Navigation & Implicit Intents

(Receiving an implicit Intent)

Navigation

These slides are partially based on the material that Google provides for the course
Android Developer Fundamentals

https://developer.android.com/courses/fundamentals-training/overview-v2

Navigation

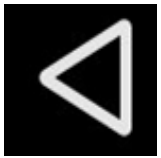
The **majority of the apps** will **include more than one Activity**

Since, few things **frustrate users** more than **basic navigation** that **behaves** in **inconsistent** and **unexpected** **ways**

Consistent navigation is an **essential component** of the **overall user experience**

Two forms of navigation

Android system **supports** **two different forms of navigation** strategies for your app



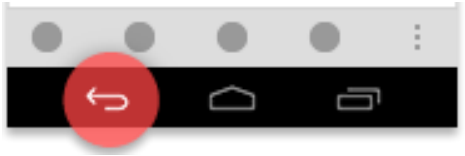
Back (or temporal) navigation

Up (or ancestral) navigation

Back Navigation

allows to **return to the previous Activity** by **tapping** the **device back button**

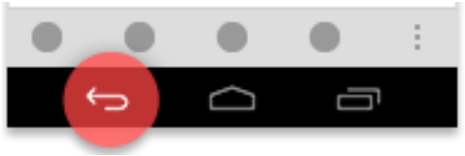
- **controlled by the Android system's back stack**
- **preserves history** of recently viewed screens



Back Navigation

allows to **return to the previous Activity** by **tapping** the **device back button**

- **controlled by the Android system's back stack**
- **preserves history** of recently viewed screens

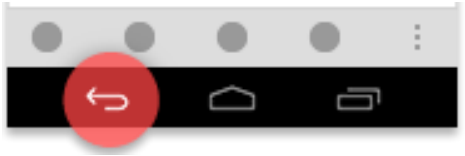


After viewing shopping cart, user decides to add more items, then places order.

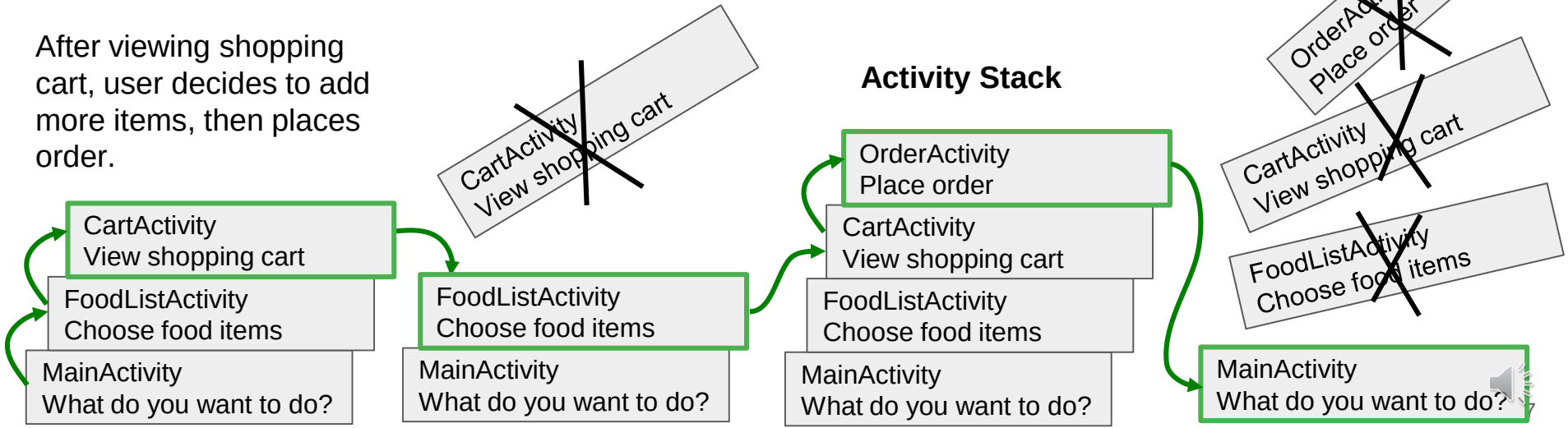
Back Navigation

allows to **return to the previous Activity** by **tapping** the **device back button**

- **controlled by the Android system's back stack**
- **preserves history** of recently viewed screens



After viewing shopping cart, user decides to add more items, then places order.



Up Navigation

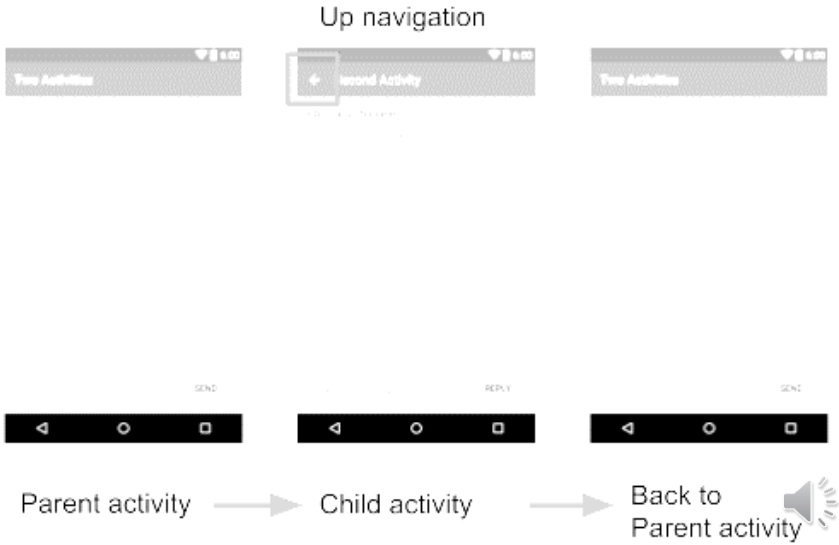
is used to **navigate within an app** based on the **explicit hierarchical relationships** between screens

