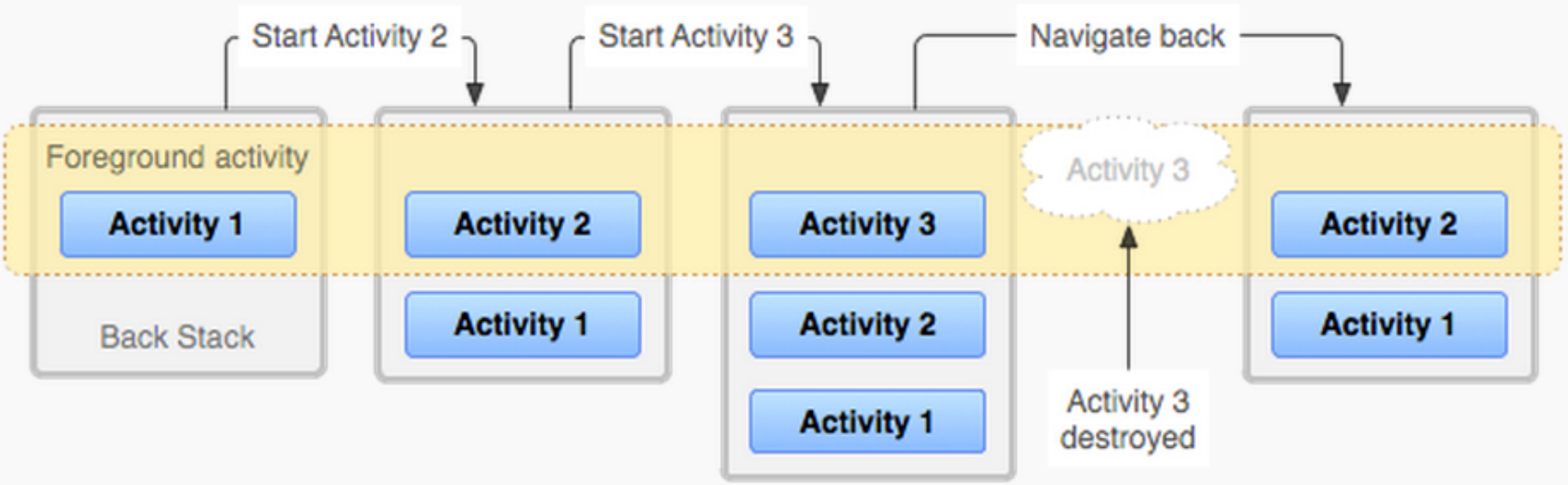
Android Fragments

The Back Stack (Introduction)

- An application is made up of multiple activities
  - And as you know one activity can start another activity, which can, in turn start another activity
- Activities are connected together as a *task*
  - Whether in the same or different applications
- This sequence of activities in a task is called the **back stack**

The Back Stack (Illustration)



Fragments (Introduction)

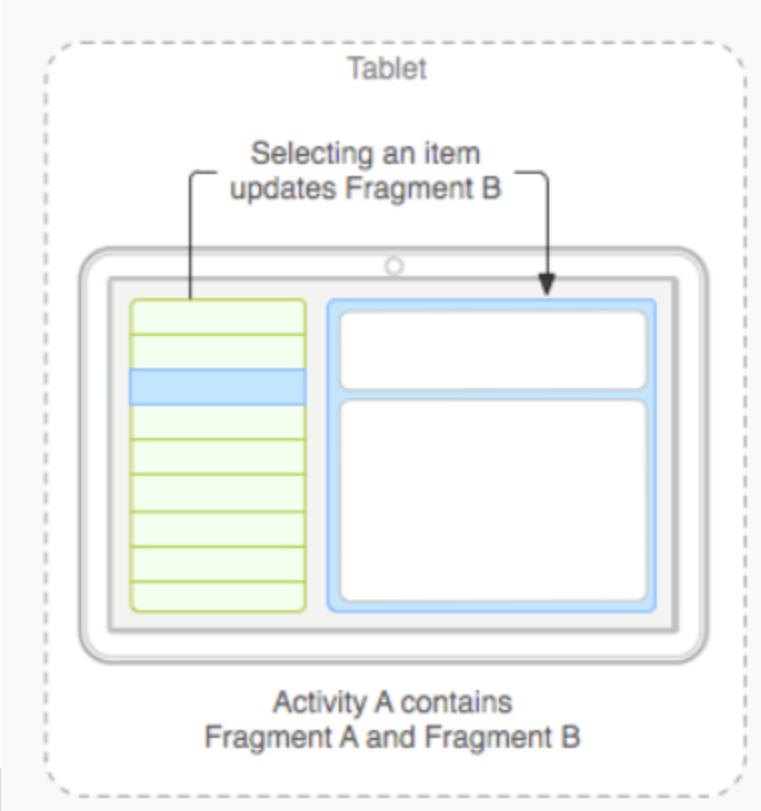
- A **fragment** is a behavior or part of a user interface that can be placed within an activity
  - Use it to create a multi-pane UI
  - Reuse the same fragment in multiple activities
- Think of a fragment as a modular section of an activity which:
  - Has its own lifecycle
  - Receives its own input events

