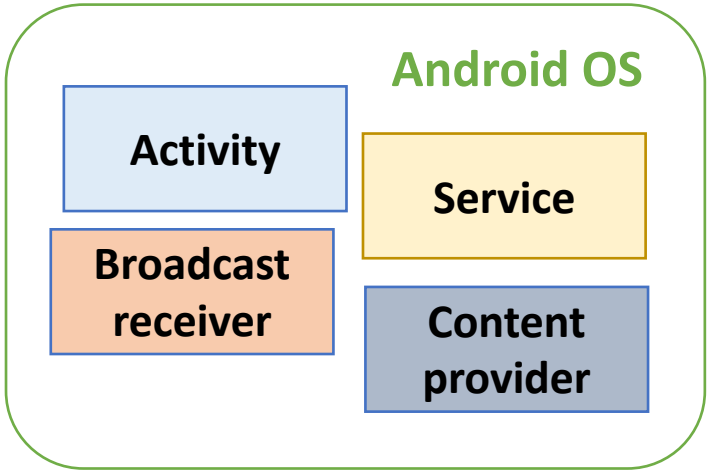
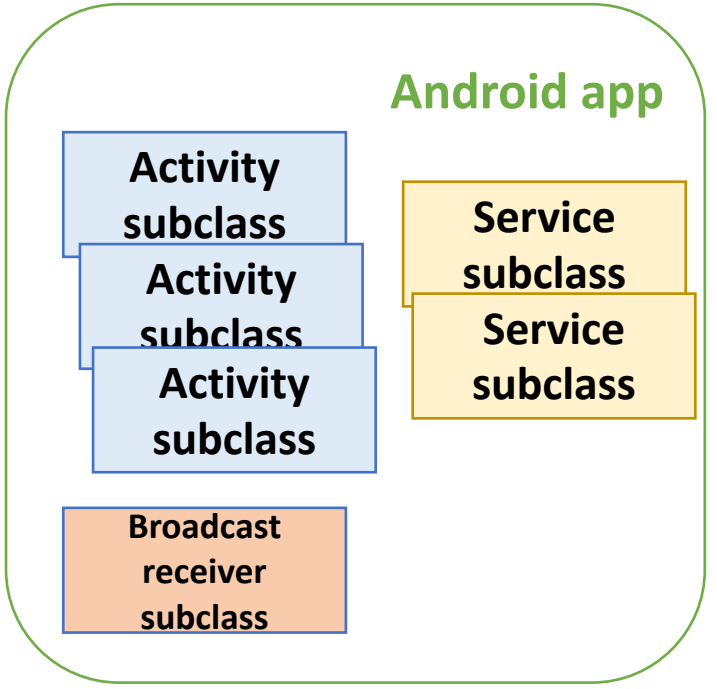
Activity lifecycle and other Android components

Android applications

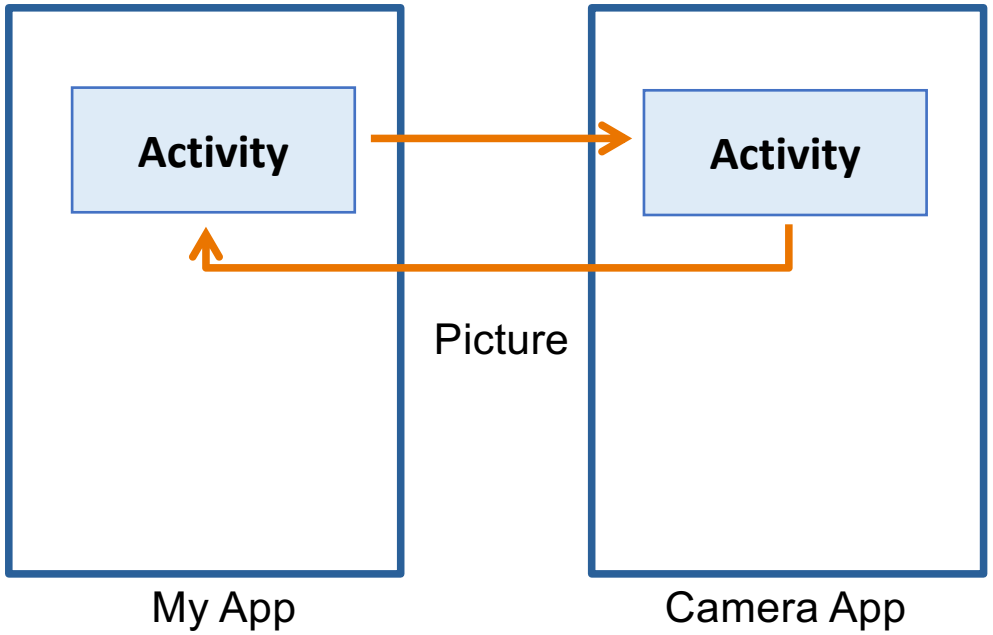
- In desktop applications execution starts from *main()*
- Android app
  - There is no *main()*
  - App is a collection of components derived from OS classes
  - Android calls methods of components according to an event-driven approach
    - User touches app icon and/or widget of UI
    - Something of interest has happened (e.g. an SMS has arrived)
- Four types of app components
  - Activities
  - Services
  - Broadcast receivers
  - Content providers





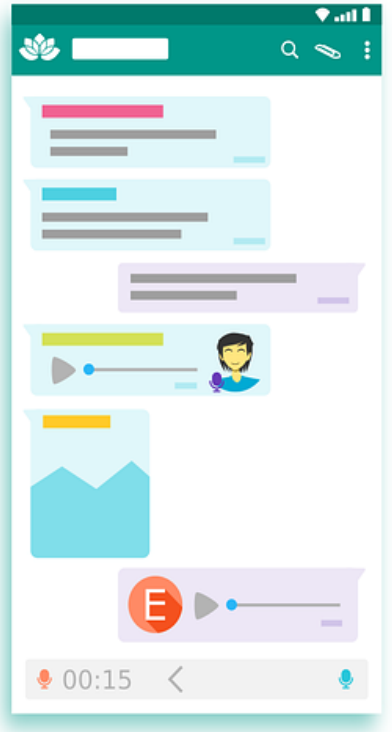
Applications

- An app can start another app's component
  - if you want the user to take a picture, there is another app that does that and your app can use it, instead of developing an activity to capture a photo yourself
- When the system starts a component, it starts the process for that application (if it is not already running) and instantiates the classes needed for the component



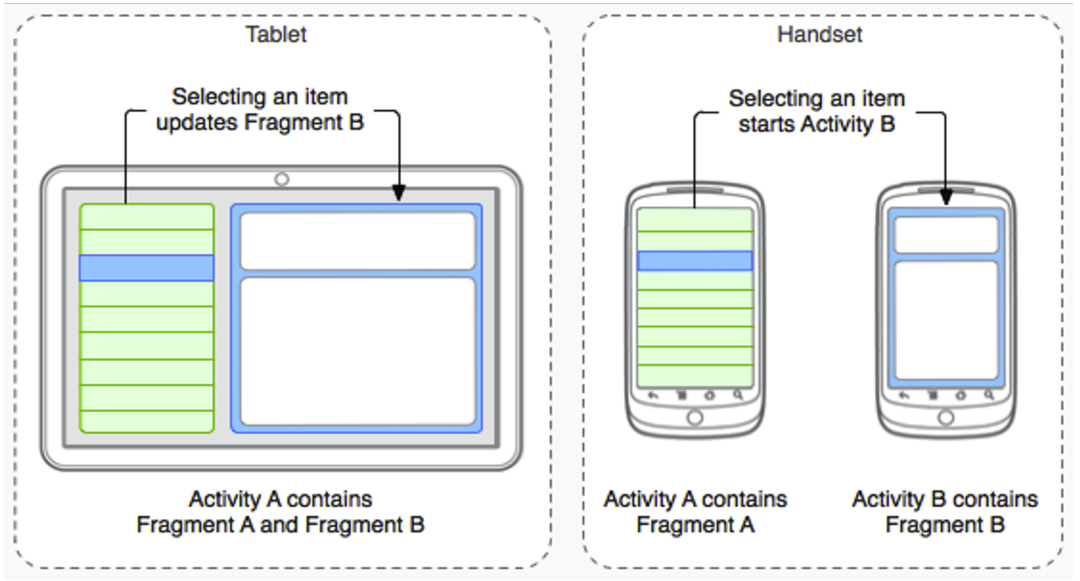
Activities (recap)

- Activity: main building block of Android UI
  - Analogous to a window in a desktop application
- Apps
  - have at least 1 activity that deals with UI
  - entry point of app similar to *main()* in C
  - typically have multiple activities
- Example: whatsapp
  - Activity 1: to select a contact from list, start activity 2
  - Activity 2: to show conversation and send new messages



Fragments

- Fragments
  - UI building blocks, can be arranged in Activities in different ways
  - Enables app to look different on different devices (e.g. phone vs tablet)
- An activity can contain multiple fragments that are organized differently on different devices (e.g. for phone vs tablet)



Services

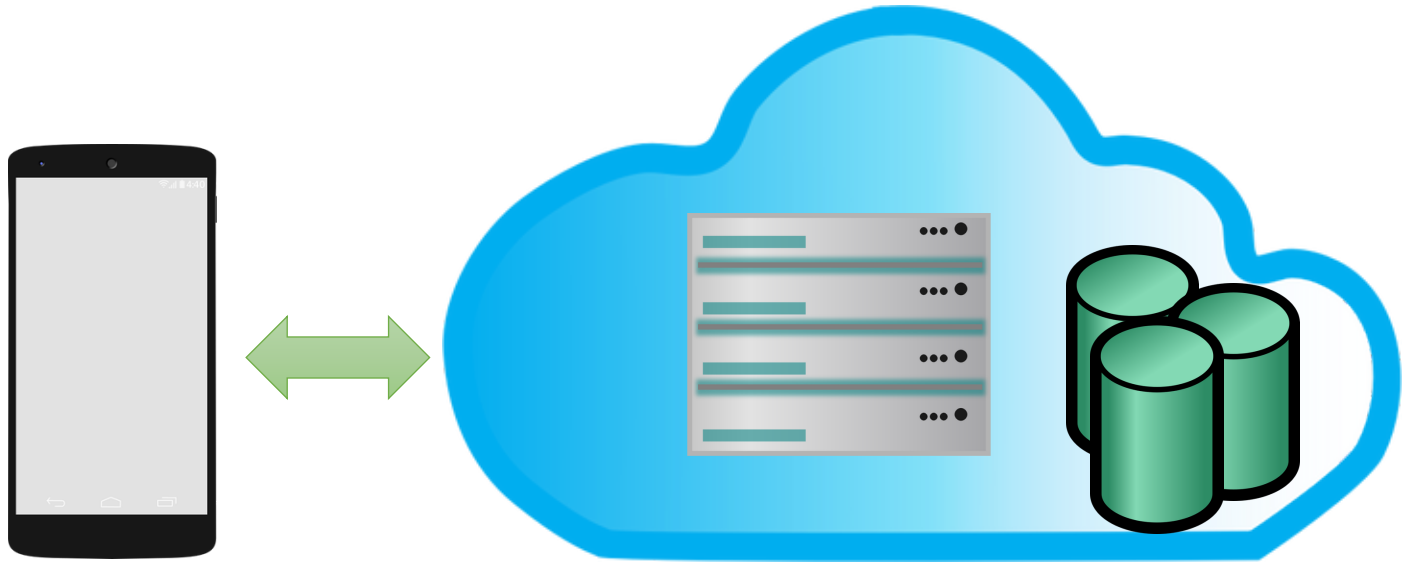
- Activities are short-lived, can be shut down anytime (e.g when user presses back button)
- Services keep running in background
  - Example uses of services:
    - Periodically check/update device's GPS location
    - Check for new email messages
- Independent of any activity
- Typically an activity will control a service
  - start it
  - pause it
  - get data from it
- Services in an App are sub-classes of Android's *Service* class

