Chapter 3:
Activities and Fragments

Outline

- Understanding fundamental concepts in Android UI
- To understand the concepts of ***activity***
- To link activity with ***intent***
- To understand the concept of ***fragment***
- To understand the concept of ***view***

Understanding Android UI

- Activity
- Fragment
- Views and ViewGroups (Topic 5)

- Activities are linked up by *intent* (Topic 4)

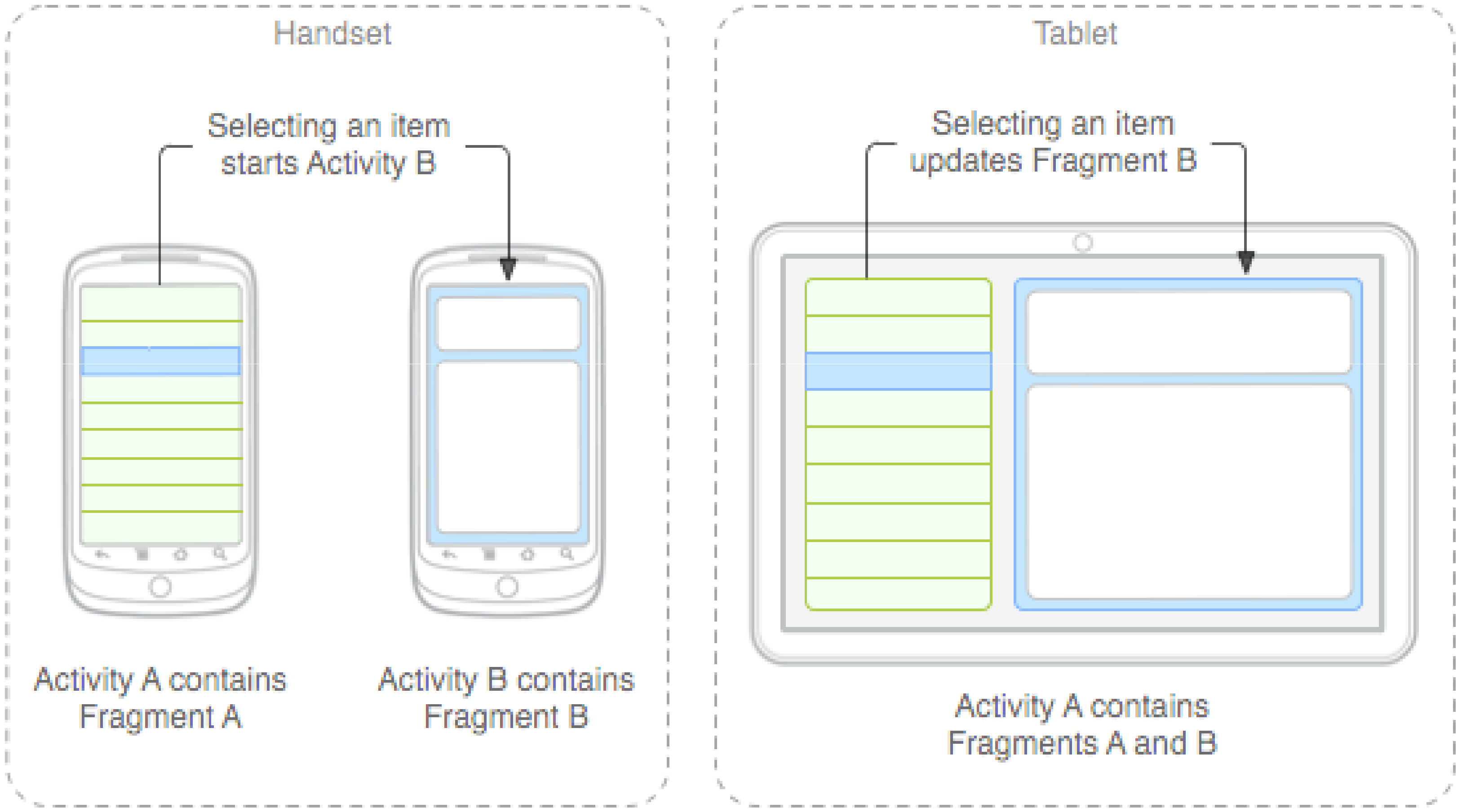
Introduction to **Activities**

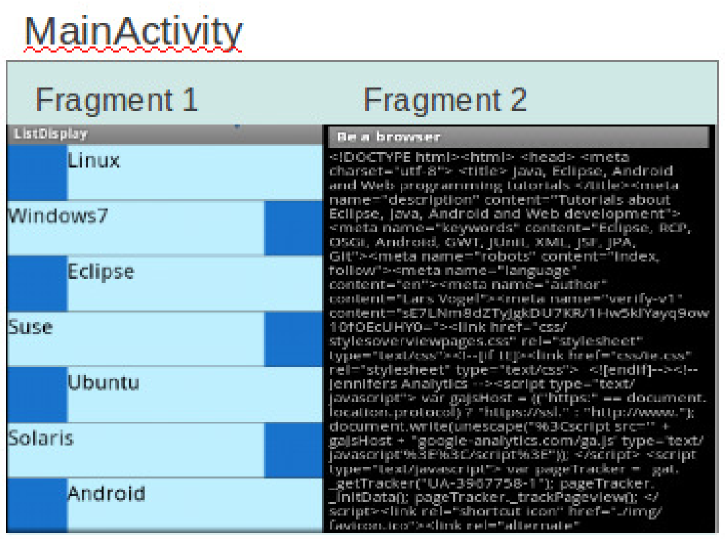
- Application component that provides a **screen** (user interface) for user interaction, such as dial the phone, take a photo, send an email, or view a map - “window”
- An application can have zero or more activities, typically 1 or more
- **Activities** can move into the background and then be resumed with their state restored
- Activities use **Views** and **Fragments** to create the user interface and to interact with the user. (Topics 4 and 5)

Introduction to Fragments

- **Fragments** are components which run in the context of an Activity.
- Used to allow the reuse of your user interface components on different sized devices.
- Make use of the Fragment class to better modularize their code, build more sophisticated user interfaces for larger screens, and help scale their application between small and large screens

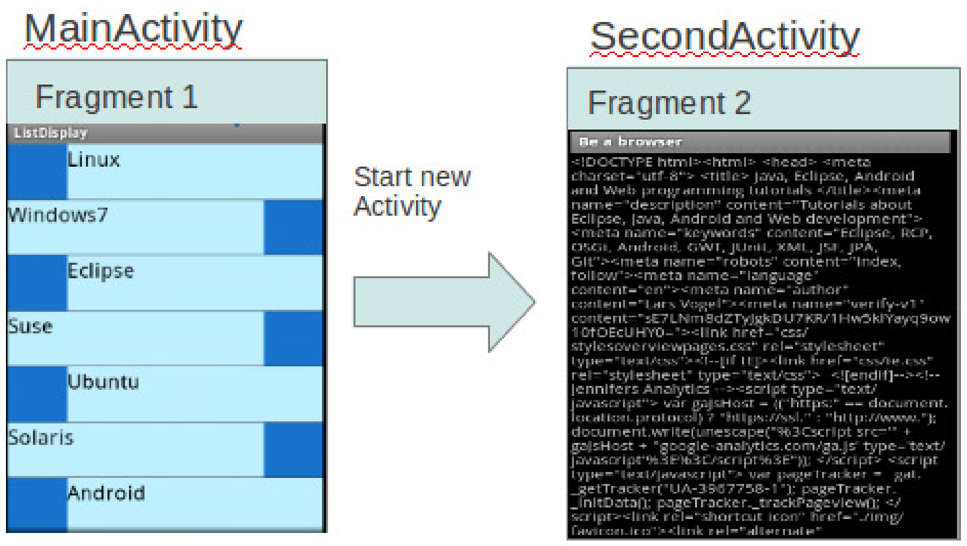
Fragments


Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B
Tablet
Selecting an item updates Fragment B
Activity A contains Fragments A and B



```
<!DOCTYPE html><html> <head> <meta
charset="utf-8"> <title> Java, Eclipse, Android
and Web programming tutorials </title><meta
name="description" content="Tutorials about
Eclipse, Java, Android and Web development">
<meta name="keywords" content="Eclipse, RCP,
OSGi, Android, GWT, JUnit, XML, JSF, JPA,
Git"><meta name="robots" content="index,
follow"><meta name="language"
content="en"><meta name="author"
content="Lars Vogel"><meta name="verify-v1"
content="sE7LNm8dZTyJgkDU7KR/1Hw5klYayq9ow
10fOEcUHY0="><link href="css/
stylesoverviewpages.css" rel="stylesheet"
type="text/css"><!--[if IE]><link href="css/ie.css"
rel="stylesheet" type="text/css"> <![endif]--><!--
Jennifers Analytics --><script type="text/
javascript"> var gaJsHost = (("https:" == document.
location.protocol) ? "https://ssl." : "http://www.");
document.write(unescape("%3Cscript src='" +
gaJsHost + "google-analytics.com/ga.js' type='text/
javascript'%3E%3C/script%3E")); </script> <script
type="text/javascript"> var pageTracker = _gat.
_getTracker("UA-3967758-1"); pageTracker.
_initData(); pageTracker._trackPageview(); </
script><link rel="shortcut icon" href="./img/
favicon.ico"><link rel="alternate"
```

