# Aula 20 – Sistema de Arquivos

Norton Trevisan Roman
Clodoaldo Aparecido de Moraes Lima

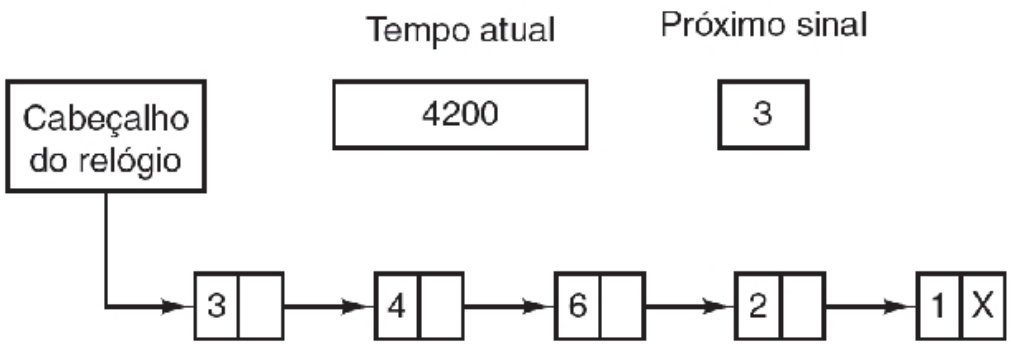
4 de dezembro de 2014

# Clocks (Timers) – Software

- Alarmes
  - Em alguns sistemas, processos podem solicitar "avisos" após um certo intervalo
    - Ex: rede, caso em que pacotes não confirmados podem ter que ser reenviados
  - Avisos podem ser: um sinal, uma interrupção ou uma mensagem
  - Se o driver gerenciar clocks suficientes, basta determinar um clock separado para cada requisição
  - Se não tiver, terá que simular clocks virtuais múltiplos com um único clock físico:
    - Manter uma lista encadeada com os tempos dos alarmes pendentes, ordenada pelo tempo
    - Cada item na lista diz quantos tiques do clock, após o tique anterior, deve-se esperar antes de se enviar o sinal

# Clocks (Timers) – Software

- Alarmes
  - Ex: Sinais esperados em 4203, 4207, 4213, 4215 e 4216




A cada tique, "Próximo sinal" é decrementado (assim como sua entrada na lista).

Quando chega a 0, o sinal correspondendo ao primeiro item da lista é emitido.

Este item é então removido da lista

"Próximo sinal" recebe o valor do começo da lista $\rightarrow$ 4

# Clocks (Timers) – Software

- Temporizadores guardiões (watchdog timer):
  - Partes do SO também precisam de temporizadores
  - Ex: acionador de disco: somente quando o disco está em rotação na velocidade ideal é que as operações de E/S podem ser iniciadas
    - Ao receber uma requisição, o driver do dispositivo inicia o motor, e então define um temporizador guardião para causar uma interrupção após um certo intervalo
  - O mecanismo usado pelo driver de relógio para tratar desse tipo de temporizador é o mesmo usado em alarmes
    - Quando um temporizador dispara, contudo, em vez de causar um sinal, o driver chama um procedimento fornecido pelo requisitante

# Clocks (Timers) – Software

- Tarefas básicas do driver de relógio (clock driver) durante uma interrupção de relógio:
  - Incrementar o tempo real do processo
  - Decrementar o quantum e comparar com 0 (zero)
  - Contabilizar o uso da CPU
  - Decrementar o contador do alarme
  - Gerenciar o tempo de acionamento de dispositivos de E/S (via temporizadores guardiões)

# Sistema de Arquivos

# Sistema de Arquivos

- Três requisitos são essenciais no armazenamento de informações de longo prazo:
  - 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 devem poder acessar as informações de forma concorrente:
    - Informações devem ser independentes de processos

# Sistema de Arquivos

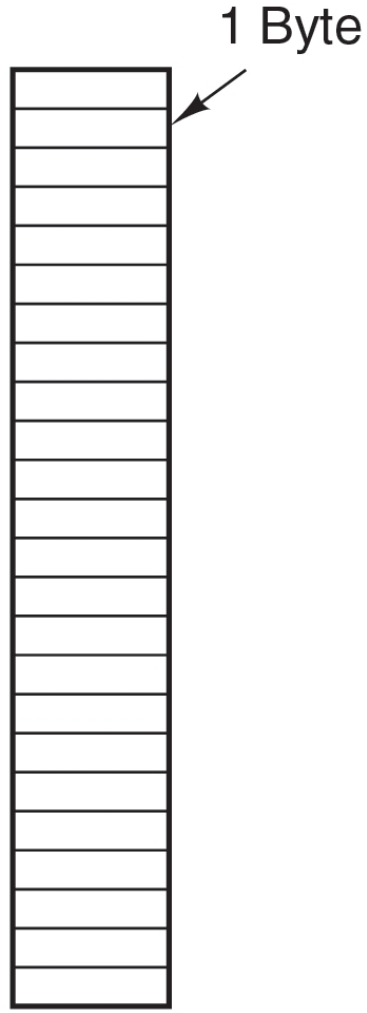
- Para atender a esses requisitos, informações são armazenadas em discos (ou alguma outra mídia de armazenamento), em unidades chamadas arquivos
  - Abstrações, unidades lógicas de informação criadas pelos processos
  - Processos podem ler ou escrever em arquivos, ou ainda criar novos arquivos
  - Informações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

# Sistema de Arquivos

- Arquivos são gerenciados pelo SO
  - Como são estruturados, nomeados, acessados, usados, protegidos e implementados
  - Manipulados por meio de chamadas (system calls) ao Sistema Operacional
- Sistema de Arquivos:
  - Parte do SO responsável por tratar dos arquivos
  - É a parte do SO mais visível ao usuário

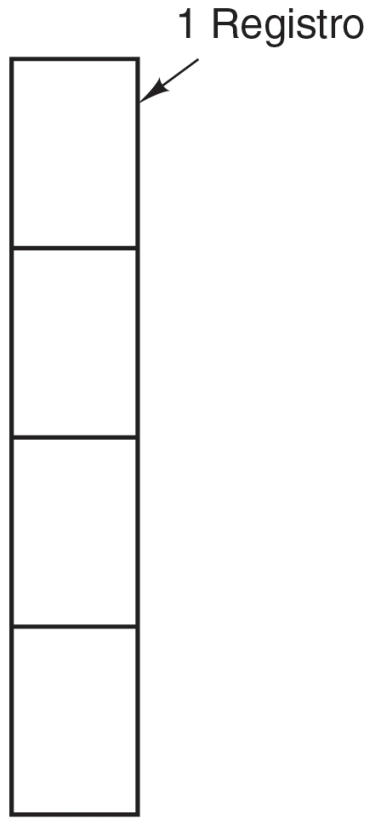
# Estrutura de Arquivos

- Arquivos podem ser estruturados de diferentes maneiras:
  - Sequência não estruturada de bytes
    - Para o SO arquivos são apenas conjuntos de bytes
    - SO não se importa com o conteúdo do arquivo $\rightarrow$ significado deve ser atribuído pelos programas em nível de usuário (aplicativos)
    - Vantagem: Flexibilidade – os usuários usam seus arquivos como quiserem
    - Ex.: UNIX, Linux, DOS e Windows



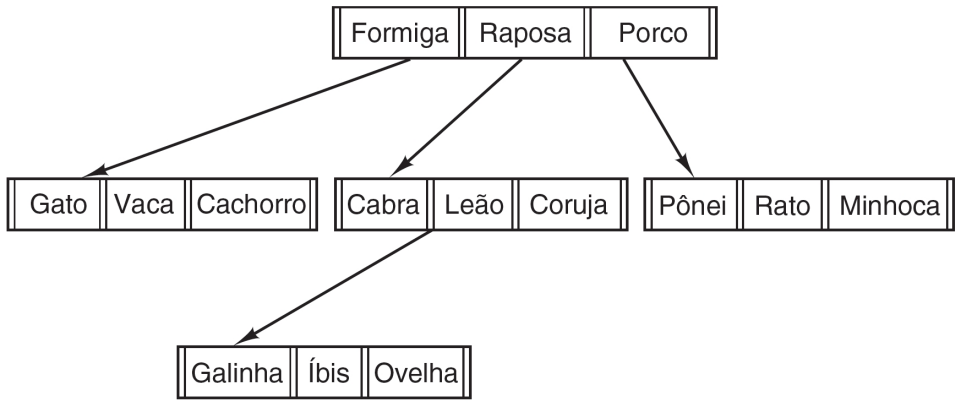
# Estrutura de Arquivos

- Arquivos podem ser estruturados de diferentes maneiras:
  - Sequência de registros de tamanho fixo
    - Cada qual com alguma estrutura interna
    - Leitura/escrita são realizadas em registros $\rightarrow$ lê, sobrescreve ou adiciona um registro
    - SOs mais antigos (mainframes) $\rightarrow$ Espelhavam os 80 caracteres do cartão perfurado
    - Nenhum sistema atual utiliza esse esquema



