Android Development vol. 2

Tomáš Kypta

@TomasKypta

Agenda

- Service
- Broadcast receivers
- Libraries - Retrofit
- Fragments
- Asynchronous processing

Service

Service

- for long running tasks independent on UI
- also for potentially exposing some functionality to other apps
- extend class `android.app.Service`
- two types:
  - started service
  - bound service

Service

- by default runs on the main thread!

Started Service

- to perform some operation without returning result to the caller

- start by calling `context.startService(Intent)`

- only one instance of the service is created

Bound Service

- for interacting with other components
- to expose functionality to other apps
- client calls **`bindService()`**
  - cannot be called from broadcast receiver

Bound Service

- implement **`onBind()`**

- return **`null`** if you don’t want to allow binding

Bound Service

- clients call `unbindService()`
- when all clients are unbound, the system destroys the service
    - no need to stop service explicitly

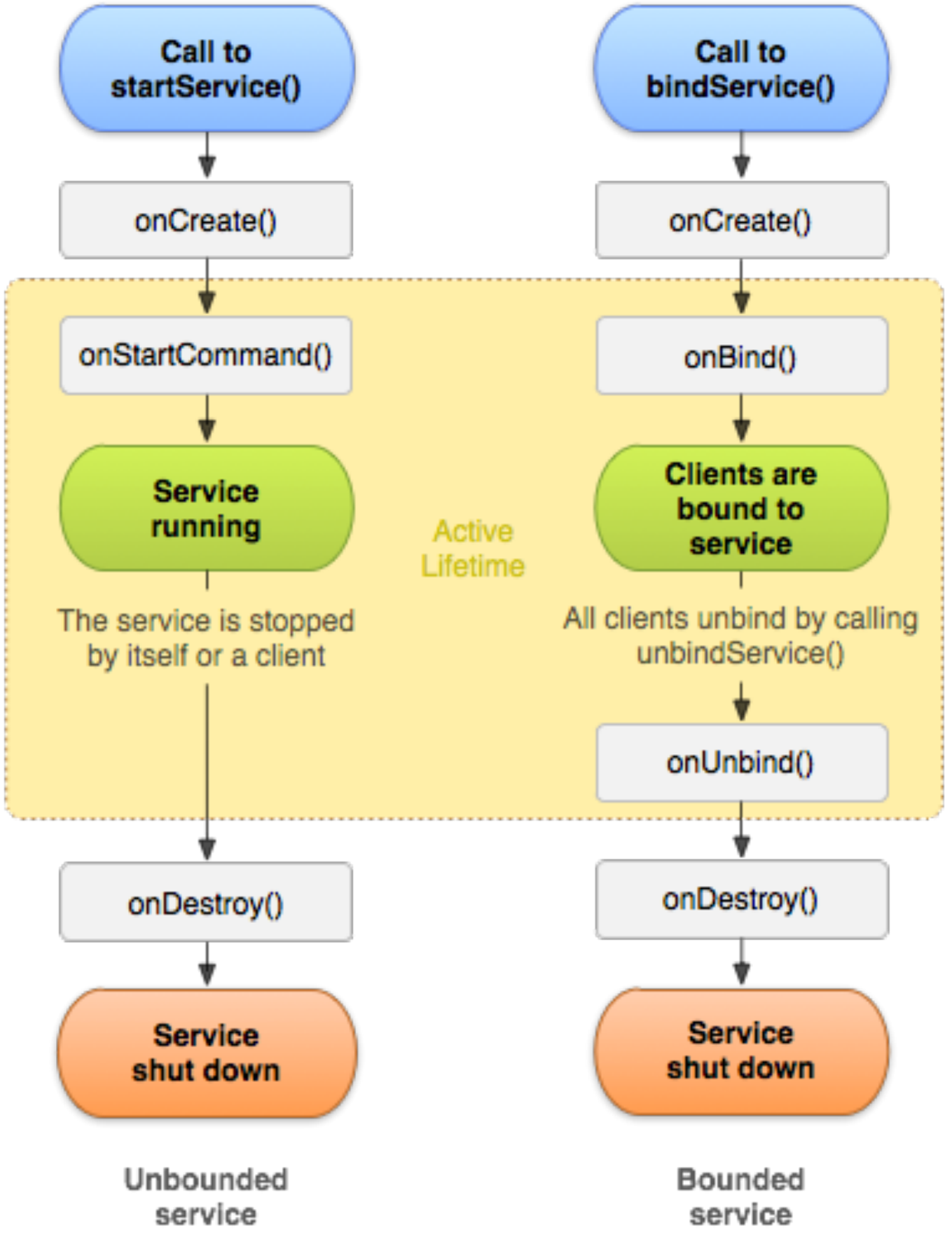
Service

- Started and Bound approaches can be mixed

Service Lifecycle

- service lifetimes:
  - **entire lifetime**
  - **active lifetime**
    - start in onStartCommand() or onBind()

Service Lifecycle



Foreground Service

- something user is actively aware of
- must provide an ongoing notification
  - cannot be dismissed
- makes app a higher priority process
- `startForeground()`
- `stopForeground()`

Intent Service

Intent Service

- service for processing on background threads
- for processing independent on UI
- `android.app.IntentService`

Intent Service

```
public class MyIntentService extends IntentService {

    public MyIntentService() {
        super("MyIntentService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        // run some code on background
    }
}
```

Exercise

1. Download the code: http://github.com/avast/android-lectures

2. Import project in “Lecture 2” to Android studio

3. Run the app on device/emulator

4. Start DownloadService from MainActivity

5. Download User in a background thread in the service, use GitHubApi class for that

Exercise

6. Start DownloadIntentService from MainActivity

7. Download list of repositories in the IntentService, use GitHubApi class for that

Broadcast Receiver

Broadcast Receiver

- responds to broadcasts
- broadcasts are system wide
- receivers can be registered statically or dynamically
- system or custom broadcasts
- examples:
  - incoming SMS, incoming call, screen turned off, low battery, removed SD card, …

Broadcast Receiver

- extend class `android.content.BroadcastReceiver`
- `void onReceive(Context, Intent)` callback
  - called on the main thread
  - don’t do any threading there!!

