INE5412 Sistemas Operacionais I

L. F. Friedrich

Sistema de Arquivos – Implementação

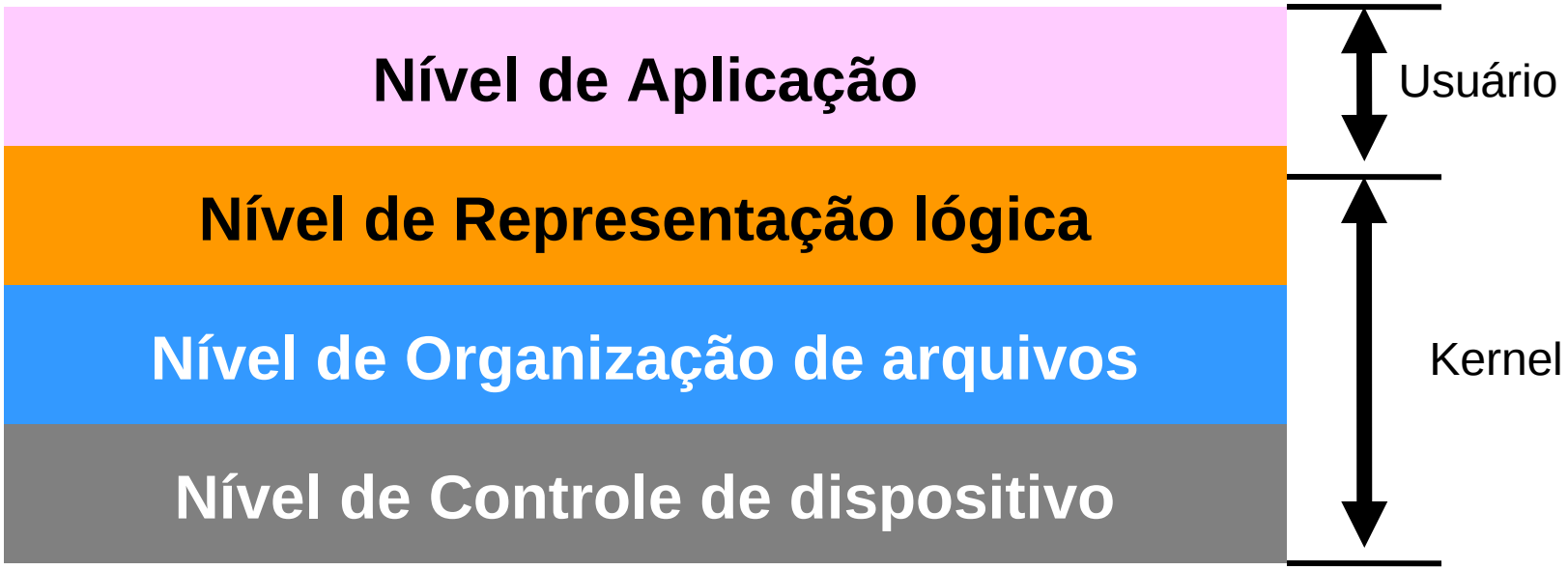
O que é um SA?

- Um sistema de arquivos (SA) trata de:
  - Como o SO armazena e localiza um arquivo?
  - Como o SO armazena e localiza um diretorio?

- Trata de como o SO controla/organiza o espaço que o meio de armazenamento oferece.

- Também trata de como SO controla o espaço de forma eficiente e confiável .
  - Eficiencia tem a ver com velocidade e utilização do armazenamento.
  - Confiabilidade tem a ver com a capacidade de recuperação para estados confiáveis

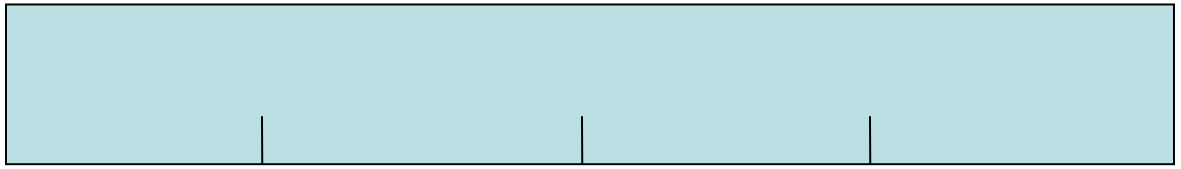
Estrutura de um SA



Organização de arquivos

- Considere o seguinte…voce é o controlador do SA, e voce recebe um *hard disk* vazio.
  - Voce pode usa-lo de qualquer das formas de utilização de disco.
  - Seu objetivo é armazenar arquivos no disco (esqueça sobre diretorio no momento).
  - Voce também quer ler e escrever arquivos armazenados de forma rápida e confiável.

- Como voce organizaria os dados no hard disk?



Alocação contígua

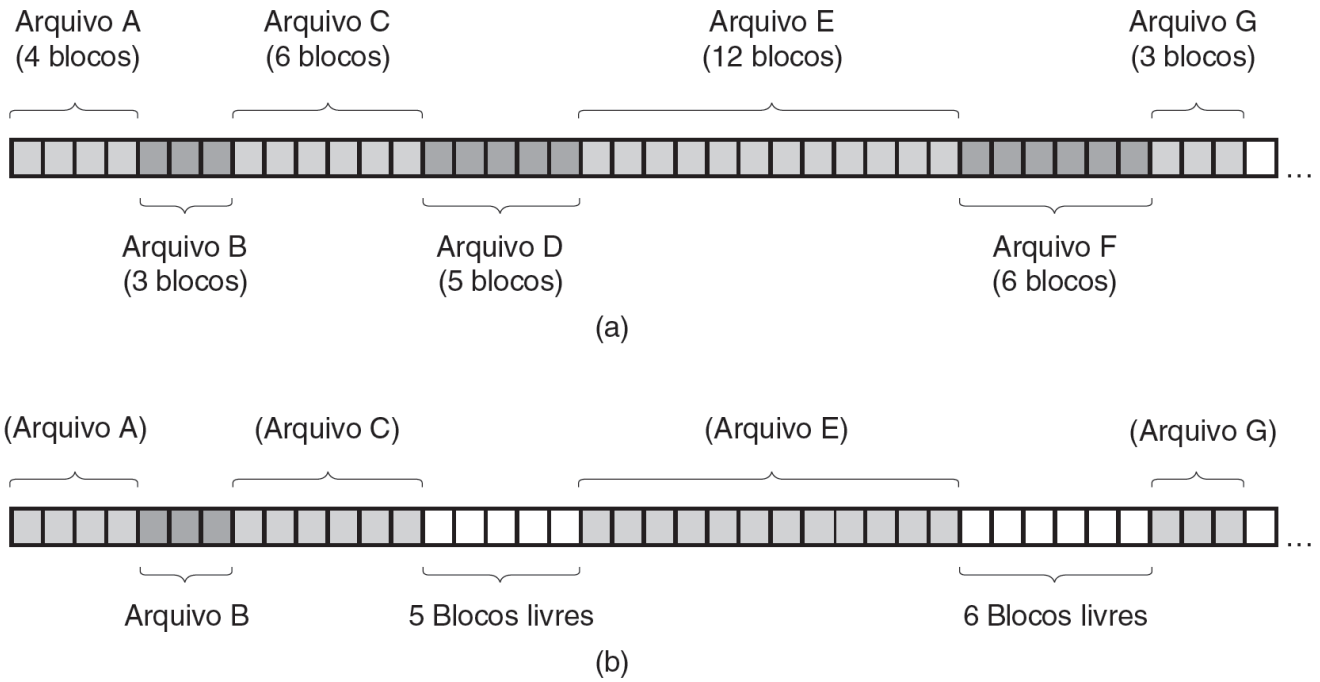
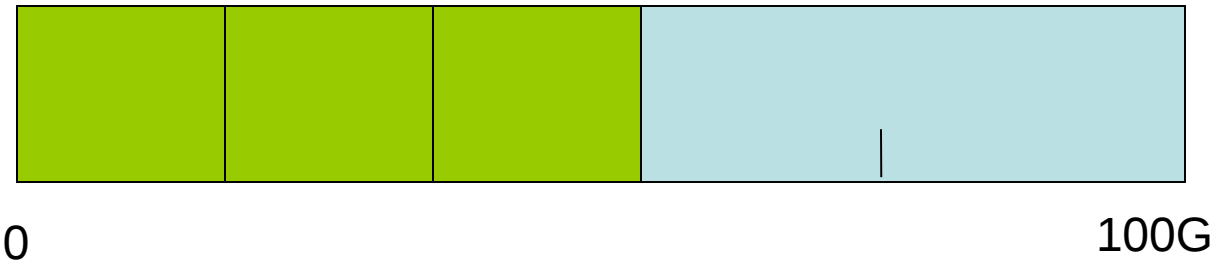


Figura 4.8 (a) A alocação contígua do espaço em disco para sete arquivos. (b) O estado do disco depois de os arquivos *D* e *F* terem sido removidos.

1a. Tentativa

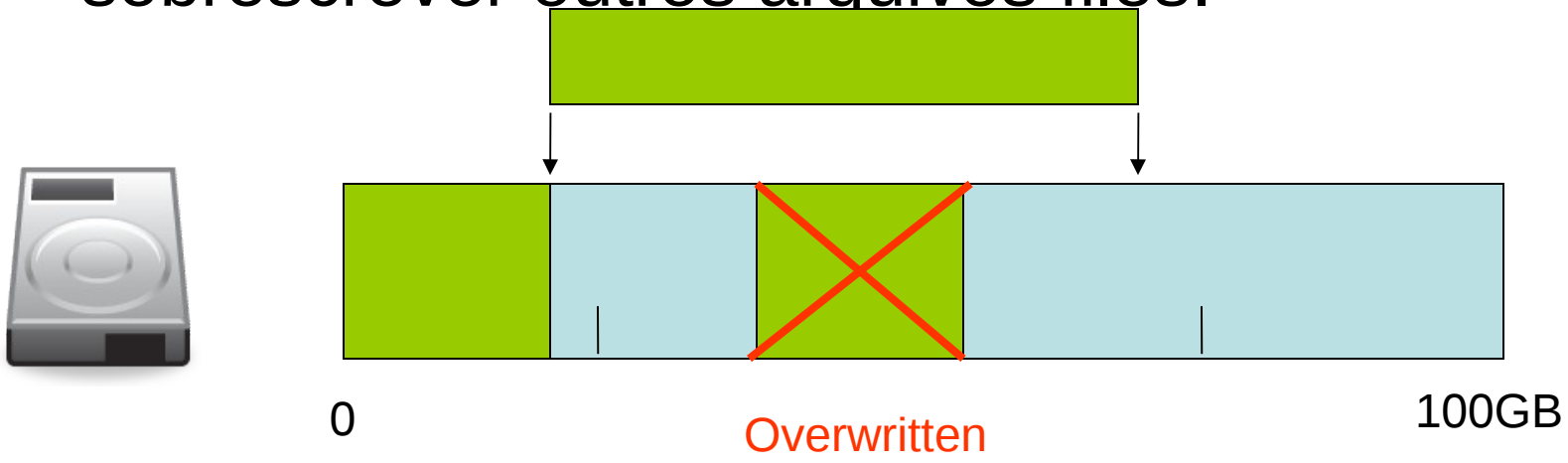
- Fácil!
  - Escrever arquivos sequencialmente.
  - Procura a partir do byte 0.
  - Quando encontrar o próximo espaço vazio, inicia escrita de um novo arquivo.

- Comentário: extremamente RUIM!
  - Extremamente lento na recuperação e escrita de dados.



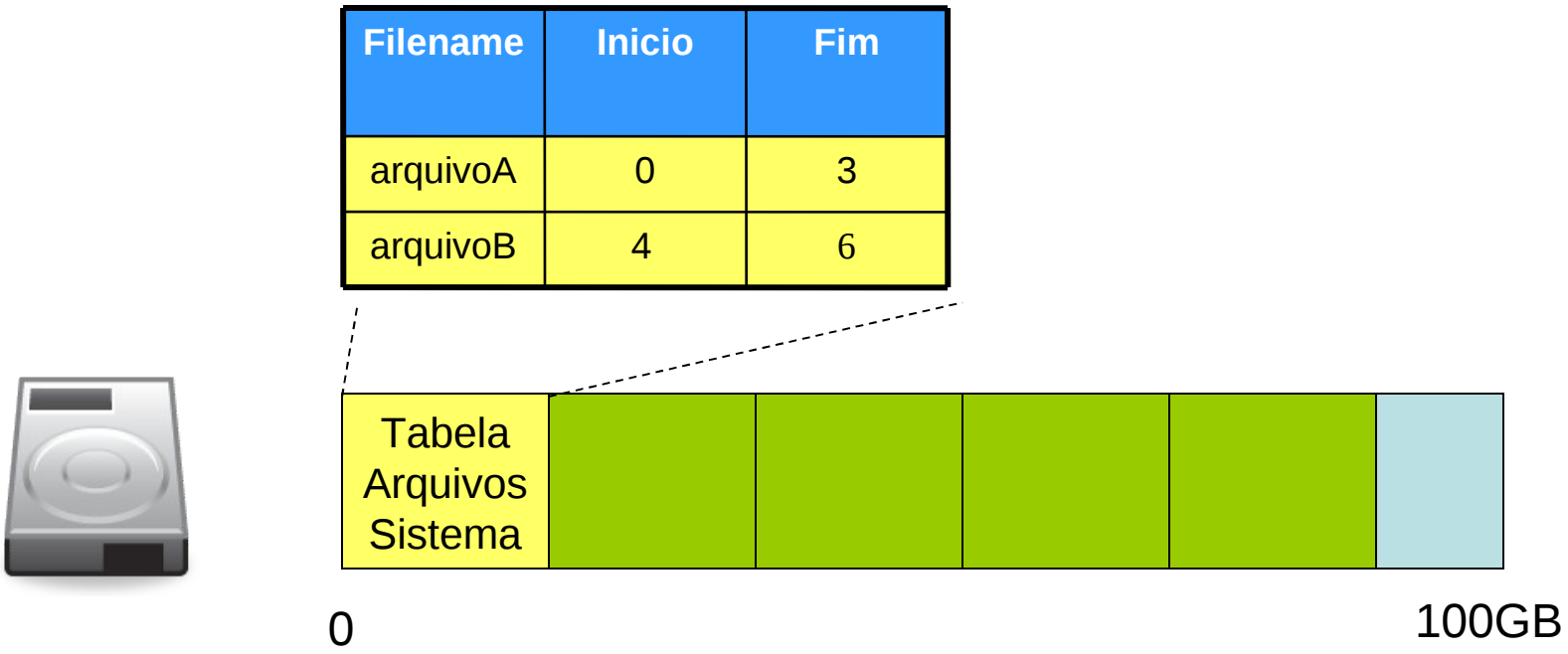
1a. Tentativa

- Outro problema: deleção de arquivo.
  - Quando deletar arquivos, ficam buracos no disco.
  - Escrever um arquivo no buraco pode sobrescrever outros arquivos files.



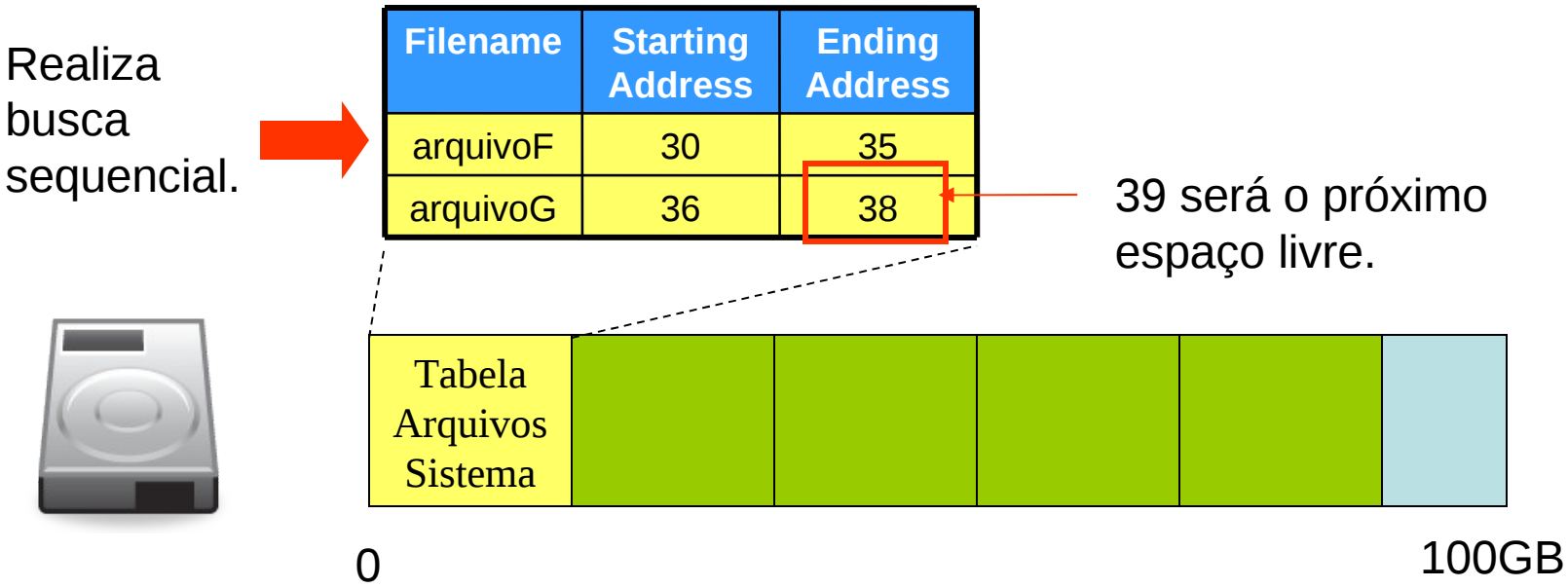
Alocação contígua - TAS

- Muito fácil resolver o problema.
- Manter uma tabela com os endereços de início e fim dos arquivos, chamada Tabela de Arquivos do Sistema.



Alocação contígua - TAS

- É possível procurar arquivos de forma mais rápida.
- Encontrar o próximo espaço vazio mais rápido.

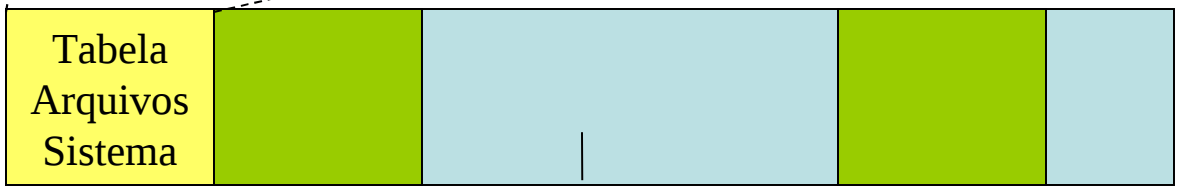


Alocação contígua - TAS

- Não tem problema na deleção.
  - A tabela de arquivos do sistema mostra os buracos no disco.
  - O problema da sobrescrita pode ser evitado.

Filename	inicio	Fim
ArquivoC	7	12
arquivoE	18	29

Um buraco é encontrado de 13 a 17.

