Sistema de Arquivos
(Adaptado de Márcio Castro)

Prof. Martín Vigil

Na aula de hoje

- Proteção
- Implementação do sistema de arquivos
  - Arquivos
  - Diretórios
- Gerenciamento
- Network File System

PROTEÇÃO

Segurança

- Problema
  - Impedir que usuários não autorizados leiam ou alterem dados
- Uma das possíveis soluções
  - Controle de acesso

Requisitos do controle de acesso

- Distinguir o dono do arquivo/diretório entre os usuários
- Permitir o dono definir
  - Quem pode acessar seus arquivos
  - Qual tipo de acesso acesso permitir

Tipos de acesso

- Acesso de baixo nível
  - Ler
  - Escrever (modificar)
  - Executar
- Acesso de alto nível
  - Deletar (pode exigir escrever)
  - Copiar (depende de ler)

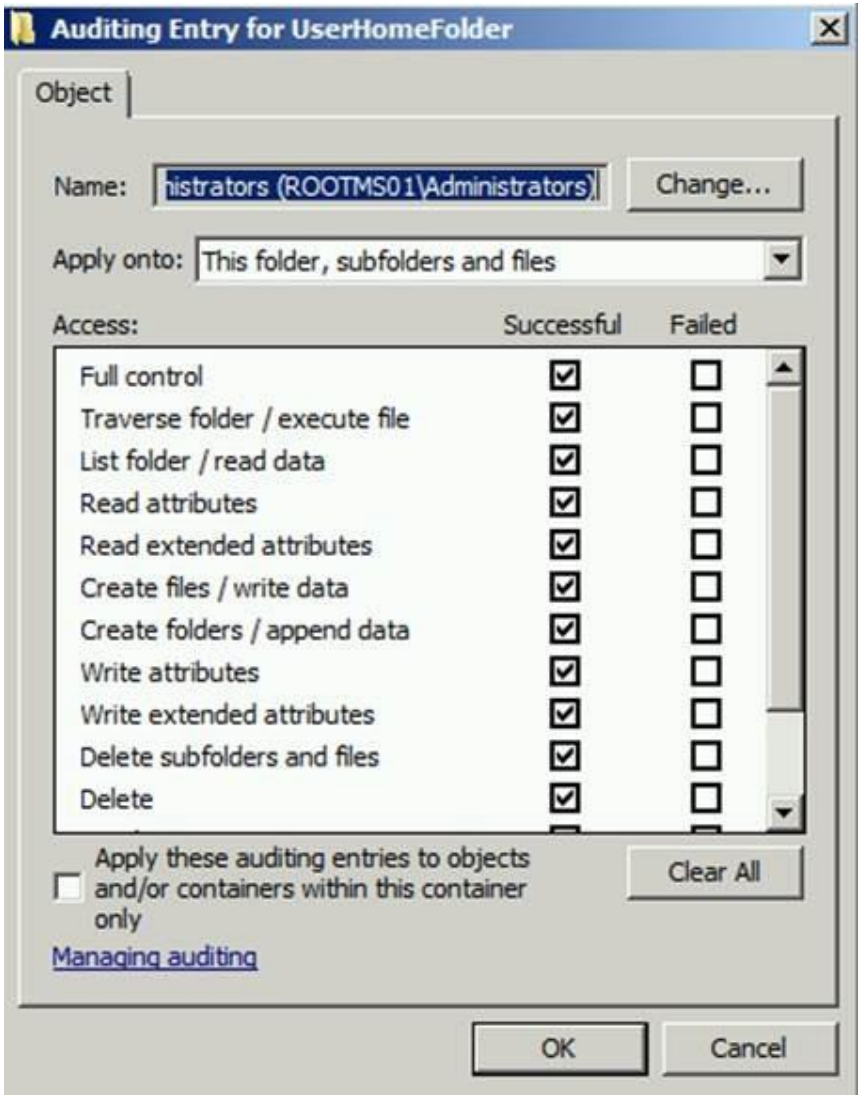
Implementação de controle de acesso

- Lista de controle de acesso (ACL):

Tabela =

arquivos x usuários x tipos de acesso

ACL no Windows 2008



Análise de ACLs

- Vantagem:
  - fina granularidade (consequentemente alta expressividade)
- Desvantagem:
  - manutenção exaustiva e propensa a erros
  - tamanho variável implica em dificuldade de armazenar ACL em diretórios

Simplificação de ACLs (sistemas Unix)

- Aumentar a granularidade (reduzir expressividade)
- Para cada arquivo, se identificam
  - Dono
  - Grupo (com quem se compartilha arquivo)
  - Outros (= Universo - {Dono} - Grupo)
  - Modos de acesso: **r**ead, **w**rite, e**x**ecute

Exemplo de ACL simplificada em sistemas Unix

1 = acesso permitido, 0 caso contrário

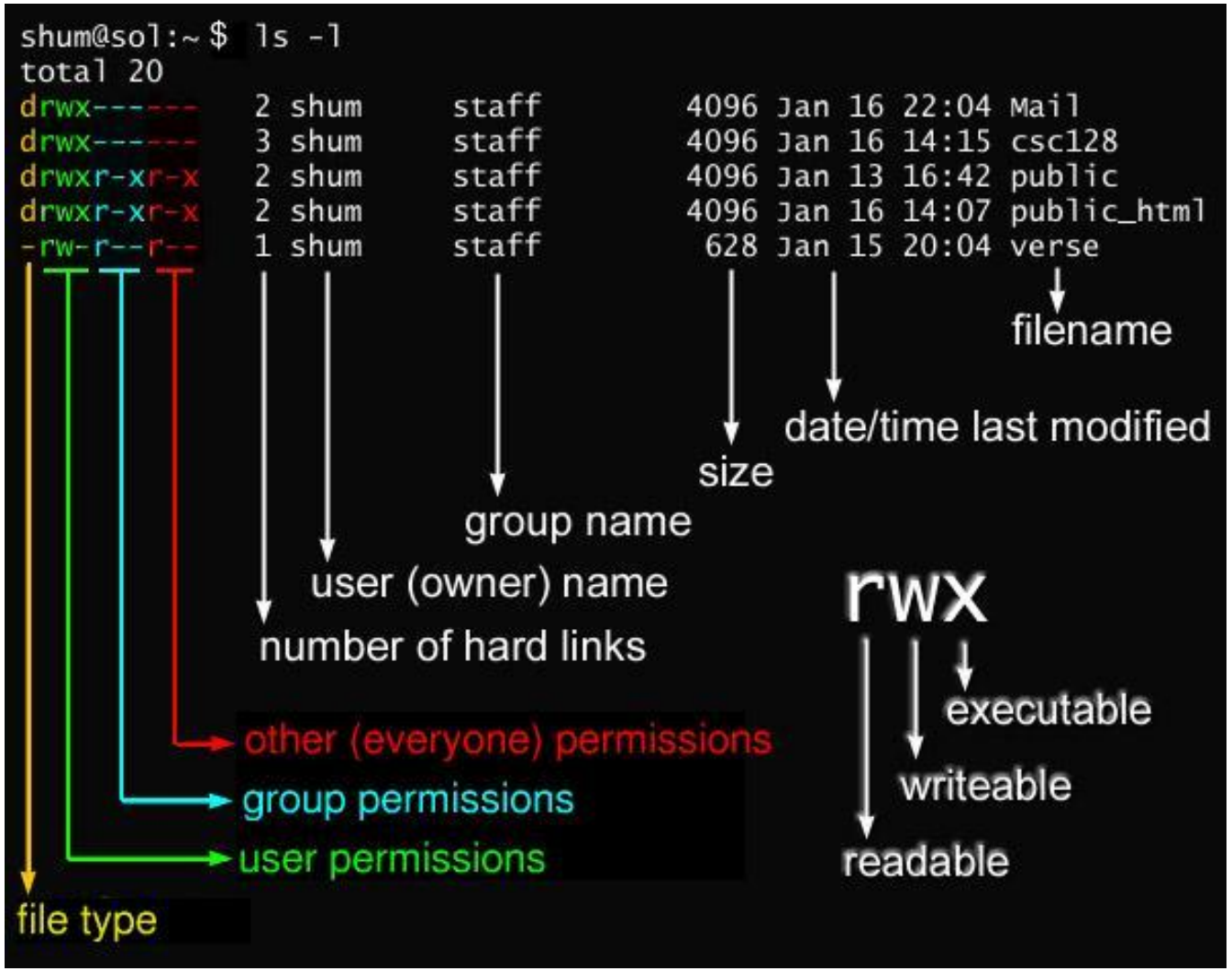
Usuário	Bit Read	Bit Write	Bit eXecute	RWX Decimal
Dono	1	1	1	**7**
Grupo	1	0	1	**5**
Outros	1	0	0	**4**

Comando: **chmod 754 nome_do_arquivo**

Exemplo de ACL simplificada em sistemas Unix

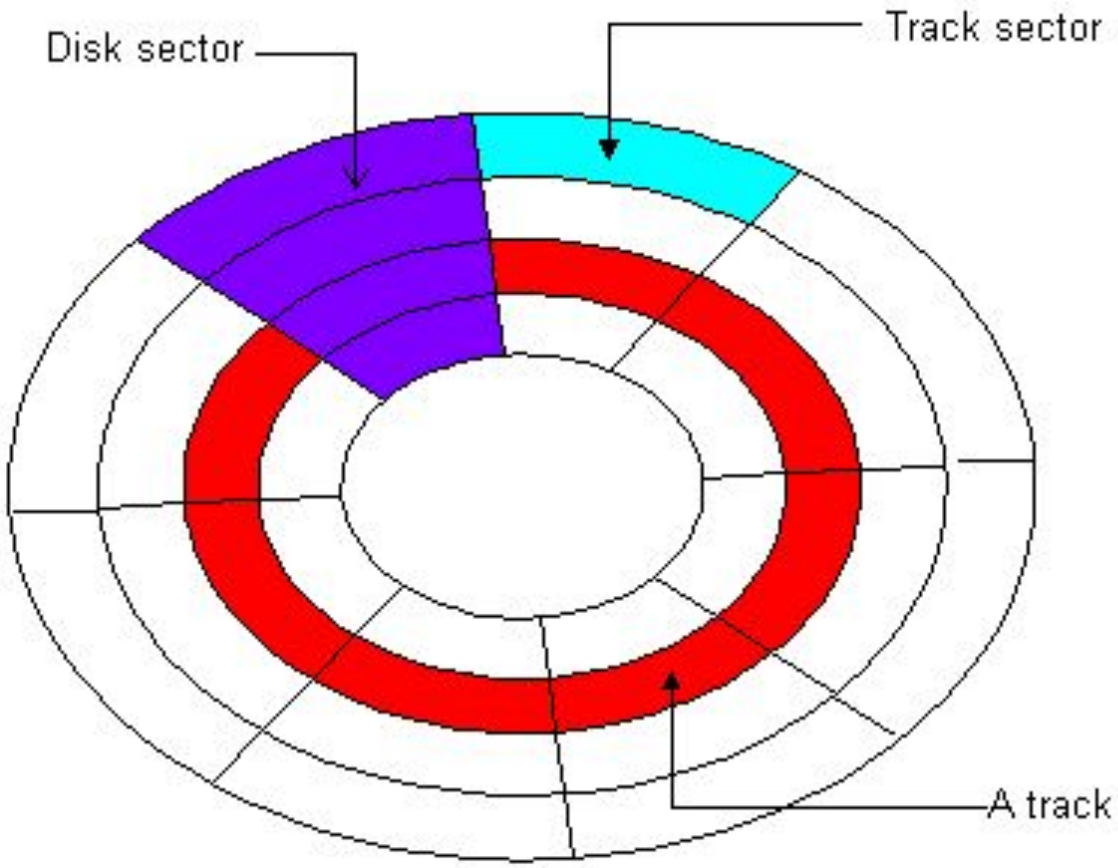
- Modificar dono e grupo:
  - **chown id_dono:id_grupo nome_do_arquivo**

Exemplo de ACL em sistemas Unix

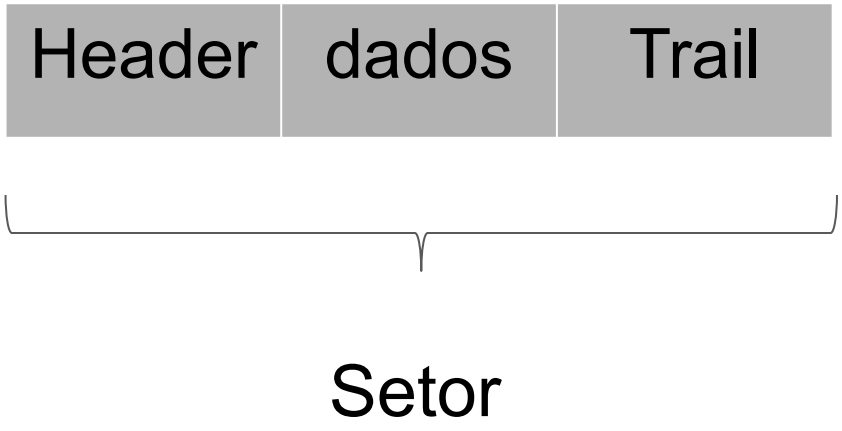


IMPLEMENTAÇÃO

Organização do disco



Setores



- Header e trail permitem verificar a **integridade** de dados
- Usam-se error-correcting codes (ECC)
- Pode-se recuperar **parcialmente** dados