Broadcast Receiver

- static registration in **`AndroidManifest.xml`**
  - **`<receiver>`** tag inside **`<application>`** tag
- publicly available
  - to make private use **`android:exported="false"`**

Broadcast Receiver

- static registration not possible for all intents
  - only some exceptions - can be found in documentation
  - e.g. ACTION_BATTERY_CHANGED

Broadcast Receiver

- dynamic registration
  - `Context.registerReceiver(BroadcastReceiver, IntentFilter)`
  - `Context.unregisterReceiver(BroadcastReceiver)`

Broadcasts

- normal
  - completely asynchronous
    - undefined order of called receivers
- ordered
  - one receiver at a time
  - `android:priority`

Sending Broadcasts

- `Context.sendBroadcast(Intent)`
  - send to all apps registered for the broadcast
    - can be restricted since ICS with `Intent.setPackage(String)`
- `Context.sendOrderedBroadcast(Intent, String)`

Exercise

8. Create BroadcastReceiver handling ACTION_USER_DOWNLOADED action and register/unregister it in onStart()/onStop()

9. Notify MainActivity about successful User download from DownloadService via broadcast

Local Broadcast

Local Broadcast

```
LocalBroadcastManager lbManager =
        LocalBroadcastManager.getInstance(context);

lbManager.registerReceiver(mReceiver, intentFilter);

lbManager.unregisterReceiver(mReceiver);

lbManager.sendBroadcast(intent);

lbManager.sendBroadcastSync(intent);
```

Local Broadcast

```
LocalBroadcastManager lbManager =
        LocalBroadcastManager.getInstance(context);

lbManager.registerReceiver(mReceiver, intentFilter);

lbManager.unregisterReceiver(mReceiver);

lbManager.sendBroadcast(intent);

lbManager.sendBroadcastSync(intent);
```

Local Broadcast

```
LocalBroadcastManager lbManager =
        LocalBroadcastManager.getInstance(context);

lbManager.registerReceiver(mReceiver, intentFilter);

lbManager.unregisterReceiver(mReceiver);

lbManager.sendBroadcast(intent);

lbManager.sendBroadcastSync(intent);
```

Local Broadcast

```
LocalBroadcastManager lbManager =
        LocalBroadcastManager.getInstance(context);

lbManager.registerReceiver(mReceiver, intentFilter);

lbManager.unregisterReceiver(mReceiver);

lbManager.sendBroadcast(intent);

lbManager.sendBroadcastSync(intent);
```

Exercise

10. Create BroadcastReceiver handling ACTION_REPOS_DOWNLOADED action and register/unregister it in onStart()/onStop() via local broadcast

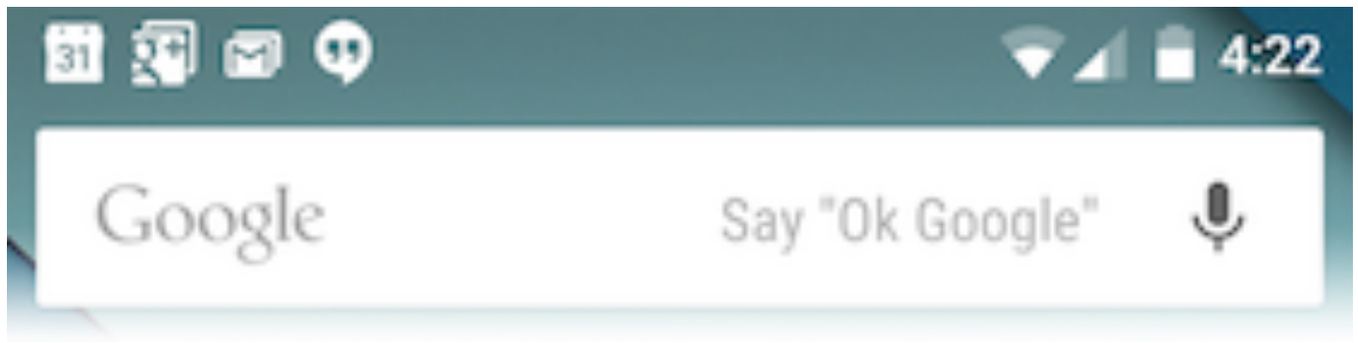
11. Notify MainActivity about successful User download from DownloadIntentService via local broadcast

