4. The Android System

- System-on-Chip Emulator
- Overview of the Android System Stack
- Anatomy of an Android Application

Help Yourself

Android Java Development Kit

- Programming Android applications:
  http://code.google.com/android/documentation.html
  - Training: http://developer.android.com/training/index.html
  - Tools: http://code.google.com/android/intro/tools.html
  - API: http://code.google.com/android/reference
  - Emulator: http://code.google.com/android/reference/emulator.html
  - Tutorial: http://code.google.com/android/intro/tutorial.html
  - Design: http://code.google.com/android/devel

- *Android is "just" a very good example and a valuable open platform to explore the design and implementation of a mobile device*

4. The Android System

- System-on-Chip Emulator
- Overview of the Android System Stack
- Anatomy of an Android Application

Full-System Emulation

QEMU

- Emulation of the *guest system* (e.g., mobile phone)...
  ... on a *host system* (e.g., your laptop)
- Starting the emulator: `$ emulator`
  (`$ emulator -help` for list of options)
- More information on the baseline emulator: `www.qemu.org`

Full-System Emulation

Goldfish SoC

- Good representative of a simple system-on-chip
- ARM11 microprocessor (ARMv5 machine language)
- Video: framebuffer
- Audio in and out
- Missing (currently) in the emulated SoC:
  GSM/UMTS, bluetooth, Wifi, camera, GPS, accelerometer
- GSM and GPS can be forwarded to a host device
  `$ emulator -radio device -gps device`

