Programming with Android: Activities and Fragments

Federico Montori

Dipartimento di Informatica: Scienza e Ingegneria

Università di Bologna

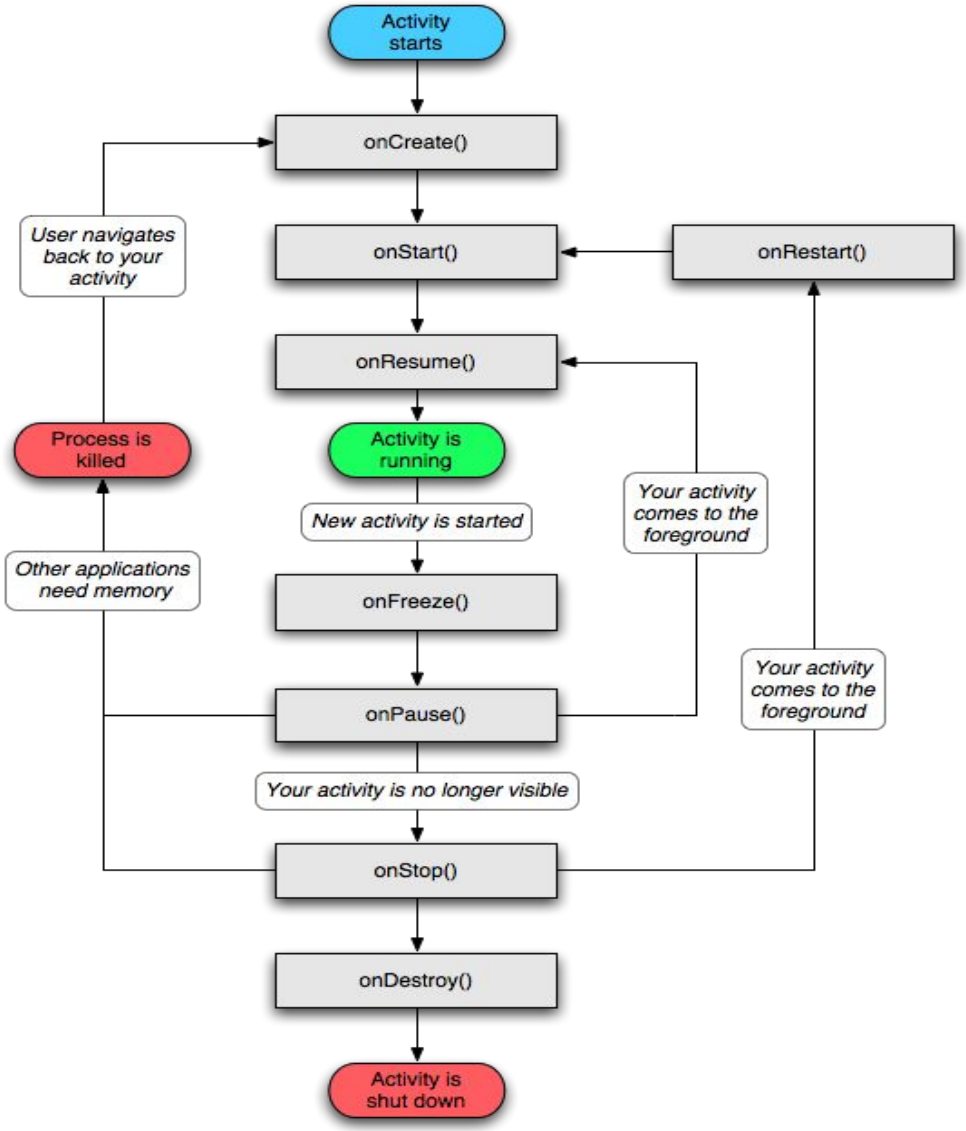
Outline

- Activities overview
- Activities Lifecycle
- Fragments Overview
- Fragments Handling
- Fragments Transitions

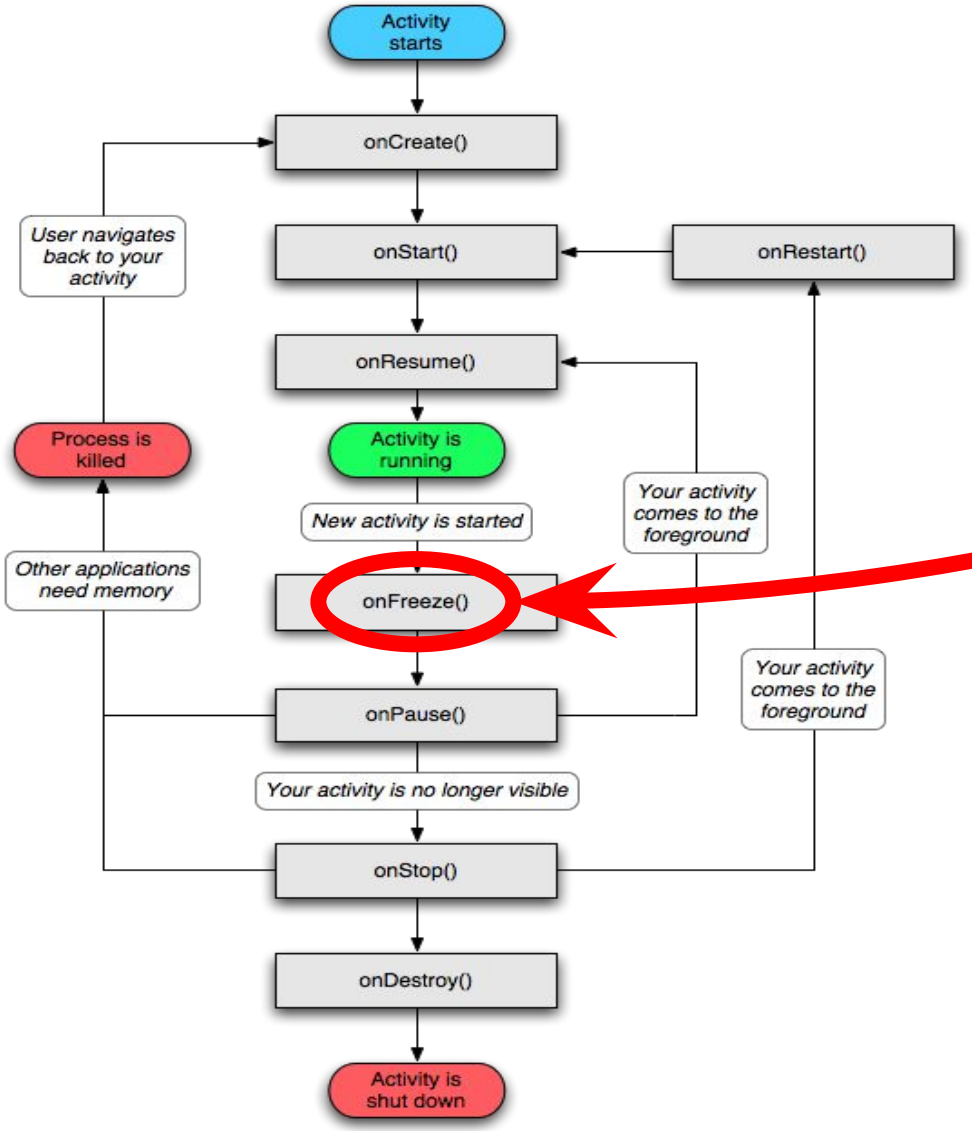
Activity

- What is started by the device
- It contains the application's informations
- Has methods to react to certain events
- An application can be composed of multiple activities
- We call activity a screen state
- Android maintains a **stack** of activities