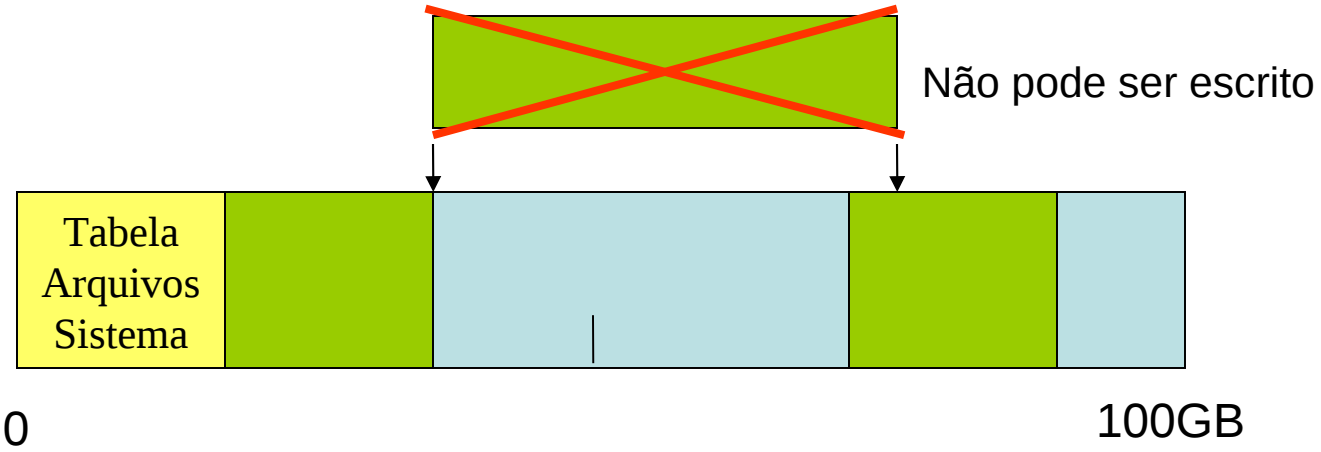


0

100GB

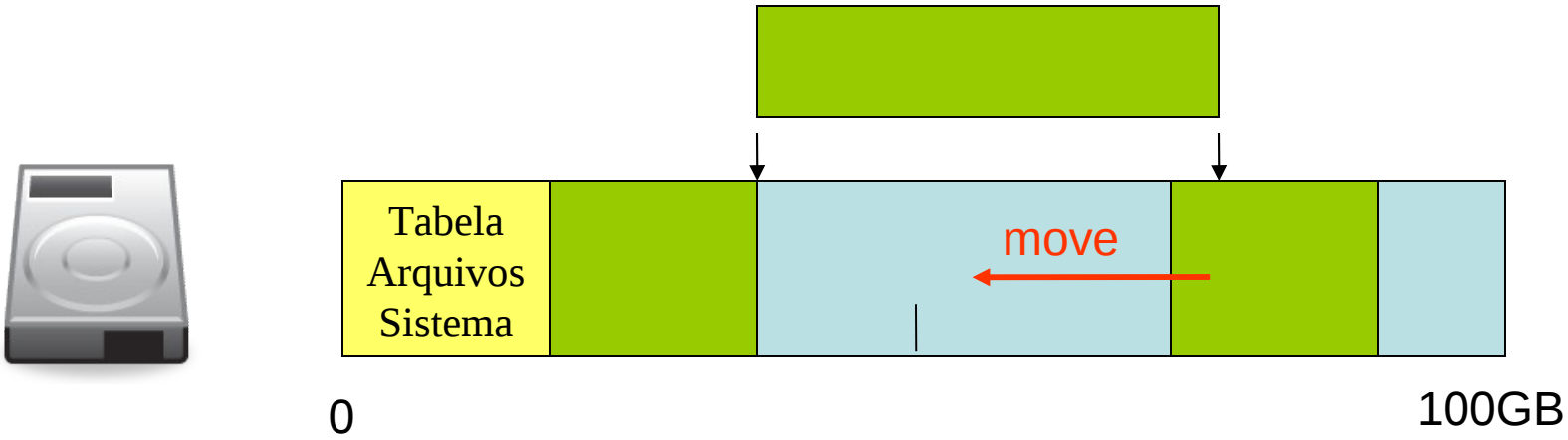
Alocação contígua - TAS

- Não é ruim. Mas, não é bom o suficiente.
  - O espaço livre do disco não é usado da melhor forma.
  - Um grande arquivo pode não encontrar um buraco grande o suficiente, mas existe espaço suficiente na realidade.
  - Este problema é conhecido como fragmentação externa.
    - O espaço alocado esta fragmentado em pedaços, contendo buracos.



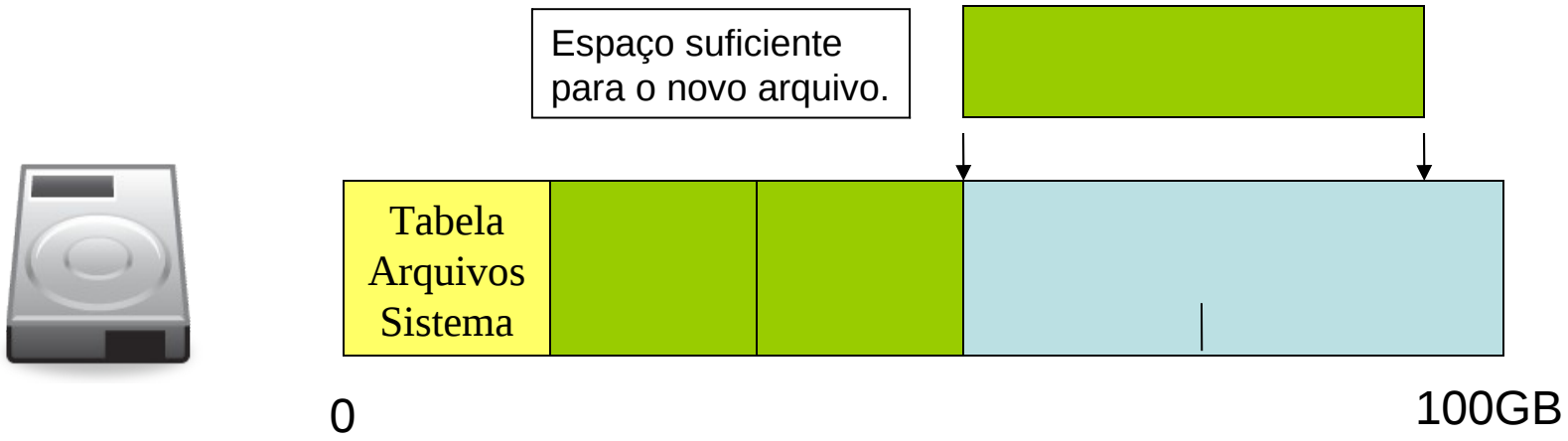
Alocação contígua - TAS

- Que tal se nós compactar o espaço vazio?
  - Alinhar todos arquivos sequencialmente.
  - O espaço alocado fica no início do disco.
  - O espaço vazio fica no final do disco.
  - A tabela de arquivos do sistema deve ser re-escrita.



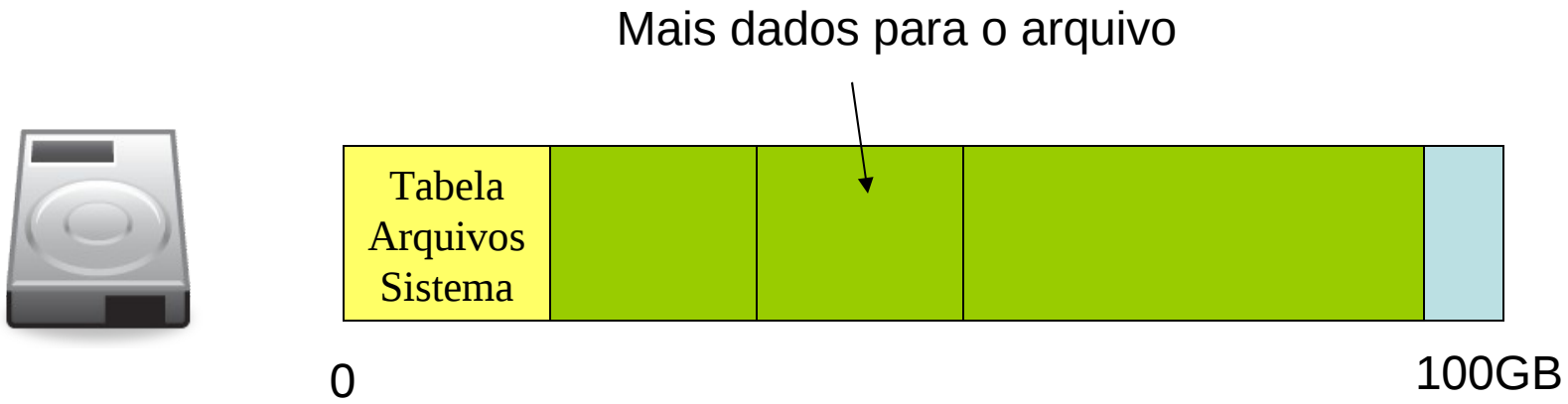
Alocação contígua - TAS

- Esta ação é chamada de desfragmentação.
  - Porém, é muito cara.
    - A ação pode ser lenta, pode involver todos os arquivos no disco.
    - O disco torna-se não disponível durante este procedimento.



Alocação contígua - TAS

- Mais um problema.
  - E se um arquivo deseja crescer mas não tem espaço vazio adjacente a ele?
  - A solução para este problema é, novamente, cara.

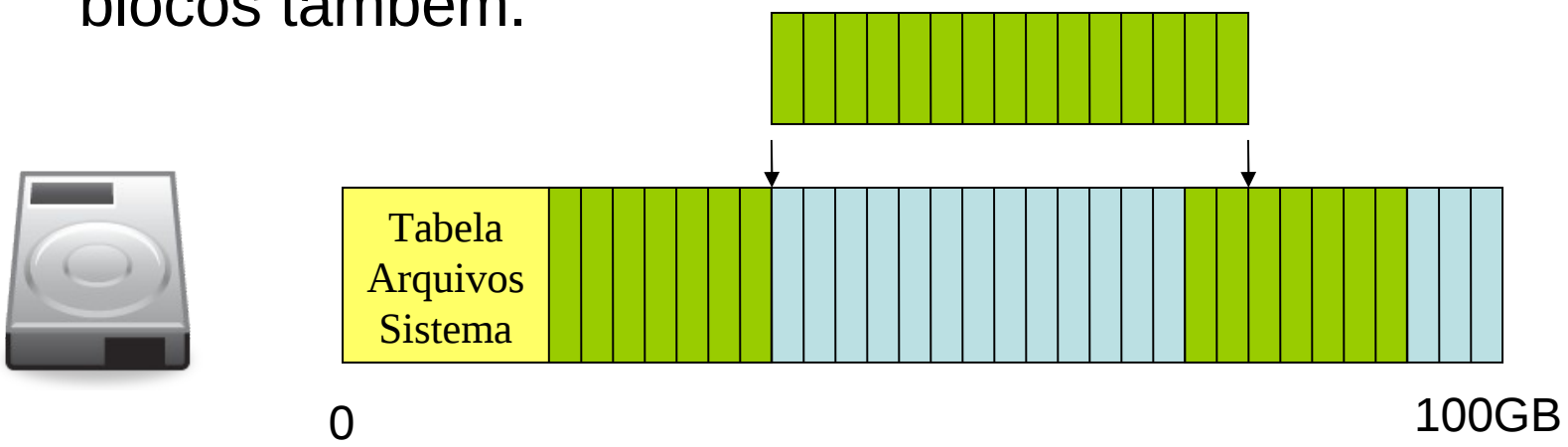


Alocação contígua

- Implementação de alocação contígua.
  - É a melhor escolha qdo
    - Arquivos não são deletados do disco;
    - O tamanho dos arquivos é fixo.
  - Que tal um exemplo real?
    - ISO 9660 - O ISO 9660 é a norma internacional de armazenamento de dados que faz a descrição da estrutura de arquivos e diretórios de um CD-ROM.
    - Juliet – extensão do ISO9660

Blocos iguais

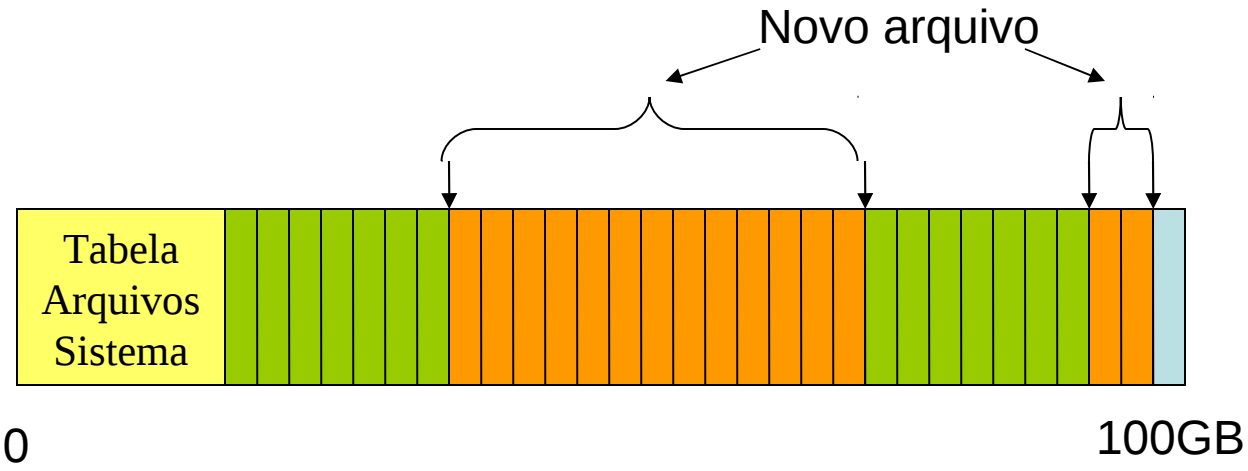
- Como melhorar a performance de alocação contígua?
- Que tal…
  - passo (1). quebrar os espaços alocado e vazio em pequenos blocos iguais.
  - passo (2). quebrar um novo arquivo em pequenos blocos também.



Blocos tamanho fixo

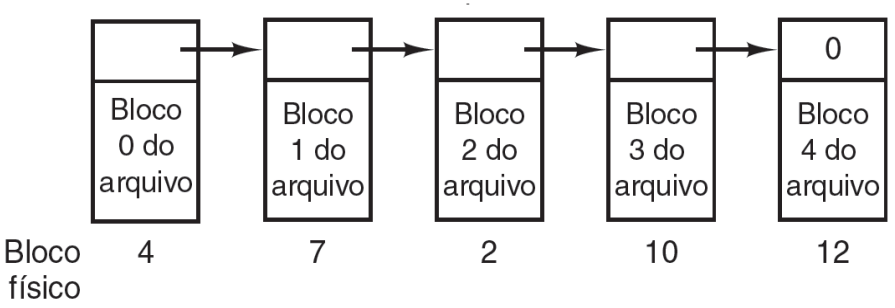
  - passo (3). colocar os novos arquivos no espaço vazio bloco a bloco.

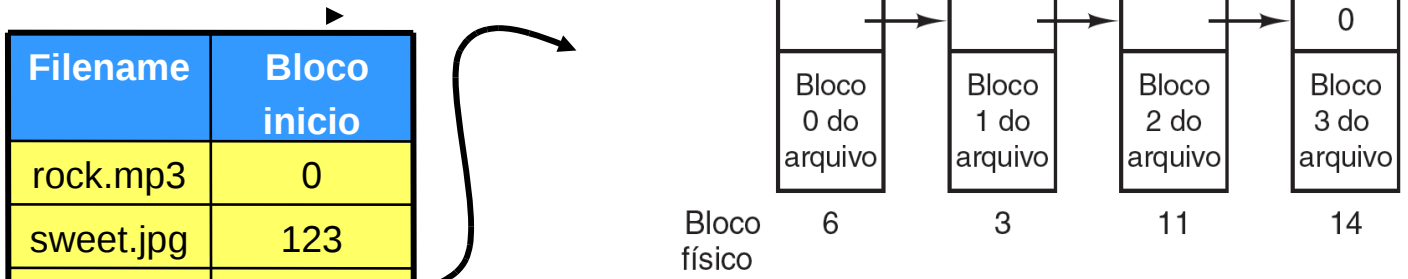
- Assim, o novo arquivo pode ser colocado no espaço vazio disponível efetivamente.
- O que acontece com a tabela de arquivos do sistema?



Ligar blocos - lista encadeada

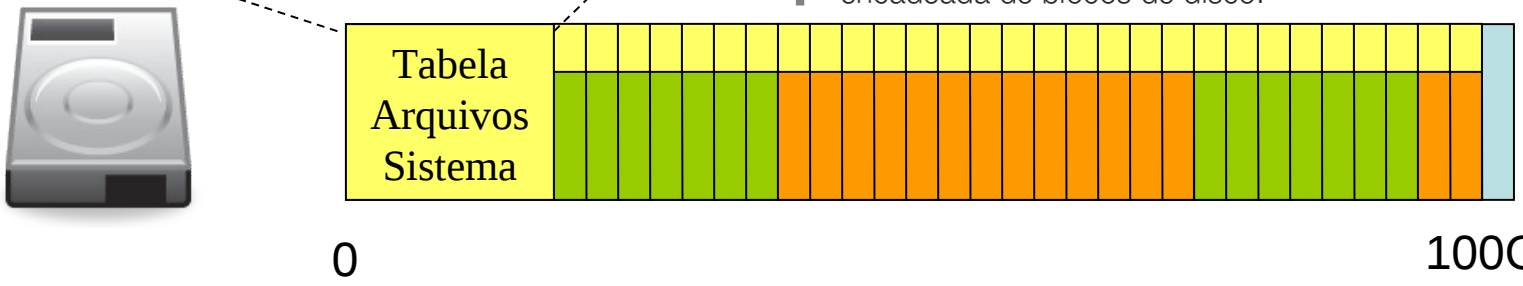
- Que tal uma implementação com lista encadeada?
  - _ Todos arquivos são organizados como uma lista ligada de blocos de dados.





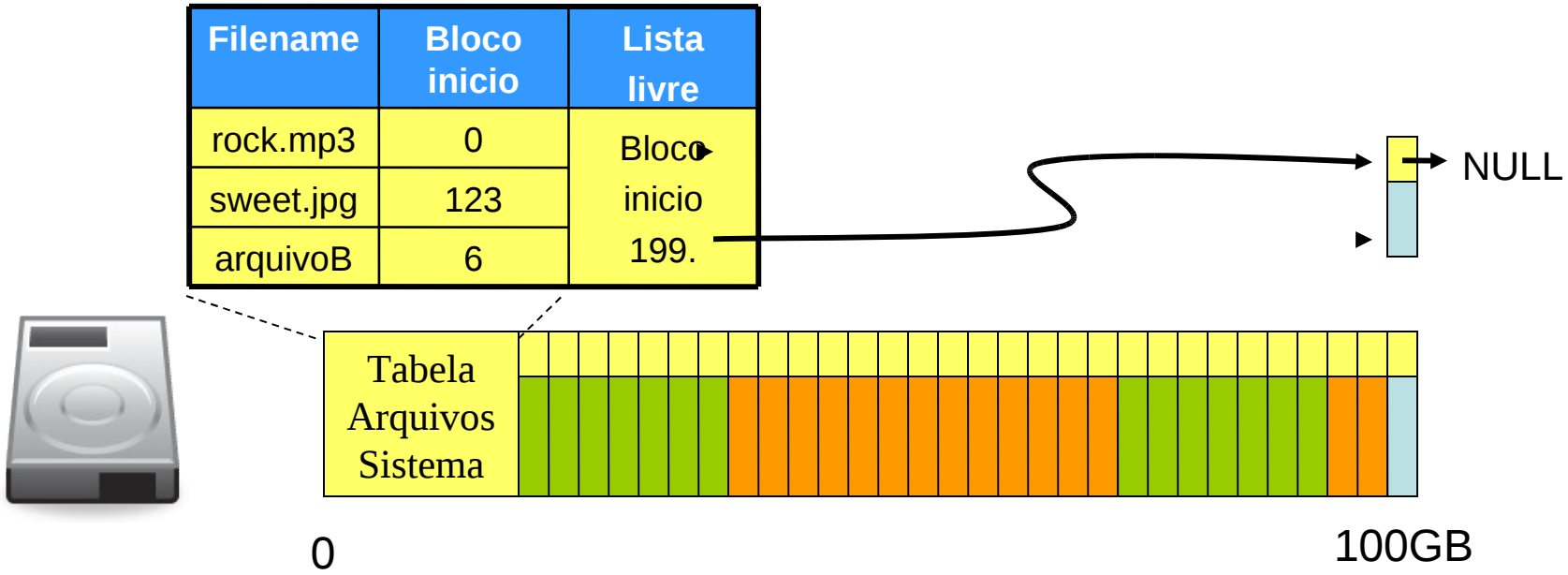
Filename	Bloco inicio
rock.mp3	0
sweet.jpg	123
arquivoB	6

Figura 4.9 Armazenamento de um arquivo como uma lista encadeada de blocos de disco.



