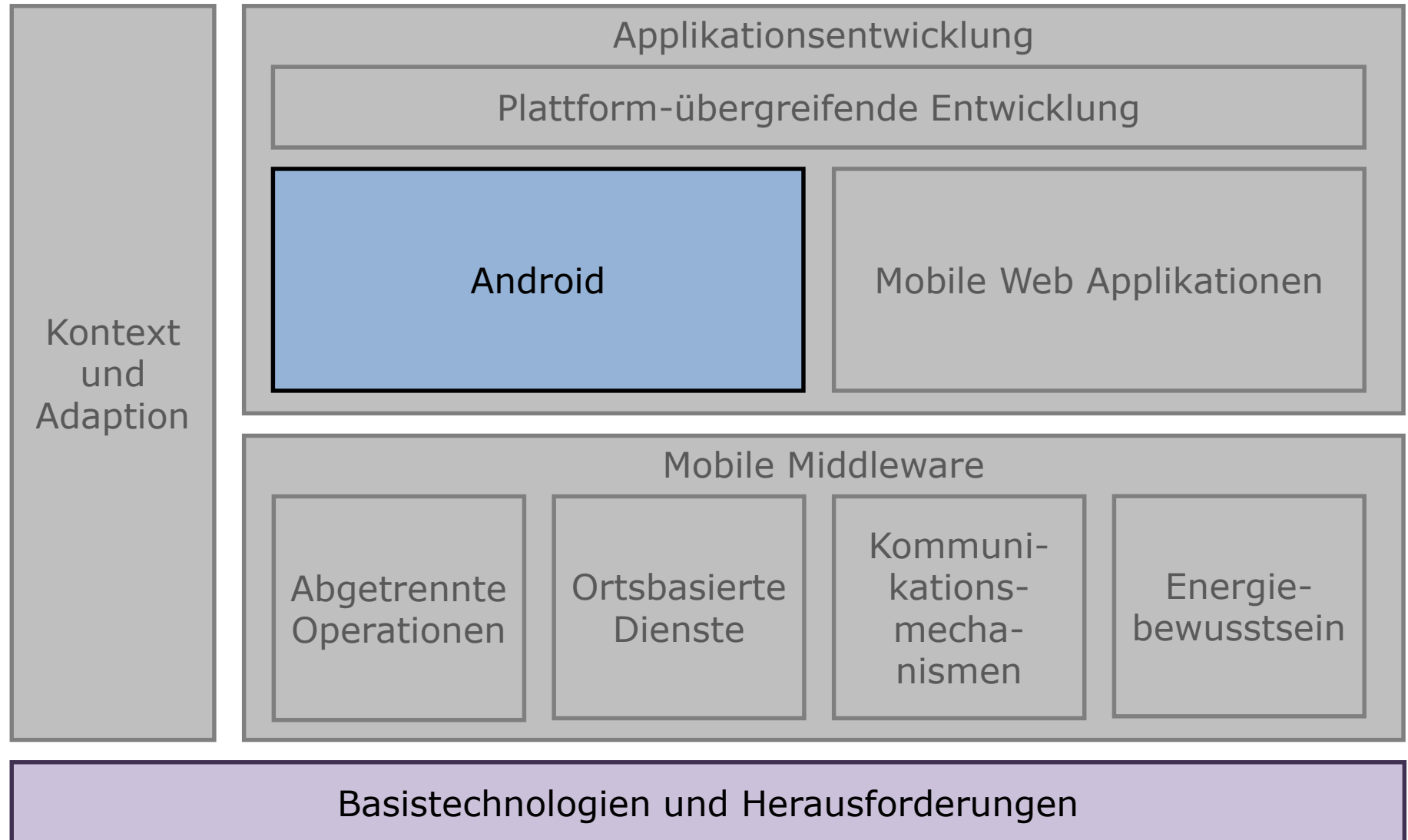
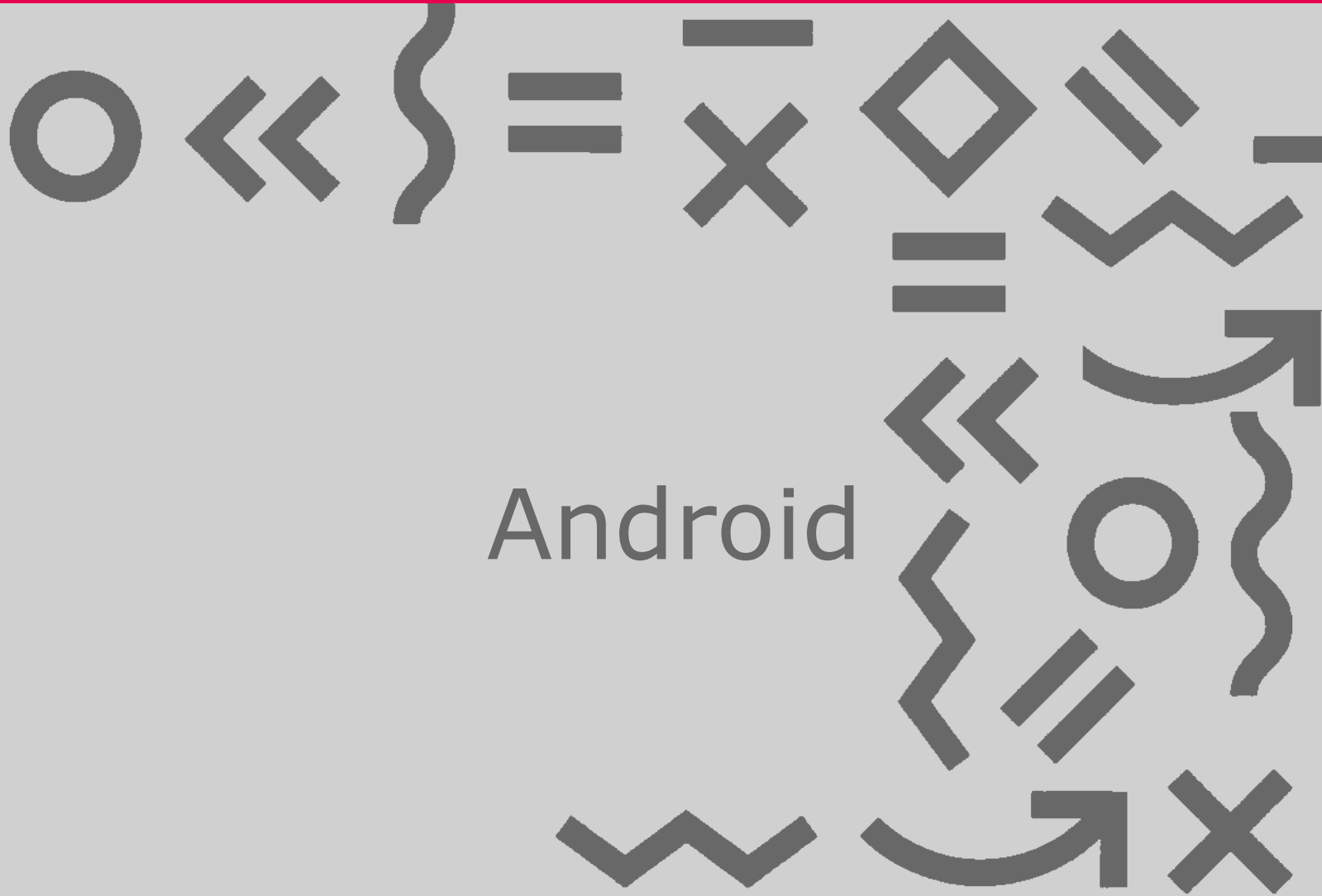


Web- und App-Programmierung Android und iOS

Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-sachsen.de

AUFBAU DER LEHRVERANSTALTUNG



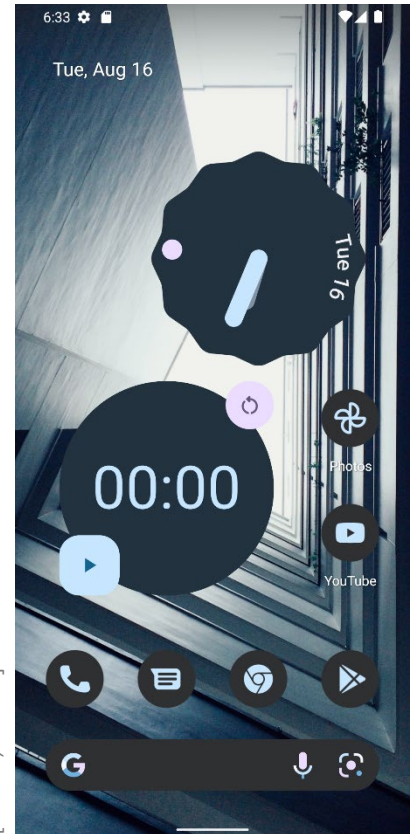


Android

ALLGEMEINES

- Entwickelt von der Open Handset Alliance, getrieben von Google
- Zielplattform: Smartphones und Tablets
- Unterstützung heterogener Hardware
- weitere Versionen für IoT-Geräte, Autos, Wearables, Fernseher, Spielekonsolen und Set-Top-Boxen
- zugrunde liegendes Betriebssystem: Linux 3.x
- Open Source (Apache 2.0, GPL 2.0)
- oft mit vorinstallierter proprietärer Google Software (Google Maps, ...)
- Programmierung von Apps in Kotlin und Java mittels Android Studio
- Android 1.0 (Beta) im November 2007; aktuell 15 (Versionen hatten von 1.5 bis 9.0 Namen von Süßigkeiten)

[AOSP; 2022]