Demonstration

Kernel boot log

```
$ emulator @inf422 -show-kernel
```

Interaction With the Emulator

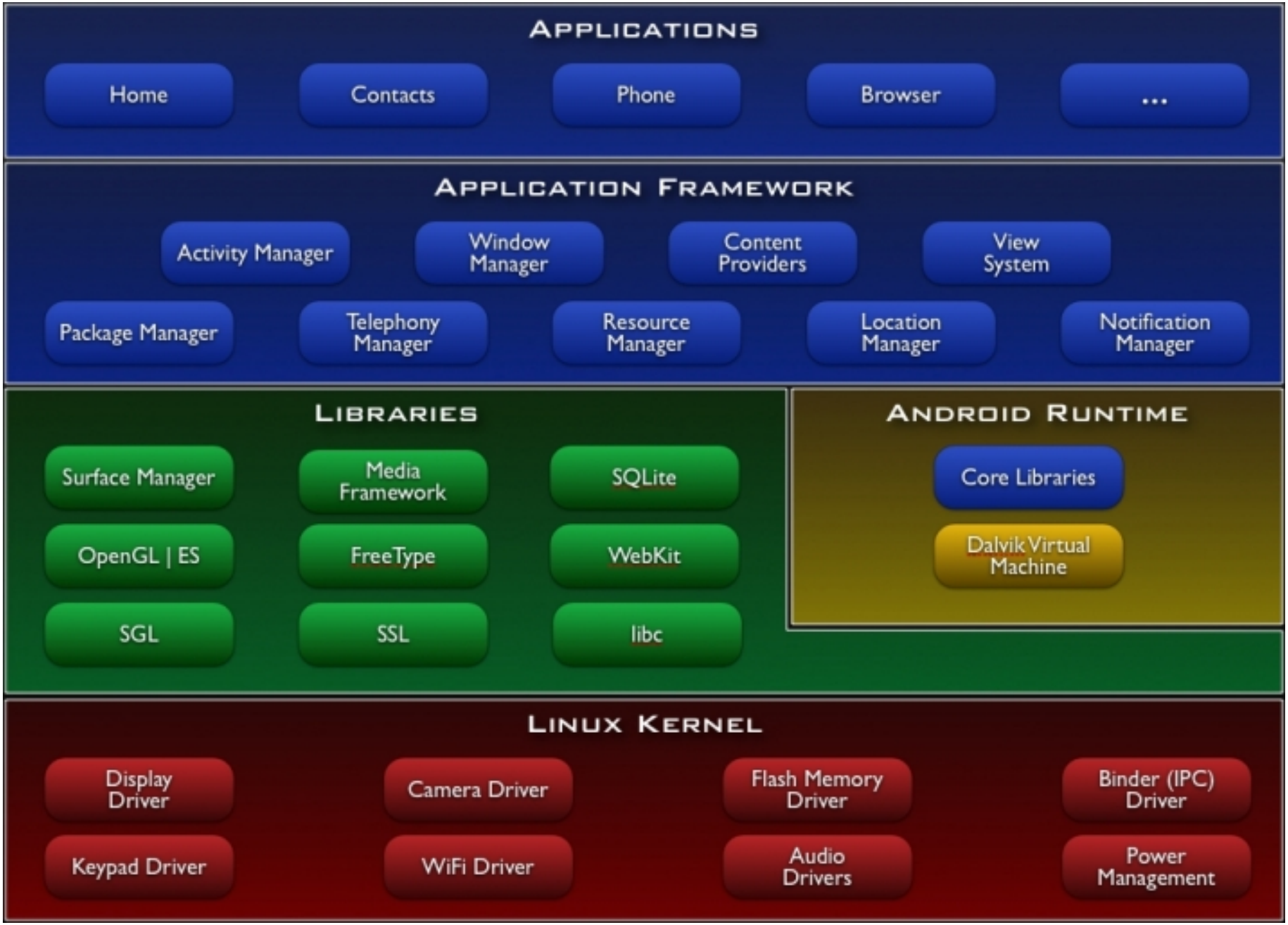
Android Debug Bridge

- Run from a command line of the host system
  `$ adb options commands`
- Copy to/from android
  `$ adb push host android` / `$ adb pull android host`
- Forward network connections from the host *to* android (and other options)
  `$ adb forward tcp:host_port tcp:device_port`

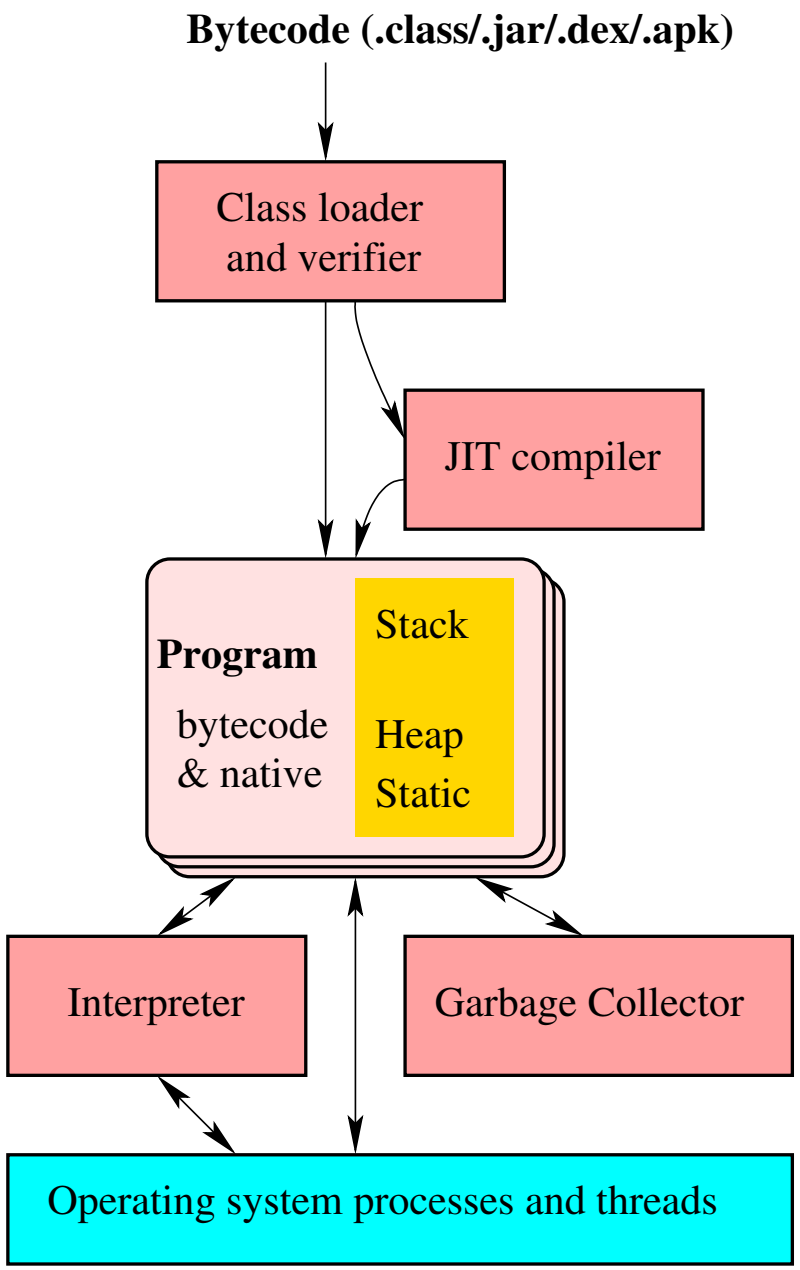
4. The Android System

- System-on-Chip Emulator
- Overview of the Android System Stack
- Anatomy of an Android Application