Blocos

- Para melhor eficiência, transferências entre memória principal e disco são feitas em blocos

Implementação do sistema de arquivos

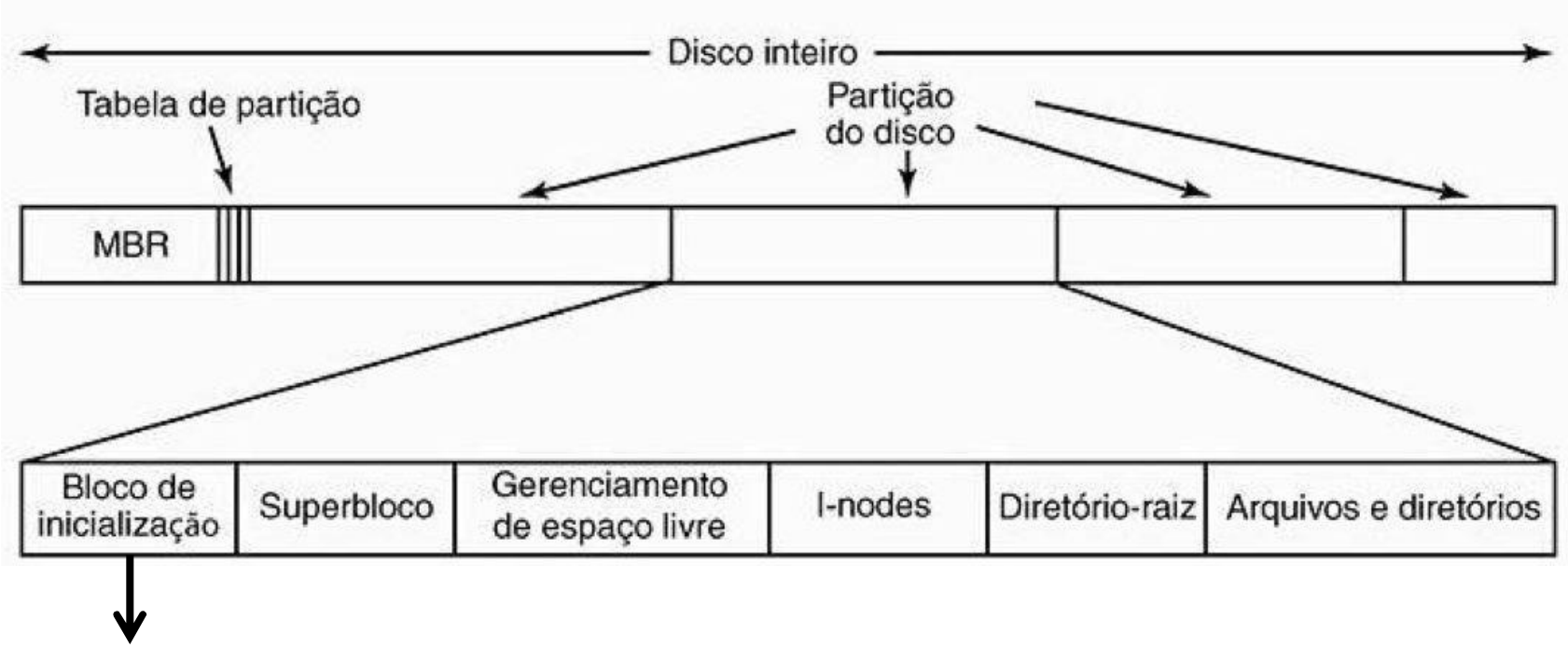
- Os sistemas de arquivos são armazenados em disco
- A maioria dos discos é dividida em uma ou mais **partições**
  - Cada partição possuirá um **sistema de arquivos independente das demais**
- O disco é dividido em **setores**
  - O setor 0 do disco é chamado de **Registro Mestre de Inicialização** (*Master Boot Record - MBR*) e é usado para inicializar o computador
  - O fim do MBR contém a tabela de partição, a qual armazena os endereços iniciais e finais de cada partição
  - Uma das partições na tabela é marcada como ativa

Implementação do sistema de arquivos

- O que acontece quando o computador é inicializado?
  1. A BIOS (*bootstrap*) lê e executa o MBR
  2. O programa do MBR localiza a partição ativa, lê o seu primeiro bloco (**bloco de inicialização**) e o executa
  3. O programa no bloco de inicialização carrega o S.O. contido naquela partição

Implementação do sistema de arquivos

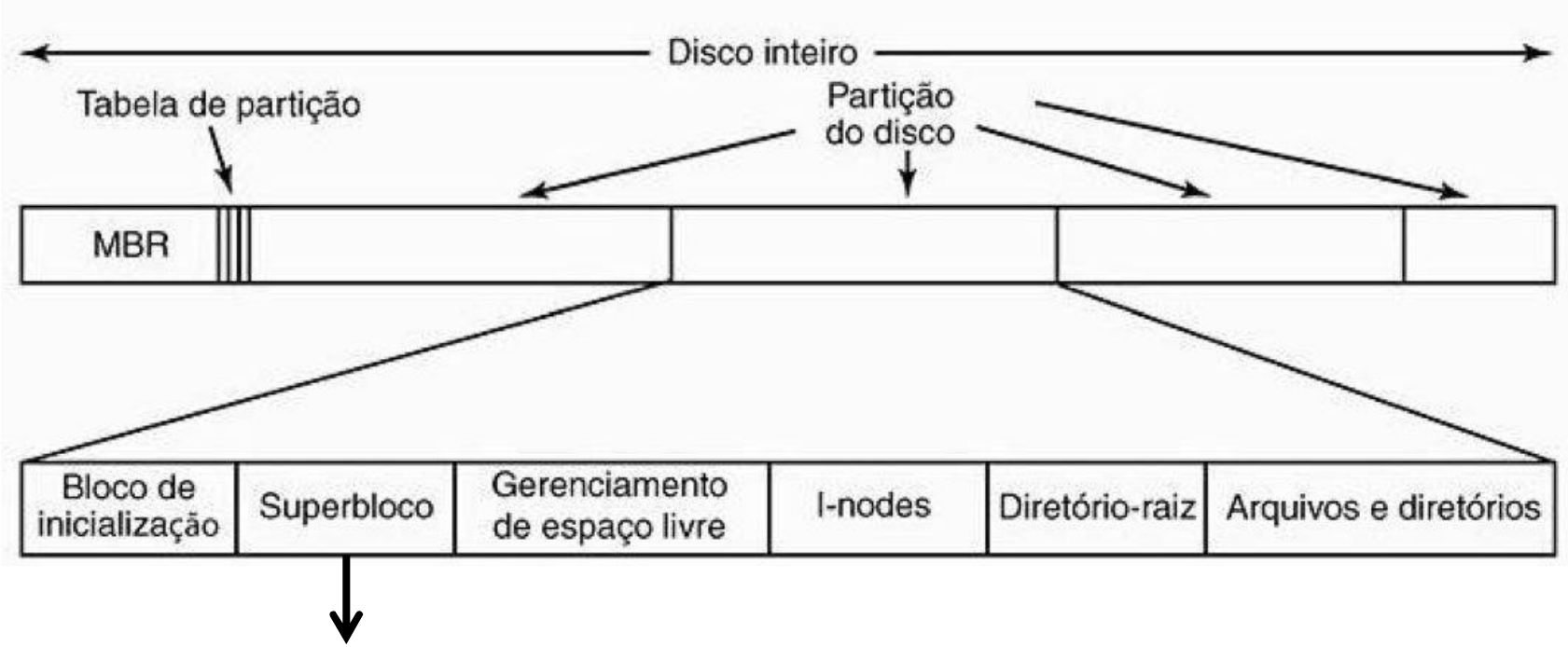
Exemplo de organização de um sistema de arquivos



Armazena o programa responsável pelo carregamento do S.O.

Implementação do sistema de arquivos

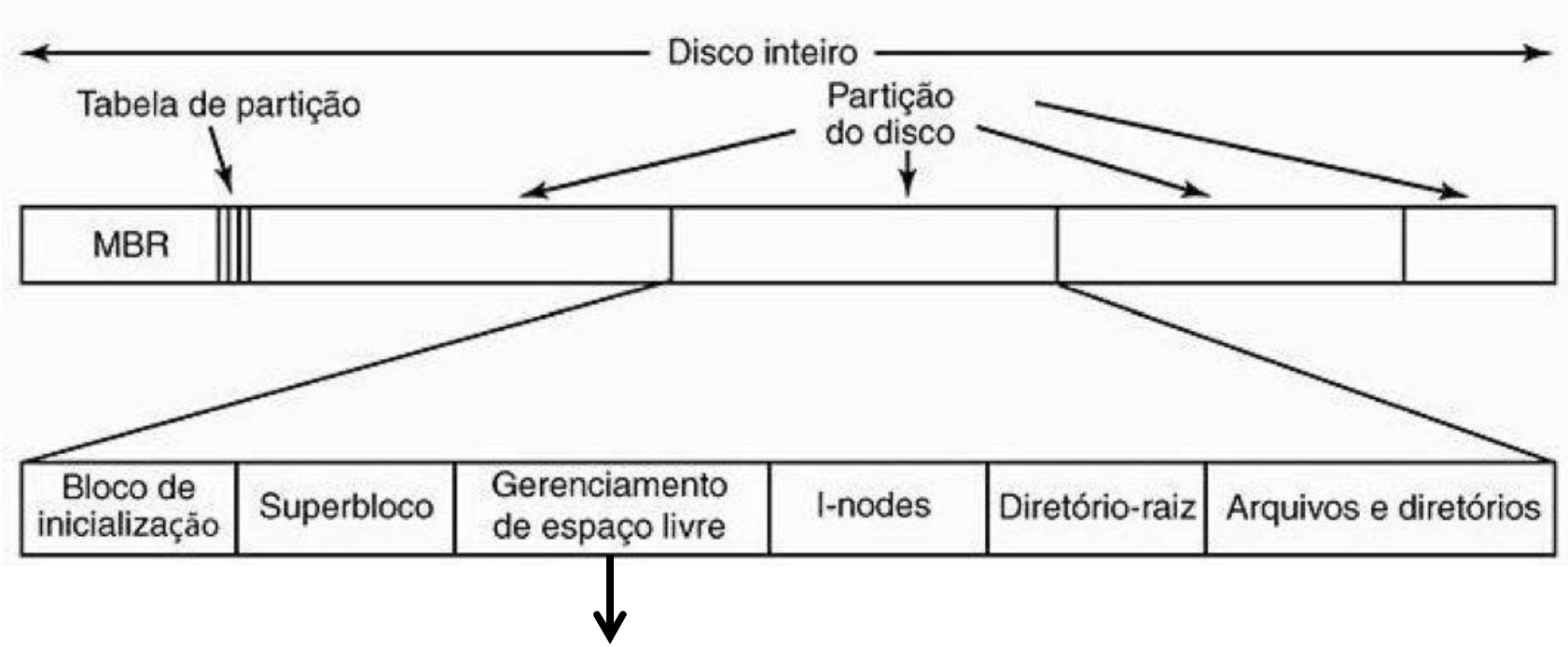
Exemplo de organização de um sistema de arquivos



