

# Banco de Dados II

09 – Indexação

# A importância da Indexação

- Quando os dados são grandes demais para caber na **memória principal**, o número de acessos a disco (memória secundária) se torna importante.
- O acesso a disco é mais demorado e caro se comparado ao acesso à memória principal.
- Atualmente o **armazenamento estável** é feito em **discos magnéticos**, e o custo de cada acesso (da ordem de **mili segundos**) é muito alto quando comparado ao acesso à **memória RAM** (ordem de **nano segundos**).

# A importância da Indexação

- **Tempo de Resposta** corresponde ao **tempo decorrido ou atraso médio entre o início de uma transação e os resultados da transação.**
- Em um **Banco de Dados** que armazene um **volume significativo de registros**, o **tempo de resposta** é uma preocupação importante.
- Consultas a uma ou mais tabelas simultaneamente, por exemplo, **não devem apresentar um tempo de resposta elevado**. Esperar minutos ou muitos segundos pela apresentação dos registros recuperados por uma consulta é algo impensável para **usuários e administradores de banco de dados**.

# A importância da Indexação

- **Métricas de avaliação do Tempo de Resposta em Bancos de Dados:**
  - Tempo de Acesso;
  - Tempo de Inserção;
  - Tempo de Eliminação de Registros;
  - Sobrecarga (Overhead) de espaço de armazenamento.

# A importância da Indexação

## Leitura Sequencial de Registros

Cursor  
(Registro Corrente)

Procurando  
Nome = "Elvira"

Leitura Sequencial.



|  | Nome      | Cidade       | Cargo                |
|--|-----------|--------------|----------------------|
|  | Ana Júlia | Vitória      | Diretor              |
|  | Jambira   | Linhares     | Gerente              |
|  | Asdrúbal  | Linhares     | Analista de Sistemas |
|  | Carlomano | Linhares     | Programador          |
|  | Andressa  | Linhares     | Programador          |
|  | Cremildo  | Colatina     | Gerente              |
|  | Egsberto  | Colatina     | Analista de Sistemas |
|  | Solomano  | Colatina     | Programador          |
|  | Elvira    | Santa Teresa | Designer             |

Tabela de 3 colunas (nome, cidade e cargo) e 9 linhas (registros).

# A importância da Indexação

## Leitura Sequencial de Registros

- (1) - A leitura começa pelo primeiro registro (nome = “**Ana Júlia**”).
- (2) - Como “Ana Júlia” é diferente de “Elvira” (nossa argumento de pesquisa), o **cursor** move-se para o **registro seguinte** (nome = “**Jambira**”).
- (3) - Enquanto **não** for o final da tabela (após o último registro existente) e nome for diferente de “**Elvira**”, o **cursor** continuará a mover-se de registro a registro.
- Note que, em uma **Leitura Sequencial**, buscando do primeiro ao último registro pelo valor de um **argumento de pesquisa**, o **tempo de resposta** é determinado pela **posição física** do registro procurado.

# A importância da Indexação

## Leitura Sequencial de Registros

- O tempo de resposta da busca pelo nome = “**Asdrúbal**” é menor que o da busca pelo nome = “**Solomano**”.
- Em consequência, as consultas tornam-se **ineficientes**.
- Duas soluções foram propostas em termos de alternativas de acesso a registros que substituam a simples leitura sequencial:
  - Acesso por meio de **cálculo do endereço de registros (hashing)**;
  - Acesso por intermédio de **estruturas de dados auxiliares (índices)**.

# Hashing

- **Hashing** é uma técnica que transforma uma entrada (chave, valor ou conjunto de dados) em uma **saída de tamanho fixo**, chamada de **hash** (ou **código hash**), por meio de uma **função matemática** conhecida como **função de hash**.
- Essa saída costuma ser um número inteiro ou uma sequência curta de caracteres.
- Uma característica importante é que pequenas mudanças na entrada produzem resultados bem diferentes.
- Em geral, o hash não permite reverter para o dado original (é uma função unidirecional).

# Hashing

- Exemplo: **Estacionamento com Hash**
- Imagine um **estacionamento com 10 vagas numeradas (0 a 9)**. Queremos guardar carros sem procurar vaga livre manualmente.
- Cada carro tem uma **placa**.
- Para saber em qual vaga colocar o carro, aplicamos uma **função de hash simples**:

$$\text{hash(placa)} = (\text{soma dos códigos numéricos dos caracteres da placa}) \bmod 10$$

# Hashing

Placa: ABC123

- Convertendo letras em números (A=65, B=66, C=67 no código ASCII)
- Soma =  $65 + 66 + 67 + 1 + 2 + 3 = 204$
- Aplicando o módulo:  $204 \bmod 10 = 4$   
👉 O carro vai para a vaga 4.

Placa: XYZ999

- X=88, Y=89, Z=90
- Soma =  $88 + 89 + 90 + 9 + 9 + 9 = 294$
- $294 \bmod 10 = 4$   
👉 Também vai para a vaga 4.

# Hashing

- Dois carros diferentes (**ABC123** e **XYZ999**) foram parar na mesma vaga (**colisão de hash**).
- **Em bancos de dados e sistemas de hash, isso é esperado.**
- A solução é usar técnicas de resolução de colisão (por exemplo, colocar em uma lista dentro da mesma vaga ou procurar a próxima vaga livre).

# Hashing

- Esse **estacionamento** é como **uma tabela hash dentro de um banco de dados**:
  - **Placa** → **chave de pesquisa (coluna com índice hash)**
  - **Número da vaga** → **posição calculada pela função hash**
  - **Carro estacionado** → **registro armazenado**
- Assim, **para buscar o carro pela placa, em vez de olhar todas as vagas, o sistema calcula o hash e vai direto na vaga correspondente.**

# Hashing

The screenshot shows a pgAdmin 4 interface. At the top, there's a connection bar with a gear icon and the text "Estacionamento/postgres@PostgreSQL 17". Below it is a toolbar with several icons: a folder, a file, a pencil, a magnifying glass, and a refresh symbol. To the right of the toolbar are dropdown menus for "No limit" and other options, followed by a copy icon and a run icon. The main area has tabs for "Query" (selected) and "Query History". The "Query" tab contains the following SQL code:

```
1 CREATE TABLE estacionamento (
2     id_carro SERIAL PRIMARY KEY,
3     placa VARCHAR(10) NOT NULL,
4     modelo VARCHAR(50) NOT NULL
5 );
6
```

Below the code, under the "Messages" tab, the output is shown:

```
CREATE TABLE
```

Query returned successfully in 84 msec.

# Hashing

The screenshot shows a PostgreSQL database client interface. The title bar reads "Estacionamento/postgres@PostgreSQL 17". Below the title bar is a toolbar with various icons. The main area is divided into sections: "Query" (selected), "Query History", and "Data Output" (selected). The "Query" section contains the following SQL code:

```
1 CREATE TABLE estacionamento (
2     id_carro SERIAL PRIMARY KEY,
3     placa VARCHAR(10) NOT NULL,
4     modelo VARCHAR(50) NOT NULL
5 );
6
7 INSERT INTO estacionamento (placa, modelo)
8 VALUES
9 ('ABC123', 'Fiat Uno'),
10 ('XYZ999', 'Honda Civic'),
11 ('DEF456', 'Toyota Corolla'),
12 ('GHI789', 'Chevrolet Onix');
13
```

The "Data Output" section shows the results of the insertion query:

```
INSERT 0 4
```

Below the output, a message states:

```
Query returned successfully in 78 msec.
```

# Hashing

The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'Estacionamento/postgres@PostgreSQL 17'. The toolbar includes standard database management icons. Below the toolbar, there are tabs for 'Query' and 'Query History', with 'Query' being the active tab. The main area displays a SQL script with numbered lines:

```
7  INSERT INTO estacionamento (placa, modelo)
8    VALUES
9      ('ABC123', 'Fiat Uno'),
10     ('XYZ999', 'Honda Civic'),
11     ('DEF456', 'Toyota Corolla'),
12     ('GHI789', 'Chevrolet Onix');
13
14  CREATE INDEX idx_estacionamento_placa_hash
15  ON estacionamento USING hash (placa);
16
```

Line 14, which creates a hash index, is highlighted with a light blue background. Below the code, there are tabs for 'Data Output', 'Messages', and 'Notifications', with 'Messages' being the active tab. The message area shows the command 'CREATE INDEX' and the success message 'Query returned successfully in 72 msec.'

# Hashing

The screenshot shows a PostgreSQL session in pgAdmin 4. The top bar displays the connection information: Estacionamento/postgres@PostgreSQL 17. Below the connection bar is a toolbar with various icons for file operations, search, and navigation. The main area is divided into two tabs: "Query" (selected) and "Query History". The "Query" tab contains the following SQL code:

```
14  CREATE INDEX idx_estacionamento_placa_hash  
15  ON estacionamento USING hash (placa);  
16  
17  SELECT *  
18  FROM estacionamento  
19  WHERE placa = 'XYZ999';  
20  
21
```

Below the code, there are three tabs: "Data Output", "Messages", and "Notifications". The "Data Output" tab is selected and shows a table with the results of the query. The table has four columns: id\_carro, placa, modelo, and a fourth column which is partially visible. The data row is as follows:

|   | id_carro<br>[PK] integer | placa<br>character varying (10) | modelo<br>character varying (50) |
|---|--------------------------|---------------------------------|----------------------------------|
| 1 | 2                        | XYZ999                          | Honda Civic                      |

# Hashing

The screenshot shows the pgAdmin 4 interface. At the top, there's a connection bar with the text "Estacionamento/postgres@PostgreSQL 17". Below it is a toolbar with various icons for file operations, search, and navigation. The main area is divided into two tabs: "Query" (which is selected) and "Query History". The "Query" tab contains the following SQL code:

```
14 ✓ CREATE INDEX idx_estacionamento_placa_hash
15   ON estacionamento USING hash (placa);
16
17 ✓ SELECT *
18   FROM estacionamento
19   WHERE placa = 'XYZ999';
20
21 -- verificar que o banco realmente está usando o índice hash
22 ✓ EXPLAIN SELECT *
23   FROM estacionamento
24   WHERE placa = 'XYZ999';
25
```

Below the code, there are three tabs: "Data Output", "Messages", and "Notifications". The "Data Output" tab is selected. It displays a "QUERY PLAN" table with two rows:

|   | QUERY PLAN<br>text  | lock |
|---|---|------|
| 1 | Seq Scan on estacionamento (cost=0.00..1.05 rows=1 width=16...) |      |
| 2 | Filter: ((placa)::text = 'XYZ999'::text)                        |      |

# Hashing

```
21 -- verificar que o banco realmente está usando o índice hash
22 EXPLAIN SELECT *
23   FROM estacionamento
24 WHERE placa = 'XYZ999';
25
```



Estávamos esperando por uma saída como “**Index Scan using idx\_estacionamento\_placa\_hash on estacionamento ...**”.

Mas não foi essa a saída apresentada: “**Seq Scan on estacionamento (cost=0.00..1.05 rows=1 width= 160) ...**” que indica leitura sequencial (Seq Scan).

# Hashing

O **PostgreSQL** raramente escolhe índices hash, porque:

- **Índices B-Tree** são muito mais versáteis (suportam `=, <, >, BETWEEN, LIKE 'ABC%' etc).`
- **Índices HASH** só ajudam em igualdade (`=`).
- Até o **PostgreSQL 10**, índices hash nem eram transacionais (não eram **WAL-logged**), o que os tornava inseguros após um **crash**. **Hoje são seguros, mas ainda pouco usados.**

# Hashing

```
20
21 -- verificar que o banco realmente está usando o índice hash
22 EXPLAIN SELECT *
23 FROM estacionamento
24 WHERE placa = 'XYZ999';
25
26 -- Como forçar o PostgreSQL a usar o índice HASH
27 SET enable_seqscan = off;
28
```

Data Output Messages Notifications

SET

Query returned successfully in 168 msec.

# Hashing

```
1  
2 -- Como forçar o PostgreSQL a usar o índice HASH  
3 SET enable_seqscan = off;  
4  
5 -- verificar que o banco realmente está usando o índice hash  
6 EXPLAIN SELECT *  
7 FROM estacionamento  
8 WHERE placa = 'XYZ999';  
9
```

Data Output Messages Notifications



QUERY PLAN  
text

|   |   |
|---|---|
| 1 | Index Scan using idx_estacionamento_placa_hash on estacionamento (cost=0.00..8.02 rows=1 width=160) |
| 2 | Index Cond: ((placa)::text = 'XYZ999'::text)  |

Agora sim: a indexação baseada na função hash foi utilizada.

# Hashing

- Observe como o **otimizador de consultas do PostgreSQL** trabalha.
- Por que o **Seq Scan (a leitura sequencial sem uso de indexação)** foi estimado como **mais barato (0 .. 1.05)**?
- O **PostgreSQL** calcula custo estimado para cada plano de execução.
- **Esse custo não é tempo em milissegundos, mas uma unidade abstrata baseada em fatores como:**
  - **Quantidade de páginas de disco que precisa ler.**
  - **Custo de CPU para comparar registros.**
  - **Custos definidos em postgresql.conf (parâmetros como seq\_page\_cost, random\_page\_cost, cpu\_tuple\_cost).**

# Hashing

- Se a tabela tem poucos registros, ler tudo sequencialmente custa quase nada.
- Por isso, o plano Seq Scan saiu com custo 0..1.05, indicando que é super barato apenas varrer a tabela inteira.
- Por que o Hash Index teve custo maior (0 .. 8.02)?
- Usar um índice não é de graça:
  - Precisa navegar no índice (acessar páginas extras em disco).
  - Depois, precisa acessar o registro real na tabela (heap).
  - Ainda existe overhead de cálculo de hash e resolução de possíveis colisões.

# Hashing

- Em **uma tabela pequena**, esse esforço extra é **mais custoso** do que simplesmente varrer tudo.
- Por isso o **otimizador** estimou **custo 0..8.02**, ou seja, “**vale a pena só se a tabela for grande**”.
- Seq Scan ganha em tabelas pequenas: custo baixo porque ler tudo é rápido.
- **Índices (Hash ou B-Tree) só compensam em tabelas grandes: quando a varredura completa seria muito cara.**
- O **PostgreSQL** escolhe automaticamente o plano com menor custo estimado.

# Hashing

- Se você inserir **milhões de registros em estacionamento**, e buscar por **WHERE placa = 'XYZ999'**, o custo do Seq Scan vai **disparar** (ex: 0..50000).
- Nesse ponto, o **Hash Index (ou o B-Tree)** passará a ser **escolhido**, porque **fica mais barato buscar direto no índice**.

# Hashing

```
9  
10  DROP TABLE IF EXISTS estacionamento;  
11  CREATE TABLE estacionamento (  
12      id_carro SERIAL PRIMARY KEY,  
13      placa VARCHAR(10) NOT NULL,  
14      modelo VARCHAR(50) NOT NULL  
15  );  
16
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 101 msec.

- Vamos montar um **mini experimento prático** no **PostgreSQL** para observar na prática quando o **otimizador** troca do **Seq Scan** para o **Index Scan**

# Hashing

```
Query  Query History
10  DROP TABLE IF EXISTS estacionamento;
11  CREATE TABLE estacionamento (
12      id_carro SERIAL PRIMARY KEY,
13      placa VARCHAR(10) NOT NULL,
14      modelo VARCHAR(50) NOT NULL
15  );
16
17  CREATE INDEX idx_estacionamento_placa_hash
18  ON estacionamento USING hash (placa);
19
20
```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 224 msec.

# Hashing

```
16
17 ✓ CREATE INDEX idx_estacionamento_placa_hash
18   ON estacionamento USING hash (placa);
19
20   -- Inserindo poucos registros (100 ao todo).
21 ✓ INSERT INTO estacionamento (placa, modelo)
22   SELECT 'CAR' || g, 'Modelo ' || g
23   FROM generate_series(1, 100) g;
24
```

Data Output Messages Notifications

INSERT 0 100

Query returned successfully in 80 msec.

# Hashing

```
20 -- Inserindo poucos registros (100 ao todo).
21 ✓ INSERT INTO estacionamento (placa, modelo)
22   SELECT 'CAR' || g, 'Modelo ' || g
23   FROM generate_series(1, 100) g;
24
25 -- liberando para o otimizador escolher a utilização ou não de indexação.
26 SET enable_seqscan = on;
27
28 -- Analisando a consulta (Query) com EXPLAIN
29 ✓ EXPLAIN SELECT *
30   FROM estacionamento
31   WHERE placa = 'CAR50';
32
```

Data Output Messages Notifications



QUERY PLAN  
text



|   |  |
|---|--|
| 1 | Seq Scan on estacionamento (cost=0.00..2.25 rows=1 width=18) |
| 2 | Filter: ((placa)::text = 'CAR50)::text)                      |

# Hashing

```
25 -- liberando para o otimizador escolher a utilização ou não de indexação.  
26 SET enable_seqscan = on;  
27  
28 -- Analisando a consulta (Query) com EXPLAIN  
29 ▼ EXPLAIN SELECT *  
30 FROM estacionamento  
31 WHERE placa = 'CAR50';  
32  
33 -- Inserindo MUITOS registros (1 milhão ao todo além dos 100 anteriores).  
34 ▼ INSERT INTO estacionamento (placa, modelo)  
35 SELECT 'CAR' || g, 'Modelo ' || g  
36 FROM generate_series(101, 1000000) g;  
37
```

Data Output Messages Notifications

INSERT 0 999900

Query returned successfully in 12 secs 111 msec.

# Hashing

```
32
33 -- Inserindo MUITOS registros (1 milhão ao todo além dos 100 anteriores).
34 ✓ INSERT INTO estacionamento (placa, modelo)
35   SELECT 'CAR' || g, 'Modelo ' || g
36   FROM generate_series(101, 1000000) g;
37
38 -- Analisando novamente a QUERY.
39 ✓ EXPLAIN SELECT *
40   FROM estacionamento
41   WHERE placa = 'CAR500000';
42
```

Data Output Messages Notifications



QUERY PLAN  
text



|   |  |
|---|--|
| 1 | Index Scan using idx_estacionamento_placa_hash on estacionamento (cost=0.00..8.02 rows=1 width=26) |
| 2 | Index Cond: ((placa)::text = 'CAR500000'::text)  |

# Hashing

- A **função Hash** (Resumo) é qualquer algoritmo que mapeie dados grandes e de tamanho variável para pequenos dados de tamanho fixo.
- De uma maneira geral, as **funções Hash** são largamente **utilizadas** para:
  - buscar elementos em bases de dados,
  - verificar a integridade de arquivos baixados ou
  - armazenar e transmitir senhas de usuários.

# Hashing

- A **busca de elementos** baseado em **hash** é usada tanto em **bancos de dados** quanto em **estruturas de dados em memória**.
- O funcionamento se baseia na construção de índices, que funcionam de forma semelhante ao **índice de livros**.

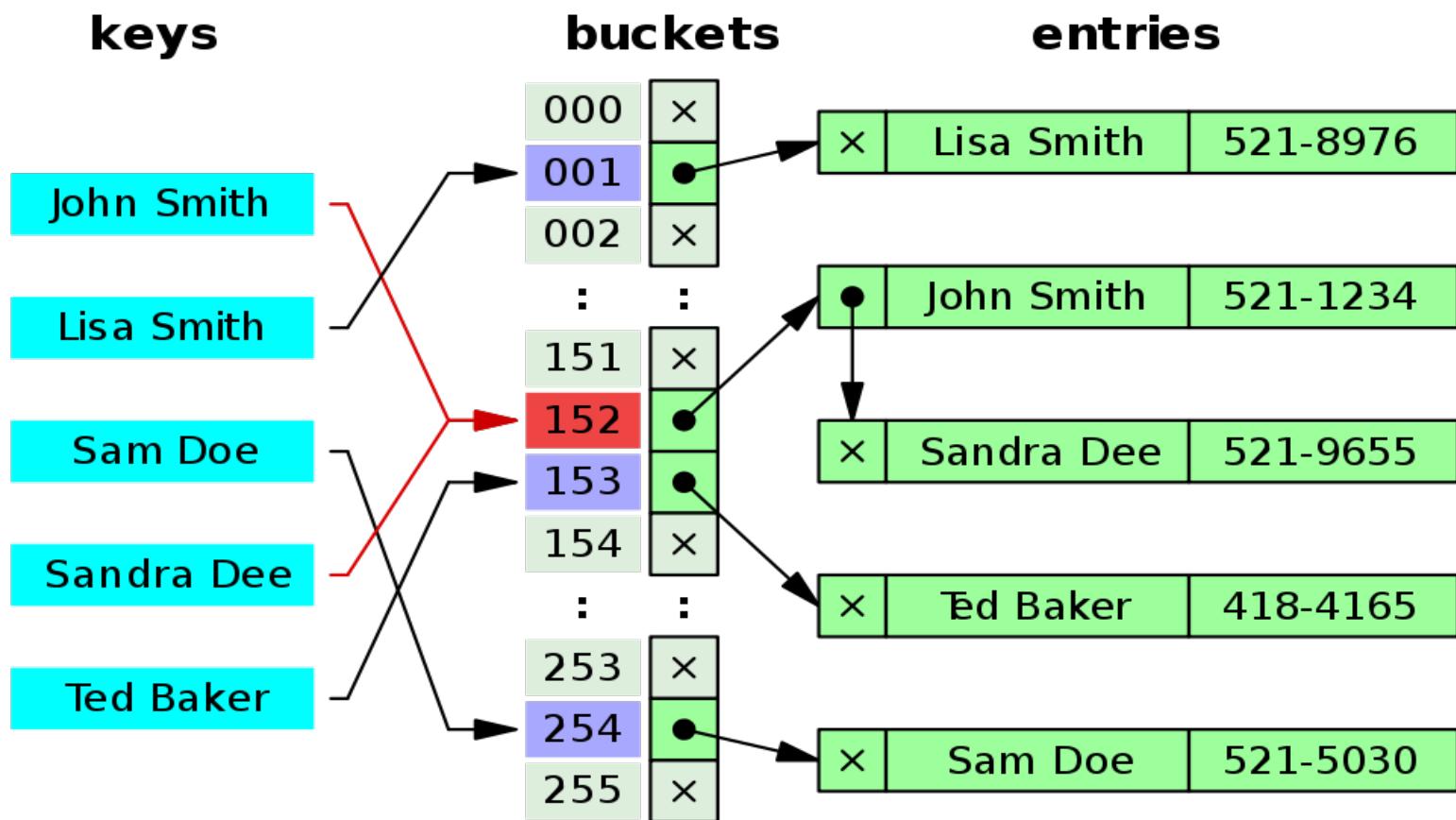
# Hashing

- Quando você busca algo em um **livro**, antes olha no **índice** para ver em que **página** está a informação que você precisa.
- Após, dirige-se diretamente para a **página** desejada.
- Nos **índices** baseados em **Hashs**, a **busca** pela **página** da informação é feita pelo **resumo** e não pelo **dado**.

# Hashing

- Em geral, calcula-se o **resumo** do dado e, com o **resumo**, sabe-se em que **página** o dado está.
- Não precisa-se varrer todo o índice para descobrir a **página** em que o dado se encontra.

# Hashing



# Hashing

## Ideias preliminares

- Exemplo 1: Imagine um pequeno país (bem menos que 100 mil cidadãos) onde os números de CPF têm apenas 5 dígitos decimais. Considere a tabela que leva CPFs em nomes:

| chave | valor associado |
|-------|-----------------|
| 01555 | Ronaldo         |
| 01567 | Pelé            |
| ...   | ...             |
| 80114 | Maradona        |
| 80320 | Dunga           |
| 95222 | Romário         |

Como armazenar a tabela? Resposta: vetor de 100 mil posições. Use *a própria chave* como índice do vetor!

- O vetor é conhecido com "tabela de hash" e
- terá muitas posições vagas (desperdício de espaço), mas
- a busca (get) e a inserção (put) serão extremamente rápidas.

# Hashing

- Exemplo 2: Imagine uma lista ligada cujas chaves são nomes de pessoas. Suponha que a lista está em ordem alfabética.

| chave           | valor associado |
|-----------------|-----------------|
| Antonio Silva   | 8536152         |
| Arthur Costa    | 7210629         |
| Bruno Carvalho  | 8536339         |
| ...             | ...             |
| Vitor Sales     | 8535922         |
| Wellington Lima | 5992240         |
| Yan Ferreira    | 8536023         |

Para acelerar as buscas, divida a lista em 26 pedaços: os nomes que começam com "A", os que começam com "B", etc. Nesse caso,

- o vetor de 26 posições é a "tabela de hash" e
- cada posição do vetor aponta para o começo de uma das listas.

# Indexação

- **Índice** é uma estrutura de dados que serve para localizar registros no arquivo de dados (tabela).
- Cada entrada do **índice** contém:
  - **Valor da chave**
  - **Ponteiro para o arquivo de dados**

Pode-se pensar então em **dois arquivos**:

**Um de índice**

**Um de dados**

- **Isso é eficiente?**

# Indexação

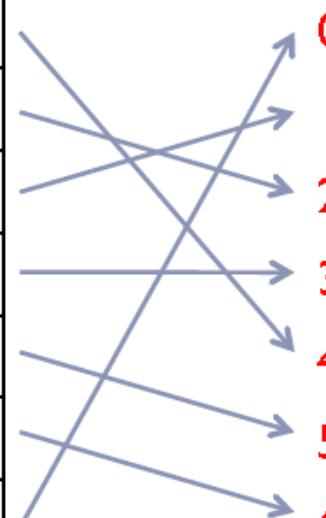
## Exemplo de Índice Plano

Arquivo de Índice

|   | CHAVE | PONTEIRO |
|---|-------|----------|
| 0 | 3     | 4        |
| 1 | 5     | 2        |
| 2 | 10    | 1        |
| 3 | 15    | 3        |
| 4 | 16    | 5        |
| 5 | 21    | 6        |
| 6 | 23    | 0        |

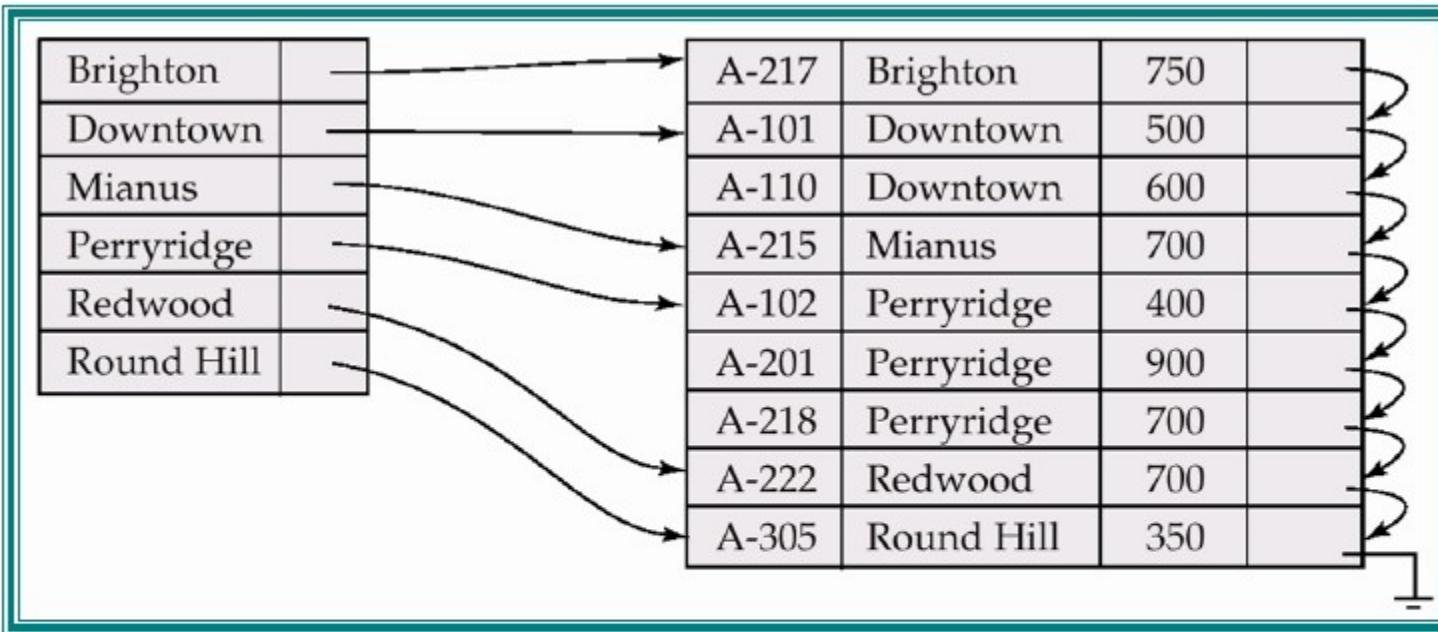
Arquivo de Dados

|   | COD | NOME    |
|---|-----|---------|
| 0 | 23  | JOSE    |
| 1 | 10  | MARIO   |
| 2 | 5   | ANA     |
| 3 | 15  | MARCIA  |
| 4 | 3   | JULIO   |
| 5 | 16  | BEATRIZ |
| 6 | 21  | CAMILA  |



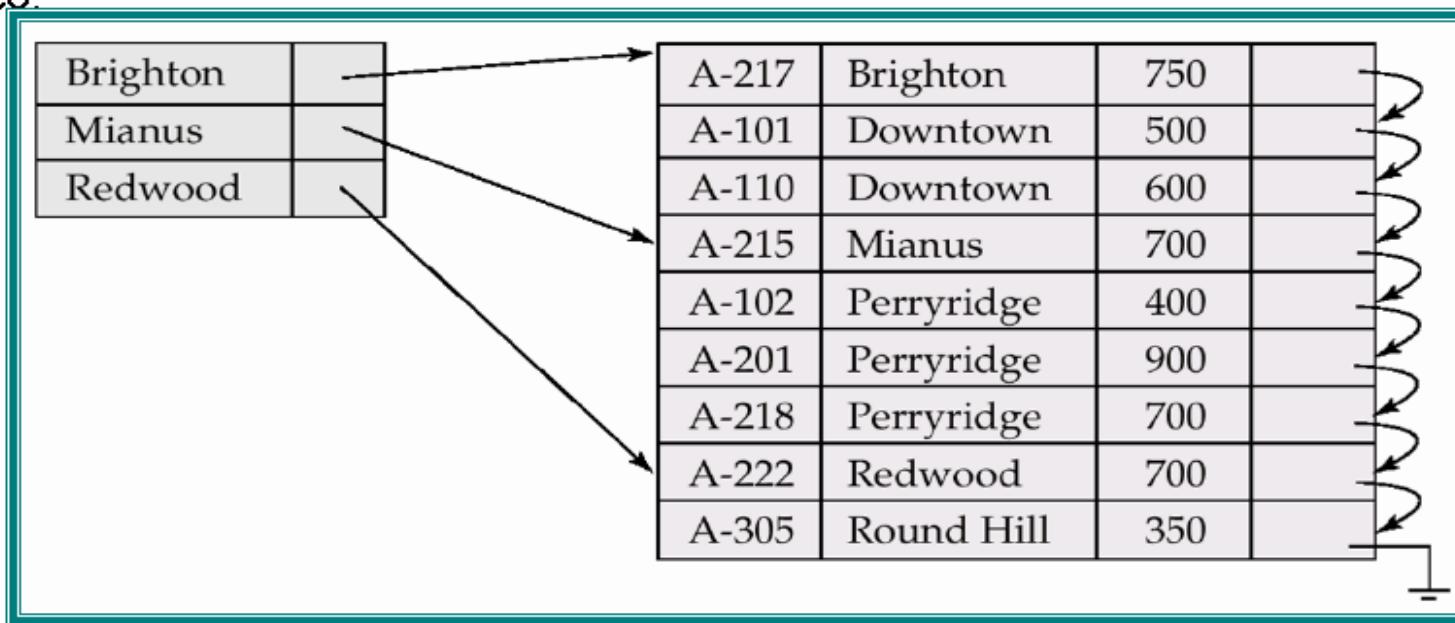
# Indexação

## Índice Denso



Índice denso — um registro de índice aparece para todo o valor da chave de pesquisa na tabela.

# Indexação Índice Esparsos



Menos espaço e menor sobrecarga de manutenção para inserções e eliminações.

Normalmente **mais lento que o índice denso** em localizar registros.

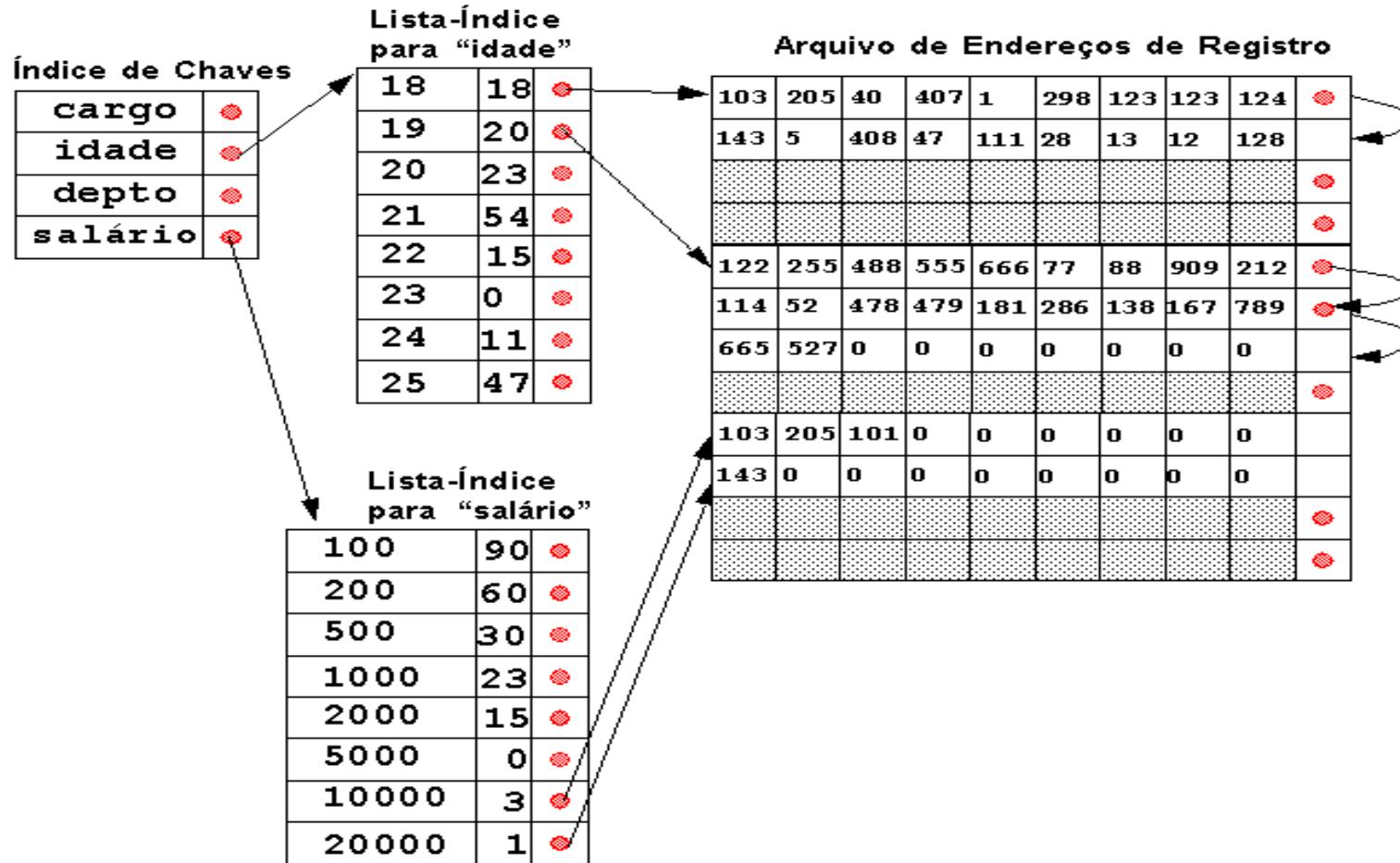
**índice esparsos com verbete de indexação para todo o bloco** numa tabela, correspondendo ao menor valor da chave de pesquisa num bloco.

# Indexação

## Lista Invertida

- Uma **lista invertida** é uma estrutura de indexação usada para localizar rapidamente registros que contêm um determinado valor ou conjunto de valores.
  - O nome vem do fato de que ela “inverte” a relação tradicional:
  - Em vez de termos documento → termos,
  - Armazenamos termo → lista de documentos.

# Indexação Lista Invertida



# Indexação

## Lista Invertida

- **Estrutura**
- Uma lista invertida é geralmente composta de dois elementos principais:
  - **Vocabulário (dicionário de termos)**: cada valor (palavra, chave ou atributo) aparece uma vez.
  - **Lista de ocorrências (postings list)**: para cada termo, mantém-se uma lista dos documentos (ou registros) em que ele aparece, possivelmente com informações adicionais (posição, frequência, etc.).

# Indexação Lista Invertida

Imagine uma tabela simplificada de filmes no banco:

| ID_Filme | Título                                |
|----------|---------------------------------------|
| 1        | Matrix Revolutions                    |
| 2        | Matrix Reloaded                       |
| 3        | Senhor dos Anéis: A Sociedade do Anel |
| 4        | O Senhor dos Anéis: As Duas Torres    |

Se usarmos uma lista invertida sobre os títulos, ela ficaria mais ou menos assim:

- Matrix → [1, 2]
- Revolutions → [1]
- Reloaded → [2]
- Senhor → [3, 4]
- Anéis → [3, 4]
- Sociedade → [3]
- Torres → [4]

# Indexação Lista Invertida

- Em uma busca pelo termo “MATRIX” começamos pela tabela invertida:

Se usarmos uma lista invertida sobre os títulos, ela ficaria mais ou menos assim:

- Matrix → [1, 2]
- Revolutions → [1]
- Reloaded → [2]
- Senhor → [3, 4]
- Anéis → [3, 4]
- Sociedade → [3]
- Torres → [4]

- Ali descobrimos que o termo “MATRIX” é encontrado nos registros 1 e 2 da Tabela.

# Indexação Lista Invertida

Imagine uma tabela simplificada de filmes no banco:

| ID_Filme | Título                                |
|----------|---------------------------------------|
| 1        | Matrix Revolutions                    |
| 2        | Matrix Reloaded                       |
| 3        | Senhor dos Anéis: A Sociedade do Anel |
| 4        | O Senhor dos Anéis: As Duas Torres    |

# Indexação Lista Invertida

