

ACH2024

Organização de arquivos,
Alocação sequencial (cont),
ligada e indexada

Prof Helton Hideraldo Bís caro

Aula passada

- Alocação sequencial
 - registros não ordenados:
 - registros ordenados (por algum campo de interesse - CHAVE)

Aula passada

- Alocação sequencial, **registros não ordenados**:
 - Inserção: $O(1)$ ou $O(b)$
 - Busca: $O(b)$
 - Remoção: assumindo que já achou o registro, $O(b)$ se remover de verdade ou $O(1)$ de usar bit de validade (mas daí exige reorganizações periódicas a custo $O(b)$ cada)
 - Modificação de qualquer campo: $O(1)$ se registros de tamanho fixo
 - Leitura ordenada:

Aula passada

- Alocação sequencial, **registros não ordenados**:
 - Inserção: $O(1)$ ou $O(b)$
 - Busca: $O(b)$
 - Remoção: assumindo que já achou o registro, $O(b)$ se remover de verdade ou $O(1)$ de usar bit de validade (mas daí exige reorganizações periódicas a custo $O(b)$ cada)
 - Modificação de qualquer campo: $O(1)$ se registros de tamanho fixo
 - Leitura ordenada:
teria que ordenar primeiro!



Aula de hoje

Alocação sequencial (ordenado)

Os r registros
estão
ordenados por
um campo
específico - a
chave

	NAME	SSN	BIRTHDATE	JOB	SALARY	SEX
block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
block 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
block 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					
block 5	Anderson, Zach					
	Angeli, Joe					
	⋮					
	Archer, Sue					
block 6	Arnold, Mack					
	Arnold, Steven					
	⋮					
	Atkins, Timothy					
⋮						
block n-1	Wong, James					
	Wood, Donald					
	⋮					
	Woods, Manny					
block n	Wright, Pam					
	Wyatt, Charles					
	⋮					
	Zimmer, Byron					

Alocação sequencial (ordenado)

- **Leitura ordenada**
- **Mínimo / Máximo**
- **Busca:**

Alocação sequencial (ordenado)

- **Leitura ordenada eficiente** (sequencial) – $O(b)$
 - O próximo registro pode estar no mesmo bloco
- **Mínimo / Máximo**
- **Busca:**

Alocação sequencial (ordenado)

- **Leitura ordenada eficiente** (sequencial) – $O(b)$
 - O próximo registro pode estar no mesmo bloco
- **Mínimo / Máximo** estão no cabeçalho do arquivo – $O(1)$
 - Isto podemos fazer para todos os tipos de alocação
- **Busca:**

Alocação sequencial (ordenado)

- **Leitura ordenada eficiente** (sequencial) – $O(b)$
 - O próximo registro pode estar no mesmo bloco
- **Mínimo / Máximo** estão no cabeçalho do arquivo – $O(1)$
 - Isto podemos fazer para todos os tipos de alocação
- **Busca:** dá para usar busca binária (baseada nos blocos!)

Alocação sequencial (ordenado)

- **Leitura ordenada eficiente** (sequencial) – $O(b)$
 - O próximo registro pode estar no mesmo bloco
- **Mínimo / Máximo** estão no cabeçalho do arquivo – $O(1)$
 - Isto podemos fazer para todos os tipos de alocação
- **Busca:** dá para usar busca binária (baseada nos blocos!)

Como é mesmo o algoritmo de busca binária (em memória)?

Alocação sequencial (ordenado)

/* busca registro com chave **k** em um arquivo com **b** blocos */

buscaBinaria (k) {

}

Alocação sequencial (ordenado)

/* busca registro com chave **k** em um arquivo com **b** blocos */

buscaBinaria (k) {

$e \leftarrow \text{primeiro_bloco}$; $d \leftarrow \text{primeiro_bloco} + b$;

 enquanto ($d \geq e$){

$m \leftarrow \text{floor}((e+d)/2)$;

 lê bloco m do disco para o buffer

 se ($k < \text{chave do primeiro registro do bloco m}$) $d \leftarrow m - 1$;

 senão se ($k > \text{chave do último registro do bloco m}$) $e \leftarrow m+1$;

 senão se ($k = \text{chave de algum registro do bloco m}$) retorna TRUE

 senão retorna FALSE

 }

}

Alocação sequencial (ordenado)

/* busca registro com chave **k** em um arquivo com **b** blocos */

buscaBinaria (k) {

$e \leftarrow \text{primeiro_bloco}$; $d \leftarrow \text{primeiro_bloco} + b$;

 enquanto ($d \geq e$){

$m \leftarrow \text{floor}((e+d)/2)$;

 lê bloco m do disco para o buffer

 se ($k < \text{chave do primeiro registro do bloco m}$) $d \leftarrow m - 1$;

 senão se ($k > \text{chave do último registro do bloco m}$) $e \leftarrow m+1$;

 senão se ($k = \text{chave de algum registro do bloco m}$) retorna TRUE

 senão retorna FALSE

 }

}

Complexidade: ?

Alocação sequencial (ordenado)

/* busca registro com chave **k** em um arquivo com **b** blocos */

buscaBinaria (k) {

$e \leftarrow \text{primeiro_bloco}$; $d \leftarrow \text{primeiro_bloco} + b$;

 enquanto ($d \geq e$){

$m \leftarrow \text{floor}((e+d)/2)$;

 lê bloco m do disco para o buffer

 se ($k < \text{chave do primeiro registro do bloco m}$) $d \leftarrow m - 1$;

 senão se ($k > \text{chave do último registro do bloco m}$) $e \leftarrow m+1$;

 senão se ($k = \text{chave de algum registro do bloco m}$) retorna TRUE

 senão retorna FALSE

 }

}

Complexidade: $O(\lg b)$

Alocação sequencial (ordenado)

- **Inserção:**

Alocação sequencial (ordenado)

- **Inserção:** cara! - $O(?)$
 - Tem que achar a posição certa : $O(?)$
 - Tem que abrir espaço para o registro (deslocar todos os registros com chave maior para frente) : $O(?)$
 - Trazer o respectivo bloco para o buffer
 - Deslocar os registros (o último registro vai para o próximo bloco, a não ser no caso de ser o último bloco e haver espaço para mais um registro)
 - Salvar o bloco no disco

Alocação sequencial ordenado

- **Inserção:** cara! - $O(b)$
 - Tem que achar a posição certa : $O(\lg b)$
 - Tem que abrir espaço para o registro (deslocar todos os registros com chave maior para frente) : $O(b)$
 - Trazer o respectivo bloco para o buffer
 - Deslocar os registros (o último registro vai para o próximo bloco, a não ser no caso de ser o último bloco e haver espaço para mais um registro)
 - Salvar o bloco no disco

