

Открытый урок курса «Android Developer»

# Основные компоненты приложения Android



• REC

**Напишите «+» в чат, если меня  
слышно и видно**





# Николай Кочетков

Руководитель Android разработки Vista

- В профессиональном IT с 2000г
- С 2016г в коммерческой разработки Android
- Работаю “Играющим тренером”

↗ @motorro

✉ motorro@gmail.com

# Правила вебинара



Активно участвуем



Задаем вопросы в чат



Вопросы вижу в чате,  
могу ответить не сразу



Запись вебинара придёт на почту

# Маршрут вебинара

Приложение и Application Sandbox

Обмен данными между приложениями

Особенности приложений ОС Android

Про курс

Основные компоненты приложения



# Давайте познакомимся

Есть ли у вас опыт работы в IT?



С какой основной целью пришли сегодня?

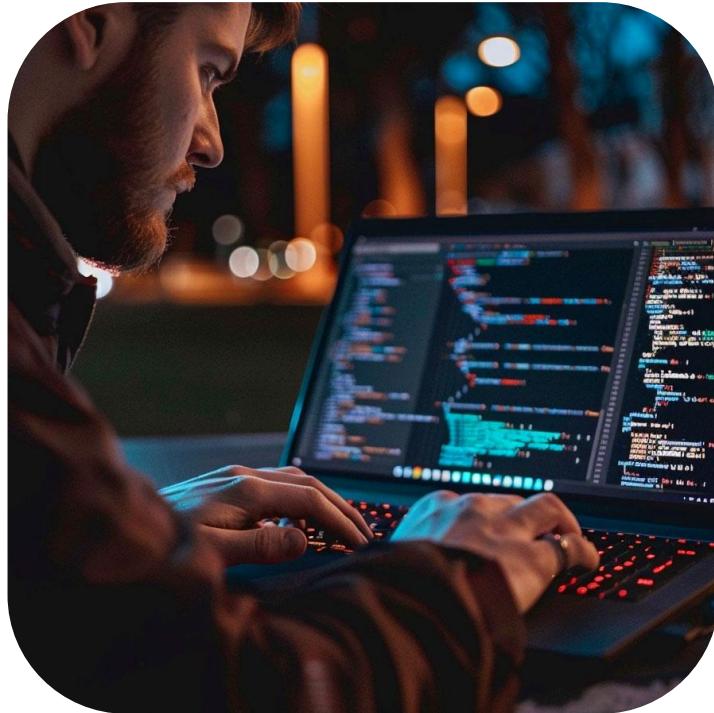
# 06 OTUS



# OTUS

Создаём авторские онлайн-курсы  
для IT-специалистов разных уровней:  
от **junior** до **lead**

Разрабатываем программы  
на основе запросов IT-компаний  
к кандидатам



# У OTUS есть образовательная лицензия

По окончании программ выдаём  
удостоверения о повышении  
квалификации и дипломы  
о профессиональной переподготовке



# Направления курсов

>130  
курсов

Программирование

Архитектура

Инфраструктура

Безопасность

Data Science

GameDev

Управление

Аналитика и анализ

Тестирование



# Обучение в OTUS



Обучение проходит в **живом формате** вебинаров



**Нетворкинг:** всегда можно задать вопрос опытным коллегам



Все записи занятий и материалы сохраняются в личном кабинете **навсегда**



**Проект**, который вы выполните, поможет расширить портфолио



По каждому домашнему заданию преподаватель даёт **развернутый фидбек**



Программа обучения на курсах обновляется **каждый запуск**



# Основные компоненты приложения Android



# Цели вебинара

К концу занятия вы сможете

# Смысл

Для чего вам это уметь

Понимать, как устроена ОС Android

Обучение станет понятнее

Узнать про обмен данными приложений

Поймете принципы работы приложений

Узнаете об основных компонентах

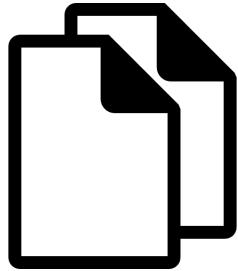
Узнаете, как строится приложение



# Приложение и Application Sandbox



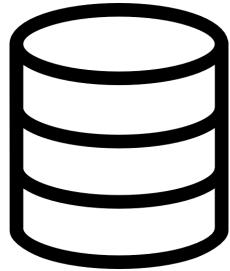
# Данные и ресурсы приложения



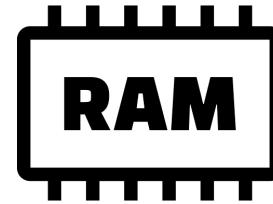
Файлы приложения



Preferences



Базы данных



Оперативные данные

# **Что мы хотим от системы?**

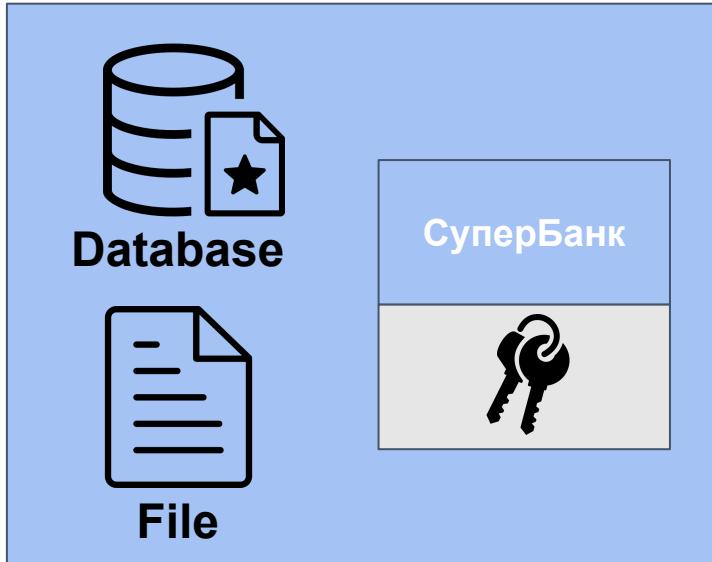
**Как пользователь...**



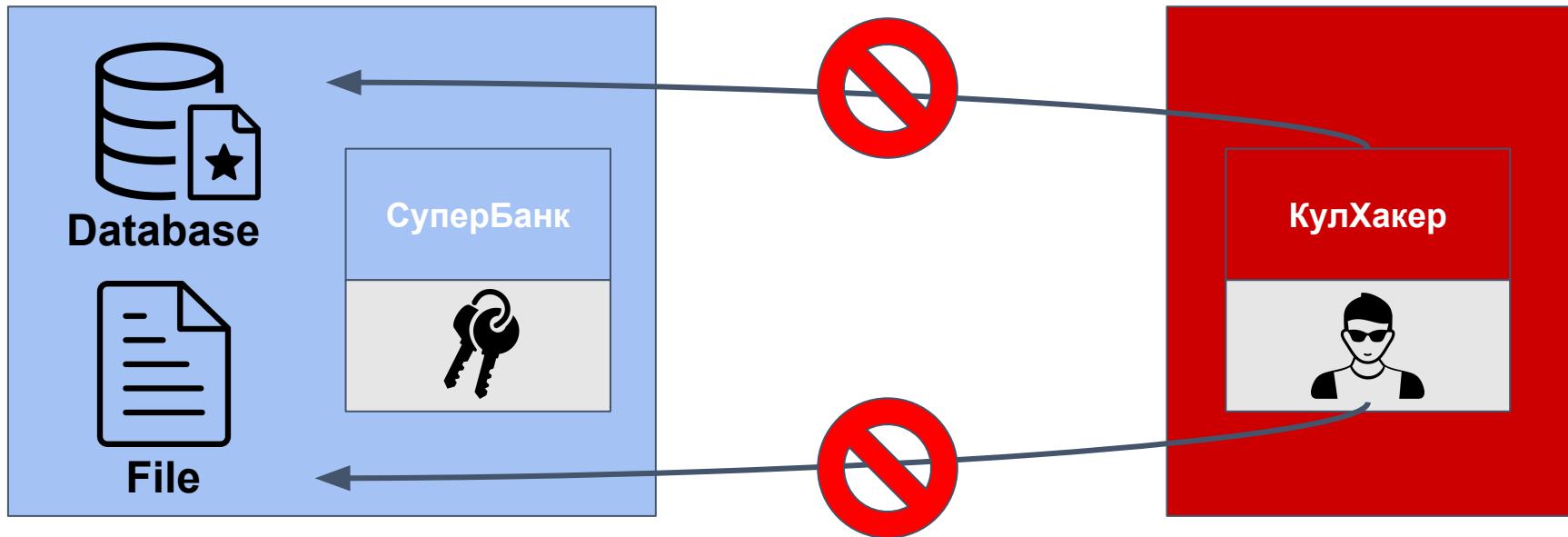
# Запрет доступа по умолчанию

- Как пользователь, я **НЕ ХОЧУ**, чтобы другие приложения имели неконтролируемый доступ к памяти моего приложения
- Как пользователь, я **НЕ ХОЧУ**, чтобы другие приложения имели неконтролируемый доступ к файлам моего приложения

# Запрет доступа



# Запрет доступа



# Доступ к данным приложения



# Добровольное предоставление доступа

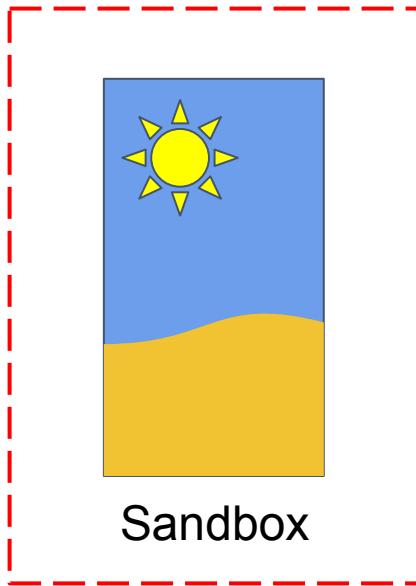
- Как пользователь, Я ХОЧУ, чтобы другие приложения могли воспользоваться моими оперативными данными
- Как пользователь, Я ХОЧУ, уметь делиться своими файлами



# Принцип

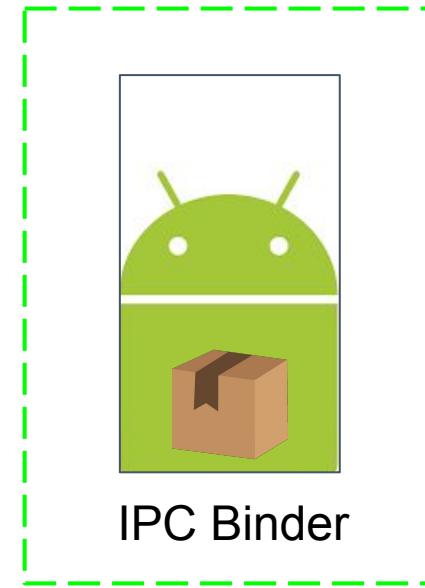
Deny All, Allow Some

# Добровольное предоставление доступа



Sandbox

Запрет  
несанкционированного  
доступа



IPC Binder

Контролируемый обмен  
данными



# Ваши вопросы



если нет вопросов,  
поставьте огонек



# Linux Application Sandbox



# Application Sandbox

*Application sandbox is a security mechanism through which individual android applications run in their own “space” and cannot interact with other installed apps or the Android OS without proper permissions.*

[Application Sandbox](#)



