A Talk for NTUST on 2011.12.19

The Device Driver Structure for Android
with Linux Kernel Driver and Android HAL

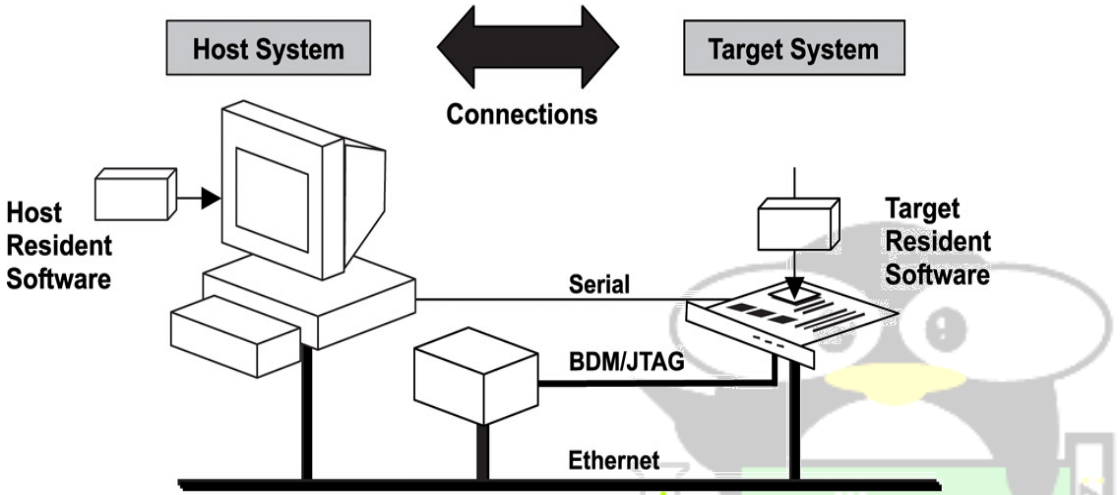
William W.-Y. Liang (梁文耀), Ph. D.
http://www.ntut.edu.tw/~wyliang

This talk was given in a seminar for National Taiwan University of Science and Technology on Dec. 2011. The slides has been partially updated in this release.

本投影片取自 2011.12 於國立台灣科技大學電子系之演講, 並進行部分更新.

Building an Android/Linux System

- Hardware
- Software development environment
- Tool chain and Library
- Boot loader
- OS Kernel
- Middleware*
- Applications

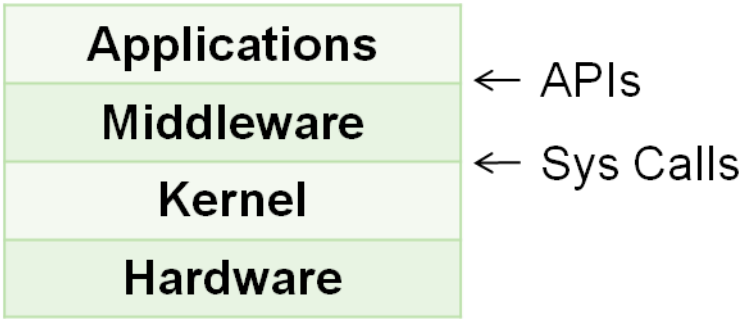


Source: Qing Li and Caroline Yao, "real-time concepts for embedded systems"

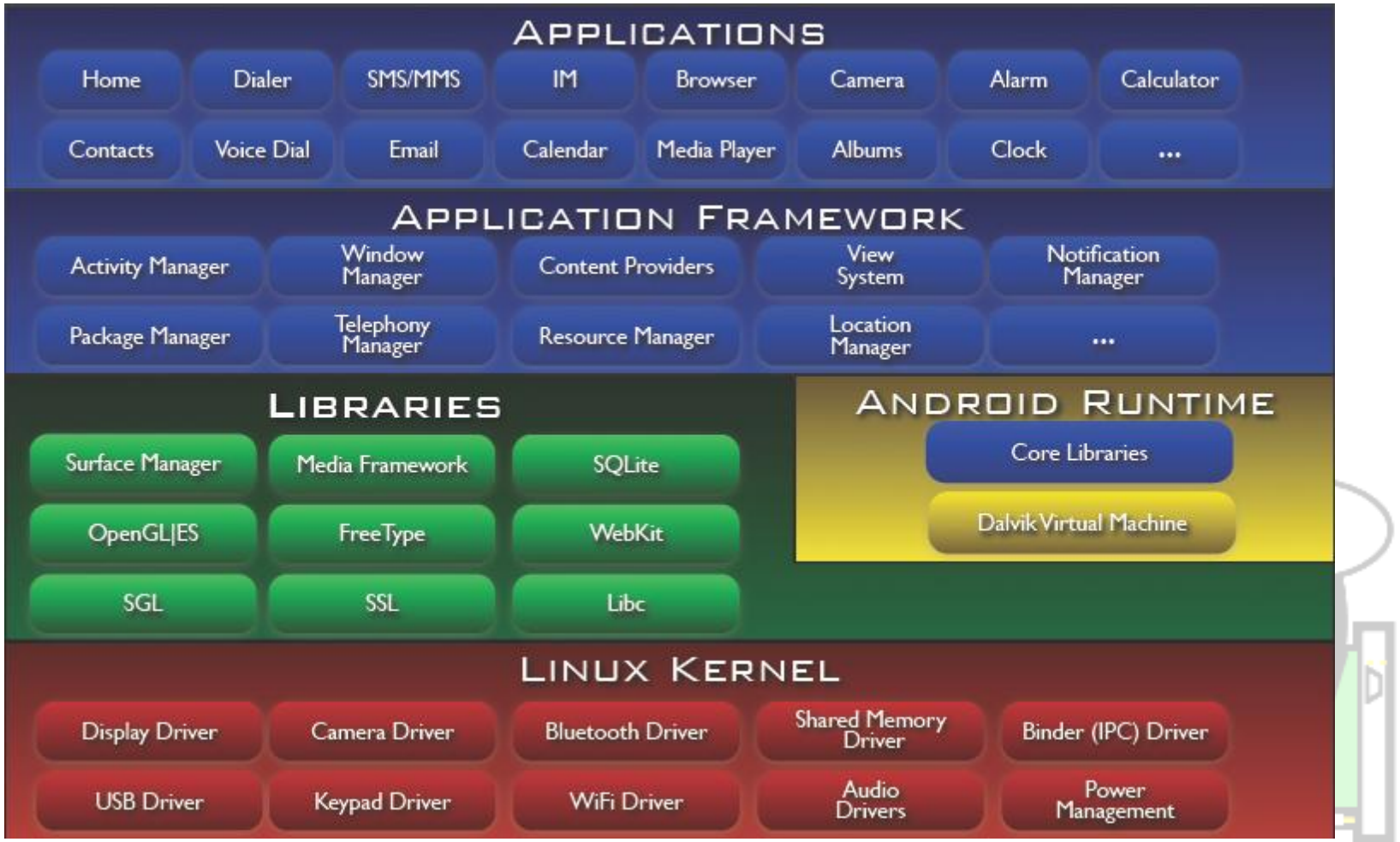
*Middleware: In Android, it's called the Application Framework.

