Android Development vol. 2

Tomáš Kypta

@TomasKypta

Agenda

- Service
- Broadcast receivers
- Fragments
- Asynchronous processing

Service

Service

- for running tasks independent on UI
  - long running tasks
    - background processing
- can potentially expose functionality to other apps
- extend class **`android.app.Service`**

Service

- types:
  - started service
  - bound service
- visibility:
  - background
    - greatly limited in Oreo
  - forground

Service

- **not** a separate process
  - however it can be specified to run in a separate process
- **not** a thread
- it runs on the main thread by default!

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 a 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

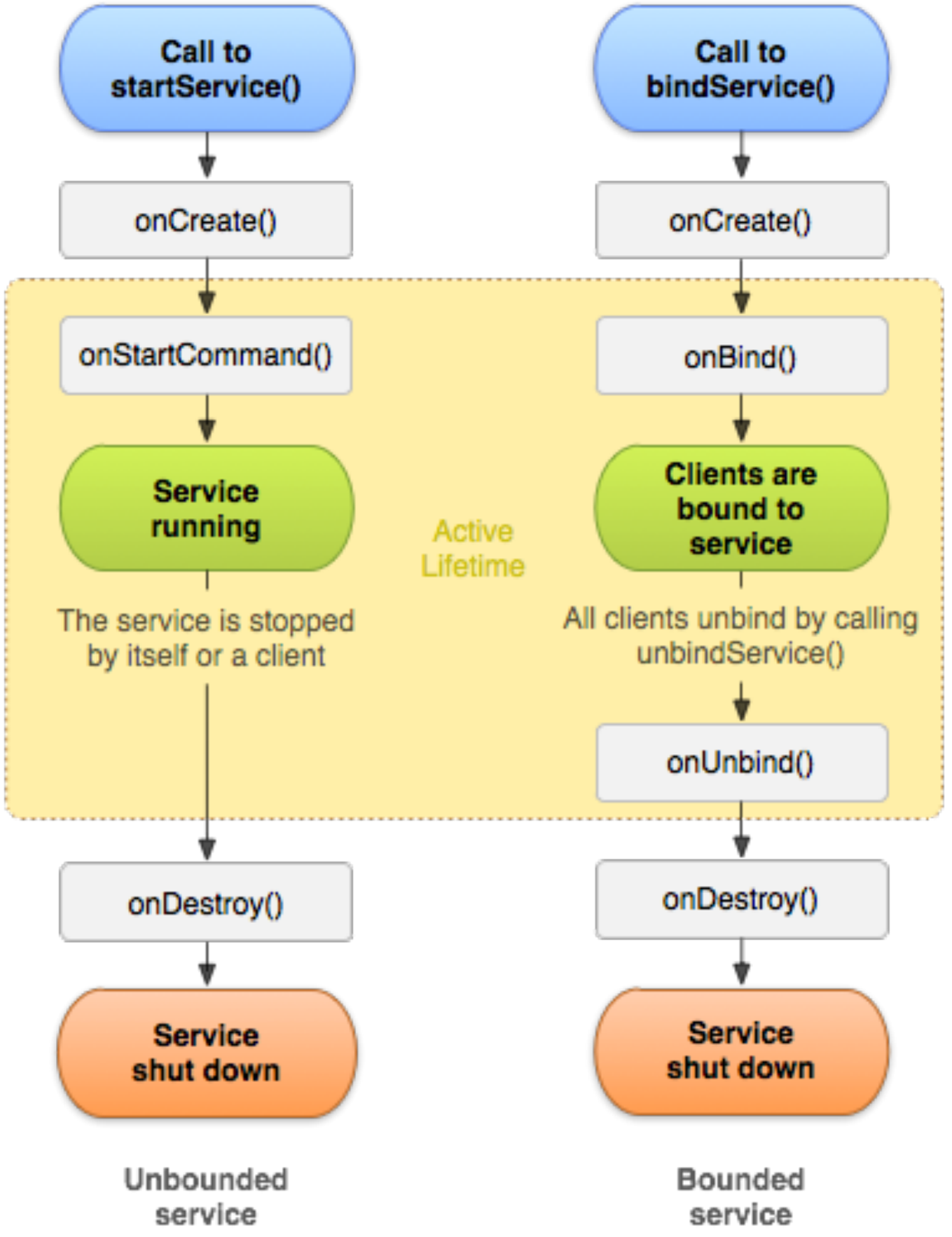
Service

- 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 the 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
    }
}
```

Service since Oreo

Service since Oreo

- limitation for apps running in the background
- by default for apps targeting Oreo (or newer)
- users can enable for any app from the settings

Service since Oreo

- `JobScheduler`
  - system schedules background jobs
  - otherwise only a short window for bg services after leaving foreground
- `JobIntentService` in the support lib instead of `IntentService`
  - uses jobs
  - needs `WAKE_LOCK` permission

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
- …
- libraries - RxJava, …

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);
```

Broadcast Receiver

Broadcast Receiver

- responds to broadcasts
- broadcasts are system wide messages
- registration
  - static - in the manifest
  - dynamic - in the code at runtime

Broadcast Receiver

- source
  - system
    - our app
- examples:
  - incoming SMS, incoming call, screen turned off, low battery, removed SD card, …

Broadcast Receiver

- type
  - implicit
    - system-wide messages
      - ACTION_PACKAGE_REPLACED
      - CONNECTIVITY_ACTION
- explicit
  - specific target
  - ACTION_MY_PACKAGE_REPLACED

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

- 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)`

Broadcasts since Oreo

Broadcasts since Oreo

- apps cannot register for implicit broadcasts statically

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);
```

Fragment

Fragment

- represents a behavior of 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();
```

