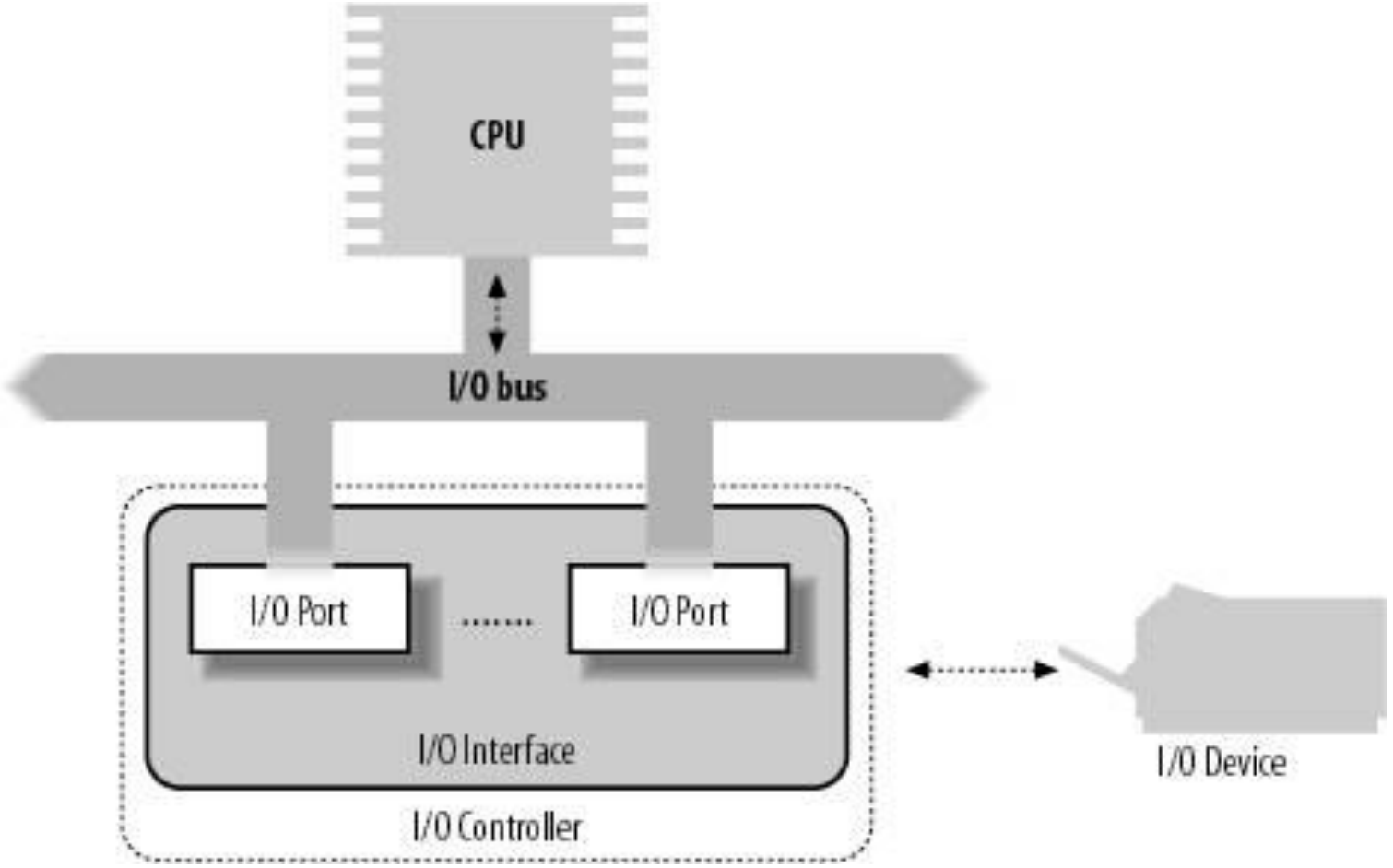
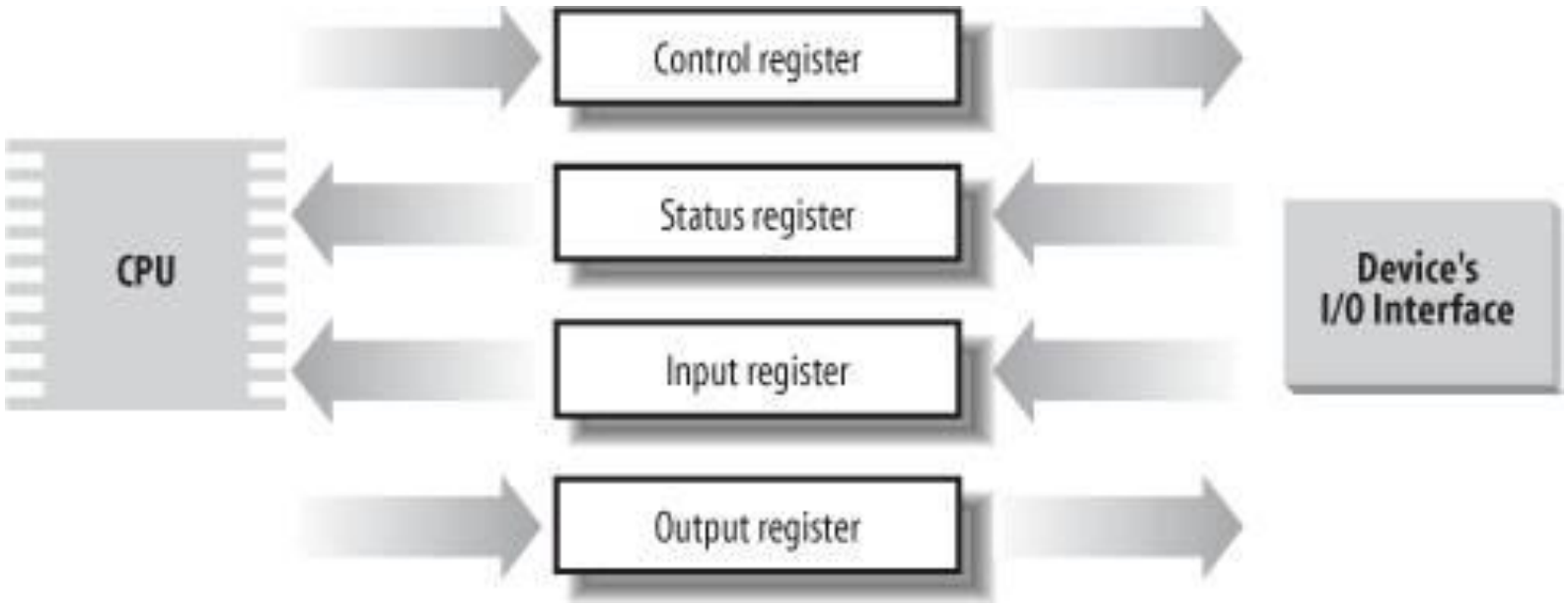
Basic Concept of Device Driver

Connect interrupt and device driver



Daniel P. Bovet, Marco Cesati, "Understanding the Linux Kernel," 3rd Edition

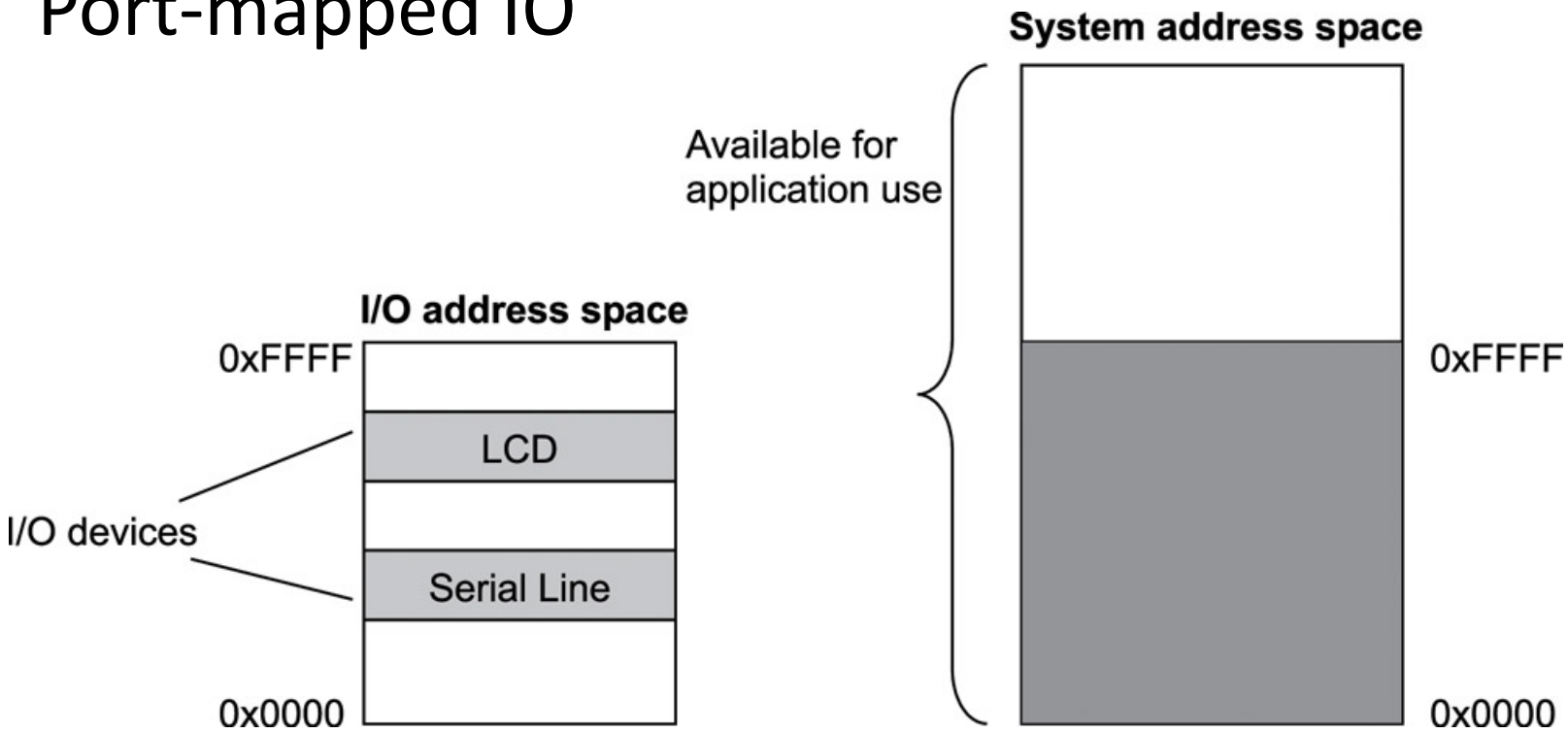
Connect interrupt and device driver



Daniel P. Bovet, Marco Cesati, "Understanding the Linux Kernel," 3rd Edition

Device Driver Basics

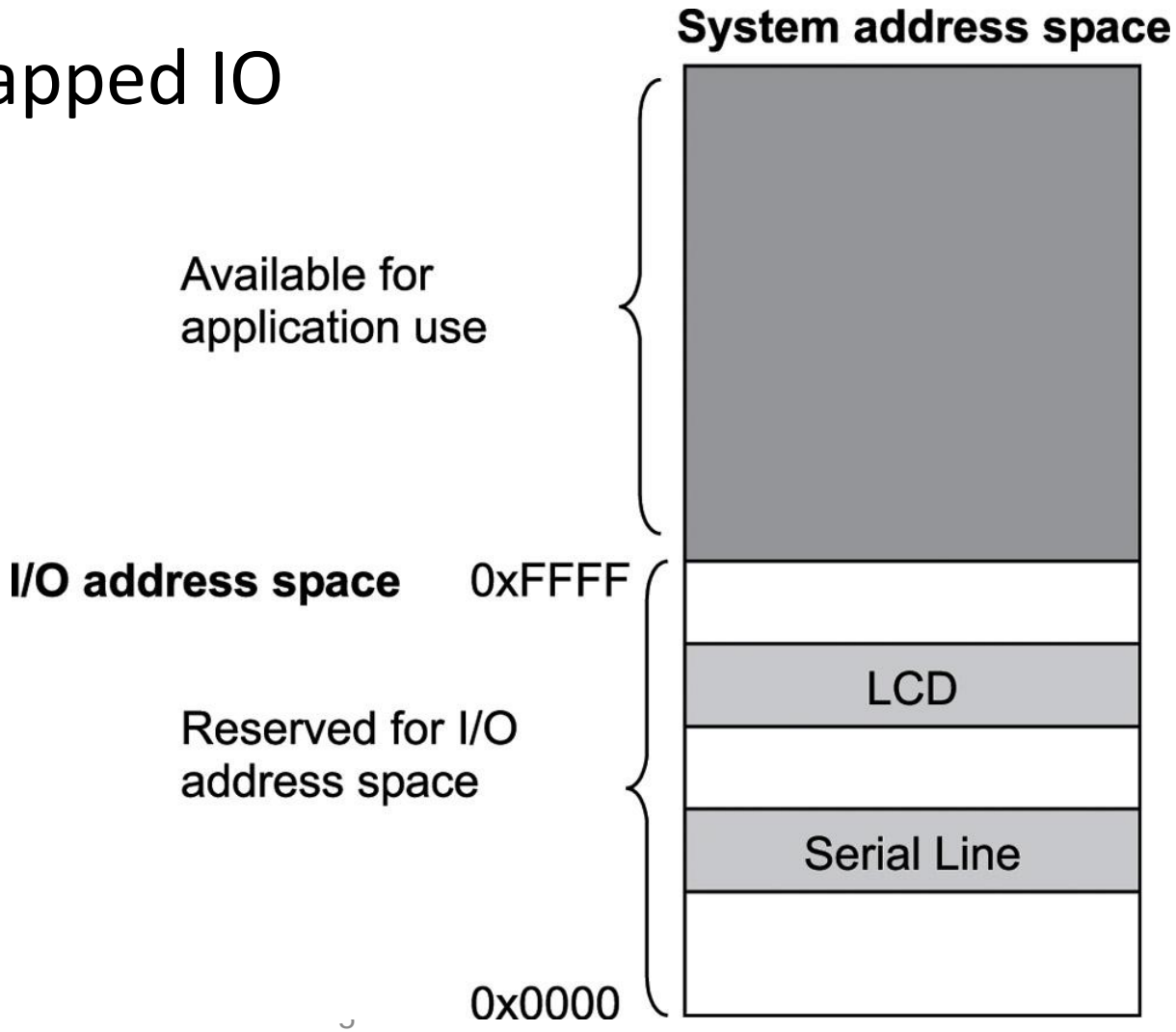
- Port-mapped IO



"Real-Time Concepts for Embedded Systems," by Qing Li and Carolyn Yao

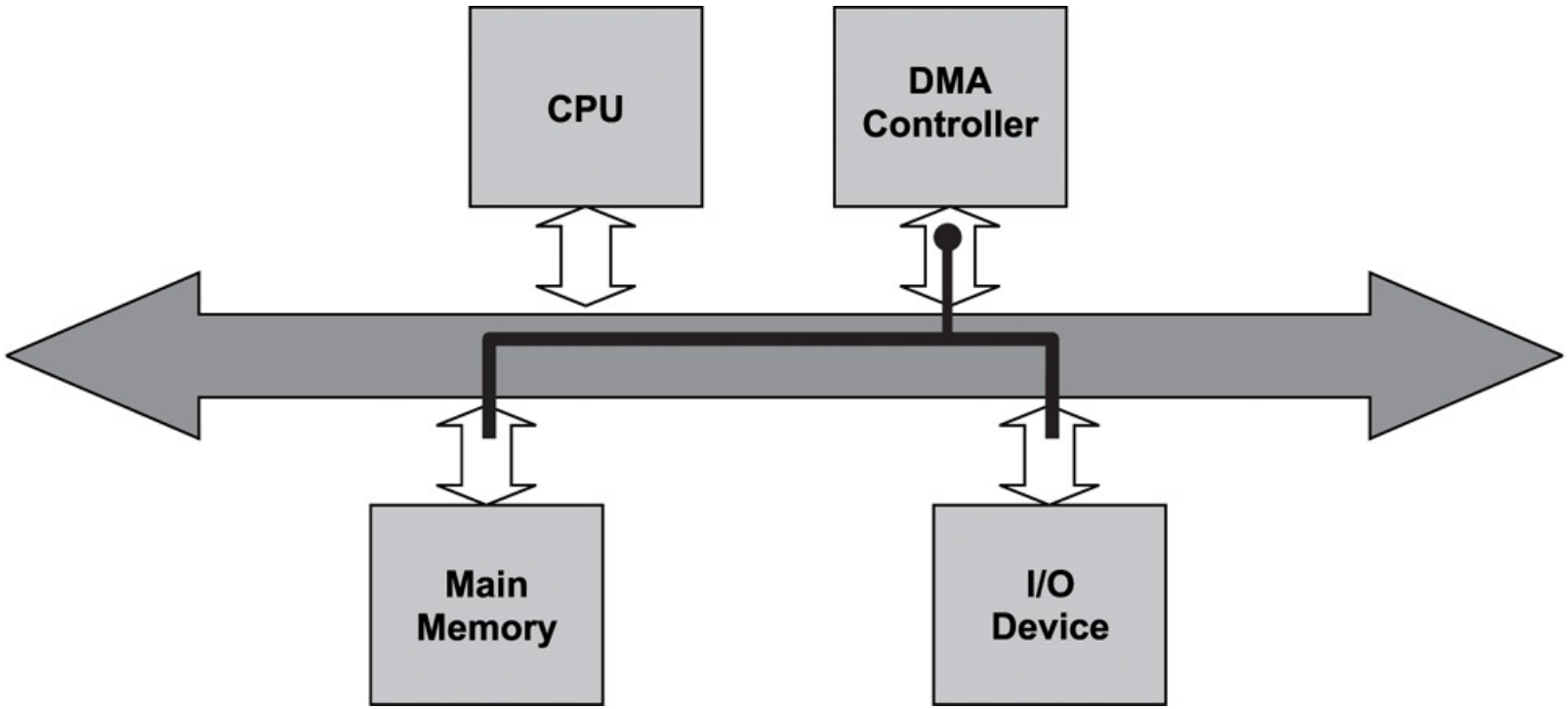
Device Driver Basics

- Memory-mapped IO

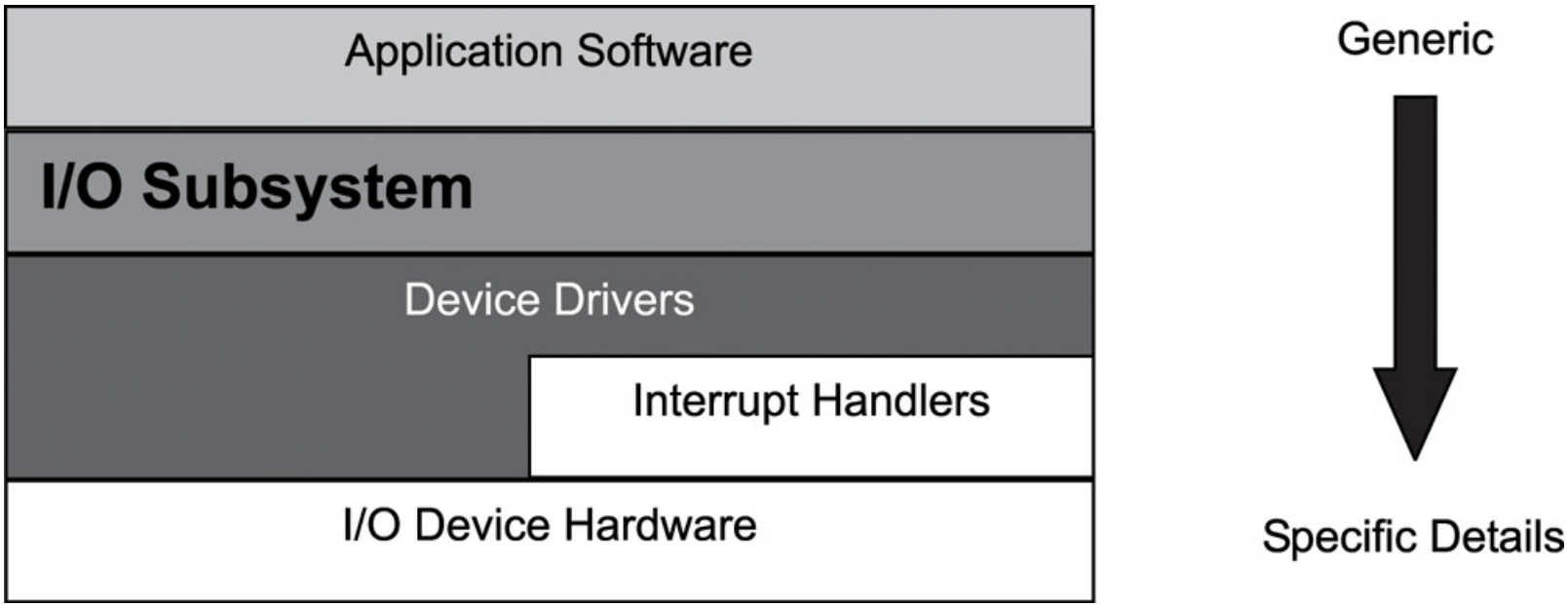


"Real-Time Concepts for Embedded Systems," by Qing Li and Carolyn Yao

Device Driver Basics



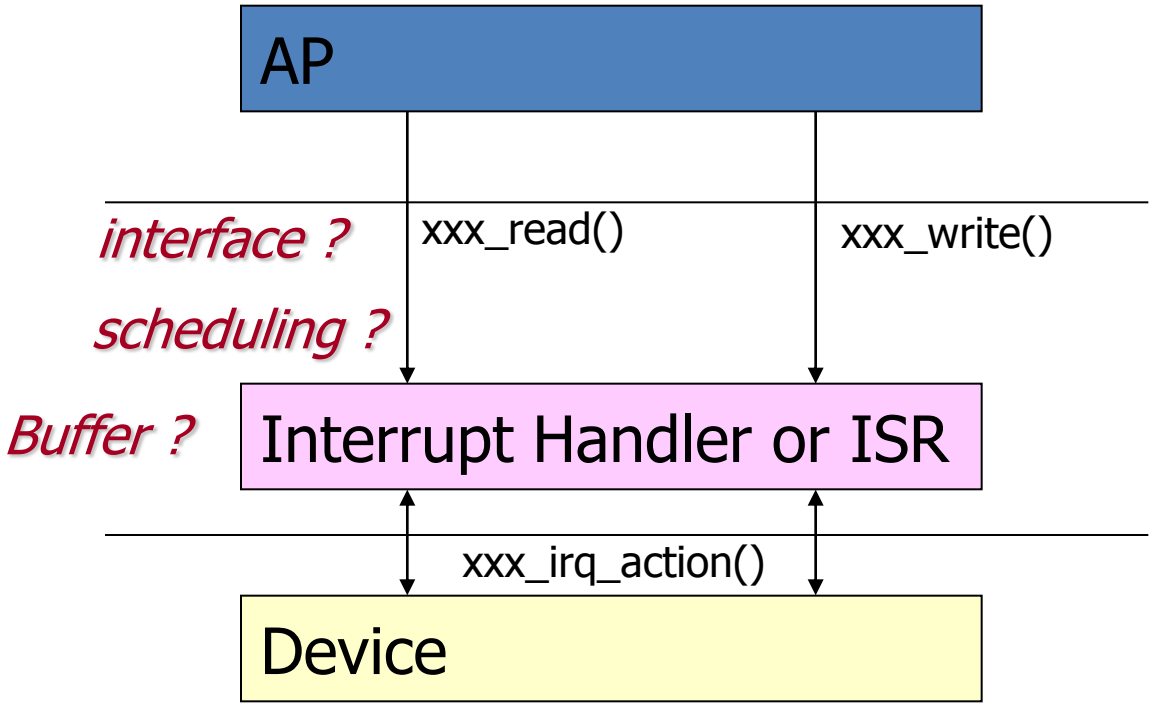
Device Driver Basics



"Real-Time Concepts for Embedded Systems," by Qing Li and Carolyn Yao

Connect interrupt and device driver

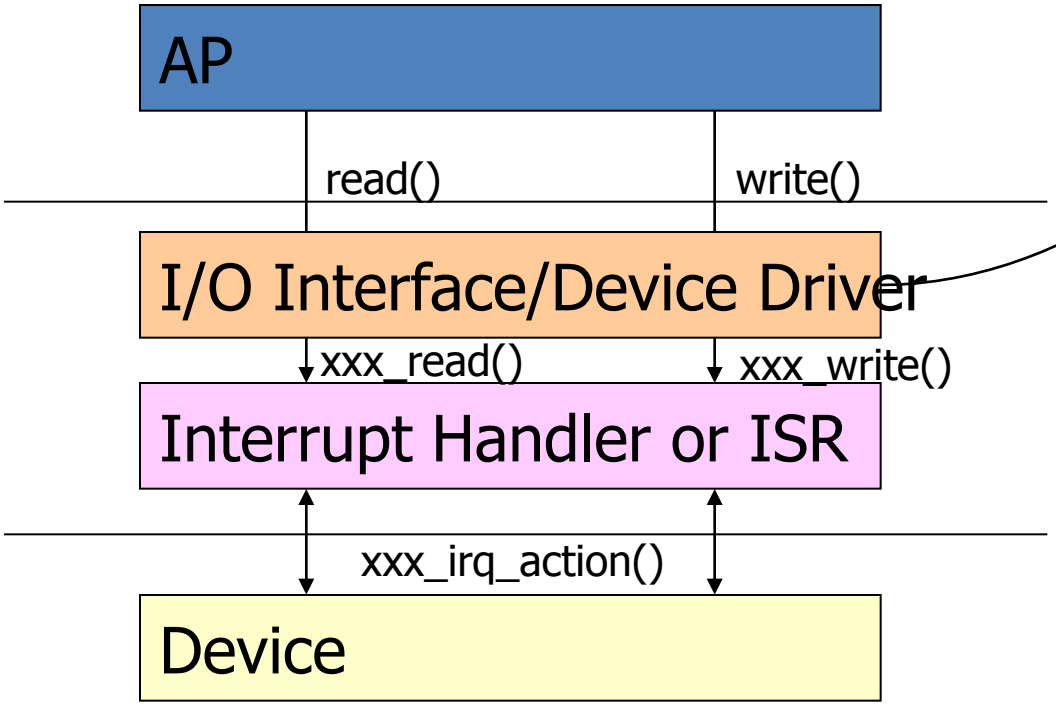
- AP + interrupt service routine



Connect interrupt and device driver

- AP + device driver + interrupt service routine

device file	open	read	write	close	...
tty0	tty0_open()	tty0_read()	tty0_write()	tty0_close()	
lp	lp_open()	lp_read()	lp_write()	lp_close()	...
...	...	...	...	...	...

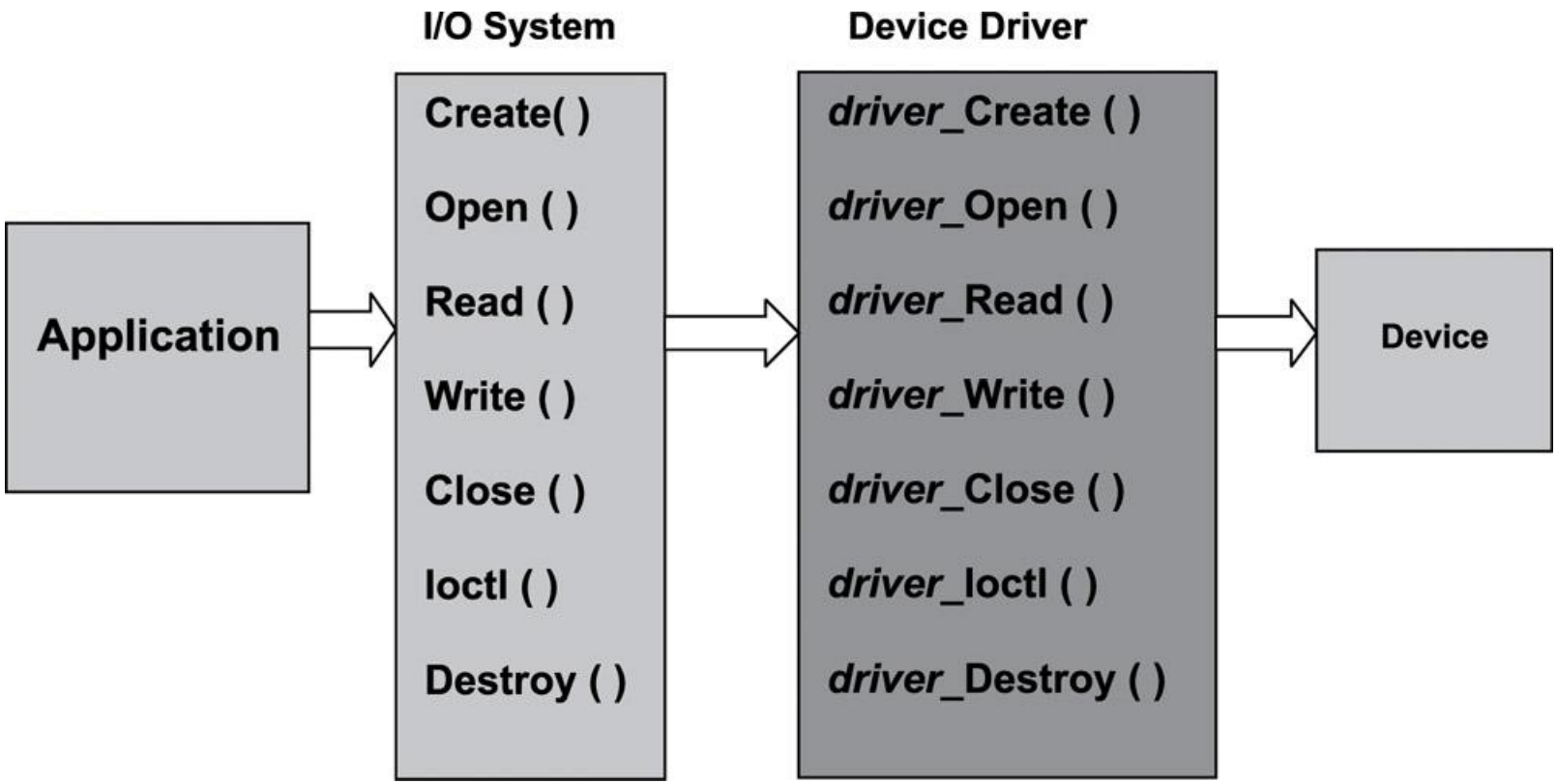


Connect interrupt and device driver

- Device file concept
  - Old-style device file

Name	Type	Major	Minor	Description
/dev/fd0	block	2	0	Floppy disk
/dev/hda	block	3	0	First IDE disk
/dev/hda2	block	3	2	Second primary partition of first IDE disk
/dev/hdb	block	3	64	Second IDE disk
/dev/hdb3	block	3	67	Third primary partition of second IDE disk
/dev/ttyp0	char	3	0	Terminal
/dev/console	char	5	1	Console
/dev/lp1	char	6	1	Parallel printer
/dev/ttyS0	char	4	64	First serial port
/dev/rtc	char	10	135	Real-time clock
/dev/null	char	1	3	Null device (black hole)

Device Driver Basics



"Real-Time Concepts for Embedded Systems," by Qing Li and Carolyn Yao

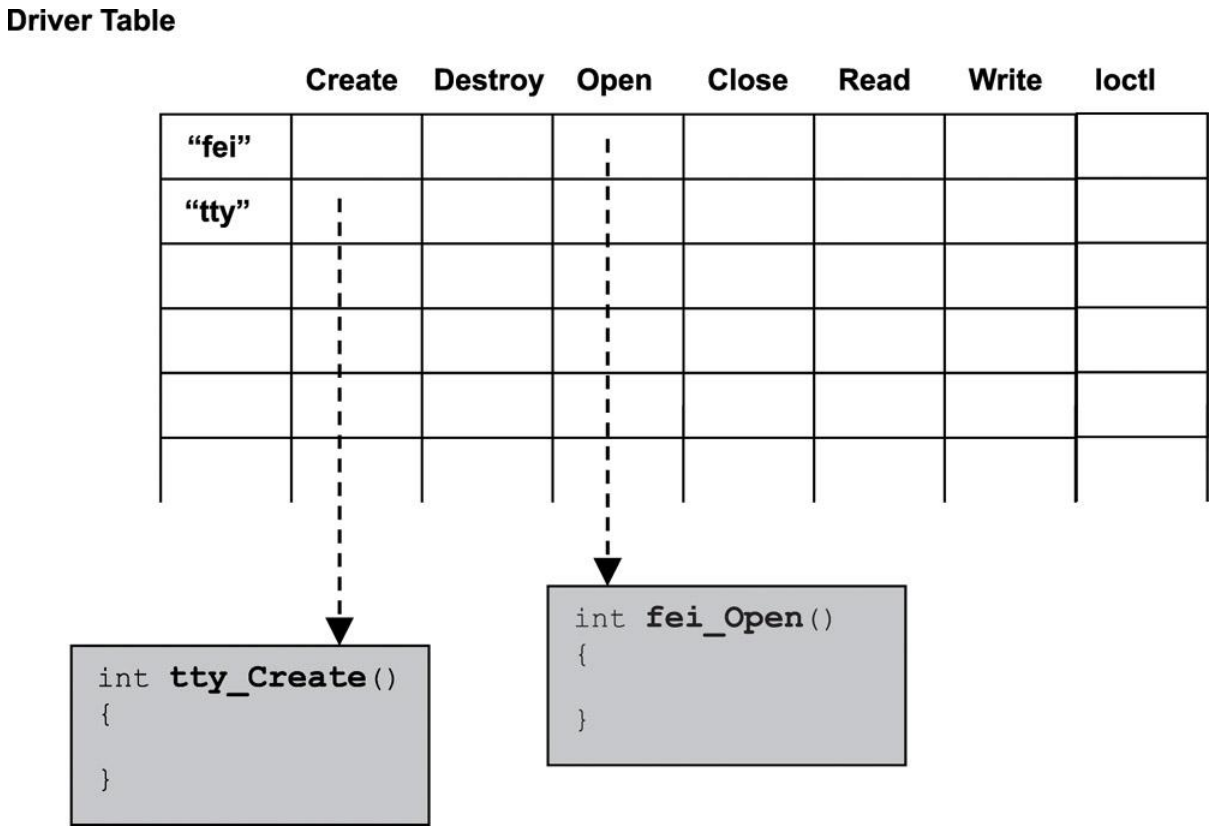
I/O subsystem (Cont.)

- Mapping generic functions to driver functions

```
typedef struct
{
        int (*Create)( );
        int (*Open) ( );
        int (*Read)( );
        int (*Write) ( );
        int (*Close) ( );
        int (*Ioctl) ( );
        int (*Destroy) ( );
} UNIFORM_IO_DRV;
```

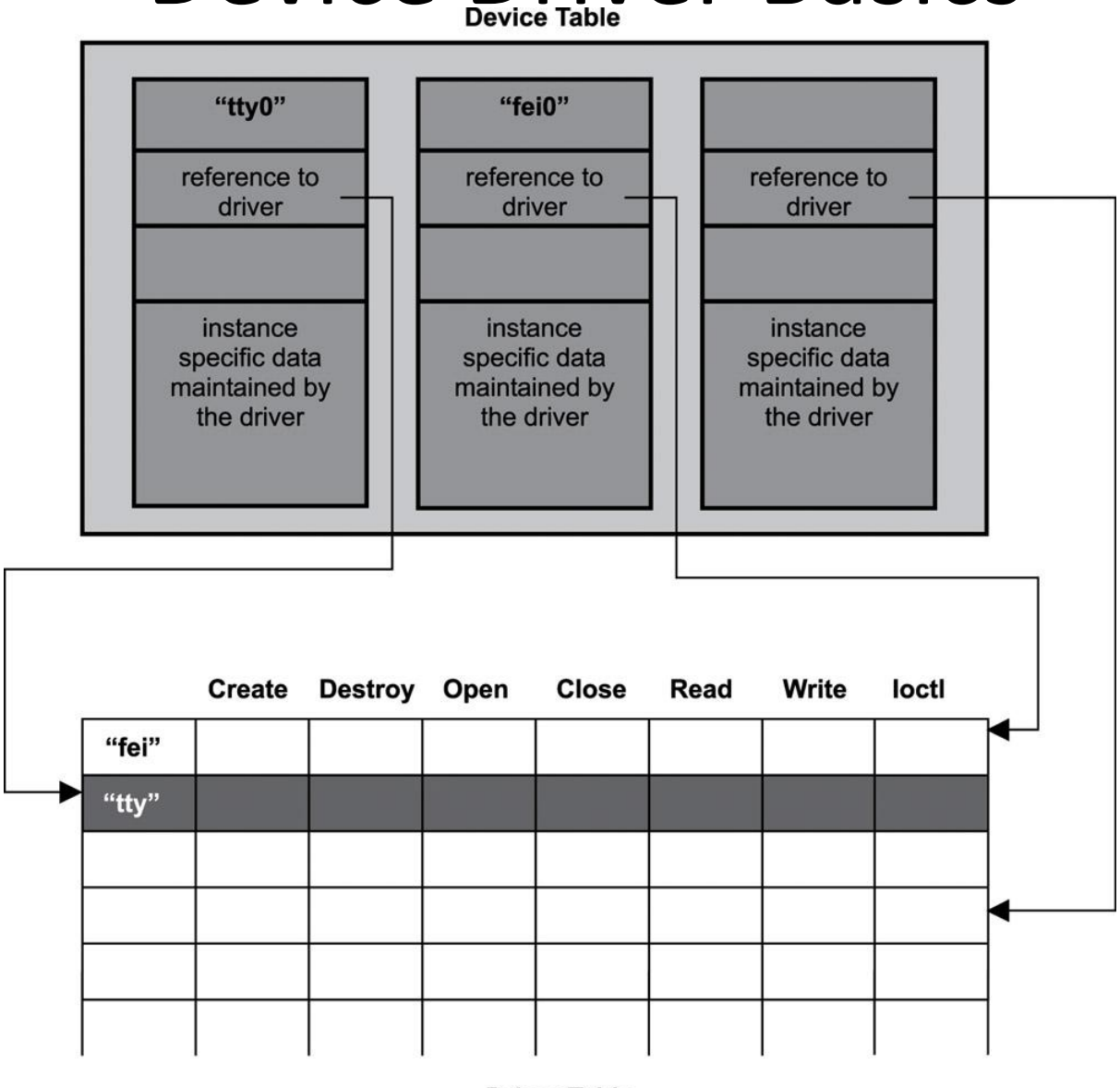
```
UNIFORM_IO_DRV ttyIOdrv;
ttyIOdrv.Create = tty_Create;
ttyIOdrv.Open = tty_Open;
ttyIOdrv.Read = tty_Read;
ttyIOdrv.Write = tty_Write;
ttyIOdrv.Close = tty_Close;
ttyIOdrv.Ioctl = tty_Ioctl;
ttyIOdrv.Destroy = tty_Destroy;
```

Device Driver Basics



"Real-Time Concepts for Embedded Systems," by Qing Li and Carolyn Yao

Device Driver Basics



"Real-Time Concepts for Embedded Systems," by Qing Li and Carolyn Yao

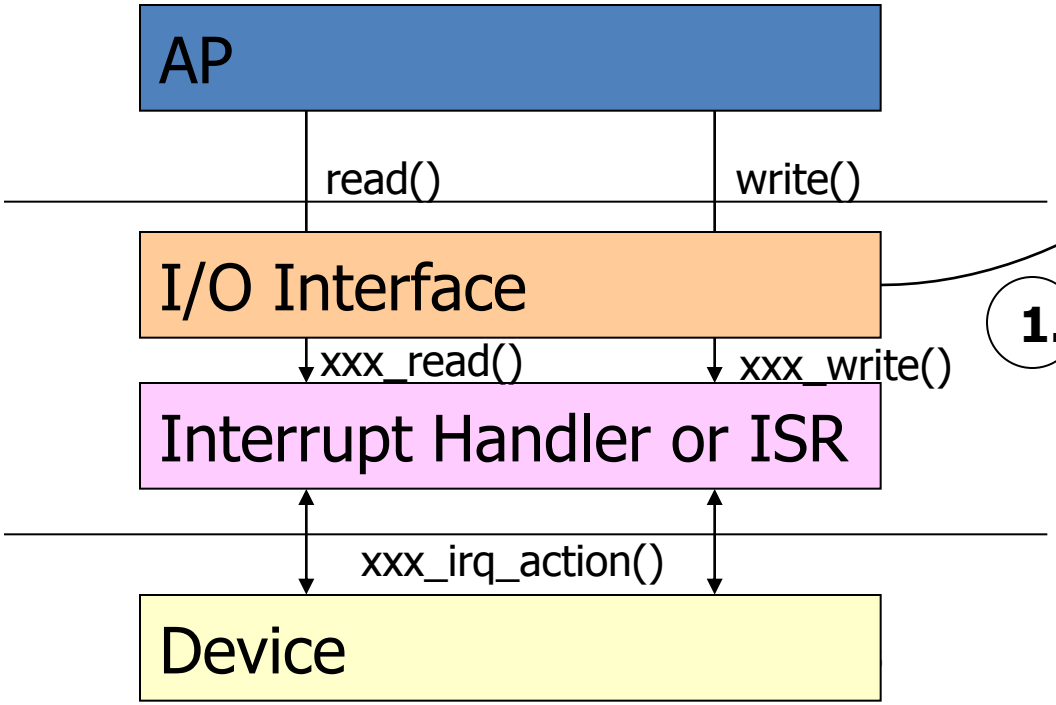
Connect interrupt and device driver

- Level of kernel support
  - No support at all
    - The application program interacts directly with the device's I/O ports by issuing suitable in and out assembly language instructions.
  - Minimal support
    - The kernel does not recognize the hardware device, but does recognize its I/O interface. User programs are able to treat the interface as a sequential device capable of reading and/or writing sequences of characters.
  - Extended support
    - The kernel recognizes the hardware device and handles the I/O interface itself. In fact, there might not even be a device file for the device.

