Sistemas de Arquivos - Implementação

MCTA026-13 - Sistemas Operacionais

Emilio Francesquini
e.francesquini@ufabc.edu.br

2019.Q1

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

Disclaimer

UFABC

- Estes slides foram preparados para o curso de **Sistemas Operacionais na UFABC**.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- Este material foi baseado nas ilustrações e texto preparados por Silberschatz, Galvin e Gagne autores do livro texto [SGG] e detentores do copyright.

Introdução

UFABC

- Estrutura do sistema de arquivos
- Implementação do sistema de arquivos
- Implementação de diretórios
- Métodos de alocação
- Manutenção do espaço livre
- Eficiência e desempenho
- Recuperação

Objetivos

UFABC

- Descrever os detalhes da implementação de sistemas de arquivos (locais) e estruturas de diretórios
- Descrever (brevemente) a implementação de sistemas de arquivos remotos
- Discutir alocação de blocos, algoritmos de controle de blocos livres e seus *trade-offs*

Organização de um Sistema de Arquivos

Estrutura de sistemas de arquivos

UFABC

- Estrutura de arquivo
  - Unidade lógica de armazenamento
  - Coleção de informações relacionadas
- Sistema de arquivos (*file system*)
  - Trabalha sobre dispositivos de armazenamento secundário (discos, SSDs, ...)
  - Provê ao usuário uma interface para armazenar arquivos assim como um mapeamento de endereçamento lógico para físico
  - Provê uma interface eficiente e conveniente que permite leituras e escritas de maneira fácil

Estrutura de sistemas de arquivos

UFABC

- Discos provêm acesso aleatório e escritas *in-place*
  - E/S são feitas em blocos ou setores (normalmente de 512 bytes)
- Bloco de controle de arquivo (*file control block*) - estrutura de armazenamento que guarda informações sobre um arquivo
- Controlador de dispositivo (*device driver*) - Controla o dispositivo físico
- A organização do sistema é feita em camadas

Organização em camadas

UFABC

application programs

↓

logical file system

↓

file-organization module

↓

basic file system

↓

I/O control

↓

devices

Camadas de um sistema de arquivos

UFABC

Device drivers

- Gerenciam dispositivos de E/S e a camada de E/S
- Recebem comandos como: ”leia o dispositivo 1, cilindro 72, trilha 2, setor 10 na posição de memória 1060”
- Emitem comandos de baixo nível específicos para o controlador do hardware em questão

Sistema de arquivos básico

- Recebem um comando como: ”leia o bloco 123”
- Emitem as chamadas apropriadas para o device driver
- Gerencia todo o processo de comunicação mantendo, por exemplo, buffers e caches
  - Buffers contêm dados em trânsito
  - Caches contêm dados frequentemente acessados

Camadas de um sistema de arquivos

UFABC

Módulo de organização de arquivos

- Responsável por arquivos, endereços lógicos, e blocos físicos
- Traduz endereços lógicos (nº do bloco) para o endereço físico
- Gerencia o espaço livre e a alocação do disco

Sistema lógico de arquivos

- Gerenciador de metadados
  - Traduz nomes de arquivos no seu identificador, file handle e localização
  - Informações mantidas em blocos de controle (inodes no UNIX)
  - Controla o diretório
  - Controla a proteção

Camadas de um sistema de arquivos

UFABC

- O uso de camadas é útil para diminuir a complexidade e redundância de código
  - Contudo pode diminuir o desempenho
  - Camadas lógicas podem ser implementadas por qualquer método de codificação (a gosto do programador do SO)
- Sistemas de arquivos diferentes são abundantes (mesmo dentro de um único SO!)
  - Cada sistema/dispositivo pode ter o seu próprio formato otimizado
    - Floppy, DVD, Blu-Ray, ...
    - CD-ROM → 9660 ISO
    - UNIX → UFS, FFS
    - Windows → FAT, FAT32, NTFS
    - Linux → 40+. EXT2, EXT3, EXT4, ReiserFS, BTRFS, ...
    - Solaris → ZFS

