

Árvore B+

Prof. Dr. Lucas C. Ribas

Disciplina: Estrutura de Dados II

Departamento de Ciências de Computação e Estatística



UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”



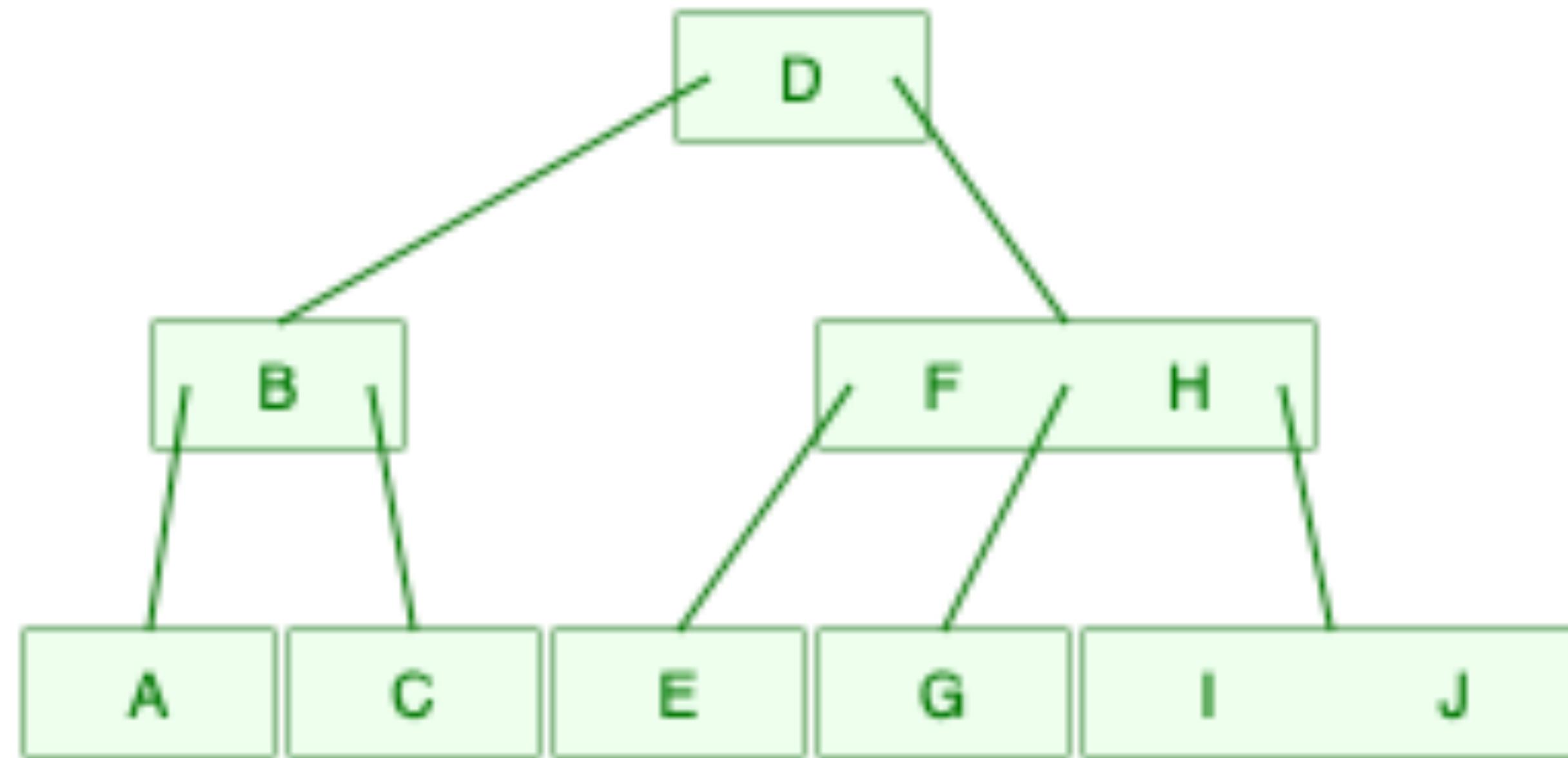
IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO



- Crie uma árvore-B de ordem 3 inserindo as chaves **A, B, C, D, E, F, G, H, I, J** nessa sequencia.

$$\begin{aligned}m &= 3 \\m-1 &= 2 \text{ (max)} \\m/2 - 1 &= 1 \text{ (min)}\end{aligned}$$

- Crie uma árvore-B de ordem 3 inserindo as chaves **A, B, C, D, E, F, G, H, I, J** nessa sequencia.



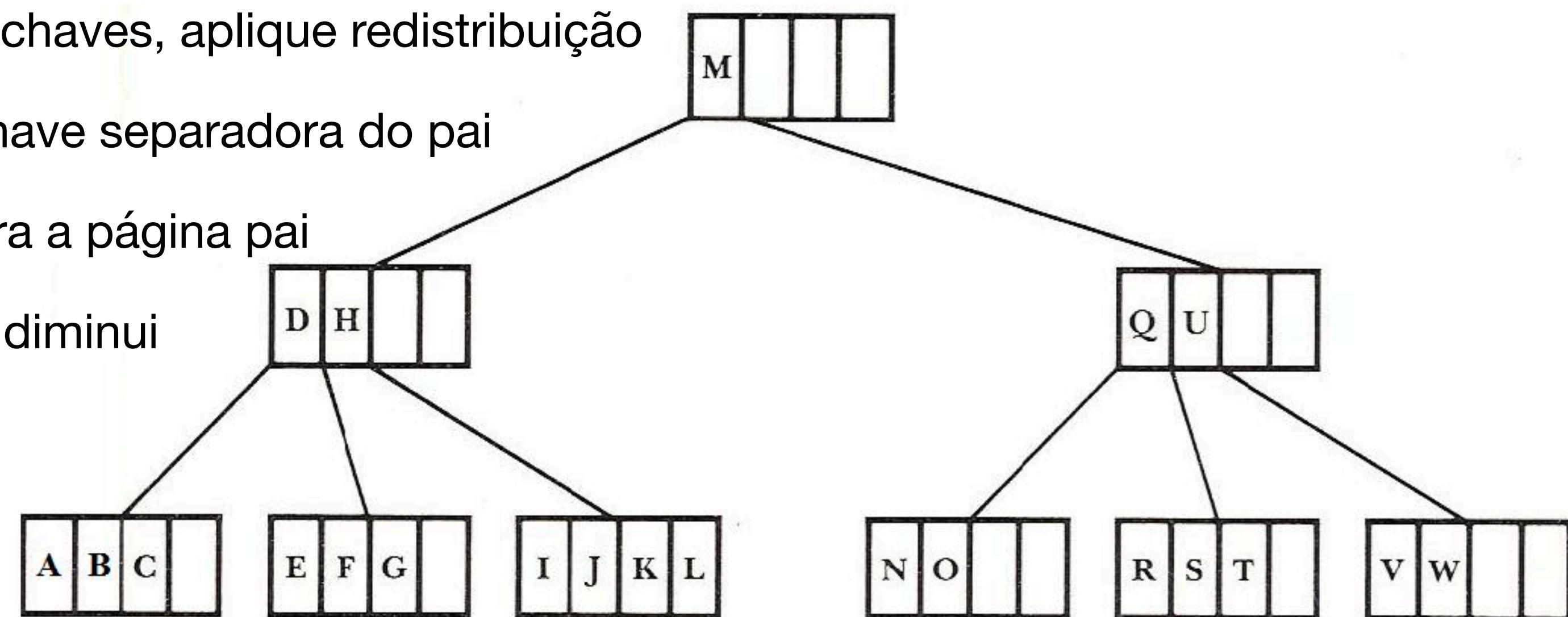
$$\begin{aligned}m &= 3 \\m-1 &= 2 \text{ (max)} \\m/2 - 1 &= 1 \text{ (min)}\end{aligned}$$



● Usando o algoritmo anterior, remova as chaves **A, B, Q, R e M** da árvore-B de ordem 5 abaixo

1. Se a chave não estiver numa folha, troque-a com sua sucessora*
2. Elimine a chave da folha
3. Se a página continuar com o número mínimo de chaves, fim
4. Se a página tem uma chave a menos que o mínimo, verifique as páginas irmãs a esquerda e a direita
- 4.1. se uma delas tiver mais do que o número mínimo de chaves, aplique redistribuição
- 4.2. senão concatene a página com uma das irmãs e a chave separadora do pai
5. Se ocorreu concatenação, aplique os passos de 3 a 6 para a página pai
6. Se a última chave da raiz for removida, a altura da árvore diminui

$$\begin{aligned}
 m &= 6 \\
 m-1 &= 5 \text{ (max)} \\
 m/2 - 1 &= 2 \text{ (min)}
 \end{aligned}$$



Tipos de Acesso a Arquivos



● Alternativas (até o momento)

- acesso indexado
 - arquivo pode ser visto como um conjunto de registros que são indexados por uma chave
- acesso sequencial
 - arquivo pode ser acessado sequencialmente (i.e., registros fisicamente contínuos)

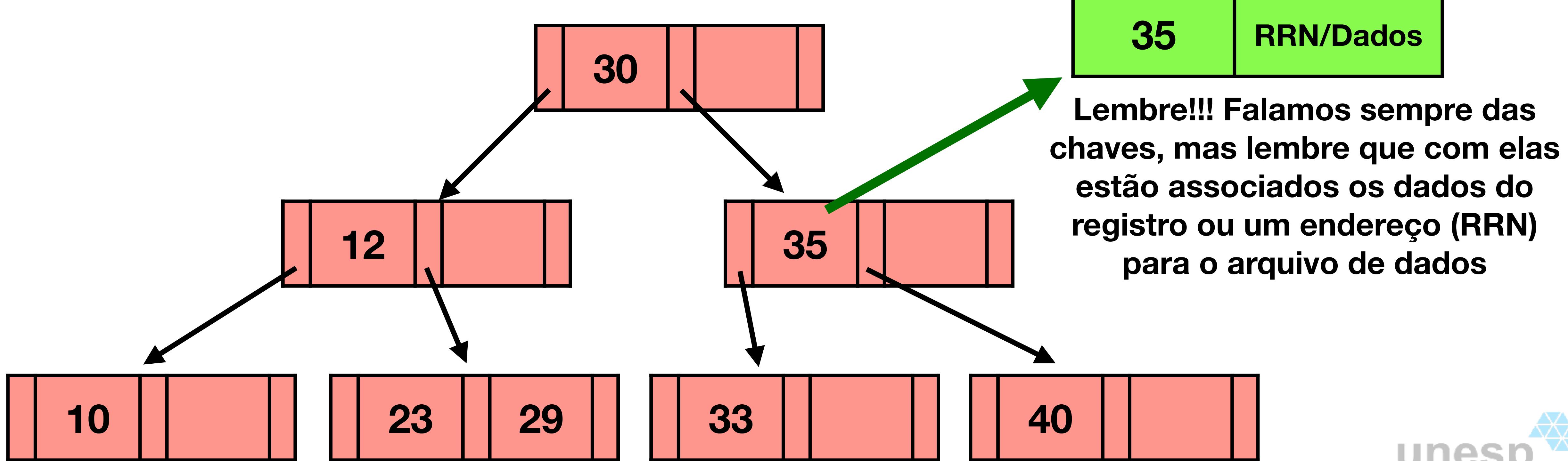
● Objetivo

- arquivos devem suportar acesso indexado eficiente, e também acesso sequencial



- Arquivo indexado por um índice árvore-B

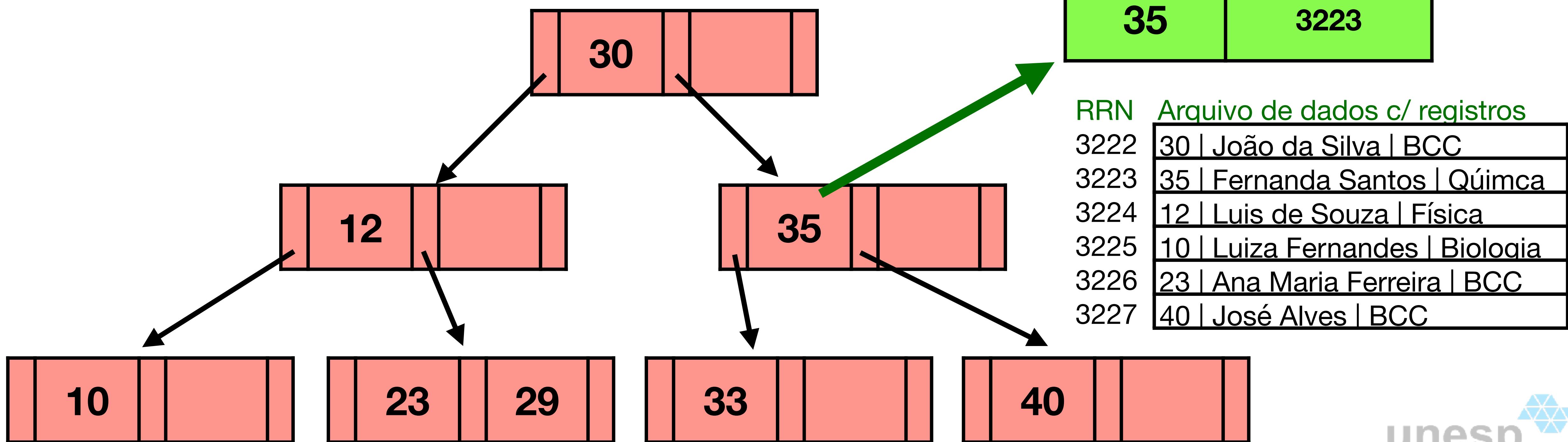
- acesso (aleatório) indexado pela chave: desempenho excelente ☺
- acesso sequencial (range) aos registros ordenados pela chave: desempenho péssimo ☹





- Arquivo indexado por um índice árvore-B

- acesso (aleatório) indexado pela chave: desempenho excelente ☺
- acesso sequencial (range) aos registros ordenados pela chave: desempenho péssimo ☹





- Arquivo com **registros ordenados** pela chave

- processamento **sequencial** (acessar todos registros ou range): **apropriado** 😊 **(buferização)**
- processamento **randômico**: **inapropriado** 😞 - **logarítmico (ordem 2 – busca binária)**

201901 João da Silva BCC
201904 Fernanda Santos Qúimica
202006 Luis de Souza Física
202008 Luiza Fernandes Matemática
202101 Ana Maria Ferreira BCC
202103 José Alves BCC



- Organizar um arquivo de modo que seja eficiente tanto para processamento sequencial quanto aleatório



- Variante da árvore B que permite acesso sequencial mais eficiente aos registros
- Desenvolvida pensando em buscas por range de elementos em bancos de dados
 - Exemplo: retorne todos os registros com chave entre 20 e 100
- Importante 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](#), [Sybase ASE](#), [PostgreSQL](#), [Firebird](#), [MariaDB](#) e [SQLite](#) permitem o uso deste tipo de árvore para indexar tabelas



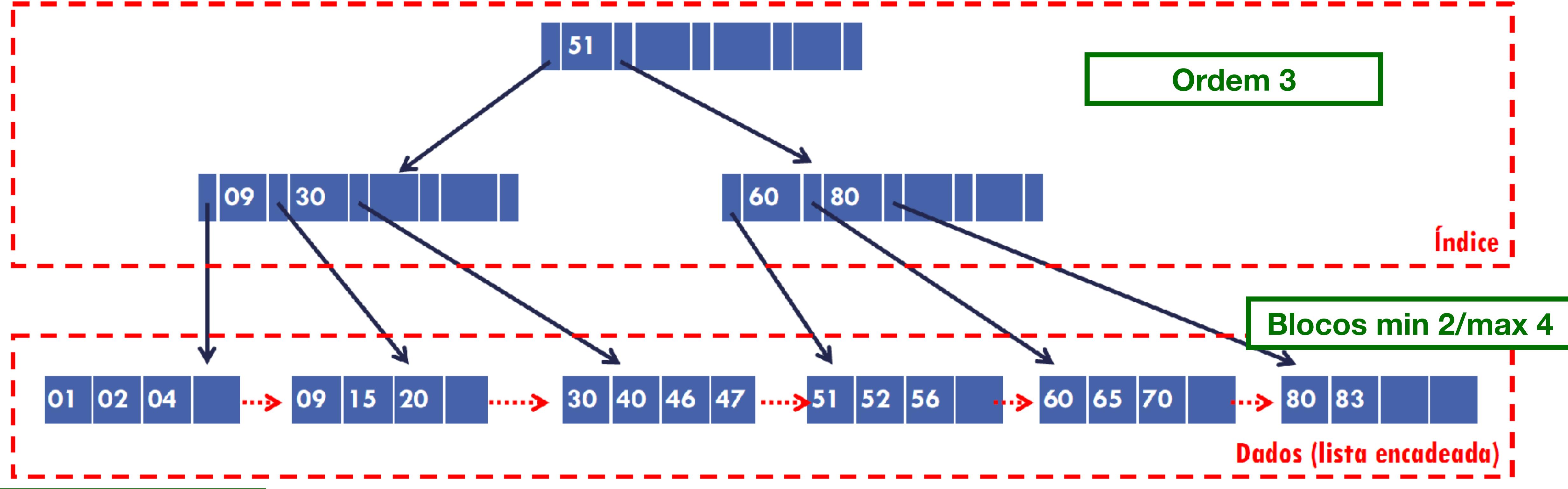
- **Conceito:** é a estrutura híbrida formada pela combinação dos conceitos de **Árvore B** e **Conjunto de Seqüência**.
- Árvore é dividida em:
 - **Conjunto de índices (Index Set):** organizado como uma **Árvore-B formado pelos nós internos**
 - **Privilegiando busca aleatória**
 - **Conjunto de seqüência (Sequence Set):** lista encadeada que **liga as folhas da árvore-B (blocos de tamanho fixo, de registros sequenciais, ordenados pelas chaves)**, de modo a oferecer um caminho seqüencial para percorrer as chaves na árvore
 - **Privilegiando o acesso sequencial**



- Mesmos princípios da árvore B, com as seguintes diferenças:
 1. Os **nós internos** não contém dados secundários dos registros, apenas cópias das chaves e servem só como referência para o percurso. Em outras palavras, abrigam apenas separadores de chaves.
 2. Apenas as **folhas** contém os **dados dos registros ou ponteiros** para os mesmos em disco (RRNs) e, por consequência, as **chaves em nós internos são replicadas nas folhas**
 3. As folhas são interconectadas via uma lista (**possivelmente duplamente**) ligada de modo que possam ser acessadas independentemente da árvore

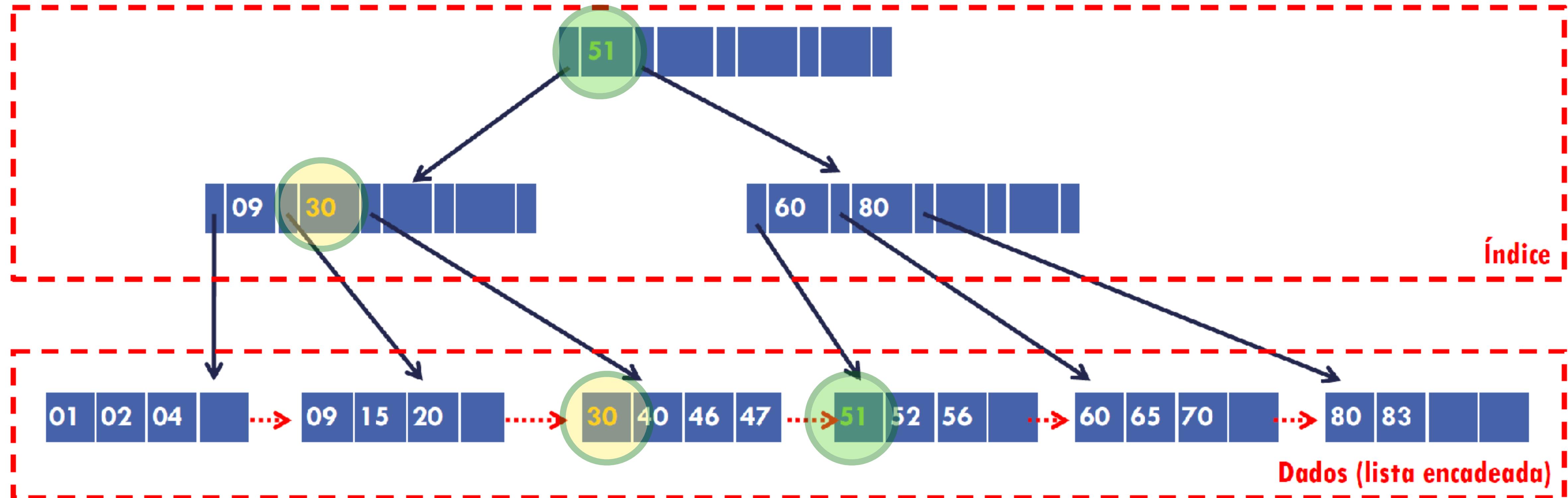


Nós internos como em uma árvore B. Servem como índice.



Blocos folhas
podem ter tamanho
diferente da ordem
da árvore

Páginas/Nós folhas formam um **Conjunto de Sequência** que estão
ligados por ponteiros formando uma lista encadeada (ou duplamente).
Este conjunto deve ser organizado em blocos (tamanho buffer E/S).



IMPORTANTE: Índices *repetem valores* de chave que aparecem nas folhas (diferente do que acontece nas árvores B)



- Mecanismo para **percorrer seqüencialmente** o arquivo de registros de dados **sem que seja necessário visitar toda a árvore**
- Mecanismo para **percorrer seqüencialmente** o arquivo de registros de dados **sem que seja necessário ordenar o arquivo de registro de dados**



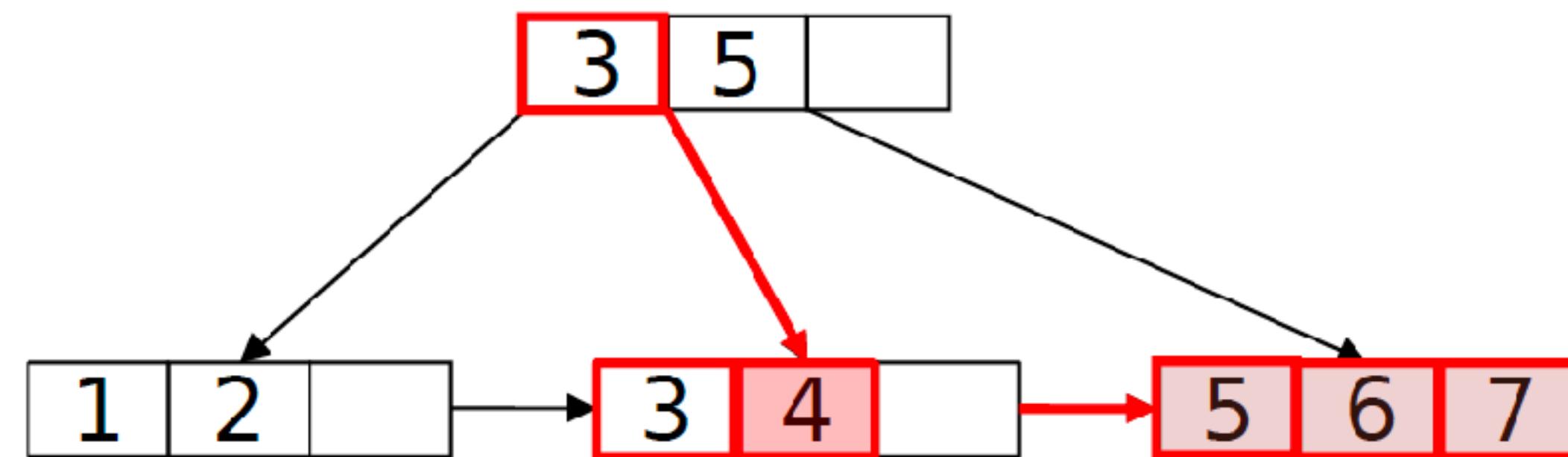
- Quando se deseja que um mesmo arquivo seja acessado seqüencialmente e/ou por índice
 - **p/ acessar seqüencialmente utilizar -> Conj.de seqüênci**
 - **p/ acessar por índice -> Árvore-B (conjunto de índices)**
- Algumas aplicações requerem 2 pontos de vista de um arquivo. Exemplo:
 - Sistema de cadastro de alunos em uma universidade
 - **Visão indexada -> Buscar um aluno**
 - **Visão sequencial -> Emissão de carteira de estudantes**



- É importante a **separação lógica** da árvore-B+ em Conjunto de Índices e Conjunto de Seqüência
 - podemos fazer a maioria das inclusões e exclusões no conjunto de seqüência sem alterar o índice
 - quando houver necessidade de inclusão ou exclusão do referido índice, usamos os algoritmos já conhecidos da árvore-B.



