Sistema de Arquivos

Prof. Marcos Ribeiro Quinet de Andrade
Universidade Federal Fluminense – UFF
Instituto de Ciência e Tecnologia – ICT

Sistema de Arquivos

- É a 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

- Arquivos são manipulados pelo Sistema Operacional
- Tarefas:
  - Estrutura de arquivos;
  - Nomes;
  - Acessos (uso);
  - Proteção;
  - Implementação;

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

Sistema de Arquivos – Nomes de arquivos

- Quando arquivos são criados, nomes são atribuídos a esses arquivos, os quais passam a ser referenciados por meio desses 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: ?, *, /, \, “, |, <, ı, :

Sistema de Arquivos – Nomes de arquivos

- Alguns Sistemas Operacionais são sensíveis a letras maiúsculas e minúsculas (*case sensitive*) e outros não;
  - UNIX (e consequentemente, Linux) é *case sensitive:*
    - Ex.: exemplo.c é diferente de Exemplo.c;
  - MS-DOS não é *case sensitive* :
    - Ex.: exemplo.c é o mesmo que Exemplo.c;
- Win95/Win98/WinNT/Win2000/WinXP/Vista herdaram características do sistema de arquivos do MS-DOS;
  - No entanto, os sistemas WinNT, Win2000, WinXP, WinVista, Win7, Win8 e Win10 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;
      - Porém, em alguns casos, o compilador pode ser recusar a compilar o arquivo.

Sistema de Arquivos – Nomes de arquivos

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

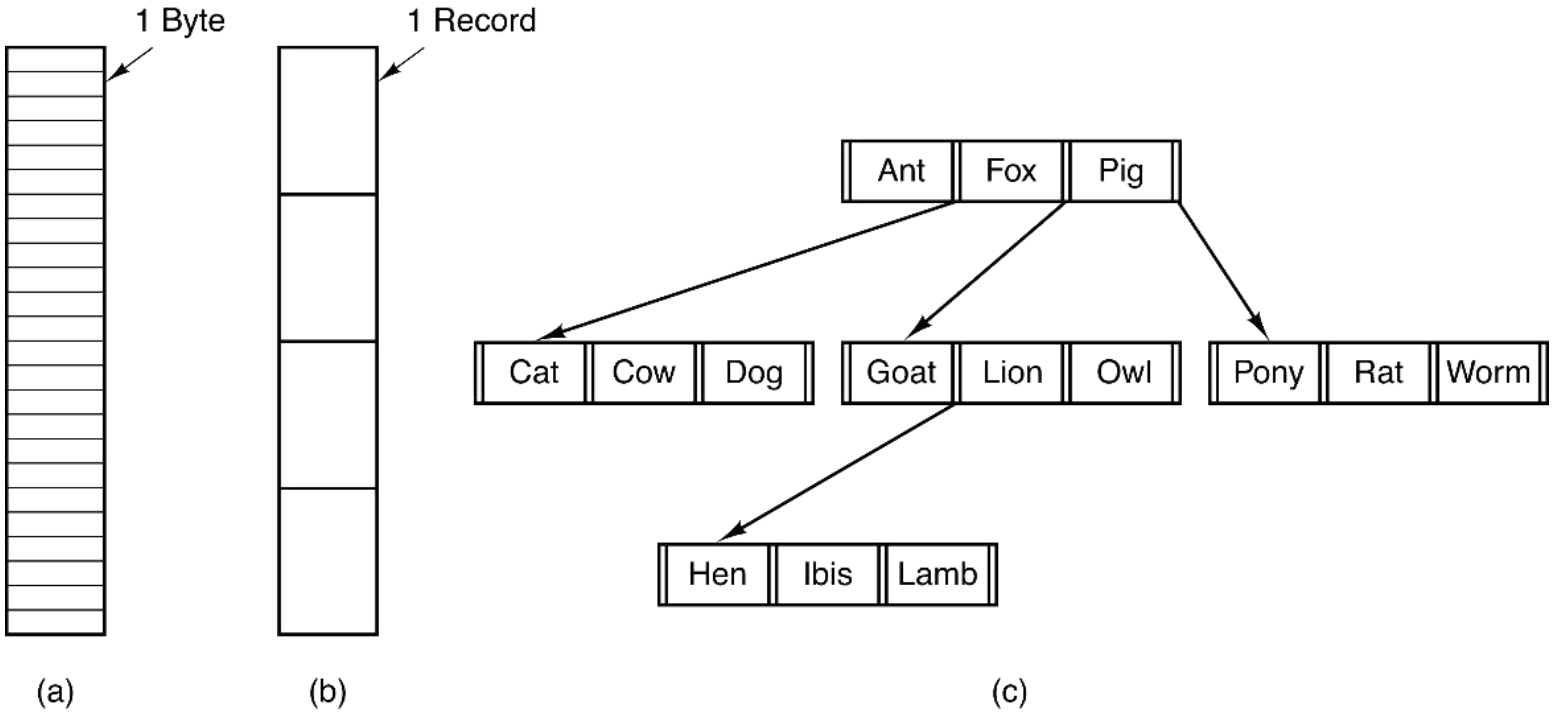
Sistema de Arquivos – Estrutura de arquivos

- Arquivos podem ser estruturados de diferentes maneiras:
  - a) Sequência 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) Sequência 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) ou sistemas de registros de 132 caracteres (impressoras de linha);
  - Nenhum sistema atual utiliza esse esquema;
- c) Árvores de registros (tamanho variado), cada qual com um campo **chave** em uma posição fixa:
  - SO decide onde colocar os arquivos;
  - Ainda usado em computadores de grande porte, em alguns sistemas de processamento de dados comerciais.

Sistema de Arquivos – Estrutura de arquivos



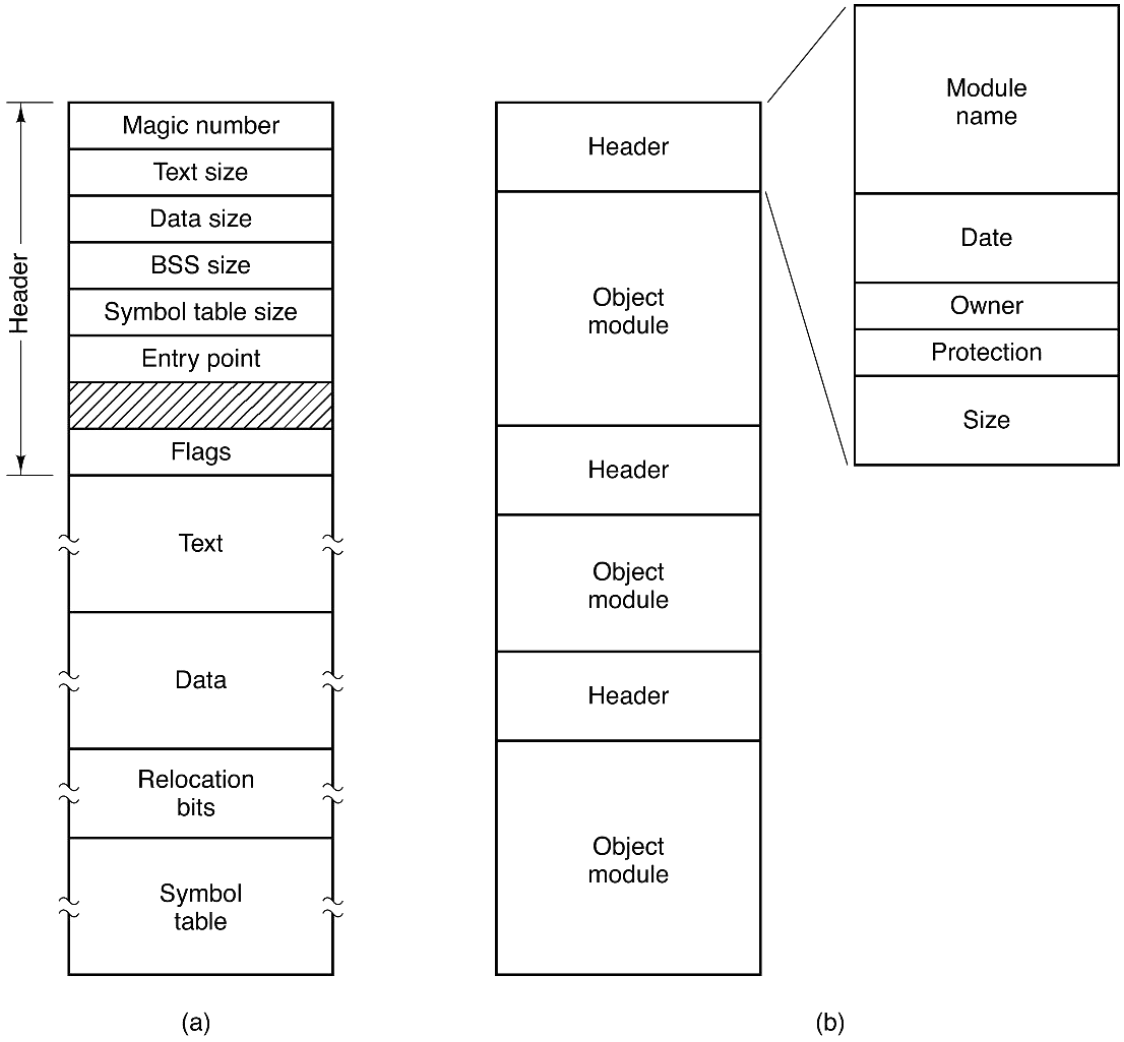
Sistema de Arquivos – Tipos de arquivos

- **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:
  - (a) Binário:
    - Todo arquivo não ASCII;
    - Possuem uma estrutura interna conhecida pelos aplicativos que os usam;
    - Ex.: programa executável → número mágico;
  - (b) 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 com extensão txt;

Sistema de Arquivos – Tipos de arquivos



Sistema de Arquivos – Acessos em arquivos

- SO's mais antigos ofereciam apenas **acesso sequencial** no disco → leitura em ordem byte a byte (registro a registro);
- SO's mais modernos fazem **acesso aleatório** aos arquivos, cujos bytes ou registros podem ser acessados em qualquer ordem;
  - Acesso feito por **chave**;
  - Métodos que podem ser usados para especificar onde iniciar leitura:
    - Operação Read → posição do arquivo em que se inicia a leitura;
    - Operação Seek → a operação seek estabelece a posição atual; depois de um seek, o arquivo pode ser lido sequencialmente a partir de sua posição atual. Usado no Linux e Windows

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, mas com algumas categorias, como os de proteção, são comuns a todos;

Sistema de Arquivos – Atributos de arquivos

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

Sistema de Arquivos – Atributos de arquivos

Atributo	Significado
Flag ASCII/Binary	0 para arquivo ASCII; 1 para arquivo binário
Flag de acesso aleatório	0 para arquivo de acesso seqüencial; 1 para arquivo de acesso randômico
Flag de temporário	0 para normal; 1 para temporário
Flag de impedido	0 para arquivo desimpedido; diferente de 0 para arquivo impedido
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 (*key*)

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 com arquivos

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

Sistema de Arquivos – exemplo de *system call* para cópia de arquivos

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                  /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);       /* ANSI prototype */

#define BUF_SIZE 4096                   /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);             /* syntax error if argc is not 3 */

    /* Open the input file and create the output file */
    in_fd = open(argv[1], O_RDONLY);    /* open the source file */
    if (in_fd < 0) exit(2);             /* if it cannot be opened, exit */
    out_fd = creat(argv[2], OUTPUT_MODE);  /* create the destination file */
    if (out_fd < 0) exit(3);            /* if it cannot be created, exit */

    /* Copy loop */
    while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE);  /* read a block of data */
        if (rd_count <= 0) break;       /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count);  /* write data */
        if (wt_count <= 0) exit(4);     /* wt_count <= 0 is an error */
    }

