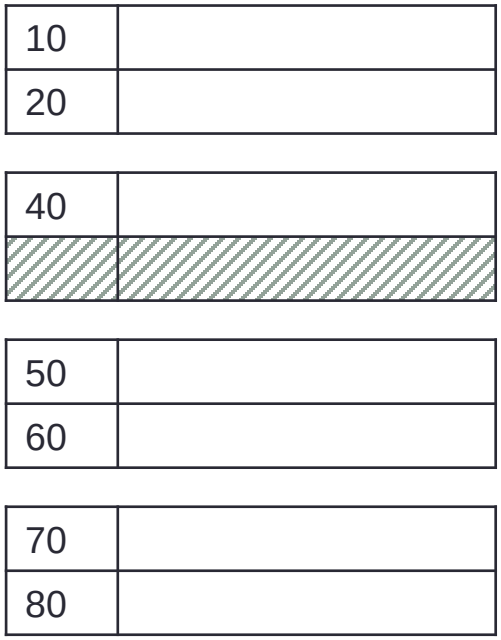
Atividade Individual

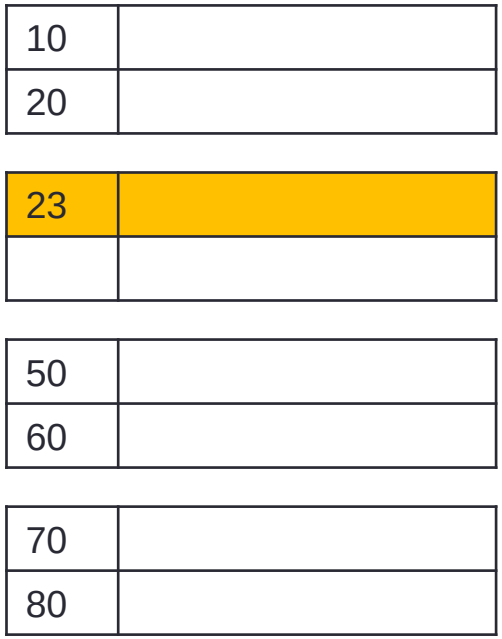
- O pacote ibd.table implementa um gerenciador de registros simples
  - Os registros são armazenados ordenados em páginas(blocos) de tamanho fixo.
  - As regras para inserção são parecidas com as vistas nos slides de aula
  - Exceção: quando o bloco está cheio, é sempre criado um bloco de estouro
    - Ou seja, não se tenta distribuir para os vizinhos
    - Além disso, a quantidade de registros que serão movidos para o bloco de estouro é parametrizável

Atividade Individual

- Exemplo 1: Inserção de registro com chave de busca 23
  - Existe espaço no bloco



