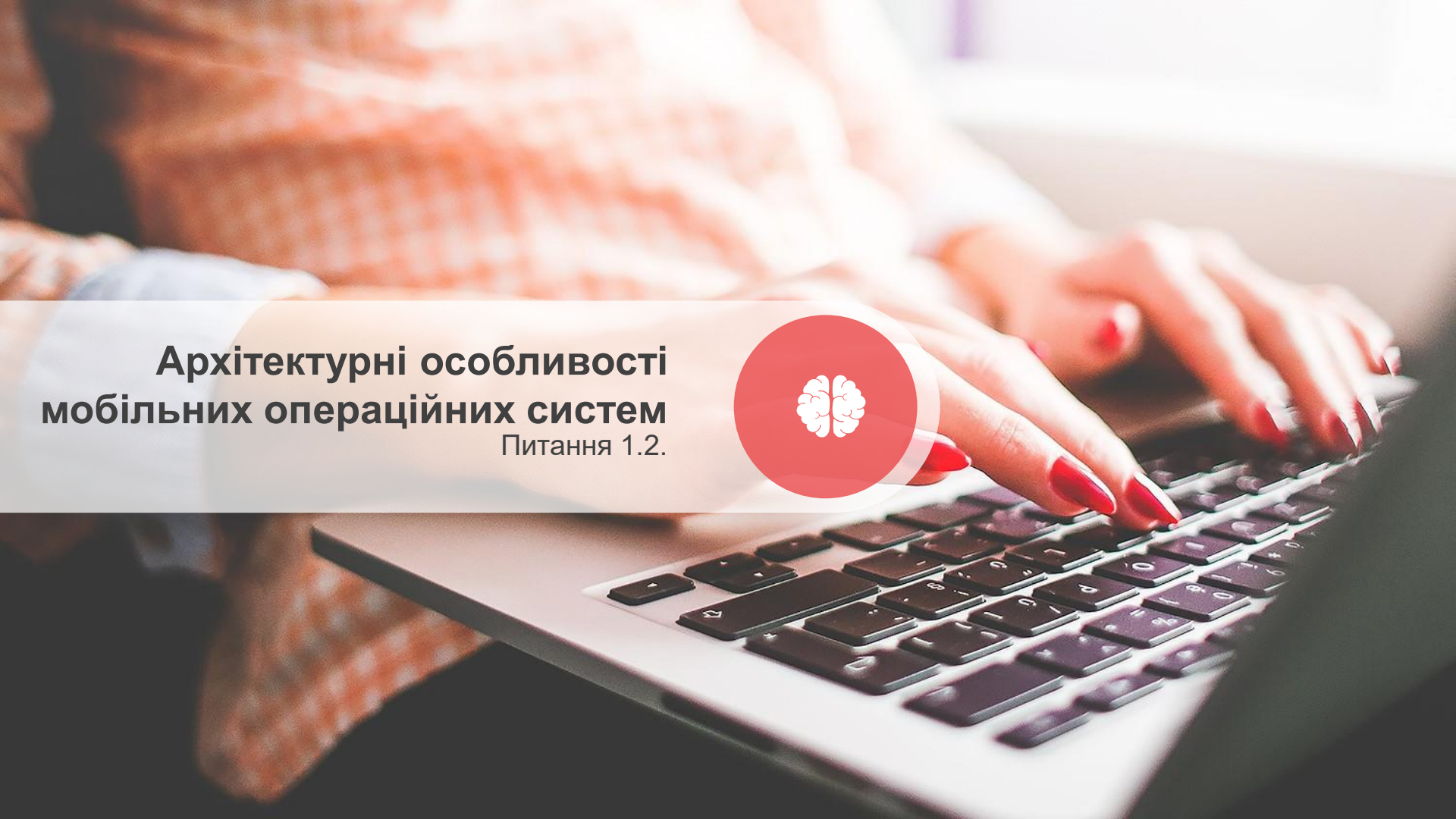
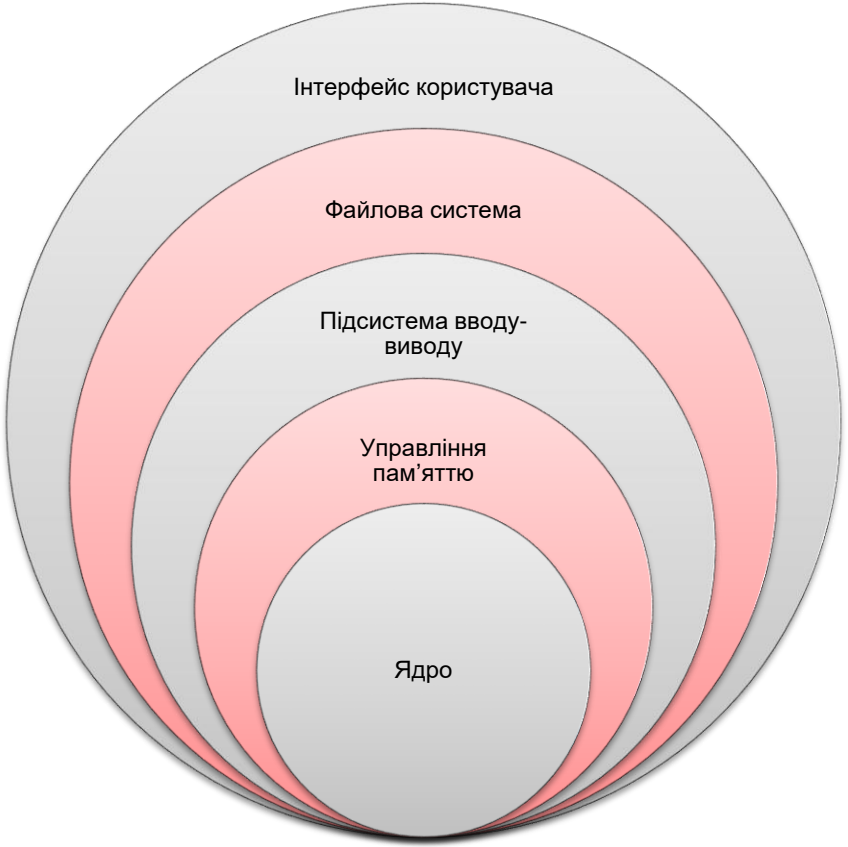
Архітектурні особливості мобільних операційних систем

Питання 1.2.



