Sistemas de Arquivos

Profa. Kalinka Regina Lucas Jaquie Castelo Branco
kalinka@icmc.usp.br

Universidade de São Paulo

Novembro de 2020

O que o sistema de arquivos precisa fazer?

- Buscar blocos de disco livre
  - Precisa saber onde colocar os dados recém-gravados
- Rastrear quais blocos contêm dados para quais arquivos
  - Precisa saber de onde ler um arquivo
- Rastrear arquivos em um diretório
  - Encontre a lista de blocos do arquivo de acordo com seu nome
- Onde mantemos tudo isso?
  - **Em algum lugar no disco**

Estruturas de dados em disco

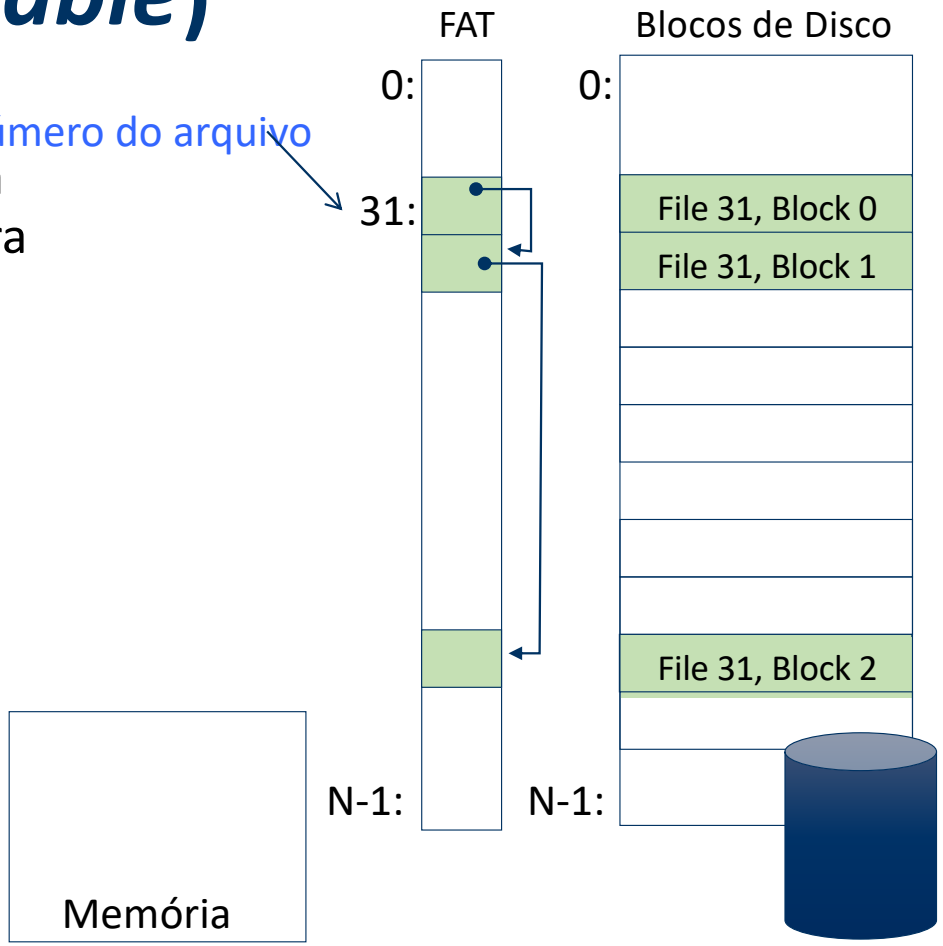
- Diferente das estruturas de dados na memória
- Acesse um bloco por vez
  - Não consegue ler / escrever de forma eficiente uma única palavra (deve ler o bloco completo)
  - Idealmente deseja padrões de acesso sequencial
- Durabilidade
  - Idealmente, o sistema de arquivos está em um estado significativo após o desligamento
  - Obviamente, nem sempre é o caso ...

FAT: *File Allocation Table*

- MS-DOS, 1977
- Ainda amplamente utilizada!

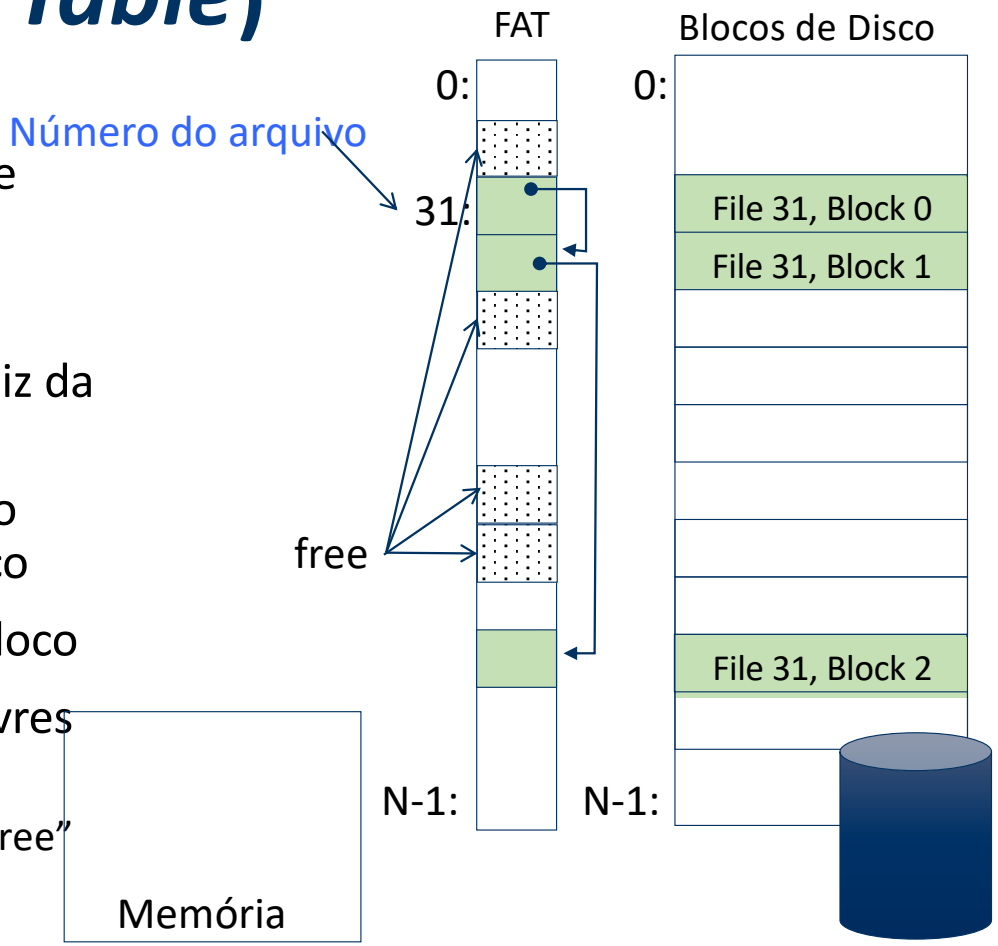
FAT (*File Allocation Table*)

- Suponha (por agora) que temos uma maneira de traduzir um caminho para um "número de arquivo"
  - ou seja, uma estrutura de diretório
- Armazenamento em disco é uma coleção de blocos
  - Basta manter os dados do arquivo (deslocamento o = <B, x>)
- Exemplo: file_read 31, <2, x>
  - Índice em FAT com número de arquivo
  - Siga a lista vinculada para os blocos
  - Leia o bloco do disco para a memória



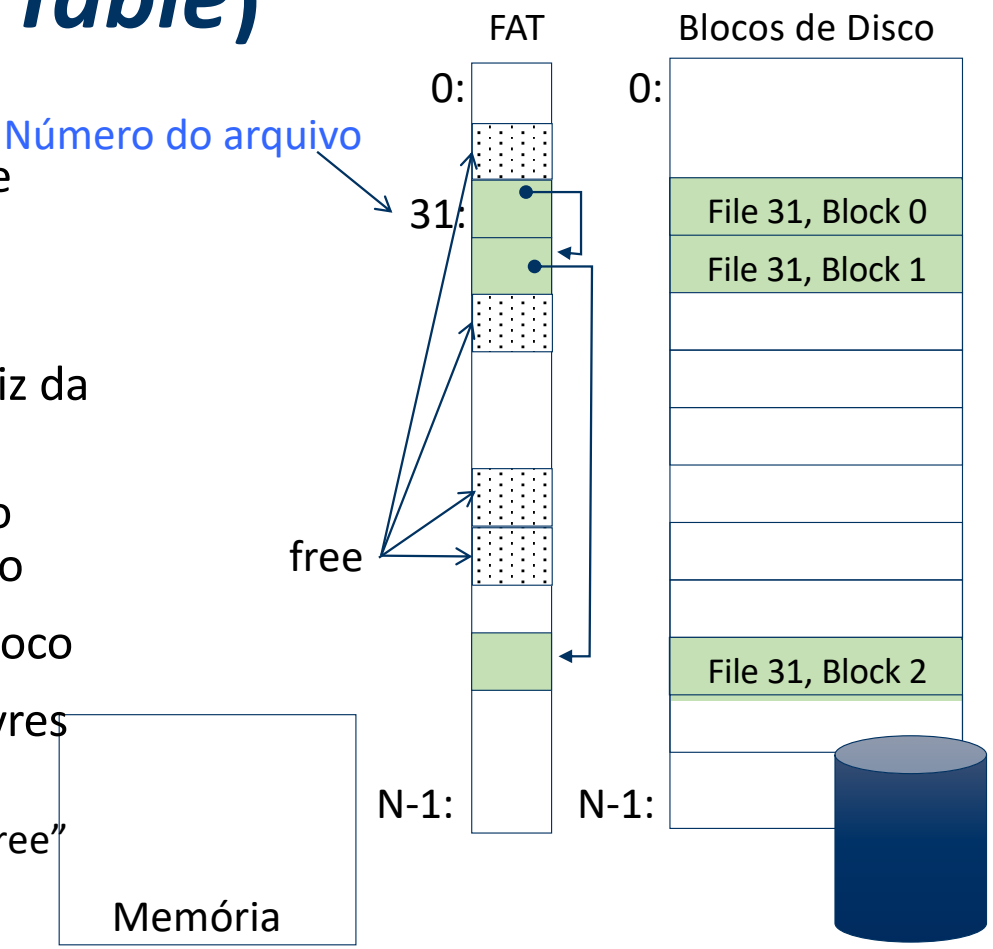
FAT (*File Allocation Table*)

- O arquivo é uma coleção de blocos de disco
- FAT é lista ligada 1-1 com blocos
- O número do arquivo é o índice da raiz da lista de blocos do arquivo
- Deslocamento do arquivo: número do bloco e deslocamento dentro do bloco
- Siga a lista para obter o número do bloco
- Blocos não usados marcados como livres
  - Pode exigir varredura para encontrar
  - Ou poderia usar uma lista de blocos “free”



FAT (*File Allocation Table*)

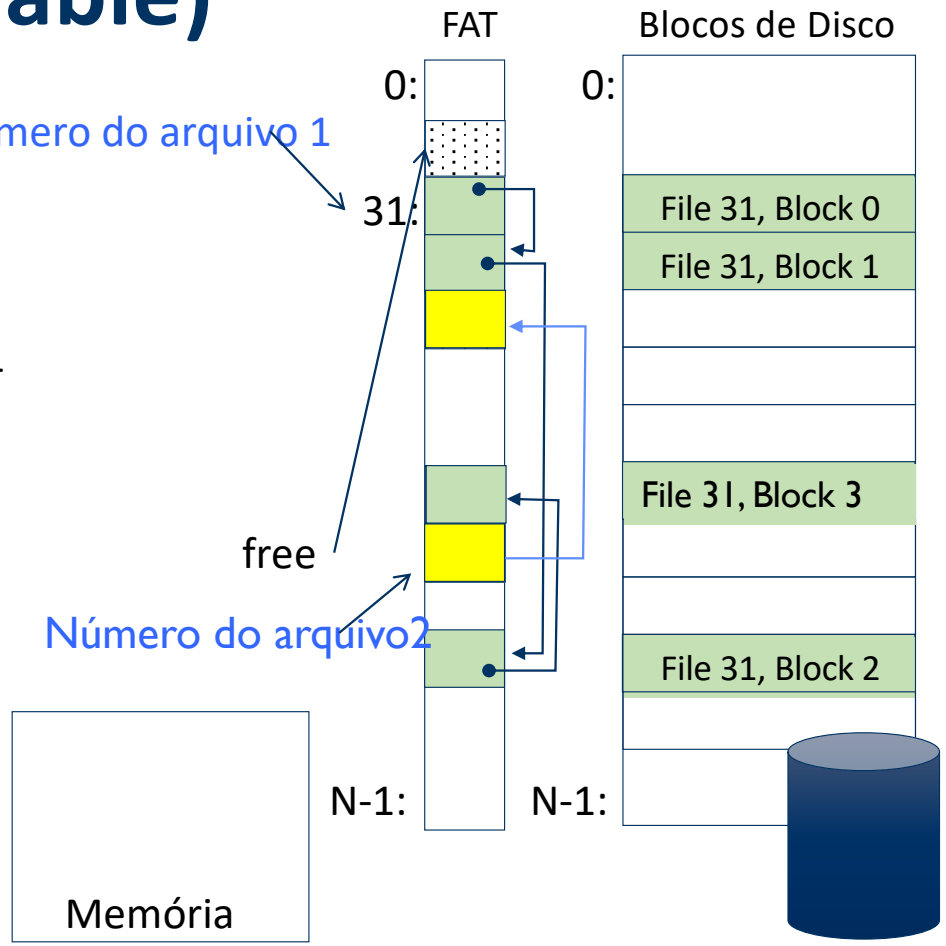
- O arquivo é uma coleção de blocos de disco
- FAT é lista ligada 1-1 com blocos
- O número do arquivo é o índice da raiz da lista de blocos do arquivo
- Deslocamento do arquivo: número do bloco e deslocamento dentro do bloco
- Siga a lista para obter o número do bloco
- Blocos não usados marcados como livres
  - Pode exigir varredura para encontrar
  - Ou poderia usar uma lista de blocos “free”



FAT (File Allocation Table)

- Onde o FAT é armazenado?
  - Em disco
- Como formatar um disco?
  - Zere os blocos, marque as entradas FAT como ”free"
- Como formatar um disco rapidamente?
  - Marcar entradas FAT como “free”

- Simples: pode implementar no *firmware* do dispositivo



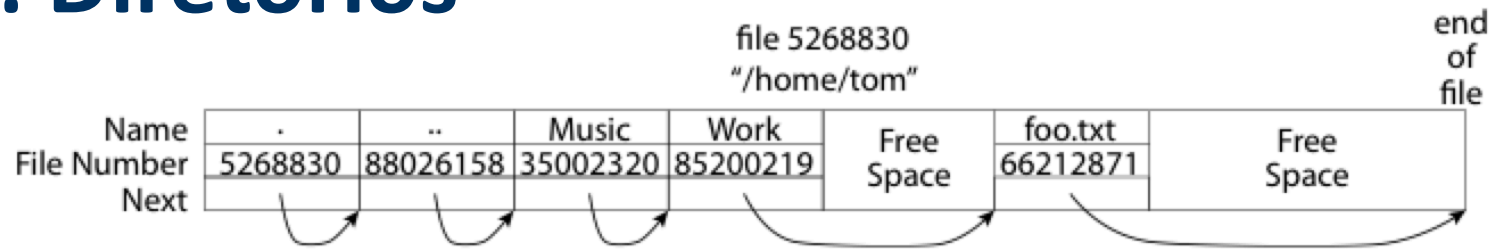
Como obter o número do arquivo?

