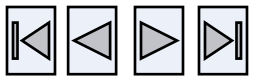
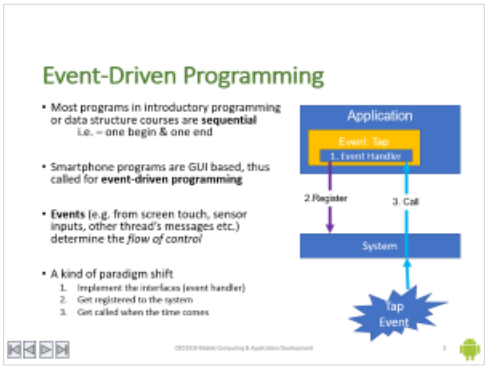
Android UI Dynamics
&
Activity Lifecycle

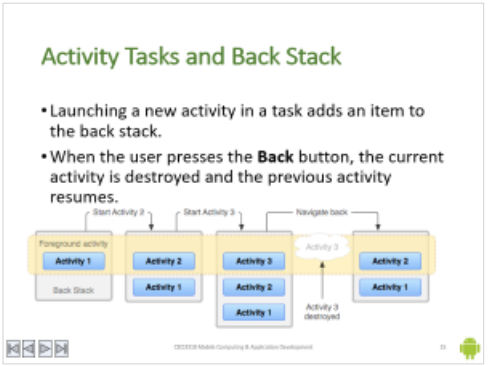
CSCI3310 Mobile Computing & Application Development

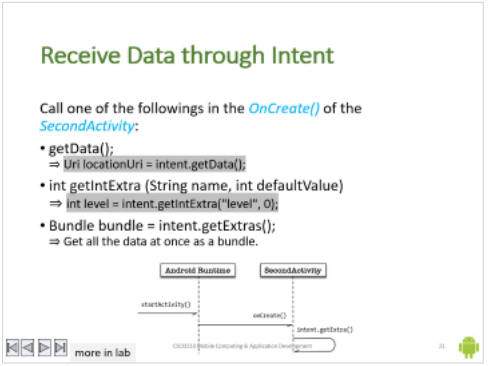


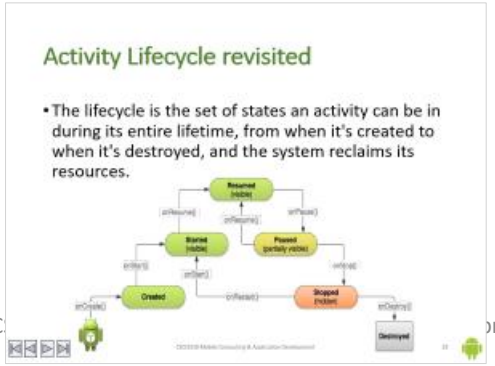
Outline

- Event Driven Programming
- Navigation and Activity Stack
- More on Intent
- Activity Lifecycle and Data Persistence



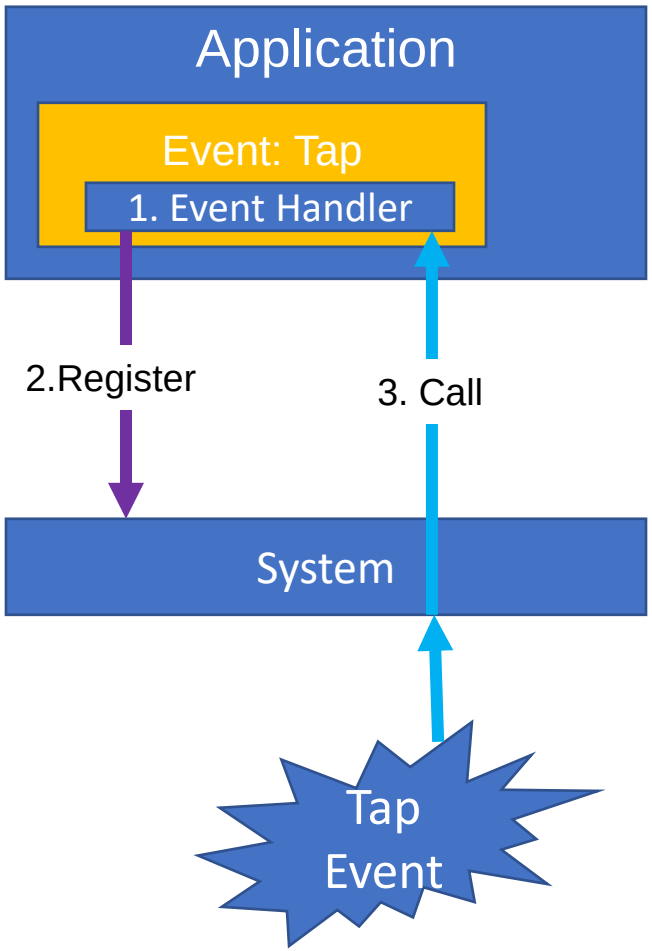






Event-Driven Programming

- Most programs in introductory programming or data structure courses are **sequential**
  i.e. – one begin & one end
- Smartphone programs are GUI based, thus called for **event-driven programming**
- **Events** (e.g. from screen touch, sensor inputs, other thread's messages etc.) determine the *flow of control*
- A kind of paradigm shift
  1. Implement the interfaces (event handler)
  2. Get registered to the system
  3. Get called when the time comes



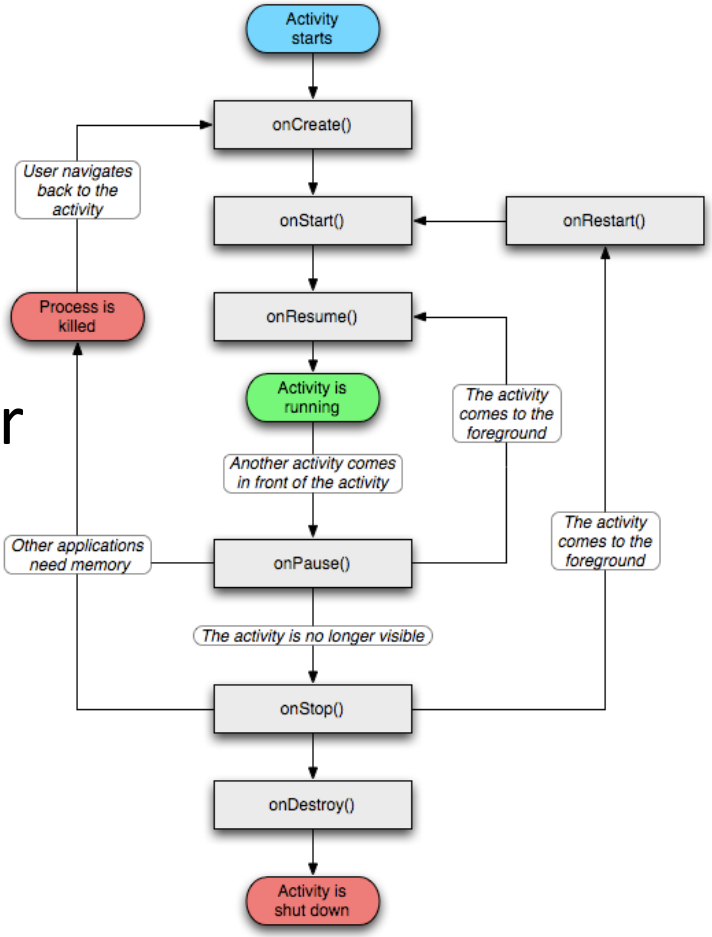
Event-Driven Programming

Infinite Loop?