Activity lifecycle

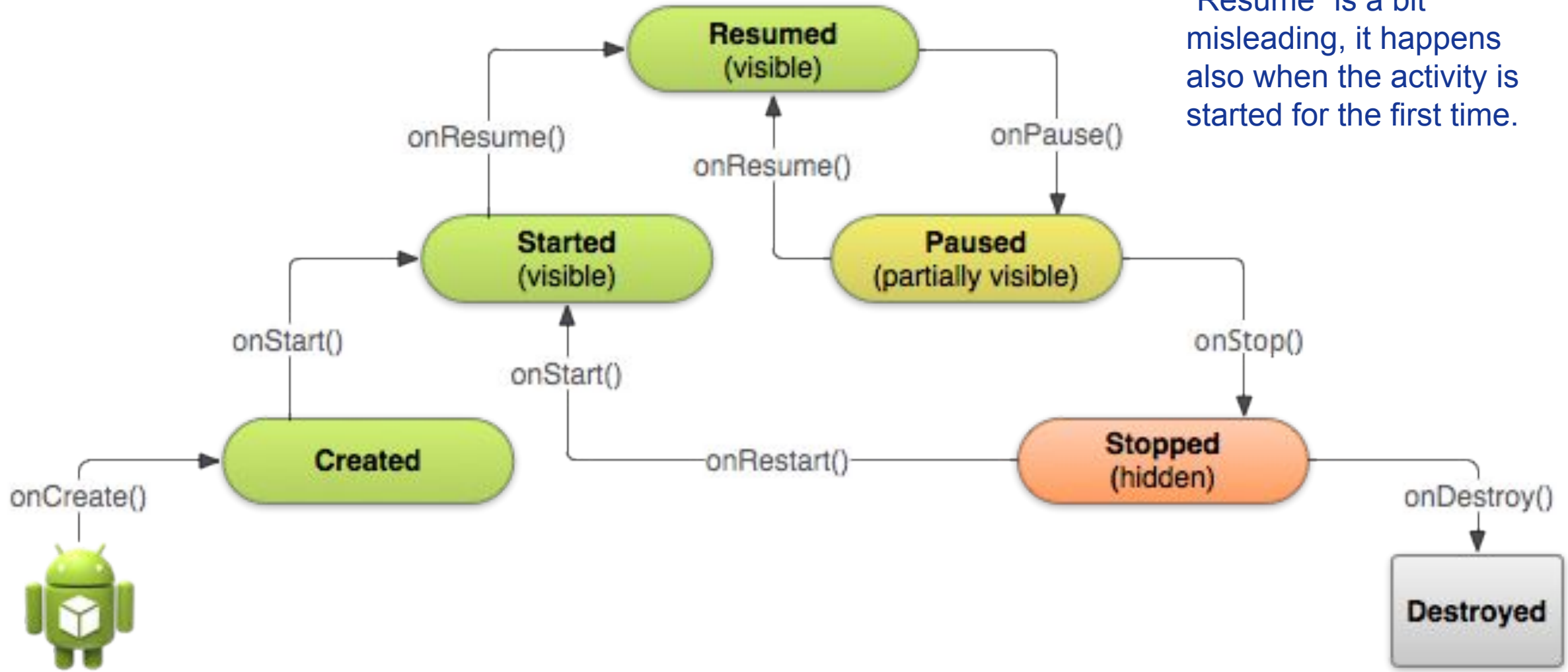


Activity lifecycle



onFreeze() is not used anymore since 2008. It is useful to know it was there, but it has been replaced by other functionalities.

Activity lifecycle



"Resume" is a bit misleading, it happens also when the activity is started for the first time.

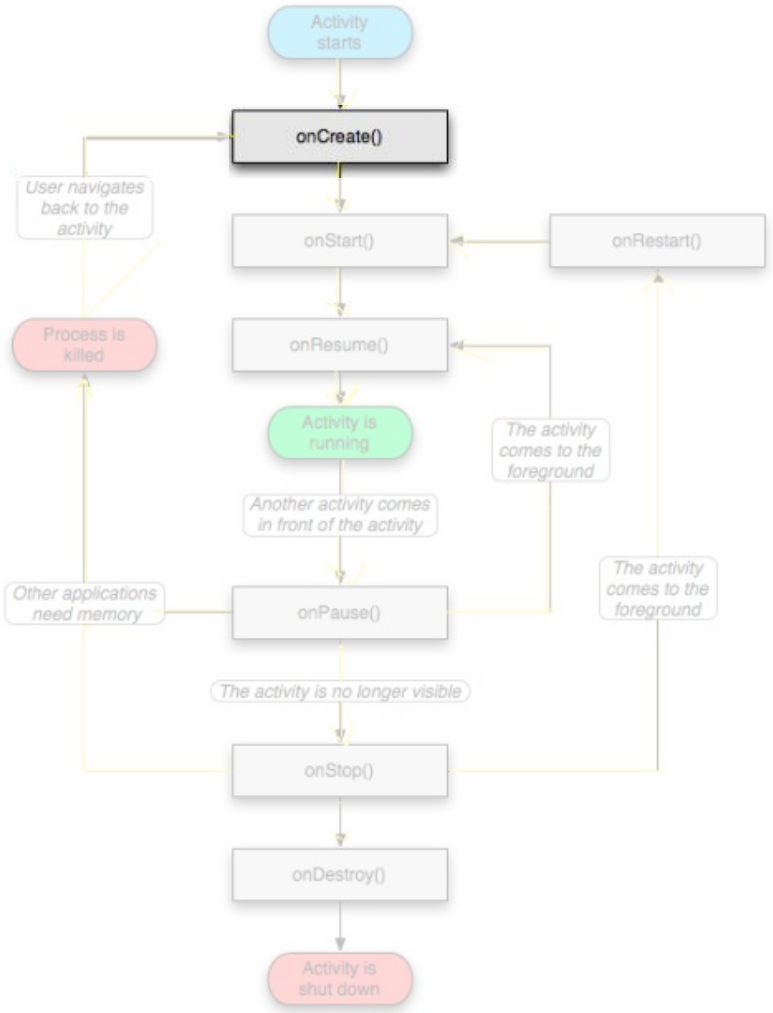
Activities

- Need to implement every single method? No!
  - It depends on the application complexity
- Why is it important to understand the activity lifecycle?
  - So your application does not crash (or do “funny” things) while the user is running something else on the smartphone
  - So your application does not consume unnecessary resources
  - So the user can safely stop your application and return to it later

Activities states

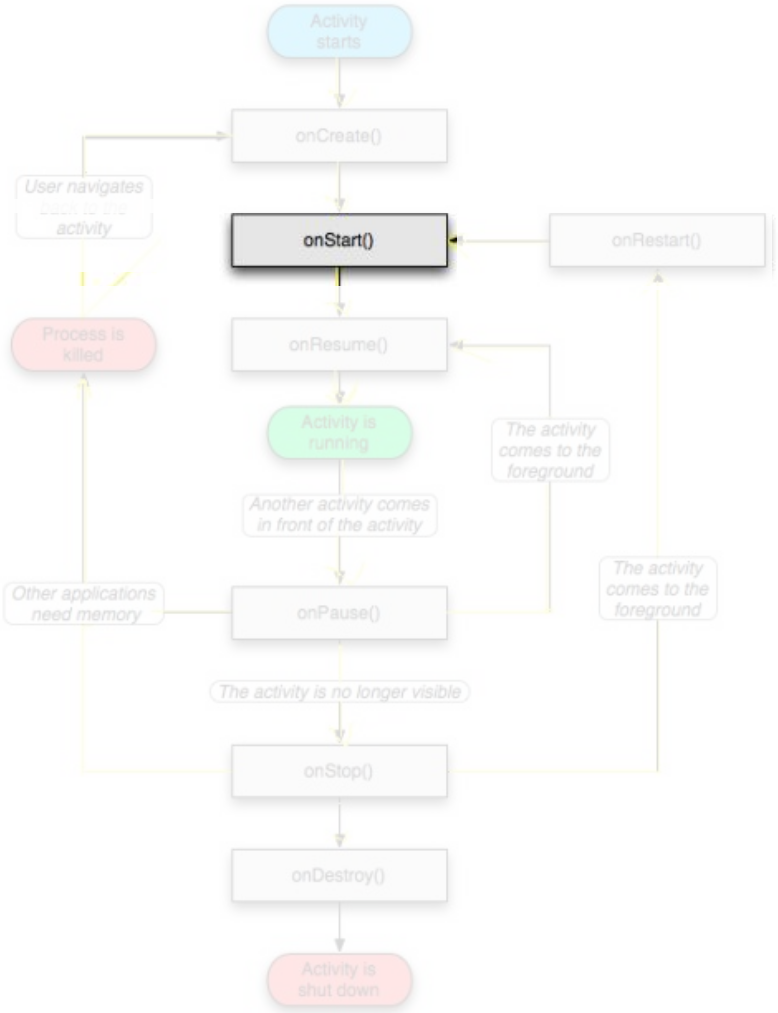
- Resumed
  - The activity is in the foreground, and the user can interact.
- Paused (but started...)
  - The activity is partially overlaid by another activity. Cannot execute any code nor receive direct inputs.
- Stopped (but created…)
  - Activity is hidden, in the background. It cannot execute any code.

Activity lifecycle



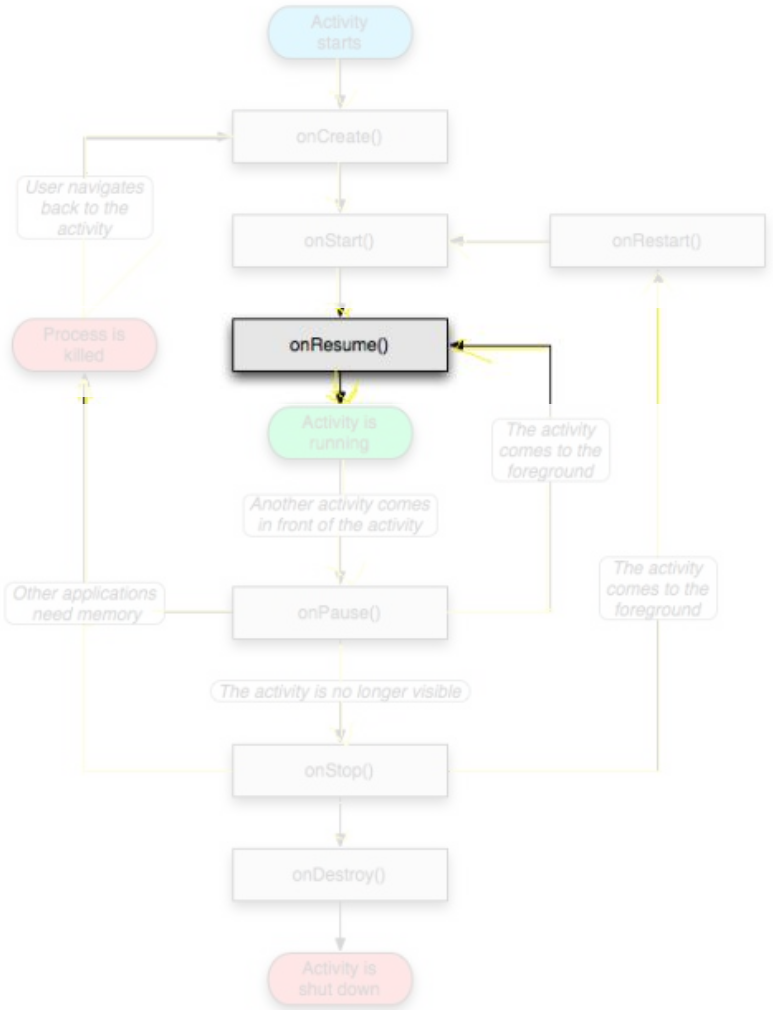
- OnCreate()
  - Called when the activity is created
  - Should contain the initialization operations
  - Has a Bundle parameter (a composite with saved data)
  - If onCreate() terminates, it calls onStart()

