Application's Life Cycle

Anatomy of Android Applications

An Android application consists of one or more *core components*.

In the case of apps made of multiple parts, collaboration among the independent core components is required for the success of the application.

A core component can be:

1. ***An Activity***
2. ***A Service***
3. ***A broadcast receiver***
4. ***A content provider***

Anatomy of Android Applications

1. Activity

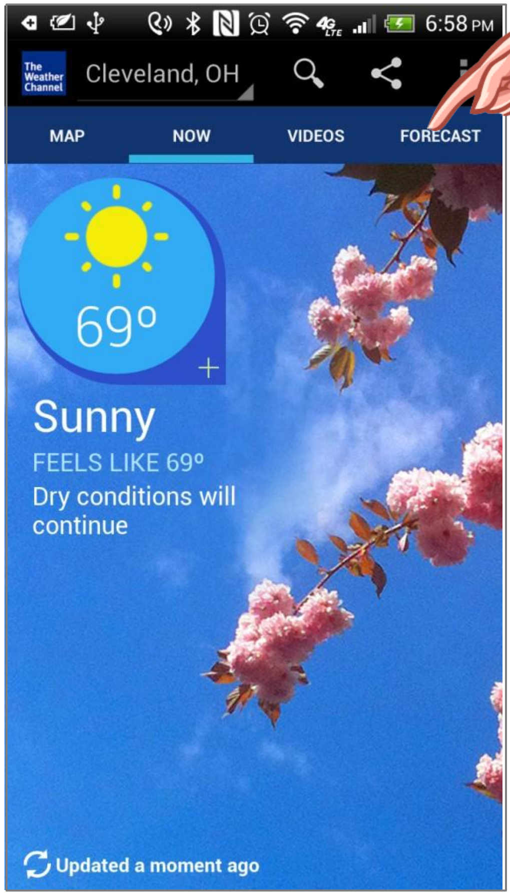
- A typical Android **application** consists of *one or more* **activities**.
- An *activity* usually shows a *single visual user interface.*
- One activity is chosen to be executed first when the application is launched.
- An activity may transfer control and data to another activity through an interprocess communication protocol called ***intents***.

Anatomy of Android Applications

Weather Channel app
GUI-1- Activity 1

Weather Channel app
GUI-2- Activity 2

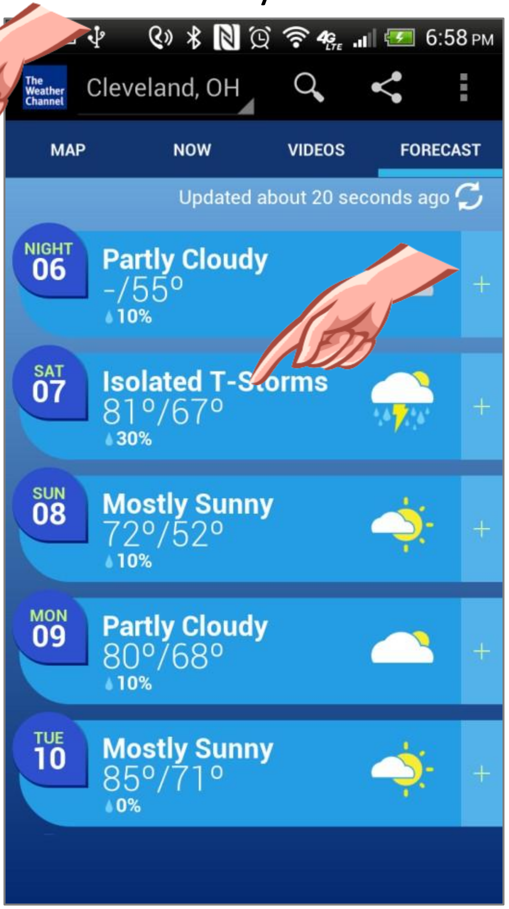
Weather Channel app
GUI-3- Activity 3



69°

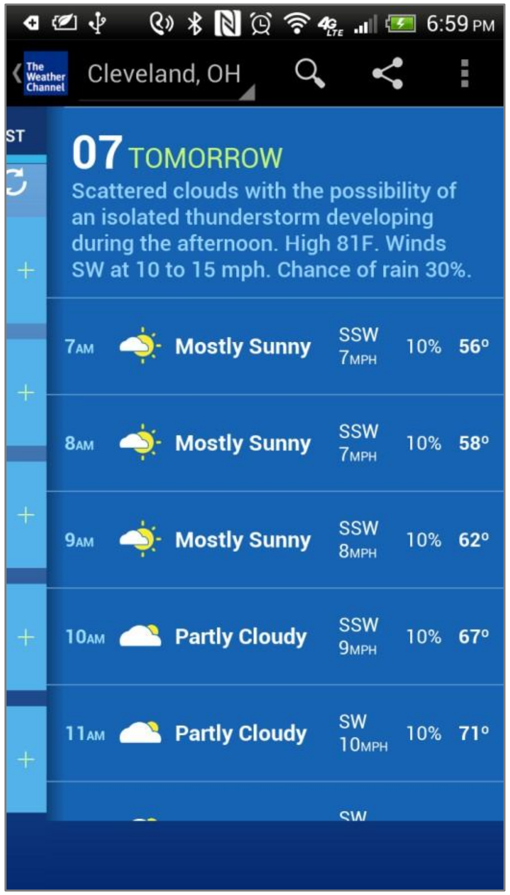
Sunny
FEELS LIKE 69°
Dry conditions will continue

Updated a moment ago



Updated about 20 seconds ago

NIGHT 06	Partly Cloudy	-/55°	10%
SAT 07	Isolated T-Storms	81°/67°	30%
SUN 08	Mostly Sunny	72°/52°	10%
MON 09	Partly Cloudy	80°/68°	10%
TUE 10	Mostly Sunny	85°/71°	0%



07 TOMORROW
Scattered clouds with the possibility of an isolated thunderstorm developing during the afternoon. High 81F. Winds SW at 10 to 15 mph. Chance of rain 30%.

