Input/output

Sistemi Operativi
Lez. 12

Ruolo del SO

- Le periferiche di I/O sono i dispositivi attraverso i quali un calcolatore scambia dati/ interagisce con la realtà esterna
- Per ogni periferica collegata ad un calcolatore il SO deve essere in grado di
  - Inviare comandi alla periferica
  - Acquisire dati dalla periferica (periferica di input)
  - Inviare dati alla periferica (periferica di output)
  - Gestire errori
- Per capire come possono essere realizzate queste funzionalità è necessario conoscere i principi di funzionamento di una periferica

Velocità di trasferimento

I/O Devices Enable Interacting with Environment

Device	Behavior	Partner	Data Rate (b/sec)
Keyboard	Input	Human	100
Mouse	Input	Human	3.8K
Sound Input	Input	Machine	3M
Voice Output	Output	Human	264K
Sound Output	Output	Human	8M
Laser Printer	Output	Human	3.2M
Graphics Display	Output	Human	800M–8G
Network/LAN	Input/Output	Machine	100M–10G
Network/Wireless LAN	Input/Output	Machine	11–54M
Optical Disk	Storage	Machine	5–120M
Flash Memory	Storage	Machine	32–200M
Magnetic Disk	Storage	Machine	800M–3G

Dispositivi di I/O

- I dispositivi di I/O hanno:
  - Una componente meccanica
  - Una componente elettronica
- La componente elettronica chiamata device controller impartisce comandi al dispositivo fisico (drive/device) e ne controlla la corretta esecuzione
- L'insieme dei comandi impartiti dal controller al device, il loro formato, i codici di errore riportati definiscono l'interfaccia tra il device ed il dispositivo
- Un controller può controllare più device purché abbiano la stessa interfaccia
- Interfacce famose: IDE (Integrated Device electronics, SCSI (Small Computer System Interface), USB (Universal Serial BUS)

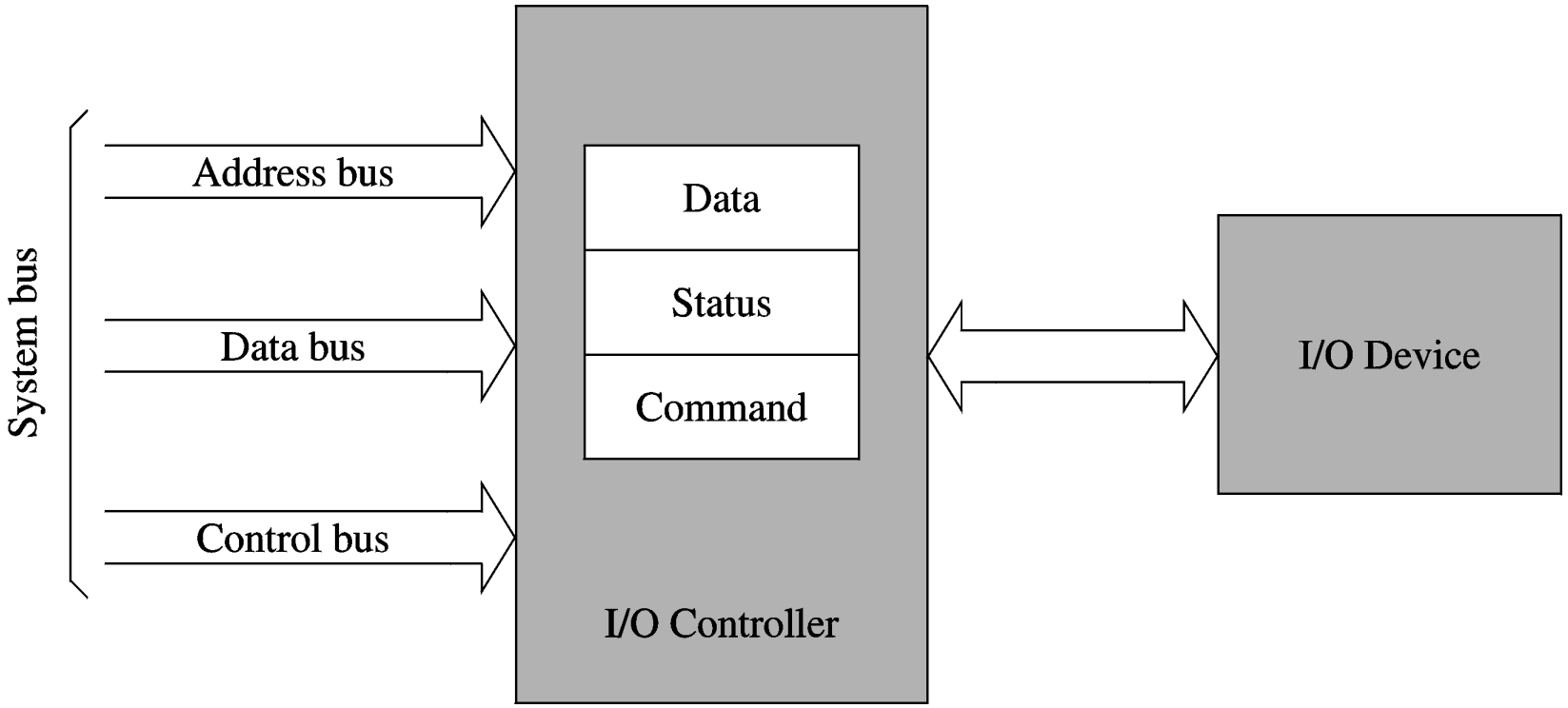
Tipi di Device

- I dispositivi possono essere suddivisi in due grosse categorie in funzione delle modalità con cui “manipolano” i dati su cui operano:
  - Dispositivi a blocchi: memorizzano e trasferiscono le informazioni in blocchi di dimensione fissa, e ciascun con un suo indirizzo. La dimensione del blocco varia da 512-32,768 byte
  - Dispositivi a carattere: memorizzano e trasferiscono stringhe di byte senza riferimento ad alcuna struttura di blocco
- Il clock è un dispositivo di I/O che non ricade in nessuna delle due predette categorie

