

---

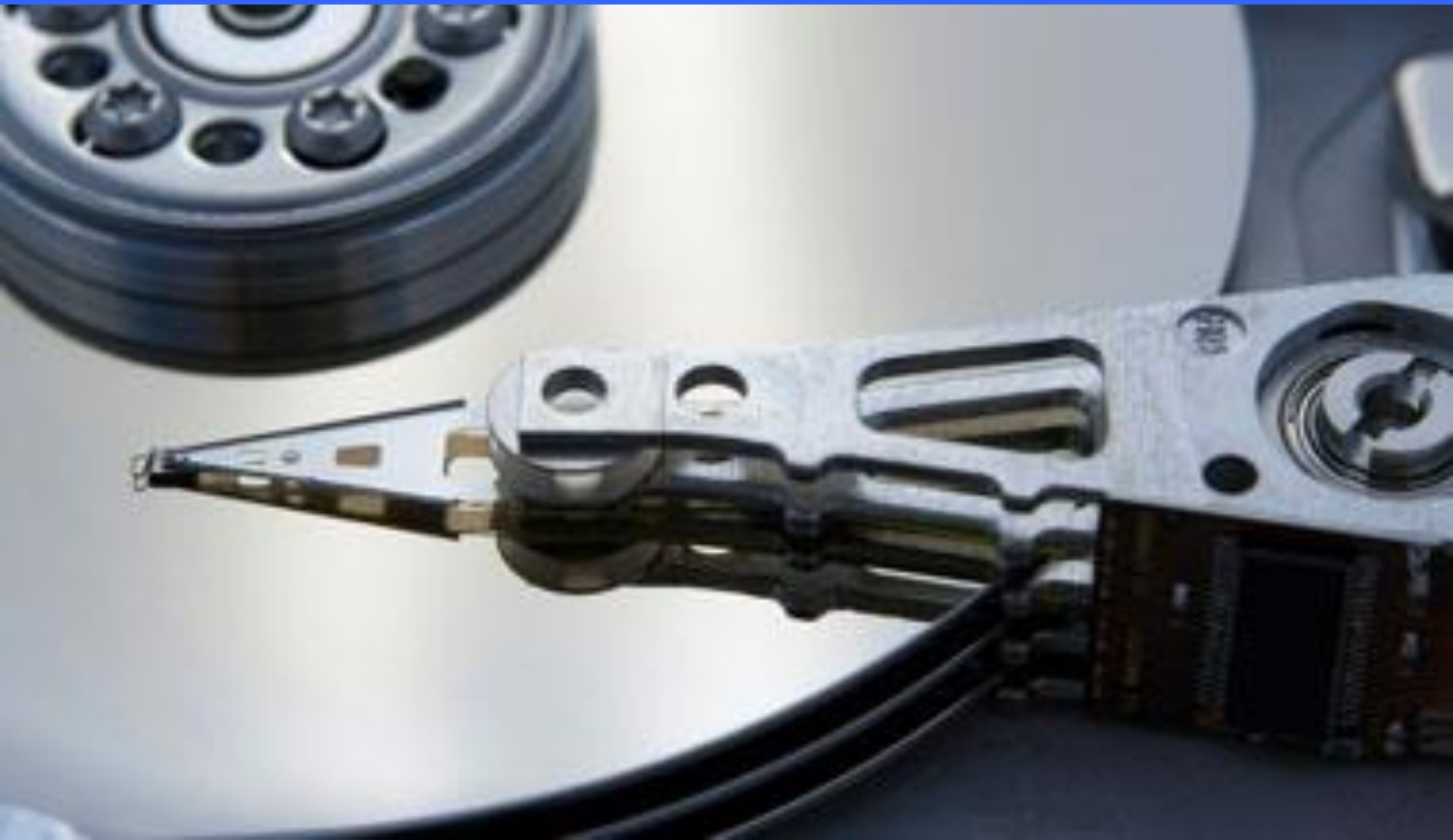
# **Sistemas Operacionais**

## **Gerência de Arquivos**

### **Alocação Física de Arquivos**

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

# Alocação Física de Arquivos



# Conteúdo

---

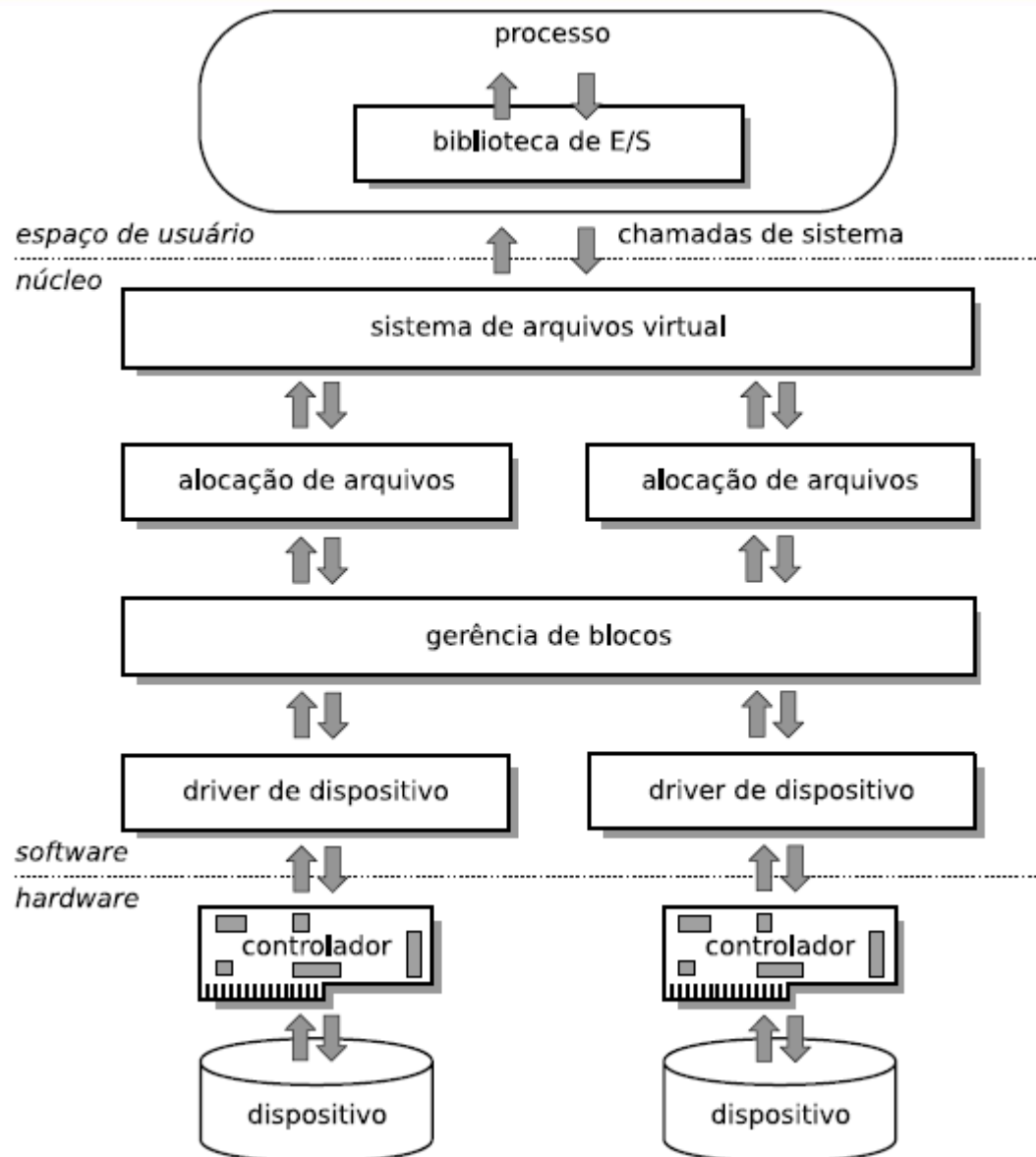
- Arquitetura geral
- Gerenciamento de espaço livre
- Alocação de arquivos
  - Contígua
  - Encadeada
    - Tabela de alocação de arquivos
  - Alocação indexada

# Conteúdo

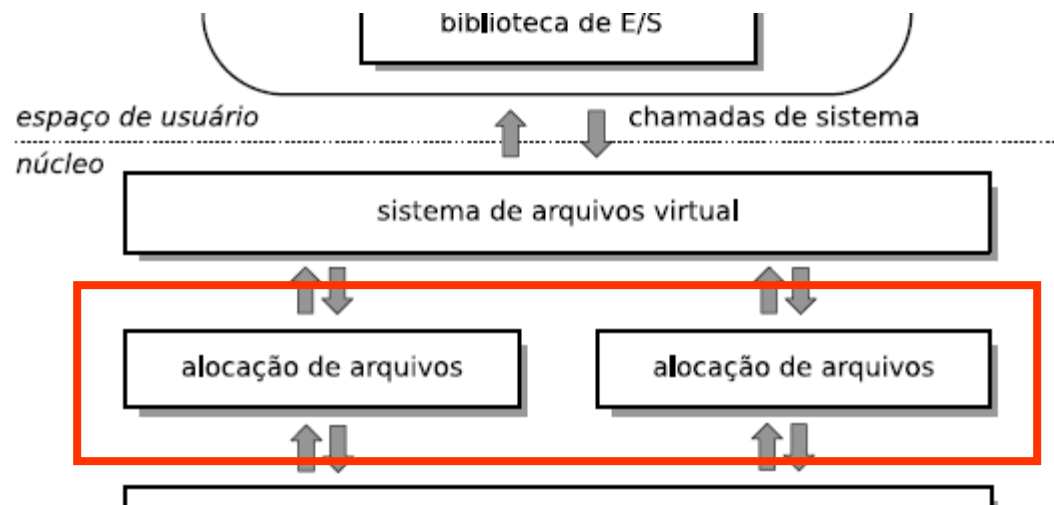
---

- **Arquitetura geral**
- Gerenciamento de espaço livre
- Alocação de arquivos
  - Contígua
  - Encadeada
    - Tabela de alocação de arquivos
  - Alocação indexada

