ÁRVORE B

Prof. André Backes | @progdescomplicada

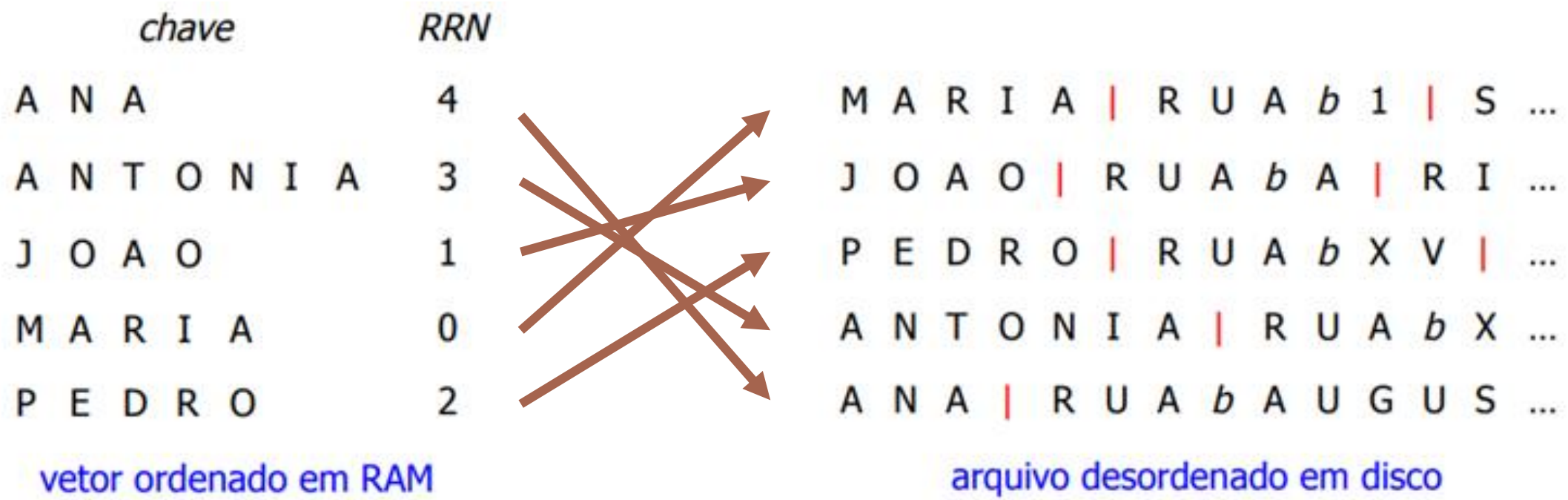
Consulta a arquivos binários grandes

- 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

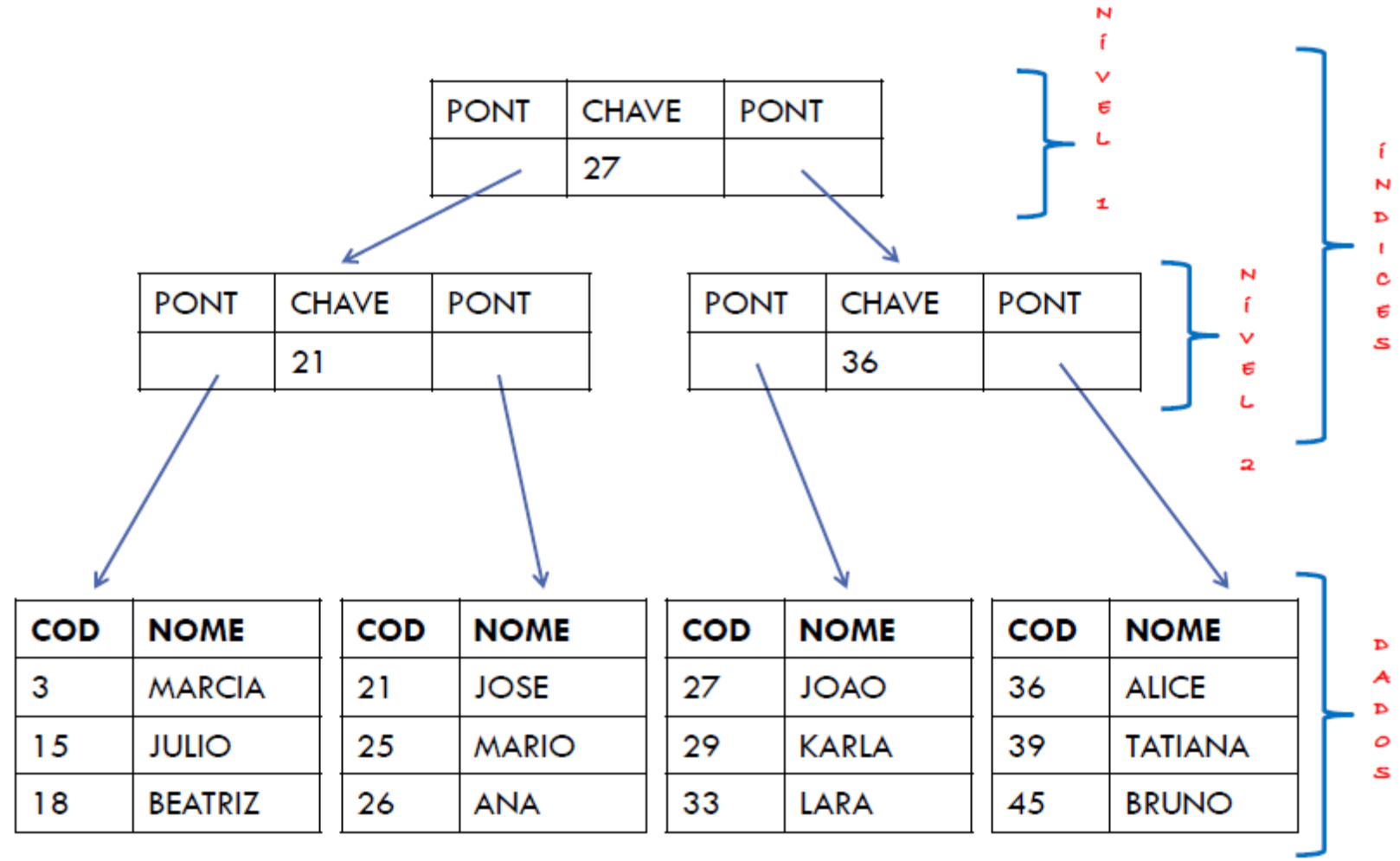
Índice

- 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

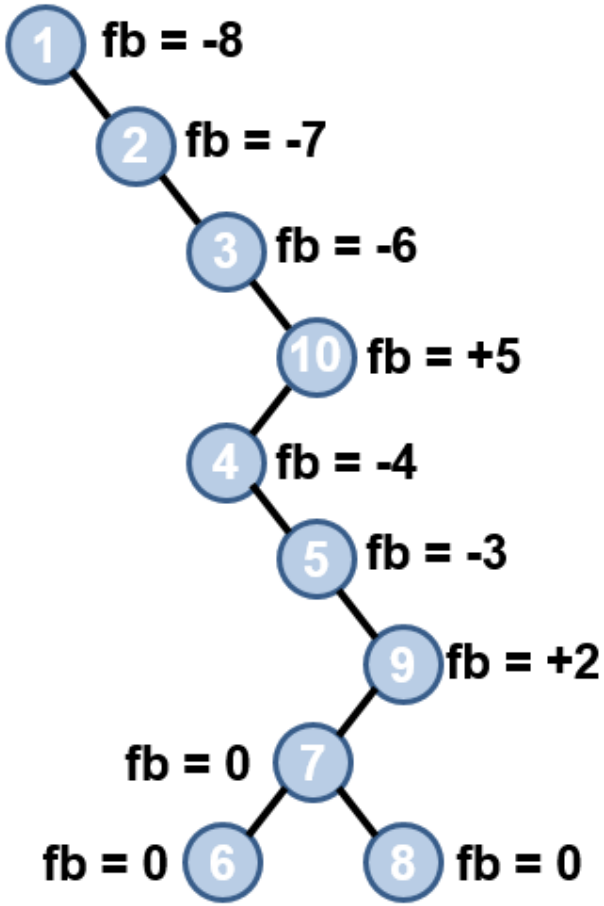
- A maioria das estruturas de índice é implementada por árvores de busca
  - Árvore Binárias de Busca
  - Árvore AVL
  - Árvore Rubro-Negra
  - Árvore de Múltiplos Caminhos

