MICROOH 麦可网

Android-从程序员到架构师之路

出品人：Sundy
讲师：高焕堂（台湾）

http://www.microoh.com

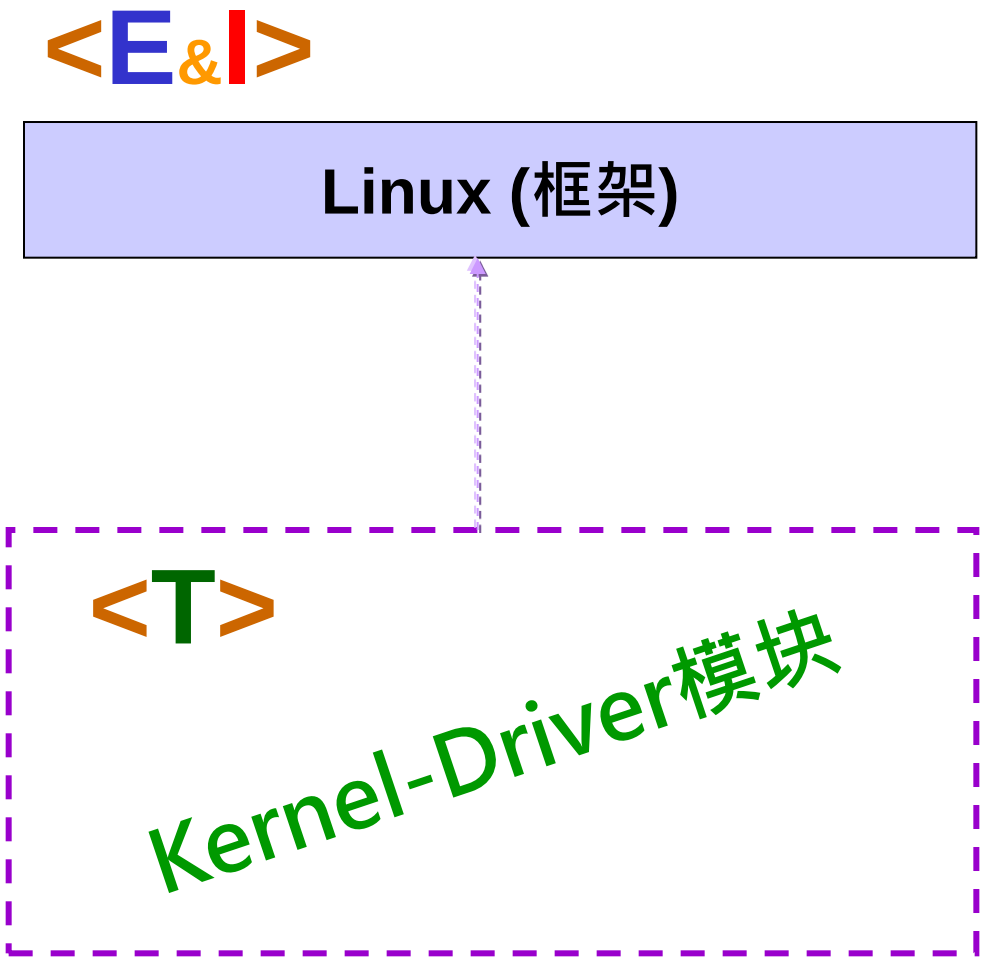
E04_d

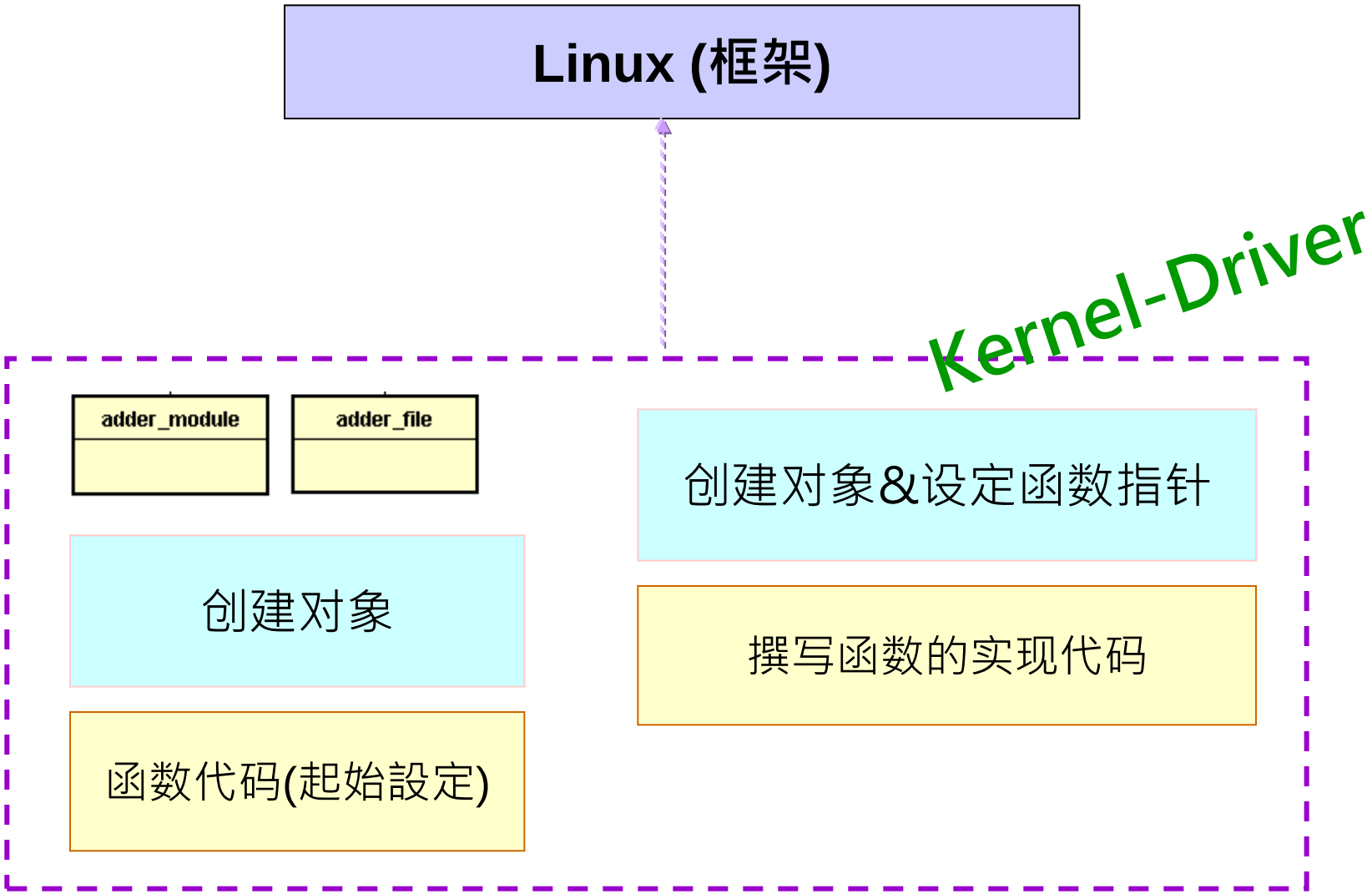
从框架看HAL和Linux驱动开发(d)

