Android多核心嵌入式多媒體系統設計與實作

Android Hardware abstraction layer

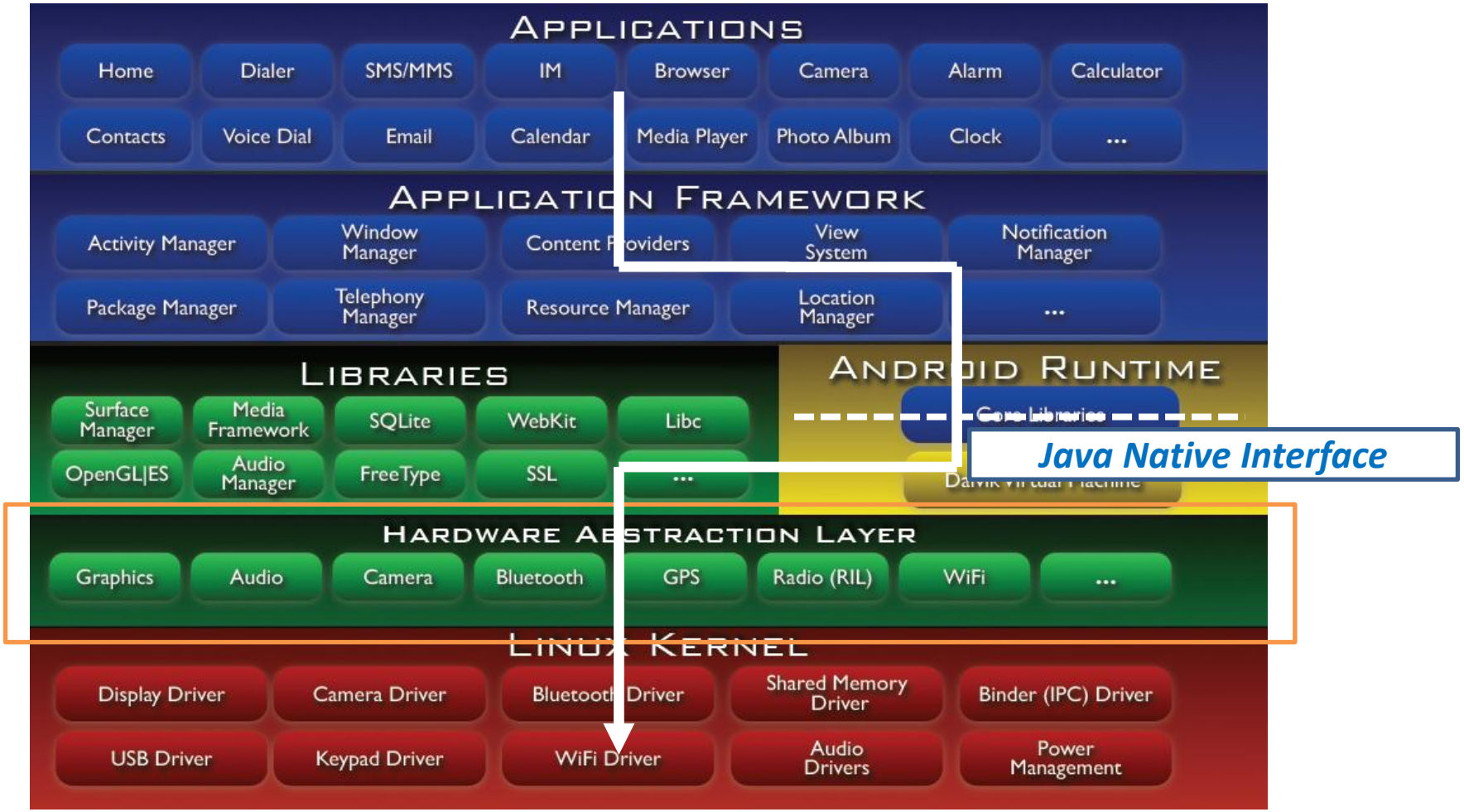
賴槿峰 (Chin-Feng Lai)
Assistant Professor, institute of CSIE, National Ilan University
Nov. 10th 2011

Outline

- Hardware Abstraction Layer Introduction
- Hardware Native Development Kit
- Android Native Server
- LAB : Run Native Application on Android

- ***Hardware Abstraction Layer Introduction***
- ***Hardware Native Development Kit***
- ***Android Native Server***
- ***LAB : Run Native Application on Android***

Hardware Abstraction Layer Introduction



Hardware Abstraction Layer Introduction

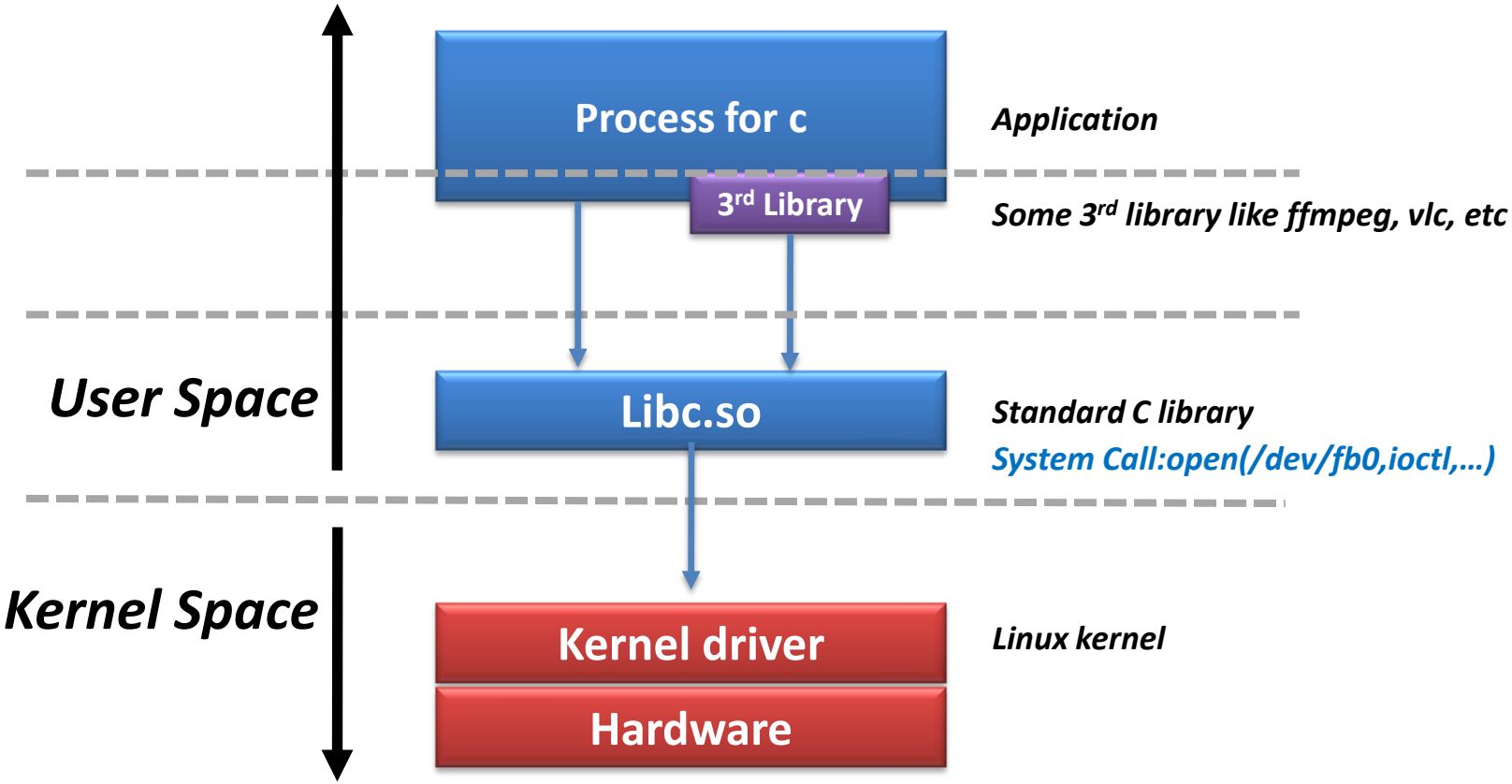
- Mentioned in Google I/O,2008
- HAL is used to separate :
  - Android framework
  - Linux kernel
- Define the hardware control interface
- Not having a standard format in Android development

Hardware Abstraction Layer Introduction

- what should we be concerned about user space and kernel space on android base on Linux kernel
  - general Linux operation system
    - Include standard lib.so
    - Dynamic library: *.so
    - Static Library: *.a
  - Android system
    - Legacy android Hardware Abstraction Layer
      - ***Define on <android_source>/hardware/libhardware_legacy***
    - HAL Stub android Hardware Abstraction Layer
      - ***Define on <android_source>/hardware/libhardware***