```
43 -----
44 -----
45 DROP TABLE IF EXISTS filmes;
46 ✓ CREATE TABLE filmes (
47     id_filme SERIAL PRIMARY KEY,
48     titulo TEXT NOT NULL
49 );
50
```

Data Output Messages Notifications

NOTA: tabela "filmes" não existe, ignorando  
CREATE TABLE

Query returned successfully in 98 msec.

# Indexação Lista Invertida

Query    Query History

---

```
44
45  DROP TABLE IF EXISTS filmes;
46  ✓ CREATE TABLE filmes (
47      id_filme SERIAL PRIMARY KEY,
48      titulo TEXT NOT NULL
49  );
50
51  ✓ INSERT INTO filmes (titulo) VALUES
52      ('Matrix'),
53      ('Matrix Reloaded'),
54      ('Matrix Revolutions'),
55      ('Senhor dos Anéis: A Sociedade do Anel'),
56      ('O Senhor dos Anéis: As Duas Torres'),
57      ('O Senhor dos Anéis: O Retorno do Rei');
58
59
```

---

Data Output    Messages    Notifications

---

INSERT 0 6

Query returned successfully in 75 msec.

# Indexação Lista Invertida

Query    Query History

```
51 ✓ INSERT INTO filmes (titulo) VALUES
52   ('Matrix'),
53   ('Matrix Reloaded'),
54   ('Matrix Revolutions'),
55   ('Senhor dos Anéis: A Sociedade do Anel'),
56   ('O Senhor dos Anéis: As Duas Torres'),
57   ('O Senhor dos Anéis: O Retorno do Rei');
58
59   -- O PostgreSQL precisa transformar o texto em tokens (palavras normalizadas).
60   -- Usamos to_tsvector para isso:
61 ✓ ALTER TABLE filmes
62   ADD COLUMN titulo_tsv tsvector;
63
64   -- Popular a coluna com tokens
65 ✓ UPDATE filmes
66   SET titulo_tsv = to_tsvector('portuguese', titulo);
67
```

Data Output    Messages    Notifications

ALTER TABLE

Query returned successfully in 77 msec.

# Indexação Lista Invertida

```
58
59 -- O PostgreSQL precisa transformar o texto em tokens (palavras normalizadas).
60 -- Usamos to_tsvector para isso:
61 ✓ ALTER TABLE filmes
62   ADD COLUMN titulo_tsv tsvector;
63
64 -- Popular a coluna com tokens
65 ✓ UPDATE filmes
66   SET titulo_tsv = to_tsvector('portuguese', titulo);
67
```

Data Output Messages Notifications

UPDATE 6

Query returned successfully in 78 msec.

# Indexação Lista Invertida

```
59  -- O PostgreSQL precisa transformar o texto em tokens (palavras normalizadas).
60  -- Usamos to_tsvector para isso:
61  ✓ ALTER TABLE filmes
62    ADD COLUMN titulo_tsv tsvector;
63
64  -- Popular a coluna com tokens
65  ✓ UPDATE filmes
66    SET titulo_tsv = to_tsvector('portuguese', titulo);
67
68  -- Criação de índice GIN (Lista Invertida):
69  ✓ CREATE INDEX idx_filmes_titulo_tsv
70    ON filmes USING gin (titulo_tsv);
71
72
```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 150 msec.

# Indexação Lista Invertida

```
64 -- Popular a coluna com tokens
65 UPDATE filmes
66 SET titulo_tsv = to_tsvector('portuguese', titulo);
67
68 -- Criação de índice GIN (Lista Invertida):
69 CREATE INDEX idx_filmes_titulo_tsv
70 ON filmes USING gin (titulo_tsv);
71
72 -- Esse índice é a lista invertida:
73 -- Cada termo do título (ex: "matrix", "senhor", "anéis") é armazenado como chave.
74 -- Ele aponta para a lista de documentos (IDs de filmes) onde aparece.
75
76 SELECT id_filme, titulo
77 FROM filmes
78 WHERE titulo_tsv @@ to_tsquery('portuguese', 'matrix');
79
```

Data Output Messages Notifications



|   | <b>id_filme</b><br>[PK] integer | <b>titulo</b><br>text |
|---|---------------------------------|-----------------------|
| 1 | 1                               | Matrix                |
| 2 | 2                               | Matrix Reloaded       |
| 3 | 3                               | Matrix Revolutions    |

# Indexação Lista Invertida

```
72 -- Esse índice é a lista invertida:  
73 -- Cada termo do título (ex: "matrix", "senhor", "anéis") é armazenado como chave.  
74 -- Ele aponta para a lista de documentos (IDs de filmes) onde aparece.  
75  
76 ▼ SELECT id_filme, titulo  
77   FROM filmes  
78 WHERE titulo_tsv @@ to_tsquery('portuguese', 'matrix');  
79  
80 -- buscando "Senhor e Aneis": Falha  
81 ▼ SELECT id_filme, titulo  
82   FROM filmes  
83 WHERE titulo_tsv @@ to_tsquery('portuguese', 'senhor & aneis');
```

Data Output Messages Notifications



|  | id_filme<br>[PK] integer | titulo<br>text |
|--|--------------------------|----------------|
|  |                          |                |

# Indexação Lista Invertida

```
79
80    -- buscando "Senhor e Aneis": Falha
81    ▾ SELECT id_filme, titulo
82        FROM filmes
83        WHERE titulo_tsv @@ to_tsquery('portuguese', 'senhor & aneis');
84
85
86    ▾ SELECT id_filme, titulo, to_tsvector('portuguese', titulo)
87        FROM filmes;
88
```

Data Output Messages Notifications

The screenshot shows a PostgreSQL pgAdmin interface. At the top, there's a toolbar with various icons for database management. Below the toolbar is a table with six rows of data. The columns are labeled: id\_filme [PK] integer, titulo text, and to\_tsvector tsvector.

|   | id_filme<br>[PK] integer | titulo<br>text                        | to_tsvector<br>tsvector                  |
|---|--------------------------|---------------------------------------|--|
| 1 | 1                        | Matrix                                | 'matrix':1                               |
| 2 | 2                        | Matrix Reloaded                       | 'matrix':1 'reloaded':2                  |
| 3 | 3                        | Matrix Revolutions                    | 'matrix':1 'revolutions':2               |
| 4 | 4                        | Senhor dos Anéis: A Sociedade do Anel | 'anel':7 'ané':3 'senhor':1 'sociedad':5 |
| 5 | 5                        | O Senhor dos Anéis: As Duas Torres    | 'ané':4 'duas':6 'senhor':2 'torr':7     |
| 6 | 6                        | O Senhor dos Anéis: O Retorno do Rei  | 'ané':4 'rei':8 'retorn':6 'senhor':2    |

# Indexação Lista Invertida

Query History

```
80 -- buscando "Senhor e Aneis": Falha
81 SELECT id_filme, titulo
82 FROM filmes
83 WHERE titulo_tsv @@ to_tsquery('portuguese', 'senhor & aneis');
84
85
86 SELECT id_filme, titulo, to_tsvector('portuguese', titulo)
87 FROM filmes;
88
89 -- Repare: “Anéis” → “anel”
90 -- “Sociedade” → “sociedad”
91 -- Como vimos, não adianta buscar aneis. O correto é usar a forma stemmed que o dicionário gerou.
92 -- Exemplo:
93
94 SELECT id_filme, titulo
95 FROM filmes
96 WHERE titulo_tsv @@ to_tsquery('portuguese', 'senhor & anel');
```

Data Output Messages Notifications

SQL

|   | id_filme<br>[PK] integer | titulo<br>text                        |
|---|--------------------------|---------------------------------------|
| 1 | 4                        | Senhor dos Anéis: A Sociedade do Anel |

# Indexação Lista Invertida

```
--  
93  
94 v SELECT id_filme, titulo  
95   FROM filmes  
96 WHERE titulo_tsv @@ to_tsquery('portuguese', 'senhor & anel');  
97  
98 -- Alternativa sem stemming (text search "literal")  
99 -- Se você quiser que o PostgreSQL não faça redução de palavras (stemmer),  
100 -- pode usar a configuração "simple" em vez de "portuguese":  
101  
102 v UPDATE filmes  
103 SET titulo_tsv = to_tsvector('simple', titulo);  
104
```

---

Data Output [Messages](#) Notifications

UPDATE 6

Query returned successfully in 184 msec.

# Indexação Lista Invertida

```
102 UPDATE filmes
103 SET titulo_tsv = to_tsvector('simple', titulo);
104
105 -- Agora os tokens ficam exatamente iguais às palavras originais (Anéis, Sociedade, etc).
106 -- Mas aí você perde a vantagem de buscar plurais e variações automaticamente.
107
108 -- buscando "Senhor e Aneis": Ainda Falha
109 SELECT id_filme, titulo
110 FROM filmes
111 WHERE titulo_tsv @@ to_tsquery('portuguese', 'senhor & anéis');
112
113
114
```

Data Output Messages Notifications



|  | id_filme     | titulo |
|--|--------------|--------|
|  | [PK] integer | text   |

# Indexação Lista Invertida

```
102 UPDATE filmes
103 SET titulo_tsv = to_tsvector('simple', titulo);
104
105 -- Agora os tokens ficam exatamente iguais às palavras originais (Anéis, Sociedade, etc).
106 -- Mas aí você perde a vantagem de buscar plurais e variações automaticamente.
107
108 -- buscando "Senhor e Aneis": Ainda Falha
109 SELECT id_filme, titulo
110   FROM filmes
111 WHERE titulo_tsv @@ to_tsquery('portuguese', 'senhor & anéis');
112
113 -- O PostgreSQL não reconheceu anéis como a mesma raiz que foi gravada no índice.
114 -- No índice, quando você rodou to_tsvector('portuguese', titulo), a palavra “Anéis”
115 -- foi reduzida para o radical anel (sem acento e no singular).
116
117 SELECT id_filme, titulo, to_tsvector('portuguese', titulo)
118   FROM filmes;
119
120
121
```

Data Output Messages Notifications

The screenshot shows a PostgreSQL client interface with a toolbar at the top and a table below. The table has four columns: id\_filme, titulo, to\_tsvector, and tsvector. The data shows that the 'to\_tsvector' column contains the inverted index representation of the movie titles.

|   | id_filme<br>[PK] integer | titulo<br>text                        | to_tsvector<br>tsvector                  |
|---|--------------------------|---------------------------------------|--|
| 1 | 1                        | Matrix                                | 'matrix':1                               |
| 2 | 2                        | Matrix Reloaded                       | 'matrix':1 'reloaded':2                  |
| 3 | 3                        | Matrix Revolutions                    | 'matrix':1 'revolutions':2               |
| 4 | 4                        | Senhor dos Anéis: A Sociedade do Anel | 'anel':7 'ané':3 'senhor':1 'sociedad':5 |
| 5 | 5                        | O Senhor dos Anéis: As Duas Torres    | 'ané':4 'duas':6 'senhor':2 'torr':7     |
| 6 | 6                        | O Senhor dos Anéis: O Retorno do Rei  | 'ané':4 'rei':8 'retorn':6 'senhor':2    |

# Indexação Lista Invertida

```
117 ✓ SELECT id_filme, titulo, to_tsvector('portuguese', titulo)
118   FROM filmes;
119
120 -- inspecionar diretamente o parser:
121 ✓ SELECT to_tsvector('portuguese', 'Senhor dos Anéis'),
122       to_tsquery('portuguese', 'senhor & anéis');
123
```

Data Output Messages Notifications



|   | to_tsvector<br>tsvector | to_tsquery<br>tsquery |
|---|-------------------------|-----------------------|
| 1 | 'ané':3 'senhor':1      | 'senhor' & 'ané'      |

# Indexação Lista Invertida

```
123
124 -- Se o usuário digitar “anéis”, você pode aplicar plainto_tsquery,
125 -- que tenta tokenizar automaticamente e aplicar o mesmo dicionário:
126 SELECT id_filme, titulo
127 FROM filmes
128 WHERE titulo_tsv @@ plainto_tsquery('portuguese', 'senhor anéis');
129
130 -- falhou!!!
131
```

Data Output Messages Notifications



|  |                                 |                       |
|--|---------------------------------|-----------------------|
|  | <b>id_filme</b><br>[PK] integer | <b>titulo</b><br>text |
|--|---------------------------------|-----------------------|

# Indexação Lista Invertida

Query    Query History

```
124 -- Se o usuário digitar "anéis", você pode aplicar plainto_tsquery,
125 -- que tenta tokenizar automaticamente e aplicar o mesmo dicionário:
126 SELECT id_filme, titulo
127   FROM filmes
128 WHERE titulo_tsv @@ plainto_tsquery('portuguese', 'senhor anéis');
129
130 -- falhou!!!
131
132 -- O que foi indexado (mostra os lexemas por título)
133 SELECT id_filme, titulo, to_tsvector('portuguese', titulo) AS tsv_calc,
134           titulo_tsv
135   FROM filmes;
136
```

Data Output    Messages    Notifications

≡+

|   | id_filme<br>[PK] integer | titulo<br>text                        | tsv_calc<br>tsvector                     | titulo_tsv<br>tsvector   |
|---|--------------------------|---------------------------------------|--|--|
| 1 | 1                        | Matrix                                | 'matrix':1                               | 'matrix':1   |
| 2 | 2                        | Matrix Reloaded                       | 'matrix':1 'reloaded':2                  | 'matrix':1 'reloaded':2  |
| 3 | 3                        | Matrix Revolutions                    | 'matrix':1 'revolutions':2               | 'matrix':1 'revolutions':2                                       |
| 4 | 4                        | Senhor dos Anéis: A Sociedade do Anel | 'anel':7 'ané':3 'senhor':1 'sociedad':5 | 'a':4 'anel':7 'anéis':3 'do':6 'dos':2 'senhor':1 'sociedade':5 |
| 5 | 5                        | O Senhor dos Anéis: As Duas Torres    | 'ané':4 'duas':6 'senhor':2 'torr':7     | 'anéis':4 'as':5 'dos':3 'duas':6 'o':1 'senhor':2 'torres':7    |
| 6 | 6                        | O Senhor dos Anéis: O Retorno do Rei  | 'ané':4 'rei':8 'retorn':6 'senhor':2    | 'anéis':4 'do':7 'dos':3 'o':1,5 'rei':8 'retorno':6 'senhor':2  |

# Indexação Lista Invertida

```
169
170 -- O que foi indexado (mostra os lexemas por título)
169
170 -- O que foi indexado (mostra os lexemas por título)
171 SELECT id_filme, titulo, to_tsvector('portuguese', titulo) AS tsv_calc,
172     titulo_tsv
173 FROM filmes;
174
175 -- Como o parser trata "senhor anéis"
176 SELECT plainto_tsquery('portuguese', 'senhor anéis') AS q1,
177     to_tsquery('portuguese', 'senhor & anéis')      AS q2;
178
179 -- Depuração detalhada do termo "anéis"
180 SELECT * FROM ts_debug('portuguese', 'anéis');
181
```

Data output Messages



|   | q1<br>tsquery    | q2<br>tsquery    |
|---|------------------|------------------|
| 1 | 'senhor' & 'ané' | 'senhor' & 'ané' |

# Indexação Lista Invertida

```
174
175 -- Como o parser trata “senhor anéis”
176 SELECT plainto_tsquery('portuguese', 'senhor anéis') AS q1,
177      to_tsquery('portuguese', 'senhor & anéis')      AS q2;
178
179 -- Depuração detalhada do termo “anéis”
180 SELECT * FROM ts_debug('portuguese', 'anéis');
181
```

Data output Messages



|   | alias<br>text | description<br>text | token<br>text | dictionaries<br>regdictionary[] | dictionary<br>regdictionary | lexemes<br>text[] |
|---|---------------|---------------------|---------------|---------------------------------|-----------------------------|-------------------|
| 1 | word          | Word, all letters   | anéis         | {portuguese_stem}               | portuguese_stem             | {ané}             |

# Indexação Lista Invertida

```
179 -- Depuração detalhada do termo “anéis”
180 SELECT * FROM ts_debug('portuguese', 'anéis');
181
182 -- 1) Extensão para remover acentos
183 CREATE EXTENSION IF NOT EXISTS unaccent;
184
```

Data output Messages

CREATE EXTENSION

Query returned successfully in 553 msec.

# Indexação Lista Invertida

```
184
185 -- 2) Crie uma configuração de busca que primeiro remove acento e depois aplica o stemmer
186 DROP TEXT SEARCH CONFIGURATION IF EXISTS portuguese_unaccent;
187 CREATE TEXT SEARCH CONFIGURATION portuguese_unaccent ( COPY = portuguese );
188 ALTER TEXT SEARCH CONFIGURATION portuguese_unaccent
189   ALTER MAPPING FOR hword, hword_part, word
190   WITH unaccent, portuguese_stem;
191
```

Data output    Messages

```
DROP TEXT SEARCH CONFIGURATION
```

```
Query returned successfully in 138 msec.
```

# Indexação Lista Invertida

```
184
185 -- 2) Crie uma configuração de busca que primeiro remove acento e depois aplica o stemmer
186 DROP TEXT SEARCH CONFIGURATION IF EXISTS portuguese_unaccent;
187 CREATE TEXT SEARCH CONFIGURATION portuguese_unaccent ( COPY = portuguese );
188 ALTER TEXT SEARCH CONFIGURATION portuguese_unaccent
189   ALTER MAPPING FOR hword, hword_part, word
190   WITH unaccent, portuguese_stem;
191
```

Data output    Messages

```
CREATE TEXT SEARCH CONFIGURATION
```

```
Query returned successfully in 125 msec.
```

# Indexação Lista Invertida

```
184
185 -- 2) Crie uma configuração de busca que primeiro remove acento e depois aplica o stemmer
186 DROP TEXT SEARCH CONFIGURATION IF EXISTS portuguese_unaccent;
187 CREATE TEXT SEARCH CONFIGURATION portuguese_unaccent ( COPY = portuguese );
188 ALTER TEXT SEARCH CONFIGURATION portuguese_unaccent
189   ALTER MAPPING FOR hword, hword_part, word
190   WITH unaccent, portuguese_stem;
191
```

Data output    Messages

---

```
ALTER TEXT SEARCH CONFIGURATION
```

```
Query returned successfully in 109 msec.
```

# Indexação Lista Invertida

```
191
192 -- 3) Recalcule a coluna TSVECTOR usando a nova config
193 UPDATE filmes
194 SET titulo_tsv = to_tsvector('portuguese_unaccent', titulo);
195
```

Data output    Messages

UPDATE 6

Query returned successfully in 106 msec.

# Indexação Lista Invertida

```
193 UPDATE filmes
194 SET titulo_tsv = to_tsvector('portuguese_unaccent', titulo);
195
196 -- 4) (Opcional) GIN index na coluna
197 DROP INDEX IF EXISTS idx_filmes_titulo_tsv;
198 CREATE INDEX idx_filmes_titulo_tsv
199 ON filmes USING gin (titulo_tsv);
200
```

Data output    Messages

CREATE INDEX

Query returned successfully in 118 msec.

# Indexação Lista Invertida

```
... , ... , ... , ... , ... ,  
200  
201 -- Versão “plain text” (boa para input de usuário)  
202 SELECT id_filme, titulo  
203 FROM filmes  
204 WHERE titulo_tsv @@ plainto_tsquery('portuguese_unaccent', 'senhor anéis');  
205
```

Data output Messages



|   | <b>id_filme</b><br>[PK] integer | <b>titulo</b><br>text                 |
|---|---------------------------------|---------------------------------------|
| 1 | 4                               | Senhor dos Anéis: A Sociedade do Anel |
| 2 | 5                               | O Senhor dos Anéis: As Duas Torres    |
| 3 | 6                               | O Senhor dos Anéis: O Retorno do Rei  |

# Indexação Lista Invertida

```
-- Versão "plain text" (boa para input de usuário)
SELECT id_filme, titulo
FROM filmes
WHERE titulo_tsv @@ plainto_tsquery('portuguese_unaccent', 'senhor anéis');

-- Versão estilo pesquisa web (aceita espaços, aspas, OR, -)
SELECT id_filme, titulo
FROM filmes
WHERE titulo_tsv @@ websearch_to_tsquery('portuguese_unaccent', 'senhor anéis');
```

Data output Messages



|   | id_filme<br>[PK] integer | titulo<br>text                        |
|---|--------------------------|---------------------------------------|
| 1 | 4                        | Senhor dos Anéis: A Sociedade do Anel |
| 2 | 5                        | O Senhor dos Anéis: As Duas Torres    |
| 3 | 6                        | O Senhor dos Anéis: O Retorno do Rei  |

# Indexação Lista Invertida

```
210
211 -- Trigger function
212 CREATE OR REPLACE FUNCTION filmes_tsv_trigger()
213 RETURNS trigger AS $$ 
214 BEGIN
215   NEW.titulo_tsv := to_tsvector('portuguese_unaccent', NEW.titulo);
216   RETURN NEW;
217 END
218 $$ LANGUAGE plpgsql;
219
```

Data output    Messages

CREATE FUNCTION

Query returned successfully in 134 msec.

# Indexação Lista Invertida

```
212 CREATE OR REPLACE FUNCTION filmes_tsv_trigger()
213 RETURNS trigger AS $$ 
214 BEGIN
215     NEW.titulo_tsv := to_tsvector('portuguese_unaccent', NEW.titulo);
216     RETURN NEW;
217 END
218 $$ LANGUAGE plpgsql;
219
220 -- Trigger
221 DROP TRIGGER IF EXISTS tsv_update ON filmes;
222 CREATE TRIGGER tsv_update
223 BEFORE INSERT OR UPDATE OF titulo ON filmes
224 FOR EACH ROW EXECUTE FUNCTION filmes_tsv_trigger();
```

Data output Messages

CREATE TRIGGER

Query returned successfully in 122 msec.

# Indexação Lista Invertida

```
225  
226  
227 INSERT INTO filmes (titulo)  
228 VALUES ('Harry Potter e a Pedra Filosofal');  
229
```

Data output Messages

```
INSERT 0 1
```

Query returned successfully in 125 msec.

# Indexação Lista Invertida

```
225
226
227 INSERT INTO filmes (titulo)
228 VALUES ('Harry Potter e a Pedra Filosofal');
229
230 SELECT id_filme, titulo, titulo_tsv
231 FROM filmes
232 WHERE titulo LIKE 'Harry Potter%';
233
```

Data output Messages

+

|  | <b>id_filme</b><br>[PK] integer | <b>titulo</b><br>text            | <b>titulo_tsv</b><br>tsvector             |
|--|---------------------------------|----------------------------------|---|
|  | 7                               | Harry Potter e a Pedra Filosofal | 'filosofal':6 'harry':1 'pedr':5 'pott':2 |

# Indexação Lista Invertida

```
230 SELECT id_filme, titulo, titulo_tsv
231 FROM filmes
232 WHERE titulo LIKE 'Harry Potter%';
233
234 UPDATE filmes
235 SET titulo = 'Harry Potter e a Câmara Secreta'
236 WHERE titulo LIKE 'Harry Potter%';
237
```

Data output    Messages

UPDATE 1

Query returned successfully in 140 msec.

# Indexação Lista Invertida

```
233
234 UPDATE filmes
235 SET titulo = 'Harry Potter e a Câmara Secreta'
236 WHERE titulo LIKE 'Harry Potter%';
237
238 SELECT id_filme, titulo, titulo_tsv
239 FROM filmes
240 WHERE titulo LIKE 'Harry Potter%';
241
242
```

Data output    Messages



|  | <b>id_filme</b><br>[PK] integer | <b>titulo</b><br>text           | <b>titulo_tsv</b><br>tsvector         |
|--|---------------------------------|---------------------------------|---------------------------------------|
|  | 7                               | Harry Potter e a Câmara Secr... | 'cam':5 'harry':1 'pott':2 'secret':6 |

# Indexação Lista Invertida

```
237
238 SELECT id_filme, titulo, titulo_tsv
239 FROM filmes
240 WHERE titulo LIKE 'Harry Potter%';
241
242 -- Testando a busca FULL TEXT
243 SELECT id_filme, titulo
244 FROM filmes
245 WHERE titulo_tsv @@ plainto_tsquery('portuguese_unaccent', 'camara secreta');
246
```

Data output Messages



|   | id_filme<br>[PK] integer | titulo<br>text                  |
|---|--------------------------|---------------------------------|
| 1 | 7                        | Harry Potter e a Câmara Secreta |

Deve encontrar “Harry Potter e a Câmara Secreta” mesmo que você não digite acento.

# Busca Full Text

- Uma **busca Full Text** (ou **busca textual completa**) é uma técnica de pesquisa em bancos de dados que permite localizar palavras e frases dentro de campos de texto, de forma inteligente, levando em conta:
  - Ocorrência de palavras (não apenas igualdade exata).
  - Variações linguísticas (singular/plural, conjugação, acentos).
  - Relevância (ordenação por quanto bem o texto corresponde à busca).
- Diferente de um **LIKE '%palavra%'**, que faz apenas comparação literal, a **busca Full Text** é otimizada com listas invertidas (índices **GIN** ou **GiST** no PostgreSQL) e dicionários linguísticos (stemming).

# Busca Full Text

```
247  
248  
249 CREATE TABLE artigos (  
250     id SERIAL PRIMARY KEY,  
251     conteudo TEXT NOT NULL  
252 );  
253
```

Data output Messages

CREATE TABLE

Query returned successfully in 142 msec.

# Busca Full Text

```
248 -----
249 CREATE TABLE artigos (
250     id SERIAL PRIMARY KEY,
251     conteudo TEXT NOT NULL
252 );
253
254 INSERT INTO artigos (conteudo) VALUES
255 ('O Senhor dos Anéis é uma trilogia épica de fantasia.'),
256 ('Matrix é um clássico da ficção científica.'),
257 ('Harry Potter e a Pedra Filosofal é o início da saga.');
258
```

Data output    Messages

INSERT 0 3

Query returned successfully in 137 msec.

# Busca Full Text

```
253  
254 INSERT INTO artigos (conteudo) VALUES  
255 ('O Senhor dos Anéis é uma trilogia épica de fantasia.'),  
256 ('Matrix é um clássico da ficção científica.'),  
257 ('Harry Potter e a Pedra Filosofal é o início da saga.');  
258  
259 -- CRIAR UMA COLUNA INDEXADA PARA FULL TEXT.  
260 ALTER TABLE artigos  
261 ADD COLUMN conteudo_tsv tsvector;  
262  
263 UPDATE artigos  
264 SET conteudo_tsv = to_tsvector('portuguese', conteudo);  
265  
266 CREATE INDEX idx_artigos_tsv  
267 ON artigos USING gin (conteudo_tsv);  
268
```

Data output    Messages

CREATE INDEX

Query returned successfully in 136 msec.

# Busca Full Text

```
268  
269 -- Busca por "Matrix"  
270 SELECT id, conteudo  
271 FROM artigos  
272 WHERE conteudo_tsv @@ to_tsquery('portuguese', 'matrix');  
273  
274
```

Data output    Messages



|   | <b>id</b><br>[PK] integer | <b>conteudo</b><br>text                    |
|---|---------------------------|--|
| 1 | 2                         | Matrix é um clássico da ficção científica. |

# Busca Full Text

## Vantagens sobre LIKE

LIKE ( WHERE conteúdo LIKE '%Matrix%' )

FULL TEXT

Faz varredura sequencial (lento em textos grandes).

Usa índice GIN/GiST (rápido).

Não entende plural/variações.

Entende "carro" ≈ "carros".

Não ordena por relevância.

Pode ordenar pelo grau de correspondência ( ts\_rank ).

Simples, mas limitado.

Poderoso para buscas textuais complexas.

# Outro Exemplo Lista Invertida

```
273
274 -----
275 -- Outro Exemplo de -----
276 -- Lista Invertida -----
277 -----
278 -----
279 CREATE TABLE funcionario (
280     matricula    SERIAL PRIMARY KEY,
281     nome         TEXT NOT NULL,
282     sexo         CHAR(1) CHECK (sexo IN ('M','F')),
283     id_setor    INT NOT NULL,
284     id_cargo    INT NOT NULL
285 );
286
287
```

Data output    Messages

CREATE TABLE

Query returned successfully in 78 msec.

# Outro Exemplo Lista Invertida

