Sistemas Operacionais
Cap 6. Sistemas de Arquivos

Particionamento

Prof. Me. Rodrigo D. Malara

- Partições e montagem
- O básico (FAT)
- I-nodes e blocos (ext)
- Grupos de blocos (ext2)
- Journaling (ext3)
- Extensões e Árvores-B (ext4/NTFS)
- SAs baseados em Log (SSDs)

A ‘construção’ do sistema de arquivos raiz

- Uma das primeiras tarefas de um sistema operacional durante inicialização é ‘construir’ o sistema de arquivos raiz

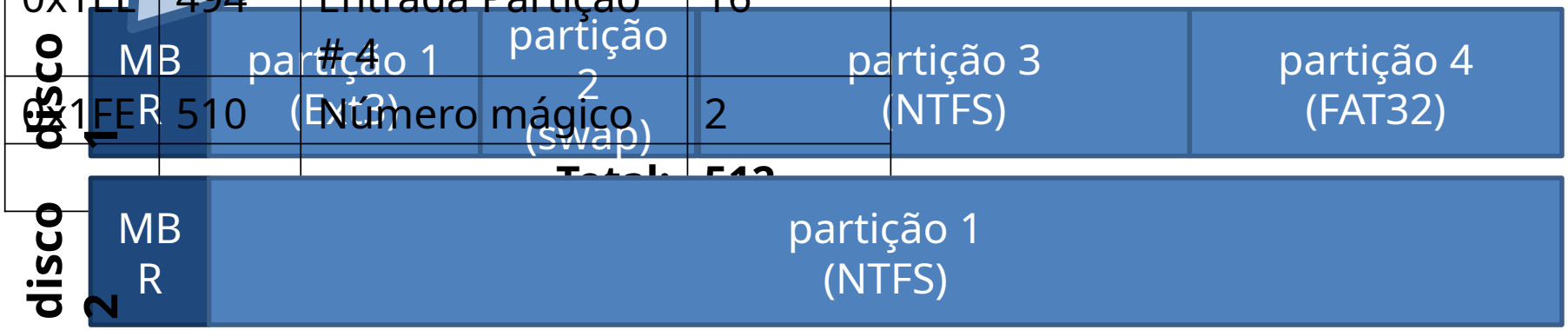
1. Localizar todas as medias *bootáveis* - BIOS
   - HDDs (Hard Disk Drives) internos e externos
   - SSD (Solid State Drives)
   - CDs, DVDs, pen drives, disquetes
2. Localize todas as partições em cada mídia - BIOS
   - Ler os MBR(s), tabelas de partição estendidas, etc.
3. montar uma ou mais partições – SO!
   - Torna o sistema (s) arquivo disponível para acesso

O Master Boot Record - MBR

Endereço		Descrição	Tamanho (Bytes)
Hex	**Dec**		
0x000	0	Código de Bootstrap	446
0x1BE	446	Entrada Partição # 1	16
0x1CE	462	Entrada Partição # 2	16
0x1DE	478	Entrada Partição # 3	16
0x1EE	494	Entrada Partição # 4	16
0x1FE	510	Número mágico	2
		Total:	**512**

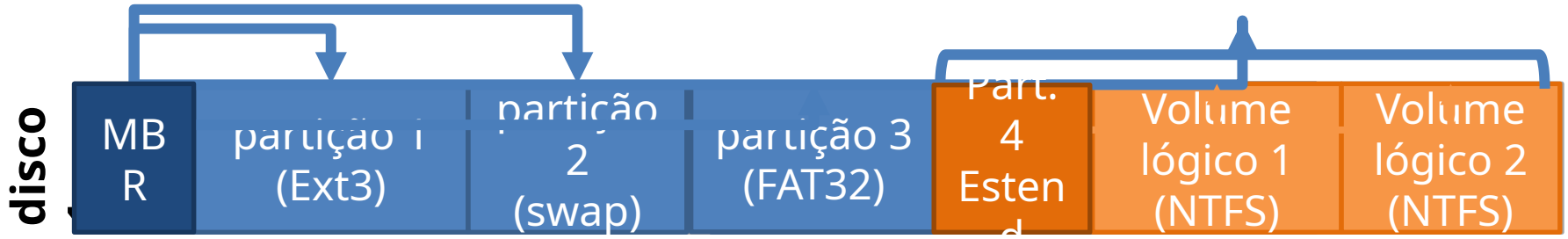
Inclui o LBA de inicialização e o tamanho da partição

LBA: Logical Block Address



Partições Estendidas

- Em alguns casos é necessário ter + que 4 partições
- Modernos SOs tem suporte a partições estendidas



- Partições estendidas podem utilizar formatos de tabela de partição específica do SO (meta-dados)
  - Assim, outros SOs podem não ser capazes de ler as partições lógicas

Tipos de Sistemas de Arquivos Raiz

```
[ CBW @ ativ9 ~] df -h
Disp.    Tamanho Usado Avail Use%    Montado em
/dev/sda7 39G    14G   23G   38%     /
/dev/sda2 296M   48M   249m  16%     /boot/efi
/dev/sda5 127g   86g   42G   68%     /media/CBW/Dados
/dev/sda4 61G    34G   27G   57%     /media/CBW/Windows
/dev/sdb1 1.9T   352k  1.9T   1%     /media/CBW/NDSS2013
```

1 unidade, 4 partições

1 unidade 1 partição

- Windows expõe um sistema multi-raiz
  - Cada dispositivo e partição é atribuída uma letr
  - Internamente, uma única raiz é mantida
- Unix tem uma única raiz
  - Uma partição é montada como /
  - Todas as outras partições são montados em algum lugar sob /
- Tipicamente, a partição que contém o núcleo é montado como / ou C:

Montar um sistema de arquivos

1. Ler o super-bloco para o sistema de arquivo de destino
   - Contém meta-dados sobre o sistema de arquivos
   - Versão, tamanho, localização de estruturas-chave no disco, etc.
2. Determinar o ponto de montagem
   - No Windows: escolher uma letra de unidade
   - No Linux: montar o novo sistema de arquivos em um diretório específico

```
Sistema de arquivo Tamanho Usado Avail Use% Montado em
/dev/sda5 127G 86G  42G  68% /media/CBW/Dados
/dev/sda4  61G 34G  27G  57% /media/CBW/Windows
/dev/sdb1 1.9T 352k 1.9T  1% /media/CBW/NDSS2013
```

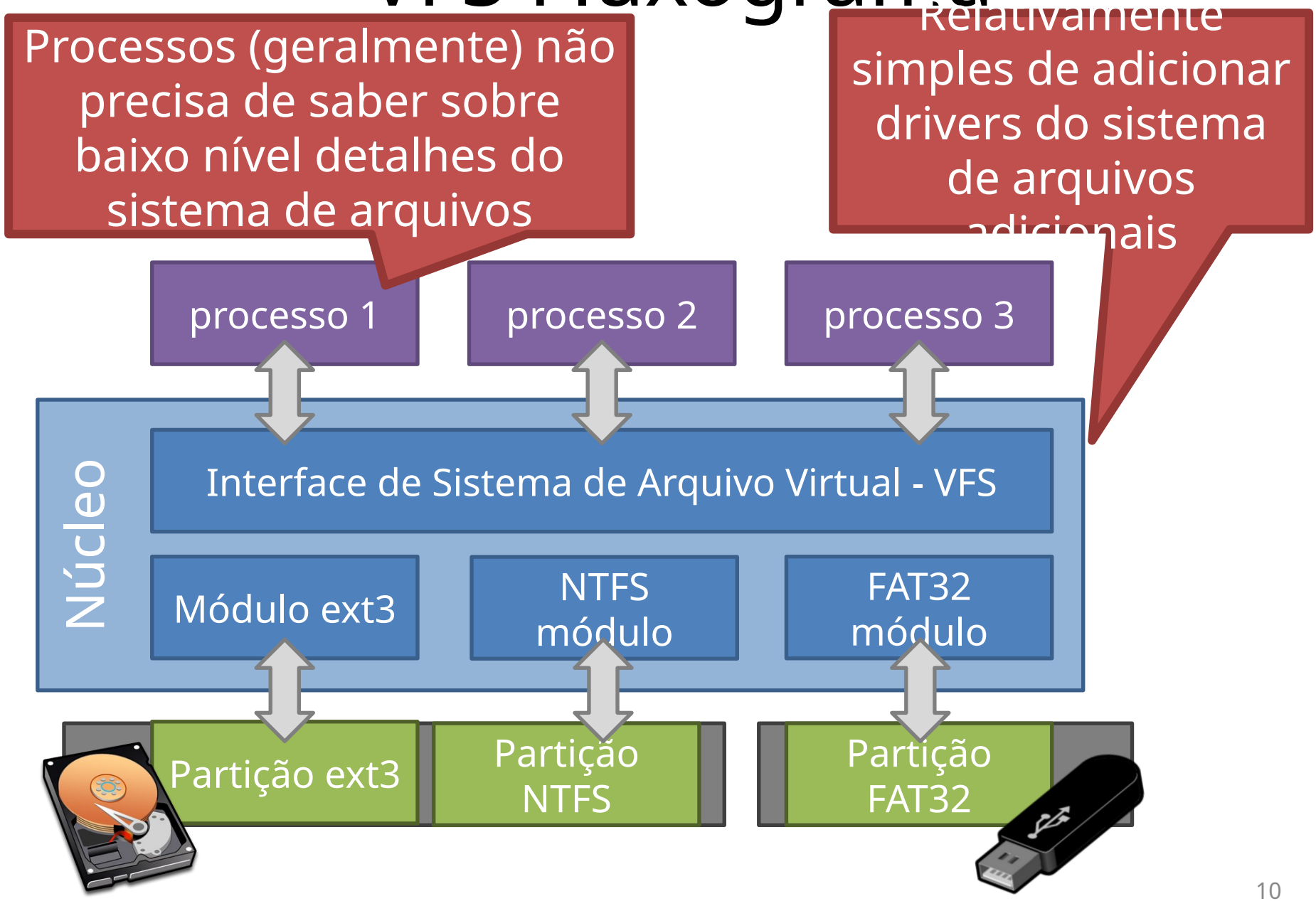
Interface de Sistema de Arquivos Virtual

- Problema: O sistema operacional pode montar várias partições contendo diferentes sistemas de arquivos subjacentes
  - Seria ruim se os processos tiveram que usar diferentes APIs para os diferentes sistemas de arquivos
- Linux usa uma interface Virtual File System (VFS)
  - Expõe APIs POSIX para processos
  - Encaminha solicitações para drivers específicos do sistema de arquivos de nível inferior

Interface de Sistema de Arquivos Virtual

- Sistema de Arquivos Virtual
- Camada que esconde diferenças de tecnologias de formatação e físicas dos dispositivos de armazenamento
- Disponível nos Sistemas Operacionais UNIX (Linux, FreeBSD, ...)
- É uma API única para acesso a diferentes tipos de sistema de arquivos
- O Windows usa um sistema semelhante

VFS Fluxograma



Mount não é apenas para boot

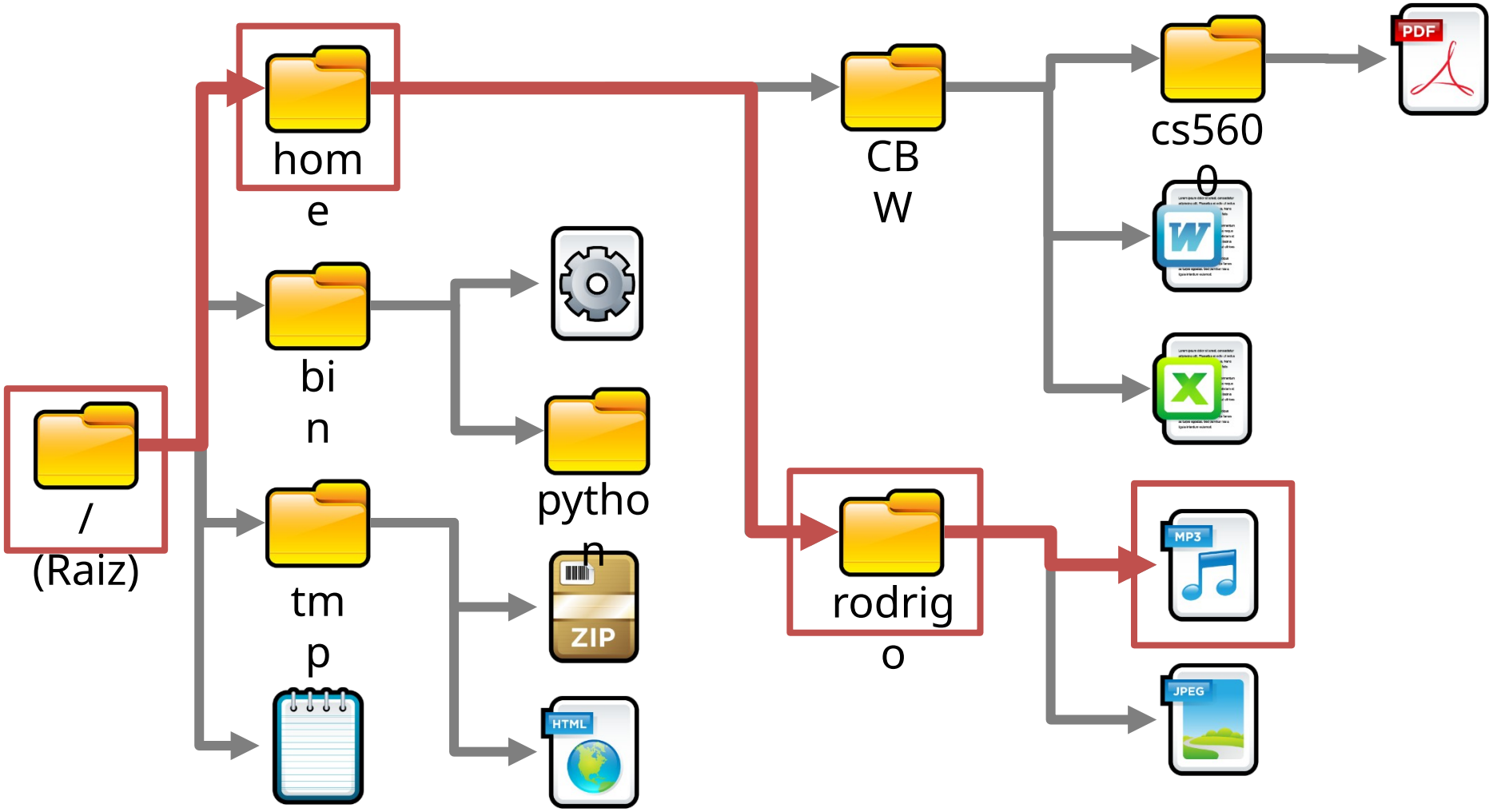
- Ao conectar dispositivos de armazenamento em seu sistema em execução, montagem é executado em segundo plano
- Exemplo: ligar um pendrive USB
- O que significa a “ejetar com segurança” um dispositivo?
  - Esvaziar o buffer de escrita do dispositivo
  - Desmontar o sistema de arquivos no dispositivo

- Partições e montagem
- O básico (FAT)
- inodes e blocos (ext)
- Grupos de blocos (ext2)
- Journaling (ext3)
- Extensões e Árvores-B (ext4)
- SAs baseados em Log

Como estamos até então...

- Neste ponto, o sistema operacional pode localizar e montar partições
- O passo seguinte: o que é o layout em disco do sistema de arquivos?
  - Esperamos que certas características de um sistema de arquivos
    - arquivos nomeados
    - hierarquia aninhada de diretórios
    - Meta-dados como o tempo de criação, permissões de arquivos, etc.
  - Como projetar estruturas de disco que suportam esses recursos?

A árvore de diretórios



- Navegada utilizando um caminho
  - Ex: /home/rodrigo/music.mp3

Caminhos absolutos e relativos

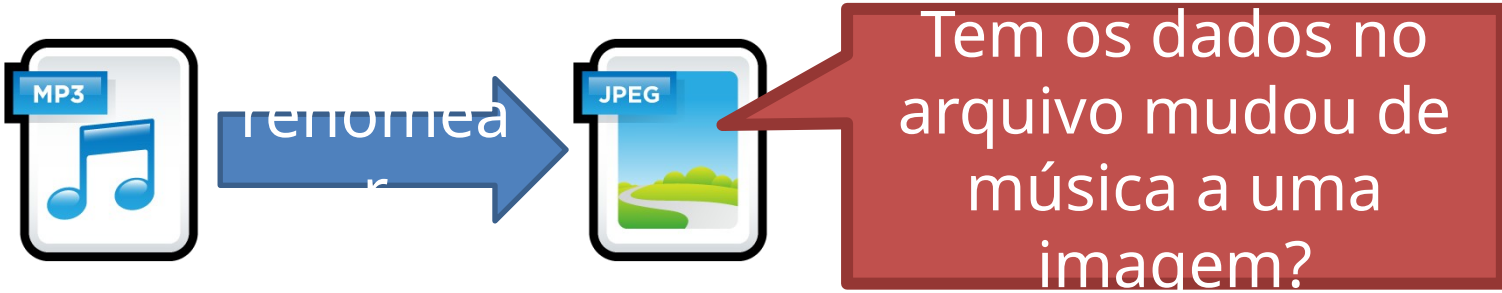
- Dois tipos de caminhos de sistema de arquivo
  - Absoluto
    - caminho completo da raiz para o objeto
    - Exemplo: /home/rodrigo/arquivo/hw4.pdf
    - Exemplo: C:\Users\cbw\Documentos\teste.txt
  - Relativo
    - OS mantém o controle do diretório de trabalho para cada processo de
    - Caminho relativo para o diretório de trabalho atual
    - Exemplos [diretório de trabalho = /home/rodrigo]:
      - syllabus.docx [→ /home/rodrigo/syllabus.docx]
      - ~/cs5600/hw4.pdf [→ /home/rodrigo/cs5600/hw4.pdf]
      - ./cs5600/hw4.pdf [→ /home/rodrigo/cs5600/hw4.pdf]
      - ../rodrigo/music.mp3 [→ /home/rodrigo/music.mp3]

arquivos

- Um arquivo é um composto de dois componentes
  - Os dados do arquivo em si
    - Um ou mais blocos (setores) de dados binários
    - Um arquivo pode conter qualquer coisa
  - Meta-dados sobre o arquivo
    - Nome, tamanho total
    - O diretório está?
    - Criado tempo, tempo modificado, tempo de acesso
    - Oculto ou sistema de arquivos?
    - grupo do proprietário e proprietário
    - Permissões: leitura / gravação / execução

Extensões de arquivo

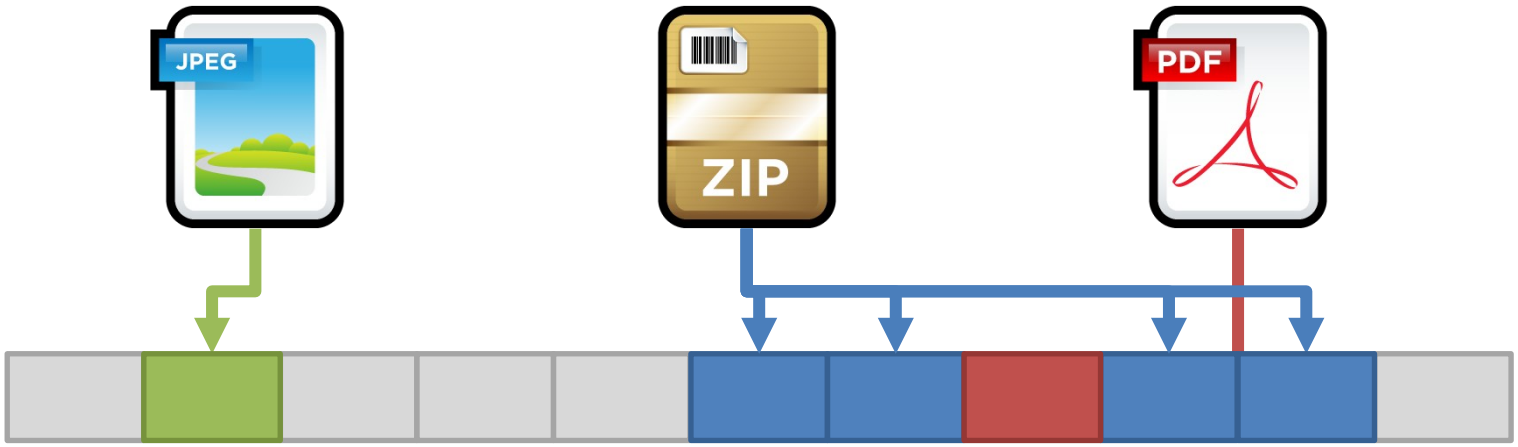
- O nome do arquivo são muitas vezes escritos separados por pontos
  - Por exemplo programa.exe, image.jpg, music.mp3
- A extensão de um arquivo **não significa nada**
  - Qualquer arquivo (independentemente do seu conteúdo) pode ser dada qualquer nome ou extensão