Connect interrupt and device driver

- Registering a device driver

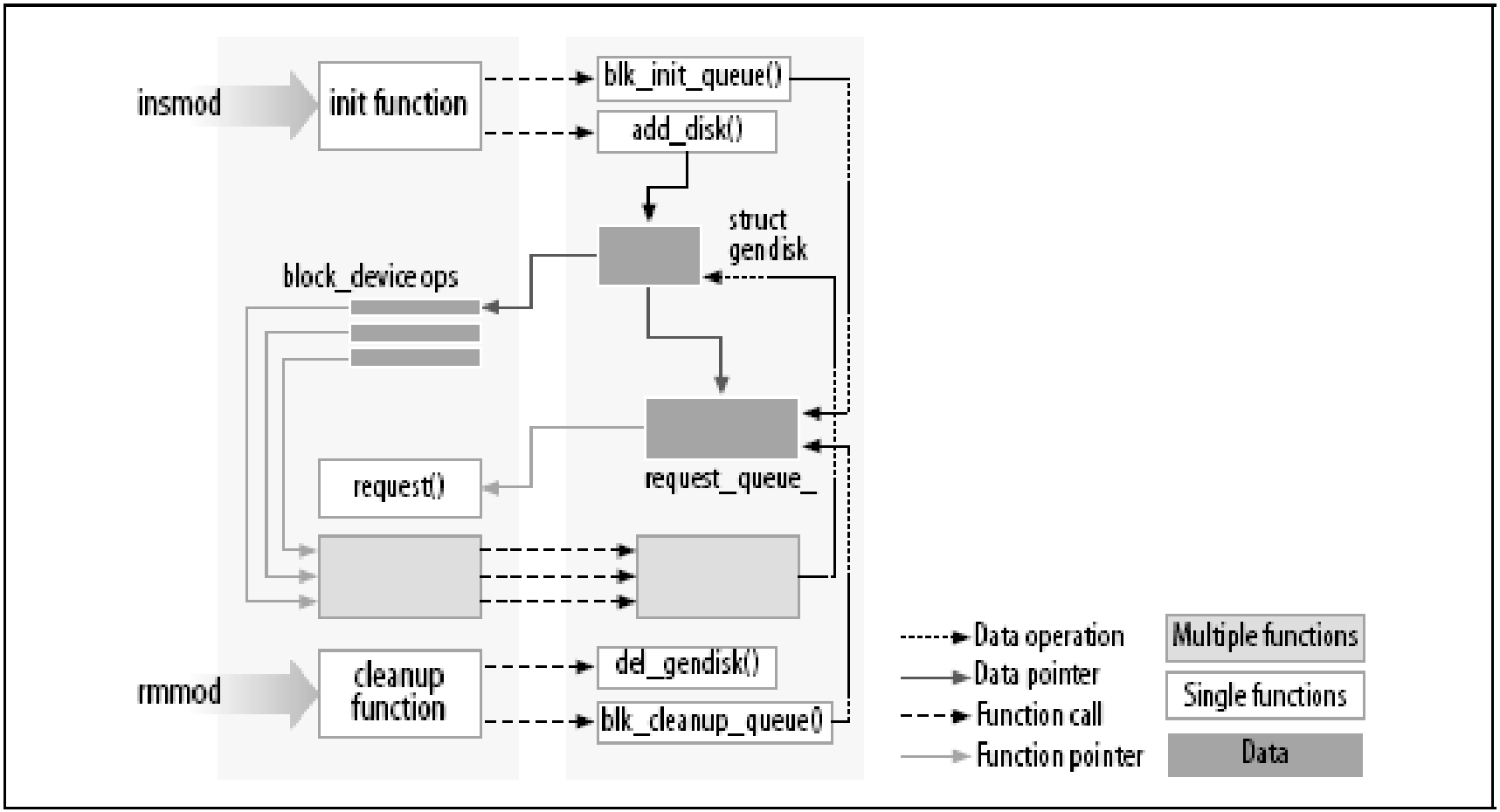
device file	open	read	write	close	...
tty0	tty0_open()	tty0_read()	tty0_write()	tty0_close()	
lp	lp_open()	lp_read()	lp_write()	lp_close()	...
...	...	...	...	...	...



2. Mapping interface

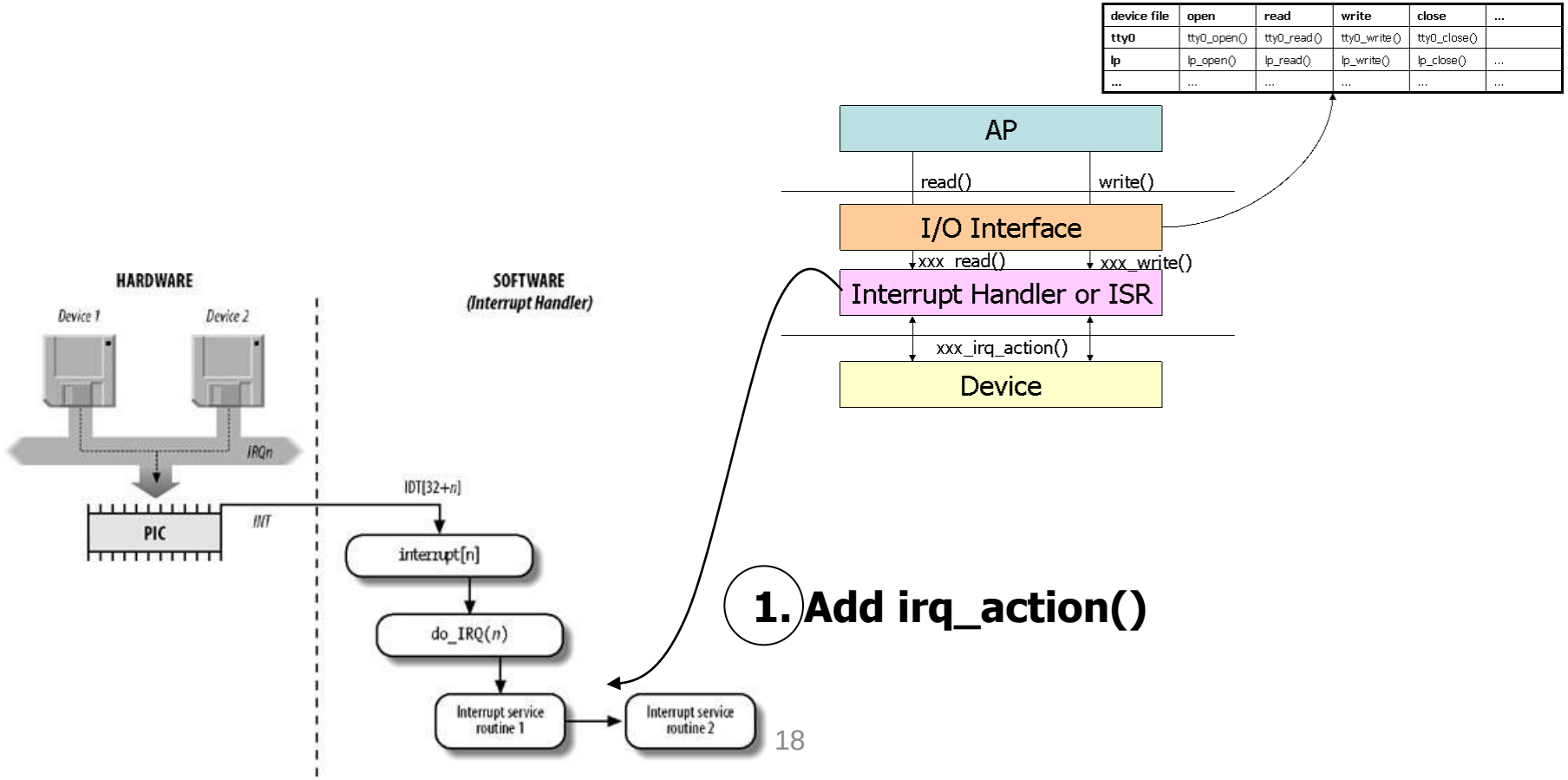
1. Register device driver

Connect interrupt and device driver



Connect interrupt and device driver

- Initializing a device driver



Device Driver Basics

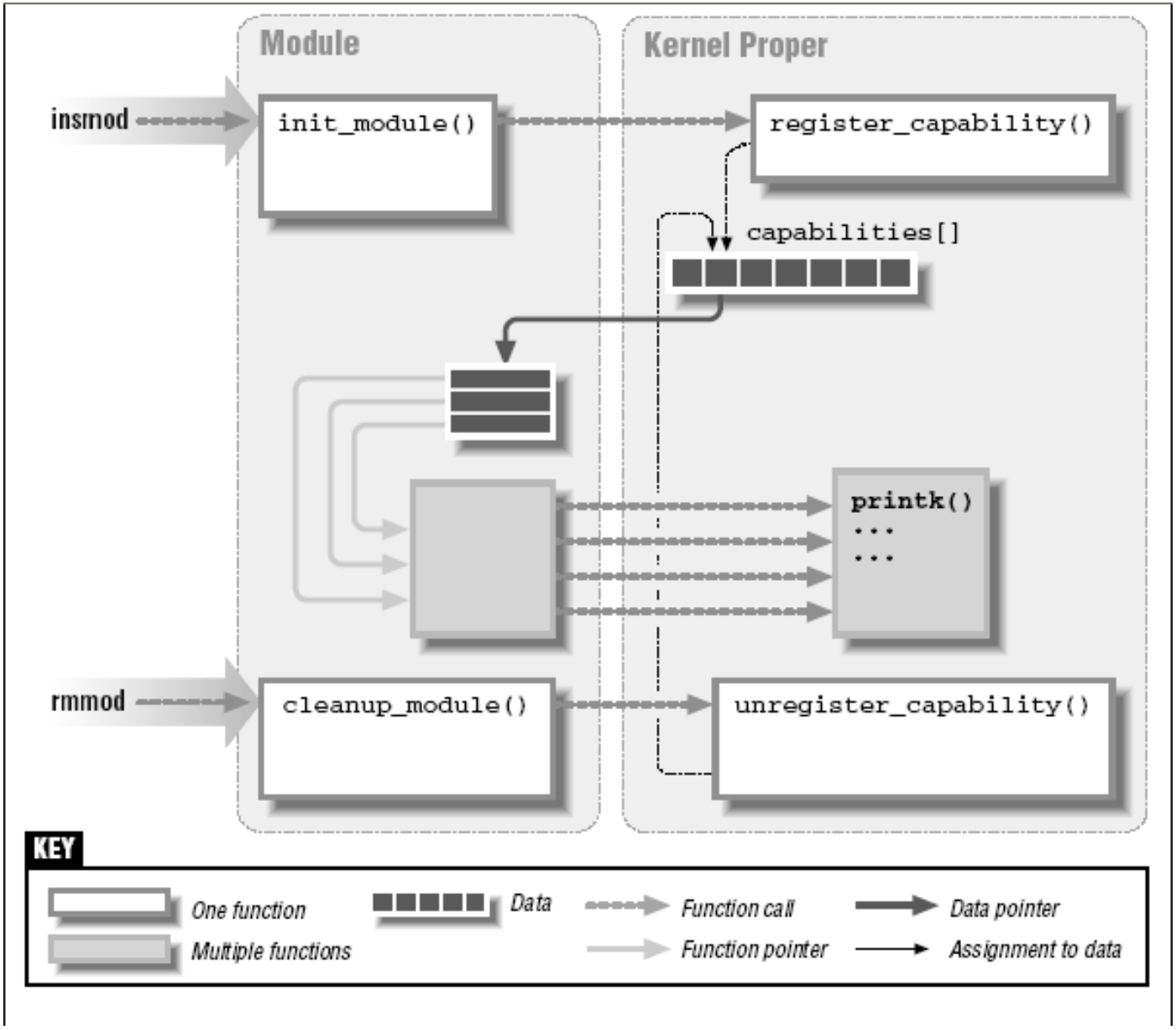
- Character-mode devices
  - for unstructured data transfers
  - data transfers typically take place in serial fashion
  - usually simple devices

- Block-mode devices
  - transfer data one block at time
  - underlying hardware imposes the block size
  - structure must be imposed on the data

Character Device Driver (Linux Example)

Outline

- Kernel module
- Char driver
- Linux examples
  - Write a kernel module
  - Writing a simple char driver


Module
Kernel Proper
insmod
init_module()
register_capability()
capabilities[]
printk()
...
...
rmmod
cleanup_module()
unregister_capability()
KEY
One function
Multiple functions
Data
Function call
Function pointer
Data pointer
Assignment to data

Loadable module

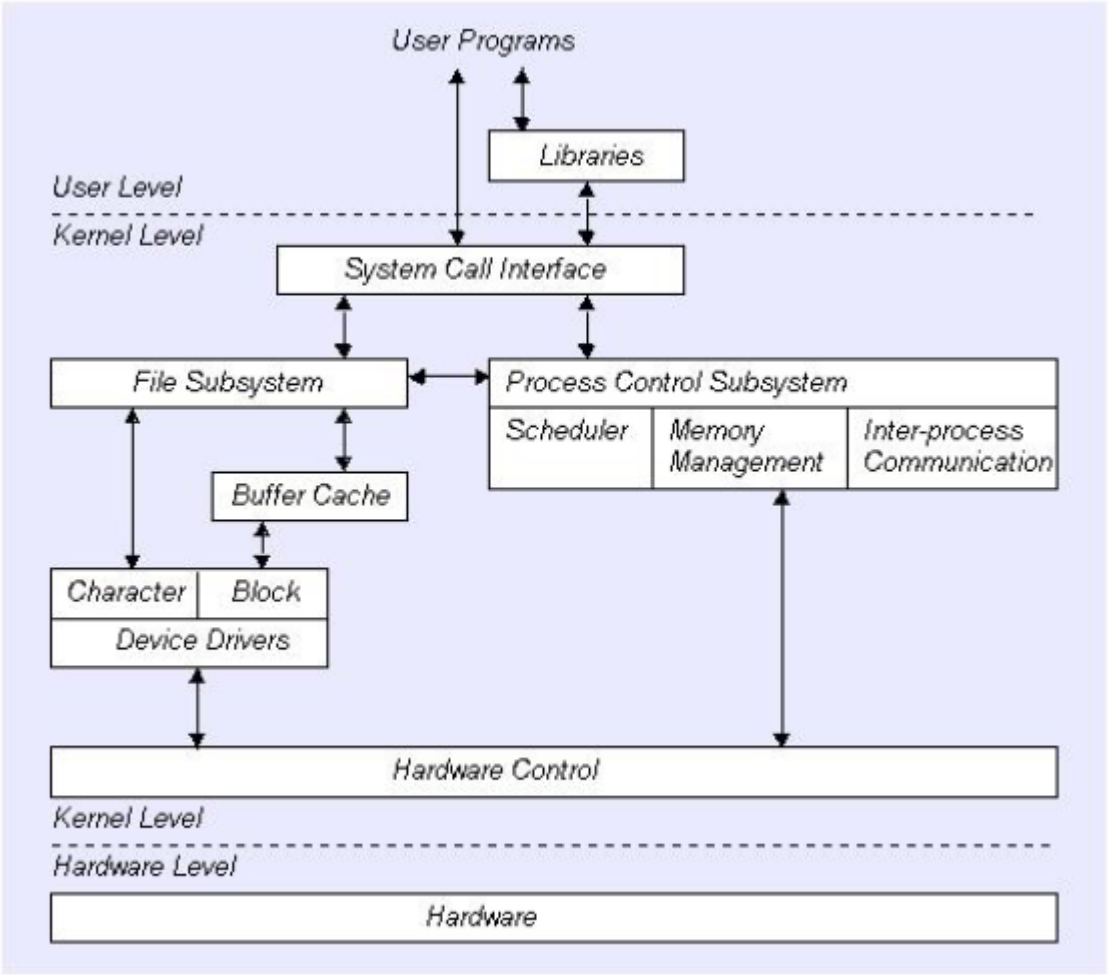
```
/*
 * hello.c  Hello, World! As a Kernel Module
 */

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

/*
 * hello_init  the init function, called when the module is loaded.
 * Returns zero if successfully loaded, nonzero otherwise.
 */
static int hello_init(void)
{
        printk(KERN_ALERT "I bear a charmed life.\n");
        return 0;
}

/*
 * hello_exit  the exit function, called when the module is removed.
 */
static void hello_exit(void)
{
        printk(KERN_ALERT "Out, out, brief candle!\n");
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Shakespeare");
```

Character devices

- can be accessed as a stream of bytes
- implements at least the open, close, read, and write system calls
- The text console (/dev/console) and the serial ports (/dev/ttyS0 and friends) are examples

User program/driver interface

- Char devices are accessed through names in the filesystem.



Major number and minor number

- Major number:
  - Identifies the driver associated with the device
  - For example, /dev/null and /dev/zero are both managed by driver 1
- Minor number:
  - Used by the kernel to determine exactly which device is being referred to
  - Example: differentiate between different partitions on an ide disk /dev/hda (i.e., /dev/hda1, /dev/hda2 all have the same major, but different minor numbers)
  - Could be used internally by the driver as an index into a local array of devices

```
crw-rw-rw-    1 root     root       1,   3 Apr 11  2002 null
crw-------    1 root     root      10,   1 Apr 11  2002 psaux
crw-------    1 root     root       4,   1 Oct 28 03:04 tty1
crw-rw-rw-    1 root     tty        4,  64 Apr 11  2002 ttyS0
crw-rw----    1 root     uucp       4,  65 Apr 11  2002 ttyS1
crw--w----    1 vcsa     tty        7,   1 Apr 11  2002 vcs1
crw--w----    1 vcsa     tty        7, 129 Apr 11  2002 vcsa1
crw-rw-rw-    1 root     root       1,   5 Apr 11  2002 zero
```

Register device drivers

- 2.4
  - register_chrdev()
  - Example: register_chrdev(6, “lp”, &lp_fops)
- 2.6
  - device_register( )
  - driver_register( )

Char device

```
int register_chrdev_region(dev_t first, unsigned int count,
                           char *name);
```

```
int alloc_chrdev_region(dev_t *dev, unsigned int firstminor,
                        unsigned int count, char *name);
```

```
if (scull_major) {
    dev = MKDEV(scull_major, scull_minor);
    result = register_chrdev_region(dev, scull_nr_devs, "scull");
} else {
    result = alloc_chrdev_region(&dev, scull_minor, scull_nr_devs,

              "scull");
      scull_major = MAJOR(dev);
  }
  if (result < 0) {
      printk(KERN_WARNING "scull: can't get major %d\n", scull_major);
      return result;
  }
```

File operations

- struct module *owner
- loff_t (*llseek) (struct file *, loff_t, int);
- ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
- ssize_t (*aio_read)(struct kiocb *, char __user *, size_t, loff_t);
- ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
- ssize_t (*aio_write)(struct kiocb *, const char __user *, 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 (*flush) (struct file *);
- int (*release) (struct inode *, struct file *);
- int (*fsync) (struct file *, struct dentry *, int);
- int (*aio_fsync)(struct kiocb *, int);
- 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 (*sendfile)(struct file *, loff_t *, size_t, read_actor_t, void *);
- 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);
- int (*check_flags)(int)
- int (*dir_notify)(struct file *, unsigned long);

Device file operations

```
struct file_operations scull_fops = {
    .owner =    THIS_MODULE,
    .llseek =   scull_llseek,
    .read =     scull_read,
    .write =    scull_write,
    .ioctl =    scull_ioctl,
    .open =     scull_open,
    .release =  scull_release,
};
```

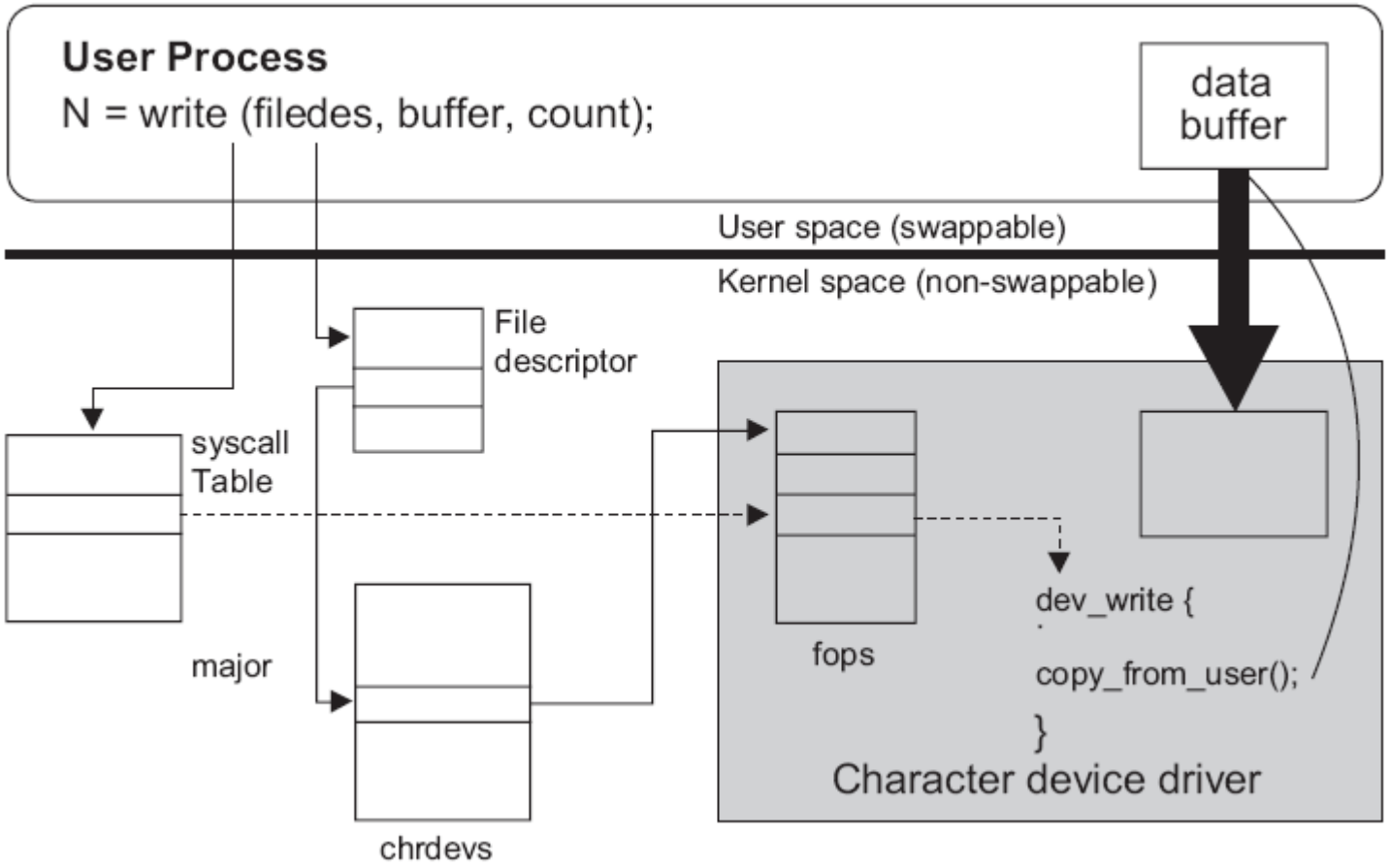
```
struct cdev *my_cdev = cdev_alloc( );
my_cdev->ops = &my_fops;
```

```
struct scull_dev {
    struct scull_qset *data;  /* Pointer to first quantum set */
    int quantum;              /* the current quantum size */
    int qset;                 /* the current array size */
    unsigned long size;       /* amount of data stored here */
    unsigned int access_key;  /* used by sculluid and scullpriv */
    struct semaphore sem;     /* mutual exclusion semaphore     */
    struct cdev cdev;     /* Char device structure      */
};
```

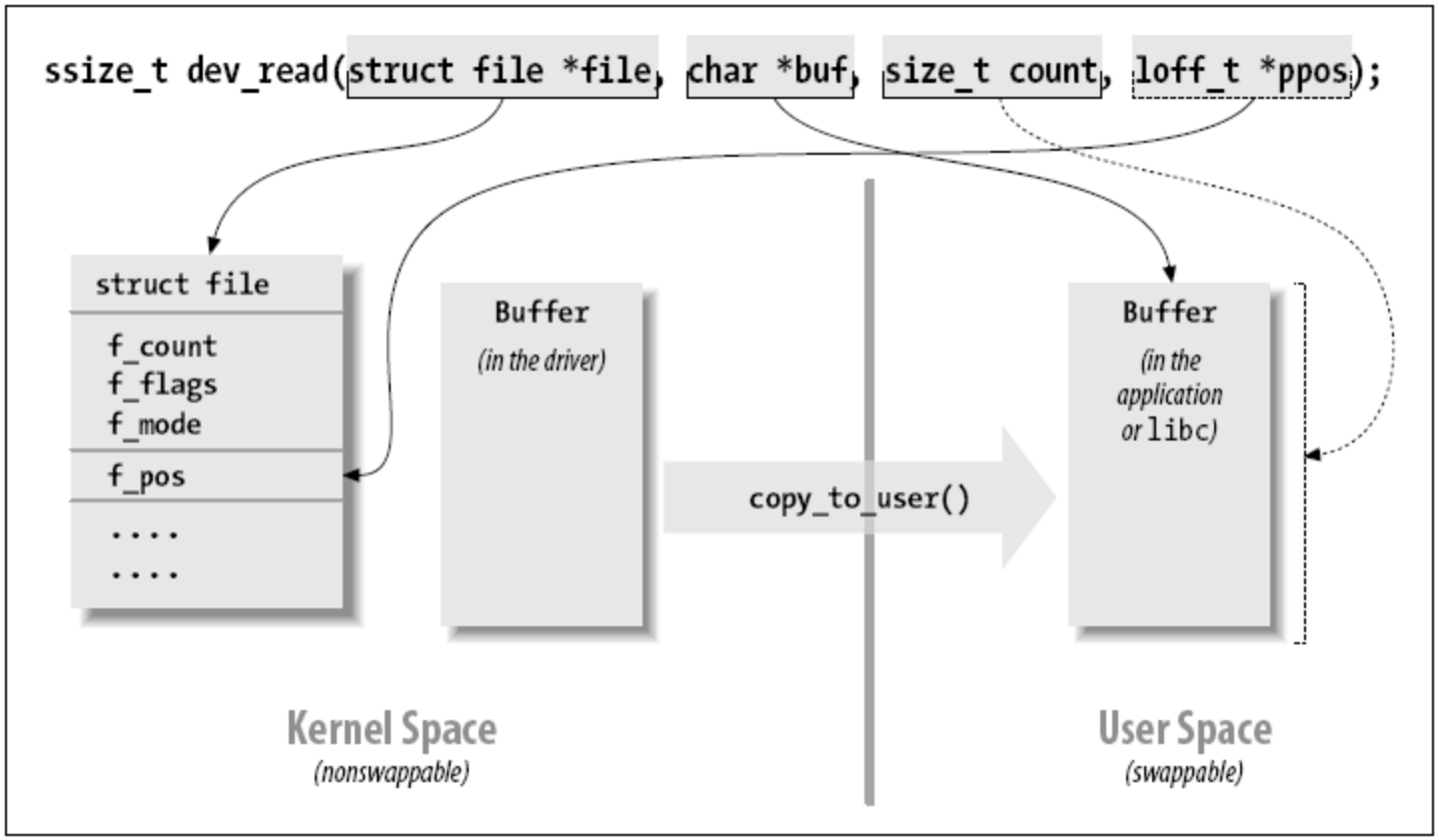
Device file operations

```
   static void scull_setup_cdev(struct scull_dev *dev, int index)
   {
       int err, devno = MKDEV(scull_major, scull_minor + index);

       cdev_init(&dev->cdev, &scull_fops);
       dev->cdev.owner = THIS_MODULE;
       dev->cdev.ops = &scull_fops;
       err = cdev_add (&dev->cdev, devno, 1);
       /* Fail gracefully if need be */
       if (err)
       printk(KERN_NOTICE "Error %d adding scull%d", err, index);
   }

int scull_open(struct inode *inode, struct file *filp)
{
    struct scull_dev *dev; /* device information */

    dev = container_of(inode->i_cdev, struct scull_dev, cdev);
    filp->private_data = dev; /* for other methods */

    /* now trim to 0 the length of the device if open was write-only */
    if ( (filp->f_flags & O_ACCMODE) == O_WRONLY) {
        scull_trim(dev); /* ignore errors */
    }
    return 0;          /* success */
}
```

Device write



Device read


ssize_t dev_read(struct file *file, char *buf, size_t count, loff_t *ppos);
struct file
f_count
f_flags
f_mode
f_pos
....
....
Buffer
(in the driver)
copy_to_user()
Buffer
(in the application or libc)
Kernel Space
(nonswappable)
User Space
(swappable)

Linux examples

Write a kernel module
Writing a simple char driver

A Simple Module

linux/module.h – the required header file must be included by a module source.
init_module – the module initialization function initializes a loadable module entry.
cleanup_module – the module cleanup function is invoked when kernel removes the module.

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

int init_module(void)
{
        /* Initialization code here */
}

void cleanup_module(void)
{
        /* Cleanup code here */
}
```

A Simple Module (Cont.)

```
#include <linux/module.h>
#include <linux/init.h>
```

```
static init initialization_function(void)
{
        /* Initialization code here */
}

static void cleanup_function(void)
{
        /* Cleanup code here */
}

module_init(initialization_function);
module_exit(cleanup_function);
```

Kernel version mismatch

```
#define MODULE
#include <linux/module.h>

int init_module(void)          { printk("<1>Hello, world\n"); return 0; }
void cleanup_module(void)    { printk("<1>Goodbye cruel world\n"); }
```

```
root# uname -r
2.4.20-8

root# rpm -qa |grep kernel
kernel-source-2.4.20-8
```

```
./hello.o: kernel-module version mismatch
        ./hello.o was compiled for kernel version 2.4.20
        while this kernel is version 2.4.20-8.
```

```
root# gcc -c -I/usr/src/linux-2.4.20-8/include hello.c
root# insmod ./hello.o
Warning: loading ./hello.o will taint the kernel: no license
   See http://www.tux.org/lkml/#export-tainted for information about tainted modules
Module hello loaded, with warnings
```

Check driver

```
root# dmesg -c
Hello, world
```

```
root# lsmod
Module                  Size  Used by    Tainted: P
hello                   764   0  (unused)
```

```
root# rmmod hello
root# dmesg -c
Goodbye cruel world
```

```
root# rmmod hello
rmmod: module hello is not loaded
```

```
root# /sbin/modinfo ./hello.o
filename:       ./hello.o
description: <none>
author:         <none>
license:        "Dual BSD/GPL"
```

License issues

```
#define MODULE
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
```

2.6 kernel

```
obj-m := hello.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /usr/src/linux-headers-2.6.24-26-generic M=$(PWD) clean
```

```
$ make
make -C /lib/modules/2.6.24-26-generic/build M=/home/driver/LDD2010/lab1 modules
make[1]: Entering directory `/usr/src/linux-headers-2.6.24-26-generic'
  CC [M]    /home/driver/LDD2010/lab1/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC          /home/driver/LDD2010/lab1/hello.mod.o
  LD [M]    /home/driver/LDD2010/lab1/hello.ko
make[1]: Leaving directory `/usr/src/linux-headers-2.6.24-26-generic'
```

2.6 kernel

```
$ ls
Makefile  Module.symvers  hello.c  hello.ko  hello.mod.c  hello.mod.o  hello.o
```

```
$ sudo insmod ./hello.ko
```

Char driver

```
/* global variables of the driver */
int hello_major = 120;                  /* major number */
char *hello_buffer;                     /* buffer to store data */
```

```
int hello_init(void)
{
	int result;

	/* registering device */
	result = register_chrdev(hello_major, "hello", &hello_fops);
	if (result < 0) {
		printk("<1>hello: cannot obtain major number %d\n", hello_major);
		return result;
	}

	/* allocating memory for the buffer */
	hello_buffer = kmalloc(1, GFP_KERNEL);
	if (!hello_buffer) {
		result = -ENOMEM;
		goto fail;
	}
```

```
    memset(hello_buffer, 0, 1);
    printk("<1>Inserting hello module\n");
    return 0;

  fail:
    hello_exit();
    return result;
}

module_init(hello_init);
```

```
void hello_exit(void)
{
        /* free the major number */
        unregister_chrdev(hello_major, "hello");

        /* free buffer */
        if (hello_buffer) {
                kfree(hello_buffer);
        }

        printk("<1>Removing hello module\n");
}

module_exit(hello_exit);
```

```
/* file operations */
struct file_operations hello_fops = {
   read:hello_read,
   write:hello_write,
   open:hello_open,
   release:hello_release
};
```

```
int hello_open(struct inode *inode, struct file *filp)
{
      return 0;                  /* success */
}
```

```
int hello_release(struct inode *inode, struct file *filp)
{
      return 0;                  /* success */
}
```

```
ssize_t hello_read(struct file * filp, char *buf,
                   size_t count, loff_t * f_pos)
{
    /* copy from hello_buffer to buf */
    copy_to_user(buf, hello_buffer, 1);

    /* changing reading position as best suits */
    if (*f_pos == 0) {
        *f_pos += 1;
        return 1;
    } else {
        return 0;
    }
}
```

```
ssize_t hello_write(struct file * filp, char *buf,
                    size_t count, loff_t * f_pos)
{
        char *tmp;
        tmp = buf + count - 1;

        /* copy from tmp to hello_buffer */
        copy_from_user(hello_buffer, tmp, 1);

        return 1;
}
```

Whole driver program

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/config.h>
#include <linux/kernel.h>        /* printk() */
#include <linux/slab.h>          /* kmalloc() */
#include <linux/fs.h>            /* everything... */
#include <linux/errno.h>         /* error codes */
#include <linux/types.h>         /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h>         /* O_ACCMODE */
#include <asm/system.h>            /* cli(), *_flags */
#include <asm/uaccess.h>           /* copy_from/to_user */
```

```
MODULE_LICENSE("Dual BSD/GPL");

/* global variables of the driver */

/* the open method */

/* the release method */

/* the read method */

/* the write method */

/* the file_operations structure */

/* the cleanup function */

/* the initialization function */

/*the declaration of the initialization and cleanup functions */
```

Compile and install

```
root# gcc -c -DMODULE -D__KERNEL__ -I/usr/src/linux-2.4.20-8/include
hello.c
```

```
root# mknod /dev/hello c 120 0
root# ls -al /dev/hello
crw-r--r--      1 root      root      120,     0 Jun 27 04:32 /dev/hello
```

```
root# chmod 666 /dev/hello
root# ls -al /dev/hello
crw-rw-rw-      1 root      root      120,     0 Jun 27 04:32 /dev/hello
```

Serial Drivers

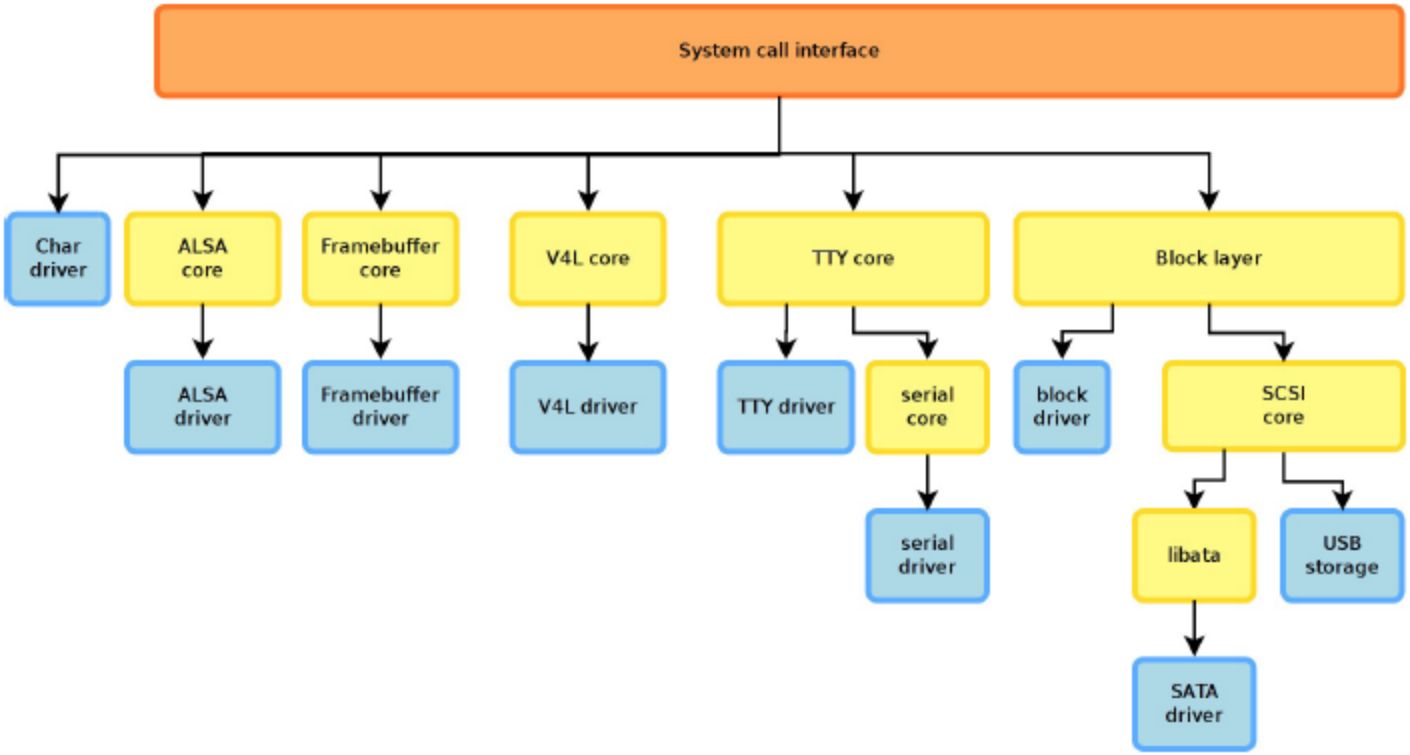
Outline

- Device driver framework
- Serial driver framework
- UART driver example
- TTY driver example
- User applications

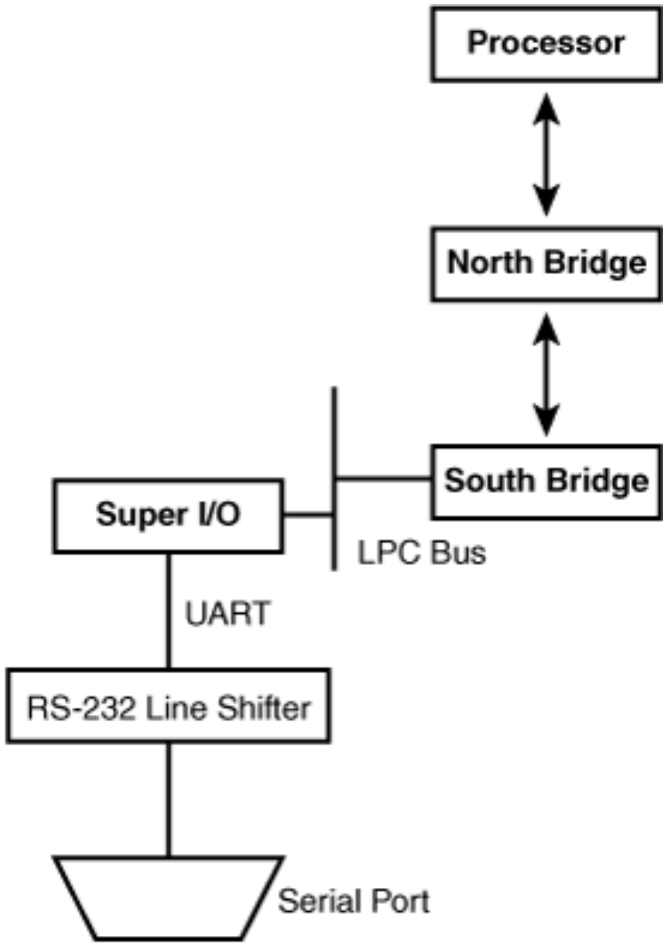
Kernel framework

- Most device drivers are not directly implemented as character devices or block devices
- They are implemented under a *framework*, specific to a device type (framebuffer, V4L, serial, etc.)
  - The framework allows to factorize the common parts of drivers for the same type of devices
  - From userspace, they are still seen as normal character devices
  - The framework allows to provide a coherent userspace interface (ioctl numbering and semantic, etc.) for every type of device, regardless of the driver

