ACH2024

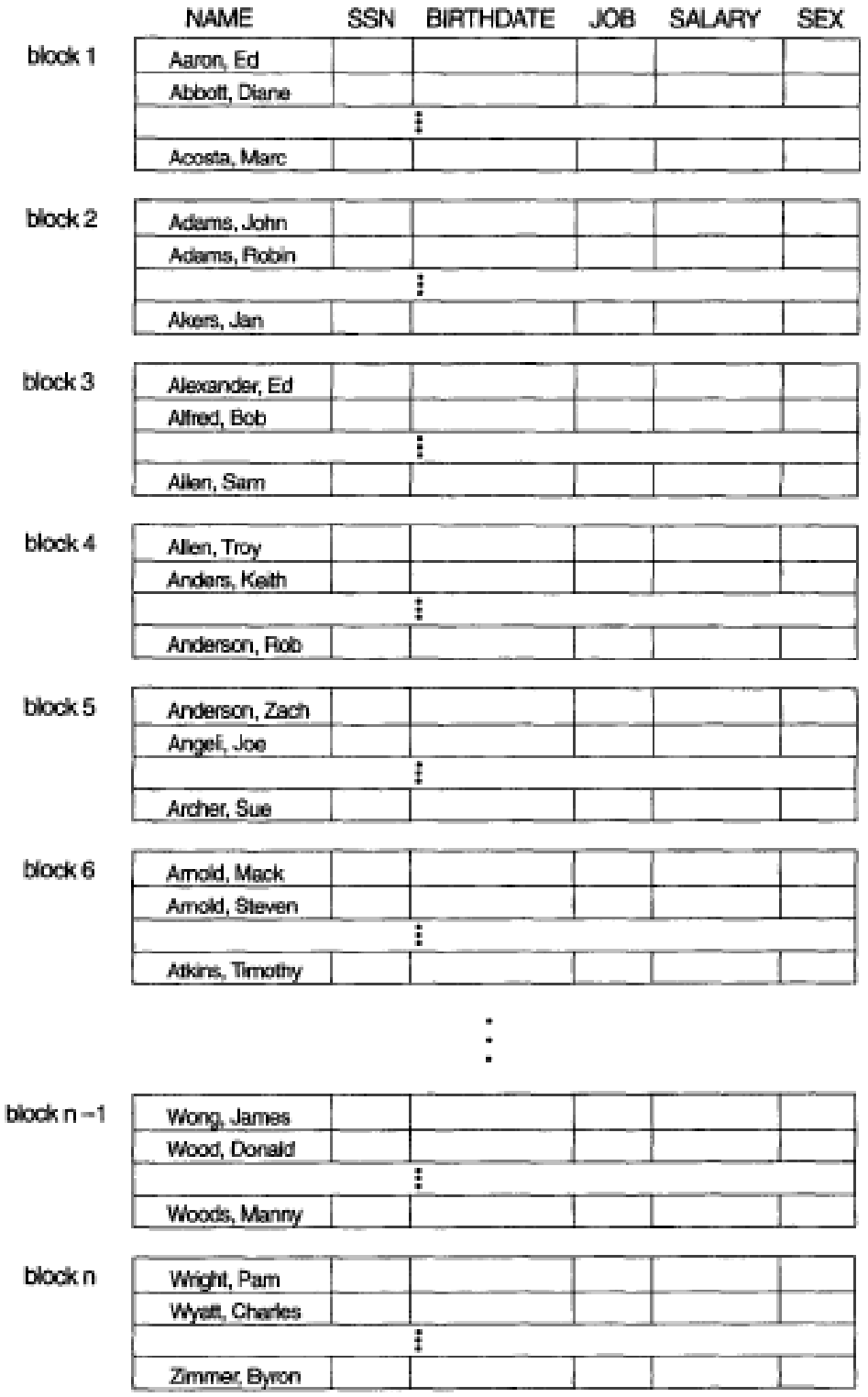
Índices (Acesso indexado)

Prof. Helton Hideraldo Bíscaro

Aula passada

Alocação sequencial (ordenado)

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

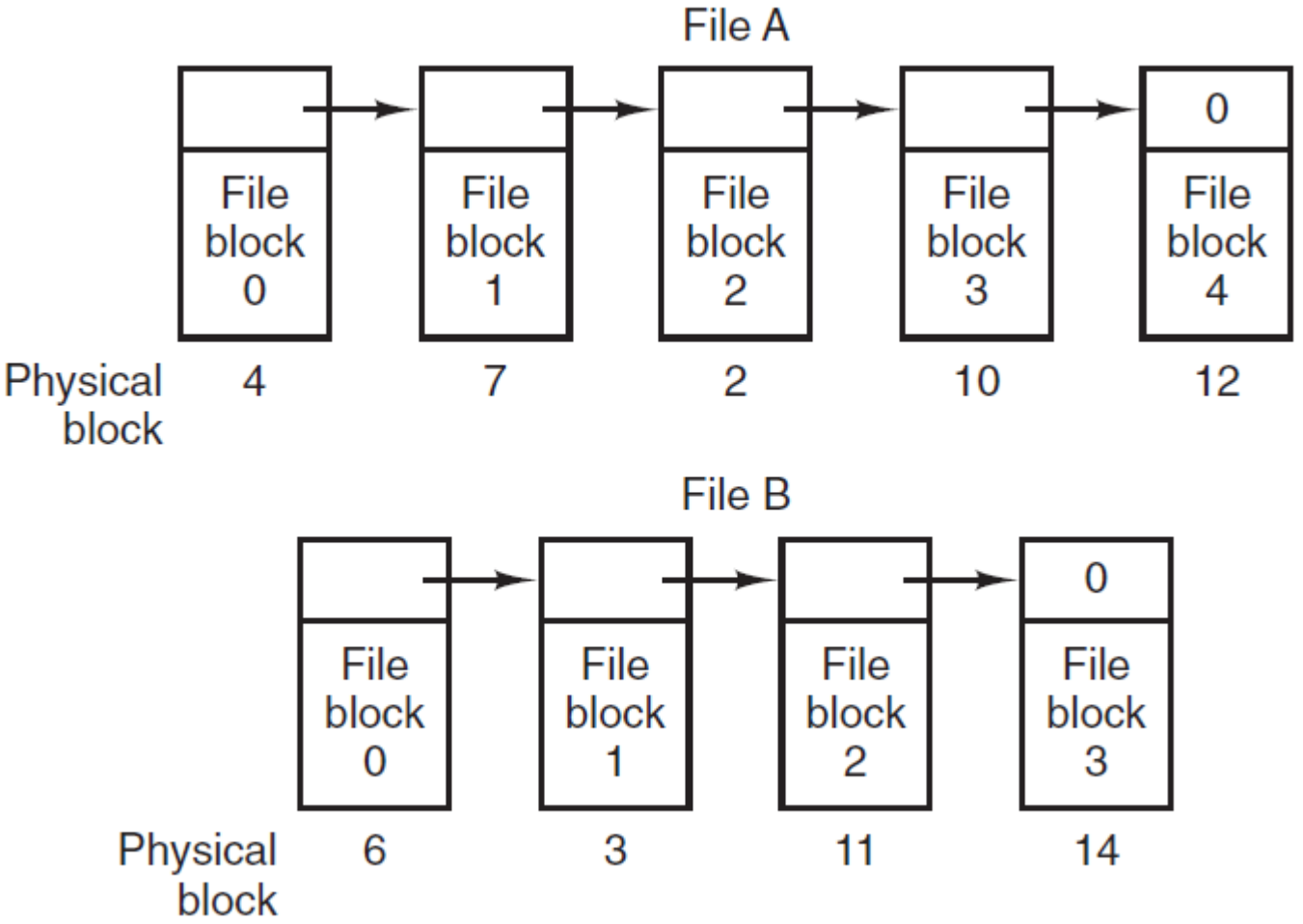


Alocação sequencial (ordenado)