- Shells gráficos (como o Windows Explorer) usam extensões para tentar associar arquivos a programas
  - Este mapeamento pode falhar por uma variedade de razões

Mais arquivo Meta-Data

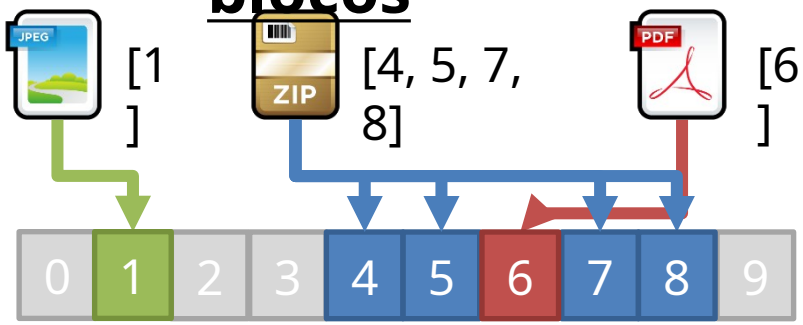
- Os arquivos têm meta-dados adicionais que normalmente não são mostrados para os usuários
  - identificador único (nomes de arquivos pode não ser único)
  - Estrutura que mapeia o arquivo para blocos no disco
- Gerenciando o mapeamento de arquivos para blocos é um dos trabalhos principais do sistema de arquivos



Arquivos mapeamento para Blocos

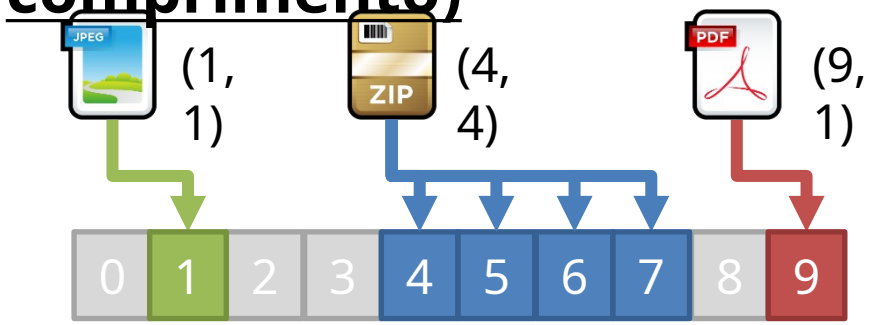
- Cada arquivo é composto por >= 1 blocos
- pergunta-chave: como mapear um arquivo de seus blocos?

Lista de blocos



- Problema?
  - arquivos realmente grandes

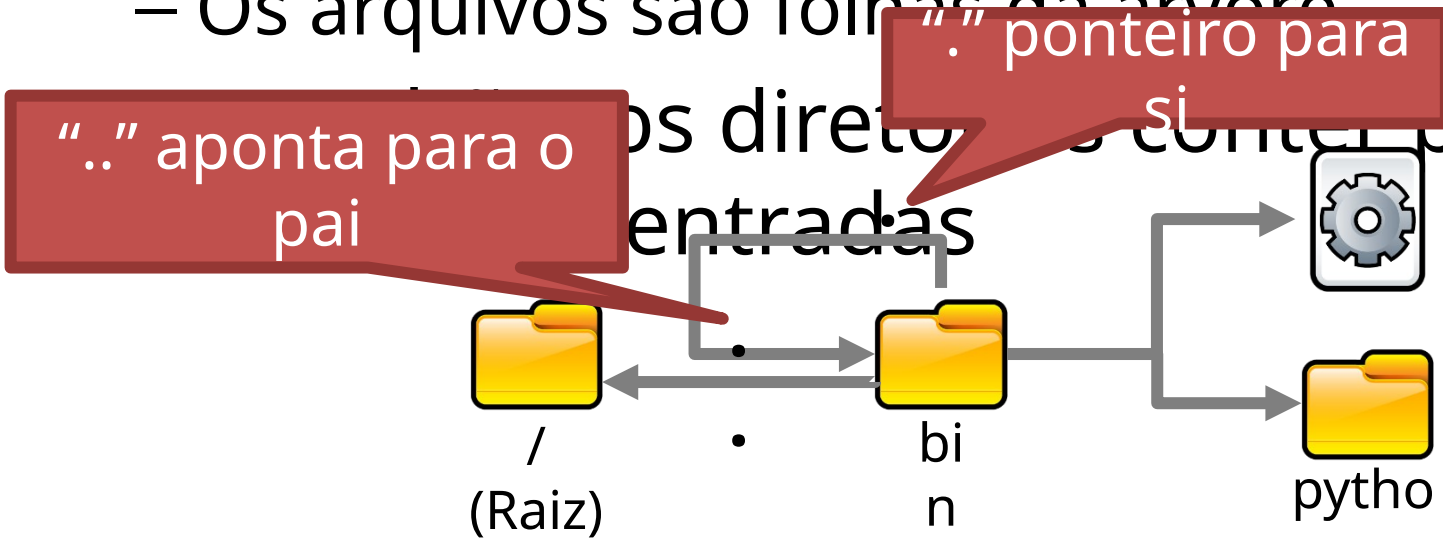
Como pares (início, comprimento)



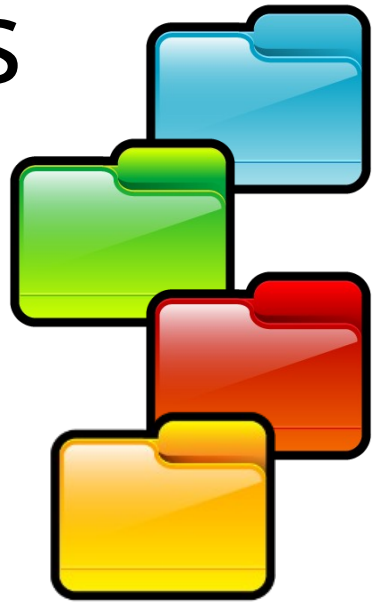
- Problema?
  - fragmentação
  - Por exemplo, tentar adicionar um novo arquivo com 3 blocos

Diretórios

- Tradicionalmente, os SAs têm utilizado um espaço de nomes hierárquico, estruturado em árvore
  - diretórios são objetos que contêm outros objetos
    - ou seja, um diretório pode (ou não) ter filhos
  - Os arquivos são folhas da árvore
  - os diretórios contêm pelo entradas

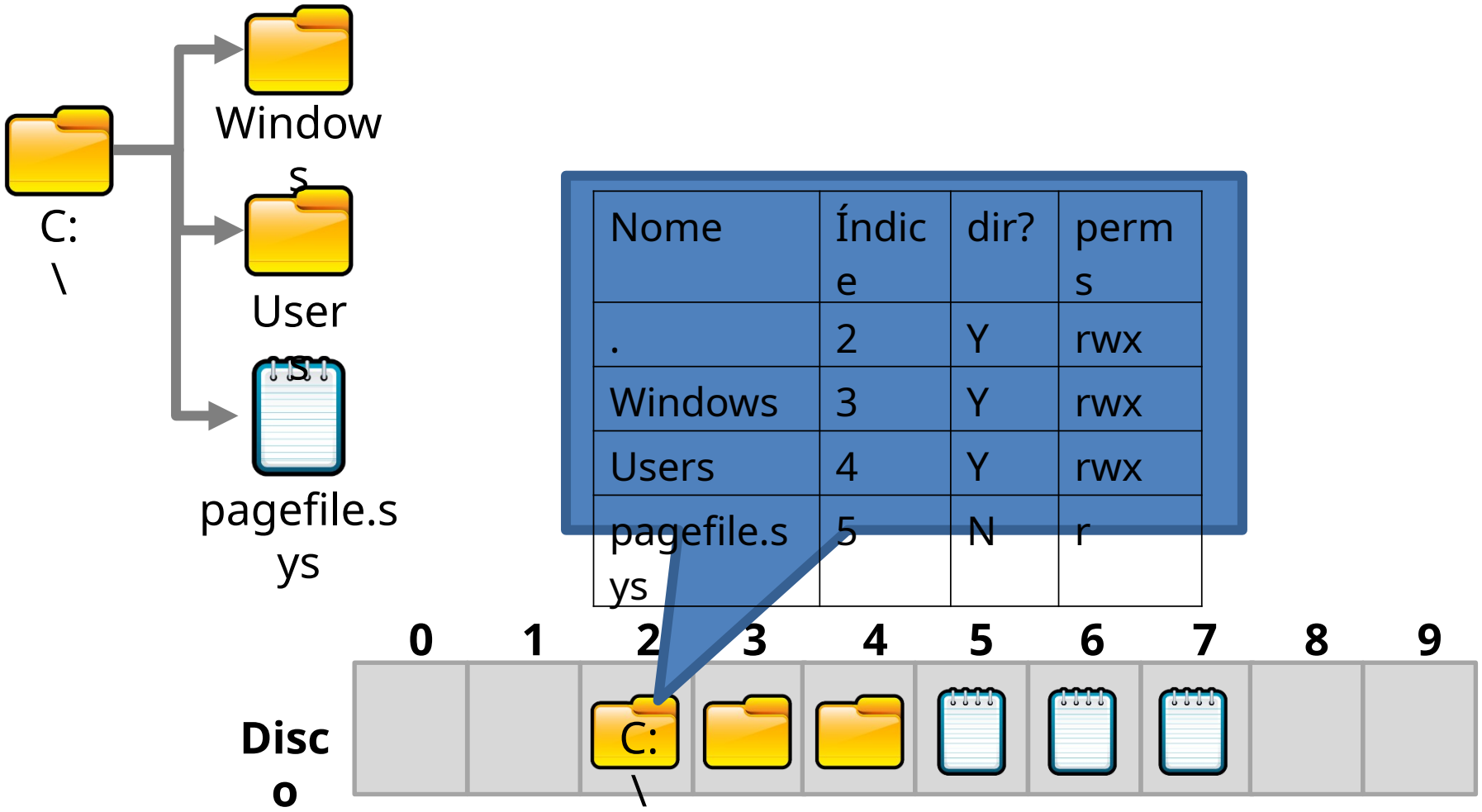


Mais sobre Diretórios



- Diretórios têm associado meta-dados
  - Nome, número de entradas
  - Data/Hora de criação, modificação e acesso
  - Permissões (leitura / gravação), proprietário e grupo
- O sistema de arquivos deve codificar diretórios e armazená-los no disco
  - Normalmente, os diretórios são armazenados como um tipo especial de arquivo
  - Arquivo contém uma lista de entradas dentro do diretório, além de alguns meta-dados para cada entrada

Arquivo Exemplo Diretório



Implementação de diretórios

- Cada diretório de arquivos armazena muitas entradas
- Pergunta-chave: como você codifica as entradas?

Lista de entradas desordenada

Nome	Índice	dir?	perms
.	2	Y	rwx
Windows	3	Y	rwx
Users	4	Y	rwx
pagefile.sys	5	N	r

- Bom: O (1) para adicionar novas entradas
  - Basta anexar ao arquivo
- Ruim: O (n) para procurar uma entrada

Lista de entradas classificada

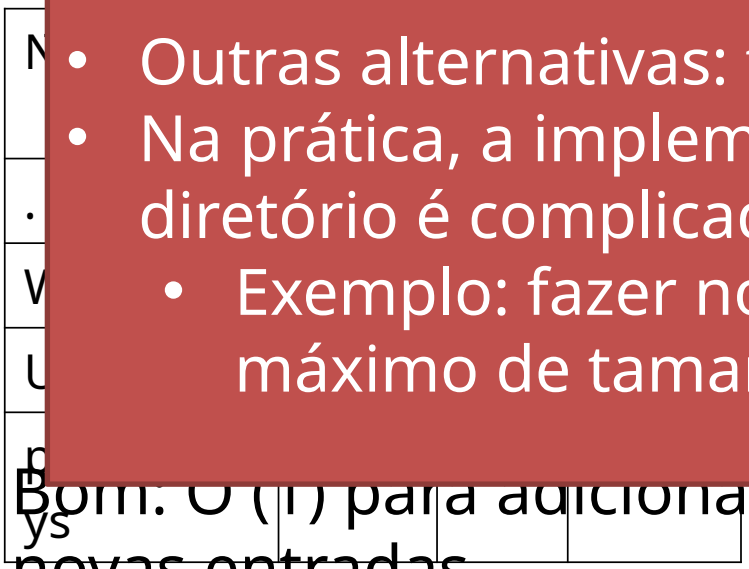
Nome	Índice	dir?	perms
.	2	Y	rwx
pagefile.sys	5	N	r
Users	4	Y	rwx
Windows	3	Y	rwx

- Bom: O(log n) para procurar uma entrada
- Ruim: O(n) para adicionar novas entradas
  - arquivo inteiro tem que ser reescrito

Implementação de diretórios

- Cada diretório de arquivos armazena muitas entradas
- Pergunta-chave: como você codifica as entradas?

Lista de entradas



- Bom: O (1) para adicionar novas entradas
  - Basta anexar ao arquivo
- Ruim: O (n) para procurar uma entrada

Lista de entradas

- Bom: O(log n) para procurar uma entrada
- Ruim: O(n) para adicionar novas entradas
  - arquivo inteiro tem que ser reescrito

- Outras alternativas: tabelas hash, árvores-B
- Na prática, a implementação de arquivos do diretório é complicada
  - Exemplo: fazer nomes têm, um comprimento máximo de tamanho fixo ou variável?

Tabelas de alocação de arquivos (FAT)

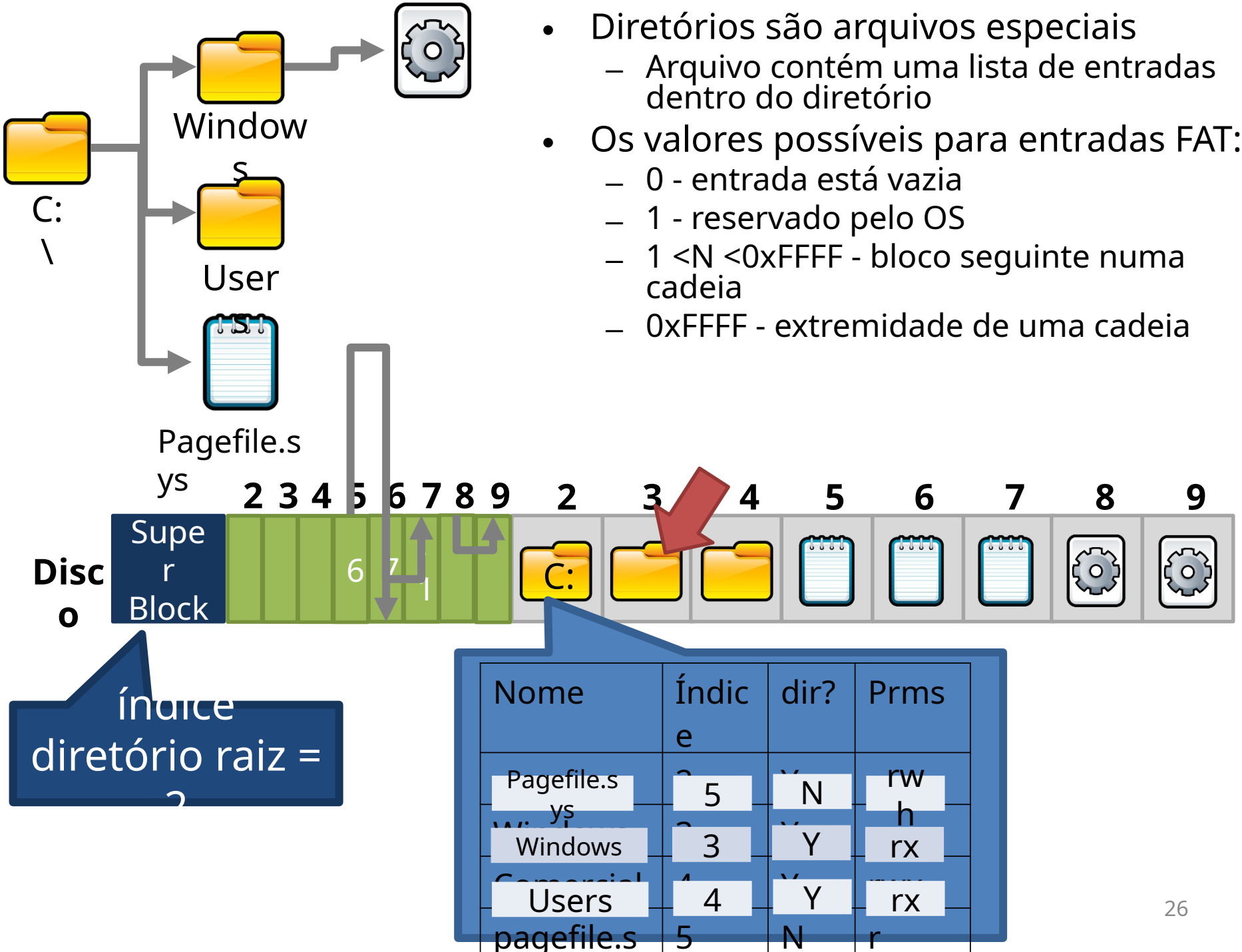
- Sistema de arquivos simples popularizado por MS-DOS
  - Introduzido pela primeira vez em 1977
  - A maioria dos dispositivos hoje usam a especificação FAT32, de 1996
    - Na verdade é FAT28 pois usa apenas 28 bits
  - FAT12, FAT16, VFAT, FAT32, etc.
- Hoje ainda é bastante popular
  - Era formato padrão para pen drives e cartões de memória
  - Usado para partições de inicialização EFI
- Nome vem do tabela de índice usado para rastrear diretórios e arquivos

- Armazena informações básicas sobre o sistema de arquivos
- Versão FAT, localização de arquivos de inicialização
- número total de blocos
- Índice do diretório raiz (root) FAT

- File Allocation Table (FAT)
- Marca quais blocos estão livres ou em uso
- **estrutura de lista ligada** para gerenciar arquivos grandes (> que 1 bloco)

- Lista de arquivos contidos dentro do diretório
- Cada bloco é um tamanho fixo (4KB - 64 KB)
- Os arquivos podem abranger vários

- Diretórios são arquivos especiais
  - Arquivo contém uma lista de entradas dentro do diretório
- Os valores possíveis para entradas FAT:
  - 0 - entrada está vazia
  - 1 - reservado pelo OS
  - 1 <N <0xFFFF - bloco seguinte numa cadeia
  - 0xFFFF - extremidade de uma cadeia

