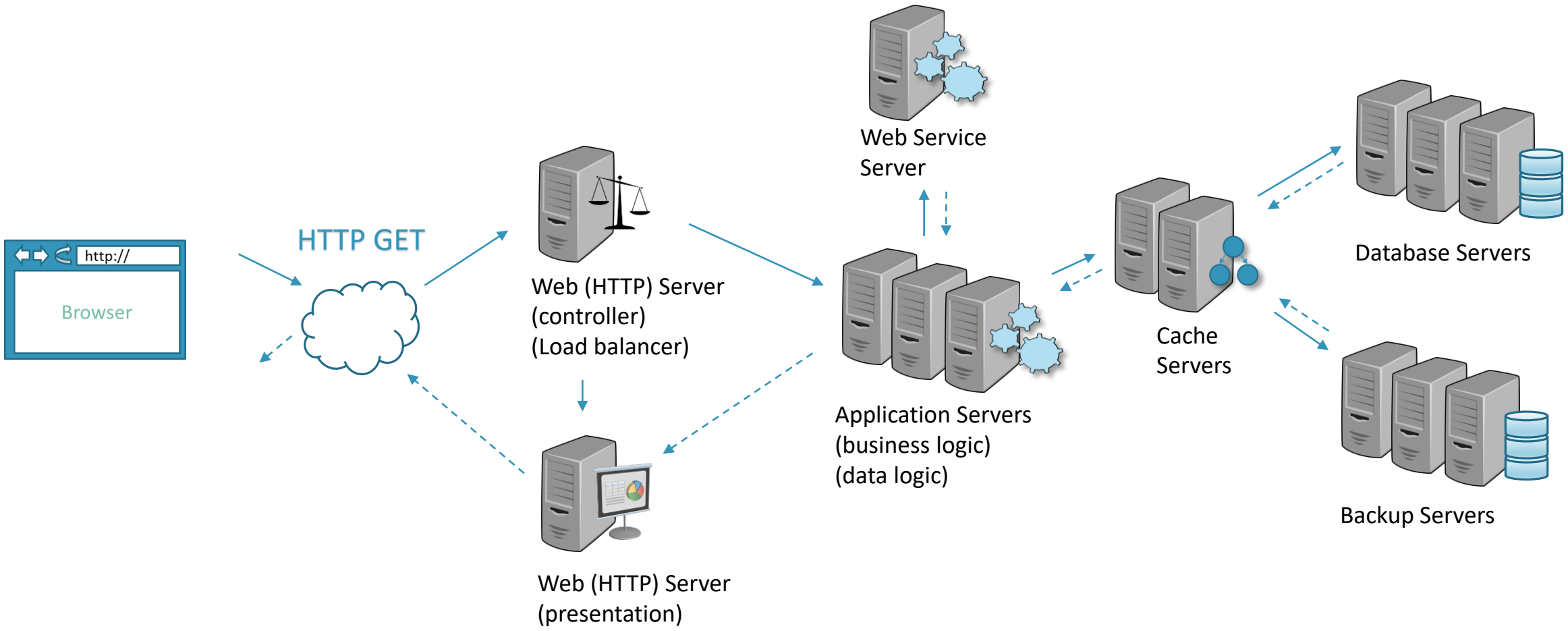
Advanced Programming 2 - Android

DR. ELIAHU KHALASTCHI

2016

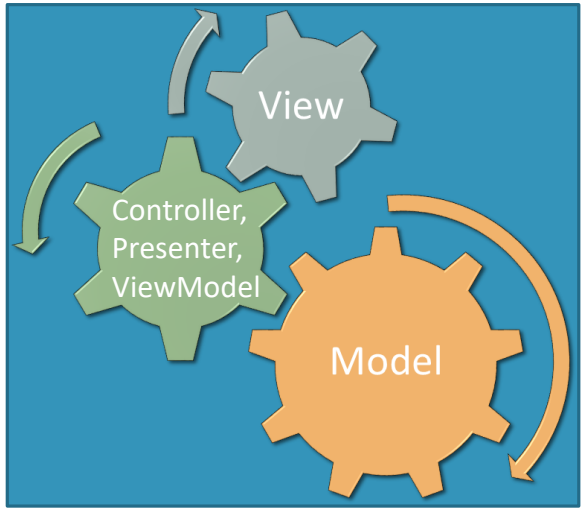
Our Enterprise

Client Side:

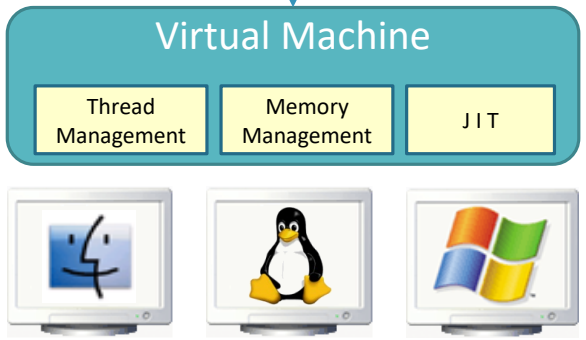


Our Enterprise

Client Source Code



Input



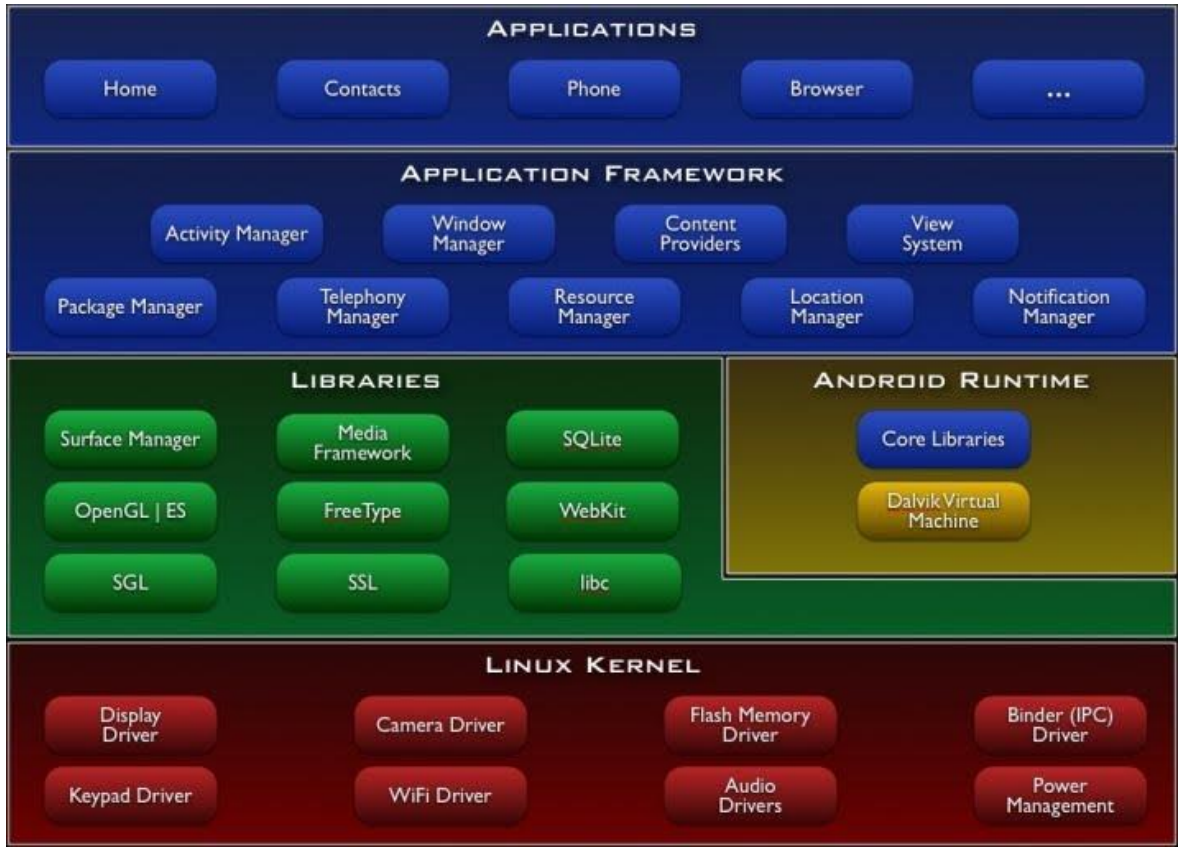
Client Side:

HTTP GET

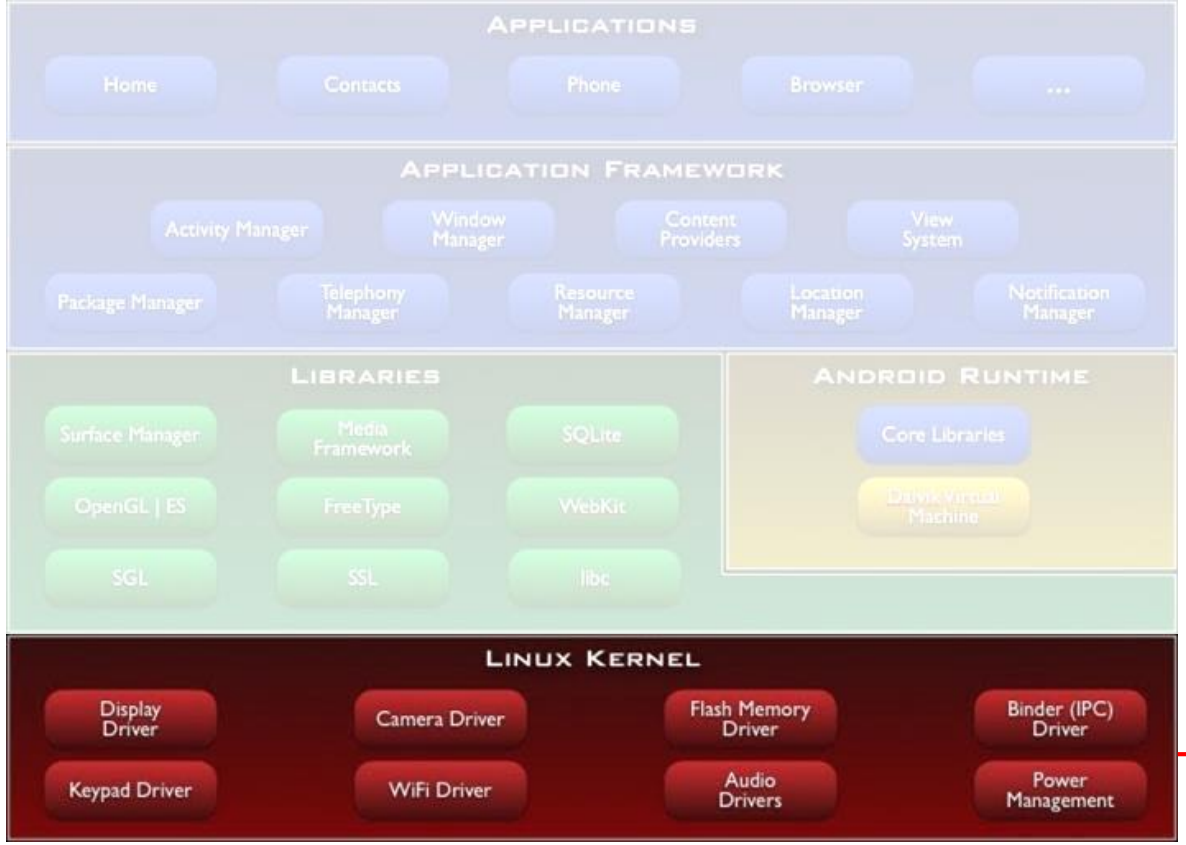
Today’s Lesson…

android

Android Architecture

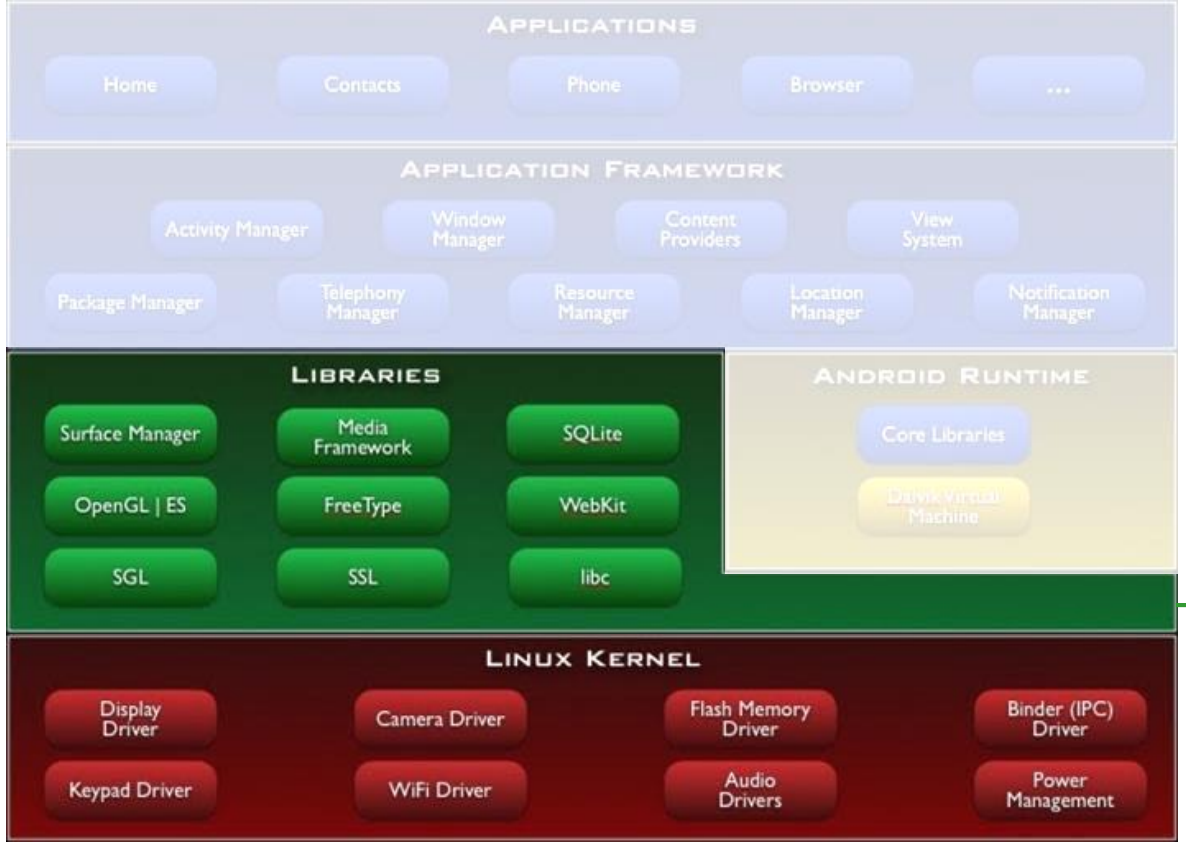


Android Architecture



- Based on Linux Kernel
- Hardware Abstraction Layer (HAL)
- Provides:
  - memory management
  - process management
  - networking

Android Architecture



- Native Libraries
  - written in C/C++, interfaced by Java

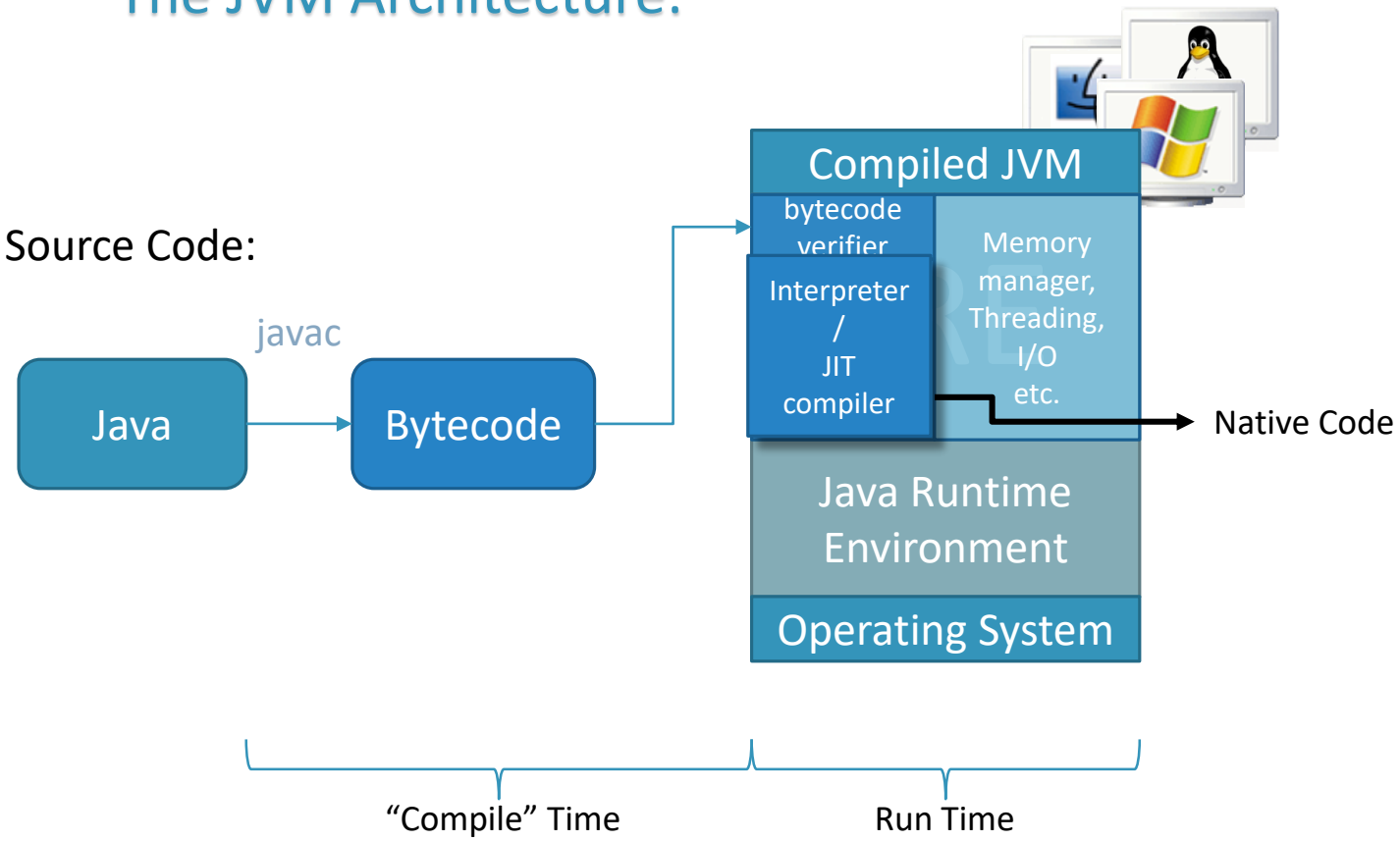
Overview:

- libc
- open-source web browser engine
- SQLite database (storage and sharing app data)
- Play and record audio and video
- SSL internet security
- Surface Manager (display / touch)
- etc

Android Architecture

Android Architecture

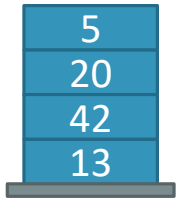
The JVM Architecture:



The JVM is a ***stack based machine***

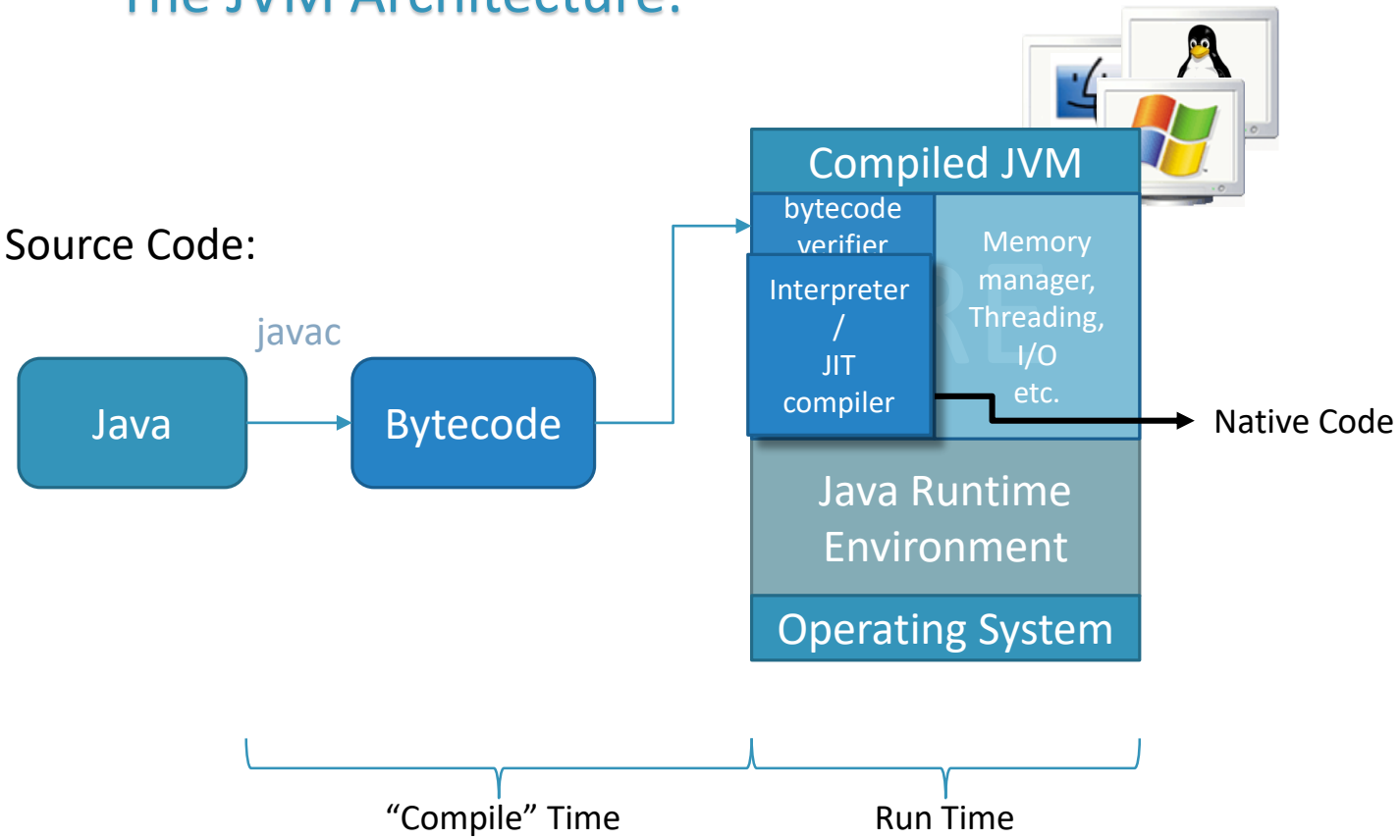
An instruction "5+20" translates to:

1. POP 5
2. POP 20
3. ADD 5, 20, result
4. PUSH result



Android Architecture

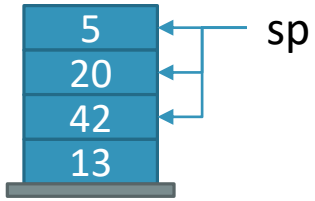
The JVM Architecture:



The JVM is a ***stack based machine***

An instruction "5+20" translates to:

1. POP 5
2. POP 20
3. ADD 5, 20, result
4. PUSH result



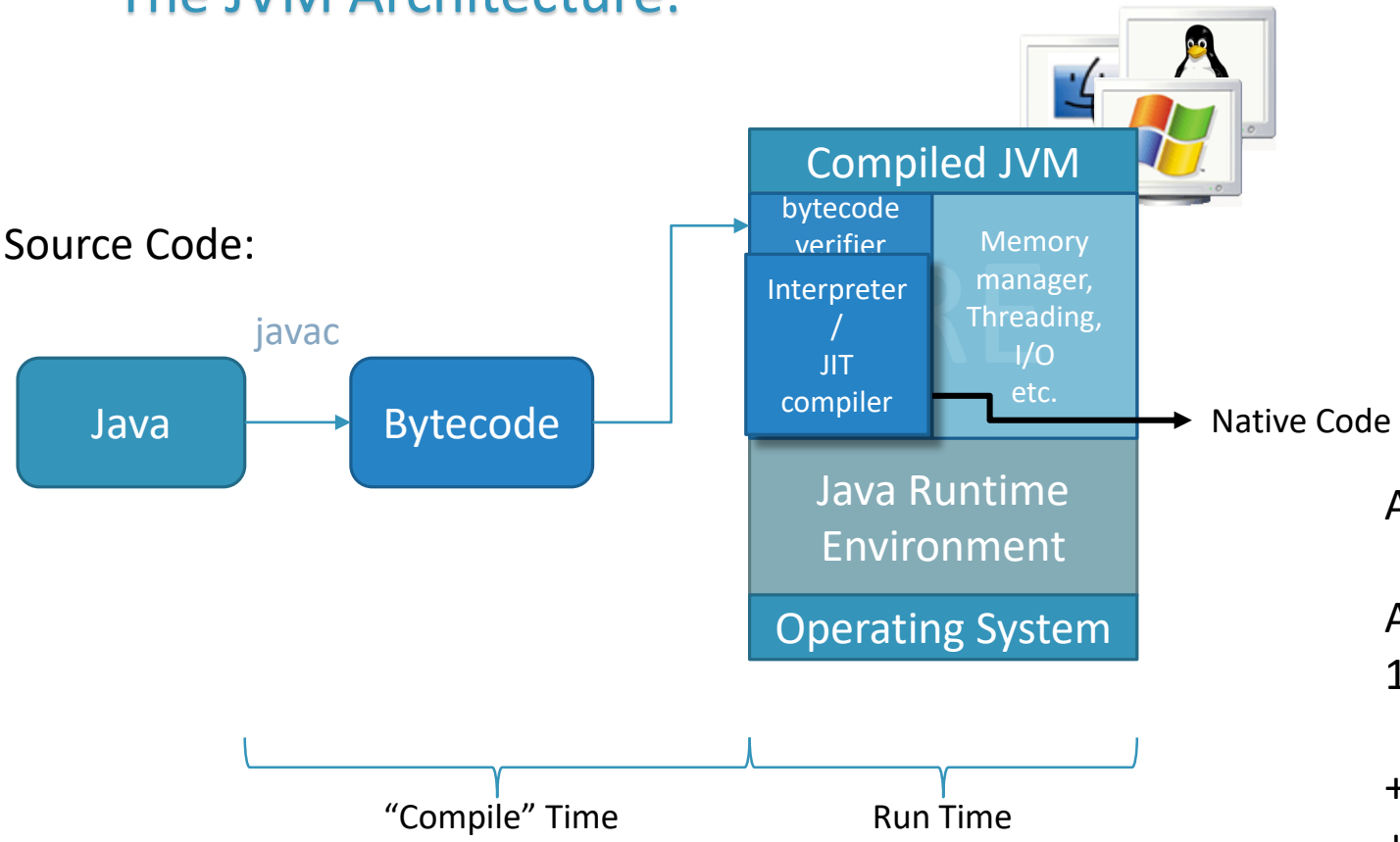
Android Architecture

Register based

Ongoing debate which is better…

The JVM Architecture:

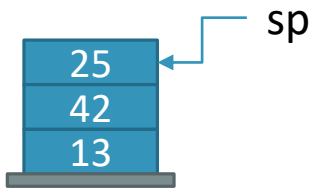
Source Code:



The JVM is a ***stack based machine***

An instruction "5+20" translates to:

1. POP 5
2. POP 20
3. ADD 5, 20, result
4. PUSH result



Another architecture ***register based machine***

An instruction "5+20" translates to:

1. ADD R1, R2, R3

+ Less instructions!
+ Possible optimizations!
- An instruction is more complex

Android Architecture

The Dalvik VM Architecture:

Source Code:

Java → javac → Bytecode (.class) → dx → Dalvik Executable (.dex / .odex)

Register-based

Dalvik VM
- bytecode verifier
- Interpreter / JIT compiler
- Memory manager, Threading, I/O etc.

→ Native Code

Android Runtime Environment

Android

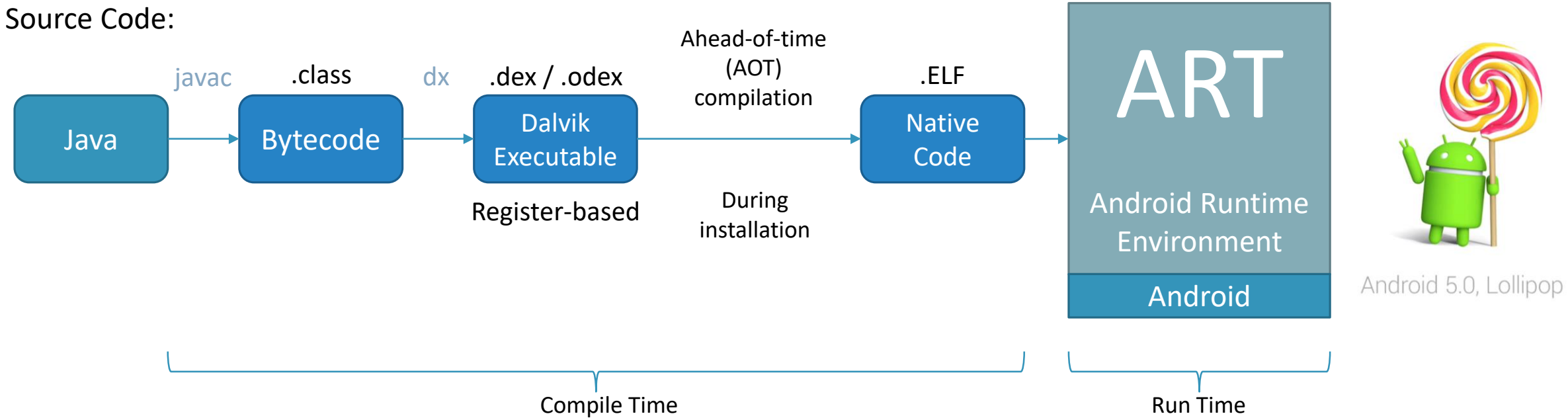
"Compile" Time

Run Time

Android 4.4 KitKat

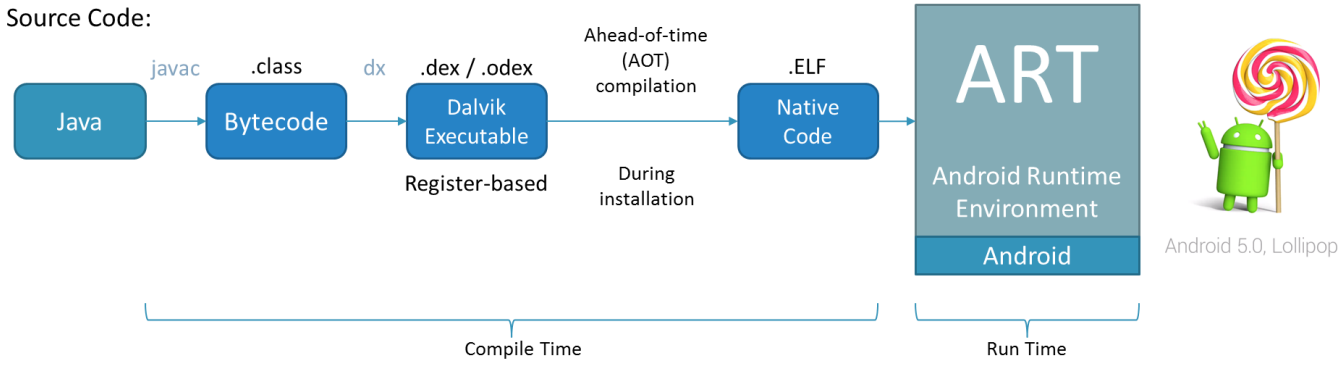
Android Architecture

The ~~Dalvik VM~~ Android Runtime (ART) Architecture:



Android Architecture

The ~~Dalvik VM~~ Android Runtime (ART) Architecture:



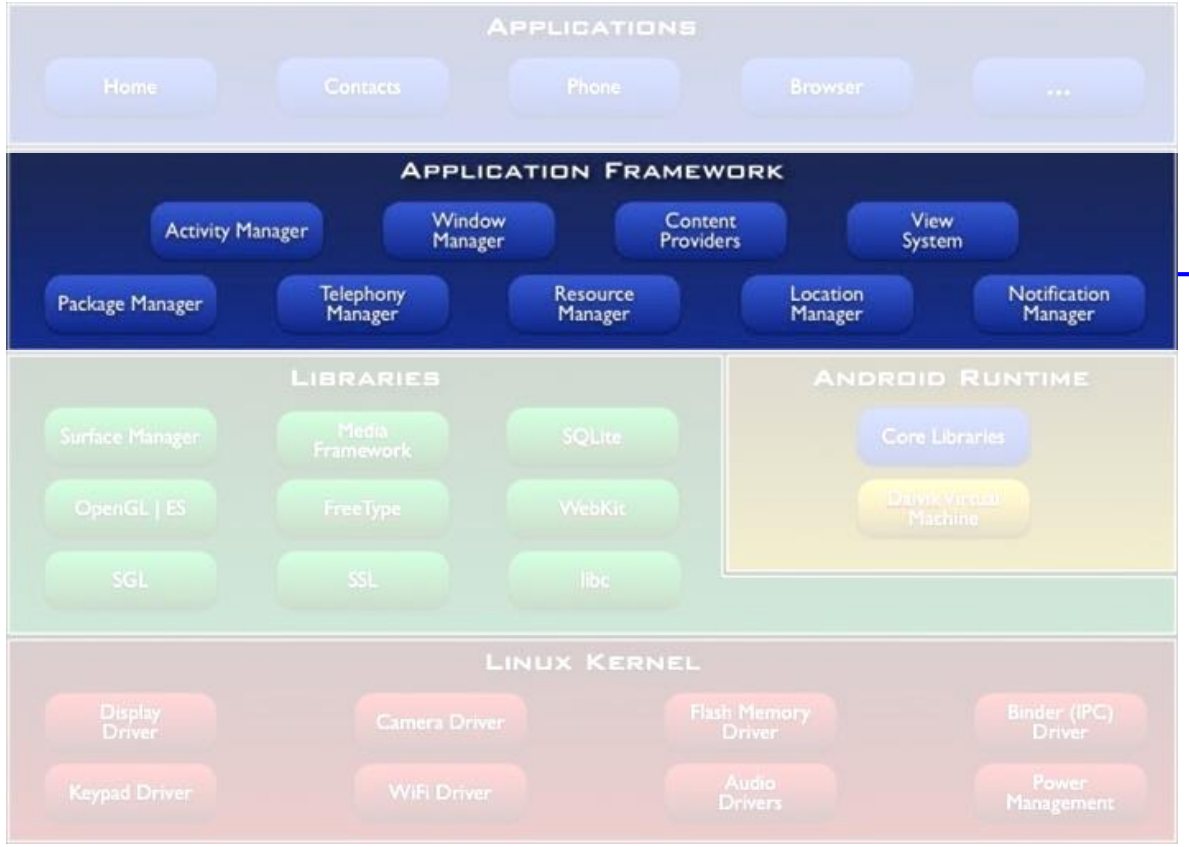
- More time of installation
- More storage for compiled code

+ Faster load!
+ Faster runtime! (no interpretation / JIT overhead)
+ Less RAM usage (no JIT code cache)
+ Less CPU usage (you run the app, not the interpreter)
+ Less power consumption (better for battery life)

At the same time, ART brings improvements in

- Performance
- Garbage collection
- Application debugging and profiling

Android Architecture



Application Framework:

- Written entirely in Java
- The framework for writing apps

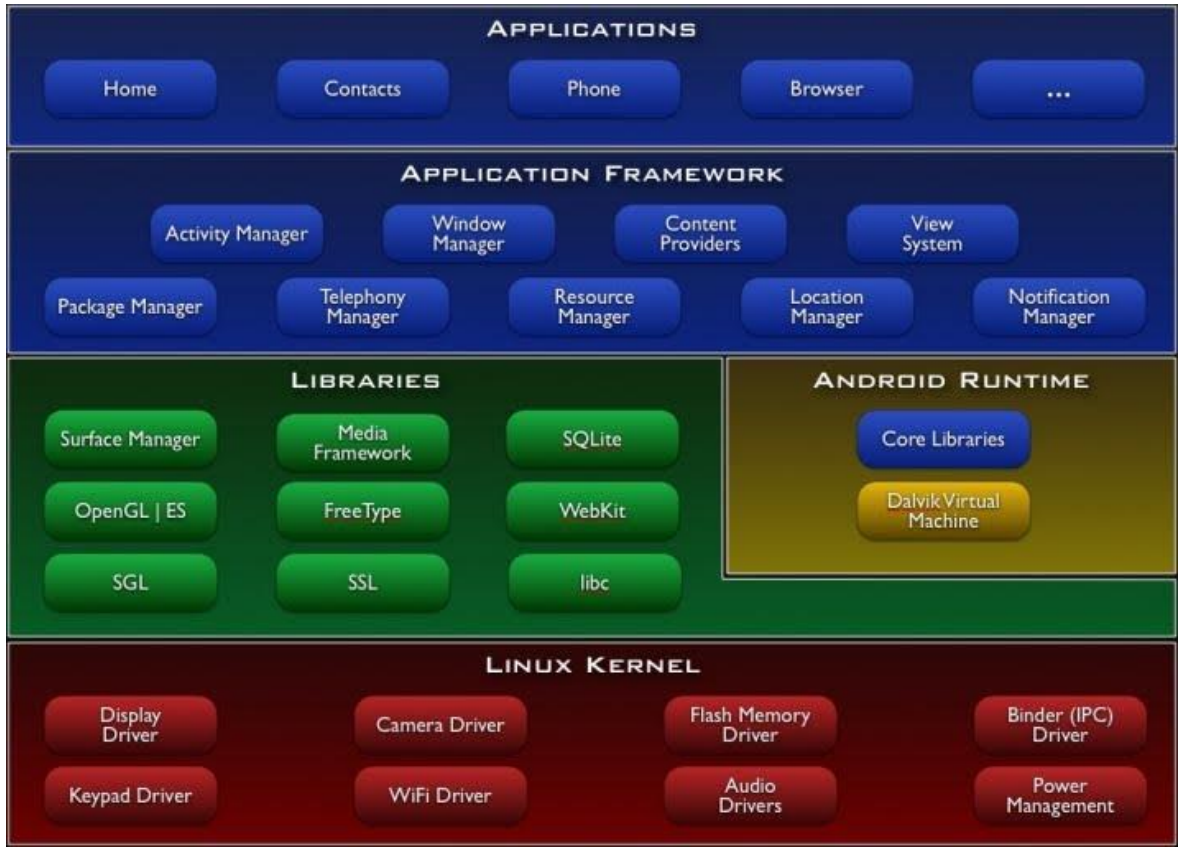
Overview:

- Activity manager – managing the life cycle of apps
- Package manager – managing installed apps
- Content providers – allows apps to share data

Android Architecture

- The (high-level) apps
- Android is shipped with existing apps
- In this layer we write our apps
- Java

Android Architecture



Android app building blocks

- **Activities**
  - A single screen with a user interface
  - An app is made of several independent activates (e.g., inbox, compose mail)
  - Apps can use the activities of other apps (e.g., a camera app may compose an email and share a picture)
  - Implemented as a subclass of Activity

- **Services**
  - A component that runs in the background to perform long-running operations (e.g., play music)
  - No user interface
  - An activity can invoke services
  - Implemented as a subclass of Service

Android app building blocks

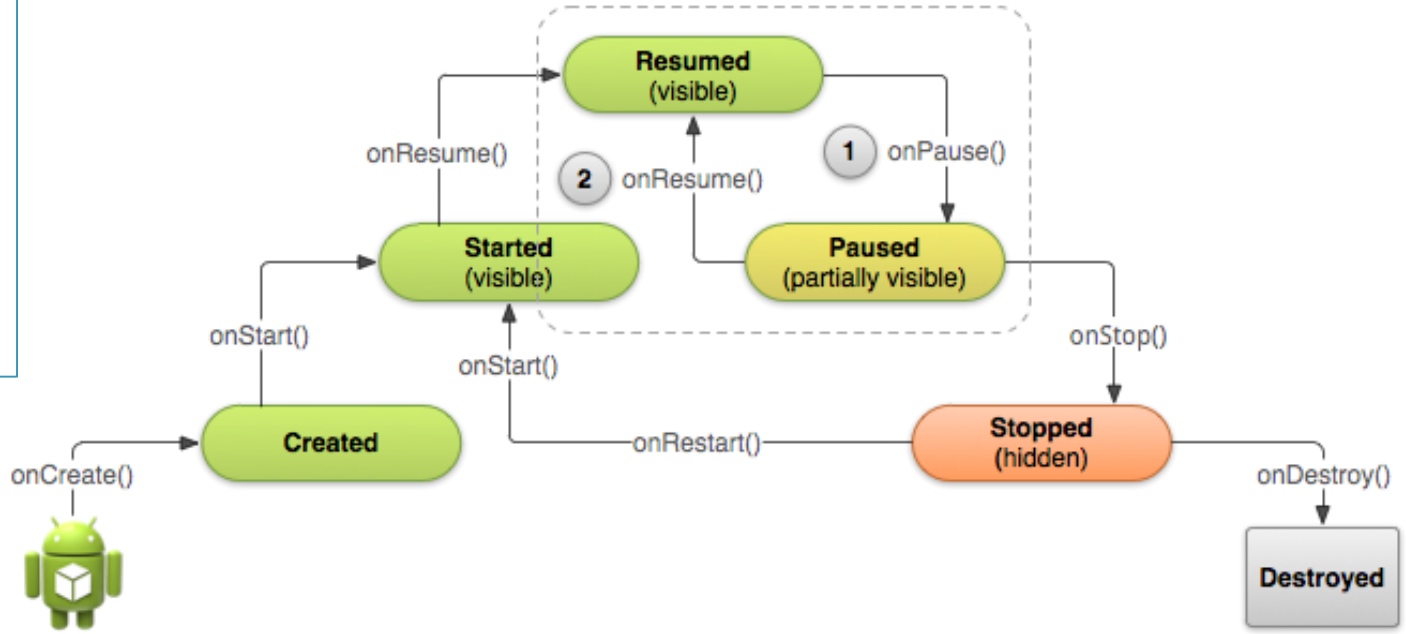
- **Content providers**
  - Manages a shared set of app data (e.g., contact list)
  - Anywhere you can access: file system, SQLite database, or on the web
  - Authorized apps can access and even change the shared data
  - Implemented as a subclass of ContentProvider