    /* Close the files */
    close(in_fd);
    close(out_fd);
    if (rd_count == 0)                  /* no error on last read */
        exit(0);
    else
        exit(5);                        /* error on last read */
}
```

Sistema de Arquivos – Arquivos mapeados em memória

- Alguns SO's 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 → S.O. 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; torna-se necessário definir:
  - Organização;
  - Operações;

Sistema de Arquivos – Diretórios

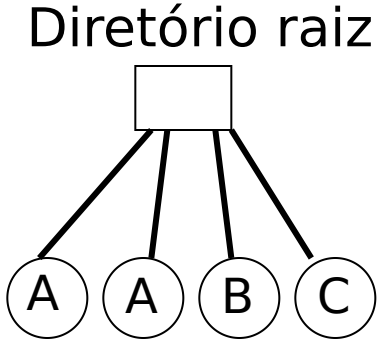
- A organização pode ser feita das seguintes maneiras:
  - Nível único (*Single-level*);
  - Dois níveis (*Two-level*);
  - Hierárquica;

Sistema de Arquivos
Diretórios – Nível único

- Apenas um diretório contém todos os arquivos → **diretório raiz** (*root directory*);
- Computadores antigos utilizavam esse método, pois eram monousuários;
- Exceção: CDC 6600 → supercomputador que utilizava-se desse método, apesar de ser multiusuário;
- Vantagens:
  - Simplicidade (porém, extremamente limitado);
  - Eficiência;
- Ainda pode ser encontrado em alguns sistemas embarcados (por exemplo, antigas câmeras digitais e MP3 *players*)

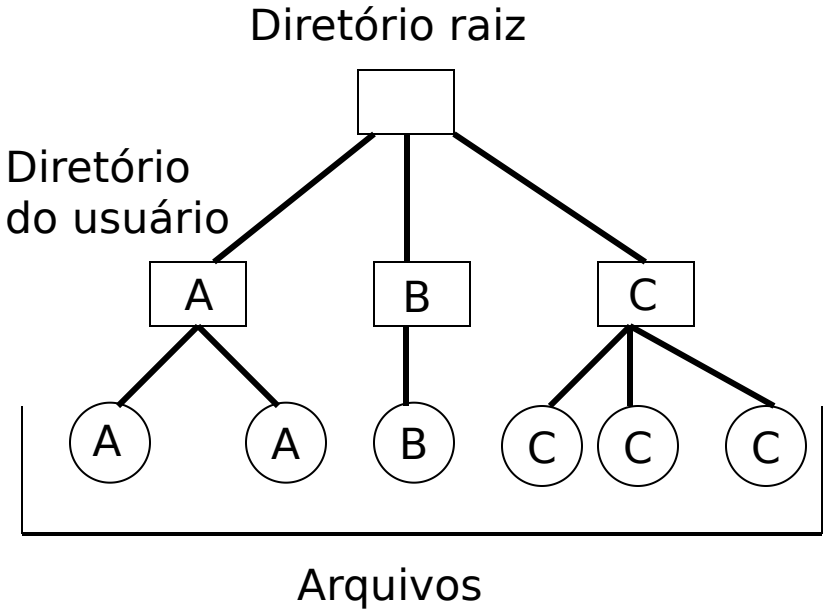
Sistema de Arquivos
Diretórios – Nível único

- Ex.: 4 arquivos, de 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

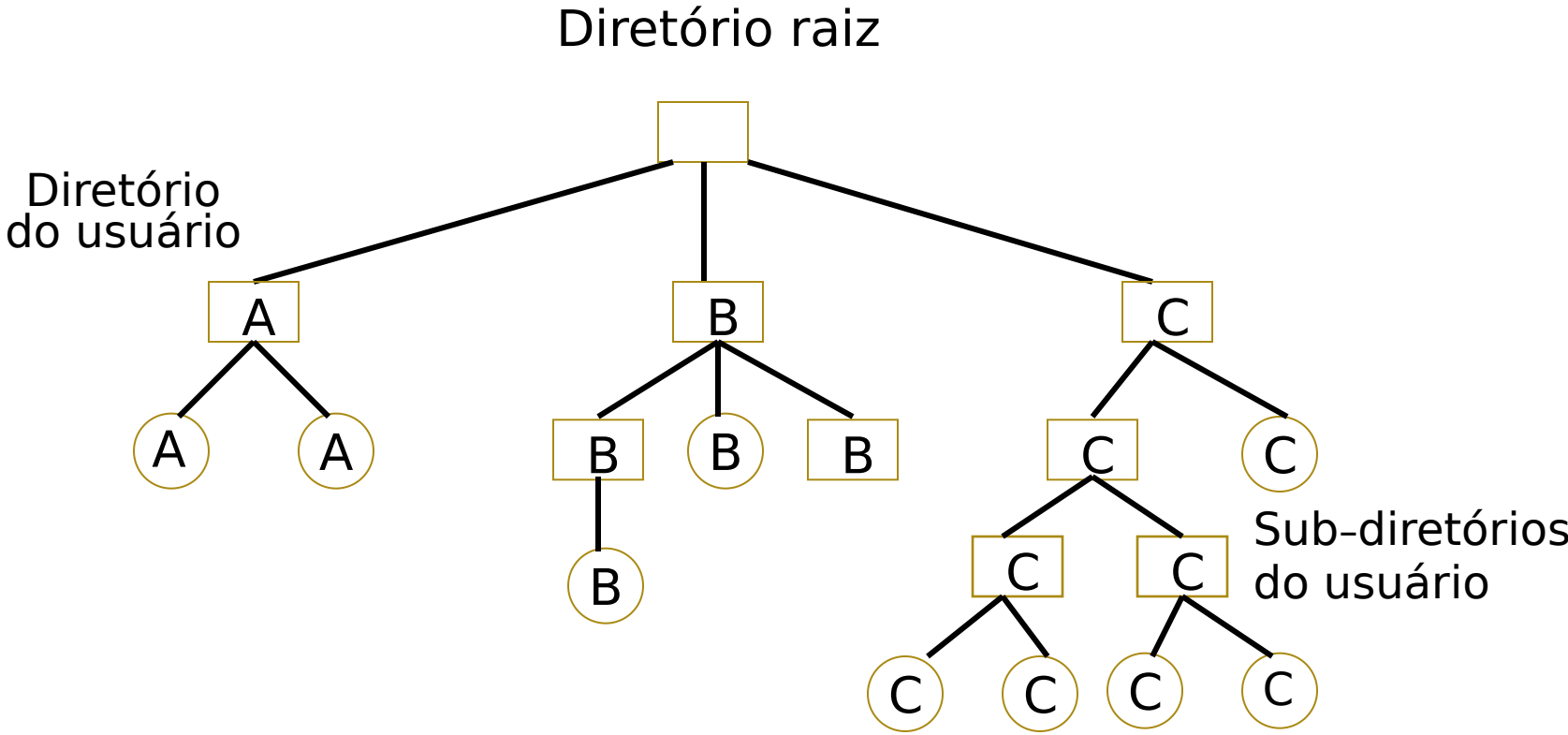
- Cada usuário possui um diretório privado;
- Sem conflitos de nomes de arquivos;
- Procedimento de *login*: identificação;
- Compartilhamento de arquivos → programas executáveis do sistema;
- Desvantagem:
  - Usuário ainda acaba ficando com muitos arquivos em um único lugar;



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;
- Permite grande flexibilidade na organização dos arquivos;

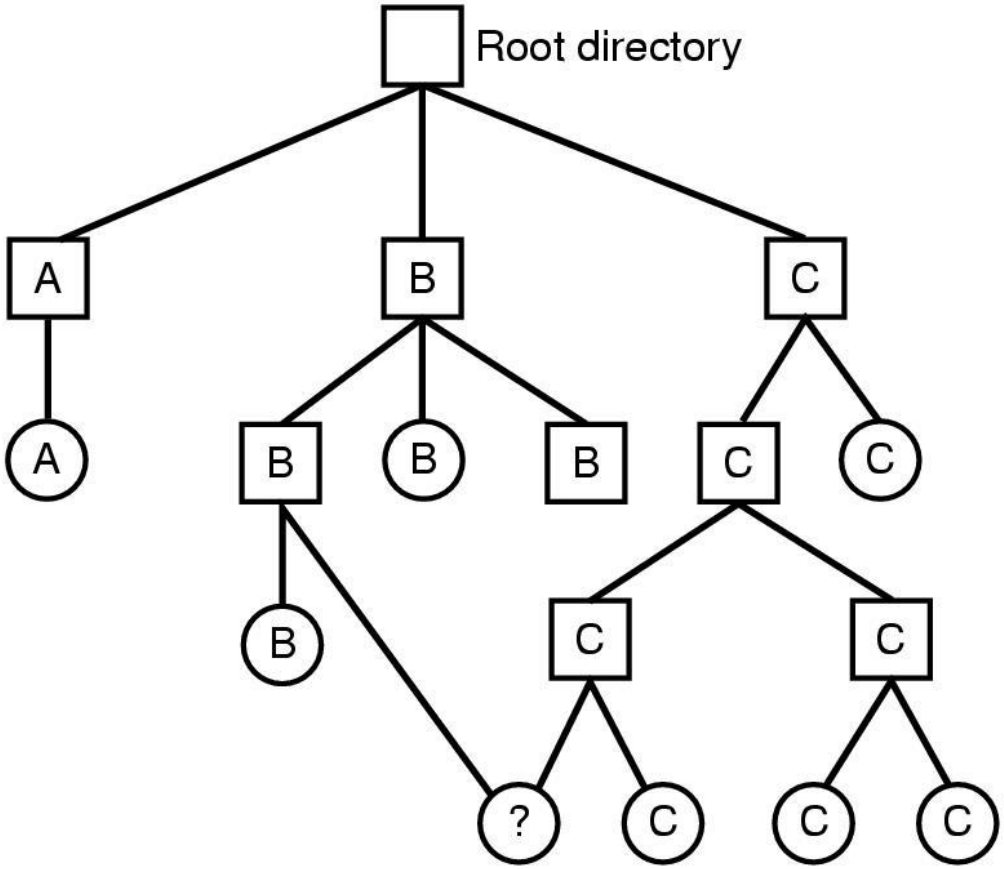
Sistema de Arquivos
Diretórios – Hierárquico



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 passa a ser um grafo acíclico direcionado (*directed acyclic graph – DAG*);
  - ***Links*** são criados;

Implementando o Sistema de Arquivos – arquivos compartilhados

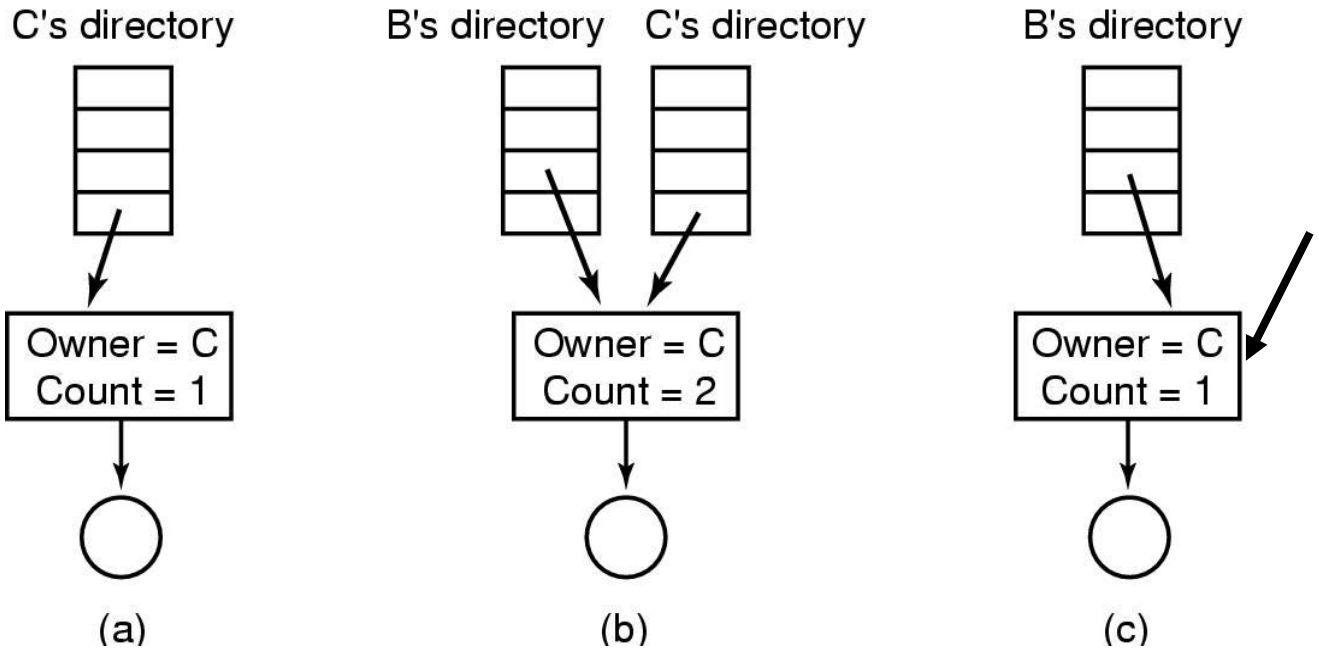


Implementando o Sistema de Arquivos – arquivos compartilhados

- Compartilhar arquivos é sempre conveniente, no entanto, alguns problemas são introduzidos:
  - Se os diretórios tiverem **endereços de disco**, então deverá ser feita um cópia dos endereços no diretório de B (supondo que o arquivo pertença a C);
  - Se B ou C adicionarem 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

Problema de compartilhamento com *hard links*



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

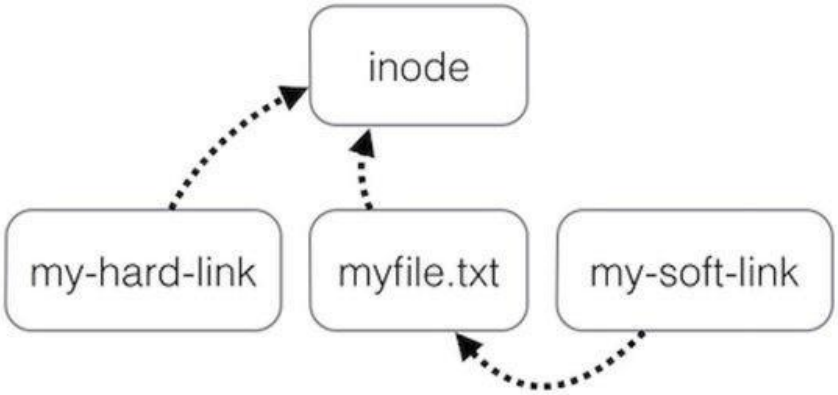
- Soluções:
  - Primeira solução: os **endereços de disco** não são relacionados nos diretórios, mas em uma estrutura de dados (*i-node*) associada ao próprio arquivo. Assim, os diretórios apontam para essa estrutura; (UNIX)
    - Ligação Estrita (*hard link*);
    - Problema com essa solução: o dono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

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 (hard link)*;
  - O arquivo ***link*** contém apenas o caminho do arquivo ao qual ele está ligado (*soft link*);
    - Assim, remoções não afetam o arquivo;
  - Problema:
    - Sobrecarga (análise do caminho);
    - Geralmente um *i-node* extra para cada ligação simbólica, e um bloco extra de disco para armazenar o caminho;

Qual a diferença entre um *hard link* e um *symbolic* (*soft*) *link*?

- Quando um arquivo é criado, é armazenado no disco rígido e associado a um *i-node*, representado por um número;
- o *i-node* armazena o endereço do arquivo, tamanho e outras informações, mas não o nome;
- o nome faz a ligação com um *i-node*, que por sua vez faz a ligação com o arquivo;
- Em resumo: hard link apointa para um i-node, neste caso, a mudança no nome do arquivo não importa; o soft link está associado ao nome, que se modificado, não se relaciona mais ao conteúdo;



Exemplo de *hard* e *soft* links

- Crie dois arquivos, por exemplo linux e windows;
  - touch linux
  - touch windows
- Insira alguns dados neles:
  - echo ‘SO bom‘ | linux
  - echo ‘SO ruim‘ | windows
- Crie dois links, um *hard* e um *soft*:
  - ln linux linux-hard
  - ln -s windows windows-soft
- liste os arquvos (ls -l)
- Mude o nome do arquivo linux e visualize o conteúdo de linux-hard, o que acontece?
- Agora mude o nome do arquivo windows e visualize o conteúdo do windows-soft, qual o resultado?

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: c:|*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
```

Todos os comandos acima realizam a mesma operação

Sistema de Arquivos
Diretórios – Operações

- Exemplos de operações do sistema de arquivos:
  - Create; Delete;
  - Opendir; Closedir;
  - Readdir;
  - Rename;
  - Link (para um arquivo aparecer em mais de um diretório);
  - Unlink;

Implementando o Sistema de arquivos

- Implementação do Sistema de Arquivos: deve estabelecer as formas e operações para as seguintes operações:
  - Como arquivos e diretórios são armazenados;
  - Como o espaço em disco é gerenciado;
  - Como tornar o sistema eficiente e confiável;

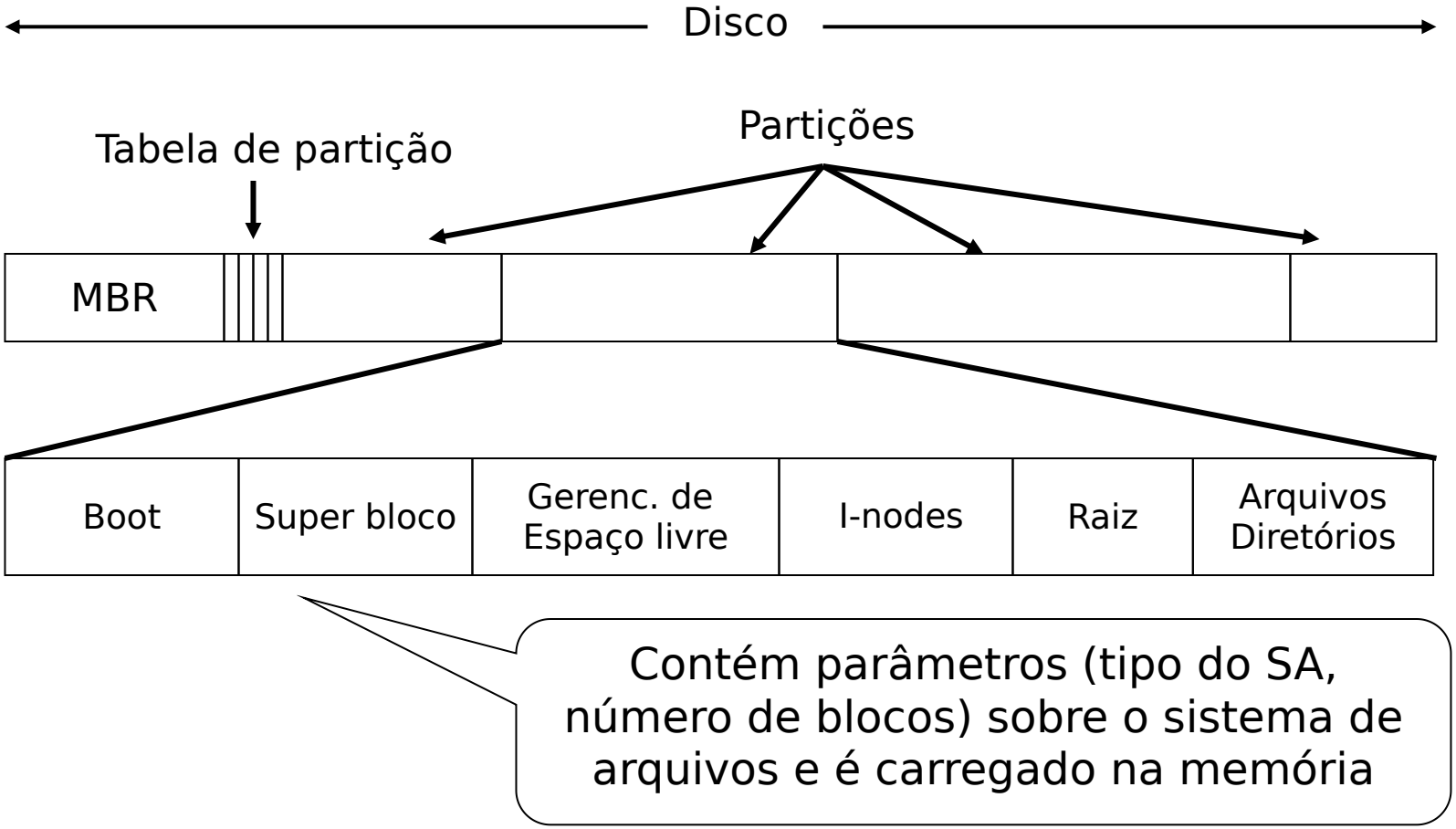
Implementando o Sistema de arquivos – *Layout*

- 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 a 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;

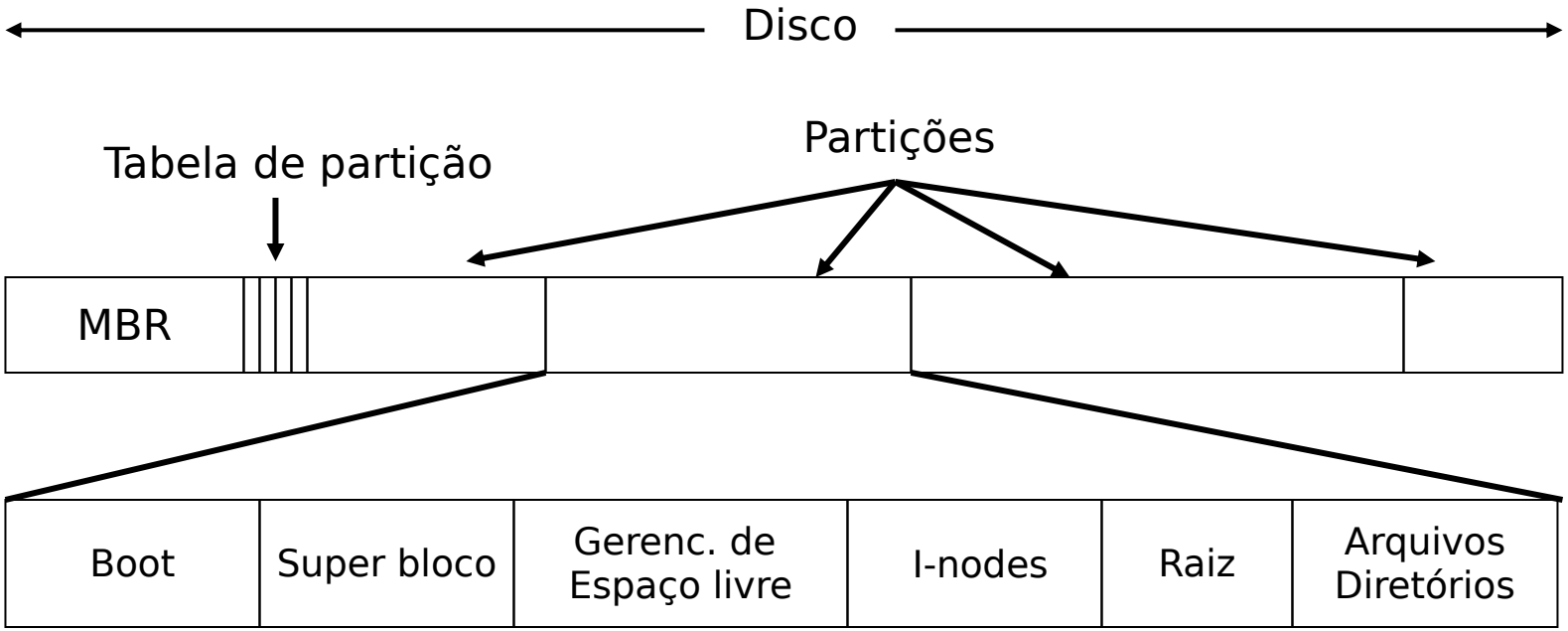
Implementando o Sistema de arquivos – *Layout*

- Tarefas básicas do MBR (podem 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 apresenta a seguir:

Implementando o Sistema de arquivos – *Layout*

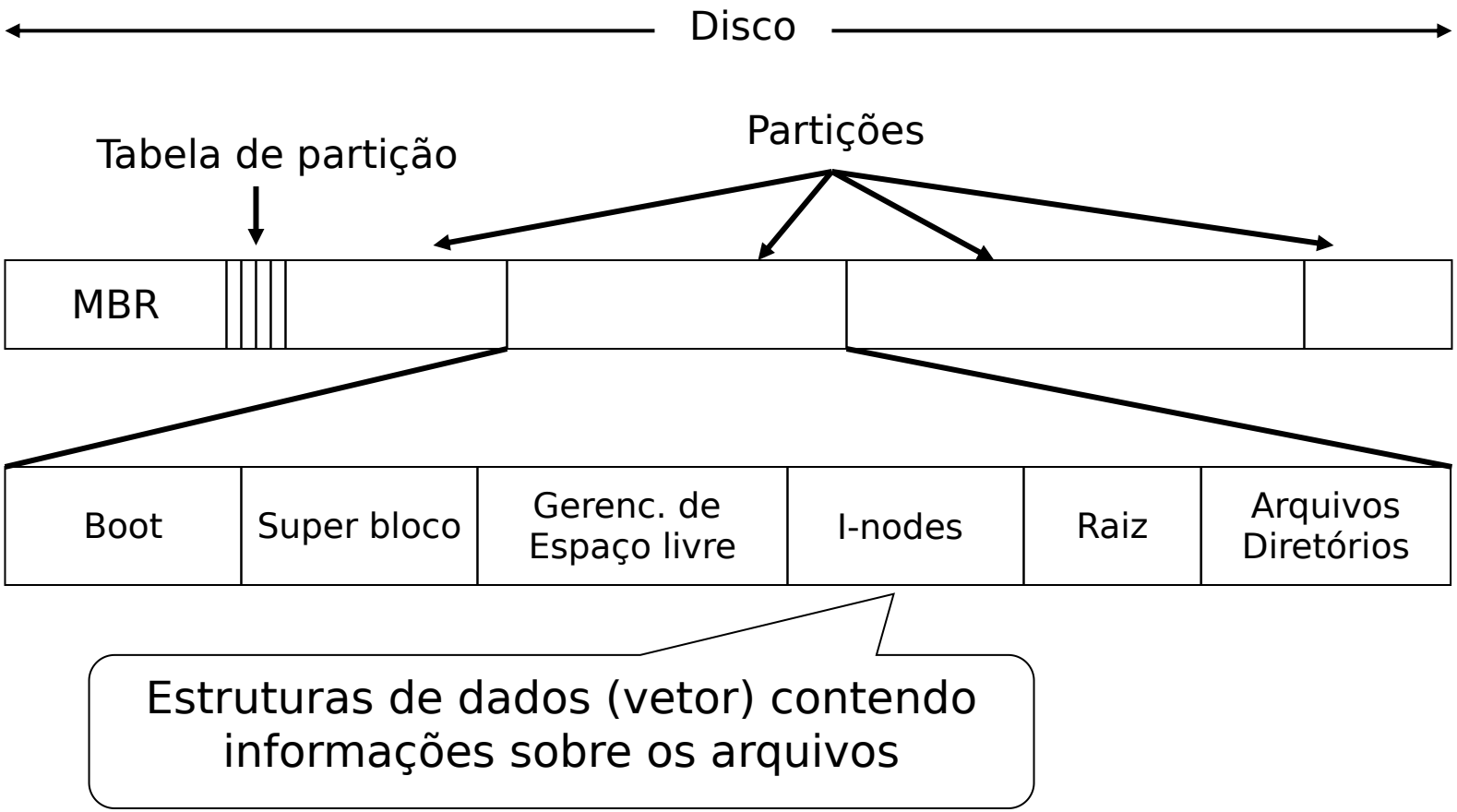


Implementando o Sistema de arquivos – *Layout*

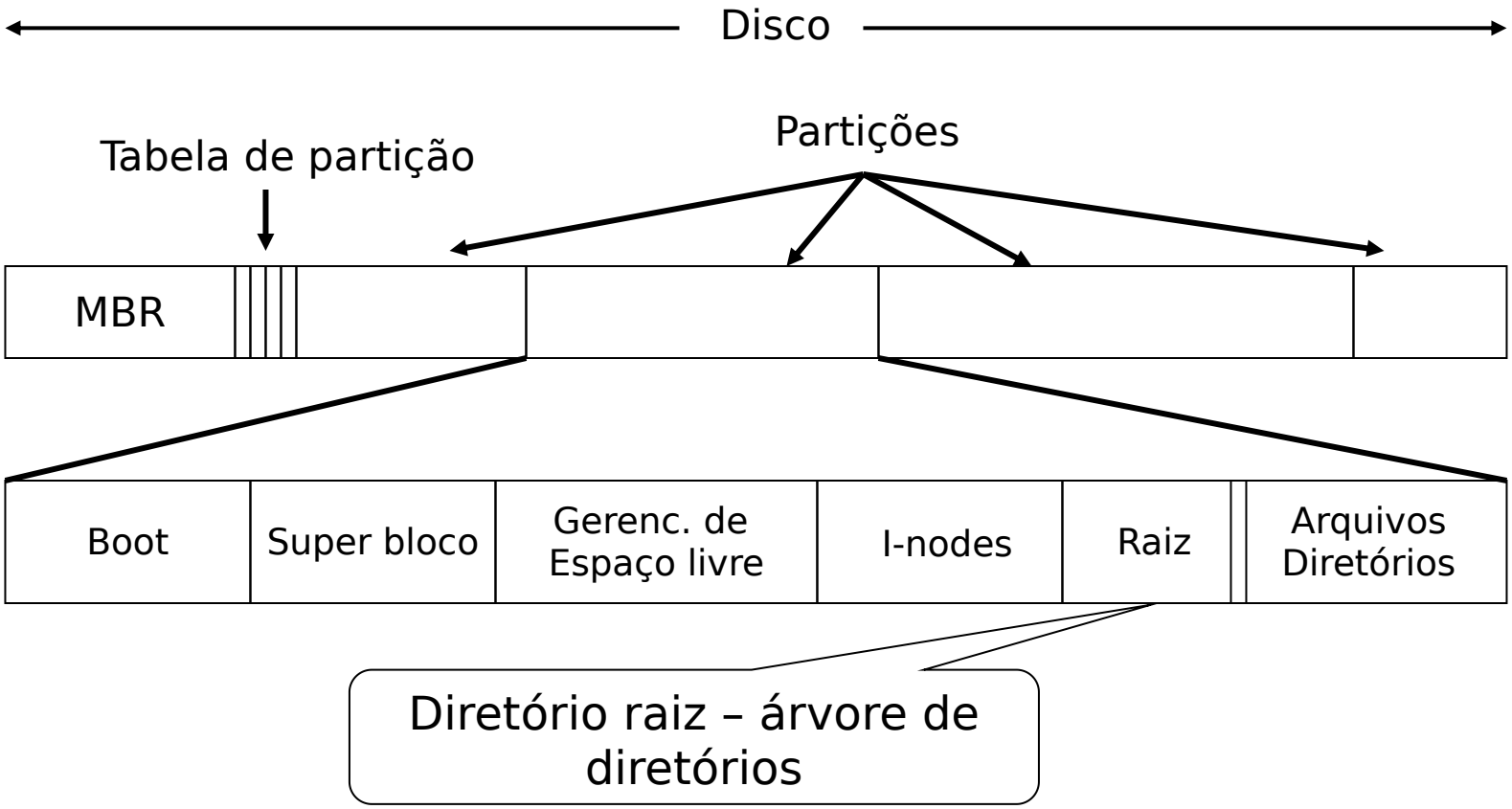


Contém informações sobre os blocos livres do disco (mapa de bits ou lista encadeada)

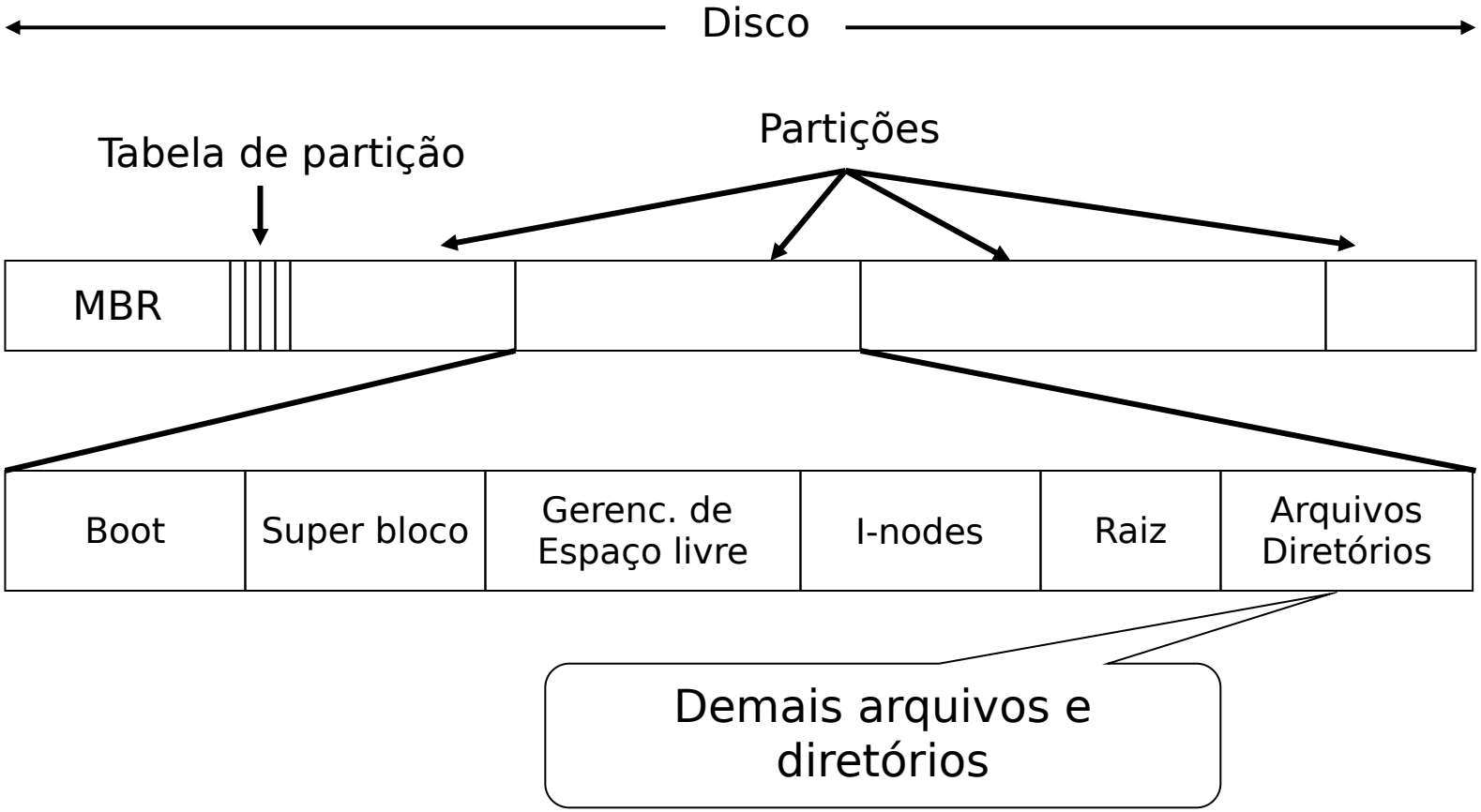
Implementando o Sistema de arquivos – *Layout*



Implementando o Sistema de arquivos – *Layout*



Implementando o Sistema de arquivos – *Layout*



Implementando o Sistema de arquivos – Arquivos

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

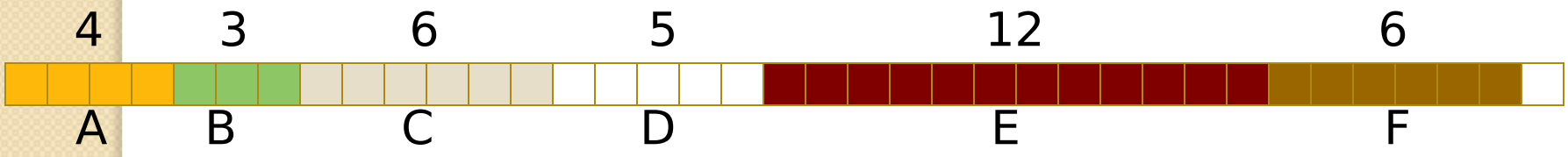
Implementando o Sistema de arquivos – Arquivos

- Alocação contígua:
  - 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;

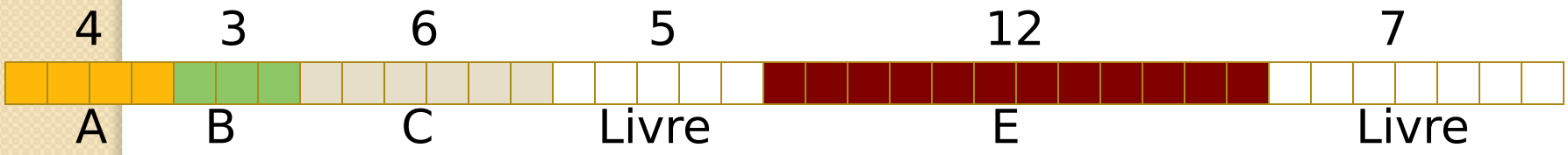
Implementando o Sistema de arquivos – Arquivos

- Alocação contígua:

37 Blocos



Removendo os arquivos D e F...



Implementando o Sistema de arquivos – Arquivos

- Alocação contígua:
  - Vantagens:
    - Simplicidade: somente o endereço do primeiro bloco e número de blocos no arquivo são necessários;
    - Desempenho para o acesso ao arquivo: acesso sequencial;
  - Desvantagens (discos rígidos):
    - Fragmentação externa:
      - Compactação → alto custo;
      - Reuso de espaço → atualização da lista de espaços livres;
        - Conhecimento prévio do tamanho do arquivo para alocar o espaço necessário;
  - Mídias ópticas (quando somente escrita);

Implementando o Sistema de arquivos – Arquivos

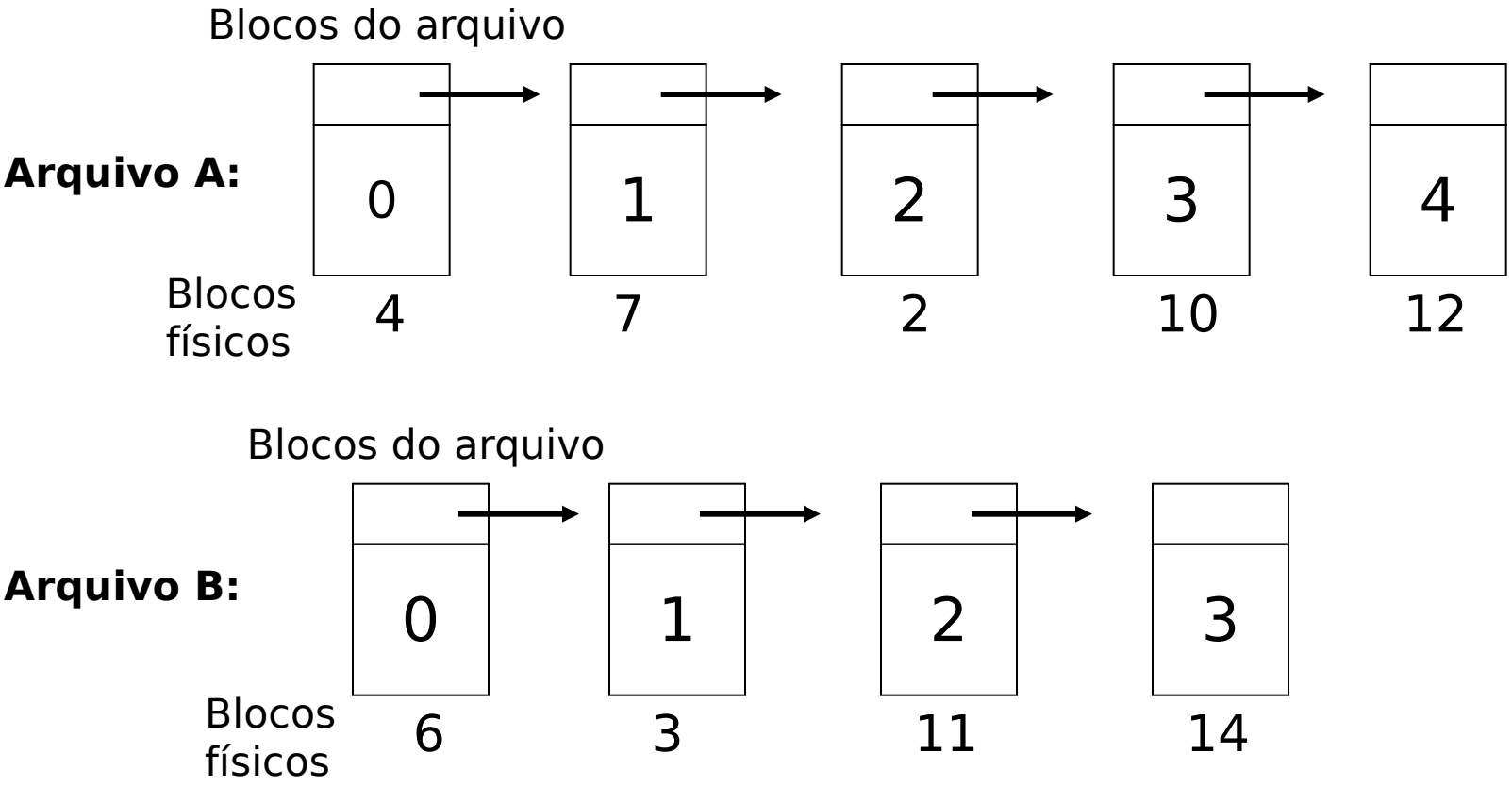
- Alocação com lista encadeada:
  - A primeira palavra de cada bloco é um ponteiro para o bloco seguinte;
  - O restante do bloco é destinado aos dados;
  - Apenas o endereço em disco do primeiro bloco do arquivo é armazenado;
    - Serviço de diretório é responsável por manter esse endereço;

Implementando o Sistema de arquivos – Arquivos

- Alocação com lista encadeada:
  - Desvantagens:
    - Acesso aos arquivos é feito aleatoriamente → 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

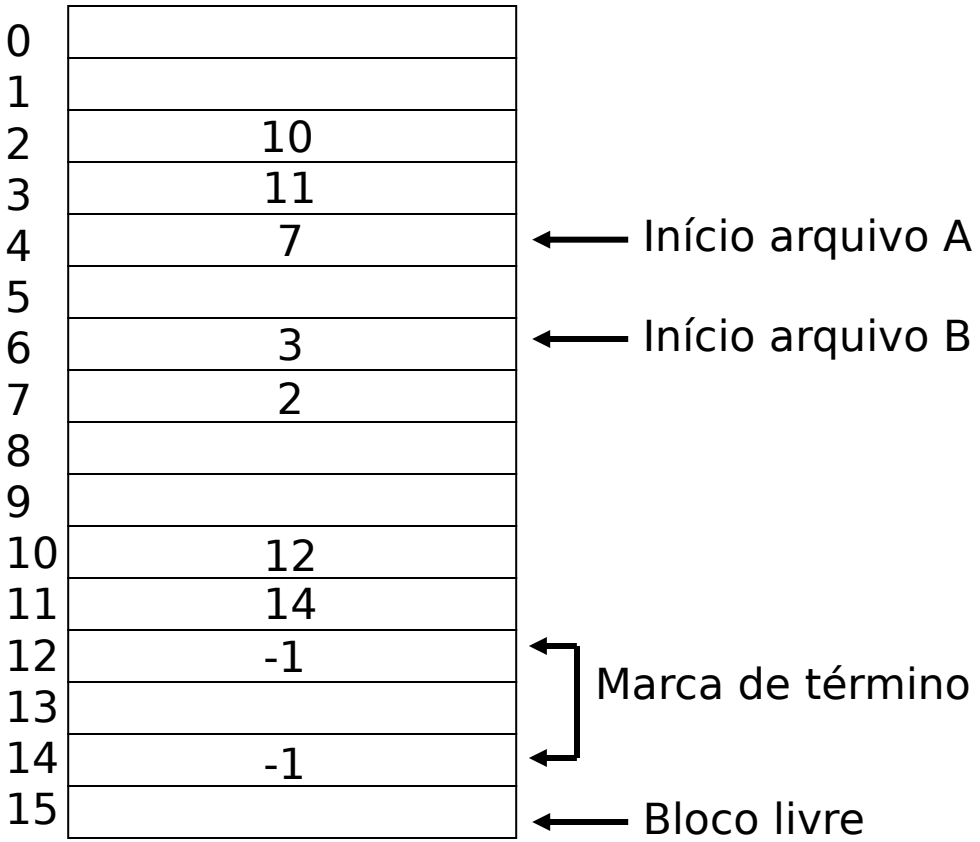
- Alocação com lista encadeada utilizando uma tabela na memória:
  - O ponteiro é colocado em uma tabela na memória ao invés de ser colocado no bloco;
    - FAT (*File Allocation Table*): Tabela de alocação de arquivos;
    - Assim, todo o bloco está disponível para alocação de dados;
  - Serviço de 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 aleatório 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



