



Sistemas Operacionais

Sistema de Arquivos

Prof. José Paulo G. de Oliveira
Eng. da Computação, UPE
jpgo@ecomp.poli.br

Conteúdo

- Visão Geral
- Drivers de dispositivos
- Gerência de blocos
- Sistema virtual de arquivos
- Exemplos

Conteúdo

- **Visão Geral**
- Drivers de dispositivos
- Gerência de blocos
- Sistema virtual de arquivos
- Exemplos

Sistemas de Arquivos

A construção de um sistema de arquivos envolve vários pontos importantes, que vão:

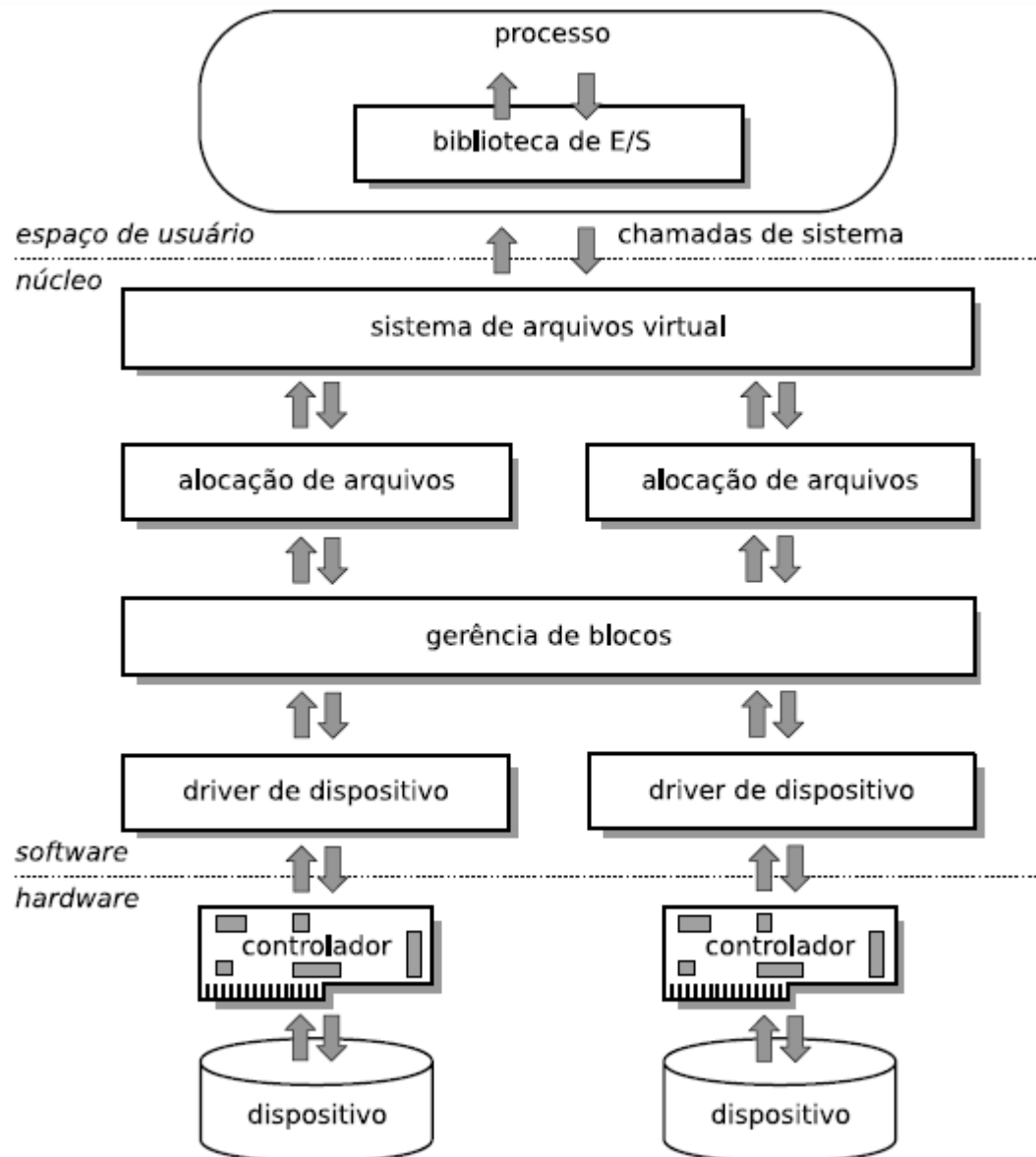
- desde o acesso de **baixo nível** aos dispositivos físicos de armazenamento

Sistemas de Arquivos

A construção de um sistema de arquivos envolve vários pontos importantes, que vão:

- desde o acesso de **baixo nível** aos dispositivos físicos de armazenamento
- à implementação da **interface de acesso** a arquivos para os programadores