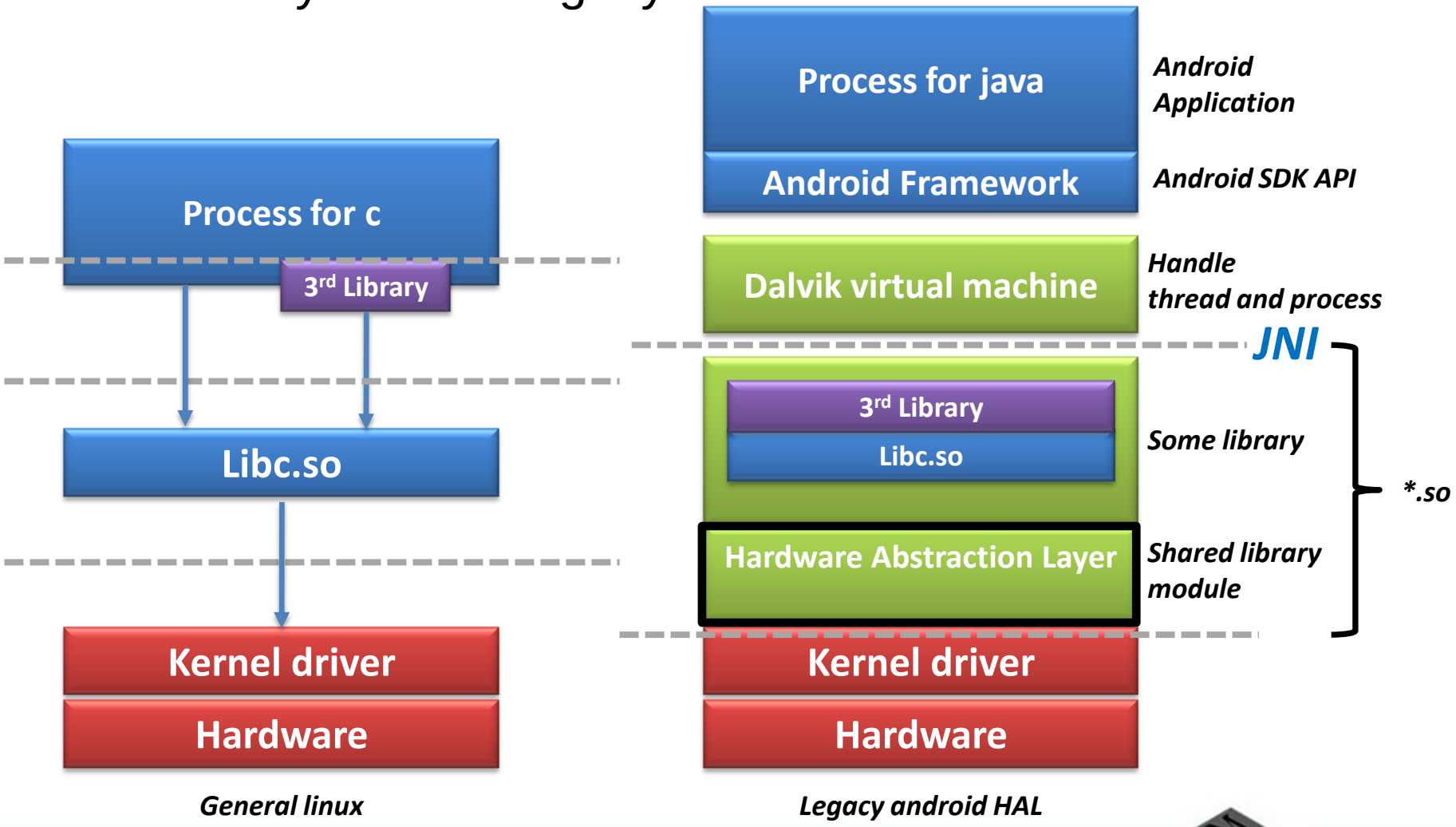
Hardware Abstraction Layer Introduction

- General Linux operation system
  - Use c code



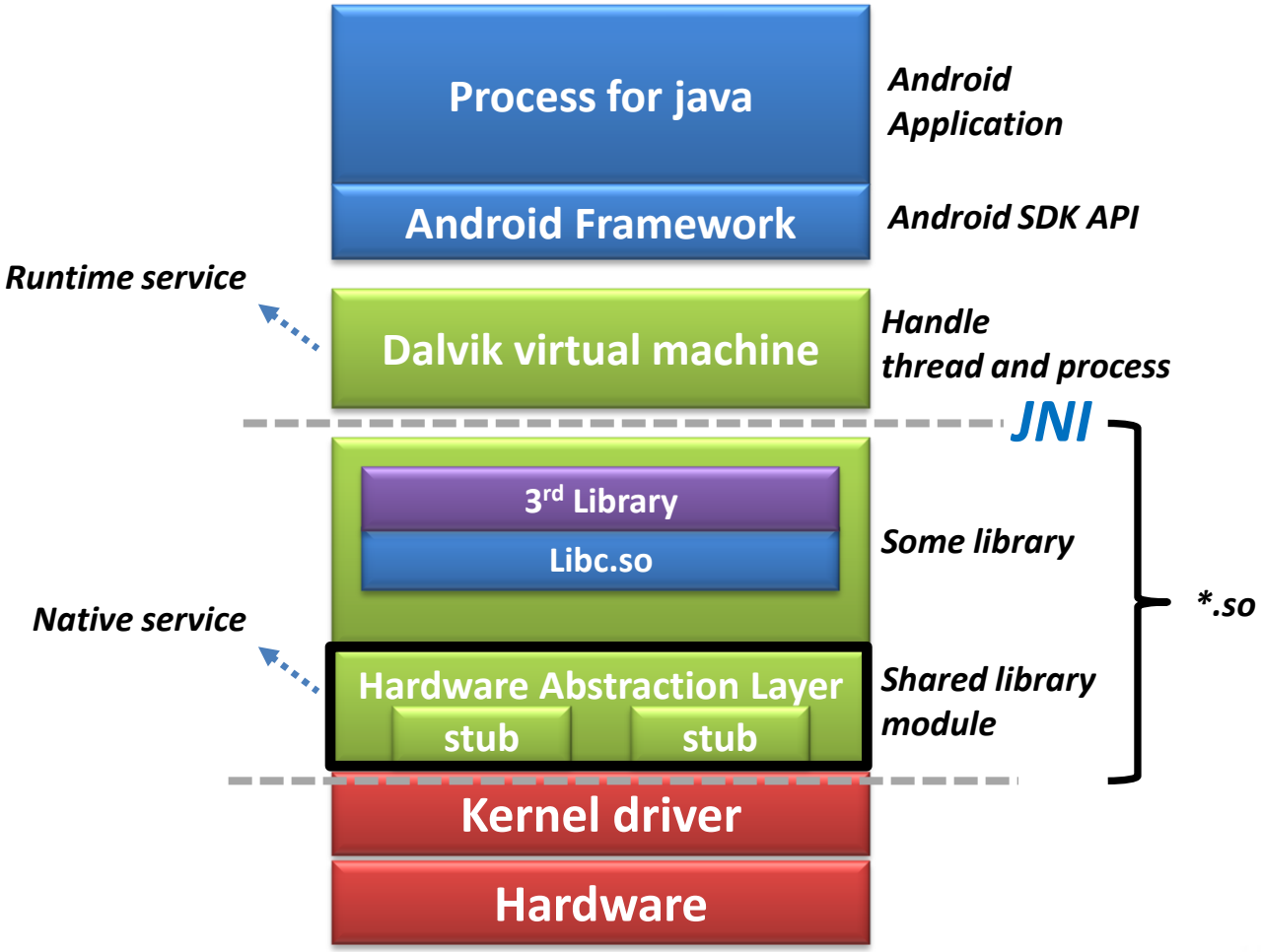
Hardware Abstraction Layer Introduction

- Android System for legacy



Hardware Abstraction Layer Introduction

- Android System for HAL Stub



Hardware Abstraction Layer Introduction

- Android Layer Analysis

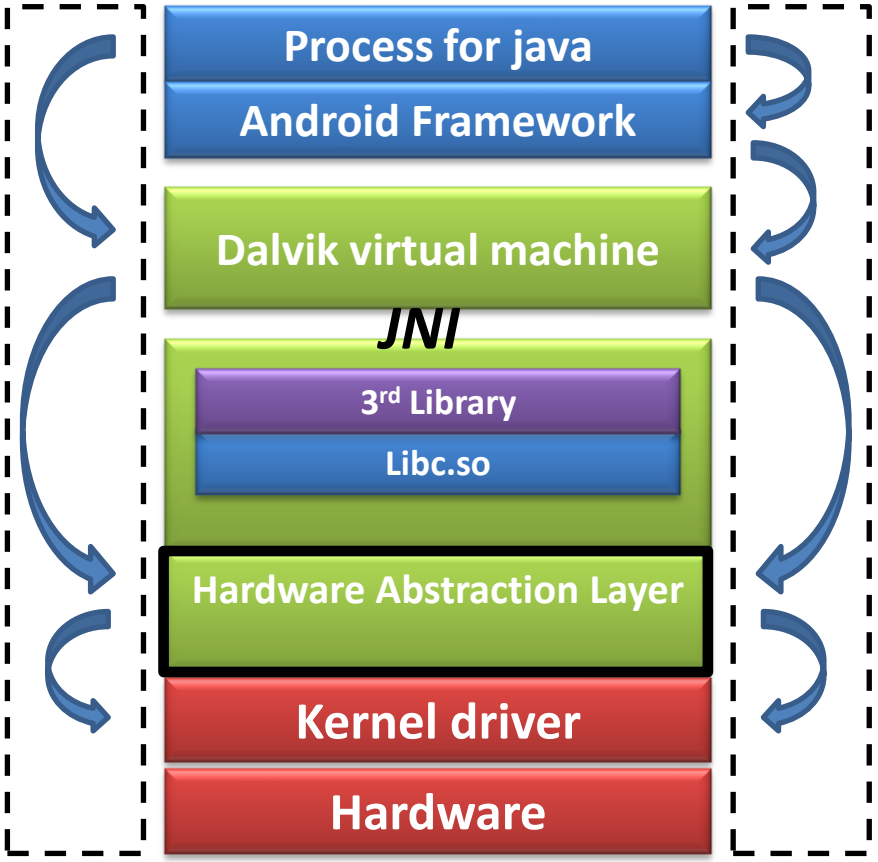
Layer	Language	Form	Ship
Application	JAVA	*.apk	*.apk/system.img
Framework	JAVA	*.jar	system.img
Libraries	C/C++	*.so	system.img
HAL	C/C++	*.so	system.img
Kernel	C/asm	*.ko	uImage

Hardware Abstraction Layer Introduction

- Legacy android Hardware Abstraction Layer
  - Define on <android_source>/hardware/libhardware_legacy
  - The controlling hardware library compiler to *.so file which will be used as ***shared library***
  - java call directly
- HAL Stub android Hardware Abstraction Layer
  - Define on <android_source>/hardware/libhardware
  - Use ***HAL module*** to direct function call
  - libhardware/include
    - » Design interface, harader file
  - libhardware/module
    - » Design reuse, override
  - harware.c
    - » Load(), hw_get_module()
  - libhardware/include/hardware/hardware.h
    - » Some structure

Hardware Abstraction Layer Introduction

- Legacy android Hardware Abstraction Layer
  - many methods



Hardware Abstraction Layer Introduction

- Example: Power Control In android
  - Reference to http://developer.android.com/reference/android/os/PowerManager.html
  - There is a set of API for Power Management

Summary

Nested Classes		
class	PowerManager.WakeLock	Class lets you say that you need to have the device on.