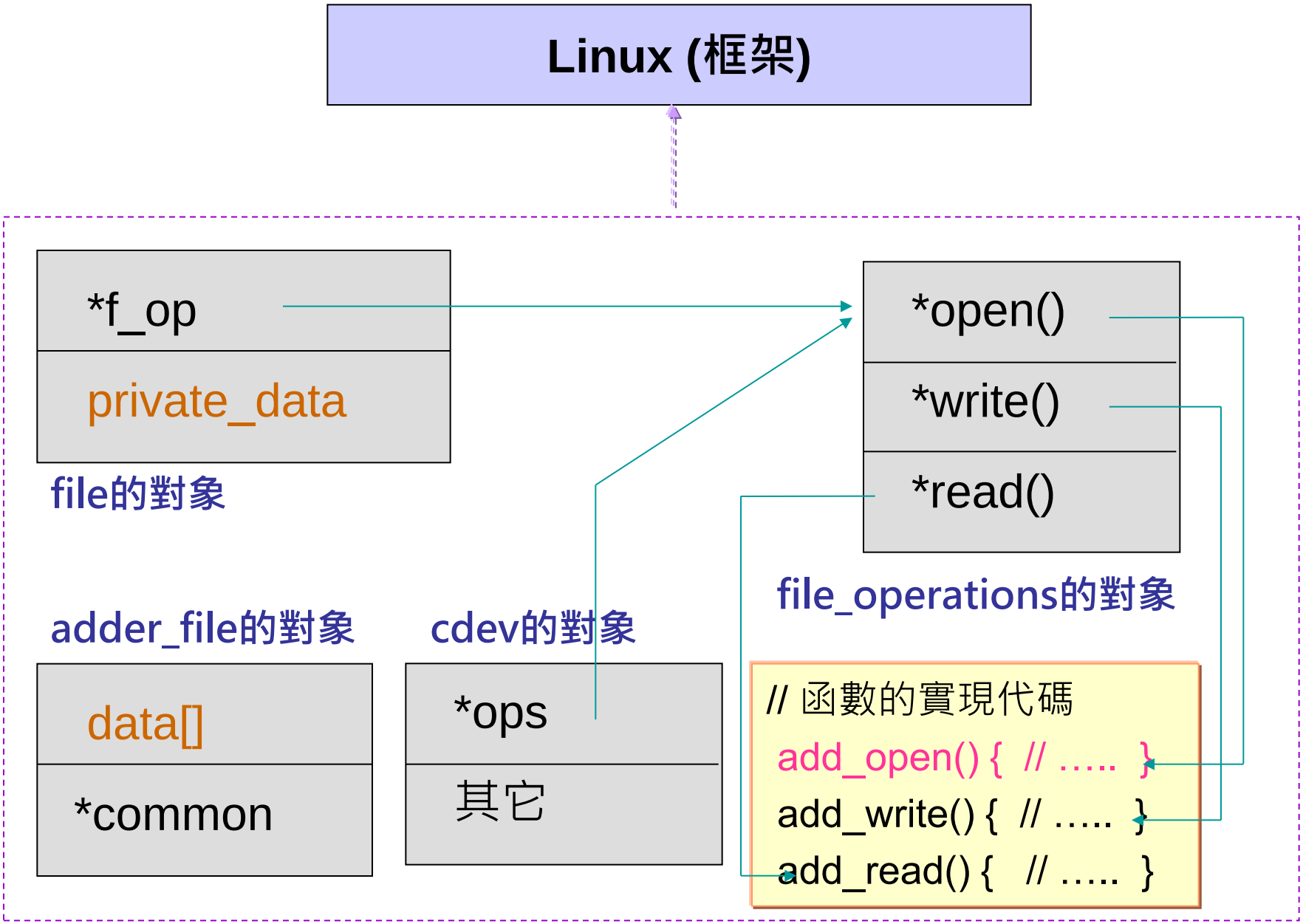
By 高煥堂

4、撰写用户态的应用程序

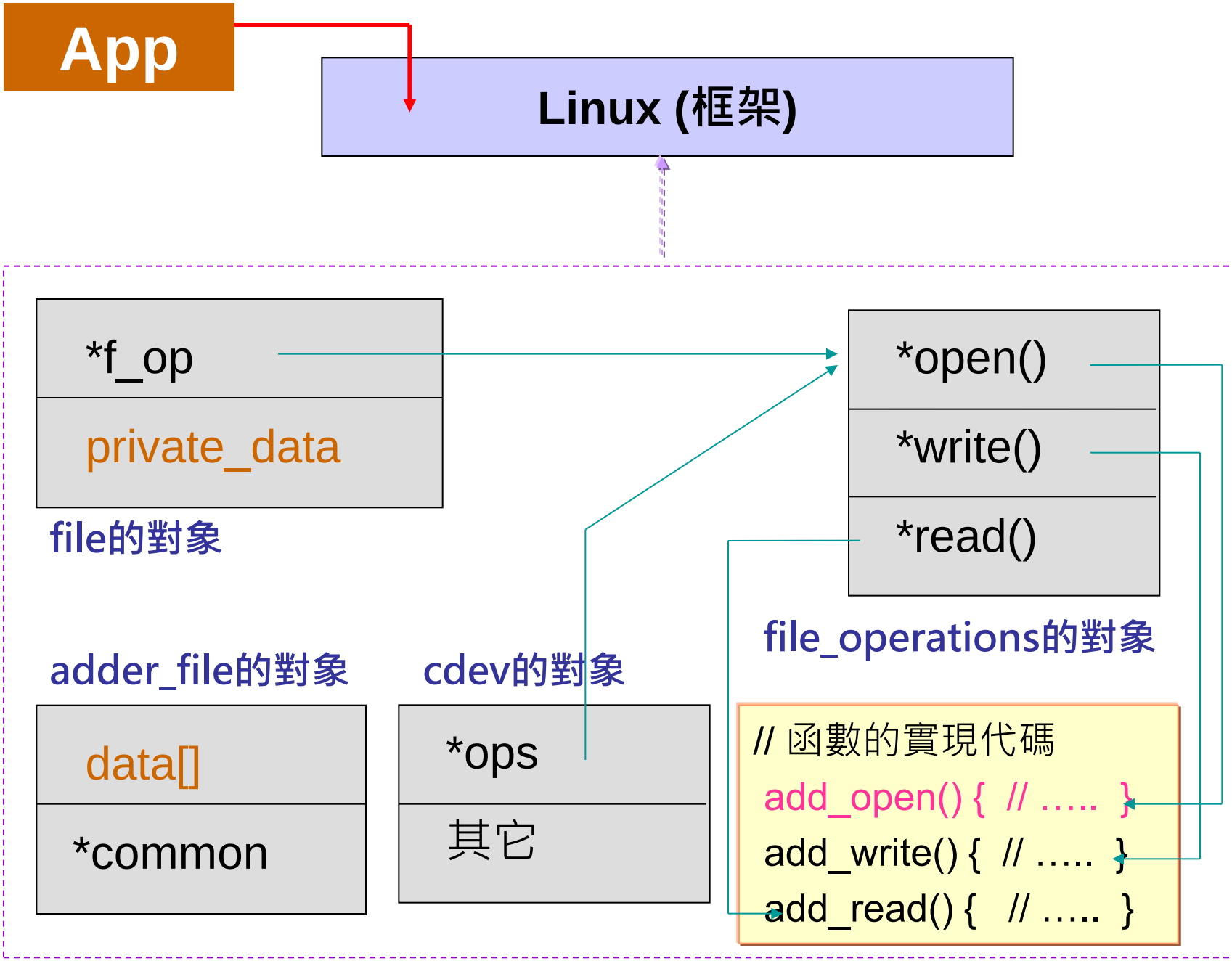
前言


<E&I>
Linux (框架)
<T>
Kernel-Driver模块


Linux (框架)
Kernel-Driver
adder_module
adder_file
创建对象&设定函数指针
创建对象
撰写函数的实现代码
函数代码(起始設定)


