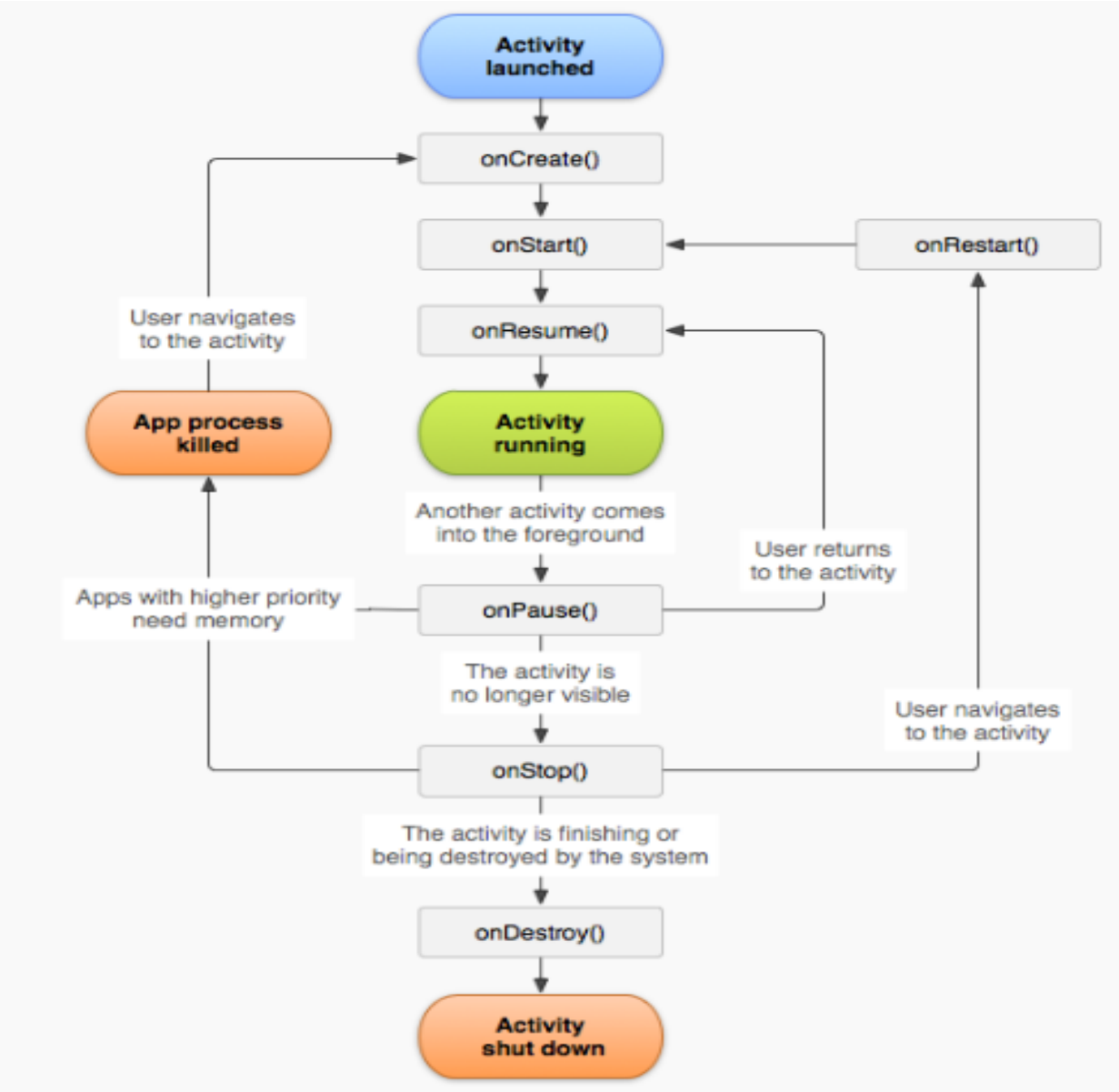
Intents & Activity

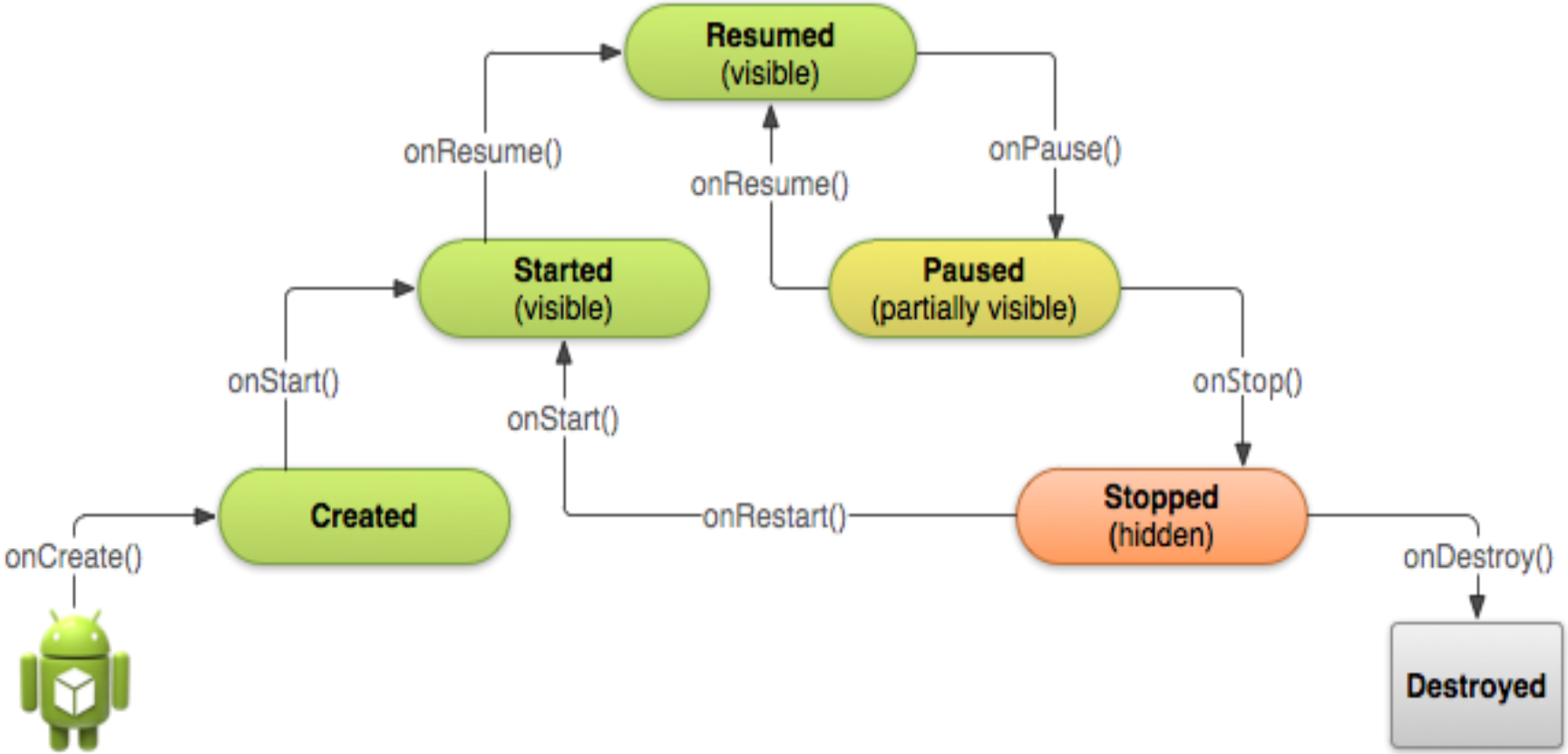
Android Activities

- An Android application could include any number of activities
- Activities are independent of each other; however they usually cooperate exchanging data and actions.
- Typically, one of the activities is designated as the first one (*main) that* should be presented to the user when the application is launched
- Moving from one activity to another is accomplished by asking the current activity to execute an ***intent.***
- Activities interact with each other in an **asynchronous mode**

Activities Lifecycle



Activities Lifecycle



Activities: Create New Ones

- A new Activity is a new class.
- Step One: Create a new Class
  - Select src and package name, right click, select 'New' and then select 'Class'. Enter the Name as NewActivity and set the Superclass to android.app.Activity, then hit Finish
- Step Two: Entry into Manifest file
  - The next step is adding the Activity to your Manifest