Activity lifecycle



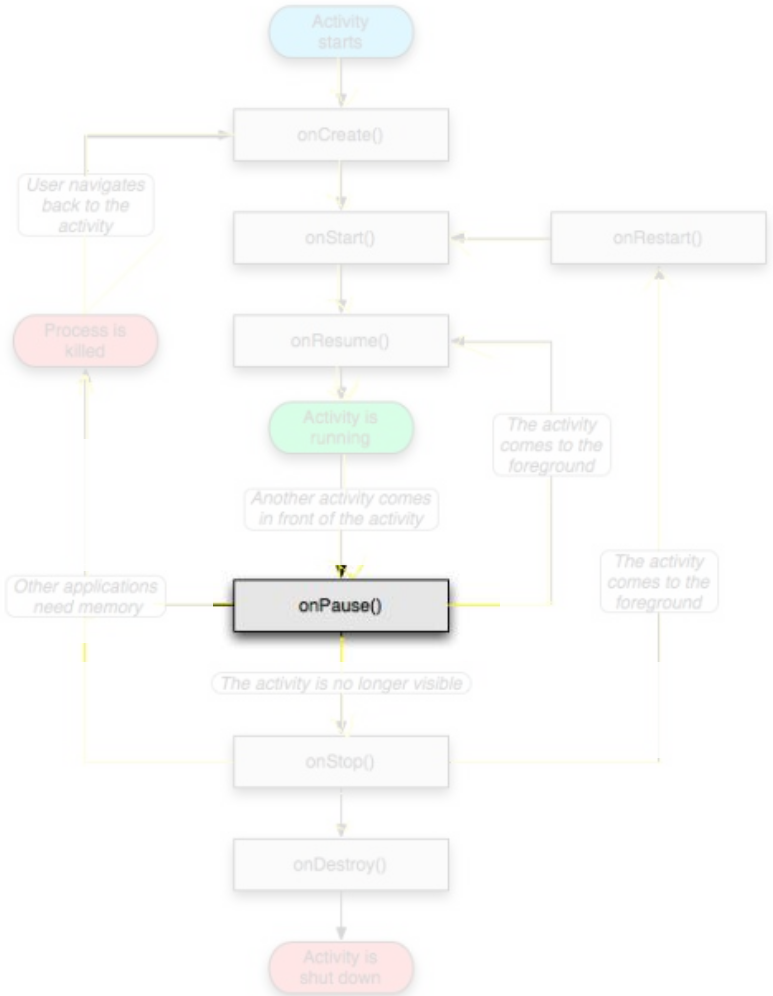
- OnStart()
  - Called when onCreate() terminates
  - Called right before it is visible to user
  - If it has the focus, then onResume() is called
  - If not, onStop() is called

Activity lifecycle



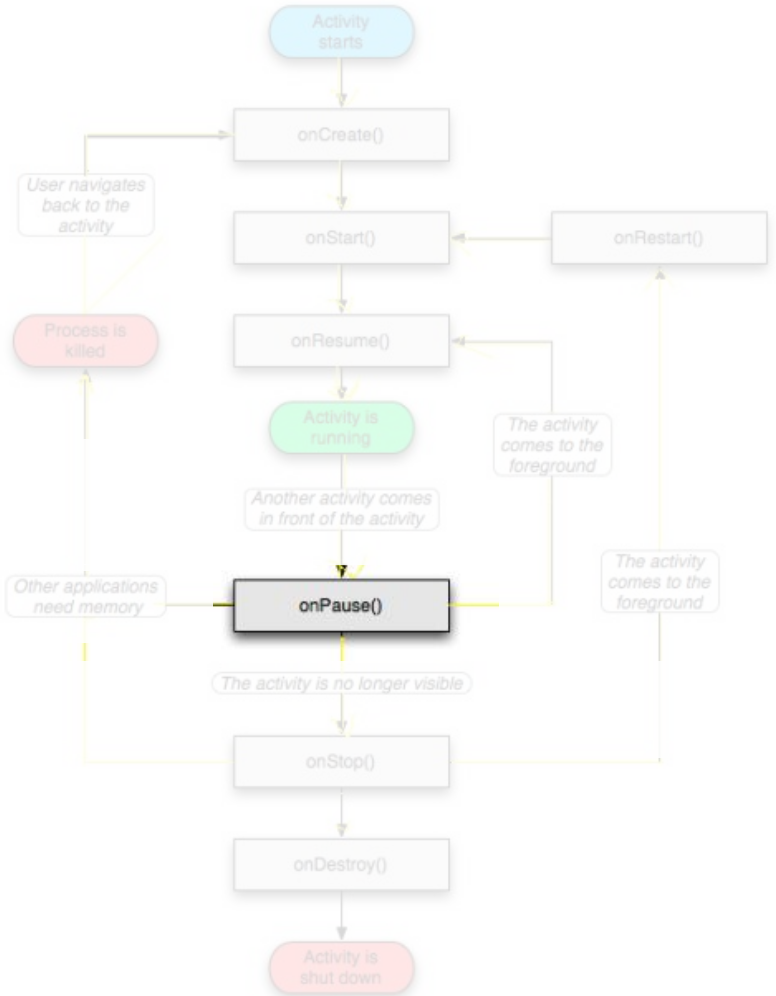
- OnResume()
  - Called when the activity is ready to get input **from users**
  - Called when the activity is resumed too (so it does not exactly mean “resume”)
  - If it successfully terminates, then the Activity is RUNNING

Activity lifecycle



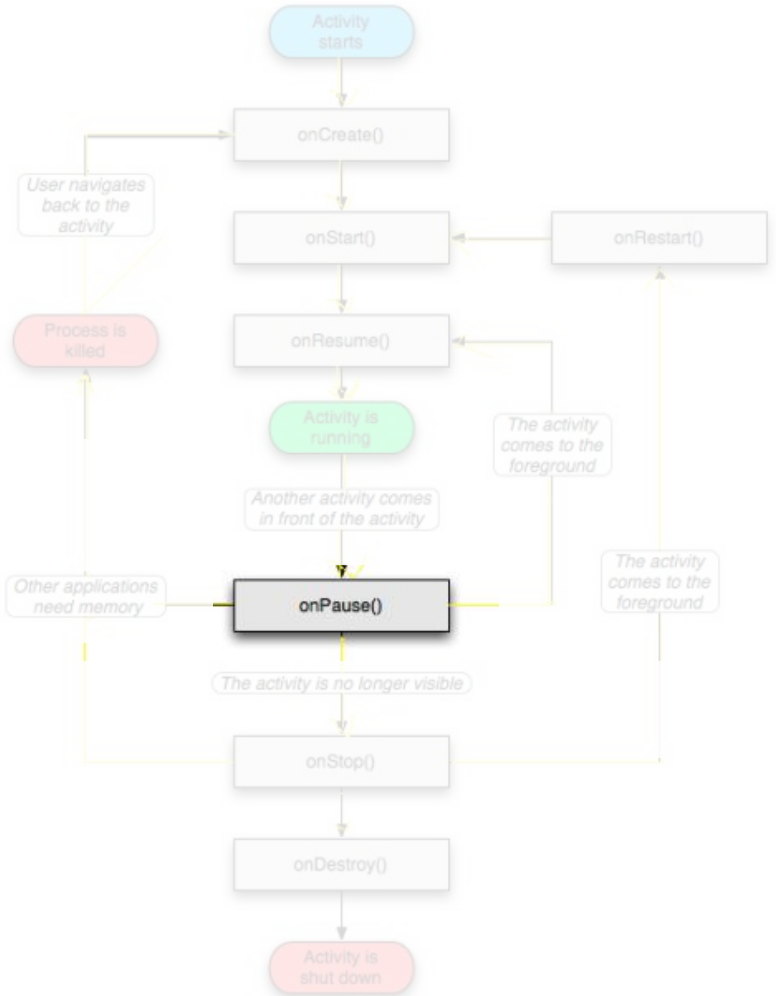
- OnPause()
  - Called when another activity comes to the foreground, or when someone presses back
  - Stop cpu-consuming processes
  - Make it fast

