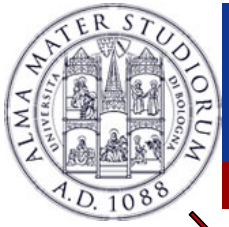



# Programming with Android:
# Activities and Fragments

**Luca Bedogni**

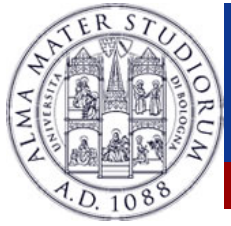
**Dipartimento di Informatica: Scienza e Ingegneria**

**Università di Bologna**



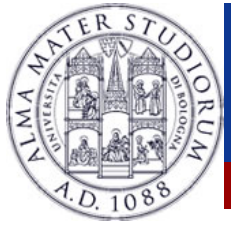
# Outline

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

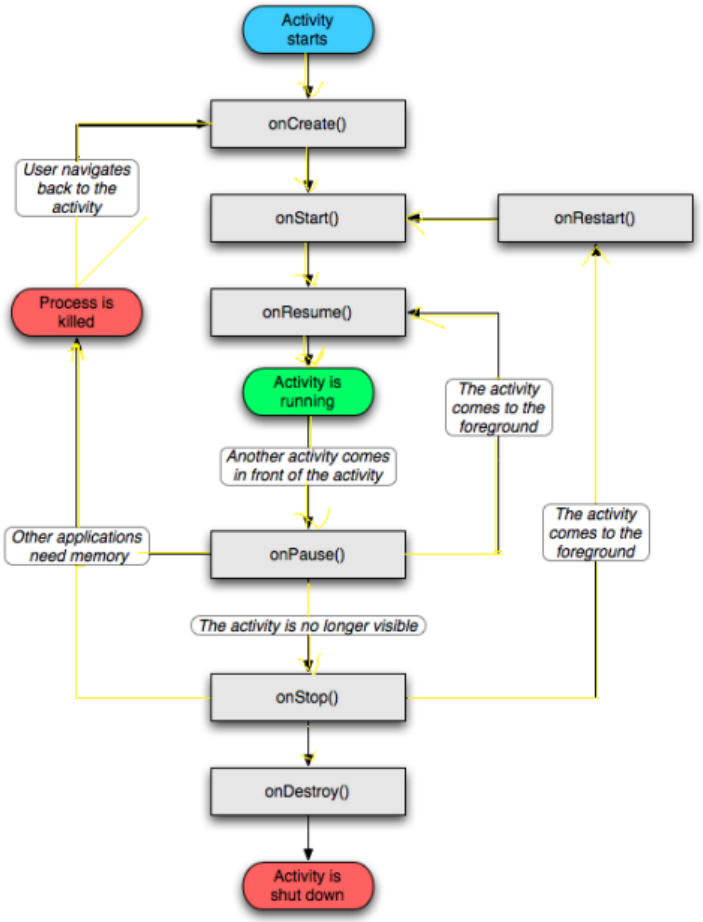


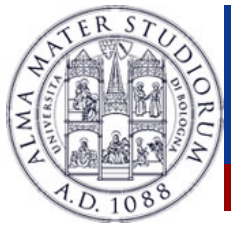
# Activity

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

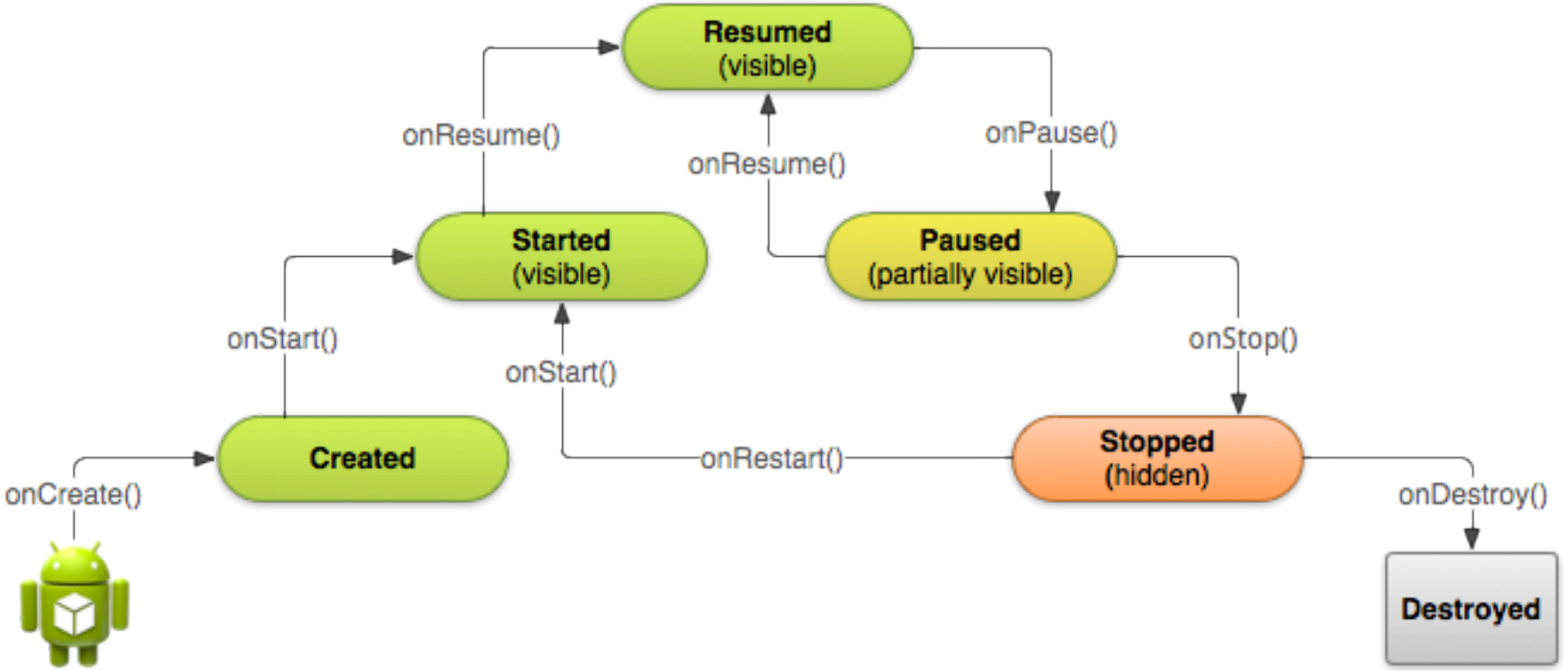


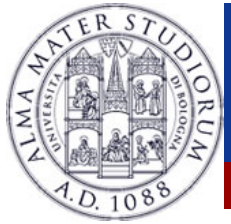
# Activity lifecycle





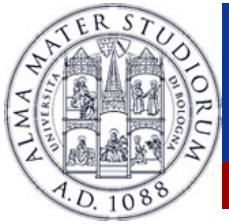
# Activity lifecycle





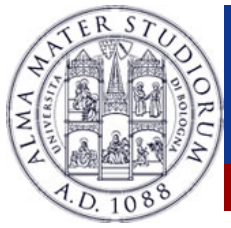