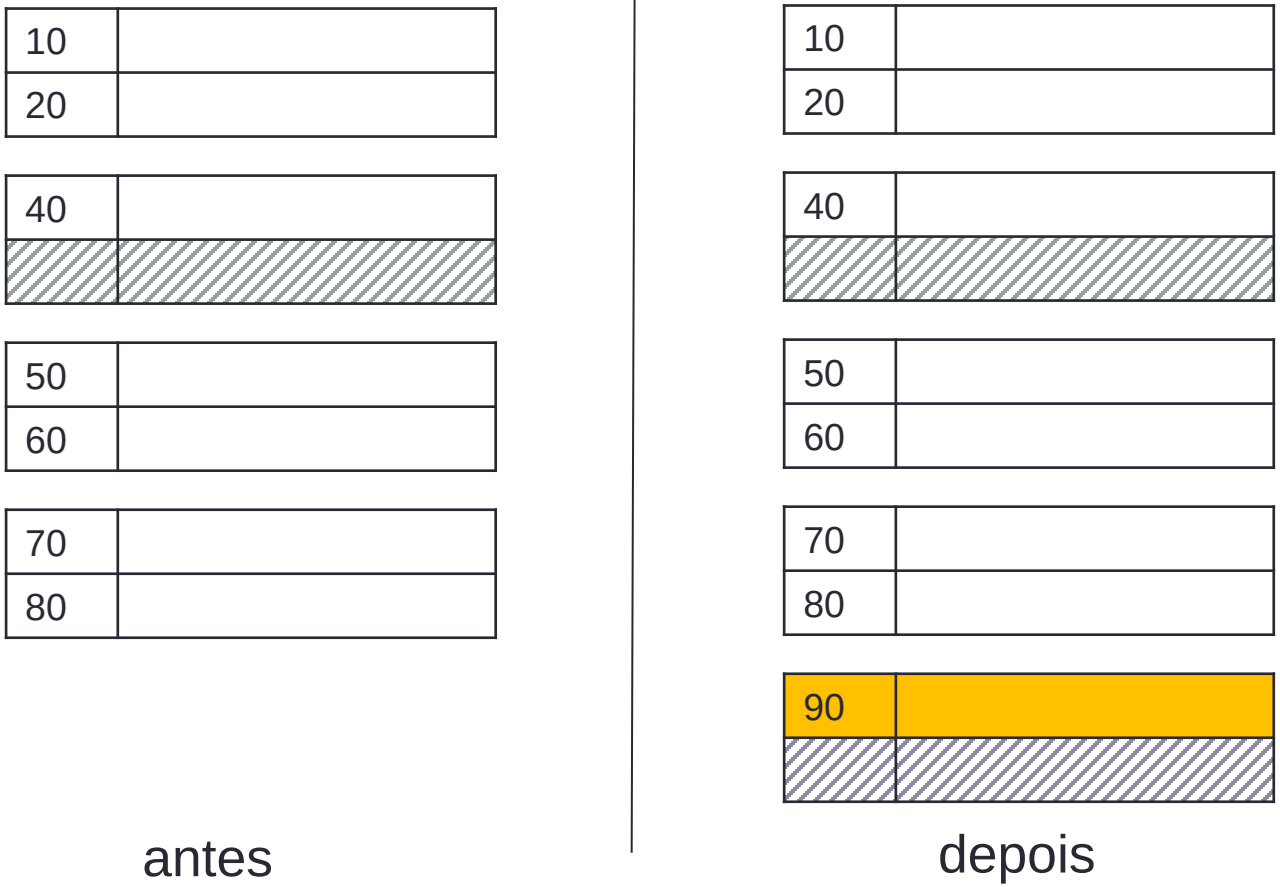
antes



depois

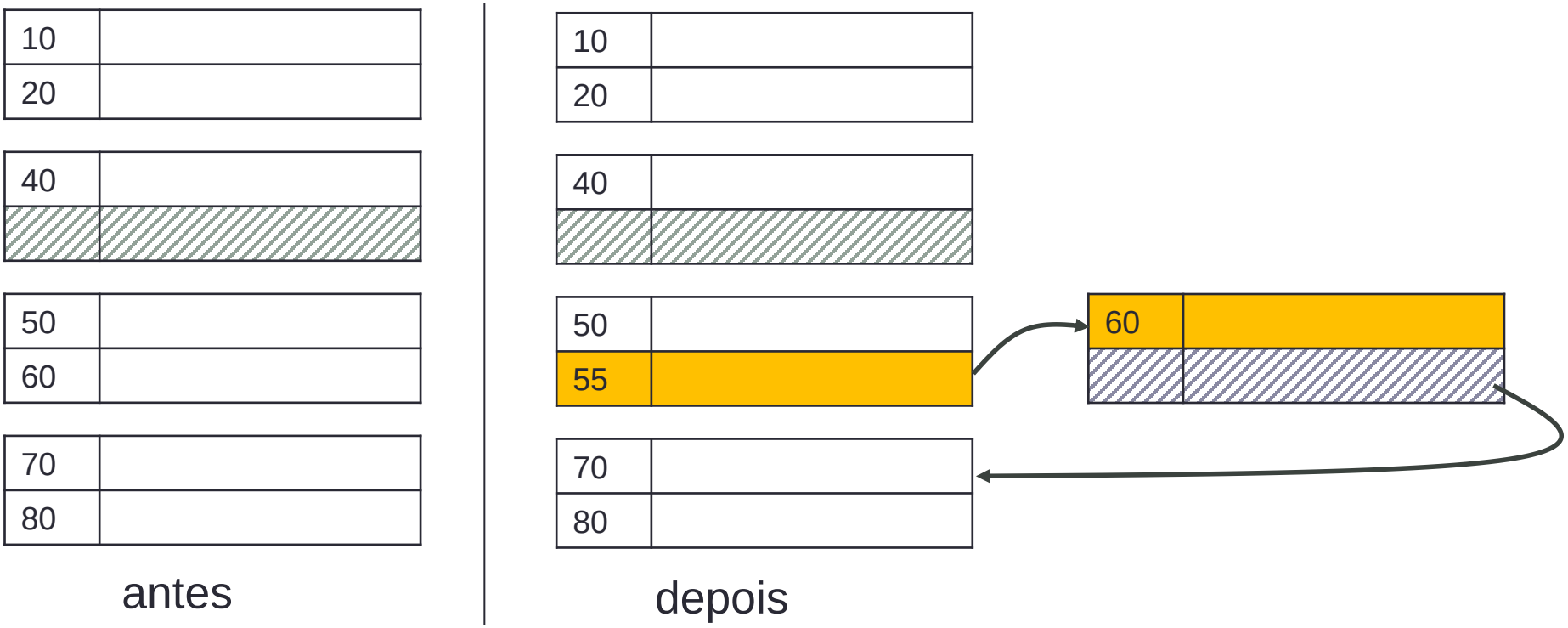
Atividade Individual

- Exemplo 2: Inserção de registro com chave de busca 90
  - Tem espaço no final



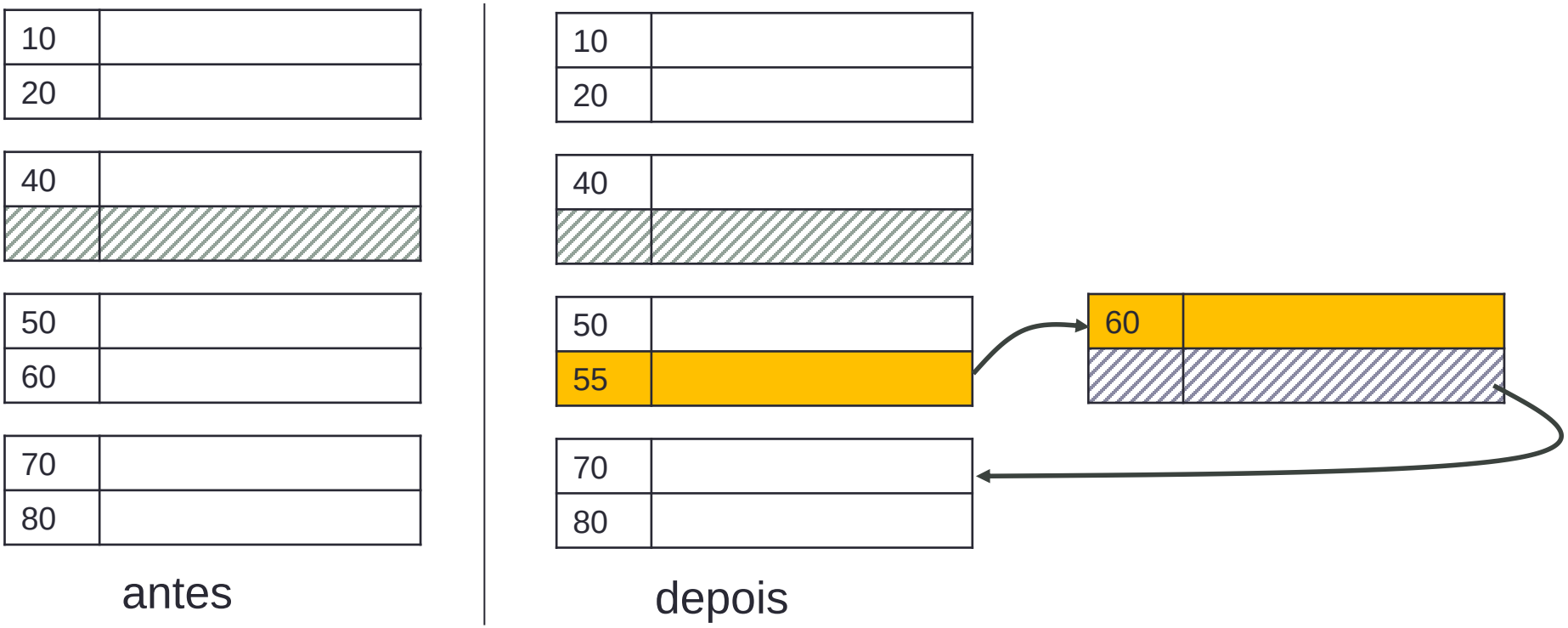
Atividade Individual

- Exemplo 3: Inserção de registro com chave de busca 55
  - Não há espaço no bloco
    - Criado bloco de estouro
    - Um registro é movido para esse bloco



Atividade Individual

- Problema dos blocos de estouro
  - Aumenta o custo no acesso sequencial, pois os blocos ordenados deixam de estar contínuos no disco



Atividade Individual

- A classe Table.java permite definir o splitFactor
  - Esse parâmetro indica quantos registros permanecem no bloco original. O restante é movido para o bloco de estouro
    - Quanto mais próximo de zero, menos registros permanecem
    - Quanto mais próximo de um, mais registros permanecem

- Ex.

```
table.createTable(256, 32, 0.5);
```

  - Parâmetros
    - blockSize: tamanho do bloco em bytes
    - recordSize: tamanho de cada registro
    - splitFactor: percentual dos registros que permanecem no bloco original

  - No exemplo, 50% dos registros permanecem no bloco original

Atividade Individual

- A classe ibd.table.Main realiza testes que medem
  - a quantidade de vezes que páginas foram escritas
  - A quantidade total de páginas

```
System.out.println("blocks write: " + BLOCKS_WRITE);

System.out.println("blocks count"+table.manager.getBlocksCount());
```

Atividade Individual

- O método main() realiza duas rodadas de inserção
  - Primeiro, registros com ids incrementais
  - Segundo, registros com ids aleatórios.

- Pode-se criar cenários de teste variando a quantidade de registros de cada tipo

```
gen.generate(0, 10000, 200, 800);
```

- A função acima gerará 1000 registros
  - Os ids variam de 0 a 10000
  - As primeiras 200 inserções serão ordenadas
  - As próximas 800 serão foram de ordem

Atividade Individual

- O objetivo da atividade é analisar como o splitFactor se comporta, variando os cenários de teste
- Quando é melhor usar um fator mais alto, ou mais baixo?
  - Que valor é indicado quando as inserções são predominantemente ordenadas?
  - E se foram desordenadas?
  - E se inicialmente forem ordenadas e então passam a ser aleatórias?
- Escreva um relatório em PDF reportando as descobertas.