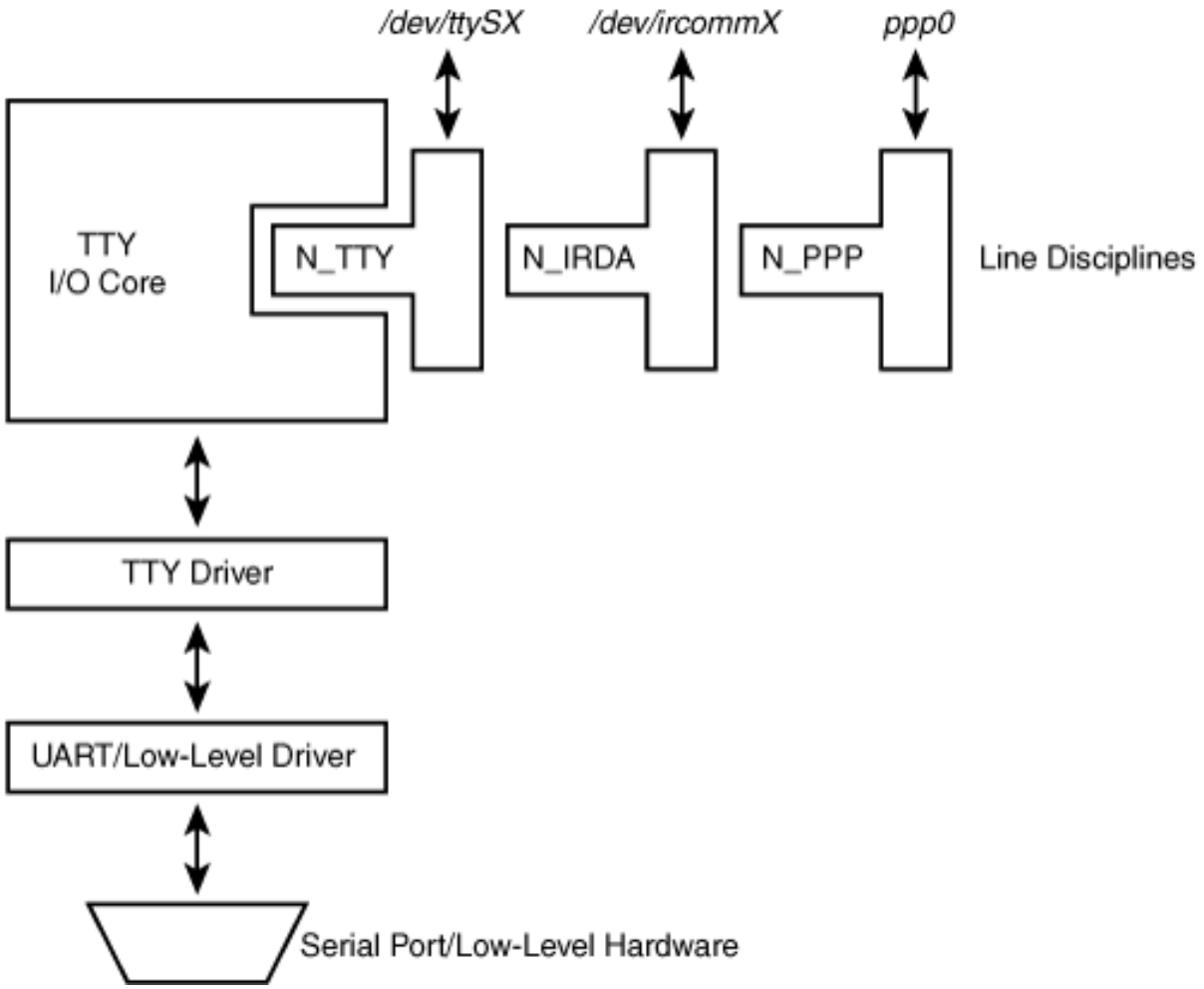
Example of frameworks



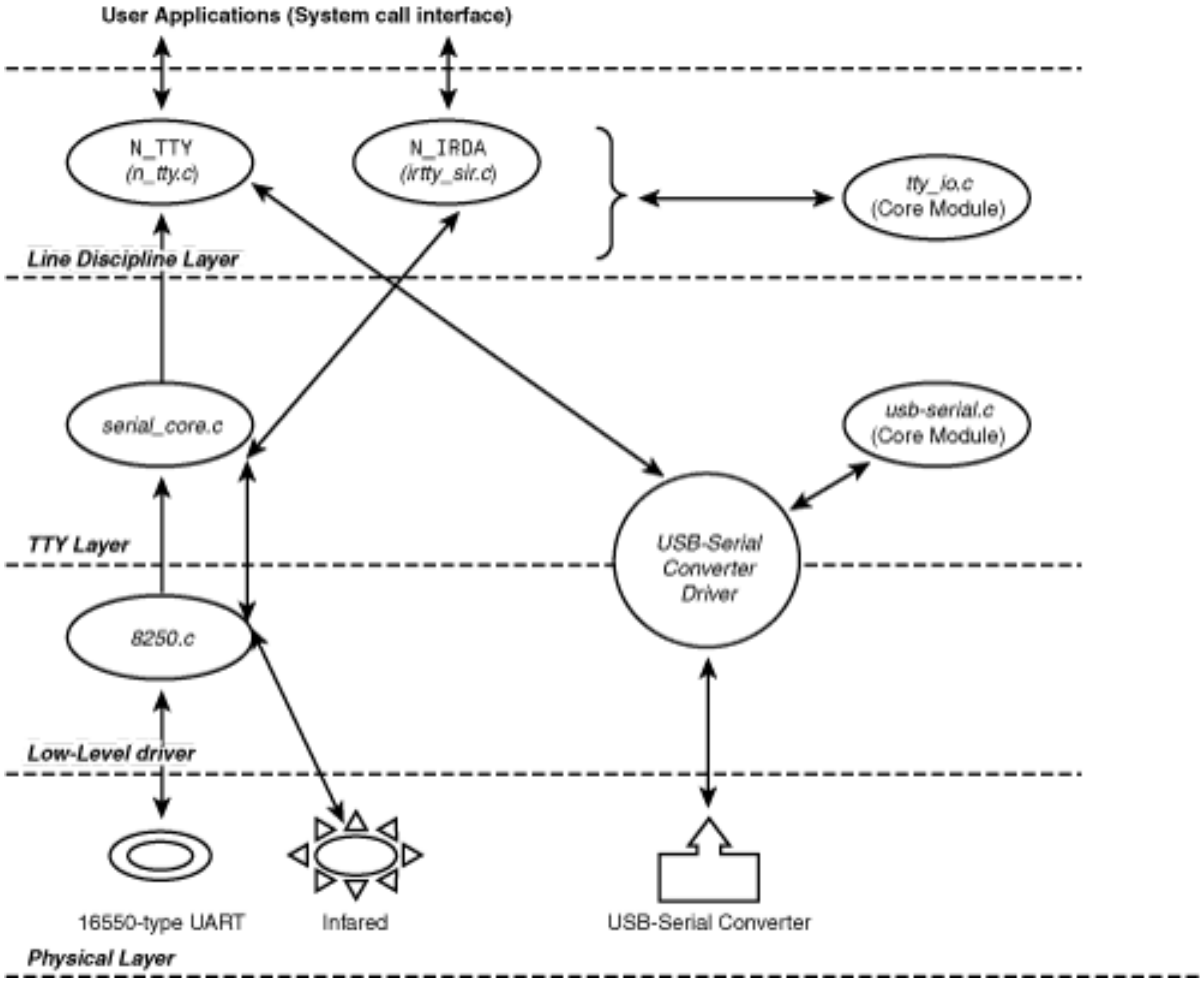
PC and serial ports



Layer structure in the serial subsystem



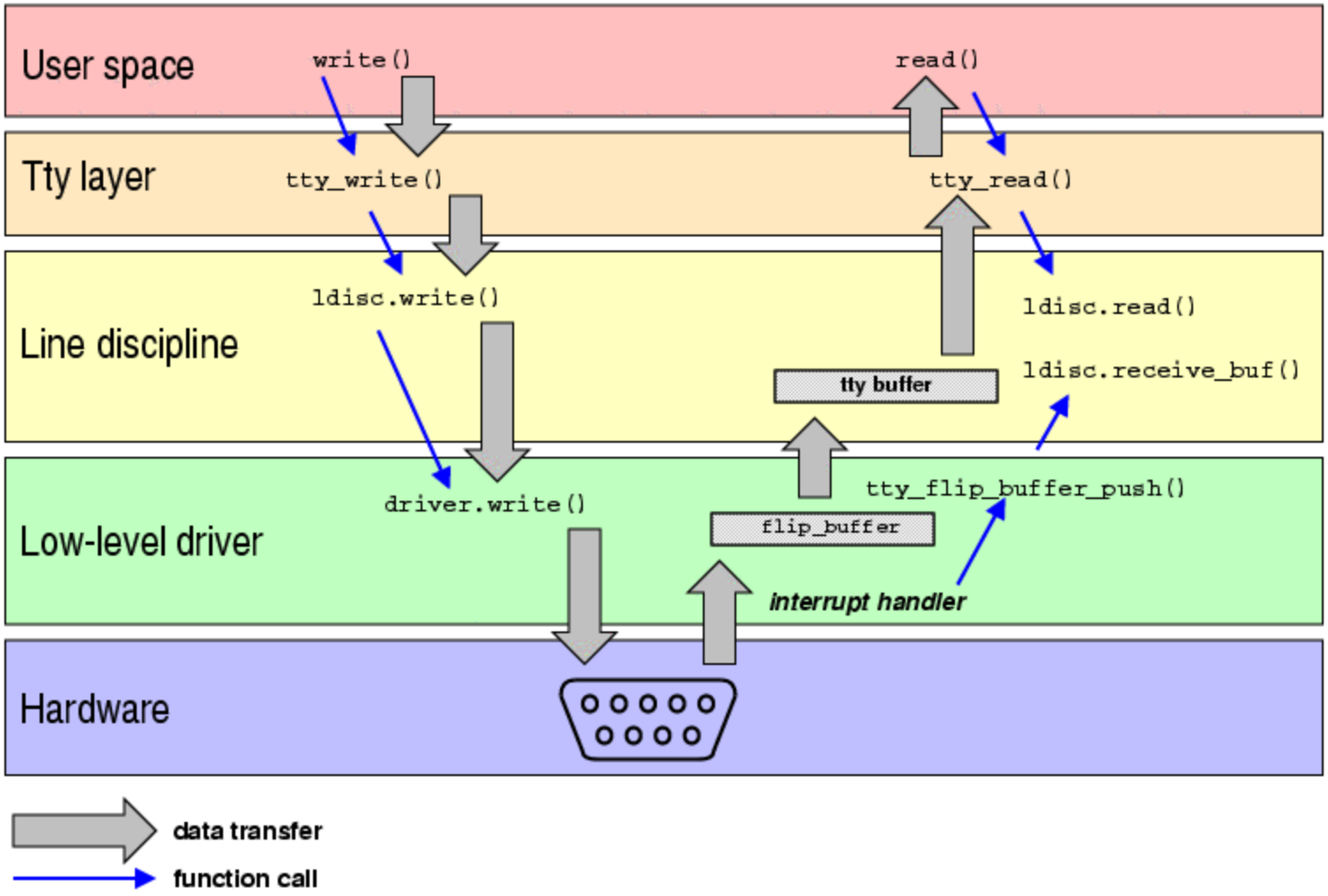
Serial layers and their mapping to Linux kernel



Linux tree for tty devices

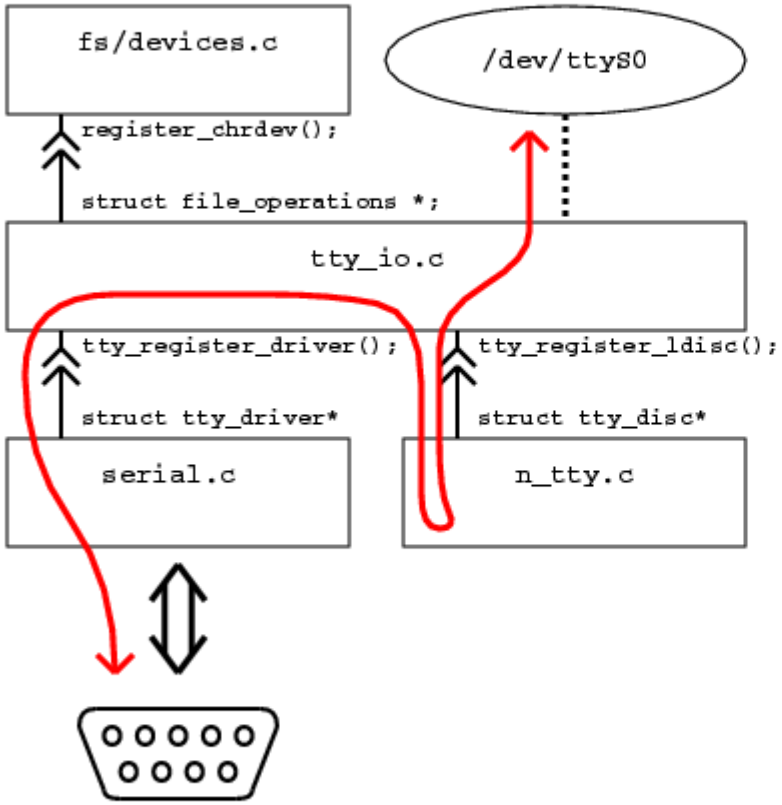
```
/sys/class/tty/
|-- console
|   `-- dev
|-- ptmx
|   `-- dev
|-- tty
|   `-- dev
|-- tty0
|   `-- dev
   ...
|-- ttyS1
|   `-- dev
|-- ttyS2
|   `-- dev
|-- ttyS3
|   `-- dev
   ...
|-- ttyUSB0
|   |-- dev
|   |-- device -> ../../../devices/pci0000:00/0000:00:09.0/usb3/3-1/3-1:1.0/ttyUSB0
|   `-- driver -> ../../../bus/usb-serial/drivers/keyspan_4
|-- ttyUSB1
|   |-- dev
|   |-- device -> ../../../devices/pci0000:00/0000:00:09.0/usb3/3-1/3-1:1.0/ttyUSB1
|   `-- driver -> ../../../bus/usb-serial/drivers/keyspan_4
|-- ttyUSB2
|   |-- dev
|   |-- device -> ../../../devices/pci0000:00/0000:00:09.0/usb3/3-1/3-1:1.0/ttyUSB2
|   `-- driver -> ../../../bus/usb-serial/drivers/keyspan_4
`-- ttyUSB3
    |-- dev
    |-- device -> ../../../devices/pci0000:00/0000:00:09.0/usb3/3-1/3-1:1.0/ttyUSB3
    `-- driver -> ../../../bus/usb-serial/drivers/keyspan_4
```

Data flow and function calls in writing and reading



http://www.linux.it/~rubini/docs/serial/serial.html

Source files involved in serial management, how they are connected and how data flows.



http://www.linux.it/~rubini/docs/serial/serial.html

UART driver

- data structures: struct uart_driver

```
struct uart_driver {
  struct module  *owner;           /* Module that owns this
                                      struct */
  const char     *driver_name;     /* Name */
  const char     *dev_name;        /* /dev node name
                                      such as ttyS */
  /* ... */
  int             major;           /* Major number */
  int             minor;           /* Minor number */
  /* ... */
  struct tty_driver *tty_driver;   /* tty driver */
};
```

struct uart_port

```
struct uart_port {
  spinlock_t     lock;              /* port lock */
  unsigned int   iobase;            /* in/out[bwl]*/
  unsigned char  __iomem *membase; /* read/write[bwl]*/
  unsigned int   irq;               /* irq number */
  unsigned int   uartclk;           /* base uart clock */
  unsigned char  fifosize;          /* tx fifo size */
  unsigned char  x_char;            /* xon/xoff flow
                                       control */
  /* ... */
};
```

struct uart_ops

```
struct uart_ops {
  uint (*tx_empty)(struct uart_port *);      /* Is TX FIFO empty? */
  void (*set_mctrl)(struct uart_port *,
                    unsigned int mctrl);     /* Set modem control params */
  uint (*get_mctrl)(struct uart_port *);     /* Get modem control params */
  void (*stop_tx)(struct uart_port *);       /* Stop xmission */
  void (*start_tx)(struct uart_port *);      /* Start xmission */
  /* ... */
  void (*shutdown)(struct uart_port *);      /* Disable the port */
  void (*set_termios)(struct uart_port *,
                      struct termios *new,
                      struct termios *old); /* Set terminal interface
                                               params */
  /* ... */
  void (*config_port)(struct uart_port *,
                      int);                  /* Configure UART port */
  /* ... */
};
```

struct console

```
static struct console serial8250_console = {
2858        .name           = "ttyS",
2859        .write          = serial8250_console_write,
2860        .device         = uart_console_device,
2861        .setup          = serial8250_console_setup,
2862        .early_setup    = serial8250_console_early_setup,
2863        .flags          = CON_PRINTBUFFER,
2864        .index          = -1,
2865        .data           = &serial8250_reg,
2866};
```

UART core initialization

```
int uart_register_driver(struct uart_driver *drv)
2312{
2326        normal = alloc_tty_driver(drv->nr);
2327        if (!normal)
2328                goto out_kfree;
2330        drv->tty_driver = normal;
2344        tty_set_operations(normal, &uart_ops);
2361        retval = tty_register_driver(normal);
2362        if (retval >= 0)
2363                return retval;
2364
2365        put_tty_driver(normal);
2366out_kfree:
2367        kfree(drv->state);
2368out:
2369        return -ENOMEM;
2370}
```

UART initialization

```
static int __init serial8250_init(void)
3216{
3230        ret = uart_register_driver(&serial8250_reg);
3235        serial8250_isa_devs = platform_device_alloc("serial8250",
3236                                                    PLAT8250_DEV_LEGACY);
3242        ret = platform_device_add(serial8250_isa_devs);
3246        serial8250_register_ports(&serial8250_reg, &serial8250_isa_devs->dev);
3248        ret = platform_driver_register(&serial8250_isa_driver);
3263}
```

UART exit

```
static void __exit serial8250_exit(void)
3266{
3267        struct platform_device *isa_dev = serial8250_isa_devs;
3268
3276        platform_driver_unregister(&serial8250_isa_driver);
3277        platform_device_unregister(isa_dev);
3278
3282        uart_unregister_driver(&serial8250_reg);
3284}
3285
```

UART startup

```
static int serial8250_startup(struct uart_port *port)
1949{
1957
1958        if (up->port.iotype != up->cur_iotype)
1959                set_io_from_upio(port);
1960
1961        if (up->port.type == PORT_16C950) {
1962                /* Wake up and initialize UART */
1972        }
1986        serial8250_clear_fifos(up);
2089        /*
2090         * Now, initialize the UART
2091         */
2092        serial_outp(up, UART_LCR, UART_LCR_WLEN8);
2094        spin_lock_irqsave(&up->port.lock, flags);
2105        serial8250_set_mctrl(&up->port, up->port.mctrl);
return 0;
2174}
```

UART start_tx

```
static void serial8250_start_tx(struct uart_port *port)
1343{
1344        struct uart_8250_port *up = (struct uart_8250_port *)port;
1345
1346        if (!(up->ier & UART_IER_THRI)) {
1347                up->ier |= UART_IER_THRI;
1348                serial_out(up, UART_IER, up->ier);
1349
1350                if (up->bugs & UART_BUG_TXEN) {
1351                        unsigned char lsr;
1352                        lsr = serial_in(up, UART_LSR);
1353                        up->lsr_saved_flags |= lsr & LSR_SAVE_FLAGS;
1354                        if ((up->port.type == PORT_RM9000) ?
1355                                (lsr & UART_LSR_THRE) :
1356                                (lsr & UART_LSR_TEMT))
1357                                transmit_chars(up);
1358} }
13591361        /*
1362         * Re-enable the transmitter if we disabled it.
1363         */
1364        if (up->port.type == PORT_16C950 && up->acr & UART_ACR_TXDIS) {
1365                up->acr &= ~UART_ACR_TXDIS;
1366                serial_icr_write(up, UART_ACR, up->acr);
1367        } }
```

TTY driver example

- tty_struct

```
struct tty_struct {
    int magic;                          /* Magic marker */
    struct tty_driver *driver;          /* Pointer to the tty
                                           driver */
    struct tty_ldisc ldisc;             /* Attached Line
                                           discipline */
    /* ... */
    struct tty_flip_buffer flip;        /* Flip Buffer. See
                                           below. */
    /* ... */
    wait_queue_head_t write_wait;       /* See the section
                                           "Line Disciplines" */
    wait_queue_head_t read_wait;        /* See the section
                                           "Line Disciplines" */
    /* ... */
  };
```

tty_flip_buffer

```
struct tty_flip_buffer {
    /* ... */
    struct semaphore pty_sem;      /* Serialize */
    char *char_buf_ptr;            /* Pointer to the flip
                                      buffer */
    /* ... */
    unsigned char char_buf[2*TTY_FLIPBUF_SIZE]; /* The flip
                                                   buffer */
    /* ... */
  };
```

```
struct tty_bufhead {
  /* ... */
  struct semaphore pty_sem;              /* Serialize */
  struct tty_buffer *head, tail, free; /* See below */
  /* ... */
};
struct tty_buffer {
  struct tty_buffer *next;
  char *char_buf_ptr;        /* Pointer to the flip buffer */
  /* ... */
  unsigned long data[0];     /* The flip buffer, memory for
                                which is dynamically
                                allocated */
};
```

tty_driver

```
struct tty_driver {
  int magic;              /* Magic number */
  /* ... */
  int major;              /* Major number */
  int minor_start;        /* Start of minor number */
  /* ... */
  /* Interface routines between a tty driver and higher
     layers */
  int  (*open)(struct tty_struct *tty, struct file *filp);
  void (*close)(struct tty_struct *tty, struct file *filp);
  int  (*write)(struct tty_struct *tty,
                const unsigned char *buf, int count);
  void (*put_char)(struct tty_struct *tty,
                   unsigned char ch);
  /* ... */
};
```

tty_ops

```
448static const struct file_operations tty_fops = {
 449        .llseek         = no_llseek,
 450        .read           = tty_read,
 451        .write          = tty_write,
 452        .poll           = tty_poll,
 453        .unlocked_ioctl = tty_ioctl,
 454        .compat_ioctl   = tty_compat_ioctl,
 455        .open           = tty_open,
 456        .release        = tty_release,
 457        .fasync         = tty_fasync,
 458};
```

tty initialization

```
int __init tty_init(void)
3178{
3179        cdev_init(&tty_cdev, &tty_fops);
3180        if (cdev_add(&tty_cdev, MKDEV(TTYAUX_MAJOR, 0), 1) ||
3181            register_chrdev_region(MKDEV(TTYAUX_MAJOR, 0), 1, "/dev/tty") <
0)
3182                panic("Couldn't register /dev/tty driver\n");
3183        device_create(tty_class, NULL, MKDEV(TTYAUX_MAJOR, 0), NULL,
3184                              "tty");
3185
3186        cdev_init(&console_cdev, &console_fops);
3187        if (cdev_add(&console_cdev, MKDEV(TTYAUX_MAJOR, 1), 1) ||
3188            register_chrdev_region(MKDEV(TTYAUX_MAJOR, 1), 1,
"/dev/console") < 0)
3189                panic("Couldn't register /dev/console driver\n");
3190        device_create(tty_class, NULL, MKDEV(TTYAUX_MAJOR, 1), NULL,
3191                              "console");
3192
3193#ifdef CONFIG_VT
3194        vty_init(&console_fops);
3195#endif
3196        return 0;
3197}
```

tty open

```
static int tty_open(struct inode *inode, struct file *filp)
1788{
1794        unsigned saved_flags = filp->f_flags;
1796        nonseekable_open(inode, filp);
1806        if (device == MKDEV(TTYAUX_MAJOR, 0)) {
1807                tty = get_current_tty();
1808                if (!tty) {
1809                        tty_unlock();
1810                        mutex_unlock(&tty_mutex);
1811                        return -ENXIO;
1812                }
1813                driver = tty_driver_kref_get(tty->driver);
1814                index = tty->index;
1815                filp->f_flags |= O_NONBLOCK; /* Don't let /dev/tty block */
1820        }
1830        if (device == MKDEV(TTYAUX_MAJOR, 1)) {
1831                struct tty_driver *console_driver = console_device(&index);
1832                if (console_driver) {
1833                        driver = tty_driver_kref_get(console_driver);
1844        }
1845
1846        driver = get_tty_driver(device, &index);
1847        if (!driver) {
```

tty open (Cont.)

```
1848                tty_unlock();
1849                mutex_unlock(&tty_mutex);
1850                return -ENODEV;
1851        }
1864        if (tty) {
1865                retval = tty_reopen(tty);
1866                if (retval)
1867                        tty = ERR_PTR(retval);
1868        } else
1869                tty = tty_init_dev(driver, index, 0);
1872        tty_driver_kref_put(driver);
1873        if (IS_ERR(tty)) {
1874                tty_unlock();
1875                return PTR_ERR(tty);
1876        }
1878        tty_add_file(tty, filp);
1880        check_tty_count(tty, "tty_open");
1881        if (tty->driver->type == TTY_DRIVER_TYPE_PTY &&
1882            tty->driver->subtype == PTY_TYPE_MASTER)
1883                noctty = 1;
1937}
```

tty_read

```
static ssize_t tty_read(struct file *file, char __user *buf, size_t count,
 929                            loff_t *ppos)
 930{
 931        int i;
 932        struct inode *inode = file->f_path.dentry->d_inode;
 933        struct tty_struct *tty = file_tty(file);
 934        struct tty_ldisc *ld;
 935
 936        if (tty_paranoia_check(tty, inode, "tty_read"))
 937                return -EIO;
 938        if (!tty || (test_bit(TTY_IO_ERROR, &tty->flags)))
 939                return -EIO;
 940
 941        /* We want to wait for the line discipline to sort out in this
 942           situation */
 943        ld = tty_ldisc_ref_wait(tty);
 944        if (ld->ops->read)
 945                i = (ld->ops->read)(tty, file, buf, count);
 946        else
 947                i = -EIO;
 948        tty_ldisc_deref(ld);
 949        if (i > 0)
 950                inode->i_atime = current_fs_time(inode->i_sb);
 951        return i;
 952}
```

User Application

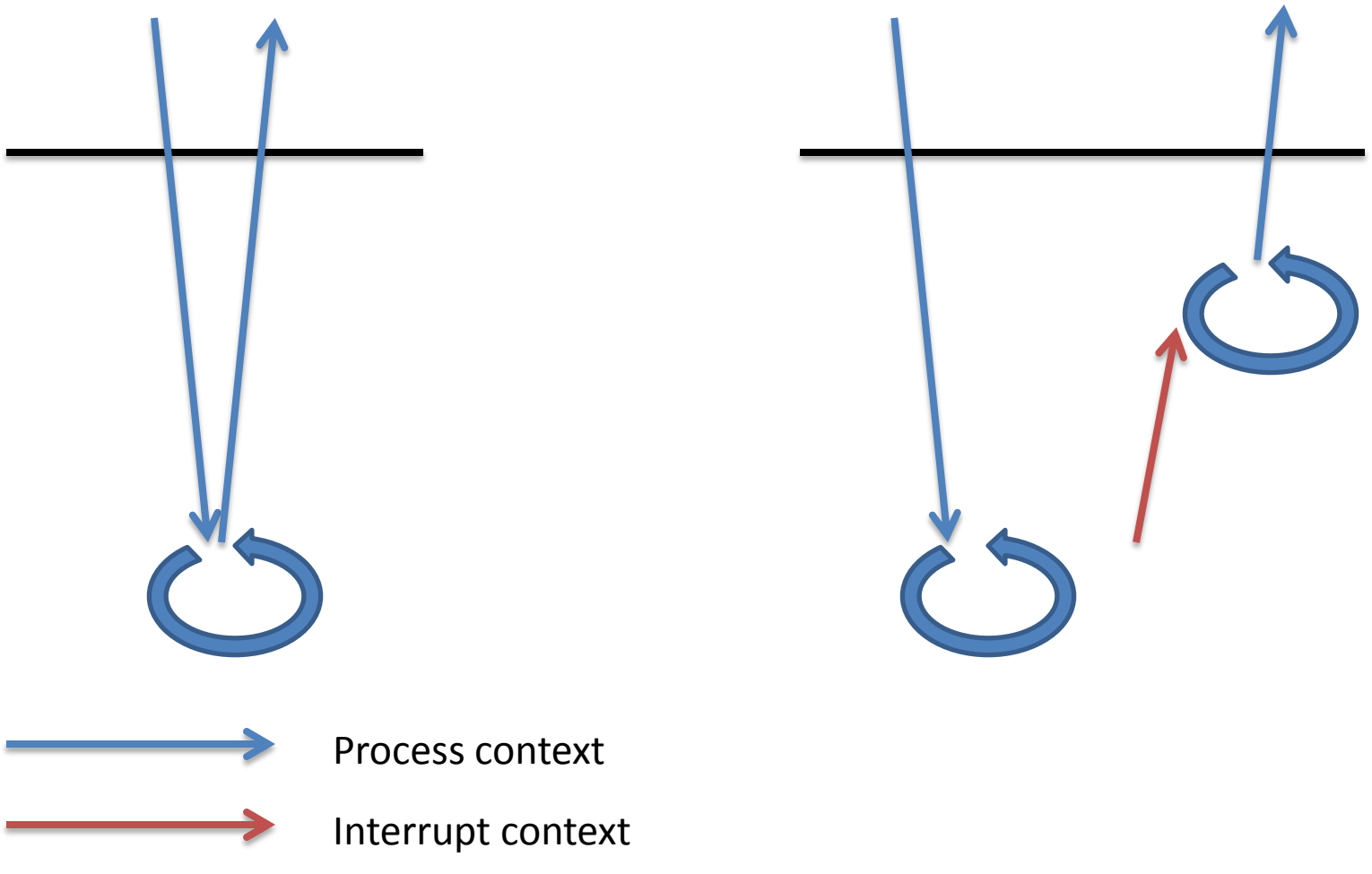
- A Linux serial test program
- send output typed on the computer keyboard after the program is started, through the serial port specified

Top Half and Bottom Half

Comparisons of Different Bottom Half Implementation

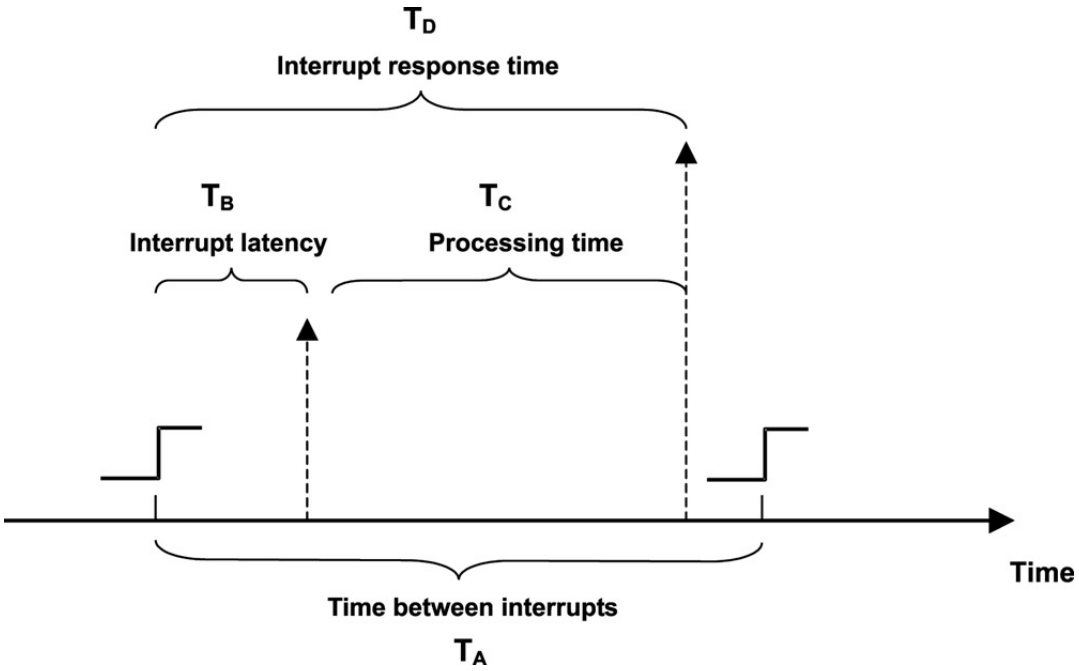
- why we need top half and bottom half
- softirq
- tasklet
- workqueue
- choice of different bottom half implementations

Process Context/Interrupt Context



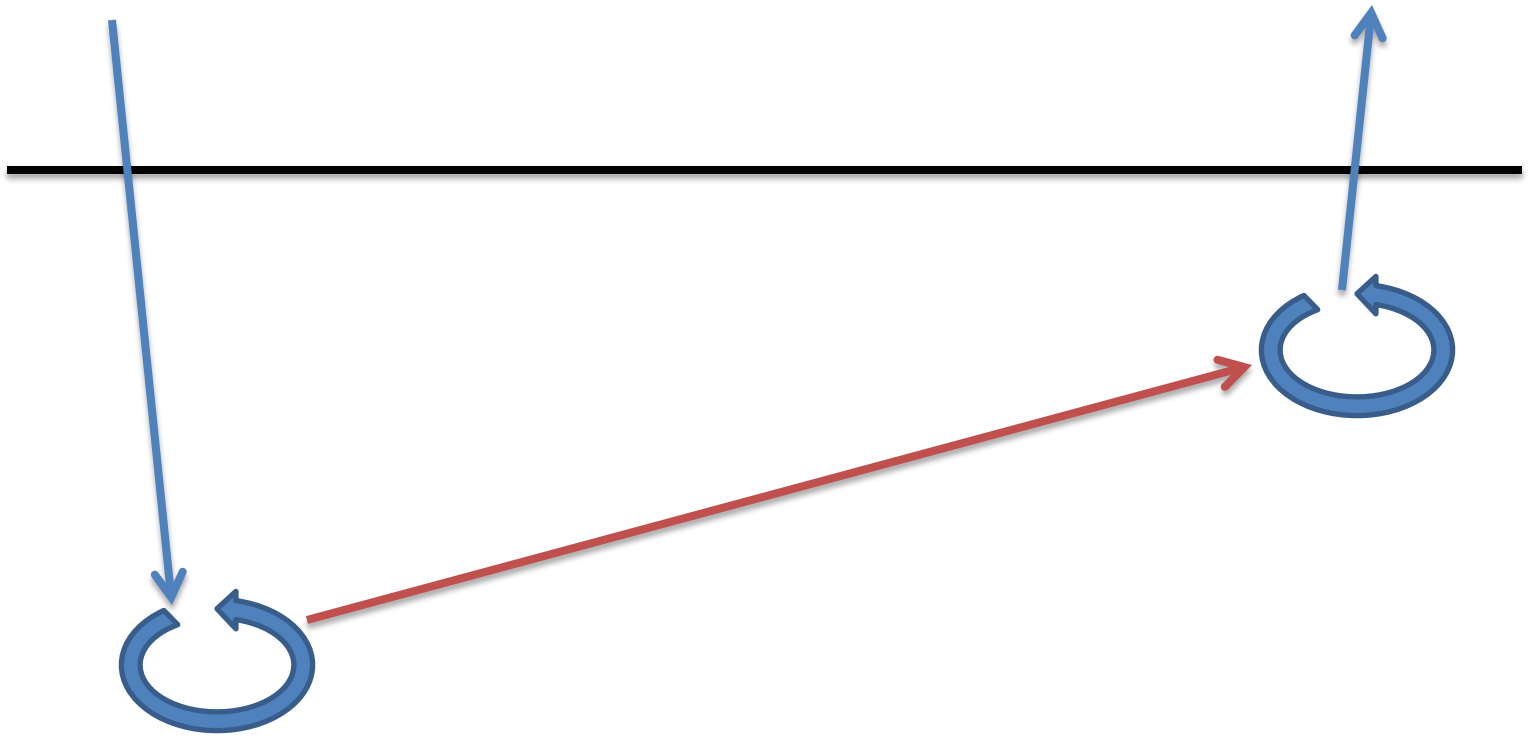
Exceptions and interrupts (Cont.)

- Exception timing



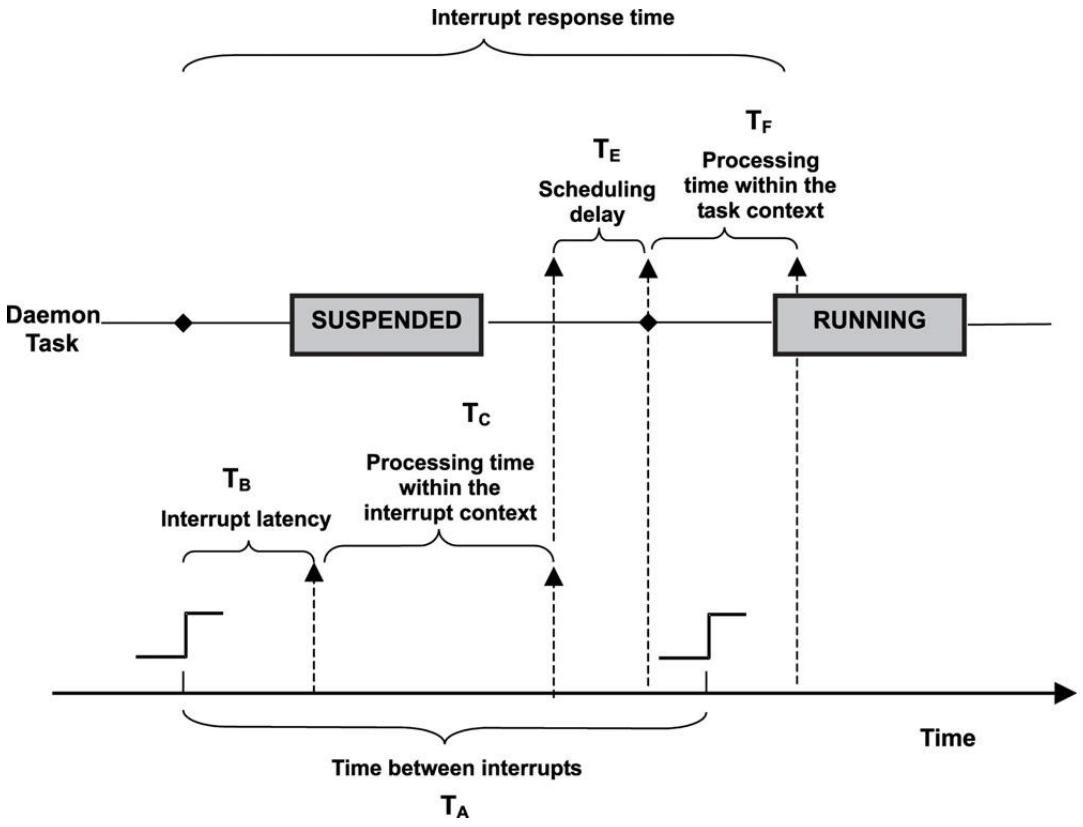
Source: Qing Li "real-time concepts for embedded s

Process Context/Interrupt Context



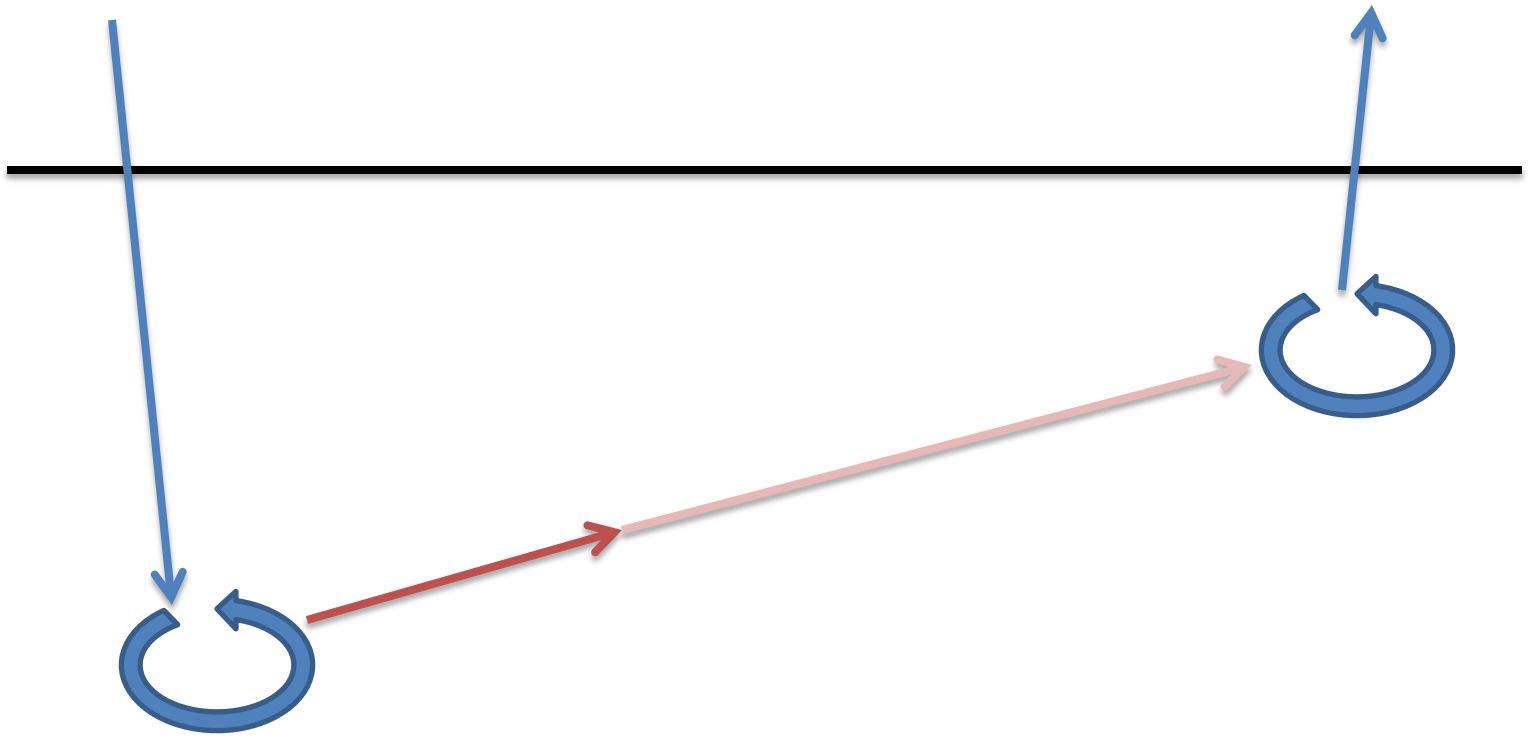
Exceptions and interrupts (Cont.)