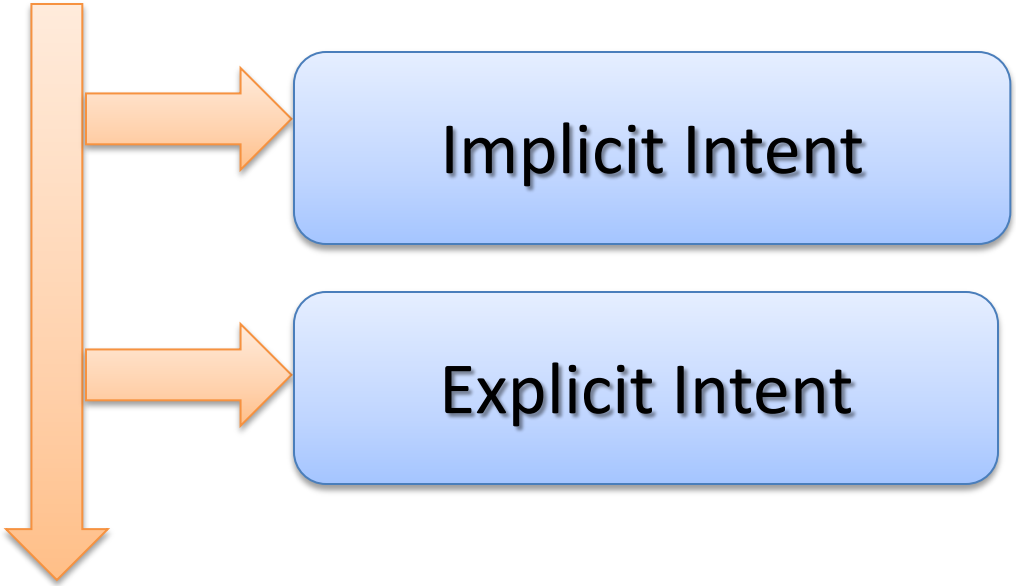
Intent

- An intent is an abstract description of an operation to be performed.
- It can be used with

Actions	Methods
Launch an activity	startActivity (intent) startActivityForResult(Intent)
Start a service	startService(intent) bindService(intent, …)
Send an intent to interested BroadcastReceivers	sendBroadcast(intent)

Types of Intent

There are two primary forms of intents:



Explicit Intent

- Explicitly defines the component which should be called by the system, by using the Java class as identifier.

- The following shows how to create an *explicit intents* and send it to the Android system. If that class represents an *activity* Intent the Android system starts it

```
Intent exp_int = new Intent(this, ActivityTwo.class);
startActivity(exp_int) ;
```

Starting Activities

- The **startActivity(Intent) method is used to start a new activity, which** will be placed at the top of the activity stack. The caller however continues to execute in its own thread.

- An intent contains the action and optionally additional data. The component which creates the Intent can add data to it via the overloaded putExtra() method. Extras are key/value pairs; the key is always a String. As value you can use the primitive data types (int, float,..) plus objects of type String, Bundle, Parceable and Serializable.

- The receiving component can access the Intent object via the getIntent() method and from the intent more information may be accessed:
  - Action using getAction()
  - Data using getData()
  - Extra data using getExtra(getIntExtra, getStrinExtra etc) or getExtras

Finishing Activity

- Calling **finish()** on the current activity closes it and removes it from the activity stack and takes the control to top activity in stack
- Pressing the hardware back button or calling **onBackPressed()** closes the activity and removes it from the activity stack and takes the control to top activity in stack
- Calling **startActivity(Intent)** will take the control to the new activity but not remove current activity stack

Implicit Intent

- *S*pecify the **action** which should be performed and optionally data which provides data for the action.

```
Intent impint=new Intent(Action, Data);
startActivity(impint);
```

Implicit Intent: Load a web page

Action ACTION_VIEW

Data URI Scheme http:*<URL>* or https:*<URL>*

MIME Type

PLAIN_TEXT_TYPE ("text/plain")

"text/html"

"application/xhtml+xml"

"application/vnd.wap.xhtml+xml"

<uses-permission android:name=*"android.permission.INTERNET"/>*

Open Facebook Using Intent :-

```
Button btn_call=(Button)findViewById(R.id.button2);
 btn_call.setOnClickListener(new View.OnClickListener()
{
 @Override
 public void onClick(View v)
 {
 // TODO Auto-generated method stub
Intent in=new intent(Intent.ACTION_VIEW,Uri.parse("http://www.facebook.com"));
 startActivity(in);
 } });
```

Implicit Intent: Initiate a Phone Call

Action ACTION_CALL

Data URI Scheme tel:<phone-number>

MIME Type None

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

Calling Using Intent

```
Button b1= (Button)findViewById(R.id.button1);
b1.setOnClickListener(new View.OnClickListener()
{
@Override
public void onClick(View arg0)
{
  Intent  incall= new  Intent(Intent.ACTION_CALL, Uri.parse("tel:877470277"));
  startActivity(incall);
} });
```

Implicit Intent: More examples of action/data pairs:

- **ACTION_VIEW** ***content://contacts/people/1*** --
  Display information about the person whose identifier is "1".