(sometimes referred to as **ancestral** or **logical navigation**)

- Navigate to parent of current `Activity`
- The `Activity` parent defined in Android manifest using `parentActivityName`

Example for ChildActivity

```
<activity
   android:name=".ChildActivity"
   android:parentActivityName=".MainActivity" >
</activity>
```



Up Navigation

is used to **navigate within an app** based on the **explicit hierarchical relationships** between screens

(sometimes referred to as **ancestral** or **logical navigation**)

- Navigate to parent of current `Activity`
- The `Activity` parent defined in Android manifest using `parentActivityName`

Example for ChildActivity

```
<activity
   android:name=".ChildActivity"
   android:parentActivityName=".MainActivity" >
</activity>
```

Up Navigation

is used to **navigate within an app** based on the **explicit hierarchical relationships** between screens

(sometimes referred to as **ancestral** or **logical navigation**)

- Navigate to parent of current `Activity`
- The `Activity` parent defined in Android manifest using `parentActivityName`

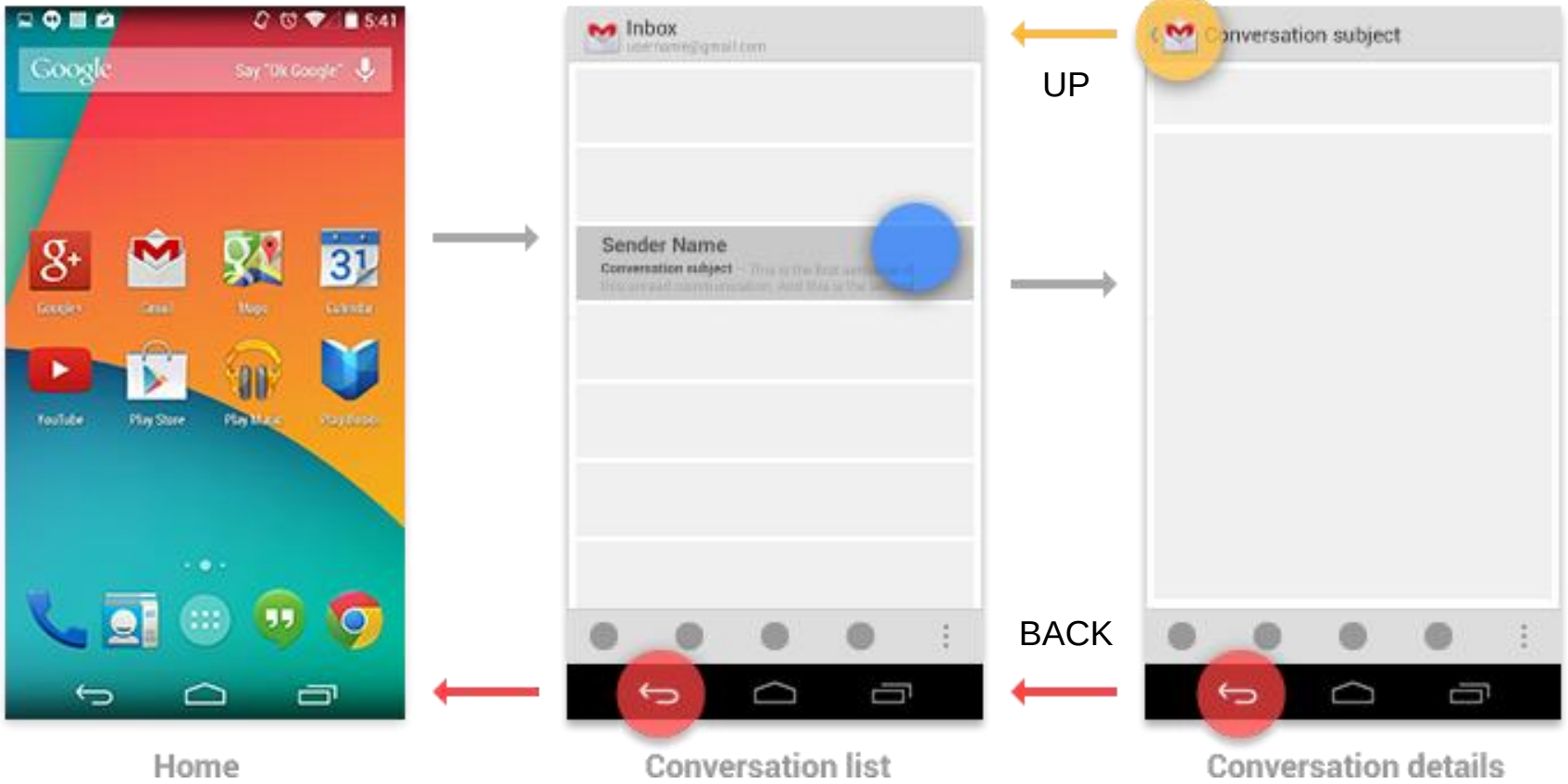
Example for ChildActivity

```
<activity
   android:name=".ChildActivity"
   android:parentActivityName=".MainActivity" >
</activity>
```

