Google Android Introduction

AESOP/고도리

2011년 07월 16일

Index

- 안드로이드 개요
- 안드로이드 특징
- 안드로이드 구성 요소 및 구조
- 안드로이드 개발방법

1.1 Android 개요

안드로이드 소개

- Google에서 개발한 스마트 폰용 오픈소스 플랫폼
  - ✓ 2011.07월 현재 관련 플랫폼 버전
    - Phone용: 2.3 Gingerbread
    - Tablet용: 3.x Honeycomb
    - Google TV
  - ✓ 2011.09월에 발표될 Icecream Sandwitch버전에서는 위의 세 플랫폼 통합 예정
    - USB host API 추가
    - Android Open Accessory Development Kit
      - 2.3.x 버전에도 존재하나 사용 용도는 Icecream Sandwitch version서부터라고 생각됨

- 플랫폼을 다루기 위하여 여러가지 프로그래밍 언어로 구성
  - ✓ Application: JAVA
  - ✓ Native Framework: C/C++
  - ✓ Kernel: C

안드로이드 소개

- 모바일 소프트웨어 개발을 위한 공개 개발 환경
- OS, 미들웨어, 어플리케이션이 완전히 분리된 구조
- Open Handset Alliance (OHA) 프로젝트
- 리눅스 운영체제 기반
  - ✓ *Linux kernel v2.6.25 이후 버전 지원 (현재 gingerbread는 v2.6.35.7)*
- 자바 기반의 손쉽고 빠른 개발 환경 지원
  - ✓ Emulator환경 지원
  - ✓ Mac, Windows, Linux
- Apache 2 라이선스 기반
  - ✓ License 부분에서 다시 설명

안드로이드 소개

- 통신사 입장
  - ✓ iPhone의 독점적 권위에 위기의식 느낌
  - ✓ 경쟁 플랫폼 도입의 필요성
  - ✓ 스마트폰 서비스 플랫폼의 부재

- 단말 제조사 입장
  - ✓ iPhone의 독주 및 폐쇄적인 정책에 따른 경쟁 플랫폼 부재
  - ✓ 기존의 Feature Phone 플랫폼을 너무 오래 유지한 덕에 변화에 대한 필요성을 느끼지 못했음
  - ✓ Apple 사의 단말 플랫폼외에 서비스 플랫폼 경쟁력의 부재(ex> 앱스토어)

- *위와 같은 이유로 현재 가장 경쟁력 있는 Android 플랫폼을 대부분 통신사와 제조사가 선택*

안드로이드 소개

- 통신사 위주의 시장에서 제조사와 사용자 위주의 모바일 시장으로 변화 기회

- 구글의 입장
  - ✓ 검색시장의 점유율 증대
  - ✓ 서비스 강화 및 광고수익 증대

- 구글의 정책의 목적은 무엇일까?
  - ✓ iPhone의 성공 요소를 그대로 따라함
  - ✓ 서비스 시장의 장악
  - ✓ Icecream Sandwitch에서 추가되는 특징을 봤을 경우
    - ➢ Game console 시장으로의 진출과 Smart TV로의 진화

1.2 Android 특징

안드로이드의 특징

- 다른 스마트폰 플랫폼과 다른 특징
  - ✓ 오픈소스 소프트웨어
    - ➢ 지속적인 소스에 대한 변화가 이루어짐
  - ✓ 구글이라는 광고/검색 시장 1위라는 기업이 만들었다는 점

- Application Framework
  - ✓ Application의 재사용과 대체가 가능함
- Dalvik VirtualMachine
  - ✓ 모바일 디바이스에 최적화됨.
- 통합 브라우저
  - ✓ 오픈 소스 Webkit 엔진 기반.
- 그래픽 최적화
  - ✓ 2D/3D(OpenGL ES 1.0 사양 - Android 2.0서부터는 OpenGL2.0 지원)
  - ✓ Hardware 가속 기능 지원

안드로이드의 특징

- SQLite
  - ✓ 데이터 저장을 위한 데이터베이스 엔진 제공
- 미디어 지원
  - ✓ 다양한 오디오, 비디오 및 이미지 포맷 지원 (MPEG4, H.264, MP3, JPG, PNG, GIF 등)
  - ✓ 기존의 Phone에 적용된 format만 지원, PC에서 사용하는 format들은 지원하지 못함(ex> AVI/WMV 등)
- 하드웨어 의존적인 기술들
  - ✓ GSM 텔레포니 기술, Bluetooth, EDGE, 3G, and WiFi, Camera, GPS, e-compass와 accelerometer
- 유용한 개발환경을 지원
  - ✓ 디버깅을 위한 Eclipse IDE 기반의 디바이스 애뮬레이터를 지원
  - ✓ 메모리 및 성능 프로파일링 기능 지원
  - ✓ Eclipse IDE에 기반한 x86 기반의 크로스 개발 환경 지원

오픈소스 라이센스

- 안드로이드는 오픈 소스이므로 관련 Open source license 정책에 영향을 받음.
- GPL
  - ✓ GPL로 작성된 소스를 기반으로 작성하여 상용화 했을 경우 반드시 소스 공개
  - ✓ version 2
    - ➢ v2로 작성된 코드와 결합되었을 경우 모든 소스가 version 2가 됨
  - ✓ version 3
    - ➢ v2의 문제점인 기존의 라이센스 코드와 GPL 코드가 결합 불가했던 부분을 수정
- LGPL(Lesser GPL)
  - ✓ Library GPL
  - ✓ 관련 소스를 수정했을 때는 공개, 동적로딩 라이브러리로 배포시 이 라이브러리를 사용한 소스는 공개하지 않아도 됨

오픈소스 라이센스

- **BSD, Apache license**
  - ✓ 진정한 공개 라이센스
  - ✓ 소스의 사용에 제약이 없음(상용화시 공개의무 없음)
  - ✓ Apache license는 BSD 라이센스에 법적인 명시조항만 추가작업한 라이센스

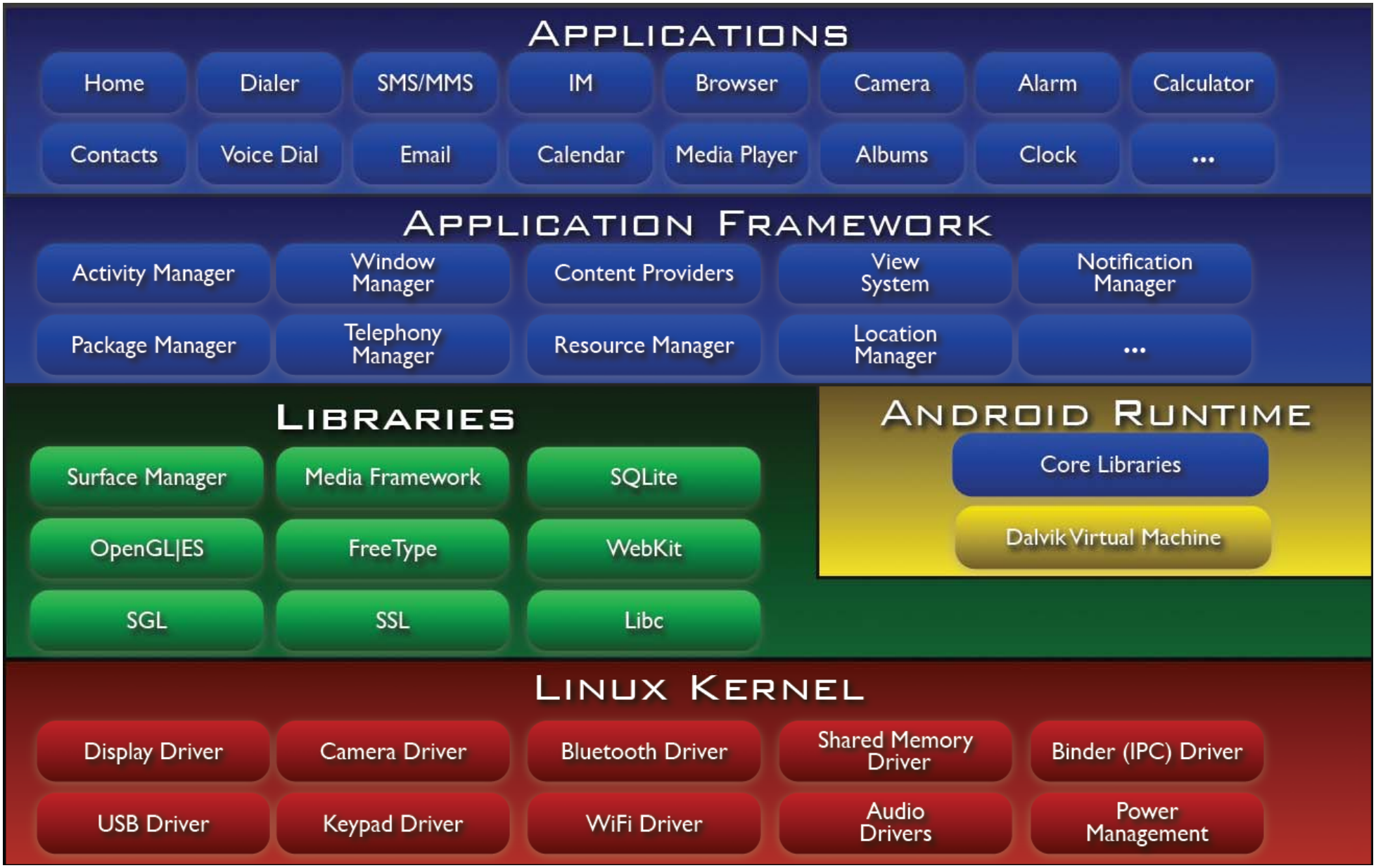
- **MPL(Mozilla Public License)**
  - ✓ 전체 소스코드내에서 MPL 라이센스를 따르는 파일에 한정하여 소스공개 요구
  - ✓ 상용화에 유리