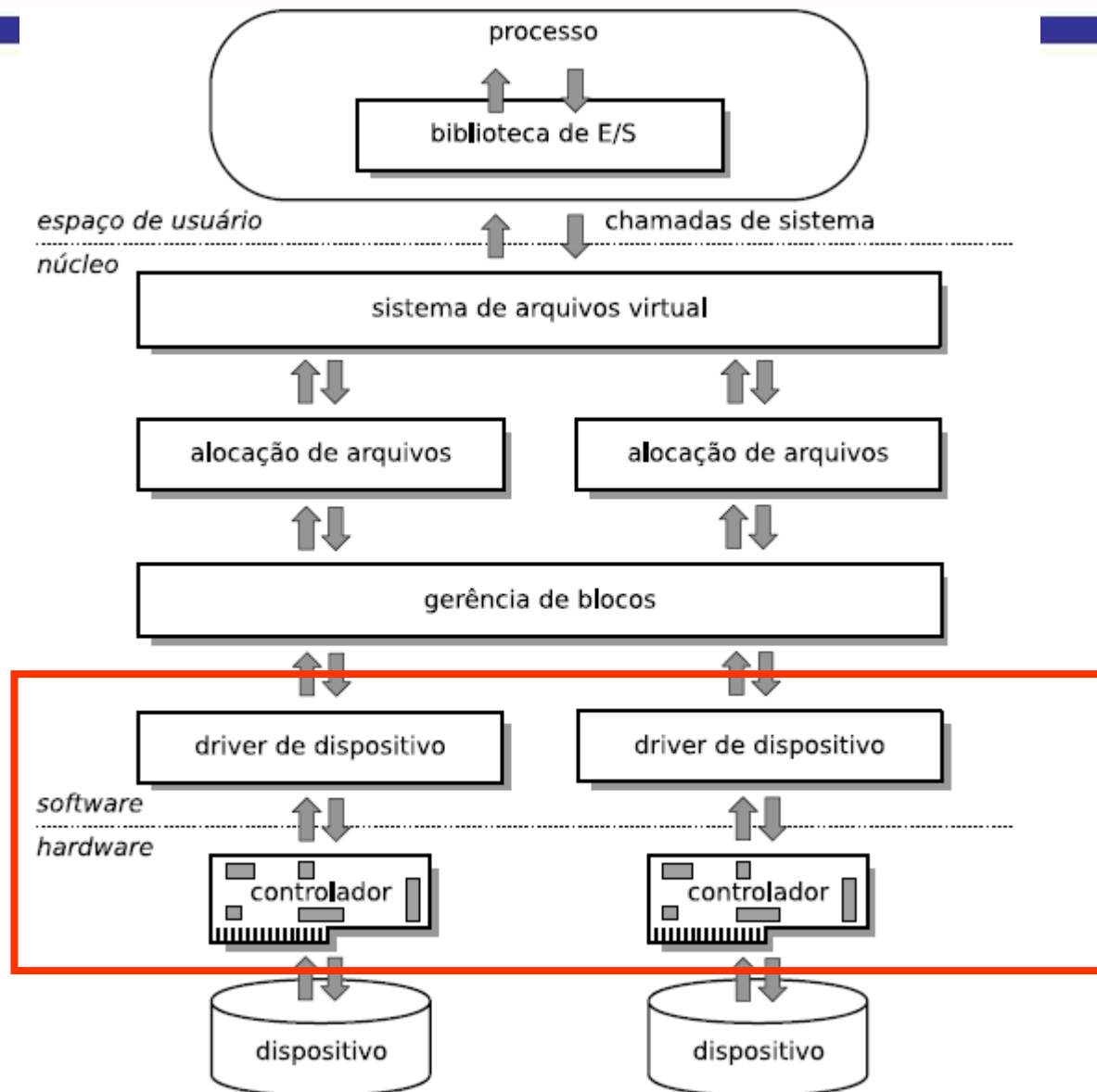
Arquitetura Geral



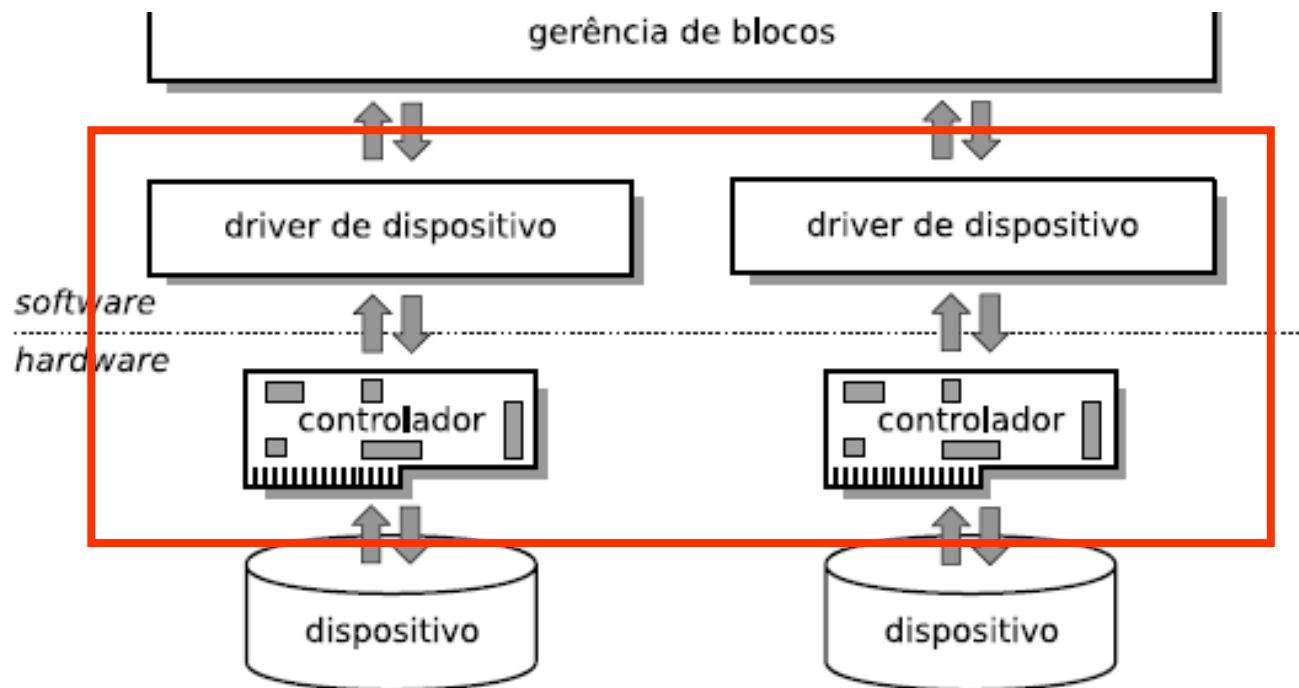
Conteúdo

- Visão Geral
- **Drivers de dispositivos**
- Gerência de blocos
- Sistema virtual de arquivos
- Exemplos

Arquitetura Geral

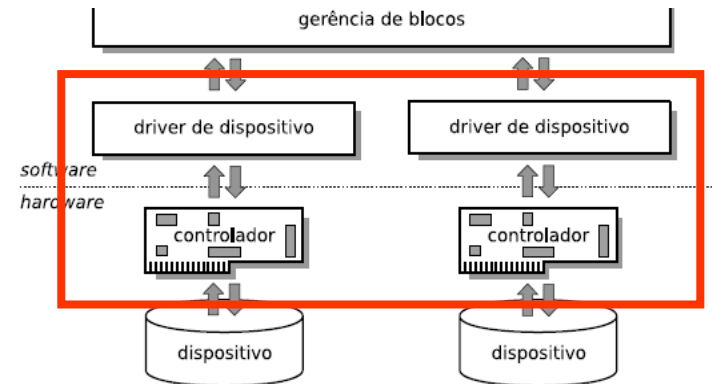


Sistemas de Arquivos



Sistemas de Arquivos

Controladores e Drivers de Dispositivos



- **Controladores:** configurados e acessados pelo núcleo do sistema operacional
 - por meio dos drivers de dispositivos
- **Drivers:** componentes de software capazes de interagir com os controladores

Sistemas de Arquivos

Pesquisar e-mail

3 de 135

Próximo Webinar: Anatomia de um Device Driver no Linux

Toradex <events@toradex.com> [Cancelar inscrição para mim](#) qui., 6 de mai. 10:01 (há 23 horas)

Toradex Swiss. Embedded. Computing.

CONVITE PARA WEBINAR


Anatomia de um Device Driver no Linux

DOULOS 26 de maio de 2021 **1 HOUR**

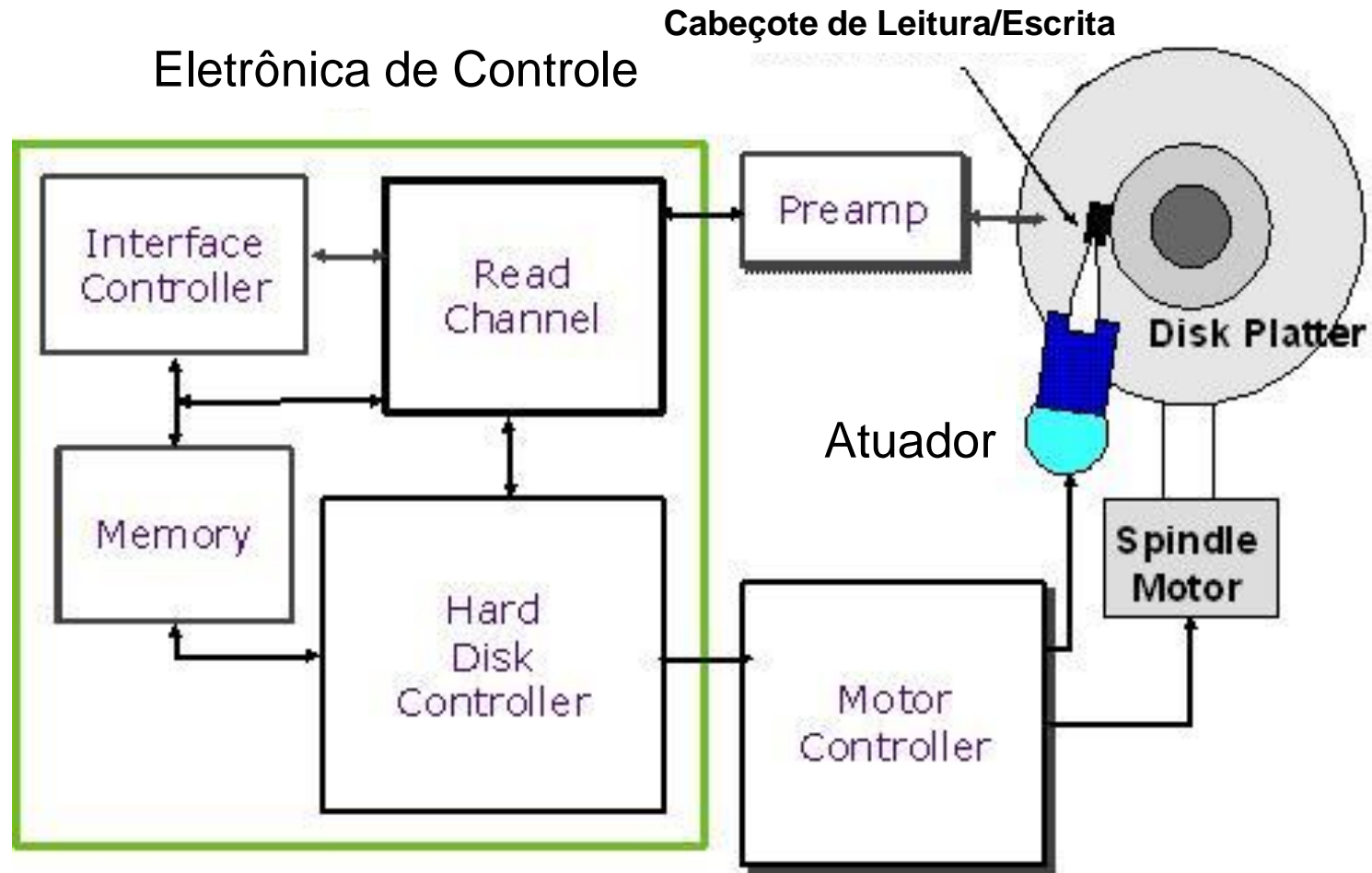
Toradex Swiss. Embedded. Computing.

