



Android. Основы создания проектов



Митрохин Максим Александрович
Кузнецов Алексей Валерьевич
Кафедра ВТ ПГУ
E-mail: vt@pnzgu.ru



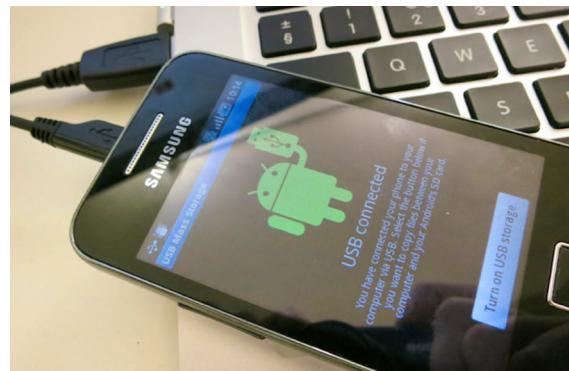
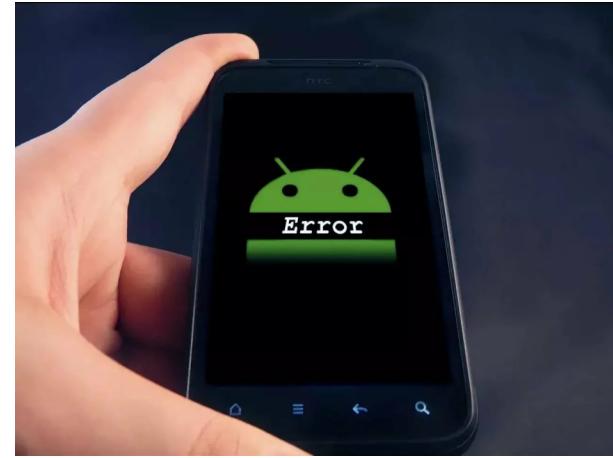


Разработка ПО под Android

Ожидание



Реальность



```
2015-07-18 19:57:43.094 2004-2004/com.example.testapplication I/ActivityManager: STARTED: Intent{ act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=com.example.testapplication/.MainActivity }
```

```
Process: com.example.testapplication, PID: 2004
java.lang.RuntimeException: Unable to start activity ComponentInfo{com.example.testapplication/com.example.MainActivity}: java.lang.NullPointerException: Attempt to invoke virtual method 'int java.lang.String.length()' on a null object reference
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2179)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2099)
    at android.app.ActivityThread.-wrap1(ActivityThread.java:2089)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1360)
    at android.os.Handler.dispatchMessage(Handler.java:102)
    at android.os.Looper.loop(Looper.java:145)
    at android.app.ActivityThread.main(ActivityThread.java:6494)
    at java.lang.reflect.Method.invoke(Native Method)
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:910)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:896)
```

```
at com.example.testapplication.MainActivity.onCreate(MainActivity.java:11)
at android.app.Activity.performCreate(Activity.java:6251)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1145)
at android.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:910)
at com.example.testapplication.MainActivity.onStart(MainActivity.java:11)
at android.app.Activity.performStart(Activity.java:6084)
at android.app.Instrumentation.callActivityOnStart(Instrumentation.java:1102)
at android.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:910)
at android.app.ActivityThread.main(ActivityThread.java:6494)
at java.lang.reflect.Method.invoke(Native Method)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:910)
at android.os.ZygoteInit.main(ZygoteInit.java:896)
```

Операционная система Android



Android — операционная система для интерактивных устройств (смартфонов, планшетов, наручных часов, фитнес-браслетов, игровых приставок, телевизоров, автомобильных развлекательных систем и бытовых роботов).

Изначально разрабатывалась компанией Android, Inc., которую в 2005 г. приобрела Google.

Современная платформа Android – это программный стек, который включает операционную систему, программное обеспечение промежуточного слоя (middleware), а также основные пользовательские приложения, входящие в состав мобильного телефона (календарь, карты, браузер, базы данных контактов, сообщений SMS и др.).



А зачем еще +1 операционная система?

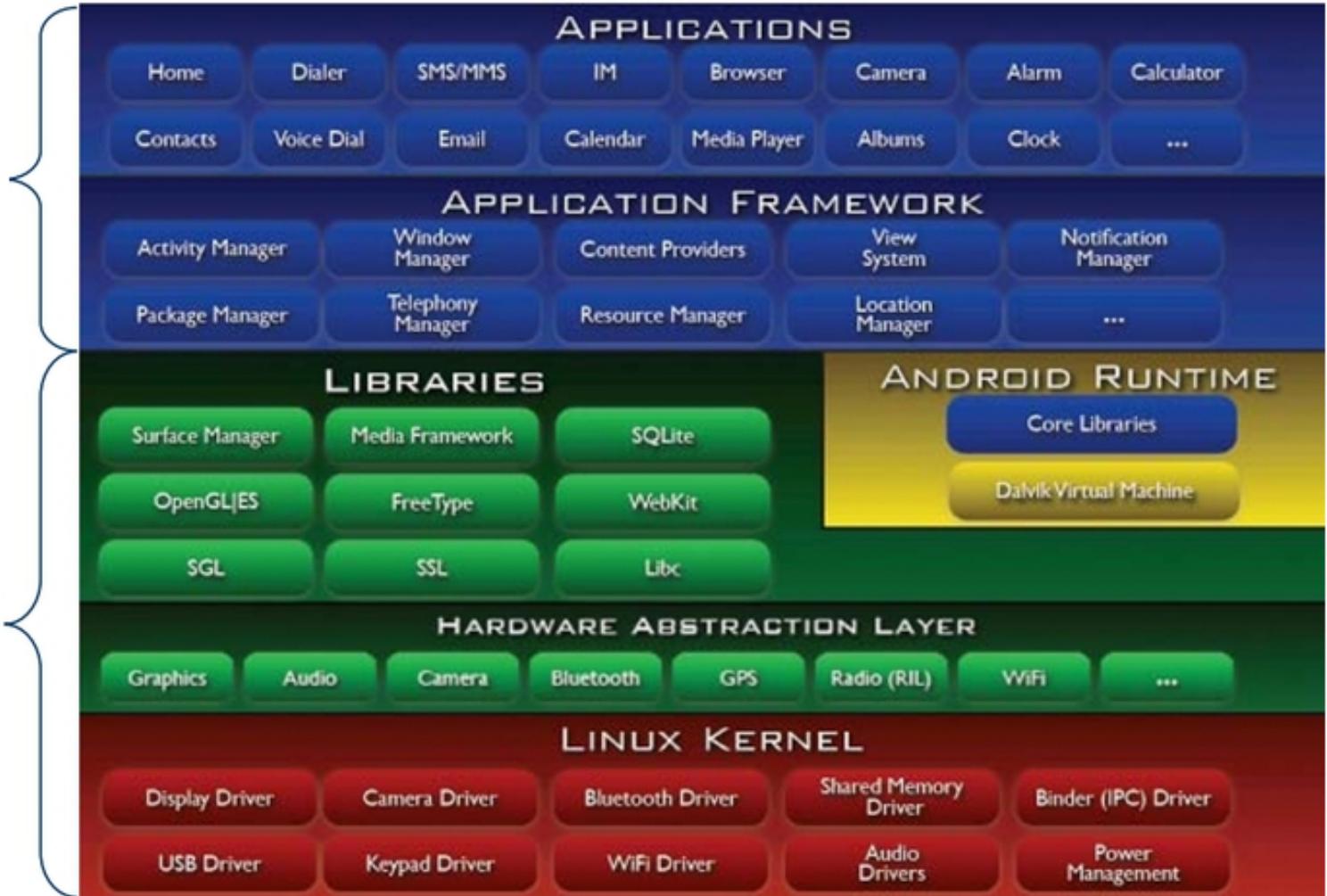




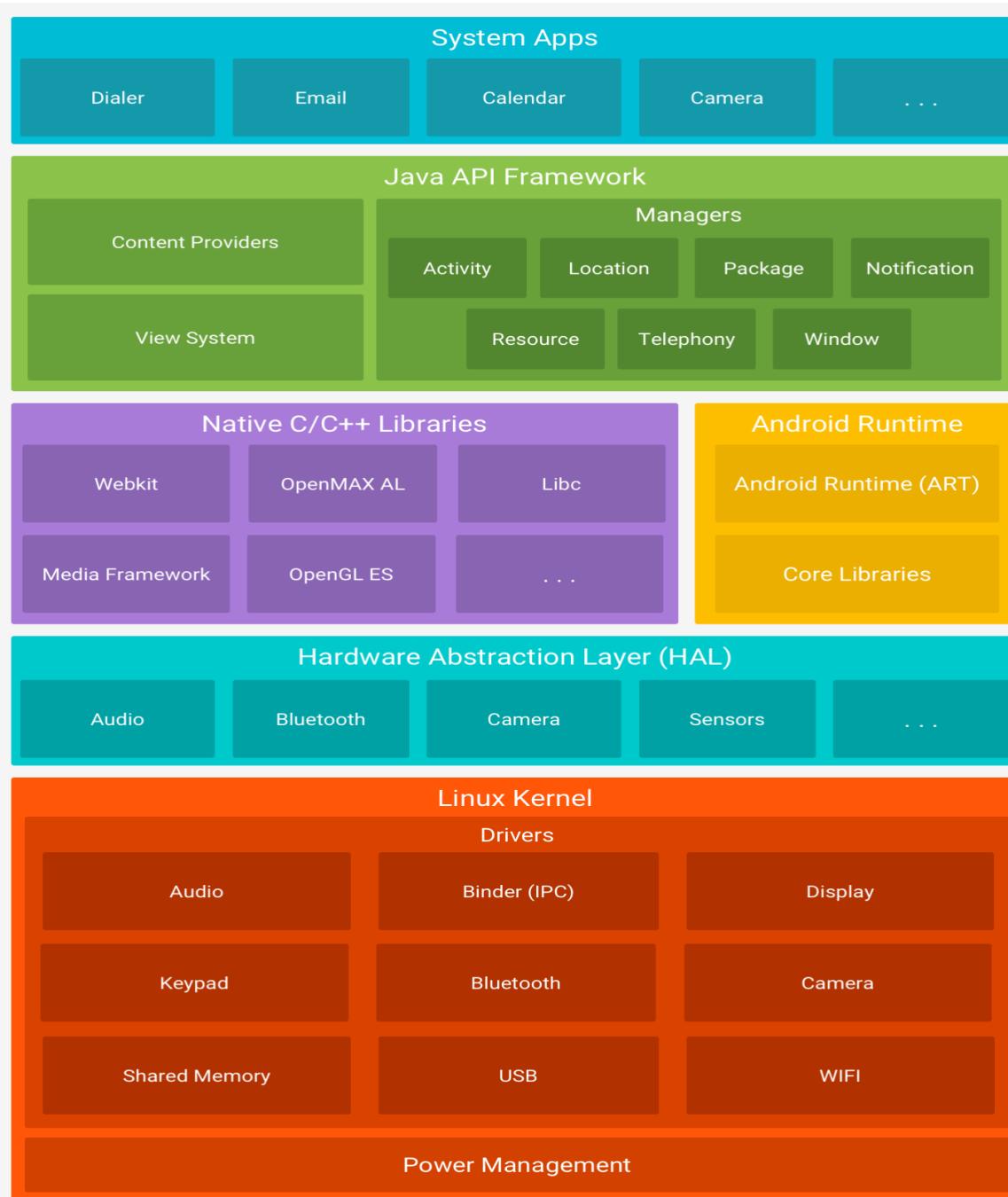
Архитектура ОС Android

Java/Kotlin

C/ASM

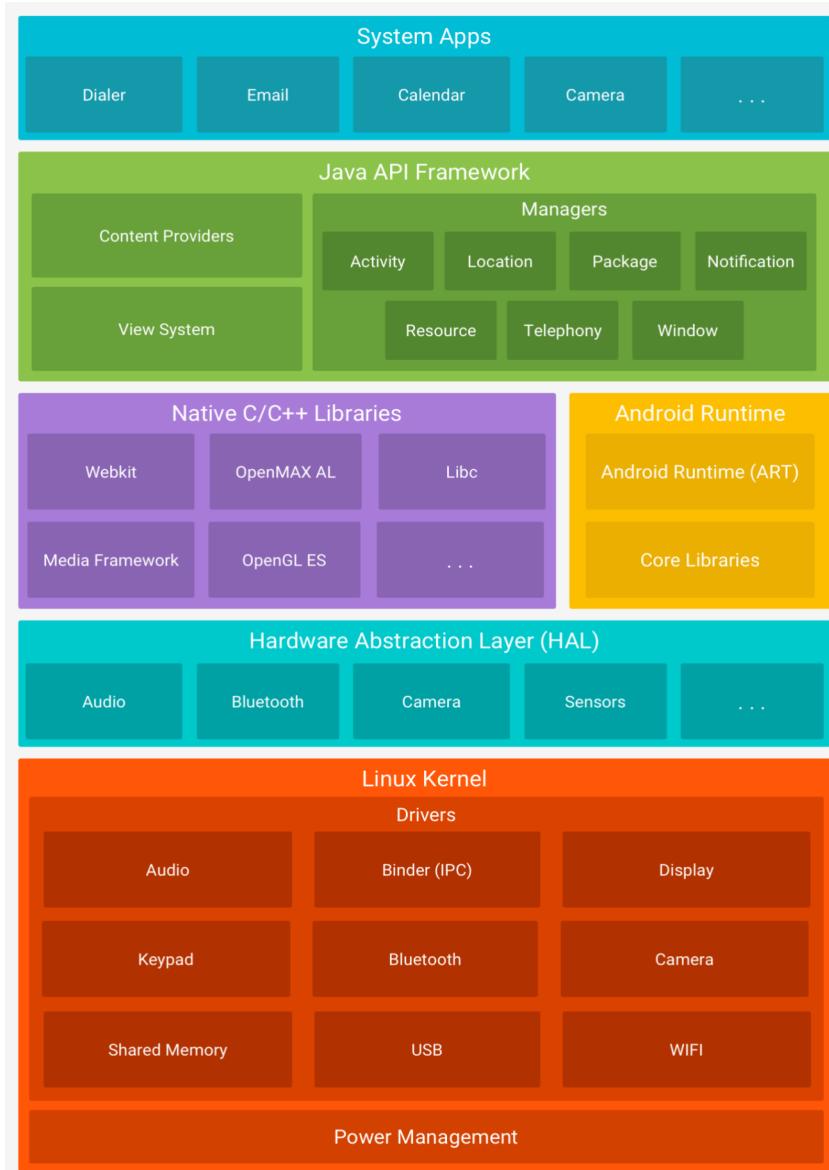


Архитектура ОС Android



На что похоже?