Implementação de um Sistema de Arquivos

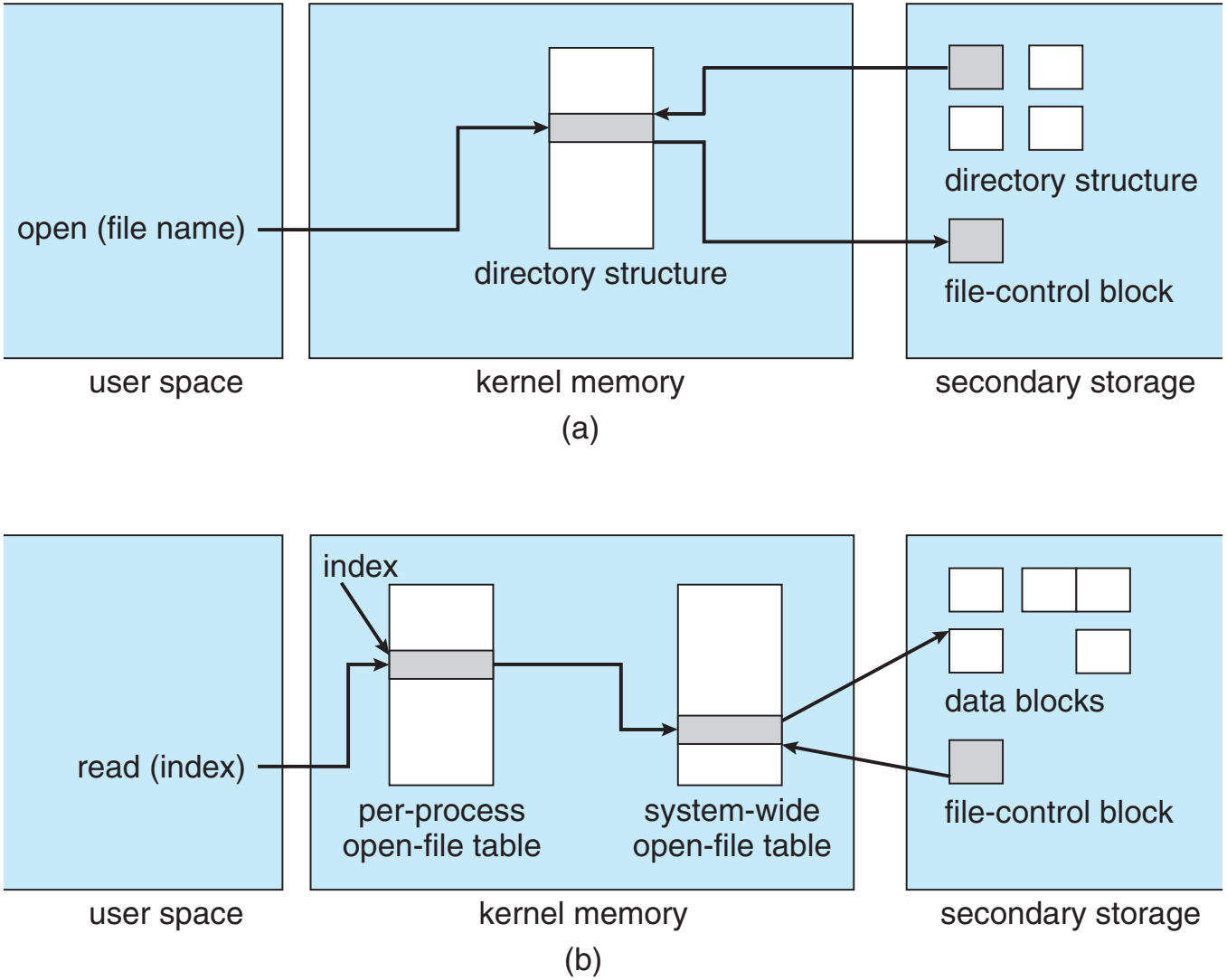
UFABC

- Aplicações utilizam *syscalls* de alto nível, mas como elas são implementadas?
  - Utiliza-se estruturas de dados mantidas em disco e em memória
- Bloco de controle de inicialização (*boot control block*)
  - Contém informações necessárias para inicializar o SO a partir daquele volume
  - Apenas é necessário se aquele volume contém um SO utilizado para iniciar a máquina
- Bloco de controle do volume (*Volume control block*, *superblock*, *master file table*)
  - Mantém os detalhes do volume
    - Número total de blocos
    - Número (e lista) de blocos livres
    - Tamanho do bloco
  - A estrutura de diretório organiza os arquivos (nomes, número dos inodes, superblock, ...)

- Cada arquivo tem um **bloco de controle de arquivo** (*FCB - File Control Block*)
  - Mantém os detalhes como número do inode, permissões, tamanho, datas
  - O NTFS, por exemplo, armazena os dados em uma tabela centralizada usando estruturas de dados similares às de um BD

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Estruturas de Sistemas de Arquivos Mantidas na Memória



(a) Abertura arquivo (b) Leitura de arquivo

UFABC

- Uma partição pode ser um volume que contém um sistema de arquivos "cozido" (*cooked*) ou "cru" (*raw*)
  - Dizemos que o sistema está "cru"quando é uma sequência de blocos sem um sistema de arquivos
- O boot block pode apontar para um volume de boot ou para um boot loader que é uma sequência de blocos que contém código apenas o suficiente para carregar o kernel a partir do sistema de arquivos
  - Ou ainda, um programa de gerenciamento de boot (Grub, Lilo, ...) para casos onde a máquina possui mais de um SO

UFABC

- Partição Raiz (*root partition*)
  - Contém o SO (outras partições podem conter outros SOs, outros sistemas de arquivos ou serem simplesmente cruas)
    - Montada durante o boot
    - Partições adicionais podem ser montadas automaticamente ou manualmente
- Durante a montagem a consistência do sistema de arquivos pode ser verificada
  - Os metadados estão todos corretos?
    - Tenta corrigir em caso de problemas e tenta novamente
    - Adiciona à tabela de pontos de montagem e permite acesso

Implementação

Implementação do diretório

UFABC

Lista linear

- Lista com os nomes dos arquivos e ponteiros para os blocos de dados
- Simples de programar
- Baixo desempenho
  - Tempo de busca linear
  - Pode-se melhorar o desempenho se mantiver os dados como lista-ligada em ordem alfabética ou B+ tree

Tabela de Hash

- Bom desempenho
- Precisa de um tratamento de **colisões**
- Implementação simples se as entradas forem de tamanho fixo. Pode-se usar métodos como *chained-overflow* caso contrário

Métodos de Alocação de Espaço em Disco

UFABC

- Discos são dispositivos com acesso direto e, portanto, dão flexibilidade para implementar arquivos
- Um dos principais problemas é como alocar espaço para estes arquivos de maneira que o espaço em disco seja utilizado de maneira eficaz e eficiente
- Há três maneiras principais em uso para alocação de espaço em disco
  - Contíguo
  - Encadeado
  - Indexado

Alocação Contígua

UFABC

Cada arquivo ocupa uma sequência contígua de blocos.

Prós

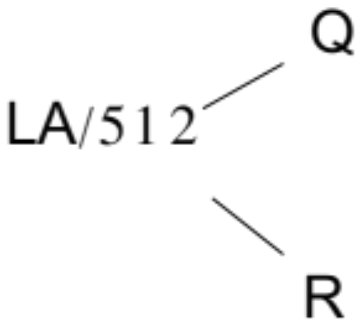
- Em geral tem o melhor desempenho
- É simples já que apenas o (nº do bloco inicial e o tamanho (em blocos) do arquivo são necessários

Contras

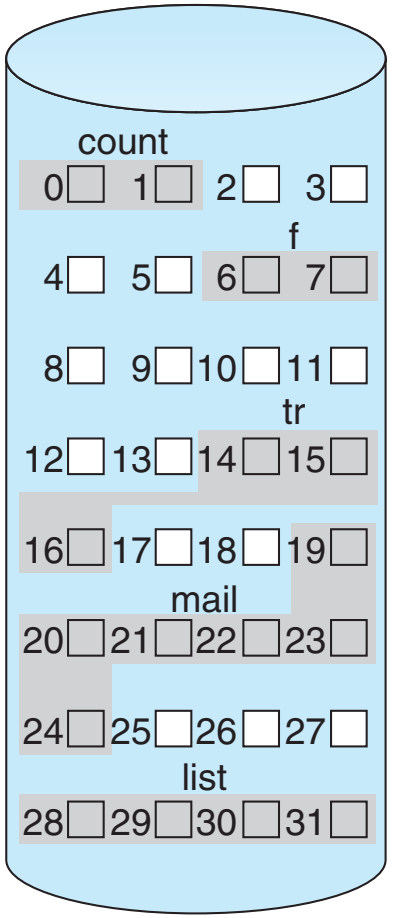
- Encontrar espaço para um arquivo
- Saber o tamanho do arquivo (tamanho máximo)
- Fragmentação externa
- Necessidade para compactação
  - Off-line (*downtime*)
  - On-line

Alocação Contígua

- Mapeamento lógico → físico (assuma o tamanho de bloco 512)



- Bloco a ser acessado → "bloco inicial"+ Q
- Deslocamento no bloco = R



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Sistemas de Arquivos Baseados em Extent

UFABC

- Muitos sistemas mais recentes utilizam um sistema de arquivos baseado em uma versão modificada do sistema de alocação contígua
- Sistemas de arquivos baseados em *extents* alocam blocos em disco baseados em *extents*
- Um extent é um conjunto de blocos contíguos no disco
  - Quando um arquivo é alocado, extents são alocados para armazená-lo
  - Um arquivo pode ser composto de um ou mais extents.

