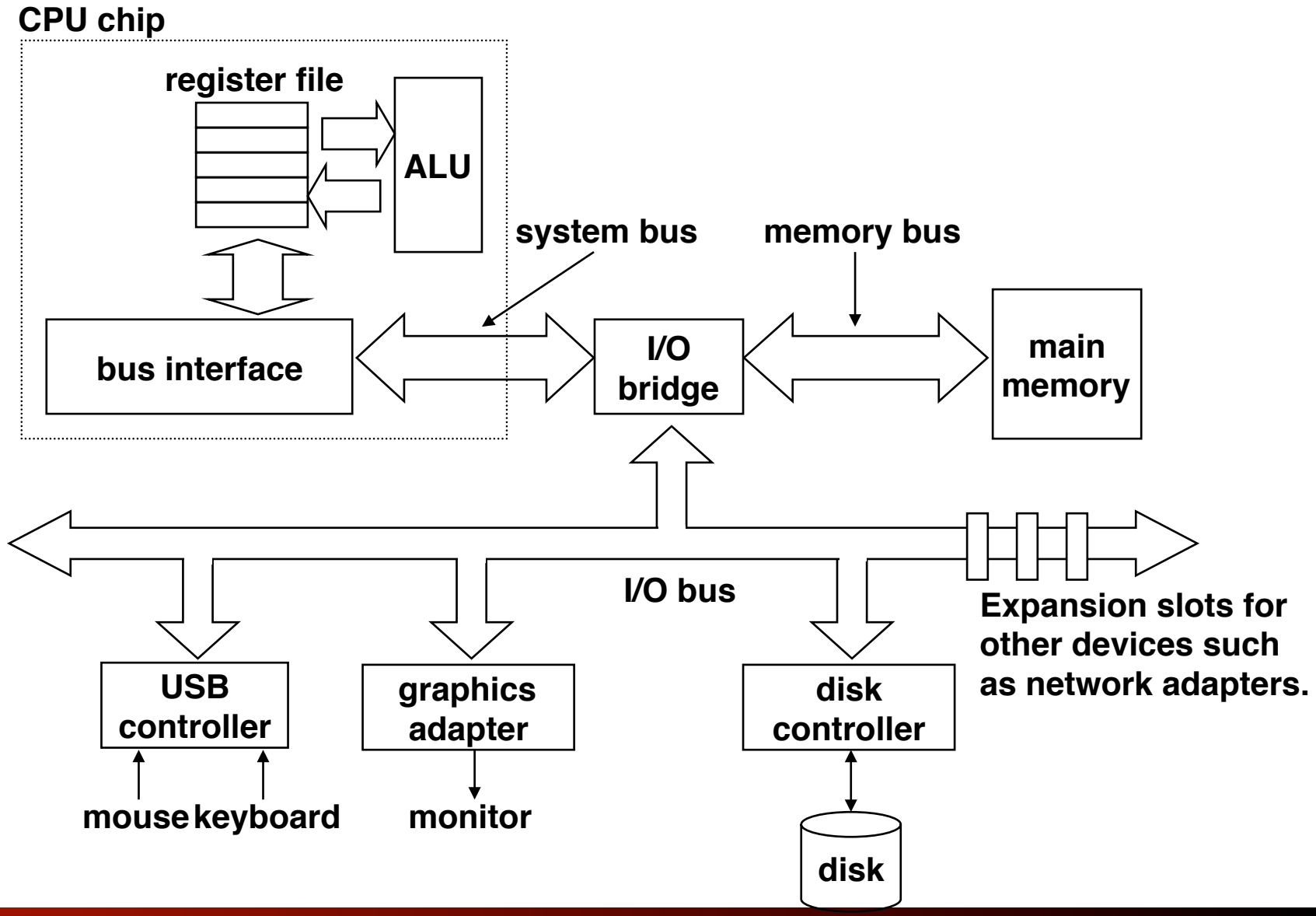
Sistema de Entradas-Saídas
Visão do Programador

