ACH2024

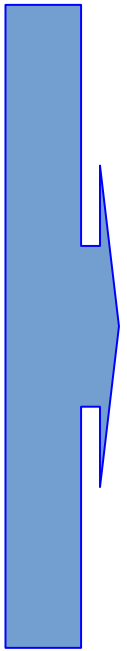
Aula 14

Acesso em memória secundária
Alocação sequencial

Profa. Ariane Machado Lima

Aula passada

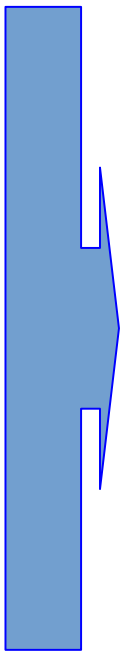
- Tipos de arquivo:
  - Tipos variados
  - Serialização de estruturas de dados (binária ou não binária)
  - Foco em tabelas: registros formados por campos
- Oganização interna de registros:
  - Comprimento fixo
  - Número fixo de campos
  - Indicador de comprimento
  - Delimitadores
  - Uso de índices
- Organização interna de campos:
  - Comprimento fixo
  - Indicador de comprimento
  - Delimitadores
  - Uso de etiquetas (tags)



Vantagens e desvantagens relacionadas a:

Aula passada

- Tipos de arquivo:
  - Tipos variados
  - Serialização de estruturas de dados (binária ou não binária)
  - Foco em tabelas: registros formados por campos
- Oganização interna de registros:
  - Comprimento fixo
  - Número fixo de campos
  - Indicador de comprimento
  - Delimitadores
  - Uso de índices
- Organização interna de campos:
  - Comprimento fixo
  - Indicador de comprimento
  - Delimitadores
  - Uso de etiquetas (tags)



Vantagens e desvantagens relacionadas a:

- aproveitamento de espaço interno
- tempo para localizar um campo ou registro

Aula passada

- Exercícios de serialização e de manipulação de arquivos de registros
- Quem ainda não fez tente fazer, depois dê uma olhada nos dois programas que deixei lá no edisciplinas

Aula de hoje:

Acesso em memória secundária

Alocação sequencial

Segunda parte da disciplina

- Organização interna de arquivos
- Tipos de alocação de arquivos na memória secundária:
  - Sequencial
  - Ligada
  - Indexada
  - Árvores-B
  - Hashing (veremos também hashing em memória principal)
- Algoritmos de processamento cossequencial e ordenação em disco

Por que se preocupar com alocação de arquivos na memória secundária?

- Por conta dos problemas envolvidos com esse dispositivo (que é o que veremos agora)

Armazenamento não volátil de arquivos

- Quando estamos falando em arquivo, normalmente estamos falando em armazenamento em memória SECUNDÁRIA
- Memória secundária:
  - HD (hard disk)
  - SSD (Solid state disk)
  - CD-ROM
  - DVD
  - Pen-drives
  - Chips de memória
  - Fita magnética (ainda usada para backup de disco - barata)
  - ...

Armazenamento não volátil de arquivos

- Quando estamos falando em arquivo, normalmente estamos falando em armazenamento em memória SECUNDÁRIA
- Memória secundária:
  - **HD (hard disk)**
  - SSD (Solid state disk)
  - CD-ROM
  - DVD
  - Pen-drives
  - Chips de memória
  - Fita magnética (ainda usada para backup de disco - barata)
  - ...

Discos X Memória Principal

- Capacidade de Armazenamento
  - HD – muito alta, a um custo relativamente baixo
  - RAM – limitada pelo custo e espaço
- Tipo de Armazenamento
  - HD – não volátil
  - RAM – volátil

MAS....

Memória primária x secundária

VELOCIDADE DE ACESSO

Mem. volátil (10^{-9} seg)

Mem. secundária (10^{-3} seg)

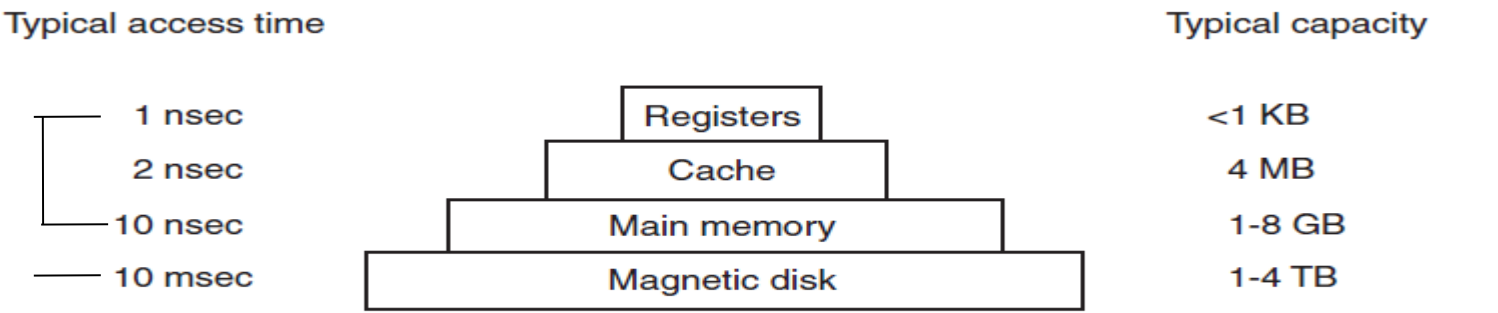
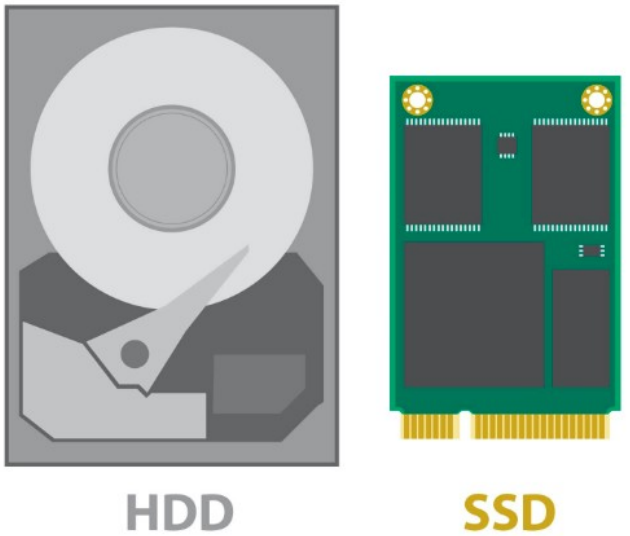


Figure 1-9. A typical memory hierarchy. The numbers are very rough approximations.

(TANEMBAUM & BOS, 2015)

Observação: HD x SSD

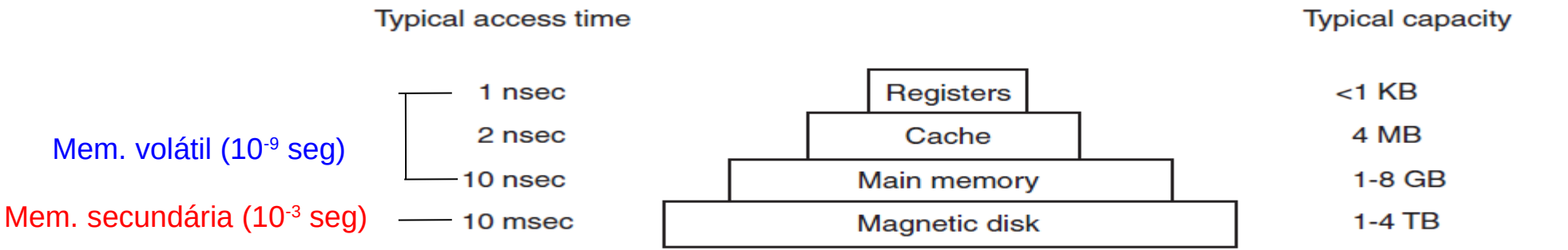


https://kazuk.com.br/blog/o-que-e-ssd-saiba-mais-sobre-essa-forma-de-armazenar/

Velocidade x custo (SSDs da ordem de 10, 20x mais rápidos apenas)

Memória primária x secundária

VELOCIDADE DE ACESSO



Mem. volátil (10^{-9} seg)

Mem. secundária (10^{-3} seg)

Figure 1-9. A typical memory hierarchy. The numbers are very rough approximations.

(TANEMBAUM & BOS, 2015)

POR QUE ESSA DIFERENÇA?