Android iOS Zusammenfassung

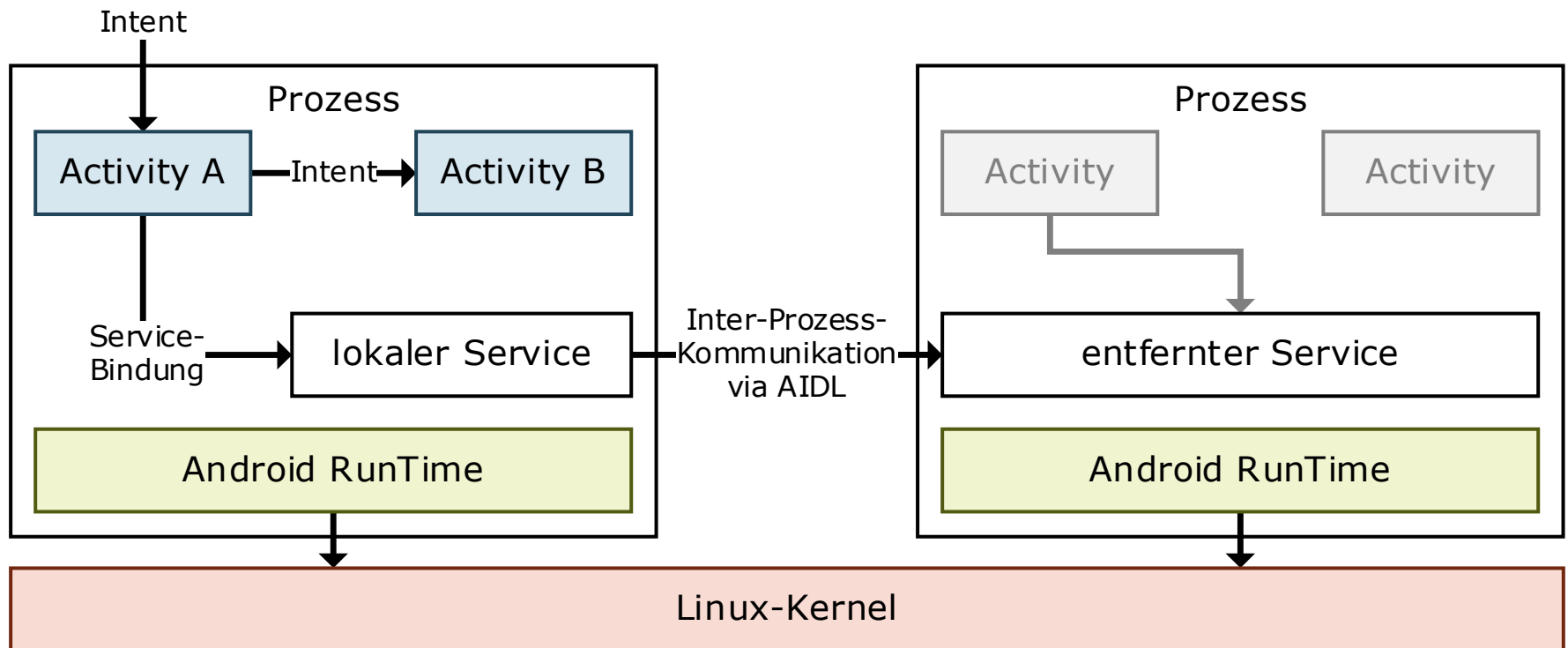
-
- The diagram illustrates the Android architecture stack, organized into layers from bottom to top:
- Hardware:** The base layer at the bottom.
 - Linux Kernel:** The second layer, providing the operating system foundation.
 - Applikations-Framework:** The third layer, which is divided into two main components:
 - Erweiterungs-bibliotheken (Extension Libraries):** Located on the left side of the framework layer.
 - Android RunTime + Basisbibliotheken (Android Runtime + Base Libraries):** Located on the right side of the framework layer.
 - Mobilgerät (Mobile Device):** The top layer, representing the user-facing applications, which are categorized into:
 - App .apk (legacy):** Legacy applications in APK format.
 - App .aab:** Modern applications in AAB (Android App Bundle) format.
 - App .pad (Spiele):** Games in PAK (Play App Bundle) format.
 - ...** Other application types.

RENDERSCRIPT UND VULCAN

- hardware-beschleunigte Grafik via CPU und GPU möglich
- seit Android 3 verfügbar; seit Android 12 deprecated: RenderScript
 - für rechenintensive, parallel verarbeitende Prozesse
 - eigene Abstraktionsschicht für heterogene Arbeit auf Multicore-CPU und Multicore-GPU (viele Hersteller erlauben nur noch CPU)
- seit Android 7: Vulkan (via Library von Google)
 - nur für GPU-Aufgaben
 - nutzt GPU-Beschleunigung

AUFBAU EINER ANDROID-APP

- drei Hauptbestandteile: **Activities**, **Services**, **Intents**
- Android-Apps werden in getrennten Prozessen ausgeführt
 - Inter-Prozess-Kommunikation via AIDL-Schnittstellen
 - verwendete Komponenten müssen in Manifest deklariert werden



Android

- Nachrichtenobjekt zur Veröffentlichung von Ereignissen oder zum Anfordern von Aktionen einer anderen Komponente
- enthält Aktionen und Daten
bspw. ACTION_VIEW content://contacts/people/1
→ Anzeige von Informationen zur Person mit der Identifikation „1“
- drei fundamentale Anwendungsfälle
 - Starten einer neuen Instanz einer Activity
 - Starten eines Service
 - Ausliefern eines Broadcasts, welcher von allen Apps auf dem Gerät empfangen werden kann

- explizit

- implizit

- 2-10

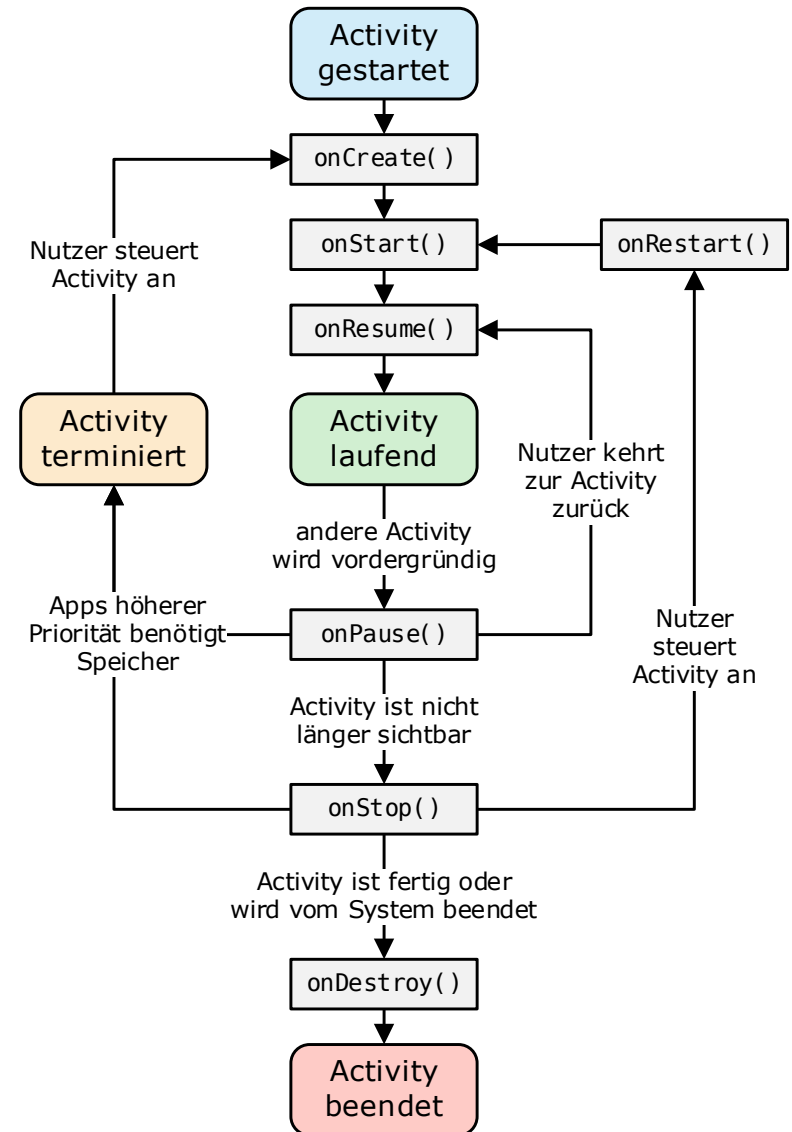
INTENT – BROADCASTS

- Empfangen von mittels `sendBroadcast()` gesendeter Intents
- System- und eigene Broadcasts
- Android vermittelt Ereignisse an Apps (Batterie niedrig, Funk aus, ...) bspw. `CONNECTIVITY_ACTION`, `AIRPLANE_MODE`
- bis Nougat: Deklaration von Broadcast-Empfängern im Manifest

```
<receiver android:name=".MeinBCEmpfaenger" android:exported="true">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED"/>
    <action android:name="android.intent.action.INPUT_METHOD_CHANGED"/>
  </intent-filter>
</receiver>
```
- seit Nougat: `registerReceiver(BroadcastReceiver, IntentFilter)` muss aufgerufen werden, bspw. für `CONNECTIVITY_ACTION`
- seit Oreo: weitere Einschränkungen für im Manifest deklarierte Empfänger und Anbieter