# Activities

- 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

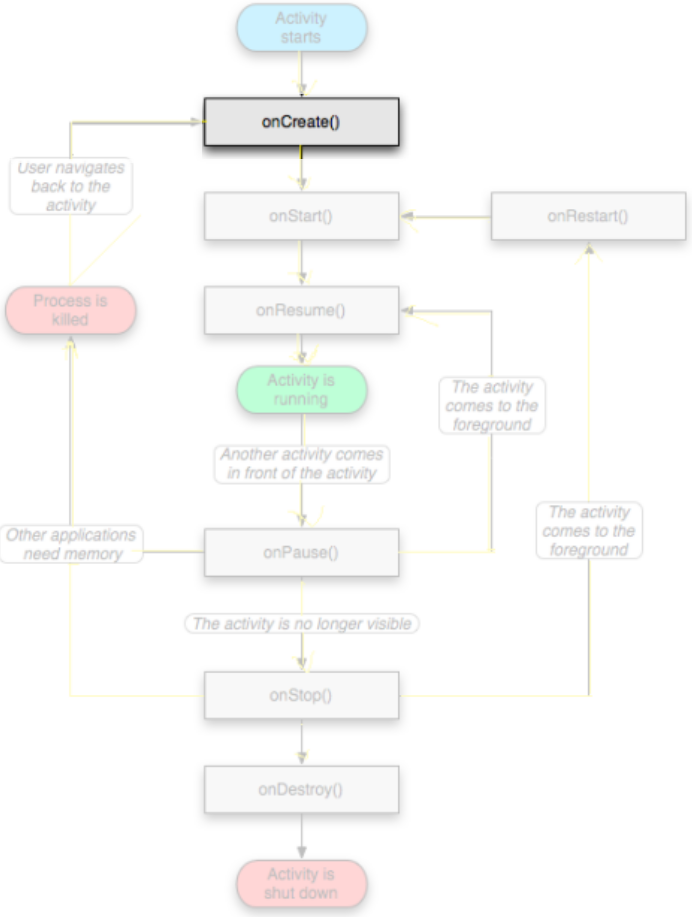


# Activities states

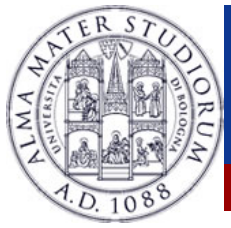
- Resumed
  - The activity is in the foreground, and the user can interact.
- Paused
  - The activity is partially overlayed by another activity. Cannot execute any code nor receive inputs.
- Stopped
  - Activity is hidden, in the background. It cannot execute any code.



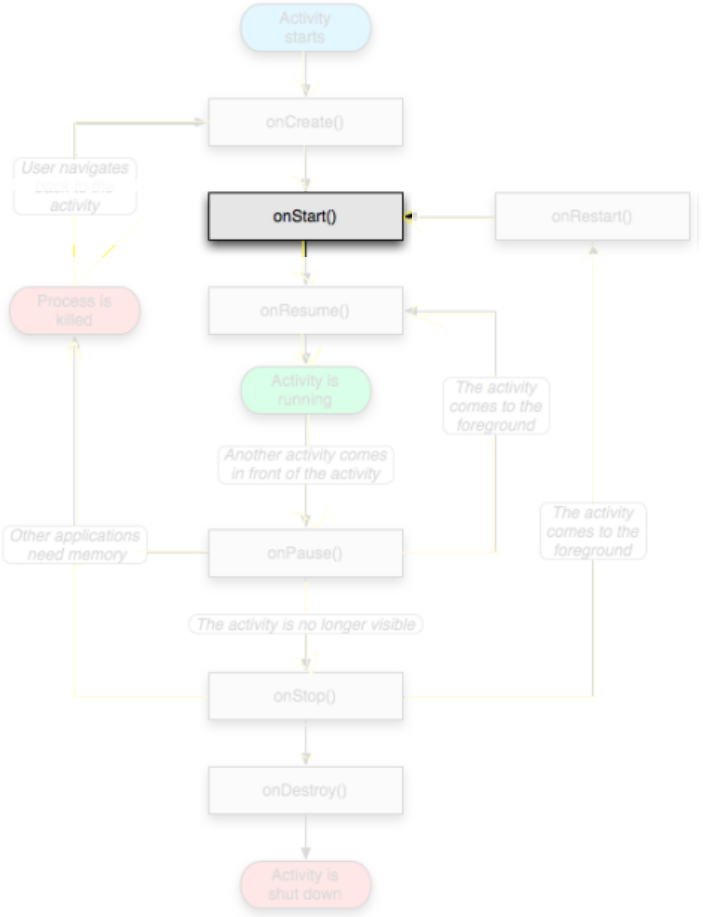
# Activity lifecycle



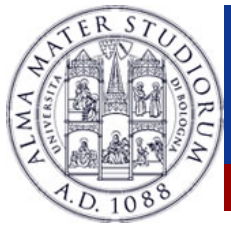

- OnCreate()
  - Called when the activity is created
  - Should contain the initialization operations
  - Has a Bundle parameter
  - If onCreate() terminates, it calls onStart()



