自己動手，豐衣足食
淺談探索 Linux 系統設計之道

Jim Huang(黃敬群) “jserv”
blog: http://blog.linux.org.tw/jserv/
From 0xlab - http://0xlab.org/

Study-Area / June 5, 2010

授權聲明

creative commons
COMMONS DEED

Attribution – ShareAlike 2.0

You are free

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions

- **Attribution**. You must give the original author credit.
- **Share Alike**. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.
- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text:
http://creativecommons.org/licenses/by-sa/2.0/legalcode

參考組態

- Ubuntu Linux 10.04 / 10.10
  - Kernel: 2.6.32-15-generic
  - gcc: 4.4.4
  - glibc: 2.11.1
- Lenovo ThinkPad X200
  - Intel Core2 Duo CPU 2.4 GHz

Agenda

- Linux 核心設計概念
- Rosetta Stone： Linux 如何建立軟硬體關聯
- Linux 驅動程式架構與發展
- 尋幽訪勝自己來

Linux 的核心設計概念

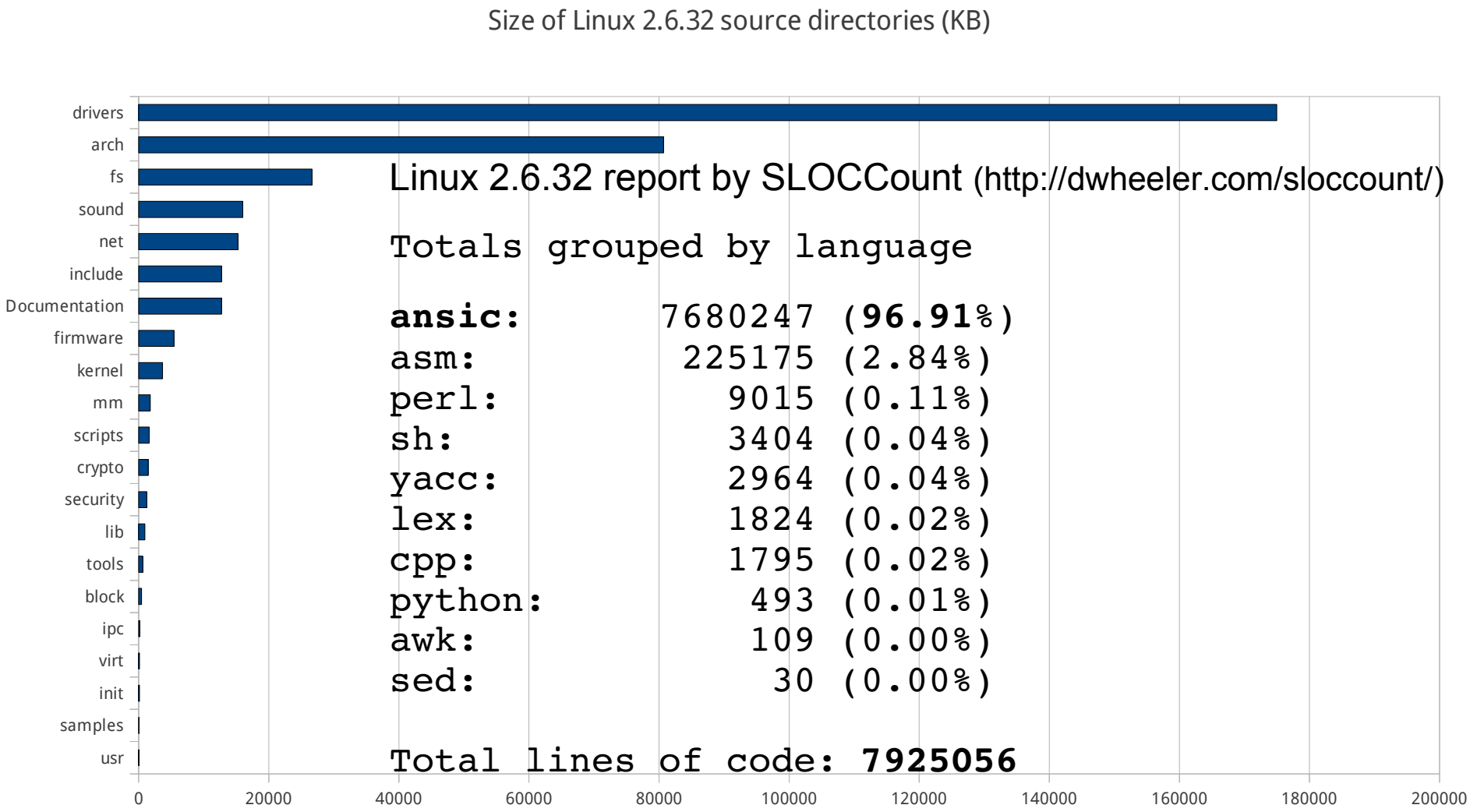
[概念I]
以C語言建構

C 語言精髓： *Pointer*

- Pointer 也就是記憶體 (Memory) 存取的替身
- 重心： Linux → Pointer → Memory

- Linux 跟其他 UNIX 一樣，採用 C 作為主要開發語言，硬體相關部份則透過組合語言
- 不採用 C++ 的緣故可見 http://www.tux.org/lkml/#s15-3
  主要考量點：**效率**

Linux kernel size



爲了效率，充斥了奇計淫 ^H^H^H 技巧

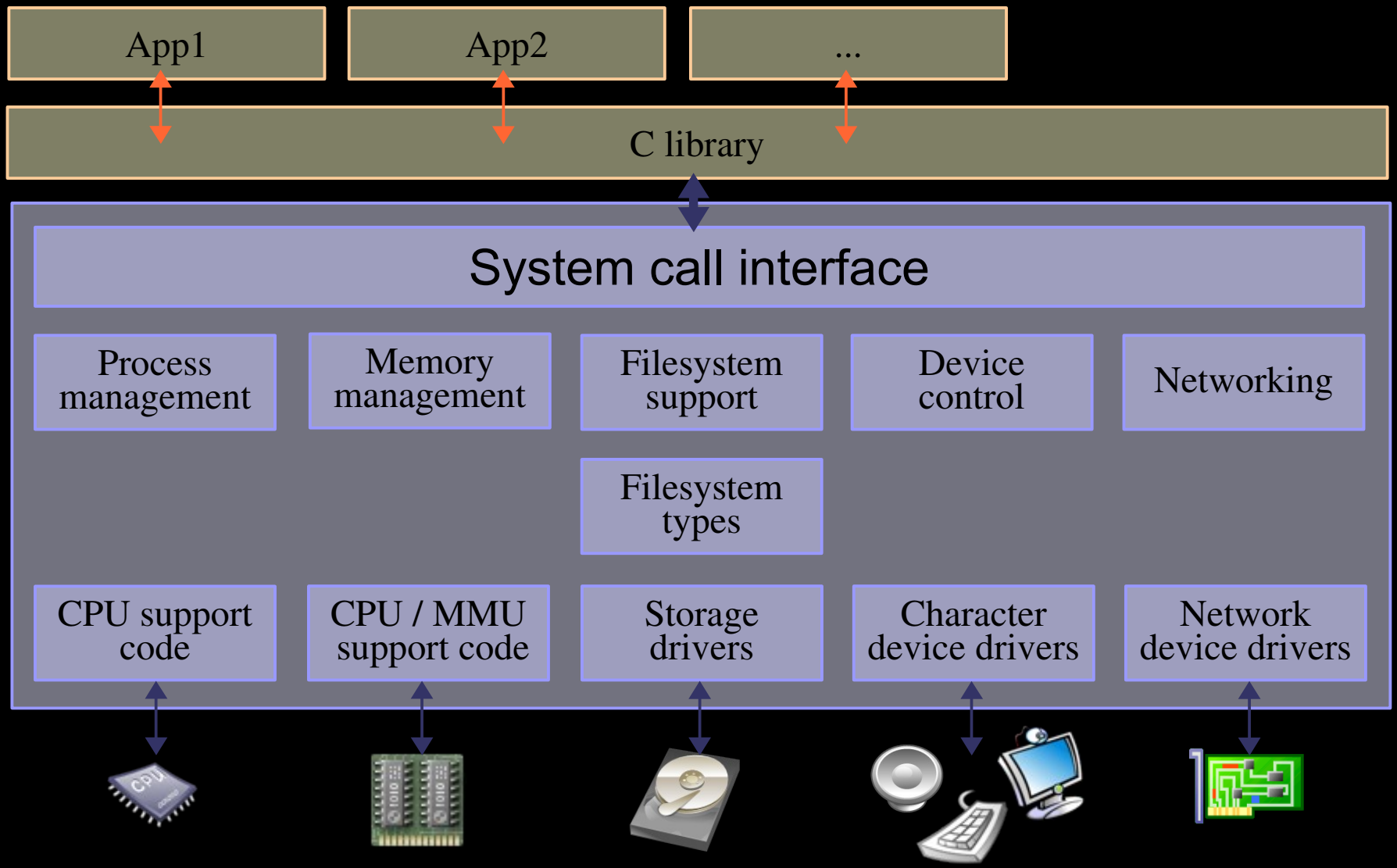
- Linux 核心依賴 GNU gcc 特有擴充
- 對編譯器作大量提示，如 likely/unlikely 擴充

```
if (unlikely(err)) {
        ...
}
```

- 平台特有優化

［概念II］
多層、多人、多工

核心架構


App1
App2
...
C library
System call interface
Process management
Memory management
Filesystem support
Device control
Networking
Filesystem types
CPU support code
CPU / MMU support code
Storage drivers
Character device drivers
Network device drivers

“From a technical standpoint, I believe **the kernel will be "more of the same"** and that all the **_really_** interesting staff will be going out in **user space.**”

-- Linus Torvalds

October 2001

[概念 III]

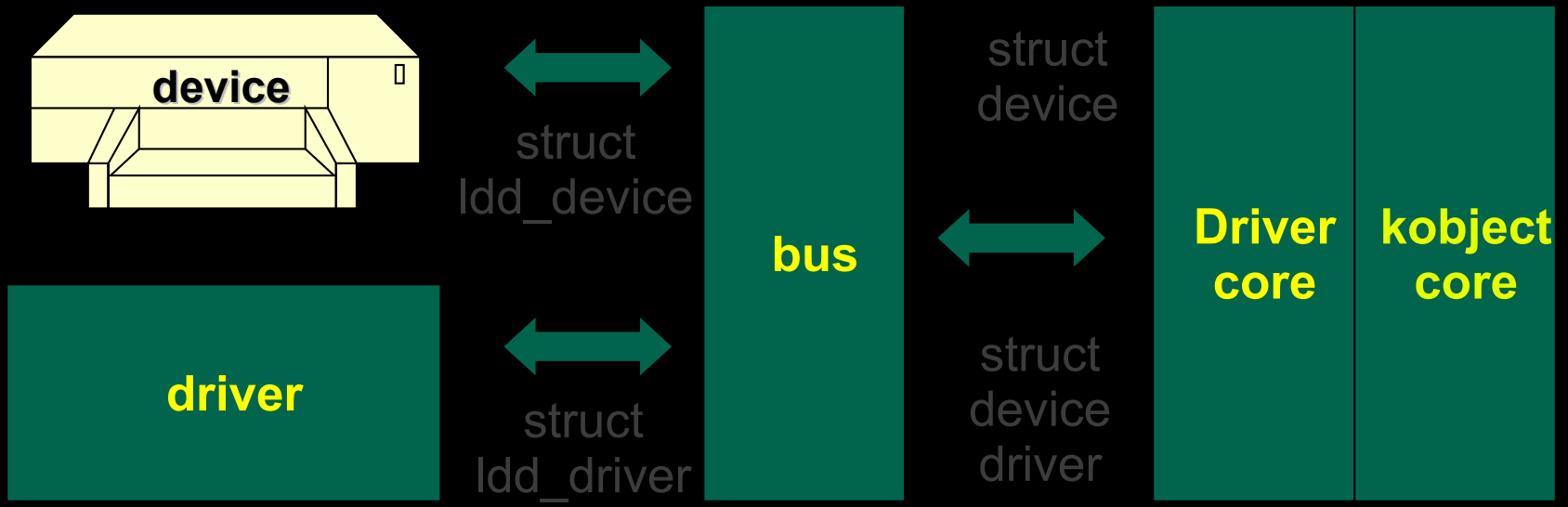
Linux 核心充斥大量的物件導向設計

kobject: ADT (Abstract Data Typing)

device driver: Inheritance/polymorphism

hotplug(), probe(): dynamic binding

- Buses：處理器與一個或多個裝置間的通道
- Devices 與對應的 device drivers



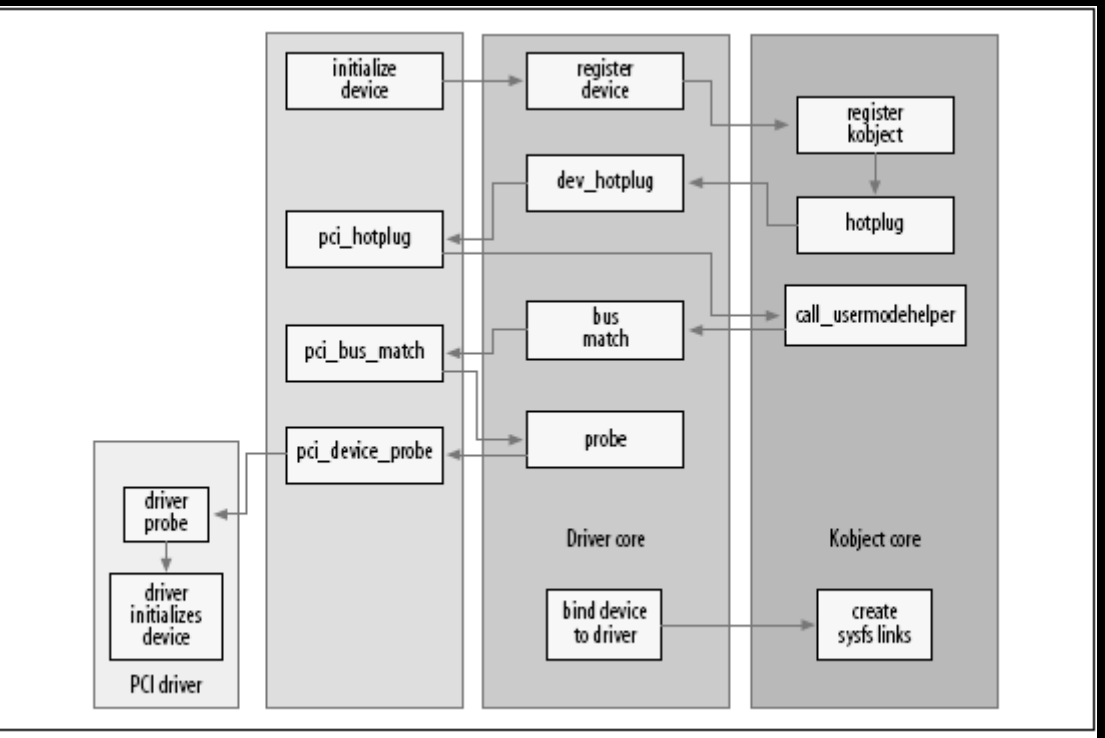


Figure 14-3. Device-creation process

Kobject 的繼承關係

- **parent** pointer 與 **ksets**
  - “parent” points to another kobject, representing the next level up
  - “kset” is a collection of kobjects
  - kset are always represented in sysfs
  - Every kobject that is a member of a kset is represented in sysfs