- Exception timing (Cont.)

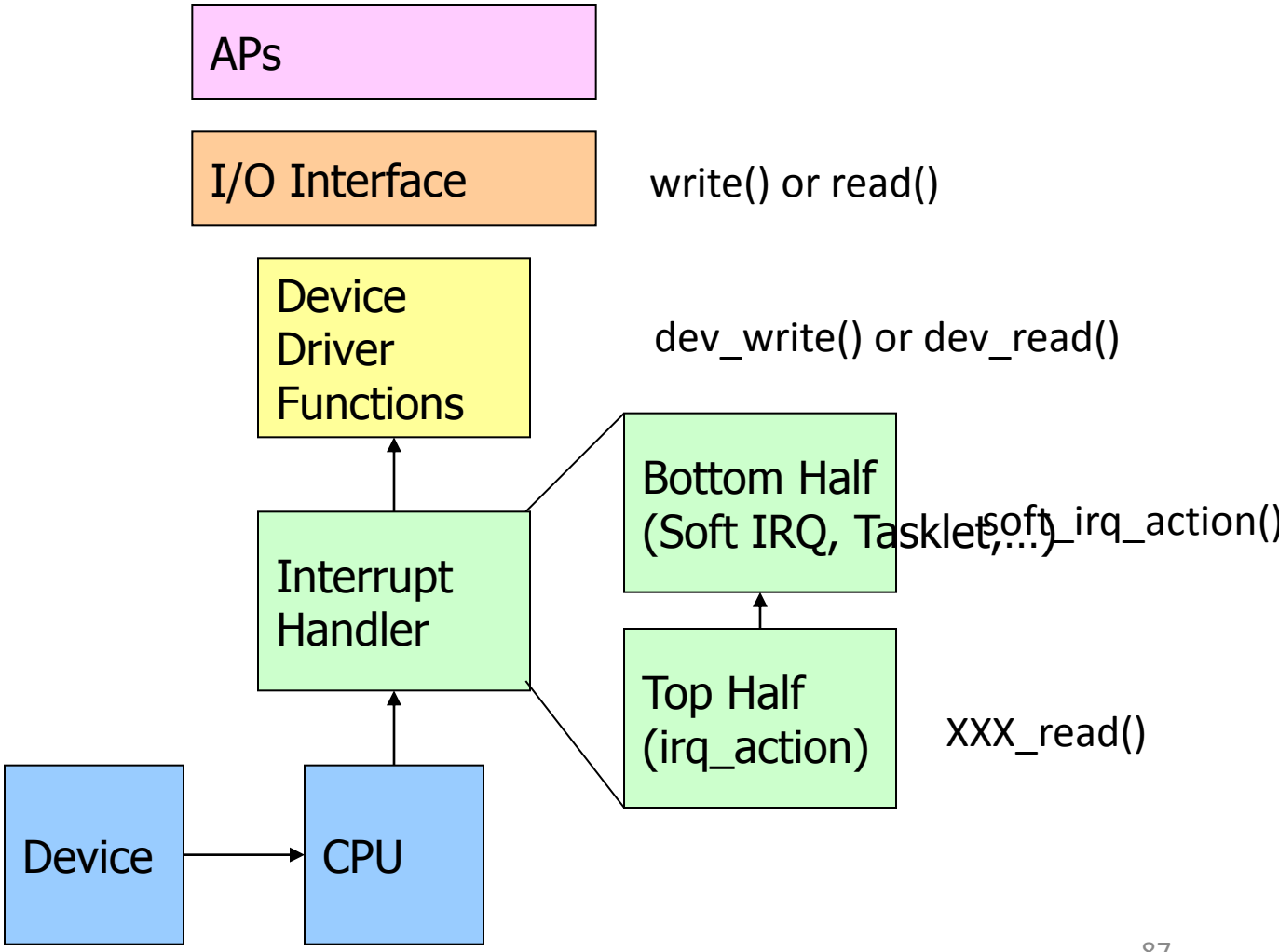


Source: Qing Li "real-time concepts for embedded s

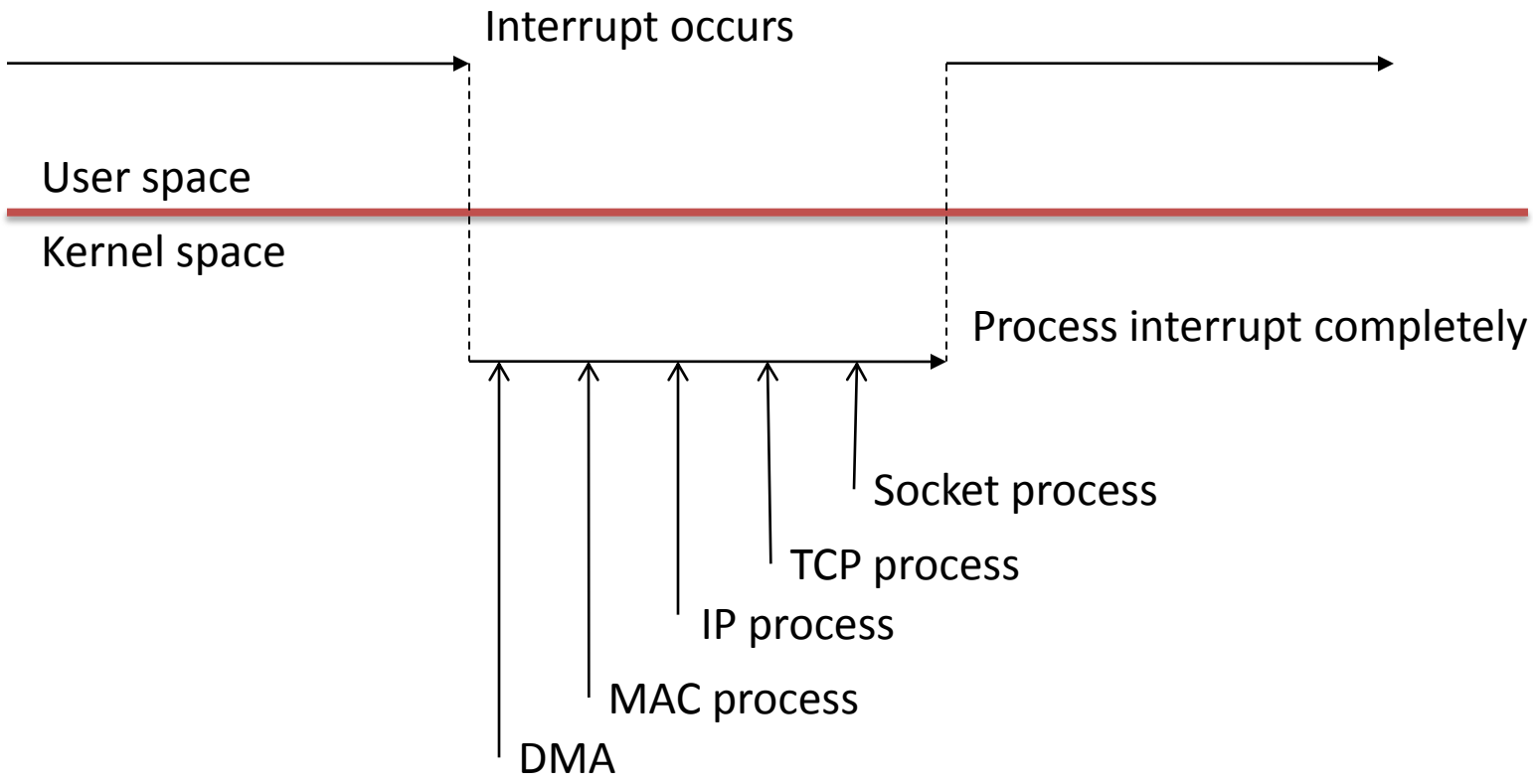
Process Context/Interrupt Context



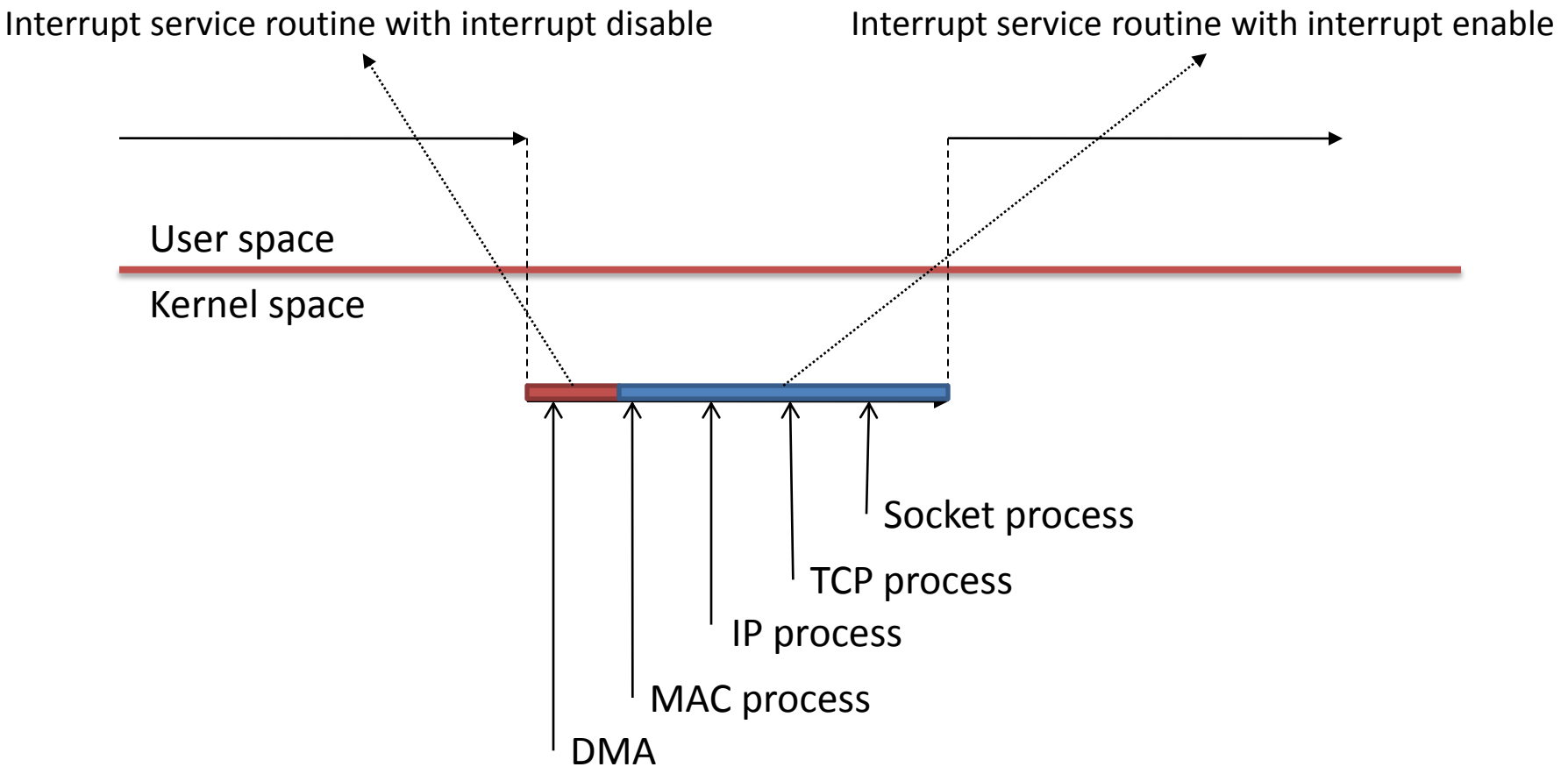
Interrupt, Interrupt Handler, and Device Driver



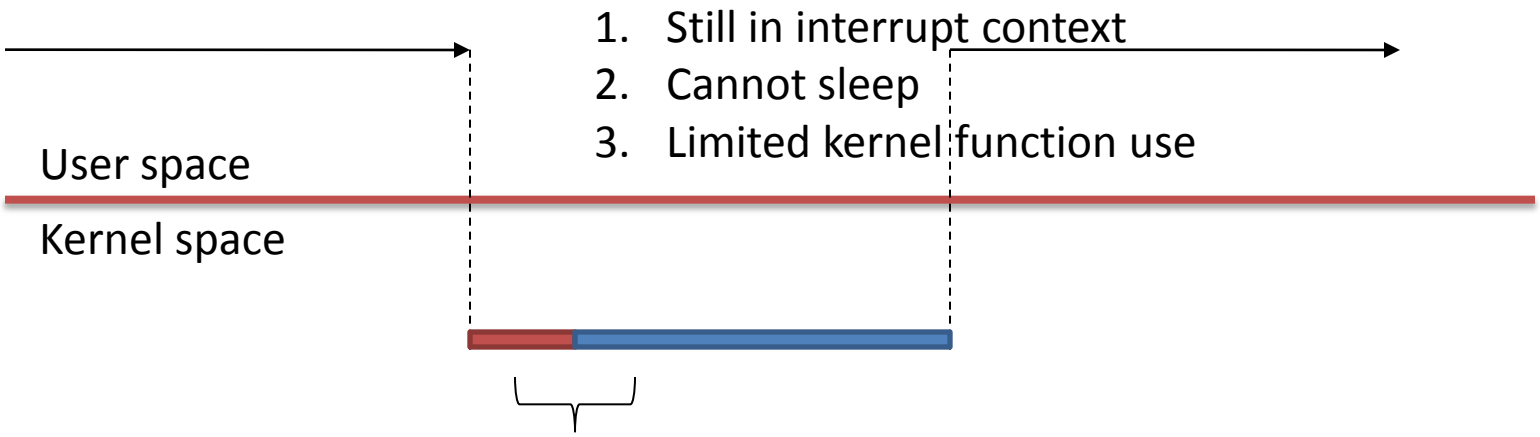
Why we need top half and bottom half



Two halves approaches



How to implement it



1. Check if bottom half is necessary
2. Call bottom half function
3. Enable interrupt

Top Halves vs. Bottom Halves

- Top halves
  - interrupt handlers (top halves), are executed by the kernel asynchronously in immediate response to a hardware interrupt
  - ASAP
- Bottom halves
  - to perform any interrupt-related work not performed by the interrupt handler itself
  - Process all interrupt related functions

Linux Implementation

- Top half: implemented entirely via the interrupt handler
- Bottom half
  - multiple implementation
  - mechanisms are different interfaces and subsystems

Bottom Half	Status
BH	Removed in 2.5
Task queues	Removed in 2.5
Softirq	Available since 2.3
Tasklet	Available since 2.3
Work queues	Available since 2.5

Linux Implementation

- BH
  - The top half could mark whether the bottom half would run by setting a bit in a 32-bit integer
  - Each BH was globally synchronized
  - No two could run at the same time, even on different processors
  - This was easy to use, yet inflexible; simple, yet a bottleneck
  - Remove since 2.5
- Task queues
  - defined a family of queues
  - Each queue contained a linked list of functions to call
  - The queued functions were run at certain times
  - Drivers could register their bottom halves in the appropriate queue
  - It also was not lightweight enough for performance-critical subsystems, such as networking
  - Remove since 2.5

Linux Implementation

- Softirq
  - Since 2.3 development series
  - Softirqs are a set of 32 statically defined bottom halves that can run simultaneously on any processor
  - Softirqs are useful when performance is critical, such as with networking.
  - Two of the same softirq can run at the same time
  - Softirqs must be registered statically at compile-time
- Tasklets
  - Since 2.3 development series
  - dynamically created bottom halves that are built on top of softirqs.
  - Two different tasklets can run concurrently on different processors, but two of the same type of tasklet cannot run simultaneously
  - Tasklets are a good tradeoff between performance and ease of use
  - For most bottom-half processing, the tasklet is sufficient

Linux Implementation

```
/*
 * structure representing a single softirq entry
 */
struct softirq_action {
        void (*action)(struct softirq_action *); /* function to run */
        void *data;                              /* data to pass to function */
};
```

```
u32 pending = softirq_pending(cpu);

if (pending) {
    struct softirq_action *h = softirq_vec;

    softirq_pending(cpu) = 0;

    do {
        if (pending & 1)
            h->action(h);
        h++;
        pending >>= 1;
    } while (pending);
}
```

```
raise_softirq(NET_TX_SOFTIRQ);
```

Tasklet	Priority	Softirq Description
HI_SOFTIRQ	0	High-priority tasklets
TIMER_SOFTIRQ	1	Timer bottom half
NET_TX_SOFTIRQ	2	Send network packets
NET_RX_SOFTIRQ	3	Receive network packets
SCSI_SOFTIRQ	4	SCSI bottom half
TASKLET_SOFTIRQ	5	Tasklets

ksoftirqd

```
for (;;) {
        if (!softirq_pending(cpu))
                schedule();

        set_current_state(TASK_RUNNING);

        while (softirq_pending(cpu)) {
                do_softirq();
                if (need_resched())
                    schedule();
        }

        set_current_state(TASK_INTERRUPTIBLE);
}
```

softirq

- rarely used (normally for interrupts occurs frequently and you really want to fast in SMP)
- statically allocated at compile-time (register and destroy softirqs)

```
static struct softirq_action softirq_vec[32];
```

```
/*
 * structure representing a single softirq entry
 */
struct softirq_action {
        void (*action)(struct softirq_action *); /* function to run */
        void *data;                              /* data to pass to function */
};
```

```
void softirq_handler(struct softirq_action *)
```

softirq

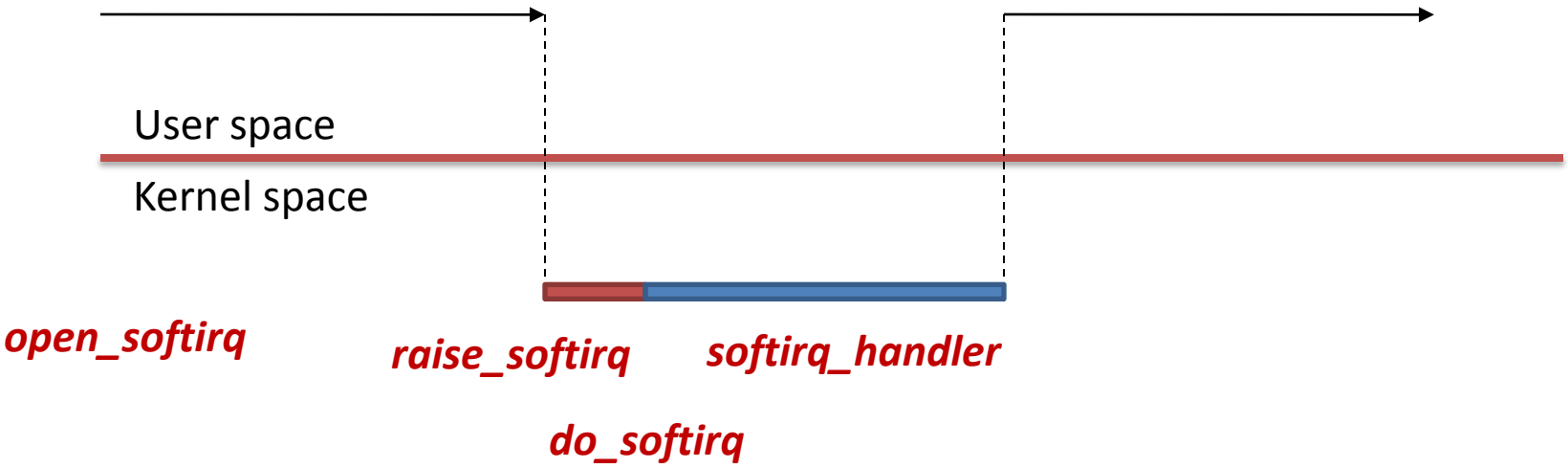
- Execution
  - softirq handler is registered at run-time via open_softirq()
  - interrupt handler marks its softirq for execution before returning (raise_softirq())
  - pending softirqs are checked for and executed in (do_softirq())
    - In the return from hardware interrupt code
    - In the ksoftirqd kernel thread
    - In any code that explicitly checks for and executes pending softirqs

softirq

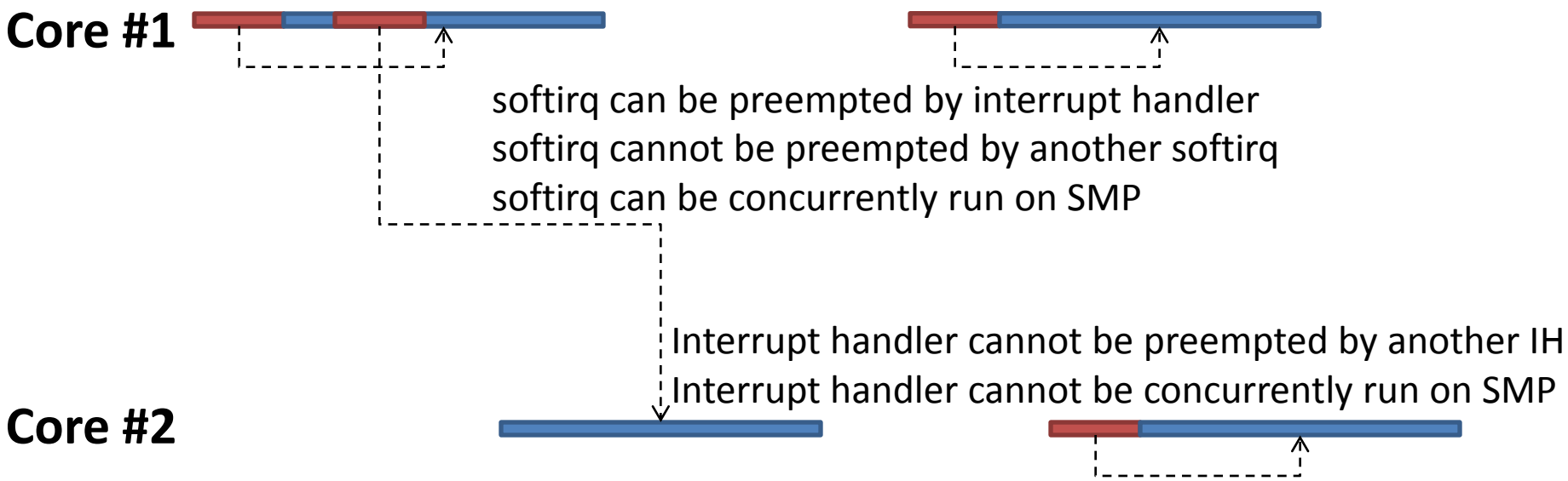
- The softirq handlers run with interrupts enabled and cannot sleep
- While a handler runs, softirqs on the current processor are disabled
- Another processor, however, can execute other softirqs
- Any shared data even global data used only within the softirq handler itself needs proper locking
- ***If a softirq obtained a lock to prevent another instance of itself from running simultaneously, there would be no reason to use a softirq***

softirq

- most softirq handlers resort to per-processor data and provide excellent scalability
- If you do not need to scale to infinitely many processors, then use a tasklet



softirq


Core #1
softirq can be preempted by interrupt handler
softirq cannot be preempted by another softirq
softirq can be concurrently run on SMP
Interrupt handler cannot be preempted by another IH
Interrupt handler cannot be concurrently run on SMP
Core #2

softirq priority

Tasklet	Priority	Softirq Description
HI_SOFTIRQ	0	High-priority tasklets
TIMER_SOFTIRQ	1	Timer bottom half
NET_TX_SOFTIRQ	2	Send network packets
NET_RX_SOFTIRQ	3	Receive network packets
SCSI_SOFTIRQ	4	SCSI bottom half
TASKLET_SOFTIRQ	5	Tasklets

ksoftirqd

- Why
  - Starve user process
  - Starve softirq
- per-processor kernel threads
- Help when the system is overwhelmed with softirqs
- not immediately process reactivated softirqs
- if the number of softirqs grows excessive, the kernel wakes up a family of kernel threads to handle the load
- The kernel threads run with the lowest possible priority (nice value of 19)

ksoftirqd

```
for (;;) {
        if (!softirq_pending(cpu))
                schedule();

        set_current_state(TASK_RUNNING);

        while (softirq_pending(cpu)) {
                do_softirq();
                if (need_resched())
                    schedule();
        }

        set_current_state(TASK_INTERRUPTIBLE);
}
```

tasklet

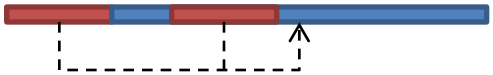
- built on top of softirqs (they are softirq)
- Execution
  - Register

```
struct tasklet_struct {
        struct tasklet_struct *next;  /* next tasklet in the list */
        unsigned long state;          /* state of the tasklet */
        atomic_t count;               /* reference counter */
        void (*func)(unsigned long);  /* tasklet handler function */
        unsigned long data;           /* argument to the tasklet function */
};
```

  - initialize a tasklet
    - tasklet_init
  - Scheduled tasklets (the equivalent of raised softirqs)
    - tasklet_schedule() & tasklet_hi_schedule()
  - do_softirq() and tasklet_handler()

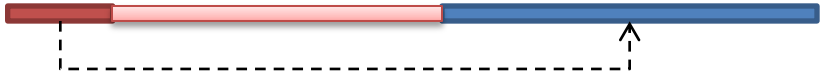
tasklet

Core #1



tasklet can be preempted by interrupt handler
tasklet cannot be preempted by the tasklet/softirq
the same tasklet cannot be concurrently run on SMP
different tasklet can be concurrently run on SMP

Core #2



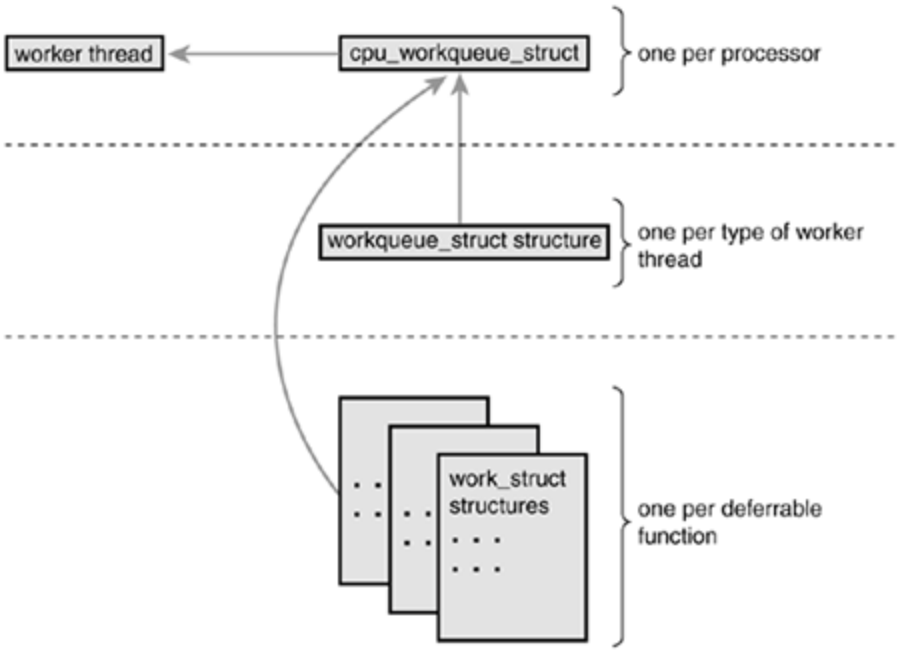
Tasklet

```
struct tasklet_struct {
        struct tasklet_struct *next;  /* next tasklet in the list */
        unsigned long state;          /* state of the tasklet */
        atomic_t count;               /* reference counter */
        void (*func)(unsigned long);  /* tasklet handler function */
        unsigned long data;           /* argument to the tasklet function */
};
```

Tasklet

Tasklet	Priority	Softirq Description
HI_SOFTIRQ	0	High-priority tasklets
TIMER_SOFTIRQ	1	Timer bottom half
NET_TX_SOFTIRQ	2	Send network packets
NET_RX_SOFTIRQ	3	Receive network packets
SCSI_SOFTIRQ	4	SCSI bottom half
TASKLET_SOFTIRQ	5	Tasklets

workqueue


worker thread
cpu_workqueue_struct
one per processor
workqueue_struct structure
one per type of worker thread
work_struct structures
one per deferrable function

workqueue

```
/*
 * The externally visible workqueue abstraction is an array of
 * per-CPU workqueues:
 */
struct workqueue_struct {
        struct cpu_workqueue_struct cpu_wq[NR_CPUS];
        const char *name;
        struct list_head list;
};
```

```
struct cpu_workqueue_struct {
        spinlock_t lock;              /* lock protecting this structure */

        long remove_sequence;         /* least-recently added (next to run) */
        long insert_sequence;         /* next to add */
        struct list_head worklist;    /* list of work */
        wait_queue_head_t more_work;
        wait_queue_head_t work_done;

        struct workqueue_struct *wq; /* associated workqueue_struct */
        task_t *thread;               /* associated thread */

        int run_depth;                /* run_workqueue() recursion depth */
};
```

workqueue structure

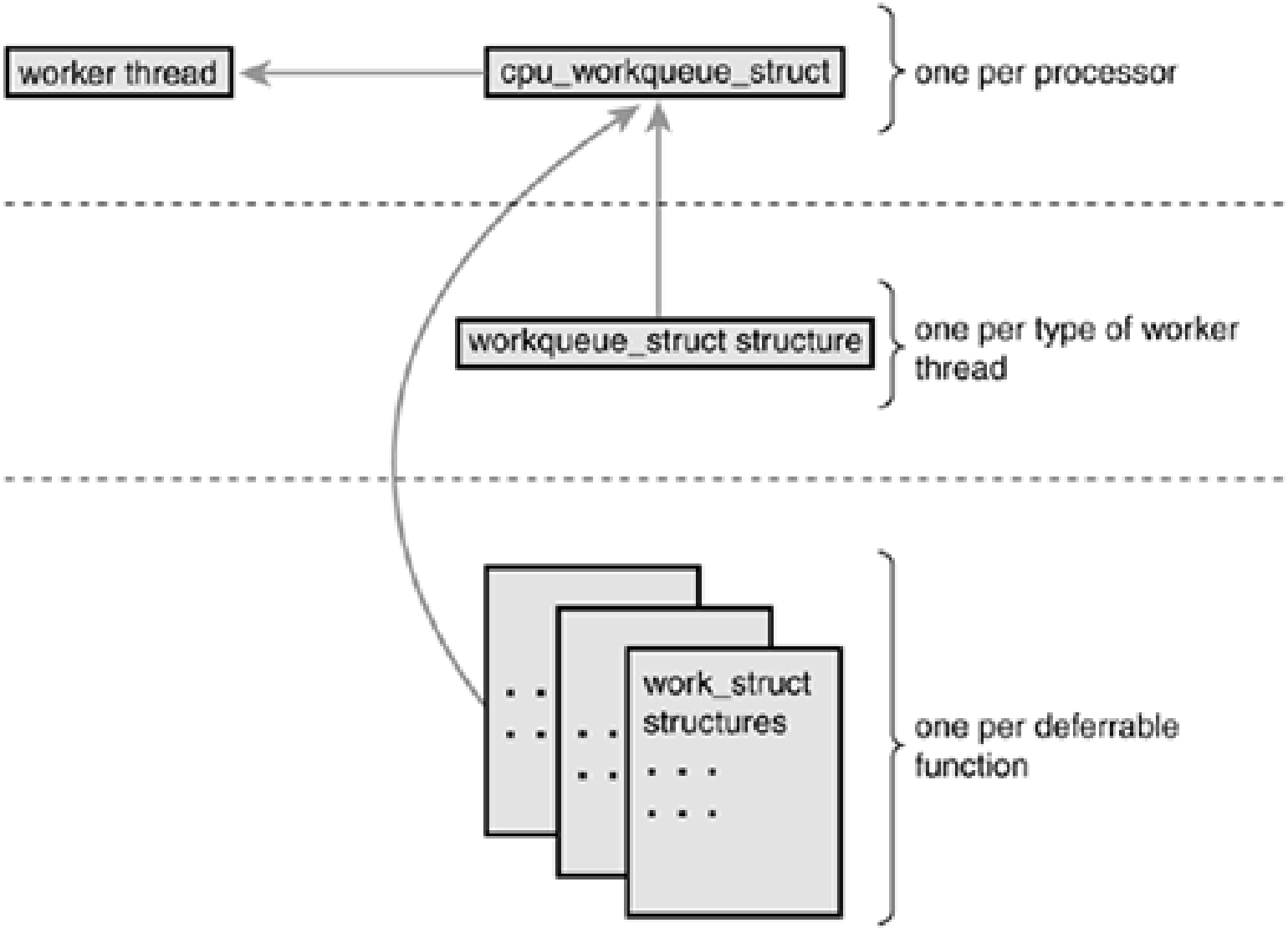
```
struct work_struct {
        unsigned long pending;      /* is this work pending? */
        struct list_head entry;     /* link list of all work */
        void (*func)(void *);       /* handler function */
        void *data;                 /* argument to handler */
        void *wq_data;              /* used internally */
        struct timer_list timer;    /* timer used by delayed work queues */
};
```

```
        for (;;) {
                set_task_state(current, TASK_INTERRUPTIBLE);
                add_wait_queue(&cwq->more_work, &wait);

                if (list_empty(&cwq->worklist))
                        schedule();
                else
                        set_task_state(current, TASK_RUNNING);
                remove_wait_queue(&cwq->more_work, &wait);

                if (!list_empty(&cwq->worklist))
                        run_workqueue(cwq);
        }
```

run_workqueue

```
while (!list_empty(&cwq->worklist)) {
        struct work_struct *work;
        void (*f)(void *);
        void *data;

        work = list_entry(cwq->worklist.next, struct work_struct, entry);
        f = work->func;
        data = work->data;

        list_del_init(cwq->worklist.next);

        clear_bit(0, &work->pending);
        f(data);
}
```

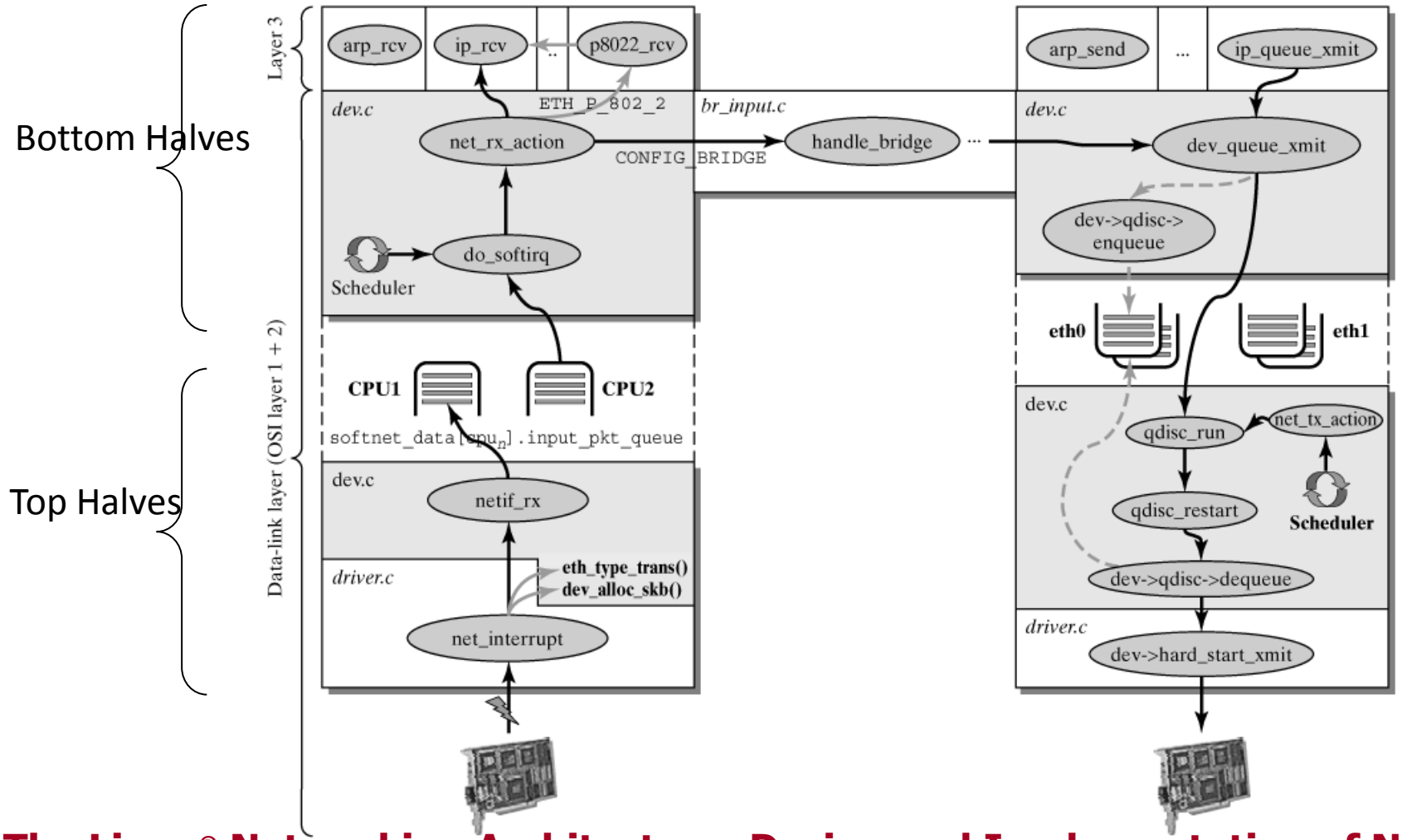
The relationship between work, work queues, and the worker threads



choice of different bottom half implementations

- Softirqs
  - Concurrency
  - Protect shared data
  - networking subsystem
  - timing-critical and high-frequency uses
  - Good for SMP
- Tasklets
  - if the code is not finely threaded
  - Do not run concurrently
  - Shared with all tasklets
- Workqueue
  - run in process context
  - Can sleep

