Vorlesung Kommunikationssysteme
Wintersemester 2023/24

Programmierung mobiler Anwendungen mit Android

Christoph Lindemann

Zeitplan

Nr.	Datum	Thema
01	18.10.24	Organisation und Internet Trends
02	25.10.24	Programmierung mobiler Anwendungen mit Android
	01.11.24	Keine Vorlesung
03	08.11.24	Protokolldesign und das Internet
04	15.11.24	Anwendungen und Netzwerkprogrammierung
05	22.11.24	LAN und Medienzugriff
06	29.11.24	Ethernet und drahtlose Netze
07	06.12.24	LAN Komponenten und WAN Technologien
08	13.12.24	Internetworking und Adressierung mit IP
09	20.12.24	IP Datagramme
10	10.01.25	Zusätzliche Protokolle und Technologien
11	17.01.25	User Datagram Protocol und Transmission Control Protocol
12	24.01.25	TCP Überlastkontrolle / Internet Routing und Routingprotokolle
13	31.01.25	Ausblick: TCP für Hochgeschwindigkeitsnetze
14	07.02.25	Review der Vorlesung

Überblick

Ziele:

- Entwicklung mobiler Anwendungen mit Android kennenlernen

Themen:

- Einführung Android
  - Vision
  - Konsortium
  - Android Developer Challenge
- Entwicklung mit Android
  - Architektur
  - Android Software Development Kit

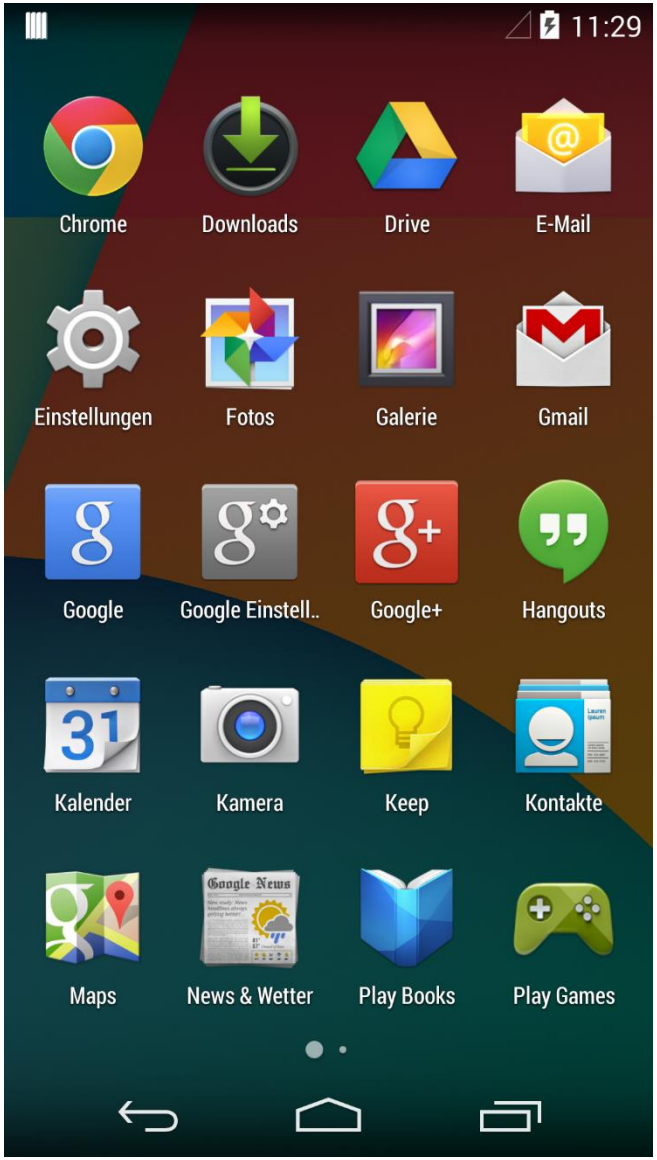
Einführung
Google Android

Vision

- 2017: Marktanteil über 80% weltweit
- Schwellen- und Entwicklungsländer: Mobile Internetzugang dominiert drahtgebundenes Internet
  - Vergleichsweise kostengünstige Netzinfrastruktur
- Eric Schmidt, CEO Google
  - „We want to make sure the thing you're looking for is on Google 100 percent of the time"
  - „In a few years, mobile advertising will generate more revenue than advertising on the normal Web"

Was ist Google Android?

- Software Framework für mobile Endgeräte
  - Betriebssystem, Middleware, Anwendungen
- Nicht an spezifische Hardware-Plattform gebunden
- Erstes Open-Source Framework
  - Apache License
- Basiert auf Linux Kernel
- Programmsprache Java
- Ursprünglich von Google entwickelt, Inzwischen durch Open Handset Alliance



Wichtige Hersteller

- Samsung
- HTC
- LG
- Motorola
- Huawei
- Sony
- Etc.

Open Handset Alliance (1)

- Industriekonsortium:
  - Mobilfunkanbieter
    - T-Mobile, Vodafone, NTT DoCoMo, …
  - Mobiltelefon-Hersteller
    - Samsung, Sony Ericsson, Motorola, …
  - Halbleiter-Hersteller
    - Intel, Atheros, Texas Instruments, …
  - Software- und Internet-Anbieter
    - Google, Ebay, …
- Ziel:
  - Schaffung einer offenen Plattform für mobile Internet-Anwendungen

Open Handset Alliance (2)

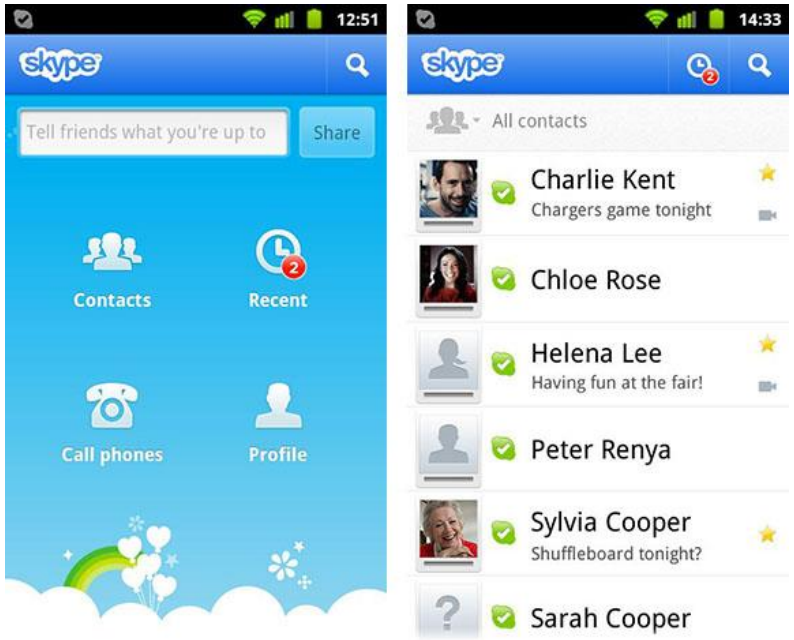
- Open Handset Alliance besteht aus über 80 Firmen
- Federführende Entwicklung von Android erfolgt durch Google
  - Andere Hersteller führen lediglich Anpassungen durch, z.B.
    - Überarbeitete Benutzeroberfläche
    - Spezielle Eingabe-Methoden für Touch-Screens
- Google verfolgt "Follow the free" Strategie
  - Produkt wird umsonst an möglichst viele Nutzer verteilt (Nutzer = Hersteller von mobilen Endgeräten)
  - Gewinn wird durch Werbung / Nutzung von Diensten erzielt
    - Dienste wie Goolge Search / Maps eng mit Android verzahnt

Beispiele für Apps (1)

- WhatsApp
  - Instant Messaging
  - Klassischer Client/Server Dienst
  - 2009 erschienen
  - Für Android, iOS, Symbian und Windows Phone verfügbar
  - > 450 Millionen Nutzer
  - 2014 an Facebook verkauft