Notification

Notification

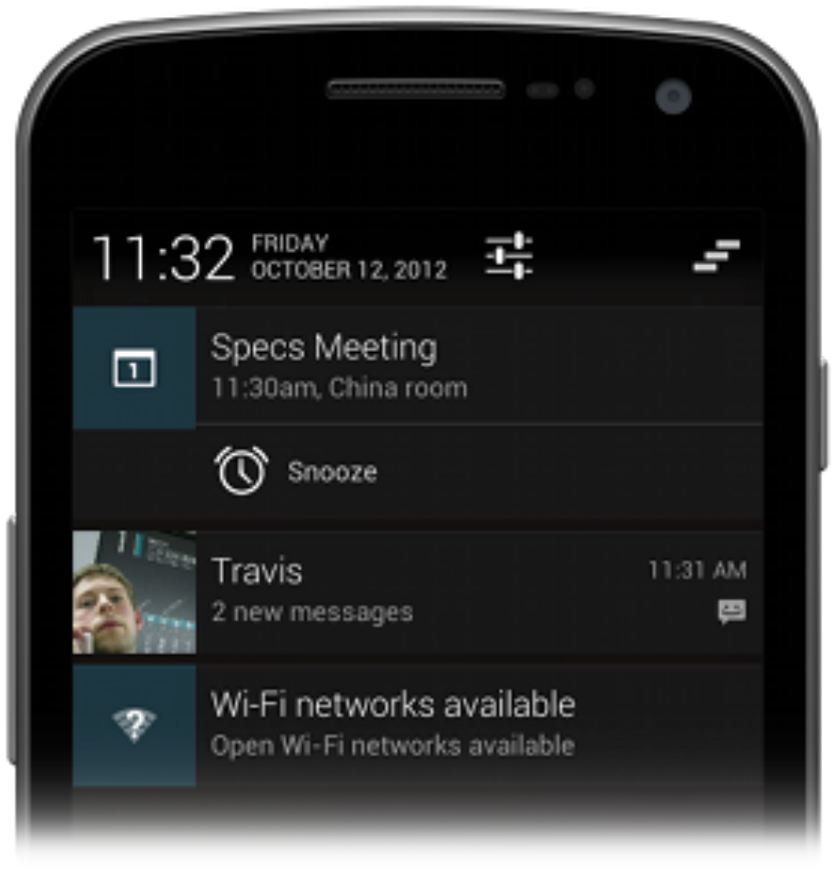
- a message that can be displayed to the user outside normal UI

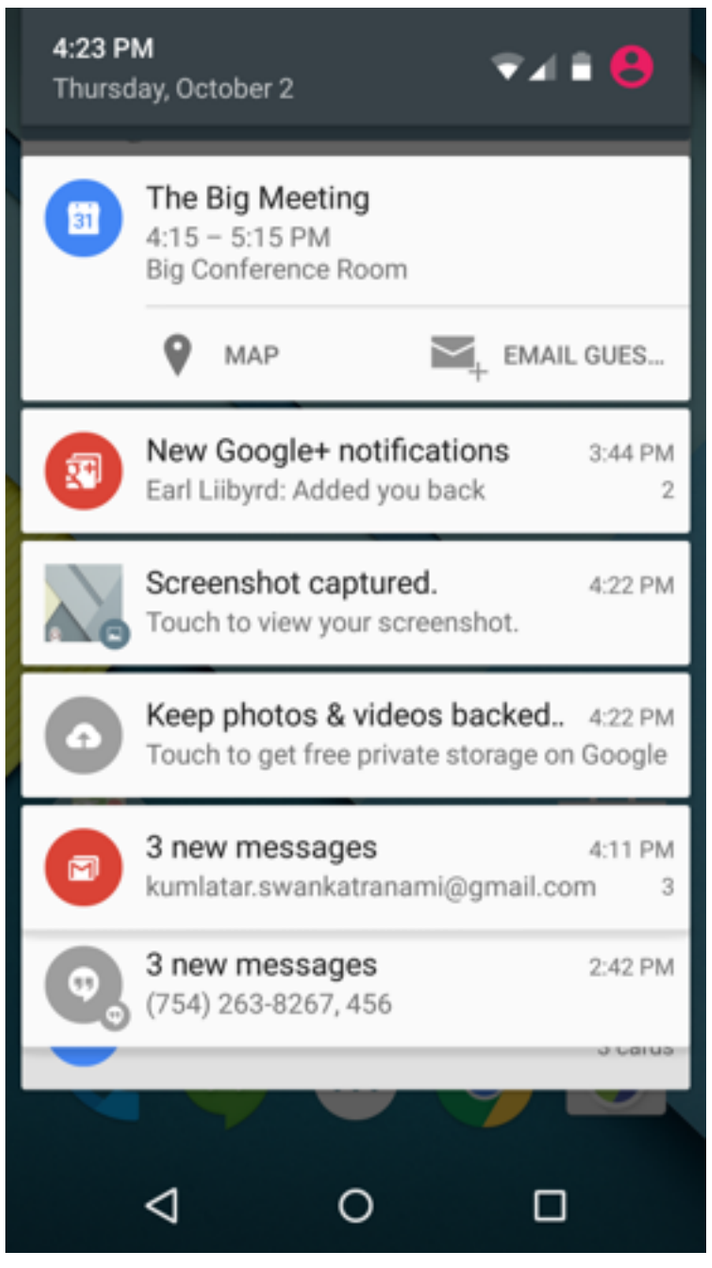
- displayed in notification area



Notification

- user can open notification drawer to see the details
- app can define UI and click action





Notification

- use `NotificationCompat.Builder`

Notification

```
NotificationCompat.Builder mBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!");
```

Notification

```
NotificationCompat.Builder mBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!");
```

