Árvore B

Prof. André Backes | @progdescomplicada

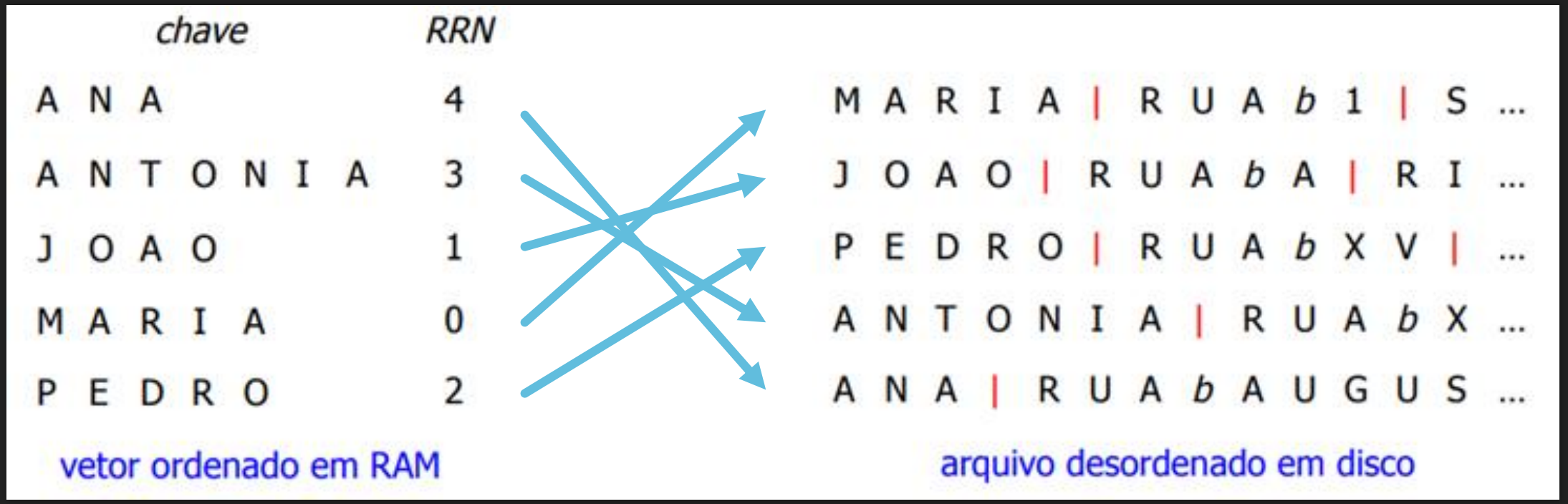
Consulta a arquivos binários grandes

- Arquivos binários grandes
  - Busca sequencial é muito custosa
  - Se arquivo estiver ordenado pode-se fazer busca binária, mas para arquivos grandes ainda não é eficiente o suficiente
- É possível acelerar a busca usando duas técnicas:
  - Acesso via cálculo do endereço do registro (*hashing*)
  - Acesso via estrutura de dados auxiliar (índice)

Índice

- Índice é uma estrutura de dados que serve para localizar registros no arquivo de dados
- 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
- Mas isso é eficiente?

Índice


chave
RRN
A N A 4
A N T O N I A 3
J O A O 1
M A R I A 0
P E D R O 2
M A R I A | R U A b 1 | S ...
J O A O | R U A b A | R I ...
P E D R O | R U A b X V | ...
A N T O N I A | R U A b X ...
A N A | R U A b A U G U S ...
vetor ordenado em RAM
arquivo desordenado em disco

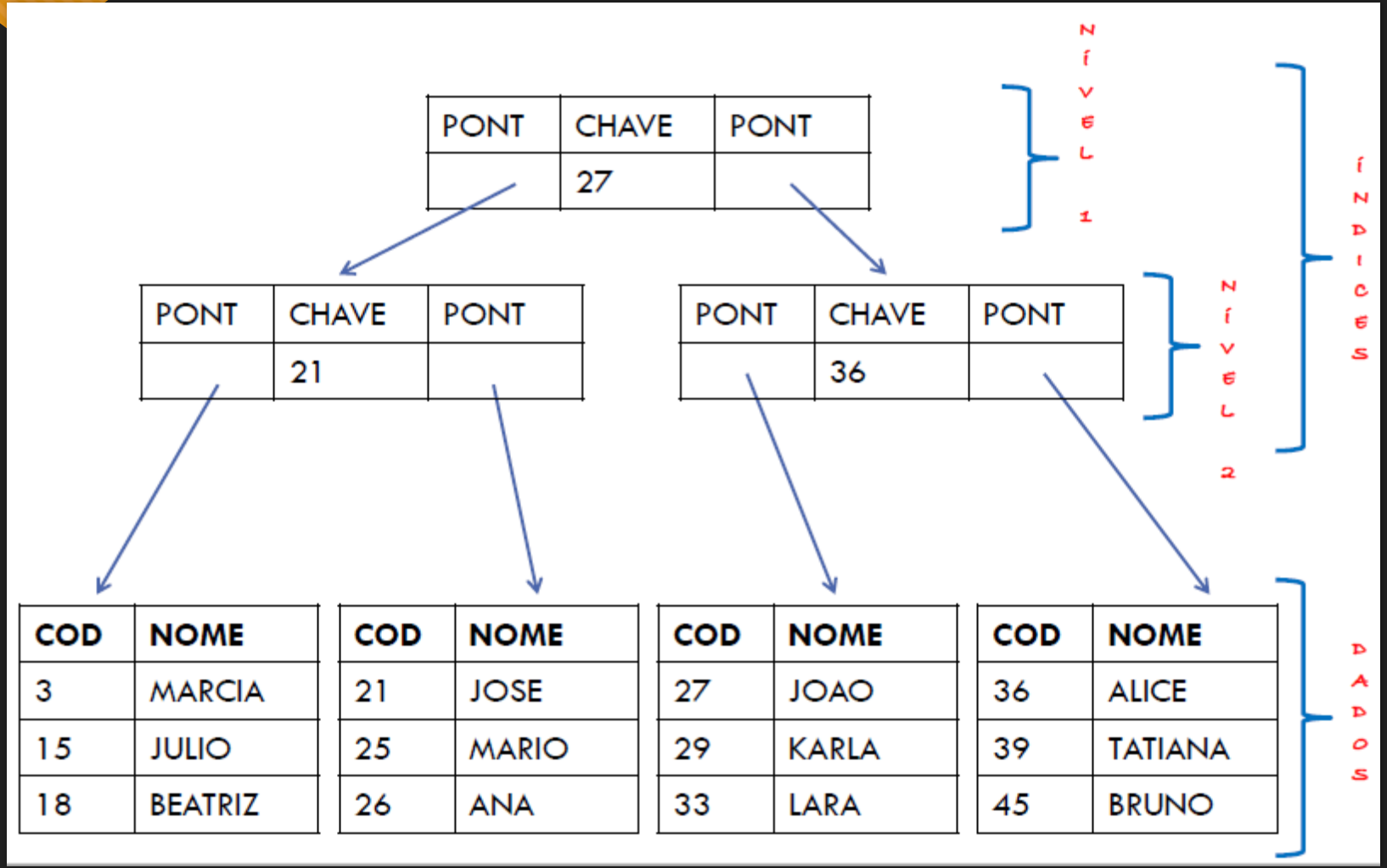
Índice

- Se tivermos que percorrer o arquivo de índice sequencialmente para encontrar uma determinada chave, o índice não terá muita utilidade
- Se o arquivo de índice estiver ordenado pode-se fazer busca um pouco mais eficiente
  - Busca binária
- Mas mesmo assim isso não é o ideal

Estruturas hierárquicas como índice

- 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
    - Dentro do bloco é feita busca sequencial
    - Isso exige que os registros dentro de um bloco estejam ordenados