Шарувата архітектура ОС

Приклад Mac OSX



COCOA Layer
Application user Interface.
Responds to User Event.
Manages App Behavior

Media Layer
Plays, records , editing audiovisual media.
Rendering 2D and 3D graphics

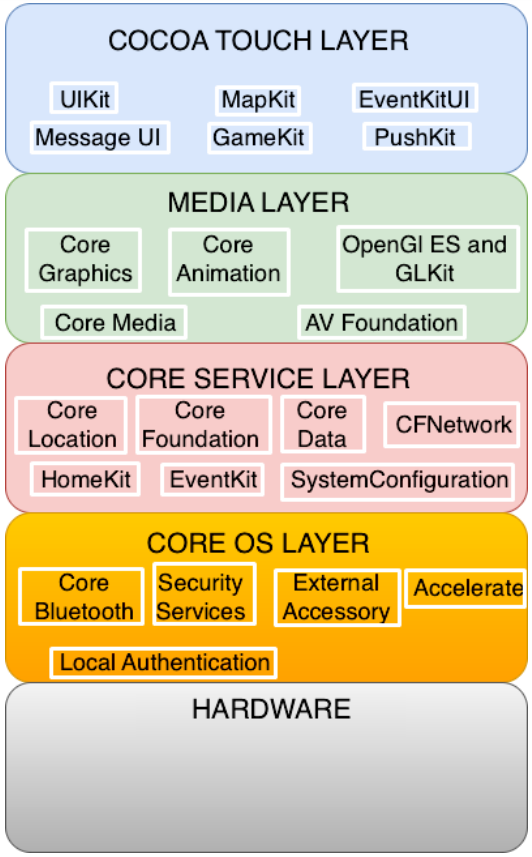
Core Services Layer
Fundamental services for low level network communication.
Automatic Reference Counting, Data Formatting and String Manipulation

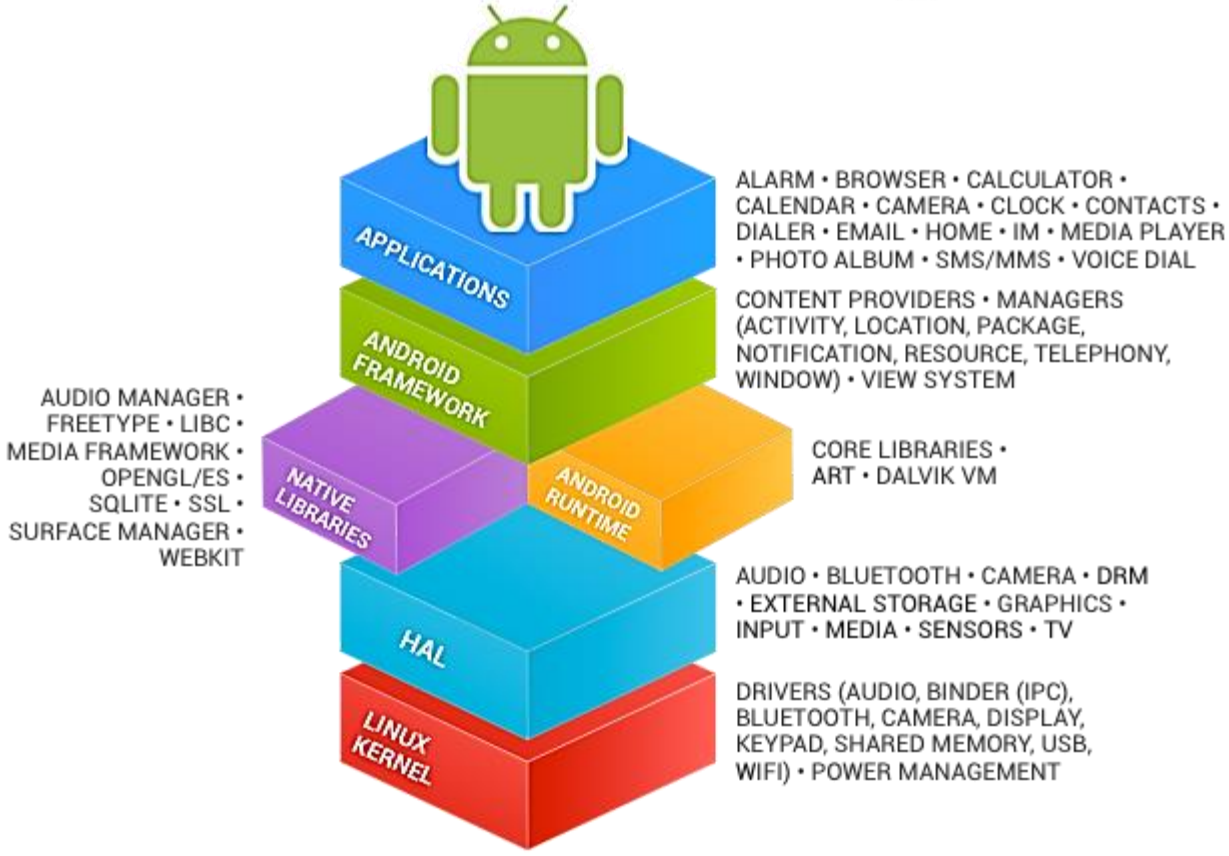
Core OS Layer
Related to hardware and networking,
Interfaces for running high-performance tasks on CPU or GPU

Kernel & Driver Layer
Device drivers and BSD Libraries, low level components.
Support for file system security, interprocess communication, device drivers etc