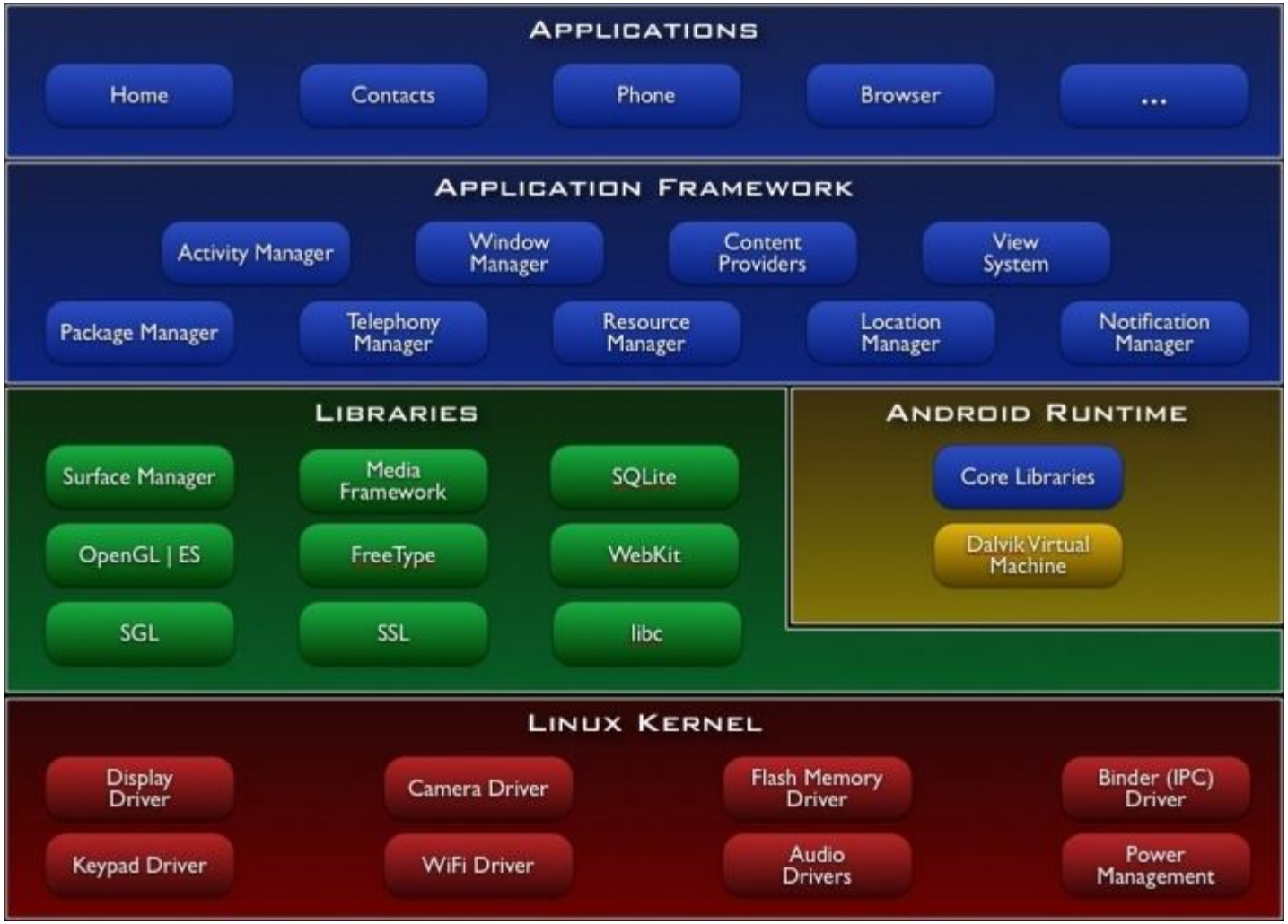
Beispiele für Apps (2)

- Skype
  - Software für IP-Telefonie und Instant Messaging
  - Dezentrales Peer-to-Peer System
  - Für PC seit 2003 verfügbar
  - Seit 2010 für Android
  - Beispiel für zahlreiche Programme die ursprünglich für PC entwickelt und nun auch als App verfügbar



Architektur von Android Anwendungen

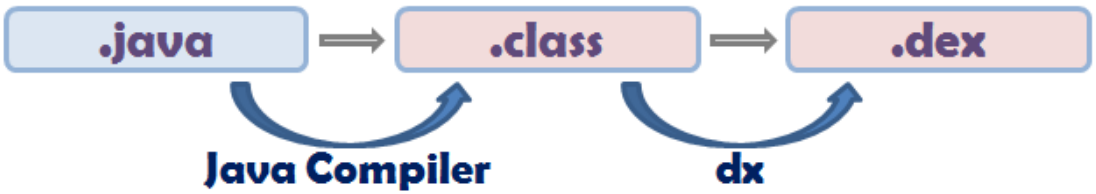
Aufbau Android Framework

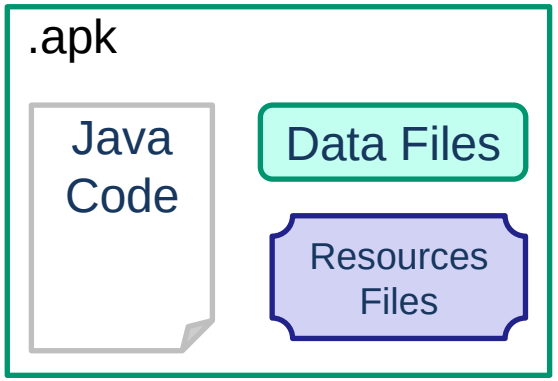


Grundlagen (1)

- Android Anwendungen sind in größtenteils in Java geschrieben
  - Kompilierte Programmcode, Daten (z.B. Kartendaten) und Ressourcen (z.B. Bilder, Sounds ..) werden zu Android package zusammengepackt

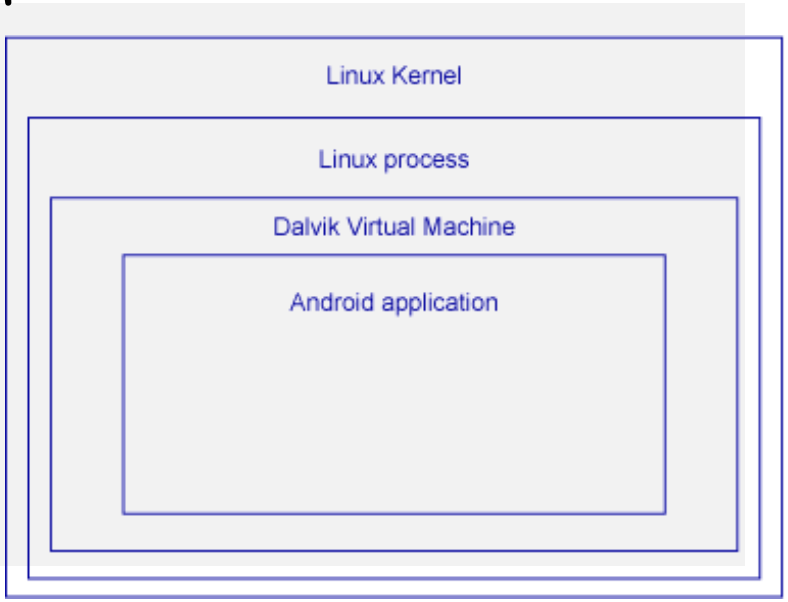
- Distribution von Android Anwendungen erfolgt über Android Package (apk)





Grundlagen (2)

- Android Anwendungen werden nicht direkt auf Hardware lauffähig
  - Anwendungen laufen auf virtueller Maschine (Dalvik Virtual Machine) bzw. seit Android 5.0 auf ART
- Jede Anwendung läuft in eigener virtueller Maschine
  - Isolation/Schutz von Anwendungen
- Jede virtuelle Maschine wird in einem Linux Prozess ausgeführt



Grundlagen (3)

- Anwendungen können gegenseitig Komponenten nutzen, z.B.
  - Navigations-Anwendung stellt Komponente zur Anzeige von Stadtpläne bereit
  - Adressbuch-Anwendung kann diese Komponente nutzen, um Kontakte anzuzeigen

- Die Nutzung erfolgt vollkommen dynamisch zur Laufzeit
  - Keine Code-Inlining
  - Keine Linking zur Übersetzungszeit

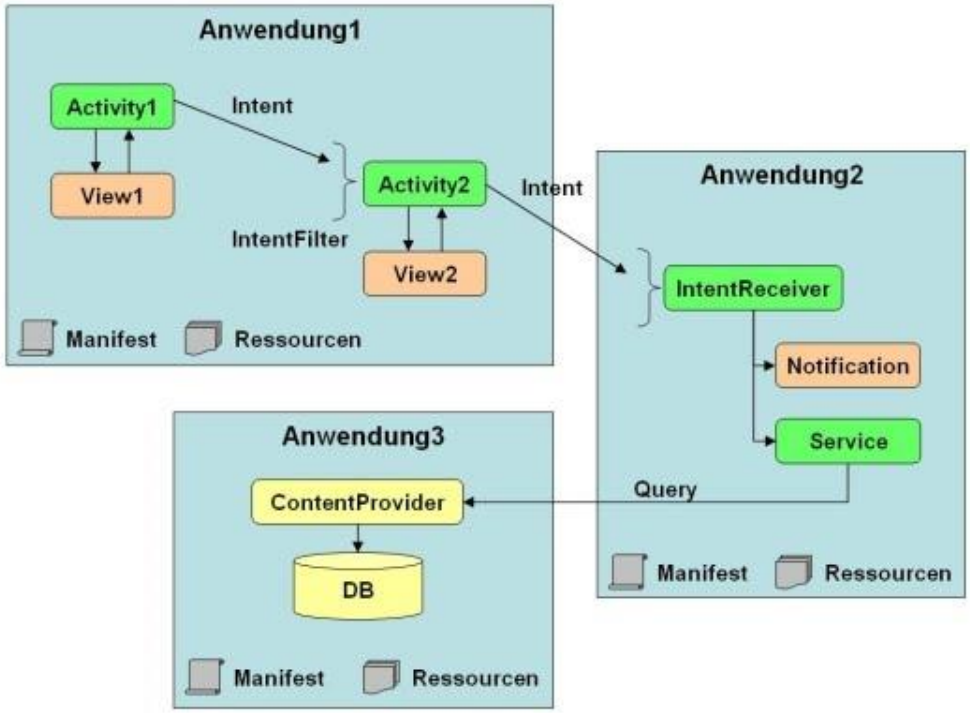
Grundlagen (4)