Yes, we call it the **Event Loop**

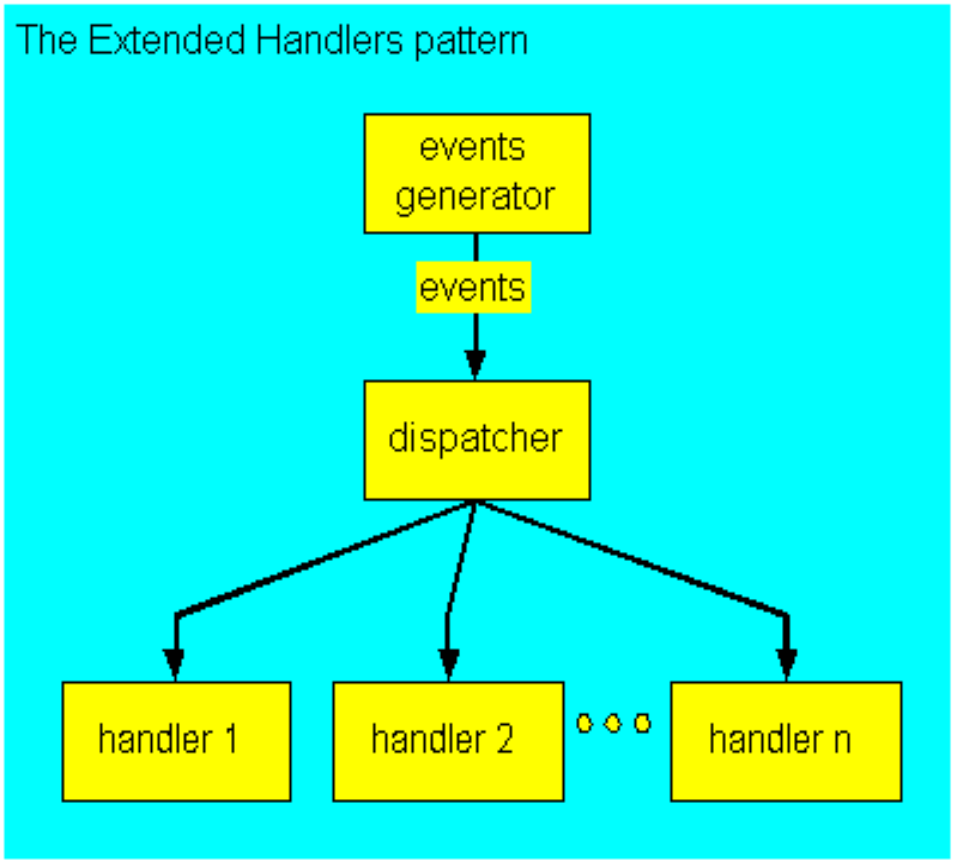
So, how does it ever quit?

Events from User or OS to trigger



Event-Dispatching in the UI thread

- Has a main loop divided into 2 sections
  - Event selection (detection)
  - Event handling
- On knowing the event, application will dispatch it to corresponding event handler



Event Handler in XML

1. Link a reference of the **UI Control** to the **View** system within the XML: use **android:onClick** attribute in the layout:

```
public class MainActivity extends AppCompatActivity {
  Button b1;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  public void updateText(View v) {
    TextView t1 = (TextView) findViewById(R.id.label);
    t1.setTextSize(30);
  }
}
```

Event Handling in Code

1. Link a reference of the **UI Control** to the **View** system
2. Set event handler (listener) from **UI Control**

```
public class MainActivity extends AppCompatActivity {
  Button b1;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

1   b1 = (Button) findViewById(R.id.ok);
2   b1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        TextView t1 = (TextView) findViewById(R.id.label);
        t1.setTextSize(30);
      }
    });
  }
}
```

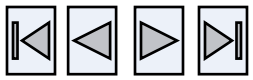
`public void setOnClickListener (View.OnClickListener l)`
- accept a reference to an interface.

Event Handling in Code

```
public interface View.OnClickListener {
    void onClick(View v);
}

... Within OnCreate()
    b1 = (Button) findViewById(R.id.ok);

    // Reference to an interface is a class implements that interface
    View.OnClickListener listener = new View.OnClickListener()
    {
      @Override
      public void onClick(View v) {
         updateText(v);
      }
    }
    b1.setOnClickListener(listener);
...
   public void updateText(View v) {
      TextView t1 = (TextView) findViewById(R.id.label);
      t1.setTextSize(30);
   }
}
```

`onClick()` is call by system when the View v is "on click", so it is a callback registeration.

Here `listener` is a reference to class that implements `View.OnClickListener`
So it ought to provide an implementation to `onClick()`
In practise, class `listener` shall not be used again, so it can be declared as **an anonymous class** like the previous example



