



ANDROID

Grundbegriffe und Architektur

von Markku Lammerz für das Seminar „Android: Plattform für mobile Geräte“
an der Universität Hamburg beim DKRZ

★ Einleitung

- Kurzer Lebenslauf von Android
- Die Entwicklungshilfen

★ Grundbegriffe

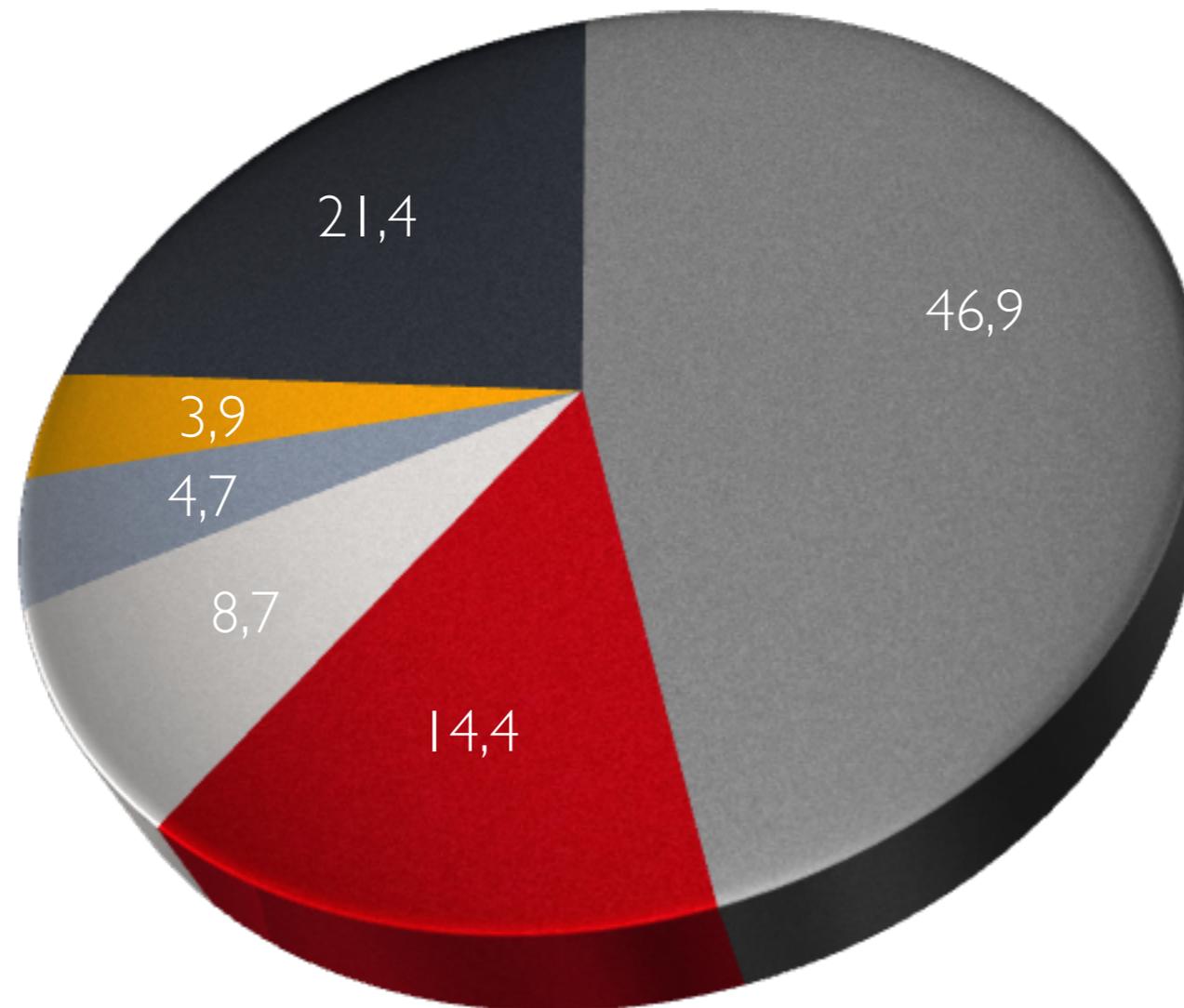
- Komponententypen (Activities, Services...)
- Prozesshandhabung und Zugriffsrechte
- Ressourcen

★ Architektur

- Aufbau der Plattform
- Performance Tipps

- ★ Wird von Google innerhalb der Open Handset Alliance entwickelt (Nov. 2007)
- ★ Erstes Gerät war das HTC Dream (22. Oktober 2008)
- ★ Zurzeit aktuell ist Version 2.1 („Eclair“)
- ★ OpenSource (Lizenz: Apache & GPL)
- ★ Vertriebsplattform für Anwendungen ist der Android Market

- ★ Es werden „60 000 Geräte pro Tag ausgeliefert“ (Feb. 2010)
- ★ Im Vorjahr lag Android bei 0,5%



★ Software Development Kit

- Unterstützte IDE ist Eclipse mithilfe des ADT Plugins
- Sonst per Text-Editor, Android Debugger, Java-Compiler, Cross-Assembler und Android Debug Bridge
- Mit Android Simulator lässt sich fast alles testen:
 - SMS empfang oder GPS-Koordinaten usw.