Android System Stack



Java Virtual Machine



- All Android applications run on top of the Dalvik JVM
- A Java method can be *interpreted* directly or *optimized and compiled to native (machine) code*
  - Delayed compilation to a specific target machine language is called *Just-In-Time (JIT) compilation*
  - Android has JIT compiler since version 2.2

4. The Android System

- System-on-Chip Emulator
- Overview of the Android System Stack
- Anatomy of an Android Application

Activity

Activity: Interactive Task

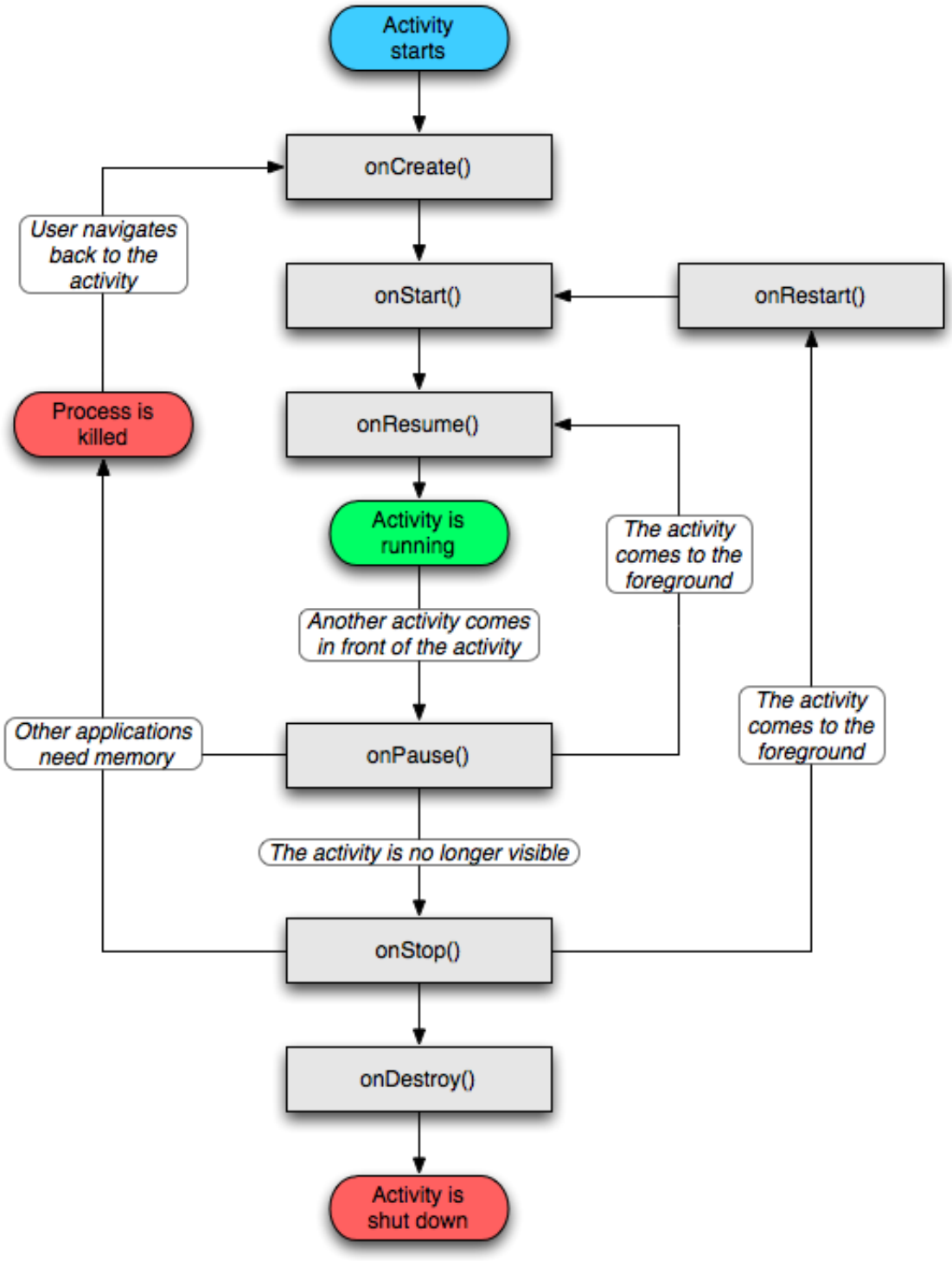
- Stack of running activities
- Events associated with creation, destruction, access to the framebuffer (screen) and user-interface
- Starts with one main thread in charge of user interactions