```
279 CREATE TABLE funcionario (
280     matricula    SERIAL PRIMARY KEY,
281     nome         TEXT NOT NULL,
282     sexo         CHAR(1) CHECK (sexo IN ('M','F')),
283     id_setor    INT NOT NULL,
284     id_cargo    INT NOT NULL
285 );
286
287 INSERT INTO funcionario (nome, sexo, id_setor, id_cargo) VALUES
288 ('João da Silva', 'M', 1, 101),
289 ('Maria Oliveira', 'F', 1, 102),
290 ('José Santos', 'M', 2, 201),
291 ('Ana Beatriz', 'F', 2, 202),
292 ('Carlos Mendes', 'M', 3, 301),
293 ('Beatriz Souza', 'F', 3, 302);
294
```

Data output    Messages

INSERT 0 6

Query returned successfully in 55 msec.

# Outro Exemplo Lista Invertida

Exemplo simples da tabela FUNCIONARIO:

| MATRICULA | NOME          | SEXO | ID_SETOR | ID_CARGO |
|-----------|---------------|------|----------|----------|
| 1         | João da Silva | M    | 1        | 101      |
| 2         | Maria Olive.  | F    | 1        | 102      |
| 3         | José Santos   | M    | 2        | 201      |
| 4         | Ana Beatriz   | F    | 2        | 202      |
| 5         | Carlos Mendes | M    | 3        | 301      |
| 6         | Beatriz Souza | F    | 3        | 302      |

# Outro Exemplo Lista Invertida

A lista invertida poderia ser construída assim:

## ◆ Para SEXO

- M → {1, 3, 5}
- F → {2, 4, 6}

## ◆ Para ID\_SETOR

- 1 → {1, 2}
- 2 → {3, 4}
- 3 → {5, 6}

## ◆ Para ID\_CARGO

- 101 → {1}
- 102 → {2}
- 201 → {3}
- 202 → {4}
- 301 → {5}
- 302 → {6}

# Outro Exemplo Lista Invertida

- Se você quiser buscar “**todos os funcionários do setor 2 e do sexo F**”, o banco de dados não precisa varrer toda a tabela.
- Ele faz **interseção** de listas invertidas:
- **ID\_SETOR = 2** → {3, 4}
- **SEXO = F** → {2, 4, 6}
- **Interseção = {4}** → funcionário **Ana Beatriz**.

# Outro Exemplo Lista Invertida

```
--  
287 INSERT INTO funcionario (nome, sexo, id_setor, id_cargo) VALUES  
288 ('João da Silva', 'M', 1, 101),  
289 ('Maria Oliveira', 'F', 1, 102),  
290 ('José Santos', 'M', 2, 201),  
291 ('Ana Beatriz', 'F', 2, 202),  
292 ('Carlos Mendes', 'M', 3, 301),  
293 ('Beatriz Souza', 'F', 3, 302);  
294  
295 CREATE INDEX idx_funcionario_setor ON funcionario (id_setor);  
296 CREATE INDEX idx_funcionario_sexo ON funcionario (sexo);  
297 CREATE INDEX idx_funcionario_cargo ON funcionario (id_cargo);  
298
```

Data output    Messages

CREATE INDEX

Query returned successfully in 56 msec.

# Outro Exemplo Lista Invertida