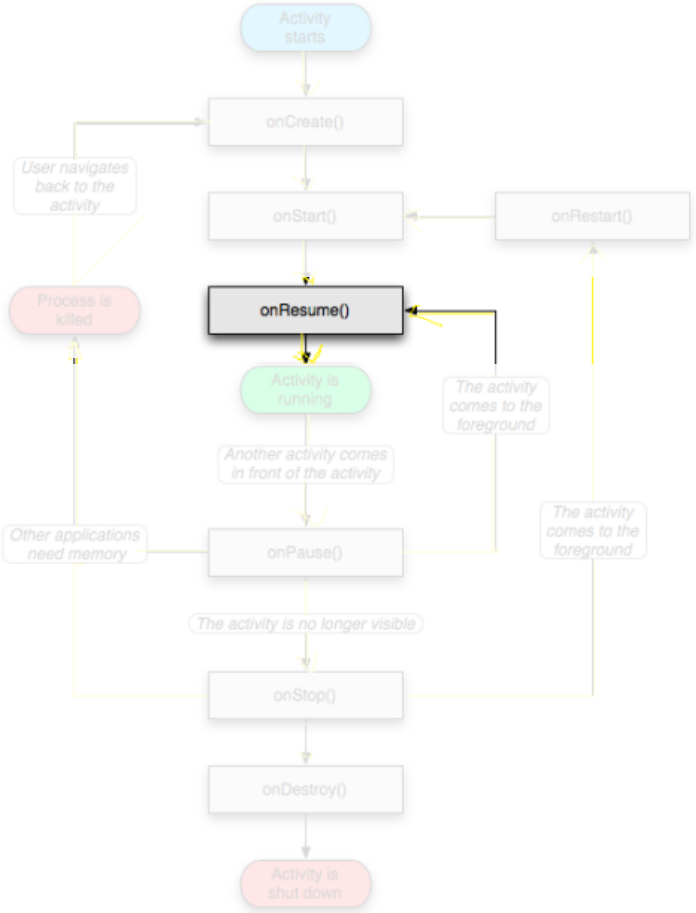
# Activity lifecycle



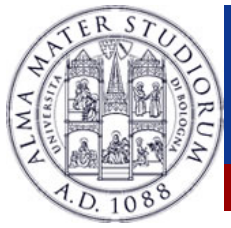

- OnStart()
  - Called when onCreate() terminates
  - Called right before it is visible to user
  - If it has the focus, then onResume() is called
  - If not, onStop() is called



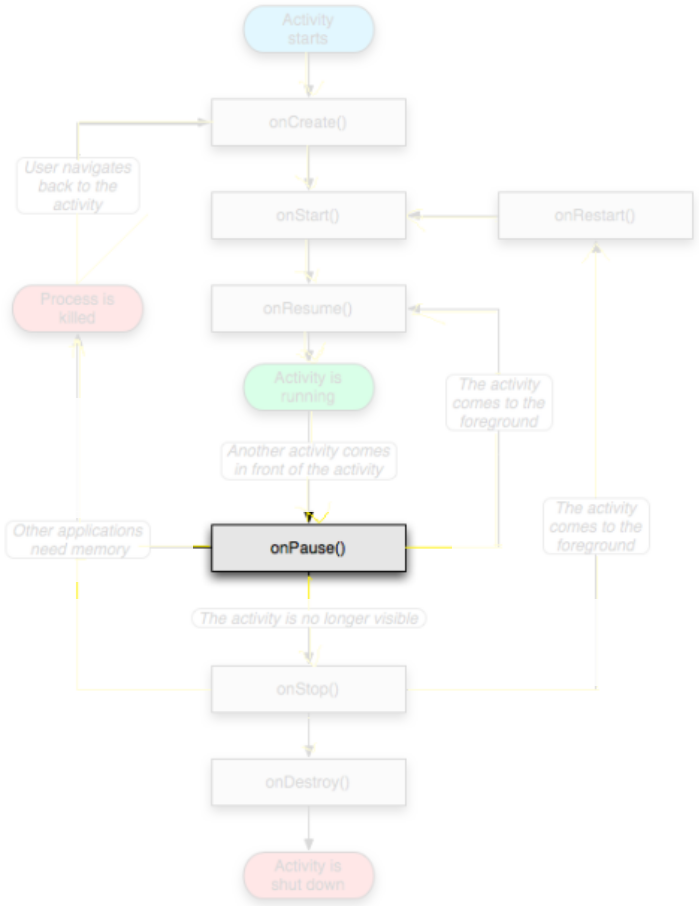
# Activity lifecycle




- OnResume()
  - Called when the activity is ready to get input from users
  - Called when the activity is resumed too
  - If it succesfully terminates, then the Activity is RUNNING