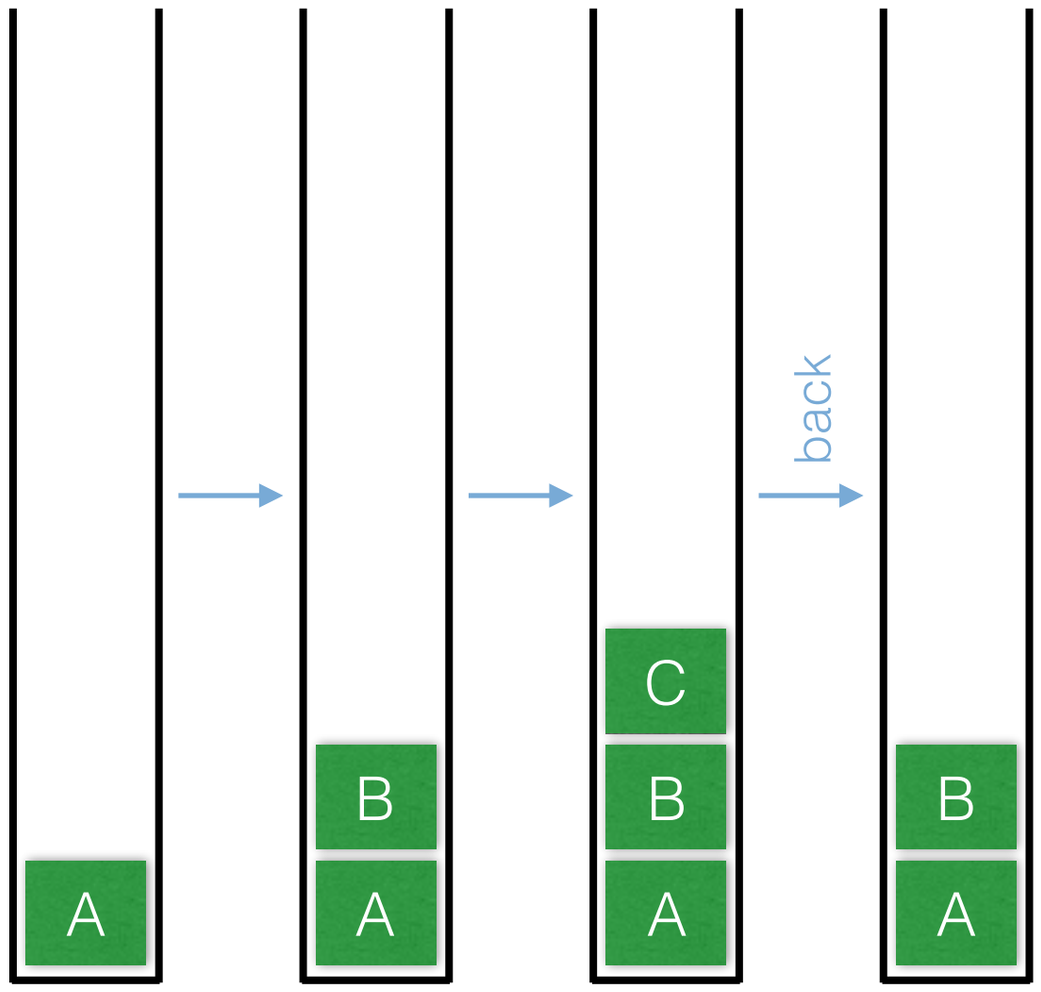
Fragment Back Stack

- 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