Linux (框架)
*f_op
private_data
file的對象
*open()
*write()
*read()
file_operations的對象
adder_file的對象
data[]
*common
cdev的對象
*ops
其它
// 函數的實現代碼
add_open() { // ..... }
add_write() { // ..... }
add_read() { // ..... }

撰写App


App
Linux (框架)
*f_op
private_data
file的對象
*open()
*write()
*read()
file_operations的對象
adder_file的對象
data[]
*common
cdev的對象
*ops
其它
// 函數的實現代碼
add_open() { // ..... }
add_write() { // ..... }
add_read() { // ..... }

<撰写App应用程序代码>

```
/* App应用程序 */
#include <stdio.h>
#include <fcntl.h>
#define DEVFILE "/dev/androidin"
#define BUFLEN 128

int main() {
    int fd = 0;
    int in[2] = {134, 2567};
```

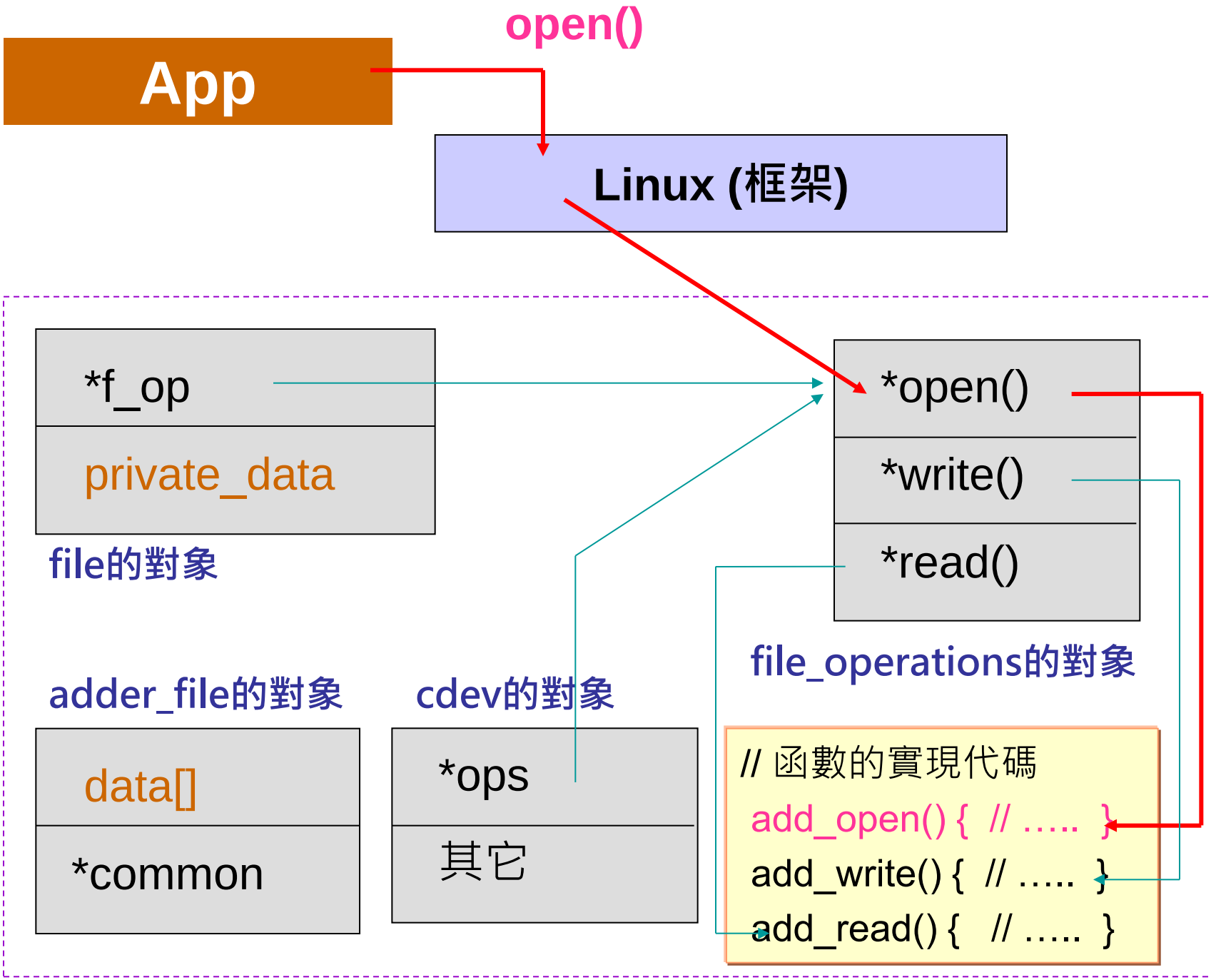
```
        int out = 0;
                    fd = open(DEVFILE, O_RDWR);
        if(fd == 0)
                   printf("open '/dev/add' failed!\n");
        printf("fd:%d\n", fd);
        write(fd, in, sizeof(in));
        read(fd, &out, sizeof(out));
        close(fd);

        printf("Input:%d %d\n", in[0], in[1]);
        printf("Output:%d\n", out);
        return 0;
}
```

- 执行到**App**的代码：

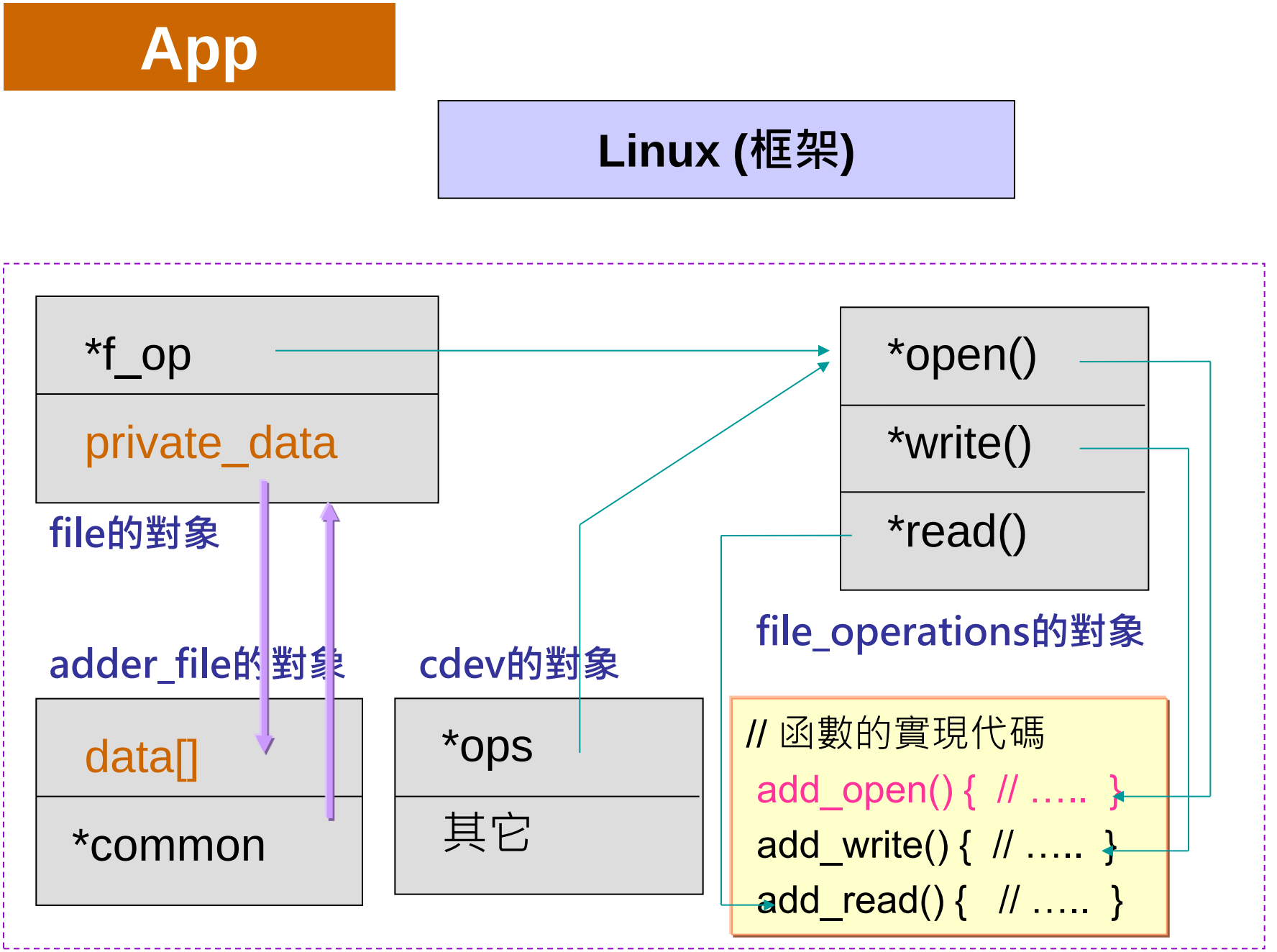
  fd = open(DEVFILE, O_RDWR);