Alocação Encadeada

UFABC

- Em uma **alocação encadeada** cada arquivo é armazenado como uma lista encadeada de blocos
- Cada bloco tem o ponteiro para o próximo bloco
- Assim como uma lista encadeada tradicional, a sequência (= o arquivo) termina no ponteiro nulo

Alocação Encadeada

UFABC

Prós

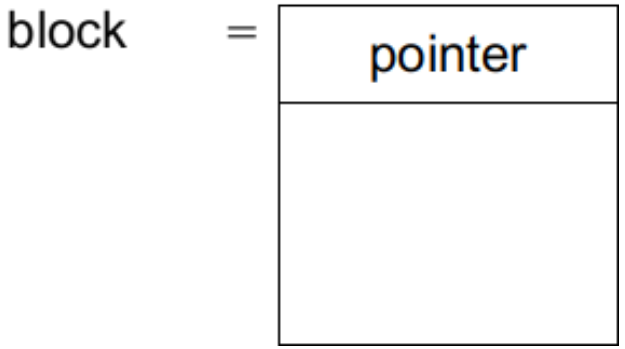
- Não sofre de problemas de fragmentação externa
- Não precisa de mecanismos de compactação
- O sistema de gerenciamento de espaço livre é ativado quando é preciso um novo bloco

Contras

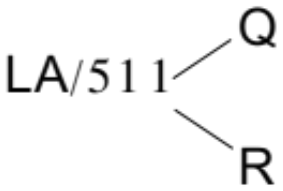
- Localizar um novo bloco pode precisar fazer várias operações de E/S e buscas no disco
- Confiabilidade pode ser um problema
  - E se apenas um ponteiro da sequência for corrompido?
- Eficiência inferior ao método de alocação contígua. Pode ser aprimorada agrupando blocos em conjuntos (porém sob pena de um aumento na fragmentação interna)

Alocação Encadeada

- Cada arquivo é uma lista encadeada de blocos do disco: os blocos podem estar espalhados pelo disco todo



- Mapeamento



  - Bloco a ser acessado é o q-ésimo bloco na cadeia de blocos que representam o arquivo
  - Deslocamento no bloco = R + 1