Arquivo A
4 7 2 10 12

Arquivo B
6 3 11 14

Implementando o Sistema de arquivos – Arquivos

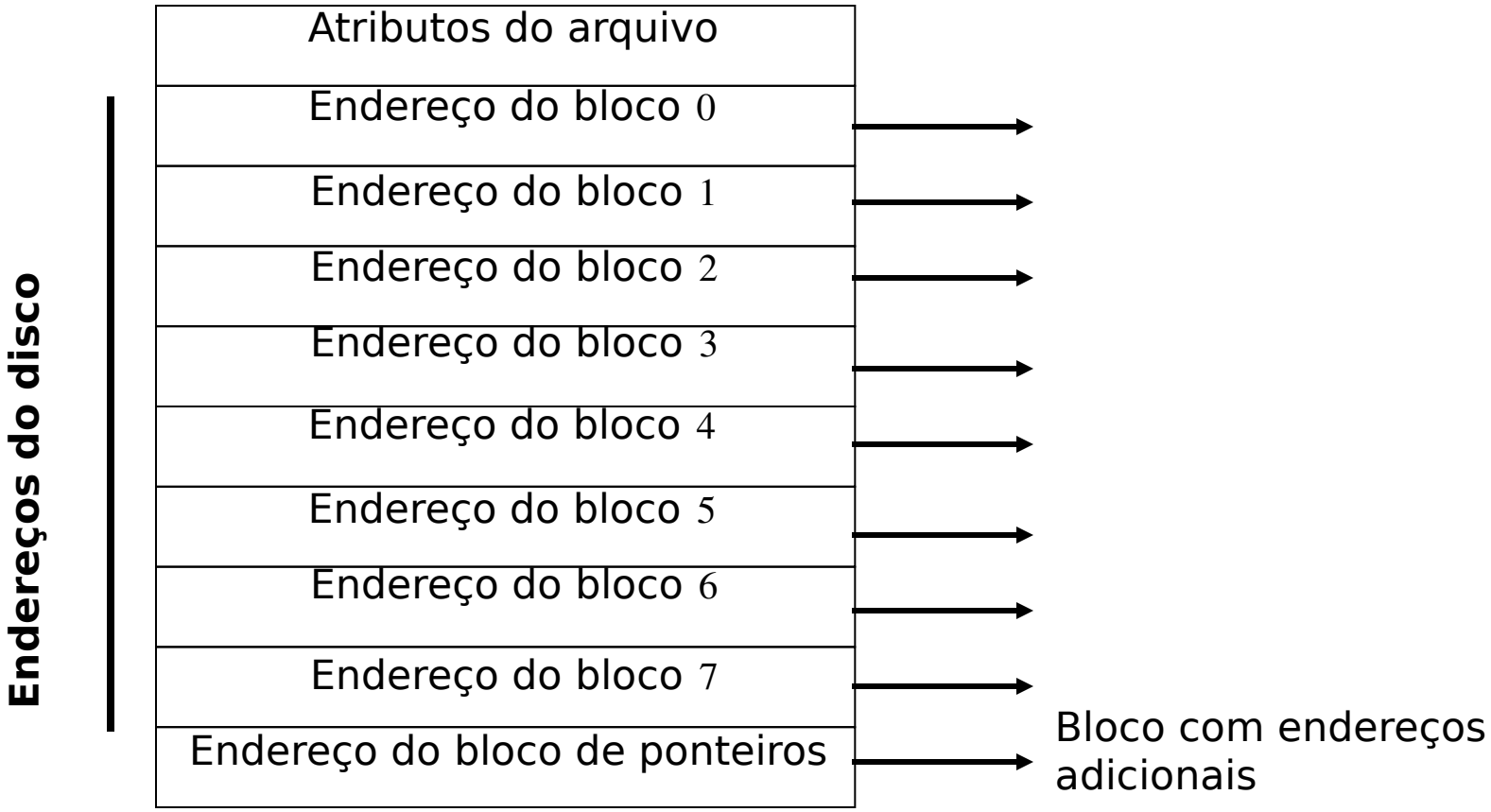
- *I-nodes*:
  - Cada arquivo possui uma estrutura de dados chamada ***i-node*** (*index-node*) que lista os atributos e endereços em disco dos blocos do arquivo;
    - Assim, dado o *i-node* de um arquivo é possível encontrar todos os blocos desse arquivo;
  - Se cada *i-node* ocupa ***n*** bytes e ***k*** arquivos podem estar aberto ao mesmo tempo → o total de memória ocupada é ***k* x *n*** 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 que acontece caso o arquivo aumente de tamanho além do número fixo de endereços de disco comportados pelo *i-node*?
    - Solução: reservar o último endereço para outros endereços de blocos;

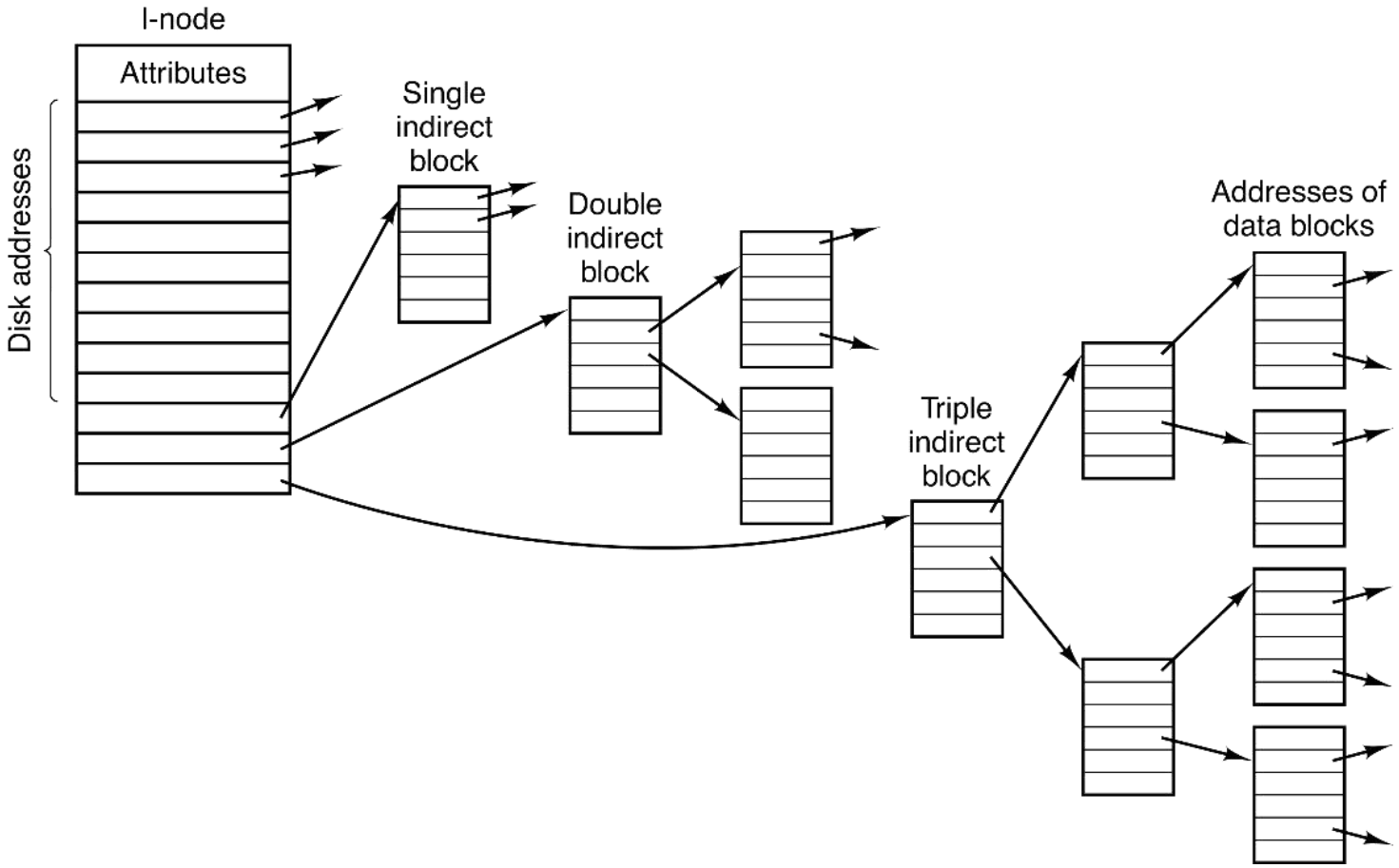
Implementando o Sistema de arquivos – Arquivos

- *I-nodes*:



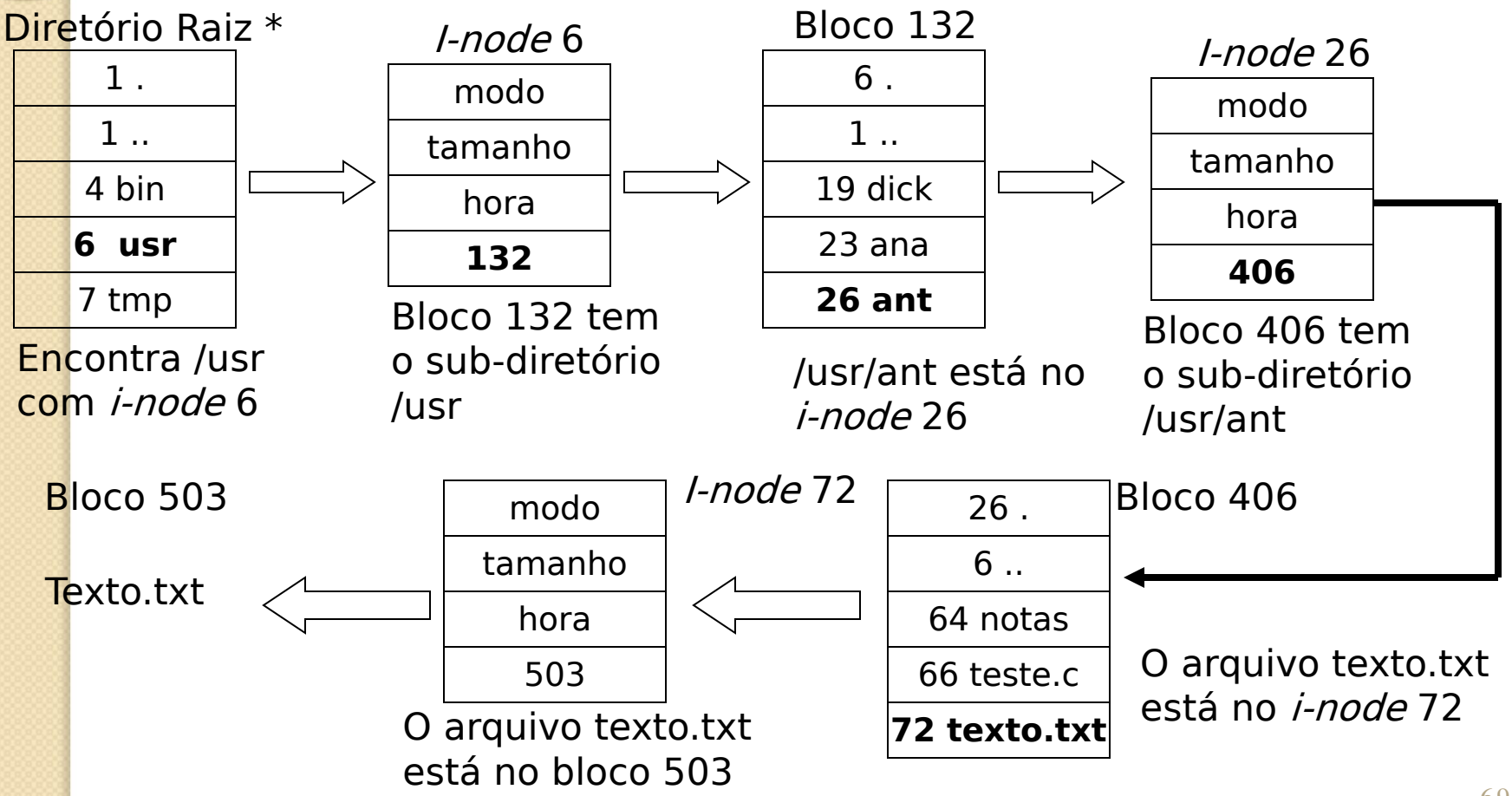
Implementando o Sistema de arquivos – Arquivos

- *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 o caminho fornecido pelo usuário 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, que pode ser:
  - Endereço do arquivo inteiro (alocação contínua);
  - Número do primeiro bloco do arquivo (alocação com listas encadeadas);
  - Número do *i-node*;
- O serviço de diretório é responsável por mapear o nome ASCII do arquivo na informação necessária para localizar os dados

Implementando o Sistema de arquivos – Diretórios

- O serviço de diretório também é responsável por manter armazenados os atributos relacionados a um arquivo:

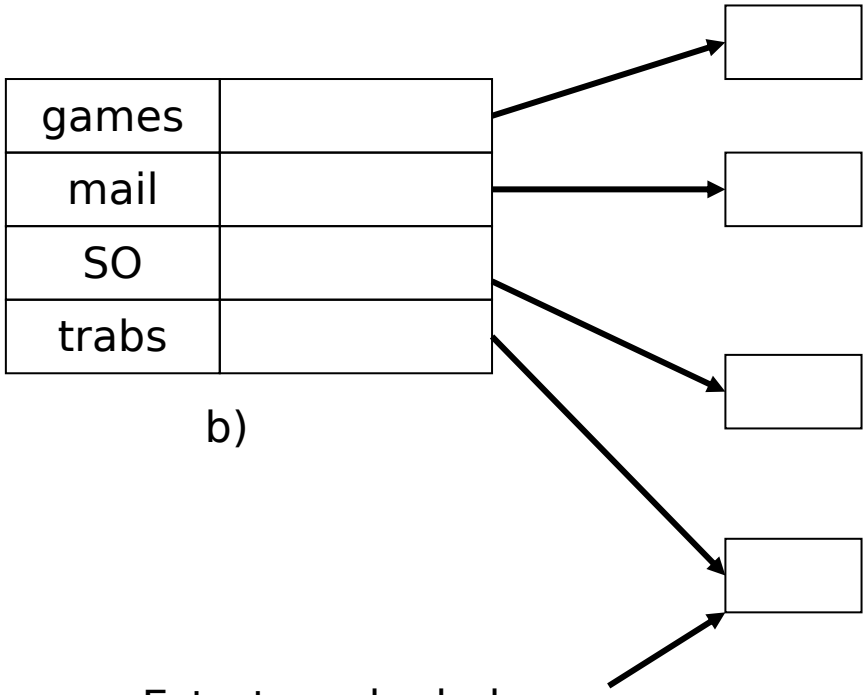
a) Entrada do Diretório: Diretório consiste de uma lista de entradas com tamanho fixo (uma para cada arquivo) contendo um nome de arquivo (tamanho fixo), uma estrutura de atributos de arquivos, e um ou mais endereços de disco; MS/DOS e Windows;

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

Implementando o Sistema de Arquivos – Diretórios

games	atributos
mail	atributos
SO	atributos
trabs	atributos

a)



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, e vem seguido por outros dados (atributos) com formato fixo

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)

- Cada nome do arquivo é preenchido de modo a ser composto por um número inteiro de palavras

- Quando não ocupar toda a palavra, preenche de modo a completar a palavra (parte sombreada);

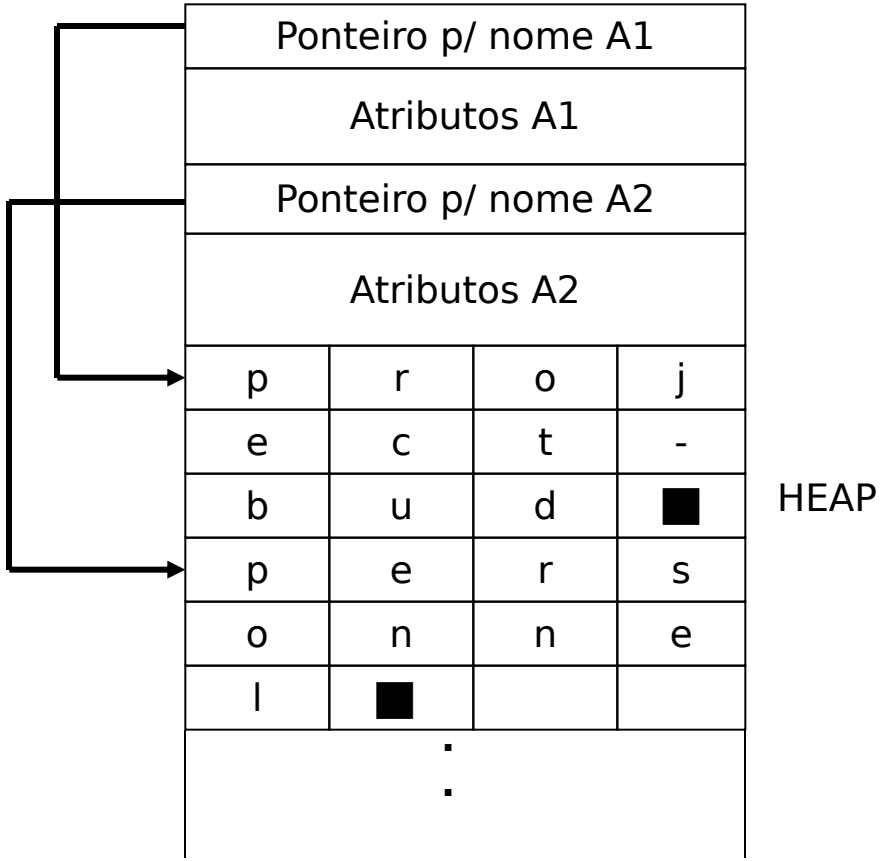
Problema: se arquivo é removido, um espaço em branco é inserido e pode ser que o nome de outro arquivo não possua o mesmo tamanho