Notification

```
Intent resultIntent = new Intent(this, MyActivity1.class);

TaskStackBuilder stackBuilder =
TaskStackBuilder.create(this);

stackBuilder.addParentStack(MyActivity1.class);
stackBuilder.addNextIntent(resultIntent);
```

Notification

```
Intent resultIntent = new Intent(this, MyActivity1.class);

TaskStackBuilder stackBuilder =
TaskStackBuilder.create(this);

stackBuilder.addParentStack(MyActivity1.class);
stackBuilder.addNextIntent(resultIntent);
```

Notification

```
Intent resultIntent = new Intent(this, MyActivity1.class);

TaskStackBuilder stackBuilder =
TaskStackBuilder.create(this);

stackBuilder.addParentStack(MyActivity1.class);
stackBuilder.addNextIntent(resultIntent);
```

Notification

```
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );
mBuilder.setContentIntent(resultPendingIntent);

NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(
        Context.NOTIFICATION_SERVICE);

mNotificationManager.notify(MY_ID, mBuilder.build());
```

Notification

```
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );
mBuilder.setContentIntent(resultPendingIntent);

NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(
        Context.NOTIFICATION_SERVICE);

mNotificationManager.notify(MY_ID, mBuilder.build());
```

Notification

```
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );
mBuilder.setContentIntent(resultPendingIntent);

NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(
        Context.NOTIFICATION_SERVICE);

mNotificationManager.notify(MY_ID, mBuilder.build());
```

Notification

- update notification by using the same ID

Exercise

12. Create notification showing repos count when we are notified about downloaded data.

13. Create click action for the notification that will open ReposActivity

14. Create back stack in the click action containing MainActivity

Application

Application

- application singleton
- `onCreate()` lifecycle callback
  - for app initialisations

Libraries

Retrofit

- for simple declaration of RESTful API
- create an interface with annotations
- create an instance fo the interface via RestAdapter

Retrofit

- support for many formats via converters
  - json, xml, protocol buffers, …
- supports sync and async download
  - and more …

Exercise

15.Create Retrofit API method for getting repository contributors

- https://developer.github.com/v3/repos/#list-contributors

Fragment

Fragment

- represents a behavior or a portion of user interface in an activity

- multiple fragments can be combined in a single activity

Fragment

Activity & Fragment

- add in the layout

```
<fragment android:name="com.example.MyListFragment"
          android:id="@+id/list"
          android:layout_width="100dp"
          android:layout_height="match_parent" />
```

Activity & Fragment

- add programmatically

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction transaction =
        fragmentManager.beginTransaction();

ExampleFragment fragment = new ExampleFragment();
transaction.add(R.id.fragment_container, fragment);
transaction.commit();
```

Activity & Fragment

- add programmatically

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction transaction =
        fragmentManager.beginTransaction();

ExampleFragment fragment = new ExampleFragment();
transaction.add(R.id.fragment_container, fragment);
transaction.commit();
```

Activity & Fragment

- add programmatically

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction transaction =
        fragmentManager.beginTransaction();

ExampleFragment fragment = new ExampleFragment();
transaction.add(R.id.fragment_container, fragment);
transaction.commit();
```

Activity & Fragment

- add programmatically

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction transaction =
        fragmentManager.beginTransaction();

ExampleFragment fragment = new ExampleFragment();
transaction.add(R.id.fragment_container, fragment);
transaction.commit();
```

Activity & Fragment

- add programmatically

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction transaction =
        fragmentManager.beginTransaction();

ExampleFragment fragment = new ExampleFragment();
transaction.add(R.id.fragment_container, fragment);
transaction.commit();
```

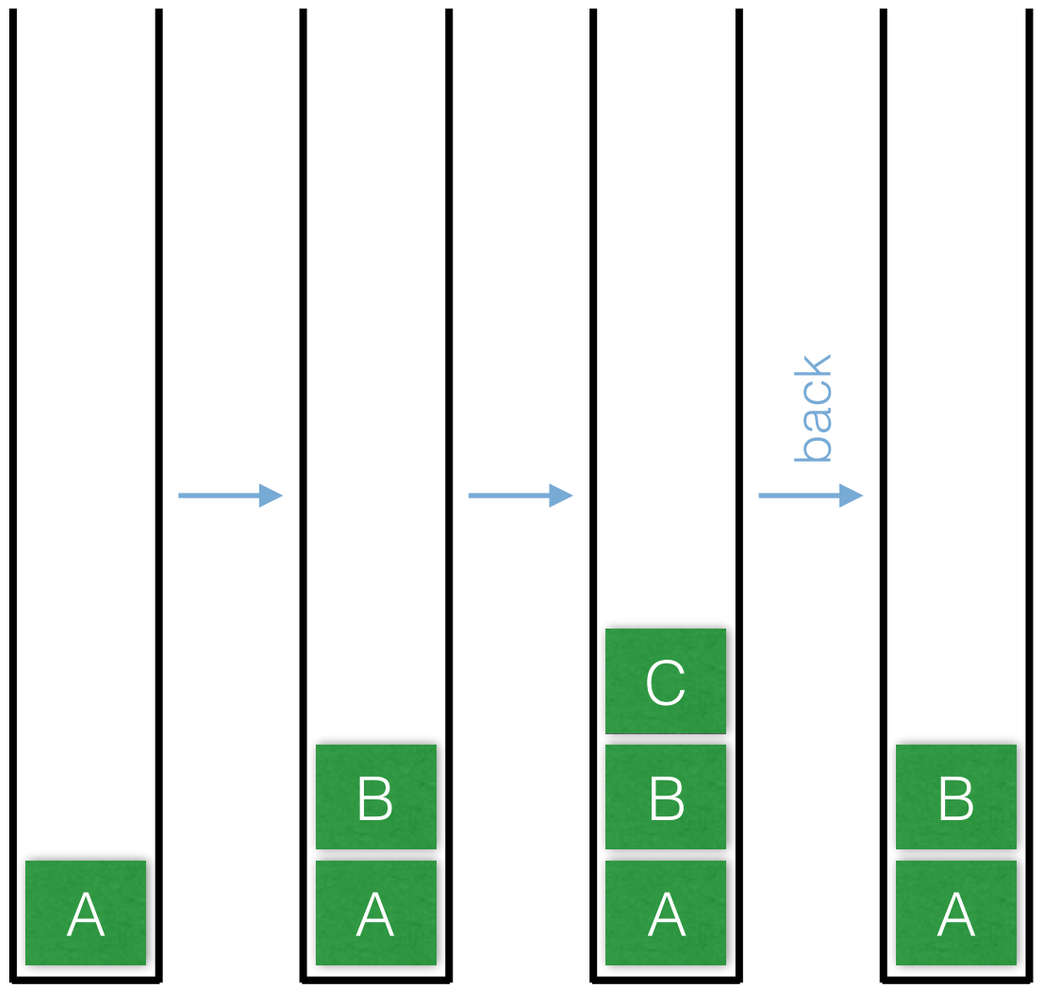
Fragment Back Stack

- fragments can be kept in a stack

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction transaction =
        fragmentManager.beginTransaction();

ExampleFragment fragment = new ExampleFragment();
transaction.add(R.id.fragment_container, fragment);
transaction.addToBackStack(null);
transaction.commit();
```