# Запрет доступа по умолчанию

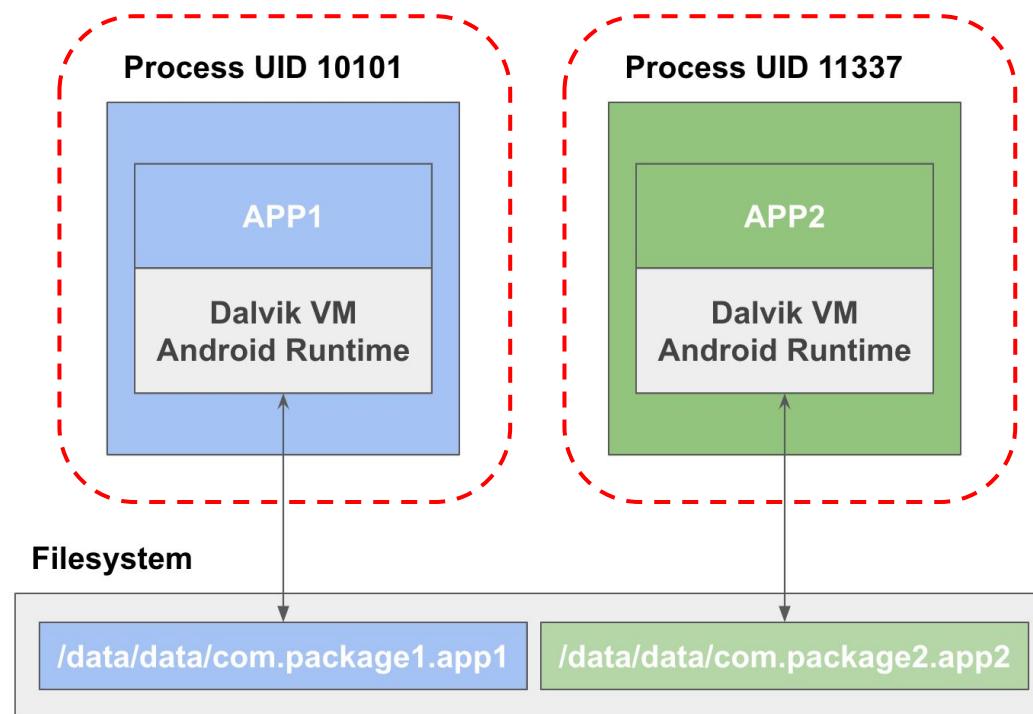
- Как пользователь, я **НЕ ХОЧУ**, чтобы другие приложения имели неконтролируемый доступ к памяти моего приложения
- Как пользователь, я **НЕ ХОЧУ**, чтобы другие приложения имели неконтролируемый доступ к файлам моего приложения

# UID

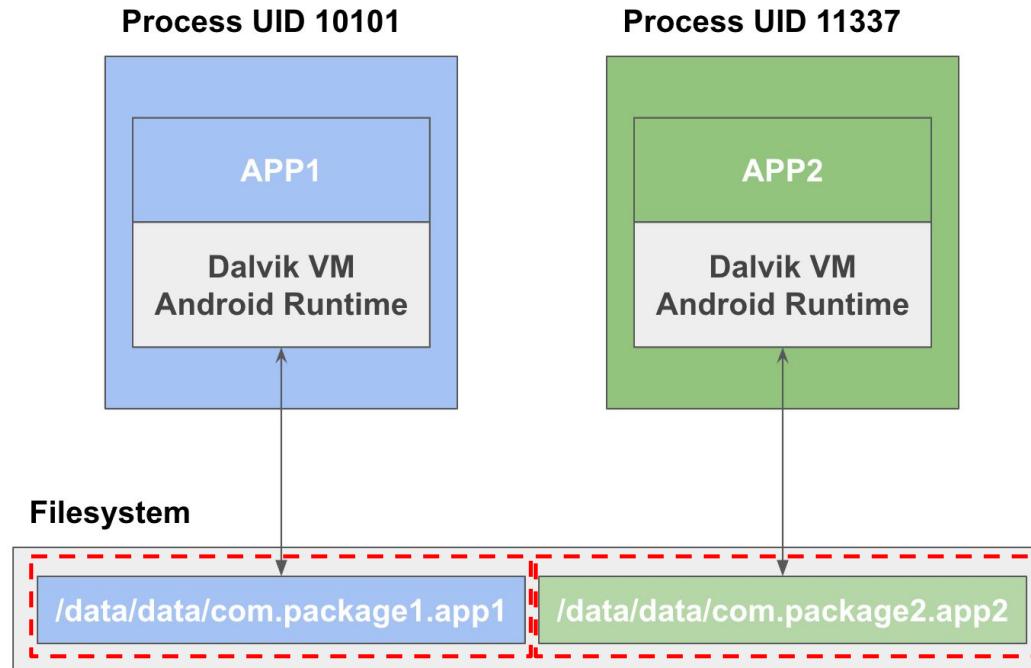
- Каждое приложение - Unique ID (UID)
- Этот UID постоянен, пока приложение установлено
- Изоляция процессов на уровне Process UID
- Изоляция доступа к диску на уровне User UID



# Application Sandbox - память



# Application Sandbox - диск



# Права на доступ к файлам приложения

Только пользователь u0\_a13

```
$ ls -la /data/data  
drwx----- 4 u0_a13      u0_a13      4096 2021-02-01 13:37 com.package1.app1  
drwx----- 6 u0_a163      u0_a163      4096 2021-02-01 13:39 com.package2.app2
```

Только пользователь u0\_a163



# Запрет доступа по умолчанию

- Как пользователь, я **НЕ ХОЧУ**, чтобы другие приложения имели неконтролируемый доступ к памяти моего приложения
- Как пользователь, я **НЕ ХОЧУ**, чтобы другие приложения имели неконтролируемый доступ к файлам моего приложения

# Ваши вопросы



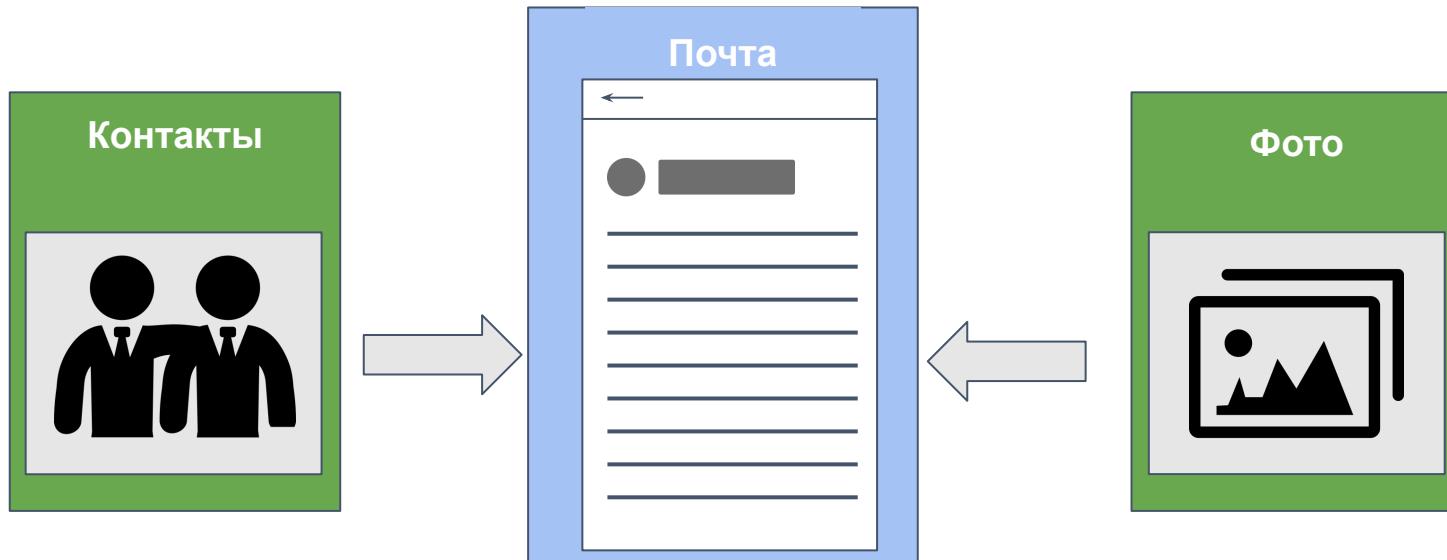
если нет вопросов,  
поставьте огонек



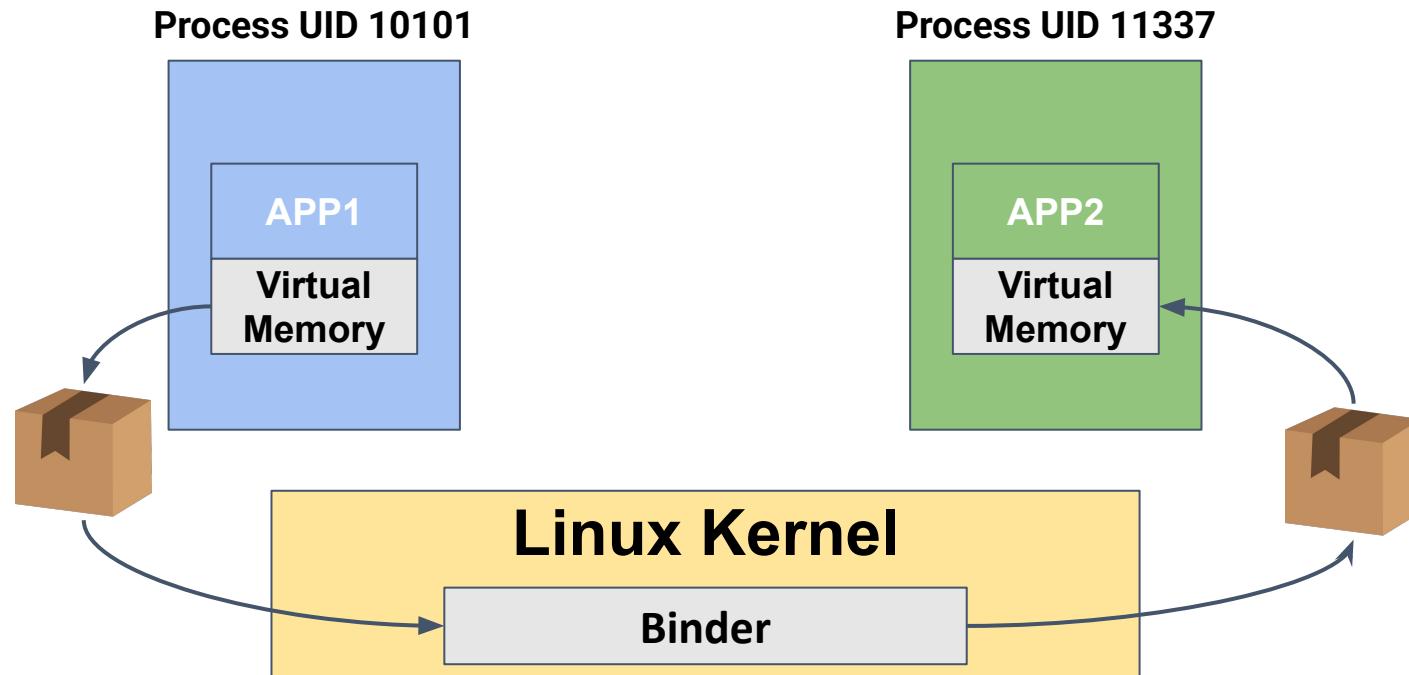
# Android Binder IPC



# Доступ к данным приложения



# Binder IPC



# Добровольное предоставление доступа

- Как пользователь, Я ХОЧУ, чтобы другие приложения могли воспользоваться моими оперативными данными
- Как пользователь, Я ХОЧУ, уметь делиться своими файлами