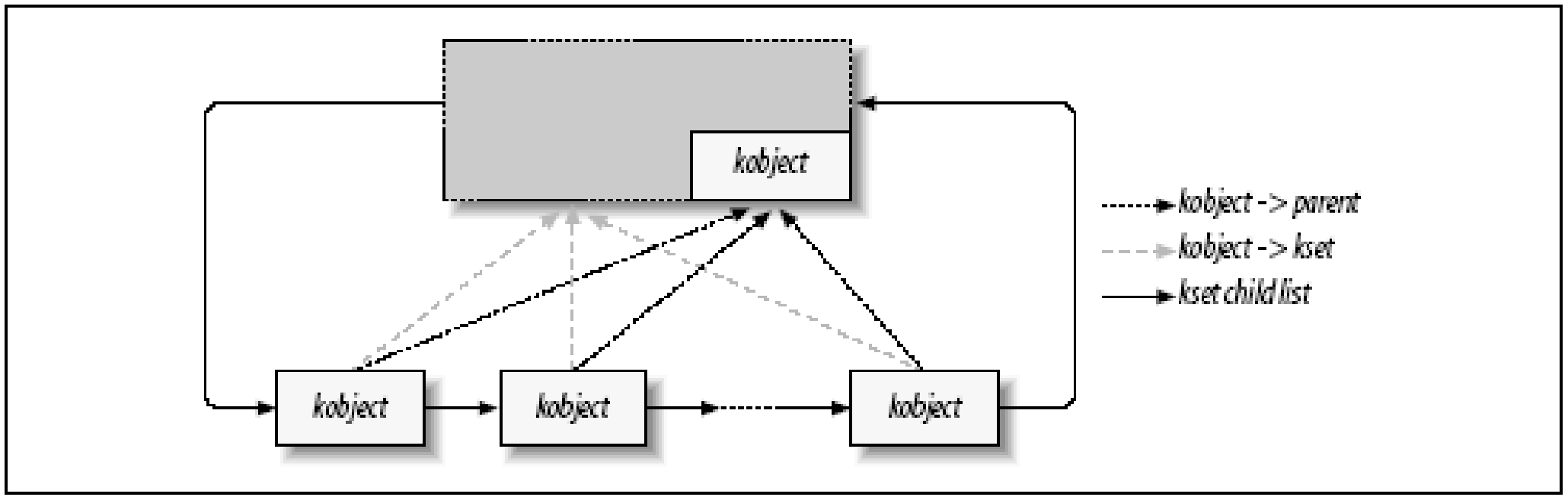
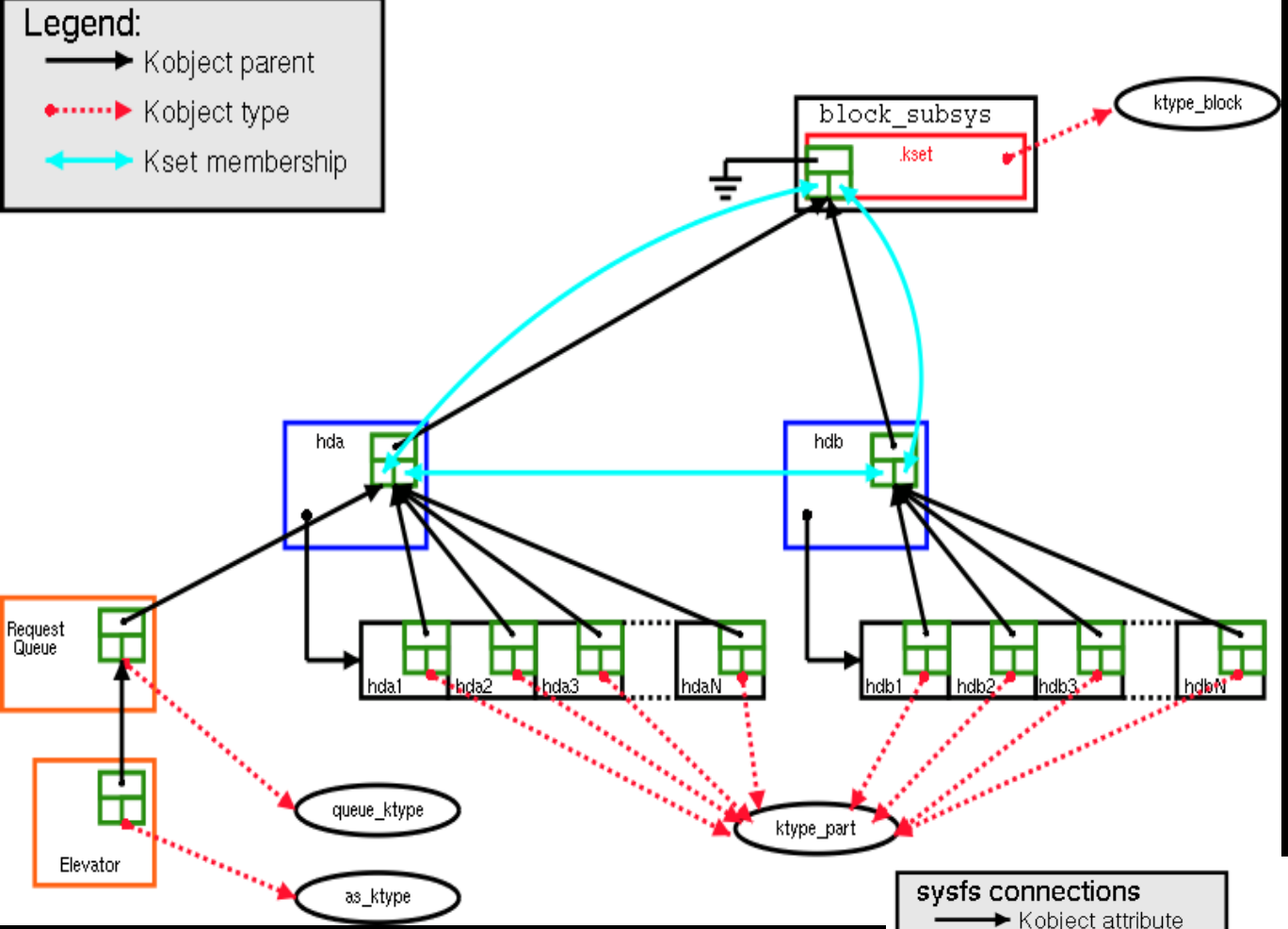
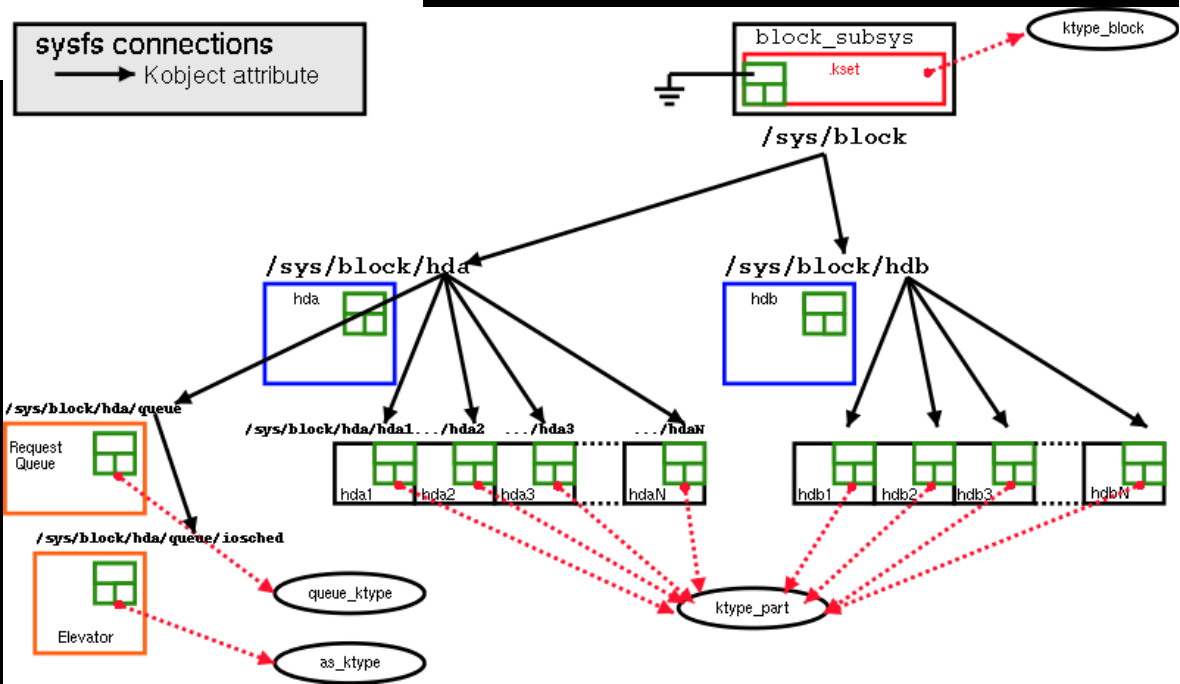


Figure 14-2. A simple kset hierarchy

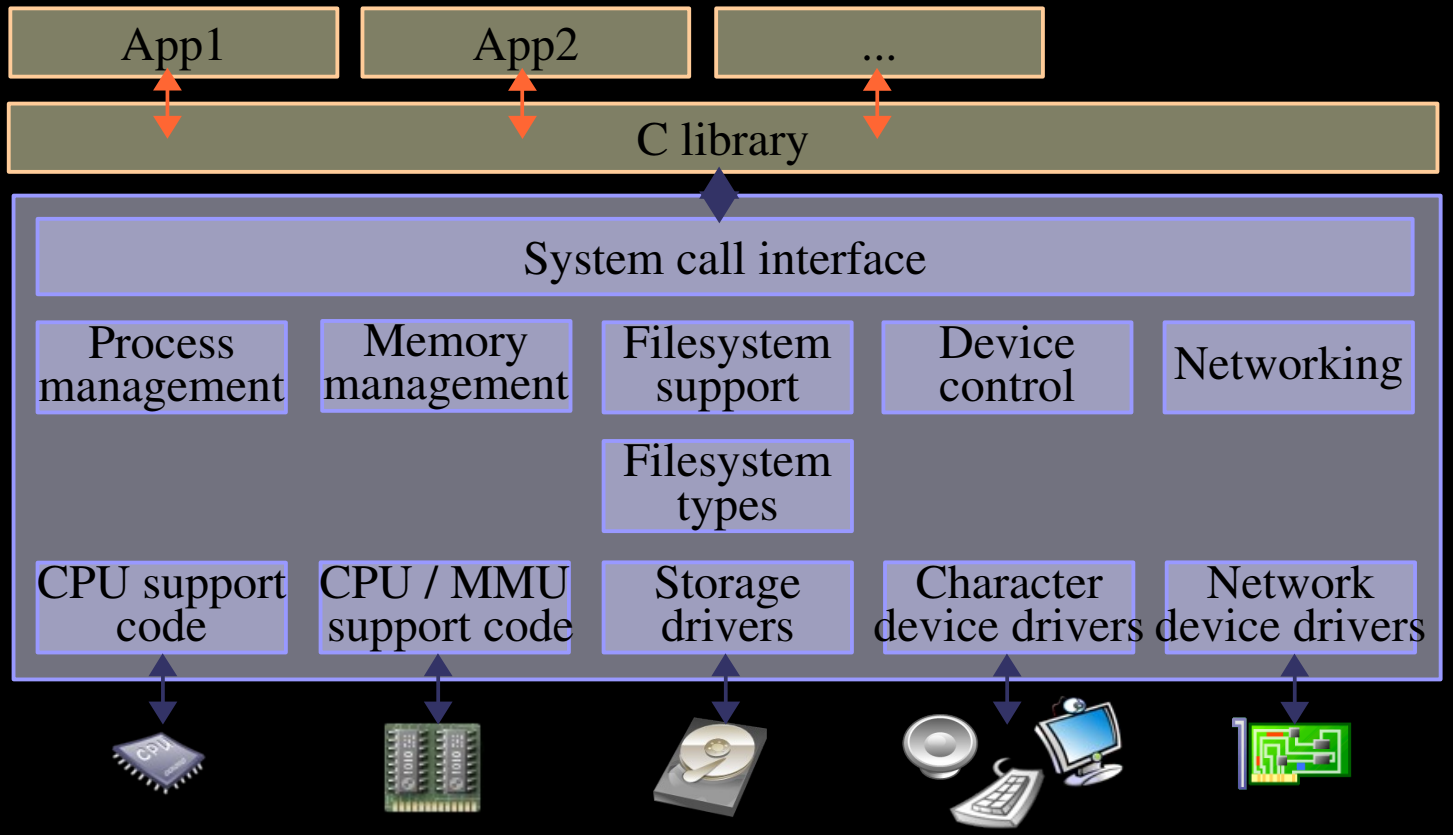

Legend:
Kobject parent
Kobject type
Kset membership
block_subsys
.kset
ktype_block
hda
hdb
Request Queue
hda1
hda2
hda3
hdaN
hdb1
hdb2
hdb3
hdbN
Elevator
queue_ktype
as_ktype
ktype_part


sysfs connections
Kobject attribute
block_subsys
.kset
ktype_block
/sys/block
/sys/block/hda
/sys/block/hdb
hda
hdb
/sys/block/hda/queue
/sys/block/hda/hda1.../hda2
.../hda3
.../hdaN
Request Queue
hda1
hda2
hda3
hdaN
hdb1
hdb2
hdb3
hdbN
/sys/block/hda/queue/iosched
Elevator
queue_ktype
as_ktype
ktype_part

[概念 IV]

Everything is **file**.

核心與週邊

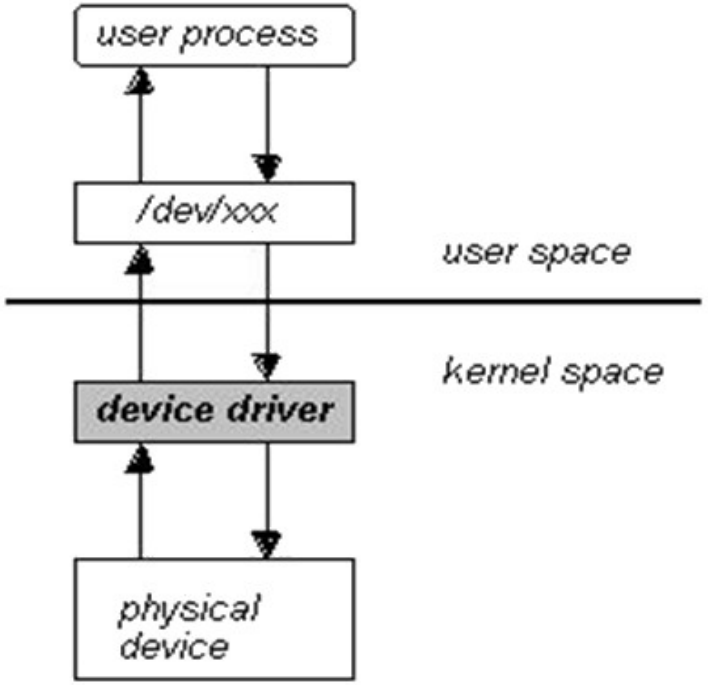


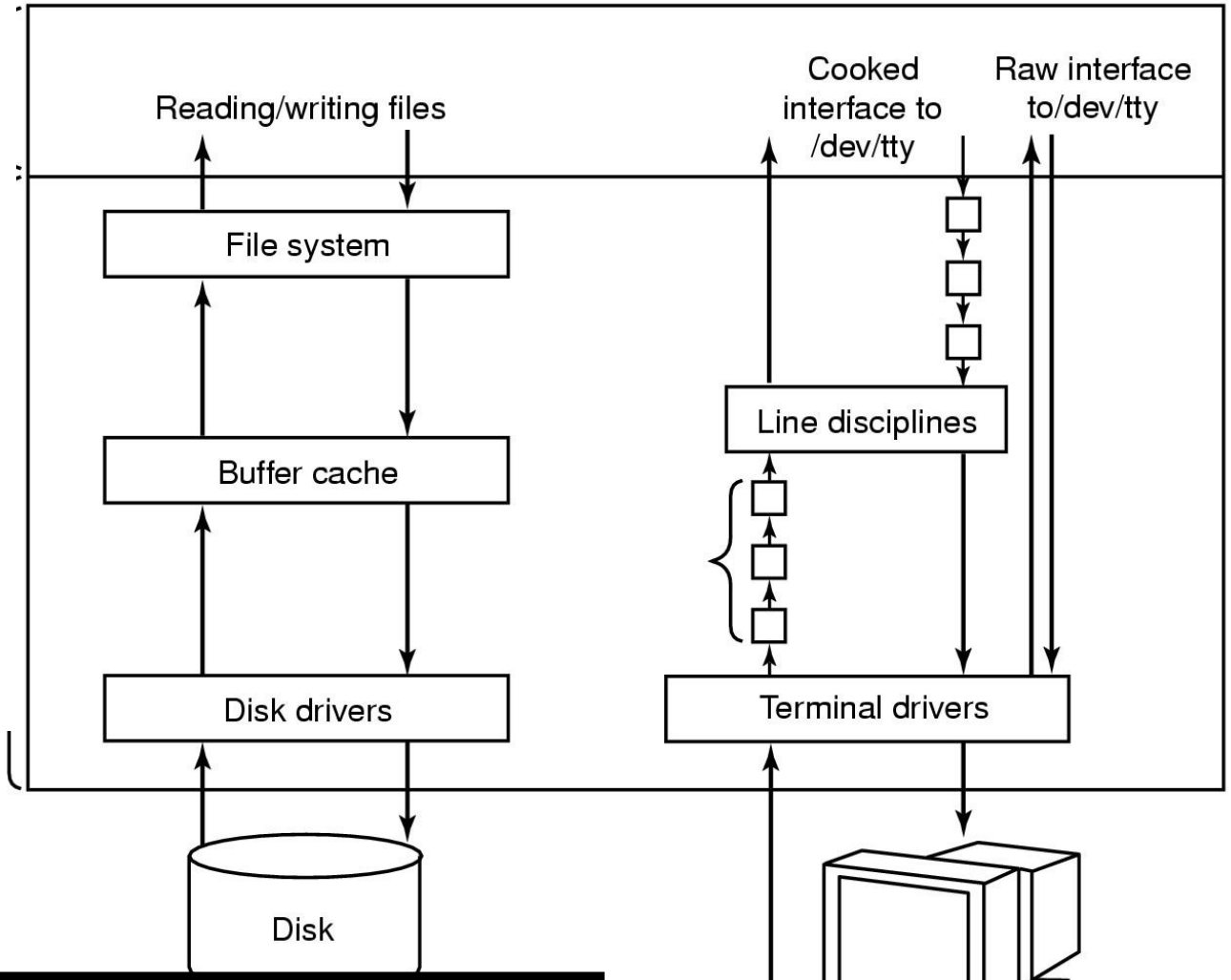
UNIX 法則：“Everything is file”

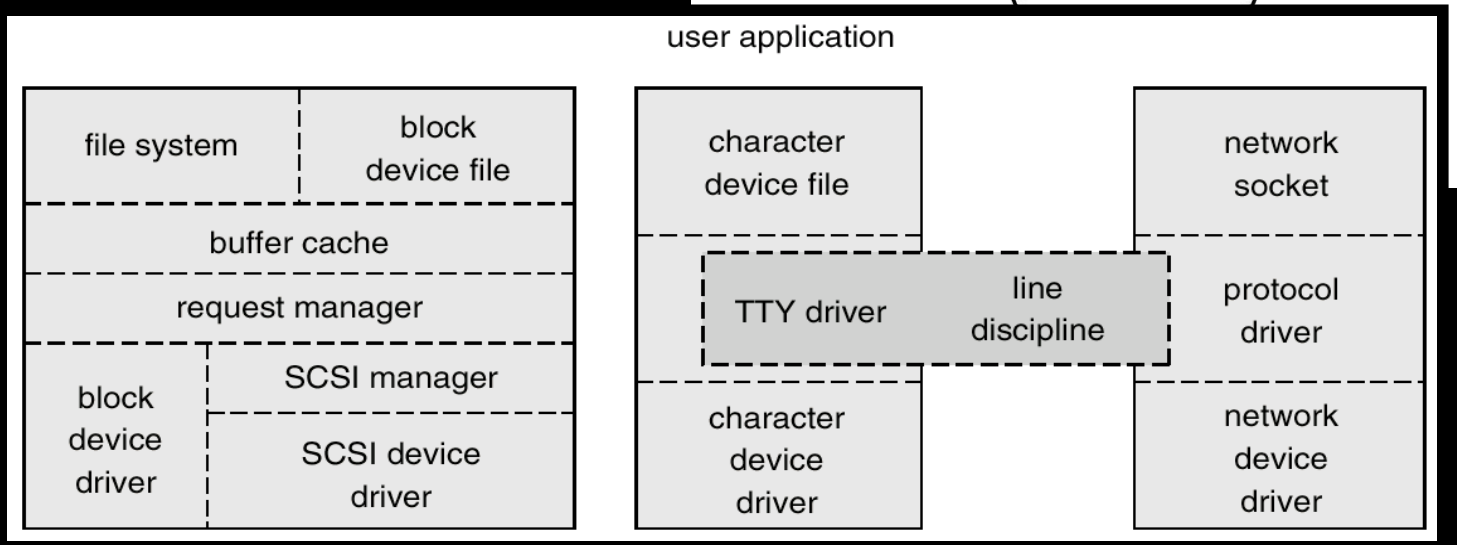
- 實體記憶體 (/dev/mem) 、網路 (socket) 、實體 / 虛擬裝置、系統資訊、驅動程式的控制、週邊組態、 . . .