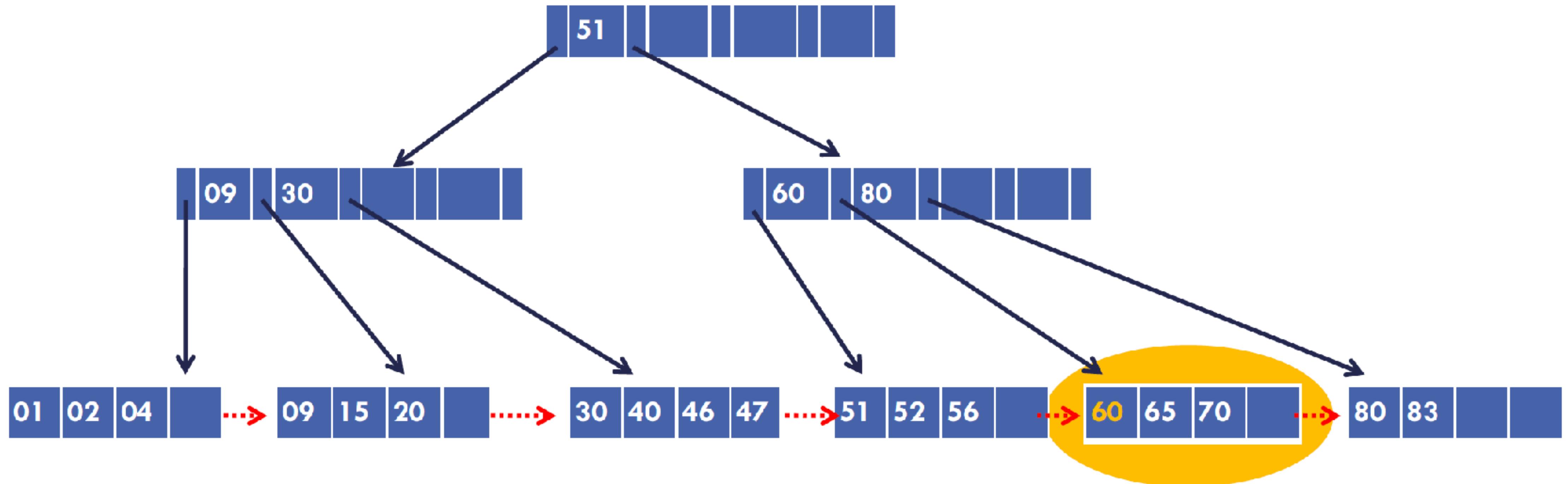
- As operações de busca, inserção e remoção são efetuadas de modo similar à arvores B
- Uma busca por range de chaves é simplificada à encontrar a primeira chave de interesse, seguida por uma operação de busca linear na lista ligada de folhas
- Exemplo: Retorne todos os registros entre [4, 7]





- Só se pode ter certeza de que o registro foi encontrado **quando se chega em uma folha**
- Notar que comparações agora devem considerar a igualdade também
 - Achou chave **maior** que a chave buscada, desce pelo ponteiro da esquerda
 - Achou chave **igual** à chave buscada ou chegou ao fim da lista de chaves do nó, desce pelo ponteiro da direita

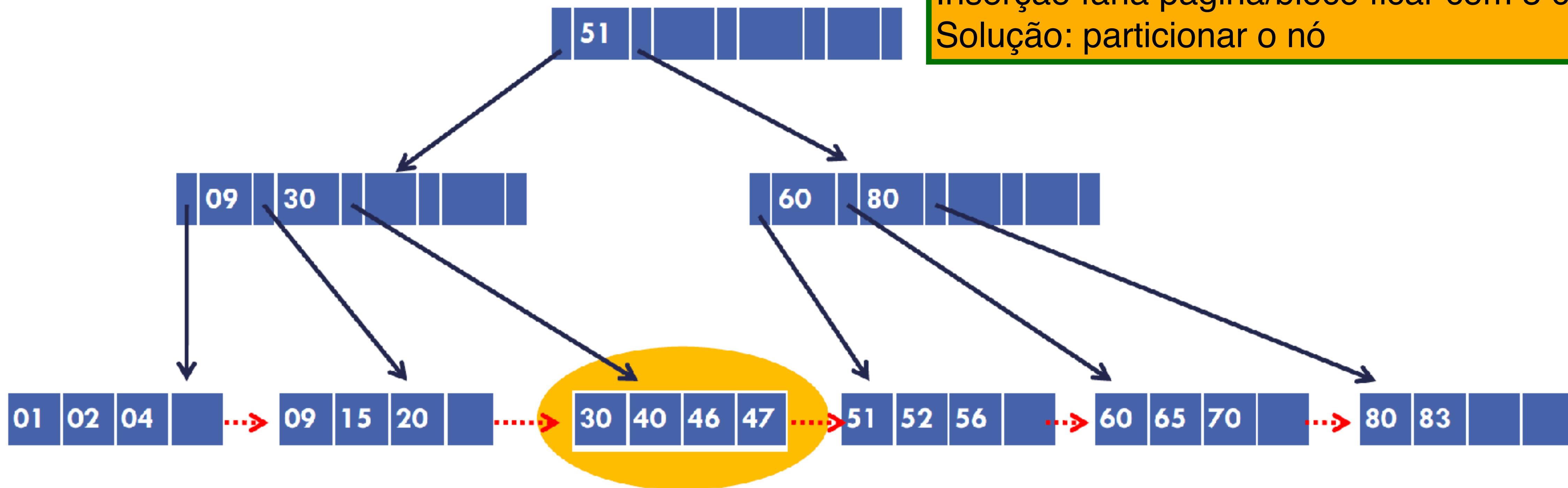
Árvore B+ Busca 60





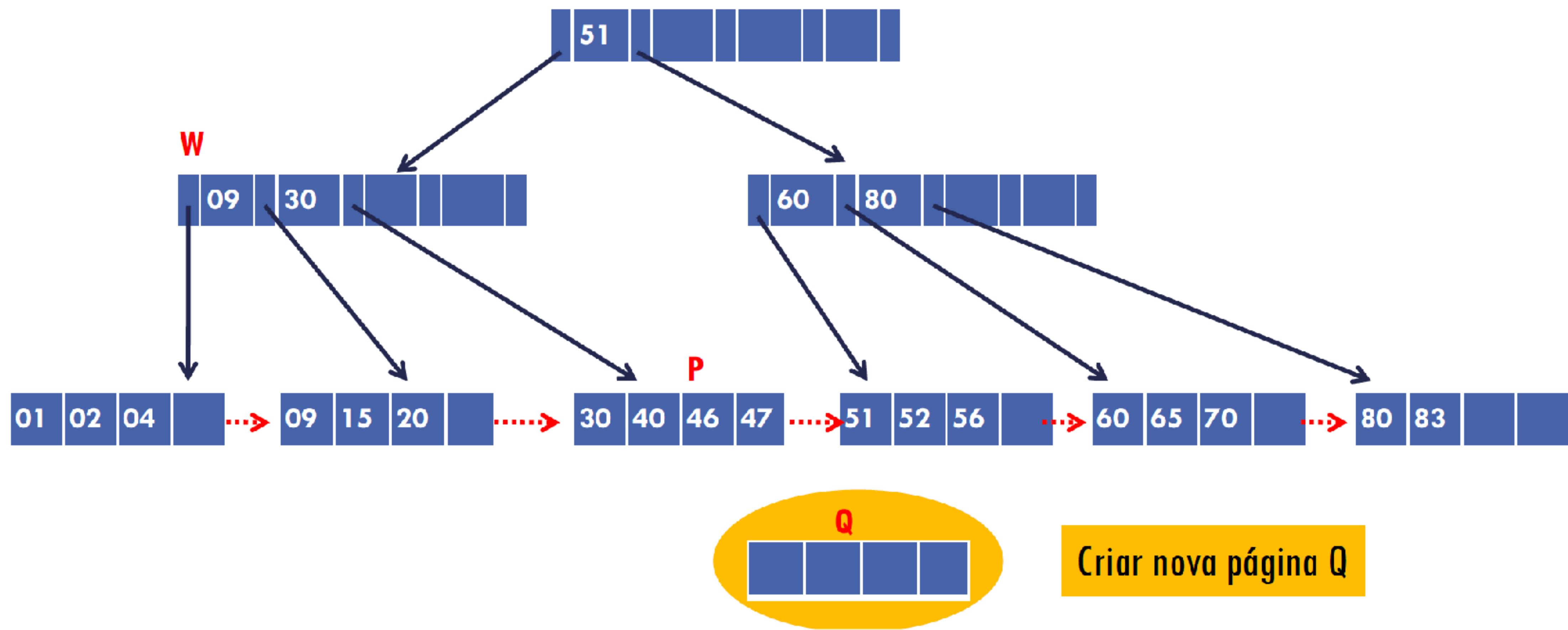
- Quando for necessário particionar um nó durante uma inserção, o mesmo raciocínio do particionamento em Árvore B é utilizado
 - Se for uma inserção (sempre em folhas): somente o valor da **chave mediana** deve ser copiada para o novo nó pai. O **registro fica na folha (isto é, o RRN daquela chave)**, juntamente com a sua chave
 - Se não for folha, o procedimento é o mesmo utilizado na árvore B. Neste caso, as chaves internas não possuem o arquivo de dados ou RRN

Árvore B+ Inserção Chave 32



Tamanho máximo do bloco/página folha: $B=4$
Inserir chave 32
Inserção faria página/bloco ficar com 5 chaves
Solução: partitionar o nó

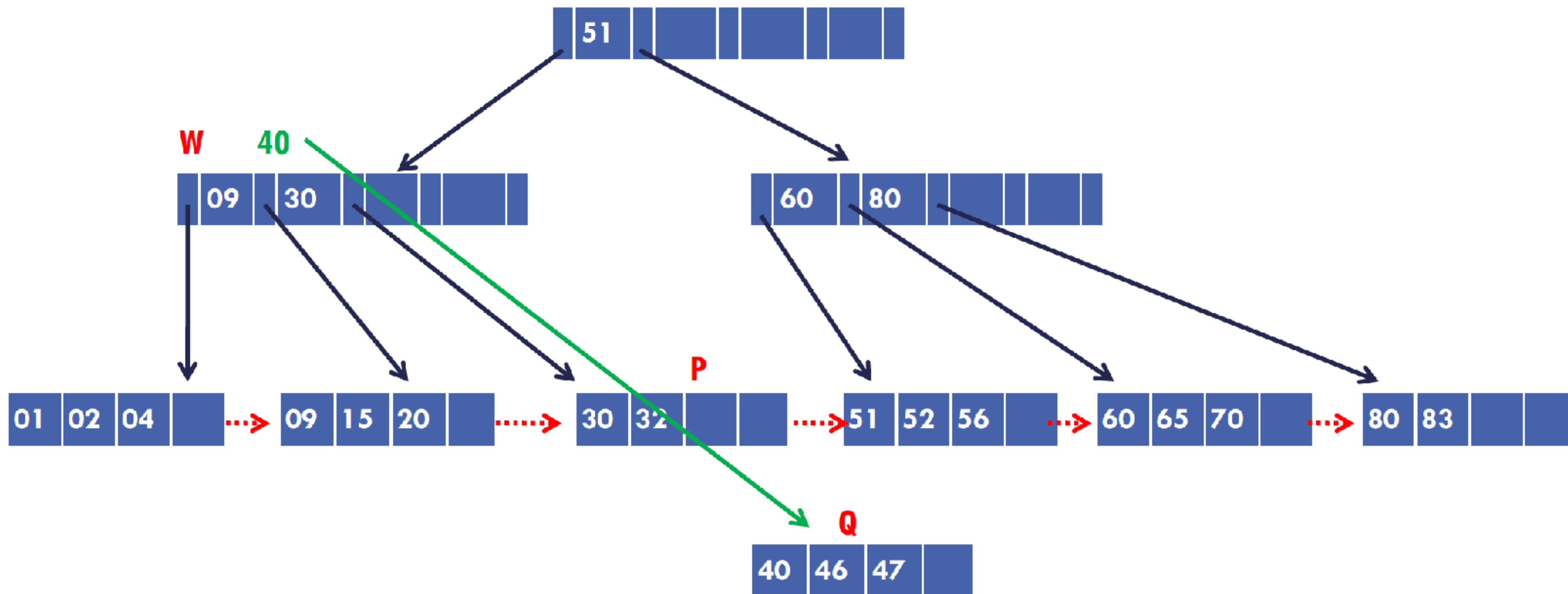
Árvore B+ Inserção Chave 32



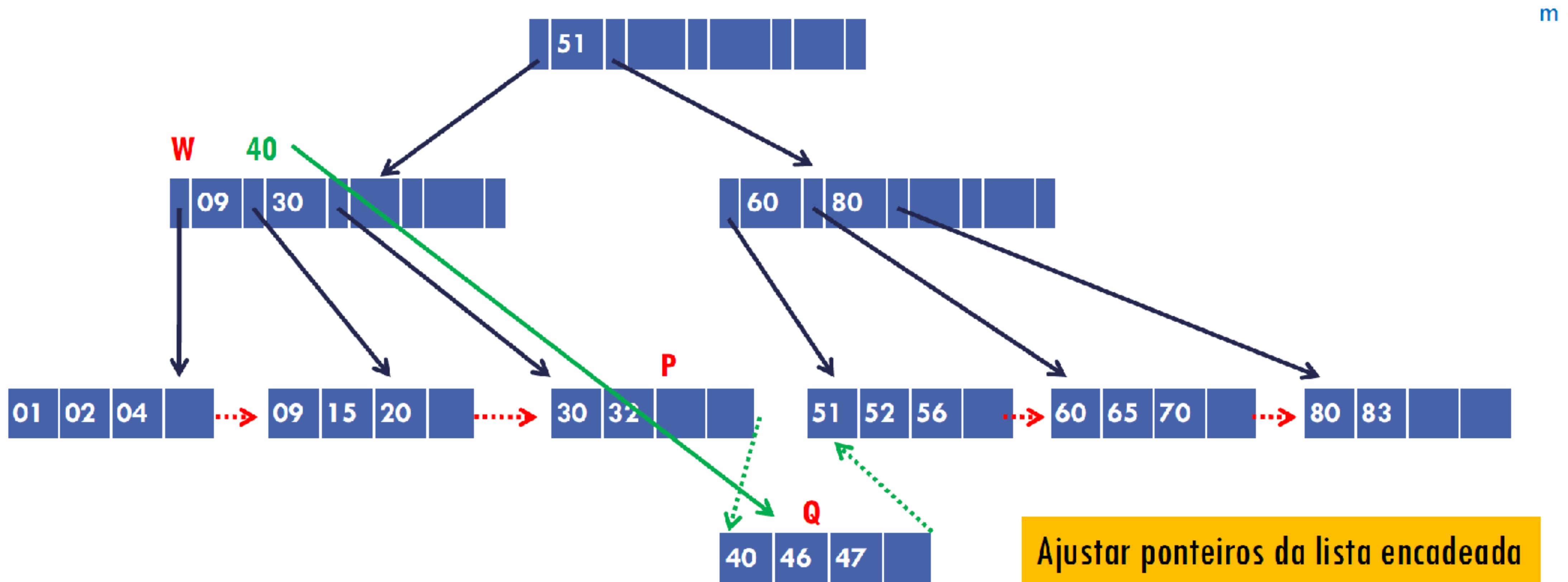
Árvore B+ Inserção Chave 32



1. Dividir as chaves entre as duas páginas (30; 32; 40; 46; 47)
2. ($B/2 = 2$) chaves na página original **P**
3. chave da posição $(B/2)+1$ sobe para nó pai **W** (*mas registro é mantido na nova página*)
4. As $(B/2)+1$ chaves restantes na nova página **Q**



Árvore B+ Inserção Chave 32

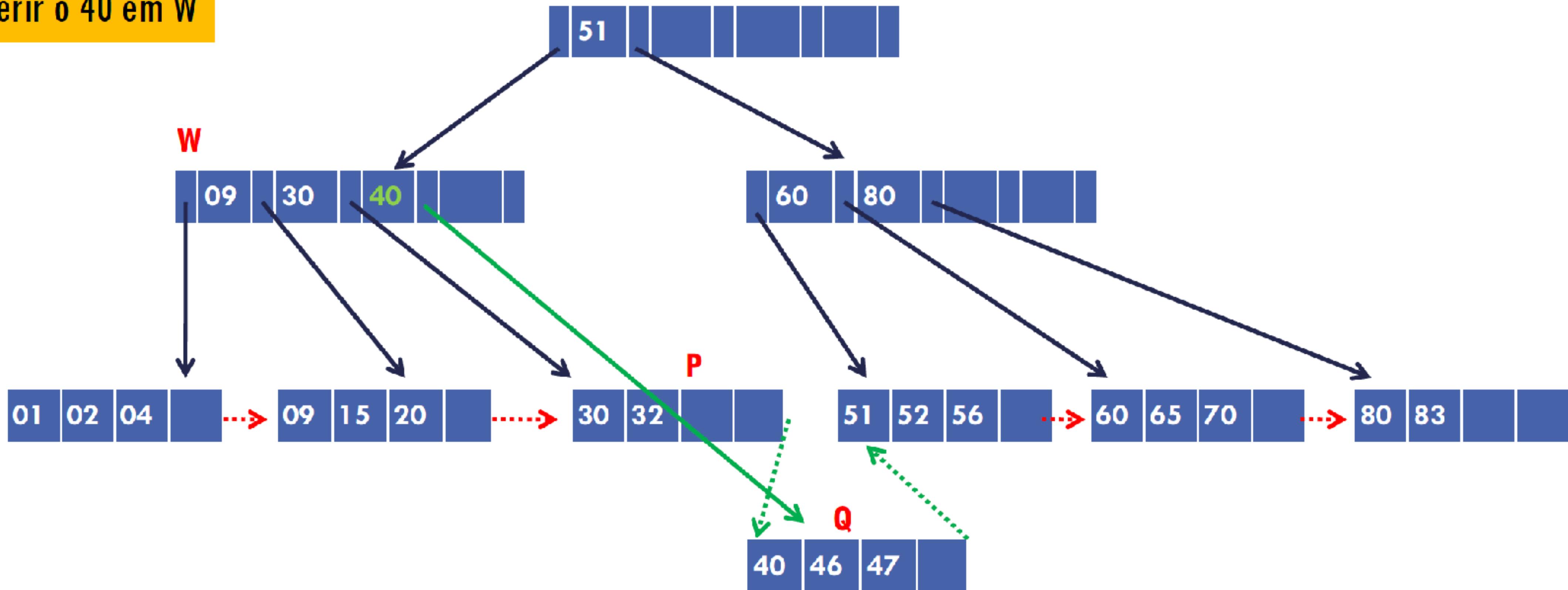


m c

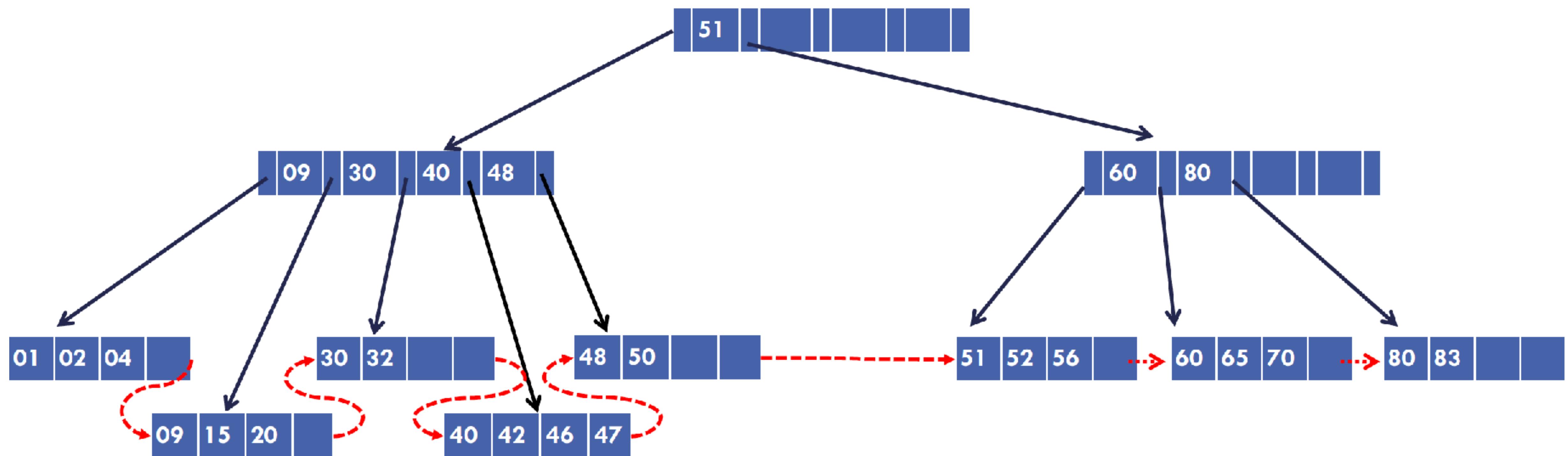
Árvore B+ Inserção Chave 32



Inserir o 40 em W



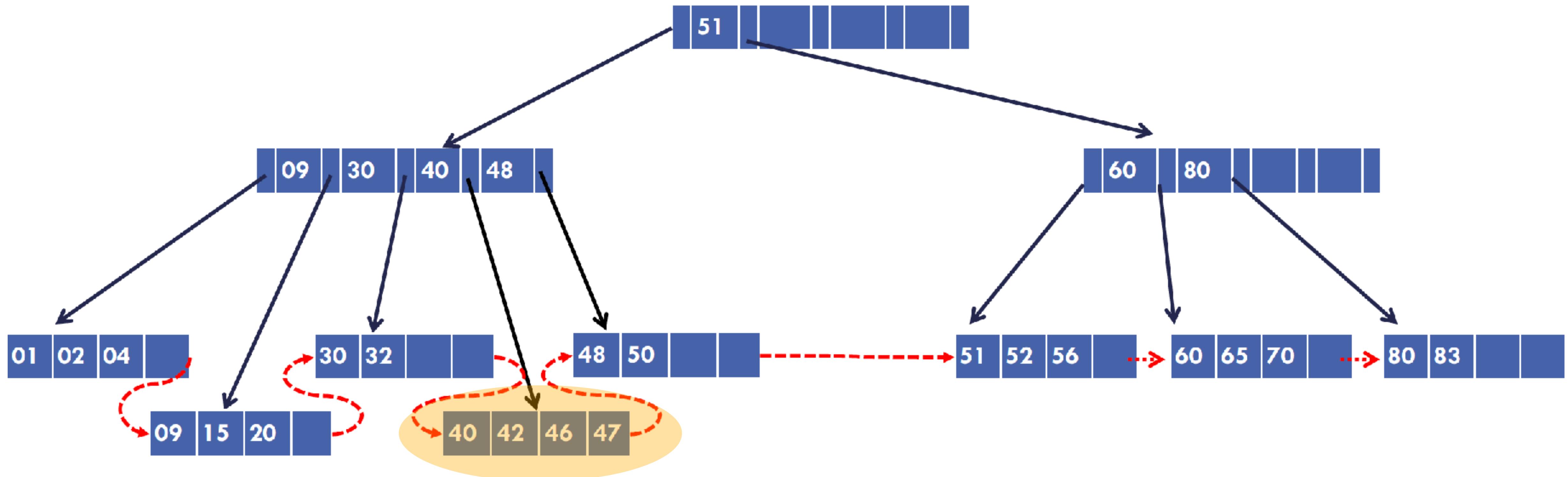
Árvore B+ Inserção particionamento de nó interno: inserir chave 44