Activity lifecycle



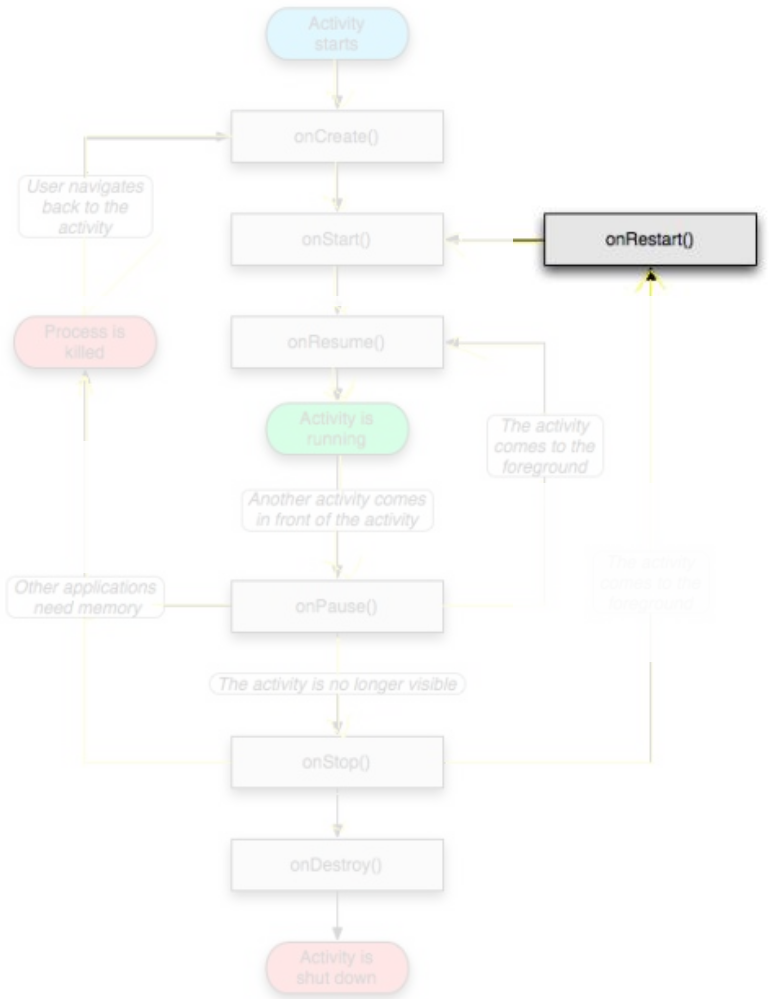
- OnPause()
  - Happens for example when:
    - Another component (NOT FROM THE SAME ACTIVITY) requests the foreground
    - A component comes in the foreground partially hiding the activity (e.g. a dialog)
    - Another window in a multi-window application is tapped.
    - Any other event that will also imply the onStop()

Activity lifecycle



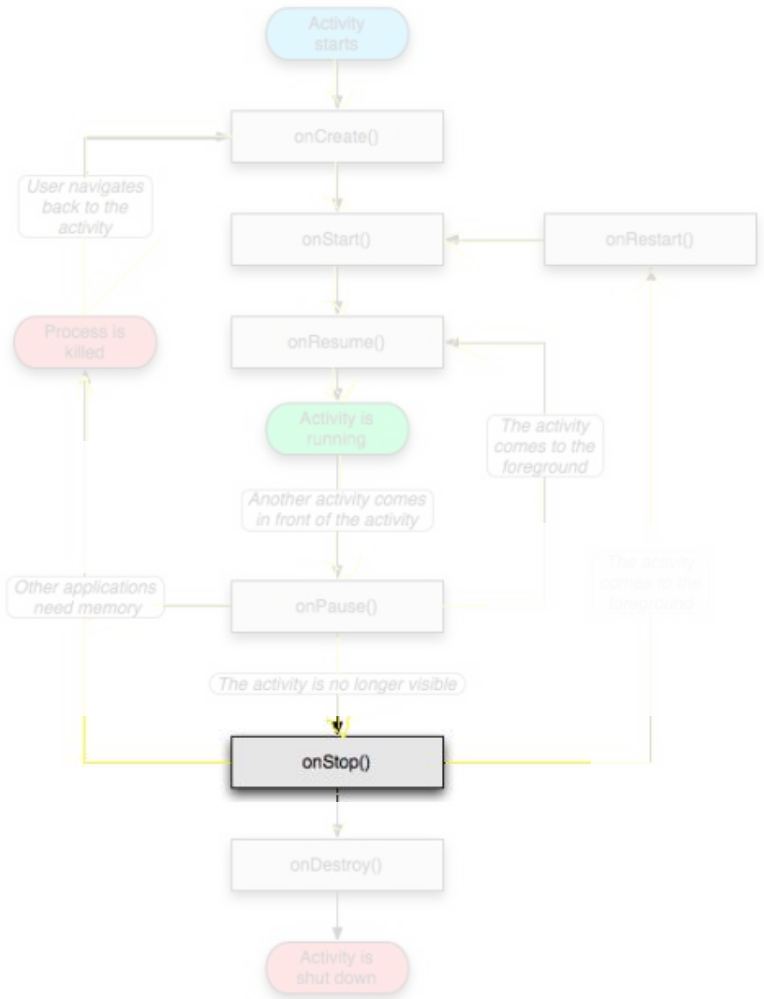
- OnPause()
  - It is FAST:
    - Do not save data
    - Do not fire time-consuming tasks
    - Do not perform database transactions.

Activity lifecycle



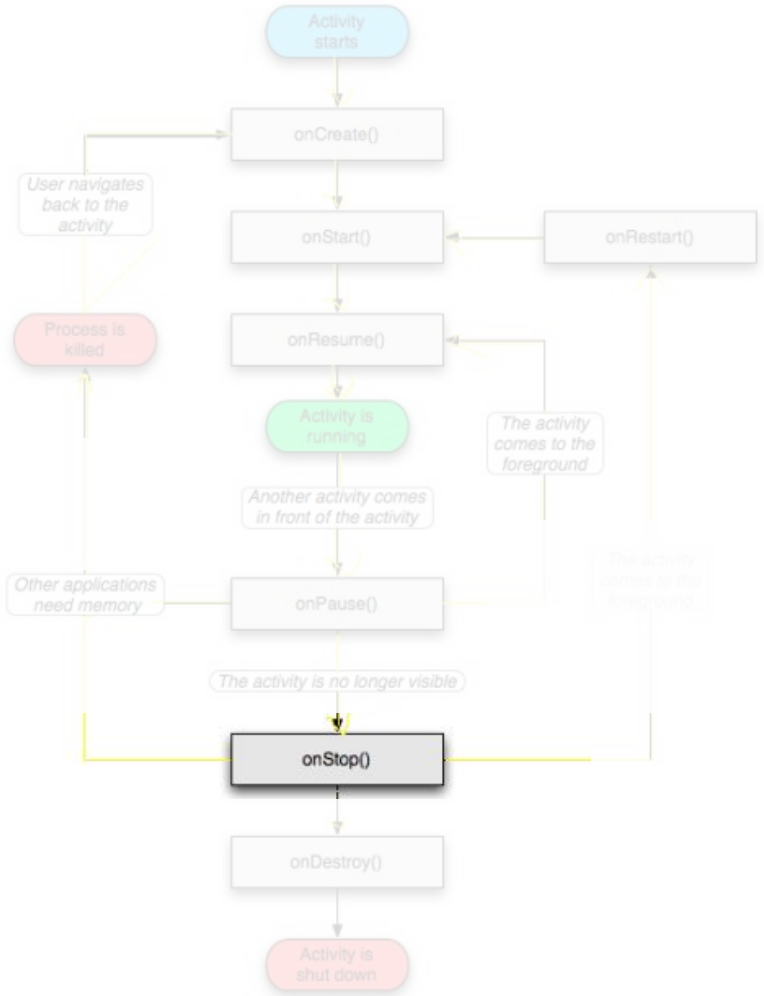
- OnRestart()
  - Similar to onCreate()
  - Only when the activity was previously stopped

Activity lifecycle



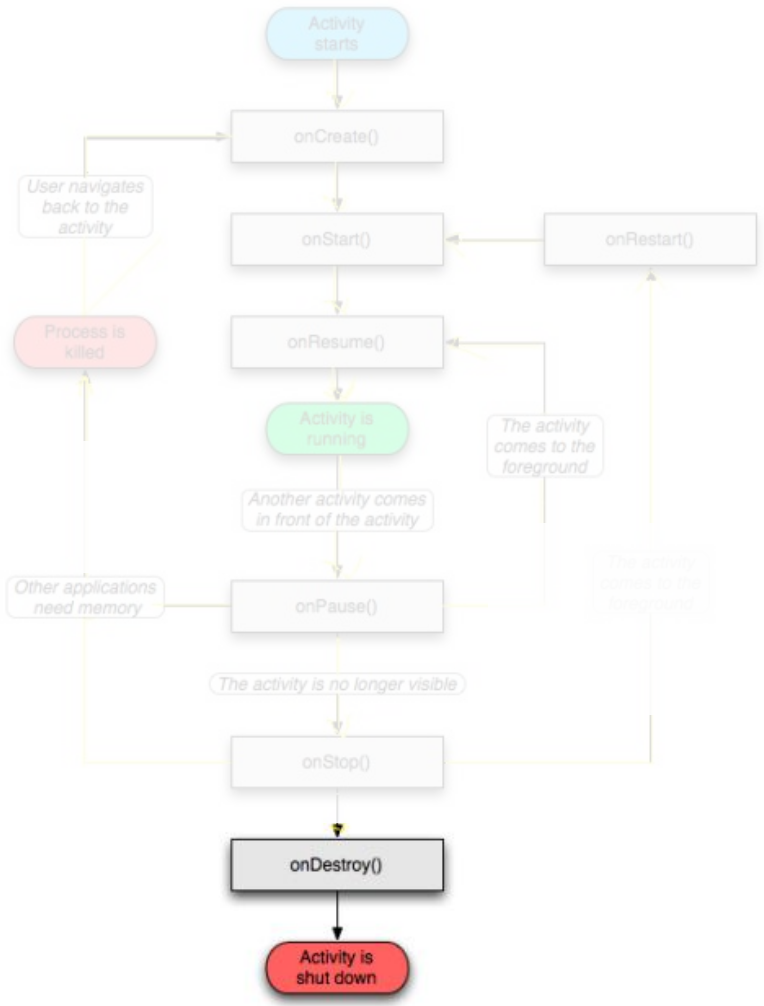
- OnStop()
  - Activity is no longer visible to the user
  - Could be called because:
    - the activity is about to be destroyed
    - another activity comes to the foreground