- Anwendungen besitzen daher mehrere Eintrittspunkte
  - Keine zentrale Main-Funktion
  - Einzelne Komponente können direkt gestartet und beendet warden

- 4 verschiedene Arten von Komponenten
  - Activities, Services, Content Provides, Broadcast Receivers

Grundlagen (5)

- Komponenten
  - Activity
    - GUI-Elemente für die Nutzer-Interaktion
  - Service
    - Hintergrundprozesse
  - Content Provider
    - Austausch von Daten zwischen Anwendungen
  - Broadcast Receivers
    - Erzeugung und Empfang von Events
- Nachrichten
  - Intents

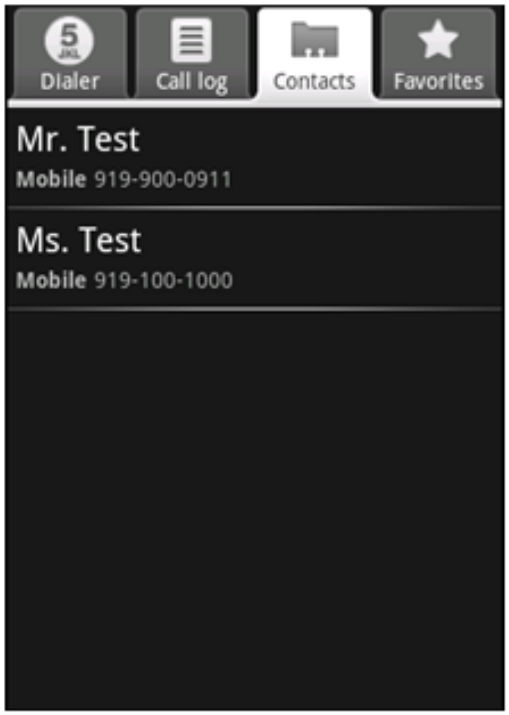


Activities (1)

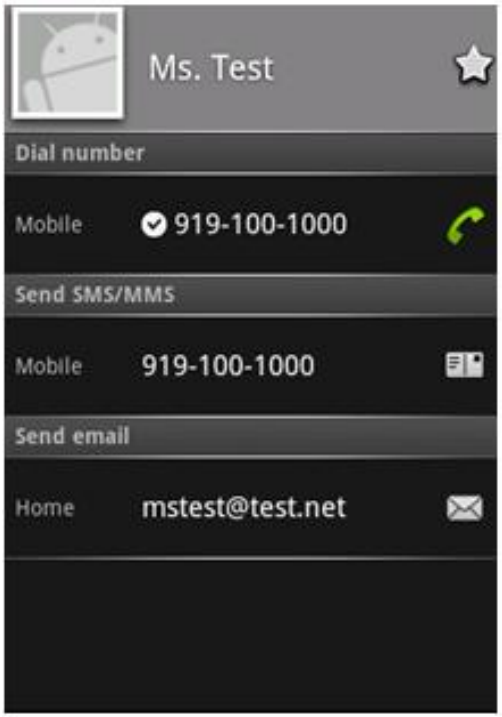
- Activities sind GUI-Komponenten für eine bestimmte Anwender-Aktionen, z.B.
  - Komponente zur Anzeige einer Auswahliste
  - Komponente zur Anzeige von Bilder
- Anwendungen besteht aus mehreren Activities, z.B.
  - Eine SMS-Anwendung besitzt 3 Activities
    - Anzeige der Kontaktliste
    - Verfassen einer Nachricht
    - Anzeige eingegangener Nachrichten
- Activities können unabhänigig voneinander genutzt werden

Activities (2)

- Beispiel Activities



Kontaktliste



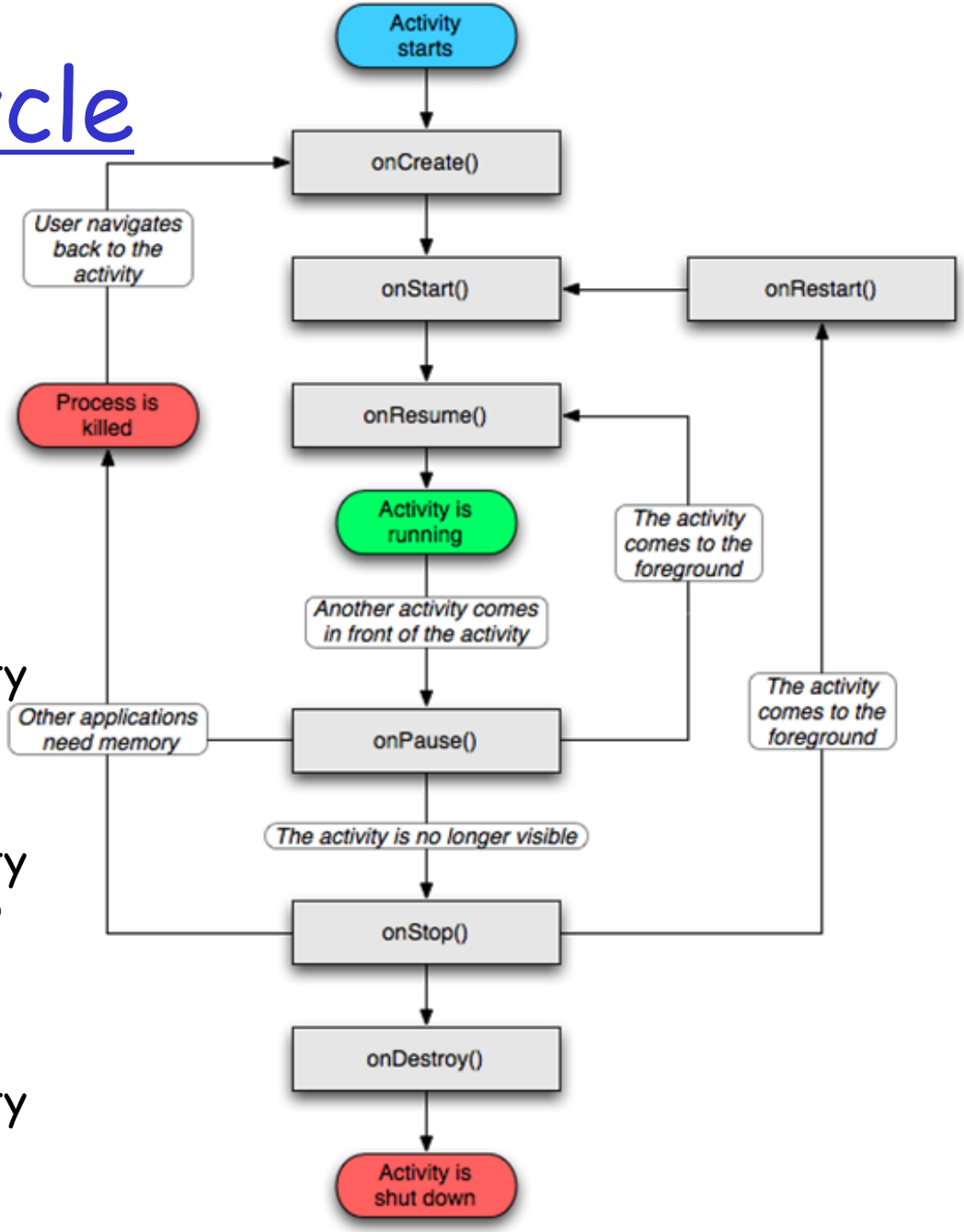
Detailanzeige Kontakt

Activity Lifecycle

- 3 verschiedene Zustände für Activity möglich
  - Running
    - Activity wird im Bildschirmvordergrund angezeigt und hat Eingabefokus
  - Paused
    - Activity ist noch teilweise sichtbar, aber andere Activity hat Eingabefokus
  - Stopped
    - Activity ist nicht sichtbar
- Activity mit Status paused oder stopped können vom System beendet werden