Activity

Skeleton of an Activity

```
package com.android.myactivity;

import android.app.Activity;
import android.os.Bundle;

public class MyActivity extends Activity
{
    // Method called when (an instance of) the Activity is created
    public void onCreate(Bundle savedInstanceState) {
        // Delegate the generic work to the parent Activity class
        super.onCreate(savedInstanceState);

        // Display the XML layout in the screan associated with the Activity
        setContentView(R.layout.main);

        // More work, typically run in a concurrent thread
    }
}
```

Activity


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Service

Service: Non-Interactive Task

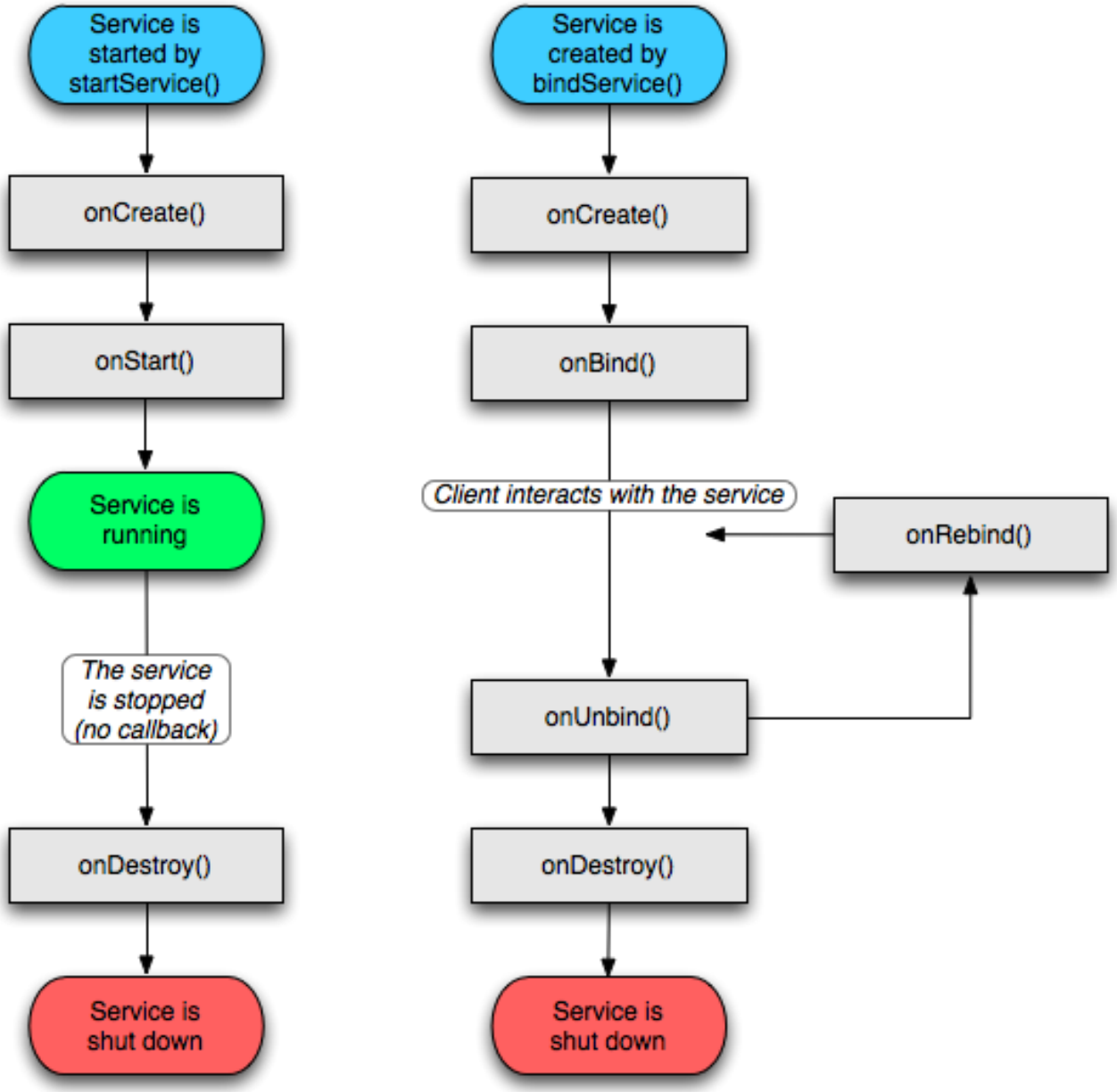
- Default system services
- Activities may launch services
- Services may have interactions through `Notification` widgets and events