- Procure na estrutura do diretório
- Um diretório é um arquivo que contém os mapeamentos <file_name: file_number>
  - O número do arquivo pode ser um arquivo ou outro diretório
  - O sistema operacional armazena o mapeamento no diretório em um formato que ele interpreta
  - Cada mapeamento <file_name: file_number> é chamado de entrada de diretório

- O processo não tem permissão para ler os bytes brutos de um diretório
  - A função de leitura não funciona em um diretório
  - Em vez disso, consulte **readdir**, que itera sobre o mapa sem revelar os bytes brutos

- Por que o sistema operacional não deve permitir que os processos leiam/gravem os bytes de um diretório?

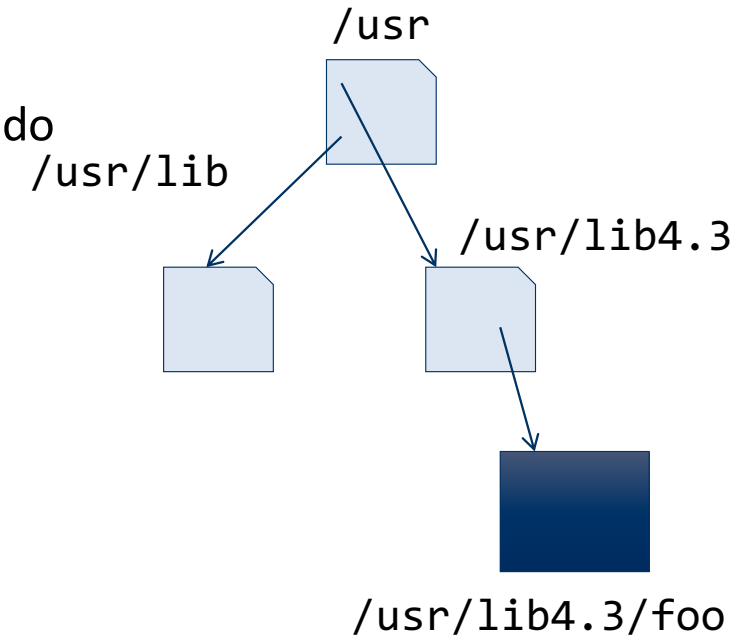
FAT: Diretórios



- Um diretório é um arquivo que contém os mapeamentos <file_name: file_number>
- Espaço livre para novas entradas
- Em FAT: os atributos do arquivo são mantidos no diretório (!!!)
- Cada diretório é uma lista vinculada de entradas
- Onde você encontra o diretório raiz ("/")?

Abstração de Diretório

- Os diretórios são arquivos especializados
  - Conteúdo: lista de pares <nome do arquivo, número do arquivo>
- Chamadas de sistema para acessar diretórios
  - abrir / criar / readdir atravessar a estrutura
  - mkdir / rmdir adicionar / remover entradas
  - ligar / desligar (rm)



- suporte libc
  - `DIR * opendir (const char *dirname)`
  - `struct dirent * readdir (DIR *dirstream)`
  - `int readdir_r (DIR *dirstream, struct dirent *entry, struct dirent **result)`

FAT Estrutura de Diretório

- Quantos acessos ao disco para resolver “/my /book/count”?
  - Ler o cabeçalho do arquivo para root (ponto fixo no disco)
  - Leia no primeiro bloco de dados para root
    - Tabela de pares de nome / índice de arquivo. Pesquise linearmente - ok, pois os diretórios geralmente são muito pequenos
  - Leia o cabeçalho do arquivo para “my”
  - Leia o primeiro bloco de dados para “my”; procure por “book”
  - Leia o cabeçalho do arquivo para ”book"
  - Leia no primeiro bloco de dados para “book”; procure por “count”
  - Leia o cabeçalho do arquivo para “count”
- Diretório de trabalho atual: ponteiro por espaço de endereço para um diretório usado para resolver nomes de arquivo
  - Permite que o usuário especifique o nome do arquivo relativo em vez do caminho absoluto (digamos CWD = “/my/book” pode resolver “count”)

Muitos buracos de segurança enormes do FAT!

- FAT não tem direitos de acesso
- FAT não tem cabeçalho nos blocos de arquivo
- Apenas fornece um índice para o FAT
  - (número do arquivo = número do bloco)

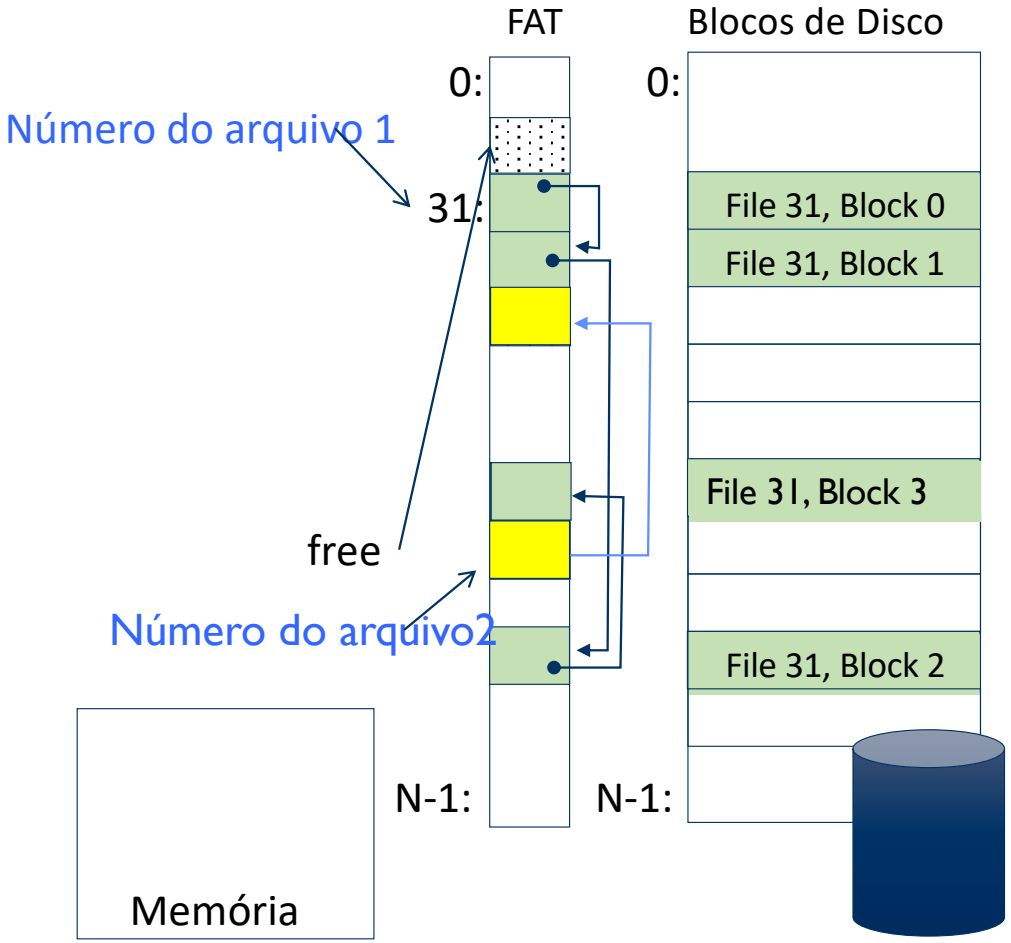
Fatores críticos no projeto do sistema de arquivos

- Desempenho dos discos (rígidos) !!!
  - Maximize o acesso sequencial, minimiza as buscas
- Abra antes de ler / escrever
  - Pode realizar verificações de proteção e pesquisar onde o recurso de arquivo real está, com antecedência
- O tamanho é determinado à medida que são usados !!!
  - Pode escrever (ou ler zeros) para expandir o arquivo
  - Comece pequeno e cresça, preciso abrir espaço
- Organizado em diretórios
  - Qual estrutura de dados (em disco) para isso?
- Necessidade de alocar / liberar blocos com cuidado
  - Para que o acesso permaneça eficiente

FAT Discussões

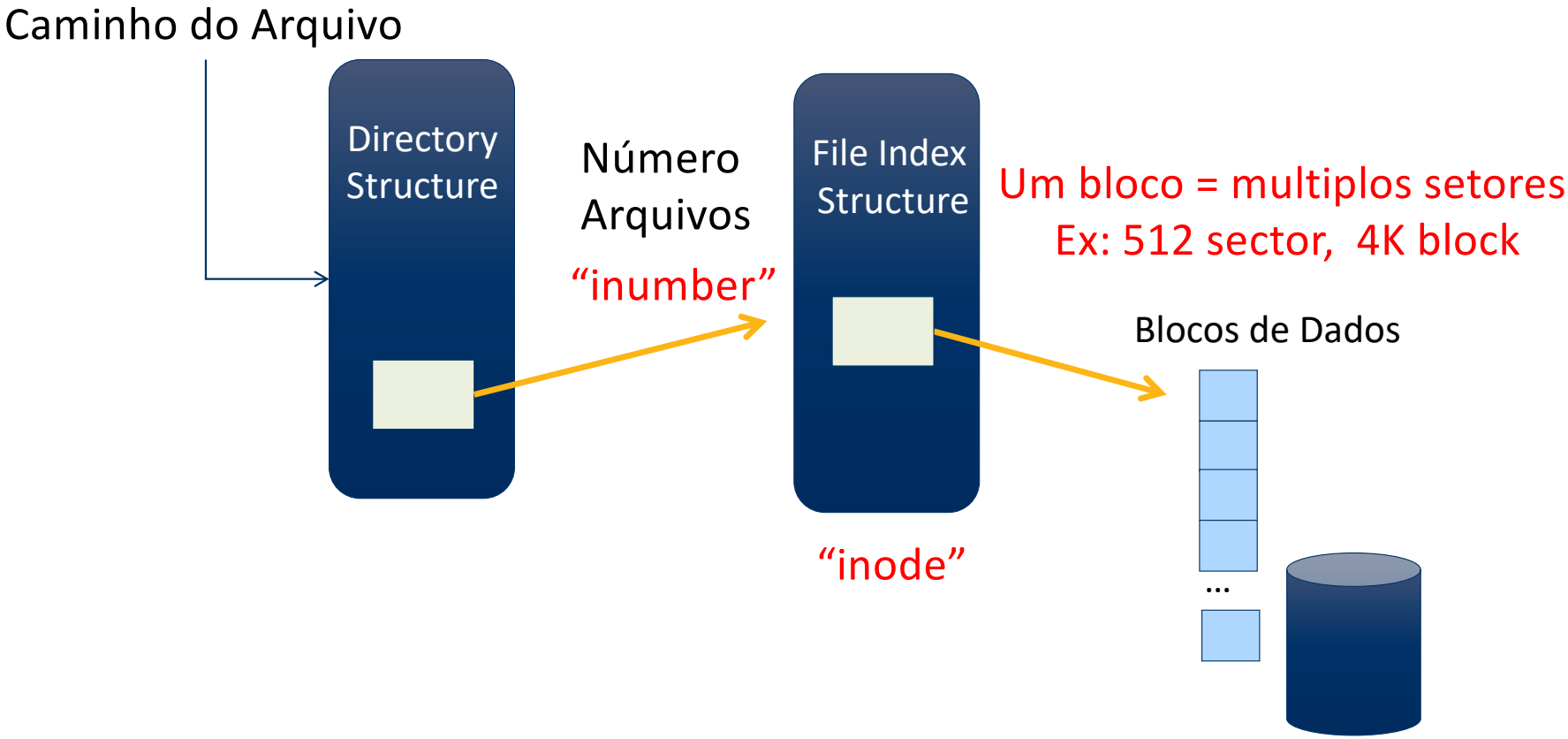
Suponha que você comece com o número do arquivo:

- Hora de encontrar o bloco?
- Layout de bloco para arquivo?
- Acesso sequencial?
- Acesso aleatório?
- Fragmentação?
- Arquivos pequenos?
- Arquivos grandes?