Managers

- Provide high-level functionalities or access to HW:
  - **LocationManager**: location-based services
  - **ClipboardManager**: access to device's clipboard, cut-and-paste content
  - **DownloadManager**: manages HTTP downloads in background
  - **FragmentManager**: manages the fragments of an activity
  - **AudioManager**: provides access to audio and ringer controls

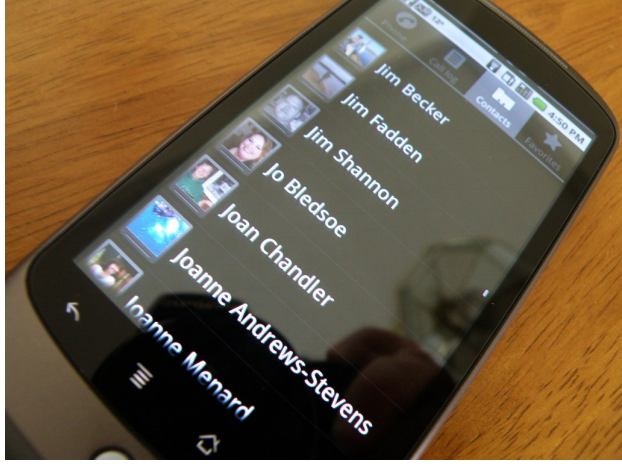
Google services

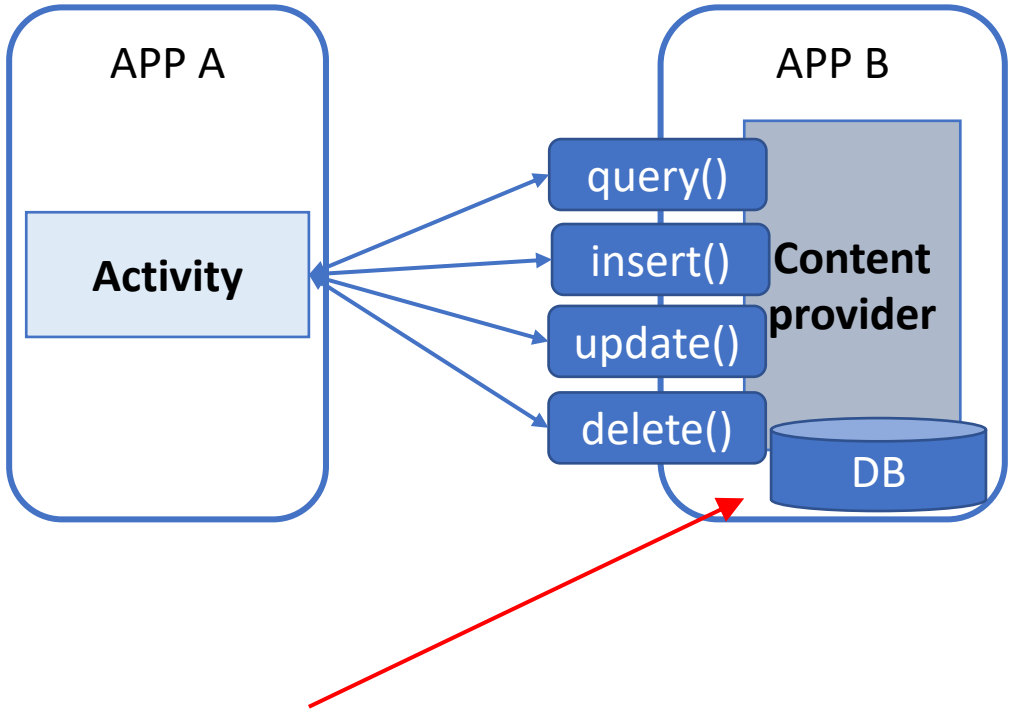
- Not components, services in the «classical» way
- Google Services (in Google cloud)
  - Maps
  - Location-based services
  - Game Services
  - Authorization APIs
  - Play Services
  - In-app Billing
  - Google Cloud Messaging
  - Google Analytics
  - Google AdMob ads
- Not open source, not part of AOSP



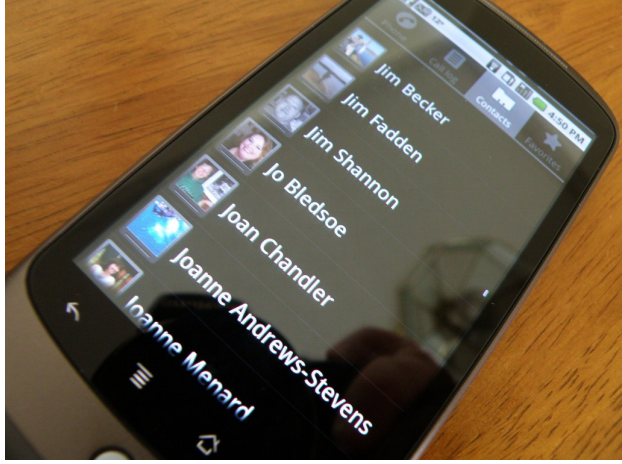
Content providers

- Android apps can share data as content providers
  - User's contact list
  - Media files
- Content Provider:
  - Abstracts shareable data, makes it accessible through methods
  - Applications can access that shared data by calling methods for the relevant content provider
- Example: query, insert, update, delete shared data

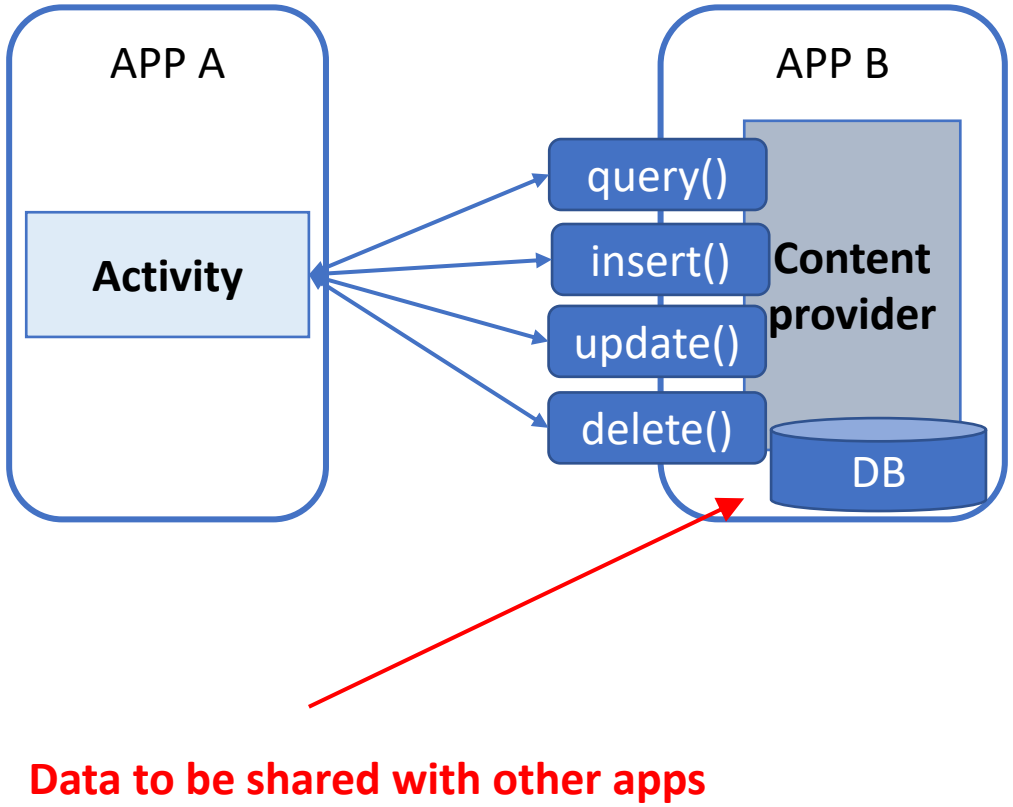




Content providers



- Data stored in Android Contacts app can be accessed by other apps
- Example: We can write an app that:
  - Retrieve's contacts list from contacts content provider
  - Adds contacts to social networking (e.g. Facebook)
- Apps can also add new data through content provider
  - E.g. Add contact
- Our app can also share its data
- Content provider in an App are sub-class of Android's *ContentProvider* class

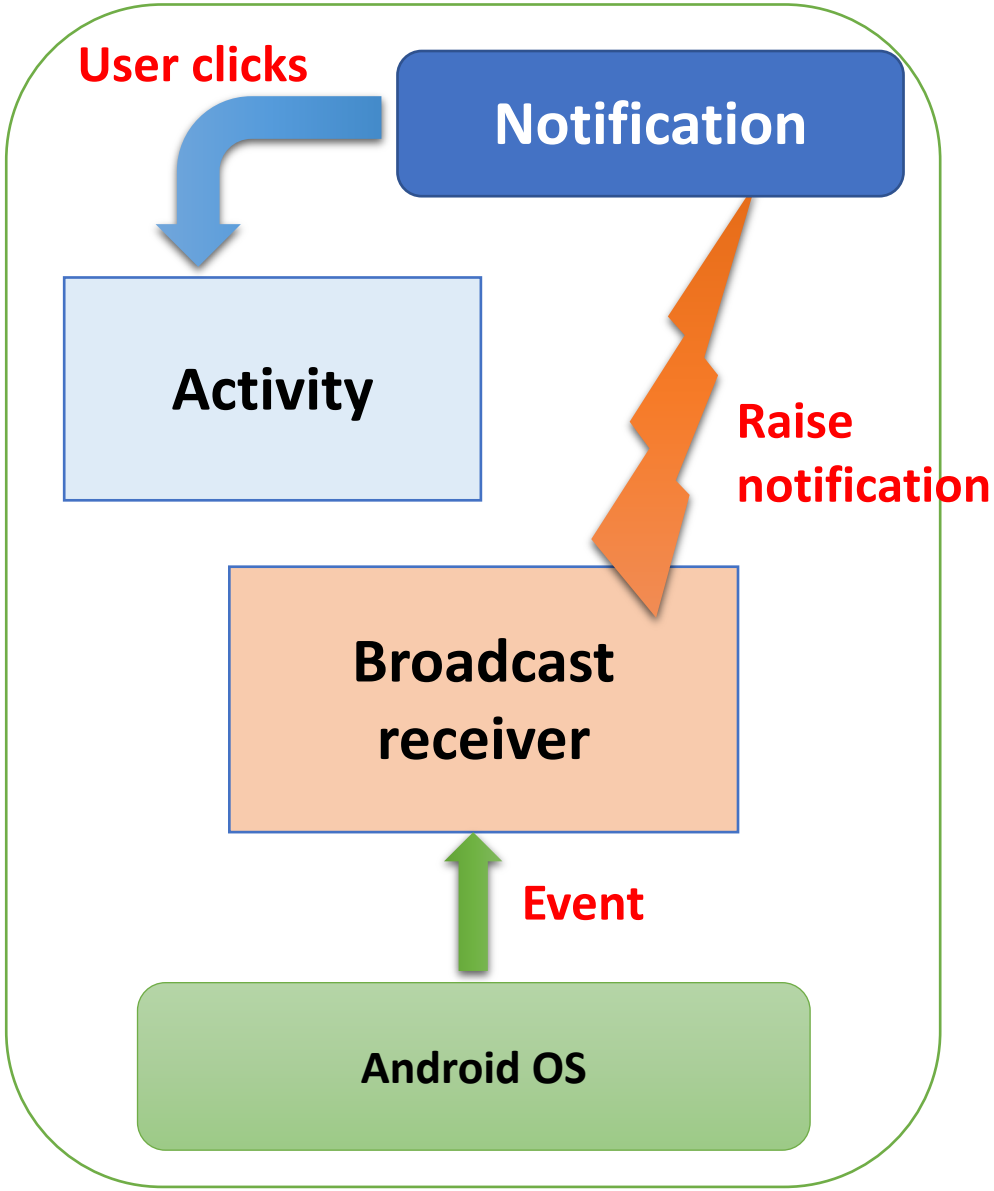


Broadcast receivers

- Android OS (system), or applications, periodically broadcasts events
- Example broadcasts:
  - Battery getting low
  - Download completed
  - New email arrived
- Any app can create a broadcast receiver to listen for broadcasts
- Our app can also initiate broadcasts
- Broadcast receivers typically
  - Don’t interact with the UI
  - Creates a status bar notification to alert the user when broadcast event occurs
- Broadcast Receiver in an App are sub-class of Android’s *BroadcastReceiver* class