- **Broadcast receivers**
  - A component that responds to system-wide broadcast announcements
  - Screen has turned off, battery is low, a picture is captured, etc.
  - Apps can also initiate broadcasts (e.g., download is complete)
  - No UI, status bar notifications
  - Implemented as a subclass of BroadcastReceiver

Activity Life Cycle

```java
public class Activity extends ApplicationContext {

    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

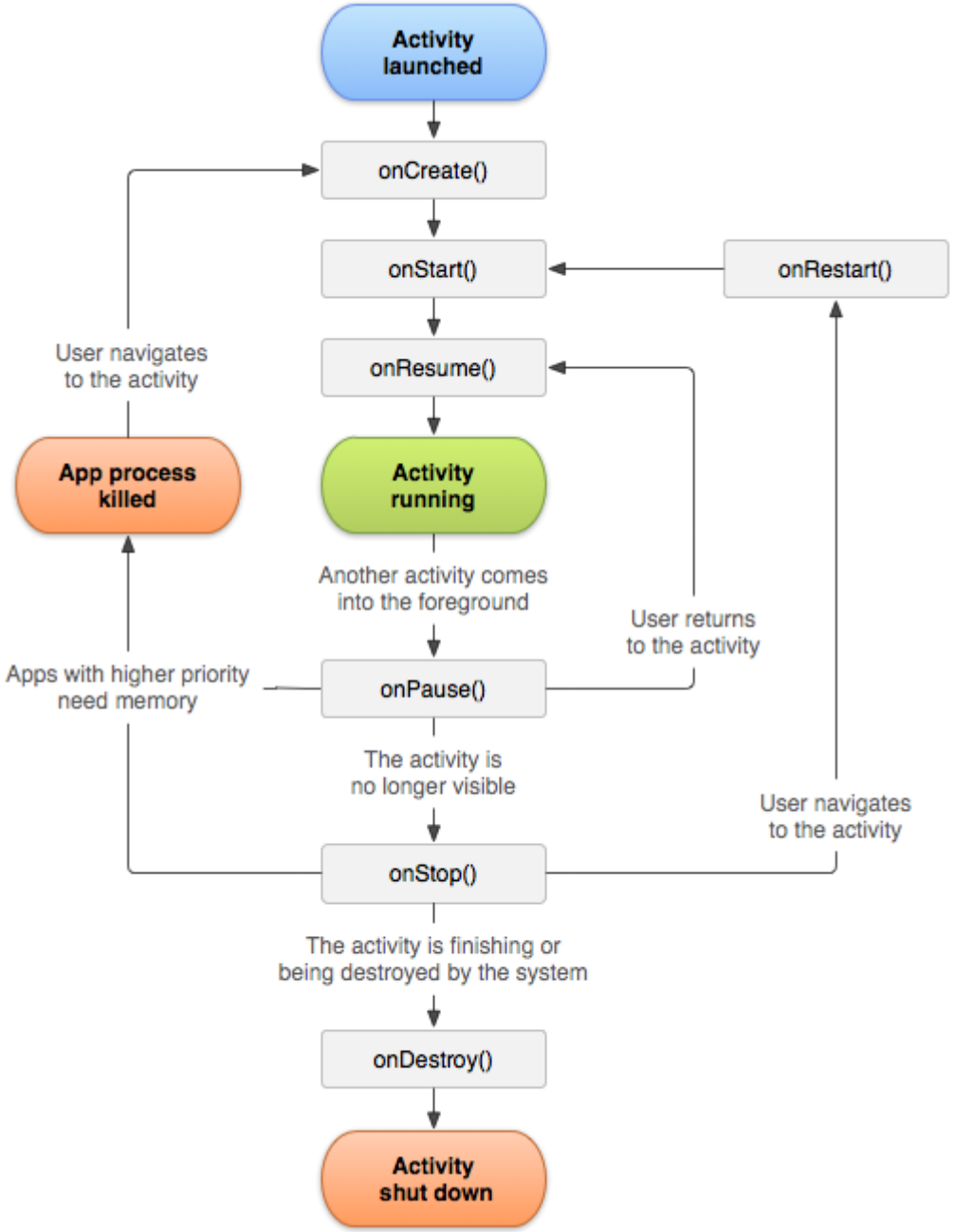
Your Activity



Activity Life Cycle