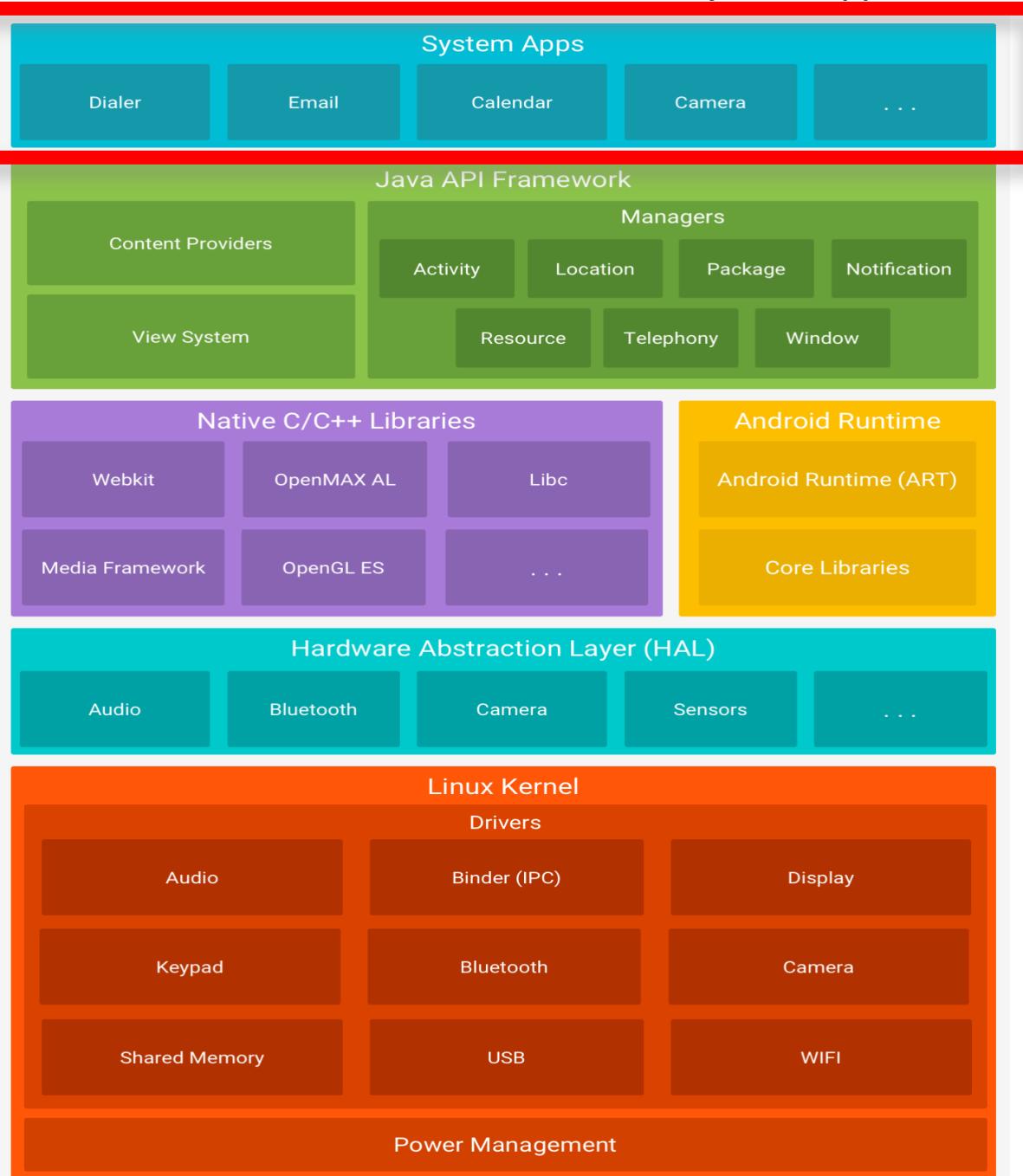
Архитектура ОС Android



System Apps

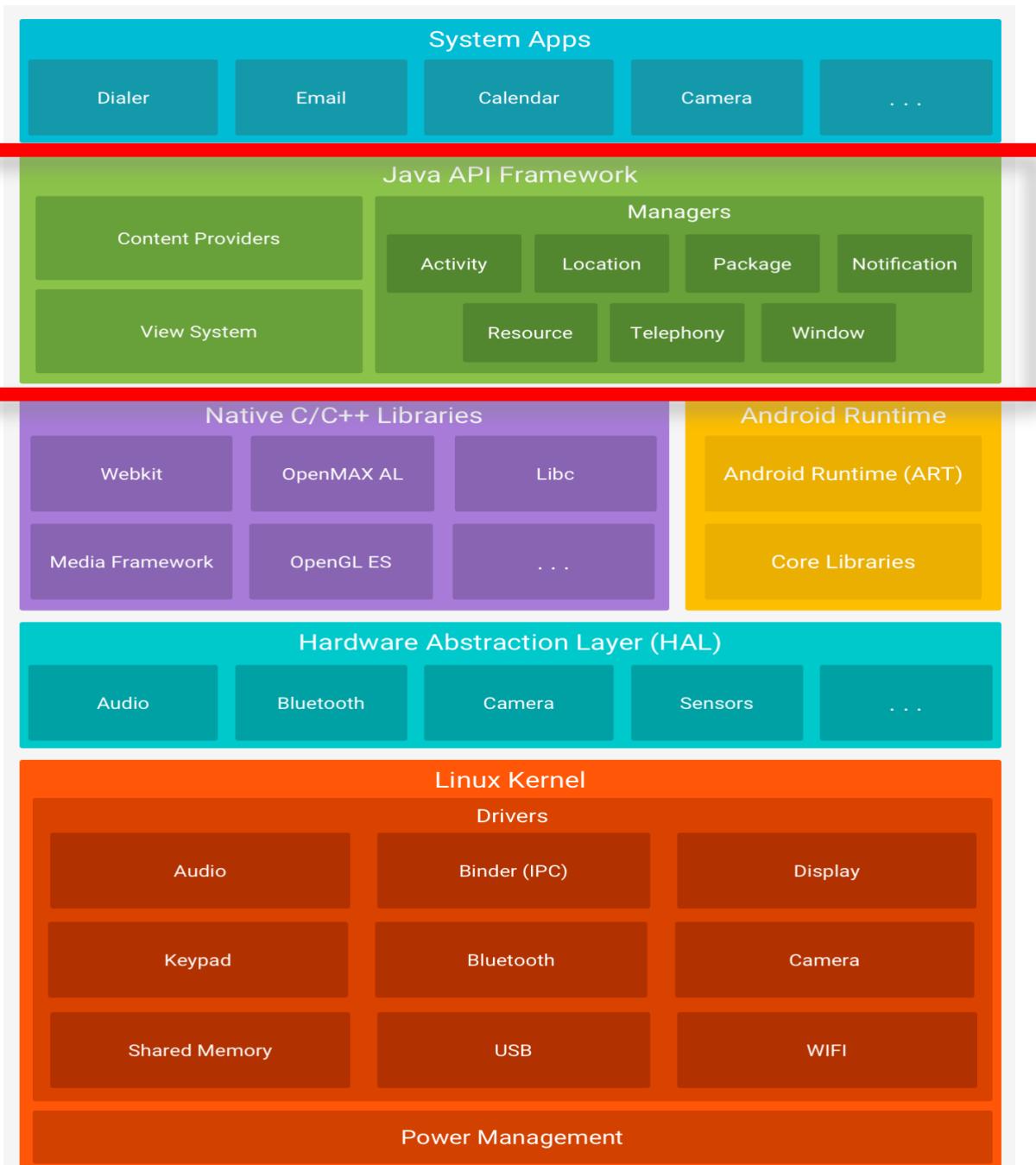


Тут будут и написанные нами приложения



Системные приложения и другие приложения:
Верхний уровень, включающий предустановленные и скачиваемые из магазина приложения.

Java API Framework (Application Framework)



Инфраструктура

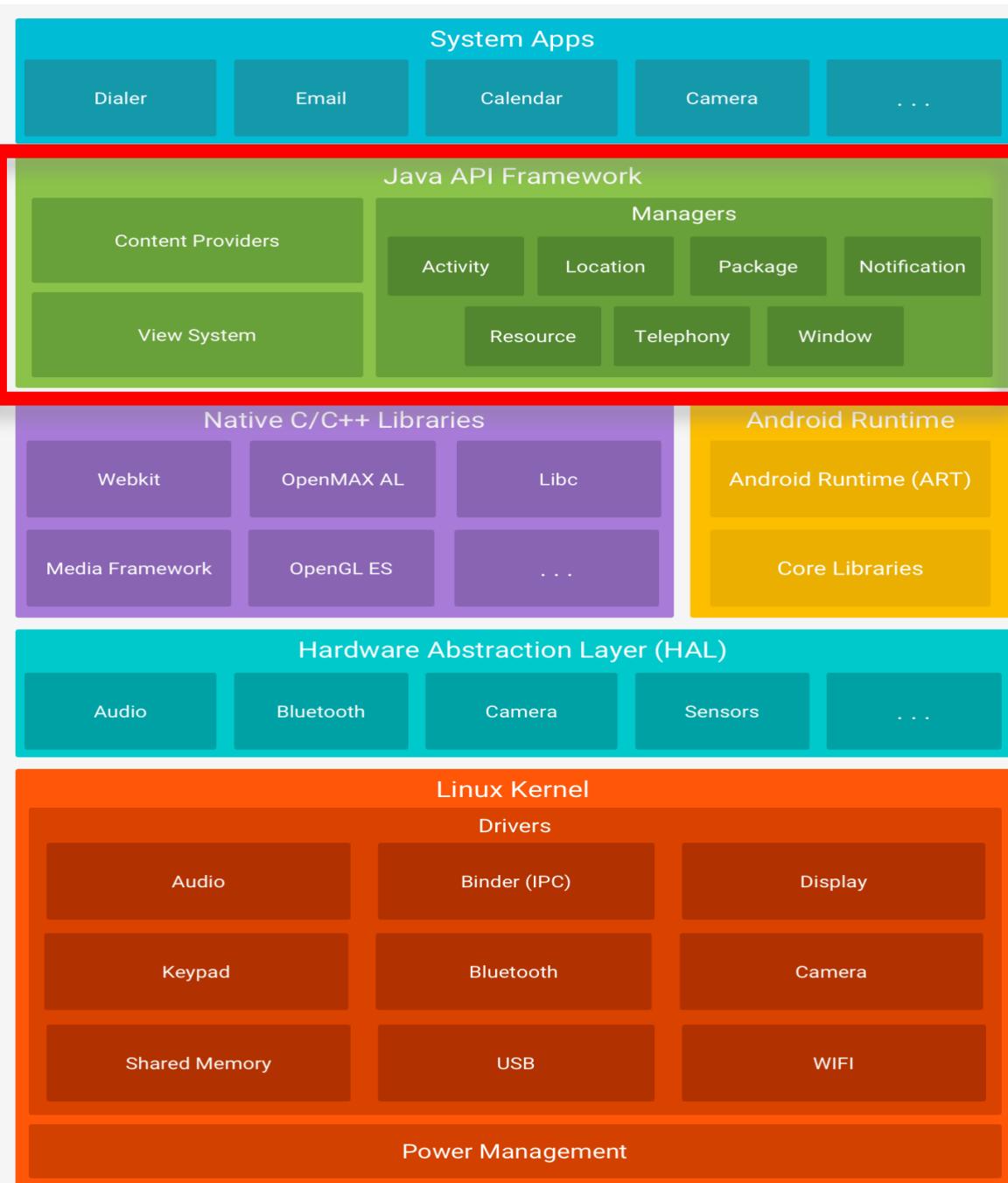
приложений:

Включает системные сервисы и компоненты, которые обеспечивают работу всех приложений.