- **ACTION_VIEW** ***content://contacts/people/*** --
  Display a list of people, which the user can browse through. This example is a typical top-level entry into the Contacts application, showing you the list of people.

- **ACTION_VIEW** ***tel:123*** --
  Display the phone dialer with the given number filled in. Note how the VIEW action does what what is considered the most reasonable thing for a particular URI.

- **ACTION_EDIT** ***content://contacts/people/1*** --
  Edit information about the person whose identifier is "1".

- **ACTION_DIAL** ***tel:123*** --
  Display the phone dialer with the given number filled in.

- **ACTION_DIAL** ***content://contacts/people/1*** --
  Display the phone dialer with the person filled in.

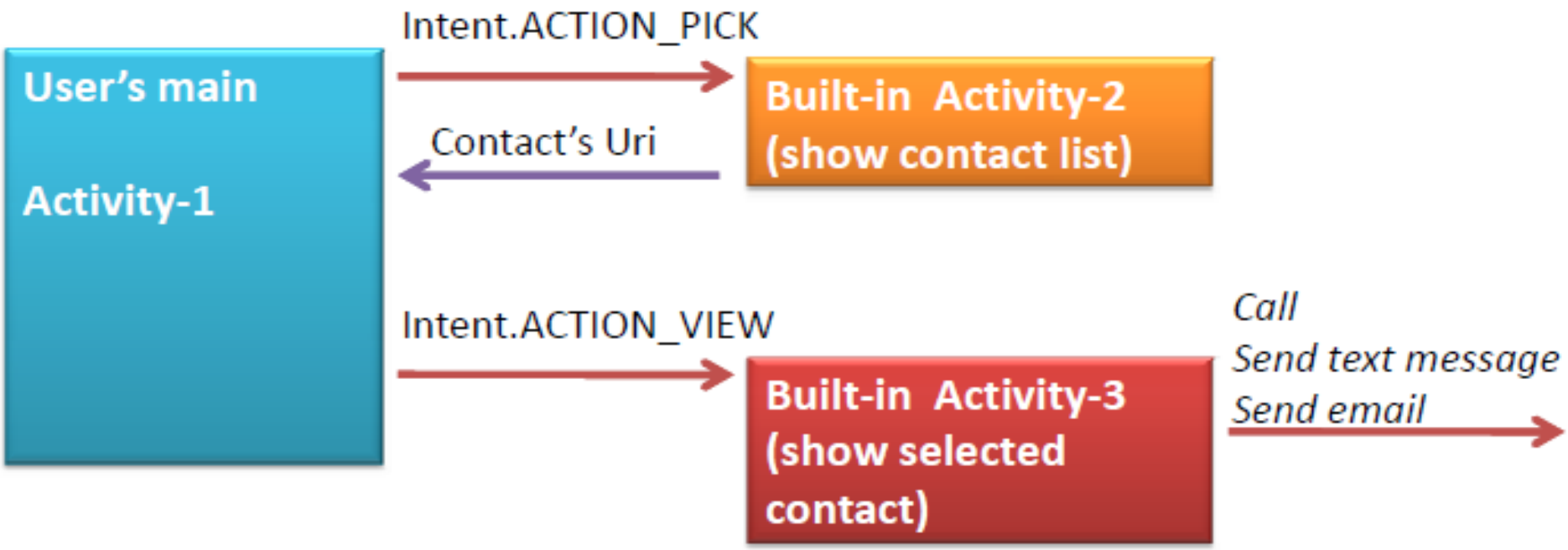
Implicit Intent

Example1.