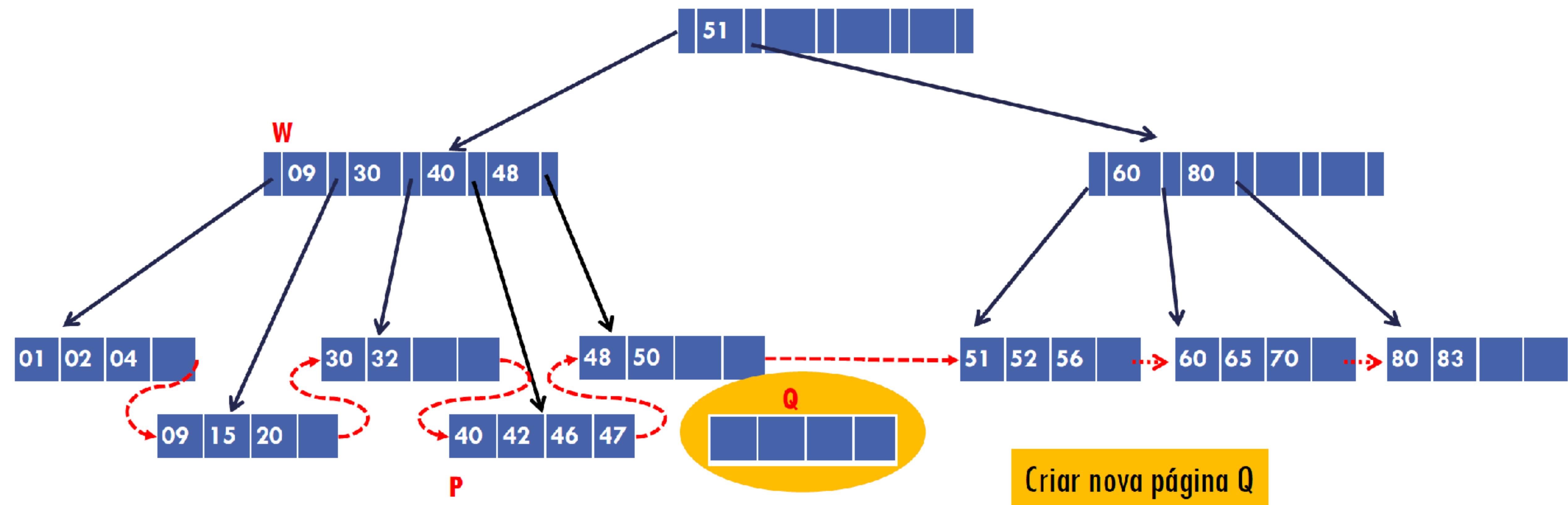


Inserir chave 44

Inserção faria página ficar com B+1 chaves
Solução: partitionar o nó

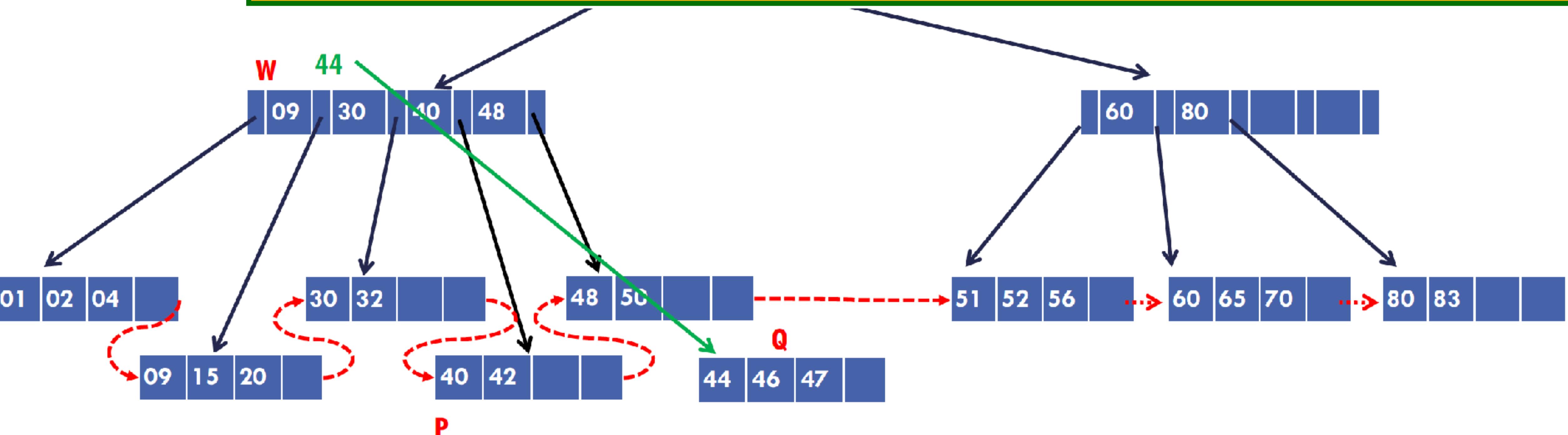


Árvore B+ Inserção particionamento de nó interno: inserir chave 44

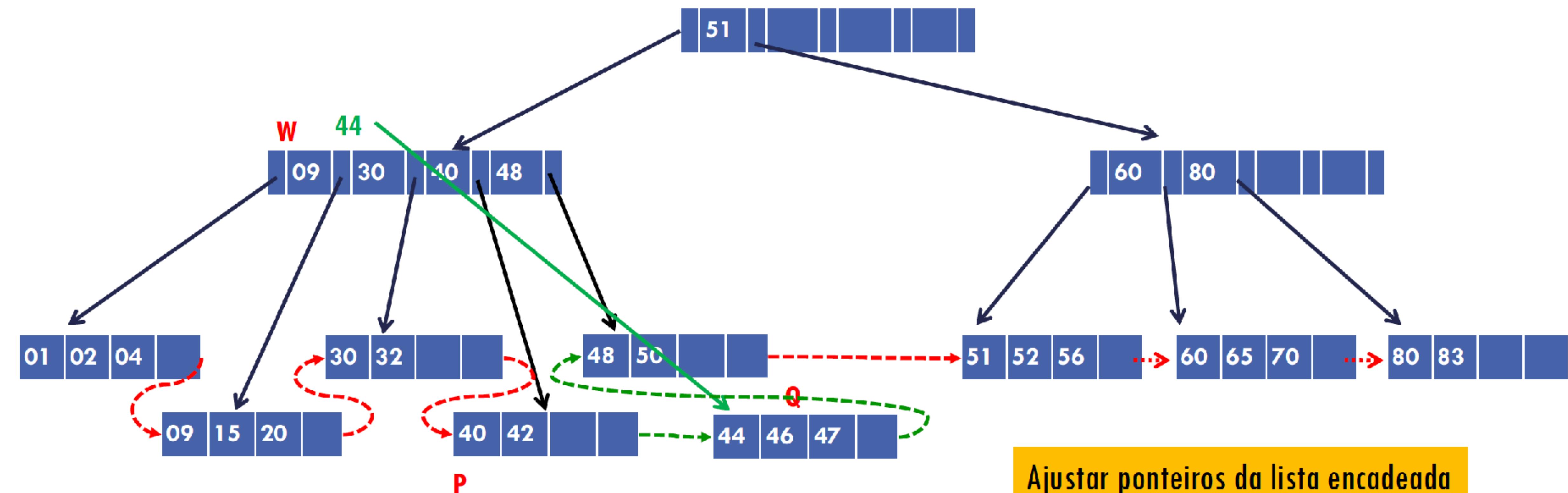




1. Dividir as chaves entre as duas páginas (30; 32; 40; 46; 47)
2. ($B/2 = 2$) chaves na página original P
3. chave da posição $(B/2)+1$ sobe para nó pai W (*mas registro é mantido na nova página*)
4. As $(B/2)+1$ chaves restantes na nova página Q



Árvore B+ Inserção particionamento de nó interno: inserir chave 44



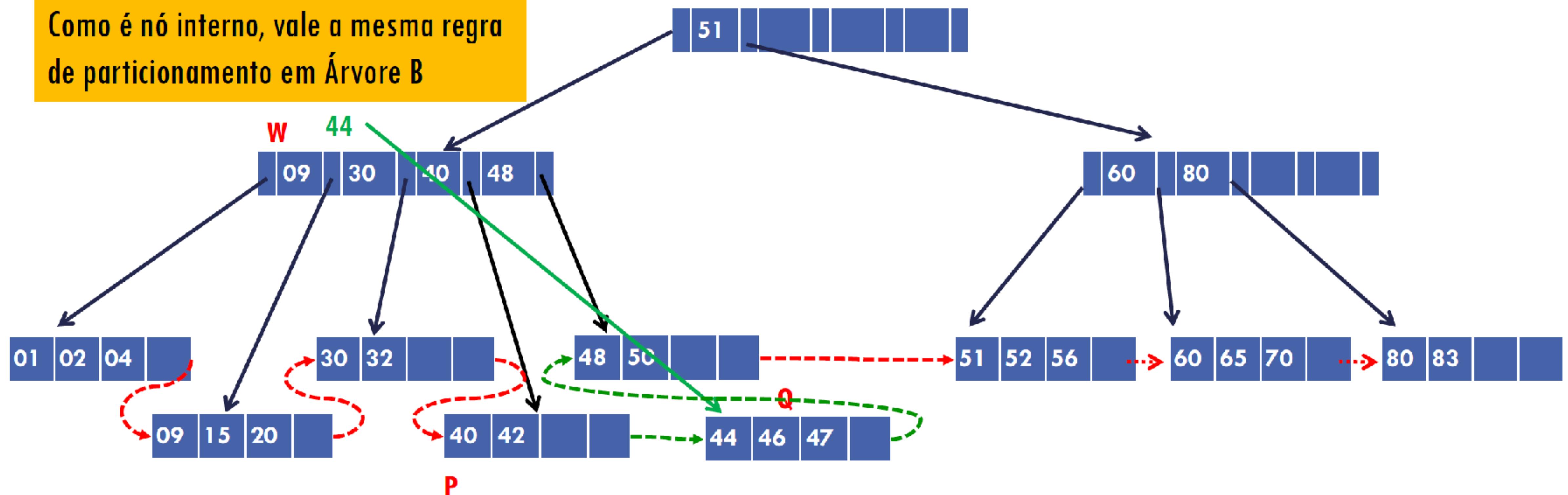
Árvore B+ Inserção particionamento de nó interno: inserir chave 44



Inserir 44 em W

Não há espaço: particionar

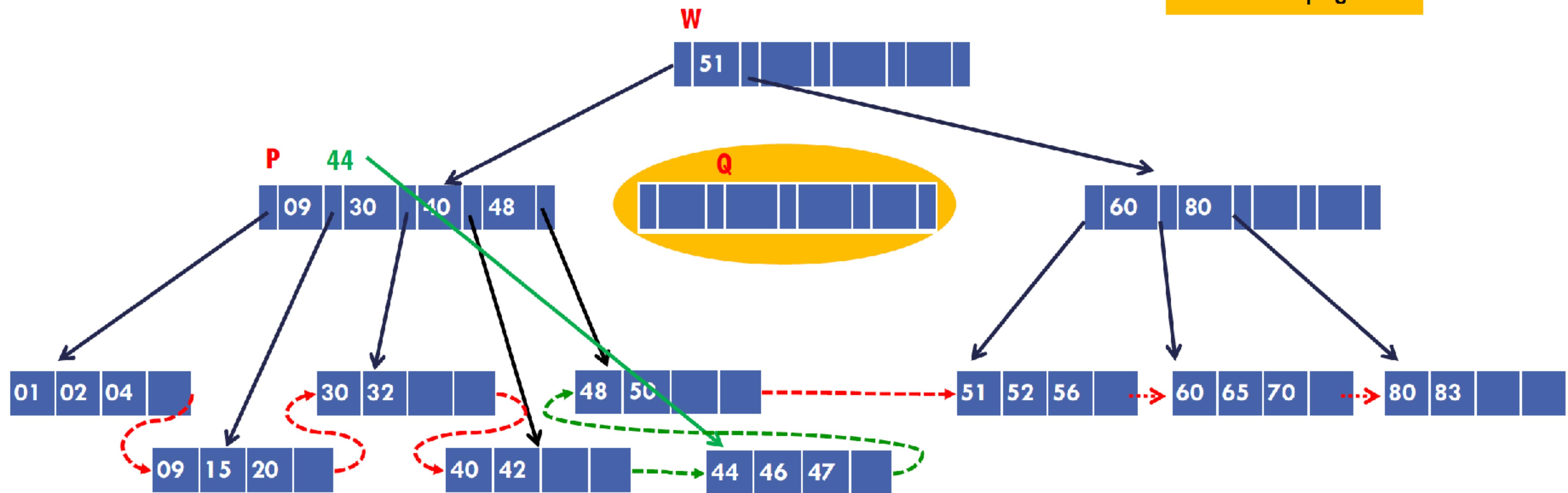
Como é nó interno, vale a mesma regra
de particionamento em Árvore B



Árvore B+ Inserção particionamento de nó interno: inserir chave 44



Criar nova página Q



Árvore B+ Inserção particionamento de nó interno: inserir chave 44

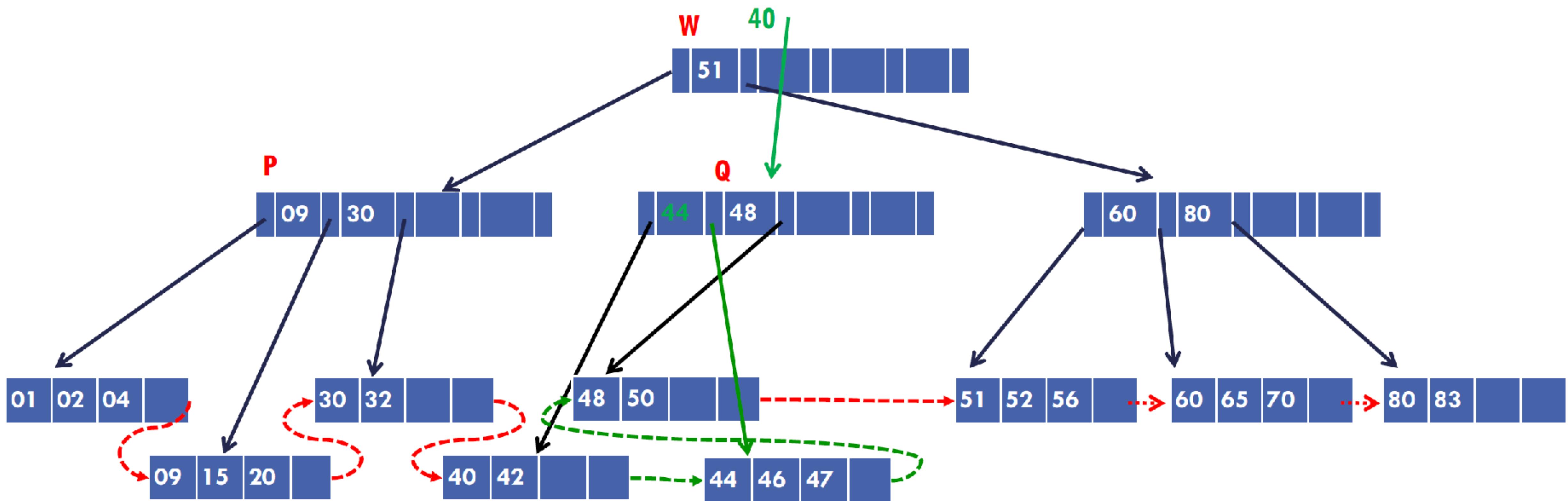


Dividir as chaves entre as duas páginas **(09; 30; 40; 44; 48)**

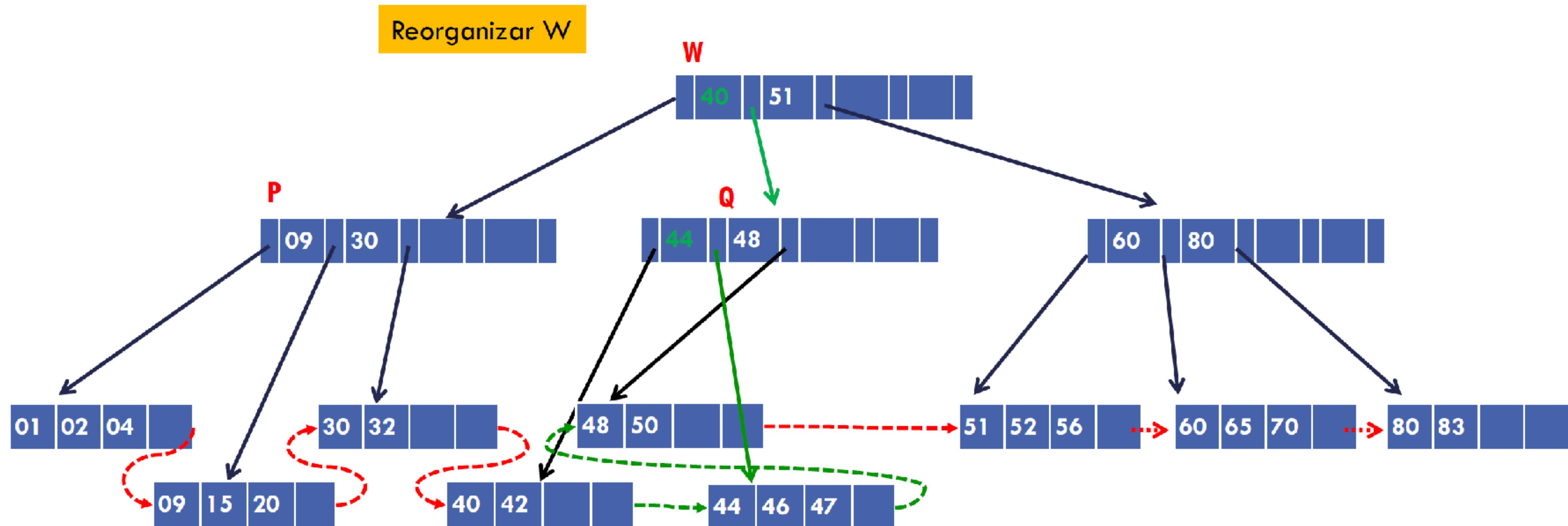
B/2 chaves na página original **P**

chave **(B/2)+1** sobe para nó pai **W**

chaves **(B/2)+1** em diante na nova página **Q**

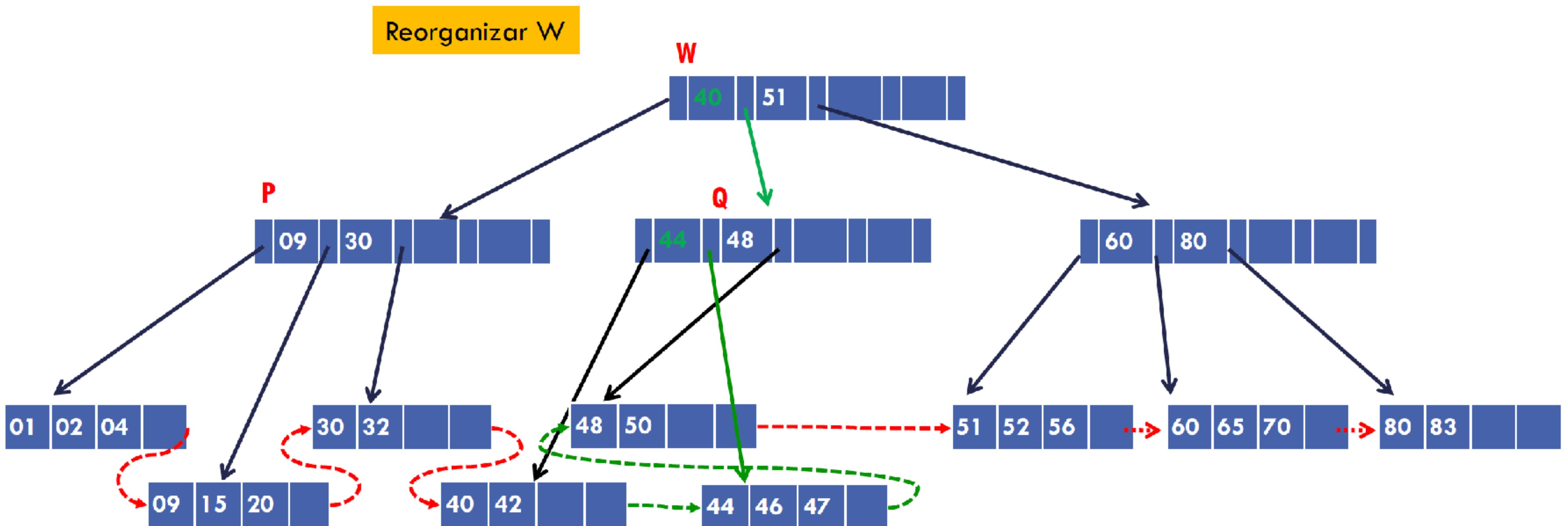


Árvore B+ Inserção particionamento de nó interno: inserir chave 44





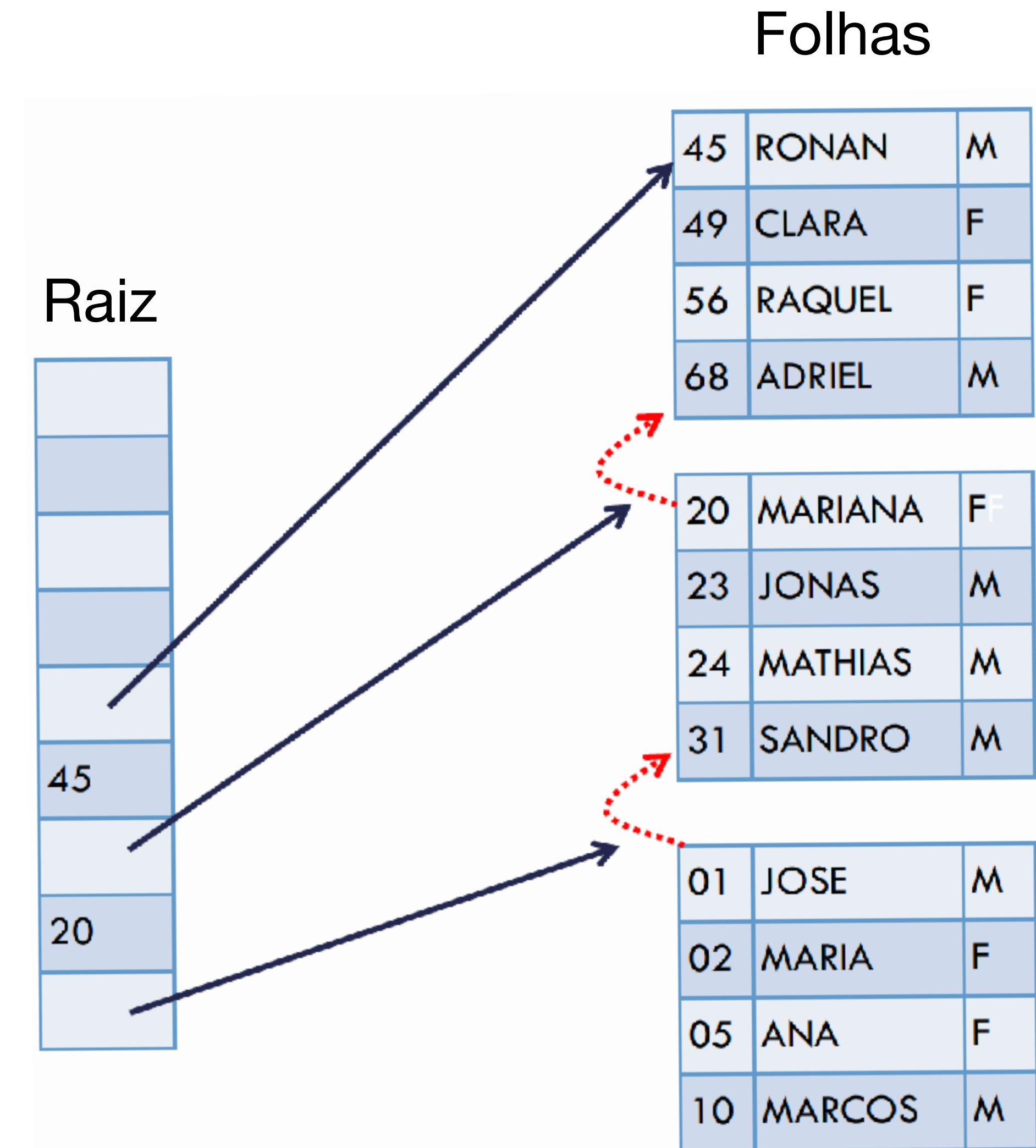
Inserir as chaves: 57 - 58 - 87 - 88 - 90 - 71 - 72