Implementando o Sistema de Arquivos – Diretórios

Tamanho do nome de arquivo variável

- Outra solução consiste em tornar fixos os tamanhos das entradas de diretórios e armazenar em um *heap* no fim do diretório os nomes dos arquivos

- Torna-se desnecessário que os nomes de arquivos comecem junto aos limites das palavras, evitando completar posições com caracteres desnecessários



b)

Implementando o Sistema de Arquivos – Diretórios

- Busca em diretório:
  - **Linear** → lenta para diretórios muito grandes;
  - Uso de 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** → armazena resultados de consultas anteriores em uma cache de busca; métodos ótimo se um número relativamente pequeno de arquivos compreender a maioria das buscas

Implementando o Sistema de Arquivos – Gerenciamento de espaço em disco

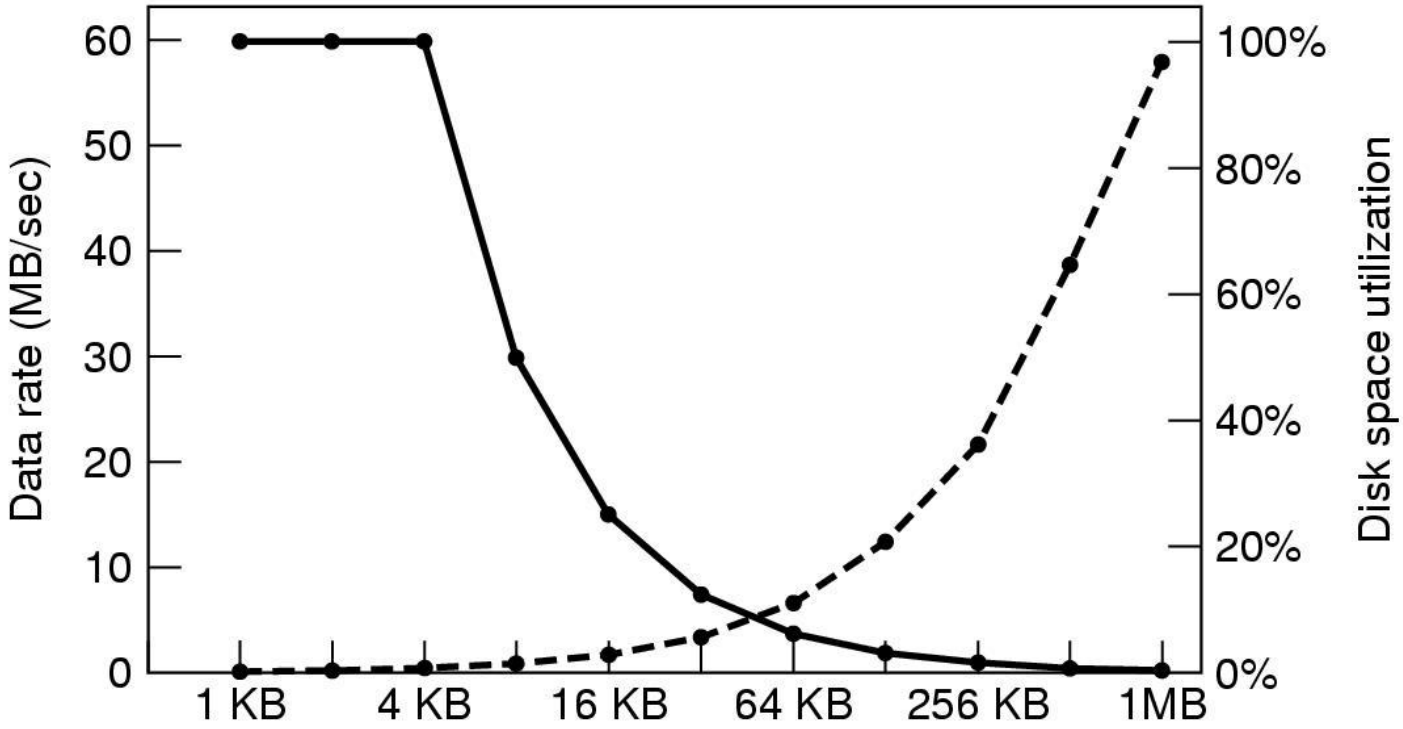
- Duas estratégias são possíveis para armazenar um arquivo de ***n*** bytes:
  - São alocados ao arquivo *n* bytes consecutivos do espaço disponível em disco;
  - Arquivo é espalhado por um número de blocos não necessariamente contínuos → blocos com tamanho fixo;
    - A maioria dos sistemas de arquivos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

Implementando o Sistema de Arquivos – Gerenciamento de espaço em disco

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

Implementando o Sistema de Arquivos – Gerenciamento de espaço em disco

Tamanho dos arquivos: 4 KB



- **Curva tracejada - Taxa de Dados x Tamanho do Bloco**
- **Curva contínua - Utilização do disco 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: 512 bytes a 4 Kb;
- WINXP em diante: 4Kb (até 16 TB); suporte até 256 TB
- Linux: 1Kb, 2Kb , 4Kb;

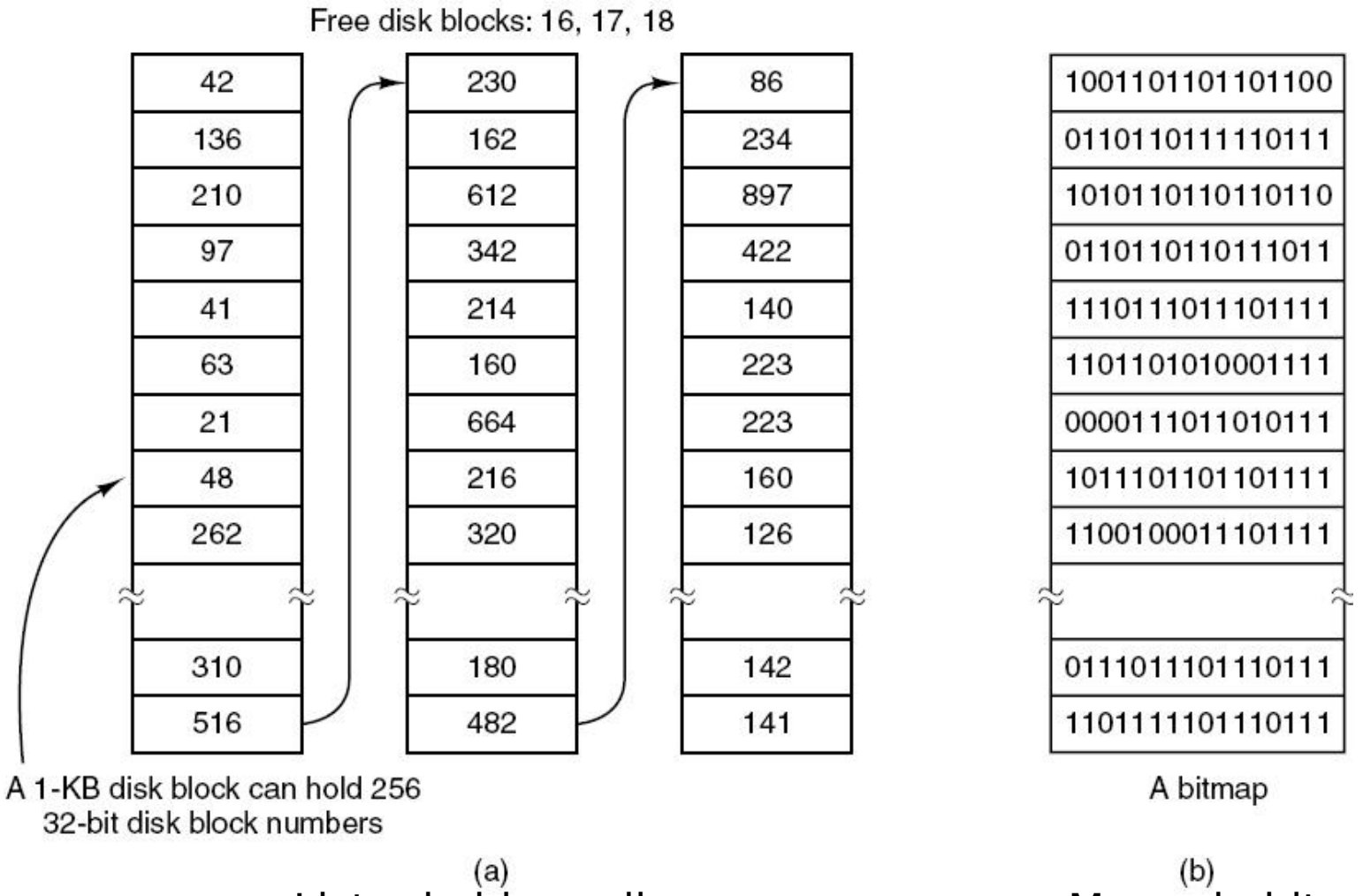
Implementando o Sistema de Arquivos – Gerenciamento de espaço em disco

- Controle de blocos livres : dois métodos:
  - Lista ligada de blocos livres: 32 (ou 64, em sistemas atuais) bits para endereçar 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

- 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;

Implementando o Sistema de Arquivos – Gerenciamento de espaço em disco



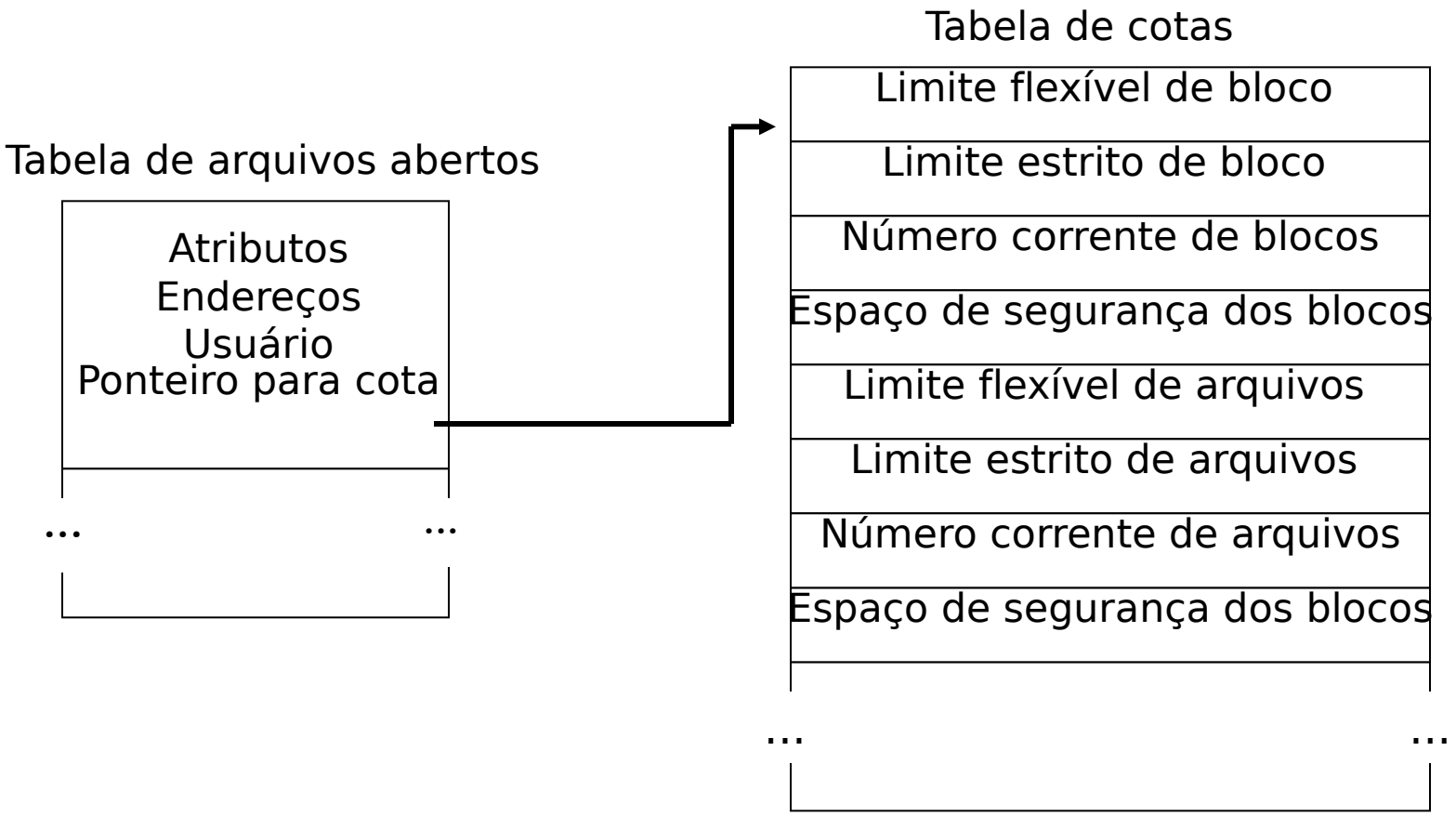
Lista de blocos livres

Mapa de bits

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;
  - Ideia: o 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 o registro de todas as cotas do usuário;

Implementando o Sistema de Arquivos – Gerenciamento de espaço em disco



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 ser (e geralmente é) 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 todo o 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

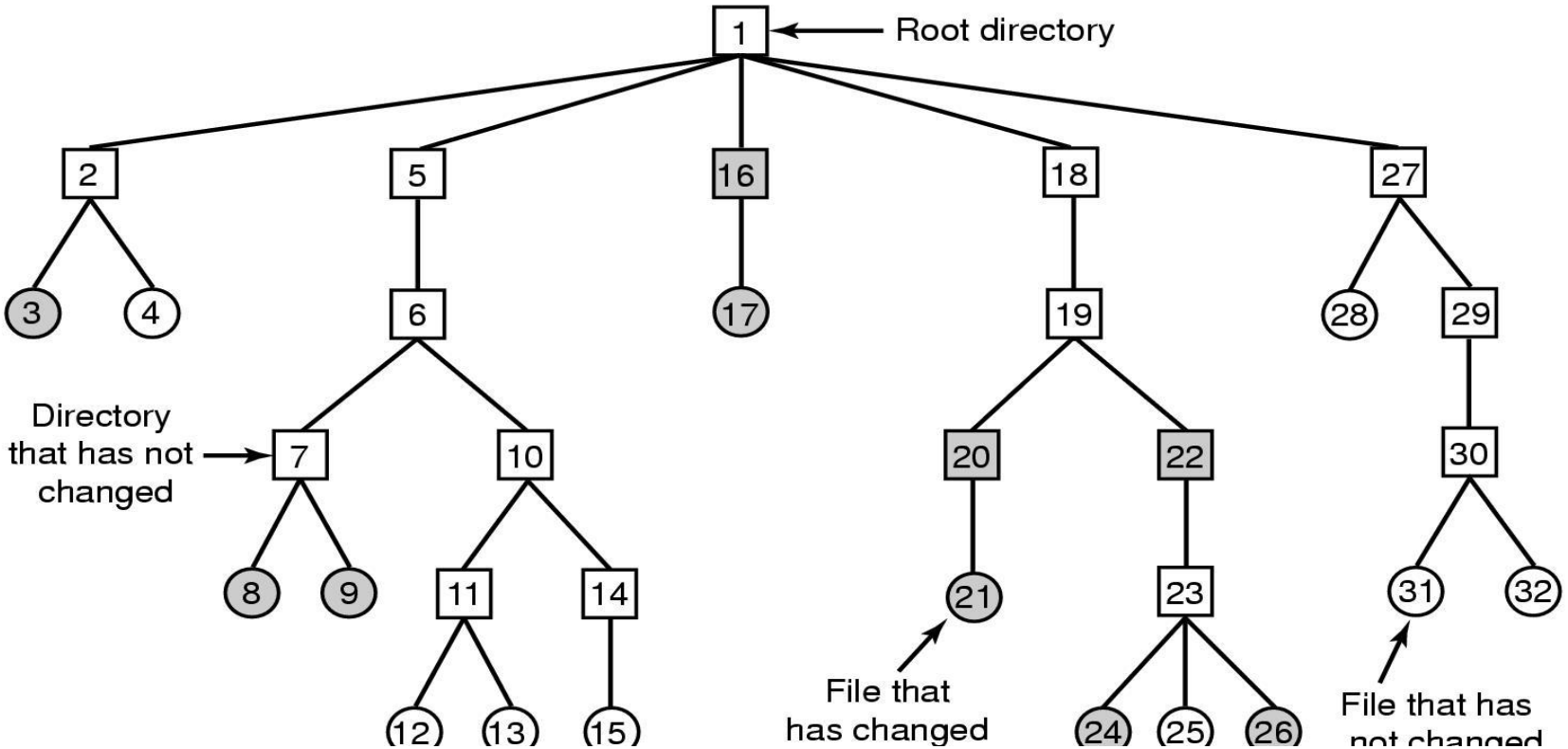
- Estratégias utilizadas para *backup*:
  - Física: a operação de cópia inicia no bloco 0 e encerra somente no último bloco, independente se existem ou não arquivos nesses blocos;
    - Desvantagens:
      - Copiar blocos ainda não utilizados não é interessante;
      - Possibilidade de copiar blocos com defeitos;
      - Difícil restaurar diretórios/arquivos específicos;
      - Incapacidade de saltar diretórios específicos;
      - Não permite cópias incrementais;
    - Vantagens:
      - Como apenas copia tudo, sem nenhum tipo de análise, é um programa extremamente simples de ser desenvolvido;