- 此时，调用add_open()函数。


open()
App
Linux (框架)
*f_op
private_data
file的對象
*open()
*write()
*read()
file_operations的對象
adder_file的對象
data[]
*common
cdev的對象
*ops
其它
// 函數的實現代碼
add_open() { // ..... }
add_write() { // ..... }
add_read() { // ..... }

- 执行到add_open()函数的实现代码：

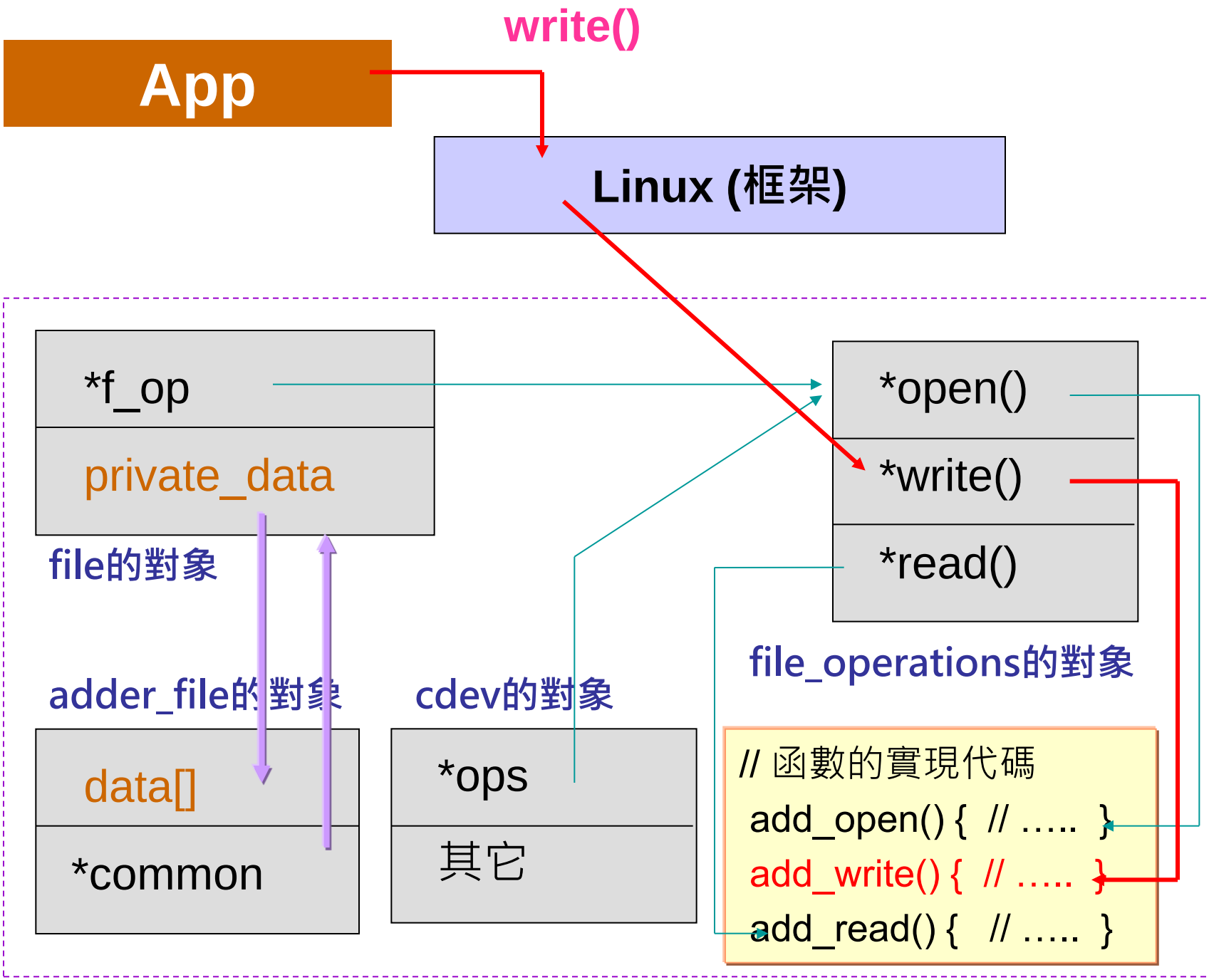
```
int add_open(struct inode *inode, struct file *filp){
        filp->private_data = &add_file;
        add_file.common = filp;
        return 0;
}
```


App
Linux (框架)
*f_op
private_data
file的對象
*open()
*write()
*read()
file_operations的對象
adder_file的對象
cdev的對象
data[]
*common
*ops
其它
// 函數的實現代碼
add_open() { // ..... }
add_write() { // ..... }
add_read() { // ..... }

- 接着，继续执行到**App**代码：

```
write(fd, in, sizeof(in));
// ……..
```

- 就调用add_write()函数。


write()
App
Linux (框架)
*f_op
private_data
file的對象
*open()
*write()
*read()
file_operations的對象
adder_file的對象
data[]
*common
cdev的對象
*ops
其它
// 函數的實現代碼
add_open() { // ..... }
add_write() { // ..... }
add_read() { // ..... }

- 执行到add_write()函数的实现代码:

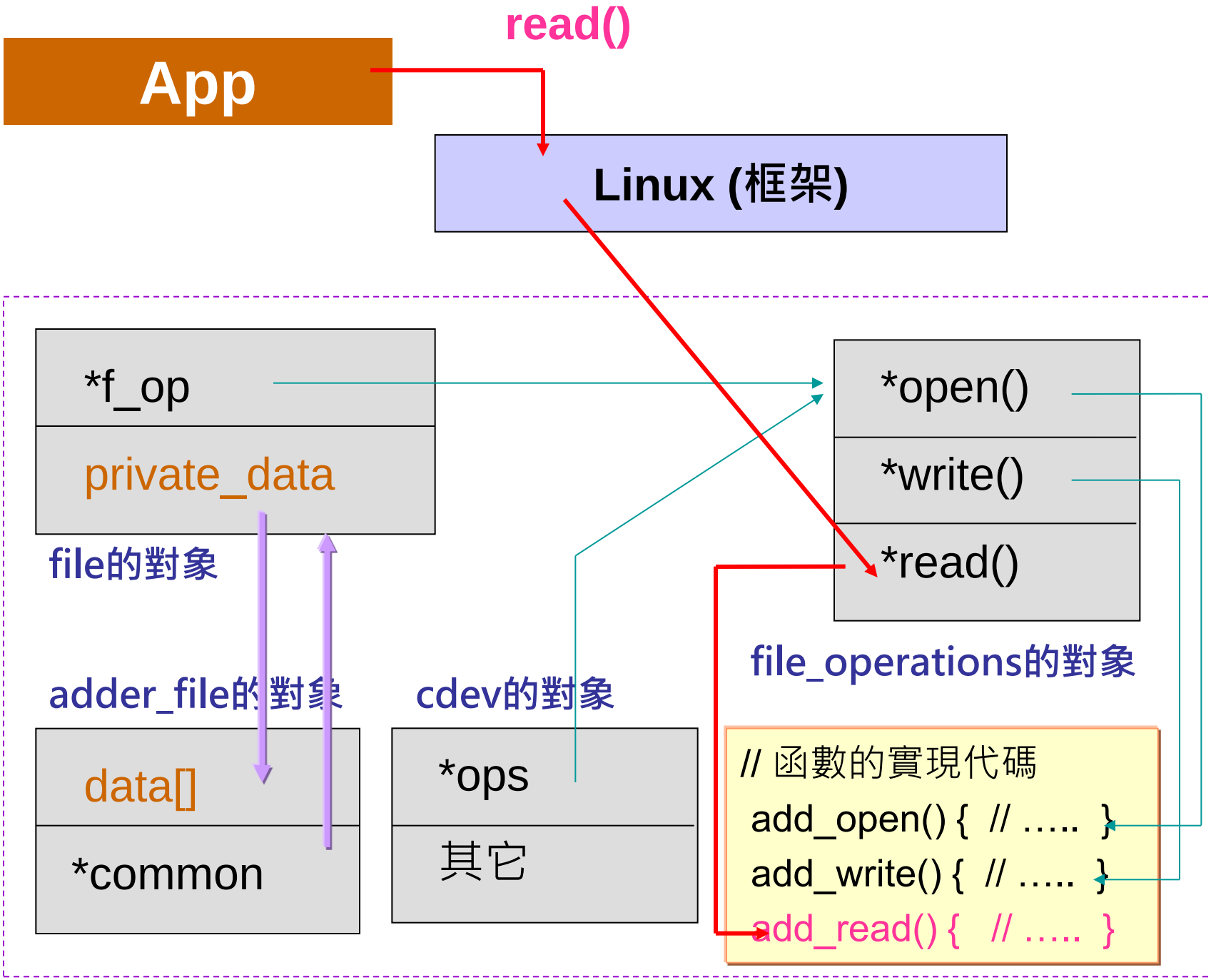
```
retval = copy_from_user(data, buf, count);
```

- 将App里的数据(存于buf内)考贝到add_file对象内的data变量里。

- 接着，继续执行到App代码：

  read(fd, &out, sizeof(out));

- 就调用到add_read()函数。


read()
App
Linux (框架)
*f_op
private_data
file的對象
*open()
*write()
*read()
file_operations的對象
adder_file的對象
data[]
*common
cdev的對象
*ops
其它
// 函數的實現代碼
add_open() { // ..... }
add_write() { // ..... }
add_read() { // ..... }

- 执行到add_read()函数的的实现代码：

```
int sum = data[0] + data[1];
//.....
retval = copy_to_user(buf, &sum, sizeof(int));
```

- 先进行加法运算，结果存于sum变量里。再将sum里的值考贝到App的buf里。

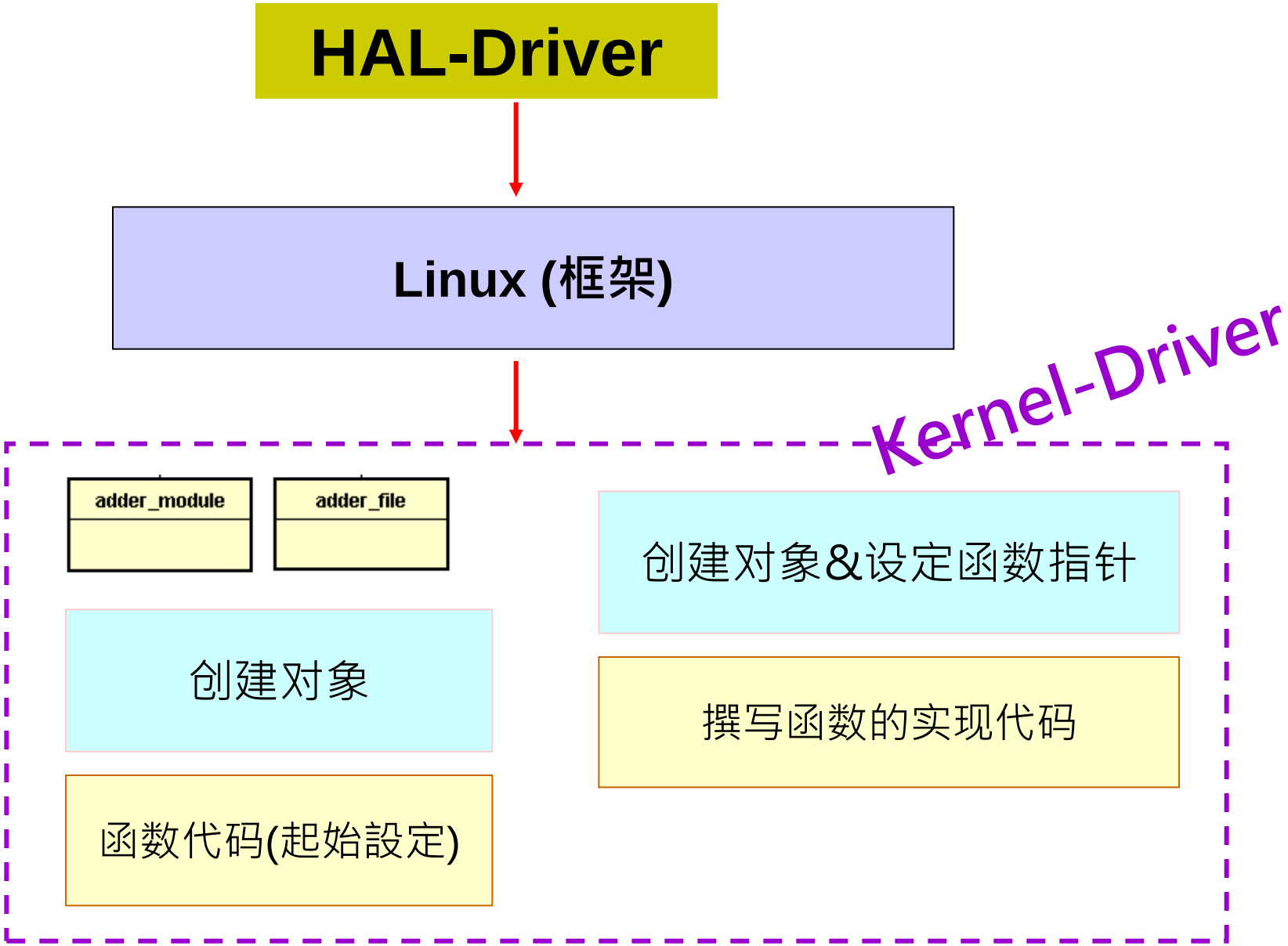
- 最后，继续执行到App代码：

```
printf("Input:%d %d\n", in[0], in[1]);
printf("Output:%d\n", out);
```

