

# SISTEMAS OPERACIONAIS 1

## 21270 A



**Departamento de Computação**  
**Prof. Kelen Cristiane Teixeira Vivaldini**

# Sistema de Arquivos

- Parte do Sistema Operacional mais visível ao usuário
- Os arquivos de um sistema computacional são manipulados por meio de chamadas (*system calls*) ao Sistema Operacional;

# Sistema de Arquivos

- Três importantes requisitos são considerados no armazenamento de informações:
  - Possibilidade de armazenar e recuperar uma grande quantidade de informação;
  - Informação gerada por um processo deve continuar a existir após a finalização desse processo:
    - Ex.: banco de dados;
  - Múltiplos processos podem acessar informações de forma concorrente:
    - Informações podem ser independentes de processos;

# Sistema de Arquivos

- Para atender a esses requisitos, informações são armazenadas em discos (ou alguma outra mídia de armazenamento) em unidades chamadas arquivos;
- Processos podem ler ou escrever em arquivos, ou ainda criar novos arquivos;
- Informações armazenadas em arquivos devem ser persistentes, ou seja, não podem ser afetadas pela criação ou finalização de um processo;

# Sistema de Arquivos

- **SISTEMA de ARQUIVOS**: parte do SO responsável por manipular arquivos!!!

# Sistema de Arquivos

- Usuário: Alto nível
  - Interface → como os arquivos aparecem;
  - Como arquivos são nomeados e protegidos;
  - Quais operações podem ser realizadas;
- SO: Baixo nível
  - Como arquivos são armazenados fisicamente;
  - Como arquivos são referenciados (*links*);

# Sistema de Arquivos

## Arquivos

- Arquivos:
  - Nomes;
  - Estrutura;
  - Tipos;
  - Acessos;
  - Atributos;
  - Operações;

# Sistema de Arquivos

## Nomes de arquivos

- Quando arquivos são criados, nomes são atribuídos a esses arquivos;
- Arquivos são referenciados por meio de seus nomes;
- Tamanho: até 255 caracteres;
  - Restrição: MS-DOS aceita de 1-8 caracteres;
- Letras, números, caracteres especiais podem compor nomes de arquivos:
  - Caracteres permitidos: A-Z, a-z, 0-9, \$, %, ', @, {, }, ~, `, !, #, (, ), &
  - Caracteres **não** permitidos: ?, \*, /, \, ", |, <, >, :



# Sistema de Arquivos

## Nomes de arquivos

- Alguns Sistemas Operacionais são sensíveis (*Case Sensitive*) a letras maiúsculas e minúsculas e outros não;
  - UNIX é sensível :
    - Ex.: exemplo.c é diferente de Exemplo.c;
  - MS-DOS não é sensível:
    - Ex.: exemplo.c é o mesmo que Exemplo.c;
- Win95/Win98/WinNT/Win2000 herdaram características do sistema de arquivos do MS-DOS;
  - No entanto, WinNT/Win2000 possuem um sistema de arquivos próprio → NTFS (*New Technology File System*);

# Sistema de Arquivos

## Nomes de arquivos

- Alguns sistemas suportam uma extensão relacionada ao nome do arquivo:
  - MS-DOS: 1-3 caracteres; suporta apenas uma extensão;
  - UNIX:
    - Extensão pode conter mais de 3 caracteres;
    - Suporta mais de uma extensão: Ex.: exemplo.c.Z (arquivo com compressão);
    - Permite que arquivos sejam criados sem extensão;

# Sistema de Arquivos

## Nomes de arquivos

- Uma extensão, geralmente, associa o arquivo a algum aplicativo (associação feita pelo aplicativo):
  - .doc – Microsoft Word;
  - .c – Compilador C;
- SO pode ou não associar as extensões aos aplicativos:
  - Unix não associa;
  - Windows associa;

# Sistema de Arquivos

## Nomes de arquivos

Extensão	Significado
File.bak	Arquivo de backup
File.gif	Formato de imagens gráficas ( <i>Graphical Interchange Format</i> )
File.hlp	Arquivo de “ajuda”
File.mp3	Arquivo de áudio padrão MPEG ( <i>Moving Picture Expert Group</i> – padrão de compressão de vídeo digital e áudio)
File.pdf	Arquivo <i>Portable Document Format</i>
File.txt	Arquivo texto

# Sistema de Arquivos

## Estrutura de arquivos

- Arquivos podem ser estruturados de diferentes maneiras:
  - a) Seqüência não estruturada de bytes
    - Para o SO arquivos são apenas conjuntos de bytes;
    - SO não se importa com o conteúdo do arquivo;
      - Significado deve ser atribuído pelos programas em nível de usuário (aplicativos);
    - Vantagem:
      - Flexibilidade: os usuários nomeiam seus arquivos como quiserem;
    - Ex.: UNIX e Windows;

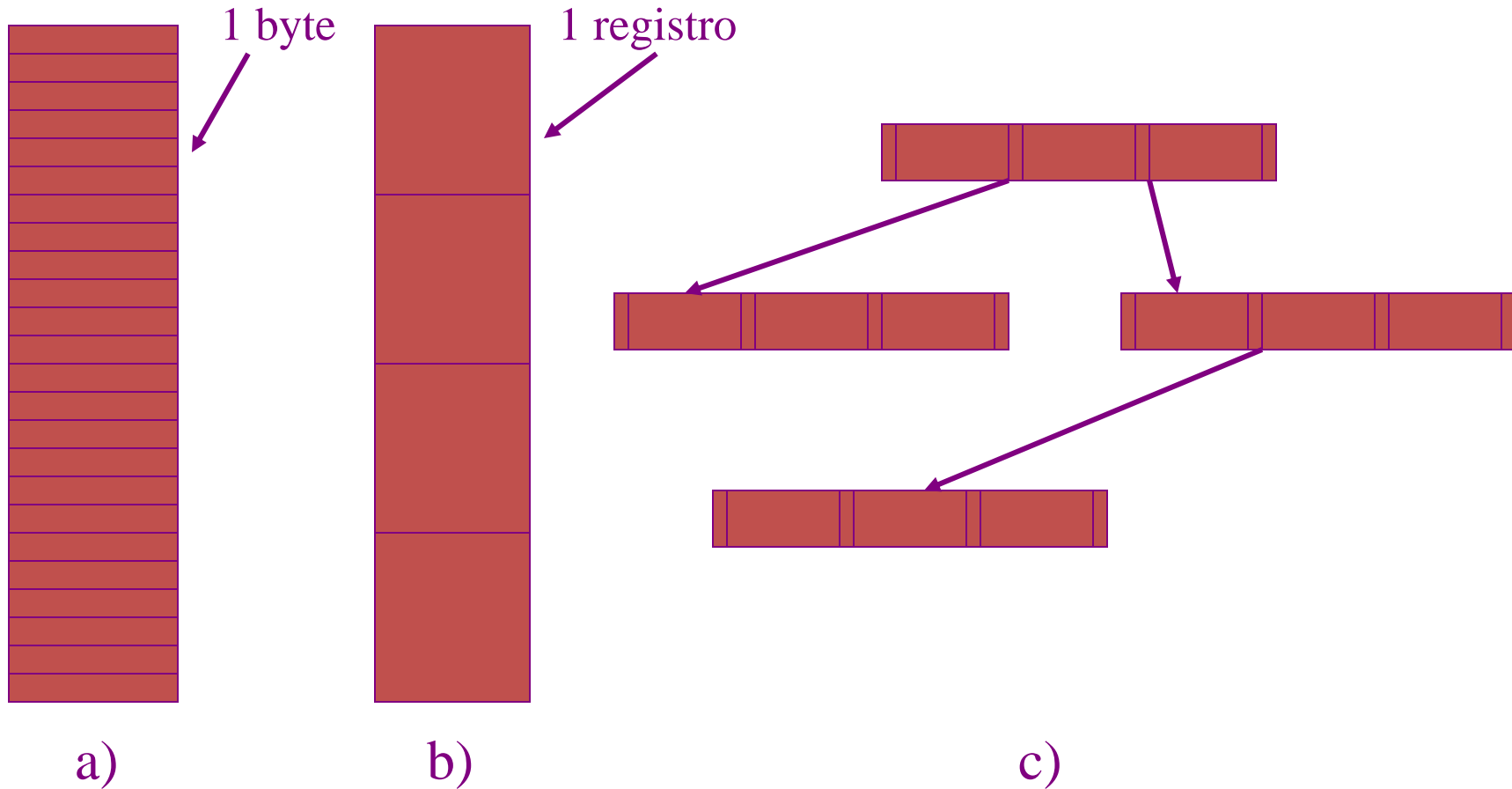
# Sistema de Arquivos

## Estrutura de arquivos

- b) Seqüência de registros de tamanho fixo, cada qual com uma estrutura interna → leitura/escrita são realizadas em registros;
  - SOs mais antigos → *mainframes* e cartões perfurados (80 caracteres);
  - Nenhum sistema atual utiliza esse esquema;
- c) Árvores de registros (tamanho variado), cada qual com um campo **chave** em uma posição fixa:
  - SO decidi onde colocar os arquivos;
  - Usado em *mainframes* atuais;

# Sistema de Arquivos

## Estrutura de arquivos



# Sistema de Arquivos

## Tipos de arquivos

- **Arquivos regulares** → são aqueles que contêm informações dos usuários;
- **Diretórios** → são arquivos responsáveis por manter a estrutura do Sistema de Arquivos;
- **Arquivos especiais de caracteres** → são aqueles relacionados com E/S e utilizados para modelar dispositivos seriais de E/S;
  - Ex.: impressora, interface de rede, terminais;
- **Arquivos especiais de bloco** → são aqueles utilizados para modelar discos;



# Sistema de Arquivos

## Tipos de arquivos

- Arquivos regulares podem ser de dois tipos:
  - ASCII:
    - Consistem de linhas de texto;
    - Facilitam integração de arquivos;
    - Podem ser exibidos e impressos como são;
    - Podem ser editados em qualquer Editor de Texto;
    - Ex.: arquivos texto;
  - Binário:
    - Todo arquivo não ASCII;
    - Possuem uma estrutura interna conhecida pelos aplicativos que os usam;
    - Ex.: programa executável;

# Sistema de Arquivos

## Acessos em arquivos

- SOs mais antigos ofereciam apenas acesso seqüencial no disco → leitura em ordem byte a byte (registro a registro);
- SOs mais modernos fazem acesso randômico ou aleatório;
  - Acesso feito por chave;
    - Ex.: base de dados de uma empresa aérea;
  - Métodos para especificar onde iniciar leitura:
    - Operação `Read` → posição no arquivo que se inicia a leitura;
    - Operação `Seek` → marca posição corrente permitindo leitura seqüencial;

# Sistema de Arquivos

## Atributos de arquivos

- Além do nome e dos dados, todo arquivo tem outras informações associadas a ele → atributos;
- A lista de atributos varia de SO para SO;

# Sistema de Arquivos

## Atributos de arquivos

Atributo	Significado
Proteção	Quem acessa o arquivo e de que maneira
Senha	Chave para acesso ao arquivo
Criador	Identificador da pessoa que criou o arquivo
Dono	Dono corrente
<i>Flag</i> de leitura	0 para leitura/escrita; 1 somente para leitura
<i>Flag</i> de oculto	0 para normal; 1 para não aparecer
<i>Flag</i> de sistema	0 para arquivos normais; 1 para arquivos do sistema
<i>Flag</i> de repositório	0 para arquivos com <i>backup</i> ; 1 para arquivos sem <i>backup</i>

# Sistema de Arquivos

## Atributos de arquivos

Atributo	Significado
<i>Flag ASCII/Binary</i>	0 para arquivo ASCII; 1 para arquivo binário
<i>Flag de acesso aleatório</i>	0 para arquivo de acesso seqüencial; 1 para arquivo de acesso randômico
<i>Flag de temporário</i>	0 para normal; 1 para temporário
<i>Flag de travamento</i>	0 para arquivo desbloqueado; diferente de 0 para arquivo bloqueado
Tamanho do registro	Número de bytes em um registro
Posição da chave	Deslocamento da chave em cada registro
Tamanho da chave	Número de bytes no campo chave ( <i>key</i> )

# Sistema de Arquivos

## Atributos de arquivos

Atributo	Significado
Momento da criação	Data e hora que o arquivo foi criado
Momento do último acesso	Data e hora do último acesso ao arquivo
Momento da última mudança	Data e hora da última modificação do arquivo
Tamanho	Número de bytes do arquivo
Tamanho Máximo	Número máximo de bytes que o arquivo pode ter

# Sistema de Arquivos

## Operações em arquivos

- Diferentes sistemas provêm diferentes operações que permitem armazenar e recuperar arquivos;
- Operações mais comuns (*system calls*):
  - Create; Delete;
  - Open; Close;
  - Read; Write; Append;
  - Seek;
  - Get attributes; Set attributes;
  - Rename;

# Sistema de Arquivos

## Arquivos mapeados em memória

- Alguns SOs permitem que arquivos sejam mapeados diretamente no espaço de endereçamento (virtual) de um processo em execução → acesso mais rápido;
- *System Calls*: Map e unmap;
- Funciona melhor em sistemas que suportam segmentação;



# Sistema de Arquivos

## Arquivos mapeados em memória

- Problemas:
  - Difícil prever o tamanho de arquivos de saída;
  - Compartilhamento de arquivos entre diferentes processos → SO não deve permitir acesso a arquivos com dados inconsistentes;
  - Arquivo pode ser maior que um segmento ou maior que o espaço virtual utilizado → mapear pequenas partes do arquivo;

# Sistema de Arquivos

## Diretórios

- **Diretórios** → são arquivos responsáveis por manter a estrutura do Sistema de Arquivos;
- Organização;
- Operações;

- Organização pode ser feita das seguintes maneiras:
  - Nível único (*Single-level*);
  - Dois níveis (*Two-level*);
  - Hierárquica;

# Sistema de Arquivos

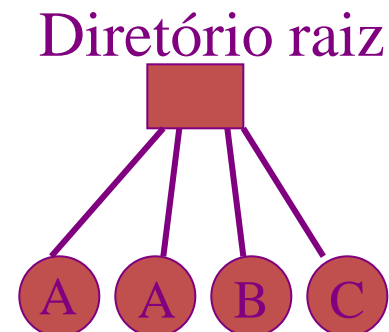
## Diretórios – Nível único

- Apenas um diretório contém todos os arquivos → diretório raiz (*root directory*);
- Computadores antigos utilizavam esse método, pois eram monousuário;
- Exceção: CDC 6600 → supercomputador que utilizava-se desse método, apesar de ser multiusuário;
- Vantagens:
  - Simplicidade;
  - Eficiência;