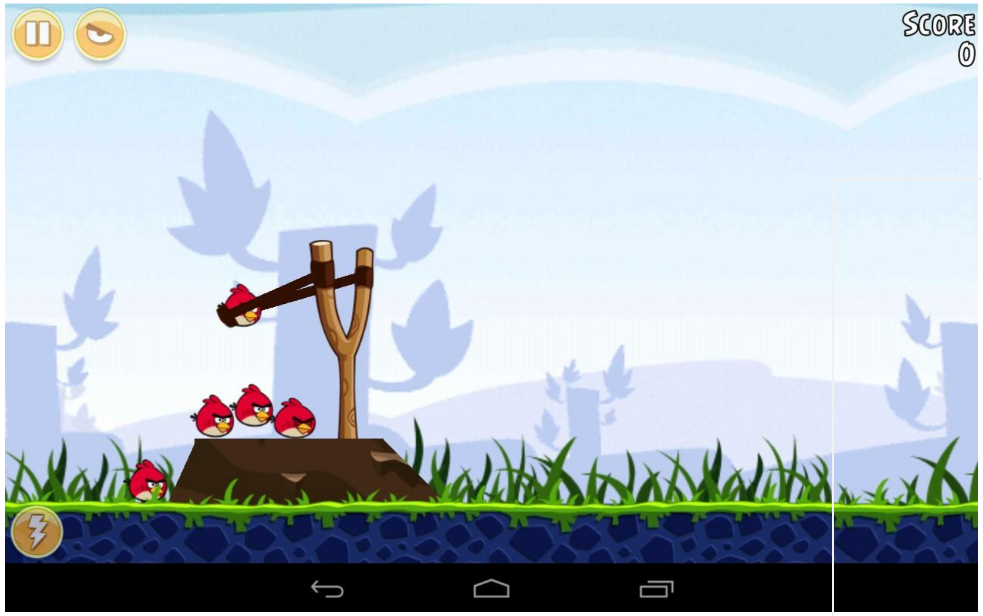
7AM	Mostly Sunny	SSW 7MPH	10%	56°
8AM	Mostly Sunny	SSW 7MPH	10%	58°
9AM	Mostly Sunny	SSW 8MPH	10%	62°
10AM	Partly Cloudy	SSW 9MPH	10%	67°
11AM	Partly Cloudy	SW 10MPH	10%	71°

Anatomy of Android Applications

2. Service

- Services are a special type of activity that *do not have a visual user interface.*
- Services usually run in the background for an indefinite period of time.