Wide screen, e.g. tablet



```
<!DOCTYPE html><html> <head> <meta
charset="utf-8"> <title> Java, Eclipse, Android
and Web programming tutorials </title><meta
name="description" content="Tutorials about
Eclipse, Java, Android and Web development">
<meta name="keywords" content="Eclipse, RCP,
OSGi, Android, GWT, JUnit, XML, JSF, JPA,
Git"><meta name="robots" content="index,
follow"><meta name="language"
content="en"><meta name="author"
content="Lars Vogel"><meta name="verify-v1"
content="sE7LNm8dZTyJgkDU7KR/1Hw5klYayq9ow
10fOEcUHY0="><link href="css/
stylesoverviewpages.css" rel="stylesheet"
type="text/css"><!--[if IE]><link href="css/ie.css"
rel="stylesheet" type="text/css"> <![endif]--><!--
Jennifers Analytics --><script type="text/
javascript"> var gaJsHost = (("https:" == document.
location.protocol) ? "https://ssl." : "http://www.");
document.write(unescape("%3Cscript src='" +
gaJsHost + "google-analytics.com/ga.js' type='text/
javascript'%3E%3C/script%3E")); </script> <script
type="text/javascript"> var pageTracker = _gat.
_getTracker("UA-3967758-1"); pageTracker.
_initData(); pageTracker._trackPageview(); </
script><link rel="shortcut icon" href="./img/
favicon.ico"><link rel="alternate"
```

Smaller screen, e.g. handheld

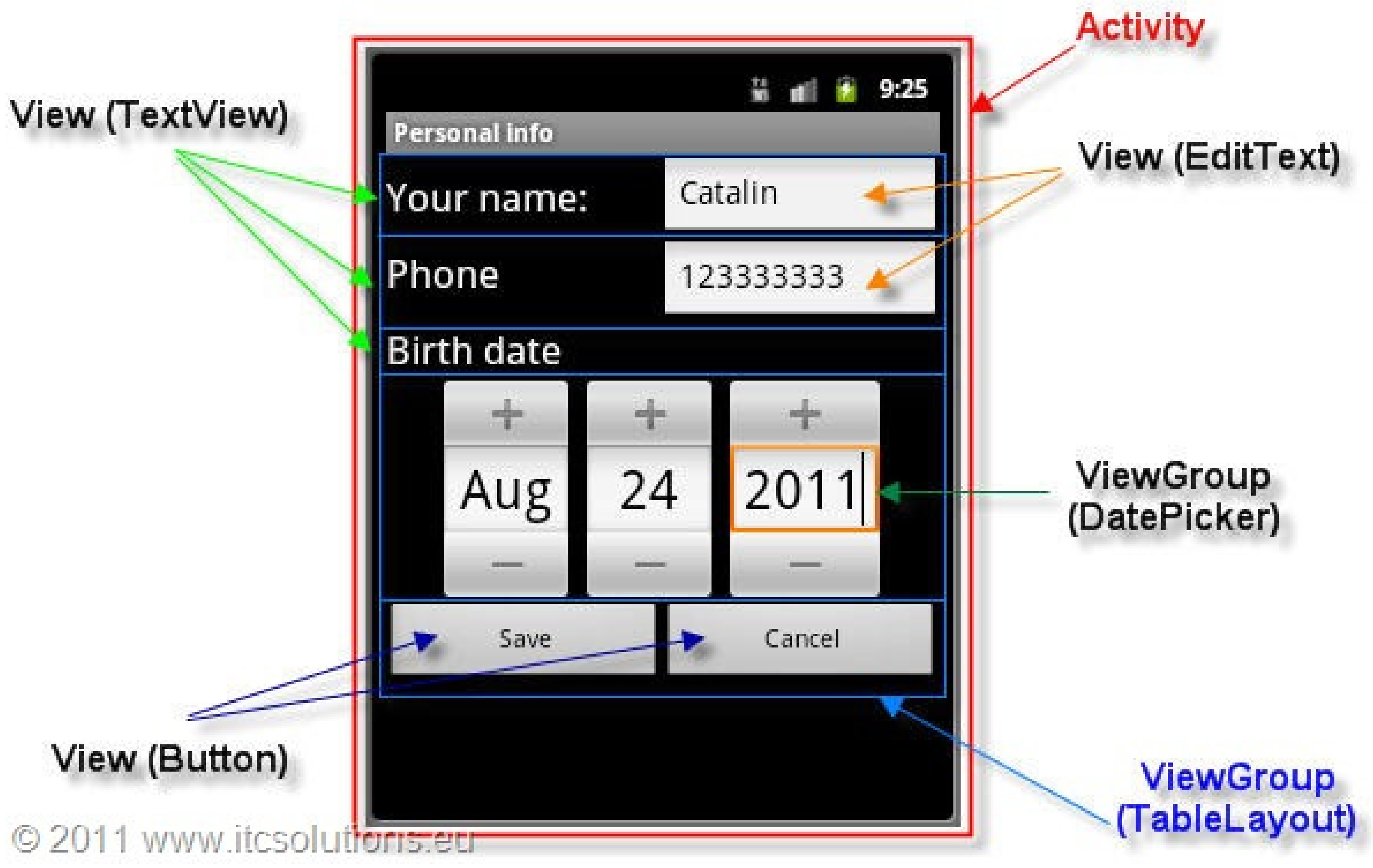
MainActivity - Fragments

- On a wide screen it shows two *Fragments*.
- On a smaller screen it shows one *Fragment* and allows that the user navigate to another *Activity* called *SecondActivity* which displays the second *Fragment*.

Introduction to **Views** and **ViewGroups**

- **Views** are user interface widgets, e.g. buttons or text fields.
  - base class： android.view.View
  - attributes ： used to configure their appearance and behavior
- **ViewGroup***:* arranging other *Views (layout managers*)
  - base class: android.view.ViewGroup (extend View class)
  - can be nestled to create complex layouts

Activity, View and ViewGroup



Activity

- **Activities** represent interfaces to the user.
- All user interactions take place through activities.
- **Activities** are defined with different layouts. These layouts can be picked based on several different factoring including the size of the actual device.
- An activity is made up of a UI component or its **layout** (e.g. *main.xml*) and a class component or its **functionality** (e.g. *MainActivity.java*).
- Understanding activities is key to creating responsive and usable applications

Activity life cycle

- Short-lived – continually being created and destroyed
- Up to developer to handle transitions between activities as user navigates an app
- Extend the Activity class (import **android.app.Activity**)
- Implement a series of callbacks that the system calls when activity transitions between states

Activity **Lifecycle**

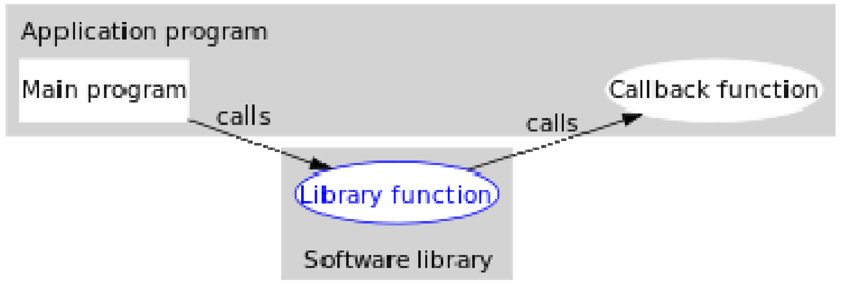
1. *Start*: “**main**” activity is presented to the user when launching the application for the first time.
2. Each activity can then start another activity in order to perform different actions.
3. Each time a new activity starts, the previous activity is stopped (gone to background), but the system preserves the activity in a "back stack” (**last in, first out**).
4. When a new activity starts, it is pushed onto the back stack and takes user focus.
5. When the user is done with the current activity and presses the *Back* button, it is popped from the stack (and destroyed) and the previous activity resumes.

Activity Lifecycle

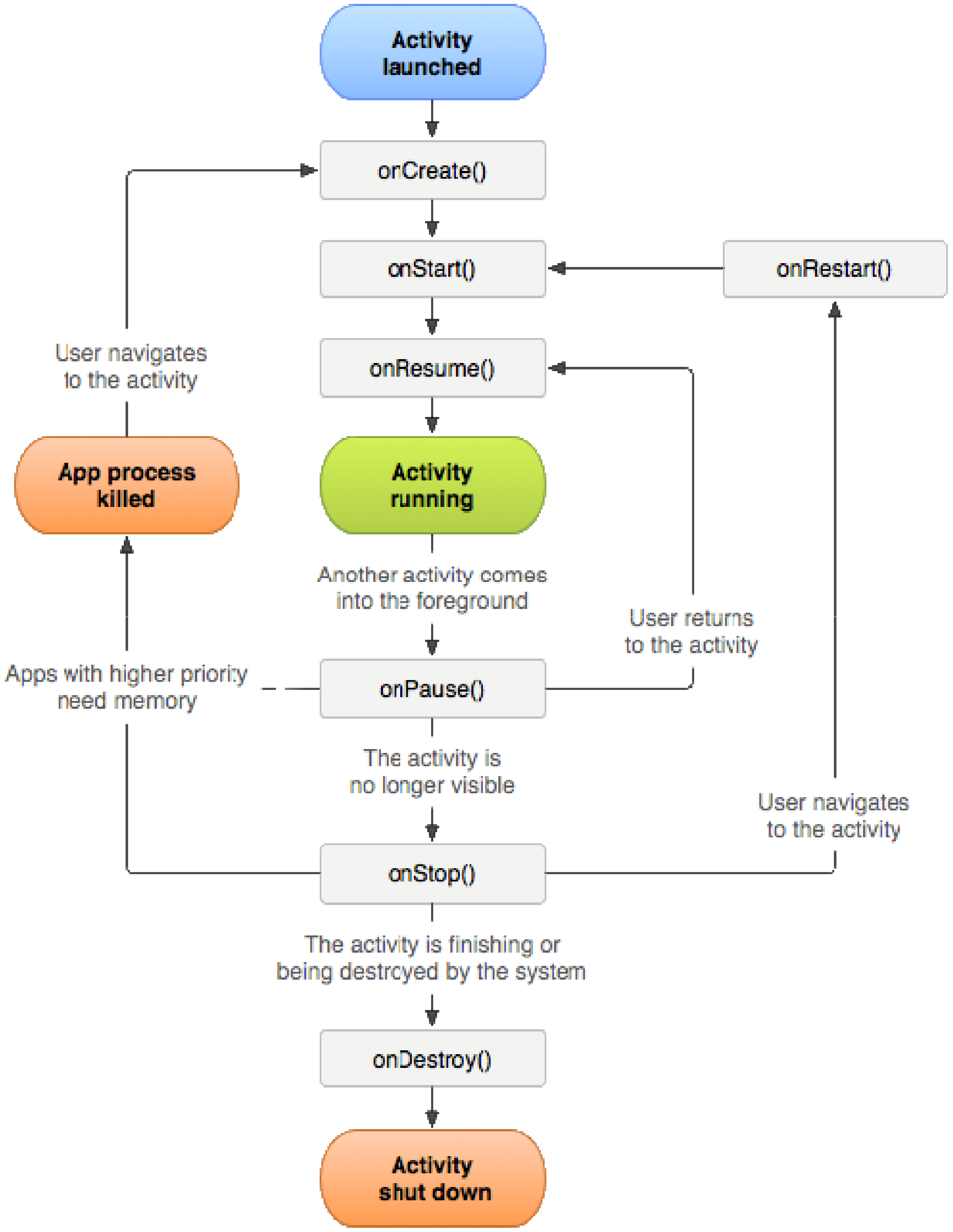
6. When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle **callback methods** due to change in its state: **create**, **stop**, **resume**, **destroy**