Exemplo: Mostrando dados nas folhas



- Neste exemplo, a árvore B+ tem o nó raiz e 3 folhas



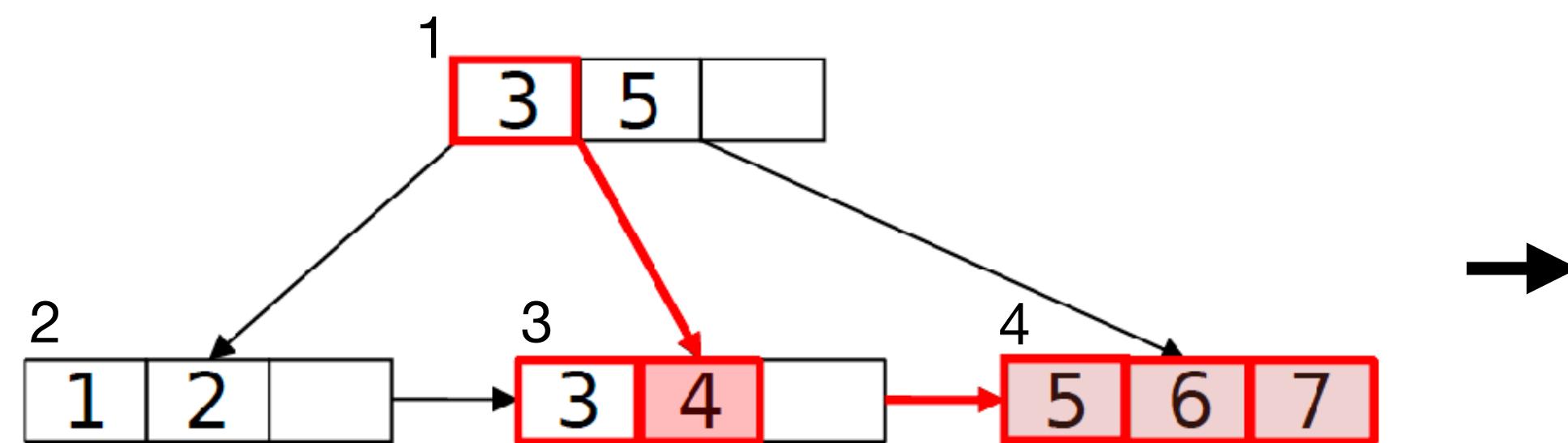
Inserção em Árvore B+



Estrutura básica de um nó/página de uma Árvore B+

```
struct node {  
    bool RNN; // RNN do próprio nó no arquivo de índice (árvore B+)  
    bool isLeaf; // Verifica se o nó/página é folha  
    int keys[order]; // vetor com valores das chaves (order-1)==bucketSize*  
    int dataRRN[order]; // RRNs de dados associados a cada chave (somente em folha)  
    int filhos[order + 1]; // Ponteiros** (RNNs dos nós no arquivo) para os filhos  
    int numKeys; // número de chaves inseridas na página  
    int parent; // referência para o pai  
    int next_node; // referência para o próximo nó/página da lista (se o nó é folha)  
};
```

Árvore B+



Arquivo armazenando a árvore B+, com cada linha representando um registro

RNN	isLeaf	keys	dataRRN	filhos	numKeys	parent	next_node
1	False	3, 5, -1	-1, -1, -1	2, 3, 4, -1	2	-1	-1
2	True	1, 2, -1	101, 102, -1	-1, -1, -1, -1	2	1	3
3	True	3, 4, -1	103, 104, -1	-1, -1, -1, -1	2	1	4
4	True	5, 6, 7	105, 106, 107	-1, -1, -1, -1	3	1	-1

*Estamos assumindo nesta implementação que os nós folhas terão o mesmo número de chaves que os nós internos ($order-1==bucketSize$)

**Como estamos tratando de arquivos, os ponteiros são os RNNs. Ou seja, o RNN de um registro que guarda o nó/página no arquivo

Inserção em Árvore B+



```
# Função para inserir no nó folha
# leaf: nó folha; key: chave a ser inserida; RNN_data: RNN da chave no arq. dados
insert_at_leaf(leaf, key, RNN_data)
    if a folha não esta vazia
        # Loop para encontrar a posição de inserção correta da chave
        temp1 = leaf.keys
        for i=0 ate size(temp1)-1
            # Inserção, quando o valor já existe no nó
            if (key == temp1[i])
                #chave ja existe
                break
            # Inserção, quando o valor é menor que os valores existentes
            elif (key < temp1[i])
                #insere key na posição i e dá um shift nas demais chaves
                break
            # Inserção, quando o valor é maior que todos os valores presentes
            elif (i + 1 == size(temp1))
                #insere key depois da posição i
                break
    else:
        # Se o nó estiver vazio, adicione o valor e a chave diretamente na primeira posição
```

Inserção em Árvore B+



```
# Função de inserção principal
#key: chave inserida; RNN_data: RNN da chave no arq. dados
insert(key, RNN_data)
    old_node = search(key) # Encontre o nó folha adequado para inserção
    insert_at_leaf(old_node, key, RNN_data) # Insere a chave e RNN_data no nó folha

# Se o nó folha estiver cheio após a inserção, é necessário dividi-lo (split)
if (old_node.numKeys == order)
    node1 = Node() # cria um novo nó
    node1.isLeaf = True #seta como folha
    node1.parent = old_node.parent #seta o nó pai
    mid = int(ceil(old_node.numKeys / 2)) - 1 # Determinando a posição média
    node1.keys = recebe as chaves de mid+1 ate old_node.numKeys
    node1.dataRRN = recebe os RNNs de mid+1 ate old_node.numKeys
    # Novo nó aponta para quem old_node apontava na lista
    node1.nextKey = old_node.nextKey
    old_node.keys = recebe as chaves de 0 a mid
    old_node.dataRRN = recebe os RNNs de 0 a mid
    old_node.nextKey = node1.RNN # Agora aponta para o RNN do novo nó na lista
    # Insira o valor médio no nó pai (que é interno)
    insert_in_parent(old_node, node1.keys[0], node1)
```

Inserção em Árvore B+



```
# Função para encontrar o nó folha adequado para inserção de uma chave
search(root, key) #root: raiz; key: chave a ser buscada
    # Inicializa o nó atual como a raiz da árvore
    current_node = root
    # Continua o loop até que o nó folha seja encontrado
    while(current_node.isLeaf == False)
        temp2 = current_node.keys # Armazena as chaves do nó atual em temp2
        for i =0 ate size(temp2)-1 # Percorre todas as chaves do nó atual
            if (key == temp2[i]) # Se a chave é igual à chave atual no nó
                # Muda para o filho à direita da chave e sai do loop
                current_node = current_node.filhos[i + 1]
                break
            elseif (key < temp2[i]) # Se a chave é menor que a chave atual no nó
                # Muda para o filho à esquerda da chave e sai do loop
                current_node = current_node.filhos[i]
                break
        # Se o loop chegou ao final e nenhuma das condições acima foi satisfeita
        elseif (i + 1 == size(current_node.filhos))
            # Muda para o último filho do nó atual e sai do loop
            current_node = current_node.filhos[i + 1]
            break

    return current_node
```

Inserção em Árvore B+



```
# Função para lidar com a inserção em nós internos (não-folha) após uma divisão
# old_node: nó que foi dividido e vira filho esquerdo; key_promovida: chave promovida
# new_node: nó criado devido a lotação de old_node e vira filho direito
insert_in_parent(old_node, key_promovida, new_node):
    # Verifica se o nó atual (old_node) é a raiz da árvore
    if (root == old_node.RNN):
        rootNode = Node() # Cria um novo nó para se tornar a nova raiz
        rootNode.keys[0] = key_promovida # A nova raiz terá a chave promovida como seu único valor
        # As chaves da nova raiz apontarão para o nó antigo (filho esquerdo) e o novo nó (direto)
        rootNode.filhos = [old_node.RNN, new_node.RNN]

        root = rootNode.RNN # Define o novo nó como a raiz da árvore

        # Define o nó pai do nó antigo e do novo nó como a nova raiz
        old_node.parent = rootNode.RNN
        new_node.parent = rootNode.RNN

    return # Como tratamos a raiz, não precisamos fazer mais nada nesta chamada
# Se chegamos aqui, significa que o nó não é a raiz. Então, pegamos o nó pai do nó atual
parentNode = old_node.parent
# Armazenamos os filhos do nó pai em temp3 para facilitar a manipulação
temp3 = parentNode.filhos

# Percorre cada filho no nó pai. Continua no proximo slide
```

old_node = P
new_node = Q
parentNode = W

Inserção em Árvore B+



```
for i=1 ate size(temp3) # Percorre cada filho no nó pai
    if (temp3[i] == old_node.RNN) #Encontra a posição do nó antigo entre as chaves do nó pai
        # Insere a chave promovida na posição correta dos valores do nó pai
        parentNode.values = parentNode.keys[ate i] + [key_promovida] + parentNode.keys[i+1 ate fim]
        # Insere o RNN do novo nó na posição correta nos filhos do nó pai
        parentNode.filhos = parentNode.filhos[ate i] + [new_node.RNN] + parentNode.filhos[i+1 ate fim]

        # Verifica se o nó pai excede a capacidade máxima após a inserção
        if (parentNode.numKeys == order)
            parentdash = Node() # Se exceder, cria um novo nó interno
            parentdash.parent = parentNode.parent # Define o pai do novo nó interno
            # continua...
```

Inserção em Árvore B+



```
# Move os valores e chaves à direita da posição média (mid = int(ceil(order / 2))
- 1) para o novo nó interno
parentdash.keys = parentNode.keys[mid + 1 ate fim]
parentdash.filhos = parentNode.filhos[mid + 1 ate fim]
# Pega o valor da posição média para ser promovido na próxima chamada
key_promovida = parentNode.keys[mid]

# Atualiza os valores e chaves do nó pai para refletir a divisão
if (mid == 0):
    parentNode.keys = parentNode.keys[mid]
else:
    parentNode.keys = parentNode.keys[0 ate mid]
parentNode.filhos = parentNode.filhos[0 ate (mid + 1)]
# Atualiza os pont de pai dos nós filhos de parentNode e parentdash após a divisão
for filho in parentNode.filhos:
    filho.parent = parentNode.RNN
for filho in parentdash.filhos:
    filho.parent = parentdash.RNN

insert_in_parent(parentNode, key_promovida, parentdash)# Recursivamente insere no
```

avô

Inserção em Árvore B+

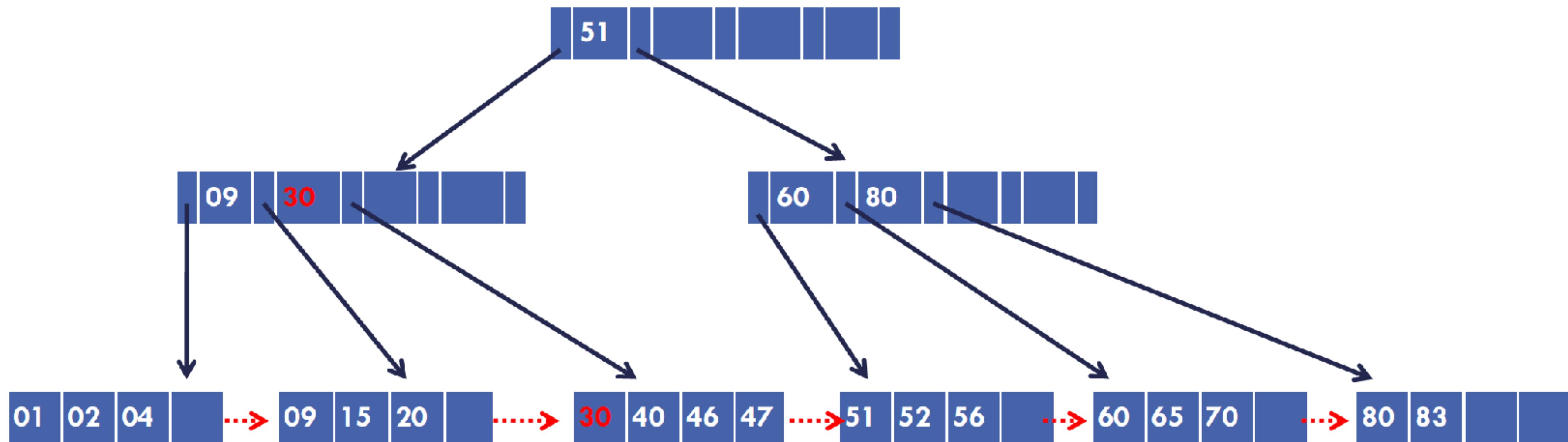


```
# Inicializando e testando
order = 3
insert(5, 33)
insert(15, 21)
insert(25, 31)
insert(35, 41)
insert(45, 10)
```



- Excluir apenas no nó folha
- Chaves excluídas continuam nos nós intermediários
- Remoção de registros pode provocar underflow em um bloco
- Solução:
 - concatenar o bloco com o seu **antecessor** ou **sucessor** na sequência lógica
 - redistribuir os **registros**, movendo-os entre blocos logicamente adjacentes

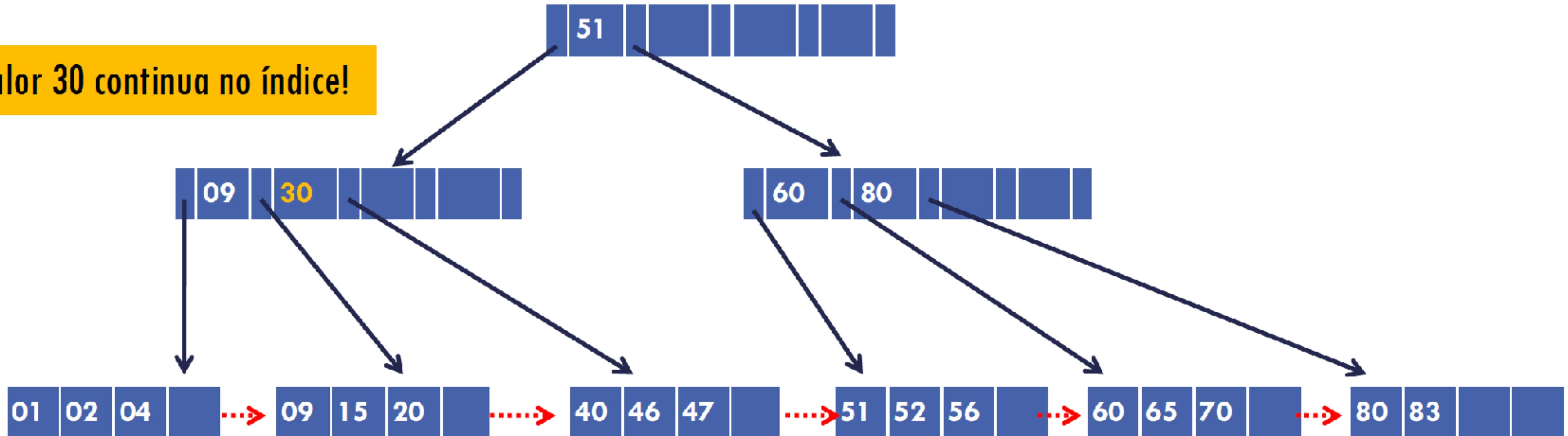
Árvore B+ Exclusão chave 30



Árvore B+ Exclusão chave 30



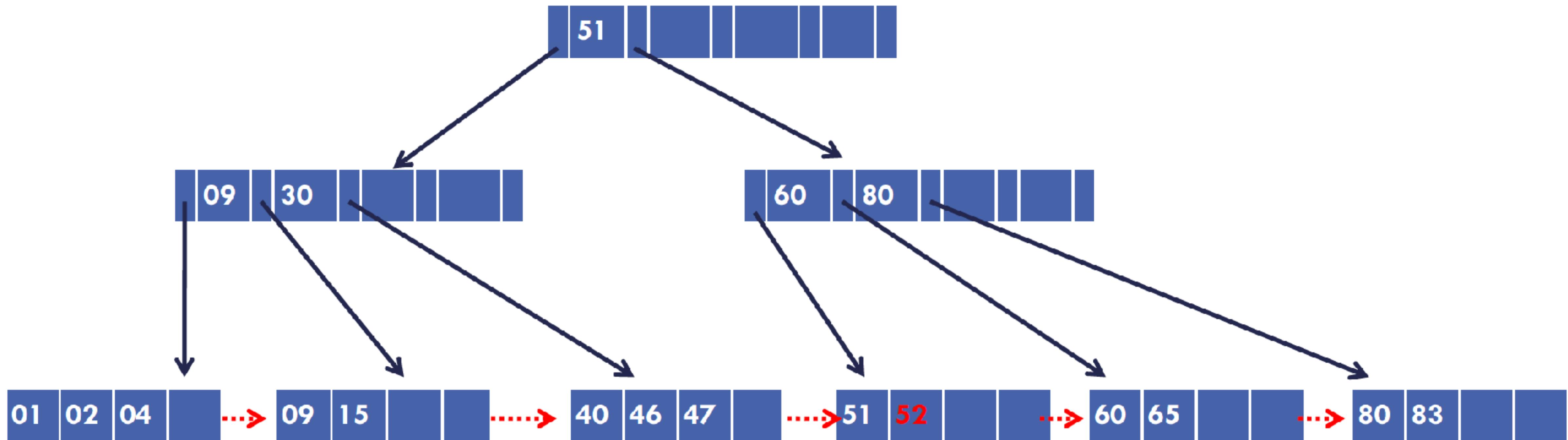
O valor 30 continua no índice!