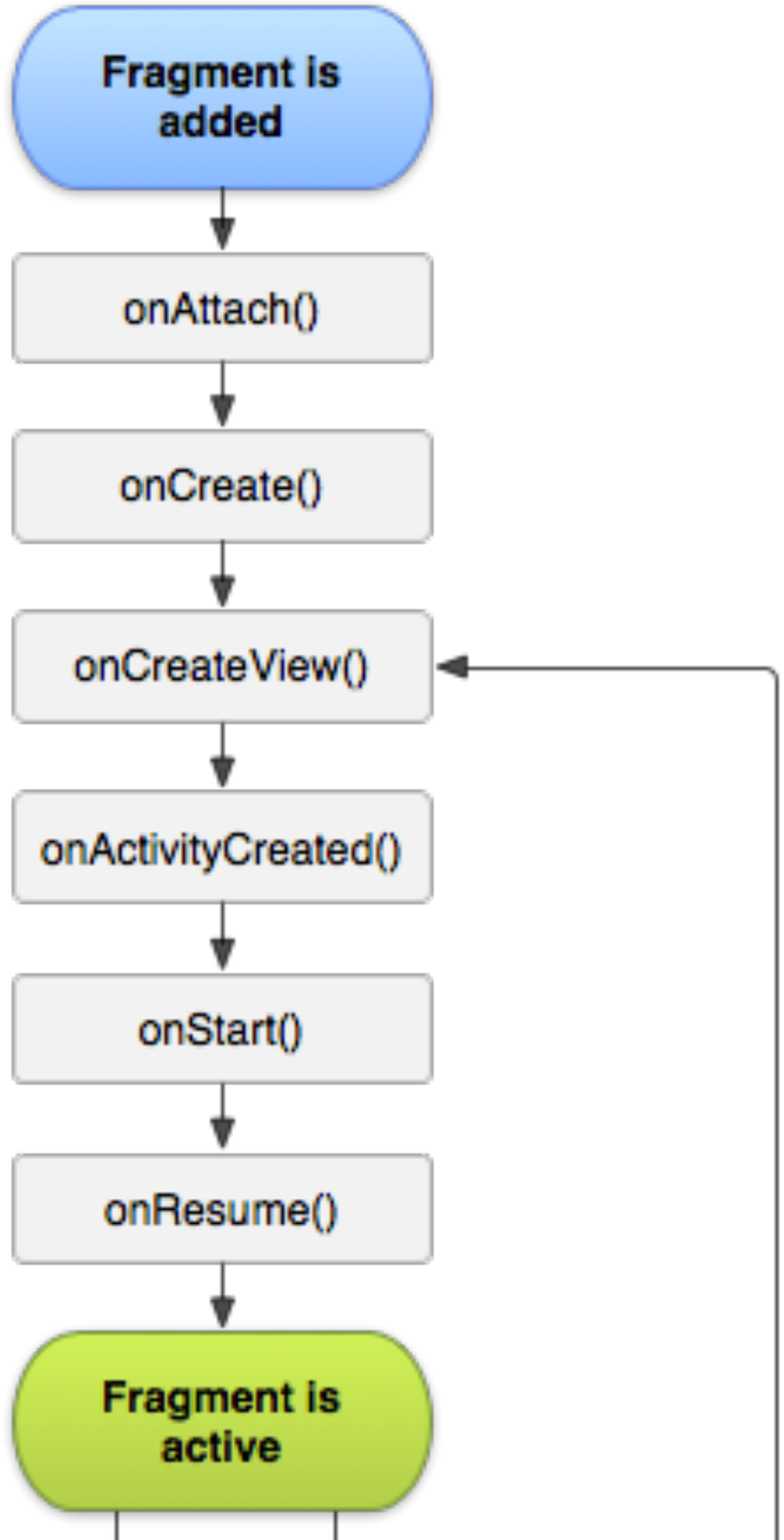


Fragment Lifecycle

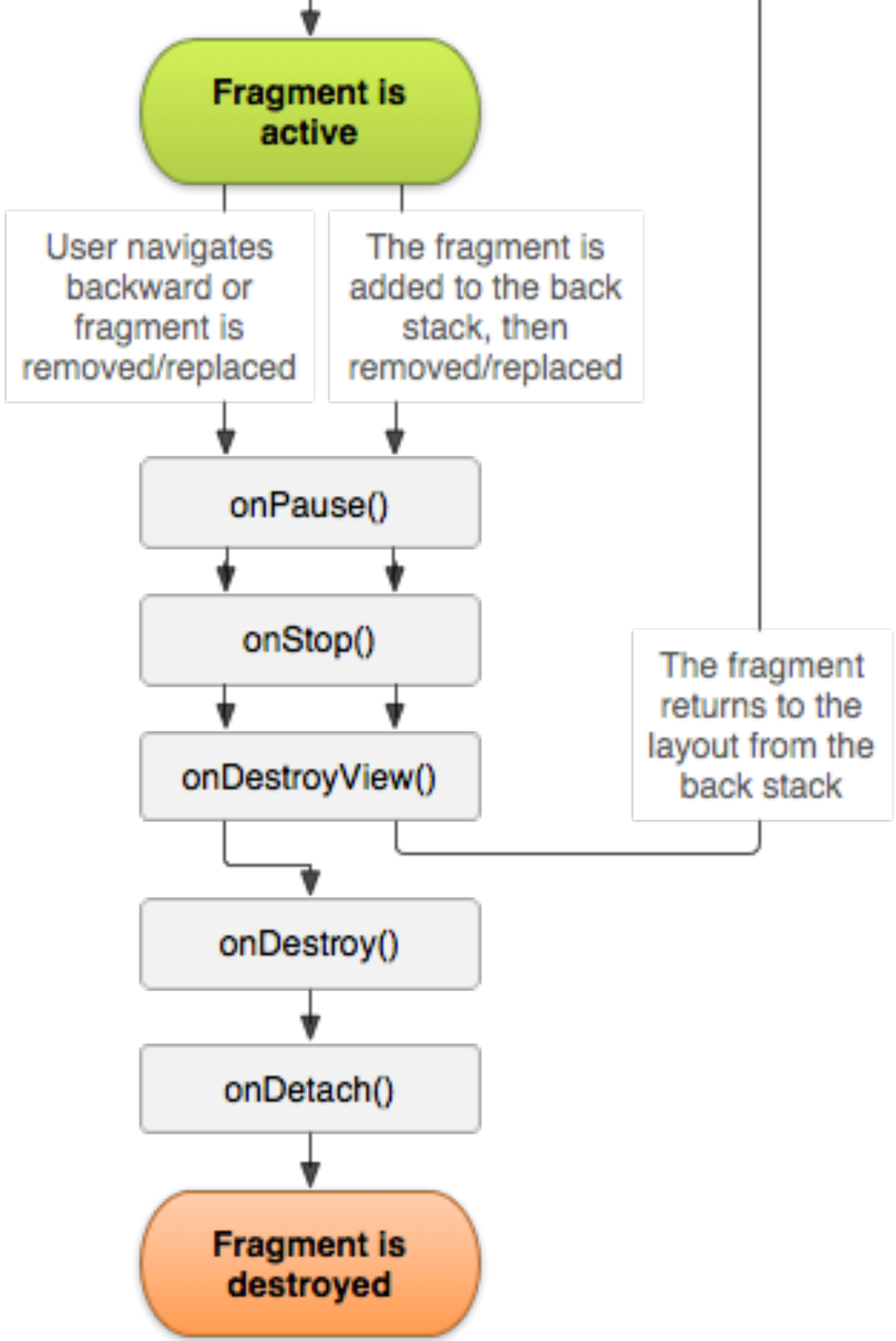