Activity Life Cycle

- Android system initiates code in an Activity instance by invoking specific **callback methods** that correspond to specific stages of its lifecycle
- During its lifetime, each activity of an Android program can be in one of several **states**
- Developers do not have control over what state your program is in (**managed by the system**)
- Developers only get notified when the state is about to change through the on*XX() method calls* (next few slides)
- Activities that are not running in the foreground may be stopped or the Linux process that houses them may be killed at any time in order to make room for new activities.



Activity Life Cycle


Activity launched
orCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Callback functions of an Activity base class

- The **Activity** base class defines a series of callback functions (events) that govern the life cycle of an activity
  - **onCreate()** — Called when the activity is first created
  - **onStart()** — Called when the activity becomes visible to the user
  - **onResume()** — Called when the activity starts interacting with the user
  - **onPause()** — Called when the current activity is being paused and the previous activity is being resumed
  - **onStop()** — Called when the activity is no longer visible to the user

Events of Activity base class

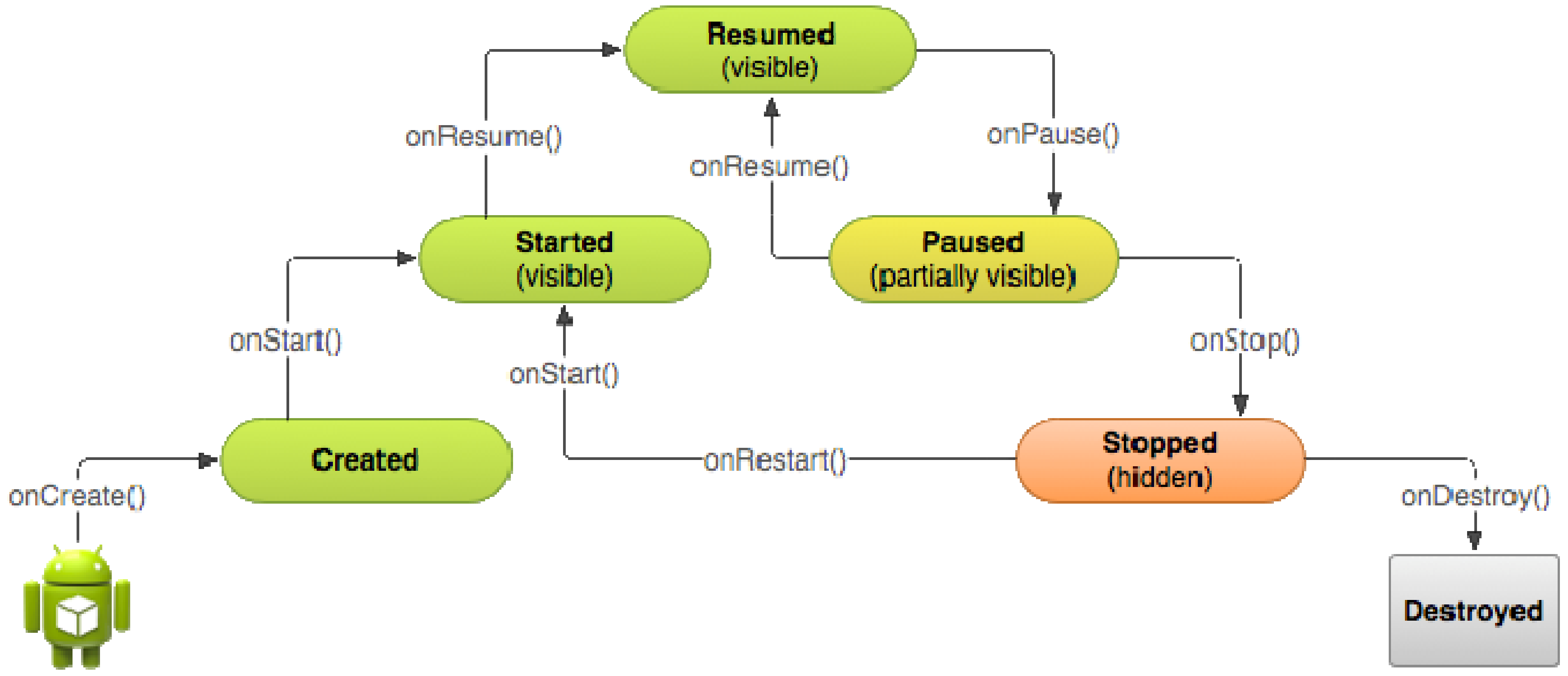
  - **onDestroy()** — Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
  - **onRestart()** — Called when the activity has been stopped and is restarting again

- You override these methods in your Activity class (.java), and Android will call them at the appropriate time

Activity states

State	Explanation
Resumed or Running	The activity is focused and visible to the user. Users interact wit the activity while it is in this state.
Paused	The Activity is still visible, but it is no longer focused. This occurs when something has popped up in front of the application, such as a dialog
Stopped	The activity is placed in this state when the user transitions to a new activity and the activity is no longer visible. The system will often destroy the activity to reclaim resources. If all activities of an app are stopped, the system will kill the entire app process to reclaim the resources.

Activity Lifecycle as a Step Pyramid



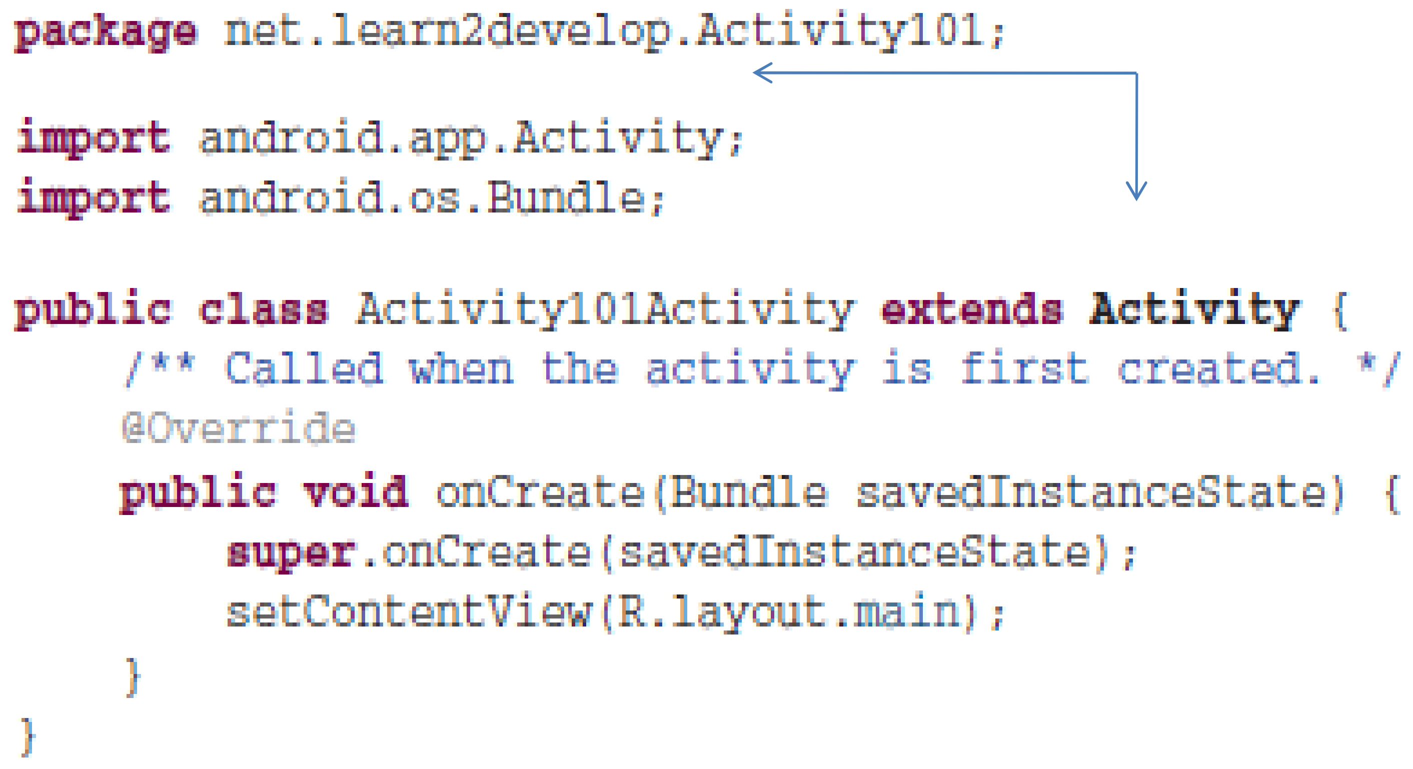
- As the system creates a new activity instance, each callback method moves the activity state one step toward the top.
- The top of the pyramid is the point at which the activity is running in the foreground and the user can interact with it.
- As the user begins to leave the activity, the system calls other methods that move the activity state back down the pyramid in order to dismantle the activity.
- In some cases, the activity will move only part way down the pyramid and wait (such as when the user switches to another app), from which point the activity can move back to the top (if the user returns to the activity) and resume where the user left off.