- **Leitura ordenada eficiente** (sequencial) – O(b)
  - O próximo registro pode estar no mesmo bloco
- **Inserção / Remoção** – O(b)
- **Busca**: dá para usar busca binária (baseada nos blocos!) - O(lg b)

Alocação por listas ligadas

Cada arquivo é uma lista ligada de blocos



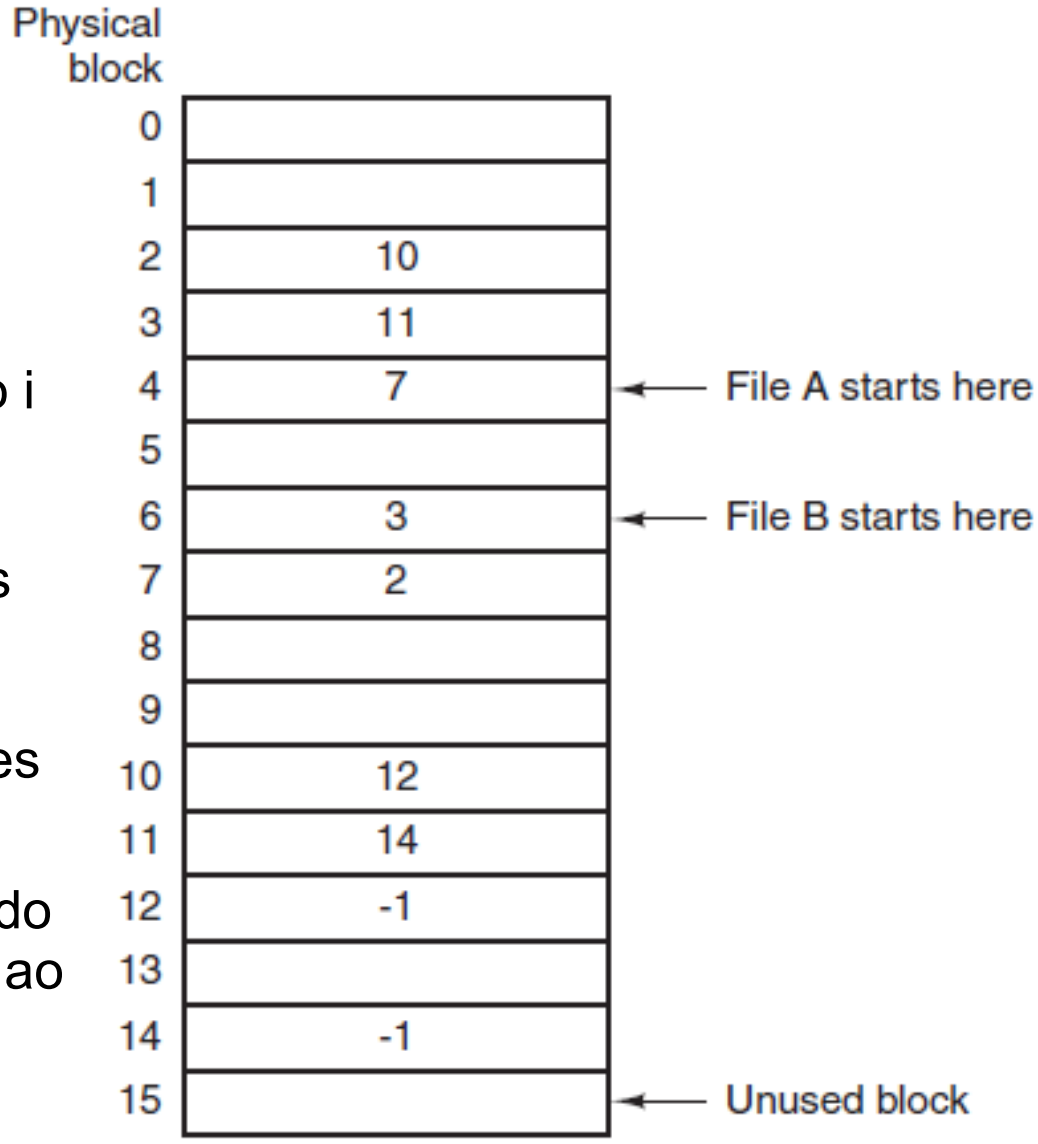
Fonte: (TANEMBAUM, 2015)

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

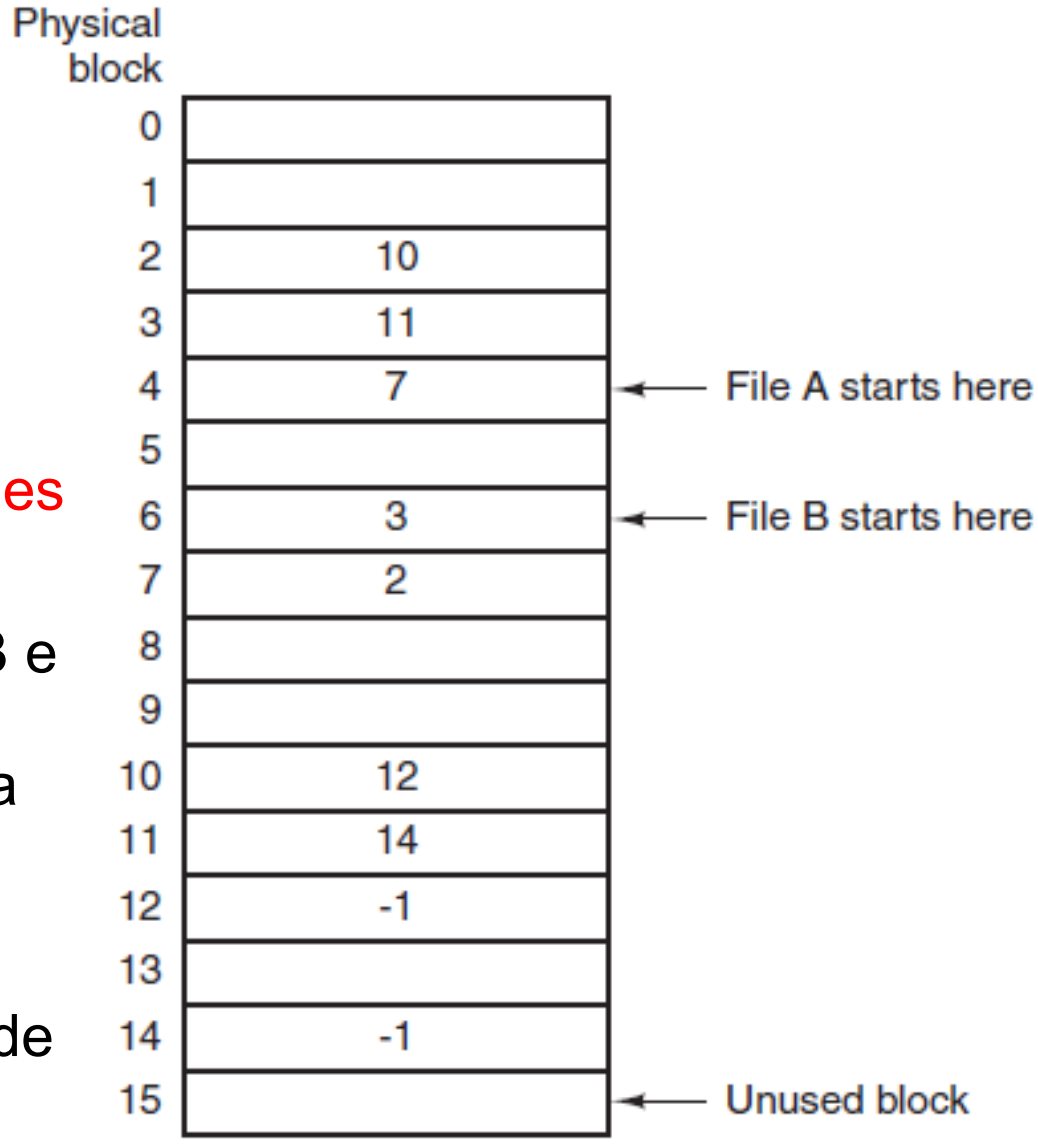
- t[i] armazena o próximo bloco do bloco i
- Vantagens:
  - Economiza espaço nos blocos de dados (que só conterão dados e não ponteiros) → b menor impacta nas velocidades dependentes de b
  - Para uma leitura aleatória (dado um deslocamento em relação ao início do arquivo), o encadeamento (para achar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