Fragment Lifecycle

- a bit more complicated than activity lifecycle

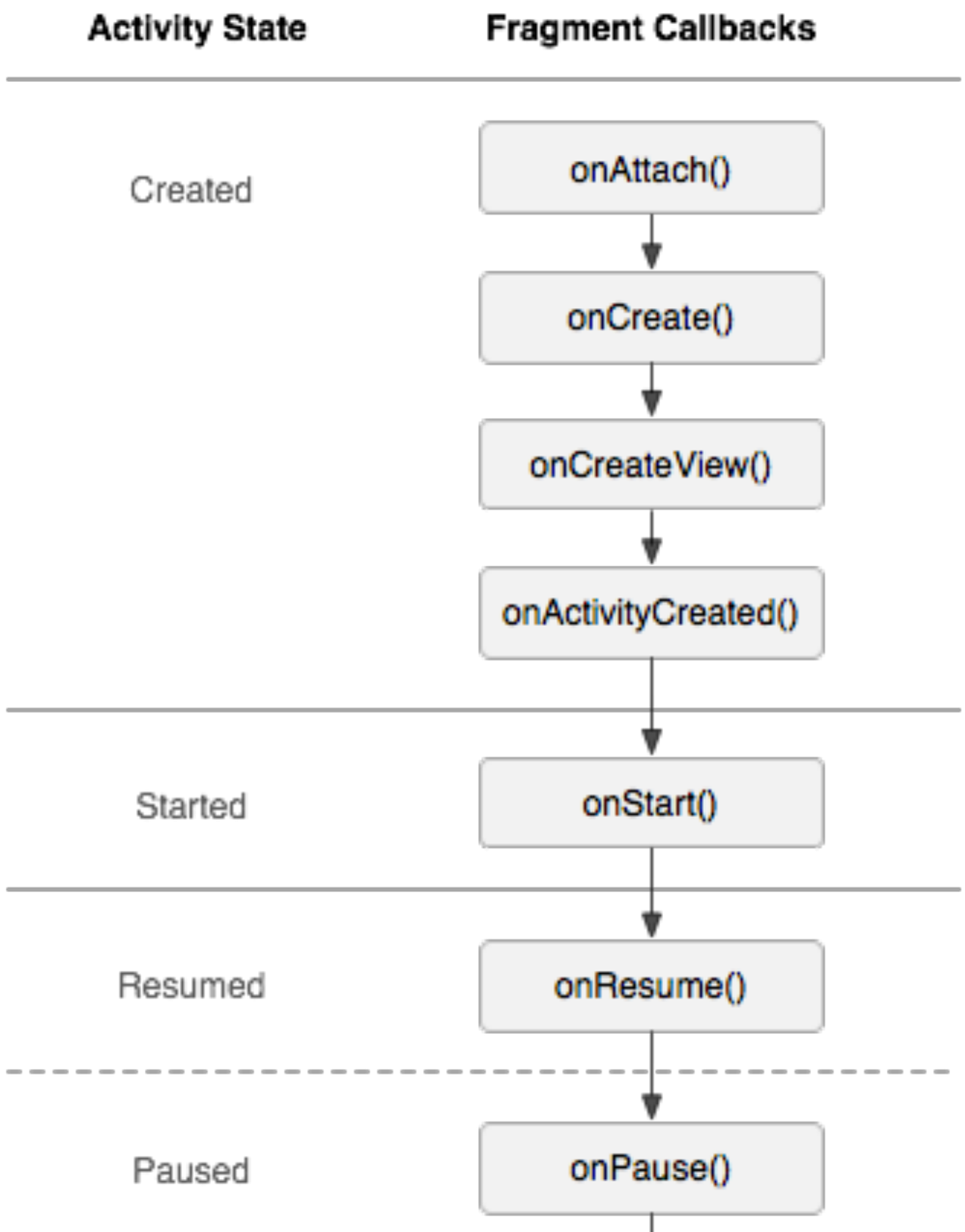