Rules of thumb

- Use the **onCreate()** method to create and instantiate the objects that you will be using in your application.
- Use the **onResume()** method to start any services or code that needs to run while your activity is in the foreground.
- Use the **onPause()** method to stop any services or code that does not need to run when your activity is not in the foreground.
- Use the **onDestroy()** method to free up resources before your activity is destroyed.

Creating an activity

- Create Java class that extends the **Activity** base class (in src/net/learn2develop/Activity101.java)



```
package net.learn2develop.Activity101;

import android.app.Activity;
import android.os.Bundle;

public class Activity101Activity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Declaring MAIN Activity in the manifest

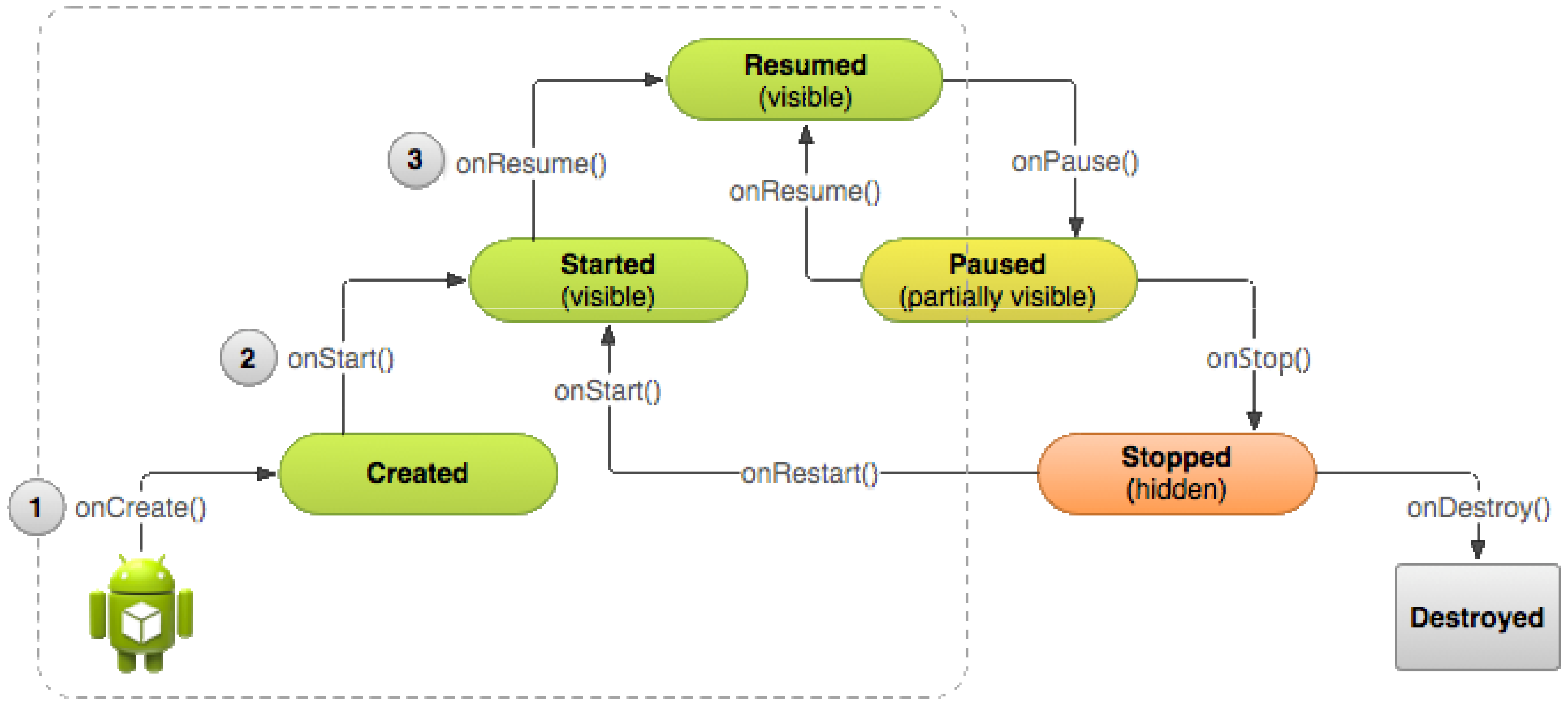
- Specify the App's Launcher (Main) activity
- The **main** activity for your app must be declared in the manifest with an *<intent-filter>* that includes the MAIN action and LAUNCHER category (in AndroidManifest.xml)
- When the user selects your app icon from the Home screen, the system calls the **onCreate()** method for the Activity in your app that you've declared to be the "launcher" (or "main") activity (main entry point to app UI)

```xml
<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>
```

Starting an Activity

- Implement the **onCreate()** method to perform basic application startup logic that should happen only once for the entire life of the activity e.g.
  - define the user interface in an XML layout file (e.g. main.xml)
  - instantiate class-scope variables
- Once the onCreate() finishes execution, the system calls the **onStart()** and **onResume()** methods in quick succession. Hence, the activity never resides in the Created or Started states.
- The activity becomes visible to the user when onStart() is called, but onResume() quickly follows and the activity remains in the **Resumed** state until something occurs to change that, such as when a phone call is received, the user navigates to another activity, or the device screen turns off.

3 main callbacks that the system calls in sequence when creating a new instance of the activity: **onCreate()**, **onStart()** and **onResume()**



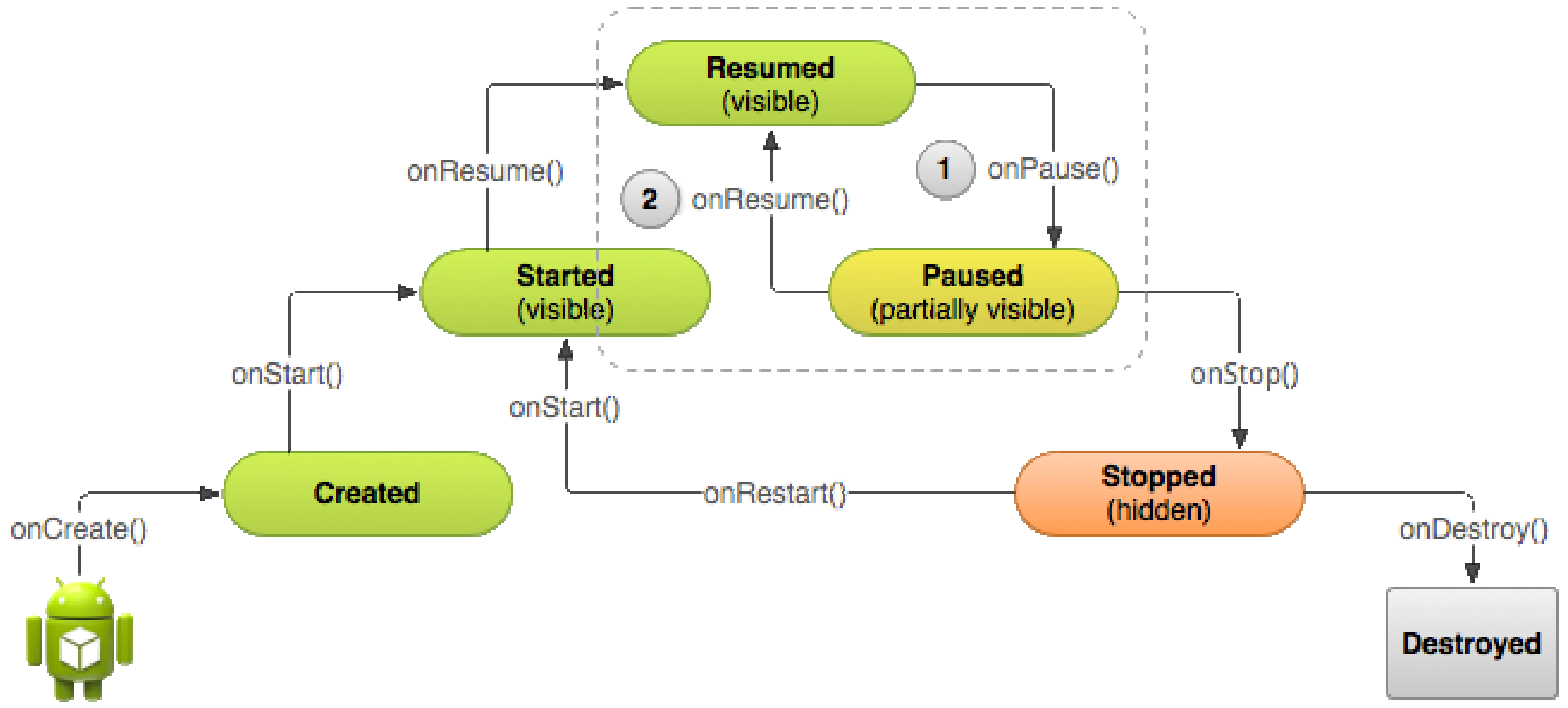
Pausing and Resuming an Activity

- Foreground activity is sometimes obstructed by other visual components that cause the activity to *pause*.
- **Paused** state: The activity is partially visible but currently not the activity in focus
- **Stopped** state: The activity is fully-obstructed and not visible

Pausing and Resuming an Activity

- As an activity enters the Paused state, the system calls the onPause() method on the Activity, which allows you to stop ongoing actions that should not continue while paused (such as a playing video) or persist any information that should be permanently saved in case the user continues to leave your app.
- If the user returns to your activity from the Paused state, the system resumes it and calls the onResume() method.

Pausing and Resuming an Activity



Pausing an Activity

- You should usually use the onPause() callback to:
  - Stop animations or other ongoing actions that could consume CPU (e.g. playing video)
  - Commit unsaved changes, but only if users expect such changes to be permanently saved when they leave (such as a draft email)
    - E.g. any text entered by the user in an email app should be saved to database during onPause callback
  - Release system resources, such as broadcast receivers, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them
- You should keep the amount of operations done in the onPause() method relatively simple to allow for a speedy transition to the user's next activity

Resuming an Activity

- When the user resumes your activity from the Paused state (activity comes into the foreground, including when it's created for the first time), the system calls the **onResume()** method.
- You should implement onResume() to **initialize components** that you release during onPause() and perform other initializations that must occur each time the activity enters the Resumed state