HD – Hard Disk

- No início, mais em sistemas corporativos
  - 1.70m de altura e de comprimento, quase 1 tonelada
  - Chamado "unidade de disco"

IBM 350 (1956)

HD – Hard Disk

- HD
  - Em 1973, IBM lançou o que é considerado o pai dos HDs modernos
    - Winchester

Estrutura de um HD

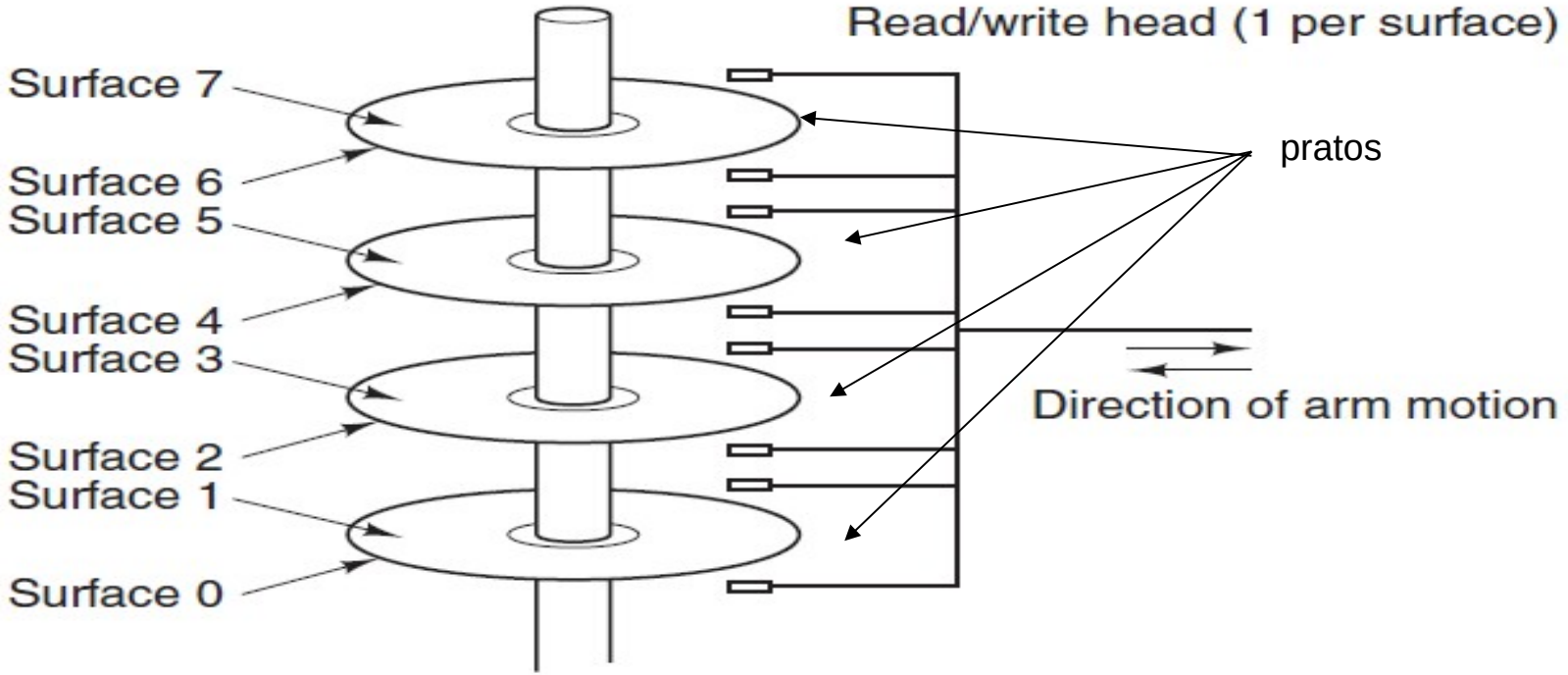
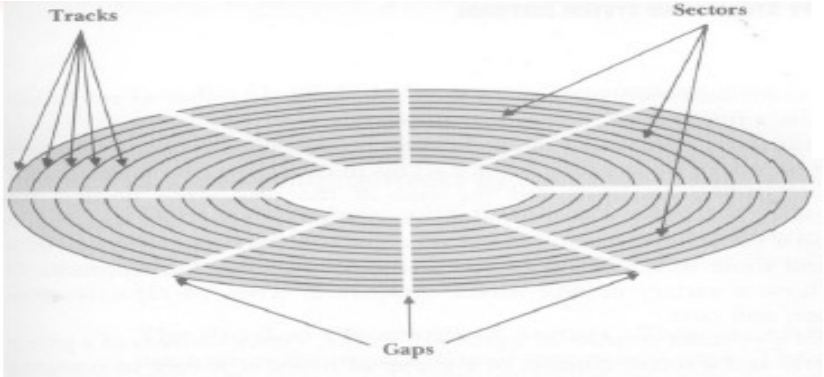


Figure 1-10. Structure of a disk drive.

(TANEMBAUM & BOS, 2015)

Organização da informação no disco



- Disco: conjunto de 'pratos' empilhados
  - Dados são gravados nas superfícies desses pratos
- Superfícies: são organizadas em trilhas
- Trilhas: são organizadas em setores
- Cilindro: conjunto de trilhas na mesma posição

Um **setor** é a menor porção endereçável do disco

A divisão de uma trilha em setores é definida pelo disco, e não pode ser mudada (geralmente 1 setor = 512 bytes).

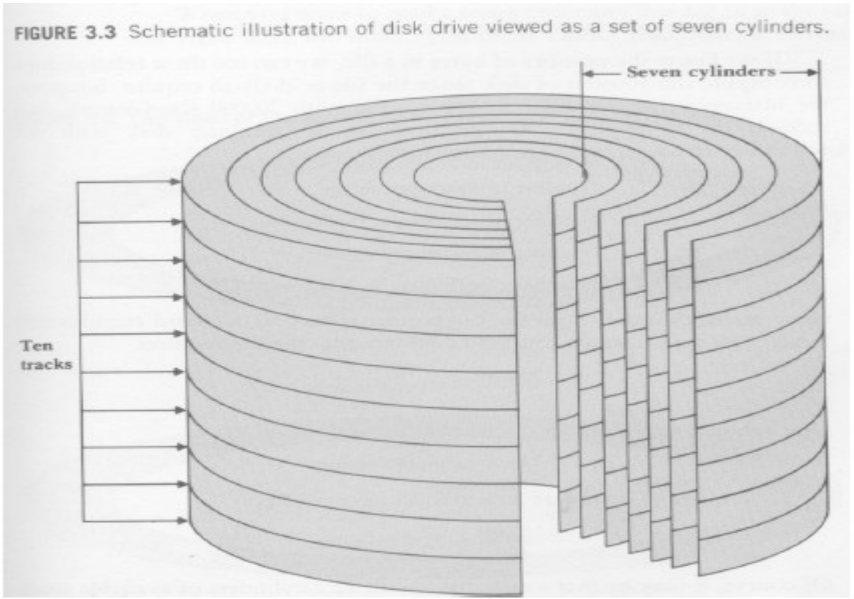
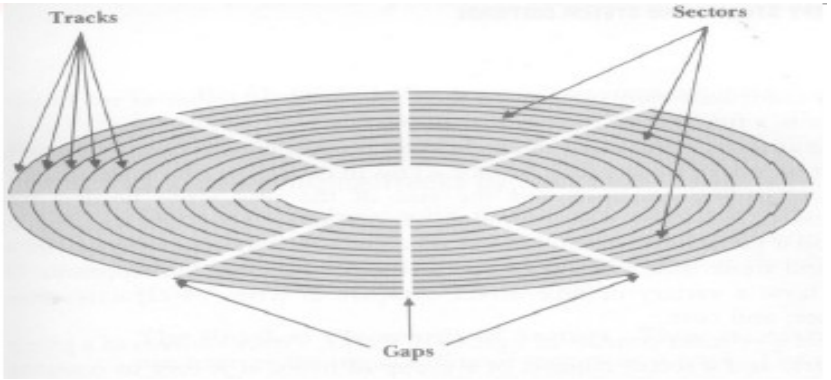


FIGURE 3.3 Schematic illustration of disk drive viewed as a set of seven cylinders.

O você acha que o computador deve fazer para ler algum dado do HD?



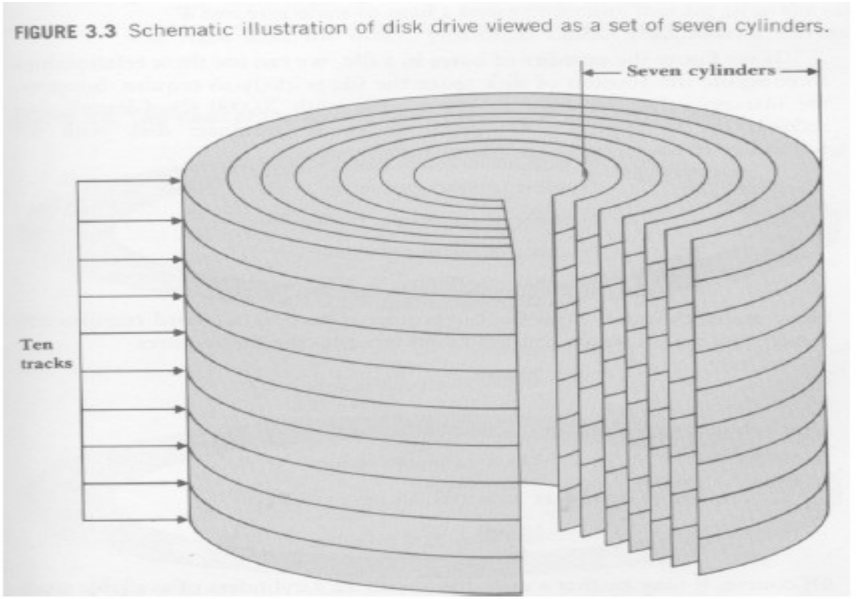
O você acha que o computador deve fazer para ler algum dado do HD?