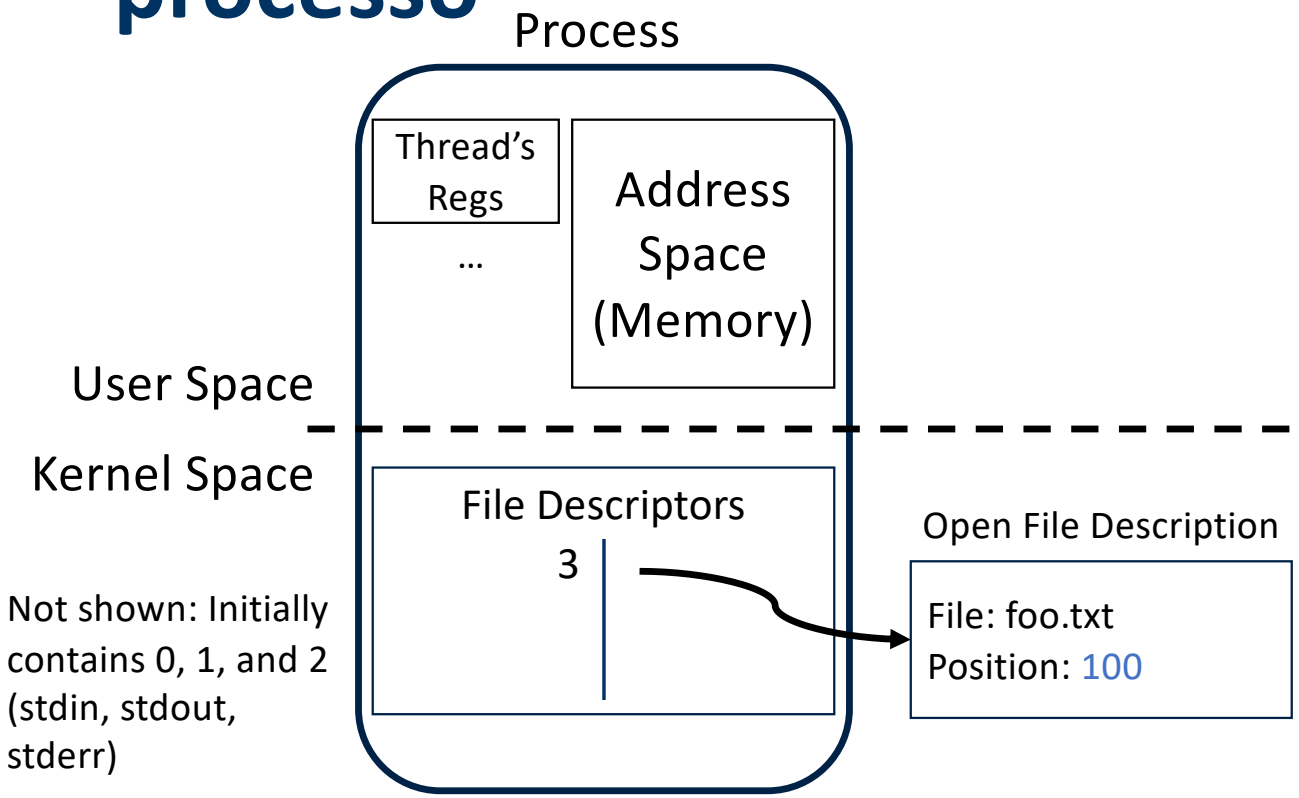


Organização do Sistema de Arquivo

Componentes do Sistema de Arquivos



Lembre-se: representação abstrata de um processo



Suponha que nós executemos um open em (“foo.txt”)

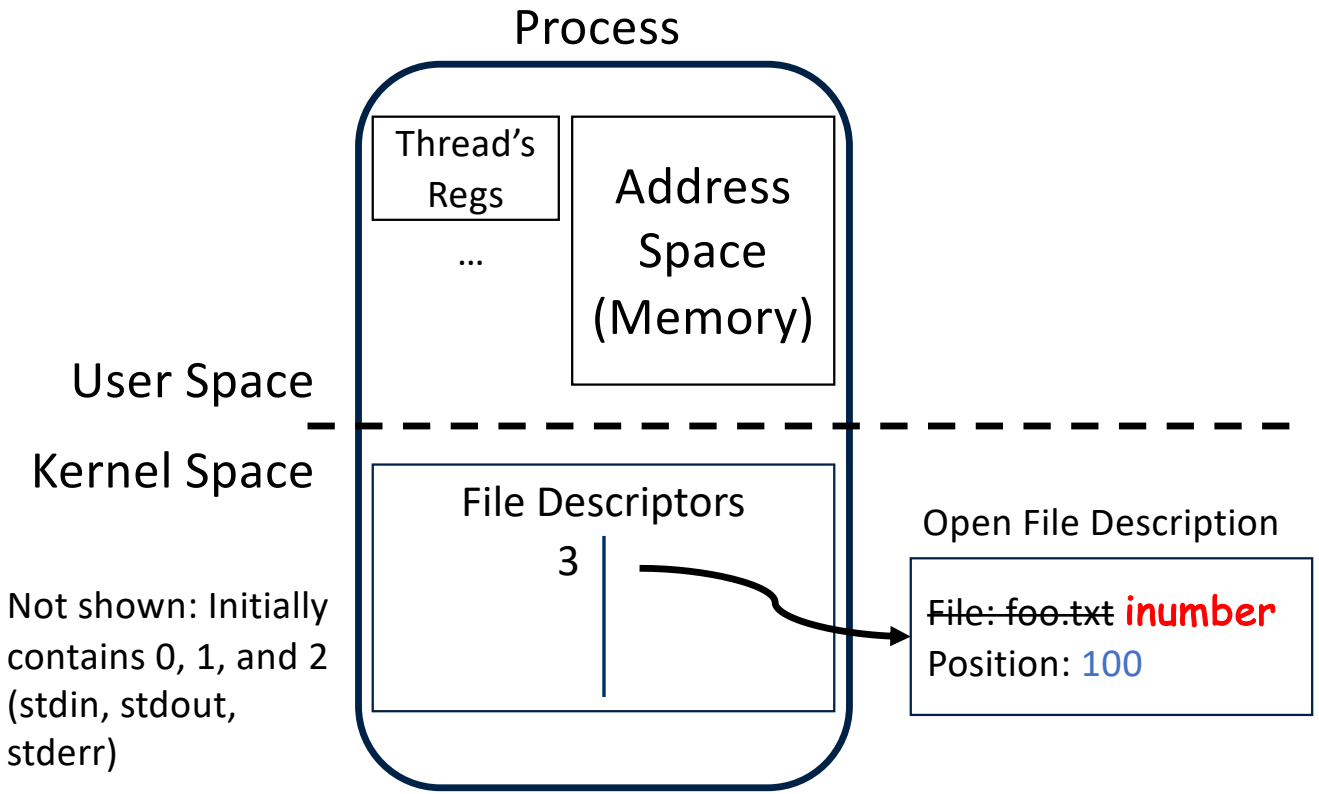
e que o resultado é 3

Em seguida, suponha que executemos,

`read(3, buf, 100)`

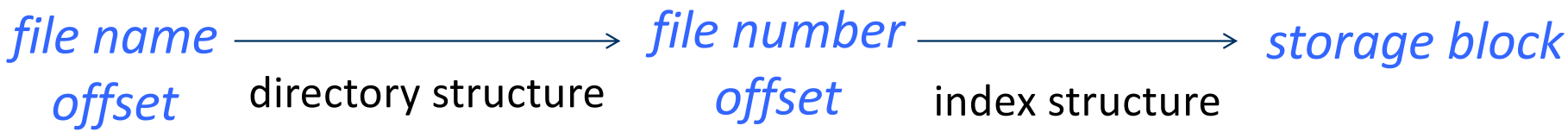
e que o resultado é 100

Componentes de um Sistema de Arquivos



A descrição do arquivo aberto é melhor descrita lembrando-se do inumber (número do arquivo) do arquivo, não de seu nome

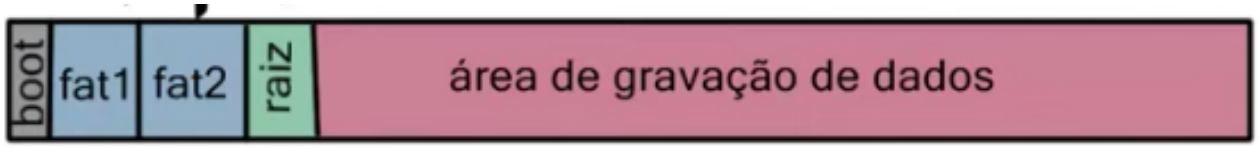
Componentes de um Sistema de Arquivos



- Open executa resolução de nomes
  - Traduz o nome do caminho em um "número de arquivo"
- Ler e escrever operam no número do arquivo
  - Use o número do arquivo como um "índice" para localizar os blocos

- **4 componentes: diretório, estrutura de índice, blocos de armazenamento, mapa de espaço livre**

Composição FAT



1- BOOT (*Master Boot Record*) no caso de sistema inicializável (tamanho, setores por clusteres, onde começam as fats, raiz, etc) - Configurações

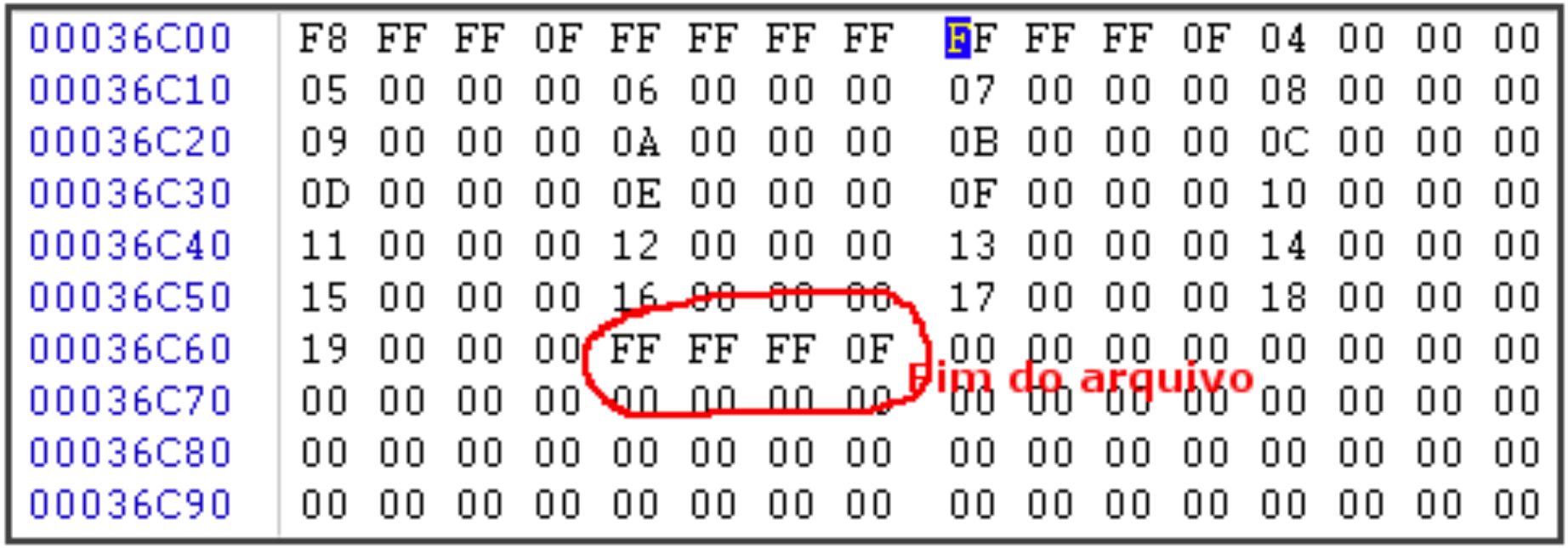
2- Depois vem o primeiro mapa do FAT (FAT 1)

3- Após o fim do Fat 1, vem o Fat 2 que é uma cópia fiel, e útil para recuperar *crash* de sistema.

4- Após o Fat 1 , Fat 2 , vem a parte de ROOT (*Raiz da unidade*).

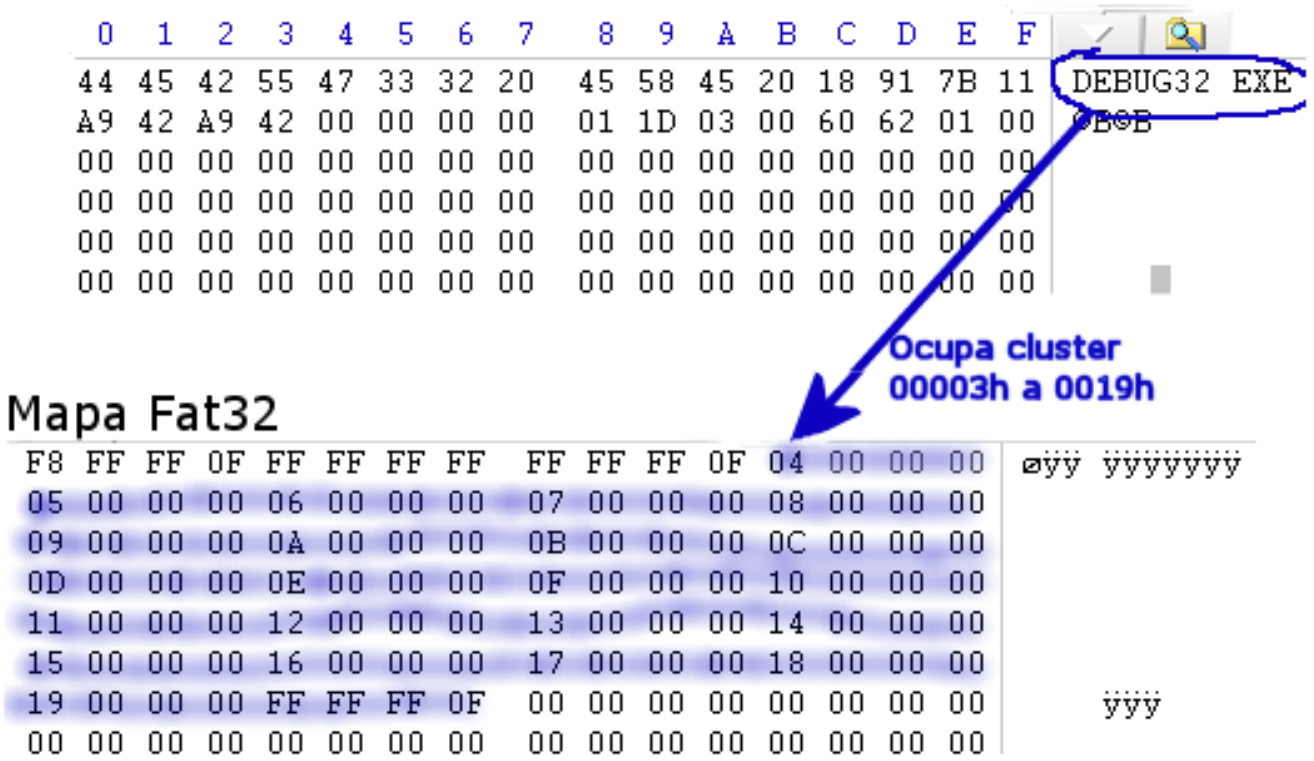
5- Depois da Raiz vem a área de gravação de dados, e a maior da unidade (onde estão os arquivo – foo.txt) – vê-se o nome do arquivo e o cluster onde inicia.