Exemplo de um índice hierárquico



Estruturas hierárquicas como índice

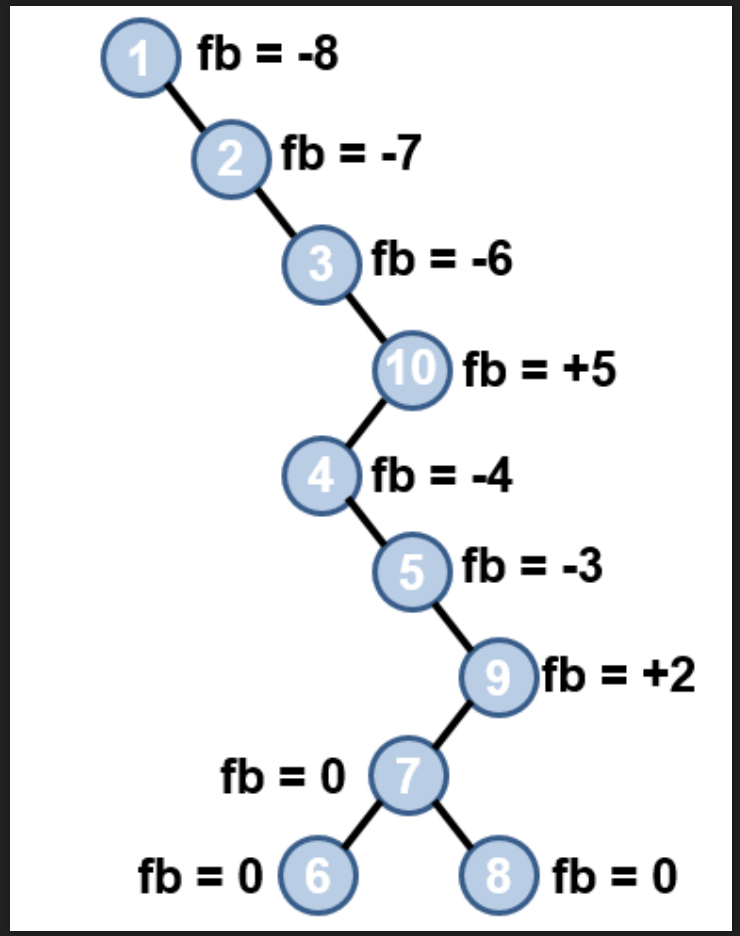
- A maioria das estruturas de índice é implementada por árvores de busca
  - Árvore Binárias de Busca
  - Árvore AVL
  - Árvore Rubro-Negra
  - Árvore de Múltiplos Caminhos

Problema da árvore binária de busca

- A eficiência da busca em uma árvore binária depende do seu balanceamento.
  - ***O(log N)***, se a árvore está balanceada
  - ***O(N)***, se a árvore não está balanceada
    - **N** corresponde ao número de nós na árvore

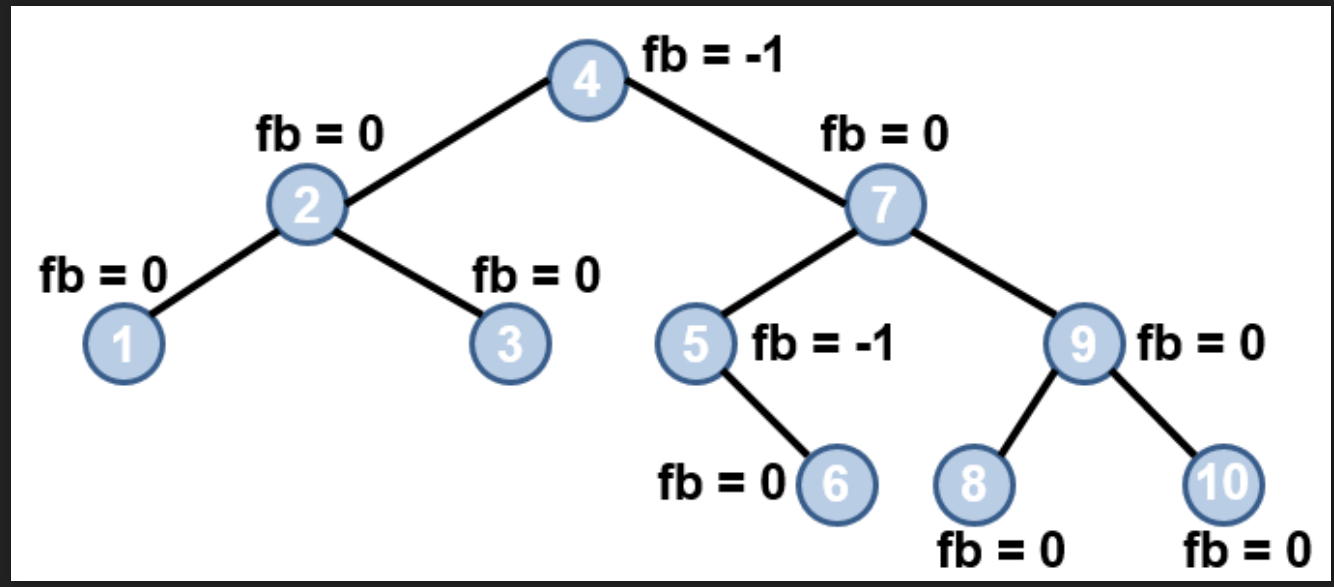
Problema da árvore binária de busca

- Dependendo da ordem em que os dados são inseridos na árvore, podemos criar uma árvore na forma de uma escada
- Exemplo
  - Inserção de valores ordenados
  - {1,2,3,10,4,5,9,7,8,6}



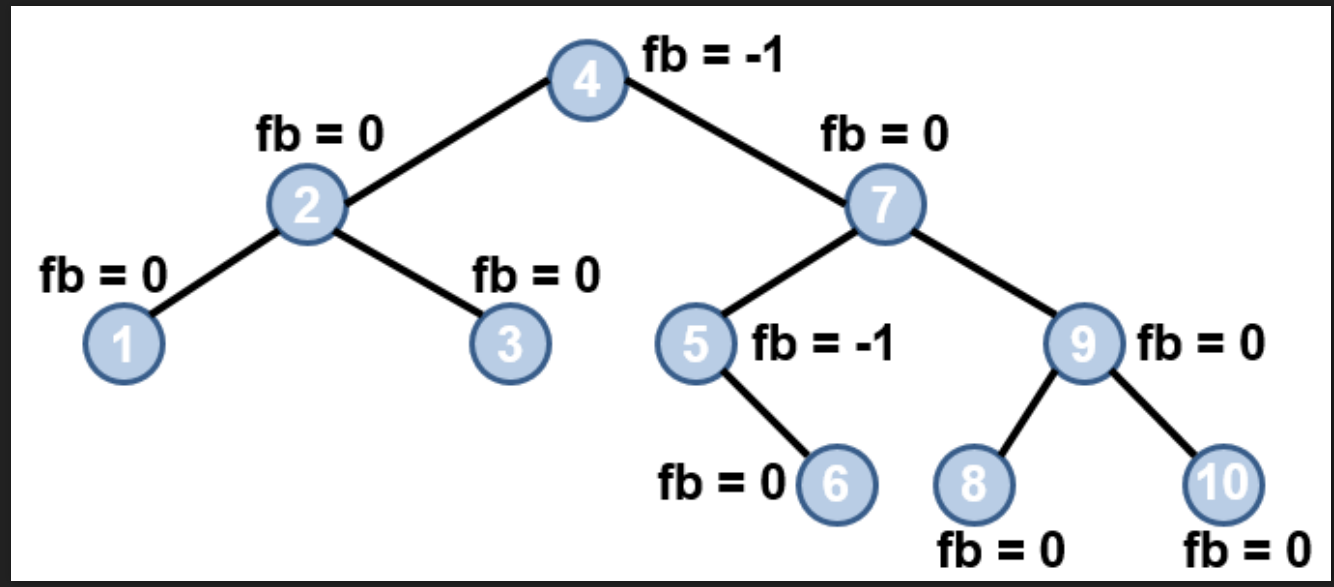
Árvore AVL

- Balanceamento
- Garante que os nós sejam distribuídos por igual, de modo que a profundidade da árvore seja minimizada para determinado conjunto de chaves