Alternativa para melhorar desempenho: deixar espaços vazios nos blocos (demanda reorganização periódica e mais complicado para calcular localização dos registros)

Alocação sequencial ordenado

- **Exclusão:**

Alocação sequencial ordenado

- **Exclusão:** cara pelos mesmos motivos! - $O(b)$
 - Custo diminuído se usar bit válido/inválido para cada registro e efetuar reorganizações periódicas

Alocação sequencial ordenado

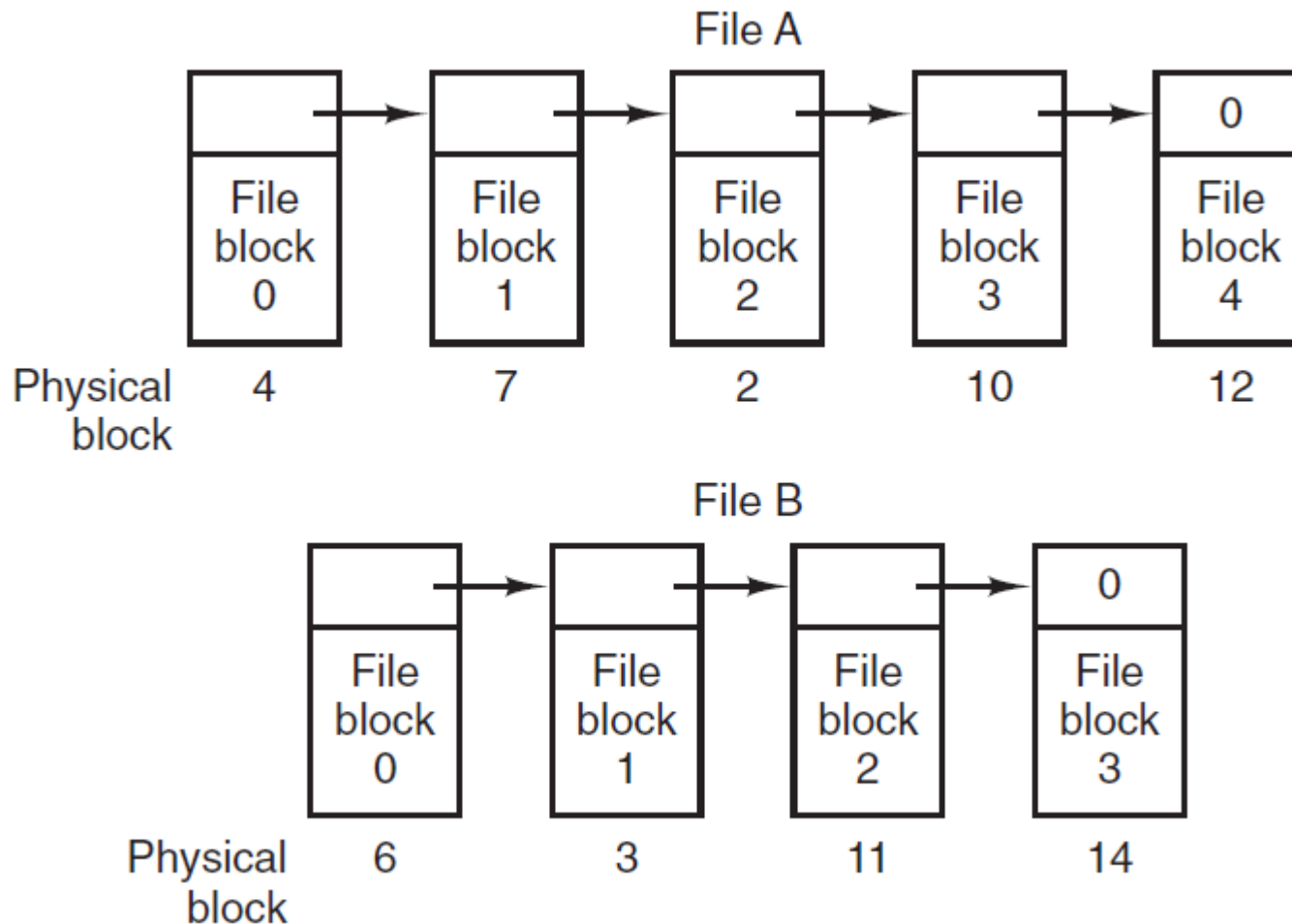
- **Modificação:**

Alocação sequencial ordenado

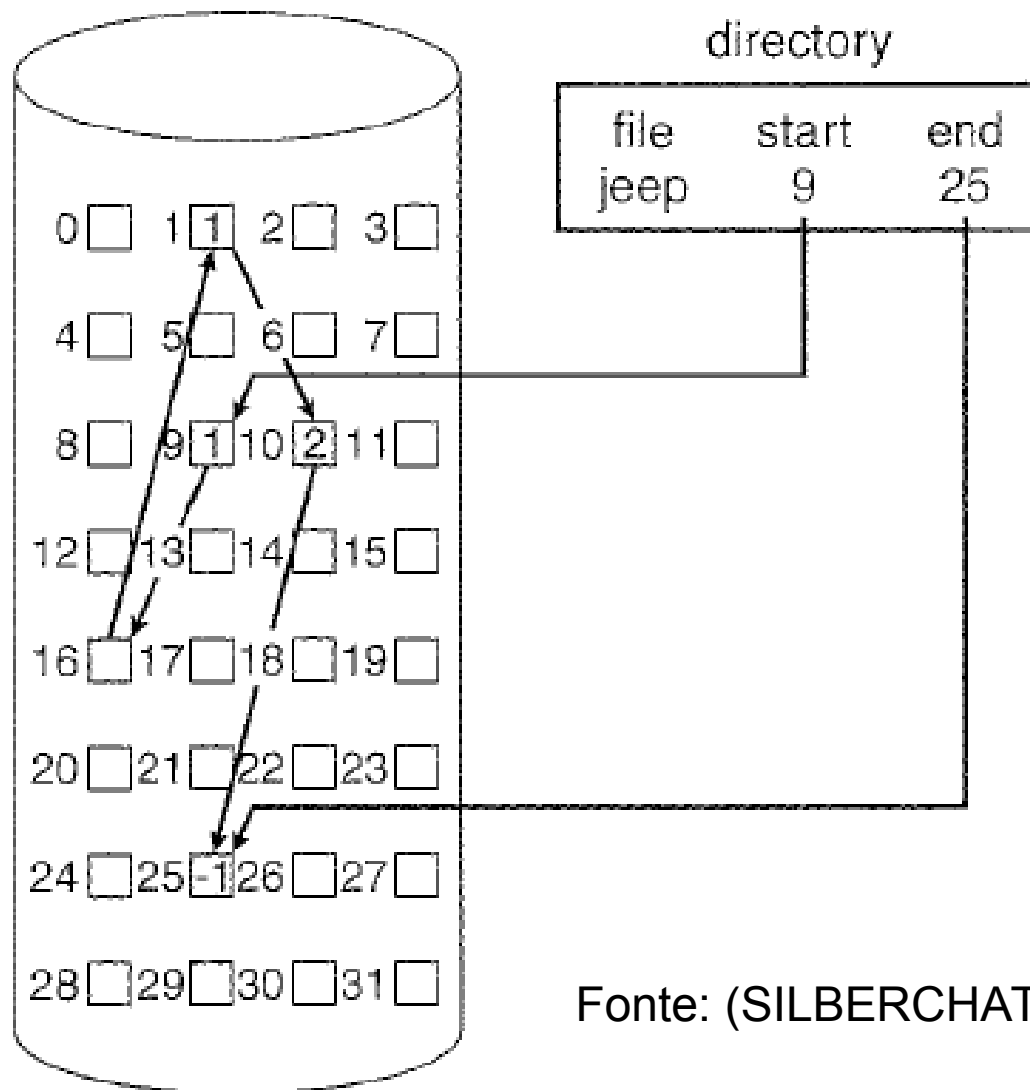
- **Modificação:** busca + atualização
 - Caro se a atualização alterar o tamanho do registro (no caso de tamanho variável)
 - Pior se for na chave: uma remoção e uma inserção