Armazena informações sobre o tipo do sistema de arquivos e o número de blocos

Implementação do sistema de arquivos

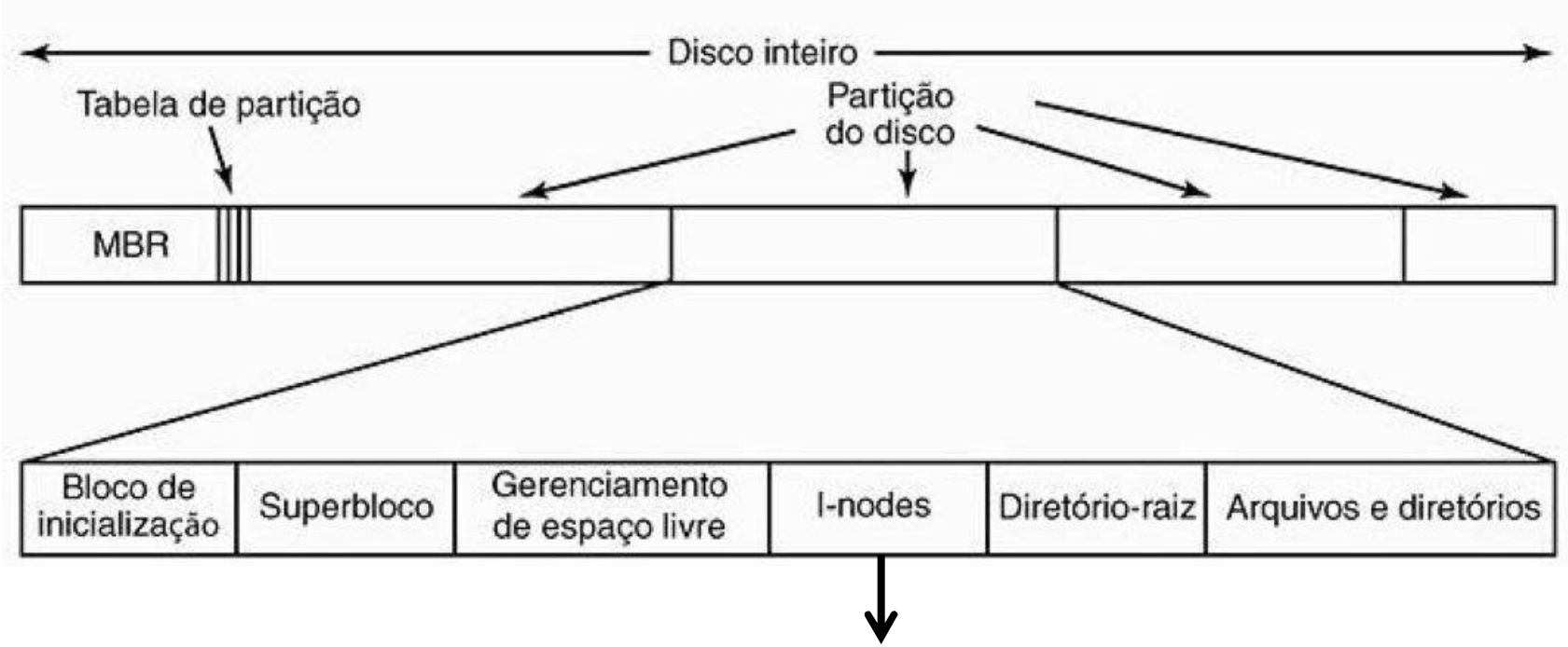
Exemplo de organização de um sistema de arquivos



Lista de blocos livres no disco

Implementação do sistema de arquivos

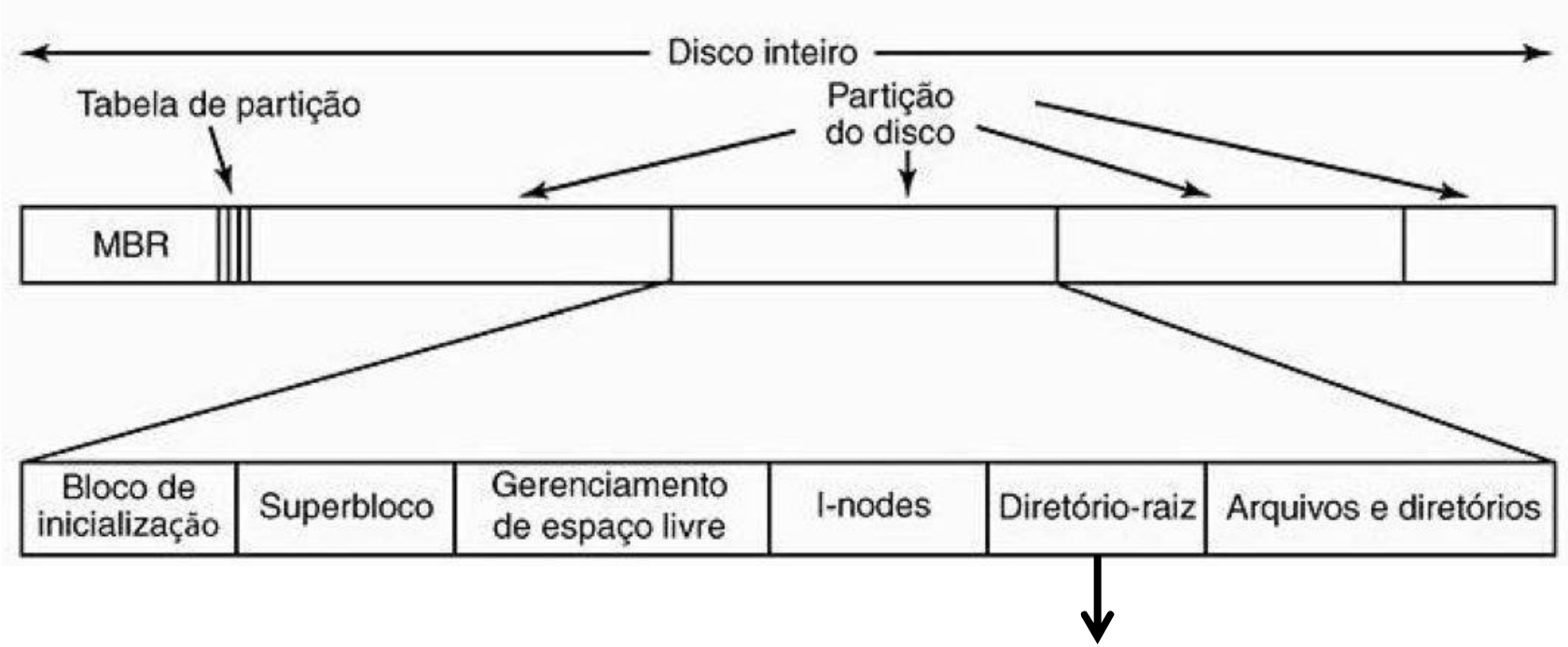
Exemplo de organização de um sistema de arquivos



Estruturas que armazenam as informações sobre cada arquivo

Implementação do sistema de arquivos

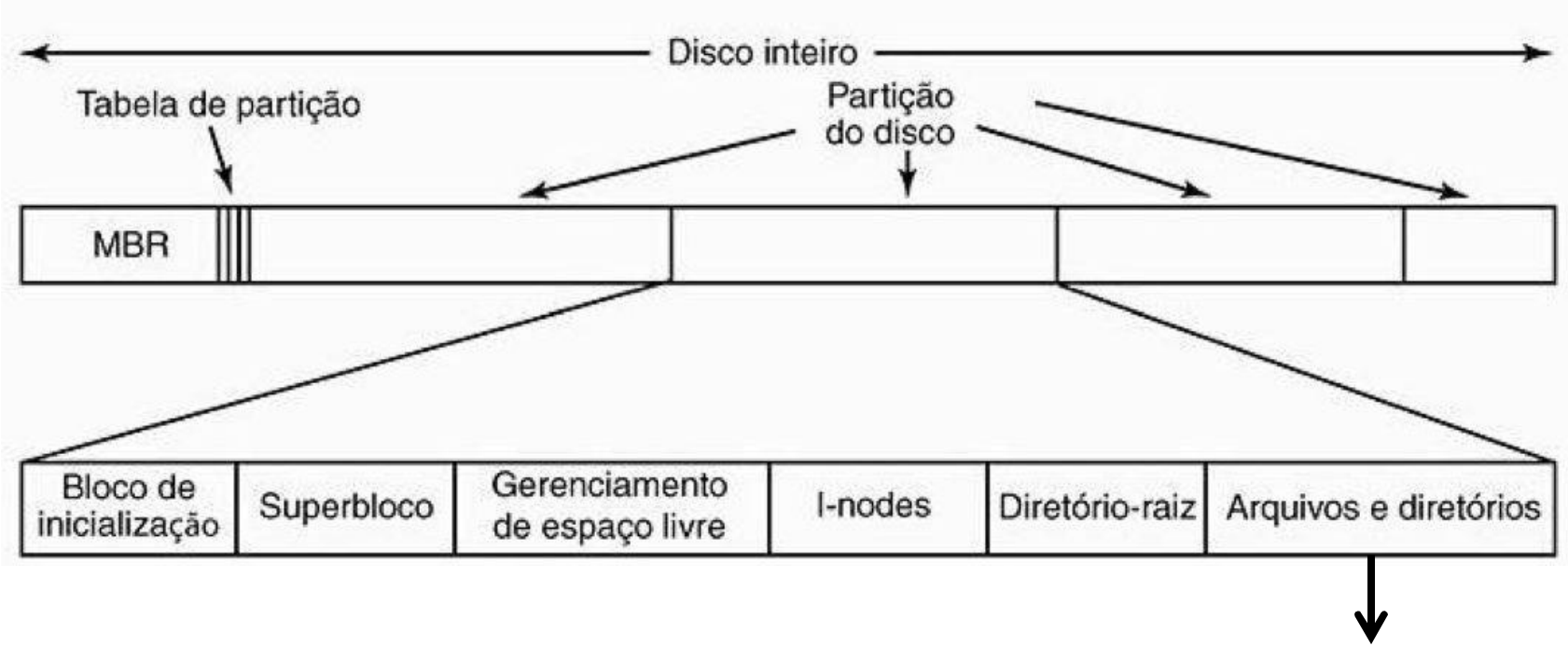
Exemplo de organização de um sistema de arquivos



Armazena o topo da árvore do sistema de arquivos

Implementação do sistema de arquivos

Exemplo de organização de um sistema de arquivos



Armazena os diretórios e arquivos propriamente ditos

Implementação de arquivos

- O gerenciamento do espaço em disco é uma das principais preocupações dos projetistas de sistemas
- **Duas estratégias gerais:** ex. arquivo de *n* bytes
  1) Alocar *n* bytes consecutivos de espaço em disco
  2) Dividir os *n* bytes em diversos blocos não necessariamente contíguos

Implementação de arquivos

1) Alocar *n* bytes consecutivos de espaço em disco (problemas)

- Necessidade de mover os dados quando o arquivo cresce e não há espaço contíguo
- Problema similar à implementação de memória virtual com **segmentação**
- **Custo muito mais alto, pois o disco é muito mais lento do que a memória RAM**
- **Logo:** quase todos os sistemas de arquivos quebram os arquivos em blocos de tamanho fixo

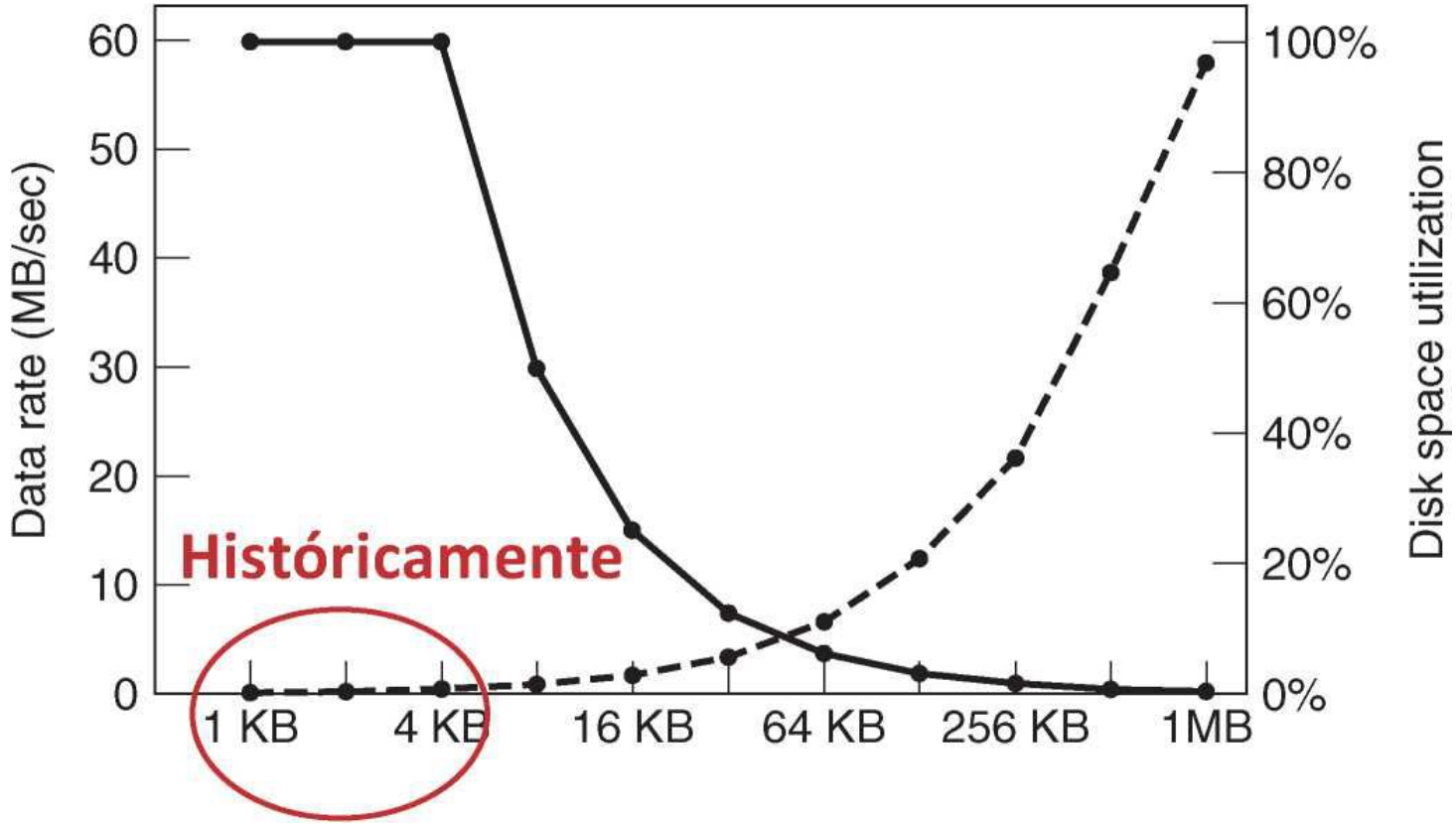
Implementação de arquivos

2) Dividir os *n* bytes em diversos blocos não necessariamente contíguos

- Qual deve ser o tamanho do bloco?
- **Discos:** setor, trilha etc.
- **Blocos grandes:** desperdício de disco
- **Blocos pequenos:** mesmo arquivos pequenos ocuparão diversos blocos no disco (múltiplas buscas e redução de desempenho), ocasionando desperdício de tempo