★ Android Market mit Unterstützung für kostenlose und kostenpflichtige Apps

- In folgenden Vorträgen: Manifest und Signierung

- ★ Anwendungsentwicklung findet in Java (V. 1.5) statt
- ★ Hat eine sehr gute Dokumentation unter developer.android.com
- ★ Eclipse ADT bietet einen GUI Editor ansonsten werden GUIs in XML definiert
- ★ Anzahl der Klassen im SDK:

Java Klassen	769
Android spezifisch	511
Andere	168

★ **Activity**

- Sozusagen das GUI, also eine Oberfläche mit der interagiert werden kann (vgl. Form)
- Activities haben verschiedene Zustände

★ **Services**

- Laufen im Hintergrund, Benutzer interagiert nicht mit ihnen
- Zwei Startmodi: Lokal oder Remote

★ **Content Provider**

- Hält Daten die von anderen Anwendungen abgerufen werden können
- Wird per eindeutiger URI identifiziert
- Speichert seine Daten beliebig ab (SQLite, File...)
- Mit Hilfe eines *ContentResolver* können Anfragen gestellt werden (zurück kommt ein *Cursor*)

★ Intent

- Sind Nachrichten auf Anwendungsebene mit Hilfe derer man z.B. mit weiteren Komponenten kommunizieren kann.
- Mit *startActivityForResult()* ist auch eine Rückgabe möglich
- Es gibt zwei Arten von Intents:
 - *Explizit*
 - direktes Adressieren
 - Beispielcode:

```
Intent meinIntent = new Intent(this, AufgerufeneActivity.class);  
meinIntent.setDate(fooBar);  
meinIntent.putExtra(„feld I“, „daten I“);  
startActivity(meinIntent);
```

- *Implizit*

- Das System entscheidet aufgrund der Parameter welche Anwendung gestartet wird
- Hilft dabei alle Anwendungen austauschbar zu halten

```
Intent meinIntent =  
new Intent(Intent.ACTION_VIEW, Uri.create(„http://wr.dkrz.de“));  
startActivity(meinIntent);
```

- ★ Das System verschickt selbst Broadcast Intents diese werden allerdings auf Application Framework Ebene verschickt

★ Broadcast Receiver

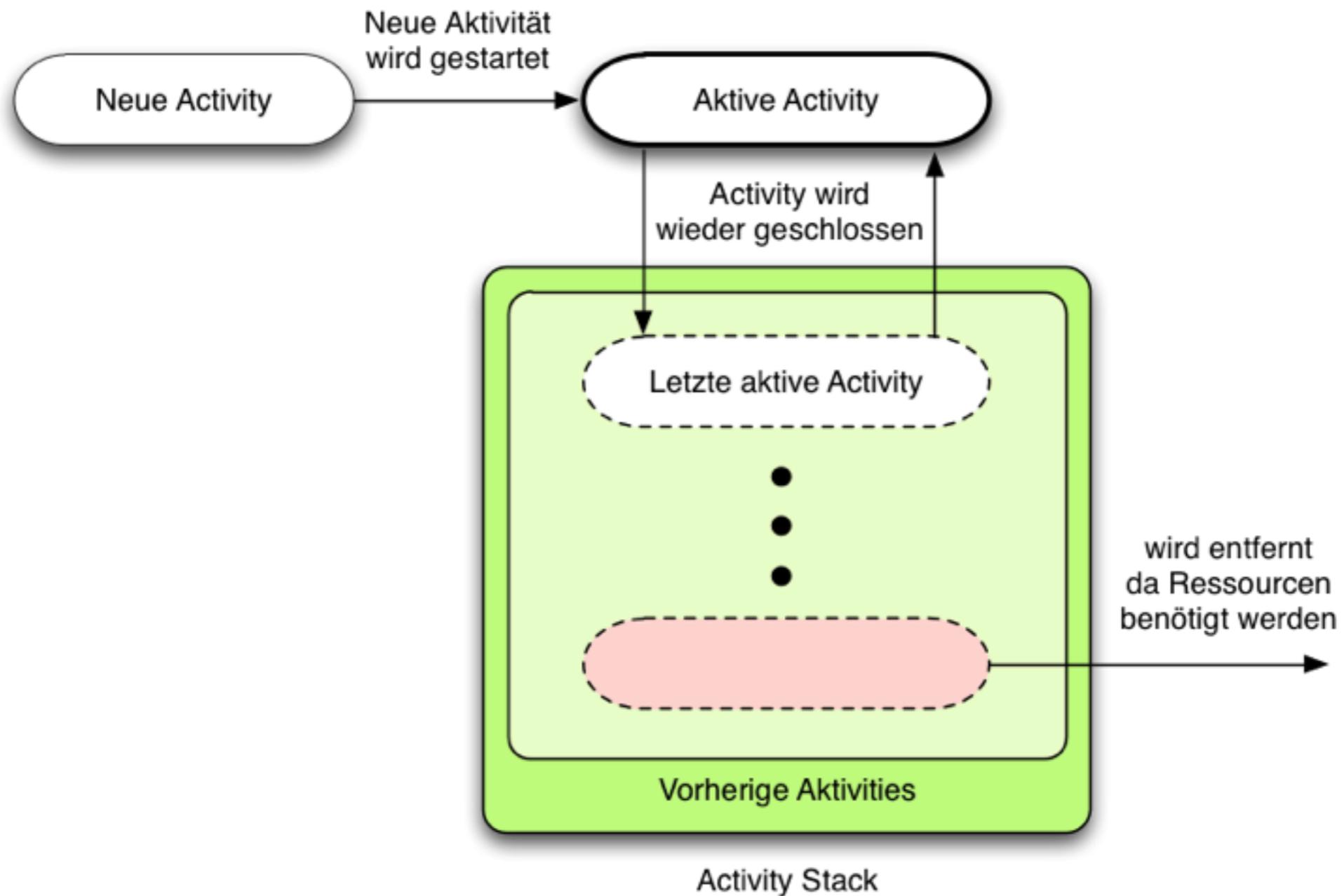
- Laufen im Hintergrund

```
private class BatterieNotStatusReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        ... melde das Batterie fast leer ...  
    }  
}
```