Fragment Back Stacks

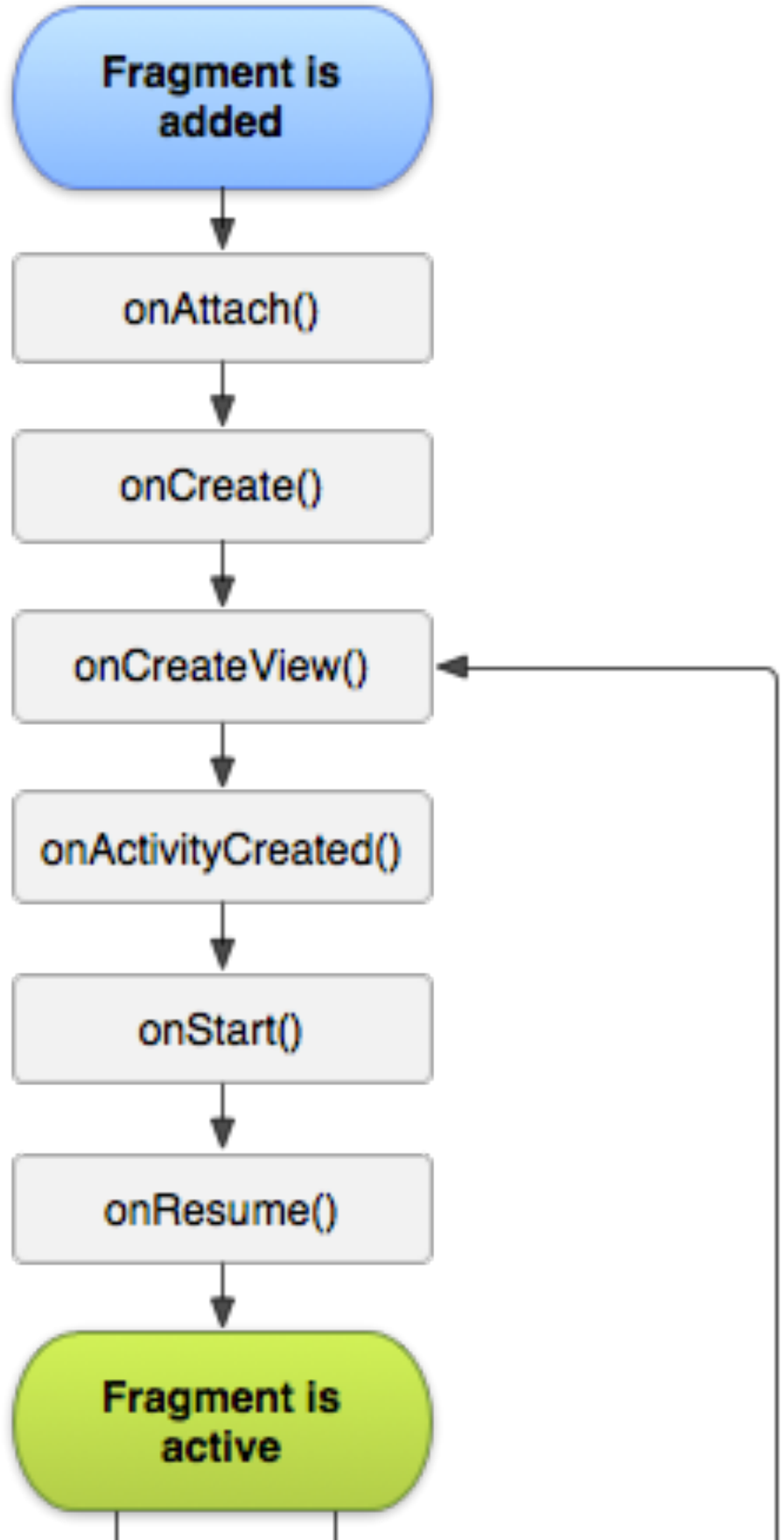


Fragment Lifecycle

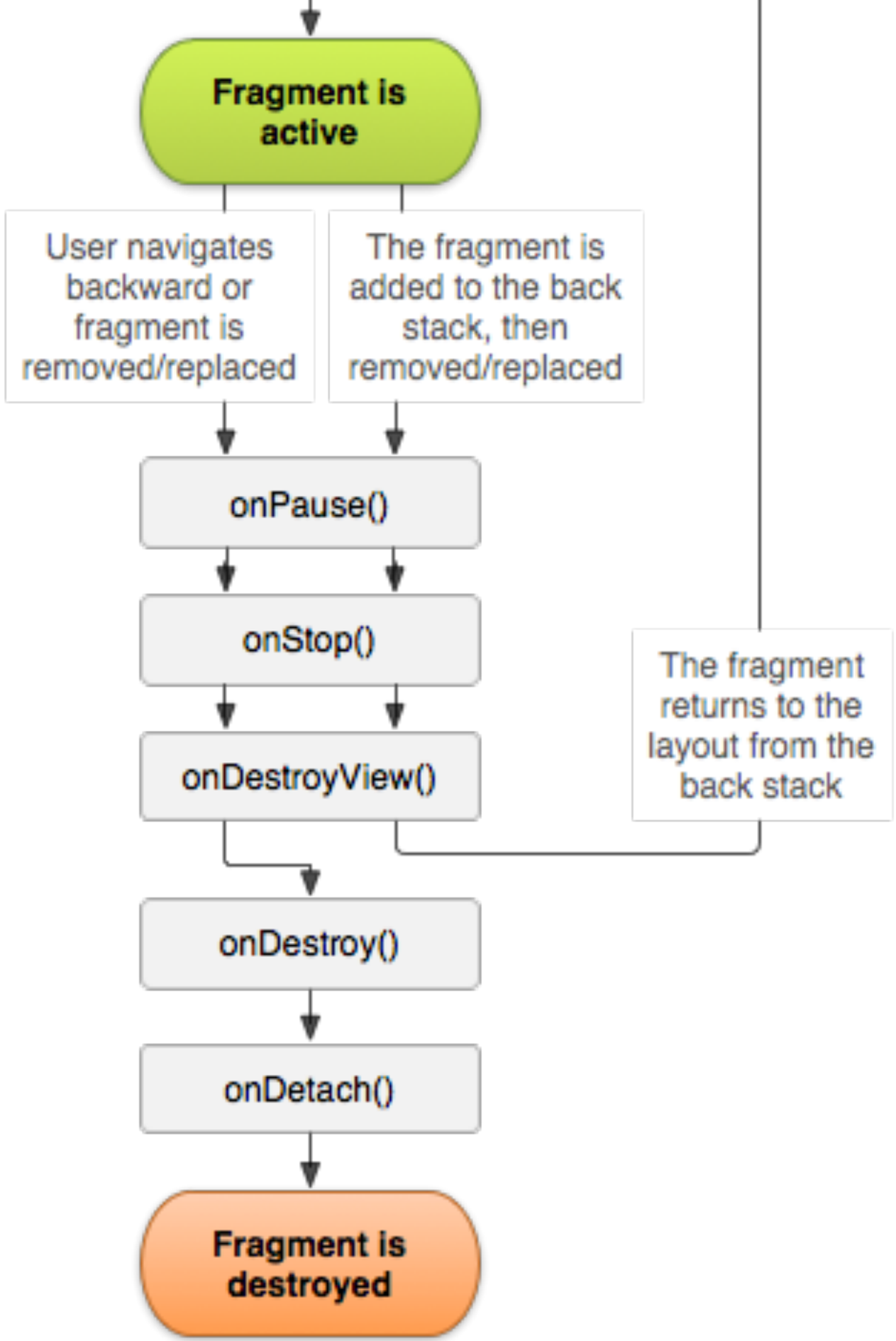
Fragment Lifecycle

- a bit more complicated than activity lifecycle

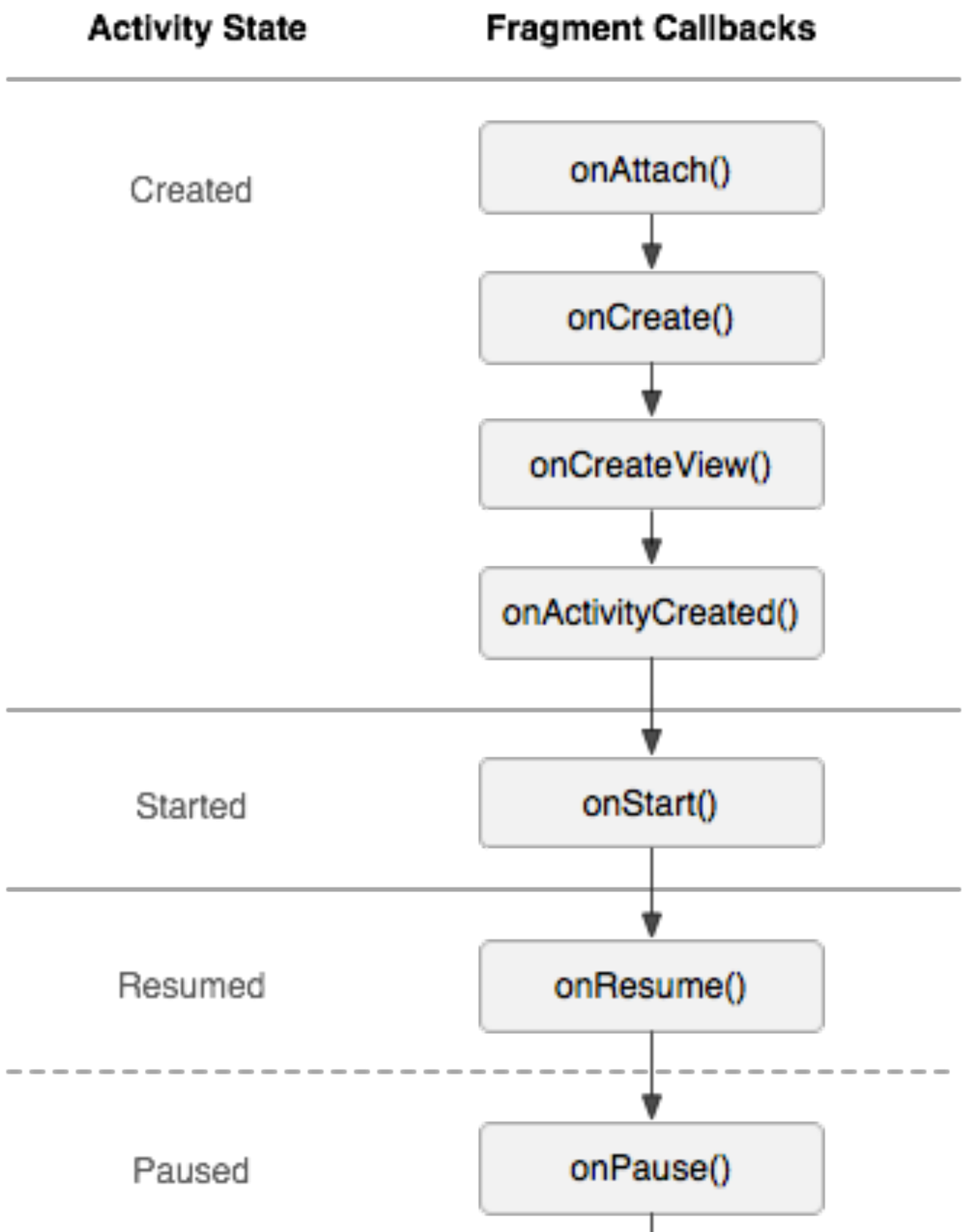
Fragment Lifecycle



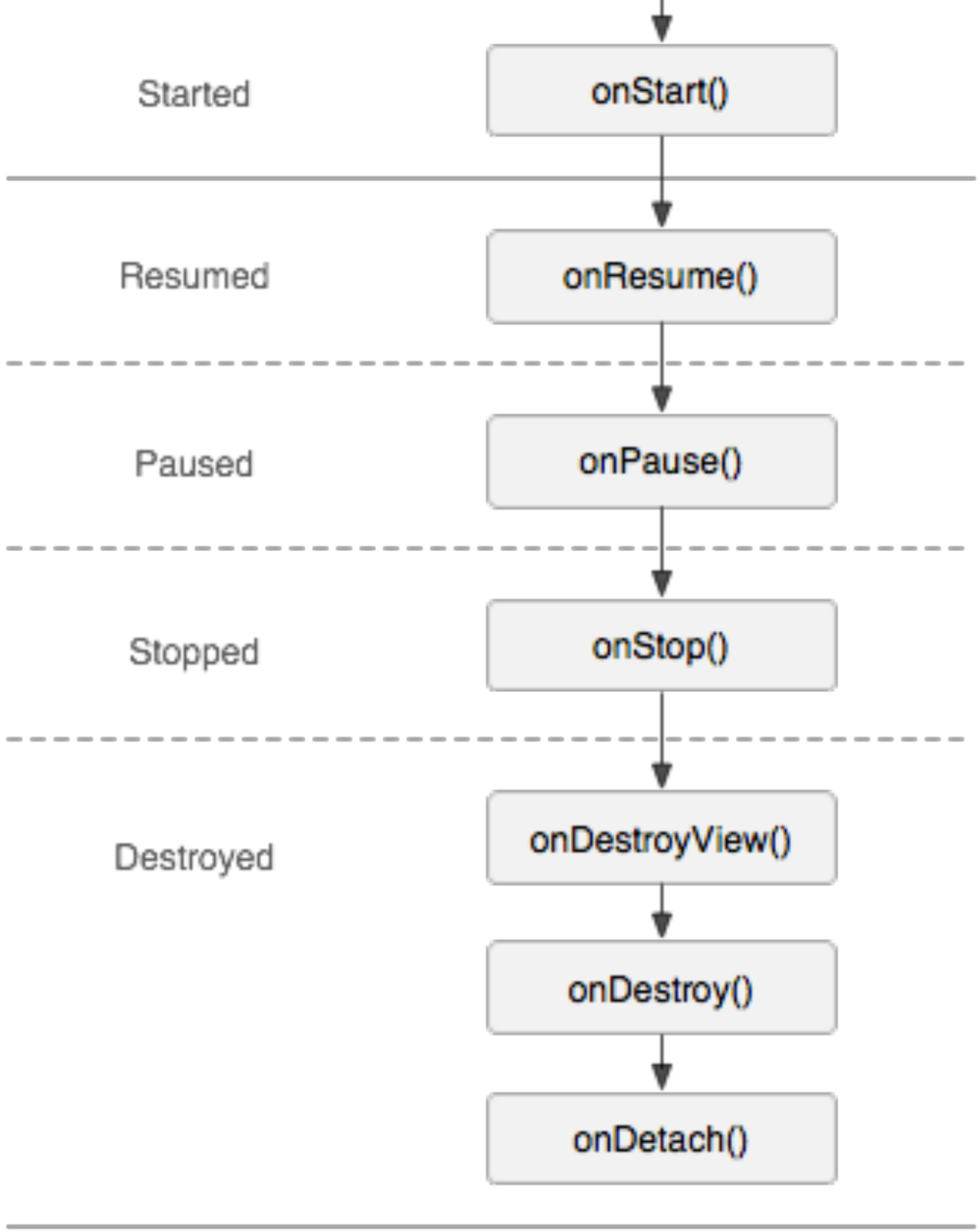
Fragment Lifecycle



Activity & Fragment Lifecycle



Activity & Fragment Lifecycle



Fragment Lifecycle Callbacks

`onAttach()`

- fragments associated with the activity

`onCreateView()`

- create fragment's view hierarchy here

`onActivityCreated()`

- activity's onCreate() method has returned

Fragment Lifecycle Callbacks

`onDestroyView()`

- view hierarchy is being removed

`onDetach()`

- fragment is being disassociated from the activity

Fragment without a UI

- aka worker fragment

```
transaction.add(workFragment, “work”);
```

Exercise

16.Create Activity with dynamically added Fragment

Background Processing

Threads

- main thread = UI thread
- never block the UI thread!!!
- use worker threads for time consuming operations
  - networking, db, filesystem, …
- UI toolkit is not thread safe
  - never manipulate UI from a worker thread!!!

Threads

- complications