Implementação de arquivos

2) Dividir os *n* bytes em diversos blocos não necessariamente contíguos



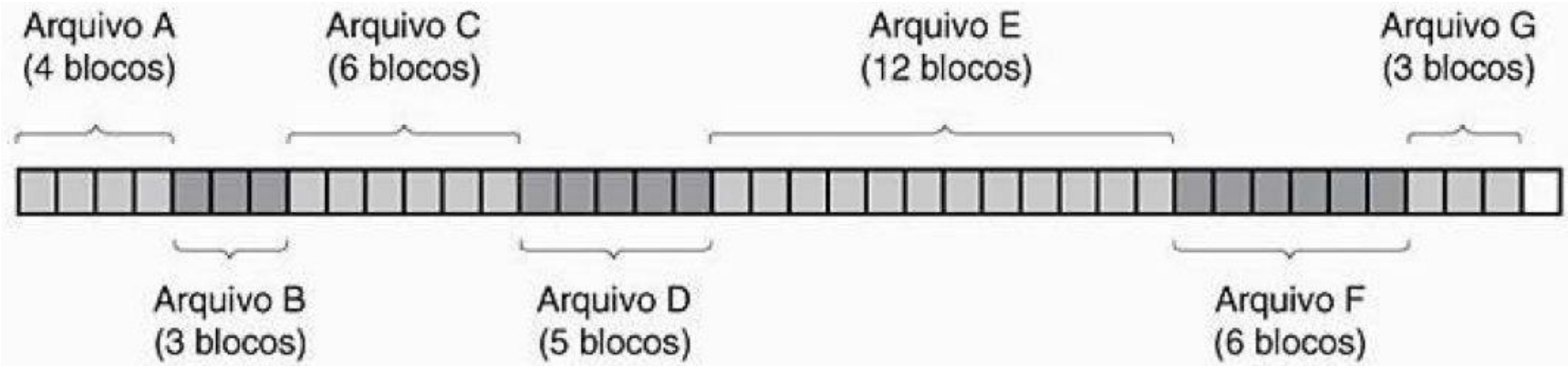
A linha pontilhada indica a taxa de transferência do disco. A linha sólida indica a utilização do disco.

Implementação de arquivos

- **Implementação de arquivos**
  - Os dados armazenados em arquivo podem ocupar 1 ou mais blocos de disco
  - O sistema de arquivos necessita **relacionar blocos do disco** com os **arquivos**
  - **Diferentes métodos podem ser usados**
    - Alocação contígua
    - Alocação por lista encadeada
    - Alocação por lista encadeada usando uma tabela na memória
    - i-nodes

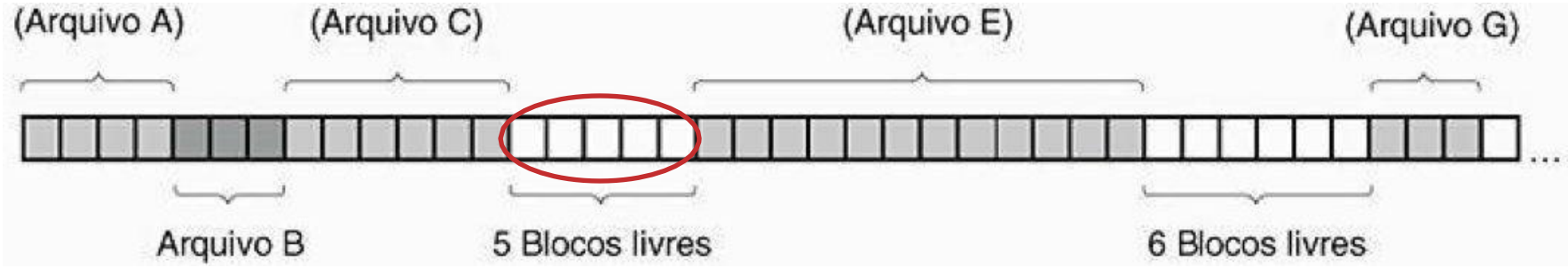
Implementação de arquivos

- **Alocação contígua**
  - Armazena cada arquivo em blocos contíguos do disco
  - **Exemplo:** em um disco com blocos de 1 KB, um arquivo contendo 50 KB de dados será alocado em 50 blocos consecutivos do disco



Implementação de arquivos

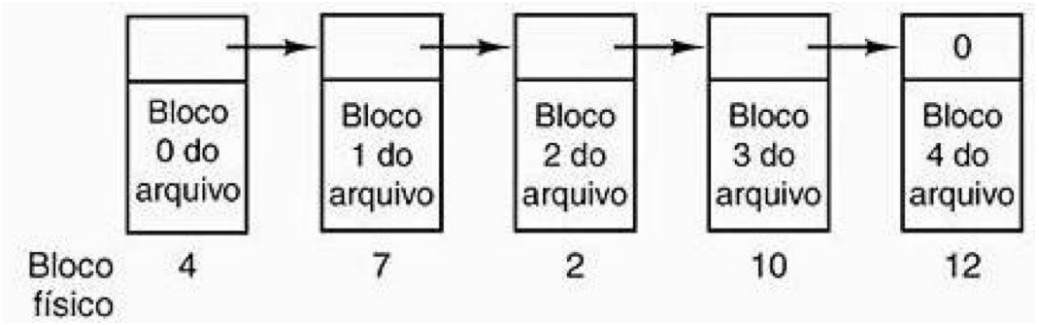
- **Vantagens**
  - Fácil de implementar
  - Excelente desempenho da leitura
- **Desvantagem**
  - Fragmentação do disco



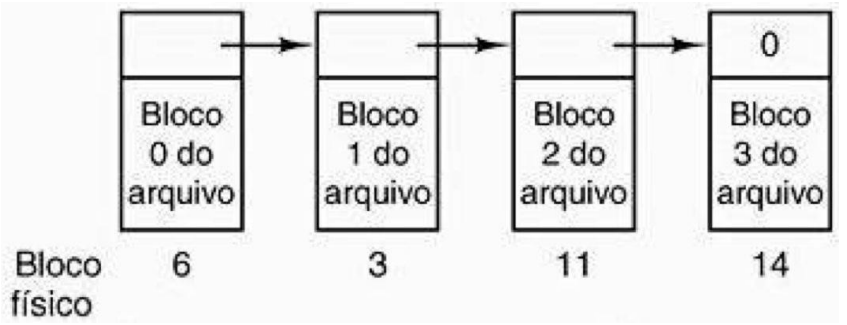
Implementação de arquivos

- **Alocação por lista encadeada**
  - Mantém os blocos de cada arquivo em uma lista encadeada de blocos de disco
  - A primeira palavra de cada bloco é usada como um ponteiro para o próximo bloco

Arquivo A



Arquivo B

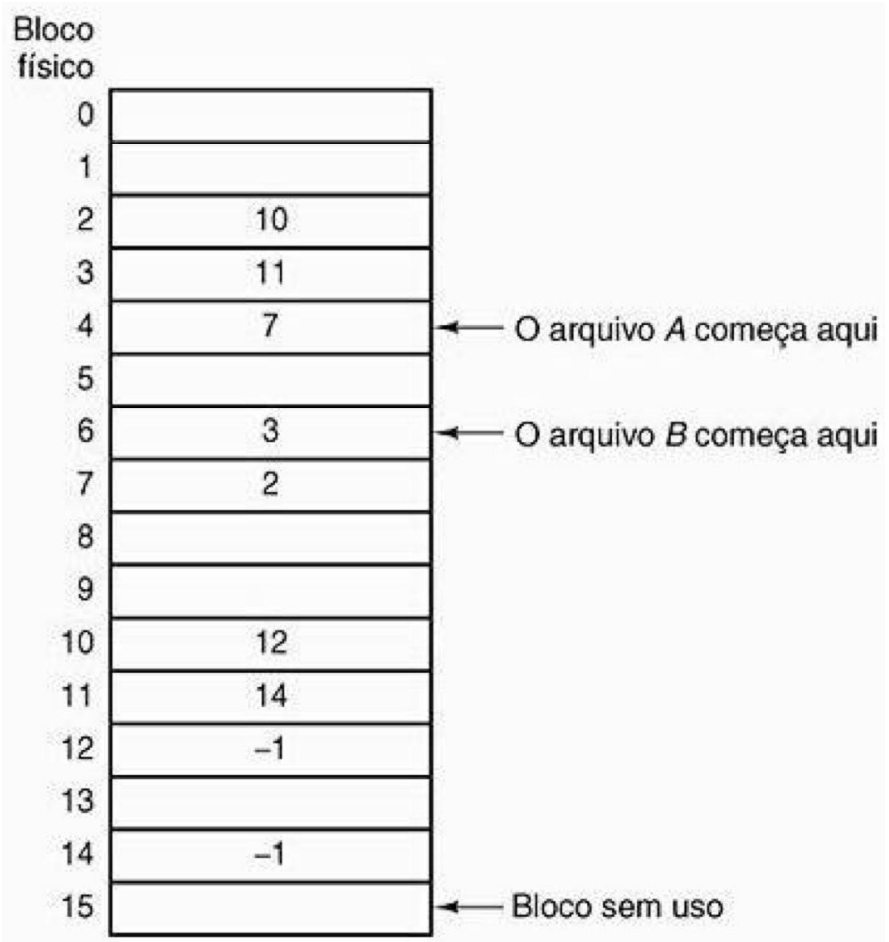


Implementação de arquivos

- **Vantagens**
  - Permite usar todos os blocos do disco, evitando o problema da fragmentação
  - Leitura sequencial com bom desempenho
- **Desvantagem**
  - Acesso aleatório é extremamente lento: para chegar ao bloco *n*, é necessário ler *n-1* blocos no disco antes dele, um de cada vez
  - Parte do bloco usada para armazenar o ponteiro para o próximo bloco não pode ser usada para armazenar dados

Implementação de arquivos

- **Alocação por lista encadeada com uma tabela na memória**
- Armazena as informações dos ponteiros da lista encadeada em **uma tabela em memória**
- Essa tabela é chamada de **Tabela de Alocação de Arquivos (FAT)**