# Arquitetura Geral

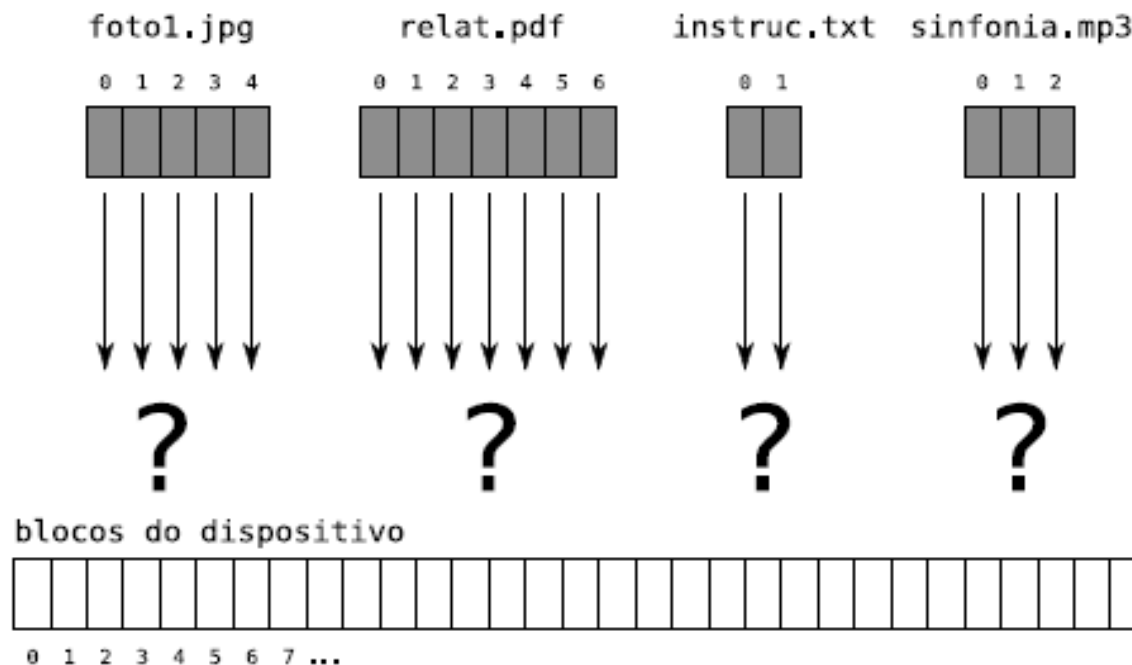


# Alocação Física de Arquivos



# Alocação Física de Arquivos

O problema da **alocação** de arquivos consiste em alocar o **conteúdo** e os **meta-dados** dos arquivos dentro dos **blocos lógicos**.



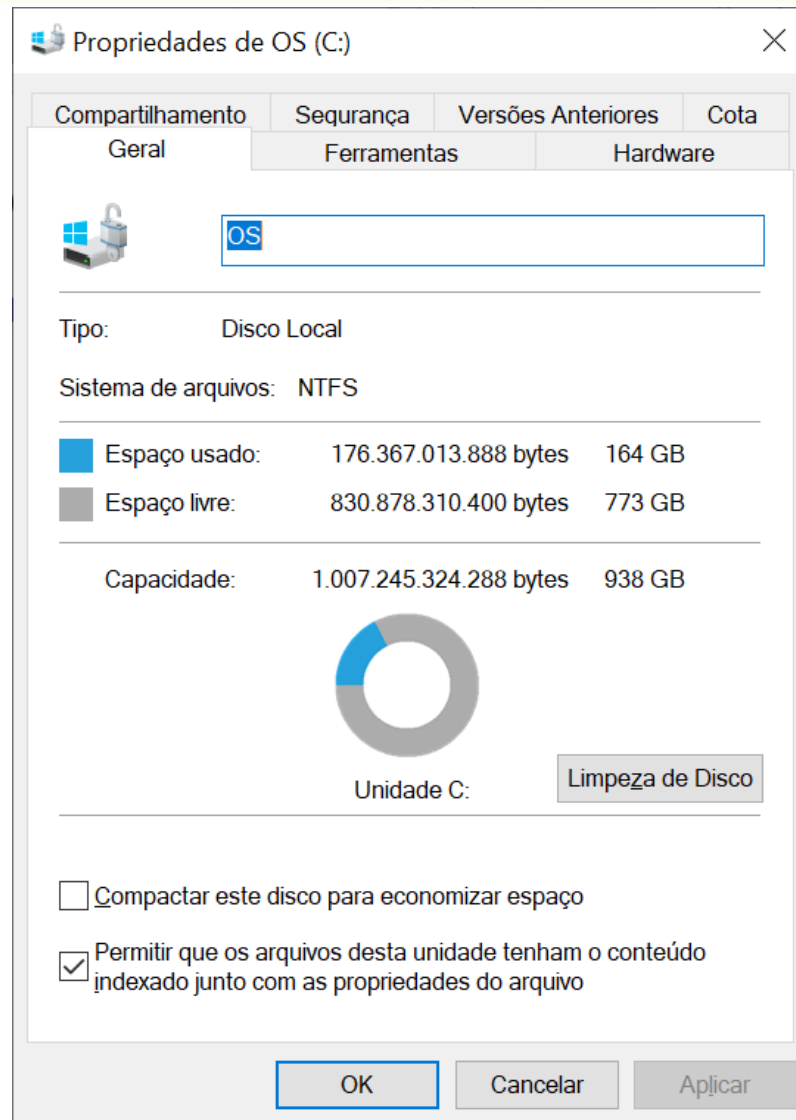
# Conteúdo

---

- Arquitetura geral
- **Gerenciamento de espaço livre**
- Alocação de arquivos
  - Contígua
  - Encadeada
    - Tabela de alocação de arquivos
  - Alocação indexada



# Gerenciamento de espaço em disco





# Gerenciamento de espaço em disco

---

- *Bit Map*
- Lista Encadeada
- Tabela de blocos livres

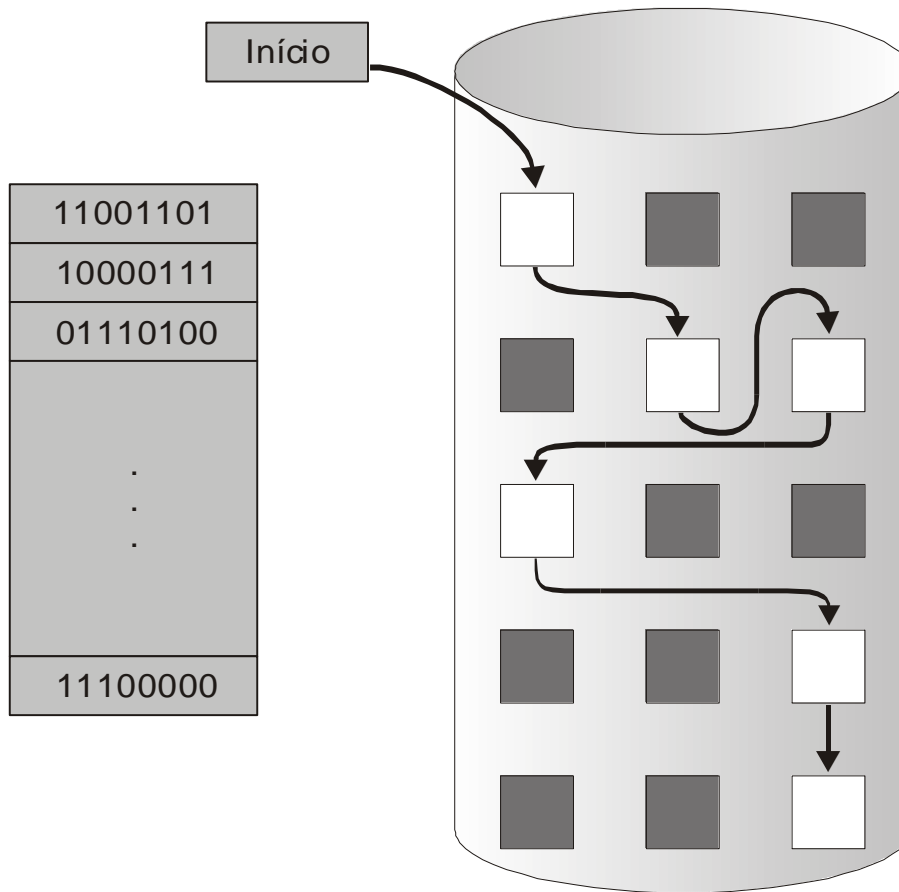
# Gerenciamento de espaço em disco



|          |
|----------|
| 11001101 |
| 10000111 |
| 01110100 |
| ⋮        |
| 11100000 |