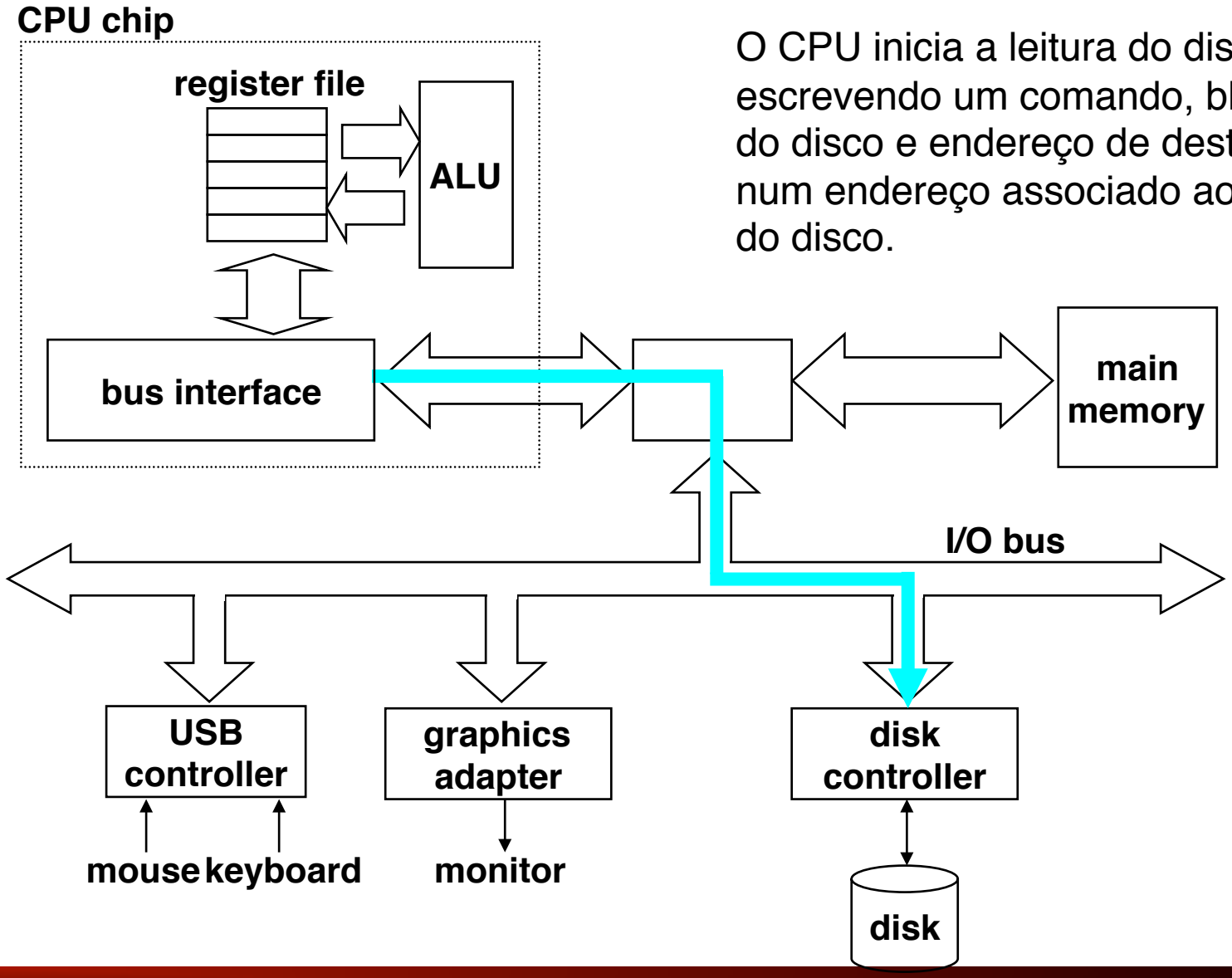
Secções 19.1

S.P. Dandamudi, *Fundamentals of Computer Organization and Design*, Ed. Springer, 2003