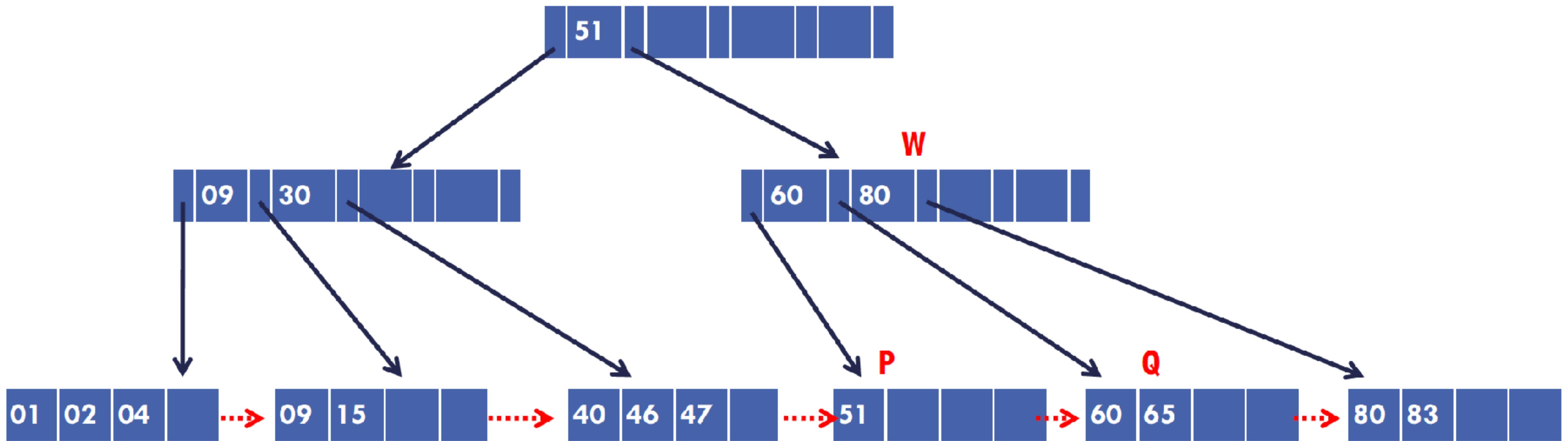


- Exclusões que causem concatenação de folhas podem se propagar para os nós internos da árvore
- Importante:
 - Se a concatenação ocorrer na folha: **a chave do nó pai não desce para o nó concatenado**, pois ele não carrega dados com ele. Ele é simplesmente apagado.
 - Se a concatenação ocorrer em nó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

Árvore B+ Excluir chave 52

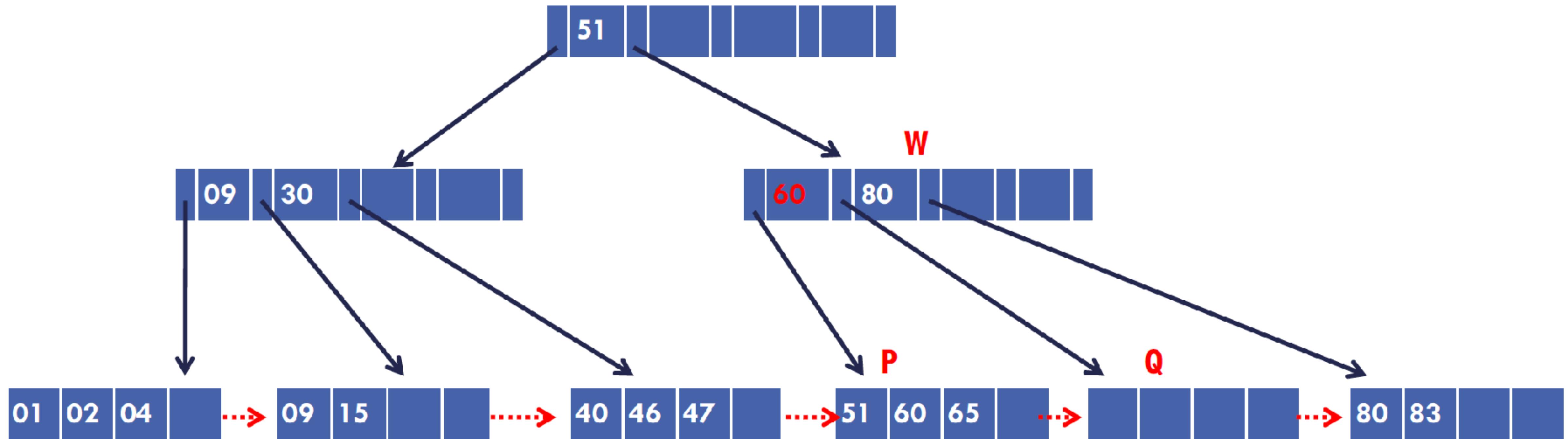


Árvore B+ Excluir chave 52 - Underflow



Nó **P** ficou com menos de $B/2$ chaves (**underflow**) – necessário tratar isso
Soma dos registros de **P** e **Q** < $(B+1)$
Usar concatenação

Árvore B+ Excluir chave 52 - Underflow

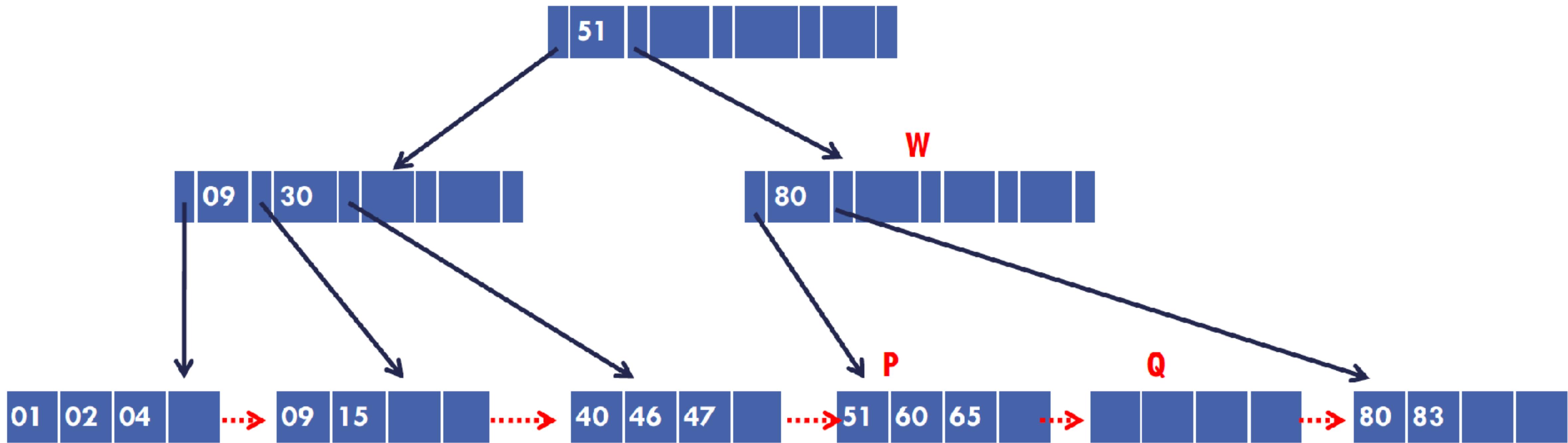


Concatenação:

Passar os registros de **Q** para **P**

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

Árvore B+ Excluir chave 52 - Underflow

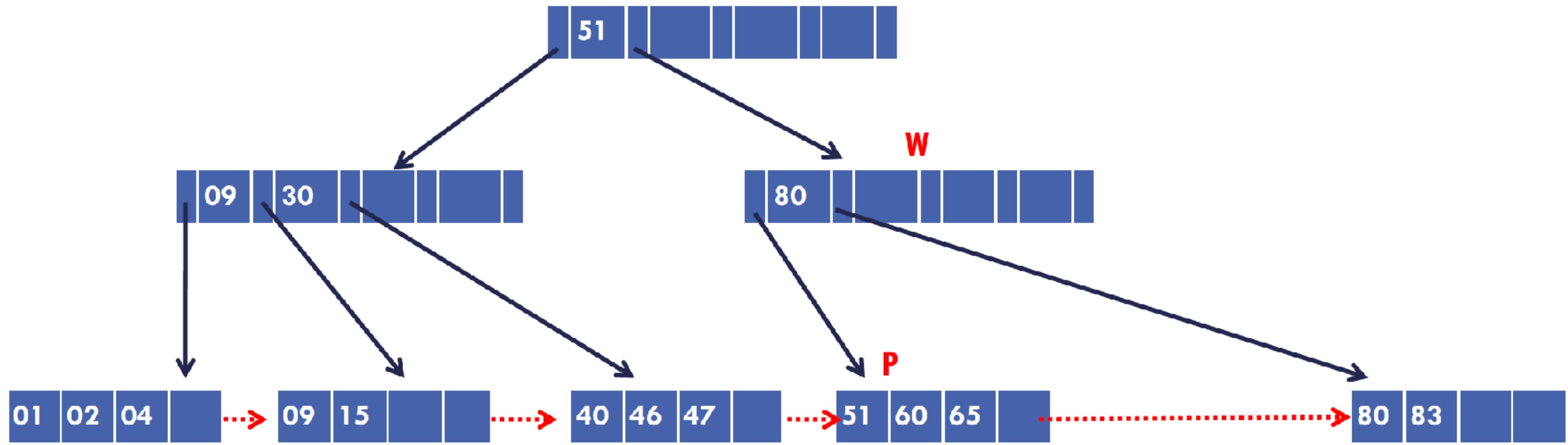


Concatenação:

Passar os registros de **Q** para **P**

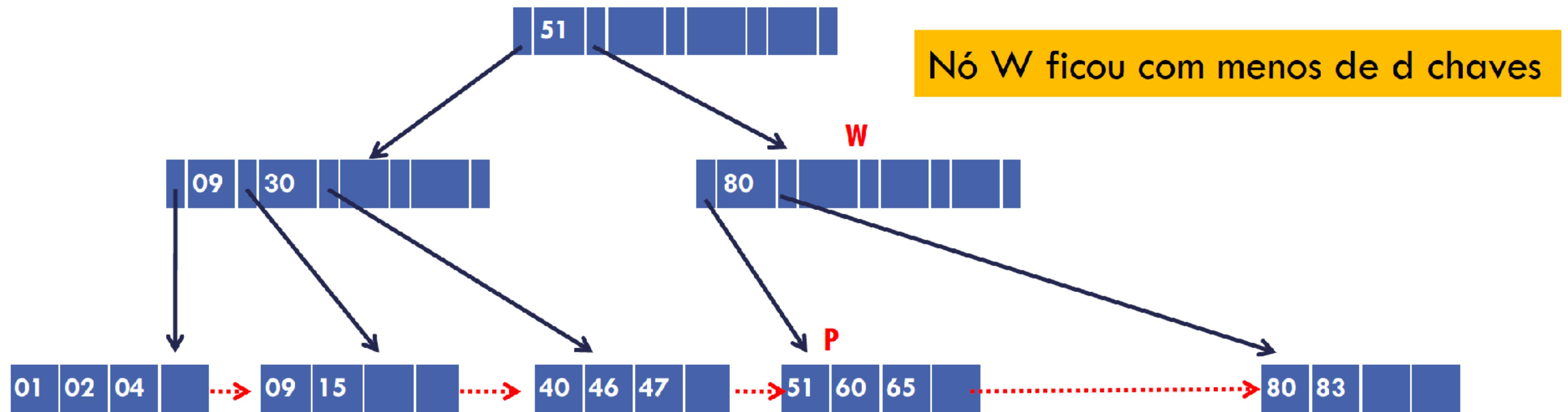
Eliminar a chave em **W** que divide os ponteiros para as páginas **P** e **Q**

Árvore B+ Excluir chave 52 - Underflow

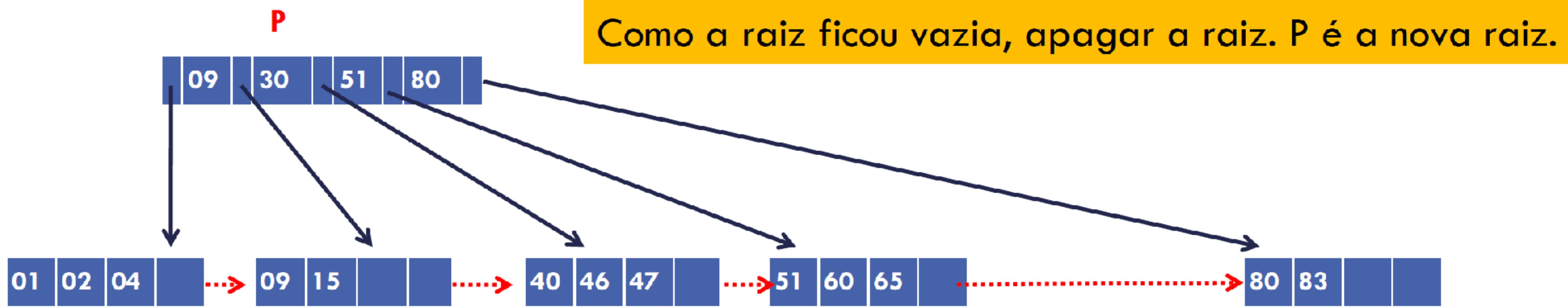


Eliminar nó Q

Árvore B+ Excluir chave 52 - Underflow



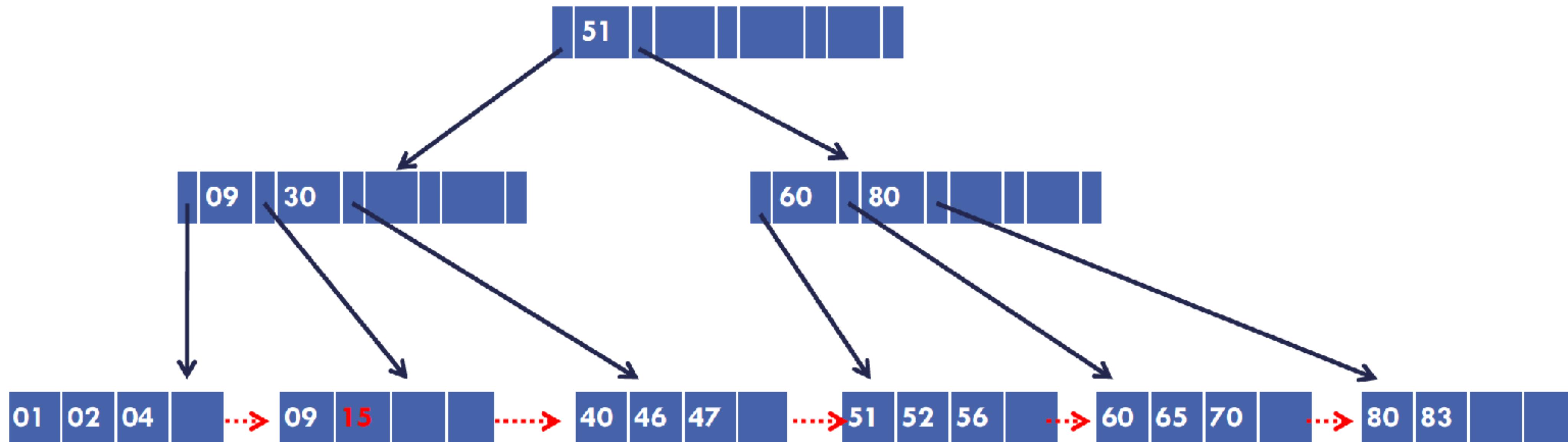
Árvore B+ Excluir chave 52 - Underflow



Árvore B+ Exclusão que causa redistribuição



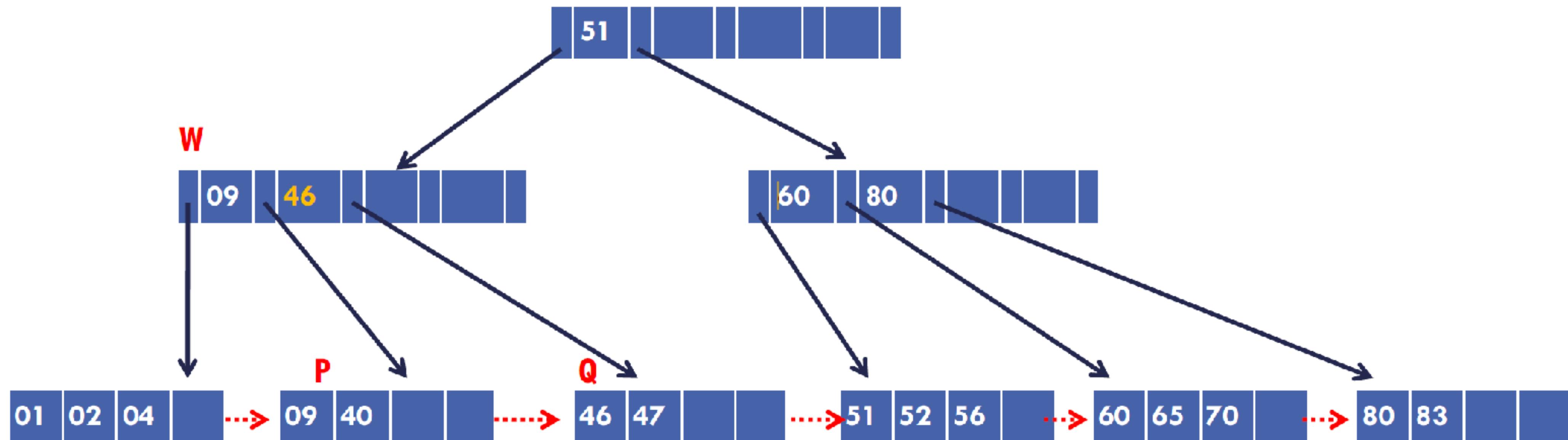
- Exclusões que causem redistribuição dos registros nas folhas provocam mudanças no conteúdo do índice, mas não na estrutura (não se propagam)
 - Semelhante a árvore B



Árvore 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)
 - Semelhante a árvore B

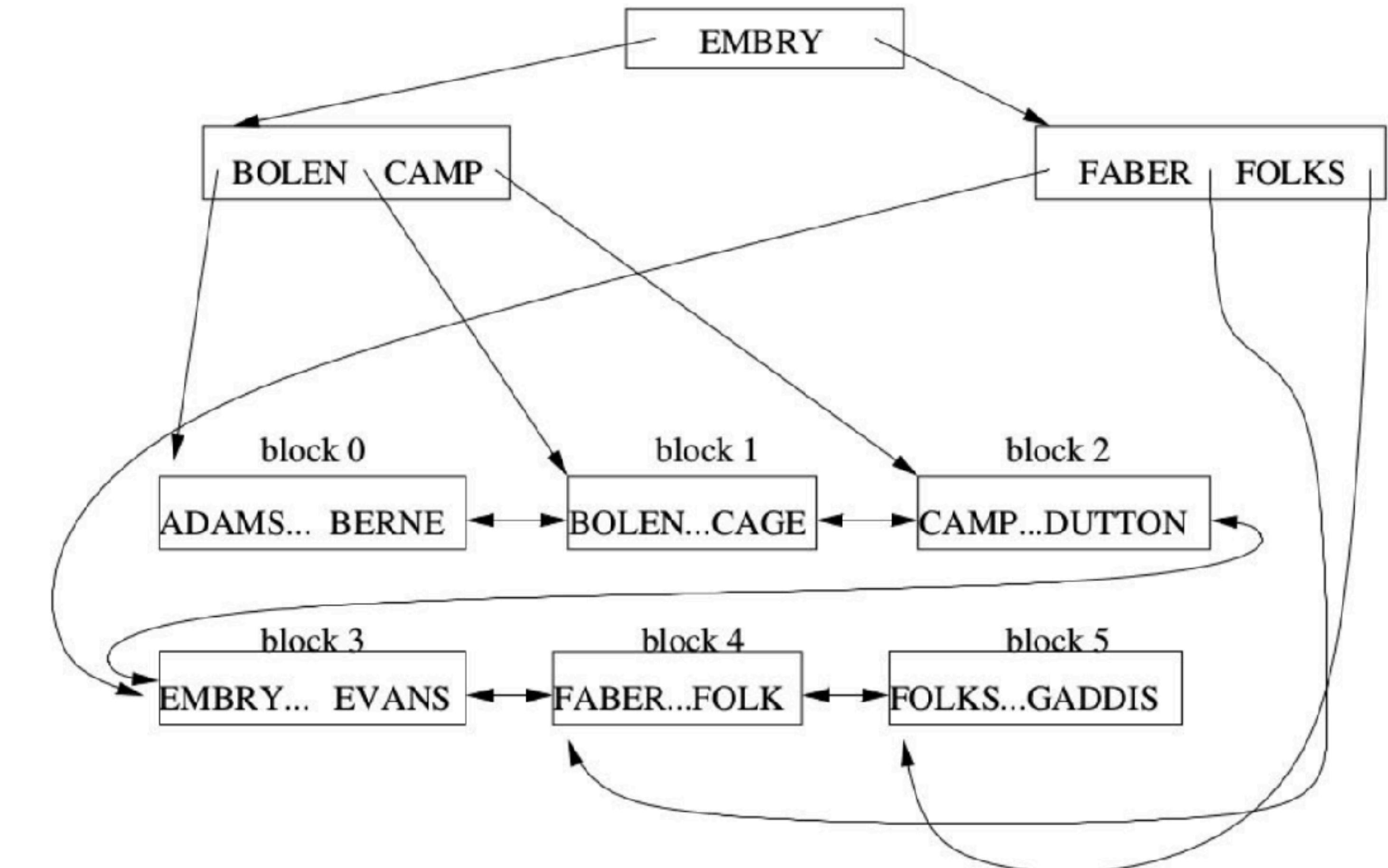


Árvore B+ com prefixo simples

Árvore B+ com prefixo simples



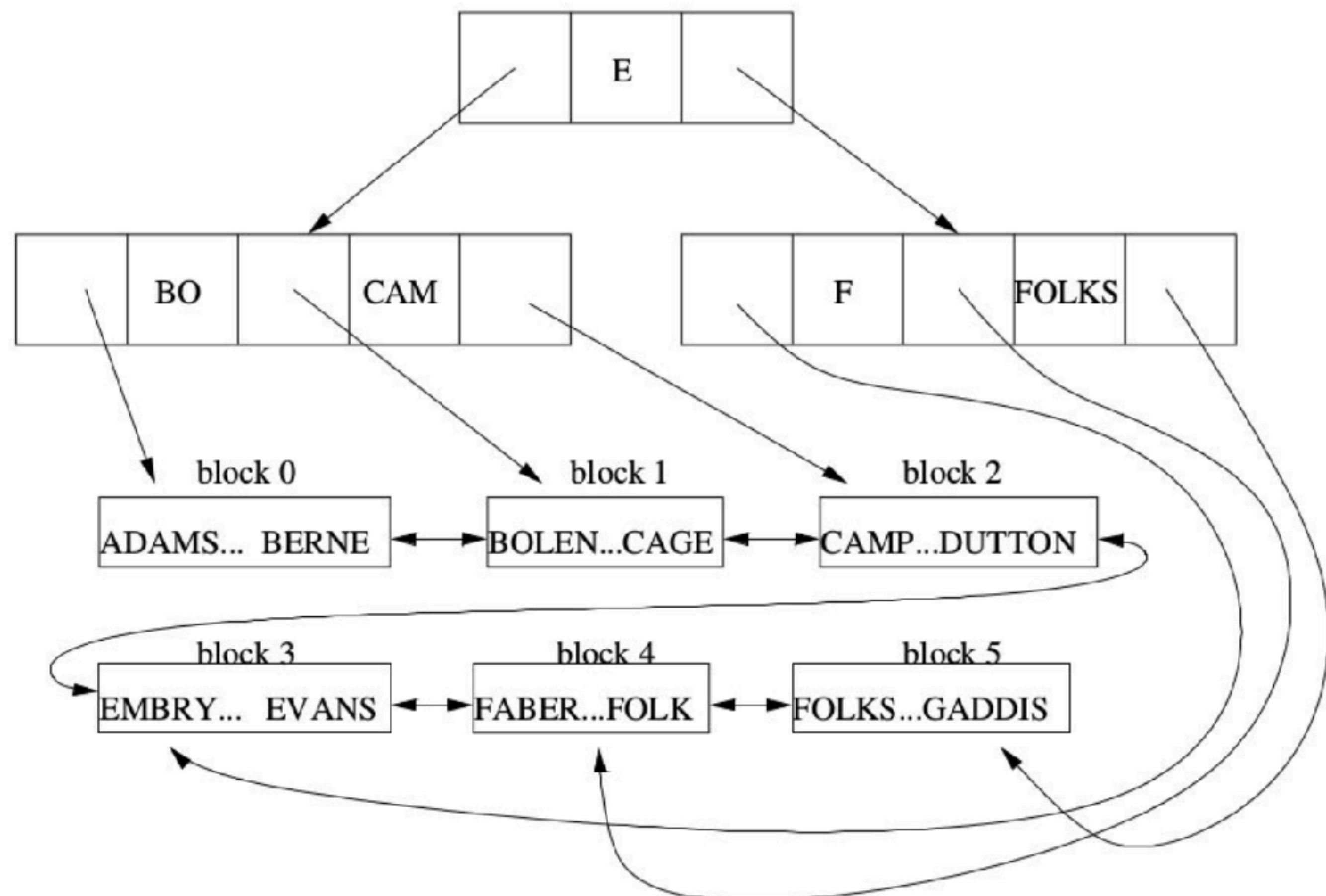
- Considere a seguinte árvore B+ que guarda em disco registros cujas chaves são nomes
- Dado que vários nomes possuem um prefixo em comum (e.g., **FOLK** e **FOLKS**), podemos guardar a árvore de modo mais eficaz visando reduzir o espaço que ela ocupa