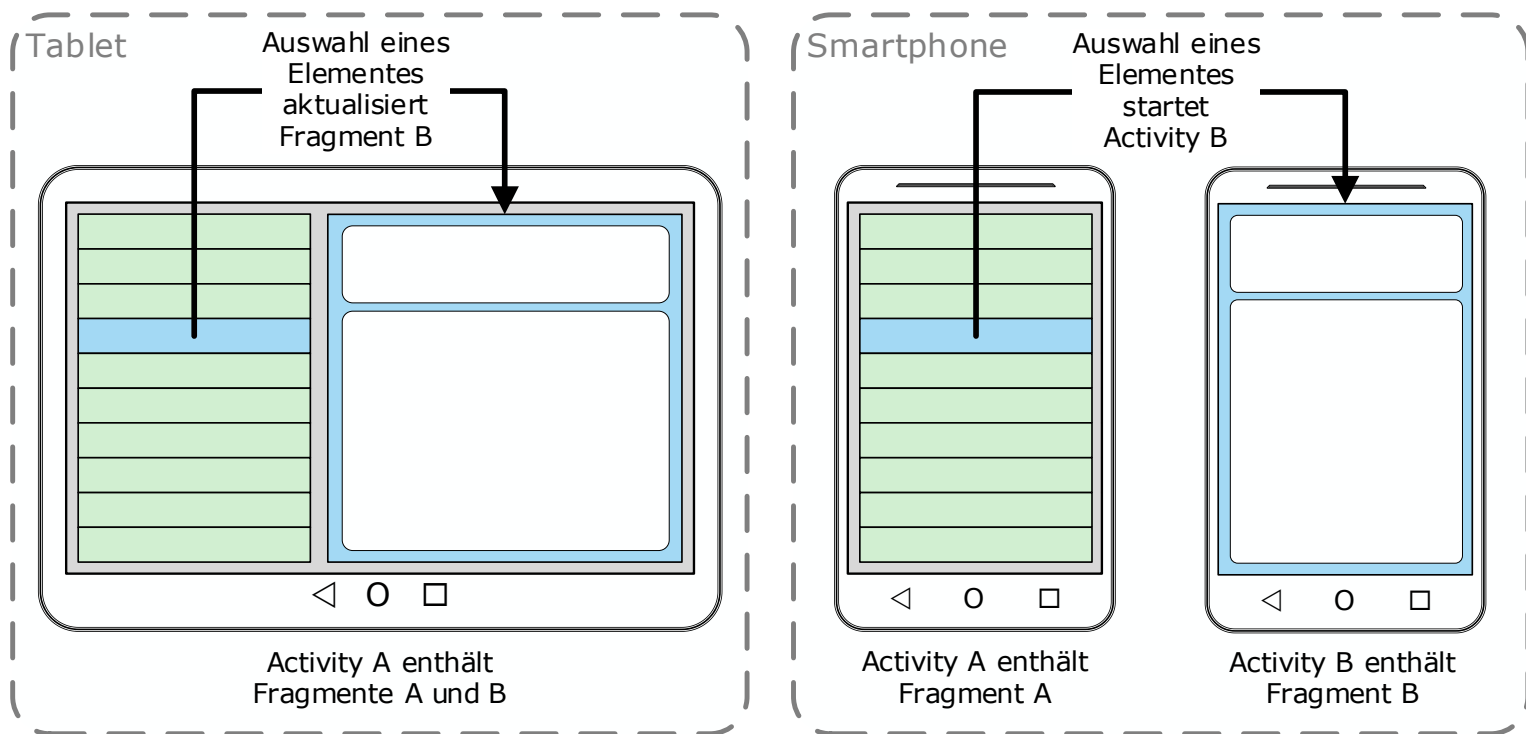
ACTIVITY

- repräsentiert eine einzelne Anzeige (**Screen**) einer App
- Umsetzung der Activity-Klasse (Erweiterung um eigene Methoden)
- enthält Nutzerschnittstellenelement (**Views**), die auf Ereignisse reagieren
- Android hält Activity-Stapel vor (bspw. für „zurück“)
- Methoden einer Activity spiegeln den Lebenszyklus wieder
- vier grundlegende Zustände
 - laufend (**Running**)
 - pausiert (**Paused**)
 - angehalten (**Stopped**)
 - beendet (**Destroyed**)



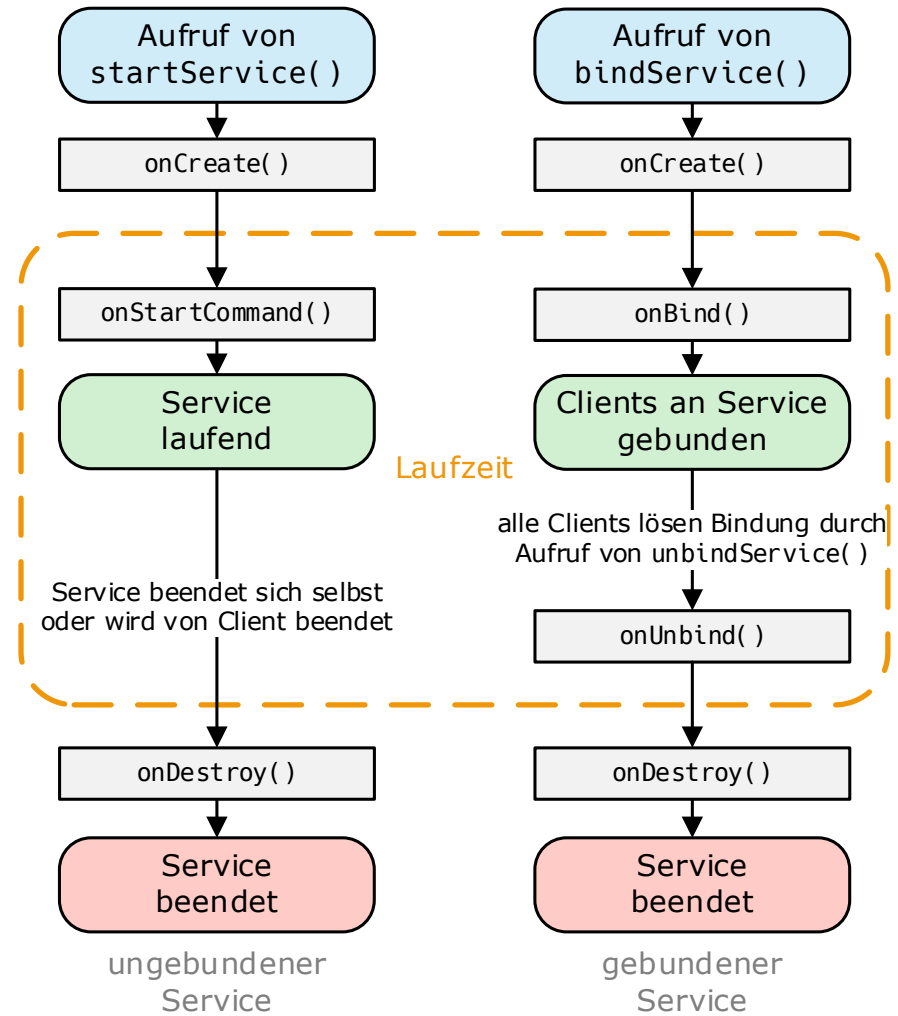
FRAGMENTS

- modulare Sektion einer Activity
- hat eigenen Lebenszyklus und eigene Eingabeereignisse
- kann zur Laufzeit der Activity beliebig hinzugefügt/entfernt werden
- kann in unterschiedlichen Activitys wiederverwendet werden



SERVICE

- Ausführung von Hintergrundaktivitäten (startService)
- Offenlegung von Funktionalität ggü. anderen Apps (bindService)
- läuft im Prozess erzeugender App
- **kein Thread**
- lokal und entfernt nutzbar
- wenn verbunden, wird via offenkommunizierender Schnittstelle kommuniziert
 - lokal: einfacher Java-Ruf
 - entfernt: Android Interface Definition Language (AIDL)



MANIFEST

- AndroidManifest.xml für jede App benötigt
- beschreibt Elemente einer App und wann diese erzeugt oder aktiviert werden
- enthält Liste der von der App benötigten und angebotenen Rechte
 - auf Basis dieser Liste wird Nutzer beim Installieren um Freigabe gebeten
 - nur vom Nutzer gewährte und im Manifest stehende Rechte sind nutzbar

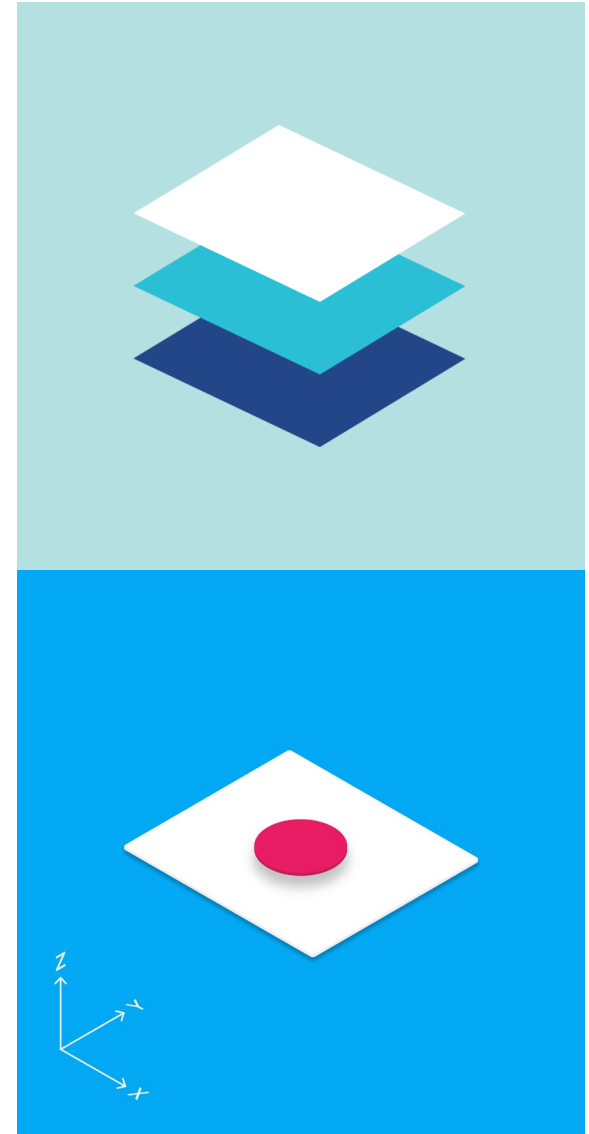
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.my_domain.app.helloactivity">
    <application android:label="@string/app_name">
        <activity android:name=".HelloActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

PAKETFORMATE DER APP-TYPEN

- Android Package (APK)
 - Legacy-Format für Apps, die bis Juli 2021 im Play Store bereitgestellt wurden
 - Standard-Format bei alternativen Stores wie F-Droid
- Android App Bundle (AAB)
 - seit August 2021 verpflichtendes Format im Play Store
 - Dynamic Delivery → kleinere Updates; On-Demand-Feature-Installation
- Play Asset Delivery (PAD), Play Feature Delivery (PFD)
 - ersetzt APK mit OBB-Assets im externen Speicher
 - erhöhte Sicherheit, da Zusatzressourcen durch Play Protect müssen
 - Adaptive Scalable Texture Compression
 - App und Zusatzressourcen müssen mit gleichem Schlüssel signiert werden (Upgrade auf APK Signature Scheme 3 mit Key Rotation)
 - für Spiele ab 150 MB verpflichtend

MATERIAL DESIGN: NATIVE OBERFLÄCHE VON ANDROID 5 BIS 11

- seit 2014 Vorgabe von Google
→ <https://material.io>
- wurde mit der Einführung von Android 12 durch *Material You* abgelöst
- Material als Metapher (Stift und Papier)
- GUI-Elemente erben Papiercharakteristiken
- flaches Design mit Ebenen (z-Achse)
- z-Position wird durch Erhöhungen und Schatten dargestellt
- Erhöhung wird durch Nutzerinteraktion geändert (bspw. Schaltfläche betätigen)
- umfangreiche Nutzung von Bildern und Animationen



MATERIAL DESIGN: NATIVE OBERFLÄCHE VON ANDROID 5 BIS 11

 MATERIAL DESIGN

Material Design

Motion

Style

Layout

Components

Bottom navigation

Bottom sheets

Buttons

Buttons: Floating Action Butt...