Hardware típico

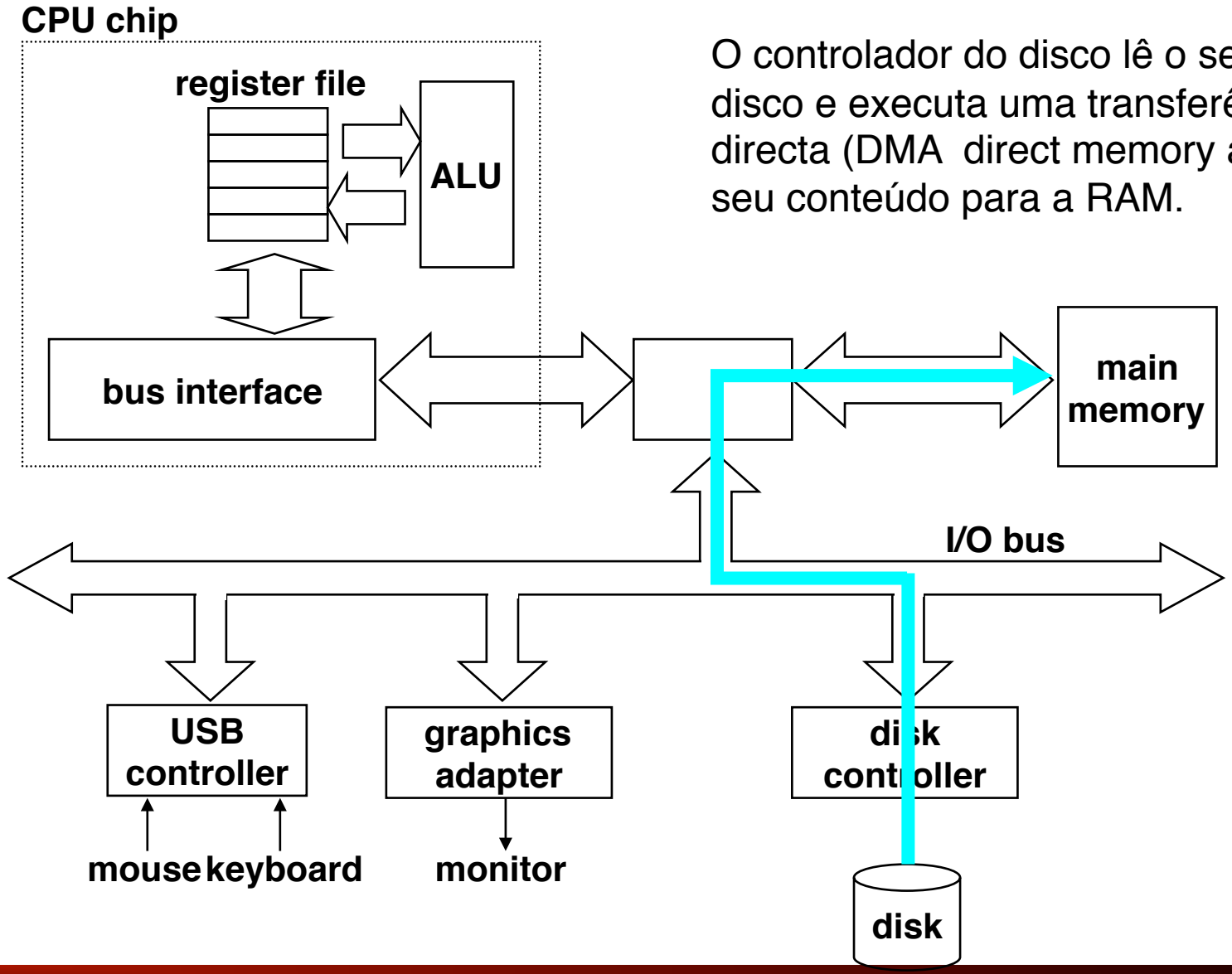


Leitura de um sector do disco: Passo 1



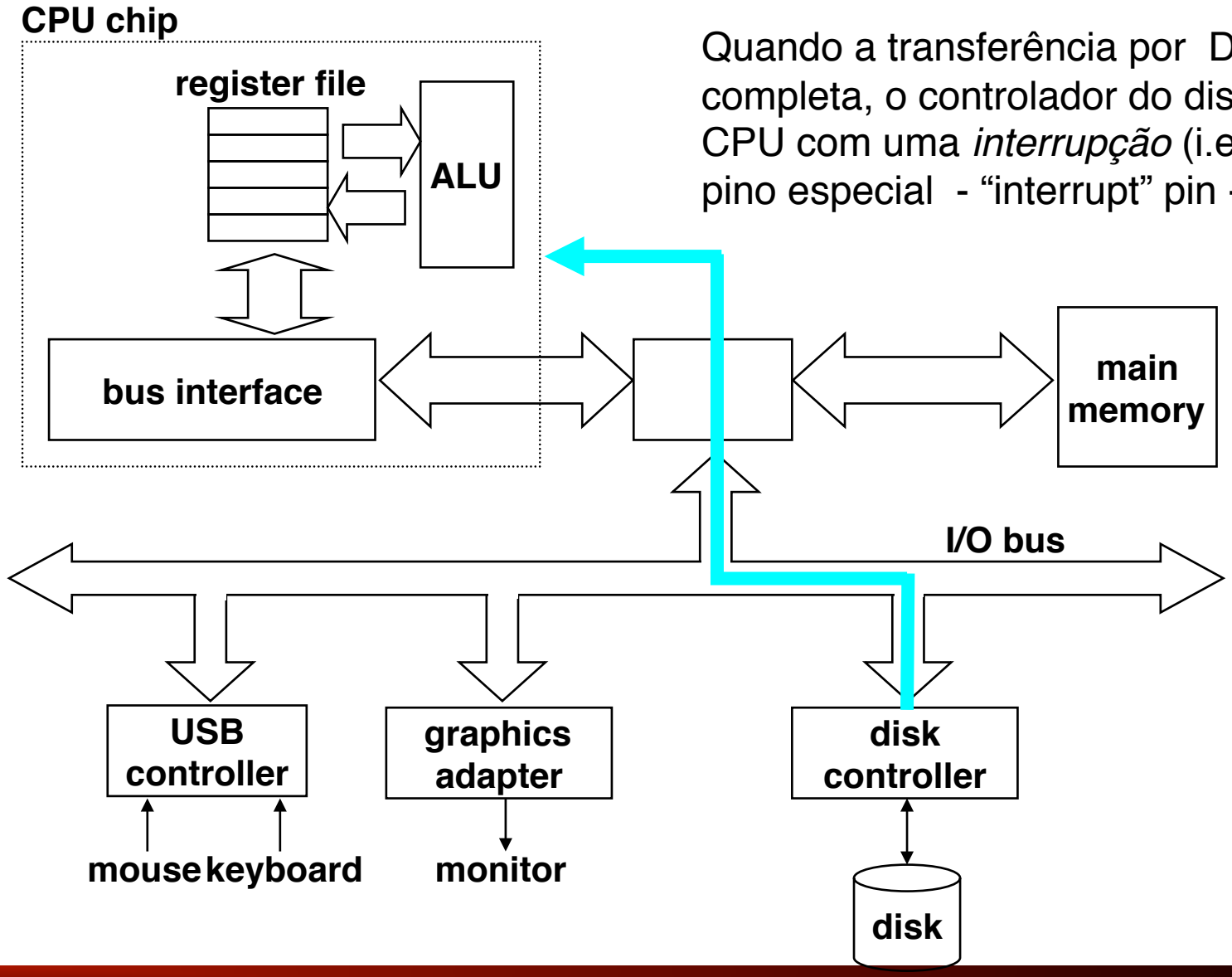
O CPU inicia a leitura do disco, escrevendo um comando, bloco lógico do disco e endereço de destino na RAM num endereço associado ao controlador do disco.

Leitura de um sector do disco: Passo 2



O controlador do disco lê o sector do disco e executa uma transferência directa (DMA direct memory access) do seu conteúdo para a RAM.