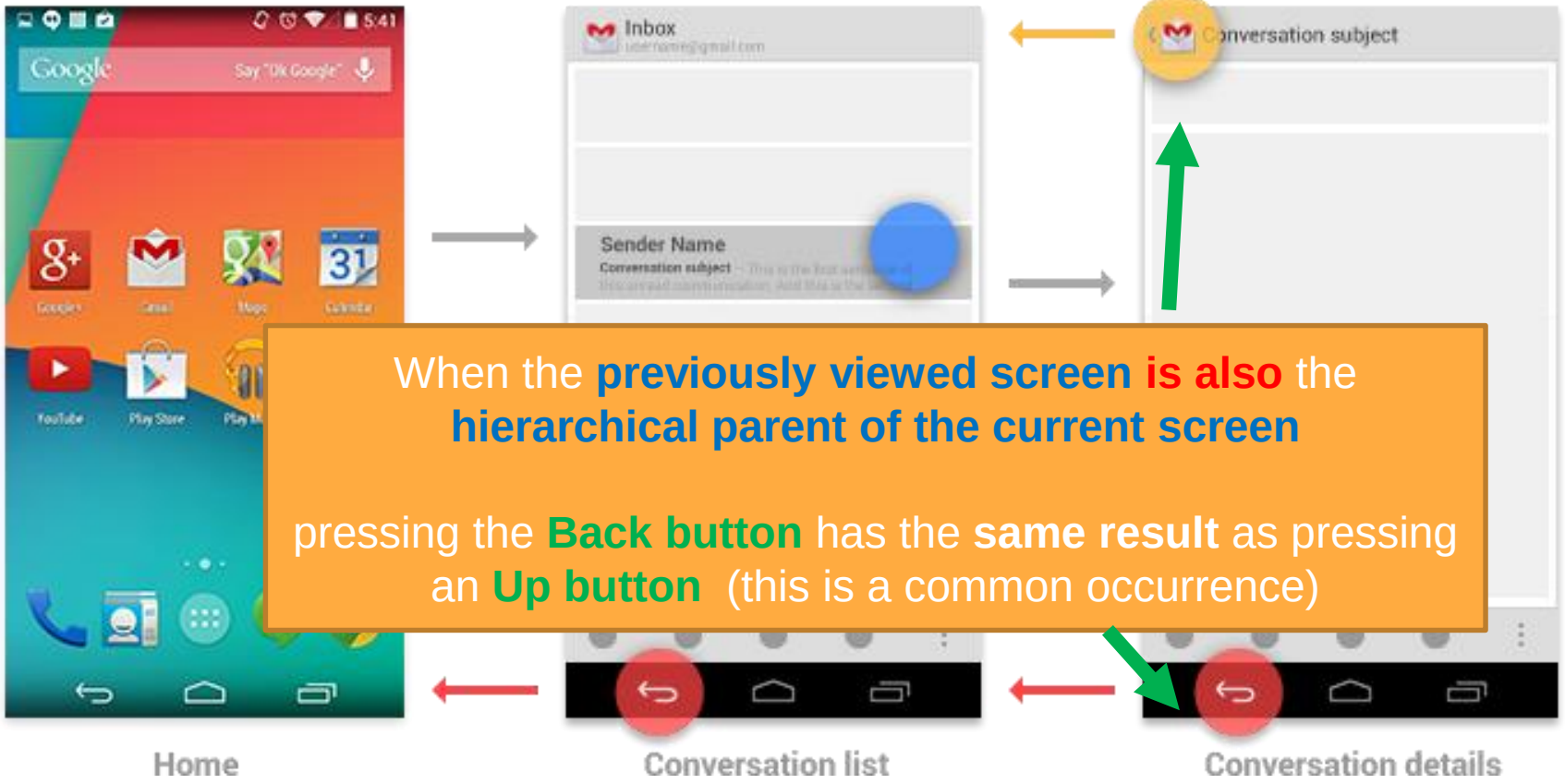
Up Navigation - Manifest

```
<application ... >
    ...

    <!-- The main/home activity (it has no parent activity) -->

    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
        ...
    </activity>

    <!-- A child of the main activity -->
    <activity
        android:name="com.example.myfirstapp.MyChildActivity"
        android:label="@string/title_activity_child"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >

        <!-- Parent activity meta-data to support 4.0 and lower -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

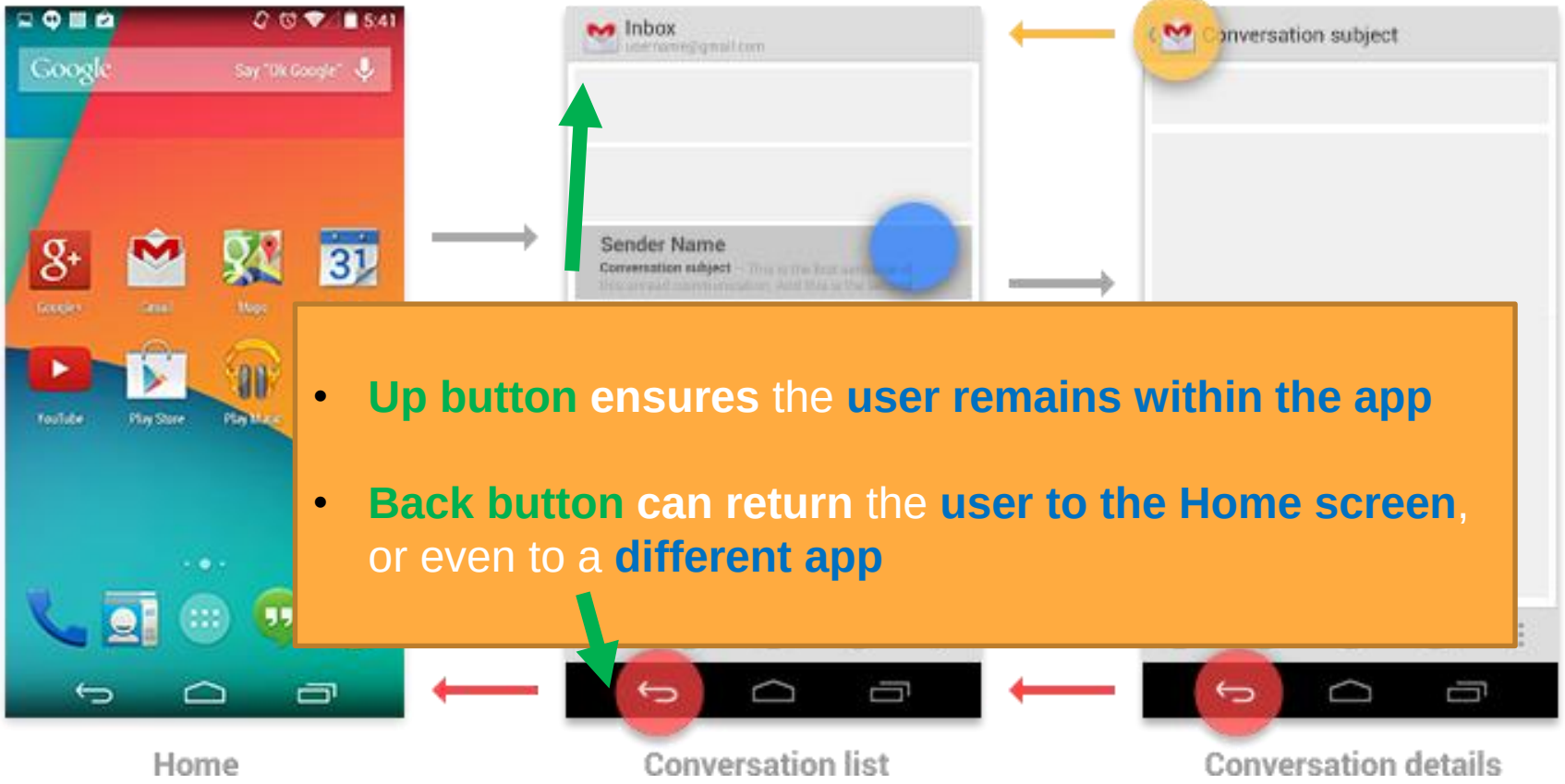
Back vs. Up Navigation



Back vs. Up Navigation



Back vs. Up Navigation



- **Up button ensures** the **user remains within the app**
- **Back button can return** the **user to the Home screen**, or even to a **different app**

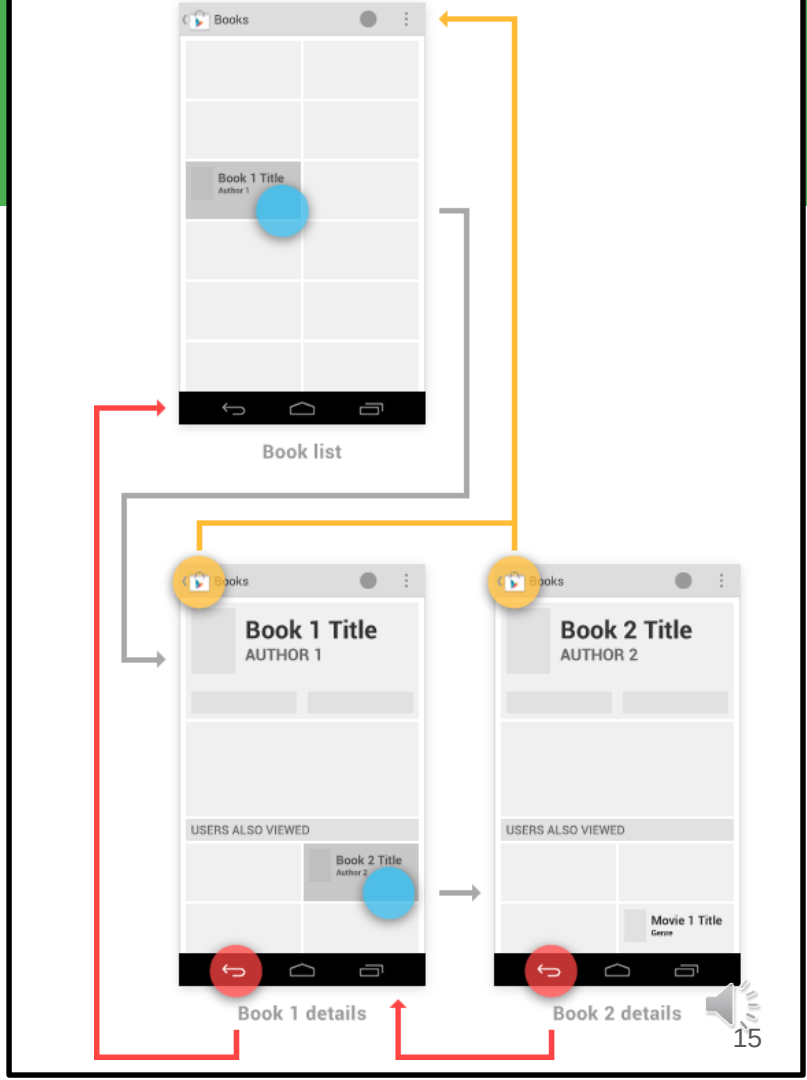
Back vs. Up Navigation

Second example:

1. Select a Book 1
2. View the details of Book 1
3. Select a Book 2
4. View the details of Book 2

Back a **Up** different behaviors

But reach the **same final activity**



Back vs. Up Navigation

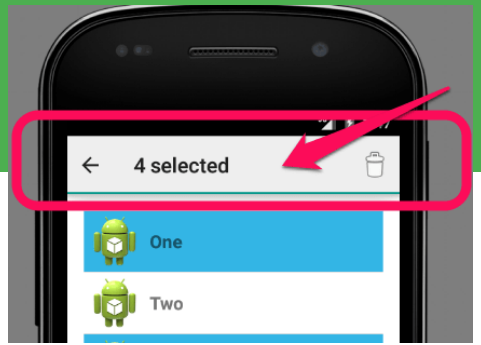
Advanced example:

1. Select a Book 1
2. View the details of Book 1
3. Select a Book 2
4. View the details of Book 2
5. Select a Movie

Back a **Up** different behaviors

and reach **different** **final activities**

Back Button



Back button also ***supports*** some **different behaviors** (not directly tied to screen-to-screen navigation):

- ***Dismisses floating windows*** (dialogs, popups)
- ***Dismisses contextual action bars***, and ***removes the highlight*** from the selected items
- ***Hides the onscreen keyboard*** (IME)

Implicit Intents

Additional Details and Receiving an implicit Intent

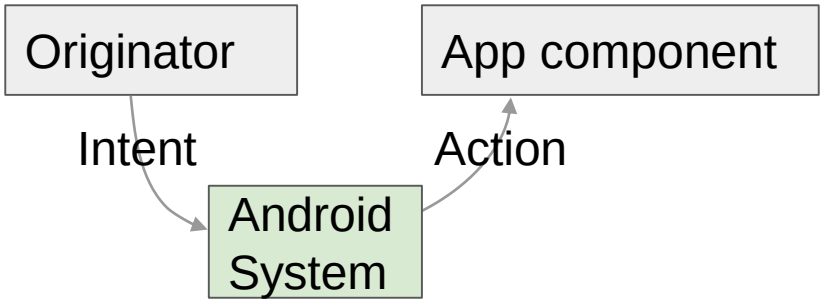
These slides are partially based on the material that Google provides for the course ***Android Developer Fundamentals***