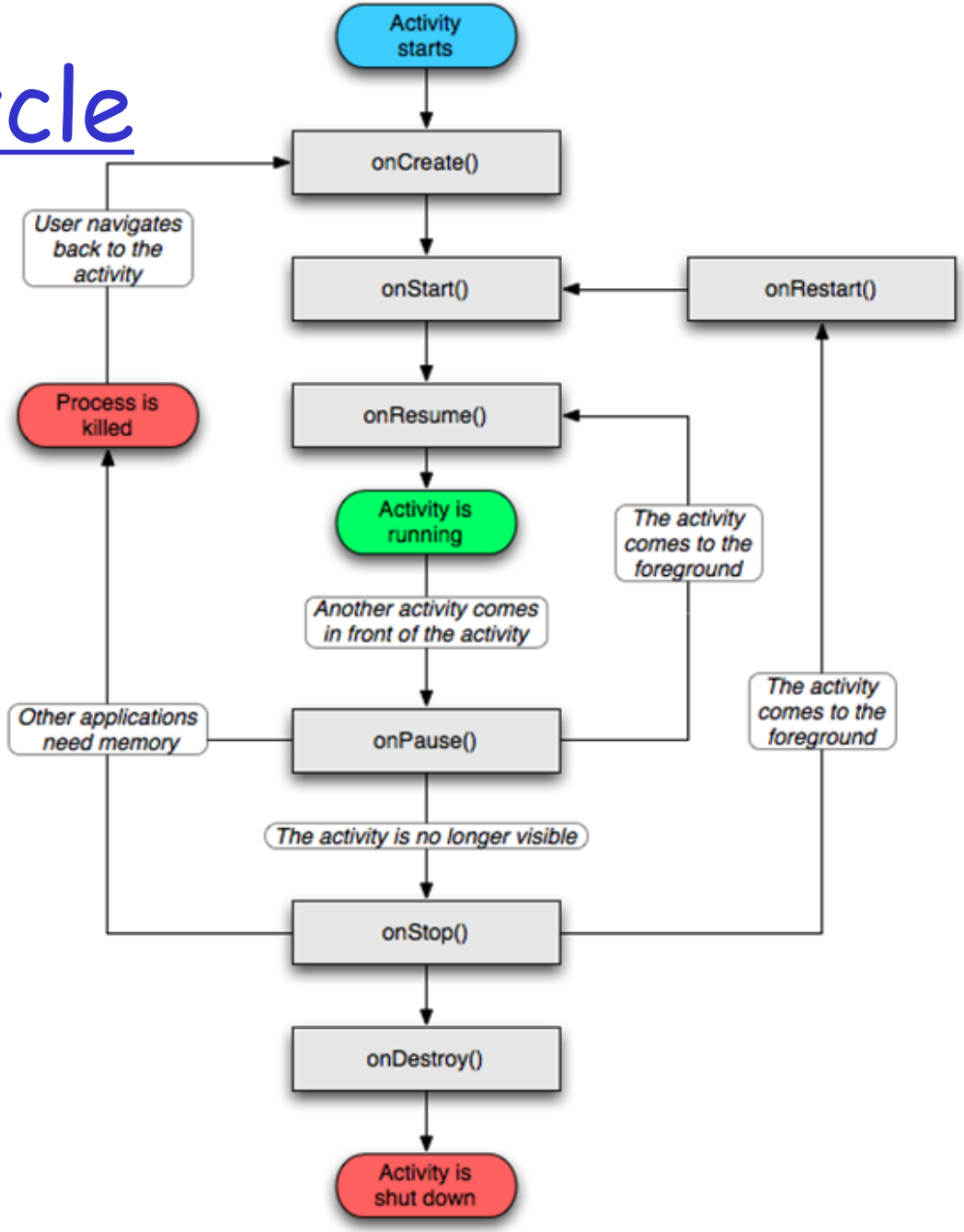
Activity Lifecycle

- Wechselt Zustand wird jeweils spezielle Methode der Activity aufgerufen
  - onCreate()
    - Aufruf, wenn die Activity ermalig erzeugt wird
  - onStart()
    - Aufruf, wenn die Activity für den Nutzer sichtbar wird
  - onRestart()
    - Aufruf, wenn die Activity erneut gestartet wird (Zustand stopped => started)



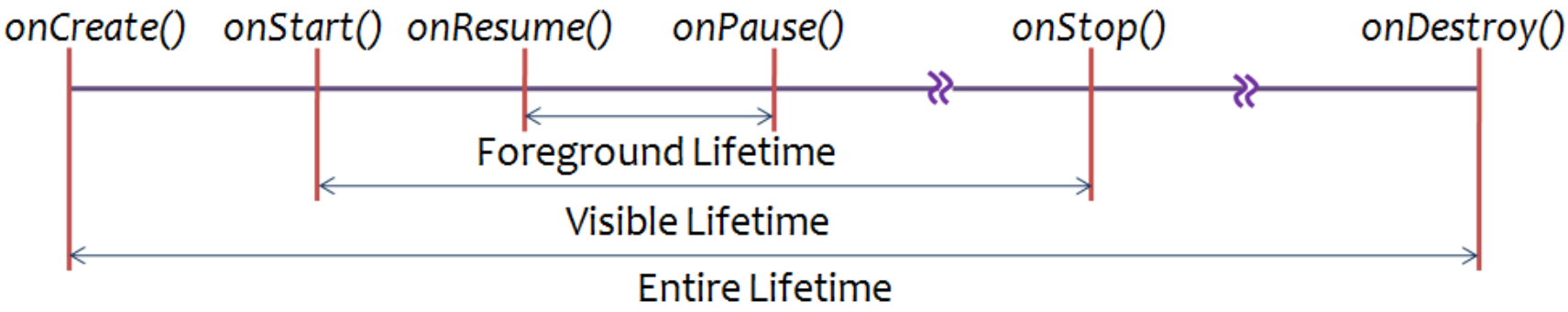
Activity Lifecycle

- onResume()
  - Aufruf, wenn Activity den Eingabefokus erhält
- onPause()
  - Aufruf, wenn eine andere Activity in den Vordergrund kommt
  - Typischerweise werden Änderungen gespeichert und CPU-intensive Operationen (Animationen etc.)
- onStop()
  - Aufruf, wenn Activity nicht mehr sichtbar
- onDestroy()
  - Aufruf, wenn Anwendung vom System beendet wird



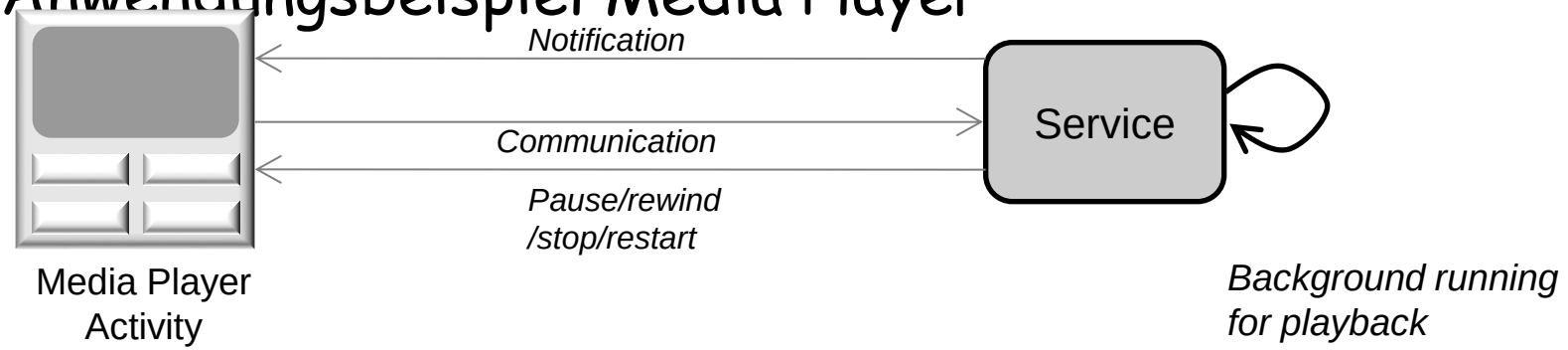
Activity Lifecycle

- Es ergeben sich 3 Phasen
  - Foreground Lifetime
    - Activity ist sichtbar und hat Eingabefokus
  - Visible Lifetime
    - Activity sichtbar, aber evt. kein Eingabefokuson
  - Entire Lifetime
    - Zeit zwischen dem Erzeugen und Beenden einer Activity durch das System