안드로이드 라이센스

- Linux Kernel은 GPL 라이선스를 사용
- Android는 대부분 Apache v2 라이선스를 사용
- GPL과 Apache
- Apache license
  - ✓ 아파치 웹서버를 포함한 아파치 재단(ASF: Apache Software Foundation)의 모든 소프트웨어에 적용
  - ✓ BSD 라이센스와 비슷하여 소스 코드 공개 등의 의무가 발생하지 않음.
  - ✓ 다만 'Apache'라는 이름에 대한 상표권을 침해하지 않아야 한다는 조항이 명시적으로 들어가 있음.
  - ✓ 특허권에 관한 내용이 포함되어 BSD 라이센스보다는 좀 더 법적으로 정리된 것이 특징.
  - ✓ 특히 Apache License 2.0에서 특허에 관한 조항이 삽입되어 GPL 2.0으로 배포되는 코드와 결합되는 것이 어려웠으나, GPL 3.0(안)에서는 이 문제를 해결하여 Apache License로 배포되는 코드가 GPL 3.0으로 배포되는 코드와 결합하는 것이 가능.

- **1.3 ~ 1.7 안드로이드 구성요소 및 구성**

안드로이드의 구성요소 및 구조 - 전체



안드로이드 플랫폼의 구조

- 크게 4~5개의 구조로 볼 수 있음
  - ✓ Applications (JAVA로 작성된 실행프로그램)
  - ✓ Application Framework (JAVA API에 해당함)
    - ➢ 이 계층과 Native framework 사이에는 JAVA Virtual Machine(Dalvik)과 JNI(JAVA Native Interface)가 존재
  - ✓ C/C++로 작성된 Libraries(Native라고도 표현한다)
    - ➢ 중요 멀티미디어, 디스플레이 엔진등이 여기에 속함
  - ✓ *HAL(Hardware Abstraction Layer): 제외되는 경우가 있음*
  - ✓ 안드로이드용으로 패치된 리눅스 커널

안드로이드용 리눅스 커널 패치

- 안드로이드에 알맞게 수정된 리눅스 커널
  - ✓ 수정된 내용
    - ➢ 안드로이드 프레임워크에서 사용되는 부분(binder, ashmem, logger 등)
    - ➢ 휴대폰에 알맞게 수정이 되는 부분(alarm, low memory killer, power management 등)

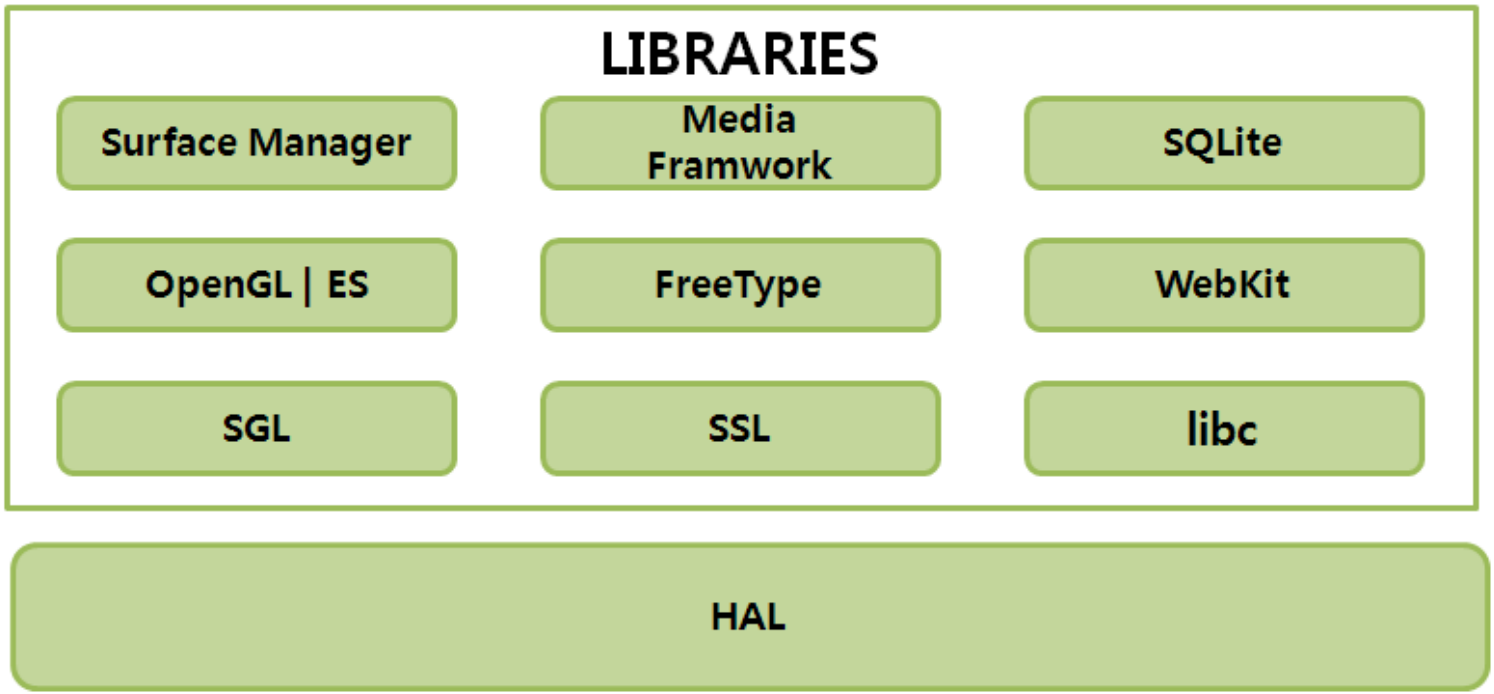
Feautures	Description
Alarm	Android.app.AlarmManager
Low Memory Killer	Android Out of Memory(OOM) Killer Driver
Ashmem	Android(Anonymous) Shared Memory Driver(Share memory kernel level)
Kernel Debugger	
Binder	1. Driver to facilitate inter-process communication(IPC). 2. Hight Performance through shared memory. 3. Per-Process thread pool for process requests. 4. Reference counting, and mapping of object references across processes. 5. Synchronous calls between processes.
Power Management	1. Built on top of standard Linux Power Management(PM). 2. More aggressive power management policy. 3. Components make requests to keep the power on through "wake locks". 4. Supports different types of wake locks.
Logger	1. Support Android Logcat Utility: Using logcat commands, Filtering Log output, Controlling Log output format, viewing Alternative Log Buffer, Listing of Logcat Command Options.

안드로이드 커널 특징

- Low memory killer
  - ✓ 기존 리눅스용의 Out of memory killer와의 차이
    - ➢ 메모리 부족시 프로세스 우선순위가 낮은 그룹을 제거
    - ➢ binder 시스템을 이용하기 때문에 하나의 응용 프로그램은 여러 개의 프로세스로 이루어진다
- binder
  - ✓ 안드로이드 프레임워크에서 중요하게 사용되는 프로세스간 통신 메커니즘
- ashmem(anonymouns shared memory)
  - ✓ 공유 메모리 메커니즘, 주로 바인더와 연동해서 사용된다
- Power management
  - ✓ wakelock과 같은 폰에서 사용이 용이한 locking 메커니즘등을 도입
  - ✓ 기존의 리눅스 전원관리는 주로 노트북에만 맞춰져 있음
- kernel logger
  - ✓ 안드로이드에서 발생하는 로그를 처리할 수 있도록 만든 메커니즘
  - ✓ printf 등과 같은 것은 사용되지 않는다

안드로이드 라이브러리 계층

- Android Native Framework 이라고 불림
- 세가지 부분으로 나눌 수 있음
  - ✓ 안드로이드 이전에 개발된 라이브러리 혹은 외부에서 기존 프로젝트에서 사용하던 라이브러리(주로 external 디렉터리)
  - ✓ 안드로이드 Framework 부분(frameworks/base 디렉터리 중 C++로 작성된 것 - JNI는 제외)
  - ✓ 안드로이드의 HAL 부분