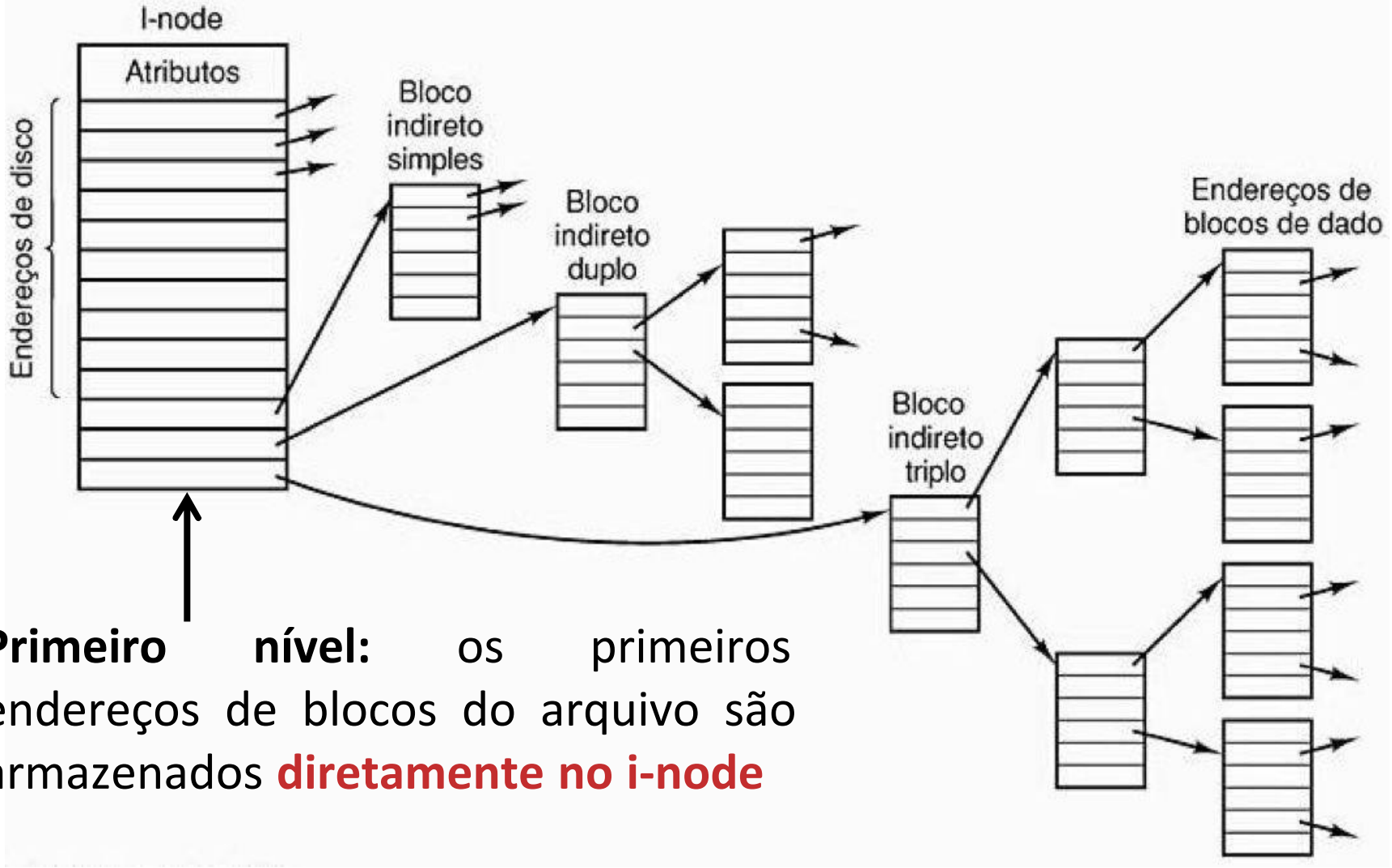
Implementação de arquivos

- **Vantagens**
  - Todo o bloco fica disponível para dados: os ponteiros são armazenados na tabela de alocação de arquivos
  - Acesso aleatório mais rápido, pois o encadeamento fica na memória, sem necessitar acessar o disco
- **Desvantagem**
  - Toda a tabela precisa estar na memória o tempo todo (ruim para discos grandes)
  - **Exemplo:** para um disco de 200 GB e blocos de disco de 1 KB, as tabelas precisarão de 200 milhões de entradas. Se cada entrada ocupa 4 bytes, a tabela ocupará 800 MB na RAM

Implementação de arquivos

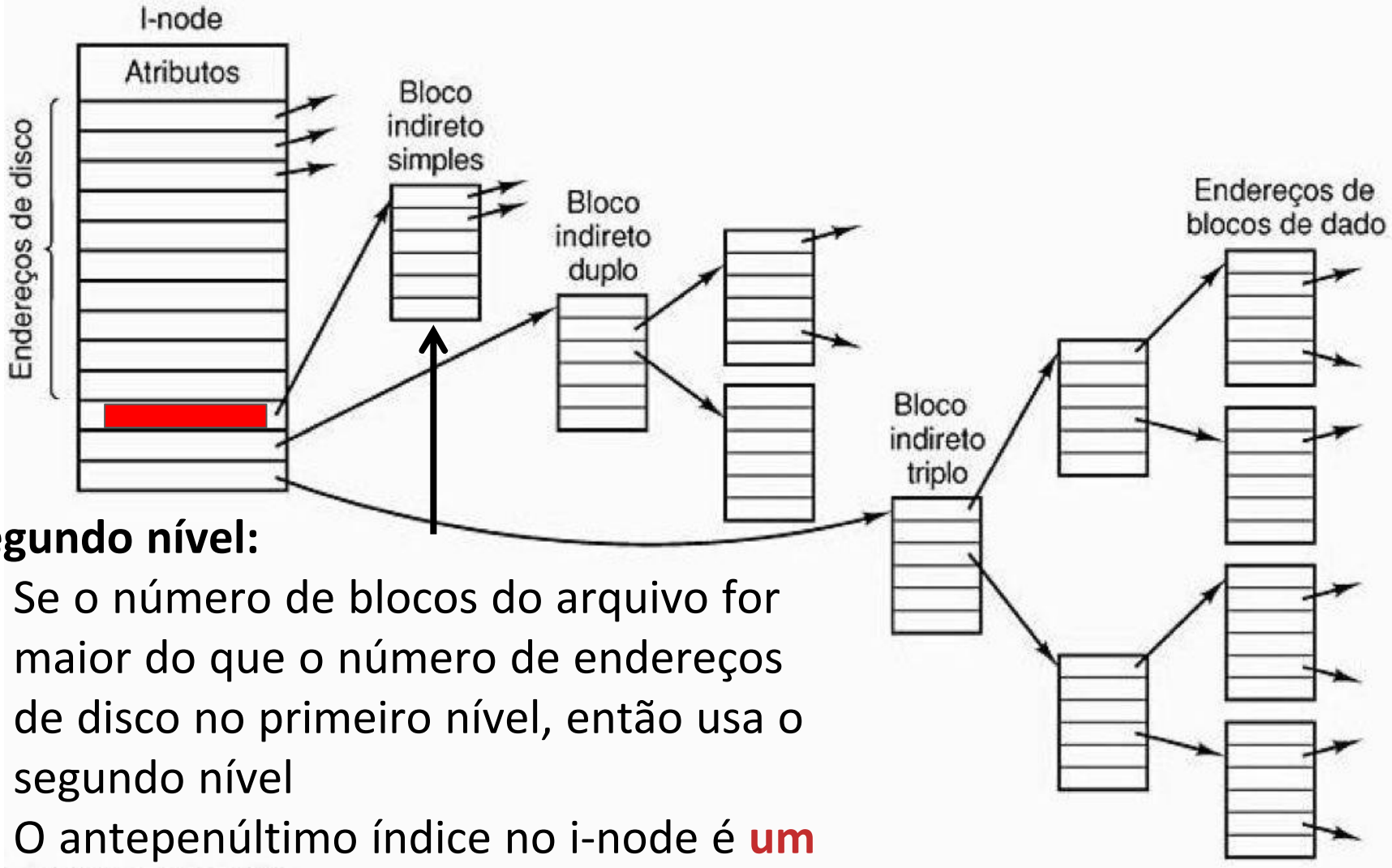
- **i-nodes**
  - É um método que associa cada arquivo a uma estrutura de dados chamada i-node (*index-node*), que relaciona os **atributos** e os **endereços em disco** dos blocos do arquivo
  - Dado o i-node de um arquivo, é possível encontrar **todos os blocos do arquivo**
  - Blocos são organizados em até **4 níveis**, cada um com um tamanho fixo

Implementação de arquivos



Primeiro nível: os primeiros endereços de blocos do arquivo são armazenados **diretamente no i-node**

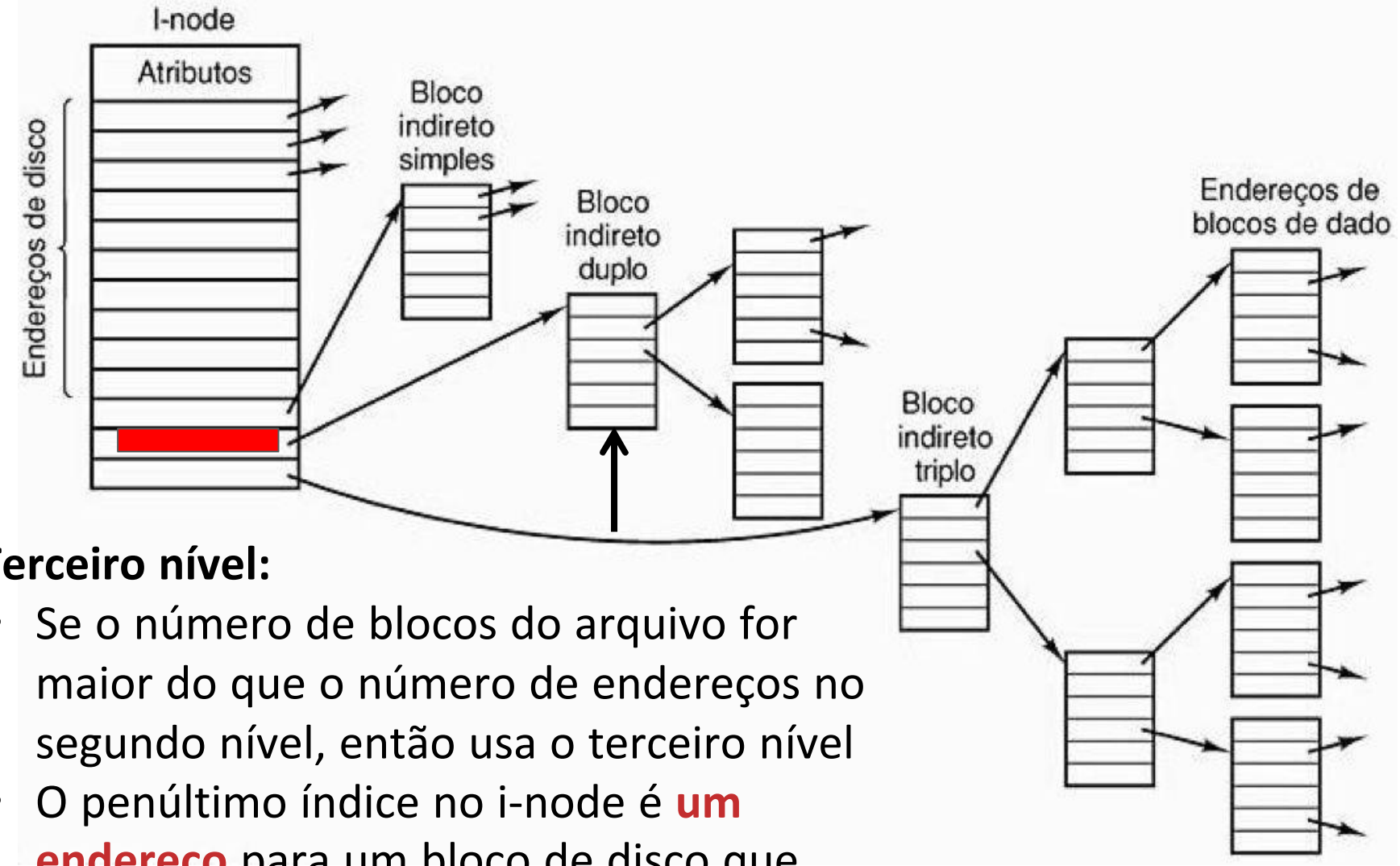
Implementação de arquivos



Segundo nível:

- Se o número de blocos do arquivo for maior do que o número de endereços de disco no primeiro nível, então usa o segundo nível
- O antepenúltimo índice no i-node é **um endereço** para um bloco de disco que contém endereços de disco adicionais