Alocação por listas ligadas

Cada arquivo é uma lista ligada de blocos



Alocação por listas ligadas



Fonte: (SILBERCHATZ et al, 2009)

Alocação por listas ligadas

- **Uso de espaço:**
- **Leitura sequencial:**
- **Leitura aleatória / Busca:**
- **Inserção:**
- **Remoção:**
- **Modificação:**

Alocação por listas ligadas

- **Uso de espaço:**
 - **Vantagem:**
 - **Desvantagem:**
- **Leitura sequencial:**
- **Leitura aleatória / Busca:**
- **Inserção:**
- **Remoção:**
- **Modificação:**

Alocação por listas ligadas

- **Uso de espaço:**
 - **Vantagem:** resolve o problema de fragmentação externa
 - **Desvantagem:** perda de espaço com os ponteiros
- **Leitura sequencial:**
- **Leitura aleatória / Busca:**
- **Inserção:**
- **Remoção:**
- **Modificação:**

Alocação por listas ligadas

- **Uso de espaço:**
 - **Vantagem:** resolve o problema de fragmentação externa
 - **Desvantagem:** perda de espaço com os ponteiros
- **Leitura sequencial:** $O(b)$
- **Leitura aleatória / Busca:**
- **Inserção:**
- **Remoção:**
- **Modificação:**

Alocação por listas ligadas

- **Uso de espaço:**
 - **Vantagem:** resolve o problema de fragmentação externa
 - **Desvantagem:** perda de espaço com os ponteiros
- **Leitura sequencial:** $O(b)$
- **Leitura aleatória / Busca:** $O(b)$
- **Inserção:**
- **Remoção:**
- **Modificação:**

Alocação por listas ligadas

- **Uso de espaço:**
 - **Vantagem:** resolve o problema de fragmentação externa
 - **Desvantagem:** perda de espaço com os ponteiros
- **Leitura sequencial:** $O(b)$
- **Leitura aleatória / Busca:** $O(b)$
- **Inserção:** $O(1)$ – assumindo que sei onde inserir
- **Remoção:**
- **Modificação:**


Alocação por listas ligadas

- **Uso de espaço:**
 - **Vantagem:** resolve o problema de fragmentação externa
 - **Desvantagem:** perda de espaço com os ponteiros
- **Leitura sequencial:** $O(b)$
- **Leitura aleatória / Busca:** $O(b)$
- **Inserção:** $O(1)$ – assumindo que sei onde inserir
- **Remoção:** $O(1)$ – assumindo que sei onde
- **Modificação:**

Alocação por listas ligadas

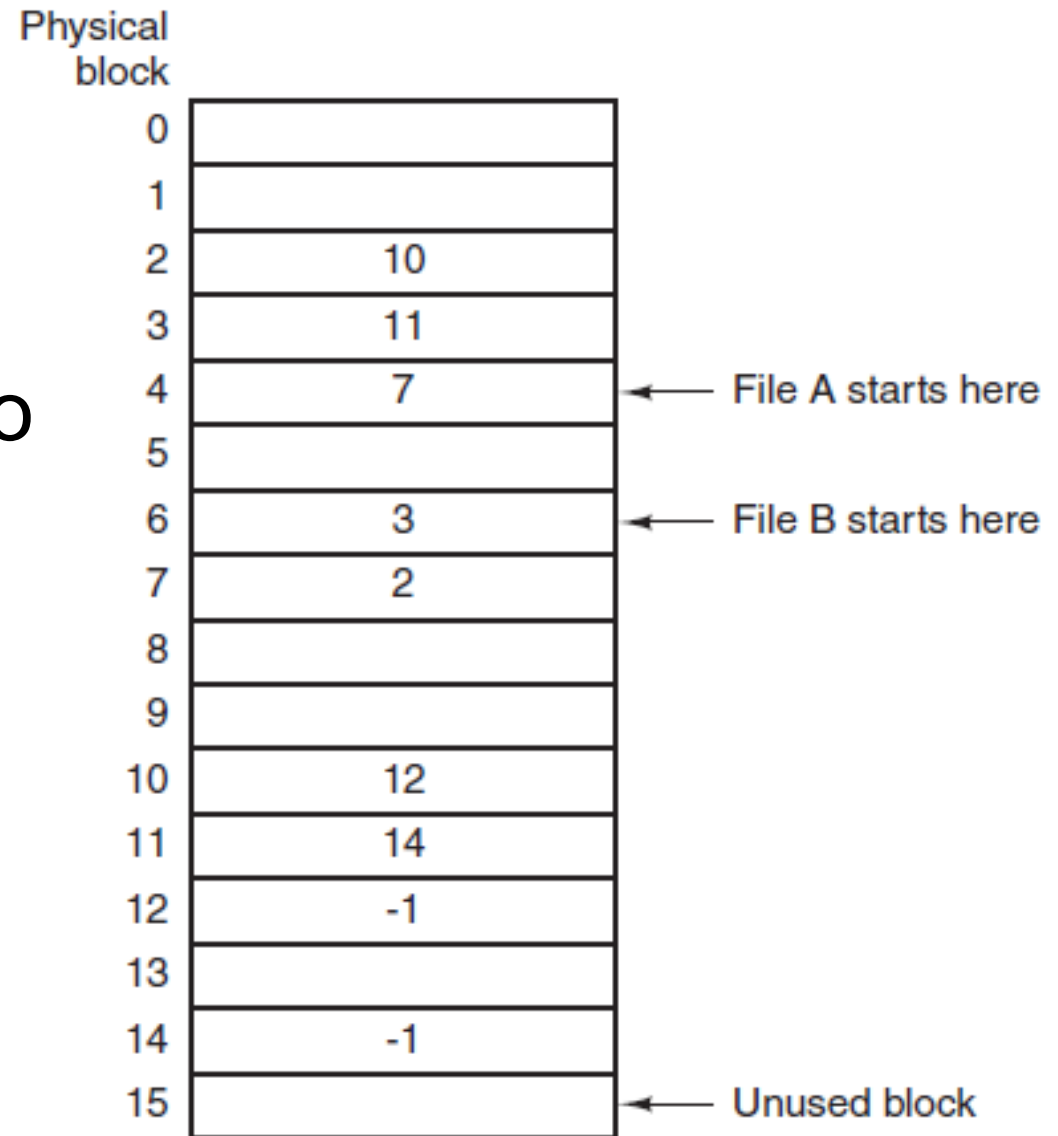
- **Uso de espaço:**
 - **Vantagem:** resolve o problema de fragmentação externa
 - **Desvantagem:** perda de espaço com os ponteiros
- **Leitura sequencial:** $O(b)$
- **Leitura aleatória / Busca:** $O(b)$
- **Inserção:** $O(1)$ – assimindo que sei onde inserir
- **Remoção:** $O(1)$ – assimindo que sei onde inserir
- **Modificação:** $O(1)$

