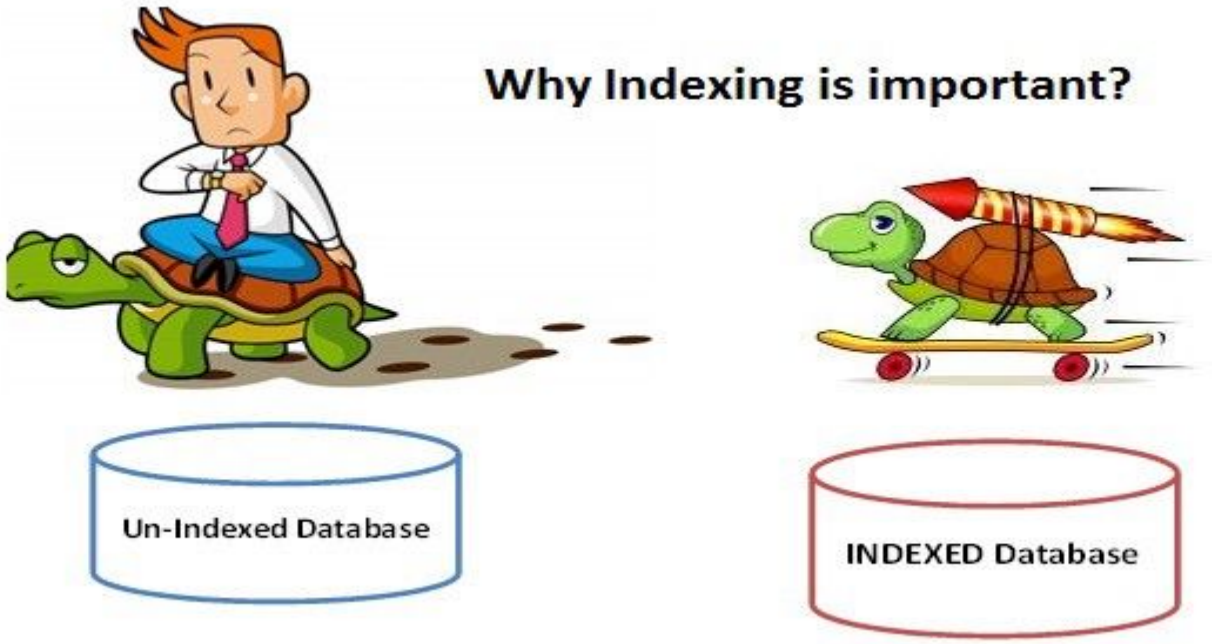
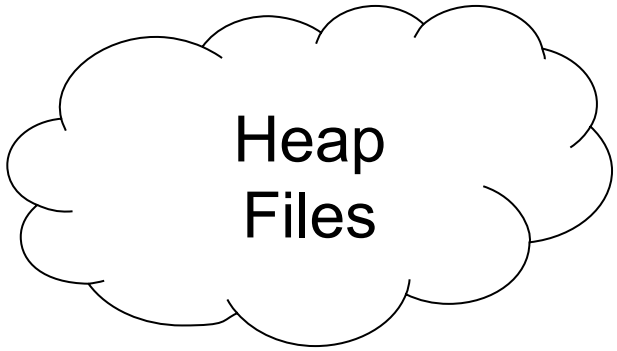
Índices


Why Indexing is important?
Un-Indexed Database
INDEXED Database

Heap files

Dados sem organização

Paulo, 44, 2000
Pedro, 35, 20000
Carlos, 44, 2000
José, 40, 2500
João, 35, 3000
Ilmério, 40, 3500
Rodrigo, 40, 3500
Maria, 30, 4000
Sara, 35, 4000
Sabrina, 31, 5000

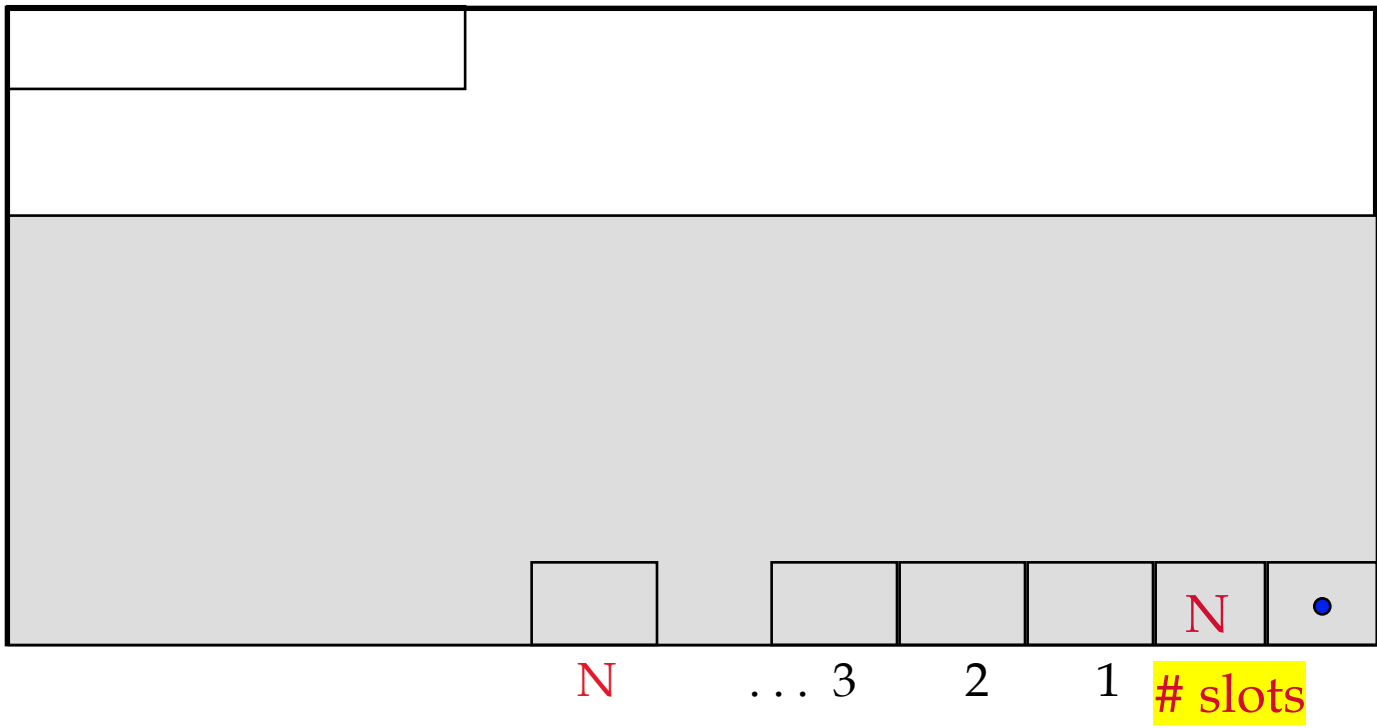


- Bons para inclusão
- Consultas: necessário fazer **scan** (varredura) na tabela

Exemplo: Heap Files

Dados sem organização

Paulo, 44, 2000
Pedro, 35, 20000
Carlos, 44, 2000



Sorted files

Dados Organizados (nome)