Problema da árvore AVL

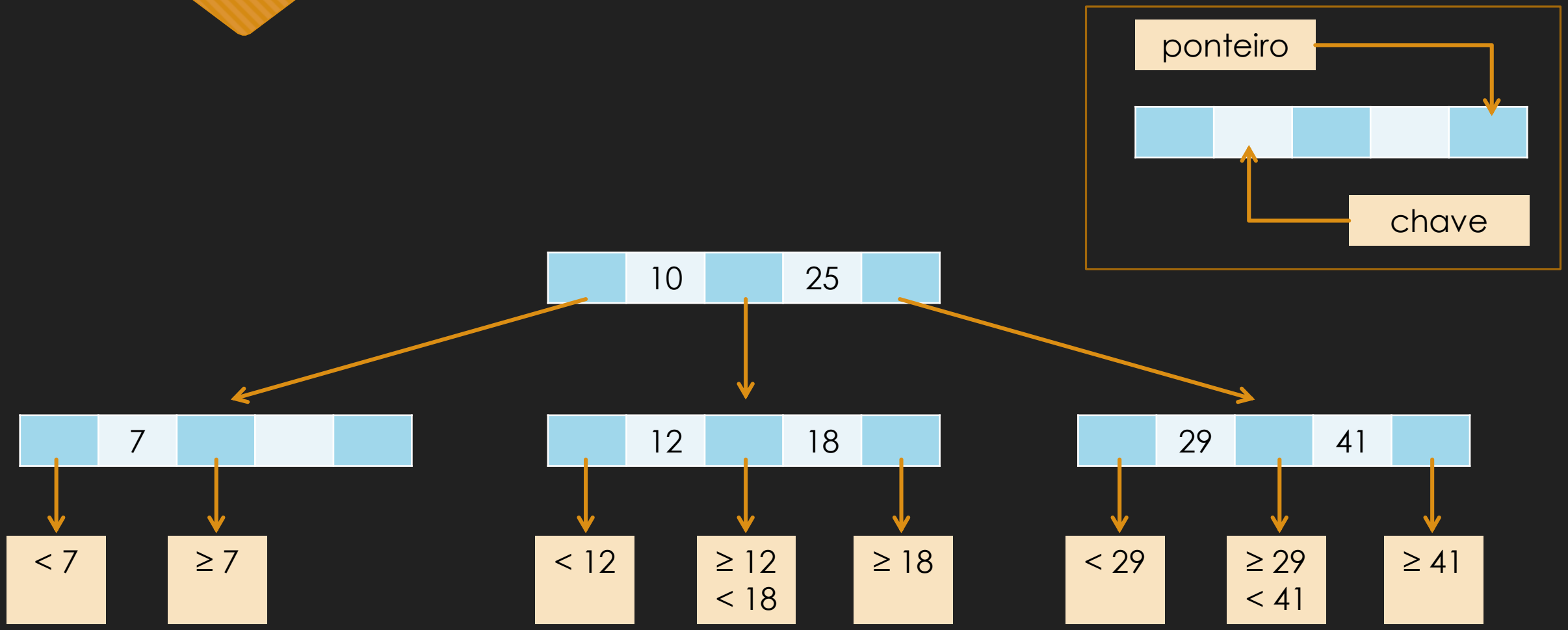
- Apesar de serem árvores binárias de busca balanceadas, ainda são excessivamente altas para uso eficiente como estrutura de índice.
  - Cada nível tem, no máximo, o dobro de elementos do anterior.



Árvores de múltiplos caminhos

- Melhor solução para o problema
- 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

Exemplo



Árvores de múltiplos caminhos

- Alguns exemplos de árvores
  - Árvore B
  - Árvore B+
  - Árvore B*

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

Árvore B

- Desenvolvida por Bayer e McCreight, 1972
  - Bayer, R.; McCreight, E. *Organization and Maintenance of Large Ordered Indexes*
  - Trabalho desenvolvido na *Boeing Scientific Researh Labs*
- São árvores de pesquisa balanceadas projetadas para funcionar bem em discos magnéticos ou outros dispositivos de armazenamento secundário
  - Voltado para arquivos volumosos
  - Proporciona rápido acesso aos dados
  - Muitos SGBD usam árvores B (ou suas variações) para armazenar informações

Árvore B

- Consegue armazenar índice e dados na mesma estrutura (mesmo arquivo físico)
- Um nó de uma árvore B é também chamado de página
  - Uma página armazena diversos registros da tabela original
  - Seu tamanho normalmente equivale ao tamanho de uma página em disco

Árvore B

- Características
  - O número de acessos ao disco exigidos para a maioria das operações em uma árvore B é proporcional a sua altura
  - Índice extremamente volumoso
- *Buffer-pool* pequeno
  - Apenas uma parcela do índice pode ser carregada em memória principal
  - Operações baseadas em disco
- Desempenho p roporcional a $\mathbf{log_K\ I}$ ou melhor
  - I: tamanho do índice
  - K: tamanho da página de disco

Nomenclatura

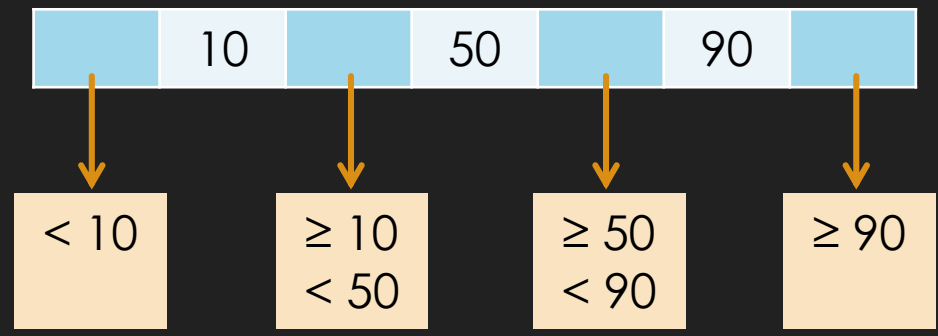
- Especifica precisamente as propriedades que devem estar presentes para uma estrutura de dados ser qualificada como árvore B
  - Direciona a implementação do algoritmo de remoção da árvore B
- Problema
  - Literatura não é uniforme no uso e definição dos termos

Ordem da Árvore B

- Depende do autor
- Bayer and McGreight (1972) Cormen (1979)
  - Número mínimo de chaves que podem estar em um nó da árvore
- Knuth (1973)
  - Número máximo de ponteiros (descendentes) que pode ser armazenado em um nó
  - **Chaves = ordem – 1 (máximo)**
  - Facilita a determinação de nó cheio

Ordem da Árvore B

- Nó
  - Também chamado de página
  - Sequência ordenada de chaves
  - Conjunto de ponteiros
  - **Chaves = ordem – 1 (máximo)**
- Exemplo
  - Árvore B de ordem 4
  - Máximo de 3 chaves e 4 ponteiros



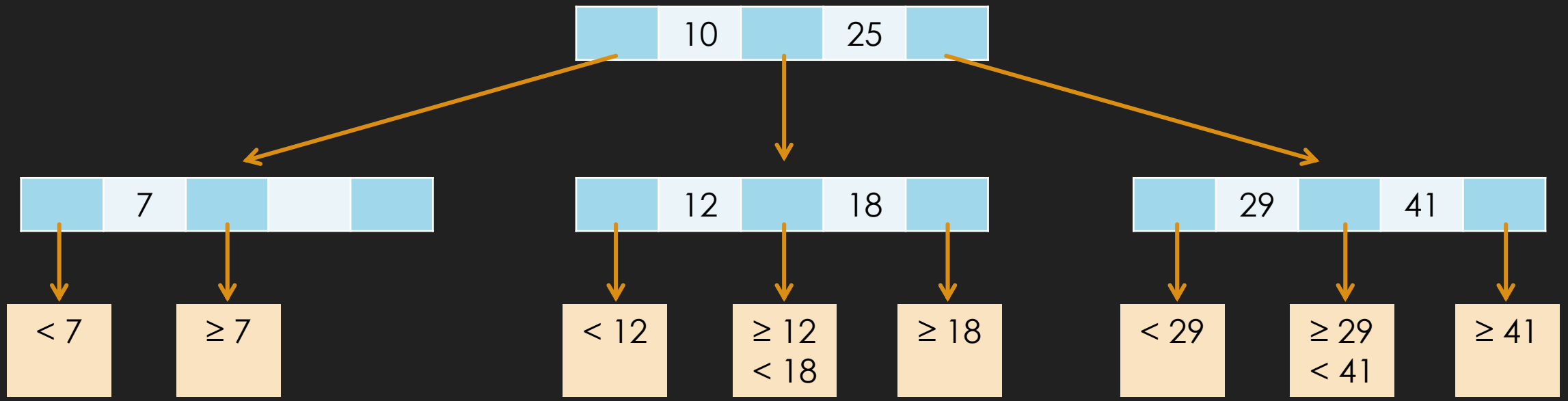
Nó folha

- Bayer and McGreight (1972)
  - Nível mais baixo das chaves
- Knuth (1973)
  - Um nível depois do nível mais baixo das chaves
  - Folhas: registros de dados que podem ser apontados pelo nível mais baixo das chaves

Árvore B de Ordem m

- Cada nó possui um máximo de **m** descendentes
- Um nó interno com **k** descendentes contém **k-1** chaves
- Cada nó, exceto a raiz e os nós folhas, tem pelo menos
  - $\lceil m/2 \rceil$ descendentes
  - $\lceil m/2 \rceil - 1$ chaves
- A raiz possui pelo menos 2 descendentes, a menos que seja um nó folha
- Todas as folhas aparecem no mesmo nível
  - Cada folha possui no mínimo $\lceil m/2 \rceil - 1$ chaves e no máximo **m - 1** chaves à (taxa de ocupação)

Árvore B de Ordem 3



Complexidade

- Profundidade do caminho de busca
  - Número máximo de acessos a disco
- O número de descendentes de um nível da árvore B é igual ao número de chaves contidas no nível em questão e em todos os níveis acima + 1

Complexidade

- Cálculo do número mínimo de descendentes de um nível para uma árvore B de ordem m

Nível	Número mínimo de descendentes
1	2
2	$2 \times \lceil m/2 \rceil$
3	$2 \times \lceil m/2 \rceil \times \lceil m/2 \rceil = 2 \times \lceil m/2 \rceil^2$
4	$2 \times \lceil m/2 \rceil \times \lceil m/2 \rceil \times \lceil m/2 \rceil = 2 \times \lceil m/2 \rceil^3$
...	...
d	$2 \times \lceil m/2 \rceil^{d-1}$