Набор API, написанный на языке Java и предоставляющий разработчикам доступ ко всем функциям ОС Android.

Эти API-интерфейсы образуют строительные блоки, необходимые для создания приложений, упрощая повторное использование основных, модульных, системных компонентов и сервисов.

Java API Framework (Application Framework)



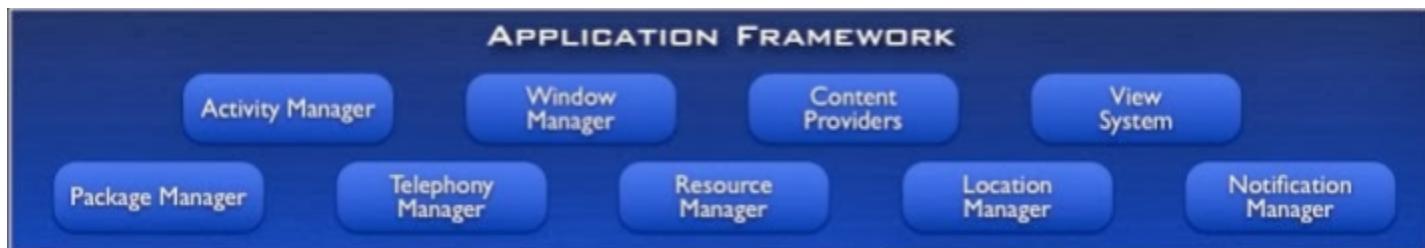
Сервисы (службы) реализуют основные возможности Android.

Content Providers - позволяет приложению получать доступ к данным из других приложений или обмениваться собственными данными, т.е. предоставляет механизм для обмена данными между приложениями. Управляющие данными, которые одни приложения открывают для других, чтобы те могли их использовать для своей работы;

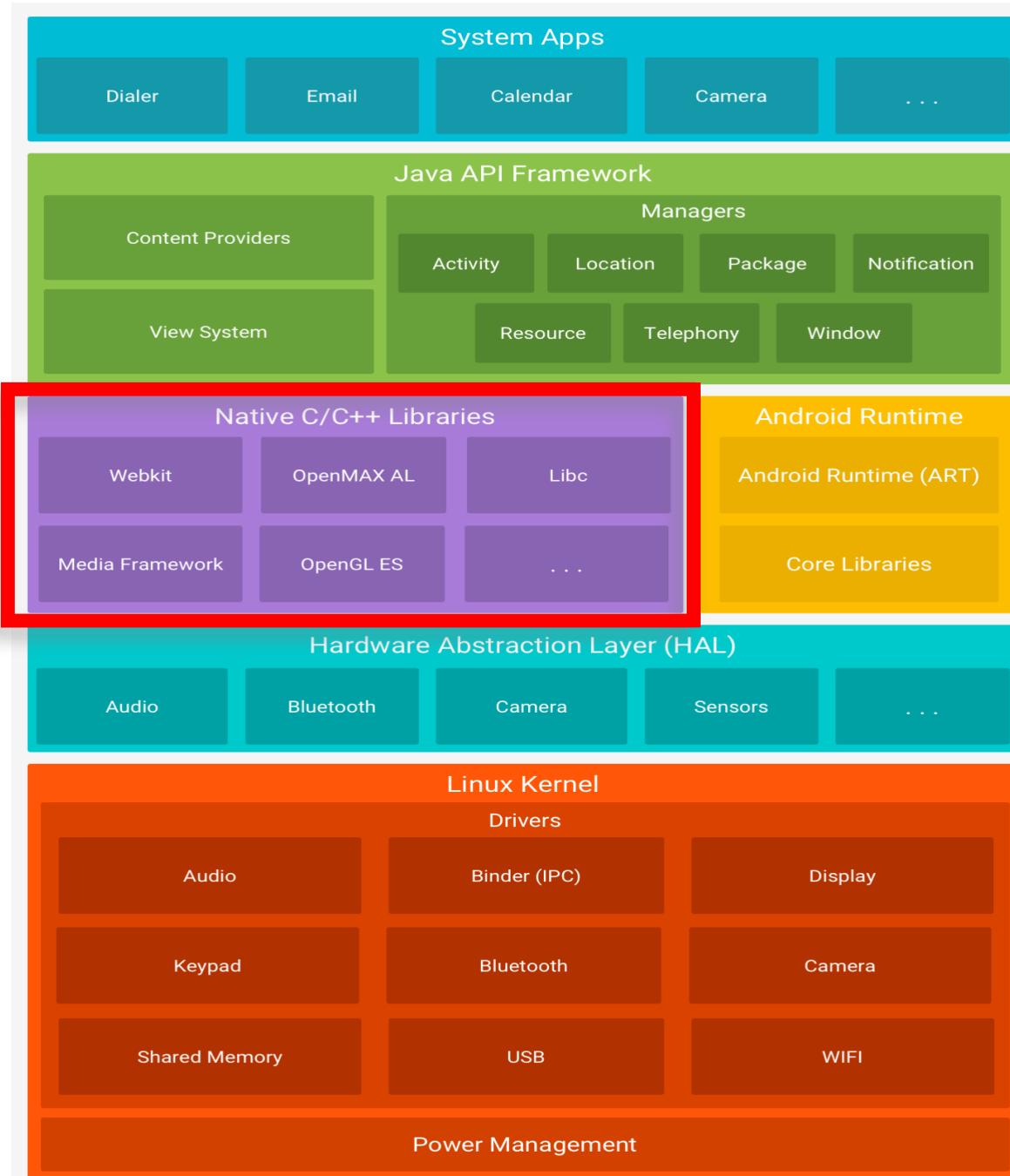


API-интерфейсы для доступа к системным компонентам и сервисам:

- **Activity Manager** - управляет жизненным циклом приложения и обеспечивает общий навигационный стек обратных вызовов, сохраняющий историю работы с действиями, предоставляющий систему навигации по действиям;
- **Window Manager** - управляет окнами и является абстракцией библиотеки Surface Manager.
- **View System** - содержит строительные блоки для создания пользовательского интерфейса приложения (списки, тексты, кнопки итд.), а также управляет событиями элементов пользовательского интерфейса;
- **Package Manager** - управляет различными видами информации, связанными с пакетами приложений, которые в настоящее время установлены на устройстве;
- **Telephony Manager** - позволяет приложению использовать возможности телефонии;
- **Resource Manager** - обеспечивает доступ к таким ресурсам, как локализованные строки, растровые изображения, графика и макеты. обеспечивающий доступ к ресурсам без функциональности (не несущим кода);
- **Location Manager** - возможность определения местоположения. позволяющий приложениям периодически получать обновленные данные о текущем географическом положении устройства;
- **Notification Manager** - отображение уведомлений в строке состояния. позволяющий приложениям отображать собственные уведомления для пользователя в строке состояния.



Native Libraries



Платформенные библиотеки:

Предоставляют ключевые функции, например, для работы с графикой, базами данных и другими системными задачами.

Приложения Android могут разрабатываться с использованием языков высокого уровня, таких как Java или Kotlin, а затем интегрировать или создавать нативные библиотеки, используя C/C++

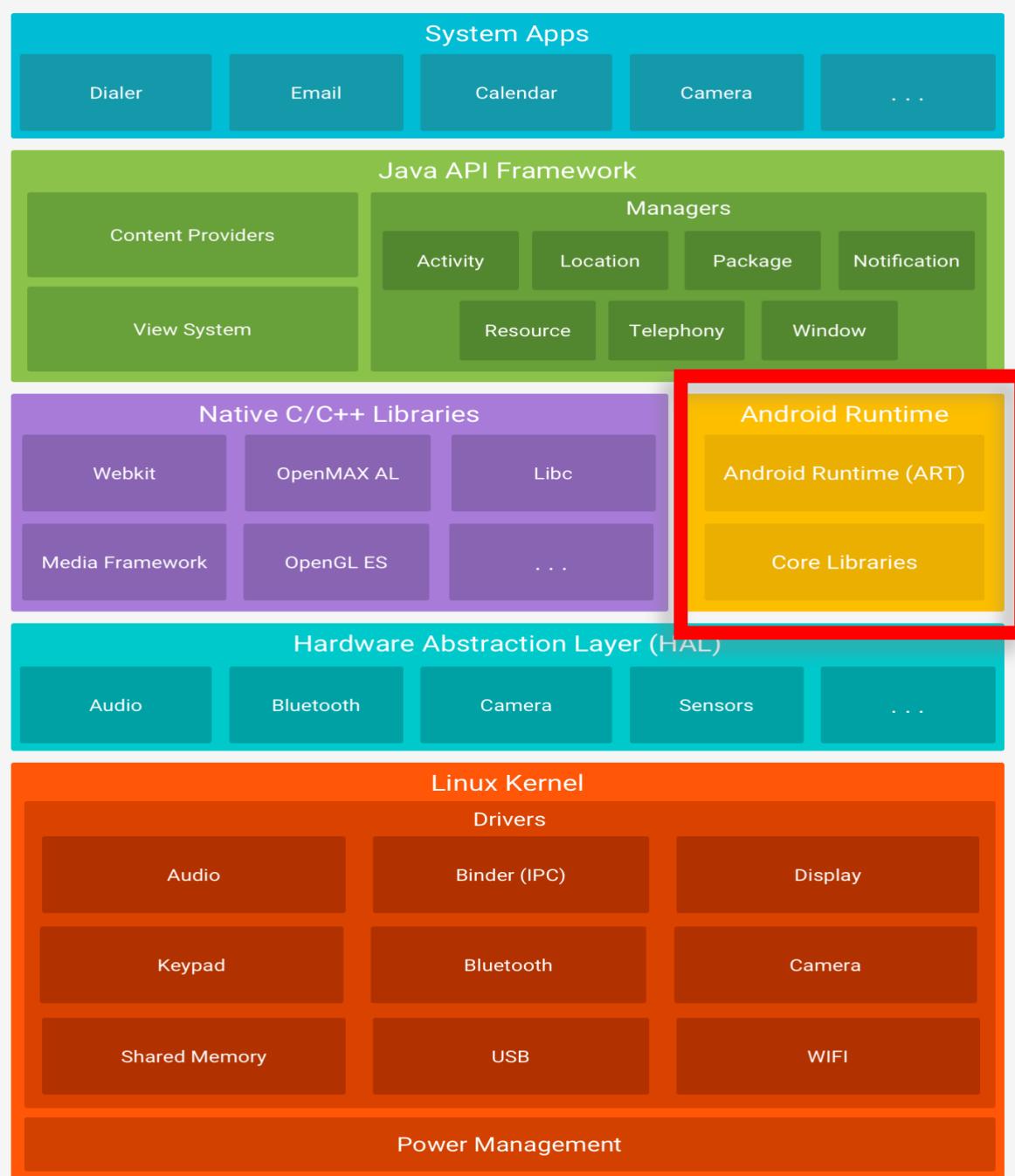


Некоторые библиотеки:

- **Surface Manager** - композитный менеджер окон. Поступающие команды отрисовки собираются в закадровый буфер, где они накапливаются, составляя некую композицию, а потом выводятся на экран. Это позволяет системе создавать интересные бесшовные эффекты, прозрачность окон и плавные переходы.
- **Media Framework** - библиотеки, реализованные на базе PacketVideo OpenCORE. Используются для записи и воспроизведения аудио и видео контента, а также для вывода статических изображений. Поддерживаются форматы: MPEG4, H.264, MP3, AAC, AMR, JPG и PNG.
- **SQLite** - легковесная и производительная реляционная СУБД, используется в Android в качестве основного движка для работы с базами данных. SQLite - предоставляет различные классы, используемые для управления базами данных.
- **3D библиотеки** - используются для высокооптимизированной отрисовки 3D-графики, при возможности используют аппаратное ускорение. Библиотеки реализованы на основе API OpenGL|ES. OpenGL|ES (OpenGL for Embedded Systems) - подмножество графического программного интерфейса OpenGL, адаптированное для работы на встраиваемых системах.
- **FreeType** - библиотека для работы с битовыми картами, для растеризации шрифтов и осуществления операций над ними.
- **LibWebCore** - библиотеки браузерного движка WebKit, используемого также в известных браузерах Google Chrome и Apple Safari.
- **SGL** (Skia Graphics Engine) - открытый движок для работы с 2D-графикой. Графическая библиотека является продуктом Google и часто используется в других программах.
- **SSL** - библиотеки для поддержки одноименного криптографического протокола. SSL - обеспечивает безопасность в Интернете.
- **Libc** - стандартная библиотека языка C, а именно ее BSD реализация, настроенная для работы на устройствах на базе Linux.
- **OpenGL** - это интерфейс Java для API рендеринга 3D-графики OpenGL ES.
- **WebKit** - это движок веб-браузера, используемый для отображения интернет-контента.

...