# **Binder - важнейший механизм Android!**

Большая часть коммуникаций внутри системы использует Binder

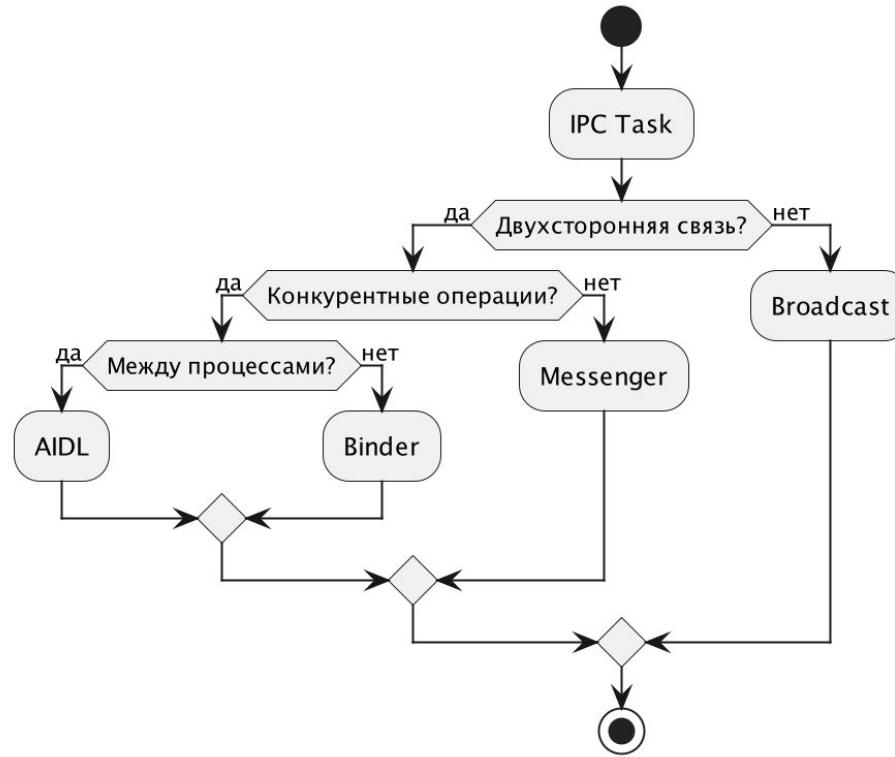


# Обмен поверх Binder IPC

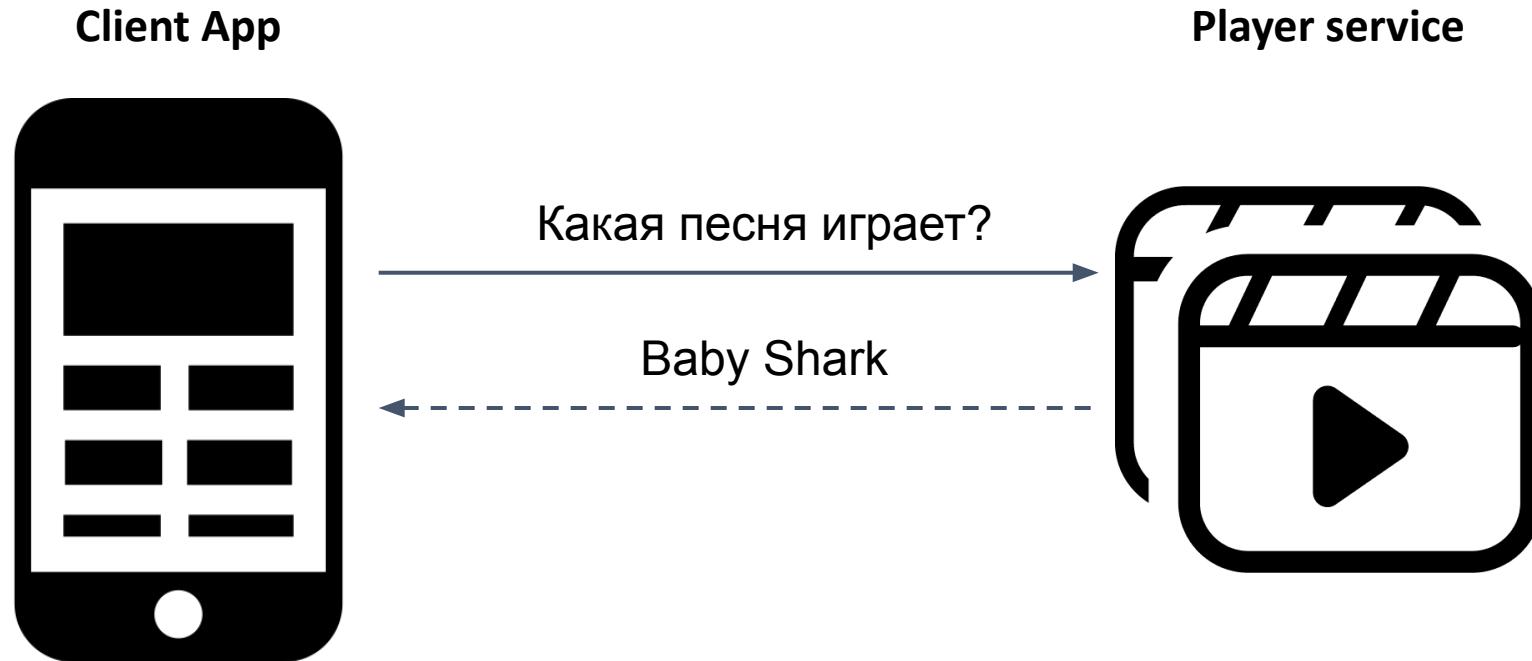
- AIDL
- Messenger
- Intent Broadcast



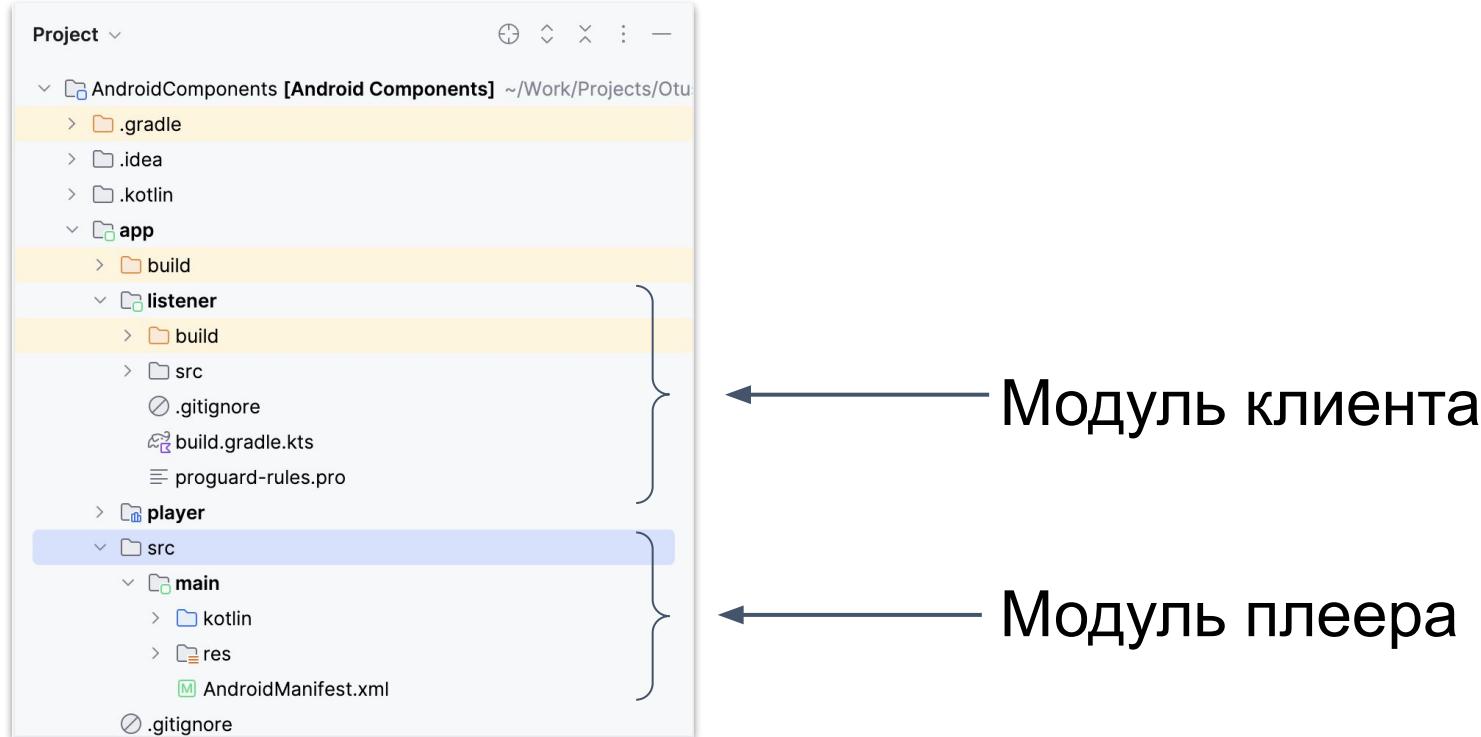
# Обмен поверх Binder IPC



# Обмен поверх Binder IPC



# Обмен поверх Binder IPC



# Описание интерфейса сервиса

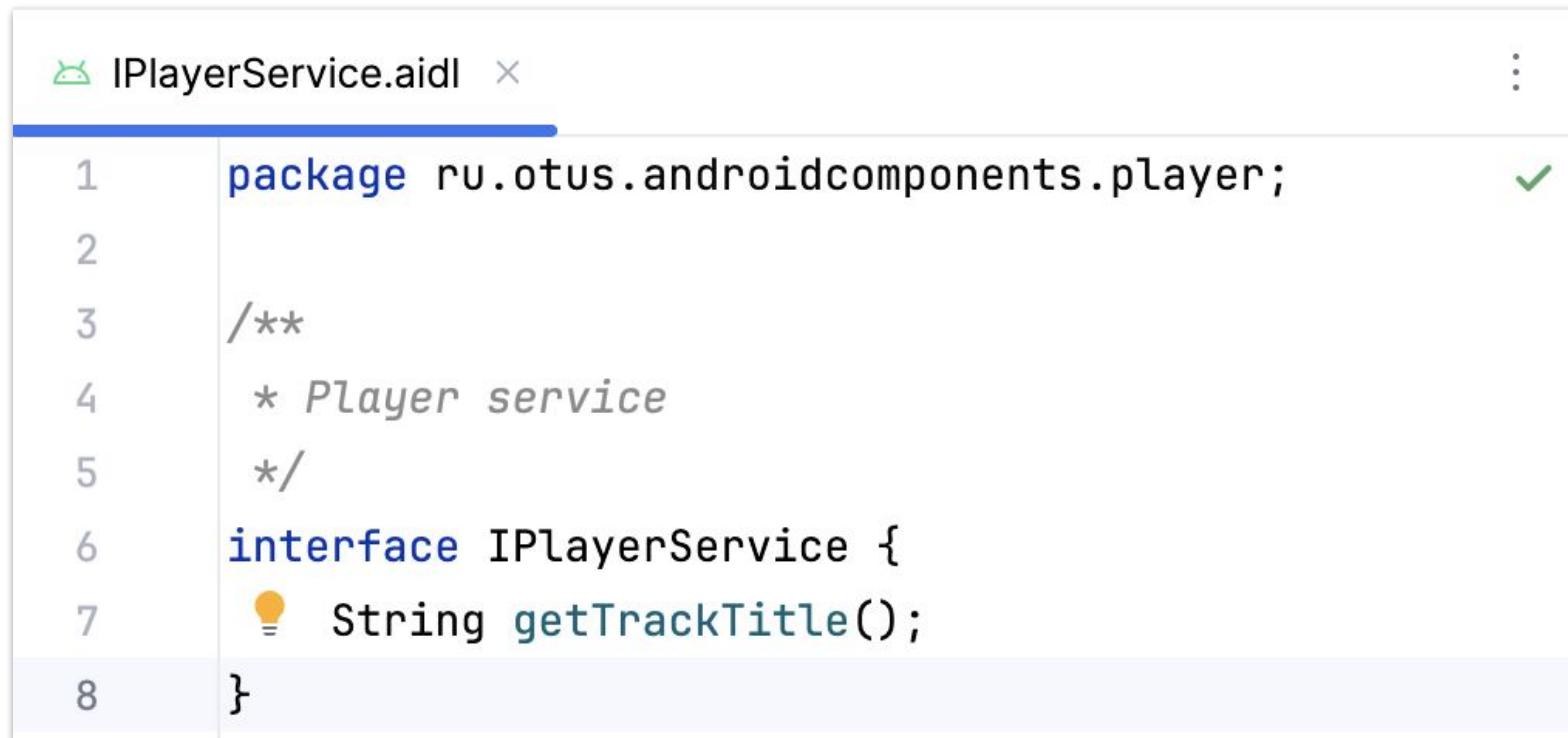
The screenshot shows a code editor window for an AIDL file named `IPlayerService.aidl`. The code defines a service interface with a single method `getTrackTitle()`.

```
1 package ru.otus.androidcomponents.player; ✓
2
3 /**
4  * Player service
5  */
6 interface IPlayerService {
7     String getTrackTitle();
8 }
```



