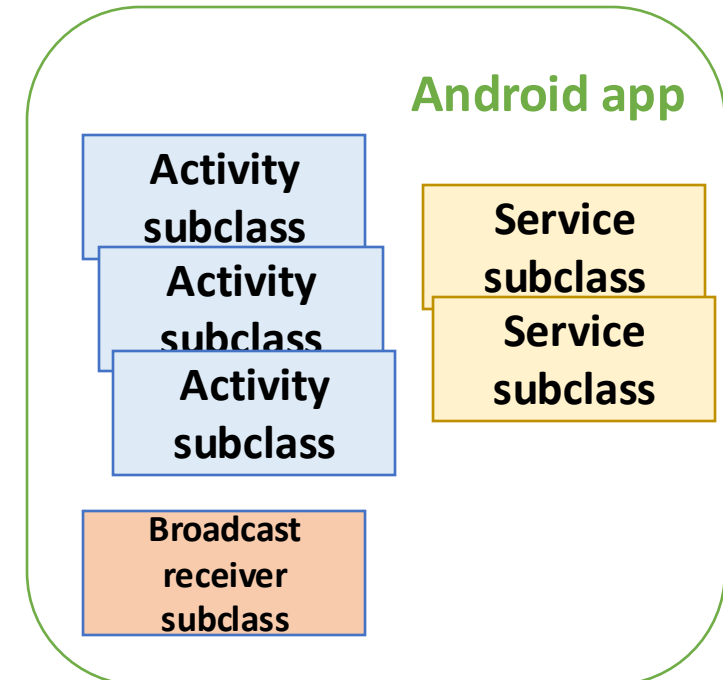
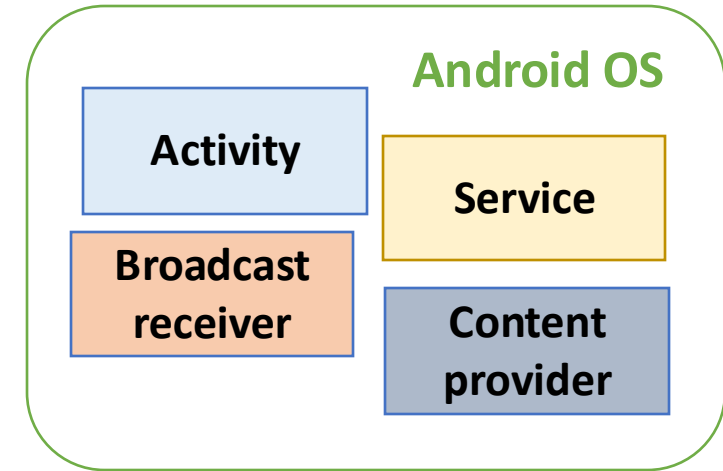


# **Activity lifecycle and other Android components**

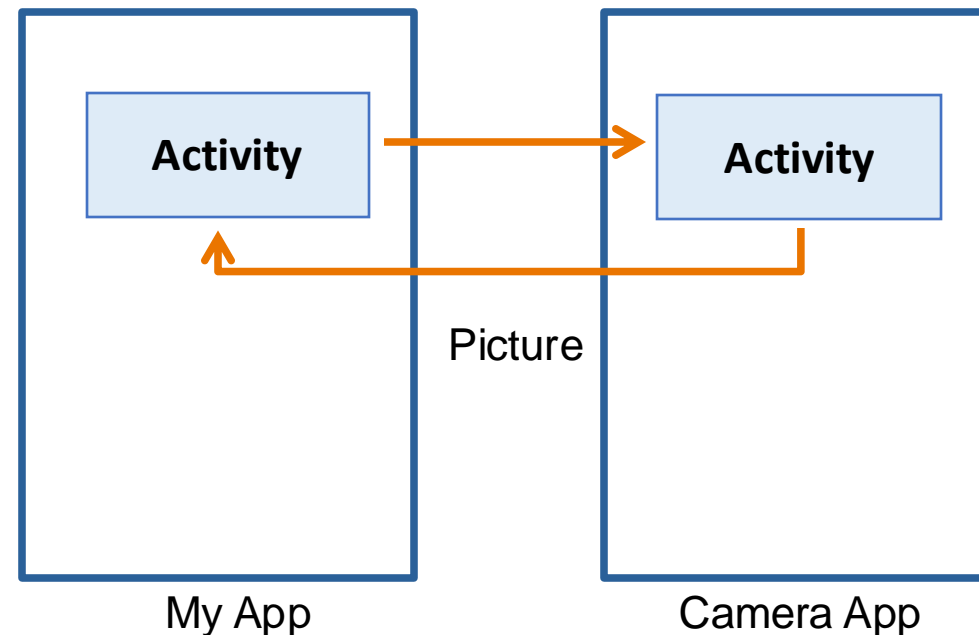
# Android applications

- 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



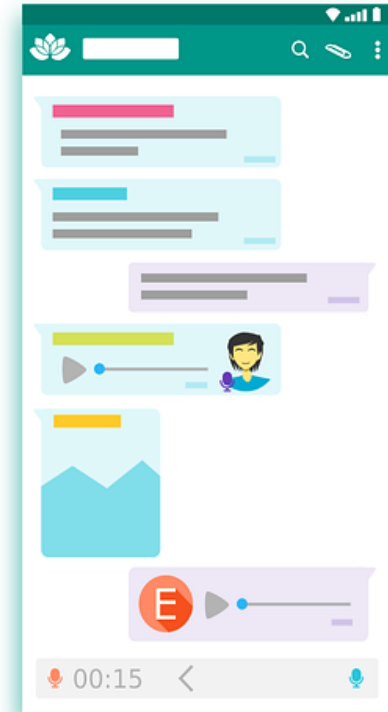
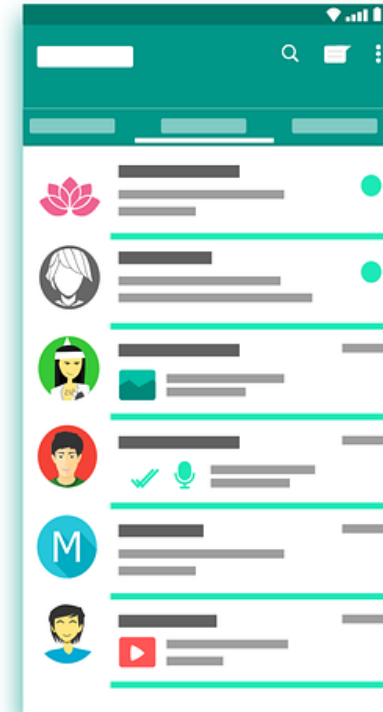
# Applications

- 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



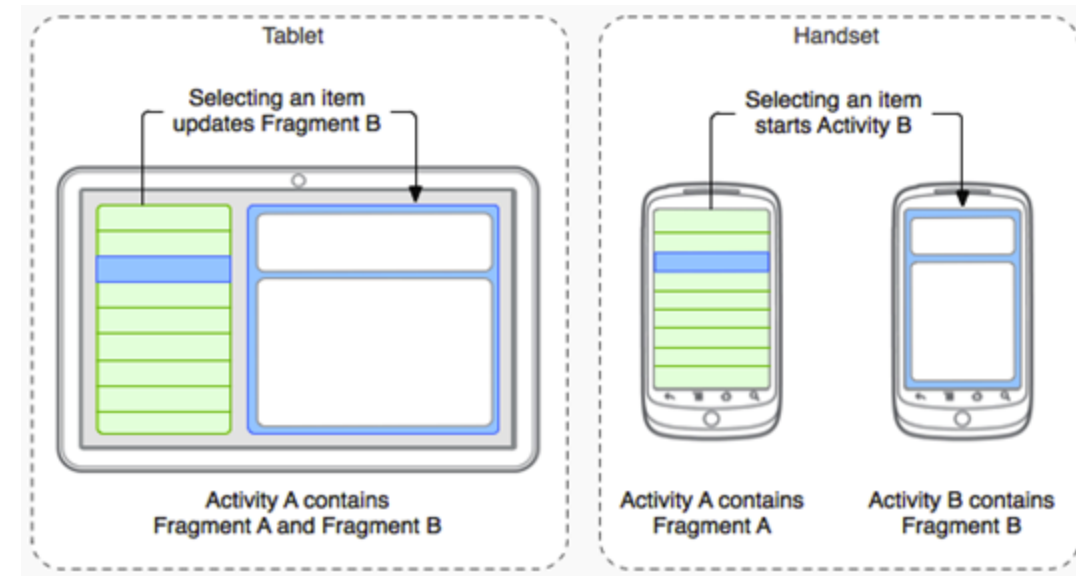
# Activities (recap)

- 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)



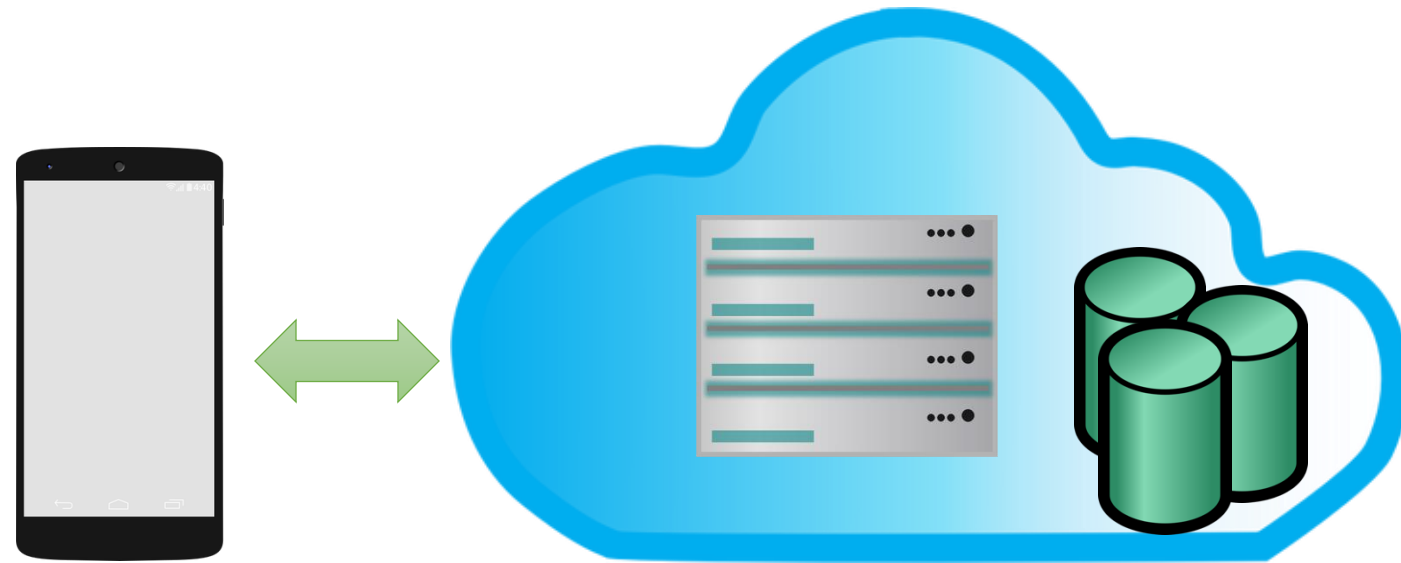
# Services

- Activities are short-lived, can be shut down anytime (e.g when user presses back button)
- Services: components without a UI
- Frequently, an activity will control a service
  - start it
  - get data from it
- Services in an App are sub-classes of Android's *Service* class
- Can be independent from activities
- Can be started by timers, e.g. to
  - Periodically check/update device's GPS location
  - Check for new email messages