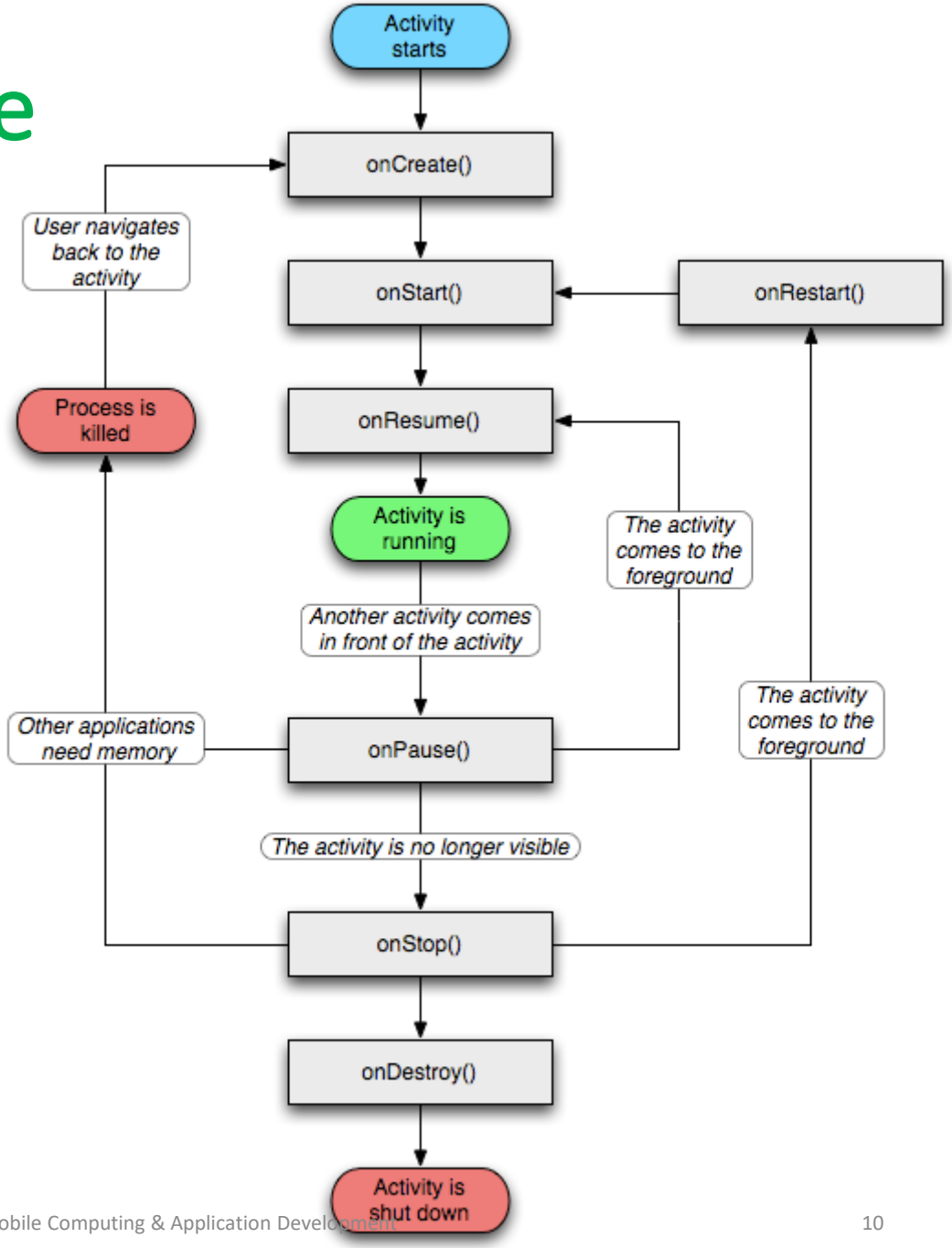
Event Handling in Code

Yet another way to register the onClick callback

```
public class MainActivity extends AppCompatActivity
      implements View.OnClickListener {
  Button b1;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    b1 = (Button) findViewById(R.id.ok);
    b1.setOnClickListener(this);

  }

  @Override
  public void onClick(View v) {
    TextView t1 = (TextView) findViewById(R.id.label);
    t1.setTextSize(30);
  }
}
```

MainActivity implements View.OnClickListener and provide an implementation to onClick() directly.

Activity Lifecycle revisited

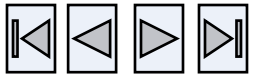


When to start?

```
…
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

- To inform the system which to launch, i.e. a "Launcher" event is **registered** to the OS