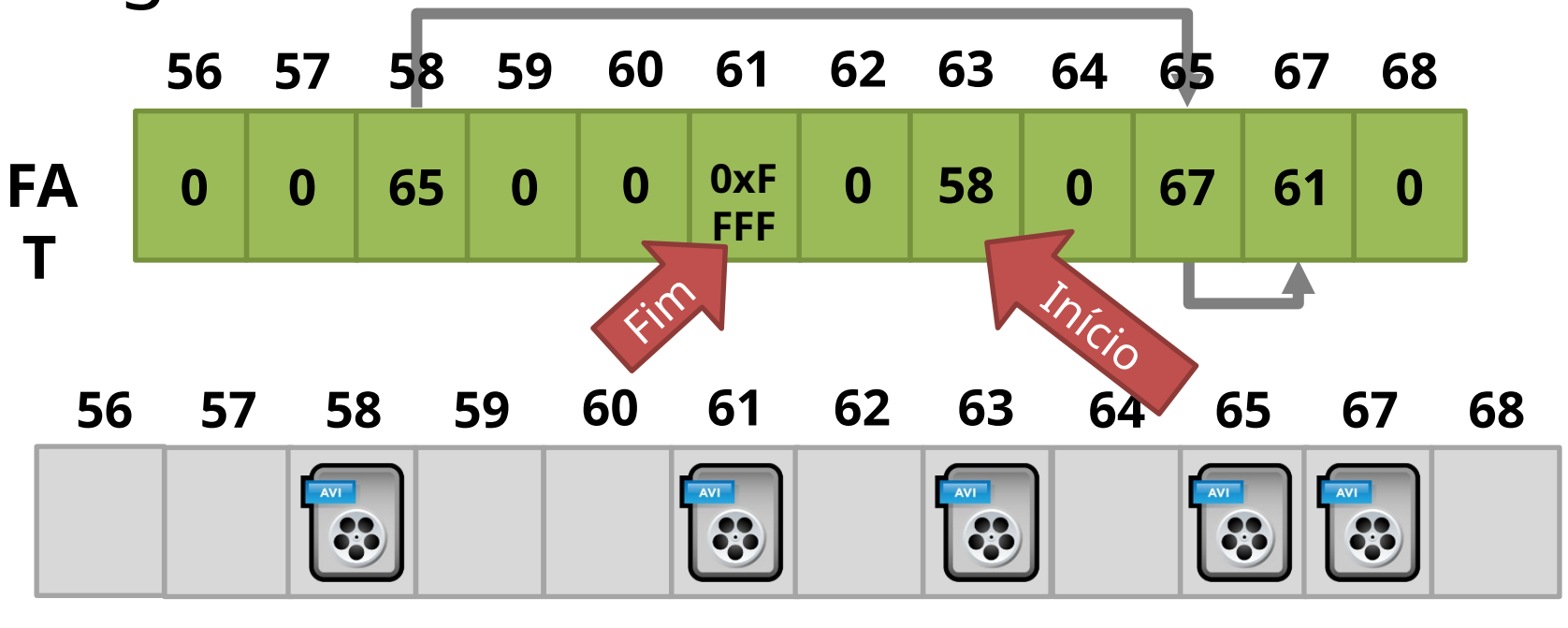


Entradas da tabela FAT

- len(FAT) == número de blocos no disco
  - Número máximo de arquivos / diretórios é delimitada
  - Decidiu quando você formatar a partição
- A versão FAT corresponde aproximadamente ao tamanho em bits de cada entrada do FAT
  - por exemplo, FAT16 → cada entrada do FAT é de 16 bits
  - Mais bits → discos maiores são suportadas

Fragmentação

- Blocos para um arquivo não precisa ser contíguo



Os valores possíveis para entradas FAT:

- 0 - entrada está vazia
- 1 <N <0xFFFF - bloco seguinte numa cadeia
- 0xFFFF - final de um cadeia

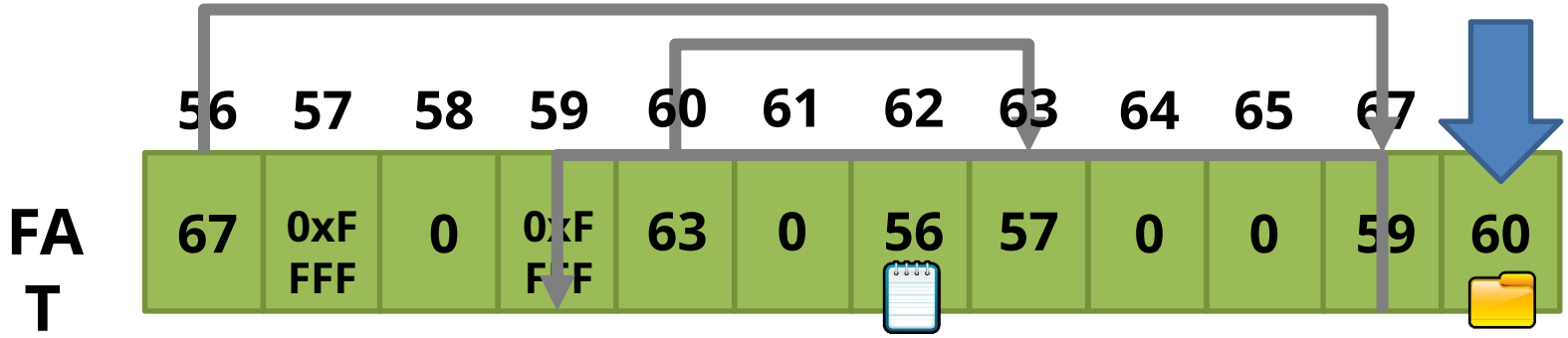
FAT: Vantagens e Desvantagens

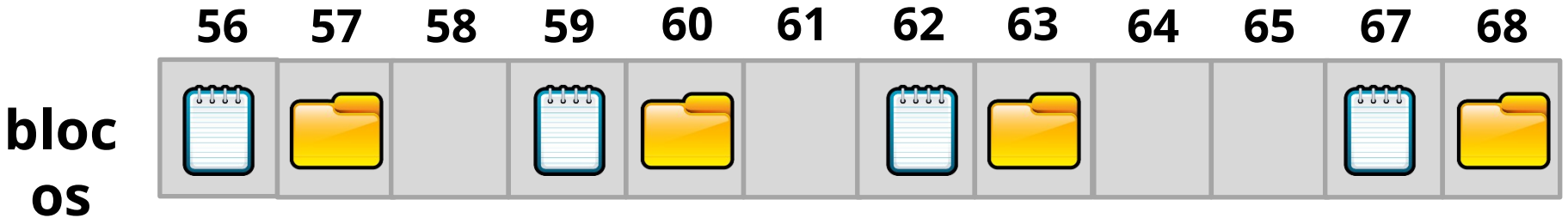
- Vantagens - FAT suporta:
  - árvore hierárquica de diretórios e arquivos
  - arquivos de tamanho variável
  - Armazena arquivos básicos e diretório de meta-dados
- Desvantagens
  - No máximo, FAT32 suporta discos de 1TB (blocos de 4k)
  - Localizando espaços livres requer a digitalização de todo o FAT
  - Propenso a fragmentação interna e externa
    - grandes blocos → fragmentação interna
  - **Leitura exige muita busca aleatória no disco**

Muitas Buscas

- Considere o seguinte código:

```
FILE *fd = fopen("C:/temp/a/Meu_arquivo.txt", "rt");
int r = fread(fd, buffer, sizeof(char), 1024 * 4 * 4); //
4 blocos de 4KB
```



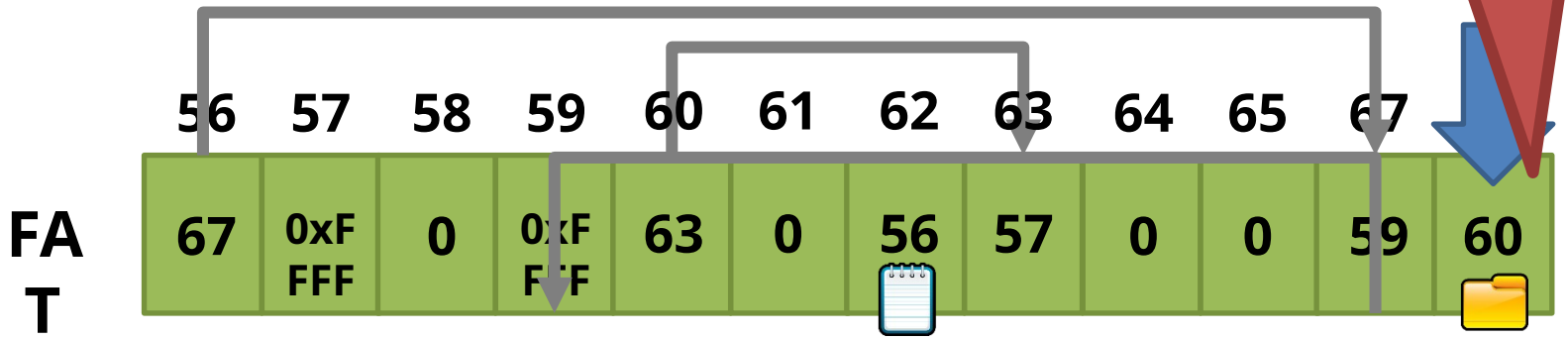


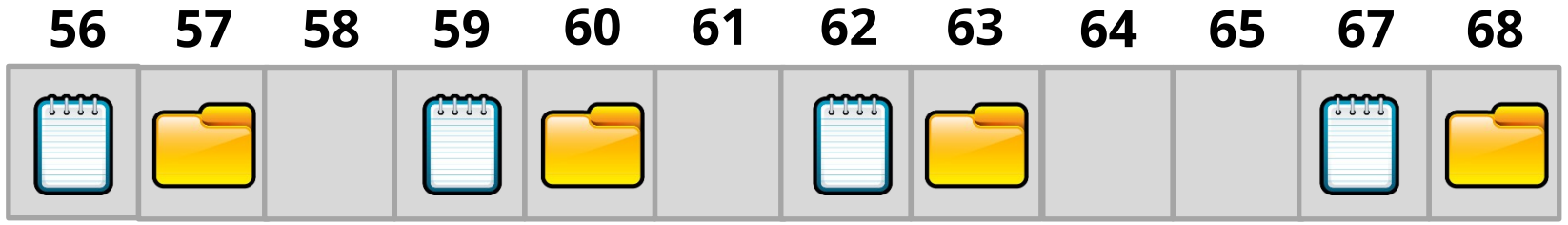
Muitas Busc

FAT pode ter muito baixa localidade espacial, assim, um monte de busca aleatória

- Considere o seguinte código

```
int fd = fopen("Meu_arquivo.txt",
int r = fread(fd, buffer, 1024 * 4 * 4); // 4 bloco
4KB
```





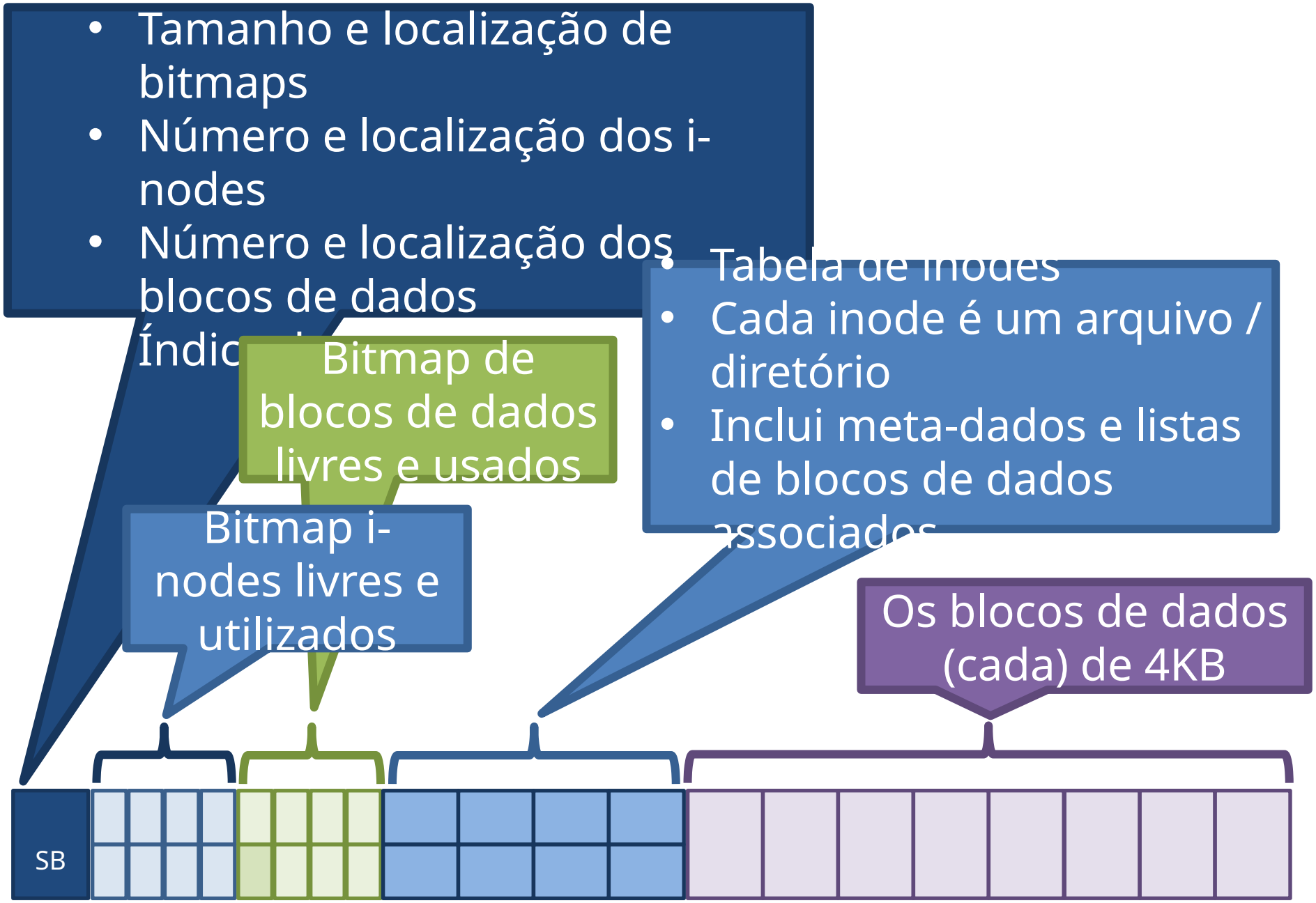
- Partições e montagem
- O básico (FAT)
- I-nodes e blocos (ext)
- Grupos de blocos (ext2)
- Journaling (ext3)
- Extensões e Árvores-B (ext4)
- SAs baseados em Log

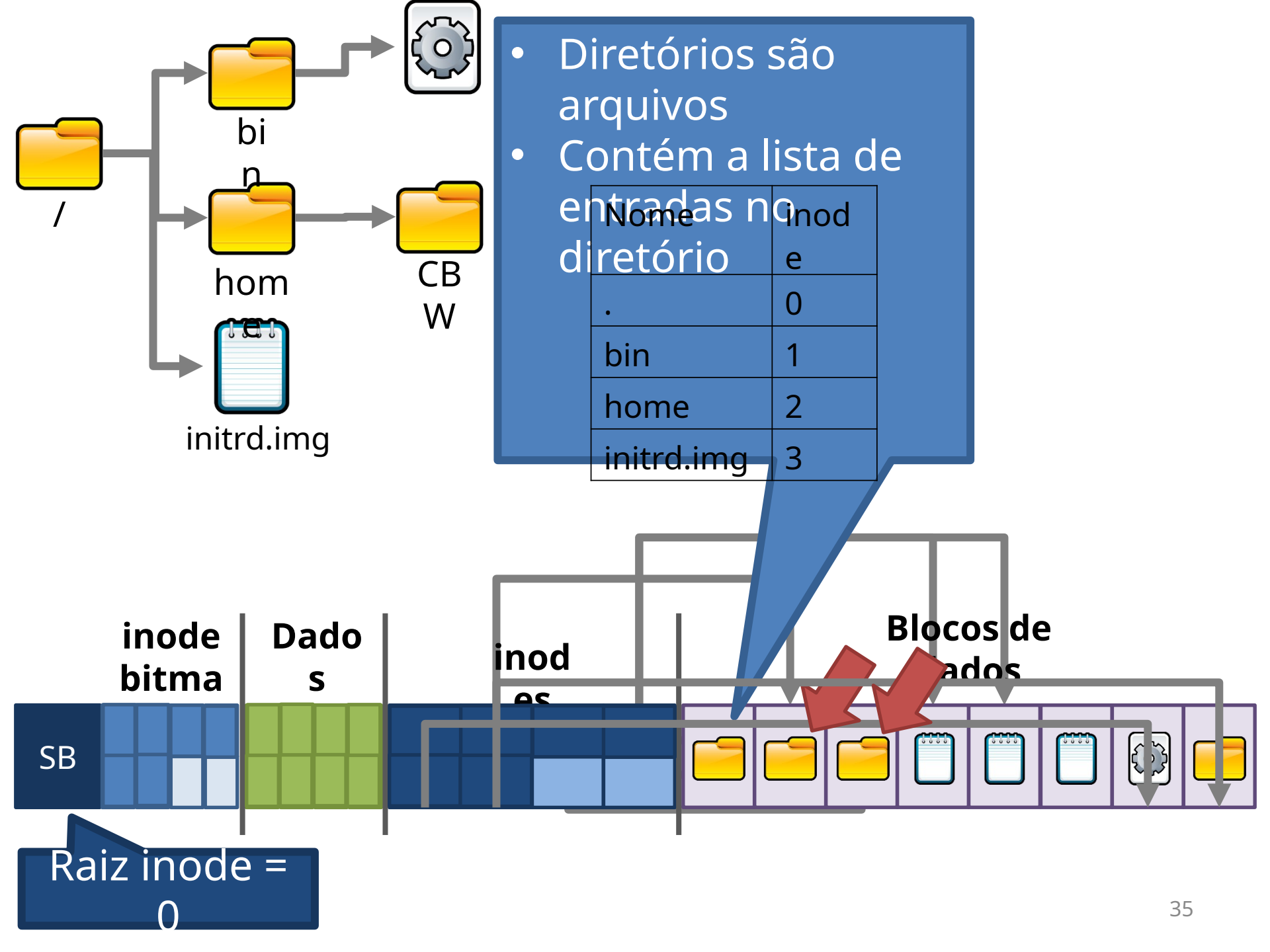
Como estamos até então...