Архітектура мобільних ОС

Загальне порівняння архітектур iOS та Android



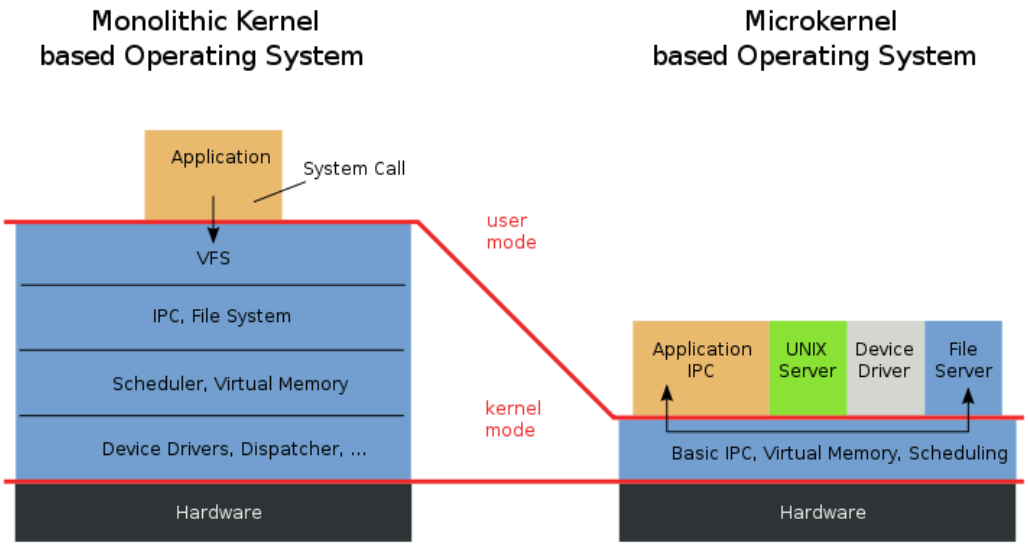


Контекст

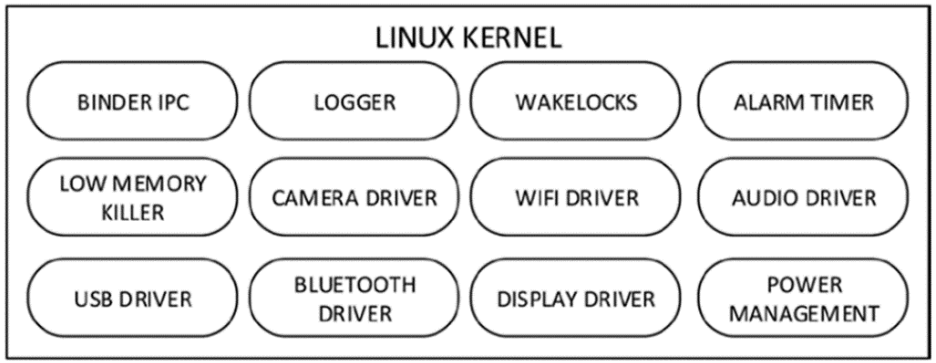
Теми для самоосвіти та самостійного опрацювання

4. ***Монолітні та мікроядра.*** Наступна ОС від Google – Fuchsia – побудована на власному ***мікроядрі*** Zircon.

Тема доповіді: Операційна система Google Fuchsia: архітектура, можливості, перспективи



Linux-ядро в ОС Android

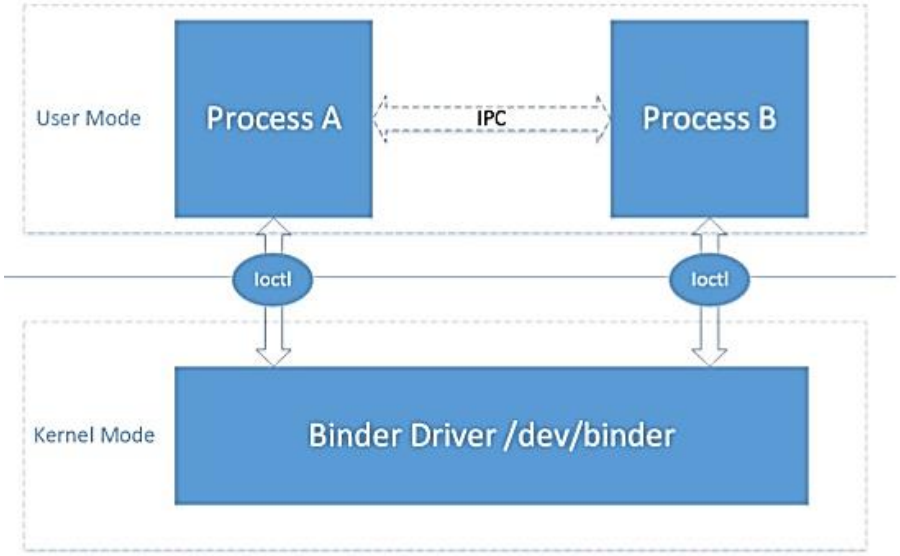


Враховуючи, що Android націлений на мобільні пристрої, відповідна версія ядра пройшла через деякі архітектурні зміни

- Binder
- Logger
- Wake Locks
- Alarm Timer
- Low Memory Killer

Механізм Android Binder

Міжпроцесна взаємодія



http://codetheory.in/an-overview-of-android-binder-framework/

Android покладається на власну реалізацію – Binder – центральний канал комунікації.

Реалізовано в якості низькорівневої служби в Linux-ядрі Android, пряма взаємодія розробників з ним не передбачена.

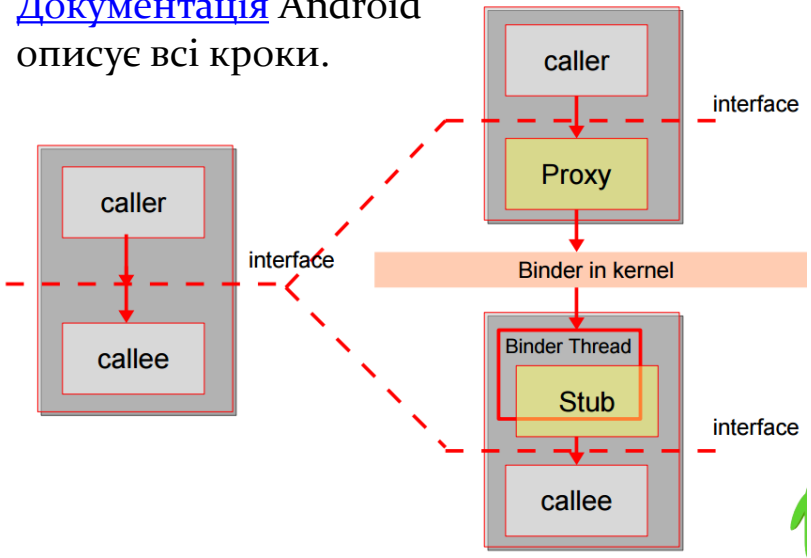
Більшість API операційної системи використовують Binder для взаємодії з платформою.

Для використання Binder з метою взаємодії з іншими застосунками в системі, Android SDK постачає Android Interface Definition Language (AIDL).

AIDL дозволяє розробнику додатка визначати інтерфейс, який будуть використовувати для взаємодії один додаток з іншим додатком або сервісом.

Механізм Android Binder

[Документація](#) Android описує всі кроки.



AIDL забезпечує можливість декомпонувати (маршалізувати) об'єкти, що передаються, в примітиви, які зрозумілі Binder.

AIDL можна назвати контрактом на взаємодію. Він визначає погоджений клієнтом та службою набір методів

Код інтерфейсу AIDL записується в .aidl-файл.

Після запису інструменти Android SDK генерують Java-код (в окремому файлі) та зберігають його разом з проектом – додаткам мають бути доступні методи для взаємодії.