Linux Device Driver 的角色即是將 file operation 映射到 Device

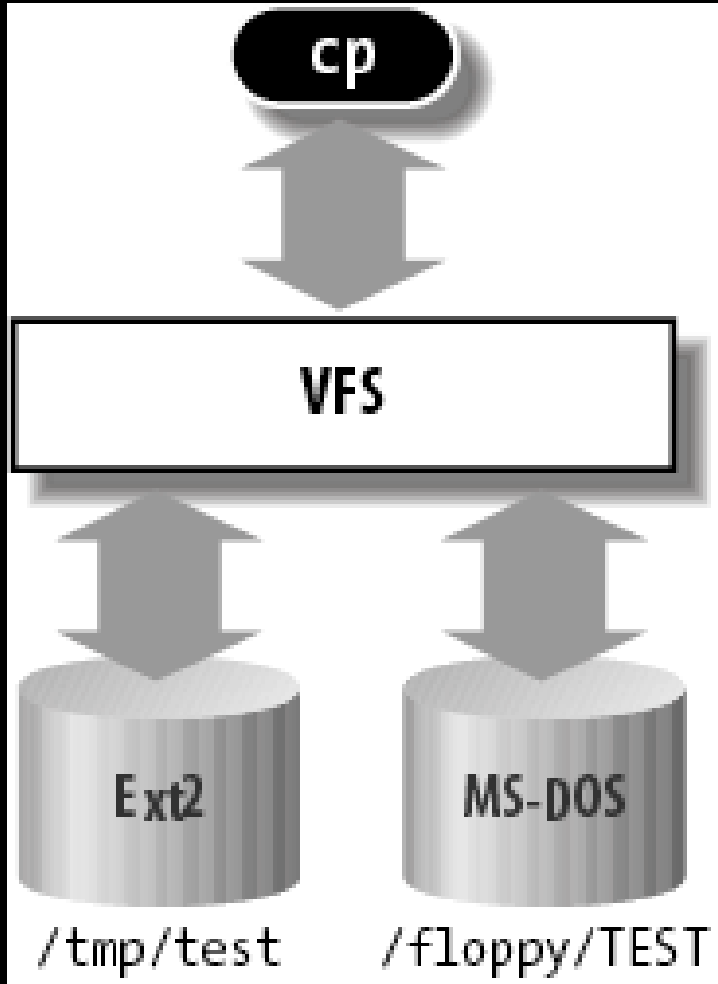
- 有明確階層概念
- 不限於 kernel-space driver
- 經典的 user-space driver 如 X11 video driver


user process
/dev/xxx
user space
kernel space
device driver
physical device


Reading/writing files
Cooked interface to /dev/tty
Raw interface to/dev/tty
File system
Buffer cache
Disk drivers
Line disciplines
Terminal drivers
Disk


user application
file system
block device file
buffer cache
request manager
block device driver
SCSI manager
SCSI device driver
character device file
TTY driver
line discipline
character device driver
network socket
protocol driver
network device driver

VFS(虛擬檔案系統)扮演的角色

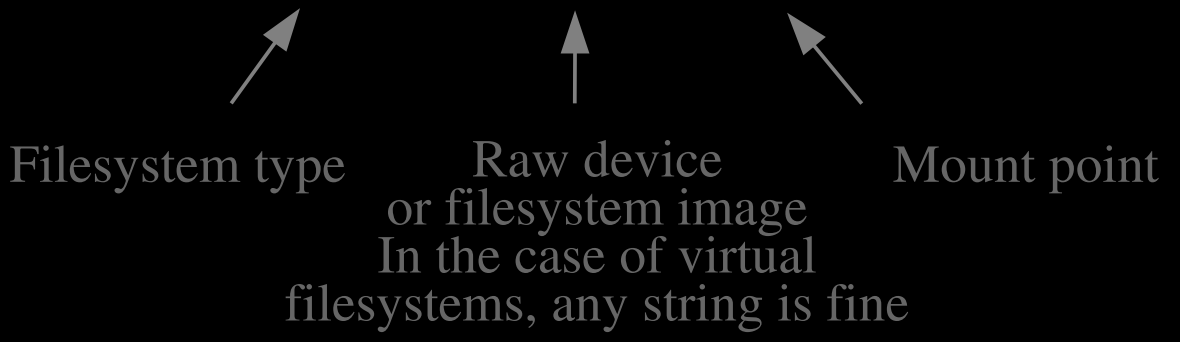



```
inf = open("/floppy/TEST", O_RDONLY, 0);
outf = open("/tmp/test",
       O_WRONLY|O_CREAT|O_TRUNC, 0600);
do {
    i = read(inf, buf, 4096);
    write(outf, buf, i);
} while (i);
close(outf);
close(inf);
```

(b)

掛載 VFS

- Mounting `/proc`:
  `sudo mount -t proc none /proc`
- Mounting `/sys`:
  `sudo mount -t sysfs none /sys`



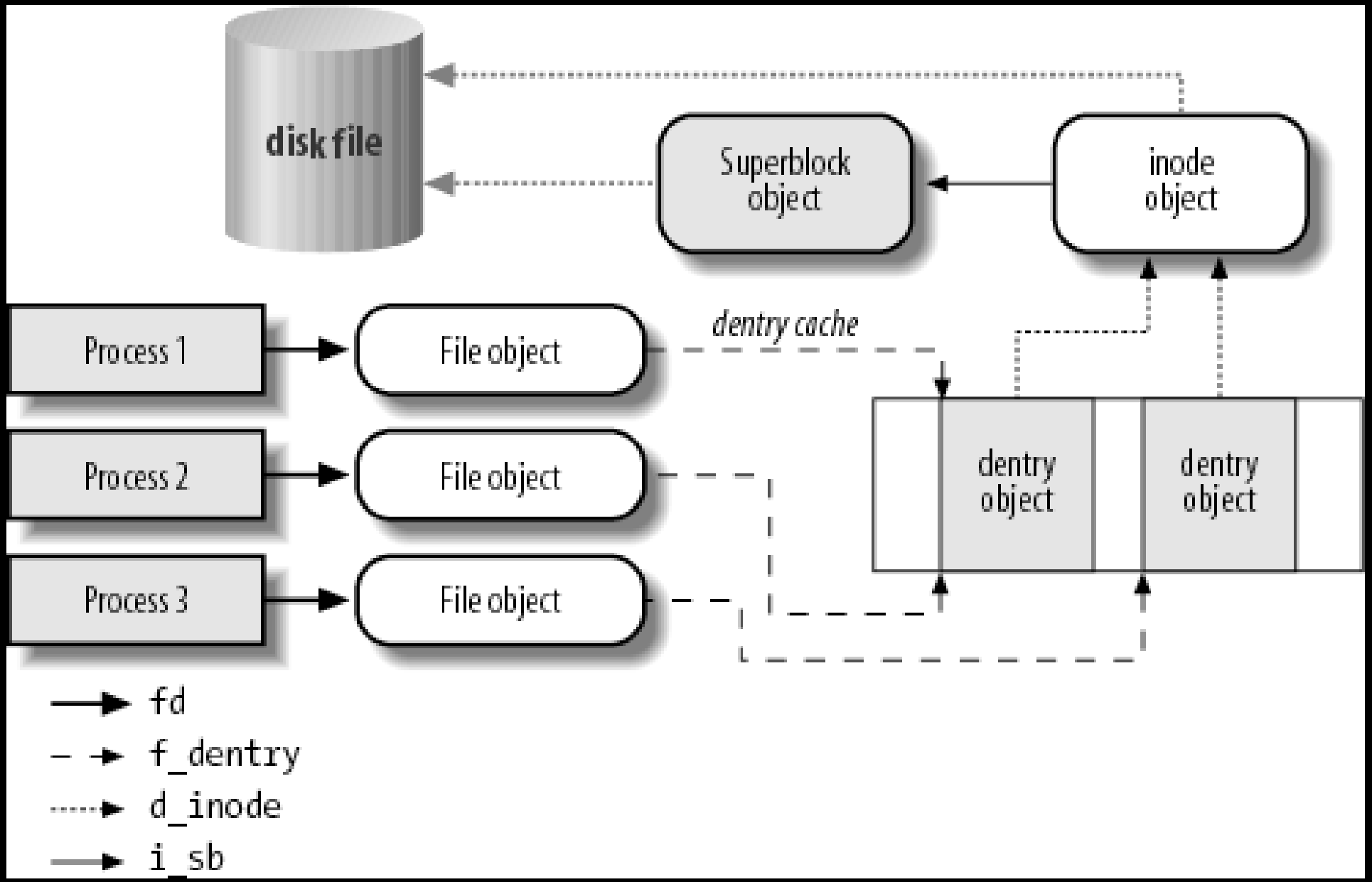
`/proc/cpuinfo`: processor information

`/proc/meminfo`: memory status

`/proc/version`: kernel version and build information

`/proc/cmdline`: kernel command line

程序與 VFS 之間的互動



debugfs

用以揭露核心資訊的虛擬檔案系統

- 核心組態: **DEBUG_FS**
  Kernel hacking -> Debug Filesystem
- 掛載方式:
  sudo mount -t debugfs none /debug
- http://lwn.net/Articles/334068/
- 搭配 Ftrace (CONFIG_FUNCTION_TRACER)

```
# cat /debug/tracing/available_tracers
blk function_graph function sched_switch nop
# echo function > /debug/tracing/current_tracer
# echo 1 >/debug/tracing/tracing_on
(進行某些動作)
# echo 0 >/debug/tracing/tracing_on
```

Ftrace

```
# head /debug/tracing/trace
# tracer: function
#
#   TASK-PID    CPU#    TIMESTAMP  FUNCTION
#      | |       |          |         |
    Xorg-1006  [000]  3055.314290: _cond_resched <-copy_from_user
    Xorg-1006  [000]  3055.314291: i915_gem_sw_finish_ioctl <-drm_ioctl
    Xorg-1006  [000]  3055.314291: mutex_lock <-i915_gem_sw_finish_ioctl
    Xorg-1006  [000]  3055.314291: _cond_resched <-mutex_lock
    Xorg-1006  [000]  3055.314291: drm_gem_object_lookup <-i915_gem_sw_finish_ioctl
    Xorg-1006  [000]  3055.314291: _spin_lock <-drm_gem_object_lookup
```

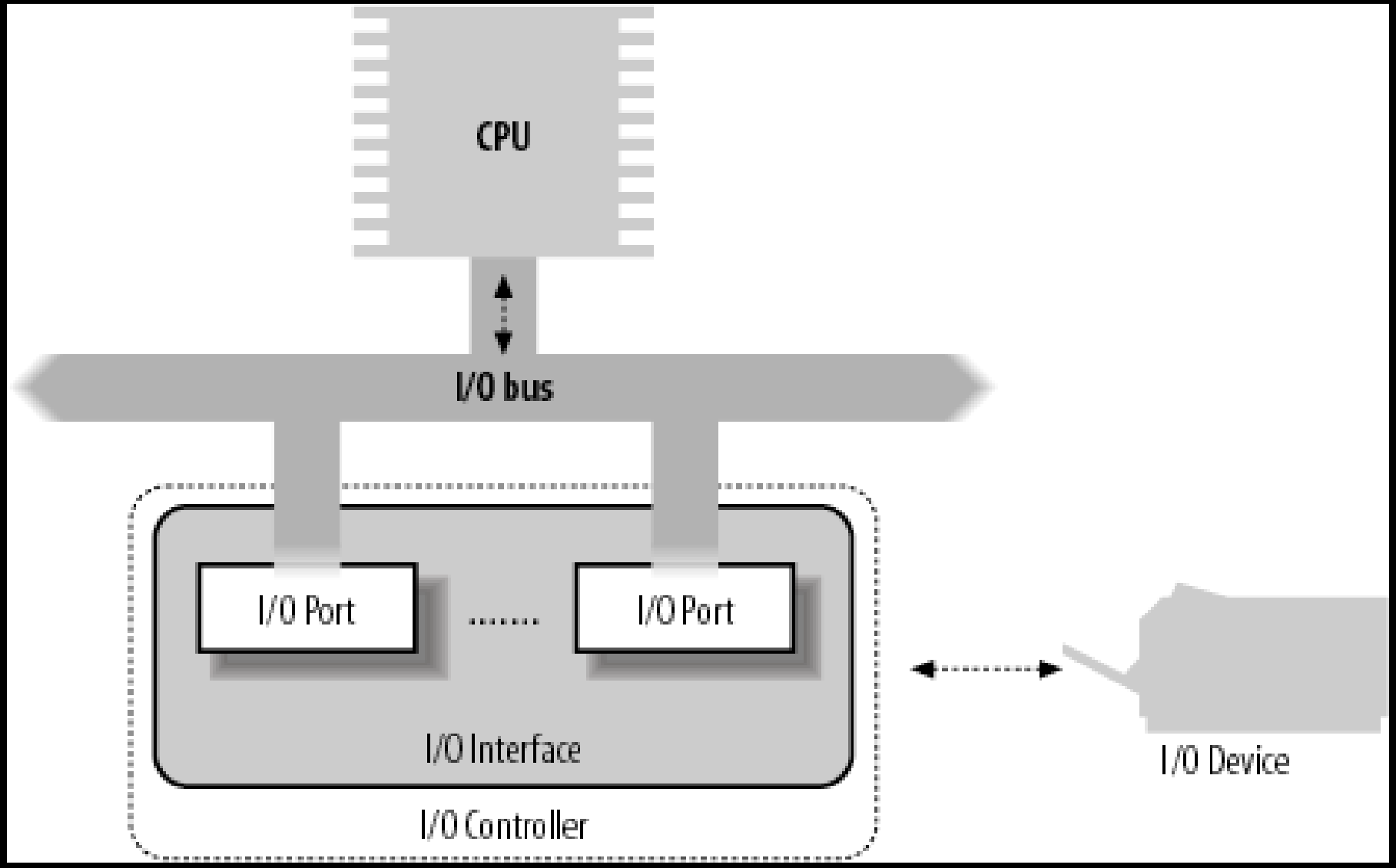
Memory is file as well.