Lista encadeada

- Que tal uma implementação com lista encadeada?
  - O espaço livre também é organizado como uma lista ligada.

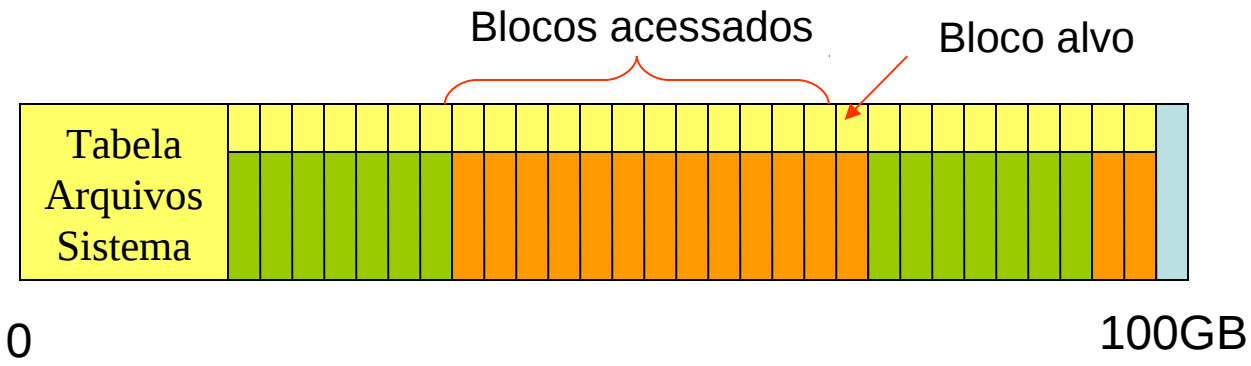


Lista encadeada

- Prós
  - Fácil de implementar.
    - O valor do ponteiro é o número (ou endereço) do próximo bloco.
  - Tabela de arquivos do sistema menor comparado com alocação contígua.
  - Sem fragmentação externa.
  - Arquivos podem crescer ou diminuir.
  - Espaço livre esta bem organizado.

Lista encadeada

- Contras
  - Acesso Randômico é problema.
    - Recuperar um bloco específico é caro.
    - Pior caso: voce quer ler o último bloco do arquivo, para isto, voce tem que percorrer desde o primeiro bloco no disco até chegar no último.
    - Grande número de acessos a disco.



Lista encadeada

- Contras (continua)
  - Fragmentação Interna é problema.
    - Arquivo nem sempre é múltiplo do tamanho do bloco.
    - Último bloco do arquivo pode não ser usado completamente.
    - Este espaço vazio será perdido uma vez que nenhum outro arquivo poderá usa-lo.

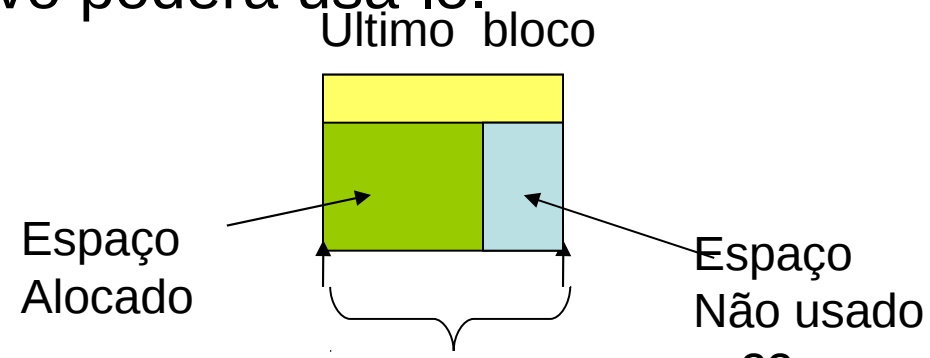


Tabela de procura: FAT

- Que tal implementar uma tabela de procura?

Filename	Bloco inicio
rock.mp3	0
sweet.jpg	123
arquivoB	6

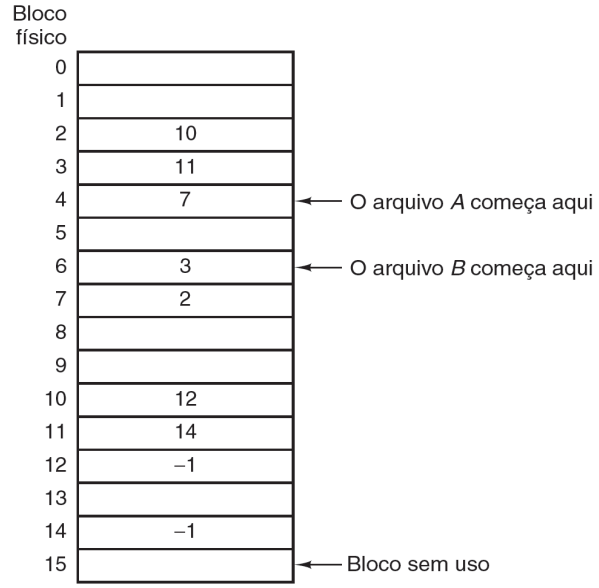
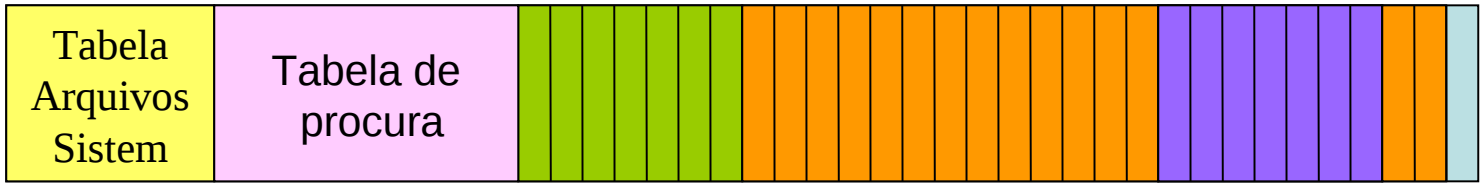


Figura 4.10 Alocação por lista encadeada usando uma tabela de alocação de arquivos na memória principal.



File Allocation Table

- Esta abordagem é chamada de file allocation table (FAT) .
- Principios de funcionamento:

  - Agrupar todos os dados da lista ligada em uma tabela.

  - As entradas na tabela são armazenadas contiguamente.

  - Cada entrada corresponde a um bloco de alocação do arquivo no dispositivo de armazenamento.

  - O tamanho da FAT depende da capacidade do dispositivo e do tamanho do bloco.

    - Ex., 100GB hard disk com bloco de 1-KByte.
      - 100 milhoes de entradas na tabela do sistema de arquivos.
      - Se uma entrada da FAT tem 4 bytes, então 400MB de memoria são necessários.

FAT

- Pros.
  - A tabela pode ser carregada na memória para uma procura rápida de um bloco.
  - Não tem problema no acesso randômico considerando que a procura pelo bloco é feita na memória.

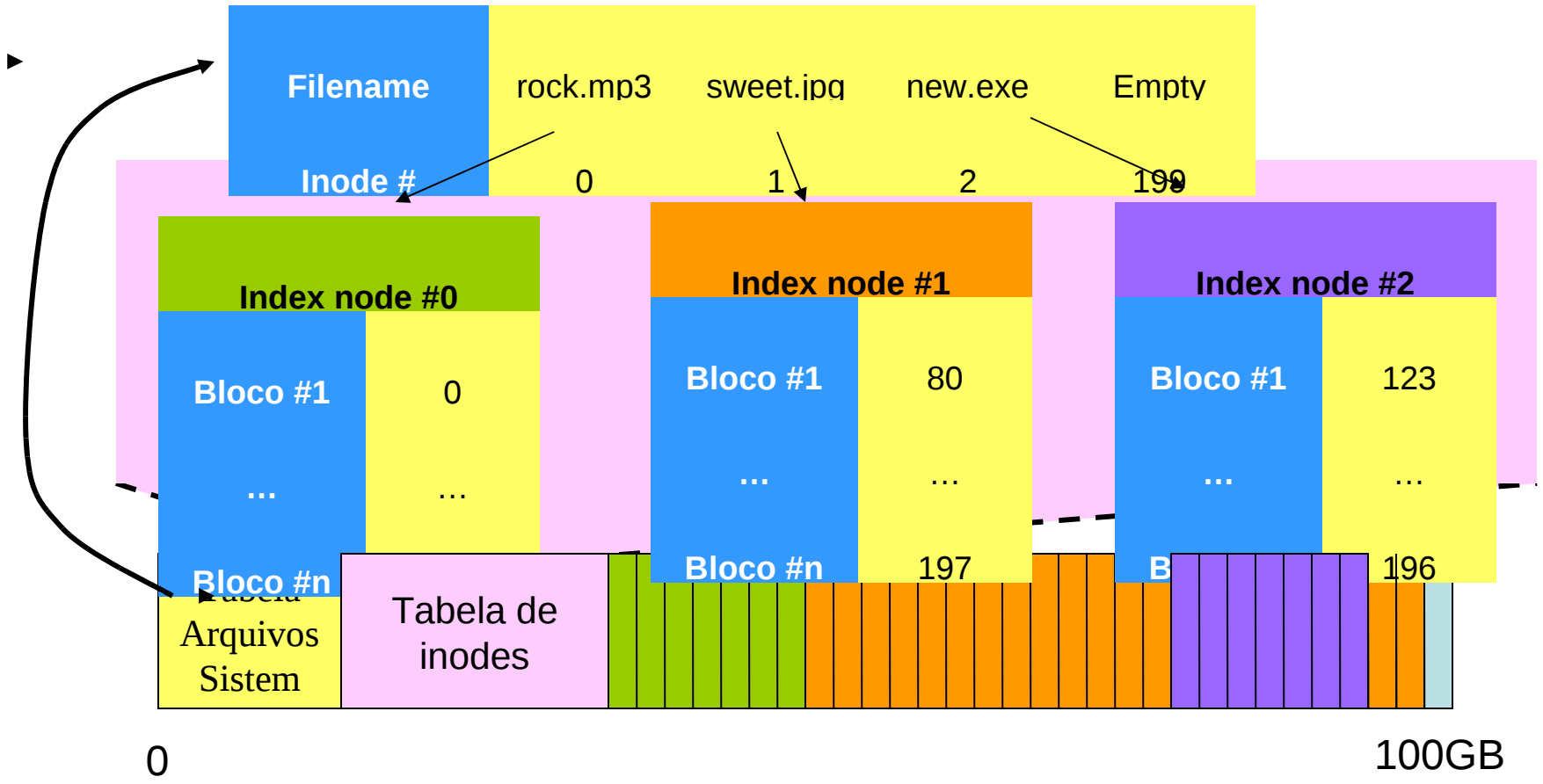
- Cons
  - Uma grande tabela é estabelecida.
  - Fragmentação Interna.

Alternativa

- O SA tem de carregar toda a tabela na memória para saber a localização dos blocos de dados alocados.
  - A própria tabela pode ser composta por milhares de blocos.

- Funcionaria se particionar a tabela em pequenos pedaços?
  - Depende de como particionar…
  - A meta é ler apenas alguns poucos blocos, de forma que o SA saiba a localização dos blocos de dados alocados.

Index-node - I-node



Index-node

- Esta abordagem é chamada de

  index-node (i-node).

- Principios de funcionamento:

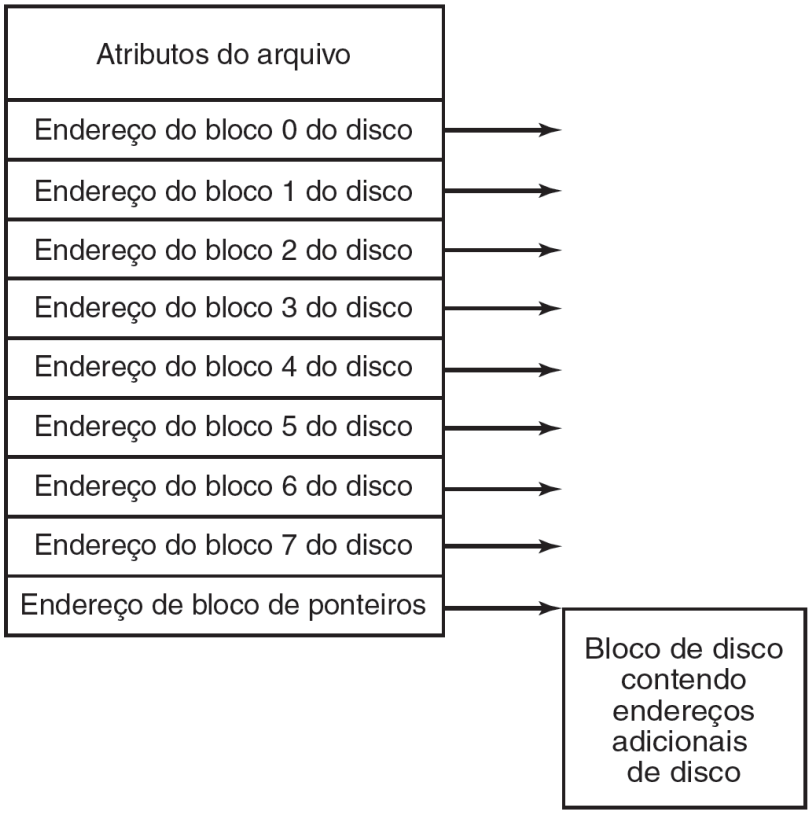
  - A tabela de SA mantem o mapeamento do nome do arquivo para o inode.

  - Um inode armazena os endereços de todos os blocos de dados.

  - Todos os inodes são armazenados na tabela de inodes.

  - O tamanho da tabela de inodes determina o número de arquivos que o SA pode armazenar.

I-nodes



- Blocos de indices encadeados
- Blocos de indices multinível

Figura 4.11 Um exemplo de i-node.

Index-node

- Prós

  - Uma operação de leitura na tabela do SA e uma operação de leitura na tabela de inodes localizam todos os blocos de dados de um arquivo.

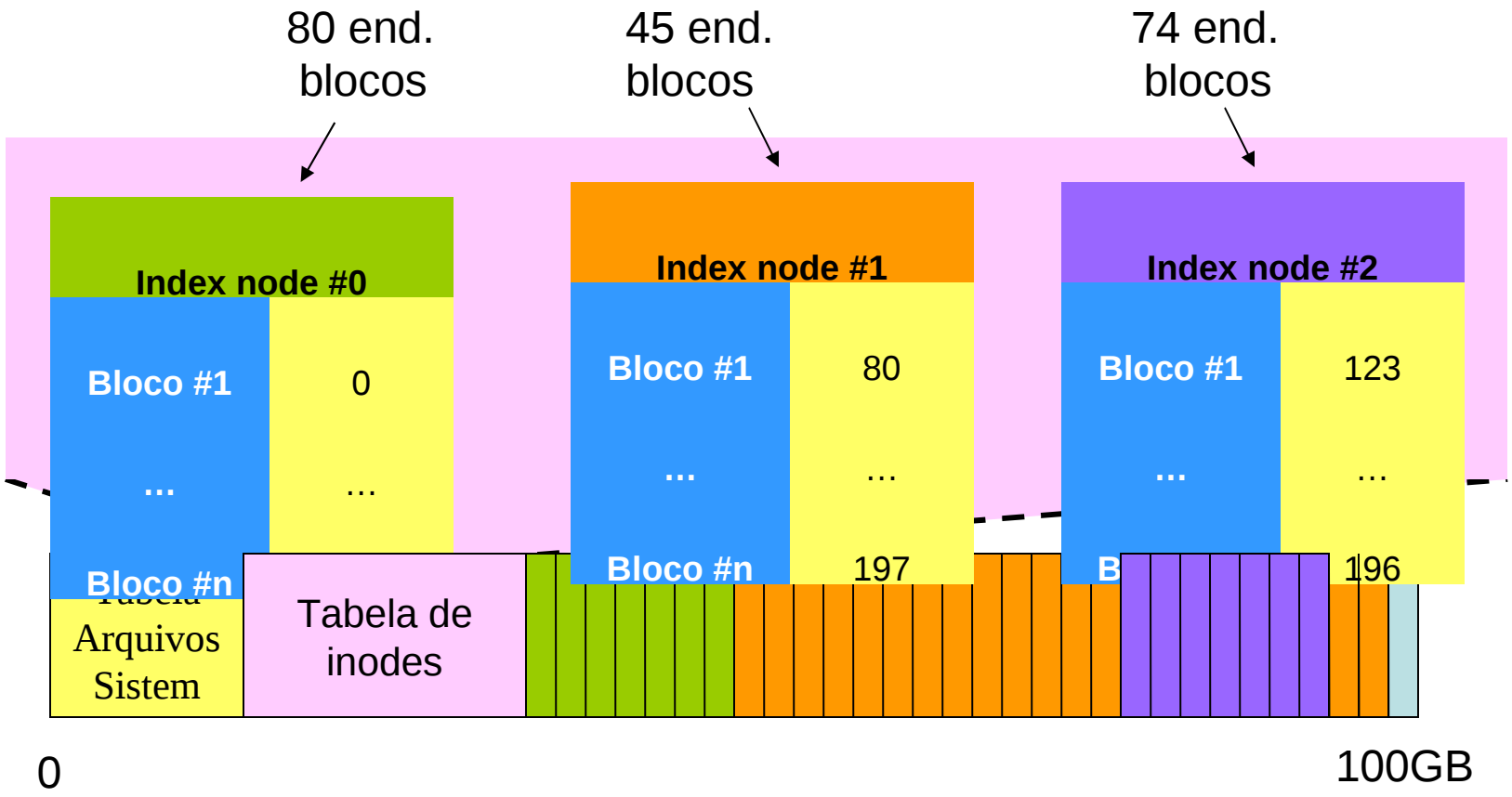
  - Necessita menos memória que a FAT.

- Contras

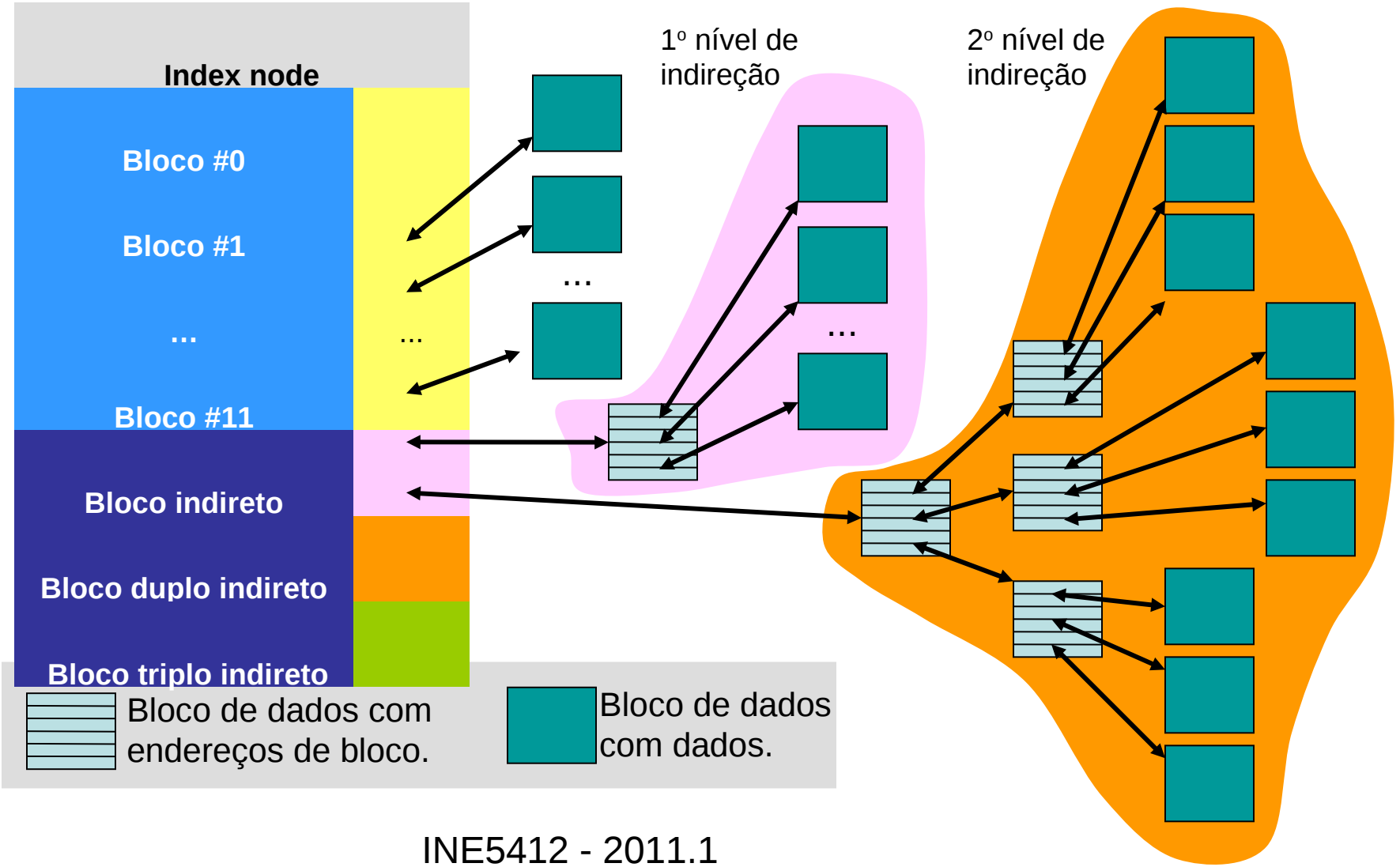
  - O tamanho do inode tem que ser variável.
    - O tamanho da lista de endereços de blocos é variável.
    - Torna difícil para o SA ler a tabela inode eficientemente.