Constants		
int	ACQUIRE_CAUSES_WAKEUP	Normally wake locks don't actually wake the device, they just cause it to remain on once it's already on.
int	FULL_WAKE_LOCK	Wake lock that ensures that the screen and keyboard are on at full brightness.
int	ON_AFTER_RELEASE	When this wake lock is released, poke the user activity timer so the screen stays on for a little longer.
int	PARTIAL_WAKE_LOCK	Wake lock that ensures that the CPU is running.
int	SCREEN_BRIGHT_WAKE_LOCK	Wake lock that ensures that the screen is on at full brightness; the keyboard backlight will be allowed to go off.
int	SCREEN_DIM_WAKE_LOCK	Wake lock that ensures that the screen is on (but may be dimmed); the keyboard backlight will be allowed to go off.

Public Methods	
void	goToSleep (long time) Force the device to go to sleep.
boolean	isScreenOn () Returns whether the screen is currently on.
PowerManager.WakeLock	newWakeLock (int flags, String tag) Get a wake lock at the level of the flags parameter.

Hardware Abstraction Layer Introduction

- Example: Power Control In android
- Using the Power Management API
  - In eclipse

```
mButtonw01.setOnClickListener(new Button.OnClickListener()
{
  public void onClick(View arg0)
  {
      mPowerManager=(PowerManager)getSystemService(Context.POWER_SERVICE);
      mPowerManager.goToSleep(5000);

      text.setText("diousk sleep!");
  }
});
```


Hardware Abstraction Layer Introduction

- Example: Power Control In ***android framework***
- Trace the power management code:
  - Android/frameworks/base/core/java/android/os/PowerManager.java

```
public void goToSleep(long time)
{
    try {
        mService.goToSleep(time);
    } catch (RemoteException e) {
    }
}
```

Hardware Abstraction Layer Introduction

- Example: Power Control In ***android runtime service***
- Trace the power management code:
  - Android/frameworks/base/services/java/com/android/server/PowerManagerService.java

goToSleep ()-> goToSleepWithReaso() -> goToSleepLocked() -> setPowerState() -> updateNativePowerStateLocked();

```
private void updateNativePowerStateLocked() {
    nativeSetPowerState(
        (mPowerState & SCREEN_ON_BIT) != 0,
        (mPowerState & SCREEN_BRIGHT) == SCREEN_BRIGHT);
}
```


Hardware Abstraction Layer Introduction

- Example: Power Control In ***JNI Table***
- Trace the power management code:
  - Android/frameworks/base/services/services/jni/***com_android_server_PowerManagerService.cpp***

```
static JNINativeMethod gPowerManagerServiceMethods[] = {
   /* name, signature, funcPtr */
   { "nativeInit", "()V“,  (void*) android_server_PowerManagerService_nativeInit },
   { "nativeSetPowerState", "(ZZ)V“,   (void*) android_server_PowerManagerService_nativeSetPowerState },
   { "nativeStartSurfaceFlingerAnimation", "(I)V“,  (void*)
android_server_PowerManagerService_nativeStartSurfaceFlingerAnimation },
};
```

```
 static void android_server_PowerManagerService_nativeSetPowerState(JNIEnv* env, jobject serviceObj,
jboolean screenOn, jboolean screenBright) {
   set_screen_stage(on);
}
```

Android/hardware/libhardware_legacy/power/power.c(HAL)

Hardware Abstraction Layer Introduction

- Example: Power Control In ***JNI Table***

Java類型	符號
Boolean	Z
Byte	B
Char	C
Short	S
Int	I
Long	L
Float	F
Double	D
Void	V

Hardware Abstraction Layer Introduction

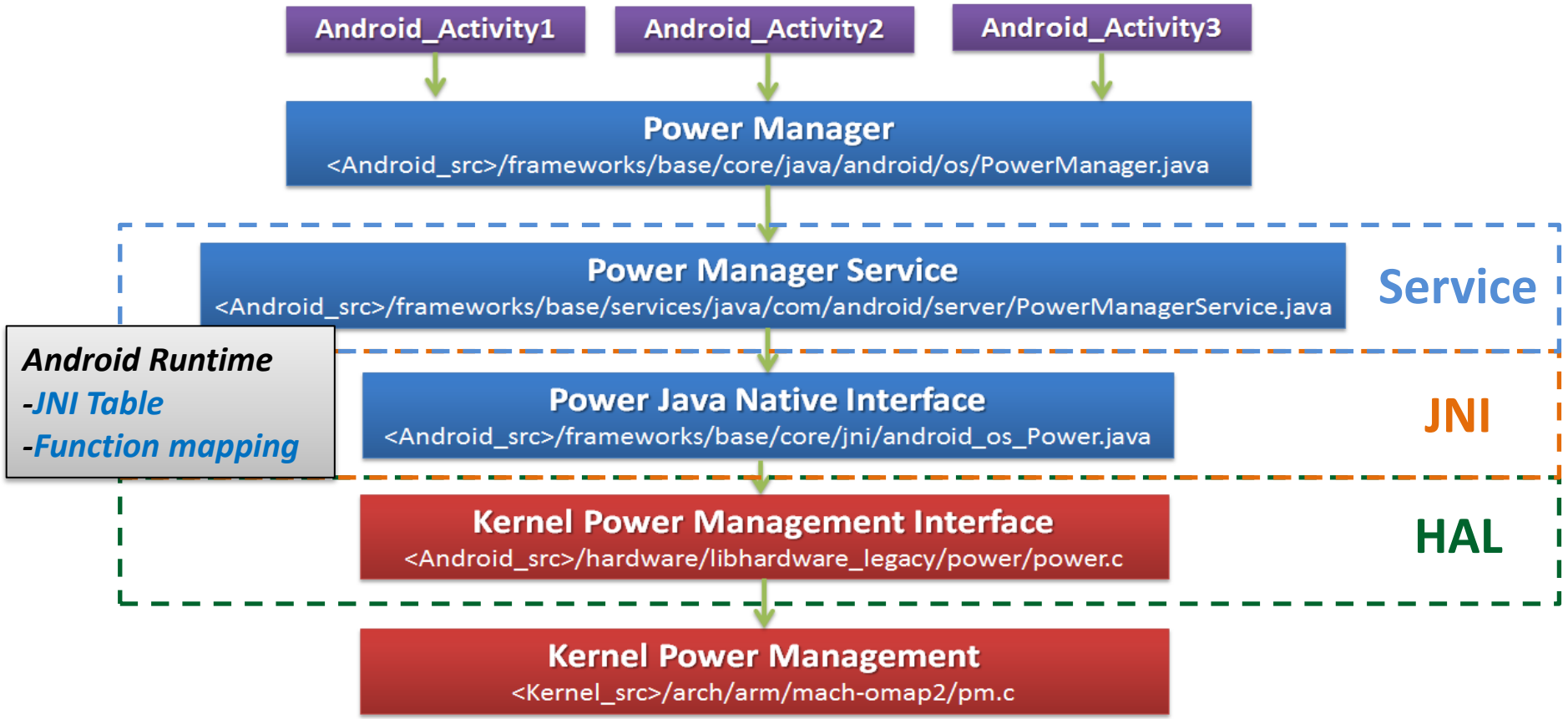
- Example: Power Control In android (Legacy_HAL)

```
<in power.c>
const char * const NEW_PATHS[] = { "/sys/power/wake_lock", "/sys/power/wake_unlock", "/sys/power/state" };

int set_screen_state(int on) {
    ...
    len = write(g_fds[REQUEST_STATE], buf, len);
    if(len < 0) {
        LOGE("Failed setting last user activity: g_error=%d\n", g_error);
    }
    ...
}
```

Android/hardware/libhardware_legacy/power/power.c(HAL)