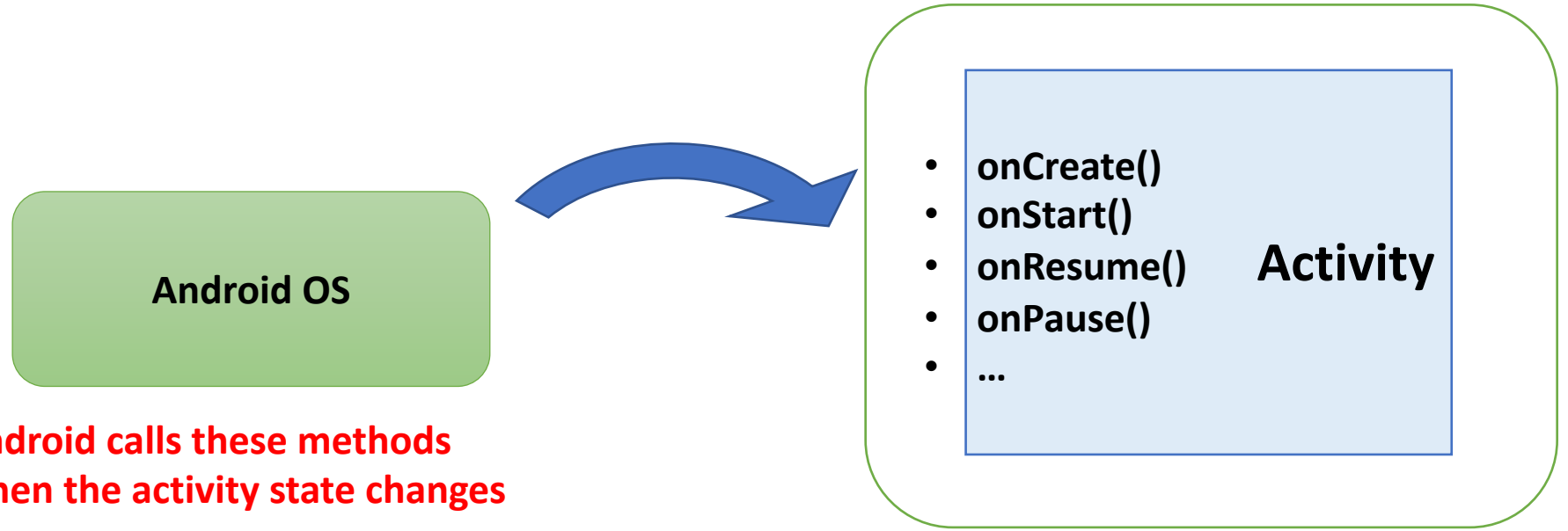
Broadcast receivers

- Interaction with the user
  - OS broadcasts event
  - Receiver catches event and raises notification
  - Notification remains in status bar until the user decides he wants to interact with app
  - Notification launches activity
- Same model also for services



Activity lifecycle

- Android Activity callbacks invoked corresponding to app state changes
- Examples:
  - When activity is created, its *onCreate()* method invoked
  - When activity is paused, its *onPause()* method invoked

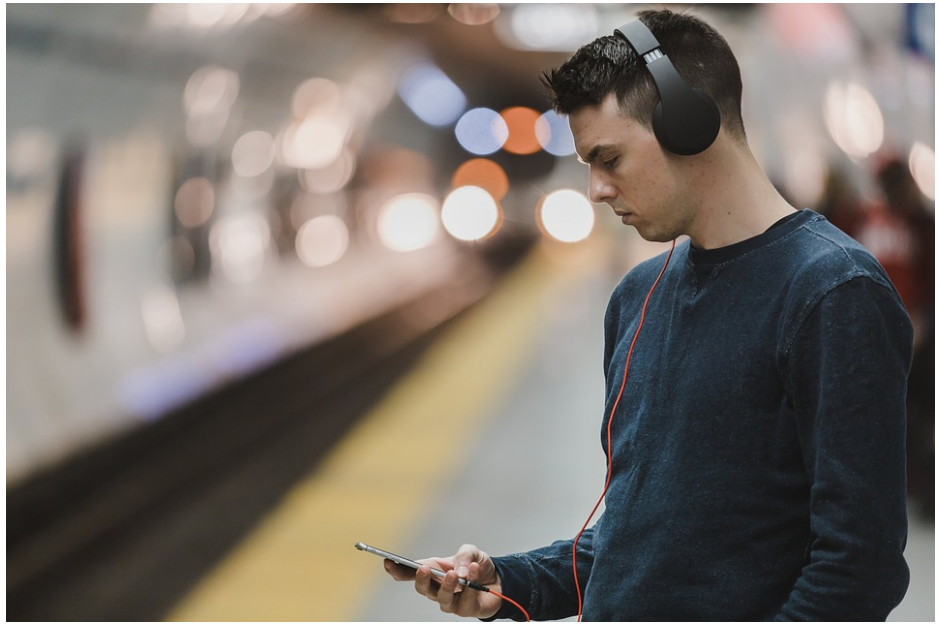


Activity methods

- `onCreate()`
- `onStart()`
- `onResume()`
- `onPause()`
- `onStop()`
- `onRestart()`
- `onDestroy()`

- Android calls **all** these methods
- Programmer re-defines a subset, only the ones relevant for the specific activity
- *onCreate()* is alway defined: the app inflates layout to provide UI

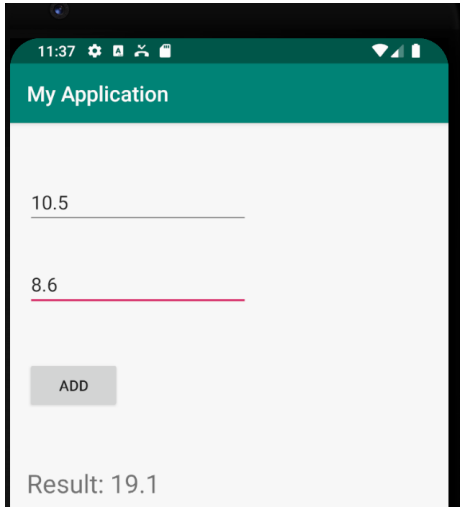
Activity lifecycle



- Many things could happen while app is running
  - Incoming call or text message
  - User switches to another app
  - Train arrives and user puts device in his pocket
- Well designed app should not:
  - Crash if interrupted, or user switches to other app
  - Lose the user's state/progress (e.g state of game app) if they leave your app and return later
  - Crash or lose the user's progress when the screen rotates between landscape and portrait orientation.
    - E.g. videos should continue at correct point after rotation
- To notify these situations, callback methods are invoked

https://developer.android.com/guide/components/activities/activity-lifecycle.html

Activity lifecycle: onCreate()

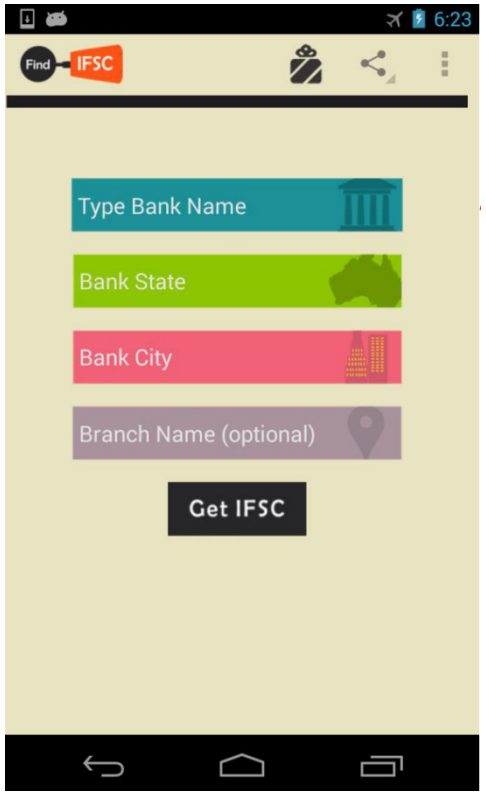


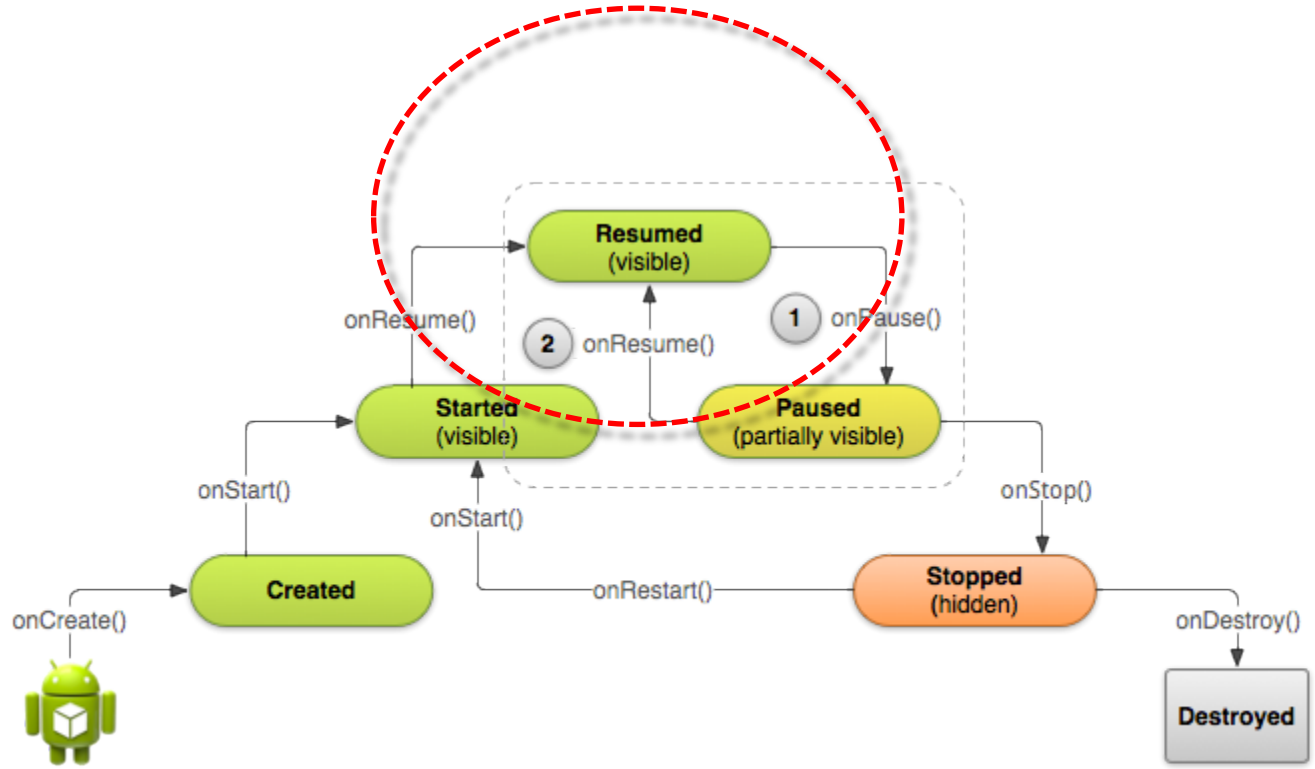
- Initializes activity once created
- Operations typically performed in *onCreate()* method:
  - Inflate (create) widgets and place them on screen
    - E.g. using layout files with *setContentView()*
  - Getting references to inflated widgets (using *findViewbyId()*)
  - Setting listeners to handle user interaction (e.g. clicking buttons)

```
public class MainActivity extends AppCompatActivity
                          implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button b = (Button) findViewById(R.id.button);
        b.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        EditText et1 = (EditText) findViewById(R.id.editText1);
        EditText et2 = (EditText) findViewById(R.id.editText2);
        TextView tv = (TextView) findViewById(R.id.textView);
        float f1 = Float.parseFloat(et1.getText().toString());
        float f2 = Float.parseFloat(et2.getText().toString());
        float r = f1 + f2;
        tv.setText("Result: " + r);
    }
}
```