```
@Override
public void onPause() {
    super.onPause();  // Always call the superclass method first

    // Release the Camera because we don't need it when paused
    // and other activities might need to use it.
    if (mCamera != null) {
        mCamera.release()
        mCamera = null;
    }
}
```

```
@Override
public void onResume() {
    super.onResume();  // Always call the superclass method first

    // Get the Camera instance as the activity achieves full user focus
    if (mCamera == null) {
        initializeCamera(); // Local method to handle camera init
    }
}
```

Stopping and Restarting an Activity

- Scenarios in which your activity is stopped and restarted:
  - The user opens the Recent Apps window and switches from your app to another app. The activity in your app that's currently in the foreground is stopped. If the user returns to your app from the Home screen launcher icon or the Recent Apps window, the activity restarts.
  - The user performs an action in your app that starts a new activity. The current activity is stopped when the second activity is created. If the user then presses the *Back* button, the first activity is restarted.
  - The user receives a phone call while using your app on his or her phone.
- Should release almost all resources (that might leak memory) that aren't needed while the user is not using it

Stopped vs. Paused State

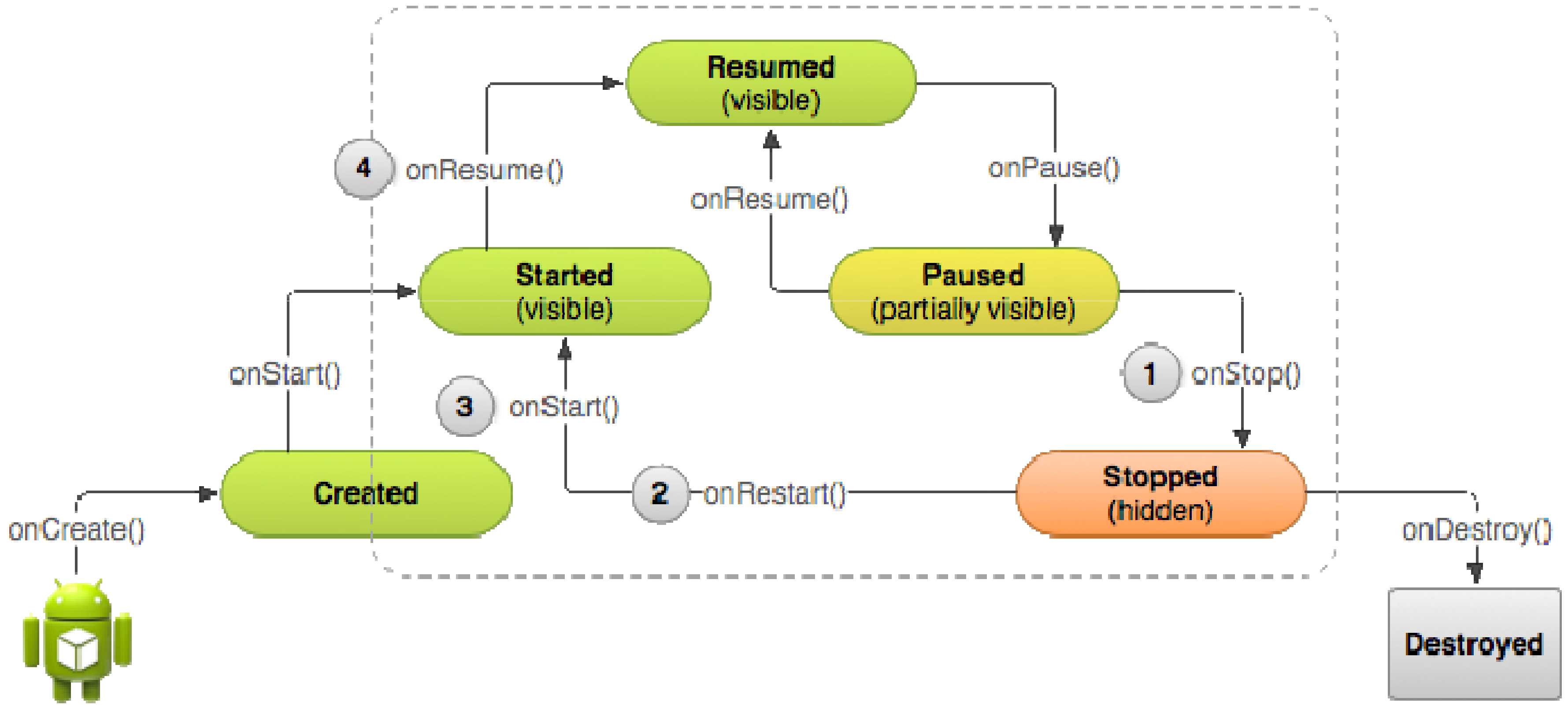
- **Stopped** state: Activity UI is no longer visible and the user's focus is in a separate activity (or an entirely separate app)
- **Paused** state: The activity is partially visible but currently not the activity in focus

Stop an Activity

- The system retains your Activity instance in system memory when it is stopped
- It is possible that we don't need to implement the onStop() and onRestart() (or even onStart()) methods at all
- Use onStop() to perform larger, more CPU intensive shut-down operations, such as writing information to a database

Restart an Activity

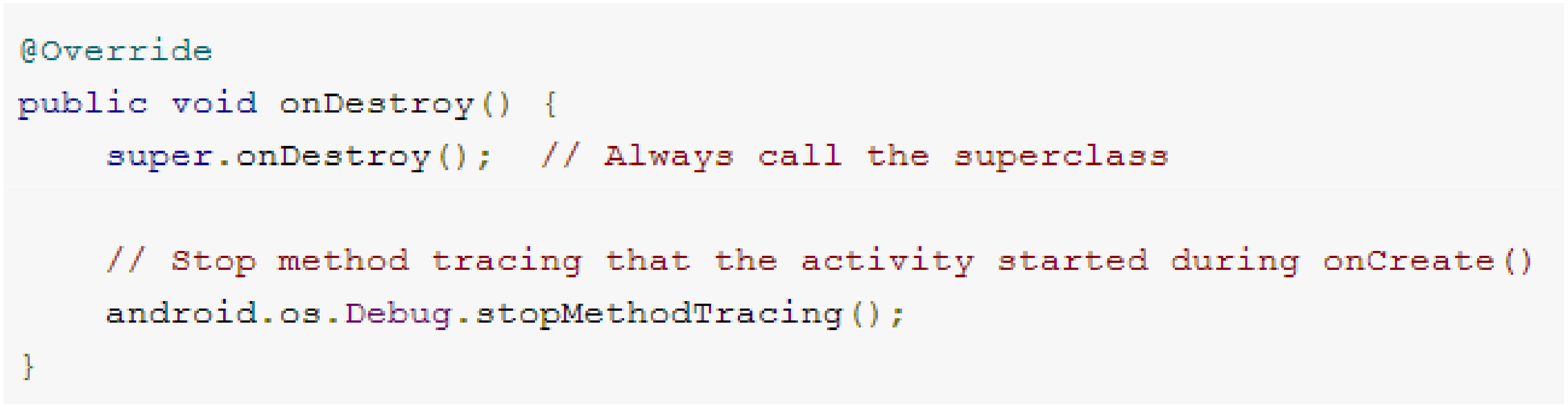
- Activity comes back to the foreground or first time created
- **onRestart():** is called only when the activity resumes from the Stopped state, not from Destroyed state
- **onStart()**: is called when an activity is *created* and when the activity is *restarted* from the stopped state


Resumed
(visible)
4
onResume()
onResume()
onPause()
Started
(visible)
Paused
(partially visible)
onStart()
3
onStart()
1
onStop()
Created
2
onRestart()
Stopped
(hidden)
onCreate()
onDestroy()
Destroyed

Shutting Down / Destroy an Activity

- An activity should perform most cleanup during onPause() and onStop().
- Hence, most apps don't need to implement onDestroy() (final signal that your activity instance is being completely removed from the system memory)
- However, if your activity includes background threads that you created during onCreate() or other long-running resources that could potentially leak memory if not properly closed, you should kill them during onDestroy()

Shutting Down / Destroy an Activity



```
@Override
public void onDestroy() {
    super.onDestroy();  // Always call the superclass