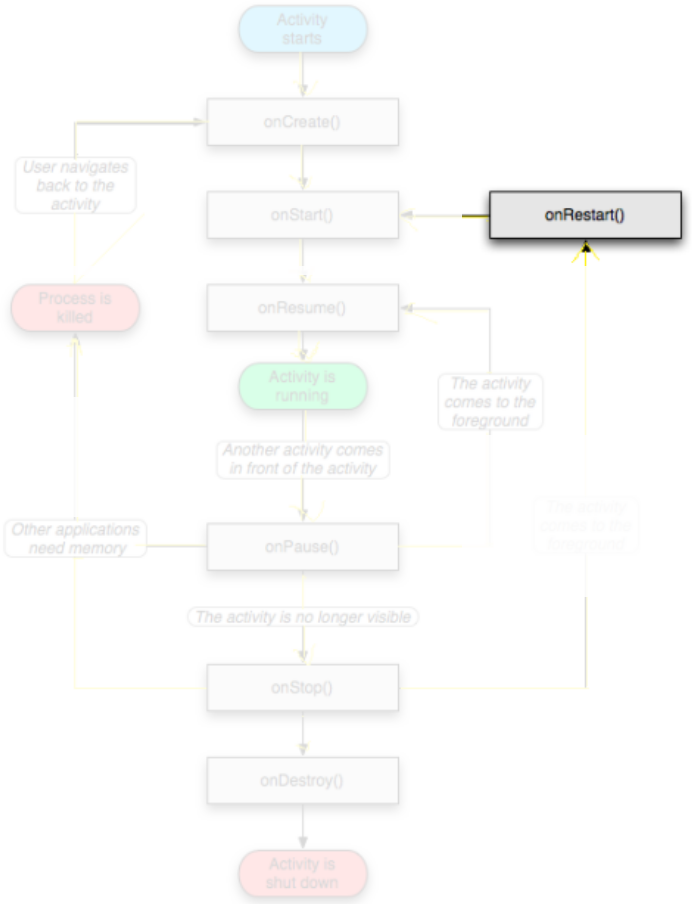


# Activity lifecycle



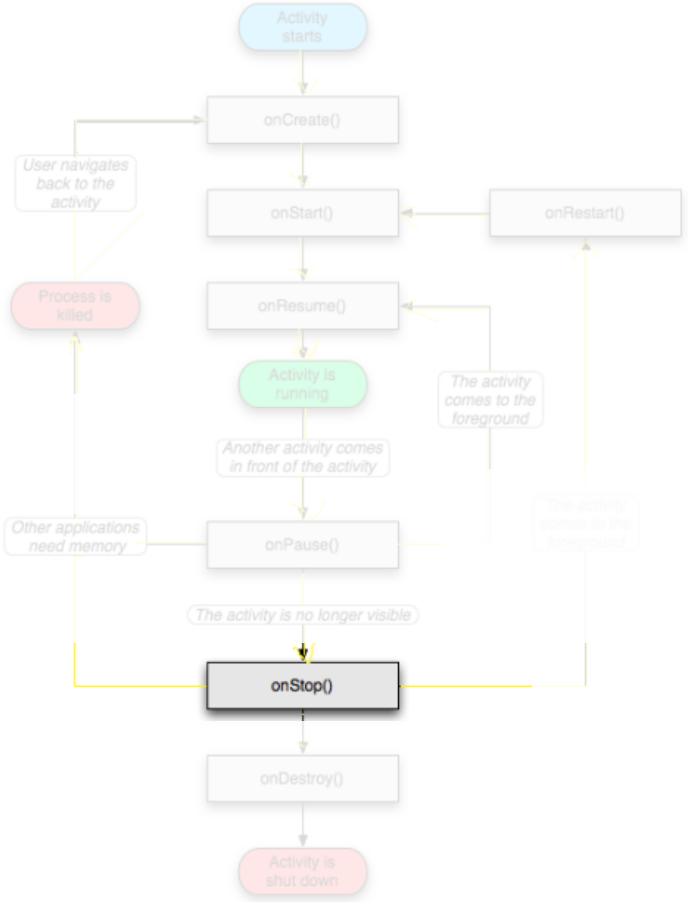

- OnPause()
  - Called when another activity comes to the foreground, or when someone presses back
  - Commit unsaved changes to persistent data
  - Stop cpu-consuming processes
  - Make it fast

# Activity lifecycle

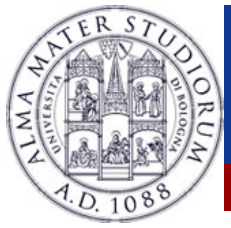



- OnRestart()
  - Similar to onCreate()
  - We have an activity that was previously stopped

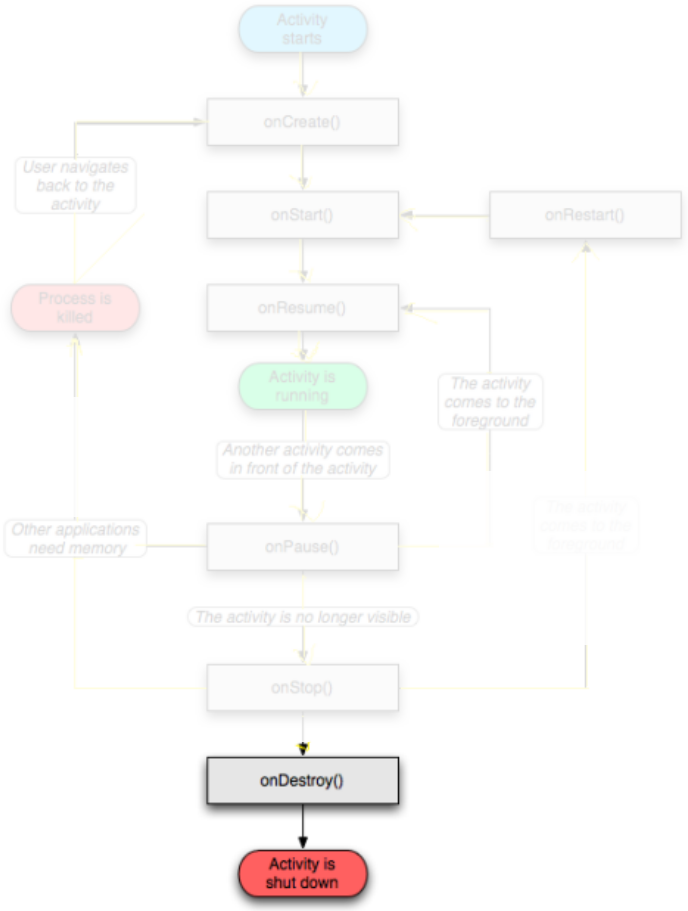
# Activity lifecycle



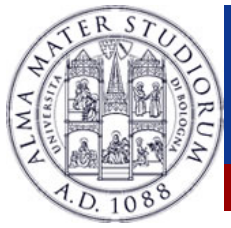

- OnStop()
  - Activity is no longer visible to the user
  - Could be called because:
    - the activity is about to be destroyed
    - another activity comes to the foreground



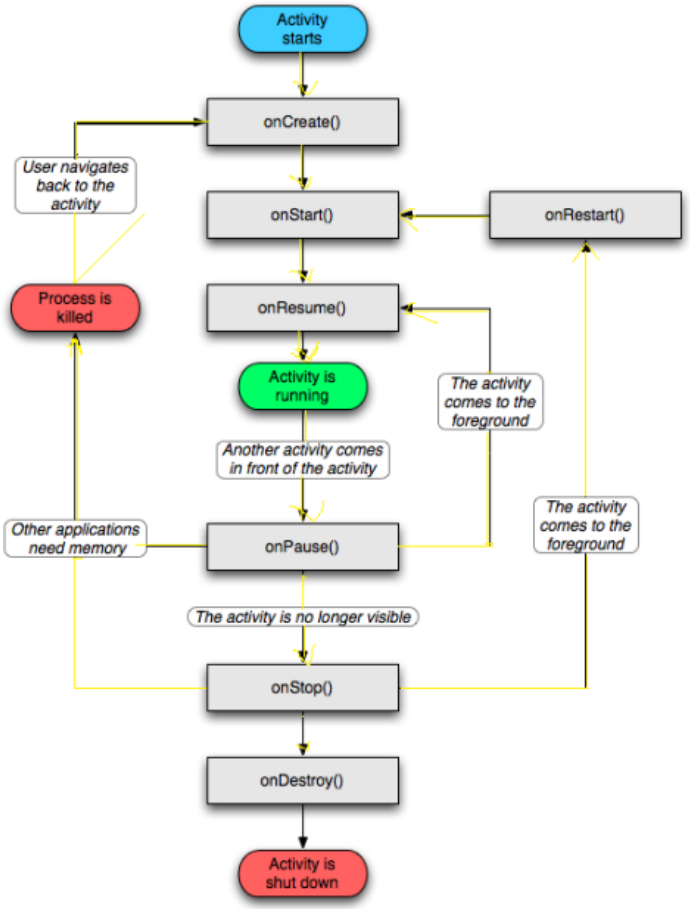
# Activity lifecycle



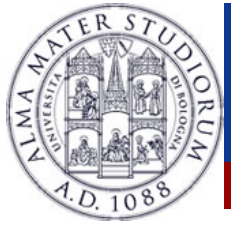

- OnDestroy()
  - The activity is about to be destroyed
  - Could happen because:
  - The systems need some stack space
    - Someone called finish() method on this activity
    - Could check with isFinishing()