Fragment Lifecycle



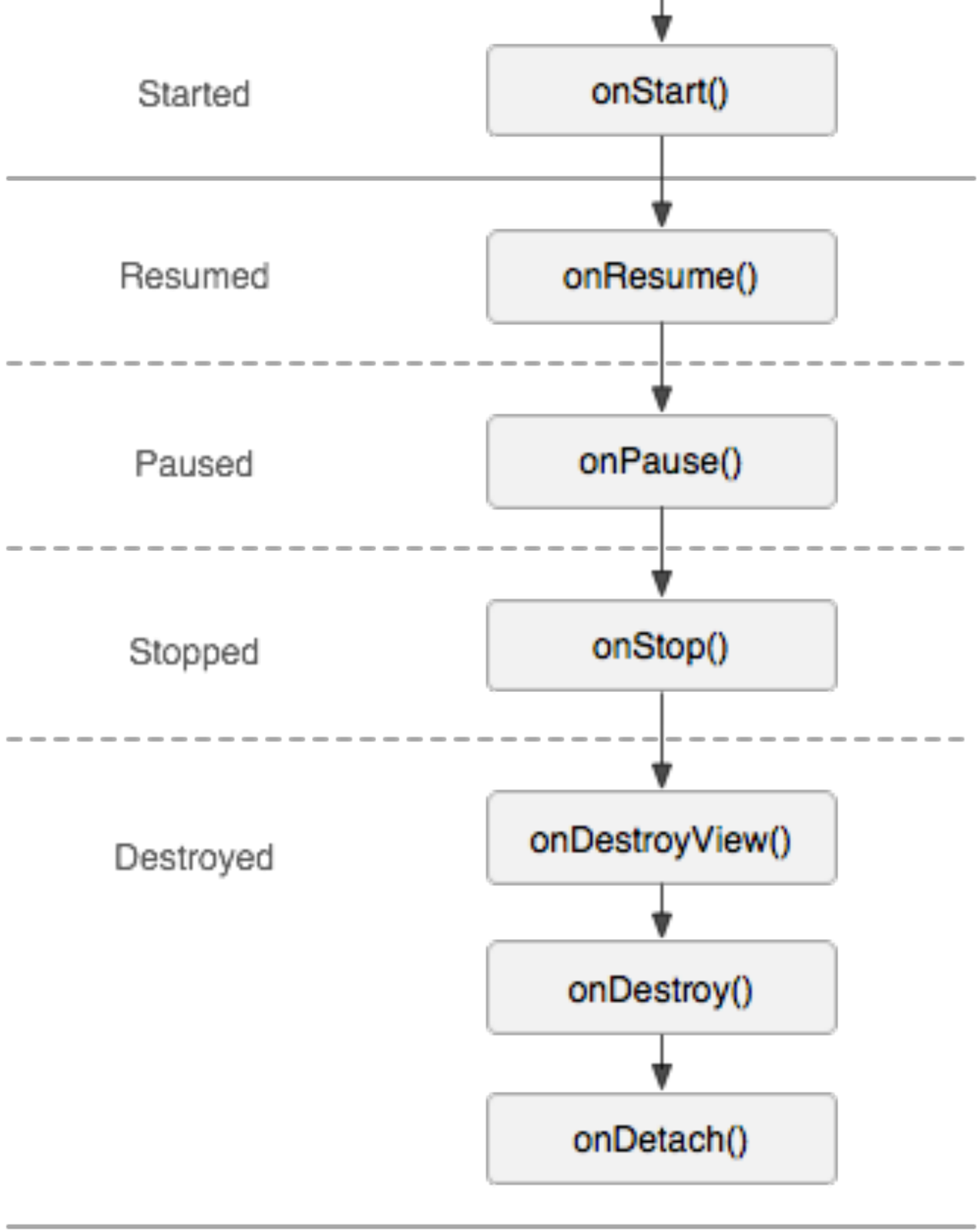
Fragment Lifecycle



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”);
```

Fragments

- can difficult to use

- highly criticised for overly complex API

THE END