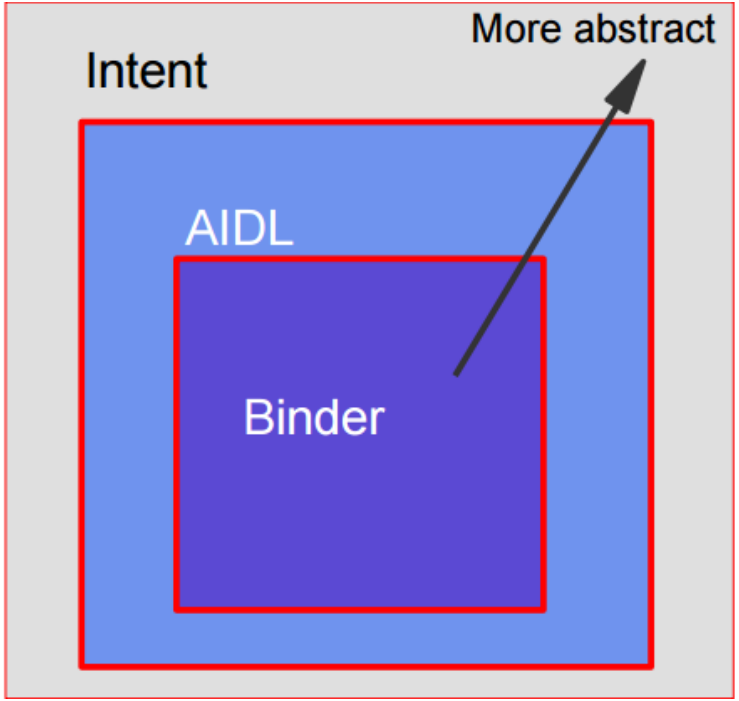
Для клієнт-серверної взаємодії цей файл має бути як у клієнта, так і в сервера.

У згенерованому aidl-файлі є головні класи: Proxy (для клієнта) та Stub (для сервера/служби)

Вони виконують надсилання та обробку транзакцій, а також маршалінг та демаршалінг даних.

Тому насправді client proxy та server stub управляють транзакціями (RPC-викликами)

Рівні абстракції міжпроцесної взаємодії



Інтенти мають найвищий рівень абстракції

Міжпроцесний виклик методу

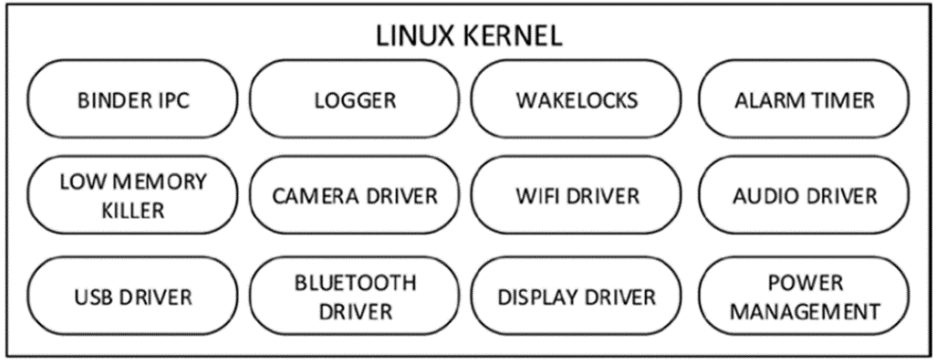
- **AIDL**: Android Interface Definition Language

binder: драйвер ядра (kernel driver)

ashmem: спільна пам’ять (shared memory)

Logger

Ведення логів є базовим механізмом вирішення проблем



Оскільки мобільні пристрої сильно залежать від свого оточення, ізольованих логів додатків недостатньо для вирішення складних проблем.

Важливо комбінувати логи від конкретного застосунку та системи в цілому

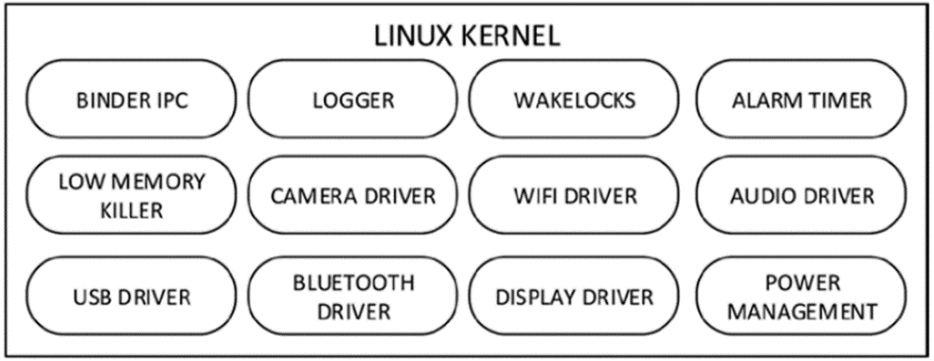
Ще ускладнюється тим, що на мобільних платформах розробка програмного забезпечення та його робота відбуваються на двох різних машинах.

Платформа Android пропонує централізований сервіс логування.

Він допомагає збирати логи як від платформи Android, так і додатків, які працюють поверх неї.

Wake Locks

Надають можливість застосункам забороняти пристрою перехід у режим сну



Механізм суттєво пришвидшує розрядку батареї

Використовувати Wake locks потрібно обережно, і відкликати їх якомога швидше після завершення необхідної операції

Перехід негативно позначається на працюючих у цей момент службах та застосунках

Android PowerManager API описує різні прапорці, що встановлюють зміну стану пристрою

Flag Value	CPU	Screen	Keyboard
PARTIAL_WAKE_LOCK	On	Off	Off
SCREEN_DIM_WAKE_LOCK	On	Dim	Off
SCREEN_BRIGHT_WAKE_LOCK	On	Bright	Off
FULL_WAKE_LOCK	On	Bright	Bright

Відключення затемнення екрану

Деяким застосункам (іграм, відеоплеєрам тощо) потрібно, щоб екран не вимикався

Доступні два варіанти: у Java-коді або в XML

На відміну від інших wake locks, не потребує спеціального дозволу (permission)

Платформа коректно управляє переходами користувача між додатками, не потрібно турбуватись про вивільнення ресурсів, що не використовуються.

```java
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
  }
```

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:keepScreenOn="true">
    ...
</RelativeLayout>
```

Підтримка процесора ввімкненим

У режимі сну жоден додаток не виконується; навіть ОС призупинена

Фоновим службам потрібно виконуватись, тобто тримати процесор ввімкненим та використовувати ресурси при вимкненому екрані

Такі практики потрібно мінімізувати, тому що вони суттєво впливають на розрядку батареї

Першим кроком у використанні wake lock є додавання дозволу до файлу маніфесту

<uses-permission android:name="android.permission.WAKE_LOCK" />

Рекомендований підхід: брати широкомовний приймач, що використовує службу для виконання роботи (WakefulBroadcastReceiver)

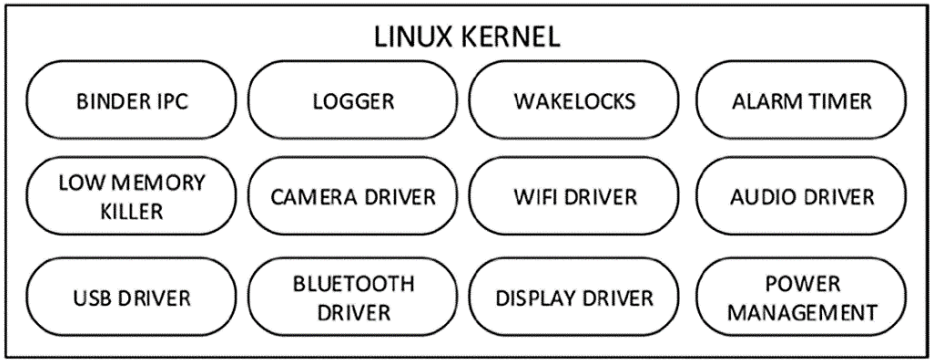
Інакше вейклок встановлюється напряму:

```
PowerManager powerManager = (PowerManager) getSystemService(POWER_SERVICE);
WakeLock wakeLock = powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
        "MyWakelockTag");
wakeLock.acquire();
```

Для вивільнення вейклоку викликають wakeLock.release().

Alarm Timer

Дає можливість для застосунків виконувати заплановані задачі



Він може розбудити пристрій, коли пройде раніше запланований період часу.

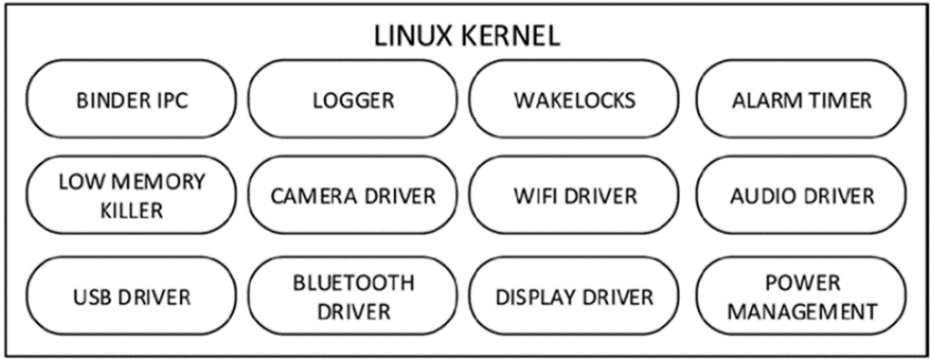
Використовується клас AlarmManager, управляє алармами.

Аларми дозволяють виконувати операції, пов'язані з часом, незалежно від використання додатку.

Наприклад, можна використовувати для ініціації довготривалих (long-running) операцій, таких як одноразовий запуск служби прогнозу погоди кожного дня.

Low Memory Killer

Пам’ять теж обмежений ресурс



Завантаження додатків у пам’ять теж дуже затратна операція

Для уникнення цього всі застосунки знаходяться в оперативні пам’яті.

Навіть ті, з якими вже не взаємодіють.

Це прискорює переключення між ними.

Ціна – швидко закінчується оперативна пам’ять.

В ядро Android додано Viking Killer, який управляє пам’яттю та відновлює її до того, як вона закінчиться

При досягненні певного порогу використаної пам’яті запускається Low Memory Killer, який прибирає найменш важливі з додатків.

Важливість застосунку у великій мірі визначається його видимістю для користувача.

Додаток на екрані – найбільш важливий.

Застосунки у фоні – менш важливі.

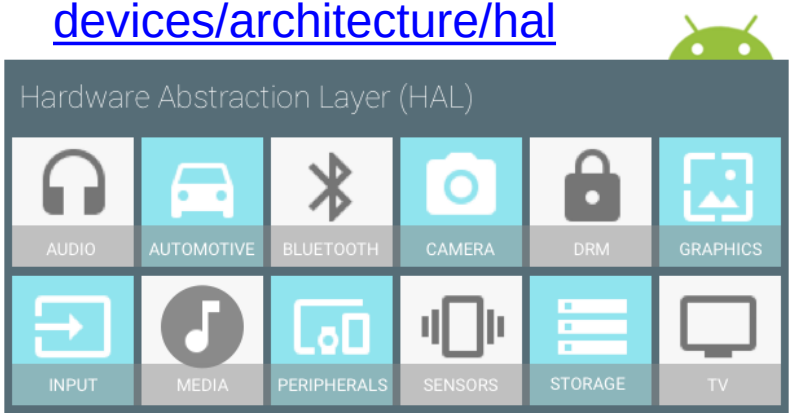
Контекст

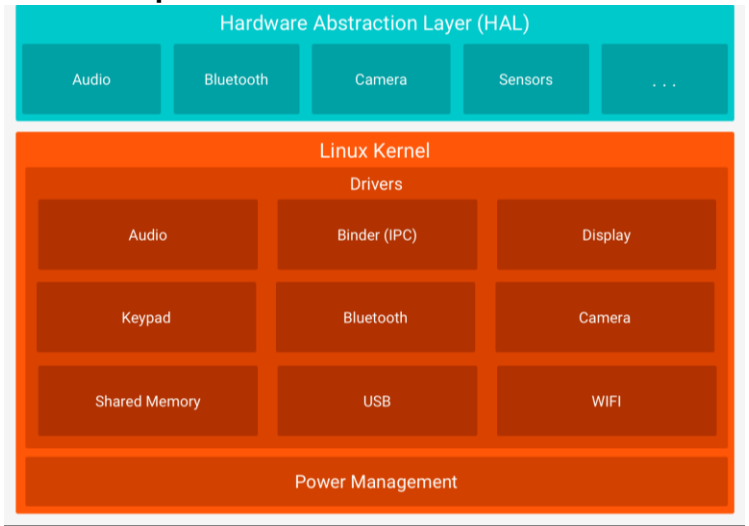
Теми для самоосвіти та самостійного опрацювання

5. ***Рівень HAL.*** Як і більшість популярних ОС, Android має окремий рівень для абстрагування програмного забезпечення від апаратного. В останніх версіях Android були внесені значні зміни в даних механізм

