Depuis une autre application,
 - Passage de paramètres
 - Un résultat produit par l'activité est attendu
 - Attente du résultat,
- **Communiquer entre deux activités**
 - Plusieurs façons
- **Permission**
- **Génie logiciel, conception**
 - Approche par composants Logiciels induite, comme méthode de conception ?
- **Annexes**

Les Permissions

- Une activité peut
 - **exiger** d'avoir la bonne permission afin d'être appelée
 - La clause **<use-permission** est installée chez l'appelant
 - Le fichier AndroidManifest.xml de l'appelé contient
 - Les déclarations des permissions
 - <permission**
 - L'attribut **android:permission** , balise **<activity**

Permission d'être poli

- La permission de dire bonjour ...

Les permissions sont installées, AndroidManifest.xml

<permission

<activity android:permission=

```
<permission android:name="cnam.essais.HELLO_WORLD"
            android:permissionGroup="android.permission-group.PERSONAL_INFO"
            android:protectionLevel="dangerous" />

<application android:icon="@drawable/icon" android:label="@string/app_name" >

    <activity android:name=".HelloWorldActivity"
            android:label="@string/app_name"
            android:permission="cnam.essais.HELLO_WORLD">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

Si j'ai la permission

- Si j'ai la permission, je pourrais démarrer l'activité
 - `<use-permission`

```
<uses-sdk android:minSdkVersion="8" />
<uses-permission android:name="cnam.essais.HELLO_WORLD" />

<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".HelloWorldMasterActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

- **Autre support en cours ...**
- **Design**
 - <http://developer.android.com/design/index.html>
 - <http://mobile.smashingmagazine.com/2012/07/26/android-design-tips/>

- **Passage de paramètres, même DVM**
 - **L'activité déclenchée partage la même application**
 - *Une instance d'une classe, pattern Singleton,*
 - *Une variable de classe et ses méthodes,*
 - *Une HashMap de WeakReference, la clé est transmise via l'intent*

Passage de paramètres, même DVM

- **Passage de paramètres, via l'intent**

Ce n'est pas la seule façon

<http://developer.android.com/guide/faq/framework.html#3>

<http://stackoverflow.com/questions/4878159/android-whats-the-best-way-to-share-data-between-activities>

<http://developer.android.com/guide/practices/performance.html>

Paramètres : le pattern Singleton

```
public class MasterActivity extends Activity {

    public static final class Singleton{
        private static List<?> instance=null;
        public static <T> List<T> getInstance(){
            if(instance==null)
                instance = new ArrayList<T>();
            return (List<T>)instance;
        }
        private Singleton(){}
    }

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        List<String> l = Singleton.getInstance();
        l.add("autre"); l.add("passage");l.add("de");l.add("parametre");

        Intent intent = new Intent();
        intent.setClassName(this,"cnam.essais.HelloWorldActivity");
        startActivity(intent);
    }
}
```

- **Attention : même DVM, pour les deux activités**
 - Variables de classes ...

Paramètres : le pattern Singleton, lecture

```
public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main2);

        List<String> l = Singleton.getInstance();
        l.add("avec"); l.add("un"); l.add("singleton_memeDVM");
    }
}
```

- **Même DVM, pour les deux activités**
- **Le singleton est ici en mémoire « globale »**
- **Autre technique : Variables et méthodes de classes, idem**
 - **Attention tout de même à cet usage, ce n'est pas de la COO (Thread safe, ...**

Paramètres: WeakHashMap

```
public static Map<Long, Object> parameters = new WeakHashMap<Long, Object>();

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    List<String> l = new ArrayList<String>();
    l.add("autre"); l.add("passage"); l.add("de"); l.add("parametre");
    long clef = SystemClock.elapsedRealtime(); // cf. WeakHashMap et le GC
    parameters.put(clef, l);

    Intent intent = new Intent();
    intent.putExtra("LISTE", clef);
    intent.setClassName(this, "cnam.essais.HelloWorldActivity");
    startActivity(intent);
}
```

- **WeakHashMap**

- Une nouvelle clé est générée,
- La clef référence une instance côté activité cliente,
- Cette clef est transmise à l'activité appelée via l'intent,
- **Weak** : Le ramasse miettes(GC) libère tous les couples <clef, valeur>, dont la clef n'est plus référencée par le programme

Paramètres : WeakHashMap, lecture

```
public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main2);

        //      List<String> l = Singleton.getInstance();
        long clef = getIntent().getLongExtra("LISTE",-1);
        List<String> l = (List<String>)parameters.get(clef);
        l.add("avec");l.add("un");l.add("WeakHashMap_memeDVM");

    }
}
```

- Le client lit la référence de la liste depuis la table
- Même DVM, pour les deux activités