Complexidade

- Para um número de chaves N, temos (N + 1) descendentes no nível das folhas
- Altura:
  - $N + 1 \geq 2 * \left\lceil \frac{m}{2} \right\rceil^{d-1}$
  - $d \leq 1 + \log_{\lceil m/2 \rceil}(\frac{N+1}{2})$
- Uma árvore B de ordem m = 12 e N = 1.000.000 de chaves, sua altura é dada por
  - $d \leq 1 + \log_{\lceil 512/2 \rceil}\left(\frac{1000000+1}{2}\right)$
  - $d \leq 3{,}37$

Árvore B: inserção

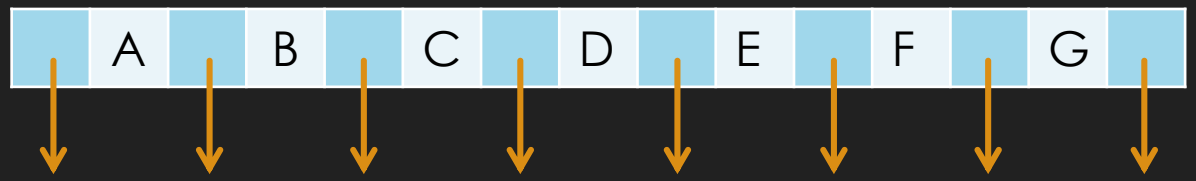
- Característica
  - Sempre realizada nos nós folhas
  - Se chave já existe, a inserção é inválida
- Situações a serem consideradas
  - Árvore vazia
  - Inserção nos nós folhas
  - *Overflow* no nó raiz

Árvore B: inserção

- Inserção na árvore vazia
  - Criação e preenchimento do nó
  - Primeira chave: criação do nó raiz
  - Demais chaves: inserção até a capacidade limite do nó

Árvore B: inserção

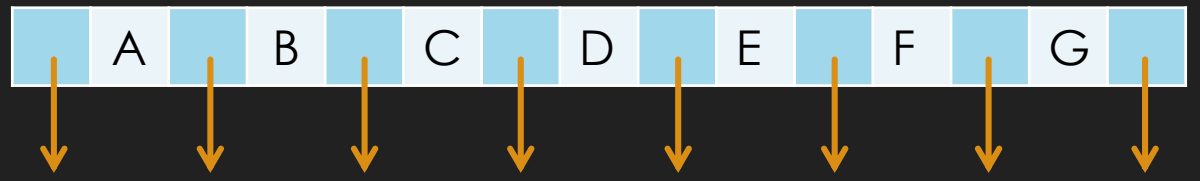
- Árvore de ordem m = 8
- Inserção das chaves: B C G E F D A
  - Inseridas desordenadamente
  - Mantidas ordenadas no nó
- Nó raiz = nó folha



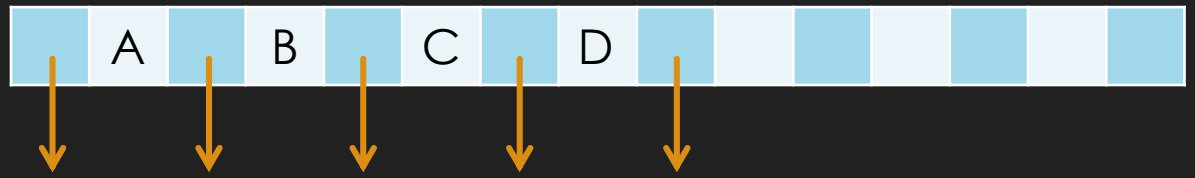
Árvore B: inserção

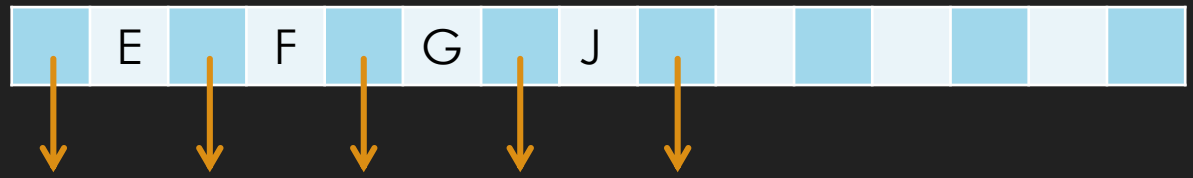
- Overflow no nó raiz
  - Nova chave não cabe no nó raiz (cheio)
- 1º passo: Particionamento do nó (*split*)
  - As chaves são distribuídas uniformemente nos dois nós
  - Realiza a inserção da nova chave

Árvore B: inserção



- Insere J

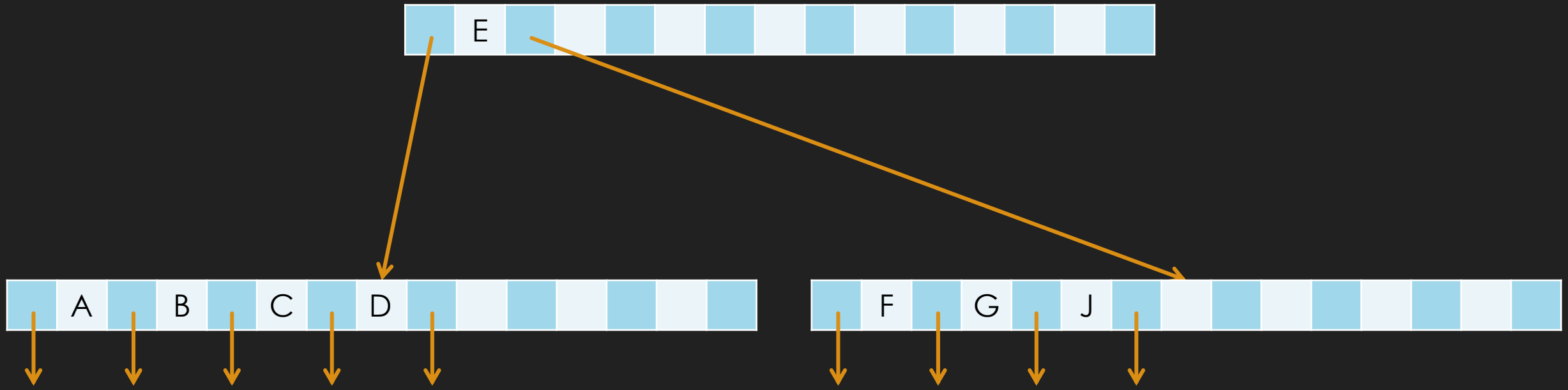




Árvore B: inserção

- 2º passo: criação de uma nova raiz
  - A existência de um nível mais alto na árvore permite a escolha das folhas durante a pesquisa
  - Qual deve ser a chave separadora?
- 3º passo: promoção de chave (*promotion*)
  - A primeira chave do novo nó resultante do particionamento é promovida para o nó raiz

Árvore B: inserção



Árvore B: inserção

- Inserção no nó folha
- 1º passo: busca
  - A árvore é percorrida até encontrar o nó folha no qual a nova chave será inserida
- 2º passo: inserção
  - Se o nó possui espaço, a chave é inserida de forma ordenada
  - Nó cheio: particionamento

Árvore B: inserção

- Particionamento do nó folha
  - Criação de um novo nó
  - Distribuição uniforme das chaves nos dois nós
  - Promoção: escolha da primeira chave do novo nó como chave separadora no nó pai
  - Ajuste do nó pai para apontar para o novo nó
  - Propagação de *overflow*

Árvore B: inserção

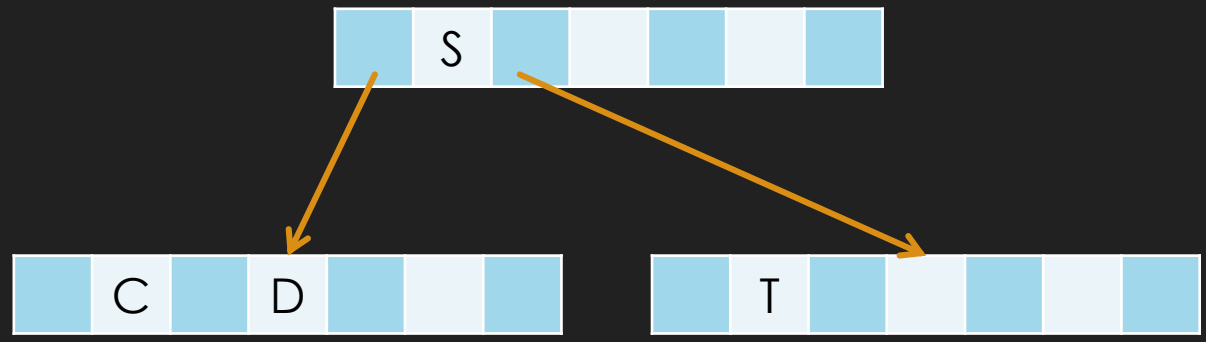
- Exemplo passo a passo
  - Ordem da árvore B: 4
  - Número de chaves: 3
  - Número de ponteiros: 4
- Chaves inseridas: **C S D T A M P I B**