```
…

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
…
```

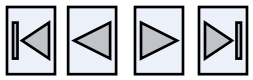


Intents (Revisited)

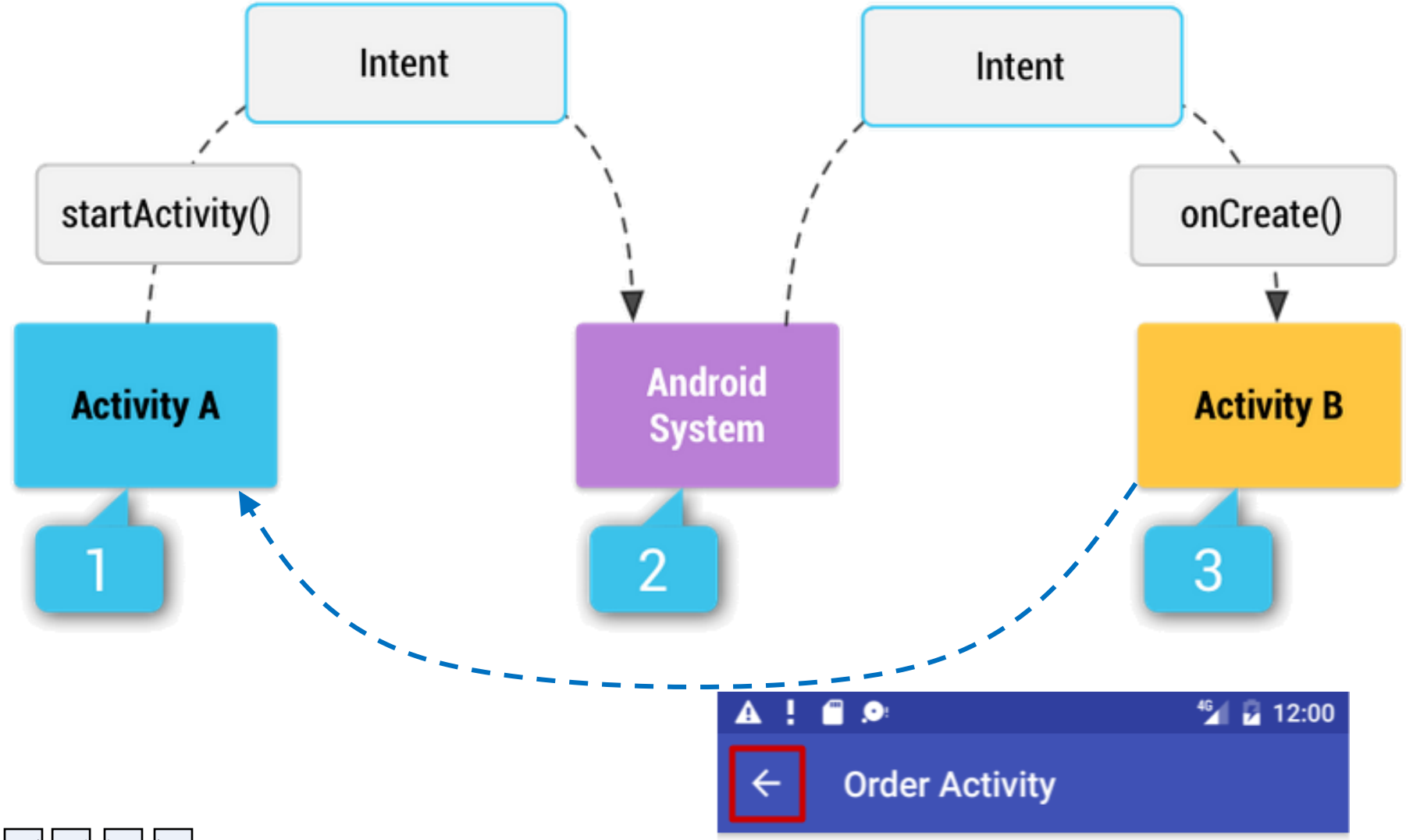
- Asynchronous messages which allow the application to send or receive data from and to other activities or services.
  - e.g. using ACTION_SEND, an application can share text via an intent to Gmail.
- Application **register** themselves via an *IntentFilter*.

- "*android.content.Intent*" is an object which contain information for the receiving component. e.g. if your application calls via an intent a browser, it may send the URL to the browser component.

- contain information for the Android system to determine which component should handle the **Request** (**Event**).

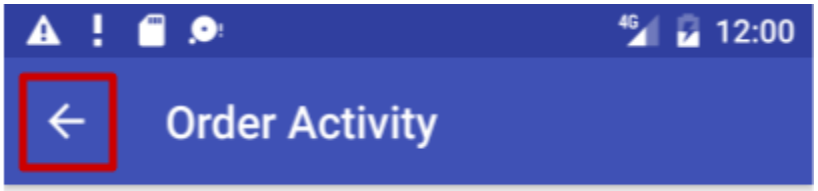


Navigation through Intent



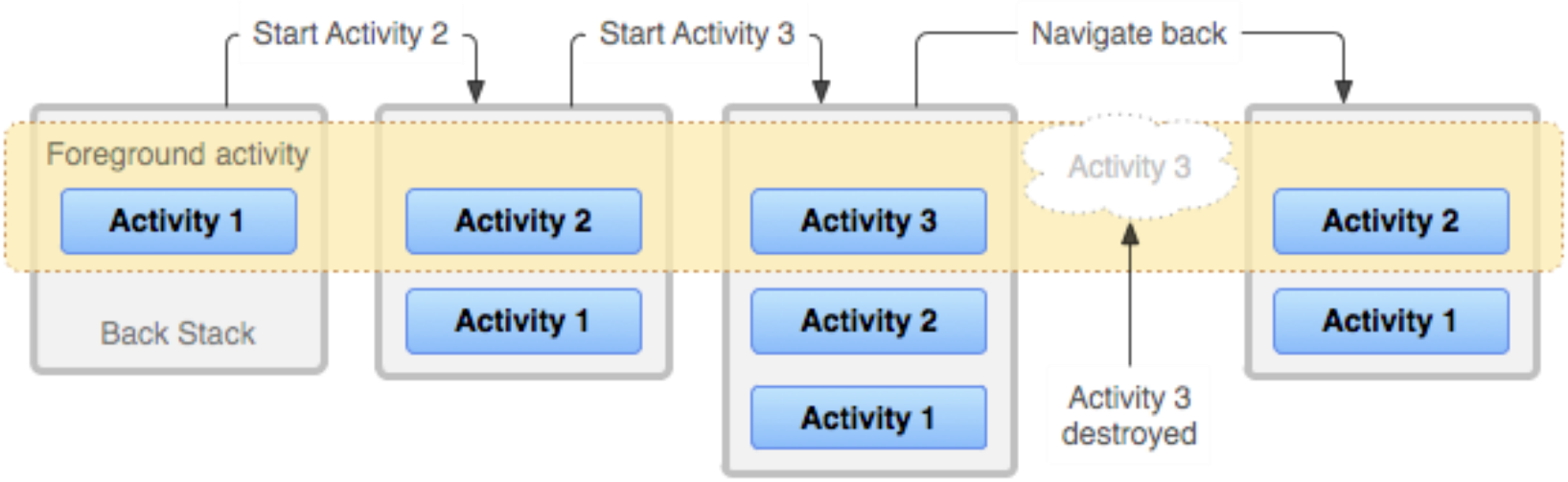
Activity Stack

- When a new Activity is started, the previous **Activity is stopped and pushed** on the Activity back stack
- Last-in-first-out-stack—when the current Activity ends, or the **user presses the Back button,** it is **popped from the stack** and the previous Activity resumes



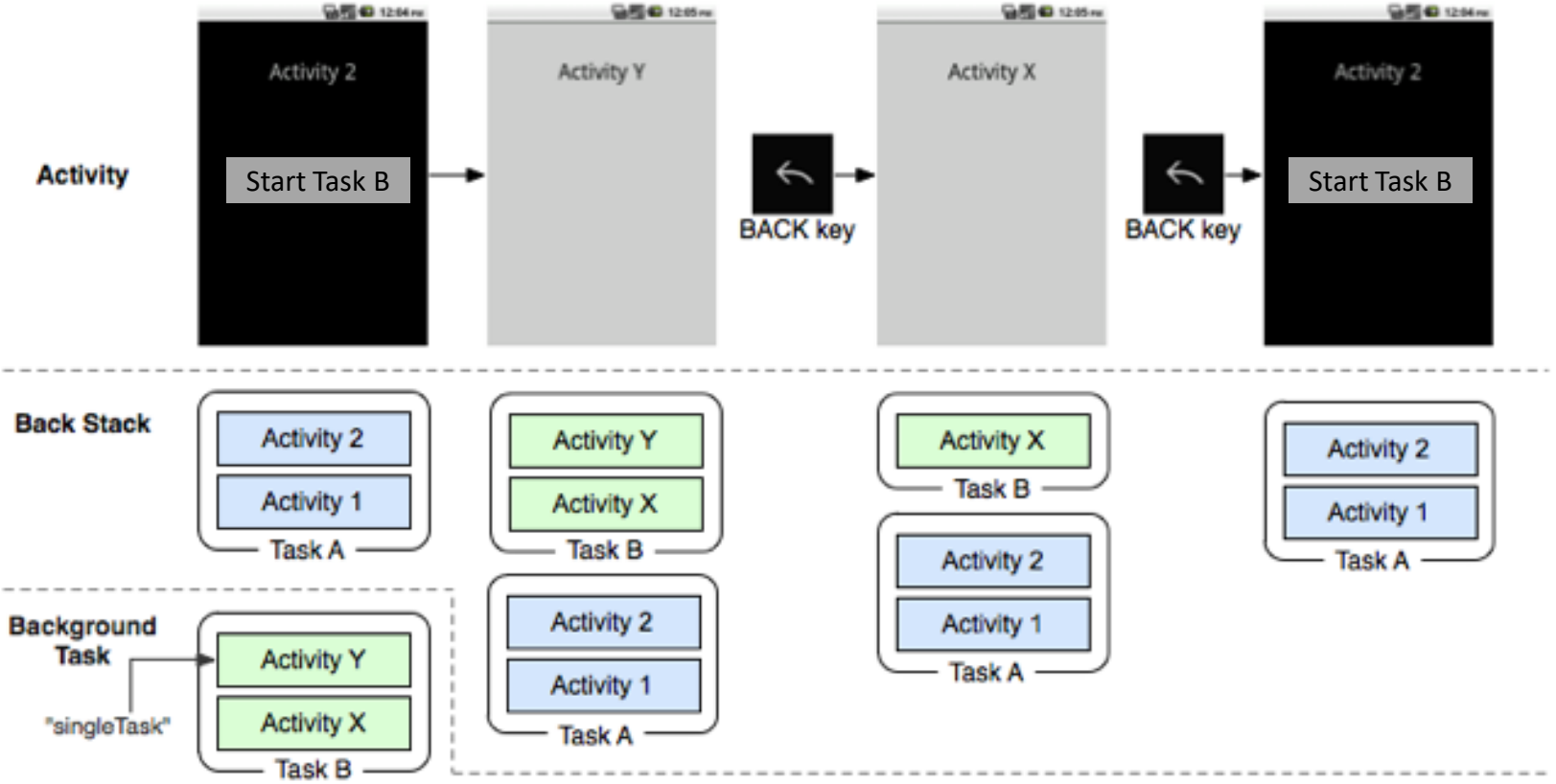
Activity Tasks and Back Stack

- Launching a new activity in a task adds an item to the back stack.
- When the user presses the **Back** button, the current activity is destroyed and the previous activity resumes.



Task as a cohesive unit

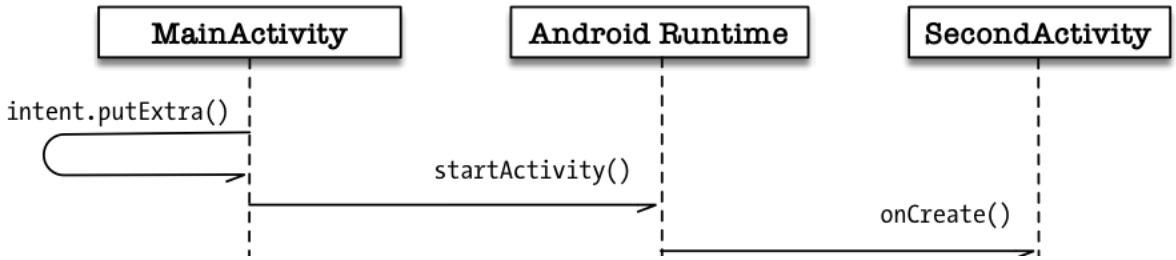
- A task contains more activities, it's manipulated as a whole



Explicit & Implicit Intents

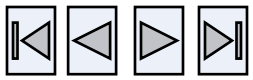
- Explicit intents : designate target component by its name for app internal use