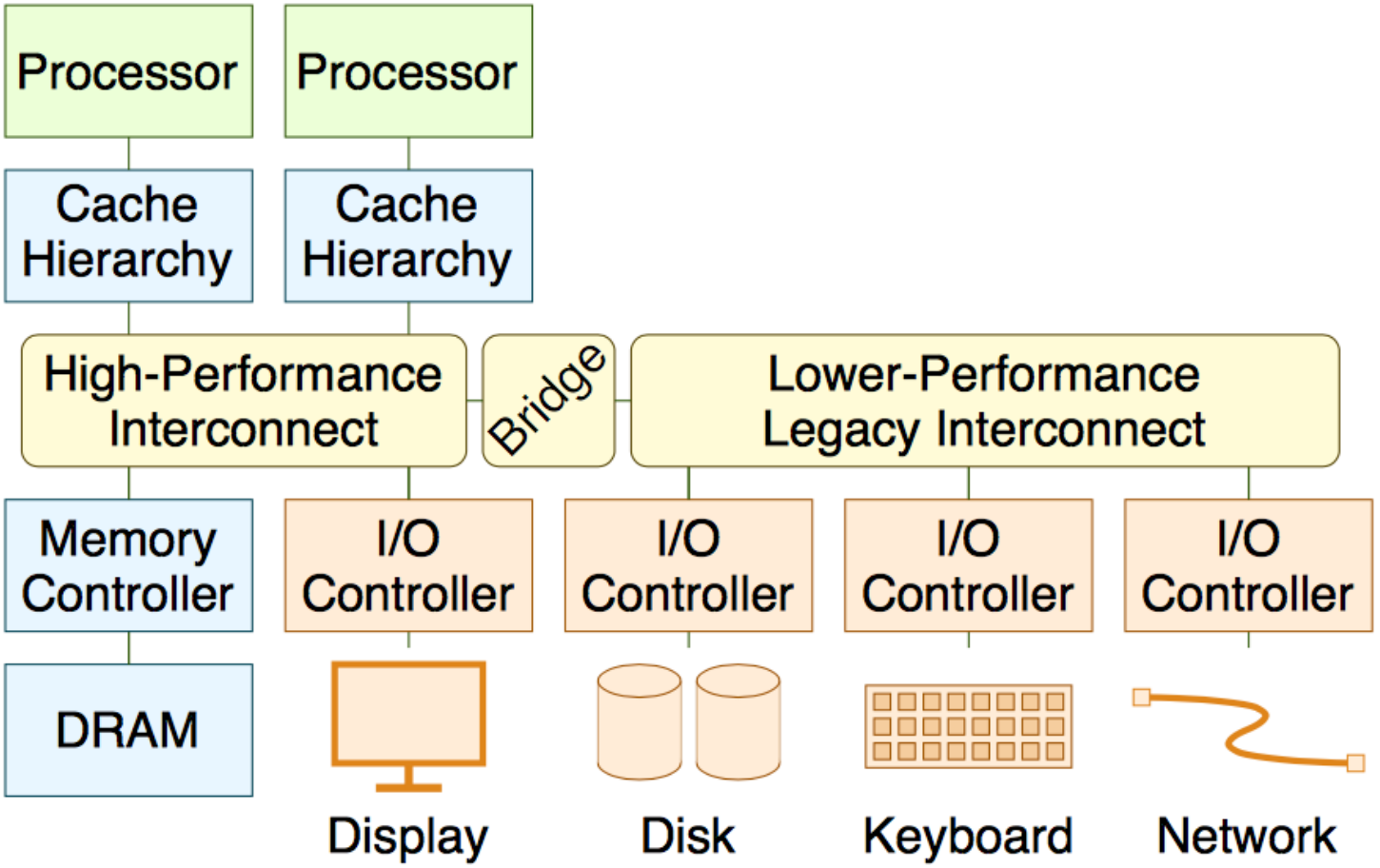
Device Controller

- Interposti tra il SO e la periferica, offrono un'interfaccia di comunicazione tra i due "mondi"
- Possono essere programmati usando appositi registri ed in alcuni casi data buffer
- Caricando opportune sequenze di bit all'interno di questi registri, il sistema operativo può richiedere l'esecuzione di comandi il cui risultato può essere prelevato sempre attraverso questi registri
- L'insieme di comandi usati per comunicare con il controller, il loro formato, ed i codici di errore costituiscono l'interfaccia del controller verso il sistema operativo, questa interfaccia è device-dependent

Device Controller



A Typical PC Bus Structure



Comunicare con il controller

- Al controller vanno impartiti ordini specifici
- Come fa un programmatore a svolgere questa operazione? I problemi da affrontare sono due:
  - Accedere ai registri del controller
    - dipende dalle modalità di I/O mapping
      - » Memory-mapped I/O
      - » Isolated I/O
  - Sincronizzarsi con un controller
    - Programmed I/O & Polling
    - Direct memory access (DMA)
    - Interrupt-driven I/O

Memory mapped I/O

- I controller condividono lo spazio fisico della memoria, cioè ai registri dei controller sono assegnate determinate locazioni di memoria centrale
- Le operazioni di I/O sono le stesse operazioni usate per la gestione delle memoria (LW,SW)
- Non esiste alcuna istruzione particolare di I/O
- A livello hw è necessario prevedere degli opportuni meccanismi (I/O bus, linee dedicate) per discernere gli indirizzi di I/O da quelli relativi alla memoria

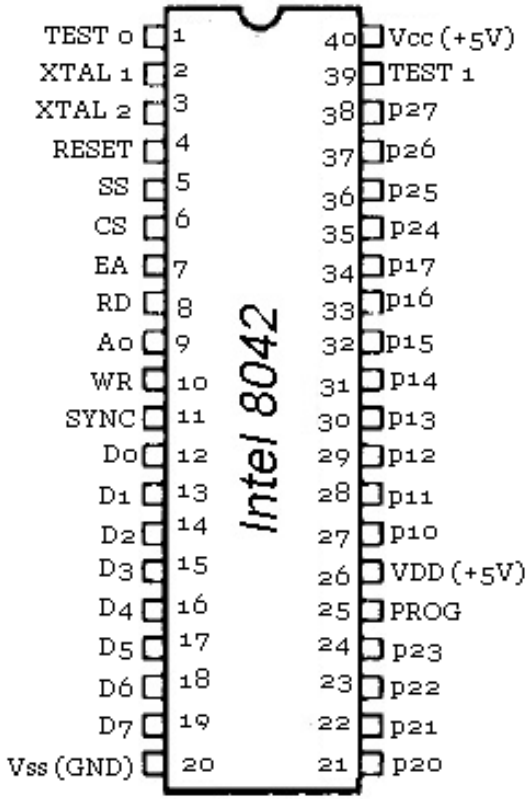
Isolated I/O

- Le spazio degli indirizzi dei controller è diverso da quello della memoria centrale
- Si adottano istruzioni di I/O dedicate alla manipolazione dei registri del controller

I/O in INTEL: sistema ibrido

- I/O port: ad ogni registro è assegnato un numero di 8/16 bit, che specifica la porta di I/O corrispondente a cui ci si riferisce usando istruzioni dedicate di I/O
  - Es. `movb $0x20, %al`
    `outb %al, 0x20`
- Sistema ibrido:
  - Buffer dati memory mapped
  - Registri di controllo come I/O port

Keyboard Controller



Pins **P10-P17** is the controllers **input port**.
Pins **P20-P27** is the controllers **output port**. And pin **T0-T1** is the controller **test port**.
The exact meaning of these pins depend on the mode of operation the controller is in.