Index-node

- Numero variável de blocos de endereço.



Alternativa – blocos indiretos



Alternativa #4.1

- Principios de funcionamento:

  - Cada inode tem um tamanho fixo.

  - Os endereços dos blocos são primeiro colocados nos endereços de blocos diretos do inode.

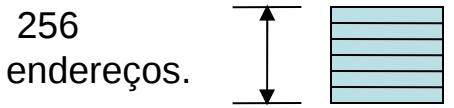
  - Depois dos endereços diretos serem usados, os endereços indiretos de bloco serão usados.

Alternativa #4.1

- Endereço Indireto de bloco do inode:

  - Um endereço indireto de bloco aponta para um bloco de endereços. Este bloco contém uma série de endereços de blocos de dados.

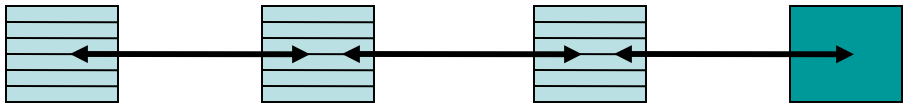
  - Ex., um endereço de bloco tem 4 bytes e um bloco de dados tem 1 Kbytes. Assim, teremos 256 endereços de blocos de dados.



Bloco 1-Kbyte

Alternativa #4.1

- Múltiplos níveis de blocos indiretos:
  - O conceito de blocos indiretos ampliado.
  - Ex., tres níveis de blocos indiretos.



- Prós
  - Faz com que o inode tenha tamanho fixo.
  - Aumentando de forma significativa o tamanho de arquivo suportado.

Alternativa #4.1

- Exemplo :

Bloco Direto x 12;

Bloco Indireto x 1;

Bloco Duplamente Indireto x 1;

Bloco Triplamente Indireto x 1;

Tamanho de Bloco = 2^x bytes.

Tamanho do endereço = 2^2 bytes.

Tamanho máximo : $12 \times 2^x + 2^{2x-2} + 2^{3x-4} + 2^{4x-6}$

Ex. Tamanho do Bloco = 1024 bytes, tamanho arquivo = 16 GBytes.

Implementação de SA

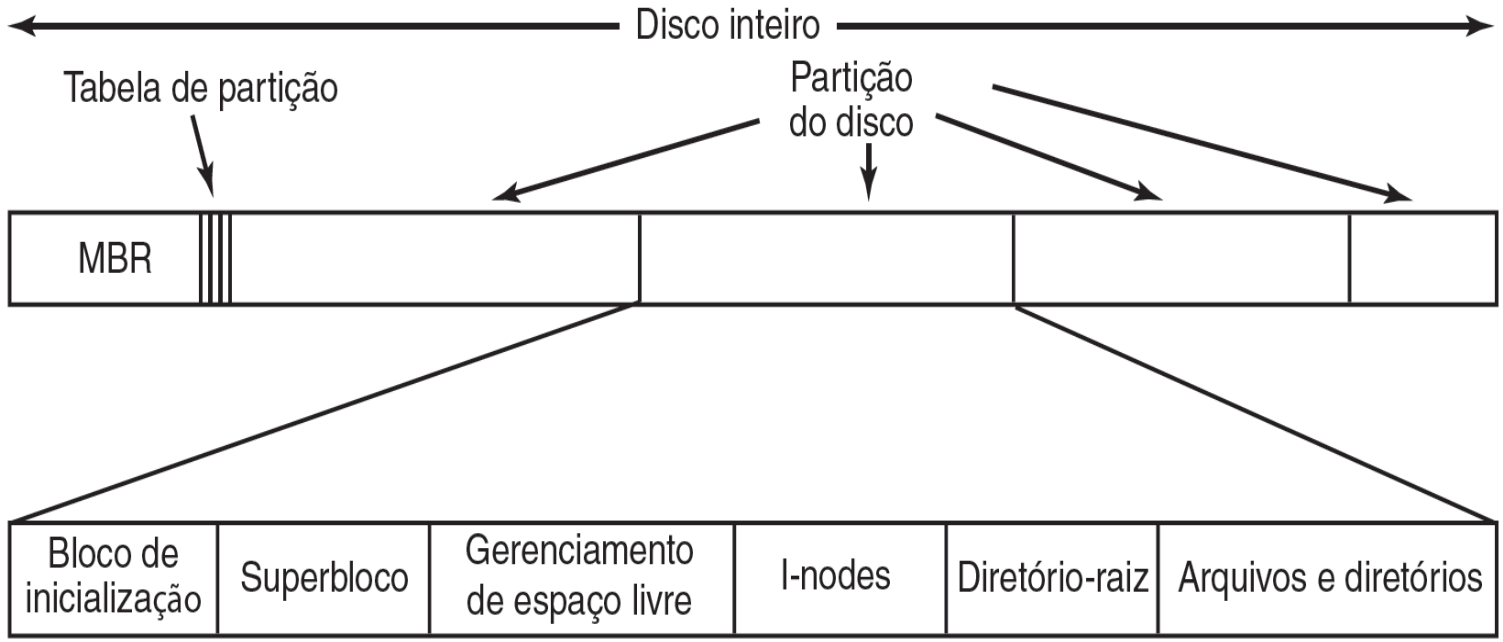


Figura 4.7 Uma organização possível para um sistema de arquivos.

Um possível layout de SA

Implementando Diretórios (1)

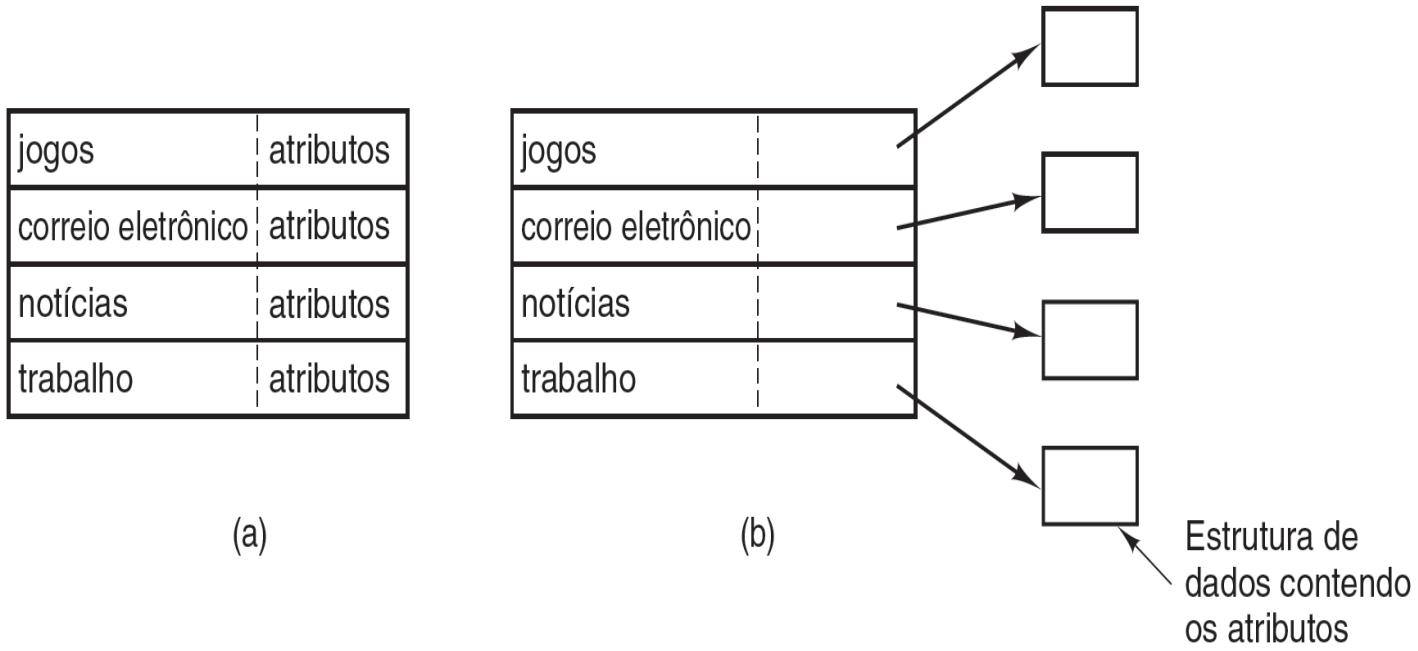
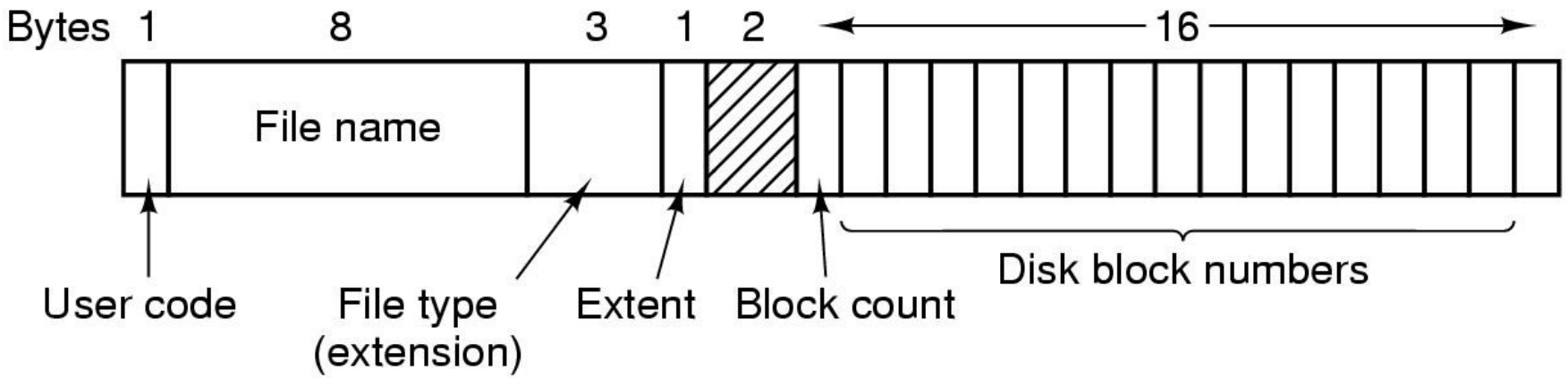


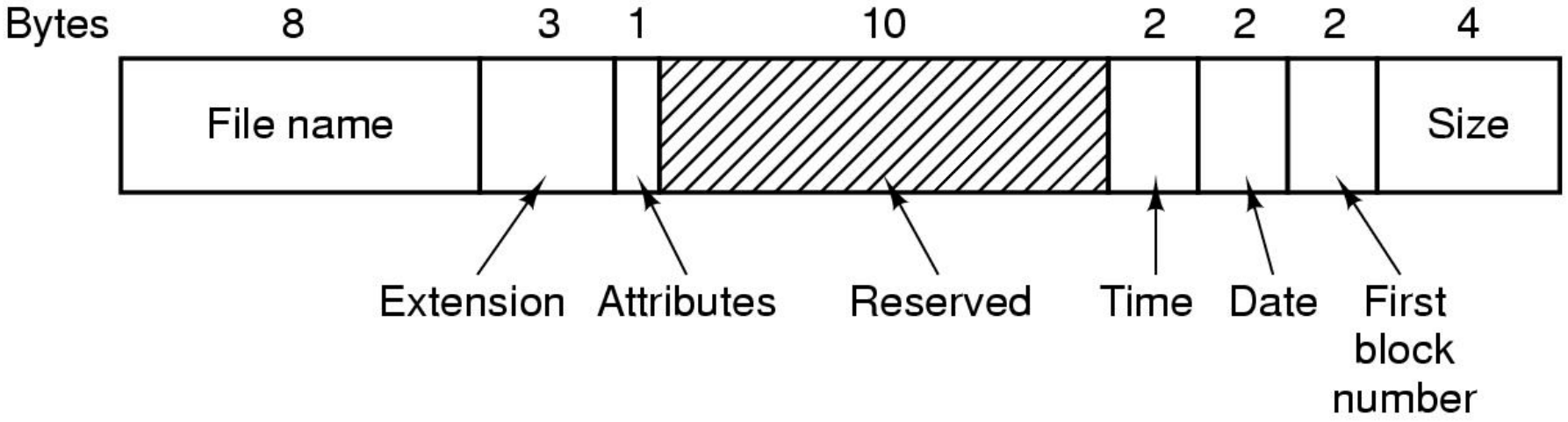
Figura 4.12 (a) Um diretório simples com entradas de tamanho fixo com os endereços de disco e atributos na entrada de diretório. (b) Um diretório no qual cada entrada se refere apenas a um i-node.

Alternativa (a) : Ex.1 - CP/M



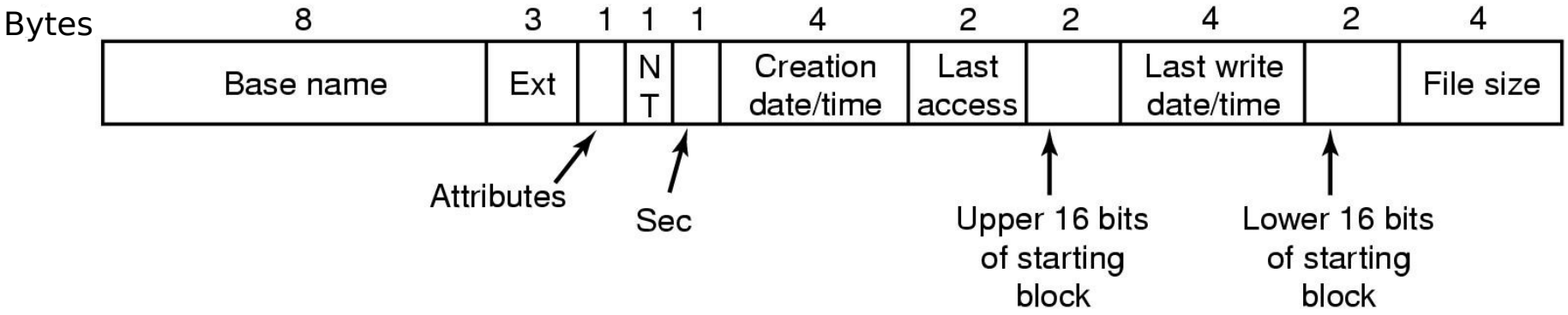
Formato de uma entrada de diretório CP/M

Alternativa (a): Ex. 2- MS-DOS



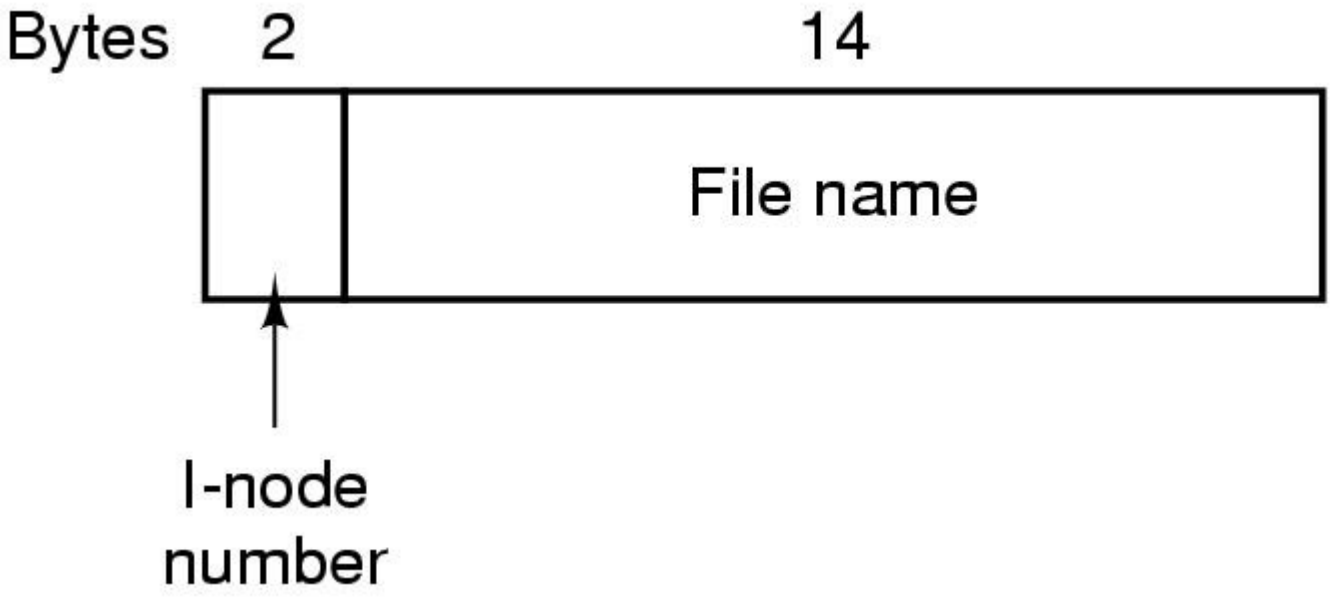
Entrada de diretório MS-DOS

Alternativa (a):Ex. 3- Windows 98



Entrada de diretório (extensão MS-DOS) usada no Win 98

Alternativa (b): Ex. 1: UNIX V7



Uma entrada de diretório UNIX V7

Implementando Diretórios (2)

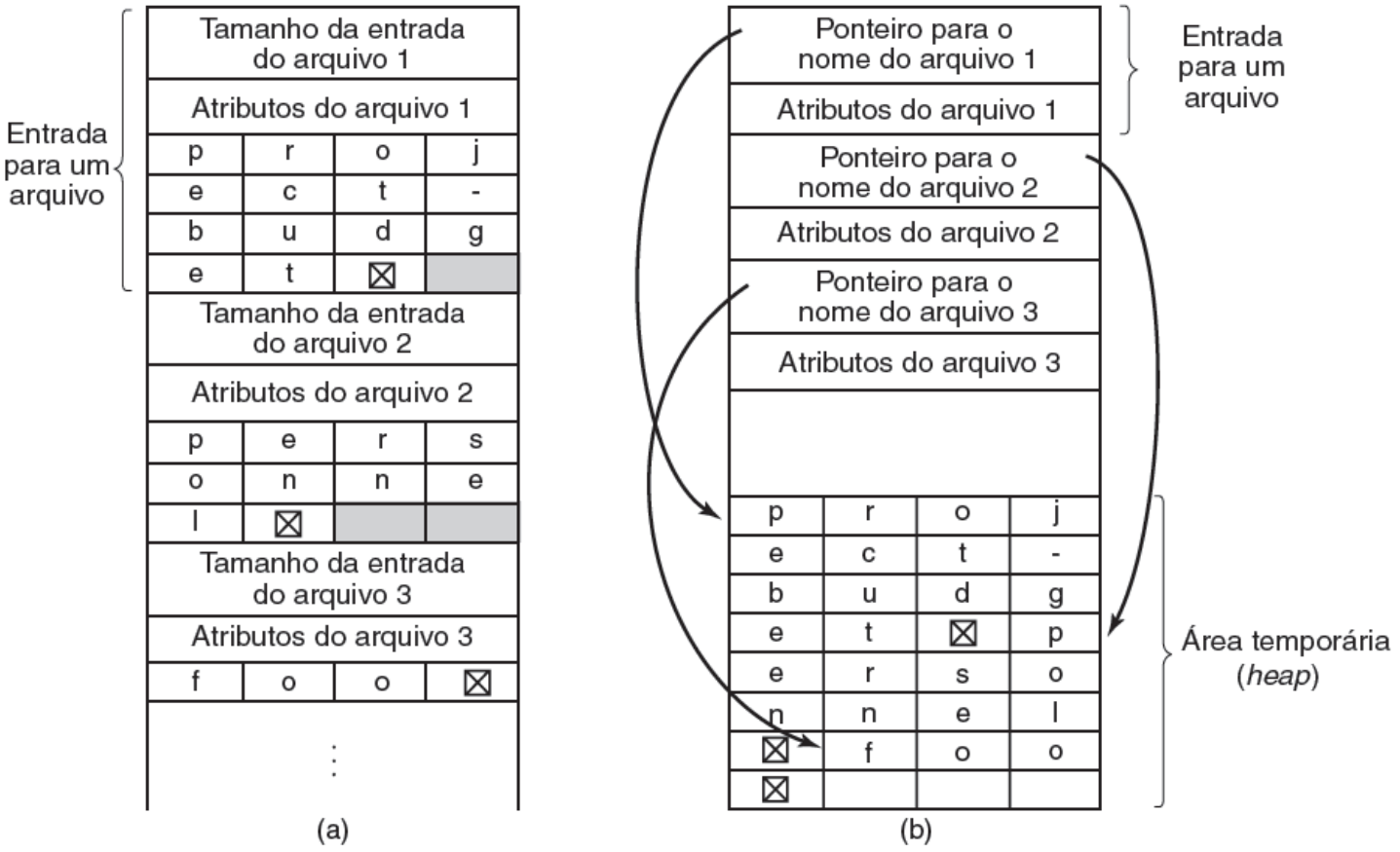


Figura 4.13 Duas maneiras de gerenciar nomes de arquivos longos em um diretório. (a) Sequencialmente. (b) Em uma área temporária.