# Sistema de Arquivos

- Arquivos podem ser estruturados de diferentes maneiras:
  - Árvores de registros (tamanho variado)
    - Cada qual com um campo chave em uma posição fixa
    - O arquivo consiste nessa árvore → a operação básica é obter o registro com uma certa chave
    - SO decide onde colocar novos registros, não o usuário
    - Usado em mainframes atuais



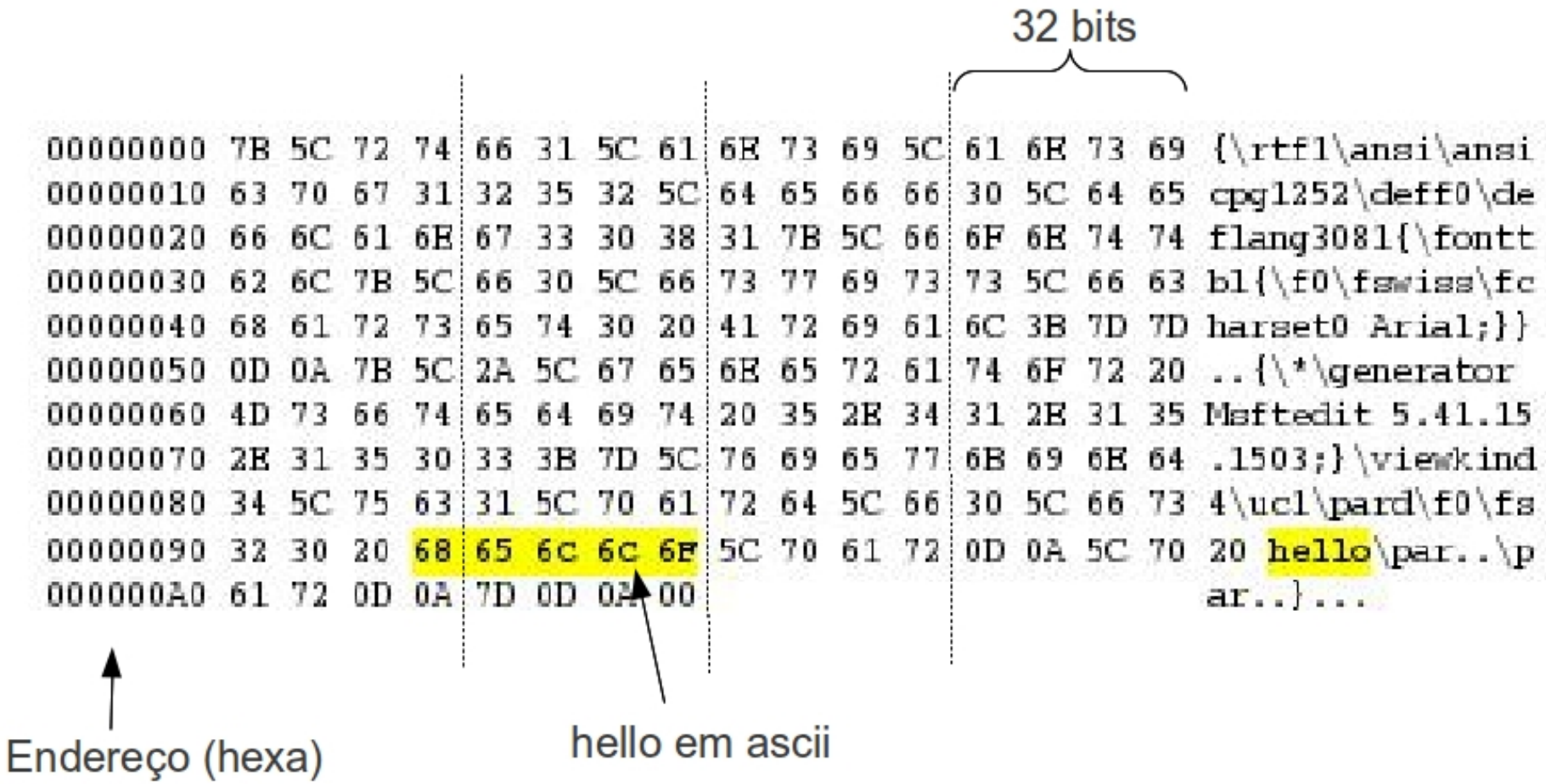
# Tipos de Arquivos

- Arquivos regulares $\rightarrow$ contêm informação do usuário
- Diretórios $\rightarrow$ arquivos responsáveis por manter a estrutura do Sistema de Arquivos
- Arquivos especiais de caracteres (Unix) $\rightarrow$ relacionados com E/S
  - Usados para modelar dispositivos seriais de E/S
  - Ex.: impressora, interface de rede, terminais
- Arquivos especiais de bloco (Unix) $\rightarrow$ usados para modelar discos

# Tipos de Arquivos

- Arquivos regulares podem ser de dois tipos:
  - ASCII:
    - Consistem de linhas de texto (terminadas em CR, LF ou CR+LF)
    - Facilitam integração de programas (via arquivo)
    - 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 apenas pelos aplicativos que os usam
    - Ex.: programa executável

# Tipos de Arquivos: Binários



# Acesso a Arquivos

- SOs mais antigos ofereciam apenas acesso sequencial no disco
  - Leitura em ordem byte a byte (registro a registro)
- SOs mais modernos fazem acesso aleatório;
  - Acesso feito fora de ordem, por chave ou posição
  - Métodos usados para especificar onde iniciar leitura:
    - Cada operação read indica a posição no arquivo onde se inicia a leitura
    - Operação Seek $\rightarrow$ estabelece a posição atual, após a qual o arquivo pode ser lido sequencialmente (usado em Unix, Linux e Windows)

# Atributos de Arquivos

- Além do nome e dos dados, todo arquivo tem outras informações associadas a ele
  - Atributos (ou metadados)
- A lista de atributos varia de SO para SO
- Nenhum SO implementa a lista toda (adiante), mas ela toda está em algum SO

# 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 |
| *Flag* de leitura | 0 para leitura/escrita; 1 somente para leitura |
| *Flag* de oculto | 0 para normal; 1 para não aparecer em listagens |
| *Flag* de sistema | 0 para arquivos normais; 1 para arquivos do sistema |
| *Flag* de repositório | 0 para "tem *backup*"; 1 para arquivos "precisa de *backup*" |

# 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 aleatório |
| *Flag* de temporário | 0 para normal; 1 para temporário (apagado quando o processo termina |
| *Flag* de travamento | 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 (*key*) |

# 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 atual | Número de bytes do arquivo |
| Tamanho Máximo | Número máximo de bytes que o arquivo pode ter |

# Operações com Arquivos

- Diferentes sistemas provêm diferentes operações para armazenar e recuperar informações
- Operações mais comuns (system calls):
  - Create: o arquivo é criado sem dados
  - Delete: o arquivo é removido do disco
  - Open: permite que o SO busque os atributos e lista dos endereços de disco e coloque na memória
  - Close: libera o espaço ocupado por Open. Também força que o último bloco de dados seja escrito no disco
  - Read: lê do arquivo para um buffer

# Operações com Arquivos

- Operações mais comuns (system calls):
  - Write: escreve dados no arquivo
  - Append: escreve dados ao final do arquivo
  - Seek: para acesso aleatório – especifica onde estão os dados, reposicionando o ponteiro de arquivo
  - Get attributes: permite que um programa olhe os atributos de um arquivo
  - Set attributes: permite que se mude alguns dos atributos do arquivo
  - Rename: muda o nome do arquivo