Services

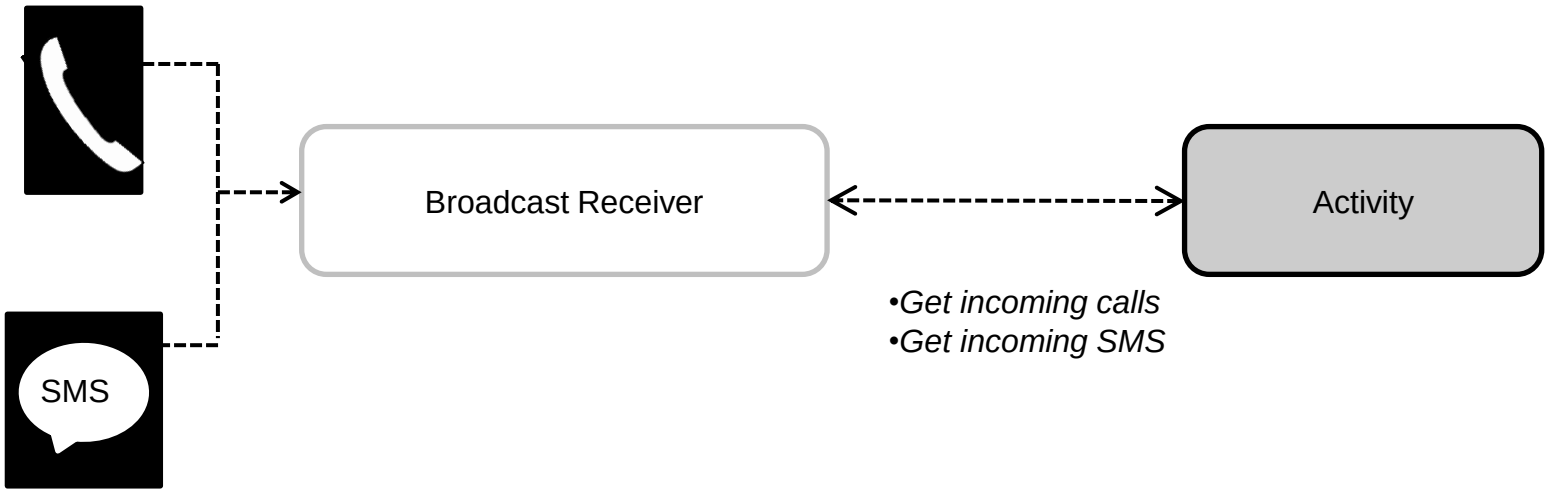
- Services sind Hintergrundprozesse ohne GUI-Elemente, z.B.
  - Download von Dateien über das Netzwerk
  - Abspielen von Musik
- Anwendungsbeispiel Media Player



- Services erben von der Basisklasse Service

Broadcast Receiver

- Broadcast Receiver reagieren auf systemweite Ereignisse
- Systemweise Ereignisse, z.B.
  - Neue Zeitzone, Eingehender Anruf, Bild wurde mit eingebauter Kamera aufgenommen
- Anwendungen können ebenfalls Ereignisse generieren, z.B.
  - Download abgeschlossen, Neue Nachricht
- Broadcast Receiver erben von Basisklasse BroadcastReceiver

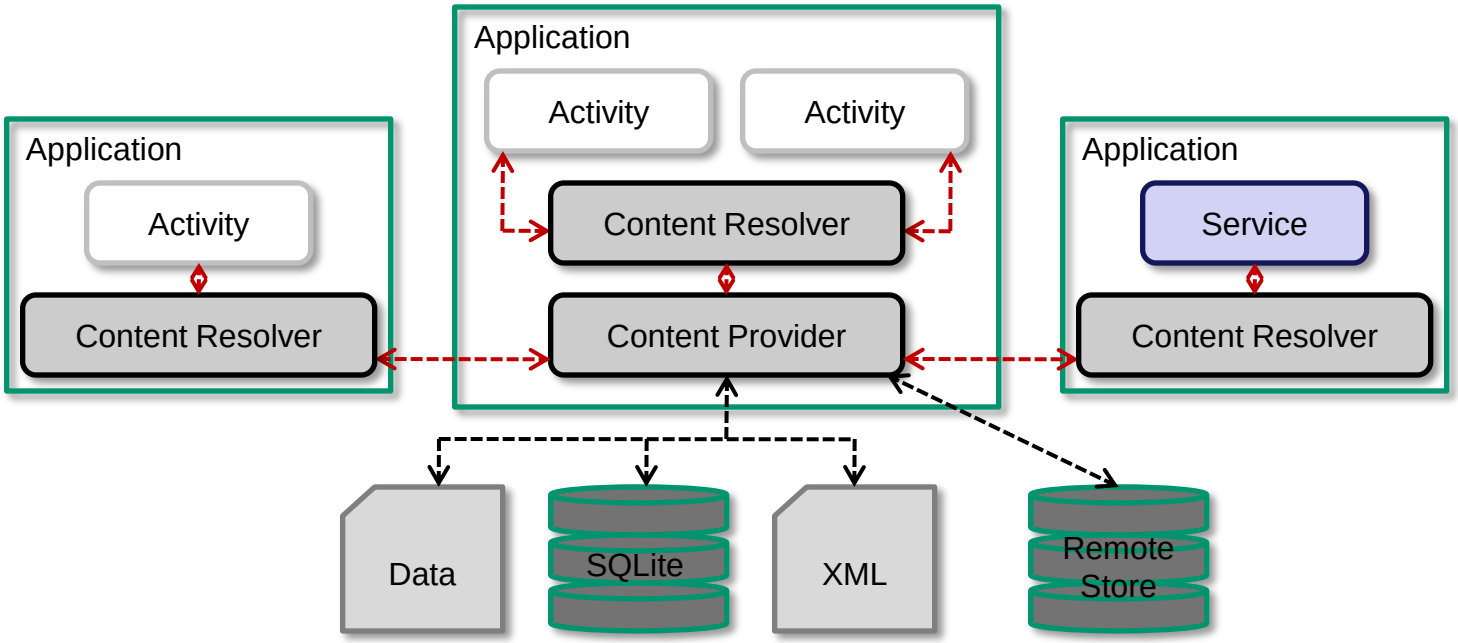


Content Provider (1)

- Anwendungen kann Daten in beliebiger Form speichern, z.B.
  - Datenbank
  - Datei
- Content Provider stellen Anwendungsdaten anderen Anwendungen zur Verfügung
  - Zugriff erfolgt über einheitliche Schnittstelle unabhängig von der Art der Speicherung
- Content Provider sind einzige Möglichkeit für Anwendungen Daten gemeinsam zu nutzen
- Content Provider erben von der ContentProvider Basisklasse

Content Provider (2)

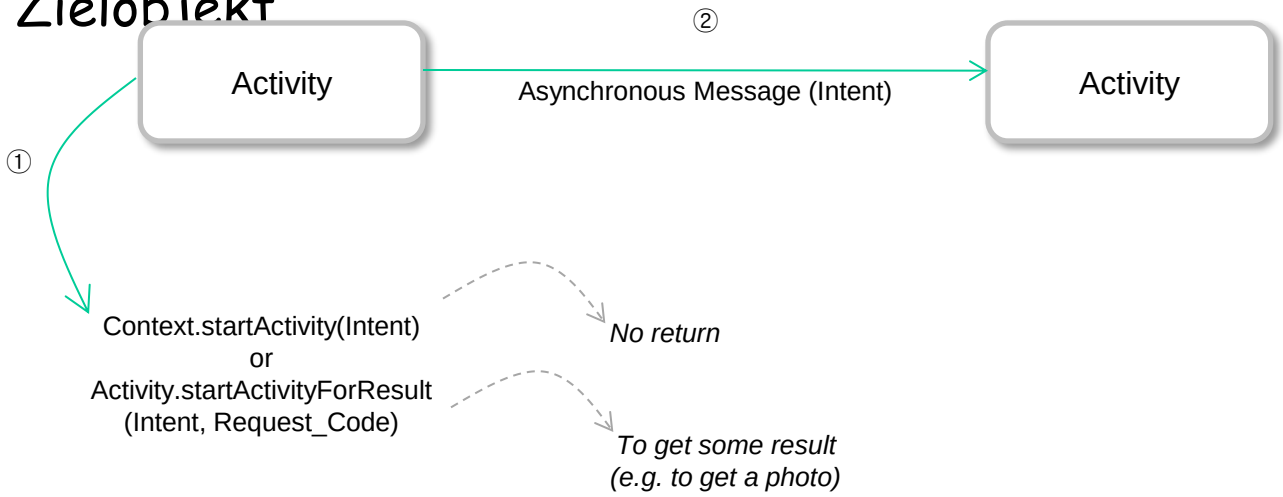
- Kein direkter Zugriff auf Content Provider
  - Indirekter Zugriff via Content Resolver



Datenzugriff mittels Content Provider

Intents (1)

- Intents sind Nachrichten-Objekte zur Kommunikation zwischen Komponenten. Bestandteile:
  - Aktion
    - (MAIN/VIEW/EDIT/PICK/DELETE/DIAL/etc)
  - Zielobjekt