Árvore B+ com prefixo simples

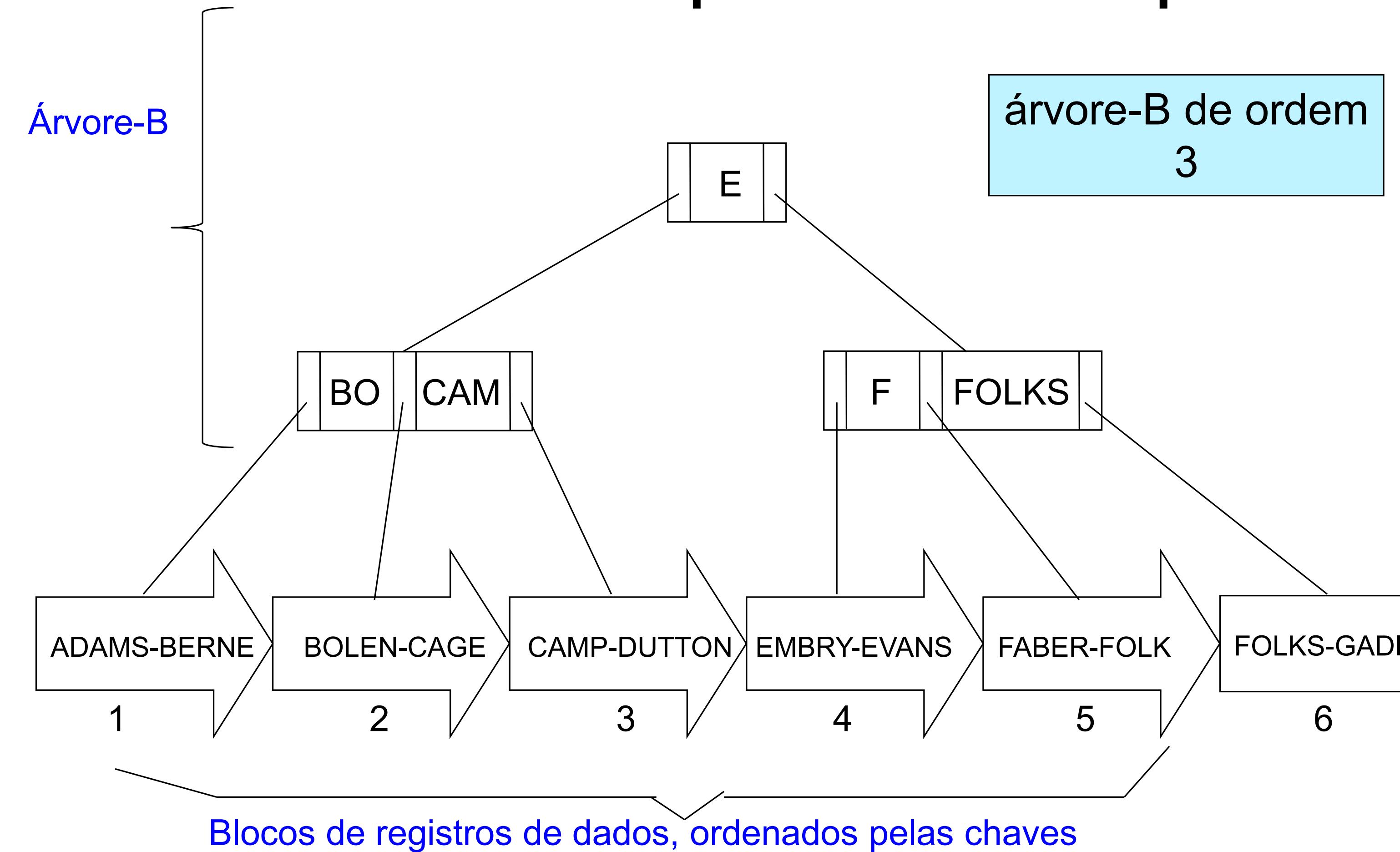


- O conceito de **separador** do tipo prefixo simples permite aumentar o número de separadores (ponteiros) por nó da árvore, reduzindo sua altura
- O **prefixo simples** que separa um bloco do anterior é a cadeia de caracteres mais curta que diferencia a última chave do bloco anterior da primeira do bloco atual
- Prefixos de chaves (de tamanho variável) compõem as páginas não folhas





Árvore-B+ de prefixos simples

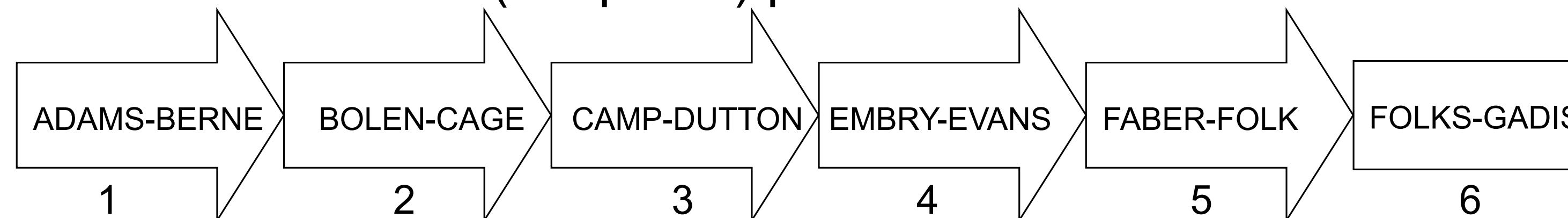




Índice Simples (Tabela)

- Se todos os índices couberem na RAM, uma tabela poderia substituir a Árvore-B:

- Busca binária (adaptada) para encontrar a chave



chave	bloco
BERNE	1
CAGE	2
DUTTON	3
EVANS	4
FOLK	5
GADIS	6

Índice de 1 nível
• registros de tamanho fixo
• contém a chave do último registro no bloco

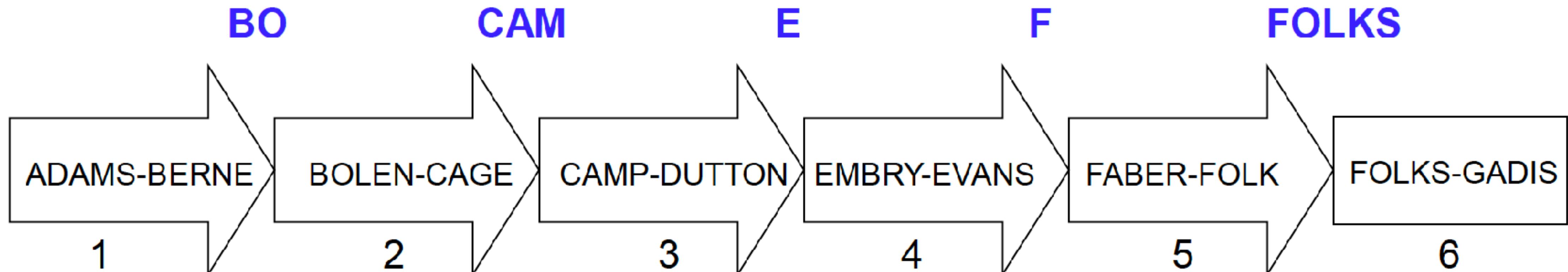
Árvore B+ com prefixo - Separadores



● Características

- são mantidos no índice, ao invés das chaves de busca
- possuem tamanho variável

● Exemplo



Árvore B+ com prefixo - Separadores



● Desafio

- escolher o menor separador para utilizar no índice

- Tabela de decisão

chave de busca x separador	decisão
chave < separador	procure à esquerda
chave = separador	procure à direita
chave > separador	procure à direita

Árvore B+ com prefixo - Separadores



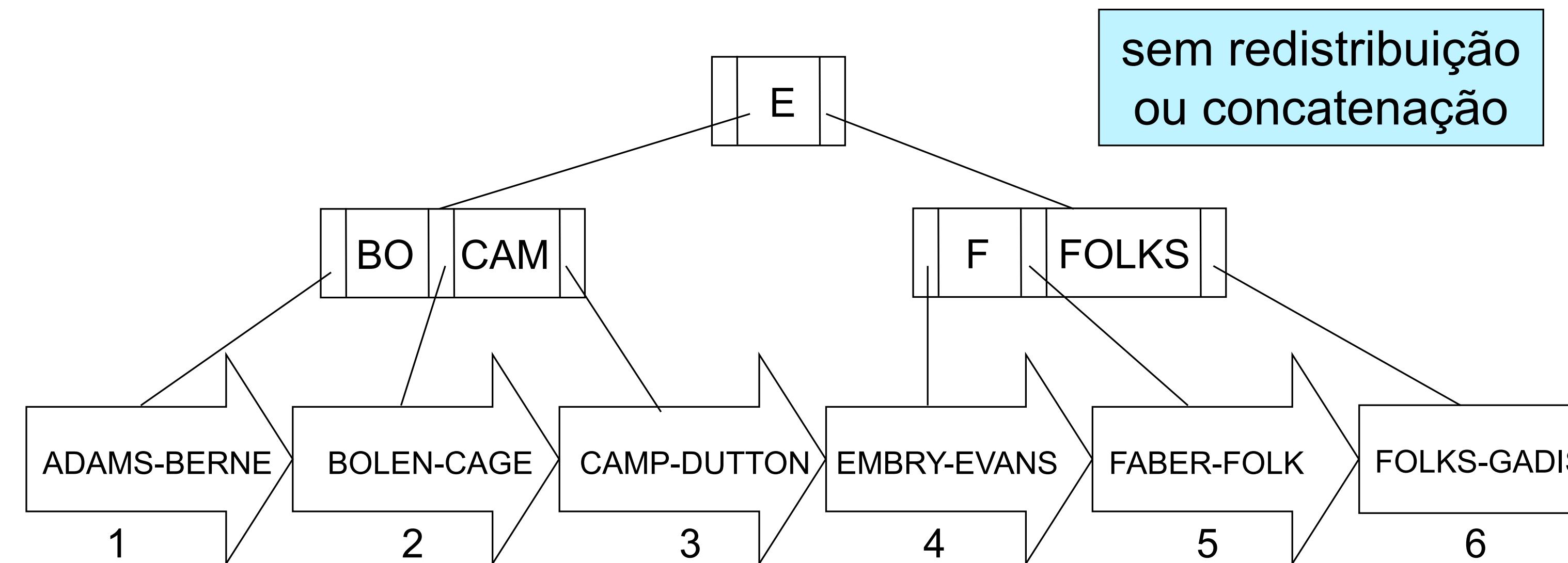
- Programa para gerar separadores mínimos

```
última chave  
de um bloco      1ª. chave do  
                próximo bloco      prefixo separador  
                                gerado  
  
find_sep(char key1[], char key2[], char sep[]) {  
    while ( (*sep++ = *key2++) == *key1++);  
    *sep='\\0';  
}
```

- Pode acontecer de o separador mínimo ser uma chave inteira

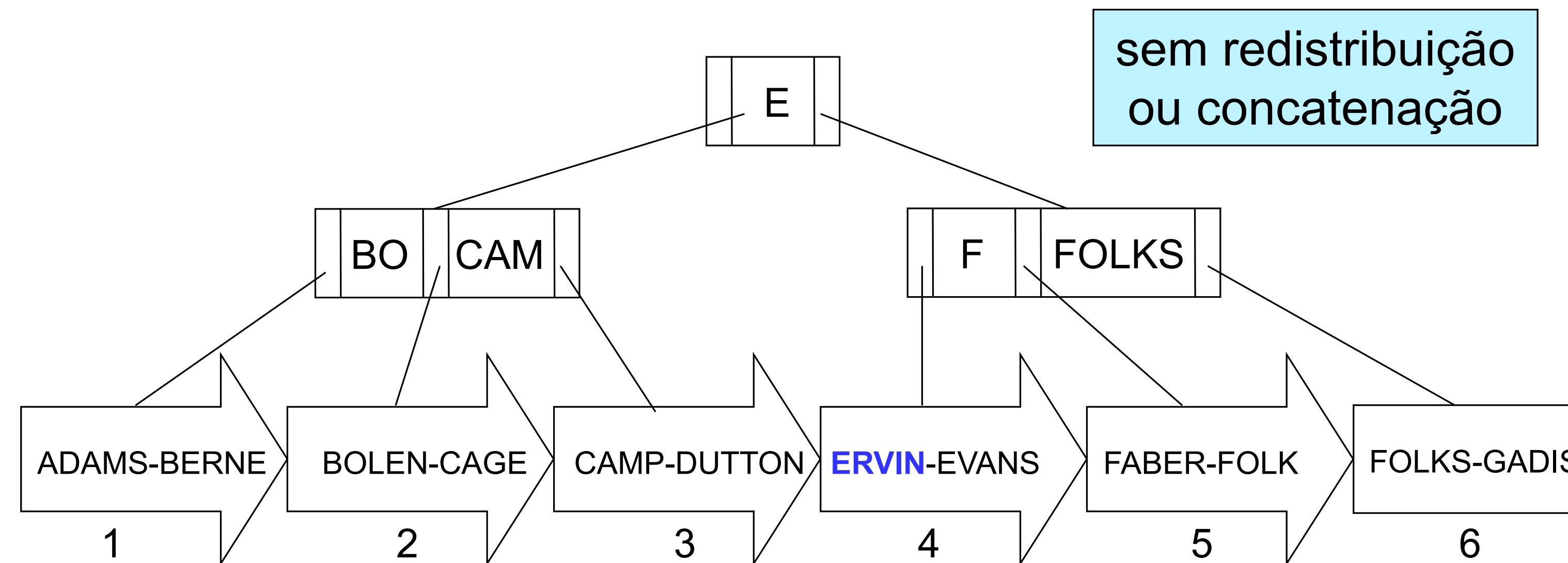


Remoção de EMBRY





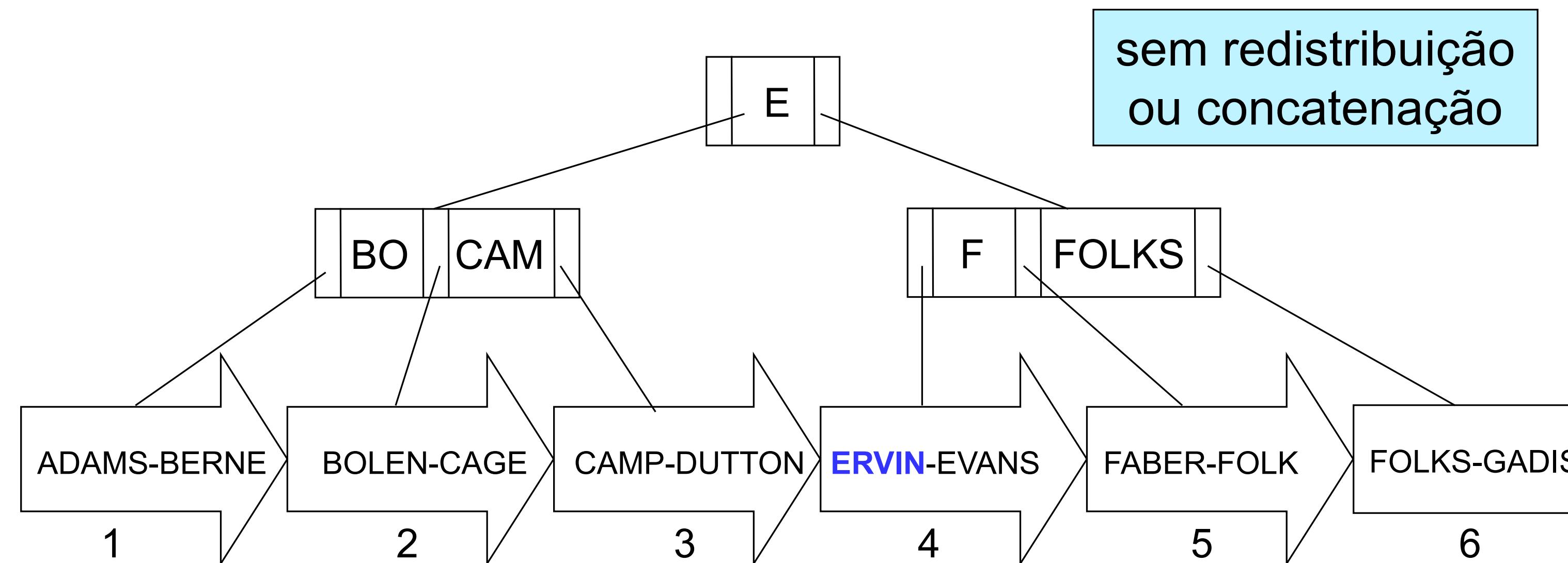
Remoção de EMBRY



- Efeito no *sequence set*
 - limitado a alterações no bloco 4



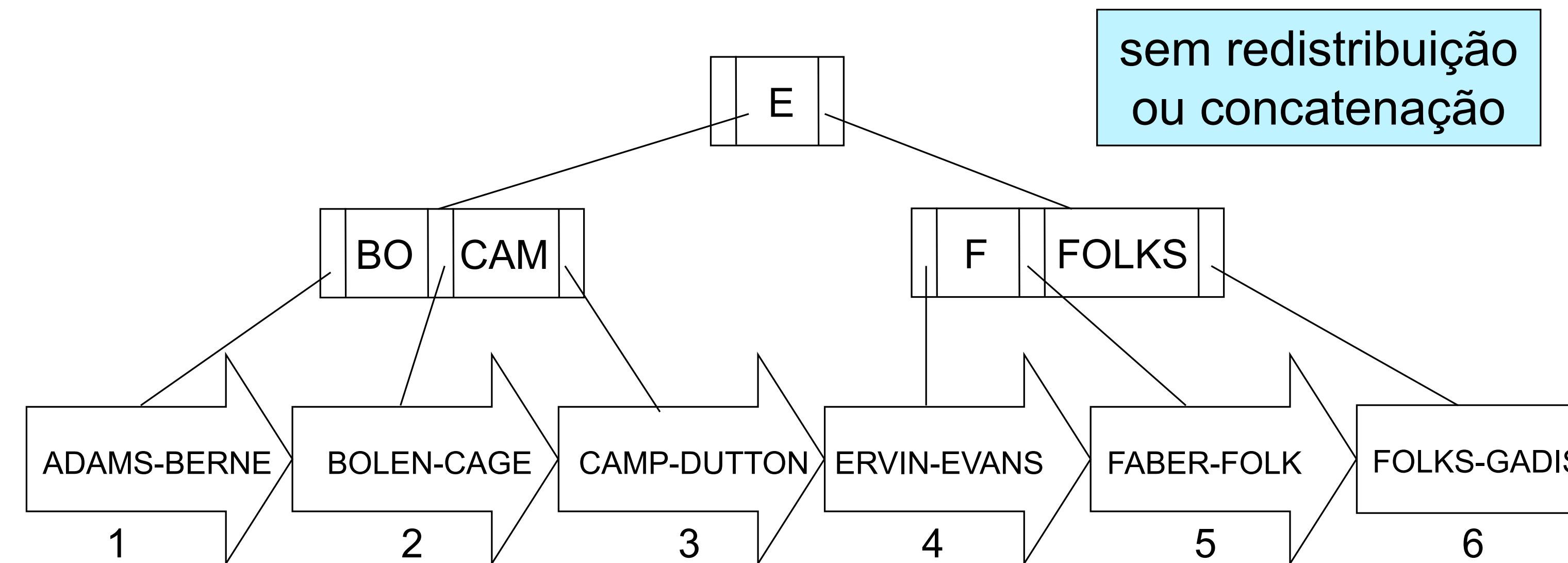
Remoção de EMBRY



- Efeito na árvore-B+
 - nenhum: ‘E’ é uma boa chave separadora

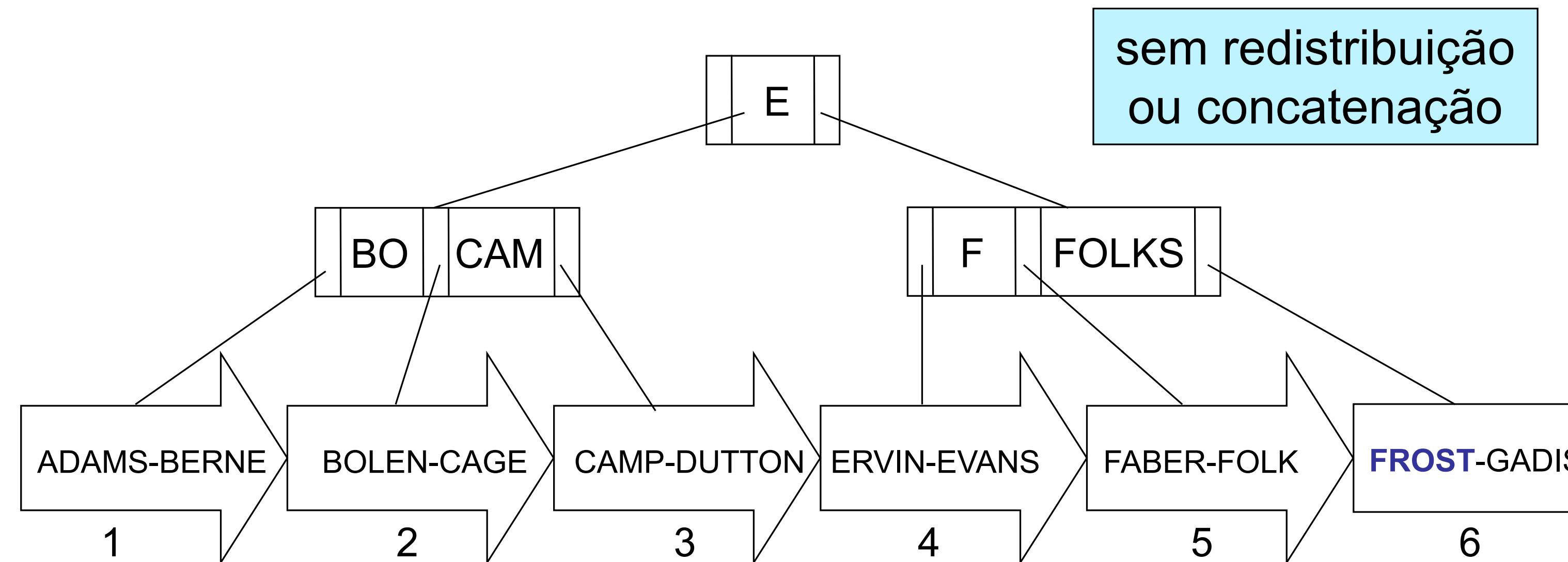


Remoção de FOLKS





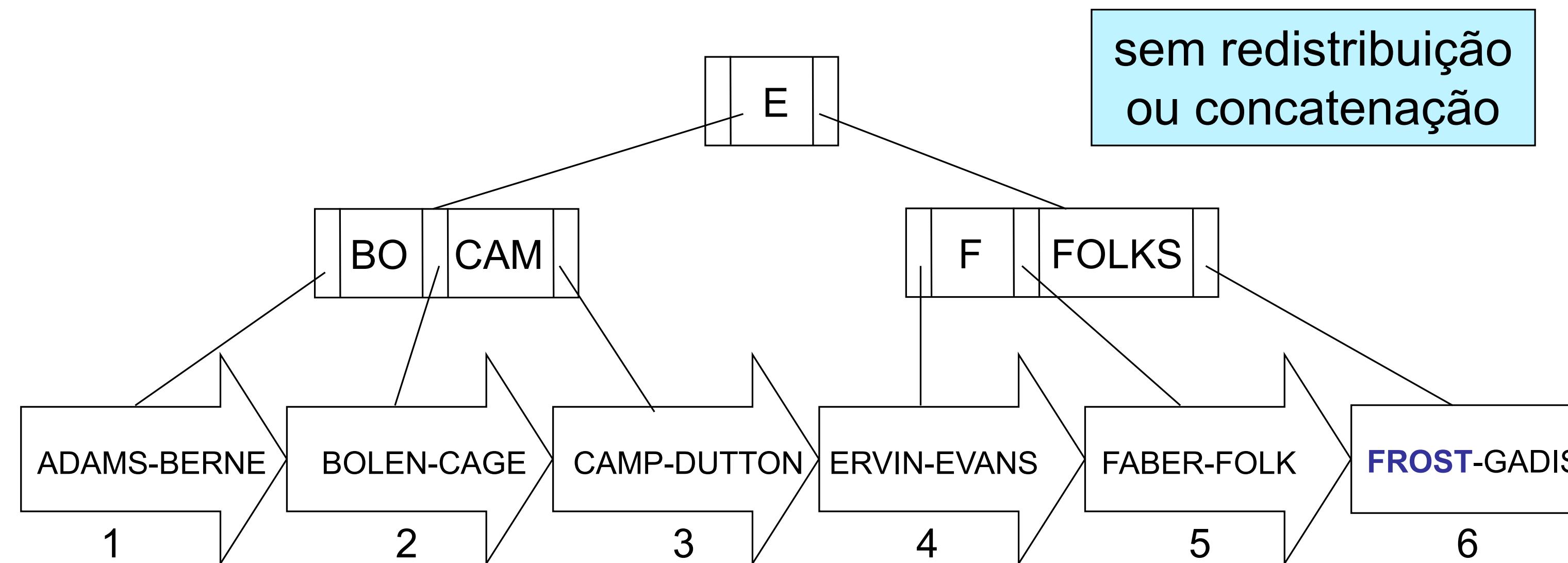
Remoção de FOLKS



- Efeito no *sequence set*
 - limitado a alterações no bloco 6



Remoção de FOLKS



- Efeito na árvore-B+
- nenhum: custos elevados se fosse arrumar



Inserção e Remoção

- Primeiro passo: Sequence Set
 - inserir ou remover o dado
 - tratar, caso necessário
 - *split*
 - concatenação
 - redistribuição

alterações são sempre realizadas a partir do arquivo de dados



Inserção e Remoção

- Segundo passo: Árvore-B⁺
 - se *split* no sequence set
inserir um novo separador no índice
 - se *concatenação* no sequence set
remover um separador do índice
 - se *distribuição* no sequence set
alterar o valor do separador no índice