Alocação Encadeada

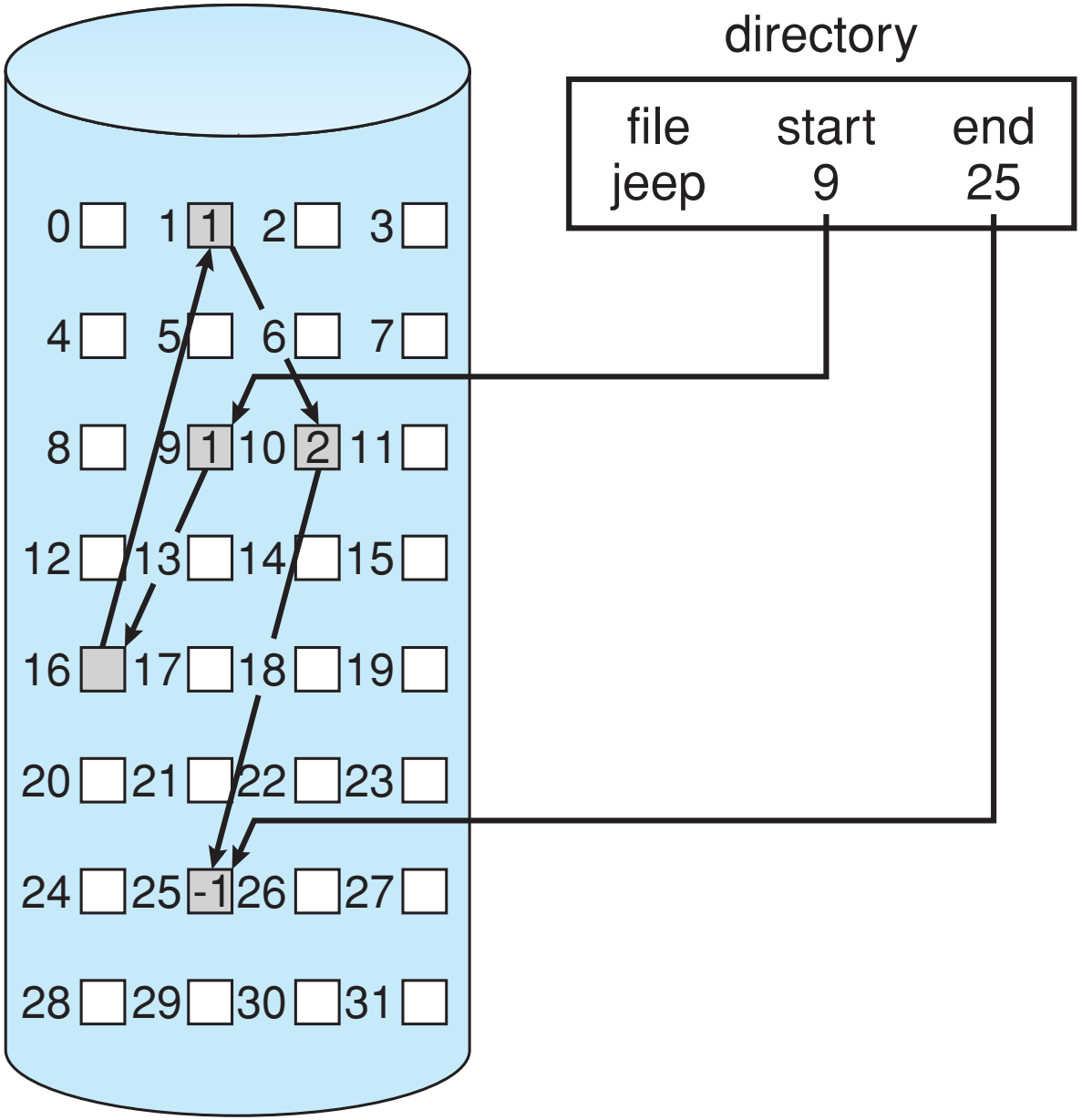
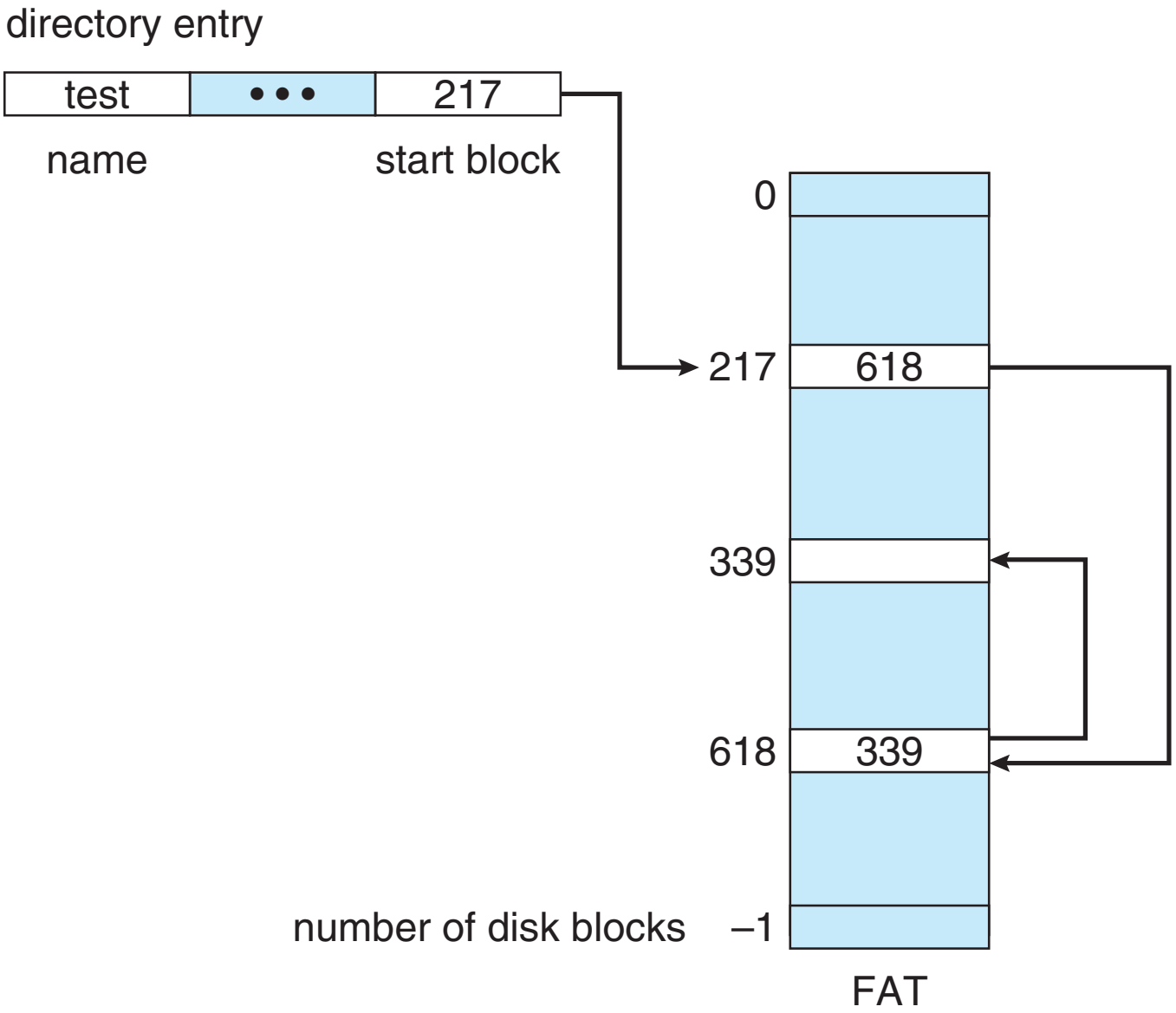
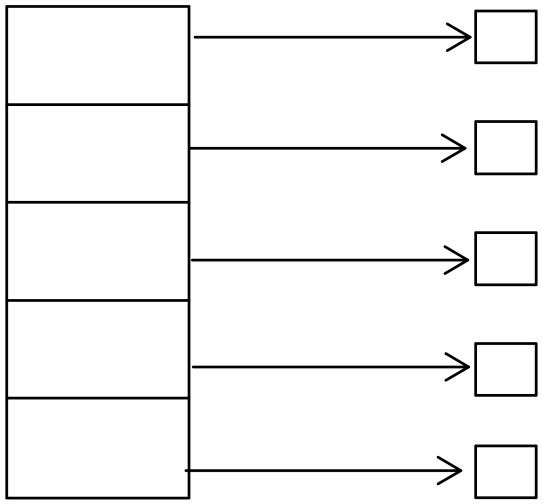


Tabela de Alocação de Arquivos (FAT)