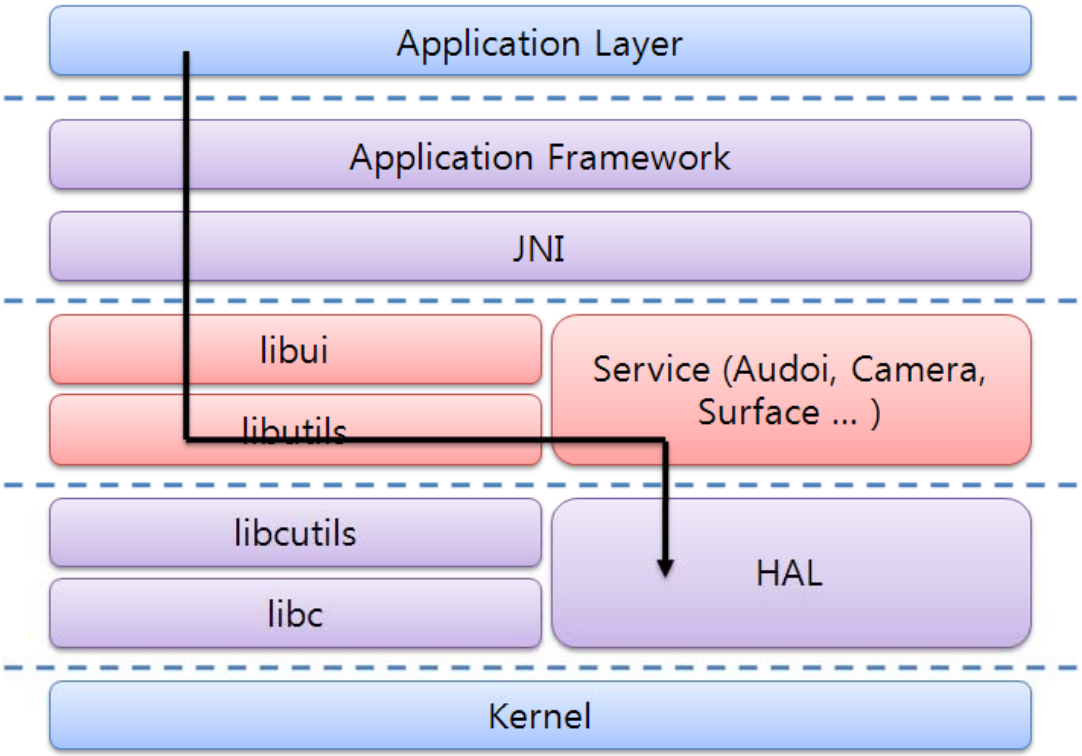
안드로이드 라이브러리 계층

- 주요 내용
  - ✓ Java와 인터페이스하기 위해 존재한다.
  - ✓ Surface Manager - UI Windows 제어를 담당한다.
  - ✓ 2D/3D 그래픽 라이브러리를 포함한다.
  - ✓ 멀티미디어 코덱, SQLite DB 엔진, WWW 엔진을 포함한다.

- 이 계층에서 주의해서 봐야할 점
  - ✓ 고속, 대용량 데이터를 처리하는 부분
  - ✓ 안드로이드 핵심인 surface manager, media framework이 존재
  - ✓ binder 메커니즘으로 구성되어 있음
  - ✓ HAL과 연동되는 부분
  - ✓ 주로 binder 서버의 형태로 구성되어 있음(Binder service의 형태)

안드로이드 라이브러리 계층 – 함수 호출 구성

- 응용 프로그램에서의 함수 호출과 라이브러리 계층에서의 함수의 움직임
  - ✓ 응용 프로그램에서는 libui, libmedia, libutils까지 호출
  - ✓ libbinder를 이용 Binder Native Service 들과 연동(*IPC이용*)
  - ✓ Binder Native Service들은 HAL을 통해서 커널로 진입

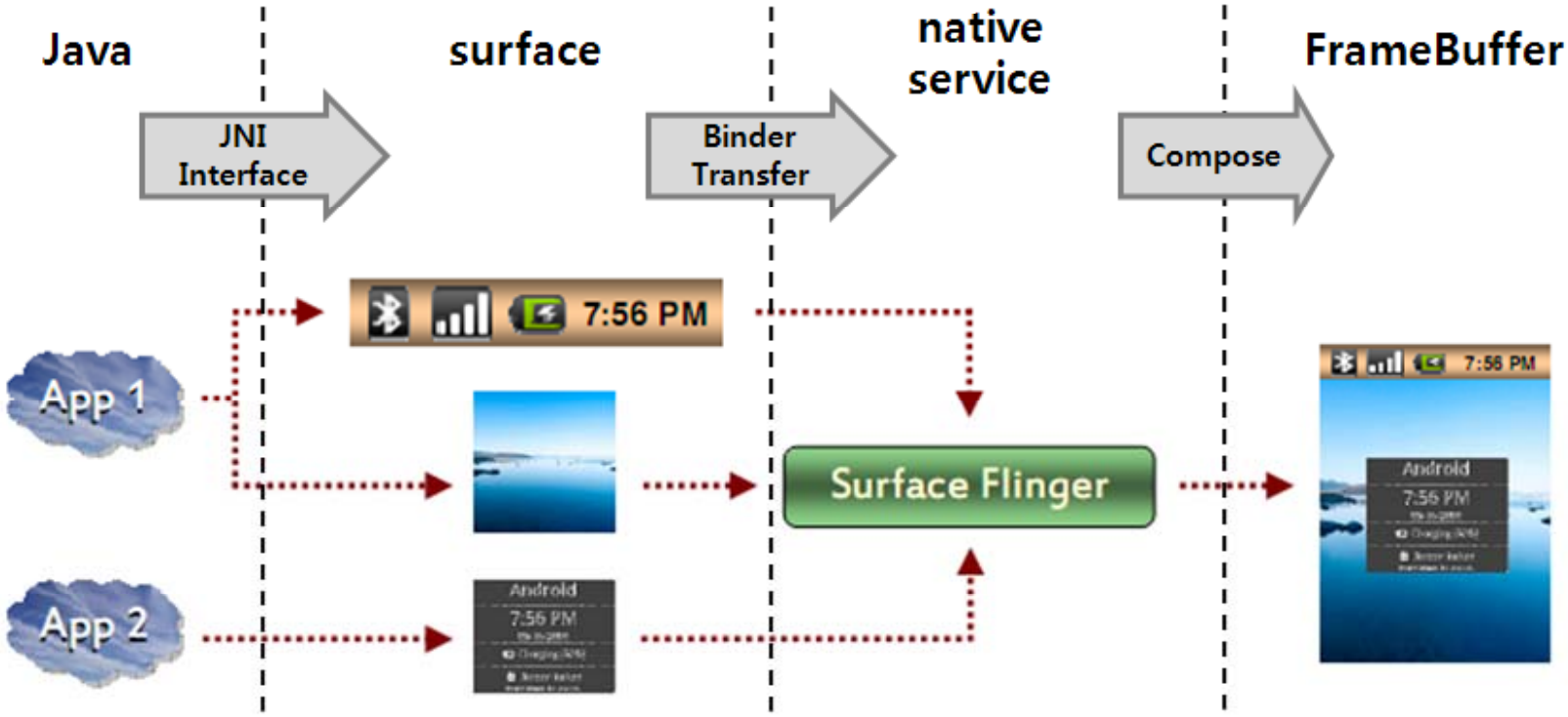


안드로이드 라이브러리 계층 – 함수 호출 구성

- 앞장의 그림과 같이 함수 호출이 구성되는 이유
  - ✓ UI에서 하드웨어를 제어할 경우 유닉스 system call의 특징이 blocking I/O의 문제 때문에 binder를 사용
  - ✓ 구성상 안드로이드 응용 프로그램에서 시스템으로 접근할 수 있는 권한을 주지 않음
    - ➢ 보안 혹은 주요 시스템의 제거등과 같은 문제
    - ➢ ex> 전화통화 관련 프로그램 삭제 등

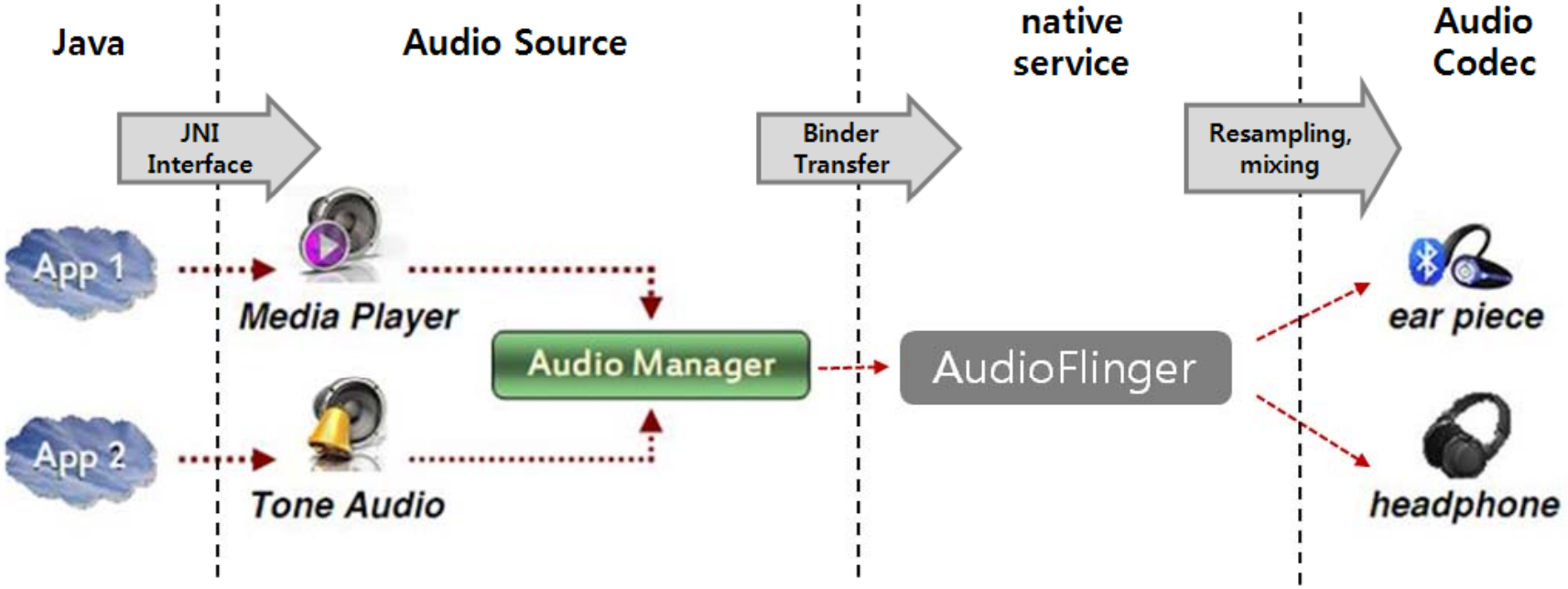
대표적인 Binder 서비스

- **Surfaceflinger**
  - ✓ Surface(Window)를 합성하는 서버
  - ✓ X-windows의 X-server와 유사한 기능