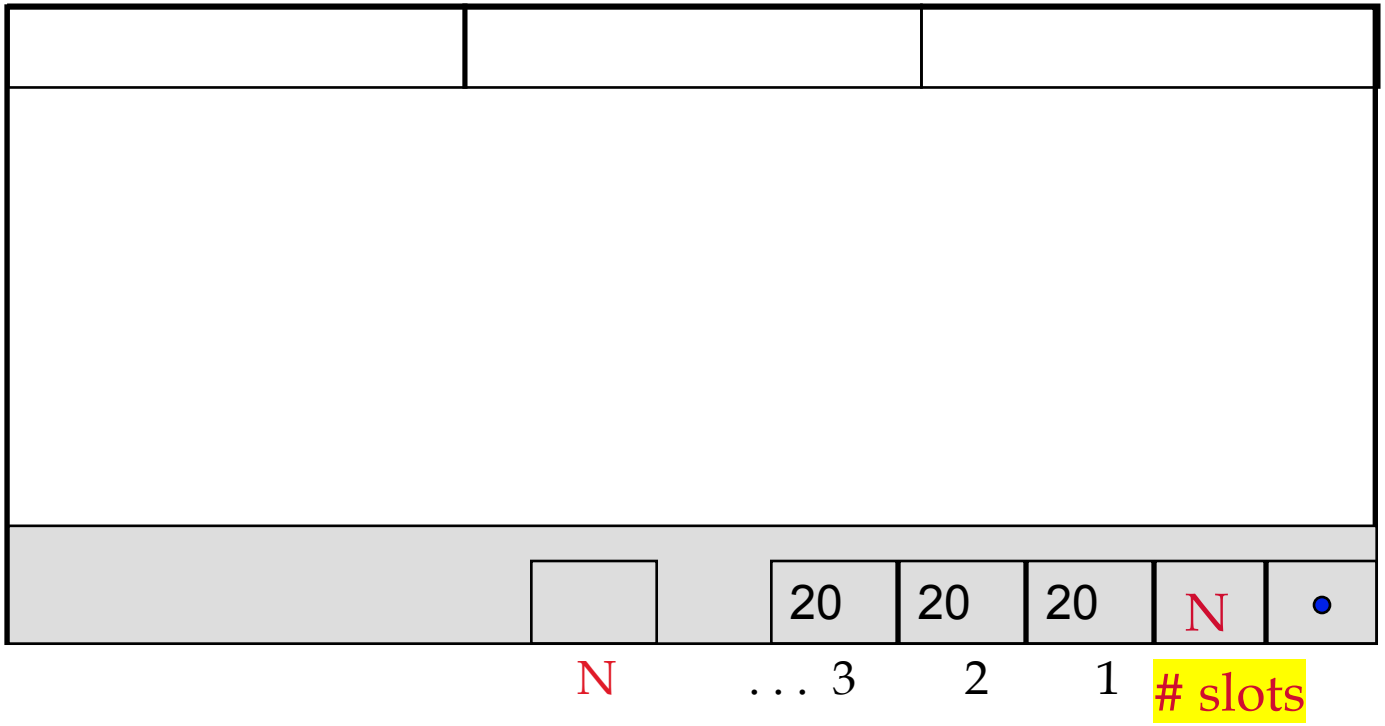
Carlos, 44, 2000
Ilmério, 40, 3500
João, 35, 3000
José, 40, 2500
Maria, 30, 4000
Paulo, 44, 2000
Pedro, 35, 2000
Rodrigo, 40, 3500
Sabrina, 31, 5000
Sara, 35, 4000

- Ruins para inclusão: difícil manter, “buracos” no arquivos de dados
- Consultas: eficientes para busca da chave

Exemplo: Sort files

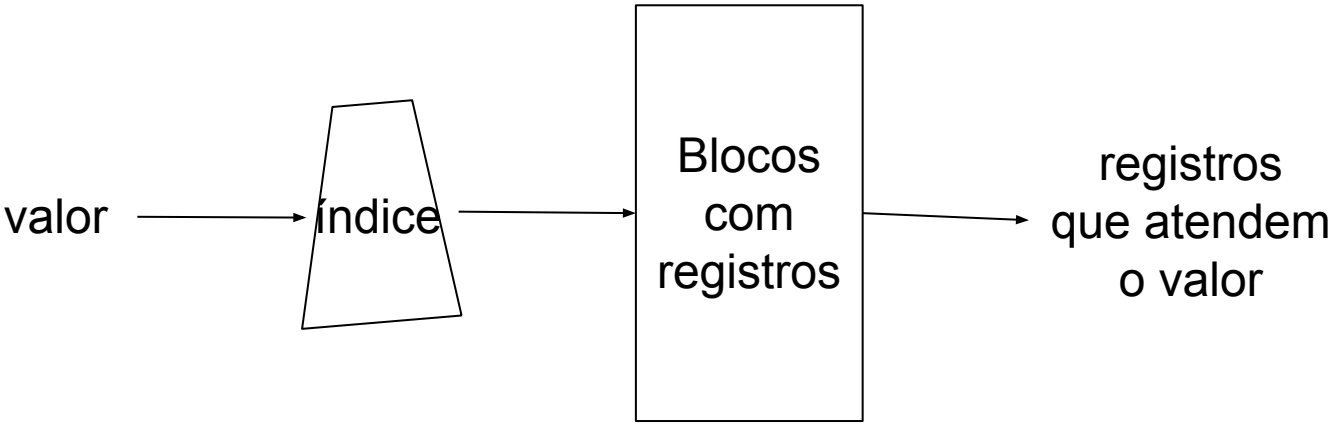
Dados sem organização

Paulo, 44, 2000
Pedro, 35, 20000
Carlos, 20, 2000



Índices

- Estrutura auxiliar projetada para agilizar operações de busca, inserção e supressão



- Alteração nos dados pode levar à alteração no índice
- Espaço extra de armazenamento

Índices

- O que armazenar em cada entrada do arquivo de índice
  - **Alternativa** 1 -> Entrada = Registro inteiro
  - **Alternativa** 2 -> Entrada = chave, rid
  - **Alternativa** 3 -> Entrada = chave, conjunto de rids