Тема доповіді: Абстрагування від апаратного забезпечення та його еволюція в операційній системі Android

https://source.android.com/devices/architecture/hal





Файлова система Android

Після Android 2.3 використовується файлова система Ext4.

Офіційна причина відмови – відсутність підтримки багатопоточності системою YAFFS2

Файлова система Android також структурована таким чином, щоб було простіше оновлювати різні частини платформи, не впливаючи на решту

Це досягається шляхом утримування різних частин платформи у різних розділах (partitions) системи

Ext4
File System

Розділи файлової системи

Системні розділи залежать від виробників пристроїв. Найбільш поширені

/boot: містить boot loader та Linux-ядро. Змінюється лише при оновленні системи, в процесі роботи дозволу на запис у неї немає.

/system: зберігає всі системні файли та додатки, які постачаються виробником пристрою.
/recovery: містить recovery-образ для відновлення системи

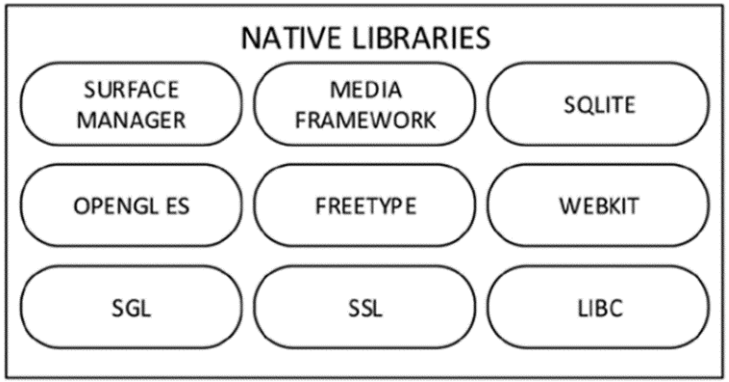
/data: зберігає встановлені користувачем додатки та їх дані. Має доступ до запису в процесі виконання ОС, проте захищена file system permissions.

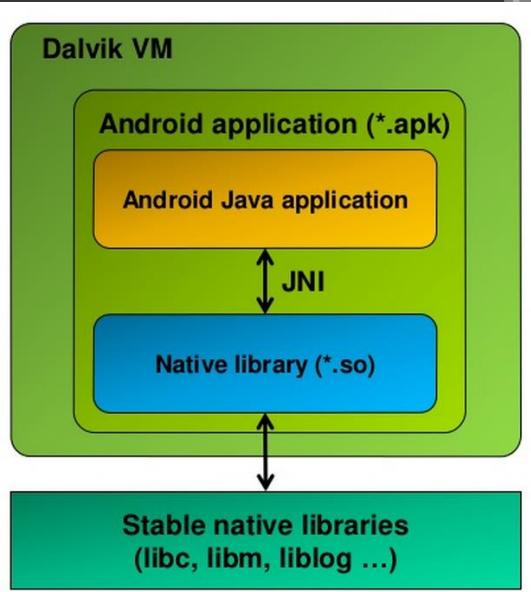
/cache: містить тимчасові файли і файли, що часто викликаються. На більшості Android-пристроїв зберігається в оперативці. Після перезавантаження очищається.

/recovery: спеціальна папка для бекапів. Розділ recovery можна розглядати як альтернативний розділ для завантаження (boot), що дозволяє пристрою перезавантажуватись у recovery-консоль.

/misc: містить різні системні налаштування у формі on/off перемикачів. Ці налаштування можуть включати CID (Carrier or Region ID), USB-конфігурацію, налаштування апаратного забезпечення тощо. Якщо цього розділу не буде, деякі функції пристрою не працюватимуть нормально.