Leitura de um sector do disco: Passo 3



Quando a transferência por DMA se completa, o controlador do disco notifica o CPU com uma *interrupção* (i.e., activa um pino especial - "interrupt" pin - no CPU)

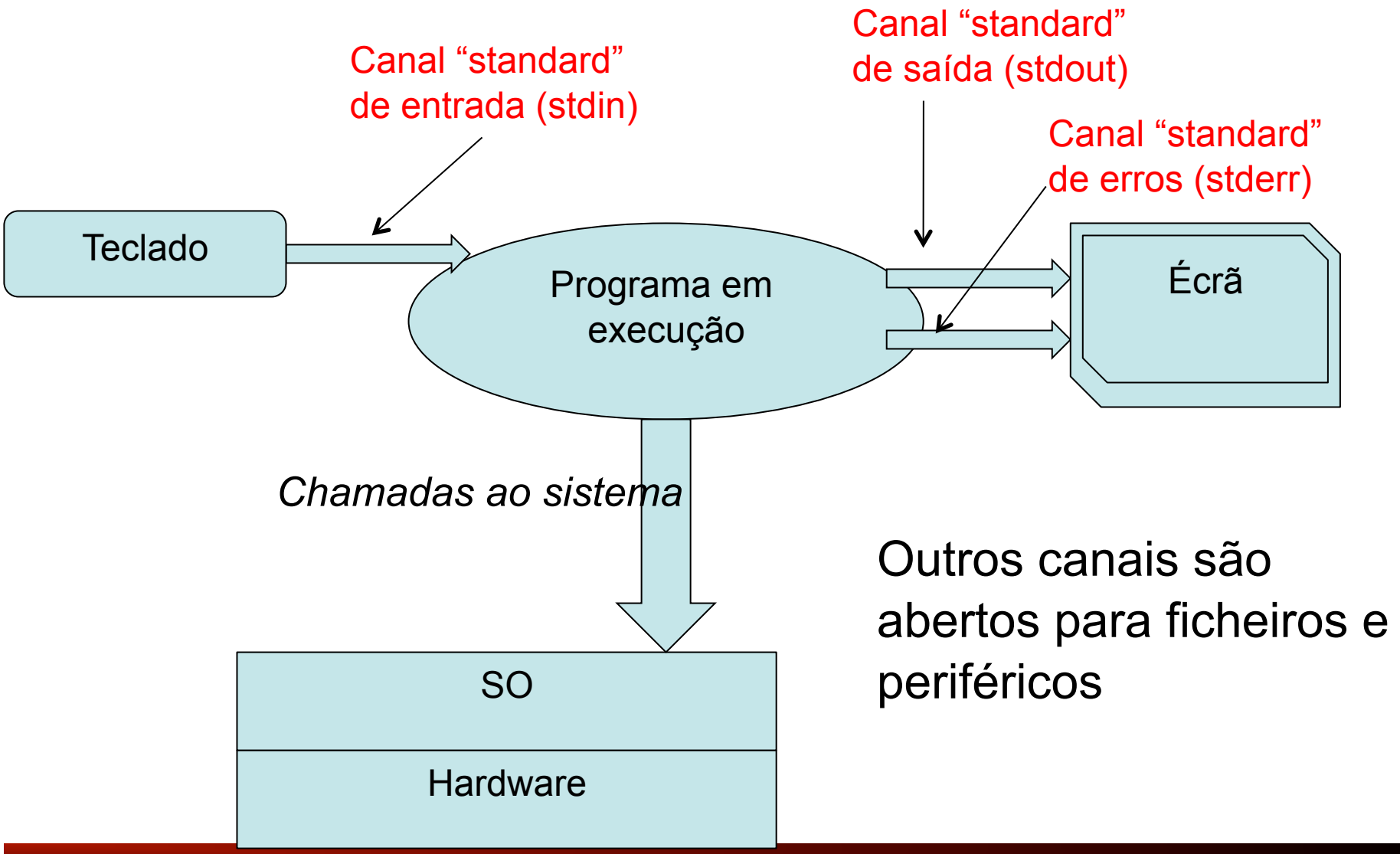
O sistema operativo faz as E/S

- Os programas não podem manipular directamente os periféricos
  - ✓ Seria demasiado complicado e pouco portável
  - ✓ Comprometeria a estabilidade do sistema; exemplo: tabela de ficheiros guardada no disco
- O SO fornece um conjunto de chamadas ao sistema que permitem aos processos fazer acesso aos periféricos de forma controlada
  - ✓ Os periféricos são abstraídos em duas grandes classes
    - ✓ Orientados para a transferência byte a byte (***stream devices***)
    - ✓ Orientados para a transferência por blocos (***block devices***)
  - ✓ Do ponto de vista do programador a visão é que os periféricos são **ficheiros especiais**