Composição FAT

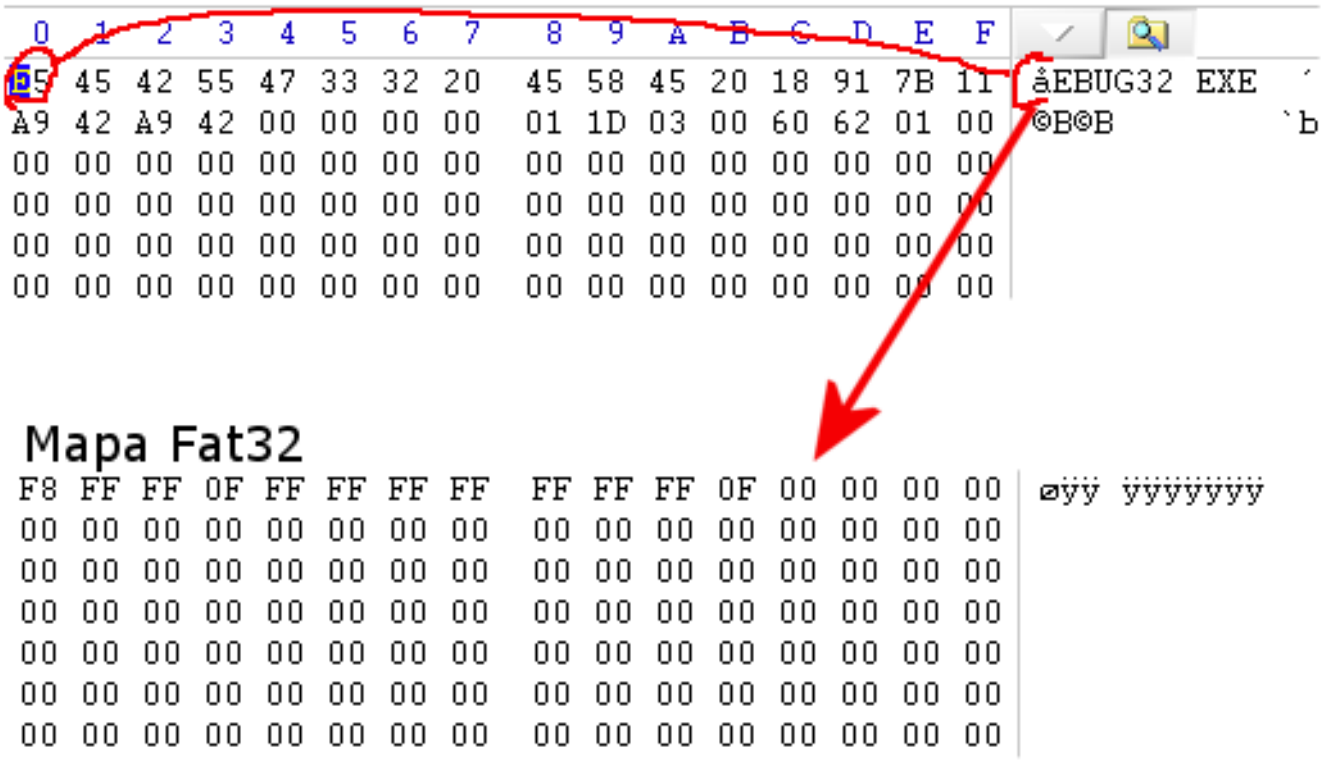


Composição FAT

Composição FAT



Composição FAT



Unix File System (Berkeley FFS)

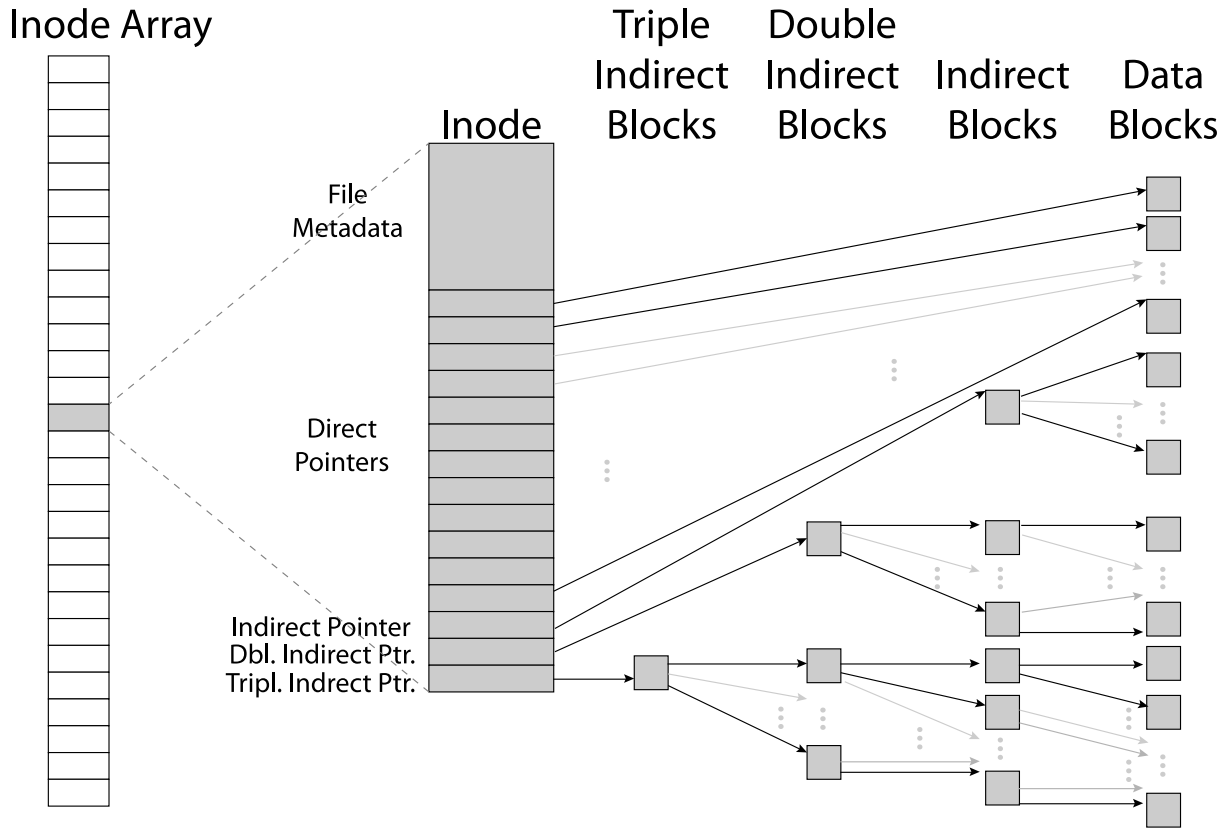
Inodes no Unix (incluindo Berkeley FFS)

- O formato **inode** original apareceu no BSD 4.1

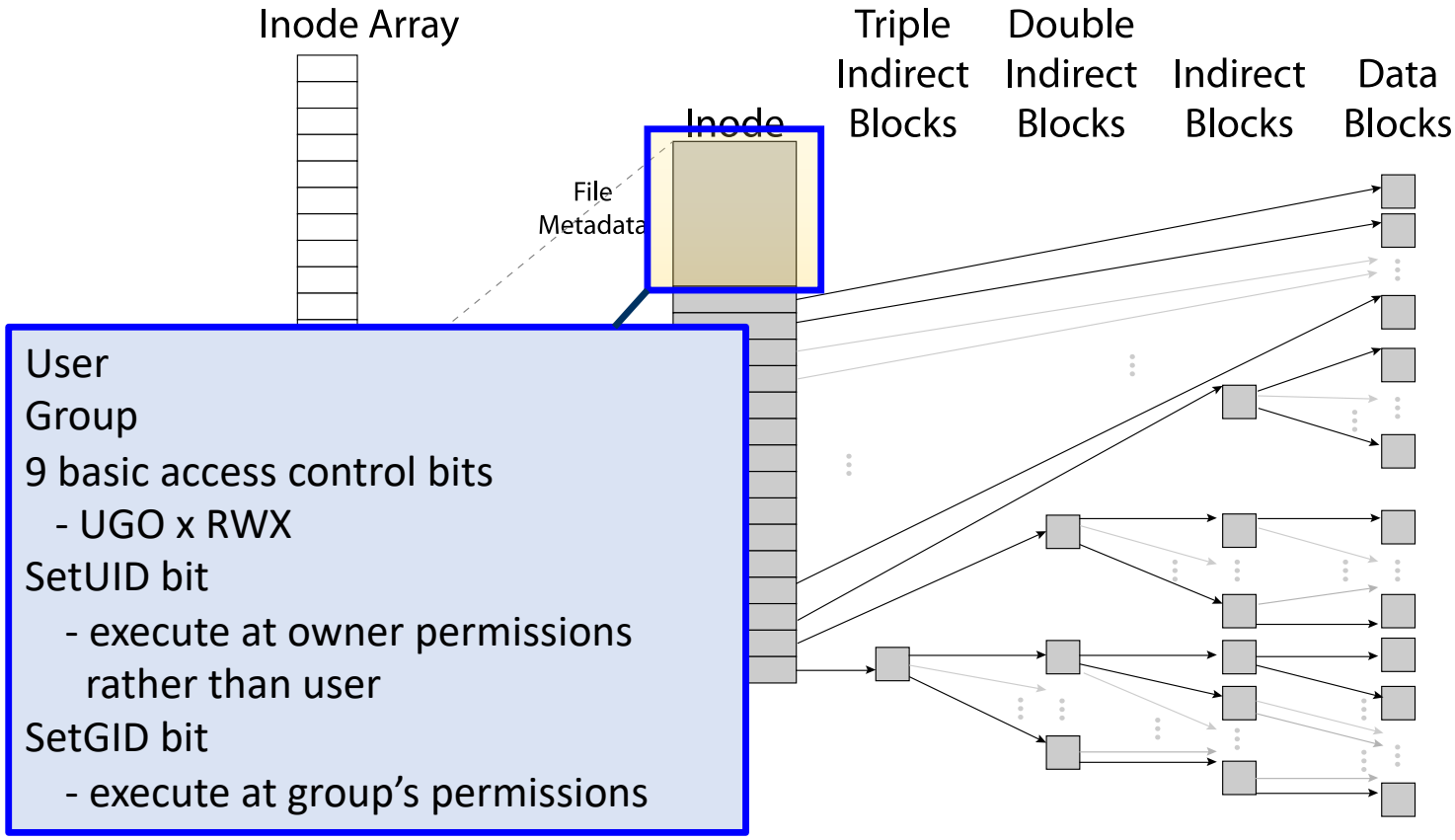
- A estrutura do índice é uma matriz de inodes
  - O número do arquivo (inumber) é um índice na matriz de inodes
  - Cada inode corresponde a um arquivo e contém seus metadados

- Inode mantém uma estrutura de árvore de vários níveis para encontrar blocos de armazenamento para arquivos
  - Ótimo para arquivos pequenos e grandes
  - Árvore assimétrica com blocos de tamanho fixo

Estrutura do Inode



Estrutura do Inode



Characteristics of Files

A Five-Year Study of File-System Metadata

NITIN AGRAWAL
University of Wisconsin, Madison
and
WILLIAM J. BOLOSKY, JOHN R. DOUCEUR, and JACOB R. LORCH
Microsoft Research

Published in FAST 2007

Observação #1: A maioria dos arquivos são pequenos

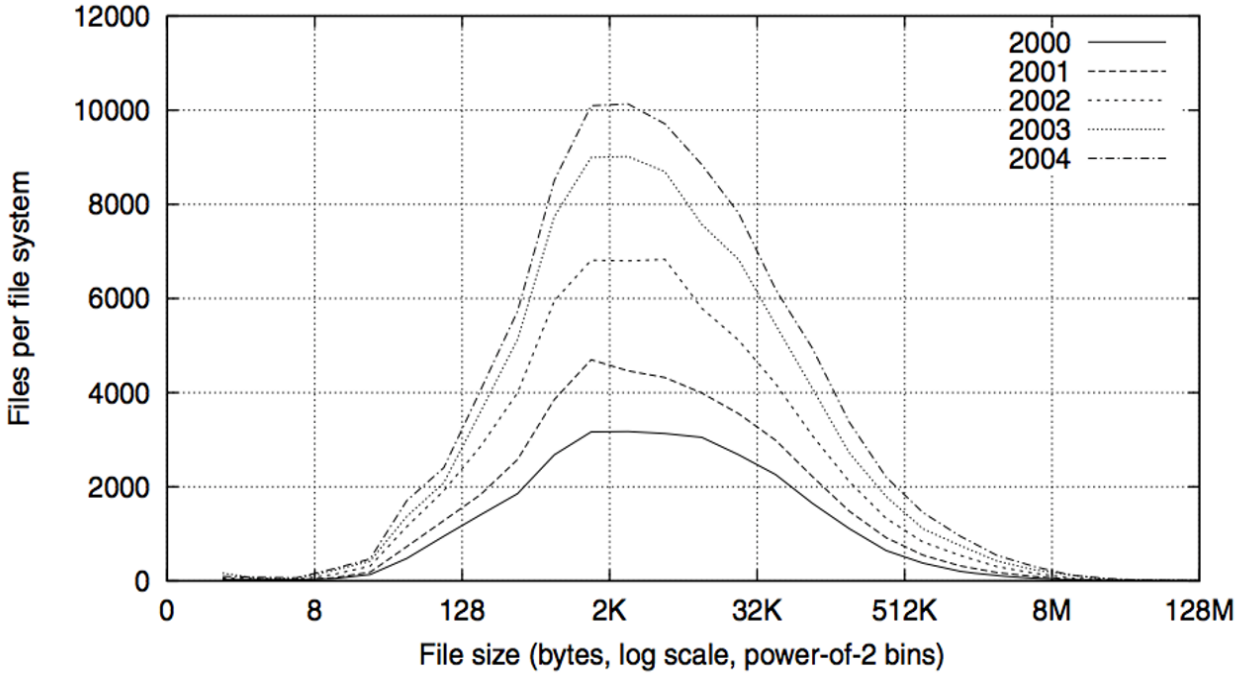


Fig. 2. Histograms of files by size.