https://developer.android.com/courses/fundamentals-training/overview-v2

What is an Intent?

An `Intent` is:

- **Description** of an **operation to be performed**
- **Messaging object used** to **request an action from another app component** via the Android system.



Explicit vs. implicit Intent

Explicit Intent — Starts an Activity of a **specific class**

generally to starts components between the application because we know the package of the components.

Implicit Intent — Asks **system to find an Activity class** with a **registered handler that can handle this request**

done by our application

for example find implicitly the best camera app inside the system to perform a camera activity

Implicit Intent

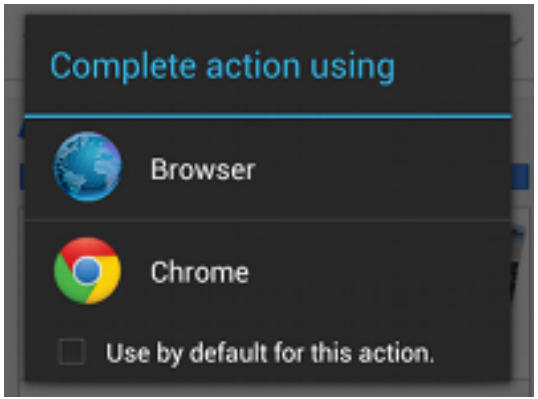
- **Start an Activity** in **another app** by **describing an action you intend to perform**
  - e.g.,: "share an article", "view a map", or "take a picture"
- **Specify an action** and ***optionally provide data*** with which to perform the action
  - **Don't specify the target Activity class**, just the intended action
- **Android runtime matches the implicit intent request with registered intent handlers**
- **If there are multiple matches**, an **App Chooser** will open to let the user decide

App Chooser

When the **Android runtime finds multiple registered activities** that can handle an implicit Intent

it displays an App Chooser to allow the user to select the handler

as usual



How does implicit Intent work?

- The **Android Runtime keeps a list of registered Apps**
- **Apps have to register** via **AndroidManifest.xml**
- **Runtime receives the request** and **looks for matches**
  - uses Intent filters for matching from **AndroidManifest.xml**
- If **more than one match**
  - shows a list of possible matches and lets the user choose one
- **Android runtime starts the requested activity**

Implicit Intent examples (Invocation)

Show a web page