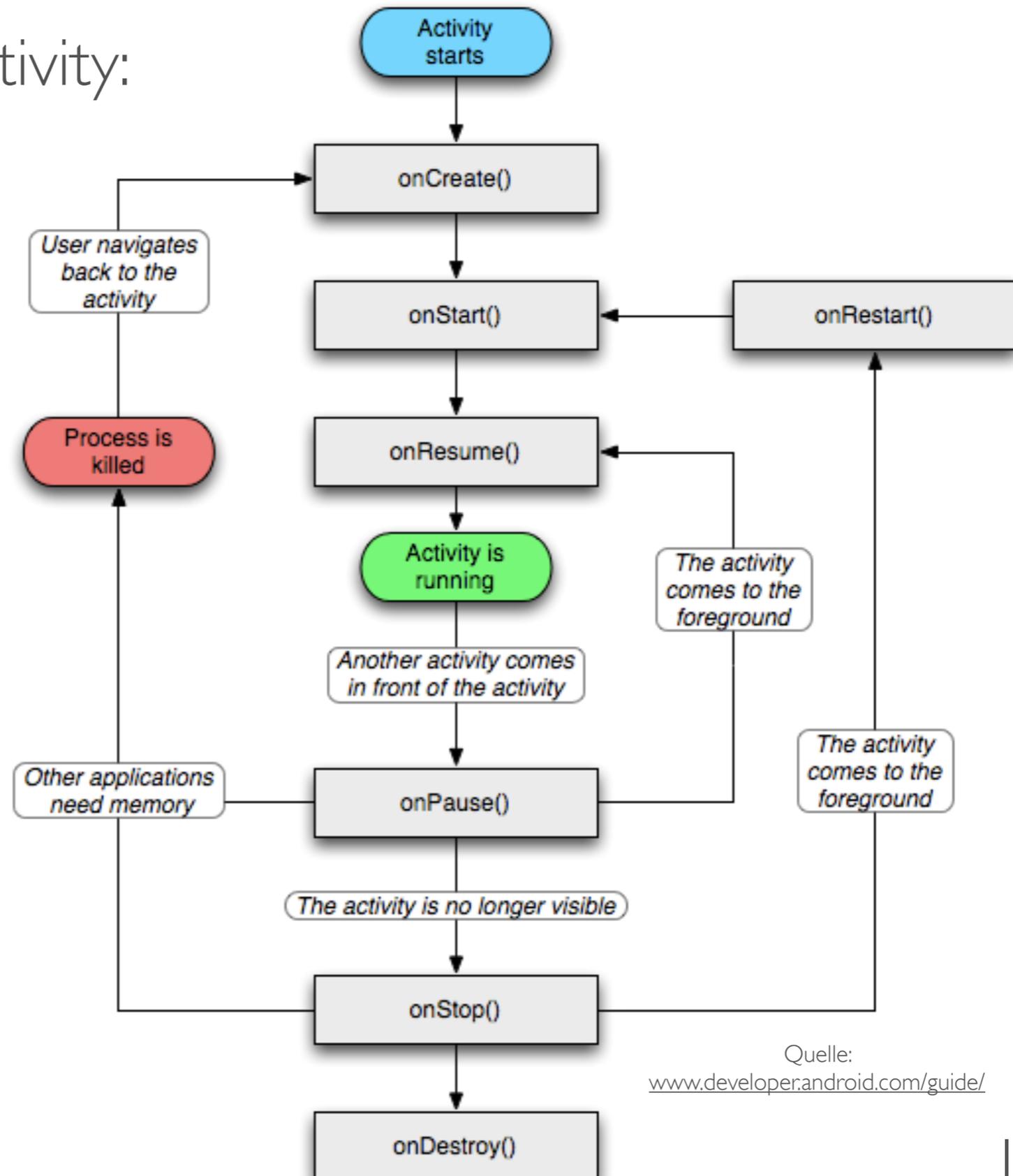
```
IntentFilter mIF = new IntentFilter(„android.intent.action.BATTERY_LOW“);  
BatterieNotStatusReceiver mBR = new BatterieNotStatusReceiver();  
context.registerReceiver(mBR, mIF);
```

- ★ Jede Instanz läuft in einem eigenen Linux-Prozess
- ★ Die Applikation sowie ihre Komponenten haben keinen direkten Einfluss auf ihre Lebenszeit
- ★ Activities einer Anwendung sind in der selben VM:
 - Es gibt die Möglichkeit diese auszulagern
- ★ In jedem Prozess können mehrere Threads laufen
 - Threads sind fest an den Prozess gekoppelt

Activity Stack:

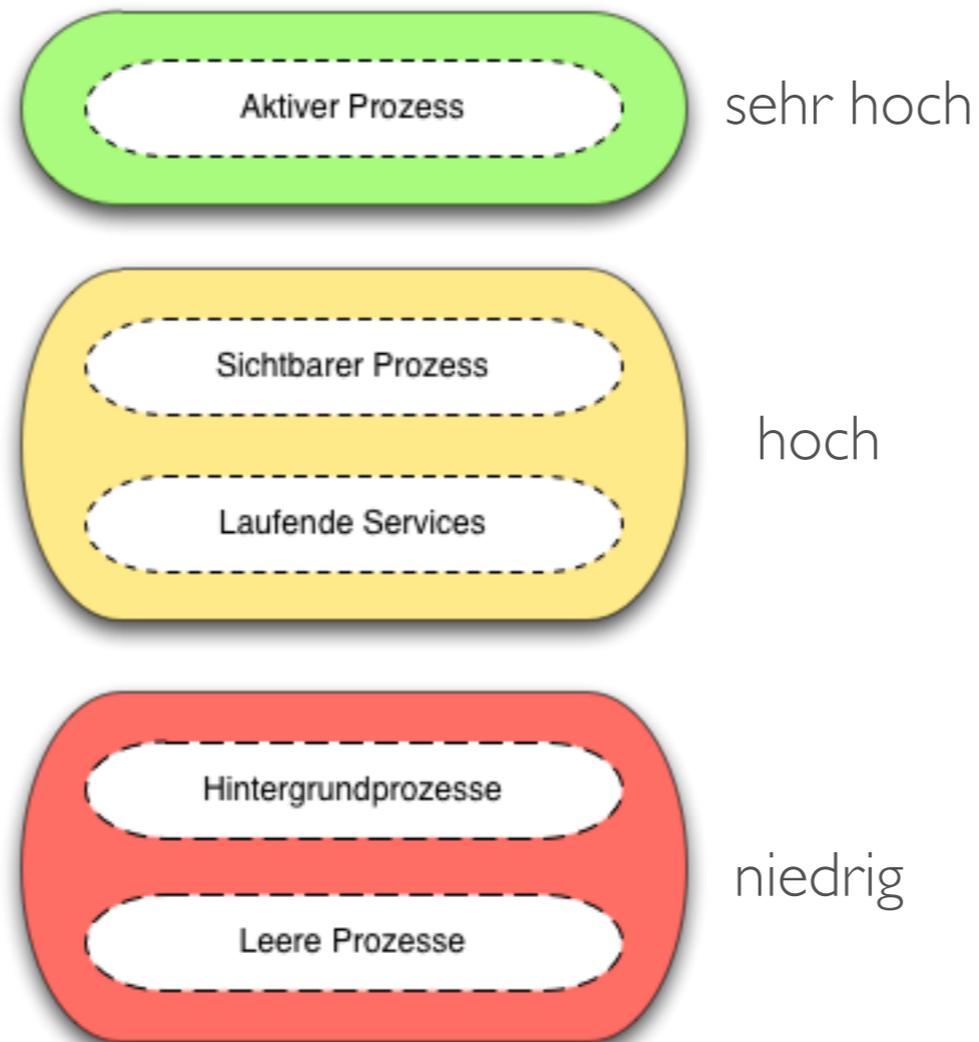


Lebenszyklus einer Activity:



Quelle:
www.developer.android.com/guide/

Prozesspriorisierung



- Jede Applikation läuft in eigener VM und hat somit einen eigenen Adressraum (hat sogar eigenen Benutzer)
- Content Provider können für Lese- und Schreib-Operationen Zugriffsbeschränkungen setzen
- ... weiteres dazu in einem der folgenden Vorträge welcher sich mit dem Manifest beschäftigt ...

- Verwalten von Daten mithilfe von XML (Vorteil: Lesbarkeit)
- Wird beim Kompilieren in Binärdaten umgewandelt
- Ressourcen-Klasse wird für Anwendung automatisch erzeugt, mit ihrer Hilfe kann auf die gespeicherten Daten zugreifen

```
package com.android.samples;
public final class R {
    public static final class string {
        public static final int start_button_text=0x02040001;
        public static final int main_screen_title=0x0204000a;
    };
    public static final class layout {
        public static final int start_screen=0x02070000;
        public static final int new_user_pane=0x02070001;
    };
    public static final class drawable {
        public static final int company_logo=0x02020005;
        public static final int smiling_cat=0x02020006;
    };
};
```