User vs. Kernel

- For a general purpose OS such as Linux
- User level (user mode)
  - User Programs
  - Compilation
  - Linking
  - Launch
  - Middleware services
- Kernel level (kernel mode)
  - Loader
  - System calls
  - Kernel features
  - Device drivers
  - Hardware manipulations
- System interface
  - User space middleware API
  - Kernel level system calls



Android Architecture

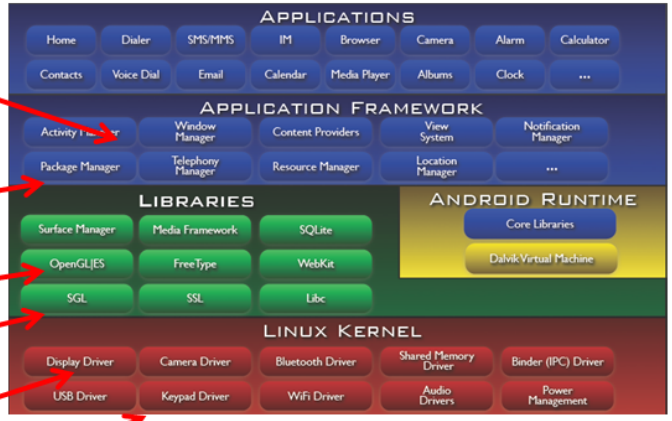


System Integration

- What do we mean by ‘System Integration’ here?
- System integration includes:
  - Application
  - Middleware
  - Operating System
  - Device Driver
  - Hardware

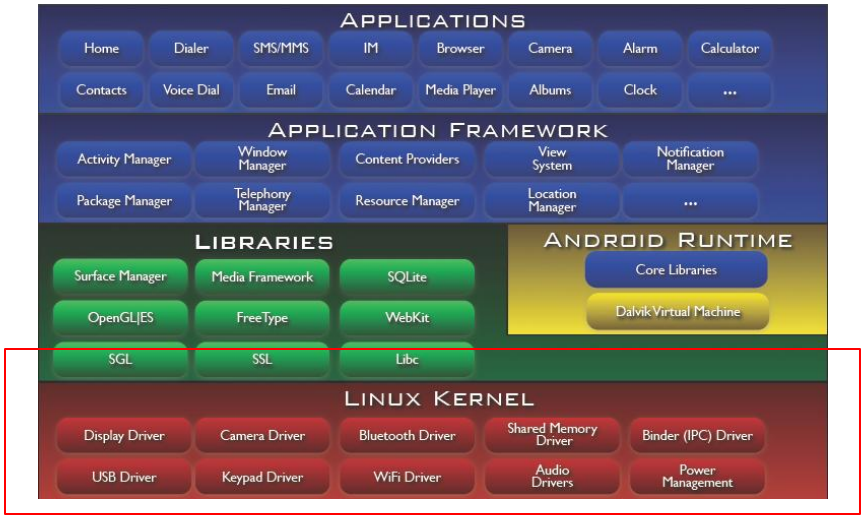
Android/Linux System Integration

- Android Application
- Android Application Framework
- Android Java/C/C++ Interface
- Android/Linux Native Code
- Linux System Calls
- Linux Device Driver
- Hardware and Control



The Linux Kernel

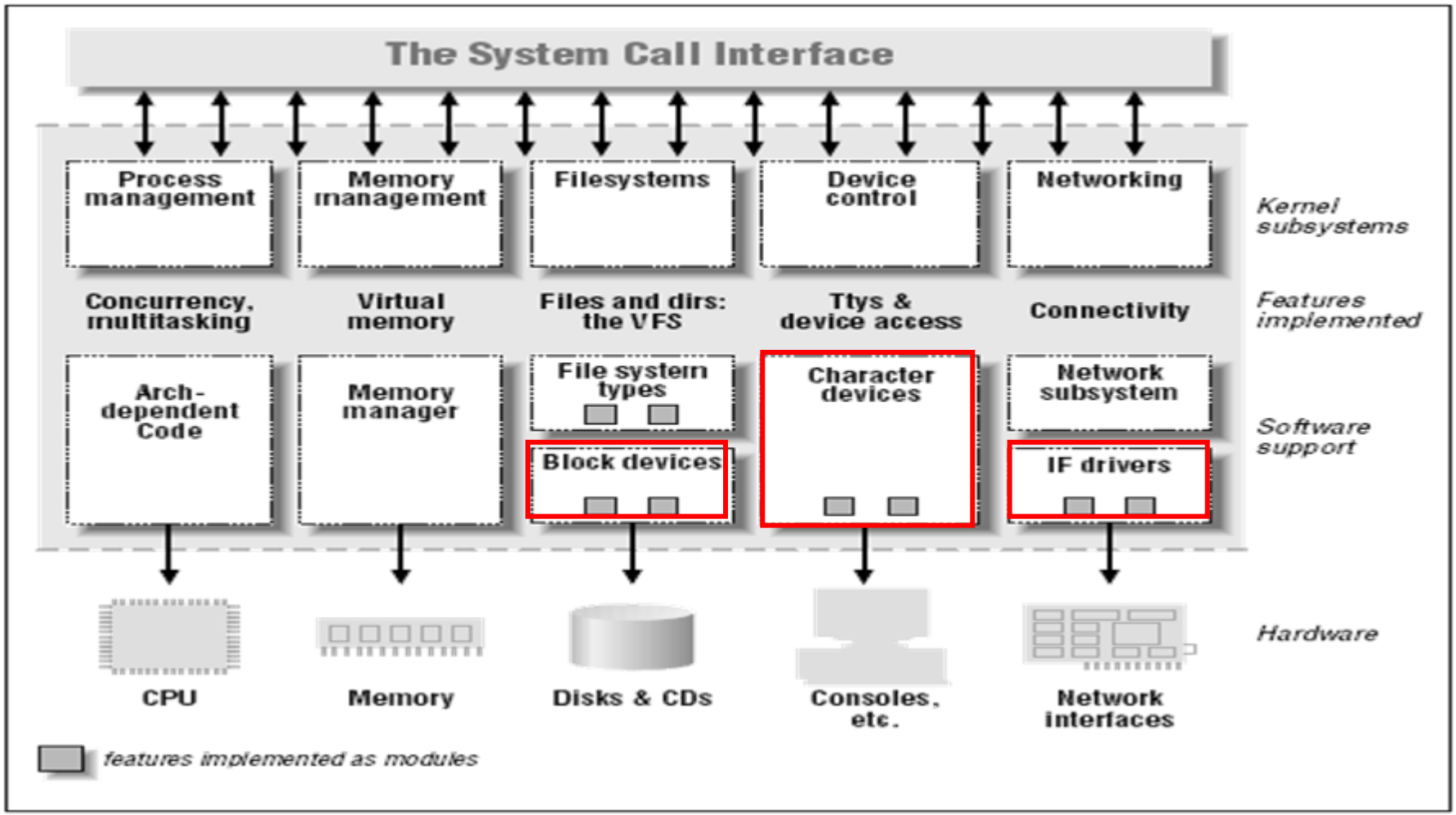
- Android relies on Linux version 2.6 and above for core system services such as security, memory management, process management, network stack, and driver model.
  - Kernel 3.0/3.4/3.10 are commonly seen on Android 4.x ~ 5.x.
- The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.



The Role of the Device Driver

- Device drivers are usually treated as black boxes for the application developers.
- They resemble as a software layer lying between the applications and the actual devices.
- A device driver performs user requests in a standardized manner.

Device Driver in Linux Kernel

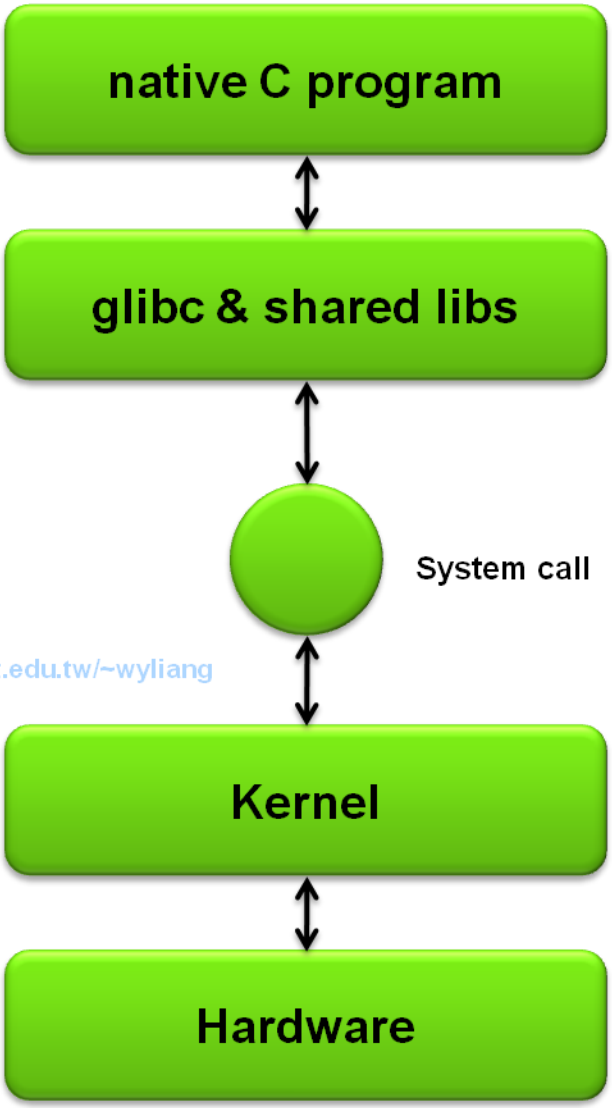


ALESSANDRO RUBINI and JONATHAN CORBET, Linux Device Drivers, Second Edition , O'Reilly & Associates, Petaluma, CA, 2001