Observação #2: A maioria dos bytes está em arquivos grandes

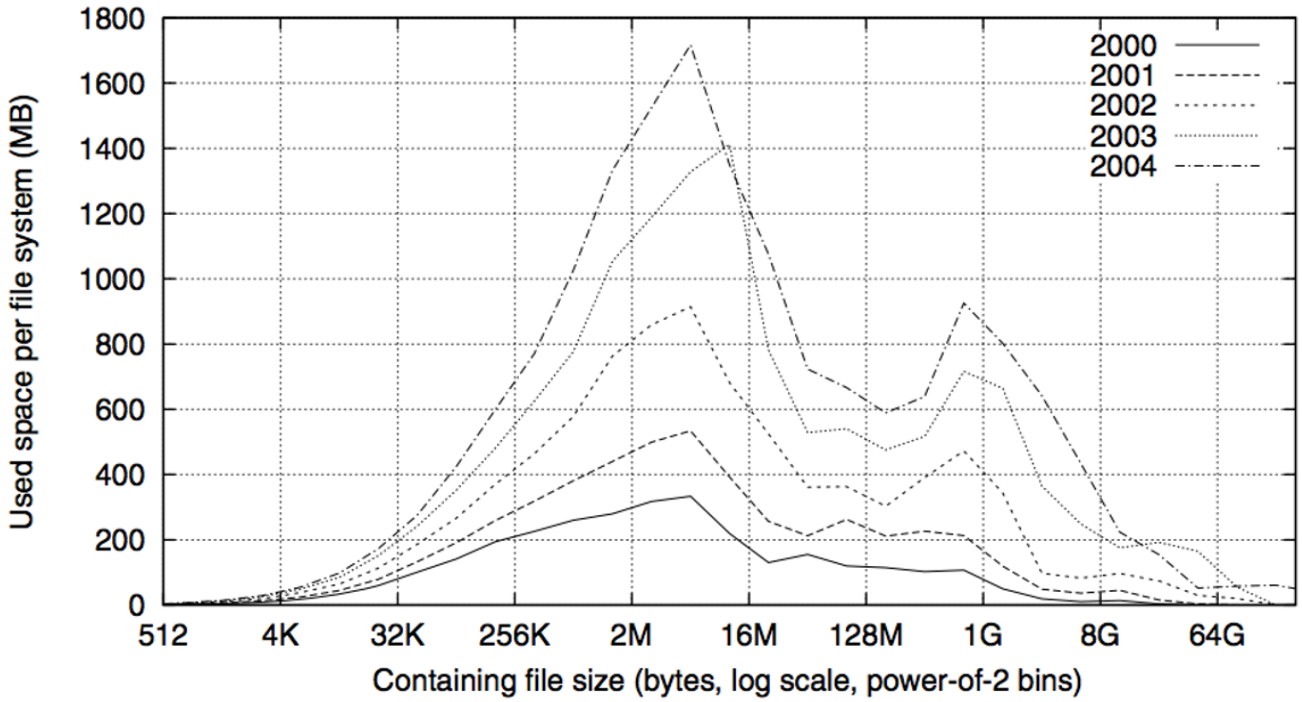
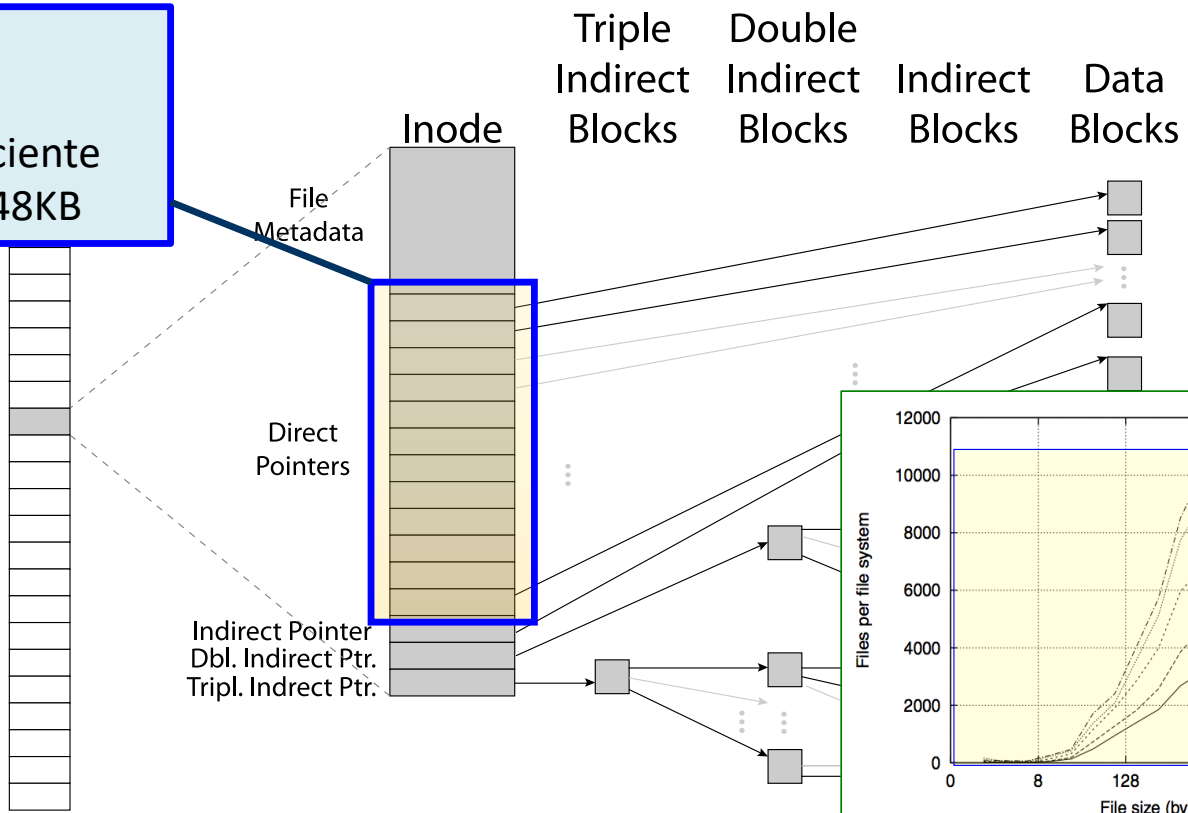


Fig. 4. Histograms of bytes by containing file size.

Arquivos pequenos: 12 Ponteiros diretos para blocos de dados

Ponteiros Diretos

4kB blocos ⇒ suficiente para arquivos até 48KB



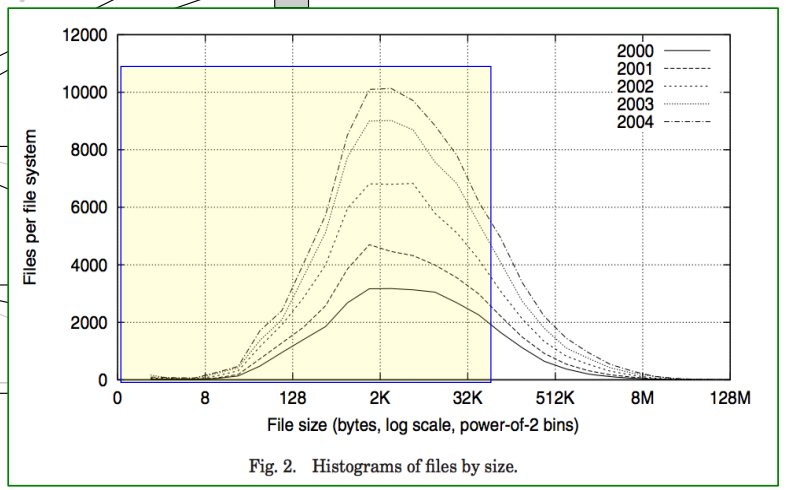


Fig. 2. Histograms of files by size.

Large Files: 1-, 2-, 3-level indirect pointers

Ponteiros indiretos
- apontar para um bloco de disco contendo apenas ponteiros
- 4 kB blocos => 1024 ptrs
  => 4 MB @ level 2
  => 4 GB @ level 3
  => 4 TB @ level 4

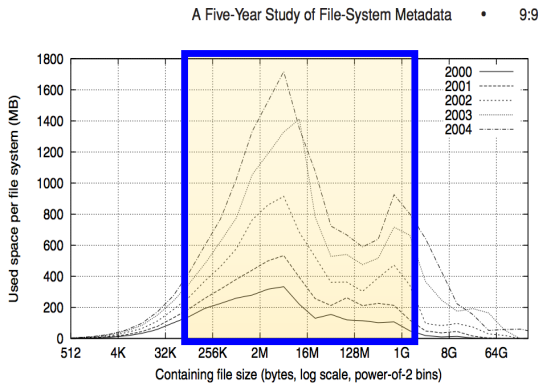
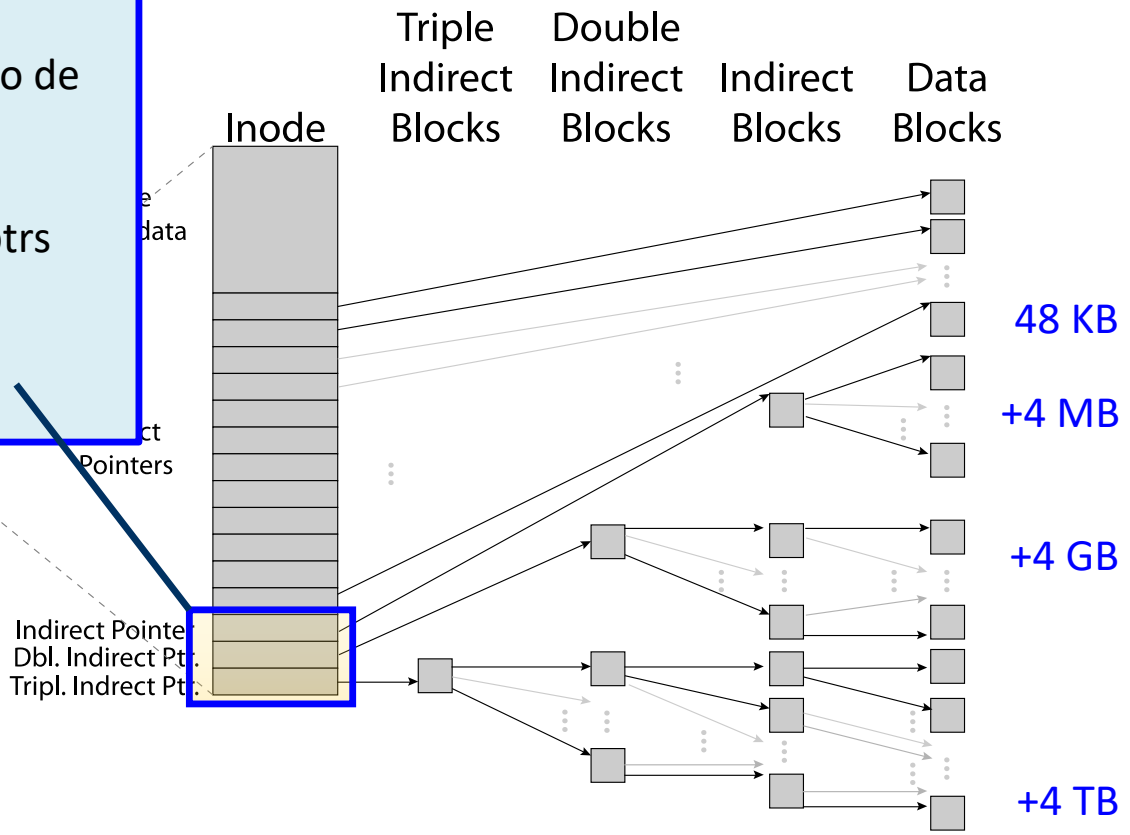
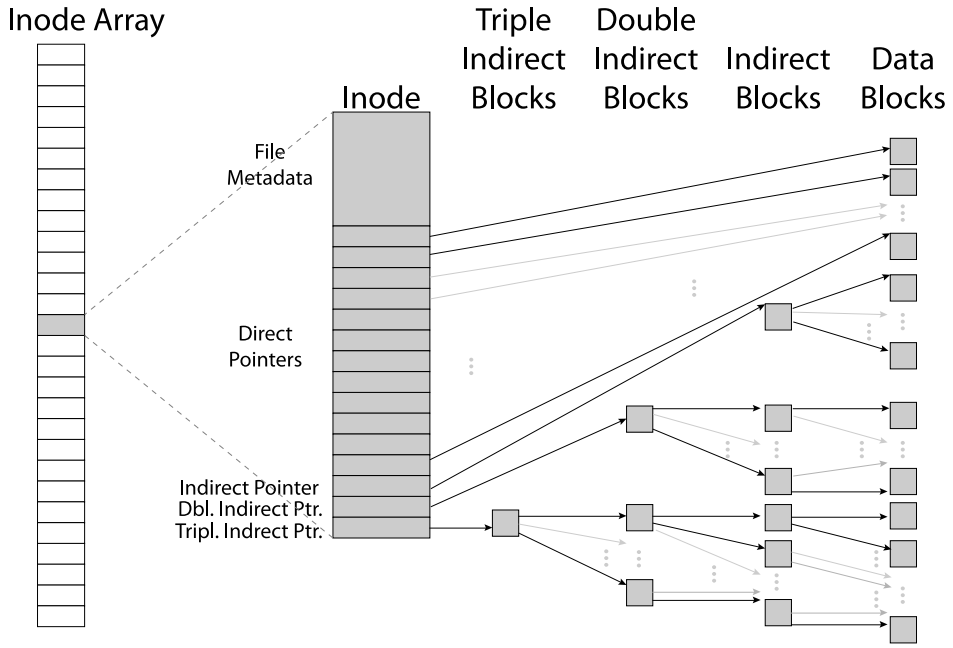


Fig. 4. Histograms of bytes by containing file size.