Palestrante:
Simon Goda
Sr. Member Technical Staff
Doulos | Linux Expert

DOULOS WEBINARS



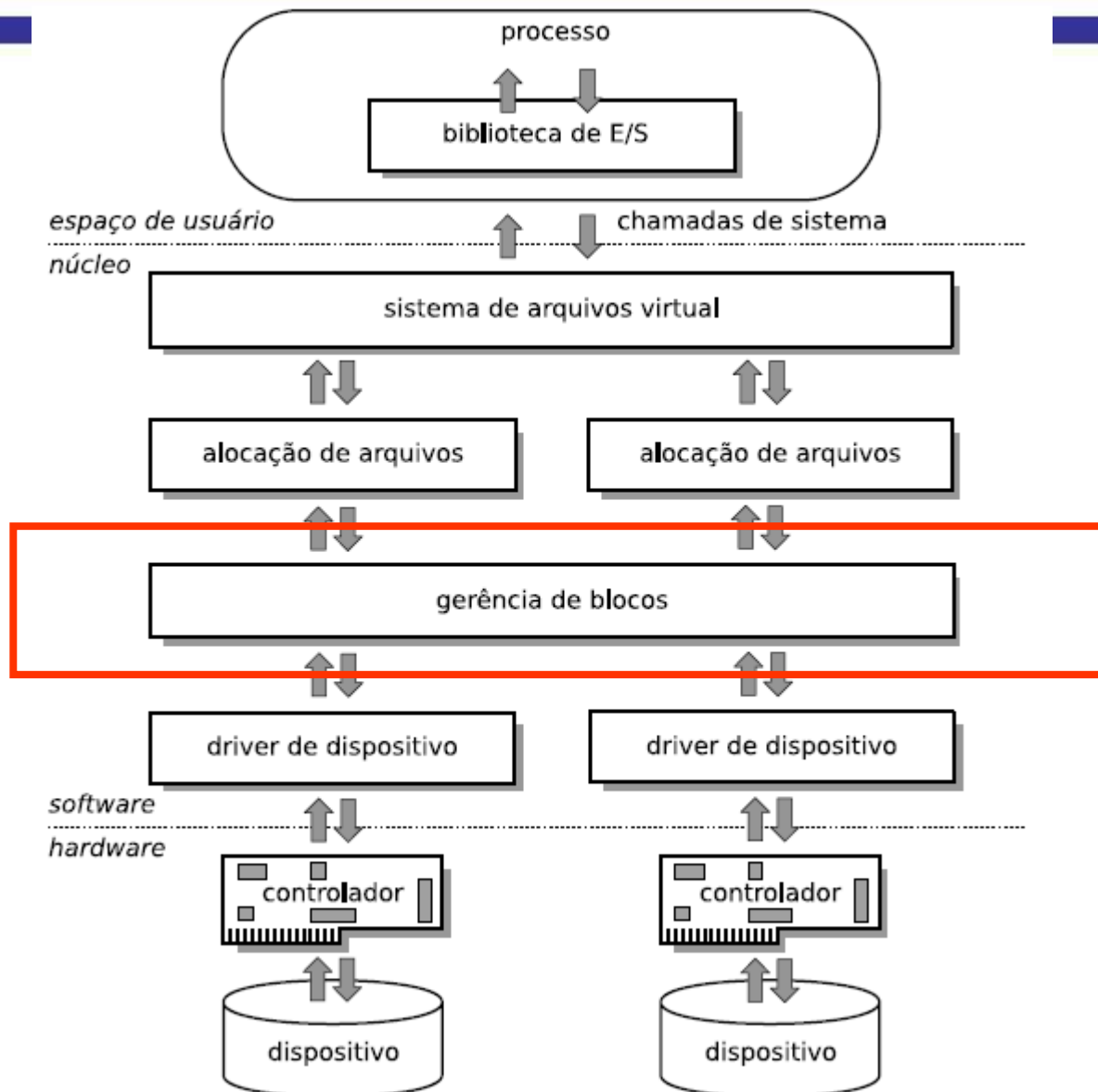
Sistemas de Arquivos



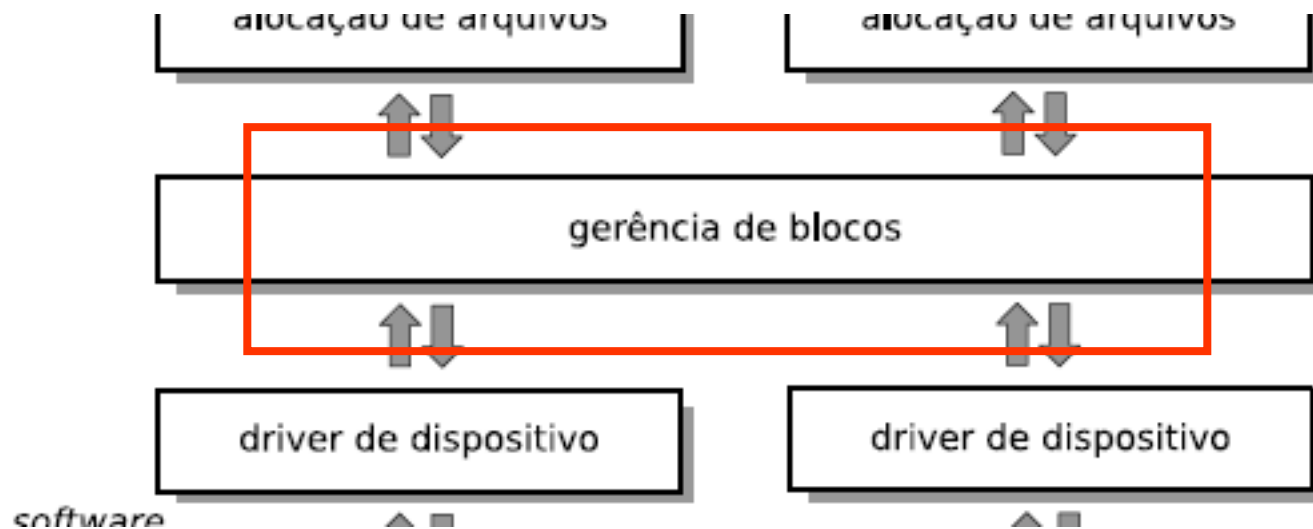
Conteúdo

- Visão Geral
- Drivers de dispositivos
- **Gerência de blocos**
- Sistema virtual de arquivos
- Exemplos

Arquitetura Geral



Sistemas de Arquivos



A camada de **gerência de blocos** gerencia o fluxo de blocos de dados entre a memória e os dispositivos de armazenamento

Sistemas de Arquivos

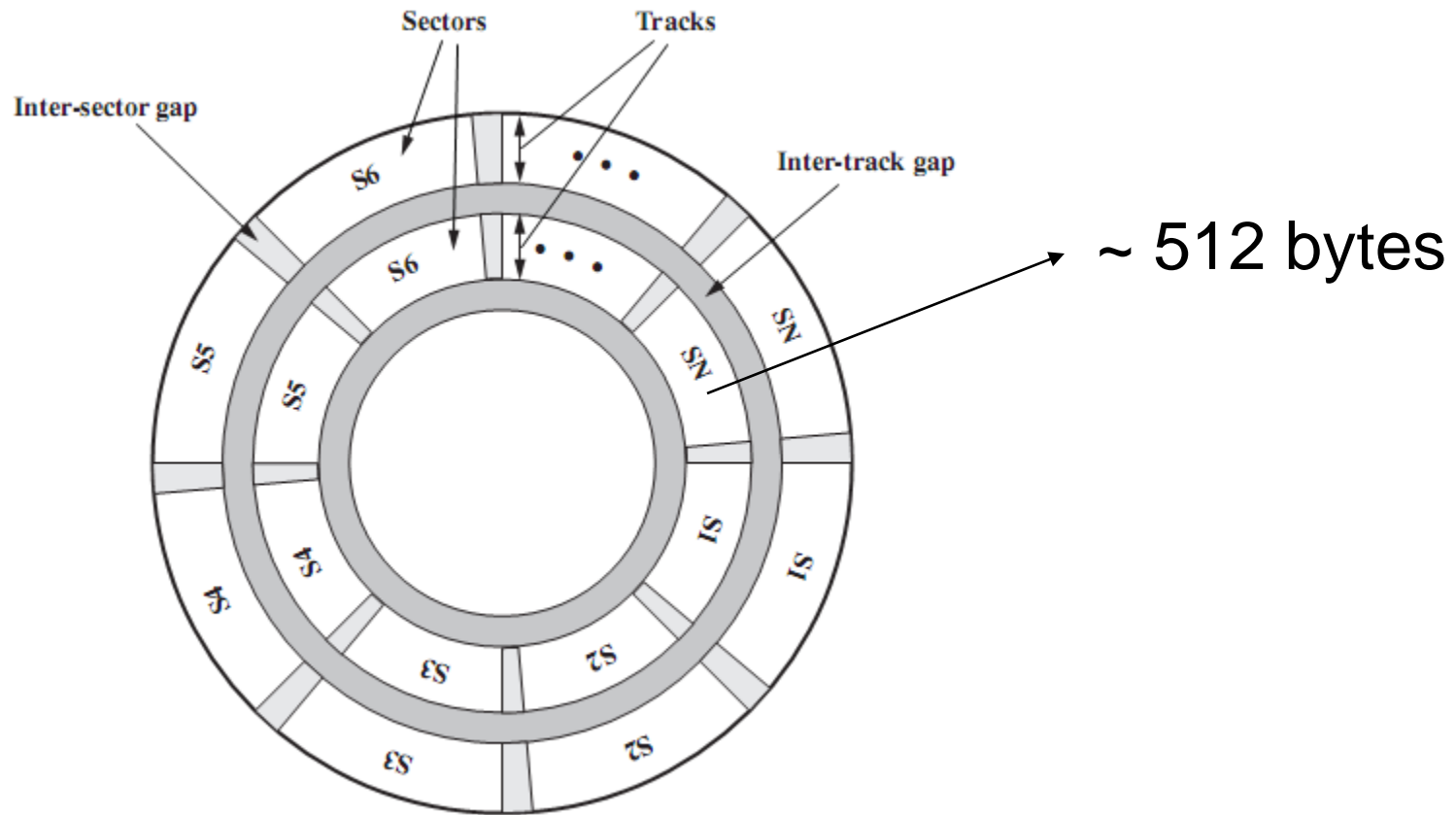
Funções:

- Efetuar o **mapeamento** de blocos lógicos nos blocos físicos do dispositivo
 - Oferecer às camadas superiores a **abstração** de cada dispositivo físico
- Efetuar o ***caching/buffering*** de blocos

Blocos Lógicos e Físicos



Blocos Lógicos e Físicos



Blocos Lógicos e Físicos

Como esses **blocos são pequenos**, o **número de blocos físicos** em um disco rígido pode ser **imenso**. Exemplo:

Disco rígido de 250 GBytes → Mais de **500 milhões** de blocos físicos!