# AIDL - удаленный вызов процедур

# AIDL - описание интерфейса сервиса

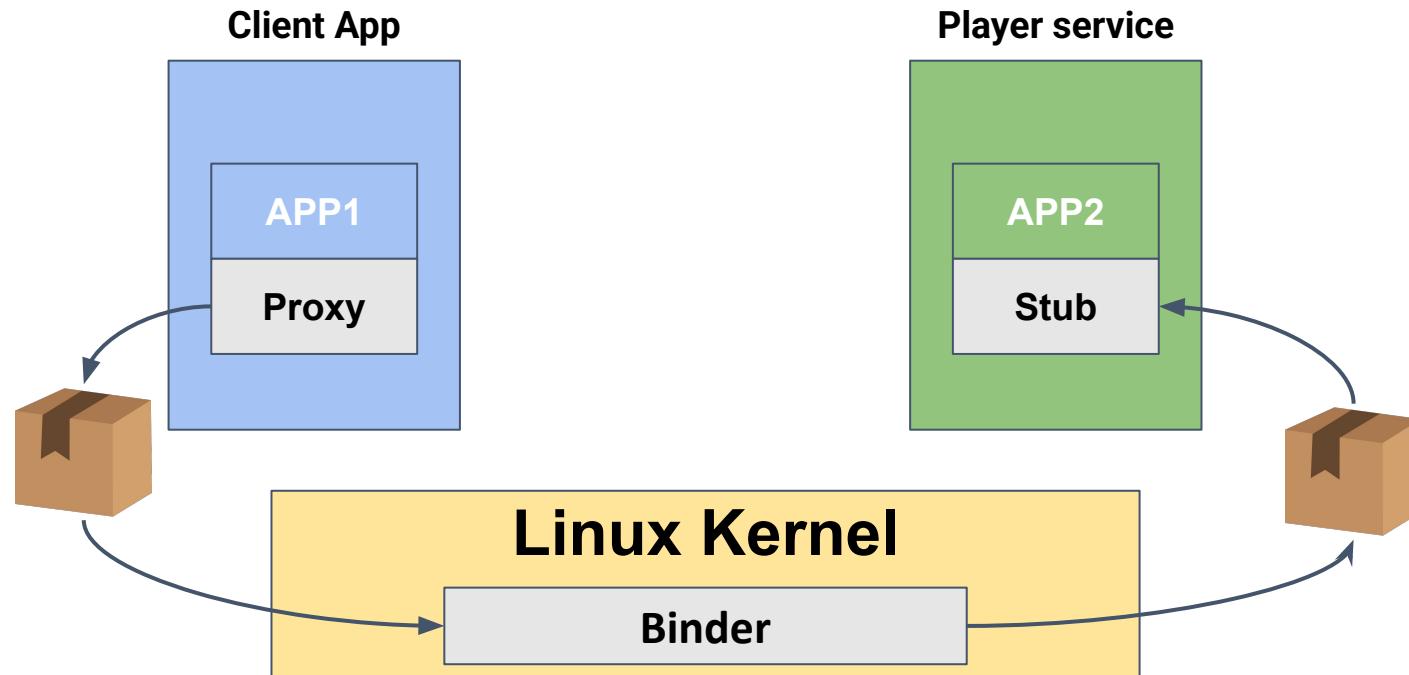


```
IPlayerService.aidl × : 

1 package ru.otus.androidcomponents.player; ✓
2
3 /**
4  * Player service
5  */
6 interface IPlayerService {
7     String getTrackTitle();
8 }
```



# Binder IPC



# AIDL - разрешения на запуск сервиса

```
<service
    android:name=".PlayerService"
    android:foregroundServiceType="mediaPlayback"
    android:enabled="true"
    android:permission="ru.otus.androidcomponents.player.PERMISSION"
    android:exported="true">← Разрешено другим приложениям
    <intent-filter>
        <action android:name="ru.otus.androidcomponents.player.BIND" />
    </intent-filter>
</service>
```

Action для подключения



# AIDL - Реализация сервиса

Работа с Binder сделана за нас

```
117     private val playerService = object : IPlayerService.Stub() {  
118         override fun asBinder(): IBinder = this  
119  
120         override fun getTrackTitle(): String = title  
121     }
```

Реализация нашей логики



# AIDL - Клиент сервиса

Работа с Binder сделана за нас

```
58     override fun onServiceConnected(name: ComponentName?, service: IBinder?) {  
59         val playerService = IPlayerService.Stub.asInterface(service)  
60         if (null != playerService) {  
61             Log.i(TAG, msg: "Service connected")  
62             player = playerService  
63         } else {  
64             Log.e(TAG, msg: "Service interface not created")  
65         }  
66     }
```



# AIDL - Клиент сервиса

```
44     getTitle.setOnClickListener { it: View!  
45         trackTitle.text = player?.trackTitle  
46     }
```

Работаем как с локальным объектом



# Ваши вопросы



если нет вопросов,  
поставьте огонек



# Android Приложение



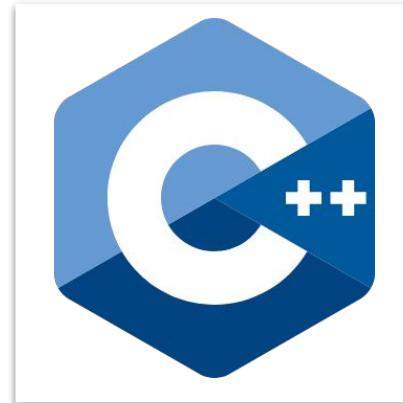
# Android Приложение - инструменты



Kotlin

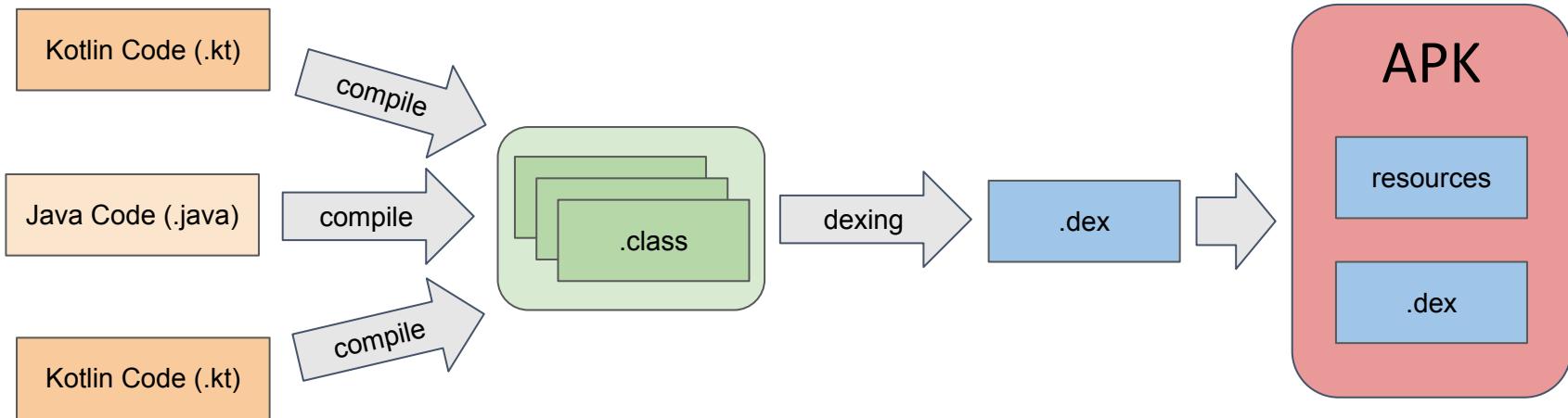


Java



C++

# Компиляция и сборка приложения



Это происходит у разработчика (исходный код, компиляция, сборка)



# Байт-код

```
1 ► fun main() {  
2     println("Hello, World!")  
3 }
```

Kotlin

```
1 ► public class Main {  
2 ►     public static void main(String[] args) {  
3         System.out.println("Hello, World!");  
4     }  
5 }
```

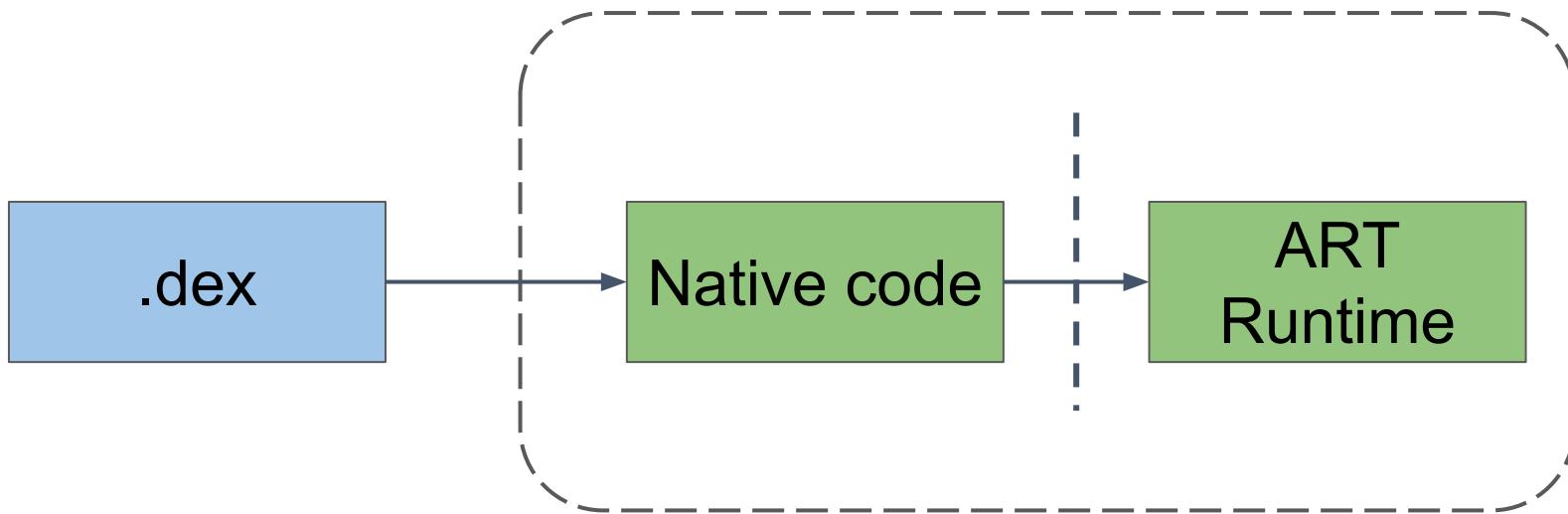
Java

```
7 // access flags 0x19  
8 public final static main()V  
9  
10    L0  
11        LINENUMBER 2 L0  
12        LDC "Hello, World!"  
13        ASTORE 0  
14  
15    L1  
16        GETSTATIC java/lang/System.out : Ljava/io/PrintStream;  
17        ALOAD 0  
18        INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/Object;)V  
19  
20    L2  
21  
22    L3  
23        LINENUMBER 3 L3  
24        RETURN  
25  
26    L4  
27        MAXSTACK = 2  
28        MAXLOCALS = 1
```

Bytecode



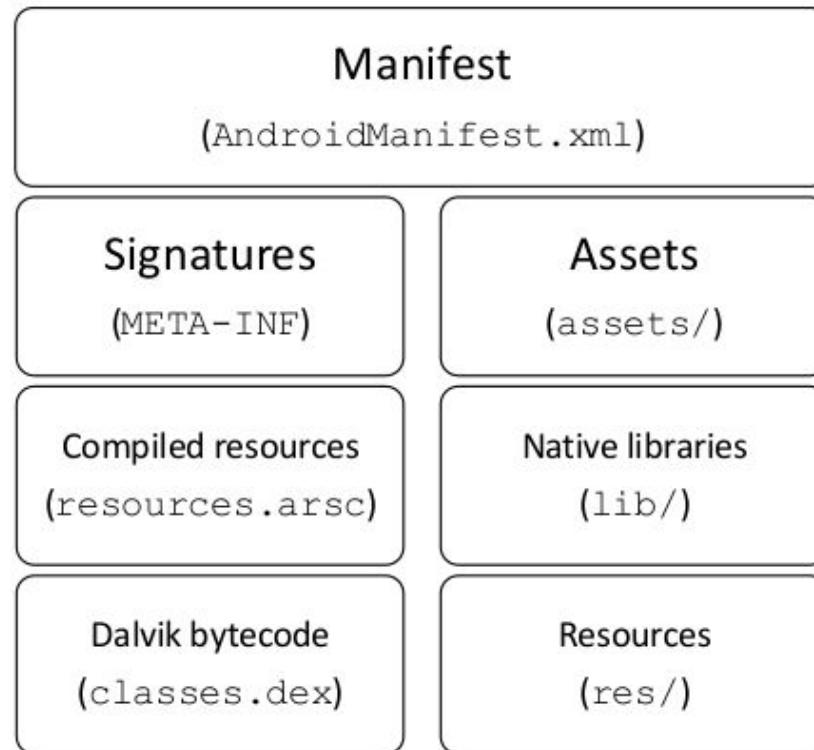
# Запуск приложения



Это происходит на устройстве  
(трансляция, запуск)