# 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



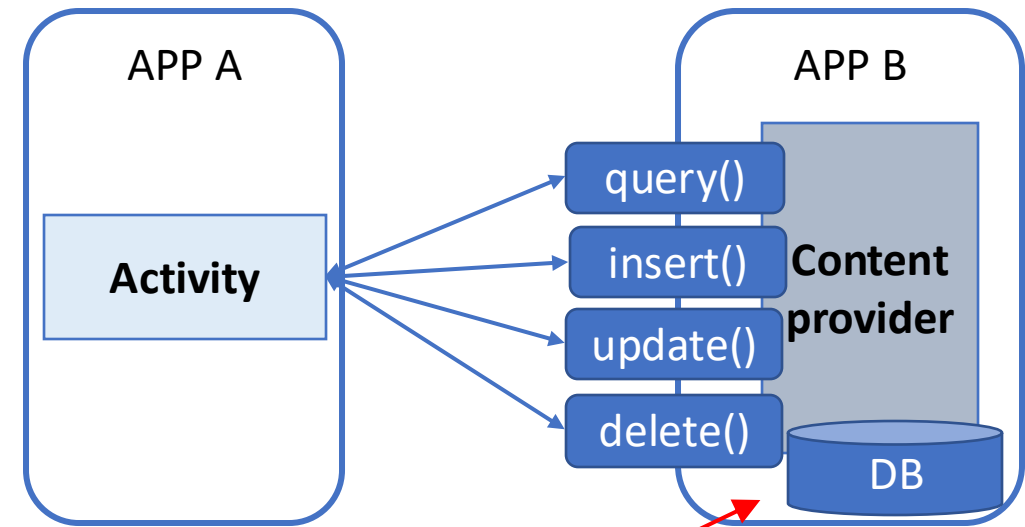
# Managers

- 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



# Content providers

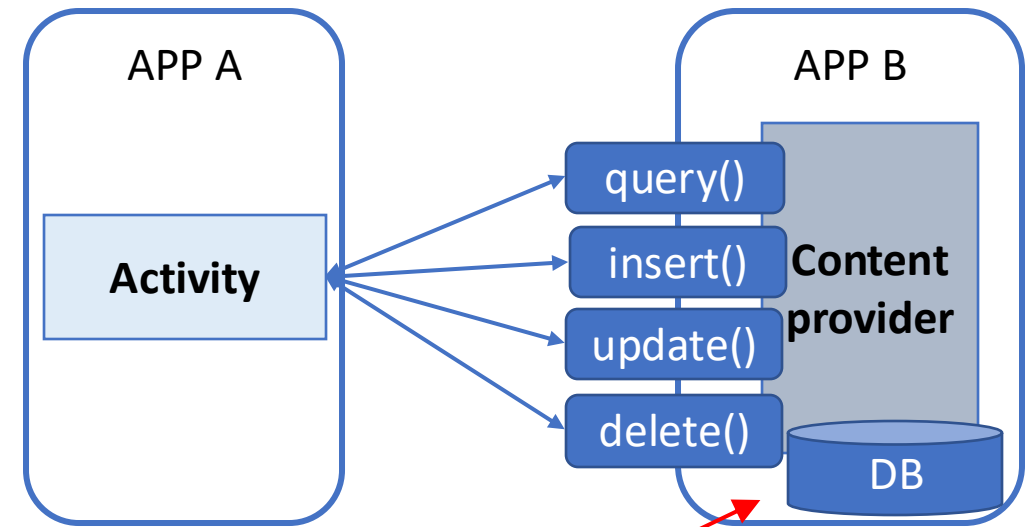
- Android apps can share data as content providers
  - User's contact list
  - Media files
- Content Provider:
  - Abstracts shareable data, makes it accessible through methods
  - Applications can access that shared data by calling methods for the relevant content provider
- Example: query, insert,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



**Data to be shared with other apps**

# 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



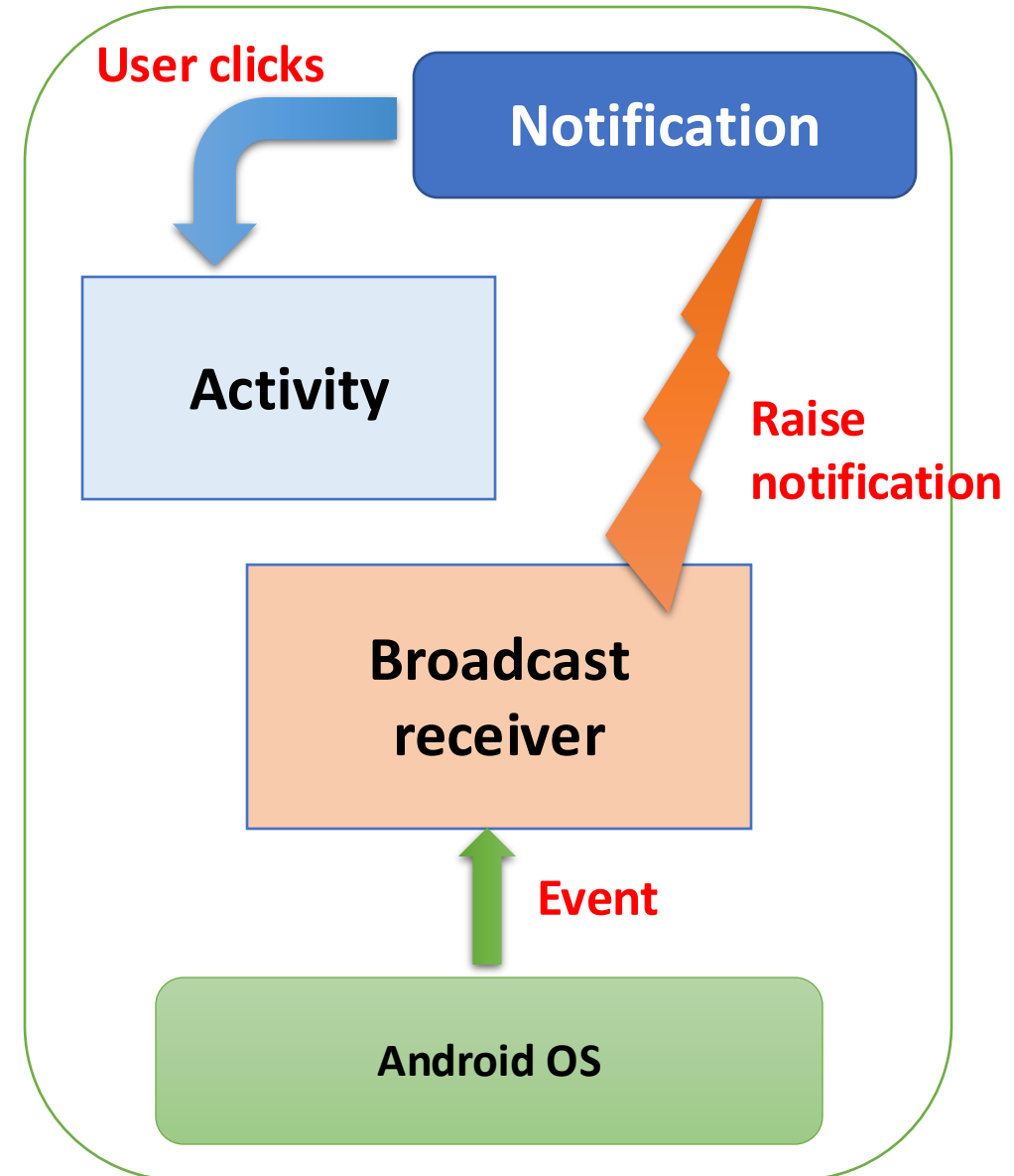
Data to be shared with other apps

# Broadcast receivers

- Android OS (system), or applications, periodically broadcasts events
- Example broadcasts:
  - Battery getting low
  - In/out of airplane mode
  - USB device attached
  - Change of network country
- 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

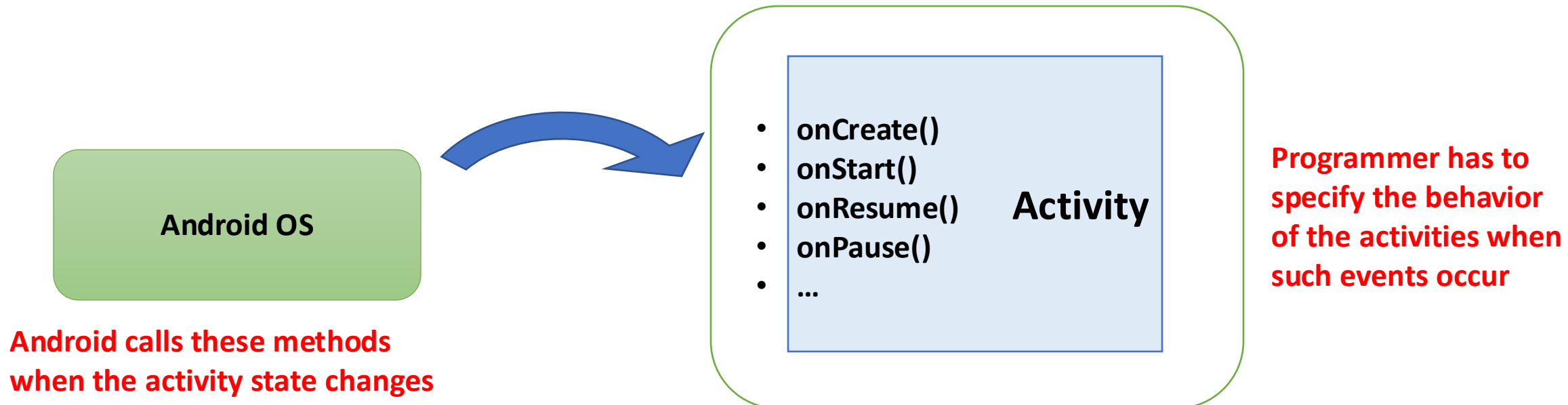
# Broadcast receivers

- 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 always defined: the app inflates layout to provide UI

# Activity lifecycle

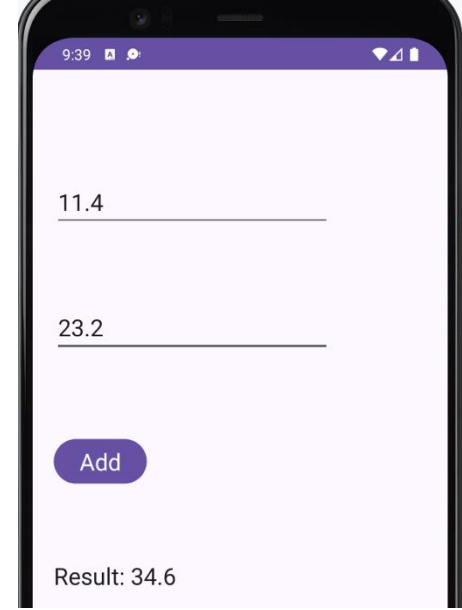
- Many things could happen while app is running
  - Incoming call or text message
  - User switches to another app
  - Train arrives and user puts device in his pocket
- Well designed app should not:
  - Crash if interrupted, or user switches to other app
  - Lose the user's state/progress (e.g state 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



# 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()* or binding)
  - Setting listeners to handle user interaction (e.g. clicking buttons)

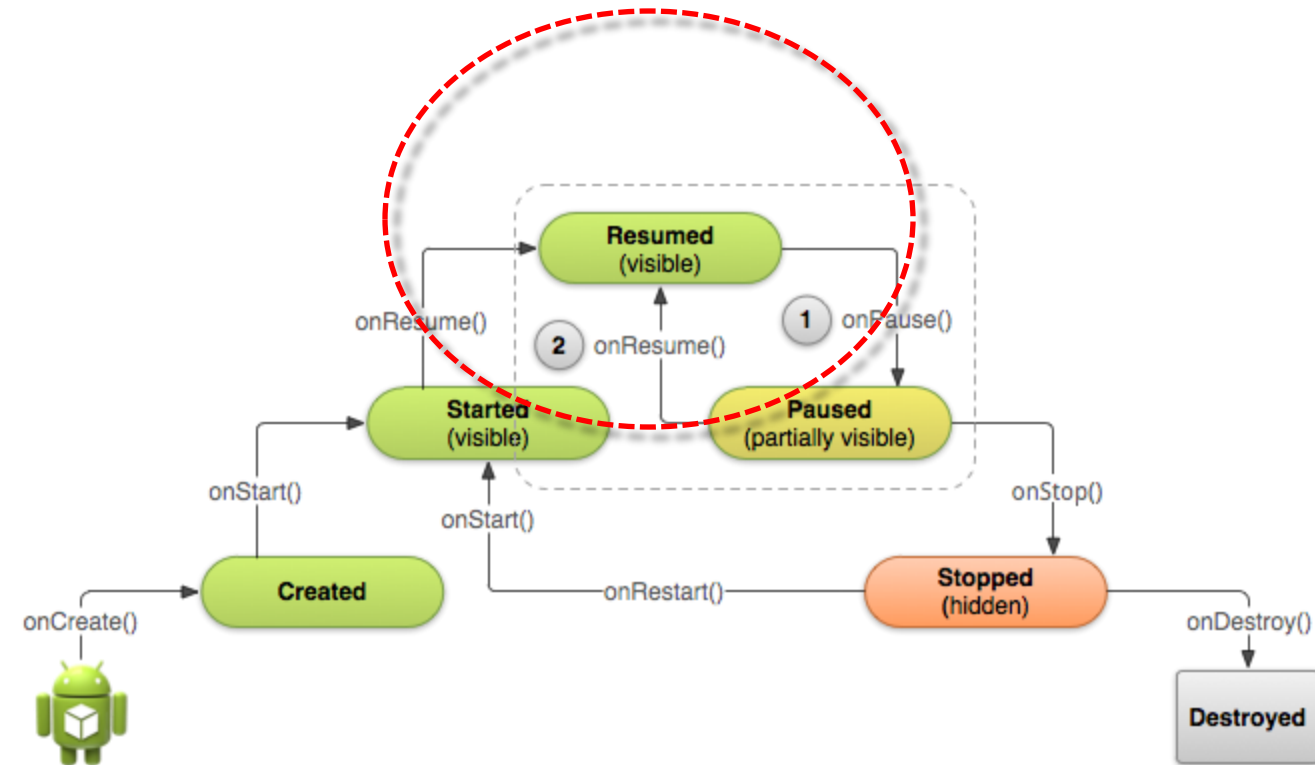
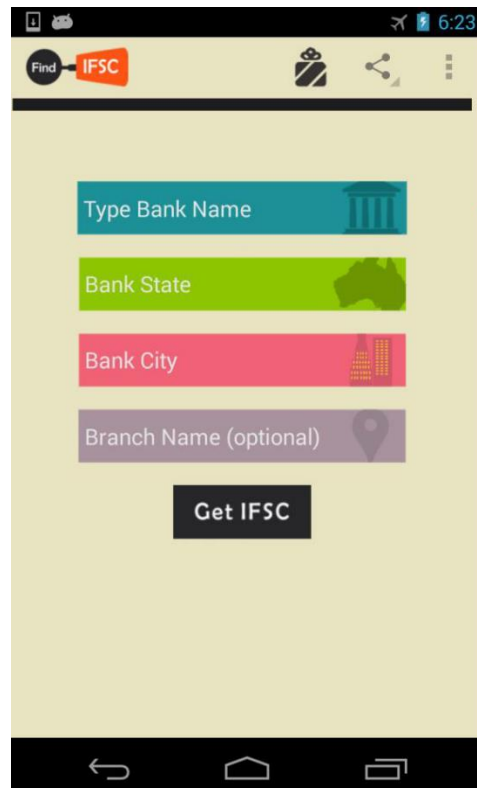
```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        val but: Button = findViewById(R.id.button)  
        but.setOnClickListener {  
            val t1: EditText = findViewById(R.id.edit1)  
            val t2: EditText = findViewById(R.id.edit2)  
            val r: TextView = findViewById(R.id.result)  
            val f1 = t1.text.toString().toDouble()  
            val f2 = t2.text.toString().toDouble()  
            r.text = "Result: ${f1+f2}"  
        }  
    }  
}
```