Blocos Lógicos e Físicos

Os sistemas operacionais trabalham com **blocos lógicos ou clusters**:

- Grupos de 2^n **blocos físicos** consecutivos
- **Blocos lógicos** com **4K, 8K, 16K e 32K** bytes são frequentemente usados

Blocos Lógicos e Físicos



O **número de blocos físicos** em cada bloco lógico é definido pelo sistema operacional ao formatar a partição

Blocos Lógicos e Físicos

O **número de blocos físicos** em cada bloco lógico é definido pelo sistema operacional ao formatar a partição

- Blocos lógicos **muito pequenos** implicam mais blocos a gerenciar e menos bytes transferidos em cada operação de leitura/escrita

Blocos Lógicos e Físicos

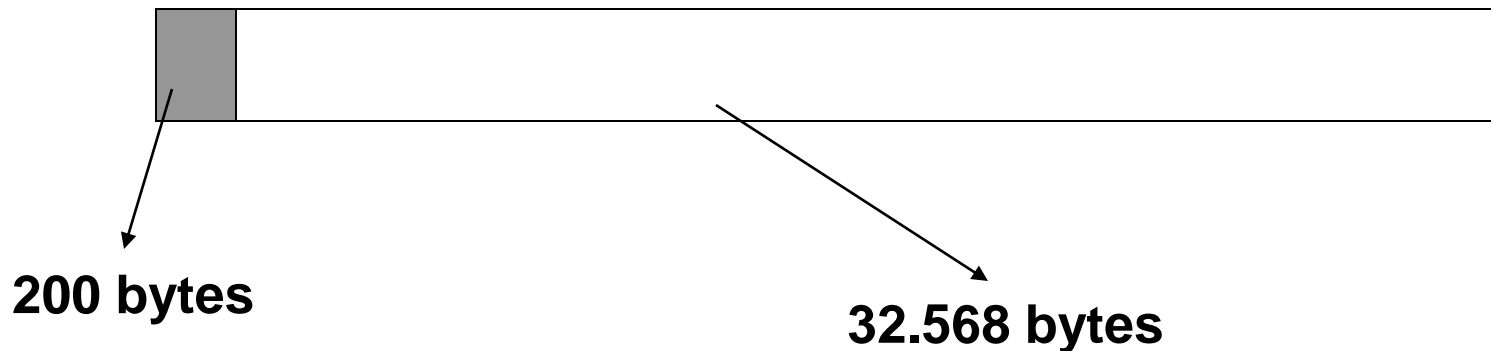
O **número de blocos físicos** em cada bloco lógico é definido pelo sistema operacional ao formatar a partição

- Blocos lógicos **muito pequenos** implicam mais blocos a gerenciar e menos bytes transferidos em cada operação de leitura/escrita
- Blocos lógicos **muito grandes** levam a *fragmentação interna*

Blocos Lógicos e Físicos

Fragmentação Interna:

Um arquivo com 200 bytes em um sistema com blocos lógicos de 32 kbytes ocupará um bloco lógico do qual 32.568 bytes serão desperdiçados



Caching de Entrada/Saída



Motivação

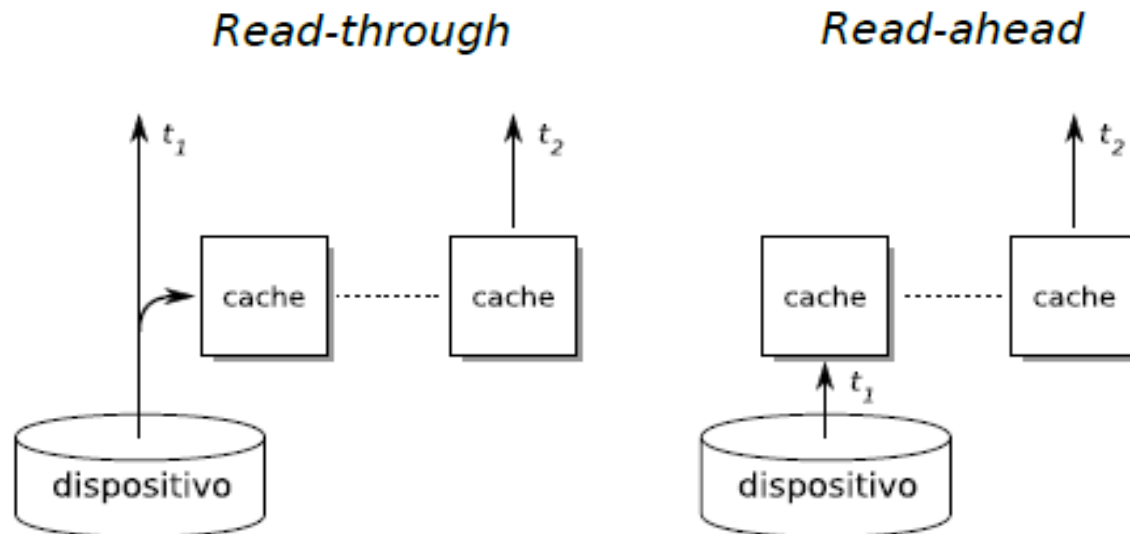
1. Acessar o disco para ler o VBR (*Volume Boot Record*) do volume
2. Nos dados lidos, descobrir onde se encontra o diretório raiz (/) daquele sistema de arquivos
3. Acessar o disco para ler o diretório raiz
4. Nos dados lidos, descobrir onde se encontra o diretório `lib`
5. Acessar o disco para ler o diretório `lib`
6. Nos dados lidos, descobrir onde se encontra o diretório `X11`
7. Acessar o disco para ler o diretório `X11`
8. Nos dados lidos, descobrir onde se encontra o arquivo `libX11.a`
9. Acessar o disco para ler o bloco de controle do arquivo `libX11.a`, que contém seus atributos
10. Criar as estruturas em memória que representam o arquivo aberto
11. Retornar uma referência ao arquivo para o processo solicitante

Motivação

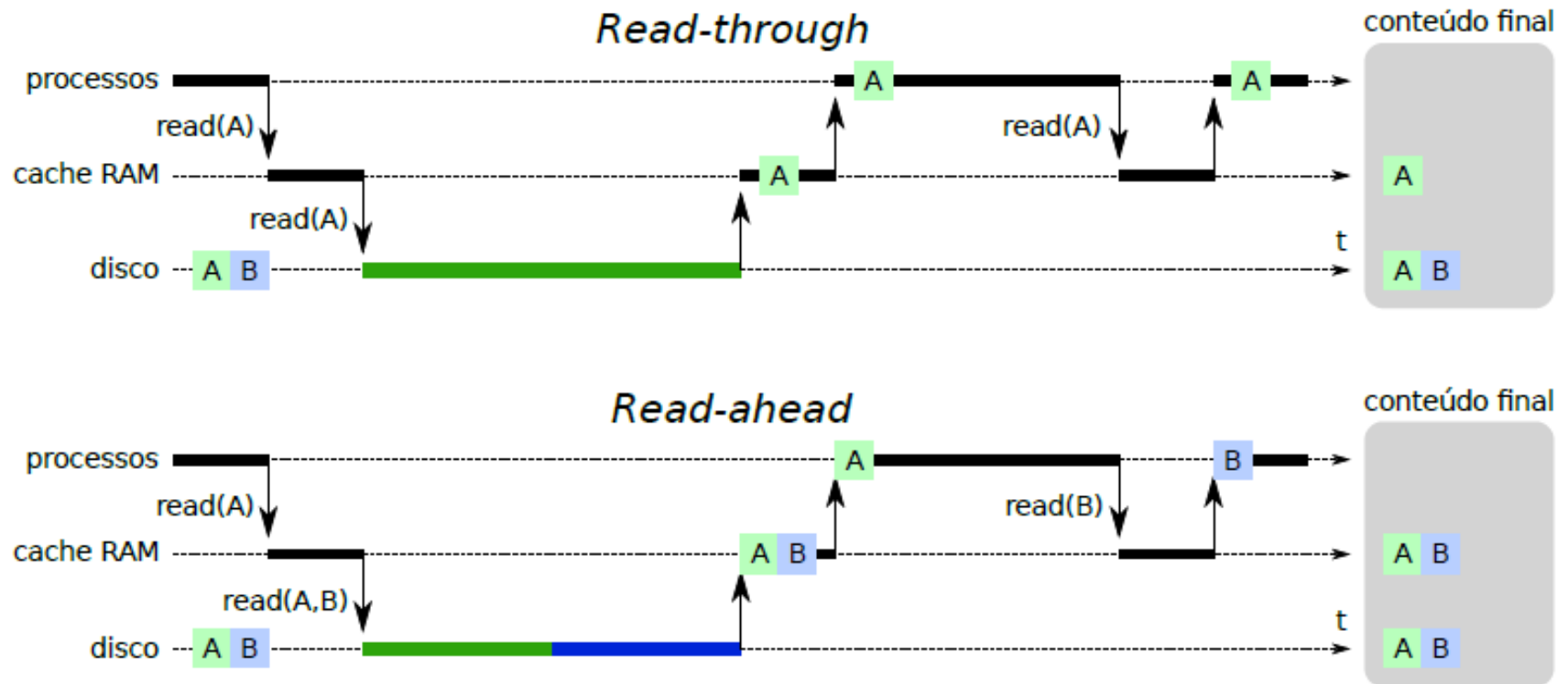
1. **Acessar o disco** para ler o VBR (*Volume Boot Record*) do volume
2. Nos dados lidos, descobrir onde se encontra o diretório raiz (/) daquele sistema de arquivos
3. **Acessar o disco** para ler o diretório raiz
4. Nos dados lidos, descobrir onde se encontra o diretório **lib**
5. **Acessar o disco** para ler o diretório **lib**
6. Nos dados lidos, descobrir onde se encontra o diretório **X11**
7. **Acessar o disco** para ler o diretório **X11**
8. Nos dados lidos, descobrir onde se encontra o arquivo **libX11.a**
9. **Acessar o disco** para ler o bloco de controle do arquivo **libX11.a**, que contém seus atributos
10. Criar as estruturas em memória que representam o arquivo aberto
11. Retornar uma referência ao arquivo para o processo solicitante

Caching de Entrada

No *caching* de **leitura**, blocos de dados **são mantidos em memória** de acordo com uma política LRU (*Least Recently Used*), para acelerar leituras posteriores dos mesmos