Árvore B: inserção

- Insere as chaves C S D
  - Criação e inserção no nó raiz

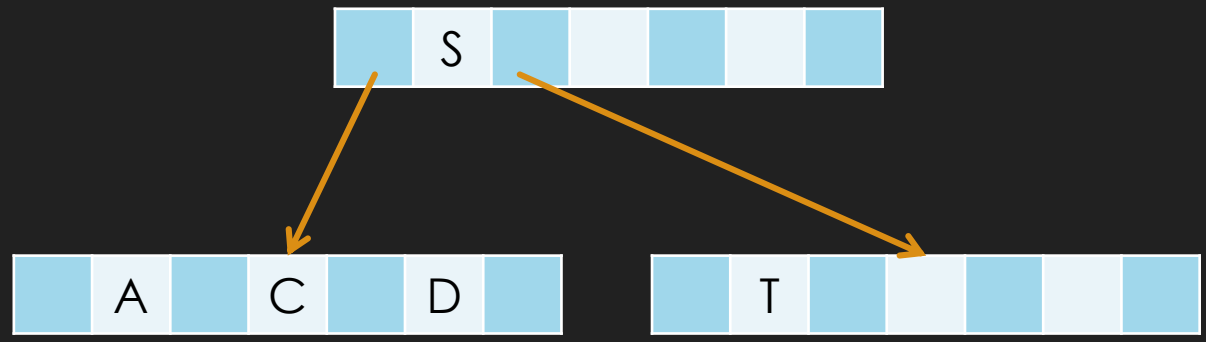
Árvore B: inserção

- Insere a chave T
  - Overflow do nó raiz
  - Particionamento do nó
  - Criação de uma nova raiz
  - Promoção de S