Activity lifecycle

- A running app is one that user is currently using or interacting with
  - Visible, in foreground

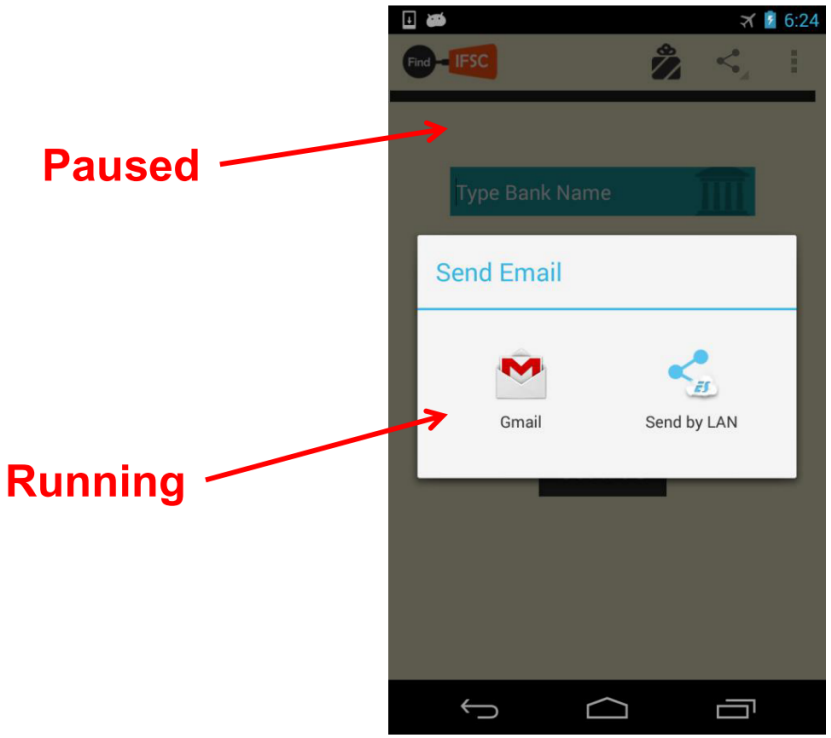


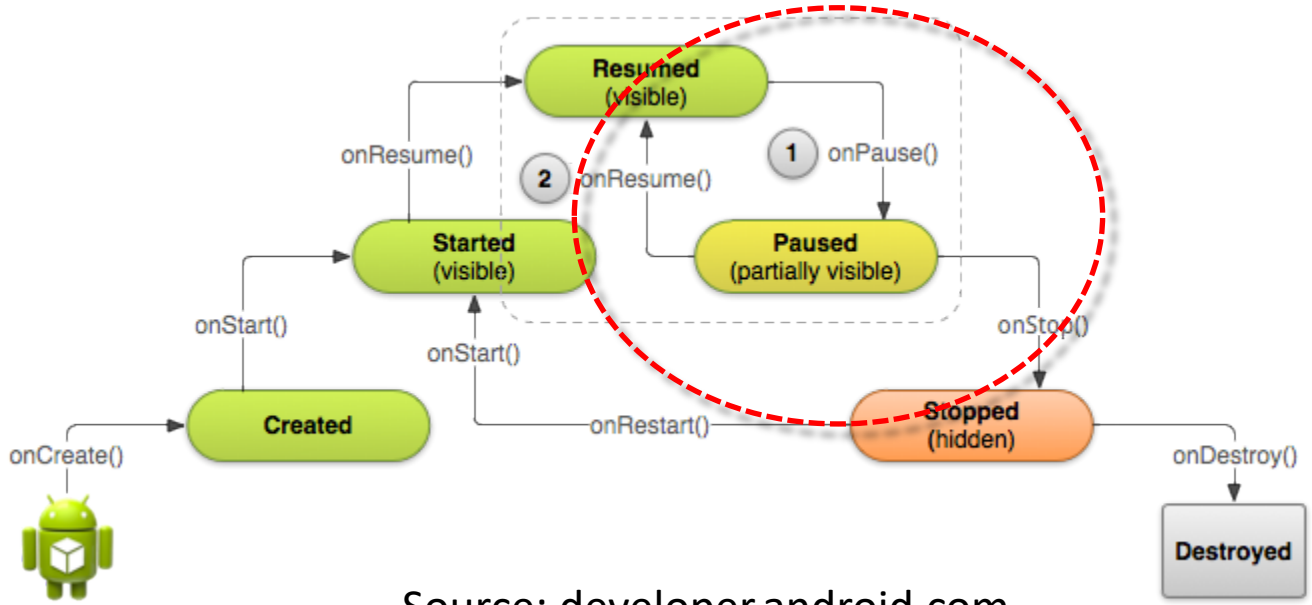


Source: developer.android.com

Paused app

- An app is paused if it is visible but no longer in foreground (e.g. blocked by a pop-up dialog box)
- App's *onPause()* method is called during transition from running to paused state

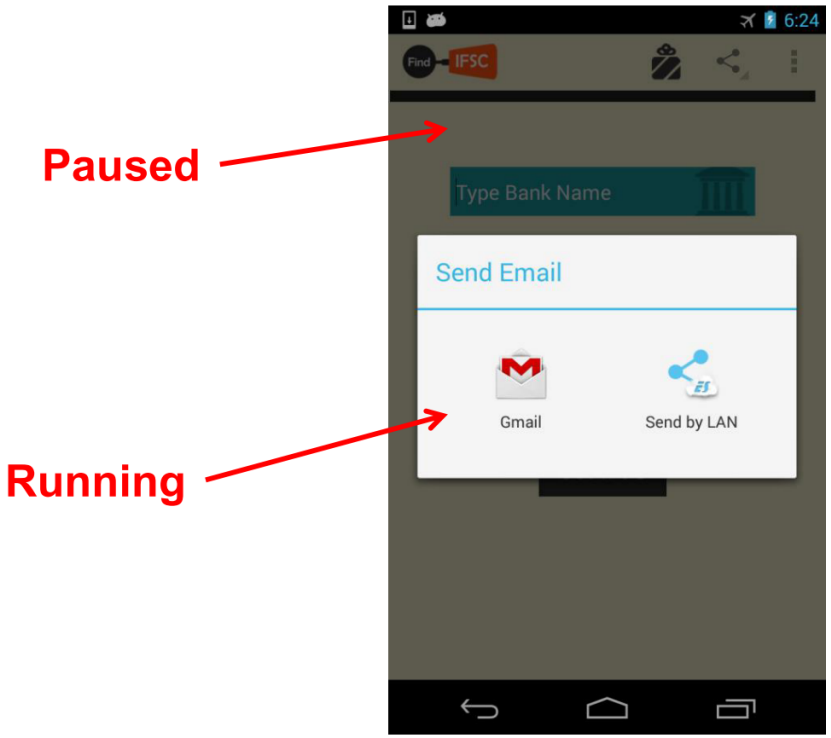


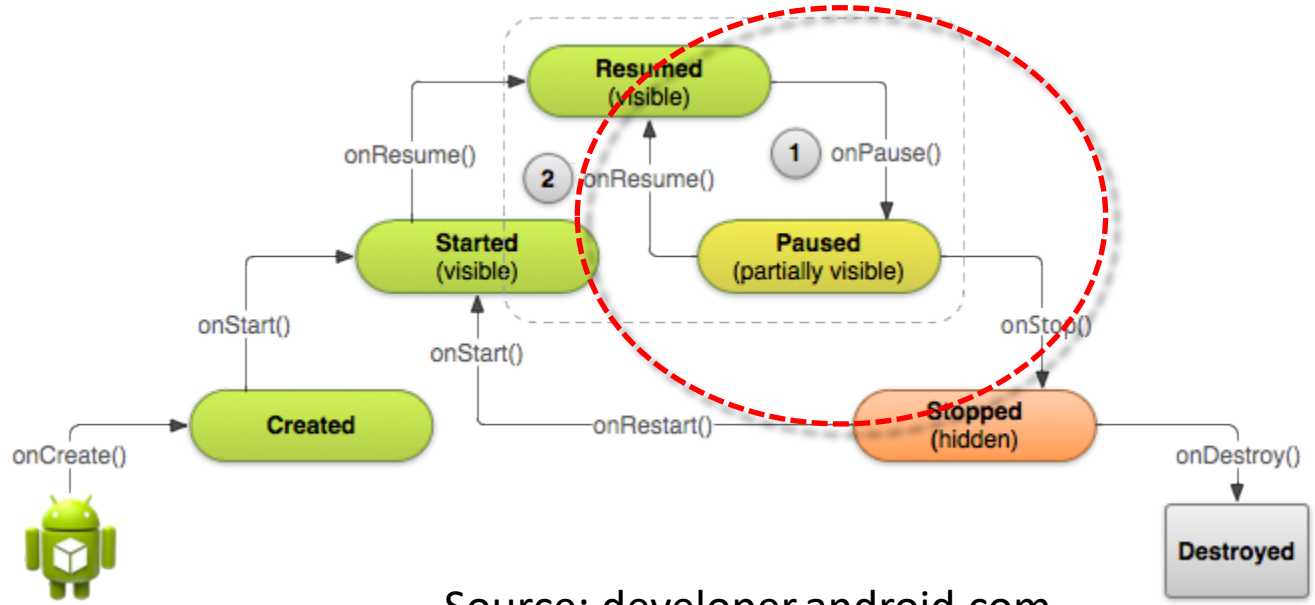


Source: developer.android.com

Paused app

- Typical actions taken in *onPause()* method
  - Stop CPU intensive tasks, stop audio, video, animations
  - Stop listening for GPS, broadcast information, release handles to sensors (e.g GPS, camera)