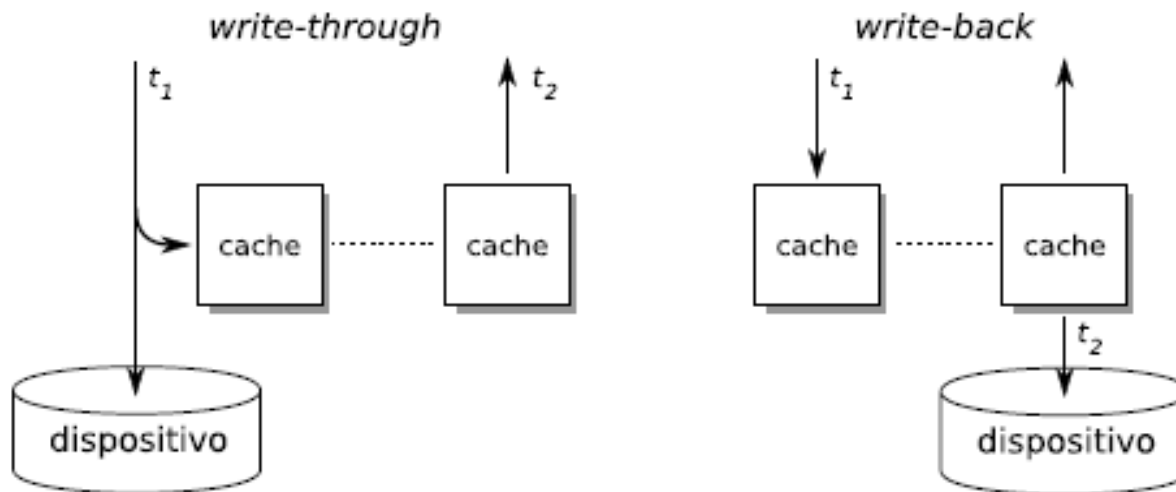


Caching de Entrada

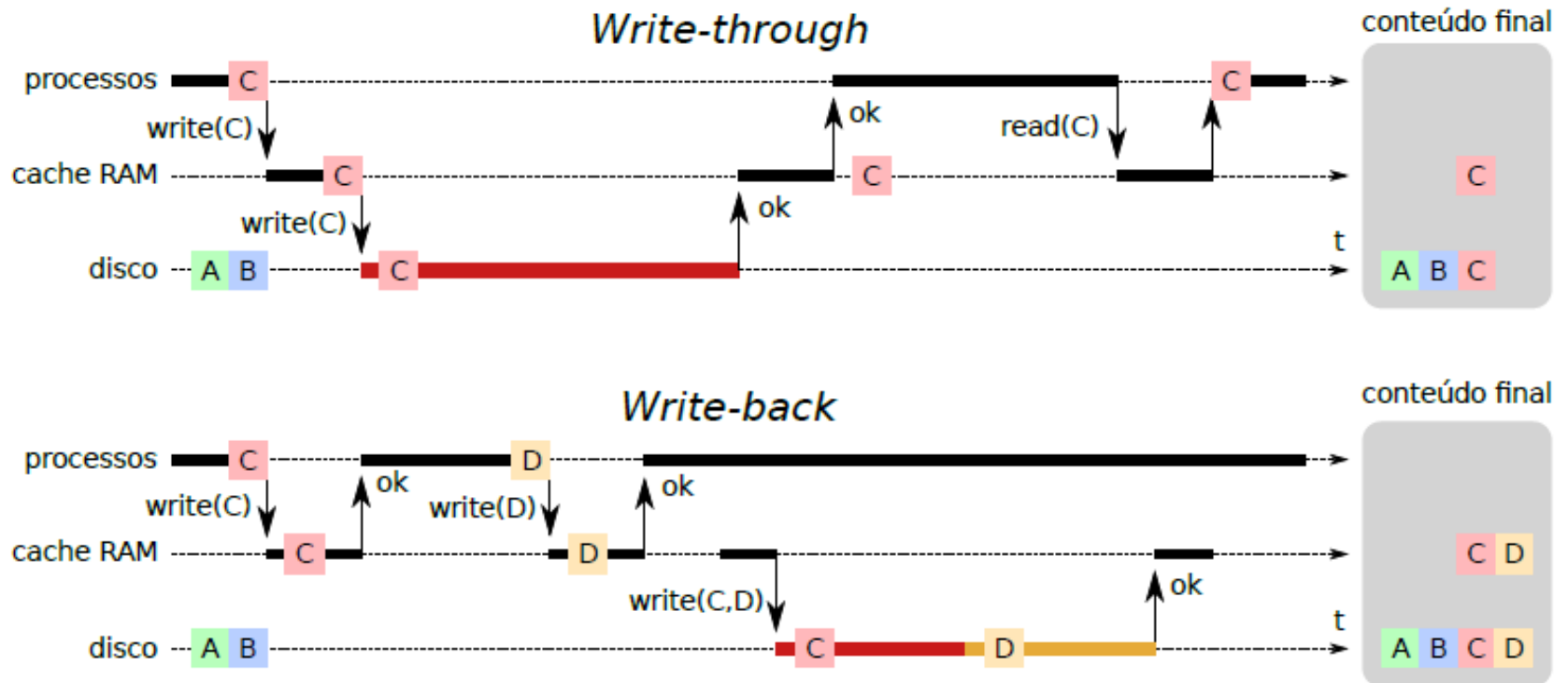


Caching de Saída

No *caching* de **escrita** dados a escrever no disco são **mantidos em memória para leituras posteriores** ou para **concentrar várias escritas** pequenas em poucas escritas maiores