Activity lifecycle



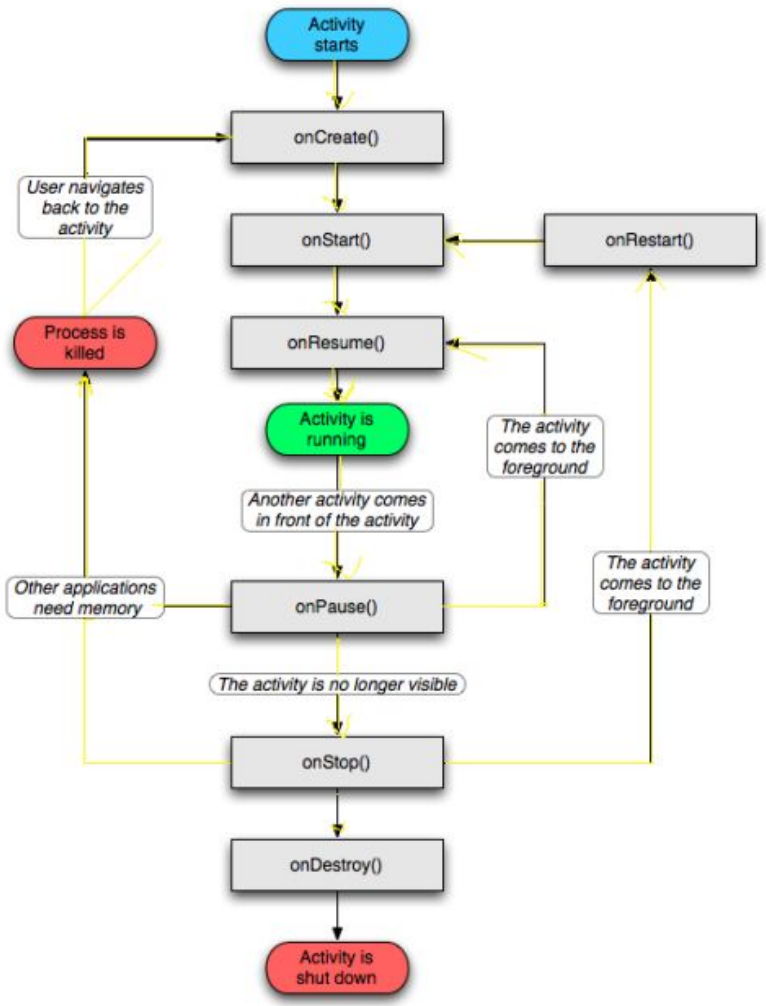
- OnStop()
  - Use to perform CPU-intense shutdown operations.
  - Even if the process is destroyed when the activity is stopped, the system keeps in a Bundle object the state of every View (no need to save).

Activity lifecycle



- OnDestroy()
  - The activity is about to be destroyed
  - Could happen because:
  - The systems need some stack space
    - Someone called finish() method on this activity
    - Can be checked with isFinishing()

Activity loops



Mainly 3 different loops:

- **Entire lifetime**
  - Between onCreate() and onDestroy().
  - Setup of global state in onCreate()
  - Release remaining resources in onDestroy()
- **Visible lifetime**
  - Between onStart() and onStop().
  - Maintain resources that has to be shown to the user.
- **Foreground lifetime**
  - Between onResume() and onPause().
  - Code should be light.

Activities in the manifest

Declare them before running them

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

Why “MAIN” and “LAUNCHER”?

To show the application in the menu

Recreating Activities

Recreating Activities

When an activity is destroyed and then navigated back, the system recreates a new instance. We typically want everything back as it was, which is saved to a Bundle called **Instance State**.

- Android keeps the state of each view
  - Remember to assign unique Ids to them
  - So, no code is needed for the “basic” behavior
- What if I want to save more data?
  - Variables, states…
- A recent alternative to this is the ViewModel (we will see it…)

Recreating Activities

- What if I want to save more data?
  - Override onSaveInstanceState() and onRestoreInstanceState()
  - Use a ViewModel (we will see that later on…)

- onSaveInstanceState() called likely right before onStop()

```
static final String STATE_SCORE = "playerScore";
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
        savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
}
```