1. Show all contacts and pick a particular one (*Intent.ACTION_PICK).*

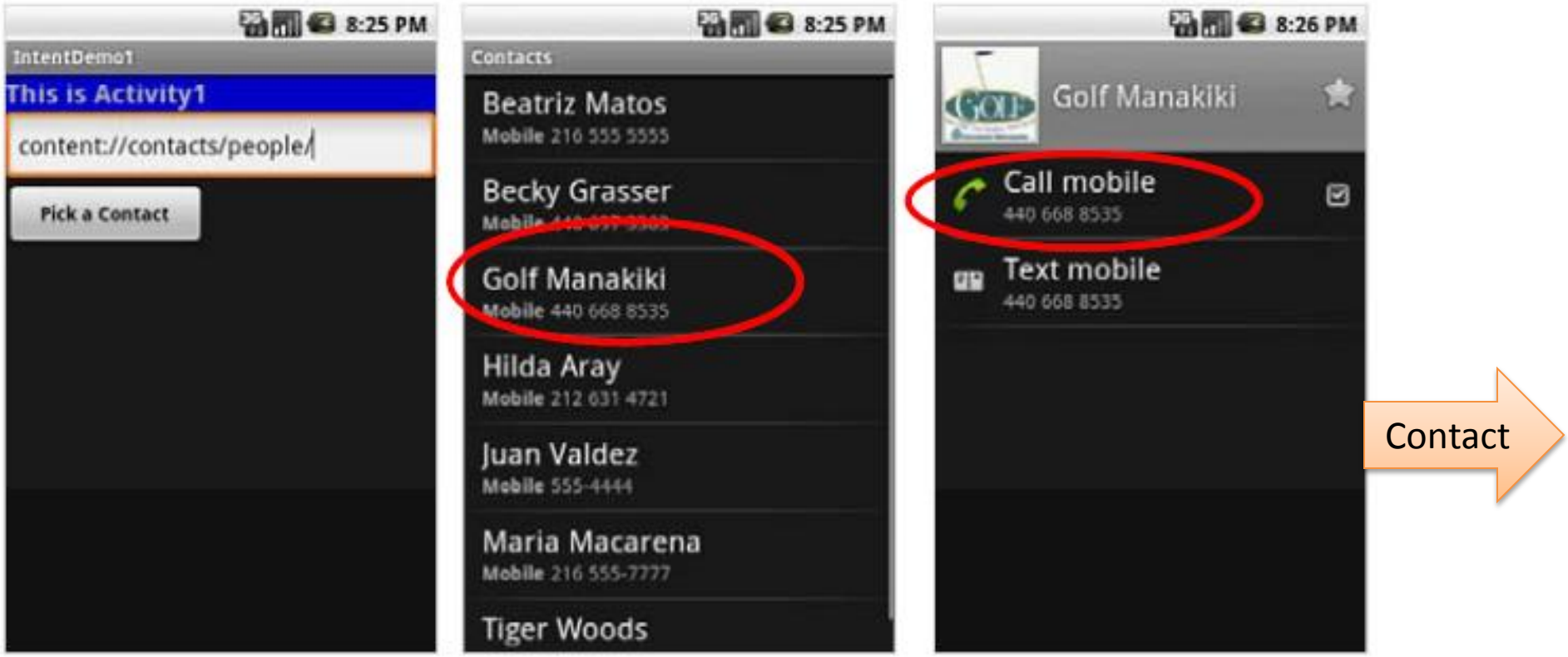
2. For a successful interaction the main-activity accepts the returned URI identifying the person we want to call (*content://contacts/people/n).*

3. 'Nicely' show the selected contact's entry allowing calling, texting, emailing actions (Intent.ACTION_VIEW).



Implicit Intent

Let's play golf - *Call for a tee-time*.



Main Activity

Intent.ACTION_PICK

Intent.ACTION_VIEW

Data transfer between activities

1. Data transfer to the Sub Activity

```
Intent  go ;
go=new Intent(this,Activity2.class);
go. putExtra("key1","sisoft");
startActivity (go);
```

Activity 1

```
String data;
Intent intent=getIntent();
data= intent.getStringExtra( "key1");
```

Activity 2

2. Getting Results Back from SubActivity

- **Calling Activity**
- In order to get results back from the called activity we use the method

```
startActivityForResult ( Intent, requestCodeID )
```

- Where *requestCodeID is an arbitrary value you choose to identify the* call.

- The result sent by the sub-activity could be picked up through the sub Listener-like asynchronous method

```
onActivityResult ( requestCodeID, resultCode, Intent )
```

2. Getting Results Back from SubActivity

Called Activity

Before an invoked activity exits, it should call **setResult (resultCode, Intent data)** to return a termination signal back to its parent.