Keyboard Registers

- I registri per la gestione della tastiera sono:
  - Input buffer (8bit- read only) – contiene i dati provenienti dalla tastiera;
  - Output buffer (8bit – write only) – contiene i dati (tipicamente comandi) che devono essere inviati alla tastiera
  - Status register – 8bit status flags; read-only
  - Command register – 8bit read/write
- La prima coppia di registri è presente su un controller della Keyb l'altra sull'8042 (di fatto ci sono due controller per la tastiera)

Keyboard Port

Keyboard Controller Ports		
Port	Read/Write	Descripton
Keyboard Encoder		
0x60	Read	Read Input Buffer
0x60	Write	Send Command
Onboard Keyboard Controller		
0x64	Read	Status Register
0x64	Write	Send Command

Registri keyb

- Status Register: fornisce informazioni sullo stato della tastiera
- IN_BUF: può essere usato per leggere:
  - Scancode
  - Valori di ritorno di operazioni svolte
  - Massaggi per il protocollo di comunicazione tra 8042 e tastiera: ACK, Resend, ecc
- OUT_BUF: può essere usato per scrivere:
  - Comandi alla keyb attraverso l'indirizzo 0x60
  - Argomenti dei comandi attraverso l'indirizzo 0x60
  - Comandi al controller attraverso l'indirizzo 0x64

Status register

Status Register:

The 8042's status flags are read from port 0x64. They contain error information, status information, and indicate whether or not data is present in the input and output buffers. The flags are defined as follows:

	MSb							LSb
AT-compatible mode:	PERR	RxTO	TxTO	INH	A2	SYS	IBF	OBF
PS/2-compatible mode:	PERR	TO	MOBF	INH	A2	SYS	IBF	OBF

- OBF (Output Buffer Full) - Indicates when it's okay to write to output buffer.

  0: Output buffer empty - Okay to write to port 0x60
  1: Output buffer full - Don't write to port 0x60
- IBF (Input Buffer Full) - Indicates when input is available in the input buffer.

  0: Input buffer empty - No unread input at port 0x60
  1: Input buffer full - New input can be read from port 0x60
- SYS (System flag) - Post reads this to determine if power-on reset, or software reset.

  0: Power-up value - System is in power-on reset.
  1: BAT code received - System has already beed initialized.

Leggere un dato da tastiera (con Int. disabilitati)

- Quando la tastiera riceve un dato, il relativo codice (scan code) viene inserito nel buffer di input ed il flag IBF viene settato a 1. La tastiera non acquisirà più alcun dato sino a che il buffer non sarà svuotato
- Per acquisire il dato letto è quindi necessario testare il valore di IBF e quando questi è == 1 acquisire il dato dal buffer di input

```
Waitloop: inb 0x64,%al
          and $0x02,%al
          jz  Waitloop
          inb 0x60,%al
```

Inviare comandi al controller

- Il sistema operativo può inviare un comando al controller attraverso la porta 0x64, prima di inviare il comando è però necessario verificare che il buffer di input sia vuoto e quindi IBF == 0

-
```
Waitloop: inb  0x64,%al
          and  $0x02,%al
          jnz  Waitloop
          movb $0xAD,%al
          outb %al,0x64
```

Comandi controller

Command Listing	
Command	Descripton
Common Commands	
0x20	Read command byte
0x60	Write command byte
0xAA	Self Test
0xAB	Interface Test
0xAD	Disable Keyboard
0xAE	Enable Keyboard
0xC0	Read Input Port
0xD0	Read Output Port
0xD1	Write Output Port
0xE0	Read Test Inputs
0xFE	System Reset
0xA7	Disable Mousr Port
0xA8	Enable Mouse Port
0xA9	Test Mouse Port
0xD4	Write To Mouse
Non Standard Commands	
0x00-0x1F	Read Controller RAM
0x20-0x3F	Read Controller RAM
0x40-0x5F	Write Controller RAM

Maschere per status register

```
enum KYBRD_CTRL_STATS_MASK {

    KYBRD_CTRL_STATS_MASK_OUT_BUF =   1,     //00000001
    KYBRD_CTRL_STATS_MASK_IN_BUF  =   2,     //00000010
    KYBRD_CTRL_STATS_MASK_SYSTEM  =   4,     //00000100
    KYBRD_CTRL_STATS_MASK_CMD_DATA    =   8, //00001000
    KYBRD_CTRL_STATS_MASK_LOCKED  =   0x10, //00010000
    KYBRD_CTRL_STATS_MASK_AUX_BUF =   0x20, //00100000
    KYBRD_CTRL_STATS_MASK_TIMEOUT =   0x40, //01000000
    KYBRD_CTRL_STATS_MASK_PARITY  =   0x80  //10000000
};
```

Leggi status register

```
enum KYBRD_ENCODER_IO {
    KYBRD_ENC_INPUT_BUF =   0x60,
    KYBRD_ENC_CMD_REG   =   0x60
};
enum KYBRD_CTRL_IO {
    KYBRD_CTRL_STATS_REG    =   0x64,
    KYBRD_CTRL_CMD_REG  =   0x64
};
//! read status from keyboard controller
uint8_t kybrd_ctrl_read_status ()
{
    return inportb (KYBRD_CTRL_STATS_REG);
}
```

Send command

```
/! send command byte to keyboard controller

void kybrd_ctrl_send_cmd (uint8_t cmd)
{
//! wait for kkybrd controller input buffer to be clear

while (1)
    if ( (kybrd_ctrl_read_status () &
          KYBRD_CTRL_STATS_MASK_IN_BUF) == 0)
              break;
outportb (KYBRD_CTRL_CMD_REG, cmd);
}
```

Read data

```
/! Read input buffer from keyboard

void kybrd_read_data (uint8_t data)
{
//! wait for kkybrd controller input buffer to be clear

while (1)
if ( (kybrd_ctrl_read_status () &
       KYBRD_CTRL_STATS_MASK_IN_BUF) == 0)
          break;
data = inportb (KYBRD_CTRL_STATS_REG);
}
```