Top Halves vs. Bottom Halves



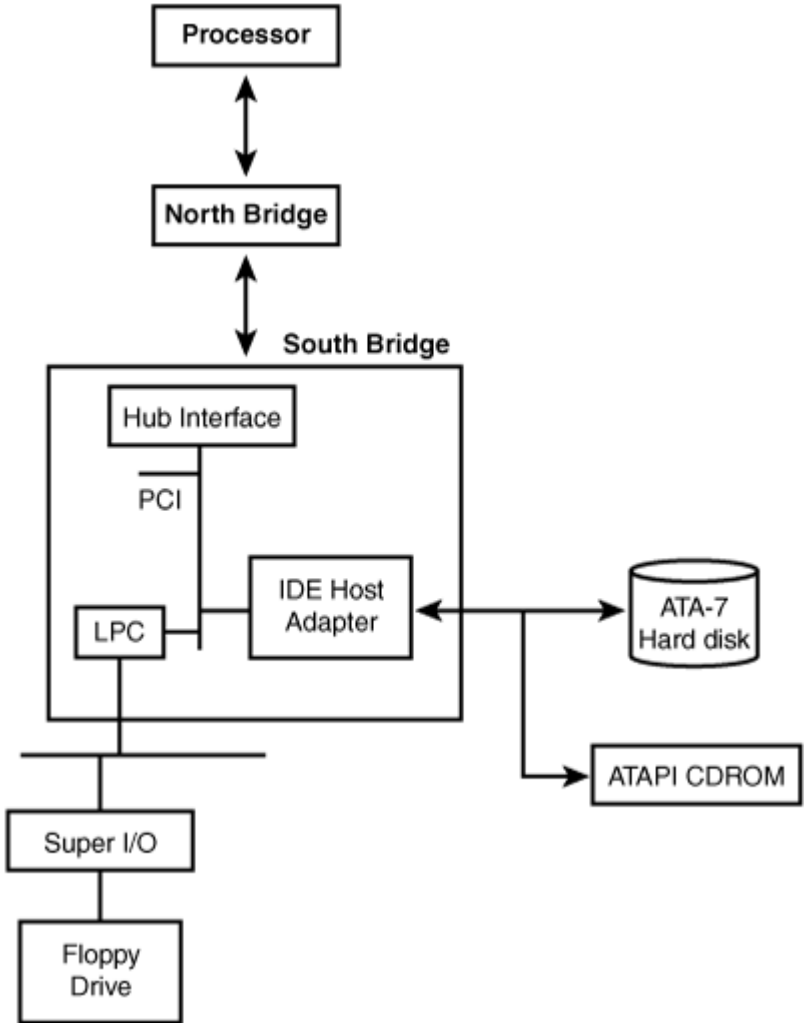
The Linux® Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel By Klaus Wehrle, Frank Pählke, Hartmut Ritter,

choice of different bottom half implementations

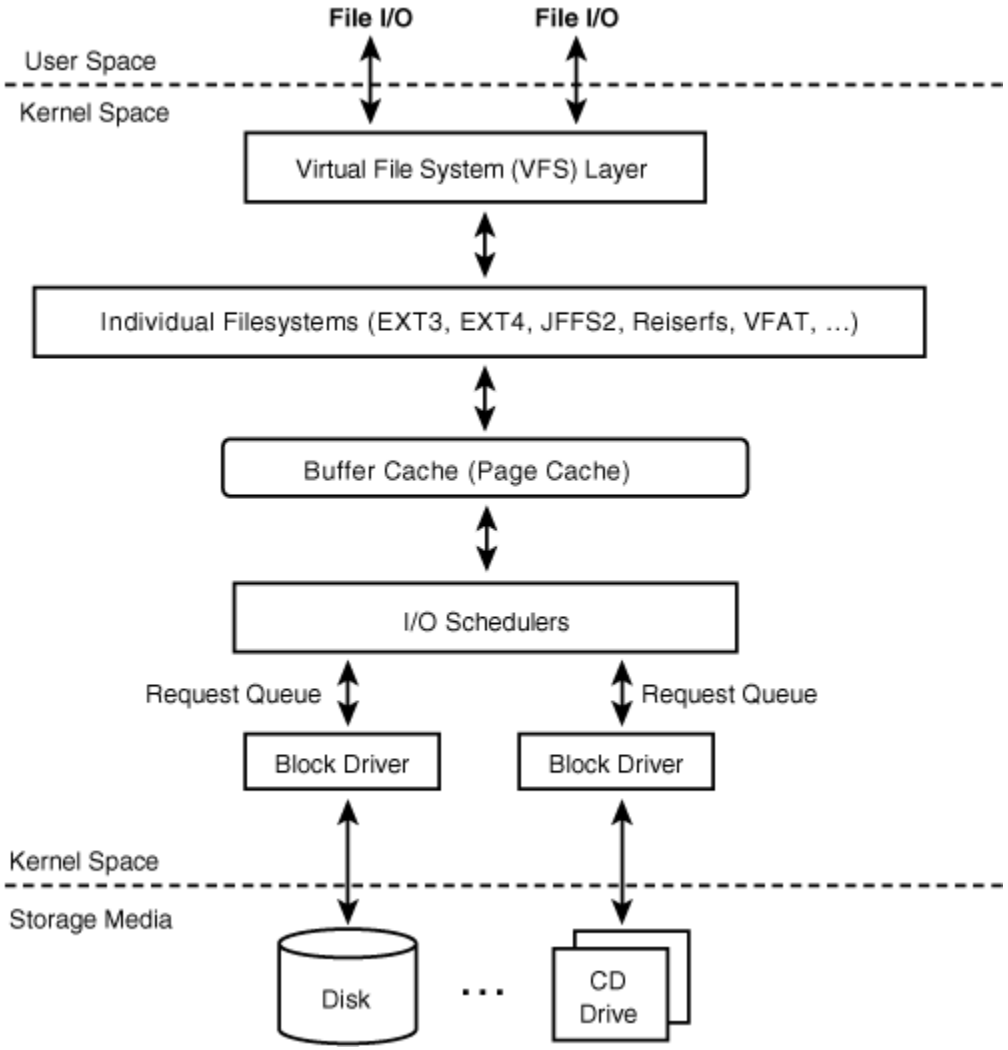
	Softirqs	**Tasklets**	**Work Queues**
Execution context	Deferred work runs in interrupt context.	Deferred work runs in interrupt context.	Deferred work runs in process context.
Reentrancy	Can run simultaneously on different CPUs.	Cannot run simultaneously on different CPUs. Different CPUs can run different tasklets, however.	Can run simultaneously on different CPUs.
Sleep semantics	Cannot go to sleep.	Cannot go to sleep.	May go to sleep.
Preemption	Cannot be preempted/scheduled.	Cannot be preempted/scheduled.	May be preempted/scheduled.
Ease of use	Not easy to use.	Easy to use.	Easy to use.
When to use	If deferred work will not go to sleep and if you have crucial scalability or speed requirements.	If deferred work will not go to sleep.	If deferred work may go to sleep.

Block device driver

Storage media in a PC system

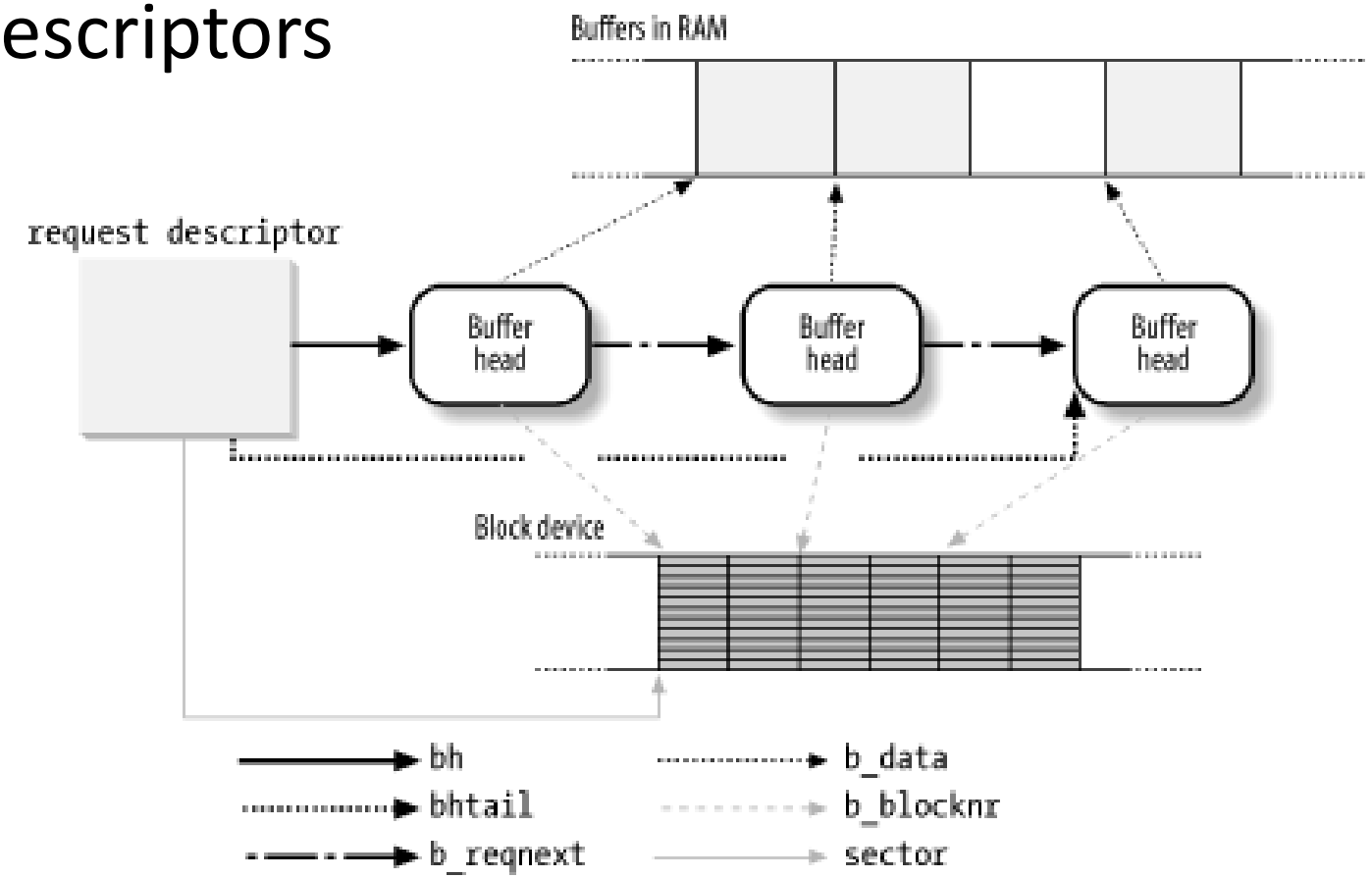


Block I/O on Linux



Block device driver

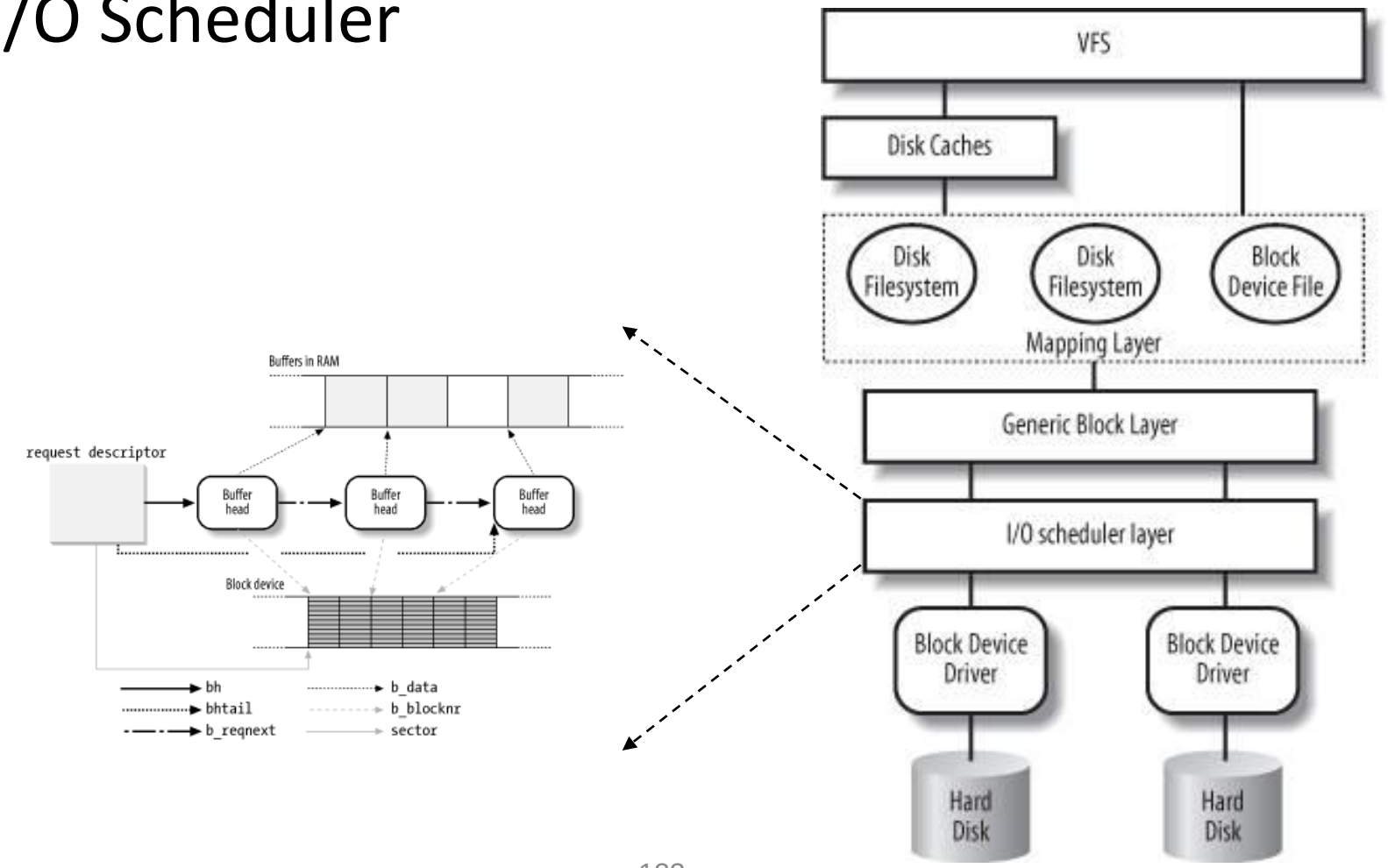
- Request descriptors, and request queue descriptors



Daniel P. Bovet, Marco Cesati, "Understanding the Linux Kernel," 2nd Edition

Block device driver

- I/O Scheduler



Block driver data structures

- The kernel represents a disk using the gendisk (short for generic disk) structure defined in include/linux/genhd.h:

```
struct gendisk {
  int major;                        /* Device major number */
  int first_minor;                  /* Starting minor number */
  int minors;                       /* Maximum number of minors.
                                       You have one minor number
                                       per disk partition */
  char disk_name[32];               /* Disk name */
  /* ... */
  struct block_device_operations *fops;
                                    /* Block device operations.
                                       Described soon. */
  struct request_queue *queue;      /* The request queue associated
                                       with this disk. Discussed
                                       next. */
  /* ... */
};
```

Block driver data structures

- The I/O request queue associated with each block driver is described using the request_queue structure defined in include/linux/blkdev.h.
- Each request in a request_queue is represented using a request structure defined in include/linux/blkdev.h:
-

```
struct request {
  /* ... */
  struct request_queue *q;  /* The container request queue */
  /* ... */
  sector_t sector;          /* Sector from which data access
                               is requested */
  /* ... */
  unsigned long nr_sectors; /* Number of sectors left to
                               submit */
  /* ... */
  struct bio *bio;          /* The associated bio. Discussed
                               soon. */
  /* ... */
  char *buffer;             /* The buffer for data transfer */
  /* ... */
  struct request *next_rq;  /* Next request in the queue */
};
```

Block driver data structures

- block_device_operations is the block driver counterpart of the file_operations structure used by character drivers.

```
struct block_device_operations {
  int (*open) (struct inode *, struct file *);   /* Open */
  int (*release) (struct inode *, struct file *);/* Close */
  int (*ioctl) (struct inode *, struct file *,
                unsigned, unsigned long);    /* I/O Control */
  /* ... */
  int (*media_changed) (struct gendisk *);   /* Check if media is
                                                available or
                                                ejected */
  int (*revalidate_disk) (struct gendisk *); /* Gear up for newly
                                                inserted media */
  /* ... */
};
```

Block driver data structures

- A bio structure is a low-level description of block I/O operations at page-level granularity. It's defined in include/linux/bio.h as follows:

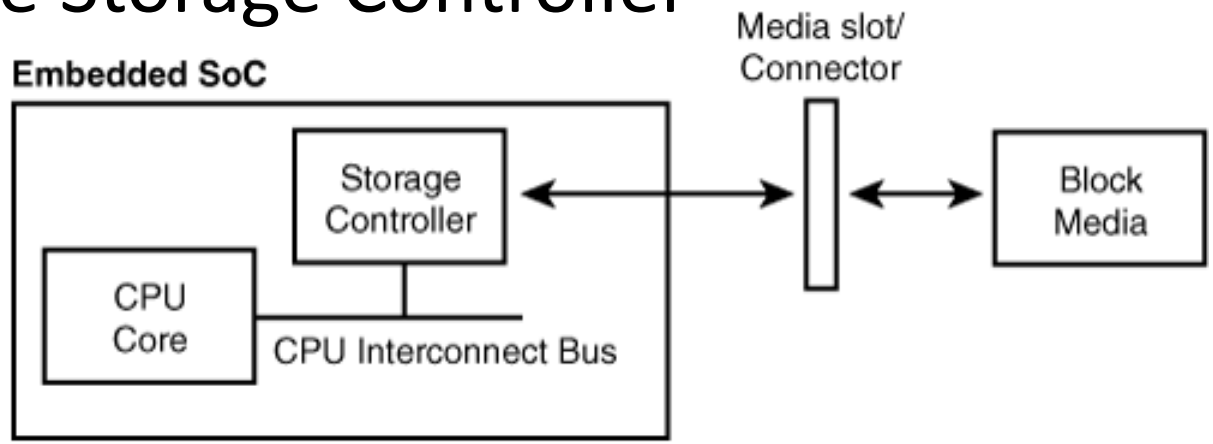
```
struct bio {
  sector_t        bi_sector;  /* Sector from which data
                                 access is requested */
  struct bio     *bi_next;    /* List of bio nodes */
  /* .. */
  unsigned long   bi_rw;      /* Bottom bits of bi_rw contain
                                 the data-transfer direction */
  /* ... */
  struct bio_vec *bi_io_vec;  /* Pointer to an array of
                                 bio_vec structures */
  unsigned short  bi_vcnt;    /* Size of the bio_vec array */
  unsigned short  bi_idx;     /* Index of the current bio_vec
                                 in the array */
  /* ... */
};
```

(page, page_offset, length)

Block driver example

Neither interrupt driven nor supports DMA

- Simple Storage Controller



Register Name	Description of Contents
SECTOR_NUMBER_REGISTER	The sector on which the next disk operation is to be performed.
SECTOR_COUNT_REGISTER	Number of sectors to be read or written.
COMMAND_REGISTER	The action to be taken (for example, read or write).
STATUS_REGISTER	Results of operations, interrupt status, and error flags.
DATA_REGISTER	In the read path, the storage controller fetches data from the disk to internal buffers. The driver accesses the internal buffer via this register. In the write path, data written by the driver to this register is transferred to the internal buffer, from where the controller copies it to disk.

Block device initialization

```
#include <linux/blkdev.h>
#include <linux/genhd.h>
static struct gendisk *myblkdisk;     /* Representation of a disk */
static struct request_queue *myblkdev_queue;
                                      /* Associated request queue */
int myblkdev_major = 0;               /* Ask the block subsystem
                                         to choose a major number */
static DEFINE_SPINLOCK(myblkdev_lock);/* Spinlock that protects
                                         myblkdev_queue from
                                         concurrent access */
int myblkdisk_size     = 256*1024;    /* Disk size in kilobytes. For
                                         a PC hard disk, one way to
                                         glean this is via the BIOS */
int myblkdev_sect_size = 512;         /* Hardware sector size */
```

```
/* Initialization */
static int __init
myblkdev_init(void)
{
  /* Register this block driver with the kernel */
  if ((myblkdev_major = register_blkdev(myblkdev_major,
                                        "myblkdev")) <= 0) {
    return -EIO;
  }
  /* Allocate a request_queue associated with this device */
  myblkdev_queue = blk_init_queue(myblkdev_request, &myblkdev_lock);
  if (!myblkdev_queue) return -EIO;
  /* Set the hardware sector size and the max number of sectors */
  blk_queue_hardsect_size(myblkdev_queue, myblkdev_sect_size);
  blk_queue_max_sectors(myblkdev_queue, 512);
  /* Allocate an associated gendisk */
  myblkdisk = alloc_disk(1);
  if (!myblkdisk) return -EIO;
  /* Fill in parameters associated with the gendisk */
  myblkdisk->fops   = &myblkdev_fops;
  /* Set the capacity of the storage media in terms of number of
     sectors */
  set_capacity(myblkdisk, myblkdisk_size*2);
  myblkdisk->queue  = myblkdev_queue;
  myblkdisk->major = myblkdev_major;
  myblkdisk->first_minor = 0;
  sprintf(myblkdisk->disk_name, "myblkdev");
  /* Add the gendisk to the block I/O subsystem */
  add_disk(myblkdisk);
  return 0;
}
```

❶ ❷ ❸ ❹ ❺ ❻

myblkdev_lock: a spinlock to protect request_queue

```
/* Exit */
static void __exit
myblkdev_exit(void)
{
  /* Invalidate partitioning information and perform cleanup */
  del_gendisk(myblkdisk);
  /* Drop references to the gendisk so that it can be freed */
  put_disk(myblkdisk);
  /* Dissociate the driver from the request_queue. Internally calls
     elevator_exit() */
  blk_cleanup_queue(myblkdev_queue);
  /* Unregister the block device */
  unregister_blkdev(myblkdev_major, "myblkdev");
}
module_init(myblkdev_init);
module_exit(myblkdev_exit);
MODULE_LICENSE("GPL");
```

Block device ioctl

```
#define GET_DEVICE_ID 0xAA00   /* Ioctl command definition */
/* The ioctl operation */
static int
myblkdev_ioctl (struct inode *inode, struct file *file,
                unsigned int cmd, unsigned long arg)
{
  unsigned char status;
  switch (cmd) {
  case GET_DEVICE_ID:
    outb(GET_IDENTITY_CMD, COMMAND_REGISTER);
    /* Wait as long as the controller is busy */
    while ((status = inb(STATUS_REGISTER)) & BUSY_STATUS);
    /* Obtain ID and return it to user space */
    return put_user(inb(DATA_REGISTER), (long __user *)arg);
  default:
    return -EINVAL;
  }
}
```

Block device access

```
#define READ_SECTOR_CMD           1
#define WRITE_SECTOR_CMD          2
#define GET_IDENTITY_CMD          3

#define BUSY_STATUS               0x10

#define SECTOR_NUMBER_REGISTER    0x20000000
#define SECTOR_COUNT_REGISTER     0x20000001
#define COMMAND_REGISTER          0x20000002
#define STATUS_REGISTER           0x20000003
#define DATA_REGISTER             0x20000004
```

```
/* Request method */
static void
myblkdev_request(struct request_queue *rq)
{
  struct request *req;
  unsigned char status;
  int i, good = 0;

  /* Loop through the requests waiting in line */
  while ((req = elv_next_request(rq)) != NULL) {
    /* Program the start sector and the number of sectors */
    outb(req->sector, SECTOR_NUMBER_REGISTER);
    outb(req->nr_sectors, SECTOR_COUNT_REGISTER);

    /* We are interested only in filesystem requests. A SCSI command
       is another possible type of request. For the full list, look
       at the enumeration of rq_cmd_type_bits in
       include/linux/blkdev.h */
     if (blk_fs_request(req)) {
       switch(rq_data_dir(req)) {
       case READ:
           /* Issue Read Sector Command */
              outb(READ_SECTOR_CMD, COMMAND_REGISTER);
           /* Traverse all requested sectors, byte by byte */
              for (i = 0; i < 512*req->nr_sectors; i++) {
           /* Wait until the disk is ready. Busy duration should be
              in the order of microseconds. Sitting in a tight loop
              for simplicity; more intelligence required in the real
              world */
             while ((status = inb(STATUS_REGISTER)) & BUSY_STATUS);

             /* Read data from disk to the buffer associated with the
                request */
             req->buffer[i] = inb(DATA_REGISTER);
           }
         good = 1;
         break;
```

```
 case WRITE:
        /* Issue Write Sector Command */
        outb(WRITE_SECTOR_CMD, COMMAND_REGISTER);

        /* Traverse all requested sectors, byte by byte */
        for (i = 0; i < 512*req->nr_sectors; i++) {
          /* Wait until the disk is ready. Busy duration should be
             in the order of microseconds. Sitting in a tight loop
             for simplicity; more intelligence required in the real
             world */
           while ((status = inb(STATUS_REGISTER)) & BUSY_STATUS);

          /* Write data to disk from the buffer associated with the
             request */
          outb(req->buffer[i], DATA_REGISTER);
        }
        good = 1;
        break;
      }
    }
    end_request(req, good);
  }
}
```

Block device driver

- I/O Scheduler
  - Sending out requests to the block devices, as soon as it issues them, results in awful performance
  - Minimizing seeks is absolutely crucial to the system's performance
  - Performs operations called merging and sorting to greatly improve the performance

I/O Scheduler	Description	Source File
Linus elevator	Straightforward implementation of the standard merge-and-sort I/O scheduling algorithm.	drivers/block/elevator.c (in the 2.4 kernel tree)
Deadline	In addition to what the Linus elevator does, the Deadline scheduler associates expiration times with each request in order to ensure that a burst of requests to the same disk region do not starve requests to regions that are farther away. Moreover, read operations are granted more priority than write operations because user processes usually block until their read requests complete. The Deadline scheduler thus ensures that each I/O request is serviced within a time limit, which is important for some database loads.	block/deadline-iosched.c (in the 2.6 kernel tree)
Anticipatory	Similar to the Deadline scheduler, except that after servicing read requests, the Anticipatory scheduler waits for a predetermined amount of time anticipating further requests. This scheduling technique is suited for workstation/interactive loads.	block/as-iosched.c (in the 2.6 kernel tree)

Complete Fair Queuing (CFQ)	Similar to the Linus elevator, except that the CFQ scheduler maintains one request queue per originating process, rather than one generic queue. This ensures that each process (or process group) gets a fair portion of the I/O and prevents one process from starving others.	block/cfq-iosched.c (in the 2.6 kernel tree)
Noop	The Noop scheduler doesn't spend time traversing the request queue searching for optimal insertion points. Instead, it simply adds new requests to the tail of the request queue. This scheduler is thus ideal for semiconductor storage media that have no moving parts and, hence, no seek latencies. An example is a Disk-On-Module (DOM), which internally uses flash memory.	block/noop-iosched.c (in the 2.6 kernel tree)

Block device driver

- I/O Scheduler
  - Merge
    - Merge multiple requests together if they are adjacent sectors
  - Sort
    - Insert requests to the list based on the location (to avoid extra seek)

Block device driver

- I/O Scheduler
  - What is the scheduler algorithm ?
  - How to prevent starvation ?
  - Is that fair to all processes ?

Block device driver

- The Linus Elevator
  - Used in 2.4
  - Algorithm
    - First, if a request to an adjacent on-disk sector is in the queue, the existing request and the new request are merged into a single request.
    - Second, if a request in the queue is sufficiently old, the new request is inserted at the tail of the queue to prevent starvation of the other, older, requests.
    - Next, if there is a suitable location sector-wise in the queue, the new request is inserted there. This keeps the queue sorted by physical location on disk.
    - Finally, if no such suitable insertion point exists, the request is inserted at the tail of the queue

Block device driver

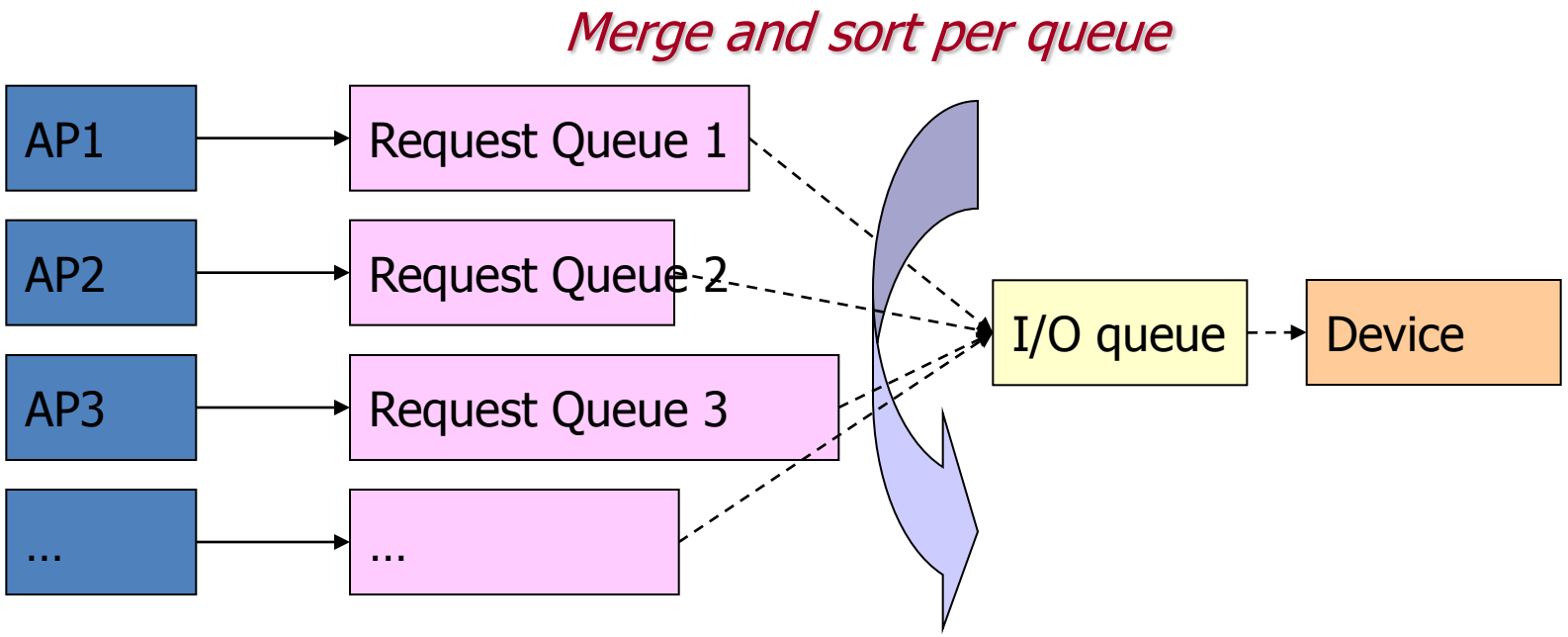
- Deadline I/O scheduler
  - Used in 2.6
  - Consider deadline
    - read: 500ms, write: 5s
  - Two queues
    - Sector queue: Sorted by sector (read/write queues)
    - FIFO queue: Sorted by expire time (read/write queues)
  - Sector queue first, if FIFO queue expire, schedule FIFO queue
  - Sort and merge request without starvation
  - Prefer read requests (deadline)

Block device driver

- Anticipatory I/O scheduler
  - Default in 2.6
  - Based on deadline I/O scheduler (good for read requests, bad to global throughout)
  - Anticipation heuristics (read requests are continue)
  - Wait for a while before performing commit read request to queue

Block device driver

- Complete fair queuing I/O scheduler
  - Used in 2.6



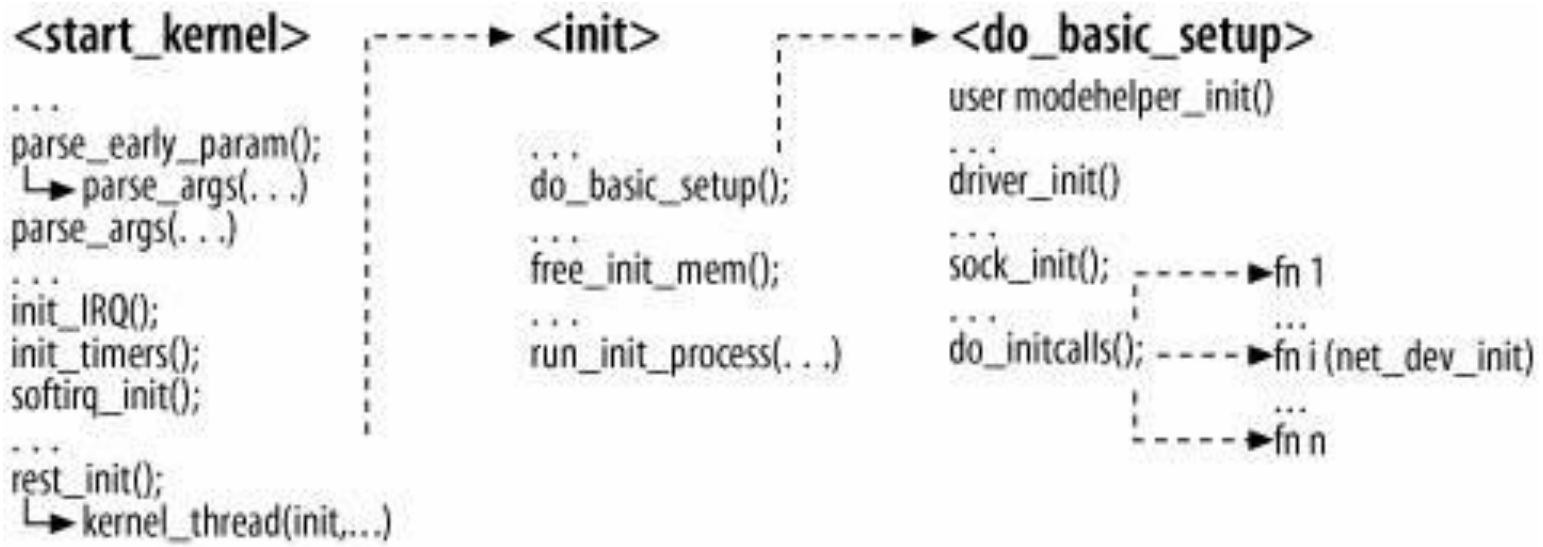
Block device driver

- Noop I/O scheduler
  - Used in 2.6
  - No sorting, and merging only
  - Merge with adjacent request only
  - Near-FIFO order
  - For truly random-access device

Network interface card and network protocol overview

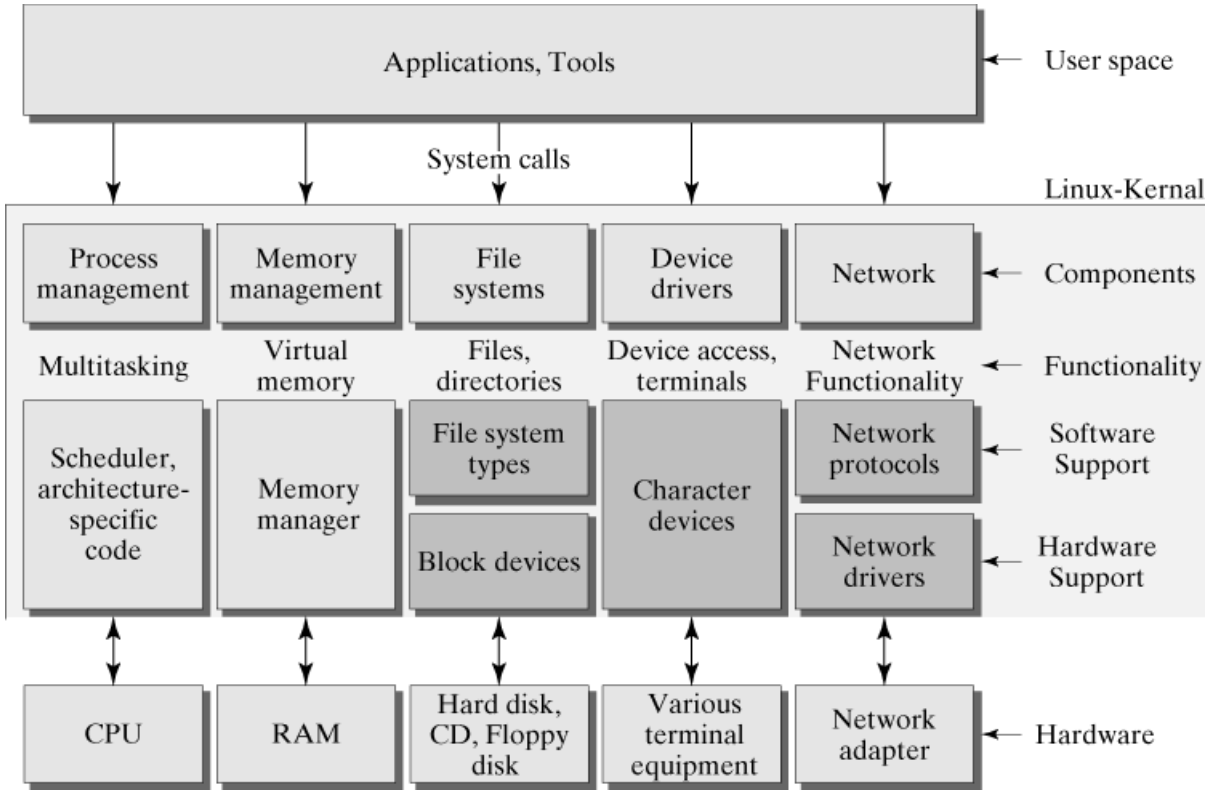
Network interface and interrupt handler

- NIC driver and protocol initiation



"Understanding Linux Network Internals," By Christian Benvenuti

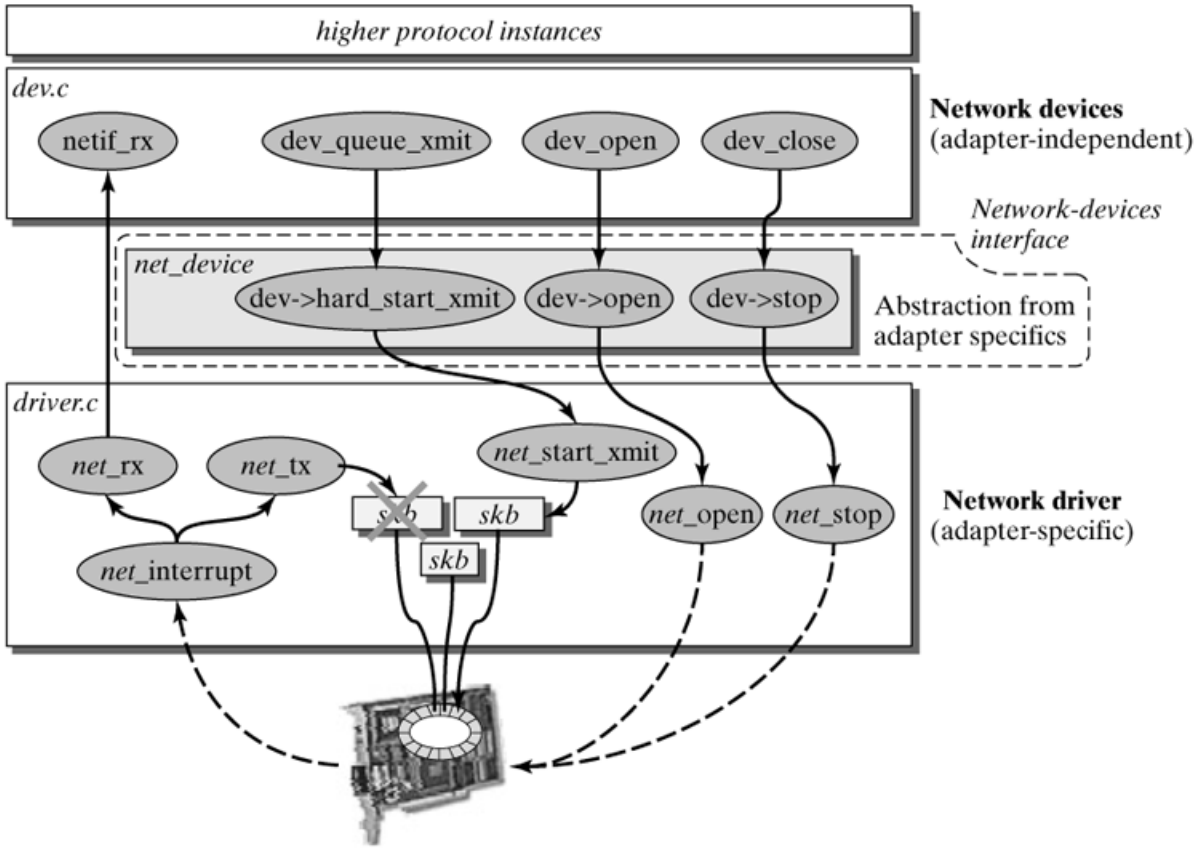
Network interface and interrupt handler



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

Network interface and interrupt handler

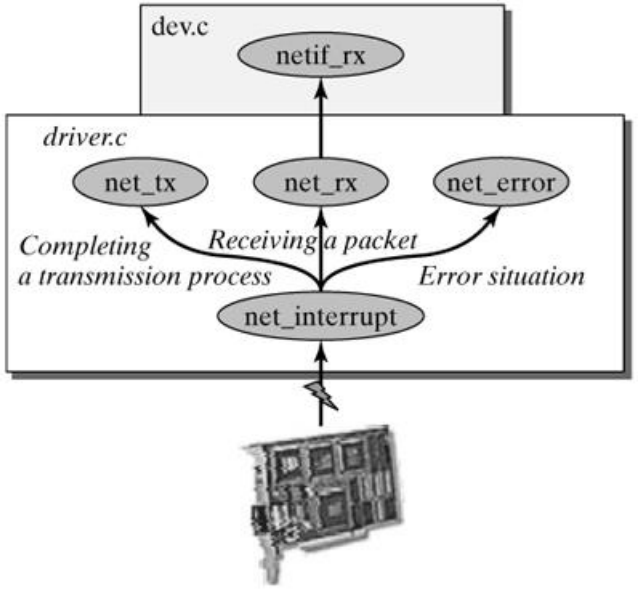
- The structure of a network device interface