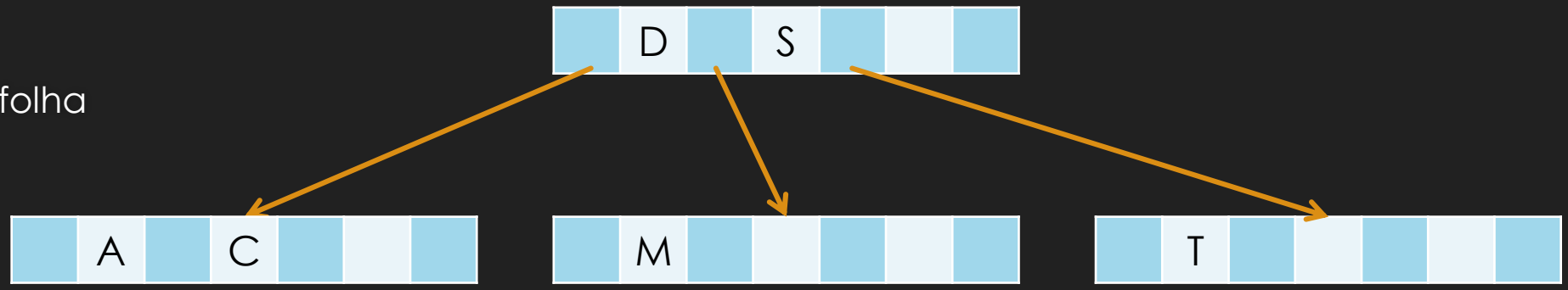
Árvore B: inserção

- Insere a chave A
  - Há espaço na folha



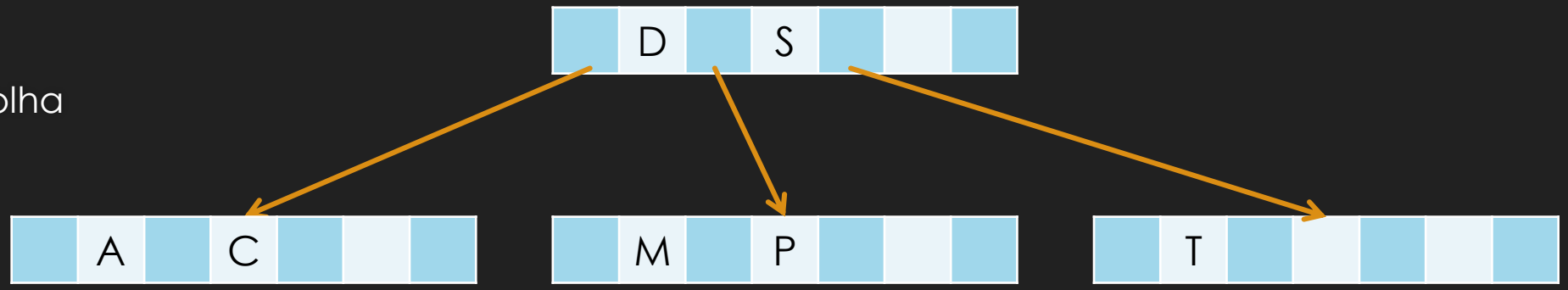
Árvore B: inserção

- Insere a chave M
  - *Overflow* do nó folha



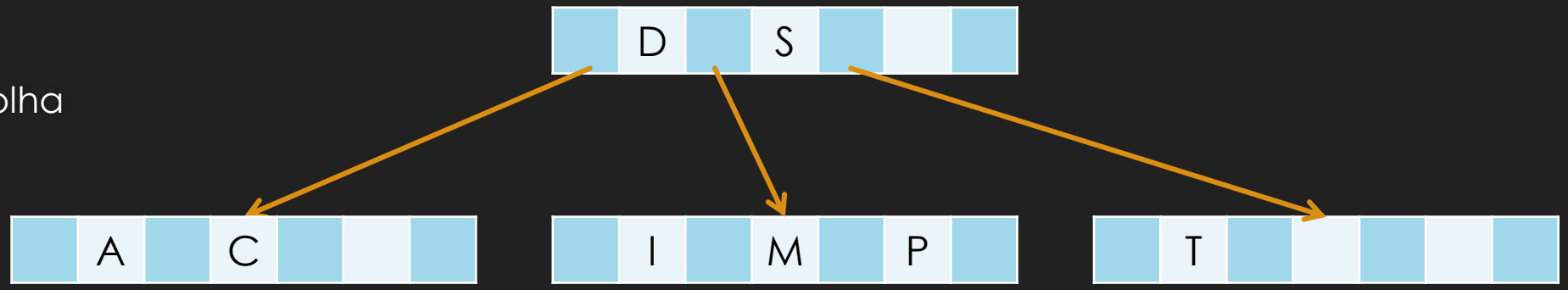
Árvore B: inserção

- Insere a chave P
  - Há espaço na folha



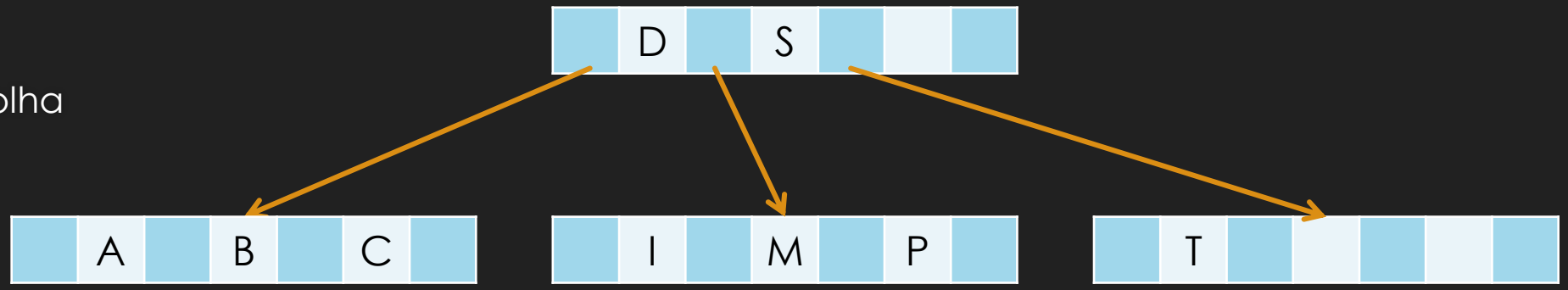
Árvore B: inserção

- Insere a chave I
  - Há espaço na folha



Árvore B: inserção

- Insere a chave B
  - Há espaço na folha



Árvore B: inserção

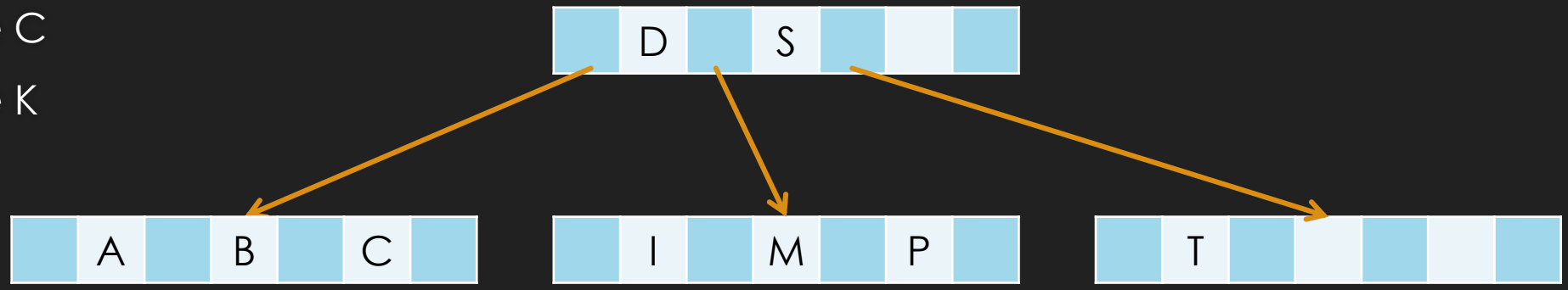
- Sobre o particionamento (ou *split*)
  - Ele se propaga para os pais dos nós, podendo, eventualmente, atingir a raiz da árvore
  - Nesse caso, o nó raiz é particionado normalmente, mas, como a raiz não tem pai, cria-se um novo nó, que passa a ser a nova raiz
  - O particionamento da raiz é a única forma de aumentar a altura da árvore

Árvore B: busca

- Semelhante a busca em uma árvore binária.
  - Decisão em cada nó não é mais binária
  - Devemos tomar uma decisão de ramificação de várias vias
- Para buscar um valor X
  - Primeiro verifique se o mesmo se encontra na raiz
  - Se X não existe na raiz
    - Percorra as chaves e acesse o ponteiro anterior a primeira chave maior que X
    - Se X for maior que todas as chaves, acesse o último ponteiro
  - Aplique o método recursivamente

Árvore B: busca

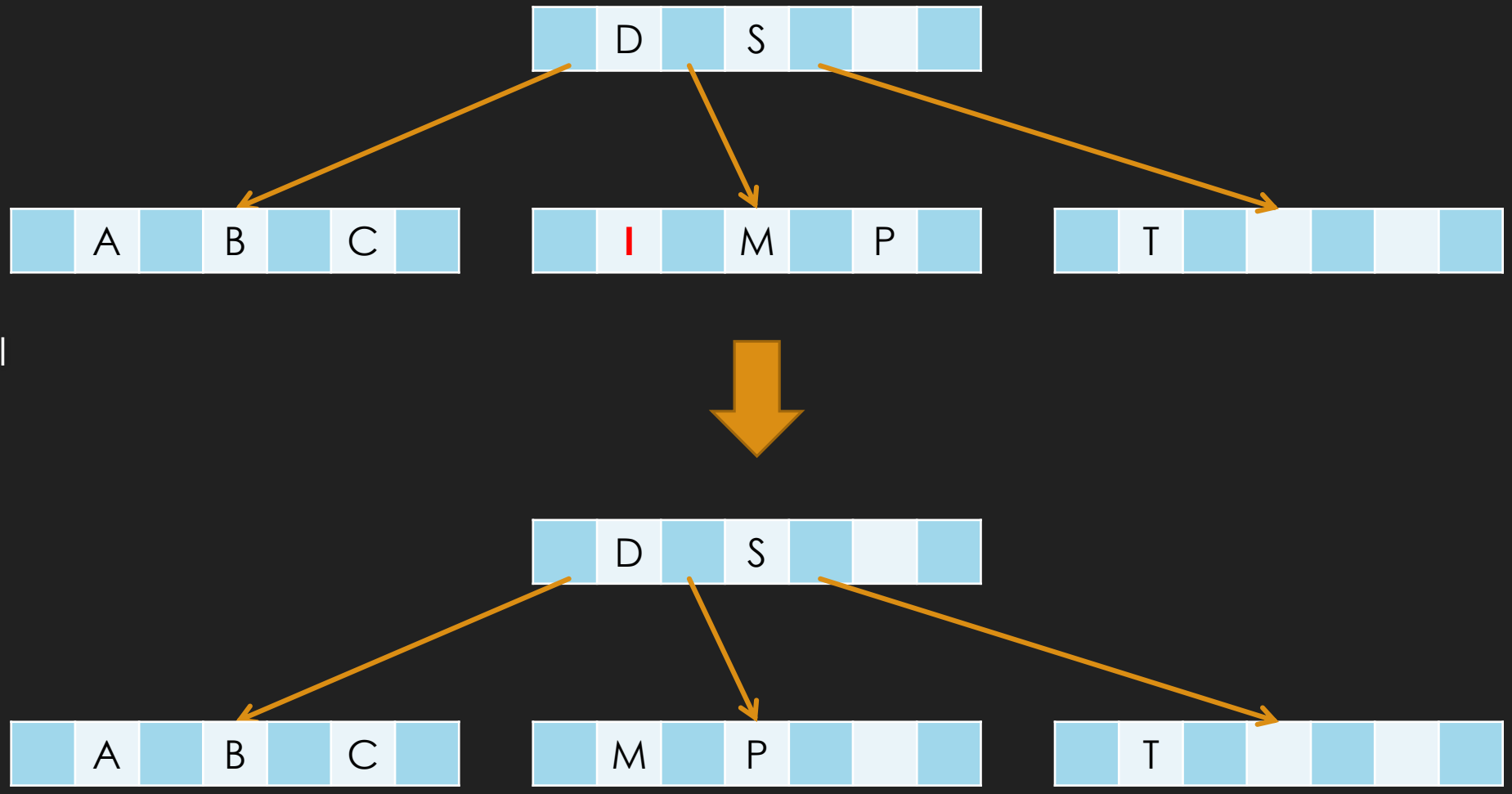
- Busca pela chave C
- Busca pela chave K



Árvore B: remoção

- Caso 1: remoção de uma chave em um nó folha, sem causar *underflow*
  - Situação mais simples possível
  - Garante a taxa de ocupação (número mínimo de chaves no nó)
- Solução
  - Eliminar a chave do nó
  - Rearranjar as chaves remanescentes dentro do nó para fechar o espaço liberado

Árvore B: remoção

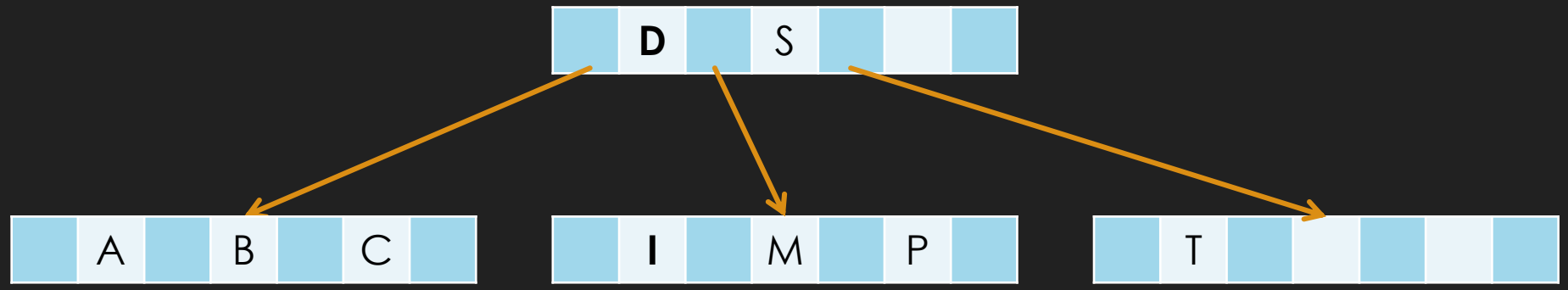


- Remove a chave I

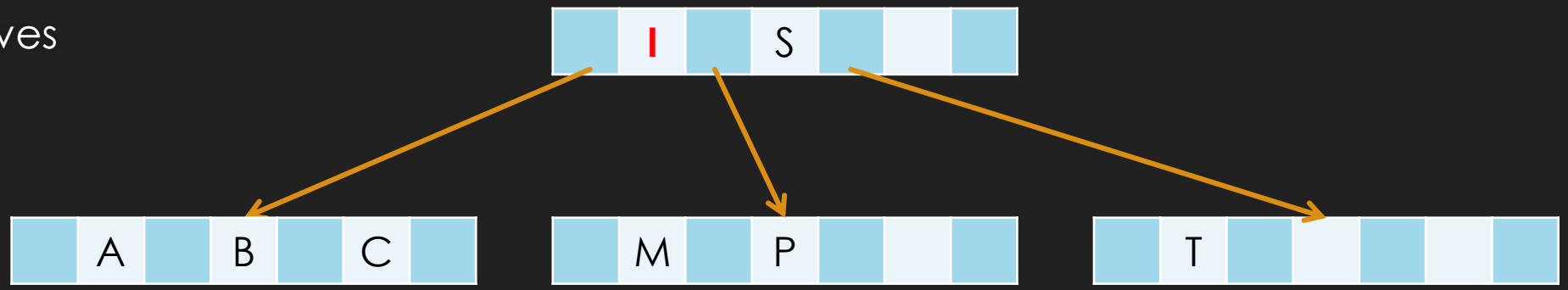
Árvore B: remoção

- Caso 2: remoção de uma chave em um nó que não seja folha
- Solução
  - Sempre remover chaves somente das folhas
- Passos
  - Trocar a chave a ser removida com a sua chave sucessora imediata
  - Essa chave deve estar em um nó folha
  - Remover a chave diretamente do nó folha