/dev/mem

virtual memory

極爲眞實的 Virtual Memory

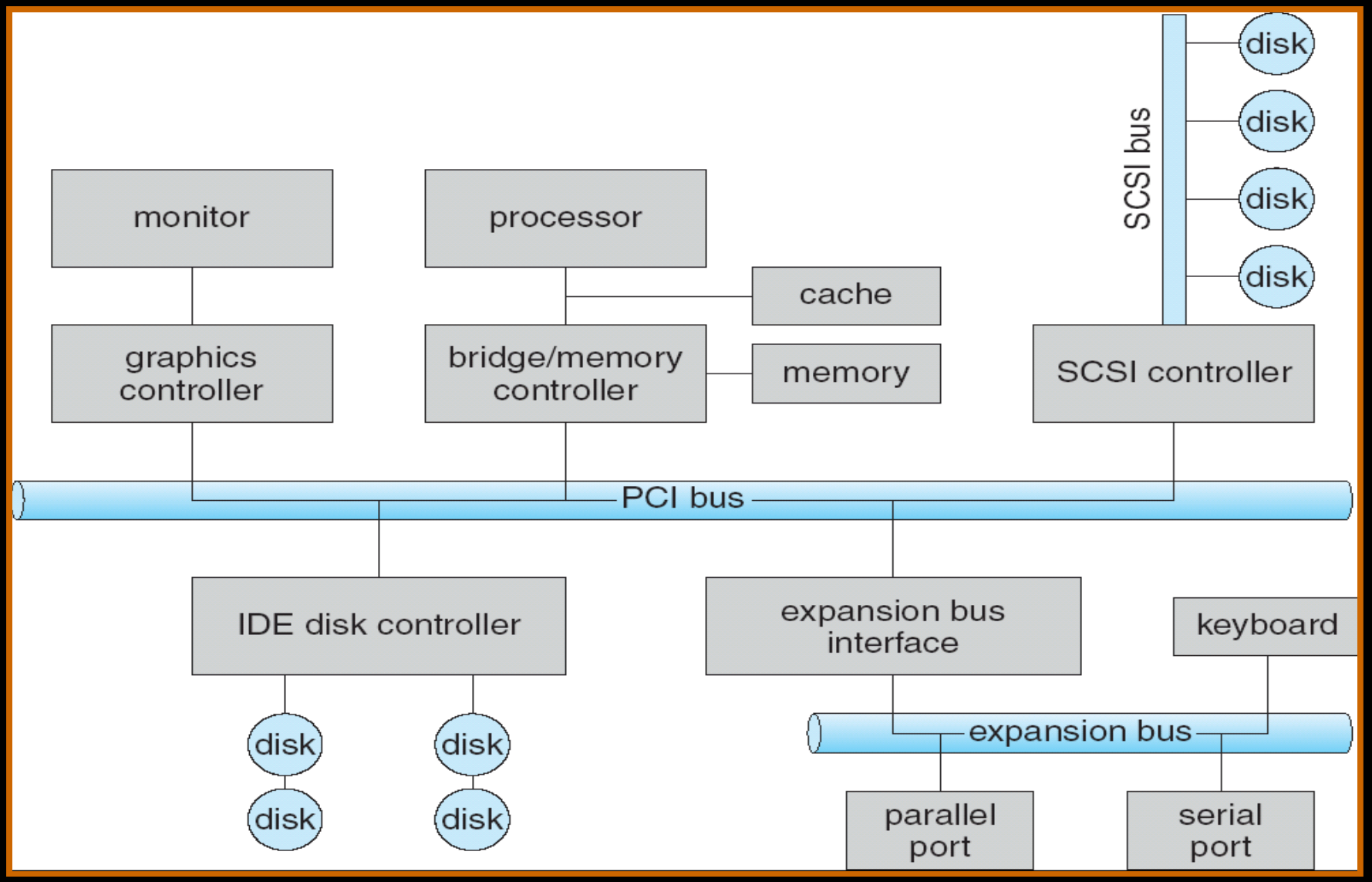
先回顧個人電腦的 I/O 架構



***PMIO** (Port-Mapped I/O) **vs MMIO** (Memory-Mapped I/O)*

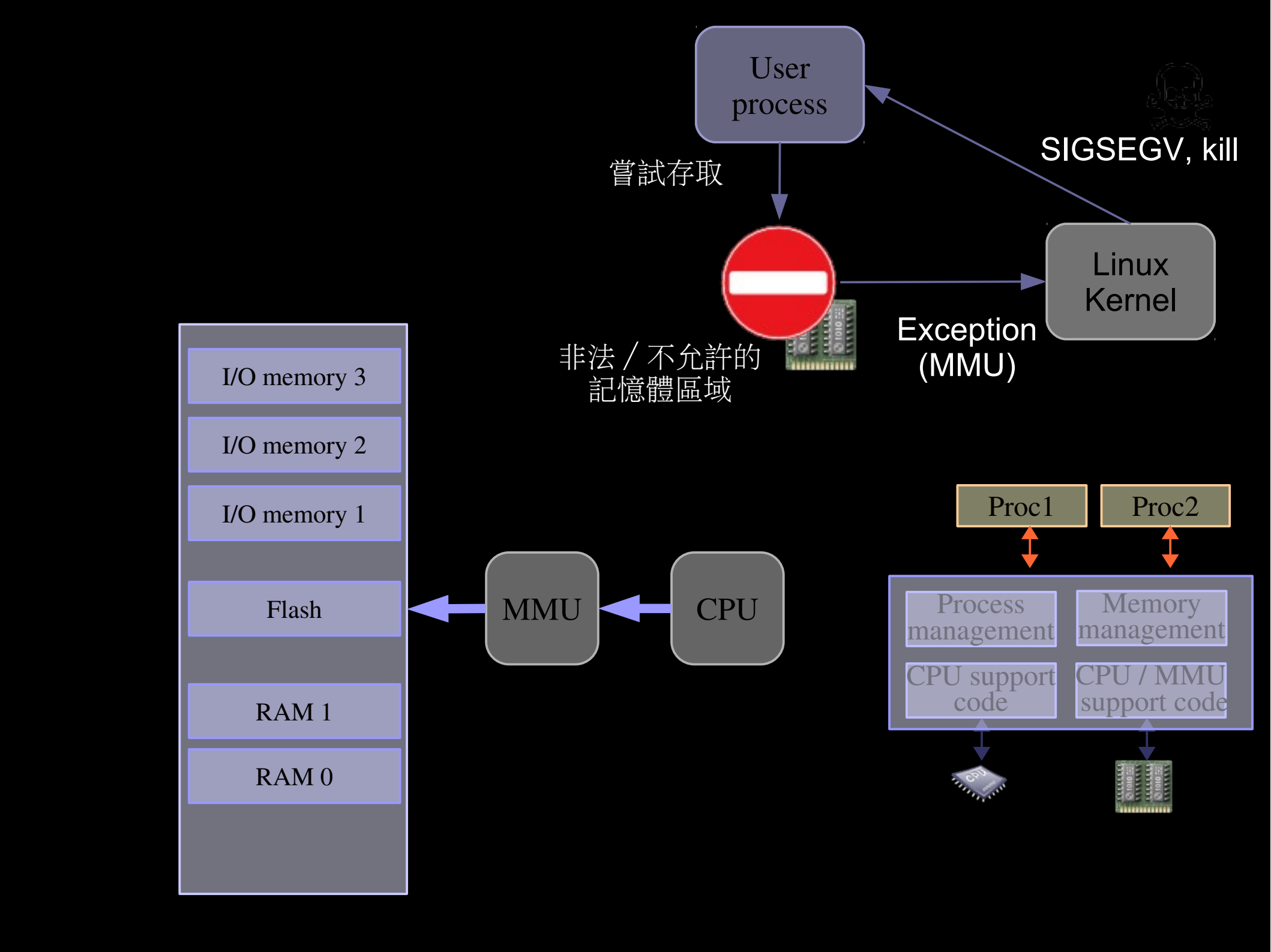
ISA vs. PCI

更全面的觀點

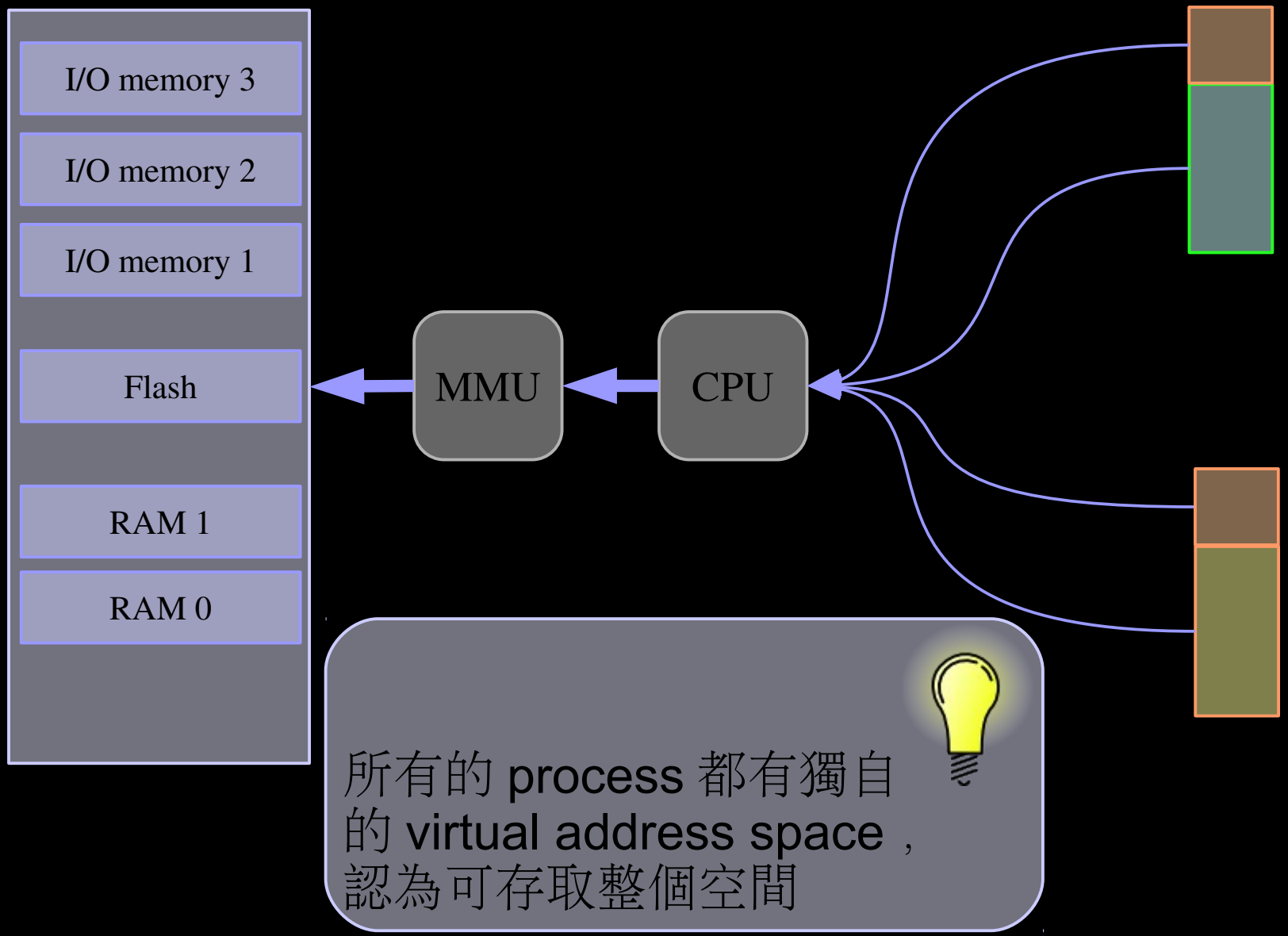

disk
disk
disk
disk
SCSI bus
monitor
processor
cache
graphics controller
bridge/memory controller
memory
SCSI controller
PCI bus
IDE disk controller
expansion bus interface
keyboard
disk
disk
disk
disk
expansion bus
parallel port
serial port

I/O 的型態

- Programmed I/O (PIO)
  - port-mapped I/O
  - memory-mapped I/O
- Interrupt-driven I/O (相對於 polling)
- Direct Memory Access (DMA)
- Channel I/O (I/O processor)


User process
SIGSEGV, kill
嘗試存取
Linux Kernel
Exception (MMU)
非法／不允許的記憶體區域
I/O memory 3
I/O memory 2
I/O memory 1
Flash
RAM 1
RAM 0
MMU
CPU
Proc1
Proc2
Process management
Memory management
CPU support code
CPU / MMU support code

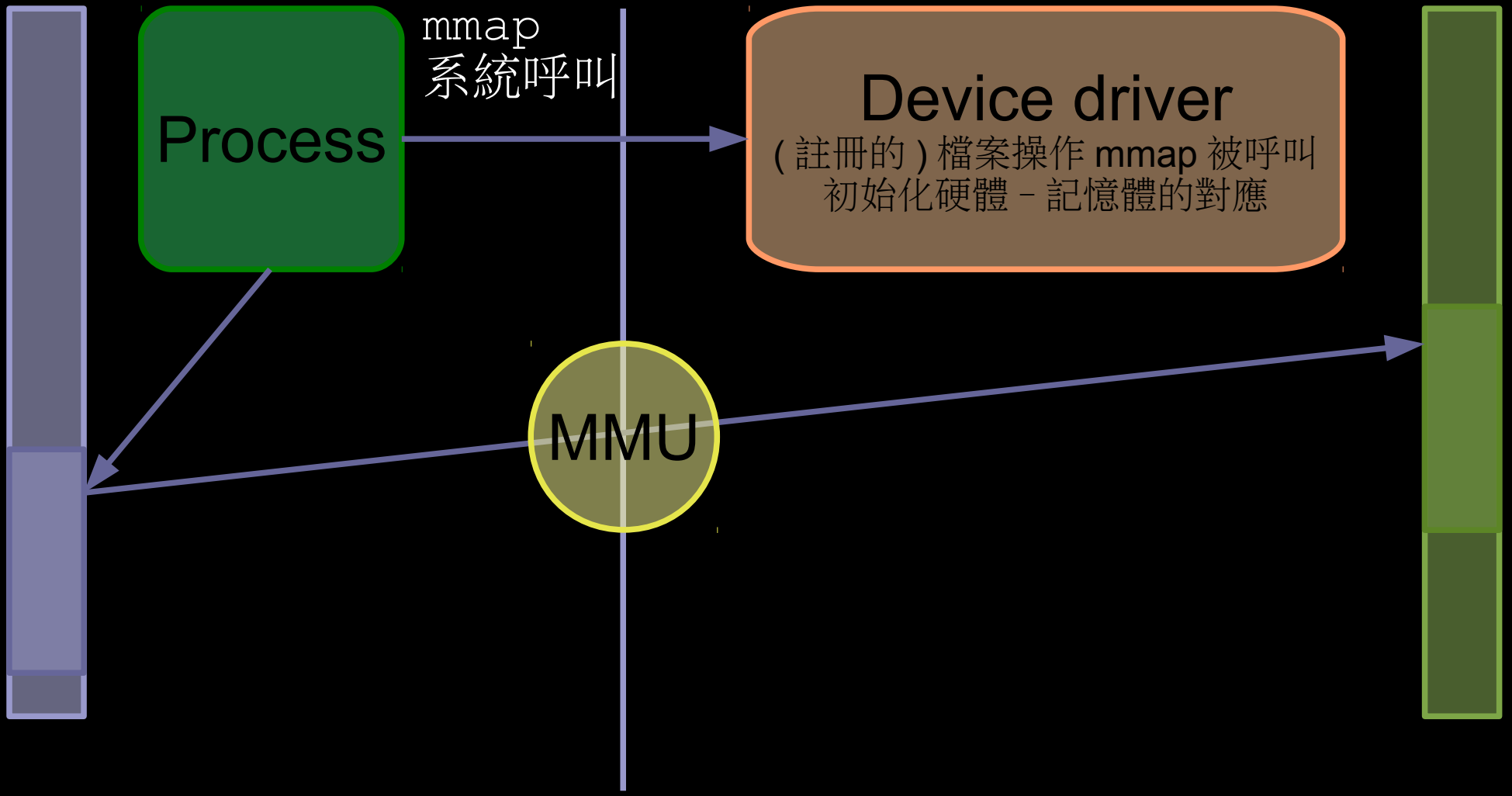
Physical / Virtual memory



Everything is file.

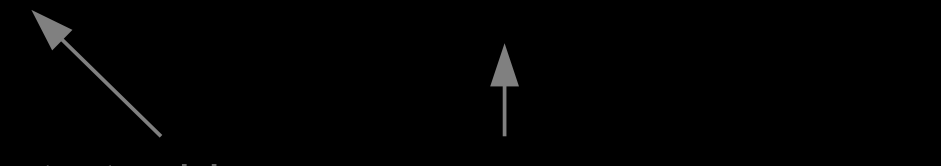
* 需將裝置對應到 Virtual Memory 才能使用

mmap


Process
mmap
系統呼叫
Device driver
(註冊的) 檔案操作 mmap 被呼叫
初始化硬體 - 記憶體的對應
MMU

以 X server 爲例

```
# pmap `pidof X` | grep dev | head
00590000     40K r-x--  /lib/libudev.so.0.6.1
0059a000      4K r----  /lib/libudev.so.0.6.1
0059b000      4K rw---  /lib/libudev.so.0.6.1
00b02000     36K r-x--  /usr/lib/xorg/modules/input/evdev_drv.so
00b0b000      4K r----  /usr/lib/xorg/modules/input/evdev_drv.so
00b0c000      4K rw---  /usr/lib/xorg/modules/input/evdev_drv.so
a75e8000    128K rw-s-  /dev/dri/card0
a7608000    128K rw-s-  /dev/dri/card0
a7628000    128K rw-s-  /dev/dri/card0
a7648000    128K rw-s-  /dev/dri/card0
```



可對照 /proc/`pidof X`/maps 的內容

mmap :: user-space

取得裝置的 fd (file descriptor) 後 ...