# Activity lifecycle

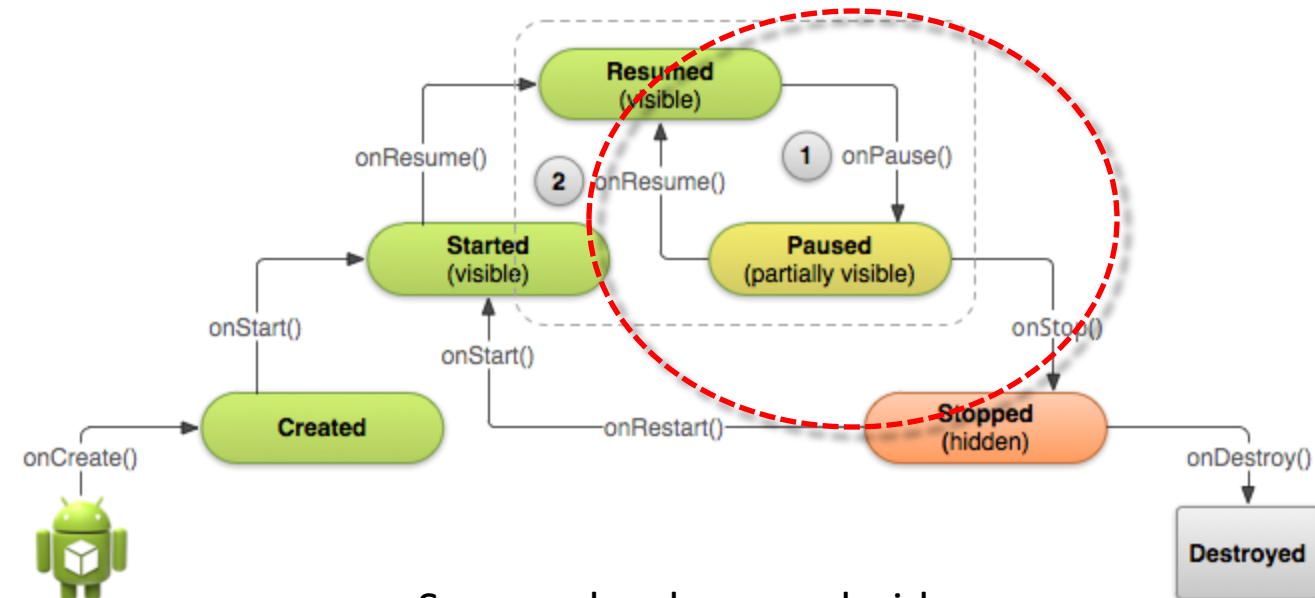
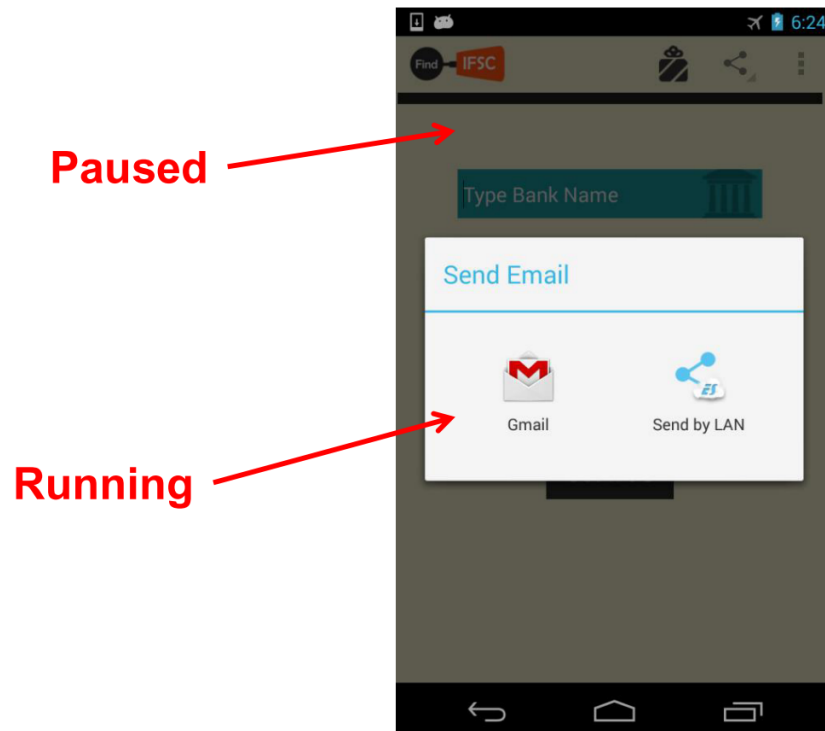
- A running app is one that user is currently using or interacting with
  - Visible, in foreground



Source: [developer.android.com](https://developer.android.com)

# Paused app

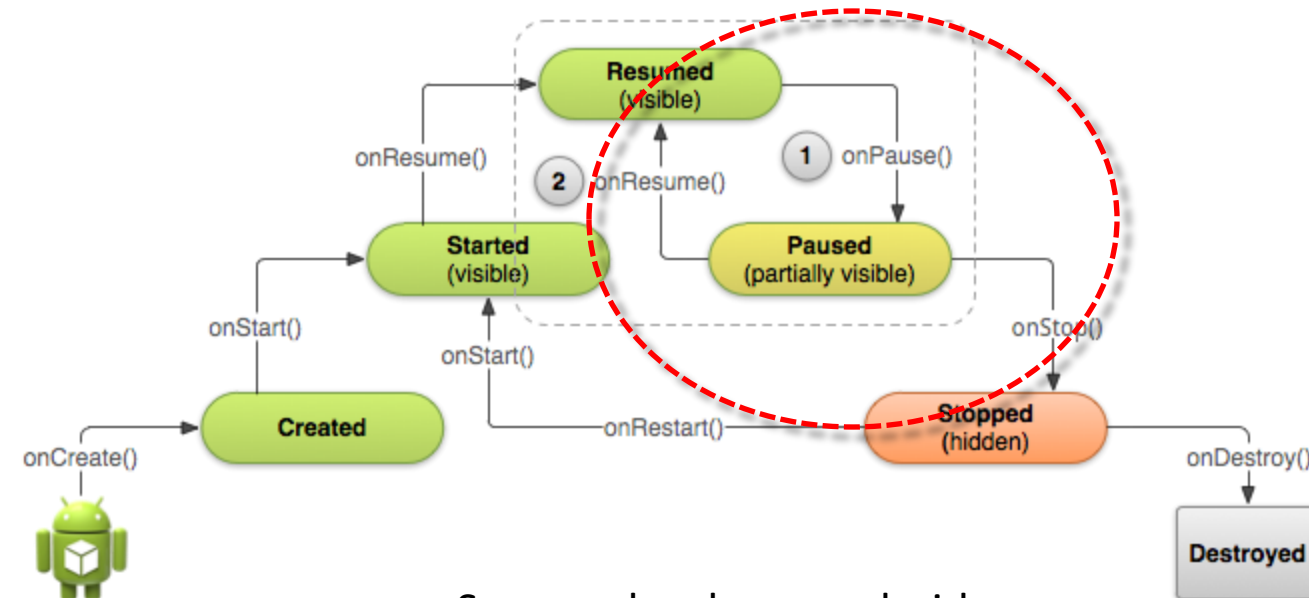
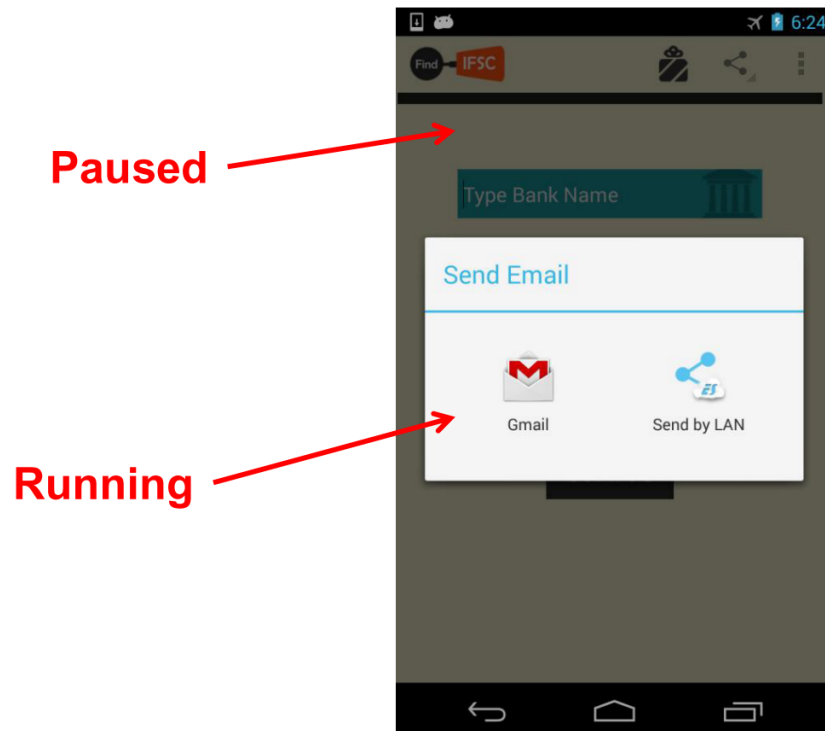
- An app is paused if it is visible but no longer in foreground (e.g. blocked by a pop-up dialog box)
- App's *onPause()* method is called during transition from running to paused state



Source: [developer.android.com](http://developer.android.com)