Árvore B: remoção



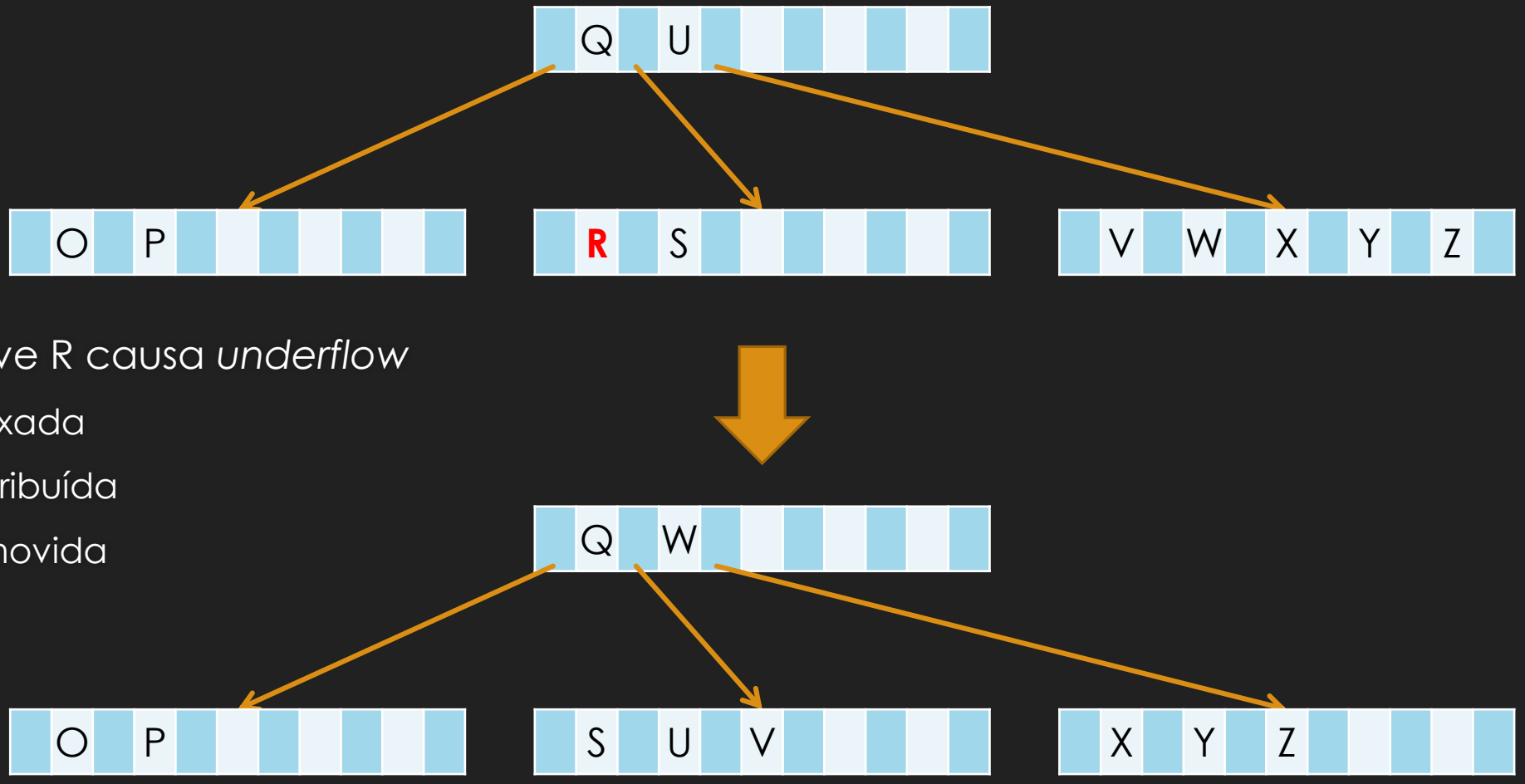
- Remoção da chave D
- Primeiro, trocar ela de lugar com a chave I
- Remover D do nó folha
- Rearranjar as chaves



Árvore B: remoção

- Caso 3: remoção de uma chave em um nó causando *underflow*
  - Nó fica com um número de chaves menor que o mínimo
- Solução: **redistribuição**
  - Procurar um nó irmão (i.e., com o mesmo pai) adjacente que contenha mais chaves do que o mínimo
- Se encontrou nó irmão
  - Redistribuir as chaves entre os nós
  - Reacomodar a chave separadora, modificando o conteúdo do nó pai

Árvore B: remoção



- Remoção da chave R causa *underflow*
  - Chave U é rebaixada
  - Chave V é redistribuída
  - Chave W é promovida

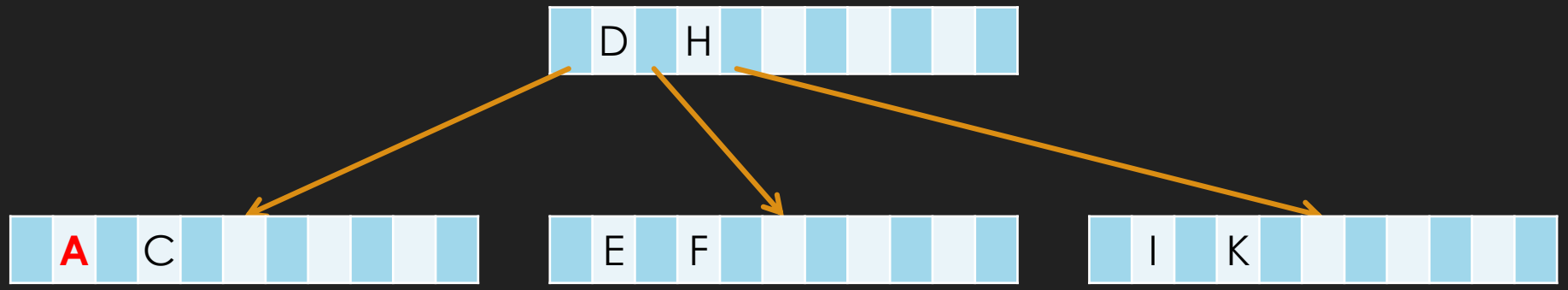
Árvore B: remoção

- Caso 4: remoção de uma chave em um nó causando *underflow*, redistribuição **não** é possível
  - Corrigir o nó afetado causa *underflow* no irmão
  - O nó irmão (i.e., com o mesmo pai) não possui mais chaves do que o mínimo
- Solução: **concatenação**
- Combinar num único nó:
  - O nó que sofreu *underflow*
  - O nó irmão adjacente
  - A chave separadora no nó pai