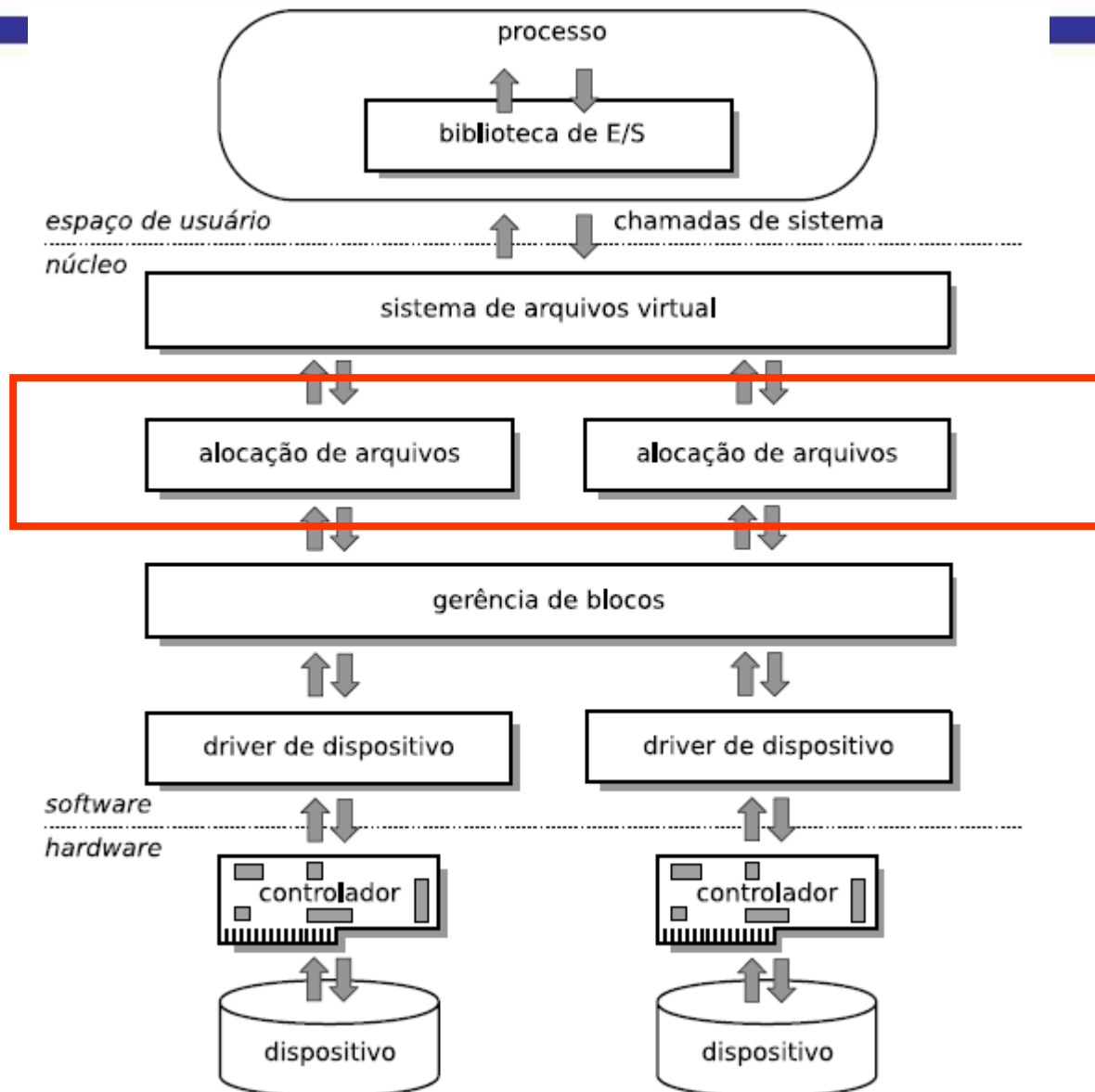
Caching de Saída



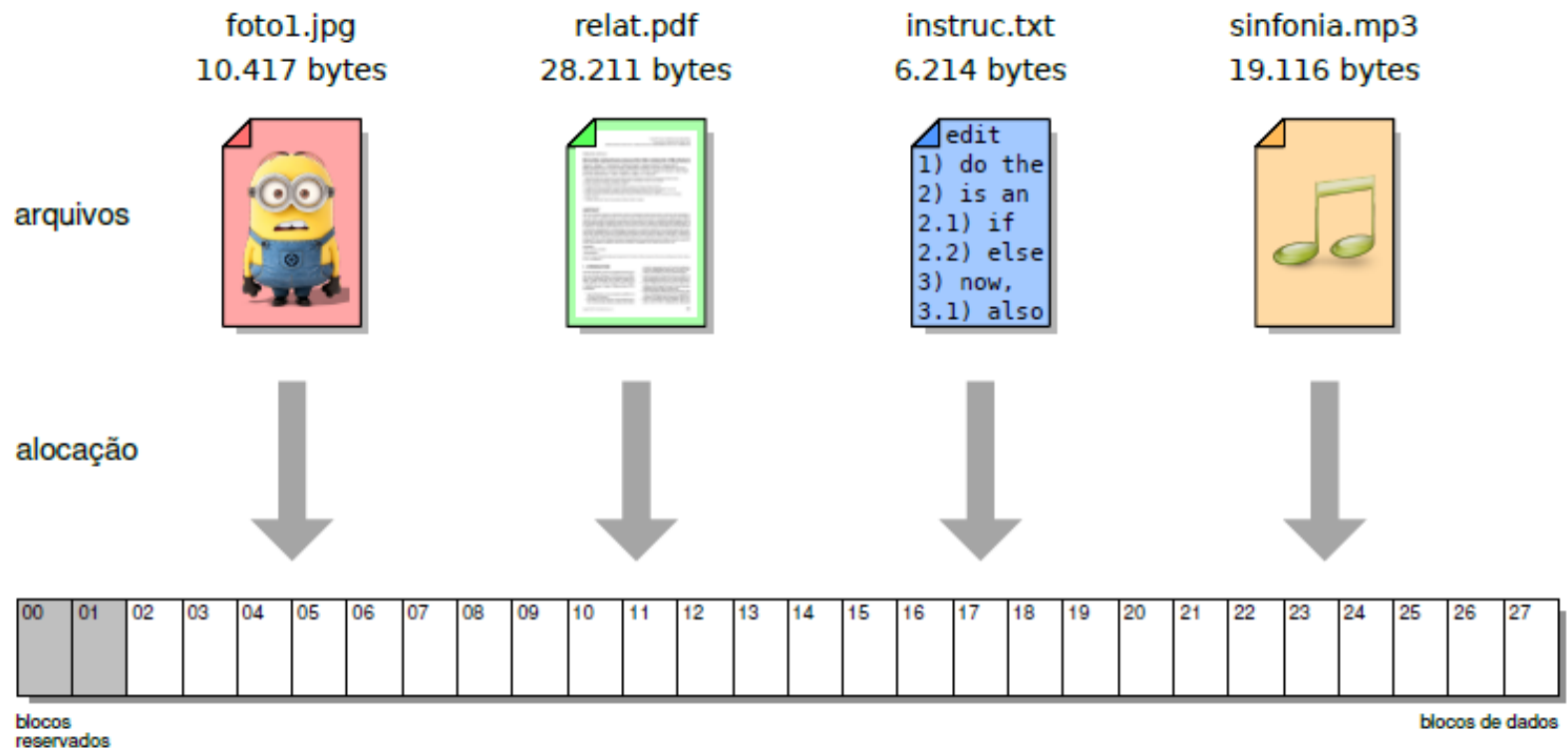
Conteúdo

- Visão Geral
- Drivers de dispositivos
- Gerência de blocos
- **Alocação de arquivos***
- Sistema virtual de arquivos
- Exemplos

Arquitetura Geral



Sistemas de Arquivos

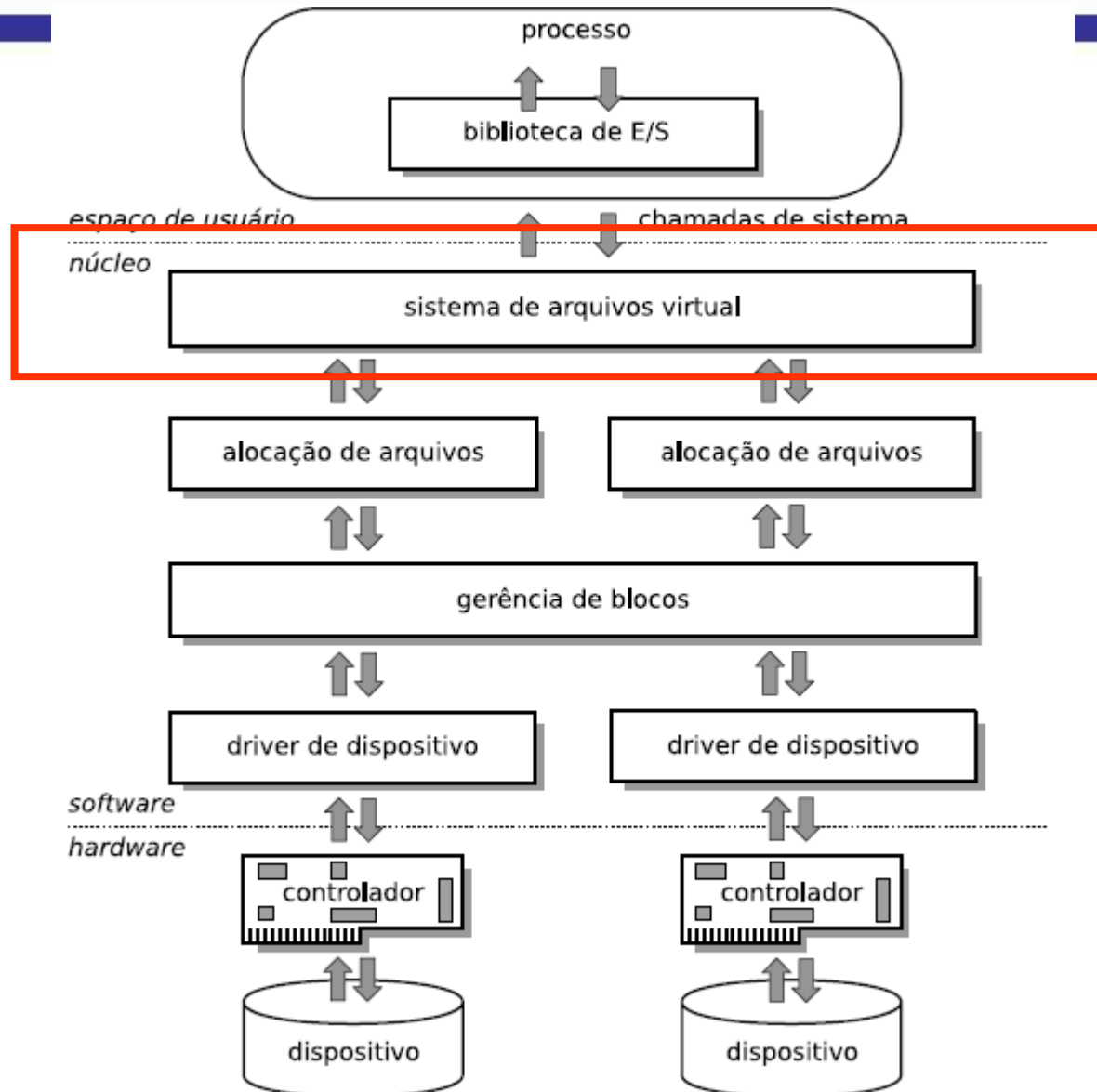


A camada de **alocação de arquivos** tem como função principal **alocar os arquivos sobre os blocos lógicos** oferecidos pela gerência de blocos

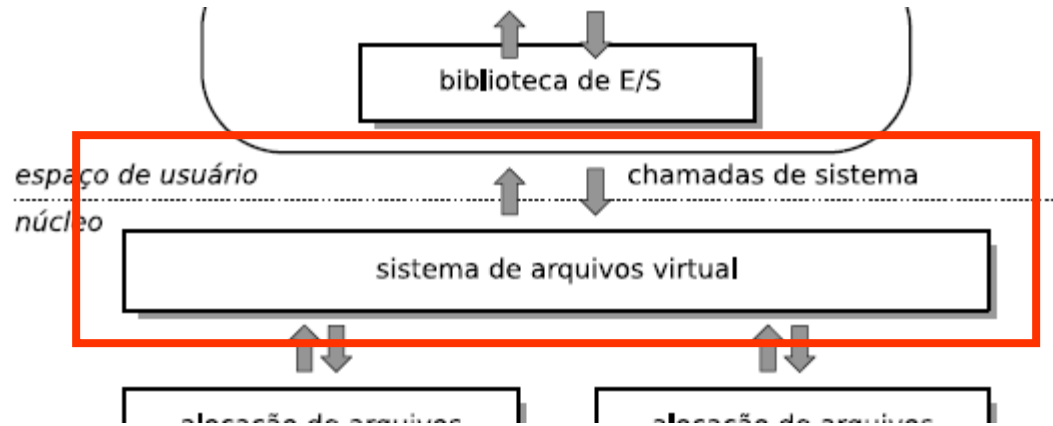
Conteúdo

- Visão Geral
- Drivers de dispositivos
- Gerência de blocos
- **Sistema virtual de arquivos**
- Exemplos

Arquitetura Geral



Sistemas de Arquivos



O **sistema virtual de arquivos** provê uma **interface de acesso a arquivos independente** dos **dispositivos físicos** e das **estratégias de alocação** de arquivos empregadas pelas camadas inferiores