```
Intent expIntent = new Intent(getBaseContext(), SecondActivity.class);
startActivity(expIntent);
```



- Implicit intents : no name, used for activating components in other applications

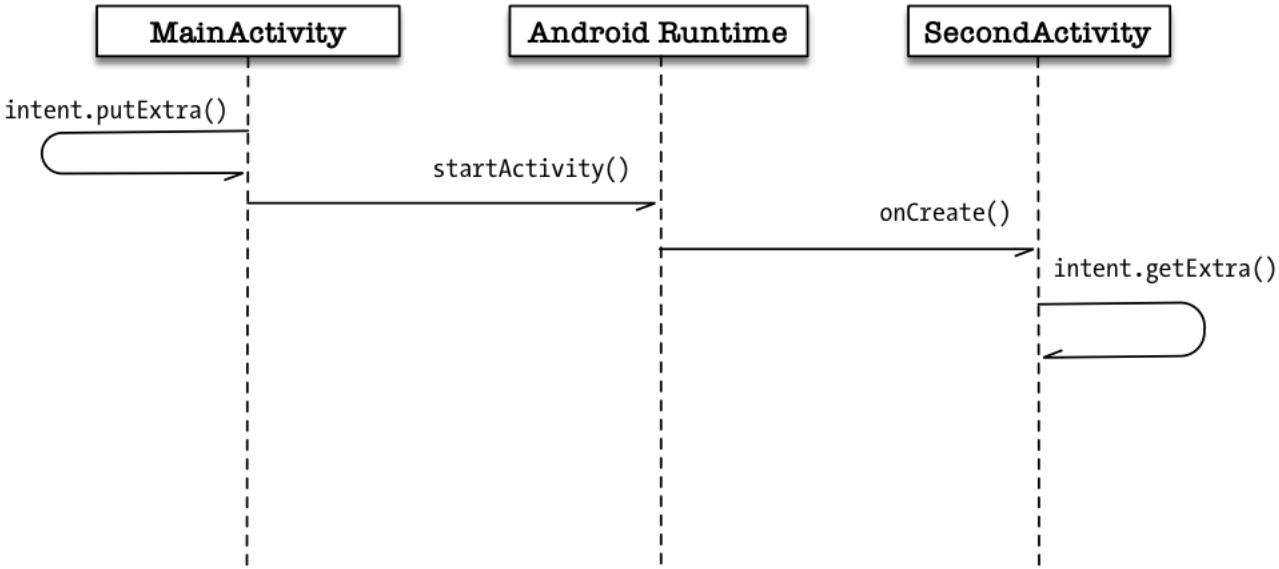
```
Intent impIntent = new Intent(Intent.ACTION_VIEW);
impIntent.setData(Uri.parse("http://www.cse.cuhk.edu.hk"));
startActivity(impIntent);
```



more in lab

Carry data via Intents

- Android Intents can do so much more than just just simply activate or launch another Activity in your app, it can also carry data.



more in lab

Send Data through Intent

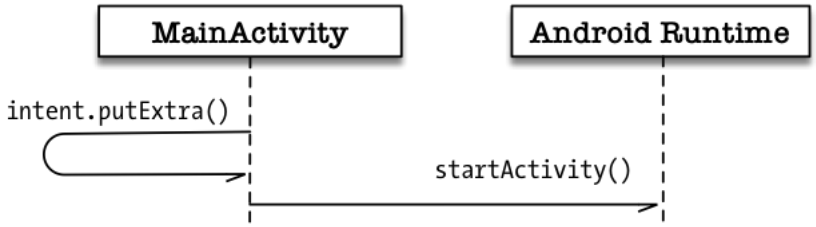
Two types of sending data with intents

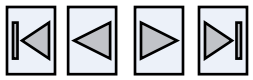
- Data—one piece of information whose data location can be represented by an URI

```
// A web page URL
intent.setData(Uri.parse("http://www.google.com"));
```

- Extras—one or more pieces of information as a collection of key-value pairs in a Bundle

```
intent.putExtra("level", 406);
```





more in lab

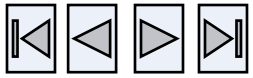
More Data to send

- if lots of data to be sent or received,
  - first create a bundle and pass the bundle.

```
putExtras(bundle);
```

- As a whole with Activity:

```
public static final String EXTRA_MESSAGE_KEY =
   "com.example.android.twoactivities.extra.MESSAGE";

Intent intent = new Intent(this, SecondActivity.class);
String message = "Hello Activity!";
intent.putExtra(EXTRA_MESSAGE_KEY, message);
startActivity(intent);
```

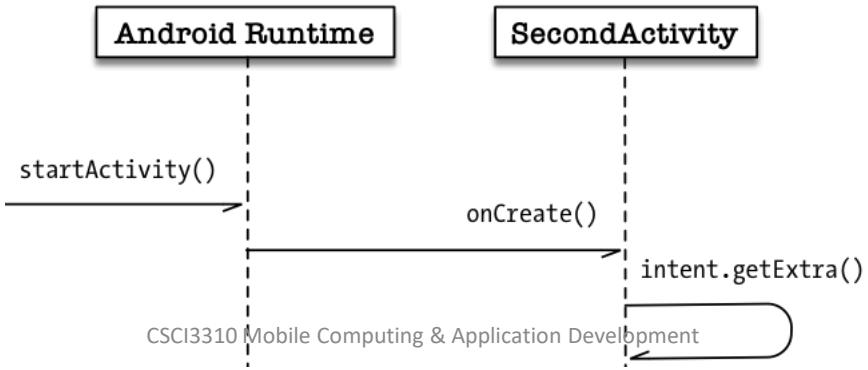


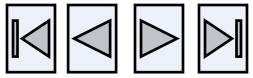
more in lab

Receive Data through Intent

Call one of the followings in the *OnCreate()* of the *SecondActivity*:

- getData();
  ⇒ Uri locationUri = intent.getData();
- int getIntExtra (String name, int defaultValue)
  ⇒ int level = intent.getIntExtra("level", 0);
- Bundle bundle = intent.getExtras();
  ⇒ Get all the data at once as a bundle.



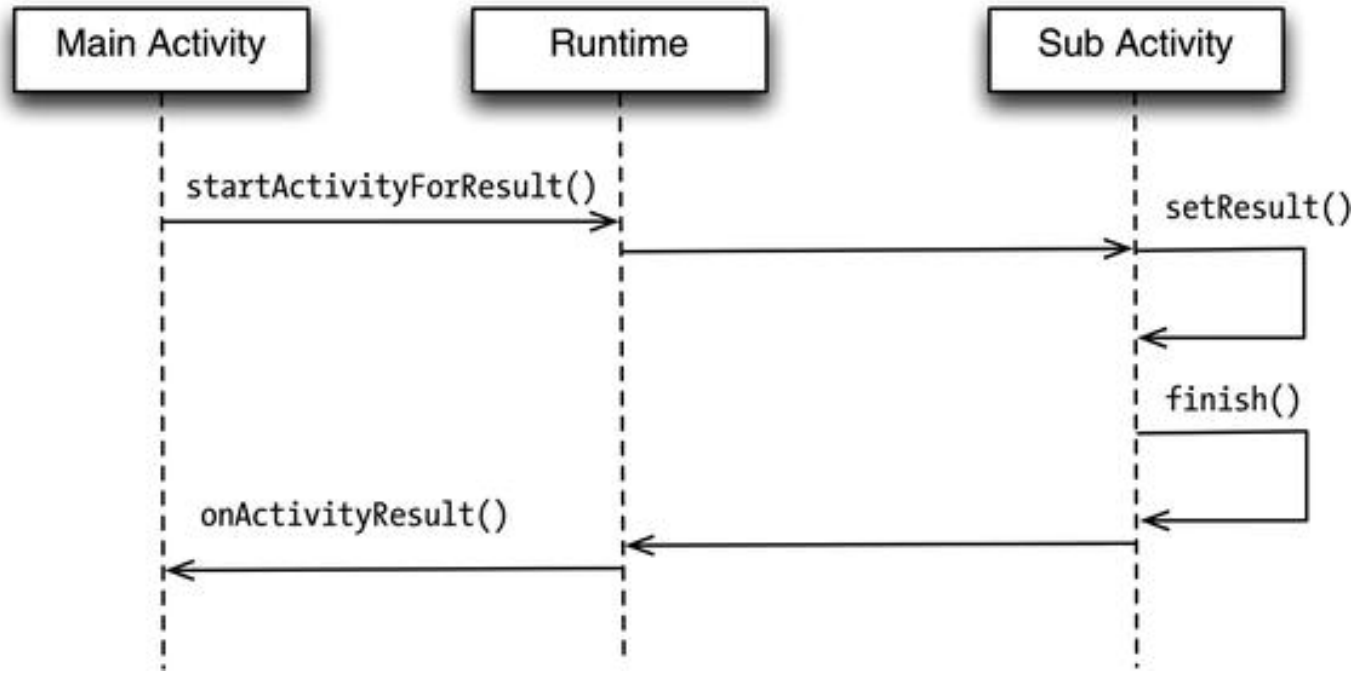


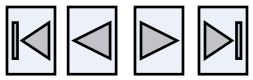
more in lab

Receive Data through Intent

- With Activity request data return, we use:

```
startActivityForResult(intent,requestCode);
```

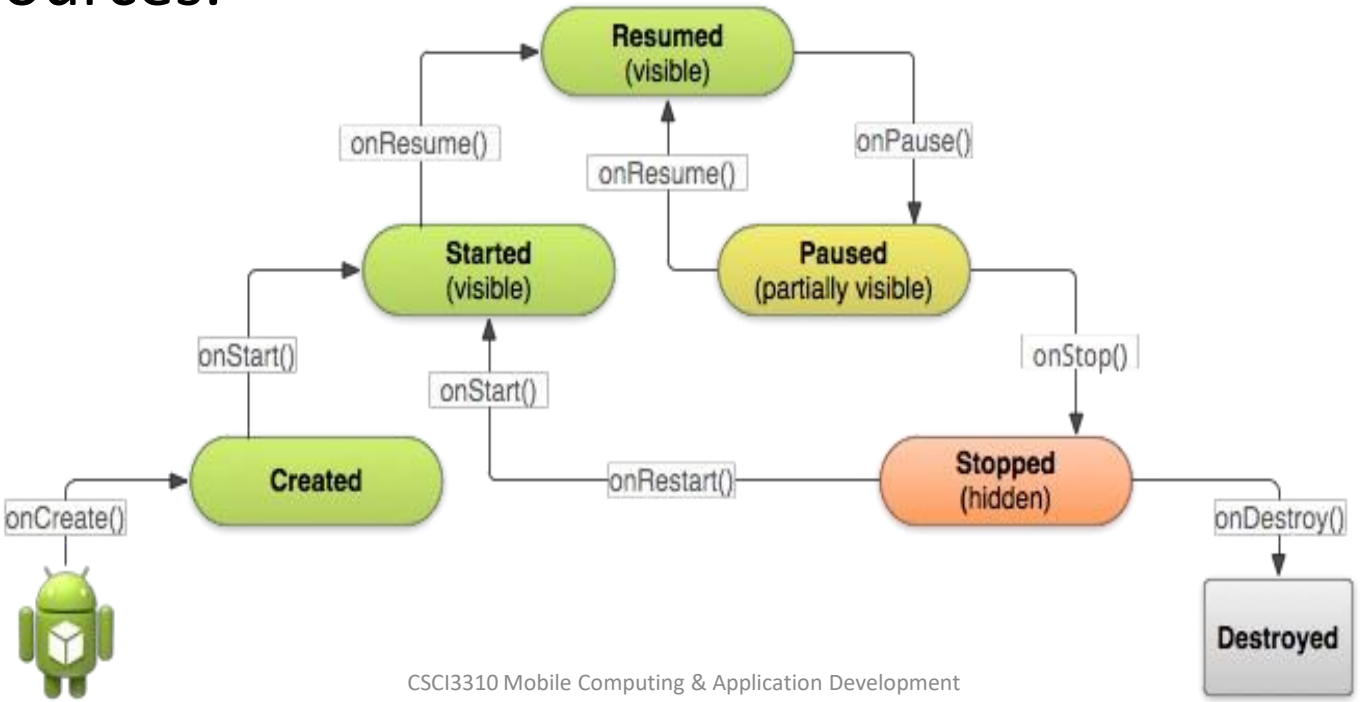




more in lab

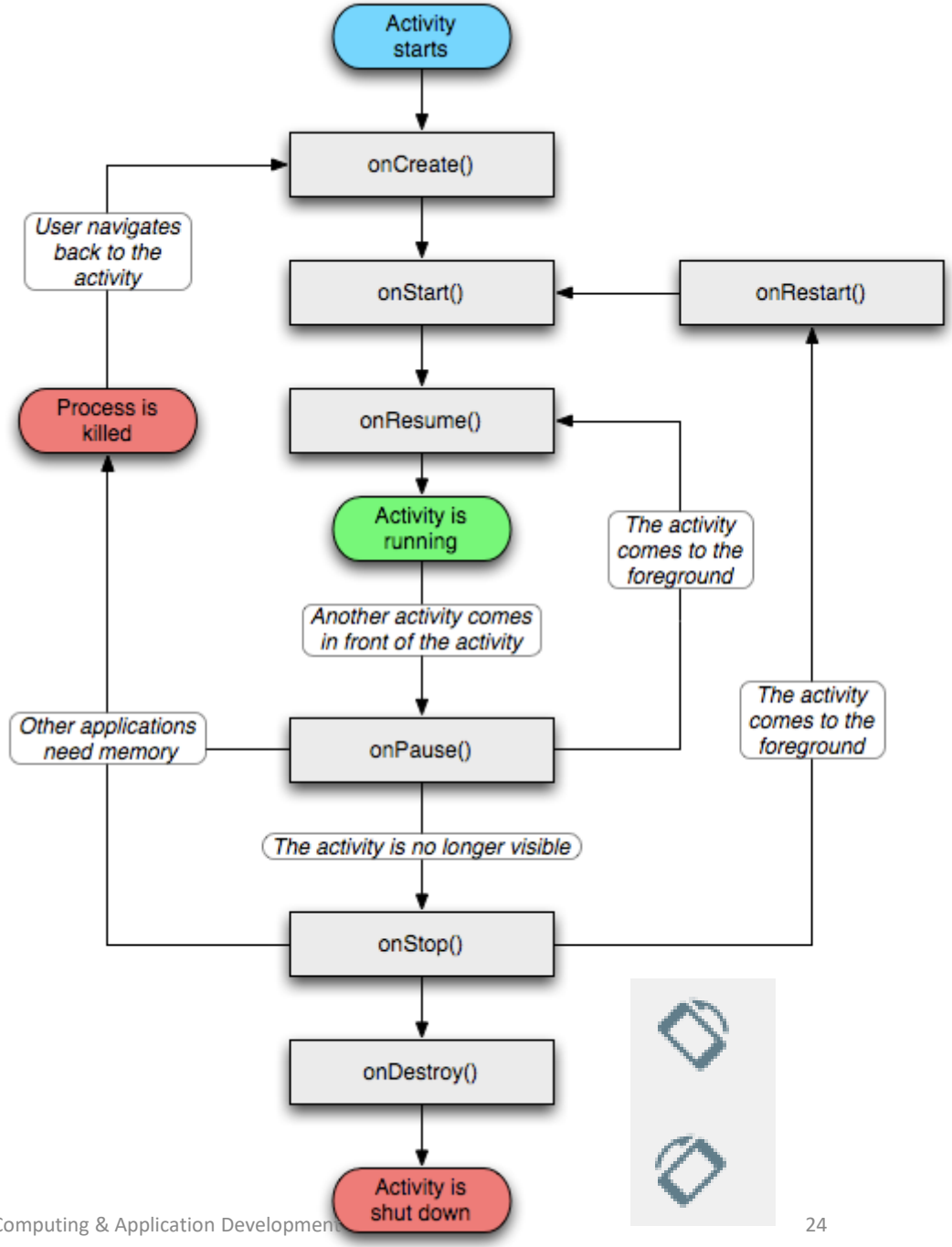
Activity Lifecycle revisited

- The lifecycle is the set of states an activity can be in during its entire lifetime, from when it's created to when it's destroyed, and the system reclaims its resources.



New config

- Rotating the device causes ***configuration change***.
- Implies **recreation** of the current Activity

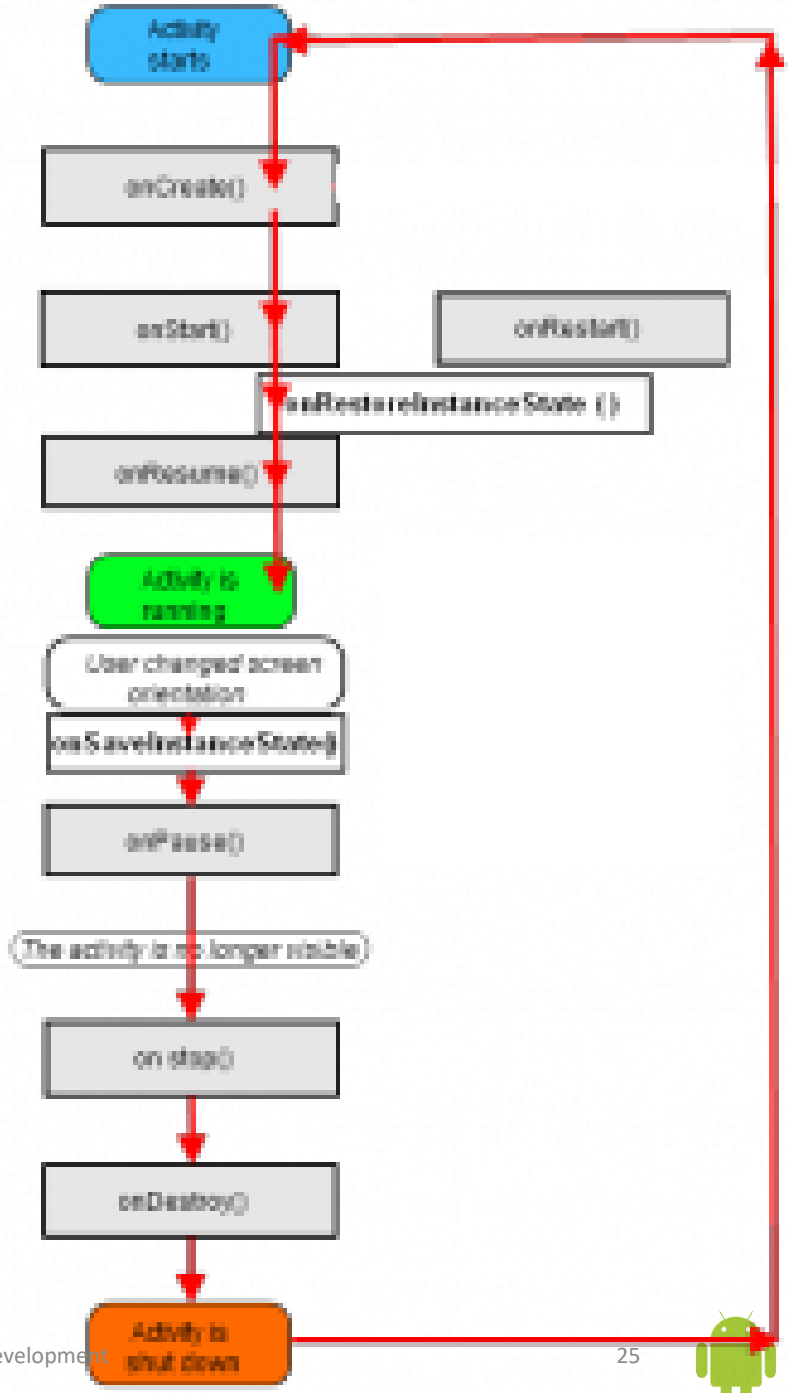


InstanceState

Simple state saving / restoring via callback:

- **`onRestoreInstanceState`** &
- **`onSaveInstanceState`**

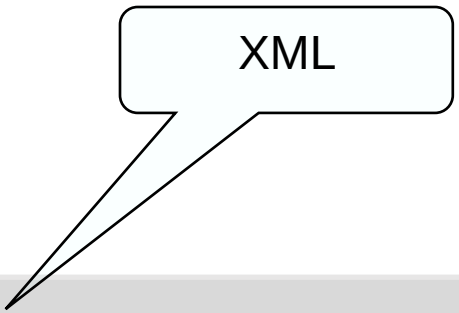
Note that the InstanceState will be lost on quitting the app



Android Data Storage

- four methods for data accessing：
  - SharedPreferences
    - a lightweight mechanism to **store and retrieve key-value pairs** of primitive data types.
    - used to store application preferences, such as a default greeting
  - File (internal, external, or cache)
    - Use java.io.* to read/write data.
  - SQLite database
    - a lightweight transactional database engine
  - ContentProvider
    - An interface used between applications.
    - server application that hosts the data manages it through basic create, read, update, and delete (CRUD) operations.

SharedPreferences



- Create