# Paused app

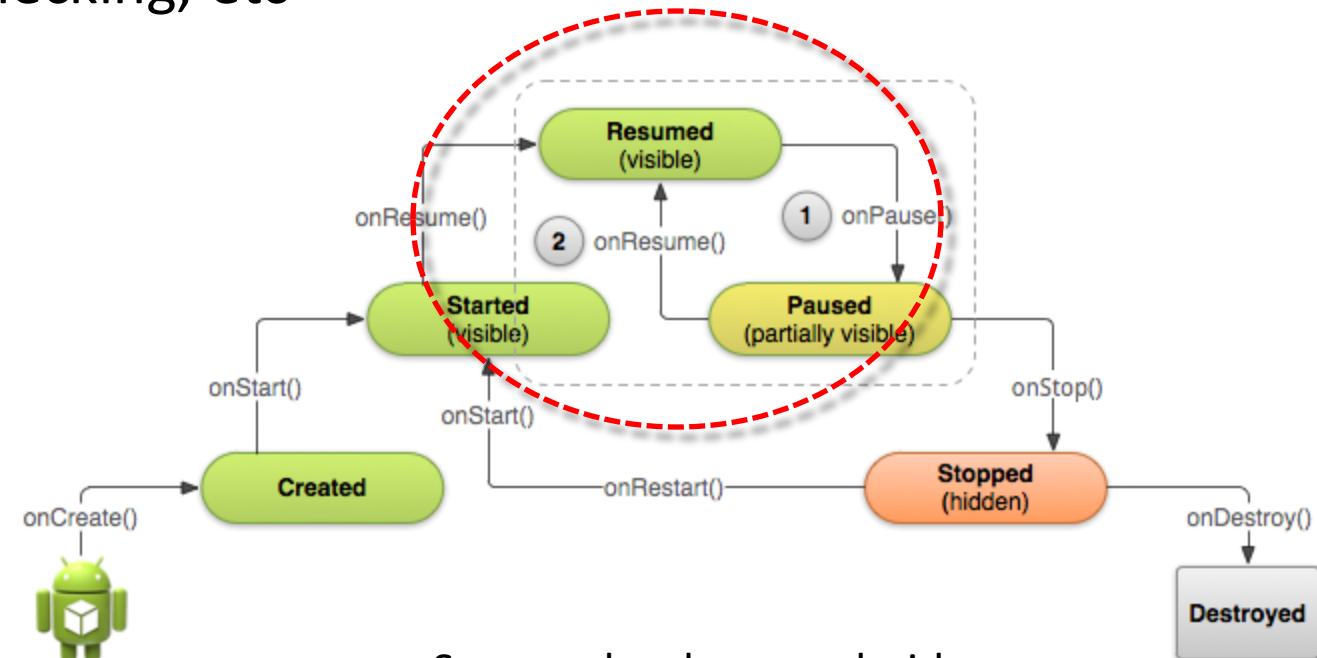
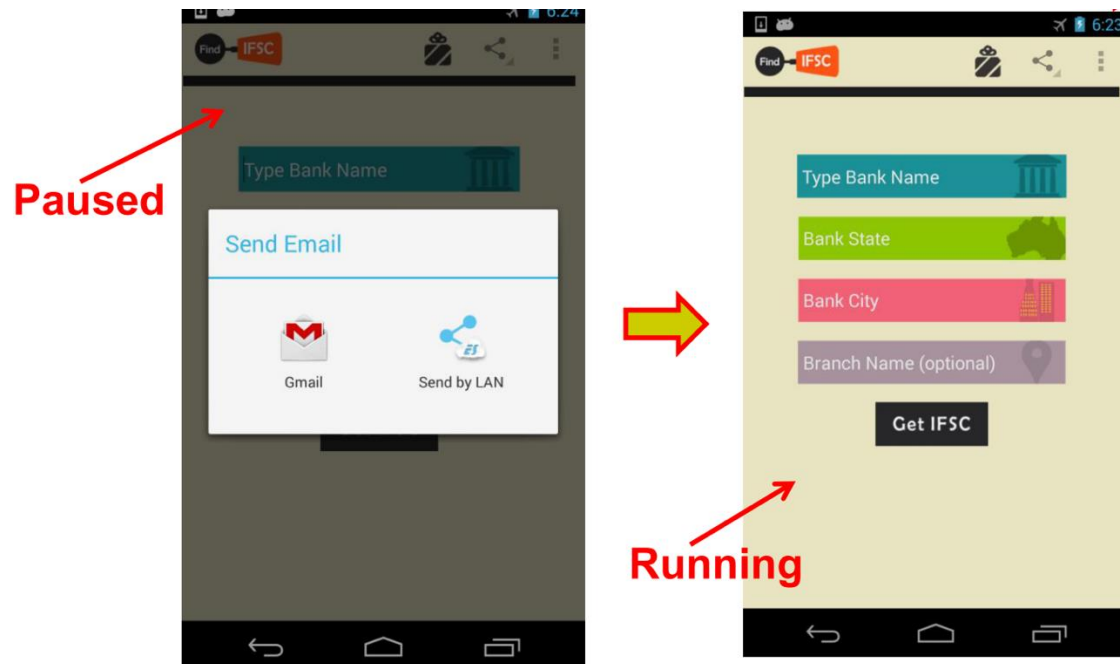
- Typical actions taken in *onPause()* method
  - Stop CPU intensive tasks, stop audio, video, animations
  - Stop listening for GPS, broadcast information, release handles to sensors (e.g GPS, accelerometer)



Source: [developer.android.com](http://developer.android.com)

# onResume(): Resuming Paused App

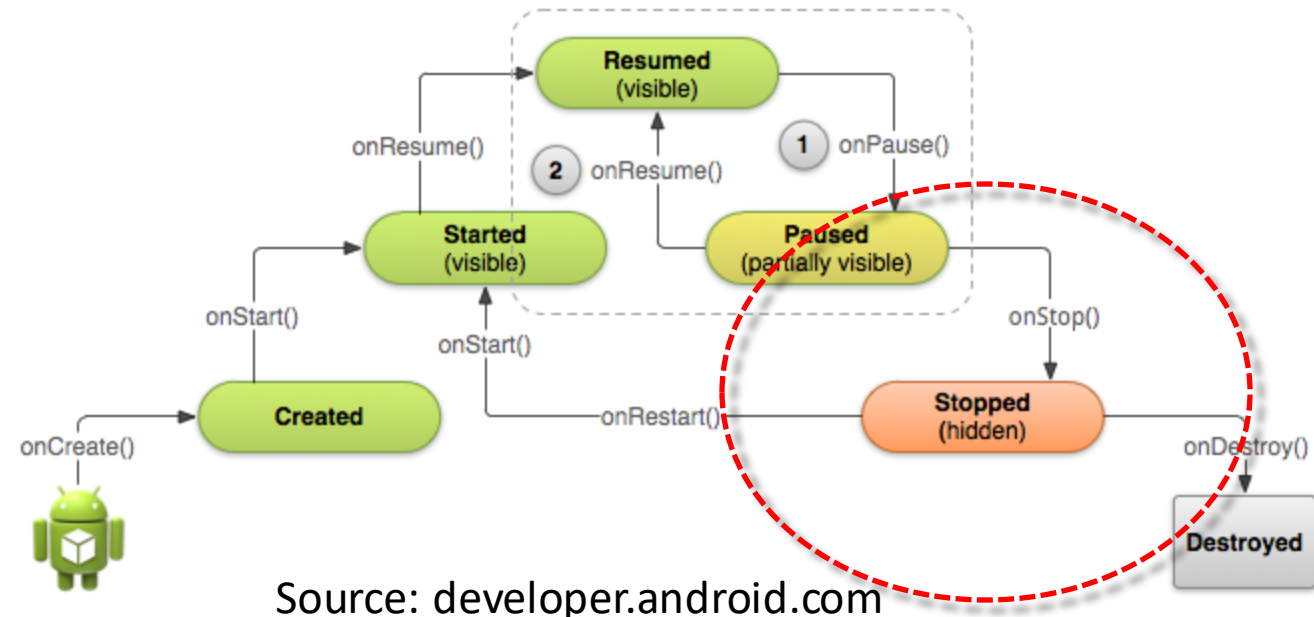
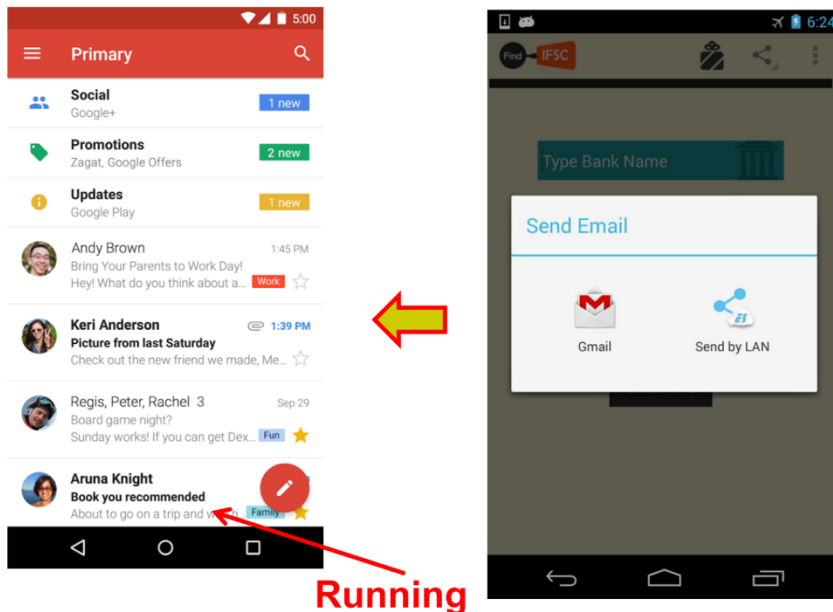
- A paused app resumes running if it becomes fully visible and in foreground
  - E.g. pop-up dialog box blocking it goes away
- App's *onResume()* method is called during transition from paused to running state
  - Restart videos, animations, GPS checking, etc



Source: developer.android.com

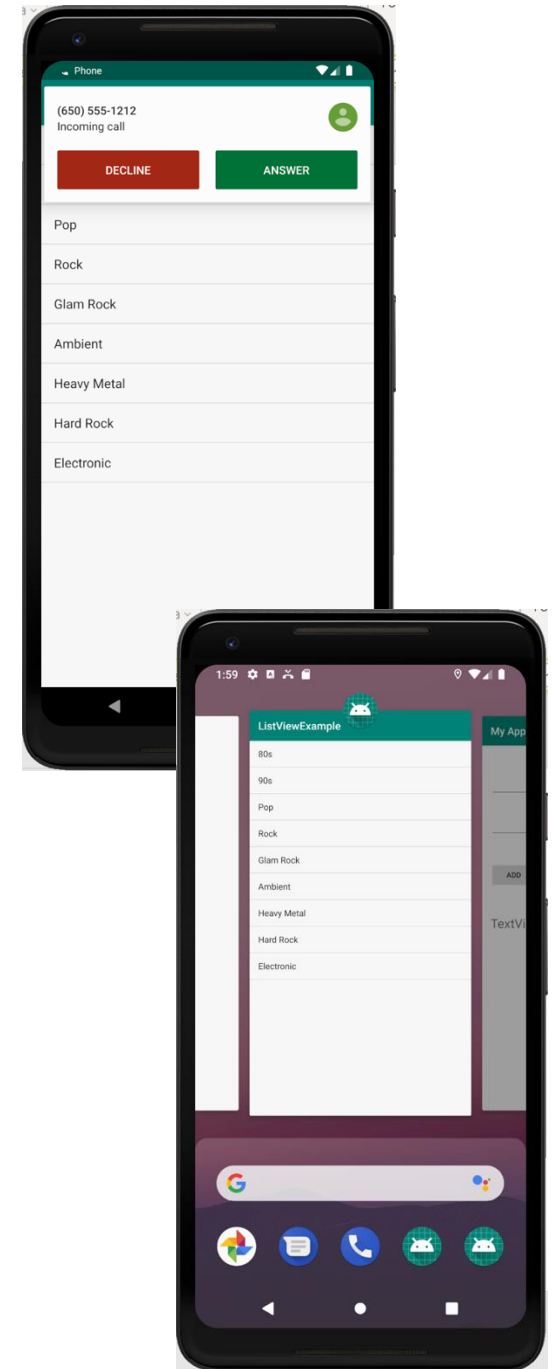
# Stopped app

- 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



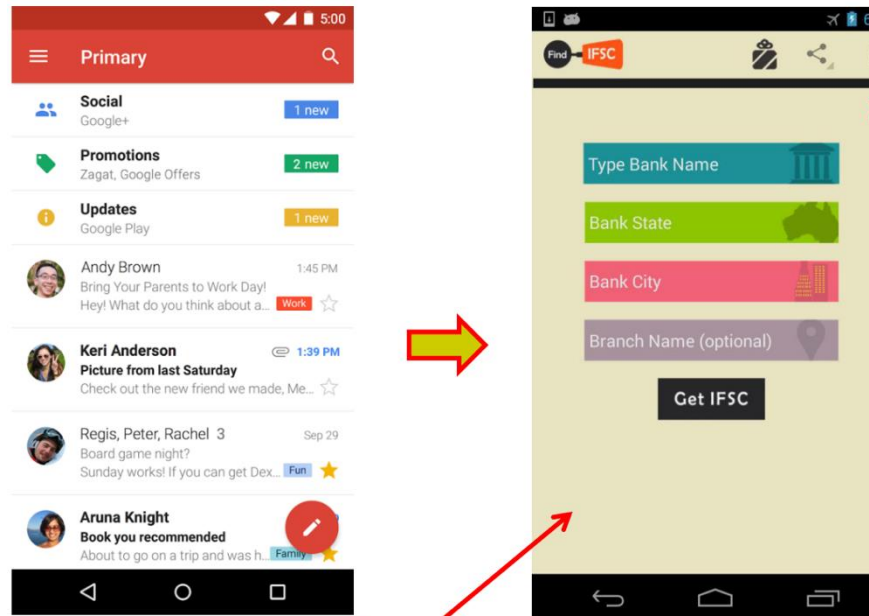
# onStop() Method

- 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 (persistence)
  - release all resources