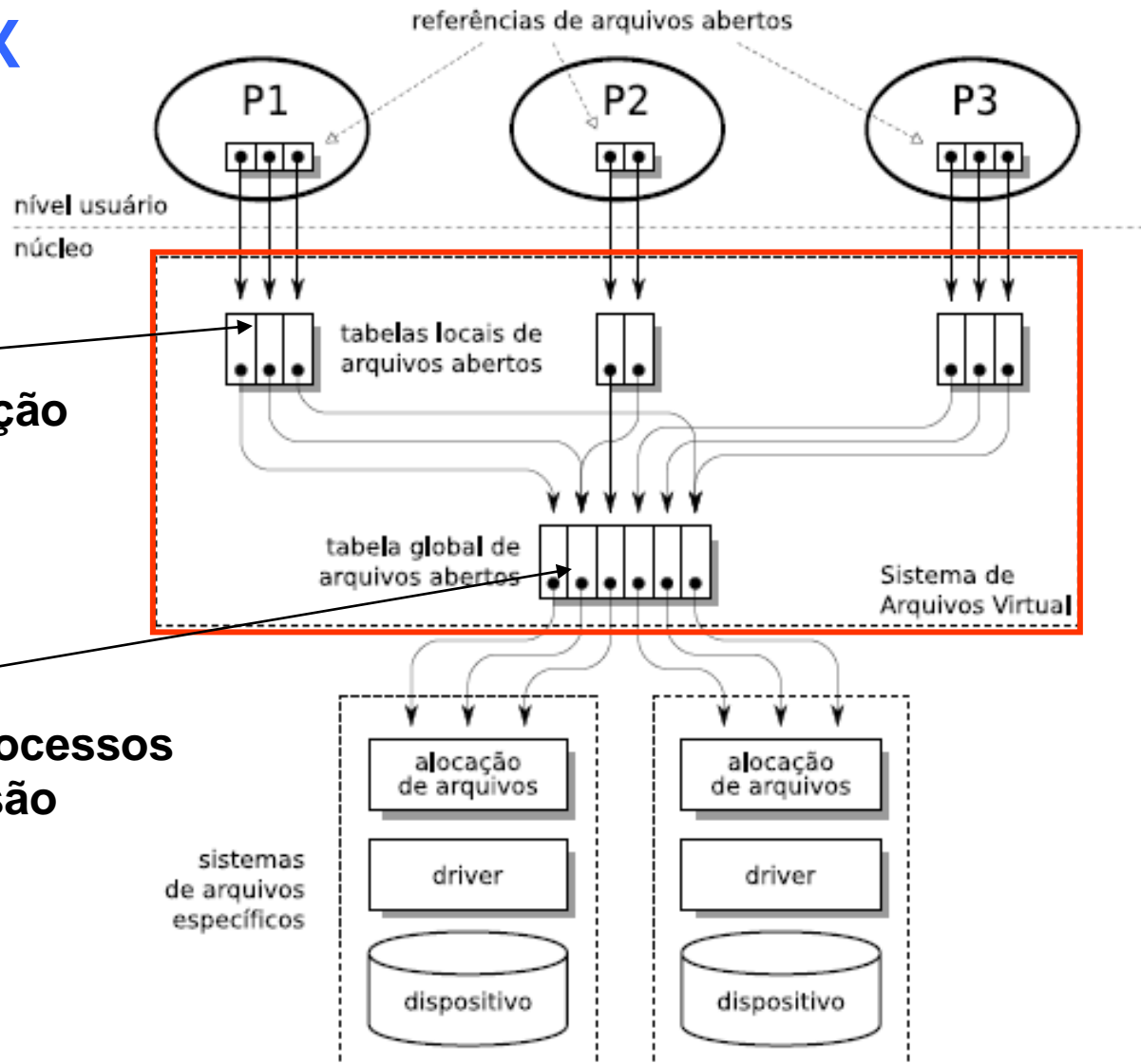
Sistemas de Arquivos

Funções

- Gerencia as permissões
- Travas de acesso compartilhado
- Mantém informações sobre cada arquivo aberto
 - Posição da última operação no arquivo
 - Modo de abertura usado

Sistema de Arquivos Virtual

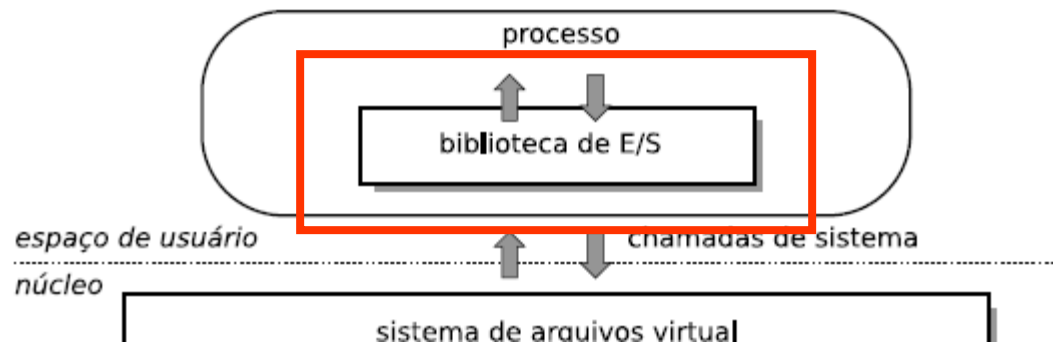
Ex.: UNIX



Ponteiro de posição
Permissões
Modo de acesso

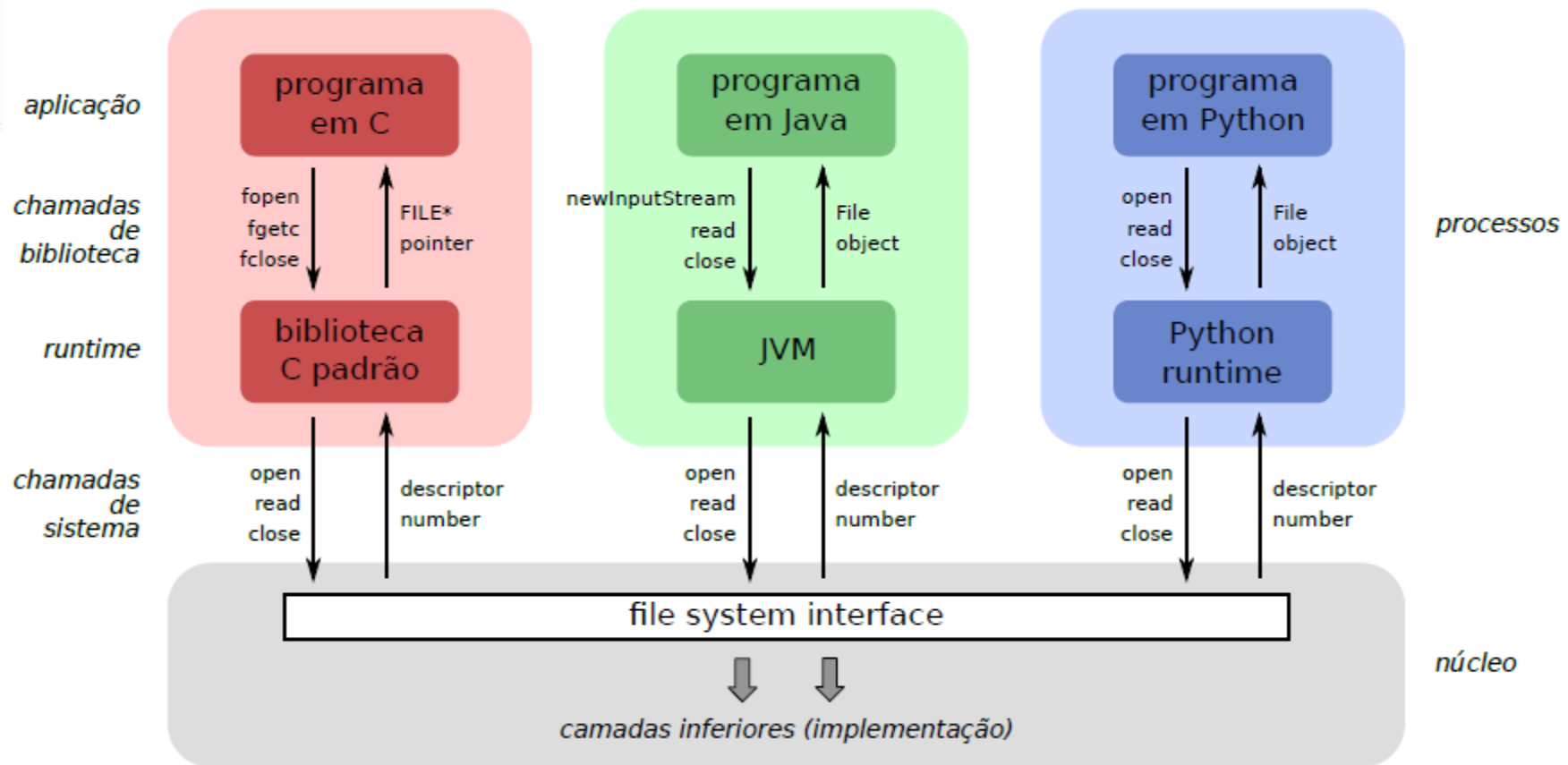
Contador de processos
Trava de exclusão

Sistemas de Arquivos



As **bibliotecas de entrada/saída** usam as **chamadas de sistema** oferecidas pelo sistema operacional para construir funções padronizadas de acesso a arquivos para cada linguagem de programação

Sistemas de Arquivos



Conteúdo

- Visão Geral
- Drivers de dispositivos
- Gerência de blocos
- Sistema virtual de arquivos
- **Exemplos**

A decorative graphic on the left side of the slide. It consists of a vertical light blue bar, a vertical orange bar, and a grey rectangle. A horizontal dark blue bar extends from the grey rectangle across the top of the slide.