Cards

Chips

Data tables

Dialogs

Dividers

Expansion panels

Grid lists

Lists

Lists: Controls

Menus

Pickers

Progress & activity

Components – Cards

Cards

A card is a sheet of material that serves as an entry point to more detailed information.

Cards may contain a photo, text, and a link about a single subject. They may display content containing elements of varying size, such as photos with captions of variable length.

A card collection is a layout of cards on the same plane.

Usage

Cards display content composed of different elements whose size or supported actions vary.

Supported gestures

Swipe
Pick-up-and-move

Related components

[Grid lists](#)

For developers

[Android cards](#)
[Polymer cards](#)



Contents

Usage

Content

Behavior

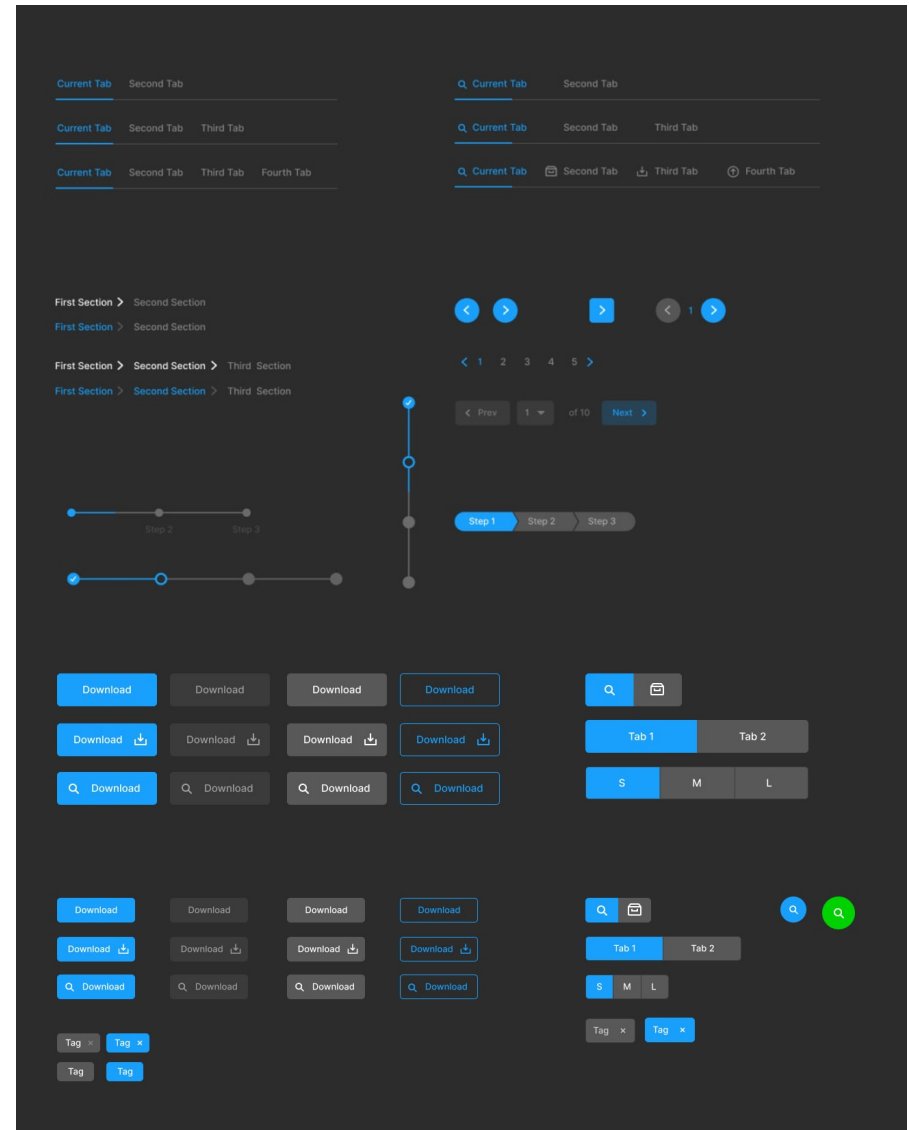
Actions

Content blocks

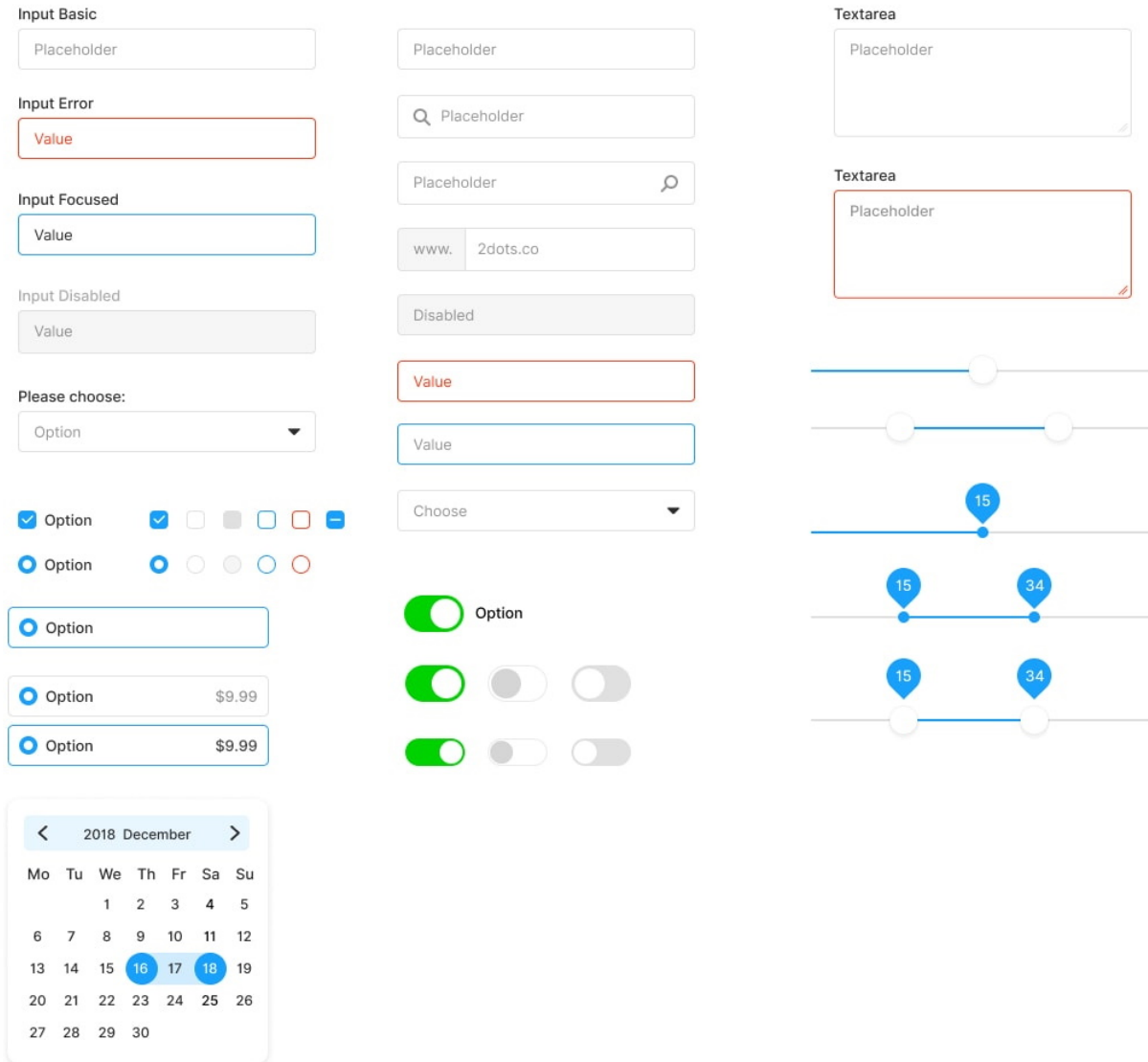
MATERIAL YOU: NATIVE OBERFLÄCHE AB ANDROID 12

- Anpassungen in den Farbpaletten (Pastelltöne)
 - dynamische Farbpalette
 - abgerundete Ecken
 - vergrößerte Tap Targets
 - neue (dynamische) Elementformen und -abstände
- <https://material.io/blog/announcing-material-you>