Types of Files

- Ordinary files for programs or data
- UNIX/Linux systems implement several kinds of 'special' files such as device files and symbolic links which enable users to employ familiar commands and functions such as *open*, *read*, *write*, and *close* when working with other kinds of objects.
- Example file operations
  - *int open(char *pathname, int flags, ...);*
  - *int read(int fd, void *buf, size_t count);*
  - *int write(int fd, void *buf, size_t count);*
  - *int lseek(int fd, loff_t offset, int whence);*
  - *int close(int fd);*

The Traditional Linux Device Control Model



Device Files

- A */dev* example

crw-rw-rw-	*1 root*	*root*	*1, 3*	*Apr 11 2007 null*
crw-------	*1 root*	*root*	*10, 1*	*Apr 11 2007 psaux*
crw-rw-rw-	***1 root***	***root***	***48, 0***	***Apr 11 2007 scull***
crw-------	*1 root*	*root*	*4, 1*	*Apr 11 2007 tty1*
crw-rw-rw-	*1 root*	*tty*	*4, 64*	*Apr 11 2007 ttys0*
crw-rw----	*1 root*	*uucp*	*4, 65*	*Apr 11 2007 ttyS1*
crw--w----	*1 vcsa*	*tty*	*7, 1*	*Apr 11 2007 vcs1*
crw--w----	*1 vcsa*	*tty*	*7, 129*	*Apr 11 2007 vcsa1*
crw-rw-rw-	*1 root*	*root*	*1, 5*	*Apr 11 2007 zero*

- The ***mknod*** system call or utility can be used to create device files, typically in the '/dev' directory.

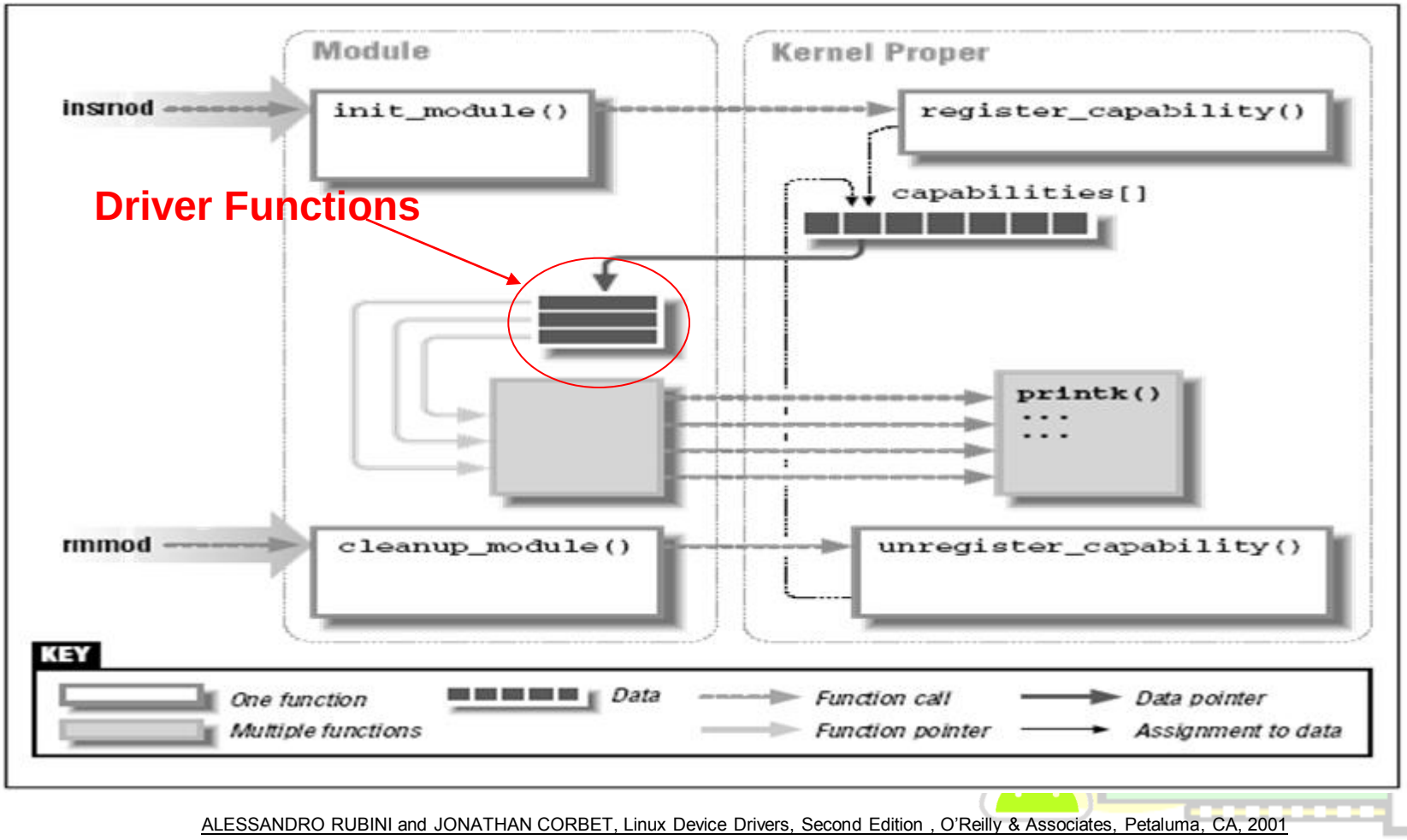
```
# mknod /dev/scull c 48 0
```

where '*/dev/scull*' is the file's pathname, '*c*' indicates that it's a character-mode device, *48* is its (unique) '***major* number**', and 0 is its '***minor* number**'.

Device Access Example

```c
#include <stdio.h>
#include <fcntl.h>

#define DEVFILE "/dev/androint"

int main() {
        int fd;
        int in[2] = {10, 20};
        int out;

        printf("AndroInt virtual adder driver test:\n");
        printf("Input A: ");
        scanf("%d", in);
        printf("Input B: ");
        scanf("%d", in+1);
        printf("Input: %d %d\n", in[0], in[1]);

        fd = open(DEVFILE, O_RDWR);
        write(fd, in, sizeof(in));
        read(fd, &out, sizeof(out));
        close(fd);

        printf("Output: %d\n", out);

        return 0;
}
```

The Relationship of Module and Kernel

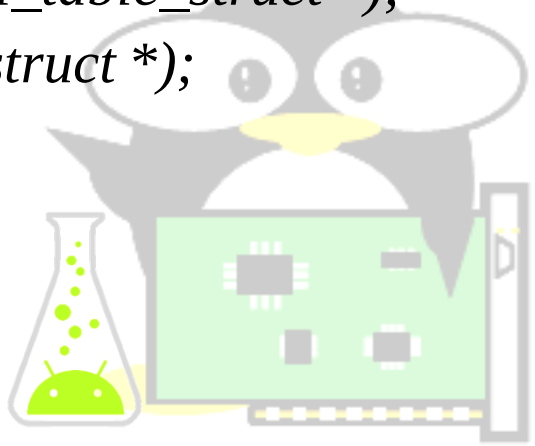


ALESSANDRO RUBINI and JONATHAN CORBET, Linux Device Drivers, Second Edition , O'Reilly & Associates, Petaluma, CA, 2001