```
SharedPreferences settings
  = this.getSharedPreferences("Demo", MODE_PRIVATE);
SharedPreferences.Editor
editor = settings.edit();
```

- To add data: <String Key, String Value>

```
editor.putString("name", "value");
editor.commit();
```

- Using Key to get value

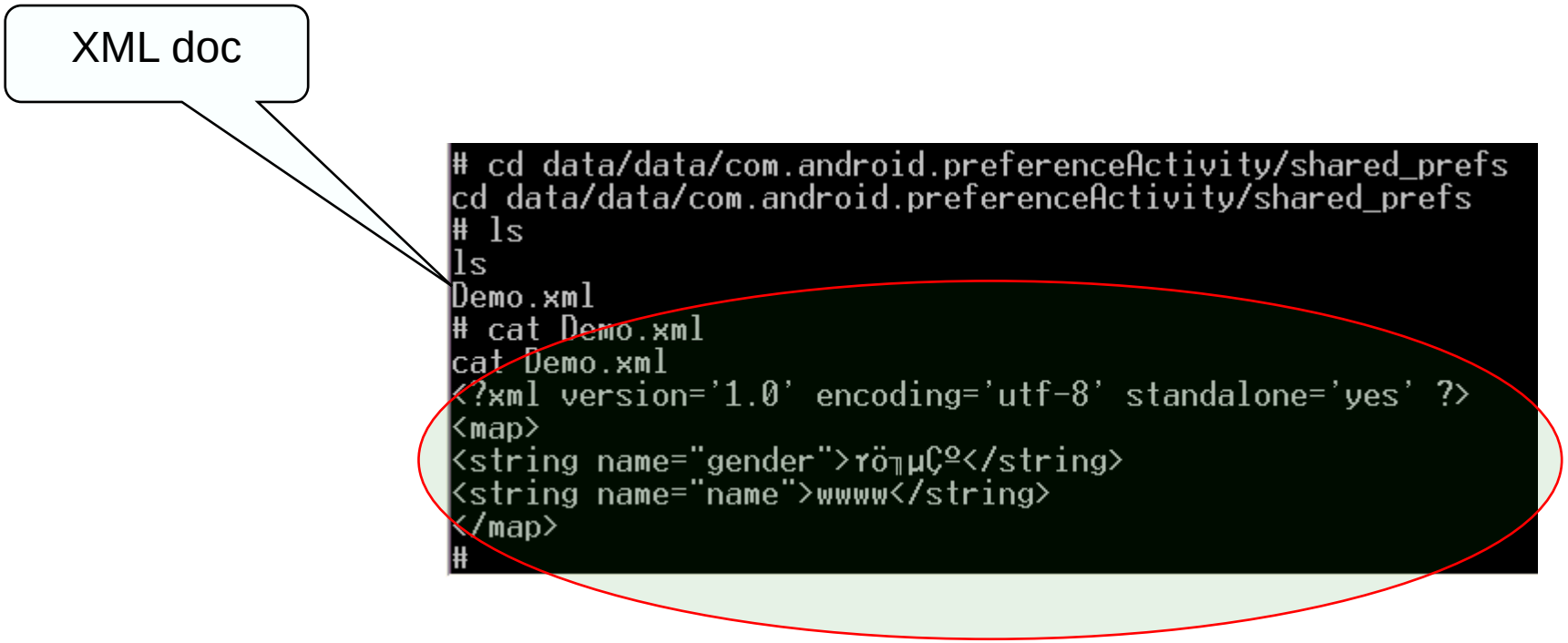
```
String str = settings.getString("name", "");
```

- clear