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

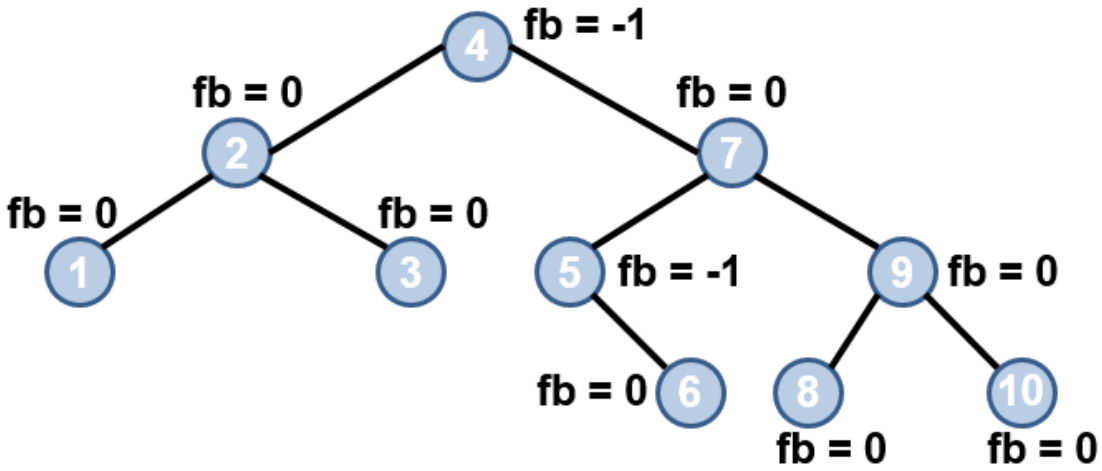
Problema da árvore binária de busca

- 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}



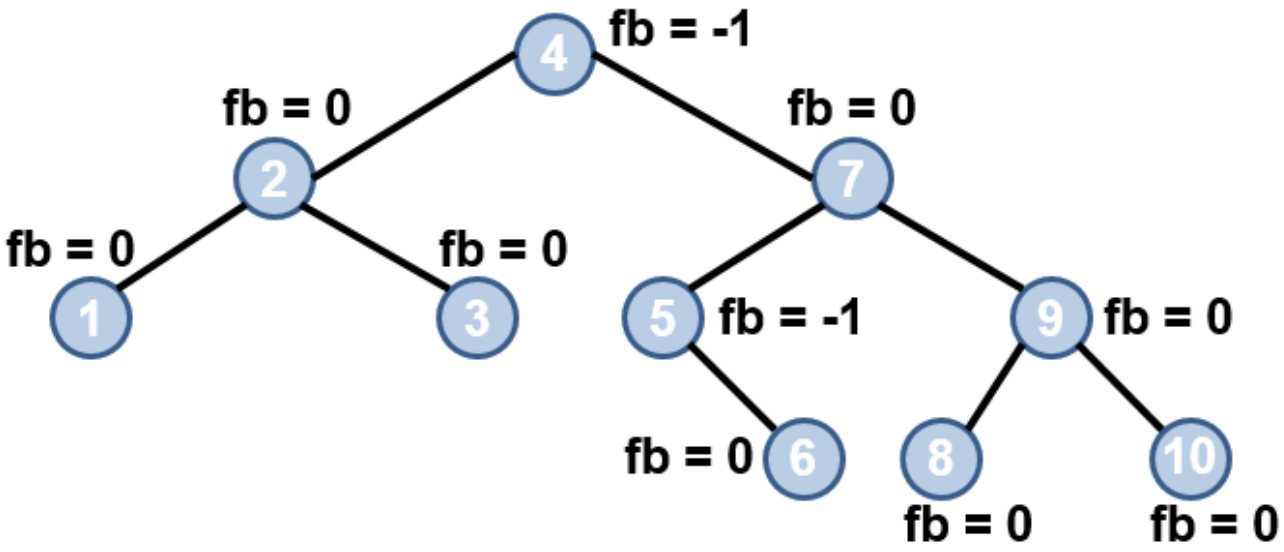
Árvore AVL

- 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

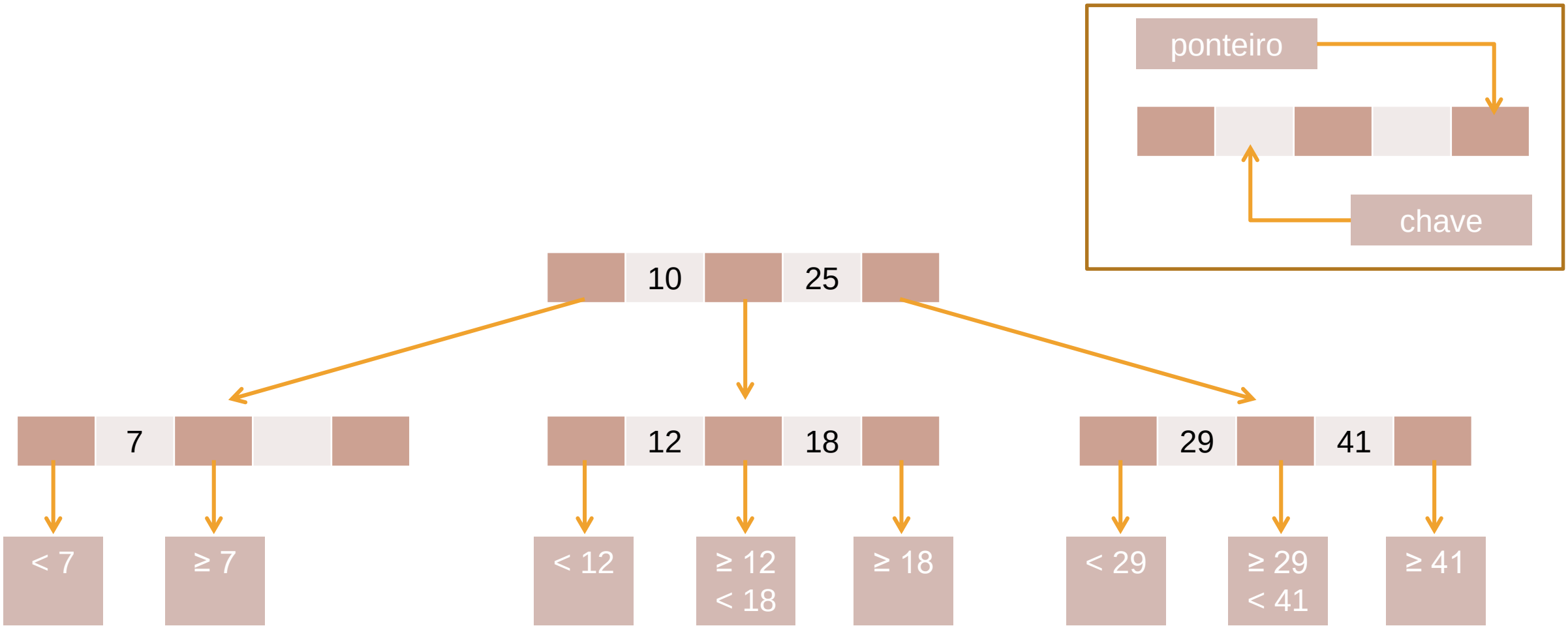
- Apesar de serem árvores binárias de busca balanceadas, ainda são excessivamente altas para uso eficiente como estrutura de índice.
  - Cada nível tem, no máximo, o dobro de elementos do anterior.