# Sistema de Arquivos

## Diretórios – Nível único

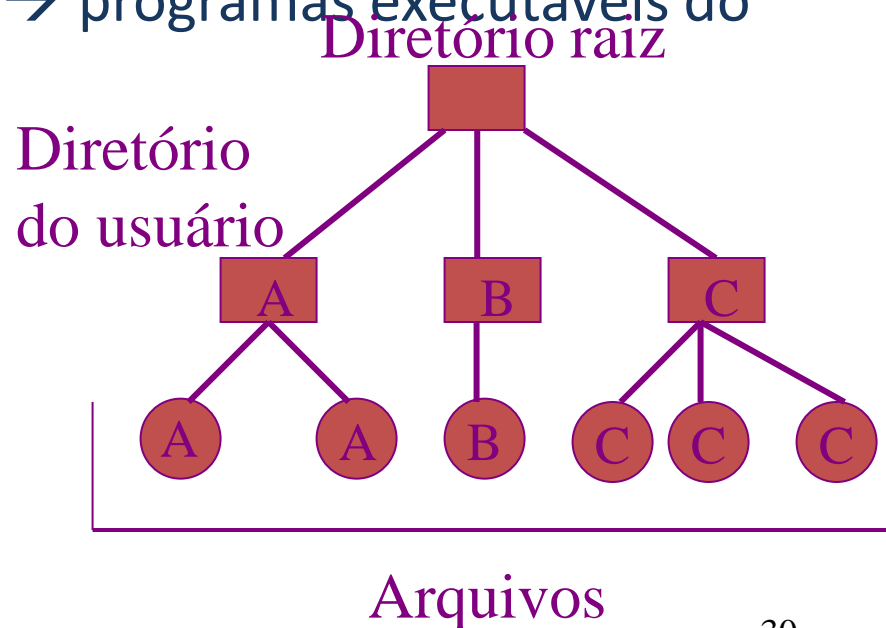
- 04 arquivos;
- Três diferentes proprietários;
- Desvantagens:
  - Sistemas multiusuários: Diferentes usuários podem criar arquivos como mesmo nome;
  - Exemplo:
    - Usuários A e B criam, respectivamente, um arquivo *mailbox*;
    - Usuário B sobrescreve arquivo do usuário A



# Sistema de Arquivos

## Diretórios – Dois níveis

- Cada usuário possui um diretório privado;
- Sem conflitos de nomes de arquivos;
- Procedimento de *login*: identificação;
- Compartilhamento de arquivos → programas executáveis do sistema;
- Desvantagem:
  - Usuário com muitos arquivos;



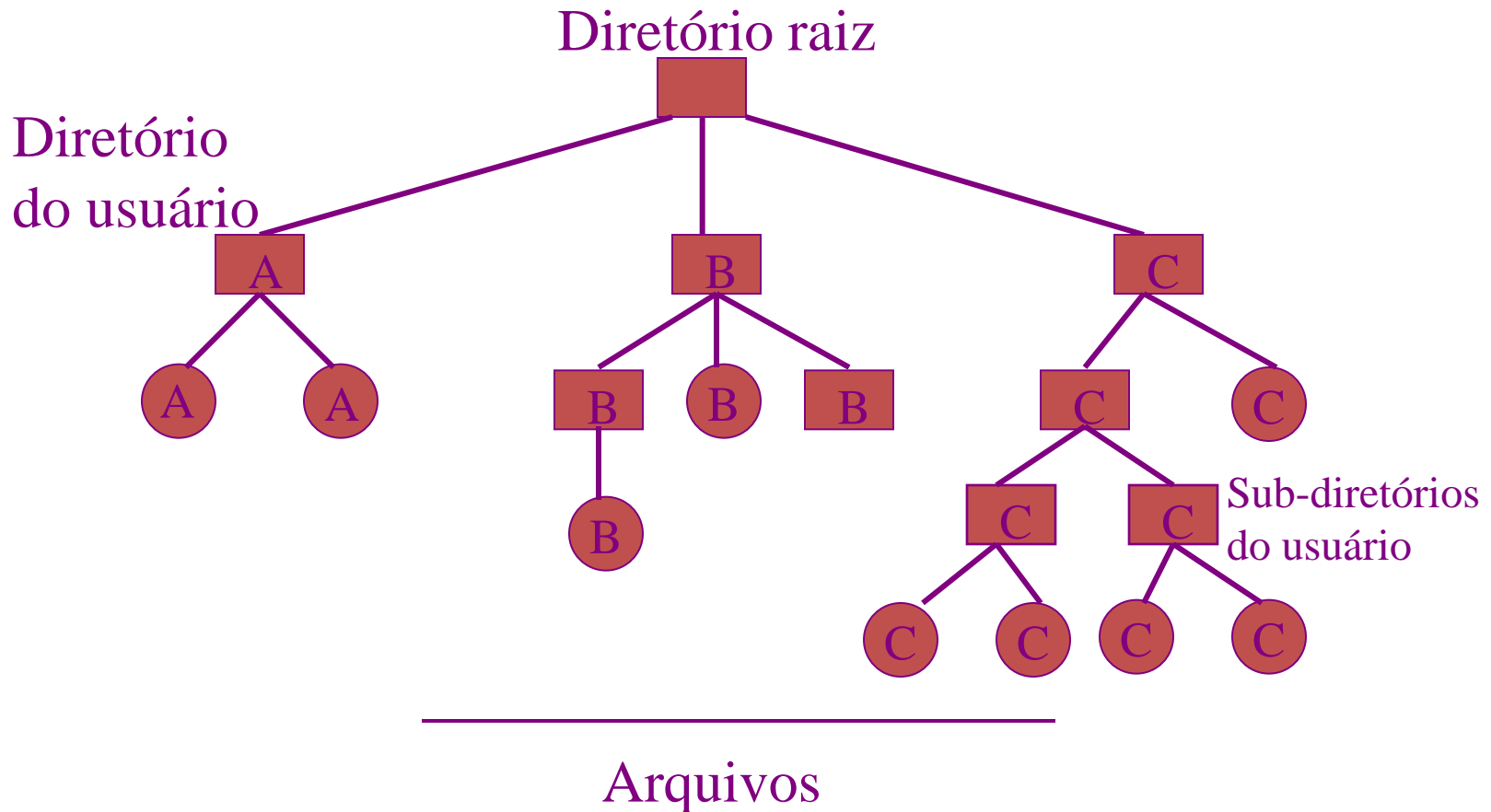
# Sistema de Arquivos

## Diretórios – Hierárquico

- Hierarquia de diretórios → árvores de diretórios;
  - Usuários podem querer agrupar seus arquivos de maneira lógica, criando diversos diretórios que agrupam arquivos;
- Sistemas operacionais modernos utilizam esse método;
- Flexibilidade;

# Sistema de Arquivos

## Diretórios – Hierárquico





# Sistema de Arquivos

## Diretórios – Caminho (*path name*)

- O método hierárquico requer métodos pelos quais os arquivos são acessados;
- Dois métodos diferentes:
  - Caminho absoluto (*absolute path name*);
  - Caminho relativo (*relative path name*);

# Sistema de Arquivos

## Diretórios – Caminho (*path name*)

- Caminho absoluto: consiste de um caminho a partir do diretório raiz até o arquivo;
  - É ÚNICO;
  - Funciona independentemente de qual seja o diretório corrente;
  - Ex.:
    - UNIX: `/usr/ast/mailbox`;
    - Windows: `\usr\ast\mailbox`;

# Sistema de Arquivos

## Diretórios – Caminho (*path name*)

- Diretório de Trabalho (*working directory*) ou diretório corrente (*current directory*);
- Caminho relativo é utilizado em conjunto com o diretório corrente;
- Usuário estabelece um diretório como sendo o diretório corrente; nesse caso caminhos não iniciados no diretório raiz são tido como relativos ao diretório corrente;
  - Exemplo:
    - `cp /usr/ast/mailbox /usr/ast/mailbox.bak`
    - Diretório corrente: `/usr/ast` → `cp mailbox mailbox.bak`

# Sistema de Arquivos

## Diretórios – Caminho (*path name*)

- “.” → diretório corrente;
- “..” → diretório pai (anterior ao corrente);
- Ex.: diretório corrente /usr/ast:

```
cp ../lib/dictionary .
```

```
cp /usr/lib/dictionary .
```

```
cp /usr/lib/dictionary dictionary
```

```
cp /usr/lib/dictionary /usr/ast/dictionary
```

# Sistema de Arquivos

## Diretórios – Operações

- Create; Delete;
- Opendir; Closedir;
- Readdir;
- Rename;
- Link (um arquivo pode aparecer em mais de um diretório);
- Unlink;

# Implementando o Sistema de arquivos

- Implementação do Sistema de Arquivos:
  - Como arquivos e diretórios são armazenados;
  - Como o espaço em disco é gerenciado;
  - Como tornar o sistema eficiente e confiável;

# Implementando o Sistema de arquivos

## *Layout*

- Arquivos são armazenados em discos;
- Discos podem ser divididos em uma ou mais partições, com sistemas de arquivos independentes;
- Setor 0 do disco é destinado ao MBR – *master boot record*; que é responsável pela tarefa de *boot* do computador;
  - MBR possui a tabela de partição, com o endereço inicial e final de cada partição;
  - BIOS lê e executa o MBR;

# Implementando o Sistema de arquivos

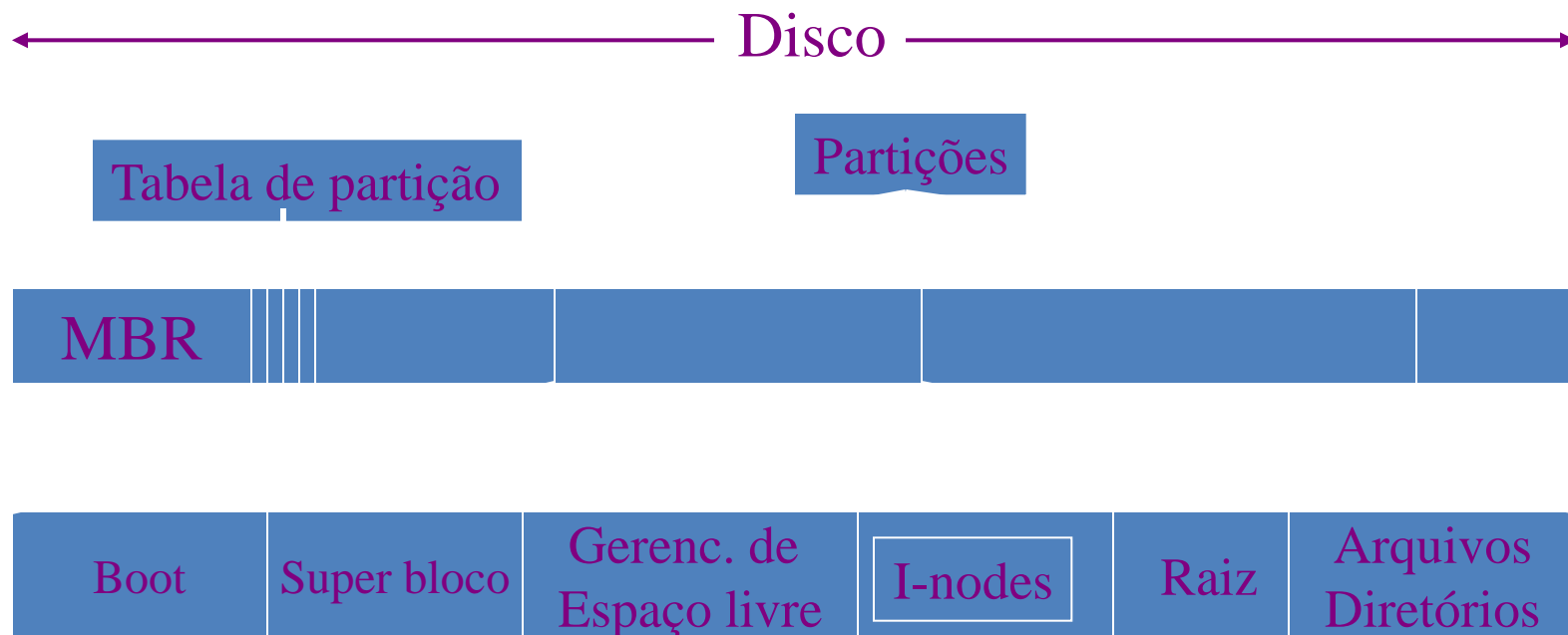
## *Layout*

- Tarefas básicas do MBR (pode variar dependendo do SO):
  - 1ª → localizar a partição ativa;
  - 2ª → ler o primeiro bloco dessa partição, chamado bloco de *boot* (*boot block*);
  - 3ª → executar o bloco de *boot* ;
- *Layout* de um Sistema de Arquivos pode variar; mas a ideia geral é a seguinte:



# Implementando o Sistema de arquivos

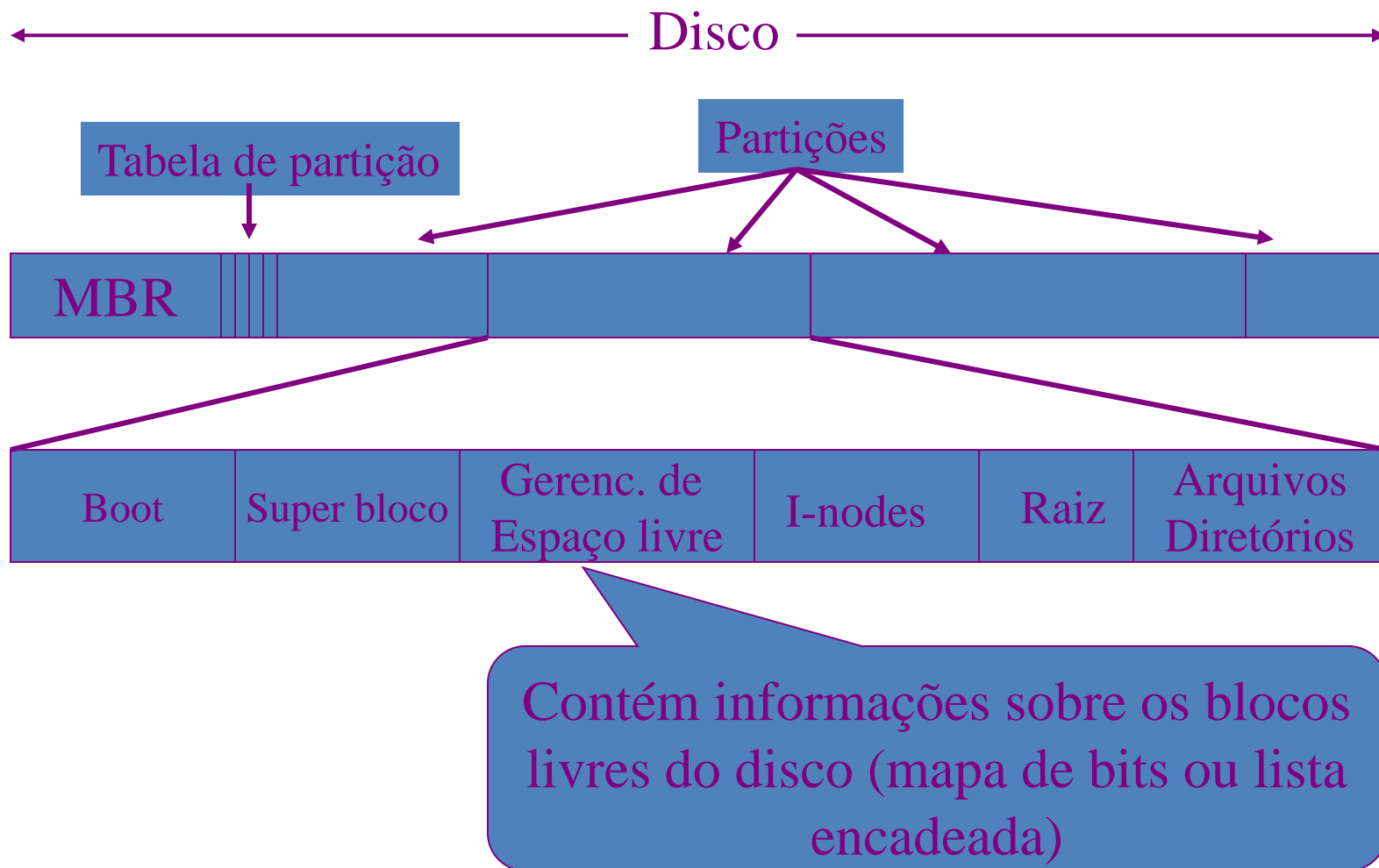
## *Layout*



Contém parâmetros (tipo do SA, número de blocos) sobre o sistema de arquivos e é carregado na memória