Среда выполнения Android

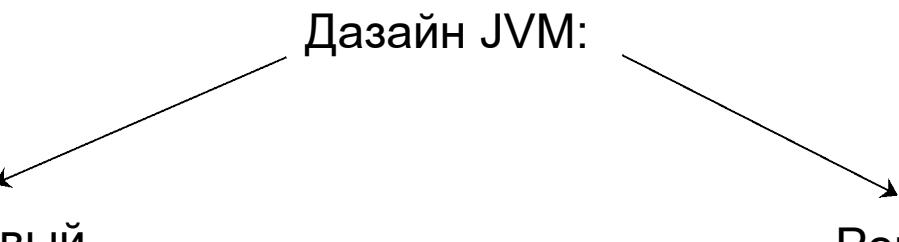
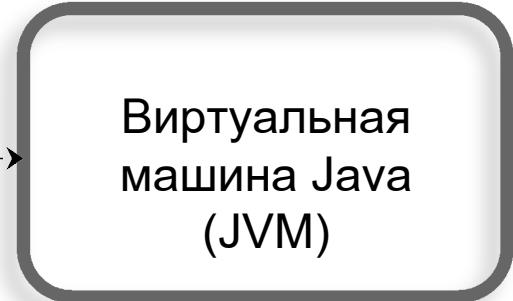
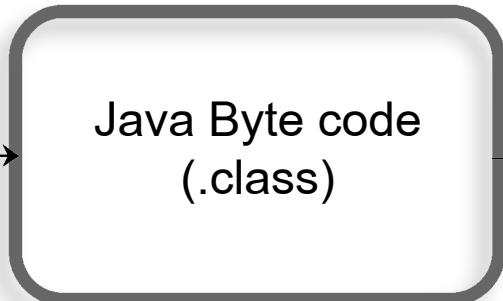
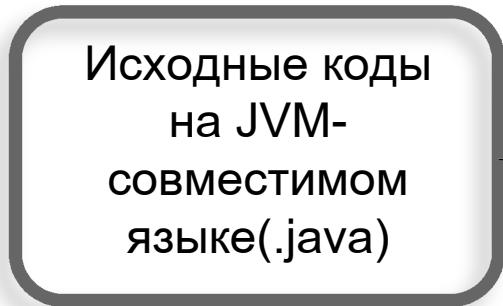


**Среда выполнения Android
(Android Runtime - ART):**
Отвечает за выполнение кода приложений.



JVM

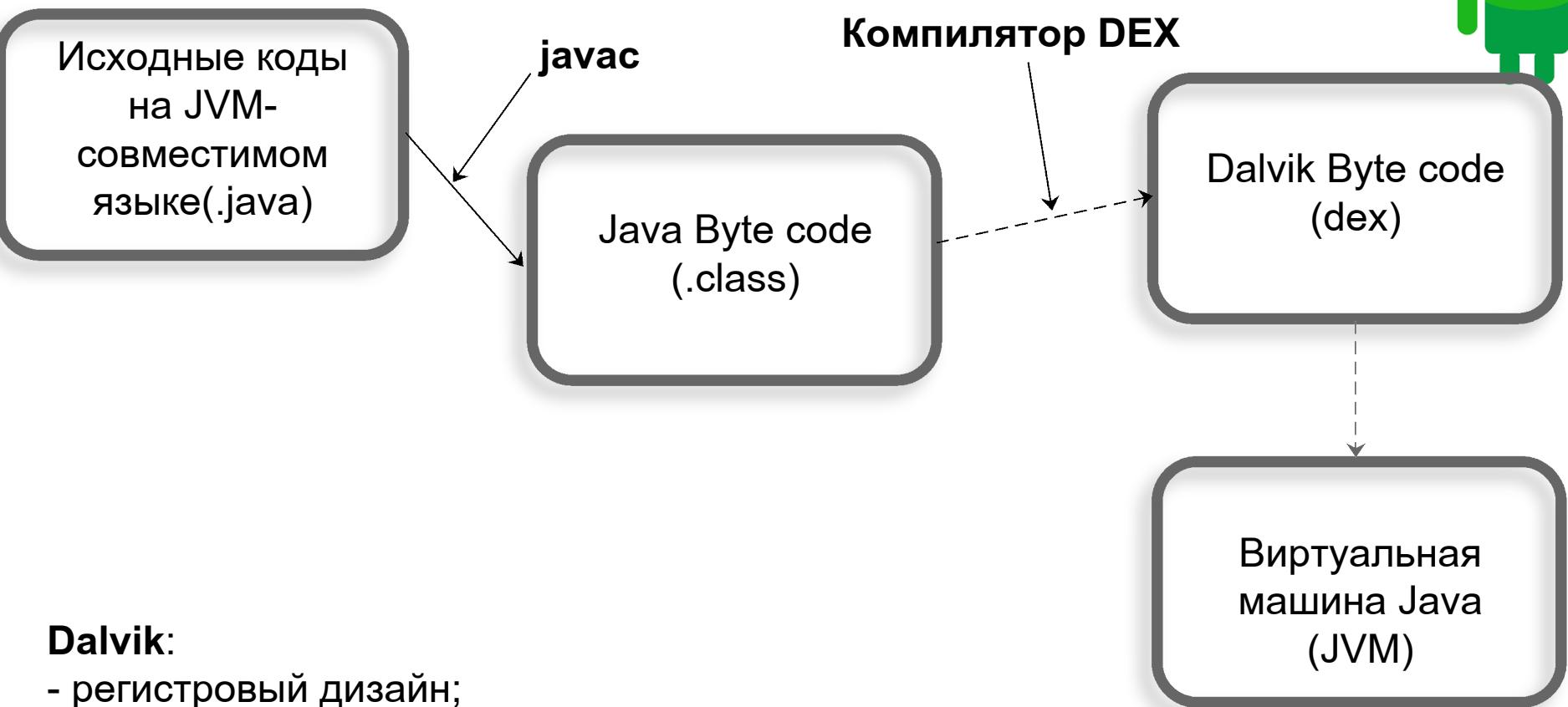
javac



JIT (Just In Time compiler) - для конвертации байткода во время выполнения

AOT (Ahead Of Time Compiler) - для перевода байткода приложений в машинные инструкции на этапе установки.

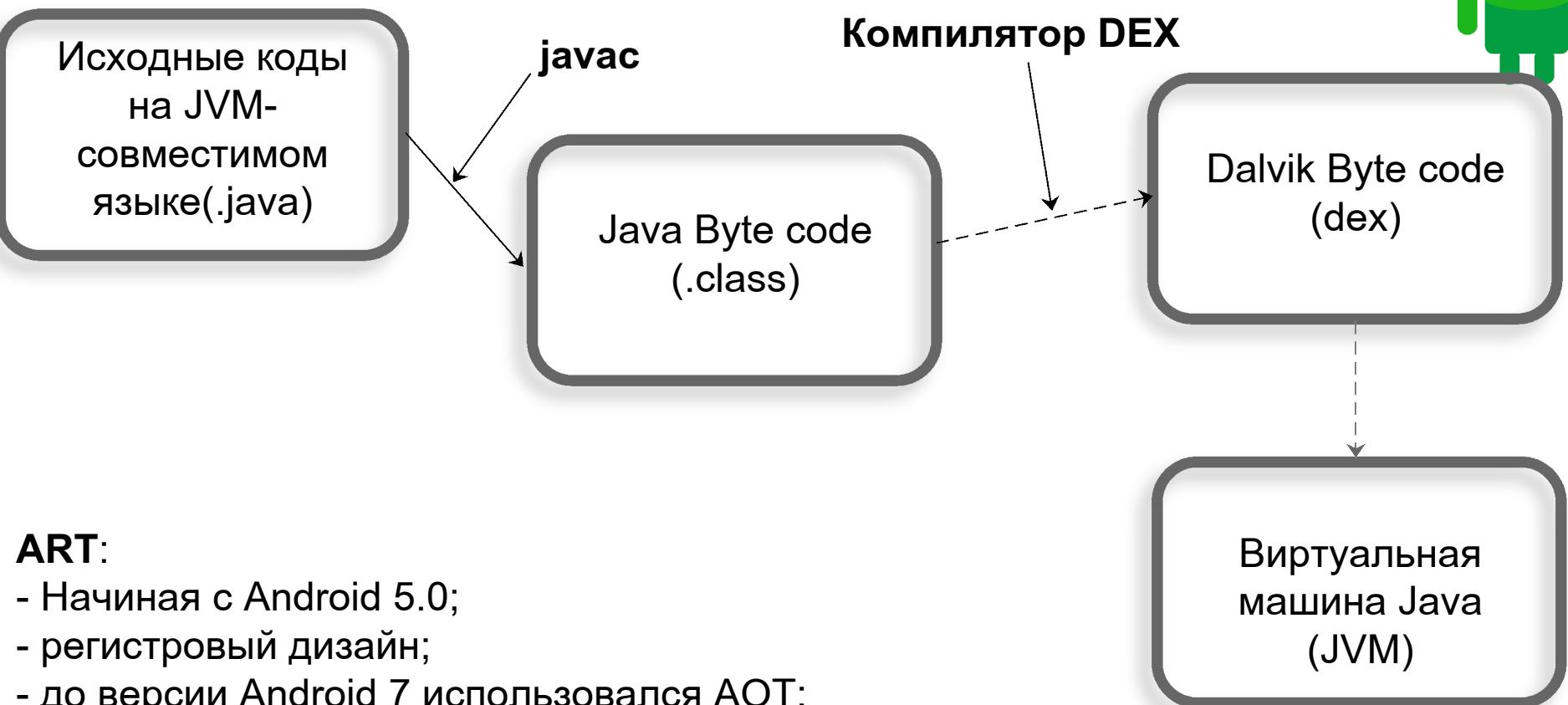
Виртуальная машина Dalvik



Dalvik:

- регистровый дизайн;
- отсутствие JIT (до версии Android 2.2).

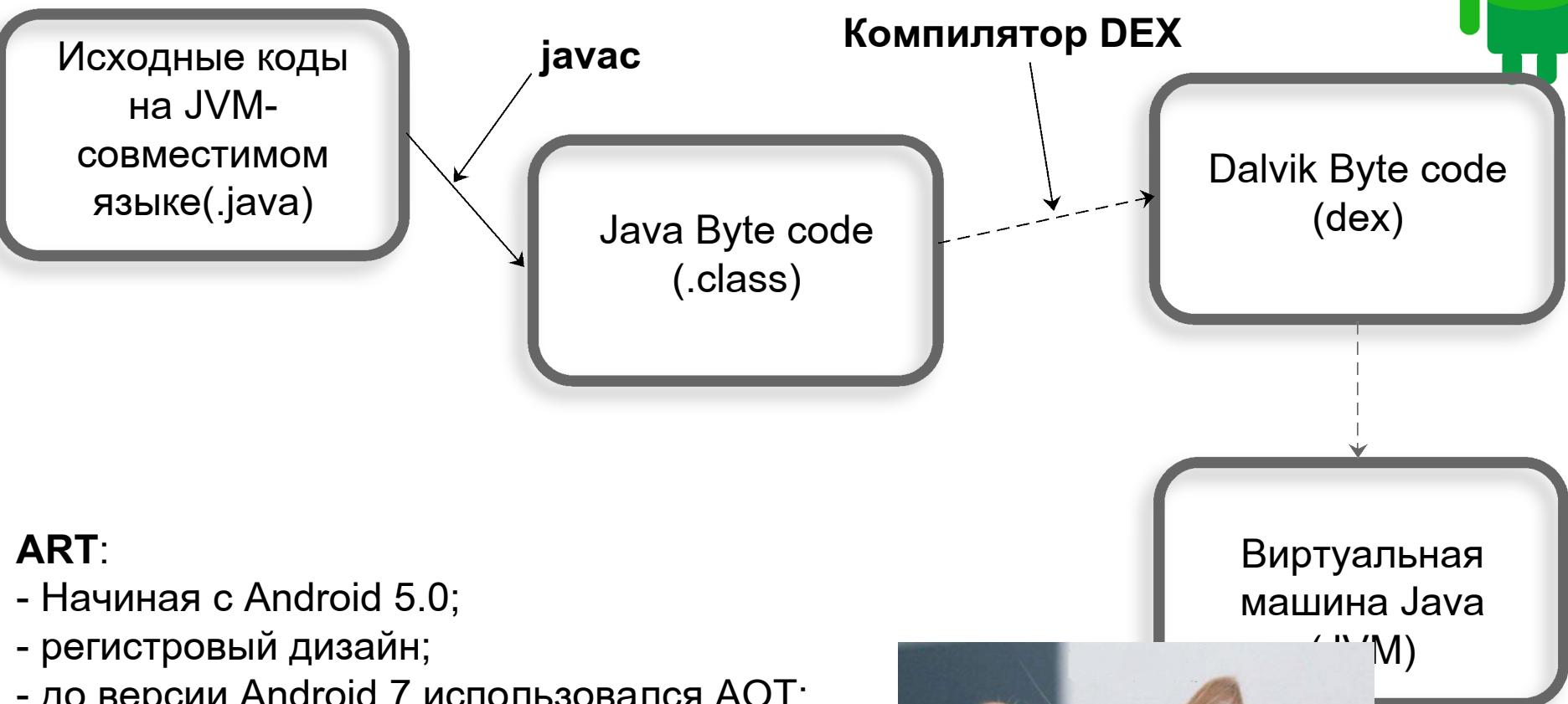
Виртуальная машина ART



ART:

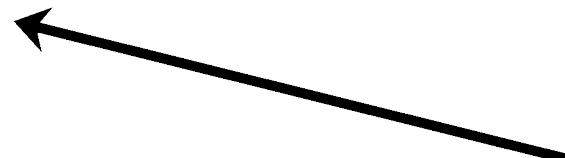
- Начиная с Android 5.0;
- регистровый дизайн;
- до версии Android 7 использовался АОТ;
- с версии Android 7 используется гибридный JIT/АОТ-компилятор.