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

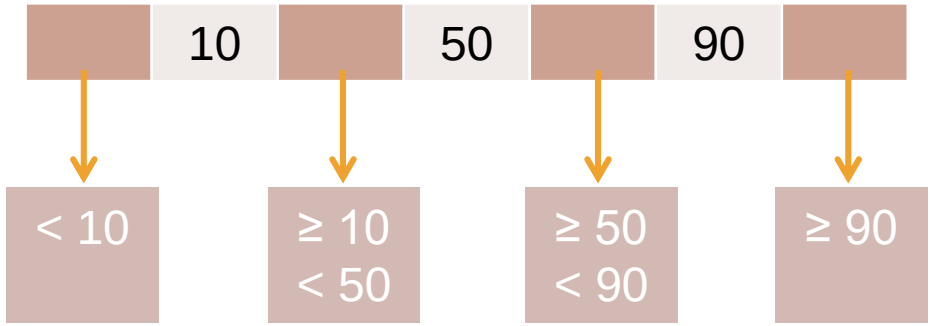
- Especifica precisamente as propriedades que devem estar presentes para uma estrutura de dados ser qualificada como árvore B
  - Direciona a implementação do algoritmo de remoção da árvore B
- Problema
  - Literatura não é uniforme no uso e definição dos termos

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



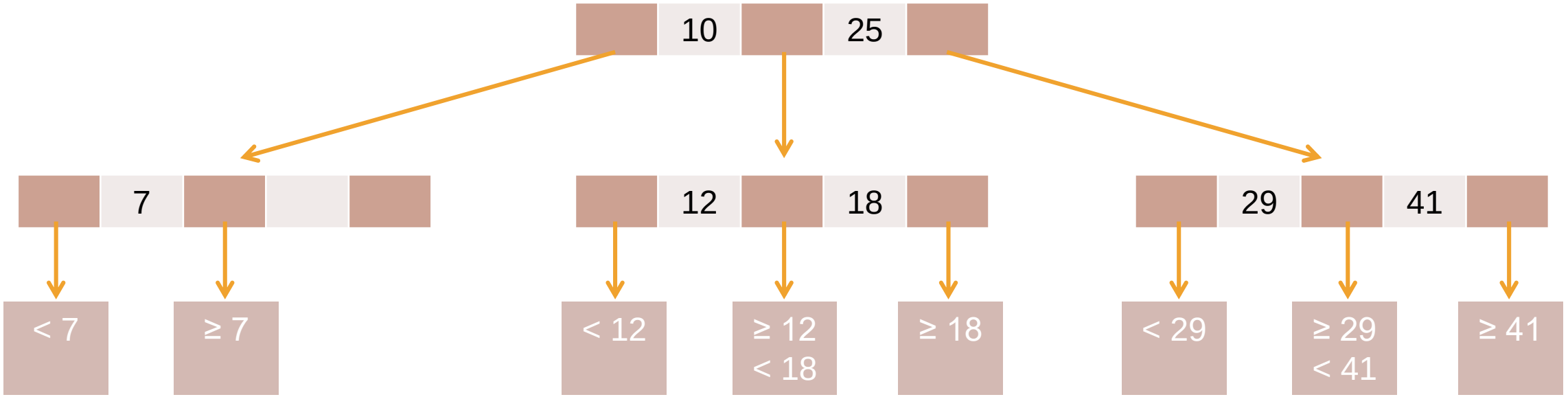
Nó folha

- 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:
  - $N + 1 \geq 2 * \left\lceil \frac{m}{2} \right\rceil^{\mathrm{d}-1}$
  - $d \leq 1 + \log_{\lceil m/2 \rceil}(\frac{N+1}{2})$
- 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

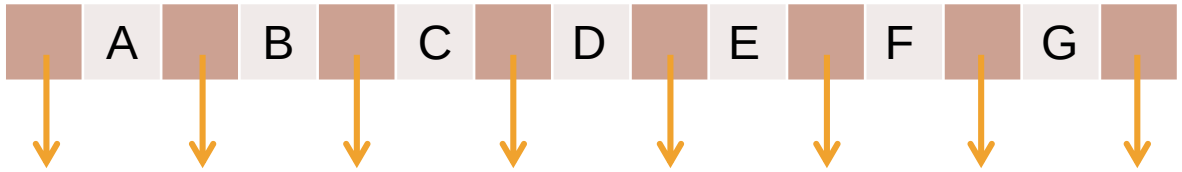
- Característica
  - Sempre realizada nos nós folhas
  - Se chave já existe, a inserção é inválida
- Situações a serem consideradas
  - Árvore vazia
  - Inserção nos nós folhas
  - *Overflow* no nó raiz

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

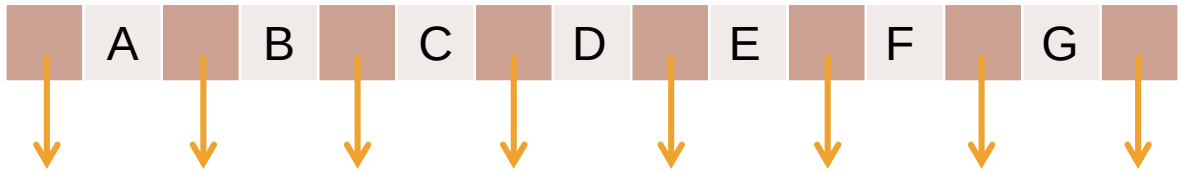


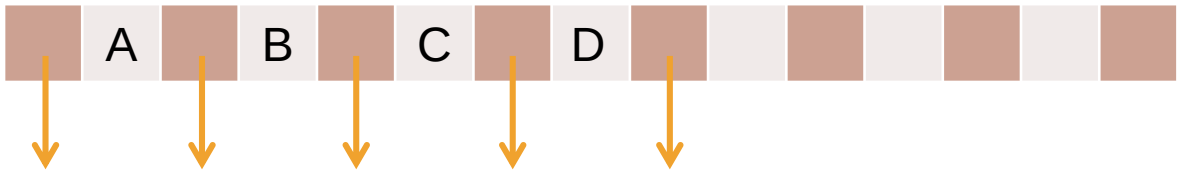
Árvore B | Inserção

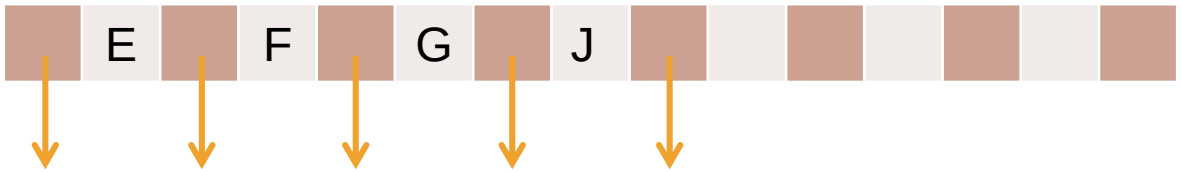
- Overflow no nó raiz
  - Nova chave não cabe no nó raiz (cheio)
- 1º passo: Particionamento do nó (*split*)
  - As chaves são distribuídas uniformemente nos dois nós
  - Realiza a inserção da nova chave

Árvore B | Inserção

- Insere J



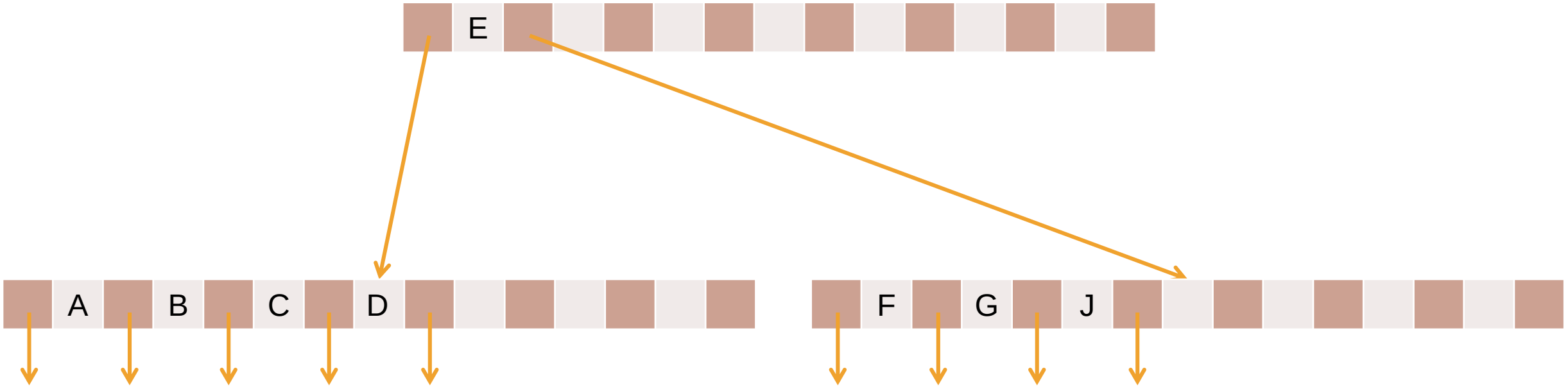




Árvore B | Inserção

- 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

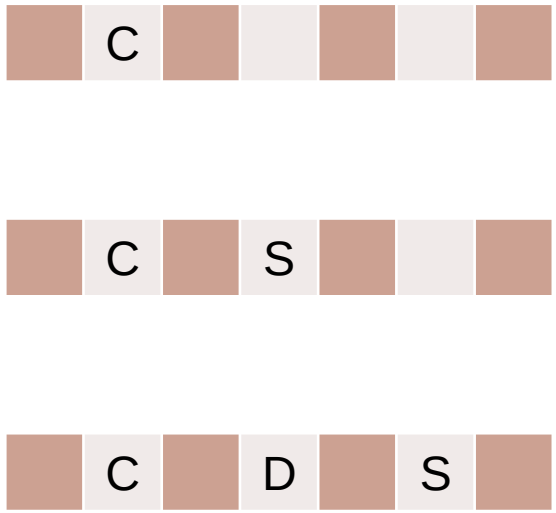
- Particionamento do nó folha
  - Criação de um novo nó
  - Distribuição uniforme das chaves nos dois nós
  - Promoção: escolha da primeira chave do novo nó como chave separadora no nó pai
  - Ajuste do nó pai para apontar para o novo nó
  - Propagação de *overflow*