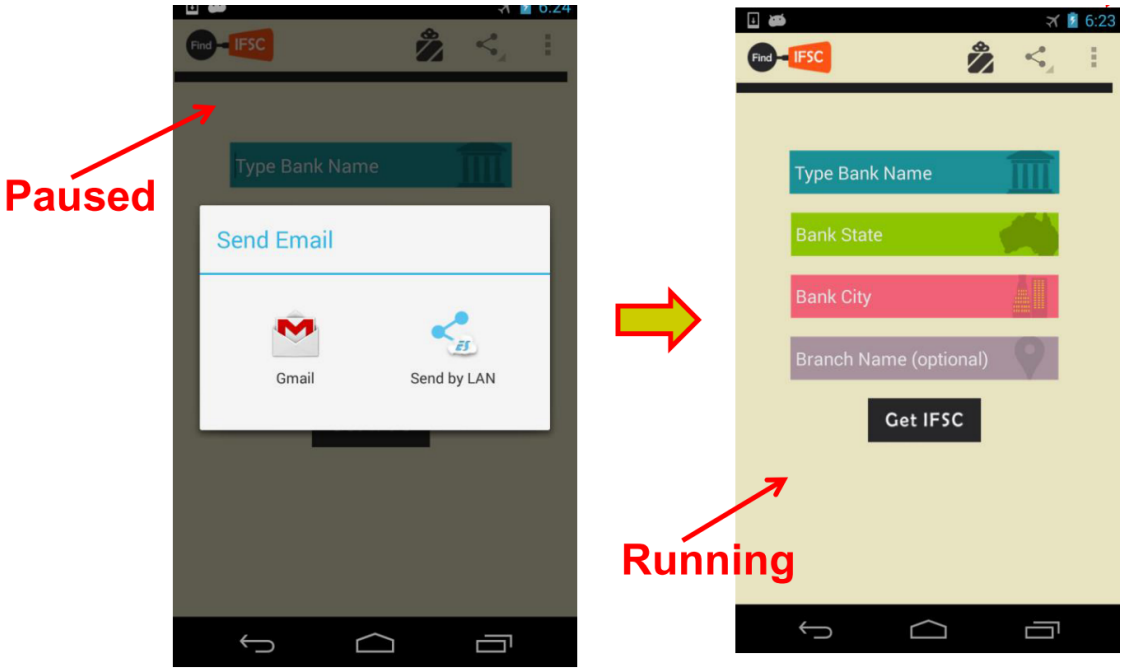


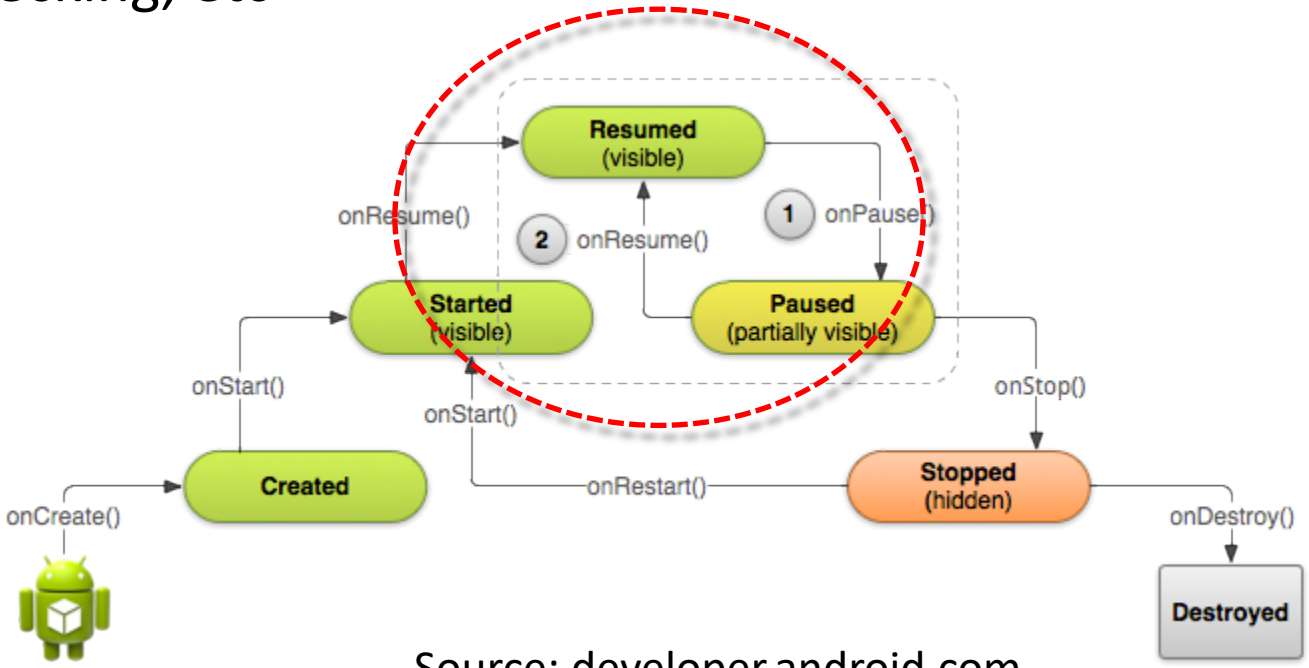


Source: developer.android.com

onResume(): Resuming Paused App

- A paused app resumes running if it becomes fully visible and in foreground
  - E.g. pop-up dialog box blocking it goes away
- App's *onResume()* method is called during transition from paused to running state
  - Restart videos, animations, GPS checking, etc

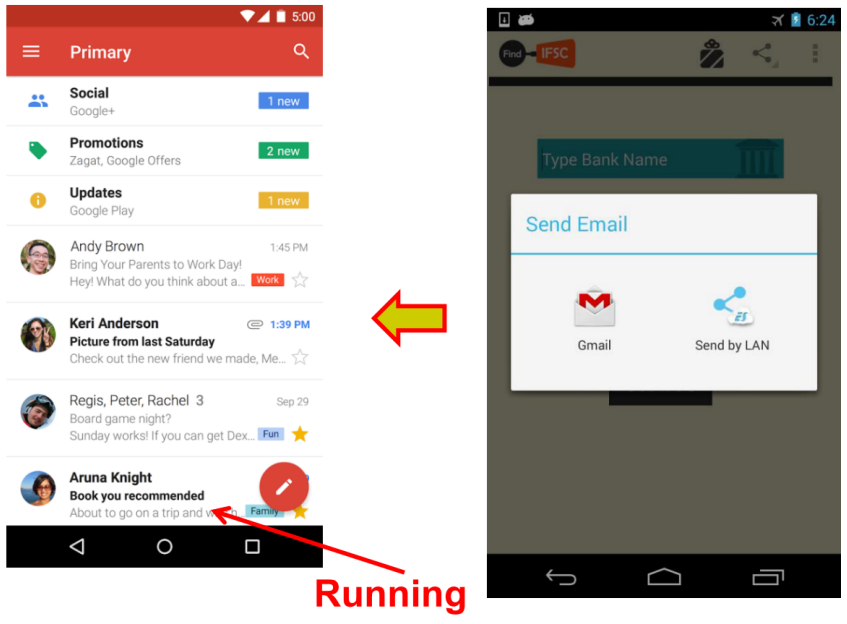


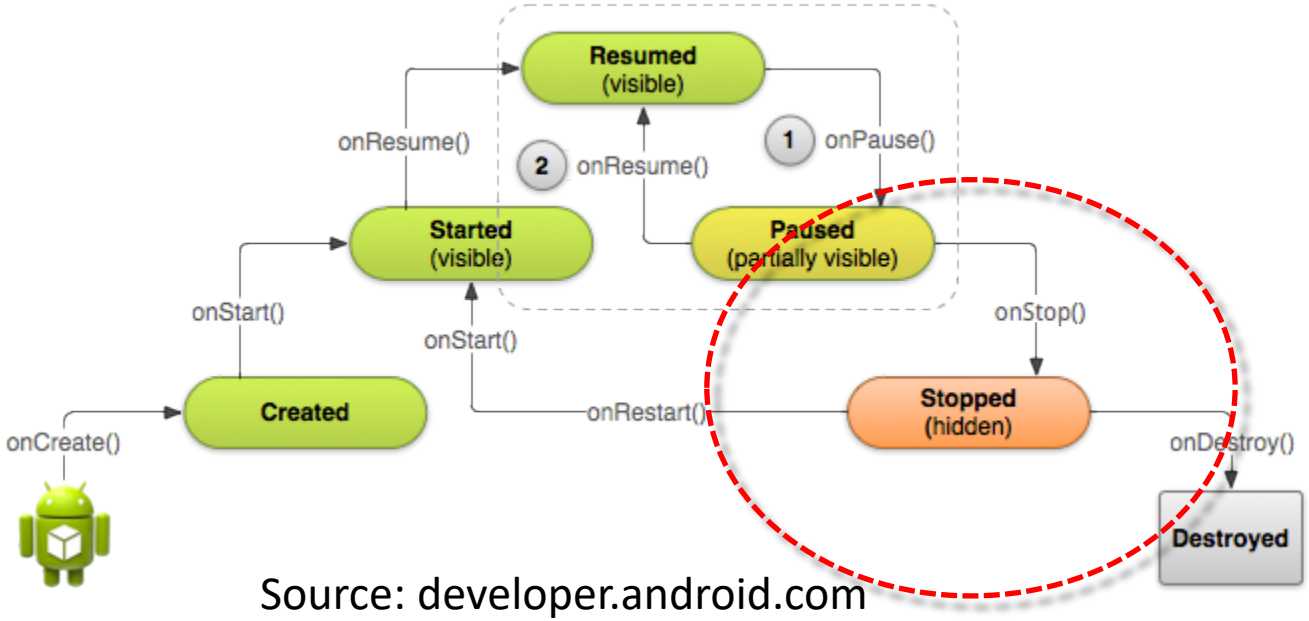


Source: developer.android.com

Stopped app

- An app is stopped if it's no longer visible + no longer in foreground
- E.g. user starts using another app
- App's *onStop()* method is called during transition from paused to stopped state