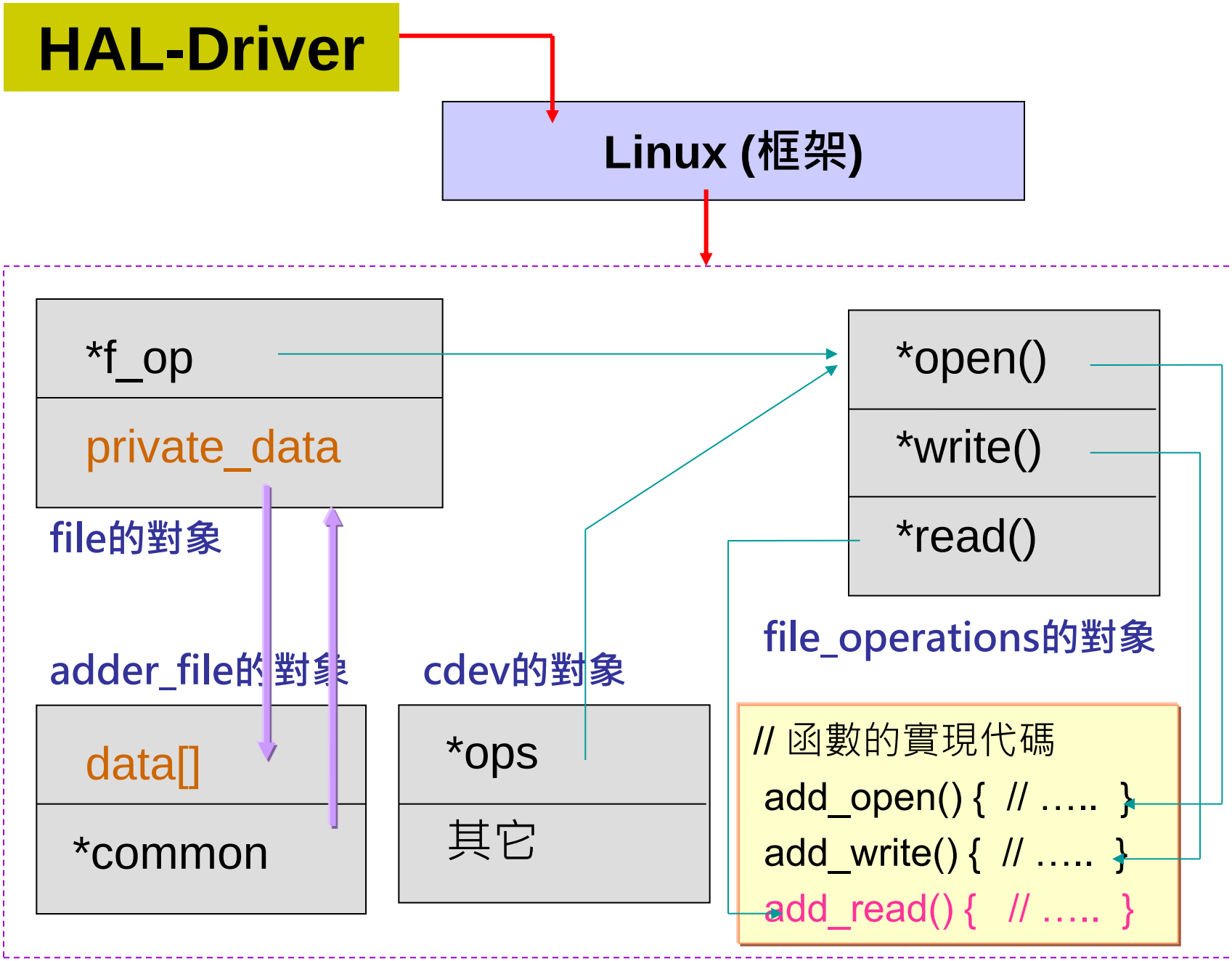
- 就将加法计算的结果打印出来了。

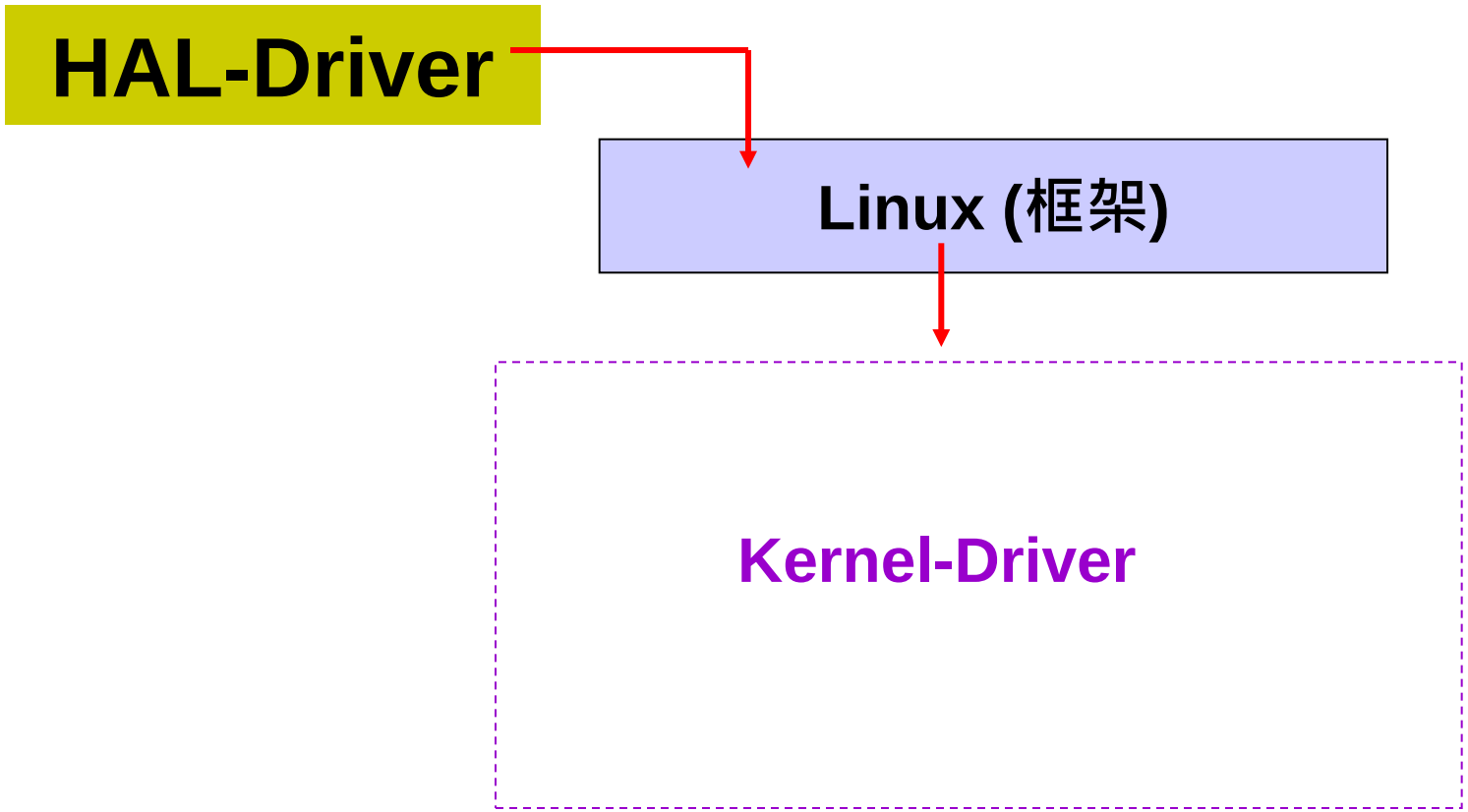
App也可以是HAL-Driver

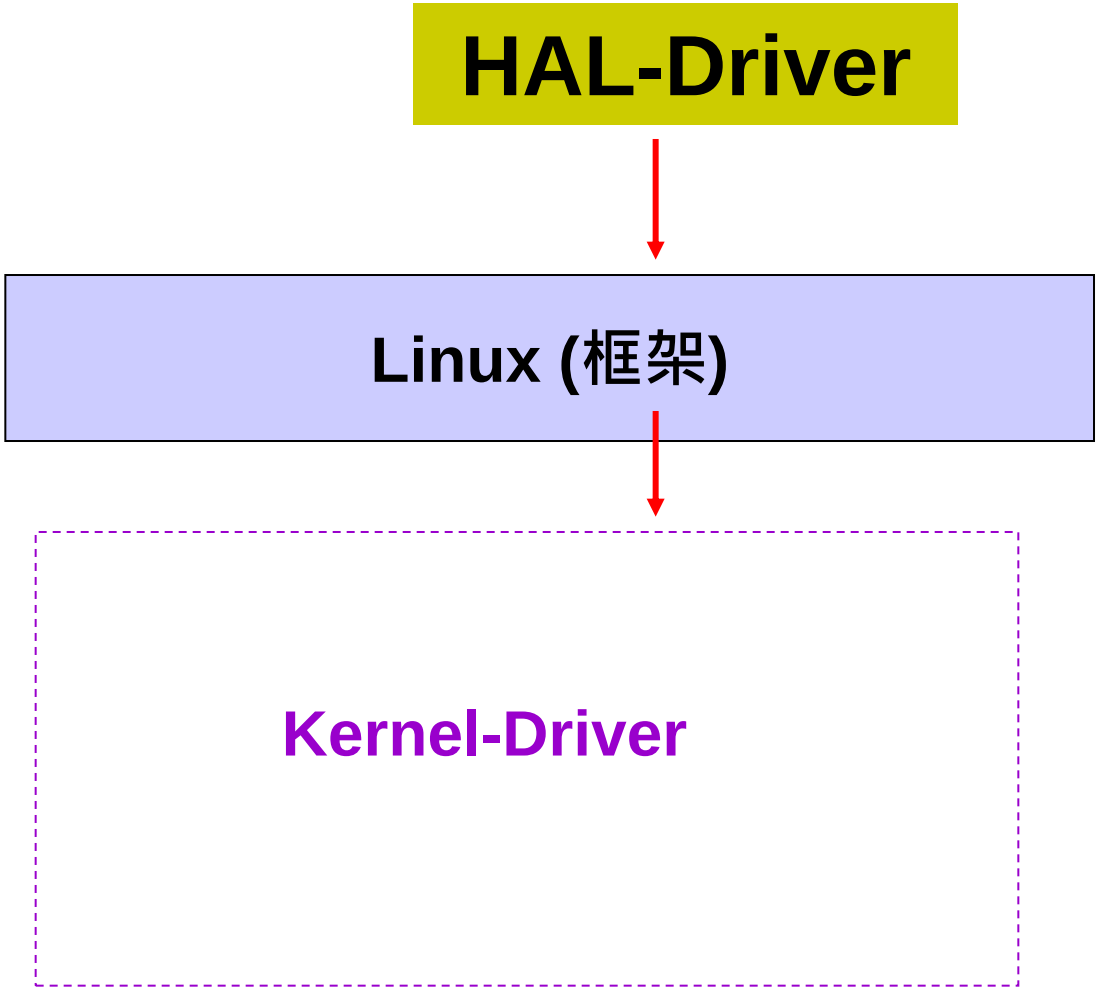