# 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.



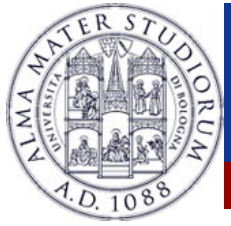
# Activities in the manifest

## Declare them before running them

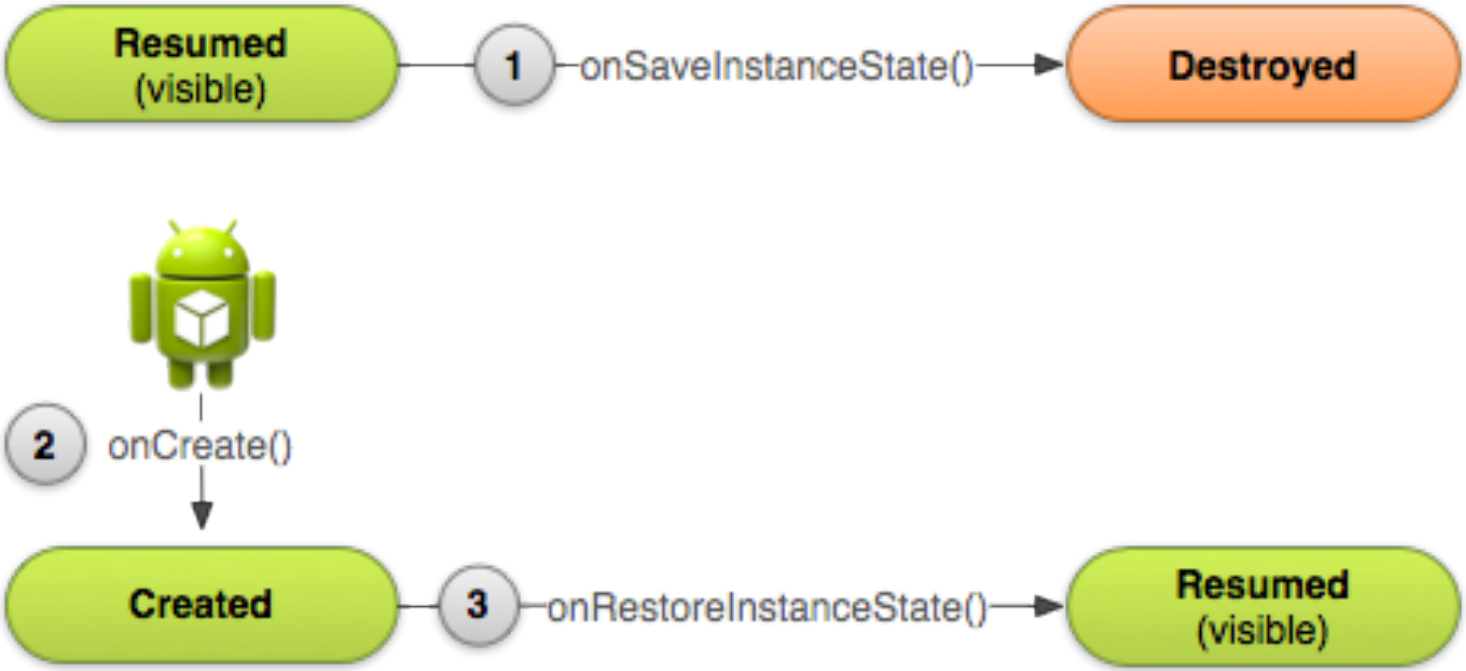
```
<activity android:name=".MainActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

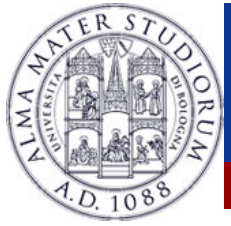
## Why “MAIN” and “LAUNCHER”?

**To show the application in the menu**



# Recreating Activities

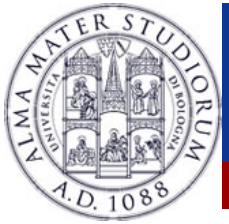




# Recreating Activities

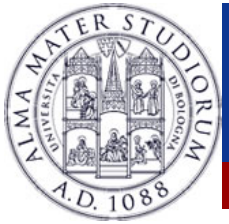
- Android keeps the state of each view
  - Remember to assign unique Ids to them
  - So, no code is needed for the “basic” behavior
- What if I want to save more data?
  - Override onSaveInstanceState() and onRestoreInstanceState()

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



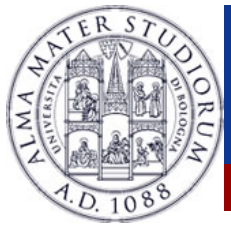
# Recreating Activities