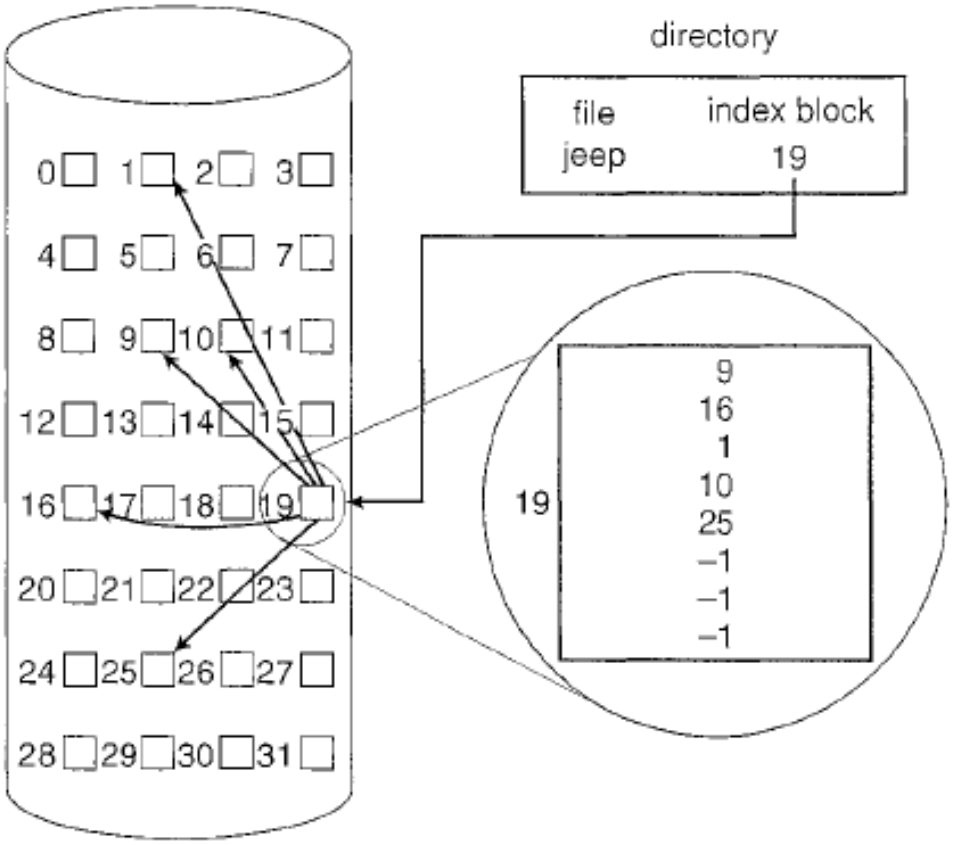


Figure 11.8 Indexed allocation of disk space.

Fonte: (SILBERCHATZ et al, 2009)

Alocação indexada

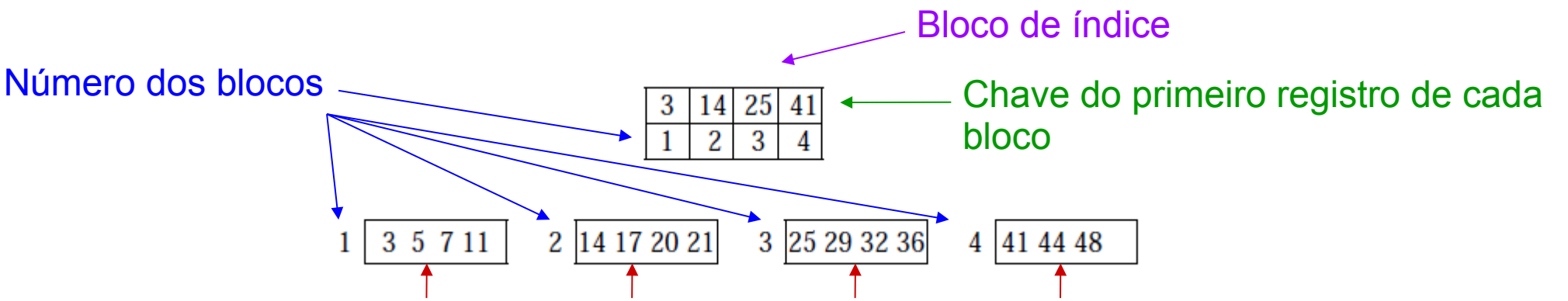
- Resolve fragmentação externa
- Leitura aleatória (acesso direto) eficiente
- Gasto de espaço com ponteiros (blocos de índice) maior que na lista ligada (principalmente para arquivos pequenos)

Aula de hoje

- Veremos **índices** em mais detalhe
- Podem ser utilizados para:
  - Organização física dos arquivos (como visto na aula passada)
  - Estruturar caminhos secundários de acesso aos dados, de forma a acelerar buscas