```
294
295 CREATE INDEX idx_funcionario_setor ON funcionario (id_setor);
296 CREATE INDEX idx_funcionario_sexo  ON funcionario (sexo);
297 CREATE INDEX idx_funcionario_cargo ON funcionario (id_cargo);
298
299 SELECT nome
300 FROM funcionario
301 WHERE id_setor = 2 AND sexo = 'F';
302
```

Data output Messages



| nome |             |
|------|-------------|
|      | text        |
| 1    | Ana Beatriz |

# Indexação

Se tivermos que percorrer o **arquivo de índice** sequencialmente para encontrar uma determinada chave, o **índice** não terá muita utilidade:

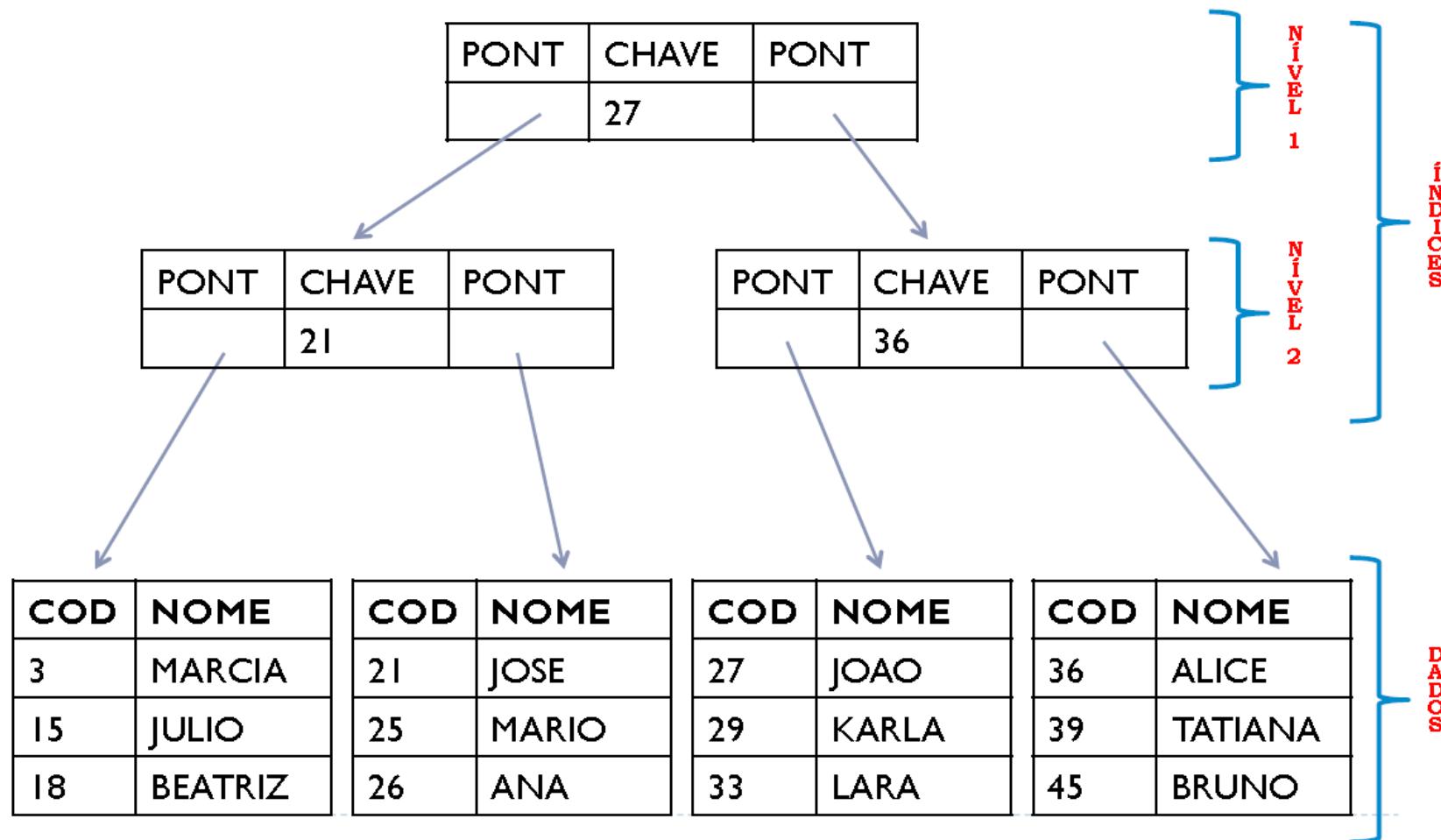
- Pode-se fazer busca um pouco mais eficiente (ex. busca binária), se o arquivo de índice estiver ordenado.
- Mas mesmo assim isso não é o ideal.

# Indexação

- Para resolver este problema:
  - os **índices** não são estruturas sequenciais, e sim hierárquicas.
  - os **índices** não apontam para um **registro específico**, mas para um **bloco de registros** (e dentro do bloco é feita **busca sequencial**) – exige que os **registros** dentro de um **bloco** estejam ordenados.

# Indexação

## Exemplo de Índice Hierárquico



# Indexação

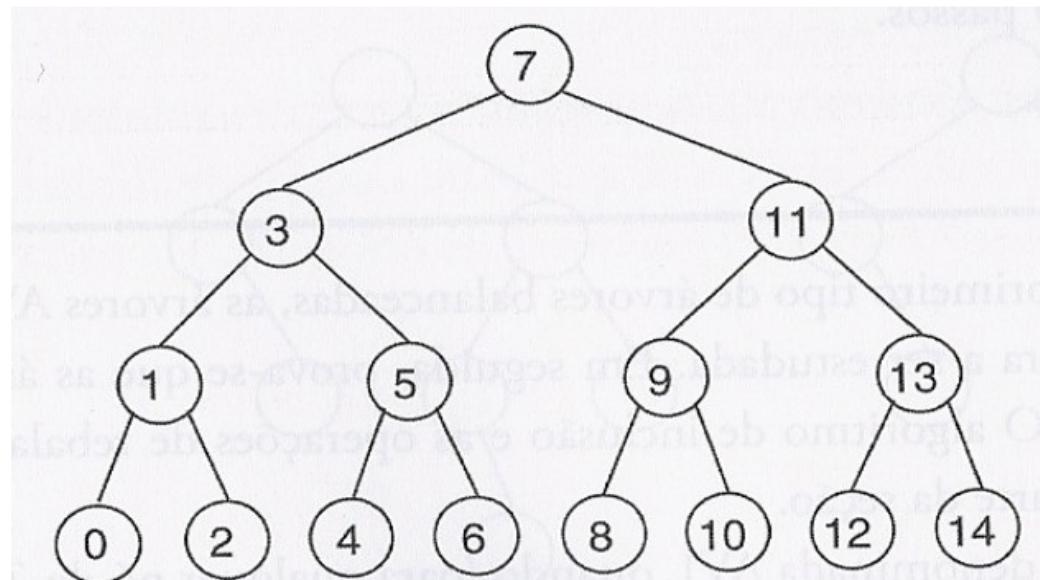
- A maioria das estruturas de índice é implementada por árvores de busca
  - Árvores Binárias
  - Árvores AVL
  - Árvores de Múltiplos Caminhos

# Indexação

## Árvore de Busca Binária

### ► Características de uma árvore de busca binária T

- ▶ todas as chaves da subárvore da esquerda de T têm valores menores que a chave do nó raiz de T
- ▶ todas as chaves da subárvore da direita de T têm valores maiores que a chave do nó raiz de T
- ▶ as subárvores esquerda e direita de T também são árvores de busca binária



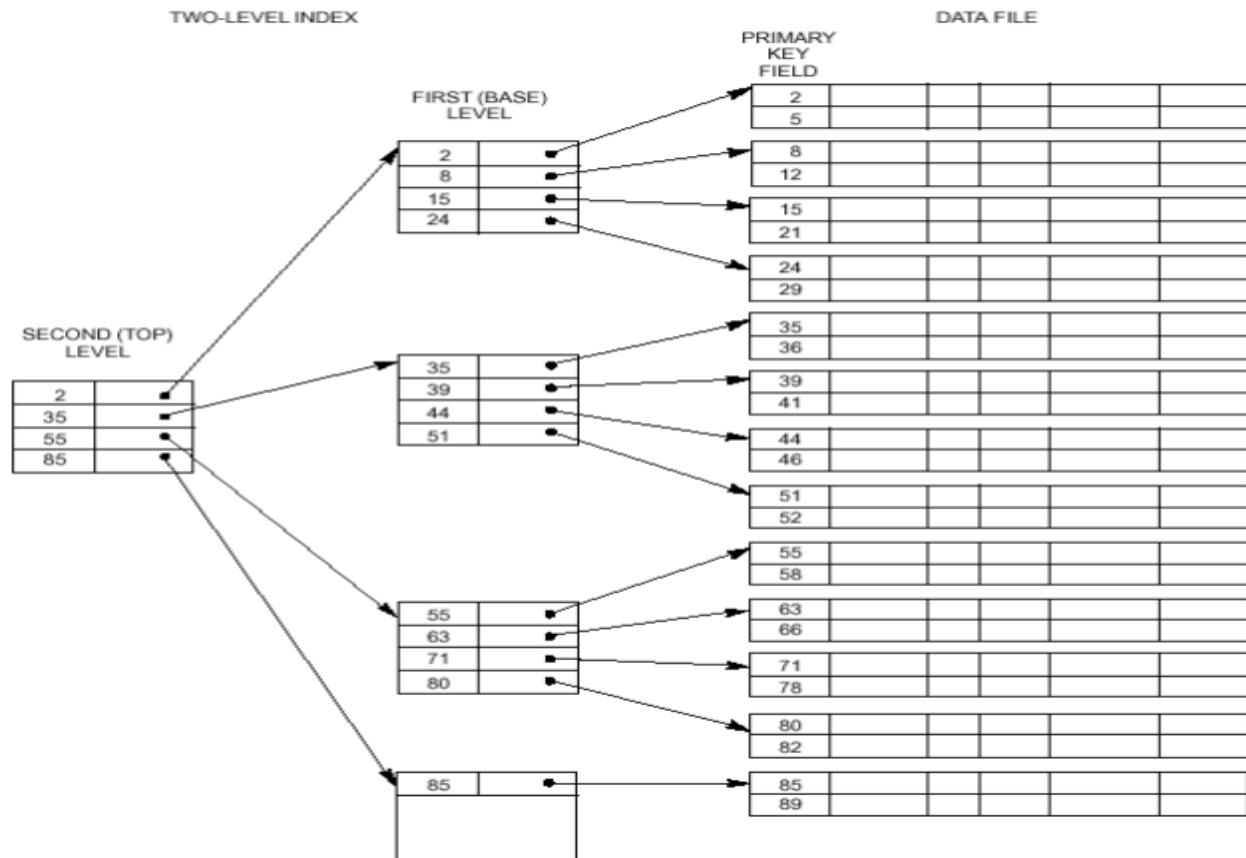
# Indexação Árvores de Múltiplos Caminhos

- Características:

- Cada nó contém  $n-1$  chaves
- Cada nó contém  $n$  filhos
- As chaves dentro do nó estão ordenadas
- As chaves dentro do nó funcionam como separadores para os ponteiros para os filhos do nó.

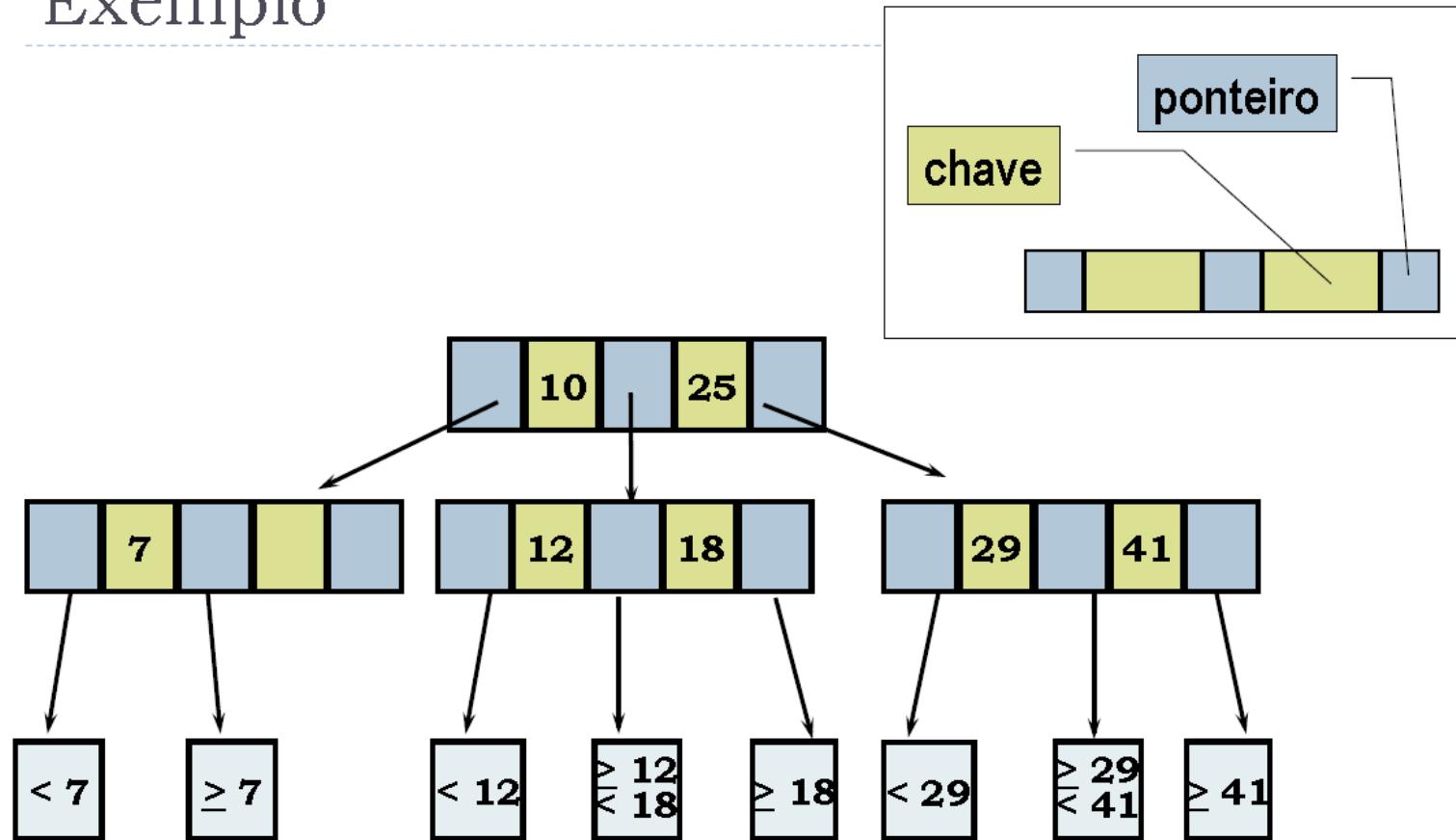
# Indexação Árvores de Múltiplos Caminhos

## Índices Multinível



# Indexação Árvores de Múltiplos Caminhos

Exemplo



# Indexação Árvores de Múltiplos Caminhos

- Vantagens
  - Têm altura bem menor que as árvores binárias.
  - Ideais para uso como índice de arquivos em disco.
  - Como as árvores são baixas, são necessários poucos acessos em disco até chegar ao ponteiro para o bloco que contém o registro desejado.
  - As árvores AVL, apesar de balanceadas, são excessivamente altas para uso eficiente como índice.

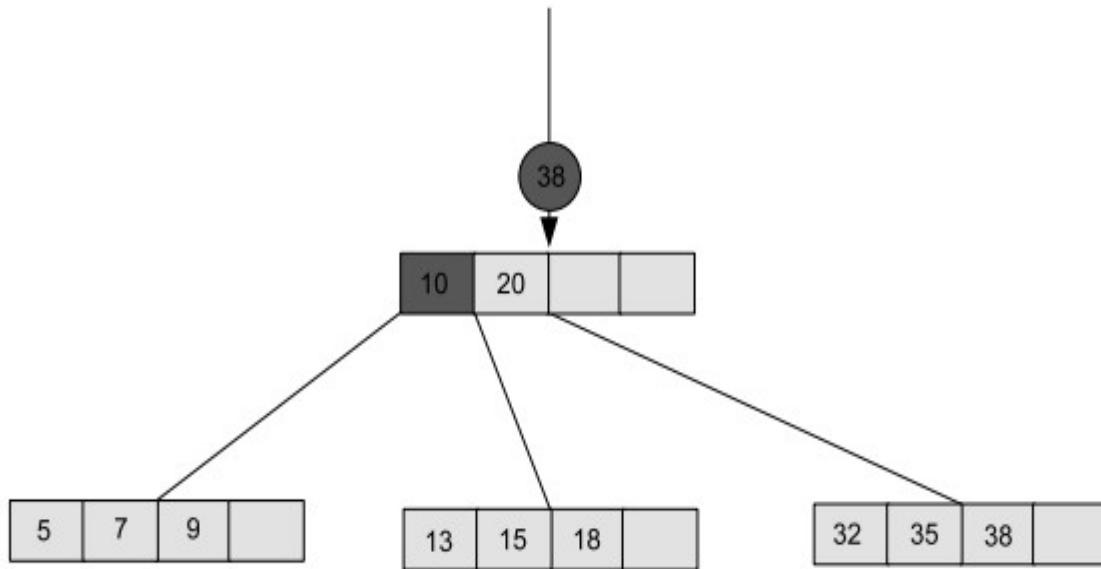
# Indexação Árvores de Múltiplos Caminhos

- Exemplos de Árvores Múltiplos Caminhos:
  - Árvore B
  - Árvore B\*
  - Árvore B+
  - Tries

# Busca em Árvores B

- A **busca** em uma **árvore-B** é **similar** à busca em uma **árvore binária**, só que ao invés de uma **bifurcação** em cada nó, temos **vários caminhos** a seguir de acordo com o **número de filhos** do nó.

# Busca em Árvores B

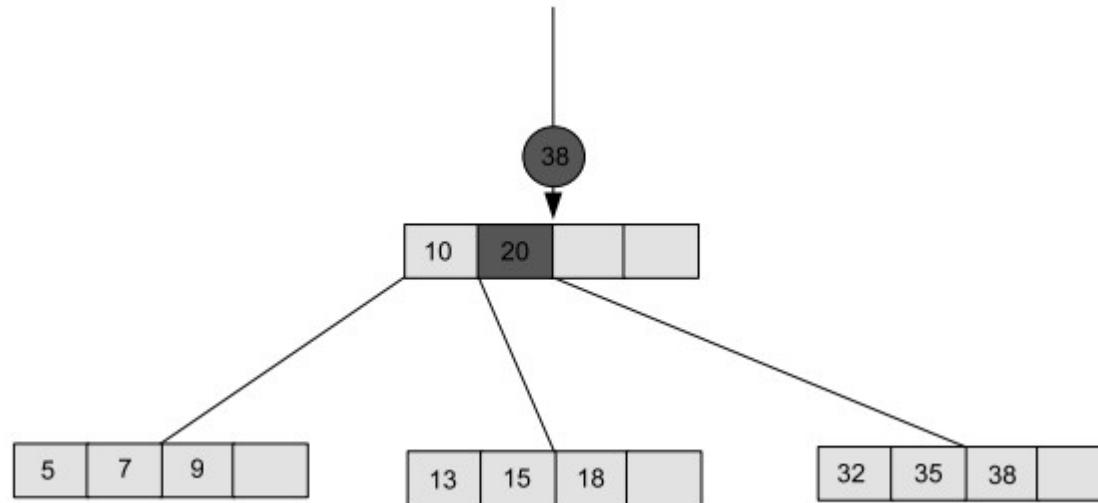


Considere a busca pelo elemento 38 na árvore acima.

A busca, como de praxe, começa pelo nó raiz.

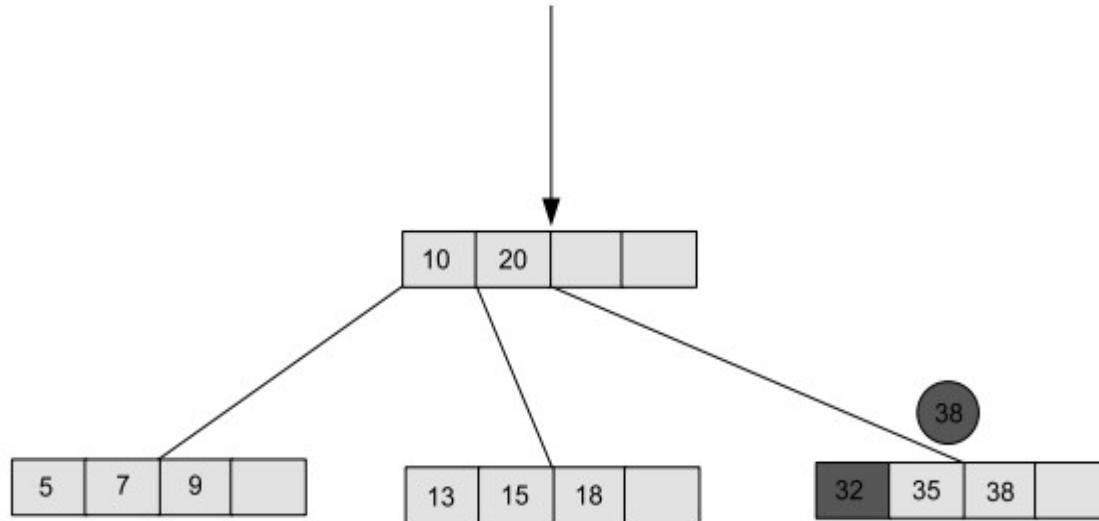
Diferente de uma árvore binária, qualquer nó de uma árvore-b pode apresentar mais de um valor. Nesse exemplo, temos dois valores (mas poderíamos ter até 4): 10 e 20.

# Busca em Árvores de Múltiplos Caminhos



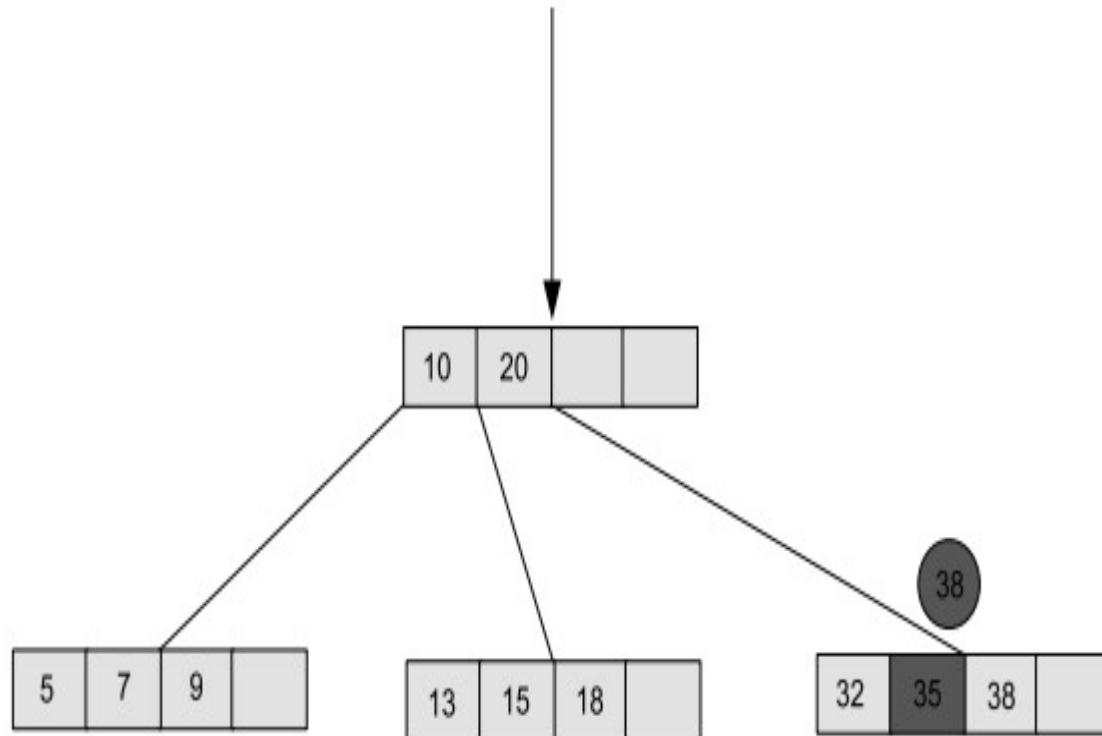
- A busca sempre começa por uma visita no valor mais à esquerda do nó.
- Como 38 é maior que 10 e existe um outro valor à direita de 10, comparamos também 38 com 20.

# Busca em Árvores de Múltiplos Caminhos

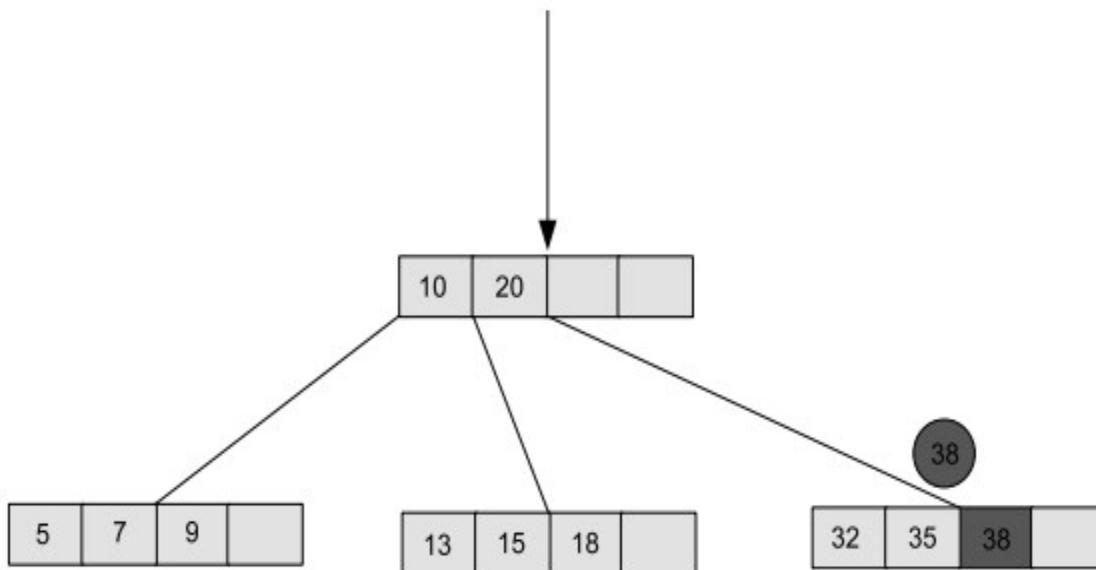


- Como 38 é maior que 20 “descemos” pelo ponteiro mais à direita de 20.
- Em seguida, iniciamos a busca pelo elemento 38 no outro nó.

# Busca em Árvores de Múltiplos Caminhos

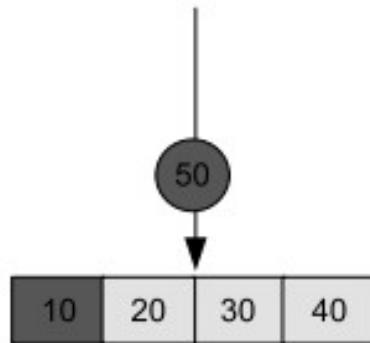


# Busca em Árvores B



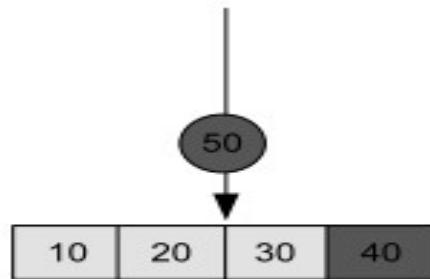
- Elemento 38 encontrado!

# Inserção em Árvores B



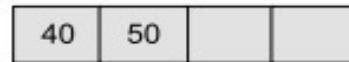
- Inserção do elemento 50 em um nó folha (que também é a raiz e está lotado): não há a possibilidade de inclusão de um quinto elemento no nó.

# Inserção em Árvores B



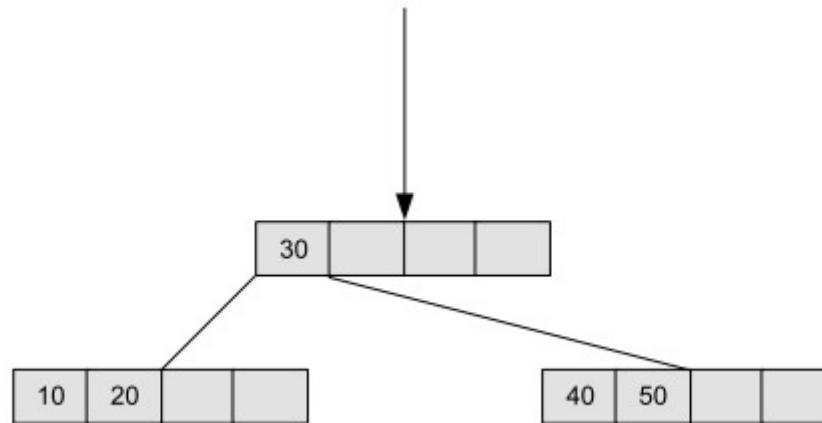
- O nó em questão é percorrido e descobre-se que o elemento 50 deveria ser inserido na quinta posição do nó (inexistente).

# Inserção em Árvores B



- O conteúdo do nó originário é distribuído igualitariamente entre este e um nó recém criado.
- 10 e 20 ficam no nó original enquanto 40 e 50 vão para o novo nó. O elemento do meio (terceira posição), o 30, é guardado para outro uso.

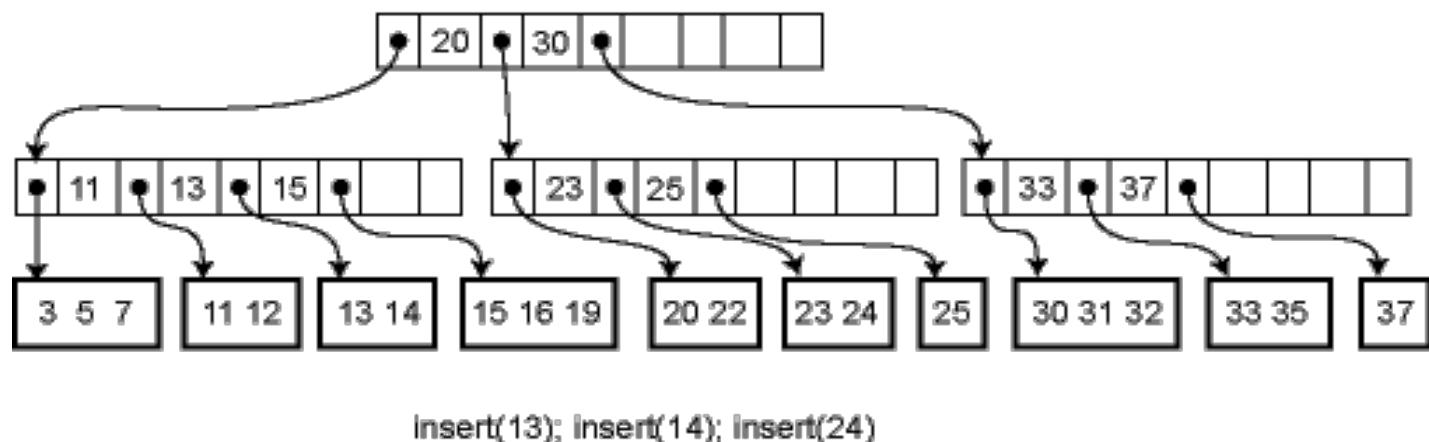
# Inserção em Árvores B



- Um terceiro nó, posicionado hierarquicamente em uma camada acima, torna-se a nova raiz e recebe o elemento 30.

# Árvore B

- Árvores B são uma **generalização das árvores binária de busca**, pois cada nó de uma árvore binária armazena uma única chave de busca, enquanto **as árvores B armazenam um número maior do que um de chaves de busca em cada nó**, ou no termo mais usual para essa árvore, **em cada página**.



# Árvore B

- A **ordem** de uma **árvore-B** é dada pelo número máximo de descendentes que uma página, ou nó, pode possuir.
- Em uma **árvore-B** de **ordem m**, o **número máximo de chaves** em uma **página** é **m-1**.
- **Exemplo:** Uma **árvore-B** de **ordem 8** tem, no máximo, **7 chaves** por **página**.

# Árvore B

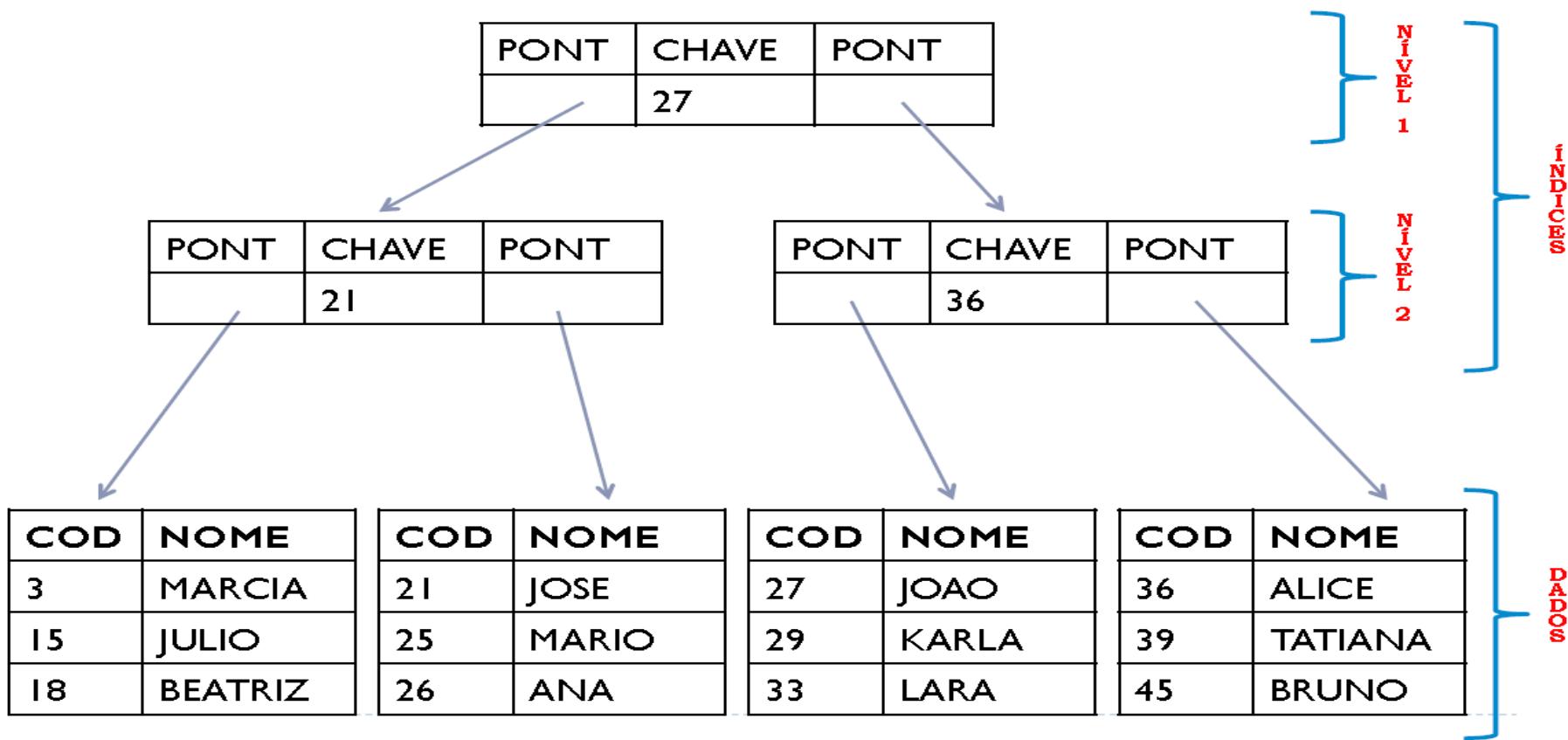
- Quando uma **página** é **sub-dividida** na **inserção**, as **chaves** são divididas (quase) igualmente entre as **páginas velha e nova**.
- Deste modo, o **número mínimo de chaves** em um **nó** é dado por  **$m/2 - 1$**  (exceto para a raiz).
- **Exemplo:** árvore B de **ordem 8**, armazena no **máximo 7 chaves por página** e tem, **no mínimo, 3 chaves por página**.
- **Nós folha:** alocados no nível mais baixo da árvore

# Árvore B

- Para uma **árvore-B** de **ordem m**:
- **1.** cada página tem, no máximo, **m descendentes**.
- **2.** cada **página, exceto a raiz e as folhas**, tem no mínimo  **$m/2$  descendentes**.
- **3.** a **raiz** tem, **no mínimo, dois descendentes – a menos que seja uma folha**.
- **4.** todas as **folhas** estão no **mesmo nível**.
- **5.** uma **página não folha** que possui **k descendentes** contém  **$k-1$  chaves**.
- **6.** uma **página folha** contém, **no mínimo  $m/2 -1$**  e, **no máximo,  $m-1$  chaves**.

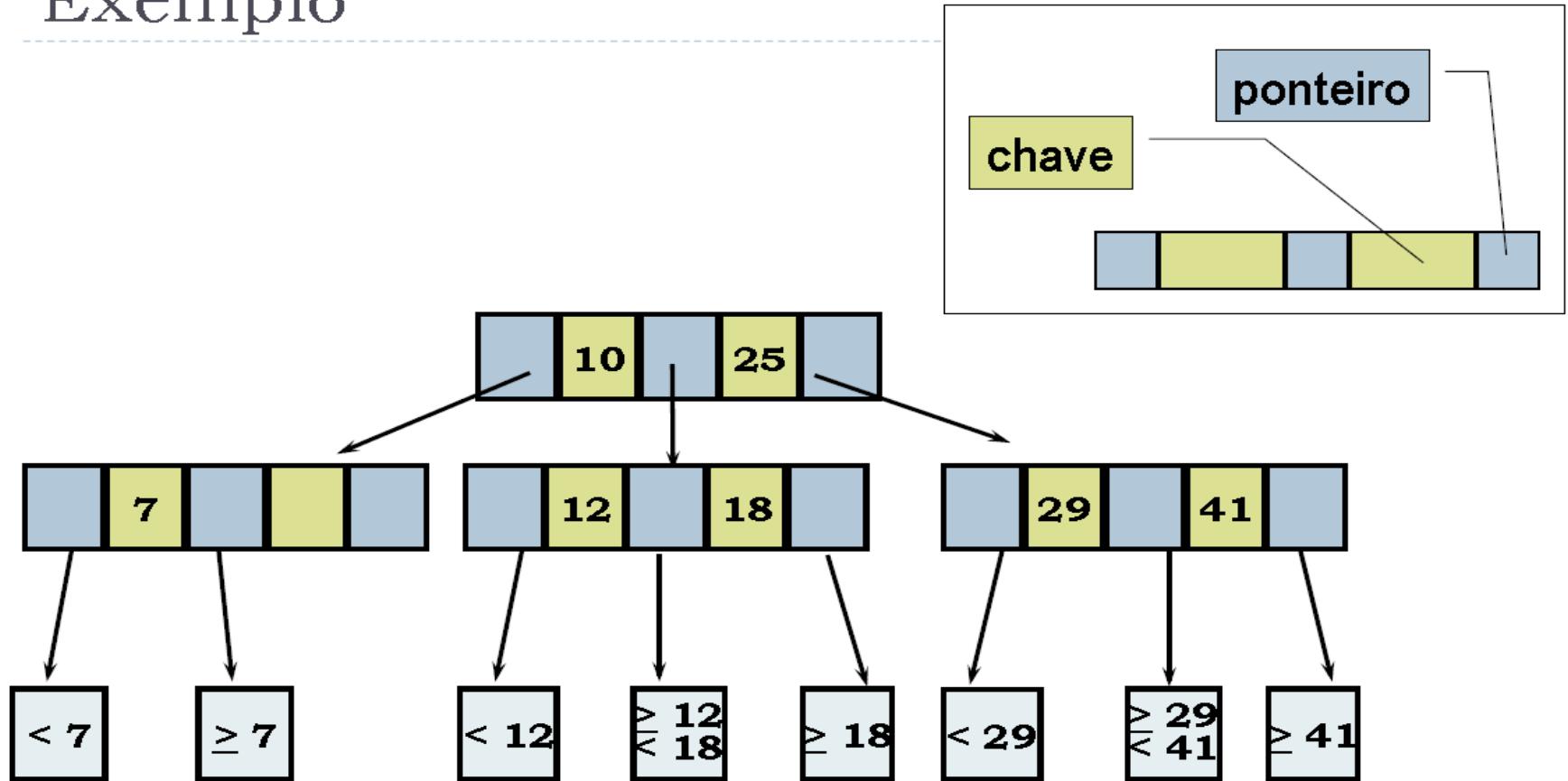
# Árvore B

Exemplo de Índice Hierárquico



# Árvore B

Exemplo



# Árvore B

## Vantagens:

- Têm altura bem menor que as árvores binárias.
- Ideais para uso como índice de arquivos em disco.
- Como as árvores são baixas, são necessários poucos acessos em disco até chegar ao ponteiro para o bloco que contém o registro desejado.