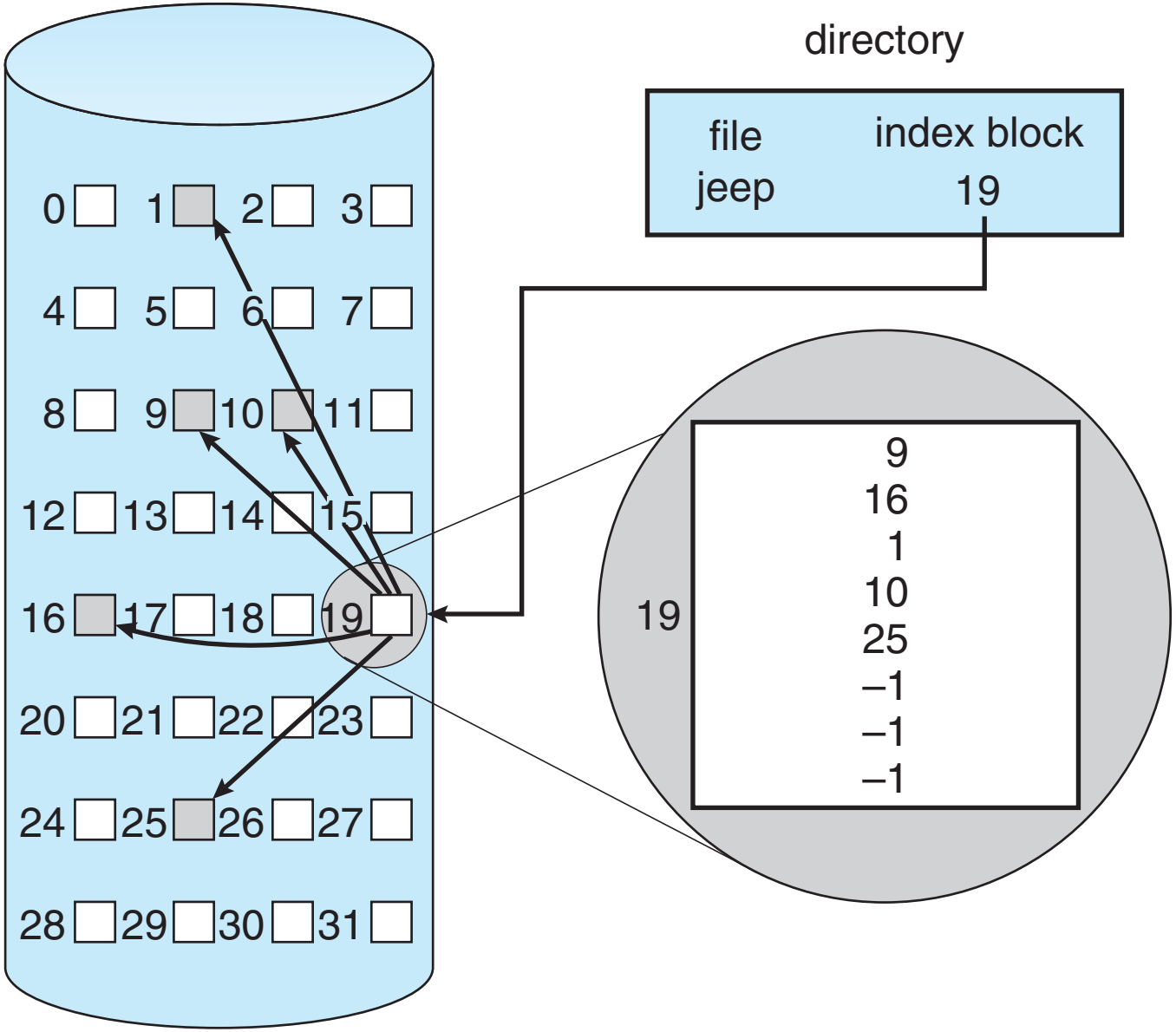
Alocação Indexada

- Cada arquivo possui o seu próprio índice de blocos com os ponteiros para os seus blocos
- Visão lógica



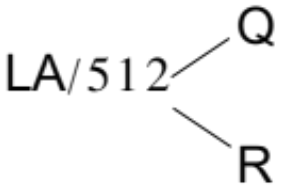
index table

Alocação Indexada



Alocação Indexada

- Precisa de uma tabela de índice
- Acesso aleatório
- Acesso dinâmico sem fragmentação externa, contudo têm o overhead do bloco de índice
- Mapeamento do lógico para o físico em um arquivo de tamanho máximo de 256KB com o tamanho de bloco de 512 bytes. Precisamos apenas um bloco por tabela de índices



  - Q = deslocamento na tabela de índices
  - R = deslocamento no bloco

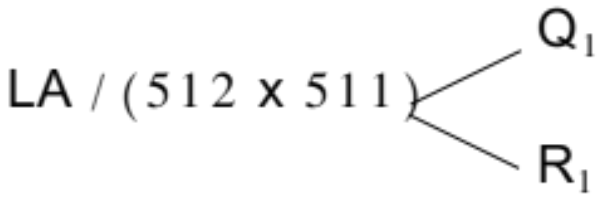
Alocação Indexada - Arquivos Grandes

UFABC

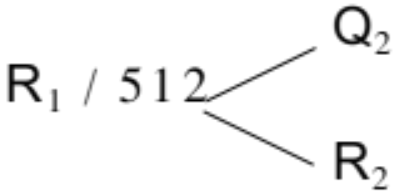
- Mapeamento do lógico para o físico em um arquivo de tamanho arbitrário (tamanho do bloco de 512 palavras)
- Pode ser alcançado com 3 métodos diferentes
  - Encadeamento
  - Índice de múltiplos níveis
  - Esquema combinado

Alocação Indexada - Encadeamento

- Encadeia-se os blocos da tabela de índice



  - Q_1 = bloco da tabela de índice
  - R_1 é usado da seguinte maneira

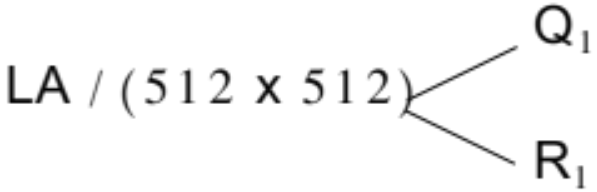


  - Q_2 = deslocamento no bloco da tabela de índice
  - R_2 = deslocamento no bloco do arquivo

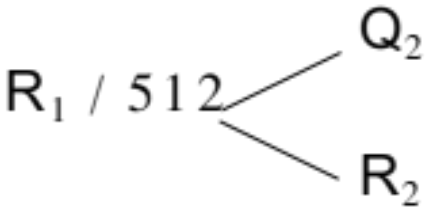
Alocação Indexada - Índice de Múltiplos Níveis

UFABC

- Índice de dois níveis (4K blocos podem armazenar 1024 ponteiros de 4 bytes no índice externo → 1.048.567 blocos de dados e tamanho de arquivo até 4GB)