2. Example: Service

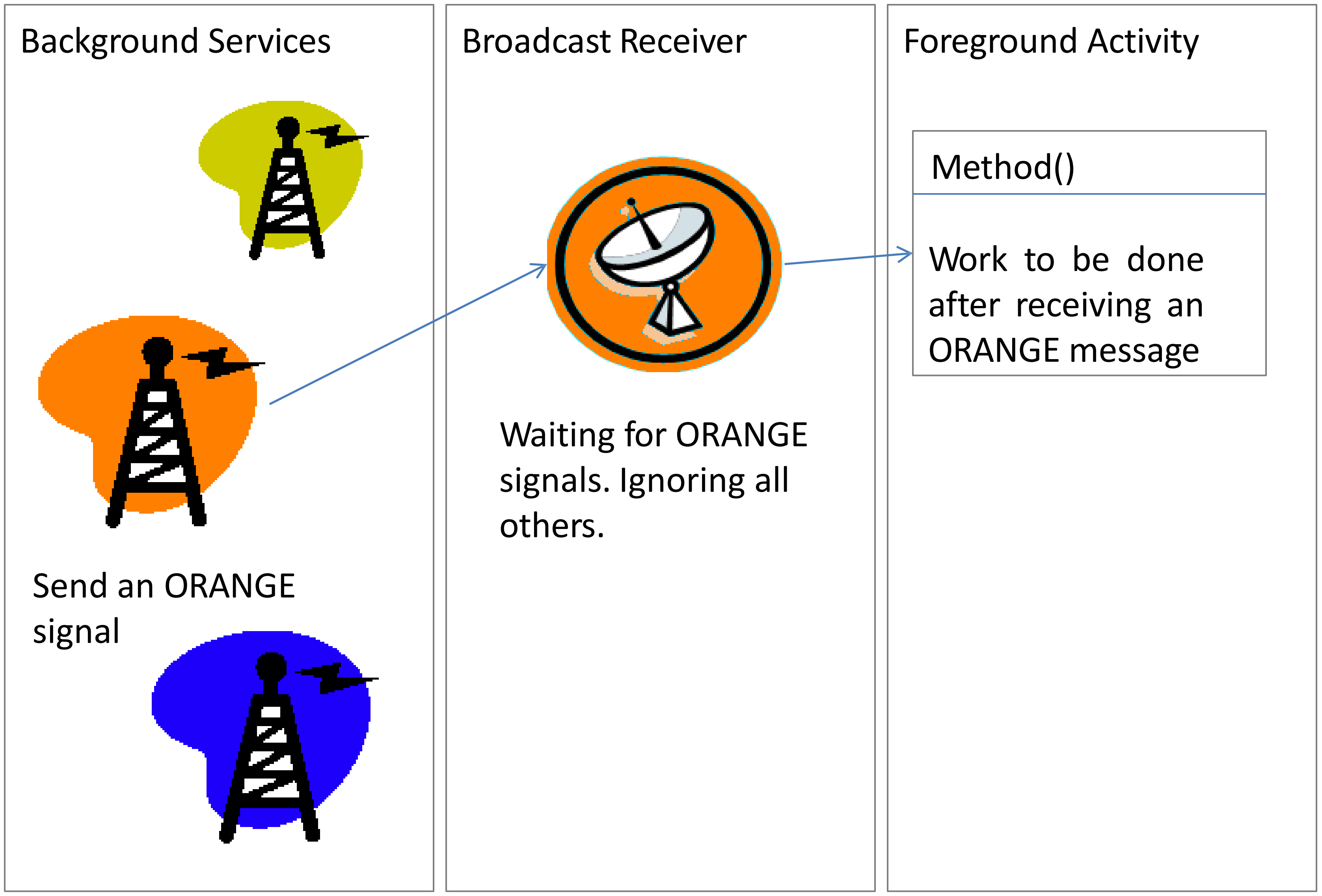
Background	Foreground
PANDORA	

Anatomy of Android Applications

3. Broadcast receiver

- A **BroadcastReceiver** is a dedicated listener that waits for messages.
- *Broadcast receivers do not display a user interface*.
- They typically register with the system to listen for specific events (e.g., system booted, battery low or okay, power connected or disconnected, package installed, etc.).
- A broadcast receiver could respond by either executing a specific activity or use the *notification* mechanism to request the user's attention.

3. Broadcast receiver



Background Services

Send an ORANGE signal

Broadcast Receiver

Waiting for ORANGE signals. Ignoring all others.

Foreground Activity

Method()

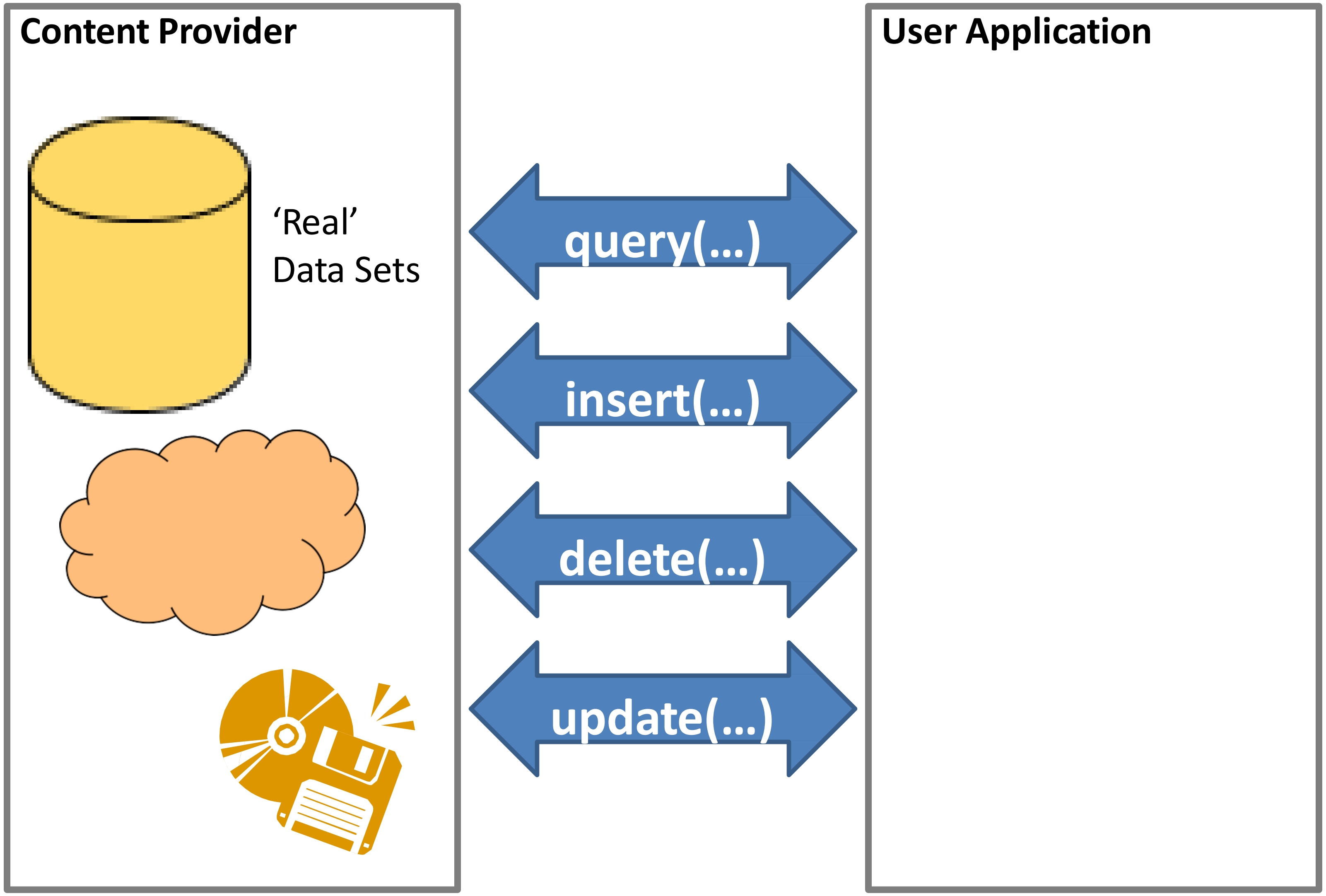
Work to be done after receiving an ORANGE message

Anatomy of Android Applications

4. Content provider

- A *content provider* is a data-centric service that makes persistent datasets available to any number of applications.
- Common global datasets include: contacts, pictures, messages, audio files, emails.
- The global datasets are usually stored in a SQLite database (however the developer does not need to be an SQLexpert)
- The content provider class offers a standard set of “database-like“ methods to enable other applications to retrieve, delete, update, and insert data items.