    // Stop method tracing that the activity started during onCreate()
    android.os.Debug.stopMethodTracing();
}
```

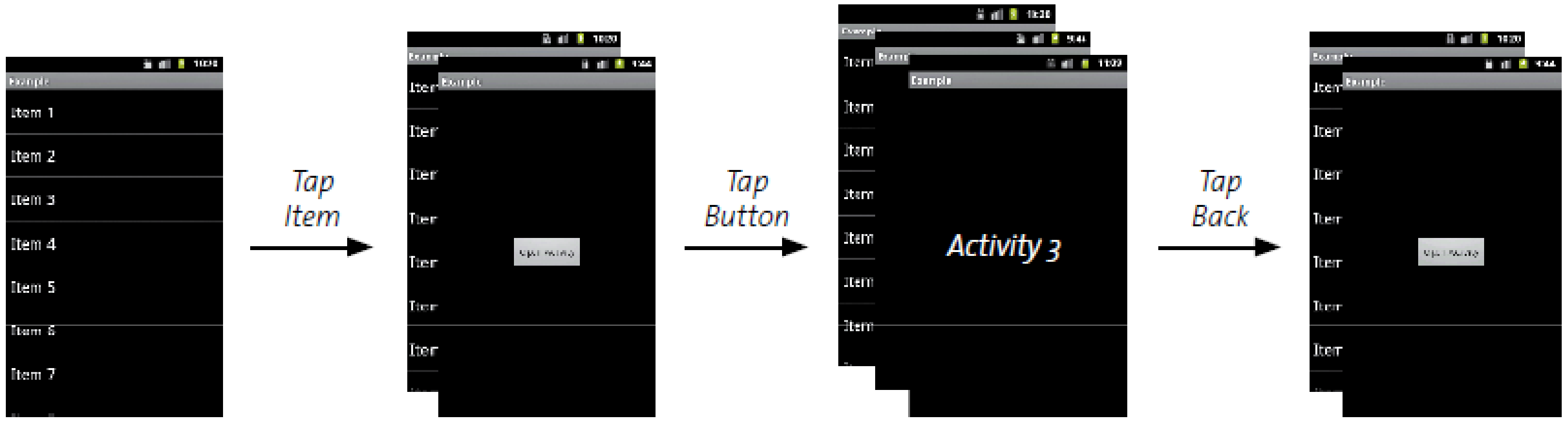
Application or Activity ≠ Process

- An application is one or more activities plus a Linux process to contain them.
- By default, all components of the same application run in the same process and thread (called the "main" thread).
- If an application component starts and there already exists a process for that application (because another component from the application exists), then the component is started within that process and uses the same thread of execution.
- However, we can arrange for different components in an application to run in separate processes.
- An application can be “alive” even if its process has been killed.
- **The activity life cycle is not tied to the process life cycle.**

Tasks and the Back Stack

- Every task has its own **back stack**.
- Android system groups these activities into tasks. Each task represents a set of activities as a stack
- activities being **pushed** onto the stack when the user navigates away from them (become invisible or inactive) and
- being **popped off** the stack when the user navigates back to them (become visible or active)
- New tasks are created when the user opens a new activity that is not associated with the current activity.

Tasks and the Back Stack

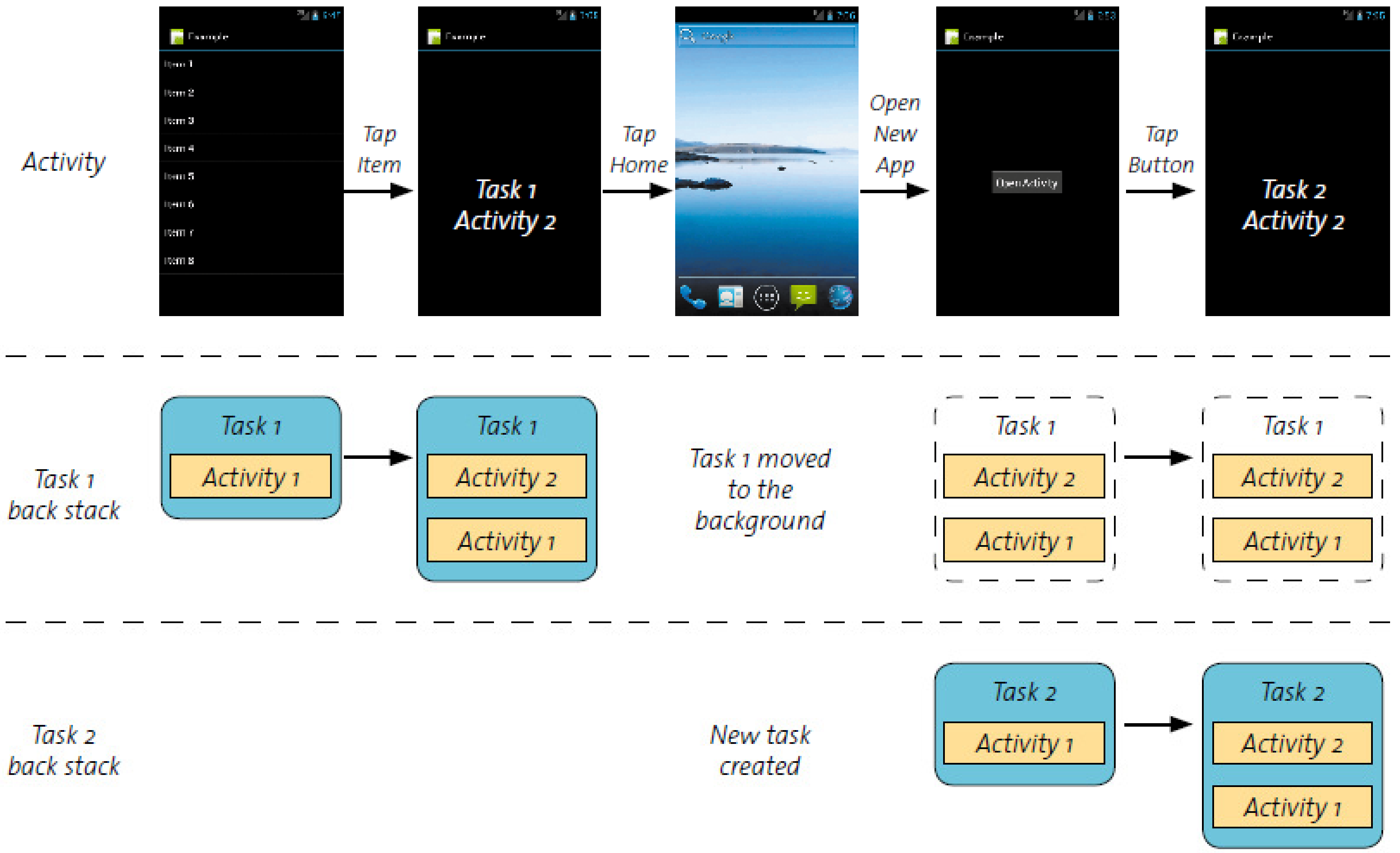


- Pressing the Back button will pop the most recent activity off the stack.

Example

1. 1.A user opens an application. This creates a new task. The example is a ListView.
2. The user navigates to a new activity by pressing a list item.
3. The user presses Home, then opens a new app. This creates a second task, containing the main activity of the new app.
4. The user navigates to a new activity in this task, again by pressing a list item.

Two tasks and their back stacks

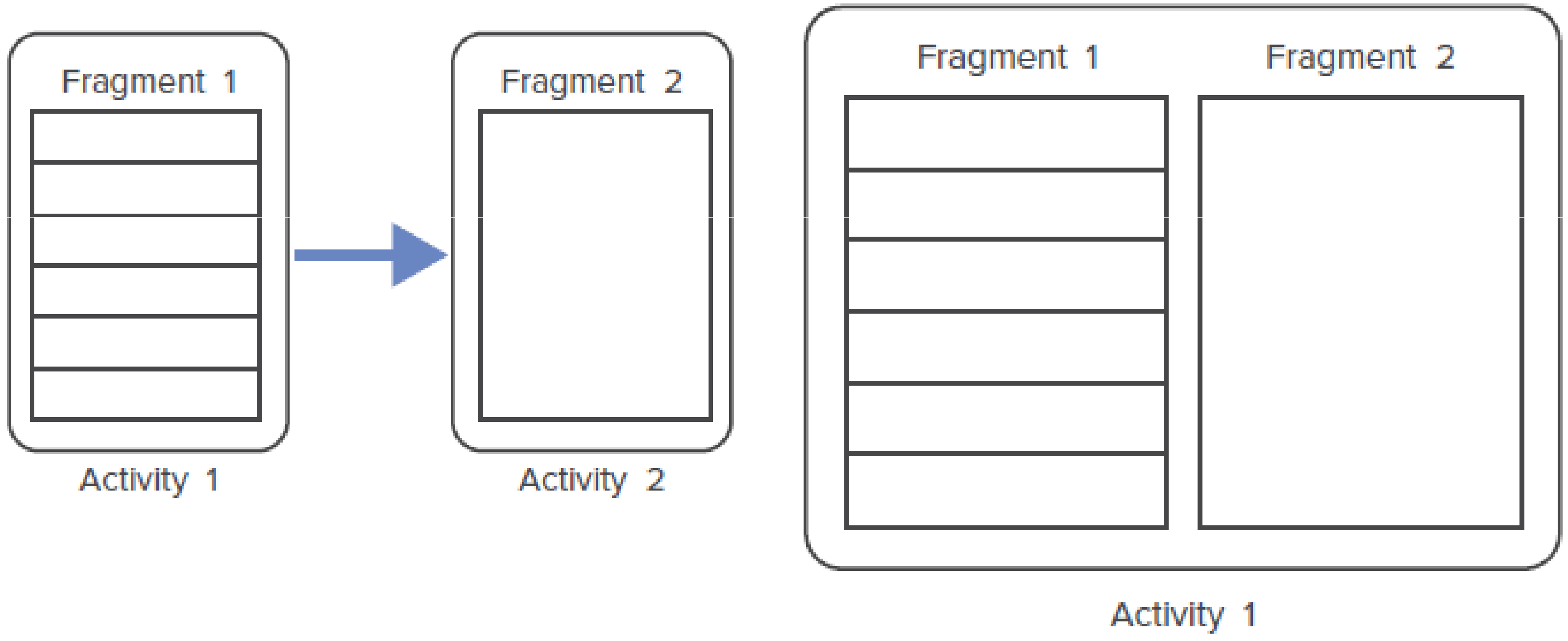


- The user can switch between the two tasks by pressing Home and tapping one of the application launchers.
- Alternatively, on Android 4.0 and later, users can press the task switcher button to switch tasks. The Back button will act on the active stack and pop the topmost activity from the task the user is viewing.
- It is possible for the same activity to appear multiple times in the back stack. This occurs when the same activity can be started from multiple places.
- You should watch for these situations, because you could easily consume large amounts of memory storing multiple copies of the same activity. This will also be annoying to users, because they will have to press Back repeatedly to exit your app.

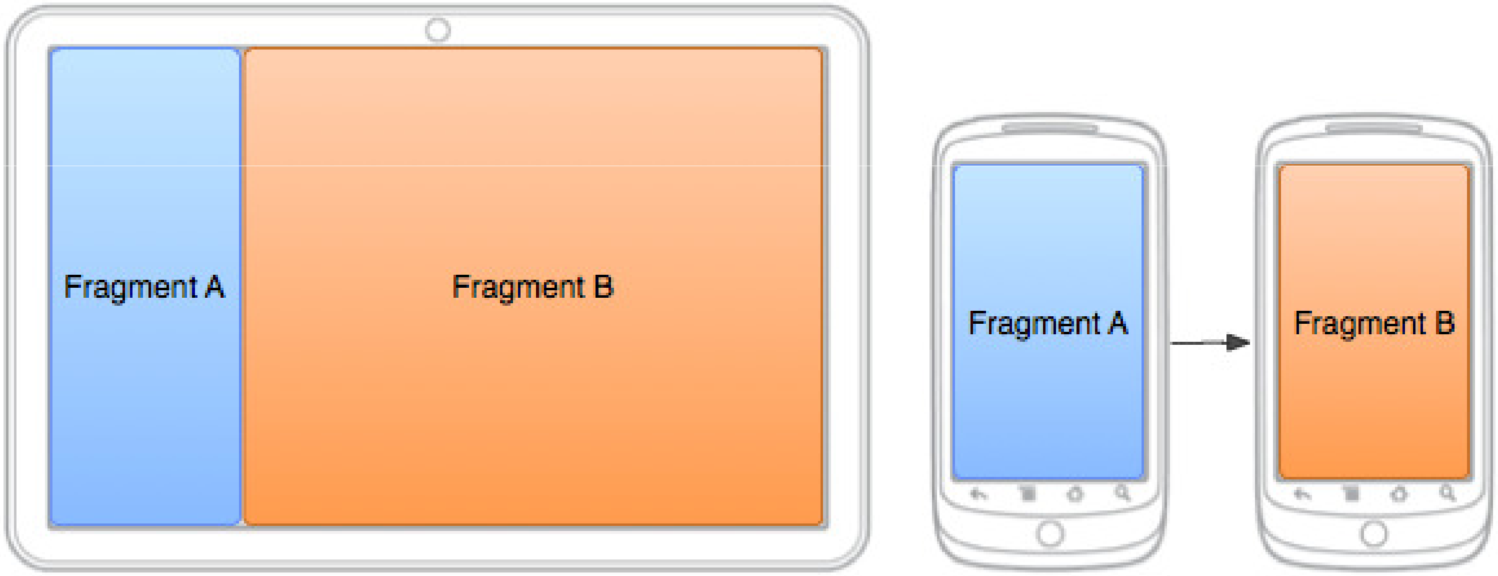
Fragments

- A "sub activity" that you can reuse in different activities (Android 3.0 or later)