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

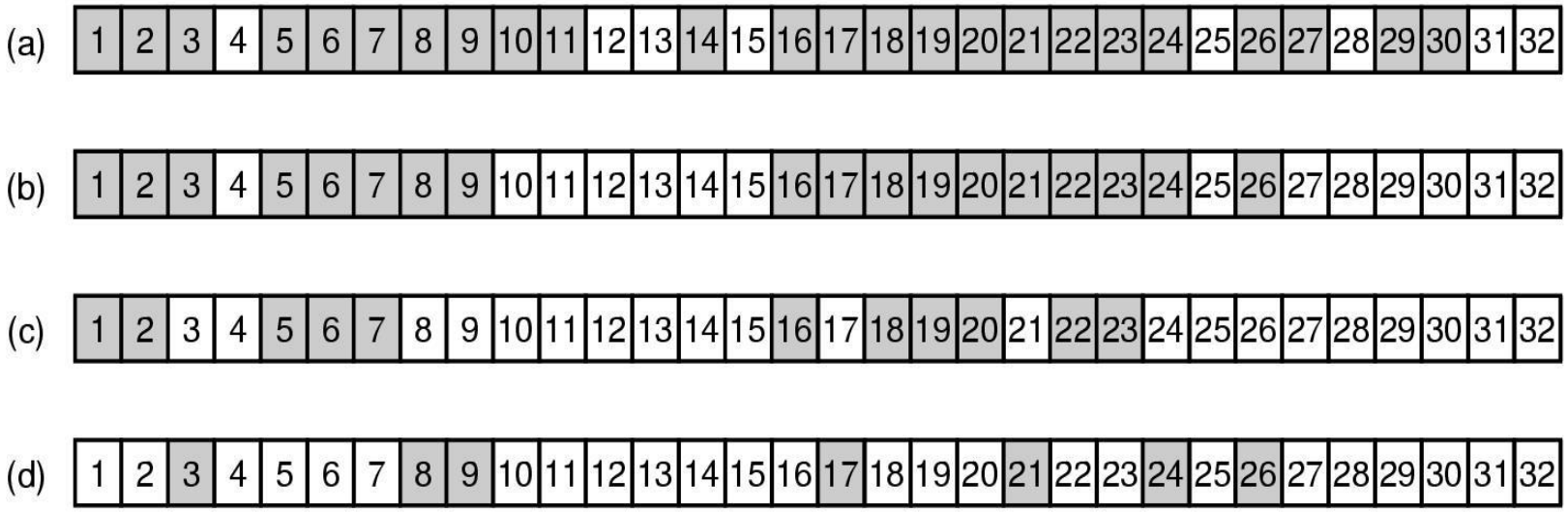


Implementando o Sistema de Arquivos – Confiabilidade

- 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

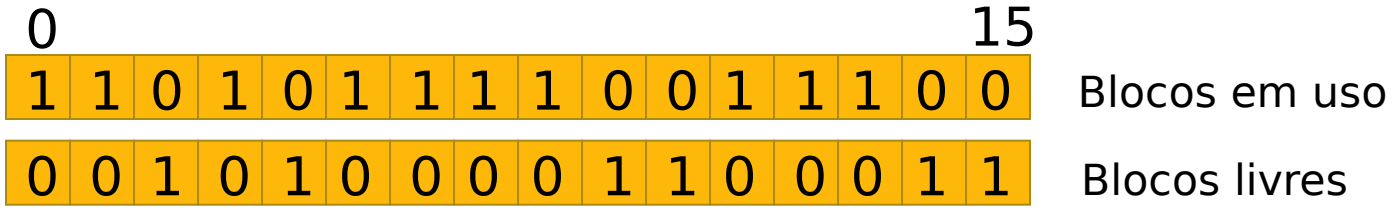
- Consistência → dados no sistema de arquivos devem estar consistentes;
- Crítico: blocos de *i-nodes*, blocos de diretórios ou blocos contendo a lista de blocos livres/mapa de bits de blocos livres;
- Diferentes sistemas possuem diferentes programas utilitários para lidar com inconsistências:
  - UNIX: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

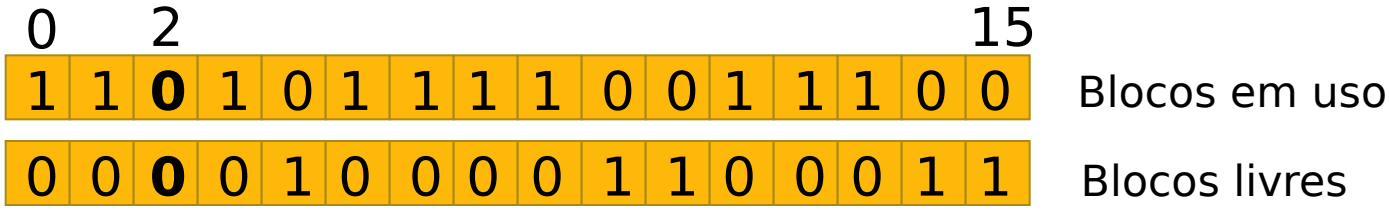
Implementando o Sistema de Arquivos – Confiabilidade

a)

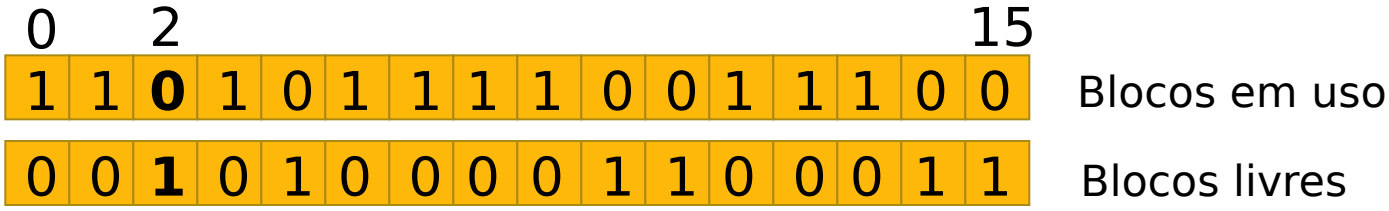


Se ocorrem problemas, podemos ter as seguintes situações:

b)

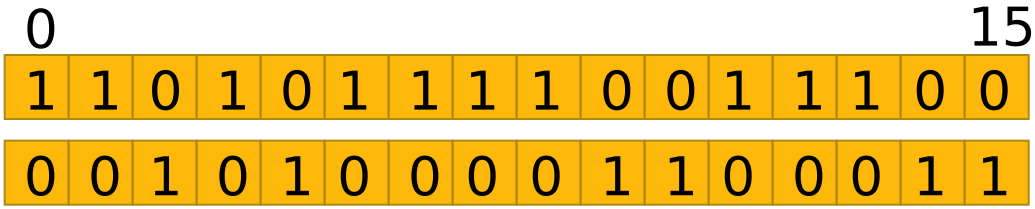


Bloco **2** perdido (*missing block*)
Solução: colocá-lo na lista de livres

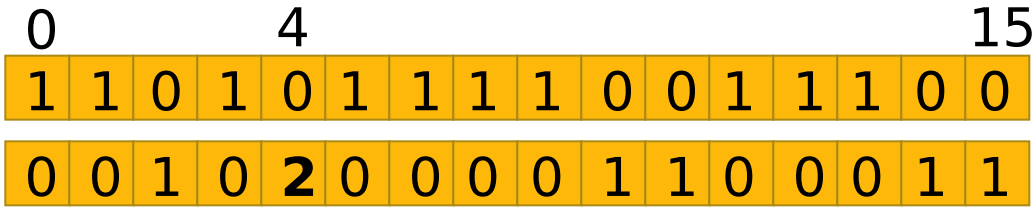


Implementando o Sistema de Arquivos – Confiabilidade

a)



c)

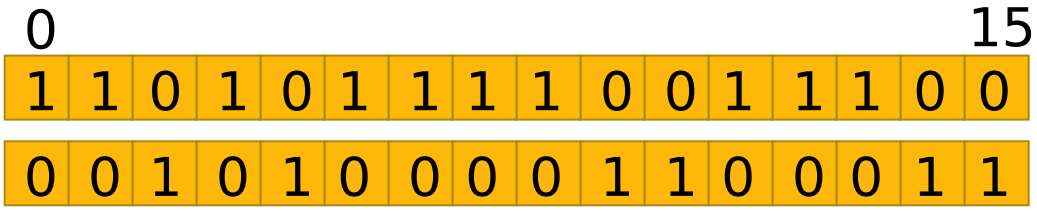


Bloco **4** duplicado na lista de livres

Solução: reconstruir a lista; Essa situação só ocorre se existir uma lista encadeada de blocos livres ao invés de um mapa de bits.

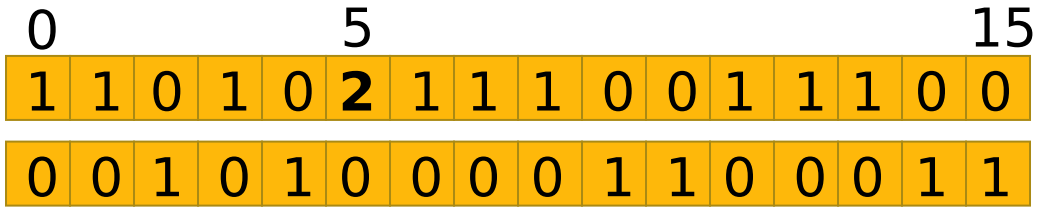
Implementando o Sistema de Arquivos – Confiabilidade

a)



Se problemas acontecerem, podemos ter as seguintes situações:

d)



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, copiar para esse bloco o conteúdo do bloco 5 e atribuir esse bloco a um dos arquivos, avisando 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;
    - Se a contagem de ligações no i-node for maior que o valor contado (contador de arquivo):
      - Problema: i-node não será removido quando o(s) arquivo(s) for(em) apagado(s)
    - Se for menor:
      - Problema: quando chegar em zero, o sistema marca o i-node como não usado e libera os blocos, mas ainda tem arquivo apontando para aquele i-node
    - Solução para ambos: atribuir o valor do contador de arquivos à contagem de ligações do i-node;

Exemplos reais de sistemas de arquivos

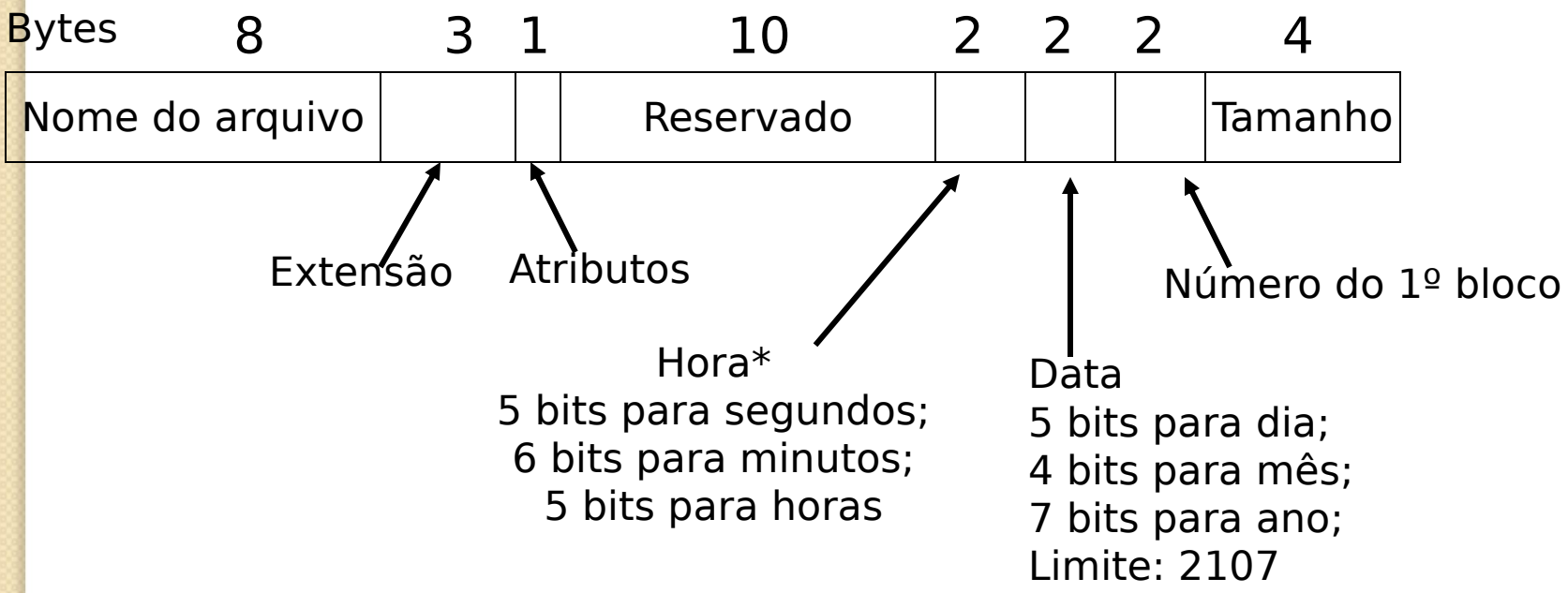
- FAT (diversas versões) e NTFS;
- *I-node* do Unix e NFS;
- *Extended File System* (EXT2, EXT3, EXT4), criado em 1992 como uma evolução do *Unix File System* (UFS);
- ReiserFS;

Sistemas de Arquivos FAT

- FAT surgiu por volta de 1976, sendo utilizada no SO do Intel 8086;
- A FAT no MS-DOS:
  - Limite de nome de arquivo: 8 + 3 (8.3) caracteres;
  - Hierarquia de diretórios: árvore começa no diretório raiz (*root directory*);
  - Não existe o conceito de diferentes usuários, portanto, todos os arquivos podem ser acessados por todos os usuários;
  - Cada entrada de diretório possui um tamanho fixo de 32 bytes;

Sistemas de Arquivos FAT

Entrada de diretório do MS-DOS:



* Defasagem de 2 segundos

Tamanho de arquivo: 2Gb

Sistemas de Arquivos FAT

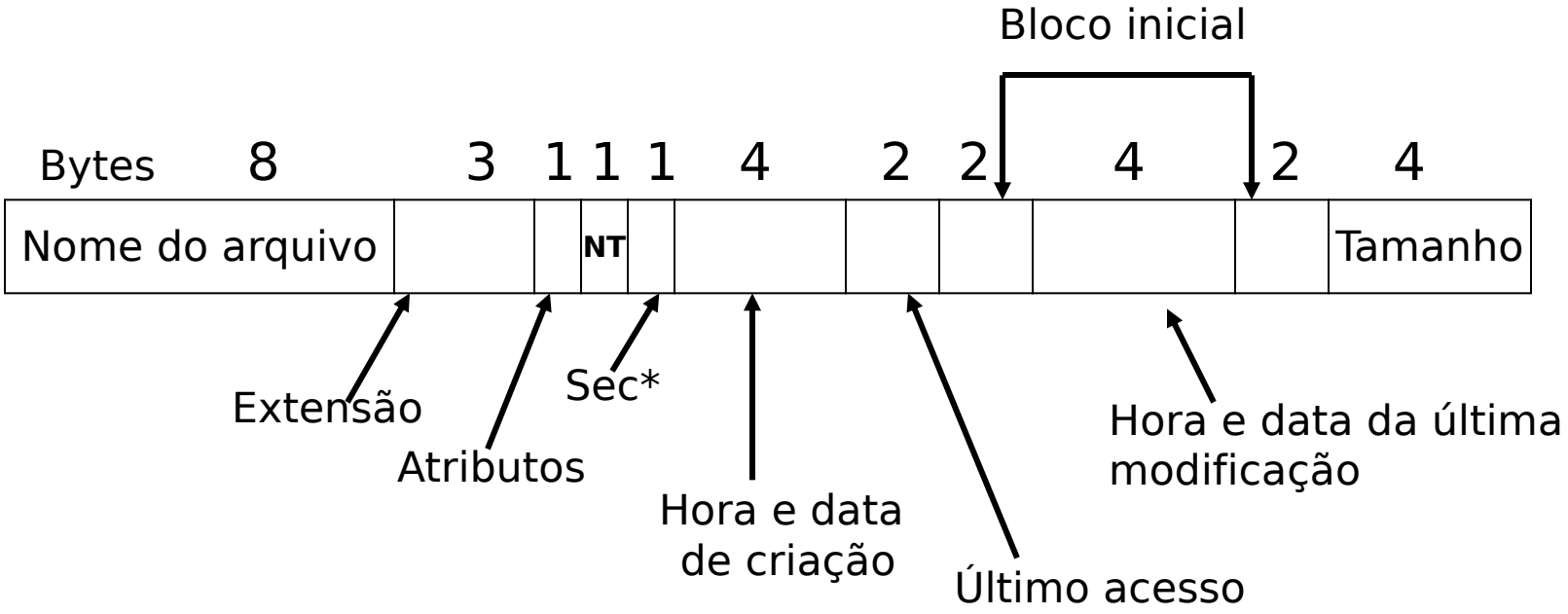
- Três versões: FAT12, FAT16, FAT32 (VFAT), dependendo de quantos bits são utilizados para armazenar cada posição na FAT;
- FAT12:
  - Tamanho de bloco: 512bytes, 1K, 2K e 4K;
  - Tamanho de partição → até 16 Mb;
- FAT16:
  - Tamanho de bloco: 8k, 16k e 32k;
  - Tamanho de partição → até 2Gb;

Sistemas de Arquivos FAT

- FAT32: iniciou-se com a segunda versão do Windows95 (Win95 OSR2 – OEM Service Release 2)
  - Tamanho de bloco: 4k, 8k, 16k e 32k;
  - Tamanho de partição – 2Tb;
- Controle de blocos livres é feito utilizando a FAT;
- Windows98 utiliza FAT32 e permite arquivos com nomes longos (mais de 8 caracteres);

Sistemas de Arquivos FAT

Entrada de diretório do Windows98:



- NT: compatibilidade com o Windows NT

* Precisão de até 10mseg na data da criação do arquivo.