- mmap 系統呼叫

```
void * mmap(
   void *start,   /* Often 0, preferred starting address */
   size_t length, /* Length of the mapped area */
   int prot ,     /* Permissions: read, write, execute */
   int flags,     /* Options: shared mapping, private copy */
   int fd,        /* Open file descriptor */
   off_t offset   /* Offset in the file */
);
```

mmap :: kernel-space

```
int (*mmap) (
   struct file *,              /* Open file structure */
   struct vm_area_struct * /* Kernel VMA structure */
);
```

實例：devmem

- 直接 peek (read) 或 poke (write) 某一段已對應的實體位址:　(b: byte, h: half, w: word)

```
devmem 0x000c0004 h (reading)
devmem 0x000c0008 w 0xffffffff (writing)
```

```
if ((fd = open("/dev/mem", O_RDWR | O_SYNC)) == -1)
    FATAL;
```

```
# devmem 0x000c0004 h
Memory mapped at address 0xb7fb5000.
Value at address 0xC0004 (0xb7fb5004): 0xE3A9
```

```
/* Map one page */
map_base = mmap(0, MAP_SIZE, PROT_READ |
PROT_WRITE, MAP_SHARED, fd, target & ~MAP_MASK);
if (map_base == (void *) -1)
    FATAL;
virt_addr = map_base + (target & MAP_MASK);
```

Linux 如何建立軟硬體關聯

/proc/iomem

...是 Rosetta Stone

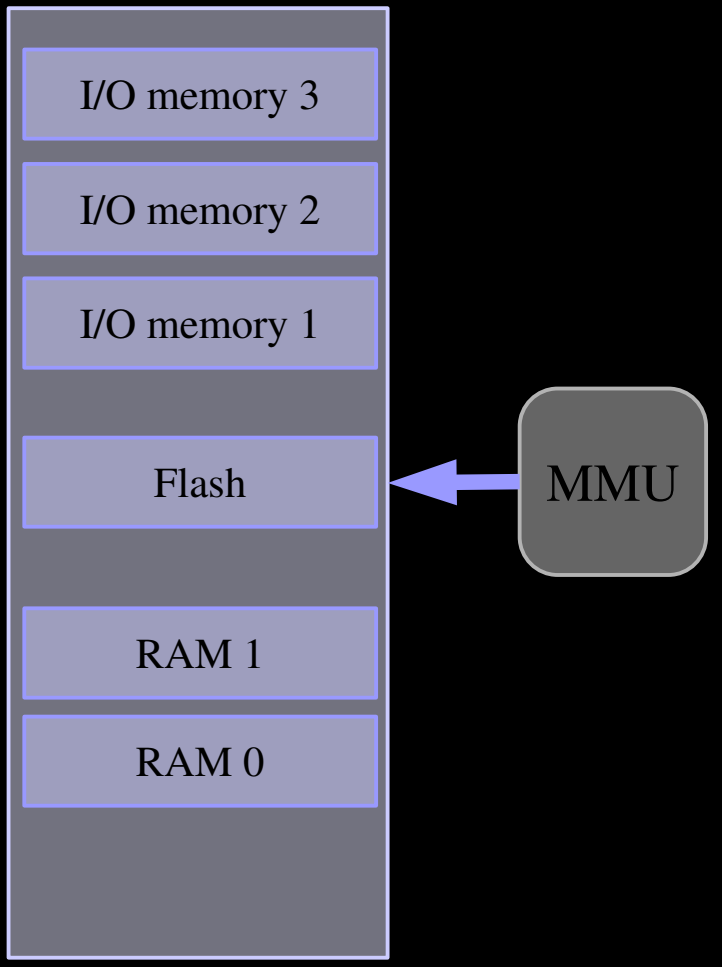
1799 年拿破崙遠征埃及時期，法軍上尉 Pierre-François Xavier Bouchard 在尼羅河口港灣城市羅塞塔 (Rosetta，今日稱為 el-Rashid）發現此碑石，自此揭開古埃及象形文字之謎

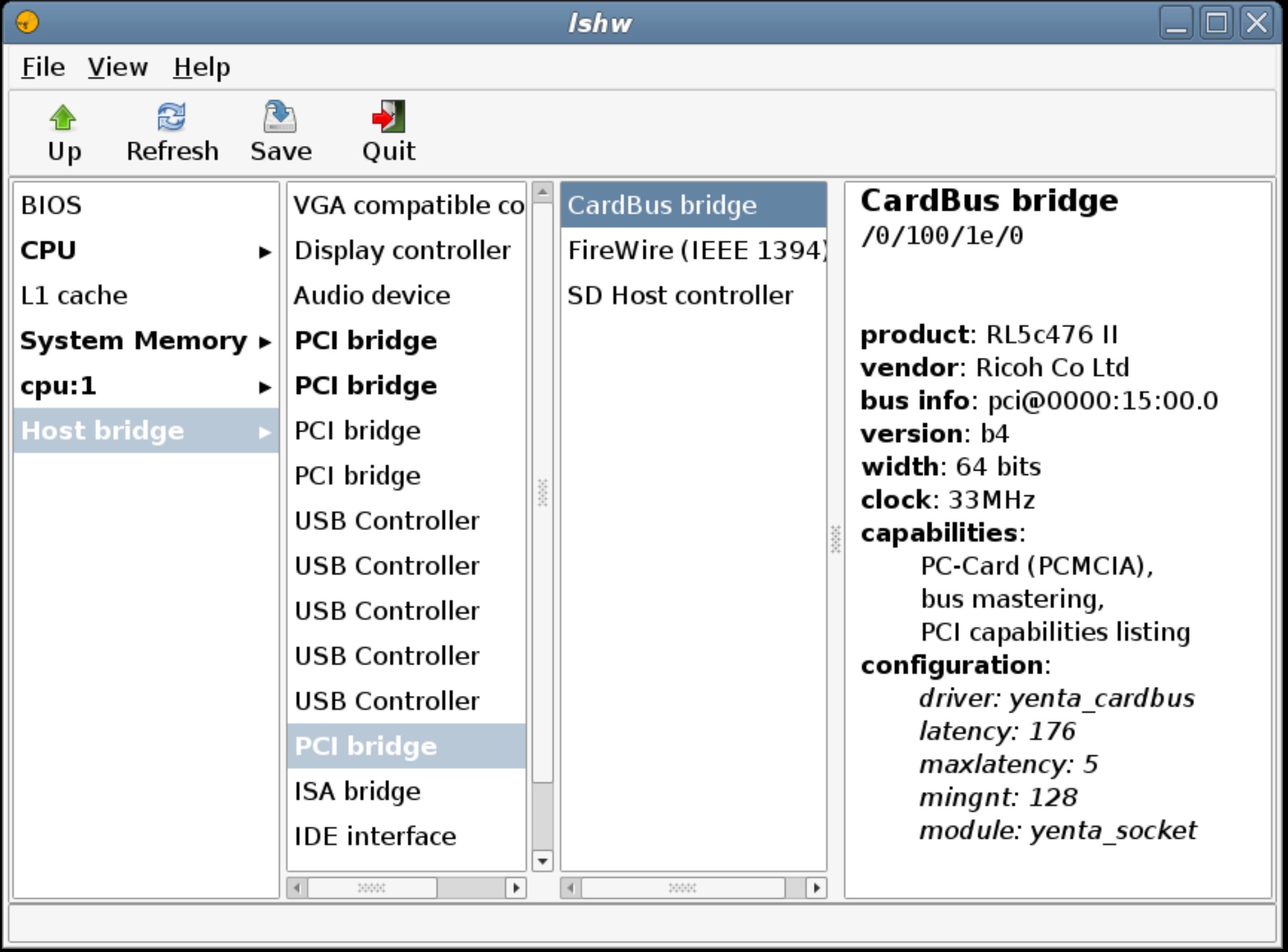
羅塞塔碑石製作於公元前 196 年，原是一塊刻有埃及國王托勒密五世召書的碑石，但由於這塊碑石同時刻有同一段文字的三種不同語言版本，使得近代的考古學家得以在對照各語言版本的內容後，解讀出已經失傳千餘年的埃及象形文之意義與結構，而成爲今日研究古埃及歷史的重要里程碑

由於破解埃及象形文這種如謎題般事物之起點，Rosetta Stone 被比喻爲解決難題或謎題的關鍵線索或工具

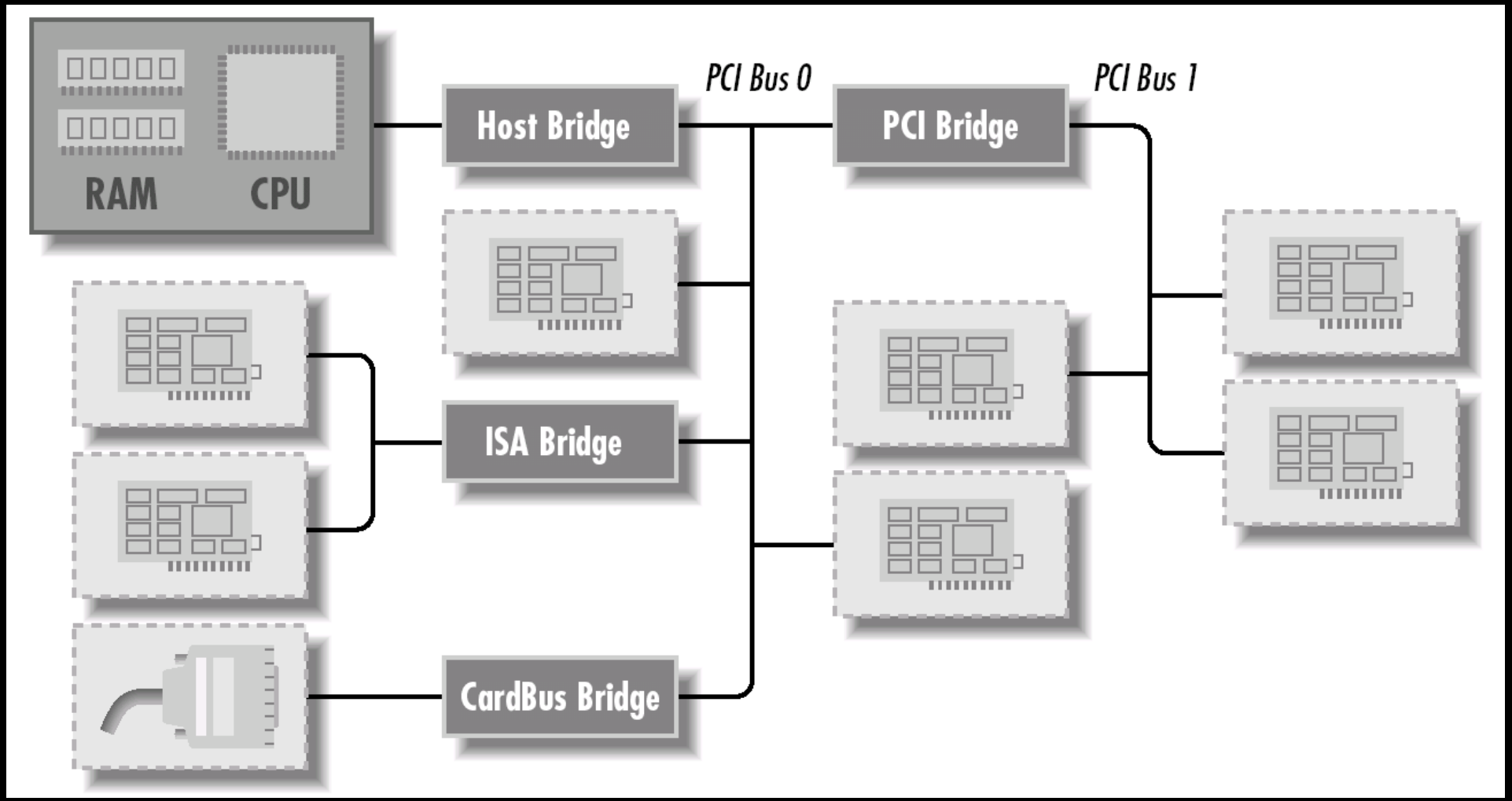
```
# cat /proc/iomem
```

```
# cat /proc/iomem
00000000-0009efff : System RAM
0009f000-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000c8000-000cbfff : pnp 00:00
000cf000-000cffff : Adapter ROM
000d0000-000d0fff : Adapter ROM
000d2000-000d3fff : reserved
000dc000-000dffff : pnp 00:00
000e0000-000e3fff : pnp 00:00
000e4000-000e7fff : pnp 00:00
000e8000-000ebfff : pnp 00:00
000f0000-000fffff : System ROM
00100000-7f6cffff : System RAM
  00100000-0031b5a3 : Kernel code
  0031b5a4-00414dc3 : Kernel data
  00476000-004eba7f : Kernel bss
7f6d0000-7f6defff : ACPI Tables
7f6df000-7f6fffff : ACPI Non-volatile Storage
7f700000-7fffffff : reserved
88000000-8bffffff : PCI CardBus #16
d0000000-dfffffff : 0000:00:02.0
...
e4300000-e7ffffff : PCI Bus #15
  e4300000-e4300fff : 0000:15:00.0
    e4300000-e4300fff : yenta_socket
  e4301000-e43017ff : 0000:15:00.1
    e4301000-e43017ff : ohci1394
  e4301800-e43018ff : 0000:15:00.2
    e4301800-e43018ff : sdhci:slot0
e8000000-e9ffffff : PCI Bus #04
ea000000-ebffffff : PCI Bus #0c
ec000000-edffffff : PCI Bus #03
  edf00000-edf00fff : 0000:03:00.0
    edf00000-edf00fff : iwl3945
...
```

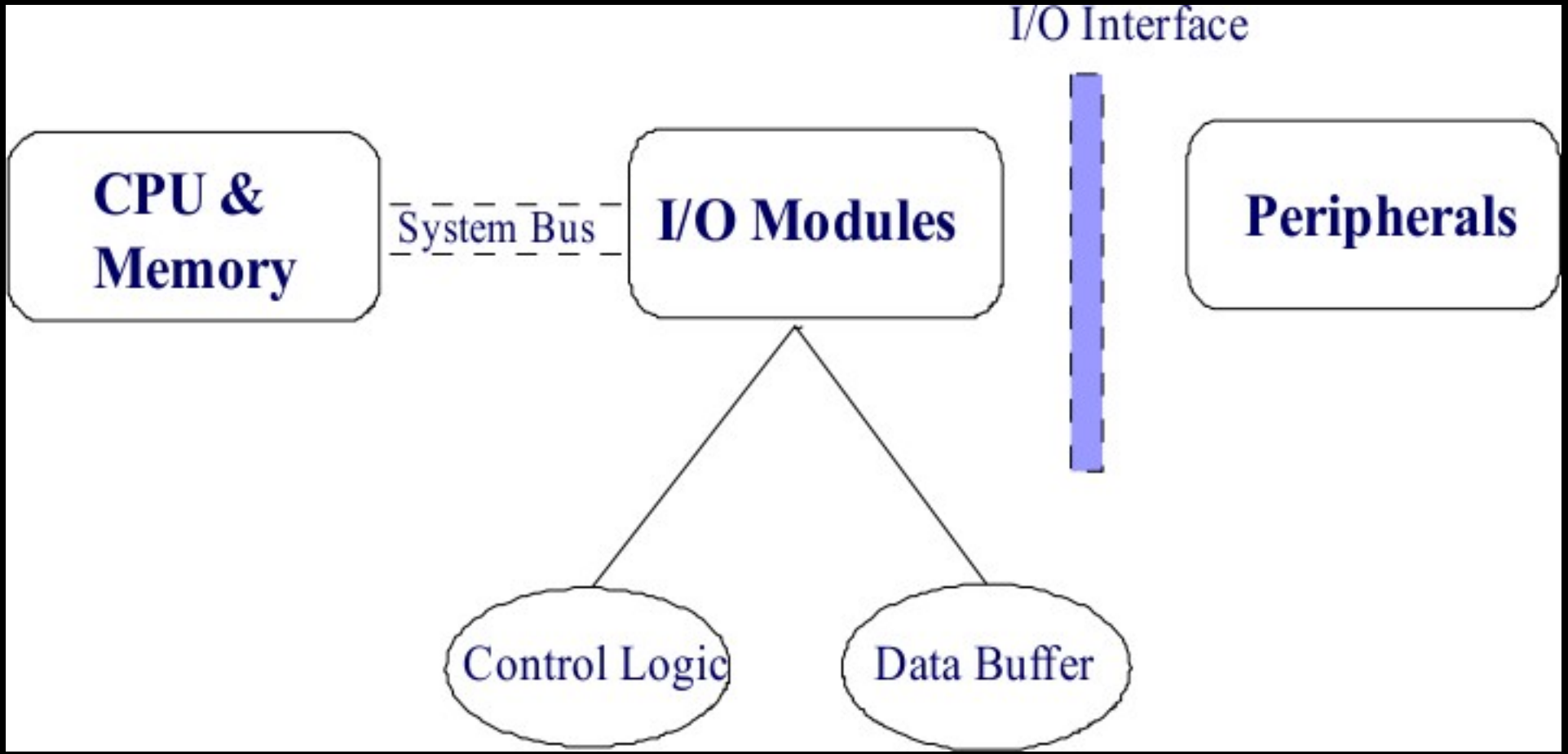



lshw
File View Help
Up
Refresh
Save
Quit
BIOS
CPU
L1 cache
System Memory
cpu:1
Host bridge
VGA compatible co
Display controller
Audio device
PCI bridge
PCI bridge
PCI bridge
PCI bridge
USB Controller
USB Controller
USB Controller
USB Controller
USB Controller
PCI bridge
ISA bridge
IDE interface
CardBus bridge
FireWire (IEEE 1394)
SD Host controller
CardBus bridge
/0/100/1e/0
product: RL5c476 II
vendor: Ricoh Co Ltd
bus info: pci@0000:15:00.0
version: b4
width: 64 bits
clock: 33MHz
capabilities:
PC-Card (PCMCIA),
bus mastering,
PCI capabilities listing
configuration:
driver: yenta_cardbus
latency: 176
maxlatency: 5
mingnt: 128
module: yenta_socket