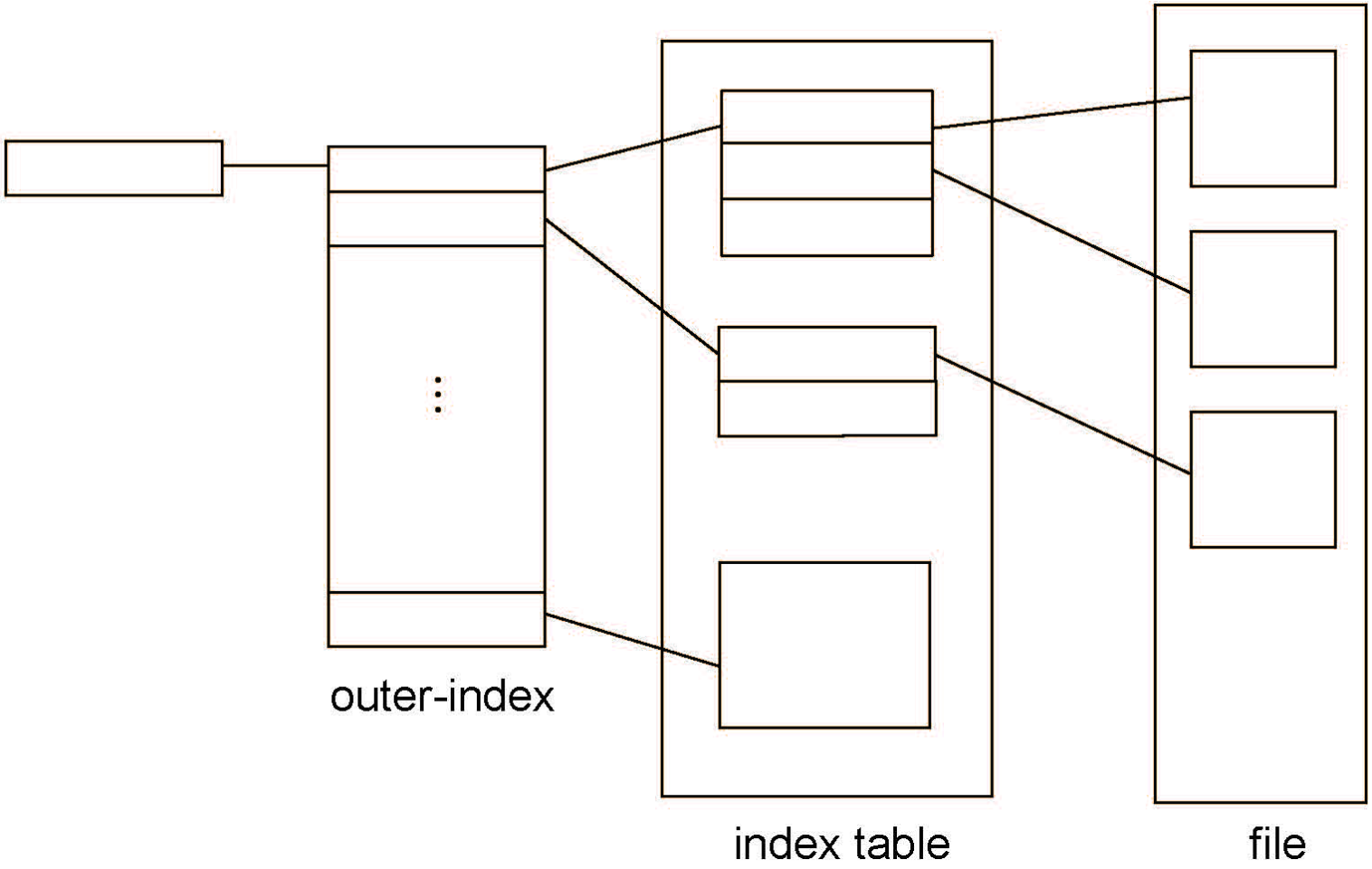


  - Q_1 = deslocamento no índice externo
  - R_1 é usado da seguinte maneira



  - Q_2 = deslocamento no bloco da tabela de índice
  - R_2 = deslocamento no bloco do arquivo

Alocação Indexada - Índice de Múltiplos Níveis



Alocação Indexada - Esquema Combinado: UNIX UFS

UFABC

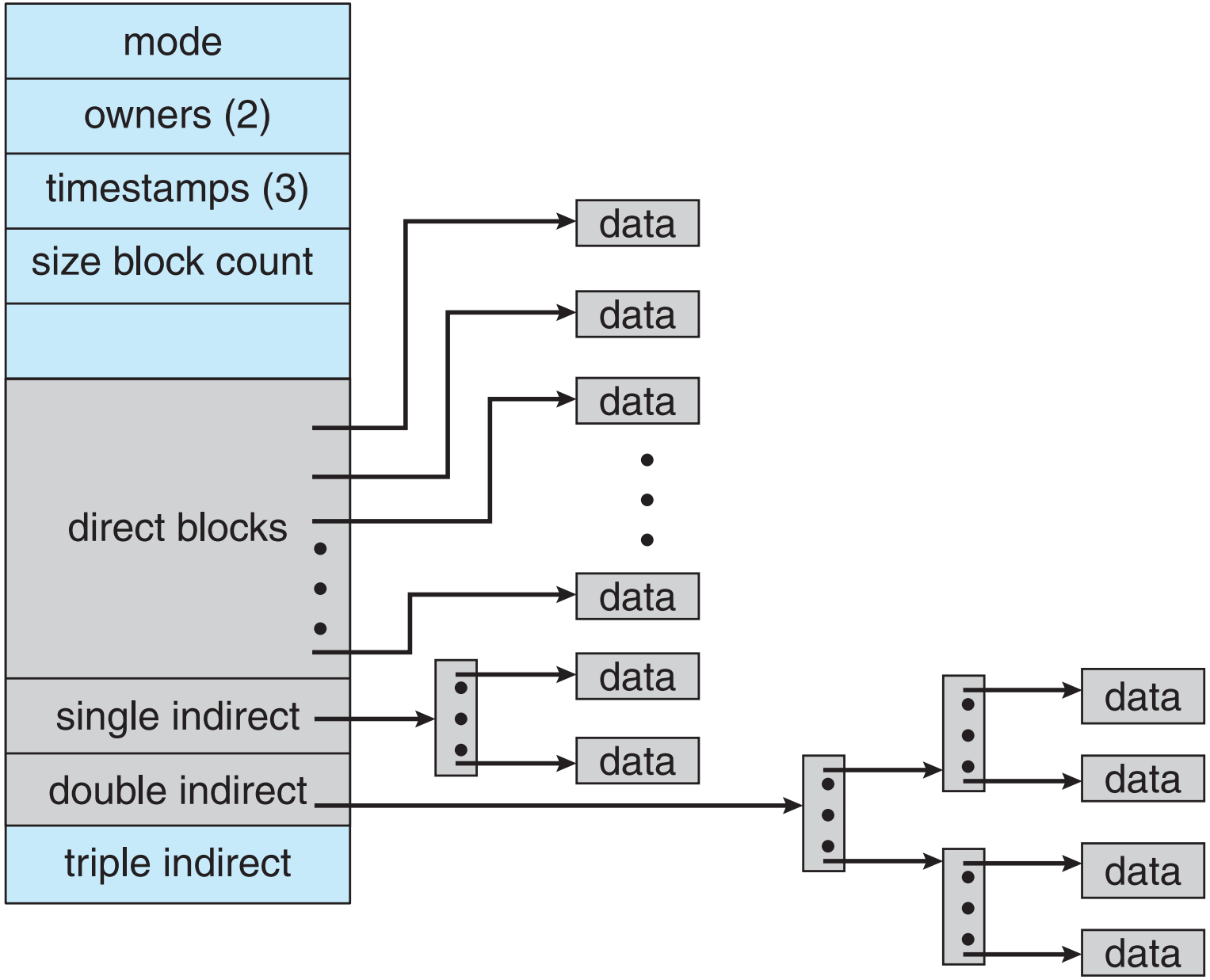
- 4K por bloco, endereços de 32 bits
- Mantém os primeiros 15 ponteiros do bloco de índice no *inode* do arquivo
- Os 12 primeiros destes ponteiros apontam diretamente para blocos do arquivo, isto é, contêm os endereços dos 12 primeiros blocos de dados do arquivo.
  - Não há a necessidade de um índice de blocos separado para arquivos pequenos (pequeno < 12 blocos). Até 48KB de dados podem ser acessados diretamente.