Hardware Abstraction Layer Introduction

- Example: Power Control In android



Hardware Native Development Kit

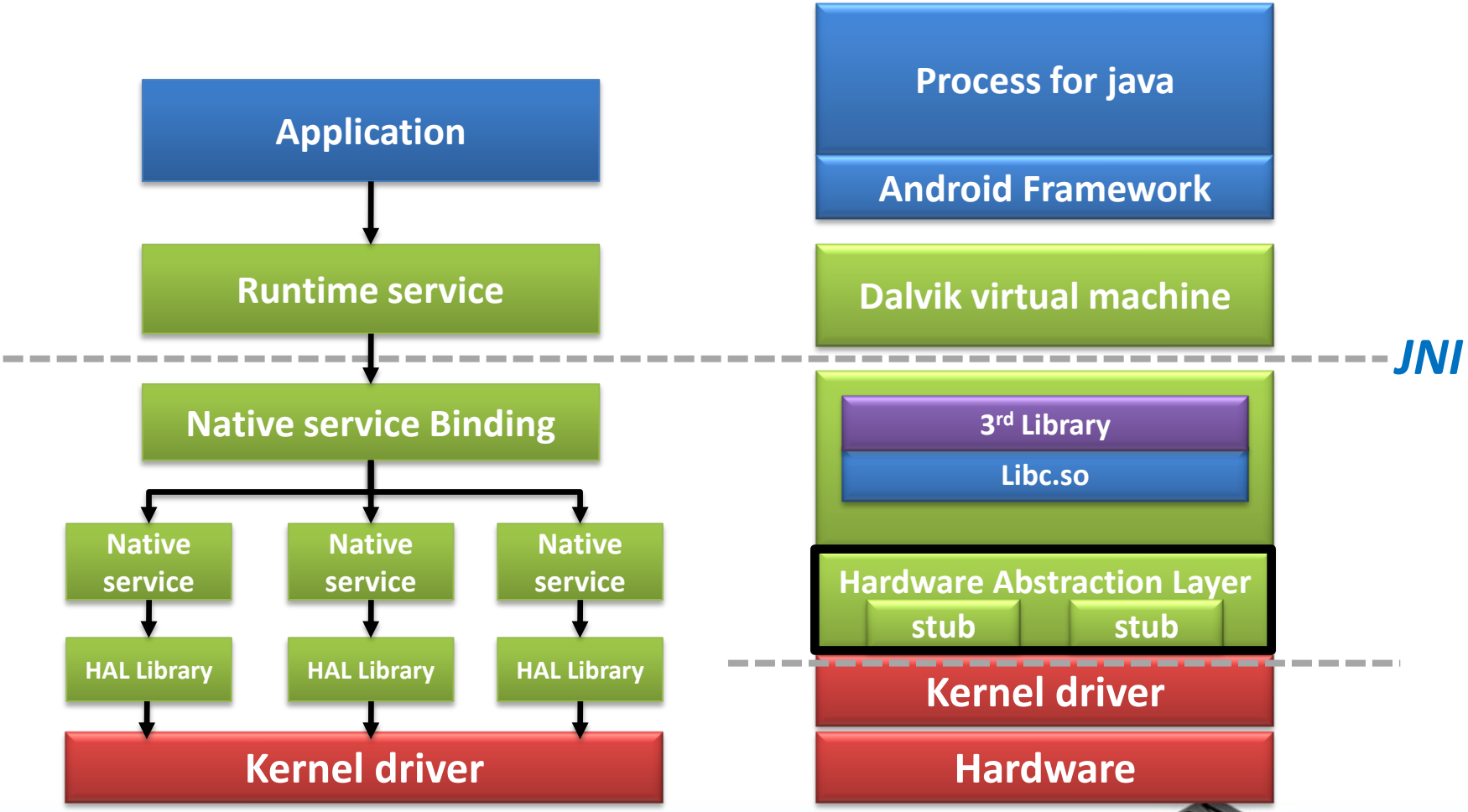
- Use Android Native Development Kit
  - Type conversion from Java to C

JAVA type	C type
jboolean	boolean
jint	int
jlong	long
jdouble	double
jfloat	float
jchar	char
Jstring	string

```
static void android_server_PowerManagerService_nativeSetPowerState(JNIEnv* env, jobject serviceObj,
jboolean screenOn, jboolean screenBright) {
   set_screen_stage(on);
}
```

Hardware Abstraction Layer Introduction

- HAL Stub android Hardware Abstraction Layer
  - Stub provide operations and callbacks of hardware
  - Services use hardware module ID to get information and methods



MMN Lab.

Hardware Abstraction Layer Introduction

- Each hardware must implement the interface in stub format for using in android service
- Stub format
  - defined in android/hardware/libhardware/include/hardware/hardware.h
    - hw_module_t
    - hw_module_method_t
    - hw_device_t
  - defined in android/hardware/libhardware/hardware.c
    - Load()
      - Use dlopen() to load *.so
    - hw_get_module()
      - Get module information and call load() function

Hardware Abstraction Layer Introduction

- Each hardware must implement the interface in stub format for using in android service

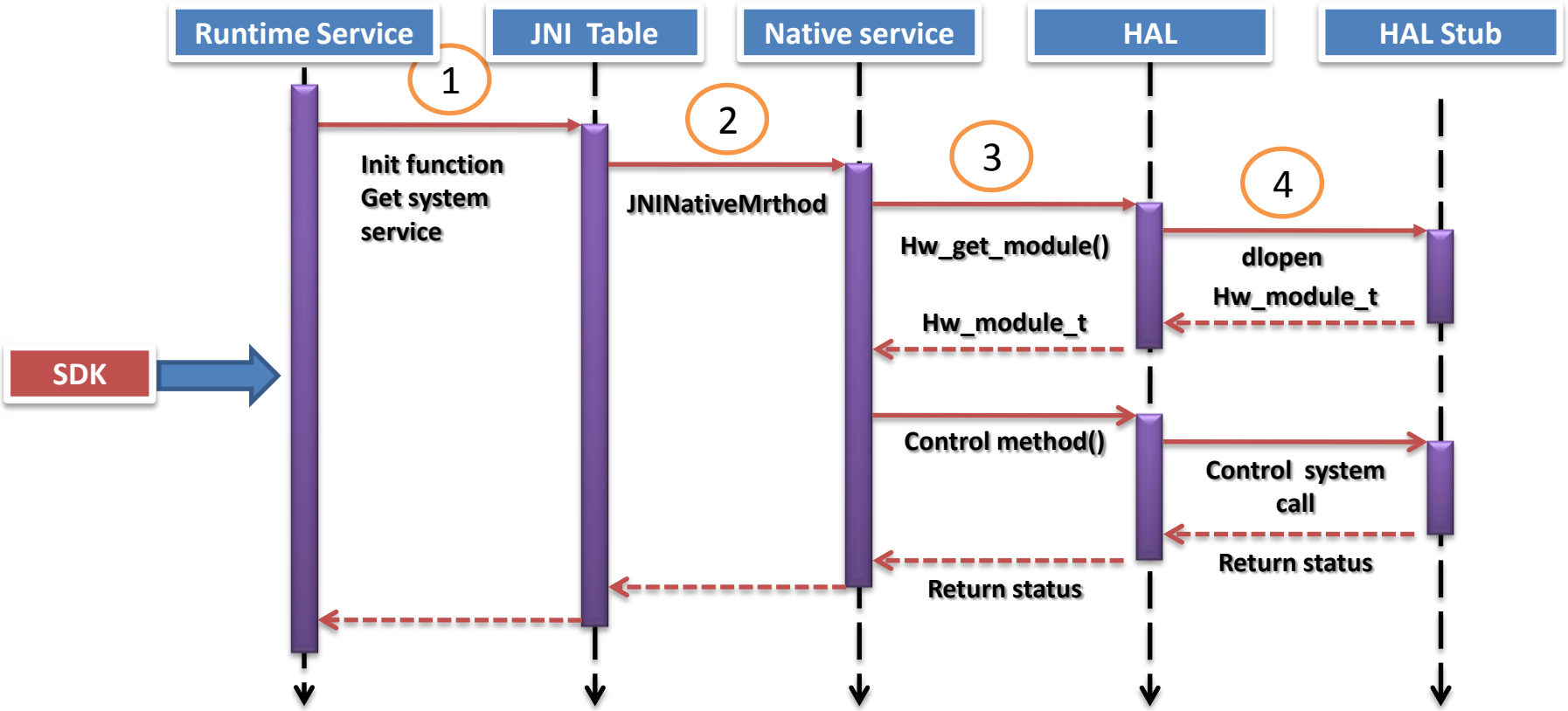
```
typedef struct hw_module_t {
        uint32_t tag;
        uint16_t version_major;
        uint16_t version_minor;
        const char *id;
        const char *name;
        const char *author;
        /** Modules methods */
        struct hw_module_methods_t* methods;
} hw_module_t;
```

```
typedef struct hw_module_methods_t {
   /** Open a specific device */
   int (*open)(const struct hw_module_t* module, const
char* id,
         struct hw_device_t** device);

} hw_module_methods_t;
```

```
typedef struct hw_device_t {
        uint32_t tag;
        uint32_t version;
        struct hw_module_t* module;
        uint32_t reserved[12];
        int (*close)(struct hw_device_t* device);
} hw_device_t;
```


Hardware Abstraction Layer Introduction

- HAL Stub android Hardware Abstraction Layer



MMN Lab.

Hardware Abstraction Layer Introduction

- runtime service
  - Java code
  - Define on <android_source>/framework/base/services/java/com/android/server
  - Define "***private static native***"

```
private native void nativeInit();
private native void nativeSetPowerState(boolean screenOn, boolean screenBright);
private native void nativeStartSurfaceFlingerAnimation(int mode);
```

- JNI Table

```
static JNINativeMethod gPowerManagerServiceMethods[] = {
   /* name, signature, funcPtr */
   { "nativeInit", "()V",  (void*) android_server_PowerManagerService_nativeInit },
   { "nativeSetPowerState", "(ZZ)V",   (void*) android_server_PowerManagerService_nativeSetPowerState },
   { "nativeStartSurfaceFlingerAnimation", "(I)V",  (void*)
android_server_PowerManagerService_nativeStartSurfaceFlingerAnimation },
};
```

<android_source>/framework/base/services/jni
<android_source>/framework/base/core/jni

Hardware Abstraction Layer Introduction

- Native service

```
static android_server_PowerManagerService_nativeInit(JNIEnv *env, jclass clazz)
{
        module_t const * module;
        hw_get_module(HARDWARE_MODULE_ID, (const hw_module_t**)&module);
        return 0;
}
```