Recreating Activities

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first
       if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
    } else {
        // Probably initialize members with default values for a new instance
    }
}
/* The difference is that onRestoreInstanceState is called after onStart() */
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // Call the superclass to restore the views
    super.onRestoreInstanceState(savedInstanceState);
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
}
```

DON'T FORGET THIS, it will restore the state of the views

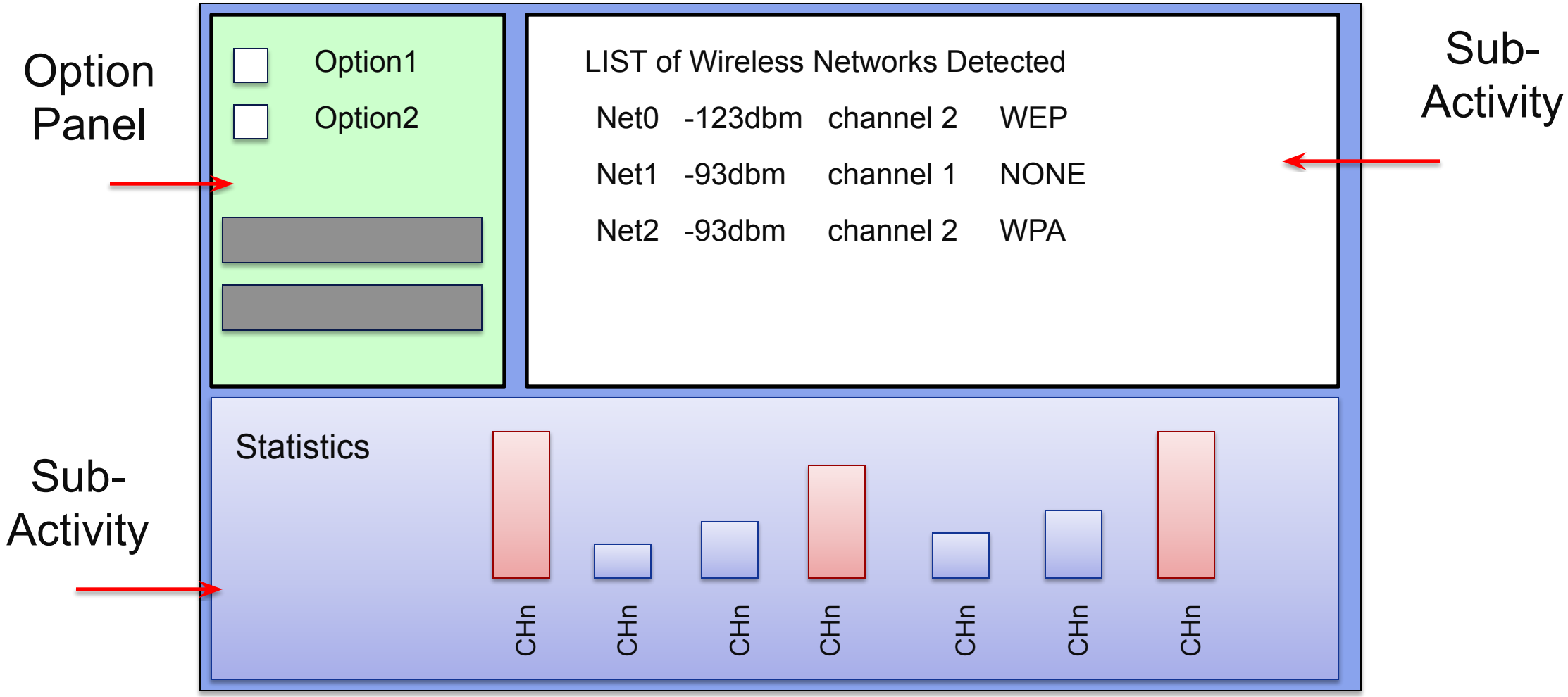
Activity: Conclusions

- Activities should be declared in the Manifest
- Extend the Activity class
- Code wisely
  - Put your code in the right place
  - Optimize it
  - Test even on low-end devices
  - Watch out, configuration changes (rotating screens) destroys the activity

Activity: Missed anything?

If you have questions to put on the table about activities, this is the right time!

Android: Application Case Study



Android: Fragments

Fragment □ A portion of the user interface in an Activity.

Introduced from **Android 3.0** (API Level 11)

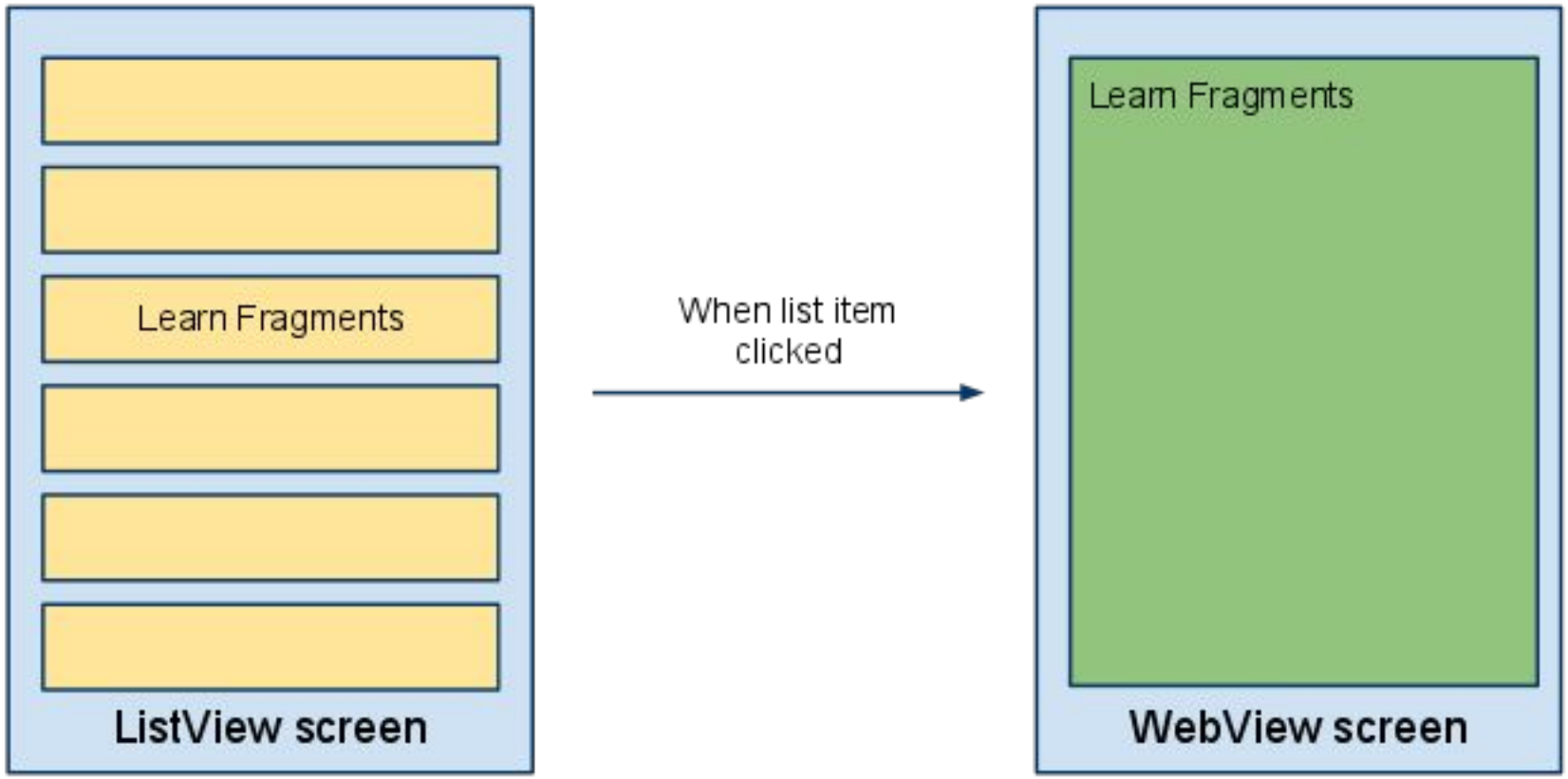
Basically, a Fragment is a modular section of an Activity (a FragmentActivity).

DESIGN PHILOSOPHY

- **Structure** an Activity as a collection of Fragments.
- **Reuse** a Fragment on different Activities …

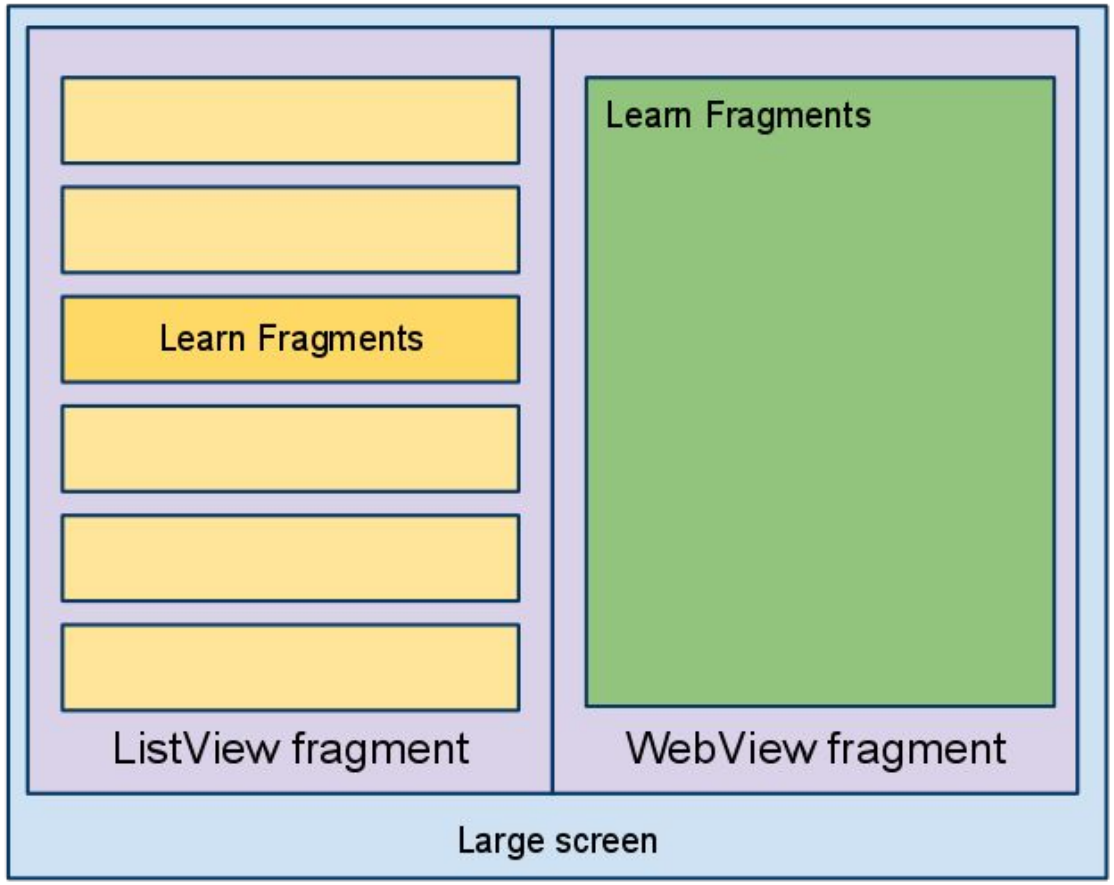
Android: Fragments Design Philosophy

EXAMPLE: Structuring an Application using multiple Activities.



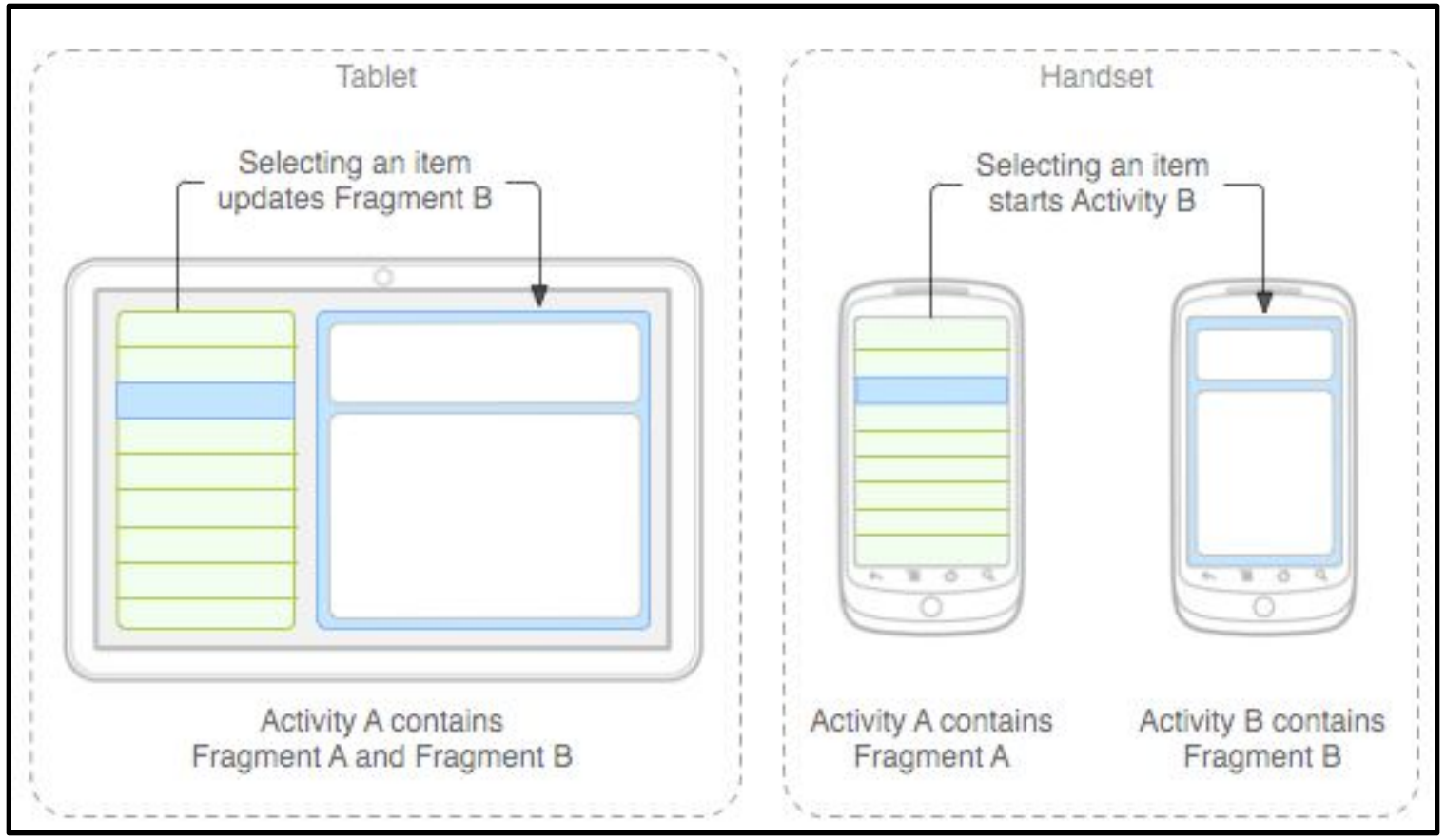
Android: Fragments Design Philosophy

EXAMPLE: Structuring an Application using 1 Activity and 2 Fragments.



Android: Fragment Transactions

EXAMPLE: Using Fragments on Different Devices (Smartphone/Tab)



Android: Fragment Creation

To define a new Fragment → create a subclass of Fragment.