I/O port INTEL

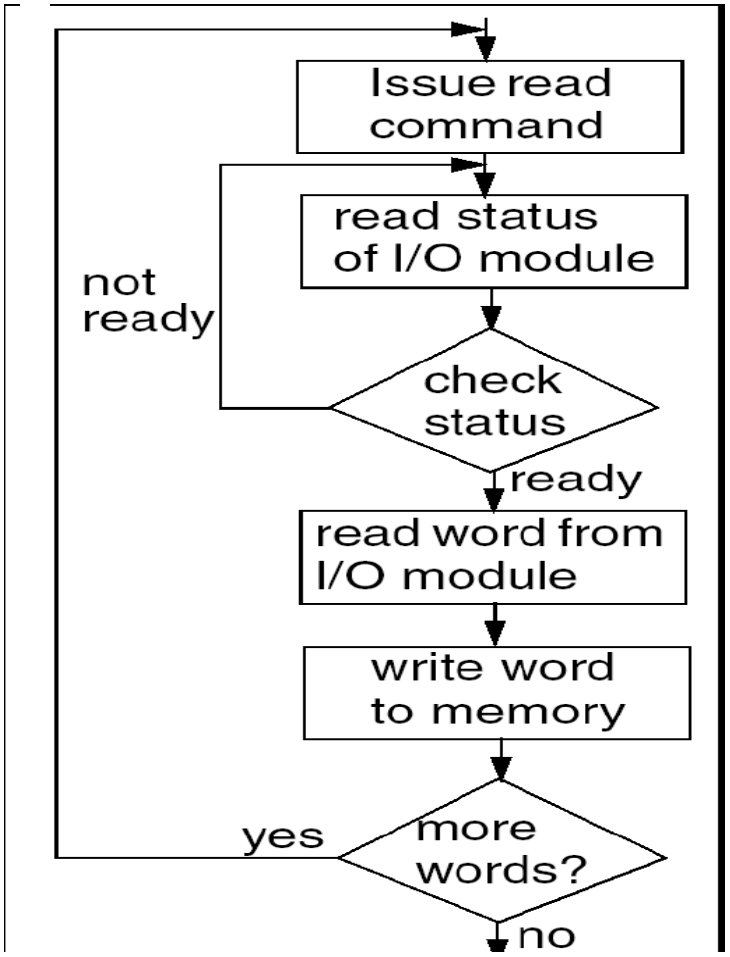
I/O address range (hexadecimal)	device
000-00F	DMA controller
020-021	interrupt controller
040-043	timer
200-20F	game controller
2F8-2FF	serial port (secondary)
320-32F	hard-disk controller
378-37F	parallel port
3D0-3DF	graphics controller
3F0-3F7	diskette-drive controller
3F8-3FF	serial port (primary)

Modalità di comunicazione tra controller e driver

Programmed I/O & Polling

- *Periodic Polling*: in alcuni casi, il driver verifica periodicamente se un dispositivo è pronto ad operare, in caso affermativo procede altrimenti si blocca e ripete l'operazione ad intervalli di tempo prefissati
  - Es: un mouse USB deve essere "interrogato" con una frequenza di 100 Hz
- *Continuous polling:* all'interno del driver è presente un loop di busy waiting che continuamente verifica se il dispositivo è pronto o meno ad accettare dati o comandi, oppure a produrre un output

Continuous Polling (Controllo Programma)



Interrupt Driven I/O

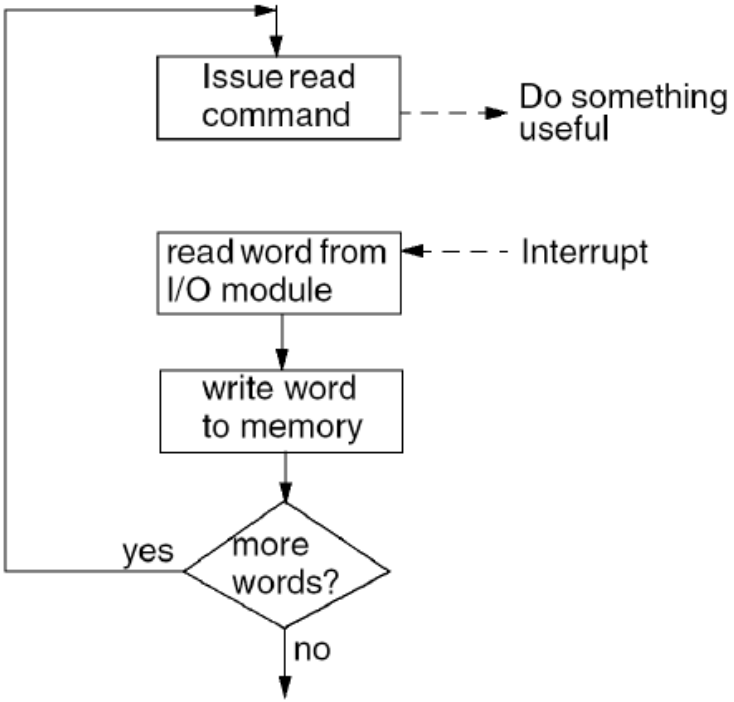
- I registri di controllo di un device controller posseggono almeno uno status bit che indica se un'operazione di ouput è stata completata oppure se c'è un nuovo dato da consumare
- Nelle periferiche più sofisticate insieme a questi bit viene asserita una linea di IRQ sul bus di sistema
- Quindi ogni volta che si inserisce un nuovo controller su un sistema va indicato l'IRQ di riferimento, per evitare conflitti questa scelta viene spesso demandata al BIOS in fase di boot

Keyboard interrupt

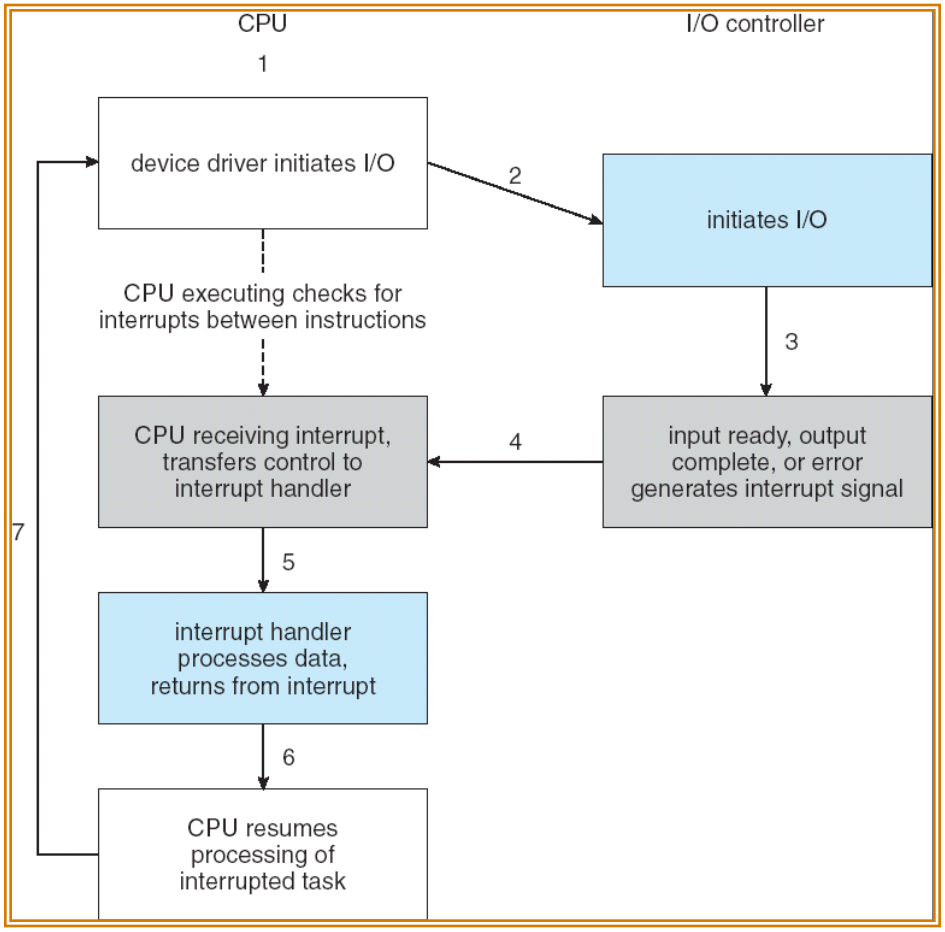
When the 8042 recieves a valid scan code from the keyboard, it is converted to its set 1 equivalent. The converted scan code is then placed in the input buffer, the IBF (Input Buffer Full) flag is set, and IRQ 1 is asserted. Furthermore, when any byte is received from the keyboard, the 8042 inhibits further reception (by pulling the "Clock" line low), so no other scan codes will be received until the input buffer is emptied.