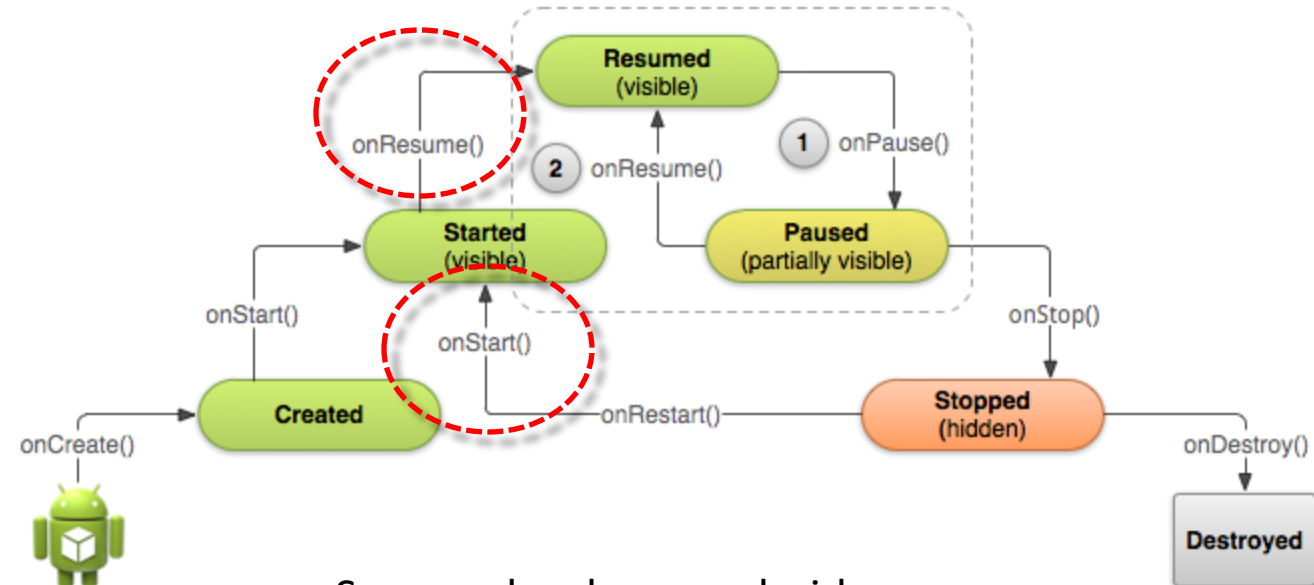


# Resuming a stopped app

- 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



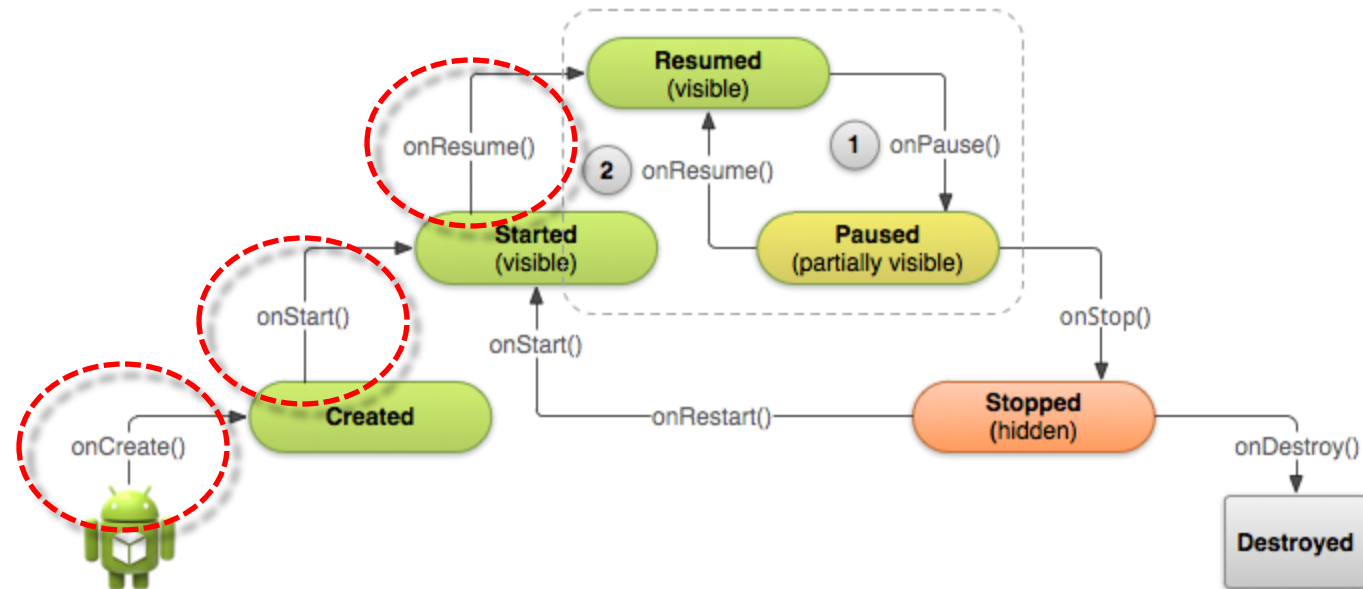
Running



Source: developer.android.com

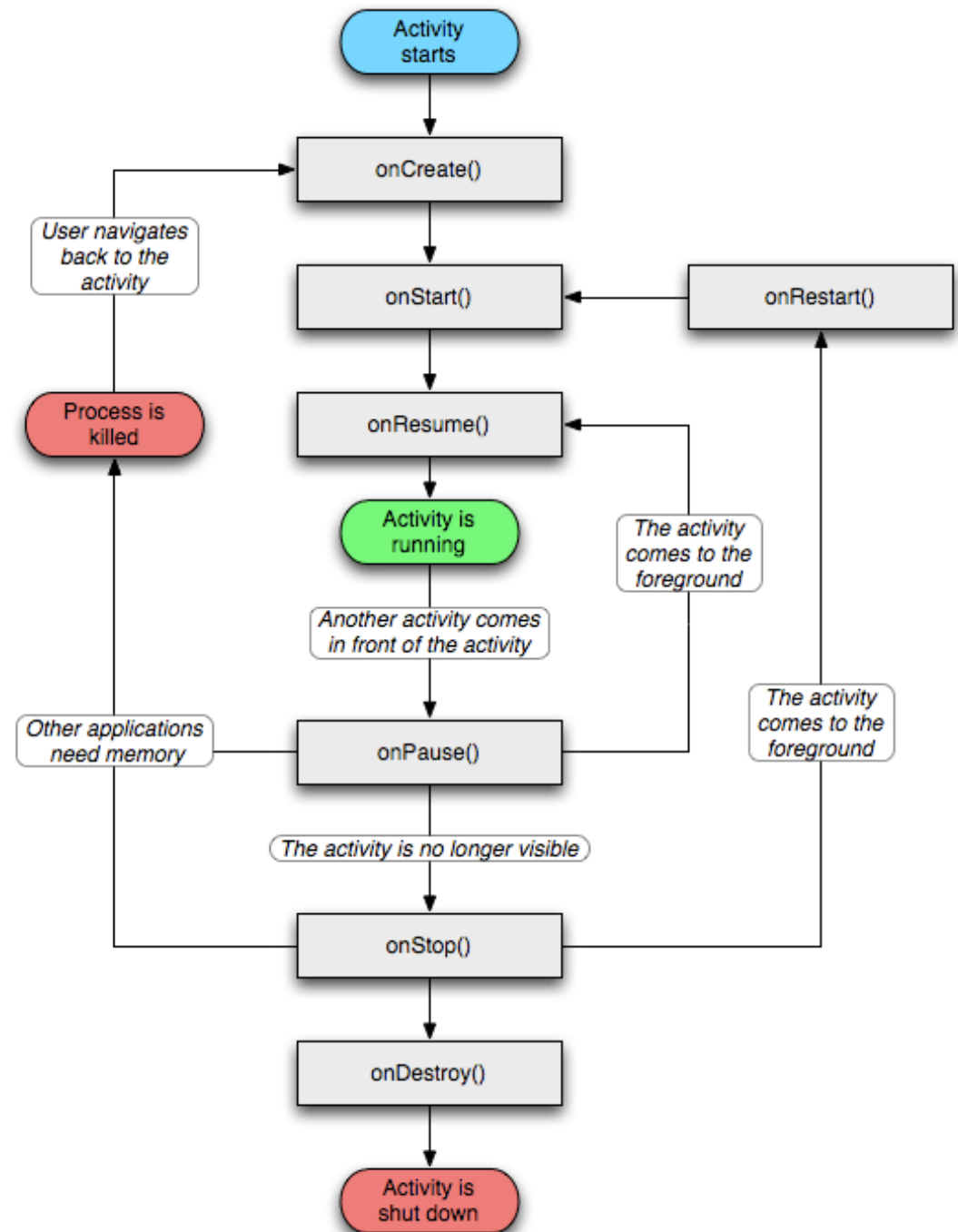
# Starting new app

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





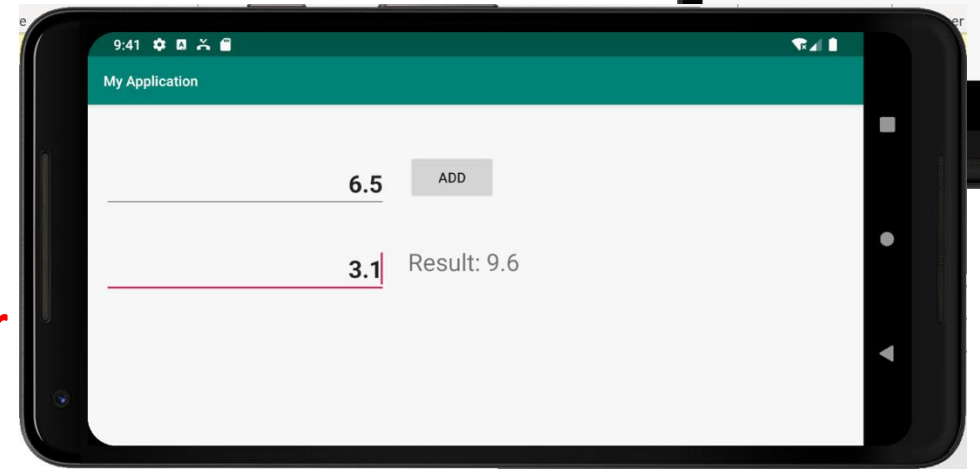
# The complete lifecycle



# Device rotation and activity lifecycle

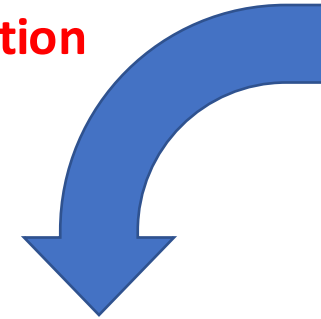
- Rotating device (e.g. portrait to landscape) kills current activity and creates new instance of same activity in landscape mode
- Rotation changes device configuration
- Device configuration: screen orientation/density/size, dock mode, language, etc.
- Apps can specify different resources (e.g. XML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