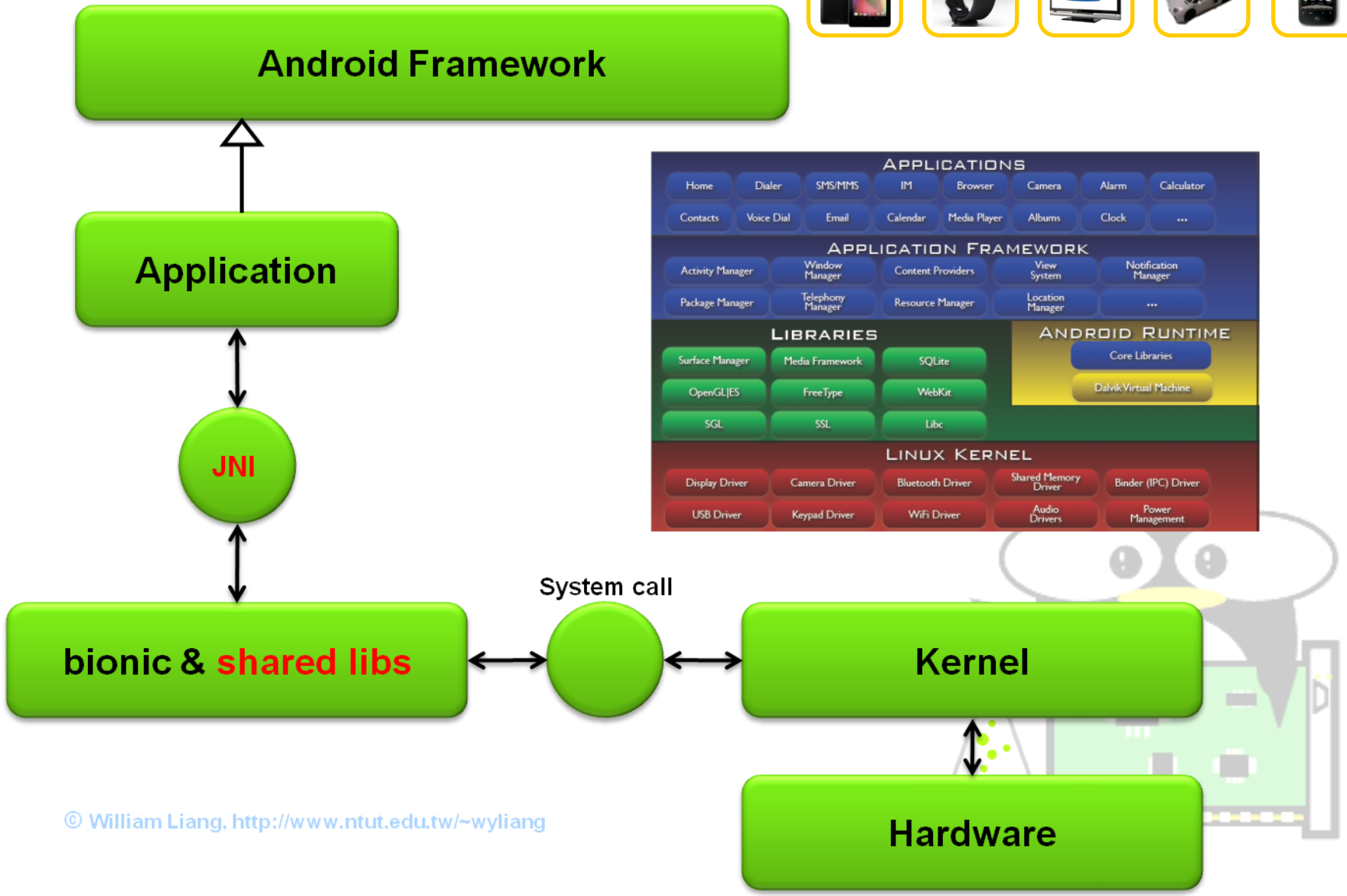
The *file_operations* Structure

- The *file_operations* structure represents the driver's methods.
  - *int (***open**) (struct inode *, struct file *);*
  - *ssize_t (***read**) (struct file *, char *, size_t, loff_t *);*
  - *ssize_t (***write**) (struct file *, const char *, size_t, loff_t *);*
  - *int (*__release__) (struct inode *, struct file *);*
  - *loff_t (*__llseek__) (struct file *, loff_t, int);*
  - *int (*__ioctl__) (struct inode *, struct file *, unsigned int, unsigned long);*
  - *unsigned int (*poll) (struct file *, struct poll_table_struct *);*
  - *int (*mmap) (struct file *, struct vm_area_struct *);*
  - ...

```
struct file_operations androint_fops = {
        .owner =    THIS_MODULE,
        .open =     androint_open,
        .read =     androint_read,
        .write =    androint_write,
        .release =  androint_release,
};
```



A Rough Android Model



Shared Library

- Libraries offer some form of sharing, allowing the same library to be used by multiple programs at the same time.
  - Dynamic loading
  - Sharing
- Examples
  - DLL (Dynamic Link Library): Windows
  - SO (Shared Object): Linux/Unix

JNI: Standard C interface

- JNI: Java Native Interface
- Standard interface between Java and C/C++

AndroIntActivity.java

```
public class AndroIntActivity extends Activity {
    private EditText mEditText1;
    private EditText mEditText2;
    private Button mButton;

    public native int addJNI(int a, int b);

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

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mEditText1 = (EditText) this.findViewById(R.id.EditText01);
        mEditText2 = (EditText) this.findViewById(R.id.EditText02);

        int inputtype = android.text.InputType.TYPE_CLASS_NUMBER|andr
        mEditText1.setInputType(inputtype);
        mEditText2.setInputType(inputtype);

        mButton = (Button) this.findViewById(R.id.Button01);
        mButton.setOnClickListener(mAddListener);
    }
```

Invoke Device Drivers from the Native Layer

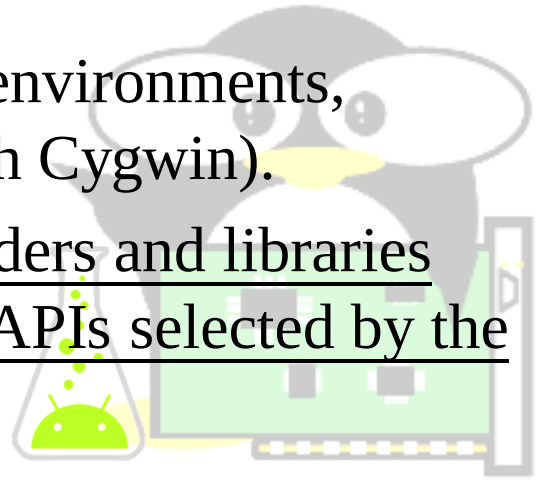
*androint-jni.c

```
#include <fcntl.h>
#include "com_eps_william_androint_AndroIntService.h"

#define DEVFILE "/dev/androint"
#define BUFLEN  128

JNIEXPORT jint JNICALL Java_com_eps_william_androint_AndroIntActivity_addJNI
  (JNIEnv *env, jobject thiz, jint a, jint b)
{
        int fd;
        int buf[2] = {a, b};
        int result;

        fd = open(DEVFILE, O_RDWR);
        write(fd, buf, sizeof(int)*2);
        read(fd, &result, sizeof(int));
        close(fd);

        return result;
}
```