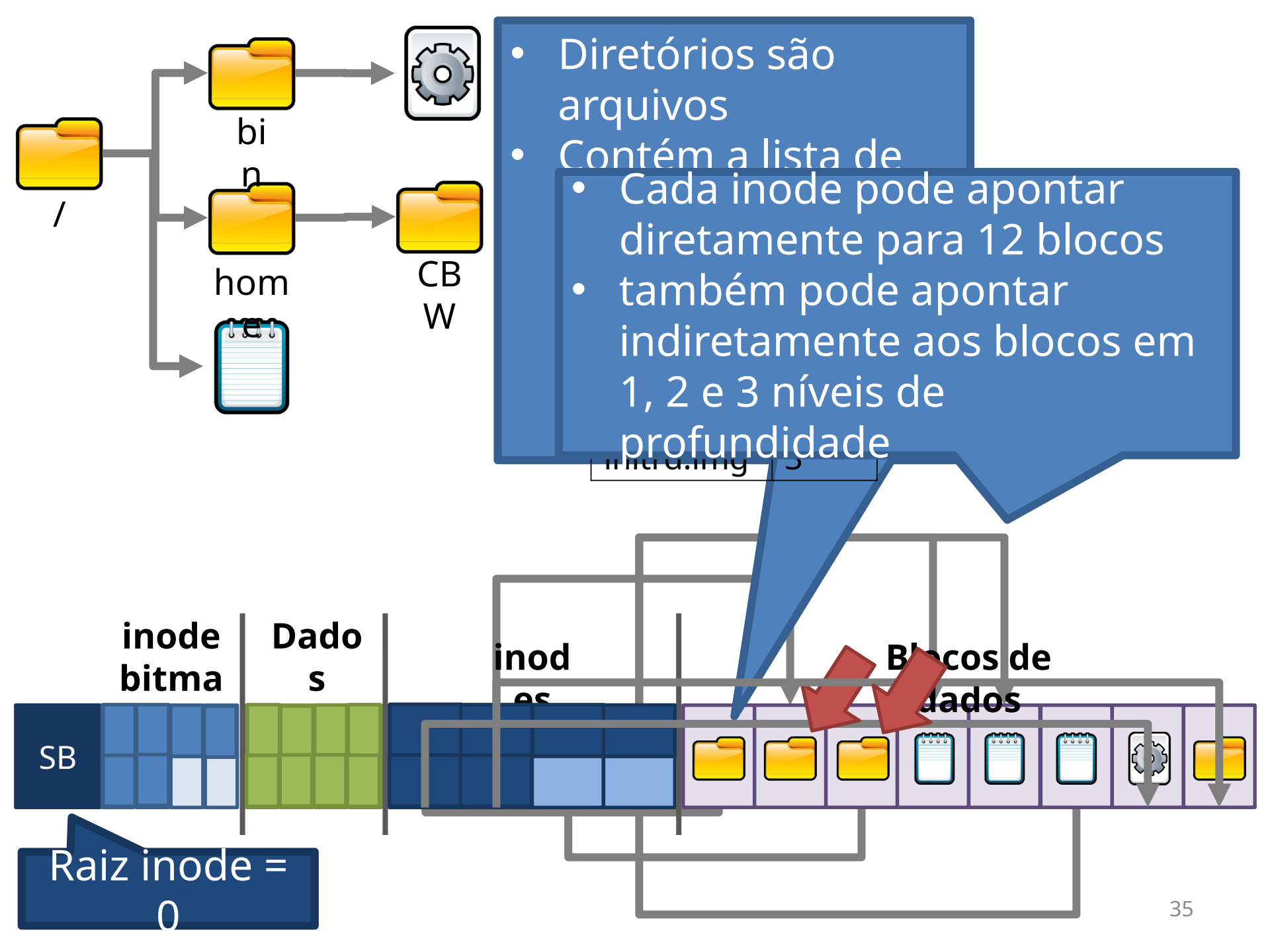
- Neste ponto, temos estruturas de disco para:
  - Construir uma árvore de diretórios
  - Armazenar arquivos de comprimento variável
- Mas, a eficiência do FAT é muito baixa
  - Lotes de busca sobre cadeias de arquivo no FAT
  - Única maneira de identificar o espaço livre é fazer a varredura em todo o FAT
- Sistema de arquivos Linux utiliza estruturas mais eficientes
  - Extend Filesystem (ext) usa nós índice (I-nodes) ou ainda Blocos de Controle de Arquivos (BCAs) para rastrear arquivos e diretórios

Tamanho Distribuição de Arquivos

- FAT usa uma lista ligada para todos os arquivos
  - mecanismo simples e uniforme
  - ... mas, não é otimizado para arquivos curtas ou longas
- Pergunta: são arquivos curtas ou longas mais comum?
  - Estudos ao longo dos últimos 30 anos mostram que os arquivos curtos são muito mais comuns
  - 2 KB é o tamanho do arquivo mais comum
  - O tamanho médio de arquivo é 200 KB (tendência para cima por alguns arquivos muito grandes)
- Ideia-chave: otimizar o sistema de arquivo para muitos arquivos pequenos


• Tamanho e localização de bitmaps
• Número e localização dos i-nodes
• Número e localização dos blocos de dados
Índic
Bitmap de blocos de dados livres e usados
Bitmap i-nodes livres e utilizados
• Tabela de inodes
• Cada inode é um arquivo / diretório
• Inclui meta-dados e listas de blocos de dados associados
Os blocos de dados (cada) de 4KB
SB

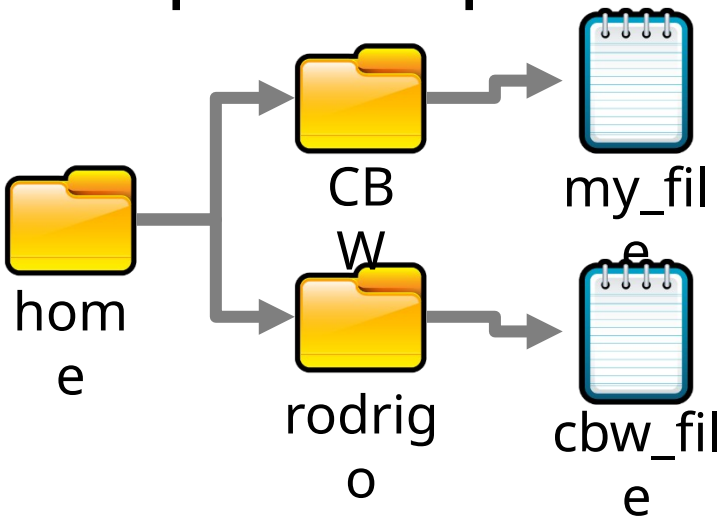

/
bin
home
CBW
initrd.img
• Diretórios são arquivos
• Contém a lista de entradas no diretório
Nome
inode
.
0
bin
1
home
2
initrd.img
3
inode bitmap
Dados
inodes
Blocos de dados
SB
Raiz inode = 0


/
bi
n
hom
e
CB
W
• Diretórios são arquivos
• Contém a lista de
• Cada inode pode apontar diretamente para 12 blocos
• também pode apontar indiretamente aos blocos em 1, 2 e 3 níveis de profundidade
inode bitma
Dado s
inod es
Blocos de dados
SB
Raiz inode = 0

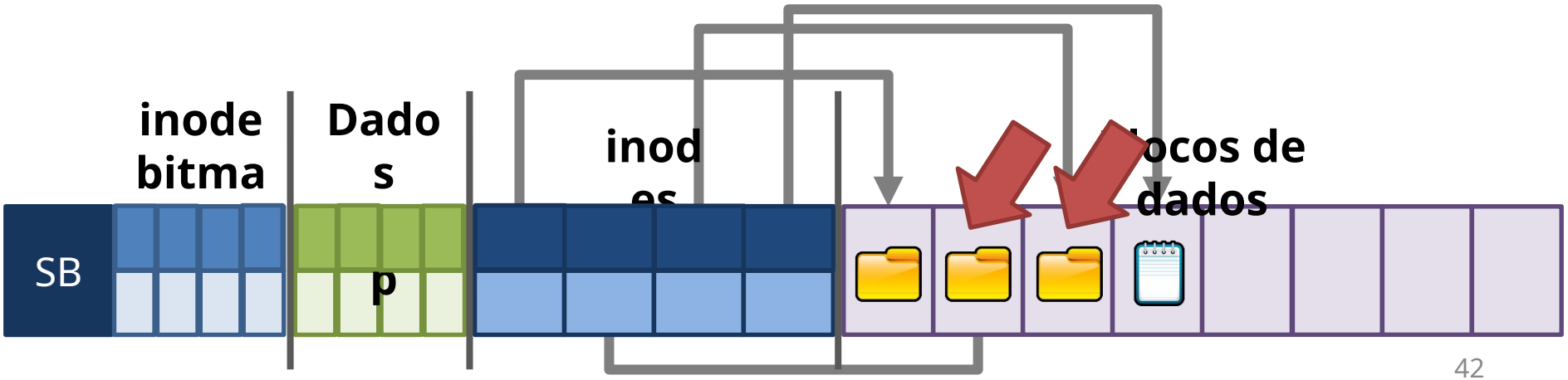
Exemplo hard link

- Várias entradas de diretório podem apontar para o mesmo inode

```
[ rodrigo @ ativ9 ~]ln -T ../CBW/myfile
cbw_file
```



1. Adicionar uma entrada ao diretório “rodrigo”
2. aumentar a LINK_COUNT do "meu arquivo” inode



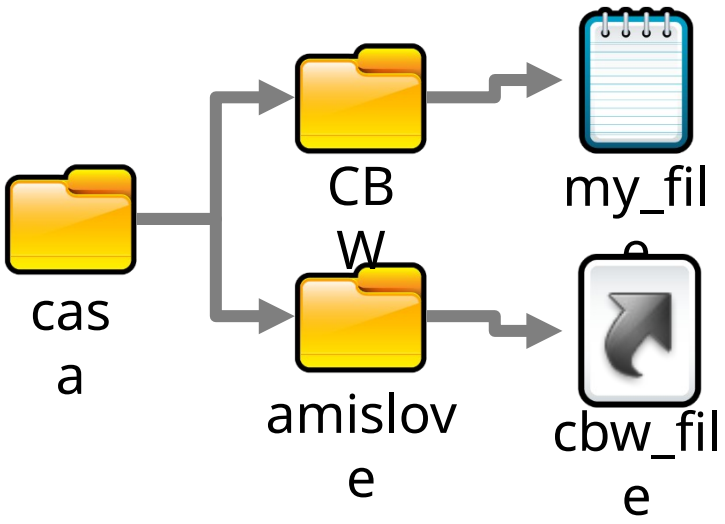
Detalhes do Hard link

- Hard links dar-lhe a capacidade de criar muitos aliases do mesmo arquivo subjacente
  - Podem estar em diferentes diretórios
- arquivo de destino não será marcada como inválida (excluído) até LINK_COUNT == 0
  - É por isso que POSIX “apagar” é chamado *desassociar ()*
- Desvantagem de hard links
  - inodes só são exclusivos dentro de um único sistema de arquivos
  - Assim, somente podem apontar para arquivos na mesma partição

Soft links

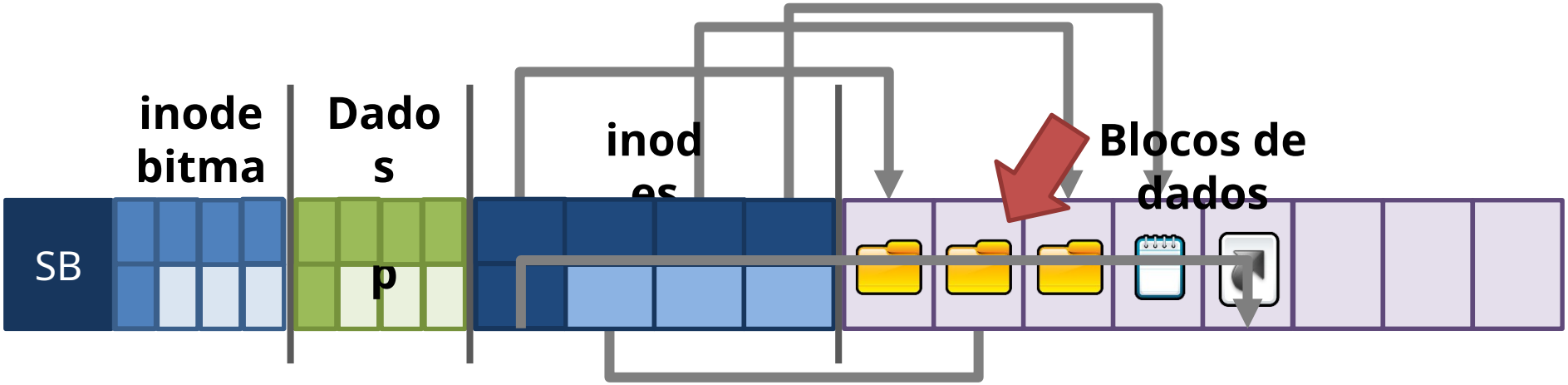
- soft links são arquivos especiais que incluem o caminho para outro arquivo
  - Também conhecido como links simbólicos
  - No Windows, conhecida como atalhos
  - Arquivo pode estar em outro dispositivo ou partição

Exemplo: Soft Link



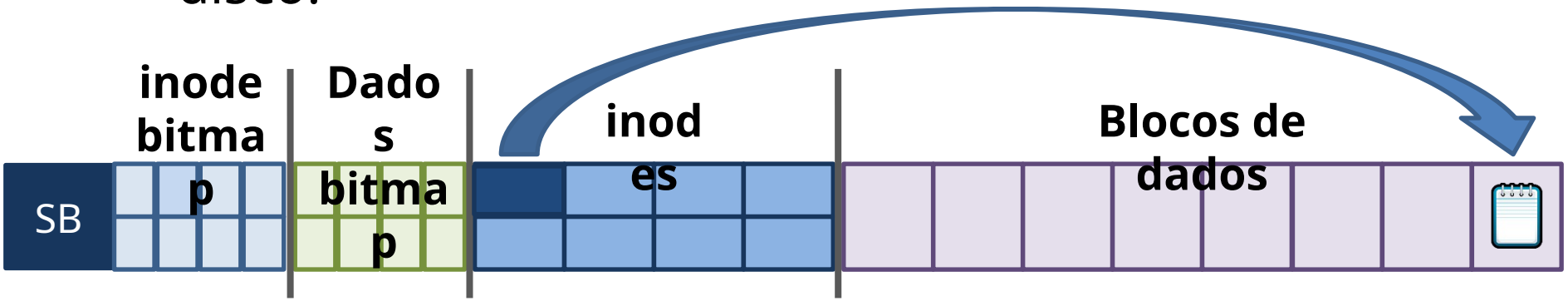

```
[ Amislove @ ativ9 ~]ln -s ../CBW/my_file
cbw_file
```

1. Criar um arquivo de ligação leve
2. Adicioná-lo ao diretório atual



ext: Vantagens e Desvantagens

- Vantagens
  - ext usa I-nodes
  - Todas as funcionalidades típicas de um SA
  - Soft e Hard links
  - Mais performance (menos seeking) do que FAT
- Desvantagens: localidade de referência ruim
  - ext é otimizado para arquivos de um certo tamanho
  - No entanto, não é otimizado para HDDs
  - inodes e dados associados estão muito distantes no disco!



- Partições e montagem
- O básico (FAT)
- inodes e blocos (ext)
- Grupos de blocos (ext2)
- Journaling (ext3)
- Extensões e Árvores-B (ext4)
- SAs baseados em Log

Como estamos até então...

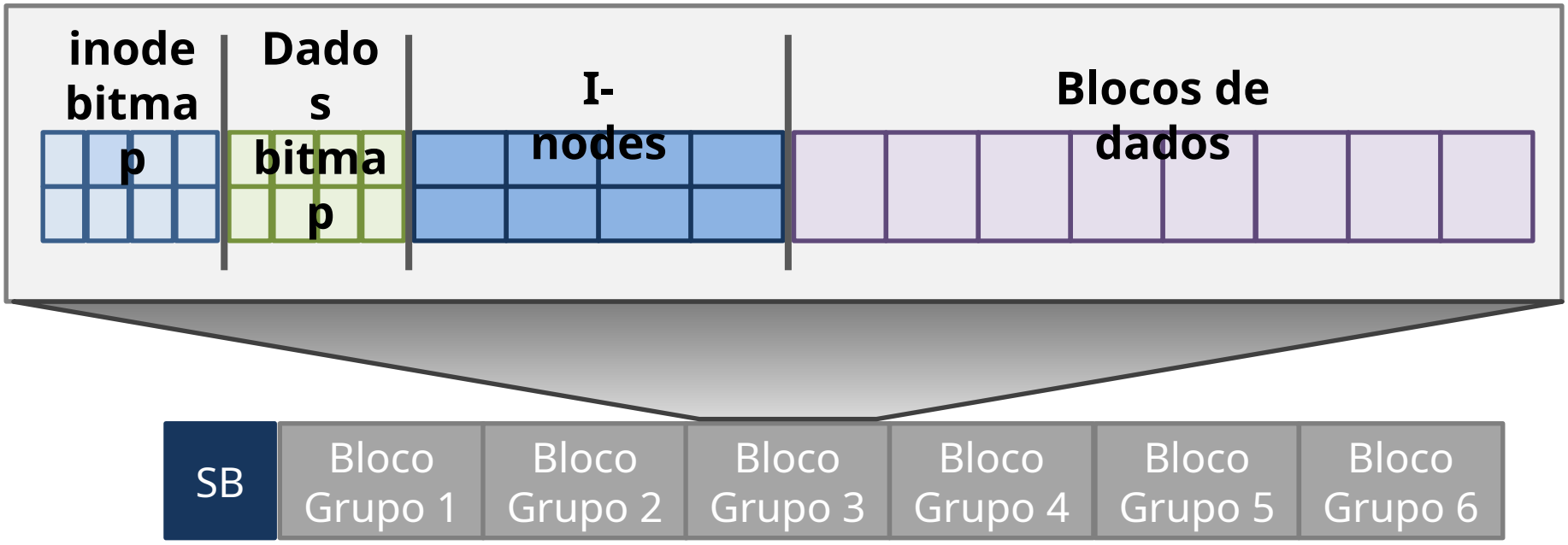
- Neste ponto saímos da FAT para o EXT
  - I-nodes são árvores debalanceadas de blocos de dados
  - Otimizado para o caso comum: pequenos arquivos
- Problema: ext tem fraca localidade de referência
  - I-nodes estão longe dos seus dados correspondentes
  - Isso vai resultar em longa busca pelo disco
- Problema: ext é propenso a fragmentação
  - ext escolhe os primeiros blocos disponíveis para novos dados
  - Nenhuma tentativa é feita para manter os blocos de um arquivo contíguo

Fast File System (FFS)

- FFS desenvolvido em Berkeley em 1984
  - Primeira tentativa de um sistema de arquivos disk aware
  - ou seja otimizado para desempenho em discos giratórios
- Observação: os processos tendem a acessar os arquivos que estão nos mesmos diretórios (ou próximos)
  - localidade espacial
- ideia-chave: colocar diretórios e seus arquivos em grupos de cilindros
  - Introduzido no ext2, chamado grupos de blocos

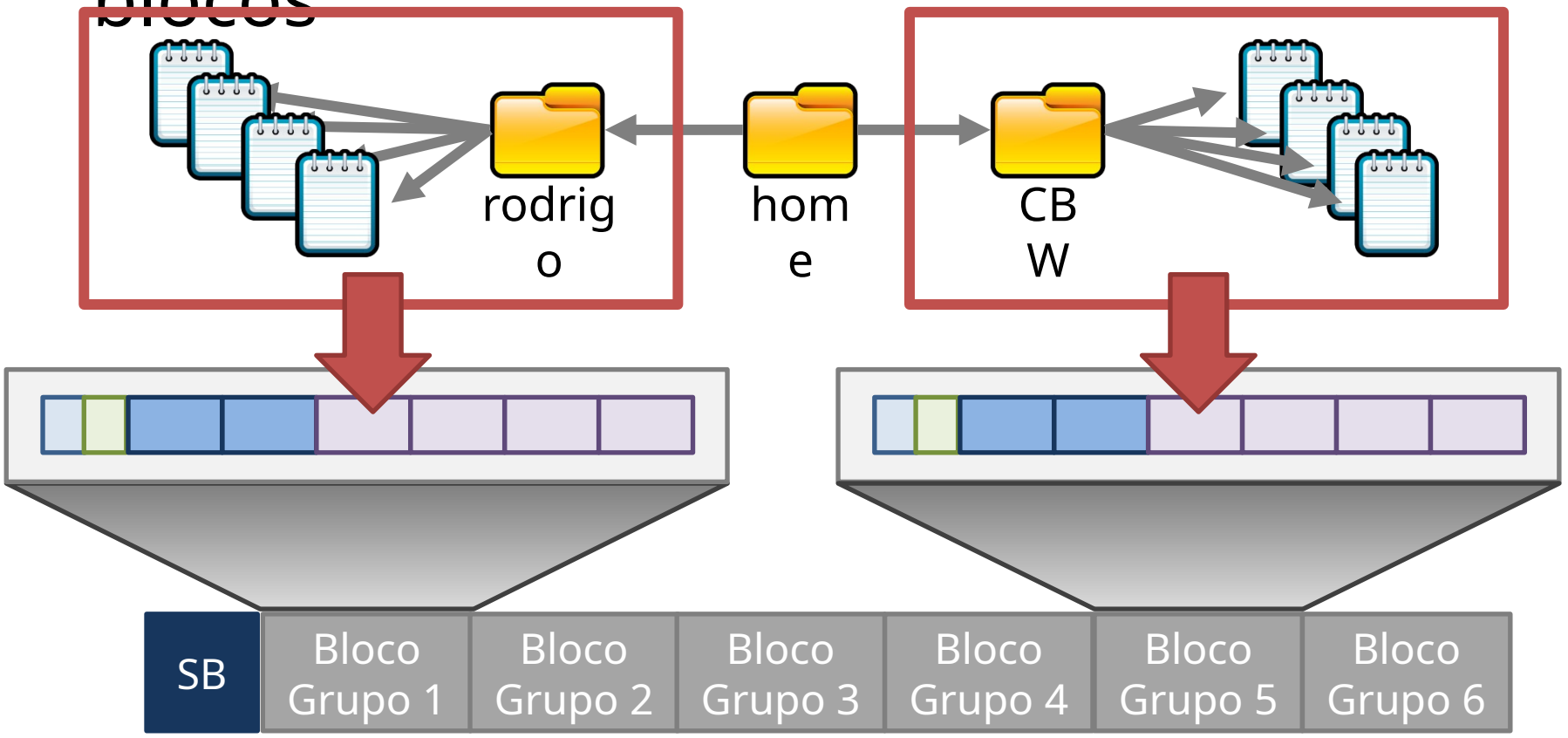
Grupos bloco

- Em ext, Existe um único conjunto de estruturas de dados chave
  - Um bitmap de dados, um inode bitmap
  - 1 inode tabela, uma série de blocos de dados
- Em ext2, cada grupo de blocos contém as suas próprias estruturas de dados chave



Política de alocação

- ext2 tenta manter arquivos e diretórios relacionados dentro do mesmo grupo de blocos



ext2: Vantagens e Desvantagens

- Vantagens - ext2 suporta:
  - Todos os recursos do ext...
  - ... com um desempenho ainda melhor (devido ao aumento da localidade espacial)
- Desvantagens
  - arquivos grandes devem cruzar os grupos de blocos
  - Como o sistema de arquivos torna-se mais complexa, a chance de sistema de arquivos corrompidos cresce
    - por exemplo i-nodes inválidos, Entradas de diretório incorretas, etc.