# APK



# Ваши вопросы



если нет вопросов,  
поставьте огонек



# Основные компоненты

Activity

(интерактивный  
компонент)

Service

(запущенное действие)

ContentProvider

(доступ к данным  
приложения)

BroadcastReceiver

(реакция на события)



# Важная особенность Android

Приложения могут запускать компоненты  
другого приложения как свои\*

*\* если компонент явно экспортован в манифесте*



# Экспорт компонентов

Экспорт  
разрешен

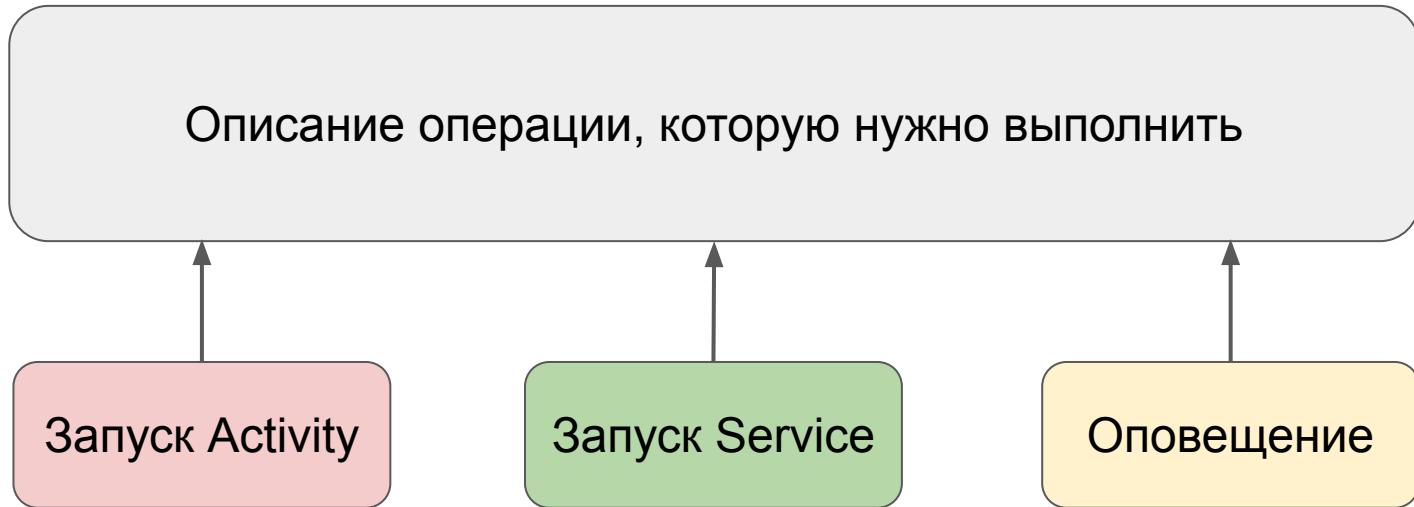
Фильтр  
действия

```
<service
    android:name=".PlayerService"
    android:foregroundServiceType="mediaPlayback"
    android:enabled="true"
    android:permission="ru.otus.androidcomponents.player.PERMISSION"
    android:exported="true">
    <intent-filter>
        <action android:name="ru.otus.androidcomponents.player.BIND" />
    </intent-filter>
</service>
```

*экспорт сервиса в приложении-сервере*



# Intent



[Подробнее об Intent](#)



# Intent

Желаемое действие  
(совпадает с фильтром сервиса)

Запрос

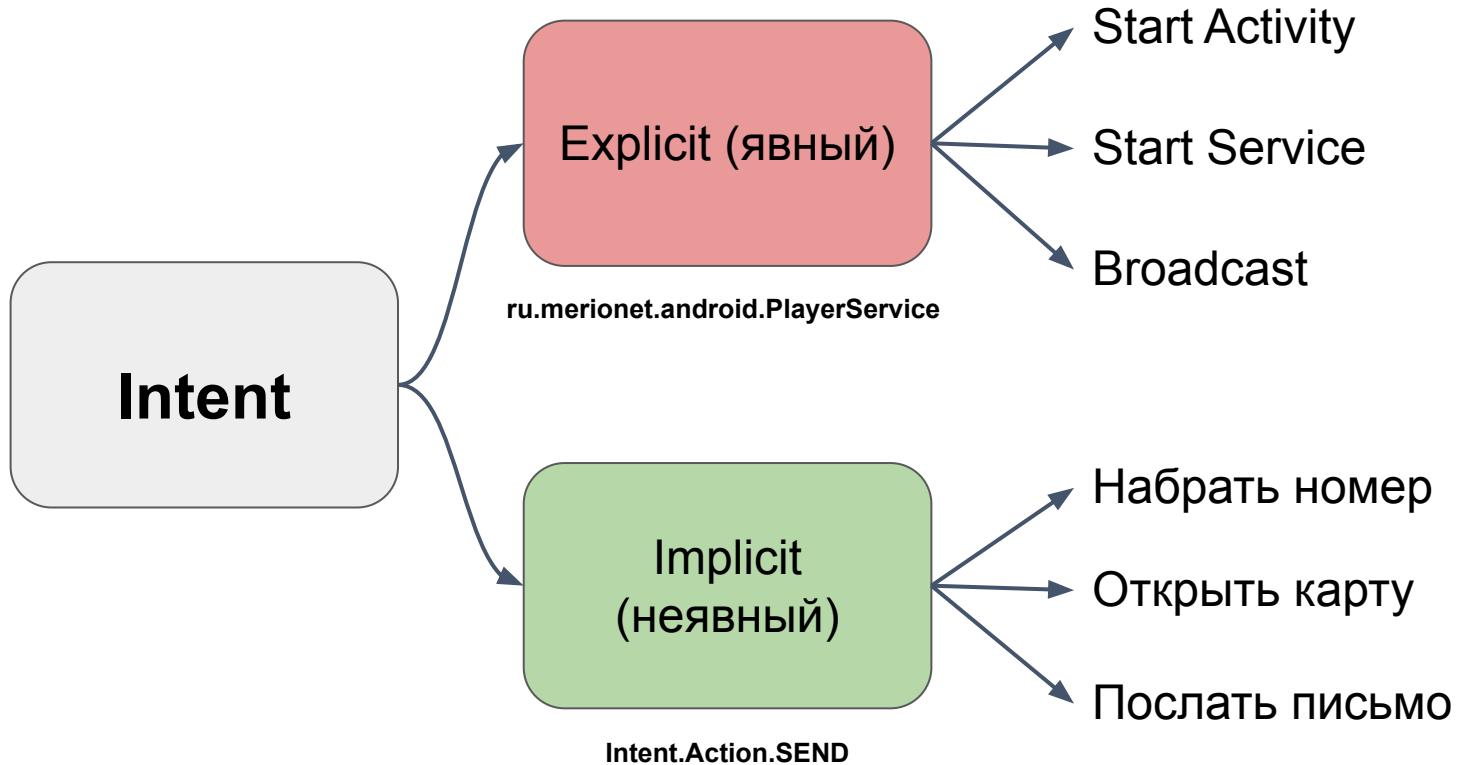


```
val intent = Intent(action: "ru.otus.androidcomponents.player.BIND")
intent.setPackage("ru.otus.androidcomponents.app")  
  
val result = bindService(
    intent,
    connection,
    Context.BIND_AUTO_CREATE
)
```

Идентификатор  
приложения



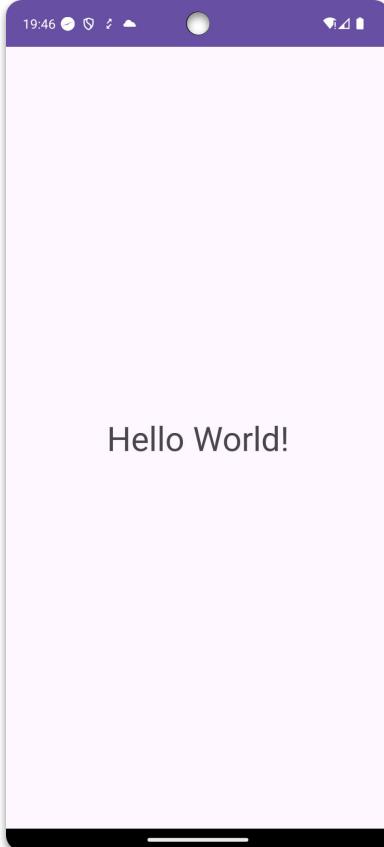
# Intent



# Activity



# Activity



- Экран с пользовательским интерфейсом
- Интерактивная точка входа в приложение
- Таких точек в приложении может быть несколько
- Activity может решать специфичную задачу для других приложений

# Activity

Фильтр

```
<activity  
    android:name=".MainActivity"  
    android:exported="true">  
    <intent-filter>  
        <action android:name="android.intent.action.MAIN" />  
        <category android:name="android.intent.category.LAUNCHER" />  
    </intent-filter>  
</activity>
```

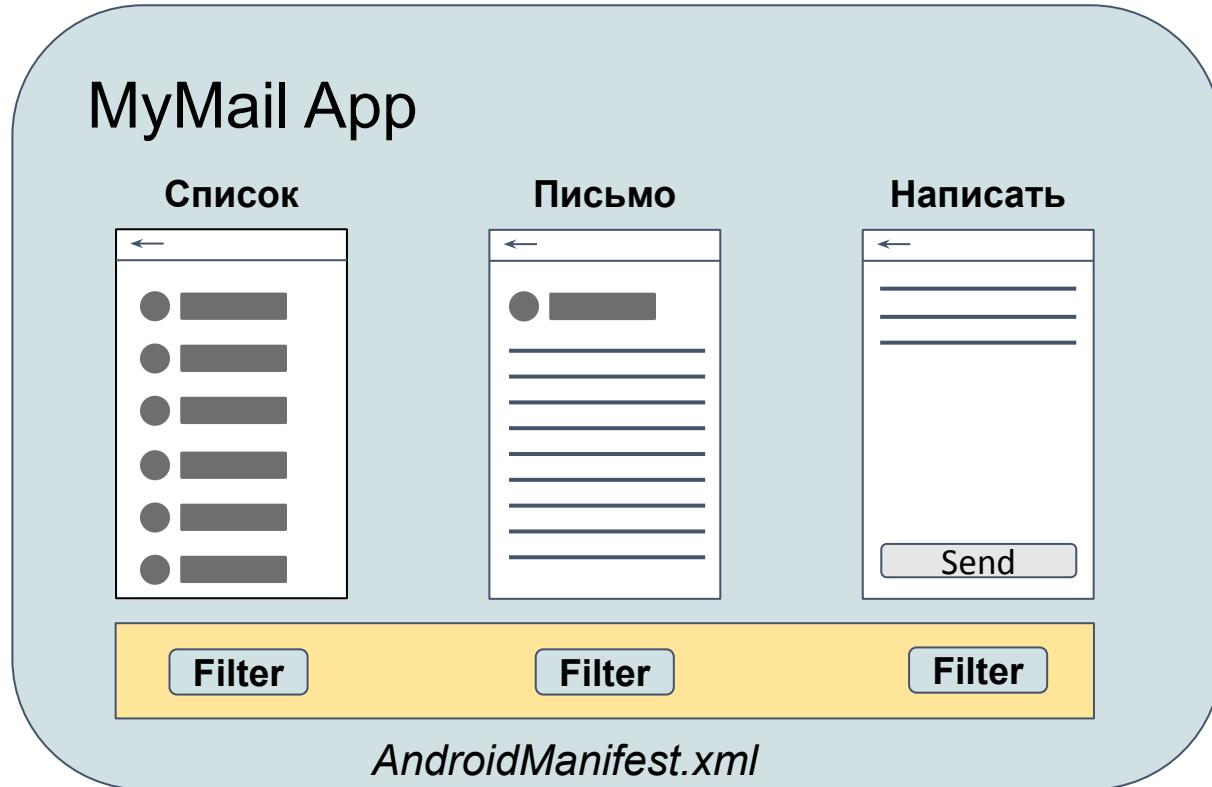
“Просмотр”