Виртуальная машина ART



ART:

- Начиная с Android 5.0;
- регистровый дизайн;
- до версии Android 7 использовался АОТ;
- с версии Android 7 используется гибридный JIT/AOT-компилятор.





А в чем преимущества?

Портабельность.

Надежность.

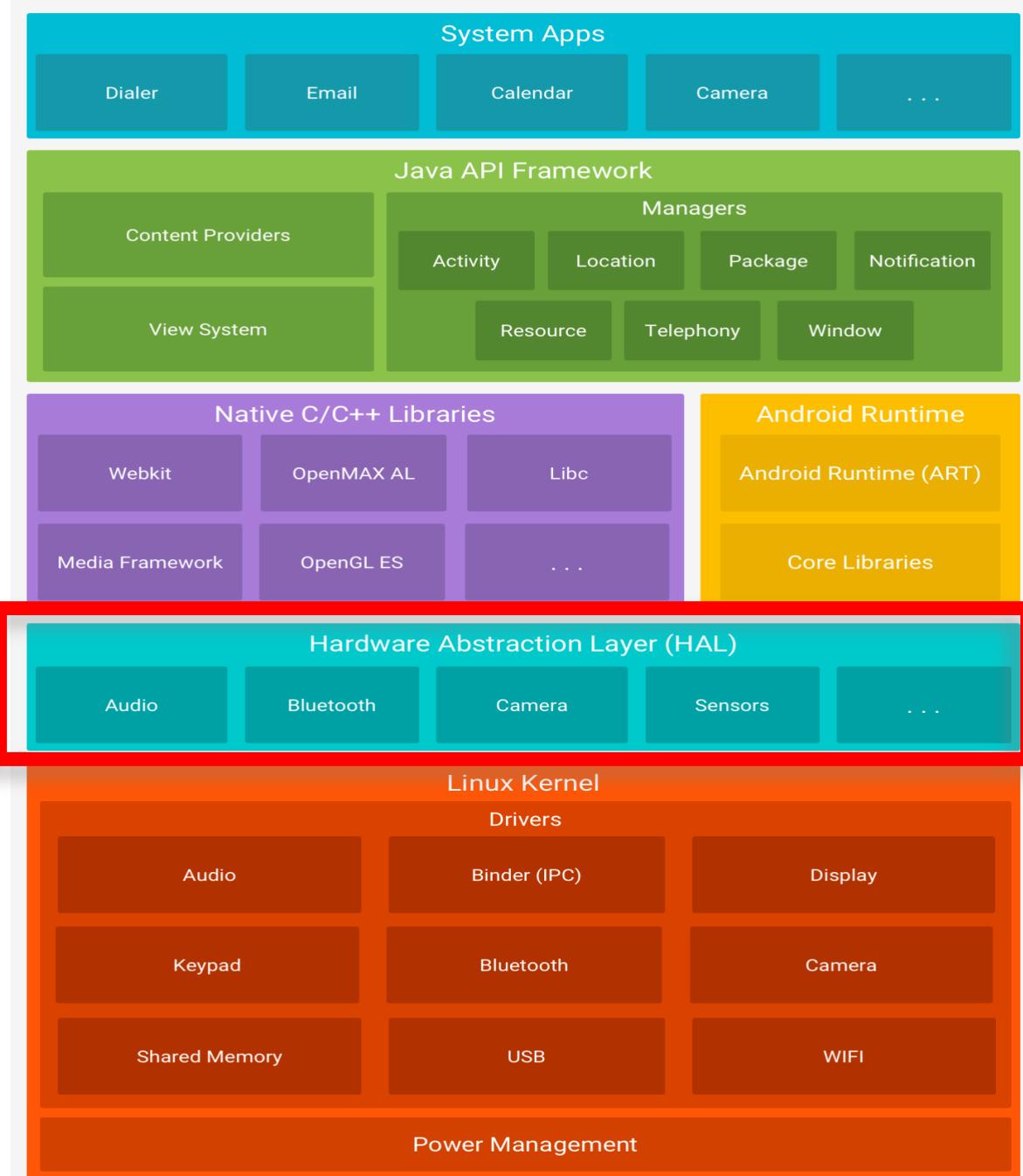
Java — один из самых популярных языков в мире, поэтому у разработчиков для Android изначально был доступ к огромному количеству Java-библиотек.

HAL



Hardware Abstraction Layer (HAL, Слой аппаратных абстракций) — слой абстрагирования, реализованный в программном обеспечении, находящийся между физическим уровнем аппаратного обеспечения и программным обеспечением, запускаемом на этом компьютере.

HAL



HAL предназначен для скрытия различий в аппаратном обеспечении от основной части ядра операционной системы, таким образом, чтобы большая часть кода, работающая в режиме ядра, не нуждалась в изменении при её запуске на системах с различным аппаратным обеспечением.

HAL обеспечивает связь между драйверами и библиотеками.

Состоит он из нескольких библиотечных модулей, каждый из которых реализует интерфейс для определенного аппаратного компонента (Bluetooth, Камера и тд.).

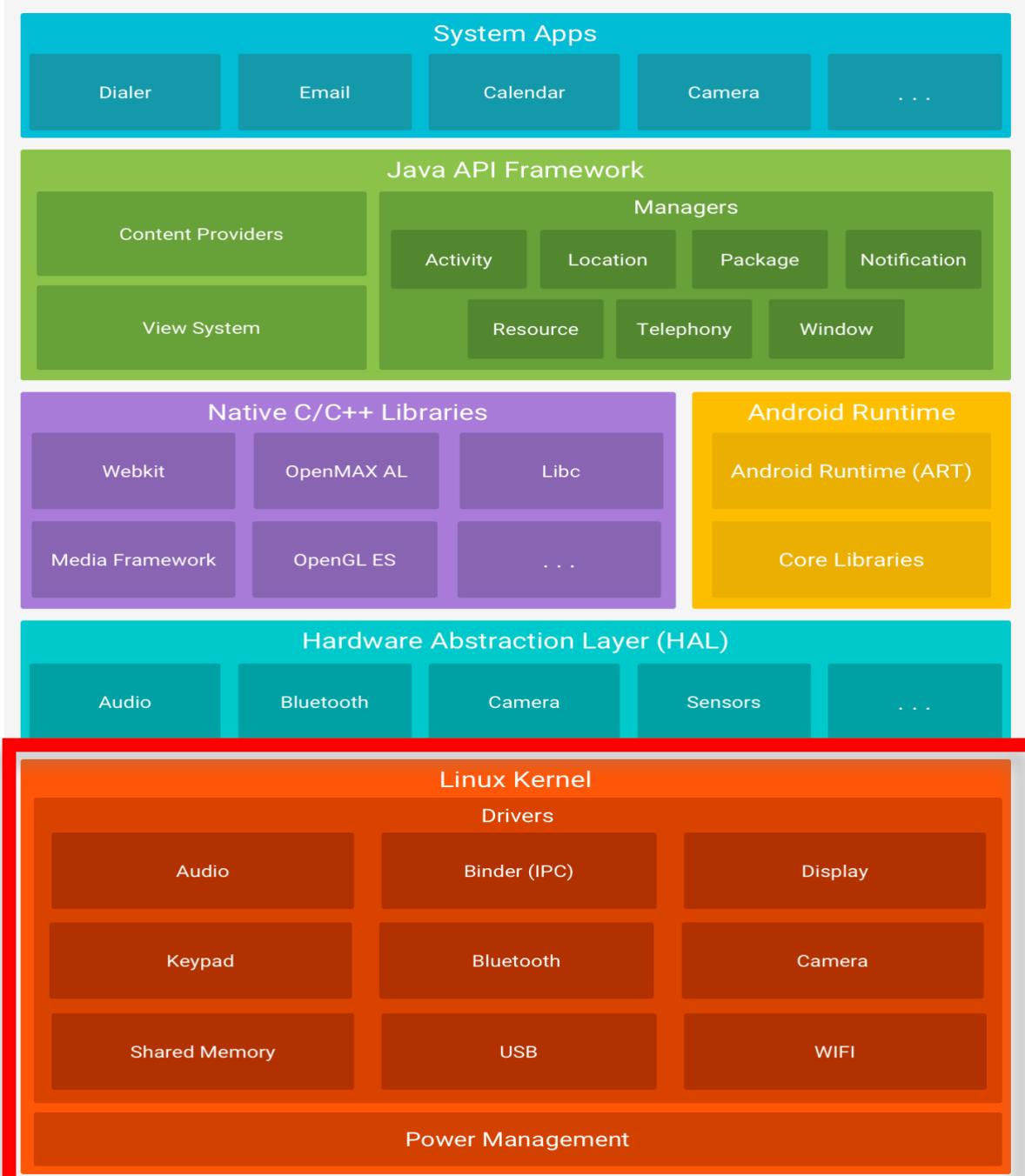
Ядро Linux



Вся система Android построена на ядре Linux с некоторыми архитектурными изменениями.

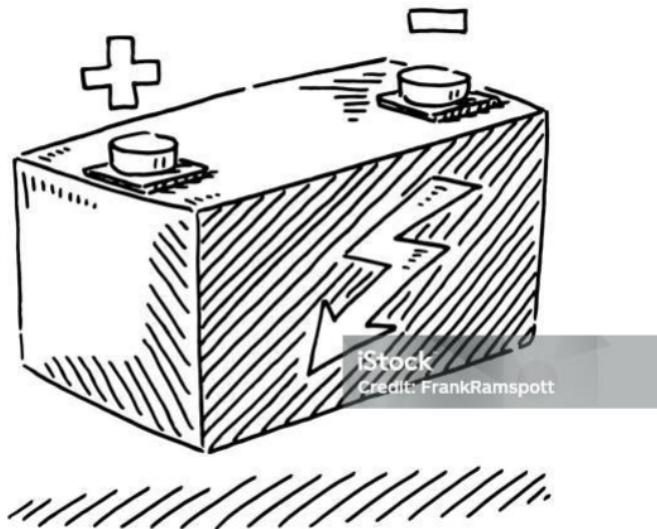
Ядро отвечает за предоставление основных служб, которые ожидают увидеть разработчики: файловая система, потоки и процессы, доступ к сети, интерфейсы для аппаратных устройств и т. д.

Ядро Linux содержит важные аппаратные драйвера, такие как дисплей, аудио, камера, Bluetooth, Wi-Fi и т. д.

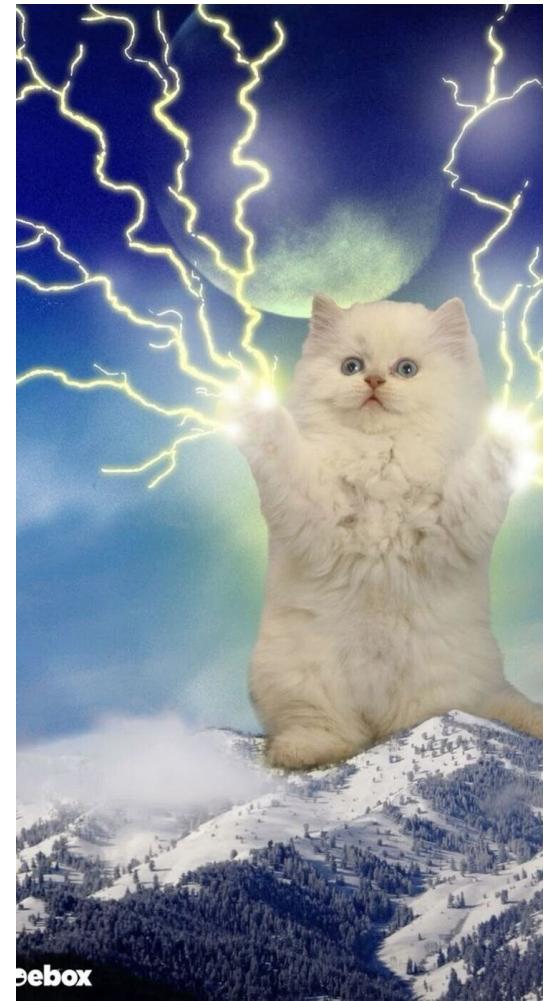




Power Management - это своего рода система управления питанием. Она предоставляет различные средства, с помощью которых приложение может реагировать на режимы питания устройства, а также поддерживать необходимые компоненты устройства активными.