Sistemas de Arquivos FAT

Atributos	Código Binário
Read-only	00000001
Hidden	00000010
System	00000100
Volume Label	00001000
Diretório	00010000
Archive	00100000

* Soma do código binário combina os atributos

\+ → 00010011

Backup:
Setado → modificado;
Valor zero → copiado;

Sistemas de Arquivos FAT

- Para compatibilidade com o MS-DOS, o Windows98 armazena dois nomes de arquivos:
  - Um com o número total de caracteres utilizados;
  - E outro com o número permitido pelo MS-DOS → uso do ~1, ~2, ..., ~n (dois últimos caracteres do nome do arquivo);
  - Análise dos 6 primeiros caracteres: os caracteres não válidos para o MS-DOS (+ , ; = [ ]) são trocados por "_" (*underscores*);
  - Todas as letras minúsculas são convertidas para letras maiúsculas;
  - Espaços em branco são removidos;

Sistemas de Arquivos FAT

Tamanho de bloco	FAT12	FAT16	FAT32
512bytes	2Mb	-	-
1kb	4Mb	-	-
2kb	8Mb	128Mb	-
4kb	16Mb	256Mb	1Tb
8kb	-	512Mb	2Tb
16kb	-	1024Mb (1Gb)	2Tb
32kb	-	2048Mb (2Gb)	2Tb

Sistemas de Arquivos NTFS

- NTFS não está baseado no Sistema de Arquivos FAT, mas se utiliza de algumas características do HPFS (*High Performance File System* – sistema de arquivos do OS/2);
- Características:
  - Confiabilidade → capacidade de se recuperar de problemas sem perda de dados; melhorara a tolerância a falhas;

Sistemas de Arquivos NTFS

- Segurança e Controle de Acesso *(DAC – Discretionary Access Control)*: estabelece diretivas que permitem implementar controle de acesso em arquivos e diretórios, inexistente no Sistema de Arquivos FAT;
- Permite maiores partições no disco;
- Sistema de caracteres: UNICODE;
- Caminho: até 32.767 caracteres;

Sistemas de Arquivos NTFS

- Suporta *Case Sensitive*, no entanto, essa característica é perdida devido à Win32 API;
- Suporte à rede;
- Compressão de arquivos;
- Tamanho de blocos → 512bytes até 64Kb;
  - Windows2000 → 4Kb;
- Baseado no conceito de transações → tarefa é cumprida até o fim ou é abortada;
- Suporte à criptografia de arquivos → *driver* EFS (*encrypting file system*); chave de 128 bits;
- *Links* simbólicos;

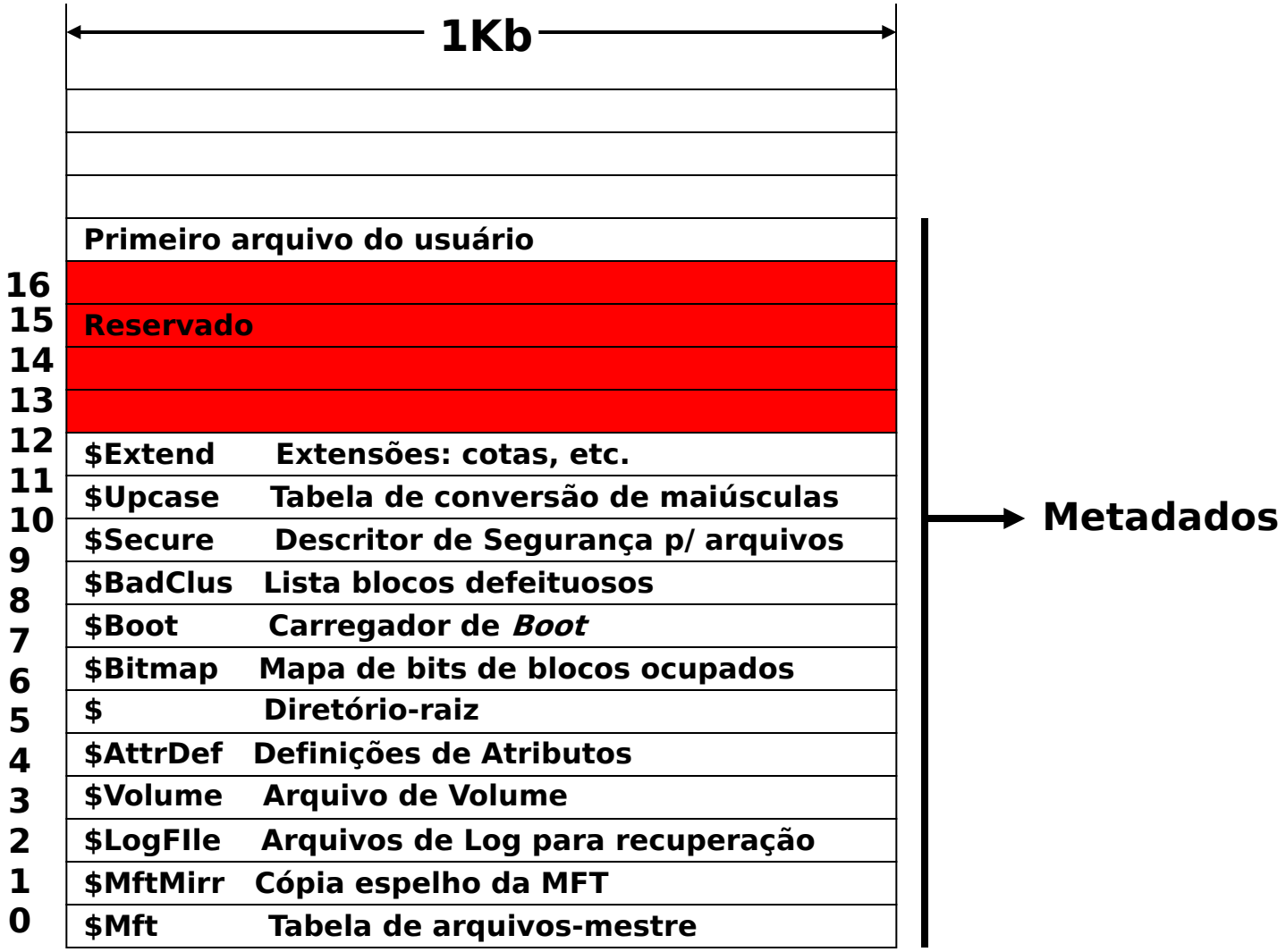
Sistemas de Arquivos NTFS

- Cada arquivo é um conjunto de atributos, cada qual representado por um *stream* de bytes;
- Sistema de arquivos hierárquico → diretório de trabalho corrente, caminho relativo e absoluto;
- *Master File Table* (MFT): armazena a estrutura do NTFS e as informações sobre arquivos/diretórios;
- Bloco de *boot* tem o endereço da MFT;

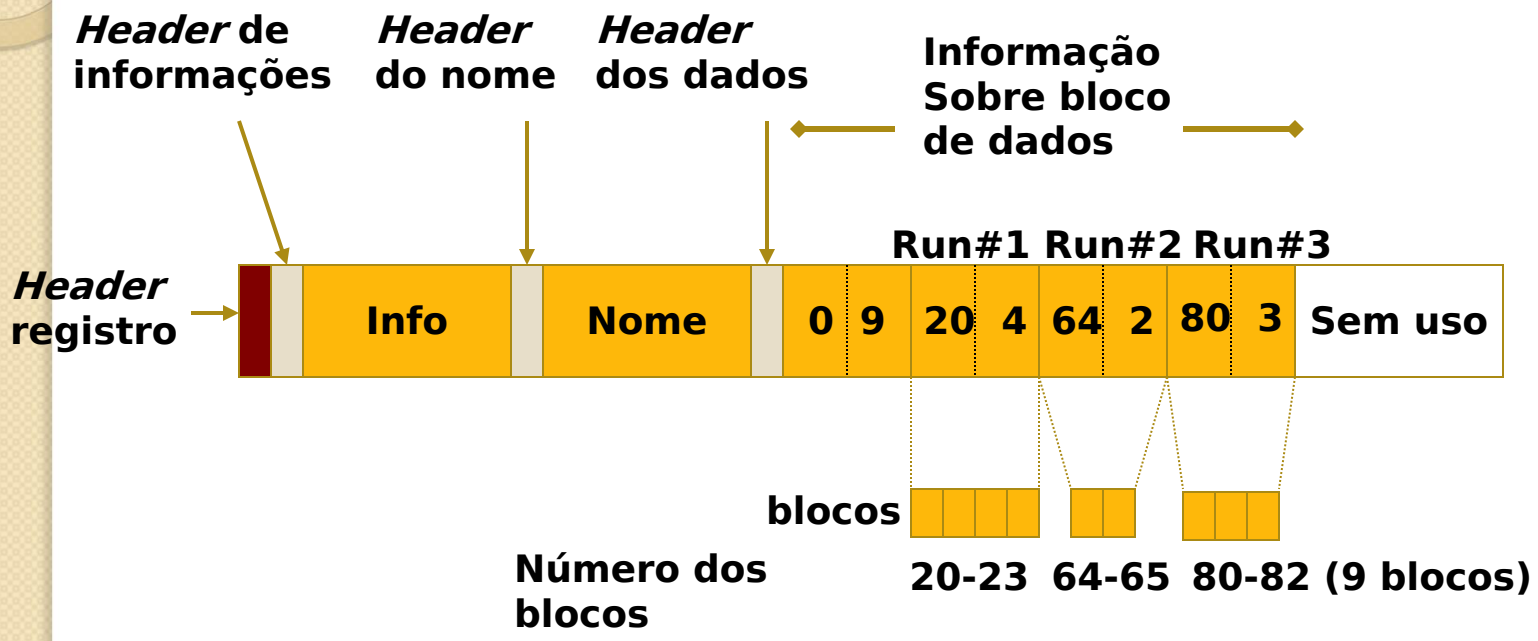
Sistemas de Arquivos NTFS

- MFT é uma sequência linear de registros de 1Kb → é um arquivo;
- Cada registro descreve um arquivo ou diretório → informações como nome, lista de endereço de onde seus blocos estão alocados;
  - Até 2^{48} registros;

Sistemas de Arquivos NTFS – MFT



Sistemas de Arquivos NTFS – MFT

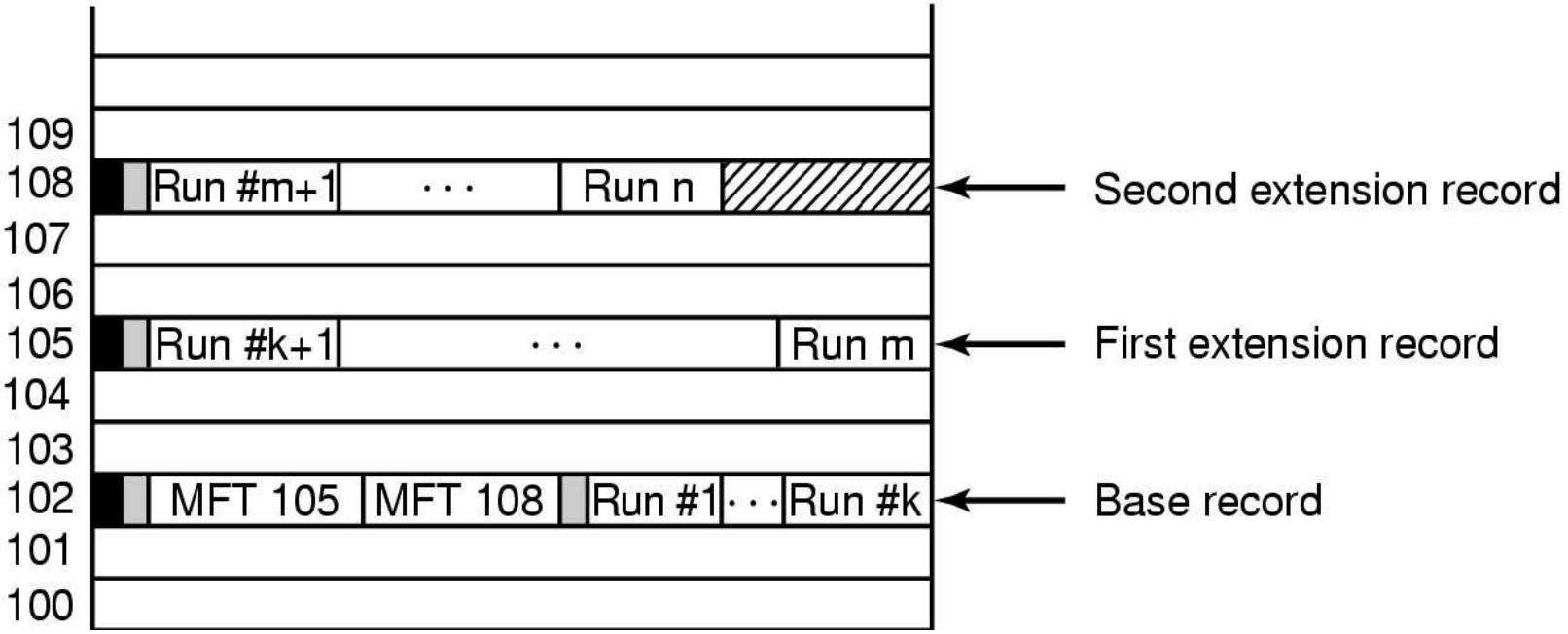


Exemplo de um Registro MFT

Sistemas de Arquivos NTFS – MFT

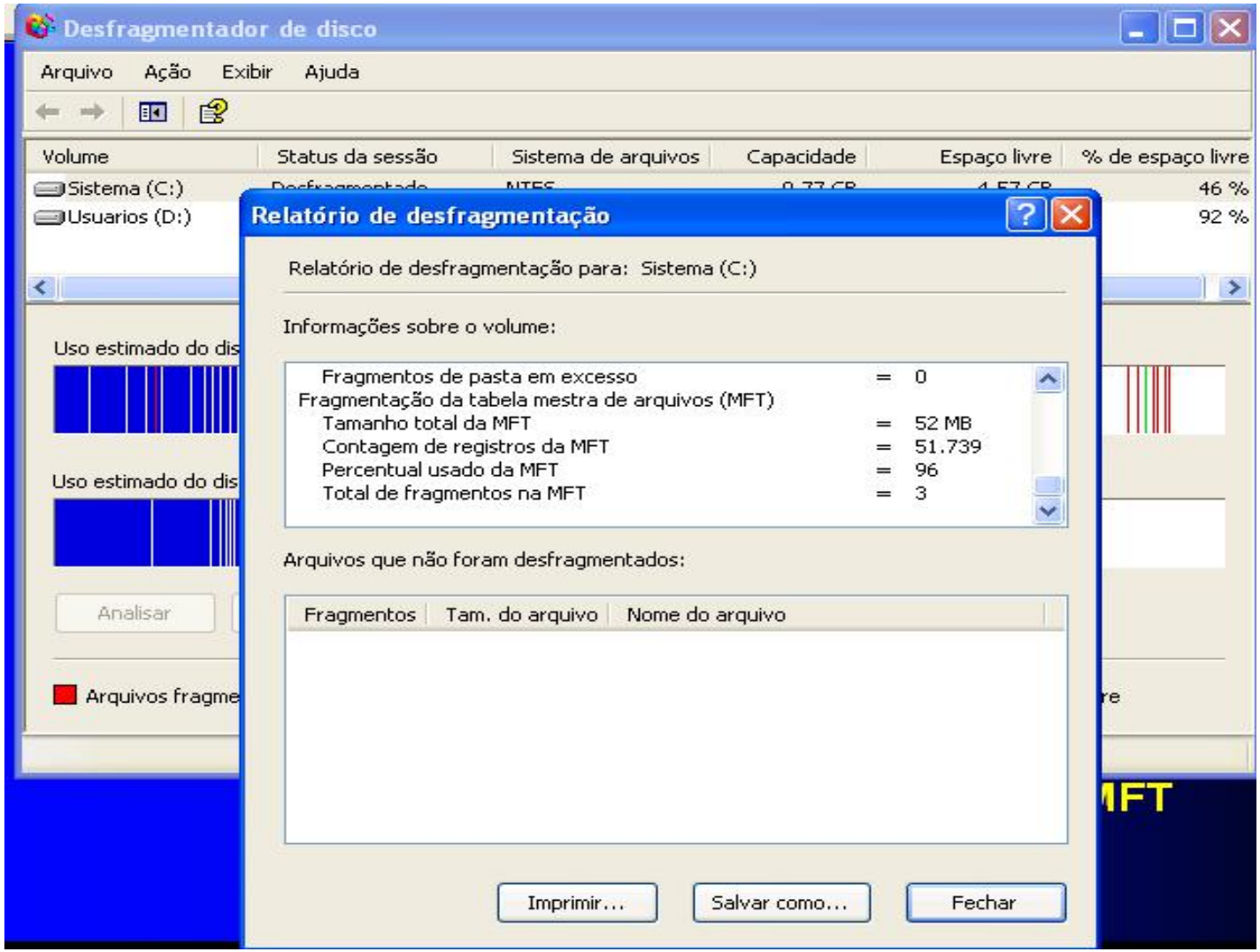
- Campo Info
  - Campo de tamanho fixo e obrigatório, contendo:
    - Proprietário do arquivo
    - Informações de segurança
    - Marcadores de tempo
    - Contador de ligações
    - Bits indicando se o arquivo é apenas de leitura
    - Etc
- Campo Nome
  - Unicode
  - Tamanho variável

Sistemas de Arquivos NTFS – MFT



Arquivo que requer três entradas na MFT

Sistemas de Arquivos NTFS – MFT

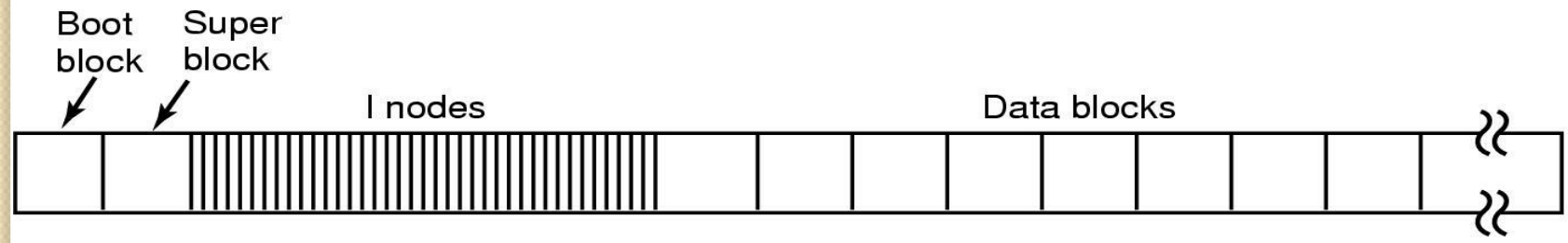


Sistemas de Arquivos

Características	FAT16	FAT32	NTFS
Sistemas	DOS, Win9x, NT	Win98, Win2000, WinXP, Win Vista	NT4, Win2000, WinXP, WinVista, 7, 8
Partição	2Gb	2Tb	Ilimitada
Número de arquivos	65000	Ilimitado	Ilimitado
Atributos	Conjunto básico	Conjunto básico	Vários
Segurança	Não	Não	Sim
Compressão	Não	Não	Sim

Sistema de Arquivos UNIX

- Para o UNIX um arquivo é uma seqüência de 0's ou mais bytes contendo dados;
- Nenhuma distinção é feita entre arquivos ASCII, binários ou outros;
- Usa o conceito de *i-nodes* (64 bytes) associados aos arquivos → tabela;
- Esquema do disco no UNIX clássico:



Sistema de Arquivos *I-node* do UNIX

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

64

Estrutura *i-node* do System V

Sistemas de Arquivos

- Diversos são os sistemas de arquivos utilizados pelo LINUX:
  - ExtFS, Ext2FS, Ext3FS, Ext4FS, Xia;
  - CFS, TCFS, VFS, GFV, NFS, HPFS, SYSV;
  - ReiserFS; JFS (IBM);
- Primeiro foi baseado no *Minix*;