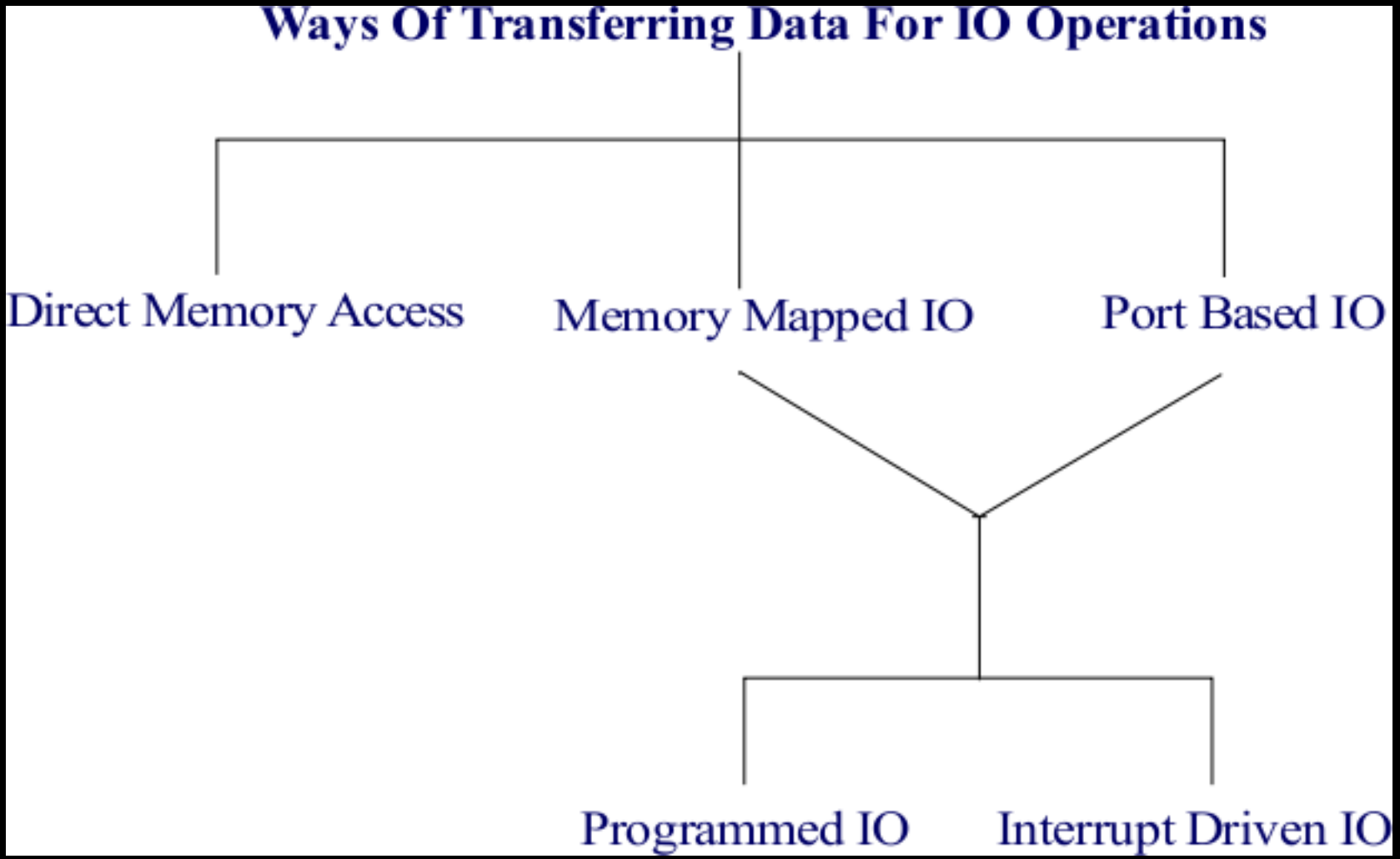
典型 PCI


RAM
CPU
Host Bridge
PCI Bus 0
PCI Bridge
PCI Bus 1
ISA Bridge
CardBus Bridge

低階「通訊」概況


I/O Interface
CPU &
Memory
System Bus
I/O Modules
Peripherals
Control Logic
Data Buffer

實際的 I/O 操作途徑


Ways Of Transferring Data For IO Operations
Direct Memory Access
Memory Mapped IO
Port Based IO
Programmed IO
Interrupt Driven IO

PMIO (Port-Mapped I/O) vs MMIO (Memory-Mapped I/O)

$ cat /proc/ioports

```
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-006f : keyboard
0070-0077 : rtc
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
02f8-02ff : serial
0376-0376 : ide1
[...]
```

$ cat /proc/iomem

```
00000000-0009fbff : System RAM
  00000000-00000000 : Crash kernel
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000cc000-000d97ff : Adapter ROM
000f0000-000fffff : System ROM
00100000-0ffeffff : System RAM
  00100000-00281514 : Kernel code
  00281515-003117b3 : Kernel data
20000000-200fffff : PCI Bus #01
[...]
```

Port-Mapped I/O 分佈

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Requesting I/O ports

/proc/ioports

```
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-006f : keyboard
0070-0077 : rtc
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
0100-013f : pcmcia_socket0
0170-0177 : ide1
01f0-01f7 : ide0
0376-0376 : ide1
0378-037a : parport0
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial
0800-087f : 0000:00:1f.0
0800-0803 : PM1a_EVT_BLK
0804-0805 : PM1a_CNT_BLK
0808-080b : PM_TMR
0820-0820 : PM2_CNT_BLK
0828-082f : GPE0_BLK
...
```

- `struct resource *request_region(`
  `unsigned long start,`
  `unsigned long len,`
  `char *name);`

嘗試保留給定區域

`request_region(0x0170, 8, "ide1");`

- `void release_region(`
  `unsigned long start,`
  `unsigned long len);`
- 參考 `include/linux/ioport.h` 與 `kernel/resource.c`

Requesting I/O memory

/proc/iomem

```
00000000-0009efff : System RAM
0009f000-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000cffff : Video ROM
000f0000-000fffff : System ROM
00100000-3ffadfff : System RAM
  00100000-0030afff : Kernel code
  0030b000-003b4bff : Kernel data
3ffae000-3fffffff : reserved
40000000-400003ff : 0000:00:1f.1
40001000-40001fff : 0000:02:01.0
  40001000-40001fff : yenta_socket
40002000-40002fff : 0000:02:01.1
  40002000-40002fff : yenta_socket
40400000-407fffff : PCI CardBus #03
40800000-40bfffff : PCI CardBus #03
40c00000-40ffffff : PCI CardBus #07
41000000-413fffff : PCI CardBus #07
a0000000-a0000fff : pcmcia_socket0
a0001000-a0001fff : pcmcia_socket1
e0000000-e7ffffff : 0000:00:00.0
e8000000-efffffff : PCI Bus #01
  e8000000-efffffff : 0000:01:00.0
...
```

- 介面一致
- ```
  struct resource * request_mem_region(
     unsigned long start,
     unsigned long len,
     char *name);
  ```
- ```
  void release_mem_region(
     unsigned long start,
     unsigned long len);
  ```

大部分驅動程式的工作大概就在讀寫
I/O port 或 I/O memory(統稱 I/O region)

io_mem.c(1)

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/ioport.h>

static int Major, result;
static struct file_operations fops;
MODULE_LICENSE("GPL");
unsigned long start = 1, length = 1;
module_param(start, long, 1);
module_param(length, long, 1);
...
module_init (memIO_init);
module_exit (memIO_cleanup);
```

io_mem.c(2)

```
int memIO_init (void)
{
    Major = register_chrdev (0, "memIO_device", &fops);
    if (Major < 0) {
        printk (" Major number allocation is failed \n");
        return (Major);
    }
    printk (" The Major number of the device is %d \n", Major);
    result = check_mem_region (start, length);
    if (result < 0) {
        printk ("Allocation for I/O memory range is failed: “
                ”Try other range\n");
        return (result);
    }
    request_mem_region (start, length, "memIO_device");
   return 0;
}
```

io_mem.c(3)

```
void memIO_cleanup (void)
{
    release_mem_region (start, length);
    printk (" The I/O memory region is released “
            ”successfully \n");
    unregister_chrdev (Major, "memIO_device");
    printk (" The Major number is released successfully \n");
}
```

在發送 Request 之前

```
# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 14 sound
 21 sg
 29 fb
108 ppp
116 alsa
128 ptm
136 pts
171 ieee1394
180 usb
189 usb_device
226 drm
...
```

```
# cat /proc/iomem
00000000-0009efff : System RAM
0009f000-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000c8000-000cbfff : pnp 00:00
000cf000-000cffff : Adapter ROM
000d0000-000d0fff : Adapter ROM
000d2000-000d3fff : reserved
...
ee444000-ee4443ff : 0000:00:1d.7
  ee444000-ee4443ff : ehci_hcd
ee444400-ee4447ff : 0000:00:1f.2
  ee444400-ee4447ff : ahci
...
```

```
# insmod io_mem.ko start=0xeeee0000
                     length=1
```

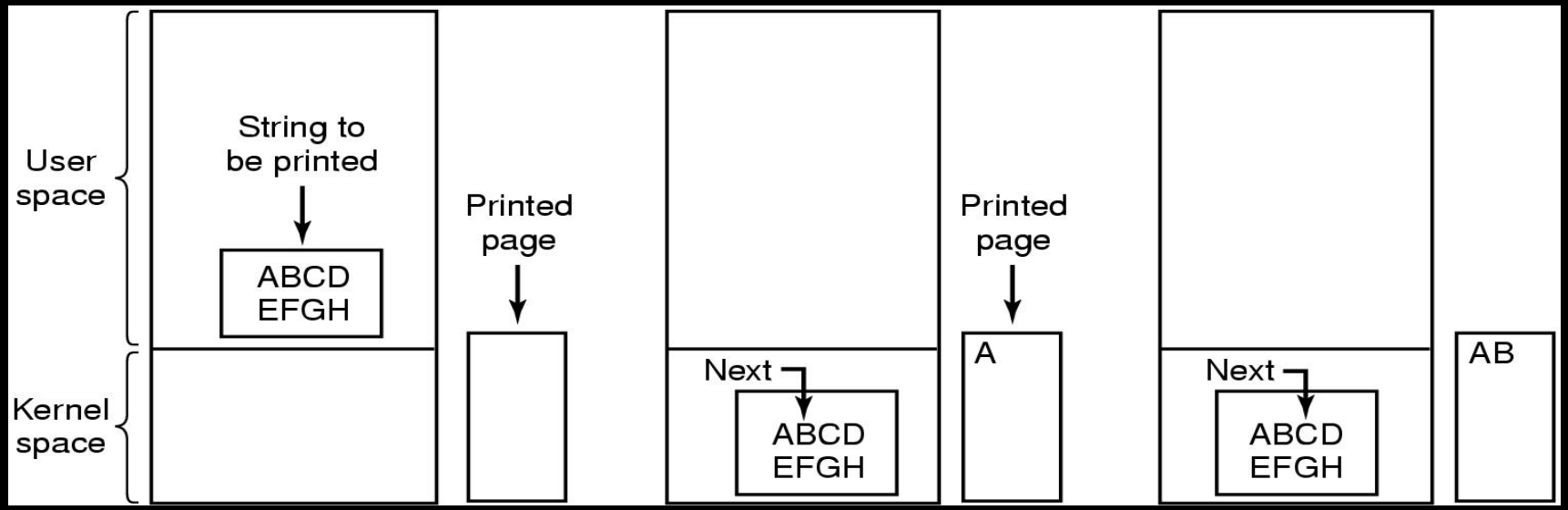
發送 Request 之後

```
# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 14 sound
 21 sg
 29 fb
108 ppp
116 alsa
128 ptm
136 pts
171 ieee1394
180 usb
189 usb_device
226 drm
252 memIO_device
...
```

```
# cat /proc/iomem
00000000-0009efff : System RAM
0009f000-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000c8000-000cbfff : pnp 00:00
000cf000-000cffff : Adapter ROM
000d0000-000d0fff : Adapter ROM
000d2000-000d3fff : reserved
...
ee444000-ee4443ff : 0000:00:1d.7
  ee444000-ee4443ff : ehci_hcd
ee444400-ee4447ff : 0000:00:1f.2
  ee444400-ee4447ff : ahci
eeee0000-eeee0000 : memIO_device
...
```

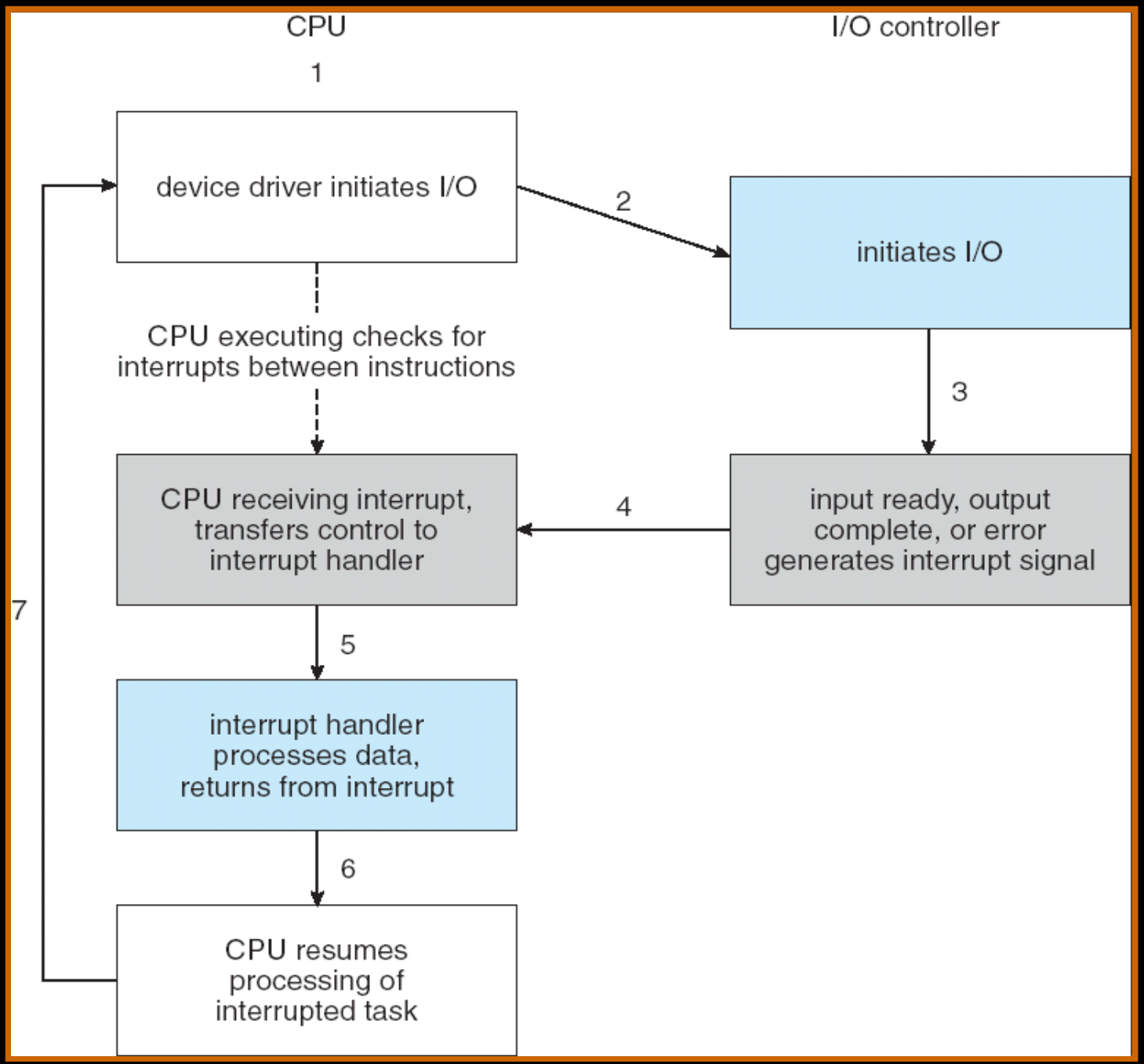
```
# dmesg
...
[11682.040057]  The Major number of the device is 251
```

Polling