- As árvores AVL, apesar de平衡adas, são excessivamente altas para uso eficiente como índice.

# Árvores de Múltiplos Caminhos

## Árvores B

### Exemplo

- Insira as seguintes chaves em um índice árvore-B
  - C S D T A M P I B W N G U R K E H O L J Y  
Q Z F X V
- Ordem da árvore-B: 4
  - em cada nó (página de disco)
    - número de chaves: 3
    - número de ponteiros: 4

# Árvores de Múltiplos Caminhos

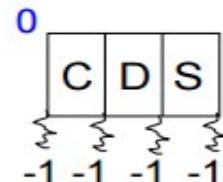
## Árvores B

C S D T A M P I B W N G U R K ...

- Passo 1 – inserção de C, S, D

– criação do nó raiz

- C
- C S
- C D S



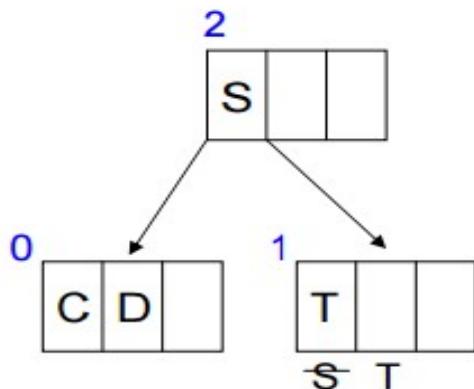
# Árvores de Múltiplos Caminhos

## Árvores B

C S D T A M P I B W N G U R K ...

- Passo 2 – inserção de T
  - nó raiz cheio

- particionamento do nó
- criação de uma nova raiz
- promoção de S

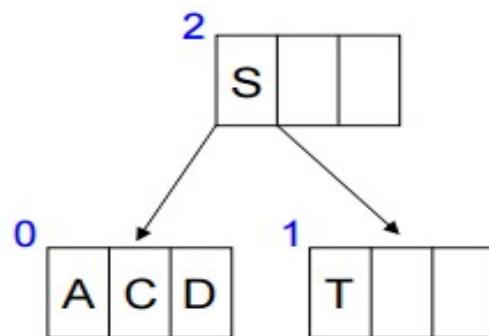


# Árvores de Múltiplos Caminhos

## Árvores B

C S D T A M P I B W N G U R K ...

- Passo 3 – inserção de A
  - nó folha com espaço



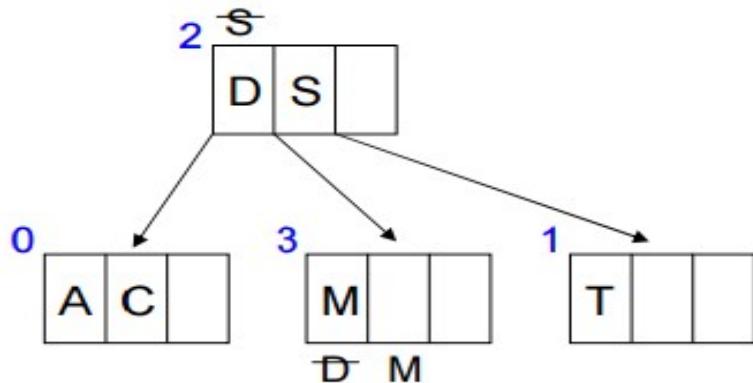
# Árvores de Múltiplos Caminhos

## Árvores B

C S D T A M P I B W N G U R K ...

- Passo 4 – inserção de M
  - nó folha 0 cheio

- particionamento do nó
- promoção de D

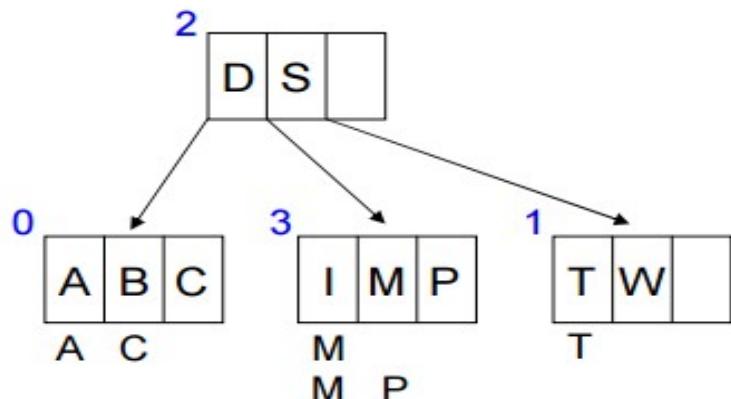


# Árvores de Múltiplos Caminhos

## Árvores B

C S D T A M P I B W N G U R K ...

- Passo 5 – inserção de P, I, B, W  
– nós folhas com espaço



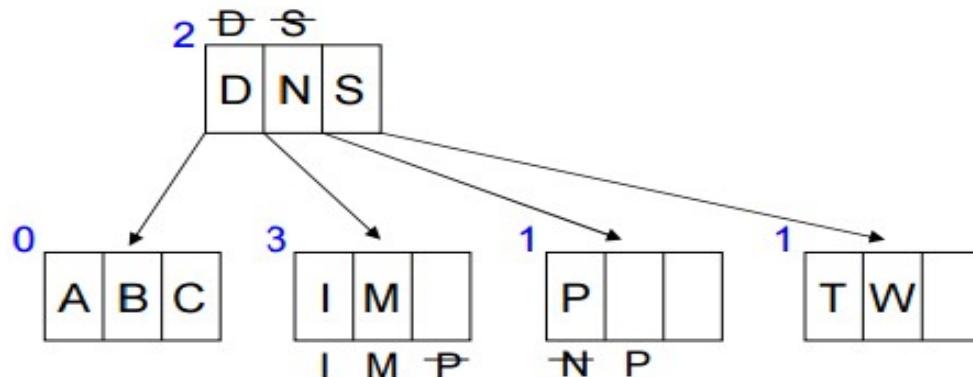
# Árvores de Múltiplos Caminhos

## Árvores B

C S D T A M P I B W N G U R K ...

- Passo 6 – inserção de N
  - nó folha 3 cheio

- particionamento do nó
- promoção de N

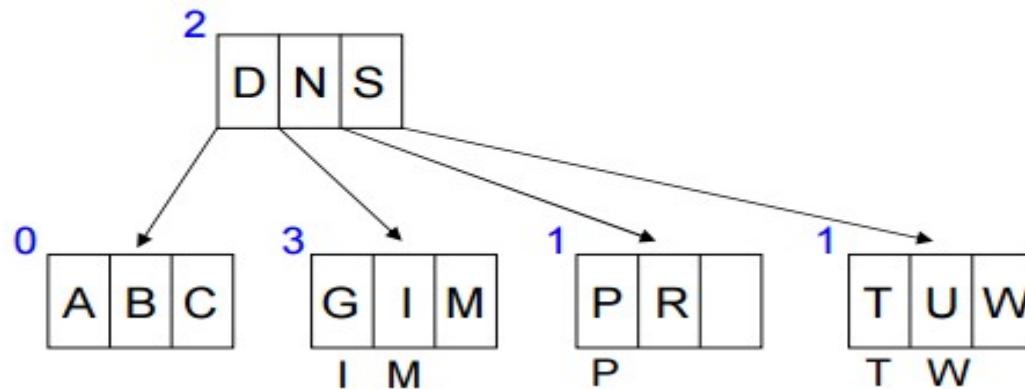


# Árvores de Múltiplos Caminhos

## Árvores B

C S D T A M P I B W N G U R K ...

- Passo 7 – inserção de G, U, R  
– nós folhas com espaço



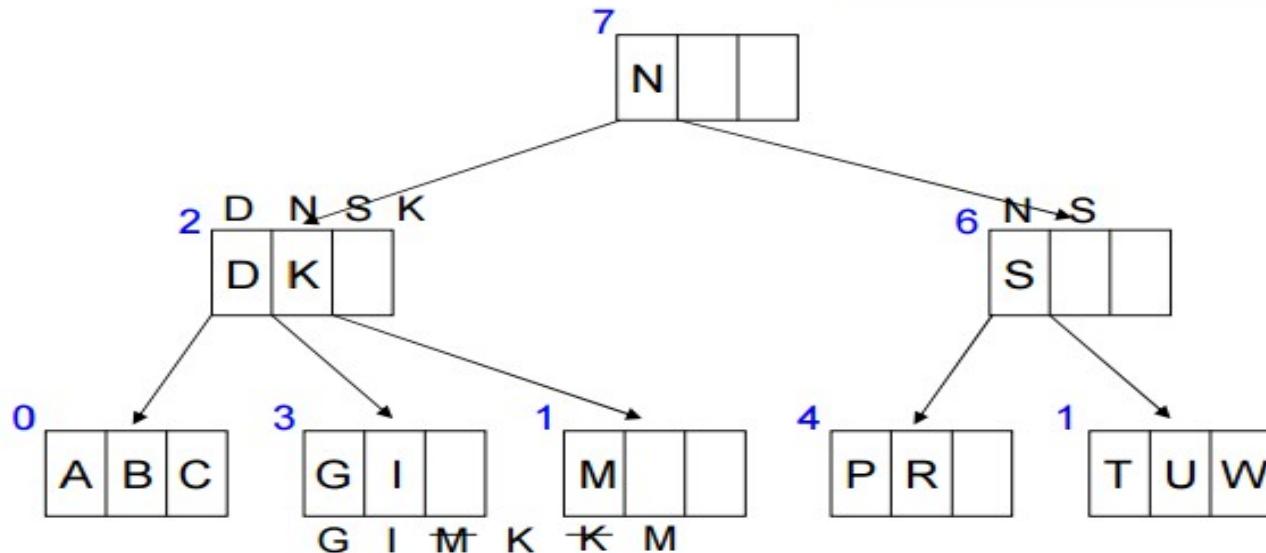
# Árvores de Múltiplos Caminhos

## Árvores B

C S D T A M P I B W N G U R K ...

- Passo 8 – inserção de K
  - nó folha 3 cheio

- particionamento do nó 3
- promoção de K
- particionamento do nó 2
- promoção de N



# Árvores B

## Remoção

- **Exclusão**

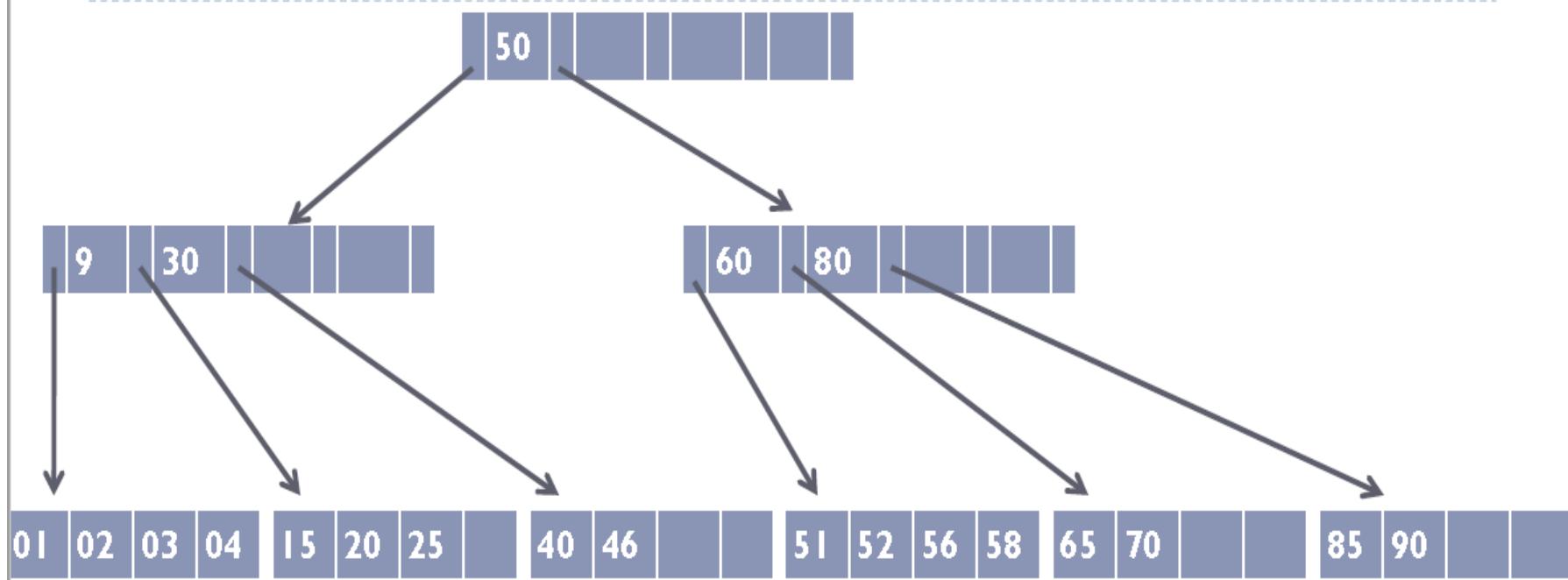
- Duas situações possíveis:

- [1] - A entrada  $x$  está em um nó folha: Neste caso, simplesmente remover a entrada  $x$
- [2] - A entrada  $x$  não está em um nó folha:
  - Substituir  $x$  pela chave  $y$  imediatamente maior.
  - Note que  $y$  necessariamente pertence a uma folha, pela forma como a árvore B é estruturada.

# Árvores B

## Remoção

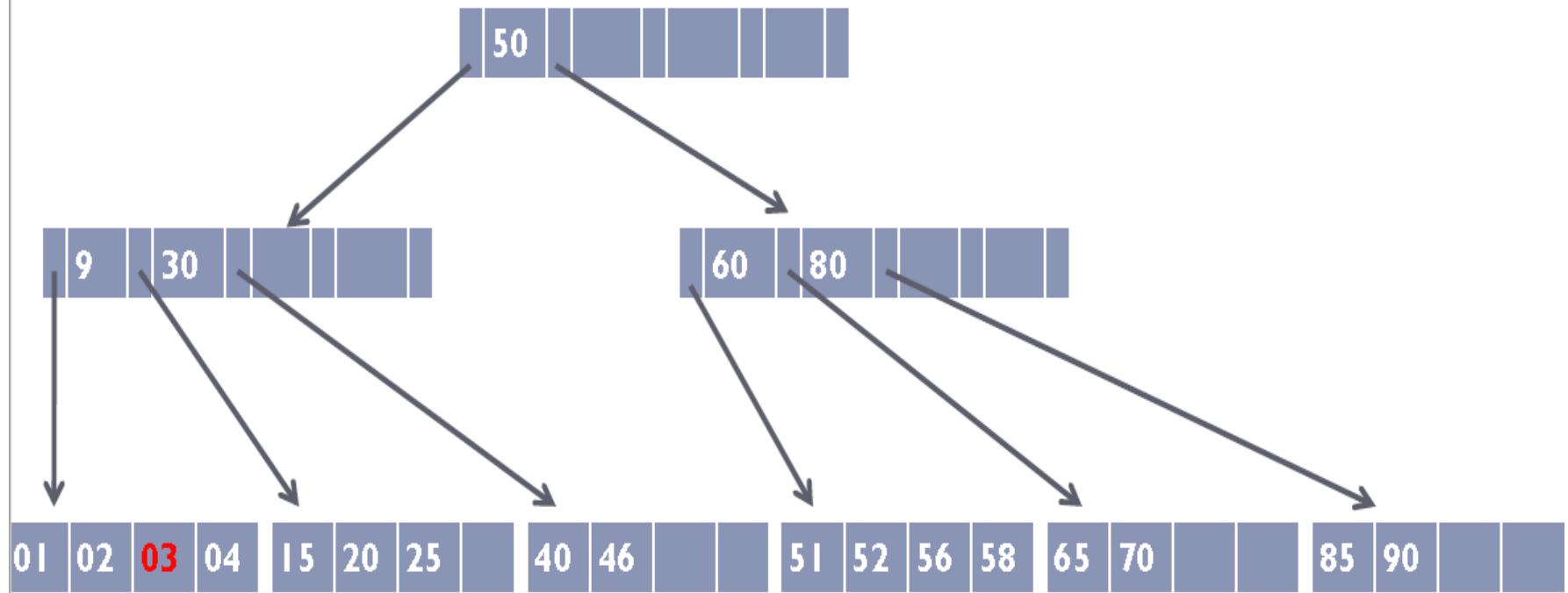
Exemplo:  
Exclusão da chave 03



# Árvores B

## Remoção

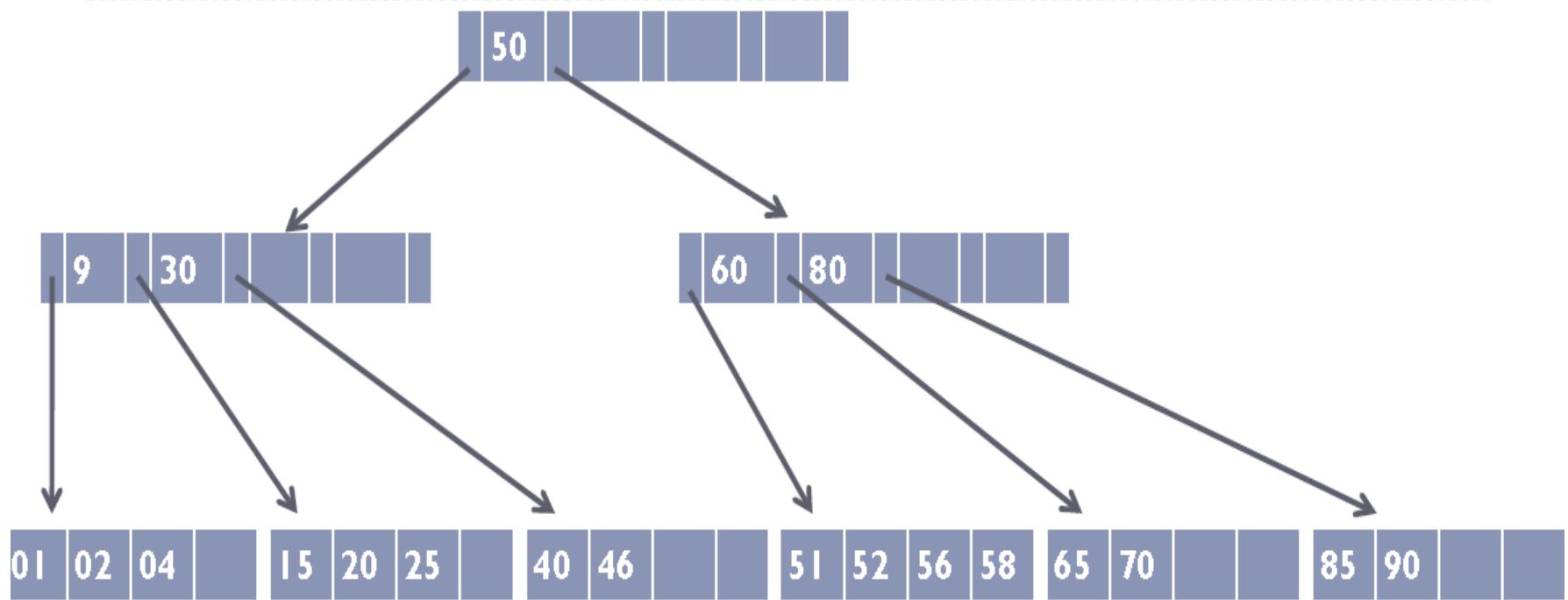
Exemplo:  
Exclusão da chave 03



# Árvores B

## Remoção

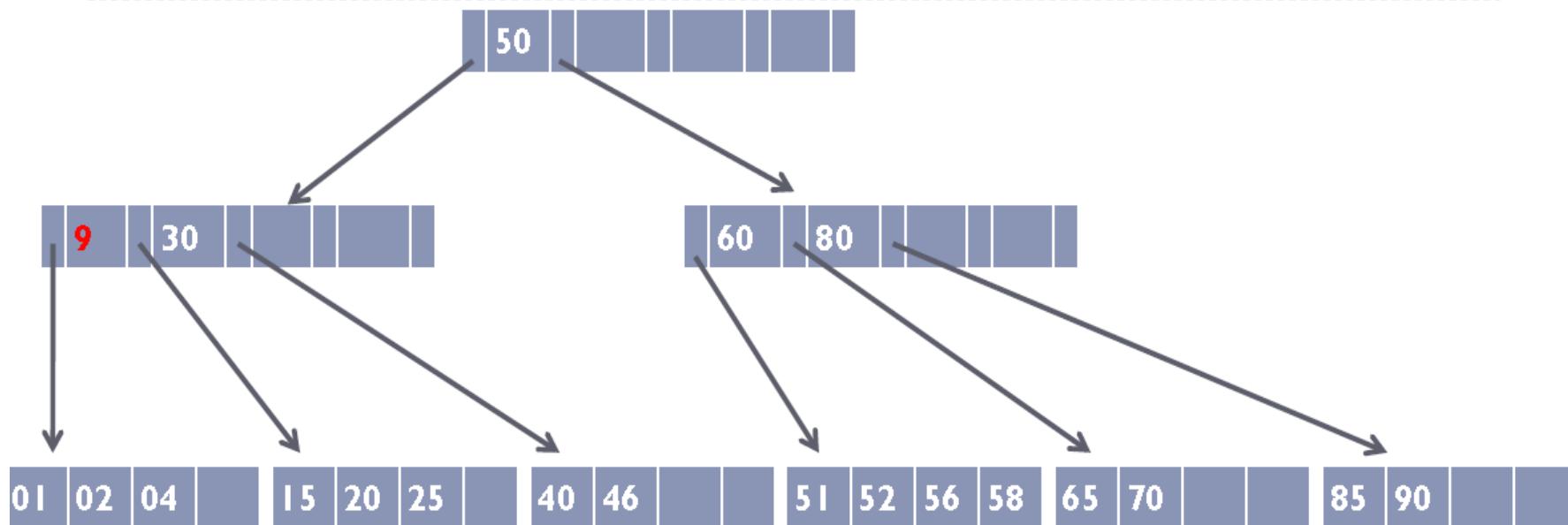
Exemplo:  
Exclusão da chave 03



# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 9

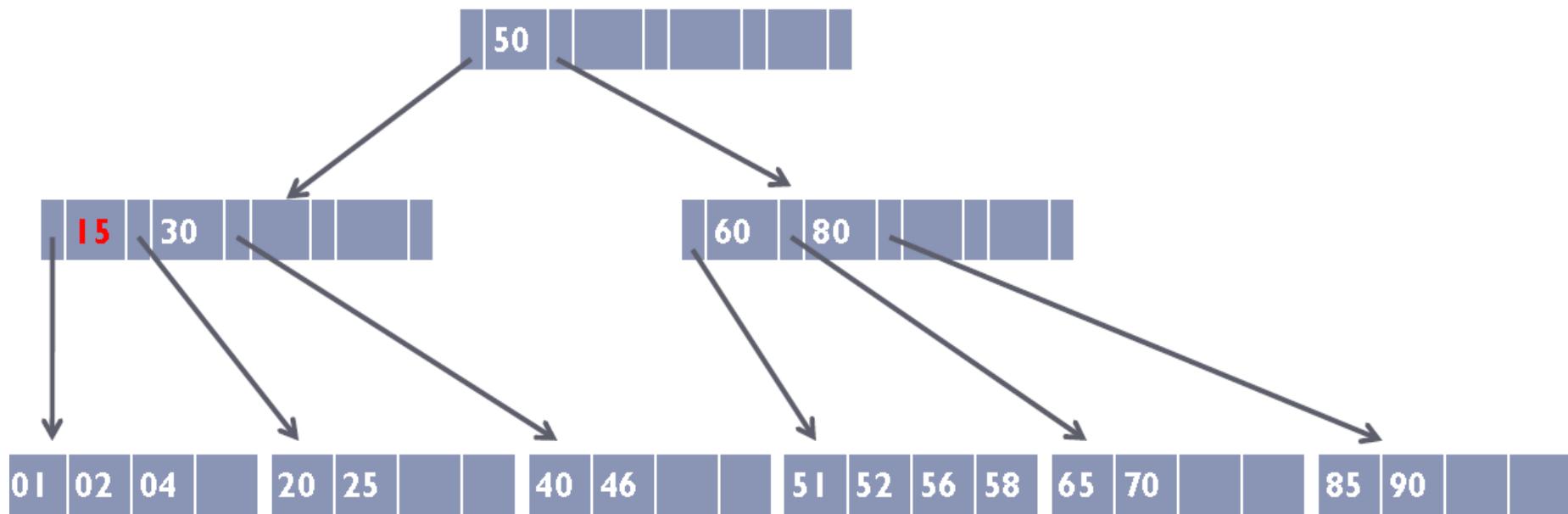


Substituir pela chave imediatamente maior

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 9



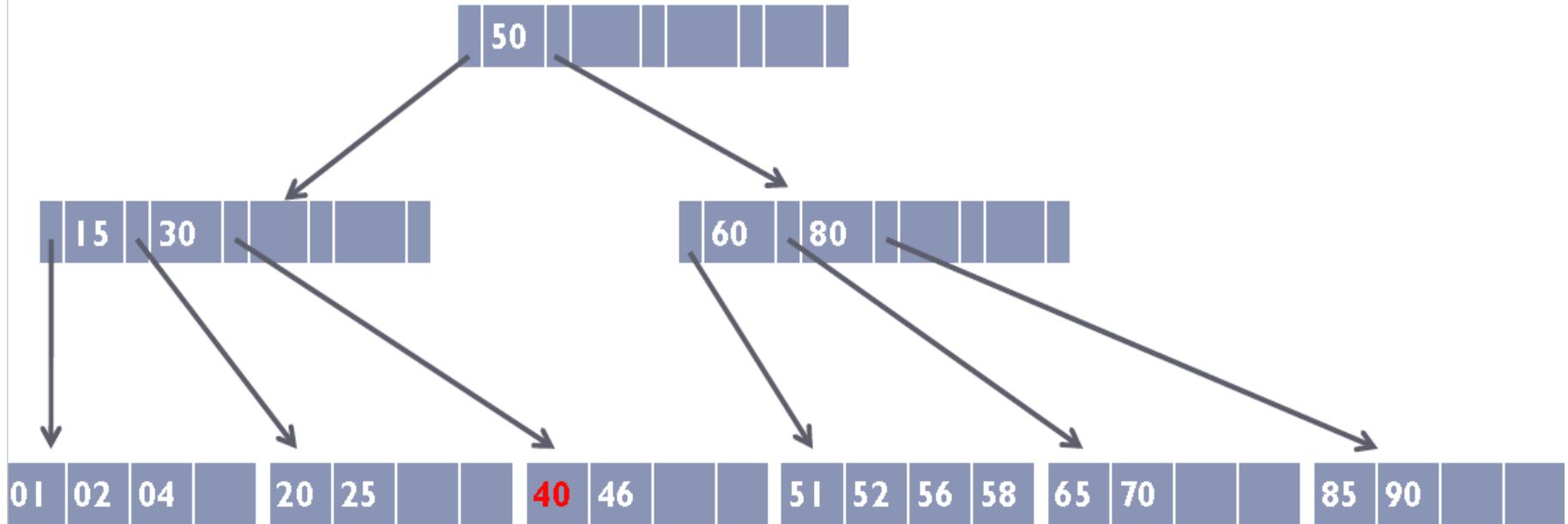
Substituir pela chave imediatamente maior

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40

$d = 2$



**Problema:** o nó ficaria com menos de  $d$  chaves, o que não é permitido

# Árvores B

## Remoção

- Solução:
  - Concatenação ou
  - Redistribuição

# Árvores B

## Remoção

- **Concatenação:**

- Duas páginas **P** e **Q** são **irmãs adjacentes** se têm o mesmo pai **W** e são apontadas por dois ponteiros adjacentes em **W**.
- **P** e **Q** podem ser concatenadas se:
  - são **irmãs adjacentes**; e
  - juntas possuem menos de **2d** chaves.

# Árvores B

## Remoção

- **Operação de concatenação de P e Q:**

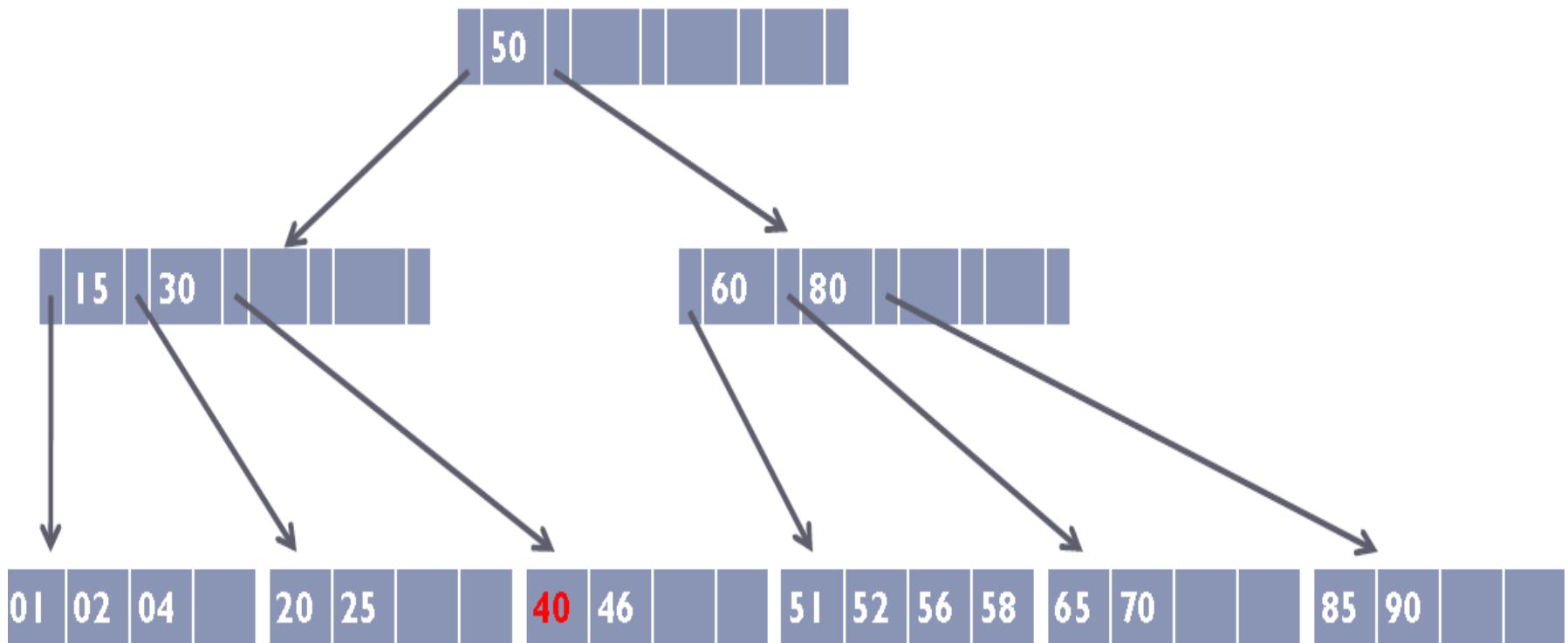
- Agrupar as entradas de Q em P.
- Em W, pegar a chave que está entre os ponteiros que apontam para P e Q, e transferi-la para P.
- Em W, eliminar o ponteiro que ficava junto à chave que foi transferida.

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40

$d = 2$

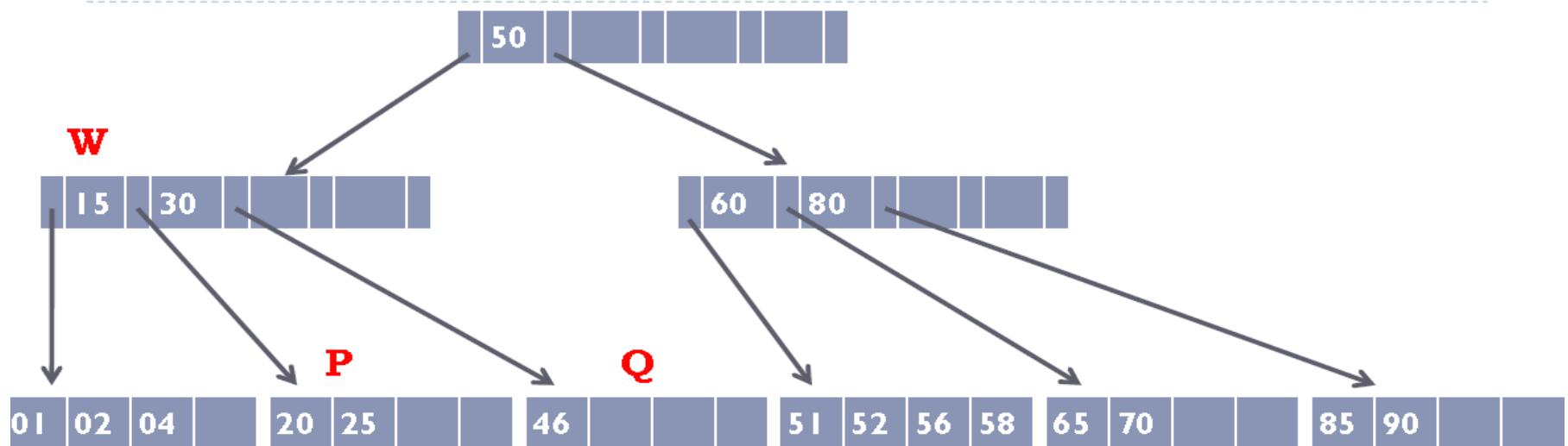


# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40

$d = 2$



Página Q ficou com menos de  $d$  chaves

Página P e Q são irmãs adjacentes

Soma de chaves de P e Q  $< 2d$

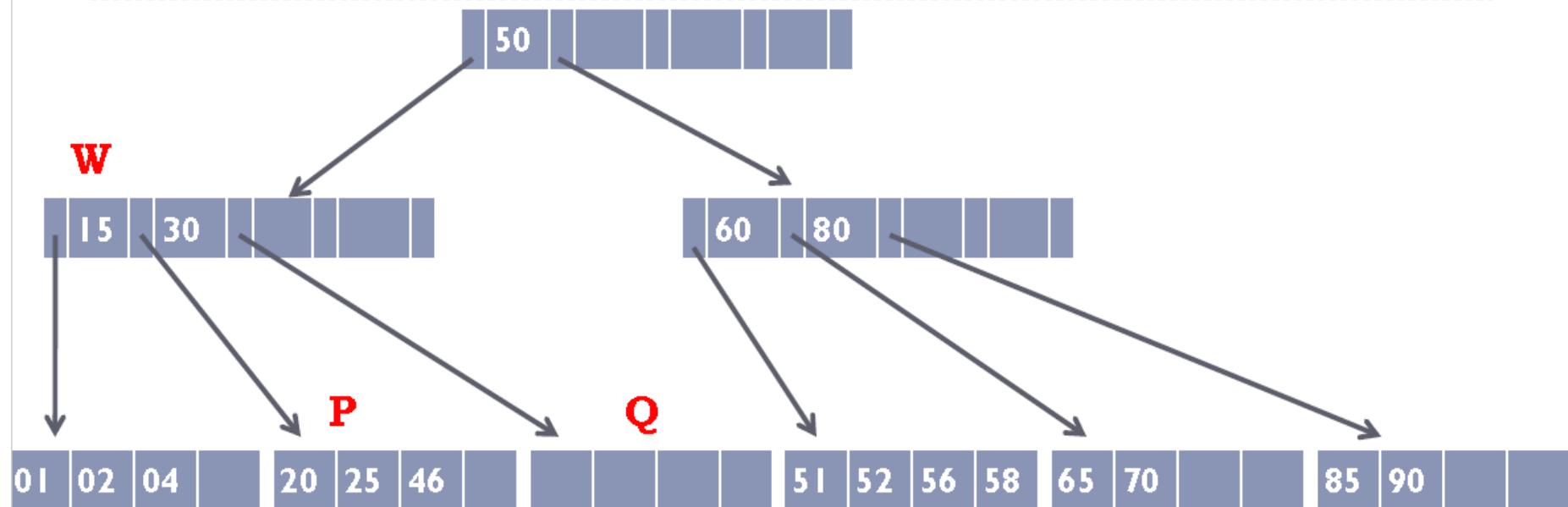
CONCATENAR P e Q

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40

$d = 2$



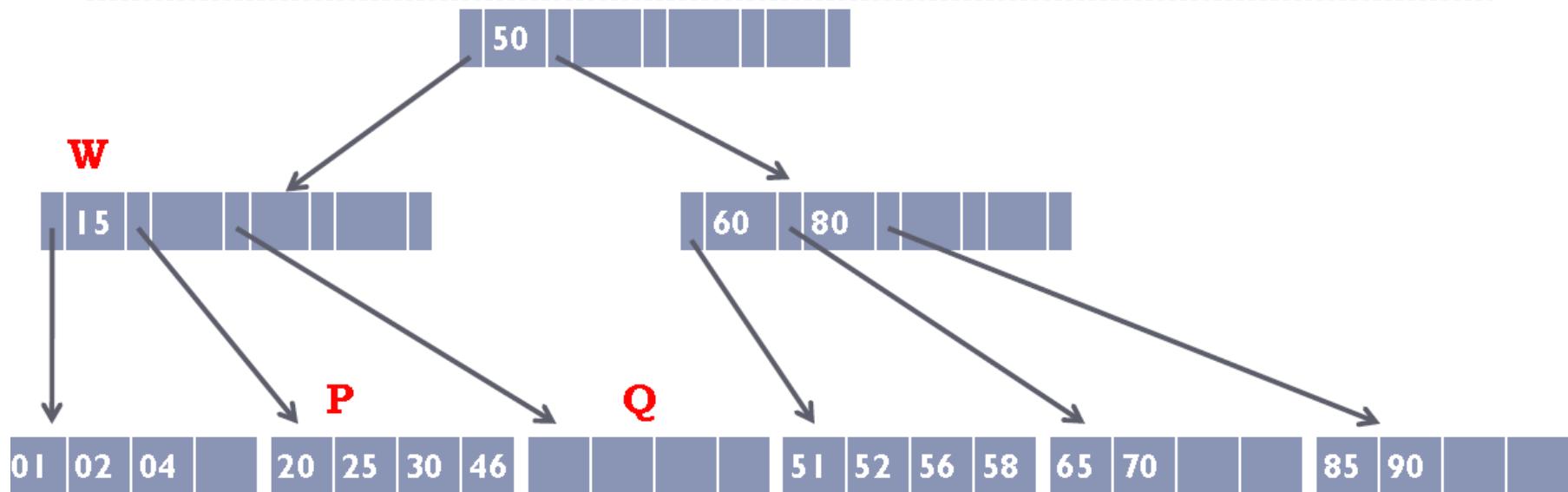
Transferir dados de Q para P

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40

$d = 2$

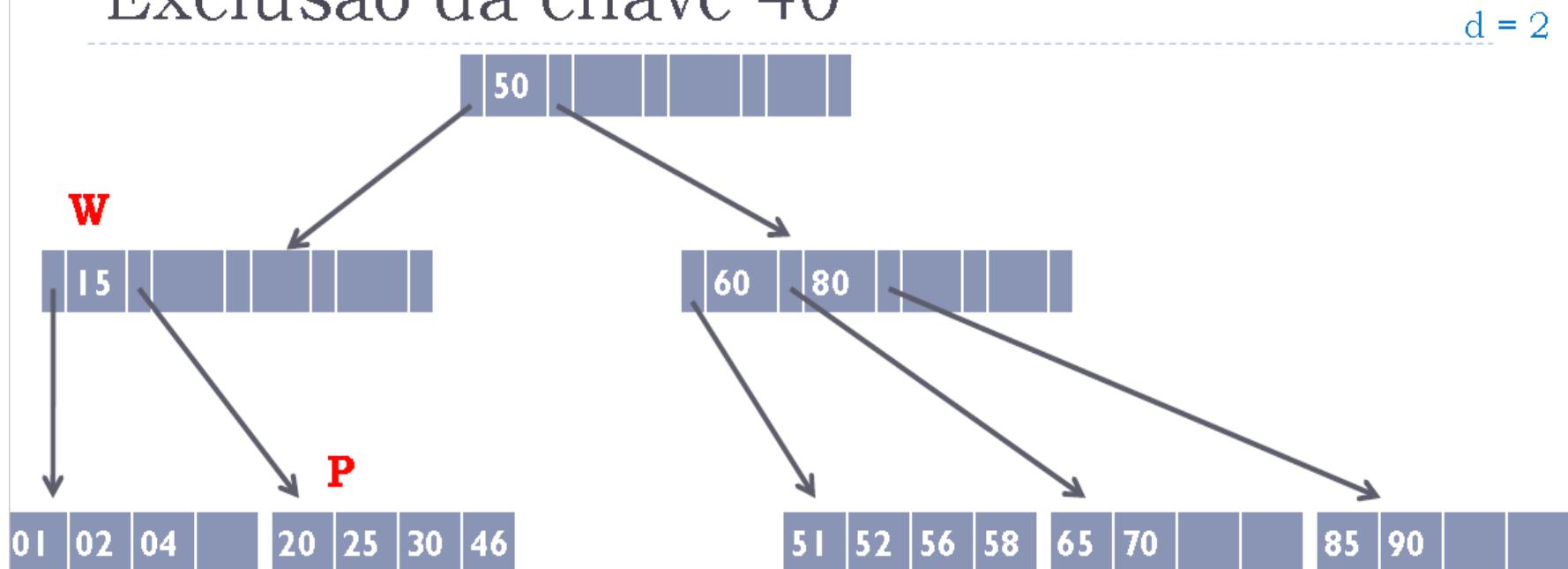


Transferir chave que separa os ponteiros de P e Q  
em W para P

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40



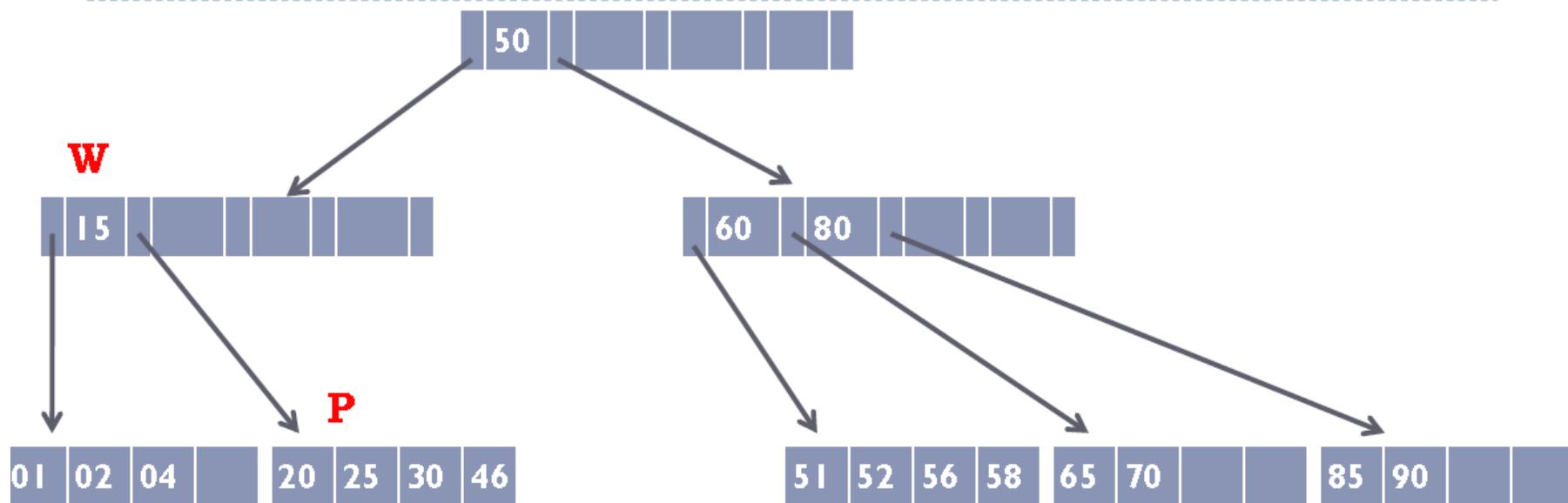
Eliminar página Q e ponteiro

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40

$d = 2$

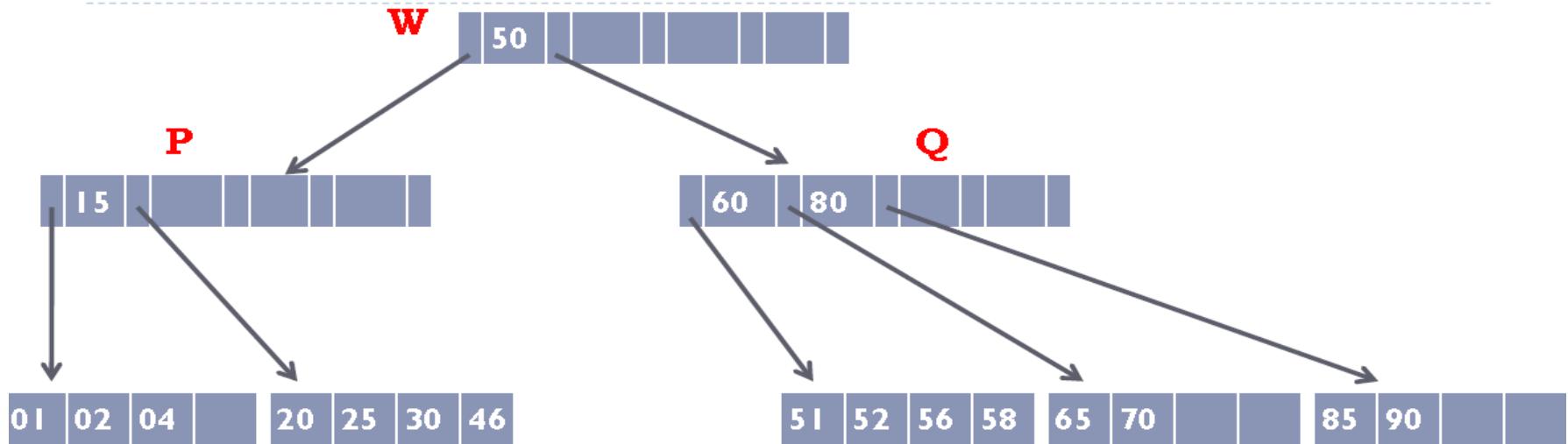


Página W ficou com menos de  $d$  chaves  
necessário propagar operação

# Indexação Árvores B - Exclusão

Exemplo:  
Exclusão da chave 40

$d = 2$

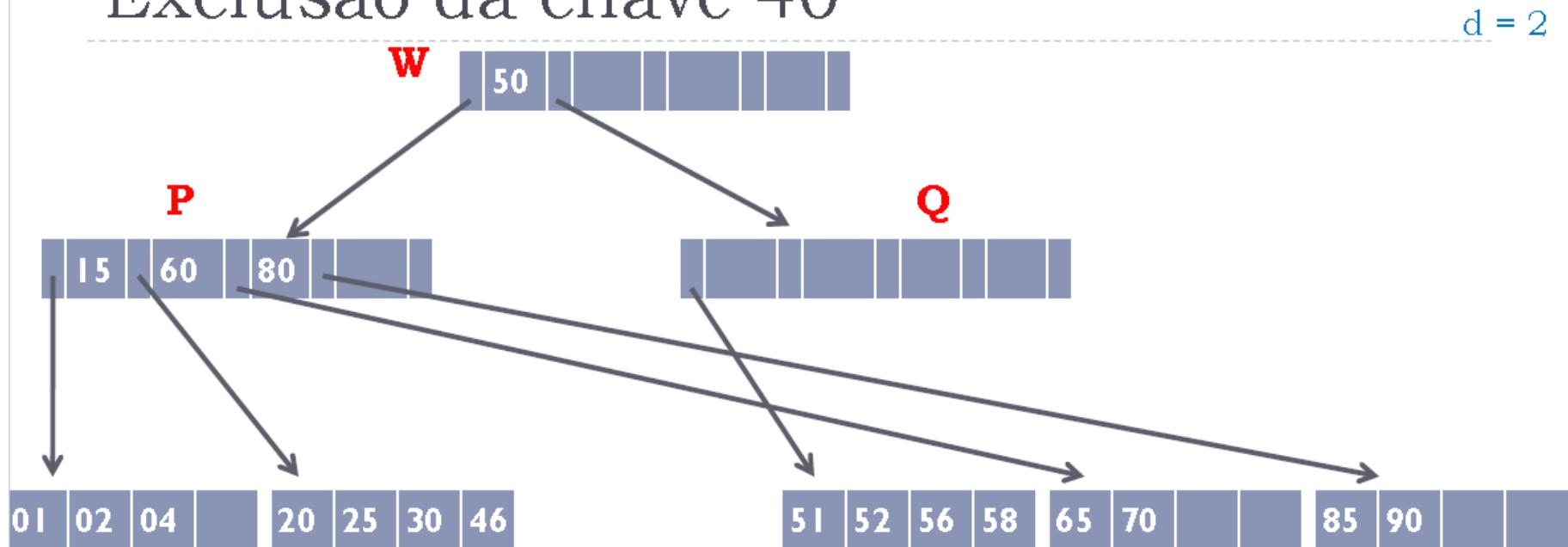


Página P e Q são irmãs adjacentes  
Soma de chaves de P e Q  $< 2d$   
CONCATENAR P e Q

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40

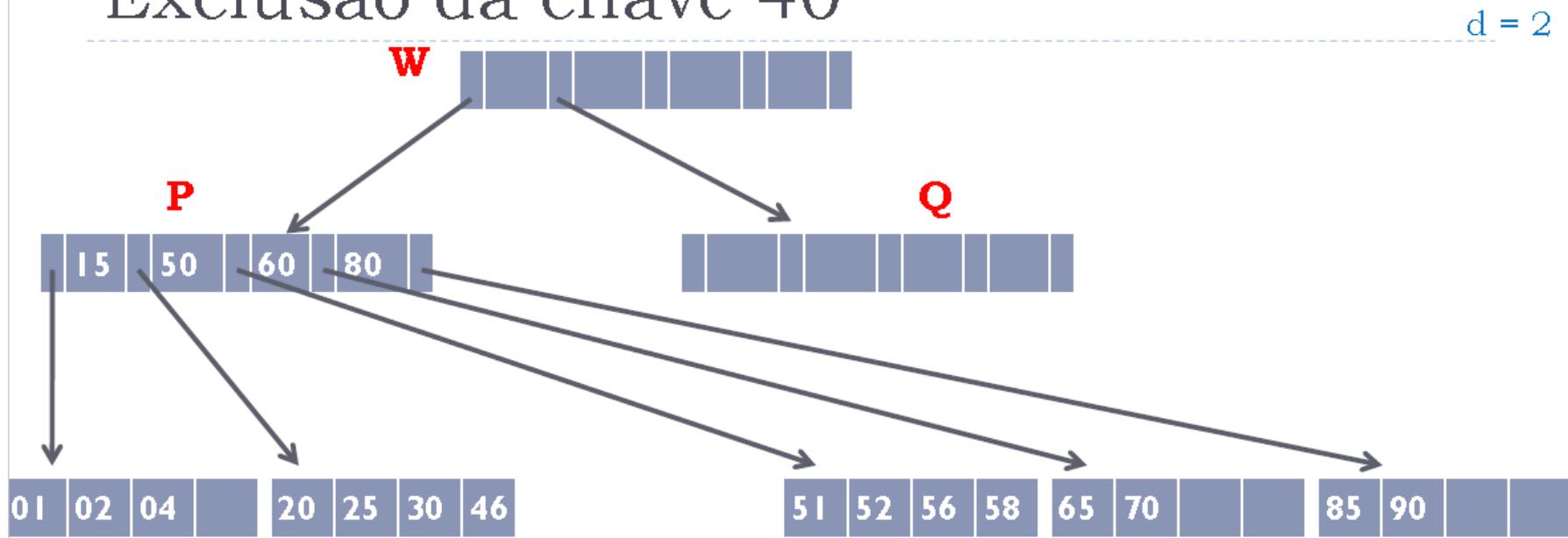


Transferir dados de Q para P

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40

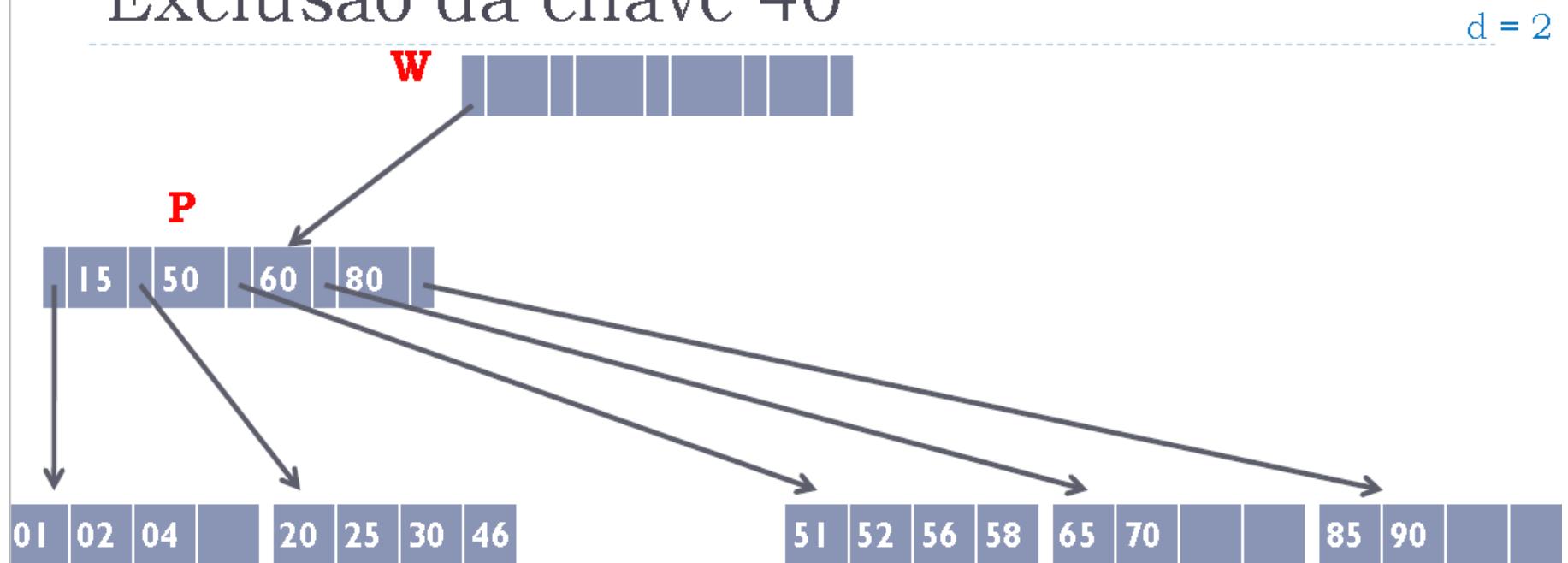


Transferir chave que separa os ponteiros de P e Q  
em W para P

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40



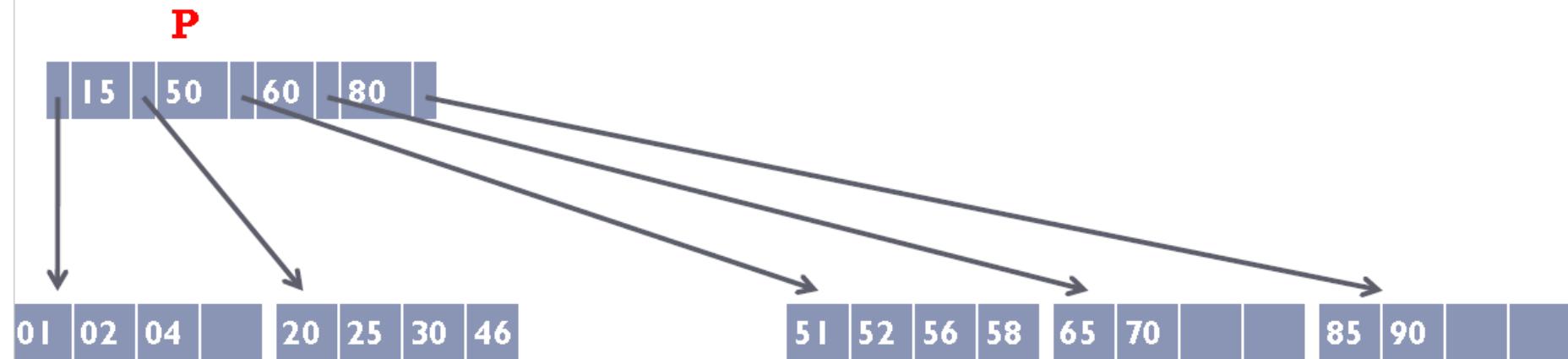
Eliminar página Q e ponteiro

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 40

$d = 2$



W ficou vazia e era a raiz: eliminá-la  
P passa a ser a nova raiz

# Árvores B

## Remoção

- **Redistribuição:**

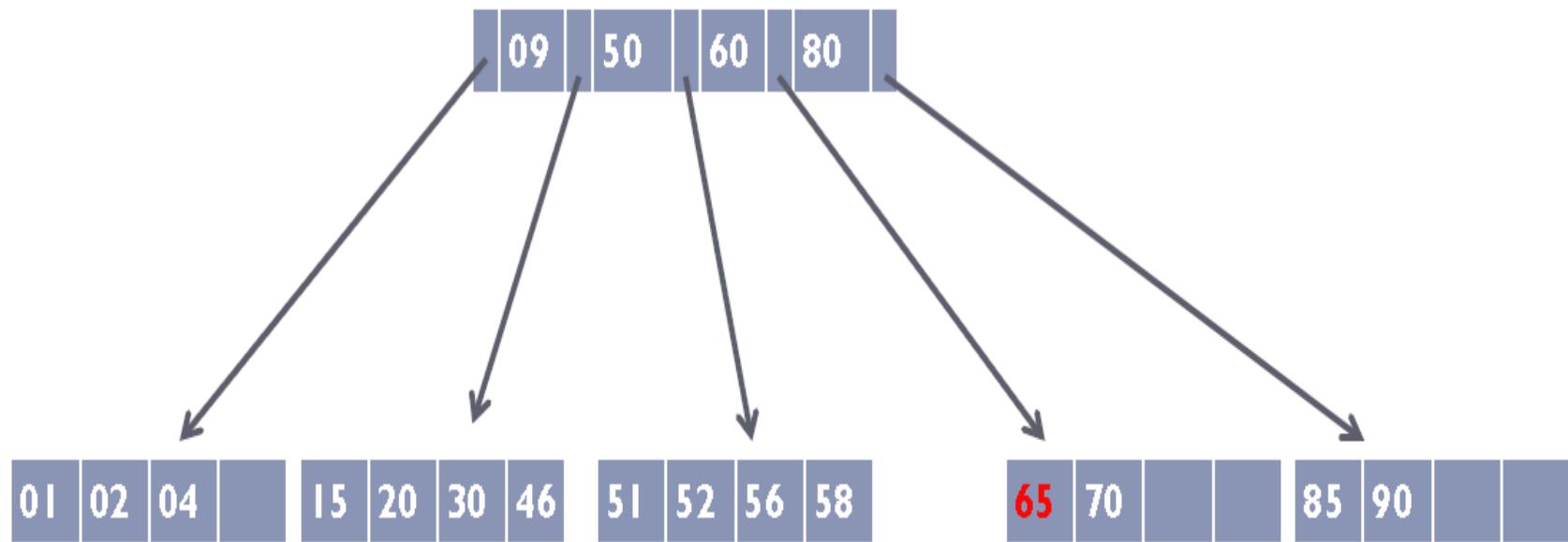
- Ocorre quando a soma das entradas de **P** e de seu irmão adjacente **Q** é maior ou igual a  $2d$ .
- **Concatenar P e Q :** Isso resulta em um nó **P** com mais de  $2d$  chaves, o que não é permitido.
- **Particionar** o nó concatenado, usando **Q** como novo nó .
- Essa operação não é propagável.
- O nó **W**, pai de **P** e **Q**, é alterado, mas seu número de chaves não é modificado.

# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 65

$d = 2$

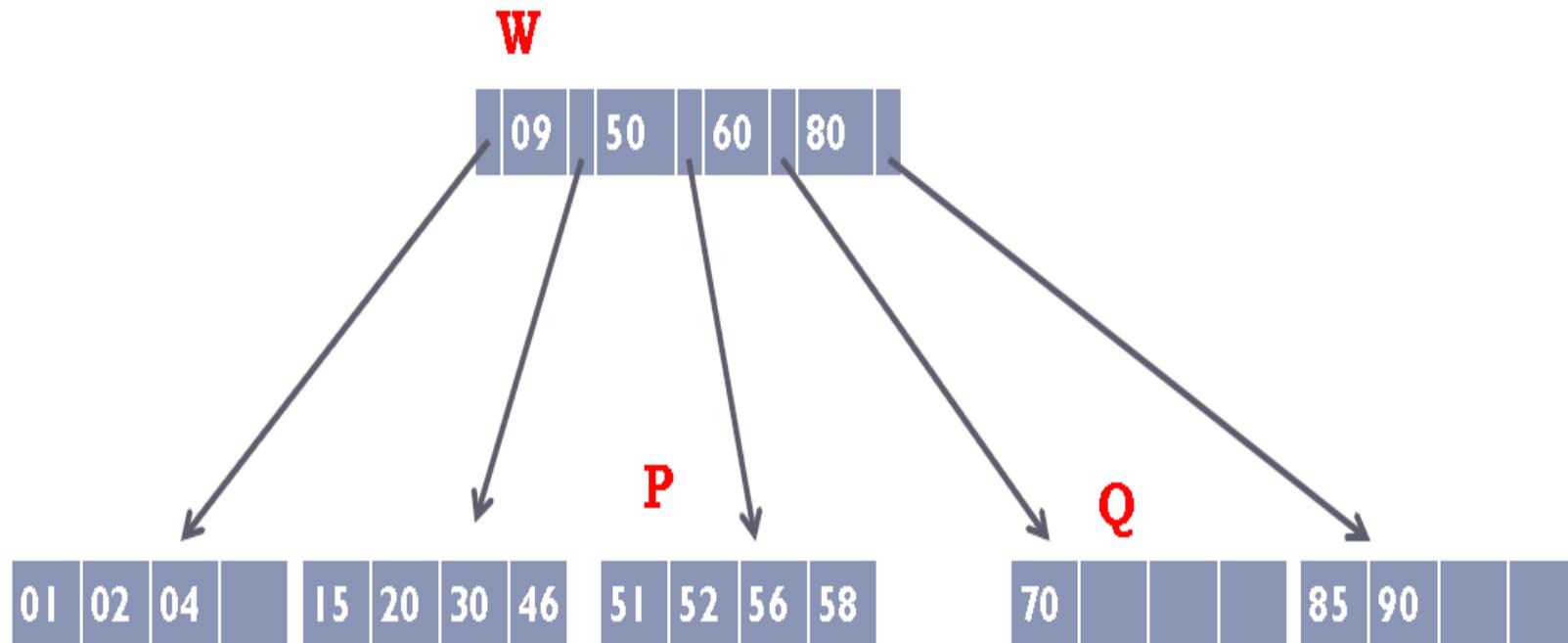


# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 65

$d = 2$

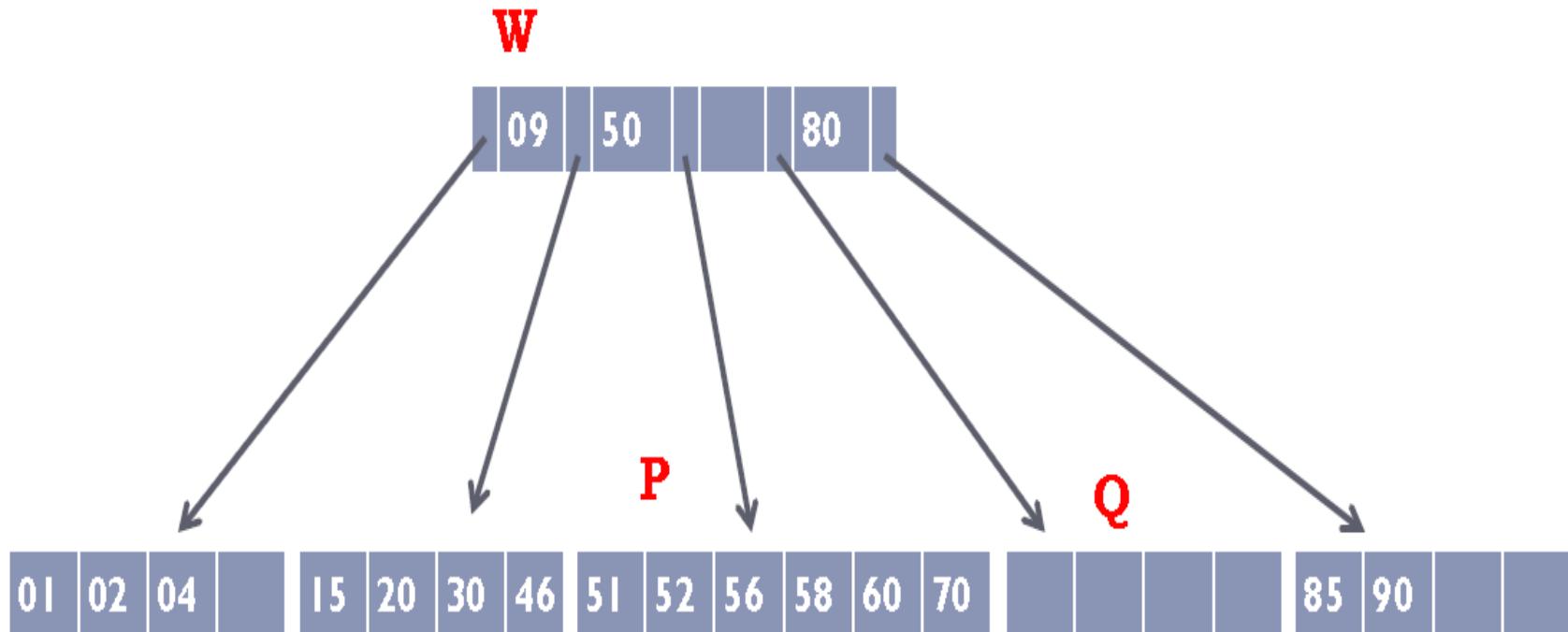


# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 65

$d = 2$

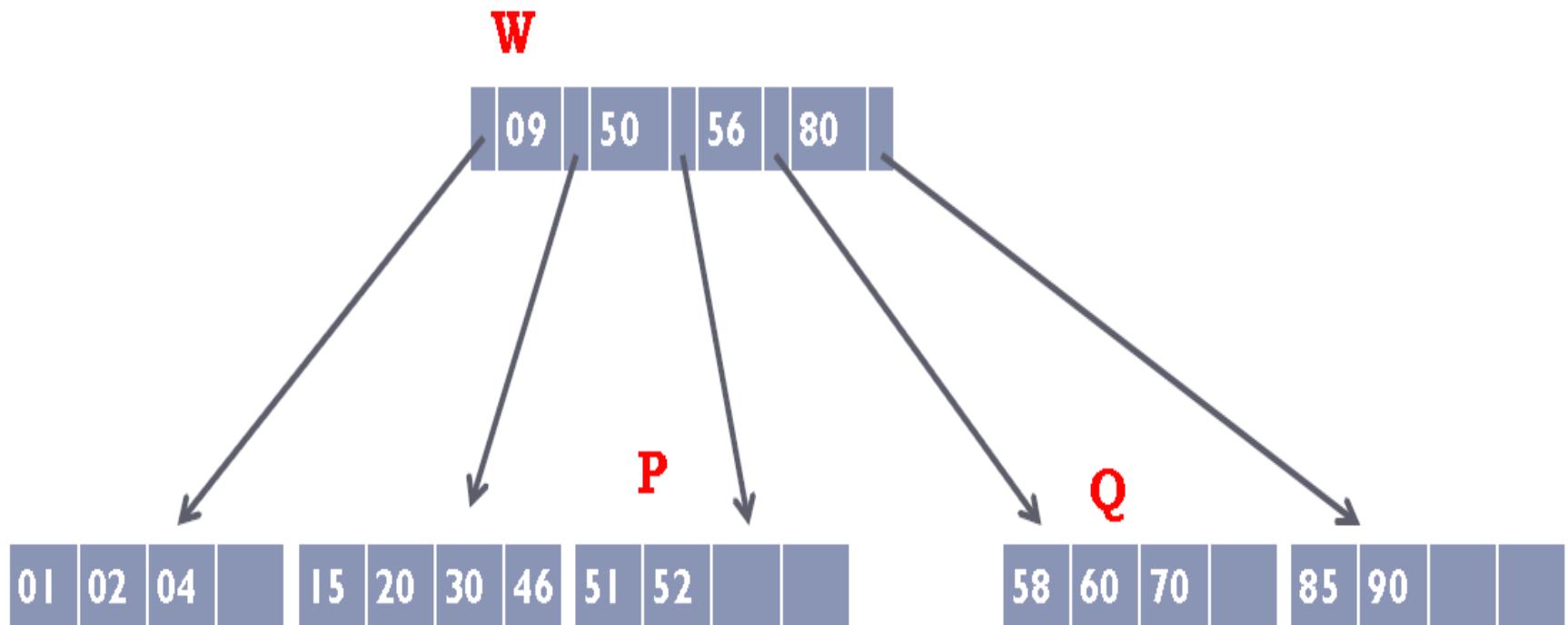


# Árvores B

## Remoção

Exemplo:  
Exclusão da chave 65

$d = 2$



# Árvores B

## Remoção

- E quando as duas alternativas são possíveis?
  - Quando for possível usar **concatenação** ou **redistribuição** (porque o nó possui 2 nós adjacentes, cada um levando a uma solução diferente), optar pela redistribuição.
  - Ela é menos custosa, pois não se propaga.
  - Ela evita que o nó fique cheio, deixando espaço para futuras inserções.

# Árvore B