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**

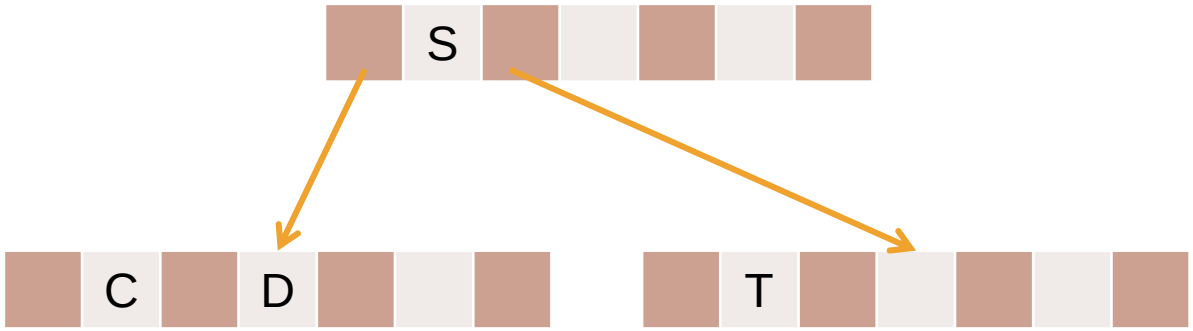
Árvore B | Inserção

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



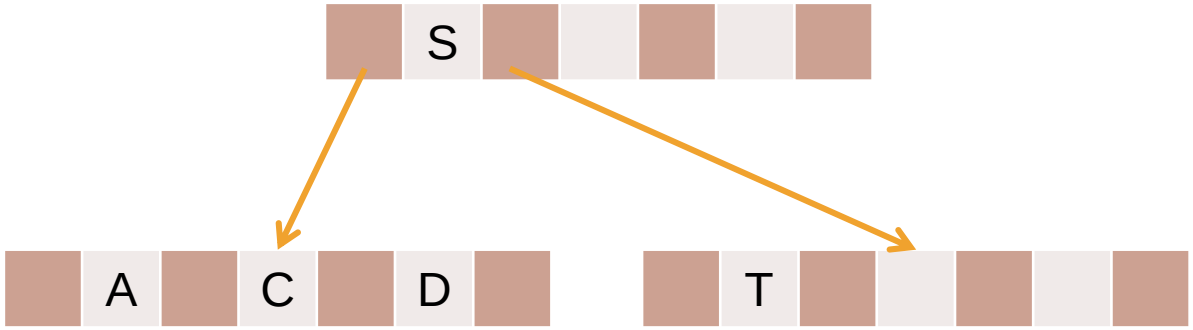
Árvore B | Inserção

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



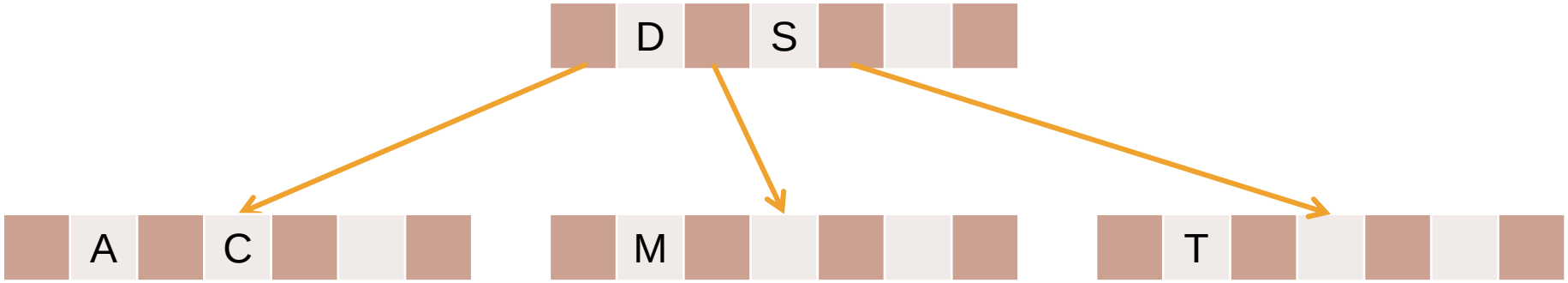
Árvore B | Inserção

- Insere a chave A
  - Há espaço na folha



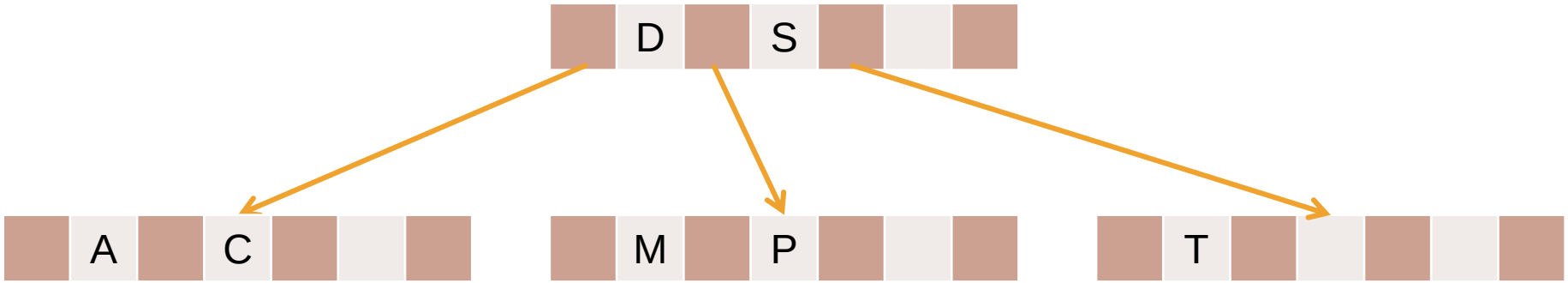
Árvore B | Inserção

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



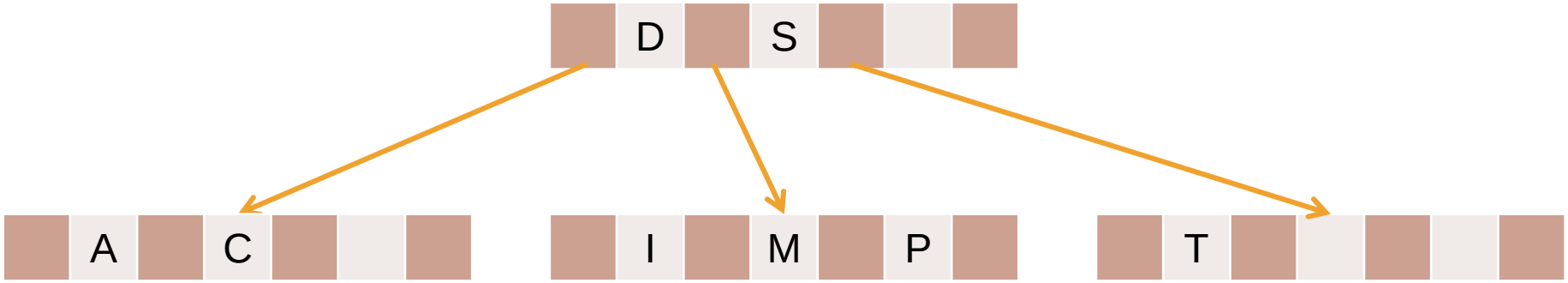
Árvore B | Inserção

- Insere a chave P
  - Há espaço na folha



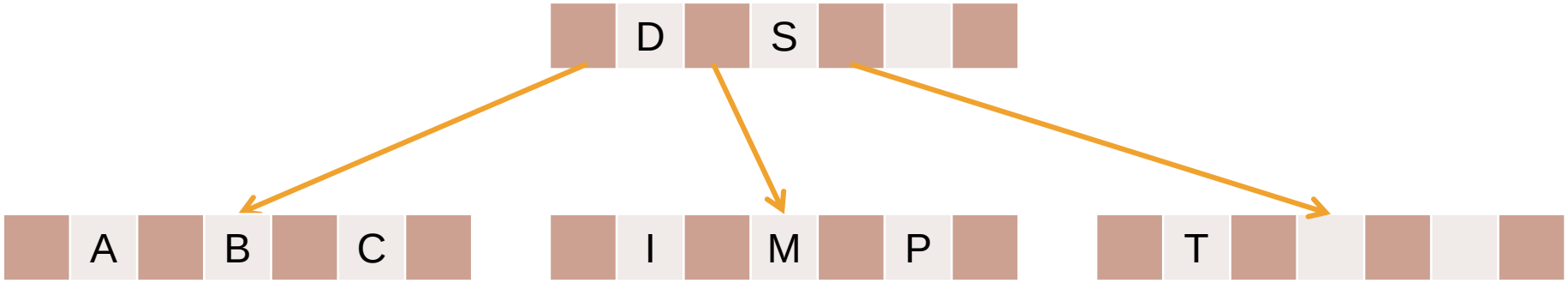
Árvore B | Inserção

- Insere a chave I
  - Há espaço na folha



Árvore B | Inserção

- Insere a chave B
  - Há espaço na folha



Árvore B | Inserção

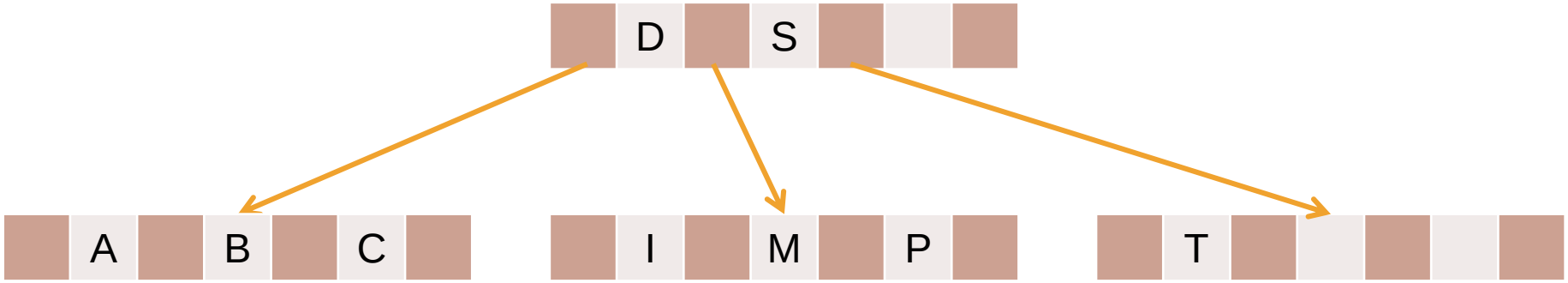
- Sobre o particionamento (ou *split*)
  - Ele se propaga para os pais dos nós, podendo, eventualmente, atingir a raiz da árvore
  - Nesse caso, o nó raiz é particionado normalmente, mas, como a raiz não tem pai, cria-se um novo nó, que passa a ser a nova raiz
  - O particionamento da raiz é a única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