# Diretórios (Pastas)

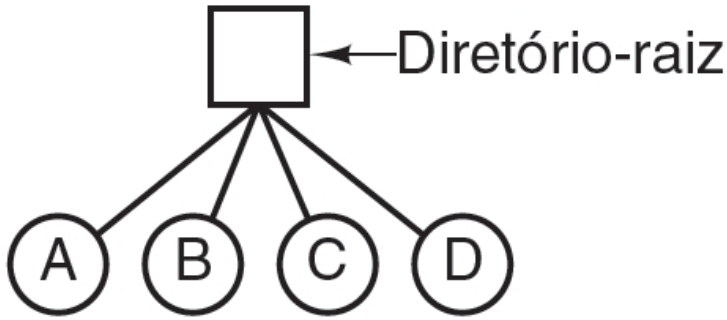
- São arquivos responsáveis por manter a estrutura do Sistema de Arquivos
  - Usados como um modo de agrupar arquivos
  - Esse modelo dá origem a uma hierarquia – o sistema de arquivos
- Podem ser organizados da seguinte maneira:
  - Nível único (Single-level)
  - Dois níveis (Two-level)
  - Hierárquica

# Diretórios: Organização de Nível Único

- Apenas um diretório contém todos os arquivos
  - Chamado de diretório raiz (root directory)
  - Contudo, é o único
- Computadores antigos utilizavam esse método, pois eram monousuário
  - Exceção: CDC 6600 $\rightarrow$ supercomputador que utilizava-se desse método, apesar de ser multiusuário
- Vantagens:
  - Simplicidade
  - Capacidade de localizar arquivos rapidamente

# Diretórios: Organização de Nível Único

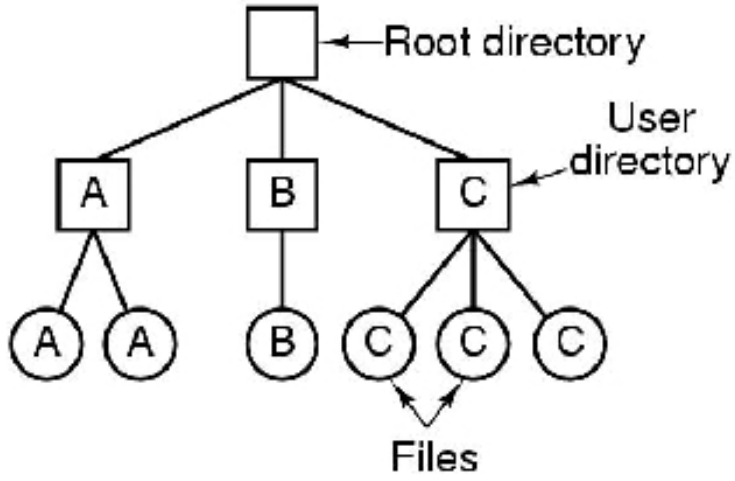
- Ex:
  - 04 arquivos
  - Três diferentes proprietários (A, B e C)




- Desvantagens:
  - Sistemas multiusuários: Diferentes usuários podem criar arquivos com o mesmo nome
  - Exemplo:
    - Usuários A e B criam, respectivamente, um arquivo mailbox
    - Usuário B sobrescreve arquivo do usuário A

# Diretórios: Organização de Dois Níveis

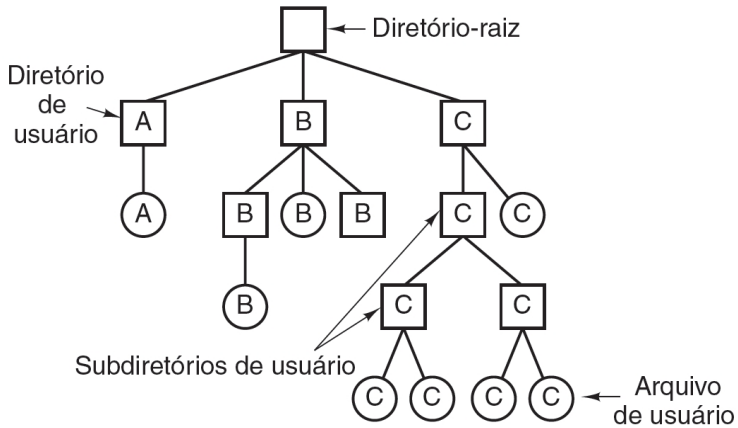
- Cada usuário possui um diretório privado
  - Sem conflitos de nomes de arquivos
  - Quando o usuário tenta abrir um arquivo, o sistema sabe qual é o usuário $\rightarrow$ sabe o diretório
  - Necessita de procedimento de login: identificação
  - O usuário somente tem acesso a seus arquivos




- Compartilhamento de arquivos
  - Programas executáveis em um diretório de sistema (acessível a todos)
- Desvantagem: Usuário com muitos arquivos

# Diretórios: Organização Hierárquica

- Hierarquia de diretórios → árvores de diretórios
  - Usuários podem querer agrupar seus arquivos de maneira lógica, criando diversos diretórios que agrupam arquivos
    - Pode haver qualquer número de diretórios e sub-diretórios
  - Sistemas operacionais modernos utilizam esse método
  - Vantagem: Flexibilidade



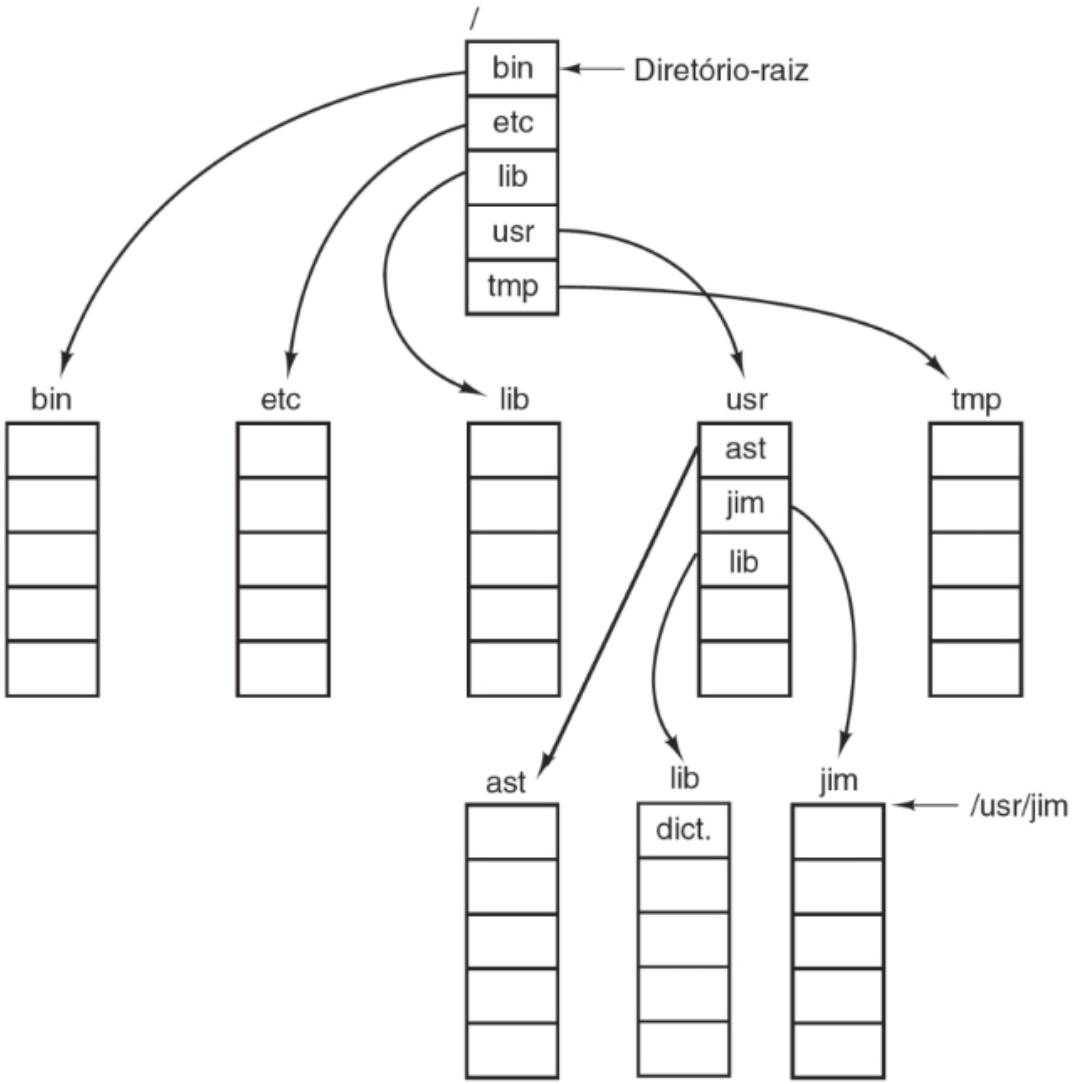
# Diretórios: Caminhos

- Conceitos:
  - Diretório raiz: o diretório mais alto na hierarquia
  - Diretório de Trabalho (working directory) ou diretório corrente (current directory): diretório sendo examinado pelo processo
- A organização hierárquica requer diferentes métodos pelos quais os arquivos são acessados:
  - Caminho absoluto (absolute path name)
  - Caminho relativo (relative path name)