**XML layout for  
landscape**



**XML layout for  
portrait**

**Different app layouts  
for portrait vs  
landscape screen  
orientation**

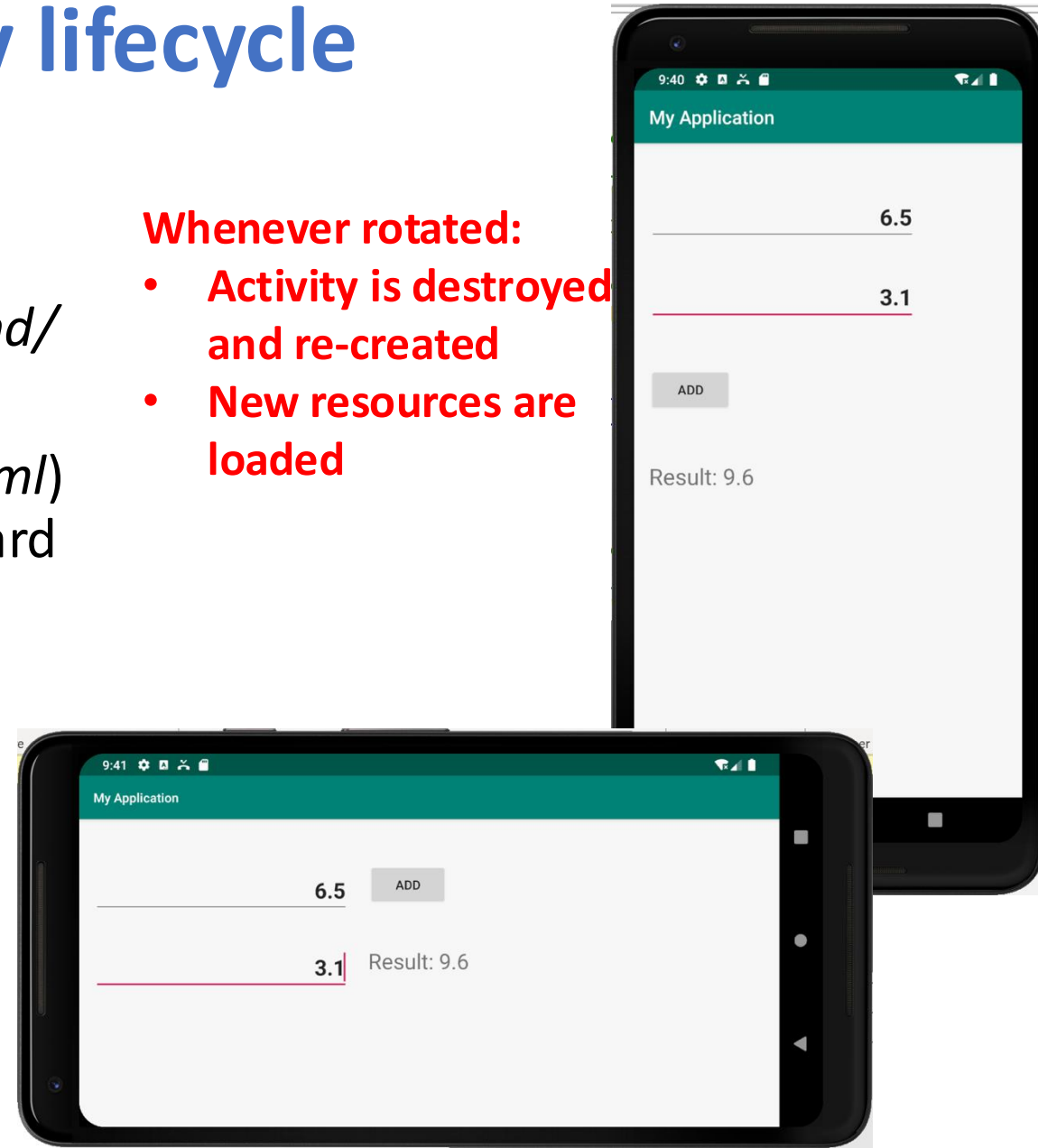


# Device rotation and activity lifecycle

- 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

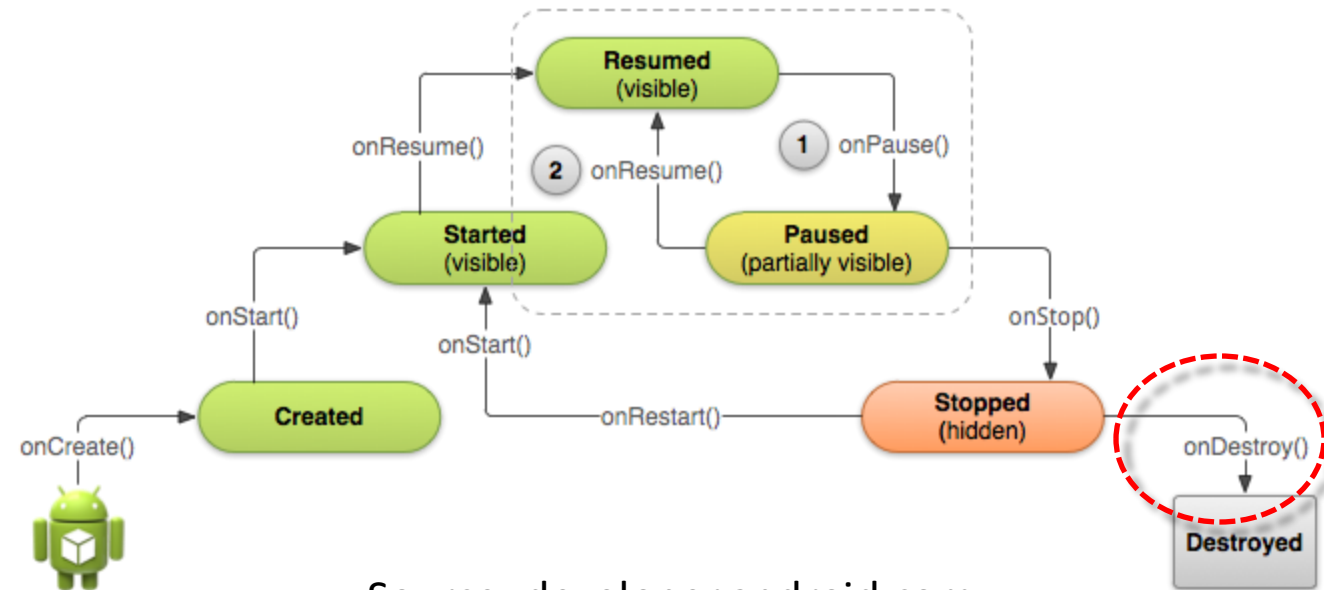
## Whenever rotated:

- 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



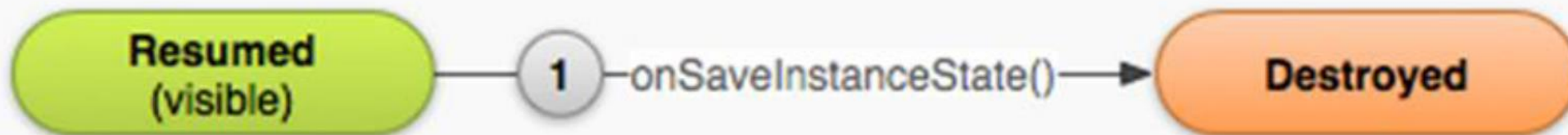
Source: [developer.android.com](http://developer.android.com)

# 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

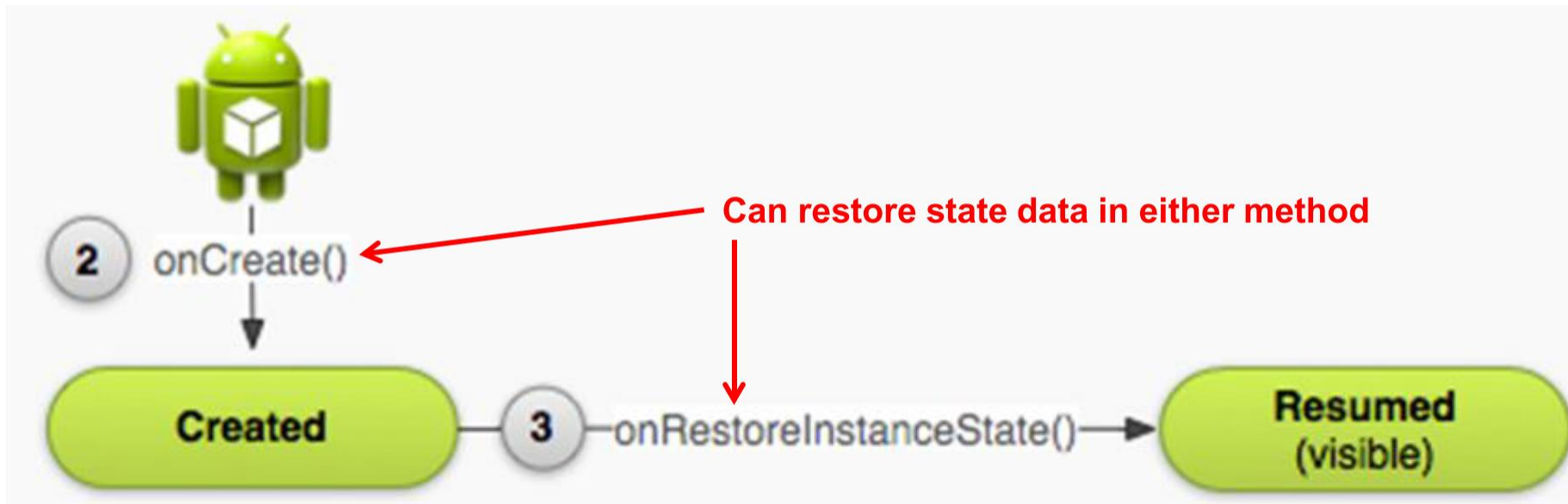
# Saving state

- 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

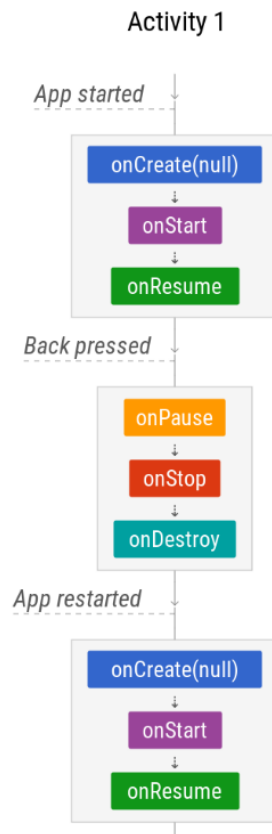
- When an Activity recreated saved data sent to *onCreate()* and *onRestoreInstanceState()*
- Can use either method to restore app state data