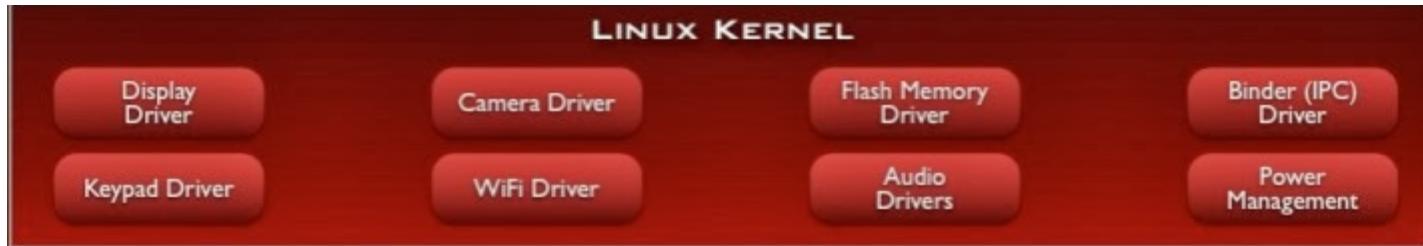
1273848893



eebox



Ядро Linux

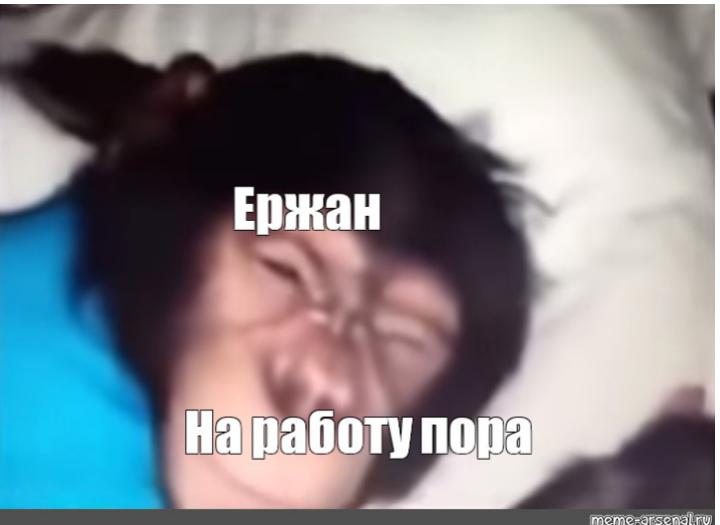


Ядро расширено некоторыми специфичными модулями:

- **Binder** - обеспечивает межпроцессное взаимодействие Inter-Process Communication/Remote Procedure Call (IPC/RPC);
- **Ashmem** - Asynchronous SHared MEMory (драйвер разделяемой памяти);
- **Wakelocks** – механизм поддержания работоспособности процессов в фоновом режиме;
- **Low Memory Killer** - механизм экстренного завершения процессов при нехватке памяти;
- и др.



Когда свободная оперативная память подойдет к концу или ее не хватит для размещения другого приложения система закроет приложения, которые не нужны в данный момент.



Android - многозадачная система.

Важный элемент системы многозадачности — службы (*services*). Это особые компоненты приложений, которые в ранних версиях Android могли работать в фоне при абсолютно любых условиях (со временем возможности служб ограничили).

Службы обращались к системе и запрашивали ресурсы процессора. В терминологии Android такой запрос к системе называется *wakelock*. Для таких запросов и используется Wakelock



Android с самых первых версий использовал песочницы для изоляции приложений.

Каждое приложение запускалось от имени отдельного пользователя Linux и, таким образом, имело доступ только к своему каталогу внутри /data/data.

А как тогда приложениям взаимодействовать друг с другом и системой?





Друг с другом и операционной системой приложения общаются только через IPC-механизм Binder.

Работу Binder обеспечивают драйвер в ядре Linux и Service Manager.

Android спроектирован так, что пользователи и даже программисты не догадываются о существовании механизма Binder.

ПРИМЕР

Если, например, программист хочет скопировать текст в буфер обмена, он просто получает ссылку на объект-сервис и вызывает один из его методов.

Под капотом фреймворк преобразует этот вызов в сообщение Binder и отправляет его в ядро через файл-устройство `/dev/binder`.

Это сообщение перехватывает Service manager, который находит в своем каталоге сервис буфера обмена, проверяет полномочия приложения на отправку ему сообщений и, если оно имеет все необходимые права, передает сообщение ему.

После получения и обработки сообщения сервис буфера обмена отправляет ответ, используя все тот же Binder.



Система оповещения базируется на интентах (intent), специальном механизме, реализованном поверх Binder.

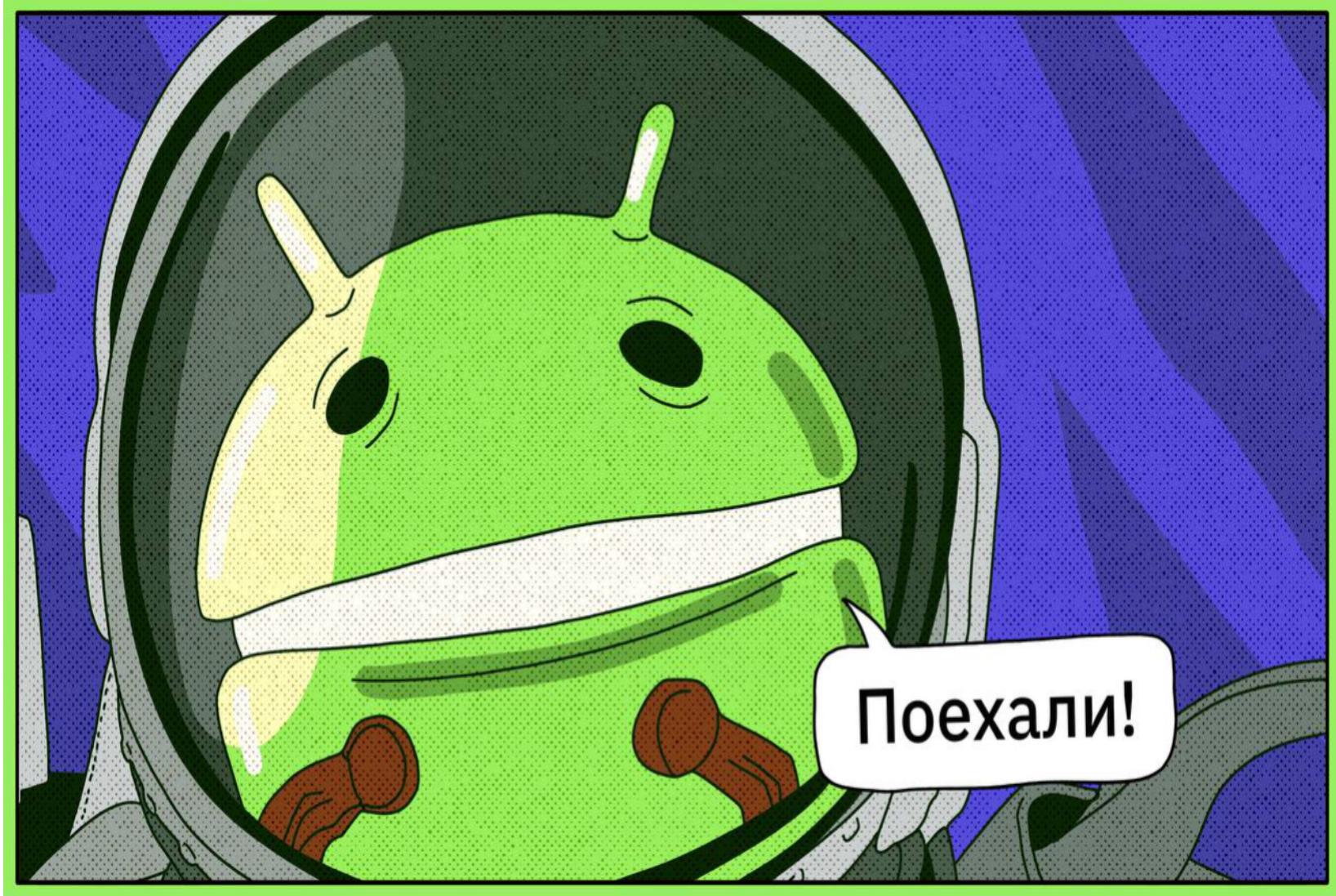
Интенты предназначены для обмена информацией между приложениями (или ОС и приложениями), а также запуска компонентов приложений.



С ними мы поработаем на Лабораторных работах!



А как происходит запуск системы? Сейчас узнаем.





Загрузка ОС Android. Кратко.

Этапы

- Bootrom
- BootLoader
- Ядро
- init()
- Среда выполнения Android (app_process -> Art/Dalvik -> Zygote)
- system_server (Тут разные *Manager и службы)
- Activity Manager
- Запуск «своего» приложения (главный поток приложения(MainThread))

BootRom — это небольшой кусочек защищенной от записи флеш-памяти, встроенный в процессорный чип.

BootLoader выполняет первичный запуск специфичных настроек перед запуском ядра.

Ядро запускает настройку кэша, защищенную память, планировщик задач и загружает драйверы.

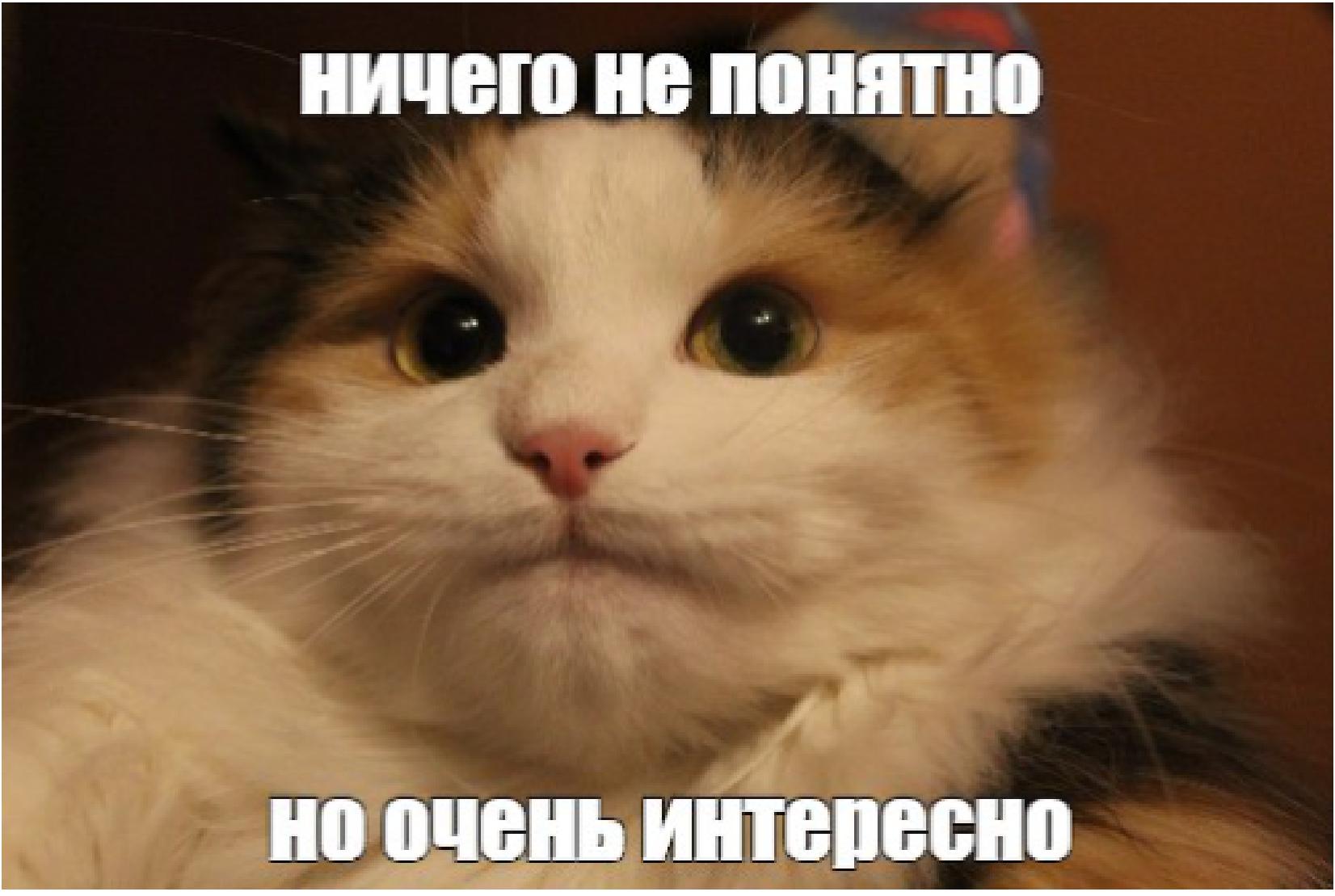
Activity Manager посылает широковещательный интент Intent.CATEGORY_HOME, чтобы найти Launcher-приложение, отвечающее за формирование рабочего стола, и отдает его имя Zygote через сокет.

Процесс init() подключает директории /sys, /dev, /proc и запускает службы(daemon), которые указаны в файле init.rc.

Запуск «своего» приложения

Zygote — ключевой компонент любой Android-системы, который ответственен за инициализацию, старт системных служб, запуск и остановку пользовательских приложений и многие другие задачи.

Загрузка ОС Android.

A close-up photograph of a fluffy white and brown cat's face, looking slightly upwards and to the left. The cat has large, dark, expressive eyes and a small pink nose. Its fur is soft and textured.