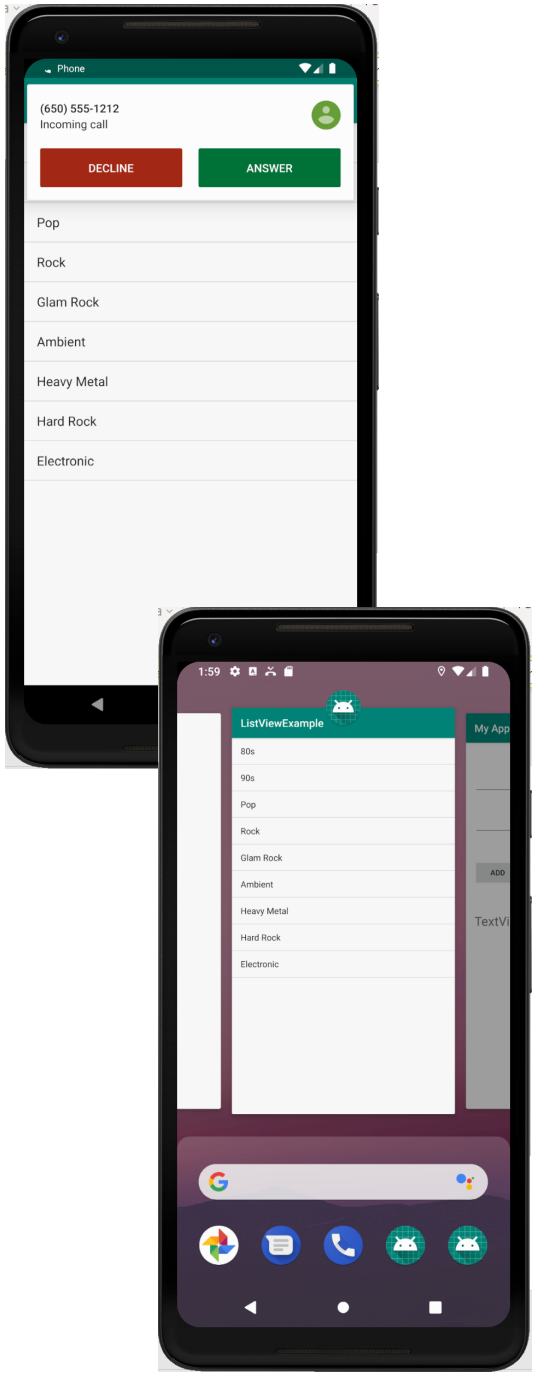




Source: developer.android.com

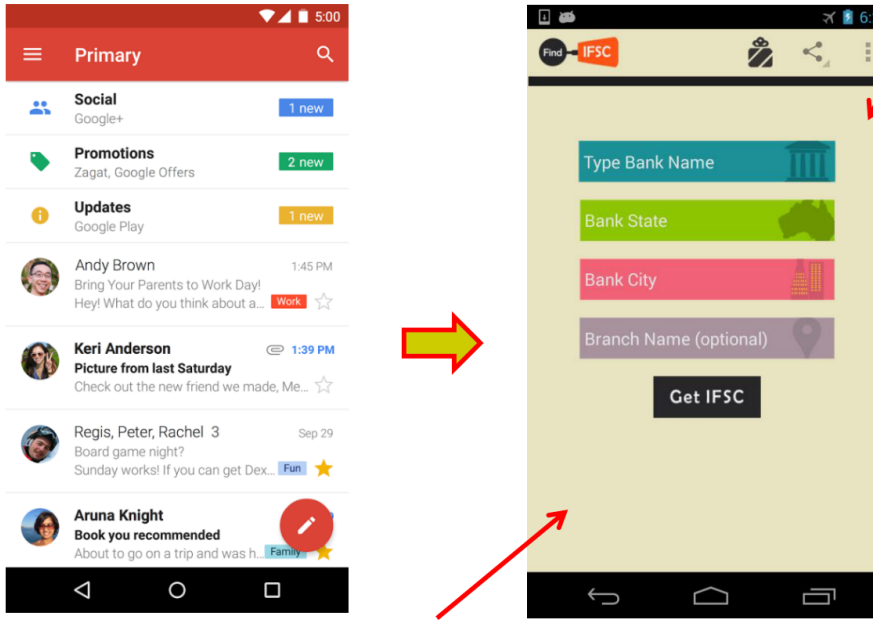
onStop() Method

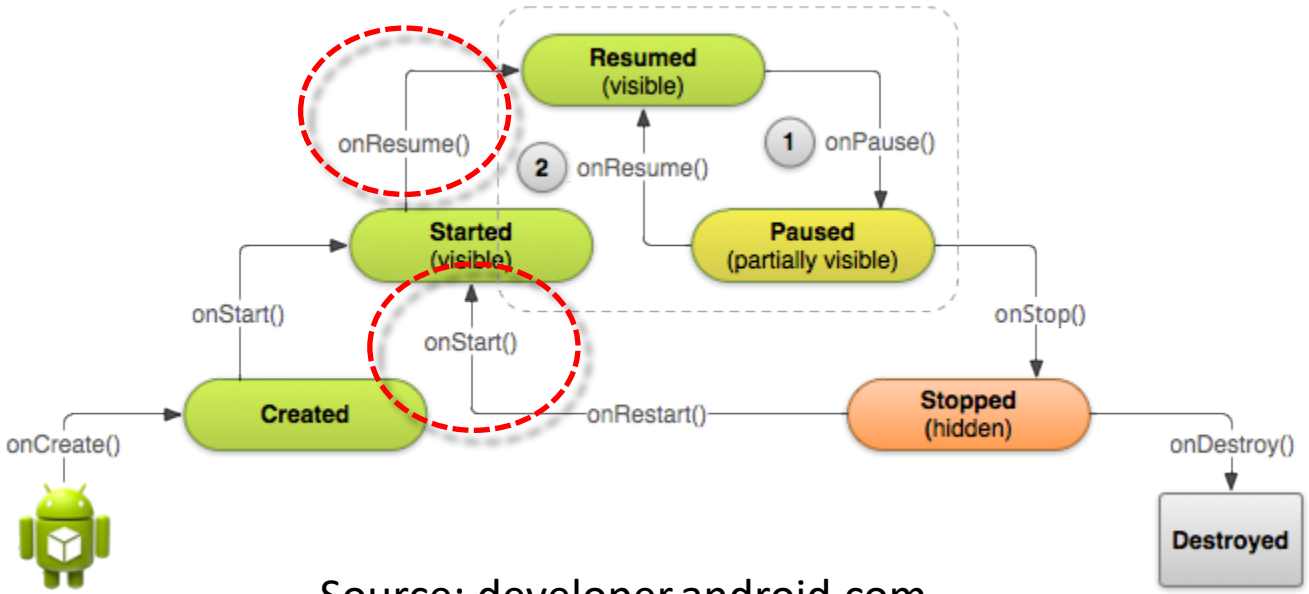
- An activity is stopped when:
  - User receives phone call
  - User starts another app
- Activity instance and variables of stopped app are retained but no code is being executed by the activity
- If activity is stopped, in *onStop()* method, well behaved apps should
  - save progress to enable seamless restart later
  - release all resources, save info (persistence)



Resuming a stopped app

- A stopped app can go back into running state if becomes visible and in foreground
- App's *onStart()* and *onResume()* methods called to transition from stopped to running state

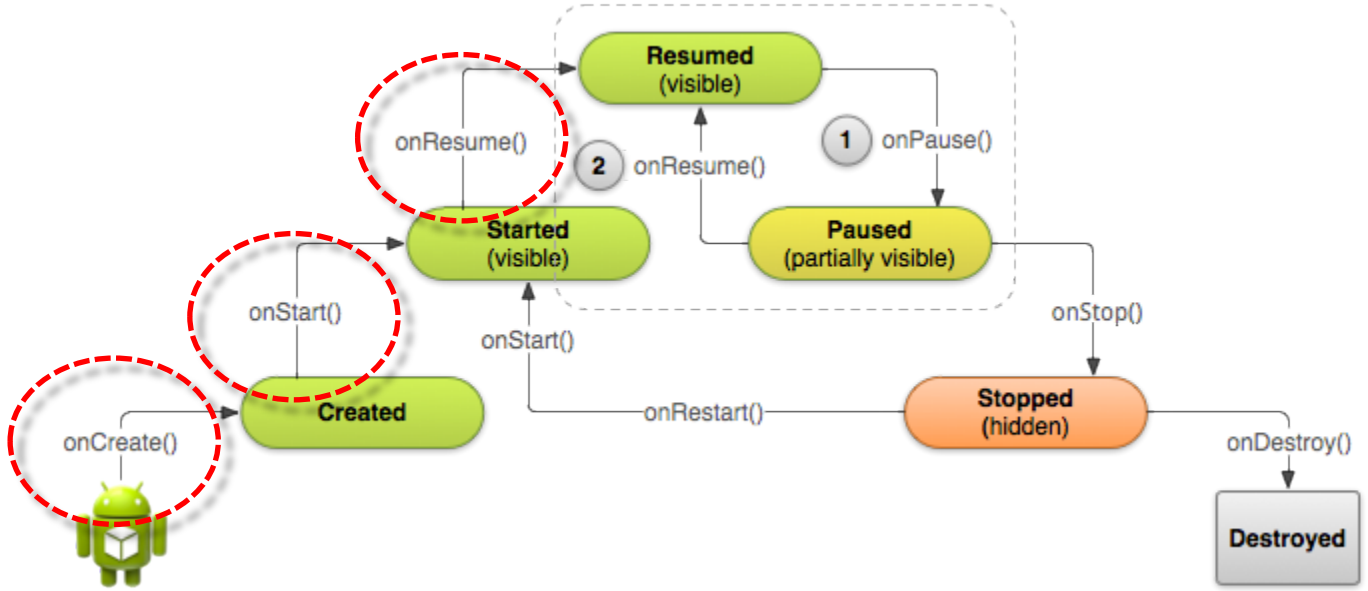




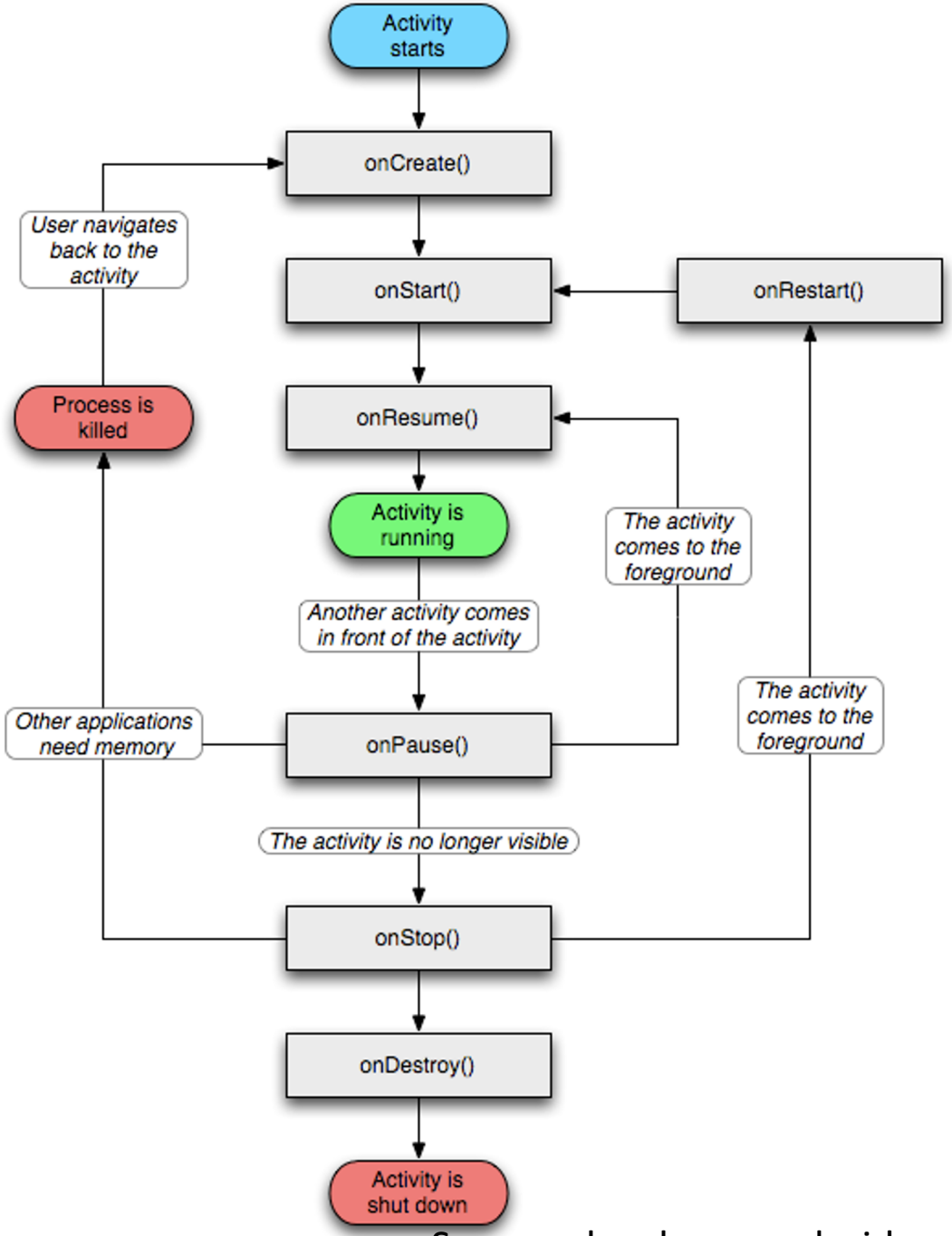
Source: developer.android.com

Starting new app

- When a new app is launched
- App's *onCreate()*, *onStart()* and *onResume()* methods are called
- Afterwards new app is running