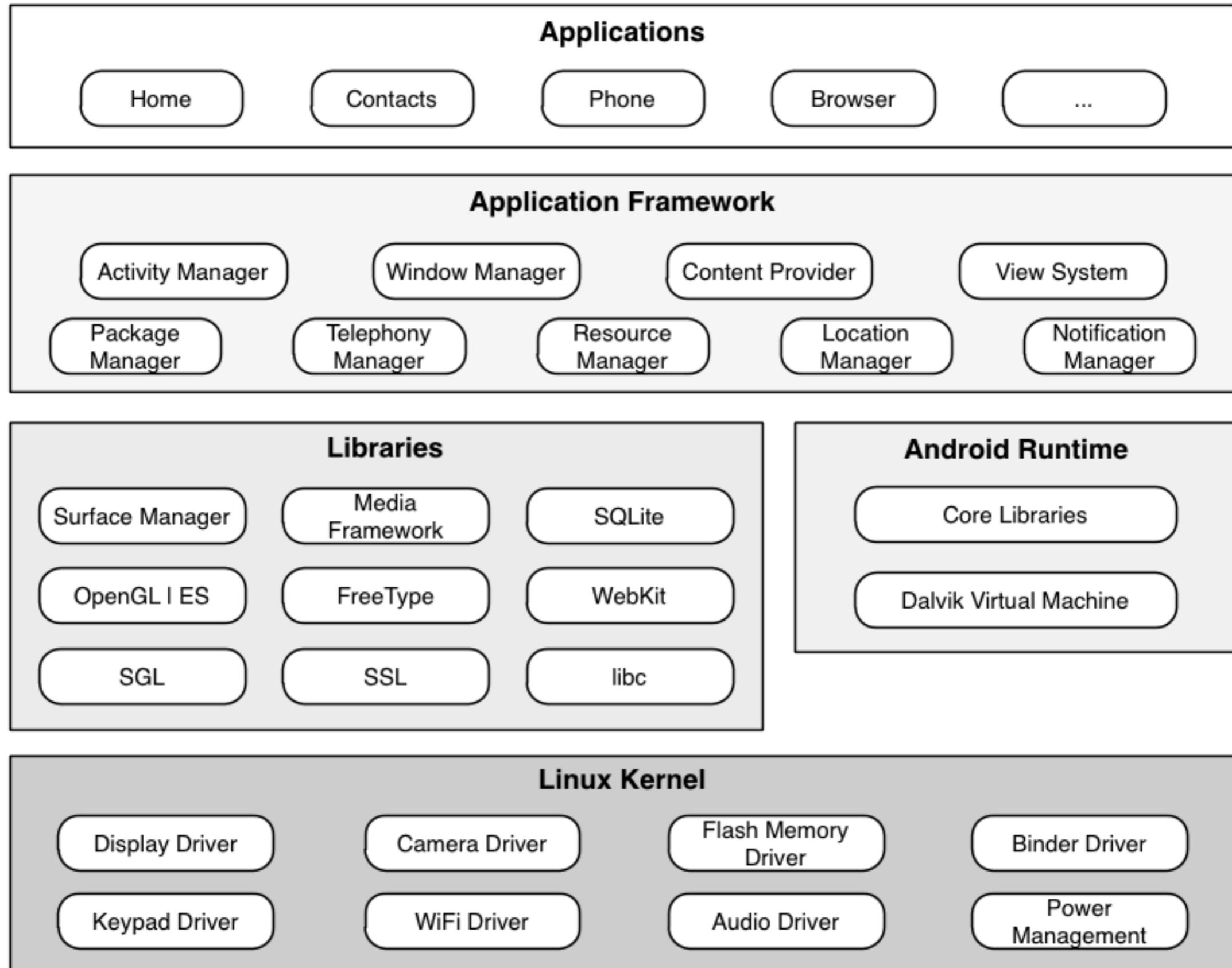
/res/values/beliebigerName.xml

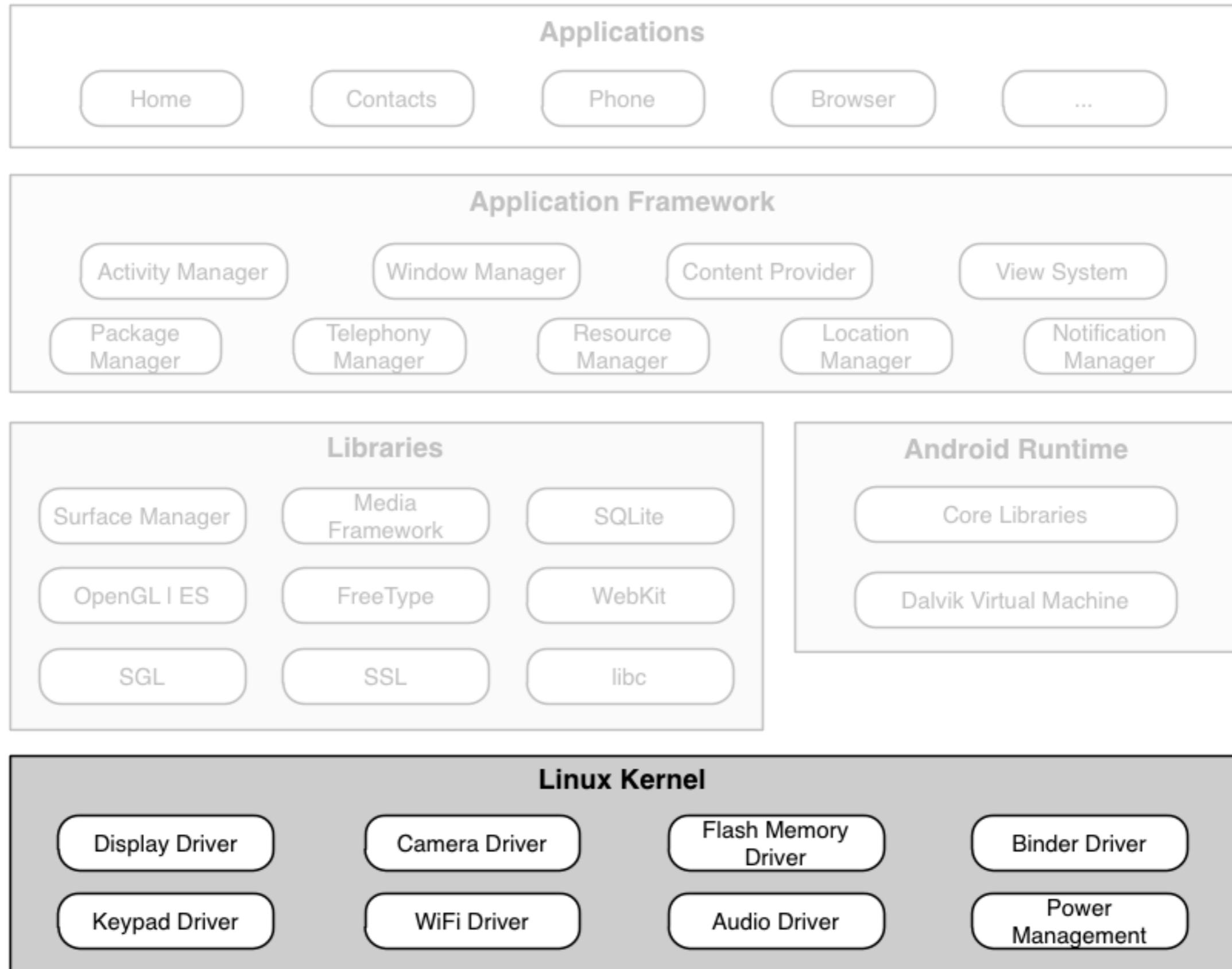
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="helloworld">"Hello World!"</string>
  <string name="beenden">"Die Anwendung wird nun geschlossen."</string>
</resources>
```

- Zugriff auf Daten über die R-Klasse mit Hilfe einer Hilfsfunktion die anhand der Adresse den Text zurückgibt