Intents (2)

```
startActivity(new Intent(Intent.VIEW_ACTION, Uri.parse("http://www.fhnw.ch"));

startActivity(new Intent(Intent.VIEW_ACTION, Uri.parse("geo:47.480843,8.211293"));

startActivity(new Intent(Intent.EDIT_ACTION,Uri.parse("content://contacts/people/1"));
```

Manifest Beschreibungsdatei (1)

- Jede Anwendung enthält eine Beschreibungsdatei
  - XML Format
- Beschreibungsdatei legt fest
  - Java Paketname der Anwendung
  - Beschreibung aller Activities, Services, Broadcast Receiver und Content Provider
    - Deklaration der jeweiligen Implementierungsklassen
  - Sicherheitseinstellungen / Zugriffsrechte
    - Abfrage / Ändern von Systemeinstellungen
    - Zugriff auf Netzwerk / Kamera etc.
  - Erforderliche Version der Android API
  - Erforderliche Bibliotheken

Android Beispielprogramm (1)

- Hello Access Point … Quellcode

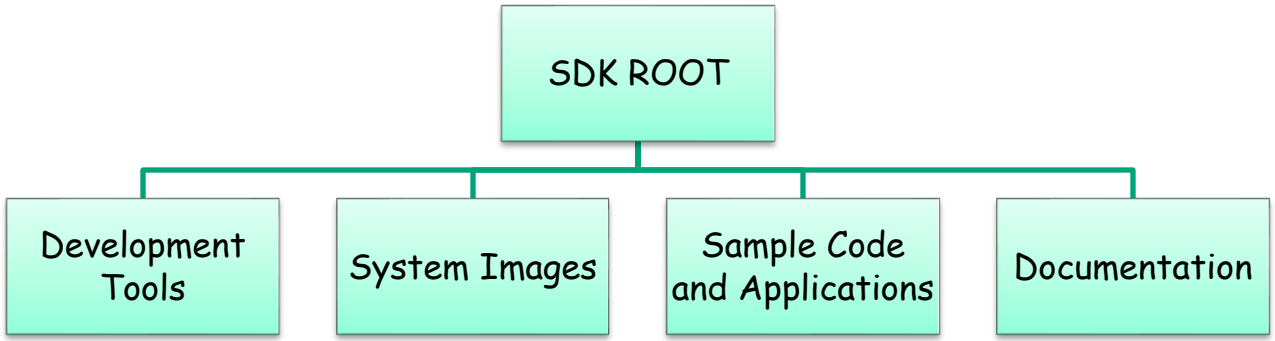
```
package rvs.WiFiData;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.widget.TextView;
import android.net.wifi.*;

public class WiFiData extends Activity {

  public void onCreate(Bundle savedInstanceState) {

     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
     TextView tv = new TextView(this);
     WifiManager wifiManager =
                 (WifiManager) getSystemService(Context.WIFI_SERVICE);
     WifiInfo wifiInfo = wifiManager.getConnectionInfo();
     tv.setText("Access Point MAC:" + wifiInfo.getMacAddress());
     setContentView(tv);
      }
}
```