- Essa escolha é ortogonal ao índice;
- Um arquivo pode ter diferentes índices

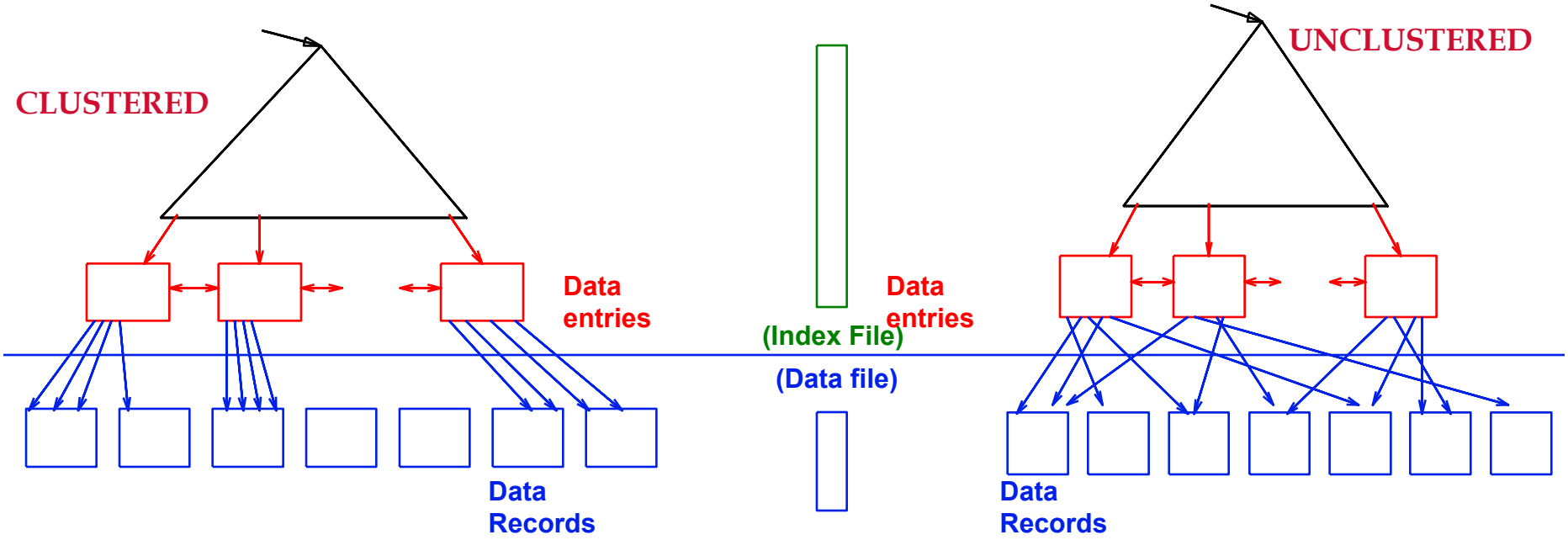
Classificação dos índices

- *Clustered (agrupado)*: se a ordem dos registros são os mesmos da ordem do índice de dados de entrada.
- Unclustered (desagrupado): os registros são armazenadas sem um ordenamento

Um arquivo pode ser clusterizado somente por uma chave

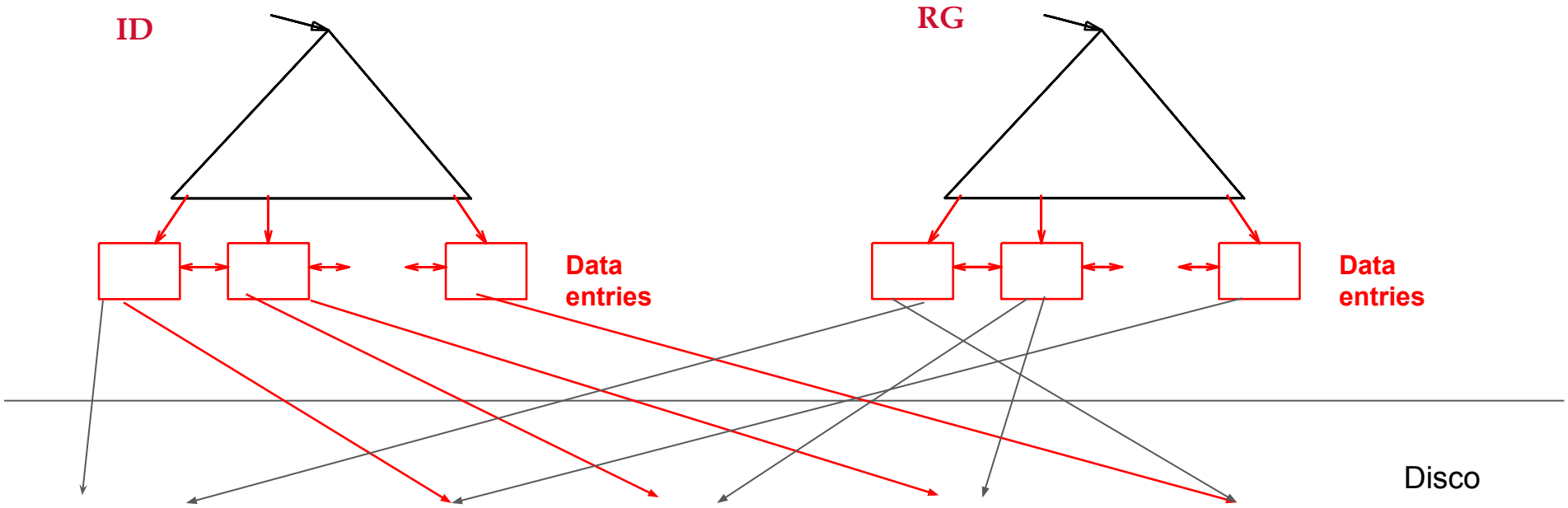
Clustered vs. Unclustered Index

- Suponha que a Alternativa (2) é usado como entrada de dados
- Ex: tenho 1000 rgs diferentes→ até 1000 consultas no disco



Clustered vs. Unclustered Index

CREATE TABLE dados (id int primary key, RG varchar(10), desc char[4000])
CREATE INDEX ON dados ((rg));



Clustered

- Clustered Pros

- Clustered Contras

Clustered

- Clustered Pros
  - Eficiente em buscas entre faixas (> <)
  - Possível benefício na localidade dos dados

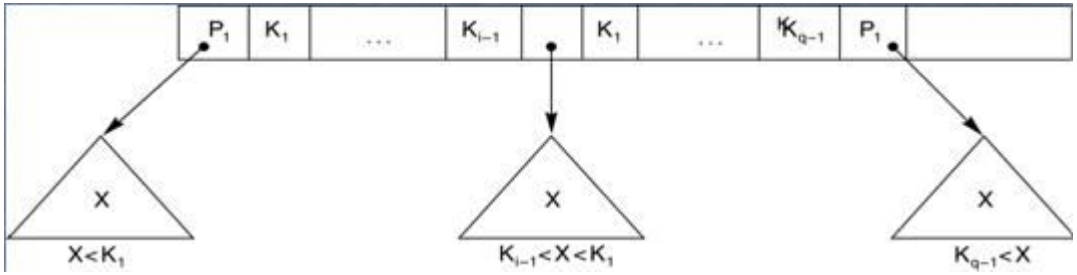
- Clustered Contras
  - Caro para manter
  - Espaço extra de armazenamento

Índices

- Primário
  - A chave do índice é composta pela chave primária da tabela
    - A maioria dos SGB cria índices primários automaticamente
  - Não permitem duplicatas
- Secundário
  - Outras colunas da tabela participam
  - Permitem duplicatas

Índices

- Os índices de múltiplos níveis podem formar uma árvore



- Atualização nos dados, implica na atualização em todos os níveis

Índices B+

- Os SGBs implementam índices multiníveis através de árvores B+
  - Atualização dos níveis mais eficientes
  - Cada nível elimina vários acessos
  - O grau (ou ordem) da árvore indica o número de acessos

Índices B+

-Insert/delete com custo de $\log_{Ordem}(n)$

-Mínimo de 50% de ocupação (exceto nó root).