Ficheiros sequenciais

- Um *ficheiro* é uma sequência de *m* bytes:
  - ✓ $B_0, B_1, \ldots, B_k, \ldots, B_{m-1}$
- Os dispositivos de E/S são representados por ficheiros especiais
  - ✓ `/dev/sda2` ou C: (partição `/usr` do disco)
  - ✓ `/dev/tty2` ou COM1: (terminal)

Canais de entrada/saída de um programa



Outros canais são abertos para ficheiros e periféricos

Chamadas ao sistema para E/S no Unix

- A correspondência entre ficheiros e periféricos permite ao SO exportar uma interface simples para manipular periféricos - usualmente chamada *Unix I/O* mas que é suportada por outros SOs
- Ideia central: Todas as entradas e saídas são suportadas numa maneira consistente e uniforme
- Operações Básicas (chamadas ao sistema):
  - ✓ Abertura e fecho de ficheiros
    - ✓ `open()` e `close()`
  - ✓ Mudar a *posição corrente no ficheiro* (seek)
    - ✓ `lseek` (não interessa para já)
  - ✓ Ler e escrever num ficheiro
    - ✓ `read()` e `write()`

Abertura de ficheiros

- Abrir um ficheiro informa o SO que se deve preparar para dar acesso ao ficheiro.

```
int fd;   /* file descriptor */

if ((fd = open("/etc/hosts", O_RDONLY)) < 0) {
   perror("open");
   exit(1);
}
```

- Retorna um inteiro *fd* (*file descriptor*) *que permite o acesso ao ficheiro*
  - ✓ `fd == -1` indica que ocorreu um erro
- No UNIX cada processo lançado a partir do *shell* começa com três ficheiros associado ao terminal:
  - ✓ 0: standard input (stdin)
  - ✓ 1: standard output (stdout)
  - ✓ 2: standard error (stderr)

Fechar um ficheiro

- Fechar um ficheiro informa o SO que o programa já não precisa de fazre acesso ao ficheiro.

```
int fd;     /* file descriptor */
int retval; /* return value */

if ((retval = close(fd)) < 0) {
   perror("close");
   exit(1);
}
```

- É importante retornar sempre códigos de erro !

Ler do ficheiros

- Ler do ficheiro copia bytes da posição corrente no ficheiro e depois actualiza a posição corrente.

```
char buf[512];
int fd;        /* file descriptor */
int nbytes;    /* number of bytes read */

/* Open file fd ...  */
/* Then read up to 512 bytes from file fd */
if ((nbytes = read(fd, buf, sizeof(buf))) < 0) {
   perror("read");
   exit(1);
}
```

- Retorna o nº de bytes lidos do ficheiro `fd` para o endereço `buf`
  - ✓ `nbytes < 0` indica que ocorreu um erro.
  - ✓ `nbytes < sizeof(buf)` é possível e não é um erro!

Escrever em ficheiros

- Escrever no ficheiro copia bytes da RAM para a posição corrente no ficheiro; a seguir a posição corrente no ficheiro é actualizada.

```
char buf[512];
int fd;        /* file descriptor */
int nbytes;    /* number of bytes read */

/* Open the file fd ... */
/* Then write up to 512 bytes from buf to file fd */
if ((nbytes = write(fd, buf, sizeof(buf)) < 0) {
   perror("write");
   exit(1);
}
```

- Retorna o nº de bytes written de `buf` para o ficheiro `fd`.
  - ✓ `nbytes < 0` indica que ocorreu um erro.
  - ✓ Pode ser retornado um número de bytes menor que o pedido sem ser um erro

Exemplo

- Copiar do *standard input* para o *standard output* um byte de cada vez.

```
#include <stdlib.h>

int main(void)
{
    char c;

    while(read(STDIN_FILENO, &c, 1) != 0)
       write(STDOUT_FILENO, &c, 1);
    exit(0);
}
```