# Diretórios: 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
  - Se o primeiro caractere do caminho for o separador ('\' ou '/', nos exemplos), o caminho é absoluto

# Diretórios: Caminho Absoluto



# Diretórios: Caminho Relativo

- É utilizado em conjunto com o diretório corrente
- Usuário estabelece um diretório como sendo o diretório de trabalho (ou corrente)
  - Nesse caso, caminhos não iniciados no diretório raiz são tido como relativos ao diretório corrente
  - Mais conveniente $\rightarrow$ elimina a necessidade de se conhecer o diretório corrente
- Exemplo:
  - cp /usr/ast/mailbox /usr/ast/mailbox.bak
  - Diretório corrente: /usr/ast $\rightarrow$ cp mailbox mailbox.bak

# Diretórios: Caminho Relativo

- "." $\rightarrow$ diretório corrente
- ".." $\rightarrow$ diretório pai (anterior ao corrente)
  - Ambos são entradas no arquivo que representa o diretório corrente
- Exemplo:
  - Diretório corrente /usr/ast
  - cp ../lib/dictionary .
  - cp /usr/lib/dictionary .
  - cp /usr/lib/dictionary dictionary
  - cp /usr/lib/dictionary /usr/ast/dictionary

# Diretórios: Operações

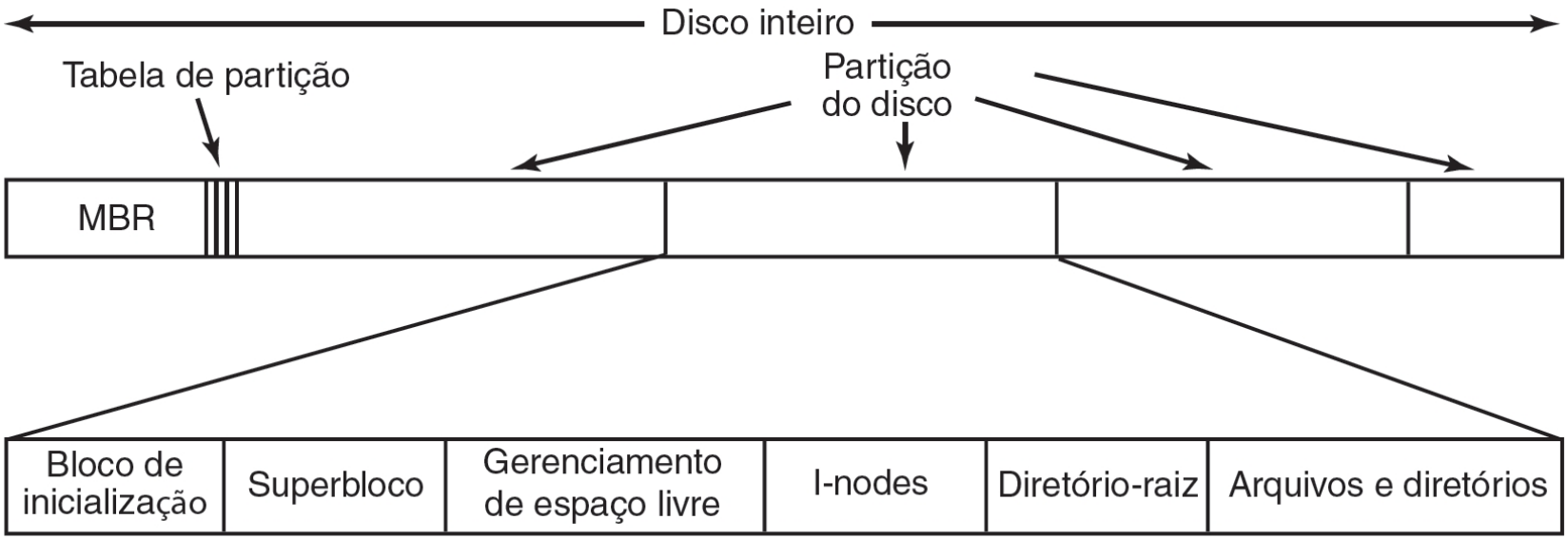
- Unix:
  - Create: cria diretório, contendo apenas "." e ".."
    - Considerado vazio pelo sistema
  - Delete: remove um diretório (desde que vazio)
  - Opendir: o diretório pode ser lido (listado)
  - Closedir: usado para remover espaço em RAM, após usar o diretório
  - Readdir: retorna a próxima entrada em um diretório aberto
  - Rename: muda o nome do diretório

# Diretórios: Operações (Unix)

- Link: permite que um arquivo apareça em mais de um diretório
  - Veremos mais adiante
- Unlink:
  - Remove uma entrada (arquivo) no diretório, decrementando o contador de links do arquivo
  - Veremos mais adiante

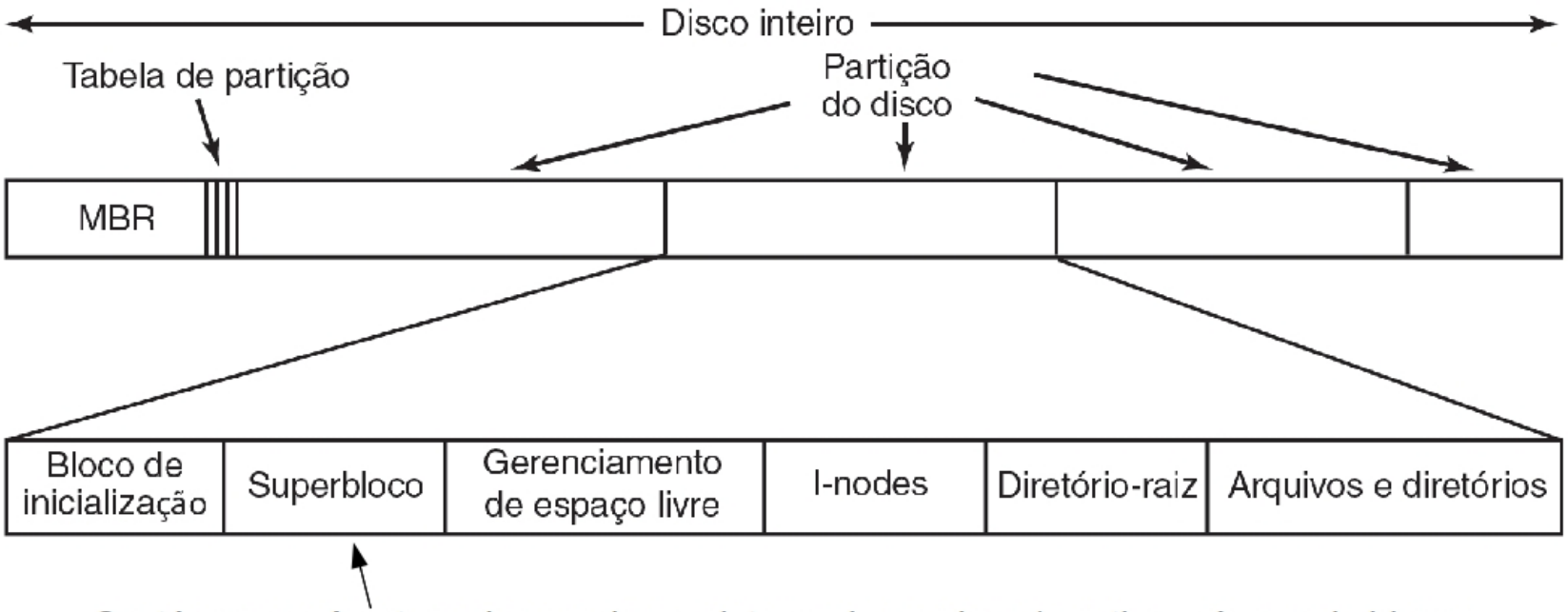
# Implementação de Sistemas de Arquivos

- Layout do sistema de arquivos:
  - O layout varia de sistema para sistema de arquivos
  - Frequentemente, conterá:



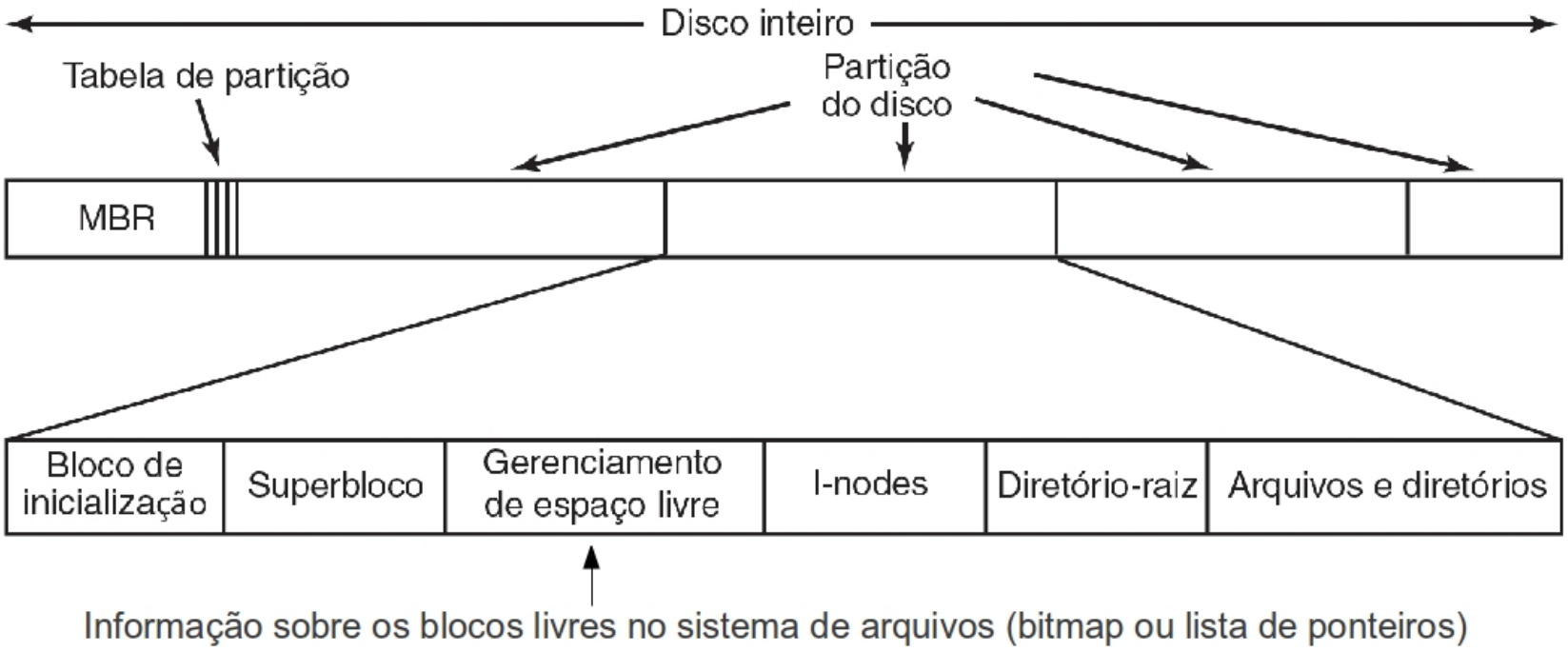
# Implementação de Sistemas de Arquivos

- Layout do sistema de arquivos:



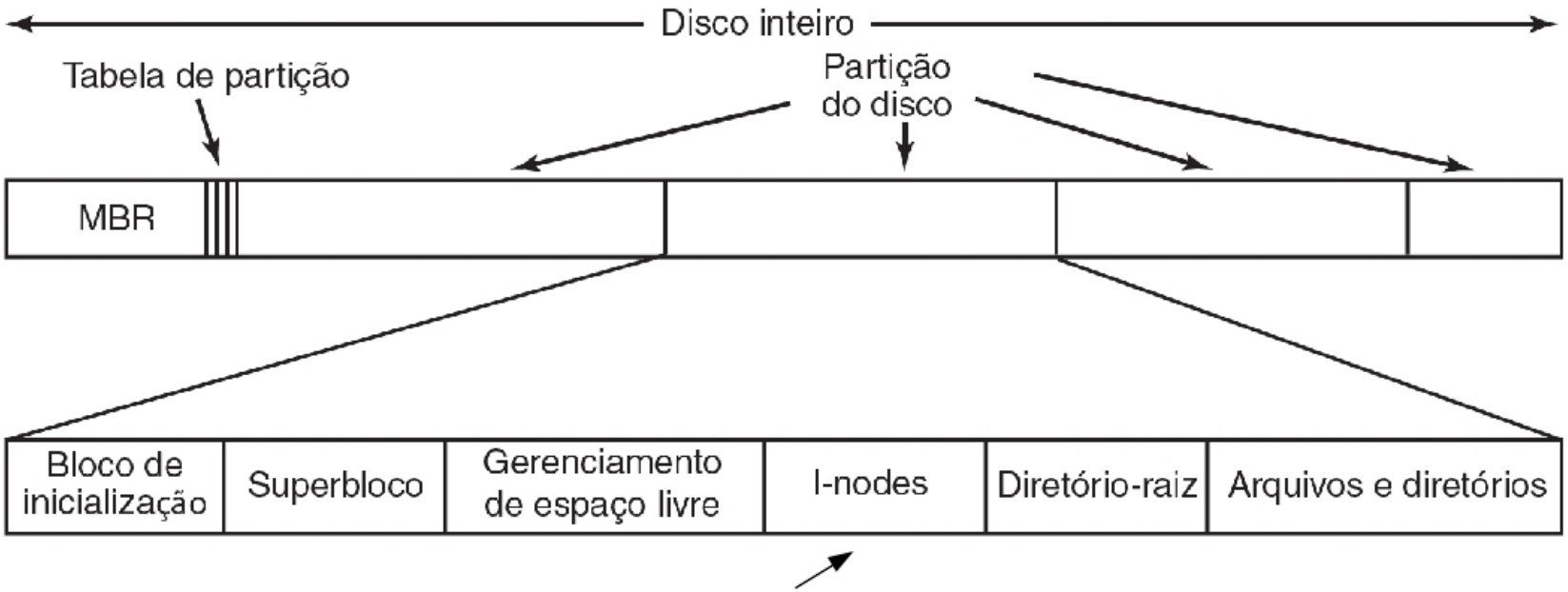
# Implementação de Sistemas de Arquivos

- Layout do sistema de arquivos:



# Implementação de Sistemas de Arquivos

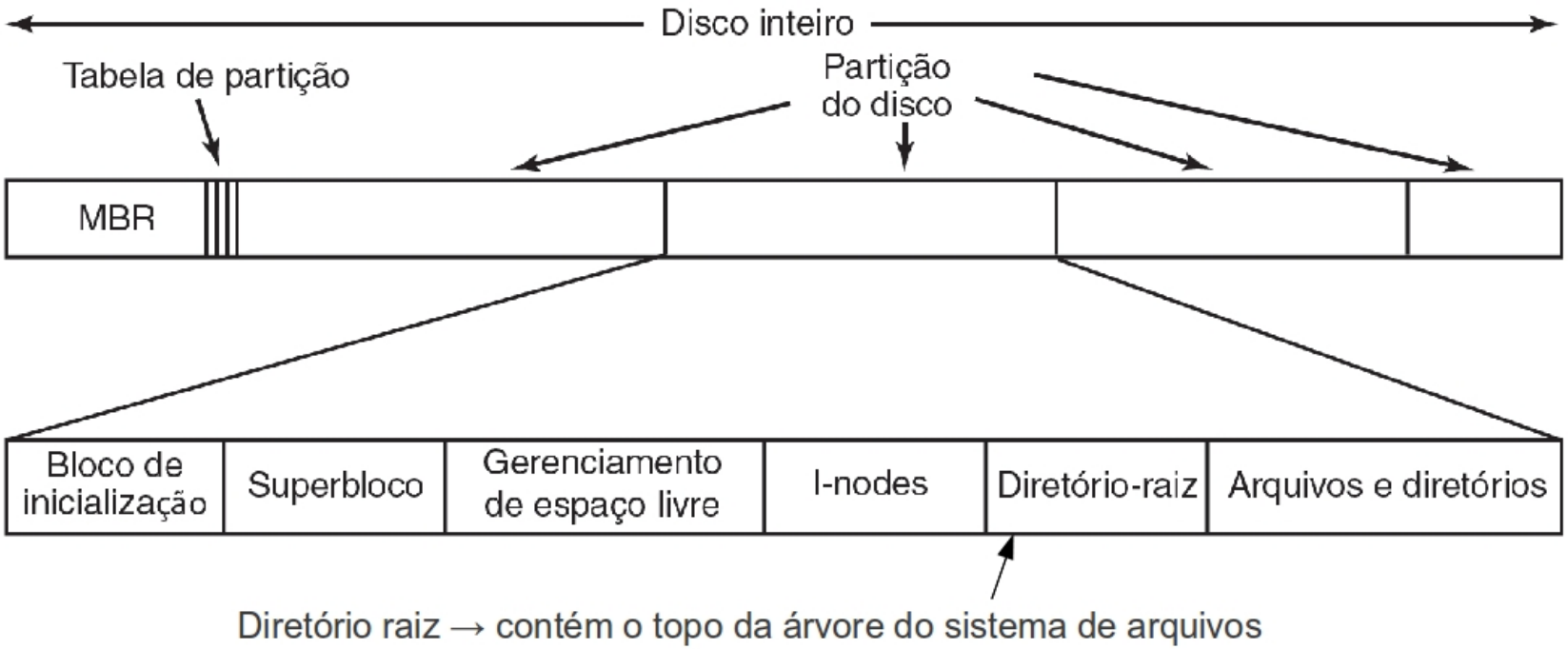
- Layout do sistema de arquivos:




Arranjo de estruturas de dados, uma por arquivo, contendo informações sobre o arquivo (Unix/Linux)

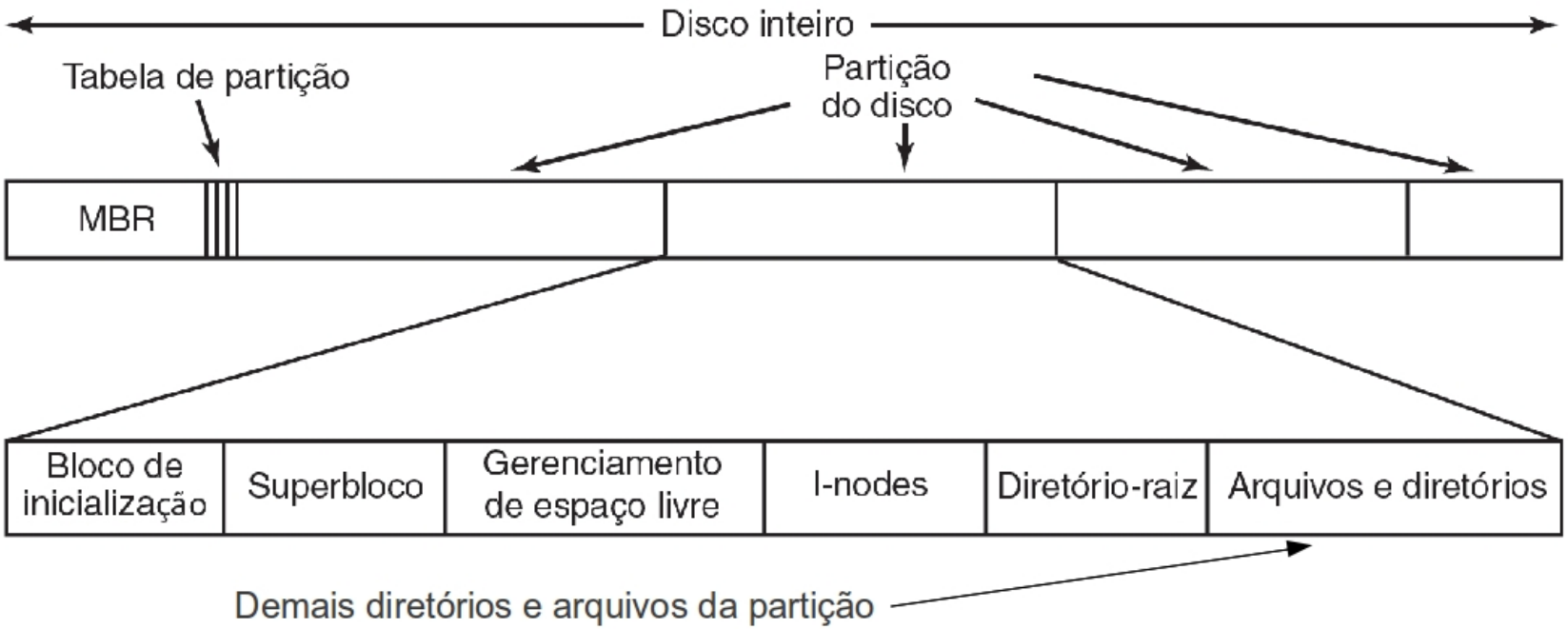
# Implementação de Sistemas de Arquivos

- Layout do sistema de arquivos:



# Implementação de Sistemas de Arquivos

- Layout do sistema de arquivos:

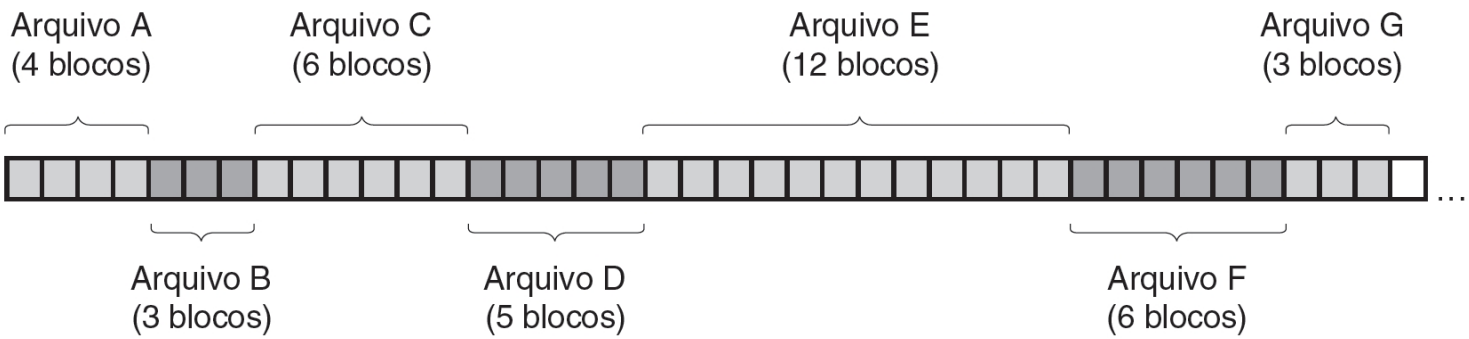


# Implementação de Sistemas de Arquivos

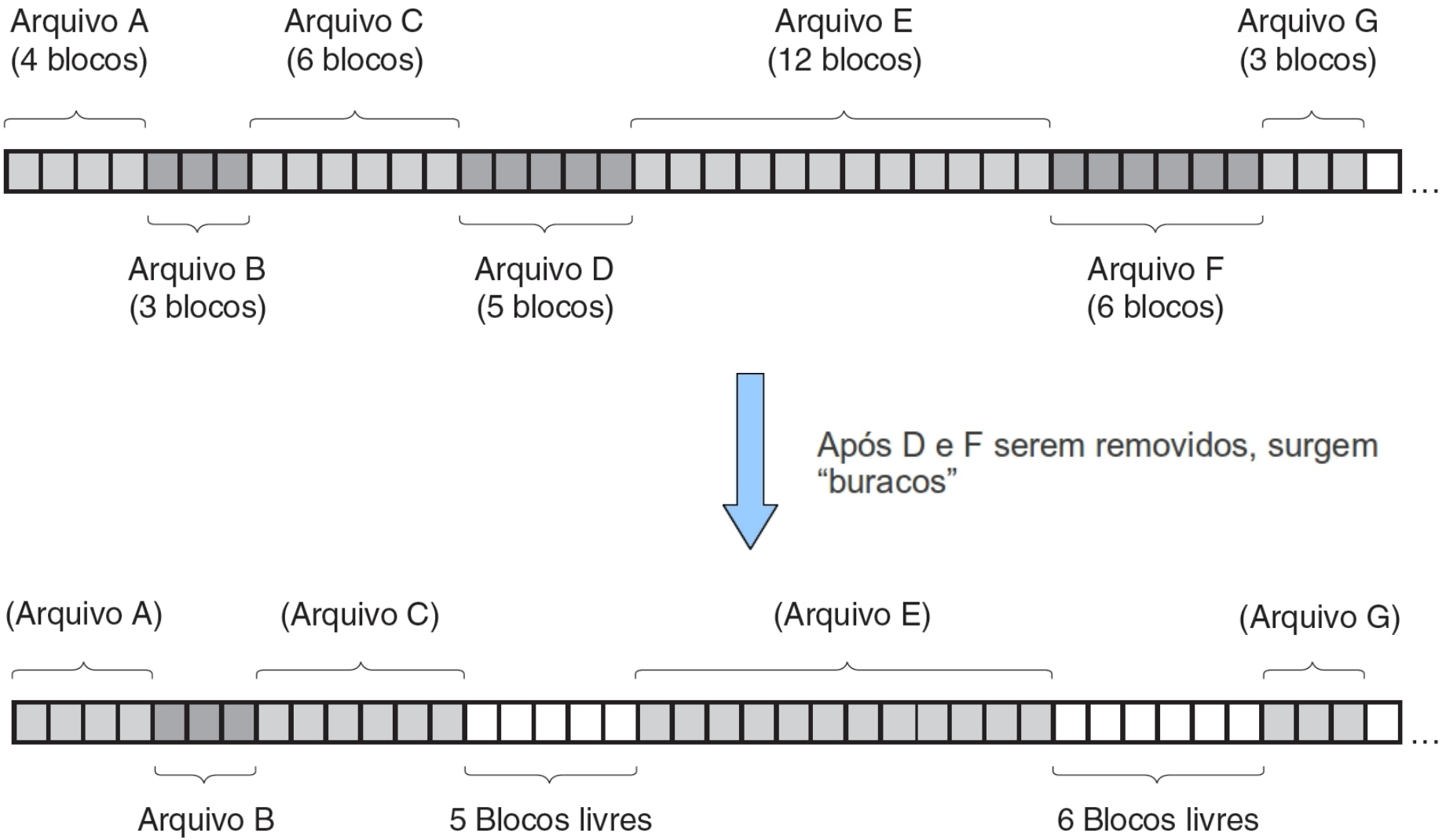
- Como arquivos são alocados no disco?
- Diferentes técnicas são implementas por diferentes Sistemas Operacionais
  - Alocação contígua
  - Alocação com lista encadeada
  - Alocação com lista encadeada utilizando uma tabela na memória (FAT)
  - I-Nodes

# Alocação Contígua

- Técnica mais simples
- Armazena arquivos em blocos contíguos no disco;
  - Ex.: em um disco com blocos de 1kB um arquivo com 50kB será alocado em 50 blocos consecutivos
  - Usado inicialmente nos HDs, e agora em CD-ROMs



# Alocação Contígua: Desvantagens

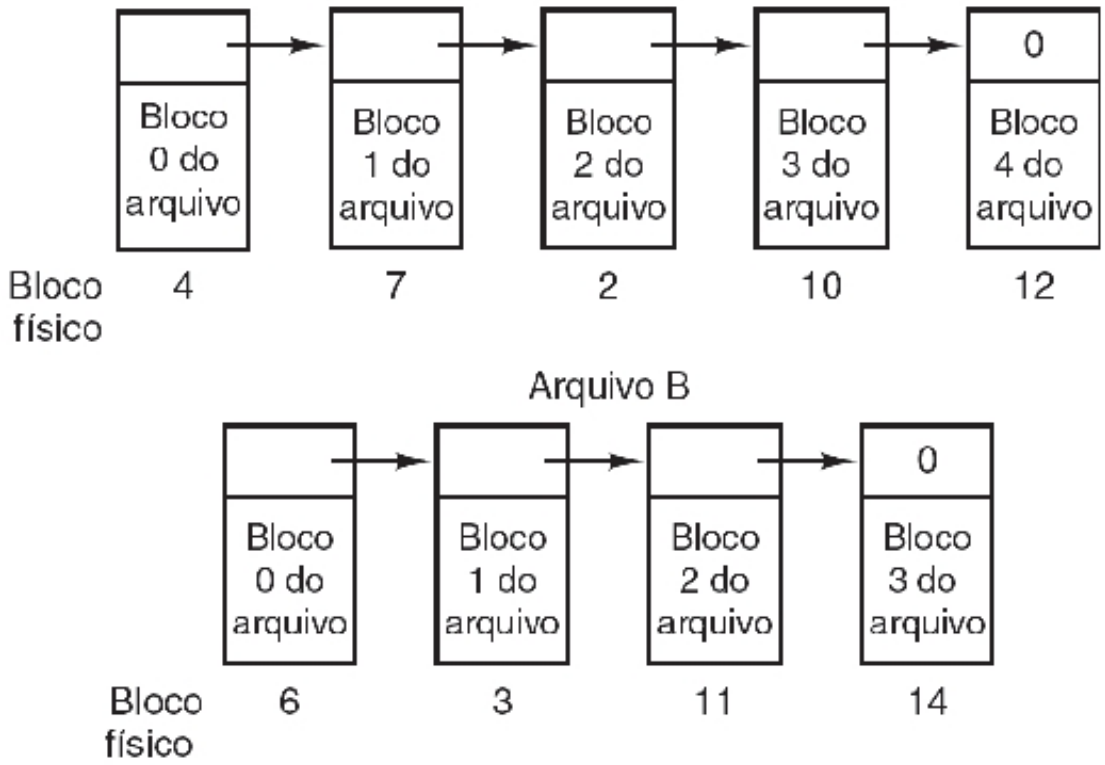


# Alocação Contígua

- Desvantagens:
  - Ao longo do tempo, o disco se torna fragmentado
  - Compactação $\rightarrow$ alto custo
  - Reuso de espaço $\rightarrow$ atualização da lista de espaços livres
    - Requer conhecimento prévio do tamanho final de novos arquivos para alocar lacunas adequadas
- Vantagens:
  - Simples de implementar: basta saber o endereço do primeiro bloco e o número de blocos no arquivo para saber onde está cada bloco
  - Alto desempenho na leitura – um seek (para o primeiro bloco), e depois segue o movimento natural

# Alocação com Lista Encadeada

- Cada arquivo é uma lista ligada de blocos do disco
  - A primeira palavra de cada bloco é um ponteiro para o bloco seguinte
  - O restante do bloco é destinado aos dados



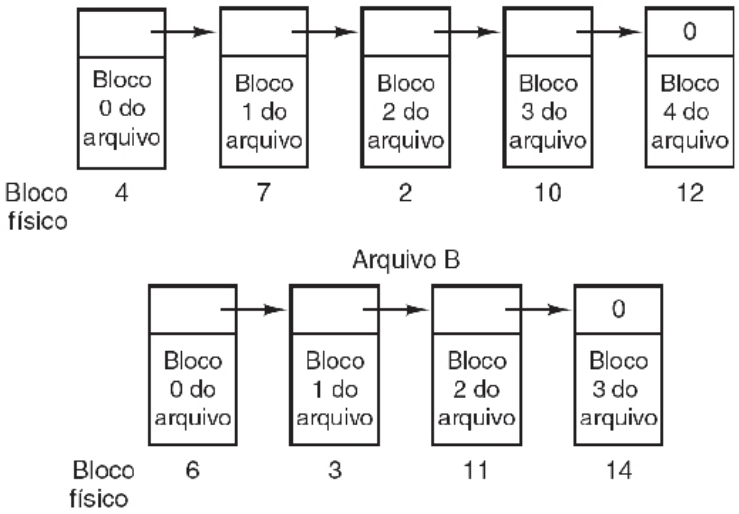
# Alocação com Lista Encadeada

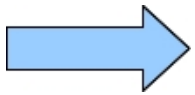
- Vantagem:
  - Não se perde espaço com a fragmentação externa
  - Uma entrada no diretório precisa apenas armazenar o endereço em disco do primeiro bloco
- Desvantagens:
  - Acesso aos arquivos é feito sequencialmente
    - Acesso aleatório lento $\rightarrow$ deve-se ler todos os blocos em sequência
  - A quantidade de informação armazenada em um bloco não é mais uma potência de dois
    - Os ponteiros ocupam alguns bytes do bloco – bloco incompatível com o assumido pelos programas