- Partições e montagem
- O básico (FAT)
- inodes e blocos (ext)
- Grupos de blocos (ext2)
- Journaling (ext3)
- Extensões e Árvores-B (ext4)
- SAs baseados em Log

Como estamos até então...

- Neste ponto, temos um sistema de arquivo cheio de recursos
  - diretórios
  - alocação de dados de granulação fina
  - Hard links / soft links
- Sistema de arquivos otimizado para discos giratórios (HDDs)
  - I-nodes são otimizados para arquivos pequenos
  - Grupos de Bloco melhorar localidade espacial
- Qual é o próximo?
  - Consistência e confiabilidade

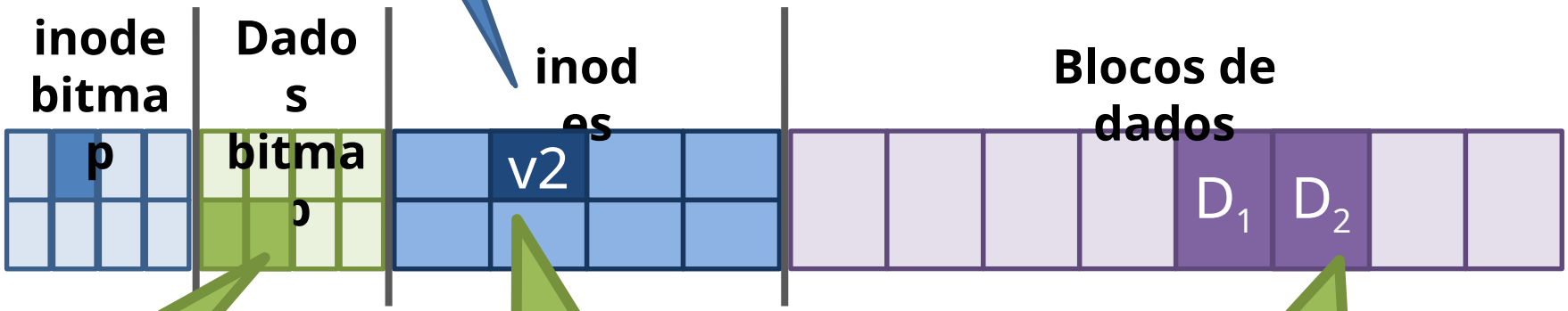
Manter a consistência

- Muitos resultados de operações em várias gravações, independentes para o sistema de arquivos
  - Exemplo: acrescentar um bloco para um arquivo existente
  1. Atualizar bitmaps de blocos livres e i-nodes livres
  2. Atualizar o I-node
  3. Escrever os dados do usuário
- O que acontece se o computador trava no meio deste processo?

Exemplo: Adicionando dados

proprietário:
João
permissões: rw
tamanho: 2
ponteiro: 4
ponteiro: 5
ponteiro: nulo
ponteiro:
nulo

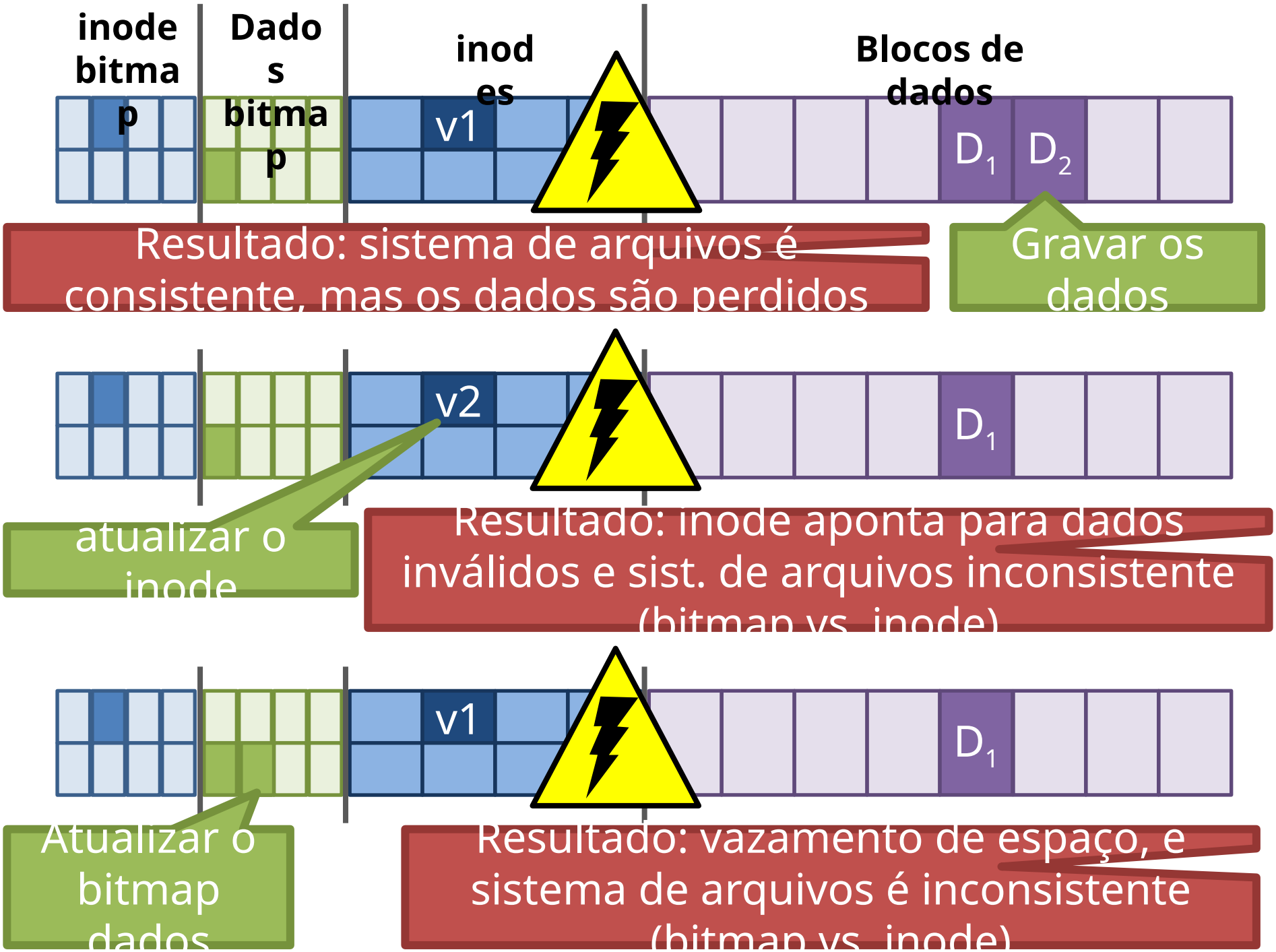
- Estas três operações potencialmente pode ser feito em qualquer ordem
- ... mas o sistema pode falhar a qualquer momento

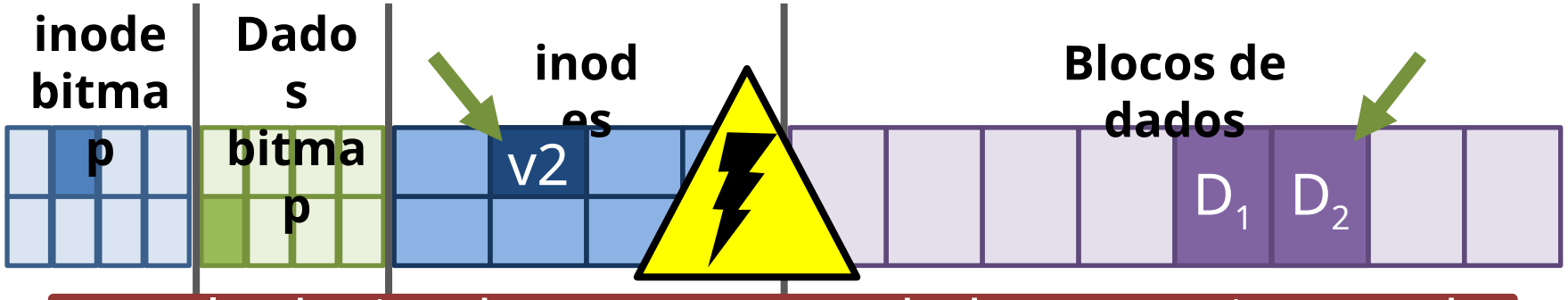


Atualizar o bitmap dados

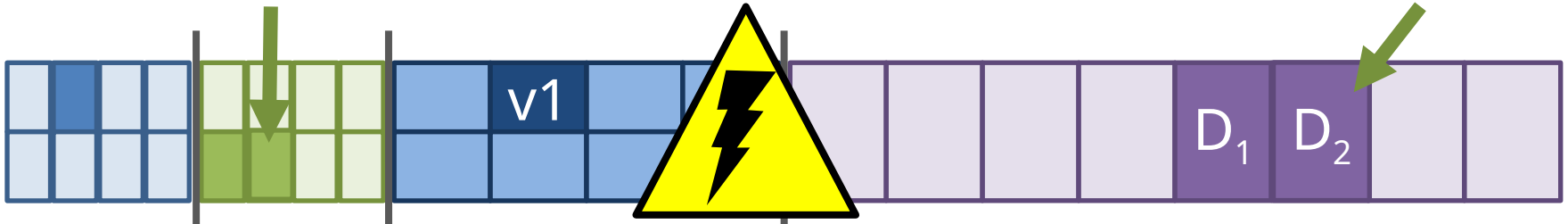
atualizar o inode

Gravar os dados

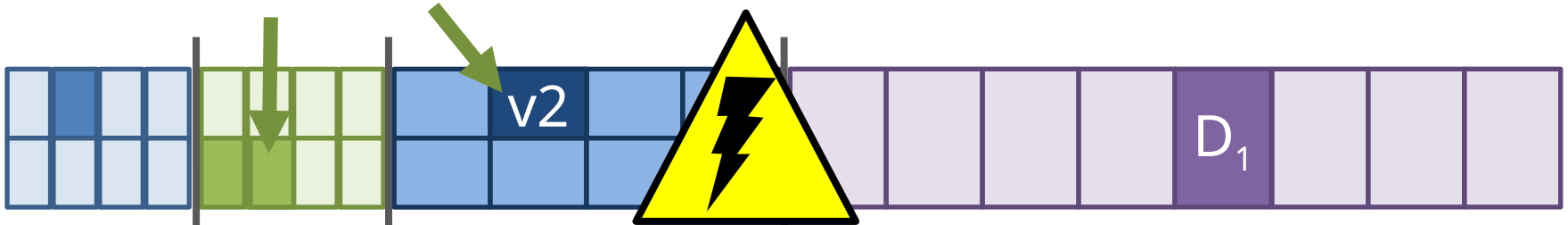

inode bitmap
Dados bitmap
inodes
Blocos de dados
v1
D1
D2
Resultado: sistema de arquivos é consistente, mas os dados são perdidos
Gravar os dados
v2
D1
atualizar o inode
Resultado: inode aponta para dados inválidos e sist. de arquivos inconsistente (bitmap vs. inode)
v1
D1
Atualizar o bitmap dados
Resultado: vazamento de espaço, e sistema de arquivos é inconsistente (bitmap vs. inode)



Resultado: inode aponta para dados, mas sistema de arquivos é inconsistente



Resultado: sistema de arquivos é inconsistente, e os dados são inúteis, uma vez que não está associada a um inode



Resultado: sistema de arquivos é consistente, mas o inode aponta para dados de lixo

Consistência em Falhas

- O disco garante que gravações em setores são atômicas
  - Nenhuma maneira de fazer escritas multi-setores ser atômica
- Como garantir a consistência depois de um acidente?
  1. Não se preocupe em assegurar a consistência
     - Aceitar que o sistema de arquivos podem ser inconsistentes depois de um acidente
     - Executar um programa que corrige o sistema de arquivos durante inicialização
     - verificador de sistema de arquivos (*fsck*)
  2. Use um log de transações para fazer a multi-escrita atômica
     - O registo guarda um histórico de todas as gravações para o disco
     - Depois de uma falha o registro pode ser “repetido” para concluir as atualizações
     - sistema de arquivos journaling

Abordagem 1: File Checker

- ideia-chave: consertar sistemas de arquivos inconsistentes durante inicialização
  - utilitário Unix chamado *fsck* (*chkdsk* no Windows)
  - Percorre todo o sistema de arquivo várias vezes, para identificar e corrigir inconsistências
- Porque durante inicialização?
  - Nenhuma outra atividade sistema de arquivos pode estar acontecendo
  - Depois que fsck o ocorre a

Tarefas do fsck

- **superbloco:** validar o superbloco, substituí-lo por um backup se ele está corrompido
- **blocos livres e inodes:** reconstruir os bitmaps escaneando todos os inodes
- **Acessibilidade:** certificar-se de tudo inodes são acessíveis a partir da raiz da sistema de arquivo
- **I-nodes:** excluir todos I-nodes corrompidos e construir suas contagens de ligação por níveis da árvore de diretórios
- **diretórios:** verificar a integridade de todos os diretórios
- ... e muitas outras verificações de consistência menores

fsck: Vantagens e Desvantagens

- Vantagens de *fsck*
  - Não requer que o sistema de arquivos faça qualquer trabalho para garantir a consistência
  - Torna a implementação do sistema de arquivos mais simples
- desvantagens de *fsck*
  - Muito complicado para implementar o programa *fsck*
    - Muitas inconsistências possíveis que devem ser identificados
    - Muitos casos de canto difícil considerar e tratar
  - *fsck* é **muito lento!!**
    - Percorre todo o sistema de arquivo várias vezes
    - Imagine quanto tempo que seria necessário para fsck uma matriz RAID de 40 TB

Abordagem 2: Journaling

- Problema: *fsck* é lento porque ele verifica todo o sistema de arquivos após um acidente
  - O que se soubessemos onde as últimas gravações foram antes do acidente, e verifiquei apenas aqueles?
- Ideia-chave: fazer escrita transacional usando um log write-ahead
  - Comumente referido como uma Journal
- Ext3 e NTFS usam journaling

Log Write-Ahead

- ideia-chave: escritas para o disco são primeiro escritas em um log
  - Depois que o log é escrito, as gravações executam normalmente
  - Em essência, o log registra transações
- O que acontece depois de um ‘acidente’ ...
  - Se as gravações no log são interrompidas?
    - A transação está incompleta
    - Os dados do usuário é perdido, mas o sistema de arquivos é consistente
  - Se as gravações no log tem sucesso, mas as gravações normais são interrompidos?
    - O sistema de arquivos podem ser inconsistentes, mas ...
    - O registo tem exatamente as informações corretas para corrigir o problema

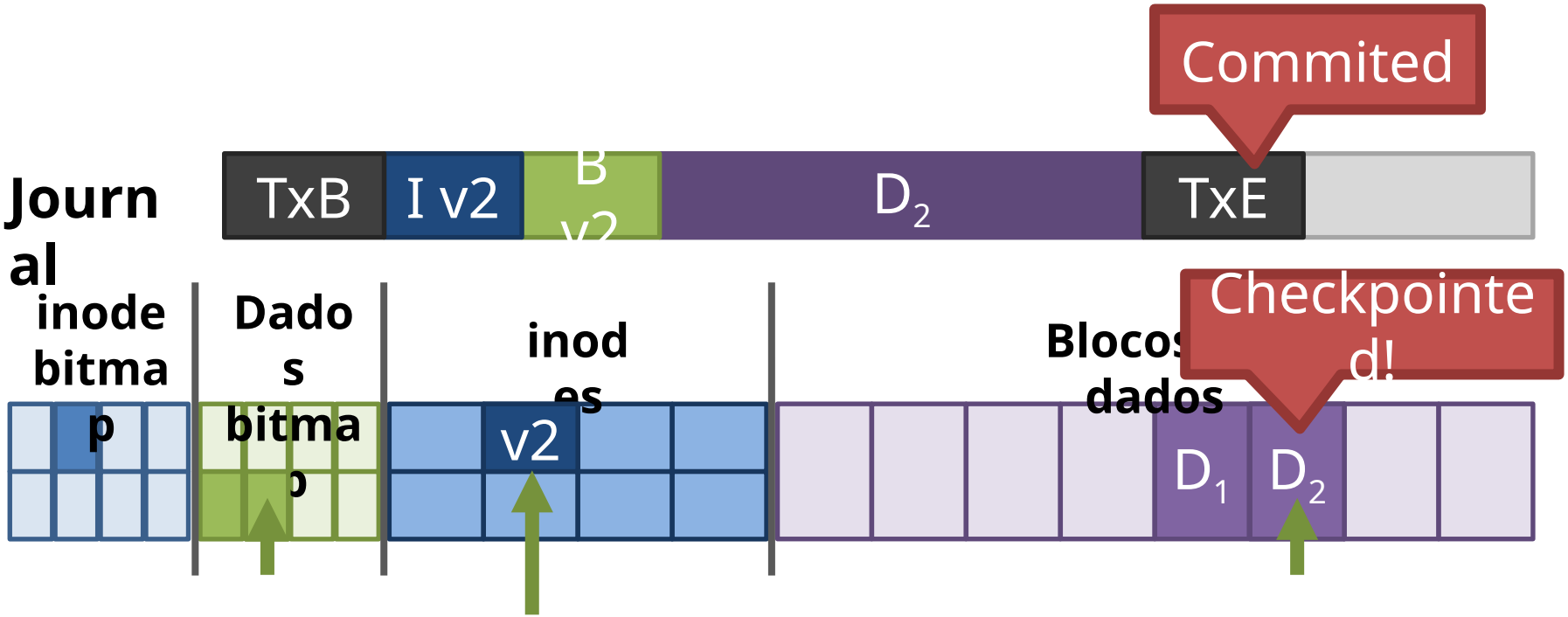
Exemplo journaling de dados

- Suponha que estamos anexando dados a um arquivo
  - Três escritas: inode V2, V2 bitmap dados e os dados D_2
- Antes de executar essas gravações, primeiro loga elas

1. Comece uma nova transação com um único ID =*k*
2. Escreva o bloco de meta-dados atualizado (s)
3. Escreva o bloco de dados de arquivo (s)
4. Escrever um fim-de-transação com ID =*k*

Commits e Checkpoints

- Dizemos uma transação é commitada depois que todas as gravações no log estão completas
- Depois de uma transação é confirmada, o sistema operacional 'checkpointa' a atualização



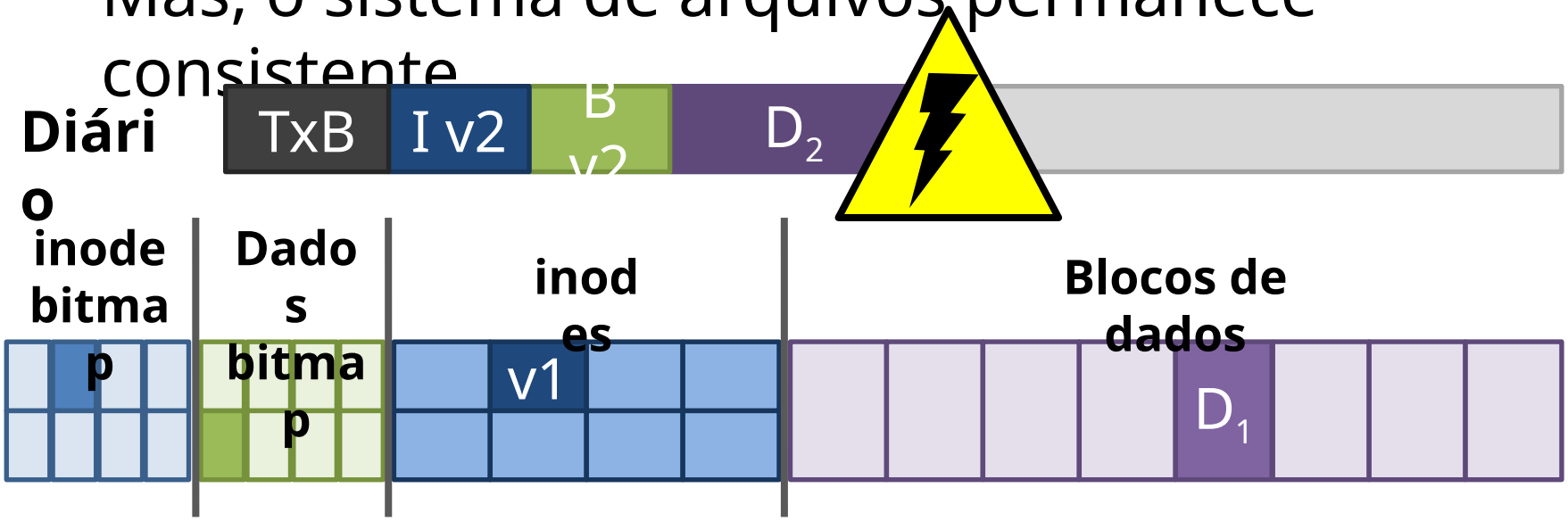
- Passo final: liberar a transação 'checkpointed'