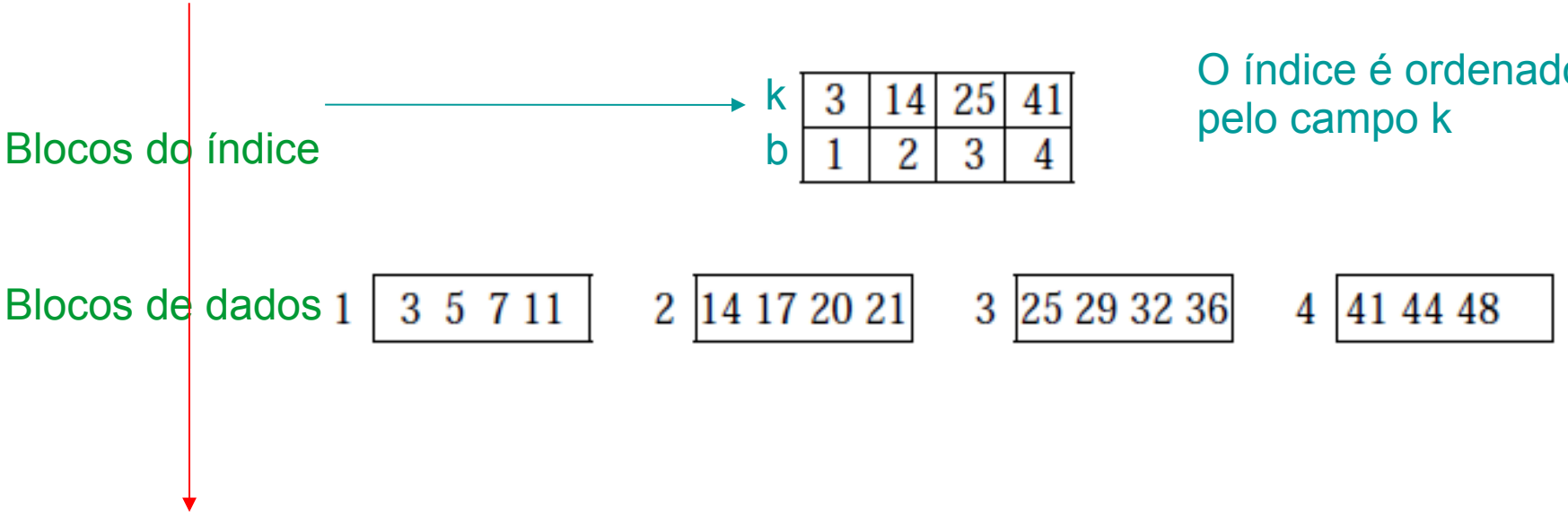
Alocação indexada – arquivos ordenados

- Os blocos de índices possuem, além dos ponteiros para os blocos de dados, a chave do primeiro registro de cada bloco de dado



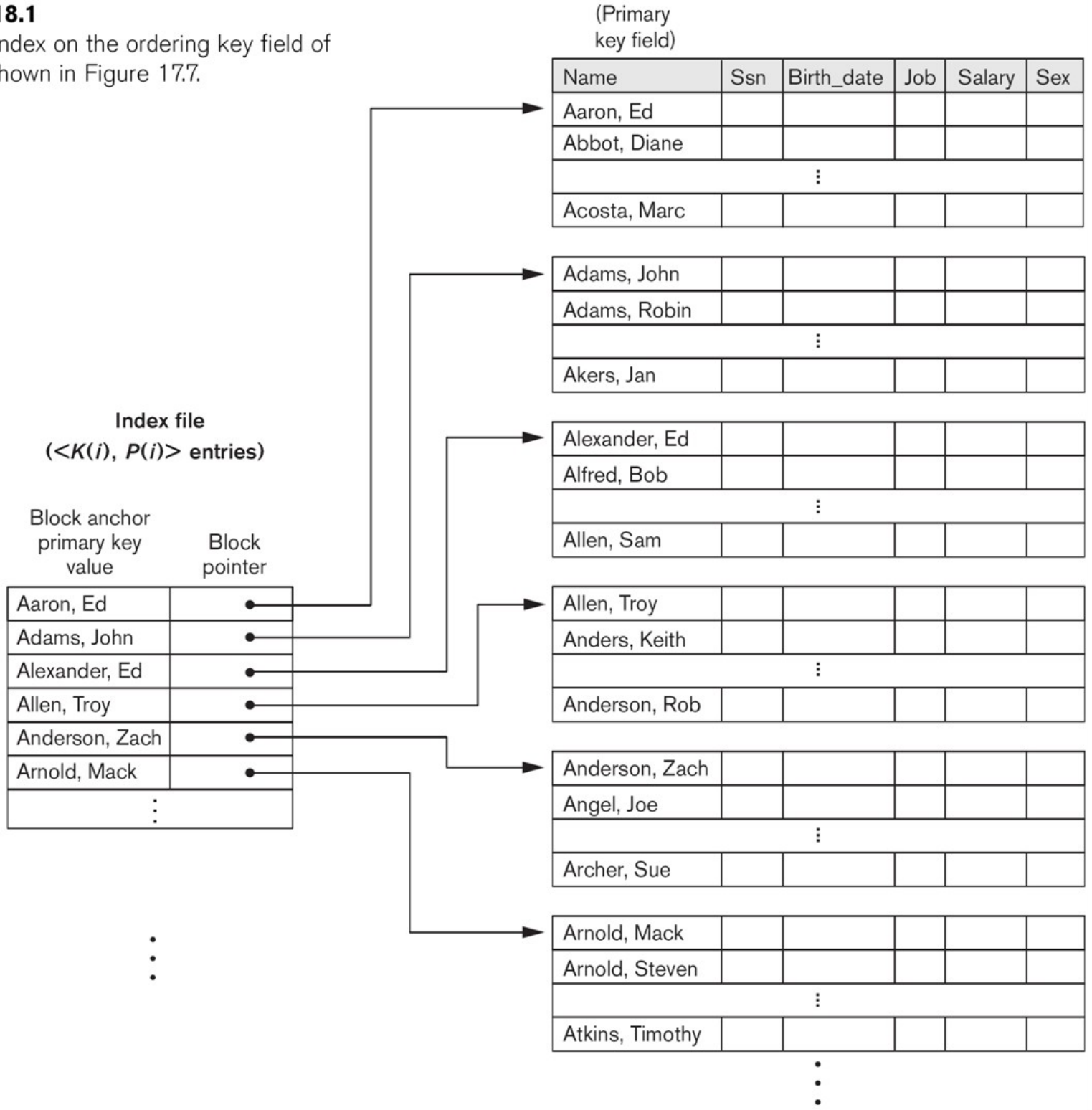
Alocação indexada

- Índice primário: arquivo ORDENADO de registros de tamanho fixo contendo os campos <k,b>, sendo k a chave (primária) do primeiro registro (âncora) do bloco b



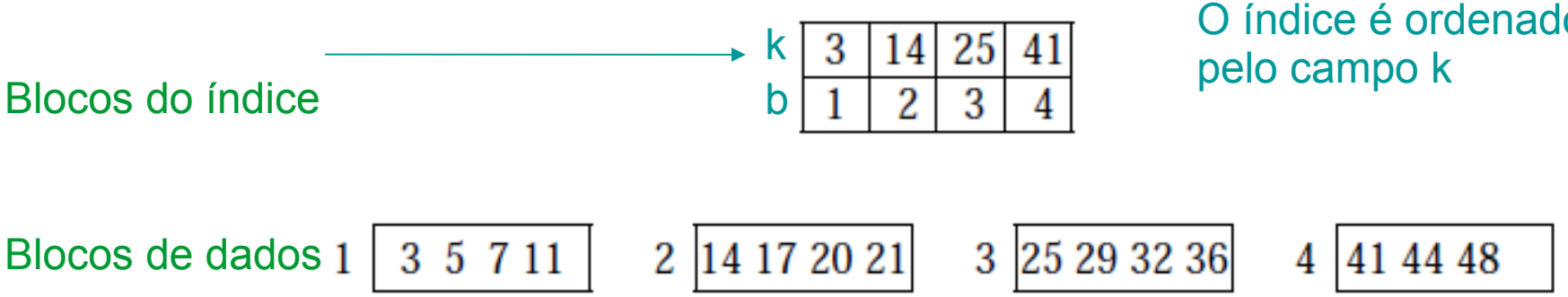
Isto é, k possui valores ÚNICOS e ORDENA FISICAMENTE os registros

Figure 18.1
Primary index on the ordering key field of the file shown in Figure 17.7.