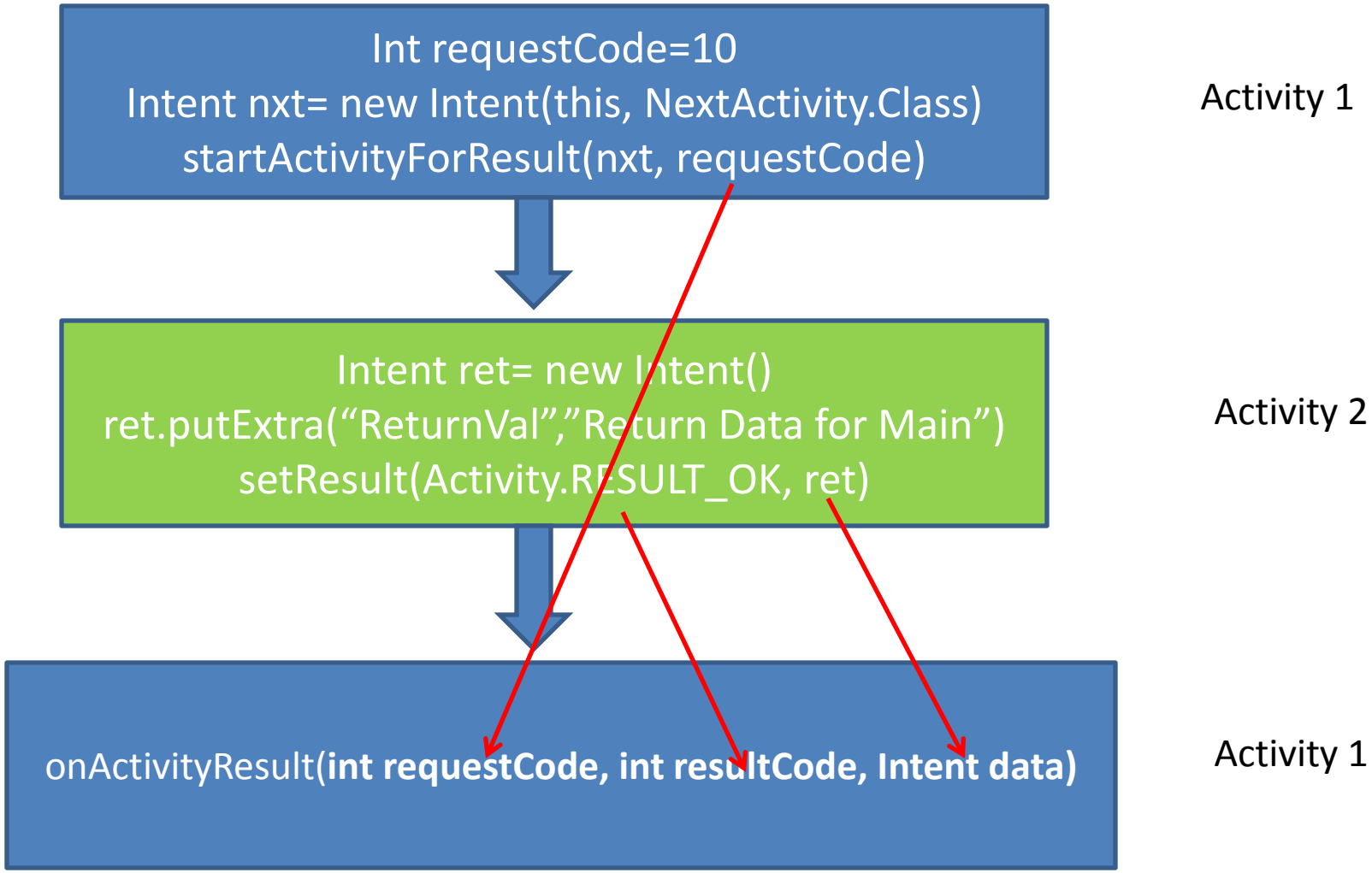
- It is convenient to supply a result code, which can be the standard results **Activity.RESULT_CANCELED, Activity.RESULT_OK,** or any custom values.