(a) Mapa de bits

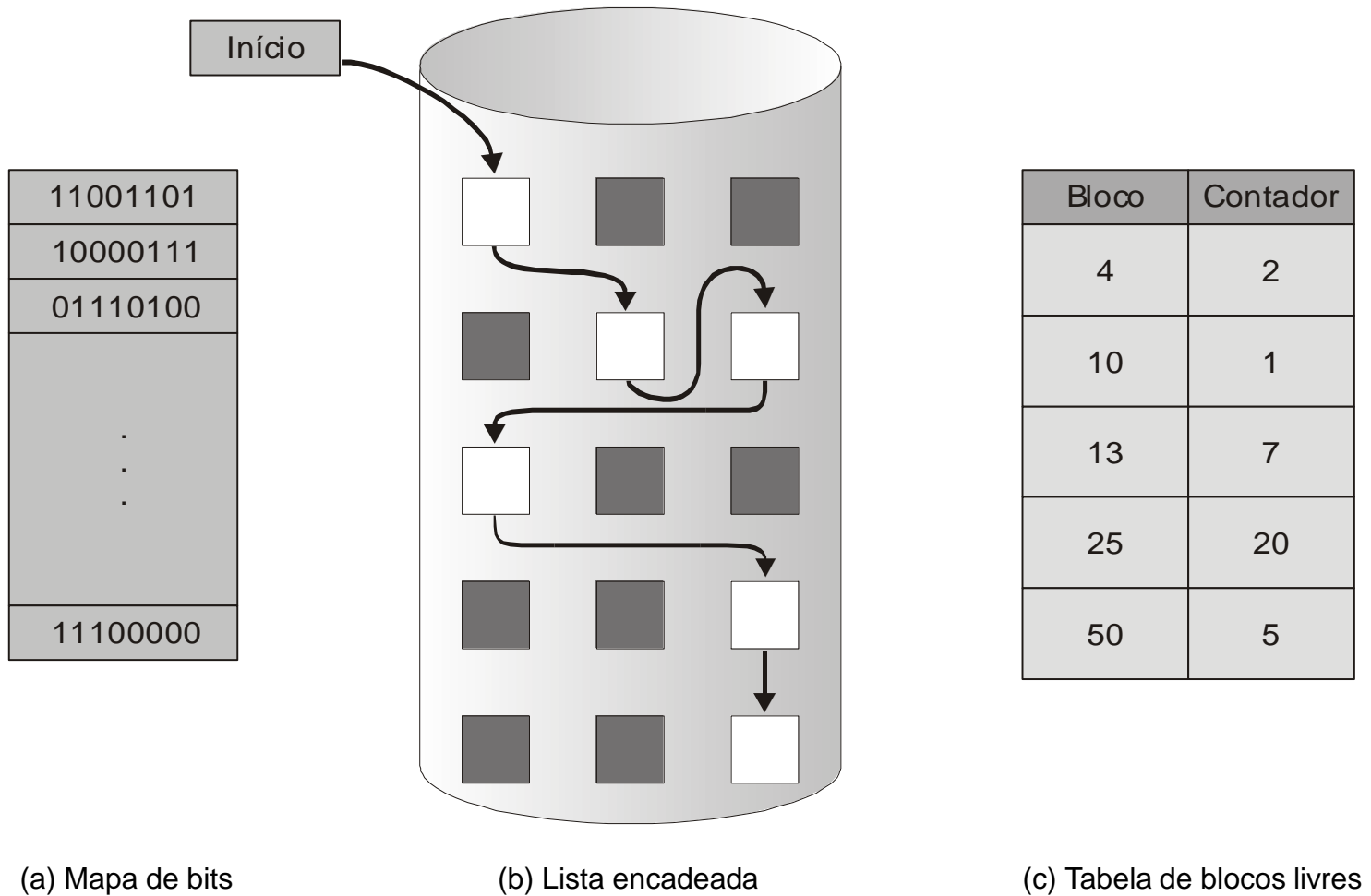
# Gerenciamento de espaço em disco



(a) Mapa de bits

(b) Lista encadeada

# Gerenciamento de espaço em disco



# Conteúdo

---

- Arquitetura geral
- Gerenciamento de espaço livre
- **Alocação de arquivos**
  - Contígua
  - Encadeada
    - Tabela de alocação de arquivos
  - Alocação indexada

# Dados / Meta-dados

---

Na implementação de um **sistema de arquivos**, considera-se que cada arquivo possui **dados** e **meta-dados**

- **Dados:** são o **conteúdo** em si (uma música, uma fotografia, um texto ou uma planilha)
- **Meta-dados:** são seus **atributos** (nome, datas, permissões de acesso, etc.)

# Alocação Física de Arquivos

---

A alocação de um arquivo tem como ponto de partida a definição de um bloco de controle de arquivo

- (FCB - *File Control Block*)
  - Meta-dados
  - Localização



# Alocação Física de Arquivos

---

## Definição:

**Bloco de controle de arquivo** (FCB - *File Control Block*) é uma **estrutura** contendo os **meta-dados** do arquivo e a **localização de seu conteúdo** no disco

# Alocação Física de Arquivos

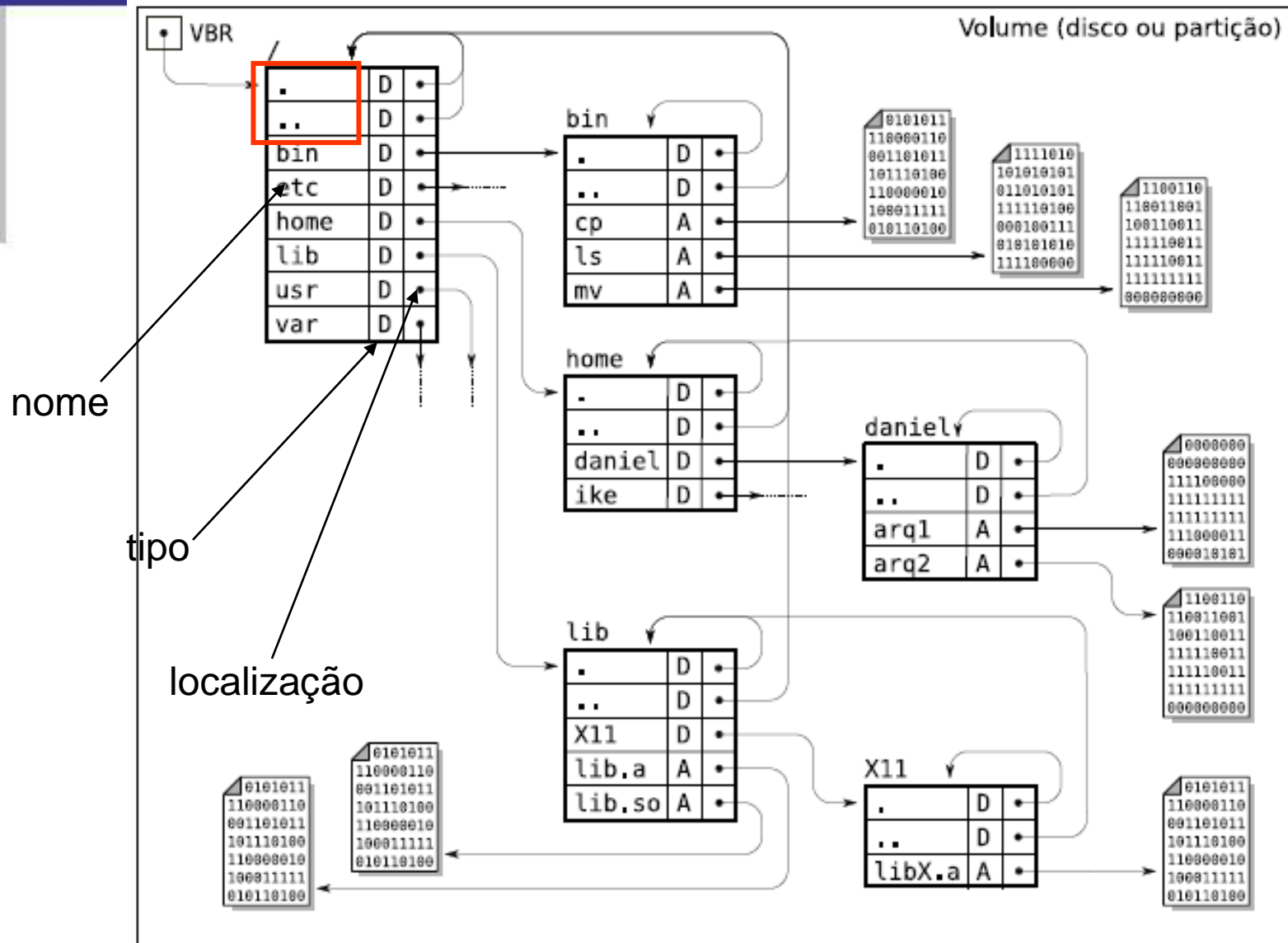
---

## Definição:

**Bloco de controle de arquivo** (FCB - *File Control Block*) é uma **estrutura** contendo os **meta-dados** do arquivo e a **localização de seu conteúdo** no disco

Em sistemas de arquivos mais simples (ex.: FAT) o FCB é pequeno o bastante e cabe na entrada correspondente ao arquivo, na **tabela de diretório**

# Implementação de Diretórios



# Alocação Física de Arquivos

---

Em sistemas de arquivos mais complexos, os blocos de controle de arquivos são definidos em estruturas separadas

- *Master File Table* do sistema NTFS
- *i-nodes* dos sistemas UNIX

# Alocação Física de Arquivos

Em sistemas de arquivos mais complexos, os blocos de controle de arquivos são definidos em estruturas separadas

- *Master F*
- *i-nodes* (

## INODE Overview

|       |                                    |
|-------|------------------------------------|
| 0     | FIL Header (38)                    |
| 38    | List node for INODE Page list (12) |
| 50    | INODE 0 (192)                      |
| 242   | INODE 1 (192)                      |
| 434   | INODE 2 (192)                      |
| 626   | ...                                |
| 15986 | INODE 83 (192)                     |
| 16178 | INODE 84 (192)                     |
| 16370 | (Empty Space, 6 bytes)             |
| 16376 | FIL Trailer (8)                    |
| 16384 |                                    |