Alocação indexada

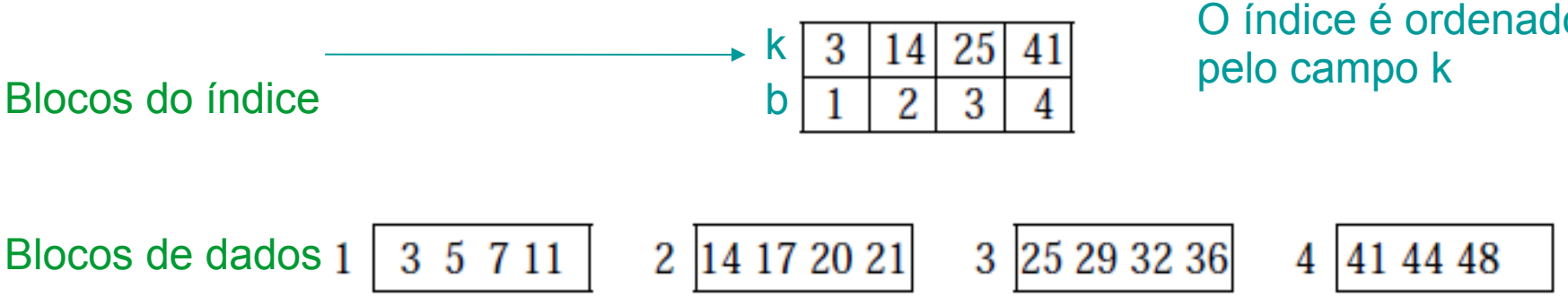
- Índice primário: arquivo ORDENADO de registros de tamanho fixo contendo os campos <k,b>, sendo k a chave (primária) do primeiro registro (âncora) do bloco b



- Qual a vantagem?

Alocação indexada

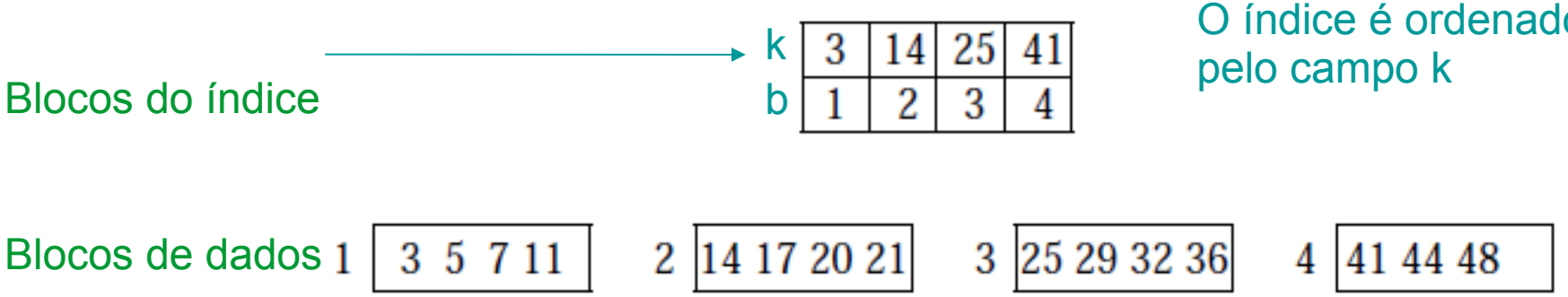
- Índice primário: arquivo ORDENADO de registros de tamanho fixo contendo os campos <k,b>, sendo k a chave (primária) do primeiro registro (âncora) do bloco b



- Qual a vantagem?
  - Índice tem bi blocos, sendo bi << B (B = nr de blocos de dados, no exemplo acima, bi = 1 e B = 4), pois cada entrada é menor que um registro, e há só uma entrada por bloco de dado representado (b)
  - →

Alocação indexada

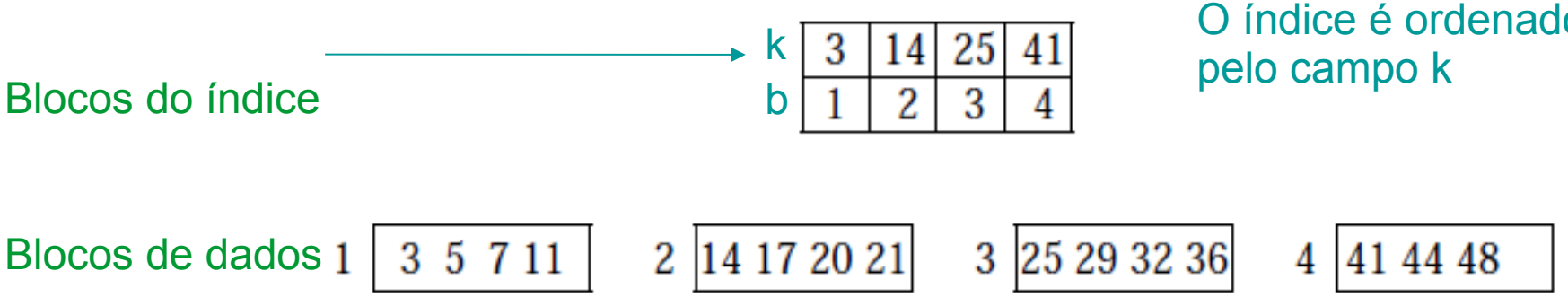
- Índice primário: arquivo ORDENADO de registros de tamanho fixo contendo os campos <k,b>, sendo k a chave (primária) do primeiro registro (âncora) do bloco b



- Qual a vantagem?
  - Índice tem bi blocos, sendo bi << B (B = nr de blocos de dados, no exemplo acima, bi = 1 e B = 4), pois cada entrada é menor que um registro, e há só uma entrada por bloco de dado representado (b)
  - → busca binária no índice é muito mais rápida!!! O(lg bi)

Alocação indexada

- Índice primário: arquivo ORDENADO de registros de tamanho fixo contendo os campos <k,b>, sendo k a chave (primária) do primeiro registro (âncora) do bloco b