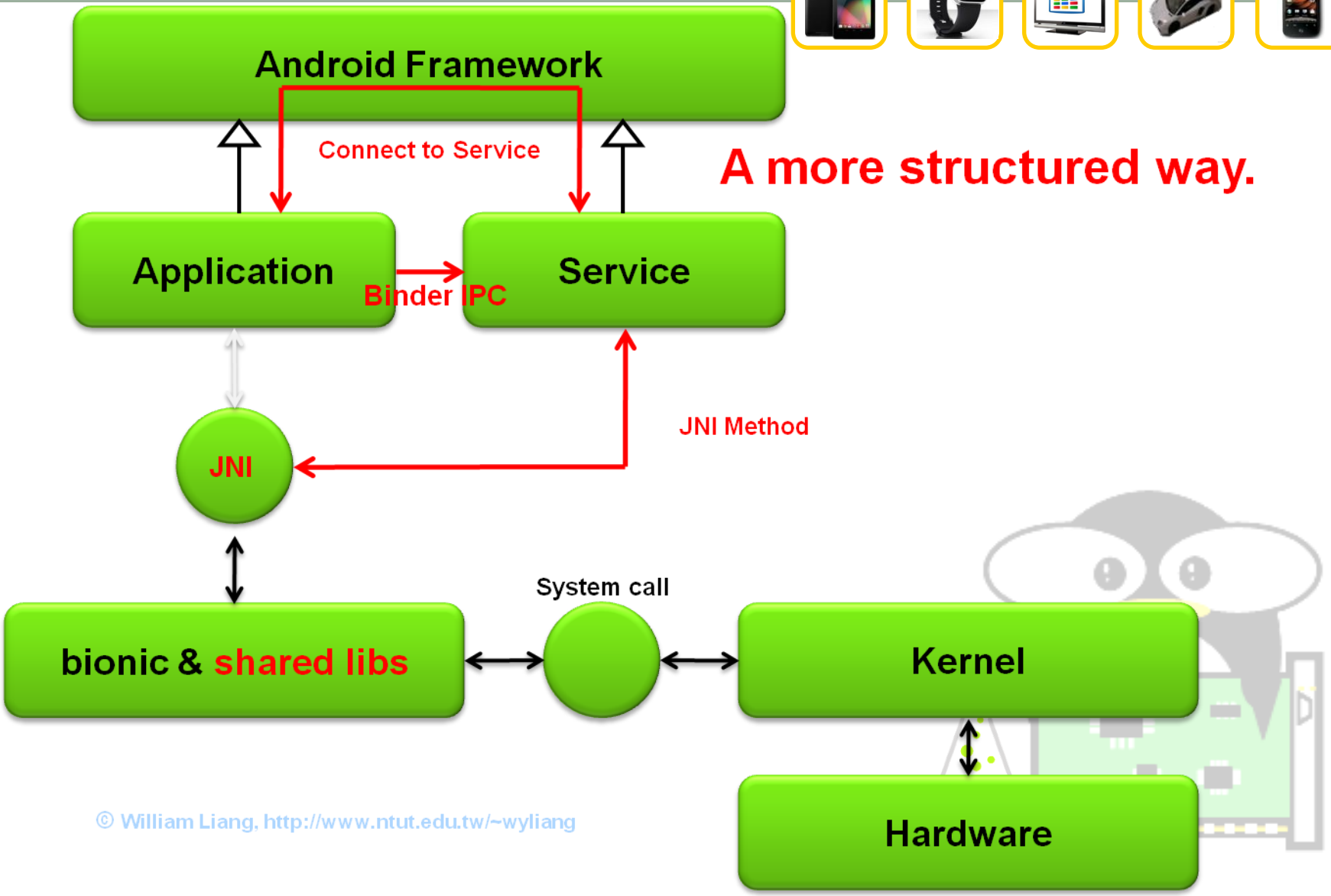
The Android Native Code Development Kit

- Android provides a set of cross-compilation tool-chains that can generate native binaries.
- It supports the ARM, x86, and MIPS target architectures.

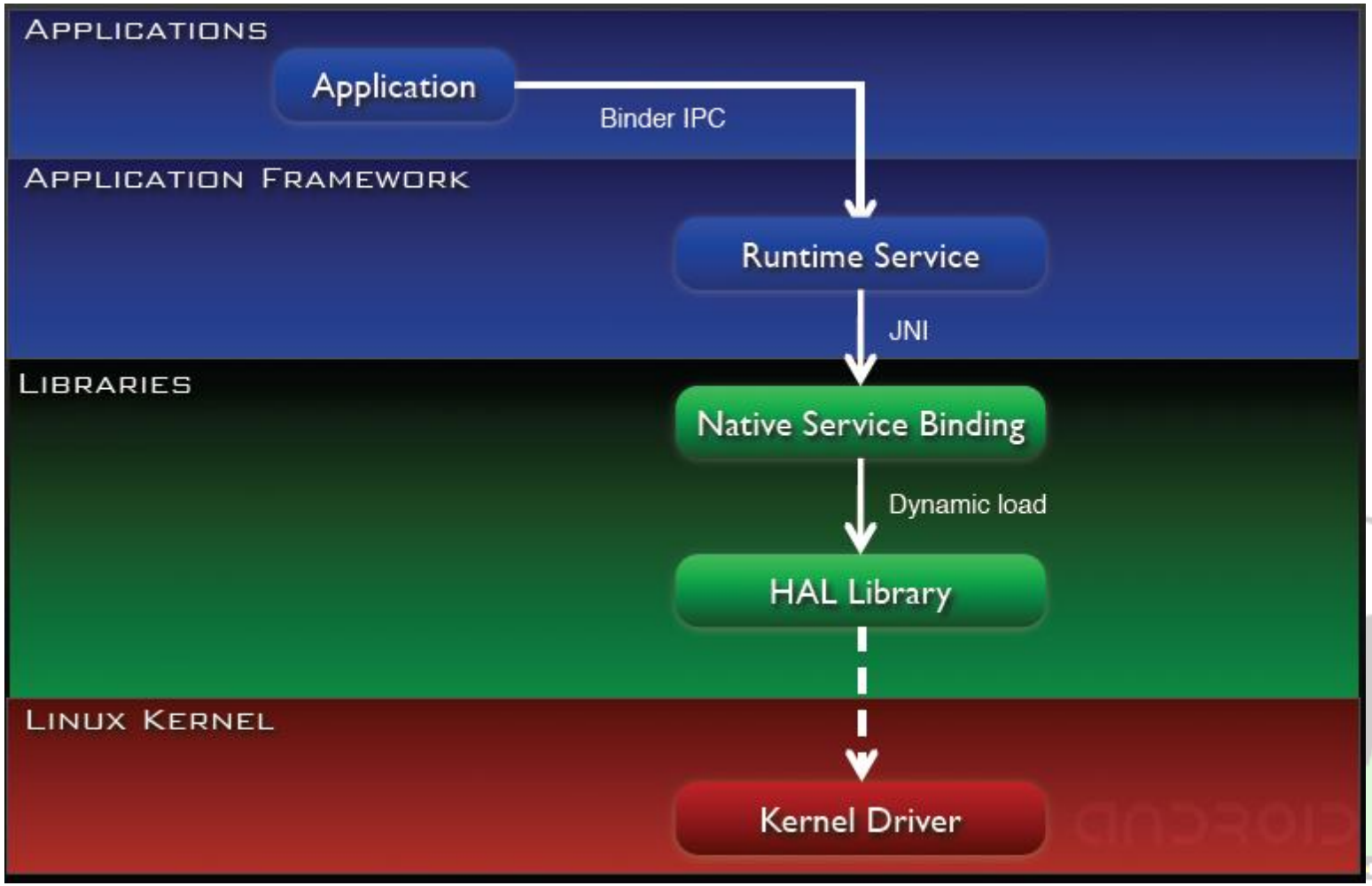
- NDK is a tools for generating JNI-compatible shared libraries without using the complete AOSP (Android Open Source Project) development environment.
- It can be run on several host development environments, including Linux, OS X, and Windows (with Cygwin).
- NDK actually provides a set of system headers and libraries corresponding to a list of the stable native APIs selected by the AOSP development team.



The Formal Android Device Control Model



Example



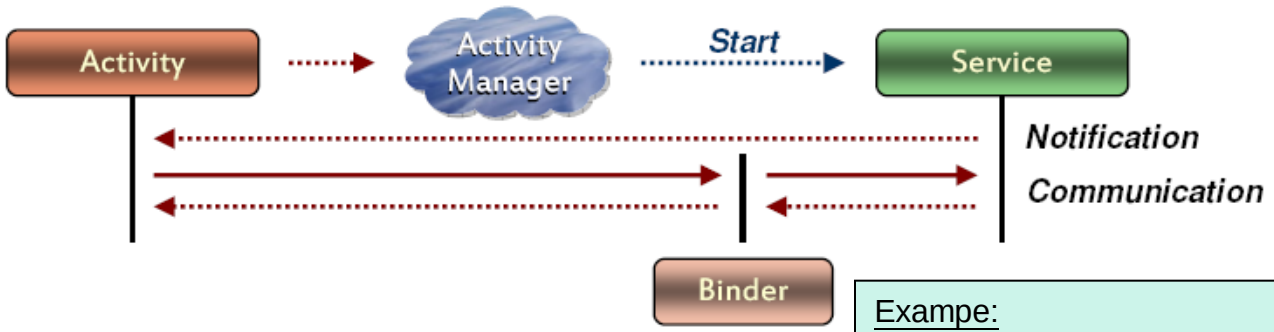
Slides from "Android Anatomy and Physiology," Patrick Brady ©

Android Services

- Similar to activities, but without UI
- For long-running background tasks
- Framework "Service" class must be extended

- Core Services
  - Activity Manager
  - Package Manager
  - Window Manager
  - Resource Manager
  - Content Providers
  - View System
- Hardware Services
  - Telephony Service
  - Location Service
  - Bluetooth Service
  - WiFi Service
  - USB Service
  - Sensor Service

It's possible to connect to (**bind** to) an ongoing service. While connected, you can communicate with the service through an **interface** that the service exposes.