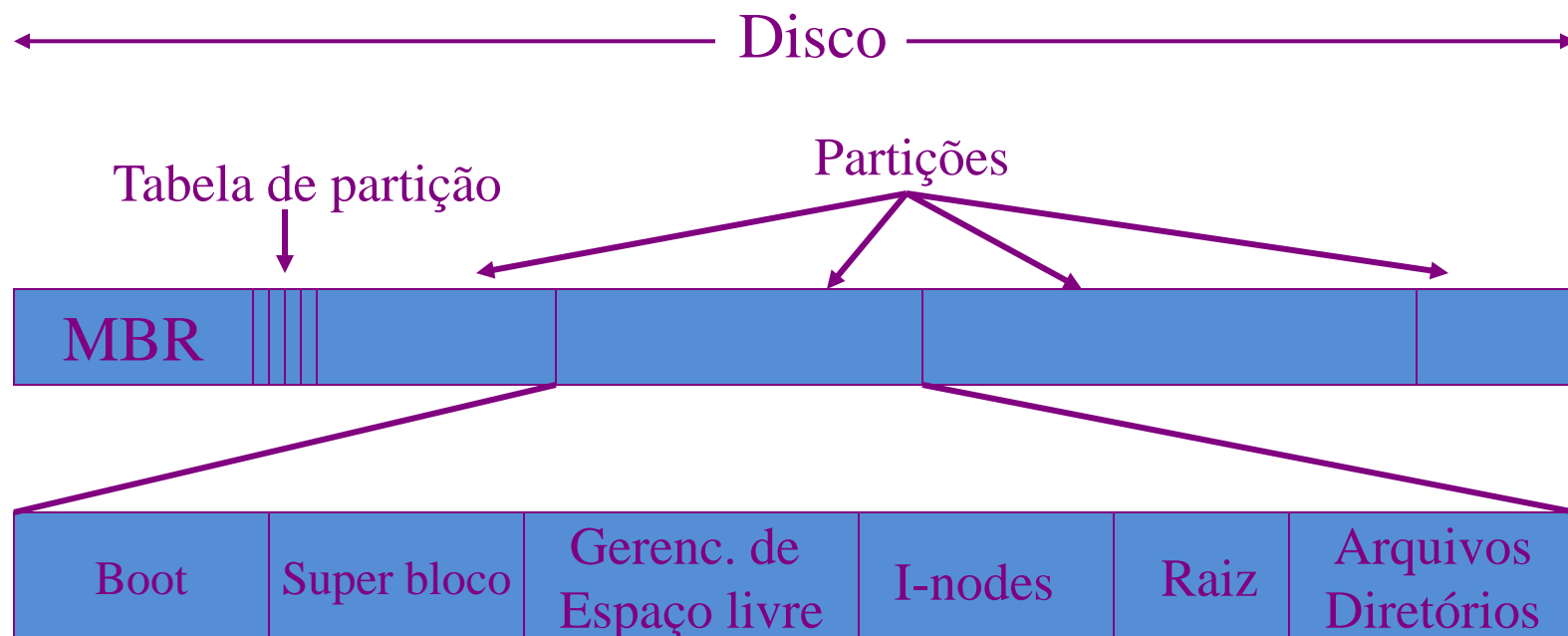
# Implementando o Sistema de arquivos

## *Layout*



# Implementando o Sistema de arquivos

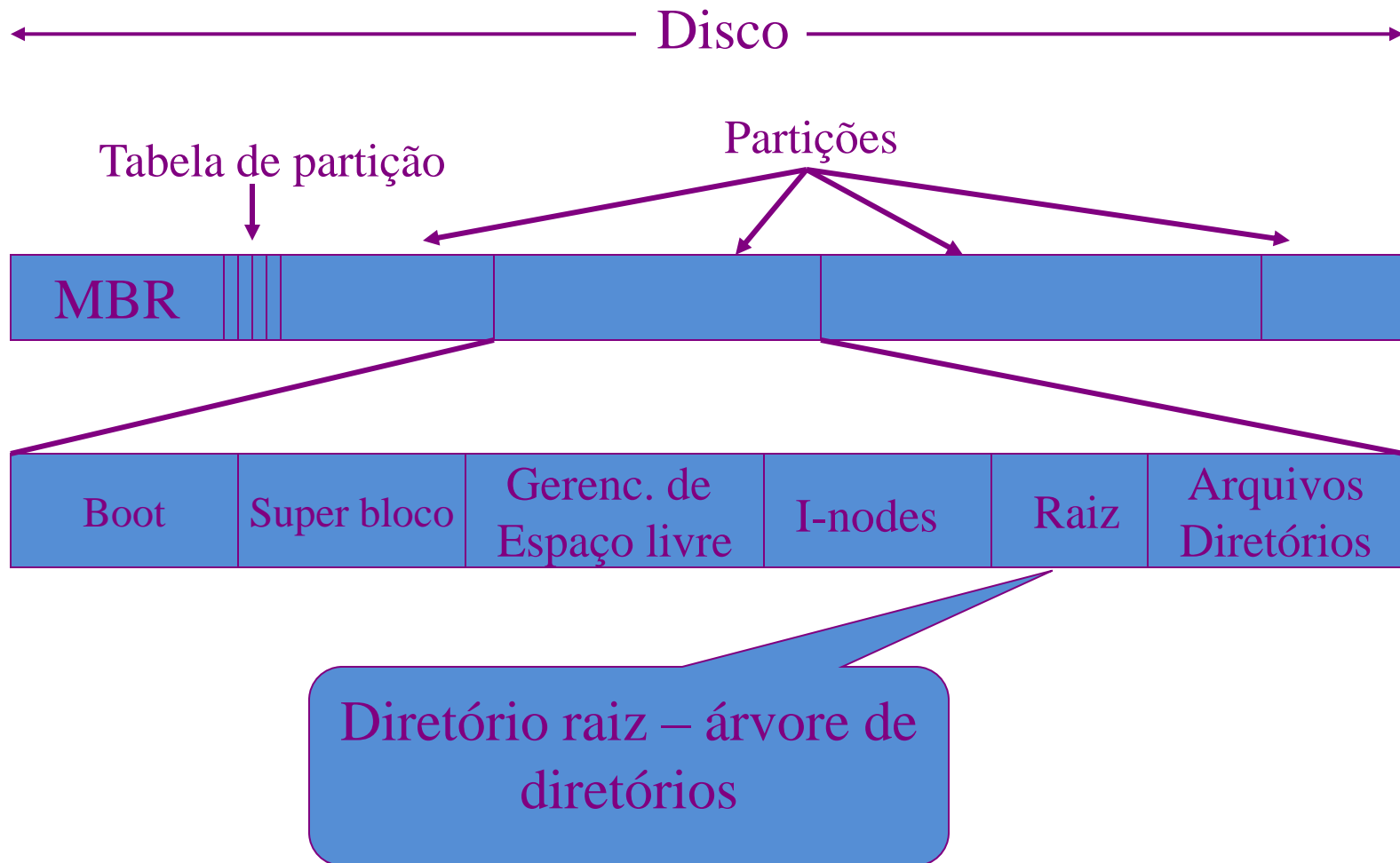
## *Layout*



Estruturas de dados (vetor) contendo informações sobre os arquivos

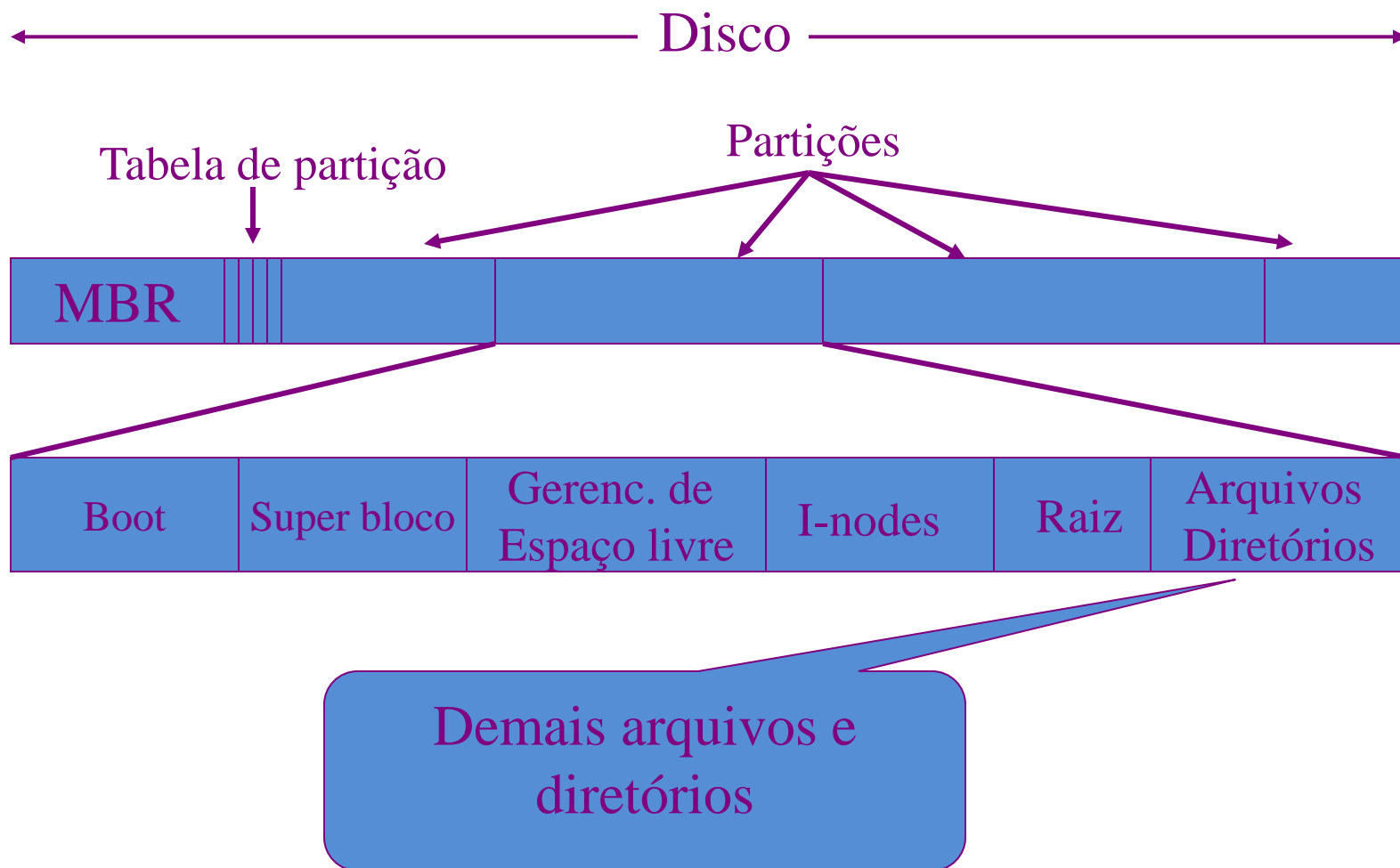
# Implementando o Sistema de arquivos

## *Layout*



# Implementando o Sistema de arquivos

## *Layout*



# Implementando o Sistema de arquivos

## Arquivos

- Armazenamento de arquivos → como os arquivos são alocados no disco;
- Diferentes técnicas são implementadas por diferentes Sistemas Operacionais;
  - Alocação contínua;
  - Alocação com lista encadeada;
  - Alocação com lista encadeada utilizando uma tabela na memória (FAT);
  - *I-Nodes*;

# Implementando o Sistema de arquivos

## Arquivos

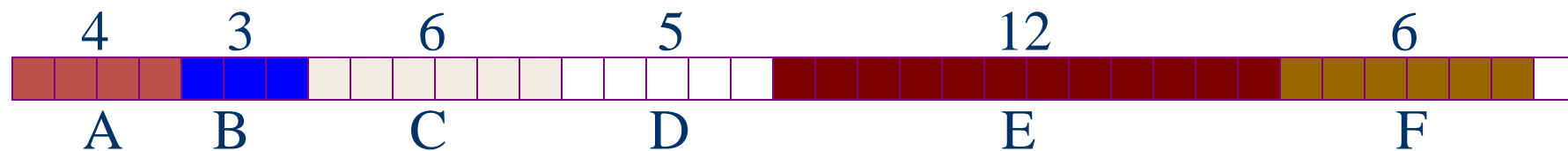
- Alocação contínua:
  - Técnica mais simples;
  - Armazena arquivos de forma contínua no disco;
    - Ex.: em um disco com blocos de 1kb um arquivo com 50kb será alocado em 50 blocos consecutivos;

# Implementando o Sistema de arquivos

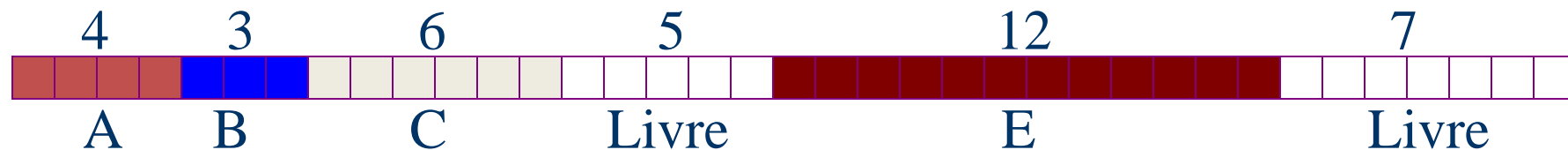
## Arquivos

### Alocação contínua:

37 Blocos



Removendo os arquivos D e F...