Android iOS Zusammenfassung



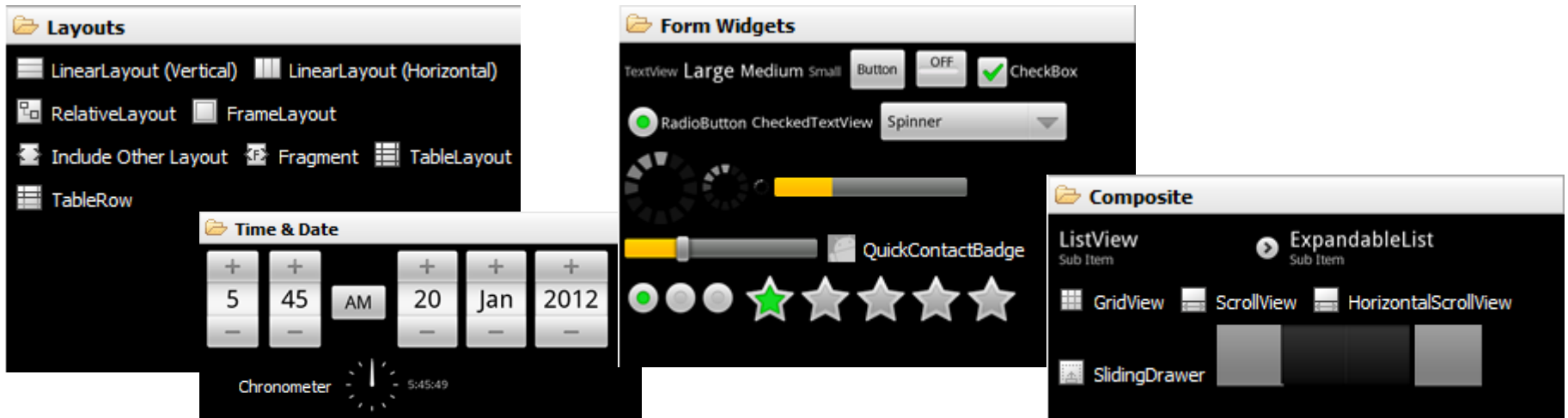
GUI-ELEMENTE



-
- ```

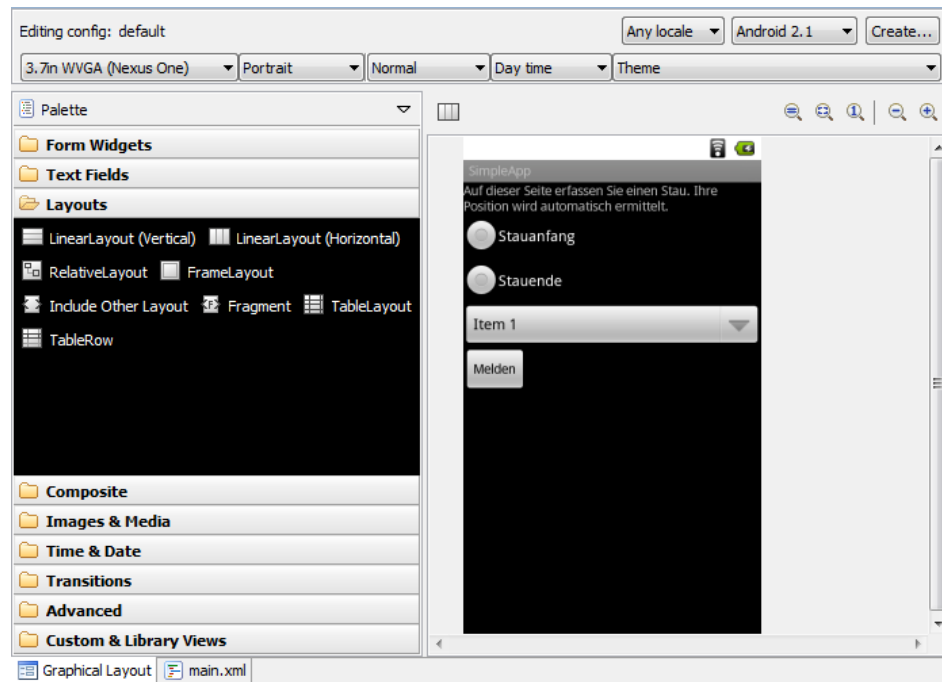
graph BT
 VG1[ViewGroup] --> VG2[ViewGroup]
 VG1 --> V1[View]
 VG1 --> V2[View]
 VG2 --> V3[View]
 VG2 --> V4[View]
 VG2 --> V5[View]

```



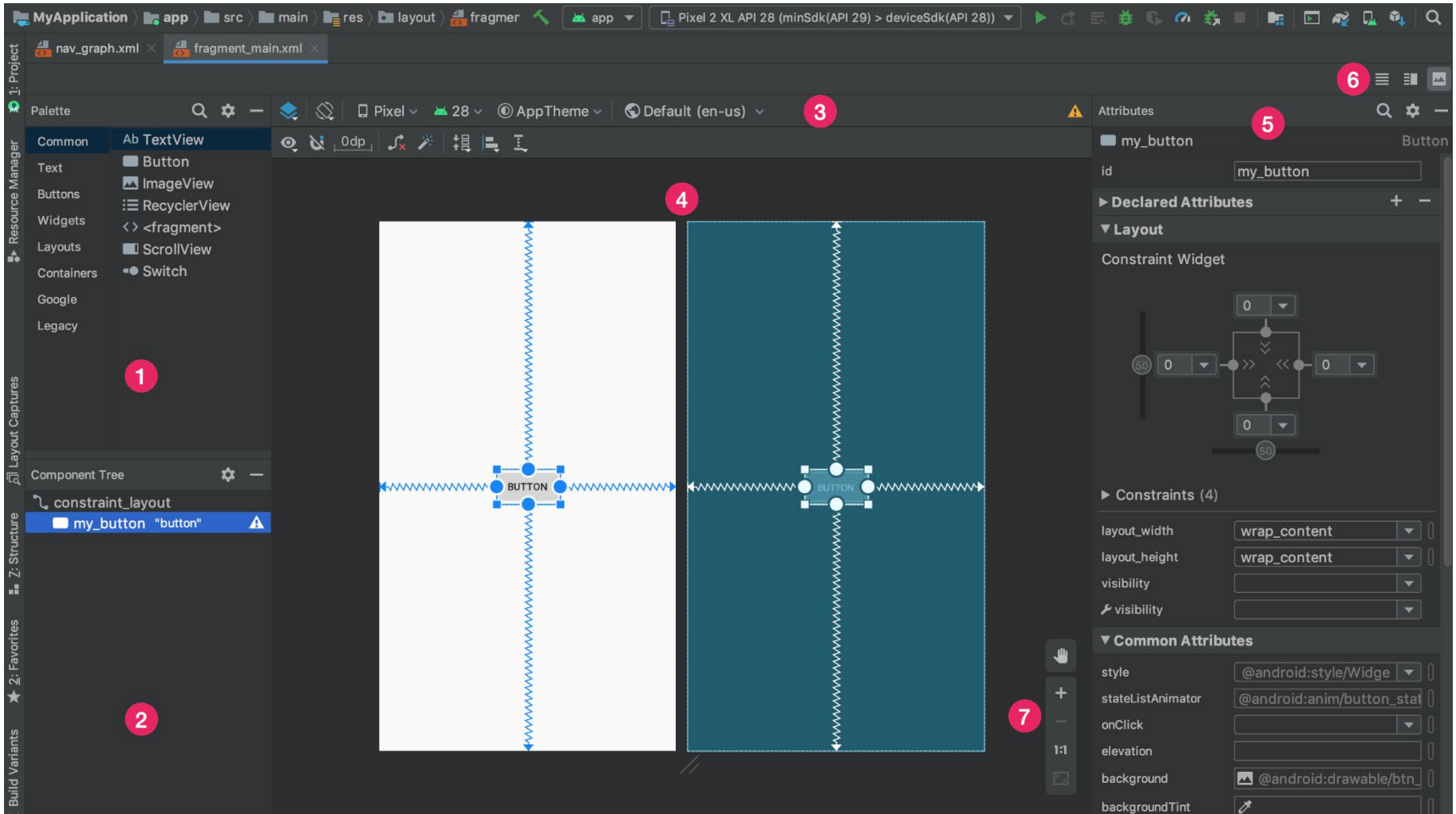
# GUI-ERZEUGUNG MIT XML

- auf XML basierende GUI-Beschreibung
- alternativ: Erzeugung der GUI im Code
- Unterstützung von Touch-Interaktionen
- ereignisbasierte Verarbeitung von Interaktionen  
(bspw. `view.setOnClickListener(callback)`)