-Exemplo:

Ordem 5-> número mínimo de chaves = (n-1)/2

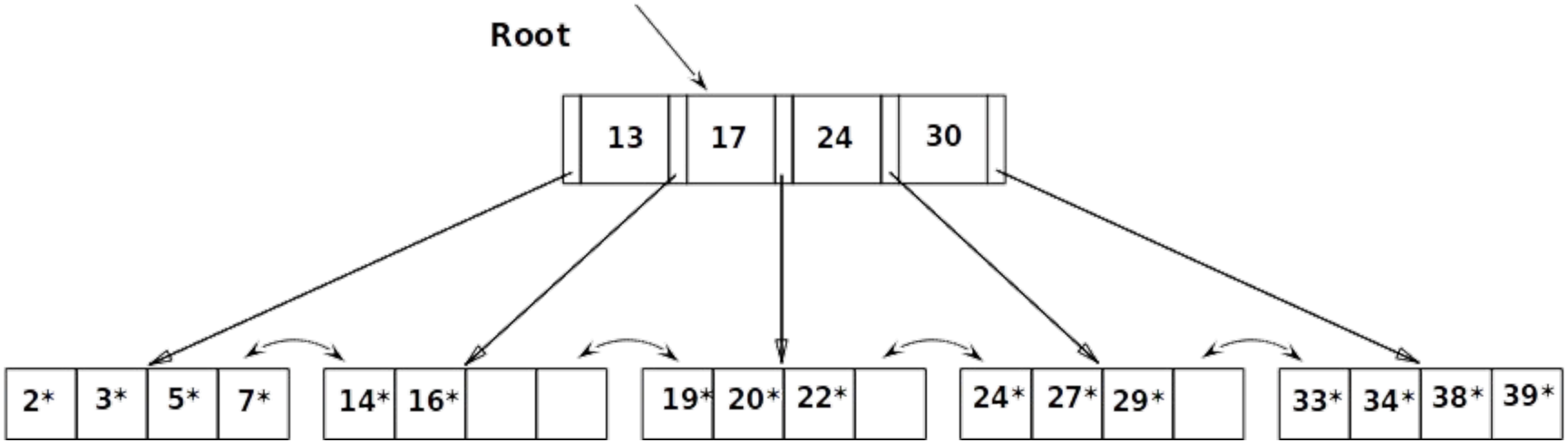
número máximo de chaves = 4

-Suporte de seleção por igualdade e por range.

-Busca sempre termina no nó folha

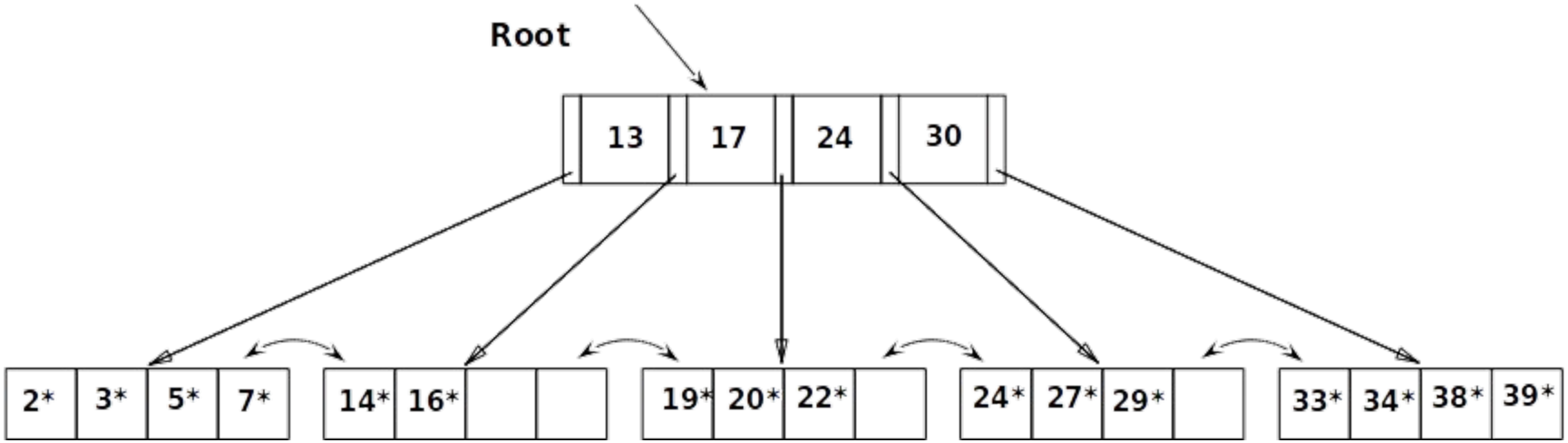
Índices B+

Ordem 5



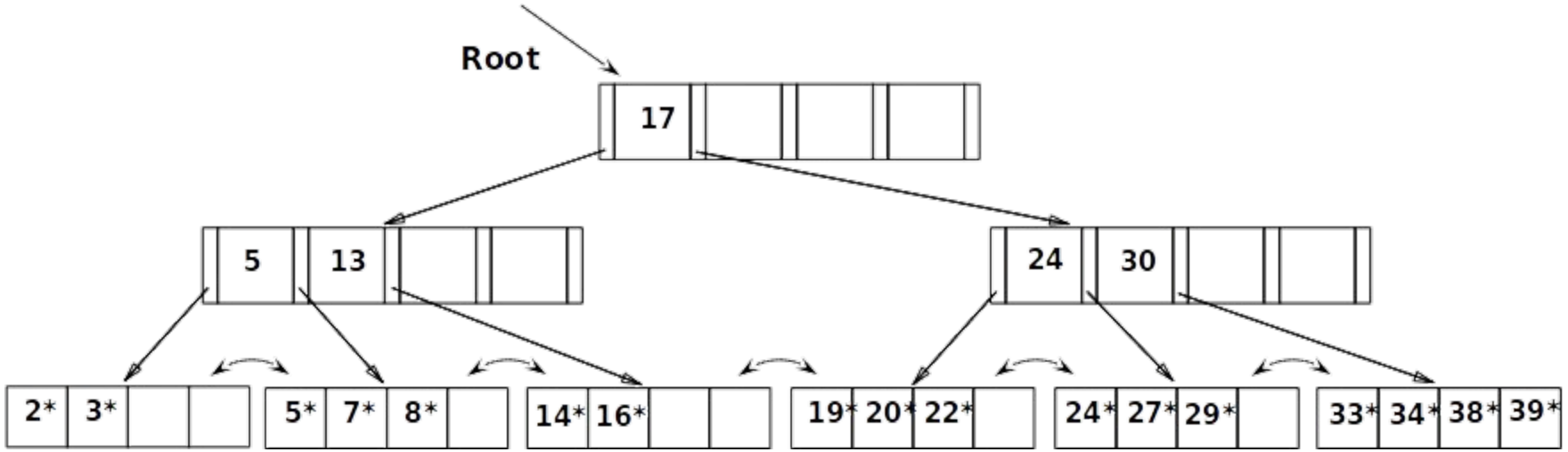
Índices B+

Inserte valor 8*



Índices B+

Deletar a entrada 19*



Índices B+

Use o simulador (http://www.cs.usfca.edu/~galles/visualization/BPlusTree.html)

1- Inserir em um B+ de ordem 5 os seguintes valores: 10, 15, 20,25,30,35,40,45,50

2 -Remover os seguintes valores: 35 30 25 20

Índices B+

- Insert/delete com custo de $\log_{Ordem}(n)$
- Mínimo de 50% de ocupação (exceto nó root).
- Exemplo:

Ordem 5-> número mínimo de chaves = (n-1)/2

número máximo de chaves = 4

- Suporte de seleção por igualdade e por range.
- Busca sempre termina no nó folha

Índices B+

Qual a altura máxima de uma árvore B+ com n chaves?