Arquivos Compartilhados

Compartilhar arquivos é conveniente mas gera problemas, ex. Diretórios contêm endereços de blocos?

- Solução 1: blocos não relacionados no diretório (Unix, i-node)
  - Remoção, contabilidade,
- Solução 2: ligar através de um arquivo (*symbolic link)*
  - *Remoção, problema?*

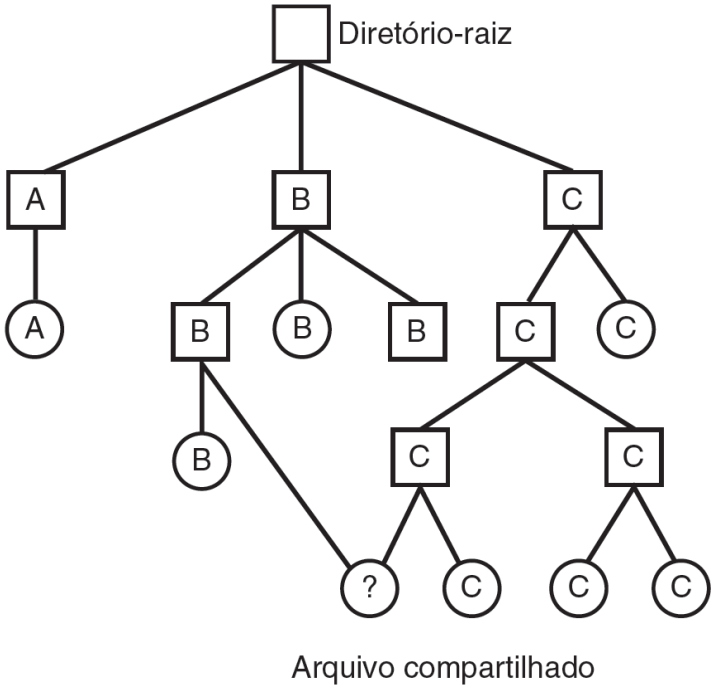


Figura 4.14 Sistema de arquivos contendo um arquivo compartilhado.

Exemplo: fig. 6.3

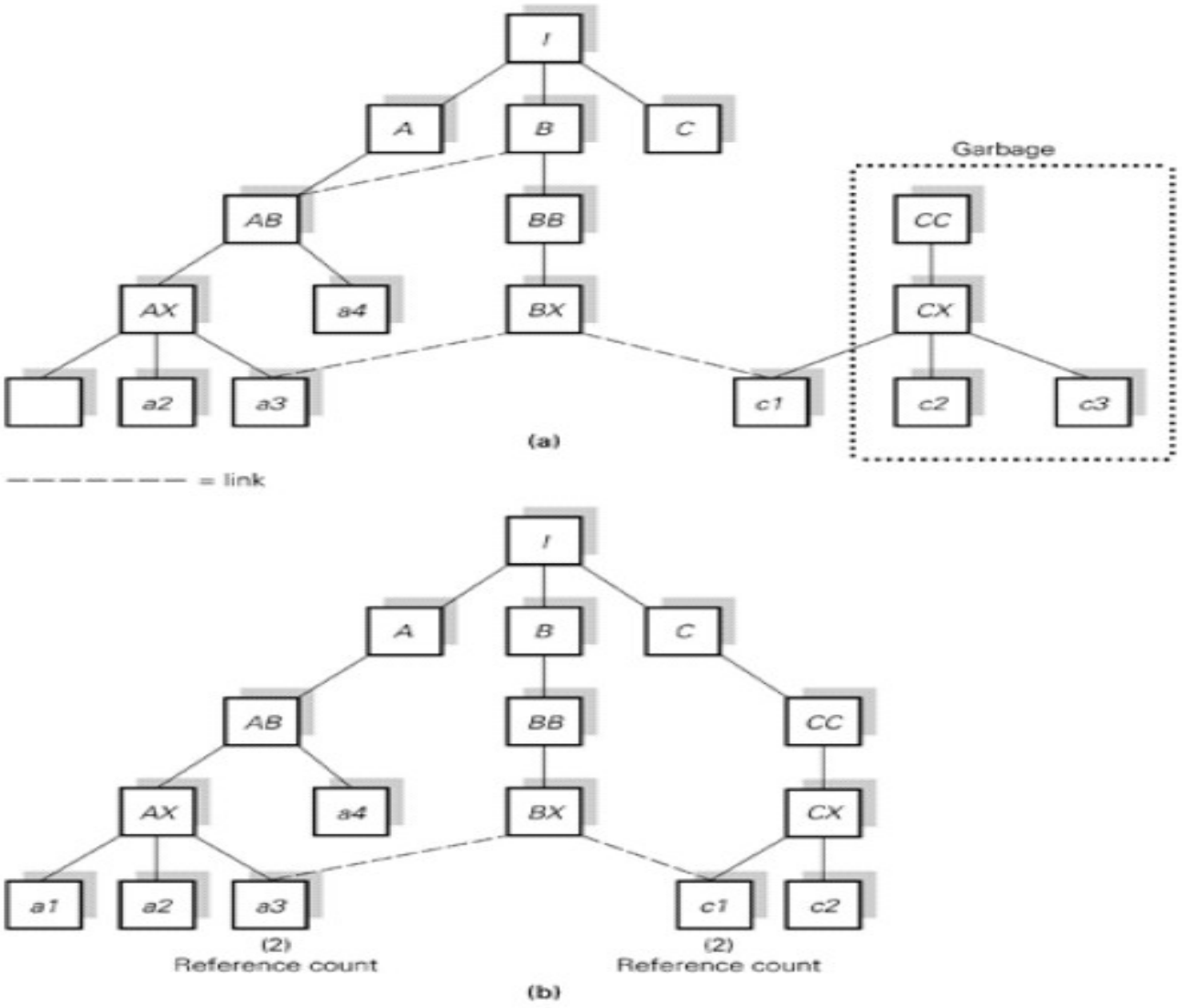


Figure 6.9. Example of a hard link.

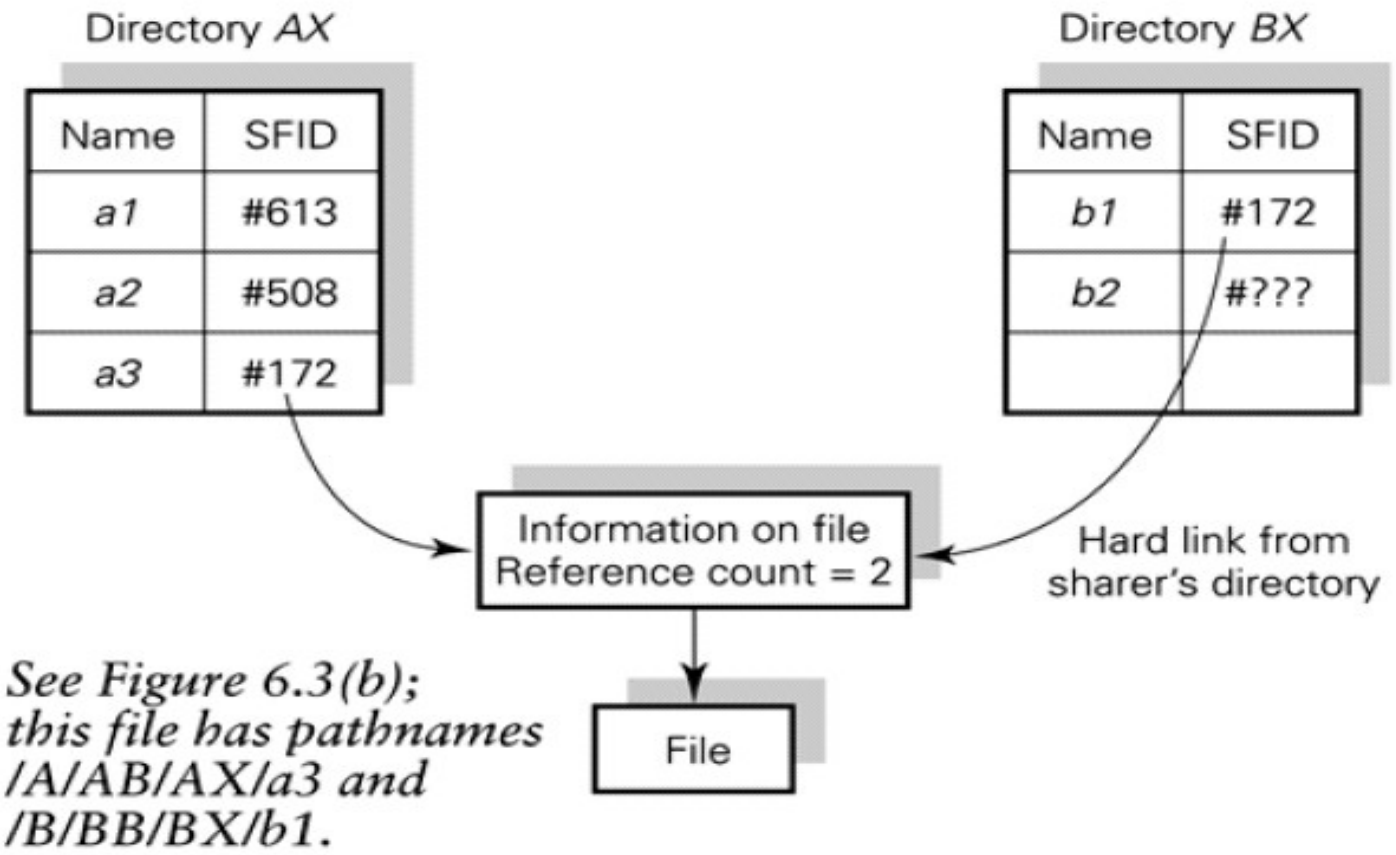
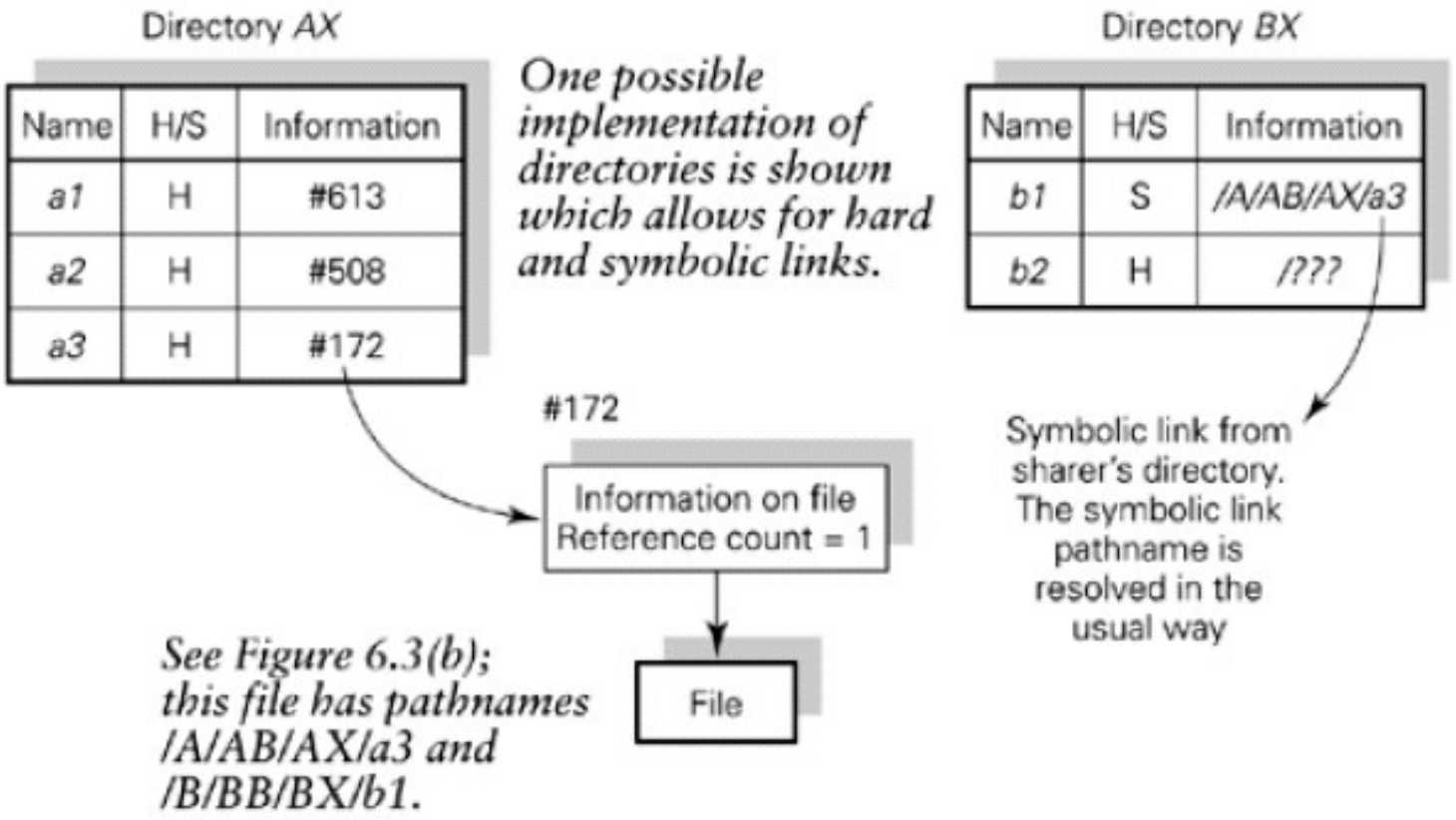
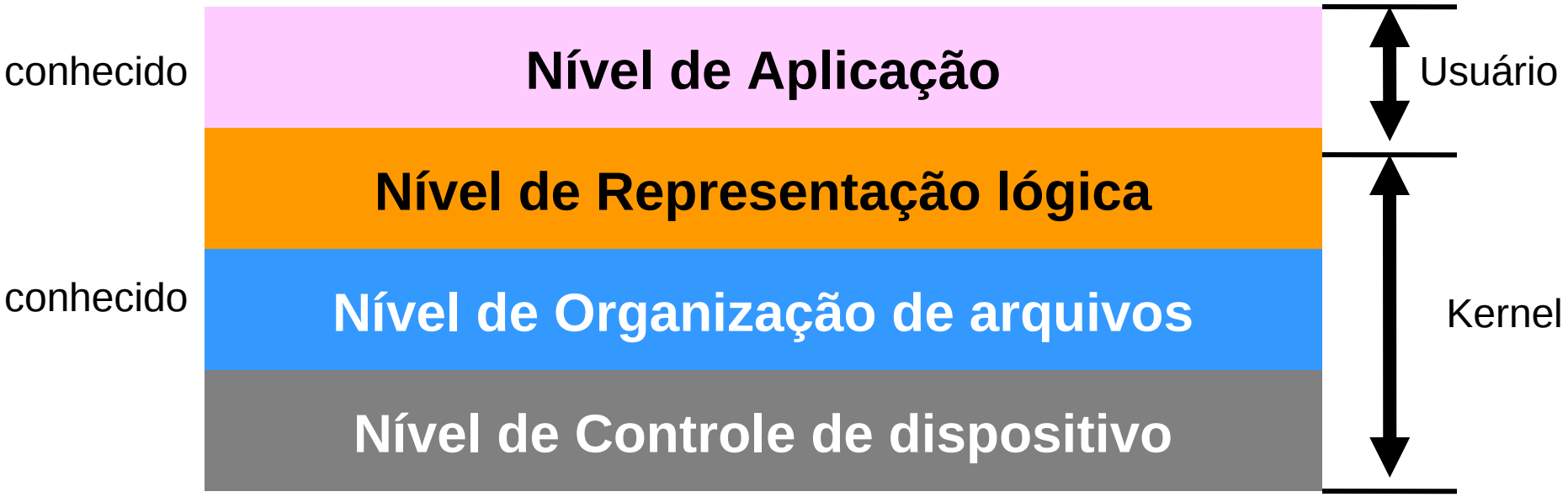


Figure 6.10. Example of a symbolic link.



Estrutura do SA

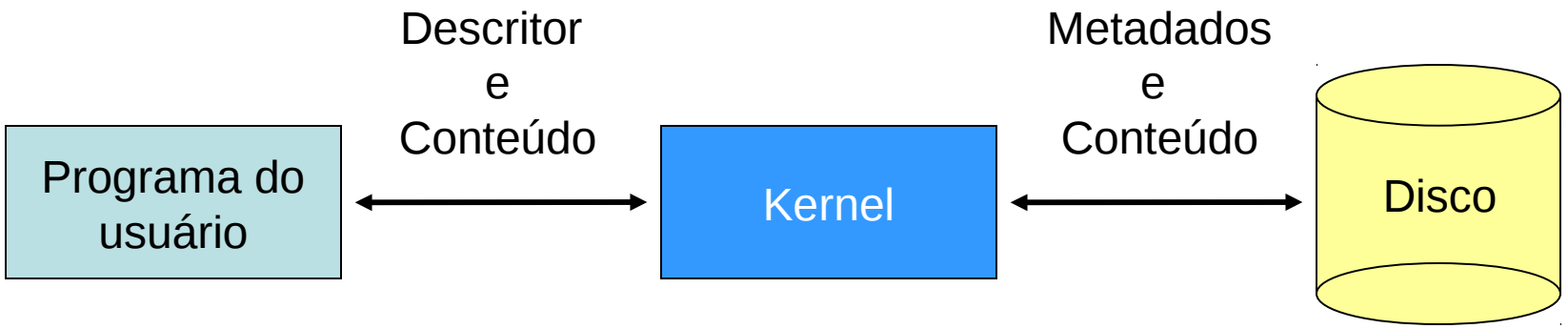


Nível de Representação lógica

Diz respeito a como o kernel modela um arquivo de forma lógica.

Como o kernel interage com o disco?

Como o kernel interage com o usuário?