Funções da Biblioteca Standard de E/S

- A biblioteca standard do C (`libc.a`) contém um conjunto de funções standard de entrada/saída de nível mais elevado standard I/O (stdio)
  - ✓ Suportada obrigatoriamente em qualquer implementação de C.
- Exemplos de funções de E/S (I/O) :
  - ✓ Abrir e fechar ficheiros (`fopen` e `fclose`)
  - ✓ Ler e escrever bytes (`fread` e `fwrite`)
  - ✓ Ler e escrever linhas de texto (`fgets` e `fputs`)
  - ✓ Leitura e escrita formatada (`fscanf` e `fprintf`)

Canais (Streams) Standard de E/S (I/O)

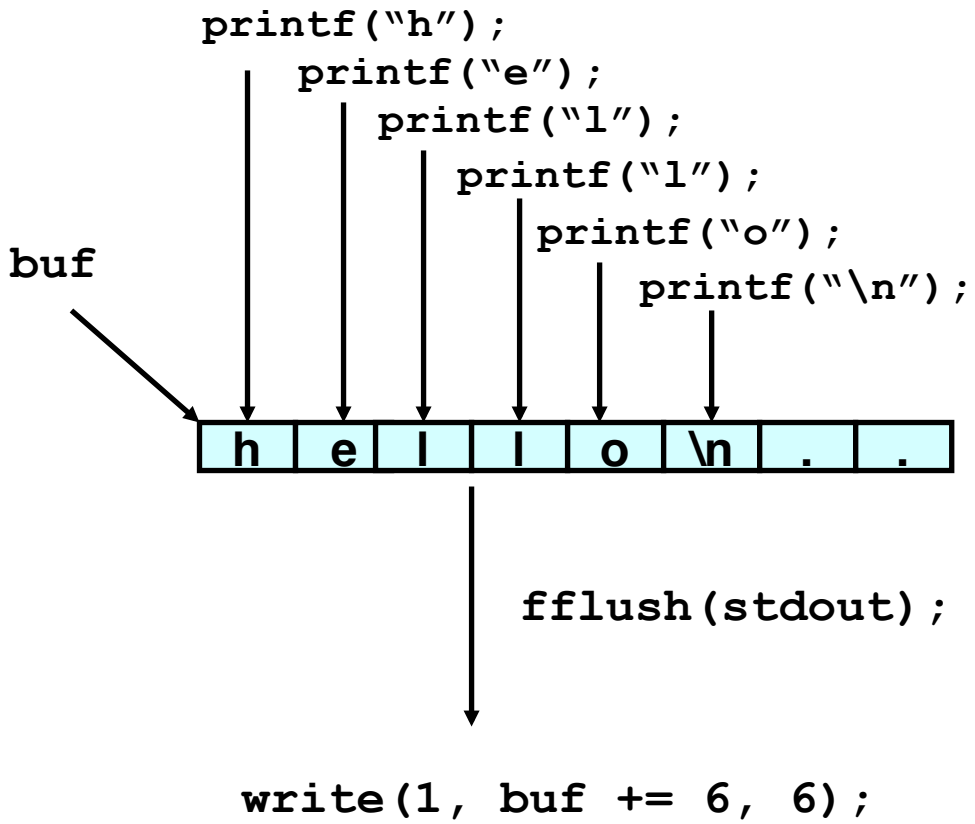
- O I/O Standard I/O modela os ficheiros abertos como um *canal* ou sequências de bytes (*streams*)
  - ✓ Abstracção para um descritor de ficheiros e um *buffer* em memória (RAM).
- Um programa em C começa com três canais abertos (definido em `stdio.h`)
  - ✓ `stdin` (standard input)
  - ✓ `stdout` (standard output)

```
#include <stdio.h>
extern FILE *stdin;  /* standard input (descriptor 0) */
extern FILE *stdout; /* standard output (descriptor 1) */
extern FILE *stderr; /* standard error (descriptor 2) */

int main() {
    fprintf(stdout, "Hello, world\n");
}
```

“Bufferização” no I/O Standard

- As funções Standard de I/O usam I/O “bufferizado”



“Bufferização” no I/O Standard

- Pode-se ver o “buffering” em acção usando o comando `strace` do UNIX:

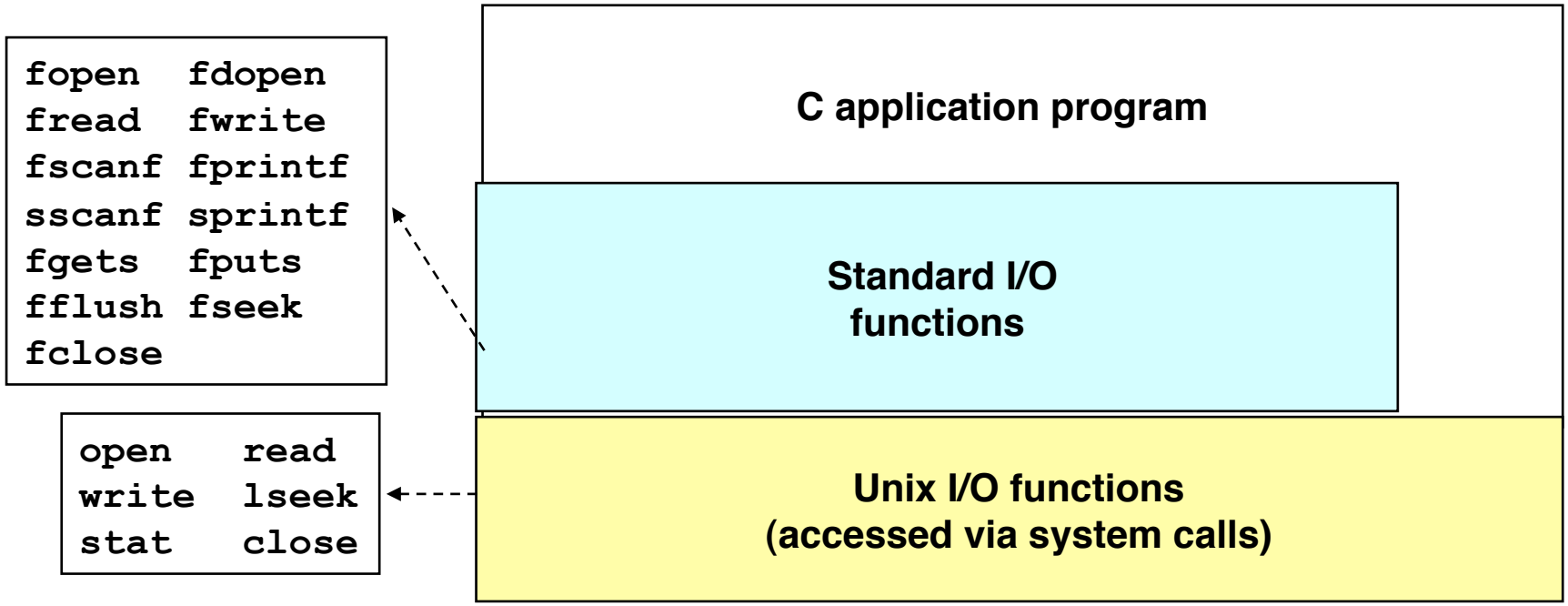
```
#include <stdio.h>

int main()
{
    printf("h");
    printf("e");
    printf("l");
    printf("l");
    printf("o");
    printf("\n");
    fflush(stdout);
    exit(0);
}
```

```
linux> strace ./hello
execve("./hello", ["hello"], [/* ... */]).
...
write(1, "hello\n", 6...)                = 6
...
_exit(0)                                 = ?
```

I/O de Sistema vs. Standard I/O

- Standard I/O é implementado usando chamadas ao sistema



Organização do SO relativa a E/S

- Suporte da noção de canal (e respectivas chamadas ao sistema) – open/ read/ write/ close
  - ✓ Independência do hardware
- Suporte de diferentes tipos de periféricos
  - ✓ Adaptação das chamadas ao sistema aos vários tipos de periféricos
  - ✓ A parte que lida directamente com o periférico é conhecida por *device driver* (gestor de periférico)
    - ✓ Tem de lidar com todos os detalhes do periférico (endereços, registos, interrupções, DMA, …)