```
305
306 CREATE TABLE cliente (
307     id_cliente SERIAL PRIMARY KEY,
308     nome VARCHAR(100) NOT NULL,
309     email VARCHAR(150) UNIQUE NOT NULL,
310     idade INT NOT NULL,
311     cidade VARCHAR(100)
312 );
313
```

Data output    Messages

CREATE TABLE

Query returned successfully in 129 msec.

# Árvore B

```
315 -- Por padrão, o PostgreSQL cria índice B-Tree em chaves primárias e colunas UNIQUE,  
316 -- mas podemos criar índices adicionais para acelerar buscas.  
317  
318 -- Índice B-Tree na coluna idade  
319 CREATE INDEX idx_cliente_idade  
320     ON cliente USING btree (idade);  
321  
322 -- Índice B-Tree na coluna cidade  
323 CREATE INDEX idx_cliente_cidade  
324     ON cliente USING btree (cidade);  
325  
326
```

Data output Messages

CREATE INDEX

Query returned successfully in 127 msec.

# Árvore B

```
319 CREATE INDEX idx_cliente_idade
320     ON cliente USING btree (idade);
321
322 -- Índice B-Tree na coluna cidade
323 CREATE INDEX idx_cliente_cidade
324     ON cliente USING btree (cidade);
325
326 INSERT INTO CLIENTE
327 VALUES (35,'ASDRUBAL SOARES NETO','asr@gmail.com',27,'COLATINA'),
328         (41,'JAMBIRA TIMBIRAS','jtimb@gmail.com',23,'LINHARES'),
329         (50,'ANACLETO SUETONIO ANDRADA','anac@yahoo.com.br',37,'VITORIA'),
330         (51,'YARAMAS SOUZA','yara@yahoo.com.br',49,'VILA VELHA'),
331         (57,'DESIDERIO PIRES JR','desid@hotmail.com', 67,'COLATINA');
```

# Árvore B

```
325
326 INSERT INTO CLIENTE
327 VALUES (35,'ASDRUBAL SOARES NETO','asr@gmail.com',27,'COLATINA'),
328      (41,'JAMBIRA TIMBIRAS','jtimb@gmail.com',23,'LINHARES'),
329      (50,'ANACLETO SUETONIO ANDRADA','anac@yahoo.com.br',37,'VITORIA'),
330      (51,'YARAMAS SOUZA','yara@yahoo.com.br',49,'VILA VELHA'),
331      (57,'DESIDERIO PIRES JR','desid@hotmail.com', 67,'COLATINA');
332
333 -- Busca por igualdade (usa índice em idade)
334 SELECT * FROM cliente WHERE idade = 37;
335
```

Data output Messages

|   | id_cliente<br>[PK] integer | nome<br>character varying (100) | email<br>character varying (150) | idade<br>integer | cidade<br>character varying (100) |
|---|----------------------------|---------------------------------|----------------------------------|------------------|-----------------------------------|
| 1 | 50                         | ANACLETO SUETONIO ...           | anac@yahoo.com.br                | 37               | VITORIA                           |

# Árvore B

```
332  
333 -- Busca por igualdade (usa índice em idade)  
334 SELECT * FROM cliente WHERE idade = 37;  
335  
336 -- Busca por intervalo (usa índice em idade)  
337 SELECT * FROM cliente WHERE idade BETWEEN 25 AND 40;  
338
```

Data output Messages



|   | id_cliente<br>[PK] integer | nome<br>character varying (100) | email<br>character varying (150) | idade<br>integer | cidade<br>character varying (100) |
|---|----------------------------|---------------------------------|----------------------------------|------------------|-----------------------------------|
| 1 | 35                         | ASDRUBAL SOARES NETO            | asr@gmail.com                    | 27               | COLATINA                          |
| 2 | 50                         | ANACLETO SUETONIO ANDRA...      | anac@yahoo.com.br                | 37               | VITORIA                           |

# Árvore B

```
335  
336 -- Busca por intervalo (usa índice em idade)  
337 SELECT * FROM cliente WHERE idade BETWEEN 25 AND 40;  
338  
339 -- Busca por igualdade em texto (usa índice em cidade)  
340 SELECT * FROM cliente WHERE cidade = 'COLATINA';  
341
```

Data output Messages



|   | id_cliente<br>[PK] integer | nome<br>character varying (100) | email<br>character varying (150) | idade<br>integer | cidade<br>character varying (100) |
|---|----------------------------|---------------------------------|----------------------------------|------------------|-----------------------------------|
| 1 | 35                         | ASDRUBAL SOARES NETO            | asr@gmail.com                    | 27               | COLATINA                          |
| 2 | 57                         | DESIDERIO PIRES JR              | desid@hotmail.com                | 67               | COLATINA                          |

# Árvore B

```
334 SELECT * FROM cliente WHERE idade = 37;
335
336 -- Busca por intervalo (usa índice em idade)
337 SELECT * FROM cliente WHERE idade BETWEEN 25 AND 40;
338
339 -- Busca por igualdade em texto (usa índice em cidade)
340 SELECT * FROM cliente WHERE cidade = 'COLATINA';
341
342 -- Ordenação pode usar índice também
343 SELECT * FROM cliente ORDER BY idade;
344
```

Data output Messages

|   | id_cliente<br>[PK] integer | nome<br>character varying (100) | email<br>character varying (150) | idade<br>integer | cidade<br>character varying (100) |
|---|----------------------------|---------------------------------|----------------------------------|------------------|-----------------------------------|
| 1 | 41                         | JAMBIRA TIMBIRAS                | jtimb@gmail.com                  | 23               | LINHARES                          |
| 2 | 35                         | ASDRUBAL SOARES NE...           | asr@gmail.com                    | 27               | COLATINA                          |
| 3 | 50                         | ANACLETO SUETONIO ...           | anac@yahoo.com.br                | 37               | VITORIA                           |
| 4 | 51                         | YARAMAS SOUZA                   | yara@yahoo.com.br                | 49               | VILA VELHA                        |
| 5 | 57                         | DESIDERIO PIRES JR              | desid@hotmail.com                | 67               | COLATINA                          |

# Árvore B