```
Uri uri = Uri.parse("http://www.google.com");
Intent it = new Intent(Intent.ACTION_VIEW,uri);
startActivity(it);
```

Dial a phone number

```
Uri uri = Uri.parse("tel:8005551234");
Intent it = new Intent(Intent.ACTION_DIAL, uri);
startActivity(it);
```

Receiving an Implicit Intent

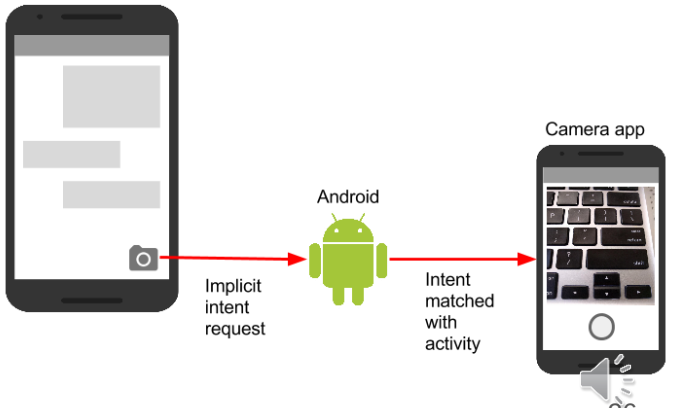
Register your app to receive an Intent

If an **Activity in your app** **have to respond** to an **implicit Intent** (from your own app or other apps)

declare one or more **Intent filters** in the **AndroidManifest.xml** file

Each **Intent filter** **specifies** **the type of Intent** it accepts based on the **action**, **data**, and **category** for the Intent

The **system will deliver** an **implicit Intent** **to your app** component **only if** that **Intent can pass through one of your Intent filters**



Intent action, data, and category

- **Action** is the generic ***action the receiving Activity should perform***. The available Intent actions are defined as constants in the Intent class and begin with the word ACTION_.
- **Category** (optional) provides additional ***information about the category of component that should handle the Intent***. Intent categories are also defined as constants in the Intent class and begin with the word CATEGORY_.
- **Data type** indicates the ***MIME type of data the Activity should operate on***. Usually, the data type is inferred from the URI in the Intent data field, but you can also explicitly define the data type with the setType() method.

Intent action, data, and category

- **Action** is the generic ***action the receiving Activity should perform***. The available Intent actions are defined as constants in the Intent class and begin with the word ACTION_.
- **Category** (optional) provides additional ***information about the category of component that should handle the Intent***. Intent categories are also defined as constants in the Intent class and begin with the word CATEGORY_.
- **Data type** indicates the ***MIME type of data the Activity should operate on***. Usually, the data type is inferred from the URI in the Intent data field, but you can also explicitly define the data type with the setType() method.

Intent action, data, and category

- **Action** is the generic ***action the receiving Activity should perform***. The available Intent actions are defined as constants in the Intent class and begin with the word ACTION_.
- **Category** (optional) provides additional ***information about the category of component that should handle the Intent***. Intent categories are also defined as constants in the Intent class and begin with the word CATEGORY_.
- **Data type** indicates the ***MIME type of data the Activity should operate on***. Usually, the data type is inferred from the URI in the Intent data field, but you can also explicitly define the data type with the setType() method.

Intent action, data, and category

Intent **actions, categories,** and **data** types are used both by

- the ***Intent object you create*** in your sending Activity
- as well as, in the ***Intent filters you define*** in the AndroidManifest.xml file for the receiving Activity.

The Android system uses this information to match an ***implicit Intent request with an Activity*** or ***other component that can handle that Intent***

Intent filter in AndroidManifest.xml