Journal Implementação

- Journals são tipicamente implementados como um buffer circular
  - Journal é **append-only**
- SO mantém ponteiros para a frente e para trás das transações no buffer
  - Quando as transações são liberadas, a parte traseira é movida para frente
- Assim, o conteúdo do journal nunca é excluído, eles são apenas substituídos ao longo do tempo
  - Evita uma escrita e melhora a performance

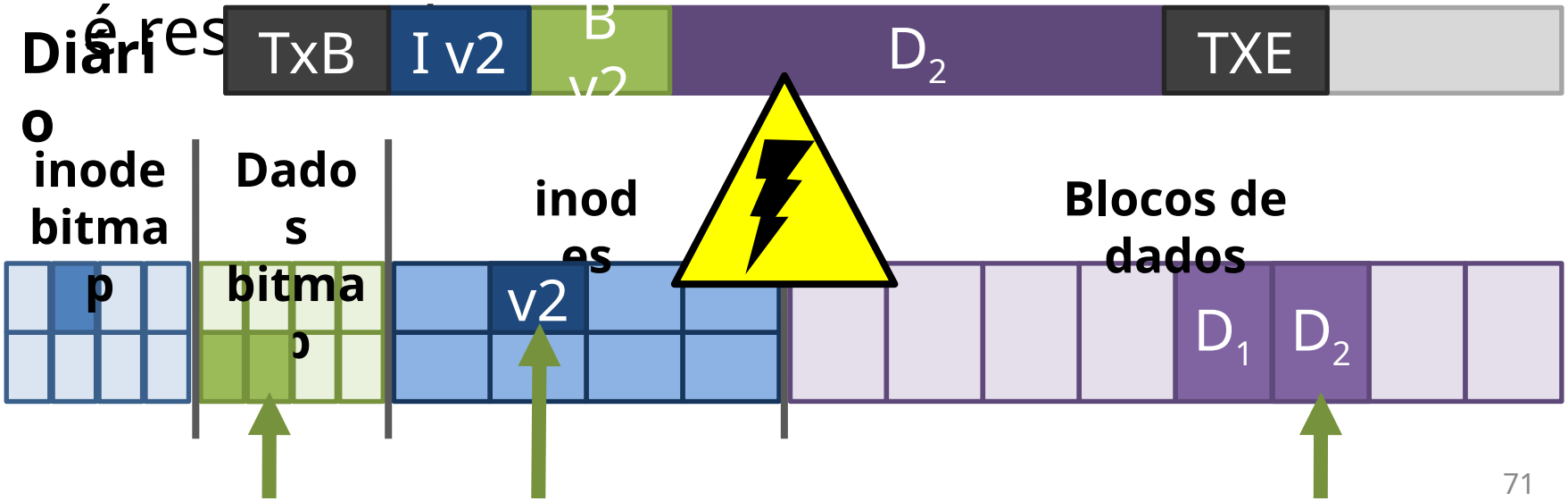
Recuperação (1)

- E se o sistema falhar durante o registro?
  - Se a transação não é confirmada, os dados são perdidos
  - Mas, o sistema de arquivos permanece consistente



Recuperação (2)

- E se o sistema falhar durante o checkpoint?
  - sistema de arquivos podem ser inconsistentes
  - Durante a reinicialização, as transações que estão commitadas, mas não livres são reproduzidos em ordem
  - Assim, nenhum dado é perdido e a consistência é res



Transações corrompidas

- Problema: o escalonador de disco pode não executar escrita em ordem
  - Registros no log podem aparecer commitadas, quando na verdade elas são inválidas

- ção: adi
‘checksum’ de verificação para TxB
- Durante a recuperação, rejeitar transações com ‘checksums’ inválidos
- Implementado no Linux em

Transa
válida, mas os dados estão faltando!
- Durante a reprodução, dados de lixo são escritos para o sistema de

Journaling: Vantagens e Desvantagens

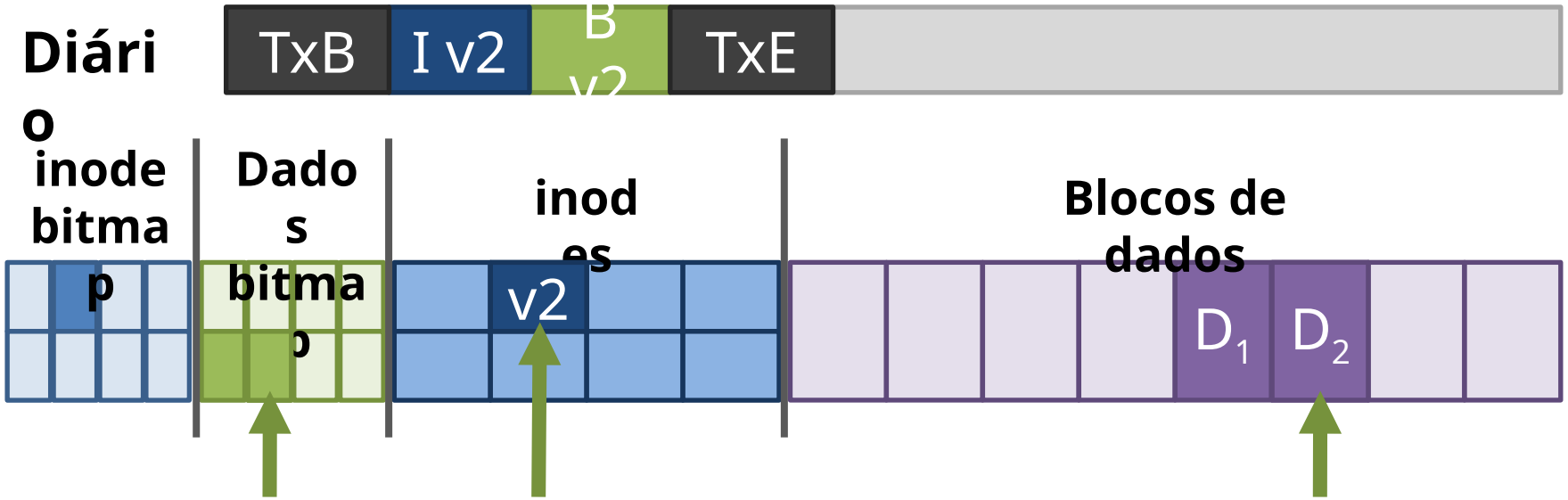
- Vantagens de journaling
  - Robusto: a recuperação do sistema de arquivos é rápida
    - Não há necessidade de percorrer todo o sistema de revista ou arquivo
  - Relativamente simples de implementar
- Desvantagens de journaling
  - Tráfego de escrita em disco é duplicado
    - Especialmente os dados do arquivo, que é provavelmente grande
  - Exclusões são muito difíceis de fazer corretamente
    - Exemplo, em alguns slides ...

Acelerando o Journaling

- Journaling gera sobrecarga de escrita
- SOs atualizações em lote vão para o Journal
  - Bufferizar gravações sequenciais na memória, em seguida, emitir uma grande gravação para o log
  - Exemplo: lotes ext3 atualizam a cada 5 segundos
- Compromisso entre desempenho e persistência
  - longo intervalo de lote = menor, maior escrita para o log
    - Melhor desempenho devido a grandes gravações seqüenciais
  - Mas, se houver um acidente, tudo no buffer será perdido

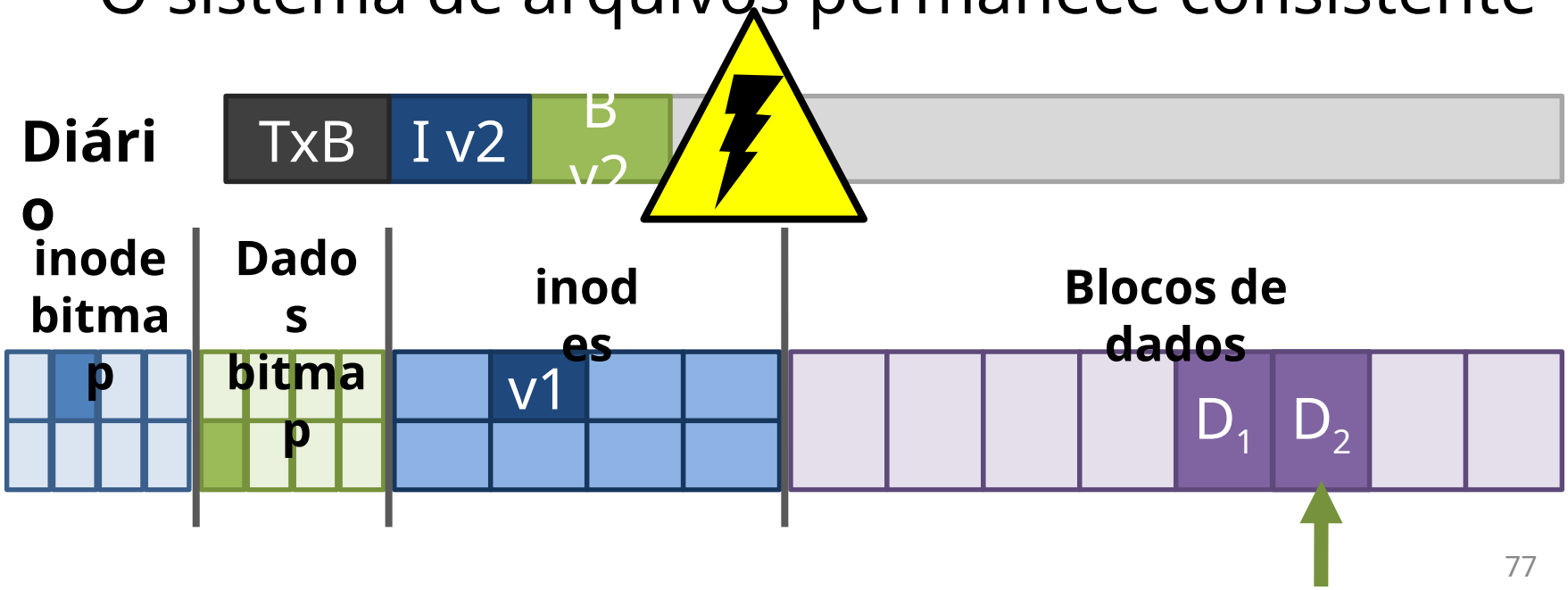
Journaling de Meta-Dados

- A parte mais cara de journaling de dados está na escrita duplicada de dados
  - Meta-dados é pequena (~ 1 setor).
  - Os dados dos arquivos é que são grandes
  - ext3 implementa journaling de meta-dados!



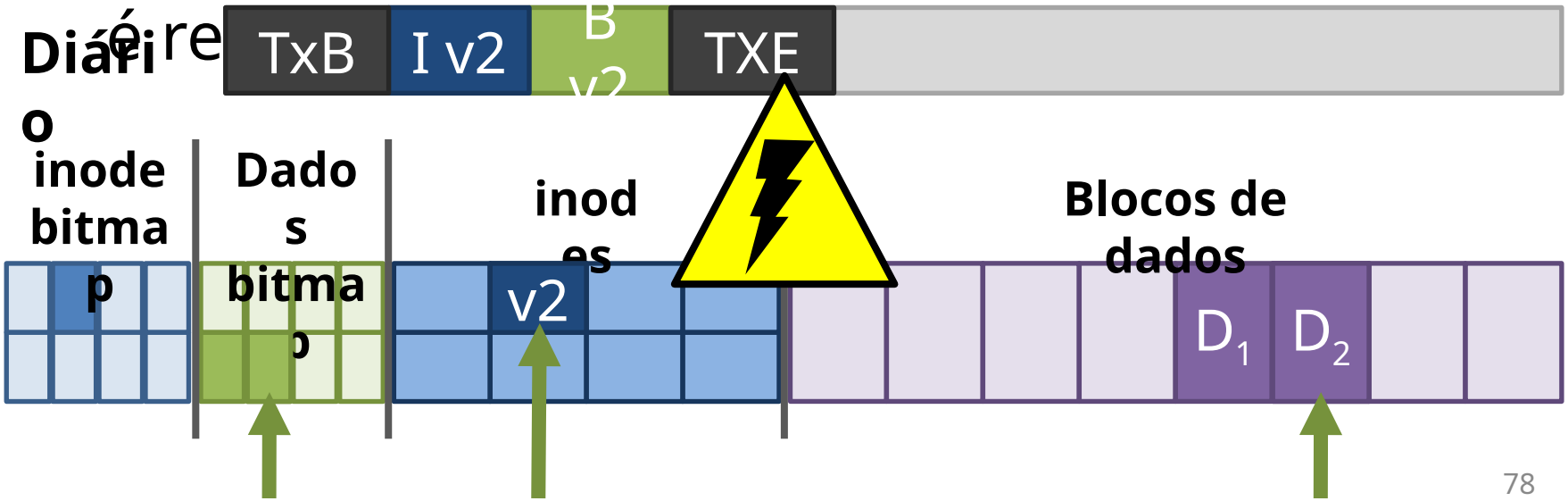
Recuperação (1)

- E se o sistema falhar durante o registro?
  - Se a transação não é confirmada, os dados são perdidos
  - D_2 acabará por ser substituído
  - O sistema de arquivos permanece consistente

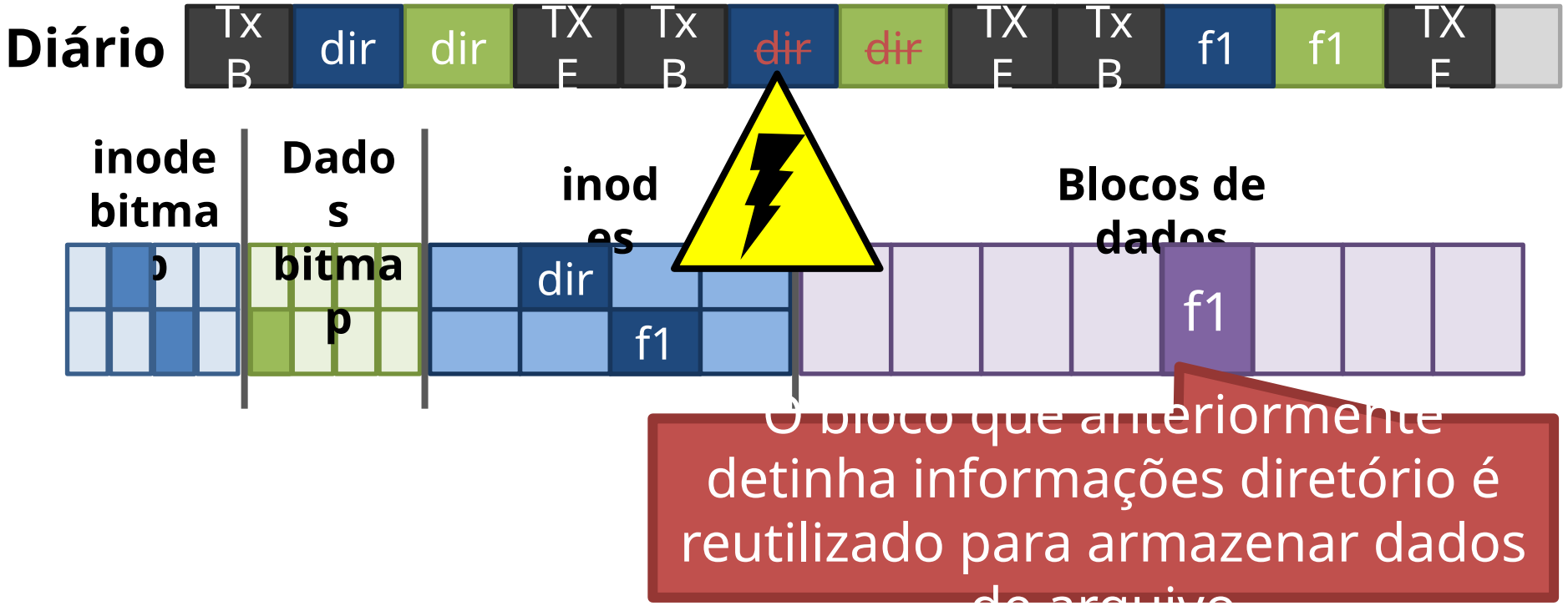


Recuperação (2)

- E se o sistema falhar durante o checkpoint?
  - sistema de arquivos podem ser inconsistentes
  - Durante a reinicialização, as transações que estão comprometas, são reproduzidas em ordem
  - Assim, nenhum dado é perdido e a consistência é re



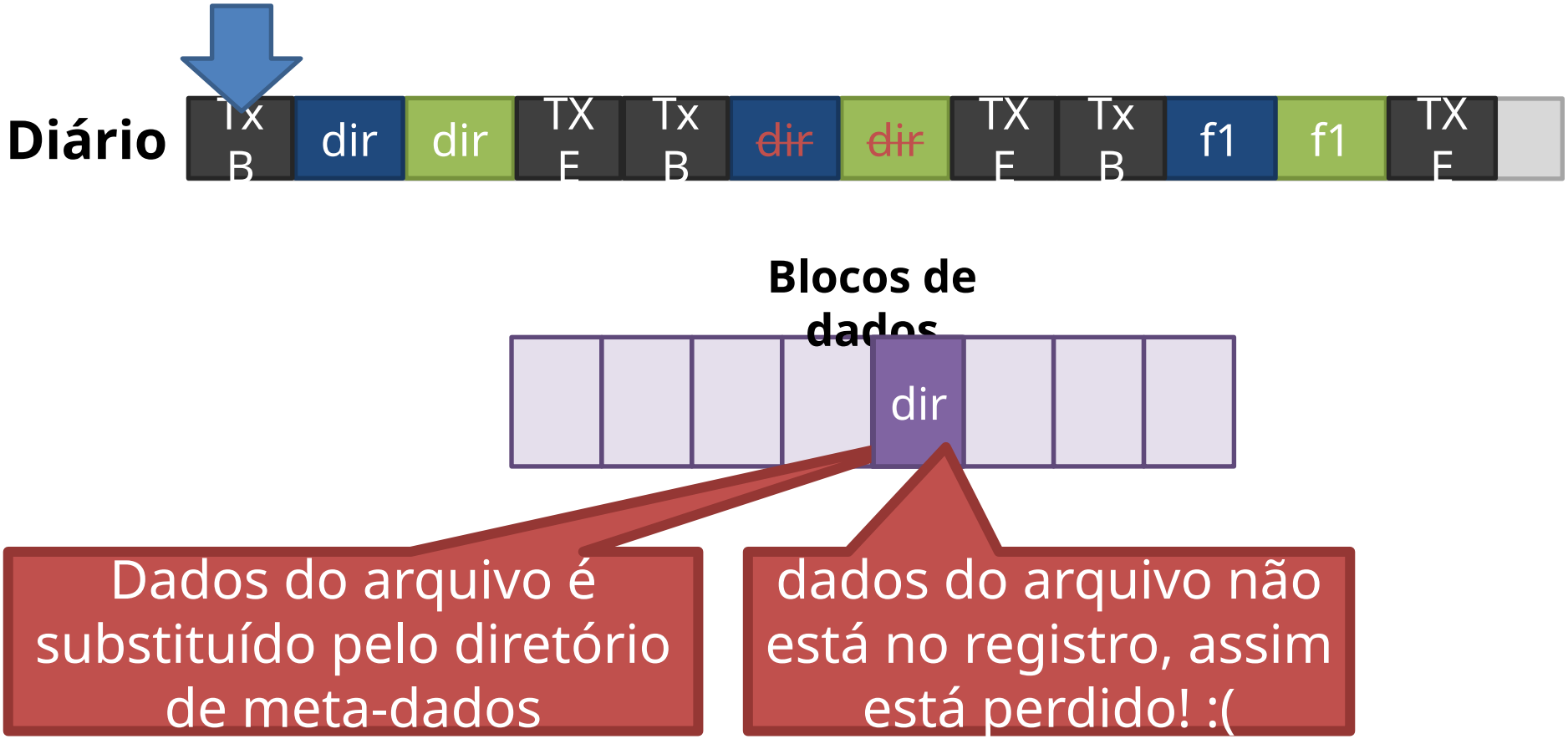
Exclusão e Reuso de Blocos



1. Criar um diretório: inode e os dados são escritos
2. Exclua o diretório: inode é removido
3. Criar um arquivo: inode e os dados são