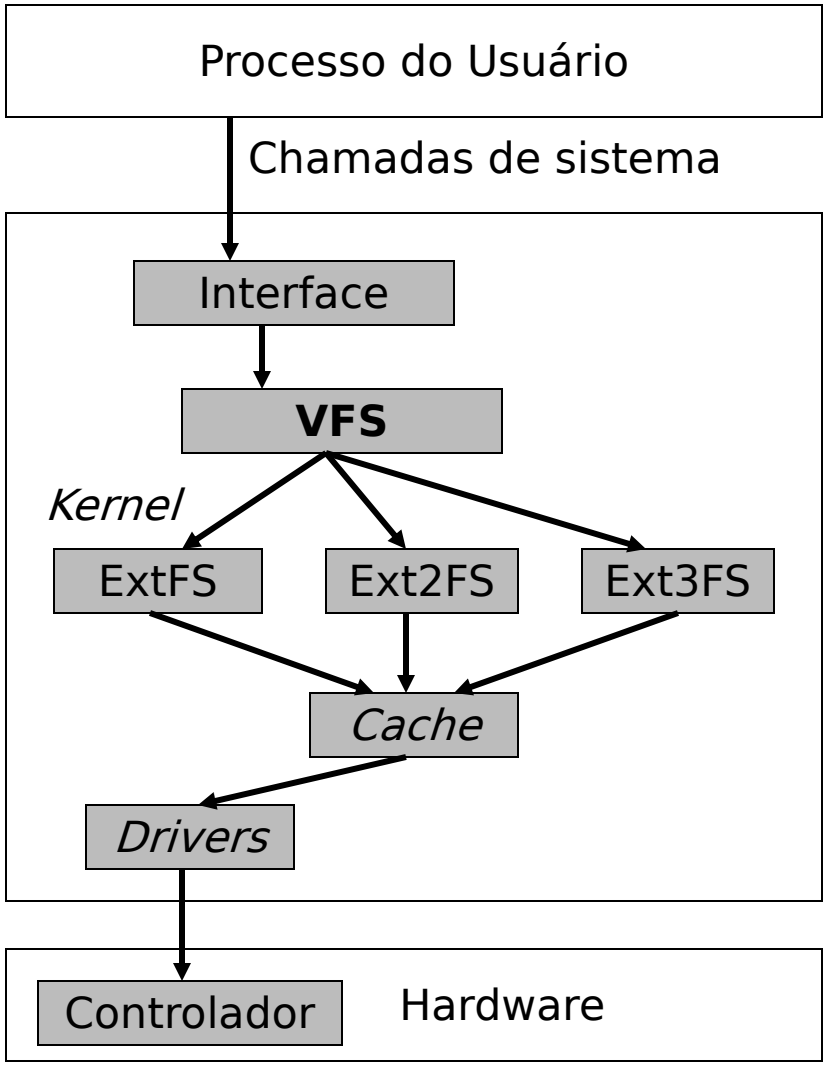
Sistemas de Arquivos

- extFS (Extended File System - 1992);
  - Quase uma versão experimental, baseada no UFS (Unix File System), apresentava problemas de fragmentação de arquivos e imutabilidade dos *i-nodes*
- ext2FS (1993);
  - Até Red Hat 7.2;
- ext3FS (2001);
  - Red Hat 7.3;
  - Conectiva 8;
- ext4FS (2008)
  - Atualização do ext3, usada na maior parte das distribuições linux atuais

Sistemas de Arquivos Ext2/Ext3

- Tamanho de blocos: 1kb, 2kb, 4kb;
- Estrutura hierárquica de diretórios;
- Assim como o UNIX, o LINUX também utiliza a estrutura de *i-nodes* vinculada a cada arquivo;
- Controle de blocos livres → mapa de *bits*;
- Tanto o mapa de bits quanto a tabela de *i-nodes* são armazenados no disco;

Sistemas de Arquivos Ext2/Ext3



VFS (*virtual file system*):

- Possui informações dos sistemas de arquivos;
- Conjunto de operações;
- Facilita a adição de novos sistemas de arquivos no *kernel*;

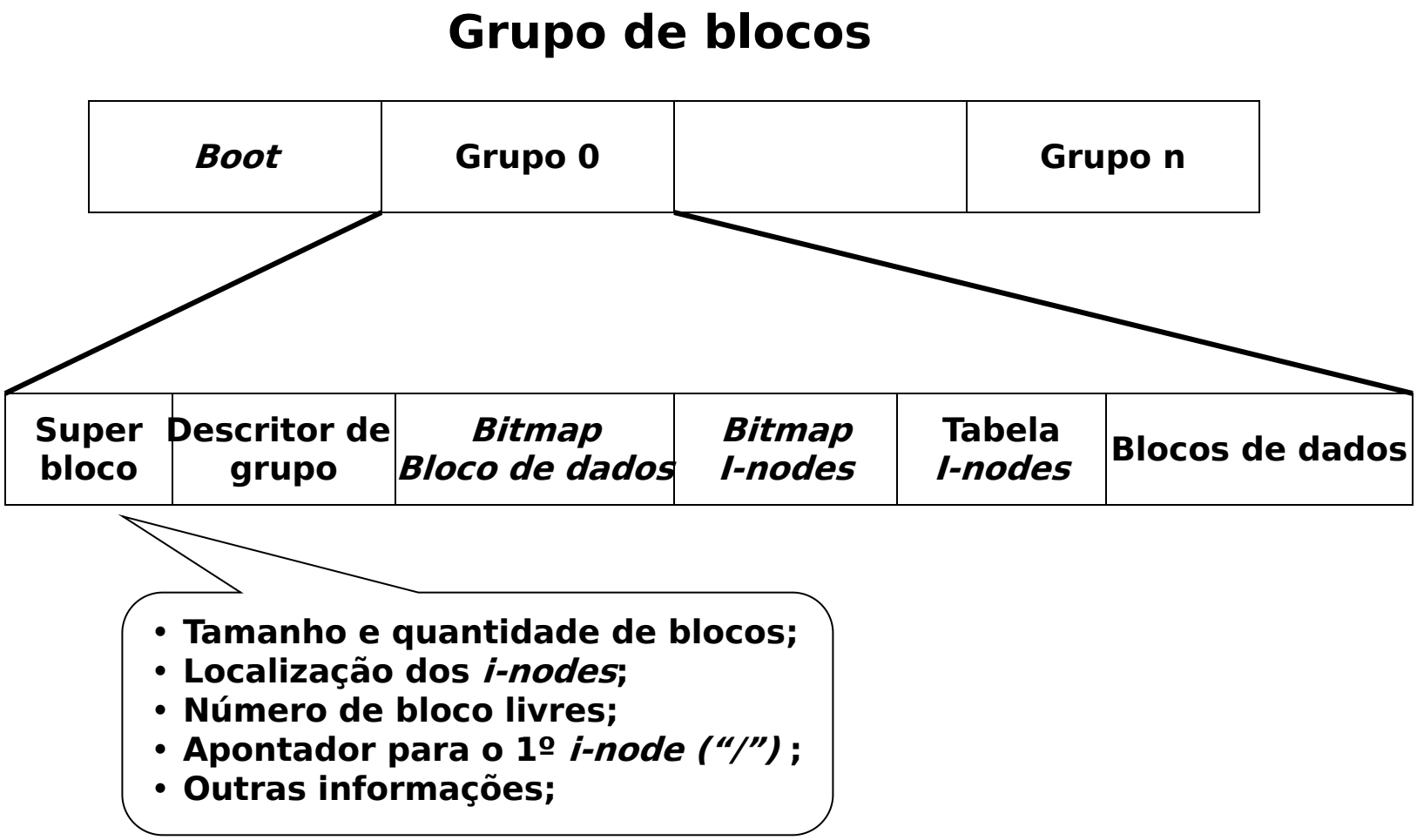
Sistemas de Arquivos Ext2/Ext3

- Características:
  - 5% dos blocos são armazenados para o administrador do sistema (*root*);
  - Permite atualizações síncronas; (*write-through* do MS-DOS);
  - *Links* simbólicos;
  - Controla o *status* do sistema de arquivos utilizando um **Superbloco**;

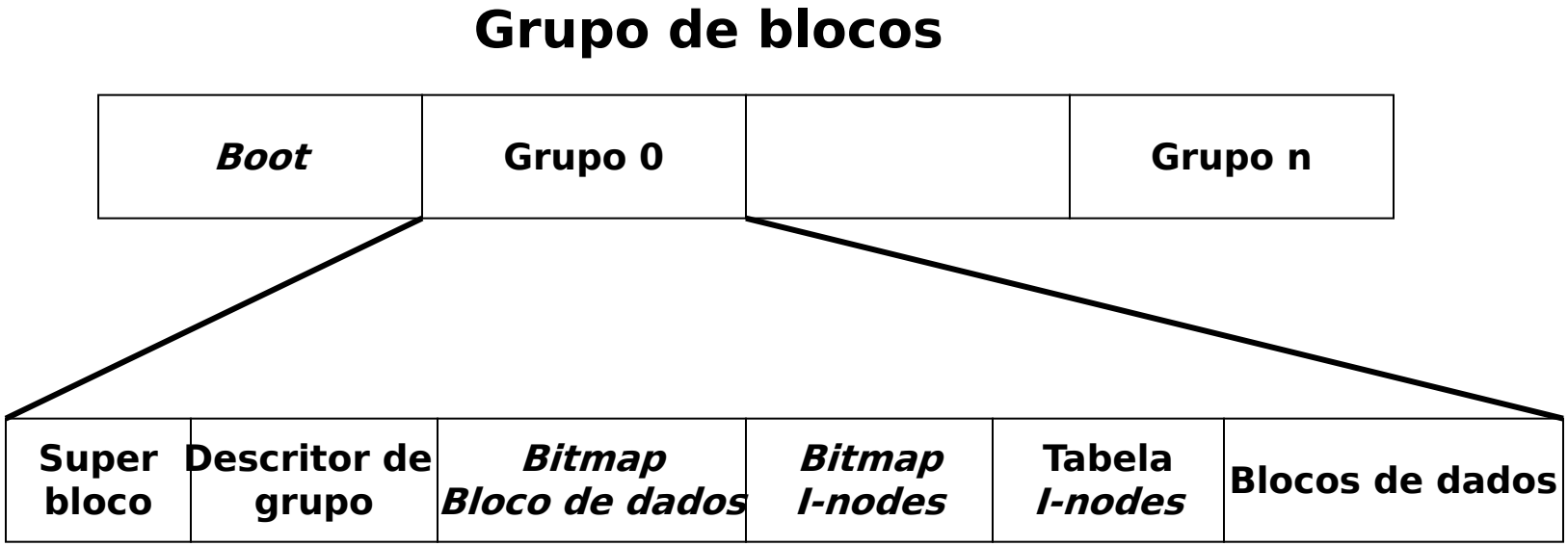
Sistemas de Arquivos Ext2/Ext3

- Pré-alocação de blocos contínuos (adjacentes) → 8 blocos;
- Partições são divididas em grupos de blocos, cada qual com um **Superbloco**, mapa de *bits*, *i-nodes*;
  - Confiabilidade;
  - Desempenho → menor número de acessos;
- Limite de tamanho de partição: 4Tb;
- Limite de tamanho de nome de arquivos: 255 caracteres (podendo ser estendido para 1012);

Sistemas de Arquivos Ext2/Ext3

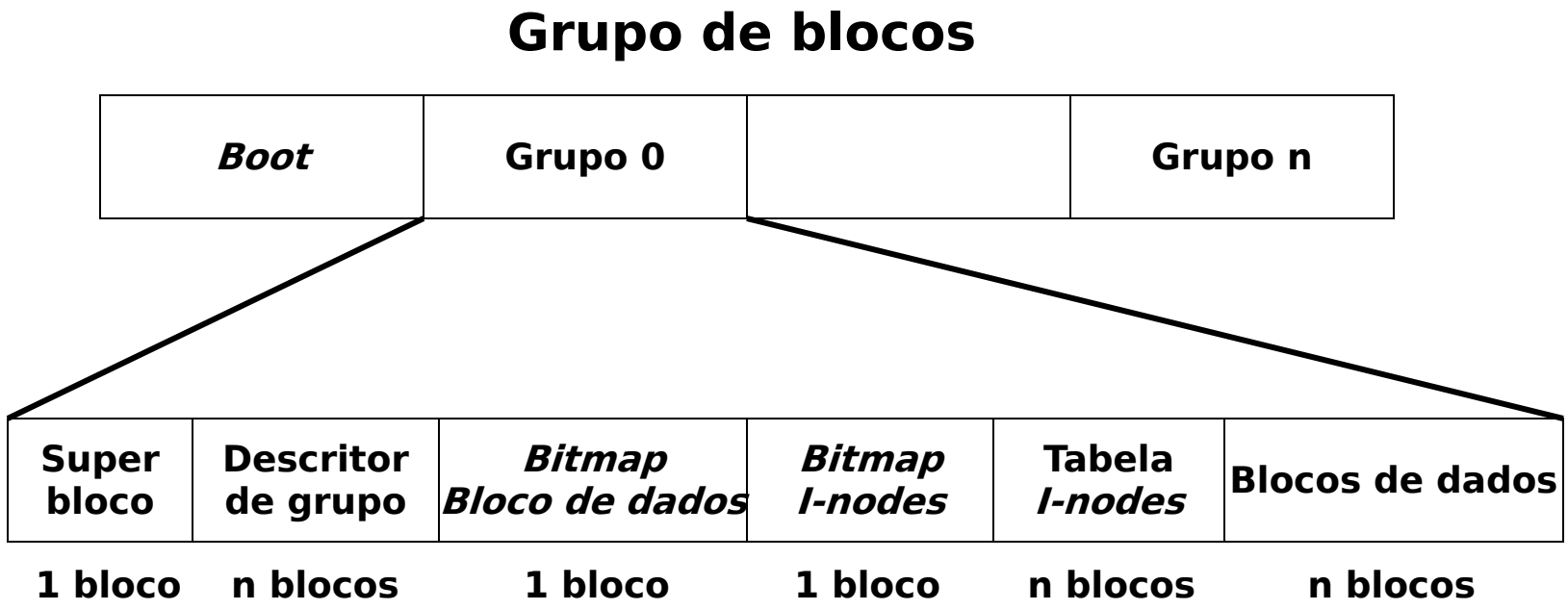


Sistemas de Arquivos Ext2/Ext3



- **Estrutura de cada grupo;**
- **Número do bloco no qual está armazenado o *bitmap* para blocos ocupados;**
- **Número de diretórios do grupo;**

Sistemas de Arquivos Ext2/Ext3



Sistemas de Arquivos Ext2/Ext3

- Diretórios são gerenciados como listas ligadas com entradas de tamanho variado;
- Cada entrada possui os seguintes campos:
  - Número do *i-node*;
  - Tamanho da entrada (Tent);
  - Tamanho do nome (Tnome);
  - Tipo do arquivo (Tipo);
  - Nome;

I-node	Tent	Tnome	Tipo	Nome arquivo

i1	16	05	Ln	file1
i2	40	14	Dir	Long_file_name

Entrada de um diretório Ext2FS

Sistemas de Arquivos Ext2/Ext3

- *I-nodes* possuem tamanho fixo, com:
  - 12 endereços diretos;
  - 1 endereço indireto, 1 duplamente indireto e 1 triplamente indireto;
  - Endereços de 4 bytes (32bits);

- Os *i-nodes* são criados no momento da formatação lógica do dispositivo;
- Assim, é possível redimensionar o número de *inodes* de acordo com a capacidade do dispositivo e o tipo e tamanho de arquivos que serão nele armazenados;

Sistemas de Arquivos Ext2/Ext3

- O GNU/Linux mantém para cada sistema de arquivos montado uma cópia do **superbloco** em memória RAM;
- A chamada de sistema sync atualiza os dados dos *superblocos* que estão armazenados em *cache* para seus locais em disco, sincronizando as informações sobre o sistema de arquivos;
  - Ext2 – a cada 30 segundos;
  - Ext3 – a cada 5 segundos;

Sistemas de Arquivos Ext2/Ext3

- Ext2FS possui uma baixa tolerância à falhas;
- Ext3FS: principal diferença → *journaling*;
  - O sistema mantém *logs* dos eventos, permitindo uma recuperação rápida;

Sistemas de Arquivos Ext3

- A introdução do '***journal***' em sistemas EXT3 modifica a abordagem de recuperação de sistemas de arquivos (*fsck*) e reduz o tempo de parada do sistema para valores muito baixos;
- Uma área é reservada para a alocação do '***journal***' ou '***log***';

Sistemas de Arquivos Ext3

- As operações são primeiramente gravadas no ***journal***;
- Quando a atualização é finalizada, um registro de complemento (***commit record***) é gravado sinalizando o final da entrada. Então, as mudanças são efetivamente gravadas em disco;
- Assim, quando uma falha ocorre, realizando uma consulta ao ***journal***, é possível a reconstrução das operações ainda não concluídas e a rápida recuperação do sistema;

Sistemas de Arquivos Ext3

- Três modos:
  - ***Journaling*** (Registro de ações): Grava todas as mudanças e usa um arquivo de registros de ações maior.
    - É o mais lento, mas possui maior capacidade de evitar perdas;
  - ***Ordered*** (Ordenado): Grava somente mudanças nos metadados (arquivos que possuem informações sobre outros arquivos), mas registra as atualizações nos arquivos de dados antes de fazer as mudanças associadas ao sistema de arquivos.
    - É o padrão do EXT3;
  - ***Writeback***: Grava mudanças nos metadados, mas utiliza o processo de escrita do sistema de arquivos em uso para gravação.
    - É o mais rápido, porém é o menos confiável e mais suscetível à corrupção de arquivos após uma falha;
    - Equivalente à instalação de um sistema com EXT2;

Sistema de Arquivos ReiserFS

- Criado por Hans Reiser e mantido pela NameSys (http://www.namesys.com);
- Arvores balanceadas e finitas (*balanced tree*) (B*) são usadas para organizar o sistema de arquivos (versão melhorada de árvores B+);
- Possui suporte a *journaling*;
- Tamanho de bloco padrão: 4kbytes;
- Alto desempenho com arquivos pequenos, pois seus dados podem ser armazenados próximos aos metadados, então, ambos podem ser recuperados com um pequeno movimento da cabeça de leitura do disco;

Sistema de Arquivos ReiserFS

- O ReiserFS é uma camada semântica com métodos e funções que são referenciados para executar tarefas no sistema de arquivos;
- Trata toda a partição como se fosse uma única tabela de banco de dados contendo diretórios, arquivos e metadados dentro de uma mesma árvore;
  - Tudo que está armazenado na árvore possui uma chave que facilita pesquisas;

Sistema de Arquivos ReiserFS

- Guarda apenas informações sobre os metadados e não as informações dos arquivos em si (como faz o Ext3);
  - Com isso, o ***journal*** armazena menos informações, aumentando as chances de não recuperar dados que estavam sendo gravados no momento de uma falha;
  - Mas melhora o desempenho de acesso;
- Foi conhecido por ser utilizado pela distribuição Linux Suse, até 2006, que passou a utilizar o ext3
- Desvantagem: alto consumo de CPU (no mínimo 7%)
- Código fonte disponível em www.kernel.org