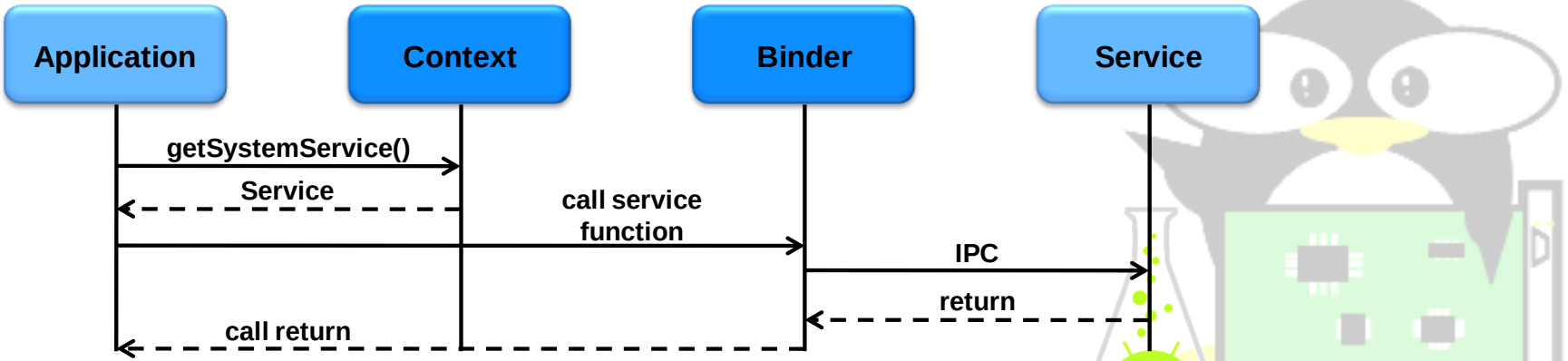


Exampe:
startService(new Intent(this_activity.this, some_service.class));

Slides based on "Deep inside Android," Gilles Printemps, Esmertec ©

Binder IPC

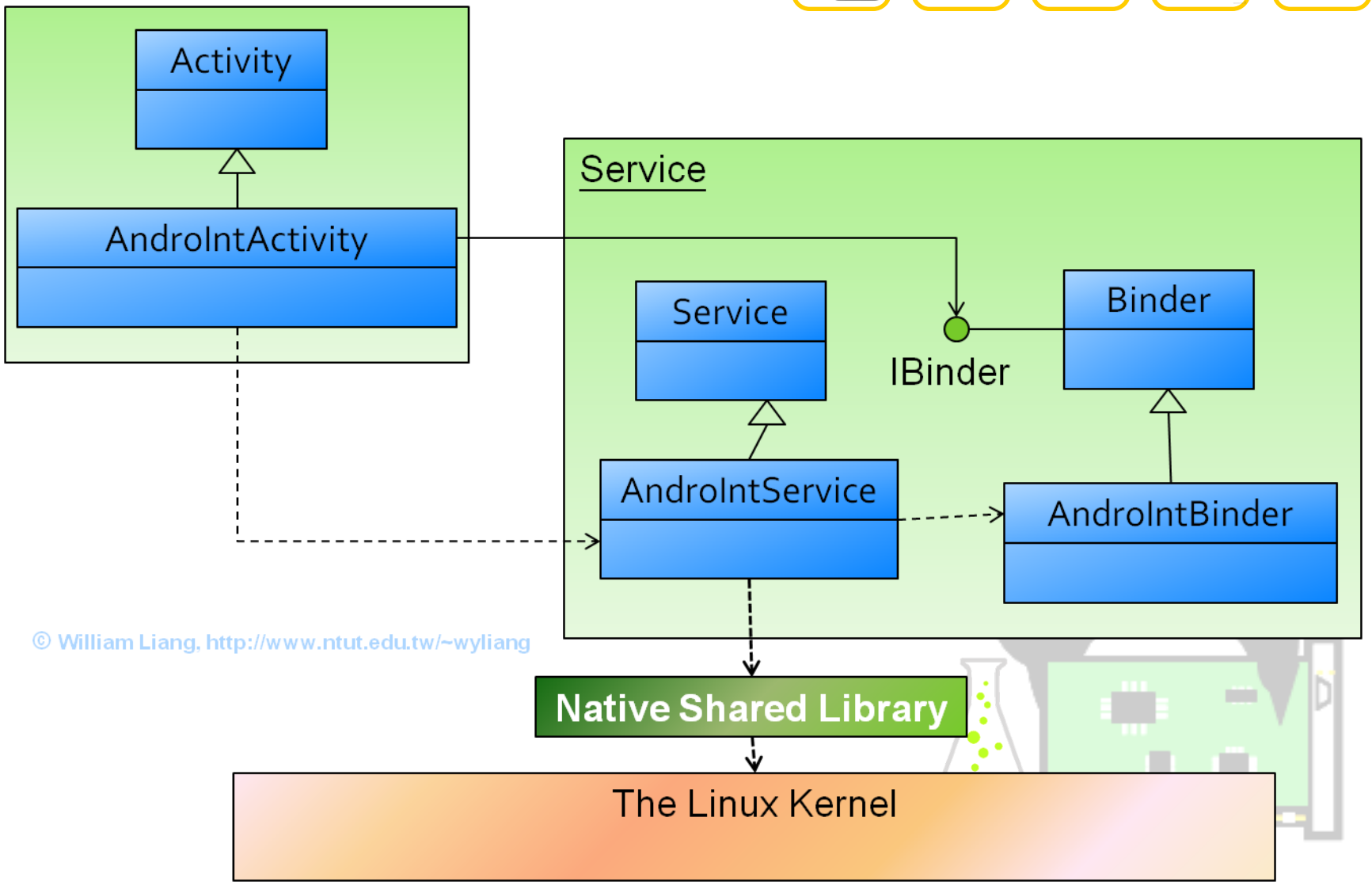
- Problem to solve
  - Applications and Services may run in separate processes but must communicate and share data.
  - IPC can introduce significant processing overhead and security holes.
- Solution
  - Driver to facilitate inter-process communication (IPC)
  - High performance through shared memory
  - Reference counting, and mapping of object references across processes
  - Synchronous calls between processes