Relacionamento Kernel-Processo

Existe um relacionamento forte entre um processo e os arquivos abertos.

O programa de usuário recebe um descritor de arquivo como retorno da chamada de sistema open, ou seja, uma representação abstrata de um arquivo aberto.

É tão **abstrato** que é só um numero!

Na realidade, o que significa este número?

```
int fd;
fd = open("/dev/zero", O_RDONLY);
```

file descriptor, no caso de retorno com sucesso.

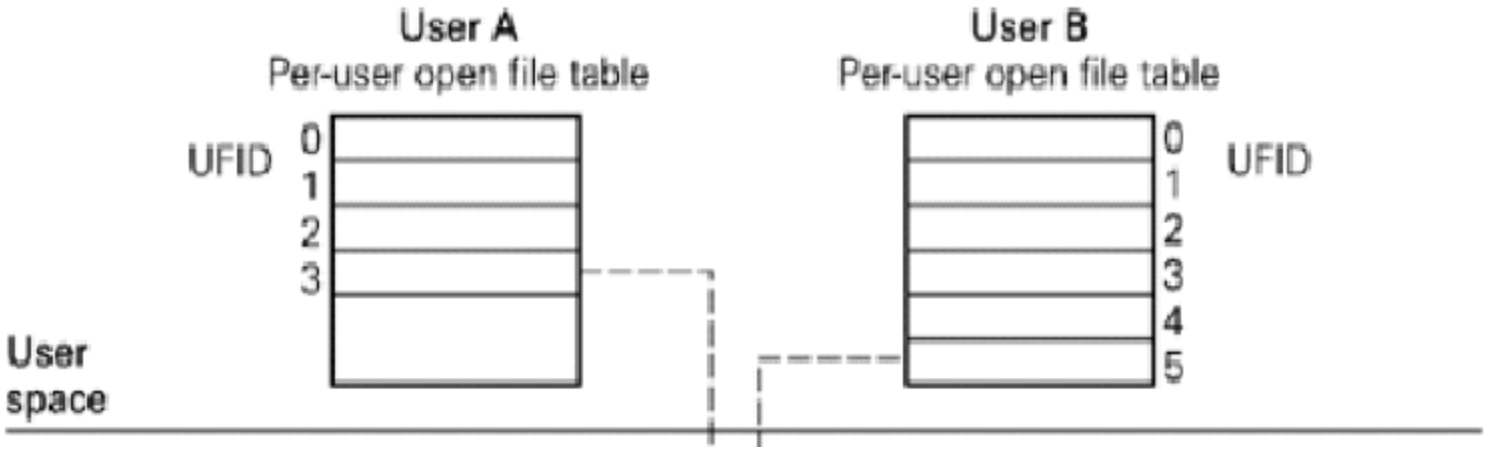
Relacionamento Kernel-Processo

O que acontece quando um processo abre um arquivo?

O retorno da chamada é o descritor do arquivo, que é um indice para uma Tabela de Arquivos Abertos por Processo – cada entrada corresponde a um arquivo aberto pelo processo

Todo processo inicialmente tem 3 arquivos abertos por default.

0 - stdin stream; 1 - stdout stream; 2 - stderr stream.



Relacionamento Kernel-Processo

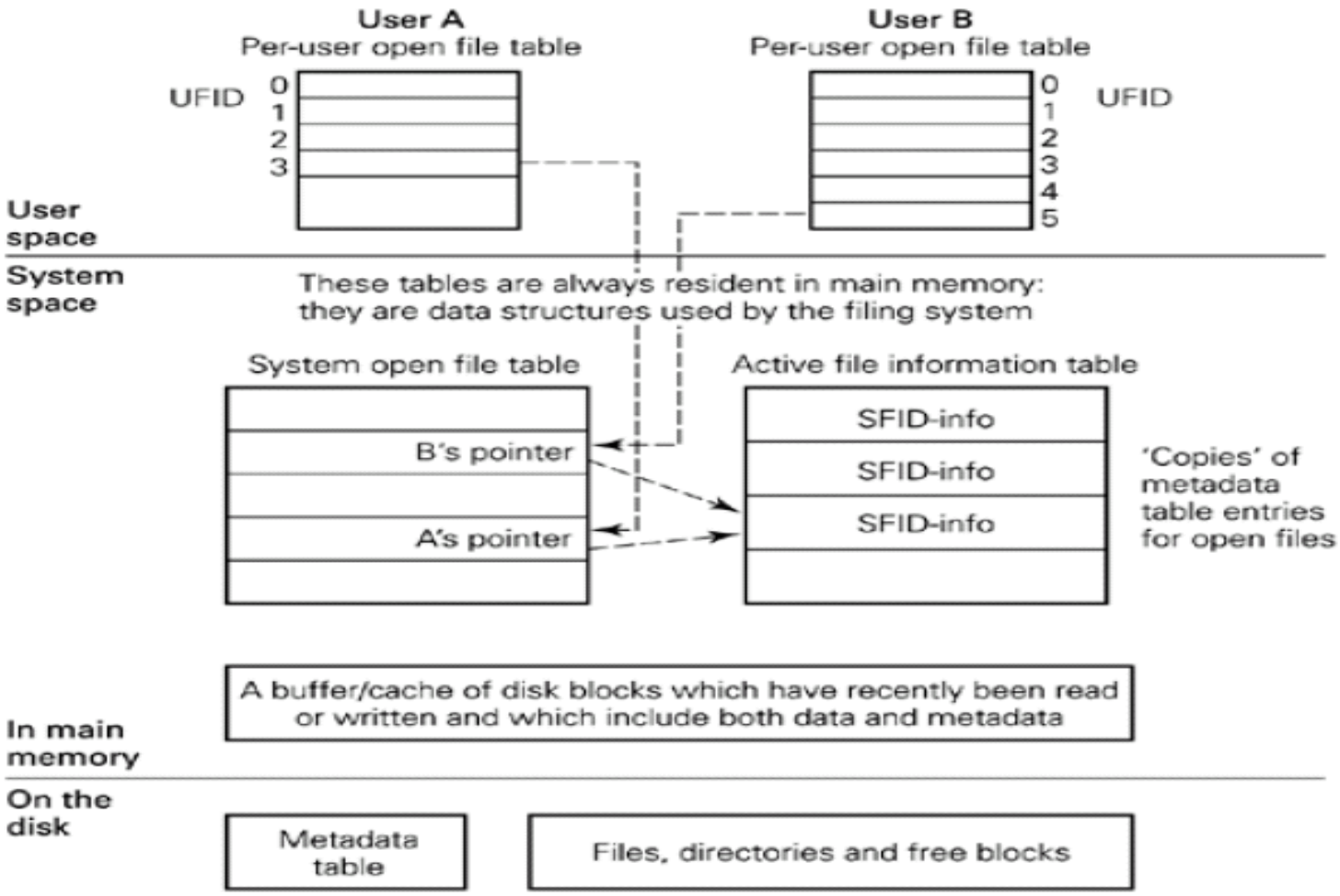
O que acontece quando um processo abre um arquivo? O descritor serve para que os arquivos possam ser acessados (read/write).

Enquanto um arquivo esta sendo acessado, o seu descritor é constantemente necessário – operações de leitura e escrita. Por isto, é necessário alguma forma de acesso rápido ao BCA.

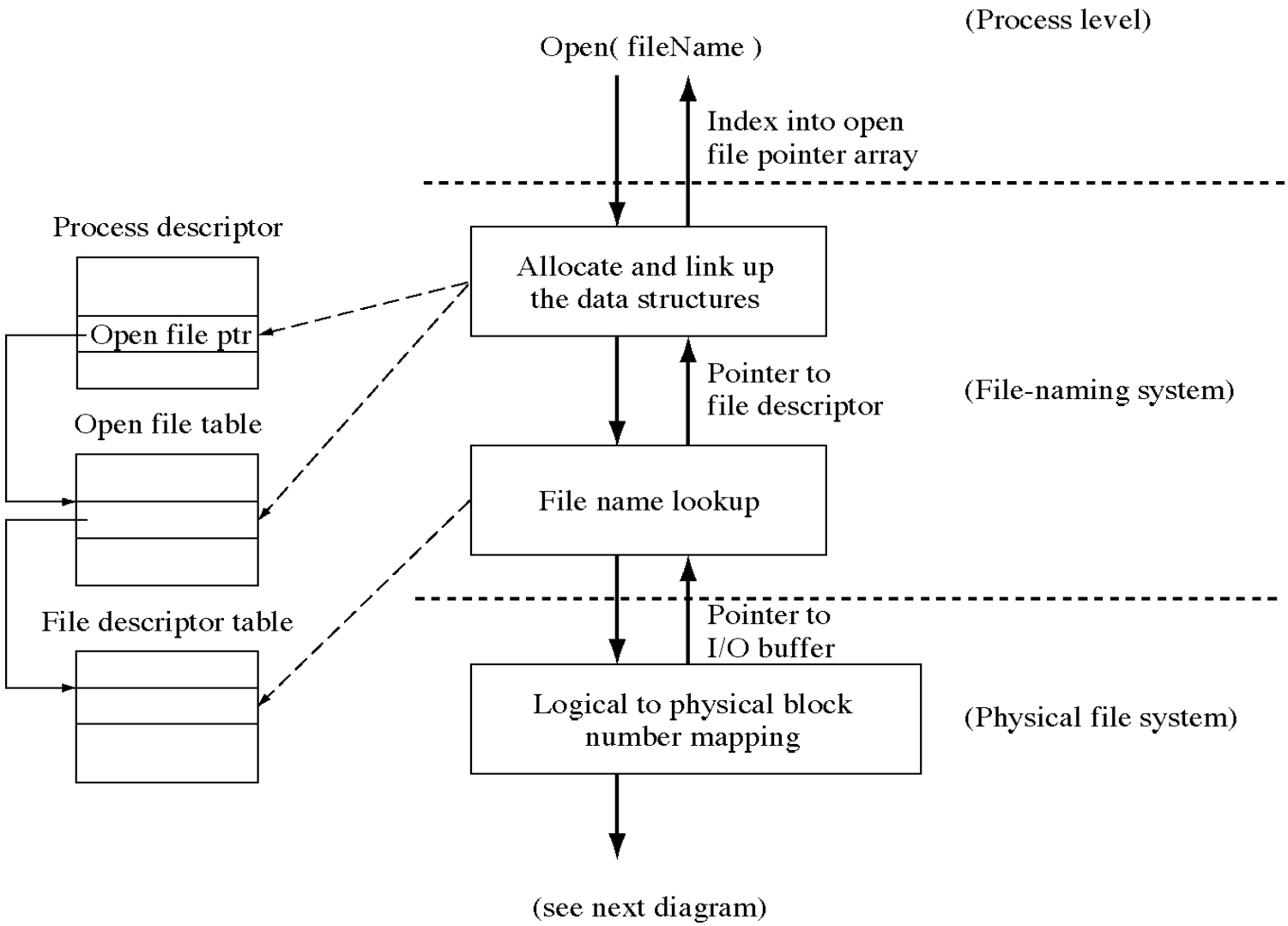
Para isto, o SA deve manter em memória descritores dos arquivos em uso, permitindo acesso rápido. Entrar/sair em uso normalmente é feito através de *open* e *close*

- Tabela de Arquivos Abertos do Sistema – mantém informações relativas aos arquivos abertos no sistema. Cada entrada = descritor + informações adicionais (processos)
- Tabela de Arquivos Abertos por Processo – cada entrada corresponde a um arquivo aberto pelo processo. Apontador para tabela do sistema
- Tabela de Descritores de arquivos - Inodes

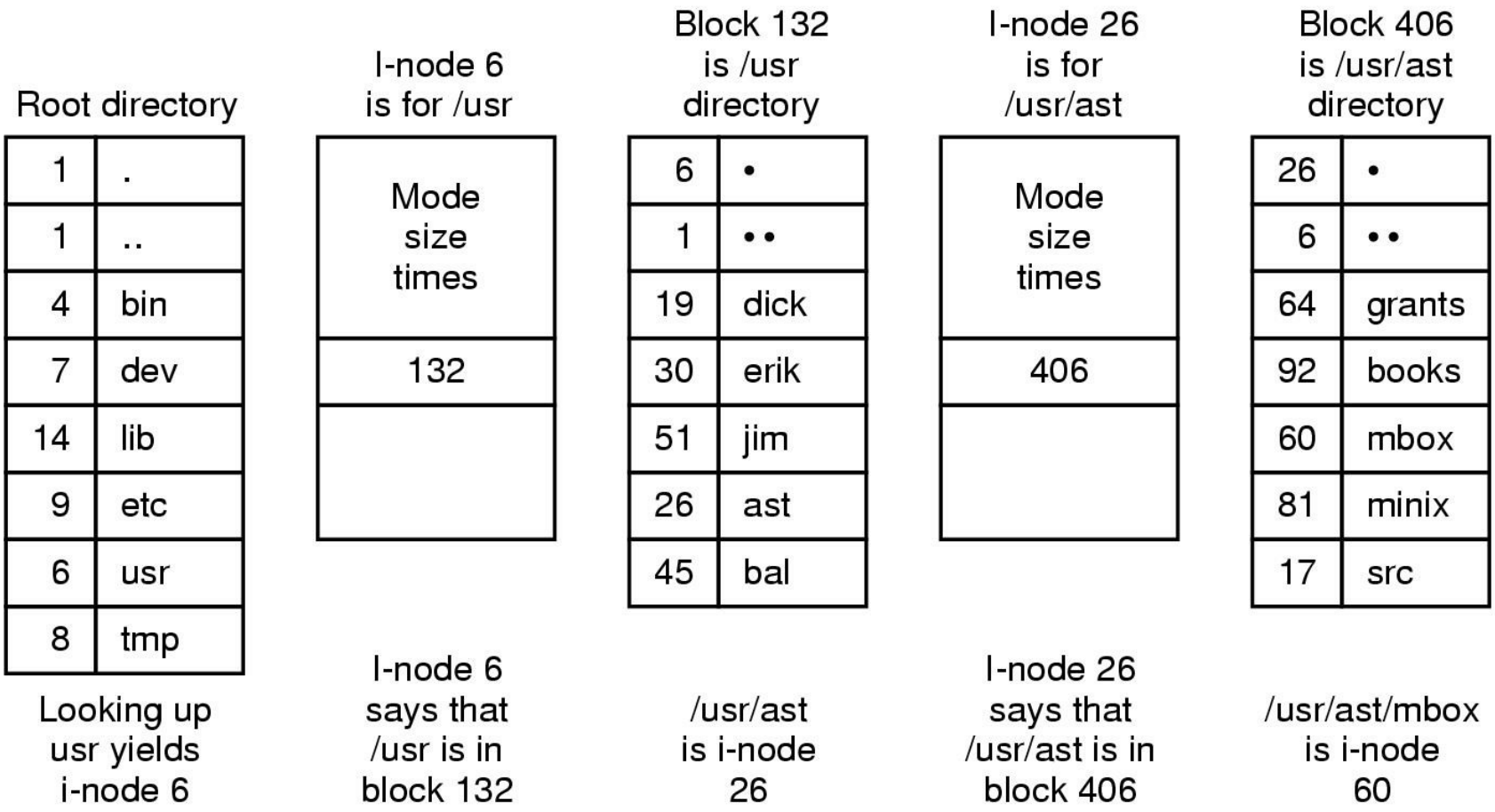
Relacionamento Kernel-Processo



Fluxo de controle para open()

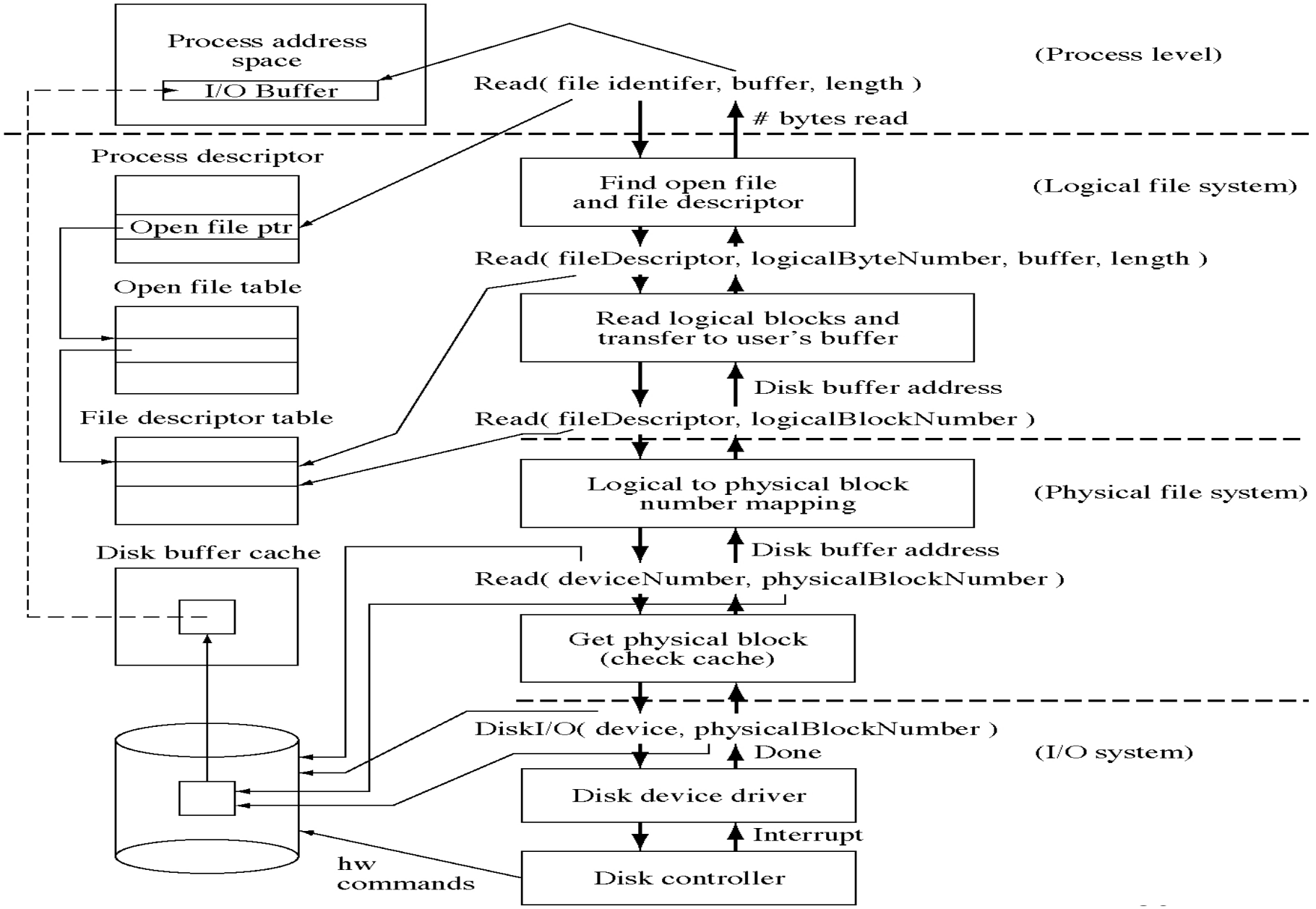


Ex. : UNIX V7

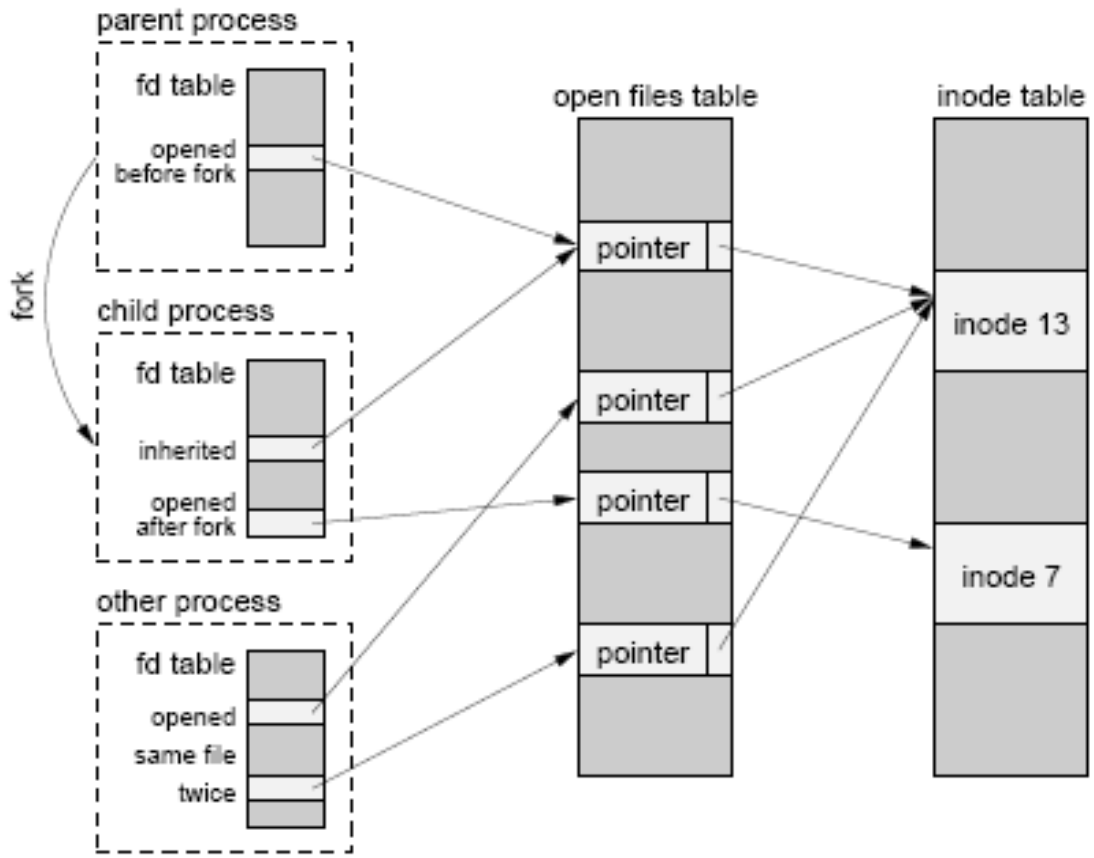


Passos na busca do arquivo */usr/ast/mbox*

Fluxo de controle para read



Fork : mudança nas estruturas



Sistema de arquivos virtual(1)

Muitos SA diferentes, na mesma máq. e SO.