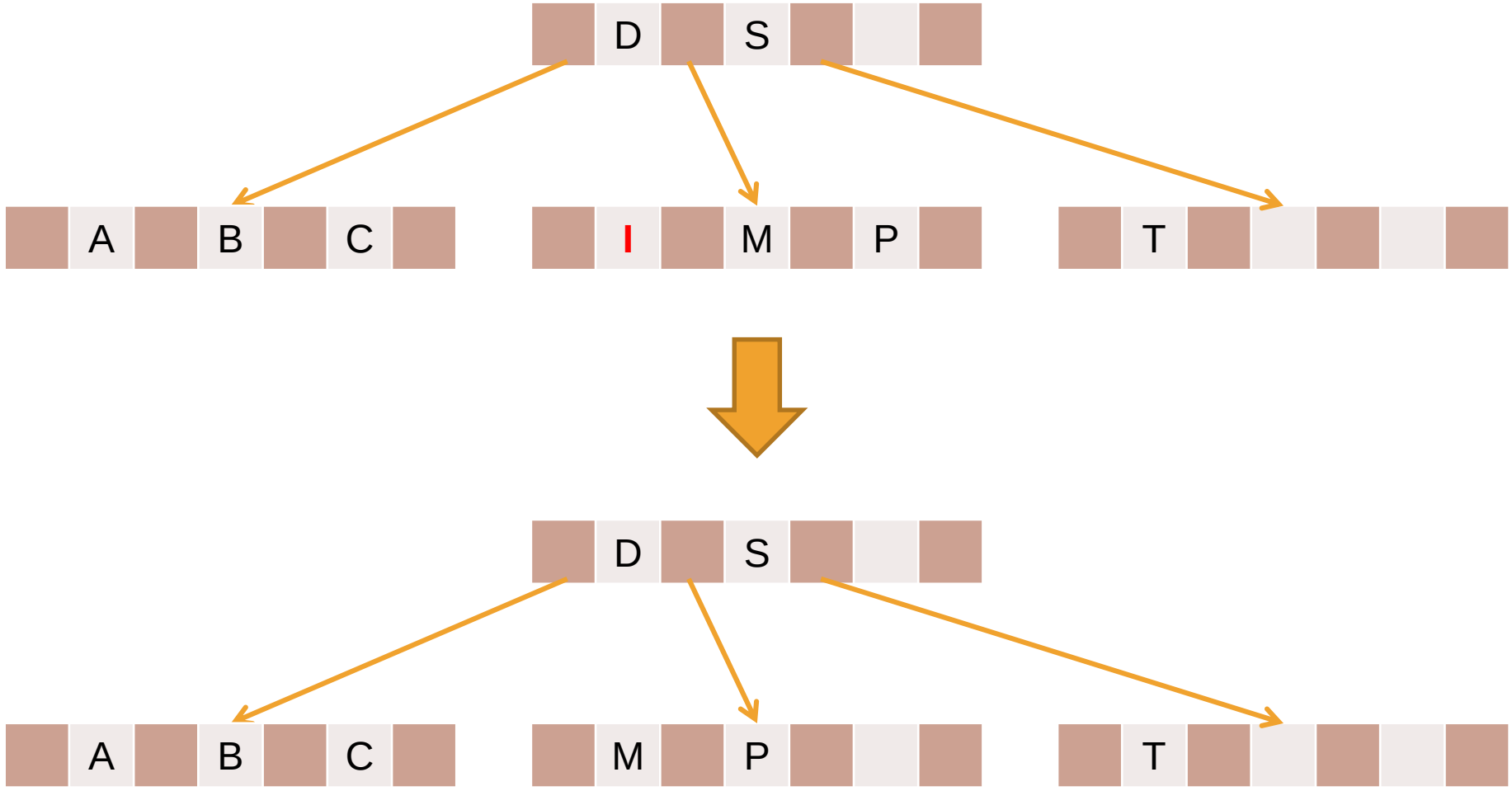


Árvore B | Remoção

- 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

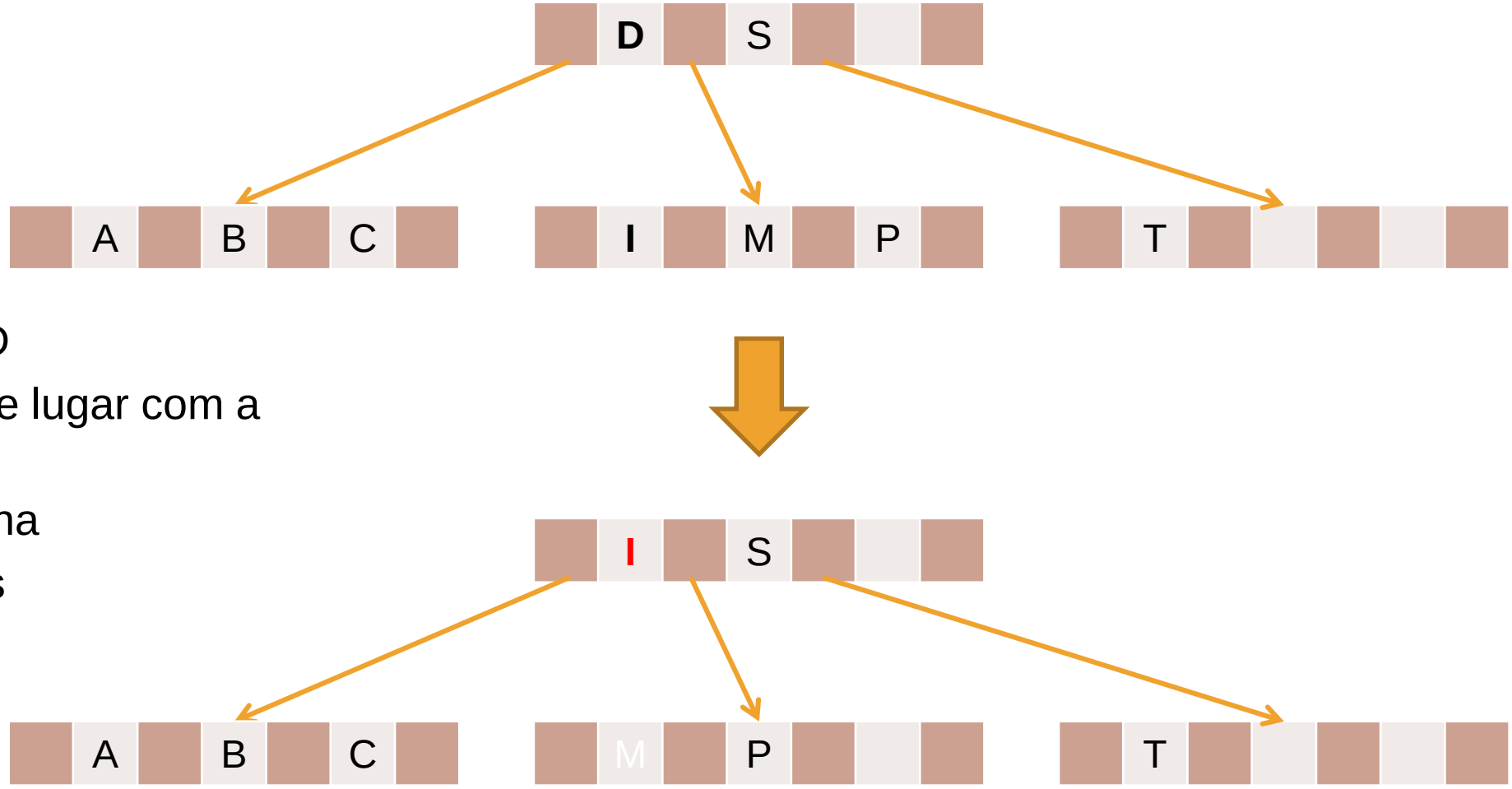
- Remove a chave I