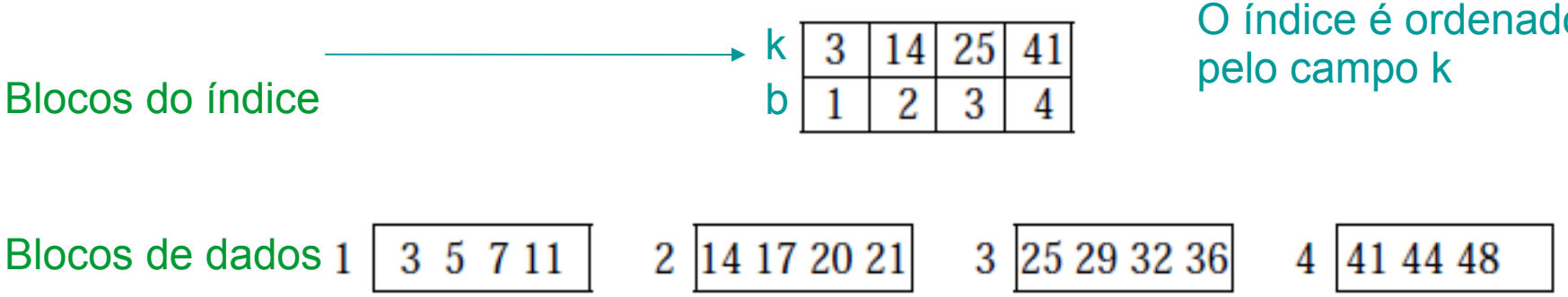
- Observação:

  este é um índice **esparso**

  se fosse **denso**, teria uma entrada por registro de dado

Alocação indexada

- Índice primário: arquivo ORDENADO de registros de tamanho fixo contendo os campos <k,b>, sendo k a chave (primária) do primeiro registro (âncora) do bloco b



- Como fica a inserção e remoção neste tipo de alocação?

Alocação indexada

- Problema: inserção / remoção
  - Altera a posição em disco de vários registros
    → altera âncora de vários blocos
    → altera várias entradas do índice primário
  - Formas de contornar o problema:
    - Remoção por bits de validade
    - Usar um arquivo desordenado de overflow (para as inserções, se necessário)
      - Ou uma lista ligada de overflow (os registros do bloco e da lista podem ser ordenados para melhorar a busca)

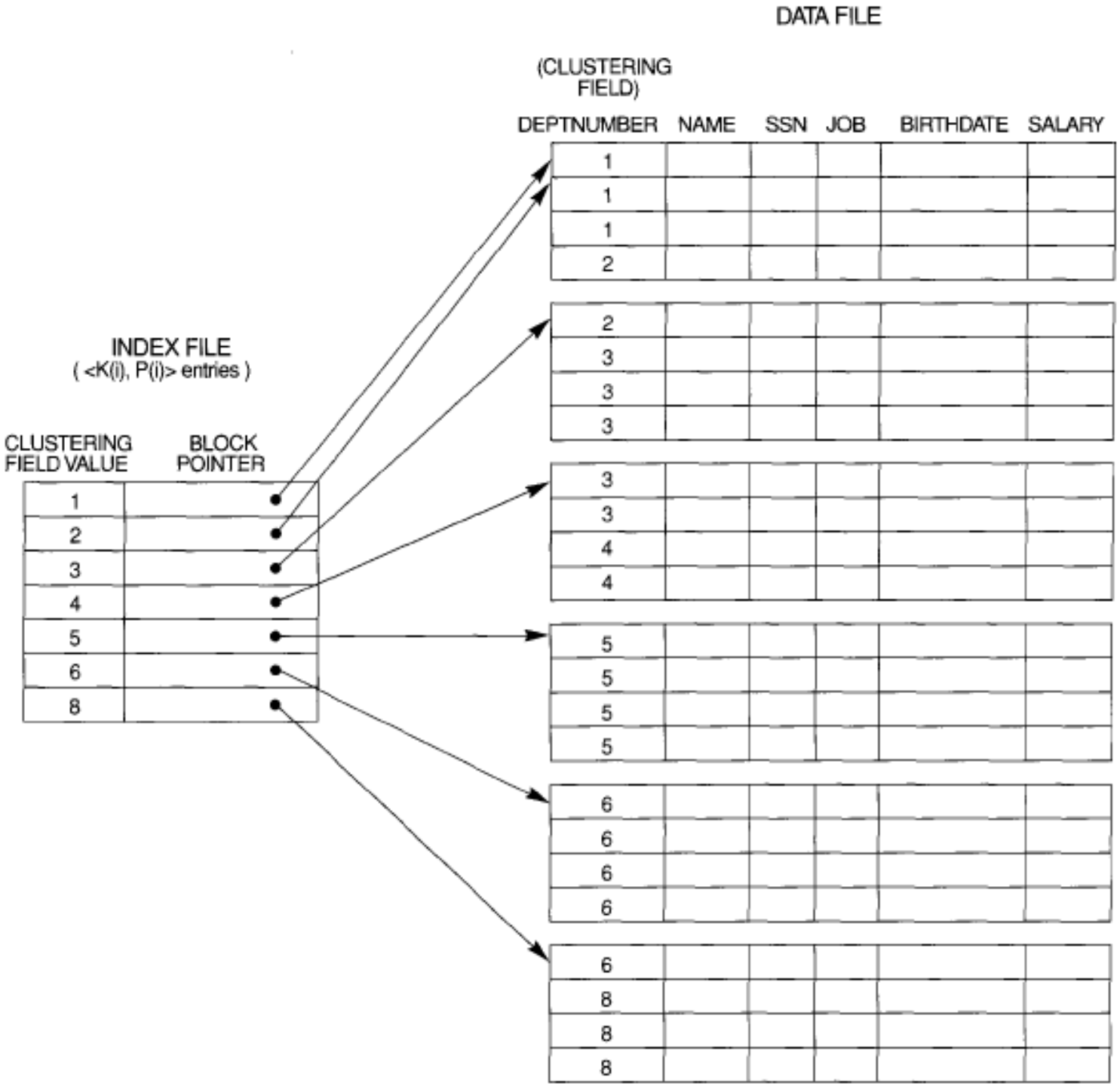
E se o campo de ordenação física não tiver valores únicos?

E se o campo de ordenação física não tiver valores únicos?

→ Índice de *clustering*

Índice de clustering

- Índice de clustering: arquivo ORDENADO de registros de tamanho fixo contendo os campos <c, b>, sendo c um campo de classificação física que não possui valores distintos
  - Uma entrada <c, b> para cada valor distinto de c, sendo b o primeiro bloco da primeira ocorrência da chave com valor c



Fonte:
Elmasri & Navathe

FIGURE 14.2 A clustering index on the DEPTNUMBER ordering nonkey field of an EMPLOYEE file.

Índice de clustering

- Inserção / remoção: ainda problemático, pois c ordena fisicamente os registros
  - Alternativas:
    - Reservar um ou mais blocos para cada valor de c
    - Remoção por bit de validade