Manifest Beschreibungsdatei

Hello Access Point … AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="rvs.WiFiData"  android:versionCode="1„  android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name"
    android:debuggable="true">
    <activity android:name=".WiFiData"  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-sdk android:minSdkVersion="3" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-
    permission>
</manifest>
```

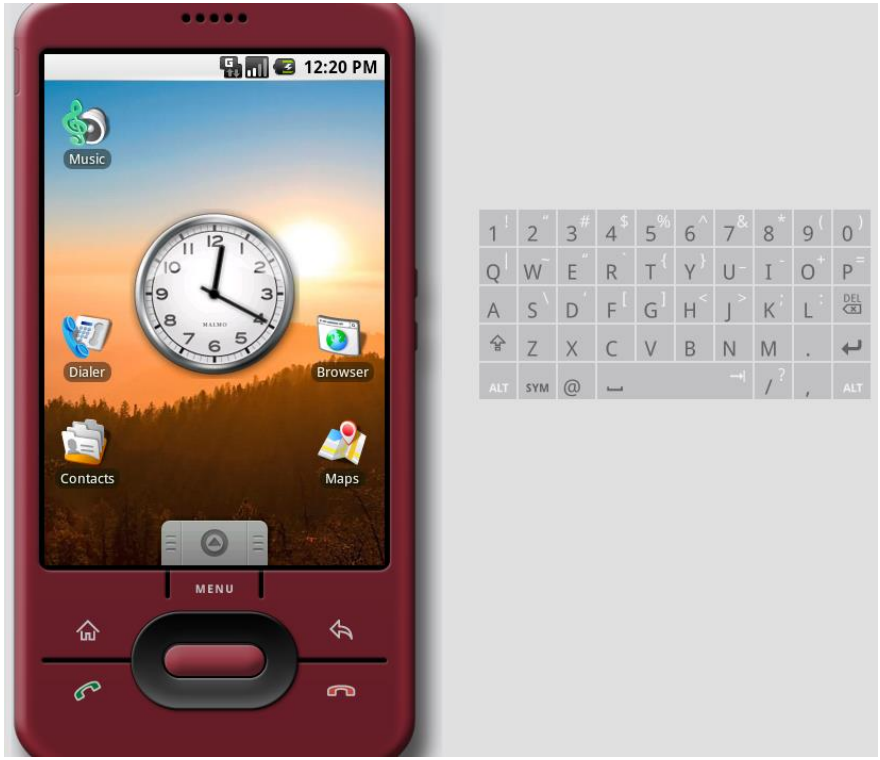
Aufbau - Android SDK



Verzeichnis	Beschreibung
Development Tools	Kompilierungs-, Debugging- und GUI-Tools
System Images	Bootfähige Images
Sample Code and Applications	Beispielcode – und Anwendungen
Documentation	Vollständige API Dokumentation

Android SDK (1)

- Emulator
  - Virtuelle Maschine mit Android
  - Anwendungen können direkt auf Entwickler-PC getestet und debuggt werden
  - Netzwerk wird simuliert
  - Eingehende Anrufe / SMS werden simuliert



Android SDK (2)

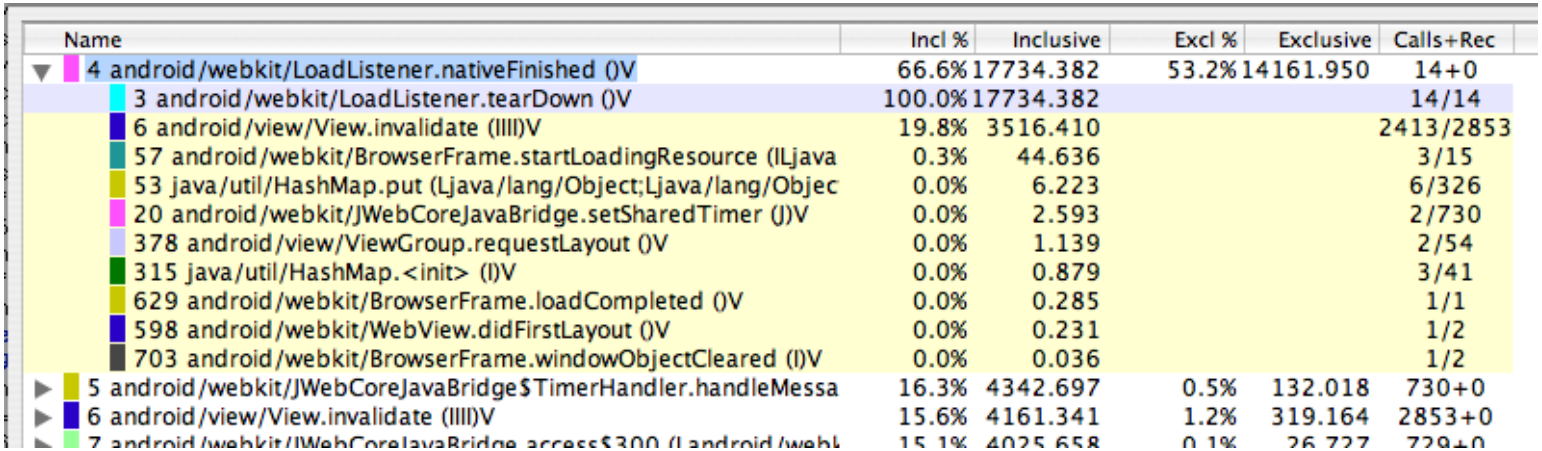
- Hierarchy Viewer
  - Debugging und Optimierung von GUIs
  - Layout Hierarchie wird dargestellt
- Android Debug Bridge (adb)
  - Debugging von Anwendungen
  - Arbeitet mit Emulator oder realen mobilen Endgerät
  - Testweise Installation von Anwendungen auf mobilen Endgerät
- Android Asset Packaging Tool (aapt)
  - Erstellen von Android Package zur Software Distribution

Android SDK (3)

- Dalvik Debug Monitor Service (ddms)
  - Verwaltung von Prozessen auf Emulator oder realen Endgerät
    - Beenden von Prozessen
    - Auswahl von Prozess zum Debugging
    - Generierung von Trace-Daten
    - Anzeige von Heap und Thread Information
    - Aufnahme von Screenshots
- Android Interface Description Language (aidl)
  - Erzeugen von Marshalling-Code für Interprozess-Kommunikation
- sqlite3
  - Auslesen von sqlite3 Datenbanken

Android SDK (4)

- Traceview
  - Profiling Tools zur Analyse der Trace-Daten



Name	Incl %	Inclusive	Excl %	Exclusive	Calls+Rec
▼ 4 android/webkit/LoadListener.nativeFinished ()V	66.6%	17734.382	53.2%	14161.950	14+0
3 android/webkit/LoadListener.tearDown ()V	100.0%	17734.382			14/14
6 android/view/View.invalidate (IIII)V	19.8%	3516.410			2413/2853
57 android/webkit/BrowserFrame.startLoadingResource (ILjava	0.3%	44.636			3/15
53 java/util/HashMap.put (Ljava/lang/Object;Ljava/lang/Objec	0.0%	6.223			6/326
20 android/webkit/JWebCoreJavaBridge.setSharedTimer (J)V	0.0%	2.593			2/730
378 android/view/ViewGroup.requestLayout ()V	0.0%	1.139			2/54
315 java/util/HashMap.<init> (I)V	0.0%	0.879			3/41
629 android/webkit/BrowserFrame.loadCompleted ()V	0.0%	0.285			1/1
598 android/webkit/WebView.didFirstLayout ()V	0.0%	0.231			1/2
703 android/webkit/BrowserFrame.windowObjectCleared (I)V	0.0%	0.036			1/2
▶ 5 android/webkit/JWebCoreJavaBridge$TimerHandler.handleMessa	16.3%	4342.697	0.5%	132.018	730+0
▶ 6 android/view/View.invalidate (IIII)V	15.6%	4161.341	1.2%	319.164	2853+0
▶ 7 android/webkit/JWebCoreJavaBridge.access$300 (Landroid/webk	15.1%	4025.658	0.1%	26.727	729+0

- Dmtracedump
  - Generierung von Aufrufgraphen aus Trace-Daten

Android SDK (5)

- dx
  - Umschreiben von Java Bytecode (*.class) in Android Bytecode (*.dex)

- UI/Application Exerciser Monkey
  - Erzeugt zufällige Input-Sequenzen auf Emulator oder realen Endgerät sowie Systemereignisse (Anrufe etc.)
    - Stress-Test von Anwendungen

Android Kommunikationstechnologie Google Nearby

Google Nearby (1)

Sammlung von APIs für „Proximity & cross device communication"

- **Nearby Connections API**
  - Datenaustausch/Kommunikation mit Geräten im näheren Umfeld
  - Keine Internetverbindung notwendig
  - Android-exklusiv
- **Nearby Messages API**
  - Austausch kleiner Datenmengen mit Geräten im näheren Umfeld
  - Internetverbindung notwendig
  - Android- und iOS cross-platform
- **Fast Pair:** One-Tap Bluetooth Pairing

- https://developers.google.com/nearby/

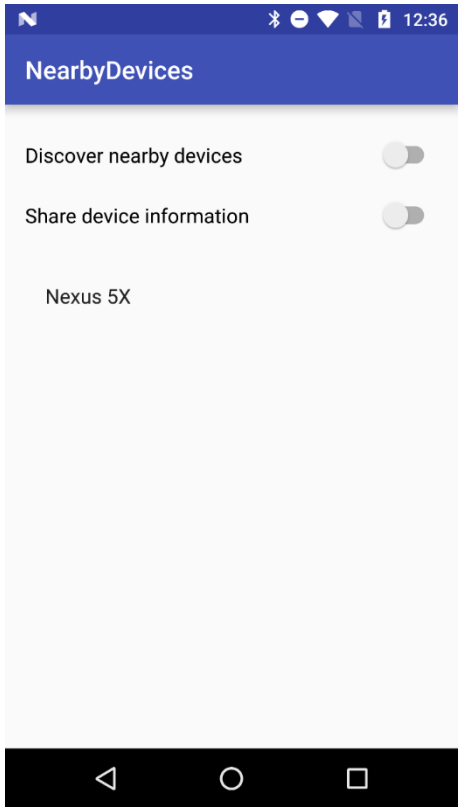
Google Nearby (2)

- Google Nearby nutzt (mehrere) bestehende Technologien
  - Bluetooth /BLE
  - Wi-Fi
  - Ultraschall
  - (Internet/IP)
- Bietet „Service" (d.h Datenübertragung/Kommunikation) für Apps an
- Kombination der Vorzüge verschiedener Technologien
  - Niedriger Verbrauch von Bluetooth (LE) Scans
  - Hohe Datenrate von Wi-Fi
  - Lokale Beschränkung durch Ultraschall

Google Nearby Messages (1)