HAL-Drive调用Kernel-Driver

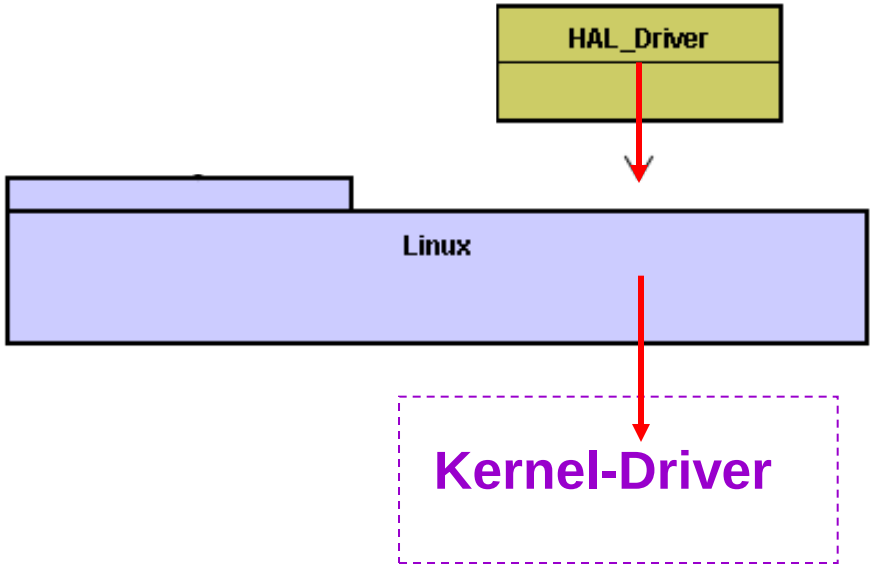

HAL-Driver
Linux (框架)
Kernel-Driver
adder_module
adder_file
创建对象&设定函数指针
创建对象
撰写函数的实现代码
函数代码(起始設定)