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

Network interface and interrupt handler

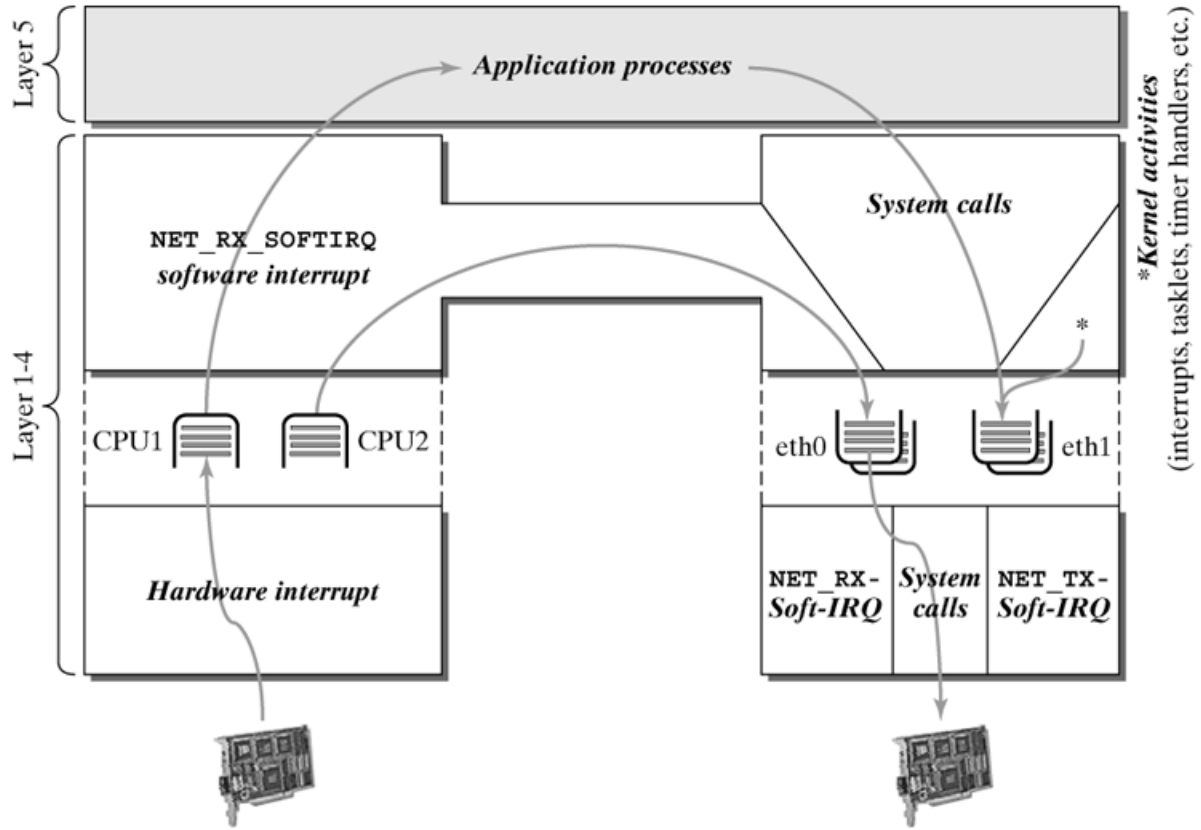
- Interrupt (top half)



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

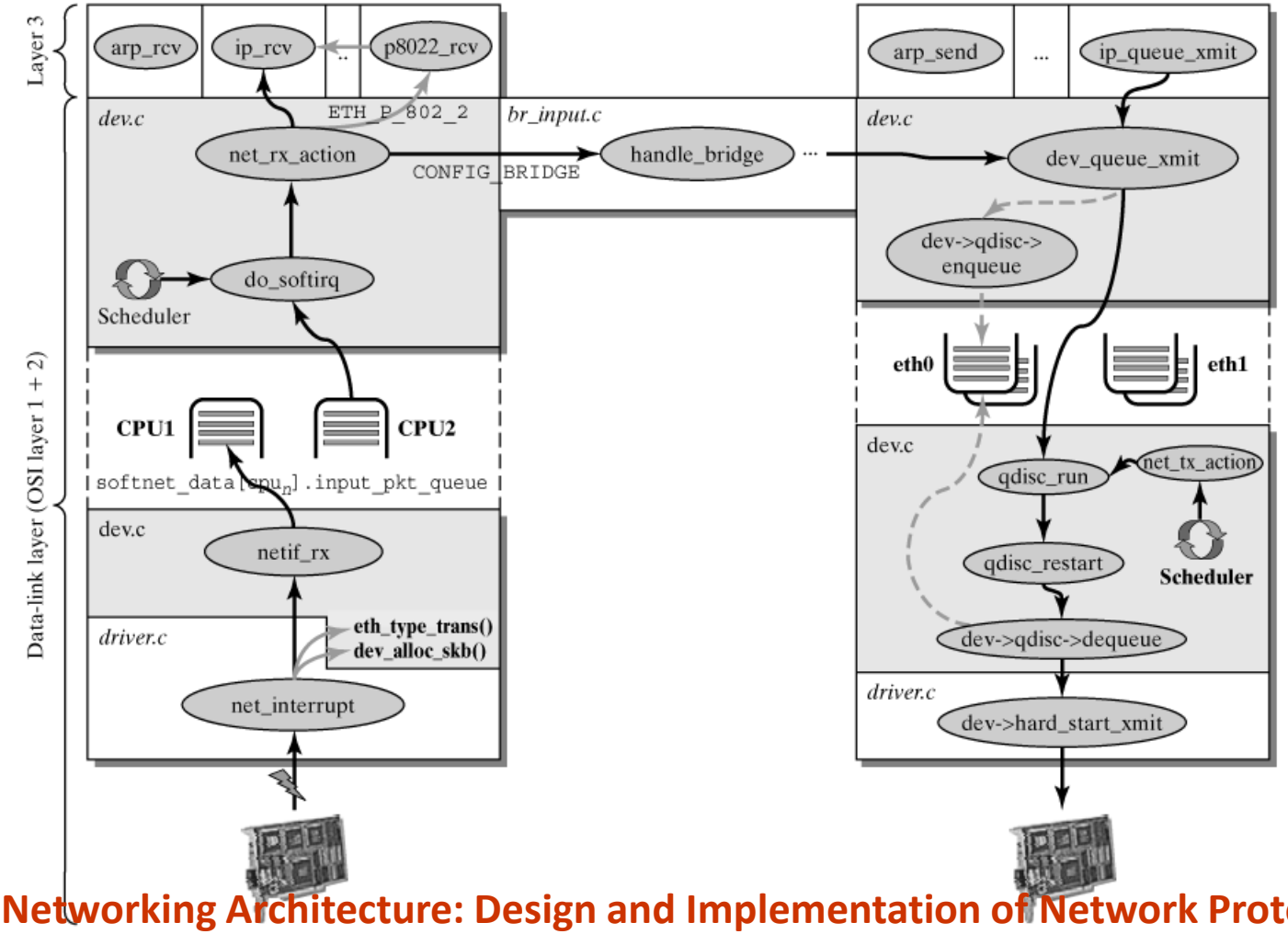
Network interface and interrupt handler

- Top halves and bottom halves



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

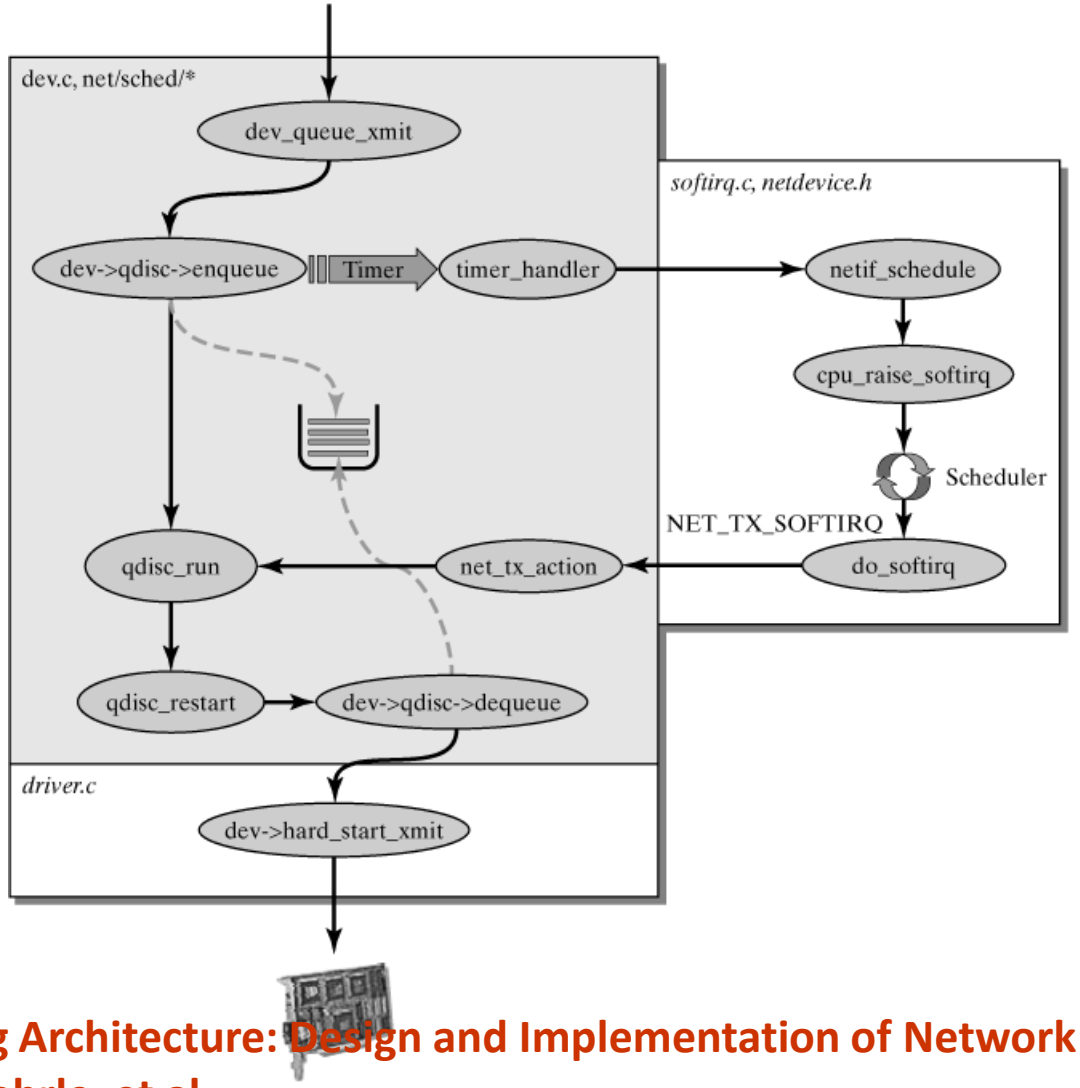
NIC driver and protocol stack



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

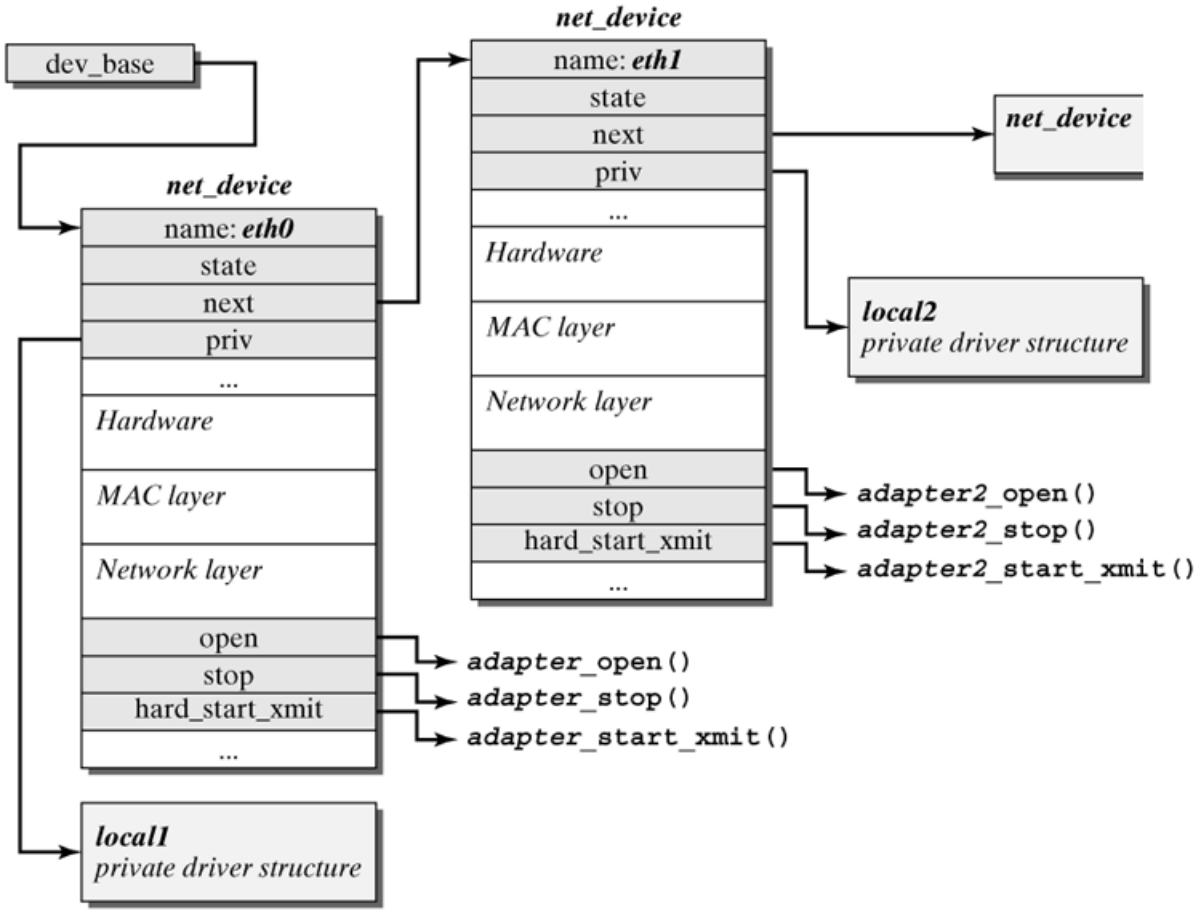
NIC driver and protocol stack

- TX



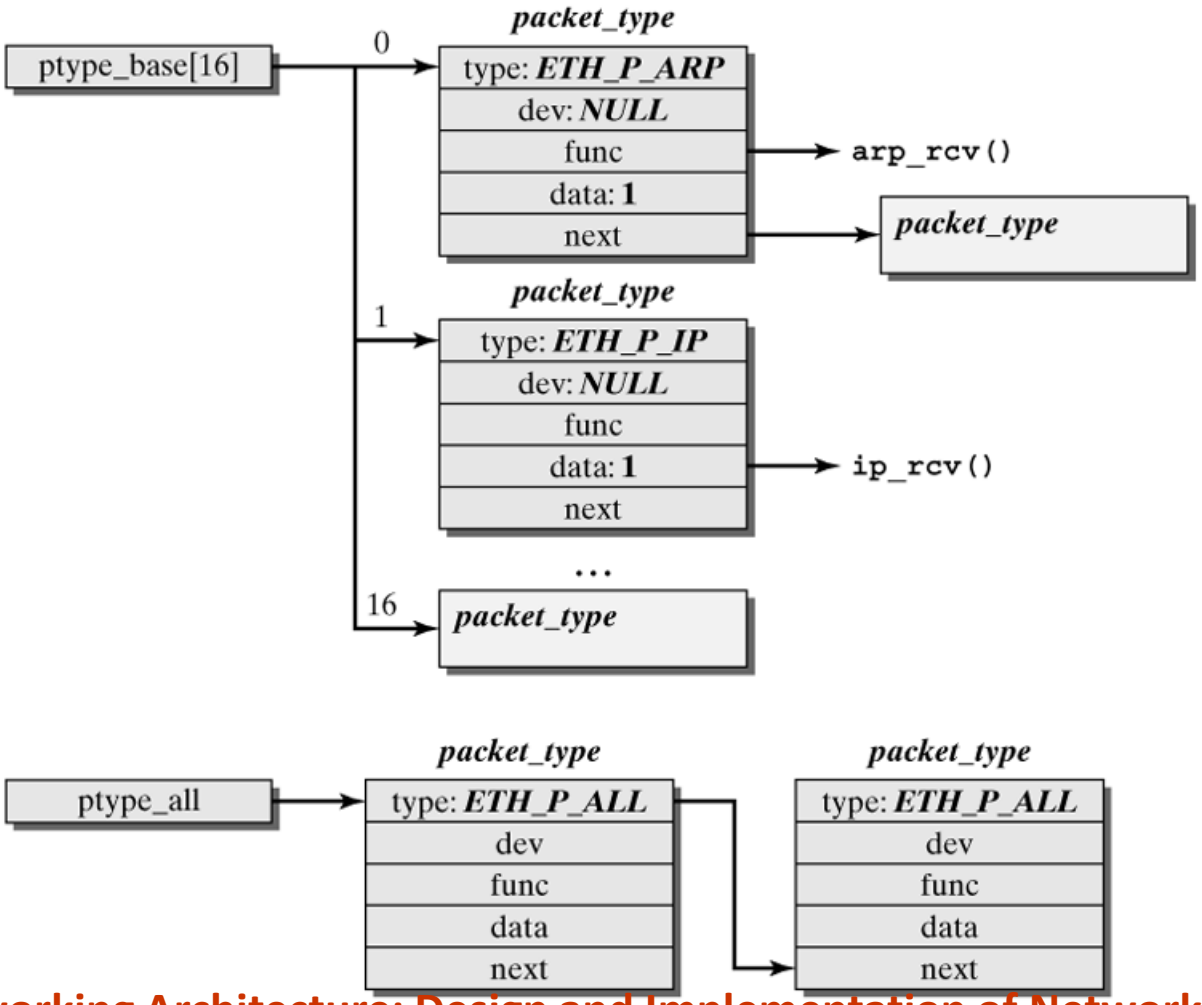
"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

NIC driver and protocol stack



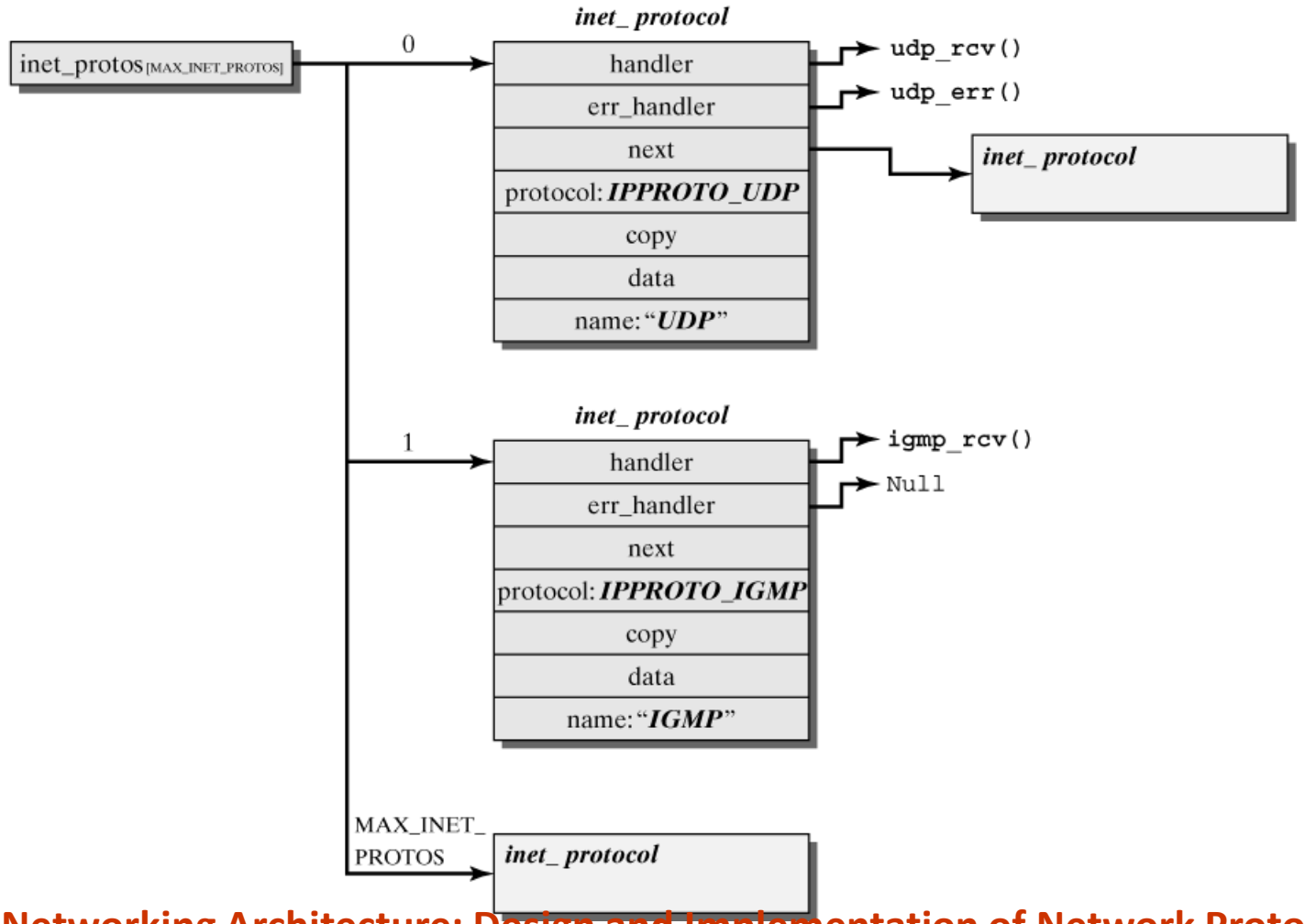
"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

NIC driver and protocol stack



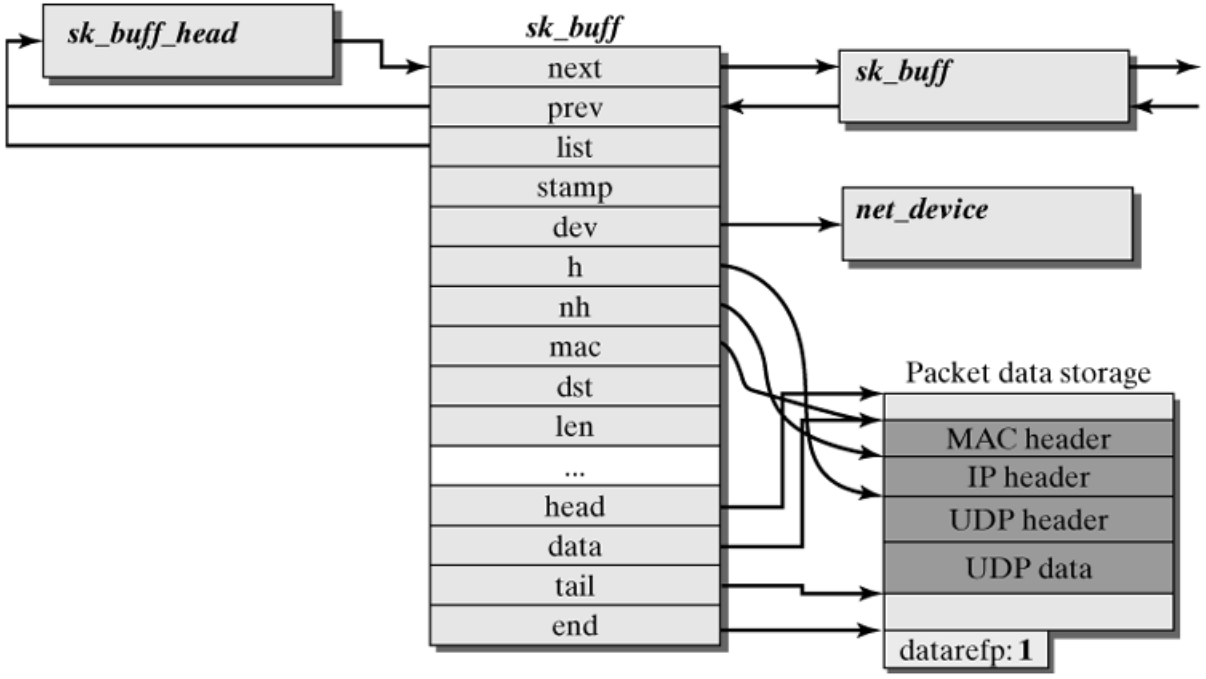
"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

NIC driver and protocol stack



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

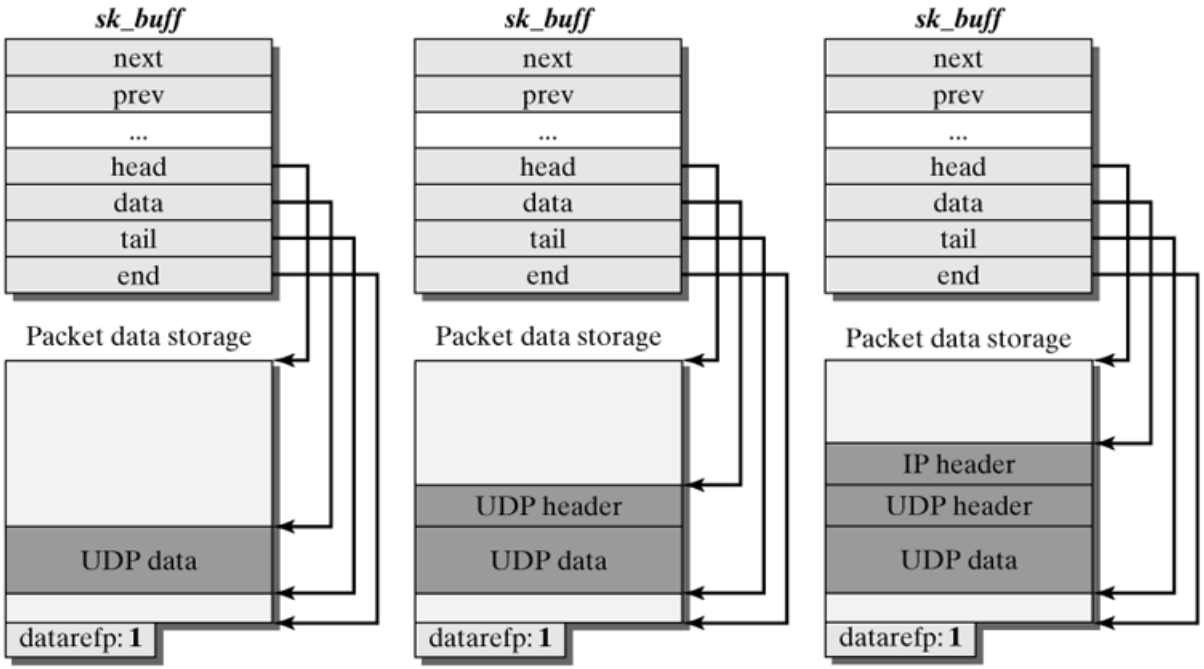
Buffer management



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

Buffer management

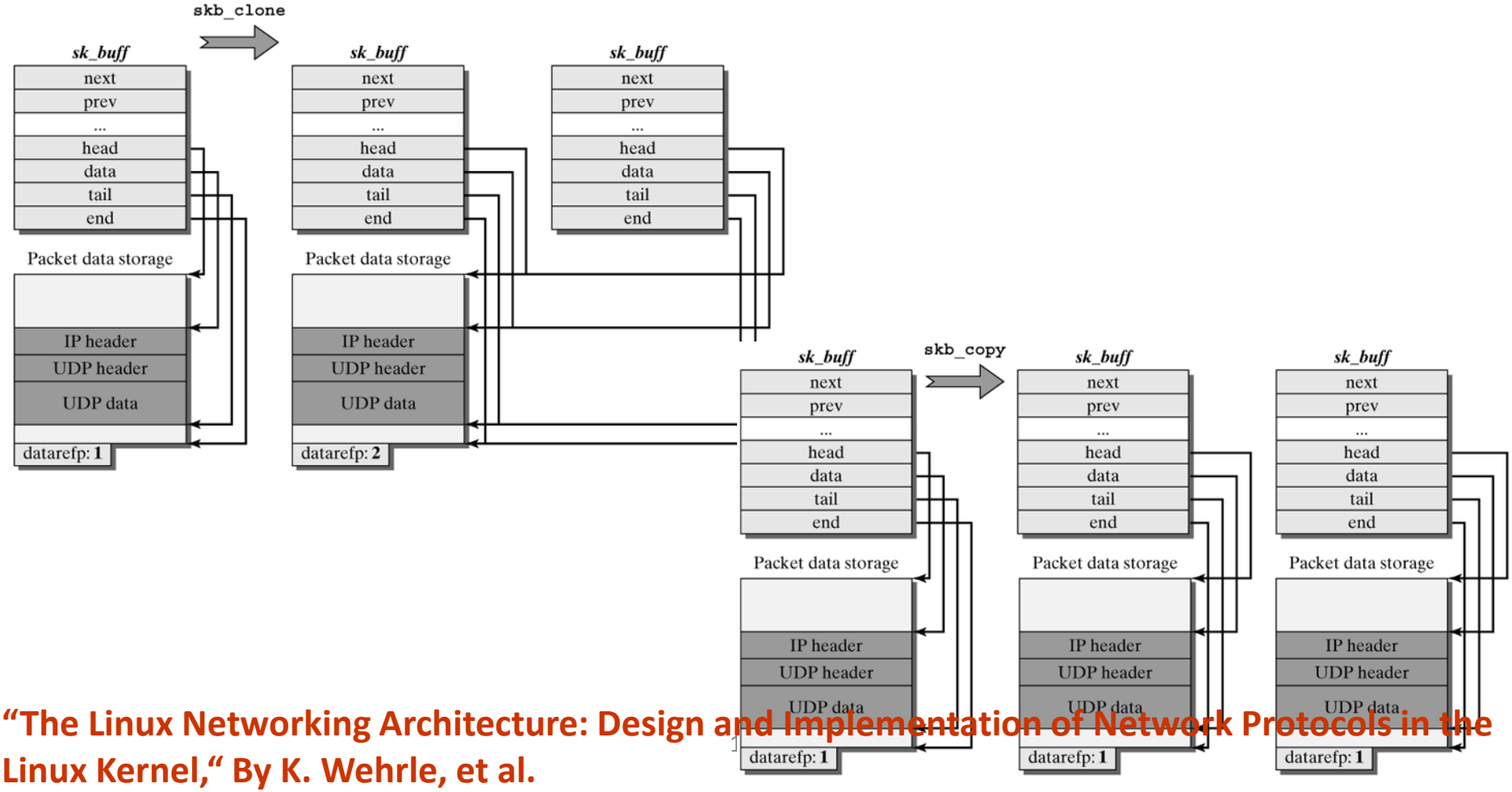
- Changes to the packet buffers across the protocol hierarchy



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

Buffer management

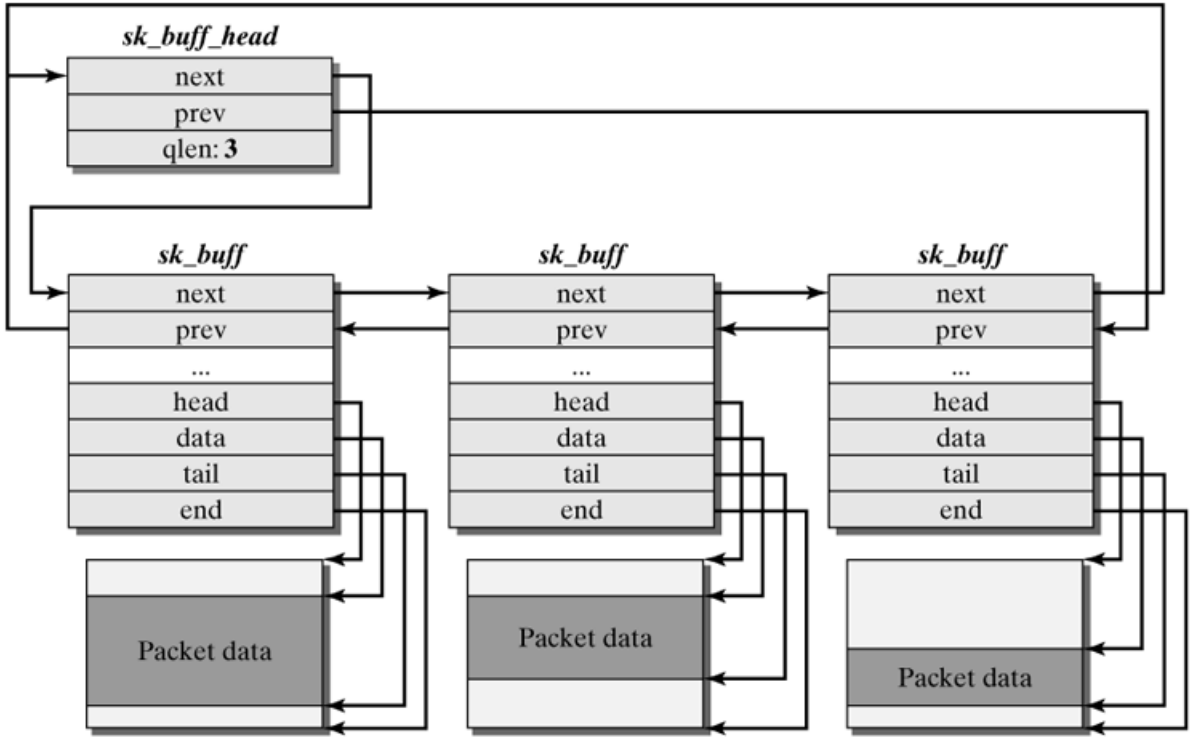
- copy vs. clone



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

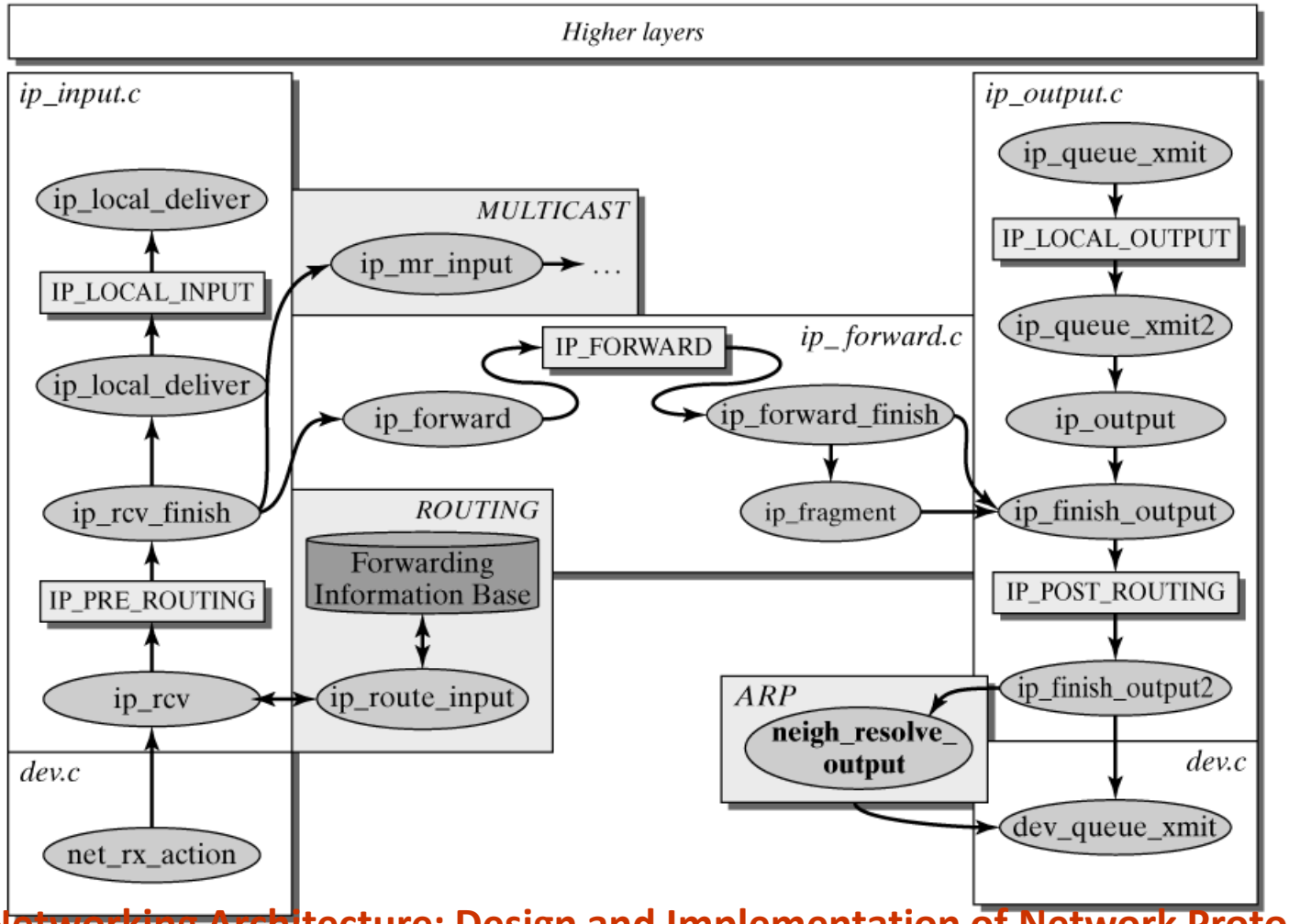
Buffer management

- Socket queue



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

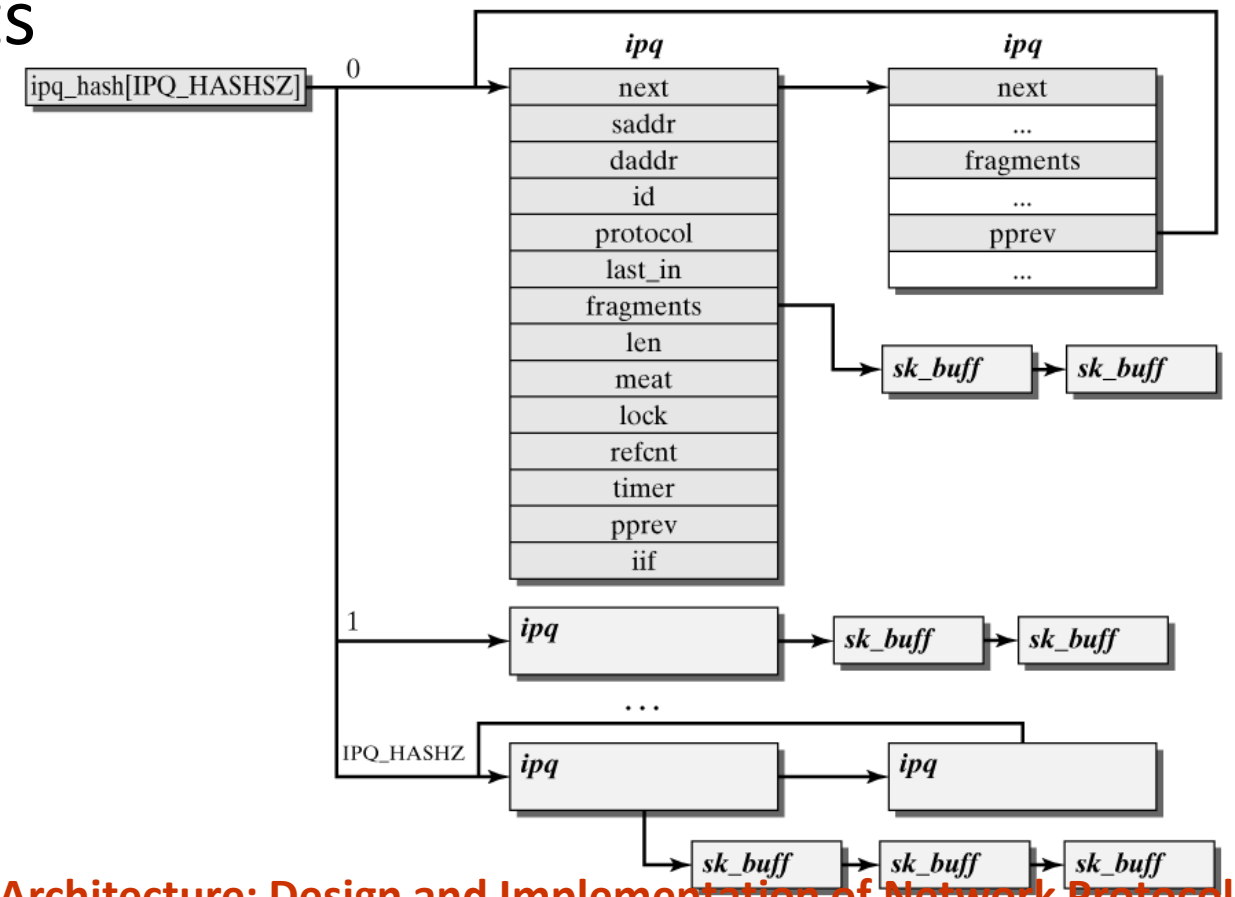
TCP/IP



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel," By K. Wehrle, et al.