Fragments (Characteristics)

- Fragments are always embedded in an activity
- Fragments are controller objects
- An activity's view contains visual space where the fragment's view will be inserted
  - An activity can have several fragments
- Fragments were introduced at API level 11 along with the first tables
  - There is a support library for older versions

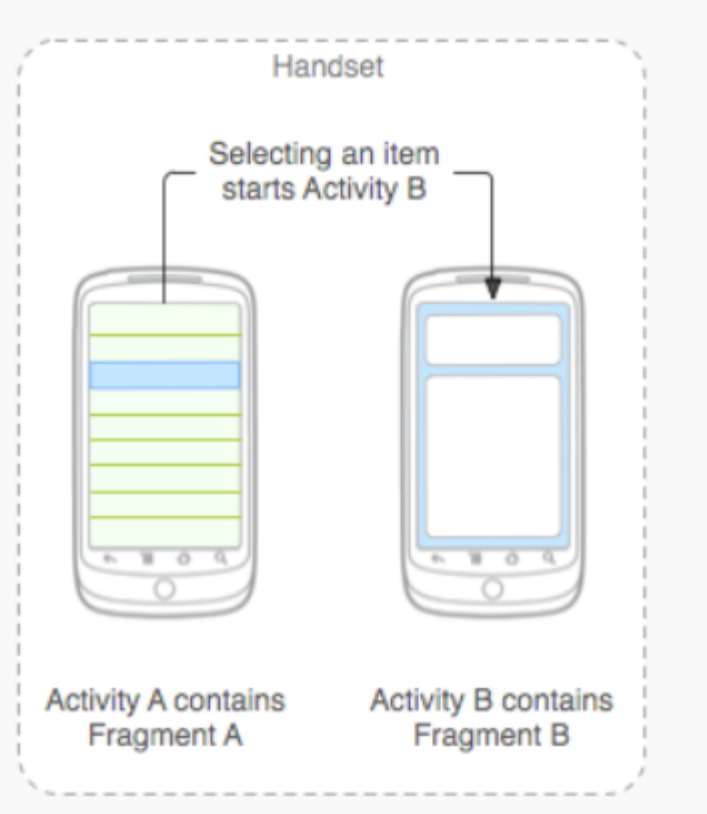
Fragments (Illustration 1)

- Activity with two fragments (tablet)



Fragments (Illustration 2)

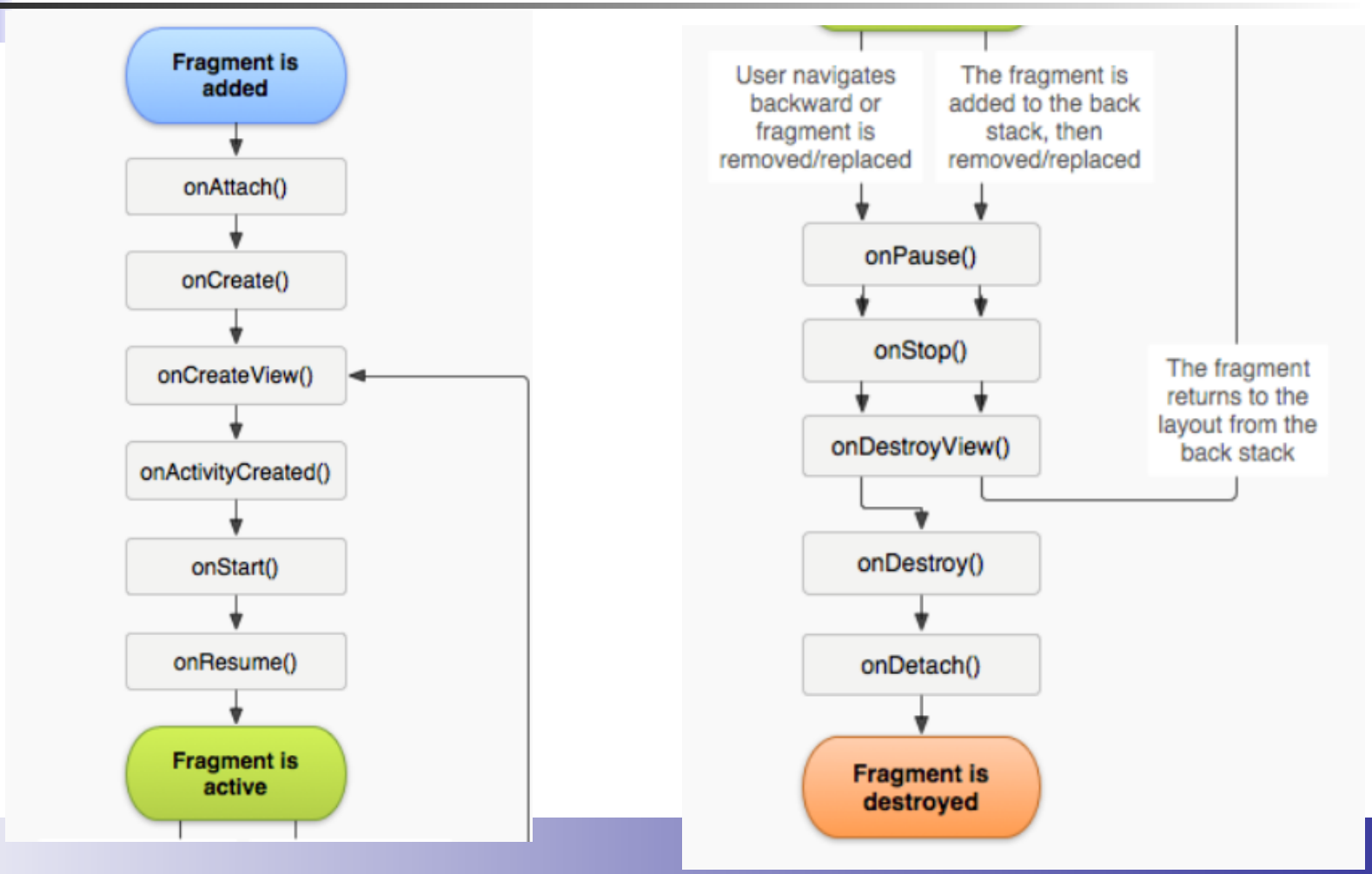
- Activity with two fragments (handset)