- Pode-se utilizar três arquivos:
- Um arquivo para armazenar os metadados
 - Ponteiro para a raiz da árvore
 - Flag indicando se a raiz é folha
- Um arquivo para armazenar o índice (nós internos da árvore)
- Um arquivo para armazenar os dados (folhas da árvore)

Árvores B+ com prefixo simples: Estrutura dos nós



- Cada nó da árvore bem como os elementos do conjunto de sequências são blocos de tamanho fixo em disco
- Para facilitar a indexação em disco, os blocos devem ter o mesmo tamanho em ambas estruturas, porém os separadores são de tamanho variável
- Em vez de um vetor de chaves/separadores, usa-se uma única cadeira de caracteres para armazenar os separadores e um vetor de índices inteiros com a posição relativa dos separadores
- O número atual de separadores e o vetor de endereços dos filhos também são armazenados
- A busca (binária) é feita através do vetor de índices e como a única limitação é o tamanho do bloco, a árvore B tem ordem variável por nó

numero	numero	separadores	caracteres	separadores	indices dos separadores	ponteiros para filhos
11	28	ASBABROCCHCRADELEEDIERRFAFL		0 2 4 7 8 10 13 17 20 23 25	00 02 04 07 08 10 13 17 20 23 25	B00 B01 B02 ... B11



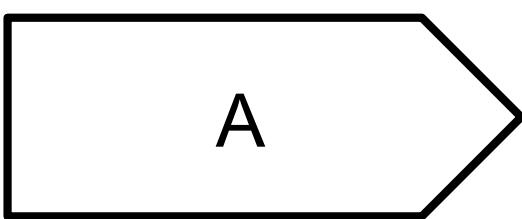
Exercício 1

- Criar uma árvore-B+ pela inserção das chaves A, B, C, D, E, F, G, H, I e J, nesta ordem
 - Ordem da árvore: 3
 - Blocos
 - mínimo de 2 registros
 - máximo de 3 registros



Solução (1/10)

- Inserindo A
 - Cria-se bloco e aloca-se A

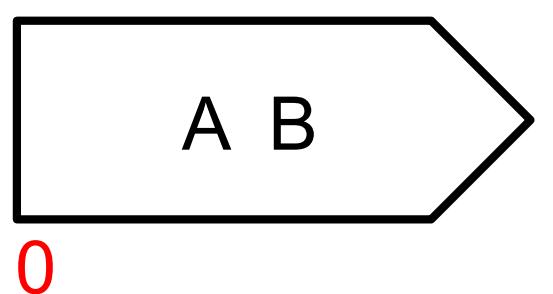


RBN: *Relative Block Number* → 0



Solução (2/10)

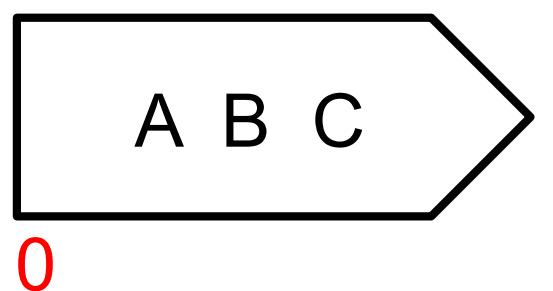
- Inserindo B
 - Adiciona-se B ao bloco existente





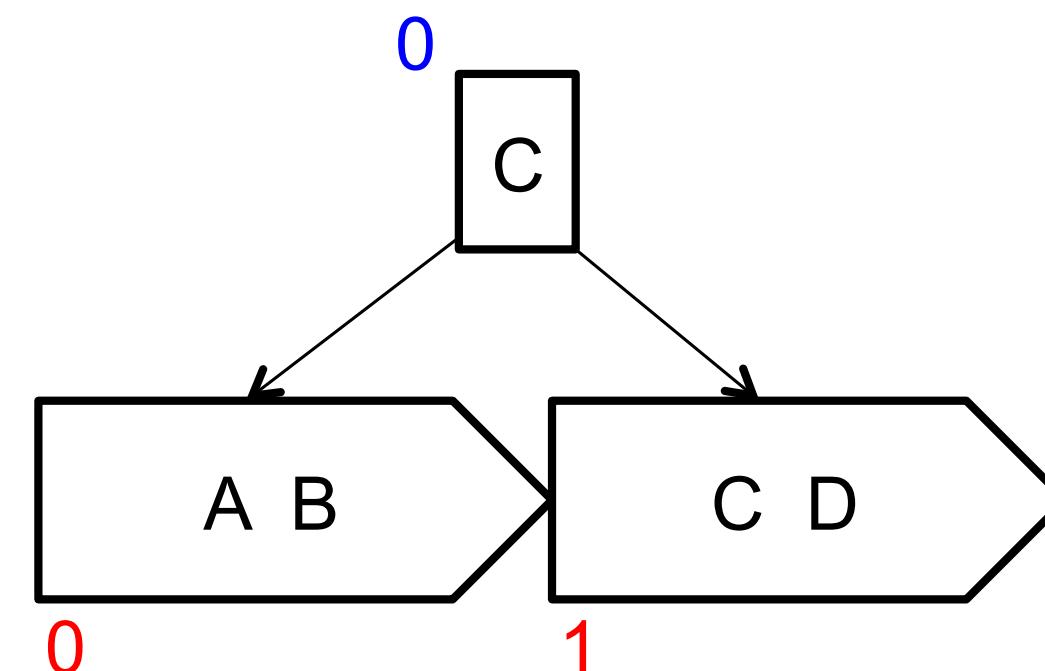
Solução (3/10)

- Inserindo C
 - Adiciona-se C ao bloco existente



Solução (4/10)

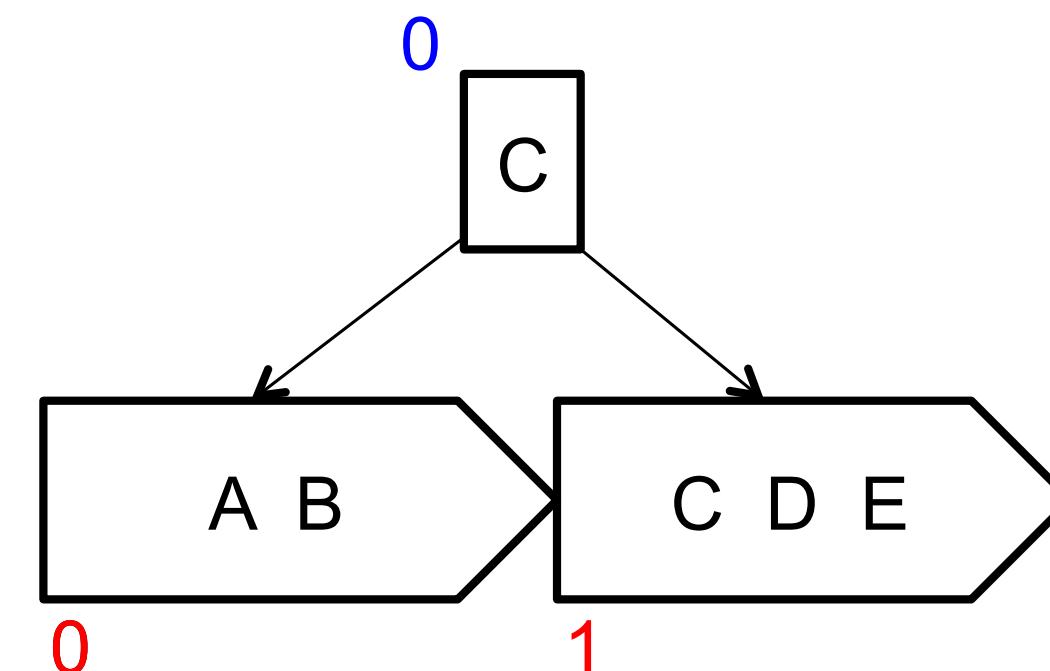
- Inserindo D
 - D não cabe no bloco existente
 - Faz-se split, dividem-se chaves entre blocos, cria-se a primeira página da árvore com o separador mínimo adequado



Registro de cabeçalho noRaiz: 0

Solução (5/10)

- Inserindo E
 - Adiciona-se E no bloco apropriado ($RBN=1$)

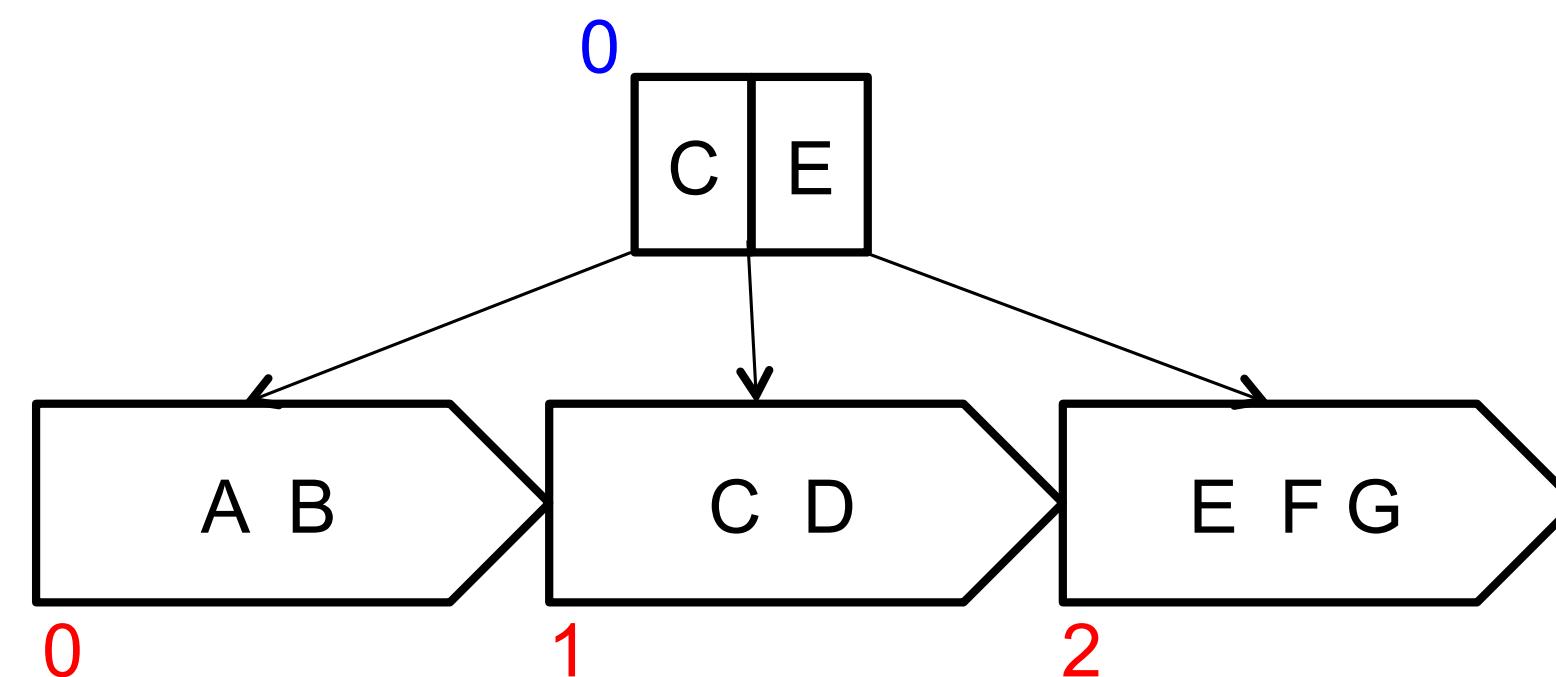


Registro de cabeçalho noRaiz: 0



Solução (7/10)

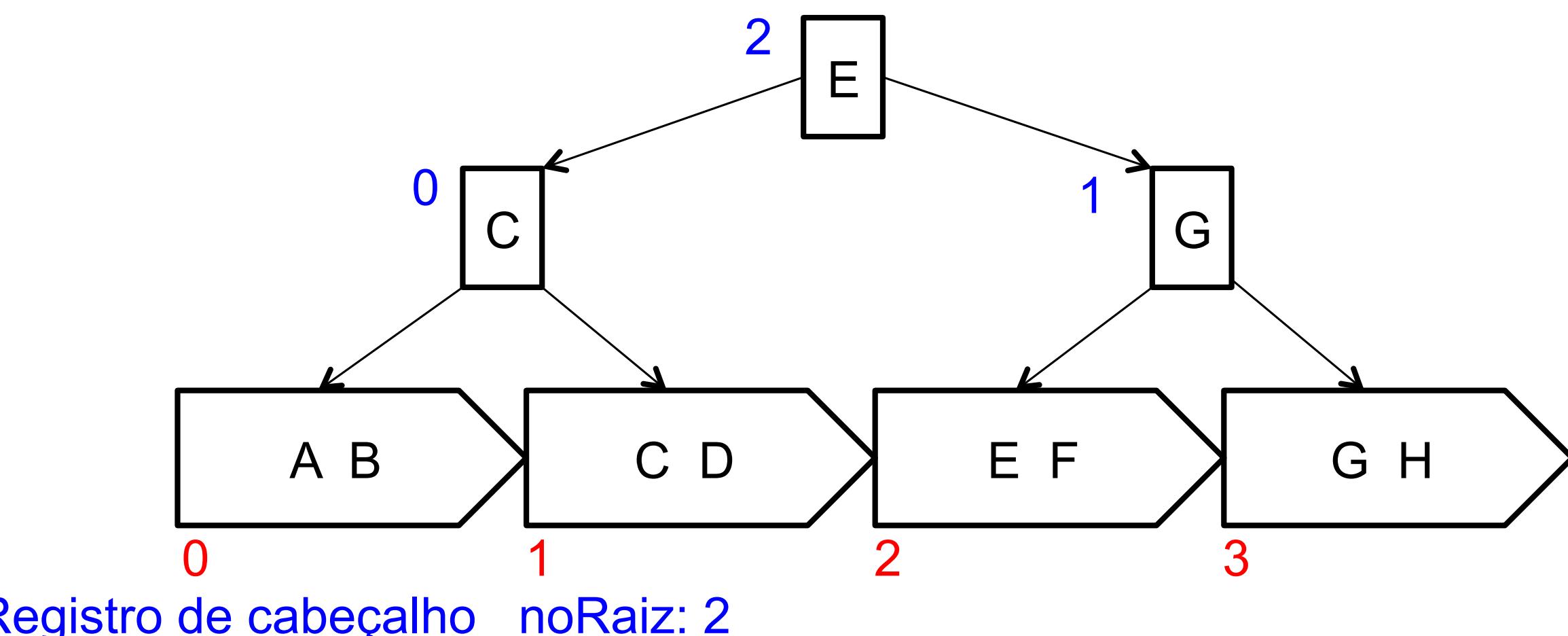
- Inserindo G
 - Adiciona-se G ao bloco apropriado (RBN=2)



Registro de cabeçalho noRaiz: 0

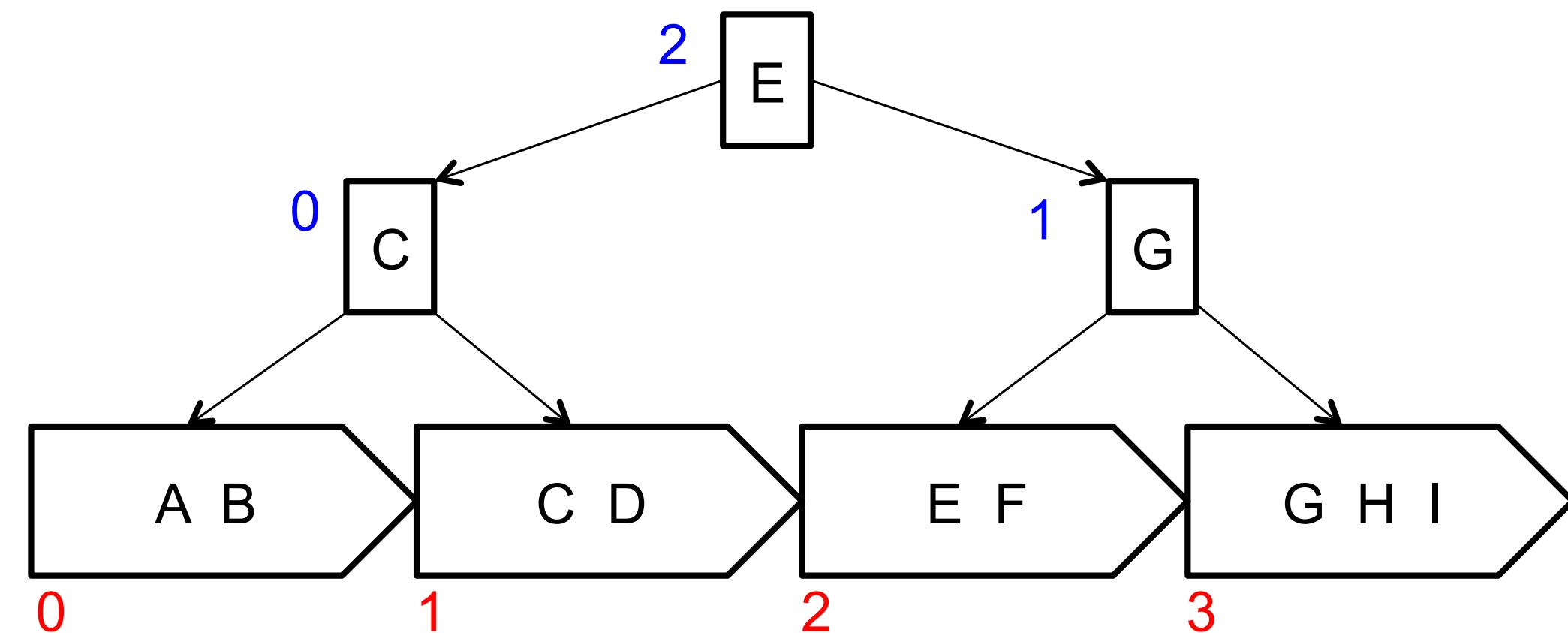
Solução (8/10)

- Inserindo H
 - H não cabe no bloco existente ($RBN=2$)
 - Faz-se split do bloco, dividem-se chaves entre blocos, cria-se um separador mínimo adequado entre as páginas e o promove ao nó da árvore
 - O nó da árvore também sofre split e promove uma chave, forçando a criação de mais um nó da árvore no nível acima



Solução (9/10)

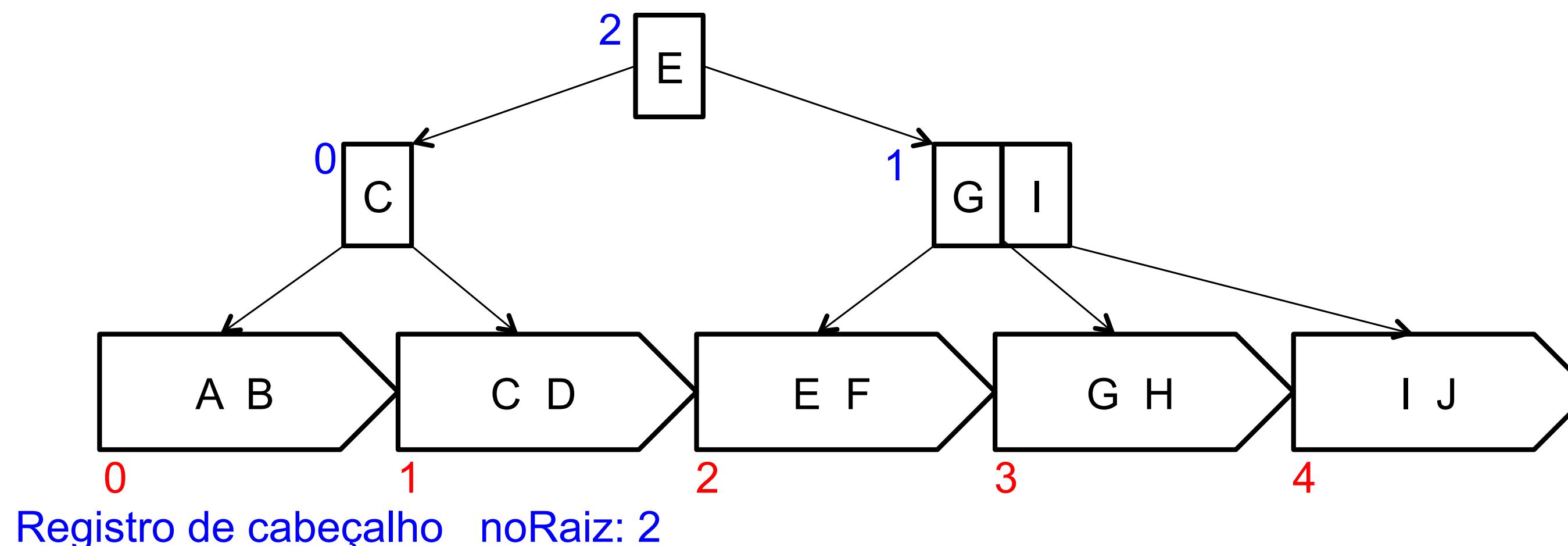
- Inserindo I
 - Adiciona-se I ao bloco apropriado ($RBN=3$)