↑  
Основная точка входа в приложение

*AndroidManifest.xml*



# Почтовый клиент



# Почтовый клиент

```
<activity
    android:name=".MailListActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity
    android:name=".LetterActivity"
    android:exported="false">
```

```
<activity
    android:name=".NewLetterActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.SENDTO" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Список

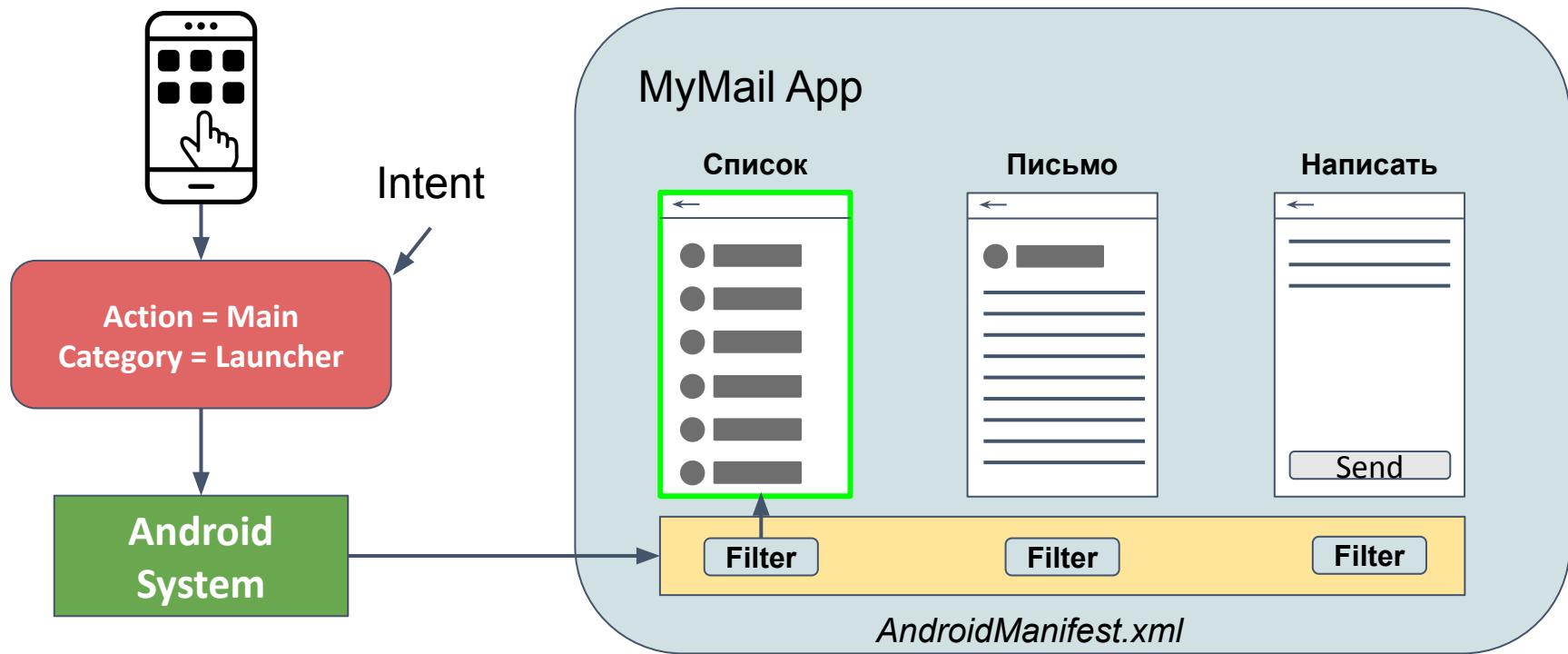
Письмо

Новое  
ПИСЬМО

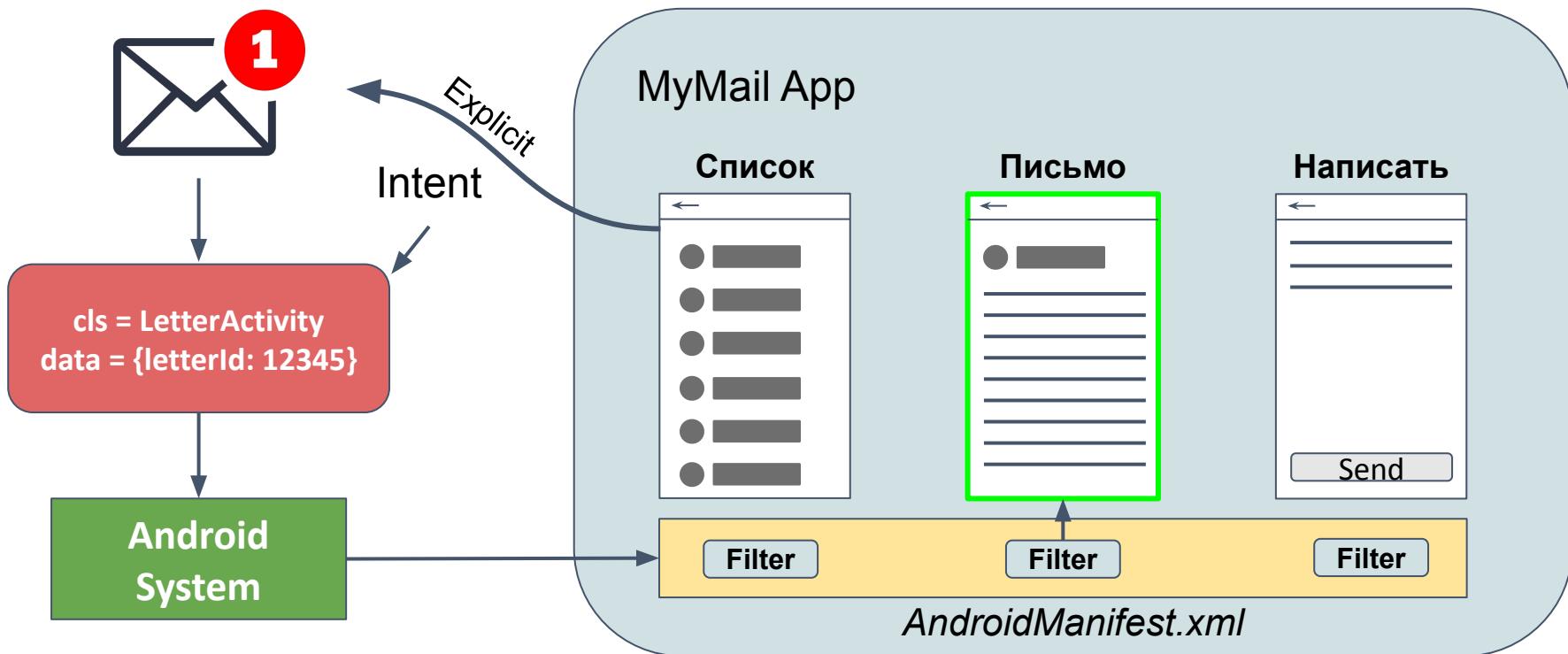
*AndroidManifest.xml*



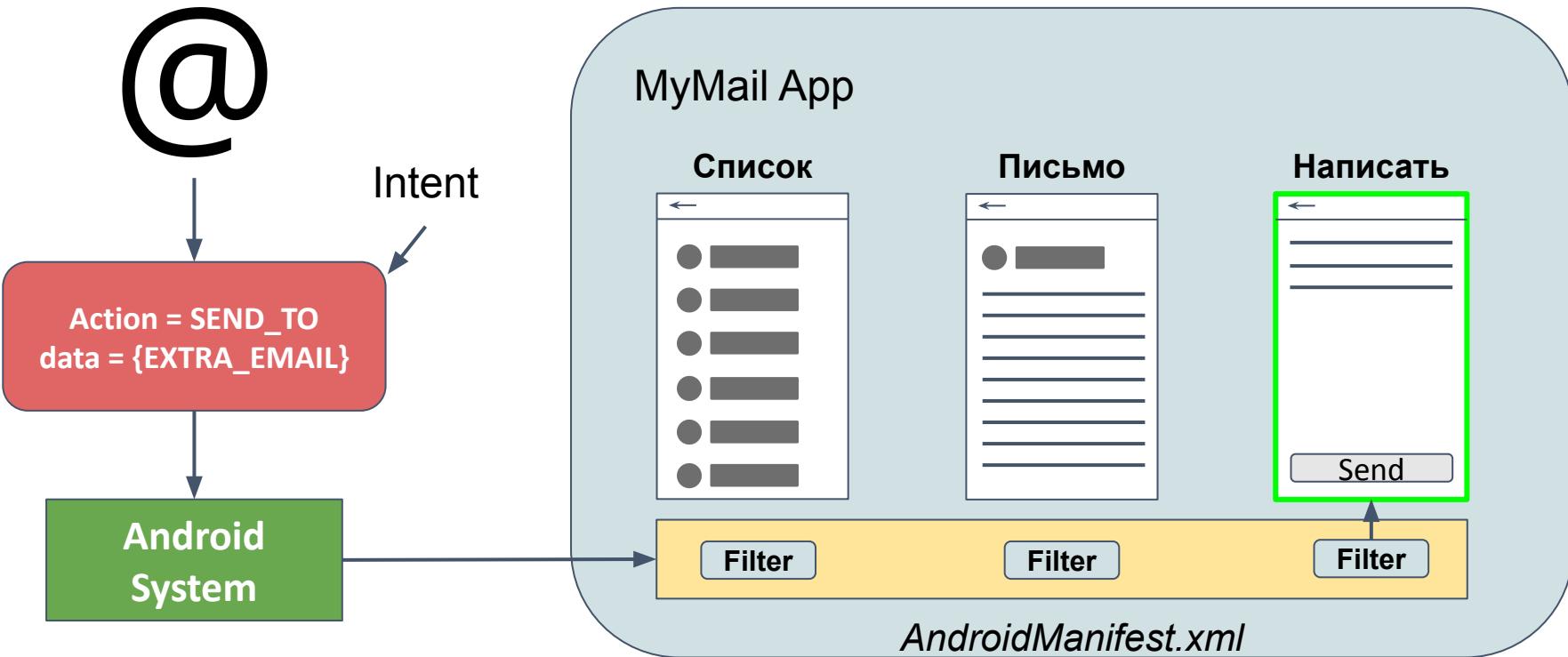
# Запуск из launcher



# Уведомление о новой почте



# Написать письмо



# Почтовый клиент

```
<activity
    android:name=".MailListActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity
    android:name=".LetterActivity"
    android:exported="false">
```

```
<activity
    android:name=".NewLetterActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.SENDTO" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Список

Письмо

Новое  
ПИСЬМО

*AndroidManifest.xml*



# Важная особенность Android

Жизненным циклом приложения (lifecycle)  
управляет операционная система\*

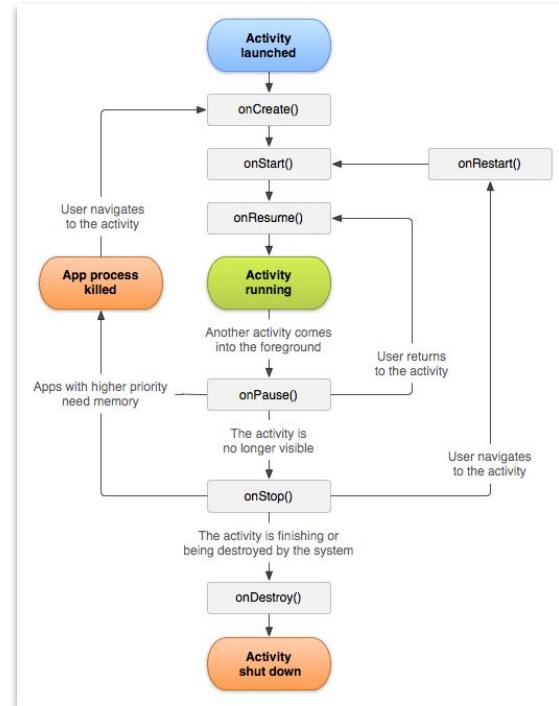
\* *создание и завершение процесса приложения не контролируется  
приложением*