Service

Skeleton of a Service

```
package com.android.myservice;

import android.app.Service;

public class MyService extends Service
{
    // Method called when (an instance of) the Service is created
    public void onCreate() {
        // Start up the thread running the Service
        // Create a separate thread because to avoid blocking the Service main thread
        Thread st = new Thread() {
            void run() { /* ... */ }
        };
        st.start();
    }

    // Method called when the (instance of) the Service is requested to terminate
    public void onDestroy() {
        /* ... */
    }
}
```

Service


Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down
Service is created by bindService()
onCreate()
onBind()
Client interacts with the service
onRebind()
onUnbind()
onDestroy()
Service is shut down

Manifest File

Example: Simple Activity With Internet Access Permission

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.android.td3">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".TD3" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
   </application>
   <uses-permission xmlns:android="http://schemas.android.com/apk/res/android"
                    android:name="android.permission.INTERNET"></uses-permission>
</manifest>
```

Resources and Intents

Resource

- External *value* or *name-value* pair defined in the Android project tree
- `res` directory of a project tree

Intent

- Binding between a class and an activity in charge of it, e.g.,
  - Displaying a given media format
  - Associating an interactive notification to a particular event

Graphical User Interface (GUI)

High-Level XML Layout

- Organized into `Layouts` and `Views` (GUI widgets)
- `res/layout/main.xml` (generated from interactive editor)
- `R` class automatically generated (unique identifier for each `View` in the GUI)

Example

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

    <EditText android:id="@+id/txtmem"
      android:editable="false"
      android:cursorVisible="false"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"/>

    <Button android:id="@+id/save_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Save"/>
</LinearLayout>
```

Graphical User Interface (GUI)

Interacting With the GUI

- Important: only the *main thread* can directly call `View` methods
  - Design choice: reduce the number of concurrent threads, makes `View` methods simpler and faster (no implicit synchronization)
- Use a *call-back* mechanism for other threads to command the execution of a `View` method in the main thread

Graphical User Interface (GUI)

Explicit Call-Back Example

- Call-back handler: command queue
  - Attached to the thread it was created from, by default
  - Each command is defined by an object of type `Runnable`

```
Handler callback = new Handler();

Runnable update_meminfo = new Runnable()
{
    public void run() {
        EditText txt_meminfo;
        txt_meminfo = (EditText)findViewById(R.id.txtmem);
        txt_meminfo.setText("Memoire libre (kB): " + /* ... */);
    }
};

/* ... */

// In the run() method of another thread
void run() {
  /* ... */
  callback.post(update_meminfo);
}
```

Graphical User Interface (GUI)

Implicit Call-Back Example

- Event-driven execution in the GUI's event loop, similar to a call-back
  - A custom command is defined by overloading the `onClick()` method

```
View.OnClickListener save_to_log = new OnClickListener()
    {
        public void onClick(View v)
        {
            EditText txt_meminfo;
            txt_meminfo = (EditText)findViewById(R.id.txtmem);
            txt_meminfo.setText("Memoire libre (kB): " + /* ... */);
        }
    };

/* ... */

// In the onCreate() method of the Activity
Button button = (Button)findViewById(R.id.save_button);
button.setOnClickListener(save_to_log);
```