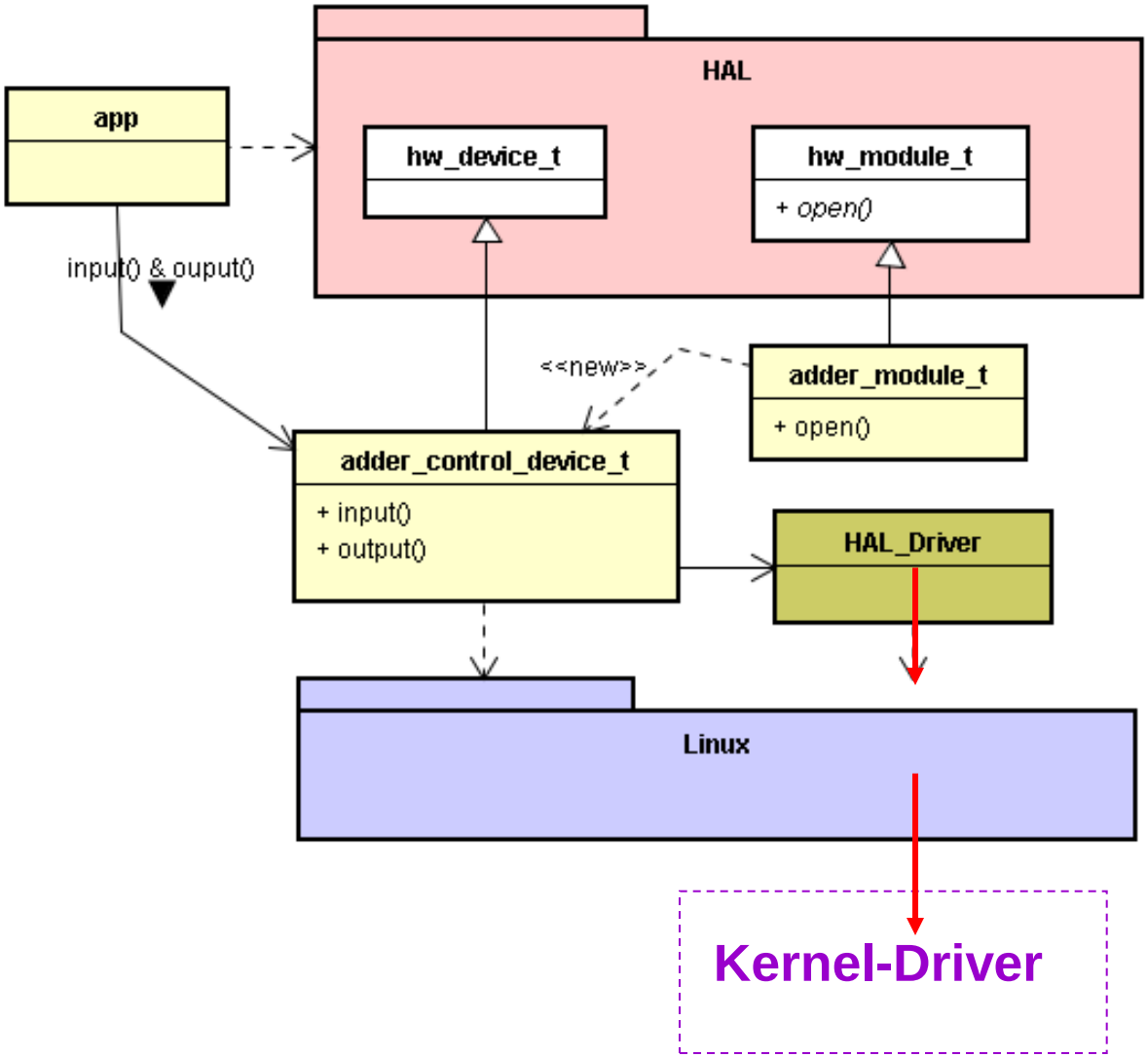
At run-time

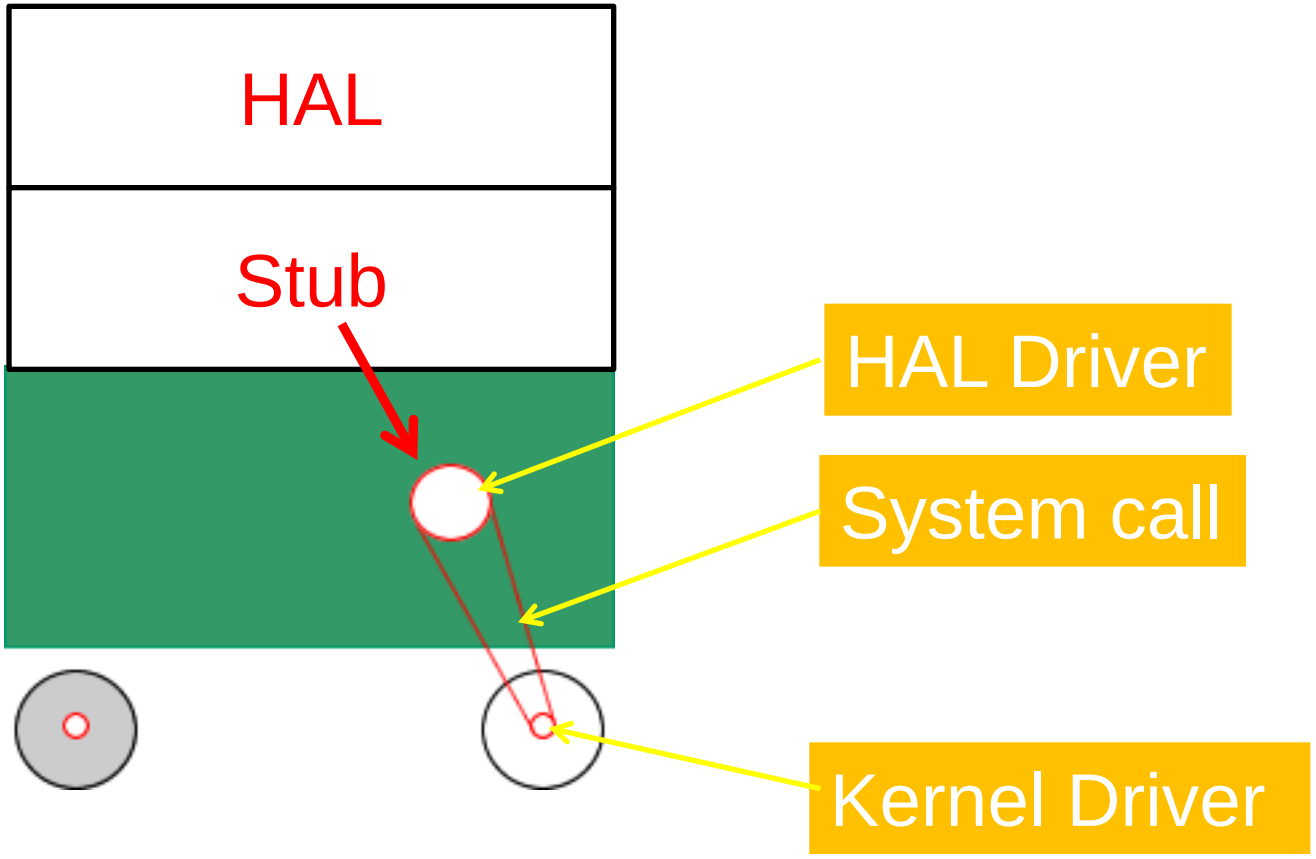

HAL-Driver
Linux (框架)
*f_op
private_data
file的對象
*open()
*write()
*read()
file_operations的對象
adder_file的對象
cdev的對象
data[]
*common
*ops
其它
// 函數的實現代碼
add_open() { // ..... }
add_write() { // ..... }
add_read() { // ..... }


HAL-Driver
Linux (框架)
Kernel-Driver


HAL-Driver
Linux (框架)
Kernel-Driver


HAL_Driver
Linux
Kernel-Driver


HAL
app
hw_device_t
hw_module_t
+ open()
input() & ouput()
<<new>>
adder_module_t
+ open()
adder_control_device_t
+ input()
+ output()
HAL_Driver
Linux
Kernel-Driver


HAL
Stub
HAL Driver
System call
Kernel Driver

Thanks...

高煥堂