- A altura da árvore dará o limite máximo de acesso ao disco
  A pior altura é

$$h = \log_{\frac{ordem}{2}} n$$

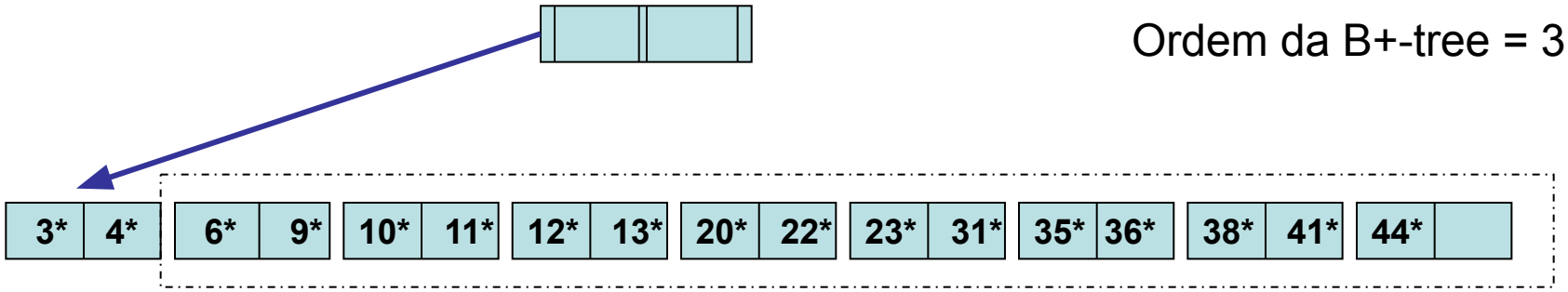
ordem=500
n=1000000
h=log $_{250}$ 1000000

Material Extra

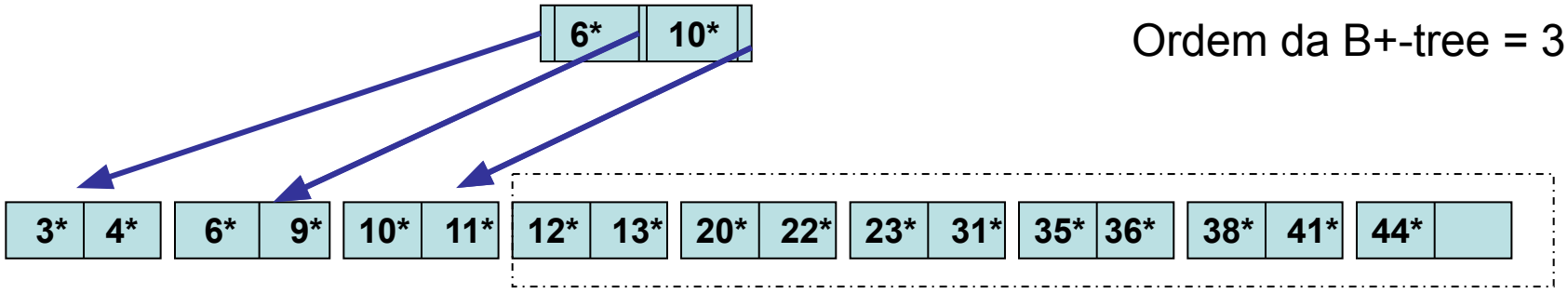
Bulking Load (Construção da B+ Tree)

- Utilizando o arquivo de índice denso (ou arquivo ordenado)
- Aloca-se uma página vazia para a raiz
- Insere nesta página um ponteiro para a primeira página do arquivo contendo as entradas.

Bulking Load (Construção da B+ Tree)



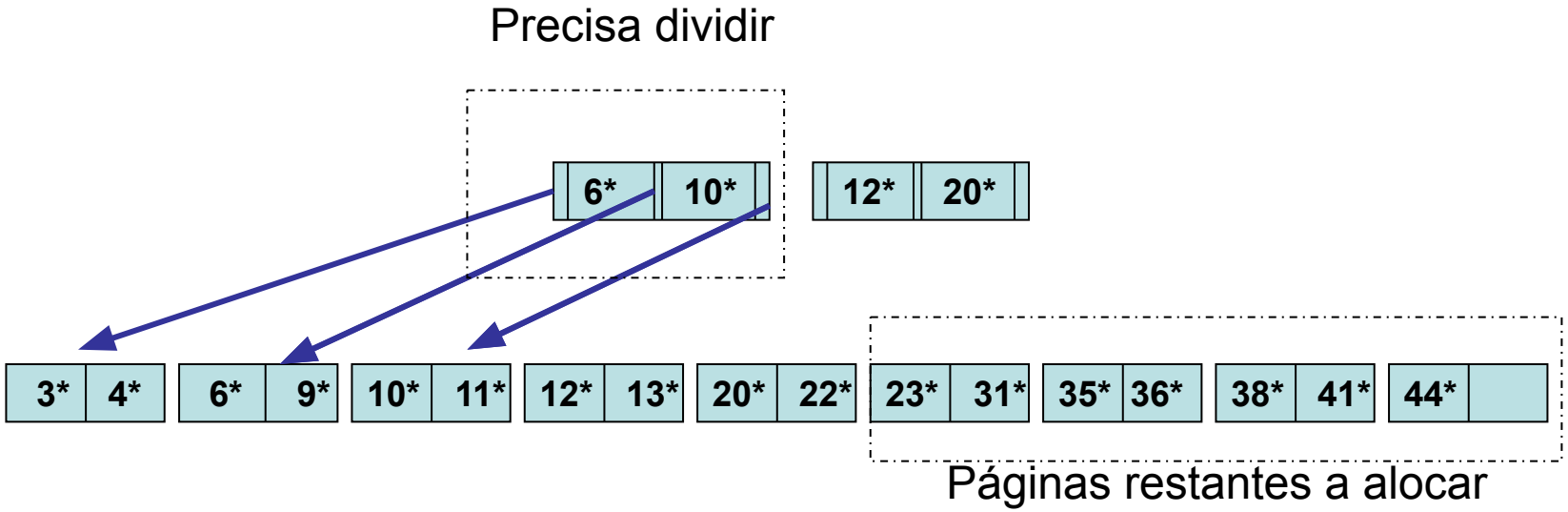
Bulking Load (Construção da B+ Tree)