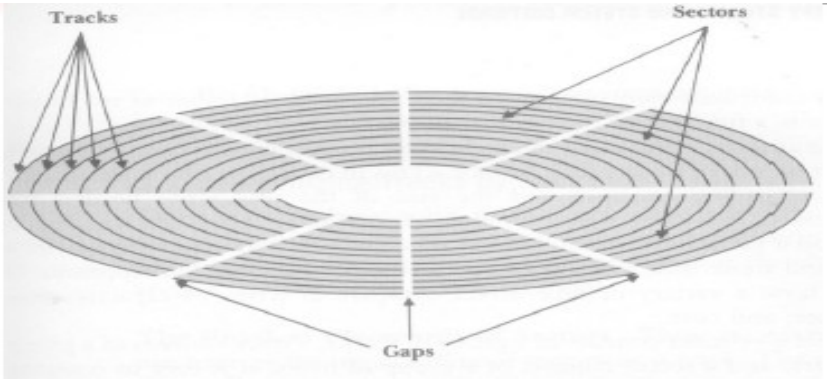
1) Identifica em que setor/trilha/superfície está a informação

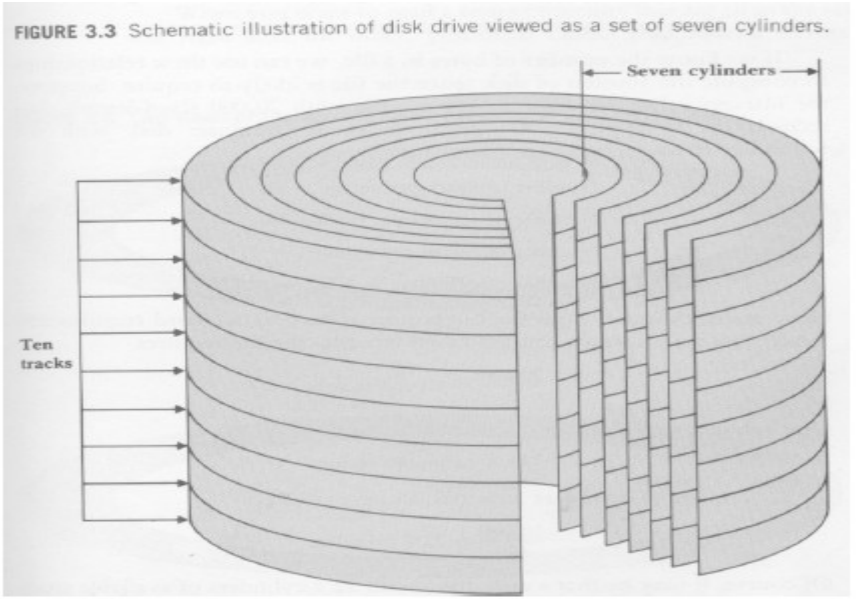
2) Movimenta o braço de leitura para o cilindro correto (para poder acessar a trilha correta)

3) Rotaciona o prato para posicionar a cabeça de leitura sobre o setor correto

4) Faz a leitura de um certo nr de bytes







O você acha que o computador deve fazer para ler algum dado do HD?

1) Identifica em que setor/trilha/superfície está a informação

2) Movimenta o braço de leitura para o cilindro correto (para poder acessa a trilha correta)

3) Rotaciona o prato para posicionar a cabeça de leitura sobre o setor correto

4) Faz a leitura de um certo nr de bytes

Passos 2 e 3 (chamados SEEK) são mecânicos! Por isso demora tanto!

Seeking

- Movimento de posicionar a cabeça de L/E sobre a trilha/setor desejado
- O conteúdo de todo um cilindro pode ser lido com 1 único *seeking*
- É o movimento mais lento da operação leitura/escrita
- Deve ser reduzido ao mínimo

Observação: Acesso sequencial x acesso aleatório (ou randômico ou direto) – com relação ao DISPOSITIVO

- Apesar do custo de um seek, o acesso é direto, pois não é necessário ler dados anteriores
    - Também chamado de acesso aleatório ou randômico
- Em contraposição, fitas demandam acesso sequencial, ou seja, é preciso passar por todo o trecho de fita anterior ao que contém o dado desejado

Quanto mais seeks, mais cara a leitura

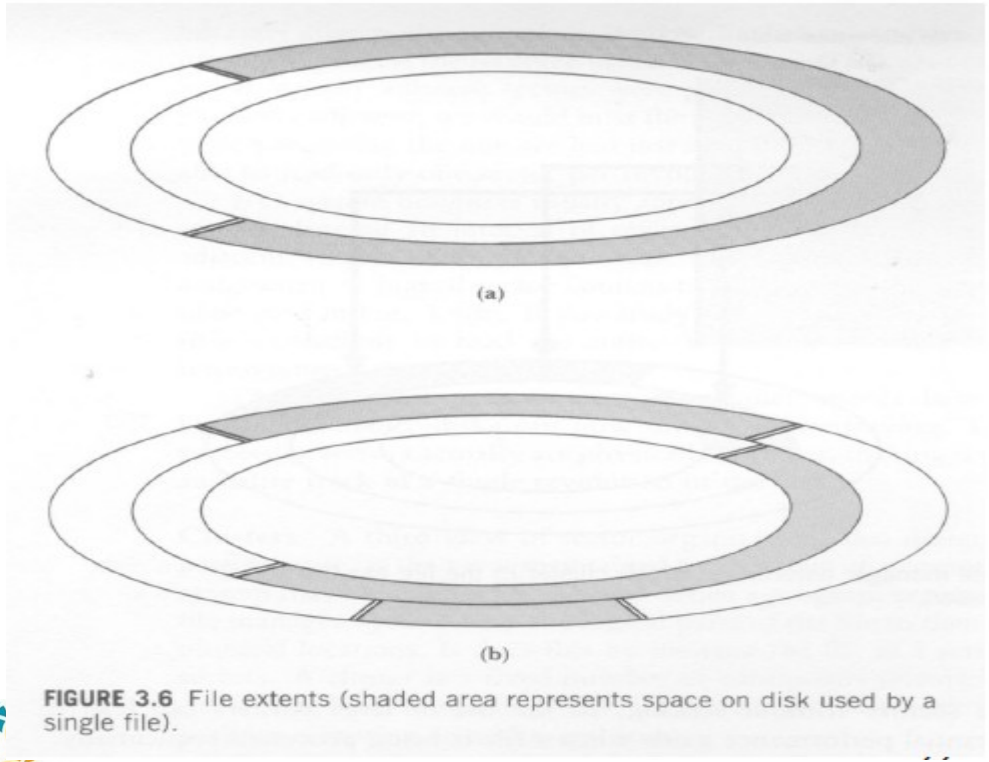


FIGURE 3.6 File extents (shaded area represents space on disk used by a single file).

Os arquivos não são estáticos, por isso podem não estar armazenados de forma contígua pelo disco

→ têm que armazenar pedaços dos arquivos nos setores (mais ou menos como se fosse uma lista ligada)

O tempo de acesso a uma informação, na prática, depende:

- da distribuição dos dados de um arquivo pelo disco
- da tecnologia do disco

Como calcular tempo de acesso

- Se acesso a disco é caro (em termos de tempo) e queremos escolher estruturas de dados que diminuam o tempo de acesso, precisamos poder calcular (ou estimar) o tempo de acesso de uma forma não muito complicada, pelo menos:

  - independente da distribuição e localização do arquivo em disco

  - independente da tecnologia

- Para simplificar os cálculos, podemos fazer a seguinte aproximação (pior caso):

  - considera-se que é necessário um *seek* por "pedaço" de arquivo a ser lido:

    1) considerando que eu não preciso ler o arquivo inteiro em um dado instante, pois posso estar interessado em apenas um "pedaço"

    2) como há outros processos sendo executados, quando eu quiser ler outro "pedaço" do meu arquivo a cabeça de leitura do disco pode não estar no mesmo lugar onde parou a última leitura desse meu arquivo

  - tempo de acesso total = nr de acessos (*seeks*) * tempo de um acesso

Como calcular tempo de acesso

- Se acesso a disco é caro (em termos de tempo) e queremos escolher estruturas de dados que diminuam o tempo de acesso, precisamos poder calcular (ou estimar) o tempo de acesso de uma forma não muito complicada, pelo menos:

  - independente da distribuição e localização do arquivo em disco

  - independente da tecnologia

- Para simplificar os cálculos, podemos fazer a seguinte aproximação (pior caso):

  - considera-se que é necessário um *seek* por "pedaço" de arquivo a ser lido:

    1) considerando que eu não preciso ler o arquivo inteiro em um dado instante, pois posso estar interessado em apenas um "pedaço"

    2) como há outros processos sendo executados, quando eu quiser ler outro "pedaço" do meu arquivo a cabeça de leitura do disco pode não estar no mesmo lugar onde parou a última leitura desse meu arquivo