Alocação por listas ligadas

- **Uso de espaço:**
 - **Vantagem:** resolve o problema de fragmentação externa
 - **Desvantagem:** perda de espaço com os ponteiros
- **Leitura sequencial:** $O(b)$
- **Leitura aleatória / Busca:** $O(b)$ 
- **Inserção:** $O(1)$ – assimindo que sei onde inserir
- **Remoção:** $O(1)$ – assimindo que sei onde inserir
- **Modificação:** $O(1)$

Alocação por listas ligadas com uso de uma File Allocation Table (FAT) em memória

- $t[i]$ armazena o próximo bloco do bloco i



Fonte: (TANEMBAUM, 2015)

FAT

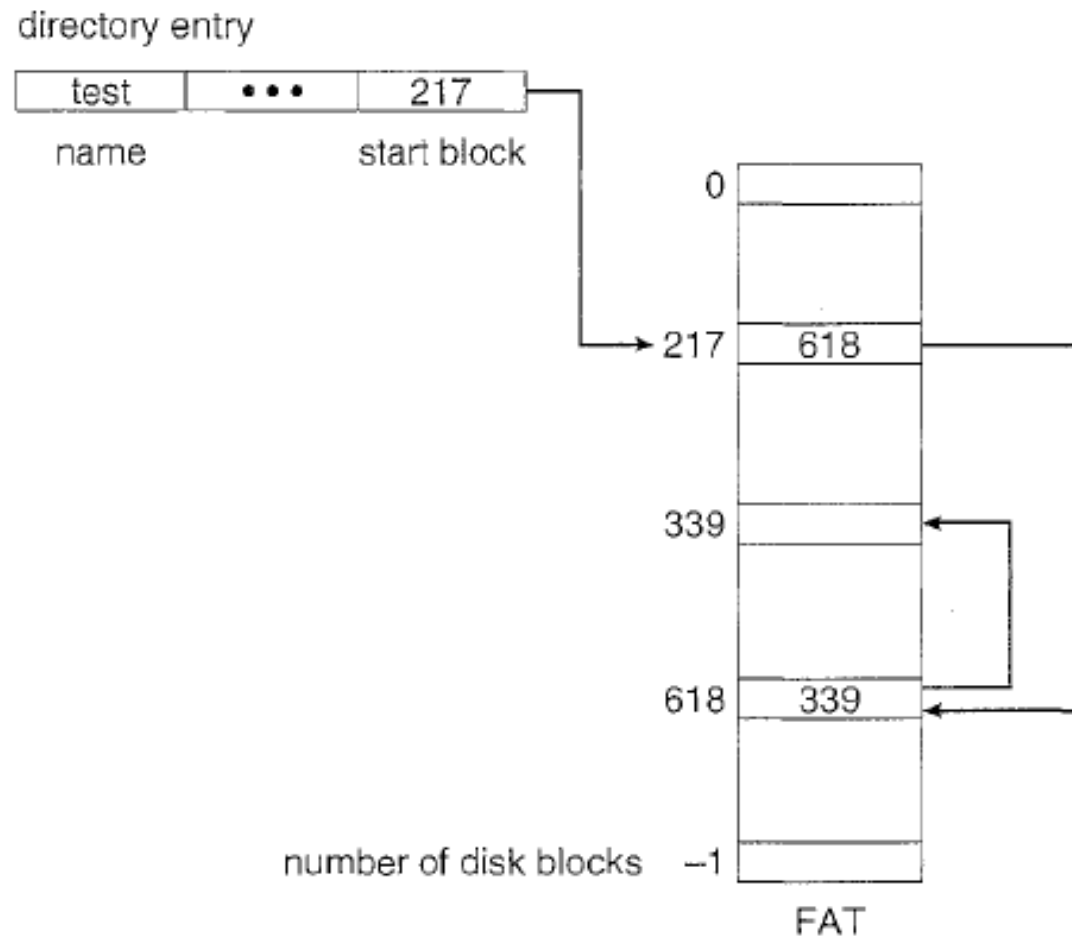
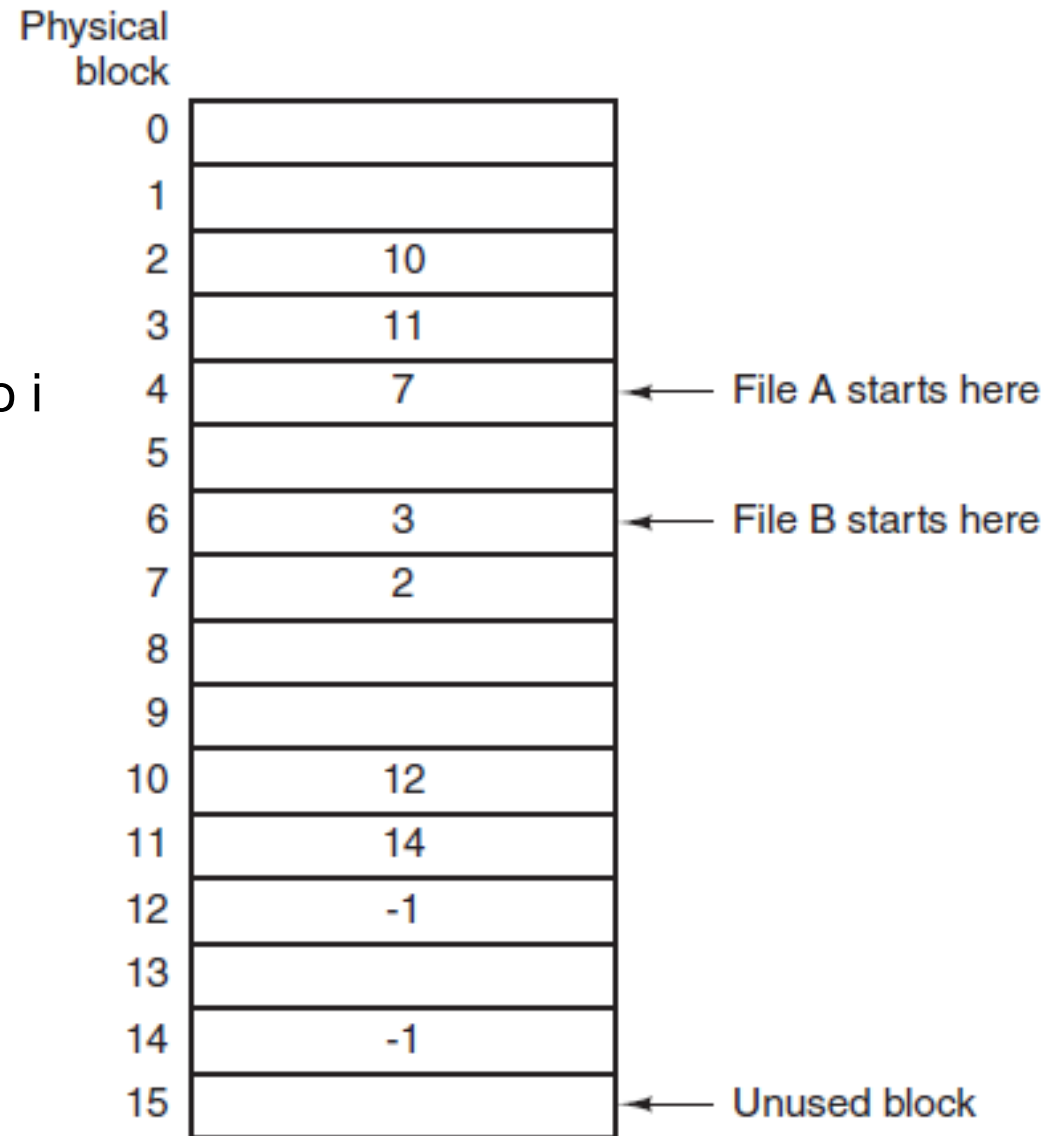