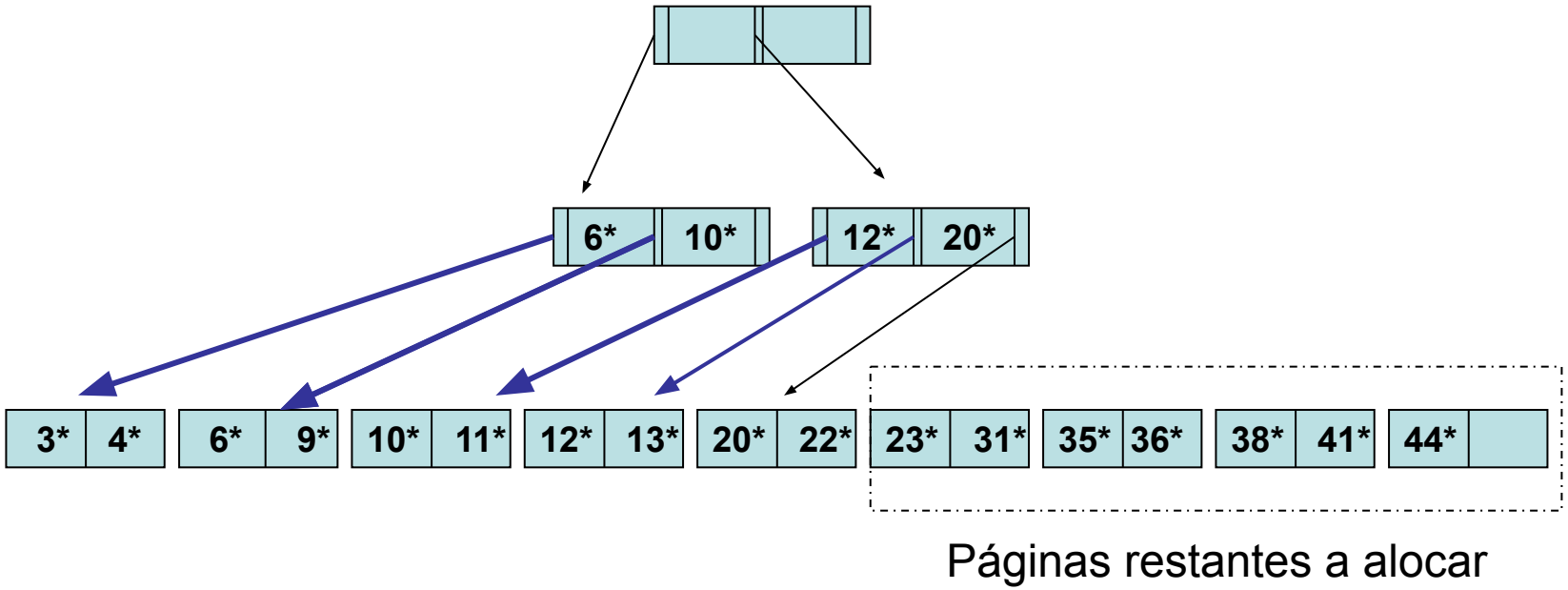
Ordem da B+-tree = 3

Páginas restantes a alocar

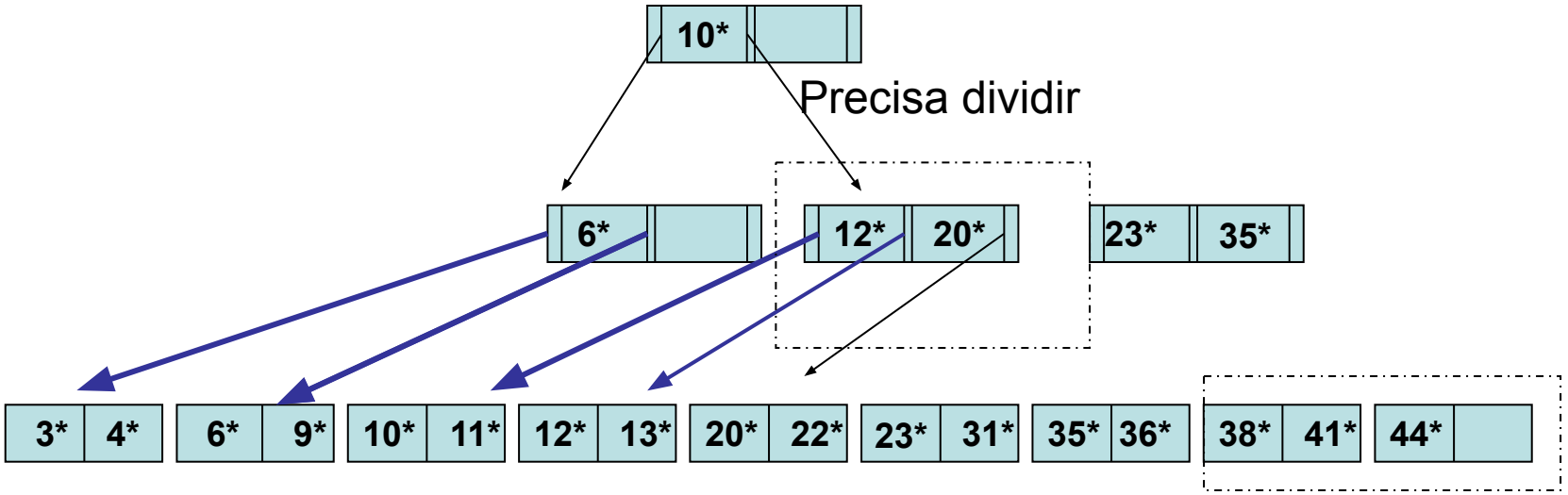
Bulking Load (Construção da B+ Tree)



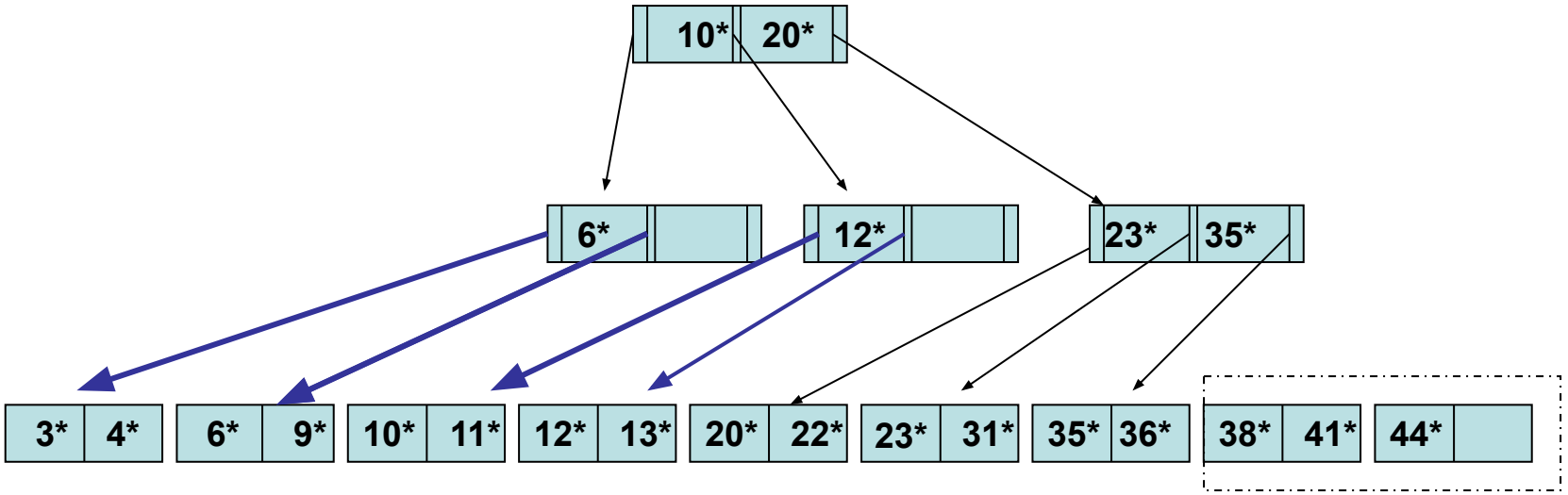
Bulking Load (Construção da B+ Tree)