```
public class MyFragment extends Fragment { …}
```

PROPERTY of a Fragment:

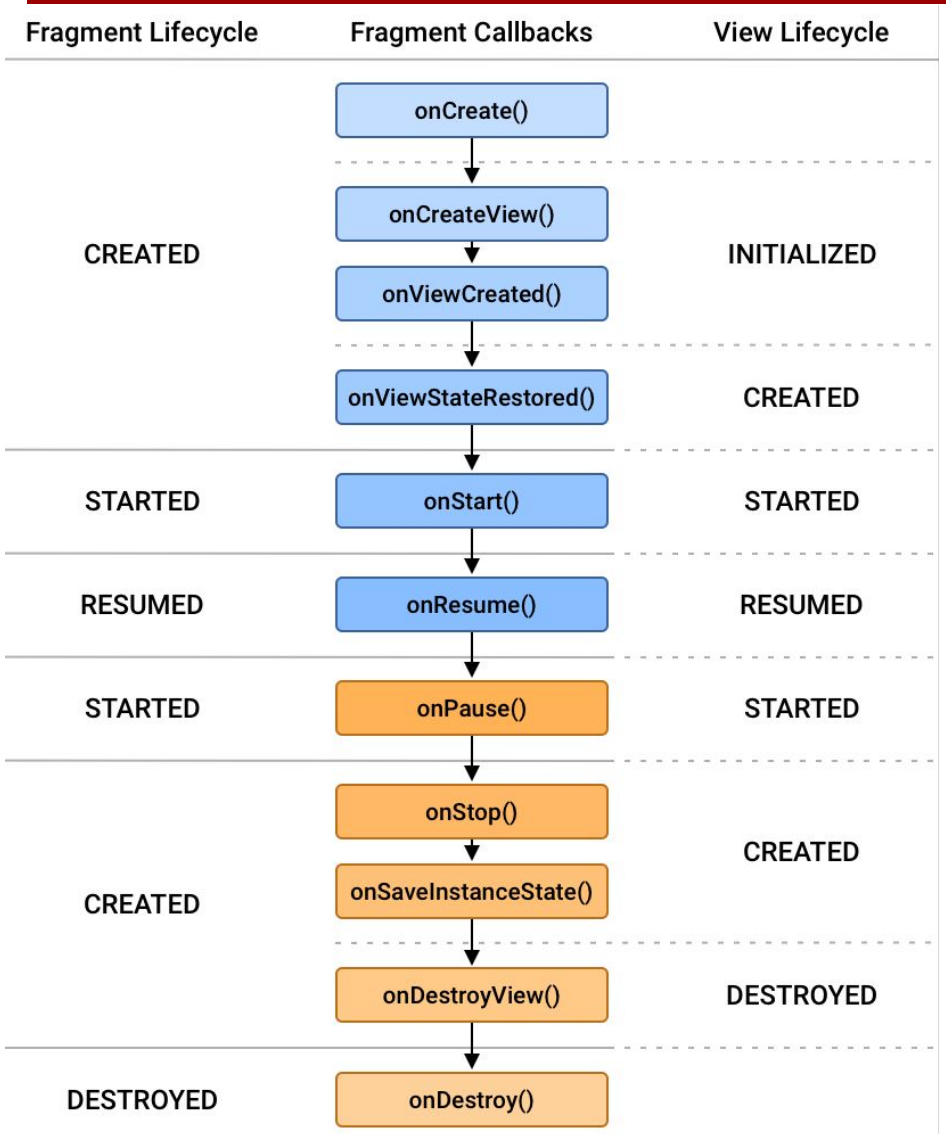
- Has its own **lifecycle** (partially connected with the Activity lifecycle)
- Has its own **layout** (or may have)
- Can receive its own **input events**
- Can be added or removed while the Activity is running.
- Cannot run by itself (always hosted by an Activity)

Android: Fragment Creation

You also might want to extend:

- **DialogFragment**: better than the normal dialog helper ‘cause you can reuse it.
- **ListFragment**: a good alternative for managing lists quite similar to ListActivity
- **PreferenceFragmentCompat**: typically used for displaying preference screens.

Android: Fragment Lifecycle



Several **callback methods** to handle various stages of a Fragment lifecycle.

onCreate() → called when creating the Fragment (elements retained when stopped).

onCreateView() → called when it is time for the Fragment to draw the user interface the first time (or coming back from the backstack). Good to set the properties in **onViewCreated**()

onPause() → called when the user is leaving the Fragment (commit changes in need of persistence).

Android: Fragment Creation

onCreateView() → must return the **View** associated to the UI of the Fragment (if any)
A **LayoutInflater** object will help in doing this...

```
public class ExampleFragment extends Fragment {

 @Override
     public View onCreateView(LayoutInflater inflater,
   ViewGroup container, Bundle savedInstanceState) {

   return inflater.inflate(R.layout.example_fragment,
   container, false);
   }
}
```

Android: Fragment Creation

- **container** is the parent ViewGroup of the Fragment
- **inflate** takes the resource to be inflated and where it should be inflated.

```
public class ExampleFragment extends Fragment {

 @Override
     public View onCreateView(LayoutInflater inflater,
   ViewGroup container, Bundle savedInstanceState) {

   return inflater.inflate(R.layout.example_fragment,
   container, false);
   }
}
```

Android: Fragment Creation

- A recent alternative is to pass the layout over to the super constructor

```
public class ExampleFragment extends Fragment {

   public ExampleFragment () {
      super(R.layout.example_fragment);
   }
}
```

- Less customizable though…

Android: Adding a Fragment to the UI

Specify layout properties for the Fragment as if it was a View.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >
   <fragment android:name="it.cs.android30.FragmentOne"
       android:id="@+id/f1"
       android:layout_width="wrap_content"
       android:layout_height="fill_parent"
       />
   <fragment android:name="it.cs.android30.FragmentTwo"
       android:id="@+id/f2"
       android:layout_width="wrap_content"
       android:layout_height="fill_parent"
       />
</LinearLayout>
```

Fragment Class

watch out, these are permanent...

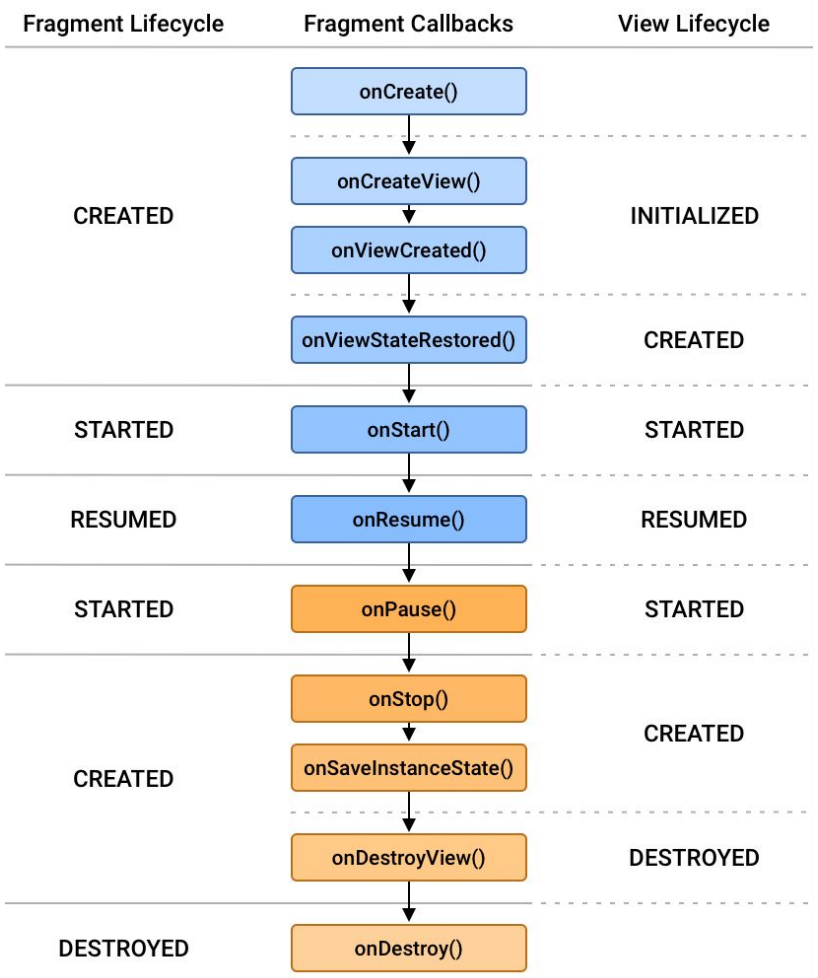
Android: Adding a Fragment to the UI

Once specified, here's what the system does:

- Assigns the layout to the Activity in the usual way
- Creates all the fragments by instantiating the classes and calling the onCreate() method.
- It calls the onCreateView() so, though the inflater, the fragment tells:
  - what is the fragment content in terms of view (par 1)
  - and where to put it (usually the container passed to the function) (par 2)

You can always do this programmatically instead

Android: Fragment Lifecycle



The lifecycle of the Activity in which the Fragment lives directly affects the lifecycle of the Fragment.

onPause (Activity) → **onPause** (Fragment)

onStart (Activity) → **onStart** (Fragment)

onDestroy (Activity) → **onDestroy** (Fragment)

Fragments have also extra lifecycle callbacks to enable runtime creation/destruction.

Android: Managing Fragments

FragmentManager, a support API element that handles the Fragments' lifecycle and scheduling:

From within an **Activity**:

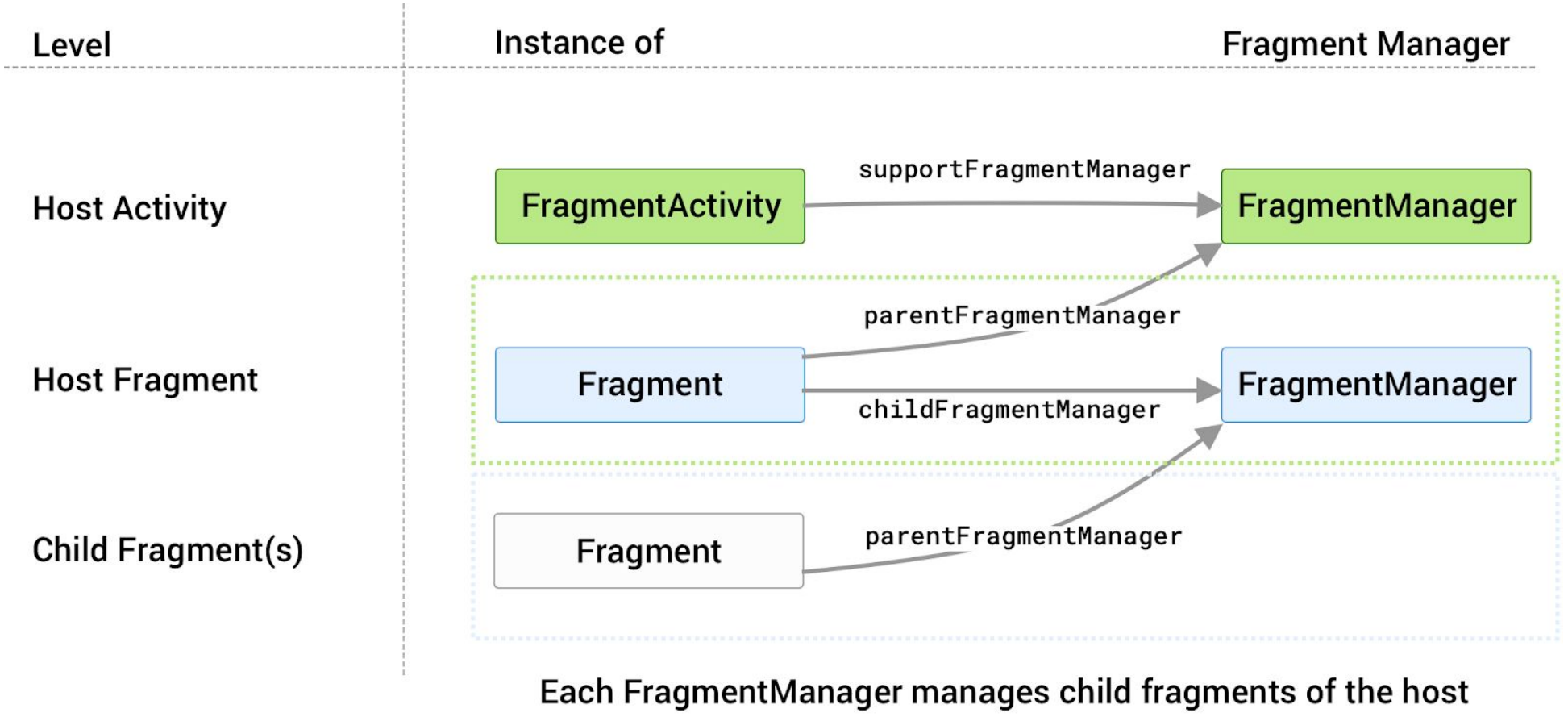
```
getSupportFragmentManager()
```

From within a **Fragment**:

```
getParentFragmentManager()
getFragmentManager() // DEPRECATED
```

The **FragmentManager** manages the Fragment associated to the current Activity.

Android: Managing Fragments



If you are using more advanced frameworks (i.e. **Navigation**), you will probably never interact with the FragmentManager

Android: Managing Fragments

A **Fragment** can get a reference to the Activity …

```
getActivity()
```

An **Activity** can get a reference to the Fragment …

```
ExampleFragment fragment=(ExampleFragment)
getSupportFragmentManager().findFragmentById(R.id.ex
ample_fragment)
```

Before a Fragment enters the lifecycle, it calls its **onAttach()** method right when it gets passed to the FragmentManager.
The dual is **onDetach()**.

Android: Managing Fragments

Need the Activity to react to Fragment events?

Fragment has to expose an interface that the activity must implement and the Fragment checks it in the **onAttach**() (activity is passed here)…

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener listener;
    ...
    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        try {
            listener = (OnArticleSelectedListener) context;
        } catch (ClassCastException e) {
            throw new ClassCastException(context.toString() + " must implement OnArticleSelectedListener");
        }
    }
}
```

Android: Fragment Transactions

- Fragments can be added/removed/replaced while the Activity is running …
- Each set of changes to the Activity is called a **Transaction**.
- **Transaction** can be saved in order to allow a user to navigate backward among Fragments when he clicks on the “Back” button.

Android: Fragment Transactions

1. **ACQUIRE** an instance of the FRAGMENT MANAGER

```
FragmentManager fm = getSupportFragmentManager();
FragmentTransaction transaction = fm.beginTransaction();
transaction.setReorderingAllowed(true);
```

2. **CREATE** new Fragment and Transaction (changes you want at the same time)

```
FragmentExample newFragment = new FragmentExample();
transaction.replace(R.id.fragment_container, newFragment);
```

3. **SAVE** to backStack and **COMMIT**

```
transaction.addToBackStack("FragmentExample");
transaction.commit();
```

FragmentActivity then automatically retrieves fragments from the back stack via onBackPressed()

Android: Fragment Transactions

A Transaction is not performed till the **commit** …

- If **addToBackStack()** is not invoked the Fragment is destroyed and it is not possible to navigate back.
- If **addToBackStack()** is invoked the Fragment is stopped and it is possible to resume it when the user navigates back.
- **popBackStack**() simulates a Back from the user.

Android: Fragment Transactions

Fragment transactions and activity transactions are quite similar:

- They both make use of a backstack

However...

- The backstack of the activities is kept by the system, whereas the backstack of the fragments is kept by the host activity.
- Saving a fragment to the backstack has to be explicitly requested.

Android: Fragment Transactions

- Watch out, you cannot replace the fragment that you declared in the XML (static)!
  - You need to do everything dynamically (i.e. start from a container instead, e.g. a **FrameLayout**) if that's what you want!
  - Why then using static fragments?? Because you can reuse them in the code!
- You can inflate fragments within the **same type** of layout
- Sometimes it is better to use a **FragmentFactory** which is a best practice
  https://www.androiddesignpatterns.com/2012/05/using-newinstance-to-instantiate.html

Android: Fragment Transactions

- However, androidx comes to help us with the **FragmentContainerView**
  - all Fragments are dynamic even when declared on the XML Layout…

```
<fragment android:name="it.cs.android30.FragmentOne"
    android:id="@+id/f1"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    />

<androidx.fragment.app.FragmentContainerView android:name="it.cs.android30.FragmentOne"
    android:id="@+id/f1"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
        />
```

Android: Fragment Transactions

With **FragmentContainerView** new Fragments can be replaced easily.

- Layout of Fragments have <u>always</u> to be within a **FrameLayout** (not necessarily true for <fragment>).
- If **FragmentContainerView** has an **android:name** or a **class** then it triggers a Fragment Transaction when the Activity starts up.
  - *i.e.* it is not static, but it behaves like so…

Android: Missed anything?

- We still need to know how to pass from an Activity to another.
- We need to know what an intent is, don't worry, it's gonna come soon!

Questions?