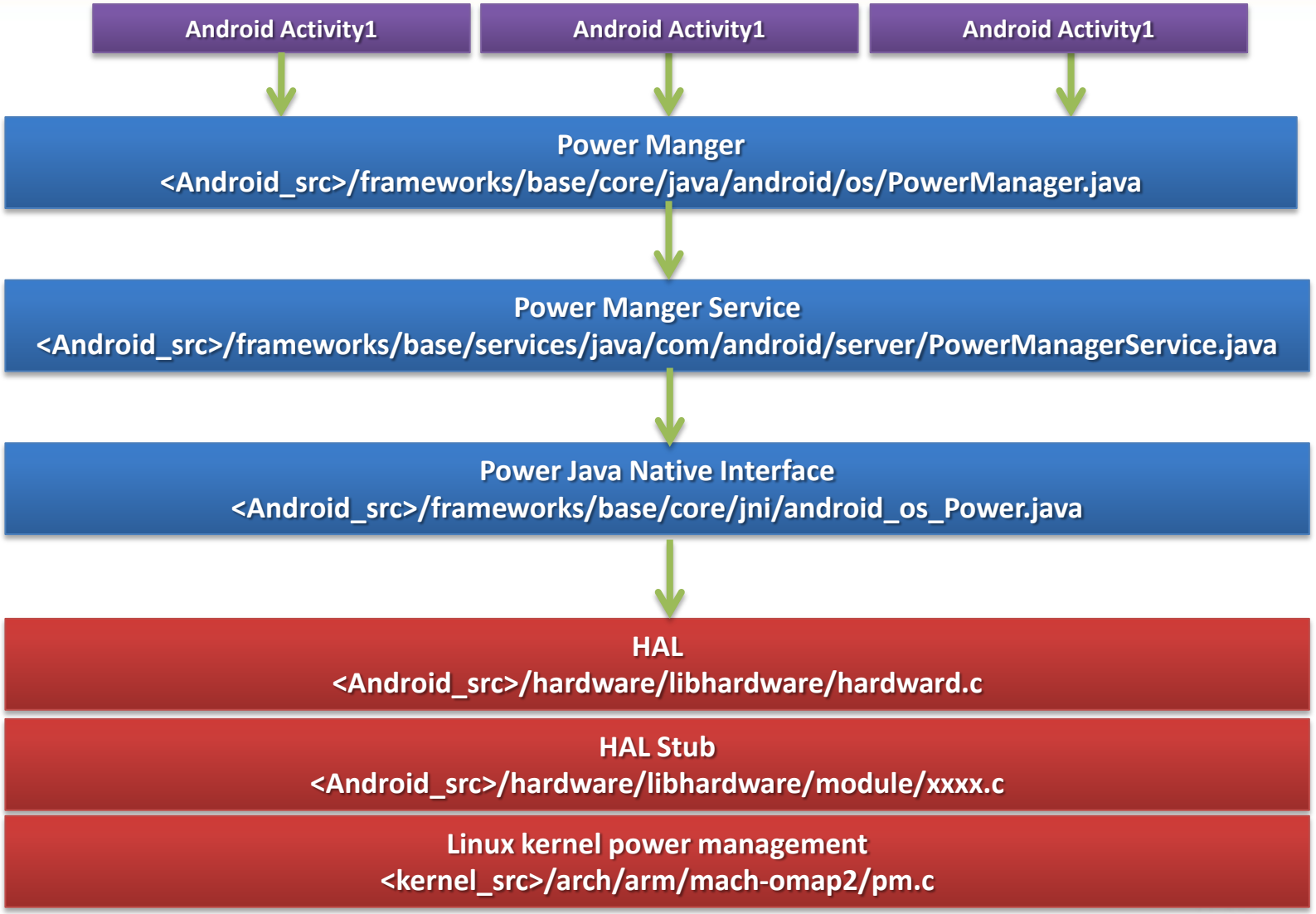
<android_source>/framework/base/services/jni
<android_source>/framework/base/core/jni

- Hal Stub

```
int hw_get_module(const char *id, const struct hw_module_t **module)
{
         status = load(id, path, module);
        return status;
}
```

<android_source>/hardware/libhardware/hardwar.c

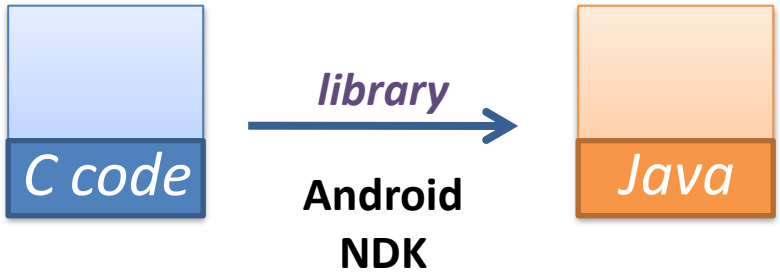
Hardware Abstraction Layer Introduction



- *Hardware Abstraction Layer Introduction*
- ***Hardware Native Development Kit***
- *Android Native Server*
- *LAB : Run Native Application on Android*

Hardware Native Development Kit

- Android Native Development Kit
  - Android NDK is a companion tool to the Android SDK that lets you build performance-critical portions of your apps in native code



Hardware Native Development Kit

- NDK
  - Generate JNI dynamic Libraries (*.so)
  - Put the file to correct path on android filesystem
- A set of tools and build files used to generate native code libraries from C and C++ sources
- way to embed the corresponding native libraries into an application package file
- The latest release of the NDK supports these ARM instruction sets
  - ARMv5TE
  - ARMv7-A
  - x86 instructions

Hardware Native Development Kit

- Download Android NDK
- http://developer.android.com/sdk/ndk/index.html

Download the Android NDK

The Android NDK is a companion tool to the Android SDK that lets you build performance-critical portions of your apps in nati
you to build activities, handle user input, use hardware sensors, access application resources, and more, when programming
are still packaged into an .apk file and they still run inside of a virtual machine on the device. The fundamental Android applica

Using native code does not result in an automatic performance increase, but always increases application complexity. If you l
framework APIs, you probably do not need the NDK. Read What is the NDK? for more information about what the NDK offers

The NDK is designed for use *only* in conjunction with the Android SDK. If you have not already installed and setup the Androic

Platform	Package	Size	MD5 Checksum
Windows	android-ndk-r5b-windows.zip	61299831 bytes	87745ada305ab639399161ab4faf684c
Mac OS X (intel)	android-ndk-r5b-darwin-x86.tar.bz2	50210863 bytes	019a14622a377b3727ec789af6707037
Linux 32/64-bit (x86)	android-ndk-r5b-linux-x86.tar.bz2	44138539 bytes	4c0045ddc2bfd657be9d5177d0e0b7e7

Hardware Native Development Kit

- NDK
  - The NDK includes a set of cross-toolchains and sample application
  - Use Android,mk

```
.
|-- GNUmakefile
|-- README.TXT
|-- RELEASE.TXT
|-- build
|-- docs
|-- documentation.html
|-- ndk-build
|-- ndk-gdb
|-- ndk-stack
|-- platforms
|-- samples
|-- sources
|-- tests
|-- tmp
`-- toolchains
```


Hardware Native Development Kit

- Develop in NDK
  1. Set Env “PATH”, “NDK_ROOT”, “NDK_Sample”
  2. Edit your JNI code on android-ndk-r6/samples/hello-jni
  3. Edit Android.mk file on android-ndk-r6/samples/hello-jni
     - LOCAL_PATH
     - LOCAL_MODULE
     - LOCAL_SRC_FILES
     - LOCAL_LDLIBS
     - BUILD_SHARED_LIBRARY
  4. Host$ ***ndk-build***

```
Gdbserver      : [arm-linux-androideabi-4.4.3] libs/armeabi/gdbserver
Gdbsetup       : libs/armeabi/gdb.setup
Compile thumb  : hello-jni <= hello-jni.c
SharedLibrary  : libhello-jni.so
Install        : libhello-jni.so => libs/armeabi/libhello-jni.so
```

  5. Generate so file on “android-ndk-r6/samples/hello-jni/libs/armeabi”

Hardware Native Development Kit

- All parameters to 'ndk-build‘
  - ndk-build
    - rebuild required machine code.
  - ndk-build clean
    - clean all generated binaries.
  - ndk-build V=1
    - launch build, displaying build commands.
  - ndk-build NDK_DEBUG=1
    - generate debuggable native code.
- On java code

```
static {
    System.loadLibrary("hello-jni");
}
```


- *Hardware Abstraction Layer Introduction*
- *Hardware Native Development Kit*
- ***Android Native Server***
- *LAB : Run Native Application on Android*

Android Native Server

- The android native service are defined in <android_rootfs>/init.rc
- init.rc includes the startup services which run in android background
  - ex: mediaserver、network daemon 、bluetooth daemon、adb daemon…etc.

Android Native Server

- The init.rc file will be loaded by <android_fs>/init as filesystem mounted
- This is why the bootargs is like:

```
bootargs=console=ttyS2,115200n8 noinitrd video=omapfb:mode:4.3inch_LCD root=/dev
/nfs ip=192.168.0.10:192.168.0.209::::eth0:off init=/init rw rootwait mem=99M om
ap_vout.vid1_static_vrfb_alloc=y omap_vout.video1_numbuffers=3 omap_vout.video1_
bufsize=829440 omap_vout.video2_numbuffers=0 nfsroot=192.168.0.209:/home/diousk/
omap_tools/android_fs
```


Android Native Server

- How to add service in init.rc ?
- Follow Init.rc format
  - Actions
  - Commands
  - Services
  - Options

Android Native Server

- Actions
  - on <trigger>
  - <command>
  - <command>
- EX:
  - on boot
  - setprop persist.sys.keylayout gpio-keys
  - mkdir /data/misc/dhcp 0770 dhcp dhcp
  - chmod 0770 /data/misc/dhcp

Android Native Server

- Triggers
  - Boot
  - <name>=<value>
  - service-exited-<name>
  - device-added-<path>
  - device-removed-<path>

Android Native Server

- Commands
  - setprop <name> <value>
  - trigger <event>
  - chmod <octal-mode> <path>
  - export <name> <value>
  - class_start <serviceclass>
  - class_stop <serviceclass>
  - mkdir <path> [mode] [owner] [group]
  - start <service>
  - stop <service>
  - insmod <modules>

Android Native Server

- Services
  - service <name> <pathname> [<argument>]*
  - <option>
  - <option>
- EX:
  - service sgx /system/etc/init.sgx.sh
  - oneshot

Android Native Server

- Options
  - critical
  - user <username>
  - oneshot
  - class <name>
  - onrestart <command>