Нативні бібліотеки



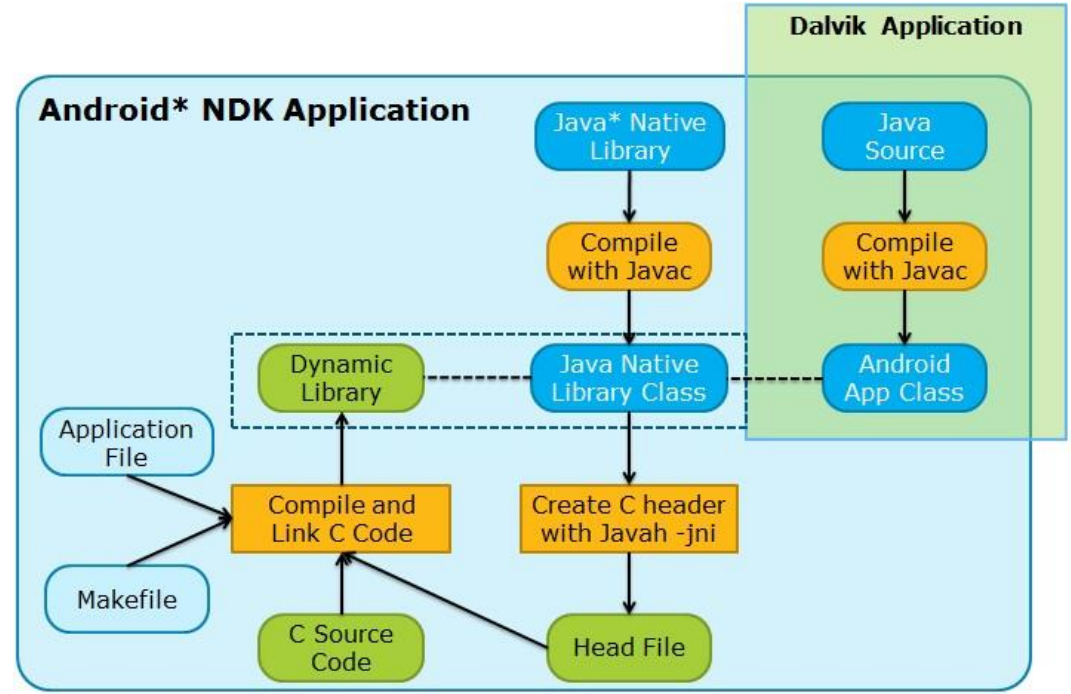


SQLite
WebKit: забезпечує HTML/CSS рендеринг та JavaScript-виконання, підтримка веб-технологій
OpenGL ES: високопродуктивний 2D/3D рендеринг
Open Core: медіа-фреймворк
OpenSSL: реалізує протоколи Secure Socket Layer (SSL) і Transport Level Security (TLS) для безпечної взаємодії Android-додатків з віддаленими службами з використанням криптографії

З NDK можна виконувати

- збірку нативних бібліотек (*.so та ін.), які викликатимуться в Java-код додатку за допомогою Java Native Interface
- збірку виконуваних файлів (не рекомендується використовувати)
- відлагоджувати нативні програми (gdb)

Android NDK



Можна писати повністю нативні додатки під Android, проте рекомендується додавати Java-обгортку (wrapper) для підтримки життєвого циклу застосунку.

Рекомендоване використання нативних функцій:

- При роботі з графікою та обчислювально затратними алгоритмами
- Для збірки нативних бібліотек

Android Runtimes

Android Runtime (ART) – нова JVM, представлена в Android 4.4 експериментально і офіційно затверджена в Android 5.0.

В попередніх версіях додатки запускались на базі Dalvik Virtual Machine.

У порівнянні з just-in-time (JIT) підходом, який використовувався у Dalvik VM, ART покладається на ahead-of-time (AOT) компіляцію

Це дозволяє транслювати байткод у машинний код протягом інсталяції застосунку, а не в процесі його виконання, як було раніше.

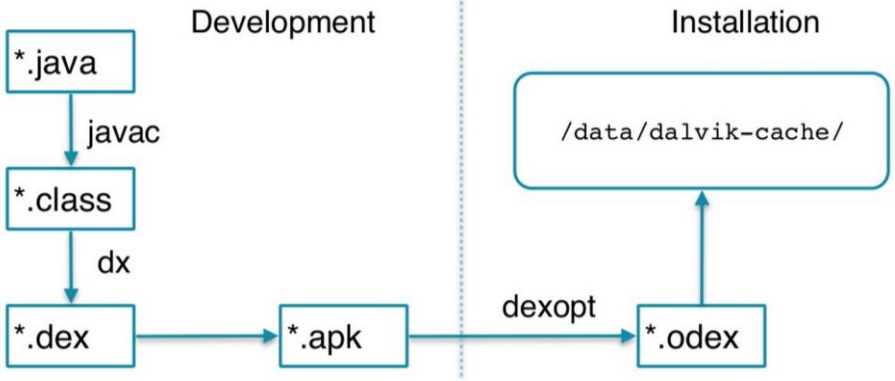
Це надає можливість коду застосунку виконуватись напряму середовищем виконання пристрою

Для того, щоб пакети та виконувані файли були сумісними як з ART, так і Dalvik VM, пакети досі готуються на основі Dalvik-специфікацій