```
341  
342 -- Ordenação pode usar índice também  
343 SELECT * FROM cliente ORDER BY idade;  
344  
345 EXPLAIN ANALYZE  
346 SELECT * FROM cliente WHERE idade BETWEEN 25 AND 40;  
347
```

Data output Messages



## QUERY PLAN

text



|   |  |
|---|--|
| 1 | Index Scan using idx_cliente_idade on cliente (cost=0.14..8.16 rows=1 width=762) (actual time=0.021..0.023 rows=2 loops=1) |
| 2 | Index Cond: ((idade >= 25) AND (idade <= 40))  |
| 3 | Planning Time: 0.140 ms  |
| 4 | Execution Time: 0.046 ms   |

# Árvores de Múltiplos Caminhos

## Árvores B\*

- Árvores B\*

- É uma variação da árvore B.
- Todos os nós, exceto a raiz, precisam estar  $\frac{2}{3}$  cheios (em contraste com  $\frac{1}{2}$  exigido pela árvore B).
- Para manter esta propriedade, os nós não são particionados logo que ficam cheios. Ao invés disso, suas chaves são compartilhadas com o nó vizinho, até que ambos fiquem cheios. Neste ponto, os dois nós são divididos em 3 nós.
- Na prática, não é muito utilizada.

# Árvores de Múltiplos Caminhos

## Árvores B+

- **Árvores B+**

– É semelhante à árvore B, exceto por duas características muito importantes:

- 1 - **Armazena dados somente nas folhas** – os nós internos servem apenas de ponteiros.
- 2 - **As folhas são encadeadas:** Isso permite o armazenamento dos dados em um arquivo, e do índice em outro arquivo separado.

# Árvores de Múltiplos Caminhos

## Árvores B+

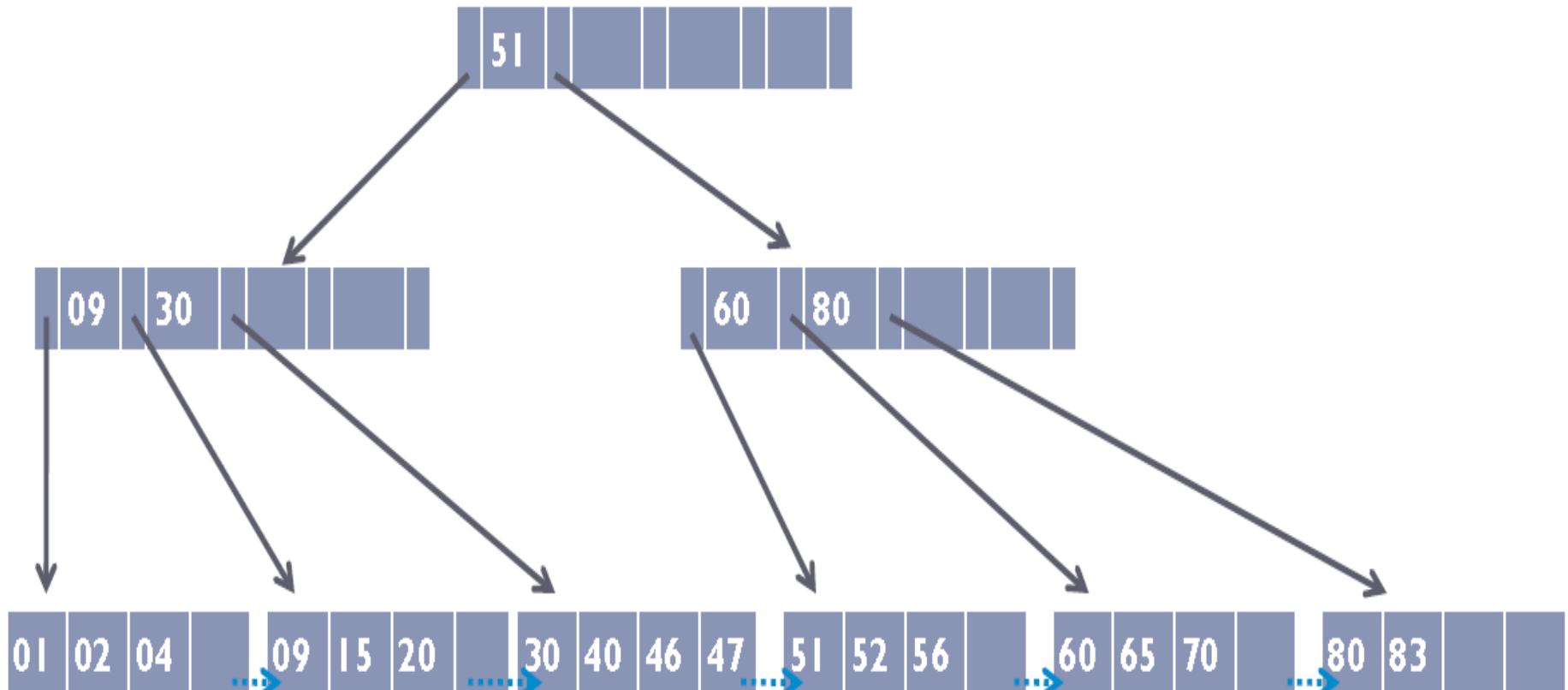
- Árvores B+ são muito importantes por sua eficiência, e muito utilizadas na prática:

- Os sistemas de arquivo **NTFS**, **ReiserFS**, **NSS**, **XFS**, e **JFS** utilizam este tipo de árvore para indexação.
- Sistemas de Gerência de Banco de Dados como **IBM DB2**, **Informix**, **Microsoft SQL Server**, **Oracle 8**, **Sybase ASE**, **PostgreSQL**, **Firebird** e **SQLite** suportam este tipo de árvore para indexar tabelas.
- O **PostgreSQL** não implementa diretamente a estrutura **B+-tree** clássica (como em livros de Estrutura de Dados), mas sim uma variação chamada **GIST** e principalmente a **árvore B (balanced tree)** nativa do sistema, que na prática é muito próxima a uma B+-tree.
- Outros sistemas de gerência de dados como o **CouchDB**, **Tokyo Cabinet** e **Tokyo Tyrant** suportam este tipo de árvore para acesso a dados.

# Árvores de Múltiplos Caminhos

## Árvores B+

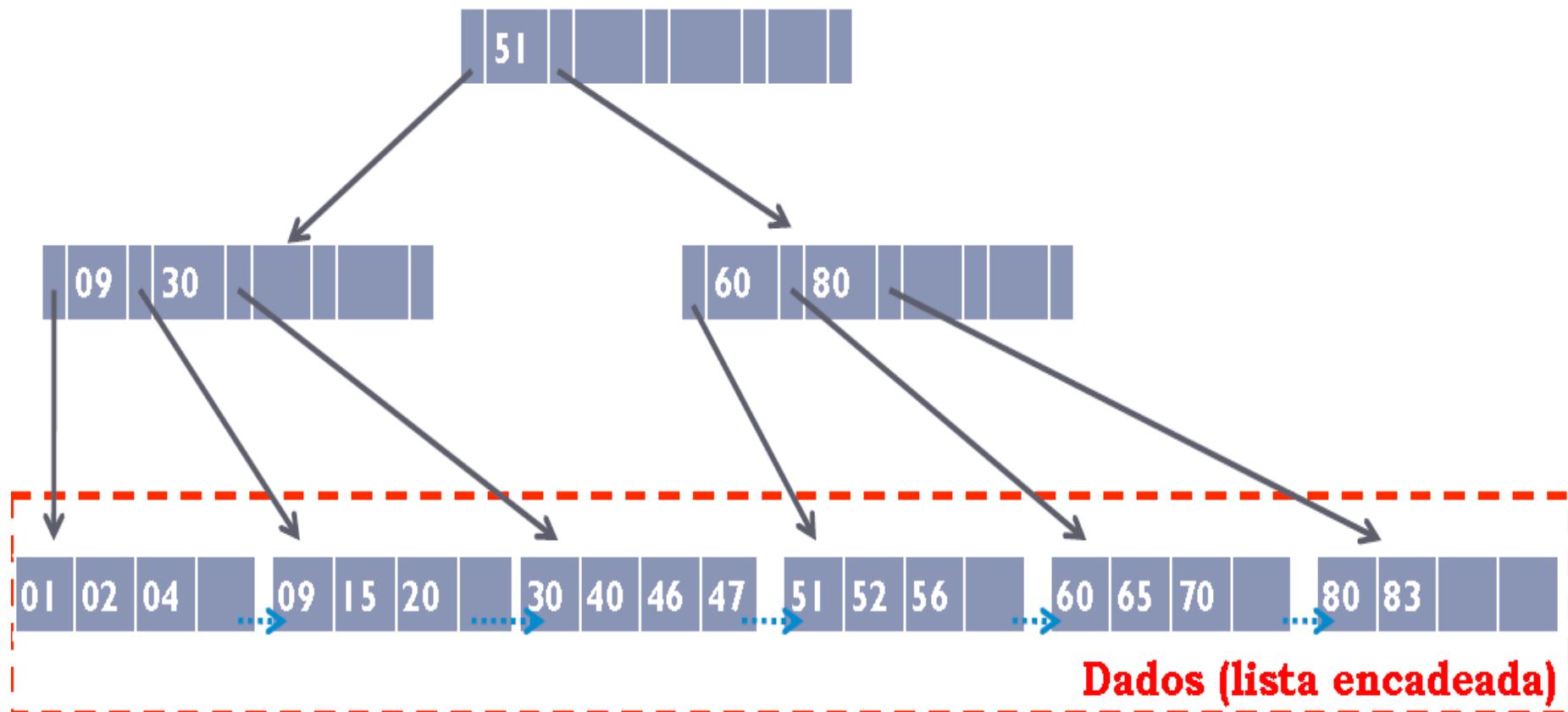
Exemplo de Árvore B+



# Árvores de Múltiplos Caminhos

## Árvores B+

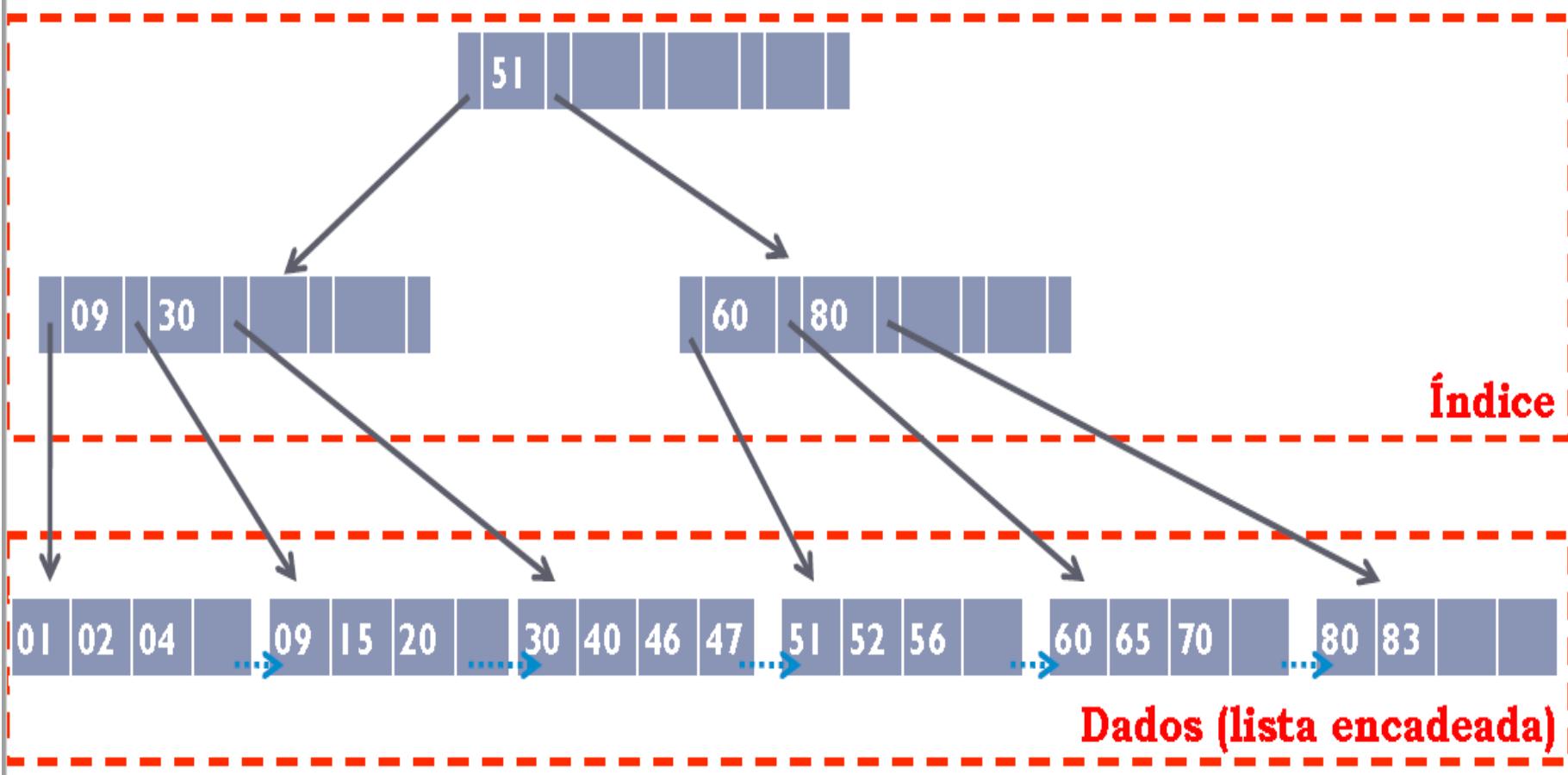
Exemplo de Árvore B+



# Árvores de Múltiplos Caminhos

## Árvores B+

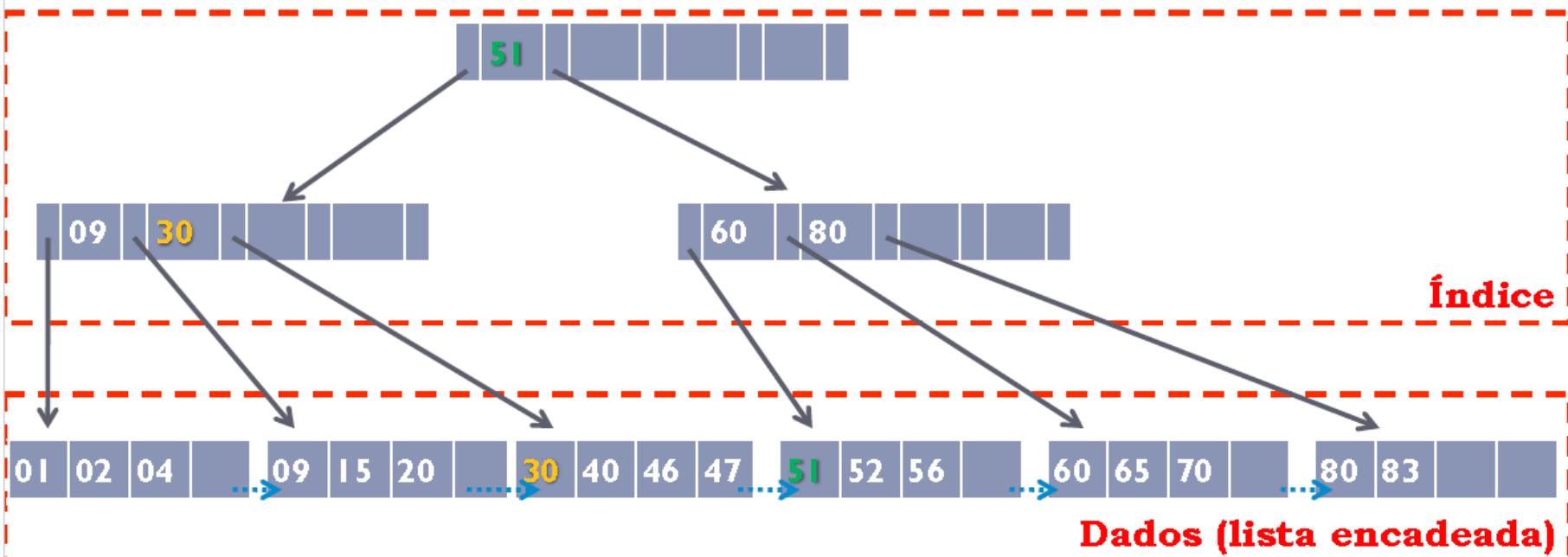
Exemplo de Árvore B+



# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Árvore B+



► **IMPORTANTE:**

- Os valores nos índices repetem valores de chave que aparecem nas folhas (diferente do que acontece nas árvores B)

# Árvores de Múltiplos Caminhos

## Árvores B+

- **Busca:** Só se pode ter certeza de que o registro foi encontrado quando se chega em uma folha.

# Árvores de Múltiplos Caminhos

## Árvores B+

- **Inserção:**

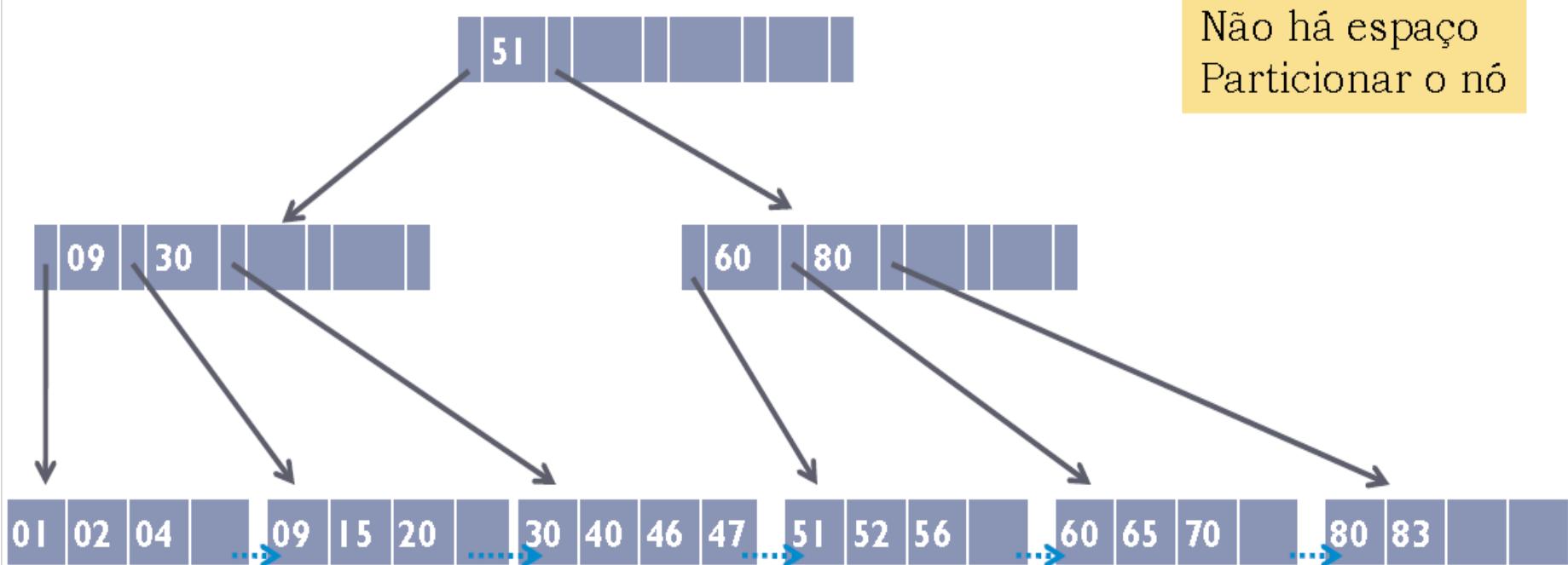
- Quando for necessário partitionar um nó durante uma inserção, o mesmo raciocínio é utilizado.
- A diferença é que para a página pai sobe somente a chave. O registro fica na folha, juntamente com a sua chave.
- ATENÇÃO: isso vale apenas se o nó que está sendo partitionado for uma folha. Se não for folha, o procedimento é o mesmo utilizado na árvore B.

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Inserção em Árvore B+  
Inserir chave 32

$d = 2$

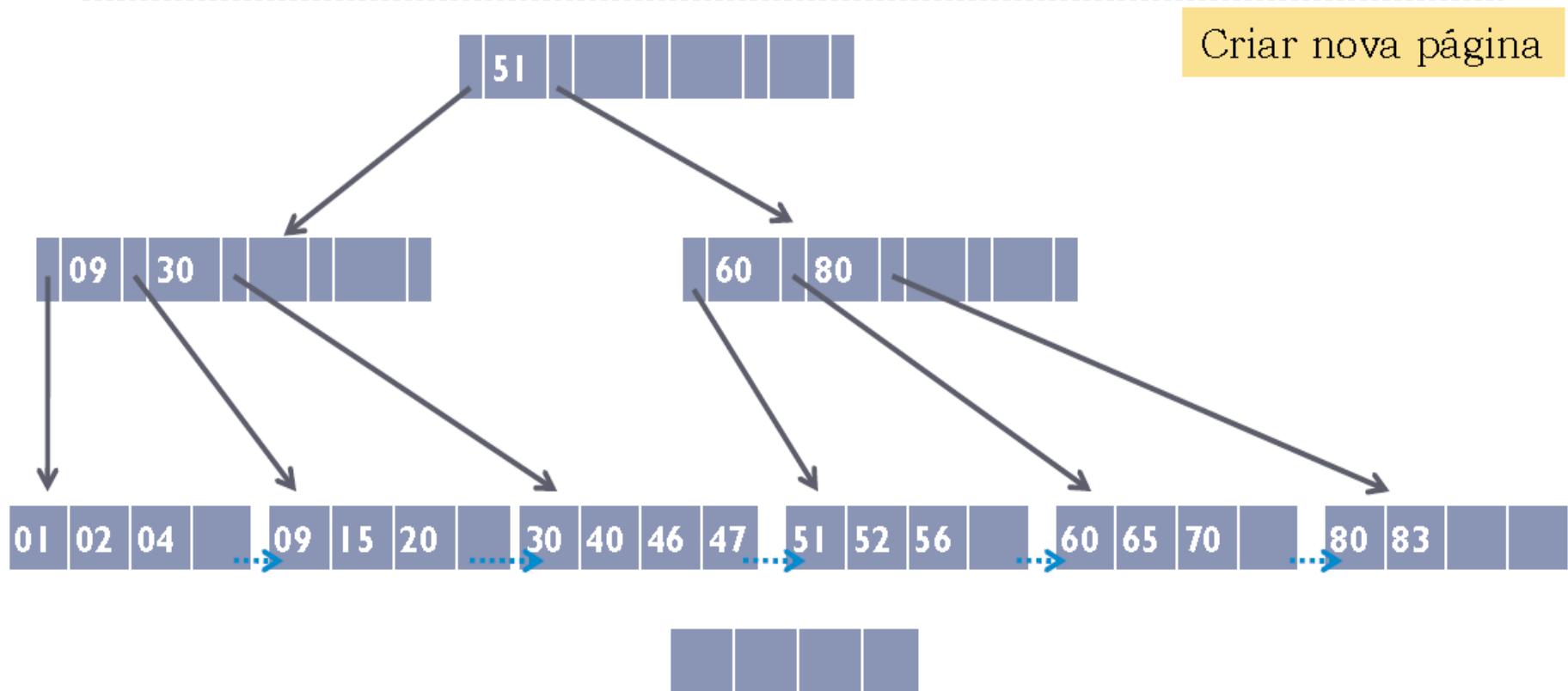


# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Inserção em Árvore B+  
Inserir chave 32

$d = 2$



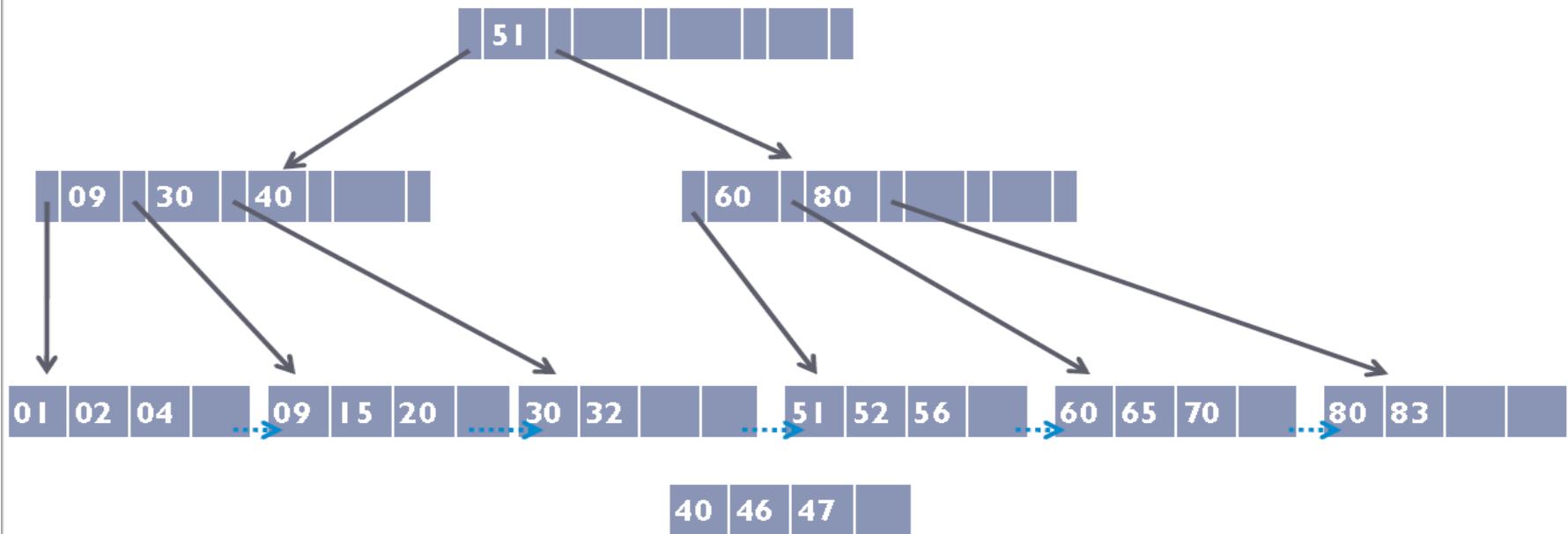
# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Inserção em Árvore B+

Inserir chave 32

$d = 2$



Dividir as chaves entre as duas páginas (30; 32; 40; 46; 47)  
 $d$  chaves na página original

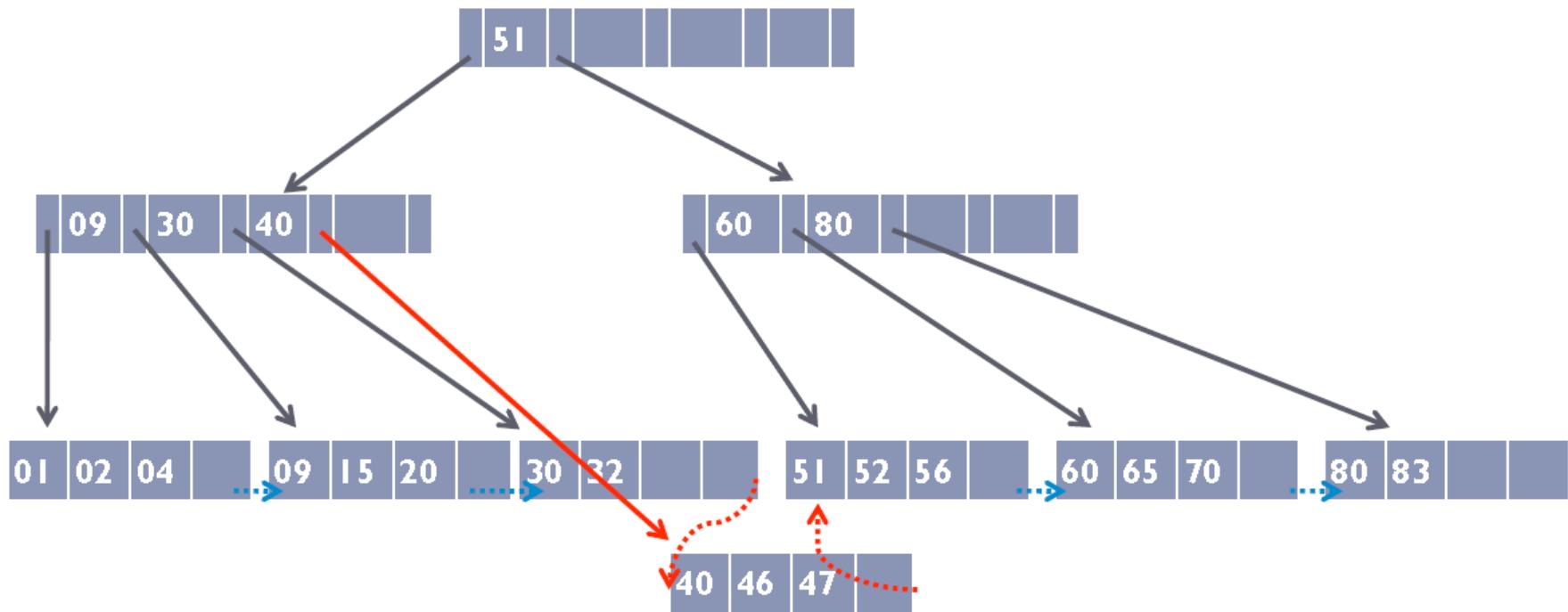
chave  $d+1$  sobe para nó pai (**mas registro é mantido na nova página**)  
 $d+1$  chaves restantes na nova página

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Inserção em Árvore B+  
Inserir chave 32

$d = 2$



Ajustar ponteiros

# Árvores de Múltiplos Caminhos

## Árvores B+

- **Exclusão:**

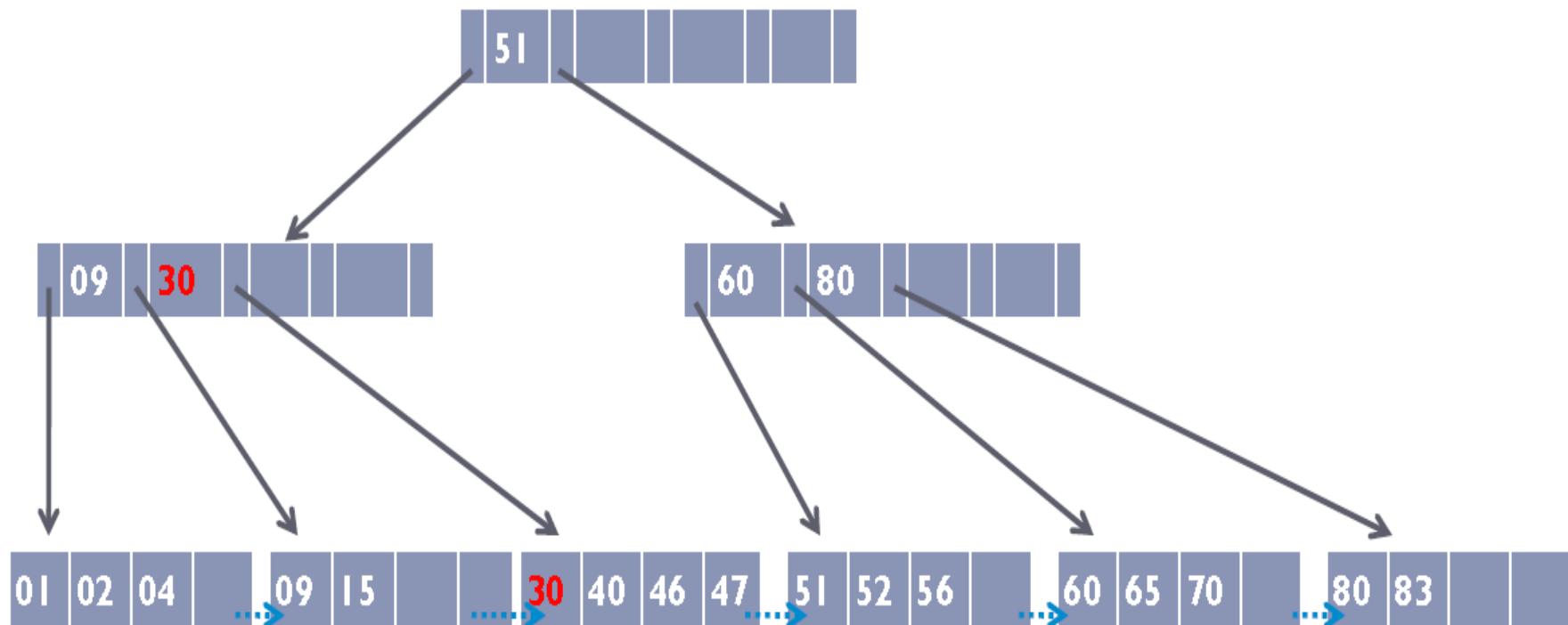
- Excluir apenas no nó folha.
- Chaves excluídas continuam nos nós intermediários.

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 30

$d = 2$

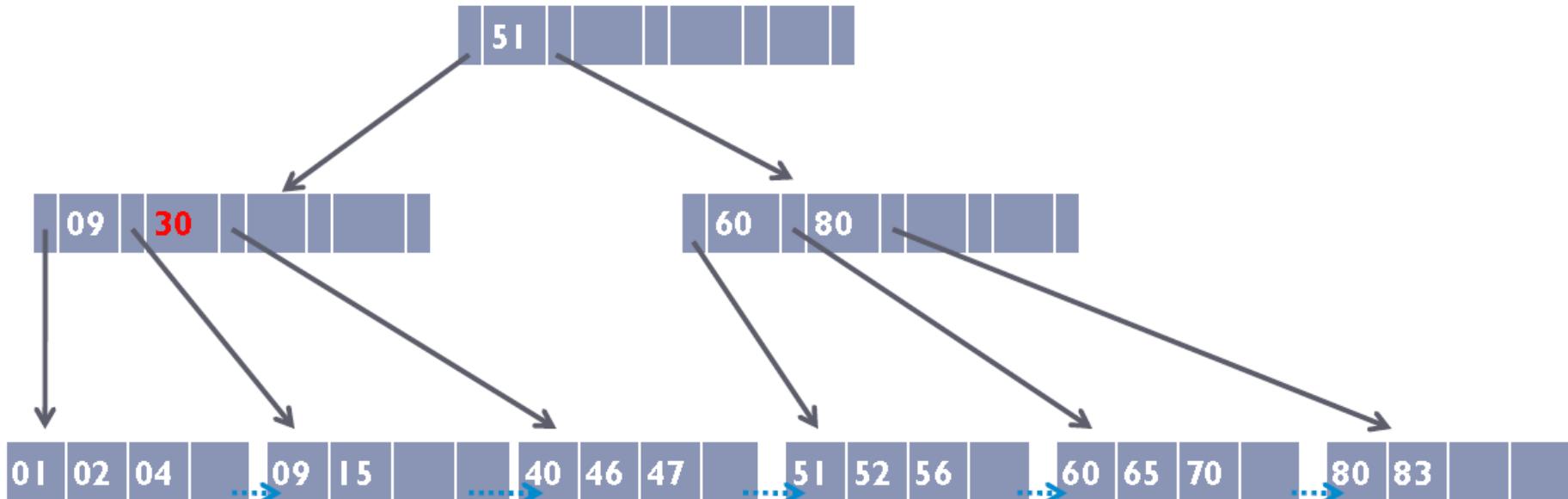


# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 30

$d = 2$



O valor 30 continua no índice!

# Árvores de Múltiplos Caminhos

## Árvores B+

- **Exclusão que causa concatenação:**

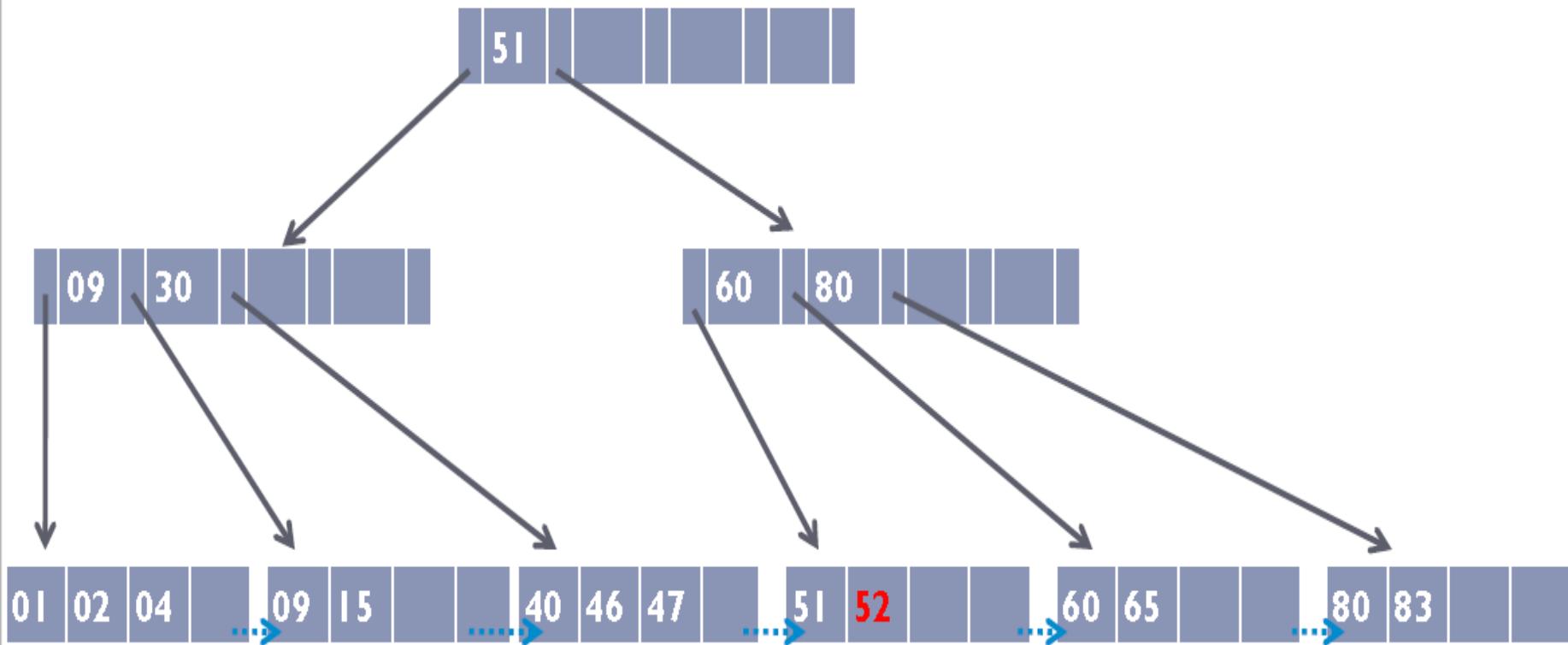
- Exclusões que causem concatenação de folhas podem se propagar para os nós internos da árvore.
- **Importante:**
  - Se a concatenação ocorrer na folha: a chave do nó pai não desce para o nó concatenado, pois ele não carrega dados com ele. Ele é simplesmente apagado.
  - Se a concatenação ocorrer em nó interno: usar a mesma lógica utilizada na árvore B.

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 52

$d = 2$

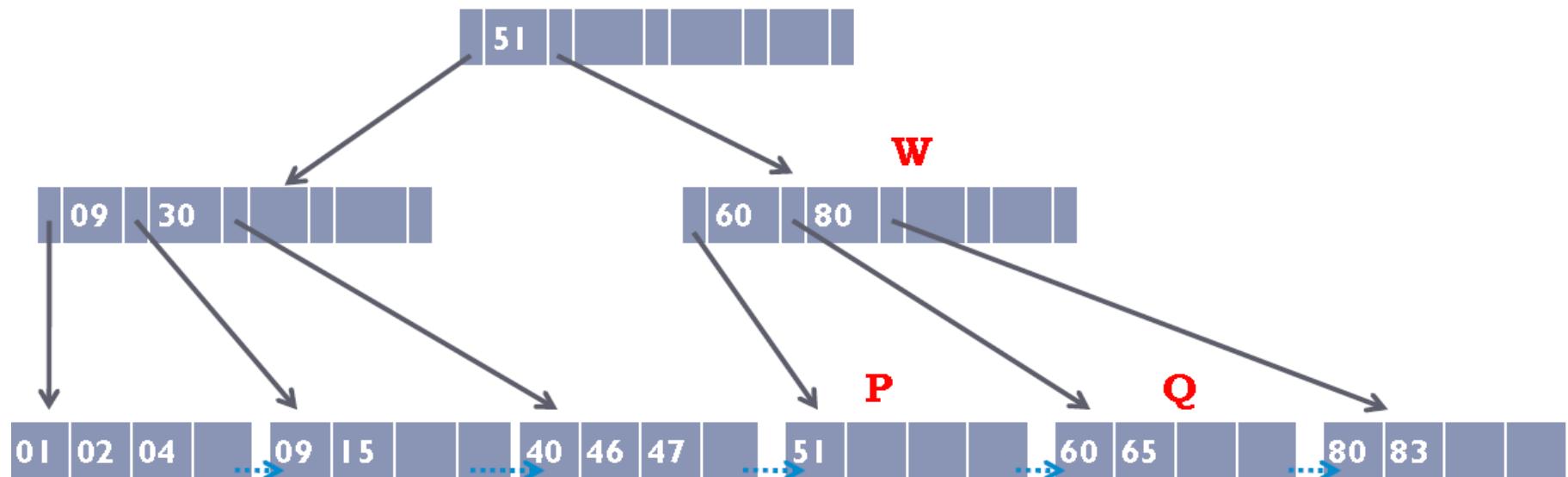


# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 52

$d = 2$



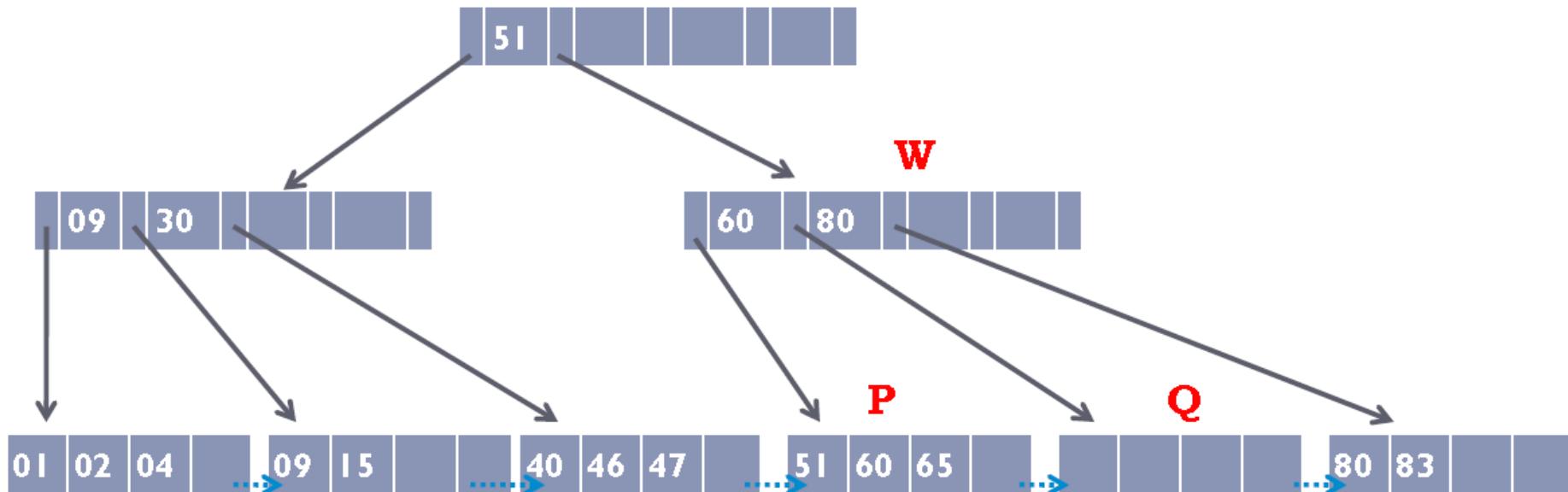
Nó ficou com menos de  $d$  entradas – necessário tratar isso  
Soma dos registros de P e Q  $< 2d$   
Usar concatenação

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Arvore B+  
Excluir chave 52

$d = 2$



Passar os registros de Q para P

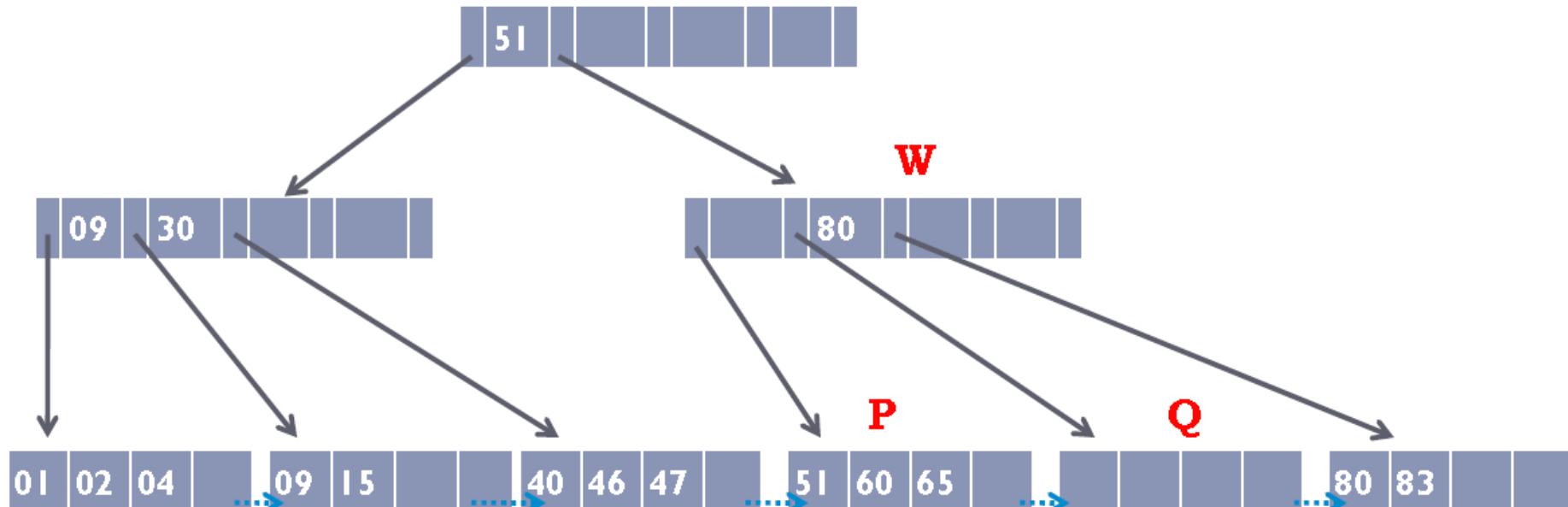
Eliminar a chave em W que divide os ponteiros para as páginas P e Q

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 52

ordem  $d = 2$



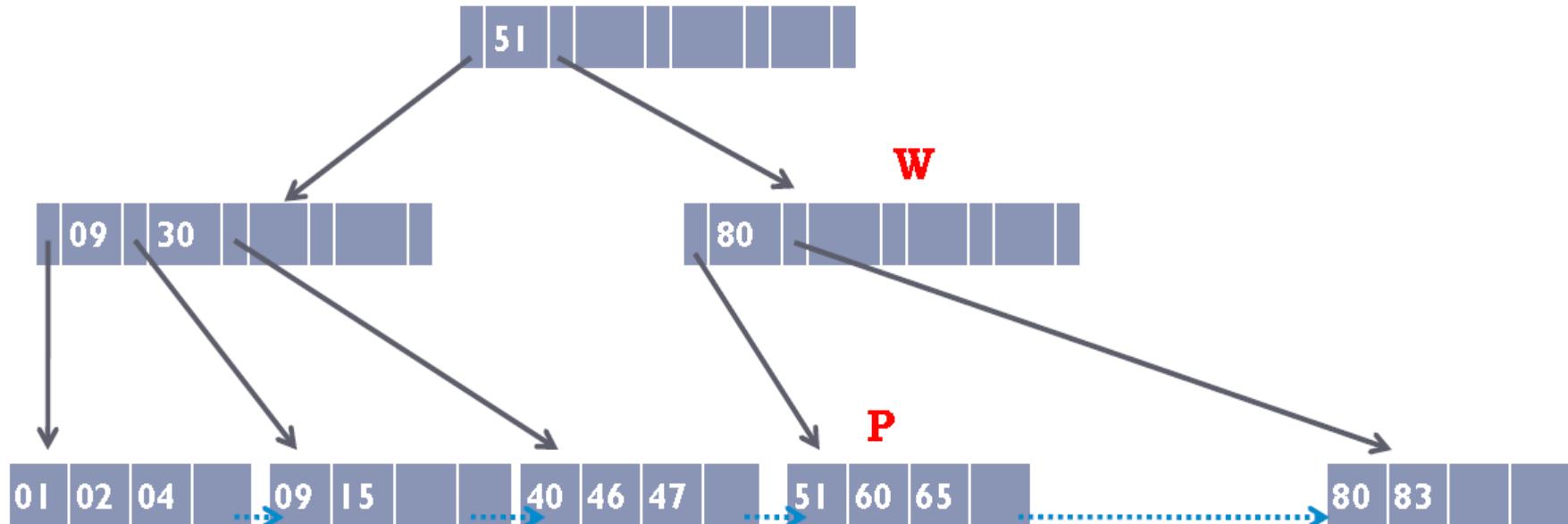
Eliminar ponteiro e nó Q

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 52

$d = 2$



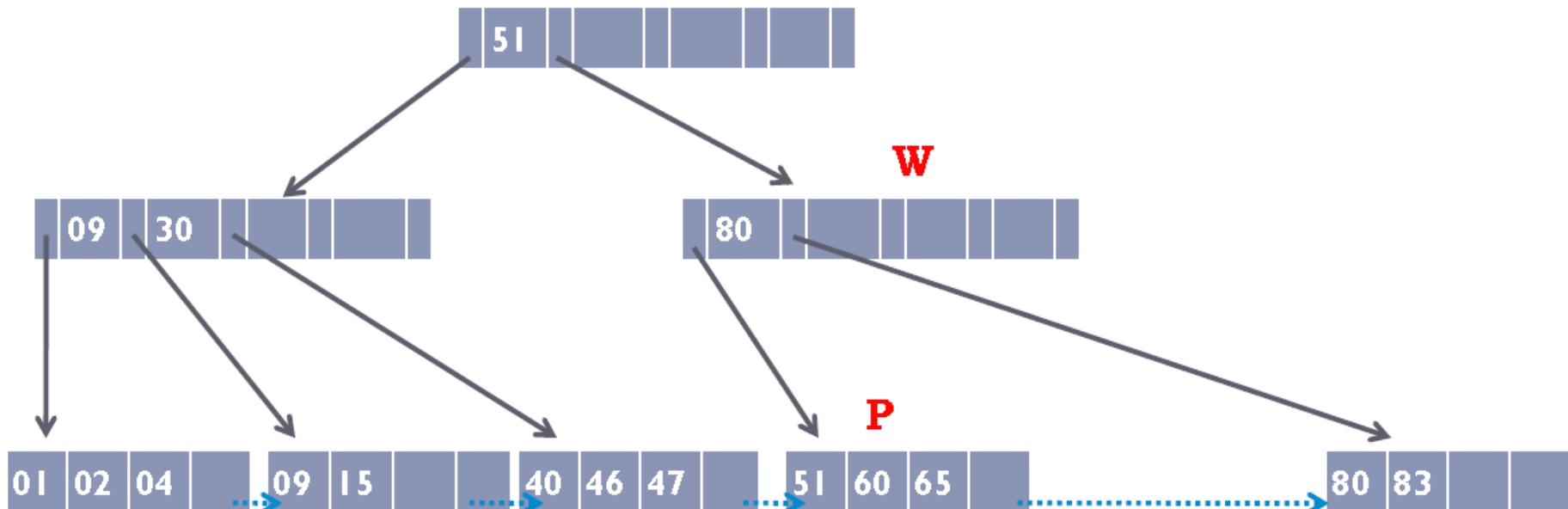
Eliminar ponteiro e nó Q, reajustar ponteiros e nó W

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 52

$d = 2$



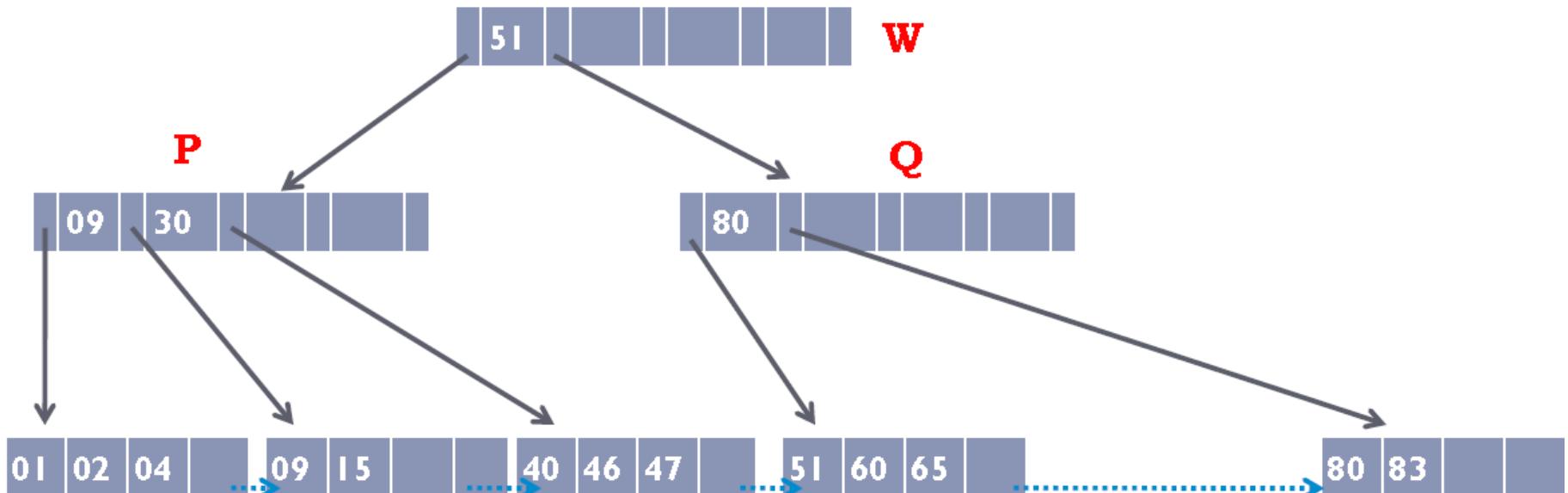
Nó W ficou com menos de  $d$  chaves

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 52

$d = 2$



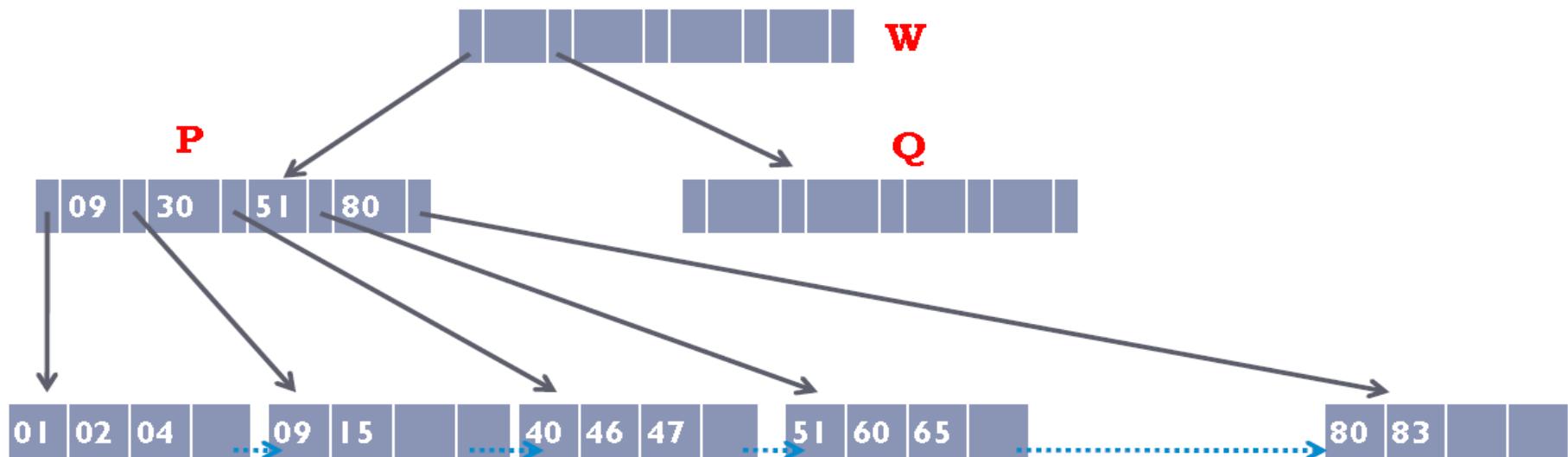
Soma de total de chaves de P e Q < 2d  
Solução: concatenação

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 52

$d = 2$



Transferir chaves para P

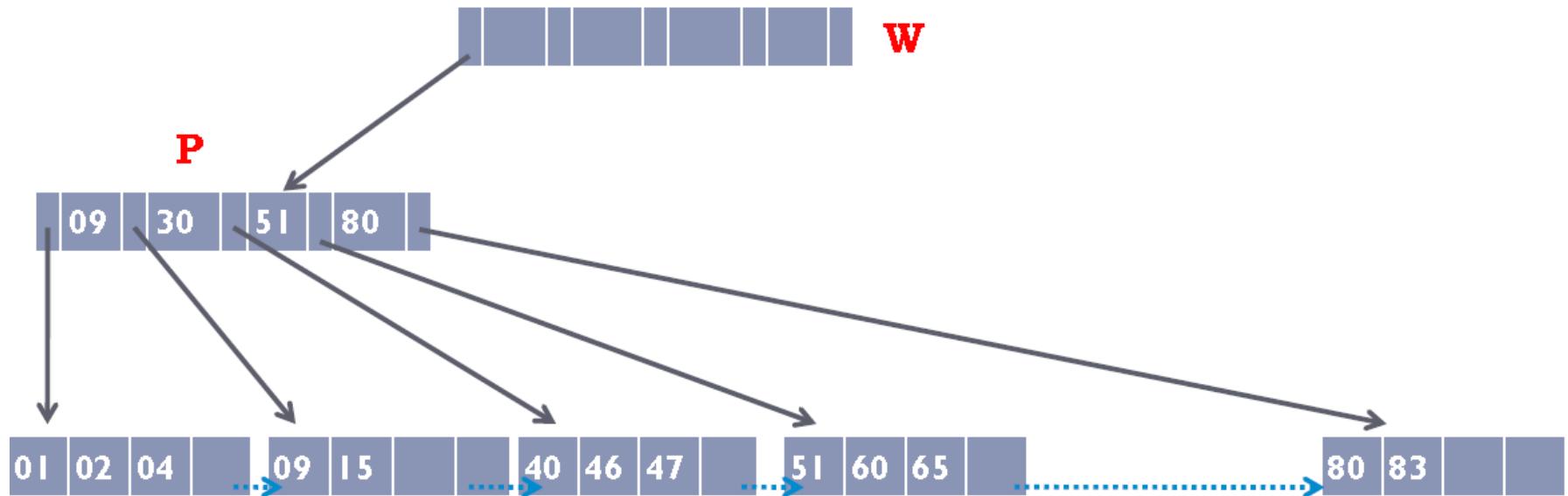
Atenção: com as páginas concatenadas não são folhas, chave em W também desce para P! (caso contrário, faltaria chave para separar os ponteiros para os filhos)

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 52

$d = 2$



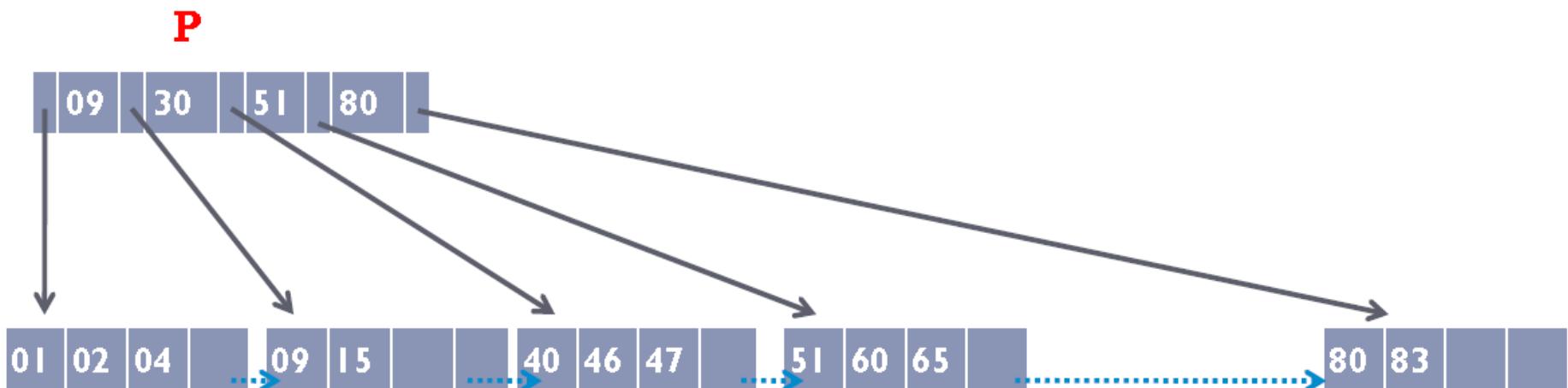
Apagar Q

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 52

$d = 2$



Como a raiz ficou vazia, apagar a raiz. P é a nova raiz.

# Árvores de Múltiplos Caminhos

## Árvores B+

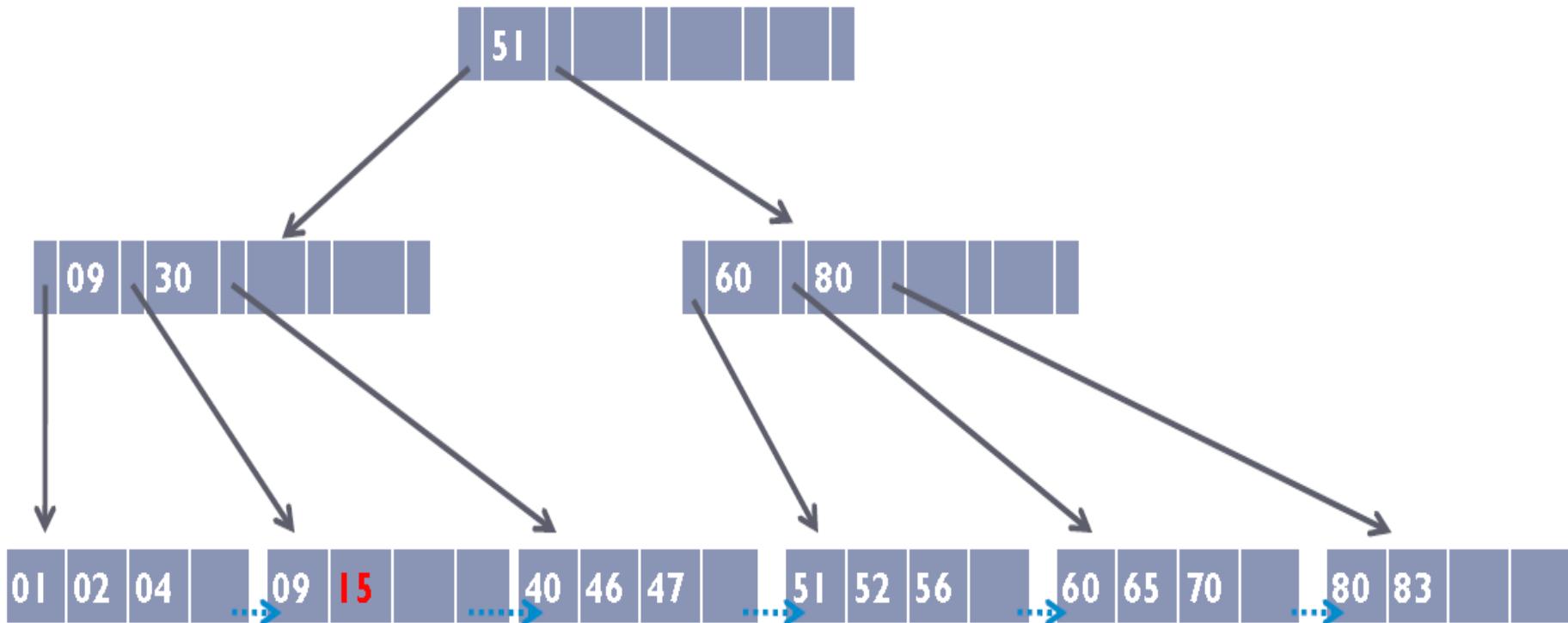
- **Exclusão que causa redistribuição:**
  - Exclusões que causem redistribuição dos registros nas folhas provocam mudanças no conteúdo do índice, mas não na estrutura (não se propagam).

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 15