Alocação Indexada - Esquema Combinado: UNIX UFS

UFABC

- Os próximos 3 ponteiros apontam para blocos indiretos
  - O primeiro aponta para um bloco de indireção única que, por sua vez, é um bloco de índice contendo endereços dos blocos que contém dados (e não os dados propriamente ditos)
  - O segundo aponta para um bloco de indireção dupla, que contém os endereços de blocos de indireção única que por sua vez aponta para blocos que contém dados do arquivo
  - O terceiro contém um endereço para um bloco de indireção tripla

Alocação Indexada - Esquema Combinado: UNIX UFS



Desempenho

UFABC

- Melhor método é dependente do tipo de acesso aos arquivos
  - Alocação contígua é ótima para acessos sequenciais ou aleatórios
  - Alocação encadeada é boa para sequencial, mas não para aleatório
    - Uma possibilidade é declarar o tipo de acesso no momento da criação do arquivo $\rightarrow$ seleciona alocação contígua ou encadeada
  - Alocação indexada é mais complicada
    - Acesso a um único bloco pode demandar 2 leituras de blocos de índices além da leitura do bloco propriamente dita

UFABC

- As vezes adicionar mais instruções na execução e evitar uma operação de E/S pode ser vantajoso
  - Intel Core I7 Extreme Edition 990x (2011), 3,46GHz = 159.000 MIPS
    - http://en.wikipedia.org/wiki/Instructions_per_second
  - Um disco típico opera a 250 E/S por segundo
    - 159.000 MIPS / 250 = 630 milhões de instruções no tempo necessário para fazer apenas uma instrução de E/S
  - SSDs opera aproximadamente 60.000 IOPS
    - 159.000 MIPS / 60.000 = 2.65 milhões de instruções por operação de E/S

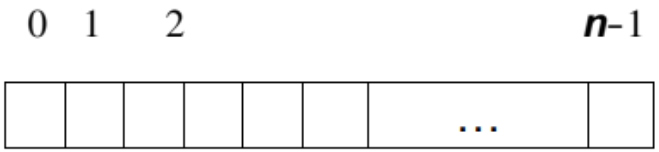
Gerenciamento de Espaço Livre

UFABC

- Sistemas de arquivos mantêm uma lista de posições livres
  - Podem se referir a blocos ou *clusters*
  - Aqui utilizaremos o termo *bloco* para simplificar
- Há quatro métodos principais
  - Mapa de bits / *bitmap*
  - Free list
  - Agrupamento
  - Contagem

UFABC

- Um mapa de bits ou um vetor de bits (n blocos)



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

- Cálculo do número do bloco
  (número de bits por palavra) * (número de palavras com valor 0) + offset do primeiro bit 1
- CPUs têm instruções específicas para devolver o offset dentro de uma palavra para o primeiro bit 1 (=eficiente)

UFABC

- O bitmap precisa de espaço extra

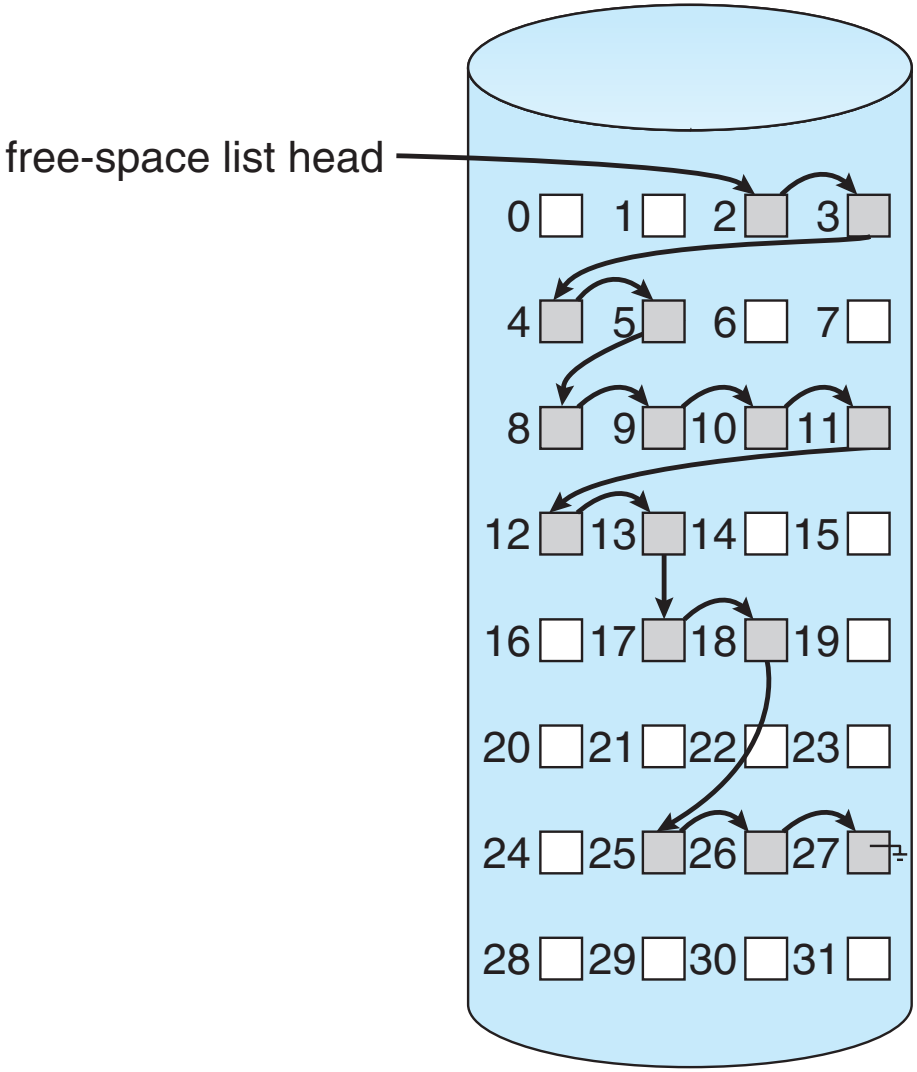
Exemplo

- Tamanho do bloco = 4KB = 2^{12} bytes
- Tamanho do disco = 2^{40} bytes = 1 terabyte
- $n = \frac{2^{40}}{2^{12}} = 2^{28}$ bits (32 MB)
- se os clusters forem de 4 blocos $\rightarrow$ 8MB de memória

- Por outro lado é fácil obter espaço para arquivos contíguos

Free List

- Lista encadeada - *freelist*
  - Não permite a obtenção de espaço contíguo de maneira fácil
  - Não desperdiça espaço
  - Não é preciso percorrer a lista completa (se o número de blocos na lista for salvo)



UFABC

Agrupamento

- Modifica o esquema de encadeamento para que a lista armazene o endereço dos próximos $n-1$ blocos livres no primeiro bloco além de um ponteiro para o próximo bloco que contém um conjunto com $n-1$ blocos livres (assim como o primeiro)