TCP/IP

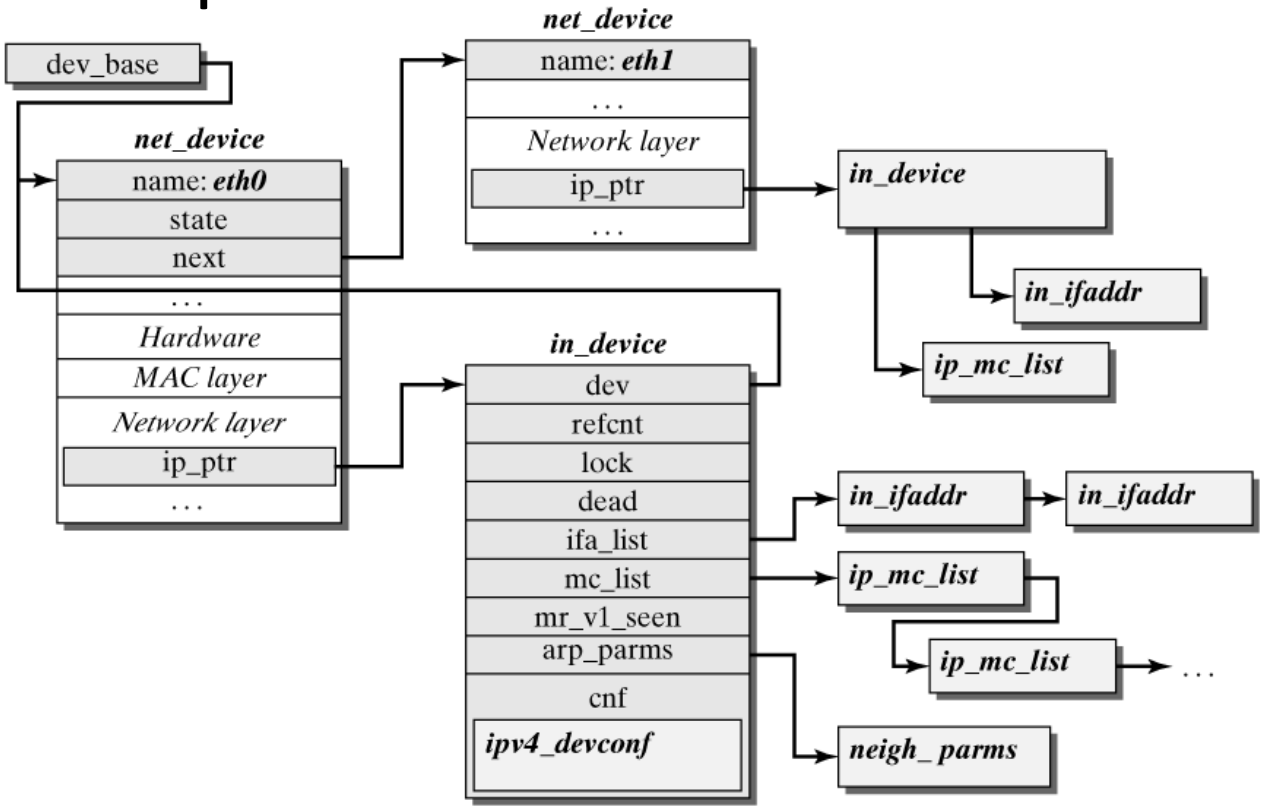
- fragment cache manages all incoming IP fragments



"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel " By K. Wehrle, et al

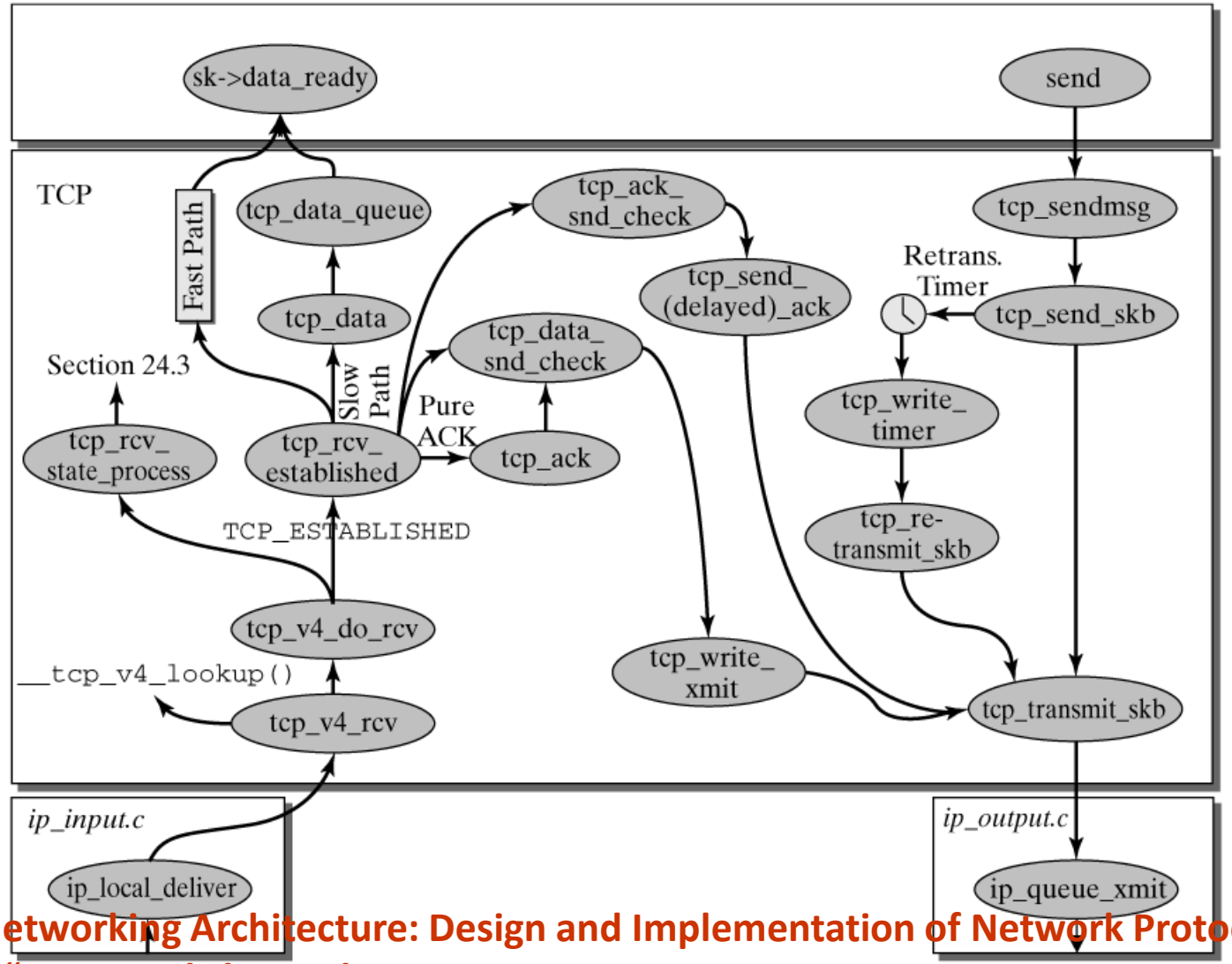
TCP/IP

- Data structures to manage IP network devices and their parameters



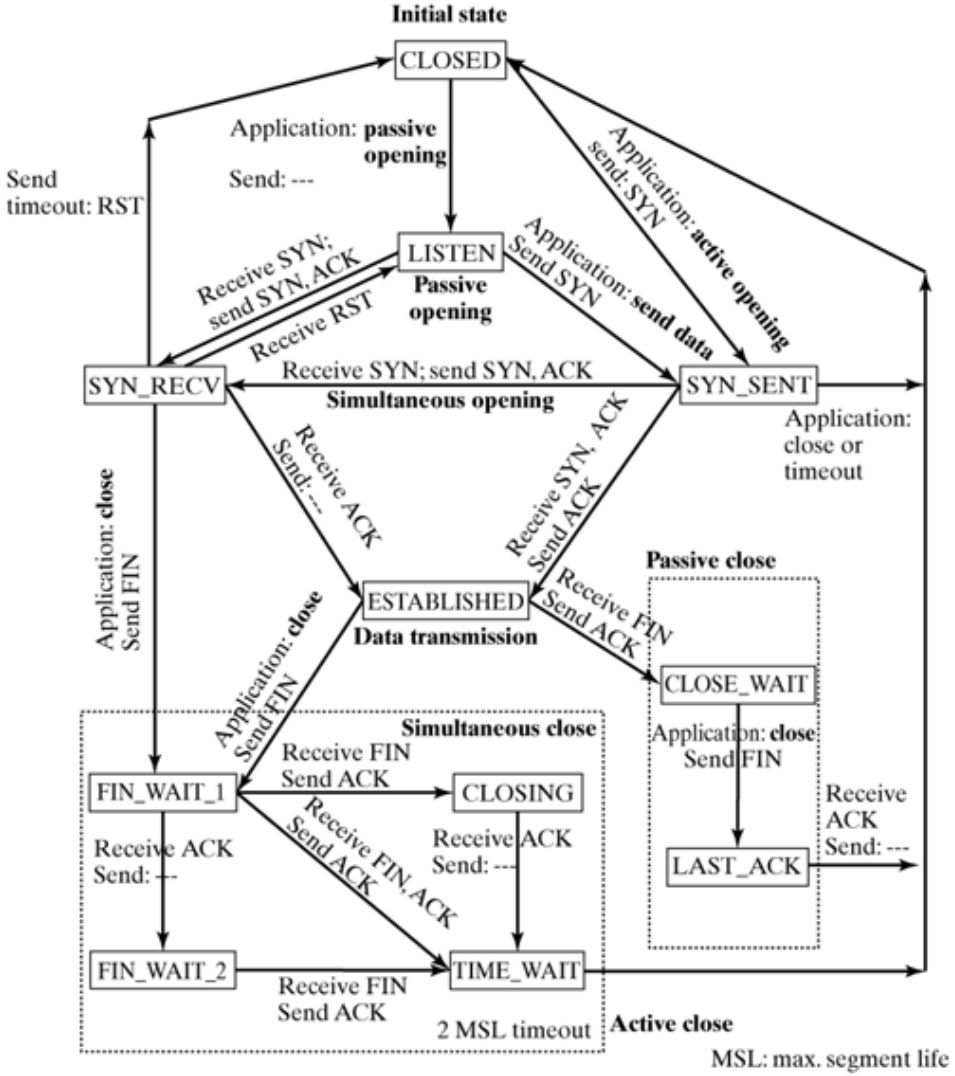
"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel" By K. Wehrle, et al

TCP/IP



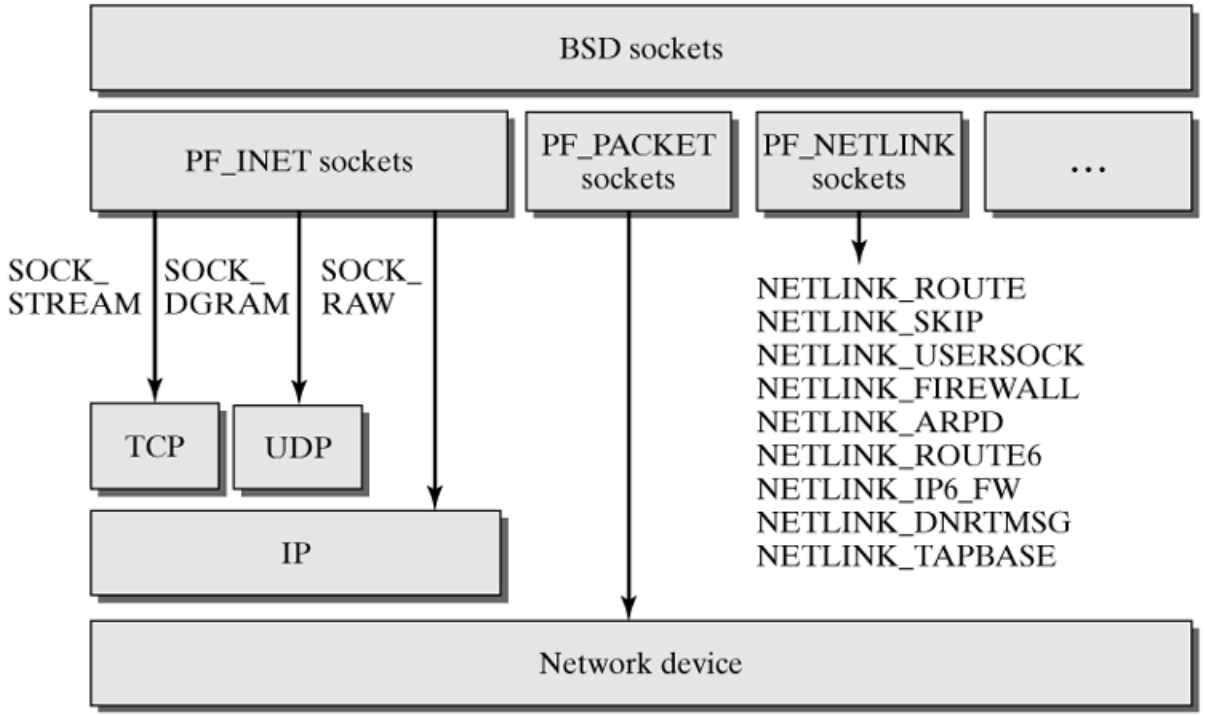
"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel" By K. Wehrle, et al

TCP state


Initial state
CLOSED
Application: passive opening
Send: ---
Send timeout: RST
Application: active opening
send: SYN
LISTEN
Passive opening
Receive SYN; send SYN, ACK
Receive RST
Application: send data
Send SYN
SYN_RECV
Receive SYN; send SYN, ACK
Simultaneous opening
SYN_SENT
Application: close or timeout
Receive ACK
Send: ---
Receive SYN, ACK
Send ACK
Application: close
Send FIN
ESTABLISHED
Data transmission
Passive close
Receive FIN
Send ACK
CLOSE_WAIT
Application: close
Send FIN
Application: close
Send FIN
Simultaneous close
Receive FIN
Send ACK
FIN_WAIT_1
CLOSING
Receive ACK
Send: ---
Receive FIN, ACK
Send ACK
Receive ACK
Send: ---
Receive ACK
Send: ---
LAST_ACK
FIN_WAIT_2
TIME_WAIT
Receive FIN
Send ACK
2 MSL timeout
Active close
MSL: max. segment life

Socket API

sys_socketcall(int call, unsigned long *args)

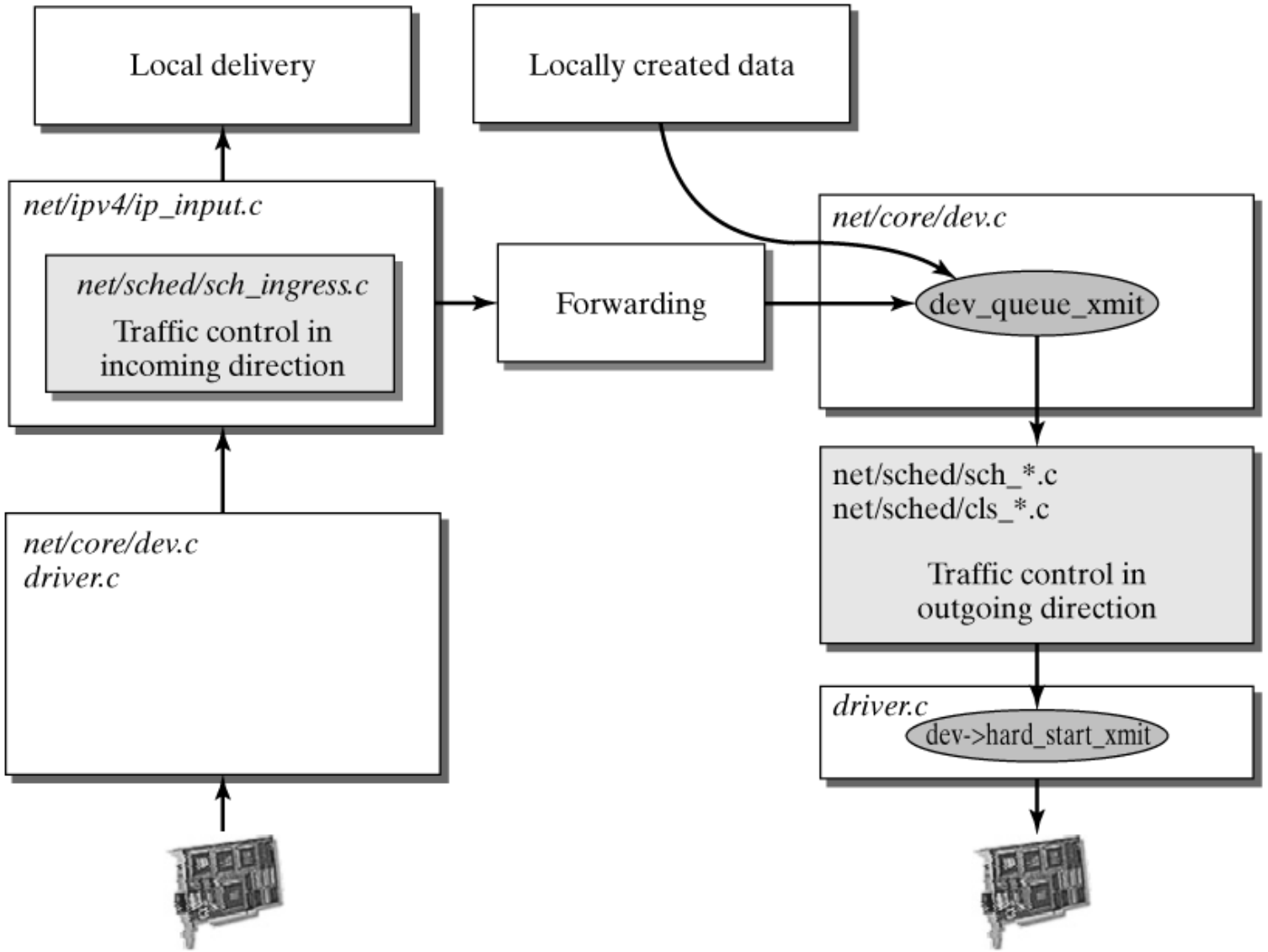


"The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel" By K. Wehrle, et al

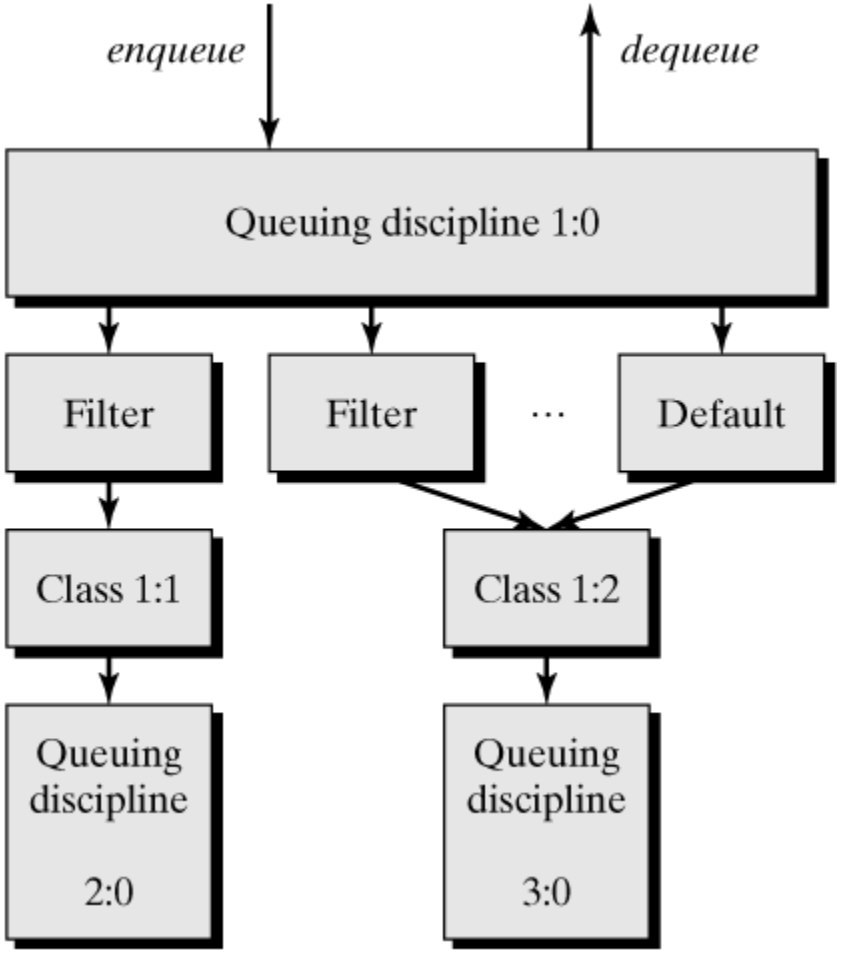
Socket API

```
if copy_from_user(a, args, nargs[call]))
       return -EFAULT;
a0=a[0];
a1=a[1];
switch(call)
      {
       case SYS_SOCKET:
            err = sys_socket(a0,a1,a[2]);
            break;
       case SYS_BIND:
            err = sys_bind(a0, (struct sockaddr *)a1, a[2]);
            break;
       case SYS_CONNECT:
            err = sys_connect (a0, (struct sockaddr *)a1, a[2]);
            break;
       case SYS_LISTEN:
            err = sys_listen (a0,a1);
            break;
       ...
       }
...
```

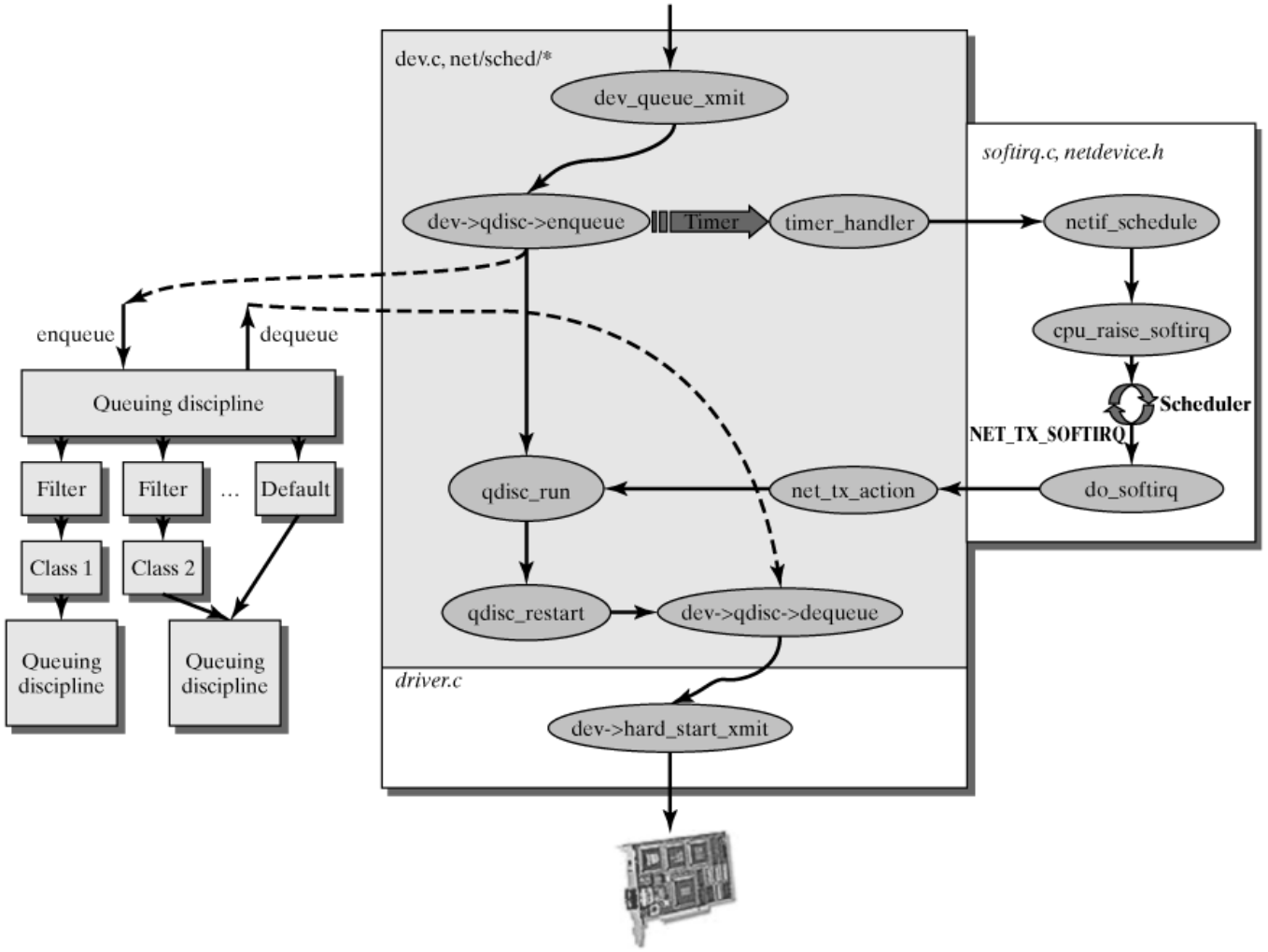
QoS


Local delivery
Locally created data
net/ipv4/ip_input.c
net/sched/sch_ingress.c
Traffic control in incoming direction
Forwarding
net/core/dev.c
dev_queue_xmit
net/sched/sch_*.c
net/sched/cls_*.c
Traffic control in outgoing direction
net/core/dev.c
driver.c
driver.c
dev->hard_start_xmit

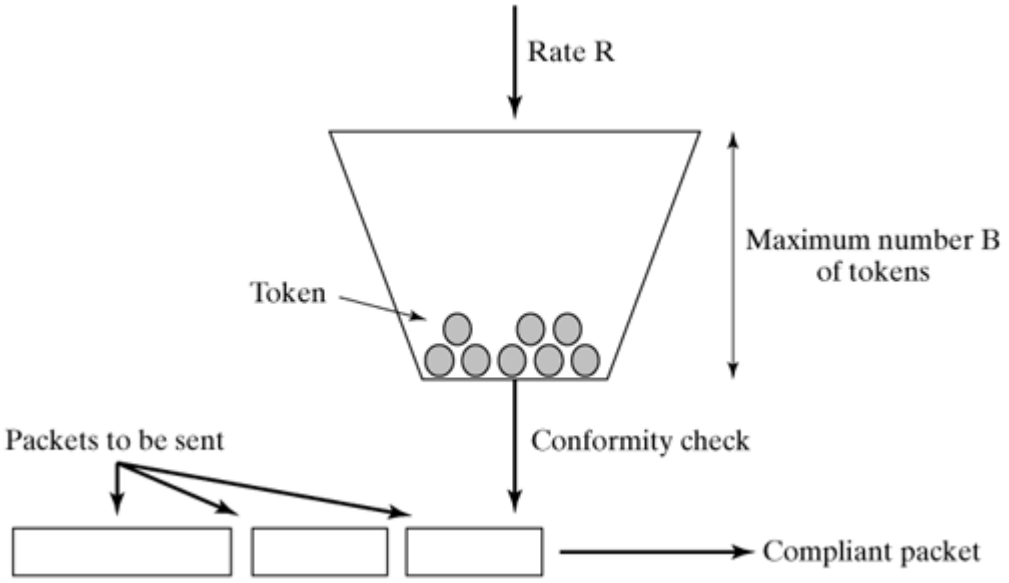
QoS (Cont.)


enqueue
dequeue
Queuing discipline 1:0
Filter
Filter
···
Default
Class 1:1
Class 1:2
Queuing discipline 2:0
Queuing discipline 3:0

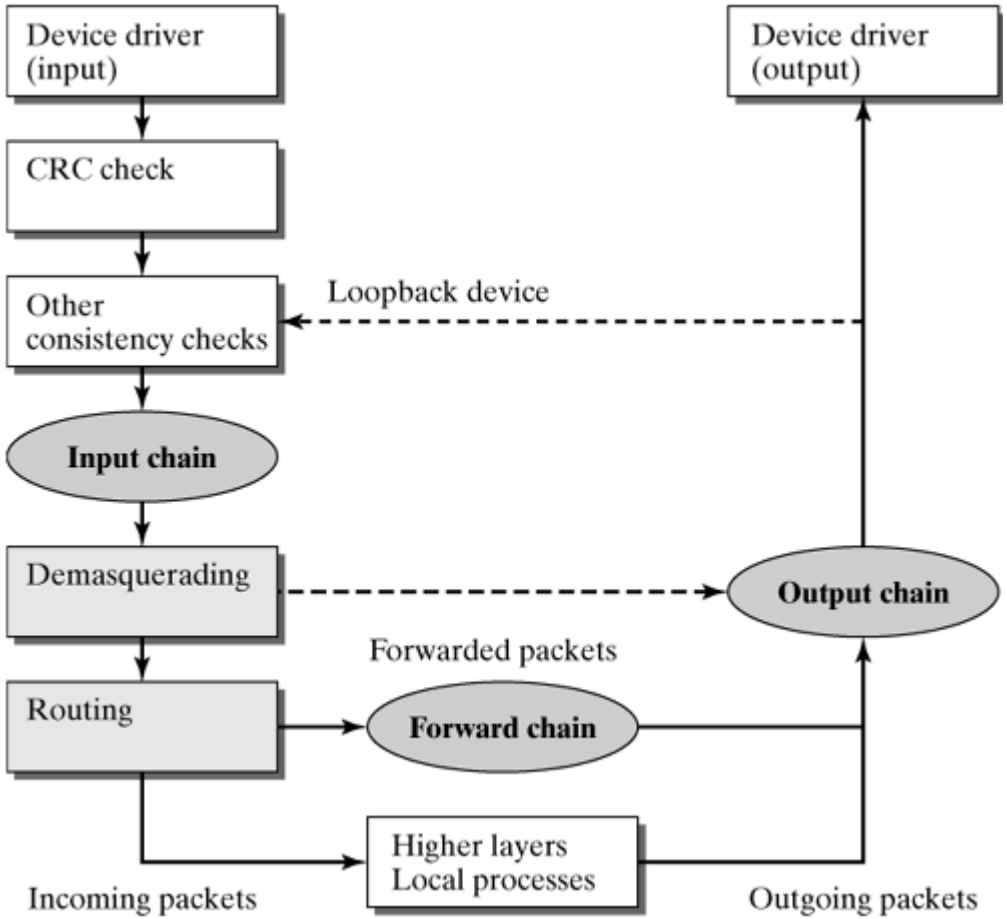
QoS (Cont.)


dev.c, net/sched/*
dev_queue_xmit
dev->qdisc->enqueue
Timer
timer_handler
softirq.c, netdevice.h
netif_schedule
cpu_raise_softirq
Scheduler
NET_TX_SOFTIRQ
do_softirq
net_tx_action
qdisc_run
qdisc_restart
dev->qdisc->dequeue
driver.c
dev->hard_start_xmit
enqueue
dequeue
Queuing discipline
Filter
Filter
...
Default
Class 1
Class 2
Queuing discipline
Queuing discipline

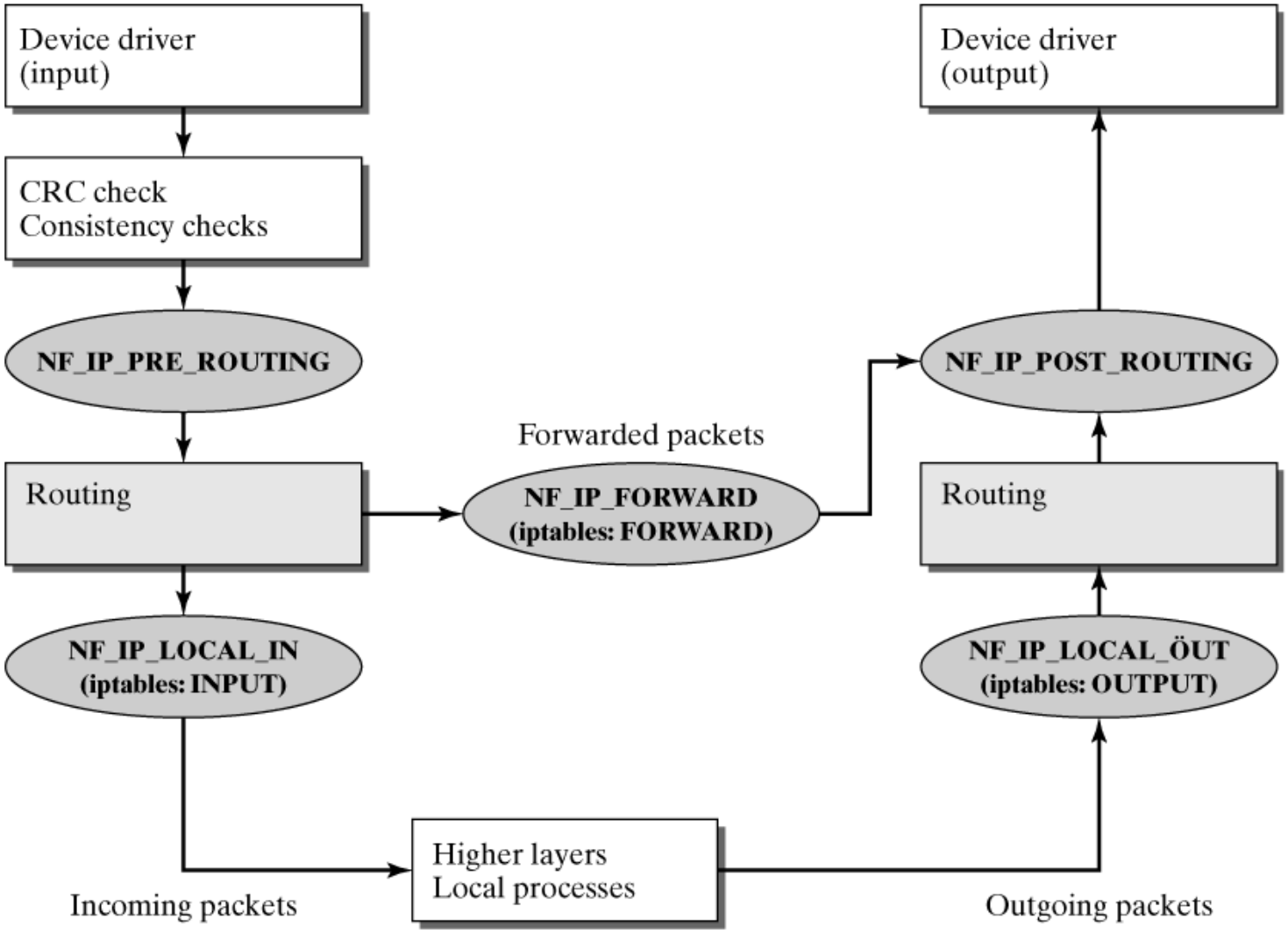
QoS (Cont.)