대표적인 Binder 서비스

- **Audioflinger**
  - ✓ 사운드 시스템의 주요부분을 담당하는 서비스
  - ✓ 오디오 믹싱기능과 하드웨어 제어가 주 목적



안드로이드 라이브러리 계층 - HAL

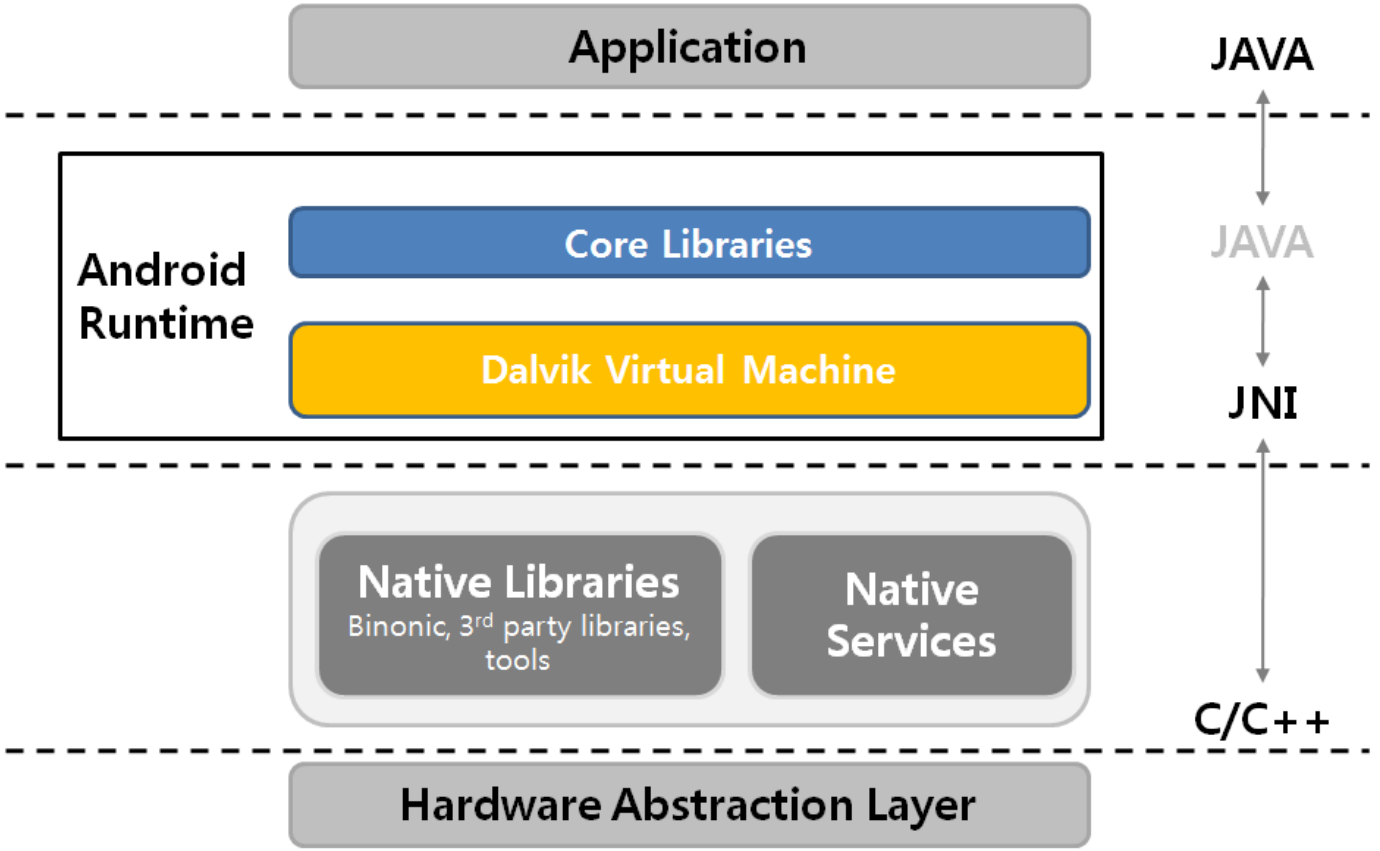
- HAL
  - ✓ 하드웨어 인터페이스 담당 모듈
  - ✓ 많은 모듈로 구성되어 있고, 대체적으로 4가지의 형태로 존재
  - ✓ 1. Input
    - ➢ key, touch, mouse 등등
    - ➢ frameworks/base/libs/ui/EventHub.cpp
  - ✓ 2. Native Service에서 직접 제어하는 경우
    - ➢ 표준 I/O interface class를 가지고 있음.
    - ➢ 단, 각 모듈별 template interface는 서로 틀림
    - ➢ Audio, Surface(Video), Camera 등...
  - ✓ 3. 기존 Daemon process에의 client로 동작하는 경우
    - ➢ wifi, ril, bluetooth
  - ✓ 4. 그 외의 hw module
    - ➢ hw 표준 인터페이스를 가지고 있음(class or structure)
    - ➢ ex> sensor관련 부분들, gralloc
    - ➢ root filesystem의 system/lib/hw/* 파일들

안드로이드 Runtime

- JAVA Runtime과 동등한 레벨
  - ✓ 프로세스가 동작 되면서 Library를 호출 할 때 사용
  - ✓ JAVA 어플리케이션에서 Libc 기반의 C/C++ Library를 호출 할 때 Runtime Library를 호출
  - ✓ Android는 Dex.(Dalvik Executables (DEX)) 실행 파일의 구조를 사용, 실행 파일이 Symbolic Resolution을 통하여 함수를 가지고 있다가 Library를 호출
  - ✓ Android Runtime은 Dalvik VM과 Core Libraries로 구성
  - ✓ Android의 개발은 Eclipse의 ADT Plug-in을 통해서 JAVA로 컴파일 되고, Class와 Resource가 Dx 컨버터를 통해 Android App(.apk)로 변환되어 Dalvik VM위에서 동작
  - ✓ Dalvik VM은 작은 메모리에서도 최적화 되는 Dalvik Executable(.dex) 포멧 파일의 실행을 지원
  - ✓ Dalvik VM은 스레딩과 저수준의 메모리 관리 지원을 위해 리눅스 커널을 이용

안드로이드 Runtime

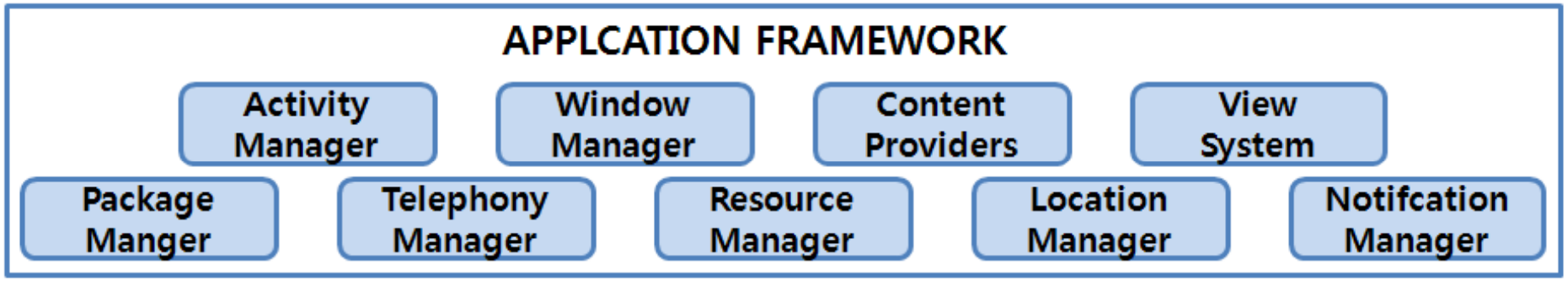
- Android Runtime의 구조



Dalvik VM

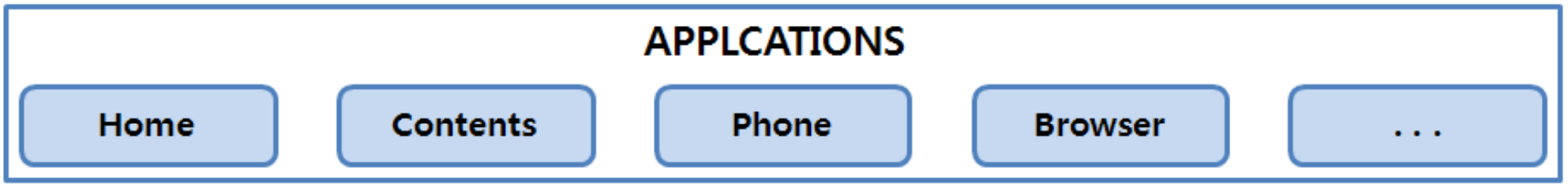
- Dalvik VM
  - ✓ 비표준 JAVA Virtual Machine
  - ✓ Android SDK로는 Java 소스를 Dalvik용 bytecode로 직접 컴파일이 불가능
  - ✓ DEX File 포멧의 Application을 수행
  - ✓ DX 툴을 사용하여 자바의 .class 파일을 Dalvik용 bytecode로 변한하여 실행 코드를 생성
  - ✓ C++ 기반의 Class보다 작고, 호환성이 좋음
  - ✓ SUN의 라이선스 정책에 따른 라이선스 비용을 회피하기 위하여 개발
  - ✓ JAVA는 GPLv2하의 배포된 오픈소스 였으나 Java ME에서 예외가 발생, 핸드셋에 JAVA를 탑해하기 위해서는 라이선스 비용 발생
  - ✓ *현재(2011.07) Sun microsystems를 인수한 Oracle과 Google과의 소송 진행중에 있음*