Exemplo de interface Linguagem C

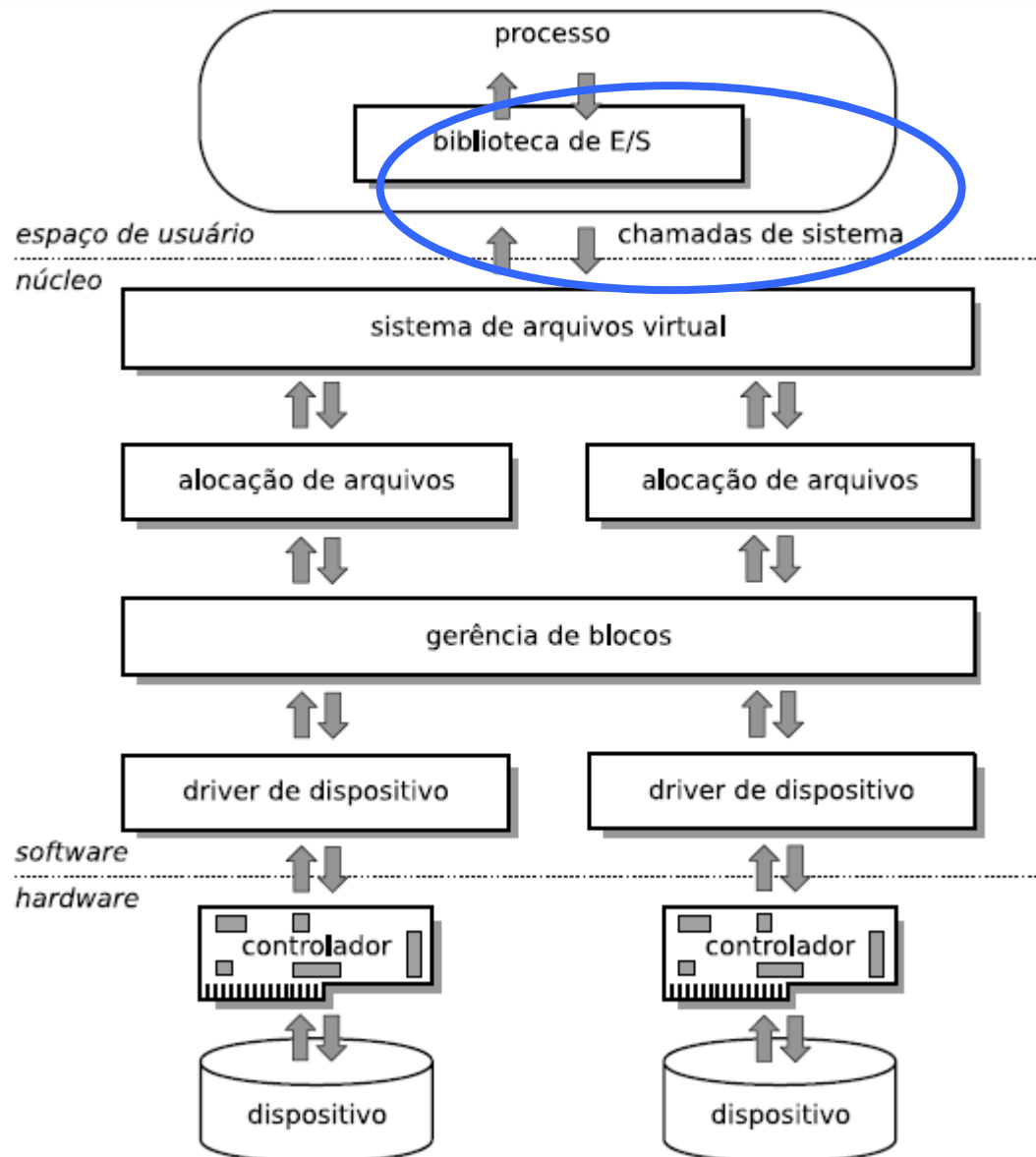
Exemplo de interface

Cada linguagem de programação define sua própria forma de representar arquivos abertos e as **funções** ou métodos usados para manipulá-los

Em **C**, cada arquivo aberto é representado por uma variável dinâmica do tipo FILE*, criada pela função *fopen()*

Função da biblioteca de E/S

Relembrando!



Exemplo de interface

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[], char* envp[])
{
    FILE *arq ;
    char c ;

    arq = fopen ("infos.dat", "r") ; /* abertura do arquivo em leitura */

    if (! arq) /* referencia de arquivo invalida */
    {
        perror ("Erro ao abrir arquivo") ;
        exit (1) ;
    }

    while (1)
    {
        c = getc (arq) ; /* le um caractere do arquivo */
        if (feof (arq)) /* chegou ao final do arquivo? */
            break ;
        putchar(c) ;      /* imprime o caractere na tela */
    }

    fclose (arq) ;      /* fecha o arquivo */
    exit (0) ;
}
```

A decorative graphic on the left side of the slide. It consists of a vertical light blue bar, a vertical orange bar, and a grey rectangle. A horizontal dark blue bar extends from the grey rectangle across the top of the slide.

Exemplo de interface Python

Exemplo de interface

```
##Script para leitura da porta serial
```

```
...
```

```
filename="/mnt/Pessoal/Poli/myfile.txt"
```

```
datafile=open(filename, 'a')
```

```
while True:
```

```
    for i in ser.read():
```

```
        seq.append(i) ## convert from ACSII?
```

```
        joined_seq = ".join(str(v) for v in seq) ## Make a string from array
```

```
## append a timestamp to each row of data
```

```
    datafile.write(str(time.time()) + " " + str(i) + "\n')
```

```
    seq = []
```

```
    count += 1
```

```
    break
```

```
datafile.close()
```

```
ser.close()
```


Exemplo de interface

Mode	Description
'r'	This is the default mode. It Opens file for reading.
'w'	This Mode Opens file for writing. If file does not exist, it creates a new file. If file exists it truncates the file.
'x'	Creates a new file. If file already exists, the operation fails.
'a'	Open file in append mode. If file does not exist, it creates a new file.
't'	This is the default mode. It opens in text mode.
'b'	This opens in binary mode.
+'	This will open a file for reading and writing (updating).

Mais operações...em C

Abertura e fechamento de arquivos

- `FILE * fopen (const char *filename, const char *opentype)`: abre o arquivo; a forma de abertura (leitura, escrita, etc.) é indicada pelo parâmetro `opentype`; em caso de sucesso, devolve uma referência ao arquivo
- `int fclose (FILE *f)`: fecha o arquivo referenciado por `f`

Leitura e escrita de caracteres e strings

- `int fputc (int c, FILE *f)`: escreve um caractere no arquivo
- `int fgetc (FILE *f)`: lê um caractere do arquivo

Mais operações...em C

Reposicionamento do ponteiro do arquivo:

- `long int ftell (FILE *f)`: indica a posição atual do ponteiro do arquivo referenciado por `f`
- `int fseek (FILE *f, long int offset, int whence)`: move o ponteiro do arquivo para a posição indicada por `offset`
 - *whence*: `SEEK_SET`, `SEEK_CUR`, ou `SEEK_END`
- `void rewind (FILE *f)`: retorna o ponteiro do arquivo à sua posição inicial
- `int feof (FILE *f)`: indica se o ponteiro chegou ao final do arquivo

Mais operações...em C

Tratamento de travas:

- `void flockfile (FILE *f)`: solicita acesso exclusivo ao arquivo, podendo bloquear o processo solicitante caso o arquivo já tenha sido reservado por outro processo
- `void funlockfile (FILE *f)`: libera o acesso ao arquivo

Ex.: Determinando o tamanho de um arquivo

```
#include <stdio.h>
#include <stdlib.h>

long filesize( FILE *fp )
{
    long int save_pos;
    long size_of_file;

    /* Save the current position. */
    save_pos = ftell( fp );

    /* Jump to the end of the file. */
    fseek( fp, 0L, SEEK_END );

    /* Get the end position. */
    size_of_file = ftell( fp );

    /* Jump back to the original position. */
    fseek( fp, save_pos, SEEK_SET );

    return( size_of_file );
}
```

Ex.: Determinando o tamanho de um arquivo

```
int main( void )
{
    FILE *fp;

    fp = fopen( "file", "r" );

    if( fp != NULL ) {
        printf( "File size=%ld\n", filesize( fp ) );
        fclose( fp );

        return EXIT_SUCCESS;
    }

    return EXIT_FAILURE;
}
```

Ex.: Determinando o tamanho de um arquivo

```
int main( void )
{
    FILE *fp;

    fp = fopen( "file", "r" );

    if( fp != NULL ) {
        printf( "File size=%ld\n", filesize( fp ) );
        fclose( fp );

        return EXIT_SUCCESS;
    }

    return EXIT_FAILURE;
}
```

Nota: rewind (FILE *f) poderia ter sido utilizada, porém, essa função não possui retorno. Portanto, não se pode testar se foi executada corretamente!

Exercício: Qual o resultado da execução desse código, caso o conteúdo do arquivo file.txt seja exercícioSO ?

```
#include <stdio.h>
```

```
int main () {  
    char str[] = " exercícioSO ";  
    FILE *fp;  
    int ch;  
  
    /* Primeiro, escreve algo no arquivo */  
    fp = fopen( "file.txt" , "w" );  
    fwrite(str , 1 , sizeof(str) , fp );  
    fclose(fp);  
  
    fp = fopen( "file.txt" , "r" );  
    while(1) {  
        ch = fgetc(fp);  
        if( feof(fp) ) {  
            break ;  
        }  
        printf("%c", ch);  
    }  
}
```

```
rewind(fp);  
printf("\n");  
while(1) {  
    ch = fgetc(fp);  
    if( feof(fp) ) {  
        break ;  
    }  
    printf("%c", ch);  
}  
fclose(fp);  
  
return(0);  
}
```