```
<activity android:name="ShareActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
```

Intent filters: action and category

- `action` — Match one or more action constants
  - `android.intent.action.VIEW` — matches any Intent with `ACTION_VIEW`
  - `android.intent.action.SEND` — matches any Intent with `ACTION_SEND`

- `category` — additional information (list of categories)
  - `android.intent.category.BROWSABLE`—can be started by web browser
  - `android.intent.category.LAUNCHER`—Show activity as launcher icon

Intent filters: data

- `data` — Filter on data URIs, MIME type
  - `android:scheme="https"`—require URIs to be https protocol
  - `android:host="developer.android.com"`—only accept an Intent from specified hosts
  - `android:mimeType="text/plain"`—limit the acceptable types of documents

An Activity can have multiple filters

```
<activity android:name="ShareActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    ...
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.SEND_MULTIPLE"/>
    ...
  </intent-filter>
</activity>
```

An Activity can have several filters

A filter can have multiple actions & data

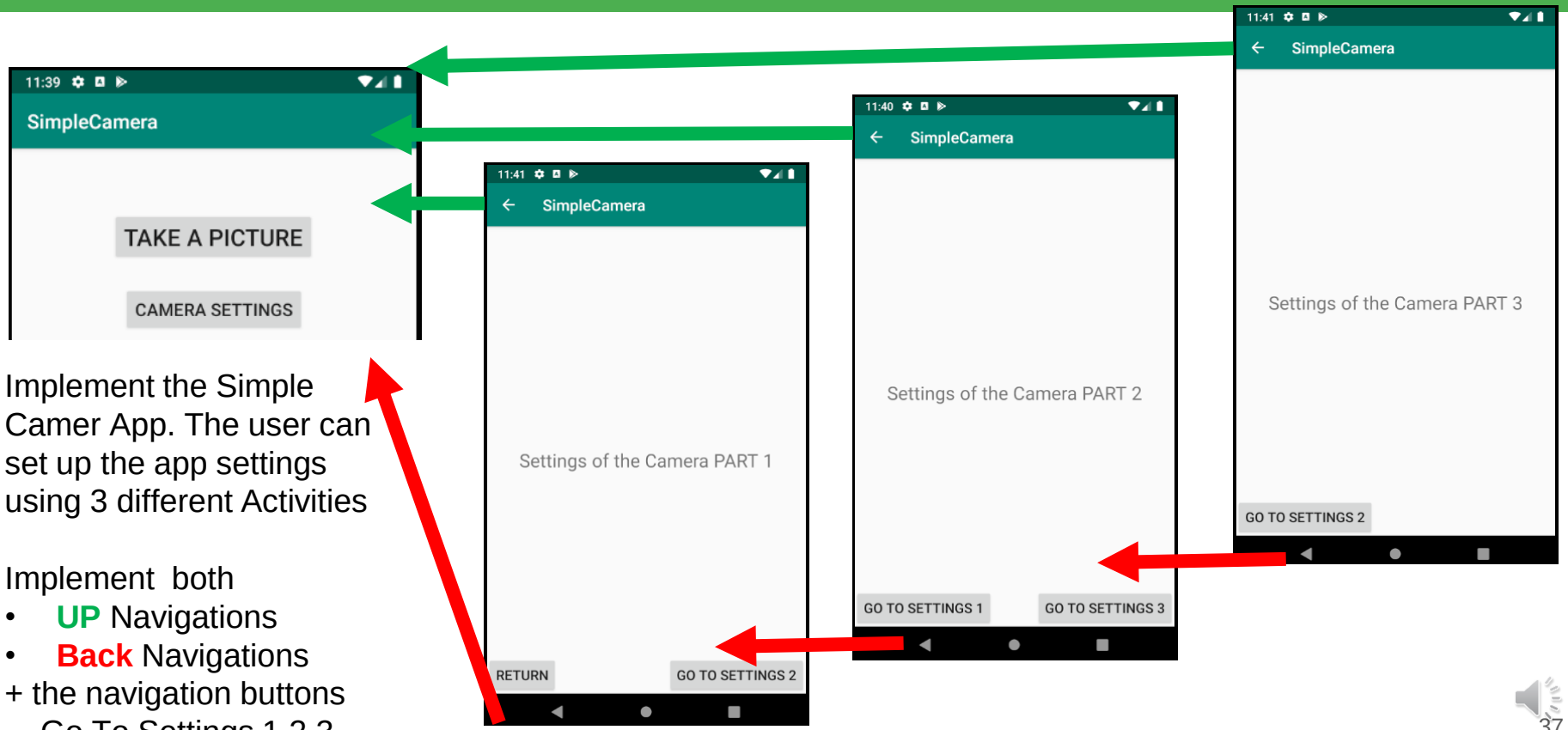
```
<intent-filter>
  <action android:name="android.intent.action.SEND"/>
  <action android:name="android.intent.action.SEND_MULTIPLE"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <data android:mimeType="image/*"/>
  <data android:mimeType="video/*"/>
</intent-filter>
```

Receiving an implicit Intent

Once the **Activity is successfully launched with an implicit Intent**, from that Activity it is possible to **handle the Intent** and **its data** in the same way you did for an explicit Intent, by:

1. Getting the Intent object with getIntent()
2. Getting Intent data or extras out of that Intent
3. Performing the task the Intent requested
4. Returning data to the calling Activity with another Intent, if needed.

Simple Camera App (1)



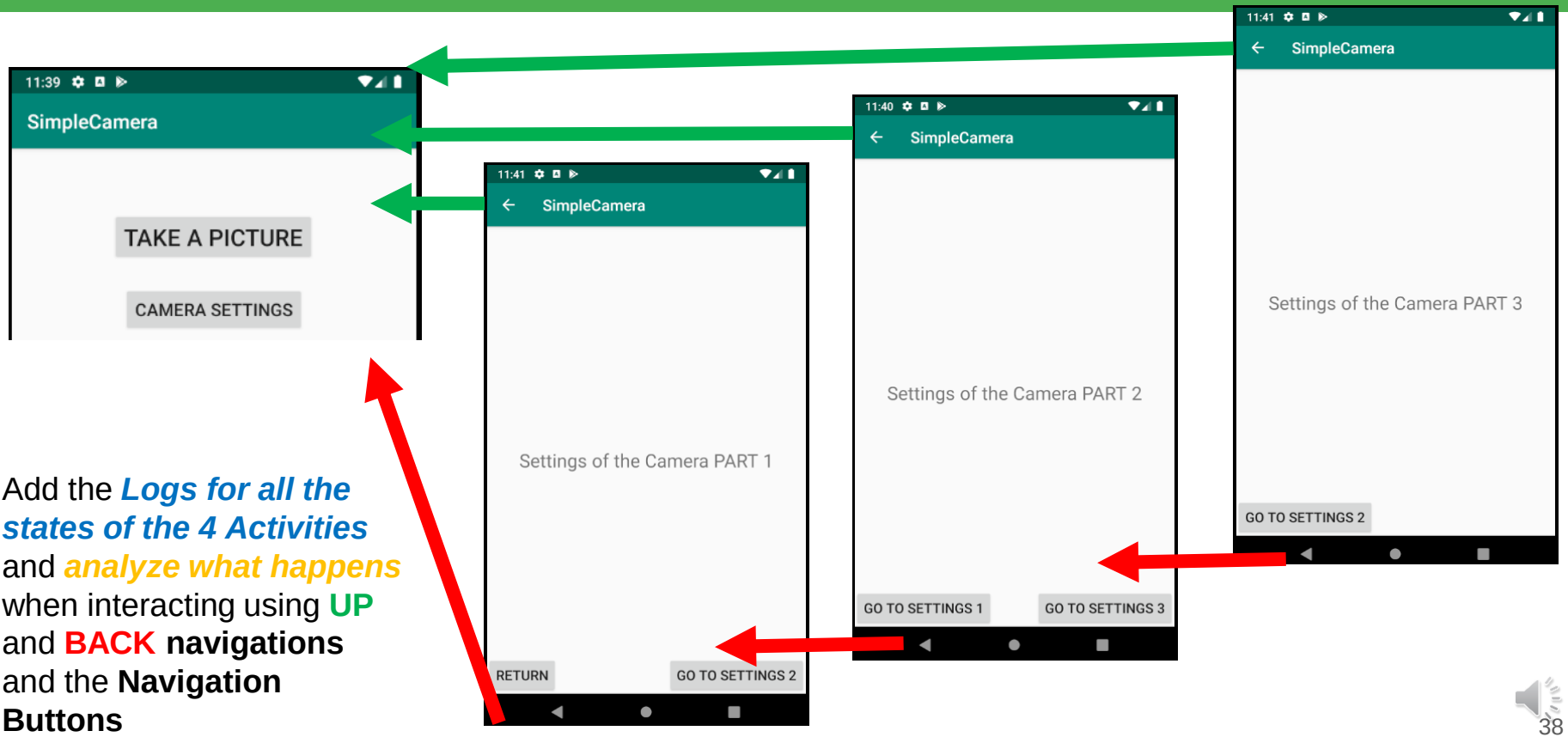
Implement the Simple Camer App. The user can set up the app settings using 3 different Activities

Implement both

- **UP** Navigations
- **Back** Navigations

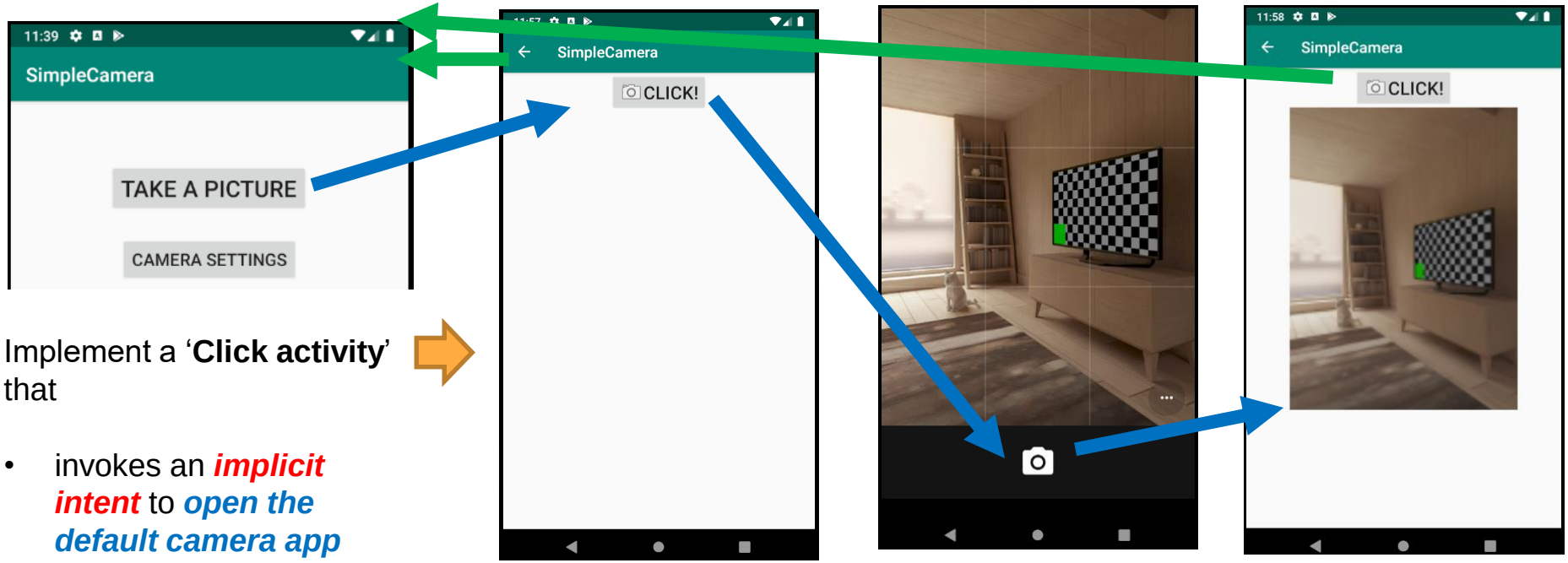