```
copy_from_user(buffer, p, count);          /* p is the kernel buffer */
for (i = 0; i < count; i++) {              /* loop on every character */
    while (*printer_status_reg != READY)   /* loop until ready */
        ;
    printer_data_register = p[i];          /* output one character */
}
return_to_user();
```

Interrupt

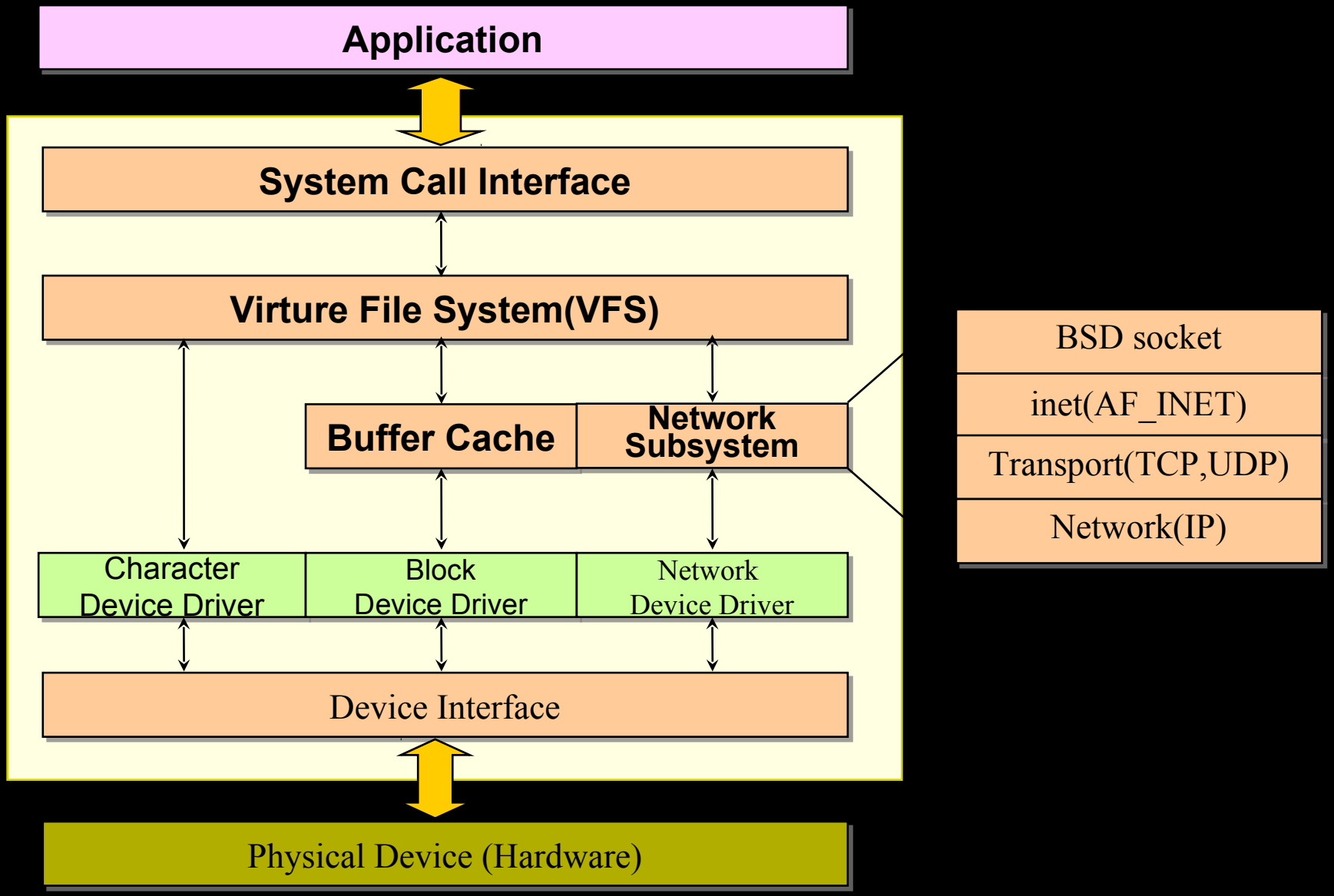

CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

IRQ lines

```
$ cat /proc/interrupts
           CPU0
  0:  111151372    IO-APIC-edge  timer
  1:          8    IO-APIC-edge  i8042
  6:          2    IO-APIC-edge  floppy
  7:          2    IO-APIC-edge  parport0
  8:          4    IO-APIC-edge  rtc
  9:          1   IO-APIC-level  acpi
 12:        114    IO-APIC-edge  i8042
 14:    3277682    IO-APIC-edge  ide0
 15:         65    IO-APIC-edge  ide1
169:   14445090   IO-APIC-level  eth0
NMI:          0
LOC:  111150116
ERR:          0
MIS:          0
```

Linux 驅動程式架構與發展

Linux 驅動程式架構



Everything is file.

::Device File::

Device File 類型


```
crw--w--w-  0 root   root      5,   1  Oct  1  1998 console
crw-rw-rw-  1 root   root      1,   3  May  6  1998 null
crw-------  1 root   root      4,   0  May  6  1998 tty
crw-rw----  1 root   disk     96,   0  Dec 10  1998 pt0
crw-------  1 root   root      5,  64  May  6  1998 cua0
```

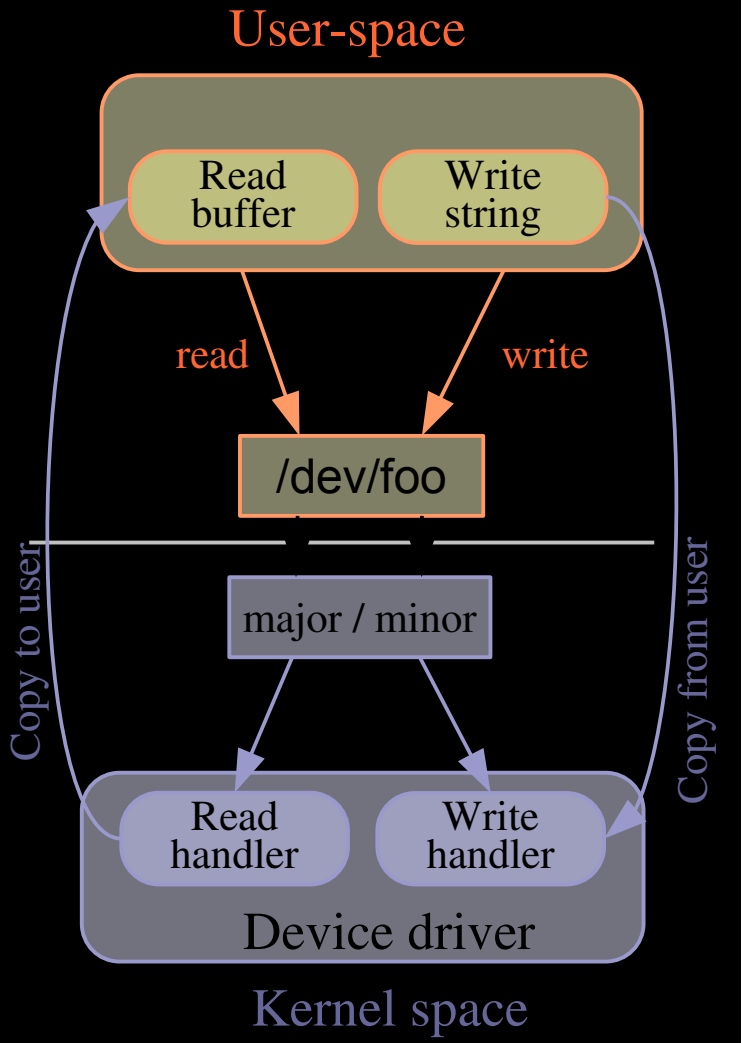
```
brw-------   1 root     floppy     2,   0 May  6  1998 fd0
brw-rw----   1 root     disk       3,   0 May  6  1998 hda
brw-rw----   1 root     disk       3,   1 May  6  1998 hda1
brw-rw----   1 root     disk       8,   0 May  6  1998 sda
brw-rw----   1 root     disk       8,   1 May  6  1998 sda1
```

Major Number 與 Minor Number

/dev 目錄下的檔案稱為 device files，用以辨識 / **對應於**裝置

Kernel 透過 major number 來指定正確的驅動程式給 device files

Minor number 只有驅動程式本身會使用到



註冊的 device driver

顯示於 /proc/devices:

```
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  6 lp
 10 misc
 13 input
 14 sound
...

Block devices:
  1 ramdisk
  3 ide0
  8 sd
  9 md
 22 ide1
 65 sd
 66 sd
 67 sd
 68 sd
```

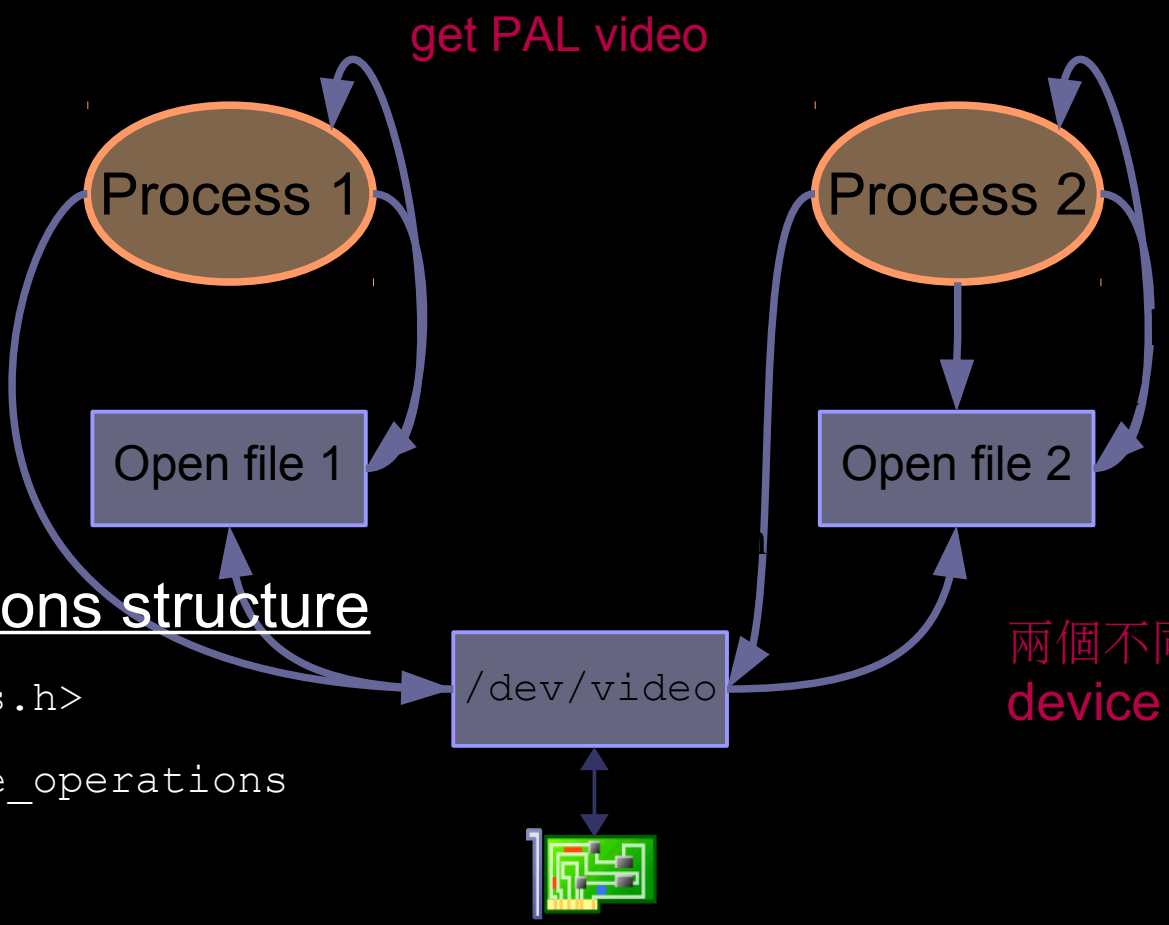
Major number

Registered name

Linux Device Driver 資料結構

```
struct   file_operations   {       /* character device drivers */
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long,
                        unsigned long, unsigned long);
};
```

檔案操作的內部



兩個不同 process 可自同個 device file 讀取不同的資料

實做 file_operations structure

```
#include <linux/fs.h>

static struct file_operations

my_fops =
{
    .owner = THIS_MODULE,
    .read = moko_read,
    .write = moko_write,
};
```

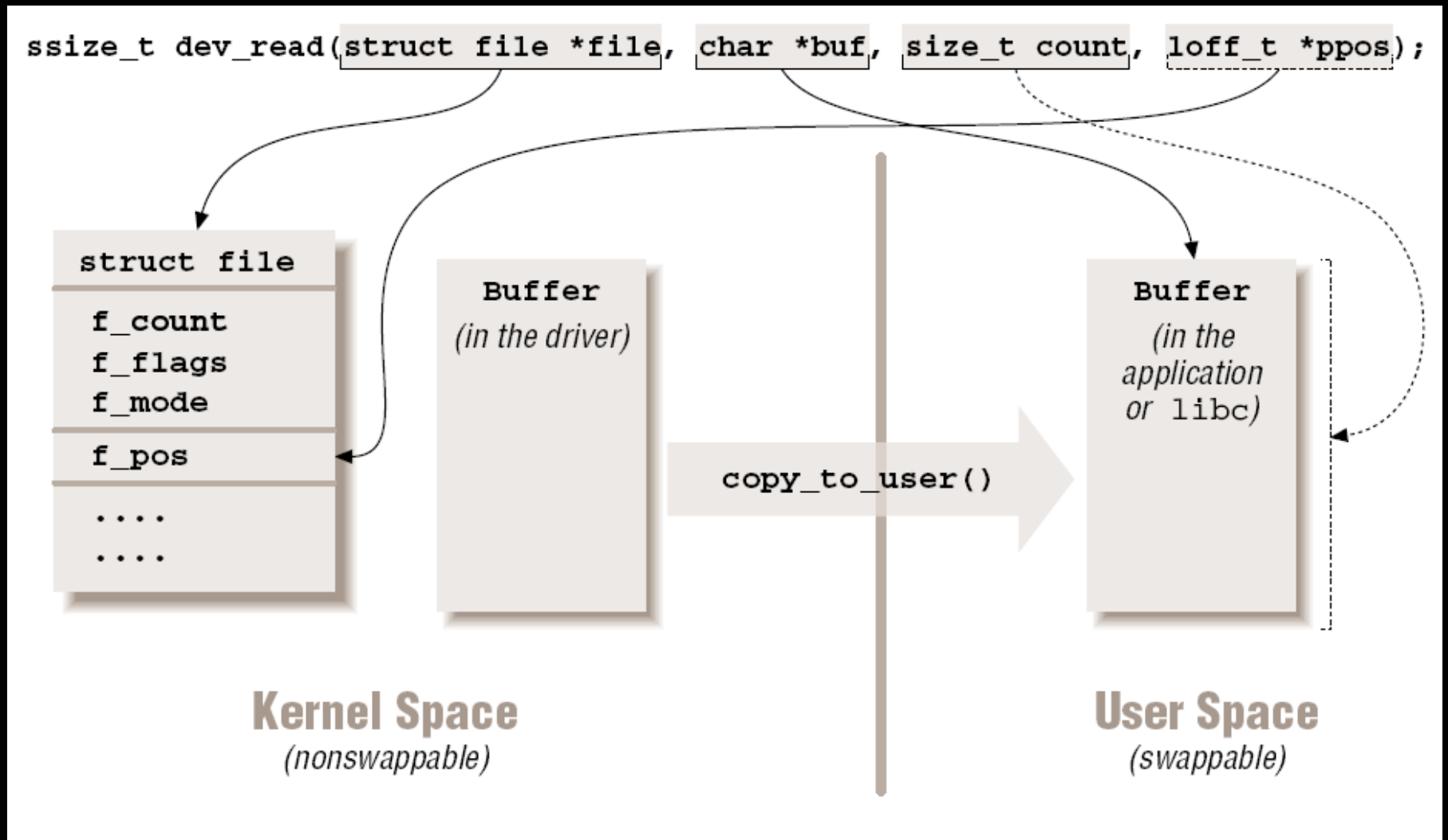
將上述 my_read/my_write 指向為具體實做

否則採用預設 function(如 open, release...)

read() 操作內部

- The *read* methods copies data to application code.