Android Native Server

- Actual init.rc in <android_fs>/init.rc

```
on init

sysclktz 0

loglevel 3

# setup the global environment
    export PATH /sbin:/system/sbin:/system/bin:/system/xbin
    export LD_LIBRARY_PATH /system/lib
    export ANDROID_BOOTLOGO 1
    export ANDROID_ROOT /system
    export ANDROID_ASSETS /system/app
    export ANDROID_DATA /data
    export EXTERNAL_STORAGE /sdcard
    export BOOTCLASSPATH /system/framework/core.jar:/system/framework/ext.jar:/sys
tem/framework/framework.jar:/system/framework/android.policy.jar:/system/framework
/services.jar

# Backward compatibility
    symlink /system/etc /etc
    symlink /sys/kernel/debug /d

# create mountpoints and mount tmpfs on sqlite_stmt_journals
insmod dvb-core.ko
insmod dib0070.ko
insmod dib3000mb.ko
insmod dibx000_common.ko
insmod dib3000mc.ko
insmod dib7000m.ko
insmod dib7000p.ko
```

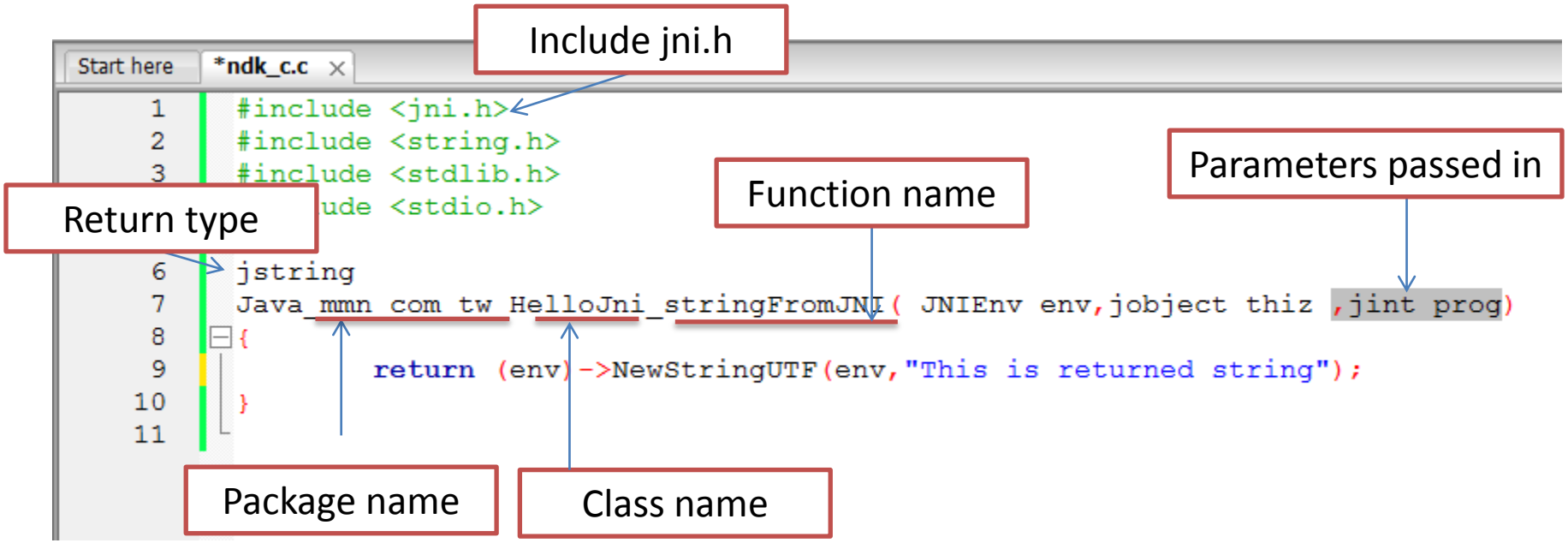

- *Android Framework Review*
- *Hardware Abstraction Layer Introduction*
- *Power Control Example*
- *In Progress HAL*
- *Android Native Development Kit*
  - *Android Application Issue*
  - *Android Native Services (init.rc)*
- ***LAB : Run Native Application on Android***

Lab files

- *socket.tar.gz
  - Client.c
    - Reference to this C code and build as library via NDK
  - Server.c
    - Build as an executable binary and define in init.rc
- android_ndk.tar.gz
  - android-ndk-1.5_r1

Lab : Run Native Application on Android

- Use Android Native Development Kit
  - Write a C code to be built as library



MMN Lab.

Lab : Run Native Application on Android

- Use Android Native Development Kit
  - Android.mk

/android-ndk-1.5_r1/sources/samples/helloqq/Android.mk

code

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := helloqq      //library name
LOCAL_SRC_FILES := helloqq.c    //jni native code

include $(BUILD_SHARED_LIBRARY)
```

  - Application.mk

/android-ndk-1.5_r1/apps/helloqq/Application.mk

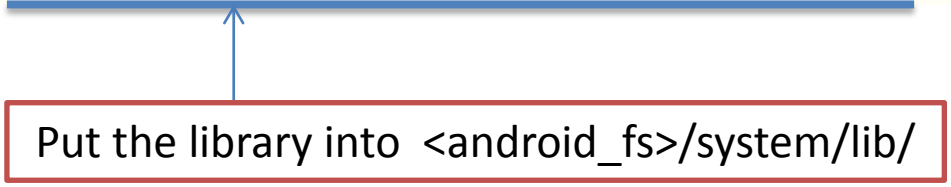
code

```
APP_PROJECT_PATH := $(call my-dir)/project
APP_MODULES      := helloqq //和Android.mk對應
```

Lab : Run Native Application on Android

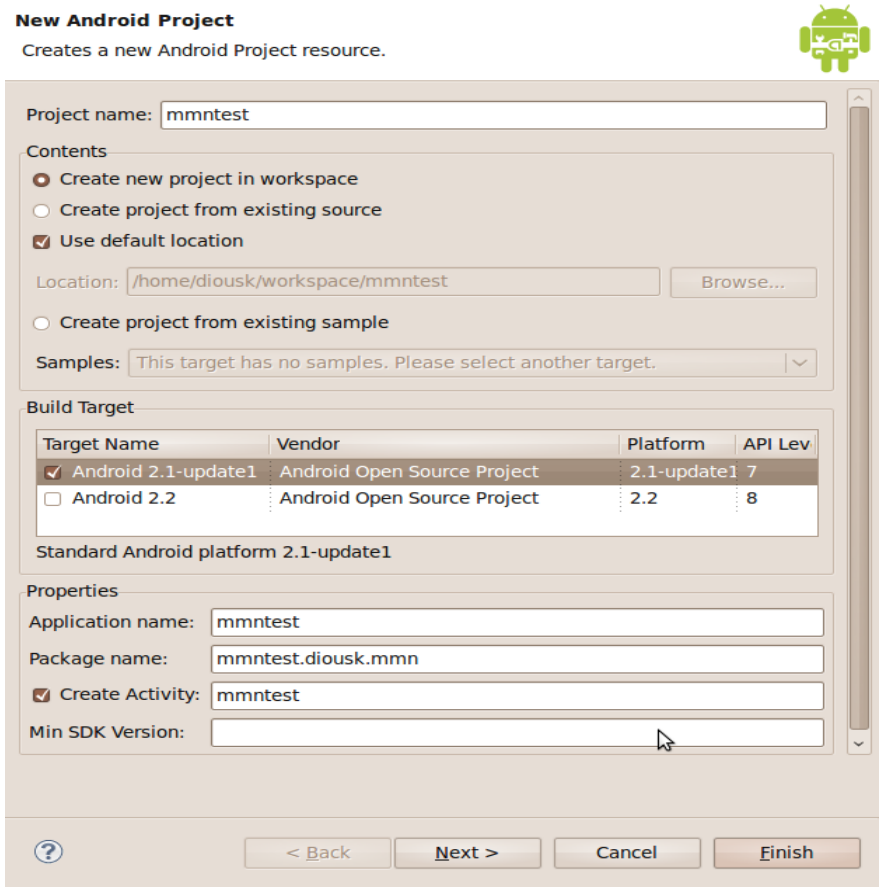
- Use Android Native Development Kit
  - Build the library

```
make APP=helloqq
Android NDK: Building for application 'helloqq'
Compile thumb  : helloqq <= sources/samples/helloqq/helloqq.c
sources/samples/helloqq/helloqq.c: In function 'socket_client':
sources/samples/helloqq/helloqq.c:88: warning: function returns address of local variable
SharedLibrary  : libhelloqq.so
Install        : libhelloqq.so => apps/helloqq/project/libs/armeabi
```



Lab : Run Native Application on Android

- In eclipse, File >> New >> Android Project



MMN Lab.

Lab : Run Native Application on Android

- Use Android Native Development Kit
  - Use in Apk source

MMN Lab.

Lab : Run Native Application on Android

- Click Right on android project >> Run as >> Android Application
  - Test Result : the text is returned from C code

Lab : Run Native Application on Android

- From /home/mmn/workspace/mmn/bin , you can find out the mmn.apk .
- Copy the apk file to <android_fs>/system/app/
- Or Copy the apk file to android_fs/data/app/
- Run your apk on devkit8000 and play video!
- Note: if the apk is updated , you have to restart booting devkit8000

Step1:Use NDK to build the native library for android

- Reference script

code

```
0.sudo tar -xzvf android_jni_dev.tgz
1.cd android-ndk-1.5_r1/sources/samples/
2.mkdir helloqq
3.cd helloqq
4.vim Android.mk
5.vim helloqq.c
6.vim android-ndk-1.5_r1/app/helloqq/Application.mk
7.cd android-ndk-1.5_r1/
8.make APP=helloqq    //APP="module name"
```

Step2:Put library into android filesystem

- Successful message

```
SharedLibrary  : libhelloqq.so
Install        : libhelloqq.so => apps/helloqq/project/libs/armeabi
```

- Copy the library to android filesystem
  - $sudo cp app/helloqq/project/libs/armeabi/libhelloqq.so <android_fs>/system/lib/

Step3:Load library in apk source code