У зв'язку з оптимізацією під мобільні платформи Dalvik VM розуміє байткод лише одного спеціального типу - Dalvik Executable (DEX).

Компіляція для Dalvik VM та ART

Compiling for Dalvik



- JIT compilation
- Increased CPU usage due to JIT
- Lags in applications due to garbage collector pauses and JIT
- Lower memory footprint due to JIT
- Apps install times lower than ART as compilation is performed later

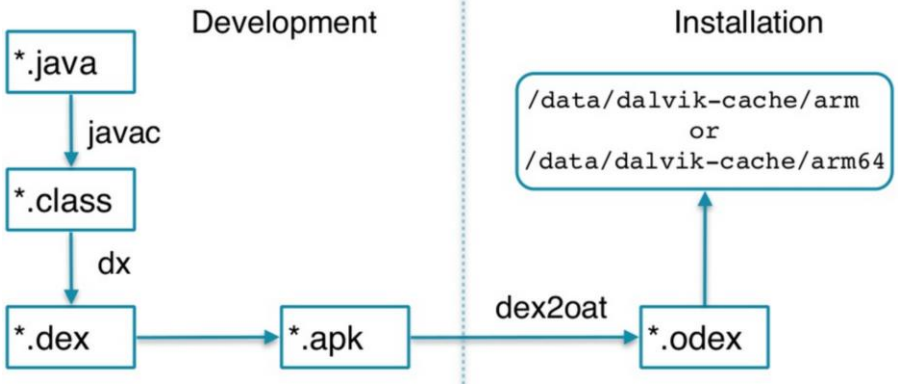
Uses Just-In-Time (JIT) approach, which results in lower storage space consumption but longer app load times

Cache builds up over time, so boot times are faster

Works better for lower internal storage devices as space occupied is lesser

Is stable and time tested – VM of choice for app developers

Compiling for ART



- AOT compilation
- Improved battery life
- Reduced application lags and better user experience
- Higher memory footprint due to AOT
- Increased apps install times as compilation is done during install
- Improved developer tools
- Improved garbage collector

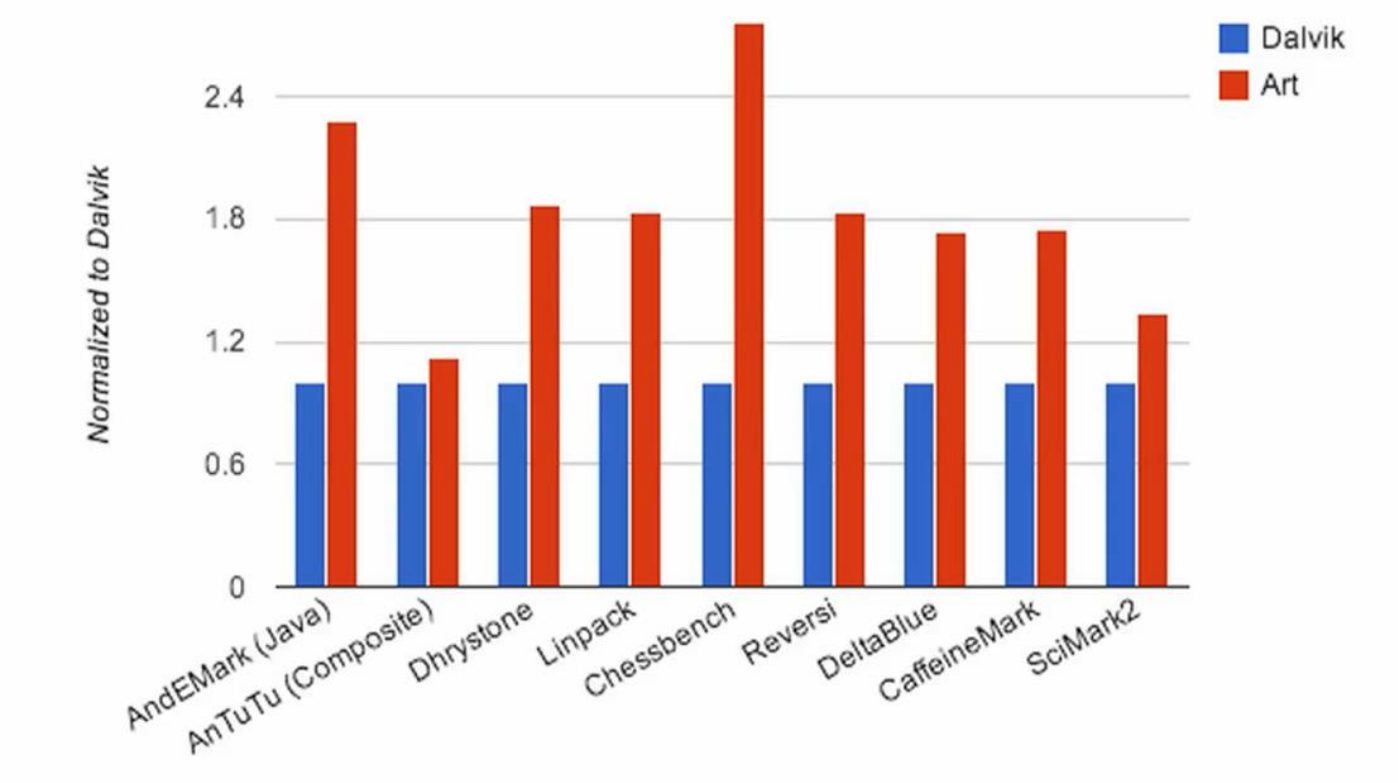
Uses Ahead-Of-Time (AOT) approach, which compiles apps when they're installed, resulting in faster load times and lower processor usage

Cache is built at first boot, hence rebooting device takes significantly longer

Consumes much more internal storage space since it stores compiled apps in addition to the APKs

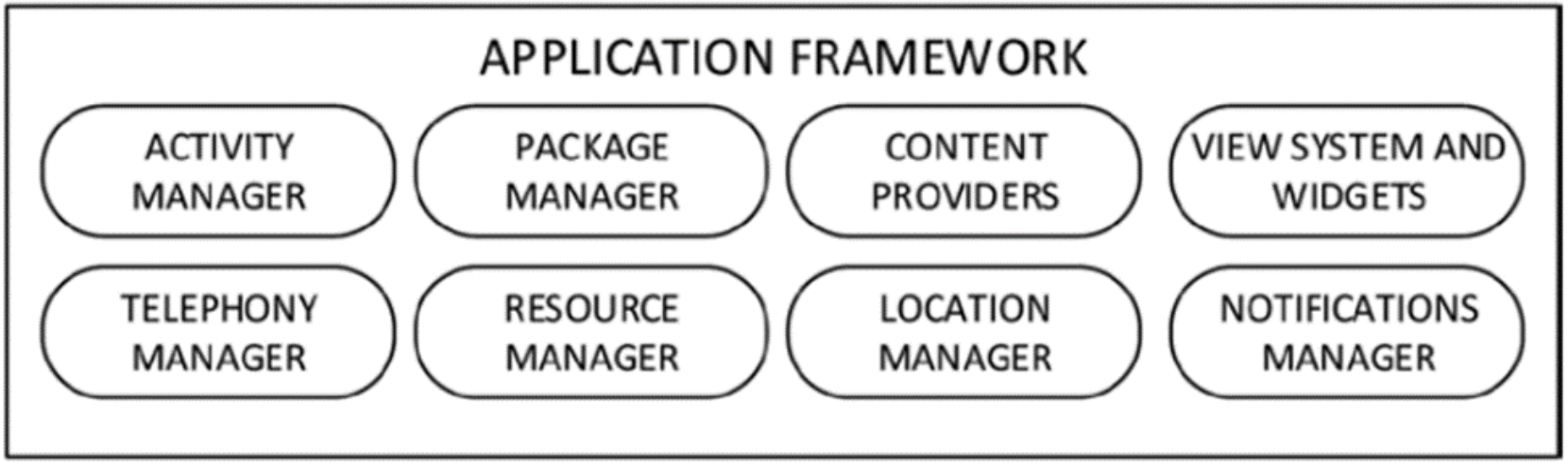
Is highly experimental and new – not a lot of support from app developers yet

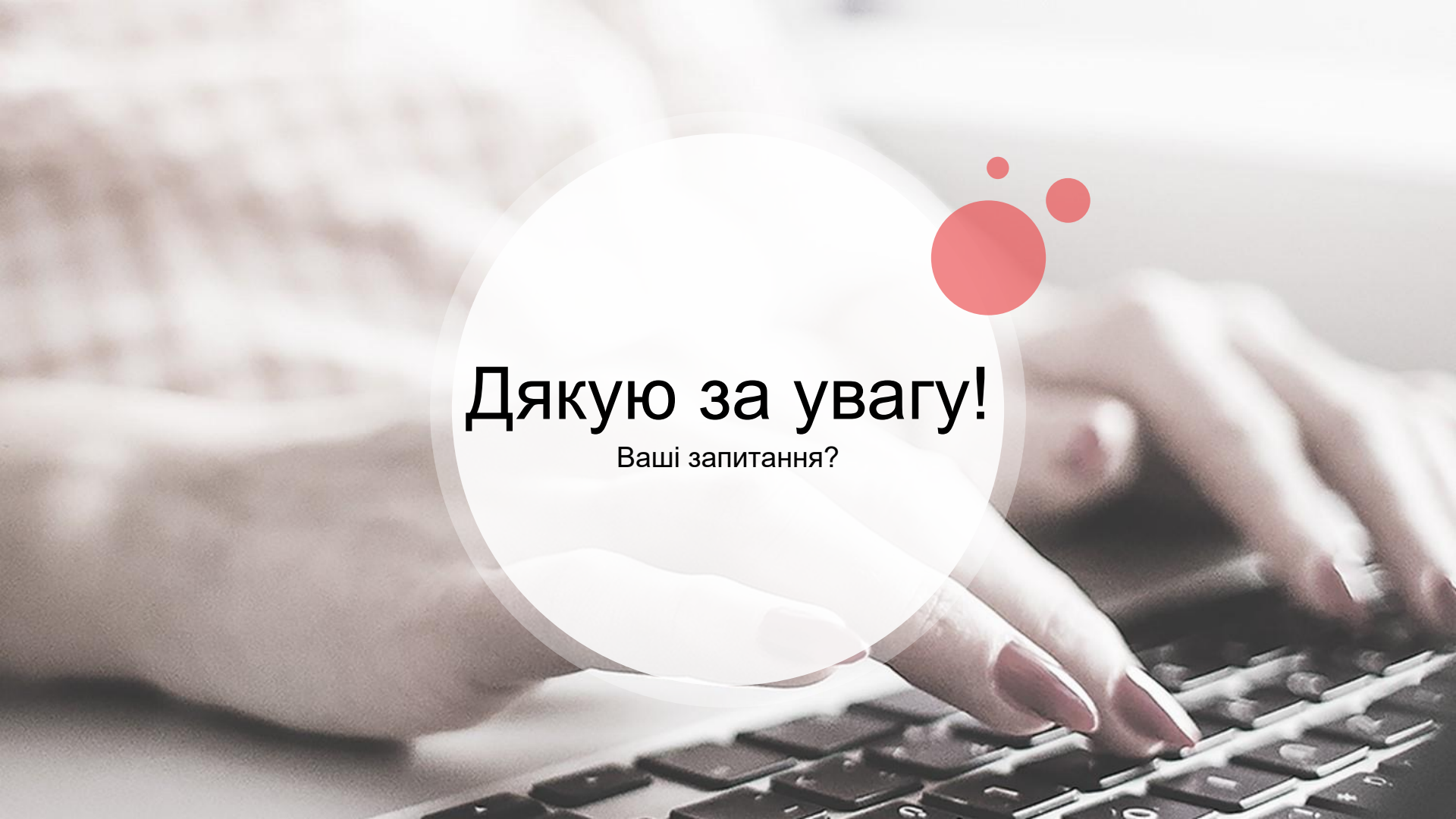
Dalvik vs ART (Nexus 5)



Application Framework

Application framework працює поверх ART VM та забезпечує інтерфейс для взаємодії Android-застосунків з платформою та пристроєм.





Дякую за увагу!

Ваші запитання?