$d = 2$

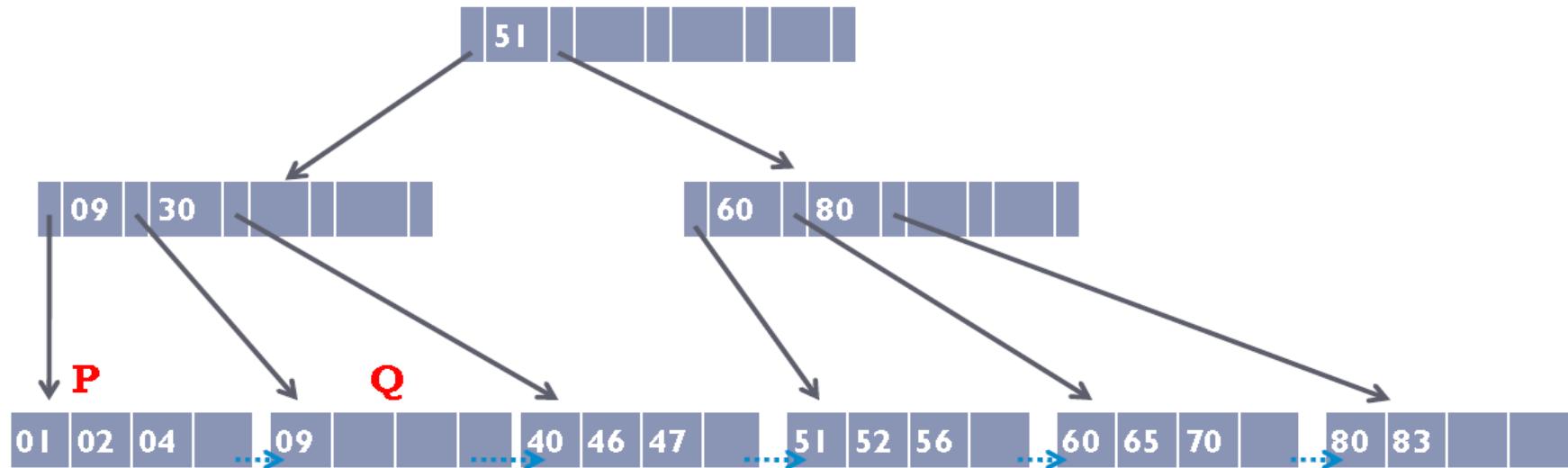


# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 15

$d = 2$



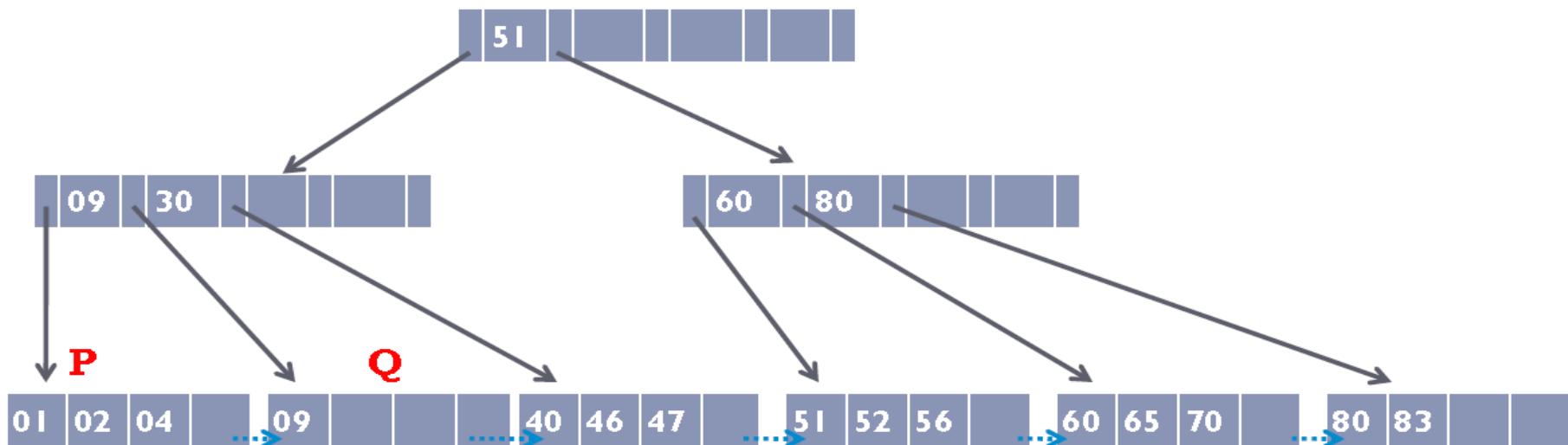
Nó ficou com menos de  $d$  entradas – necessário tratar isso  
P e Q não podem ser concatenadas, pois a soma dos registros  
não é menor  $2d$   
Solução: redistribuição

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 15

ordem  $d = 2$



**MAS...** Se a chave do nó pai não precisa descer (porque não tem conteúdo, tem apenas a chave), porque não podemos concatenar P e Q?

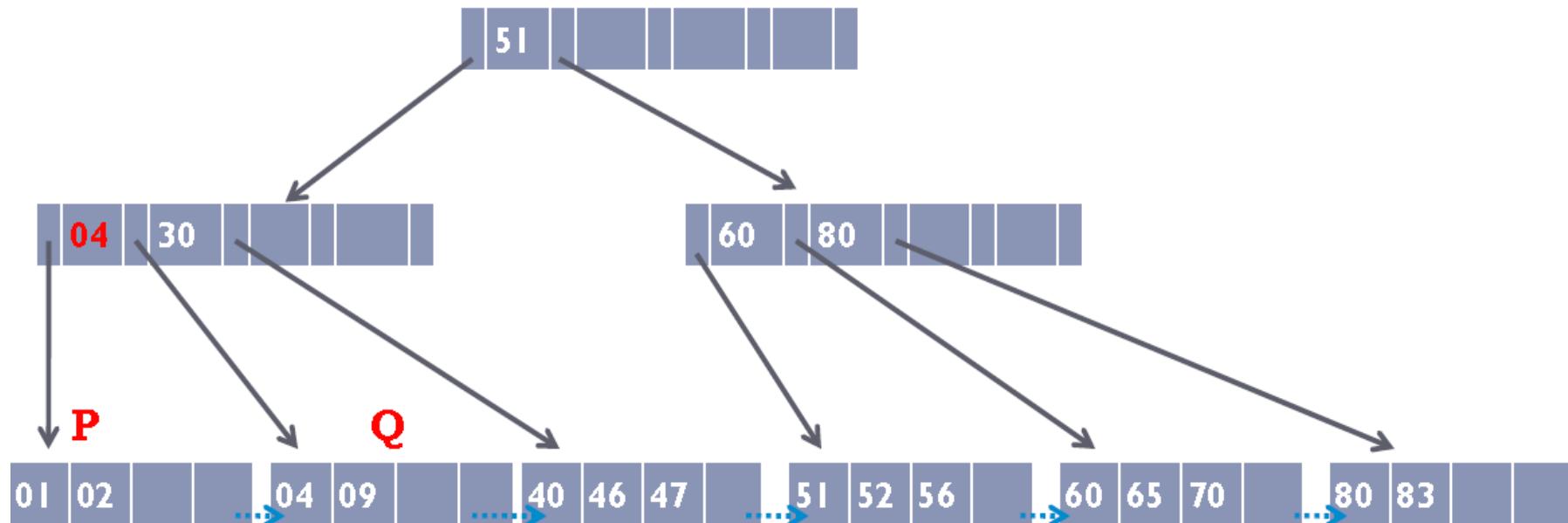
Resposta: ao concatenar P e Q, a página concatenada ficaria cheia, e a próxima inserção neste nó causaria um particionamento. Para evitar isso, continuamos obedecendo o critério : soma da quantidade de chaves  $< 2d$

# Árvores de Múltiplos Caminhos

## Árvores B+

Exemplo de Exclusão em Árvore B+  
Excluir chave 15

$d = 2$



Note que a chave 4 sobe para W, mas o registro correspondente é colocado em Q

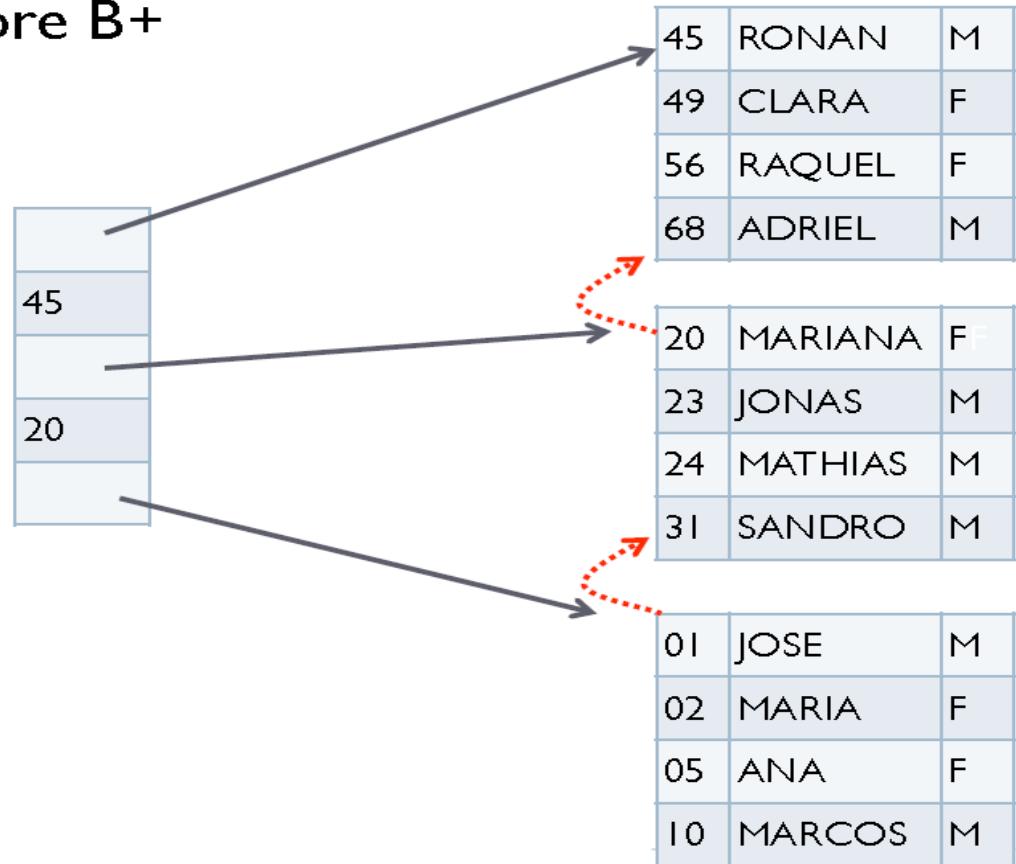
# Árvores de Múltiplos Caminhos

## Árvores B+

### Exemplo

(Mostrando os dados nas folhas)

- Neste exemplo, a árvore B+ tem apenas o nó raiz



# Quando usar qual Índice?

- Os índices **B-tree** podem tratar **consultas de igualdade e de faixa**, em dados que podem ser classificados em alguma ordem.
- Em particular, o **planejador de comandos do PostgreSQL** leva em consideração utilizar um **índice B-tree** sempre que uma **coluna indexada** está envolvida em uma **comparação utilizando um dos seguintes operadores: < , <=, = , >= e >**.

# Quando usar qual Índice?

- As construções equivalentes a combinações destes **operadores**, tais como **BETWEEN** e **IN**, também podem ser implementadas com **procura de índice B-tree** (Mas deve ser observado que **IS NULL** não é equivalente a = e **não é indexável**).

# Quando usar qual Índice?

- O otimizador também pode utilizar um índice **B-tree** nos **comandos** envolvendo os operadores de **correspondência** com padrão **LIKE**, **ILIKE**, **~** e **~\***, se o padrão estiver ancorado ao **início** da cadeia de caracteres como, por exemplo, em **col LIKE 'foo%'** ou **col ~ '^foo'**, mas não em **col LIKE '%bar'**.

# Quando usar qual Índice?

- Os **índices R-tree** são adequados para consultas a dados espaciais.
- Para criar um **índice R-tree** deve ser utilizado um **comando** da forma:
- **CREATE INDEX nome ON tabela USING RTREE (coluna);**

# Quando usar qual Índice?

- O **planejador de comandos** do PostgreSQL considera utilizar um **índice R-tree** sempre que a **coluna indexada** está envolvida em uma **comparação** utilizando um dos seguintes **operadores**:
- <<
- &<
- &>
- >>
- @
- ~=
- &&

**Tabela 9-30. Operadores geométricos**

| Operador | Descrição   | Exemplo  |
|----------|---|--|
| +        | Translação  | box '((0,0),(1,1))' + point '(2,0,0)'              |
| -        | Translação  | box '((0,0),(1,1))' - point '(2,0,0)'              |
| *        | Escala/rotação  | box '((0,0),(1,1))' * point '(2,0,0)'              |
| /        | Escala/rotação  | box '((0,0),(2,2))' / point '(2,0,0)'              |
| #        | Ponto ou caixa de interseção                                | '((1,-1),(-1,1))' # '((-1,1),(-1,-1))'             |
| #        | Número de pontos do caminho ou do polígono                  | # '((1,0),(0,1),(-1,0))'                           |
| @-@      | Comprimento ou circunferência                               | @-@ path '((0,0),(1,0))'                           |
| @@       | Centro  | @@ circle '((0,0),10)'                             |
| ##       | Ponto mais próximo do primeiro operando no segundo operando | point '(0,0)' ## lseg '((2,0),(0,2))'              |
| <->      | Distância entre   | circle '((0,0),1)' <-> circle '((5,0),1)'          |
| &&       | Se sobrepõem?   | box '((0,0),(1,1))' && box '((0,0),(2,2))'         |
| &<       | Não se estende à direita de?                                | box '((0,0),(1,1))' &< box '((0,0),(2,2))'         |
| &>       | Não se estende à esquerda de?                               | box '((0,0),(3,3))' &> box '((0,0),(2,2))'         |
| <<       | Está à esquerda?  | circle '((0,0),1)' << circle '((5,0),1)'           |
| >>       | Está à direita?   | circle '((5,0),1)' >> circle '((0,0),1)'           |
| <^       | Está abaixo?  | circle '((0,0),1)' <^ circle '((0,5),1)'           |
| >^       | Está acima?   | circle '((0,5),1)' >^ circle '((0,0),1)'           |
| ?#       | Se intersectam?   | lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))'     |
| ?-       | É horizontal?   | ?- lseg '((-1,0),(1,0))'                           |
| ?-       | São alinhados horizontalmente?                              | point '(1,0)' ?- point '(0,0)'                     |
| ?        | É vertical?   | ?  lseg '((-1,0),(1,0))'                           |
| ?        | São alinhados verticalmente                                 | point '(0,1)' ?  point '(0,0)'                     |
| ? -      | São perpendiculares?  | lseg '((0,0),(0,1))' ? - lseg '((0,0),(1,0))'      |
| ?        | São paralelos?  | lseg '((-1,0),(1,0))' ?   lseg '((-1,2),(1,2))'    |
| ~        | Contém?   | circle '((0,0),2)' ~ point '(1,1)'                 |
| @        | Está contido ou sobre?                                      | point '(1,1)' @ circle '((0,0),2)'                 |
| ~=       | O mesmo que?  | polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))' |

# Quando usar qual Índice?

- Os **índices hash** podem tratar apenas comparações de igualdade simples.
- O **planejador de comandos** do **PostgreSQL** considera utilizar um índice **hash** sempre que a coluna indexada está envolvida em uma **comparação** utilizando o **operador =**.

# Quando usar qual Índice?

- O seguinte comando é utilizado para criar um índice **hash**:
- **CREATE INDEX nome ON tabela USING HASH (coluna);**
- Nota: Os testes mostraram que os índices **hash** do PostgreSQL **não têm desempenho melhor do que os índices B-tree**, e que o **tamanho** e o **tempo de construção** dos índices **hash** são muito piores. Por estas razões, desencoraja-se a utilização dos índices **hash**.

# Quando usar qual Índice?

- Os índices **GiST** não são um único tipo de índice, mas em vez disto uma infraestrutura dentro da qual podem ser implementadas muitas estratégias de indexação diferentes.
- Assim sendo, os operadores em particular com os quais o índice **GiST** pode ser utilizado variam dependendo da estratégia de indexação (a classe de operadores).

# Quando usar qual Índice?

- O método de **índice B-tree** é uma implementação das **árvores B** de alta-simultaneidade de Lehman-Yao.
- O método de **índice R-tree** implementa árvores R padrão utilizando o algoritmo de partição quadrática de Guttman.
- O método de **índice hash** é uma implementação do **hashing linear** de Litwin.
- São mencionados os **algoritmos** utilizados somente para indicar que todos estes métodos de índice são inteiramente **dinâmicos, não necessitando de otimização periódica** (como é o caso, por exemplo, dos métodos de acesso **hash estáticos**).