```java
public class Activity extends ApplicationContext {

    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

Your Activity



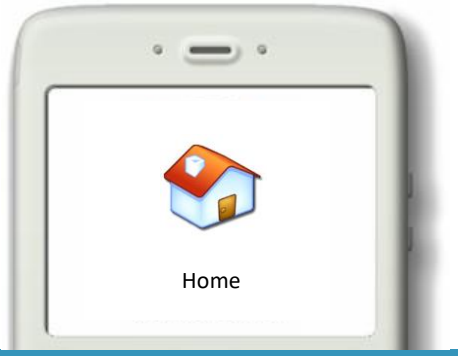
Activity Life Cycle

System process
Activity Manager

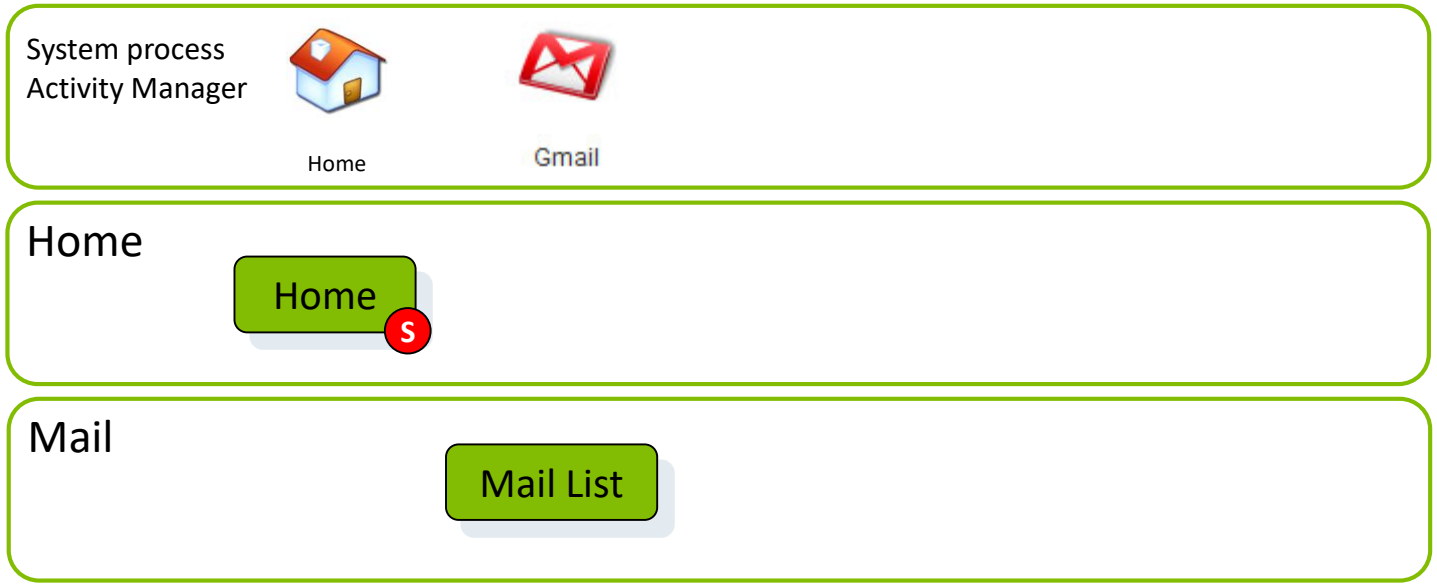
Home

Home

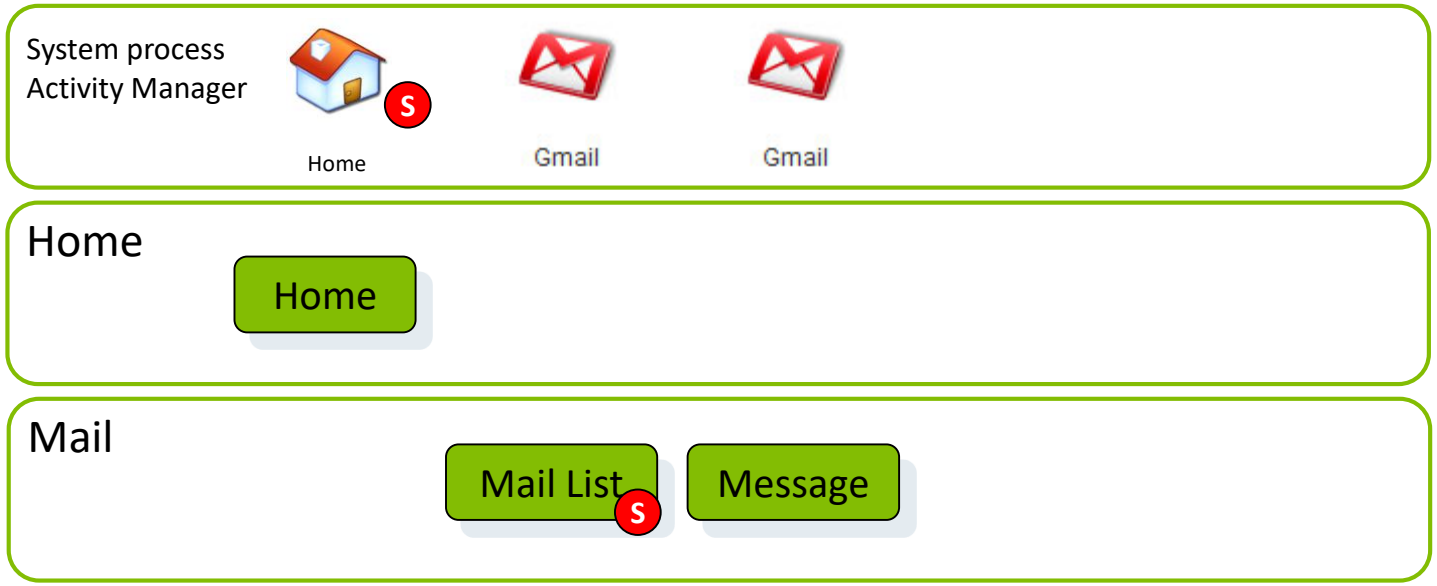
Home



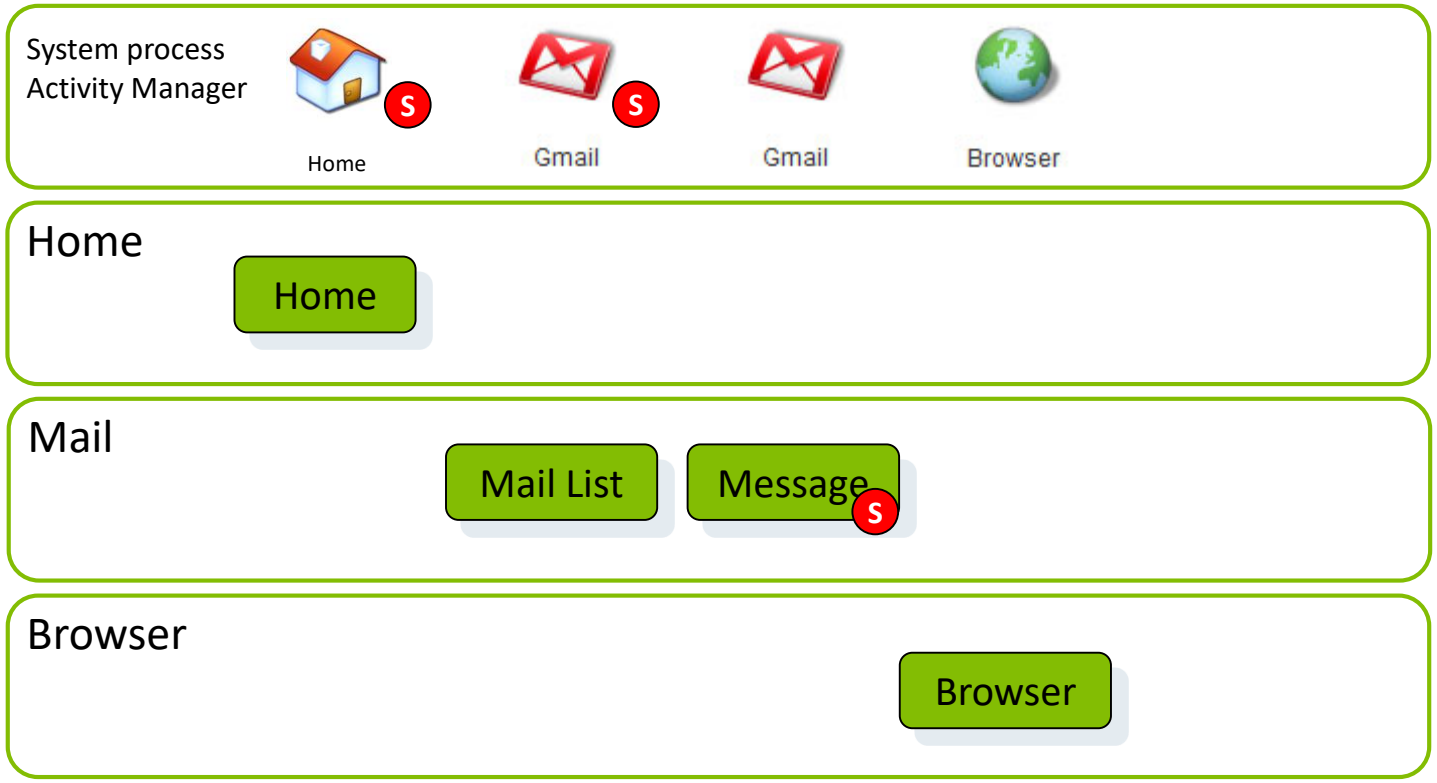
Activity Life Cycle

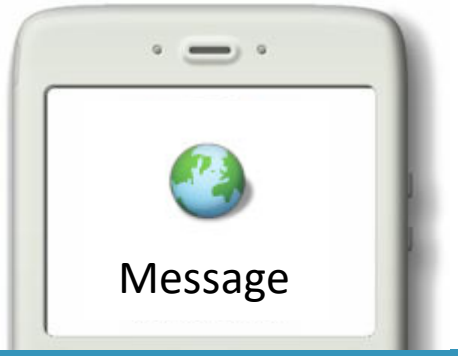


Activity Life Cycle

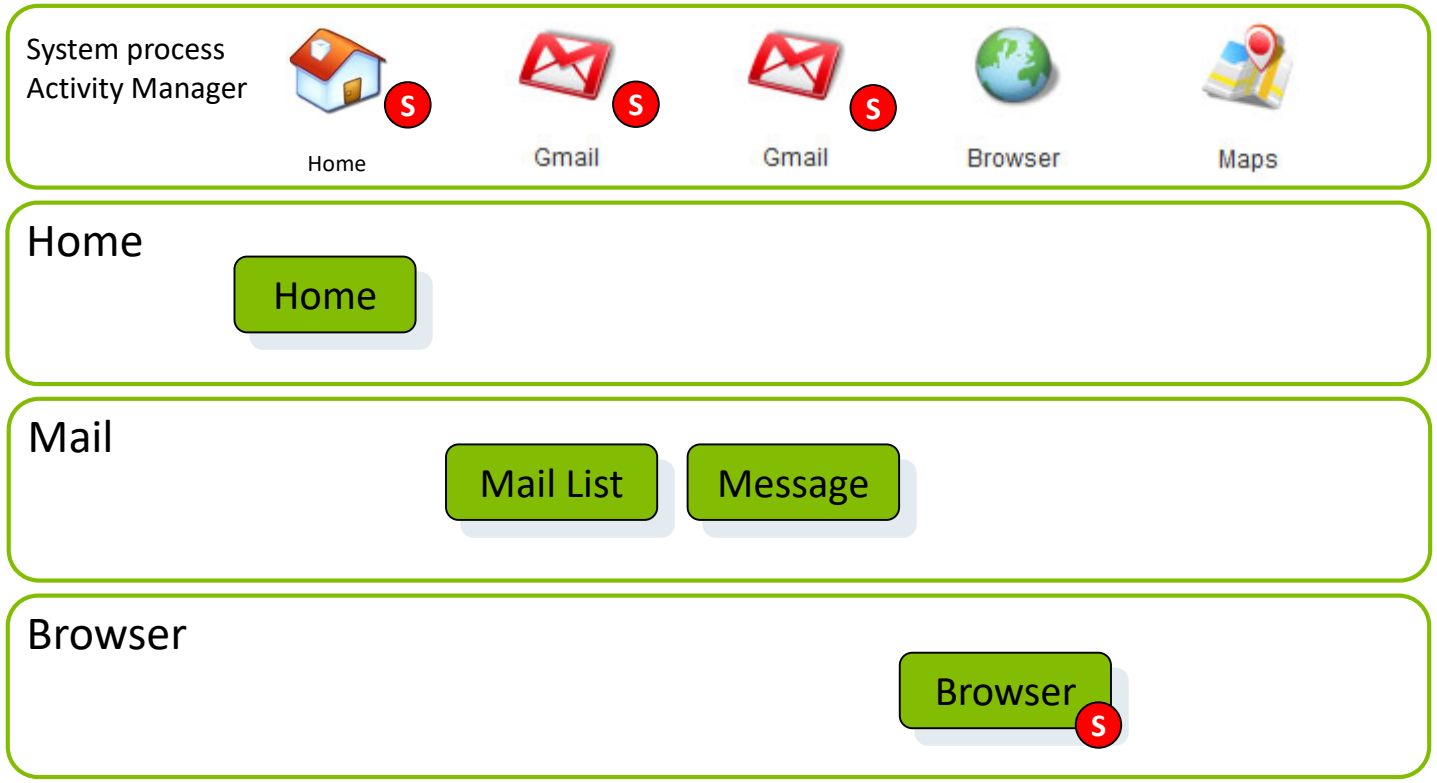


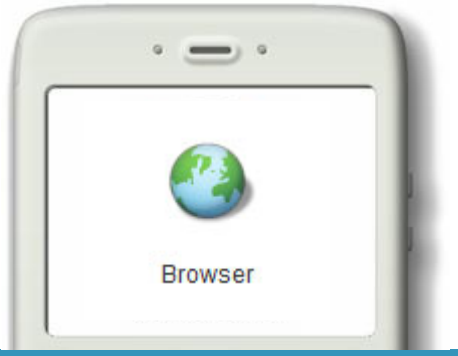
Activity Life Cycle



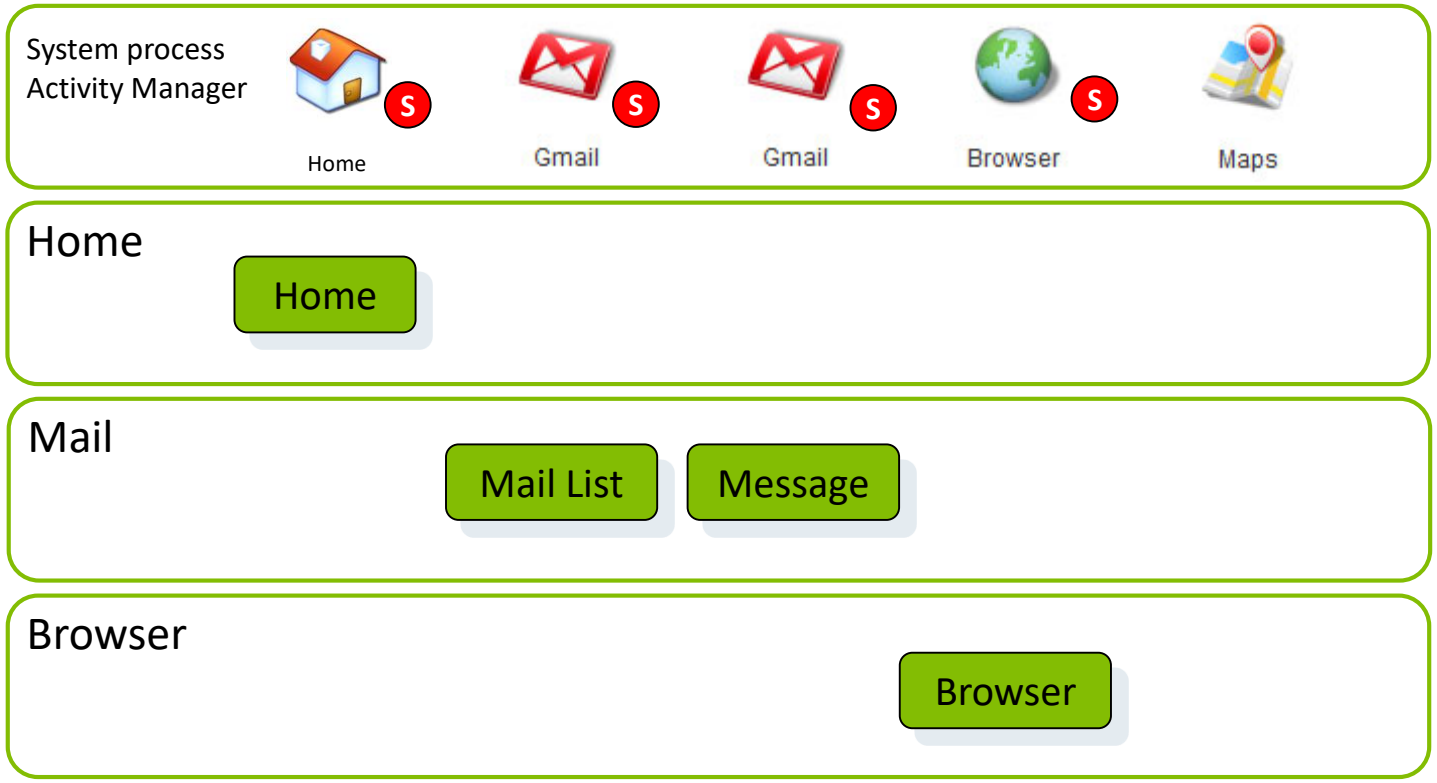


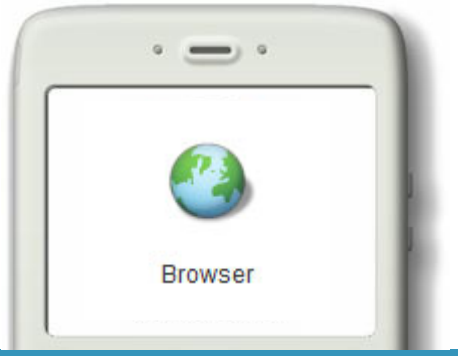
Activity Life Cycle



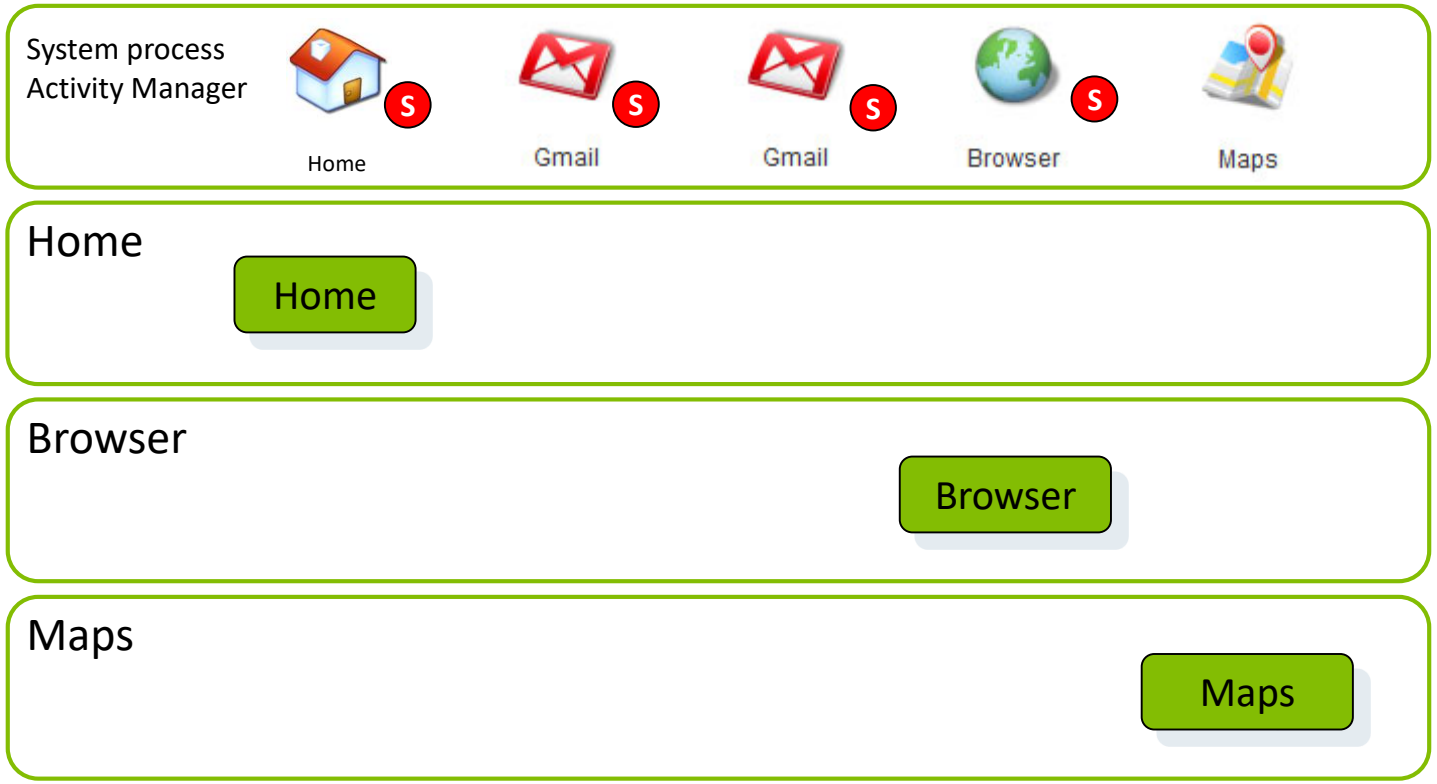


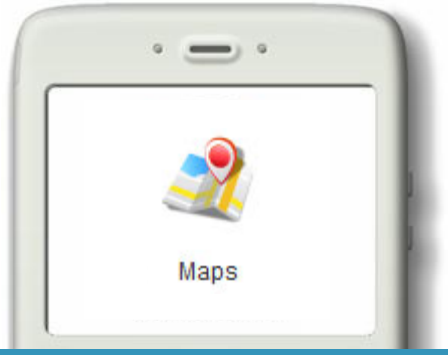
Activity Life Cycle



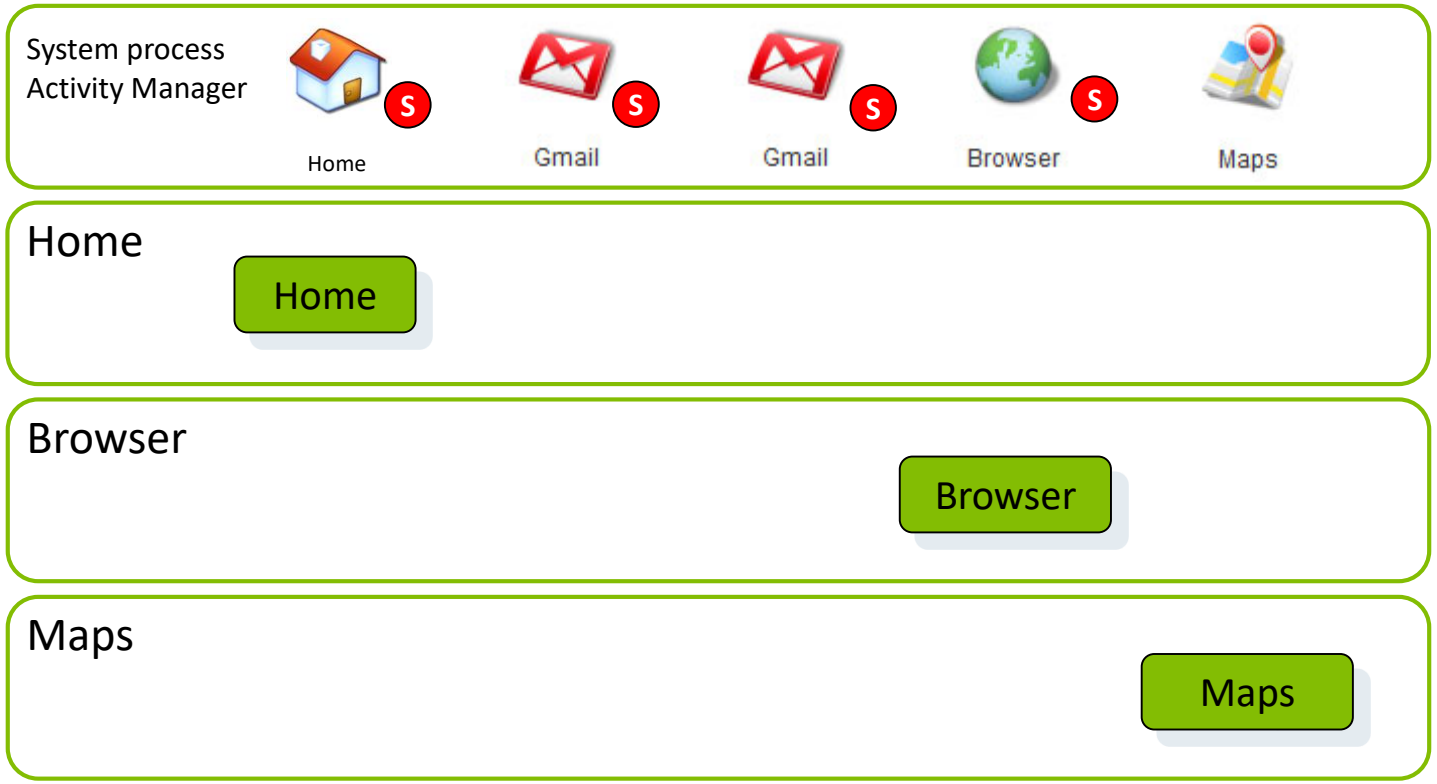