# Component Lifecycle

- ОС создает процесс по нашей просьбе - Intent
- ОС следит за активностью приложения
- ОС завершает процесс приложения исходя из состояния ресурсов

*Компонент получает  
инструкции ОС через функции  
жизненного цикла (lifecycle  
callbacks)*



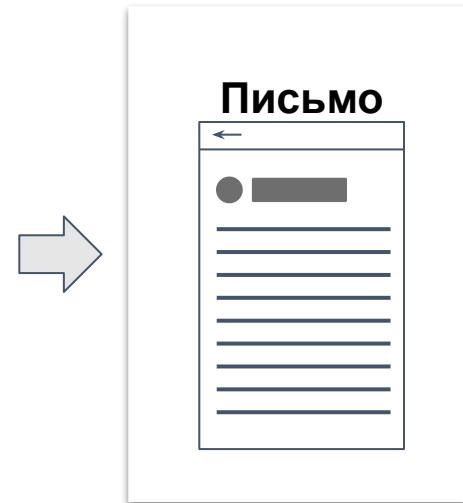
*Activity  
Lifecycle*



# Запуск другого компонента

```
// Go to Letter activity
toLetter.setOnClickListener {
    val intent = Intent(requireActivity(), LetterActivity::class.java)
    intent.putExtra("letterId", 12345)
    startActivity(intent)
}
```

*MailListActivity.kt*



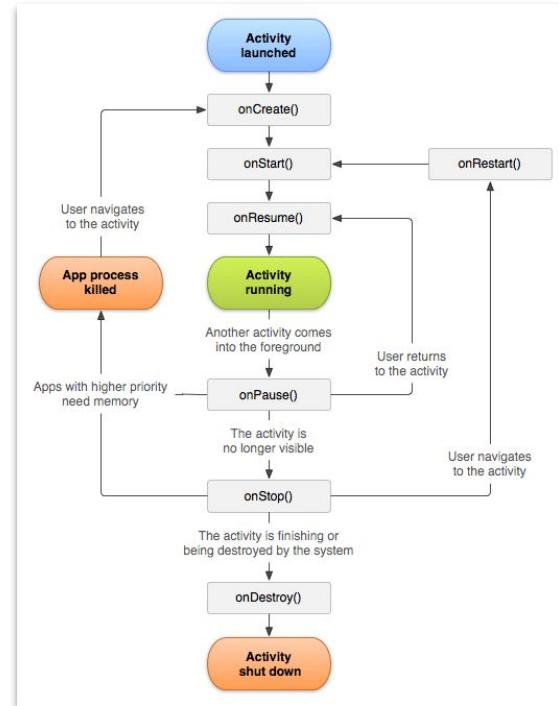
*LetterActivity.kt*



# Component Lifecycle

- ОС создает процесс по нашей просьбе - Intent
- ОС следит за активностью приложения
- ОС завершает процесс приложения исходя из состояния ресурсов

*Компонент получает  
инструкции ОС через функции  
жизненного цикла (lifecycle  
callbacks)*



*Activity  
Lifecycle*



# Component Lifecycle

onCreate()

```
16 > class MainActivity : AppCompatActivity() {  
17  
18     private lateinit var binding: ActivityMainBinding  
19  
20     override fun onCreate(savedInstanceState: Bundle?) {  
21         super.onCreate(savedInstanceState)  
22         binding = ActivityMainBinding.inflate(layoutInflater)  
23         setContentView(binding.root)  
24     }  
25 }
```

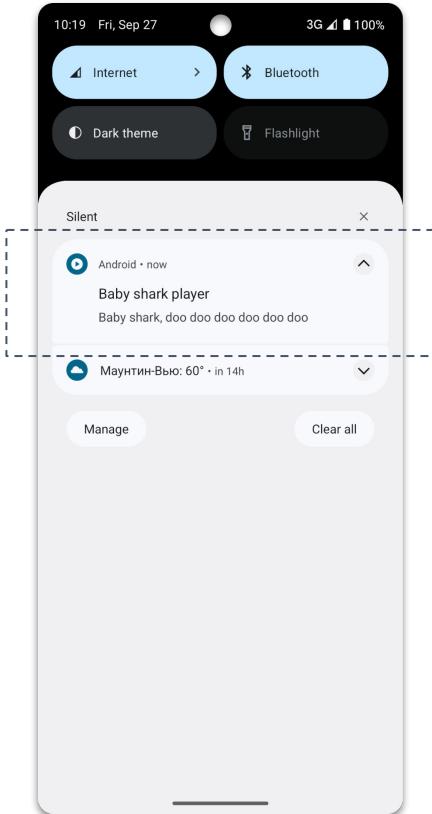


# Lifecycle - контроль ограниченных ресурсов

# Service

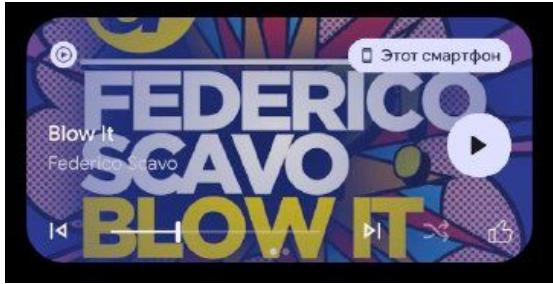


# Service

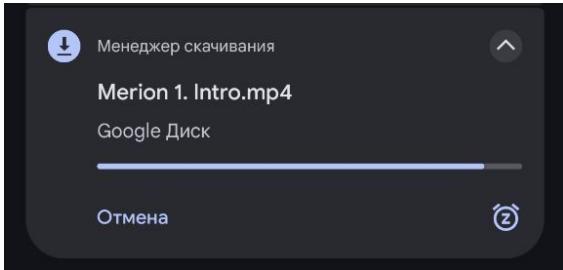


- Запущенная задача, работающая в фоновом режиме
- Не имеет пользовательского интерфейса
- Может выводить уведомление
- Таких точек в приложении может быть несколько
- Service может решать специфичную задачу для других приложений

# Service

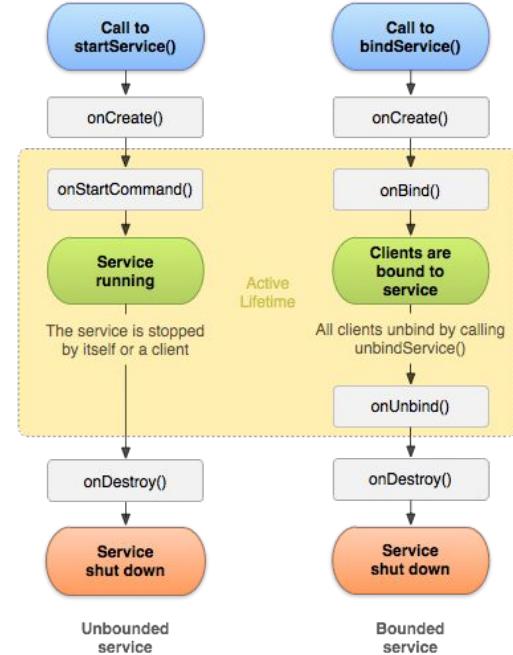


- Проигрывание музыки в фоне
- Скачивание больших файлов
- Выполнение длительных операций



# Service

- Жизненный цикл управляет ОС
- Разные типы сервисов имеют разный жизненный цикл
- Foreground сервис имеет особый приоритет процесса



*Service  
Lifecycle*



# Service

Может  
использоваться  
другими  
приложениями

```
<service
    android:name=".PlayerService"
    android:foregroundServiceType="mediaPlayback"
    android:enabled="true"
    android:permission="ru.otus.androidcomponents.player.PERMISSION"
    android:exported="true">
    <intent-filter>
        <action android:name="ru.otus.androidcomponents.player.BIND" />
    </intent-filter>
</service>
```



# Service

Запускается  
при помощи  
Intent с  
заданными  
свойствами

```
<service
    android:name=".PlayerService"
    android:foregroundServiceType="mediaPlayback"
    android:enabled="true"
    android:permission="ru.otus.androidcomponents.player.PERMISSION"
    android:exported="true">
    <intent-filter>
        <action android:name="ru.otus.androidcomponents.player.BIND" />
    </intent-filter>
</service>
```



# Ваши вопросы



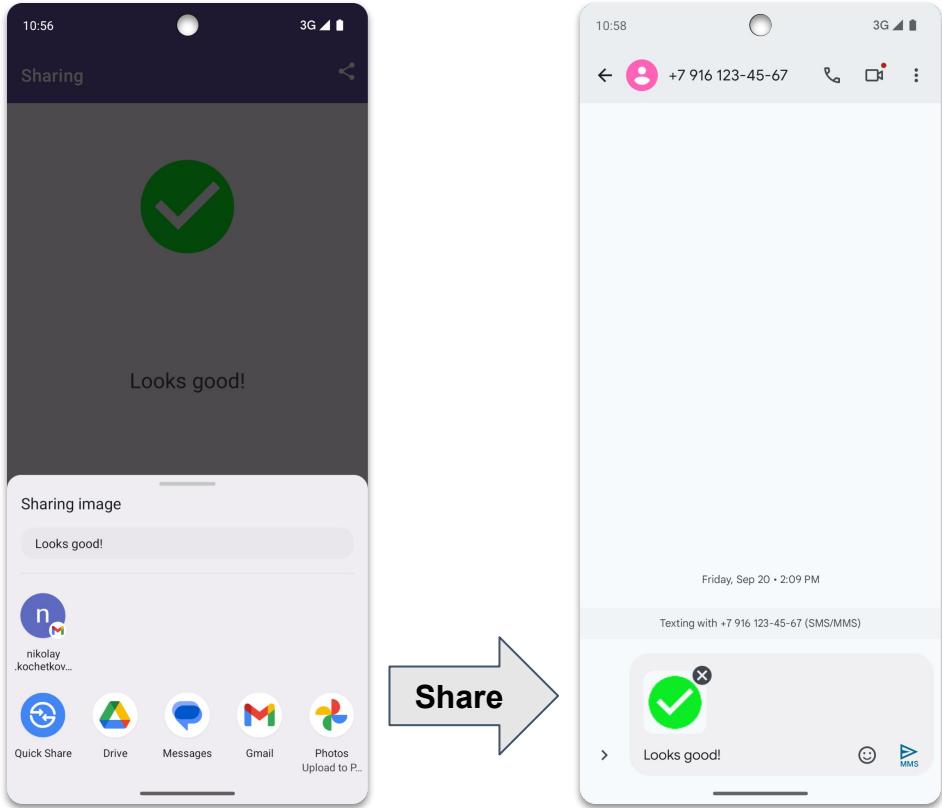
если нет вопросов,  
поставьте огонек



# Content Provider



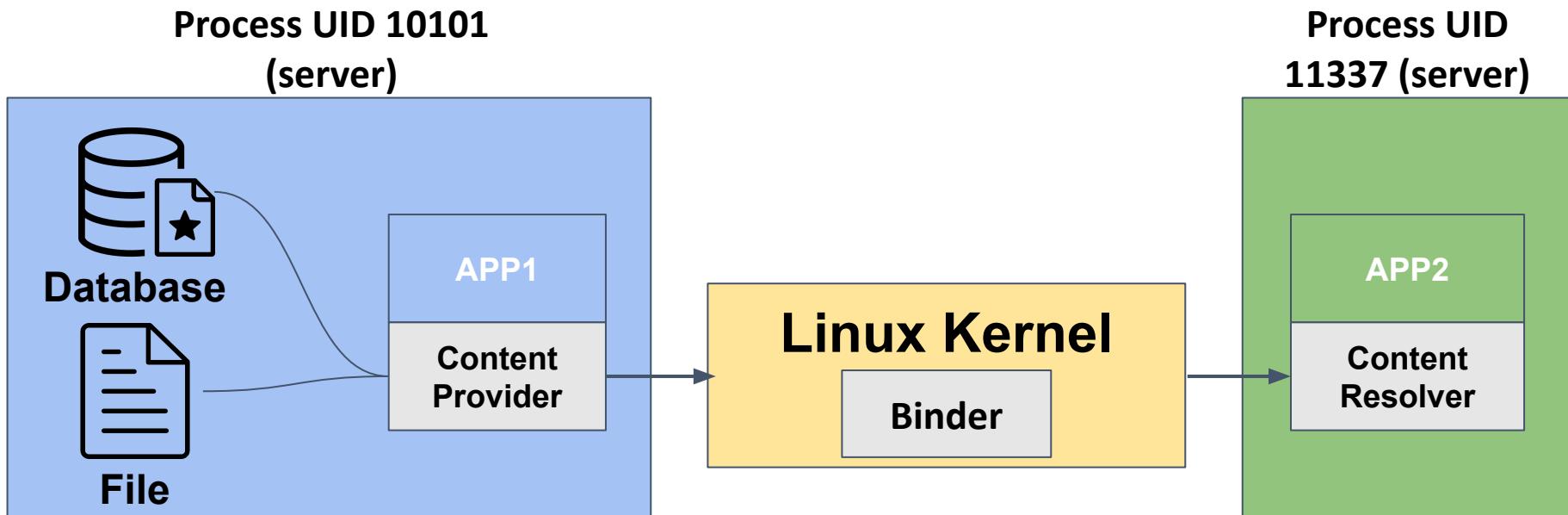
# Content Provider



- Явно открываем доступ к нужным данным
- Клиенту не важно, как реализовано хранение на самом деле
- Общие данные в системе (контакты, календарь, виджеты, словари) доступны через механизм ContentProvider