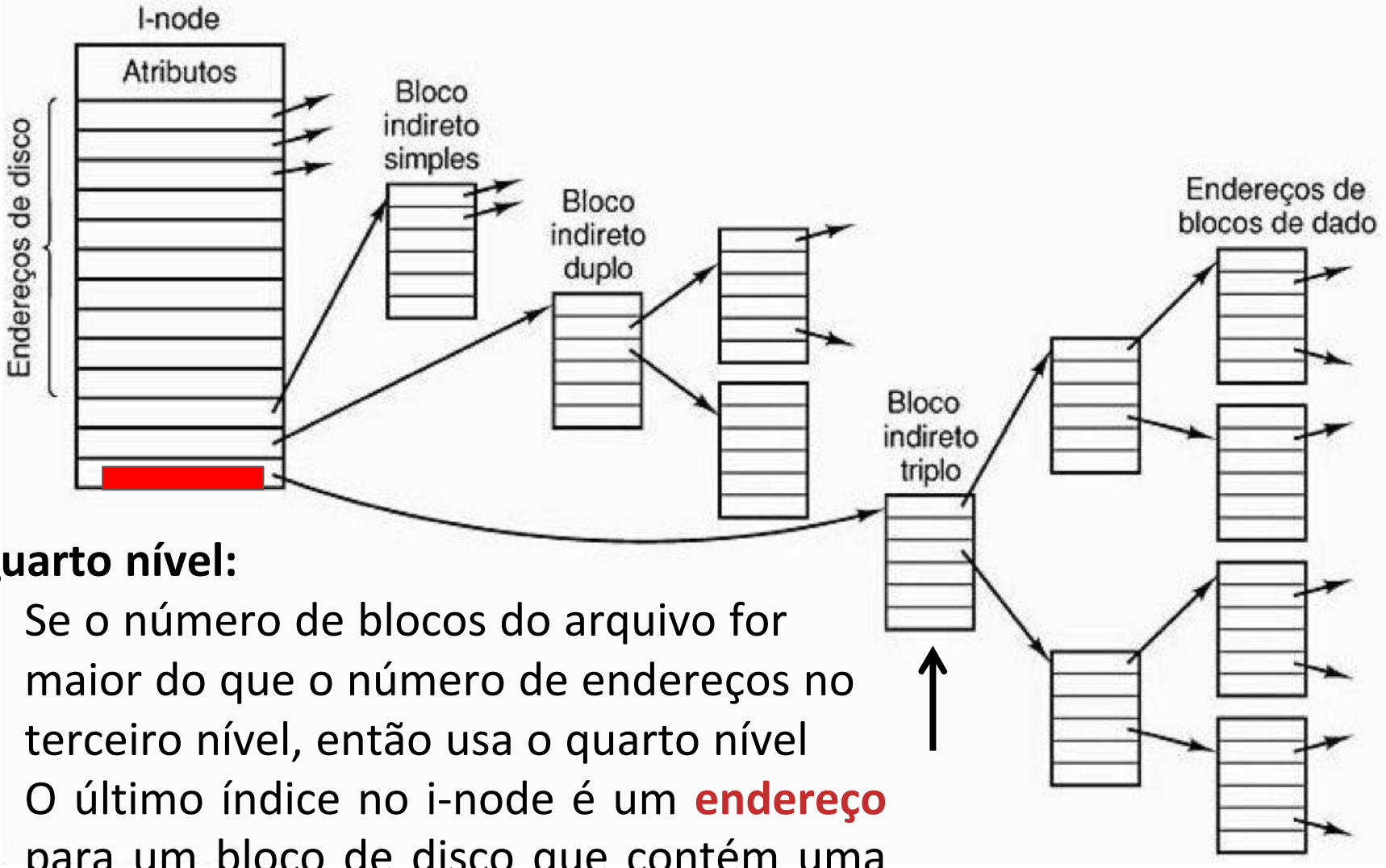
Implementação de arquivos



Terceiro nível:

- Se o número de blocos do arquivo for maior do que o número de endereços no segundo nível, então usa o terceiro nível
- O penúltimo índice no i-node é **um endereço** para um bloco de disco que contém uma **lista de blocos indiretos simples**

Implementação de arquivos



Quarto nível:

- Se o número de blocos do arquivo for maior do que o número de endereços no terceiro nível, então usa o quarto nível
- O último índice no i-node é um **endereço** para um bloco de disco que contém uma **lista de blocos indiretos duplos**

Exercício

Assuma um sistema de arquivos onde cada bloco mede 2 Kbytes (ou seja, 2 * 1024 bytes). Os endereços dos blocos são números inteiros que medem 4 bytes. Quantos acessos a disco são necessários para carregar na RAM o byte 204800 de um arquivo de 400 Kbytes usando i-nodes?

Considere que cada acesso a disco consiste de uma chamada fseek e uma chamada fread. Lembre-se que uma única chamada a fread pode ler n≥0 byte(s) consecutivo(s). Assuma que você sabe o endereço do inode do arquivo. Assuma também que cada inode ocupa somente um bloco e que os ponteiros para os conteúdos dos arquivos são da seguinte forma: 12 ponteiros diretos, 1 indireto, 1 duplamente indireto e 1 triplamente indireto.

Exercício

Assuma um sistema de arquivos onde cada bloco mede 2 Kbytes (ou seja, 2 * 1024 bytes). Os endereços dos blocos são números inteiros que medem 4 bytes. Quantos acessos a disco são necessários para carregar na RAM o byte 204800 de um arquivo de 400 Kbytes usando i-nodes?

Considere que cada acesso a disco consiste de uma chamada fseek e uma chamada fread. Lembre-se que uma única chamada a fread pode ler n≥0 byte(s) consecutivo(s). Assuma que você sabe o endereço do inode do arquivo. Assuma também que cada inode ocupa somente um bloco e que os ponteiros para os conteúdos dos arquivos são da seguinte forma: 12 ponteiros diretos, 1 indireto, 1 duplamente indireto e 1 triplamente indireto.
Resposta: 3 acessos

Implementação de arquivos

- **Vantagens**
  - Permite lidar com arquivos muito pequenos ou muito grandes de forma eficiente
  - **Acesso rápido a arquivos pequenos:** arquivos pequenos têm seus endereços armazenados diretamente no i-node

IMPLEMENTAÇÃO DE DIRETÓRIOS

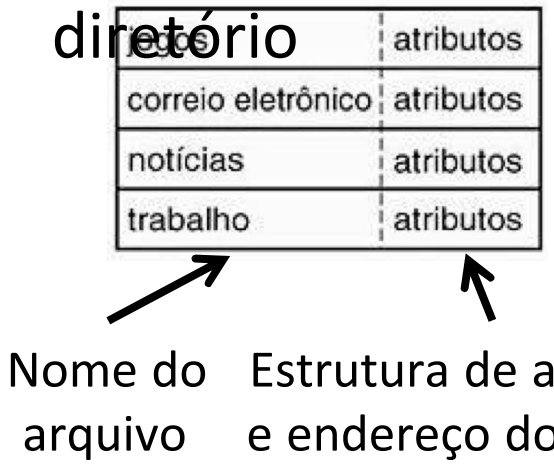
Implementação de diretórios

- A função do sistema de diretórios é **mapear** o **nome do arquivo (ou caminho)** na informação necessária para **localizar os seus dados**
  - S.O. usa o nome do caminho fornecido pelo usuário para localizar a entrada do diretório
  - A entrada do diretório fornece a informação necessária para encontrar os blocos de disco do arquivo solicitado
- Essa informação depende do método de alocação de blocos no disco:
  - **Alocação contígua:** o endereço de disco onde começa o arquivo
  - **Lista encadeada:** o número do primeiro bloco
  - **i-node:** o número do i-node

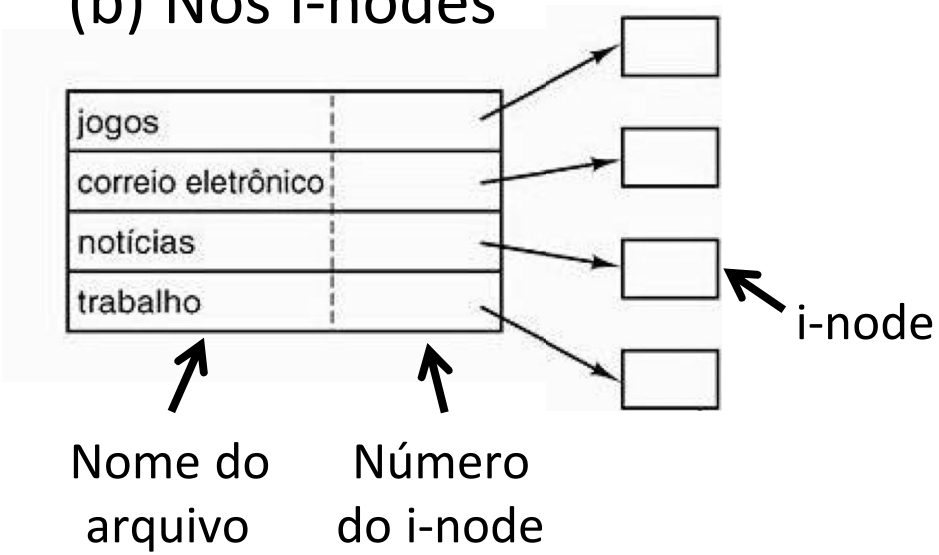
Implementação de diretórios

- Duas formas de armazenar os atributos do arquivo em um diretório

(a) Diretamente na entrada do diretório



(b) Nos i-nodes



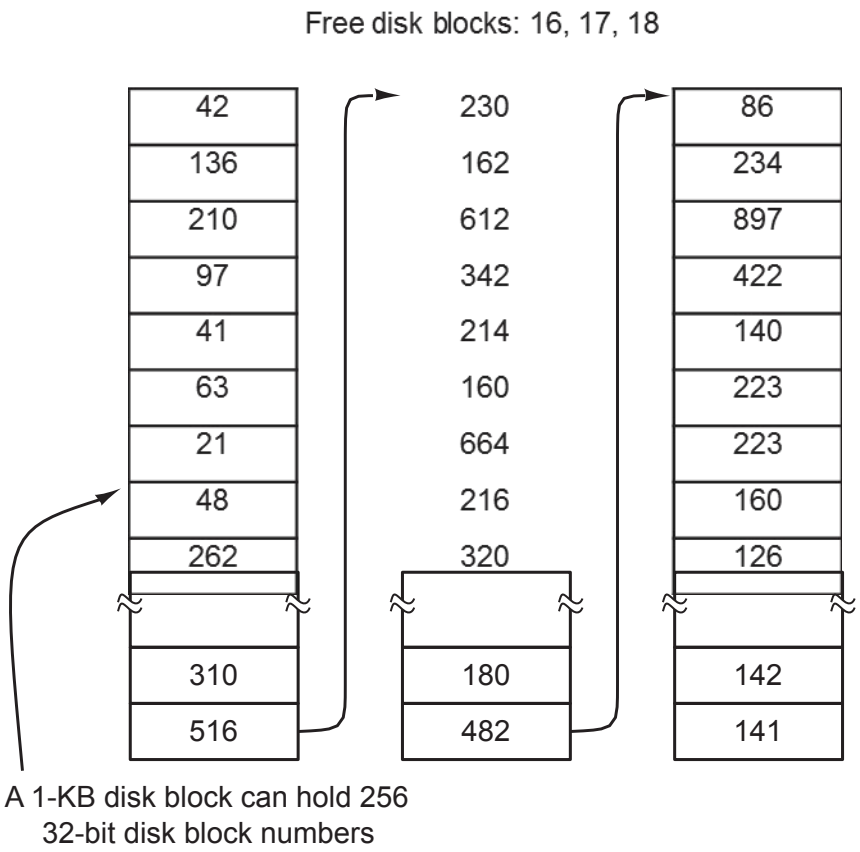
GERENCIAMENTO DE BLOCOS LIVRES

Gerenciamento dos sistemas de arquivos

- **Monitoramento dos blocos livres**
  - Lista encadeada de blocos livres
  - Mapa de bits

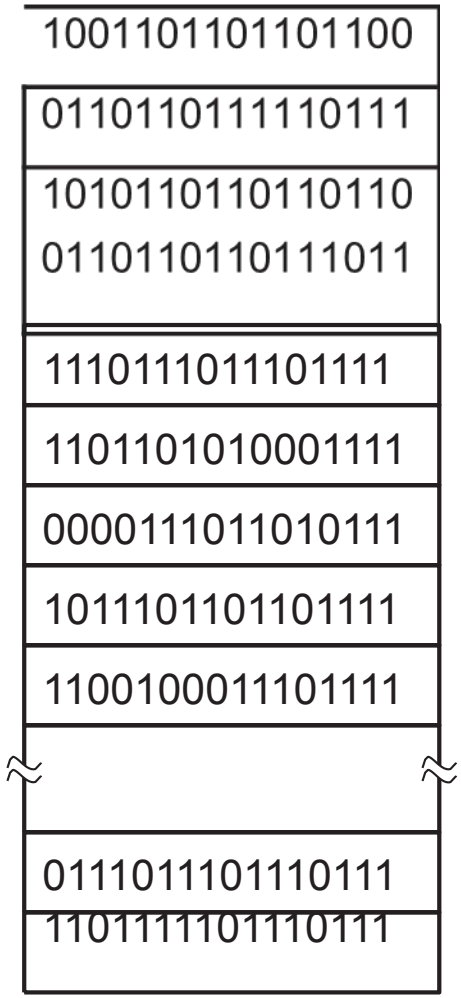
Gerenciamento dos sistemas de arquivos

- **Lista encadeada de blocos livres**
  - Lista encadeada de blocos
  - Cada bloco contém tantos blocos livres quantos couberem nele
- **Exemplo:**
- Bloco de 1KB e um número de bloco de disco de 32 bits
- Cada bloco da lista conterá números de 255 blocos livres + ponteiro para o bloco seguinte



Gerenciamento dos sistemas de arquivos

- **Mapa de bits**
  - Os blocos livres são representados no mapa por 0s e os blocos alocados por 1s
  - Um disco com *n* blocos requer um mapa de bits com *n* bits