# Alocação Física de Arquivos

---

Há **três estratégias** usuais de alocação de arquivos nos blocos lógicos do disco

- Contígua
- Encadeada
- Indexada

# Alocação Física de Arquivos

---

## **Crítérios de análise:**

- **Desempenho** no acesso aos dados do arquivo
  - Sequencial e Direto

# Alocação Física de Arquivos

---

## CrITÉrios de análise:

- **Desempenho** no acesso aos dados do arquivo
  - Sequencial e Direto
- **Robustez** frente a erros
  - Blocos defeituosos e Dados corrompidos



# Alocação Física de Arquivos

---

## **Critérios de análise:**

- **Desempenho** no acesso aos dados do arquivo
  - Sequencial e Direto
- **Robustez** frente a erros
  - Blocos defeituosos e Dados corrompidos
- **Flexibilidade**
  - Criação, Modificação e Exclusão de arquivos e diretórios

# Conteúdo

---

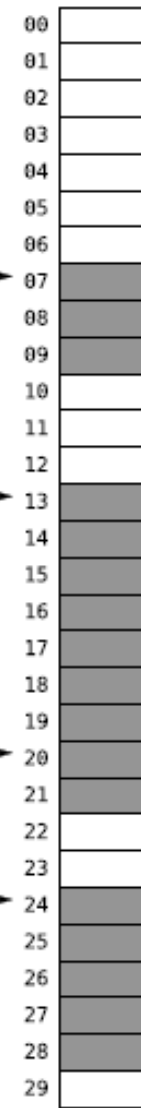
- Arquitetura geral
- Gerenciamento de espaço livre
- **Alocação de arquivos**
  - **Contígua**
  - Encadeada
    - Tabela de alocação de arquivos
  - Alocação indexada

# Alocação Contígua

Tabela de diretório

| nome         | bytes | blocos | início |
|--------------|-------|--------|--------|
| foto1.jpg    | 10417 | 3      | 7 •    |
| relat.pdf    | 28211 | 7      | 13 •   |
| instruc.txt  | 6214  | 2      | 20 •   |
| sinfonia.mp3 | 19116 | 5      | 24 •   |

blocos lógicos com 4096 bytes



# Relembrando

---

## **Critérios:**

**rapidez** no acesso aos dados do arquivo

**robustez** frente a erros como blocos defeituosos e dados corrompidos

**flexibilidade** para a criação, modificação e exclusão de arquivos e diretórios

# Alocação Contígua

**Algoritmo:** Localizar a posição do  $i$ -ésimo byte do arquivo no disco

$i$ : número do byte a localizar

$B$ : tamanho dos blocos lógicos, em bytes

$b_0$ : número do bloco do disco onde o arquivo inicia

# Alocação Contígua

**Algoritmo:** Localizar a posição do  $i$ -ésimo byte do arquivo no disco

$i$ : número do byte a localizar

$B$ : tamanho dos blocos lógicos, em bytes

$b_0$ : número do bloco do disco onde o arquivo inicia

$b_i$ : número do bloco do disco onde se encontra o byte  $i$

$o_i$ : posição do byte  $i$  dentro do bloco  $b_i$  (offset)

# Alocação Contígua

**Algoritmo:** Localizar a posição do  $i$ -ésimo byte do arquivo no disco

$i$ : número do byte a localizar

$B$ : tamanho dos blocos lógicos, em bytes

$b_0$ : número do bloco do disco onde o arquivo inicia

$b_i$ : número do bloco do disco onde se encontra o byte  $i$

$o_i$ : posição do byte  $i$  dentro do bloco  $b_i$  (offset)

$\div \rightarrow$  divisão inteira

$\text{mod} \rightarrow$  módulo (resto da divisão inteira)

$$b_i = b_0 + i \div B$$

$$o_i = i \bmod B$$

return ( $b_i$ ;  $o_i$ )

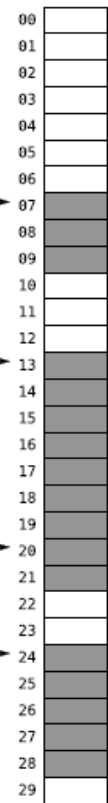
# Alocação Contígua

O byte de número **14.372** do arquivo *relat.pdf* está na posição **2.084** do bloco **16** do disco rígido

Tabela de diretório

| nome         | bytes | blocos | início |
|--------------|-------|--------|--------|
| foto1.jpg    | 10417 | 3      | 7 •    |
| relat.pdf    | 28211 | 7      | 13 •   |
| instruc.txt  | 6214  | 2      | 20 •   |
| sinfonia.mp3 | 19116 | 5      | 24 •   |

blocos lógicos com 4096 bytes



$$bi = 13 + 14.372 \div 4.096$$

$$oi = 14.372 \bmod 4.096$$



# Análise

---

## Rapidez

Acesso **sequencial é rápido** por exigir pouca movimentação da cabeça de leitura do disco

O acesso **direto é relativamente rápido** pois a posição de cada byte do arquivo pode ser facilmente calculada a partir da posição do bloco inicial

# Análise

---

## **Alta robustez a falhas de disco:**

Caso um bloco apresente defeito apenas o conteúdo daquele bloco é perdido

# Análise

---

## **Alta robustez a falhas de disco:**

Caso um bloco apresente defeito apenas o conteúdo daquele bloco é perdido

## **Baixa flexibilidade:**

O tamanho final de cada arquivo precisa ser conhecido no momento de sua criação

Está sujeita à fragmentação externa

# Conteúdo

---

- Arquitetura geral
- Gerenciamento de espaço livre
- **Alocação de arquivos**
  - Contígua
  - **Encadeada**
    - Tabela de alocação de arquivos
  - Alocação indexada

# Alocação Encadeada

---

Esta forma de alocação foi proposta para **contornar** a **pouca flexibilidade** da alocação contígua e **eliminar** a **fragmentação externa**

# Alocação Encadeada

---

Esta forma de alocação foi proposta para **contornar** a **pouca flexibilidade** da alocação contígua e **eliminar** a **fragmentação externa**

Cada bloco do arquivo no disco contém dados do arquivo e também um ponteiro para o próximo bloco

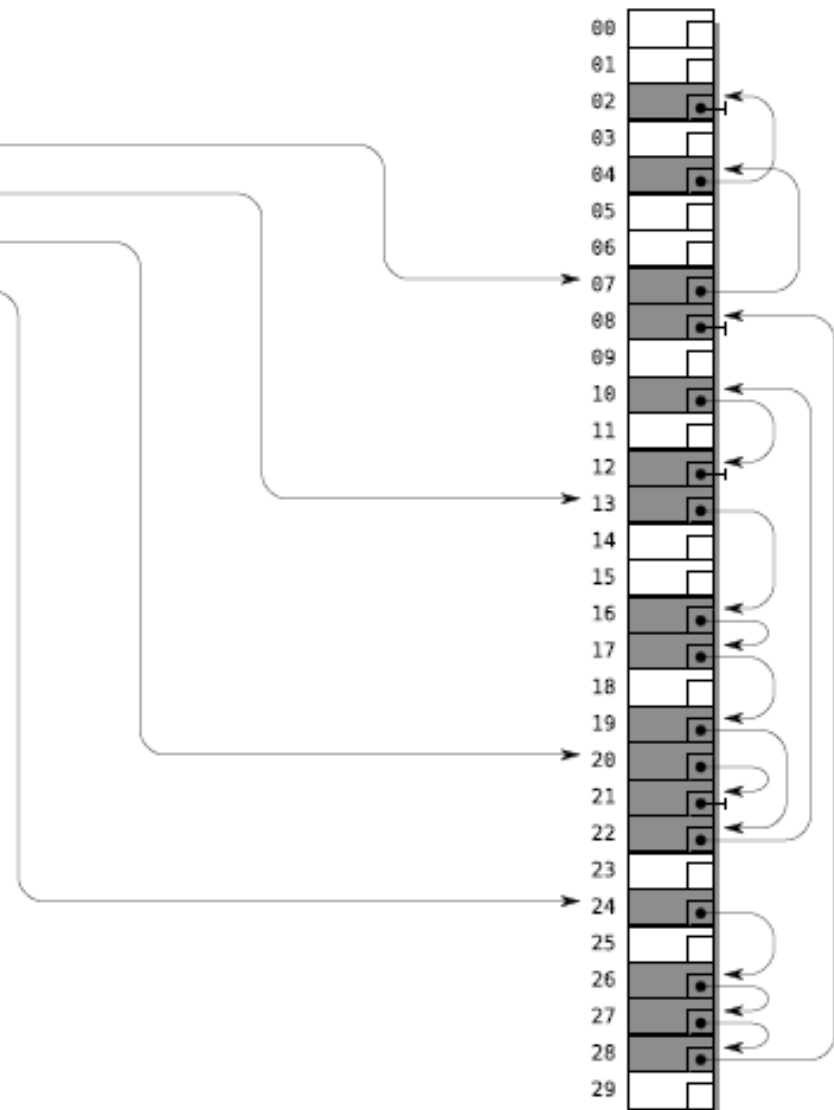
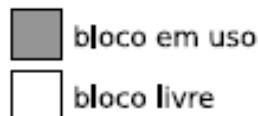
- Um campo indicando o número do próximo bloco do arquivo no disco

# Alocação Encadeada

Tabela de diretório

| nome         | bytes | bloco | início |
|--------------|-------|-------|--------|
| foto1.jpg    | 10417 | 3     | 7 •    |
| relat.pdf    | 28211 | 7     | 13 •   |
| instruc.txt  | 6214  | 2     | 20 •   |
| sinfonia.mp3 | 19116 | 5     | 24 •   |

bloco lógico com 4096 bytes



# Análise

---

## Baixa Rapidez

O **acesso sequencial** aos dados do arquivo é relativamente **simples e rápido**:

Cada bloco contém o ponteiro do próximo bloco do arquivo



# Análise

---

## Baixa Rapidez

O **acesso sequencial** aos dados do arquivo é relativamente simples e rápido:

Cada bloco contém o ponteiro do próximo bloco do arquivo

O **acesso direto** é dispendioso:

Caso se deseje acessar um bloco no meio do arquivo todos os blocos anteriores terão de ser lidos em sequência

# Alocação Encadeada

**Algoritmo:** Localizar a posição do  $i$ -ésimo byte do arquivo no disco

$i$ : número do byte a localizar

$B$ : tamanho dos blocos lógicos, em bytes

$P$ : tamanho do ponteiro de blocos, em bytes

$b_0$ : número do bloco do disco onde o arquivo inicia

$b_i$ : número do bloco do disco onde se encontra o byte  $i$

$o_i$ : posição do byte  $i$  dentro do bloco  $b_i$  (offset)

# Alocação Encadeada

// define bloco inicial do percurso

b\_aux = b0

// calcula número de blocos a percorrer

$Nb = i \div (B - P)$

while Nb > 0 do

    block = read\_block (b\_aux)

    b\_aux = ponteiro extraído de block

    Nb = Nb - 1

end while

# Alocação Encadeada

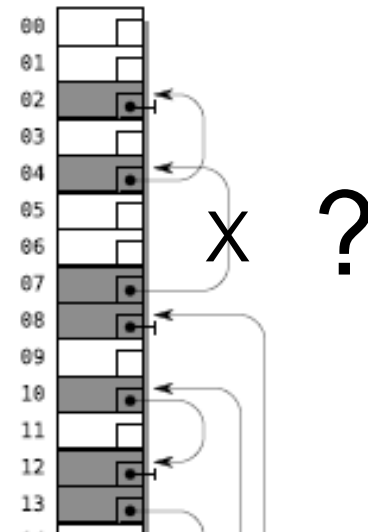
```
// define bloco inicial do percurso
b_aux = b0
// calcula número de blocos a percorrer
Nb = i ÷ (B - P)
while Nb > 0 do
    block = read_block (b_aux)
    b_aux = ponteiro extraído de block
    Nb = Nb - 1
end while
bi = b_aux
oi = i mod (B - P)

return (bi, oi)
```

# Análise

## Baixa robustez

A dependência dos blocos anteriores acarreta **problemas de robustez**: caso um bloco do arquivo seja corrompido todos os blocos posteriores a este também ficarão **inacessíveis**



# Análise

---

## Alta flexibilidade

Não há necessidade de se definir o tamanho máximo do arquivo

Qualquer bloco livre pode ser usado por qualquer arquivo eliminando a **fragmentação externa**

# Análise

---

## Baixa rapidez e Baixa robustez

**CAUSA:** Os **ponteiros** dos blocos são **armazenados nos próprios blocos**, juntamente com os dados do arquivo

# Análise

---

**Baixa rapidez e Baixa robustez**

**CAUSA:** Os **ponteiros** dos blocos são armazenados nos próprios blocos, juntamente com os dados do arquivo

**Solução?**



# Conteúdo

---

- Arquitetura geral
- Gerenciamento de espaço livre
- **Alocação de arquivos**
  - Contígua
  - **Encadeada**
    - **Tabela de alocação de arquivos**
  - Alocação indexada

# Tabela de Alocação de Arquivos

---

- Os ponteiros dos blocos de cada arquivo são mantidos em uma **tabela única**
  - Armazenada no início da partição
- Cada **entrada** da tabela corresponde a um **bloco lógico** e contém um **ponteiro** indicando o próximo bloco do mesmo arquivo

# Tabela de Alocação de Arquivos

---

- As entradas da tabela também podem conter valores especiais para indicar:
  - último bloco de cada arquivo
  - blocos livres
  - blocos defeituosos
  - blocos reservados
- Uma cópia dessa tabela é mantida em *cache* na memória durante o uso do sistema

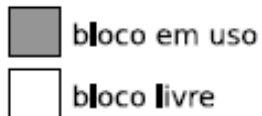
# Tabela de Alocação de Arquivos

Tabela de diretório

| nome         | bytes | blocos | início |
|--------------|-------|--------|--------|
| foto1.jpg    | 10417 | 3      | 7 •    |
| relat.pdf    | 28211 | 7      | 13 •   |
| instruc.txt  | 6214  | 2      | 20 •   |
| sinfonia.mp3 | 19116 | 5      | 24 •   |

blocos lógicos com 4096 bytes

Cópia em cache

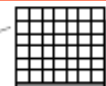


## Entradas da tabela de alocação

- 17 : número do próximo bloco
- L : último bloco do arquivo (Last)
- F : bloco livre (Free)
- R : bloco reservado (Reserved)
- B : bloco defeituoso (Bad)

Tabela de alocação de arquivos

|    |    |
|----|----|
| 00 | R  |
| 01 | R  |
| 02 | L  |
| 03 | F  |
| 04 | 02 |
| 05 | F  |
| 06 | F  |
| 07 | 04 |
| 08 | L  |
| 09 | F  |
| 10 | 12 |
| 11 | F  |
| 12 | L  |
| 13 | 16 |
| 14 | F  |
| 15 | F  |
| 16 | 17 |
| 17 | 19 |
| 18 | F  |
| 19 | 22 |
| 20 | 21 |
| 21 | L  |
| 22 | 10 |
| 23 | F  |
| 24 | 26 |
| 25 | F  |
| 26 | 27 |
| 27 | 28 |
| 28 | 8  |
| 29 | F  |



# Tabela de Alocação de Arquivos

---

## FAT – File Allocation Table

- Base dos sistemas de arquivos FAT12, FAT16 e FAT32 usados nos sistemas operacionais MSDOS e Windows
  - Pen-drives
  - Reprodutores MP3
  - Câmeras fotográficas digitais

# Ex.: FAT 32

|     |             |             |             |             |
|-----|-------------|-------------|-------------|-------------|
| 000 | F8 FF FF 0F | FF FF FF 7F | FF FF FF FF | FF FF FF FF |
| 010 | 05 00 00 00 | 06 00 00 00 | 07 00 00 00 | 08 00 00 00 |
| 020 | 09 00 00 00 | 0A 00 00 00 | 0B 00 00 00 | 0C 00 00 00 |
| 030 | 0D 00 00 00 | 0E 00 00 00 | 0F 00 00 00 | 10 00 00 00 |
| 040 | 11 00 00 00 | 12 00 00 00 | 13 00 00 00 | 14 00 00 00 |
| 050 | 15 00 00 00 | 16 00 00 00 | 17 00 00 00 | 18 00 00 00 |
| ⋮   | ⋮           | ⋮           | ⋮           | ⋮           |
| 1C0 | 71 00 00 00 | 72 00 00 00 | 73 00 00 00 | FF FF FF FF |
| 1D0 | 75 00 00 00 | 76 00 00 00 | 77 00 00 00 | 78 00 00 00 |
| 1E0 | 79 00 00 00 | 7A 00 00 00 | 7B 00 00 00 | 7C 00 00 00 |
| 1F0 | 7D 00 00 00 | 7E 00 00 00 | 7F 00 00 00 | 80 00 00 00 |
|     |             |             |             |             |
| 000 | 81 00 00 00 | 82 00 00 00 | 83 00 00 00 | 84 00 00 00 |
| 010 | 85 00 00 00 | 86 00 00 00 | 87 00 00 00 | 88 00 00 00 |
| 020 | 89 00 00 00 | 8A 00 00 00 | 8B 00 00 00 | 8C 00 00 00 |
| 030 | 8D 00 00 00 | 8E 00 00 00 | 8F 00 00 00 | 90 00 00 00 |
| ⋮   | ⋮           | ⋮           | ⋮           | ⋮           |
| 1D0 | F5 00 00 00 | F6 00 00 00 | F7 00 00 00 | F8 00 00 00 |
| 1E0 | F9 00 00 00 | FA 00 00 00 | FF FF FF FF | FC 00 00 00 |
| 1F0 | FD 00 00 00 | FE 00 00 00 | FF 00 00 00 | 00 01 00 00 |
|     |             |             |             |             |
| 000 | 01 01 00 00 | 02 01 00 00 | 03 01 00 00 | 04 01 00 00 |
| 010 | 05 01 00 00 | 06 01 00 00 | 07 01 00 00 | 08 01 00 00 |
| 020 | 09 01 00 00 | 0A 01 00 00 | 0B 01 00 00 | 0C 01 00 00 |
| 030 | 0D 01 00 00 | 0E 01 00 00 | 0F 01 00 00 | 10 01 00 00 |
| ⋮   | ⋮           | ⋮           | ⋮           | ⋮           |
| 1D0 | 75 01 00 00 | 76 01 00 00 | 77 01 00 00 | 78 01 00 00 |
| 1E0 | 79 01 00 00 | FF FF FF FF | 00 00 00 00 | 00 00 00 00 |
| 1F0 | 00 00 00 00 | 00 00 00 00 | 7F 01 00 00 | 80 01 00 00 |