Figure 11.7 File-allocation table.

Alocação por listas ligadas com uso de uma File Allocation Table (FAT) em memória

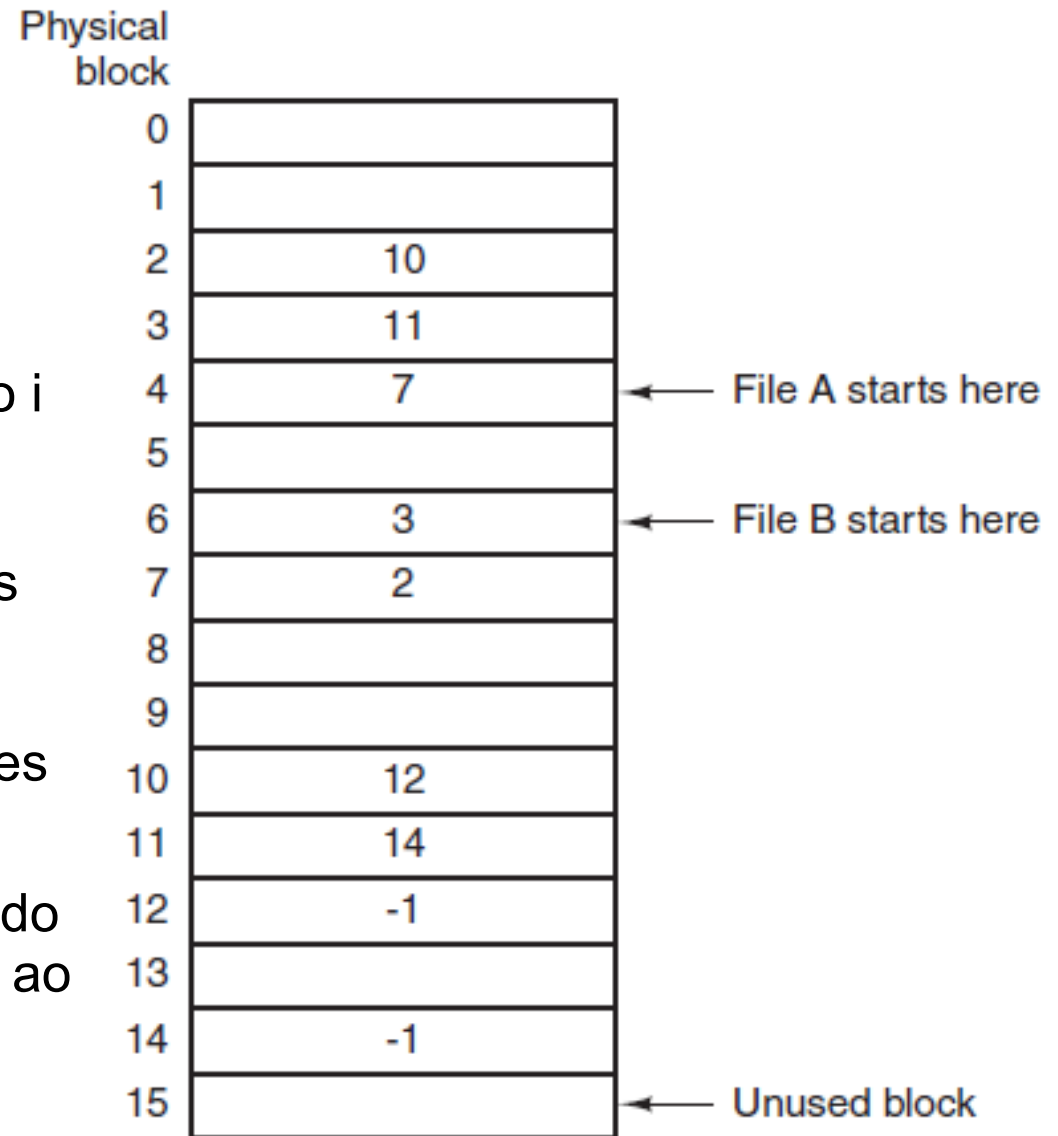
- $t[i]$ armazena o próximo bloco do bloco i
- Vantagens:



Fonte: (TANEMBAUM, 2015)

Alocação por listas ligadas com uso de uma File Allocation Table (FAT) em memória

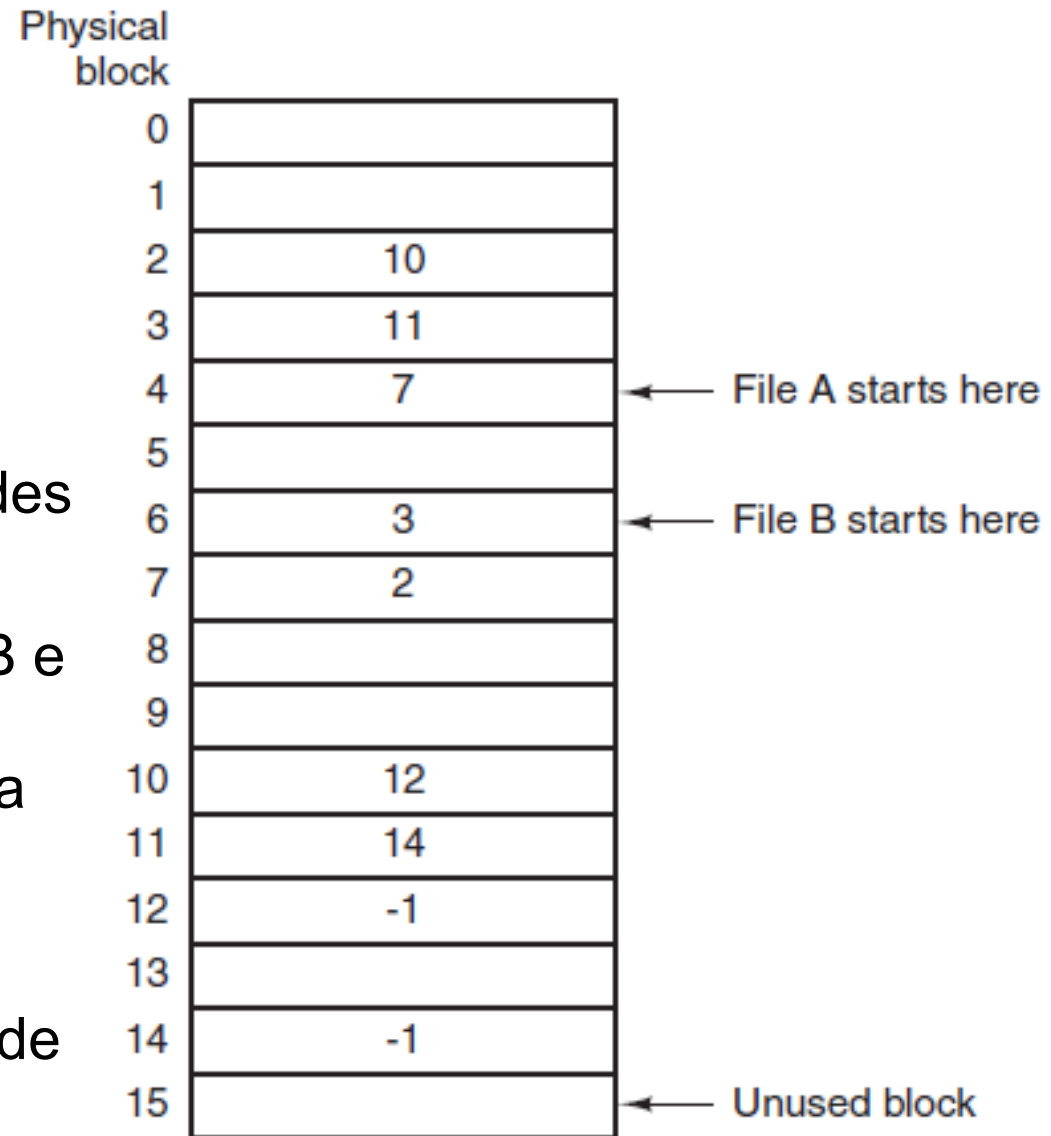
- $t[i]$ armazena o próximo bloco do bloco i
- Vantagens:
 - Economiza espaço nos blocos de dados (que só conterão dados e não ponteiros) \rightarrow b menor impacta nas velocidades dependentes de b
 - Para uma leitura aleatória (dado um deslocamento em relação ao início do arquivo), o encadeamento (para achar o bloco certo) é seguido apenas sobre a tabela (que está toda em memória) $\rightarrow O(1)$



Fonte: (TANEMBAUM, 2015)

Alocação por listas ligadas com uso de uma File Allocation Table (FAT) em memória

- Desvantagem:
 - Não escalável para grandes discos
 - Ex: para um disco de 1TB e blocos de 1KB, a tabela ocuparia 3GB de memória
 - Sistema de arquivos FAT32, por ex, impõe:
 - Tamanho máximo de arquivo: 4GB
 - Tamanho máximo de disco de 2TB



Fonte: (TANEMBAUM, 2015)

Outra alternativa?

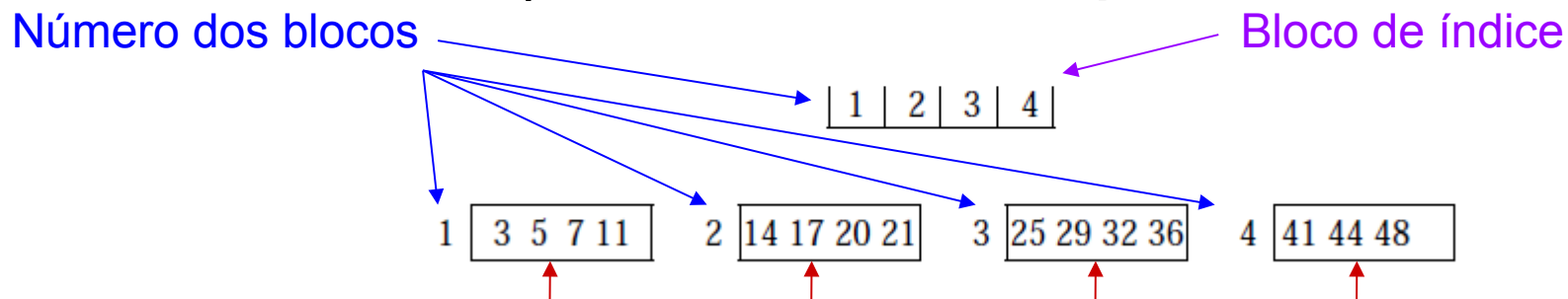
- Lista ligada aproveita espaço (resolve fragmentação externa), mas leitura aleatória fica horrível (sem tabela de alocação)
- Tabela de alocação acelera a leitura aleatória mas gasta muita memória
- Como diminuir esse último problema?