- If a child activity fails for any reason (such as crashing), the parent activity will receive a result with the code **RESULT_CANCELED.**

- All of this information can be capture back on the parent's

onActivityResult (int requestCodeID, int resultCode, Intent data)

2. Getting Results Back from SubActivity



3. Share intent (Intent.ACTION_SEND)

Lots of Android applications allow you to share some data with other people, e.g. the Facebook, G+, Gmail and Twitter application. You can send data to one of this components. The following code snippet demonstrates that.

```
Intent intent = new Intent(Intent.ACTION_SEND);
 intent.setType("text/plain");
intent.putExtra(android.content.Intent.EXTRA_TEXT, "News for you!");
startActivity(intent);
```

Share Intent: Send an Email

```
Intent email = new Intent(Intent.ACTION_SEND);
email.putExtra(Intent.EXTRA_EMAIL, new String[]{ to});

email.putExtra(Intent.EXTRA_SUBJECT, subject);
 email.putExtra(Intent.EXTRA_TEXT, message);
  //need this to prompts email client only
email.setType("message/rfc822");
startActivity(Intent.createChooser(email, "Choose an E
    mail client :"));
```

Intent Structure

- **Component :** Specifies an explicit name of a component class to use for the intent. If this attribute is set then none of the evaluation is performed, and this component is used exactly as is. By specifying this attribute, all of the other Intent attributes become optional

- **Action :** A string that specifies the generic action to perform. The action largely determines how the rest of the intent is structured—particularly what is contained in the data and extras

- **Category :** Gives additional information about the action to execute.

  For example,

  CATEGORY_LAUNCHER means it should appear in the Launcher as a top-level application, while CATEGORY_ALTERNATIVE means it should be included in a list of alternative actions the user can perform on a piece of data.

Intent Structure

- **Type :** Specifies an explicit type (a MIME type) of the intent data. Normally the type is inferred from the data itself. By setting this attribute, you disable that evaluation and force an explicit type.

- **Extras :** This is a **Bundle** of any additional information. This can be used to provide extended information to the component. For example, if we have a action to send an e-mail message, we could also include extra pieces of data here to supply a subject, body, etc.

Intent Filter

Intent Filter

- An *IntentFilters* specifies the types of *Intent* that an *activity*, *service*, or *Broadcast Receiver* can respond to.
- If a component does not define an *Intent* filter, it can only be called by explicit *Intents*
- IntentFilters are typically defined via the AndroidManifest.xml file.
- If an *Intents* is send to the Android system, it will determine suitable applications for this *Intents*.
- If several components have been registered for this type of *Intents*, Android offers the user the choice to open one of them. This determination is based on *IntentFilters*.

Intent Filter

- An *Intent Filter* declares the capabilities of a component. It specifies what an *activity* or *service* can do and what types of broadcasts a *Receiver* can handle. It allows the corresponding component to receive *Intents* of the declared type.

- For BroadcastReceiver it is also possible to define them in coding. An IntentFilters is defined by its category, action and data filters. It can also contain additional metadata.

Intent Filter

- Filters are specified for each component in the *AndroidManifest.xml* file

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ...>
  <application android:icon="@drawable/app_notes" ... >

      <activity android:name="NoteEditor"
            android:theme="@android:style/Theme.Light"
            android:label="@string/title_note" >
        <intent-filter android:label="@string/resolve_edit">
            <action android:name="android.intent.action.VIEW" />
            <action android:name="android.intent.action.EDIT" />
            <action android:name="com.android.notepad.action.EDIT_NOTE" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.INSERT" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
        </intent-filter>
      </activity>
```

Defining intent filters

1. Defining intent filter

You can register your own components via intent filters. If a component does not define one, it can only be called by explicit intent. The key for this registration is that your component registers for the correct action, mime-type and specifies the correct meta-data

2. Example: Register an activity as Browser

The following code will register an *Activity* for the *Intent* which is triggered when someone wants to open a webpage.

```
<activity android:name=".BrowserActivitiy" android:label="@string/app_name">
<intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http"/>
</intent-filter>
</activity>
```

Defining intent filters

3. Example: Register an activity for the share intent

- The following example registers an activity for the ACTION_SEND intent. It declares itself only relevant for the *text/plain* mime type

```
<activity android:name=".ActivityTest" android:label="@string/app_name" >
<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 Resolution

Intent Resolution:

"The intent resolution mechanism basically revolves around matching an Intent against all of the <intent-filter> descriptions in the installed application packages."

- When Intents are issued, Android looks for the most appropriate way of responding to the request.
- The decision of what to execute is based on how descriptive the call is based on types of intent.

Intent Resolution