# Implementando o Sistema de arquivos

## Arquivos

### Alocação contínua:

- Vantagens:
  - Simplicidade: somente o endereço do primeiro bloco e número de blocos no arquivo são necessários;
  - Desempenho para o acesso ao arquivo: acesso seqüencial;
- Desvantagens (discos rígidos):
  - Fragmentação externa:
    - Compactação → alto custo;
    - Réuso de espaço → atualização da lista de espaços livres;
      - » Conhecimento prévio do tamanho do arquivo para alocar o espaço necessário;
- CD-ROM e DVD-ROM;

# Implementando o Sistema de arquivos

## Arquivos

- **Alocação com lista encadeada:**
  - A primeira palavra de cada bloco é um ponteiro para o bloco seguinte;
  - O restante do bloco é destinado aos dados;
  - Apenas o endereço em disco do primeiro bloco do arquivo é armazenado;
    - Serviço de diretório é responsável por manter esse endereço;

# Implementando o Sistema de arquivos

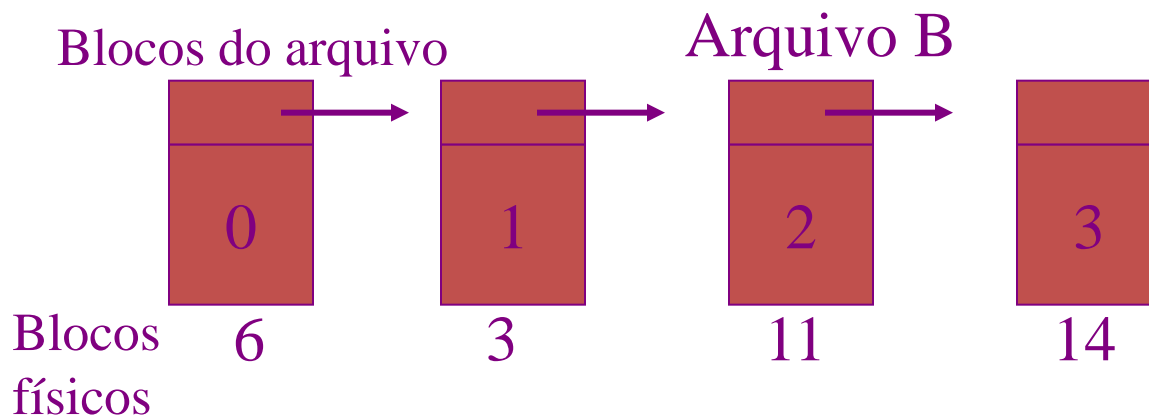
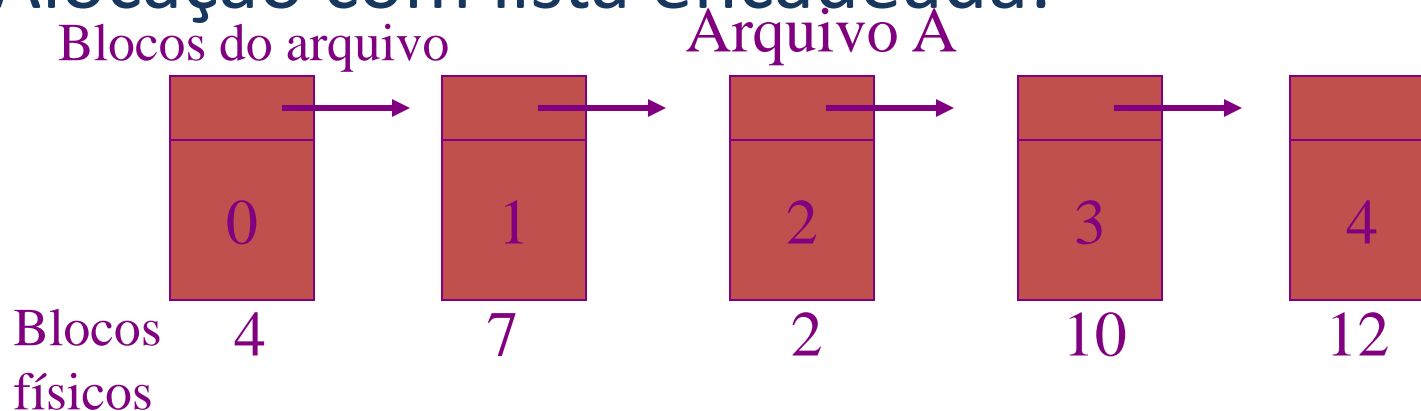
## Arquivos

- **Alocação com lista encadeada:**
  - Desvantagens:
    - Acesso aos arquivos é feito randomicamente → processo mais lento;
    - A informação armazenada em um bloco não é mais uma potência de dois, pois existe a necessidade de se armazenar o ponteiro para o próximo bloco;
  - Vantagem:
    - Não se perde espaço com a fragmentação externa;

# Implementando o Sistema de arquivos

## Arquivos

- Alocação com lista encadeada:



# Implementando o Sistema de arquivos

## Arquivos

- Alocação com lista encadeada utilizando uma tabela na memória:
  - O ponteiro é colocado em uma tabela na memória ao invés de ser colocado no bloco;
    - FAT → Tabela de alocação de arquivos (*File Allocation Table*);
    - Assim, todo o bloco está disponível para alocação de dados;
  - Serviço de diretório é responsável por manter o início do arquivo (bloco inicial);
  - MS-DOS e família Windows 9x (exceto WinNT, Win2000 e WinXP - NTFS);

# Implementando o Sistema de arquivos

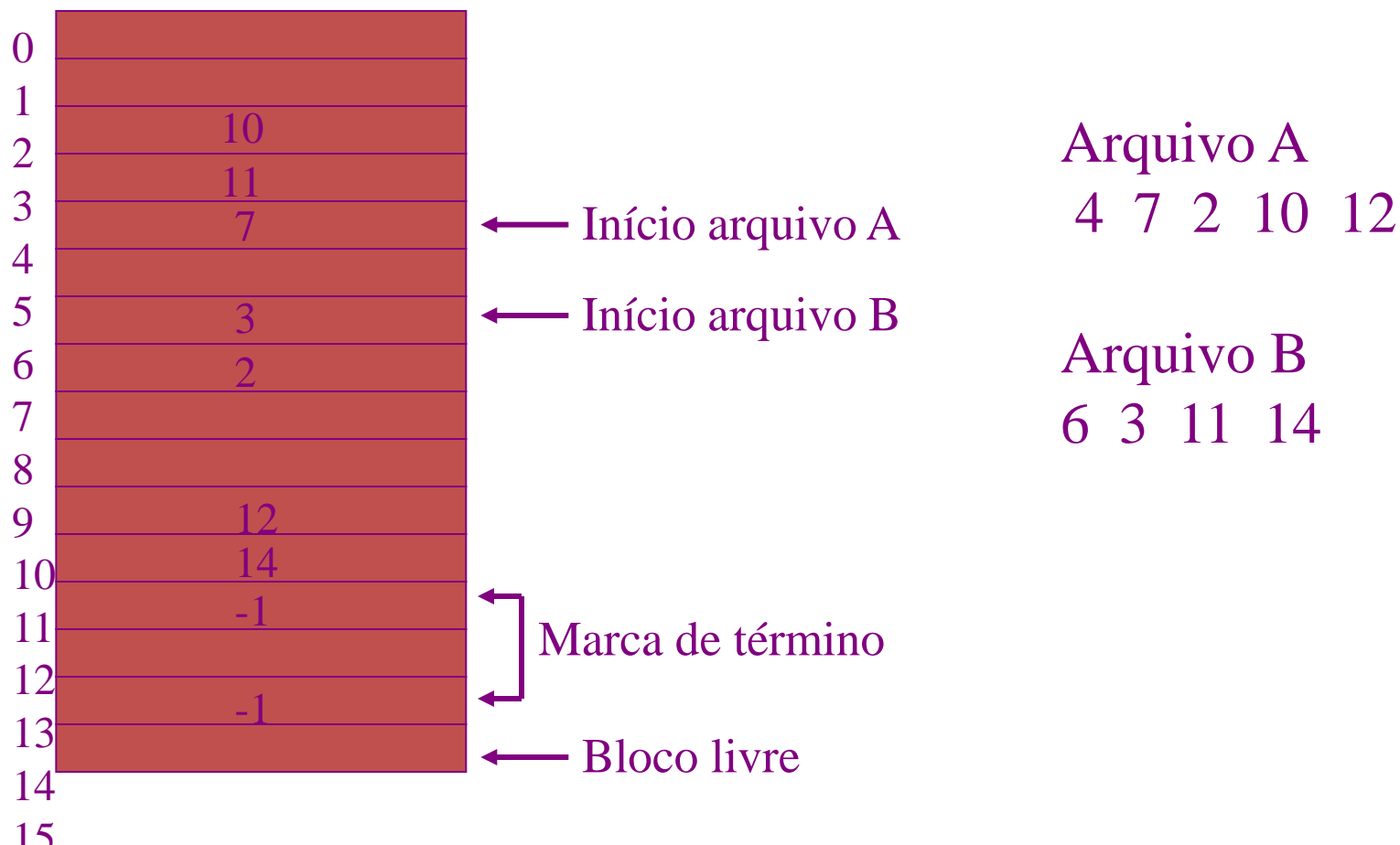
## Arquivos

- Acesso randômico se torna mais fácil devido ao uso da memória;
- Desvantagem:
  - Toda a tabela deve estar na memória;
  - Exemplo:
    - Com um disco de 20Gb com blocos de 1kb, a tabela precisa de 20 milhões de entradas, cada qual com 3 bytes (para permitir um acesso mais rápido, cada entrada pode ter 4 bytes) ocupando 60 (80) Mb da memória;

# Implementando o Sistema de arquivos

## Arquivos

### ■ Alocação com lista encadeada utilizando FAT



# Implementando o Sistema de arquivos

## Arquivos

- *I-nodes*:
  - Cada arquivo possui uma estrutura de dados chamada *i-node* (*index-node*) que lista os atributos e endereços em disco dos blocos do arquivo;
    - Assim, dado o *i-node* de um arquivo é possível encontrar todos os blocos desse arquivo;
  - Se cada *i-node* ocupa **n** bytes e **k** arquivos podem estar aberto ao mesmo tempo → o total de memória ocupada é **kn** bytes;
  - UNIX e Linux;



# Implementando o Sistema de arquivos

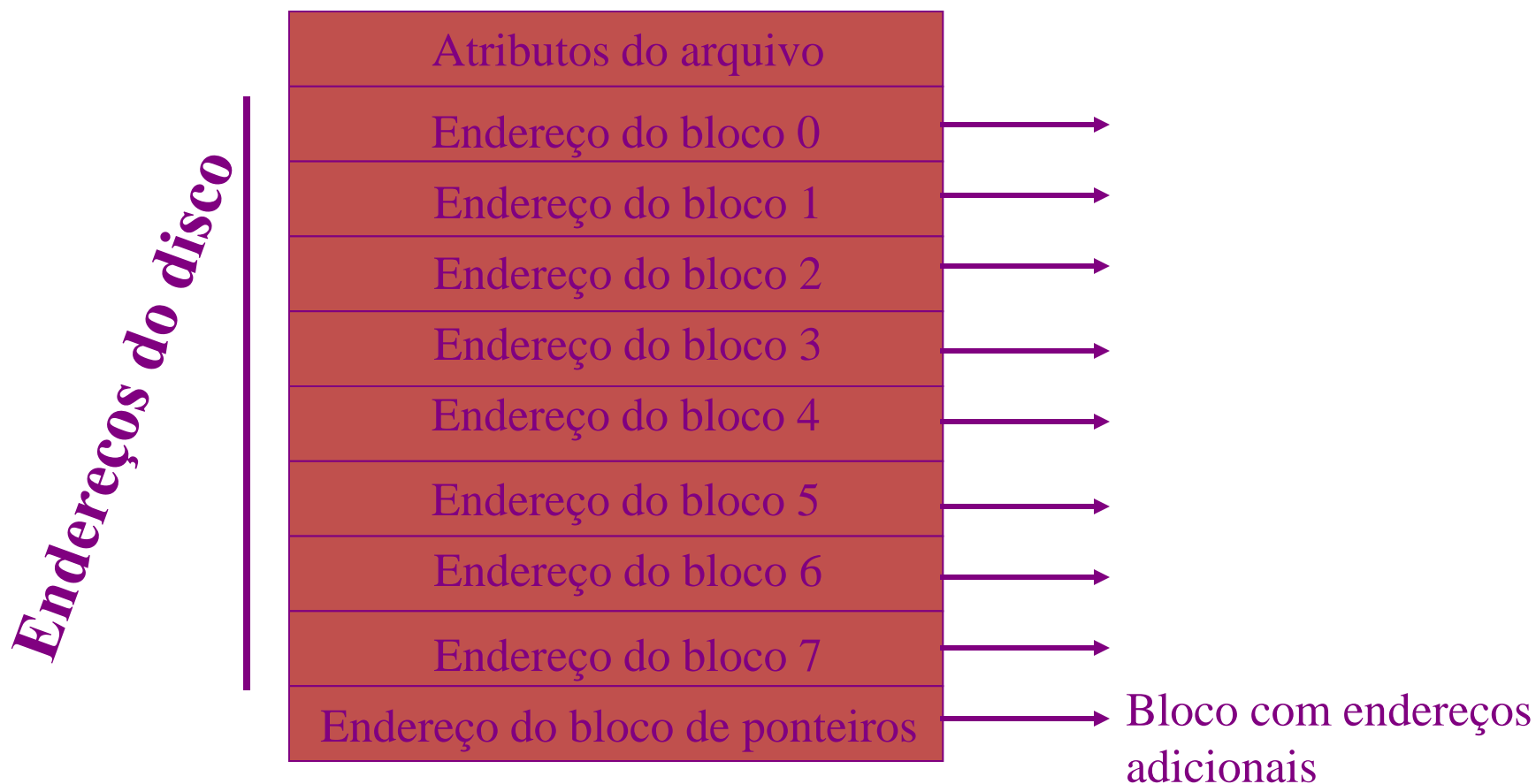
## Arquivos

- Espaço de memória ocupado pelos *i-nodes* é proporcional ao número de arquivos abertos; enquanto o espaço de memória ocupado pela tabela de arquivo (FAT) é proporcional ao tamanho do disco;
- Vantagem:
  - O *i-node* somente é carregado na memória quando o seu respectivo arquivo está aberto (em uso);
- Desvantagem:
  - O tamanho do arquivo pode aumentar muito
    - Solução: reservar o último endereço para outros endereços de blocos;