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
 xmlns:android="http://schemas.android.com"
 android:orientation="vertical"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent">
 <TextView
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Auf dieser Seite ..." />
 <RadioGroup
 android:id="@+id/position"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:orientation="vertical">
 <RadioButton
 android:id="@+id/stauAnfang"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Stauanfang" />
 <RadioButton
 android:id="@+id/stauEnde"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="StauEnde" />
 </RadioGroup>
 <!-- weitere Elemente... -->
</LinearLayout>
```

# GUI-ERZEUGUNG MIT ANDROID STUDIO LAYOUT EDITOR



# NETZWERKKOMMUNIKATION

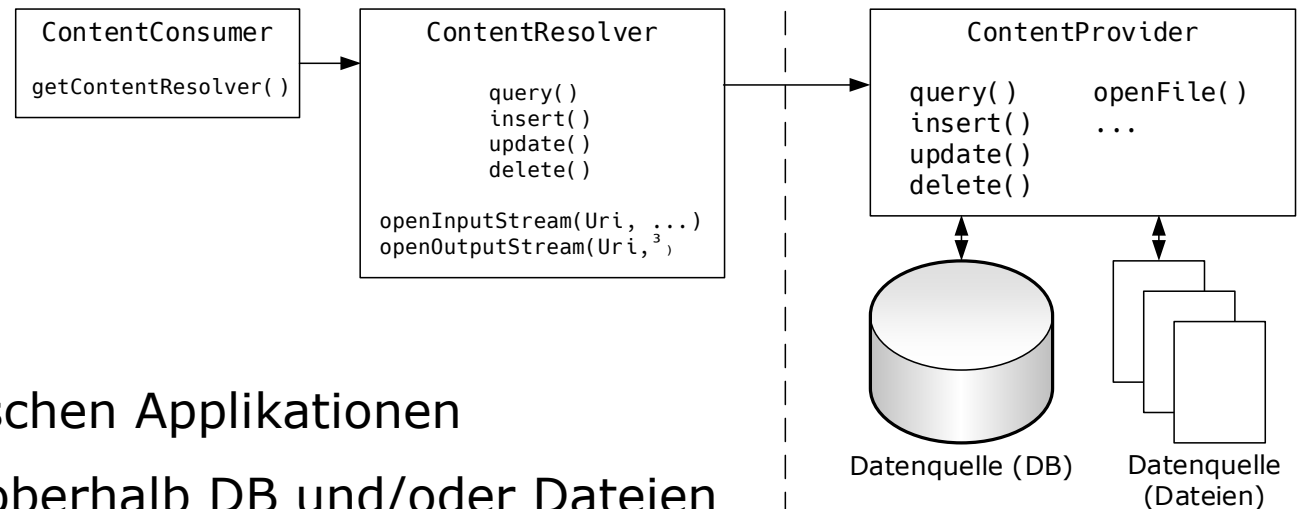
- `Android.net.ConnectivityManager`
  - überwacht Verbindungszustand
  - sendet Broadcast-Intent sobald sich Zustand ändert
  - stellt Methoden zum Zugriff auf Netzwerkzustand bereit
    - `getActiveNetworkInfo( )`
    - `getAllNetworkInfo( )`
    - `getNetworkInfo(int networkType)`
- `Java.net.*`-API können genutzt werden
  - werden als Teil der Android-Plattform vorgehalten
  - `HttpClient` als Basis für HTTP-Verbindungen
- alternative Bibliotheken auch verwendbar (bspw. Google Volley)

# PERSISTENTER SPEICHER

- SQLite
  - lokale Datenbank mit SQL- und Transaktionsunterstützung in einzelner Datei
  - Ausgangsklasse: `android.database.sqlite.SQLiteDatabase`
  - Abfragen mittels `SQLiteDatabase.query( )`
- gemeinsame Voreinstellungen
  - Ausgangsklasse: `android.content.SharedPreferences`
  - speichert primitive Daten in Schlüssel/Wert-Paaren
- interner Speicher für private Daten in Dateien (getrennt nach App-Id)
- (virtueller) externer Speicher für gemeinsame Daten in gemeinschaftlich zugreifbaren Dateien

## Teilen von Daten zwischen Applikationen

- 2-28

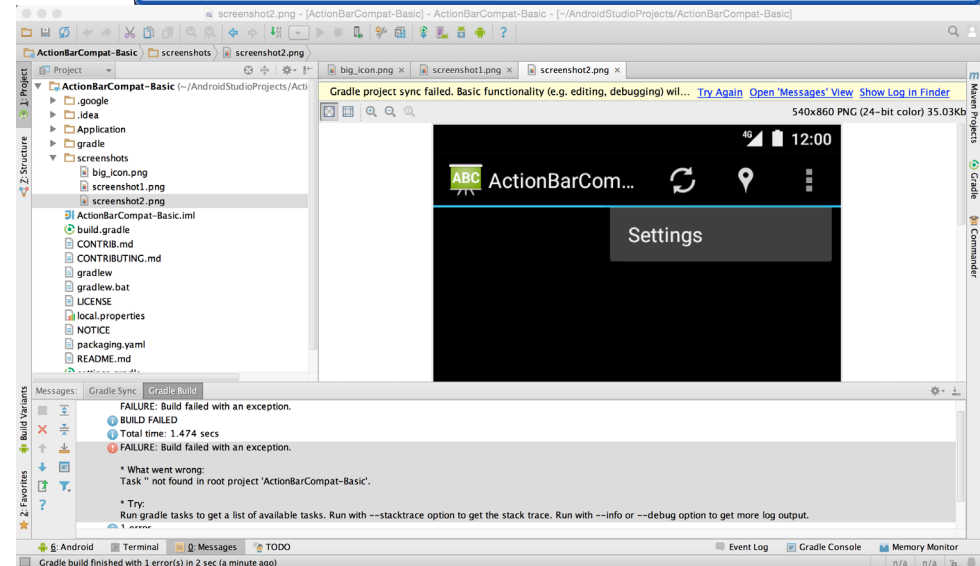
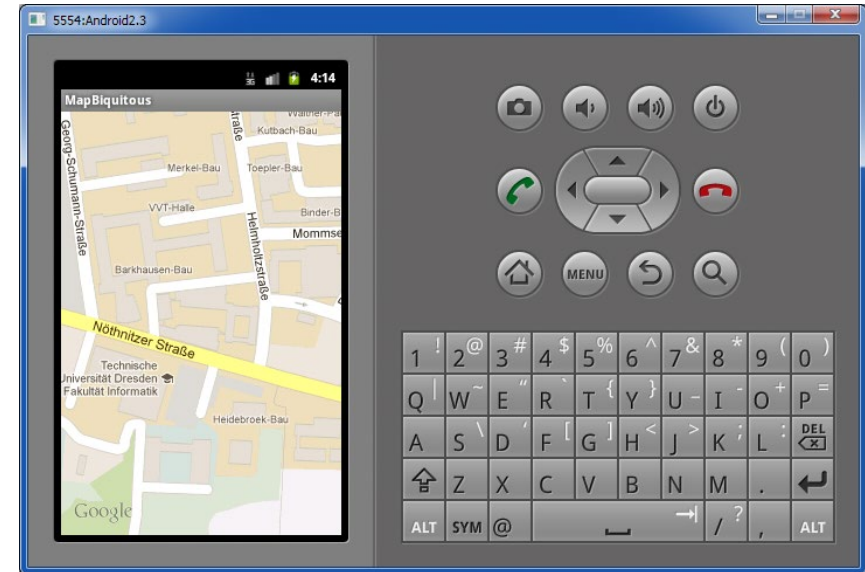
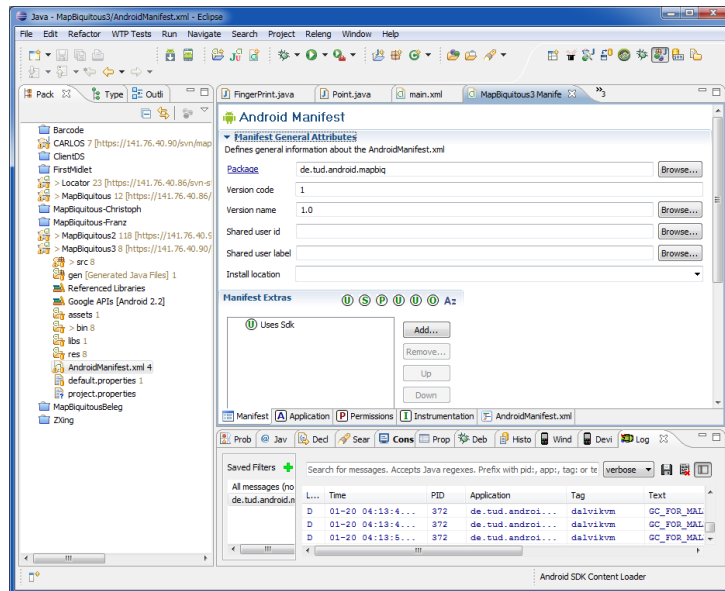


# ENTWICKLUNGSWERKZEUGE

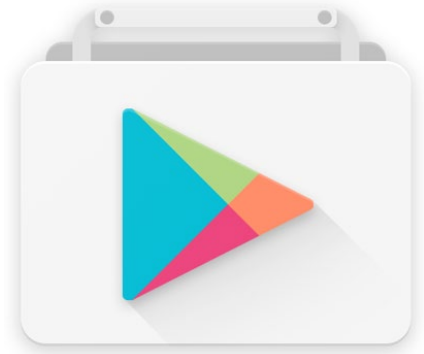
## Android Studio

- basiert auf IntelliJ IDE
- Projektverwaltung
- Geräteemulator
- Debugger

früher: Eclipse (geht immer noch)



# BEREITSTELLUNG UND BEZUG VON APPS



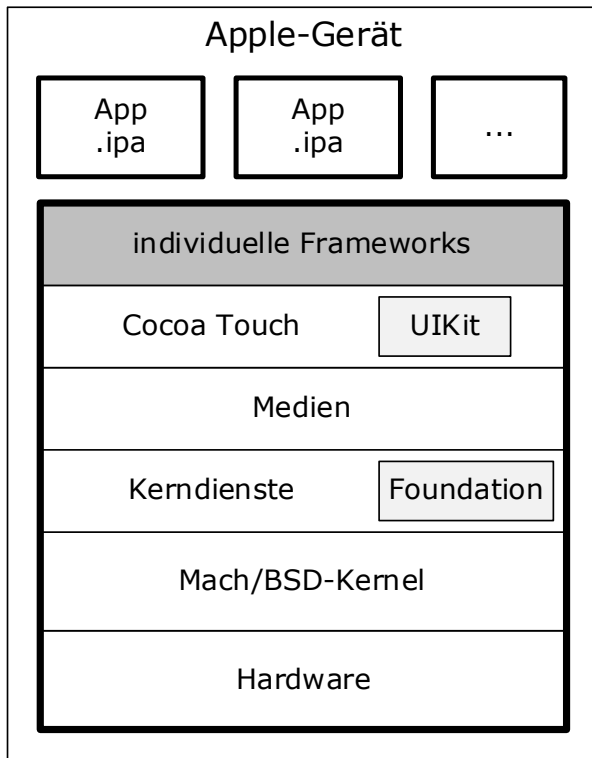
- Bereitstellung von Apps
  - im Google Play Store (verifiziert)
  - als APK auf Webseiten, ... („unbekannte Quelle“)
- Entwicklung verursacht nur Selbstkosten
- Bereitstellung im Play Store erfordert einmalige Registrierung als Entwickler (25 USD)
- Entwickler können Preis für App-Bezug festlegen; Google behält 30% des Umsatzes als Provision ein
- provisionsfreie Bereitstellung nur außerhalb des Play Stores (s.o.)

## SONSTIGES

- spezielle Eingabemethoden für Sehbehinderte:  
<https://support.google.com/accessibility/android/answer/9728765>
- Android wird oft mit unerwünschte Software (Bloatware) ausgeliefert;  