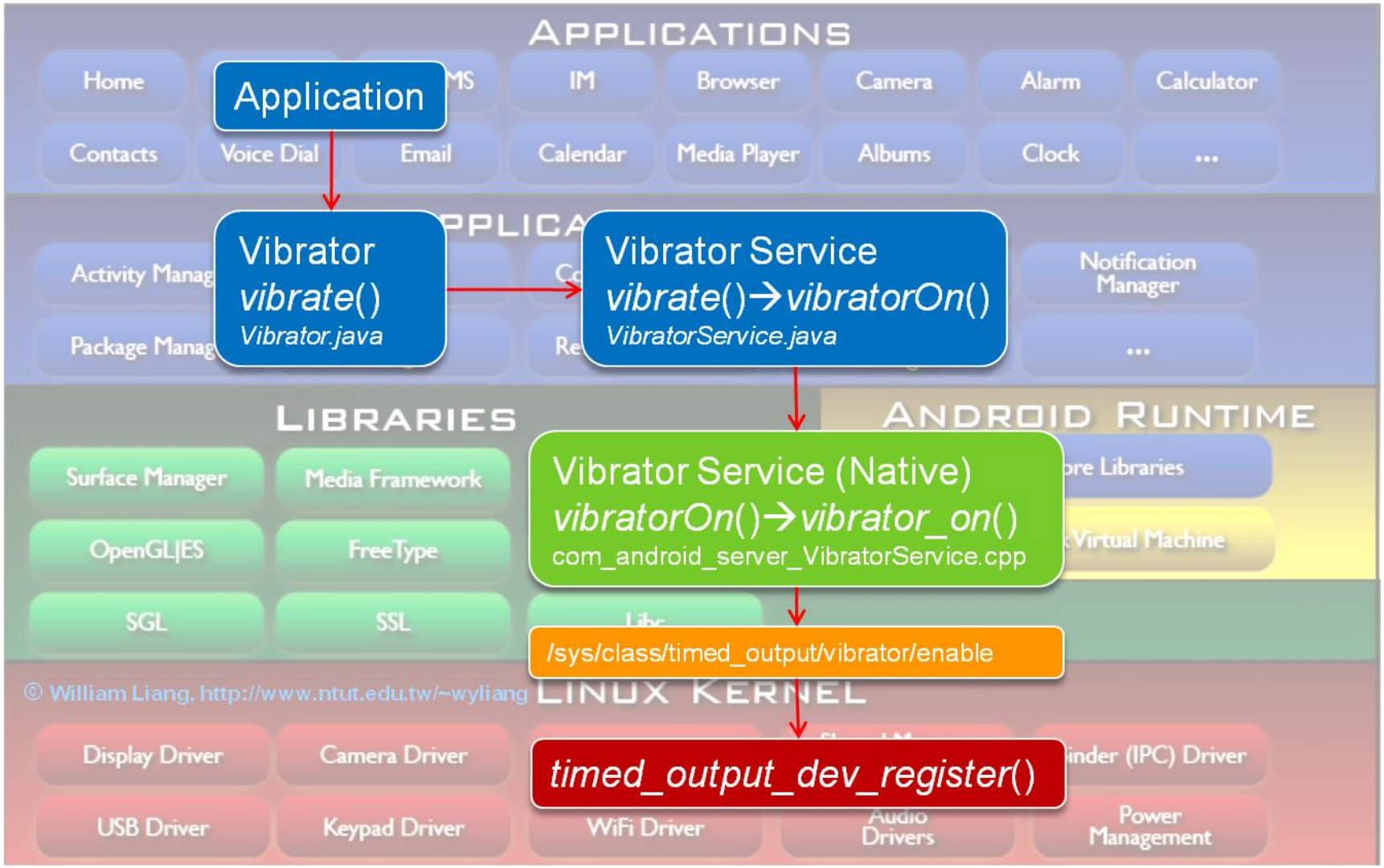
Slides from "Android Anatomy and Physiology," Patrick Brady ©

AIDL:
Android Interface Definition Language

Example Service Structure

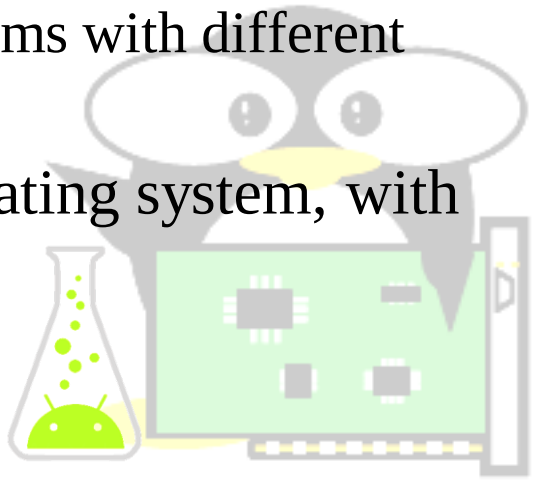


A Real Example: Vibrator Service



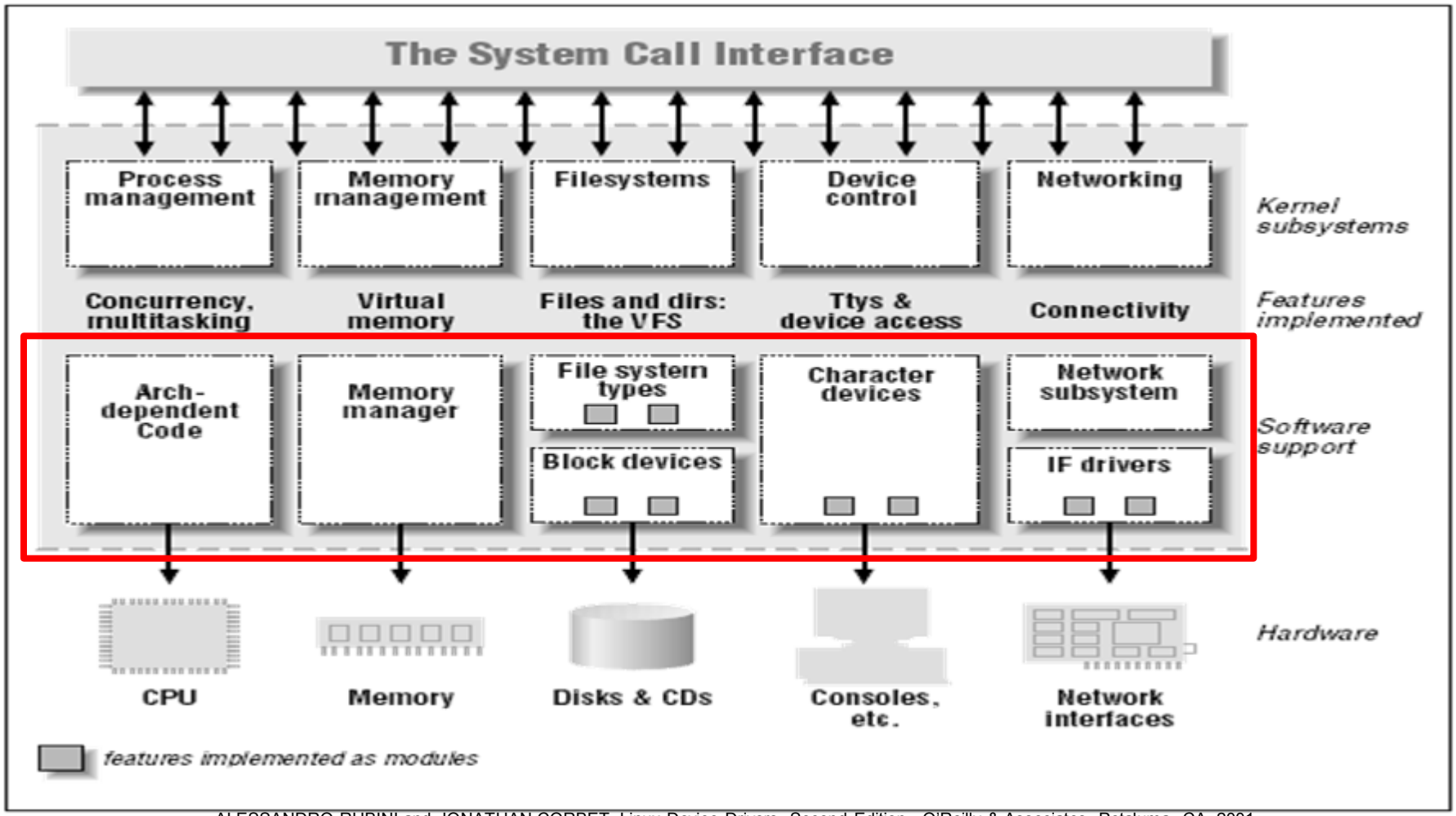
The Hardware Abstraction Layer

- A hardware abstraction layer (HAL) is an abstraction layer, implemented in software, between the physical hardware of a computer and the software that runs on that computer.
- Typically seen in portable operating system design, such as Linux.
  - Its function is to hide differences in hardware from most of the operating system kernel, so that most of the kernel-mode code does not need to be changed to run on systems with different hardware.
- Realized in user-space of the Android operating system, with the concept of user-space device driver.