# Content Provider



# Content Provider

Уникально  
определяется  
URI

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="${applicationId}"
    android:exported="false"
    android:grantUriPermissions="true">

    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/provider_paths" />

</provider>
```



# Чтение календаря

```
// Провайдер идентифицируется URI
val uri: Uri = Calendars.CONTENT_URI
// То же самое, что WHERE в SQL
val selection = "(${Calendars.VISIBLE}) = 1 AND (${Calendars.ACCOUNT_TYPE} != ?))"
// Аргументы для знаков вопроса `selection`
val selectionArgs: Array<String> = arrayOf(CalendarContract.ACCOUNT_TYPE_LOCAL)
// Сортировка записей
val sort = Calendars.CALENDAR_DISPLAY_NAME
// Запросить курсор
val cursor: Cursor? = requireActivity().contentResolver.query(
    uri,
    EVENT_PROJECTION,
    selection,
    selectionArgs,
    sort
)
```

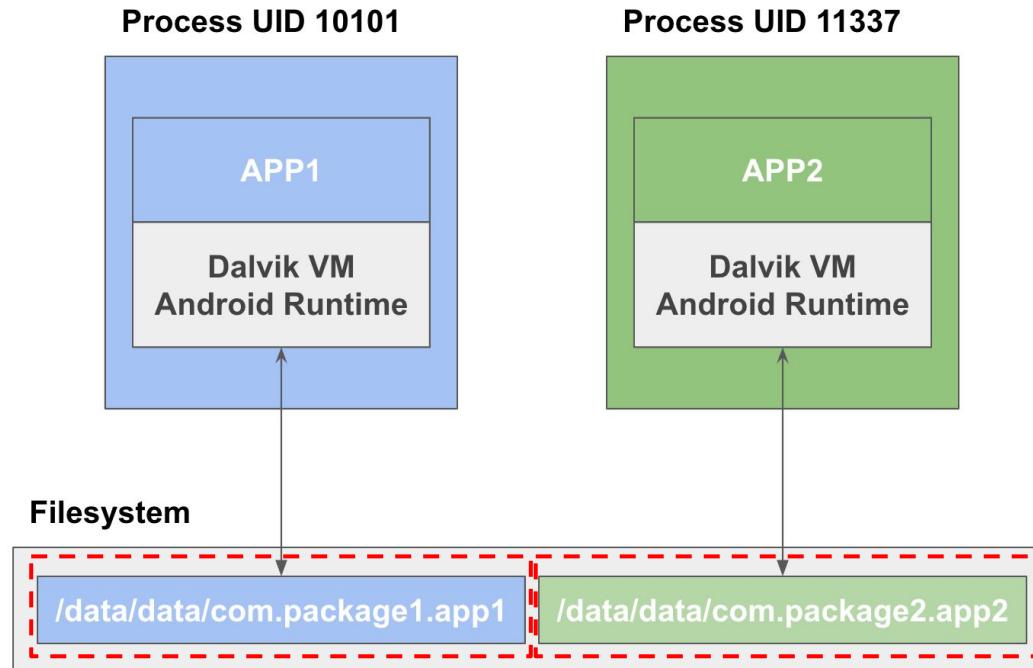


# File Provider

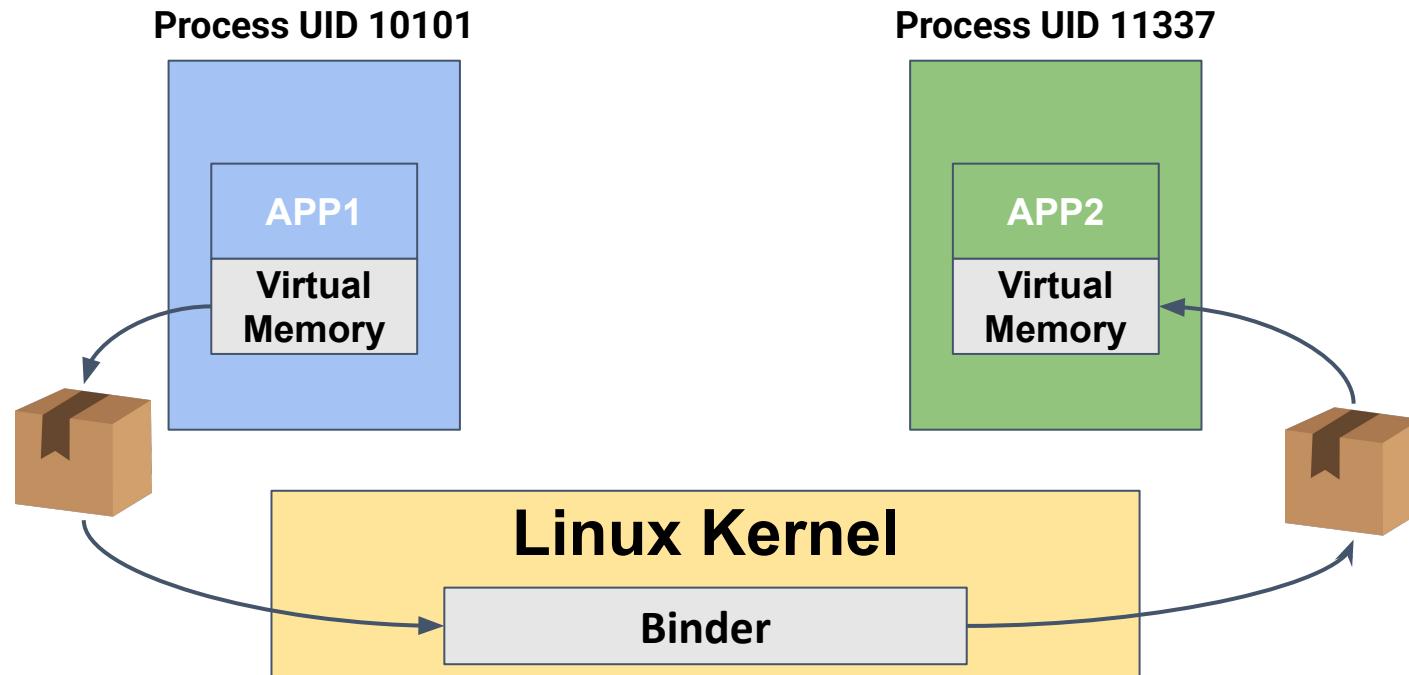
- Основана на ContentProvider
- Скрывает внутреннюю структуру файлов
- Дает доступ на чтение или запись к выбранному файлу внешнему миру
- Простота использования



# Application Sandbox - диск



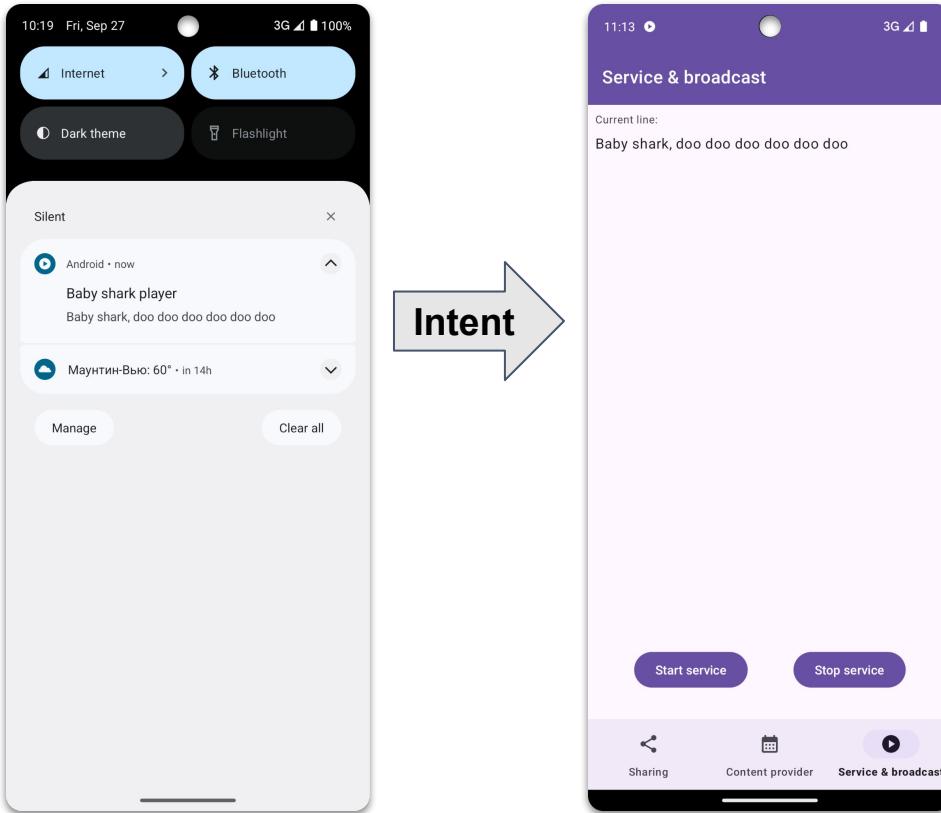
# Binder IPC



# Broadcast Receiver



# Broadcast Receiver



- Простой способ передачи сообщений через IPC
- Может получать системные\* и внутренние оповещения
- Регистрируется в манифесте или динамически



# Системные уведомления

Доступен извне

Настроен на  
определенные  
события

```
<receiver  
    android:name=".LocaleBroadcastReceiver"  
    android:exported="true">  
  
    <intent-filter>  
        <action android:name="android.intent.action.LOCALE_CHANGED" />  
    </intent-filter>  
</receiver>
```



# Внутренние уведомления

```
// Broadcast current message
val intent = Intent(SONG_ACTION)
intent.putExtra(SONG_LINE, message)
LocalBroadcastManager
    .getInstance( context: this)
    .sendBroadcast(intent)
```

Отправляем...



# Внутренние уведомления

```
private fun listenToSong() {  
    val receiver = object : BroadcastReceiver() {  
        override fun onReceive(context: Context?, intent: Intent?) {  
            binding.currentLine.text = intent?.getStringExtra(PlayerService.SONG_LINE)  
        }  
    }  
    LocalBroadcastManager.getInstance(requireContext()).registerReceiver(  
        receiver,  
        IntentFilter(PlayerService.SONG_ACTION) ← Фильтр  
    )  
}
```

...получаем!



# Ваши вопросы



если нет вопросов,  
поставьте огонек



# Подведём итоги занятия



# Маршрут вебинара

Приложение и Application Sandbox

Обмен данными между приложениями

Особенности приложений ОС Android

Про курс

Основные компоненты приложения



**Мы ещё не закончили,  
но вы уже можете  
заполнить опрос о занятии**



Давайте  
узнаем больше  
о курсе





 Специализация

# Android Developer

**Создайте с нуля первые мобильные приложения на Kotlin и  
освойте продвинутые технологии для решения Middle задач**

<https://otus.ru/lessons/spec-android/>



**Пожалуйста, заполните  
опрос о мероприятии!**

**Мы читаем все ваши сообщения  
и берем их в работу**



# Приходите учиться в OTUS!