Contagem

- Como espaço em disco é continuamente alocado e liberado com alocação contígua, extents ou agrupamentos
  - Mantém endereços do primeiro bloco livre e conta quantos dos próximos blocos estão livres
  - A freelist contém então uma relação dos endereços e contagens

Técnicas para Eficiência e Desempenho

UFABC

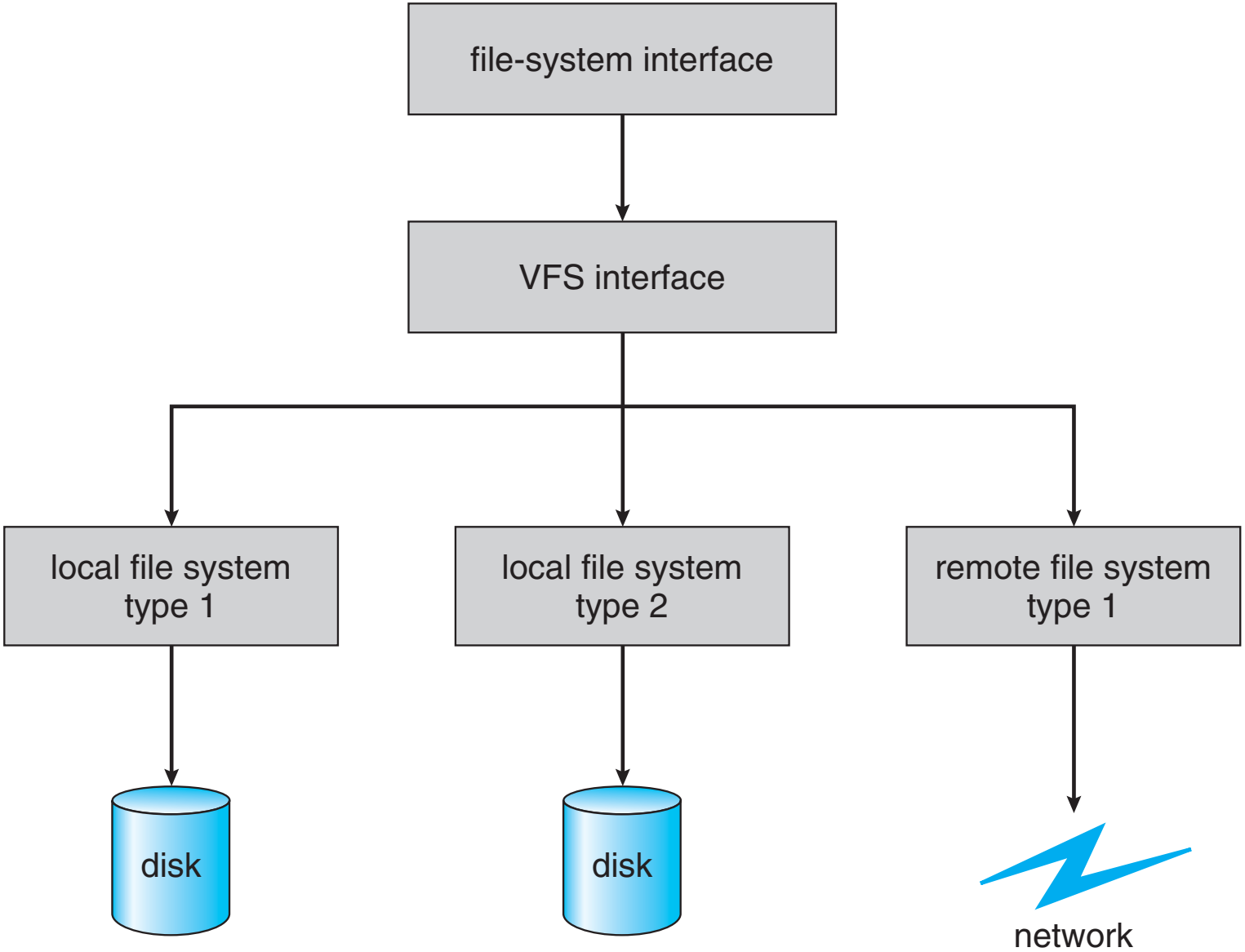
- Manter dados e metadados próximos
- Buffer cache - separe um espaço na memória para manter os blocos mais acessados
- Escritas síncronas são as vezes requisitadas por aplicações ou pelo próprio SO
  - Neste caso não é possível utilizar buffers ou caching - as escritas devem ser feitas ao disco antes de devolver o controle para a aplicação (write-through)
  - Por outro lado, boa parte das escritas podem ser feitas de maneira assíncrona em um buffer intermediário (write-back)
- Free-behind, read-ahead - Técnicas para acelerar o acesso sequencial
- Leituras podem são tipicamente mais lentas que escritas
  - Há exceções! NVMs por exemplo.

Sistemas de Arquivos Virtuais

UFABC

- O Linux provê uma maneira de implementar sistemas de arquivos de maneira orientada a objetos chamada Virtual File Systems (VFS)
- VFS permite que a mesma interface de chamadas de sistema (API) seja usada para diferentes tipos de sistemas de arquivos
  - Separa chamadas genéricas relativas a sistemas de arquivos de chamadas específicas
  - A implementação pode ser feita para os mais variados sistemas de arquivos (locais ou remotos)
    - Fornece a abstração de um vnode que é a generalização de um inode que pode conter um inode propriamente dito ou dados para conexão de um arquivo na rede
    - Despacha operações recebidas para as rotinas apropriadas do sistema de arquivos

A API é do VFS, e não de um sistema de arquivos específico.



Implementação do VFS

UFABC

- Tome como exemplo o Linux que tem 4 objetos distintos
  - inode, file, superblock, dentry
- O VFS define um conjunto de operações nos objetos que precisam ser implementadas
  - Todo objeto tem um ponteiro para uma tabela de funções
    - A tabela contém o endereço da função que implementa aquela função para aquele objeto
    - Por exemplo

```
1 int open(...) // Open a file
2 int close(...) // Close an already-open file
3 ssize t read(...) // Read from a file
4 ssize t write(...) // Write to a file
5 int mmap(...) // Memory-map a file
```

Recuperação

UFABC

- Verificação de consistência - Compara dados na estrutura de diretório com os blocos de dados em disco e tenta consertar possíveis inconsistências
  - Pode levar um bom tempo e as vezes falha
- Utilização de programas para fazer cópias de segurança (*backup*) para outro dispositivo de armazenamento como fita magnética, discos magnéticos, discos ópticos, ...)
  - Recupera arquivos perdidos ou corrompidos restaurando (*restore*) do backup