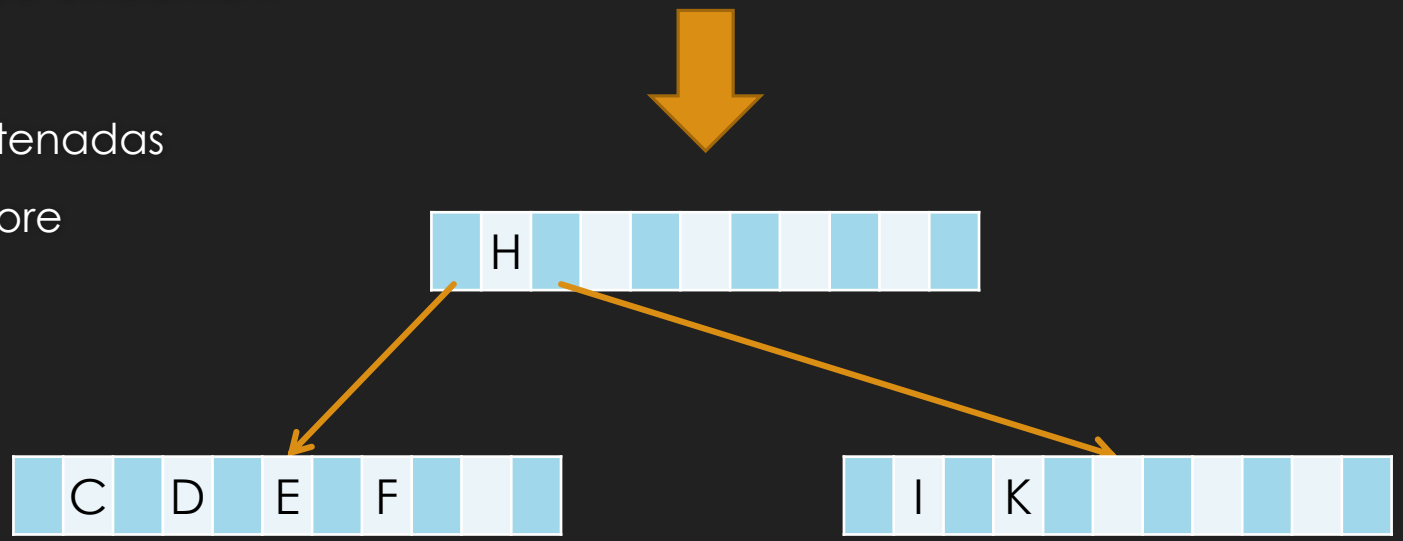
Árvore B: remoção

- Sobre a concatenação
  - Processo inverso do particionamento (*split*)
  - Reduz o número total de nós da árvore
  - Reverte a promoção de uma chave
  - Pode causar *underflow* no nó pai, o qual deve ser tratado
  - Pode ser propagada em direção ao nó raiz

Árvore B: remoção



- Remoção da chave A causa *underflow*
  - Chave D é rebaixada
  - Chaves C D E F são concatenadas
  - Redução de um nó na árvore
  - Tratar *underflow* no nó pai

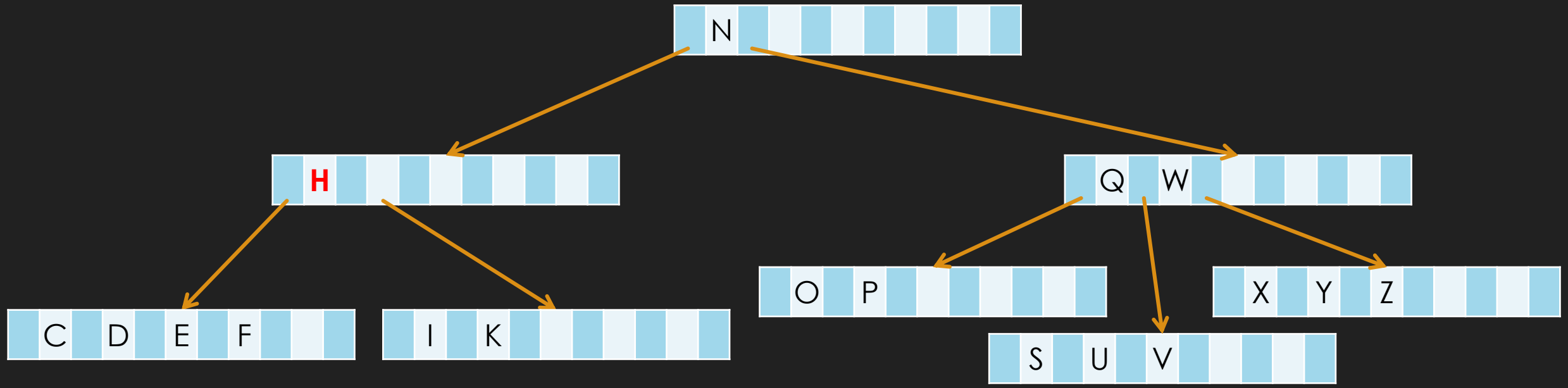


Árvore B: remoção

- Caso 5: remoção de uma chave em um nó filho causa *underflow* no nó pai
- Solução
  - Utilizar redistribuição ou concatenação
  - Depende da quantidade de chaves que o nó irmão adjacente contém

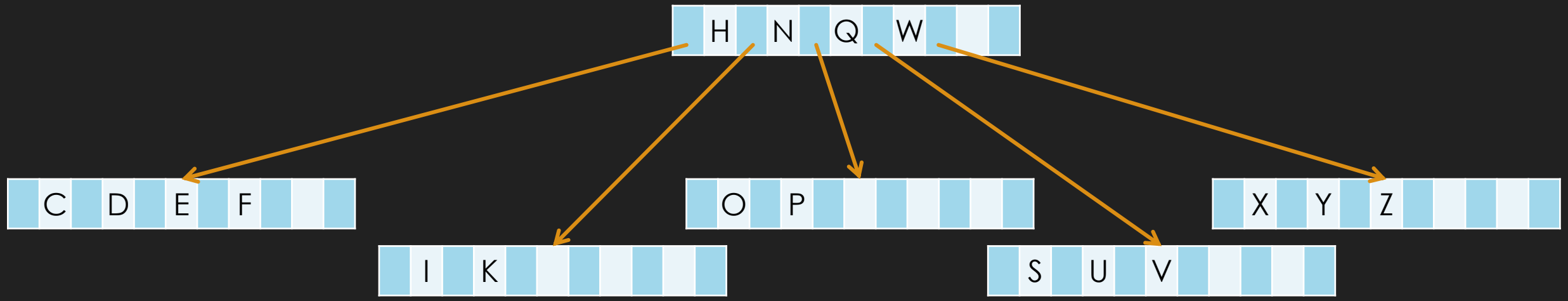
Árvore B: remoção

- Remoção de uma chave no filho causou *underflow* no nó contendo H



Árvore B: remoção

- Solução: concatenação com nó irmão
  - Redução no número de nós na árvore
  - Redução na altura da árvore



Árvore B: remoção

- Caso 6: remoção causa diminuição da altura da árvore
  - O nó raiz possui uma única chave
  - A chave é absorvida pela concatenação de seus nós filhos
- Solução
  - Eliminar a raiz antiga
  - Tornar no nó resultante da concatenação dos nós filhos a nova raiz da árvore
  - Redução na altura da árvore
  - Ocorreu no Caso 5

Remoção: passo a passo

1. Chave a ser removida não está nem nó folha, trocar com sua sucessora imediata que está em um nó folha
2. Remova a chave
3. Após a remoção, se o nó possui o número mínimo de chaves, algoritmo termina
4. Se ocorreu *underflow*, verifique o número de chaves nos nós irmãos
   a) **Redistribuição**: um nó irmão possui mais do que o número mínimo de chaves
   b) **Concatenação**: nenhum nó irmão possui mais do que o número mínimo de chaves
5. Após concatenação, repita os passos 3 a 5 para o nó pai
6. Se a última chave da raiz for removida, a altura da árvore é diminuída

Redistribuição

- Representa uma ideia diferente do particionamento (*split*)
  - Não se propaga para os nós superiores
  - Efeito local na árvore
  - Baseada no conceito de nós irmãos adjacentes
  - Dois nós logicamente adjacentes, mas com pais diferentes não são irmãos

Redistribuição

- Existem várias formas das chaves serem redistribuídas
  1) Mover somente uma chave, mesmo que a distribuição das chaves entre as páginas não seja uniforme
  2) Mover ***k*** chaves
  3) Distribuição uniforme das chaves entre os nós (**mais comum**)

Concatenação x redistribuição

- Concatenação
  - Dois nós podem ser concatenadas se são irmãos adjacentes e juntos possuem menos de **(m-1)** chaves
- Redistribuição
  - Ocorre quando a soma das chaves de dois nós irmãos é maior ou igual a **(m-1)**
  - Funciona como uma concatenação seguida de particionamento
    - Nós são concatenados. Total de chaves fica igual ou maior do que **(m-1)**, o que não é permitido
    - Particionar o nó concatenado

Concatenação x redistribuição

- Redistribuição é sempre uma opção melhor
  - Operação menos custosa, pois não é propagável: o conteúdo do nó pai é modificado, mas o número de chaves é mantido
  - Ela evita que o nó fique cheio, deixando espaço para futuras inserções

Redistribuição durante a inserção

- Pode ser utilizada como uma alternativa ao particionamento
  - Permite melhorar a taxa de utilização do espaço alocado para a árvore
- Particionamento
  - Divide uma página com *overflow* em duas páginas semivazias
- Redistribuição
  - A chave que causou *overflow* (além de outras chaves) pode ser colocada em outra página

Redistribuição durante a inserção

- Vantagens
  - A rotina de redistribuição já está codificada para prover suporte à remoção
  - A redistribuição evita, ou pelo menos adia, a criação de novas páginas
  - Tende a tornar a árvore B mais eficiente em termos de utilização do espaço em disco
  - Garante um melhor desempenho na busca

Particionamento x Redistribuição

- Somente particionamento na inserção
  - No pior caso, a utilização do espaço é de cerca de 50%
  - Em média, para árvores grandes, o índice de ocupação é de ~69%
- Com redistribuição na inserção
  - Em média, para árvores grandes, o índice de ocupação é de ~86%