ничего не понятно

НО ОЧЕНЬ ИНТЕРЕСНО

Загрузка ОС Android. Boot ROM



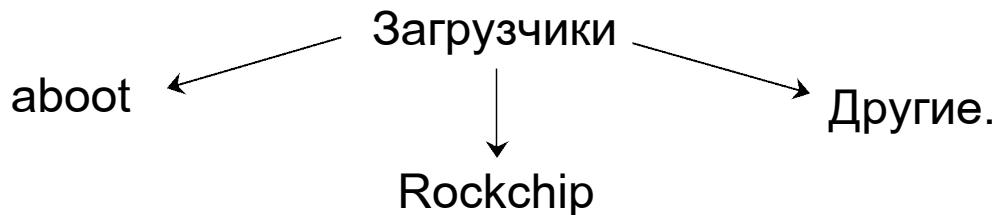
После подачи питания исполняется код в boot ROM, записанного в постоянную, неперезаписываемую память устройства.

Boot ROM извлекает из памяти устройства загрузчик и передает управление ему.

Чаще всего роль загрузчика выполняет загрузчик aboot со встроенной поддержкой протокола fastboot, но производитель мобильного чипа или смартфона/планшета имеет право выбрать любой другой загрузчик на его вкус.

Получив управление, aboot проверяет таблицу разделов и передает управление ядру, прошитому в раздел с именем boot, после чего ядро начинает загрузку либо Android, либо консоли восстановления.

Память в Android-устройствах поделена на семь условно-обязательных разделов.



Загрузка ОС Android. Разделы



boot — содержит ядро и RAM-диск, на устройствах с A/B-разметкой также содержит консоль восстановления (recovery);

recovery — консоль восстановления, состоит из ядра, набора консольных приложений и файла настроек, раздел отсутствует на устройствах с A/B-разметкой;

system — содержит саму ОС Android, системные библиотеки, системные приложения, стандартные рингтоны, обои и т. д.;

cache — раздел предназначен для хранения кешированных данных, например файла обновления, отсутствует на устройствах с A/B-разметкой;

vendor — содержит драйверы и все необходимые прослойки для работы с «железом»;

userdata — содержит настройки, приложения и данные пользователя;

vbmeta — специальный раздел для Android Verified Boot 2.0, содержащий контрольные суммы компонентов системы.



Загрузка ОС Android. Раздел boot

В случае отсутствия «флага загрузки в recovery» в разделе misc загрузчик передает управление коду, расположенному в разделе boot.

Там расположено ядро Linux. оно находится в начале раздела, а сразу за ним следует упакованный с помощью архиваторов cpio и gzip образ RAM-диска.

Изначально в RAM-диске размещался своего рода скелет операционной системы: нужные для начальной загрузки каталоги и файлы инициализации. Ядроподключало RAM-диск в качестве корня файловой системы, а далее уже к нему подключались другие разделы.

При появлении в седьмой версии Android A/B-разметки необходимость в RAM-диске отпала, и инженеры Android решили использовать его для хранения консоли восстановления (вместо использования отдельного раздела recovery).

Загрузка ОС Android. Раздел recovery



В том случае, если «флаг загрузки recovery» в разделе misc установлен или пользователь включил смартфон с зажатой клавишей уменьшения громкости, загрузчик передаст управление коду, ответственному за запуск консоли восстановления.

RAM-диск recovery содержит миниатюрную операционную систему, которая никак не связана с Android. У recovery свой набор приложений (команд) и свой интерфейс, позволяющий пользователю активировать служебные функции.

Загрузка ОС Android. Инициализация



Ядро инициализирует свои подсистемы и драйверы, затем запускает процесс init, с которого начинается инициализация Android.

У init есть конфигурационный файл init.rc, содержащий инструкции о том, что нужно сделать, чтобы проинициализировать систему.

Каждый блок определяет стадию загрузки, или, выражаясь языком разработчиков Android, действие.

В конце блока boot init, скорее всего, встретит команду class start default, которая сообщит, что далее следует запустить все перечисленные в конфиге службы, имеющие отношение к классу default.

Описание служб начинается с директивы service, за ней следует имя службы и команда, которая должна быть выполнена для ее запуска. В отличие от команд, перечисленных в блоках, службы должны работать все время, поэтому на протяжении всей жизни смартфона init будет висеть в фоне и следить за тем, чтобы они работали.

Загрузка ОС Android. Zygote и app_process



На определенном этапе загрузки init запустит службу Zygote.

Zygote, ключевой компонент Android, который ответственен за инициализацию, старт системных служб, запуск и остановку пользовательских приложений и многие другие задачи.

Zygote запускается с помощью бинарного файла /system/bin/app_process

Задача app_process — запустить виртуальную машину Dalvik/ART, код которой располагается в разделяемой библиотеке /system/lib/libandroid_runtime.so, а затем поверх нее запустить Zygote.

Загрузка ОС Android. Zygote и app_process



Когда все это будет сделано и Zygote получит управление, она начинает формирование среды исполнения Java-приложений с помощью загрузки всех Java-классов фреймворка, запускает system server, который включает в себя большинство высокоуровневых (написанных на Java) системных сервисов: Window Manager, Status Bar, Package Manager, **Activity Manager** и и другие.

После этого Zygote открывает сокет /dev/socket/zygote и уходит в сон, ожидая данные.

Запущенный ранее Activity Manager находит приложение, реагирующее на интент intent.category home, и отдает его имя Zygote через сокет.

Zygote делает fork (форкается) и запускает приложение поверх виртуальной машины. На экране появляется Рабочий стол, найденный Activity Manager и запущенный Zygote, и статусная строка, запущенная system server в рамках службы Status Bar.



Хорошо, а как запустить приложение?

ВСЕ ХОРОШО



ПРОСТО НЕМНОГО НЕПОНЯТНО

Загрузка ОС Android. Запуск приложения

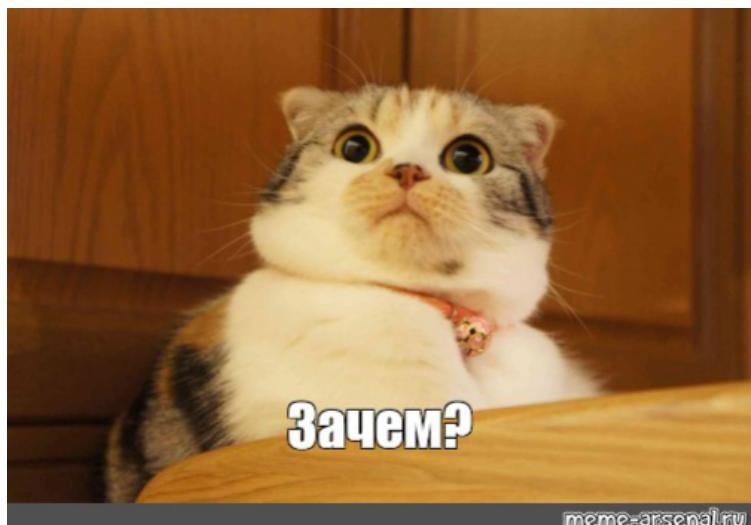


После тапа по любому значку Рабочий стол пошлет интент с именем этого приложения, который примет Activity Manager и передаст команду на старт приложения демону Zygote.

Zygote сделает fork и запускает приложение поверх виртуальной машины.

И так для любого приложения!!!

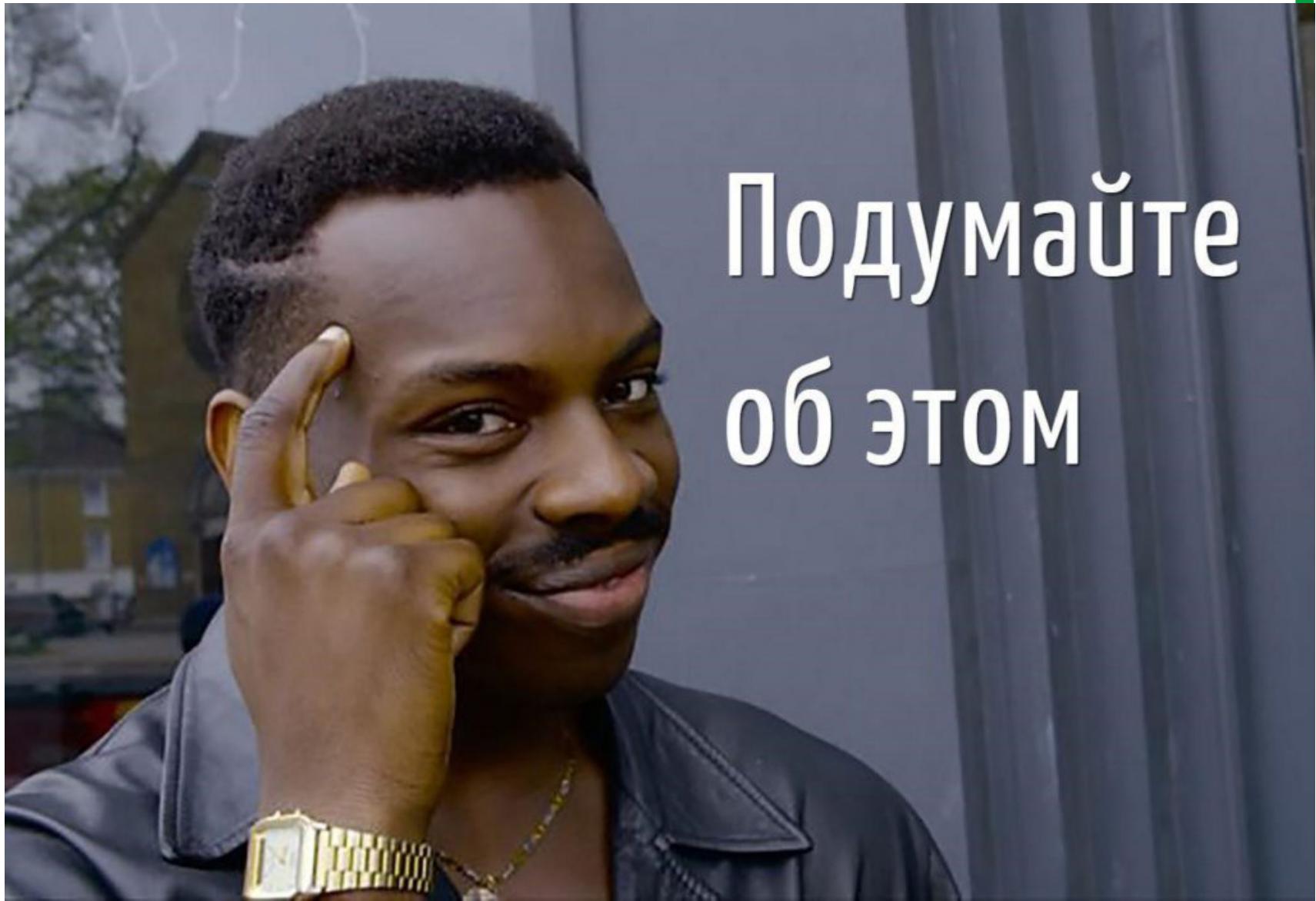
А зачем так делать?



Загрузка ОС Android. Запуск приложения

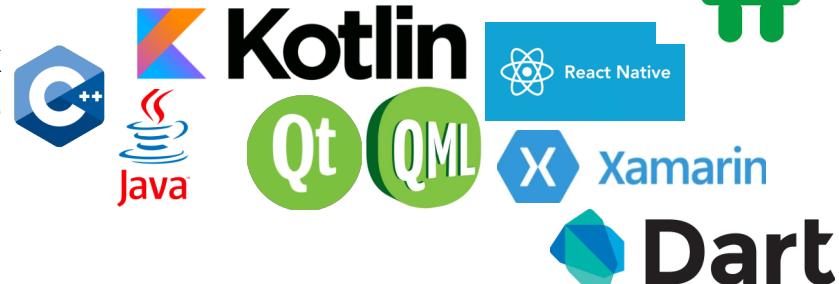


Подумайте
об этом



Android приложение

Приложения для Android можно писать на языках Kotlin, Java и C ++ (считаются основными), а также многих других языках и фреймворках.



Для создания мобильных приложений под ОС Android программисту необходимы пакет JDK, предоставляющий базовые сервисы Java, а также специализированные инструментальные средства – интегрированная среда разработки (IDE) и инструментальный программный пакет Android SDK (Software Development Kit).

Мобильные Java-программы компилируются в нестандартный байт-код, исполняемый виртуальной машиной (ранее – Dalvik, а начиная с ОС Android 5.0 –ART). Для компиляции используется пакет Android SDK.

На самом деле приложение Android - набор файлов в заранее определенных каталогах. При построении приложения все эти файлы собираются воедино, и вы получаете приложение, которое можно запустить на вашем устройстве.

Каждое приложение выполняется в собственном процессе, используя исполнительную среду Android (ART).

Дистрибуция Android приложений

APK



Для установки приложения на устройствах с ОС Android создается файл с расширением *.apk (Android package), который содержит исполняемые файлы, а также вспомогательные компоненты, например, файлы с данными и файлы ресурсов.

Android package, который представляет собой архивный файл с расширением **.apk**, содержит содержимое приложения Android, которое требуется во время выполнения (байт-код приложения, библиотеки и ресурсы), и это файл, который устройства под управлением Android используют для установки приложения.

Установка приложения на устройстве осуществляется установкой его пакета APK.



Дистрибуция Android приложений



AAB

Пакет Android App Bundle, который представляет собой архивный файл с расширением .aab, содержит содержимое проекта приложения Android, включая некоторые дополнительные метаданные, которые не требуются во время выполнения.