```
@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
    }
}

public void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
}
```

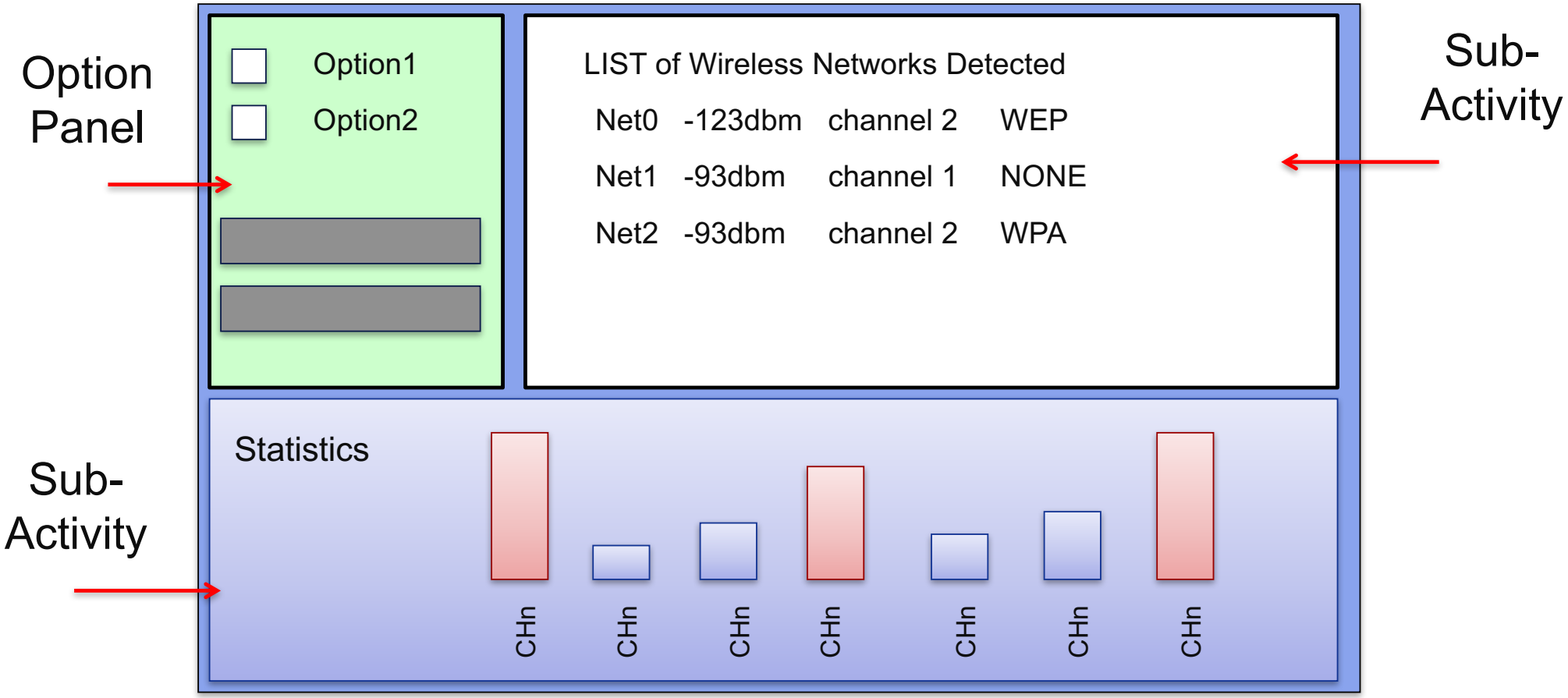


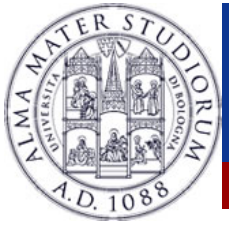
# Activity: Conclusions

- 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



# Android: Application Case Study





# Android: Fragments

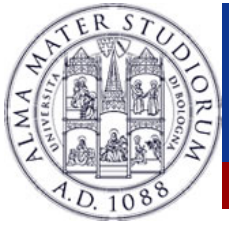
**Fragment** → A portion of the user interface in an Activity.

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

Basically, a Fragment is a modular section of an Activity.

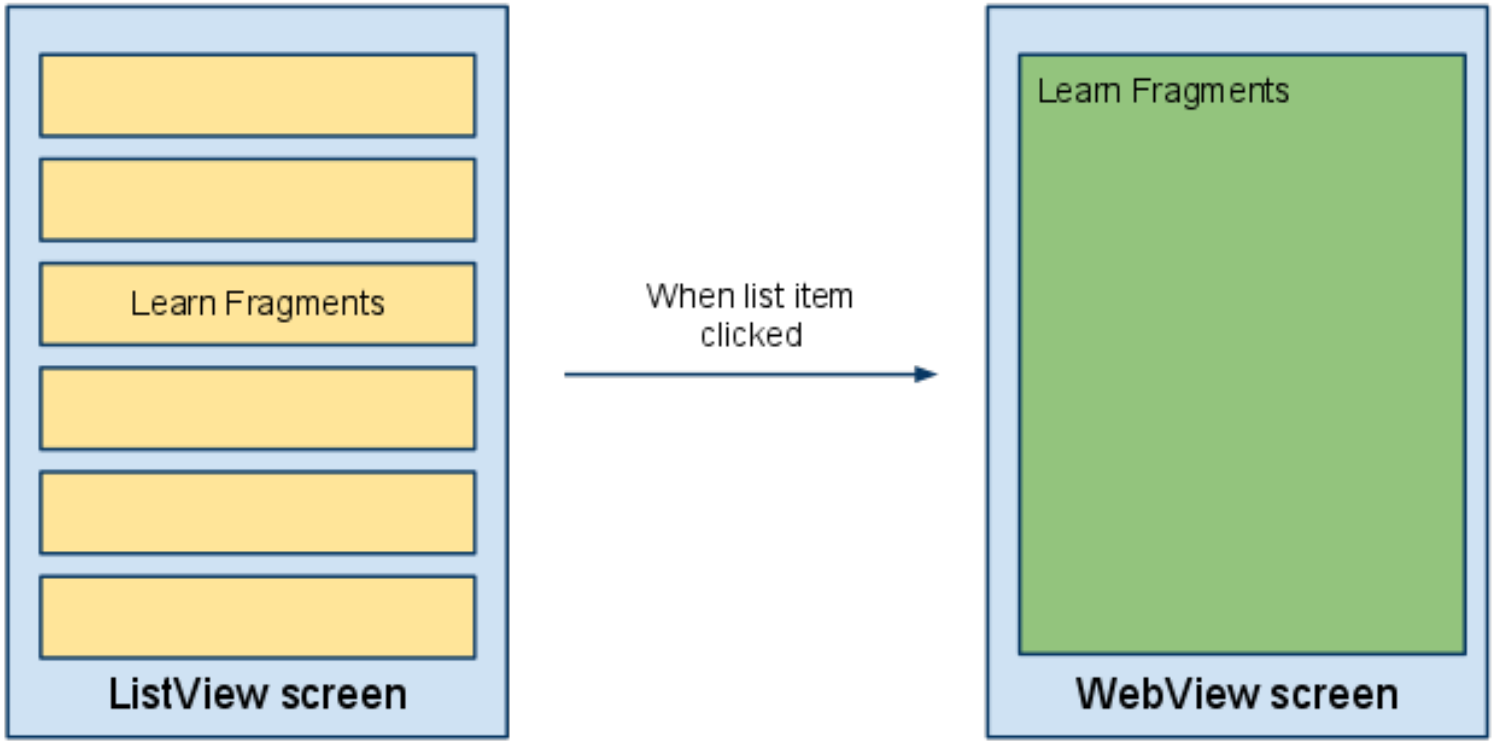
DESIGN PHILOSOPHY

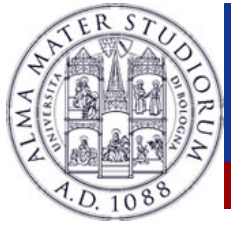
- **Structure** an Activity as a collection of Fragments.
- **Reuse** a Fragment on different Activities …



# Android: Fragments Design Philosophy

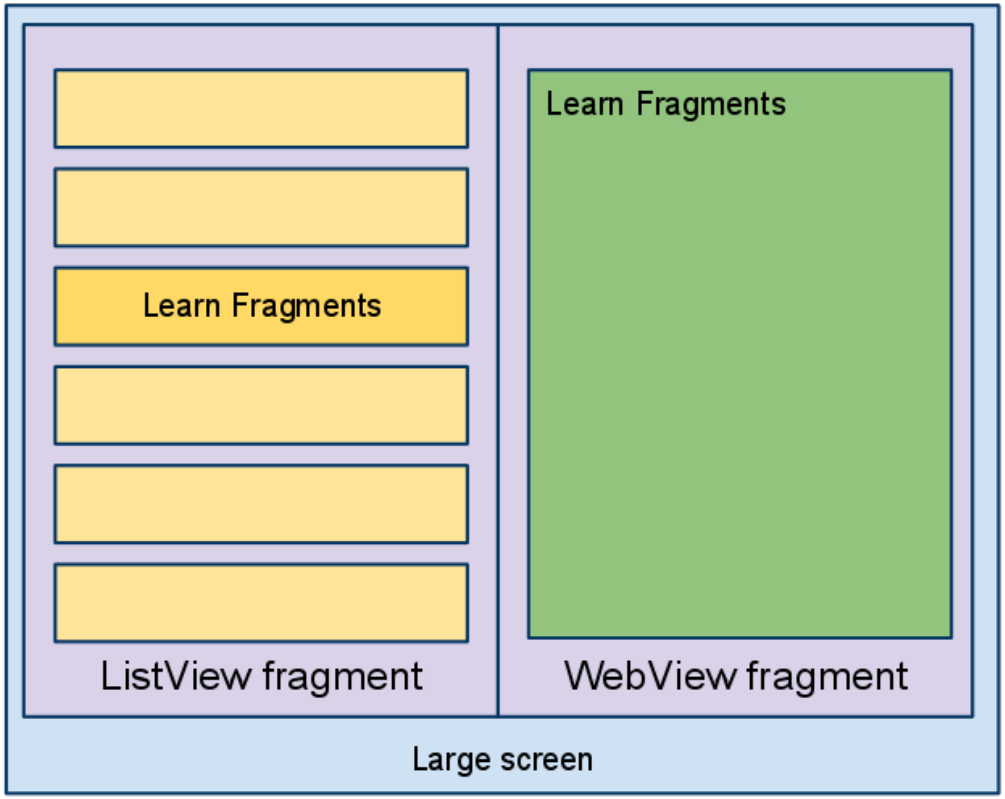
**EXAMPLE**: Structuring an Application using multiple Activities.

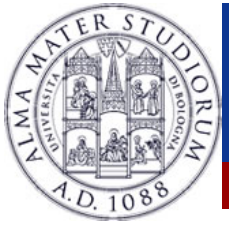