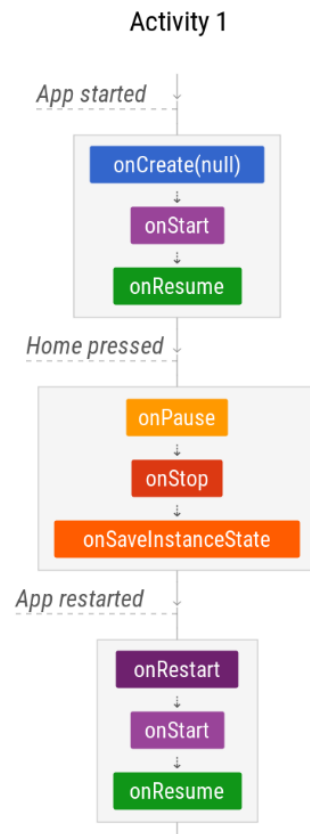
# Common scenarios

Single activities

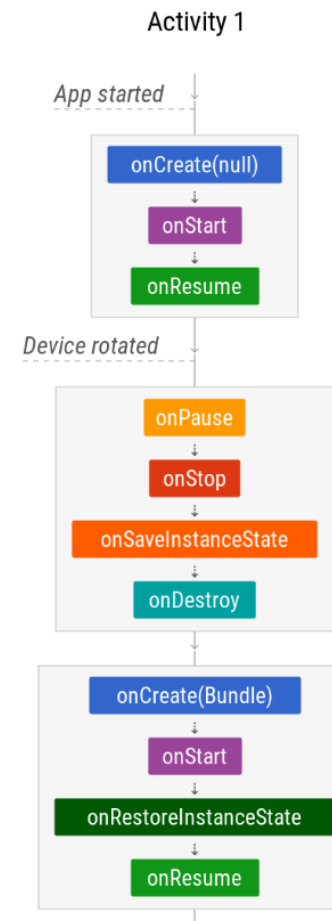
## Start and finish



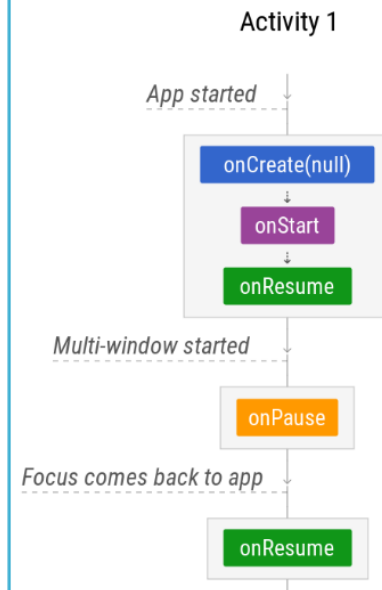
## User navigates away



## Rotation (configuration changes)



## App is paused

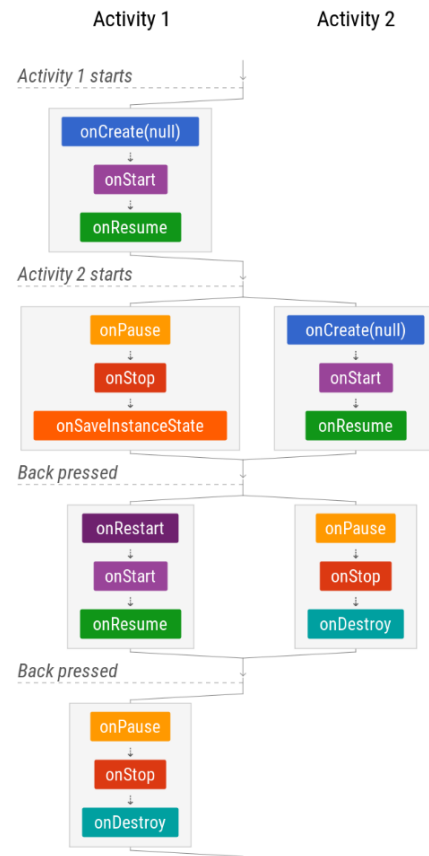




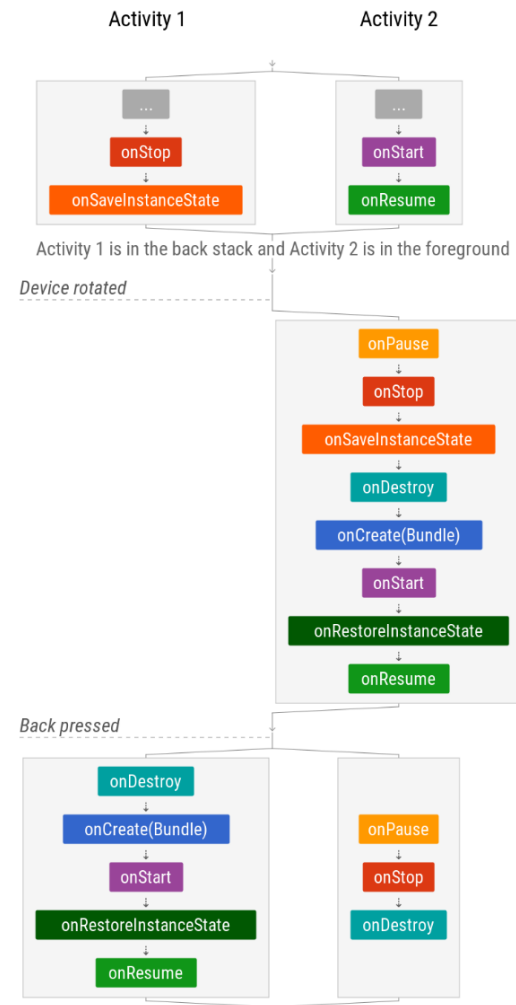
# Common scenarios

Multiple activities / back stack

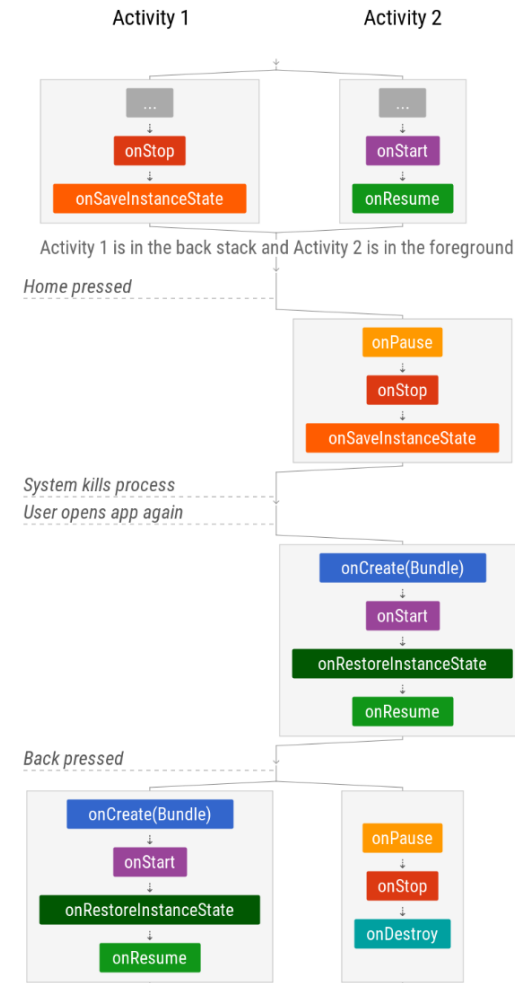
## Navigating between activities



## Configuration changes and navigation



## App's process killed and restarted



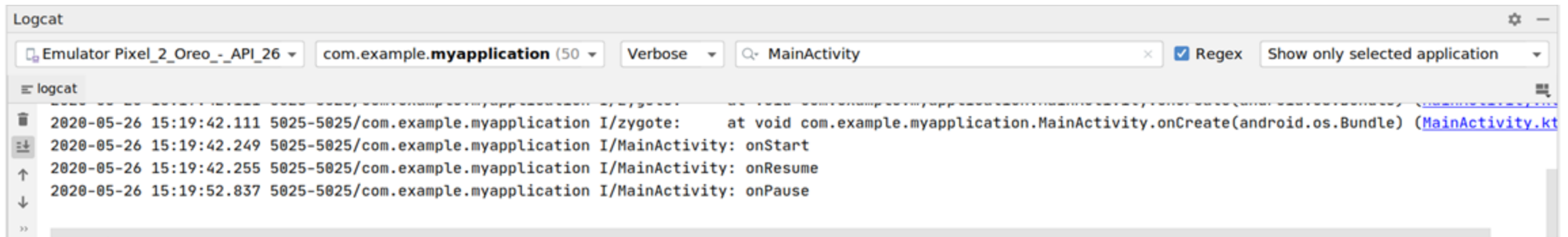
# References

- 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
- <http://developer.android.com>

# Logging

# Logging in Android

- Monitor the flow of events or state of your app.
- Use the built-in `Log` class or third-party library.
- Example `Log` method call: `Log.d(TAG, "Message")`

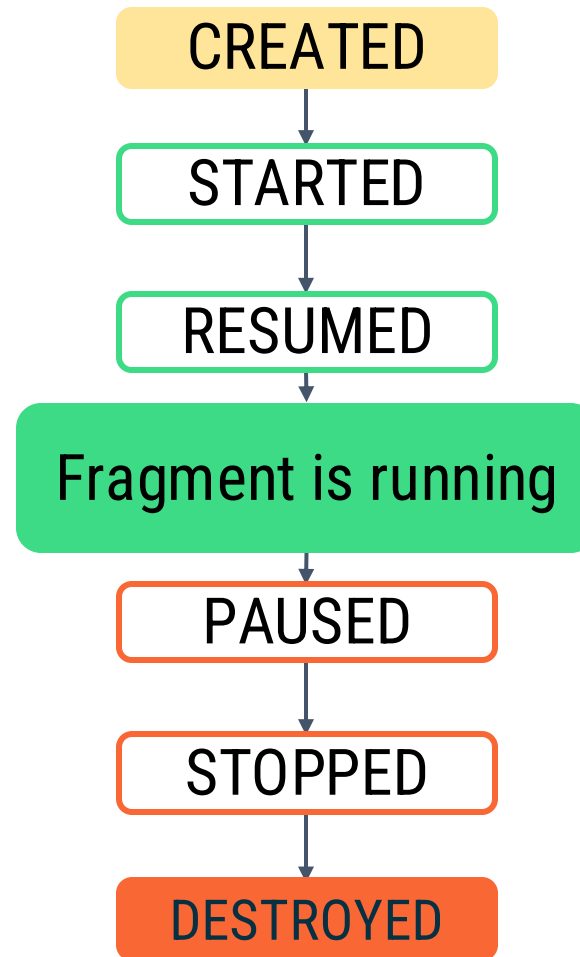


# Write logs

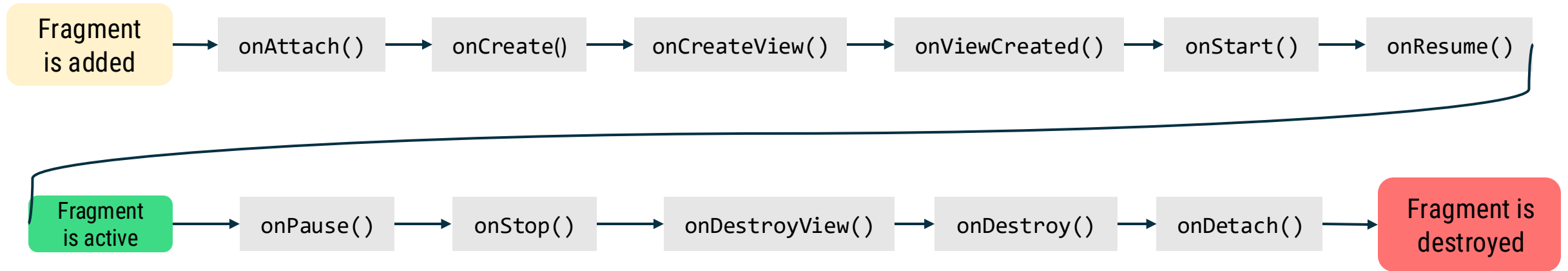
Priority level	Log method
Verbose	<code>Log.v (String, String)</code>
Debug	<code>Log.d (String, String)</code>
Info	<code>Log.i (String, String)</code>
Warning	<code>Log.w (String, String)</code>
Error	<code>Log.e (String, String)</code>

# Fragment lifecycle

# Fragment states



# Fragment lifecycle diagram





# onAttach()

- Called when a fragment is attached to a context
- Immediately precedes `onCreate()`

# onCreateView()

- Called to create the view hierarchy associated with the fragment
- Inflate the fragment layout here and return the root view

# onViewCreated()

- Called when view hierarchy has already been created
- Perform any remaining initialization here (for example, restore state from `Bundle`)

# onDestroyView() and onDetach()

- `onDestroyView()` is called when view hierarchy of fragment is removed.
- `onDetach()` is called when fragment is no longer attached to the host.

# Summary of fragment states

State	Callbacks	Description
Initialized	<code>onAttach()</code>	Fragment is attached to host.
Created	<code>onCreate()</code> , <code>onCreateView()</code> , <code>onViewCreated()</code>	Fragment is created and layout is being initialized.
Started	<code>onStart()</code>	Fragment is started and visible.
Resumed	<code>onResume()</code>	Fragment has input focus.
Paused	<code>onPause()</code>	Fragment no longer has input focus.
Stopped	<code>onStop()</code>	Fragment is not visible.
Destroyed	<code>onDestroyView()</code> , <code>onDestroy()</code> , <code>onDetach()</code>	Fragment is removed from host.

# Save fragment state across config changes

Preserve UI state in fragments by storing state in `Bundle`:

- `onSaveInstanceState(outState: Bundle)`

Retrieve that data by receiving the `Bundle` in these fragment callbacks:

- `onCreate()`
- `onCreateView()`
- `onViewCreated()`

# Lifecycle-aware components

# Lifecycle-aware components

Adjust their behavior based on activity or fragment lifecycle

- Use the `androidx.lifecycle` library
- `Lifecycle` tracks the lifecycle state of an activity or fragment
  - Holds current lifecycle state
  - Dispatches lifecycle events (when there are state changes)



# LifecycleOwner

- Interface that says this class has a lifecycle
- Implementers must implement `getLifecycle()` method

Examples: `Fragment` and `AppCompatActivity` are implementations of `LifecycleOwner`

# LifecycleObserver

Implement `LifecycleObserver` interface:

```
class MyObserver : LifecycleObserver {  
    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)  
    fun connectListener() {  
        ...  
    }  
}
```

Add the observer to the lifecycle:

```
myLifecycleOwner.getLifecycle().addObserver(MyObserver())
```

# Example

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        val ob = ObserverExample()  
        lifecycle.addObserver(ob)  
    }  
}
```

```
/*  
 * It's better to use DefaultLifecycleObserver  
 * instead of LifecycleObserver  
 */  
class ObserverExample: DefaultLifecycleObserver {  
  