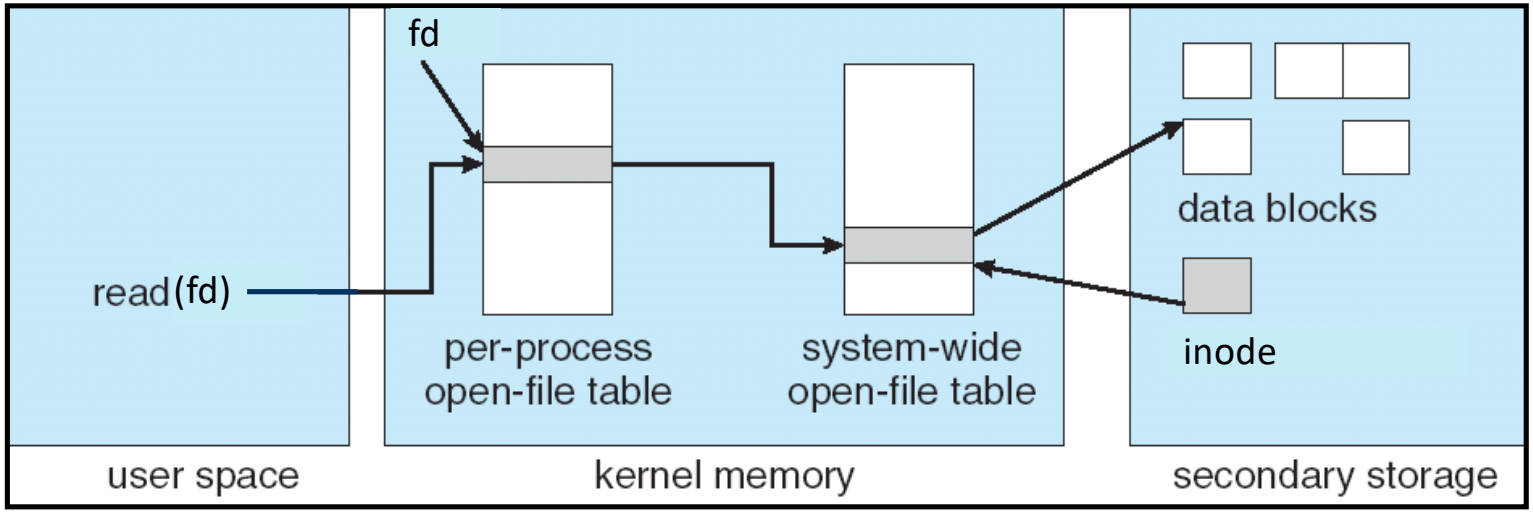


Juntando tudo: índice no disco

- Arquivo de amostra em formato indexado em vários níveis:
  - 10 ptrs diretos, blocos de 1K
  - Quantos acessos para o bloco # 23? (assume o cabeçalho do arquivo acessado ao abrir)?
    - Dois: um para bloco indireto, um para dados
- Que tal o bloco # 5?
  - Um: um para dados
- Bloco # 340?
  - Três: bloqueio indireto duplo, bloqueio indireto e dados



Estruturas do sistema de arquivos na memória



- Abrir syscall: encontrar inode no disco a partir do nome do caminho (percorrendo diretórios)
  - Crie "in-memory inode" na tabela de arquivos abertos em todo o sistema
  - Uma entrada nesta tabela, não importa quantas instâncias do arquivo estejam abertas
- Syscalls de leitura / gravação procuram inode na memória usando o identificador de arquivo

Lembre-se: Fatores críticos no projeto do sistema de arquivos

- **Desempenho do disco (rígido) !!!**
  - **Maximize o acesso sequencial, minimize as buscas**
- Abra antes de ler / escrever
  - Pode realizar verificações de proteção e pesquisar onde o recurso de arquivo real está, com antecedência
- O tamanho é determinado à medida que são usados !!!
  - Pode escrever (ou ler zeros) para expandir o arquivo
  - Comece pequeno e cresça, preciso abrir espaço
- Organizado em diretórios
  - Qual estrutura de dados (em disco) para isso?
- Necessidade de alocar / liberar blocos com cuidado
  - Para que o acesso permaneça eficiente

Berkeley FFS: Otimizando para o disco

- BSD 4.1 tinha a estrutura inode que estudamos

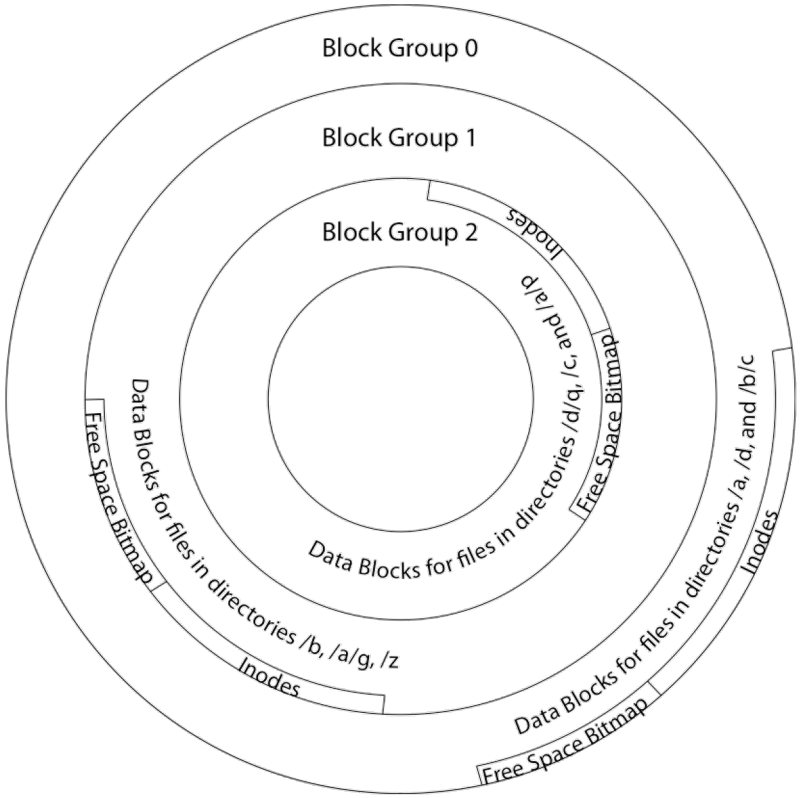
- BSD 4.2 (1984) incorporou políticas de alocação e co-alocação para melhorar o desempenho
  - Chamado FFS : sistema de arquivos rápido
  - Ideia principal: otimizar as propriedades do disco rígido subjacente (evitar buscas!)

Berkeley FFS: Onde os Inodes são armazenados?

- Versões posteriores do UNIX moveram as informações do cabeçalho para ficar mais perto dos blocos de dados
  - Frequentemente, inode para arquivo armazenado no mesmo "grupo de cilindros" do diretório pai do arquivo (faz com que um ls desse diretório seja executado rapidamente)
- Prós:
  - UNIX BSD 4.2 coloca bits de matriz de cabeçalho de arquivo em muitos cilindros
  - Para diretórios pequenos, pode caber todos os dados, cabeçalhos de arquivo, etc. no mesmo cilindro - sem buscas!
  - Cabeçalhos de arquivo muito menores do que o bloco inteiro (algumas centenas de bytes), então vários cabeçalhos buscados do disco ao mesmo tempo
  - Confiabilidade: aconteça o que acontecer com o disco, você pode encontrar muitos dos arquivos (mesmo se os diretórios estiverem desconectados)

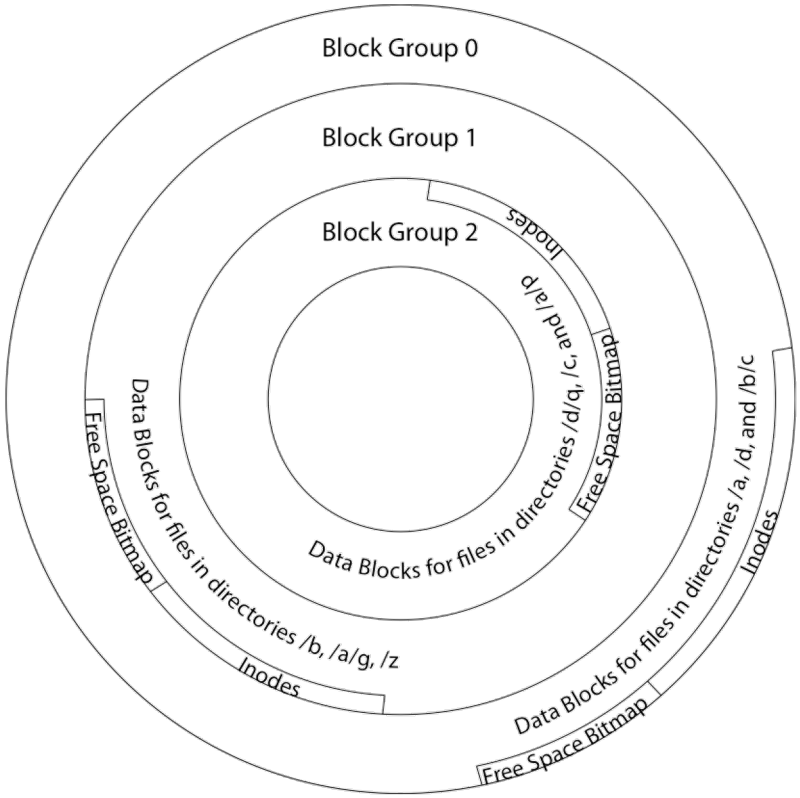
Berkeley FFS: Localidade

- O volume do sistema de arquivos é dividido em um conjunto de **grupos de blocos**
  - Conjunto fechado de trilhas (termo generalizado para grupos de cilindros)
- Blocos de dados, metadados e espaço livre intercalados dentro do grupo de blocos
  - Evita buscas enormes entre os dados do usuário e a estrutura do sistema
- Coloqua o diretório e seus arquivos em um grupo de blocos comum
  - Armazena-os perto um do outro

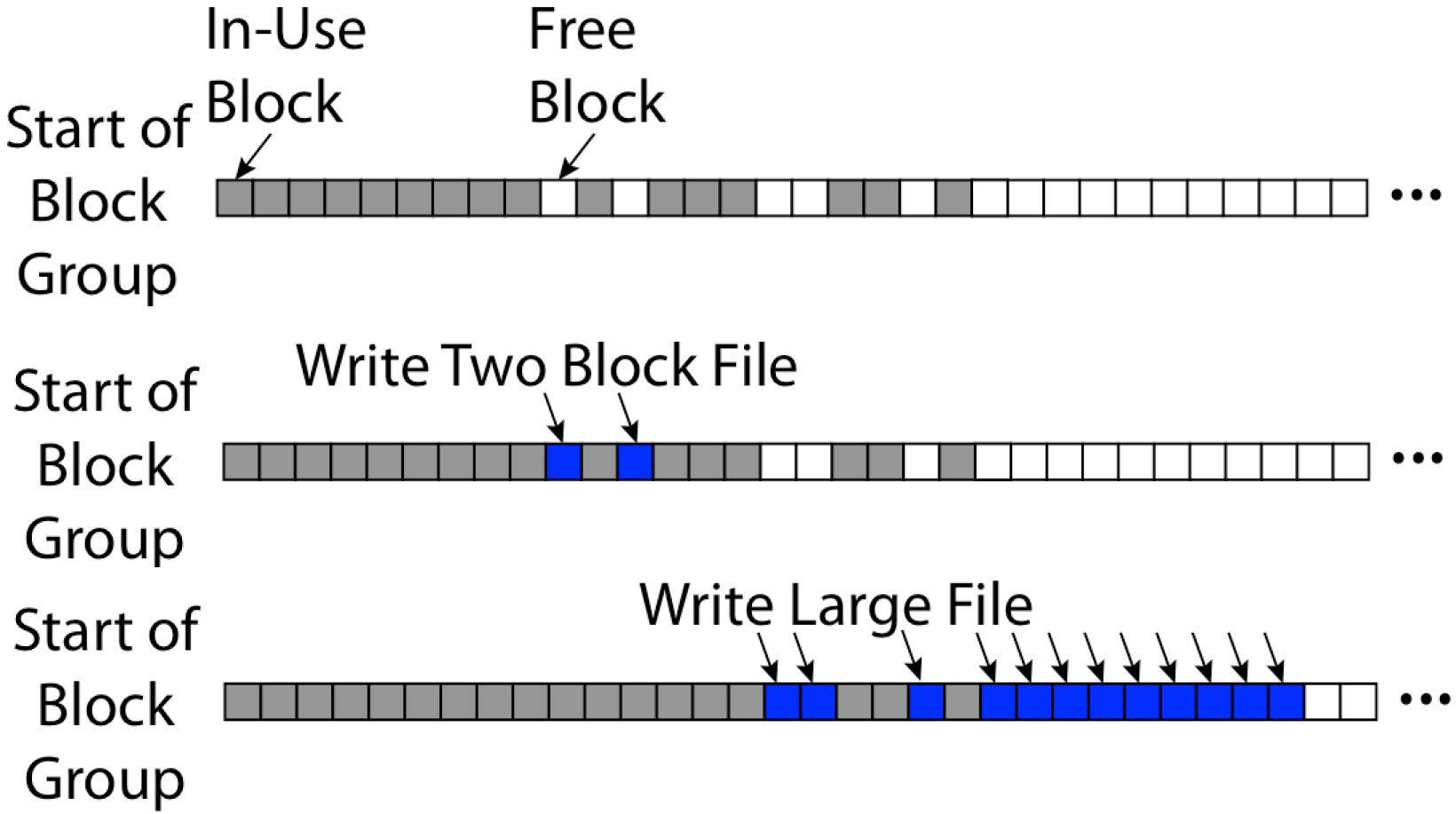


Unix BSD 4.2 (1984): Berkeley FFS

- Idéia: tornar o arquivo contíguo no disco
- Problema nº 1: quanto espaço contíguo deve ser alocado?
  - Não sei como os arquivos grandes vão crescer, com antecedência
- Solução: primeira alocação gratuita
  - Para expandir o arquivo, primeiro tente blocos sucessivos no bitmap e, em seguida, escolha um novo intervalo de blocos
  - Poucos buracos no início, grandes execuções sequenciais no final do grupo
  - Evita fragmentação
  - Layout sequencial para arquivos grandes
- Importante: mantenha 10% ou mais free!
  - Reserva espaço no Grupo do Bloco