The complete lifecycle



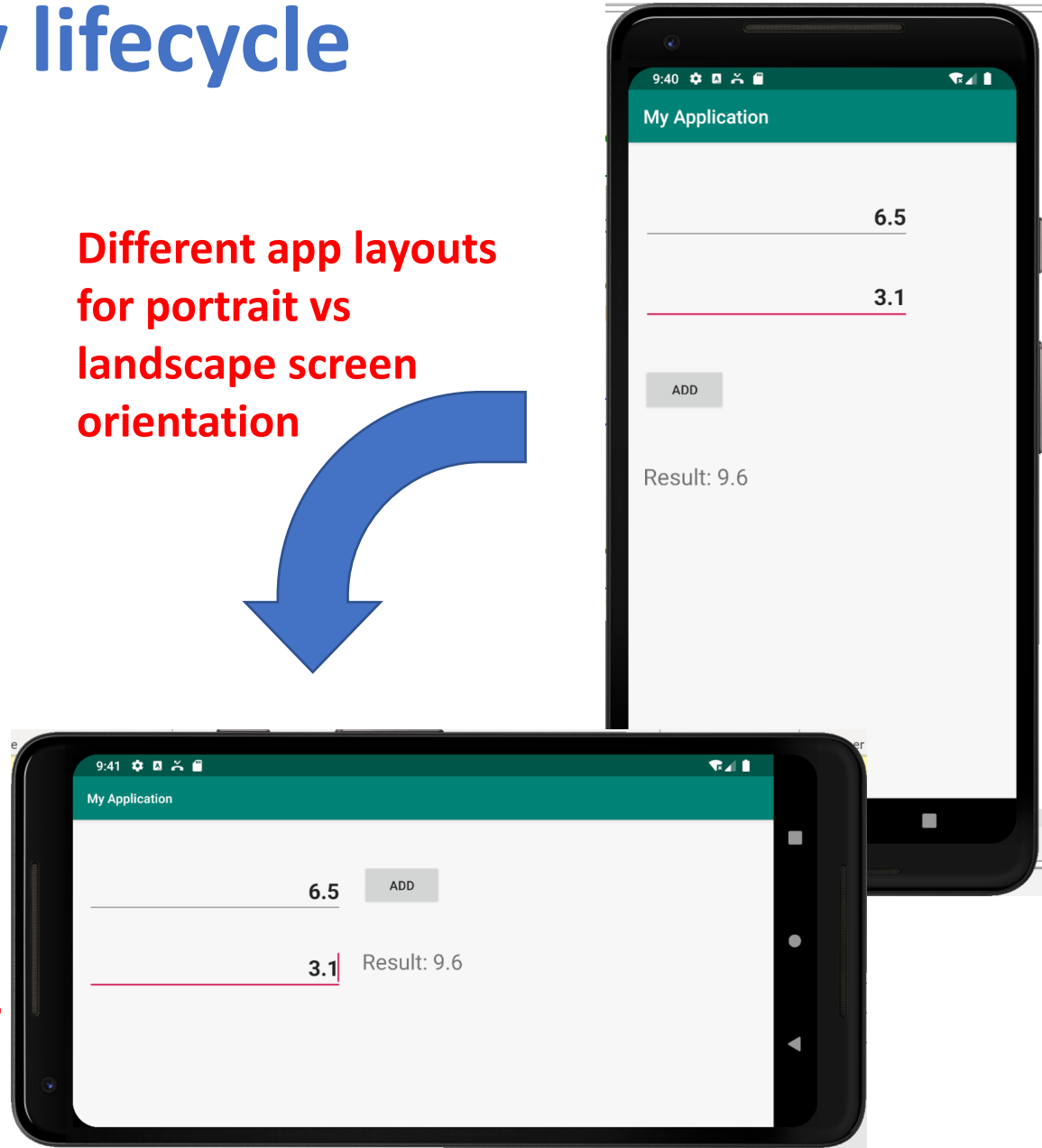
Source: developer.android.com

Device rotation and activity lifecycle

- Rotating device (e.g. portrait to landscape) kills current activity and creates new instance of same activity in landscape mode
- Rotation changes device configuration
- Device configuration: screen orientation/density/size, dock mode, language, etc.
- Apps can specify different resources (e.g. XML layout files, images) to to be used for the different device configurations

XML layout for portrait

Different app layouts for portrait vs landscape screen orientation



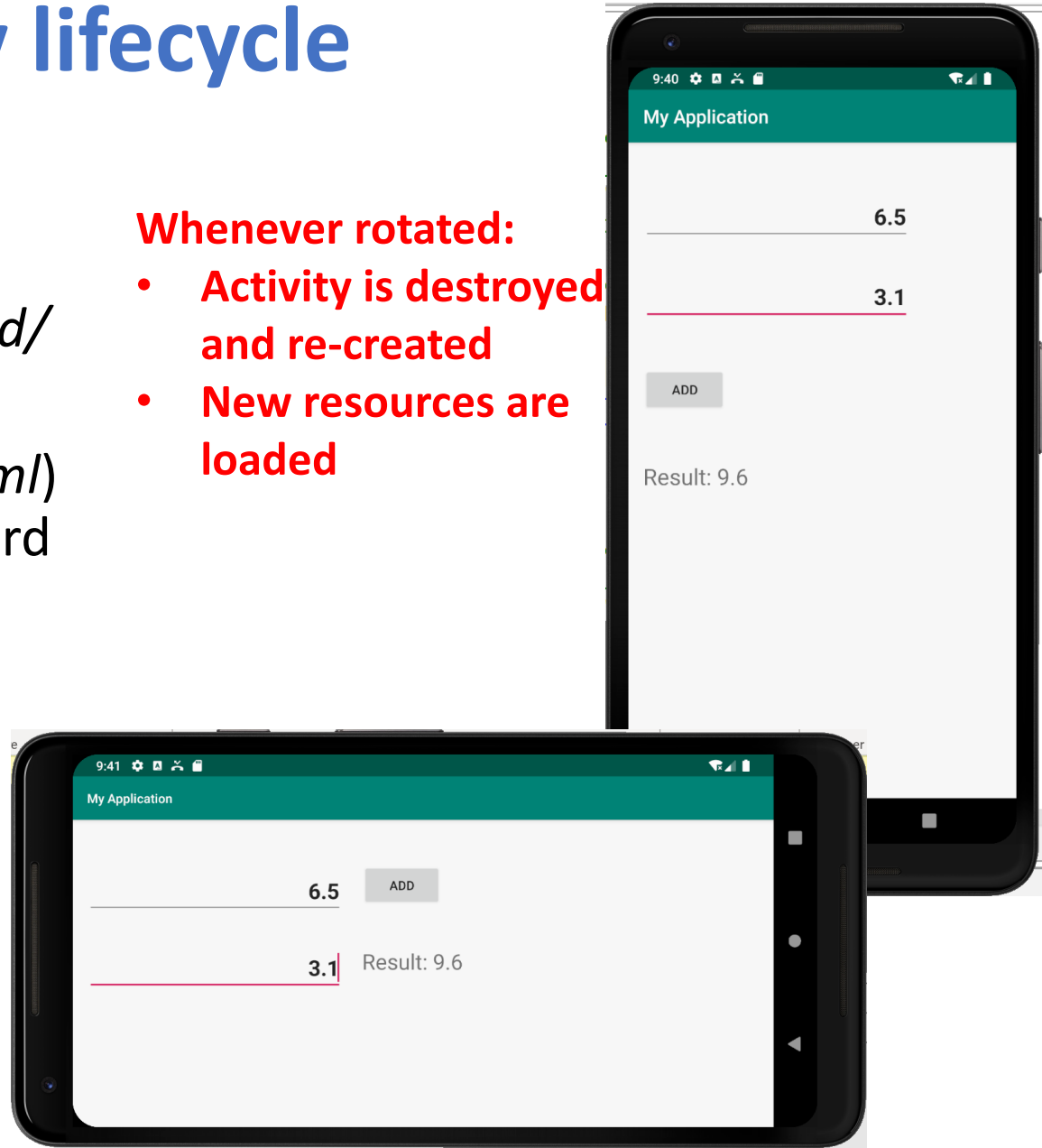
XML layout for landscape

Device rotation and activity lifecycle

- Portrait: use XML layout file in *res/layout*
- Landscape: use XML layout file in *res/layout-land/*
- Right click on layout folder: «New» -> «Layout resource file», use same name (*activity_main.xml*) specify an «orientation» qualifier using the wizard then modify the XML file
- If configuration changes, current activity destroyed, *onCreate() -> setContentView (R.layout.activity_main)* called again

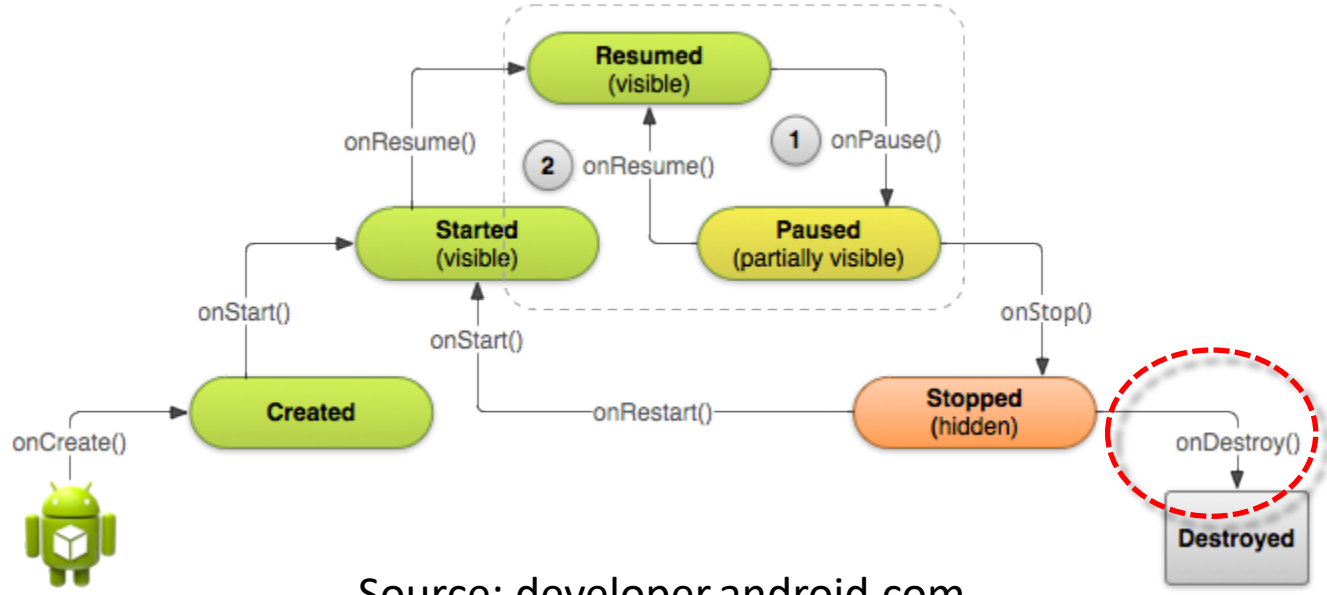
Whenever rotated:

- **Activity is destroyed and re-created**
- **New resources are loaded**



When app is destroyed

- The *onDestroy()* method is called just before the app is destroyed
- It can be implemented to free some resources like terminating additional threads created by the application

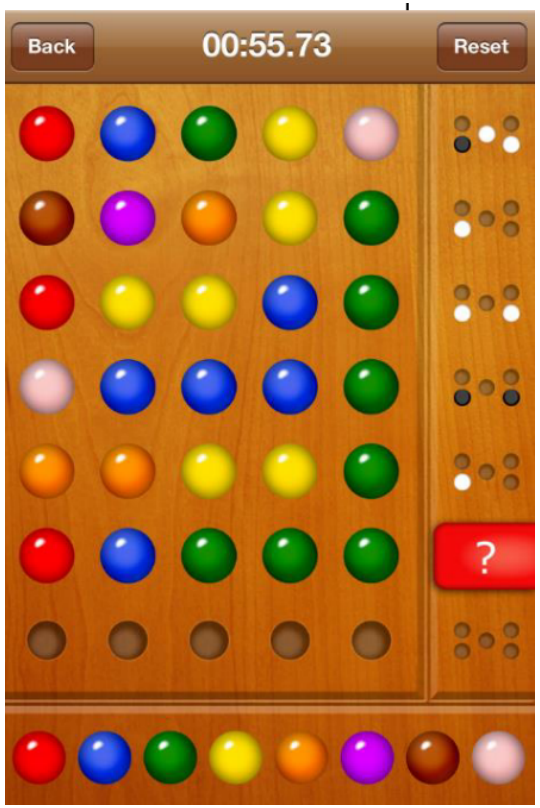


Source: developer.android.com

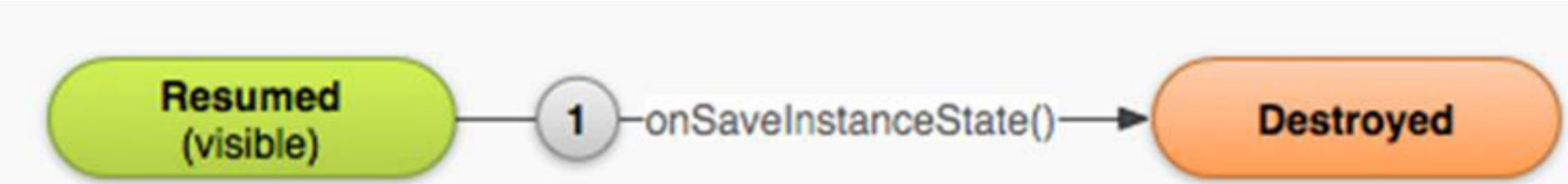
Saving state

- **Problem:** when an activity is destroyed and created again some state may be lost
- Some widgets automatically save their state and restore it when the activity is re-created
- In some cases the programmer has to implement his own status saving strategies
  - E.g. for possible instance variables of an Activity
- When the user presses the back button the activity is destroyed