4. Content provider



A Content Provider is a wrapper that hides the actual physical data. Users interact with their data through a common object interface.

Application's Life Cycle

Each Android application runs inside its own instance of a Dalvik Virtual Machine.

At any point in time several parallel DVM instances could be active.

Unlike a common Windows or Unix process, an Android application does not *completely* control the completion of its lifecycle.

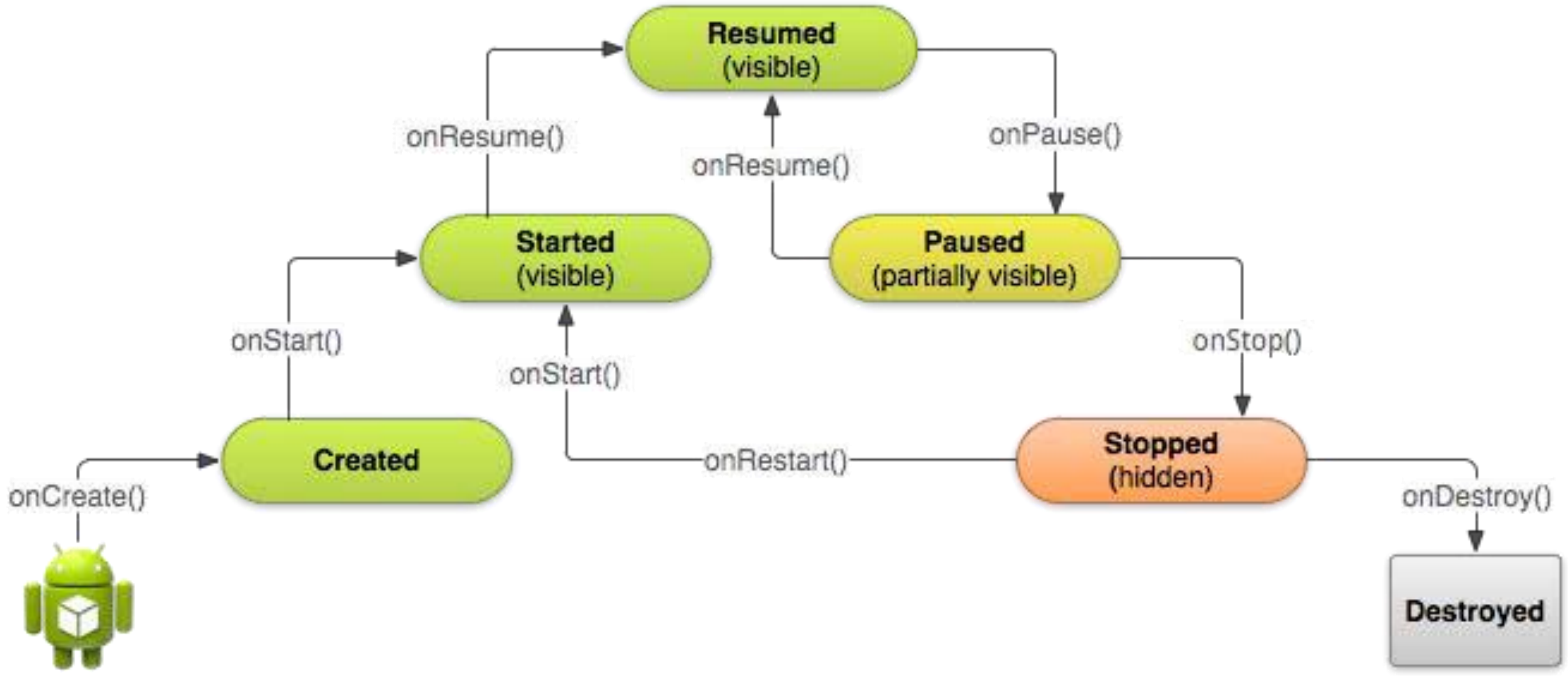
Life Cycle Events

Life Cycle States

When progressing from one state to the other, the OS notifies the application of the changes by issuing calls to the following protected *transition methods*:

void onCreate(Bundle *savedInstanceState*) **void onStart()** **void onRestart()** **void onResume()**	**void onPause()** **void onStop()** **void onDestroy()**