  - **tempo de acesso total = nr de acessos (*seeks*) * tempo de um acesso**

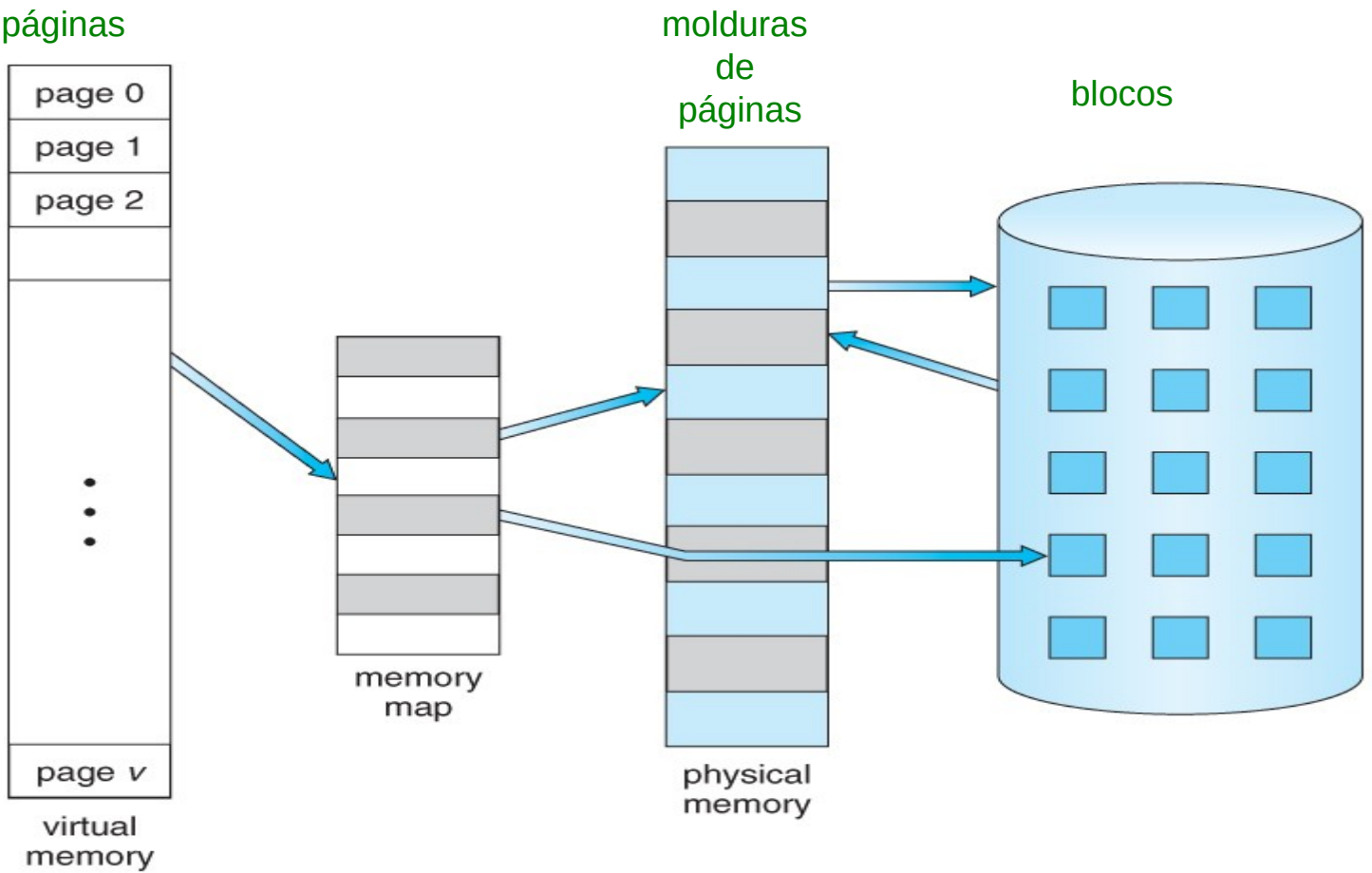
De que tamanho tem que ser esse pedaço?
Pode ser um byte? E se meu programa quiser ler um byte de cada vez?
E quando eu leio um "pedaço", armazeno onde essa informação?

Paginação

- Disco é grande, mas o acesso é lento
- Fazer várias pequenas leituras no disco tornaria os programas inviáveis…
- Solução: ler um “pedaço” razoável do disco, trazer para a memória principal, processá-la lá conforme necessário, se for salvar reescrever o pedaço todo no disco novamente
- Ou seja, tenho um subconjunto do meu disco em memória principal
- O conteúdo desse “pedaço”, contendo x setores (x um número inteiro), será virtualmente chamado de “página” ou bloco, e será armazenado fisicamente em um “pedaço” da memória chamado de “moldura de página”
- O tamanho de uma página (que é igual ao tamanho de uma moldura de página) é definida pelo Sistema Operacional (SO) durante a formatação do disco, e não pode ser alterada dinamicamente.

Memória virtual

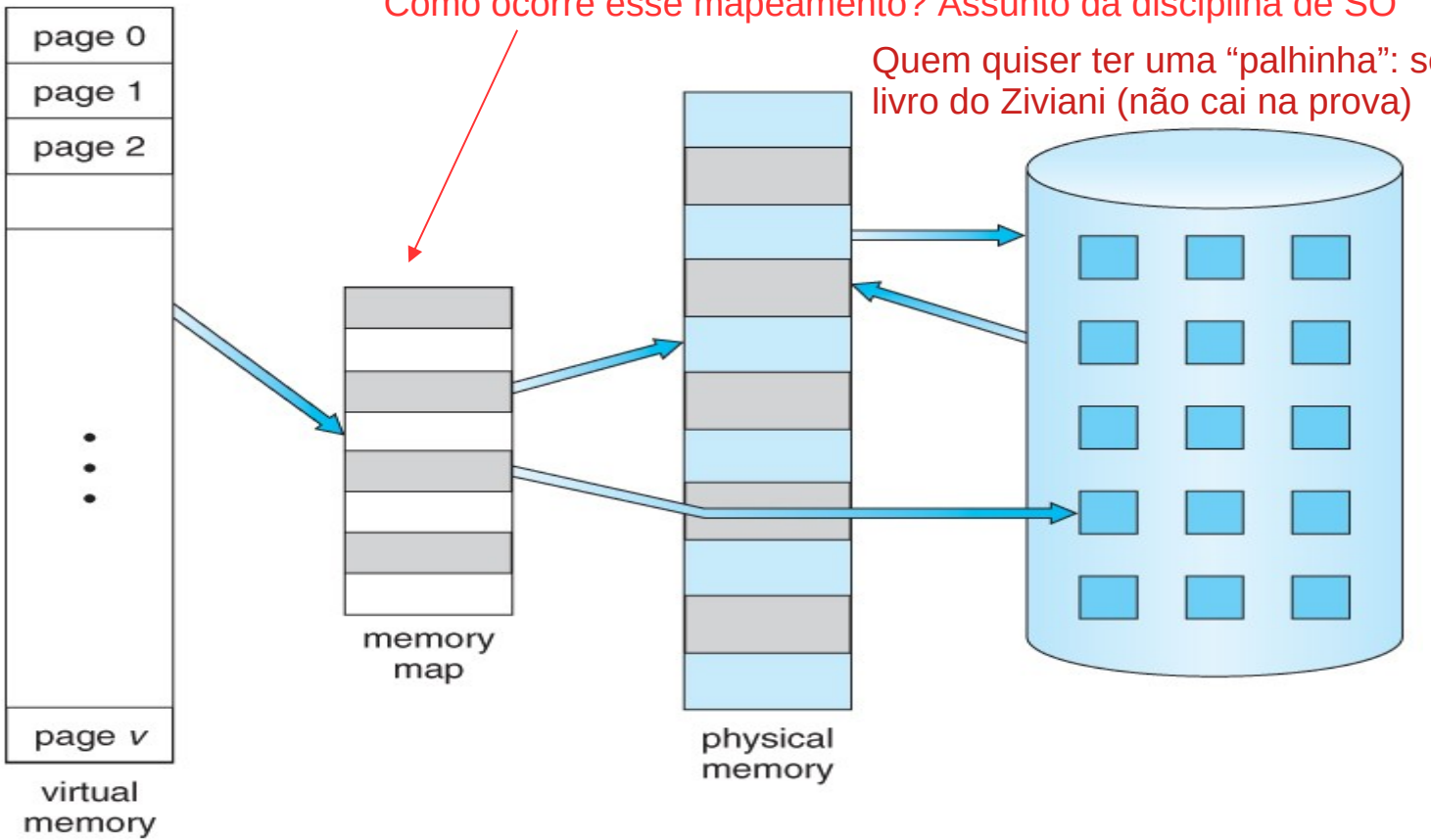
- O Sistema Operacional (SO) gerencia se a informação (ou seja, a página que contém informação) já está em memória
  - Se não estiver, precisa carregá-la (em uma moldura de página, ie, trecho que memória principal destinado a armazenar uma "página" - ou bloco - do disco)
  - Se não tiver moldura disponível, precisa descarregar alguma no disco antes
- Com isso os programas podem endereçar todo o disco como se ele estivesse em memória principal (memória virtual)



https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/9_VirtualMemory.html

Como ocorre esse mapeamento? Assunto da disciplina de SO

Quem quiser ter uma “palhinha”: seção 6.1 do livro do Ziviani (não cai na prova)



https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/9_VirtualMemory.html

Organização de arquivos na memória secundária

- **Bloco**: unidade de transferência de dados entre memória principal e a memória secundária
  - Obs: para o disco (hardware) o bloco físico é um setor (ex: 512 bytes); já o SO (software) define um bloco lógico, que corresponde a uma página, com tamanho = 1 ou mais setores (usualmente 4kb), que também é usado pelos gerenciadores de bancos de dados.
  - Usaremos aqui o termo bloco de forma genérica como bloco lógico, a não ser quando especificado (ie, o bloco tem o tamanho de uma página)

Cabeçalhos de arquivos

- Cabeçalho do arquivo (descritor) pode conter informações como:
  - Descrição dos formatos dos campos de um registro
  - Códigos de tipos de registros para registros de tamanho variável
  - Primeiro e último bloco
  - Informações para determinar os endereços dos seus blocos
    - Abrir um arquivo significa trazer para a memória o cabeçalho do arquivo (blocos contendo essas informações que ficarão em memória até o arquivo ser fechado)

Primeiro bloco

último bloco

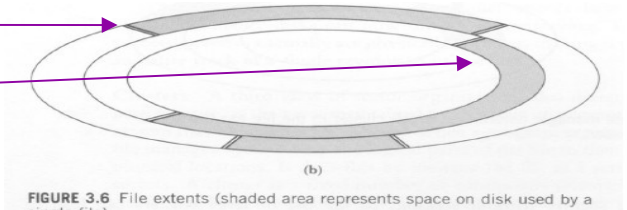


FIGURE 3.6 File extents (shaded area represents space on disk used by a

Organização de arquivos na memória secundária

- Pensando em arquivos de dados, é comum considerar que:
  - Nenhum registro é maior que um bloco