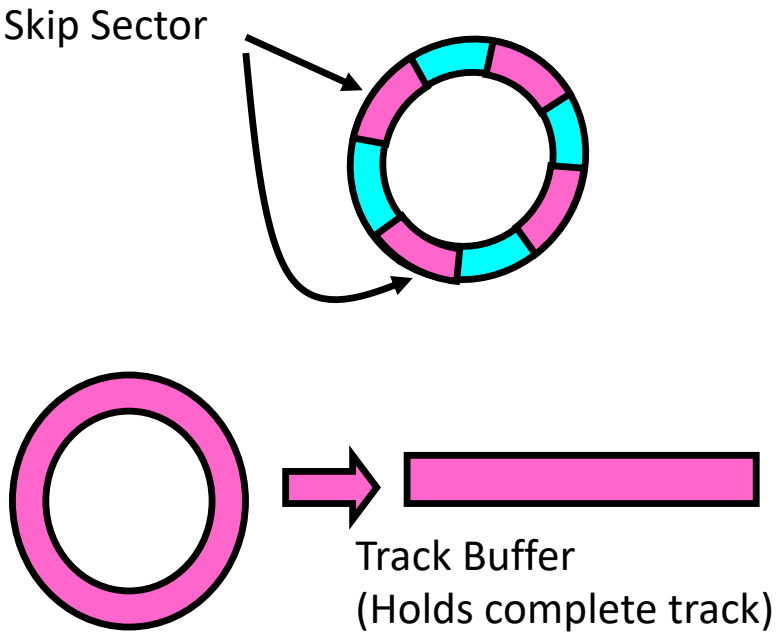


Berkeley FFS: Alocação de Bloco *First-Fit*



Unix BSD 4.2 (1984): Berkeley FFS

- Problema # 2: blocos ausentes devido ao atraso rotacional
  - Leia um bloco, mas quando você emitir ler para o próximo bloco, ele "acabou de passar" a cabeça do disco
- Solução # 1: pule o posicionamento do setor
  - Coloque blocos de um arquivo em cada outro bloco de uma trilha
- Solução 2: Leia mais à frente
  - Leia o próximo bloco antes que o usuário peça por ele
- Discos/controladores modernos fazem muitas coisas por baixo dos panos: buffers de trilha, filtragem de blocos defeituosos, etc ...



Avaliação de Berkeley FFS

+Armazenamento eficiente para arquivos grandes e pequenos

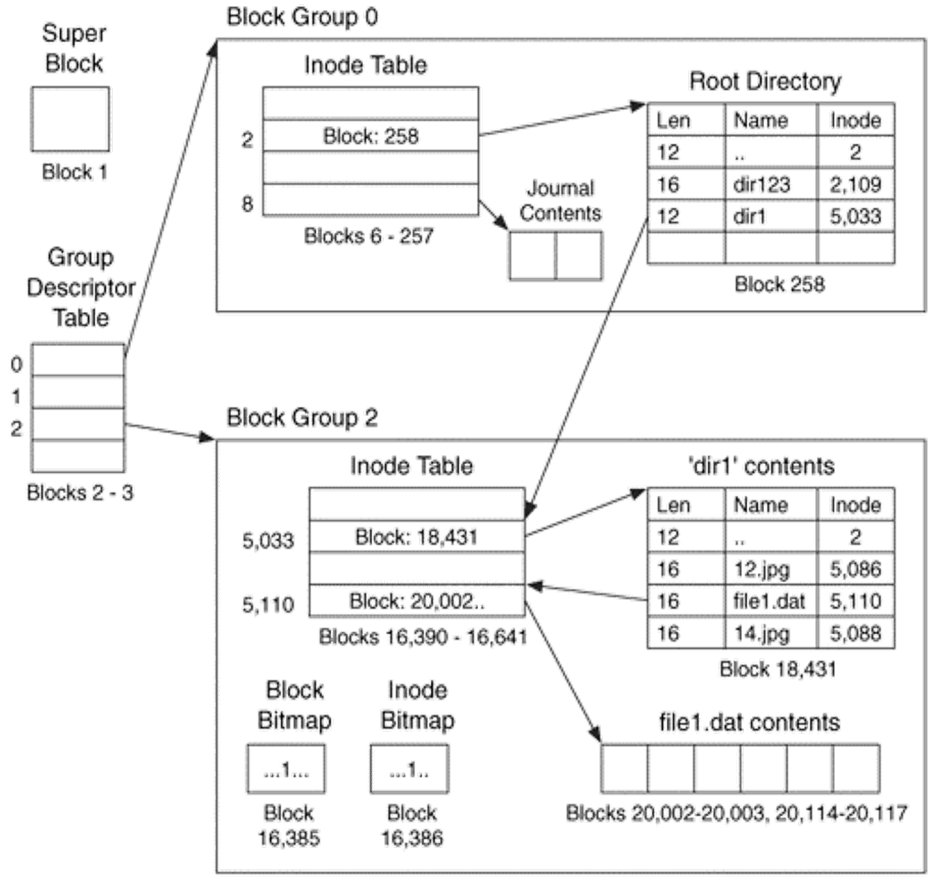
+Localidade para conteúdo e metadados do arquivo

- Ineficiente para arquivos pequenos
  - Por exemplo, um arquivo de um byte requer 8 KB de espaço em disco: inode e bloco de dados

- Codificação ineficiente para intervalos contíguos de blocos pertencentes ao mesmo arquivo (por exemplo, blocos 4815 - 162342)

Linux Exemplo: ext2/3 Layout do Disco

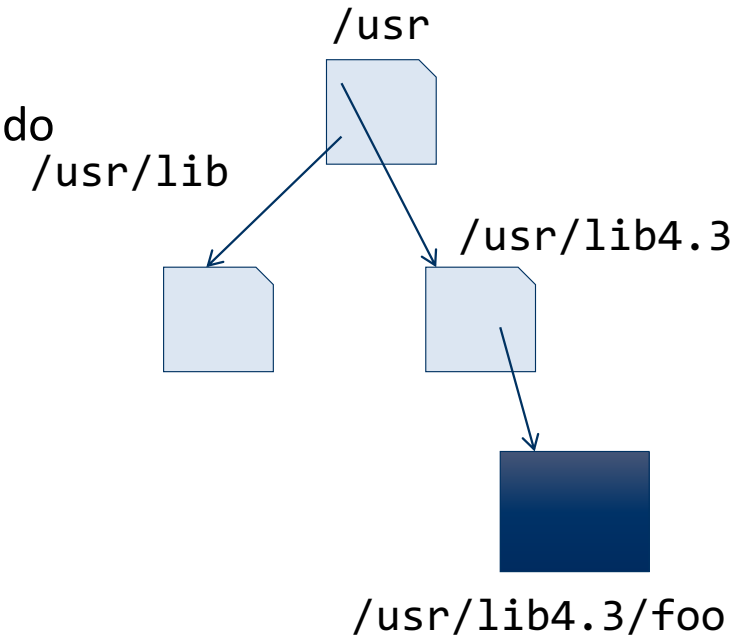
- Disco dividido em grupos de blocos
  - Fornece localidade
  - Cada grupo tem dois bitmaps de tamanho de bloco (blocos / inodes livres)
  - Tamanhos de bloco configuráveis no formato de tempo: 1K, 2K, 4K, 8K ...
- Estrutura real do inode semelhante a 4.2 BSD
- Ext3: Ext2 com registro em diário
  - Vários graus de proteção com sobrecarga comparável



- Example: create a file1.dat under /dir1/ in Ext3

Lembre-se: Abstração de Diretório

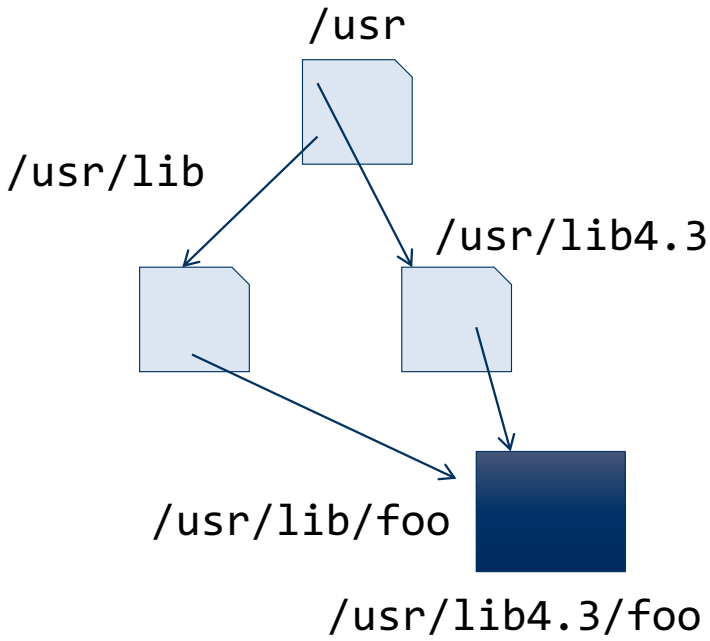
- Os diretórios são arquivos especializados
  - Conteúdo: lista de pares <nome do arquivo, número do arquivo>
- Chamadas de sistema para acessar diretórios
  - abrir / criar / readdir atravessar a estrutura
  - mkdir / rmdir adicionar / remover entradas
  - ligar / desligar (rm)



- suporte libc
  - `DIR * opendir (const char *dirname)`
  - `struct dirent * readdir (DIR *dirstream)`
  - `int readdir_r (DIR *dirstream, struct dirent *entry, struct dirent **result)`

Lembre-se: Abstração de Diretório

- **Hard link**: mapeamento de nome para arquivo na estrutura de diretório
- O primeiro link físico para um arquivo é feito quando o arquivo é inicialmente criado
- Crie links físicos extras para um arquivo com **link syscall**
- Remover links com ***unlink*** (rm)
- Quando o conteúdo do arquivo pode ser excluído?
  - Quando não houver mais links físicos para o arquivo
  - Inode mantém a contagem de referência para este propósito



Soft Links (Links Simbólicos)

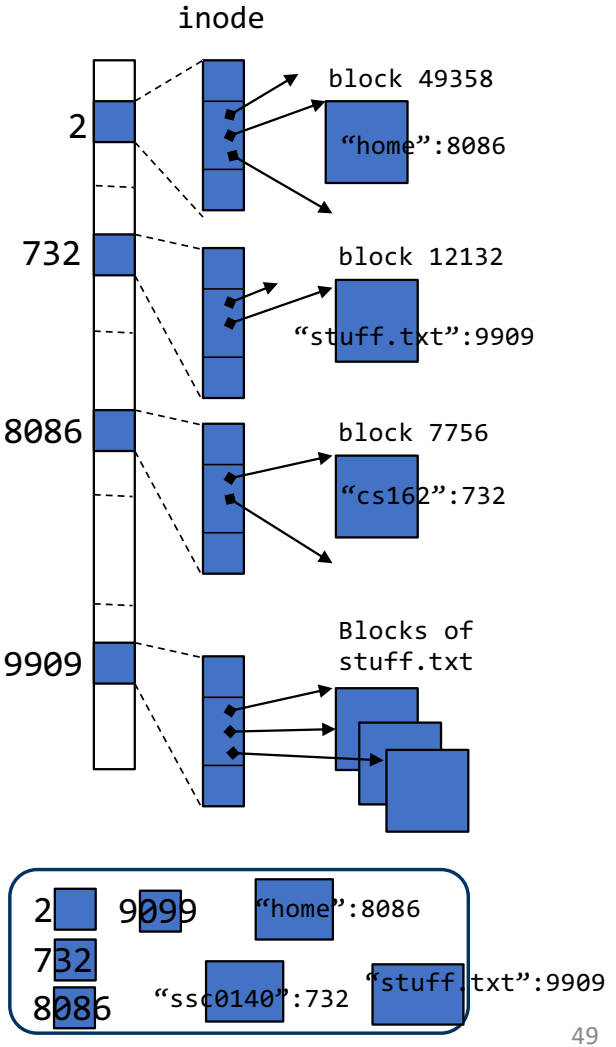
- Entrada normal do diretório : <file name, file #>
- Link Simbólico: <source file name, dest. file name>

- O sistema operacional procura o nome do arquivo de destino cada vez que o programa acessa o nome do arquivo de origem
  - A pesquisa pode falhar (resultado de erro de abertura)

- Unix: Cria **soft links** com **`symlink`** syscall

Directory Traversal

- O que acontece quando abrimos /home/ssc0140/stuff.txt?
- / - inumber para root inode está configurado no kernel, digamos 2
  - Lê o inode 2 a partir de sua posição na matriz de código no disco
  - Extraia os ponteiros de bloco direto e indireto
  - Determine o bloco que contém o diretório raiz (digamos, bloco 49358)
  - Leia esse bloco, procure ”home" para obter o número deste diretório (digamos 8086)
- Leia o inode 8086 para /home, extraia seus blocos, leia o bloco (digamos 7756), faça uma varredura em busca de “ssc0140” para obter seu inumber (digamos 732)
- Leia o inode 732 para /home/ssc0140, extraia seus blocos, leia o bloco (digamos 12132), faça a varredura para "stuff.txt" para obter seu inumber, digamos 9909
- Leia o inode 9909 para /home/ssc0140/stuff.txt
- Configure o descritor de arquivo para se referir a este inode para que a leitura / gravação possa acessar os blocos de dados referenciados por seus ponteiros diretos e indiretos
- Verifique as permissões no inode final e em cada inode do diretório ...