If enabled, IRQ 1 will activate the keyboard driver, pointed to by interrupt vector 0x09. The driver reads the scan code from port 0x60, which causes the 8042 to de-assert IRQ 1 and reset the IBF flag. The scan code is then processed by the driver, which responds to special key combinations and updates an area of the system RAM reserved for keyboard input.

Interrupt driven I/O



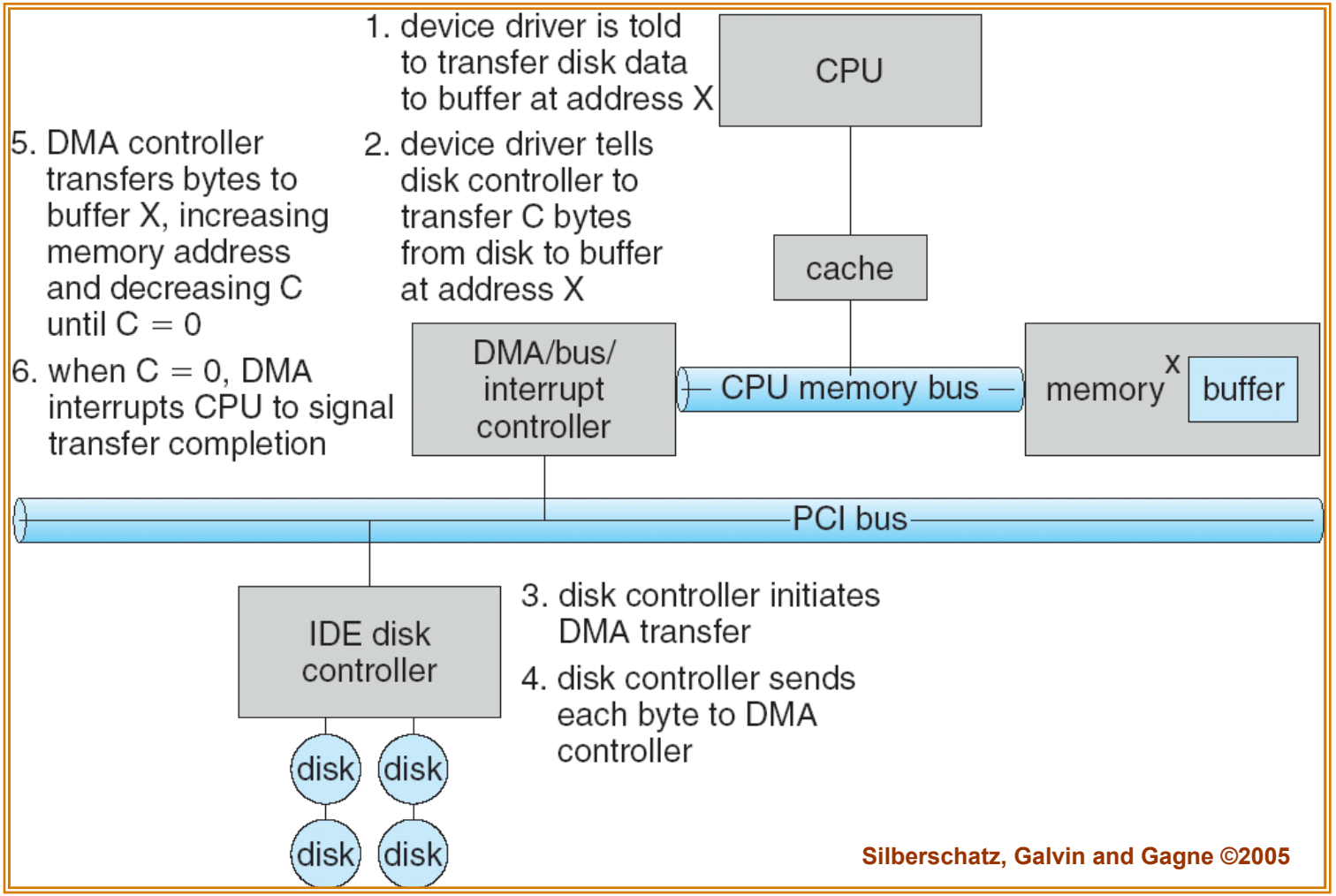
Interrupt-driven I/O



DMA-CPU

- La CPU (il driver) programma il DMA controller, predisponendone i registri con i seguenti dati:
  - indirizzo di memoria da dove prelevare o depositare i dati;
  - Numero di byte da acquisire/prelevare
  - Informazione di controllo (read/write)
- Dopodiché riprende la normale operatività
- Il DMA controller prende il controllo del memory bus direttamente e impartisce i comandi all' I/O controller per effettuare lo spostamento di dati richiesto dalla CPU
- Il Disk controller transferisce i dati in memoria principale
- Terminato il trasferimento il disk controller invia un ACK al DMA controller
- Il DMA controller invia un interrupt al processore per avvertirlo del completamento dell'operazione

Six Step Process to Perform DMA Transfer



DMA

- PRO
  - Allegerisce la CPU dall'overhead imposto dalla gestione degli interrupt nel caso di periferiche veloci
- CONS
  - Il DMA è in genere più lento della CPU, in molti sistemi la CPU (più lenta) si trova spesso in attesa del DMA
  - Il DMA aumenta i costi del sistema

	Polled Waiting Loop	Interrupt-Driven	Direct Memory Access
Maximum Transfer Rate		Slowest	Fastest
Worst-Case Latency	Unpredictable		Best
Hardware Cost	Least	Low	Moderate
Software Complexity	Low	Moderate	Moderate

Polled Waiting Loops

- Test device status in a ***waiting loop*** before transferring each data byte.
- Maximum data rate: Time required to execute one iteration of the waiting loop plus the transfer.
- Latency: Time from device ready until the moment that the CPU transfers data. Unpredictable - no guarantee when the program will arrive at the waiting loop.

Estimating Performance

- Performance limited by memory bandwidth (bytes/second):
  - Typical memory cycle time $\cong$ 60 ns = 60×10^{-9} sec.
  - Determines how fast instructions can be fetched.
  - We ignore speedup due to cache, so estimate is pessimistic.