Bulking Load (Construção da B+ Tree)

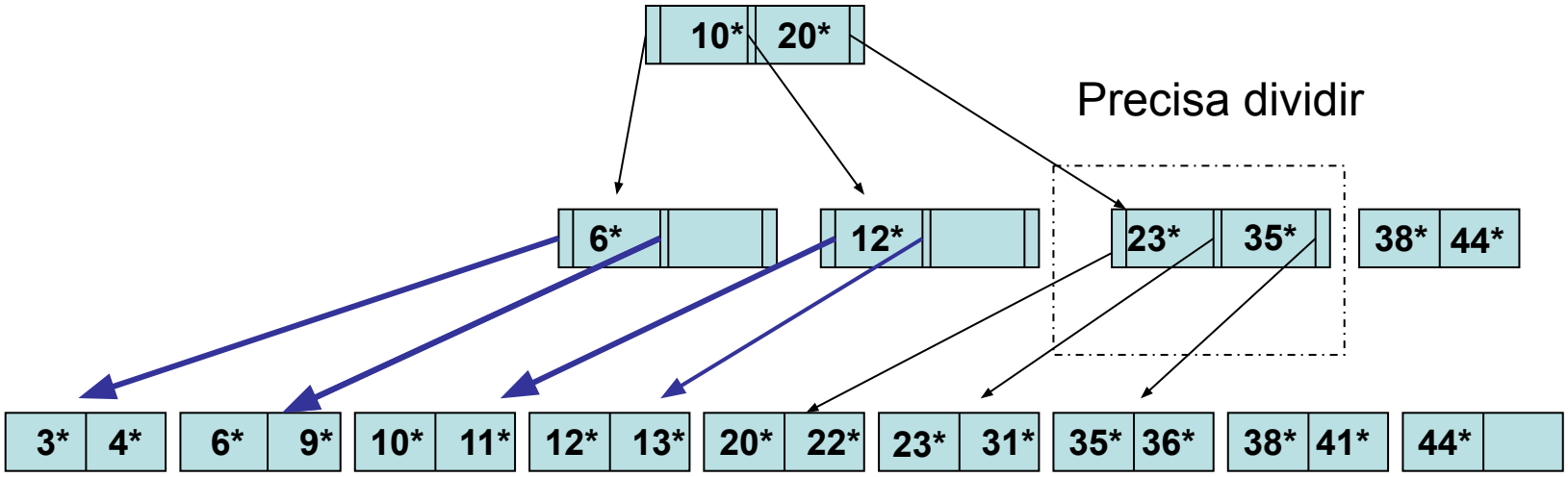


Bulking Load (Construção da B+ Tree)