```
String titel = Resources.getText(R.string.helloworld);
```

- Es gibt vordefinierte Ordner in /res/ für die verschiedenen Dateitypen z.B. /res/drawable/ für Bilder

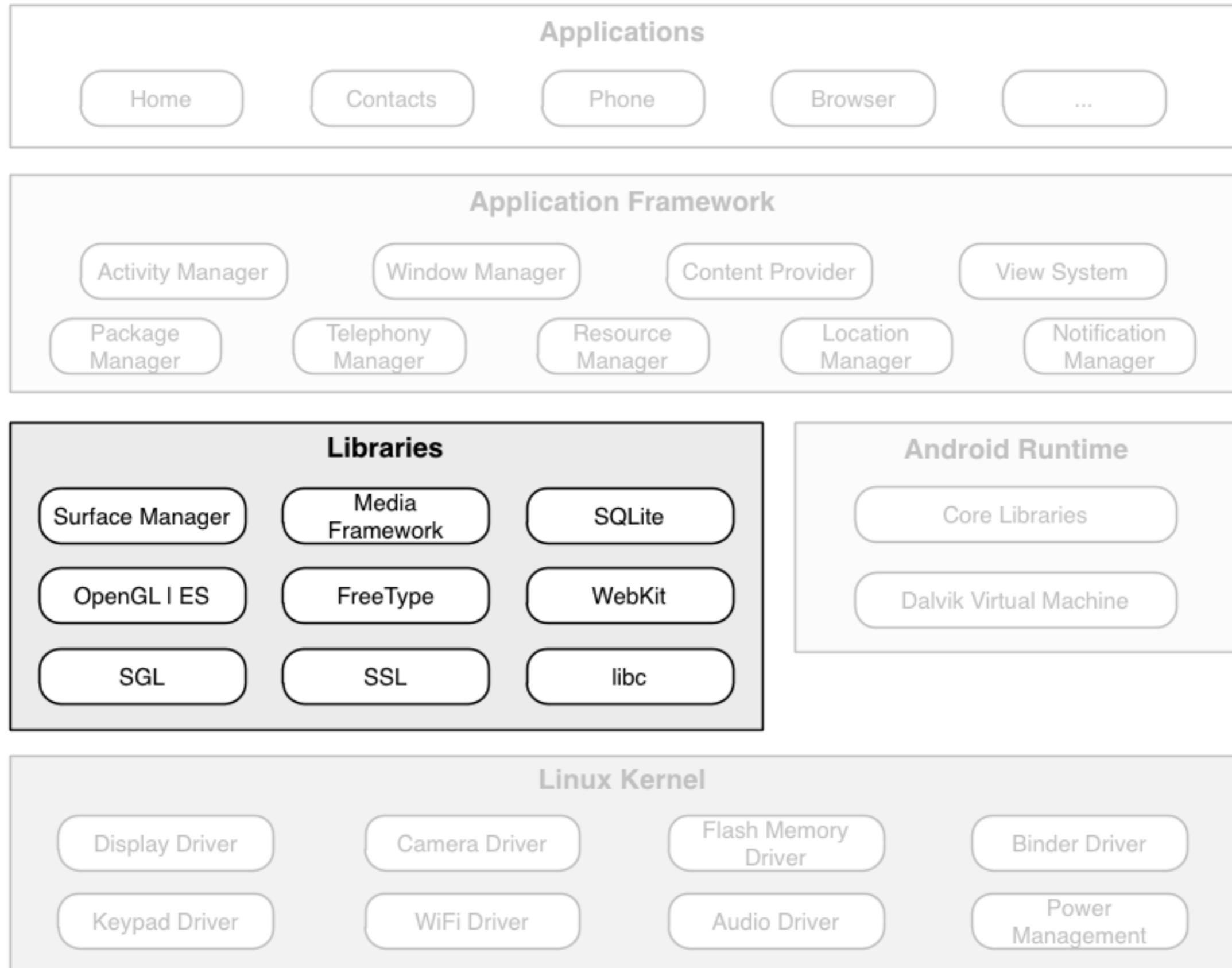




★ Zur Zeit aktuell Version: 2.6

★ Zuständig für:

- Hardwareabstraktion (Display, Wifi, Kamera etc.)
 - Durch Abstraktion sind Hardwareunterschiede kein Problem
- PowerManagement
- Prozessverwaltung
- Netzwerkkommunikation



- Sind in C / C++ geschrieben
- Die meisten Libraries sind schon lange im OpenSourcebereich in Verwendung wie z.B. SQLite, OpenGL ES...
- Bilden Schnittstelle zwischen Treibern und dem „Application Framework“

★ **SQLite**

- Unterstützt einen Großteil des SQL-92 Standards
- Sehr kompakte Datenbank
 - Minimale Stack Größe 4KiB und Heap 100KiB
- Komplette public domain (Rechtsverzicht des Rechteinhabers)
- Nach Einfügen der SQLite Bibliotheken läuft der Zugriff auf Datenbank prozessintern

★ **OpenGL ES** (Open Graphics Library for Embedded Systems)

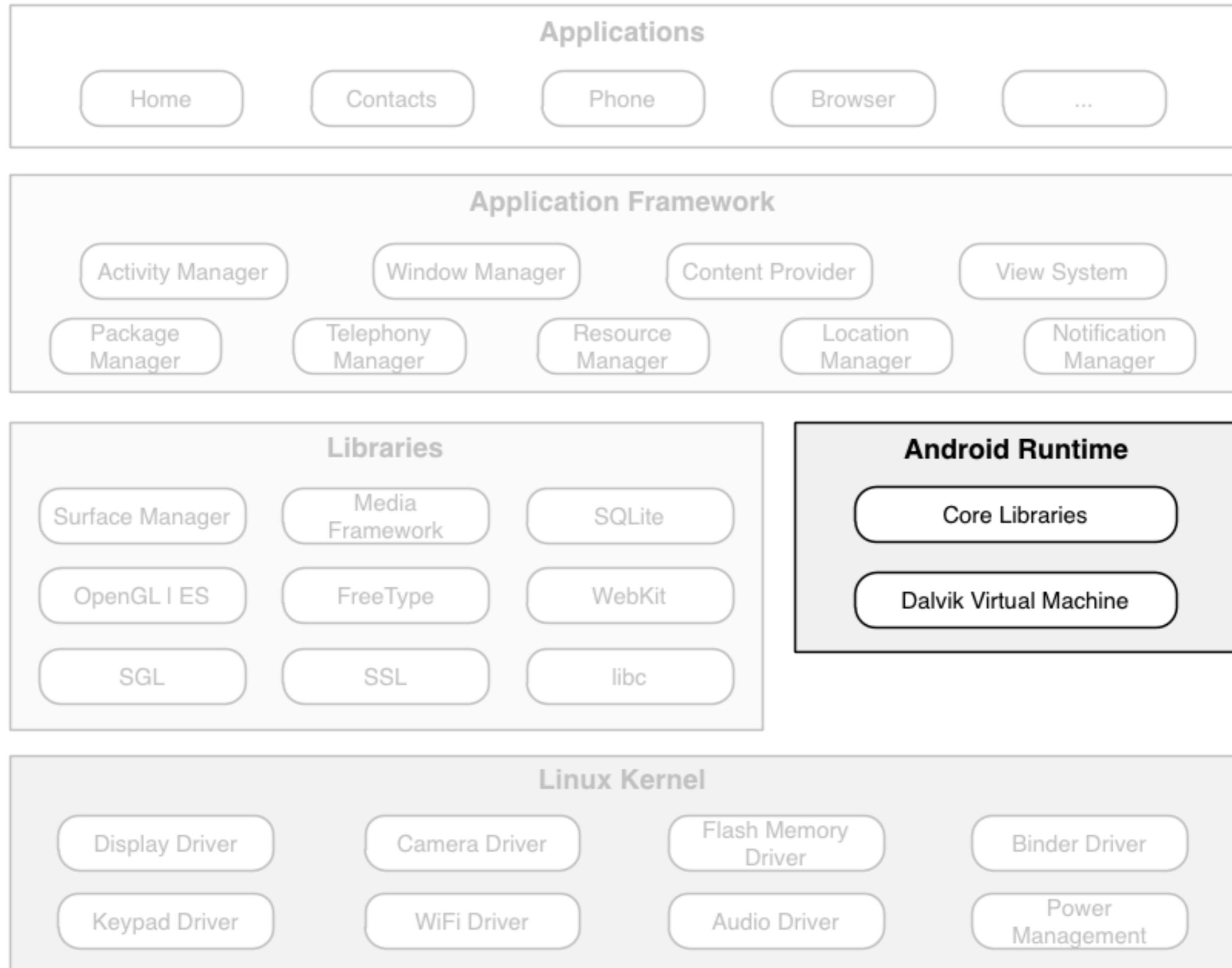
- Eine Programmierschnittstelle für 2D- & 3D-Anwendungen
- Hardwareunterstützung in Grafikchips
- In Android steckt zurzeit OpenGL ES 2.0
- Eine Hardwarebeschleunigung ist grundsätzlich bei den meisten Android Geräten möglich

★ **Medien Framework**

- Basiert auf „Open Core“ von PacketVideo
- Unterstützt (Abspielen und Aufnehmen):
 - H.264
 - MP3
 - AAC, JPG, PNG, MPEG4 etc.

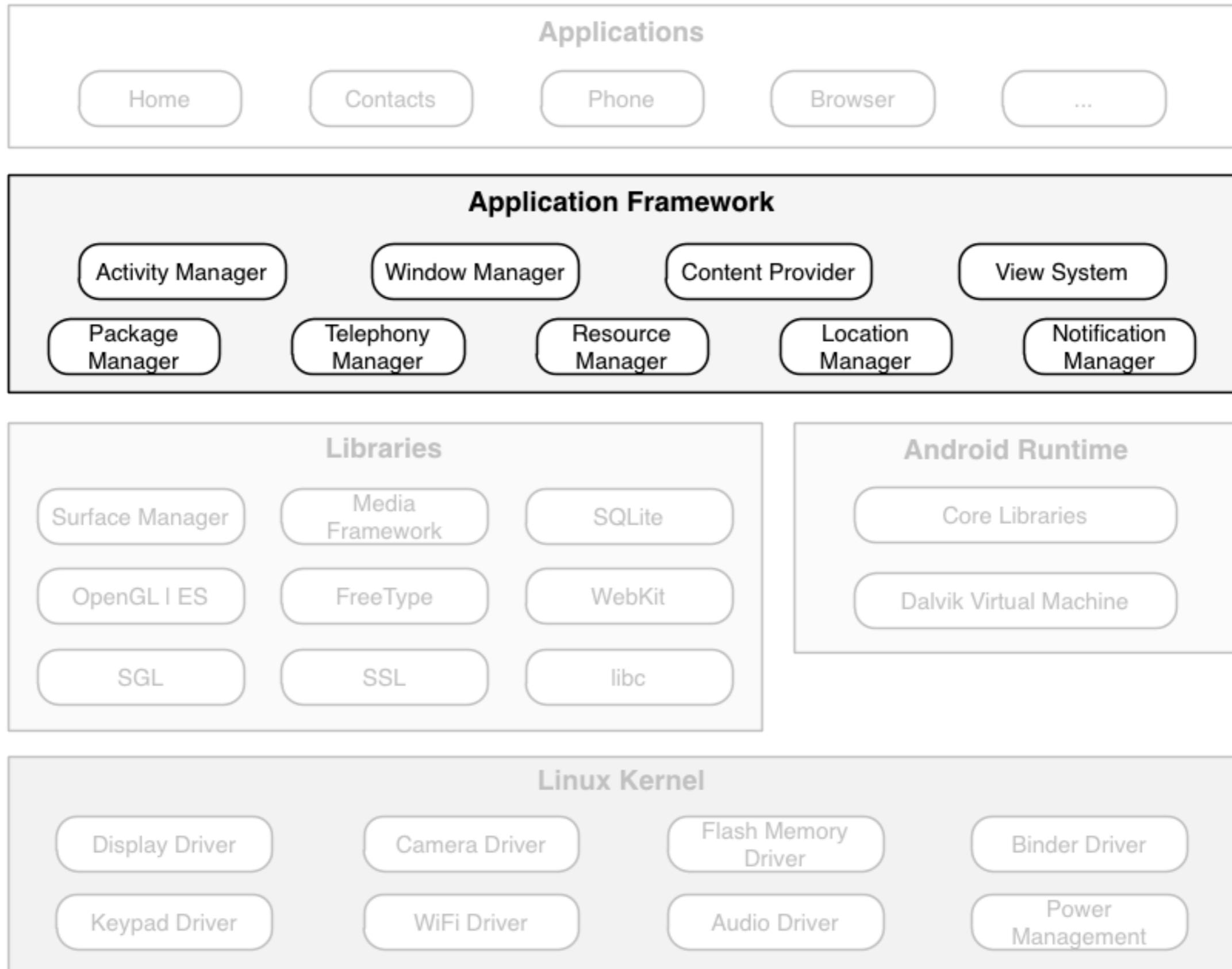
★ **SGL** (Scalable Graphics Library)

- Die 2D Grafikengine zur Darstellung von Vektoren und Grafiken
- Anwendungen wie WindowManager und SurfaceManager nutzen SGL