```
ssize_t read(struct file *filp, char *buff, size_t count, loff_t *offp);
```



Device number 配置方式

```
#include <linux/fs.h>
int register_chrdev_region(
  dev_t from,   /* Starting device number */
  unsigned count,/* Number of device numbers */
  const char *name);/* Registered name */
```

Returns 0 if the allocation was successful.

Example

```
if (register_chrdev_region(
       MKDEV(202, 128),
       moko_count, "moko")) {
   printk(KERN_ERR "Failed to allocate "
                  "device number\n");
   ...
```

Allocating fixed device numbers

```
#include <linux/fs.h>
int alloc_chrdev_region(
  dev_t *dev, /* Output: device number */
  unsigned baseminor,
    /* Starting minor number, usually 0 */
  unsigned count,
    /* Number of device numbers */
  const char *name /* Registered name */
);
```

Returns 0 if the allocation was successful.

Example

```
if (alloc_chrdev_region(&moko_dev, 0,
   moko_count, "moko")) {
   printk(KERN_ERR "Failed to allocate
   device number\n");
   ...
```

Dynamic allocation of device numbers

Safer: have the kernel allocate free numbers for you!

要點

當 LKM 即將從系統釋放， major number 必須交還系統

檔案操作與結構

- Device ID By file structure
- 核心使用 file ops 以存取 driver 提供的函式

- open / close
  - 初始化裝置並調整 usage count
- Memory
  - Device-memory Link List
- Read / Write
  - Transfer data from Kernel to User

進階 I/O 控制

- Blocking / Non-blocking I/O
  Polling: poll and select function
  Asynchronous notification
  Access control of device file

考量點

Synchronization / Avoid race condition
- semaphore / mutex / kernel lock

Reentrance

Preemption in kernel code

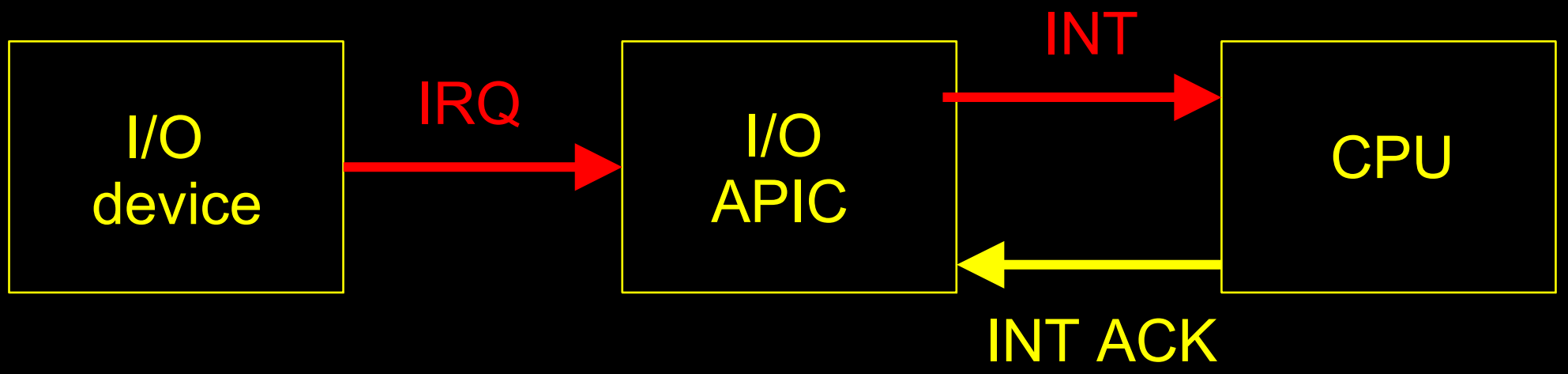
Interrupt timer / peripheral

Take care of bandwidth issues

DMA vs. PIO (Programming I/O) model

SMP

HW Concurrency (1)



- I/O APIC 對裝置不斷作 poll 並發出 interrupt
- 在 CPU 回應 (ACK/acknowledge) 前，沒有新的 interrupt 可被發出
- 通常核心得在多個 interrupt 產生的情況下執行

HW Concurrency (2)

- Symmetrical MultiProcessor (SMP) 顧名思義，擁有兩個以上的 CPU
- SMP kernel 必須在有效的 CPU 上同步執行
- 情境：在其中一顆 CPU 上執行了網路相關的 service routine，而另一個 CPU 則執行檔案系統相關的動作

```
# cat /proc/interrupts
           CPU0       CPU1
  0:    4988074    5094948   IO-APIC-edge      timer
...
 29:          7      11515   PCI-MSI-edge      eth0
 30:     290411     379267   PCI-MSI-edge      i915
 31:     651550     496152   PCI-MSI-edge      iwlagn
NMI:          0          0   Non-maskable interrupts
...
TRM:          0          2   Thermal event interrupts
THR:          0          0   Threshold APIC interrupts
MCE:          0          0   Machine check exceptions
MCP:         77         77   Machine check polls
ERR:          1
MIS:          0
```

interrupt 總量

```
# cat /proc/stat | grep intr
intr 8190767 6092967 10377 0 1102775 5 2 0 196 ...
```

Total number of interrupts

IRQ1 total

IRQ2 total

IRQ3 ...

Linux 2.6+ 新的 Driver Model

Device Model 特徵 (1)

- 從簡化能源管理處理出發，現在已大幅改進
- 充分表現硬體系統架構
- 提供 user-space 層級的表示途徑： sysfs 比 /proc 更全面
- 更一致、容易維護的 device interface: `include/linux/device.h`

Device model 特徵 (2)

允許從多角度檢視系統

- 從系統上已存在的裝置來看： power state 、連結上的 bus ， 與對應的 driver
- 從 system bus 結構來看： bus 與 bus 之間的連結方式（如 PCI bus 上的 USB bus controller)、對應的 driver
- 從裝置本身來看： `input`, `net`, `sound`...

得以簡便地找到裝置，而不需顧慮實體連結方式

“From a technical standpoint, I believe **the kernel will be "more of the same"** and that all the **_really_** interesting staff will be going out in **user space.**”

-- Linux Torvalds

October 2001

sysfs

- Device Model 的 user-space 表現方式
- Kernel 配置編譯選項
  `CONFIG_SYSFS=y` (Filesystems -> Pseudo filesystems)
- 掛載方式:
  `sudo mount -t sysfs none /sys`
- 探索 `/sys` 是理解硬體設計便利的途徑

sysfs tools

http://linux-diag.sourceforge.net/Sysfsutils.html

- `libsysfs` - 提供一致且穩定的介面，得以查閱由 sysfs 公開的系統硬體資訊
- `systool` – 使用 `libsysfs` 的應用程式，可列印裝置的 `bus, class, topology`

Device Model 參考資訊

- 最方便且清晰的文件就內建於核心程式碼中
- Documentation/driver-model/
- Documentation/filesystems/sysfs.txt

udev / hotplug

原本 /dev 暴露的問題

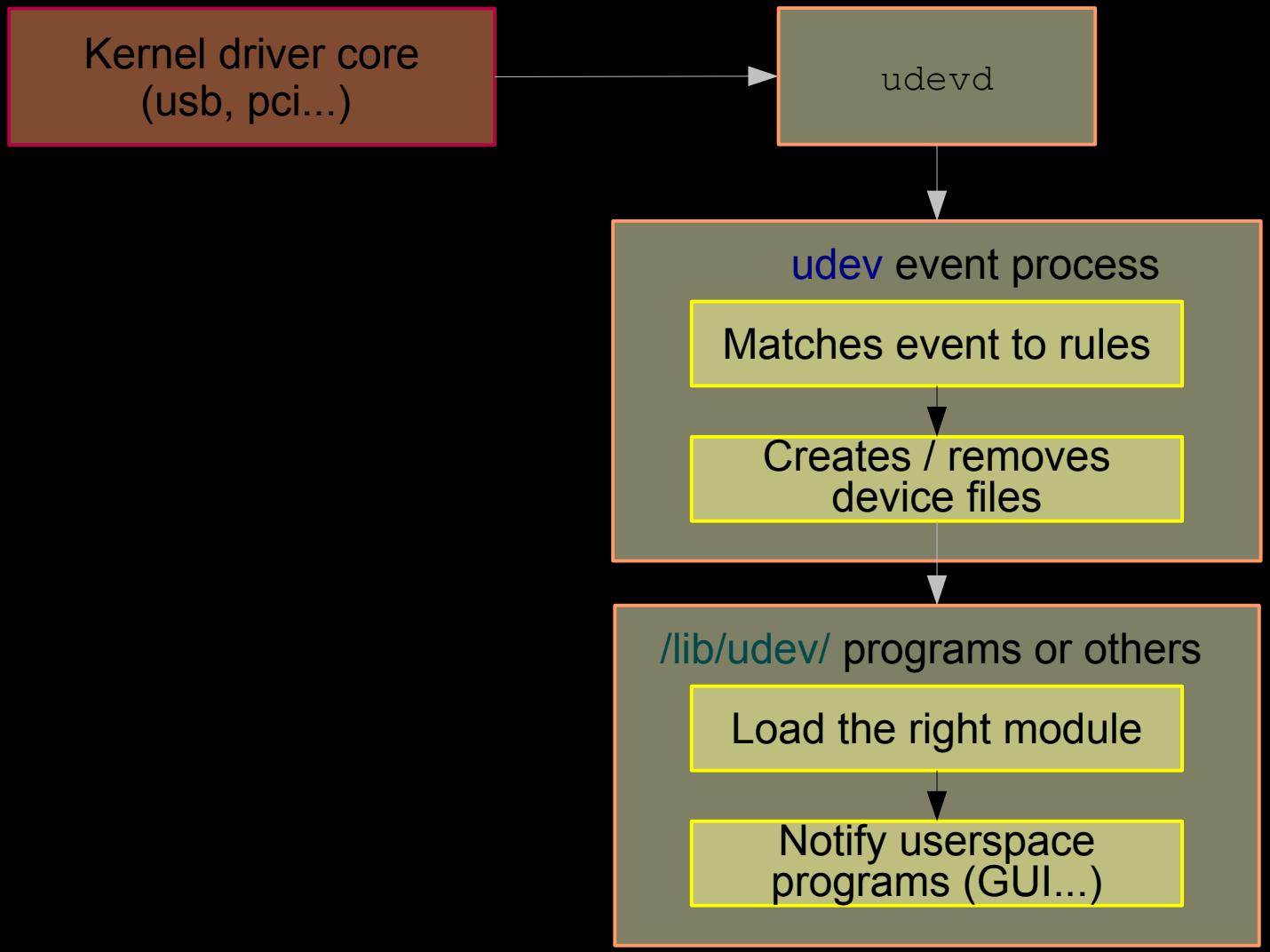
- Red Hat 9 Linux 上，為了支援可能的應用，必須在 /dev 擺放 18000 個 device files
- 需要對 major number 作授權與協調分配 http://lanana.org/: Linux Assigned Names and Numbers Authority
- 對於 user-space 應用程式來說，既不知曉在系統上的裝置，也不知曉對應 /dev 上的 device files

udev 解決方案

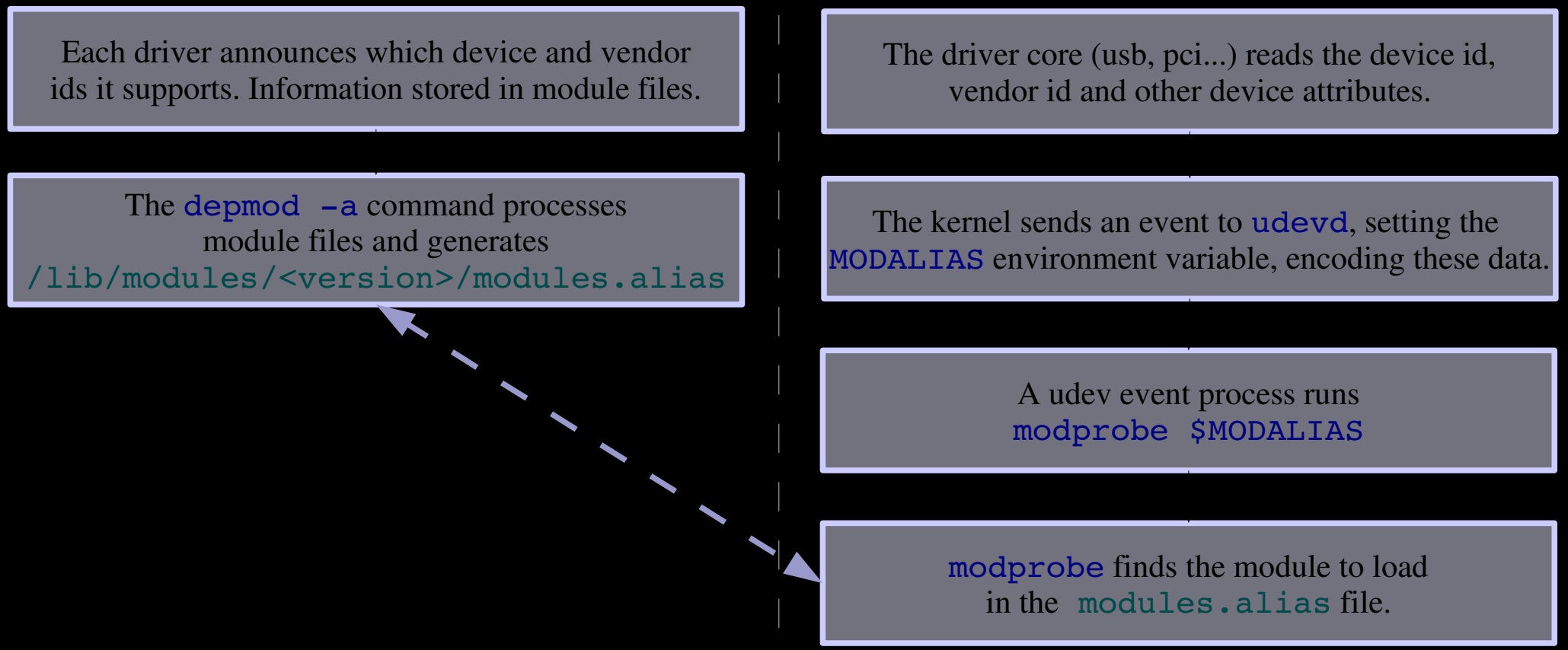
善用 sysfs 機制

- 全部在 user-space
- 依據實際上硬體新增與移除的裝況，自動建立與刪除 `/dev/` 底下的 `device file`
- Major and minor device transmitted by the kernel.
- Requires no change to driver code.
- Fast: written in C
  Small size: `udevd` version 117: 67 KB in Ubuntu 8.04

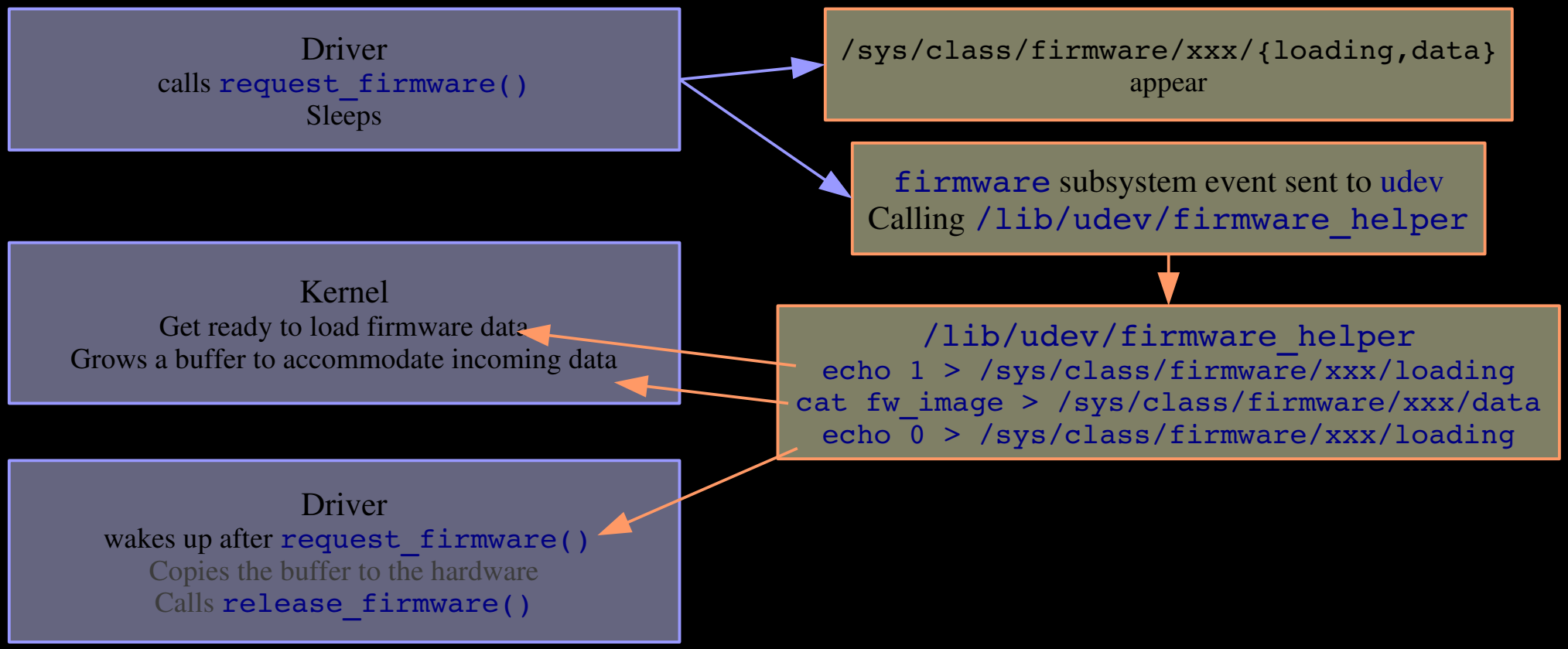
udev 典型操作


Kernel driver core
(usb, pci...)
udevd
udev event process
Matches event to rules
Creates / removes
device files
/lib/udev/ programs or others
Load the right module
Notify userspace
programs (GUI...)

識別 device driver modules



Firmware hotplugging 實做



See Documentation/firmware_class/ for a nice overview

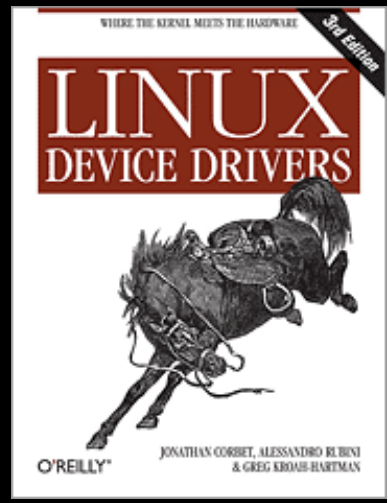
參考資訊

KernelTrap
http://kerneltrap.org/

- Forum website for kernel developers
- News, articles, whitepapers, discussions, polls, interviews

Linux Device Drivers, 3rd edition, Feb 2005

- Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, O'Reilly



Linux Kernel in a Nutshell, Dec 2006

- Greg Kroah-Hartman, O'Reilly