- A modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running .
- During runtime, an activity can contain one or more of these mini-activities (fragments)
- Can group multiple fragments (“miniature ” activities) in a single activity
- Can be reused in multiple activities
- Form the atomic unit of user interface and can be **dynamically** added (or removed) to activities in order to create the best user experience possible for the target device (normally large-screen devices)

Fragments


Fragment 1
Activity 1
Fragment 2
Activity 2
Fragment 1
Fragment 2
Activity 1

Fragments



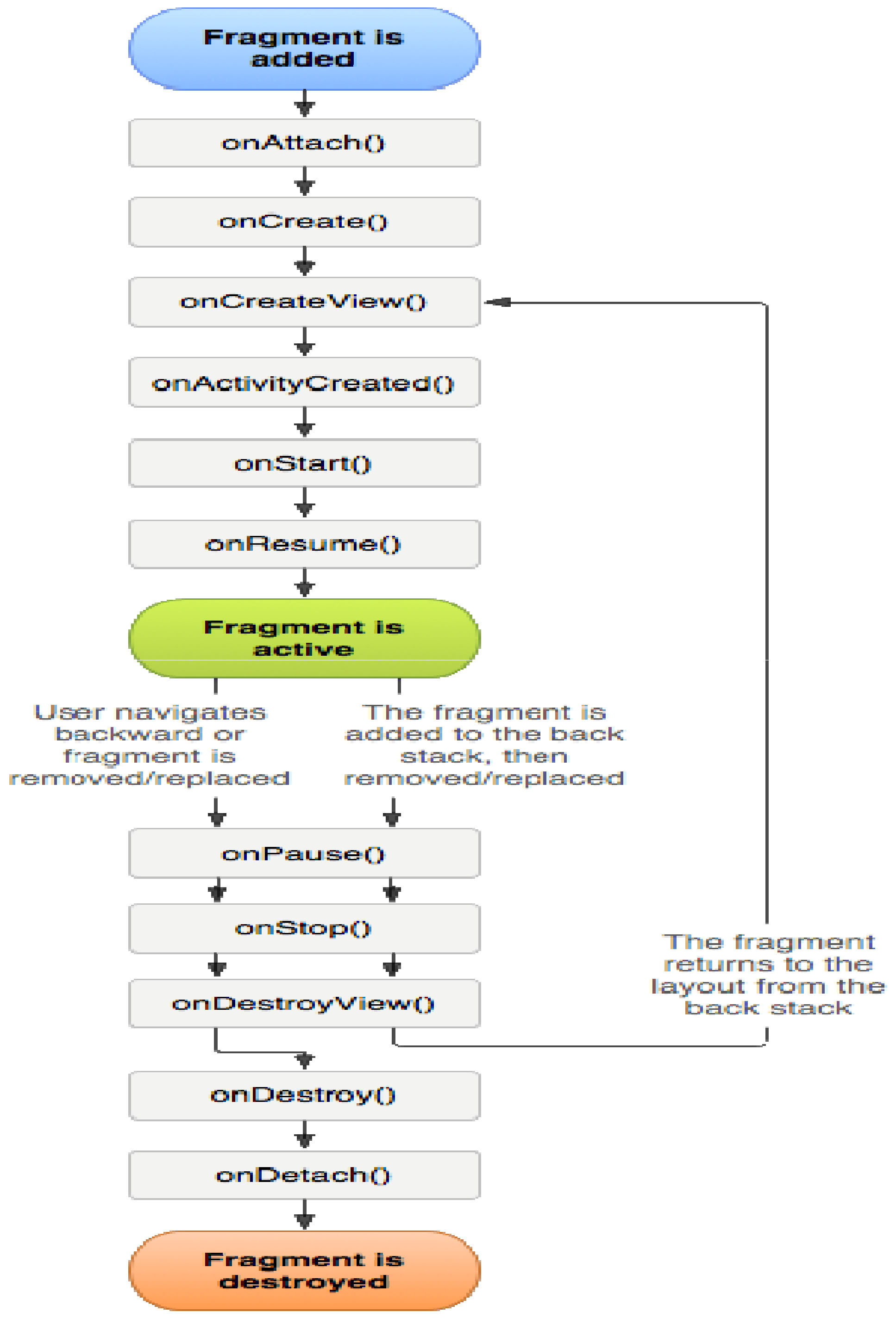
Fragments

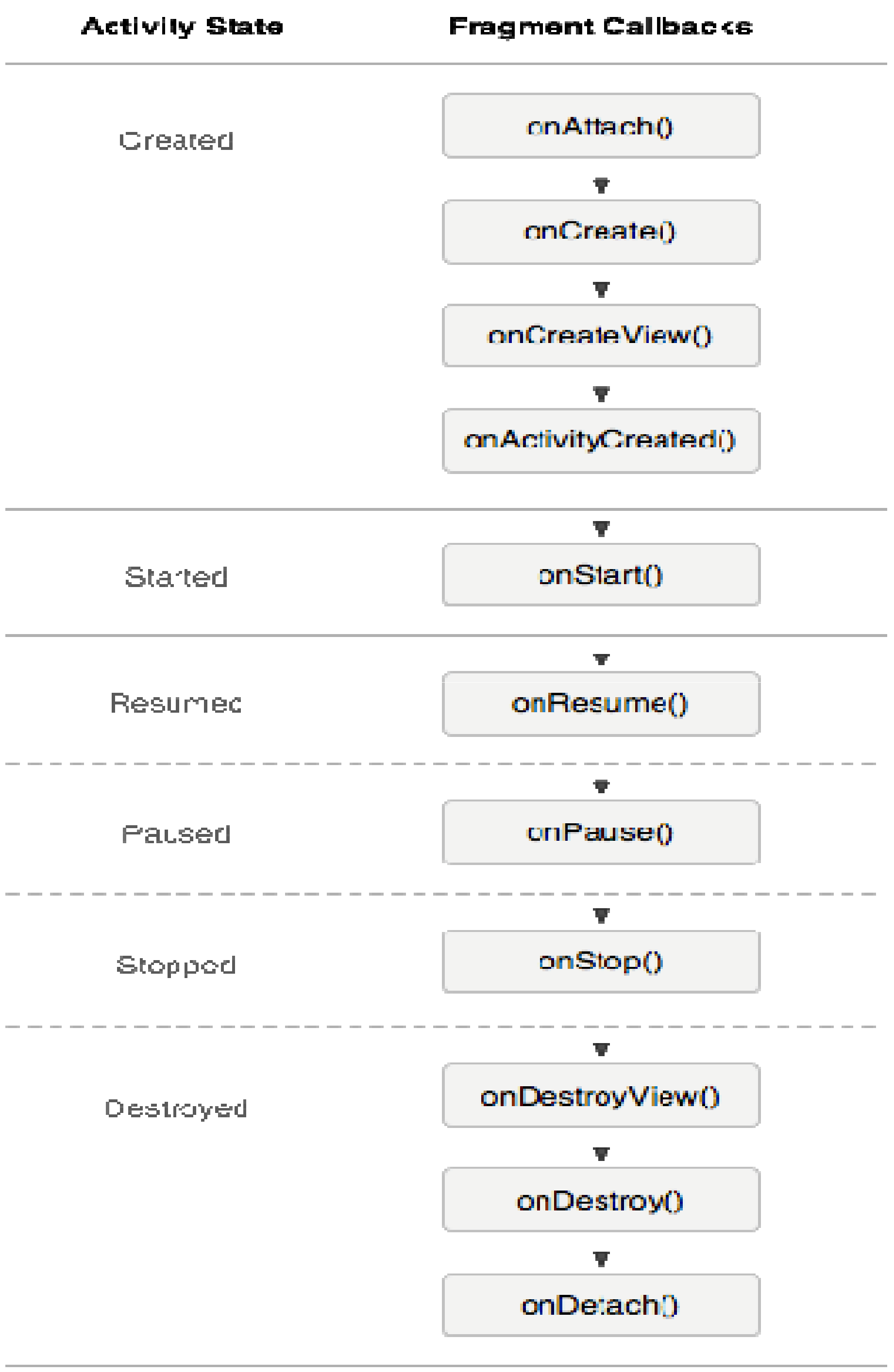
- A Fragment object is somewhere between a View and an Activity.
- Like a View, it can be added to a ViewGroup or be part of a layout
- 2 purposes:
  1. Compartmentalize UI into various configurable parts
  2. Customizable UI: Add them dynamically to activities during runtime
- Its life cycle is dependent on that of the Activity to which it belongs.

Life Cycle of a Fragment

- A fragment can exist in 3 states:
  - **Resumed**: The fragment is visible in the running activity.
  - **Paused**: Another activity is in the foreground and has focus, but the activity in which this fragment lives is still visible (the foreground activity is partially transparent or doesn't cover the entire screen).
  - **Stopped**: The fragment is not visible. Either the host activity has been stopped or the fragment has been removed from the activity but added to the back stack. A stopped fragment is still alive (all state and member information is retained by the system). However, it is no longer visible to the user and will be killed if the activity is killed

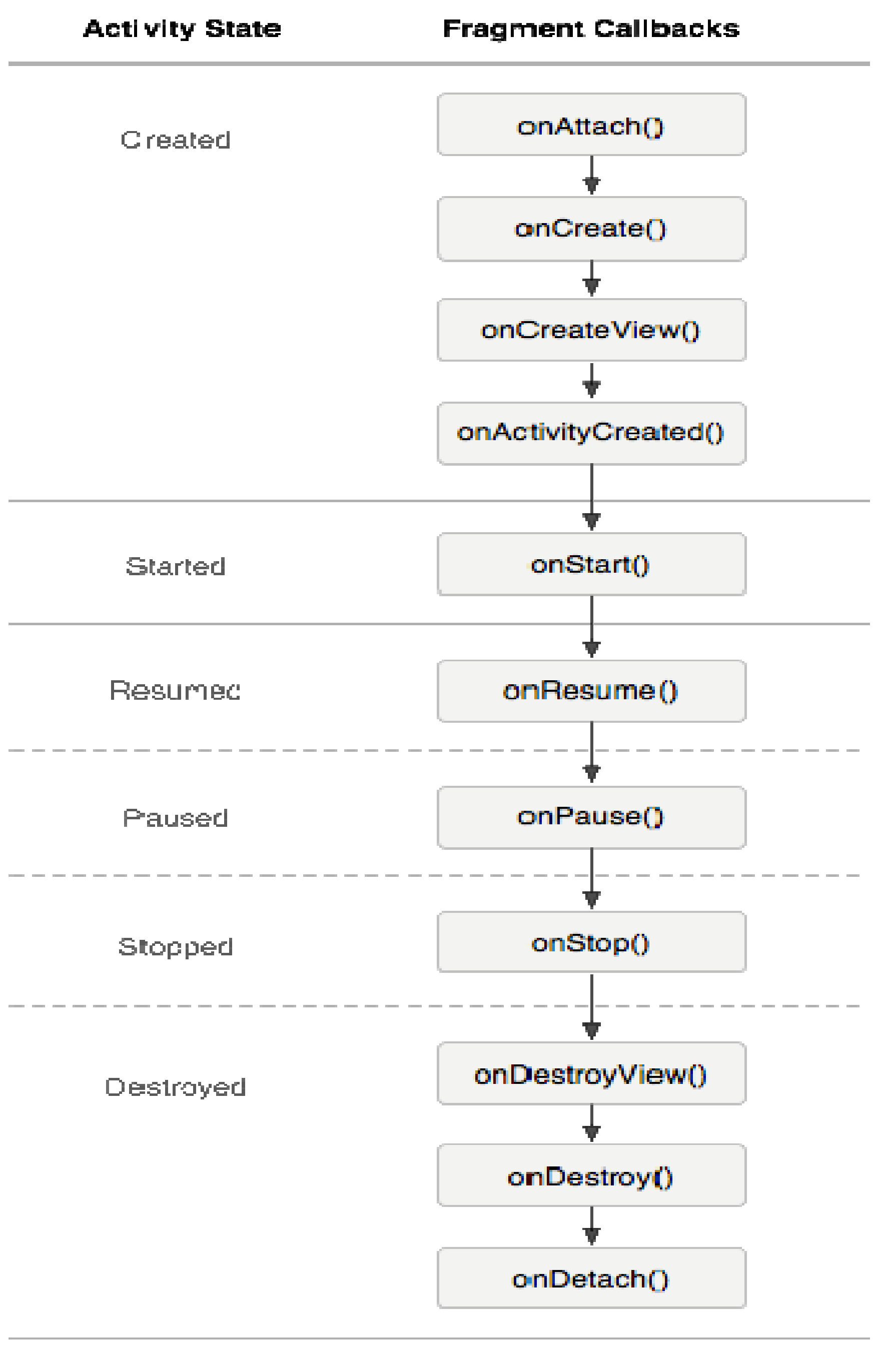
Life Cycle of a Fragment




Activity State
Fragment Callbacks
Created
onAttach()
onCreate()
onCreateView()
onActivityCreated()
Started
onStart()
Resumed
onResume()
Paused
onPause()
Stopped
onStop()
Destroyed
onDestroyView()
onDestroy()
onDetach()

Life Cycle of a Fragment

- Callbacks specific to fragments:
  - **onAttached()** — Called when the fragment has been associated with the activity
  - **onCreateView()** — Called to create the view for the fragment
  - **onActivityCreated()** — Called when the activity's onCreate() method has been returned
  - **onDestroyView()** — Called when the fragment's view is being removed
  - **onDetach()** — Called when the fragment is detached from the activity


Acti vity State
Fragment Callbacks
Created
onAttach()
onCreate()
onCreateView()
onActivityCreated()
Started
onStart()
Resumec
onResume()
Paused
onPause()
Stopped
onStop()
Destroyed
onDestroyView()
onDestroy()
onDetach()

Life Cycle of a Fragment