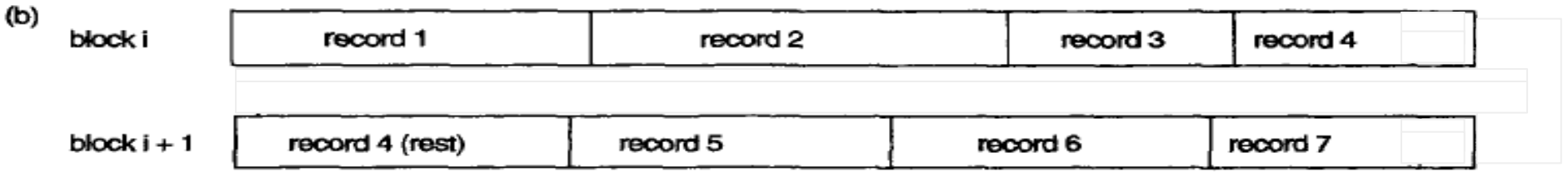
- Se R (tamanho fixo do registro, para simplificar) e B (tamanho do bloco) e R ≤ B: o que normalmente é feito por questões de desempenho...

  fator de blocagem fb = floor(B/R)

  = número de registros inteiros que cabem em um bloco

Organização de arquivos na memória secundária

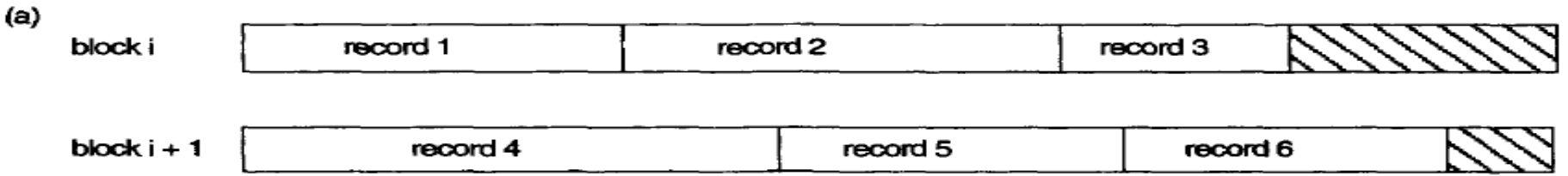
- **Organização *espalhada***: os blocos são totalmente preenchidos; se um registro não cabe inteiramente na parte vazia do bloco, coloca o que couber e um ponteiro para o próximo bloco



(ELSMARI & NATATHE)

Organização de arquivos na memória secundária

- **Organização não *espalhada***: registros não podem ser divididos. Cada bloco pode conter até fb registros.



(ELSMARI & NATATHE)

Organização de arquivos na memória secundária

- **Organização não *espalhada***: registros não podem ser divididos. Cada bloco pode conter até fb registros.
  - Se os registros tiverem tamanho fixo = R, blocos de tamanho B e taxa de blocagem = fb:

Perda de espaço em cada bloco:

Organização de arquivos na memória secundária

- **Organização não *espalhada***: registros não podem ser divididos. Cada bloco pode conter até fb registros.
  - Se os registros tiverem tamanho fixo = R, blocos de tamanho B e taxa de blocagem = fb:

Perda de espaço em cada bloco: B - (fb*R)

Organização de arquivos na memória secundária

- **Organização não *espalhada***: registros não podem ser divididos. Cada bloco pode conter até fb registros.
  - Se os registros tiverem tamanho fixo = R, blocos de tamanho B e taxa de blocagem = fb:

Perda de espaço em cada bloco: B - (fb*R)

ORGANIZAÇÃO BASTANTE COMUM
Por quê?

Organização de arquivos na memória secundária

- **Organização não *espalhada***: registros não podem ser divididos. Cada bloco pode conter até fb registros.
  - Se os registros tiverem tamanho fixo = R, blocos de tamanho B e taxa de blocagem = fb:

Perda de espaço em cada bloco: B - (fb*R)

ORGANIZAÇÃO BASTANTE COMUM,
pois facilita muito localizar os registros.

Alocação de blocos na memória secundária

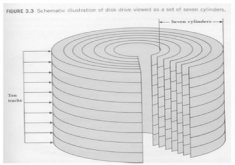
- Leituras, escritas, buscas, etc., são realizadas por blocos.
- Os arquivos não são estáticos, eles crescem e diminuem.
- Estratégias de alocação de blocos no disco e organização de registros pelos blocos devem considerar esse fato
  - Sequencial não ordenado (heap files)
  - Sequencial ordenado (sorted files)
  - Por listas ligadas
  - Indexado
  - Árvores B / B+
  - Hashing
- Para cada estratégia analisaremos a complexidade de leitura sequencial (ler o arquivo do início ao fim), leitura aleatória (busca de um dado registro), inserção e remoção de registros
- Complexidade será definida em termos de número de número de seeks (estimado no pior caso pelo número de blocos a serem lidos); **assume-se que o arquivo já foi aberto e que o cabeçalho do arquivo está em memória**

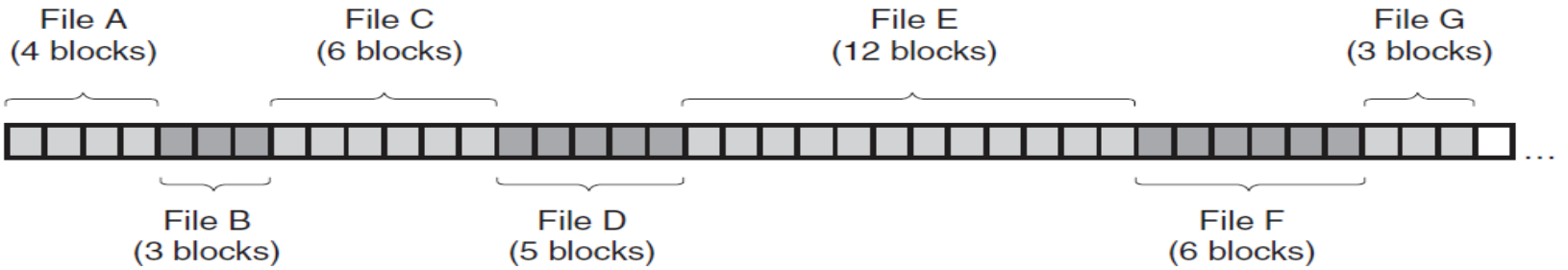
Alocação de blocos na memória secundária

- Leituras, escritas, buscas (DE REGISTROS), etc., são realizadas por blocos.
- Os arquivos não são estáticos, eles crescem e diminuem
- Estratégias de alocação de blocos no disco e organização de registros pelos blocos devem considerar esse fato
  - **Sequencial** não ordenado (heap files)
  - **Sequencial** ordenado (sorted files)
  - Por listas ligadas
  - Indexado
  - Árvores B / B+
  - Hashing
- Para cada estratégia analisaremos a complexidade de leitura sequencial (ler o arquivo do início ao fim), leitura aleatória (busca de um dado registro), inserção e remoção de registros
- Complexidade será definida em termos de número de número de seeks (estimado no pior caso pelo número de blocos a serem lidos); **assume-se que o arquivo já foi aberto e que o cabeçalho do arquivo está em memória**

Alocação sequencial

- Blocos alocados sequencialmente no disco (pelos cilindros)

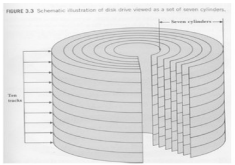


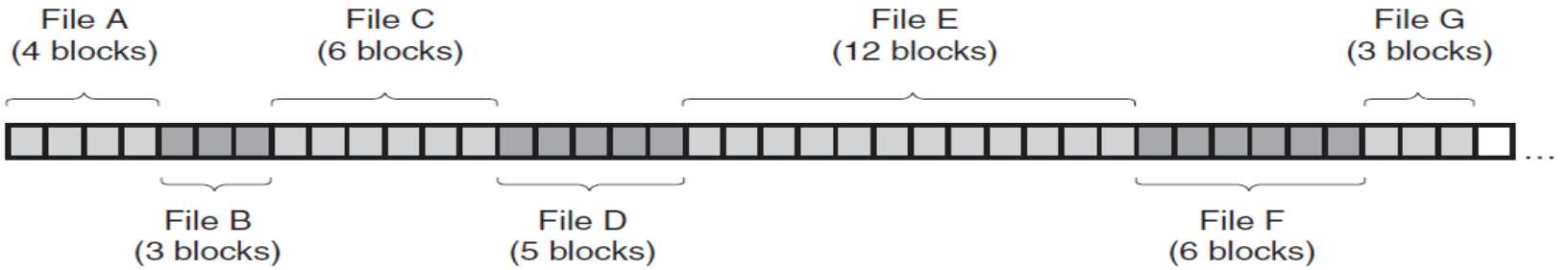


- **Vantagens e desvantagens?**

Alocação sequencial

- Blocos alocados sequencialmente no disco (pelos cilindros)