Application Framework



- JAVA API 기반의 인터페이스 제공
- 주로 JAVA로 이루어진 Binder 서비스와 매니저들이 존재
  - ✓ Activity Manager
    - ➢ 안드로이드 애플리케이션의 수명 주기를 관리한다.
  - ✓ View System
    - ➢ 표준 widget을 담당한다.
  - ✓ Window Manager
    - ➢ 모든 응용프로그램과 관련된 화면을 담당한다.
  - ✓ Package Manager
    - ➢ 시스템에서 동작 중인 응용프로그램의 정보를 담당한다.
  - ✓ etc...

Applications



- JAVA로 구성되어 있음
- 기본 프로그램은 안드로이드 코드 안에 존재
- Dalvik VM에서 실행되므로 메모리를 관리하는데 주의 요함

안드로이드 개발환경

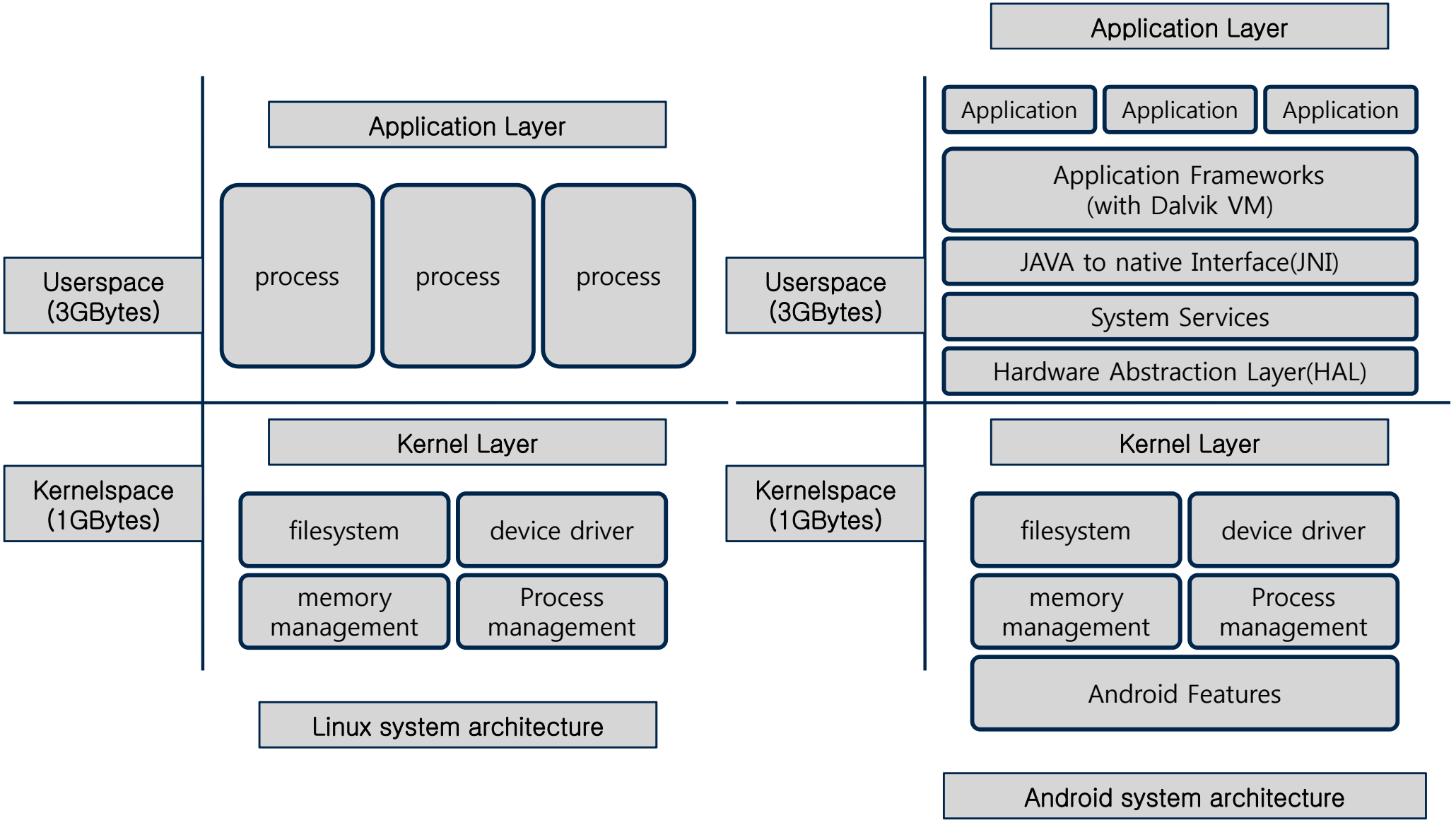
- SDK 개발
  - ✓ Eclipse IDE
    - ➢ 자바 응용 프로그램을 개발하기 위한 통합 개발 플랫폼
    - ➢ 안드로이드 emulator와 연동되도록 구성

- 안드로이드 Emulator
  - ✓ QEMU 기반의 ARM emulator
  - ✓ Eclipse IDE와 연동

- 타겟보드
  - ✓ 안드로이드 플랫폼 개발용 혹은 안드로이드 폰 형태의 보드
  - ✓ Emulator로 처리가 불가능한 실제 하드웨어와 플랫폼 개발용
  - ✓ Evaluation board의 형태로 많이 존재
  - ✓ Nexus-one, Nexus-S와 같은 폰은 직접 개발이 가능하다

리눅스와 안드로이드 시스템 구조

- 리눅스와 안드로이드의 시스템은 다음과 같이 구성된다



Android 부팅의 요소와 Root filesystem

- 안드로이드는 리눅스를 기반으로 한다
  - ✓ 리눅스 부팅의 3가지 요소
    - ➢ 부트로더
    - ➢ 커널
    - ➢ Root Filesystem
      - 안드로이드는 Root filesystem을 새롭게 구성한 것이다.

- 안드로이드 소스 컴파일시 출력되는 결과물
  - ✓ 컴파일 후
  - ✓ out/target/product/{productname}/ 디렉토리에 결과물이 생성
  - ✓ ramdisk.img, system.img, userdata.img 세개의 파일이 생성
  - ✓ root, system, data 란 디렉토리도 생성되어 있음

Android Root Filesystem

- Root filesystem 관련 파일과 디렉토리 관련
  - ✓ ramdisk.img
    - ➢ root 디렉토리를 cpio 형태의 ramdisk로 만든 파일
  - ✓ system.img
    - ➢ system 디렉토리를 yaffs2 filesystem image로 만든 파일
  - ✓ userdata.img
    - ➢ data 디렉토리를 yaffs2 filesystem image로 만든 파일
  - ✓ 이 파일들은 각각 폰의 NAND 파티션 영역에 write된다.