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

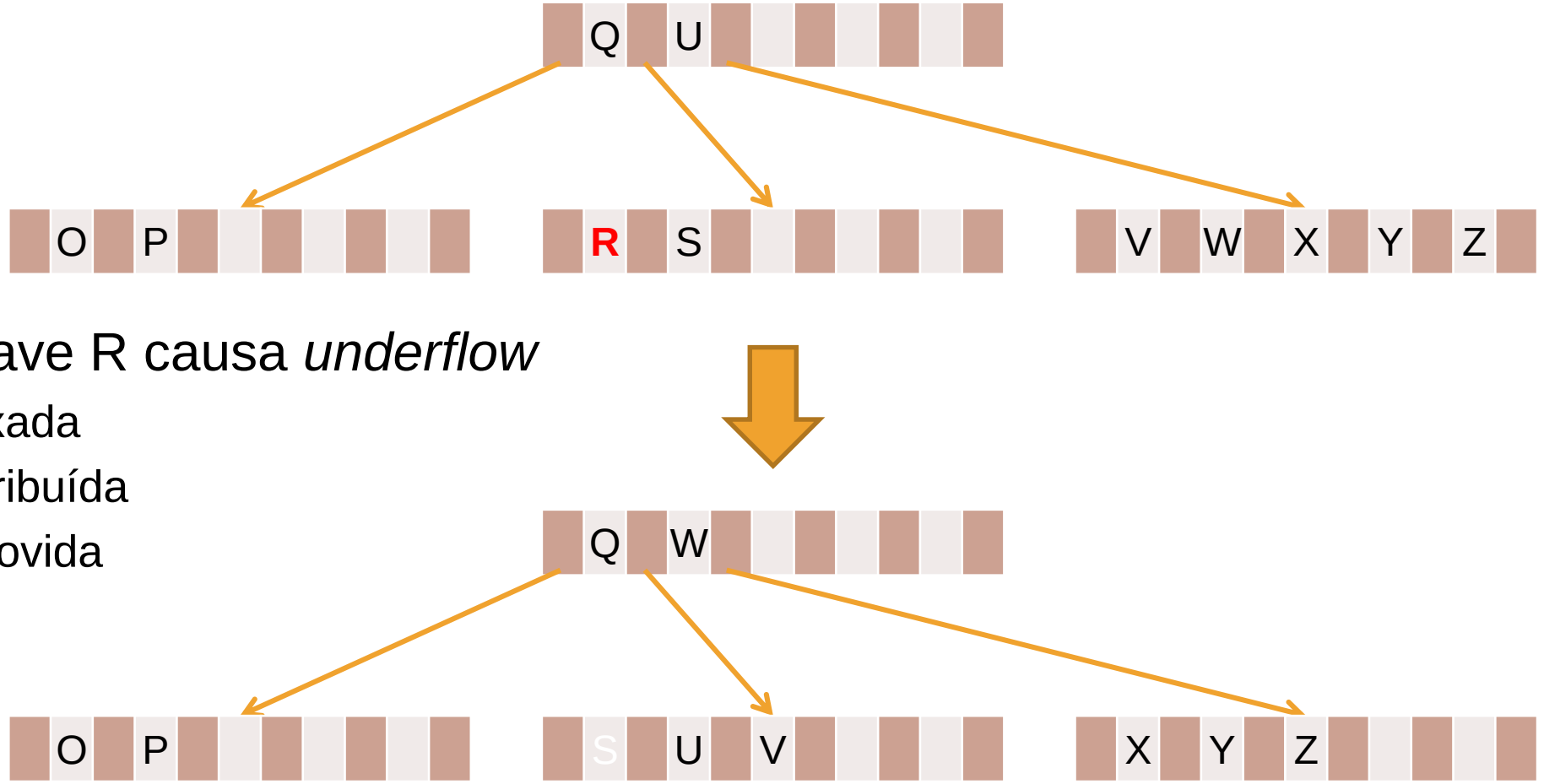


- Remoção da chave D
- Primeiro, trocar ela de lugar com a chave I
- Remover D do nó folha
- Rearranjar as chaves