- ★ „Core libraries“ beinhalten größten Teil der Funktionalität von Java
 - Eine kleinere Java SE 5 Version die aber immer noch mächtiger ist als Java Mobile Edition
- ★ Jede Anwendung läuft in seiner eigenen Instanz der DalvikVM
- ★ .Dex ist das Format der DalvikVM, welche mit dem „dx“-Tool aus mit Java kompilierten Klassen erzeugt wird

- ★ Googles Implementation einer JavaVM basiert auf einer Registermaschine (JavaVM = Kellerautomat)
 - Ausgelegt für ARM-Architektur welche darauf basiert
- ★ Kompiliert aus JavaBytecode per Cross-Assembler seinen DalvikBytecode
- ★ DalvikVM Kompilat unkomprimiert kleiner als JAR komprimiert
 - Wird nicht mehr komprimiert zwecks Performanccsteigerung



- ★ Sind in Java geschrieben
- ★ Entwickler interagieren mit den Frameworks
- ★ Vorinstallierte Anwendungen greifen auch nur auf Frameworks zu
 - dies hilft bei der Austauschbarkeit von solchen Anwendungen

★ **View System**

- Verwaltet viele der UI Elemente sowie Benutzerevents
- Einige Beispiele:
 - View (Basis Block der mittels XML-Layouts manipuliert werden kann)
 - GestureDetector (OnDoubleTapListener)
 - Menu
 - OrientationEventListener

★ **Resource Manager**

- Kümmert sich um genau das was schon in Ressourcen besprochen wurde:
 - Verwaltet die im `/res/` Ordner liegenden Daten
 - ▶ Anhand von `ResourceTables`
 - Hilfsfunktionen zum Auslesen von Daten anhand von Adressen

★ Notification Manager

- Kann in verschiedenen Formen aufmerksam machen
 - Icon in der Statusbar
 - LED-Hardware ansteuern
 - Hintergrundbeleuchtung, Sound oder Vibrationsalarm