```
public class HelloTest extends Activity {
    /** Called when the activity is first created. */
    TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv=(TextView)findViewById(R.id.myTextView1);
        tv.setText("123");
        tv.setText(stringFromJNI(0));

    }

    public native String  stringFromJNI(int prog);

    static {
        System.loadLibrary("helloqq");
    }
}
```

Step4:Put the modified apk and socket-server into android filesystem

- After compile the apk source the apk will appear under <apk_project>/bin/xxx.apk

*Step5:Run Native Application on Android

- Compile and put the socket-server to <android_fs>/system/bin/
- Write a native service in <android_fs>/init.rc
- Reboot devkit8000
- Run your apk to see whether the socket-server do something or not.

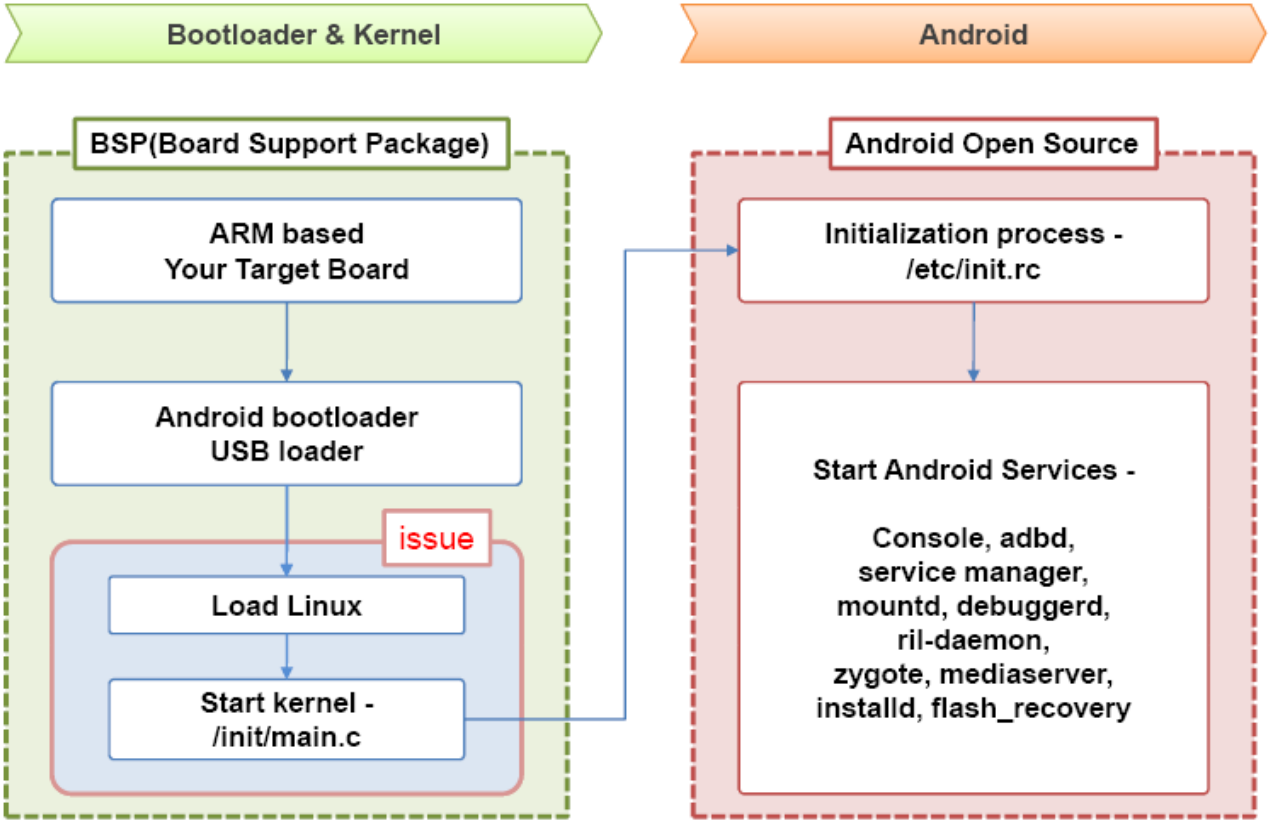
  (refer to system/etc/omap_android.sh)

附錄

- *Introduction to Android*
- *Android Architecture*
- *Android Multimedia Framework*
- *Android Porting*
- ***Android start-up programming***
- *LAB : Mount Android Filesystem*

Android start-up program

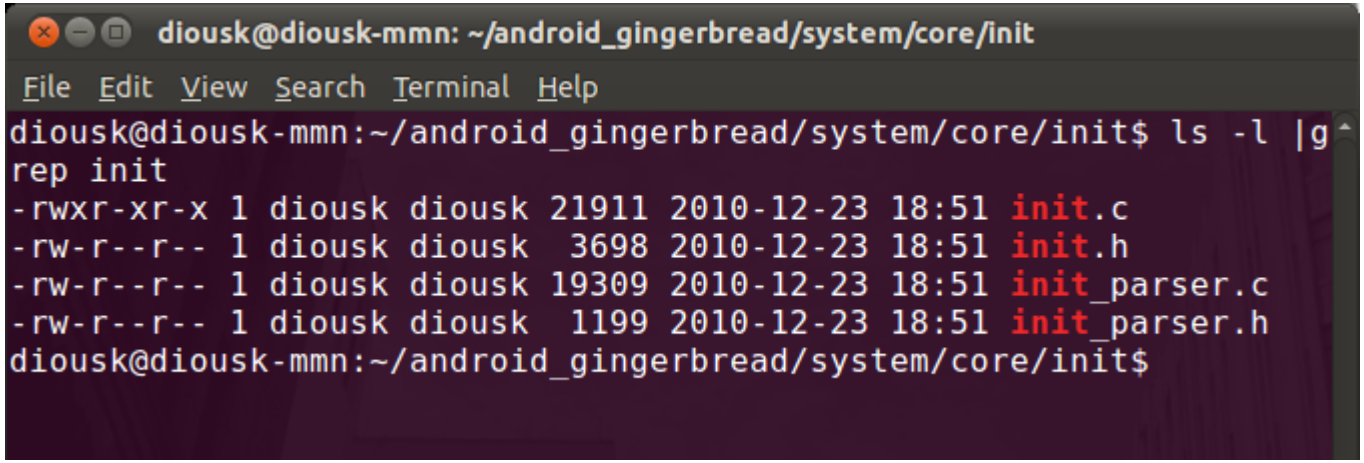
- What happened during Android booting stage ?(con'd)



From Korea Android Community- www.kandroid.org

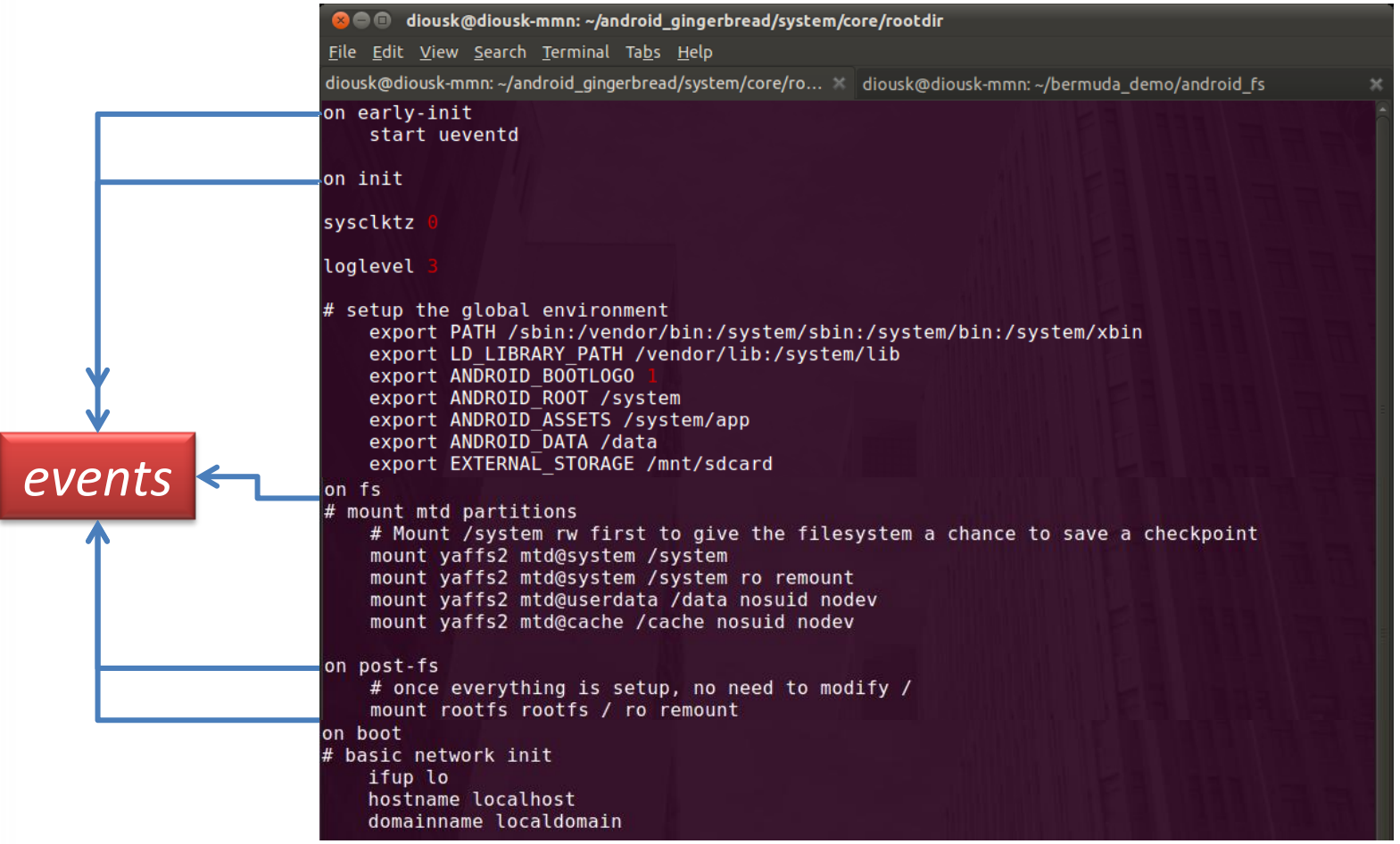
Android start-up program

- What happened during Android booting stage ?(con'd)
  - Kernel will execute "**init**" for starting android initialization
  - "**init**" will read the init.rc file to set up the environment variable or properties and start android services
- "**init**" is the first process after kernel started. The corresponding source code lies in <android_src>/system/core/init



Android start-up program

- Init.rc (under android_src/system/core/rootdir/init.rc)



Android start-up program

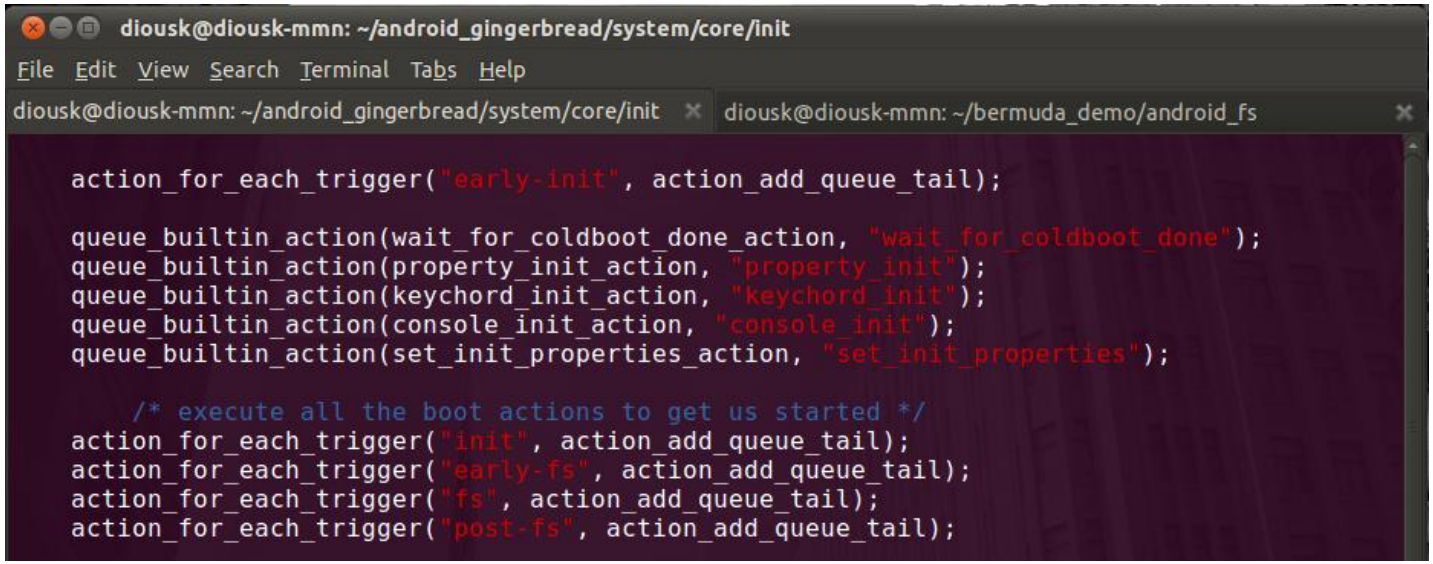
- "**init**" does the following tasks step by step:
  - 1.Initialize log system.
  - 2.Parse /init.rc
  - 3.Execute ***early-init action*** parsed in step 2.

<init.c>