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

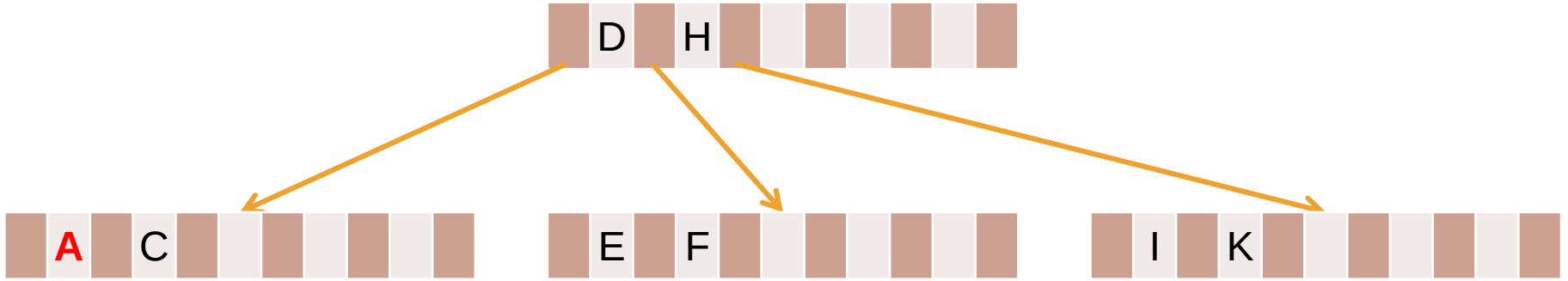
Árvore B | Remoção

- 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

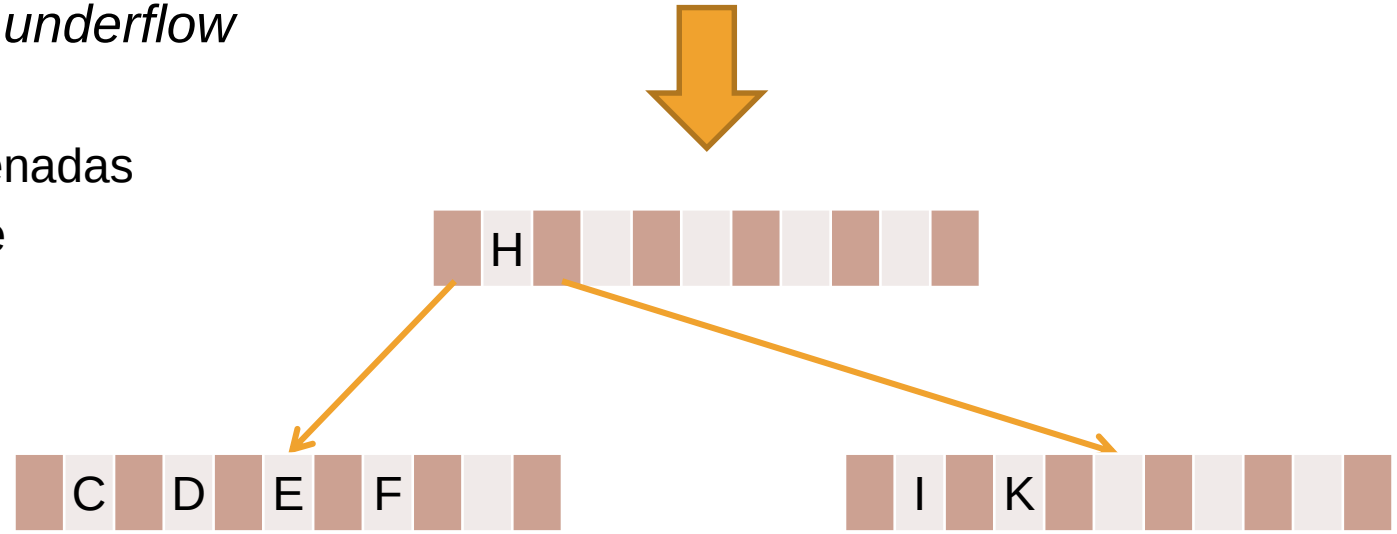
Árvore B | Remoção

- 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

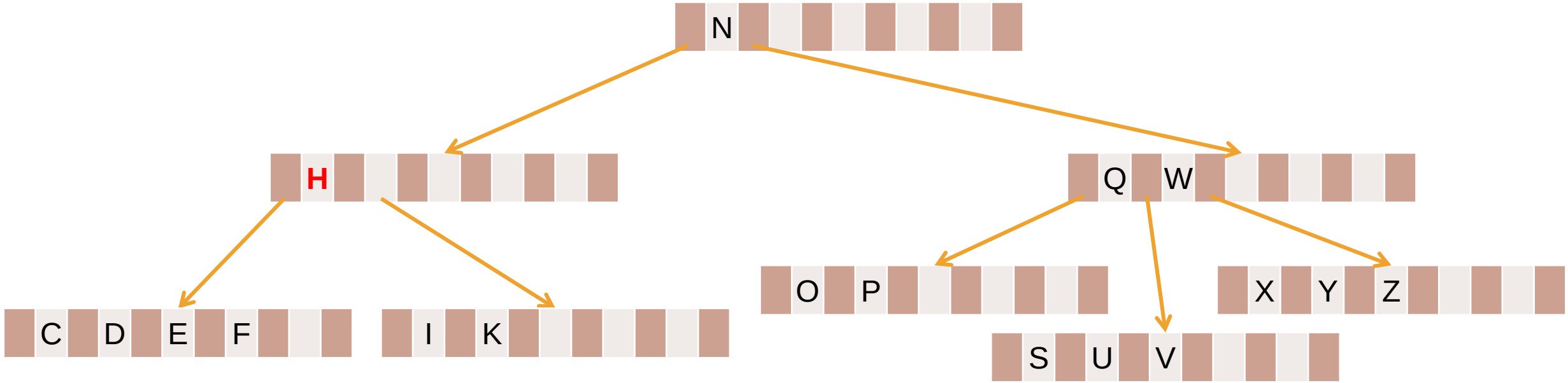


Árvore B | Remoção

- 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

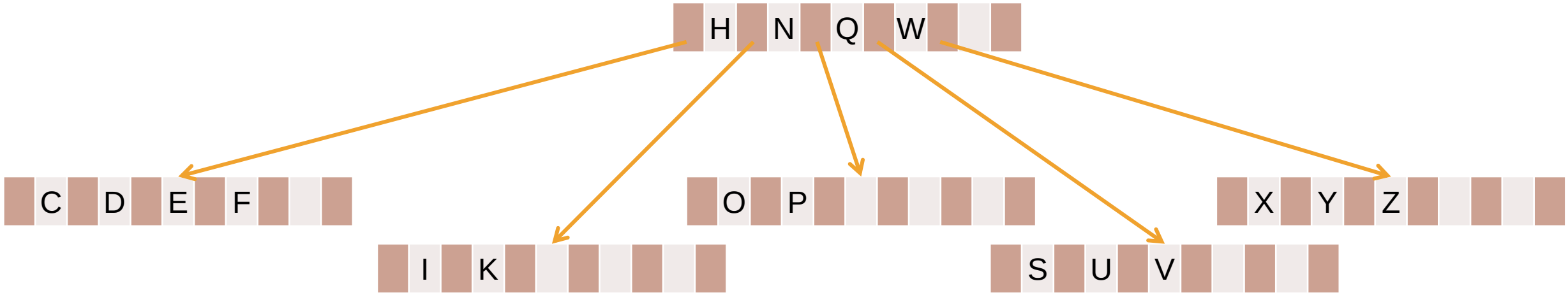
Árvore B | Remoção

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

Árvore B | 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%

VARIAÇÕES DA ÁRVORE B

Árvore B

- Uma árvore B é uma árvore *n*-ária usada como estrutura de armazenamento
  - Muito eficiente e flexível
  - Mantém propriedades de balanceamento mesmo após inserções e remoções
  - Provém busca a qualquer chave com poucos acessos a disco

Árvore B

- Problema com acessos a disco
  - Não é porque uma árvore B tem 3 níveis, que toda busca tenha que fazer 3 acessos a disco
  - Encontrar uma maneira de fazer um uso eficiente de índices que são muito grandes para serem armazenados inteiramente em memória principal (i.e., RAM)
- Objetivo
  - Encontrar uma maneira de diminuir o número médio de acessos a disco para pesquisa

Árvore B

- Diferentes variações foram propostas para aumentar a sua eficiência
  - Árvore B Virtual
  - Árvore B*
  - Árvore B+

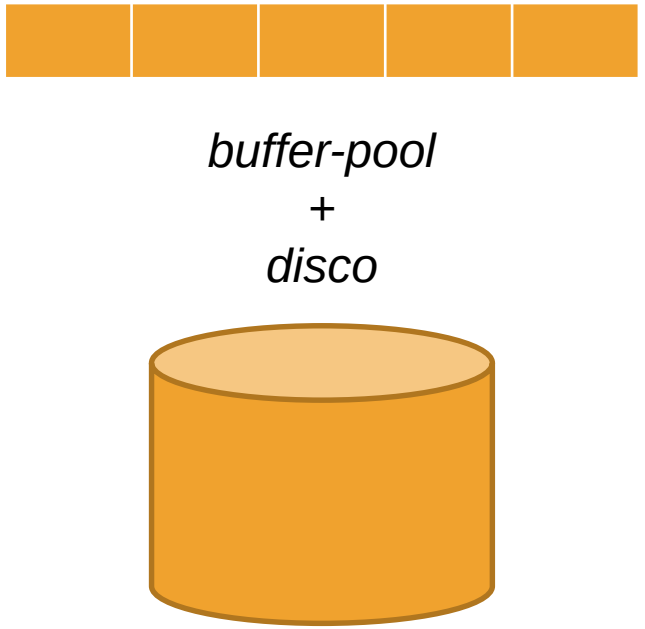
ÁRVORE B VIRTUAL

Árvore B Virtual

- Uma forma de melhorar o desempenho da árvore B
- Busca manter a página raiz da árvore em memória principal
  - Ainda deixa espaço disponível em RAM
  - Diminui o número de acessos a disco em 1 no pior caso