Contents from Wikipedia

HAL in Linux Kernel



ALESSANDRO RUBINI and JONATHAN CORBET, Linux Device Drivers, Second Edition , O'Reilly & Associates, Petaluma, CA, 2001

The Android Hardware Abstraction Layer

Application

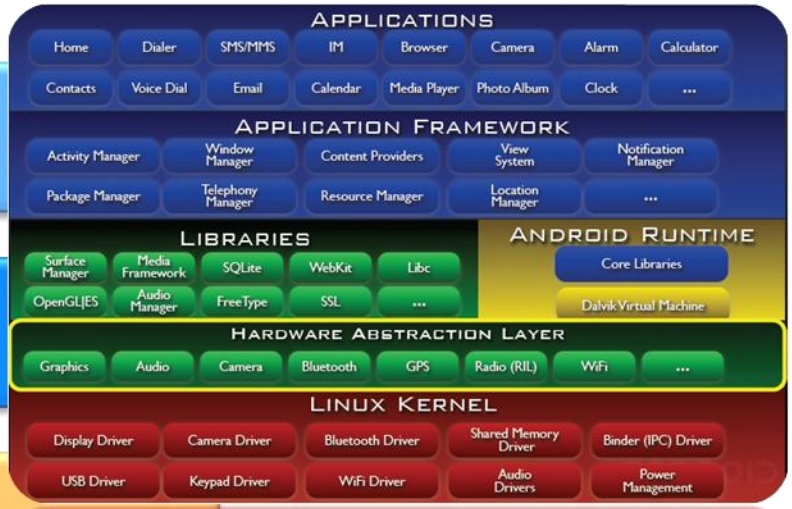
Application Framework

Libraries

Android Runtime

Hardware Abstraction Layer

Linux Kernel



Flow for Retrieving HAL Module and Methods

Framework

↓

Native HAL Stub

...

hw_get_module()

...

1

Native HAL Module

hw_module_t

...

...

hw_module_methods_t

open()

2

hw_device_t

...

...

close()

device_specific_methods()

3

device_specific_methods() {

...

}

Example: HAL Module Invocation

```
JNIEXPORT jint JNICALL Java_com_eps_william_androint_AndroIntService_addJNI
  (JNIEnv *env, jobject thiz, jint a, jint b)
{
    struct androint_device_t* androint_dev = NULL;
    int result;

    if (g_androint_dev == NULL) {
        int err;
        hw_module_t* module;

        if (hw_get_module(ANDROINT_HARDWARE_MODULE_ID, (hw_module_t const**)&module)) {
            LOGD("hw_get_module failed ...");
            return -99999999;
        }
        LOGD("hw_get_module ok");

        if (module->methods->open(module, ANDROINT_ID, (struct androint_device_t*)&androint_dev)) {
            LOGD("module-open failed ...");
            return -99999999;
        }
        LOGD("module-open ok");

        g_androint_dev = androint_dev;
    } else
        androint_dev = g_androint_dev;

    LOGD("ready to call add %d, %d", a, b);

    result = androint_dev->add(androint_dev, a, b);

    LOGD("get result %d", result);

    return result;
```

androint.h

```
struct androint_device_t {
    struct hw_device_t common;

    int (*add)(struct androint_device_t* dev, int a, int b);
};
```

A Real HAL Example: Lights Service

Lights Service (**Framework**)
LightsService()→init_native()
set()→setLightLocked()→setLight_native()*
LightsService.java

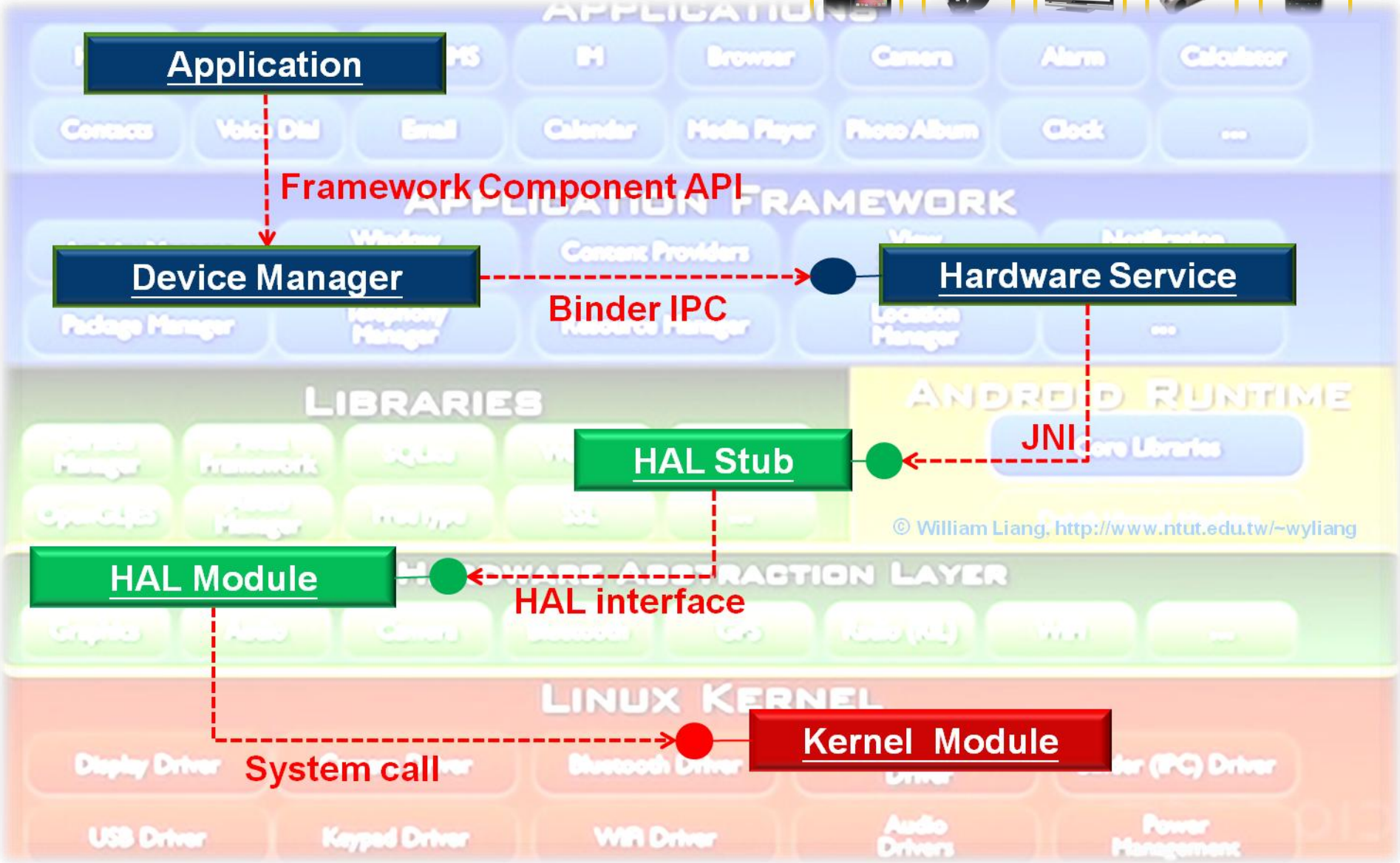
Lights Service (**Native HAL Stub**)
init_native()→ hw_get_module()→ module->methods->open()
setLight_native()→ light_device_t->set_light()
com_android_server_LightsService.cpp

Lights Service (**HAL Module**)
struct hw_module_t, lights_module_methods, open_lights(), set_light_*()
lights.c

//sys/class/leds/*/brightness

led_classdev_register()

Overall Android Device Driver Architecture



Q&A

- 其他分享: http://goo.gl/6qxlSv
- Email: william.wyliang@gmail.com
- Home: http://www.ntut.edu.tw/~wyliang
- FB: http://www.facebook.com/william.wyliang

關於講者：梁文耀 (William W.-Y. Liang)

- Professional Consultant and Open Source Developer (2014.10~)
- 鴻海科技集團創新數位系統事業群資深處長 (2013.01~2014.09)
- 安佐立科技顧問公司技術總監 (2012.08~2013.07)
- 國立台北科技大學資訊工程系專任助理教授 (2005.02~2012.07)
- 先前經歷: 聚興科技研發處協理、晶慧資訊研發副總經理、晶慧資訊研發部經理、晶慧資訊資深工程師、美商 Avant! 軟體工程師
- 專長領域：作業系統、嵌入式系統、計算機結構、平行與分散式系統
  - ✓ Linux 系統軟體 (1993~Now)
  - ✓ Android 與 Linux 嵌入式系統核心及軟硬整合開發 (2001~Now)
- 國立台灣大學資訊工程博士、國立清華大學資訊科學碩士