    override fun onCreate(owner: LifecycleOwner) {  
        Log.d("LifecycleExample", "onCreate()")  
    }  
    override fun onStart(owner: LifecycleOwner) {  
        Log.d("LifecycleExample", "onStart()")  
    }  
    override fun onPause(owner: LifecycleOwner) {  
        Log.d("LifecycleExample", "onPause()")  
    }  
    ...  
}
```

Logcat Logcat × +

Pixel 5 API 31 (emulator-5554) Android 12, API 31

tag=:LifecycleExample

2024-03-12 22:09:44.728	8568-8568	LifecycleExample	com...otlinlifecycleobserverexample	D	onCreate()
2024-03-12 22:09:44.730	8568-8568	LifecycleExample	com...otlinlifecycleobserverexample	D	onStart()
2024-03-12 22:09:44.731	8568-8568	LifecycleExample	com...otlinlifecycleobserverexample	D	onResume()
2024-03-12 22:10:38.443	8568-8568	LifecycleExample	com...otlinlifecycleobserverexample	D	onPause()
2024-03-12 22:10:38.444	8568-8568	LifecycleExample	com...otlinlifecycleobserverexample	D	onStop()
2024-03-12 22:10:38.447	8568-8568	LifecycleExample	com...otlinlifecycleobserverexample	D	onDestroy()

# Tasks and back stack

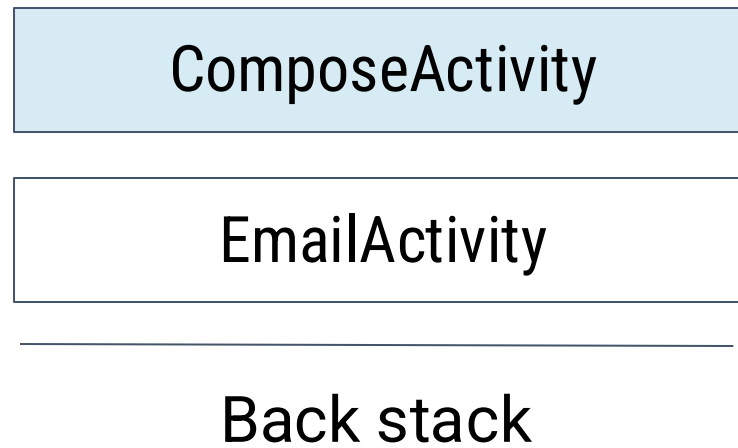
# Back stack of activities

EmailActivity

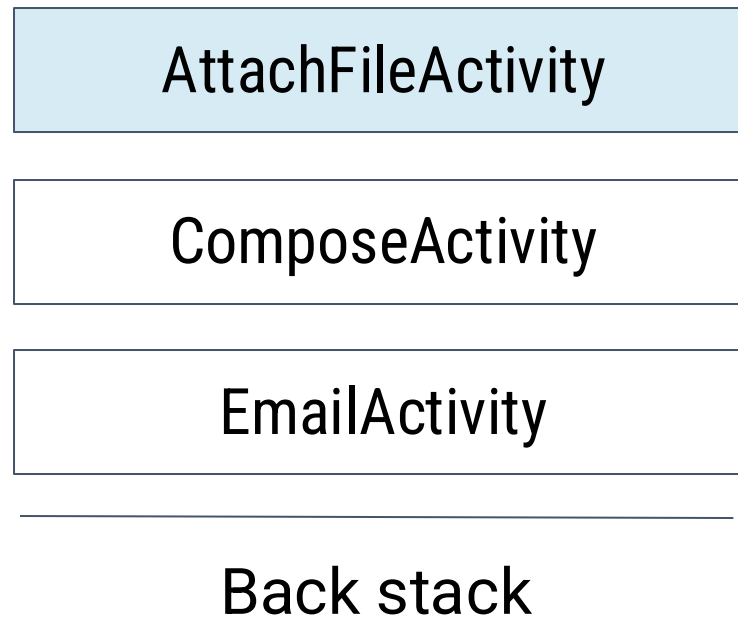
---

Back stack

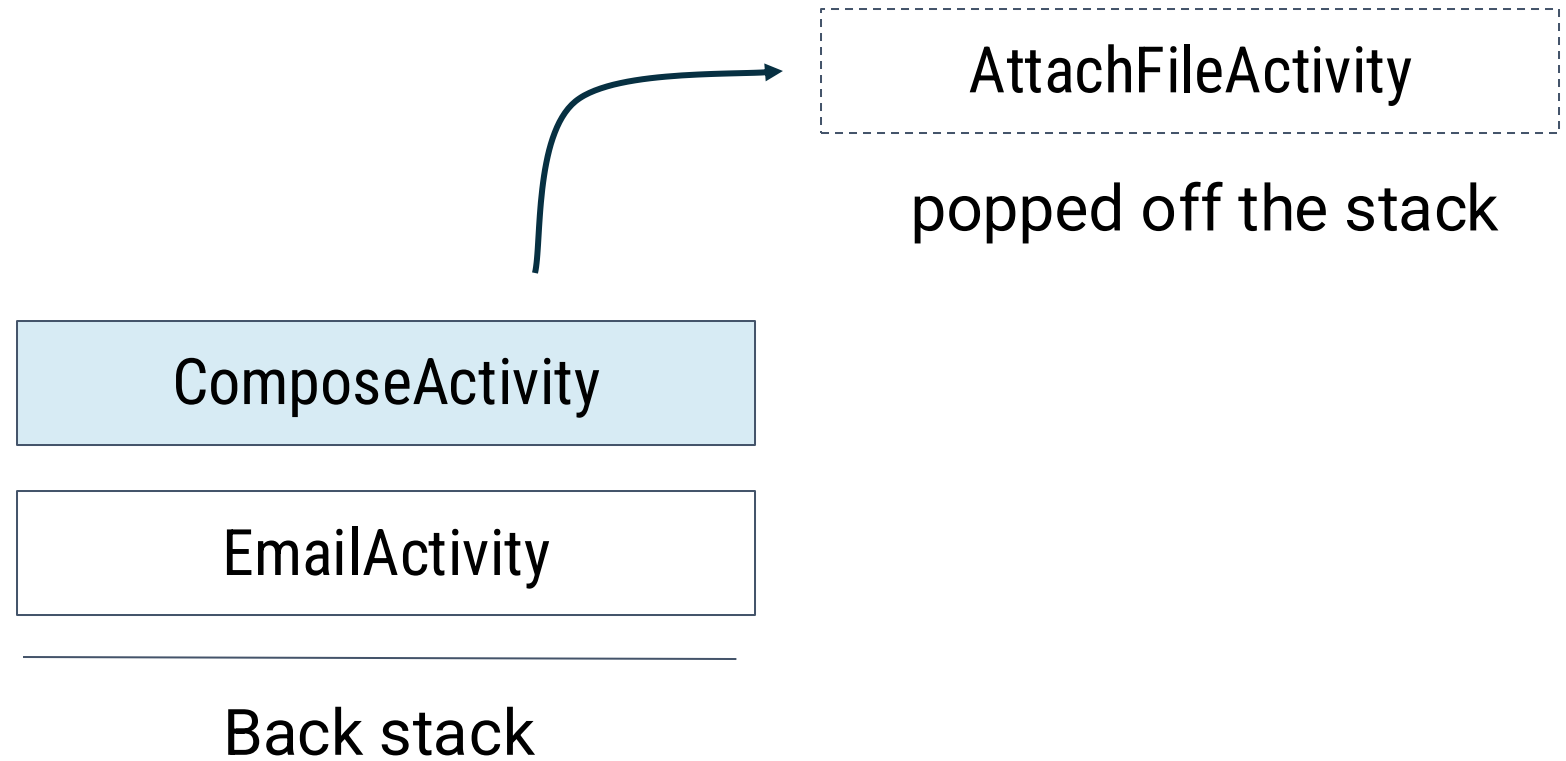
# Add to the back stack



# Add to the back stack again

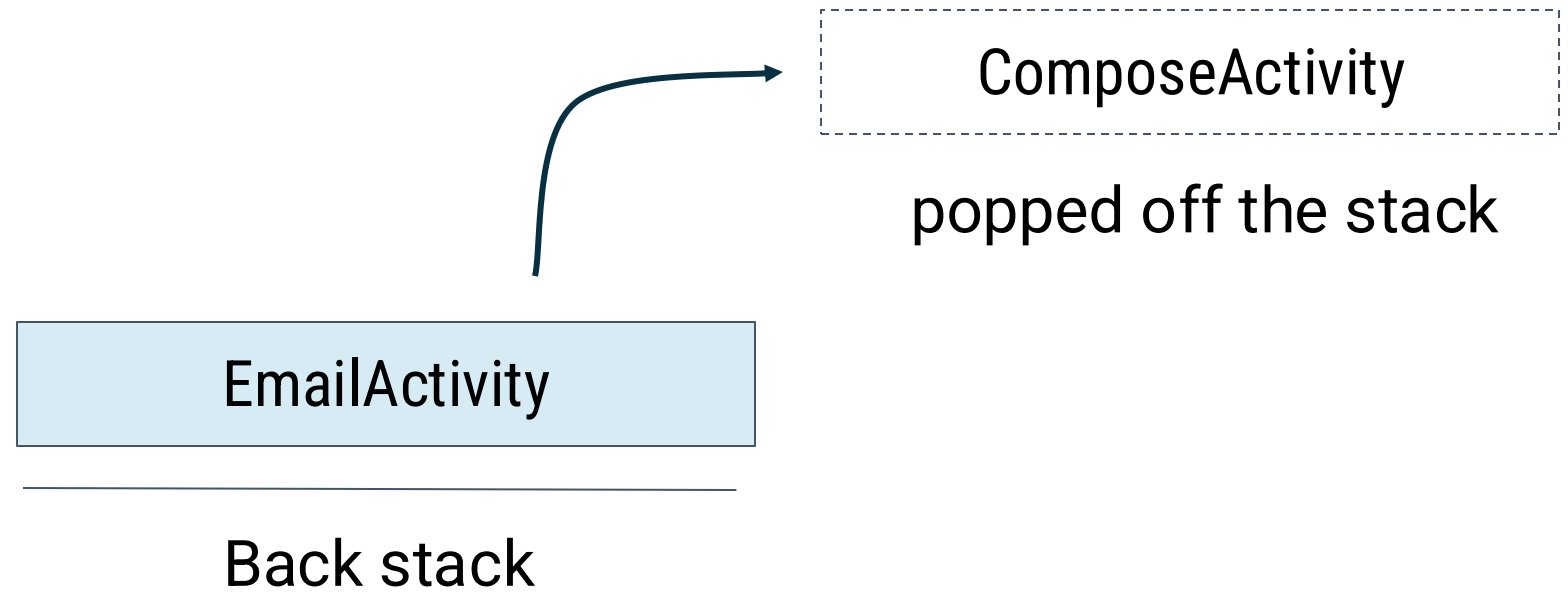


# Tap Back button





# Tap Back button again



# First destination in the back stack

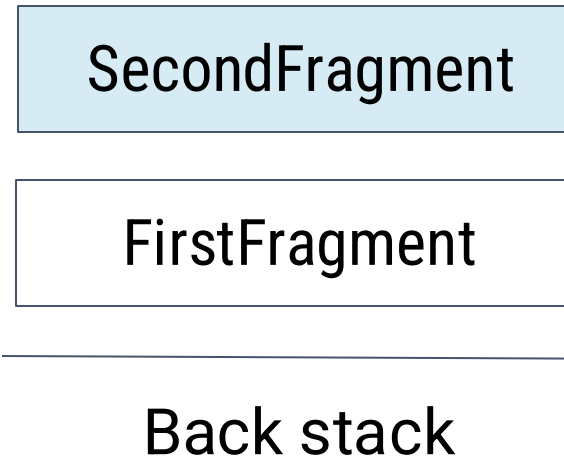


FirstFragment

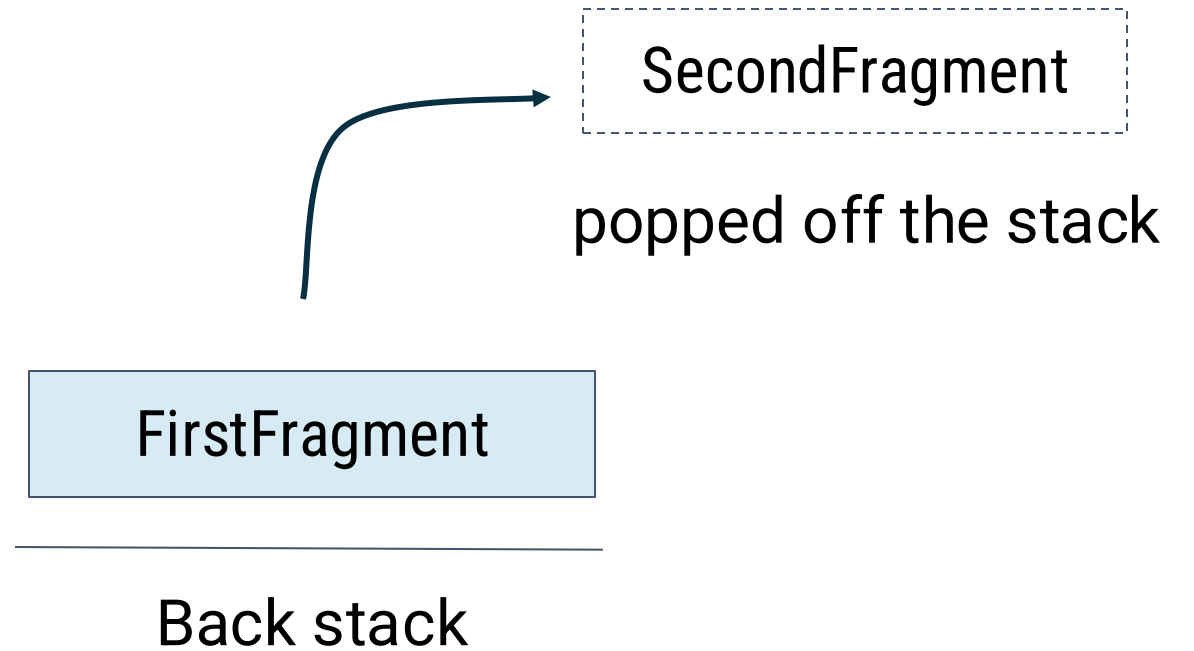
---

Back stack

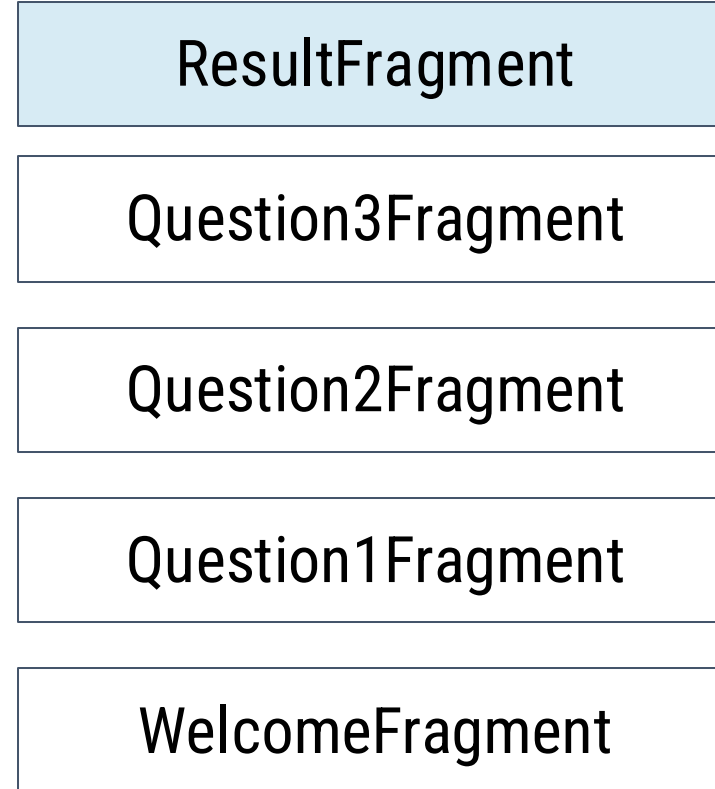
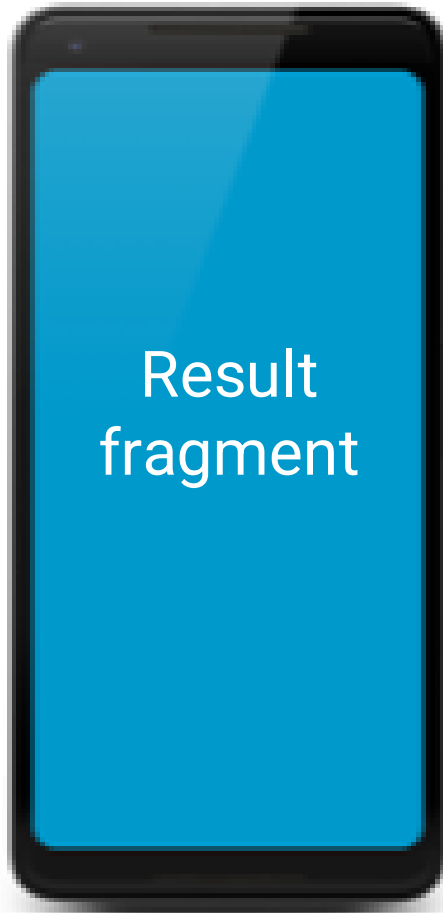
# Add a destination to the back stack



# Tap Back button



# Another back stack example



Back stack

# Modify Back button behavior



pop additional destinations  
off the back stack