# Android: Fragments Design Philosophy

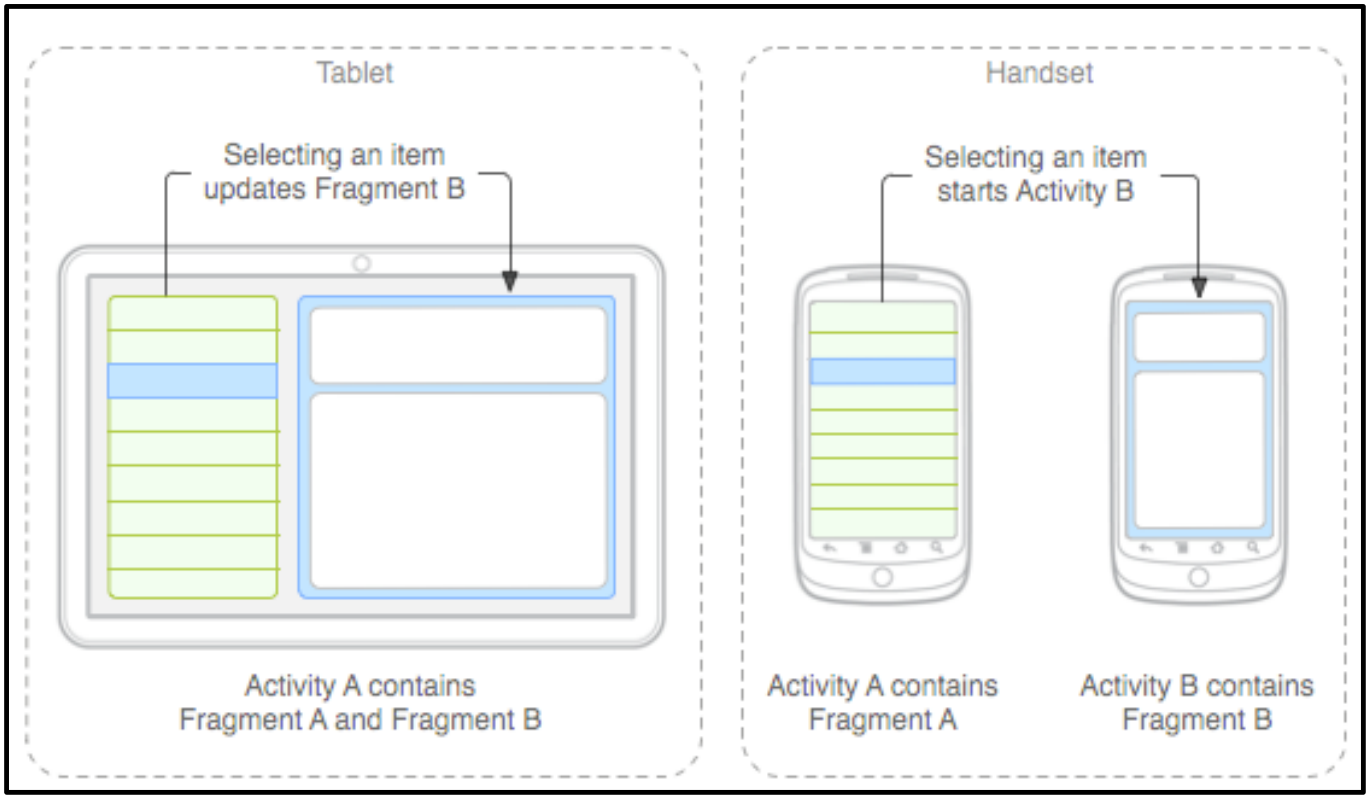
**EXAMPLE**: Structuring an Application using 1 Activity and 2 Fragments.

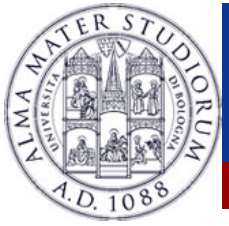




# Android: Fragment Transactions

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





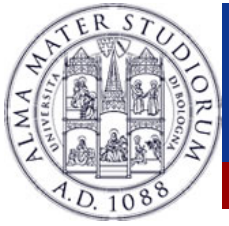
# Android: Fragment Creation

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

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

PROPERTY of a Fragment:

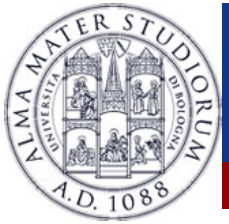
- Has its own **lifecycle** (partially connected with the Activity lifecyle)
- Has its own **layout** (or may have)
- Can receive its own **input events**
- Can be added or removed while the Activity is running.



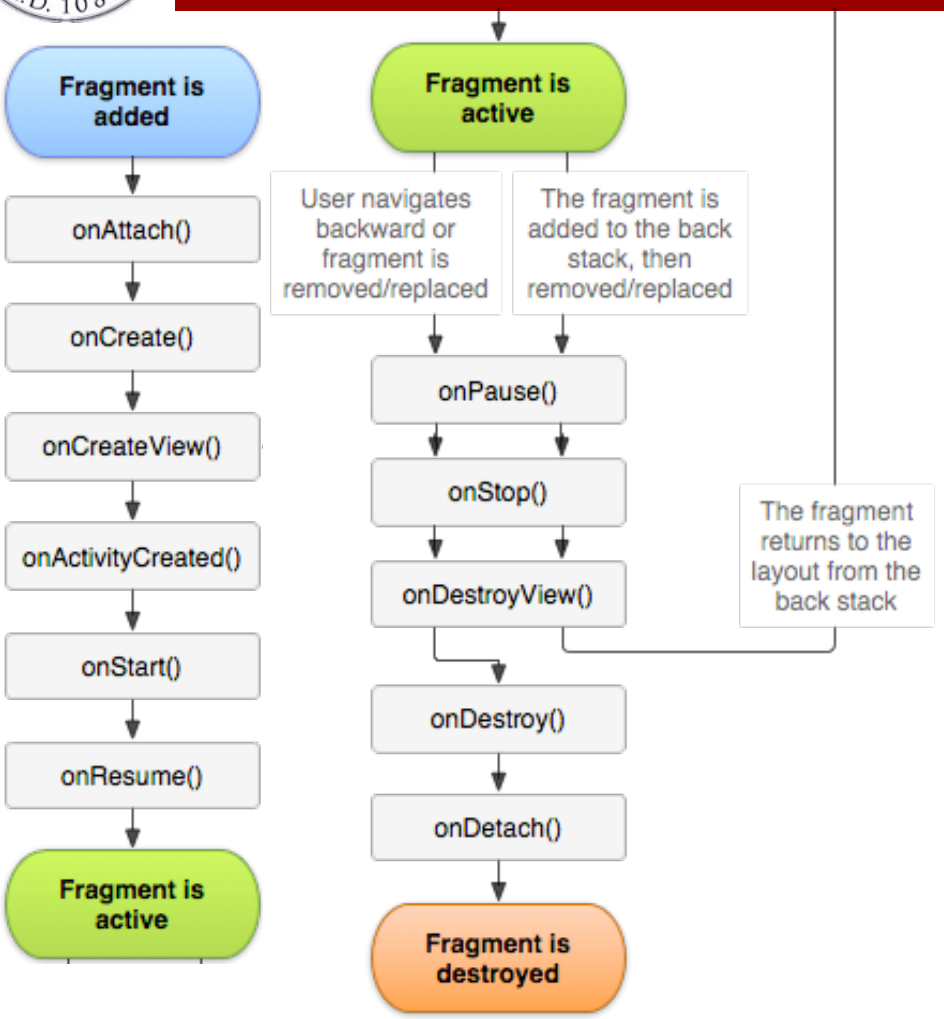
# Android: Adding a Fragment to the UI

Specify layout properties for the Fragment as 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>
```