- When a fragment is being created, it goes through the following states:
  - onAttach()
  - onCreate()
  - onCreateView()
  - onActivityCreated()

Life Cycle of a Fragment

- When the fragment becomes visible, it goes through these states:
  - onStart()
  - onResume()
- When the fragment goes into the background mode, it goes through these states:
  - onPause()
  - onStop()

Life Cycle of a Fragment

- When the fragment is destroyed (when the activity it is currently hosted in is destroyed), it goes through the following states:
  - onPause()
  - onStop()
  - onDestroyView()
  - onDestroy()
  - onDetach()

Life Cycle of a Fragment

- Like activities, we can restore an instance of a fragment using a Bundle object, in the following states:
  - onCreate()
  - onCreateView()
  - onActivityCreated()

Introduction to **Intent**

- **Intents** are **asynchronous messages** which allow the application to request functionality from other components of the Android system, e.g. from *Services* or *Activities*.
- For example the application could implement sharing of data via an Intent and all components which allow sharing of data would be available for the user to select.
- Applications register themselves to an *Intent* via an **IntentFilter**.
- **Intents** allow to combine loosely coupled components to perform certain tasks.

Linking Activities using **Intent**

- How, then does one activity invoke another, and pass information about what the user wants to do? The unit of communication is the **Intent** class.
- Intent is an abstract description of an *operation* to be performed or *function* that one activity requires another activity to perform.
- A facility for performing late runtime binding between the code in different applications ("glue" between activities)
- When an app dispatches an intent (request for some functionality), it's possible that several different activities might be registered (with <intent-filter>) to provide the desired operation.

Linking Activities using **Intent**

- Class: android.content.Intent
- Can be used with **startActivity** to launch an Activity
- In *AndroidManifest.xml*: **<intent-filter>** element defines how your activity can be invoked by another activity
- In *MainActivity.java*: **startActivity()** method invokes another activity but does not return a result to the current activity.
- More details in Topic 4

References

- Textbook
- Android Developers: http://developer.android.com/guide/components/index.html
- Android - A beginner's guide: http://www.codeproject.com/Articles/102065/Android-A-beginner-s-guide
- Android Development Tutorial: http://www.vogella.com/articles/Android/article.html#overview
- http://androiddevelopement.blogspot.com/2011/12/android-40-development-tutorial.html