Application's Life Cycle



Reference:
http://developer.android.com/reference/android/app/Activity.html

Life Cycle Callbacks

Most of your code goes here

Save your important data here

```
public class ExampleActivity extends Activity {


    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be "paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

Reference:
http://developer.android.com/reference/android/app/Activity.html

Life Cycle States

An activity has essentially three states:

1. It is ***active or running***
2. It is ***paused*** or
3. It is ***stopped*** .

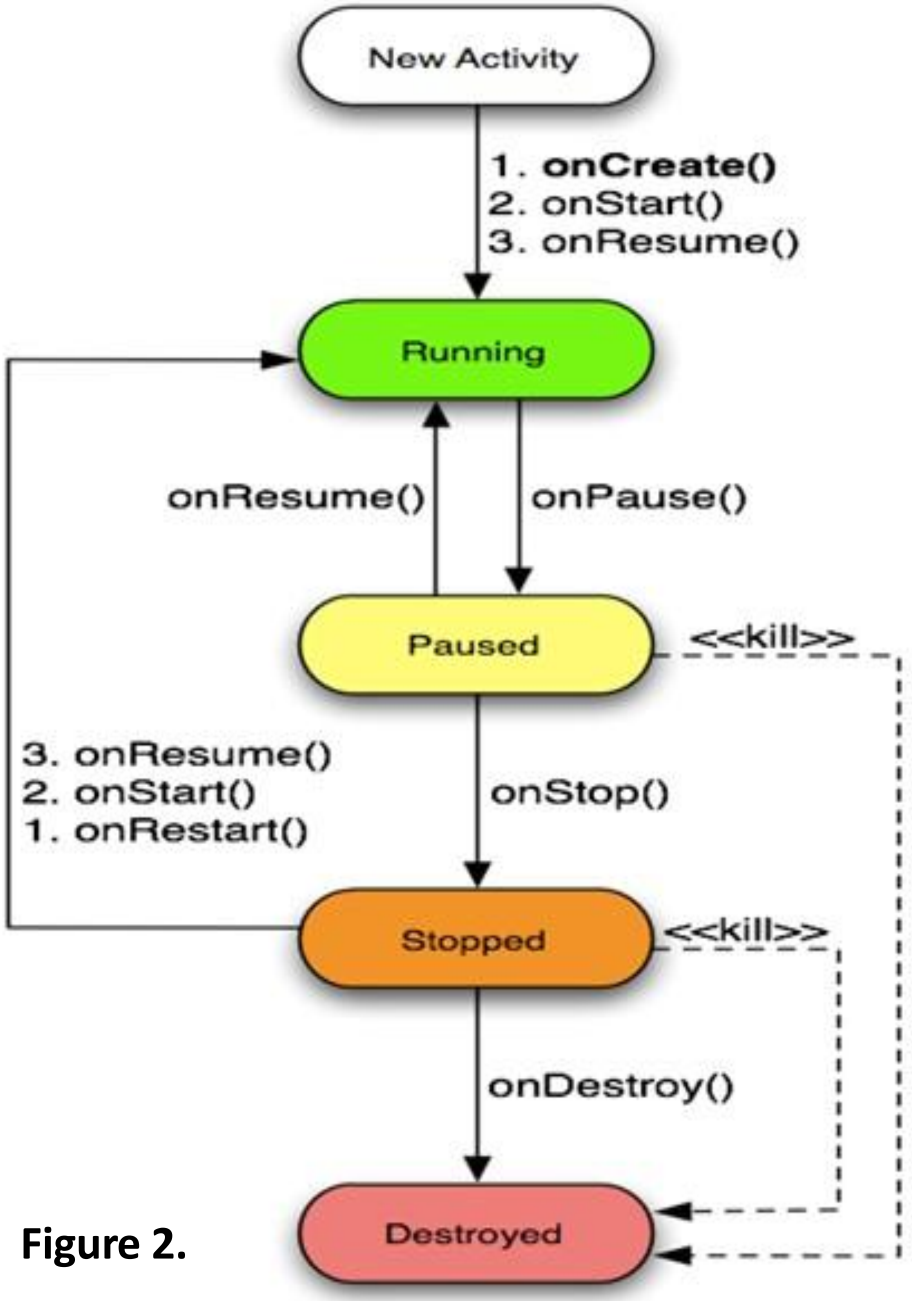


Figure 2.

Image from: http://ganiworldofandroid.blogspot.com/2011/07/complete-understanding-of-activity-life.html

Life Cycle States

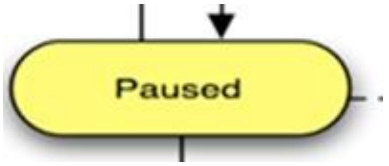
An activity has essentially three states:

1. It is ***active* or *running*** when it is in the *foreground* of the screen (at the top of the *activity stack*).

   This is the activity that has "focus" and its graphical interface is responsive to the user's interactions.

Life Cycle States

An *activity has essentially three states (cont.) :*



2. It is ***paused*** if it has lost focus but is still visible to the user.

   That is, another activity seats on top of it and that new activity either is *transparent* or *doesn't cover the full screen*.

   A paused activity is *alive* (maintaining its state information and attachment to the window manager).

   Paused activities can be killed by the system when available memory becomes extremely low.

Life Cycle States

An activity has essentially three states (cont.):

3. It is ***stopped*** if it is completely *obscured* by another activity.

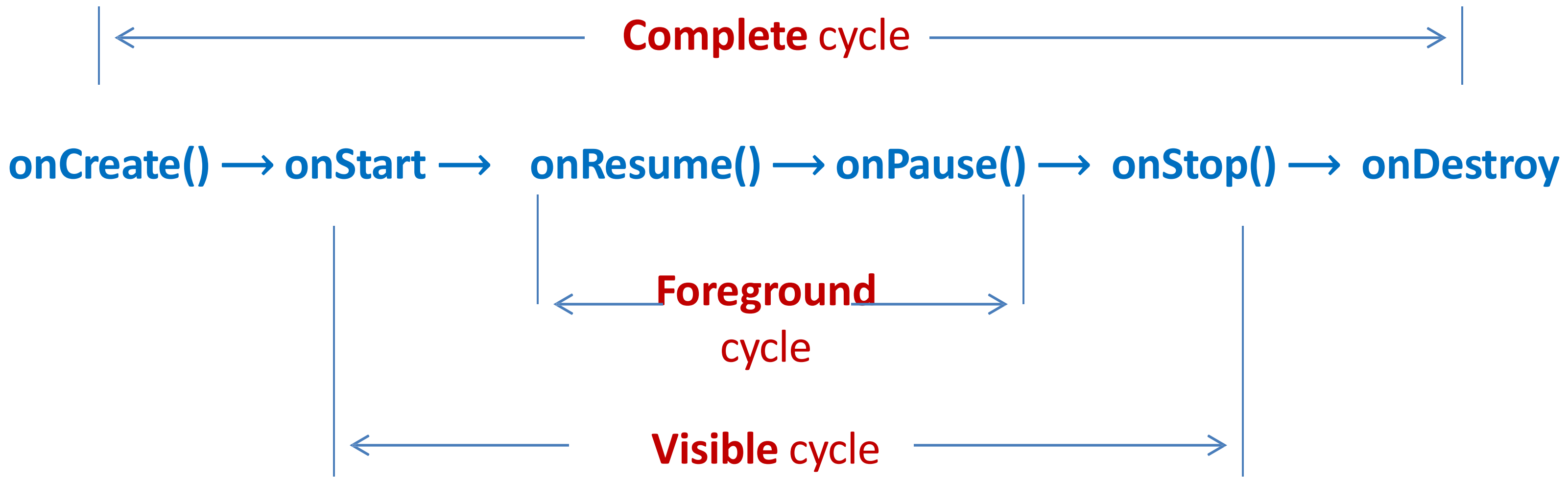