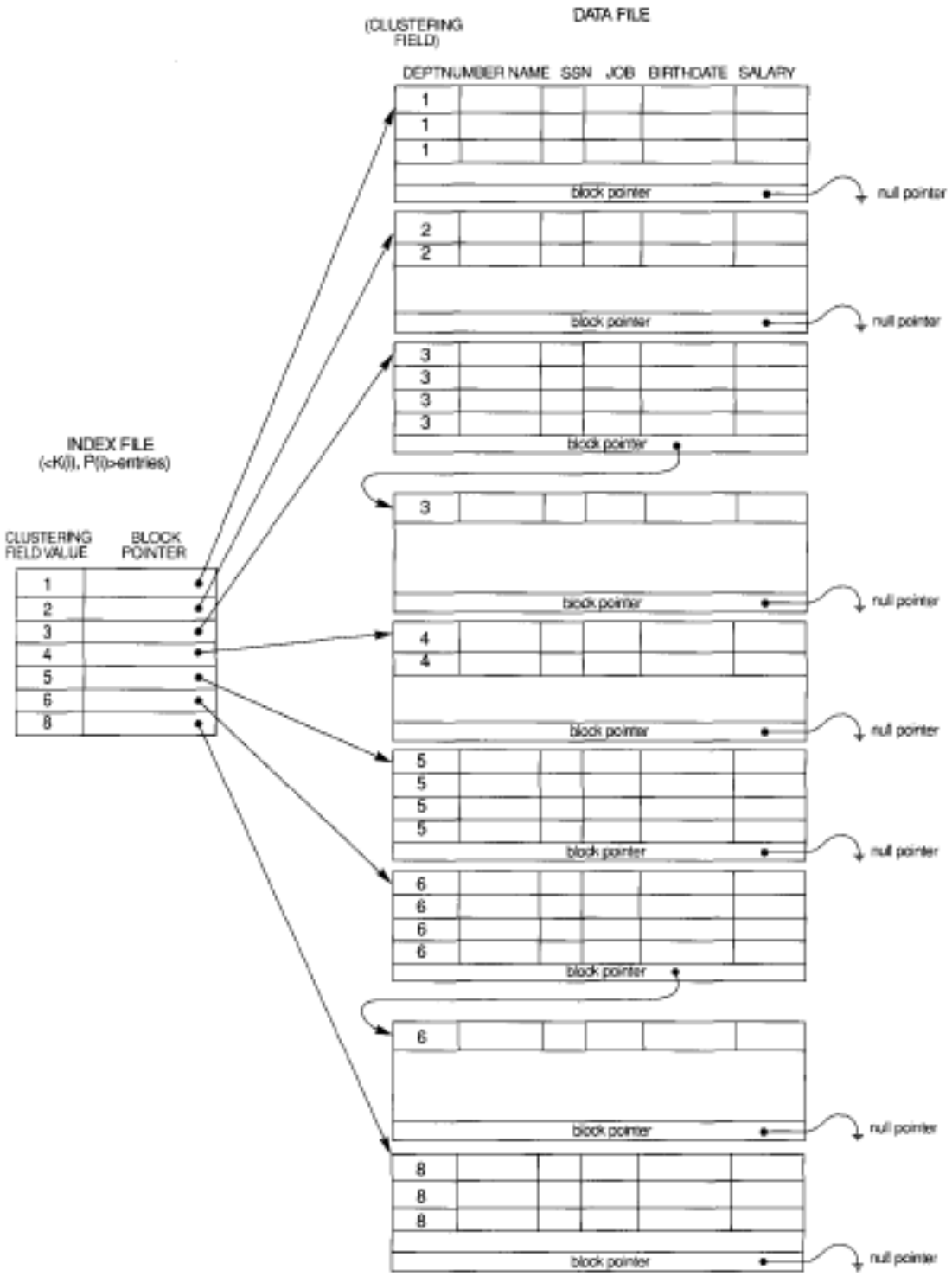


FIGURE 14.3 Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.

Fonte:
Elmasri & Navathe

Índice de clustering

- Busca:
  - sem sucesso: NÃO acessa bloco de dado
  - Quer só o primeiro registro ou todos?
    - Primeiro: um bloco de dado (informado no índice)
    - Todos: tem que ler q blocos
  - Quanto maior o número de valores distintos, maior o tempo de busca binária nos índices

Índices primários, clustering, e...

- Índices primários e de clustering são baseados no campo de ordenação física de um arquivo (que é único)
  - → cada arquivo pode ter no máximo um índice primário OU um índice de clustering
- E para os campos que não ordenam fisicamente o arquivo? Podemos ter algo semelhante?
  - Quantos **índices secundários** quiser!

Índices secundários

- Índice secundário: arquivo ORDENADO de registros de tamanho fixo contendo os campos <i, w>, sendo i um campo de indexação que NÃO ordena fisicamente os registros, podendo ter valores distintos ou não
  - w aponta para um bloco ou registro

- Podem existir vários índices secundários

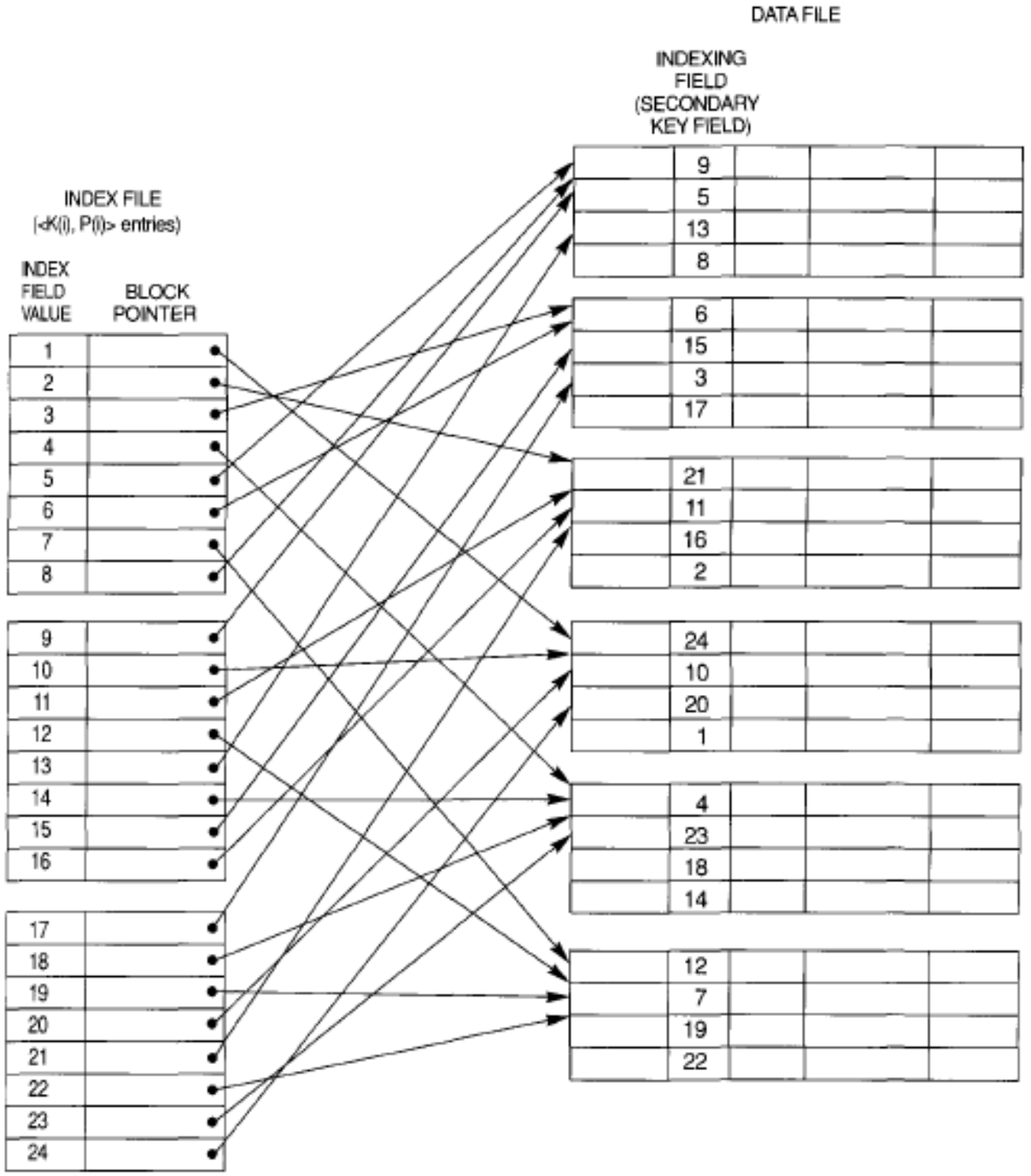


FIGURE **14.4** A dense secondary index (with block pointers) on a nonordering key field of a file.