muss teilweise mühsam entfernt werden:  
<https://github.com/0x192/universal-android-debloater>
- Austauschbarkeit auch bei Kernkomponenten, z.B. Nutzung von  
Here Maps anstatt Google Maps API/SDK:  
<https://developer.here.com/products/here-sdk>



# ARCHITEKTUR



- Cocoa Touch: Objective-C-APIs für untere Ebenen (bspw. Multi-Touch, Kamera, Accelerometer)  
→ was am häufigsten verwendet wird
- Medien: OpenGL ES, Core Audio, OpenAL, PDF, PNG, JPG, TIFF, Quartz 2D  
→ auf Performance optimiert
- Kerndienste: Adressbuch, SQLite, Netzwerk, Lokation, Threading
- Kernel: BSD und Mach 3.0, Dateisystem, Energieverwaltung, Sicherheit  
→ sehr begrenzter Zugriff für Entwickler

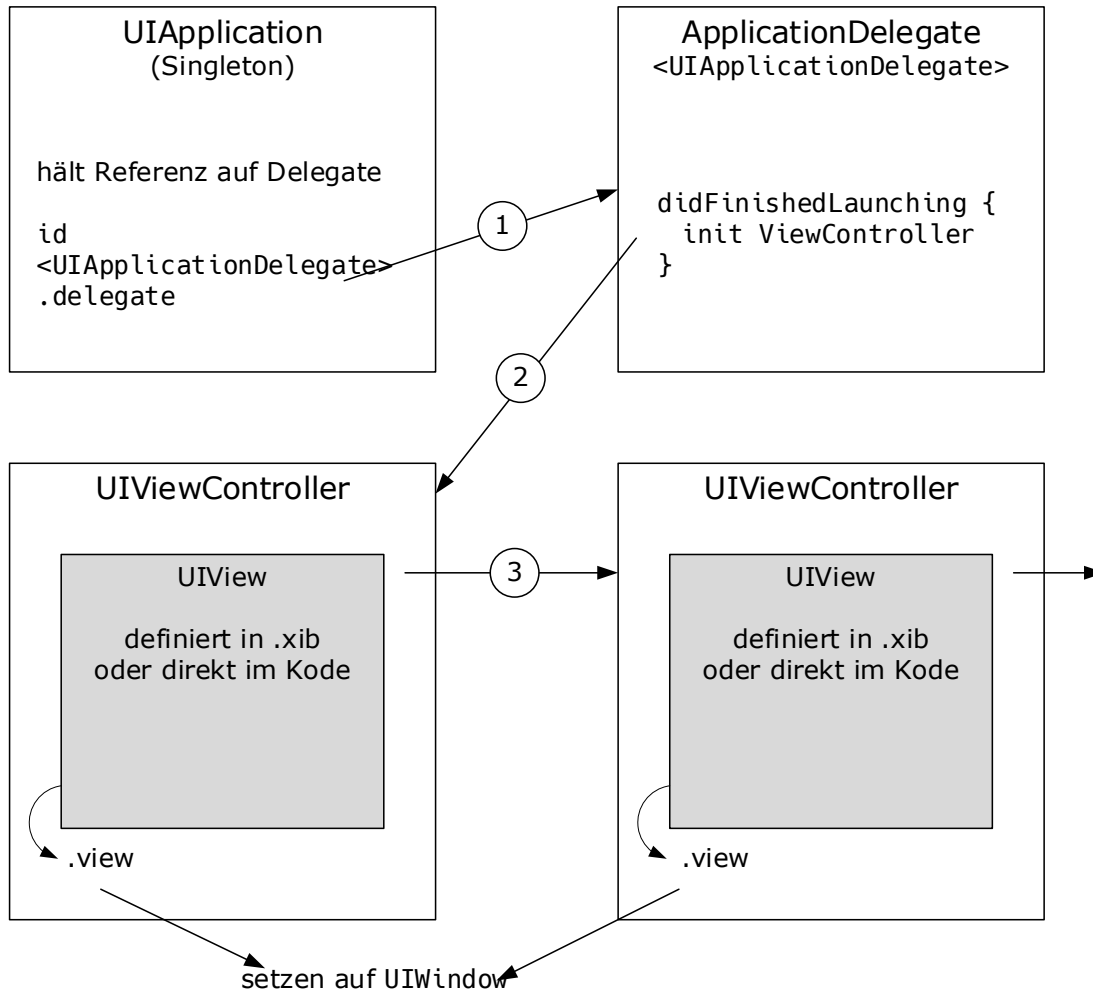
# OBJECTIVE-C VS. SWIFT

- Swift (Apple-eigene Sprache)
  - ersetzt Objective-C als Standardsprache für iOS-Entwicklung
  - durch Expressivität von Skriptsprache (JavaScript, Ruby) beeinflusst
  - leichtgewichtige Syntax
  - Typsicherheit und Typgeneralisierung (**Generics**)
  - keine Unterscheidung zw. Interface- und Implementierungsklassen
  - keine Pointer → Speichersicherheit
  - Playground → Just-in-Time-Evaluation von Code
- Swift-, Objective-C- und C-Code kann in einem Projekt vermischt werden

# ANATOMIE EINER IOS-APPLIKATION

- Cocoa-Touch-Framework
  - **Foundation** – Basisklassen für Apps (NS-Präfix)
    - Datentypen und -strukturen (NSStrings, NSArray, ...)
    - Dienste und Funktionskapselung (Datum, Kalender, Timer, ...)
  - **UIKit** – Objekte mit GUI-Bezug („Views“)
- alle Applikationen basieren auf drei Entwurfsmustern
  - Model-View-Controller – allgemeine App-Struktur
  - Delegation – erledigt Informationstransfer zwischen Objekten
  - Target-Action – übersetzt Nutzerinteraktionen in Laufzeitkontrollcode

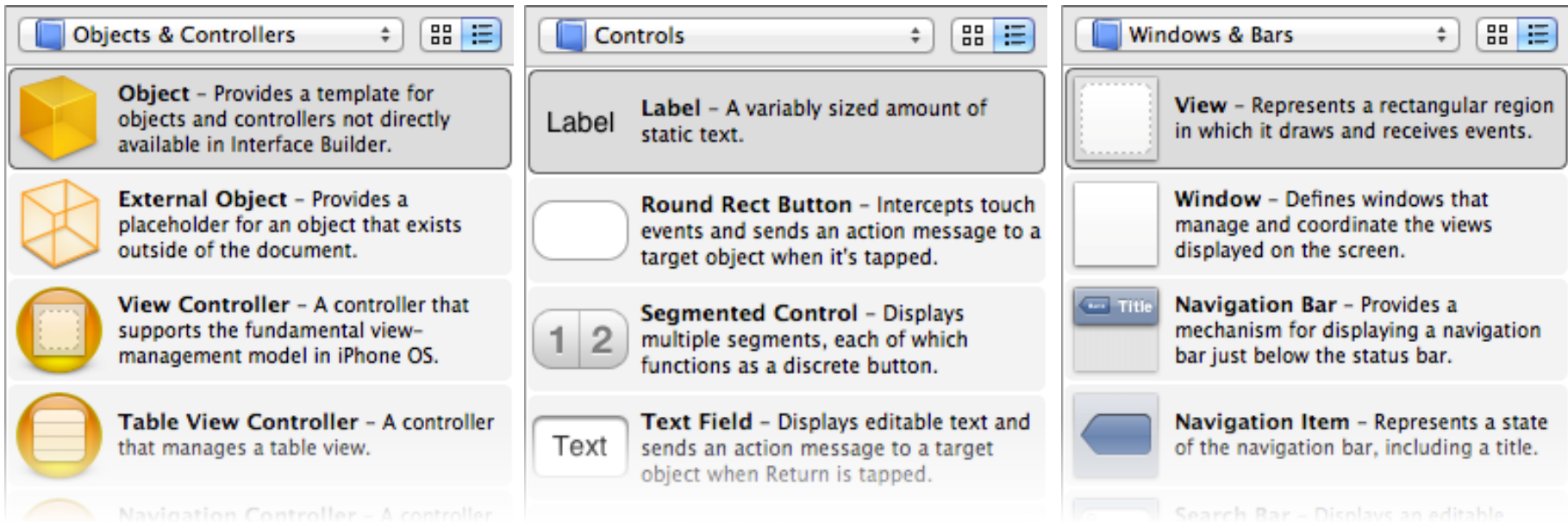
# ANATOMIE EINER IOS-APPLIKATION



- ähnlich zu Activities unter Android: ein Screen pro ViewController
- AppDelegate handhabt grundlegende Aufrufe im Applikationslebenszyklus und initiiert ViewController für ersten View

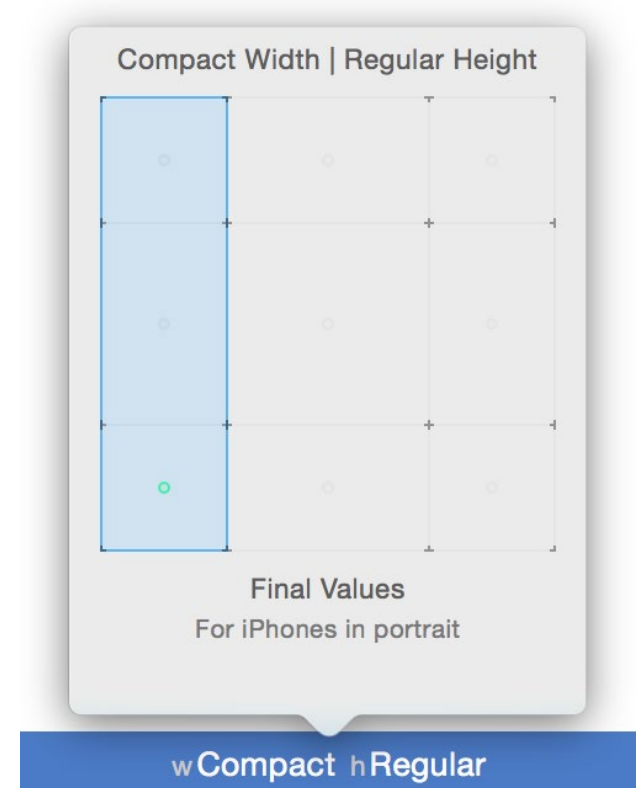
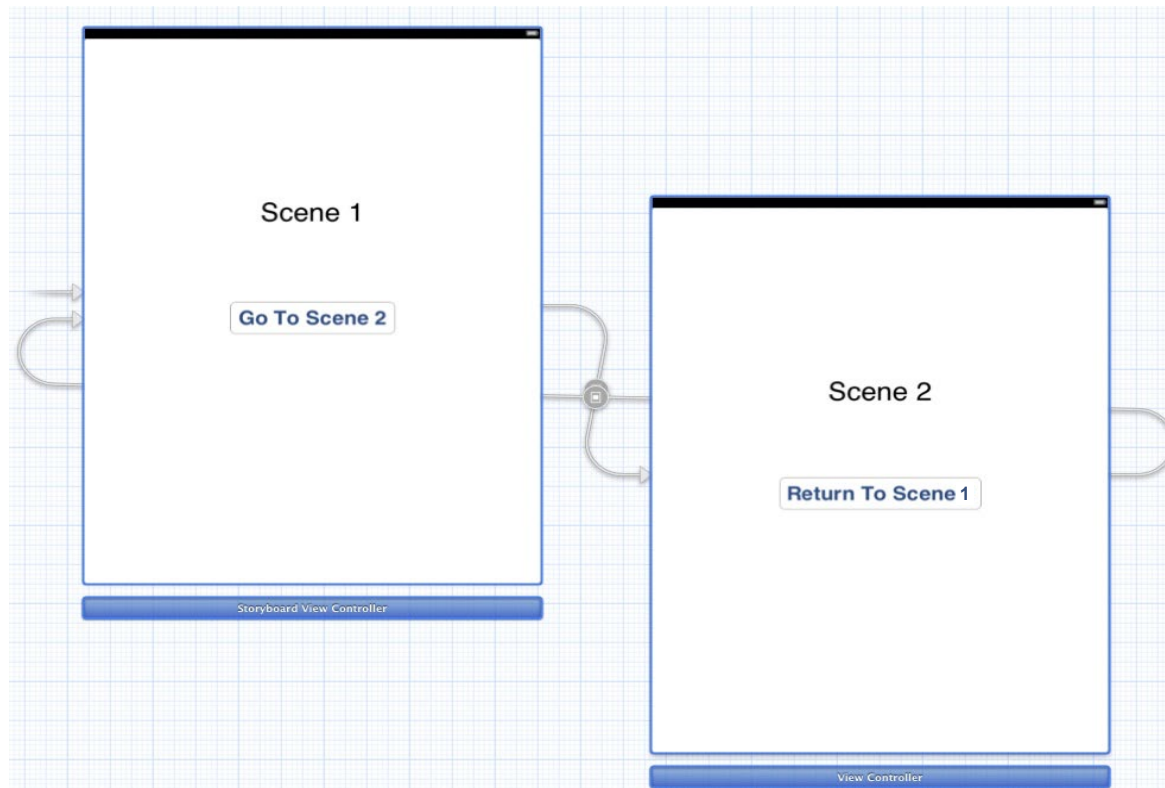
# ERZEUGUNG DER IOS-GUI

- basierend auf Interface Builder
  - .xib-Dateien beschreiben View-Hierarchien (Binärdatei: .nib)
  - i.d.R. eine .xib je Screen
  - Erzeugung/Änderung mittels Interface Builder, nicht direkt in .xib/xml
- UIKit-Klasse enthält vordefinierte Views, ViewController und Controls



# ERZEUGUNG DER IOS-GUI – STORYBOARDS

- graphische Bearbeitung einzelner Screen-Inhaltsplatzhalter (**Scene**)
- Übergang zwischen Scenes beschreibt Transition zwischen Inhalten
- erlaubt Definition eines Workflows (**Storyboard**) für wechselnde Layouts (Orientierung oder iPhone vs. iPad vs. iPod)

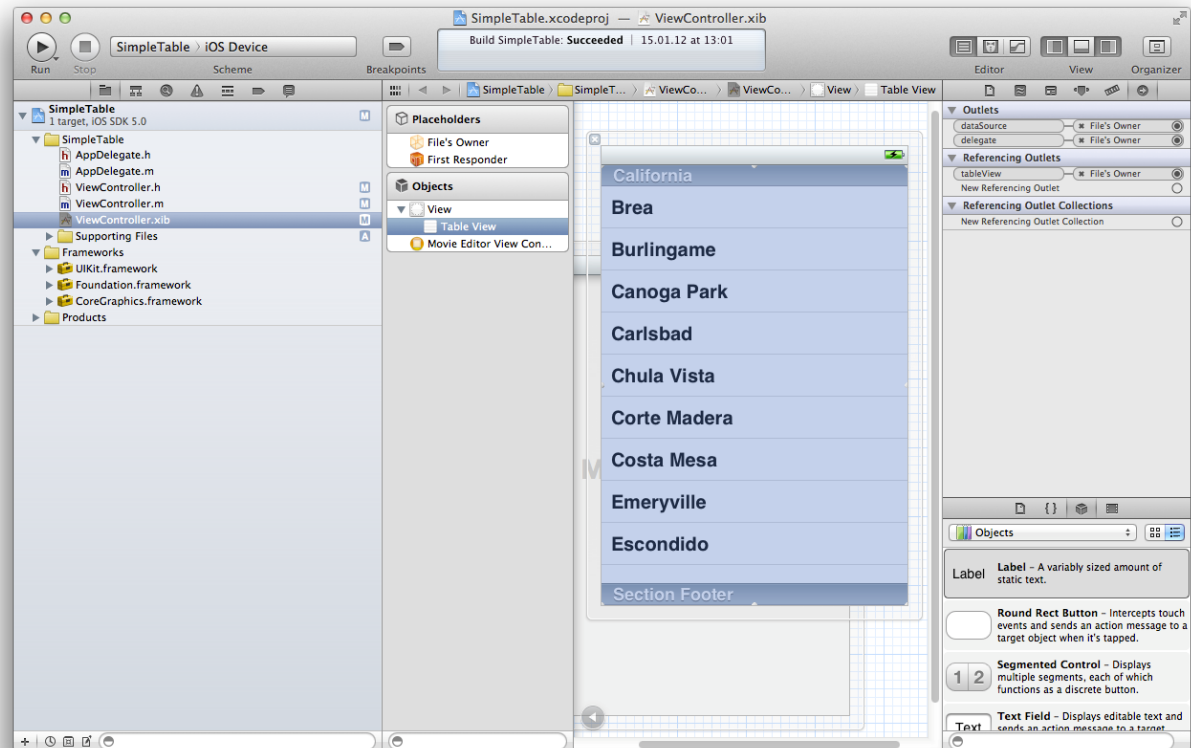


# PERSISTENTER SPEICHER IN IOS

- Schlüssel/Wert-basierter Speicher