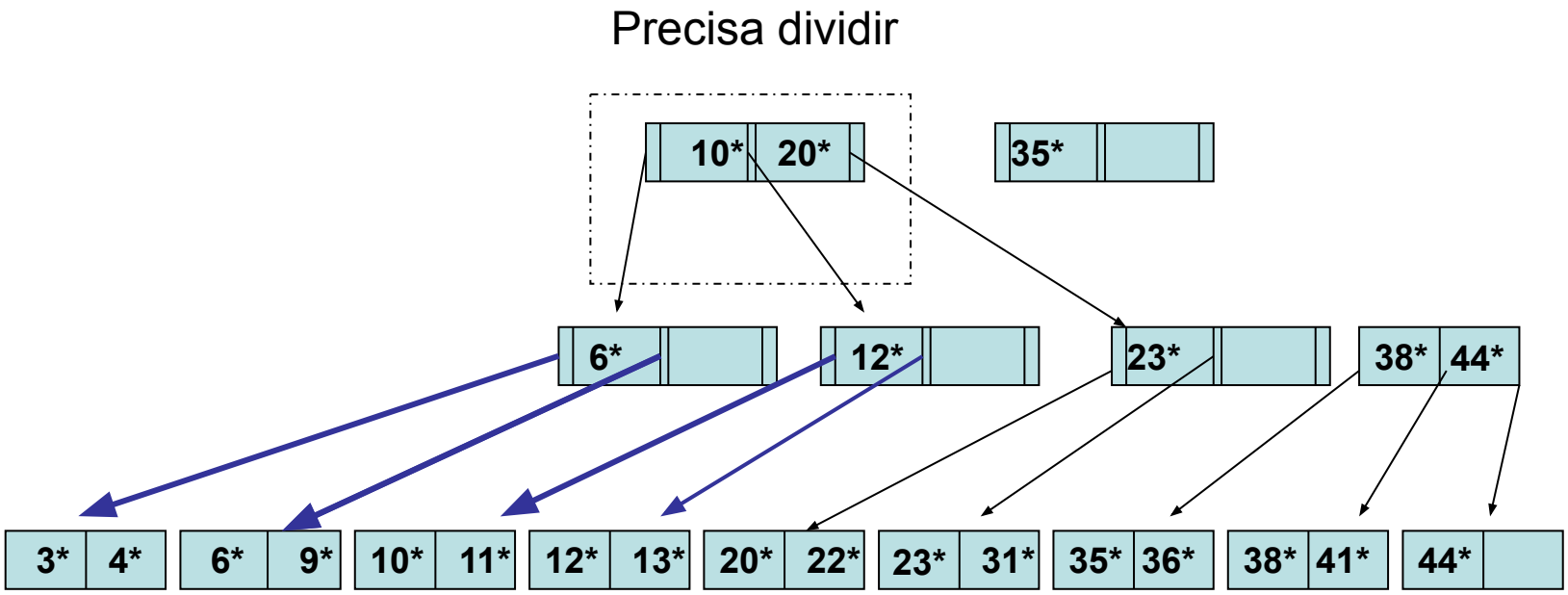


Páginas restantes a alocar

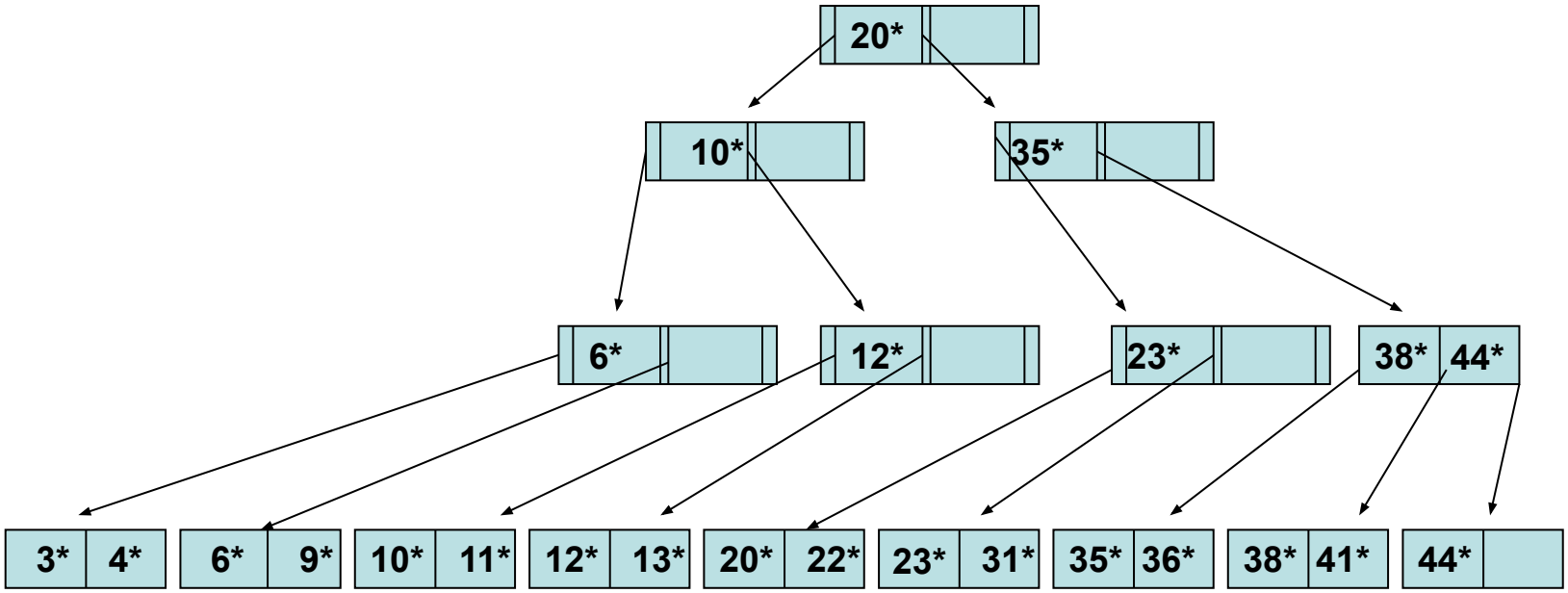
Bulking Load (Construção da B+ Tree)



Bulking Load (Construção da B+ Tree)



Bulking Load (Construção da B+ Tree)



Árvore construída!

Atividade

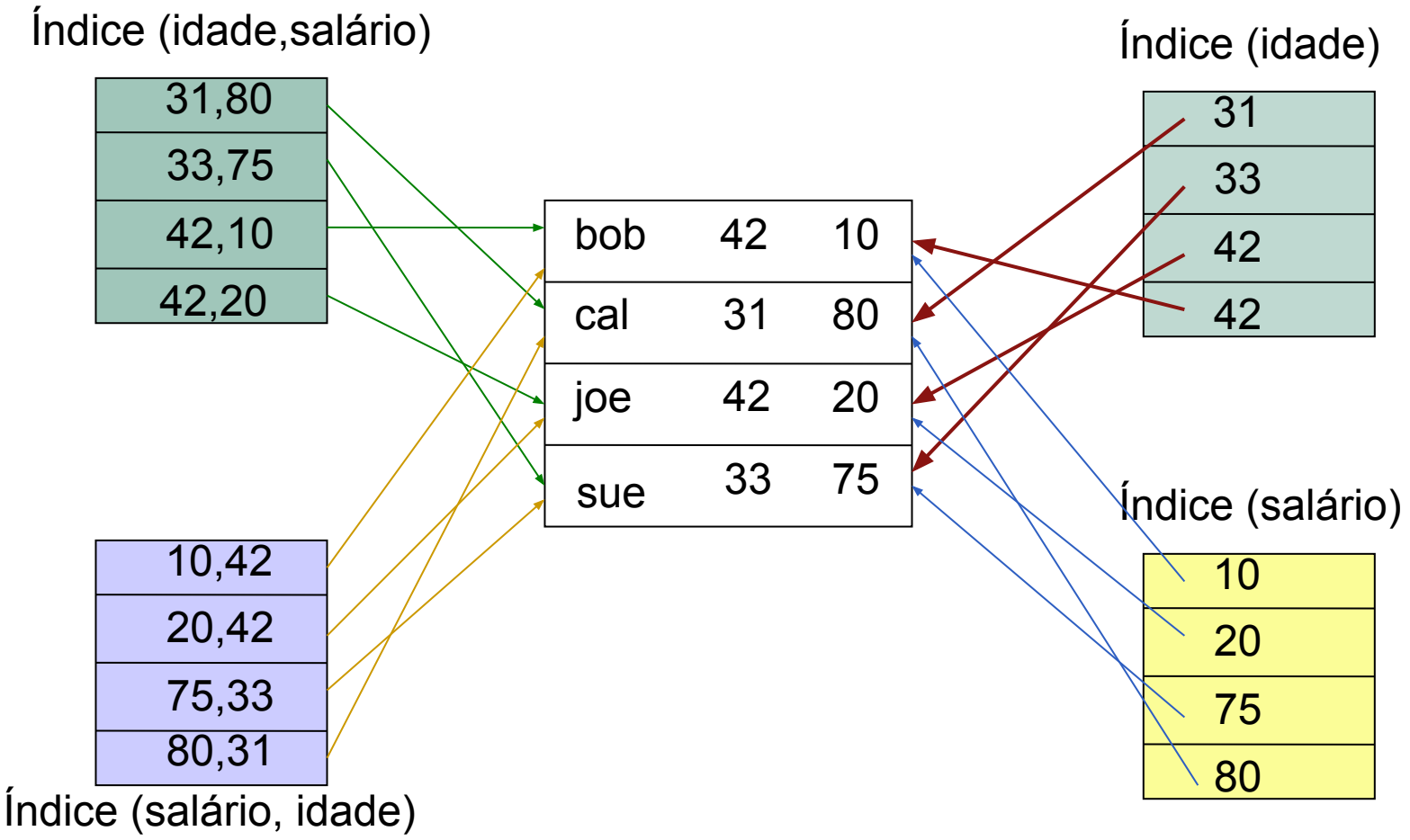
1) Faça a leitura usando bulking load com a ordem 4 dos seguintes valores: 1 ,2 ,5 ,10 ,15 , 20, 40,60, 80 ,100 ,120 ,140,

Índices

- Conclusões
  - Os dados são mais acessados que atualizados
  - Necessário existir uma estrutura auxiliar para melhorar o desempenho das consultas
  - Para dados relacionais árvores B+ são os índices mais utilizados
  - O Otimizador de consultas utiliza índices sempre que possível

Índices

- Compostos



Prática

```
create table table_teste as SELECT generate_series(1,10000) AS id,
md5(random()::text) AS descr;
```

```
select *,ctid from teste_index;
```

```
create index Iteste on table_teste(descr);
```

```
select *,ctid from table_teste;
```

```
  cluster teste_index using index_teste;
```