Rate R
Maximum number B
of tokens
Token
Packets to be sent
Conformity check
Compliant packet

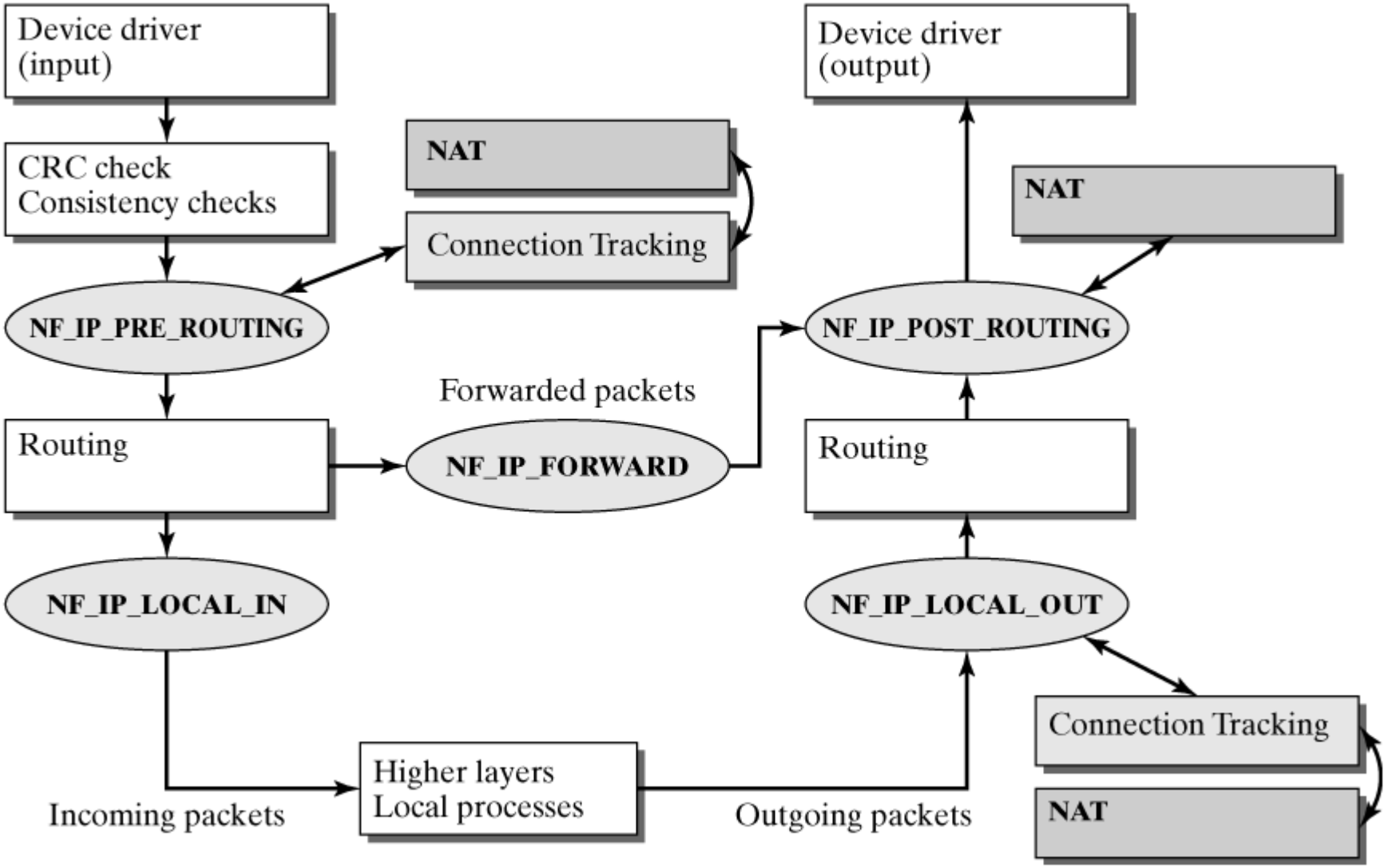
Packet Filters and Firewalls



Packet Filters and Firewalls



NAT

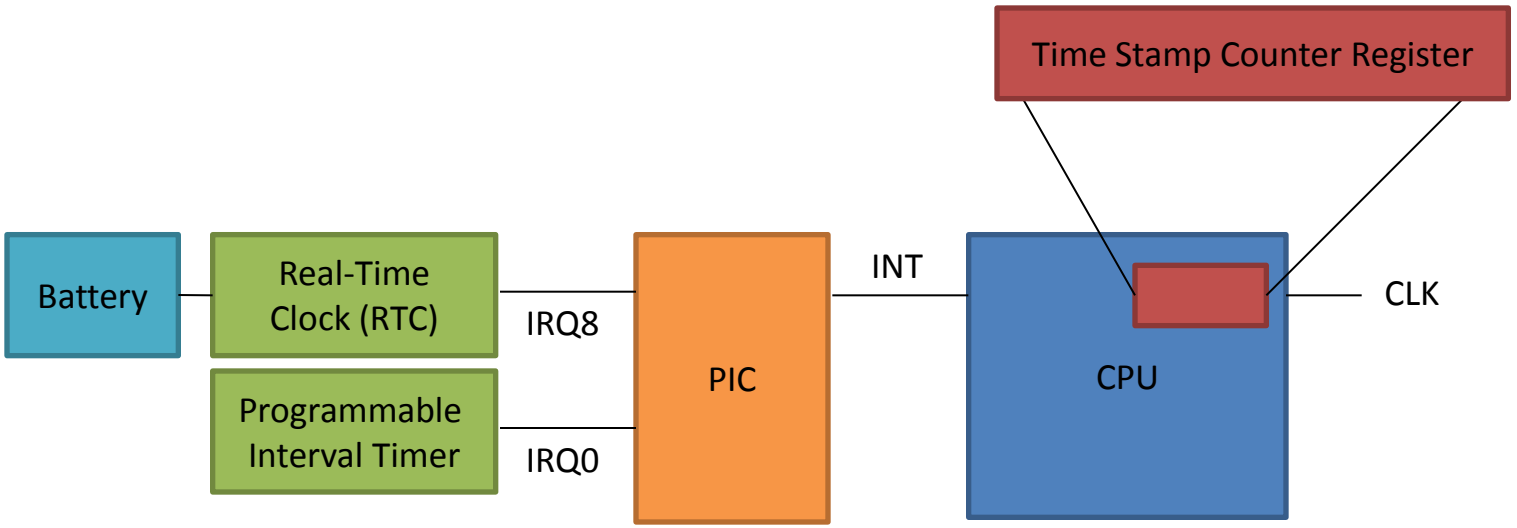

Device driver
(input)
CRC check
Consistency checks
NF_IP_PRE_ROUTING
Routing
NF_IP_LOCAL_IN
Incoming packets
Higher layers
Local processes
Outgoing packets
NF_IP_LOCAL_OUT
Routing
NF_IP_POST_ROUTING
Device driver
(output)
Forwarded packets
NF_IP_FORWARD
NAT
Connection Tracking
NAT
Connection Tracking
NAT

Timer management in kernel

- Overall HW architecture
- Overall SW architecture
- Initial procedures
- Top half procedures
- Bottom half procedures
- Software timer management
- Interface to user processes
- Long delay in kernel
- Short delay in kernel
- Tick-less kernel

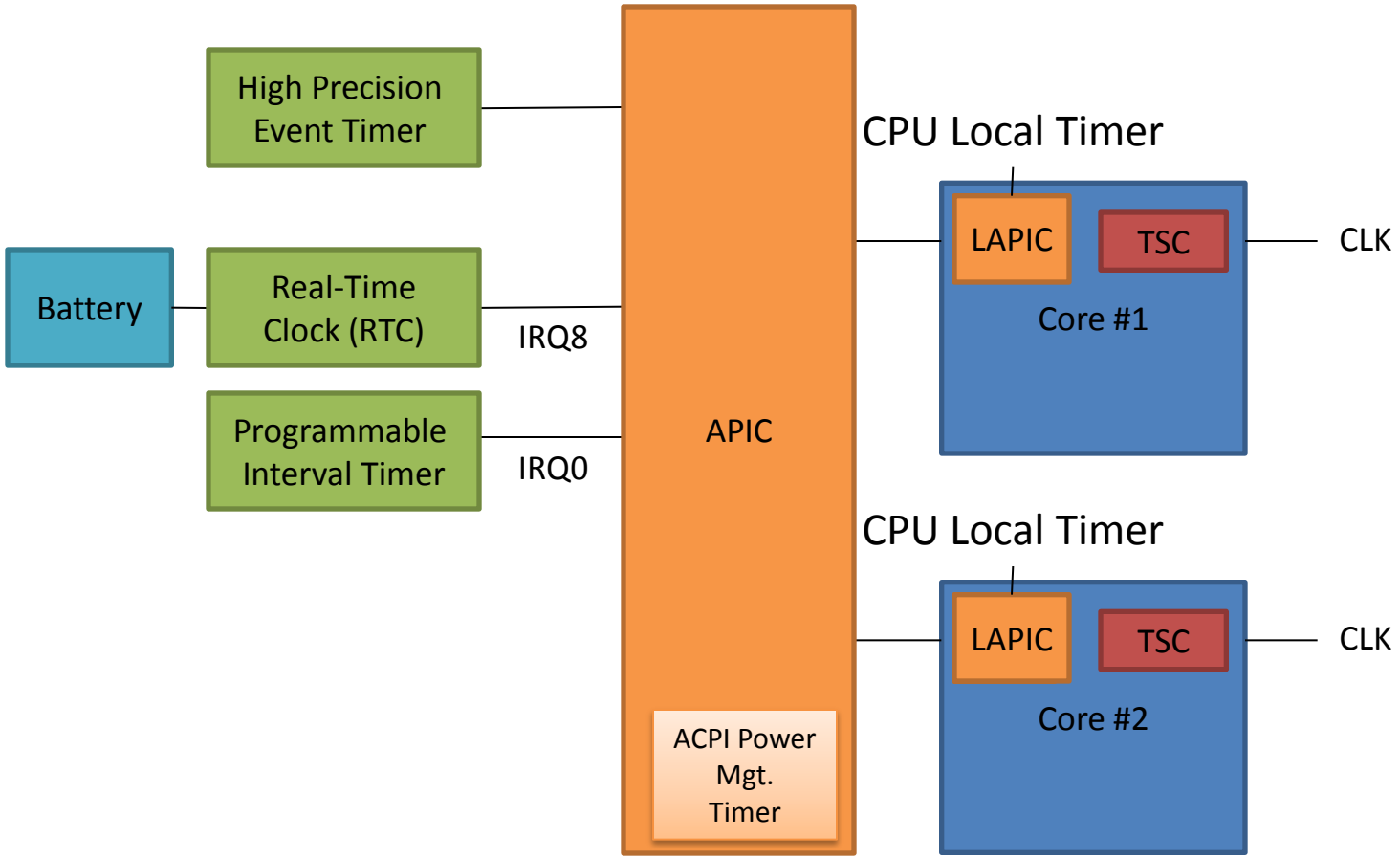
Overall HW architecture

- Single core

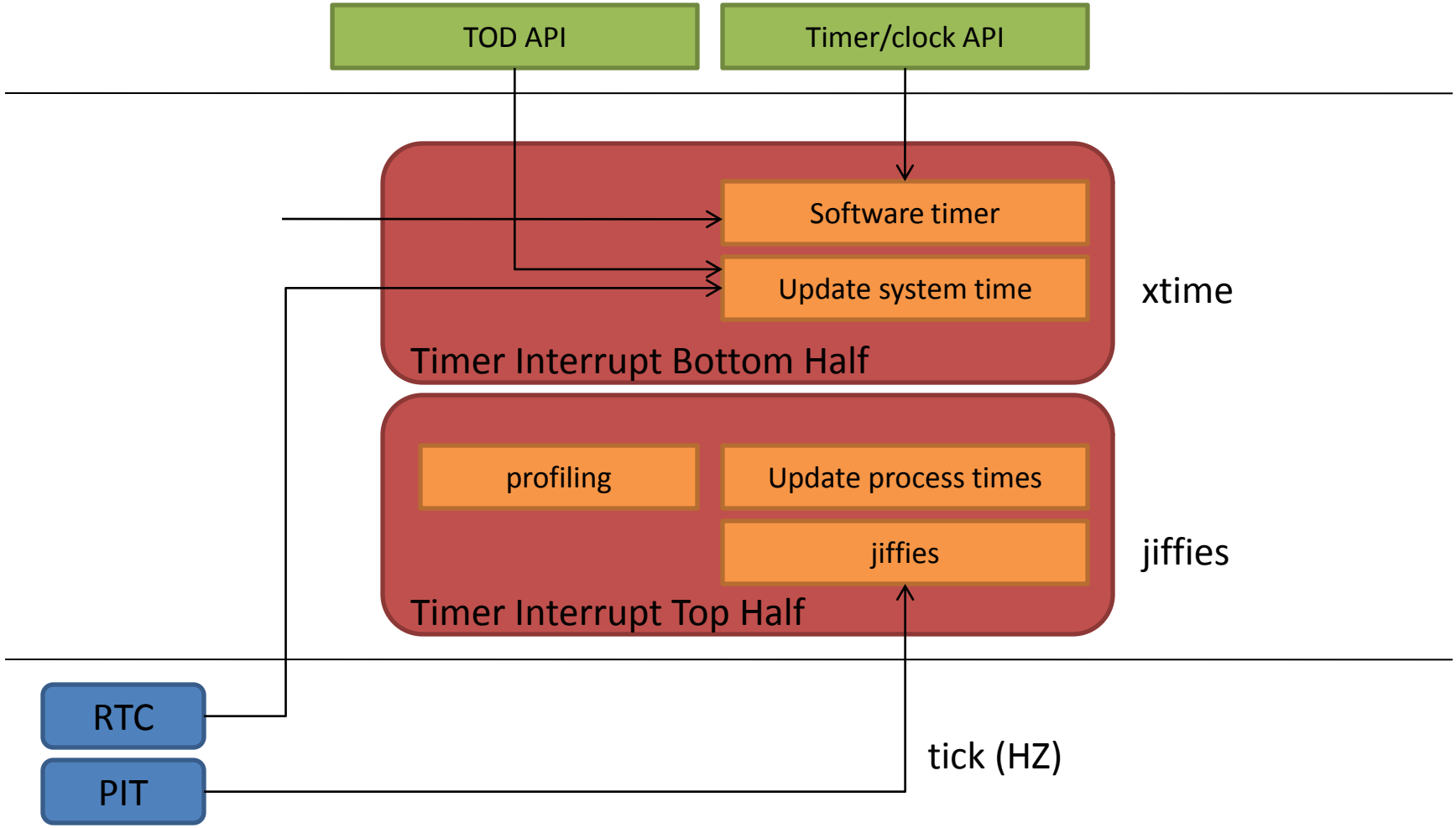


Overall HW architecture

- Multi-core



SW architecture



Terms

- HZ: number of ticks per second
  - ARM: 100
  - X86: 1000
- Jiffies: number of ticks that have occurred since the system booted
- xtime: current time of day (the wall time)

Initial procedure

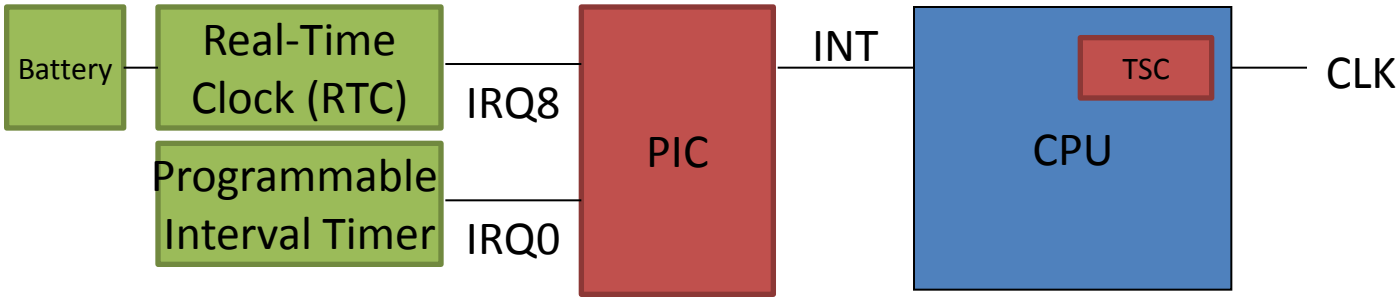
OS

3. **Set PIT/IRQ**

2. **Run time_init()/Retrieve RTC**

HW

1. **Booting/Retrieve RTC**



Top half procedures

4. update_times

3. update_process_times

2. increment jiffies

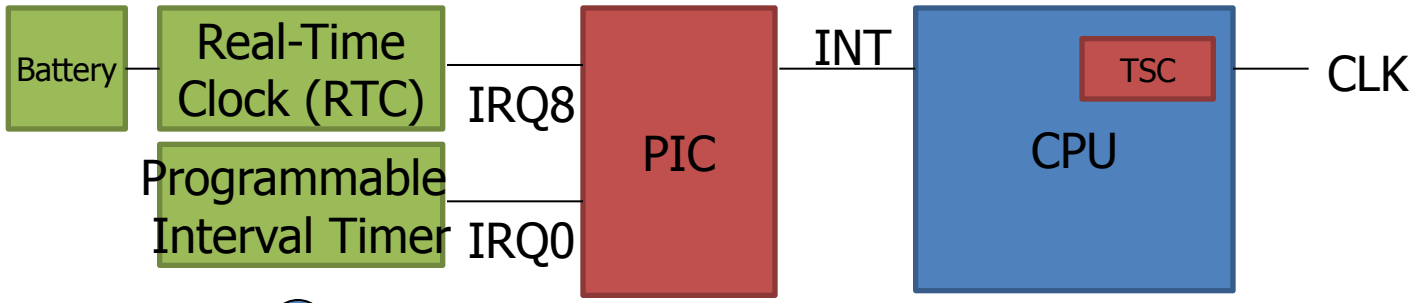
3.3. scheduler_tick: decrements running process's timeslice and sets need_resched

3.2. run_local_timers: marks a softirq to handle expired timers

3.1. update_one_process

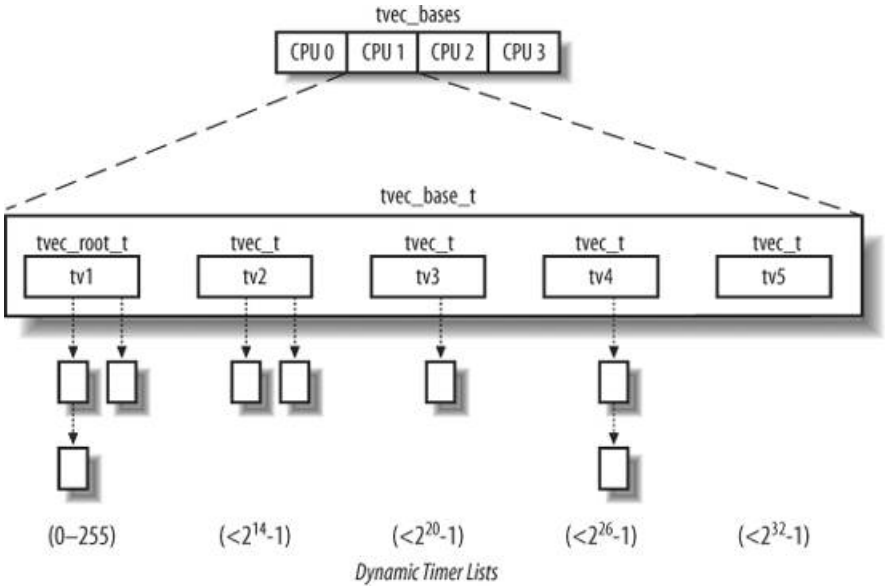
OS

HW



1. PIT interrupt occurs (1ms)

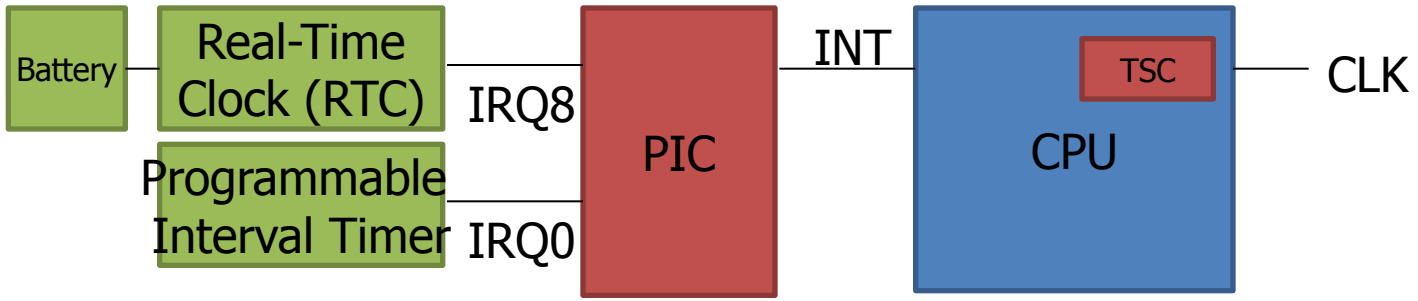
Bottom half procedures



2. **Check software timer list**
1. **Update system date/time (xtime)**

OS

HW



Long delays

- Delays in the order of jiffies
- Busy waiting

```
unsigned long timeout = jiffies + HZ; while
(time_before(jiffies, timeout)) continue;
```

- Sleep waiting (wait_event_timeout/msleep)

```
unsigned long timeout = jiffies + HZ;
schedule_timeout(timeout);
```

```
while (time_before(jiffies, delay))
cond_resched();
```

```
signed long schedule_timeout(signed long timeout)
{
        timer_t timer;
        unsigned long expire;

        switch (timeout)
        {
        case MAX_SCHEDULE_TIMEOUT:
                schedule();
                goto out;
        default:
                if (timeout < 0)
                {
                        printk(KERN_ERR "schedule_timeout: wrong timeout "
                            "value %lx from %p\n", timeout,
                            __builtin_return_address(0));
                        current->state = TASK_RUNNING;
                        goto out;
                }
        }

        expire = timeout + jiffies;

        init_timer(&timer);
        timer.expires = expire;
        timer.data = (unsigned long) current;
        timer.function = process_timeout;

        add_timer(&timer);
        schedule();
        del_timer_sync(&timer);
```

Long delays

- Using kernel timer

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

struct timer_list my_timer;

init_timer(&my_timer);            /* Also see setup_timer() */
my_timer.expire = jiffies + n*HZ; /* n is the timeout in number
                                     of seconds */
my_timer.function = timer_func;   /* Function to execute
                                     after n seconds */
my_timer.data = func_parameter;   /* Parameter to be passed
                                     to timer_func */
add_timer(&my_timer);             /* Start the timer */


static void timer_func(unsigned long func_parameter)
{
  /* Do work to be done periodically */
  /* ... */

  init_timer(&my_timer);
  my_timer.expire   = jiffies + n*HZ;
  my_timer.data     = func_parameter;
  my_timer.function = timer_func;
  add_timer(&my_timer);
}
```

Long delays

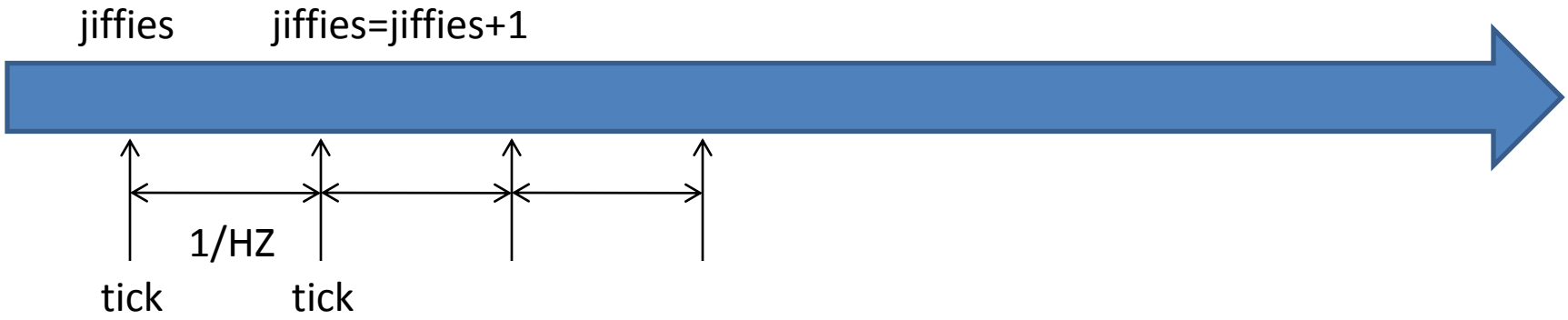
- Interrupt handler: busy waiting (avoid)
- Process context: sleep waiting

Short delays

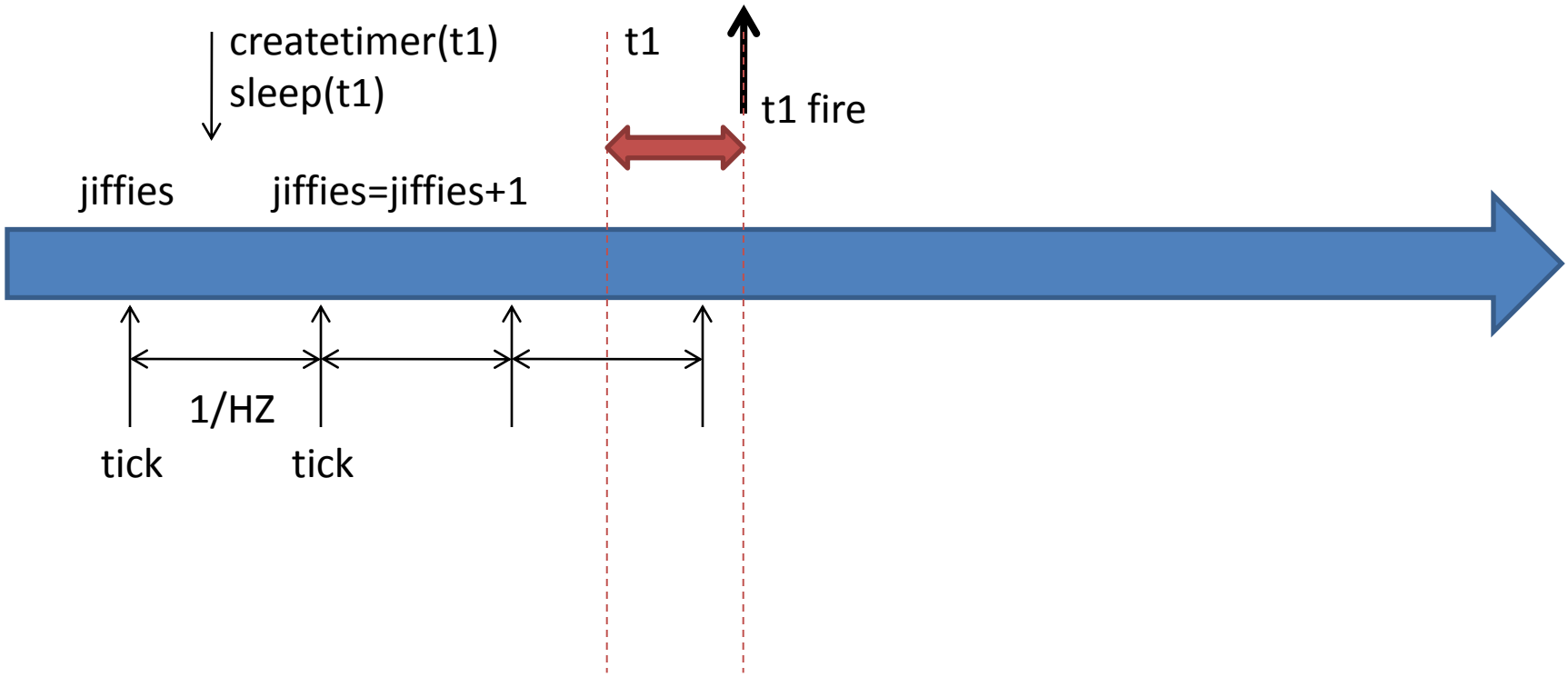
- sub-jiffy delays
- mdelay(), udelay(), and ndelay()
- actual implementations may not be available on all platforms

```
do {
  result = ehci_readl(ehci, ptr);
  /* ... */
  if (result == done) return 0;
  udelay(1);      /* Internally uses loops_per_jiffy */
  usec--;
} while (usec > 0);
```

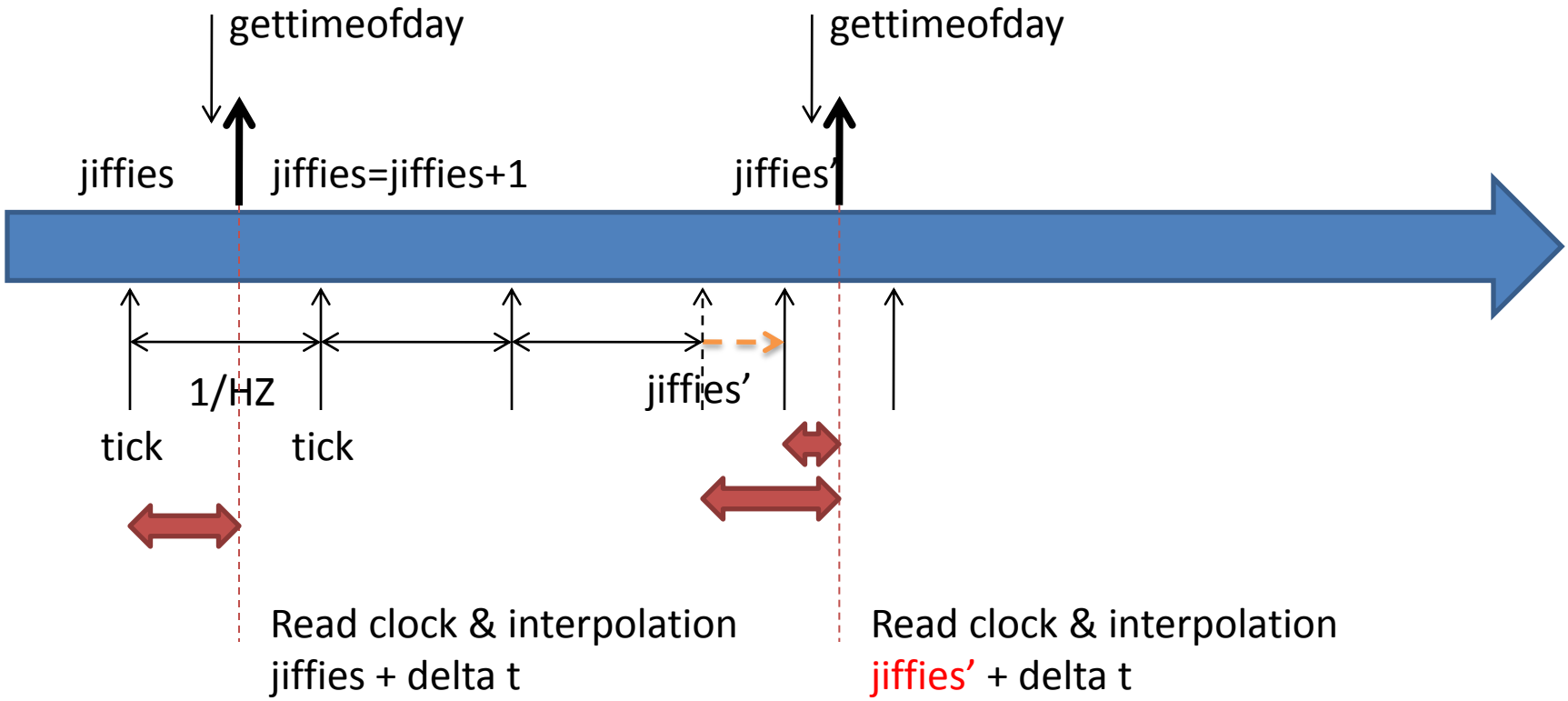
Traditional Timer vs. High Resolution Timer



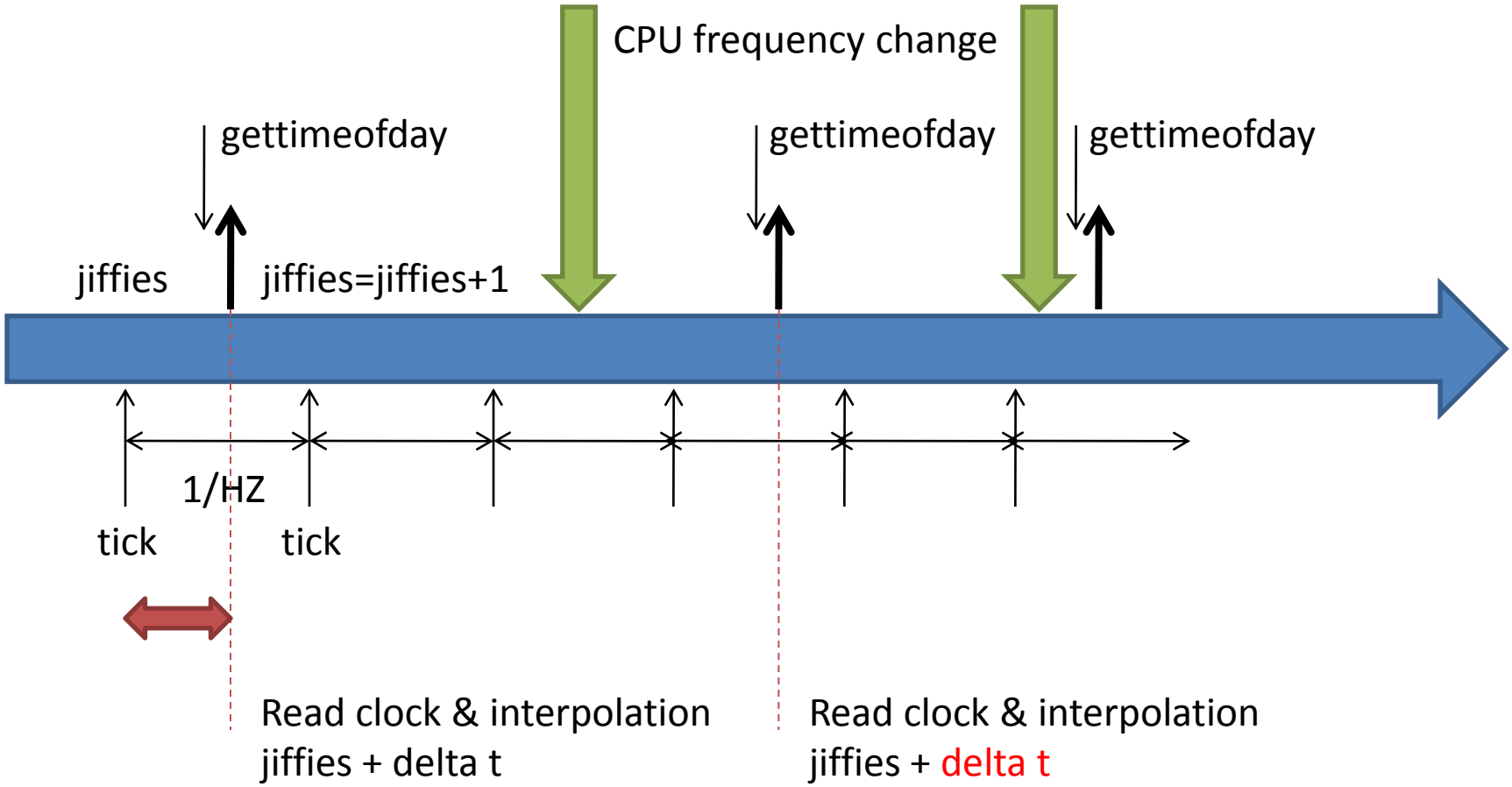
Traditional Timer vs. High Resolution Timer



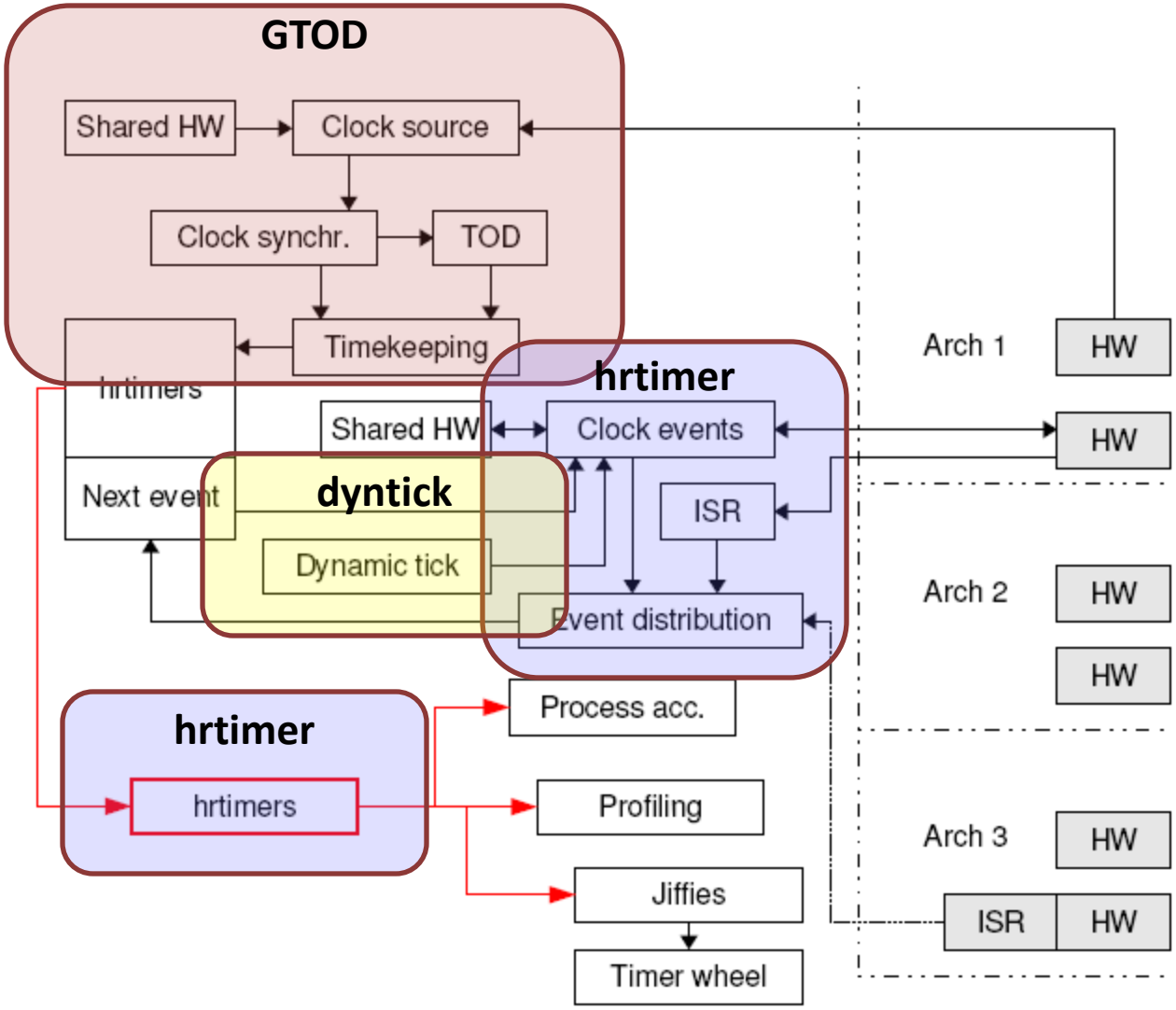
Traditional Timer vs. High Resolution Timer



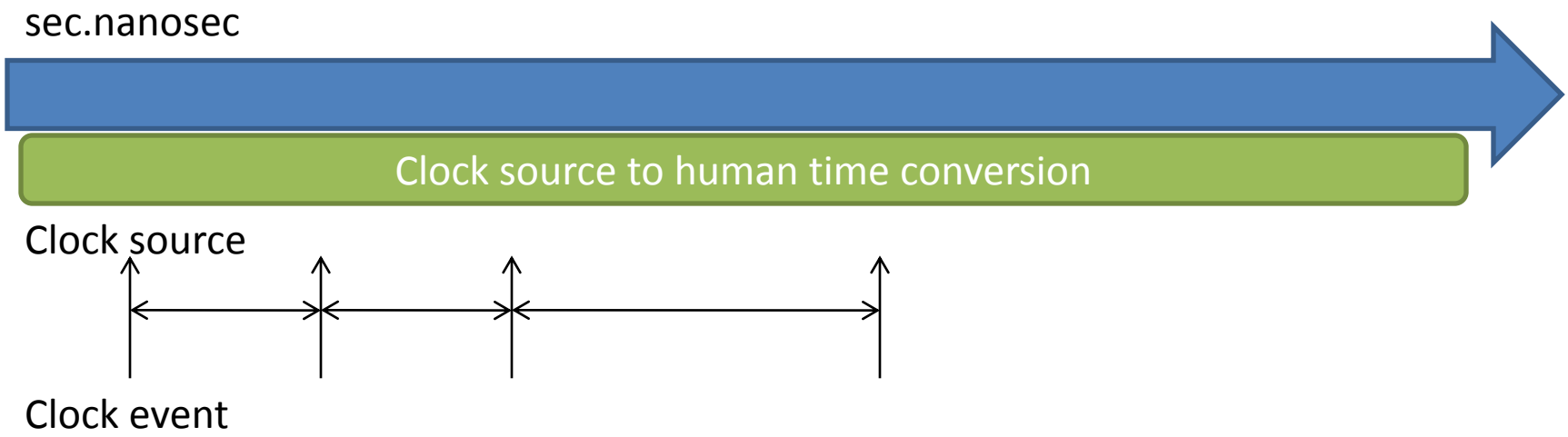
Traditional Timer vs. High Resolution Timer



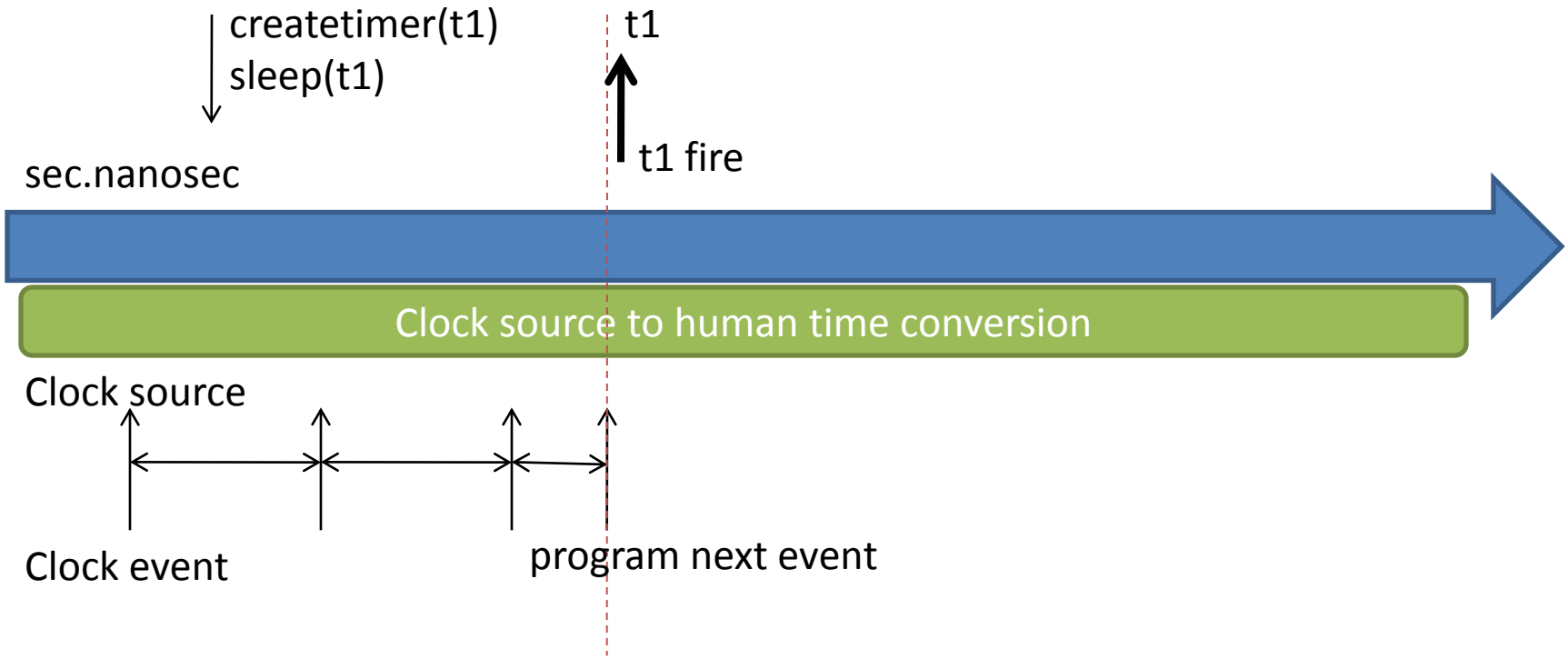
New Linux timer architecture



Traditional Timer vs. **High Resolution Timer**



Traditional Timer vs. **High Resolution Timer**



Traditional Timer vs. **High Resolution Timer**