- activities are restarted
- memory leaks
- crashes

Background Processing

- Thread
- AsyncTask
- IntentService
- Loader
- ThreadPoolExecutor
- AbstractThreadedSyncAdapter

Background Processing

Some people, when confronted with a problem, think, "I know, I'll use threads," and then two they hav erpoblesms.

HandlerThread

HandlerThread

- holds a queue of tasks
- other threads can push tasks to it
- the thread processes its queue, one task after another
- when the queue is empty, it blocks until something appears

Looper and Handler

- `Looper`
  - class that runs a message loop for a thread
- **`Handler`**
  - provides interaction with the message loop

Looper and Handler

- `sendMessage(Message)`
  - `Message` object, retrieve with `Message.obtain()`
- `post(Runnable)`
- delayed versions, at time versions

Looper and Handler

- UI thread has **`Looper`**
- you can create easily another **`Handler`**
  - **`Handler`** is bound to the **`Looper`** of the current thread
  - or you can explicitly provide different **`Looper`**

Loader

Loader

- asynchronous loading of data
- bound to activity or fragment
- monitor source of data, deliver changes
- reconnect after config change, don't need to requery
- managed by `LoaderManager`

Loader

- `AsyncTaskLoader`

- `CursorLoader`

Loader

- you have to implement

  `LoaderManager.LoaderCallbacks`

Loader

```
@Override
public Loader<D> onCreateLoader(int id, Bundle args) {
    // instantiate a new loader
    return null;
}

@Override
public void onLoadFinished(Loader<D> loader, D data) {
    // called when loader finished its loading
}

@Override
public void onLoaderReset(Loader<D> loader) {
    // called when loader is being reset
}
```

Loader

```
@Override
public Loader<D> onCreateLoader(int id, Bundle args) {
    // instantiate a new loader
    return null;
}
```

```
@Override
public void onLoadFinished(Loader<D> loader, D data) {
    // called when loader finished its loading
}

@Override
public void onLoaderReset(Loader<D> loader) {
    // called when loader is being reset
}
```

Loader

```
@Override
public Loader<D> onCreateLoader(int id, Bundle args) {
    // instantiate a new loader
    return null;
}
```

```
@Override
public void onLoadFinished(Loader<D> loader, D data) {
    // called when loader finished its loading
}

@Override
public void onLoaderReset(Loader<D> loader) {
    // called when loader is being reset
}
```

Loader

```
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    return new CursorLoader(getActivity(), mUri,
        mProjection, "value > ?",
        new String[]{String.valueOf(5)}, "value ASC”);
}
```

Loader

```
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor
cursor) {
    mAdapter.swapCursor(cursor);
}

@Override
public void onLoaderReset(Loader<Cursor> loader) {
    mAdapter.swapCursor(null);
}
```

Loader

```
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor
cursor) {
    mAdapter.swapCursor(cursor);
}

@Override
public void onLoaderReset(Loader<Cursor> loader) {
    mAdapter.swapCursor(null);
}
```

Loader

```
// prepare the loader
// either re-connect with an existing one,
// or start a new one.

getLoaderManager().initLoader(0, null, this);
```

Loader

```
// restart the loader to do a new query

getLoaderManager().restartLoader(0, null, this);
```

Exercise

17. Load repository contributors via loader in the Fragment and display the data

18. Create statically defined receiver that will show toast when airplane mode is enabled/disabled - action android.intent.action.AIRPLANE_MODE

THE END