Registro de cabeçalho noRaiz: 2

Solução (10/10)

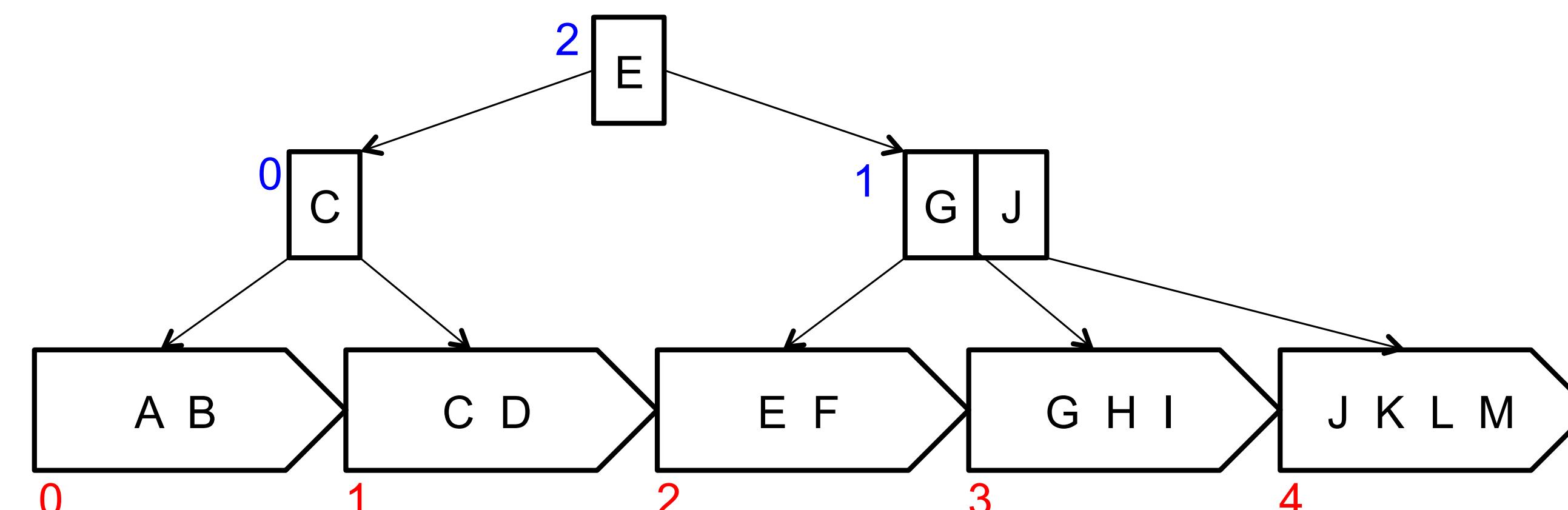
- Inserindo J
 - J não cabe no bloco existente ($RBN=3$)
 - Faz-se split do bloco, dividem-se chaves entre blocos, cria-se um separador mínimo adequado entre as páginas e o promove ao nó da árvore





Exercício 2

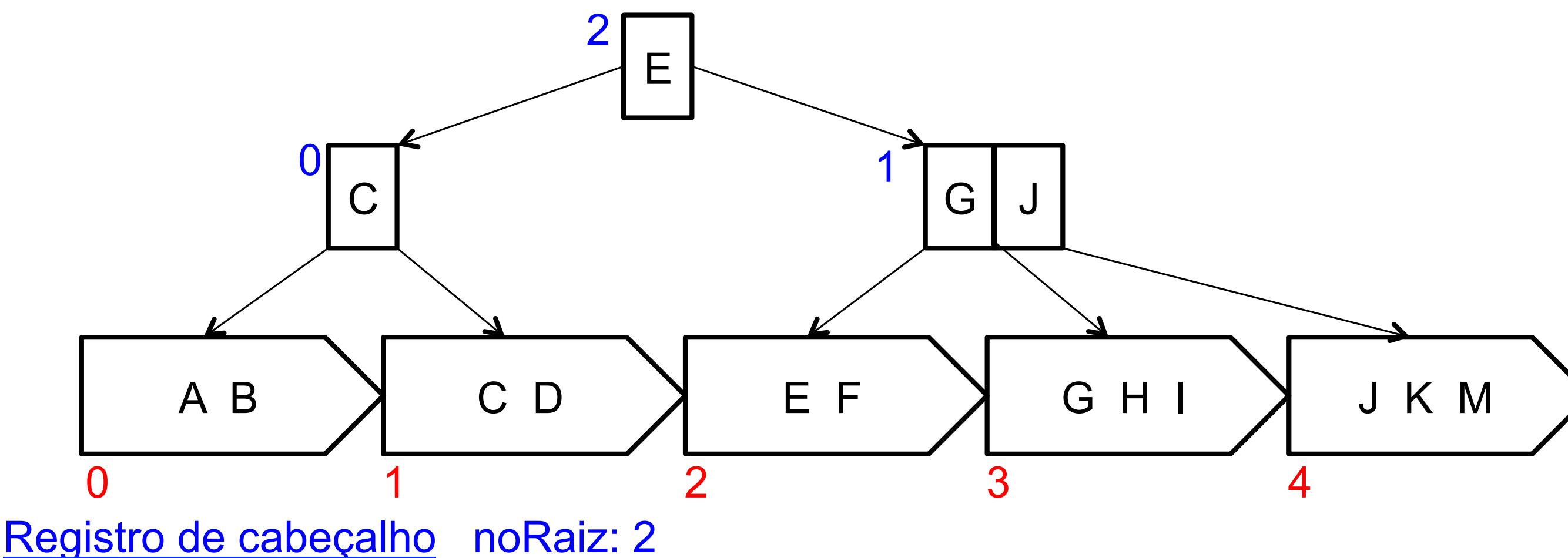
- Dada a árvore-B+ abaixo
 - Número mínimo de registros por bloco=2
 - Número máximo de registros por bloco=4
 - Ordem da árvore=3
- Remova L, M, K e A, nesta ordem



Registro de cabeçalho noRaiz: 2

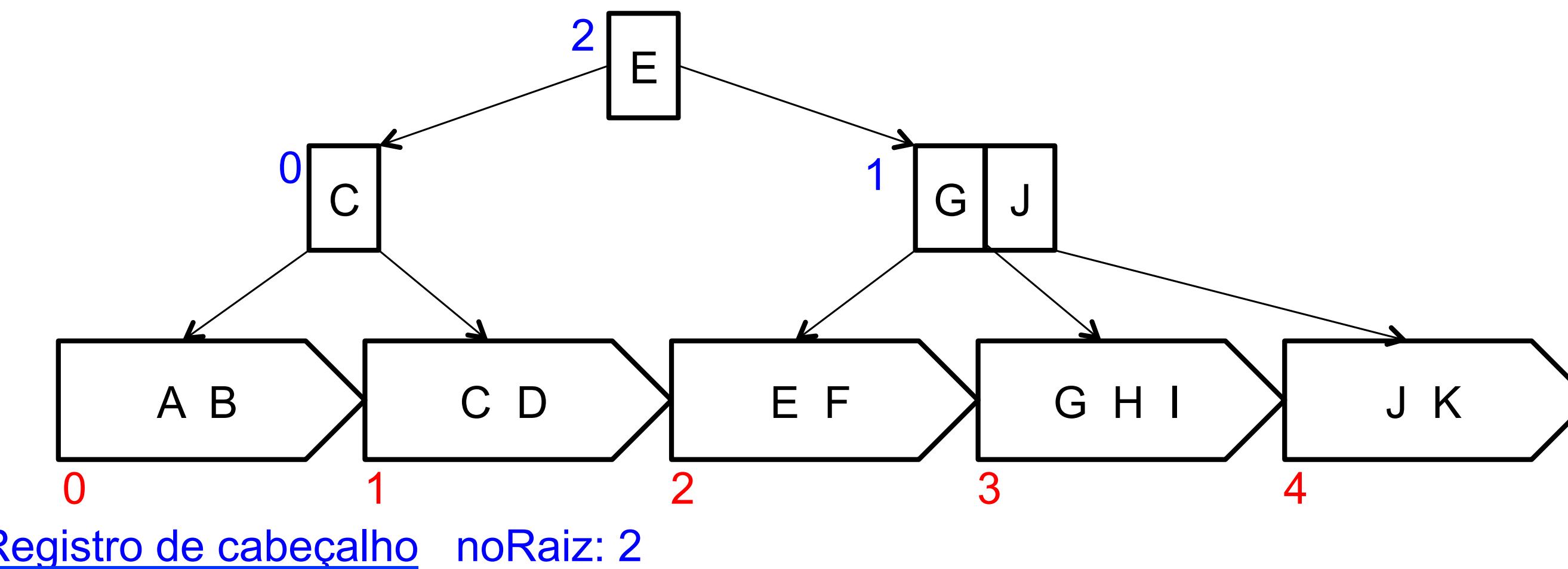
Solução (1/4)

- Removendo L
 - Bloco 4 mantém mais do que o mínimo de registros esperados



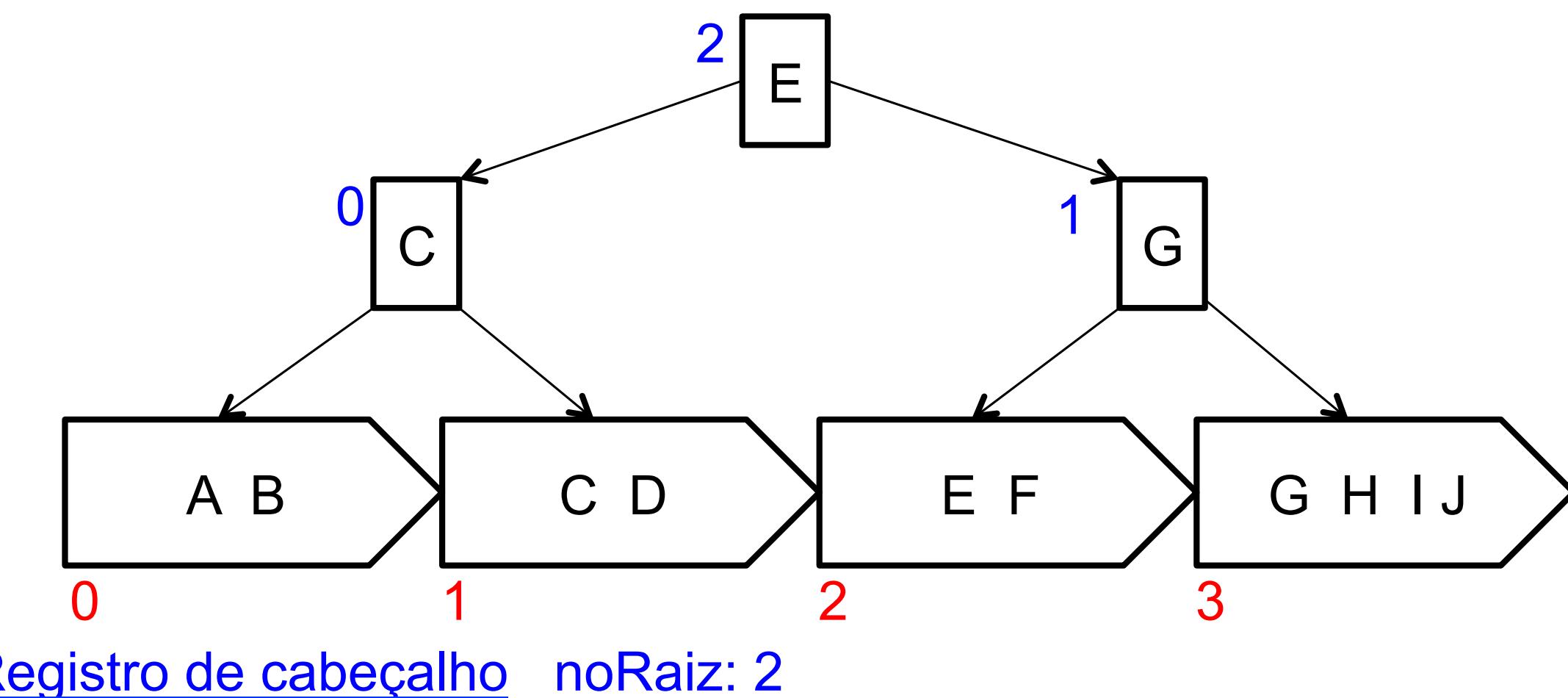
Solução (2/4)

- Removendo M
 - Bloco 4 mantém o mínimo de registros esperados



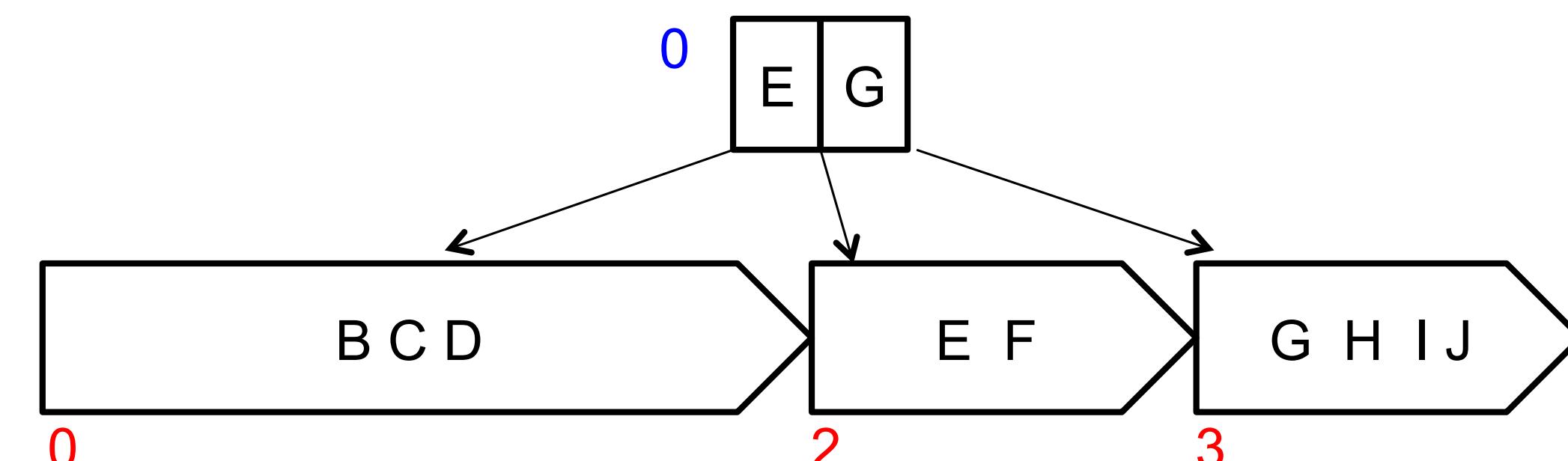
Solução (3/4)

- Removendo K
 - Concatena blocos 3 e 4
 - Remove separador I do índice



Solução (4/4)

- Removendo A
 - Concatena blocos 0 e 1
 - Remove separador C do índice, underflow, concatena 0, 1, 2, diminui a altura da árvore, atualiza o registro de cabeçalho.



Registro de cabeçalho noRaiz: 0



Características

- Árvore (*index set*)
 - ordem: 3
- Blocos de registros (*sequence set*)
 - número máximo de registros: 4
 - número mínimo de registros: 2

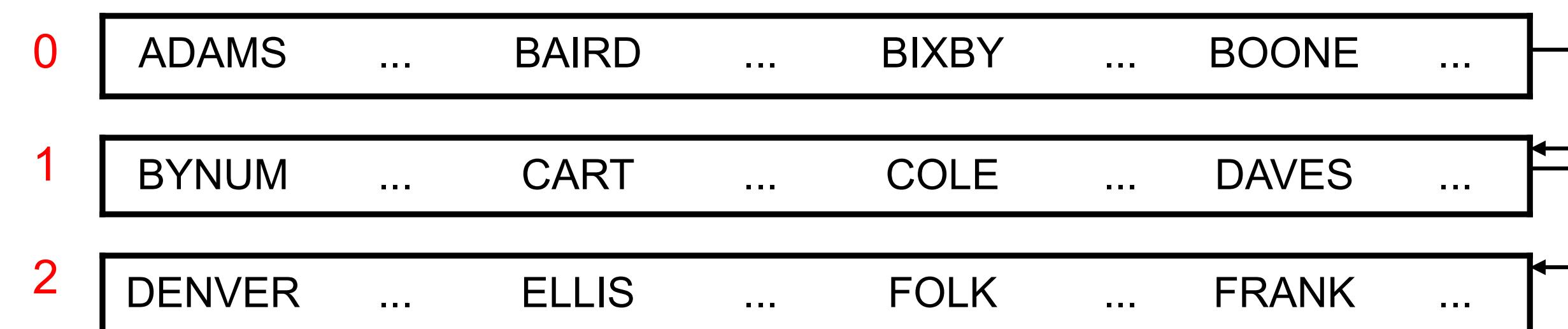
Exercício 3

Quais os separadores dos blocos, considerando uma árvore-B+ pré-fixada?

0	ADAMS	...	BAIRD	...	BIXBY	...	BOONE	...
1	BYNUM	...	CART	...	COLE	...	DAVES	...
2	DENVER	...	ELLIS	...	FOLK	...	FRANK	...

Resposta (1/1)

Quais os separadores dos blocos, considerando uma árvore-B+ pré-fixada?

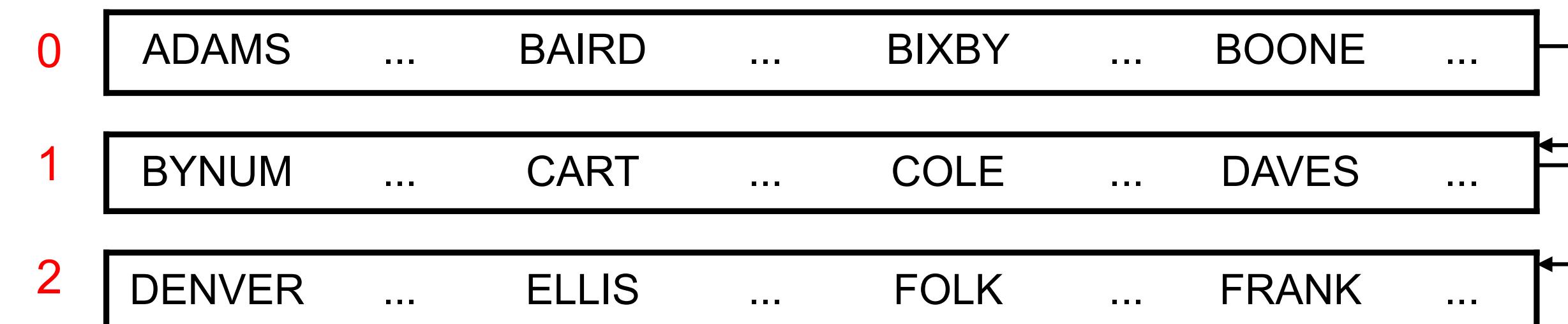


BY DE



Exercício 4

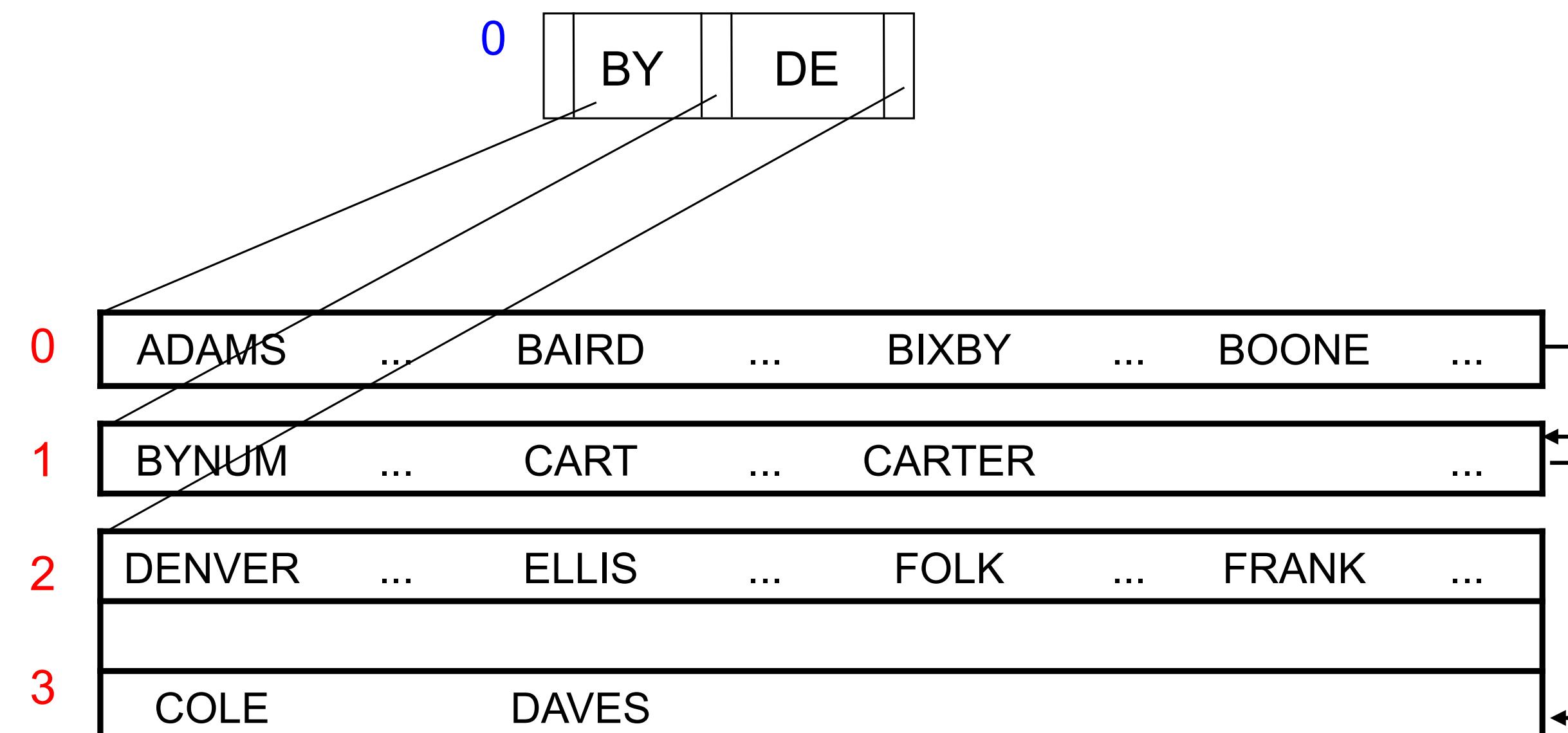
Construa a árvore-B+ pré-fixada sobre os blocos
do exercício 3



BY DE

Resposta (1/1)

Construa a árvore-B+ pré-fixada sobre os blocos do exercício 3



Registro de cabeçalho noRaiz: 0



- FOLK, M.J. File Structures, Addison-Wesley, 1992.
- File Structures: Theory and Practice”, P. E. Livadas, Prentice-Hall, 1990;
- Szwarcfiter, J.; Markezon, L. Estruturas de Dados e seus Algoritmos, 3a. ed.
LTC. Cap. 5
- Contém material extraído e adaptado das notas de aula dos professores
Moacir Ponti, Thiago Pardo, Leandro Cintra, Vanessa Braganholo, Thelma
Cecília Chirossi e Maria Cristina de Oliveira.