# Ex.: FAT 32

|     |             |             |             |             |
|-----|-------------|-------------|-------------|-------------|
| 000 | F8 FF FF 0F | FF FF FF 7F | FF FF FF FF | FF FF FF FF |
| 010 | 05 00 00 00 | 06 00 00 00 | 07 00 00 00 | 08 00 00 00 |
| 020 | 09 00 00 00 | 0A 00 00 00 | 0B 00 00 00 | 0C 00 00 00 |
| 030 | 0D 00 00 00 | 0E 00 00 00 | 0F 00 00 00 | 10 00 00 00 |
| 040 | 11 00 00 00 | 12 00 00 00 | 13 00 00 00 | 14 00 00 00 |
| 050 | 15 00 00 00 | 16 00 00 00 | 17 00 00 00 | 18 00 00 00 |
| ... | ...         | ...         | ...         | ...         |
| 1C0 | 71 00 00 00 | 72 00 00 00 | 73 00 00 00 | FF FF FF FF |
| 1D0 | 75 00 00 00 | 76 00 00 00 | 77 00 00 00 | 78 00 00 00 |
| 1E0 | 79 00 00 00 | 7A 00 00 00 | 7B 00 00 00 | 7C 00 00 00 |
| 1F0 | 7D 00 00 00 | 7E 00 00 00 | 7F 00 00 00 | 80 00 00 00 |
| ... |             |             |             |             |
| 000 | 81 00 00 00 | 82 00 00 00 | 83 00 00 00 | 84 00 00 00 |
| 010 | 85 00 00 00 | 86 00 00 00 | 87 00 00 00 | 88 00 00 00 |
| 020 | 89 00 00 00 | 8A 00 00 00 | 8B 00 00 00 | 8C 00 00 00 |
| 030 | 8D 00 00 00 | 8E 00 00 00 | 8F 00 00 00 | 90 00 00 00 |

32 bits → FAT32

Primeiro setor → **F8 FF FF**

Espaço livre → **00 00 00 00**

EoF → **FF FF FF F8 - FF FF FF FF**

Bloco defeituoso → **FF FF FF F7**

|     |             |             |             |             |
|-----|-------------|-------------|-------------|-------------|
| 1D0 | 75 01 00 00 | 76 01 00 00 | 77 01 00 00 | 78 01 00 00 |
| 1E0 | 79 01 00 00 | FF FF FF FF | 00 00 00 00 | 00 00 00 00 |
| 1F0 | 00 00 00 00 | 00 00 00 00 | 7F 01 00 00 | 80 01 00 00 |

# Conteúdo

---

- Arquitetura geral
- Gerenciamento de espaço livre
- **Alocação de arquivos**
  - Contígua
  - Encadeada
    - Tabela de alocação de arquivos
  - **Alocação indexada**



# Alocação Indexada

---

Qual a principal restrição de desempenho da alocação encadeada?

# Alocação Indexada

---

Qual a principal restrição de desempenho da alocação encadeada?

Resp.: O fato de o acesso ser sequencial (encadeado)

# Alocação Indexada

---

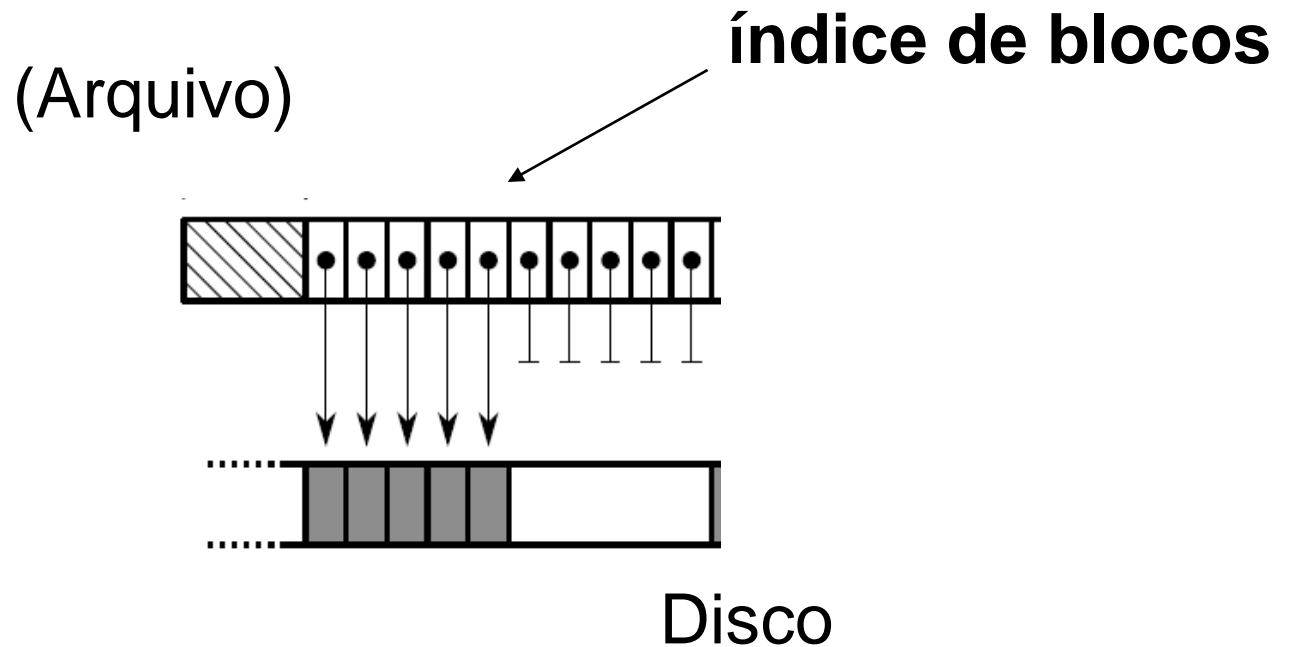
- A estrutura em **lista encadeada** da estratégia anterior é substituída por um **vetor** contendo um **índice de blocos** (*para cada arquivo*)

# Alocação Indexada

---

- A estrutura em **lista encadeada** da estratégia anterior é substituída por um **vetor** contendo um **índice de blocos** (*para cada arquivo*)
- Cada entrada desse **índice** corresponde a um bloco do arquivo e aponta para a posição desse bloco no disco

# Alocação Indexada

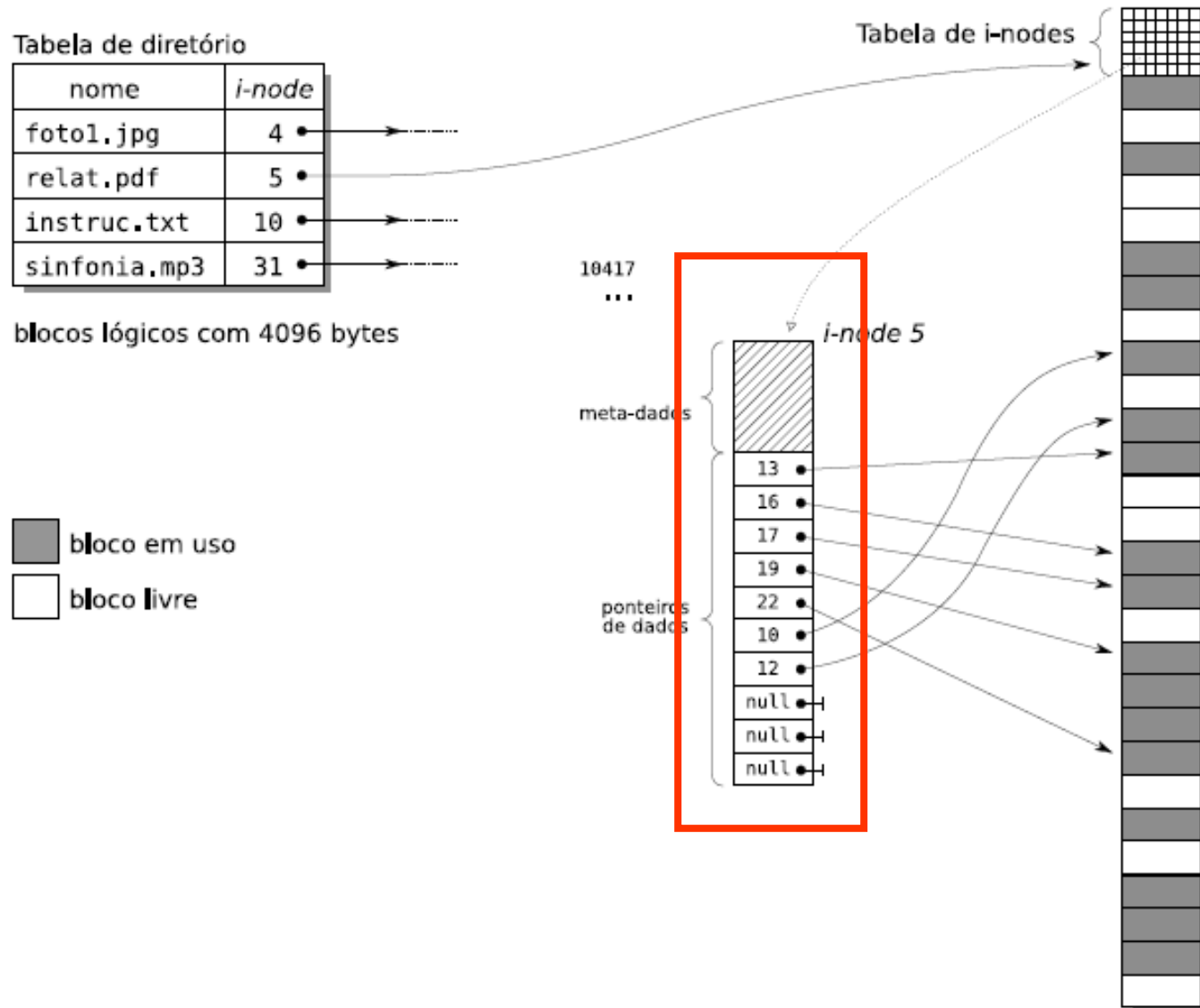


# Alocação Indexada

---

- O **índice de blocos** de cada arquivo é mantido em uma estrutura denominada nó de índices (*index node* ou ***i-node***)
- *i-node* contém:
  - índice de blocos
  - principais atributos do arquivo
    - tamanho, permissões, etc.