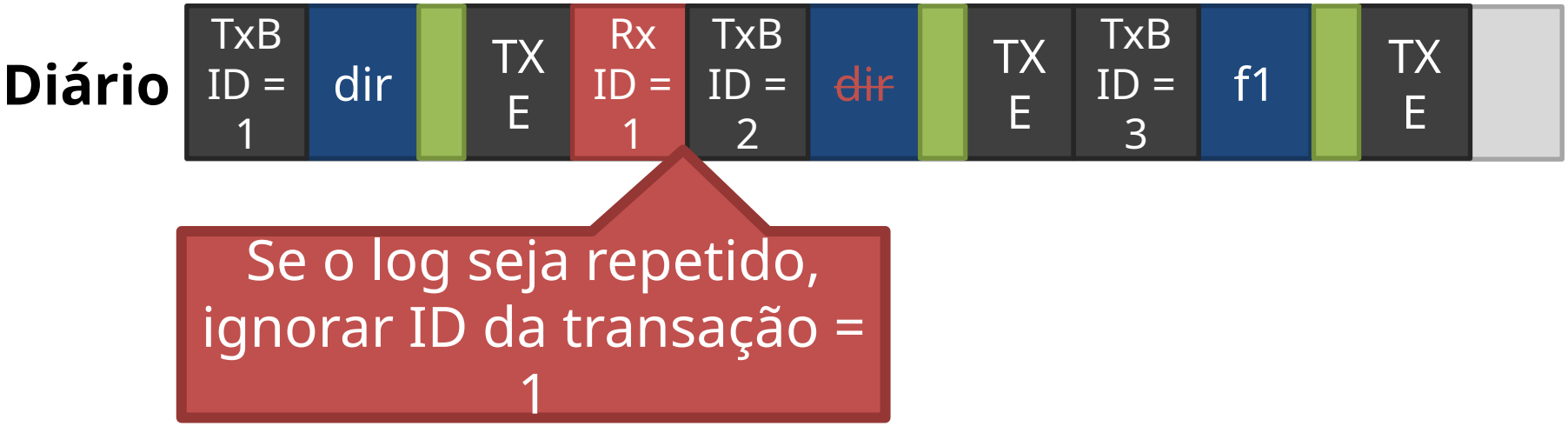
Problemas com Exclusão

- O que acontece quando o log for repetido depois de um crash?



Exclusão

- Estratégia 1: não reutilizar blocos até que a exclusão é checkpointed e liberada
- Estratégia 2: adicionar um revogar registro para o log
  - ext3 utiliza registros Revogar



Journaling wrap-up

- Hoje, a maioria SOs usar diário sistemas de arquivos
  - ext3 / ext4 no Linux
  - NTFS no Windows
- Proporciona excelente recuperação de falhas com relativamente pouco espaço e sobrecarga de desempenho
- Próxima geração SOs provavelmente vai passar para sistemas com a semântica copy-on-write arquivo
  - bTRFs e zfs no Linux

- Partições e montagem
- O básico (FAT)
- i-nodes e blocos (ext)
- Grupos de blocos (ext2)
- Journaling (ext3)
- Extensões e Árvores-B (ext4)
- SAs baseados em Log (btrfs, zfs)

Como estamos até então...

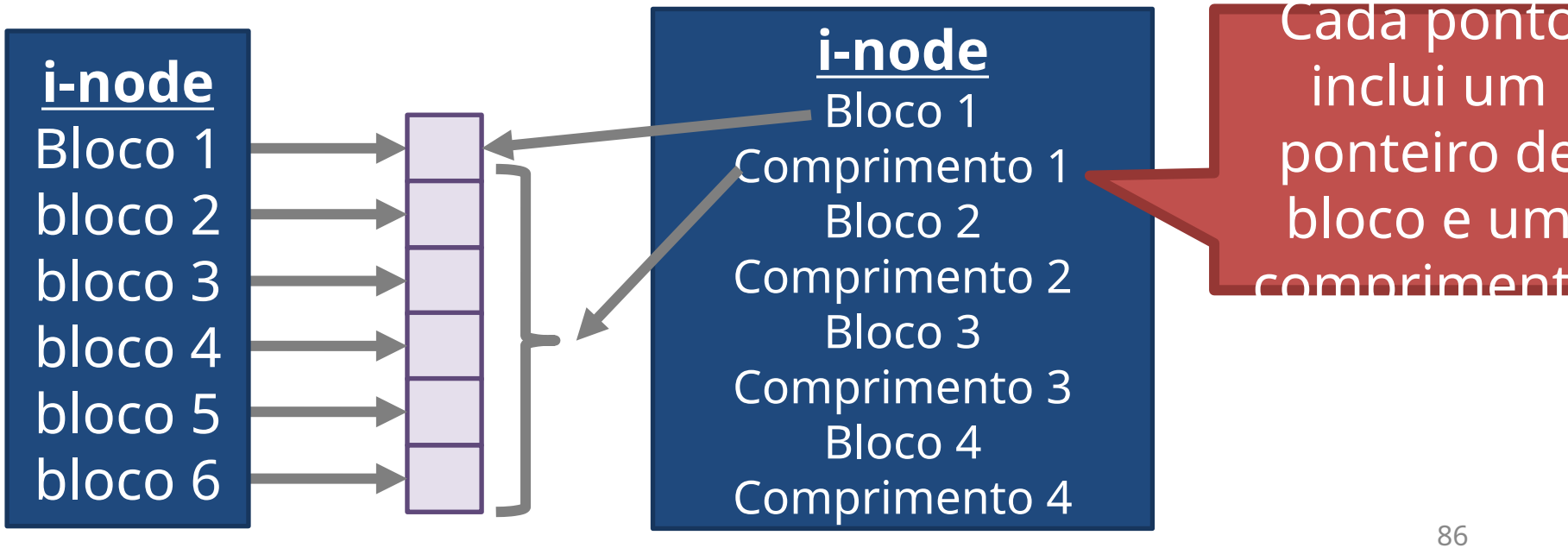
- Neste ponto:
  - Não só temos um sistema de arquivos rápido
  - Mas também é resistente contra a corrupção
- Qual é o próximo?
  - Mais melhorias de eficiência!

Revisando os nodes

- Lembre-se: i-nodes usam indireção para adquirir blocos adicionais de ponteiros
- Problema: i-nodes não são eficientes para arquivos grandes
  - Exemplo: para uma arquivo de 100MB, você precisa 25600 ponteiros bloco (assumindo blocos de 4 KB)
- Isto é inevitável se o arquivo é 100% fragmentada
  - No entanto, se grandes grupos de blocos são contíguos?

De Ponteiros para Extensões

- Sistemas de arquivos modernos se esforçam para minimizar a fragmentação
  - Uma vez que resulta em muitos procura, assim, baixo desempenho
- Extensões (ou extents) são mais adequados para arquivos contíguos



Implementação Extensões

- ext4 e NTFS usam extensões (ou extents)
- i-nodes ext4 incluem 4 extensões ao invés de ponteiros de bloco
  - Cada medida pode dirigir-se a mais de 128 MB de espaço contíguo (assumindo blocos de 4 KB)
  - Se forem necessários mais extensões, um bloco de dados é alocado
  - Semelhante a um bloco de ponteiros indiretos

Revisitando Diretórios

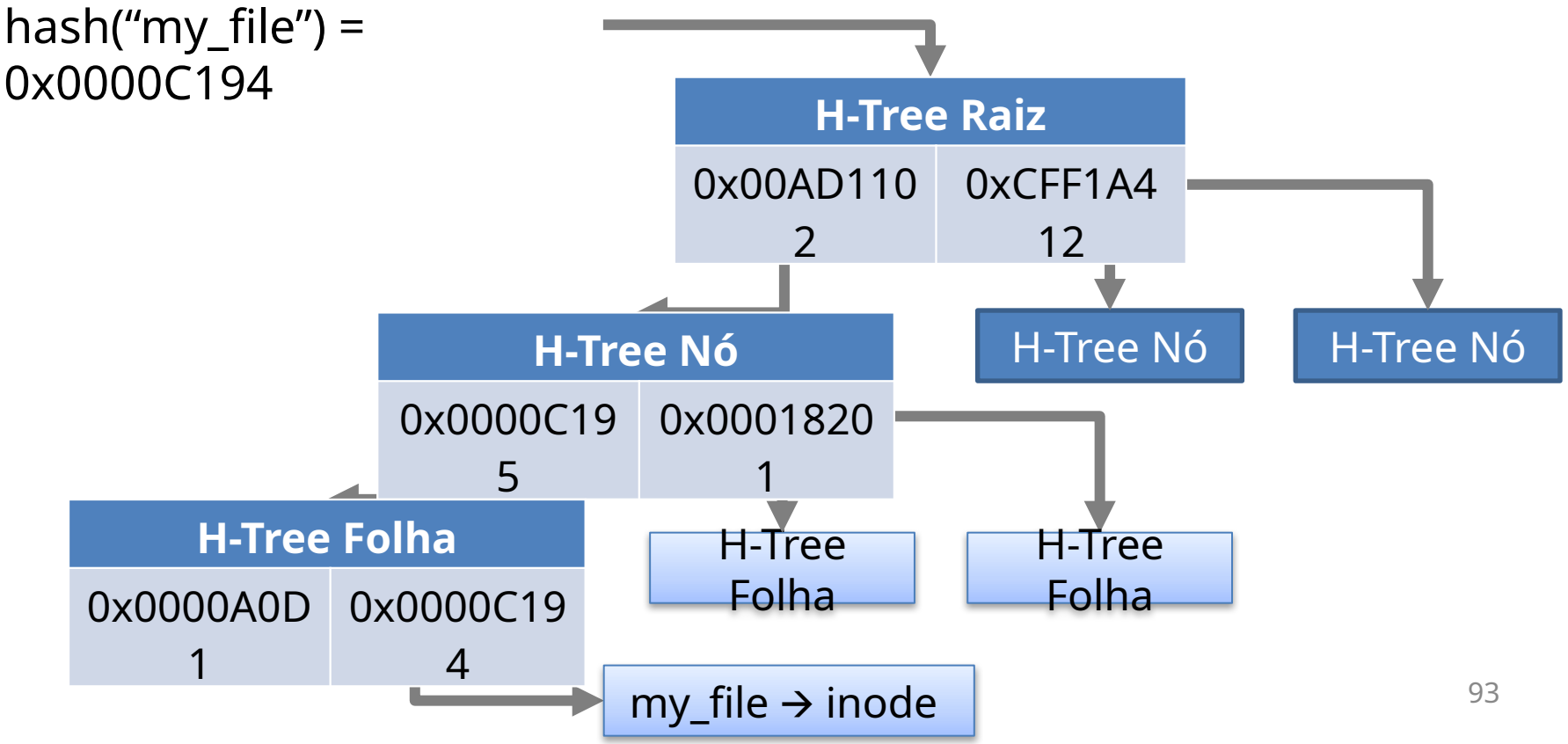
- Em ext, ext2, ext3 e, cada diretório é um arquivo com uma lista de entradas
  - Entradas não são armazenados na ordem de classificação
  - Algumas entradas podem estar em branco, se eles foram apagados
- Problema: em busca de arquivos em diretórios grandes leva tempo O (n)
  - Praticamente, você não pode armazenar mais que 10K arquivos em um diretório
  - Leva muito tempo para localizar e abrir arquivos

De listas de B-Trees

- Diretórios ext4 e NTFS são armazenados como Árvores-B para melhorar o tempo de pesquisa para O(log N)
- A Árvore-B é um tipo de árvore balanceada que é otimizado para armazenamento em disco
  - Os itens são armazenados na ordem de classificação em blocos
  - Cada bloco armazena entre *m* e 2*m* elementos

Exemplo B-Tree

- ext4 utiliza uma variante da Árvore-B conhecida como H-Tree
  - *H* de hash (Alguns a chamam de Árvore B+)
- Suponha que você tente fopen(“my_file”, “r”);



ext4: Vantagens e Desvantagens

- Vantagens - ext4 (e NTFS) suporta:
  - Todas as funcionalidades básicas do sistema de arquivos que requerem
  - Melhor desempenho de grupos de blocos de ext3
  - Ganhos adicionais de desempenho de extensões (extents) e arquivos do diretório de Árvore-B
- Desvantagens:
  - ext4 é uma melhoria incremental sobre ext3
  - Sistemas de arquivos mais modernos têm características ainda mais agradáveis
    - semântica copy-on-write (BTRFS e ZFS)

- Partições e montagem
- O básico (FAT)
- inodes e blocos (ext)
- Grupos de blocos (ext2)
- Journaling (ext3)
- Extensões e Árvores-B (ext4)
- SAs baseados em Log (btrfs/zfs)

Como estamos até então...

- Neste ponto:
  - Chegamos a um sistema de arquivos moderno, como ext4 e NTFS
- Qual é o próximo?
  - Voltar à prancheta de desenho e reavaliar a partir de requisitos principais

Reavaliando Desempenho de HDs

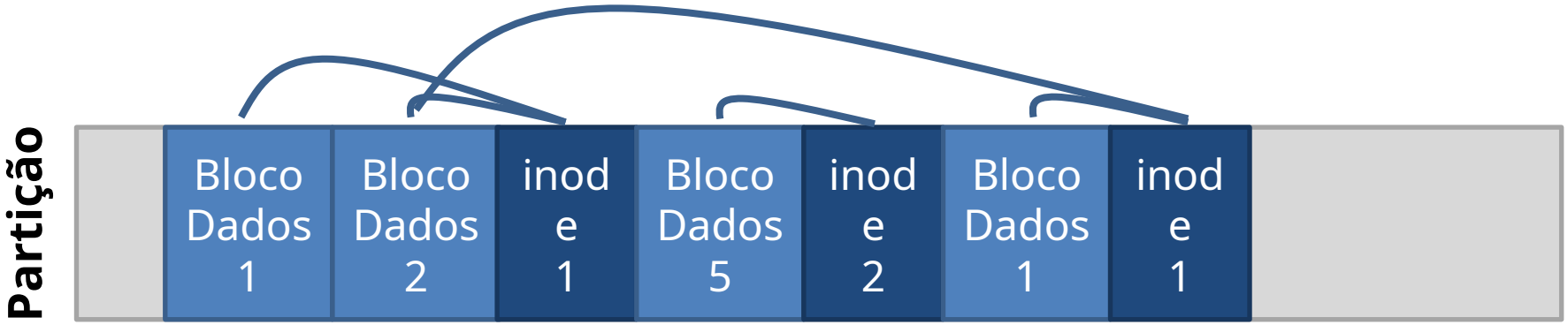
- Como o hardware de computador tem evoluido?
  - RAM tornou-se mais barata e maior :)
  - Acesso aleatório em disco permaneceu muito lento :(
- Esta mudança altera a dinâmica como os discos são usados
  - Mais dados podem ser armazenados em cache na RAM = menos leituras de disco
  - Assim, a escrita vai dominar a E/S em disco
- Podemos criar um sistema de arquivos que é otimizado para escritas seqüenciais?

File System estruturado em Log

- Ideia-chave: bufferizar todas as gravações (incluindo meta-dados) na memória
  - Escrever segmentos longos sequencialmente no disco
- Tratar o disco como um buffer circular, ou seja, não substituir blocos de dados
- Vantagens:
  - As gravações se tornariam grandes e seqüenciais
- Grande pergunta:
  - Como você administra meta-dados e manter a estrutura do SA neste tipo de projeto?

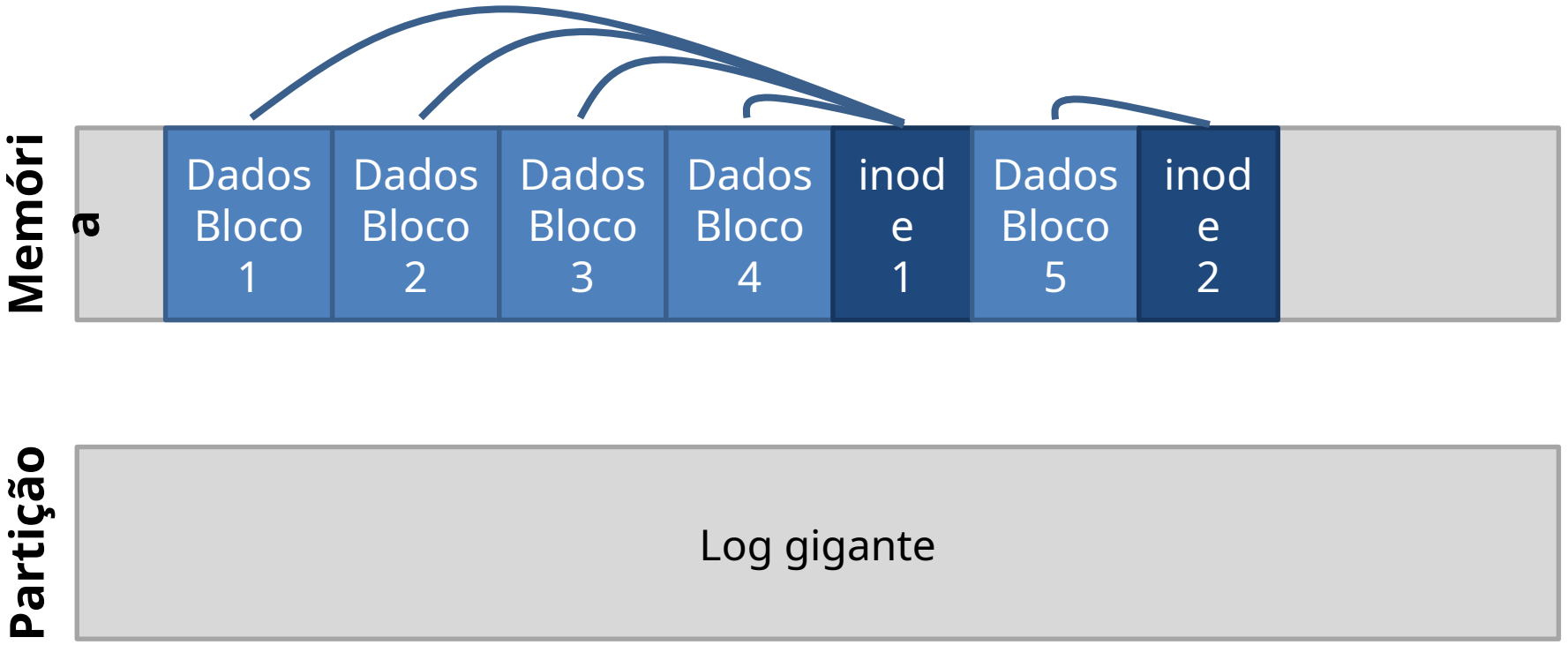
Tratar o disco como um Log

- Mesmo conceito como o “journal”
  - Dados e meta-dados são anexados a um log
  - Dados obsoletos não são sobrescritos, mas sim substituídos – copy-on-write



Buffer de Gravação

- LFS buffers escreve na memória em pedaços (chunks)