```
int main(int argc, char **argv)
{
    INFO("reading config file\n");
    init_parse_config_file("/init.rc");

    /* pull the kernel commandline and ramdisk properties file in */
    import_kernel_cmdline(0);

    get_hardware_name(hardware, &revision);
    snprintf(tmp, sizeof(tmp), "/init.%s.rc", hardware);
    init_parse_config_file(tmp);

    action_for_each_trigger("early-init", action_add_queue_tail);
```

Android start-up program

- "**init**" does the following tasks step by step(con'd):
  - 4.Device specific initialize. For example, make all device node in /dev
  - 5.Initialize property system. Actually the property system is working as a share memory. Logically it looks like a registry under Windows system.
  - 6.Execute ***init action*** in the two files parsed in step 2.



Android start-up program

- "**init**" does the following tasks step by step(con'd):
  - 7.Start property service.
  - 8.Execute ***early-boot and boot actions*** in the two files parsed in step 2.
  - 9.Execute property action in init.rc parsed in step 2.
  - 10.Enter into an indefinite loop to wait for device/property set/child process exit events.

```
/* execute all the boot actions to get us started */
action_for_each_trigger("early-boot", action_add_queue_tail);
action_for_each_trigger("boot", action_add_queue_tail);
```


Android start-up program

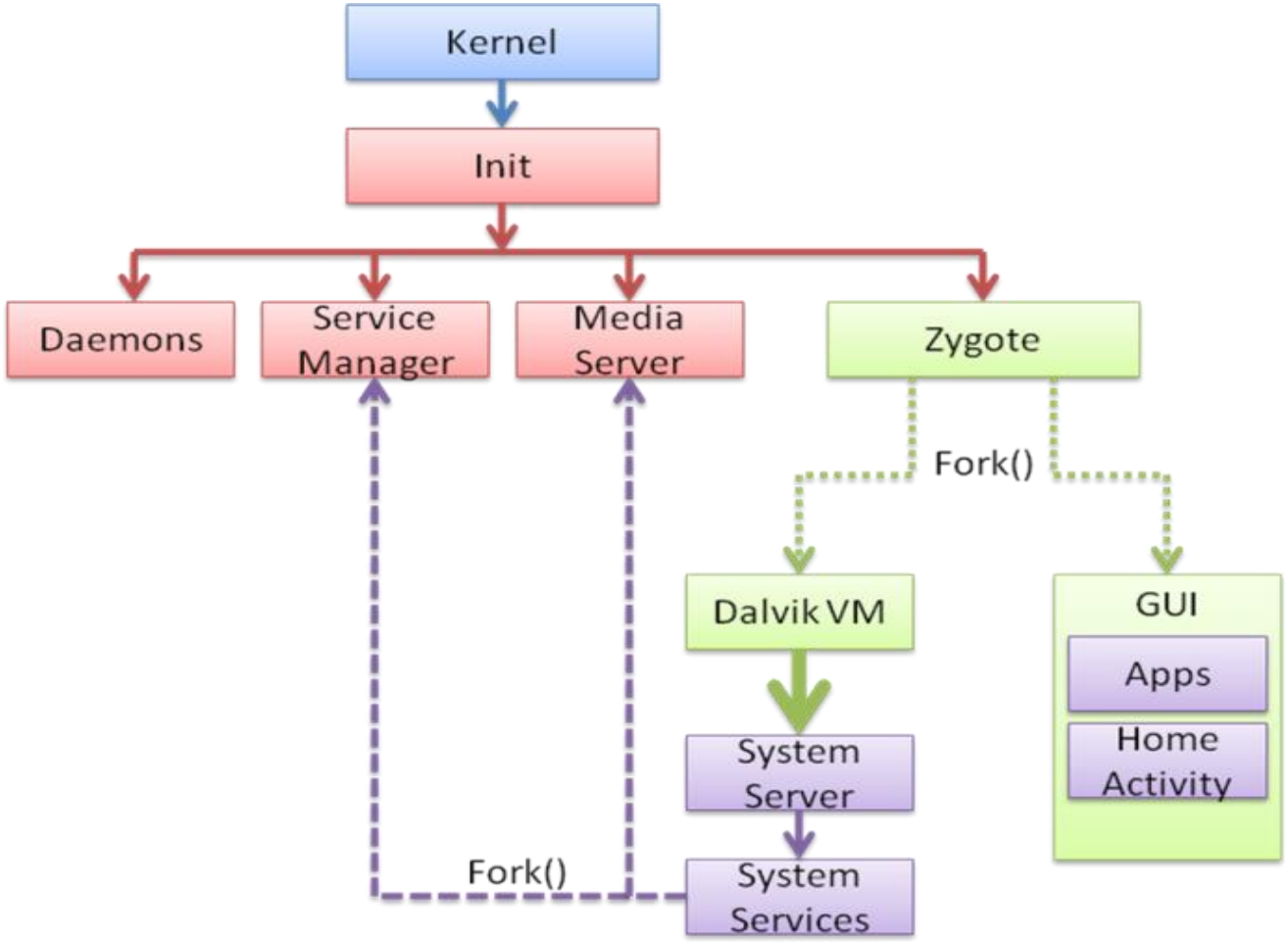
- After init process , there are two main functions (Zygote, System server)in booting:
  - Zygote does the following tasks step by step:
    - 1.Create JAVA VM.
    - 2.Register android native function for JAVA VM.
    - 3.Call the main function in the JAVA class named com.android.internal.os.ZygoteInit
      - **Call Zygote::forkSystemServer (implemented in dalvik/vm/native/dalvik_system_Zygote.c)to fork a new process.**
    - 4. call IPCThreadState::self()->joinThreadPool() to enter into service dispatcher.

Android start-up program

- SystemServer will start a new thread to start all JAVA services as follows:
- Core Services:
  - 1.Starting Power Manager
  - 2.Creating Activity Manager
  - 3.Starting Telephony Registry
  - 4.Starting Package Manager
  - 5.Set Activity Manager Service as System Process
  - 6.Starting Context Manager
  - 7.Starting System Context Providers
  - 8.Starting Battery Service
  - 9.Starting Alarm Manager
  - 10.Starting Sensor Service
  - 11.Starting Window Manager
  - 12.Starting Bluetooth Service
  - 13.Starting Mount Service

Android start-up program

- Booting diagram



Android start-up program

- Service in android