# Alocação Indexada



# Alocação Indexada

---

## Problema

Os *i-nodes* têm **tamanho fixo**, o **número de entradas** no índice de blocos de um arquivo é **limitado**

→ **Tamanho máximo** para os arquivos



# Alocação Indexada

---

## Problema

Os *i-nodes* têm **tamanho fixo**, o **número de entradas** no índice de blocos de um arquivo é **limitado**

→ **Tamanho máximo** para os arquivos

A tabela de *i-nodes* também tem **tamanho fixo**

→ **Número máximo** de arquivos ou diretórios

# Alocação Indexada

---

## Problema

- Por exemplo, se o sistema usar blocos de 4 kB e o índice de blocos tiver 64 entradas:
  - só poderão ser armazenados arquivos com até  $64 \times 4\text{kB} = 256\text{ kB}$

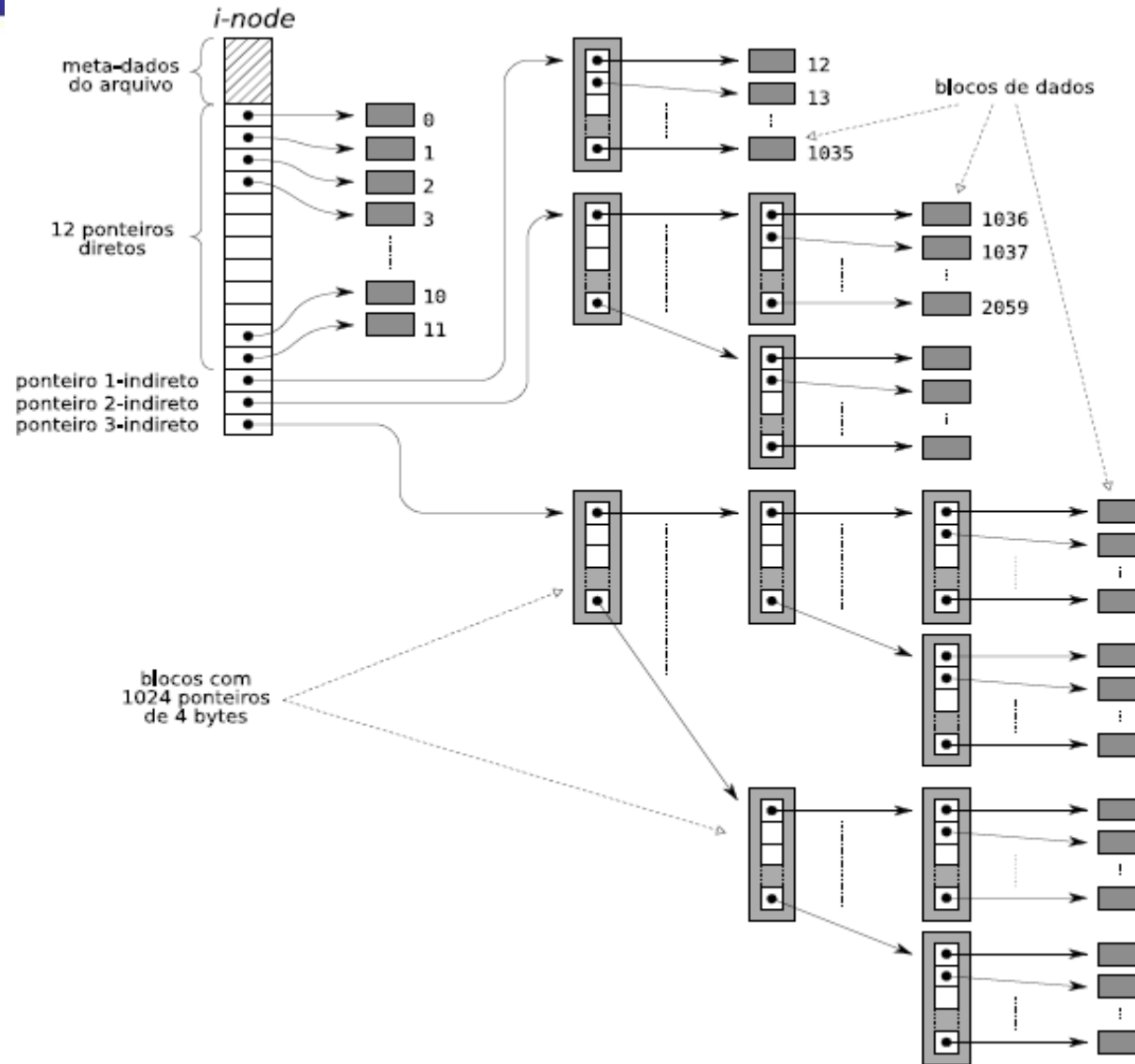
# Alocação indexada multi-nível

---

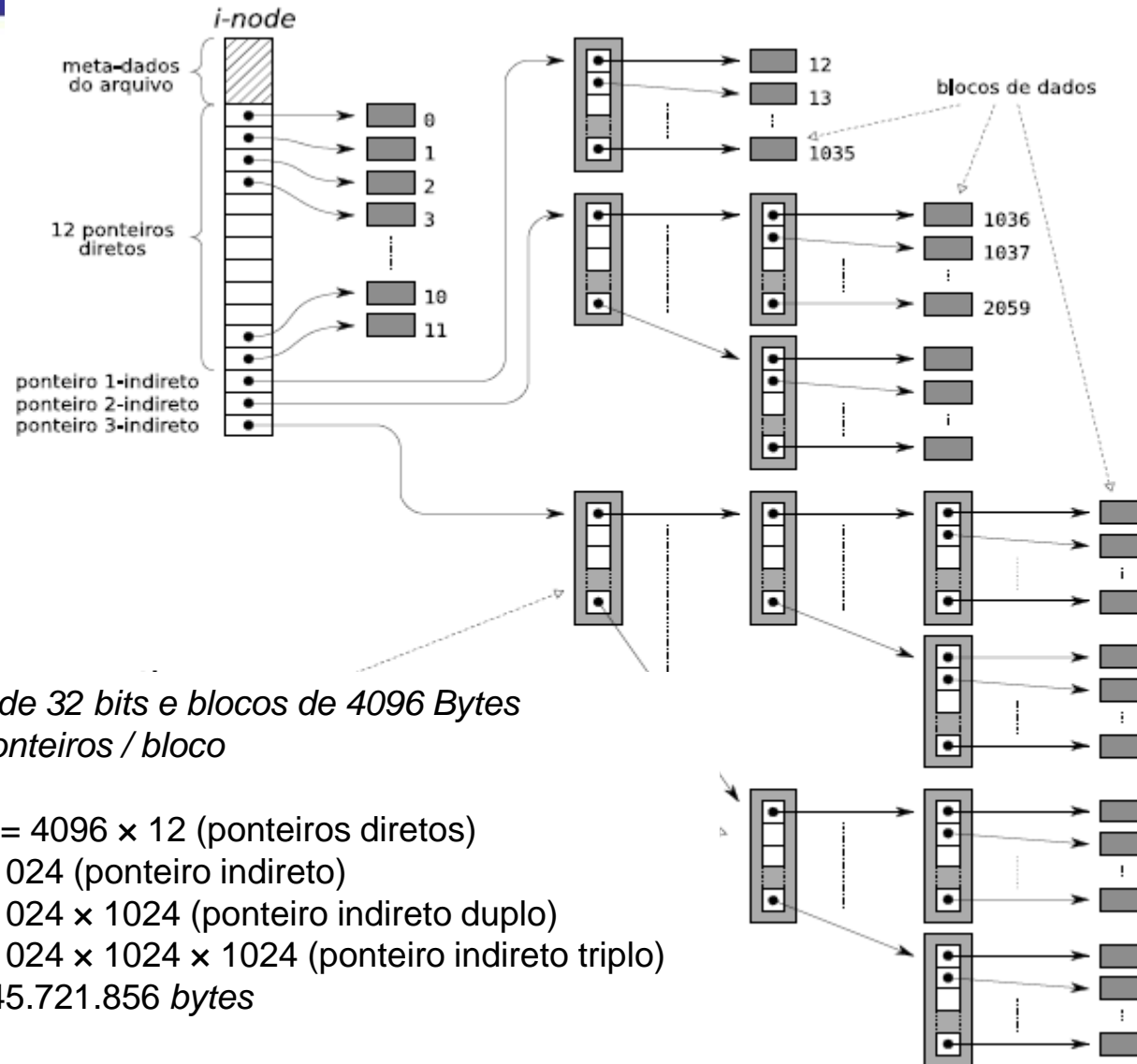
## Solução

Para **aumentar o tamanho máximo** dos arquivos armazenados, algumas das entradas do índice de blocos podem ser transformadas em **ponteiros indiretos**