AAB - это формат для публикации в маркете, который нельзя установить на устройствах Android, он откладывает создание APK и подписание на более поздний этап.

Например, при распространении вашего приложения через Google Play серверы Google Play генерируют оптимизированные APK-файлы, которые содержат только ресурсы и код, которые требуются конкретному устройству, запрашивающему установку приложения.





Структура проекта Android приложения



Каждый проект может содержать несколько модулей.

Каждый модуль является отдельным Android-приложением.

Приложение может включать:

1. **Активности** (activities) – компоненты, которые содержат пользовательский интерфейс приложения т.е. это экран, который видит и с которым взаимодействует пользователь.

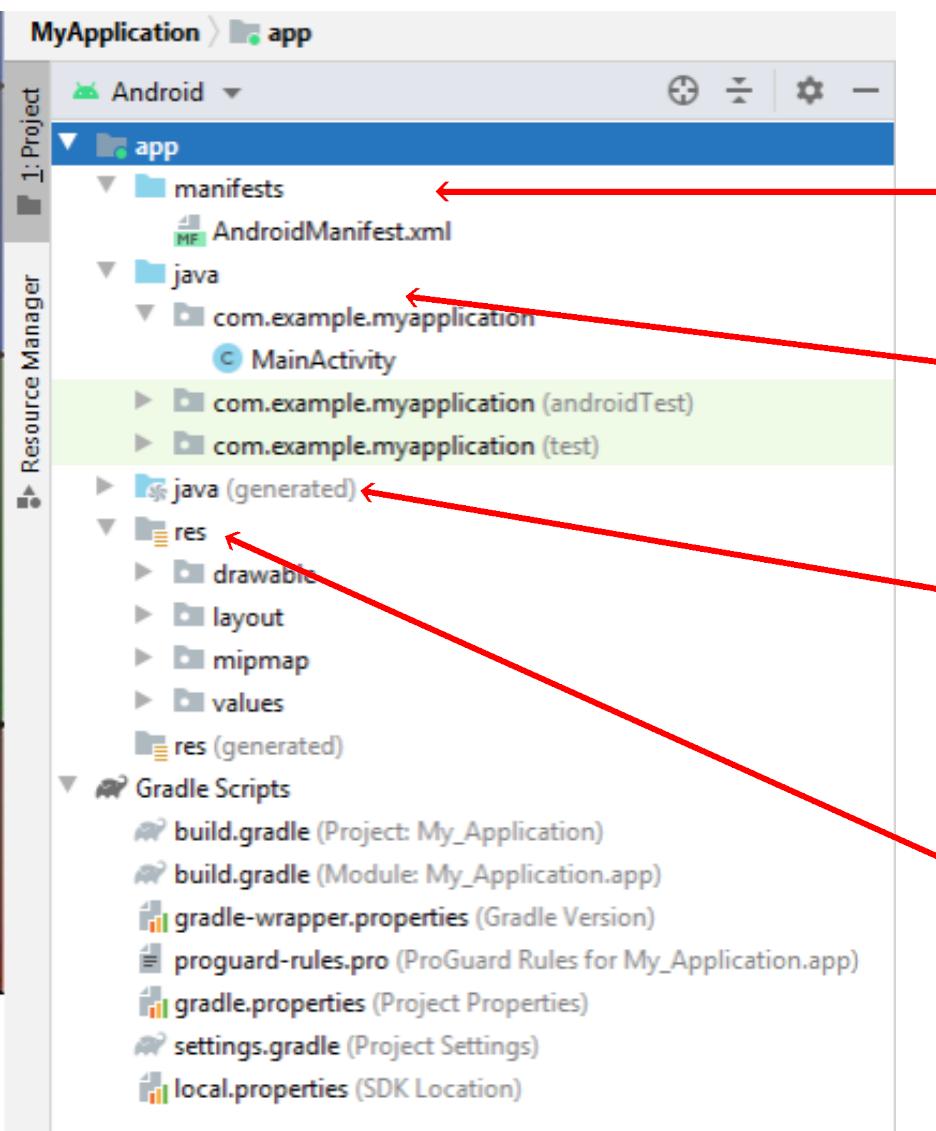
2. **Службы** (services) – компоненты, работающие в фоне, не имеющие пользовательского интерфейса.

3. **Широковещательные приемники** (Broadcast receivers) – это компоненты, для получения информации о внешних событиях. Настраиваются на определенные рассылки и ждут сообщения об отслеживаемых изменениях (например, о текущем заряде батареи или наличии интернета).

4. **Контент-провайдеры** – компоненты, позволяющие делиться данными приложения со всеми желающими. Через них другие приложения могут запрашивать или изменять данные в хранилищах контент-провайдеров. Например, контент-провайдер ОС Android по запросам от других приложений предоставляет им информацию о списке контактов пользователя.



Структура проекта Android приложения



В структуре проекта выделяются:

- раздел *manifest* – содержит файл с основными параметрами приложения и его структурой для учета в ОС.
- раздел *java* – содержит исходные коды приложения, который включает основной код, юнит тесты (*test*) и инструментальные тестов (*androidTest*).
- раздел *javagenerated* содержит java файлы, которые генерируются во время сборки приложения. Они не являются частью исходного кода проекта, но используются в приложении.
- раздел *res* содержит ресурсы, не являющиеся исходным кодом – изображения, тексты, видео, верстка и т.п.



Структура проекта Android приложения

The screenshot shows the 'Project' tab in the Android Studio interface. The 'app' module is selected. The structure is as follows:

- app**:
 - manifests**: Contains `AndroidManifest.xml`.
 - java**: Contains a package `com.example.myapplication` with a file `MainActivity`, and two test-related files: `com.example.myapplication (androidTest)` and `com.example.myapplication (test)`. This entire section is highlighted with a light green background.
 - java (generated)**: A folder containing generated Java files.
 - res**: Contains subfolders `drawable`, `layout`, `mipmap`, `values`, and `res (generated)`.
- Gradle Scripts**: A section containing several Gradle configuration files:
 - `build.gradle (Project: My_Application)`
 - `build.gradle (Module: My_Application.app)`
 - `gradle-wrapper.properties (Gradle Version)`
 - `proguard-rules.pro (ProGuard Rules for My_Application.app)`
 - `gradle.properties (Project Properties)`
 - `settings.gradle (Project Settings)`
 - `local.properties (SDK Location)`This section is highlighted with a red border.

Также в структуре проекта отдельно существует раздел *Gradle Scripts*.

Раздел содержит файлы системы сборки проекта Gradle. Это система автоматической сборки, при помощи которой собираются Android приложения.

Практический интерес тут представляют файлы **build.gradle**, который относится к модулю (рядом с ним в скобках будет написано название модуля), и файл **build.gradle**, который относится к проекту (рядом с ним в скобках будет написано название проекта).



Манифест

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>

<uses-permission />
<permission />
<permission-tree />
<permission-group />
<instrumentation />
<uses-sdk />
<uses-configuration />
<uses-feature />
<supports-screens />
<compatible-screens />
<supports-gl-texture />

<application>

    <activity>
        <intent-filter>
            <action />
            <category />
            <data />
        </intent-filter>
        <meta-data />
    </activity>

    <activity-alias>
        <intent-filter> . . . </intent-filter>
        <meta-data />
    </activity-alias>

    <service>
        <intent-filter> . . . </intent-filter>
        <meta-data/>
    </service>

    <receiver>
        <intent-filter> . . . </intent-filter>
        <meta-data />
    </receiver>

    <provider>
        <grant-uri-permission />
        <meta-data />
        <path-permission />
    </provider>

    <uses-library />
</application>
</manifest>
```

Файл манифеста **AndroidManifest.xml** является главным конфигурационным файлом Android-приложения и предоставляет основную информацию о программе системе.

Приложение обязательно должно иметь свой файл **AndroidManifest.xml**.

Корневым является элемент **<manifest>**. В нем в атрибуте **package** задается имя пакета. Он также должен включать атрибут **xmlns:android**, который определяет пространство имен. И может содержать атрибут **android:versionCode** – версию приложения.

Элементы **<uses-permission>** используются для декларации разрешений на доступ к функциям устройства. Например, разрешение **android.permission.READ_CONTACTS** запрашивается для доступа к контактам устройства, а разрешение **android.permission.INTERNET** – для загрузки содержимого из сети Интернет.

Если приложения пытается получить доступ к функциям, требующим разрешений и не перечисленных в файле манифеста, то генерируется исключение. При установке пользователь должен подтвердить доступ приложения к запрашиваемым функциям.



Манифест

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>
        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>
        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>
        <uses-library />
    </application>
</manifest>
```

Элемент **<application>** один из основных элементов манифеста, содержащий описание компонентов приложения.

Он описывает приложение в целом.

Атрибуты `android:label` и `android:icon` этого элемента определяют имя приложения и его иконку, отображаемые в списке приложений на устройстве.

Внутри элемента **<application>** размещаются определения компонентов приложения.

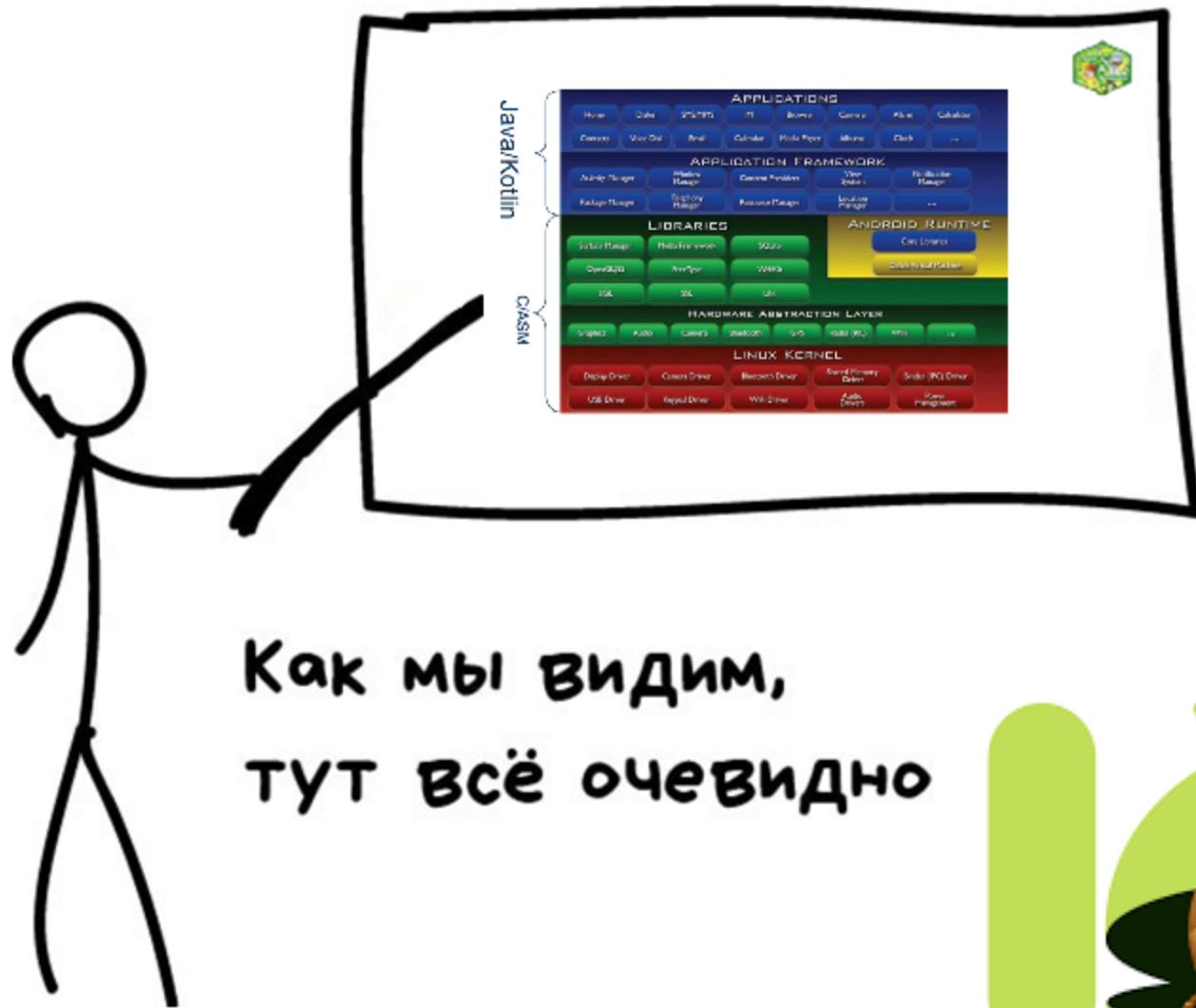
<activity> – активность,

<receiver> – Broadcast Receiver,

<provider> – контент провайдер,

<service> – служба.

Элемент **<uses-library>** определяет библиотеки, с которыми должно быть собрано приложение. Этот элемент указывает системе на необходимость включения кода библиотеки в загрузчик классов для пакета приложения. Если приложение использует пакеты от сторонних разработчиков, необходимо сделать явное связывание с этими библиотеками и манифест обязательно должен содержать отдельный элемент **<uses-library>**.



Как мы видим,
тут всё очевидно