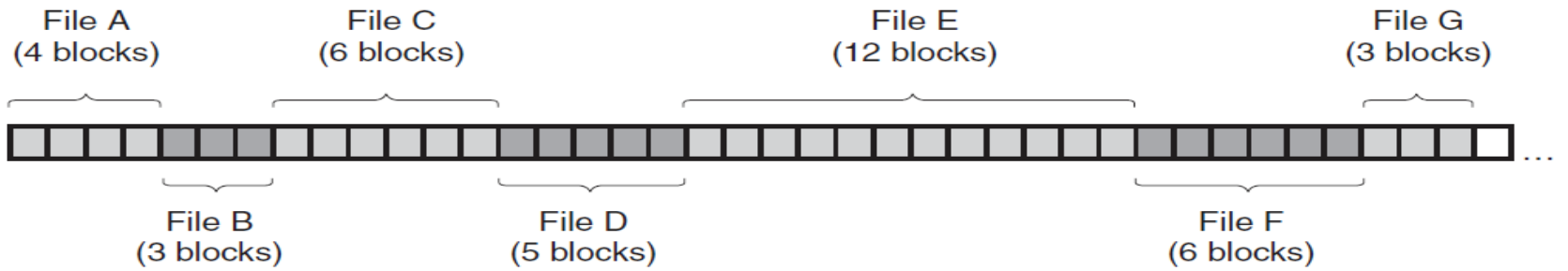




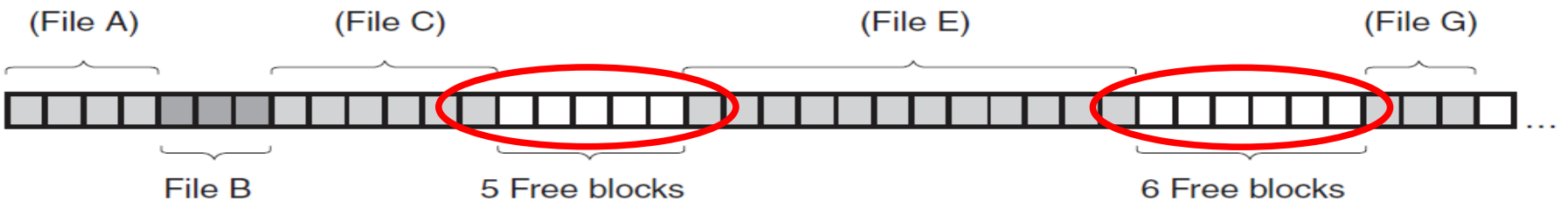
- **Vantagens e desvantagens?**
  - Leitura fácil (leitura sequencial é ótima, e na leitura aleatória depende da facilidade de localização do deslocamento do registro dentro do arquivo)
  - Expansão complicada: se não houver espaço disponível até o próximo arquivo tem que ser removido para outro local
  - Fragmentação externa (buracos entre os arquivos): maior ou menor dependendo da política de alocação

Alocação sequencial – Fragmentação externa

- Blocos alocados sequencialmente no disco (pelos cilindros)



- Após algumas remoções

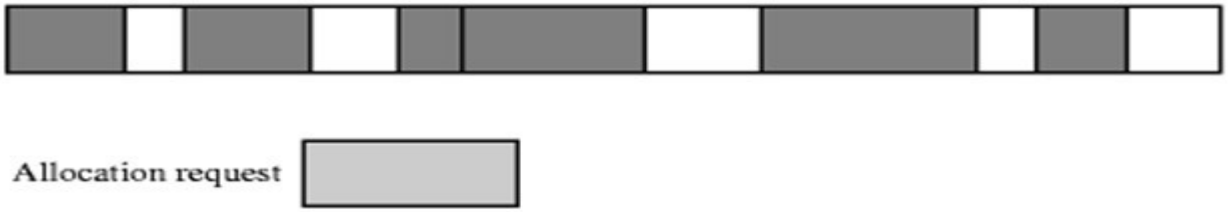


Fonte: (TANEMBAUM, 2015)

Alocação sequencial

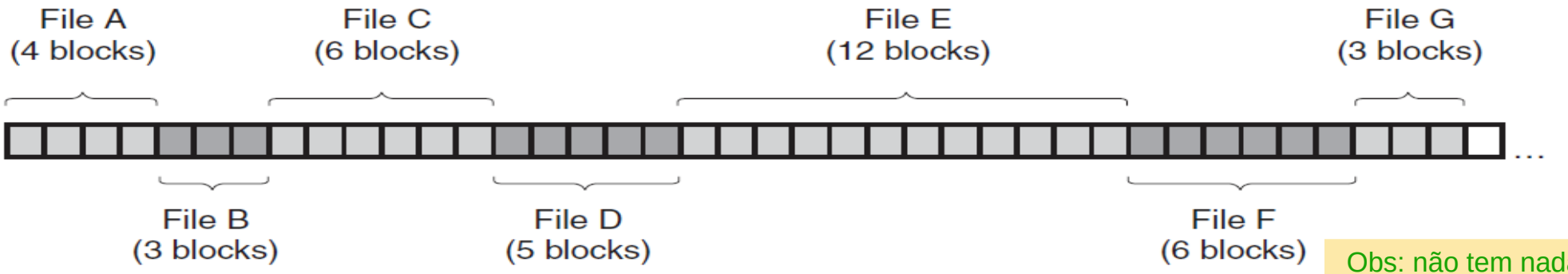
Fragmentação externa:

- Com o tempo (após alocações/desalocações sucessivas), o disco pode ficar fragmentado, isto é, com vários trechos disponíveis intercortados por trechos utilizados



Alocação sequencial

- Blocos alocados sequencialmente no disco (pelos cilindros)

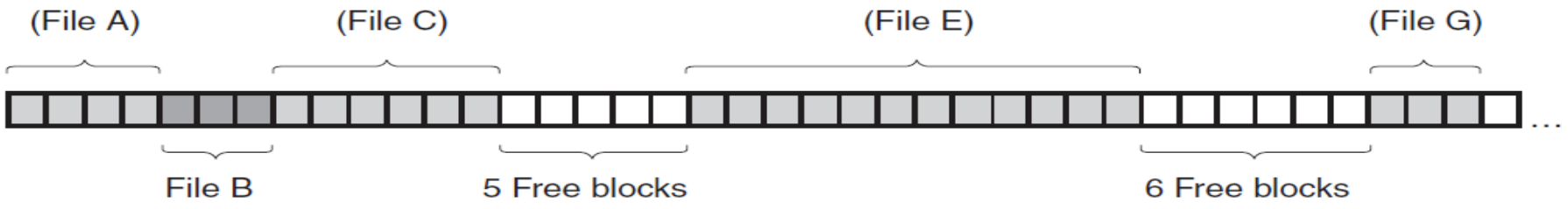


- E como os registros são organizados pelos blocos?
  - Não ordenados (**sequencial não ordenado** – heap files)
  - Ordenados por um campo chave (**sequencial ordenado** - sorted files)

Obs: não tem nada a ver com a estrutura de dados "heap"

Alocação sequencial (não ordenado)

- O arquivo, de r registros espalhados em b blocos, não está ordenado nem indexado



Exercícios em arquivos simples (sem índices)

Considere arquivos de registros de tamanho fixo do tipo *REGISTRO como segue:*

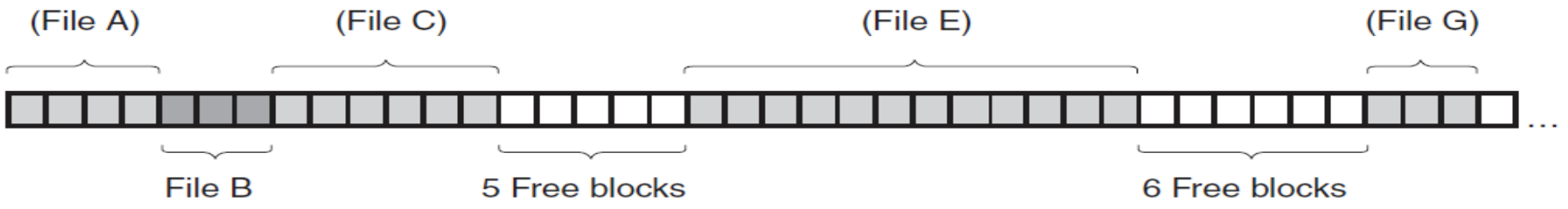
```
typedef struct {
        int NroUSP; // chave primária
        int curso;
        int estado;
        int idade;
        bool valido; // para exclusão lógica
} REGISTRO;
```

Todos os arquivos são mantidos em ordem aleatória de chaves. Para os exercícios a seguir, não há nenhuma estrutura de índices disponível.

1. Reescreva um arquivo arq1 em um novo arquivo arq2 eliminado os registros inválidos.
2. Faça uma cópia invertida de arq1 em um novo arquivo arq2, ou seja: copie o último registro (n) de arq1 para o início de arq2, depois copie o registro n-1 para a segunda posição etc.
3. Escreva uma função para inserir um novo registro r no arquivo, tomando cuidado para evitar chaves duplicadas.
4. Escreva uma função que, dada um nroUSP X, retorne o registro correspondente.
5. Escreva uma função para excluir todos os registros do curso X.
6. Escreva uma função para alterar o curso de um aluno de nroUSP X para o curso Y.
7. Implemente o procedimento de ordenação *KeySort*, que dado um arquivo arq1 cria uma tabela temporária de chaves em memória (idêntica a uma tabela de índices primários) e então reescreve o arquivo em um novo arquivo de saída arq2, na ordem correta de chaves (exercício completo e altamente recomendável).