The Fragment Lifecycle

- It operates similar to the activity lifecycle
- When an activity is paused, all fragments in it are paused
- Fragments can be manipulated independently of the parent activity

The Fragment Lifecycle



Overriding Fragment Methods (Create)

- To create a fragment you create a subclass of **`Fragment`** (looks like an activity)
- You almost always implement the following methods
  - The system calls **`onAttach()`**, when the fragment is associated with an activity
  - The system calls **`onCreate()`** when creating the fragment

Overriding Fragment Methods (Create)

- The systems calls **`onCreateView()`** when the fragment is rendered the first time
- **`onActivtyCreated()`** tells the fragment that the activity has completed it's call to **`Activity.OnCreate()`**
- **`onStart()`** makes the fragment visible (same as activity)
- **`onResume()`** starts user interaction (same as activity)

Overriding Fragment Methods (Destroy)

- The system calls **`onPause()`** when the user is leaving the fragment
- **`onStop()`** is called when the fragment is no longer visible
- **`onDestroyView()`** allows the fragment to clean up resources associated with the currentview
- **`onDestroy()`** is called to do final cleanup
- **`onDetach()`** is called when the fragment is disassociated with the activity

Hosting Fragments

- There are two ways to host fragments:
- In the activity's layout (static definition)
  - Inflexible because the fragment cannot be swapped out during the activity lifetime
- In the activity's code (dynamic a.k.a. "programmatic" definition)
  - Better because you can dynamically add and remove different fragments
  - More complicated

Hosting Fragments (Static Definition)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">

   <fragment
   android:name="com.example.fragments"
   android:id="@+id/lm_fragment"
   android:layout_weight="1"
   android:layout_width="0dp"
   android:layout_height="match_parent" />

   <fragment
   android:name="com.example.fragments"
   android:id="@+id/pm_fragment"
   android:layout_weight="2"
   android:layout_width="0dp"
   android:layout_height="match_parent" />