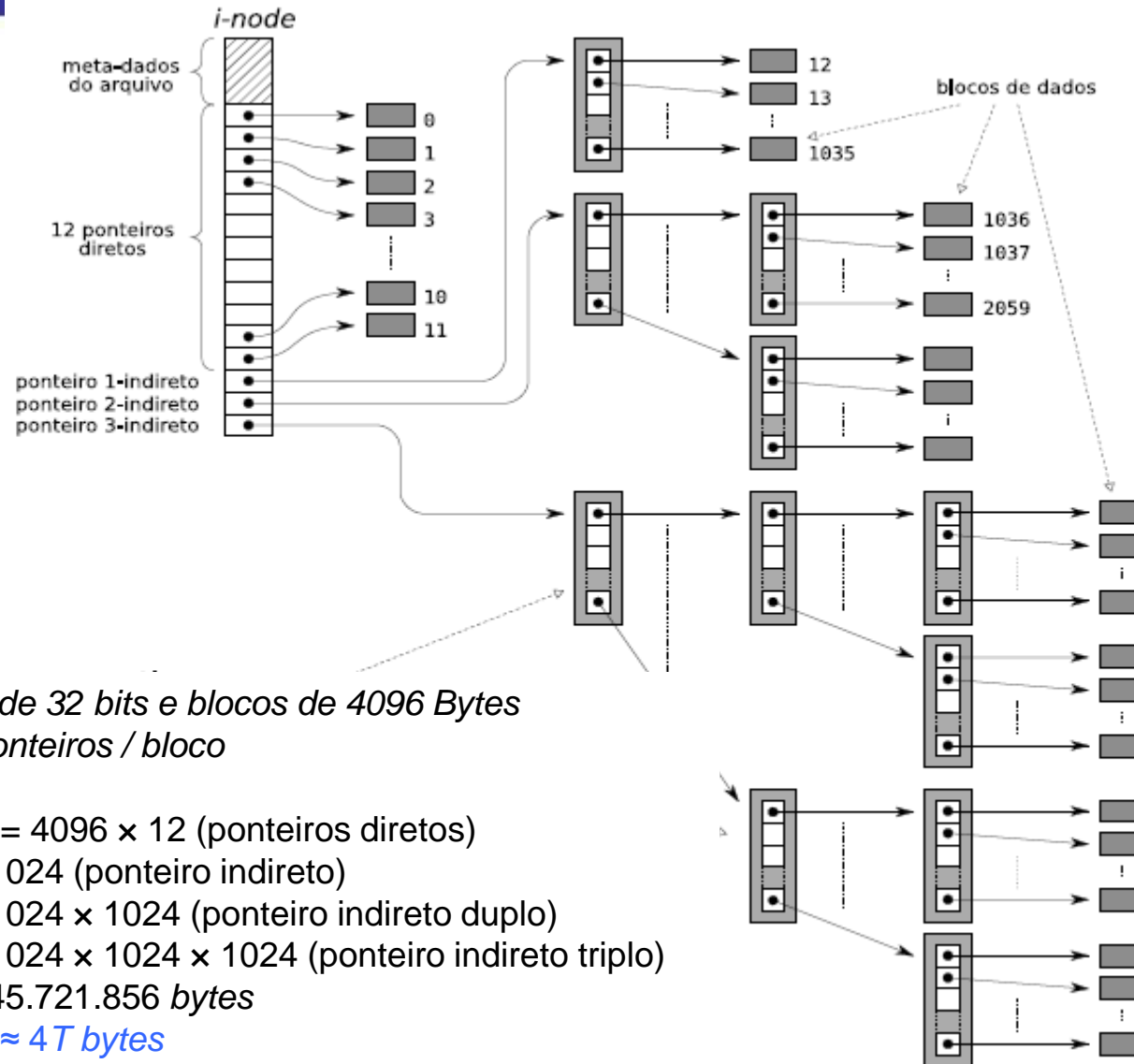
# Exemplo: Ext2/Ext3 do Linux



# Exemplo: Ext2/Ext3 do Linux



# Exemplo: Ext2/Ext3 do Linux



Ponteiros de 32 bits e blocos de 4096 Bytes  
→ 1024 ponteiros / bloco

$Tam. max = 4096 \times 12$  (ponteiros diretos)  
+  $4096 \times 1024$  (ponteiro indireto)  
+  $4096 \times 1024 \times 1024$  (ponteiro indireto duplo)  
+  $4096 \times 1024 \times 1024 \times 1024$  (ponteiro indireto triplo)  
= 4.402.345.721.856 bytes

*Tam. max ≈ 4 T bytes*

# Alocação Indexada

---

Apesar dessa estrutura aparentemente complexa, a localização e acesso de um bloco do arquivo no disco permanece relativamente **simples**

- Estrutura homogênea de ponteiros permite calcular a localização dos blocos com exatidão
- Estratégia rápida para acessos **sequenciais** e acessos **diretos**
  - devido aos índices de ponteiros dos blocos presentes nos *i-nodes*

# Alocação Indexada

---

- **Defeitos** em blocos de dados **não afetam** os demais blocos de dados
- Todavia, defeitos nos meta-dados (o *i-node* ou os blocos de ponteiros) podem danificar grandes extensões do arquivo
- Muitos sistemas que usam esta estratégia implementam técnicas de redundância de *i-nodes* para melhorar a robustez



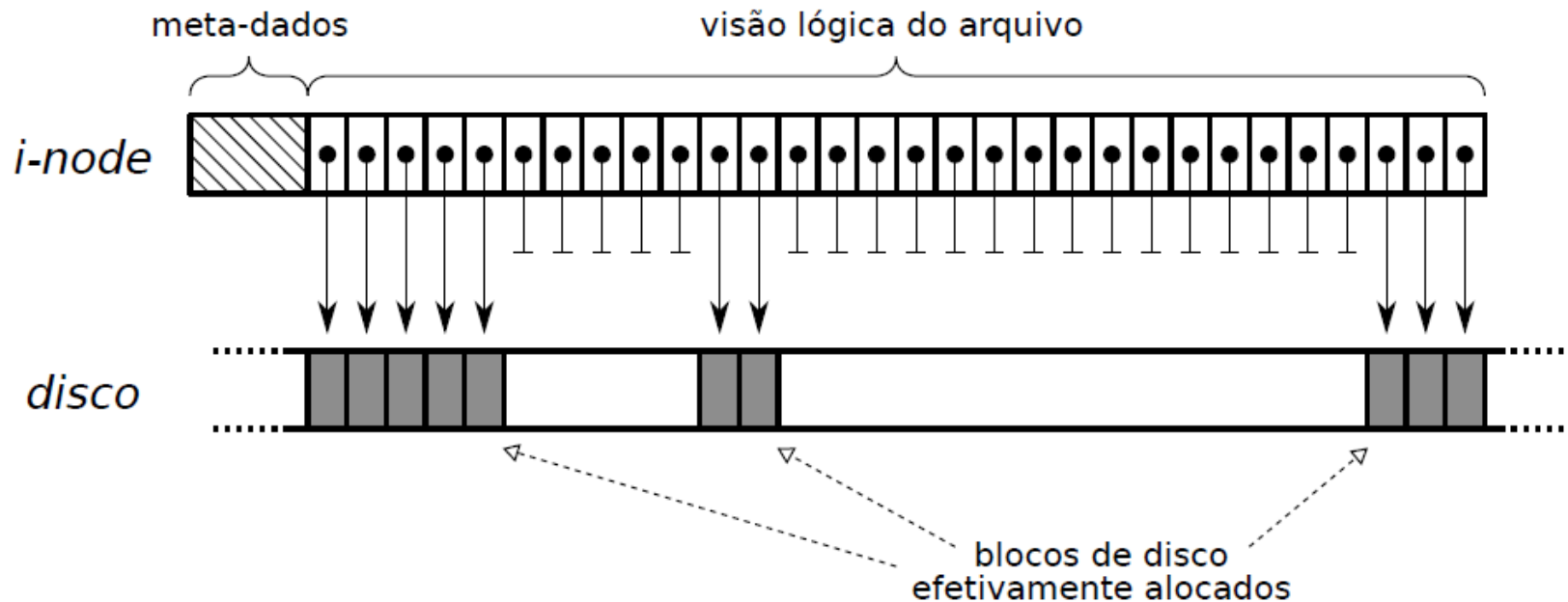
# Alocação Indexada

---

- Tão flexível quanto a alocação encadeada
  - não apresenta fragmentação externa e
  - permite o uso de todas as áreas do disco para armazenar dados
- Porém, o tamanho máximo dos arquivos criados é limitado
- Bem como o número máximo de arquivos na partição

# Alocação Indexada

## Alocação de arquivos esparsos



→ muito usados por gerenciadores de bancos de dados

# Comparação

| Estratégia | Rapidez   | Robustez  | Flexibilidade   |
|------------|---|---|---|
| Contígua   | Alta, pois acessos seqüencial e direto rápidos, pois os blocos do arquivo estão próximos no disco.  | Alta, pois blocos defeituosos não impedem o acesso aos demais blocos do arquivo.  | Baixa, pois o tamanho máximo dos arquivos deve ser conhecido a priori; nem sempre é possível aumentar o tamanho de um arquivo existente.  |
| Encadeada  | Acesso seqüencial é rápido, se os blocos estiverem próximos; o acesso direto é lento, pois é necessário ler todos os blocos a partir do início do arquivo até encontrar o bloco desejado. | Baixa, pois um bloco defeituoso leva à perda dos dados daquele bloco e de todos os blocos subsequentes, até o fim do arquivo. | Alta, pois arquivos podem ser criados em qualquer local do disco, sem risco de fragmentação externa.  |
| FAT        | Alta, pois acessos seqüencial e direto são rápidos, se os blocos do arquivo estiverem próximos no disco.  | Mais robusta que a alocação encadeada, desde que não ocorram erros na tabela de alocação.                                     | Alta, pois arquivos podem ser criados em qualquer local do disco, sem risco de fragmentação externa.  |
| Indexada   | Alta, pois os acessos seqüencial e direto são rápidos, se os blocos do arquivo estiverem próximos no disco.   | Alta, desde que não ocorram erros no <i>i-node</i> nem nos blocos de ponteiros.   | Alta, pois arquivos podem ser criados em qualquer local do disco, sem risco de fragmentação externa. No entanto, o tamanho máximo dos arquivos é limitado pelo número de ponteiros definidos nos <i>i-nodes</i> . |