Fonte:
Elmasri & Navathe

Índices secundários

- Se i tem valores distintos, o índice é denso
- Se i tem valores duplicados:
  - Opção 1: diversas entradas para um mesmo i, cada uma com w apontando para um registro (denso)
  - Opção 2: 1 entrada para cada valor de i, e w multivalorado (campo de tamanho variável) aponta para blocos (esparso)
  - Opção 3: uma entrada para cada valor de i e w (tamanho fixo) aponta para um bloco de ponteiros de registros (esparso) – mais usada

Pensem na busca, inserção e remoção em cada caso...

Para esses 3 tipos de índices:

- Busca binária – O(log bi)
- O que dá para fazer se o arquivo é muito grande e o próprio índice ficou grande (com muitos blocos, ie, grande bi)?

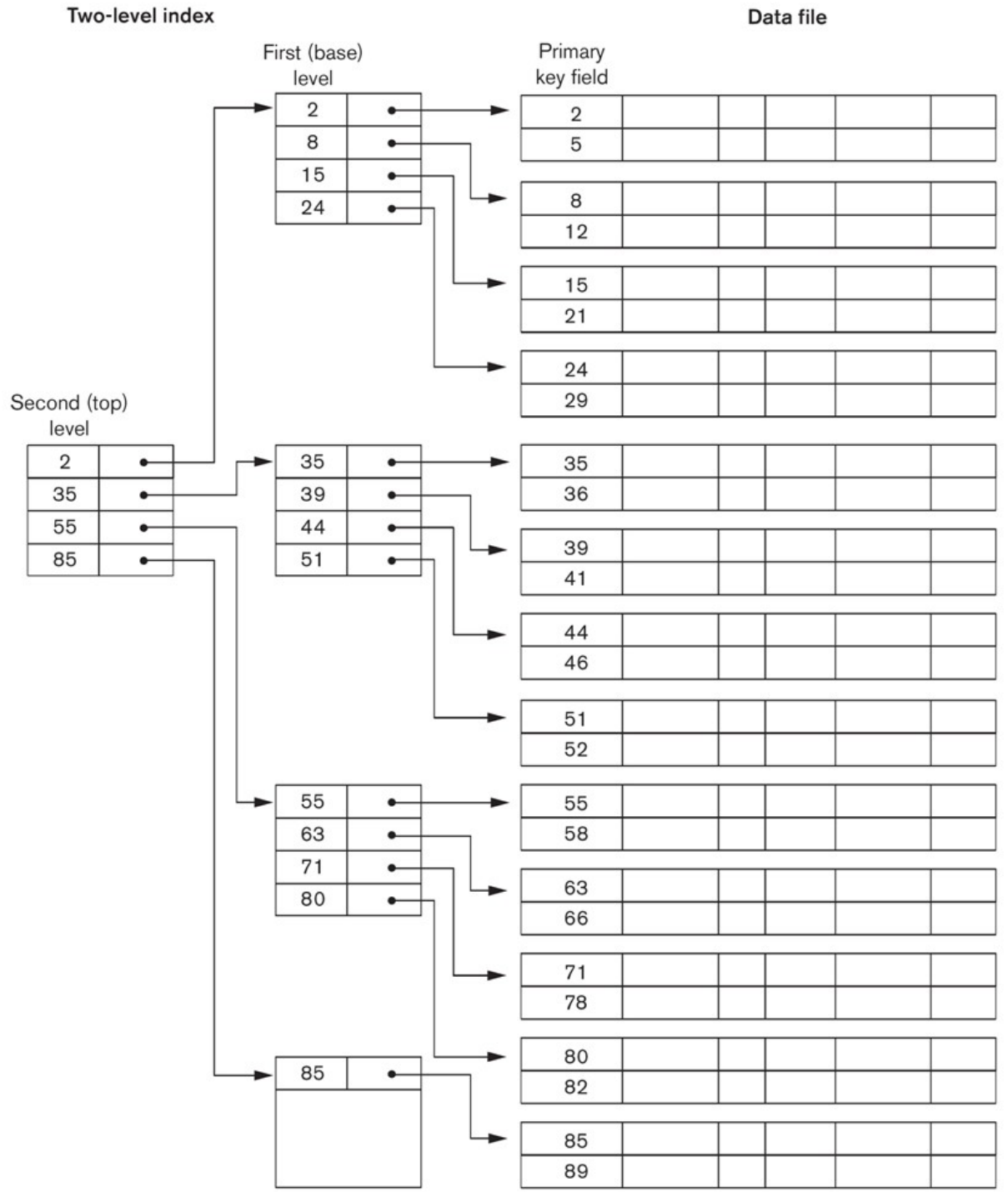
Para esses 3 tipos de índices:

- Busca binária – O(log bi)
- O que dá para fazer se o arquivo é muito grande e o próprio índice ficou grande (com muitos blocos, ie, grande bi)?
  - ÍNDICE DO ÍNDICE!!!

Organização indexada multiníveis

- Nível 1: arquivo de índices para os dados
- Nível 2: arquivo de índices para o arquivo de índices nível 1
- Se no nível 2 precisar de mais de um bloco, criar nível 3!
- ….

Figure 18.6
A two-level primary index resembling ISAM (Indexed Sequential Access Method) organization.



Organização indexada multiníveis

- Nível 1: arquivo de índices para os dados
- Nível 2: arquivo de índices para o arquivo de índices nível 1
- Se no nível 2 precisar de mais de um bloco, criar nível 3!
- ….
- **Busca**:

Organização indexada multiníveis

- Nível 1: arquivo de índices para os dados
- Nível 2: arquivo de índices para o arquivo de índices nível 1
- Se no nível 2 precisar de mais de um bloco, criar nível 3!
- ….
- **Busca**: 1 acesso em cada nível (rápida!) - O(t) - precisa acessar t blocos, sendo t o número de níveis

  t =

Organização indexada multiníveis

- Nível 1: arquivo de índices para os dados
- Nível 2: arquivo de índices para o arquivo de índices nível 1
- Se no nível 2 precisar de mais de um bloco, criar nível 3!
- ….
- **Busca**: 1 acesso em cada nível (rápida!) - O(t) - precisa acessar t blocos, sendo t o número de níveis

$t = \text{ceil}(\log_{fbi} r^1)$,

fbi = nr de registros que cabem em um bloco de índice (fator de blocagem dos blocos de índice)

r^1 = nr total de registros de índice no nível 1

Organização indexada multiníveis

- **Inserção / remoção**: ?

Organização indexada multiníveis

- **Inserção / remoção**: cada vez mais complicado!!!
  - Posso ter que alterar tudo!

Referências

- Slides da Profa. Graça (ICMC) - 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 14.
- 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