- Erlaubt Austausch kurzer Nachrichten

  zwischen Geräten in vers. Netzwerken
  - Verfügbar für Android und iOS
  - Benötigt bestehende Internetverbindung
  - Nutzt Bluetooth (Low Energy),Wi-Fi und Töne nahe des Ultraschallbereichs für Pairing-Code



- Zwei Rollen
  - Publisher
  - Subscriber

Google Nearby Messages (2)

1. Publisher-App möchte „Payload" mit Token assoziieren
   - Server erzeugt temporäre Verbindung Payload <-> Token
2. Publisher-Gerät sendet Token über o.g. Kanäle
   - Scannt auch auf selbigen Kanälen
3. Subscriber-App assoziiert ihr „Abo" mit einem Token und sendet dieses über obige Kanäle
   - Scannt ebenfalls
4. Wenn Token mitgehört: Server benachrichtigen
5. Server ermöglicht Nachrichtenaustausch zweier Geräte, wenn beide denselben Token und denselben App-Bezug aufweisen

Google Nearby Connections

- API für P2P-Networking in Android
  - Ab Android 6.0
  - kein iOS-Support
- Ermöglicht Discovery und Datenaustausch ohne bestehende Internetverbindung
  - Nutzt Bluetooth (Low Energy) und Wi-Fi
  - Management dieser Schnittstellen (AP öffnen; nach Dateitransfer wieder mit vorherigem Netzwerk verbinden, …) wird vor Nutzer weitestgehend verborgen
- WalkieTalkie-Demo-App verfügbar

https://github.com/googlesamples/android-nearby/tree/master/connections/walkietalkie

Persönliche Erfahrungen

- Verbindungsaufbau dauert z.T. lange
- Geräte finden sich lange nicht, wenn Nearby-Suche zur gleichen Zeit gestartet wird
  - ggf. Überlappung der Discovery-Intervalle
- Existierende Verbindungen brechen ab
- Internetverbindung (via Wi-Fi) bricht ab
  - Geräte können Wi-Fi kurzzeitig für Nearby-Datenübertragung reservieren

Google Fast Share

Einführung

Anwendungsszenario: schneller und unkomplizierter Austausch von Dateien zwischen zwei Geräten (Smartphone, Notebook, …) über kurze Distanzen (wenige Meter)

- Apple bietet AirDrop
- Android bietet bisher nur langsame (Beam) oder wenig verbreitete (S-Beam) Lösungen
- Fast Share (ab Android 10)
  - Soll schnelles, universelles Protokoll für Android-Geräte bieten
  - Baut auf existierenden Technologien auf

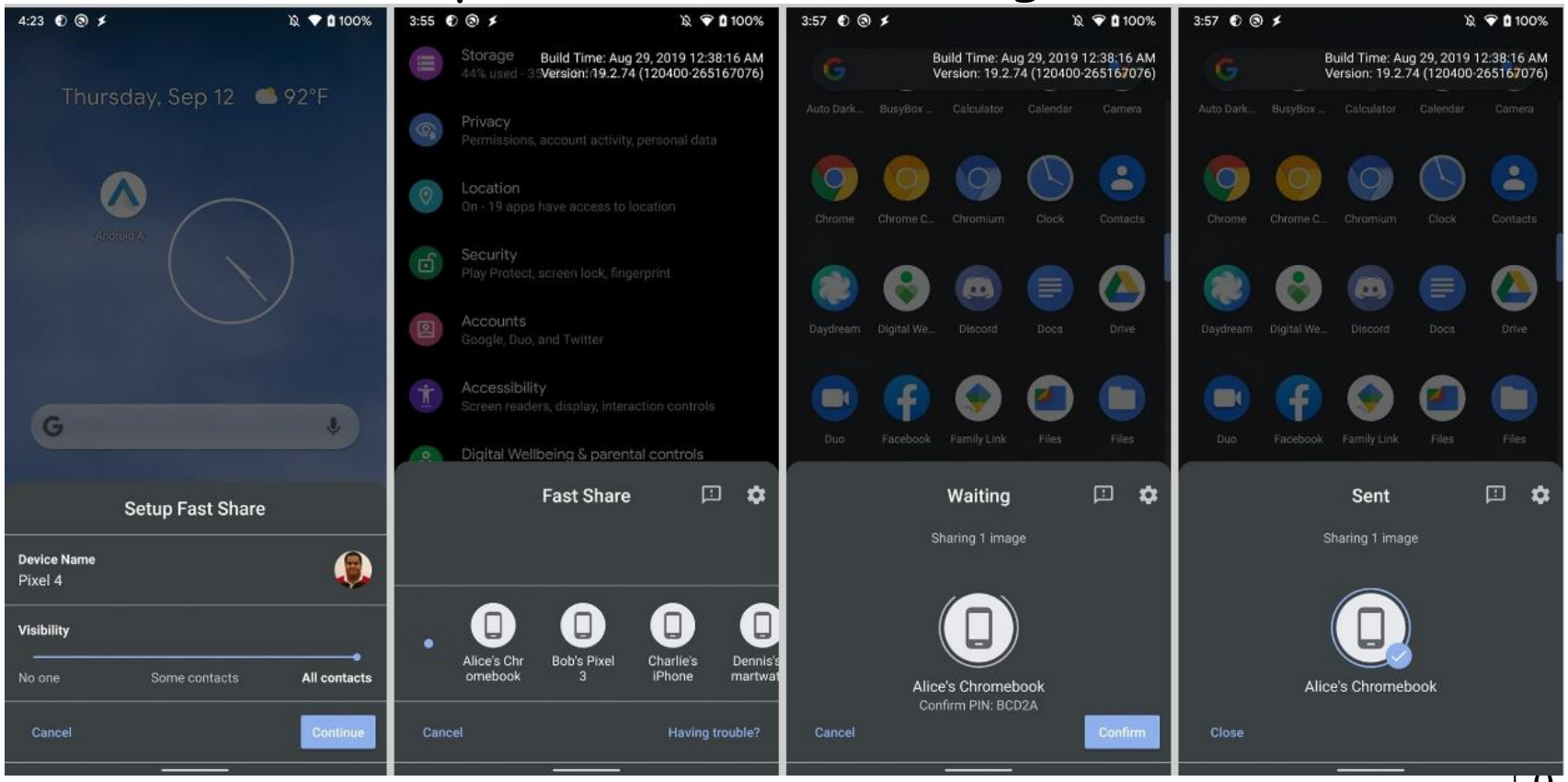
Technische Grundlagen

- Vorgänger: **Android Beam** (Android 4.0 – pre-Android 10)
  - Erlaubt Austausch kleiner Datenmengen via **NFC**
  - Seit Android 4.1: Initiierung einer **Bluetooth**-Verbindung mit Hilfe von NFC zum Transport größerer Datenmengen (Fotos, Videos) möglich
  - Erweiterung **„S Beam"** (Samsung): nutzt **Wi-Fi Direct** statt Bluetooth für höhere Datenraten

- **Fast Share** (ab Android 10)
  - Handshake via **Bluetooth** und Datentransfer via **Wi-Fi Direct**
  - Nutzt Google Nearby
  - (weiterhin keine Internetverbindung erforderlich)

Senden einer Datei

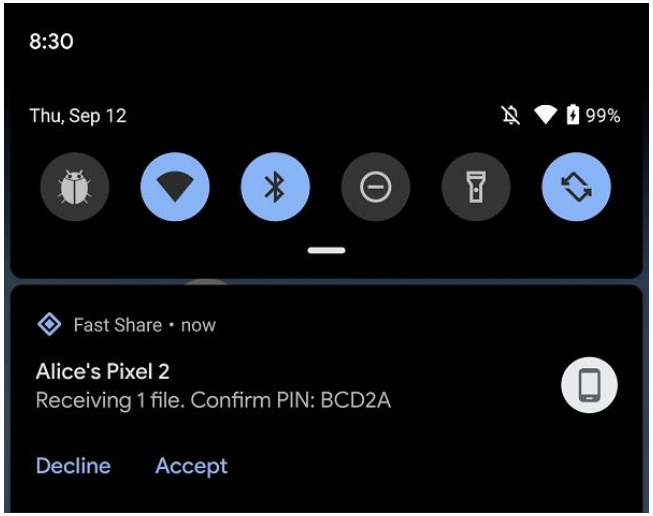
- Fast Share im „Teilen"-Systemdialog wählen > ggf. initiales Setup > Auswahl des Zielgeräts > PIN-



Bildquelle: xda-developers.com [2]

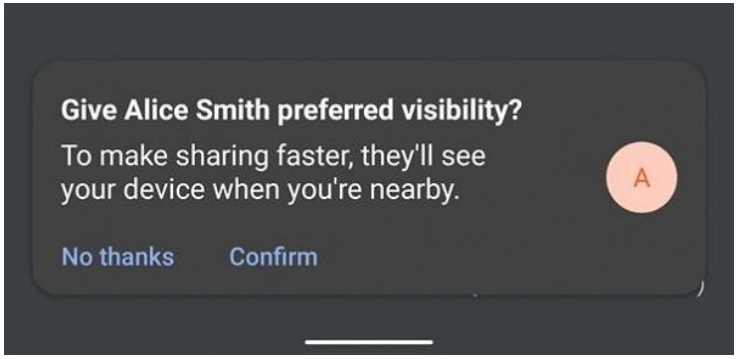
Empfang von Daten

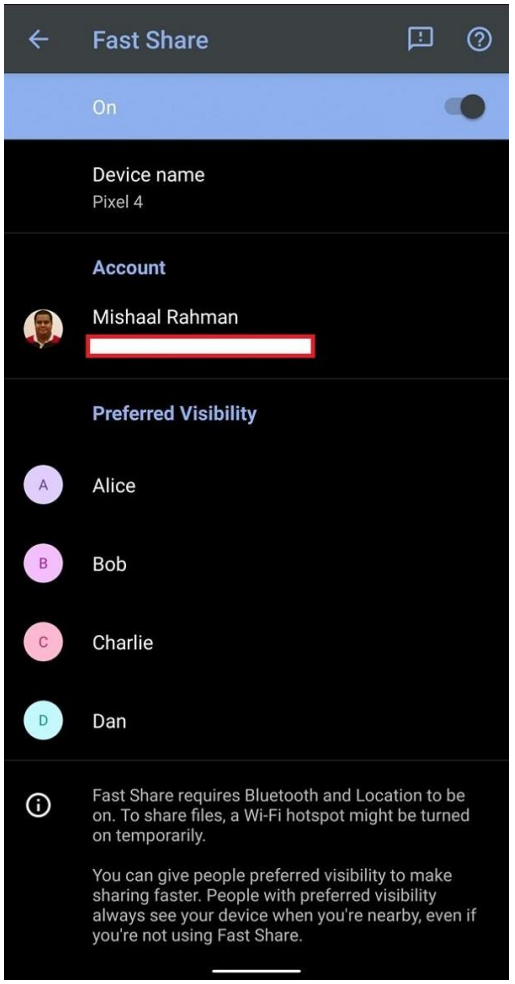
- Empfang möglich, solange Gerät im Zustand „sichtbar" für Fast Share ist
- Angebotene Dateien tauchen als Benachrichtigung auf
  - PIN wird für Verifikation zum Abgleich mit Sender eingeblendet
  - Übertragung kann sofort angenommen oder abgelehnt werden



Favorisierte Geräte

- Tauschpartner können favorisiert werden
  - Erlaubt schnelleren Zugriff
  - Bessere Übersicht in größeren Menschenansammlungen





Vor- und Nachteile

+ Hohe Verbreitung möglich, da existierende Standards verwendet werden
+ Cross-Plattform theoretisch möglich

- Wi-Fi-Übertragung mittels Google Nearby kann u.U. bestehende Verbindungen unterbrechen
- Gegenwärtige Android-Implementierung erfordert Google Nearby
  - Erfordert Play Store (viele Rechte, Datenschutz?)
  - Cross-Plattform-Kompatibilität noch unklar
- Fast Share ist Teil der Google Play Services (d.h. Closed Source; Android Beam war Teil des AOSP)

Zusammenfassung

- Einführung Google Android
  - Vision
  - Konsortium
  - Android Developer Challenge
- Entwicklung mit Google Android
  - Architektur
  - Aufbau von Anwendungen
  - Android Software Development Kit
  - Google Nearby als Kommunikationstechnologie
- Google Fast Share
  - Ermöglicht unkomplizierten, lokalen Datenaustausch (mit anderen Android-Geräten)
  - Nutzung von Bluetooth, Wi-Fi, Standortdaten
  - Ersetzt Android Beam