# Android: Fragment Lifecycle

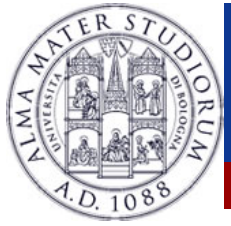



Several **callback methods** to handle various stages of a Fragment lifecycle:

**onCreate**() → called when creating the Fragment.

**onCreateView**() → called when it is time for the Fragment to draw the user interface the first time.

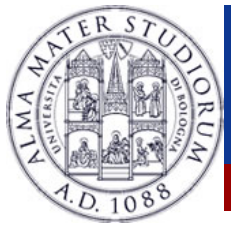
**onPause**() → called when the user is leaving the Fragment.



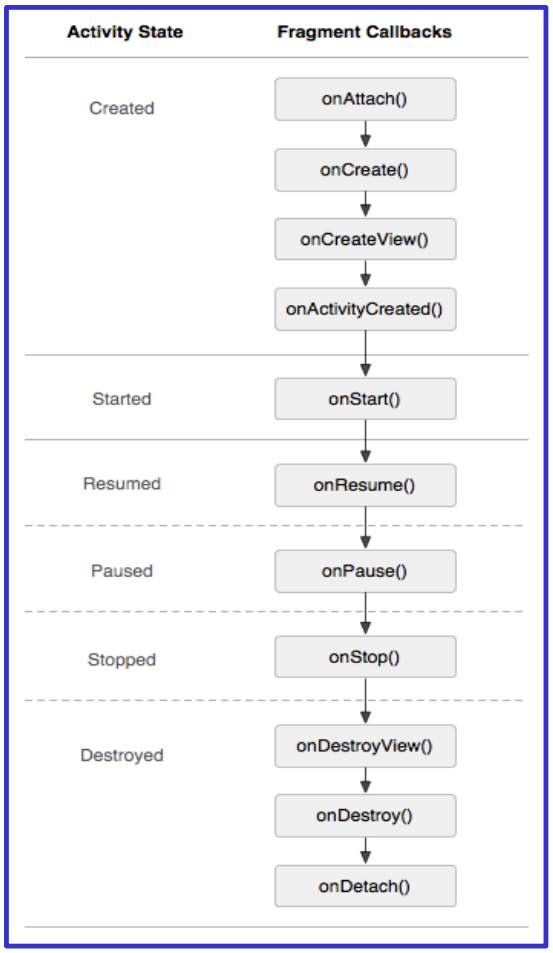
# Android: Fragment Creation

**onCreateView**() → must return the **View** associated to the UI of the Fragment (if any) …

```
  public class ExampleFragment extends Fragment {

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

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

 }
```



# Android: Fragment Lifecycle



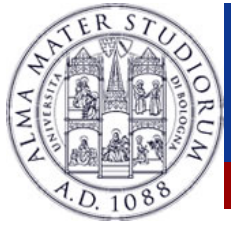

The lifecycle of the Activity in which the Fragment lives directly affects the lifecycle of the Fragment.

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

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

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

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



# Android: Managing Fragments

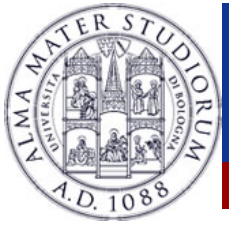
A **Fragment** can get a reference to the Activity …

```
getActivity()
```

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

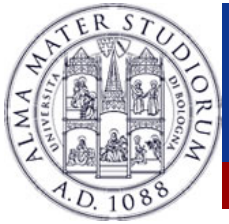
```
ExampleFragment fragment=(ExampleFragment)
getFragmentManager().findFragmentById(R.id.example_f
ragment)
```

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



# 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 man=getFragmentManager();
FragmentTransaction transaction=man.beginTransaction();
```

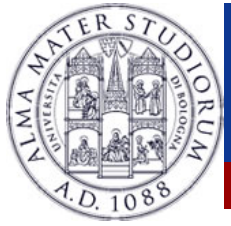
2. **CREATE** new Fragment and Transaction

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

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

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

```
transaction.commit();
```



# 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.