Android Root Filesystem – 부팅 후 구성

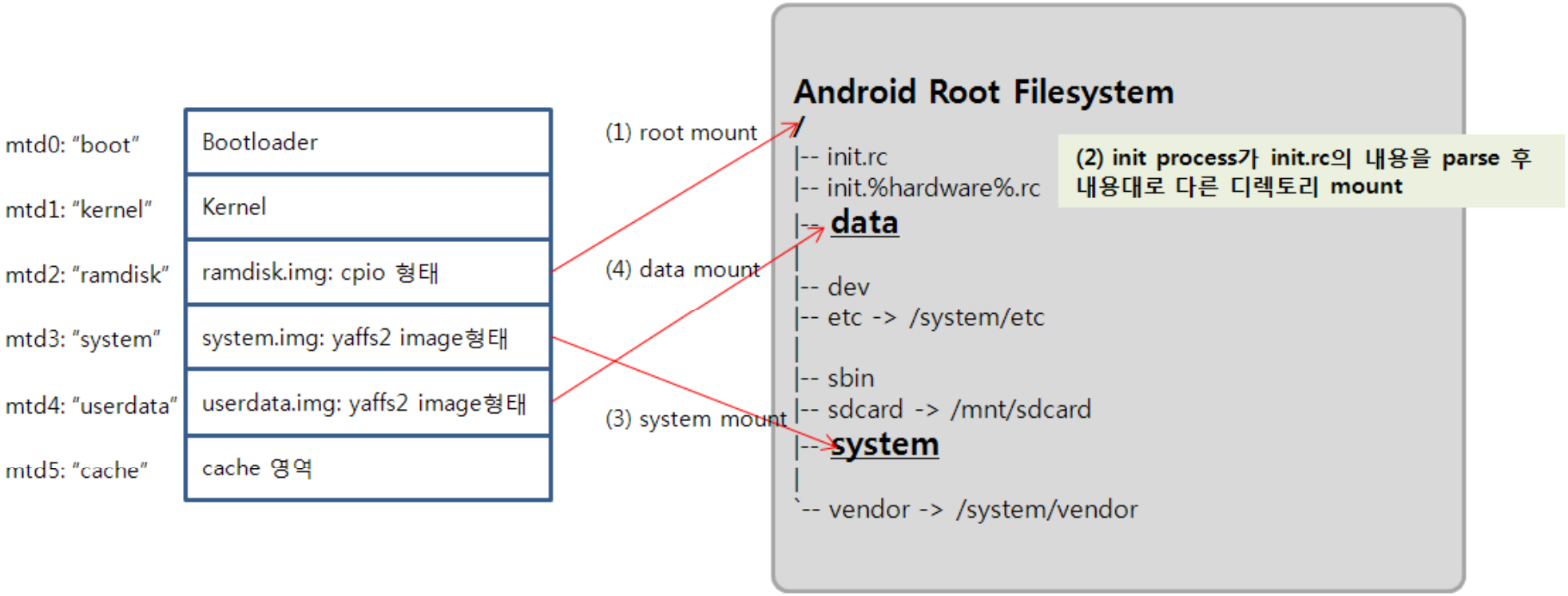
- 안드로이드가 부팅된 후의 디렉토리 구성은 다음과 같다

Android Root Filesystem

```
/
|-- acct                                  Account 관련 시스템 UID 정의
|-- cache                                 Cache 파티션을 위한 영역
|-- config                                Account(google/host)관련 설정
|-- d -> /sys/kernel/debug                시스템 리소스 디버깅 디렉토리
|-- data                                  사용자 데이터, DB, property 리소스
|   |-- anr, app, app-private, backup, dalvik-cache, data, dontpanic
|   `-- local, lost+found, misc, property, secure, system, tombstones
|
|-- dev                                   디바이스 노드 관련 디렉토리
|-- etc -> /system/etc                    각종 설정파일 관련 디렉토리
|-- mnt                                   sdcard 등의 filesystem 및 storage 마운트 포인트
|   `-- asec, obb, sdcard, secure
|
|-- proc                                  procfs 마운트 포인트이자 디버깅 포인트
|-- sbin                                  superuser 용 바이너리 디렉토리
|-- sdcard -> /mnt/sdcard                 sdcard 마운트 포인트
|-- sys                                   sysfs 마운트 포인트, 시스템 리소스 관리
|-- system                                Android Platform 관련 핵심 디렉토리
|   |-- app, bin, etc, fonts, framework, lib
|   |-- media, tts, usr, vendor
|   `-- xbin
|
`-- vendor -> /system/vendor              Chipvendor 관련 리소스 디렉토리
```

Android Root Filesystem과 구성요소의 저장

- **NAND 혹은 Storage의 구성과 Root Filesystem의 구성 매칭**



Root filesystem관련 부팅 순서

- *부팅순서*
  - ✓ 부트로더 실행
  - ✓ 커널로 부팅
  - ✓ ramdisk.img를 ramdisk로 로딩: 이 때 / 디렉토리로 로딩
  - ✓ init.rc 와 init.%hardware%.rc 파일을 로딩
  - ✓ system.img를 yaffs2 filesystem으로 /system 디렉토리로 mount
  - ✓ userdata.img를 yaffs2 filesystem으로 /data 디렉토리로 mount
  - ✓ NAND 파티션의 cache영역을 /cache 디렉토리로 mount
  - ✓ 부팅완료

Android toolchain, Root Filesystem

- Android 관련 toolchain
  - ✓ gcc: GNU compiler사용
  - ✓ bionic C
    - ➢ BSD의 libc를 개선
    - ➢ 라이선스 : User Application에서 GPL 문제 회피
    - ➢ Size : 약 200k로 Glibc의 절반 크기
    - ➢ 개별 프로세스마다 포함되어야 하는 부분이기 때문에 제한된 CPU 파워에서 빠른 속도 동작할 수 있도록 하기 위함
  - ✓ Linker의 재구성

- Root filesystem binary 구성
  - ✓ 기존의 Linux에서 많이 사용하는 busybox를 사용하지 않음
  - ✓ toolbox를 사용
  - ✓ 개발시는 busybox를 사용하는게 상대적으로 개발에 용이함

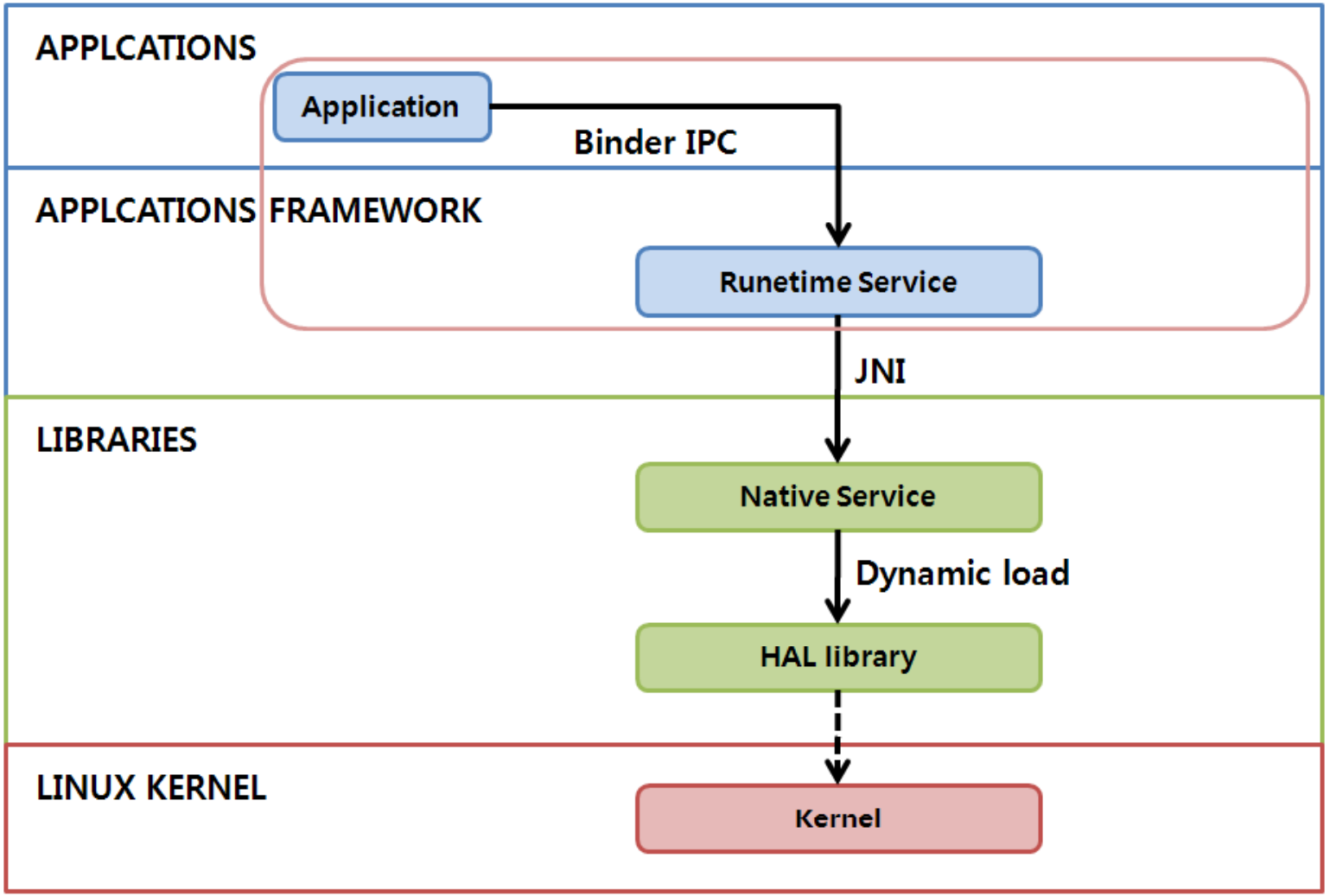
1.8 안드로이드 개발방법

안드로이드 개발방법

- 안드로이드 플랫폼 개발에서의 여러가지 방법
  - ✓ SDK(Software Development Kit)
    - ➢ 응용 프로그램 개발
  - ✓ PDK(Platform Development Kit)
    - ➢ Android 소스를 가지고 개발하는 방법
    - ➢ HAL의 구성에 따라 개발방법에 차이가 남
  - ✓ NDK(Native Development Kit)
    - ➢ 응용 프로그램 개발
    - ➢ C/C++로 작성한 라이브러리를 JAVA에서 직접 호출할 수 있도록 구성
    - ➢ JNI를 자동으로 매핑해주는 툴이라고 보는 편이 적합함
    - ➢ PDK에서도 NDK를 이용해 개발이 가능하다
  - ✓ SDK와 NDK는 안드로이드 소스없이 개발이 가능
    - ➢ 단, NDK는 PDK에도 포함이 되어 있으므로, 소스가 있어도 되고 없어도 개발 가능