Maximum Data Rate
(Polled Waiting Loop)

- Time for one iteration of waiting loop plus subsequent I/O transfer:

  memory cycles: 300 ns

  I/O cycles: 60 ns

  total time: 360 ns

  maximum data rate = 1/360 ns per byte

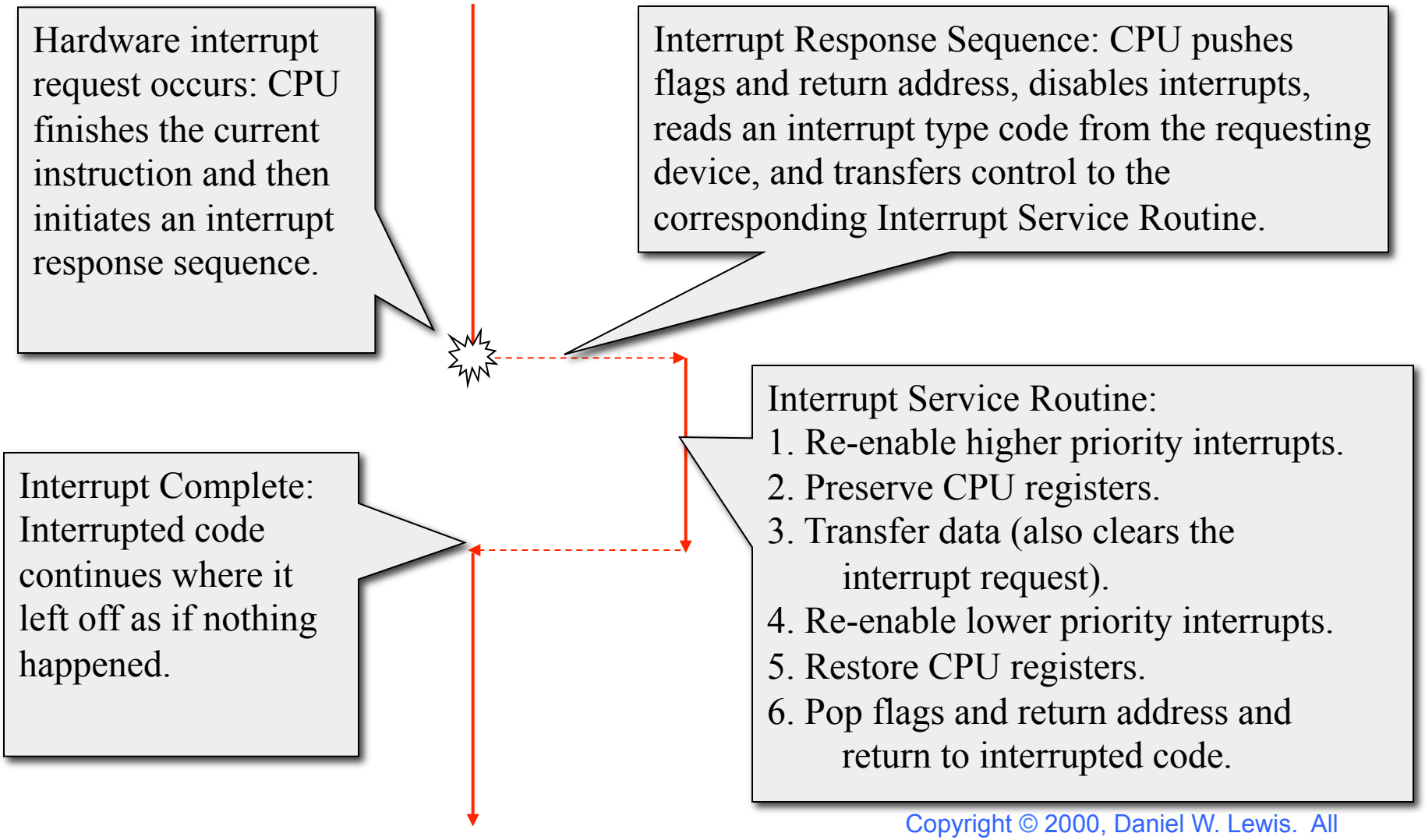
  = 2.78 MB/Sec

Fastest serial I/O:

= 115,000 bps

≅ 10 KB/Sec.

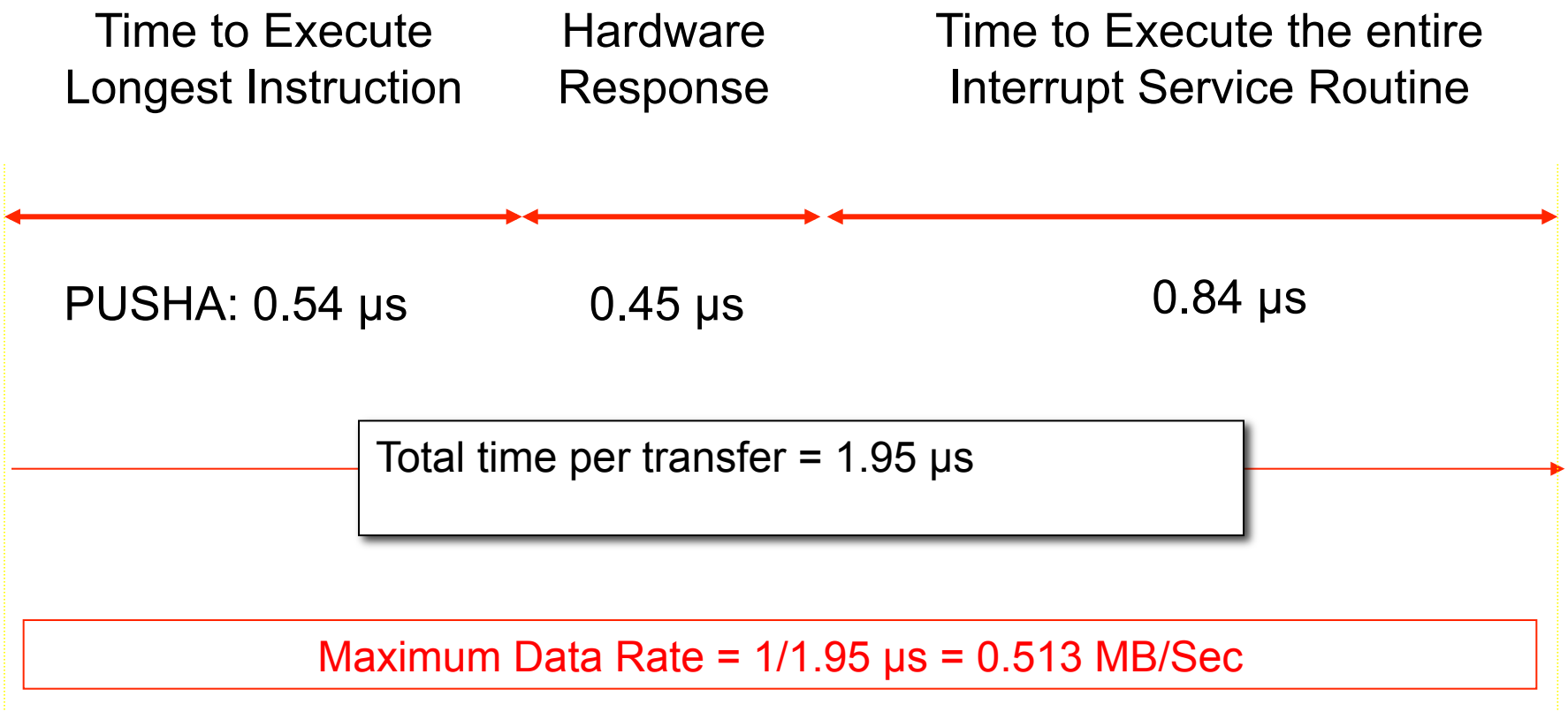
Interrupt-Driven I/O



Interrupt-Driven Latency

- Time to finish longest instruction
- Time of hardware response
- Time in ISR until data is transferred.