Mapa de bits

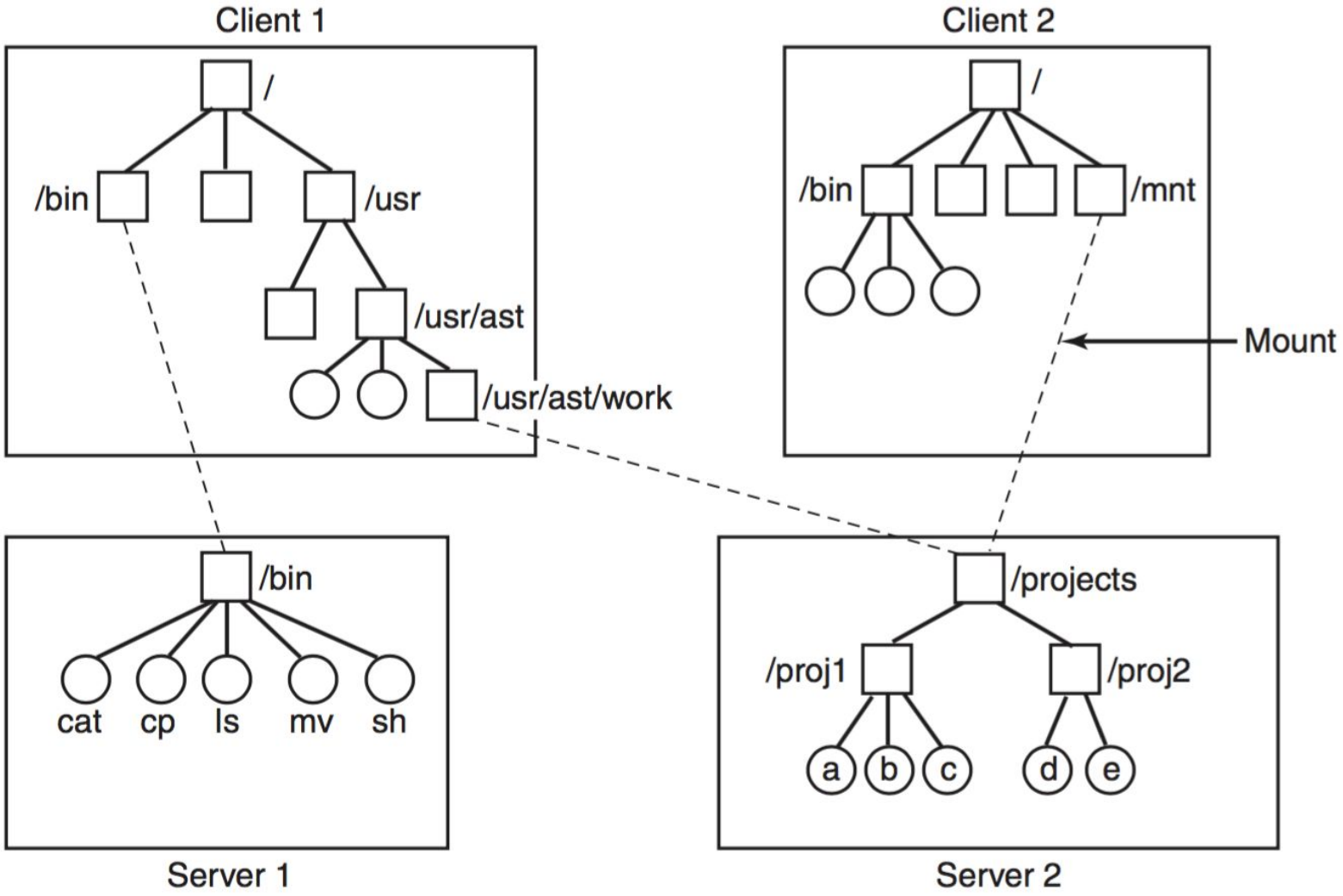
- Vantagem:
  - encontrar blocos livres é eficiente se o mapa está em memória RAM e usam-se instruções de manipulação de bits
- Desvantagem:
  - consumo de RAM (ex um disco de 1TB, 4KB bloco requer 32MB RAM)

Network File System

Definição

- NFS é um sistema de arquivos para compartilhar arquivos entre computadores remotos.
- O compartilhamento é baseado na arquitetura cliente-servidor.
- Servidor disponibiliza diretório a ser compartilhado
- Cliente acessa remotamente diretório compartilhado para fazer leituras ou escritas
- Abstrai diferenças de sistemas de arquivos locais entre cliente e servidor

Ilustração

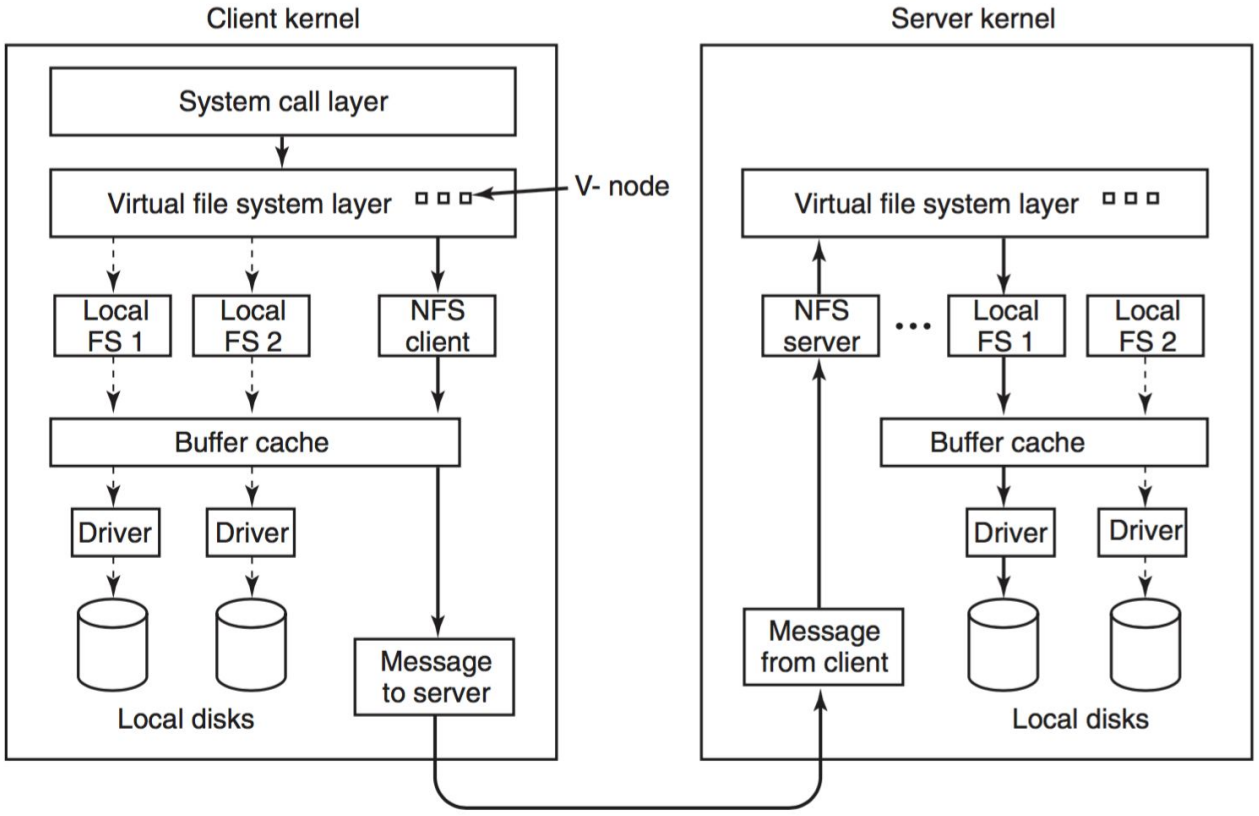

Client 1
/
/bin
/usr
/usr/ast
/usr/ast/work
Client 2
/
/bin
/mnt
Mount
/bin
cat
cp
ls
mv
sh
Server 1
/projects
/proj1
/proj2
a
b
c
d
e
Server 2

Protocolo NFS

- Provê um conjunto de operações remota (RPC).
  - Buscar um arquivo em um diretório
  - Ler um conjunto de entradas em um diretório
  - Manipular links e diretórios
  - Acessar atributos de arquivos
  - Ler e escrever arquivos
- Essas operações são permitidas somente quando o diretório remoto estiver montado no sistema de arquivos local do cliente.

Implementação do NFS: Virtual File System (VFS)

- VFS permite acessar arquivos locais ou remotos da mesma forma



OTIMIZAÇÕES

Otimizações

Duas formas de melhorar o desempenho

- Cache de blocos (*buffer cache*)
- Leitura antecipada (*prefetching*)

Otimizações

Cache de buffer (*buffer cache*)

- **Ideia:** manter na memória blocos do disco que estão sendo utilizados para melhorar o desempenho
- Blocos são trazidos do disco para a buffer cache para poderem ser lidos/modificados
- Se o bloco não se encontrar na buffer cache, primeiro ele será lido do disco para a buffer cache e, então, será acessado

Otimizações

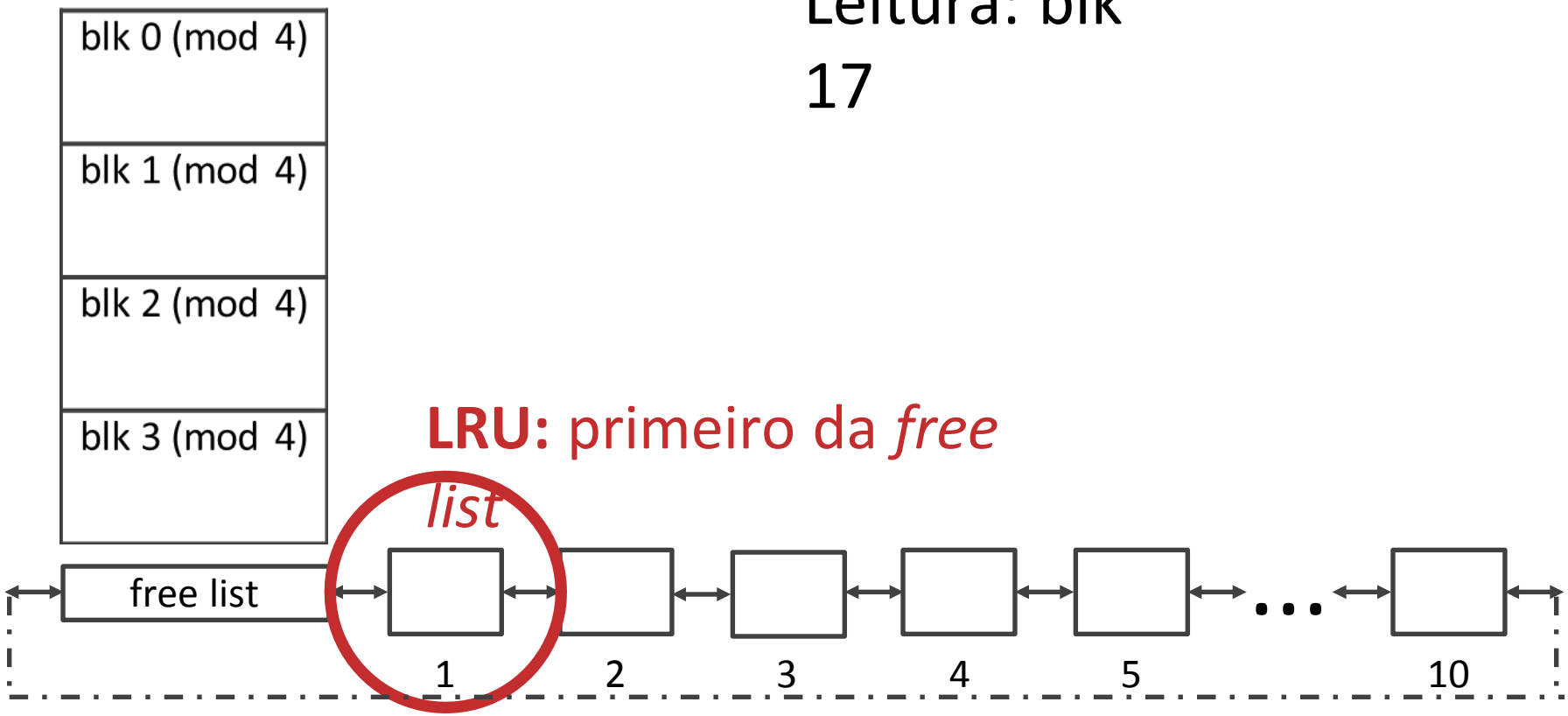
Cache de buffer (*buffer cache*)

- **Estrutura de dados**:
  - hash table + hash queue (lista dupl. encadeada)
  - free list (lista dupl. encadeada): contém buffers que não estão em uso atualmente
- Endereços de disco são usados para consultar a hash table
- Cada buffer na buffer cache contem:
  - Número de bloco alocado
  - Status**:** válido ou delayed write