</LinearLayout>
```

Hosting Fragments (Layout)

- You connect the layout using the same techniques that you have seen

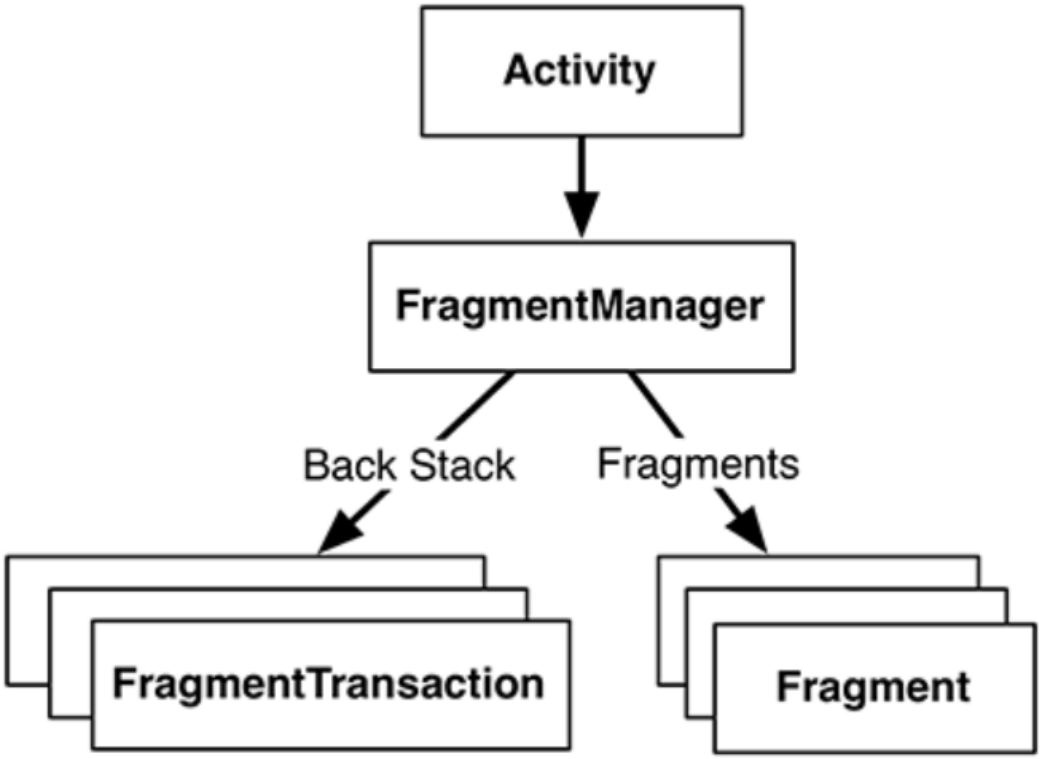
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.fragment_layout);
}
```

Hosting Fragments (Code)

- Add a fragment to an existing **`ViewGroup`**
  - (Layout)
- The **`FragmentManager`** is responsible for managing fragments and adding views to the activity's view hierarchy
- It manages
  - A list of fragments
  - And a back stack of fragment transactions

FragmentManager (Illustration)



Getting the Fragment Manager

- Use **`getFragmentManager()`** to get the current fragment manager
- If you need compatibility with older versions, use **`getSupportFragmentManager()`**

```
FragmentManager fragmentManager =
    getFragmentManager()
```

Managing Fragments (Introduction)

- The **`FragmentManager`** allows you to
  - Get the fragments associated with an activity
  - Create a fragment transaction, which, in turn, affects the back stack (add, replace, remove)
  - **`beginTransaction()`** method starts a series of edit operations on the fragments associated with this **`FragmentManager`**
- The method returns a **`FragmentTransaction`**

Managing Fragments

- Using the **`FragmentManager`** you can
- Get activity fragments with **`findFragmentById()`** or **`findFragmentByTag()`**
- Pop fragments off the back stack with **`popBackStack()`**
  - Acts like the user clicking the back button
- Register a listener for back stack changes with **`addOnBackStackChangedListener()`**

The `FragmentTransaction`

- call **`beginTransaction()`** to start a fragment transaction and **`commit()`** to cause the changes to take effect
- In between the two calls
  - **`add`** places one fragment on top of another fragment (add to the back stack)
  - **`replace`** causes the current fragment to be destroyed specified fragment created (replace item in the back stack

The FragmentTransaction (Example)

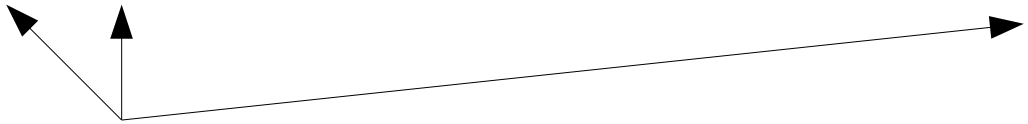
- Create a fragment and transaction:

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

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

The FragmentTransaction ("Fluent Interface" Style)

```
FragmentManager fm = getSupportFragmentManager();
Fragment fragment = fm.findFragmentById(R.id.fragment_container);
if (fragment == null) {
    fragment = createFragment();
    fm.beginTransaction().add(R.id.fragment_container, fragment).commit();
}
```



This usage means that `FragmentManager.beginTransaction` and `FragmentManager.add` must be returning our FragmentManager instance (fm), instead of being `void` (returning nothing).

Fragment (Best Practices)

- The following methods are commonly implemented
  - Display a floating dialog with **`DialogFragment`**
  - Display a list with **`ListFragment`**
  - Create a setting activity with **`PreferenceFragment`**