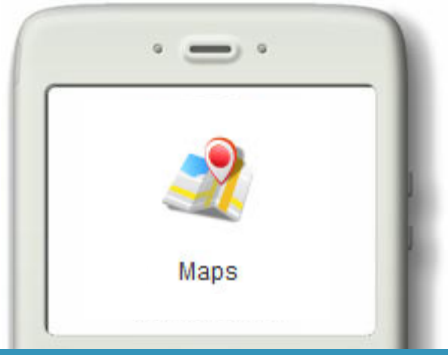
Activity Life Cycle



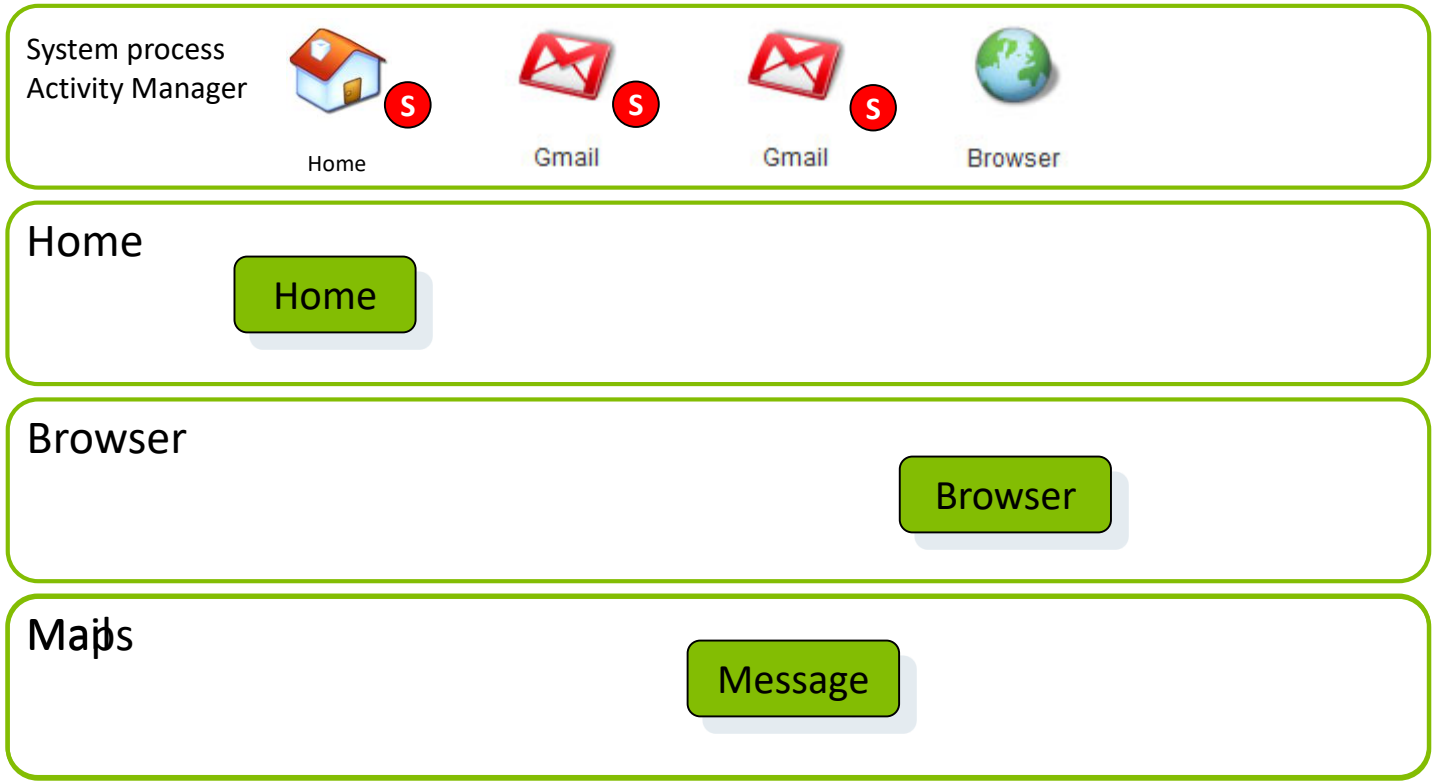


Activity Life Cycle

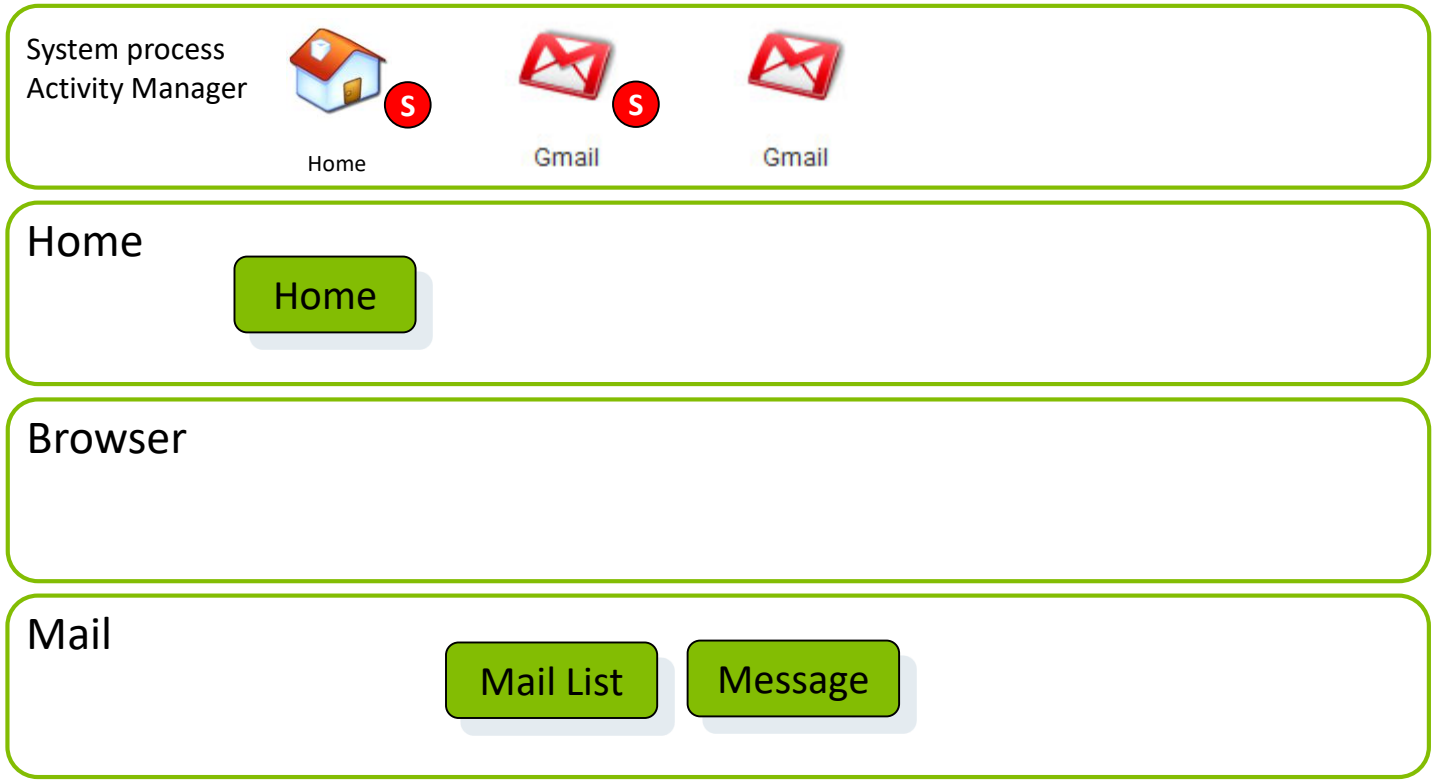




Activity Life Cycle



Activity Life Cycle



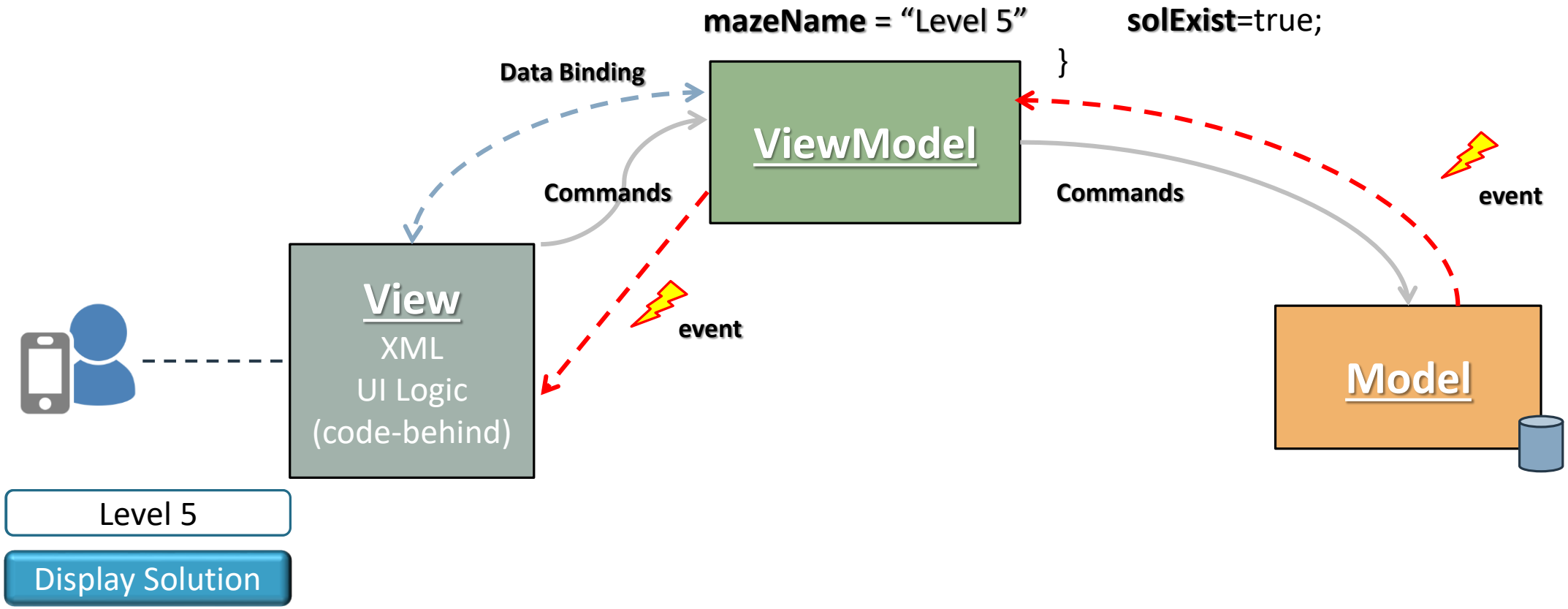
MVVM in Android

EXAMPLE

The MVVM Architecture

Data Binding Library

```
if(model.soltionExist(mazeName)){
    sol=model.getSolution(mazeName);
    solExist=true;
}
```



View with binding to the ViewModel

```
1.<layout xmlns:android="http://schemas.android.com/apk/res/android">
2.
3.   <data>
4.       <variable
5.           name="viewModel"
6.           type="myPackage.MyViewModel"/>
7.   </data>
8.
9.   <LinearLayout ... />
10.       <EditText
11.           style="@style/EditText"
12.           android:text="@{viewModel.mazeName}"
13.           android:onEditorAction="@{viewModel.onE
14.
15.         <Button
16.           style="@style/ButtonSignIn"
17.           android:enabled="@{viewModel.solExist}"
18.           android:onClick="@{viewModel.onDisplays
19.   </LinearLayout>
20.</layout>
```

```
// in the ViewModel class
public boolean onEditorAction(   @NonNull final TextView textView,
                                 final int actionId,
                                 @Nullable final KeyEvent event) {

    if (actionId == EditorInfo.IME_ACTION_DONE ||
        event.getKeyCode() == KeyEvent.KEYCODE_ENTER) {

        // mazeName = textView.getText();
        if(model.soltionExist(mazeName)){
            sol=model.getSolution(mazeName);
            solExist=true;
        }
    }
    return solExist;
}
```