# Implementando o Sistema de arquivos

## Arquivos

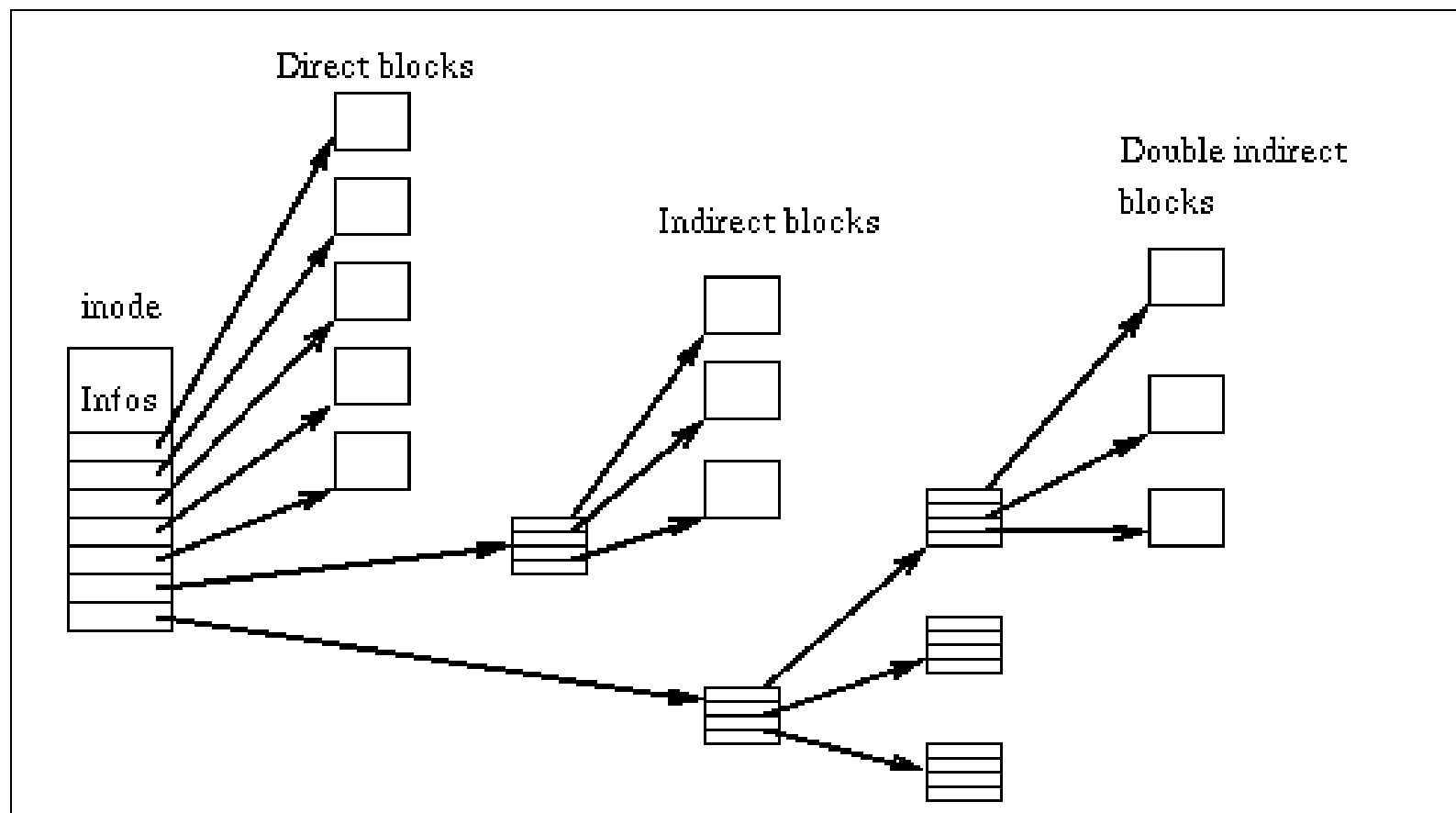
### ■ *I-nodes*:



# Implementando o Sistema de arquivos

## Arquivos

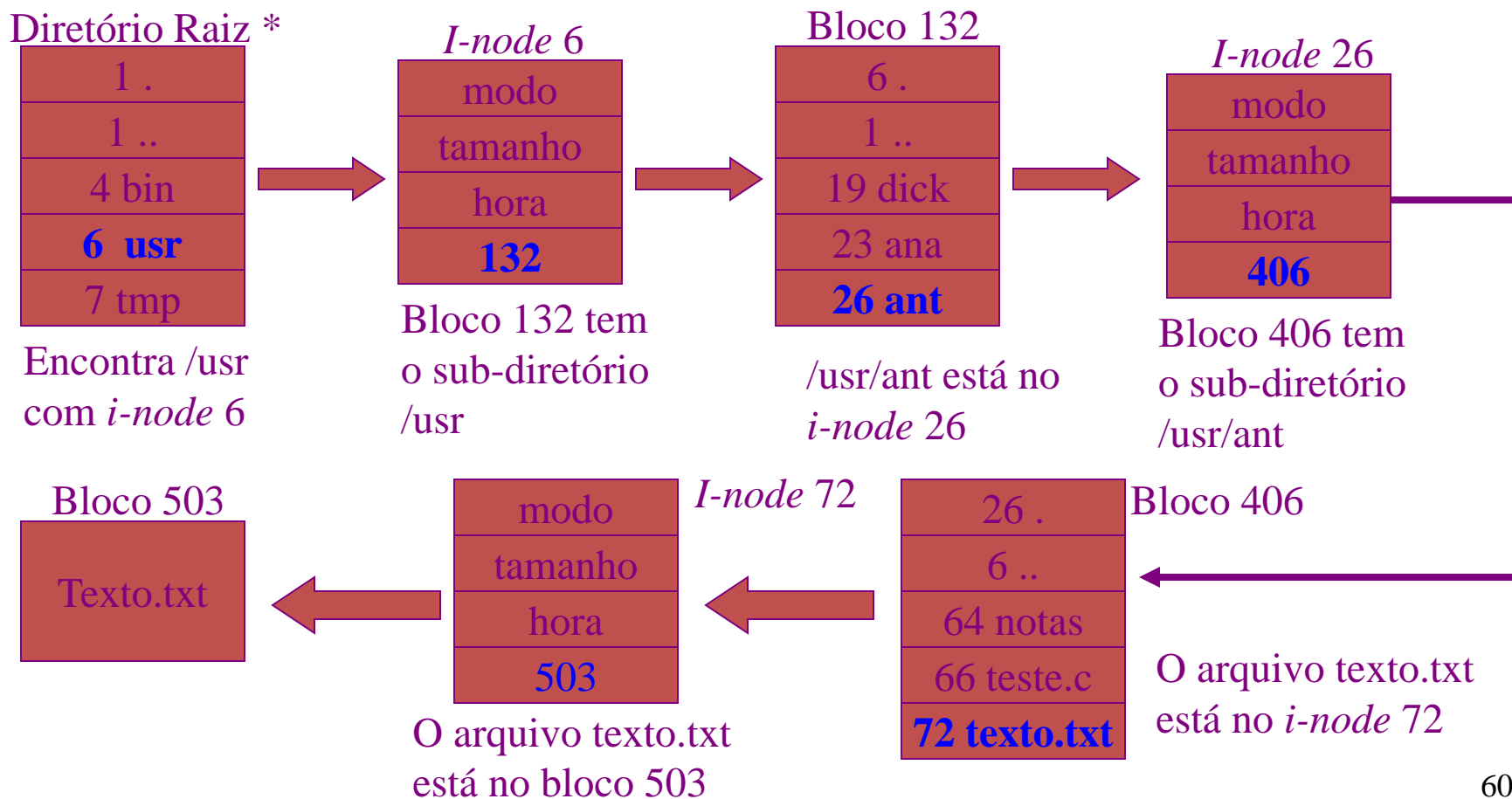
### ■ *I-nodes:*



# Implementando o Sistema de arquivos

## Arquivos

### ■ *I-nodes*



# Implementando o Sistema de Arquivos Diretórios

- Quando um arquivo é aberto, o Sistema Operacional utiliza-se do caminho para localizar o diretório de entrada;
- O diretório de entrada provê as informações necessárias para encontrar os blocos no disco nos quais o arquivo está armazenado → serviço de diretório é responsável por mapear o nome ASCII do arquivo na informação:
  - Endereço do arquivo inteiro (alocação contínua);
  - Número do primeiro bloco do arquivo (alocação com listas encadeadas);
  - Número do *i-node*;

# Implementando o Sistema de Arquivos Diretórios

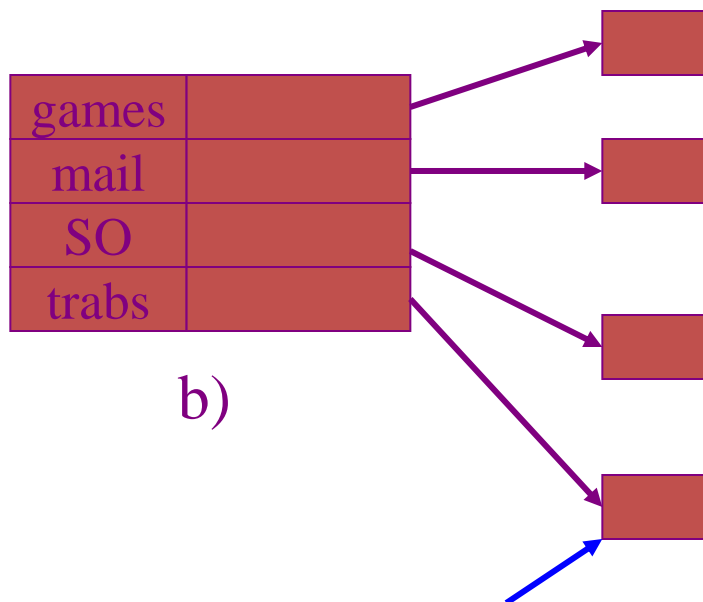
- O serviço de diretório também é responsável por manter armazenados os atributos relacionados a um arquivo:
  - A) Entrada do Diretório: Diretório consiste de uma lista de entradas com tamanho fixo (uma para cada arquivo) contendo um nome de arquivo, uma estrutura de atributos de arquivos, e um ou mais endereços de disco; MS/DOS e Windows;

# Implementando o Sistema de Arquivos Diretórios

- B) *I-node*: nesse caso, o diretório de entrada é menor, armazenando somente o nome de arquivo e o número do *i-node*; UNIX

games	atributos
mail	atributos
SO	atributos
trabs	atributos

a)



b)

Estrutura de dados  
contendo atributos (*i-nodes*)

# Implementando o Sistema de Arquivos Diretórios

- Tratamento de nomes de arquivos:
  - Maneira mais simples: limite de 255 caracteres reservados para cada nome de arquivo:
    - Toda entrada de diretório tem o mesmo tamanho;
    - Desvantagem: desperdício de espaço, pois poucos arquivos utilizam o total de 255 caracteres;
  - Maneira mais eficiente: tamanho do nome do arquivo é variável;



# Implementando o Sistema de Arquivos Diretórios

## Tamanho do nome de arquivo variável

Tamanho da entrada do A1			
Atributos A1			
p	r	o	j
e	c	t	-
b	u	d	☒
Tamanho da entrada do A2			
Atributos A2			
p	e	r	s
o	n	n	e
l	☒		

- Cada nome do arquivo é preenchido de modo a ser composto por um número inteiro de palavras (parte sombreada);

**Problema:** se arquivo é removido, um espaço em branco é inserido;

•  
•  
a)

# Implementando o Sistema de Arquivos Diretórios

## Tamanho do nome de arquivo variável

Tamanho da entrada do A1			
Atributos A1			
p	r	o	j
e	c	t	-
b	u	d	⊠
Tamanho da entrada do A2			
Atributos A2			
p	e	r	s
o	n	n	e
l	⊠		
•			
•			

a)

Ponteiro p/ nome A1			
Atributos A1			
Ponteiro p/ nome A12			
Atributos A2			
p	r	o	j
e	c	t	-
b	u	d	⊠
p	e	r	s
o	n	n	e
l	⊠		
•			
•			

HEAP

b)

# Implementando o Sistema de Arquivos Diretórios

- Busca em diretório:
  - Linear → lenta para diretórios muito grandes;
  - Uma tabela *Hash* para cada diretório:
    - O nome do arquivo é submetido a uma função *hash* para selecionar uma entrada na tabela *hash*;
      - Cria-se uma lista encadeada para todas as entradas com o mesmo valor *hash*;
    - Vantagem: busca mais rápida;
    - Desvantagem: gerenciamento mais complexo;
  - *Cache* de busca → ótima para poucas consultas de arquivos;

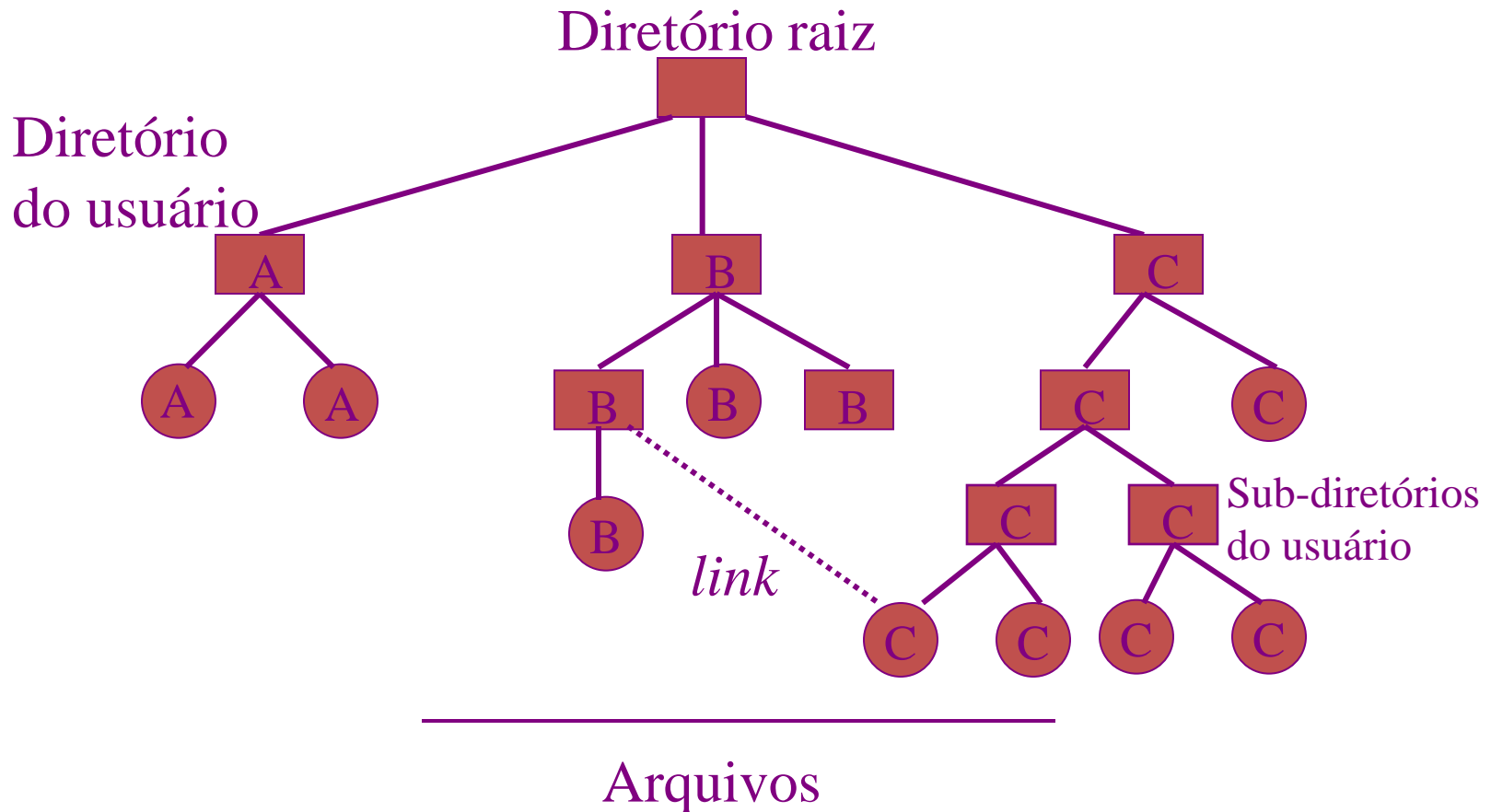
# Implementando o Sistema de Arquivos

## Arquivos Compartilhados

- Normalmente, o sistema de arquivos é implementado com uma árvore;
- Mas quando se tem arquivos compartilhados, o sistema de arquivos é implementado como um grafo acíclico direcionado (*directed acyclic graph – DAG*);
  - *Links* são criados;