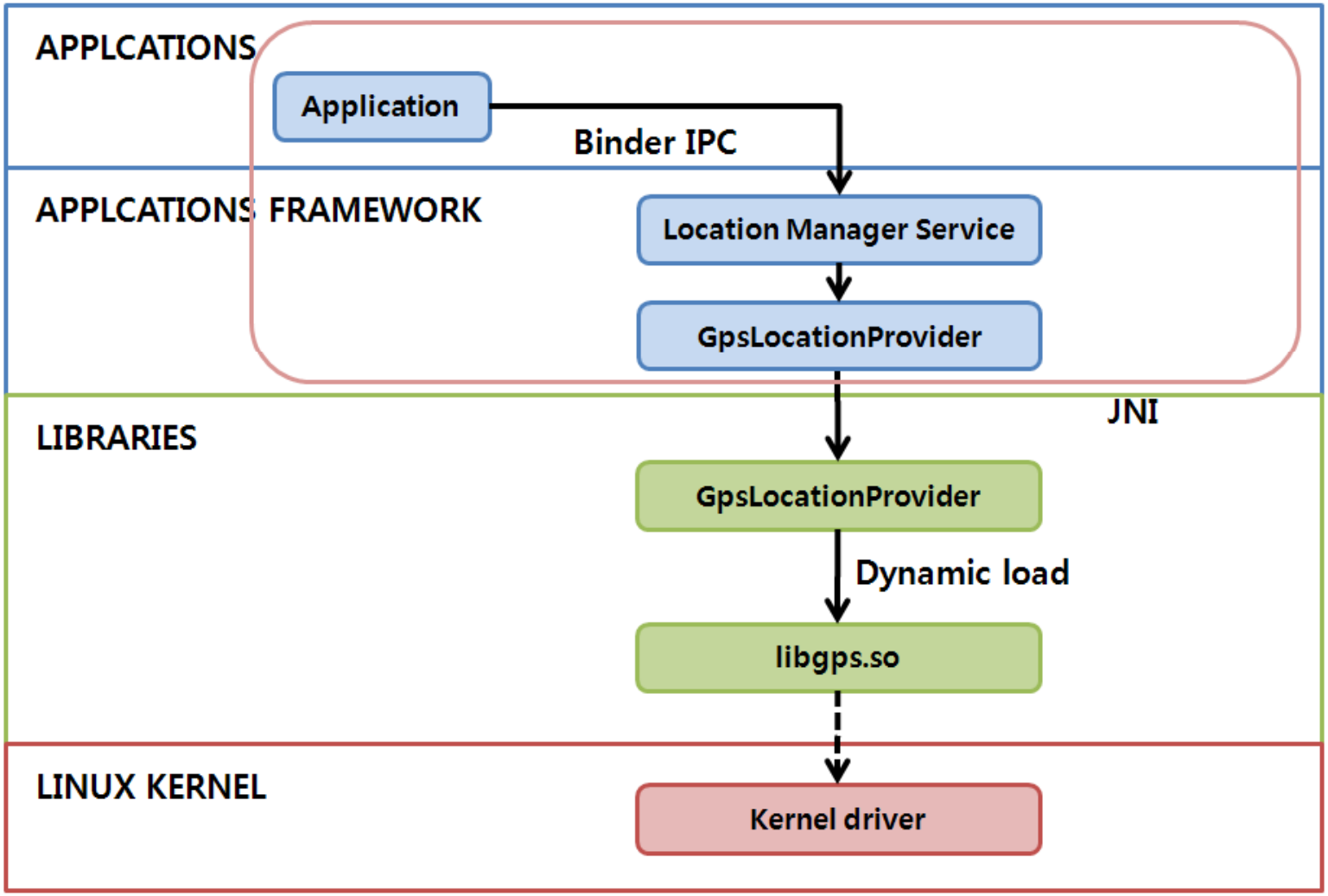
SDK를 이용한 개발

- SDK를 이용한 개발 – 주로 Application의 경우 이 방법 사용
  - ✓ Android Emulator와 SDK를 이용해서 개발



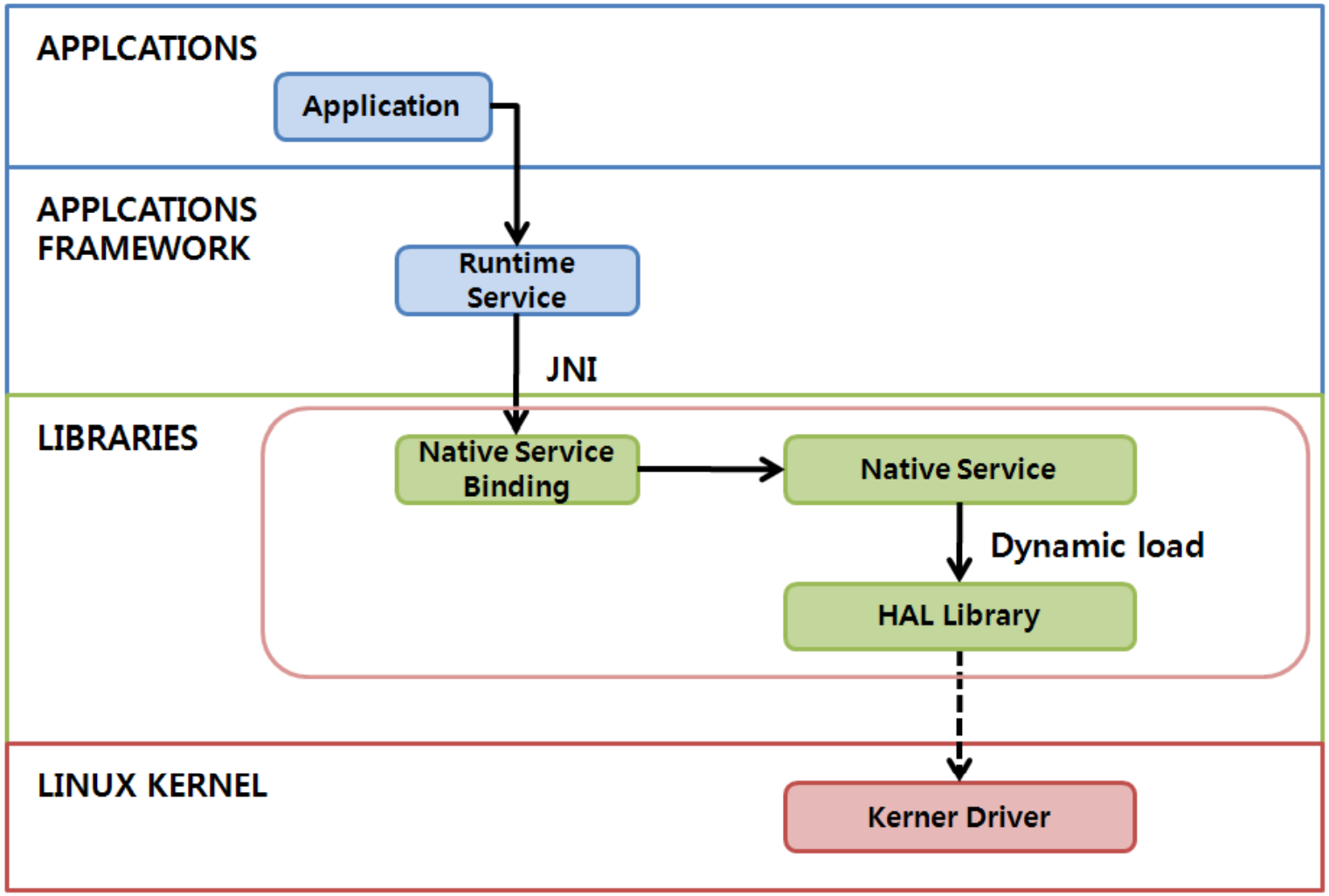
SDK를 이용한 개발

- GPS의 예



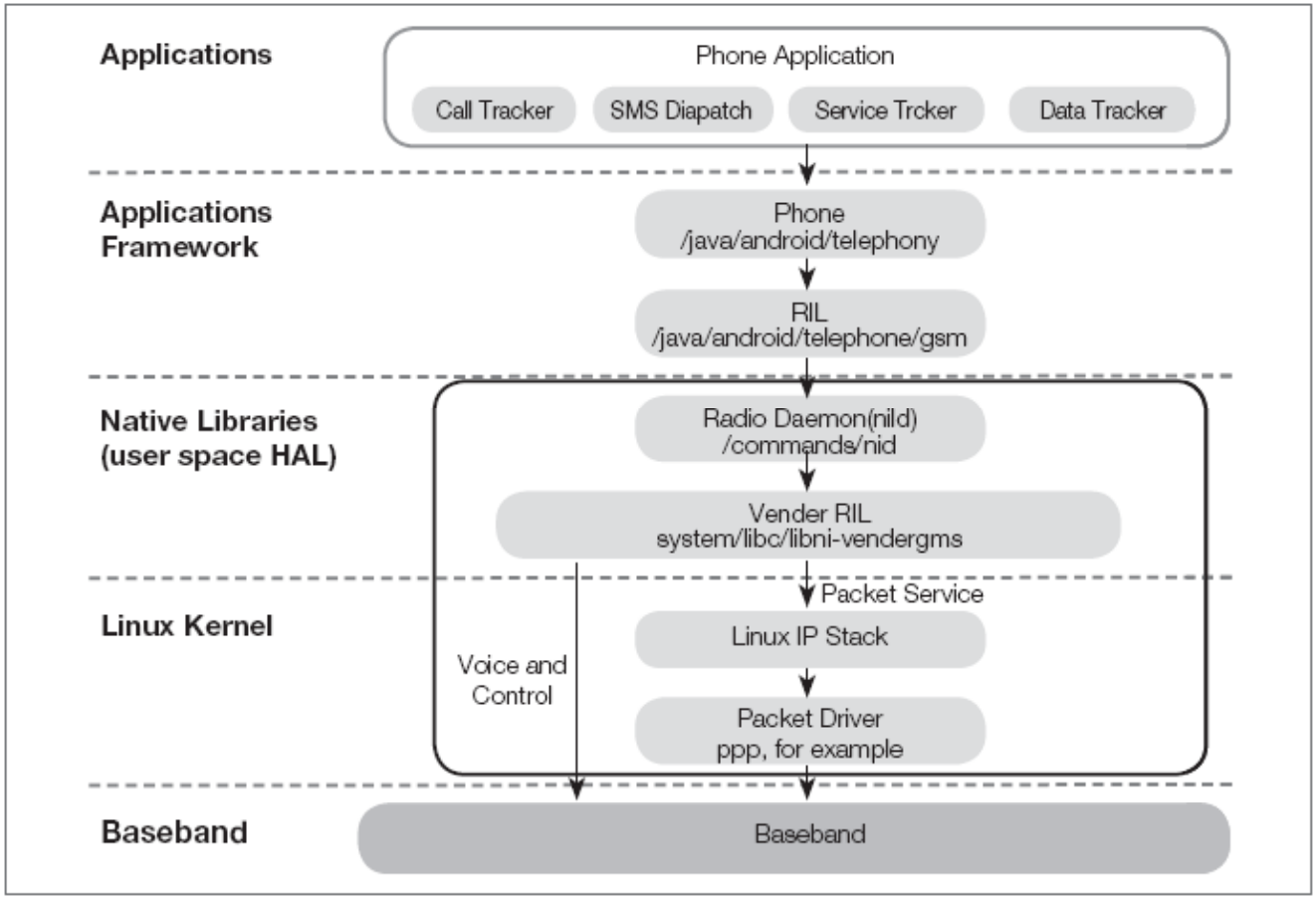
PDK를 이용한 개발 – (1)