Windows – NTFS, FAT, DVD, etc.(C:, D: etc)

- Sem unificação

Unix – ext2, ext3, RaiserFS, ISO9660

- Montagem em apenas uma hierarquia, totalmente visível à implementação
- Sun Microsystems (1986) VFS

Ideia principal: abstrair a parte comum aos diferentes sistemas e colocar o código em uma camada separada que chama o SA para o gerenciamento do dado

Sistema de arquivos virtual(2)

Visão geral

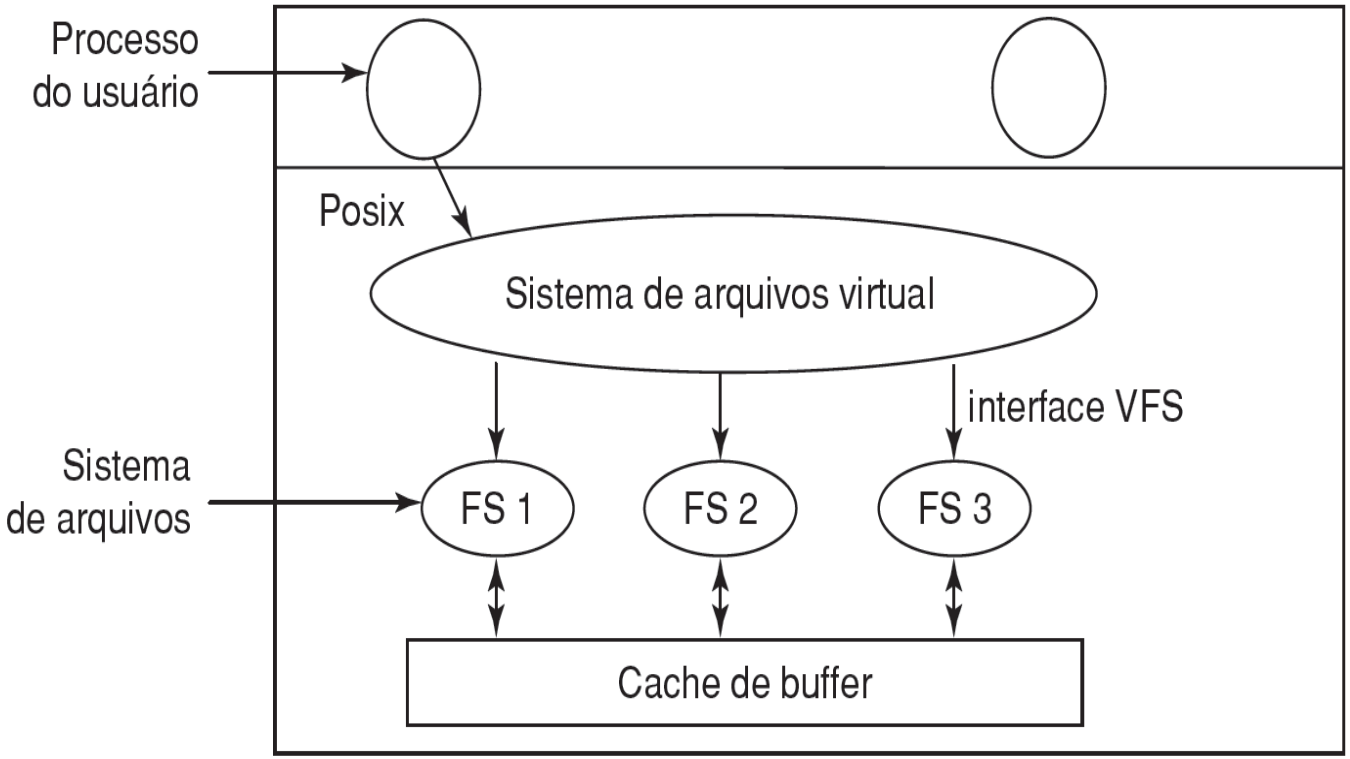


Figura 4.16 Posição do sistema de arquivos virtual.

Sistema de arquivos virtual(3)

Visão geral de funcionamento:

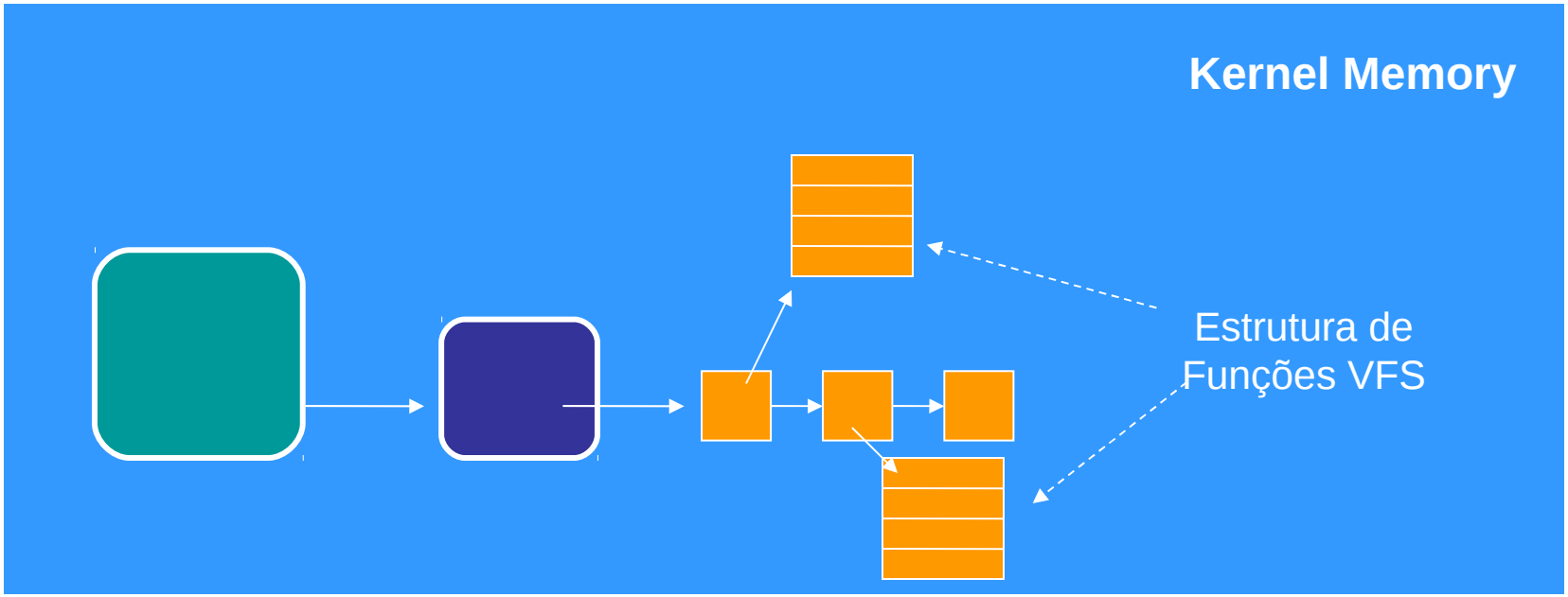
- Todos SAs registrados com o VFS
  - Fornece lista de endereço das funções VFS
  - VFS sabe como executar o SA
- Depois de montado pode ser usado (/usr)
  - open(“/usr/include/unistd.h”, O_RDONLY)
  - Busca o superbloco do SA, diretório raiz e caminho, cria um v-node (ponteiro para tabela de funções), retorna descritor.
  - read(descritor, buffer, tamanho)

VFS - Projeto

Para cada arquivo aberto, um conjunto de funções do SA estão associadas com ele.

São apenas ponteiros para função.

Estes ponteiros para função são chamados funções VFS .



VFS – Projeto

As funções VFS são invocadas pelas chamadas de sistema do SA.

Códigos disponíveis em:

- [kernel root]/fs/open.c, [kernel root]/fs/read_write.c, etc.

* funções definidas em “struct file_operations” no arquivo “[kernel root]/include/linux/fs.h”.

Sistema de arquivos virtual(4)

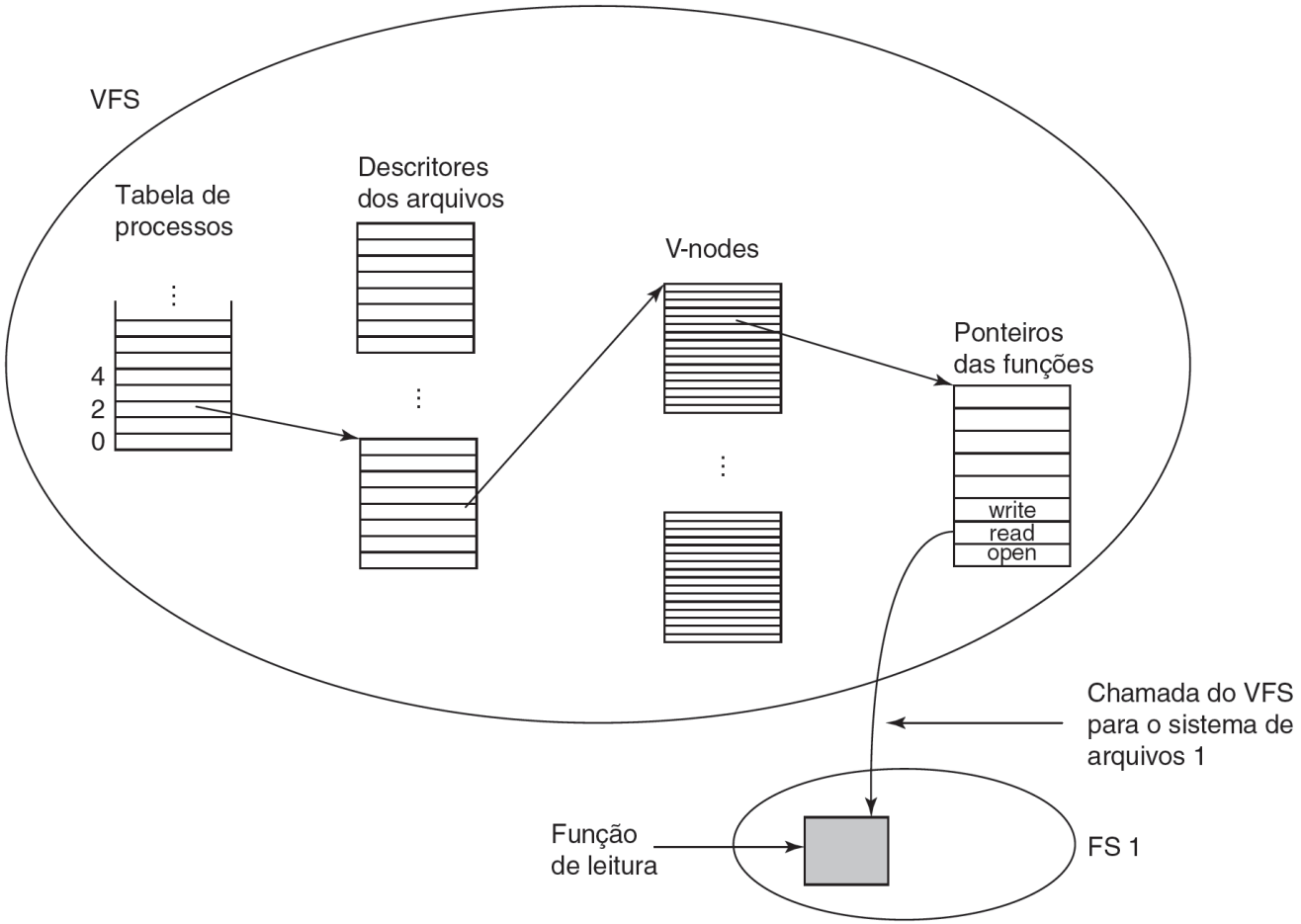


Figura 4.17 Uma visão simplificada das estruturas de dados e do código utilizado pelo VFS e pelo sistema de arquivos real para uma operação read.

VFS Prós e Contras

- Prós
  - Permite o SO dar suporte para diferente sistemas de arquivos.
    - É sabido que, o open() para Ext2 é muito diferente do open() para FAT32.
  - Suportar um futuro SA torna-se fácil.
    - O que é preciso fazer é implementar as funções do SA baseado na estrutura VFS.

- Contras
  - Não pode utilizar funções especiais de um SA.
    - Ex., um SA permite uma operação que combina open() e read() juntos para arquivos pequenos, apenas de leitura.
    - Mas, a estrutura VFS não suporta isto.

Gerência em SA(1)

- Gerenciar o espaço livre nos discos é tarefa do SA
  - Devido ao problema da fragmentação, a grande maioria dos SA quebram os arquivos em blocos de tamanho fixo, que não precisam ser adjacentes.
  - Blocos de tamanho fixo ==> tamanho do bloco?? Ex :
    - Disco 16G → bloco de 1K, número de bloco 32bits
    - Escolha baseada em informações – Tanenbaum (2006)
      - U. Vrije, servidor WEB (www.electoral-vote.com)
      - Bloco 1K , 4K (93% blocos utilizados – 10% maiores arq.)
    - Windows NT – U. Cornell (Vogels, 1999)
      - 1KB lidos, 2,3KB escritos, 4,2KB lidos e escritos

Tamanho de arquivos

Tamanho	UV 1984	UV 2005	Web
1	1,79	1,38	6,67
2	1,88	1,53	7,67
4	2,01	1,65	8,33
8	2,31	1,80	11,30
16	3,32	2,15	11,46
32	5,13	3,15	12,33
64	8,71	4,98	26,10
128	14,73	8,03	28,49
256	23,09	13,29	32,10
512	34,44	20,62	39,94
1 KB	48,05	30,91	47,82
2 KB	60,87	46,09	59,44
4 KB	75,31	59,13	70,64
8 KB	84,97	69,96	79,69

Tamanho	UV 1984	UV 2005	Web
16 KB	92,53	78,92	86,79
32 KB	97,21	85,87	91,65
64 KB	99,18	90,84	94,80
128 KB	99,84	93,73	96,93
256 KB	99,96	96,12	98,48
512 KB	100,00	97,73	98,99
1 MB	100,00	98,87	99,62
2 MB	100,00	99,44	99,80
4 MB	100,00	99,71	99,87
8 MB	100,00	99,86	99,94
16 MB	100,00	99,94	99,97
32 MB	100,00	99,97	99,99
64 MB	100,00	99,99	99,99
128 MB	100,00	99,99	100,00

Tabela 4.3 Porcentual de arquivos menores do que um determinado tamanho (em bytes).

Uma questão de Performance

- Escolha do tamanho do bloco.
  - Se o tamanho do bloco é **PEQUENO**, o número total de blocos é grande.
  - O número de acessos a disco aumenta conforme o número de blocos de cada arquivo aumenta.
  - Porém, isto ajuda reduzir o efeito da fragmentação interna, i.e., perde menos espaço.
  - Ainda, o espaço usado para os dados das listas de representação aumenta, i.e., menos espaço para conteúdo real.
  - Agora, como fica se o tamanho do bloco é grande?

Uma questão de Performance

Arquivos de 4KB e blocos de 1KB, 2KB ou 4KB --- desperdício?
Bloco de 8KB? Bloco de 16KB?
Historicamente tamanhos na faixa 1KB a 4KB

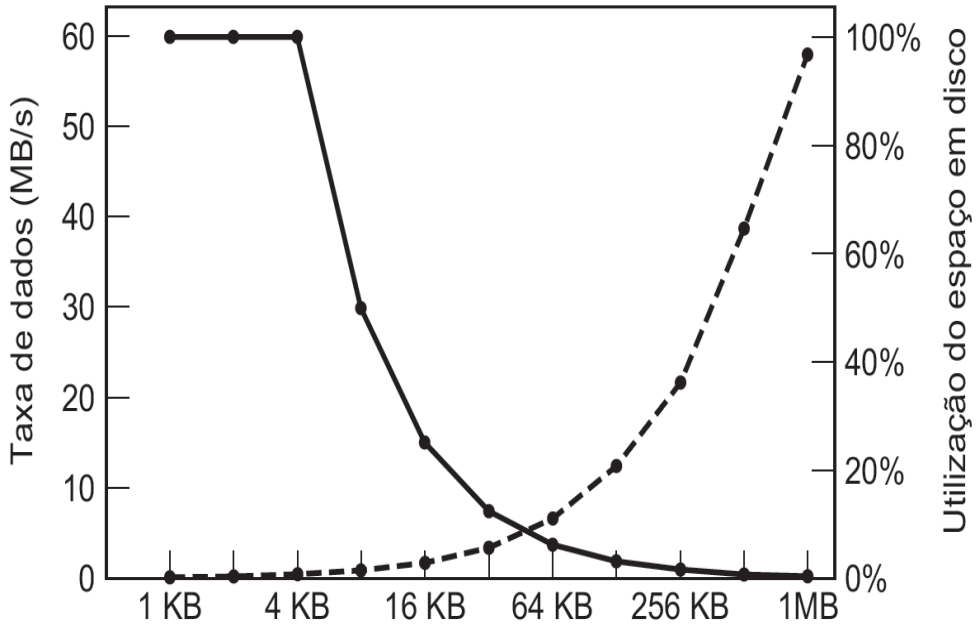


Figura 4.18 A curva tracejada (escala da esquerda) mostra a taxa de dados de um disco. A curva contínua (escala da direita) traz a eficiência do espaço em disco. Todos os arquivos têm 4 KB.