  - simpler mit Dateien synchronisierter Hash (NSUserDefaults)
  - automatisch vom System synchronisiert
  - in Sandbox der jeweiligen App gespeichert
- Framework Core Data
  - Abstraktionsebene zum persistenten Speichern von Objekten
  - kann auf SQLite oder Datei (Sandbox oder global) abgebildet werden
- iCloud
  - Schlüssel/Wert-basierter Speicher in iCloud
  - Datenobjekte erben von UIDocument; können daher einfach in iCloud synchronisiert werden

# ENTWICKLUNGSUMGEBUNG FÜR IOS: XCODE

- IDE für Mac- und iOS-Entwicklung
- Projektverwaltung
- Kodeeditierer
- Compiler  
(zum Gerät oder in den Simulator)
- Debugger  
(vom Gerät oder aus dem Simulator)
- Repository-Verwaltung
- Performance-Tuning



# Zusammenfassung und Aufgaben

# PLATTFORMVERGLEICH

|                          | Android                               | iOS                                             |
|--------------------------|---------------------------------------|-------------------------------------------------|
| Anbieter                 | Google, Inc.                          | Apple, Inc.                                     |
| aktuelle Version         | 15 / API 35                           | 17.6.1 / 18.0RC / 18.1B3                        |
| Geräte-Hardware          | diverse Hersteller und Gerätetypen    | nur Apple-Geräte (iPod, iPhone, iPad, Apple TV) |
| Betriebssystem           | Linux-Kernel                          | Mach/BSD-Kernel                                 |
| Laufzeitumgebung         | Android RunTime (ART)                 | nativer Kode auf Mach/BSD-Kernel                |
| Programmiersprache       | Kotlin, Java                          | Swift, Objective-C                              |
| Open Source              | Apache 2.0, GPL 2, manches proprietär | nein                                            |
| Entwicklerbeschränkungen | keine                                 | Apple-Hardware und OS notwendig                 |
| Entwicklerprogramm       | einmalig 25 USD                       | generell: 99 USD/a, Enterprise: 299 USD/a       |

# AUFGABEN

- Recherchieren/Suchen Sie Informationen zu *Dalvik*. Diskutieren Sie dann mit Ihren Kommilitonen die Unterschiede zwischen Dalvik und ART.
  - Weshalb war die Umstellung sinnvoll?
  - Welche Vorteile und Nachteile ergeben sich für Endanwender?
  - Welche Vorteile und Nachteile ergeben sich für Entwickler?
- Erstellen Sie eine Demo-App für Android mit üblichen GUI-Elementen. Beachten Sie die Hinweise auf <https://material.io> !
- Ergänzen Sie Ihre Demo-App um die Speicherung eingegebener Daten. Demonstrieren Sie die Speicherung sowohl in Dateien als auch in Datenbanken.

# REFERENZEN

<https://developer.android.com>

<https://developer.android.com/guide/components/aidl.html>

<https://material.io>

<https://developer.apple.com/>

<https://developer.apple.com/ios/human-interface-guidelines>