Otimizações

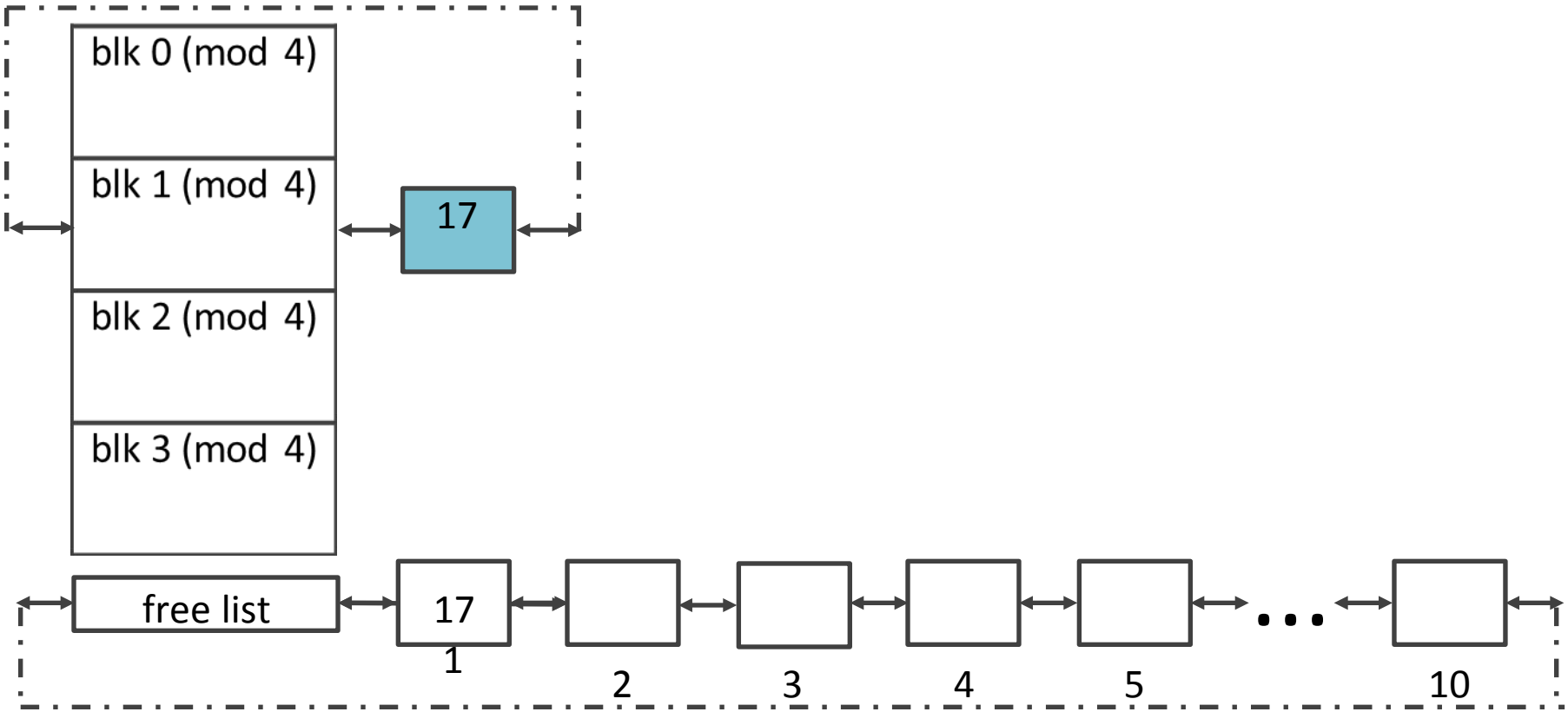
– **Exemplo:**
  - hash table de 4 entradas
  - buffer cache com 10 buffers

Leitura: blk 17



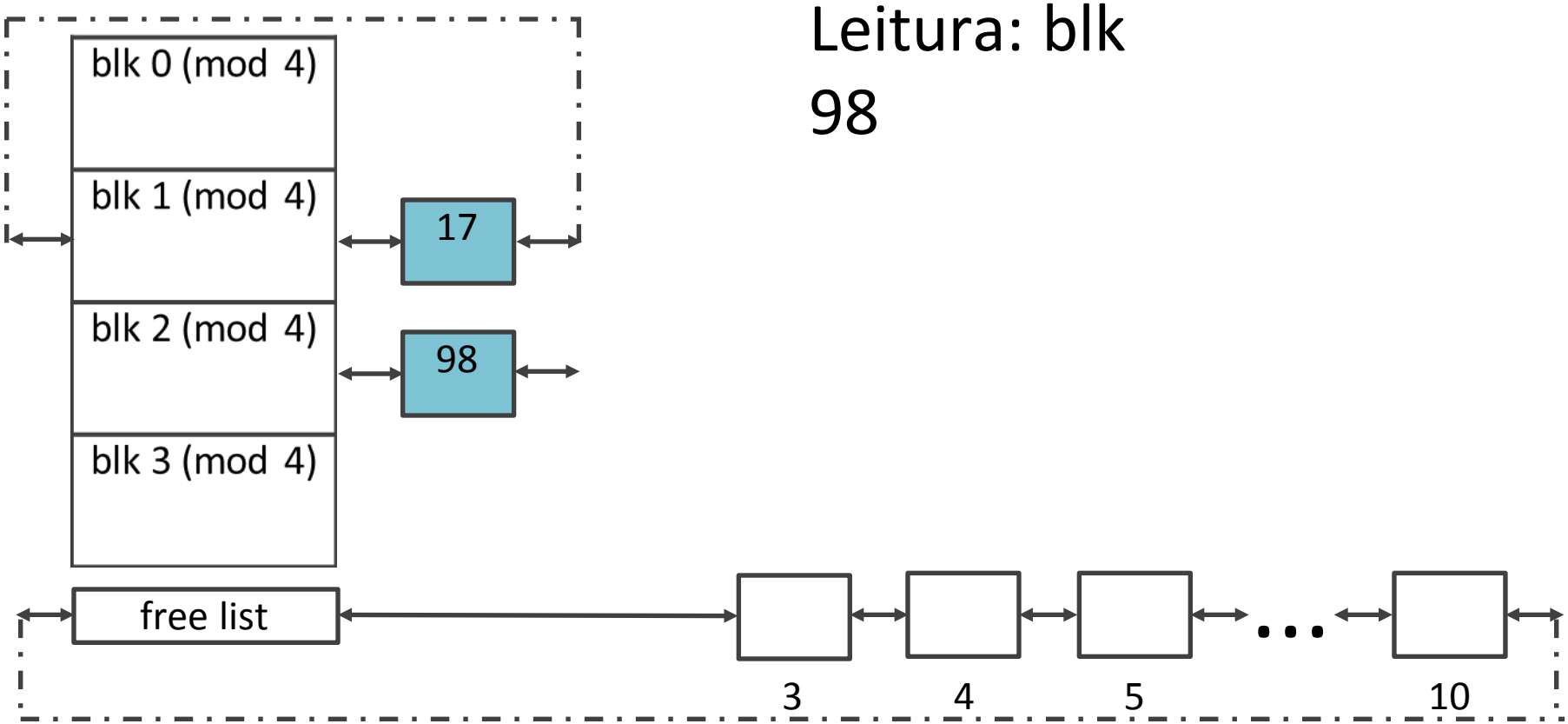
Otimizações

- **Exemplo:**
  - hash table de 4 entradas
  - buffer cache com 10 buffers



Otimizações

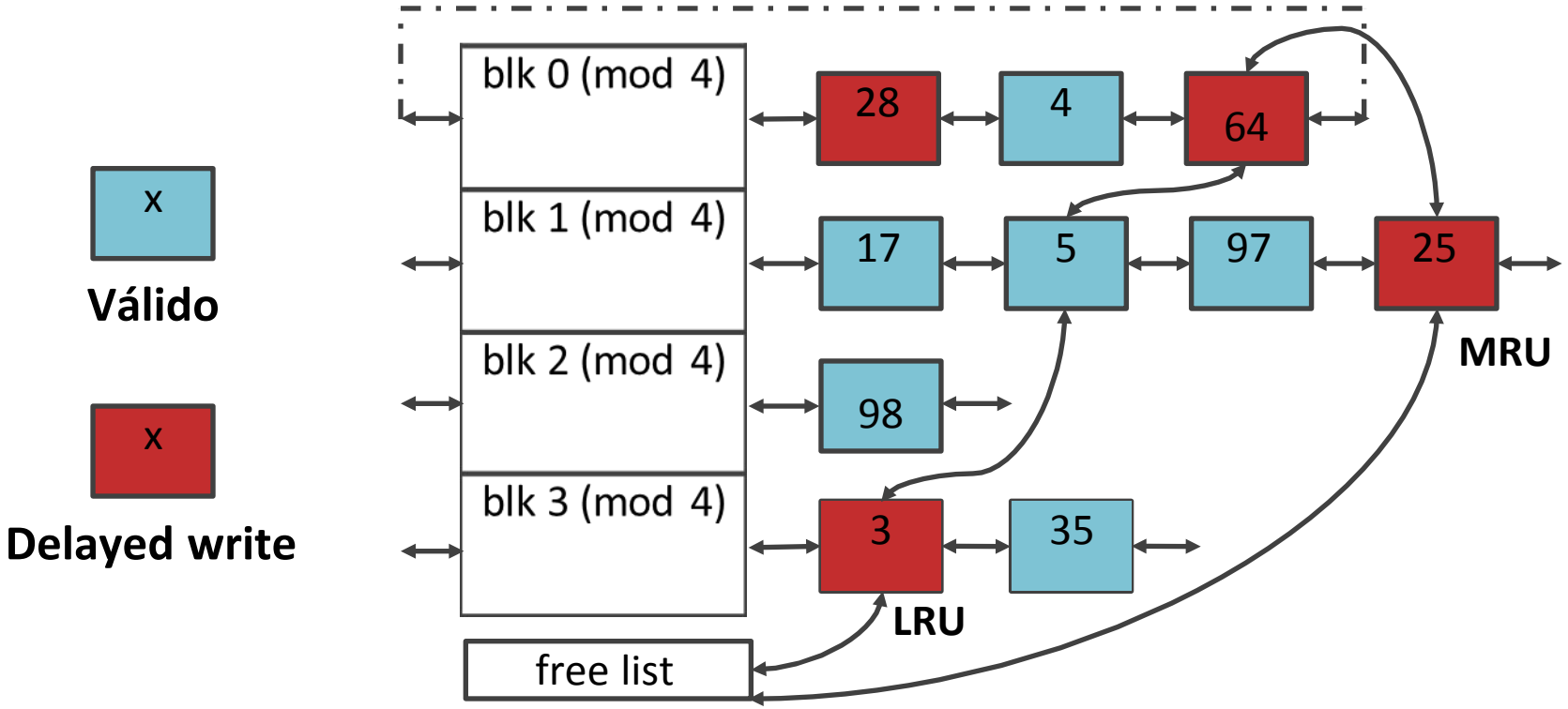
- **Exemplo:**
  - hash table de 4 entradas
  - buffer cache com 10 buffers



Leitura: blk 98

Otimizações

– **Exemplo:**
  - hash table de 4 entradas
  - buffer cache com 10 buffers



Otimizações

getblk:

```
while   (buffer not found) { if  (blk     in   hash    queue){
        if (buffer locked)     {
            sleep(event: buffer  becomes   free); continue;
        }
        lock  buffer;
        remove  buffer  from    F.L.; return      buffer;
    }
    else{…}
```

Otimizações

```
} else {
    if (there are no buffer on F.L.) {
        sleep(event: any buffer becomes free);
        continue;
    }
    lock buffer;
    remove buffer from F.L.;
    if (buffer marked delayed-write) { asynchronous
        write buffer to disk; continue;
    }
    remove buffer from old H.Q. (if needed);
    write data from disk to buffer;
    put buffer in new H.Q.;
    return buffer;
}
}
```

Otimizações

Liberação do buffer pelo processo:

- **Procedimento padrão:** insere o buffer **no final da free list (política LRU)**
- **No caso de delayed write**: o buffer é inserido no **inicio da free list** quando a escrita terminar, pois ele continua sendo o menos recentemente usado. Após a inserção, o buffer é destravado.

Otimizações

Leitura antecipada (*prefetching*)

- Transferir blocos do disco para a buffer cache antes mesmo dos mesmos serem necessários
- **Exemplo:** leitura sequencial de arquivos
  - Lê o primeiro bloco *k* do disco
  - Carrega antecipadamente na buffer cache o bloco *k*+1
- **Modo de acesso aleatório**
  - Leitura antecipada pode piorar o desempenho
  - Quando desabilitar o *prefetching*?

Otimizações

Leitura antecipada (*prefetching*)

- S.Os. monitoram o padrão de acesso em arquivos
- O *prefetching* pode ser desabilitado se o S.O. identificar que o acesso ao arquivo deixou de ser sequencial
- Pode habilitar posteriormente se o acesso voltar a ser sequencial