Gerência em SA(2)

- Formas de gerência do espaço livre
  - Lista ligada – lista encadeada de blocos livres
    - Bloco contêm blocos livres - 1K e 32bits ==> 255 blocos/bloco. Ex: 500GB (488 mlhões) requer 1,9 milhão.
  - Mapa de bits – bit representa bloco (1 -l, 0-o)
    - Ex: 60 mil blocos (1-32)
    - Qdo é melhor lista encadeada?
  - Apenas qdo o disco estiver cheio lista ligada precisa menos blocos
  - Se existe muitos blocos livres consecutivos a lista de livre pode controlar conjunto de blocos: contador indica a quantidade de blocos livres, disco fragmentado?
  - Alternativa: manter apenas 1 bloco na memória
    - Ex: fig 4.20 – arquivo de 3 blocos é liberado, depois escrito

Gerência em SA(3)

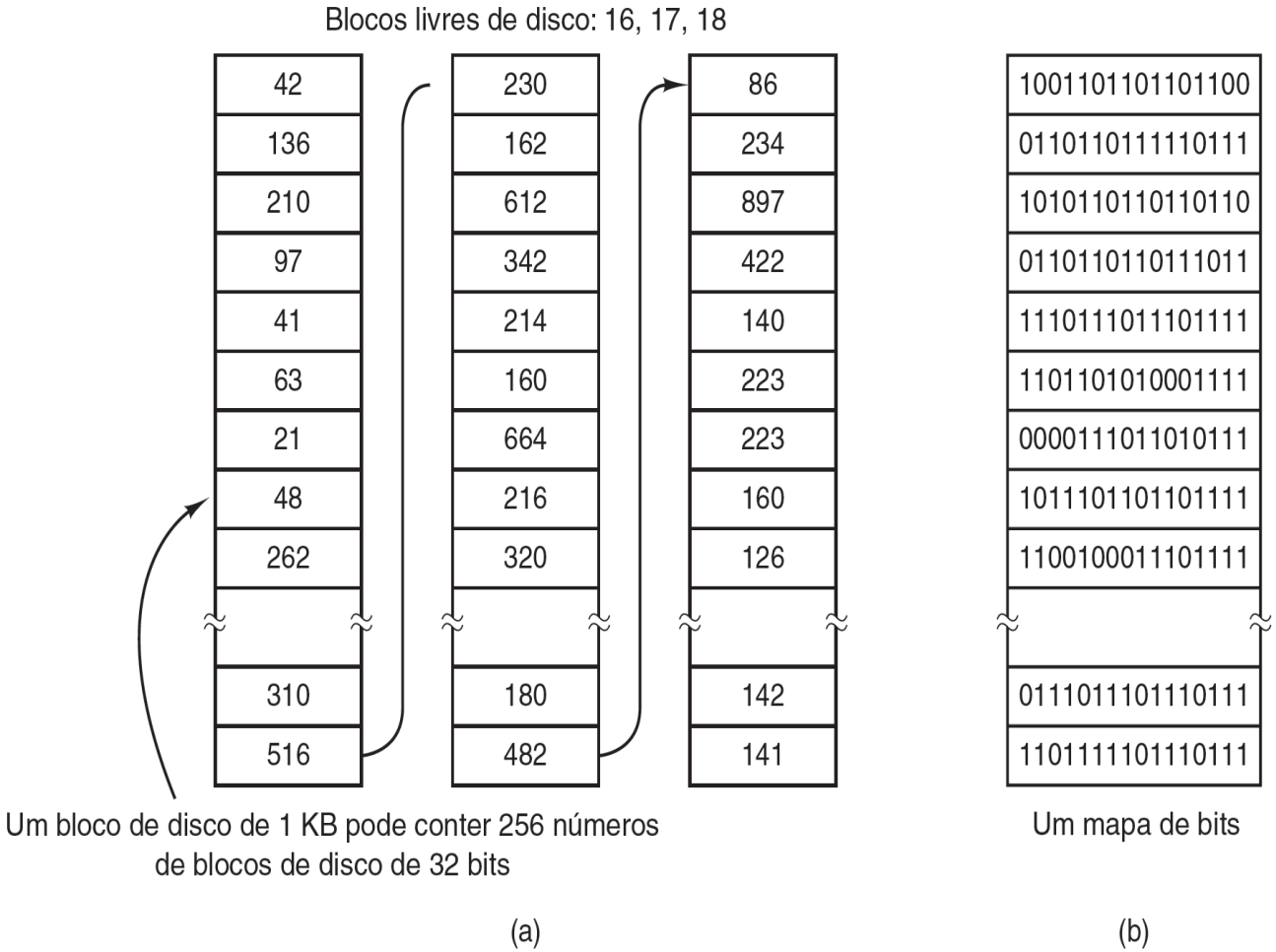


Figura 4.19 (a) Armazenamento da lista de blocos livres em uma lista encadeada. (b) Um mapa de bits.

Gerência em SA(4)

Alternativa

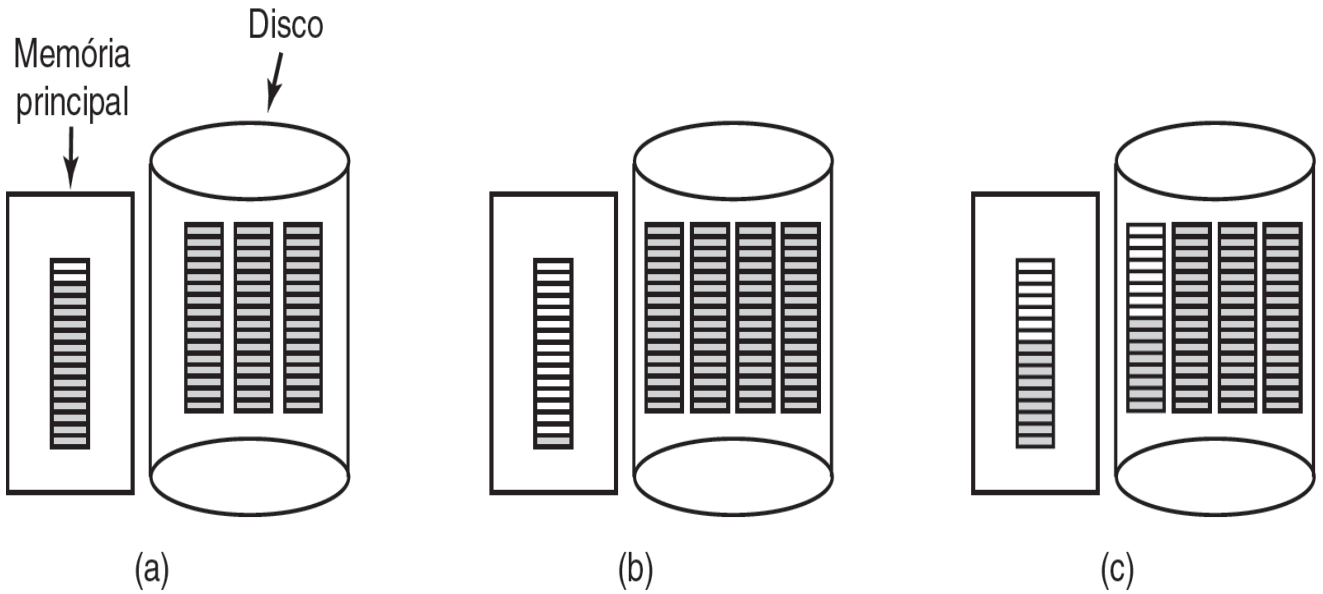


Figura 4.20 (a) Um bloco na memória quase cheio de ponteiros para blocos de disco livres e três blocos de ponteiros em disco. (b) Resultado da liberação de um arquivo de três blocos. (c) Uma estratégia alternativa para lidar com os três blocos livres. As entradas sombreadas representam ponteiros para blocos de disco livres.

Confiabilidade do SA (1)

- Consistência dos sistemas de arquivos
  - Muitos sistemas: lêem blocos, modificam e depois escrevem
  - Se o sistema cair antes da escrita -> inconsistência
  - Crítico se alguns blocos são i-node, blocos de diretório, blocos da lista de livres
  - Utilitários *fsck*, *scandisk*
  - Tipos de verificações
    - Por blocos
      - 2 tabelas (contador para cada bloco=0)
      - Tabela 1- quantas vezes cada bloco está presente arquivo
      - Tabela 2 – quantas vezes cada bloco esta na lista de livres
      - Ler I-nodes, construir lista dos blocos incrementando tabela 1. Depois verifica lista livres, incrementando tabela 2.
    - Por arquivos
      - Tabela de contadores por arquivo
      - Percorre a árvore
        - » Para cada arquivo no diretório incrementa contador, lista por I-node quantos diretórios contém cada arquivo
        - » Contagem no I-node mais alta ou mais baixa -> erro

Confiabilidade do SA (2)

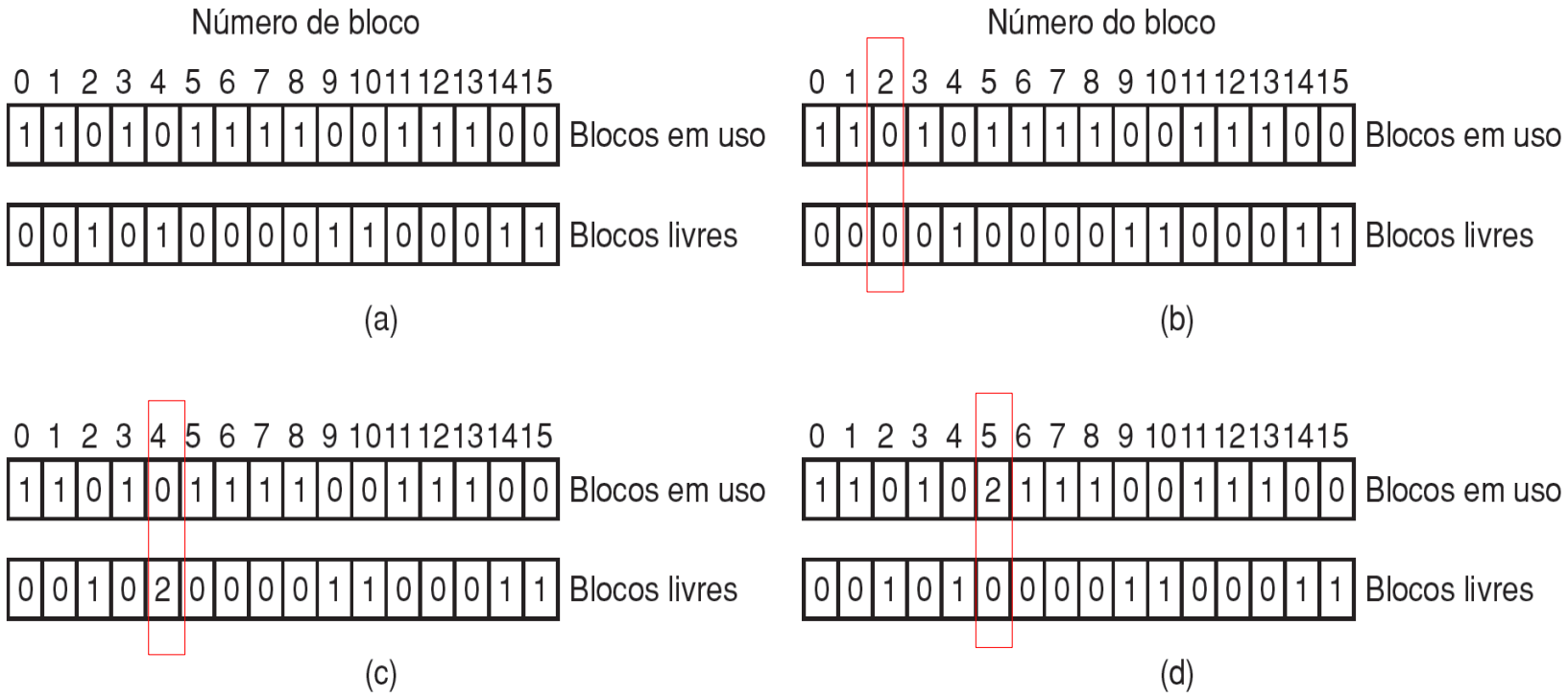


Figura 4.24 Estados do sistema de arquivos. (a) Consistente. (b) Bloco desaparecido. (c) Bloco duplicado na lista de livres. (d) Bloco de dados duplicados.

Confiabilidade do SA (3)

- Robustez diante de falhas – Sistemas *journaling*
  - A premissa básica é a de manter um registro sobre o que o sistema de arquivos irá fazer antes que ele efetivamente o faça, de modo que, se o sistema falhar antes da execução do trabalho planejado, é possível, após a reinicialização do sistema, recorrer ao log para descobrir o que estava acontecendo no momento da parada e retomar o trabalho.
    - Remoção de um arquivo: remove do diretório, libera i-node e libera blocos.
  - O sistema de *journaling* escreve uma entrada no log com as 3 operações e grava em disco, depois de concluir a operação com sucesso a entrada é excluída do log
  - Para que o *journaling* funcione as operações devem ser **idempotentes**: podem ser repetidas sempre que necessário sem causarem nenhum dano
    - Atualize o mapa e assinale o i-node k ou o bloco n como livre
    - Inclusão de novos blocos na lista de livres
  - Ex.: NTFS, ext3

Performance do SA (1)

- O acesso a disco é muito mais lento que o acesso a memória.
  - Cache de blocos é a técnica mais usada para reduzir o tempo de acesso a disco
    - Coleção de blocos de disco que são mantidos na memória, a partir de uma tabela hash onde a remoção é feita a partir do MRU.
    - Modificações no MRU devido a questão da consistência
  - Leitura antecipada de blocos
  - Redução do movimento do braço do disco

Performance do SA (2)

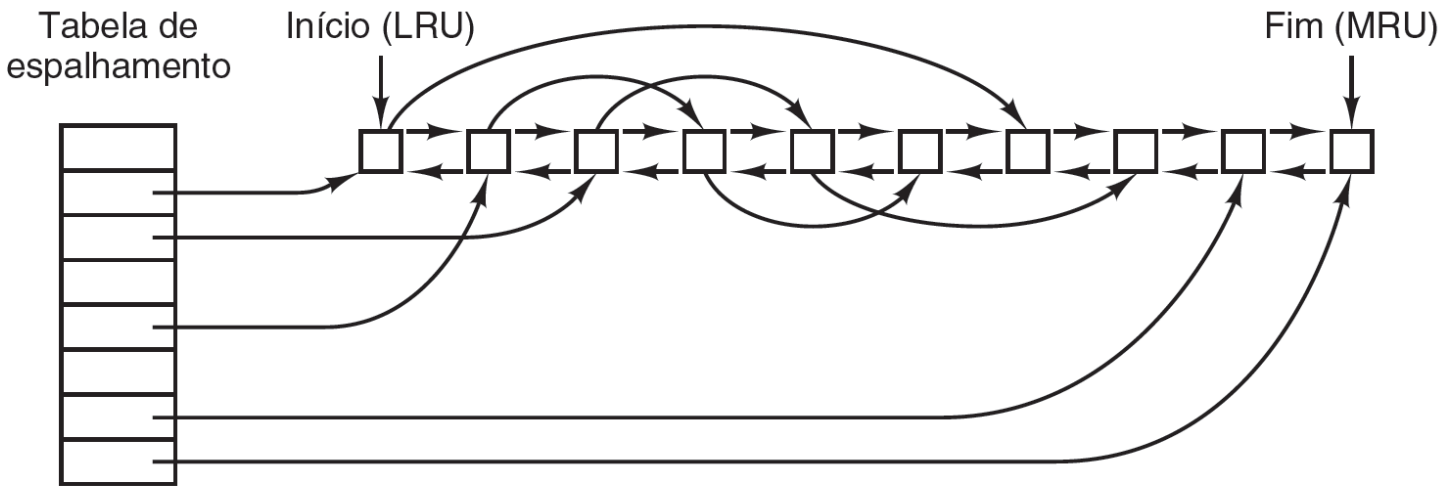


Figura 4.25 A estrutura de dados da cache de buffer.

- É possível manter ordem LRU exata, mas não é viável. Porque?
  - É provável que bloco seja necessário?
  - O bloco é essencial para consistência?
- Categorias de blocos:
  - I-nodes, Blocos indiretos, Blocos de diretório, Blocos de dados totalmente preenchidos e Blocos de dados parcialmente preenchidos

Performance do SA (3)

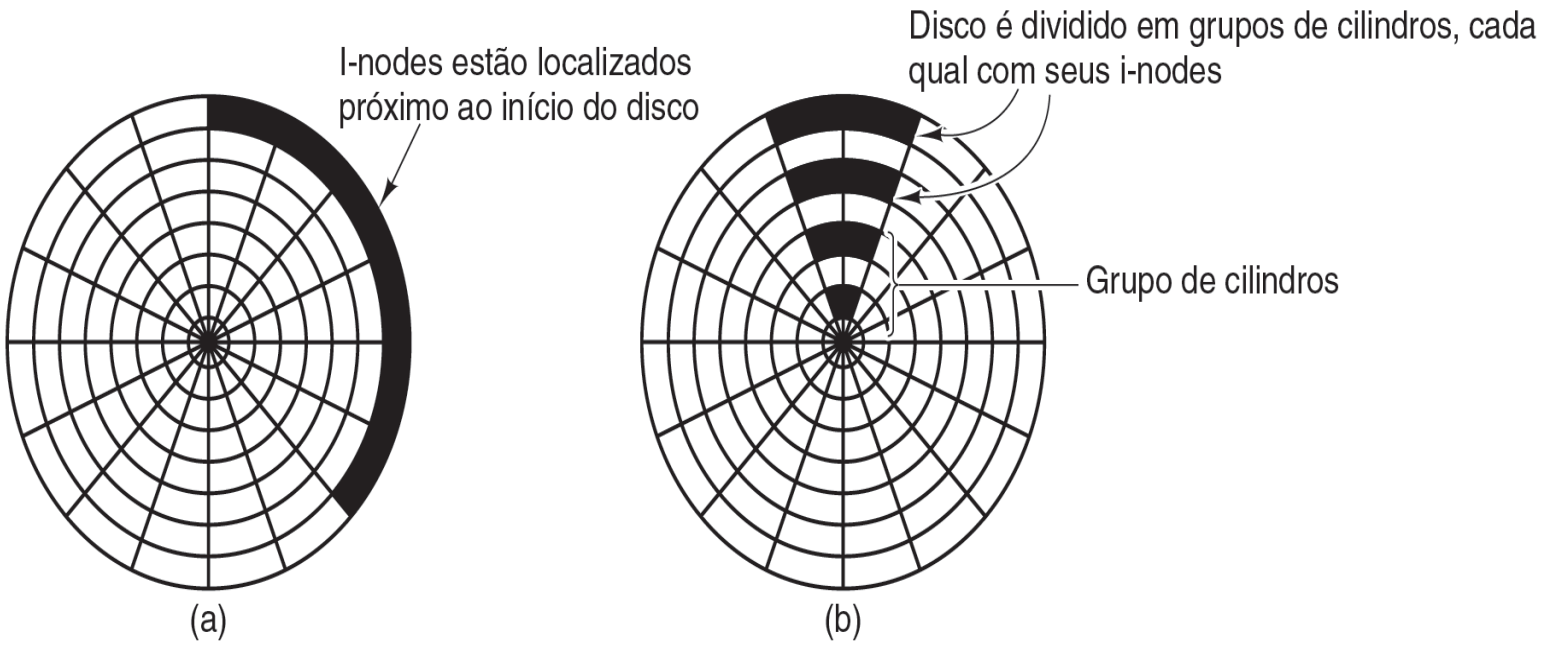


Figura 4.26 (a) I-nodes posicionados no início do disco. (b) Disco dividido em grupos de cilindros, cada um com seus próprios blocos e i-nodes.

Controle de Acesso (1)

- Importante controlar o acesso aos arquivos devido a questões de segurança e de confidencialidade
- Objetivo é evitar acessos indevidos a arquivos
- Baseado na identificação dos usuários
  - Sistema de autenticação padrão (*login name* + senha)
  - Usuários possuem direitos de acessos
- Solução típica:
  - Lista de acesso e grupo

Lista de Acesso

- Consiste em associar a cada arquivo e/ou diretório uma lista de acesso que determina que tipos de acessos são permitidos para cada usuário
- Maior inconveniente é o tamanho da lista
- Uma solução consiste em:
  - Criar classes de usuários
    - e.g.: proprietário, grupo, universo
  - Tipos de acessos
    - e.g: *read, write, modify, execute*

Ex.: UNIX

- Cada objeto oferece 3 bits (*rwx*) para três domínios diferentes: *owner, group* e *others*
- Problema de flexibilidade
  - Quando um usuário pertence a vários grupos ele é identificado por um grupo primário e o arquivo (/etc/groups) mantém todos os grupos a que ele pertence

Exemplo:

```
rwx r-- r--    1  mary  staff    214056  May 30 22:19  windbind.pdf
```