Outra alternativa?

- Lista ligada aproveita espaço (resolve fragmentação externa), mas leitura aleatória fica horrível (sem tabela de alocação)
- Tabela de alocação acelera a leitura aleatória mas gasta muita memória
- Como diminuir esse último problema?
- Por que manter em memória as informações de arquivos que não foram abertos?

Alocação indexada

- Um ou mais blocos de índices contém ponteiros para os blocos de fato
- Blocos de índices são como uma tabela de alocação específica daquele arquivo
 - i -ésima entrada do primeiro bloco de índice contém o número do i -ésimo bloco de dado do arquivo
- Blocos de índice carregados na memória sob demanda (assim como arquivos de dados)



Alocação indexada

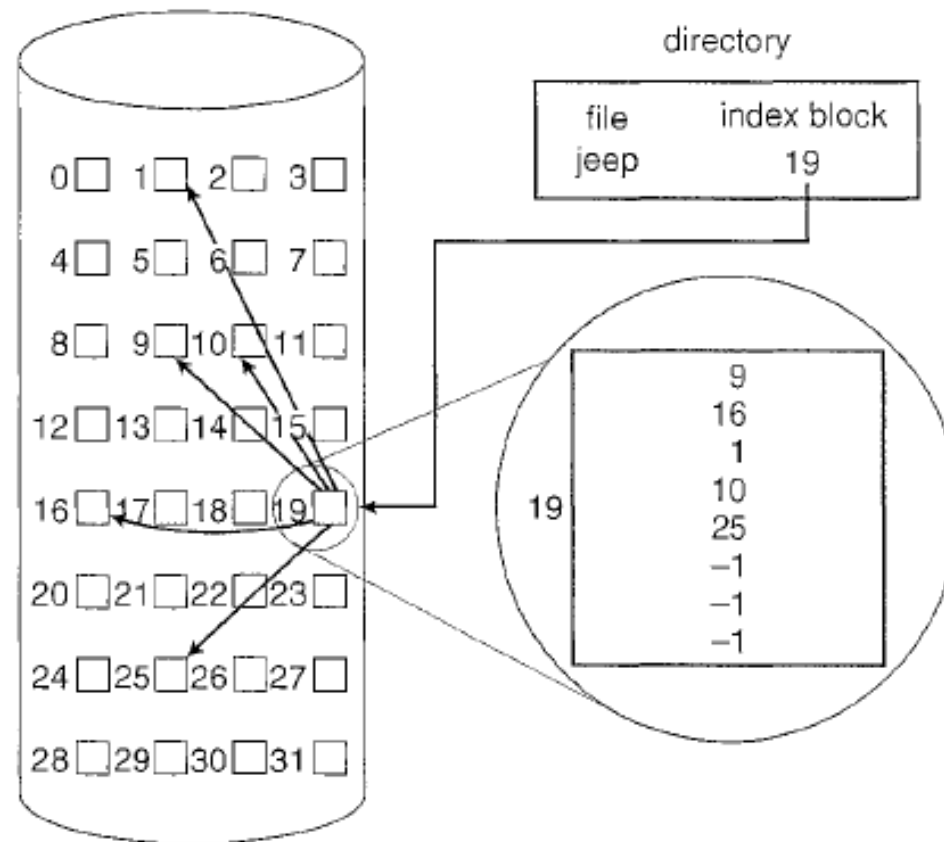


Figure 11.8 Indexed allocation of disk space.