Total Time Per Byte



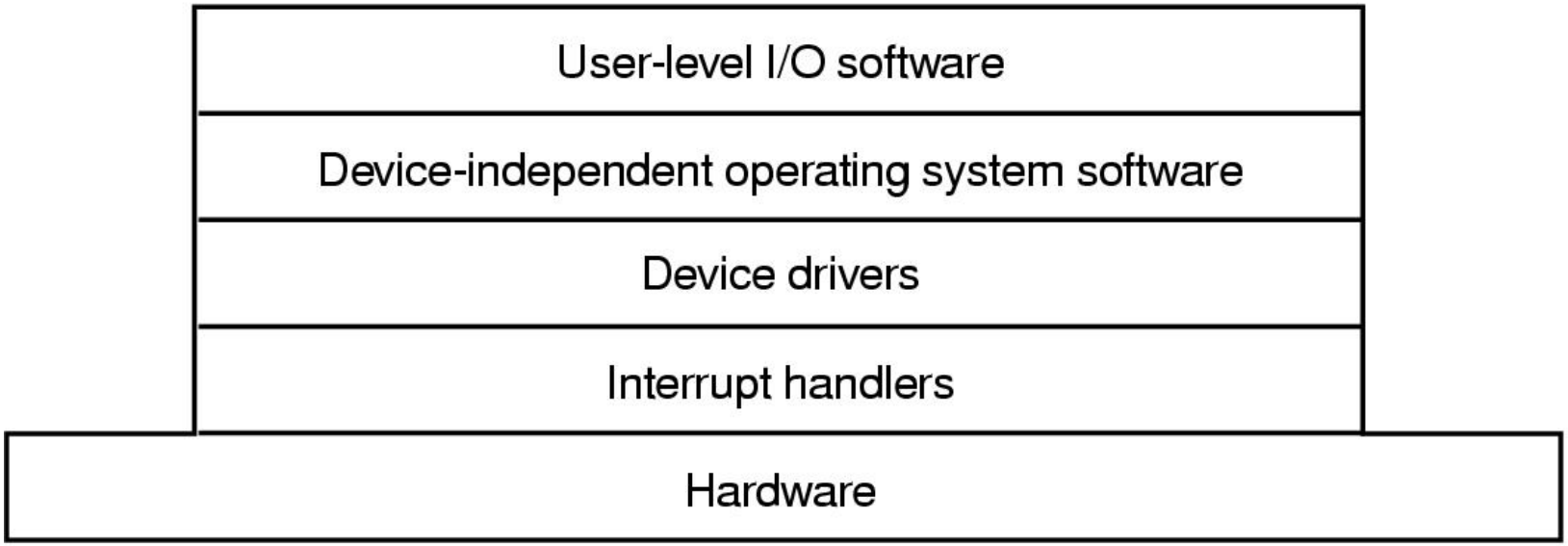
Estimate Summary

	Worst-Case Latency	Maximum Data Rate
Polled Waiting Loop	Unpredictable	2.78 MB/Sec
Interrupt-Driven	1.26 μs	0.513 MB/Sec
DMA	?	?

Estimate Summary

	Worst-Case Latency	Maximum Data Rate
Polled Waiting Loop	Unpredictable	2.78 MB/Sec
Interrupt-Driven	1.26 μs	0.513 MB/Sec
DMA	0.06 μs	66.7 MB/Sec

I/O sw system



- Il sottosistema per la gestione dell'I/O viene solitamente strutturato in 5 livelli, ciascuno formato da diverse componenti

Interrupt Handler

- Gli Interrupt handler sono la parte più nascosta dell'architettura e sono caratterizzati da stretti vincoli di tempo, eseguono quindi lo stretto necessario per
  - Acquisire i dati dell'ultima operazione di I/O
  - Predisporre le condizione per garantire l'esecuzione delle prossima operazione di I/O
  - Risvegliare il driver della periferica che stava attendendo il completamento dell'operazione di I/O