- PDK(Platform Development Kit)를 이용한 개발 방법 1
  - ✓ Android Porting의 개념: Android 전체 소스를 가지고 개발
  - ✓ 주로 Android 소스만으로 이루어진 서비스 개발시 사용



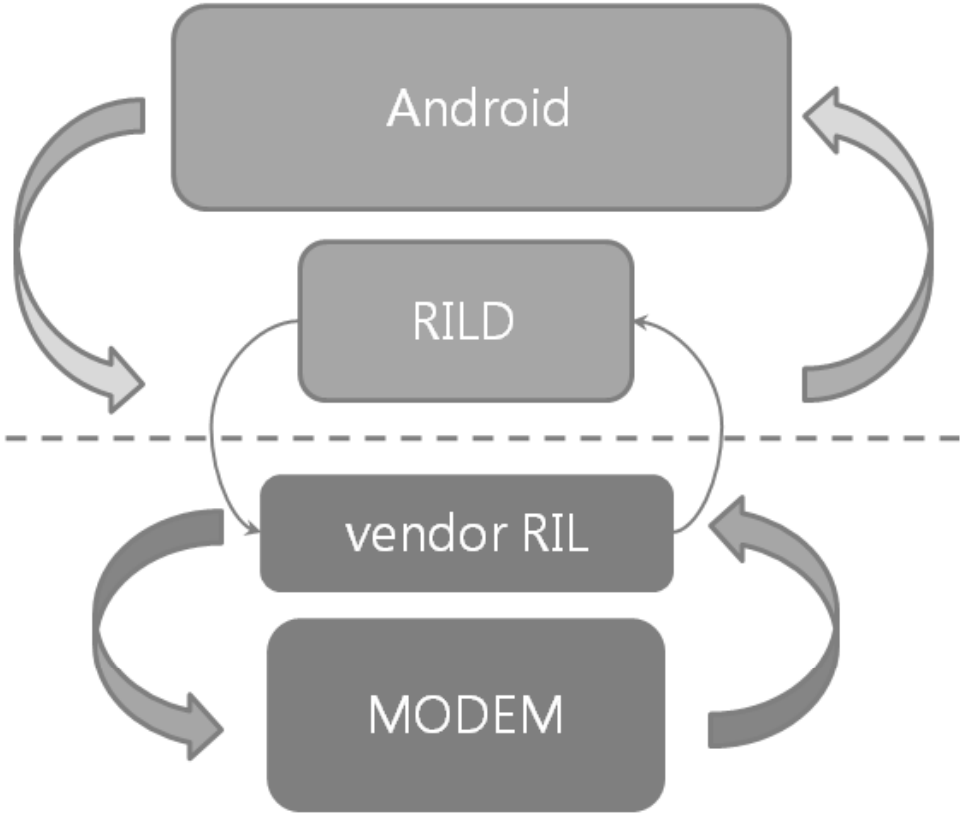
PDK를 이용한 개발 – (2)

- **PDK(Platform Development Kit)를 이용한 개발 방법 2**
  - ✓ **Android Porting의 개념: Android 전체 소스를 가지고 개발**
  - ✓ **기존의 Linux용 Daemon 프로그램을 Android가 이용하는 방법**
    - ➢ ***RILD(Radio Interface Layer Daemon), WiFi, Bluetooth***



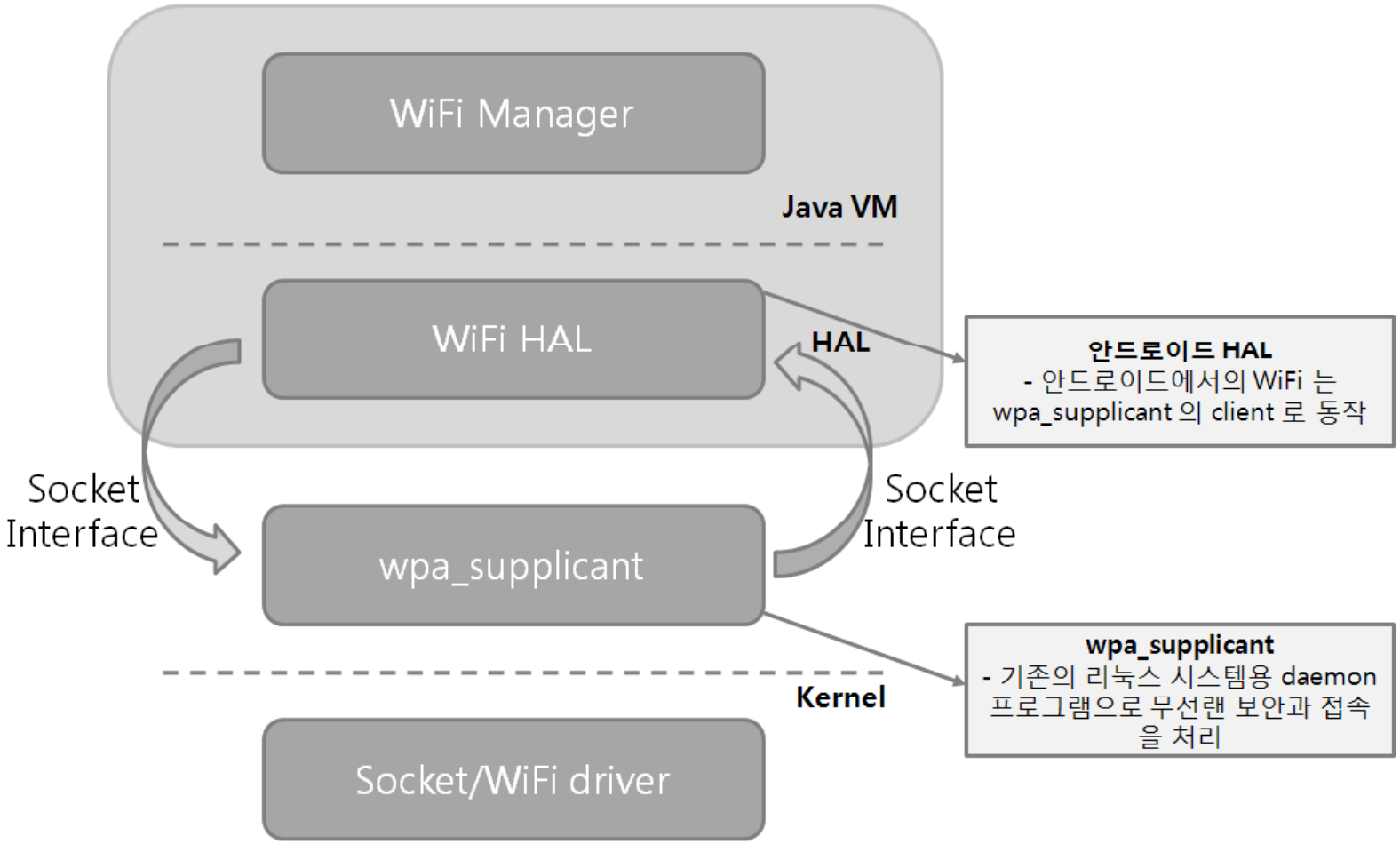
PDK를 이용한 개발 – (2)

- **RILD의 예**
  - ✓ **RILD와 Vendor RIL과의 관계**



PDK를 이용한 개발 – (2)

- **Wifi의 예**



NDK를 이용한 개발

- NDK(Native Development Kit)를 이용한 개발 방법
  - ✓ Application을 개발할 때 사용 – 기존의 C/C++ 라이브러리 이용
  - ✓ Library는 C/C++로 사용, Application개발은 SDK사용