Cabeçalhos de arquivo do tipo I-nodes (index-nodes): visão geral

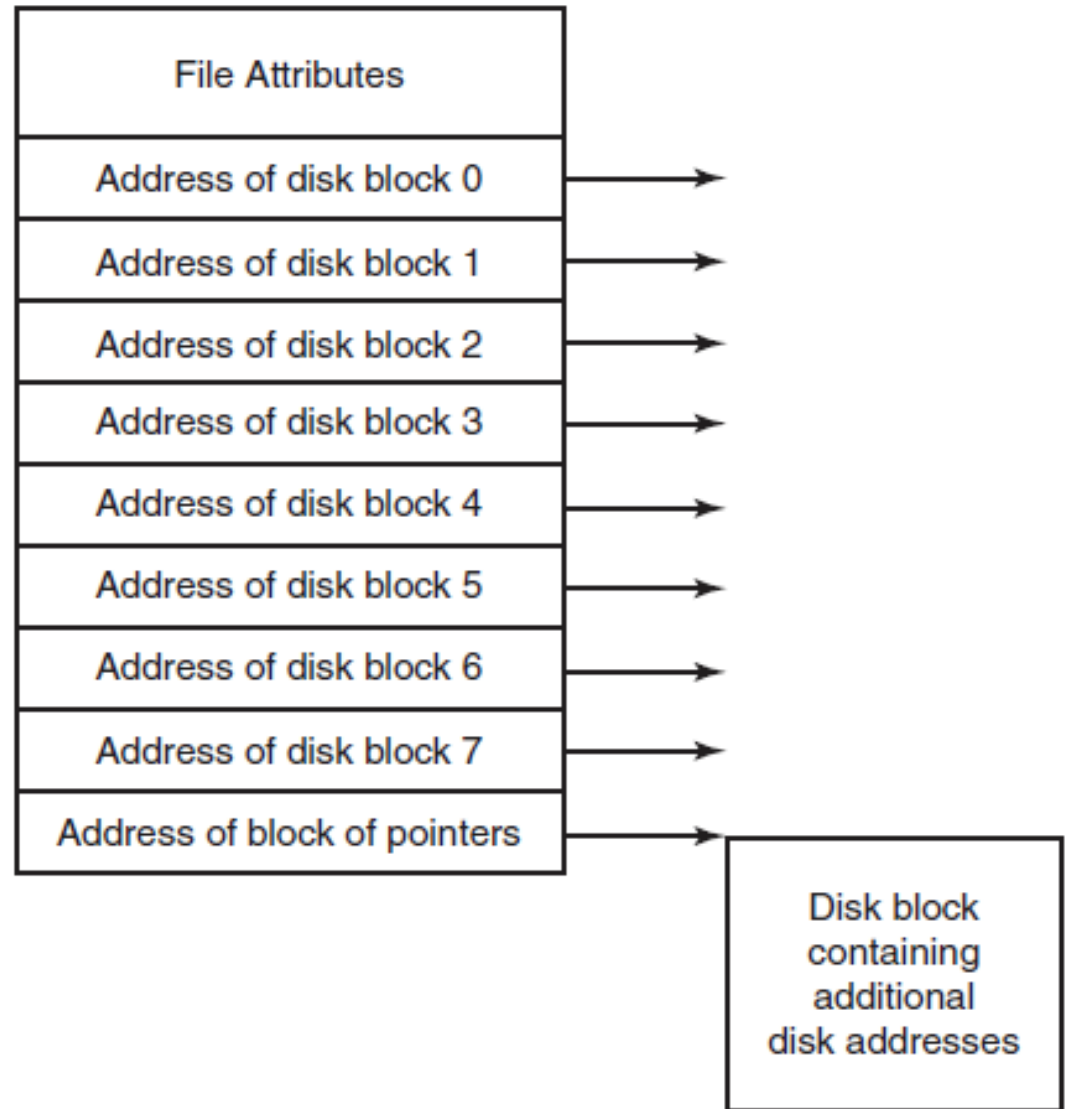
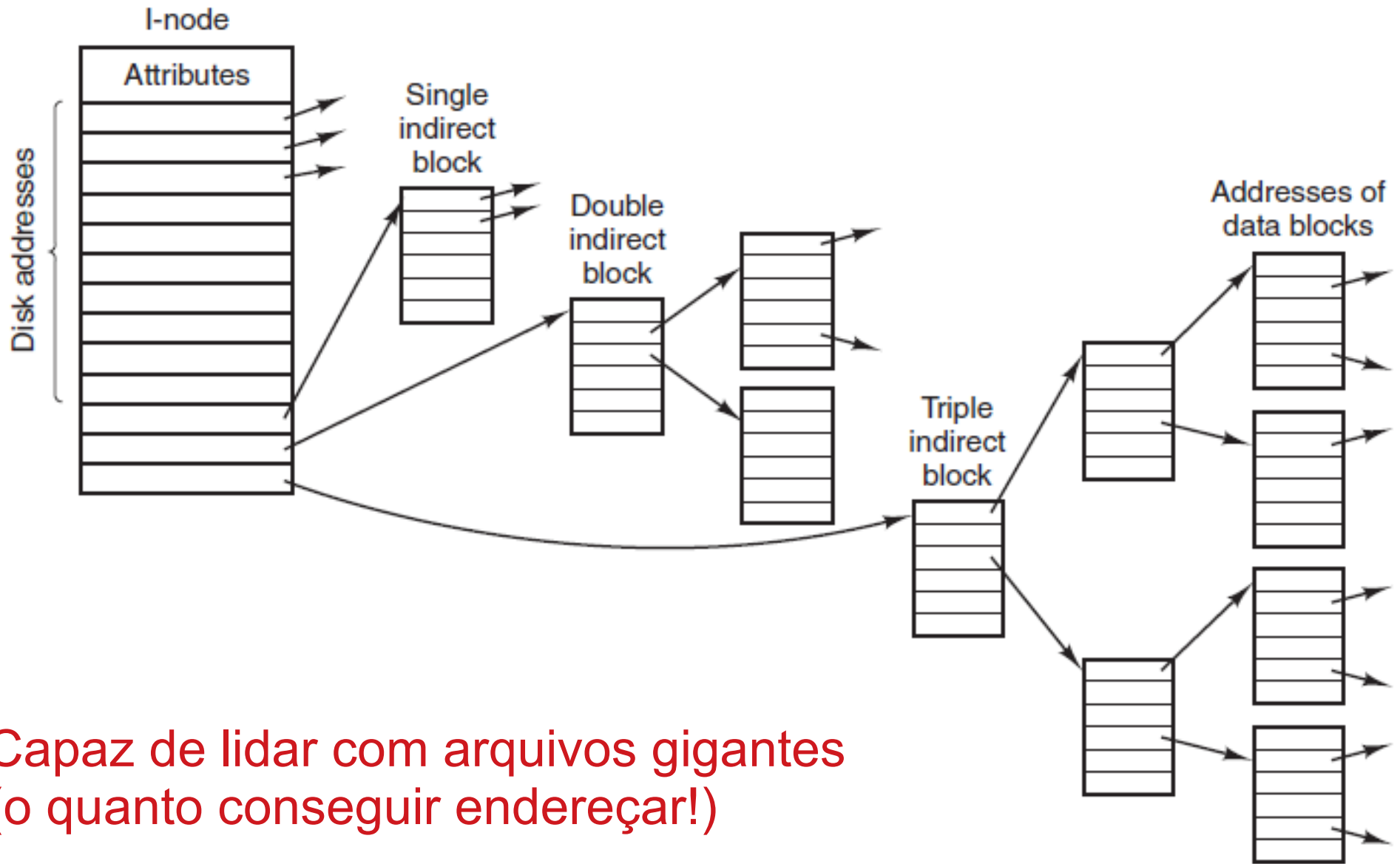


Figure 4-13. An example i-node.

Fonte: (TANEMBAUM, 2015)

I-nodes (index-nodes) do UNIX/LINUX:



Capaz de lidar com arquivos gigantes
(o quanto conseguir endereçar!)

Alocação indexada

- Resolve fragmentação externa
- Leitura aleatória (acesso direto) eficiente
- Gasto de espaço com ponteiros (blocos de índice) maior que na lista ligada (principalmente para arquivos pequenos)

Leitura complementar

- Mais detalhes sobre o sistema de arquivos do Linux e Windows: cap 4, 10 e 11 do livro do Tanenbaum (referência no último slide)

Referências

- Slides da Profa. Graça (ICMC) - [http://wiki.icmc.usp.br/index.php/SCC-203_\(gracan\)](http://wiki.icmc.usp.br/index.php/SCC-203_(gracan)) (Arquivos 8, 9 e 12)
- Slides do cap 6 do Ziviani
- ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 4 ed. Ed. Pearson-Addison Wesley. Cap 13 (até a seção 13.7).
- GOODRICH et al, **Data Structures and Algorithms in C++**. Ed. John Wiley & Sons, Inc. 2nd ed. 2011. Seção 14.2
- RAMAKRISHNAN, R.; GEHRKE, J. **Data Management Systems** 3ed. Ed McGraw Hill. 2003. cap 8 e 9.
- SILBERSCHATZ, A.; GALVIN, p. B.; GAGNE, G. **Operating Systems Concepts**. 8 ed. Ed. John Wiley & Sons. 2009. Cap 11
- TANEMBAUM, A. S. & BOS, H. **Modern Operating Systems**. Pearson, 4th ed. 2015. Cap 4