```
editor.clear().commit();
```

SharedPreferences

- To add data to XML file
  - Directory: /data/data/<package_name>/shared_prefs/*.xml



File access

- Store data to file
- using **java.io.* to read/write file**
- Only local file can be visited
  - Advantages: can **store large amounts of data**
  - Disadvantages: file format changes or updates may result in significant programming

https://developer.android.com/reference/java/nio/file/Files

Read from file

- Openfile

  `Context.openFileInput(String name)`

  - If failure then throw a FileNotFoundException

```
FileInputStream in = this.openFileInput(“rt.txt”);
 ……
in.close(); //
```

https://developer.android.com/reference/android/content/Context.html#openFileInput(java.lang.String)

Write to file

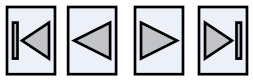
- Open a file to Write

  `Context.openFileOutput(String name,int mode)`

  - If failure then creating a new one
  - Append mode: to add data to file

```
FileOutputStream out = this.openFileOutput("wt.txt",MODE_APPEND);
 ……
out.close();
```

https://developer.android.com/reference/android/content/Context.html#openFileOutput(java.lang.String,%20int)



Static file

- save the file in your project in *res/raw/myFile.txt*, and then open it with `Resources.openRawResource (R.raw.myFile)`

```
InputStreamin =
this.getResources().openRawResource(R.raw.myFile);
 … //reading data
in.close();
```

- It returns an InputStream object that you can use to **read** from the file.

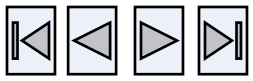
Write a file to SDCard

- To get permission for SDCard r/w in *AndroidManifest.xml:*

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

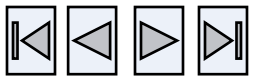
- In **Java**:

```
if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)){
  File sdCardDir = Environment.getExternalStorageDirectory();
  File saveFile = new File(sdCardDir, "itcast.txt");
  FileOutputStream outStream = new FileOutputStream(saveFile);
  outStream.write("AndroidDevelopment".getBytes());
  outStream.close();
}
```



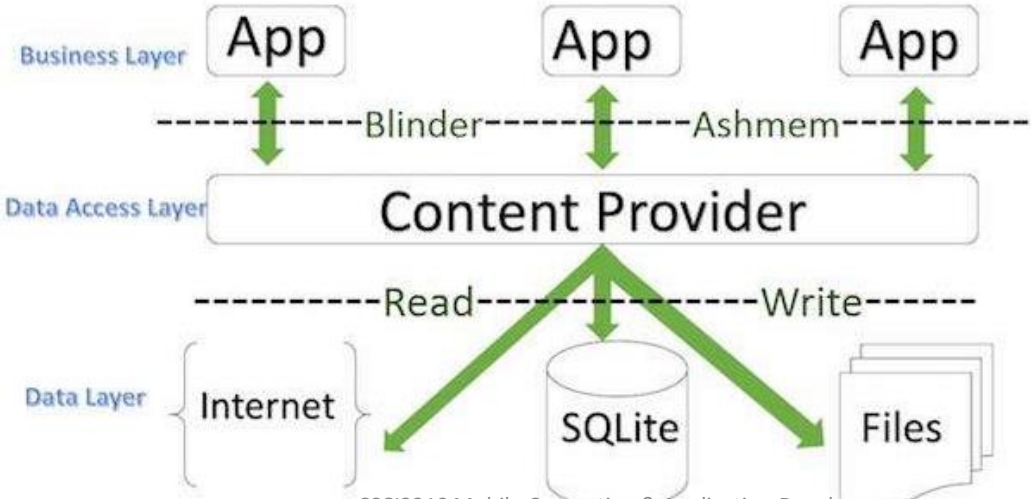
SQLite database

- The content provider API enables full **CRUD** access to the content.
  - Create new records
  - Retrieve one, all, or a limited set of records
  - Update records
  - Delete records if permitted



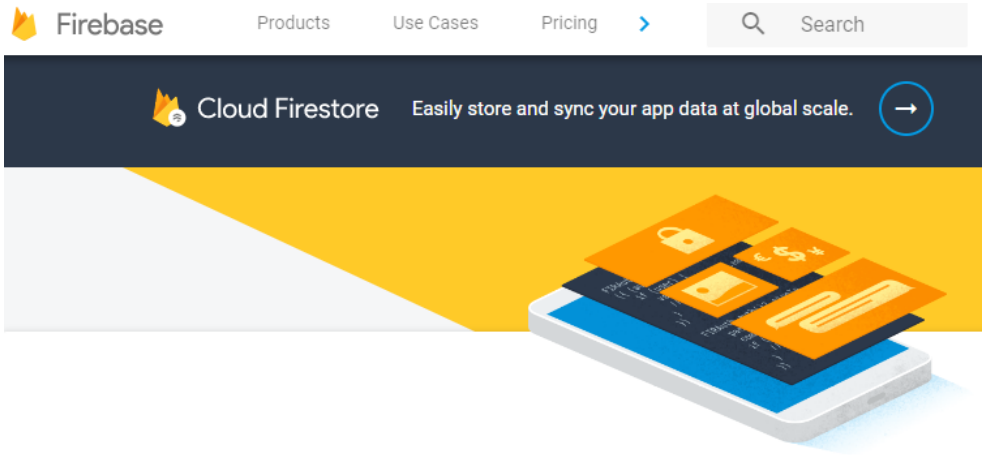
ContentProvider

- An interface used between applications.
- The server application that hosts the data manages it through basic create, read, update, and delete (CRUD) operations.
- The client application uses a similar API, but the Android framework transmits the client's requests to the server.



Firebase

- Backend-as-a-Service (Baas)
- categorized as a NoSQL database program, which stores data in JSON-like documents
- built on Google's infrastructure



Firebase helps you build better mobile apps and grow your business.

Reference

- Input events overview | Android Developers
  https://developer.android.com/guide/topics/ui/ui-events
  https://developer.android.com/reference/android/view/View.OnClickListener.html
- Start another activity | Android Developers
  https://developer.android.com/training/basics/firstapp/starting-activity
- Understand Tasks and Back Stack | Android Developers
  https://developer.android.com/guide/components/activities/tasks-and-back-stack
- Data and file storage overview | Android Developers
  https://developer.android.com/training/data-storage