   Continues to retains all its state information.

   It is no longer visible to the user (its window is hidden and its life cycle could be terminated at any point by the system if the resources that it holds are needed elsewhere).

Application’s Lifetime

Complete / Visible / Foreground Lifetime

- An activity begins its lifecycle when entering the **onCreate()** state .
- If not interrupted or dismissed, the activity performs its job and finally terminates and releases its acquired resources when reaching the **onDestroy()** event.


Complete cycle
onCreate() ⟶ onStart ⟶ onResume() ⟶ onPause() ⟶ onStop() ⟶ onDestroy
Foreground cycle
Visible cycle

Life Cycle Events

Associating Lifecycle Events with Application's Code

Applications do not need to implement each of the transition methods, however there are mandatory and recommended states to consider

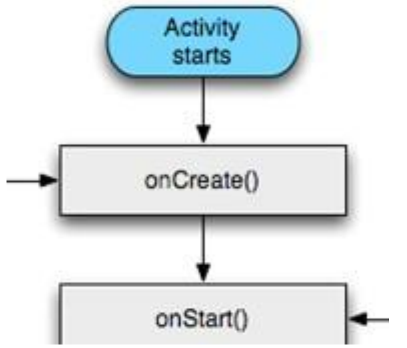
(**Mandatory**)
All activities must implement **onCreate()** to do the initial setup when the object is first instantiated.

(**Highly Recommended**)
Activities should implement **onPause()** to commit data changes in anticipation to stop interacting with the user.

Life Cycle Methods

Method: **onCreate()**

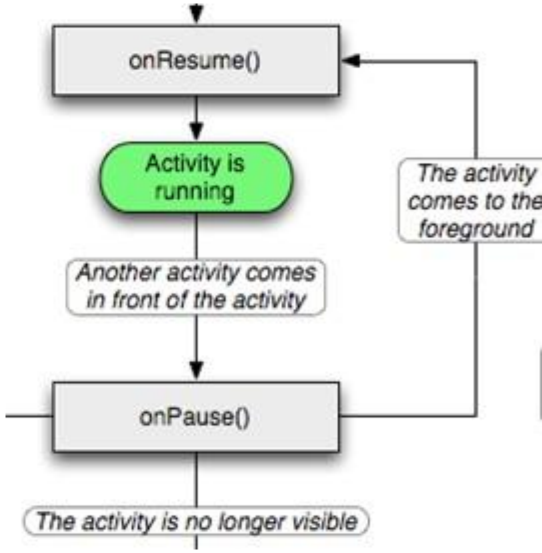
- Called when the activity is first created.
- Most of your application's code is written here.
- Typically used to define listener's behavior, initialize data structures, wire-up UI view elements (buttons, text boxes, lists) with local Java controls, etc.
- It may receive a data *Bundle* object containing the activity's previous state (if any).
- Followed by *onStart*()



Life Cycle Methods

Method: **onPause()**

1. Called when the system is about to transfer control to another activity.