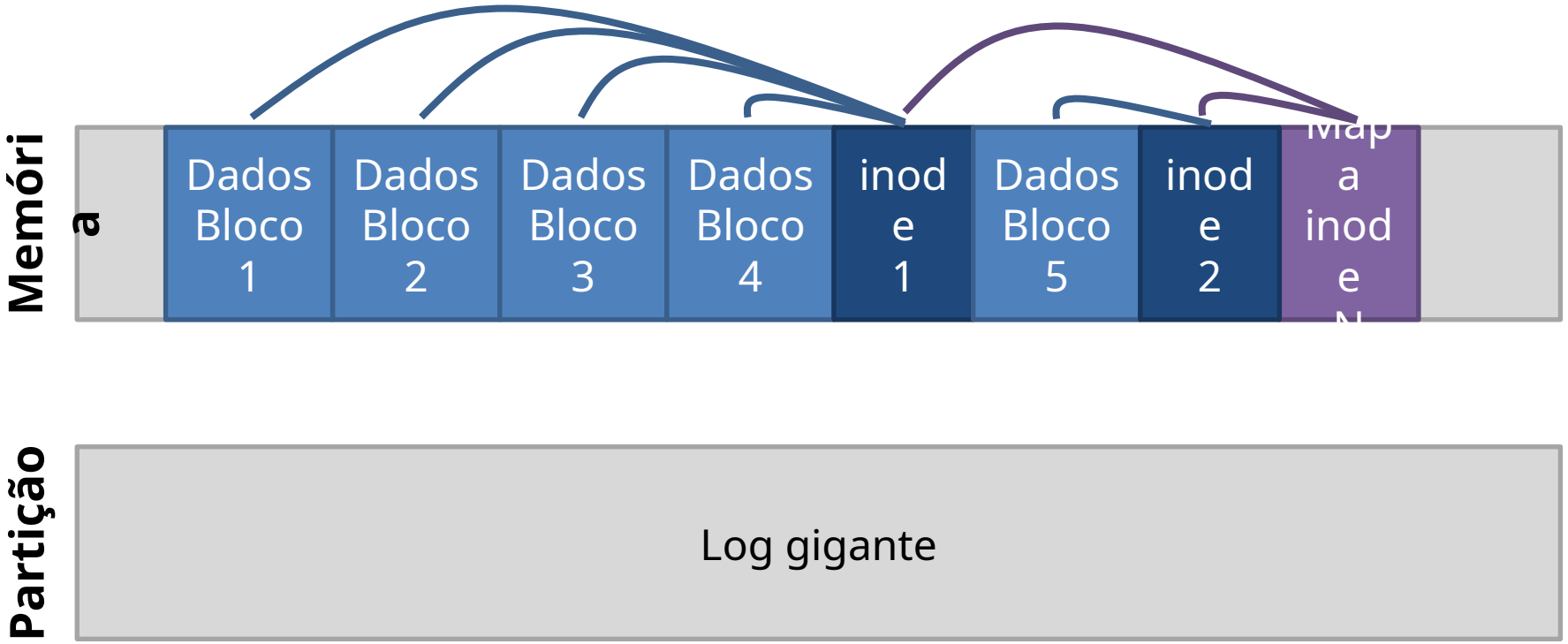


- Pedaços (chunks) são anexados ao log quando eles forem suficientemente grandes

Como encontrar I-nodes

- Em um sistema de arquivos típico, o i-nodes são armazenados em locais fixos (relativamente fáceis de se encontrar)
- Como você encontra i-nodes no log?
  - Lembre-se, pode haver várias cópias de um determinado i-node
- Solução: adicionar um nível de indireção
  - O tradicional mapa de inodes pode ser quebrado em partes
  - Quando uma porção do mapa de i-nodes for atualizado, escrevê-lo para o log!

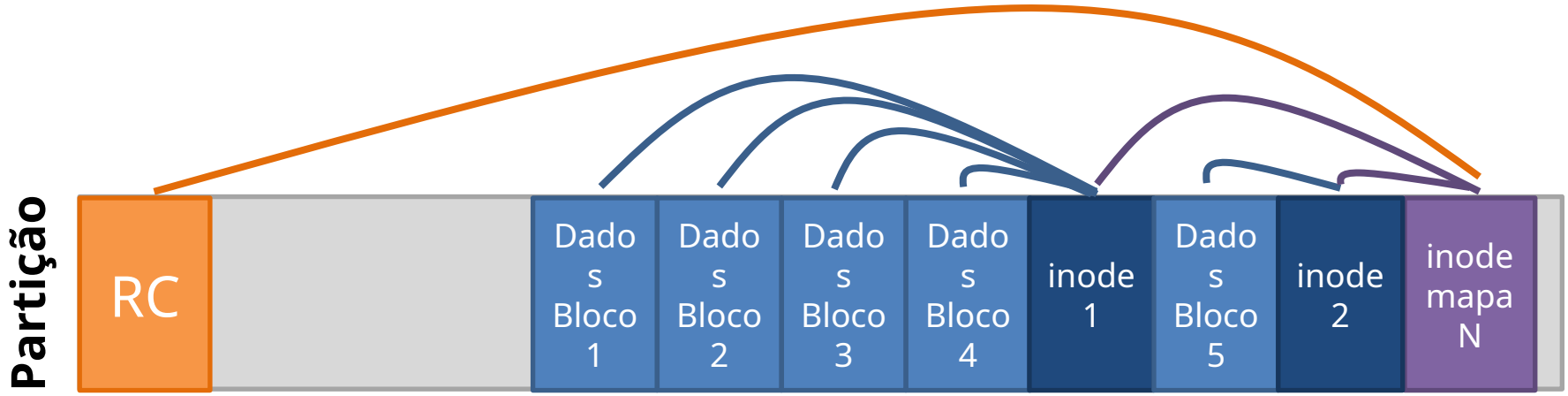
Mapas de I-nodes



- Novo problema: o mapa de i-nodes é espalhado por todo o log
  - Como é que vamos encontrar os dados mais atualizados?

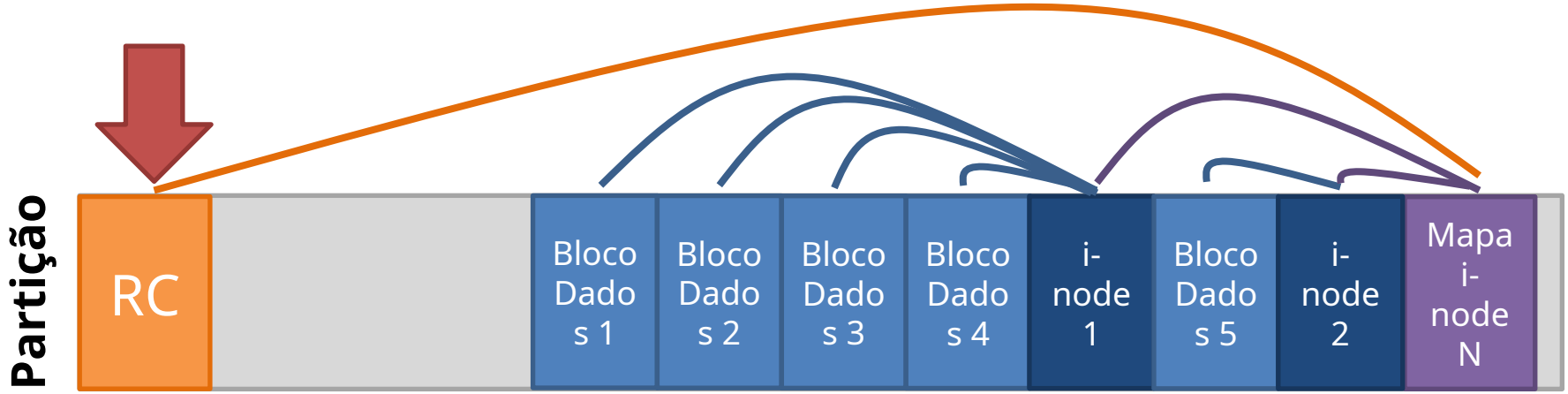
A Região Checkpoint

- O superbloco no LFS contém ponteiros para todos os mapas de i-nodes atualizados ➔ Região Checkpoint
  - LFS = Log File System ou Sistema de Arquivos em Log
  - A região checkpoint é sempre armazenada na memória
  - Salva periodicamente em disco de 30 em 30 segundos
  - É a única parte do LFS que não é mantida no log



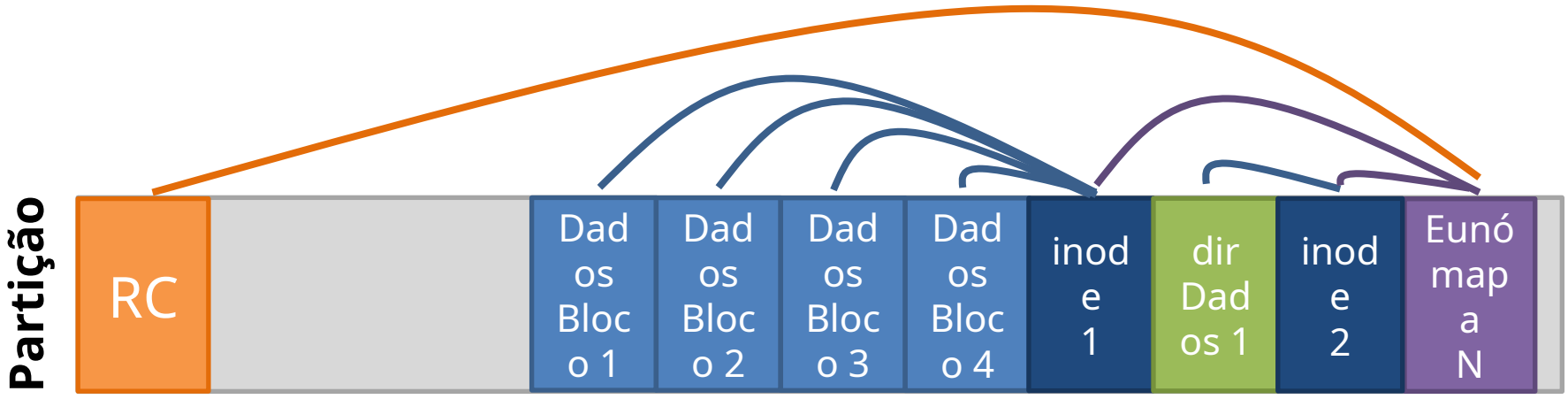
Como ler um arquivo no LFS

- Suponha que você quer ler o i-node 1
  1. Busca i-node 1 na Região de Checkpoint
     - O mapa de i-nodes contend o i-node 1 está no setor *X*
  2. Leia o mapa de i-nodes do setor *X*
     - Lá consta que o i-node 1 está no setor *Y*
  3. Ler i-node 1
     - dados do arquivo estão nos setores *A*, *B*, *C*, Etc.



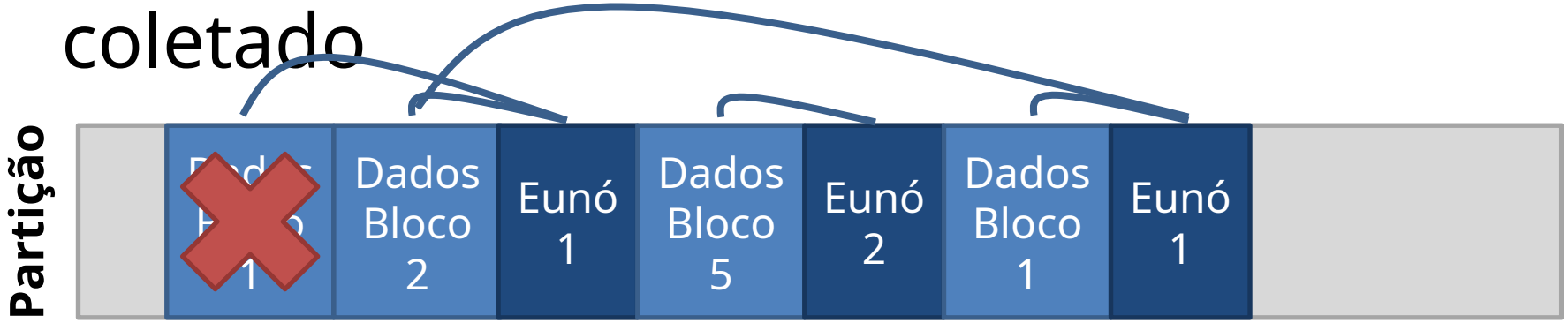
Diretórios em LFS

- Diretórios são armazenados assim como em sistemas de arquivos típicos
  - Dados de diretório armazenados em um arquivo
  - I-node aponta para o arquivo de diretório
  - Arquivo de diretório contém mapeamentos de nomes para i-nodes



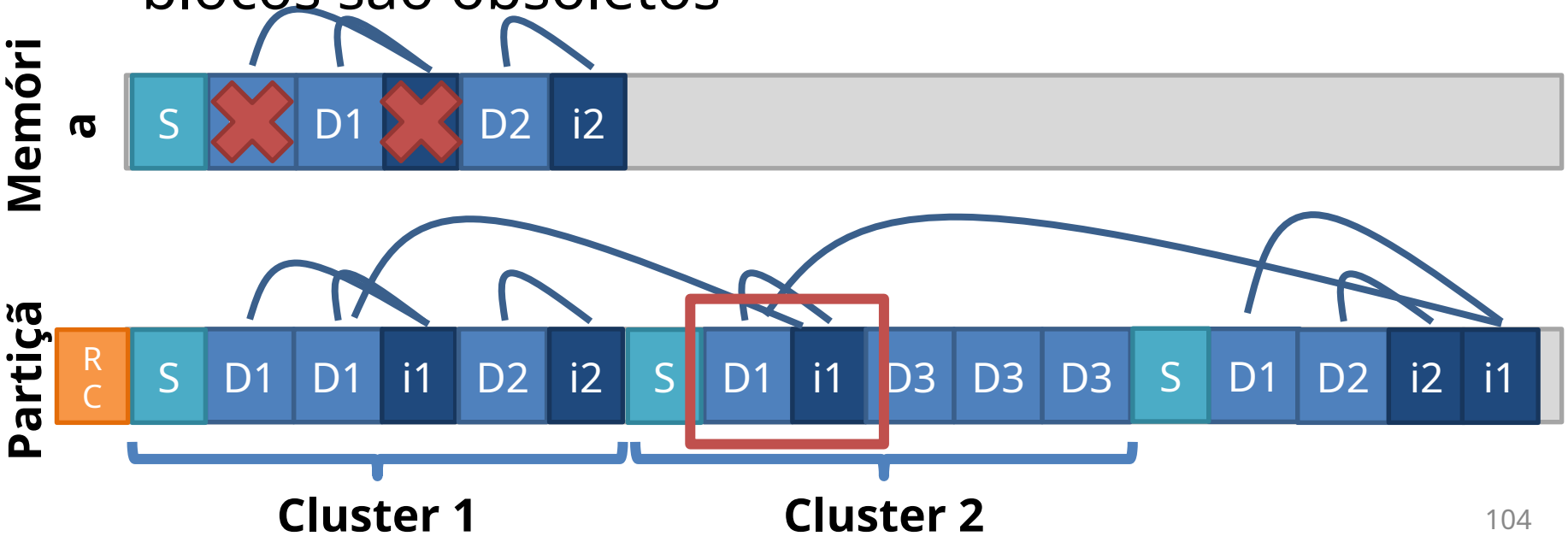
Lixo

- Ao longo do tempo, o journal vai encher-se com dados obsoletos
  - Dados válidos altamente fragmentados misturados com dados obsoletos
- Periodicamente, o log deve ter seu lixo coletado



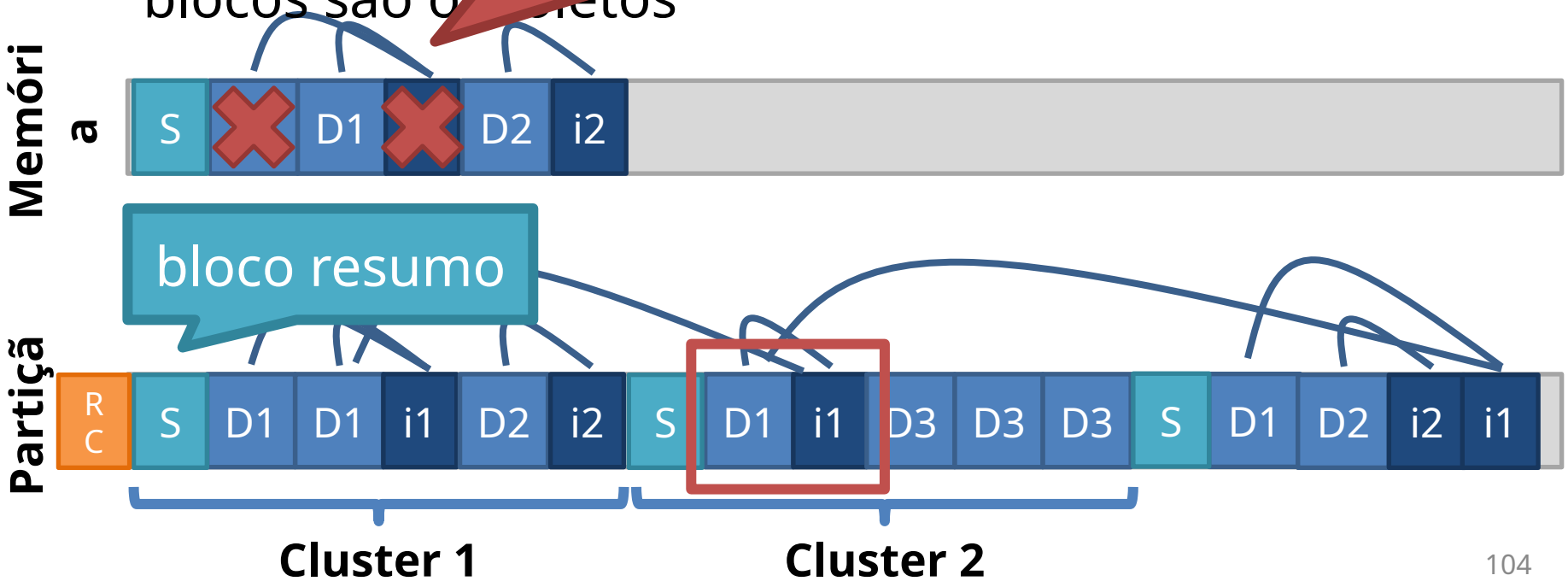
Coleta de Lixo em LFS

- Cada grupo possui um bloco de síntese
  - Contém o mapeamento bloco → inode para cada bloco no cluster
- Checar existência: O GC lê cada arquivo com blocos no cluster
  - Se a informação atual não corresponder ao resumo, blocos são obsoletos



Coleta de Lixo em LFS

- Cada grupo possui um bloco de síntese
  - Contém o mapeamento bloco → inode para cada bloco
- Quais blocos são obsoletos?
- Ponteiros de outros clusters são invisíveis
- ê cada arquivo com blocos
- o corresponder ao resumo, blocos são ob letos



Uma idéia cujo tempo chegou

- LFS parece ser um projeto muito estranho
  - Totalmente diferente de estruturas do sistema de arquivos tradicionais
  - Não mapeia bem às nossas ideias sobre hierarquia de diretórios
- Inicialmente, as pessoas não gostavam LFS
- No entanto, hoje as suas características são amplamente utilizadas

Sistemas de Arquivos para SSDs

- Restrições de hardware SSD
  - Para implementar nivelamento de desgaste as escritas devem ser espalhadas através dos blocos de Flash
  - Periodicamente, blocos antigos precisam ser coletados para prevenir dificuldades com sobrescritas
- Será que isso soa familiar?
- LFS é o sistema de arquivos ideal para SSDs!
- Internamente, SSDs gerenciam todos os arquivos em um LFS
  - Isto é transparente para o sistema operacional e os usuários finais
  - Ideal para nivelamento de desgaste e evitar dificuldades com sobrescritas

Copy-on-write

- Sistemas de arquivos modernos incorporam idéias de LFS
- Semantica Copy-on-write
  - Dados atualizados são escritos para o espaço vazio no disco, em vez de substituir os dados originais
  - Ajuda a prevenir a corrupção de dados, melhora o desempenho de gravação seqüencial
- Lançado pela LFS, agora usado em ZFS e BTRFS
  - BTRFS provavelmente será o próximo sistema

Versionamento sistemas de arquivos

- LFS mantém cópias antigas de dados, por padrão
- Versões antigas de arquivos podem ser uteis!
  - Exemplo: eliminação de arquivo acidental
  - Exemplo: acidentalmente fazendo *open (arquivo, 'w')* em um arquivo cheio de dados
- Transforma uma falha do LFS em uma virtude
- Muitos sistemas de arquivos modernos são Versioned
  - Cópias antigas de dados são expostos para o usuário
  - O usuário pode roll-back um arquivo para recuperar versões antigas