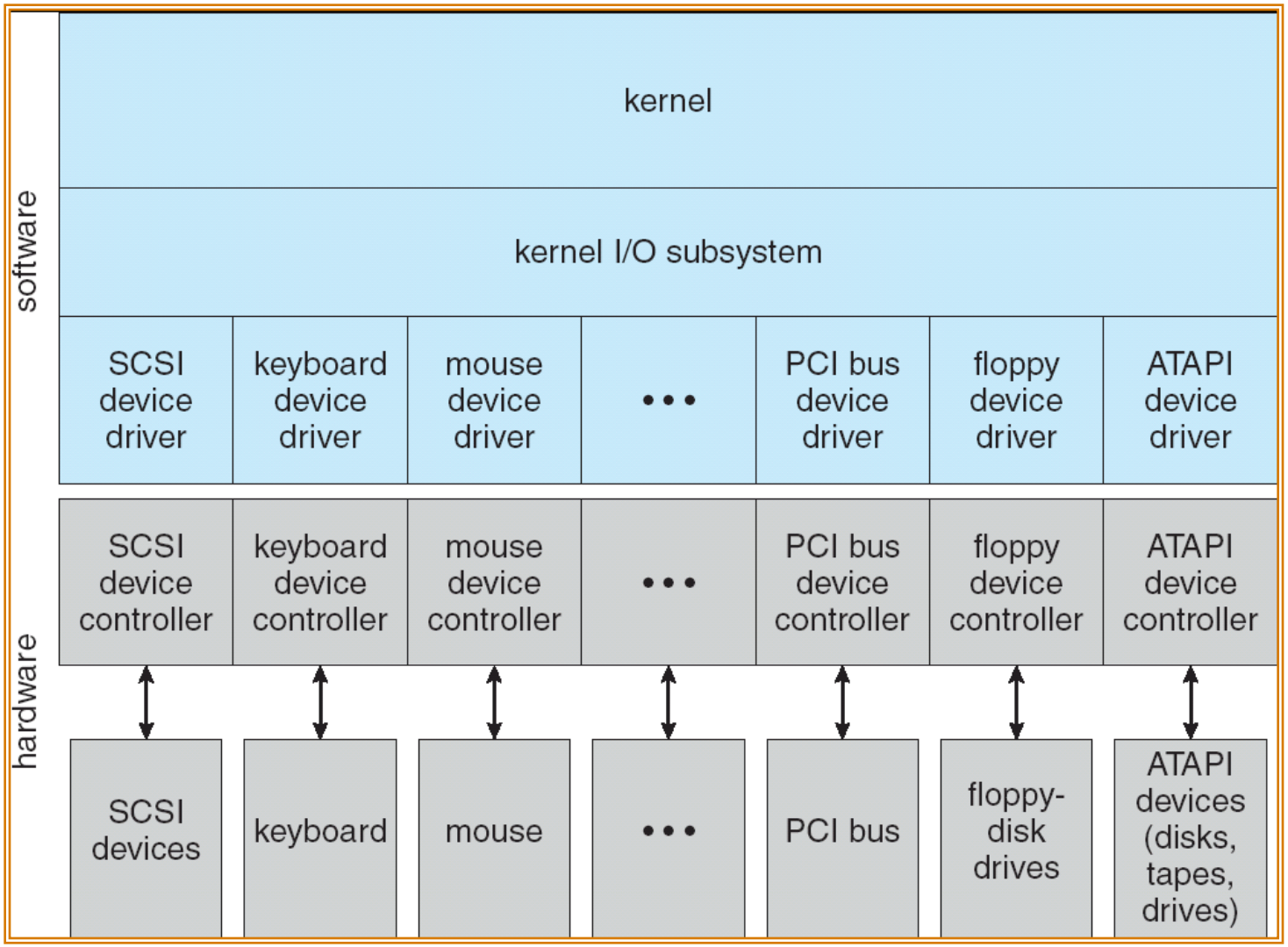
Device Driver

- Ogni periferica è caratterizzata da un proprio insieme di registri e di comandi, affinché la periferica funzioni correttamente è necessario che registri e comandi siano correttamente predisposti e impartiti
- Il device driver è un programma che svolge esattamente queste funzioni
- Esiste un device driver per ogni periferica solitamente realizzato dal costruttore della periferica stessa
- Solitamente concepito come parte del kernel, per consentire l'accesso rapido alle periferiche, in Minix i driver operano in user mode

Device driver

- Il driver costituisce quindi l'interfaccia tra il SO e il controller della periferica
- Riceve dal device independent sw la richiesta per l'esecuzione di comandi d'alto livello, che traduce in una sequenza di comandi per il controller
- Avvia il controller e resta in attesa di essere risvegliato da un interrupt, tramite l'interrupt handler

A Kernel I/O Structure

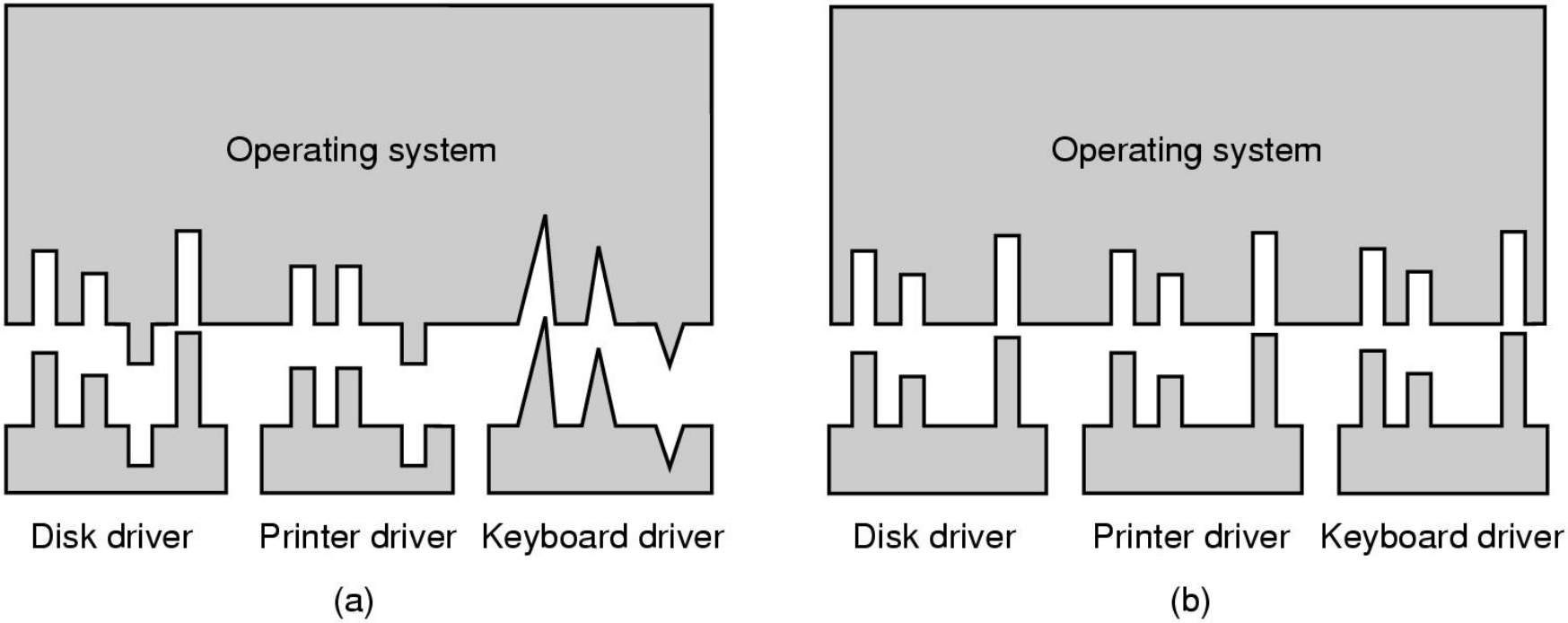


Device-Independent I/O Software

- Scopi:
  - Fornire un'interfaccia standard alle applicazioni per la gestione dell'I/O
  - fornire le funzionalità comuni a tutti i dispositivi di I/O

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicate devices
Providing a device-independent block size

Interfaccia verso i driver



- Interfaccia Driver-SO: le funzioni che il driver deve mettere a disposizione e le funzioni di kernel che può chiamare
- Un interfaccia uniforme facilita la stesura di driver, e rende possibile l'aggiunta di driver senza ricorrere alla modifica del kernel

Device-Independent I/O Software

- Uniform naming: si preoccupa di mappare il nome simbolico di un dispositivo nel corrispondente driver
- Buffering: si preoccupa di gestire la diversità tra i dati acquisiti da un dispositivo e i dati richiesti da un'applicazione
- Error Reporting: si preoccupa di fornire un'interfaccia comune verso i diversi messaggi di errore che possono derivare dai driver, a seguito di errori nei dispositivi

Device-Independent I/O Software

- Allocating/Releasing : si preoccupa di allocare deallocare l'uso di risorse ai processi utente, funzionalità particolarmente critica per risorse non condivisibili, a causa del deadlock
- Device independent block size: nasconde al livello superiore alcuni dettagli implementativi come ad esempio la diversa dimensione dei blocchi

Riassumendo