# Implementando o Sistema de Arquivos

## Arquivos Compartilhados



# Implementando o Sistema de Arquivos

## Arquivos Compartilhados

- Compartilhar arquivos é sempre conveniente, no entanto, alguns problemas são introduzidos:
  - Se os diretórios tiverem **endereços de disco**, então deverá ser feita uma cópia dos endereços no diretório de B;
  - Se B ou C adicionar blocos ao arquivo (*append*), os novos blocos serão relacionados somente no diretório do usuário que está fazendo a adição;
    - Mudanças não serão visíveis ao outro usuário, comprometendo o propósito do compartilhamento;

# Implementando o Sistema de Arquivos

## Arquivos Compartilhados

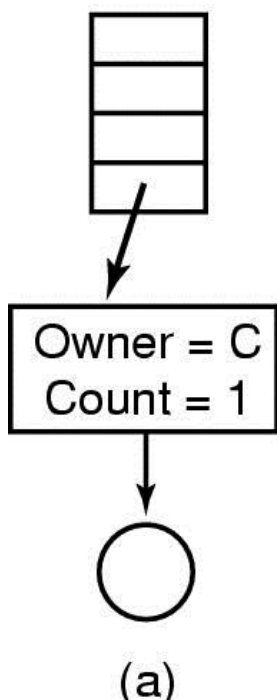
- Soluções:
  - Primeira solução: os **endereços de disco** não estão relacionados nos diretórios, mas em uma estrutura de dados (*i-node*) associada ao próprio arquivo. Assim, os diretórios apontam para essa estrutura; (UNIX)
    - Ligação Estrita (*hard link*);
    - Problema com essa solução: o dono do arquivo que está sendo compartilhado apaga o arquivo;

# Implementando o Sistema de Arquivos

## Arquivos Compartilhados

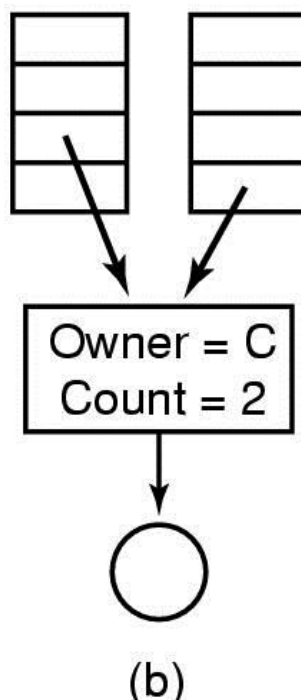
### Ligação Estrita

C's directory

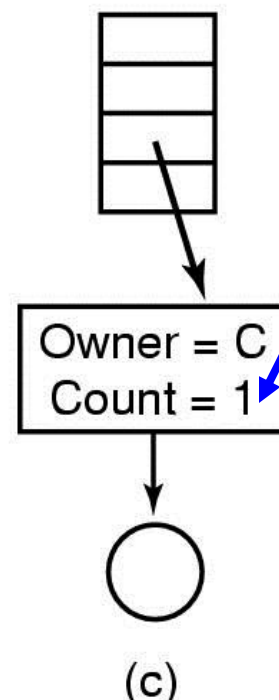


B's directory

C's directory



B's directory



Remove entrada de C, mas deixa o contador i-node intacto; B continua a usar o arquivo;

a) Antes da ligação;

b) Depois da ligação;

c) Depois de remover a entrada de C para o arquivo;

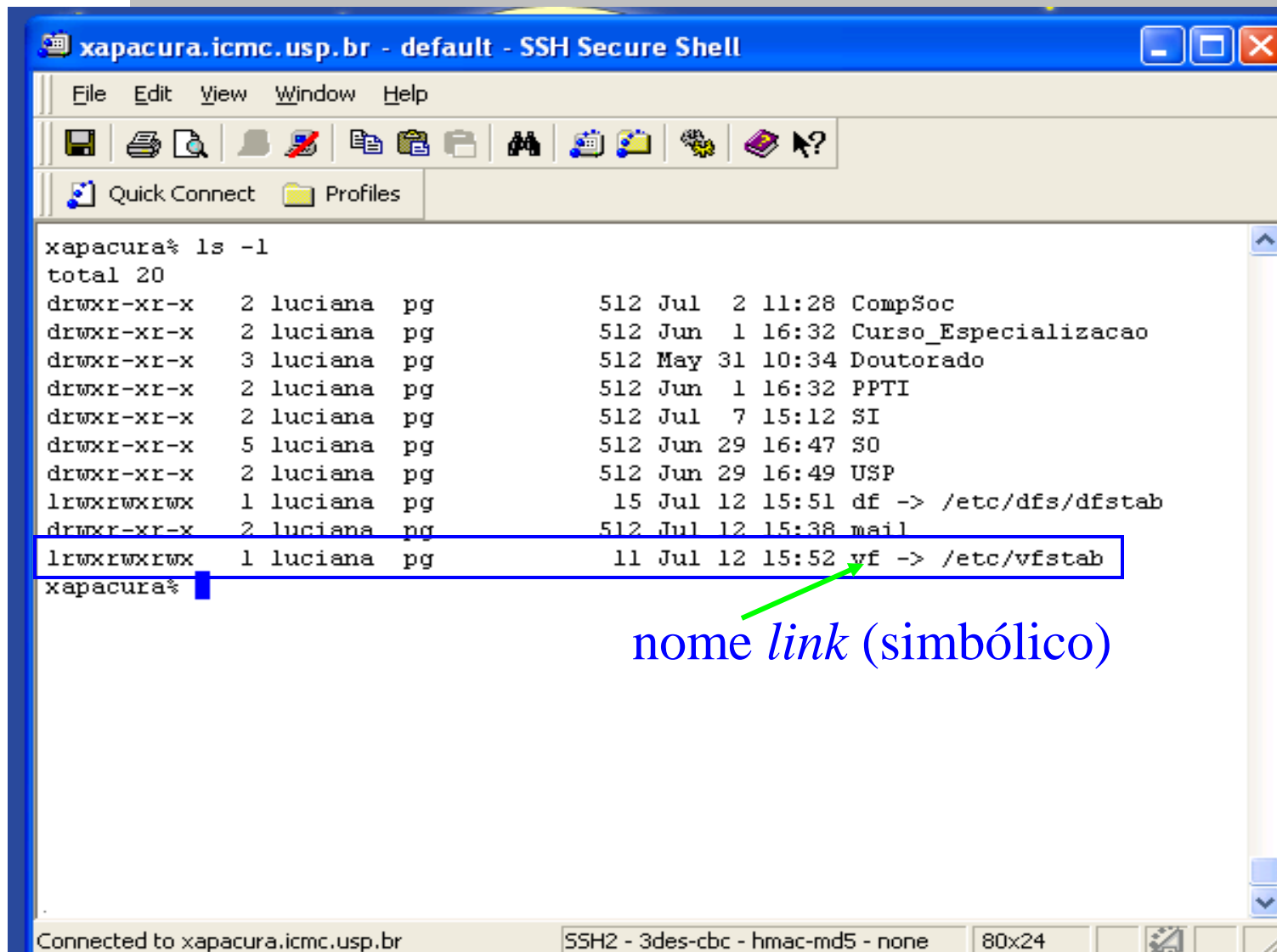


# Implementando o Sistema de Arquivos

## Arquivos Compartilhados

- Segunda Solução: Ligação Simbólica → B se liga ao arquivo de C criando um arquivo do tipo ***link*** e inserindo esse arquivo em seu diretório;
  - Somente o dono do arquivo tem o ponteiro para o *i-node*;
  - O arquivo ***link*** contém apenas o caminho do arquivo ao qual ele está ligado;
    - Assim, remoções não afetam o arquivo;
  - Problema:
    - Sobrecarga;
    - Geralmente um *i-node* extra para cada ligação simbólica;

# Implementando o Sistema de Arquivos Arquivos Compartilhados



```
xapacura% ls -l
total 20
drwxr-xr-x  2 luciana  pg           512 Jul  2 11:28 CompSoc
drwxr-xr-x  2 luciana  pg           512 Jun  1 16:32 Curso_Especializacao
drwxr-xr-x  3 luciana  pg           512 May 31 10:34 Doutorado
drwxr-xr-x  2 luciana  pg           512 Jun  1 16:32 PPTI
drwxr-xr-x  2 luciana  pg           512 Jul  7 15:12 SI
drwxr-xr-x  5 luciana  pg           512 Jun 29 16:47 S0
drwxr-xr-x  2 luciana  pg           512 Jun 29 16:49 USP
lrwxrwxrwx  1 luciana  pg             15 Jul 12 15:51 df -> /etc/dfs/dfstab
drwxr-xr-x  2 luciana  pg           512 Jul 12 15:38 mail
lrwxrwxrwx  1 luciana  pg             11 Jul 12 15:52 vf -> /etc/vfstab
xapacura%
```

nome *link* (simbólico)

Connected to xapacura.icmc.usp.br | SSH2 - 3des-cbc - hmac-md5 - none | 80x24

## Implementando o Sistema de Arquivos

### Gerenciamento de espaço em disco

- Duas estratégias são possíveis para armazenar um arquivo de  $n$  bytes:
  - São alocados ao arquivo  $n$  bytes consecutivos do espaço disponível em disco;
  - Arquivo é espalhado por um número de blocos não necessariamente contínuos → blocos com tamanho fixo;
    - A maioria dos sistemas de arquivos utilizam essa estratégia;

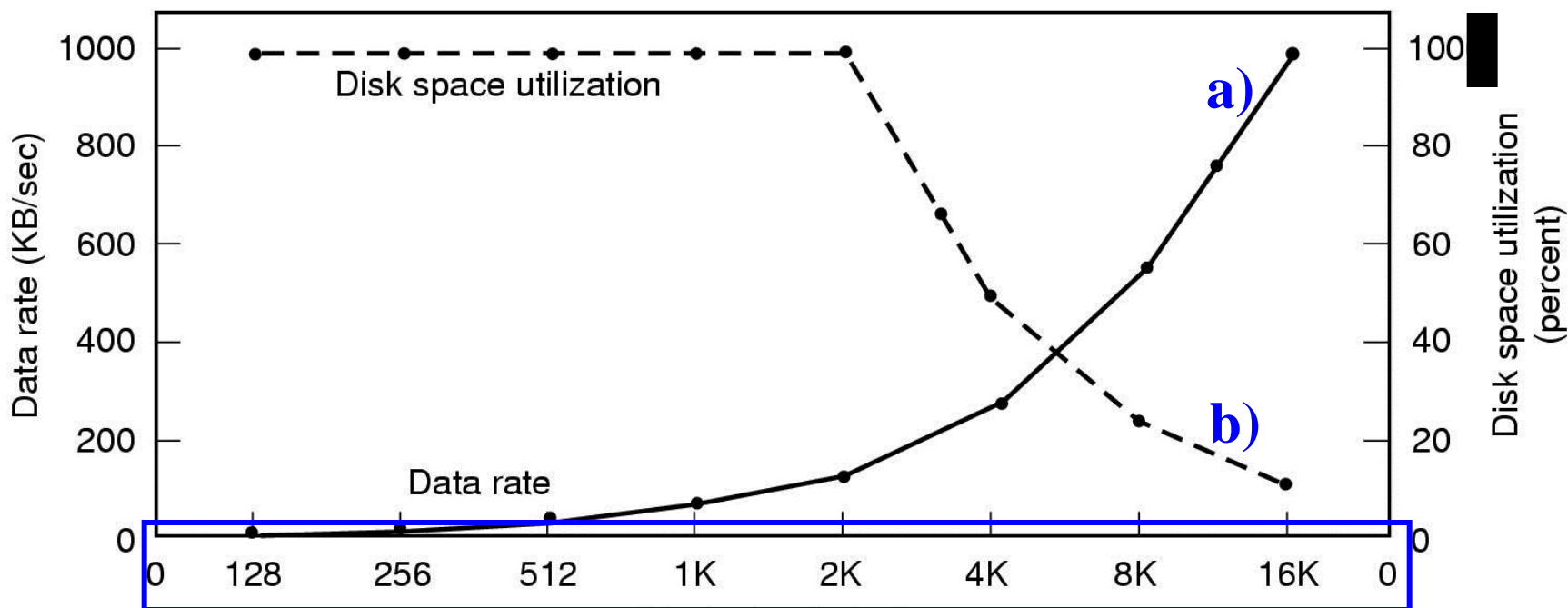
## Implementando o Sistema de Arquivos

### Gerenciamento de espaço em disco

- Questão importante: Qual é o tamanho ideal para um bloco?
  - Se for muito grande, ocorre desperdício de espaço;
  - Se for muito pequeno, um arquivo irá ocupar muitos blocos, tornando o acesso/busca lento;
- Assim, o tamanho do bloco tem uma grande influência na eficiência de utilização do espaço em disco e no acesso ao disco (desempenho);

# Implementando o Sistema de Arquivos

## Gerenciamento de espaço em disco



a) Taxa de Dados (curva contínua) X Tamanho do Bloco

b) Utilização do disco (curva tracejada) X Tamanho do Bloco

# Implementando o Sistema de Arquivos

## Gerenciamento de espaço em disco

- Conflito entre performance (desempenho) e utilização do disco → blocos pequenos contribuem para um baixo desempenho, mas são bons para o gerenciamento de espaço em disco;
- UNIX → 1kb;
- MS-DOS → 512 bytes a 32 kb (potências de 2);
  - Tamanho do bloco depende do tamanho do disco;
  - Máximo número de blocos =  $2^{16}$ ;
- WinNT → 2Kb; WINXP → 4kb;
- Linux → 1kb, 2kb , 4kb;