Saving state

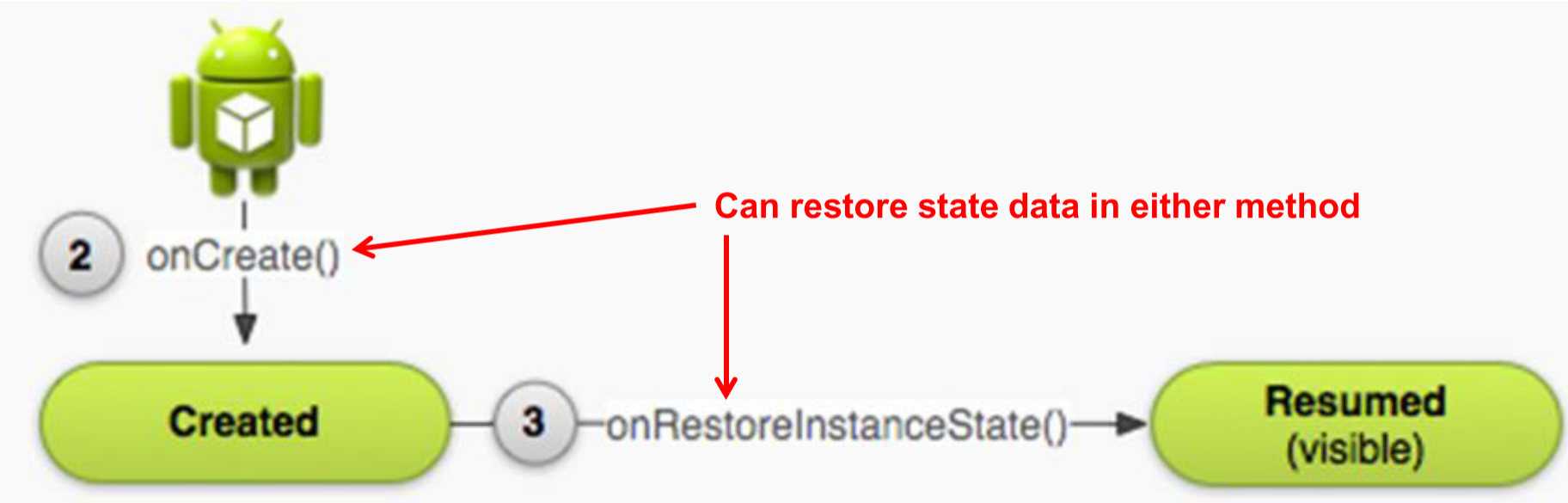


- Before Activity destroyed, system calls *onSaveInstanceState()*
- Can save state required to recreate Activity later
  - E.g. Save current positions of game pieces
- A *Bundle* object is used as container of information to be saved
- The Bundle is given to the system when the Activity is going to be destroyed



Restoring app state

- When an Activity recreated saved data sent to *onCreate()* and *onRestoreInstanceState()*
- Can use either method to restore app state data

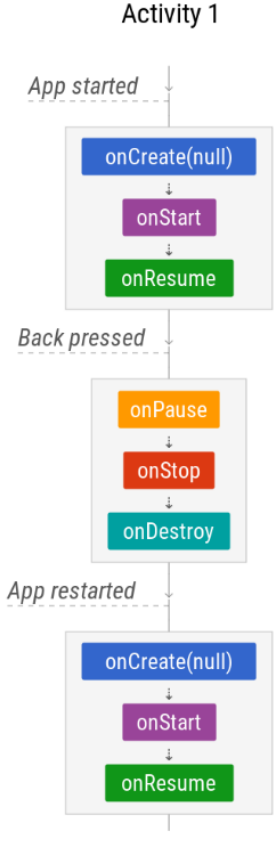


https://developer.android.com/guide/components/activities/activity-lifecycle

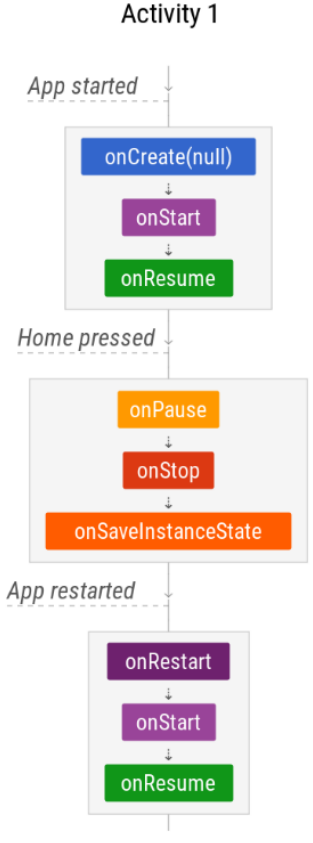
Common scenarios

Single activities

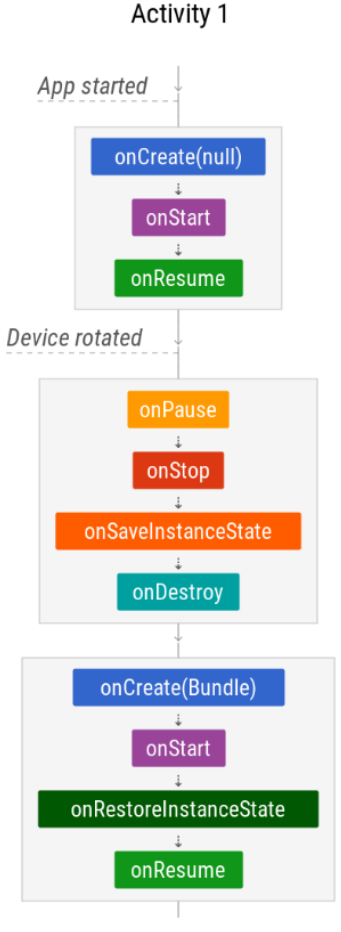
Start and finish



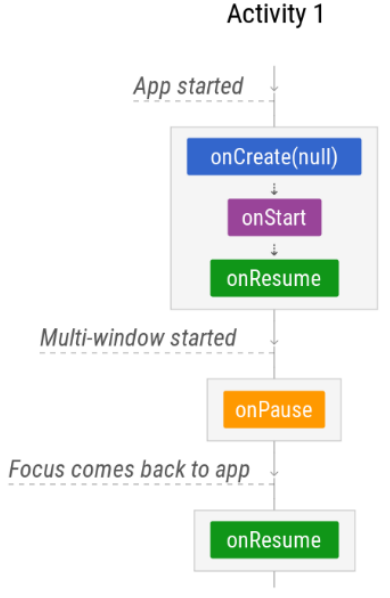
User navigates away



Rotation (configuration changes)

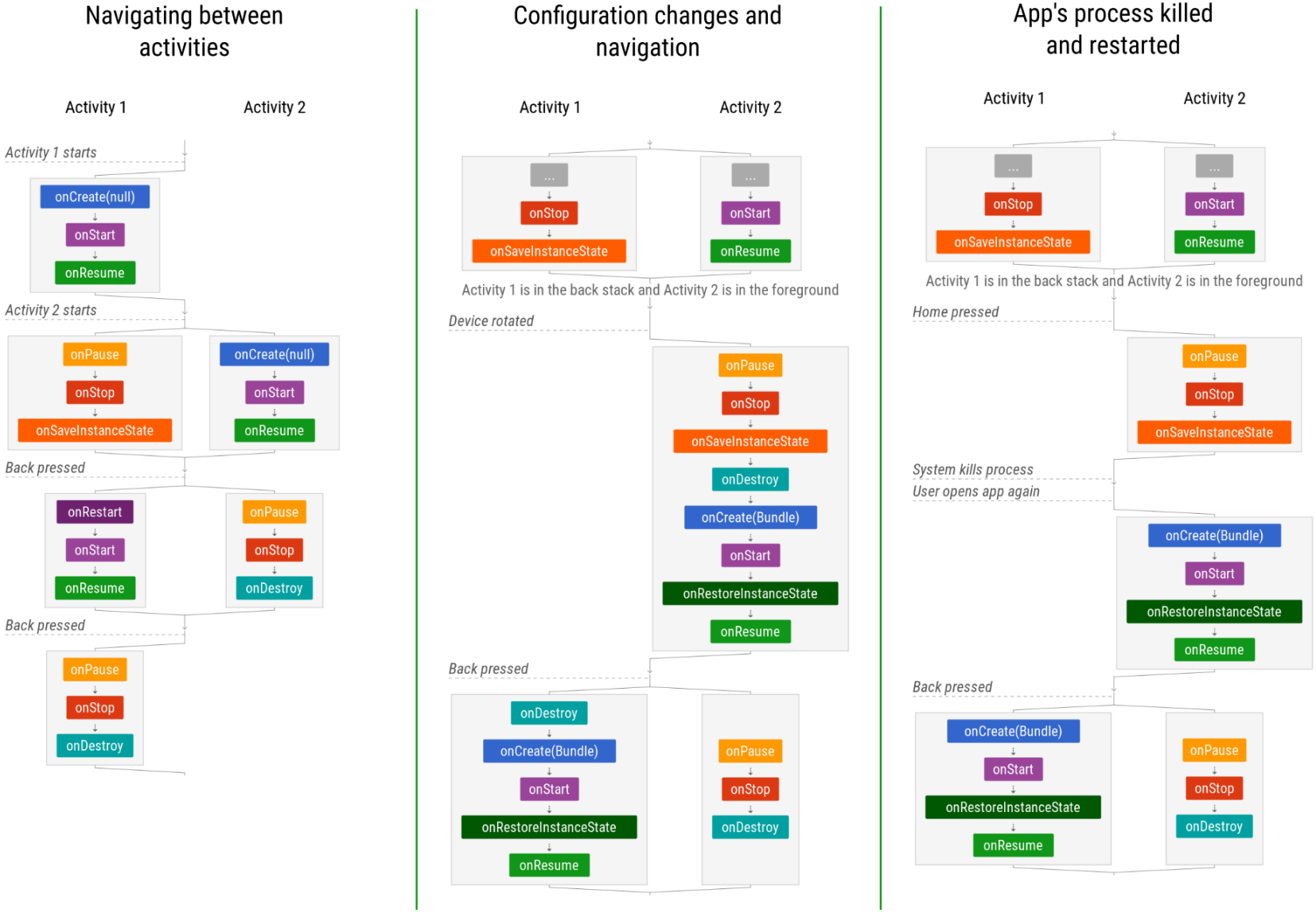


App is paused



Common scenarios

Multiple activities / back stack



References

- CS 528 Mobile and Ubiquitous Computing, WPI
- CS 65/165 slides, Dartmouth College
- CS 371M slides, University of Texas Austin
- Busy Coder’s guide to Android version 4.4
- http://developer.android.com