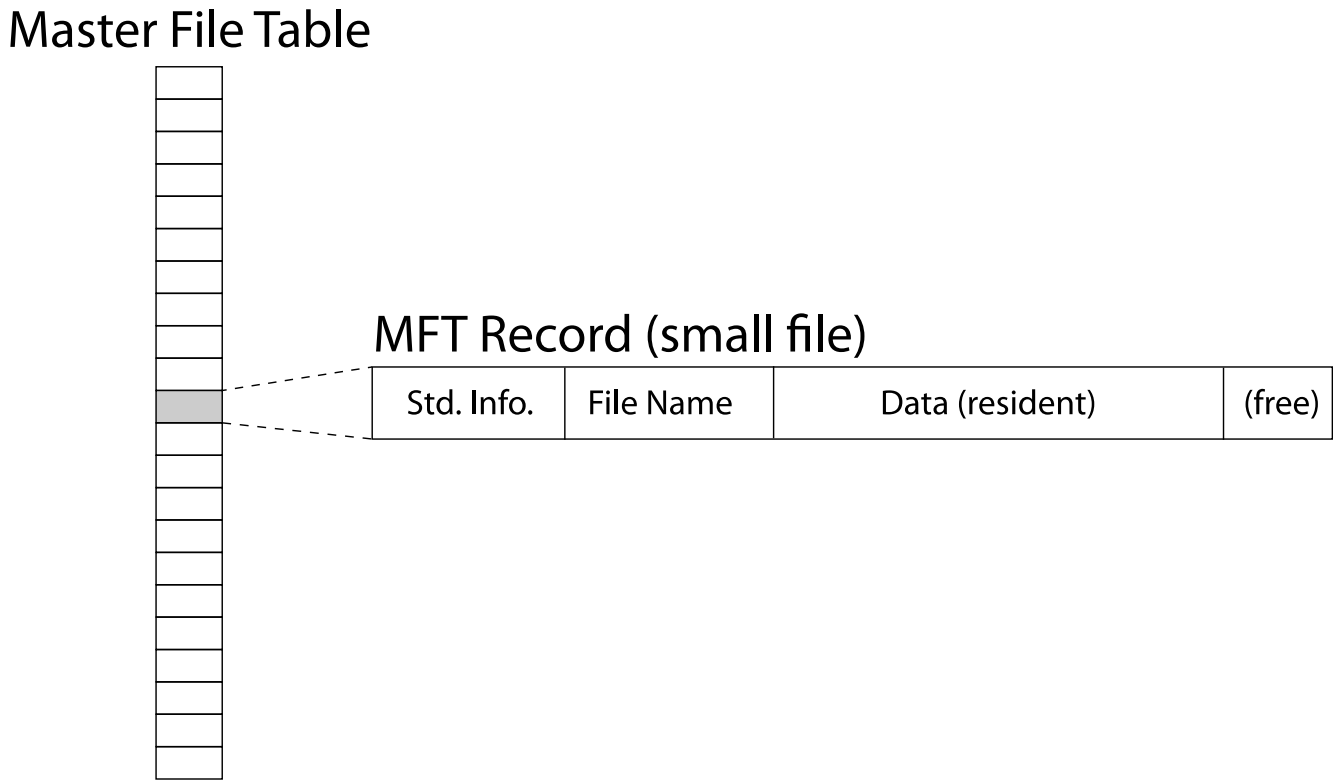


Windows NTFS

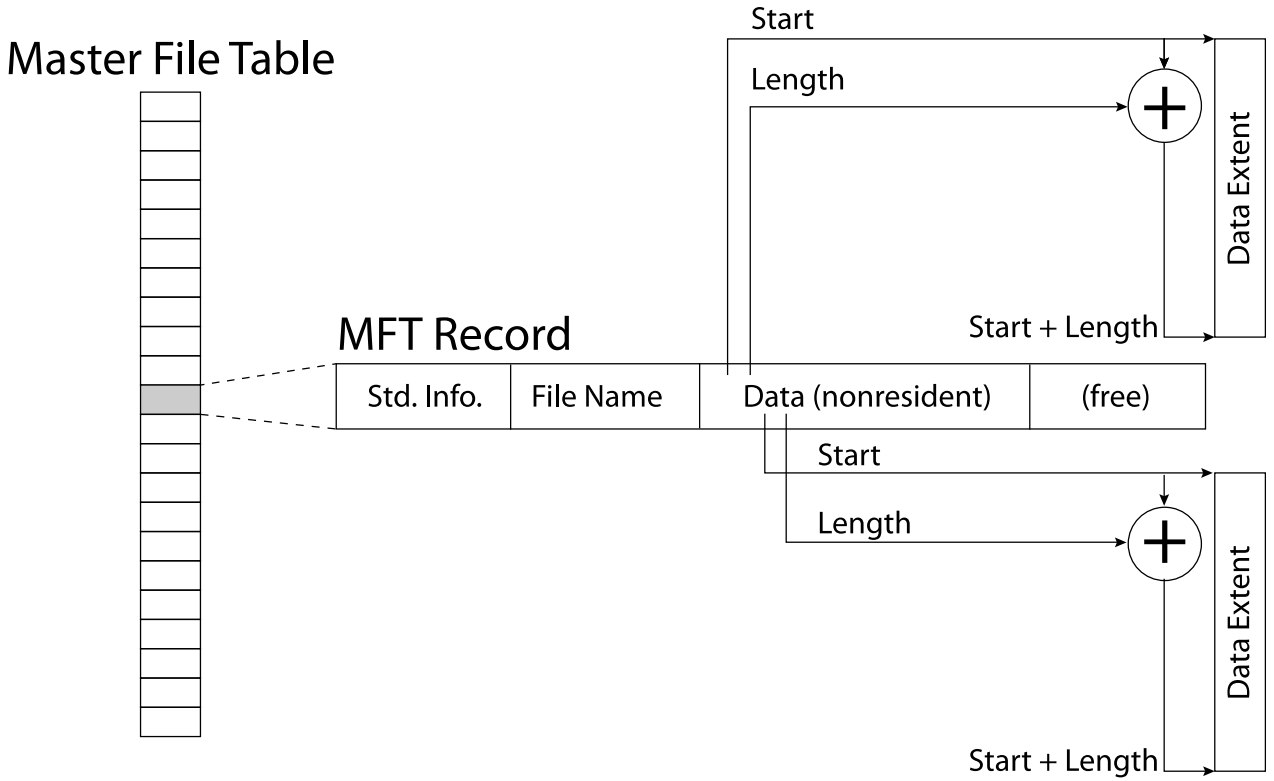
New Technology File System (NTFS)

- Padrão em sistemas Windows modernos
- Em vez de FAT ou matriz de inode: Tabela de arquivos mestre
  - Tamanho máximo de 1 KB para cada entrada da tabela
- Cada entrada na MFT contém metadados e:
  - Dados do arquivo diretamente (para arquivos pequenos)
  - Uma lista de extensões (bloco inicial, tamanho) para os dados do arquivo
  - Para arquivos grandes: ponteiros para outras entradas MFT com listas mais extensas

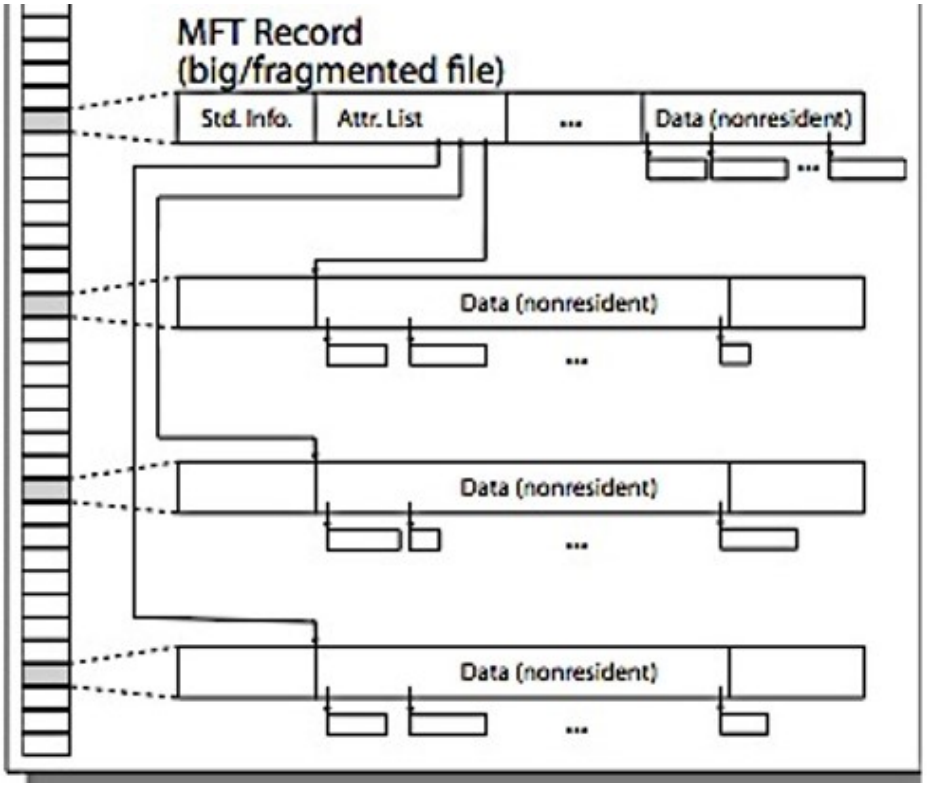
NTFS Arquivos pequenos: Dados no Registro MFT



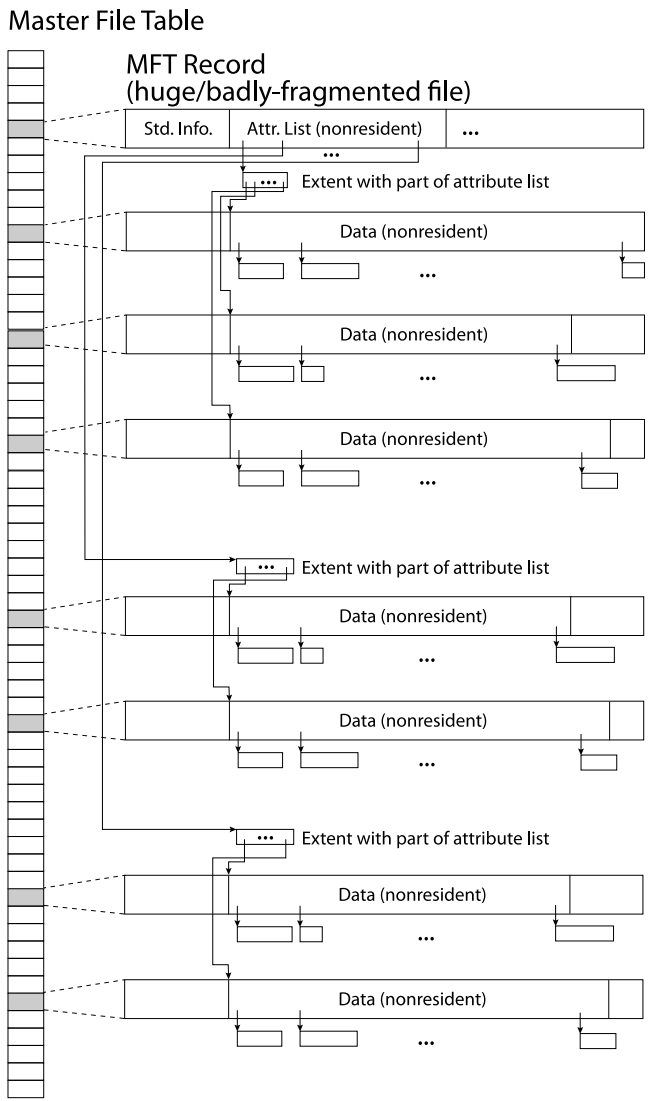
NTFS Arquivos Médios: Extensões para dados de arquivo



NTFS Arquivos Grandes: Indicadores para outros Registros MFT



NTFS Arquivo enorme e fragmentado: muitos registros MFT



NTFS Diretórios

- Diretórios implementados como Árvores B
- O número do arquivo identifica sua entrada no MFT
- A entrada MFT sempre tem um atributo de nome de arquivo
  - Nome legível por humanos, número do arquivo do diretório pai
- Link físico? Vários atributos de nome de arquivo na entrada MFT

Conclusões (1/2)

- Sistema de arquivo:
  - Transforma blocos em arquivos e diretórios
  - Otimize para tamanho, acesso e padrões de uso
  - Maximiza o acesso sequencial, permita um acesso aleatório eficiente
  - Projeta a proteção do sistema operacional e o regime de segurança (UGO vs ACL)
- Arquivo definido pelo cabeçalho, denominado “inode”
- Nomenclatura: tradução de nomes visíveis para o usuário em recursos reais do sistema
  - Diretórios usados para nomear sistemas de arquivos locais
  - Estrutura vinculada (*linked*) ou em árvore armazenada em arquivos
- Esquema Indexado Multinível
  - inode contém informações do arquivo, ponteiros diretos para blocos, blocos indiretos, duplamente indiretos, etc.
  - NTFS: extensões variáveis, não blocos fixos, dados de arquivos minúsculos no cabeçalho

Conclusões (2/2)

- 4.2 Arquivos de índice BSD Multilevel
  - Inode contém ptrs para blocos reais, blocos indiretos, blocos indiretos duplos, etc.
  - Otimizações para acesso sequencial: iniciar novos arquivos em intervalos abertos de blocos livres, otimização rotacional
- Layout de arquivo impulsionado pelo gerenciamento de espaço livre
  - Integra espaço livre, tabela de inode, blocos de arquivos e dirs no grupo de blocos