```
meinNotificationManger = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);  
Notifciation meineNotification = new Notification(icon, string, when); <= nur Signatur als Bsp.  
meinNotificationManager.notify(APP_ID, meineNotification);
```

★ Location Manager

- Standortbestimmung über z.B. GPS
- Möglichkeit per Broadcast-Intent periodisch informiert zu werden

```
locationManger = (LocationManager) getSystemService(Context.LOCATION_SERVICE)  
Location location = locationManager.getCurrentLocation(„gps“);
```

```
/** Beispiel wie man die Distanz zwischen der eigenen Position und einer gegeben  
bekommt, dabei ist die Rückgabe in Metern */
```

```
String distanz = String.valueOf(location.diatanceTo(andereLocation)) + „ Meter“;
```

★ Vermeide...

- Interfaces
- Typen: Enums, Float
- Interne Getter und Setter aufrufe

★ Verwende...

- Static-Methoden da diese schneller sind als Instanz-Methoden (es wird kein zugriff auf die vtable benötigt)
- Lokale Cache-Variablen für Ergebnisse von Gettern
- Final für Konstanten

Aktion	Zeit
Lokale Variable	1
Objekt Variable	4
String.length()	5
native Methode	5
static Methode	12
virtuelle Methode	12,5
Interface Methode	15
put() bei HashMap	600
Starten einer leeren Activitie	3.000.000

- ★ Die Entwicklungshilfen
- ★ Komponententypen
 - Activities, Services, ContentProvider
 - Intents und Broadcast-Intents
- ★ Prozesshandhabung
 - Lebenszyklen von Aktivites
 - Prozesspriorisierung
- ★ Ressourcen-Klasse
- ★ Schichtenmodell der Android Architektur
- ★ Performance Tipps



NOCH FRAGEN?