# Implementando o Sistema de Arquivos

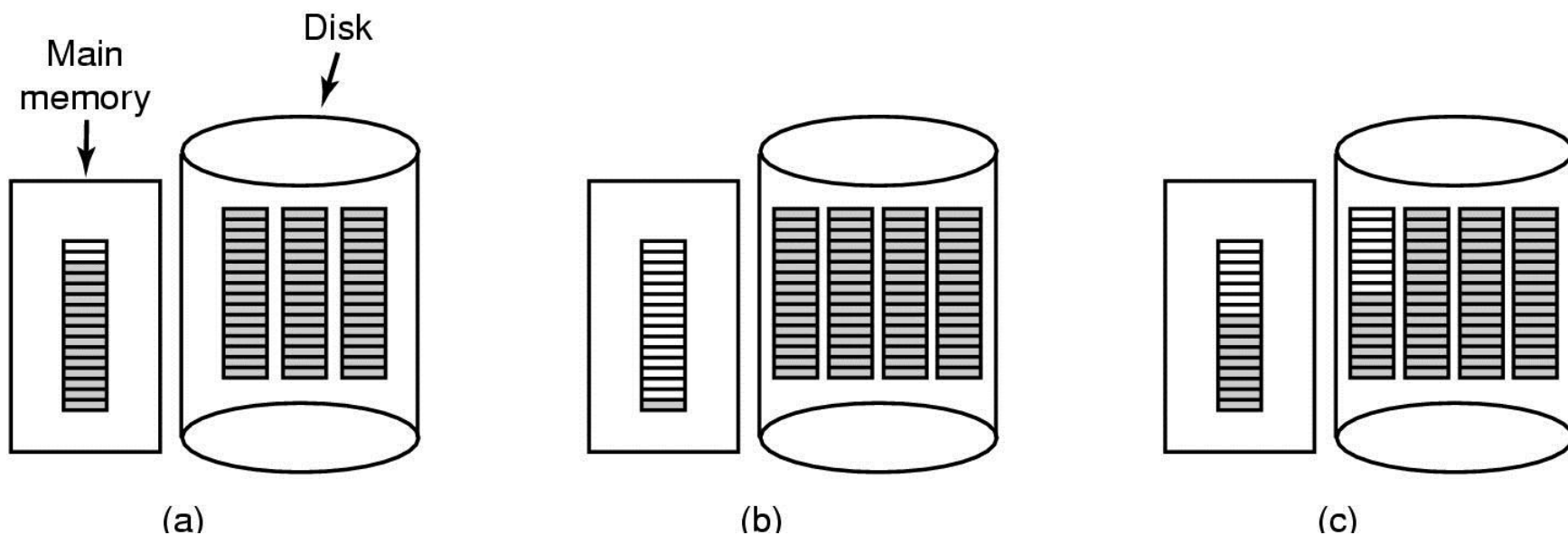
## Gerenciamento de espaço em disco

- Controle de blocos livres → dois métodos:
  - Lista ligada de blocos livres: 32 bits para cada bloco; mantida no disco;
    - Somente um bloco de ponteiros é mantido na memória principal → quando bloco está completo, esse bloco é escrito no disco;
    - Vantagens:
      - Requer menos espaço se existem poucos blocos livres (disco quase cheio);
      - Armazena apenas um bloco de ponteiros na memória;
    - Desvantagens:
      - Requer mais espaço se existem muitos blocos livres (disco quase vazio);
      - Dificulta alocação contínua;
      - Não ordenação;

# Implementando o Sistema de Arquivos

## Gerenciamento de espaço em disco

**Situação:** três blocos são liberados



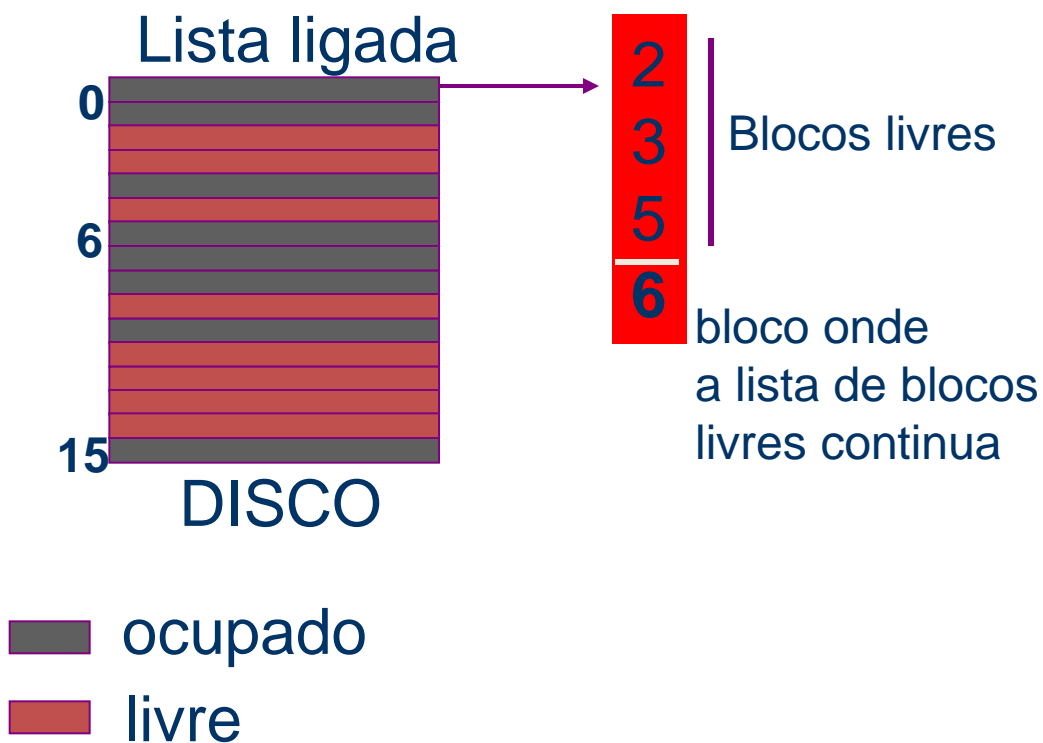
## Blocos de ponteiros

- Entradas sombreadas representam ponteiros para blocos livres no disco



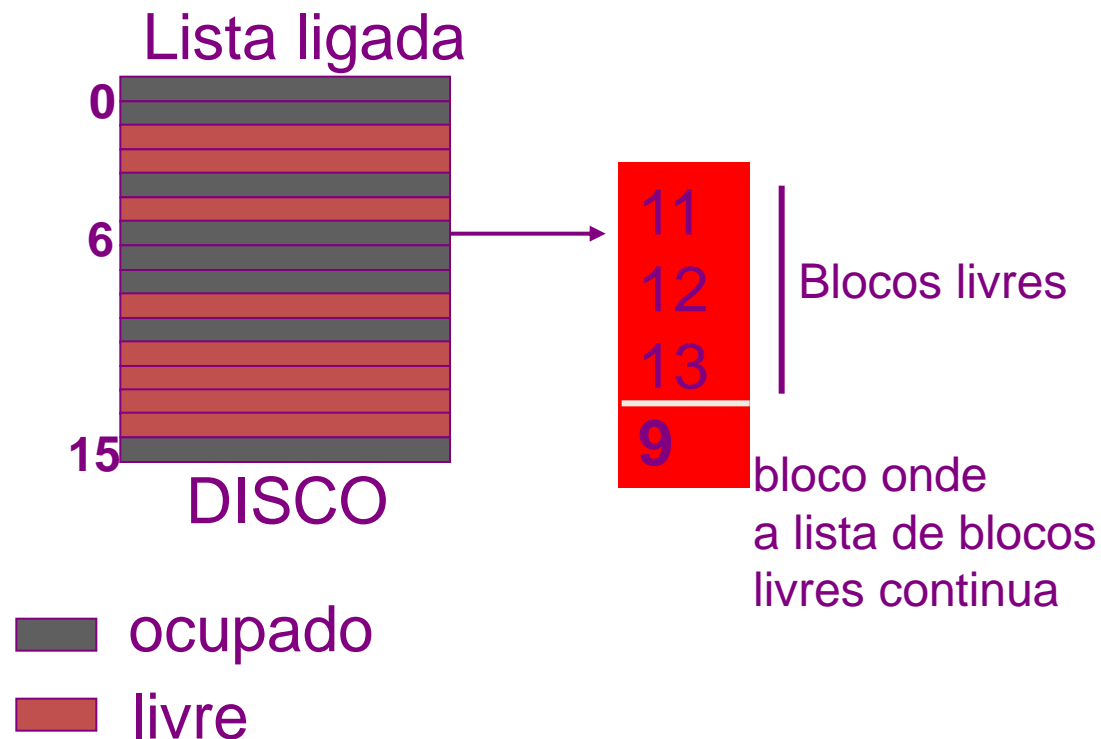
# Implementando o Sistema de Arquivos

## Gerenciamento de espaço em disco



# Implementando o Sistema de Arquivos

## Gerenciamento de espaço em disco





# Implementando o Sistema de Arquivos

## Gerenciamento de espaço em disco

- Mapa de bits (*bitmap*): depende do tamanho do disco:
  - Um disco com  $n$  blocos, possui um mapa de bits com  $n$  *bits*, sendo um bit para cada bloco;
  - Mapa é mantido na memória principal;
  - Vantagens:
    - Requer menos espaço;
    - Facilita alocação contínua;
  - Desvantagens:
    - Torna-se lento quando o disco está quase cheio;

**Blocos livres = 1**  
**Blocos ocupados = 0**  
**ou vice-versa;**

10011011
01101101
10101101
01101101
11101110
 
11011111

## Implementando o Sistema de Arquivos Gerenciamento de espaço em disco

- Controle de cotas do disco: feito para que um usuário não ocupe muito espaço do disco;
  - Idéia → administrador do sistema atribui para cada usuário uma cota máxima de espaço;
- Na memória principal:
  - Tabela de arquivos abertos com ponteiro para uma tabela que mantém registro de todas as cotas do usuário;