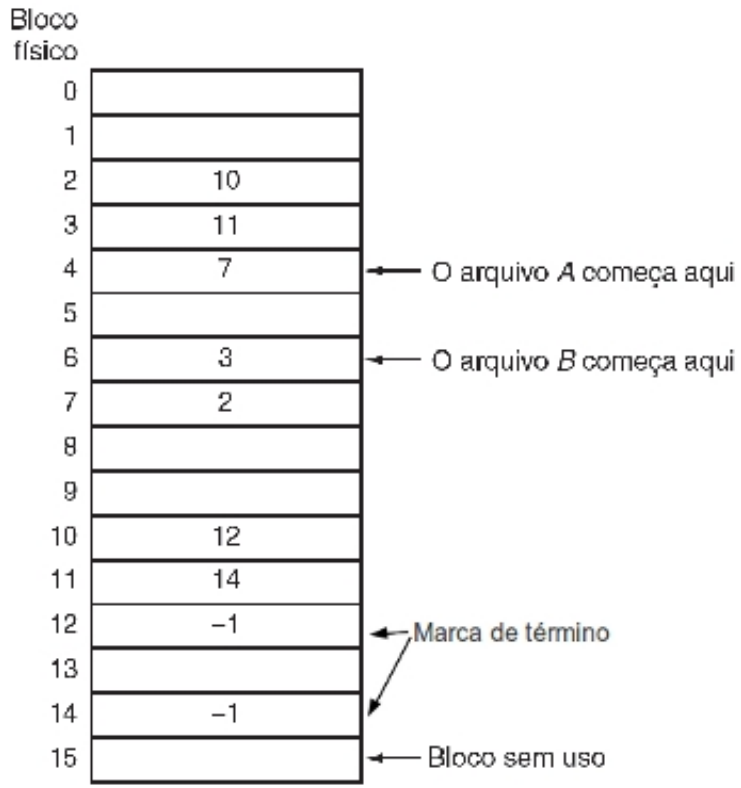
# Lista Encadeada Com Tabela na Memória

- Solução: colocar o ponteiro em uma tabela na memória em vez de estar no próprio bloco
  - FAT $\rightarrow$ Tabela de alocação de arquivos (File Allocation Table)
  - O bloco inteiro está disponível para os dados (não guarda mais o ponteiro)
  - Toda a cadeia pertencente a um arquivo está na memória $\rightarrow$ embora ainda tenha que seguir a cadeia, o acesso aleatório fica mais rápido
  - Entradas no diretório precisam somente armazenar o número (endereço em disco) do bloco inicial

# Lista Encadeada Com Tabela na Memória






O endereço do bloco é o índice da tabela. O valor nessa posição é o endereço/índice do próximo bloco no arquivo
A FAT é única no sistema

# Lista Encadeada Com Tabela na Memória

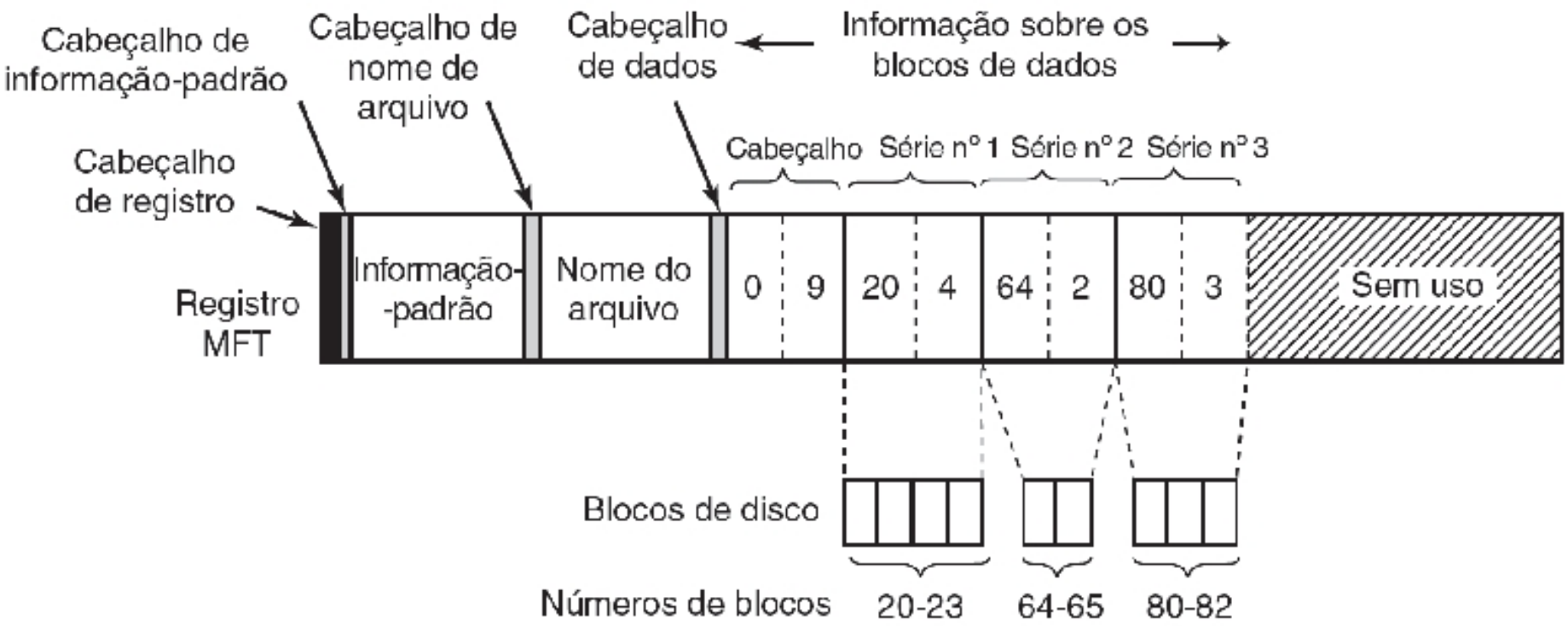
- SOs:
  - MS-DOS, Windows 9x
  - WinNT, Win2000 e WinXP $\rightarrow$ NTFS
    - Possui uma tabela, mas não funciona como a FAT
- Desvantagem:
  - Toda a tabela deve estar na memória o tempo todo
    - Com um disco de 200GB e blocos de 1KB, a tabela precisa de 200 milhões de entradas, cada qual com 4 bytes, ocupando 800 MB da memória o tempo todo (note que são blocos, não setores – cabe ao SO, via driver, mapear)
    - Inviável com discos grandes (no caso da Microsoft, limitada a $2^{32}$ setores, não blocos)

# Lista Encadeada Com Tabela na Memória

- NTFS:
  - Cada volume possui uma estrutura de dados – MFT (master file table)
    - Sequência linear de registros com tamanho fixo de 1KB
    - Cada registro descreve um arquivo ou diretório, contendo seus atributos e a lista de blocos em disco – a lista está na linha da tabela
  - Caso o arquivo seja muito grande, pode-se usar dois ou mais registros na MFT, para a lista de blocos
    - Nesse caso, o primeiro registro (registro-base) aponta para os demais.

# Lista Encadeada Com Tabela na Memória

- NTFS:
  - Registro na MFT (simplificado):




Qual o efeito da alta fragmentação e de desfragmentar o HD nisso?

# Referências Adicionais

- http://www.cyberciti.biz/tips/understanding-unixlinux-filesystem-superblock.html
- http://140.120.7.20/LinuxKernel/LinuxKernel/node17.html