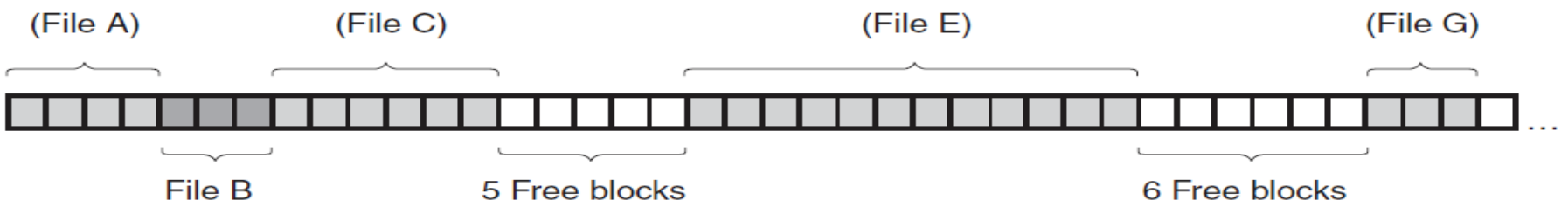
Alocação sequencial (não ordenado)

- Vamos analisar as complexidades de inserção, busca, remoção, modificação, etc, de REGISTROS que estão nestes arquivos



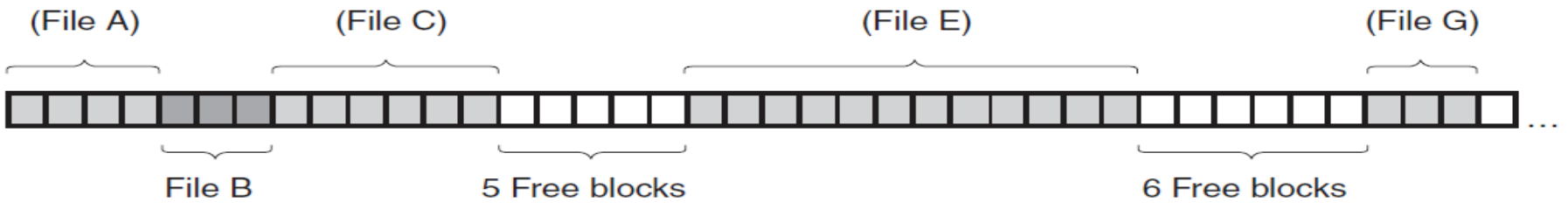
Alocação sequencial (não ordenado)

- O arquivo, de r registros espalhados em b blocos, não está ordenado nem indexado
- **Inserção** : Onde?



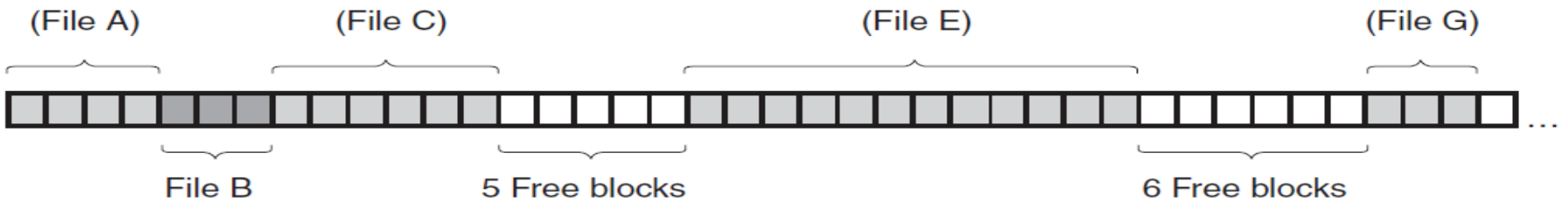
Alocação sequencial (não ordenado)

- O arquivo, de r registros espalhados em b blocos, não está ordenado nem indexado
- **Inserção** : no final do arquivo.



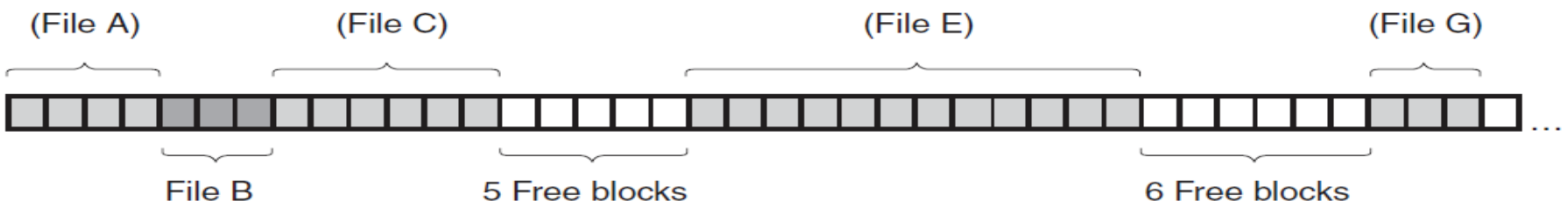
Alocação sequencial (não ordenado)

- O arquivo, de r registros espalhados em b blocos, não está ordenado nem indexado
- **Inserção** : no final do arquivo. Há espaço disponível (dentro do último bloco ou após ele até o próximo arquivo)?
  - SIM: Eficiente: O(1) seeks
    - Copia último bloco (ou próximo) no buffer de memória (localização está no cabeçalho) – 1 seek
    - Insere registro (na memória)
    - Reescreve bloco no disco – 1 seek
  - NÃO: Ineficiente: O(b) seeks – tem que realocar todo o arquivo em outro lugar no disco



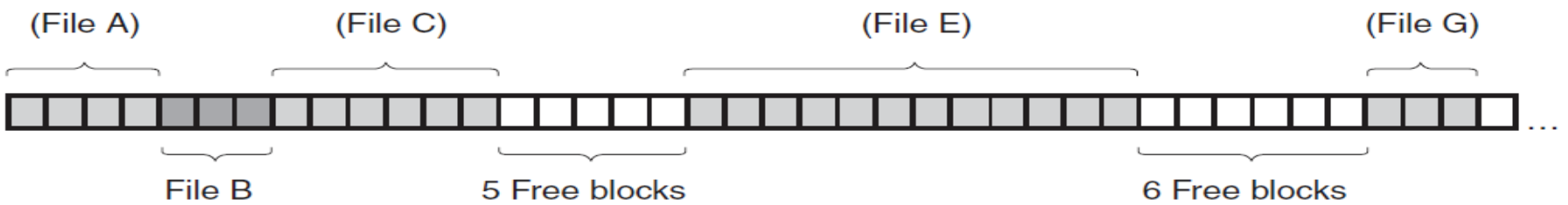
Alocação sequencial (não ordenado)

- **Busca**:



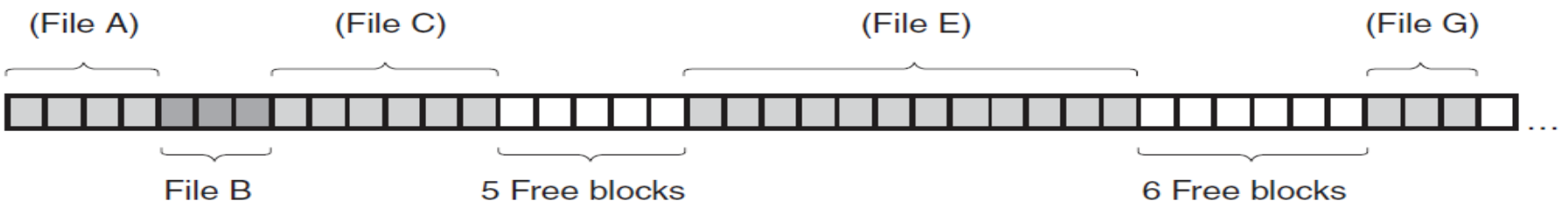
Alocação sequencial (não ordenado)

- **Busca**: sequencial não ordenada
  - Tenho que olhar todos os blocos – O(b)
  - Lembrando que a busca é por registros, não por blocos



Alocação sequencial (não ordenado)

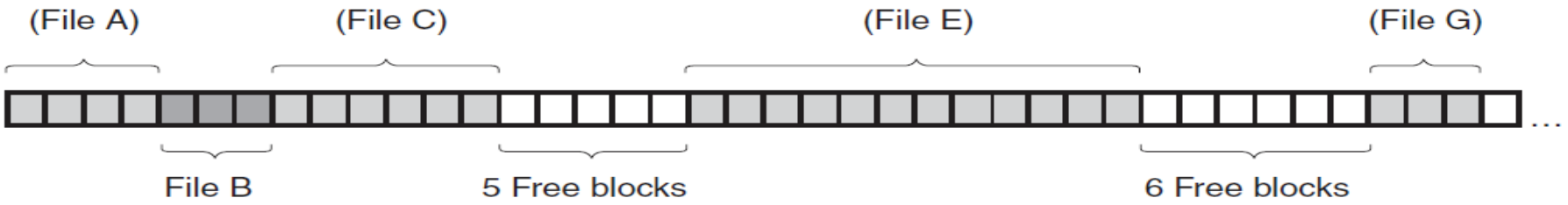
- **Remoção**:



Alocação sequencial (não ordenado)

- **Remoção**:
  - Vamos considerar que já buscamos o bloco que contém o registro
  - Carrega bloco contendo o registro para a memória (1 seek)
  - Exclui registro do bloco (que está no buffer): resetar bit para inválido
  - Reescrever bloco de volta ao disco (com um espaço vazio) – 1 seek

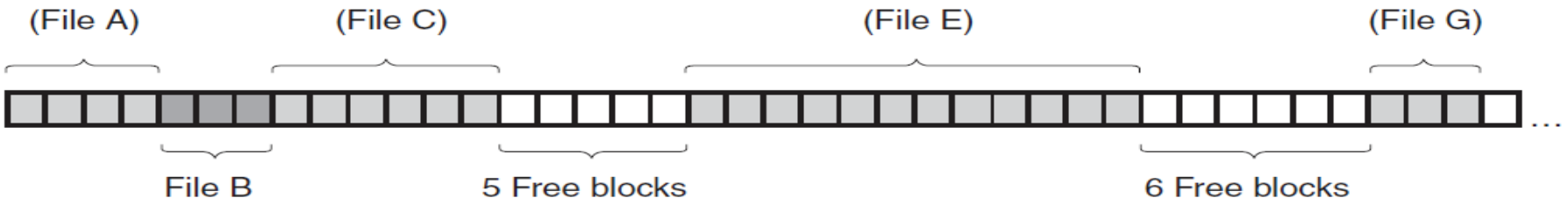
    => Demanda uma reorganização periódica



Alocação sequencial (não ordenado)