# Implementando o Sistema de Arquivos

## Gerenciamento de espaço em disco

Tabela de arquivos abertos

Atributos	
Endereços	
Usuário	
Ponteiro para cota	
~~~~~	

Tabela de cotas

Limite flexível de bloco
Limite estrito de bloco
Número corrente de blocos
Espaço de segurança dos blocos
Limite flexível de arquivos
Limite estrito de arquivos
Número corrente de arquivos
Espaço de segurança dos blocos

# Implementando o Sistema de Arquivos

- Algumas características importantes:
  - Confiabilidade:
    - *Backups*;
    - Consistência;
  - Desempenho:
    - *Caching*;

# Implementando o Sistema de Arquivos

## Confiabilidade

- Danos causados ao sistema de arquivos podem ser desastrosos;
- Restaurar informações pode, e geralmente é, ser custoso, difícil e, em muitos casos, impossível;
- Sistemas de arquivos são projetados para proteger as informações de danos lógicos e não físicos;



# Implementando o Sistema de Arquivos

## Confiabilidade

- *Backups*
  - Cópia de um arquivo ou conjunto de arquivos mantidos por questão de segurança;
    - Mídia mais utilizada → fitas magnéticas;
  - Por que fazer *backups*?
    - Recuperar de desastres: problemas físicos com disco, desastres naturais;
    - Recuperar de “acidentes” do usuários que “acidentalmente” apagam seus arquivos;
      - Lixeira (diretório especial – *recycle bin*): arquivos não são realmente removidos;

# Implementando o Sistema de Arquivos

## Confiabilidade

- *Backups* podem ser feitos automaticamente (horários/dias programados) ou manualmente;
- *Backups* demoram e ocupam muito espaço → eficiência e conveniência;
- Questões:
  - O que deve ser copiado → nem tudo no sistema de arquivos precisa ser copiado;
    - Diretórios específicos;

# Implementando o Sistema de Arquivos

## Confiabilidade

- Não fazer *backups* de arquivos que não são modificados há um certo tempo;
  - *Backups* semanais/mensais seguidos de *backups* diários das modificações → *incremental dumps*;
    - Vantagem: minimizar tempo;
    - Desvantagem: recuperação é mais complicada;
- Comprimir os dados antes de copiá-los;
- Dificuldade em realizar *backup* com o sistema de arquivos ativo:
  - Deixar o sistema *off-line*: nem sempre possível;
  - Algoritmos para realizar *snapshots* no sistema: salvam estado atual do sistema e suas estruturas de dados;
- As fitas de *backup* devem ser deixadas em locais seguros;

# Implementando o Sistema de Arquivos

## Confiabilidade

- Estratégias utilizadas para *backup*:
  - Física: cópia se inicia no bloco 0 e pára somente no último bloco, independentemente se existem ou não arquivos nesses blocos;
    - Desvantagens:
      - Copiar blocos ainda não utilizados não é interessante;
      - Possibilidade de copiar blocos com defeitos;
      - Difícil restaurar diretórios/arquivos específicos;
      - Incapacidade de saltar diretórios específicos;
      - Não permite cópias incrementais;
    - Vantagens:
      - Simples e rápida;

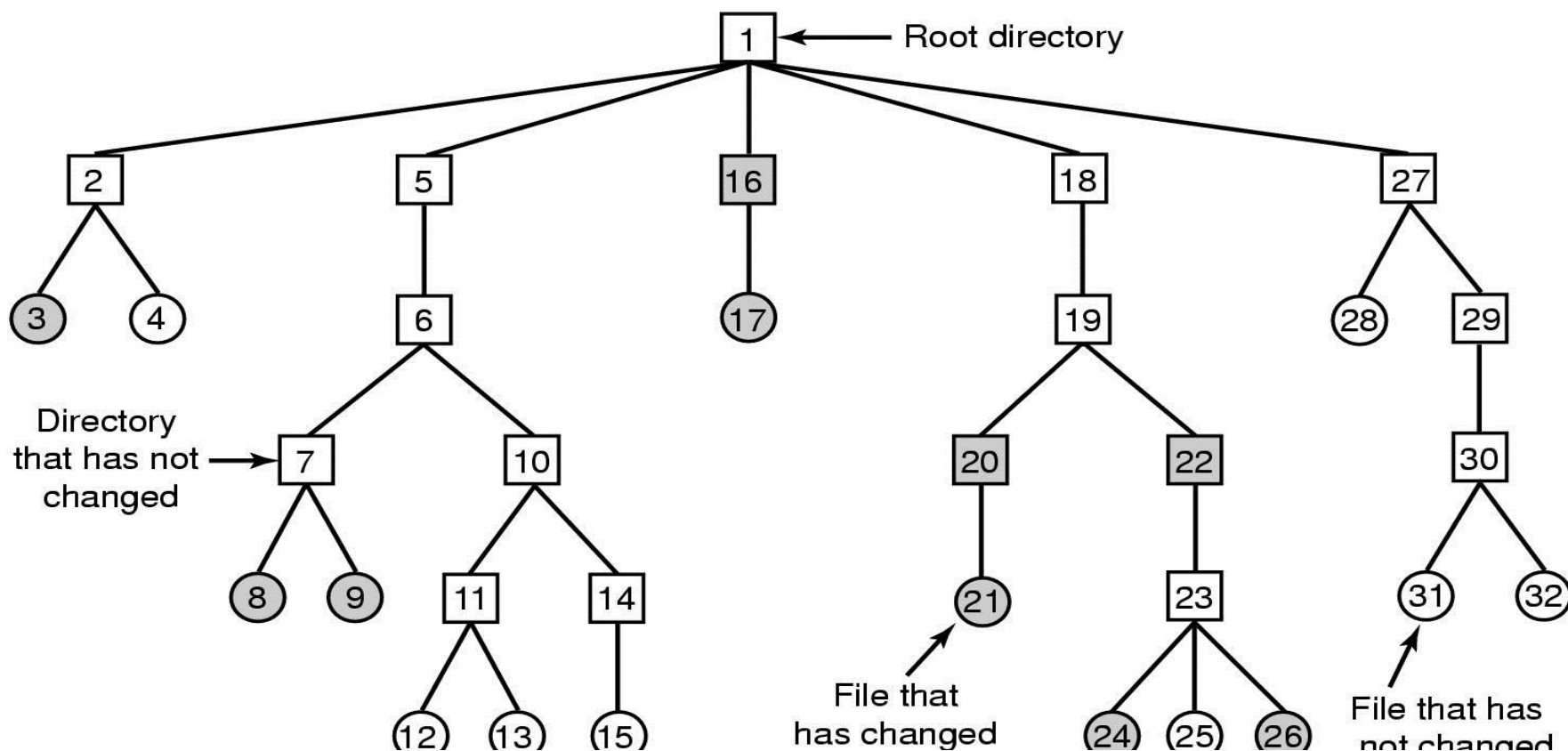
# Implementando o Sistema de Arquivos

## Confiabilidade

- Lógica: inicia-se em um diretório específico e recursivamente copia seus arquivos e diretórios; A idéia é copiar somente os arquivos (diretórios) que foram modificados;
  - Vantagem:
    - Facilita a recuperação de arquivos ou diretórios específicos;
  - Forma mais comum de *backup*;
  - Cuidados:
    - *Links* devem ser restaurados somente uma vez;
    - Como a lista de blocos livres não é copiada, ela deve ser reconstruída depois da restauração;

# Implementando o Sistema de Arquivos Confiabilidade

## Algoritmo para Cópia Lógica



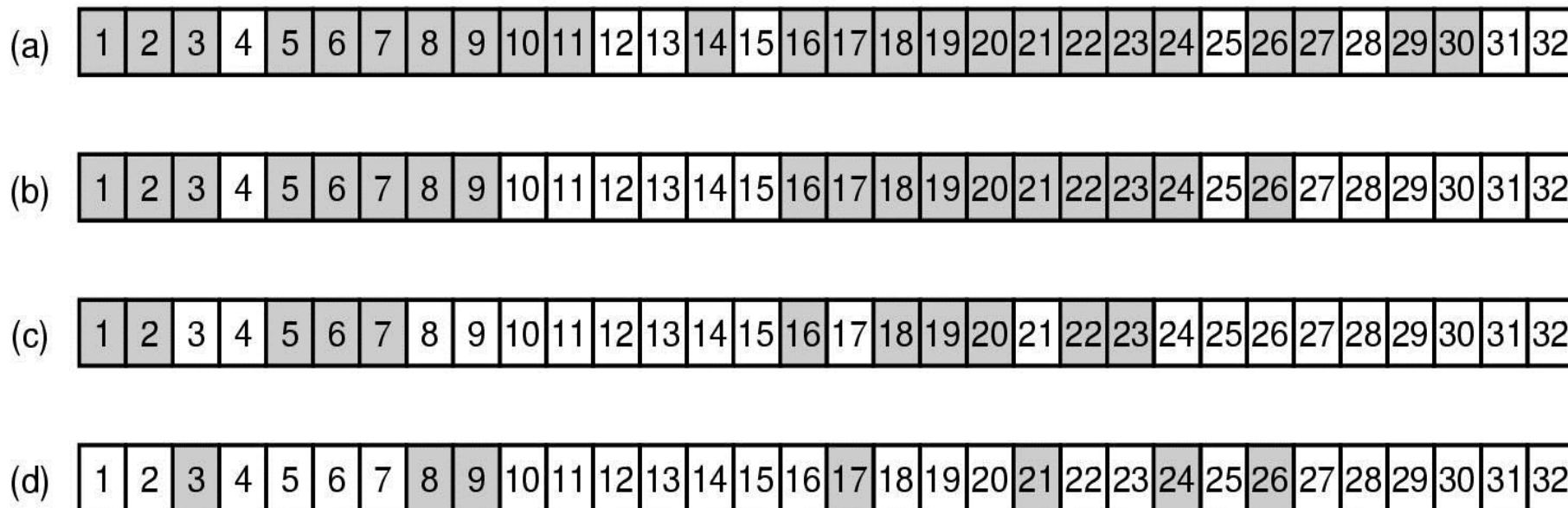
# Implementando o Sistema de Arquivos

## Confiabilidade

- Algoritmo para cópia lógica:
  - Fase 1 (a): marcar todos os arquivos modificados e os diretórios modificados ou não;
    - Diretórios marcados: 1, 2, 5, 6, 7, 10, 11, 14, 16, 18, 19, 20, 22, 23, 27, 29, 30;
    - Arquivos marcados: 3, 8, 9, 17, 21, 24, 26;
  - Fase 2 (b): desmarcar diretórios que não tenham arquivos/sub-diretórios abaixo deles modificados;
    - Diretórios desmarcados: 10, 11, 14, 27, 29, 30;
  - Fase 3 (c): varrer os *i-nodes* (em ordem numérica) e copiar diretórios marcados;
    - Diretórios copiados: 1, 2, 5, 6, 7, 16, 18, 19, 20, 22, 23;
  - Fase 4 (d): arquivos marcados são copiados.
    - Arquivos copiados: 3, 8, 9, 21, 24, 26;

# Implementando o Sistema de Arquivos Confiabilidade

## Algoritmo para Cópia Lógica



**Mapa de bits indexado pelo número do *i-node***



# Implementando o Sistema de Arquivos

## Confiabilidade

- Consistência → dados no sistema de arquivos devem estar consistentes;
- Crítico: blocos de i-nodes, blocos de diretórios ou blocos contendo a lista de blocos livres/mapa de bits de blocos livres;
- Diferentes sistemas possuem diferentes programas utilitários para lidar com inconsistências:
  - UNIX: *fsck*;
  - Windows: *scandisk*;

# Implementando o Sistema de Arquivos

## Confiabilidade

- FSCK (*file system checker*):
  - Blocos: o programa constrói duas tabelas; cada qual com um contador (inicialmente com valor 0) para cada bloco;
    - os contadores da primeira tabela registram quantas vezes cada bloco está presente em um arquivo;
    - os contadores da segunda tabela registram quantas vezes cada bloco está presente na lista de blocos livres;
    - Lendo o *i-node*, o programa constrói uma lista com todos os blocos utilizados por um arquivo (incrementa contadores da 1ª tabela);
    - Lendo a lista de bloco livres ou *bitmap*, o programa verifica quais blocos não estão sendo utilizados (incrementa contadores da 2ª tabela);
    - Assim, se o sistema de arquivos estiver consistente, cada bloco terá apenas um bit 1 em uma das tabelas (a);

# Implementando o Sistema de Arquivos Confiabilidade

a)

0															15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

Blocos em uso

Blocos livres

Se problemas acontecerem, podemos ter as seguintes situações:

b)

0		2													15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1

Blocos em uso

Blocos livres

Bloco 2 perdido (*missing block*)

Solução: colocá-lo na lista de livres

c)

0		2													15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

Blocos em uso

Blocos livres

# Implementando o Sistema de Arquivos Confiabilidade

a)

0															15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

Blocos em uso

Blocos livres

Se problemas acontecerem, podemos ter as seguintes situações:

d)

0				4											15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1

Blocos em uso

Blocos livres

Bloco 4 duplicado na lista de livres

Solução: reconstruir a lista

Essa situação só ocorre se existir uma lista encadeada de blocos livres ao invés de um mapa de bits.

# Implementando o Sistema de Arquivos Confiabilidade

a)

0															15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

Blocos em uso

Blocos livres

Se problemas acontecerem, podemos ter as seguintes situações:

e)

0					5										15
1	1	0	1	0	<b>2</b>	1	1	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

Blocos em uso

Blocos livres

Bloco **5** duplicado na lista de “em uso” (dois arquivos)

Problemas:

- Se um arquivo for removido, o bloco vai estar nas duas listas;
- Se ambos forem removidos, o bloco vai estar na lista de livres duas vezes;

Solução: alocar um bloco livre e copiar para esse bloco o conteúdo do bloco 5, e avisar o administrador/usuário do problema;

# Implementando o Sistema de Arquivos

## Confiabilidade

- FSCK (*file system checker*)
  - Arquivos: Além do controle de blocos, o verificador também armazena em um contador o uso de um arquivo  
→ tabela de contadores por arquivos:
    - *Links* simbólicos não entram na contagem;
    - *Links* estritos (*hard link*) entram na contagem (arquivo pode aparecer em dois ou mais diretórios);
    - Cria uma lista indexada pelo número do i-node indicando em quantos diretórios cada arquivo aparece (contador de arquivos);
    - Compara esses valores com a contagem de ligações existentes (começa em 1 quando arquivo é criado);

# Implementando o Sistema de Arquivos Confiabilidade

- FSCK (*file system checker*)
  - Arquivos:
    - Se o sistema estiver consistente, os contadores devem ser iguais;
    - Caso esses contadores não sejam iguais, sempre o que está registrado no diretório é o que vale, ou seja, em quantos diretórios o arquivo aparece;

# Implementando o Sistema de Arquivos

## Desempenho

- Acessar memória RAM é mais rápido do que acessar disco;
  - Movimentação do disco;
  - Movimentação do braço;
  - Técnicas para otimizar acesso:
    - *Caching*;
    - Leitura prévia de blocos;
    - Reduzir a quantidade de movimentos do braço do disco;



# Implementando o Sistema de Arquivos

## Desempenho

- *Caching*: técnica conhecida como *cache* de bloco ou *cache* de *buffer*;
  - *Cache*: um conjunto de blocos que pertencem logicamente ao disco mas são colocados na memória para melhorar o desempenho do sistema (reduzir acesso em disco);
- Quando um bloco é requisitado, o sistema verifica se o bloco está na *cache*; se sim, o acesso é realizado sem necessidade de ir até o disco; caso contrário, o bloco é copiado do disco para a *cache*;

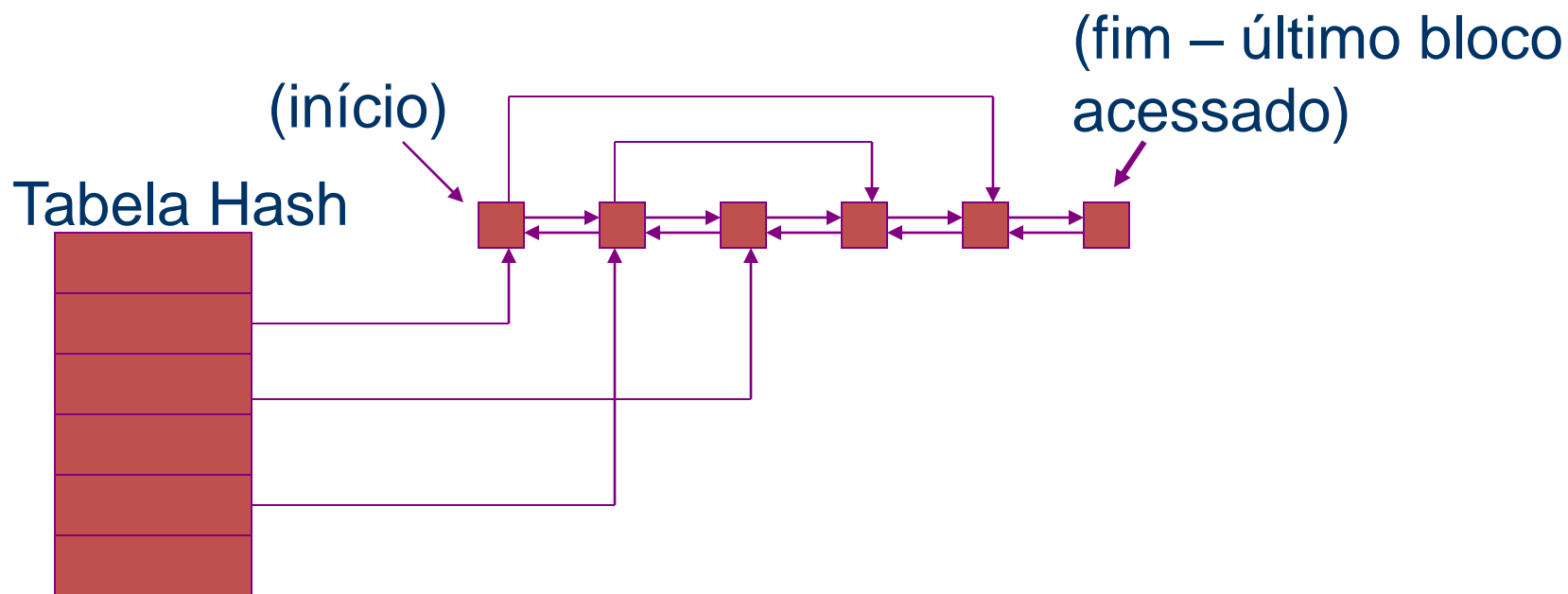
# Implementando o Sistema de Arquivos

## Desempenho

- Para se carregar um novo bloco na *cache*, poderá ser necessário remover um dos blocos que estão armazenados, reescrevendo-o no disco caso tenha sido modificado → troca de blocos (semelhante à troca de páginas);
  - Algoritmos utilizados na paginação podem ser utilizados nesse caso;
- O algoritmo mais utilizado é o LRU (*least recently used*) com listas duplamente encadeadas;
- Para determinar se um bloco está na *cache* pode-se usar uma Tabela *Hash*;

# Implementando o Sistema de Arquivos Desempenho

Todos os blocos com o mesmo valor *hash* são encadeados na lista



# Implementando o Sistema de Arquivos

## Desempenho

- Importante: Não convém manter blocos de dados na *cache* por um longo tempo antes de escrevê-los de volta ao disco;
- Alguns sistemas realizam cópias dos blocos modificados para o disco de tempos em tempos;
  - UNIX/Windows: uma chamada ***update*** realiza a cada 30 segundos uma chamada ***sync***;
  - MS-DOS: copia o bloco para o disco assim que esse tenha sido modificado → *cache* de escrita direta (*write-through*)
    - Estratégia usada para discos flexíveis;

# Implementando o Sistema de Arquivos Desempenho

- Leitura prévia dos blocos: blocos são colocados no *cache* antes de serem requisitados;
  - Só funciona quando os arquivos estão sendo lidos de forma sequencial;

# Implementando o Sistema de Arquivos Desempenho

- Reduzir o movimento do braço do disco: colocando os blocos que são mais prováveis de serem acessados próximos uns dos outros em sequência (mesmo cilindro do disco)
  - O gerenciamento do disco é feito por grupos de blocos consecutivos e não somente por blocos;

# Implementando o Sistema de Arquivos

## Desempenho

- Para sistemas que utilizam os *i-nodes*, são necessários dois acessos: um para o bloco e outro para o *i-node*;
- Três estratégias podem ser utilizadas para armazenamento dos *i-nodes*:
  - A) Os *i-nodes* são colocados no início do disco, assim a distância média entre o *i-node* e seus blocos é de metade do número de cilindros do disco;
  - B) Dividir o disco em grupos de cilindros, nos quais cada cilindro tem seus próprios *i-nodes*, blocos e lista de blocos livres (*bitmap*);
  - C) Os *i-nodes* são colocados no meio do disco;

# Implementando o Sistema de Arquivos Desempenho