2. Gives you a chance to *commit* unsaved data, and stop work that may unnecessarily burden the system.
3. The next activity waits until completion of this state.
4. Followed either by *onResume*() if the activity returns back to the foreground, or by *onStop*() if it becomes invisible to the user.
5. A paused activity could be *killed* by the system.


onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
onPause()
The activity is no longer visible

Life Cycle Methods

Killable States

- Activities on killable states can be terminated by the system when memory resources become critically low.
- Methods: `onPause()`, `onStop()`, and `onDestroy()` are *killable.*
- `onPause()` is the only state that is *guaranteed* to be given a chance to complete before the process is killed.
- You should use `onPause()` to write any pending persistent data.

Life Cycle Methods

As an aside…

Android Preferences

Preferences is a simple Android *persistence mechanism* used to store and retrieve **<key,value>** pairs, where **key** is a string and **value** is a primitive data type. Similar to a Java HashMap. Appropriate for storing small amounts of state data.

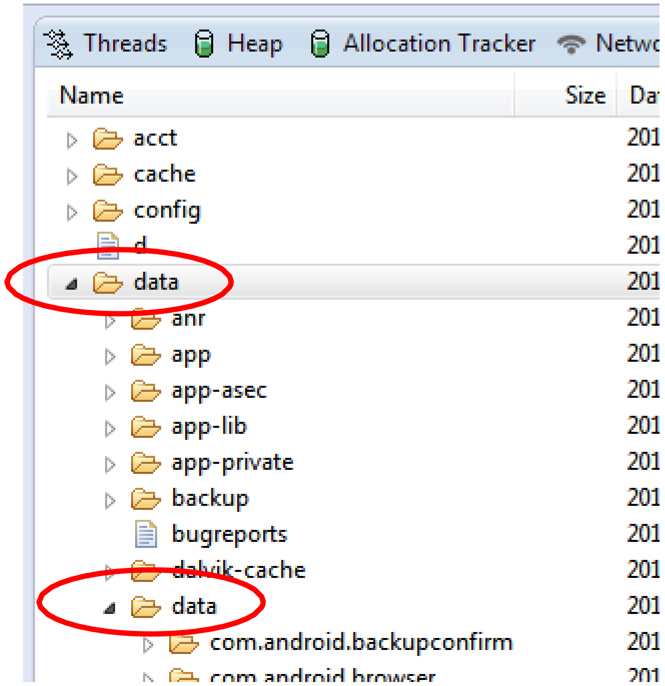
```
SharedPreferences myPrefSettings =
       getSharedPreferences(MyPreferrenceFile, actMode);
```

- A named *preferences file* could be shared with other components in the *same* application.
- actMode set to `Activity.MODE_PRIVATE` indicates that you cannot share the file across applications.

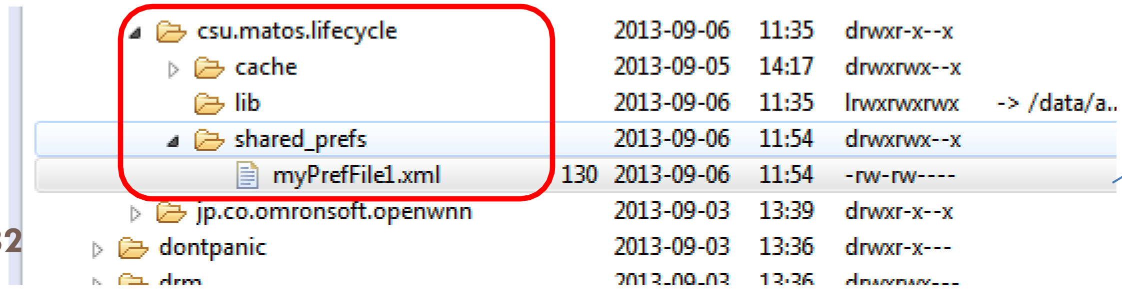
As an aside…

Android Preferences

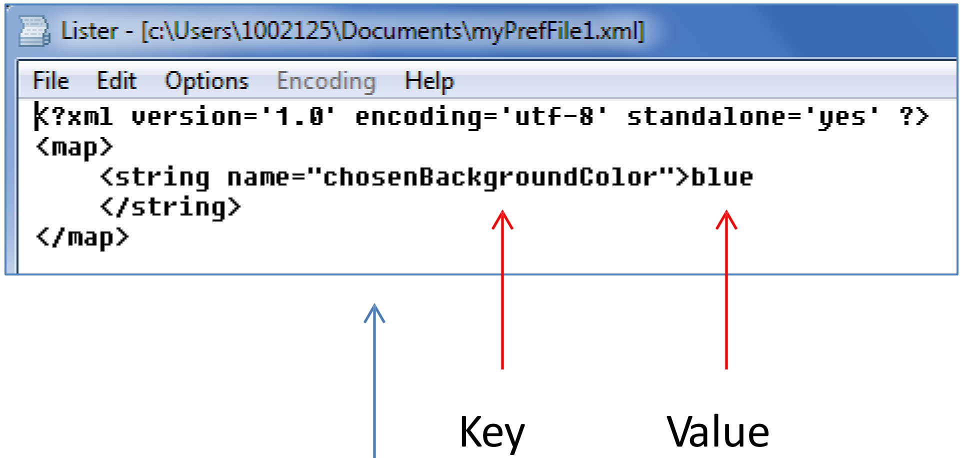
SharedPreference files are permanently stored in the application's process space.
Use DDMS file explorer to locate the entry:
data/data/your-package-name/shared-prefs