+ the navigation buttons
Go To Settings 1,2,3

Simple Camera App (2)



Add the ***Logs for all the states of the 4 Activities*** and ***analyze what happens*** when interacting using **UP** and **BACK** **navigations** and the **Navigation Buttons**

Simple Camera App (3)

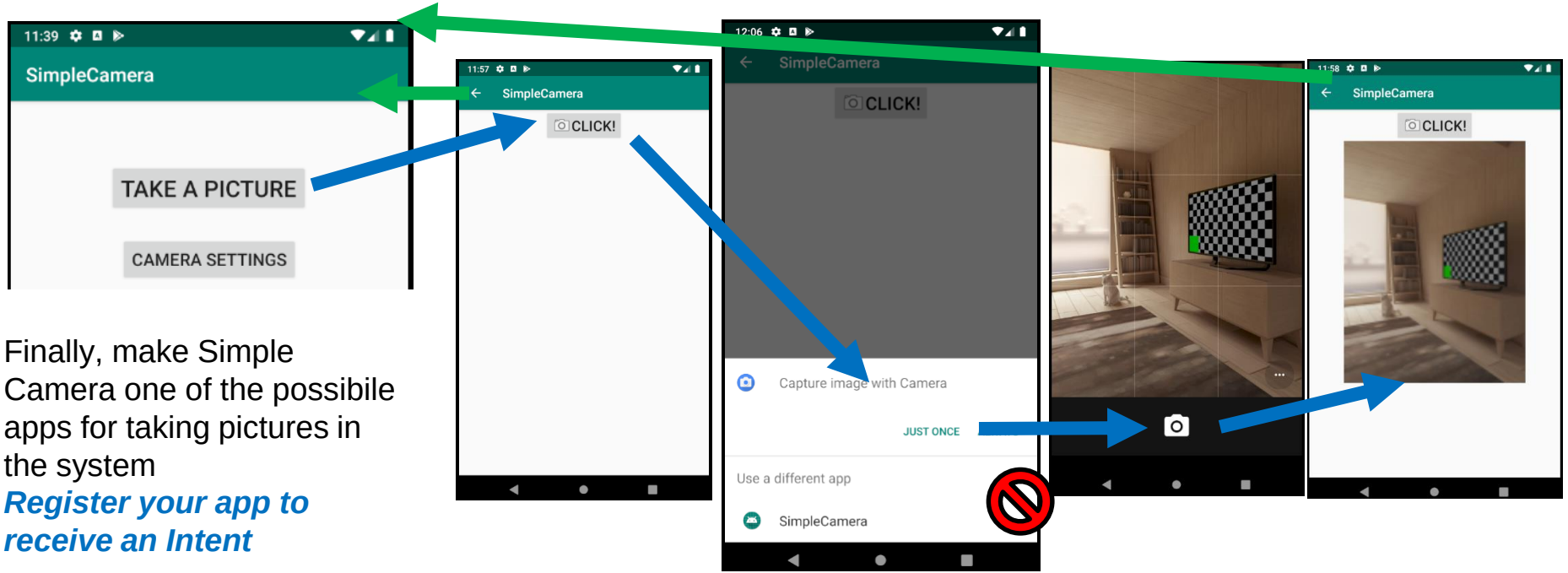


Implement a '**Click activity**' that

- invokes an ***implicit intent*** to ***open the default camera app***
- and then ***visualizes a thumbnail of the picture***

Add also the **UP navigations** as shown in the figures

Simple Camera App (4)



Finally, make Simple Camera one of the possibile apps for taking pictures in the system

Register your app to receive an Intent

Modify the AndroidManifest.xml.... **NOTE:** this will allow to select also Simple Camera app and does not make any sense since the intent starts from the same app... Simple Camera is not able to take 'directly' a picture but relies on the default app... We try this only for making an exercise... Thus in the 'Chooser' select always the smartphone or emulator default camera app...