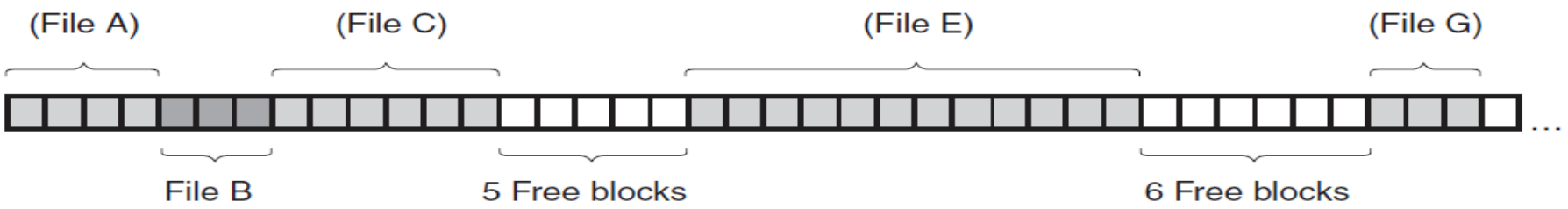
- **Remoção**: Total: O(1)
  - Vamos considerar que já buscamos o bloco que contém o registro
  - Carrega bloco contendo o registro para a memória (1 seek)
  - Exclui registro do bloco (que está no buffer): resetar bit para inválido
  - Reescrever bloco de volta ao disco (com um espaço vazio) – 1 seek

    => Demanda uma reorganização periódica



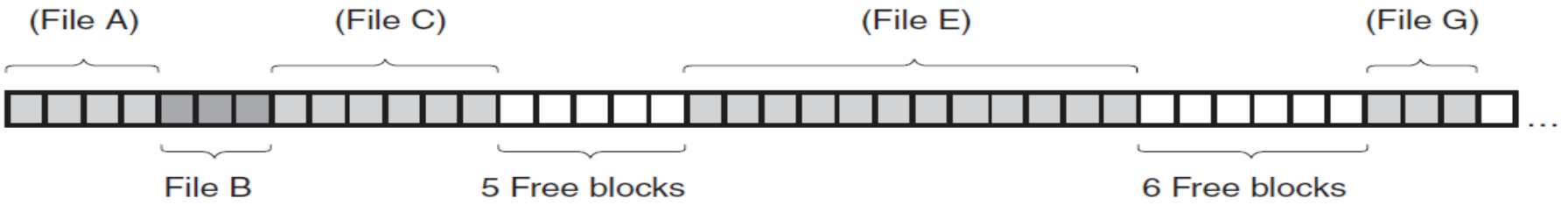
Alocação sequencial (não ordenado)

- **Leitura ordenada**:



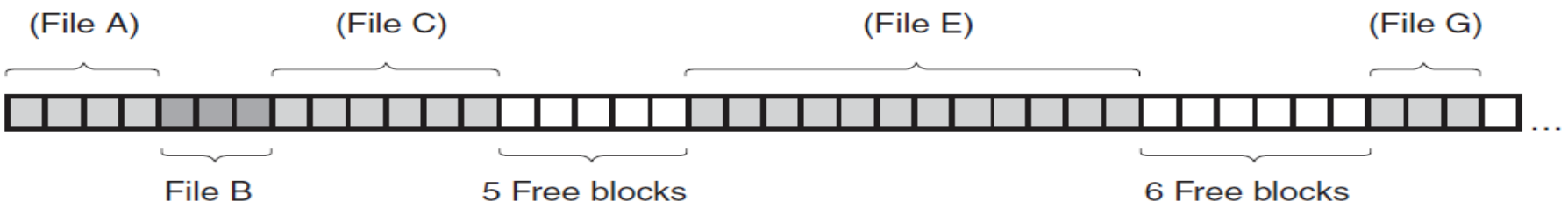
Alocação sequencial (não ordenado)

- **Leitura ordenada**: MUITO INEFICIENTE
  - Exige uma ordenação primeiro! (exigirá ordenação externa se o arquivo inteiro não couber na memória)



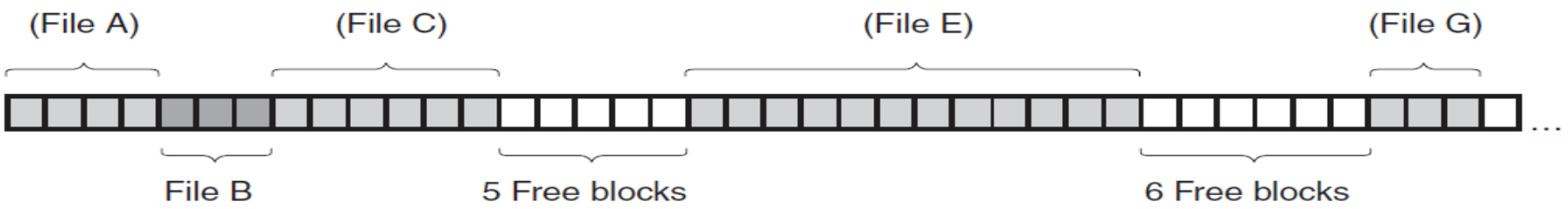
Alocação sequencial (não ordenado)

- **Mínimo** (menor chave) **/ Máximo** (maior chave)**:**



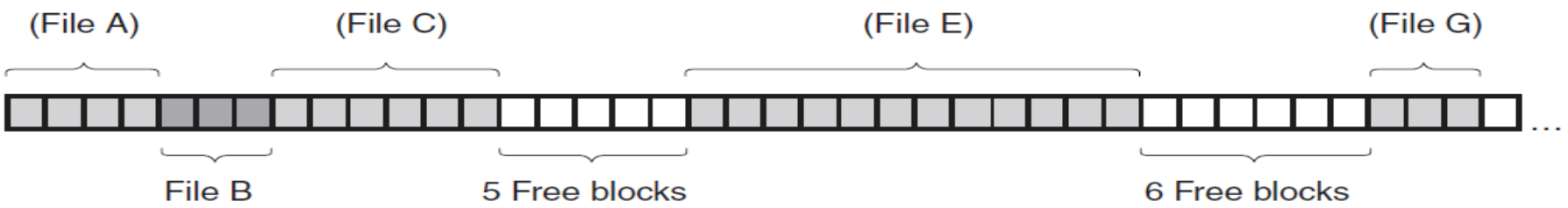
Alocação sequencial (não ordenado)

- **Mínimo** (menor chave) **/ Máximo** (maior chave)**:** O(b)



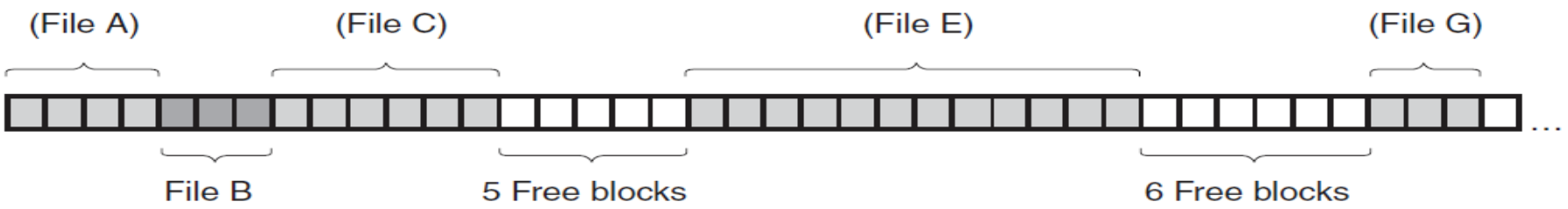
Alocação sequencial (não ordenado)

- **Modificação** de um campo:



Alocação sequencial (não ordenado)

- **Modificação** de um campo: (assumindo que já buscou)
  - O(1) em qualquer campo



Alocação sequencial (não ordenado)

	Sequencial		
	Não-Ordenado		
Busca	O(b)		
Inserção**	O(1) se tiver espaço no final; O(b) c.c.		
Remoção* , **	O(1)		
Leitura ordenada	ω(b) (depende do alg de ord. externa)		
Mínimo/máximo	O(b)		
Modificação**	O(1)		
* considerando uso de bit de validade			
** considerando que já se sabe a localização do registro (busca já realizada)			

Exercícios em arquivos simples (sem índices)

Considere arquivos de registros de tamanho fixo do tipo *REGISTRO como segue:*

```
typedef struct {
        int NroUSP; // chave primária
        int curso;
        int estado;
        int idade;
        bool valido; // para exclusão lógica
} REGISTRO;
```

Exercício: como seria a complexidade para estas operações?

Todos os arquivos são mantidos em ordem aleatória de chaves. Para os exercícios a seguir, não há nenhuma estrutura de índices disponível.

1. Reescreva um arquivo arq1 em um novo arquivo arq2 eliminado os registros inválidos.
2. Faça uma cópia invertida de arq1 em um novo arquivo arq2, ou seja: copie o último registro (n) de arq1 para o início de arq2, depois copie o registro n-1 para a segunda posição etc.
3. Escreva uma função para inserir um novo registro r no arquivo, tomando cuidado para evitar chaves duplicadas.
4. Escreva uma função que, dada um nroUSP X, retorne o registro correspondente.
5. Escreva uma função para excluir todos os registros do curso X.
6. Escreva uma função para alterar o curso de um aluno de nroUSP X para o curso Y.
7. Implemente o procedimento de ordenação *KeySort*, que dado um arquivo arq1 cria uma tabela temporária de chaves em memória (idêntica a uma tabela de índices primários) e então reescreve o arquivo em um novo arquivo de saída arq2, na ordem correta de chaves (exercício completo e altamente recomendável).

Referências

- ELMARIS, 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 & GEHRKE. **Data Management Systems**. 3ª ed. McGrawHill. 2003. Cap 8 e 9.
- TANEMBAUM, A. S. & BOS, H. **Modern Operating Systems**. Pearson, 4th ed. 2015