```
Name                         Size  Da
▷ acct                             201
▷ cache                            201
▷ config                           201
  d                                201
◢ data                             201
  ▷ anr                            201
  ▷ app                            201
  ▷ app-asec                       201
  ▷ app-lib                        201
  ▷ app-private                    201
  ▷ backup                         201
    bugreports                     201
  ▷ dalvik-cache                   201
  ◢ data                           201
    ▷ com.android.backupconfirm    201
    ▷ com.android.browser          201
```



```
◢ csu.matos.lifecycle           2013-09-06  11:35  drwxr-x--x
  ▷ cache                       2013-09-05  14:17  drwxrwx--x
    lib                         2013-09-06  11:35  lrwxrwxrwx  -> /data/a..
  ◢ shared_prefs                2013-09-06  11:54  drwxrwx--x
      myPrefFile1.xml      130  2013-09-06  11:54  -rw-rw----
▷ jp.co.omronsoft.openwnn       2013-09-03  13:39  drwxr-x--x
▷ dontpanic                     2013-09-03  13:36  drwxr-x---
▷ drm                           2013-09-03  13:36  drwxrwx---
```



```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="chosenBackgroundColor">blue
    </string>
</map>
```

Activity Stack

- Activities in the system are scheduled using an ***activity stack***.
- When a new activity is *started*, it is placed on *top* of the stack to become the *running* activity
- The previous activity is pushed-down one level in the stack, and may come back to the foreground once the new activity finishes.
- If the user presses the *Back Button* the current activity is terminated and the next activity on the stack moves up to become active.

Activity Stack

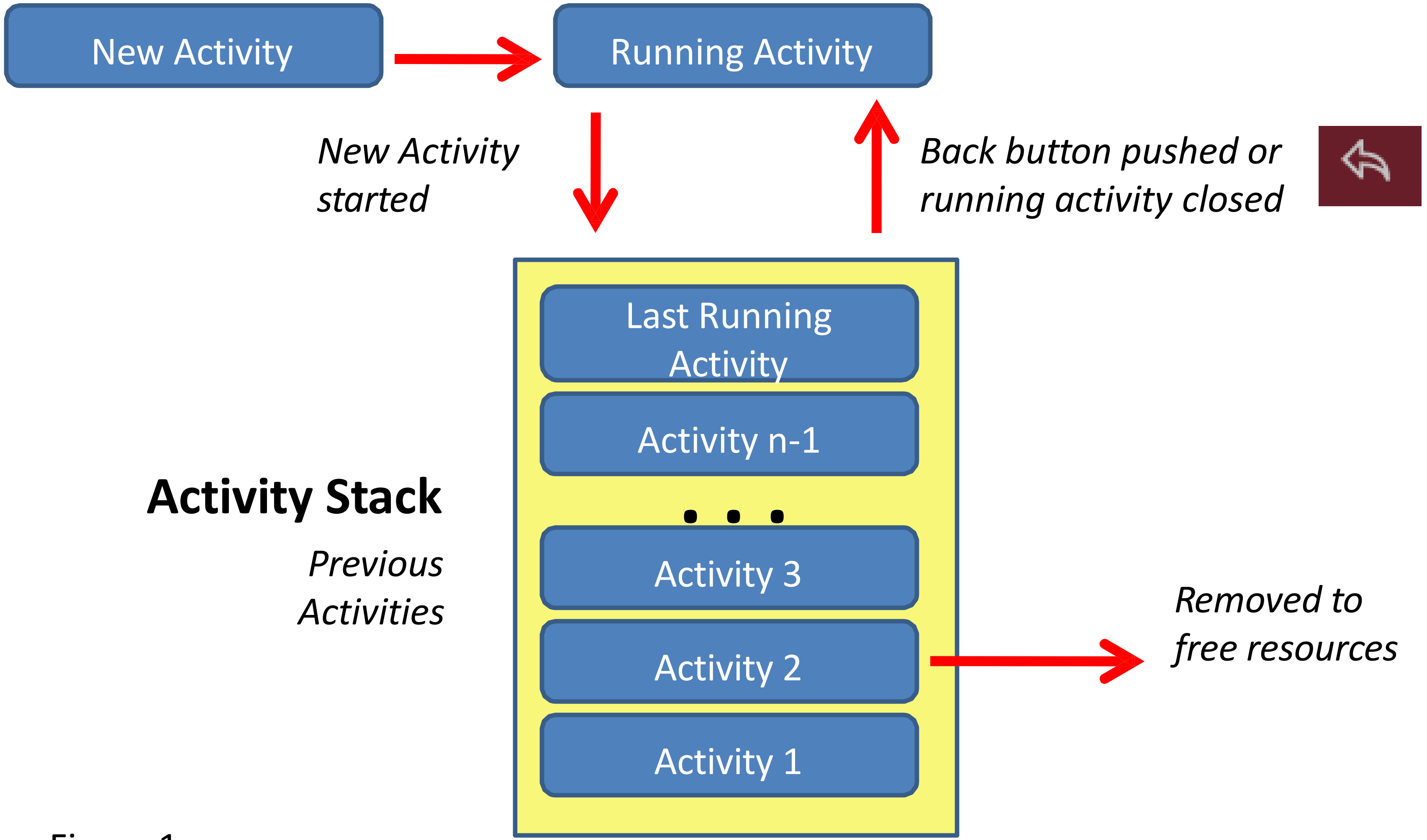


Figure 1.

Activity --- example 2 Activity classes

```
public class LabA3_4521Activity extends Activity {
        /** Called when the activity is first created. */
        @Override public void onCreate(Bundle savedInstanceState) {      super.onCreate(savedInstanceState);


                    //YOUR CODE HERE --here we setup to generate
                     // interface from layout main.xml file
                     setContextView(R.layout.main);
            }


        //OTHER METHODS OVERRIDDEN OR YOUR OWN
 }


 //another Activity class
class public class MyListActivity extends ListActivity { **********CODE HERE ******* }
```

First Activity Calling second

```
public class LabA3_4521Activity extends Activity {
        /** Called when the activity is first created. */
        @Override public void onCreate(Bundle savedInstanceState) {     super.onCreate(savedInstanceState);

                //YOUR CODE HERE --here we setup to generate
                 // interface from layout main.xml file
                 setContextView(R.layout.main);
           }

        //OTHER METHODS OVERRIDDEN OR YOUR OWN
        //start the other Activity --- possibly triggered by some GUI event in this
        //LabA3_45221Activity
        public void startListActivity() {
              startActivity(new Intent("packagename.MyListActivity");

             }
 }
 //another Activity class
class public class MyListActivity extends ListActivity { **********CODE HERE ******* }
```

Asking to Kill an Active Activity

```
//another Activity class
class public class MyListActivity extends ListActivity {
        //**********CODE HERE *******


    //method that may be called on this Activity
    // maybe as result of hitting a "end" button that is in the GUI of this
    Activity
    // so the End button's OnClickListener will have
    // its onClick method call endThisActivity()
     public void endThisActivity(){
             finish();
      }
}
```

finish() is a method inherited from Activity
will request that this activity be closed