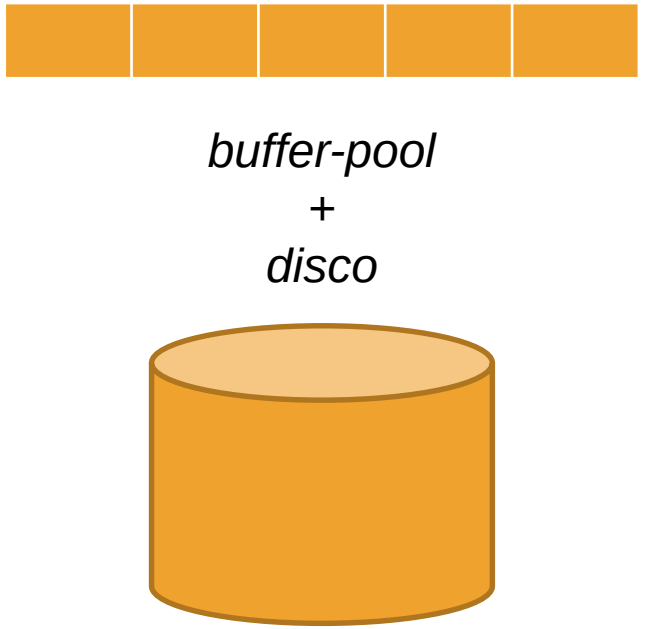
Árvore B Virtual

- Uso de um buffer de páginas (*buffer-pool*) para guardar um certo número de páginas da árvore B
  - Abordagem mais genérica
  - *Buffer-pool* fica em memória principal (i.e., em RAM)



Árvore B Virtual

- Funcionamento da busca
  - Primeiro procura a página no *buffer-pool* para evitar acessos a disco
  - Se a página não está em memória, ela é lida do disco para o *buffer*, substituindo alguma página previamente lida



Árvore B Virtual

- Há necessidade de substituição de páginas
  - ***Page Fault****: p*rocesso de acessar o disco para trazer uma página que não está no *buffer-pool*
- Causas
  - A página nunca foi utilizada
  - A página foi substituída no *buffer-pool* por outra página
- Decisão crítica
  - Qual página deve ser substituída no *buffer*, quando este encontra-se cheio?

Árvore B Virtual

- Necessidade de gerenciamento do buffer
  - A decisão mais crítica é qual página substituir quando o *buffer* se encontra cheio
- Quais páginas manter no Buffer?
  - Apenas a raiz
  - Política LRU (*least recently used*)
  - Substituição baseada na altura da página (*page height*)

LRU (*least recently used*)

- Uma estratégia bastante comum é substituir pela página menos recentemente usada
  - Substitui a página que foi acessada menos recentemente, isto é, a página que ficou mais tempo sem ser requisitada para uso
  - Baseia-se no fato de que é mais comum necessitar de uma página que foi recentemente usada do que de uma página que foi usada a mais tempo

LRU (*least recently used*)

- Página menos recentemente usada é diferente de substituir pela página menos recentemente lida
  - O tempo é dado pela último uso da página, e não o momento em que ela foi lida
- Localidade temporal
  - Assume algum tipo de agrupamento no uso das páginas ao longo do tempo
  - hipótese pode não ser sempre válida, mas aplica-se bem nas árvores B

Substituição baseada na altura da página

- Modo mais direto que a estratégia LRU para guiar as decisões de substituição de página no buffer
  - Mantém as páginas que estão nos níveis mais altos da árvore (i.e., próximas à raiz)
  - Utiliza a política LRU para as demais páginas (i.e., páginas mais utilizadas)

ÁRVORE B*

Árvore B*

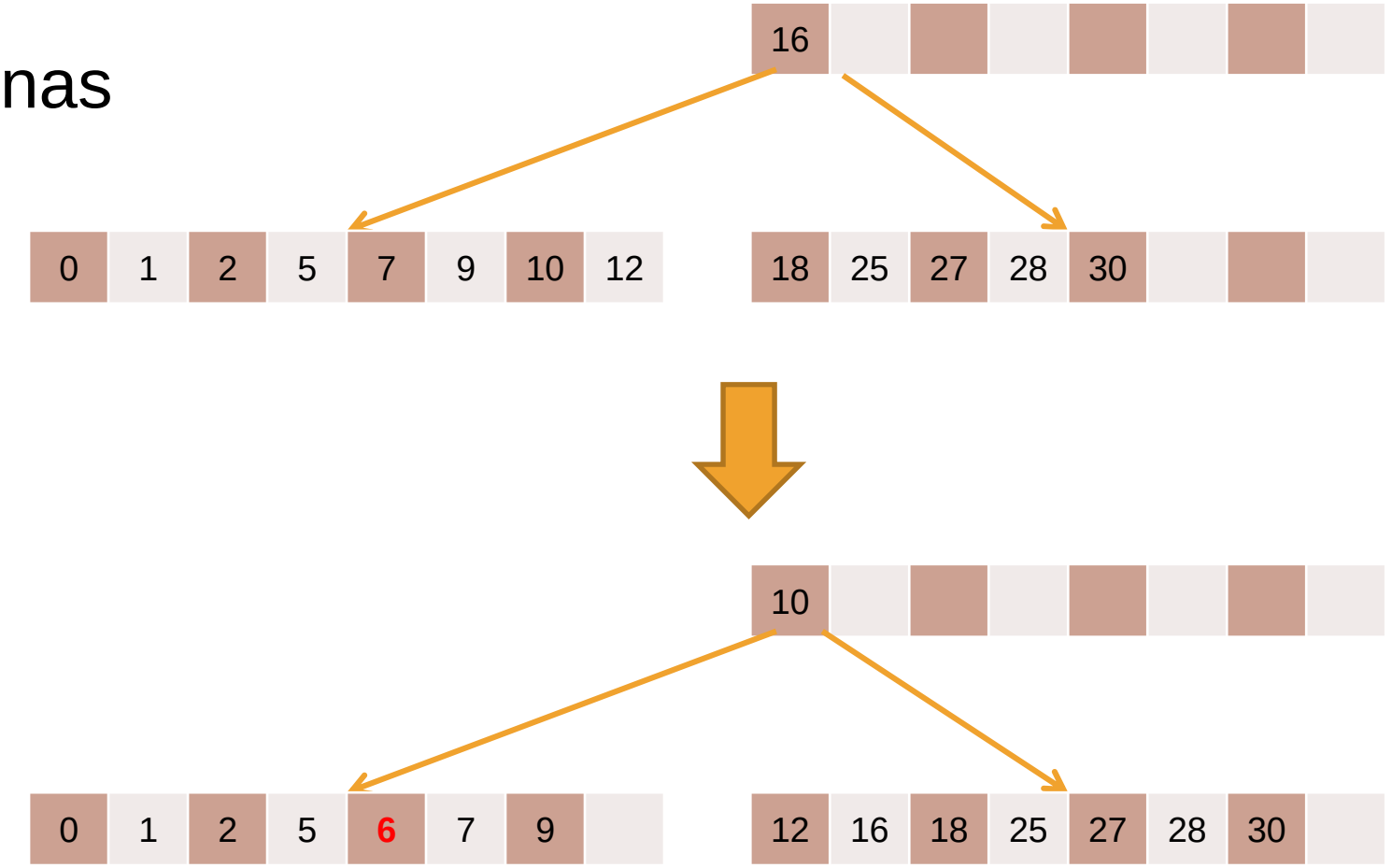
- Proposta por Knuth em 1973 como uma variação de árvore B
- Característica
  - Cada nó contém, no mínimo, ***2/3*** do número máximo de chaves
  - Na árvore B de ordem **m**, o número mínimo de chaves é $\lceil m/2 \rceil - 1$
  - Geração da árvore é feita por um processo de redistribuição e subdivisão

Árvore B*

- Postergar divisão de páginas (*split*) através da redistribuição na inserção
  - Estende a noção de redistribuição durante a inserção para incluir novas regras para o particionamento de nós
  - A subdivisão é adiada até que duas páginas irmãs estejam cheias

Árvore B*

- Insere a chave **6**
- Postergar divisão de páginas através da redistribuição

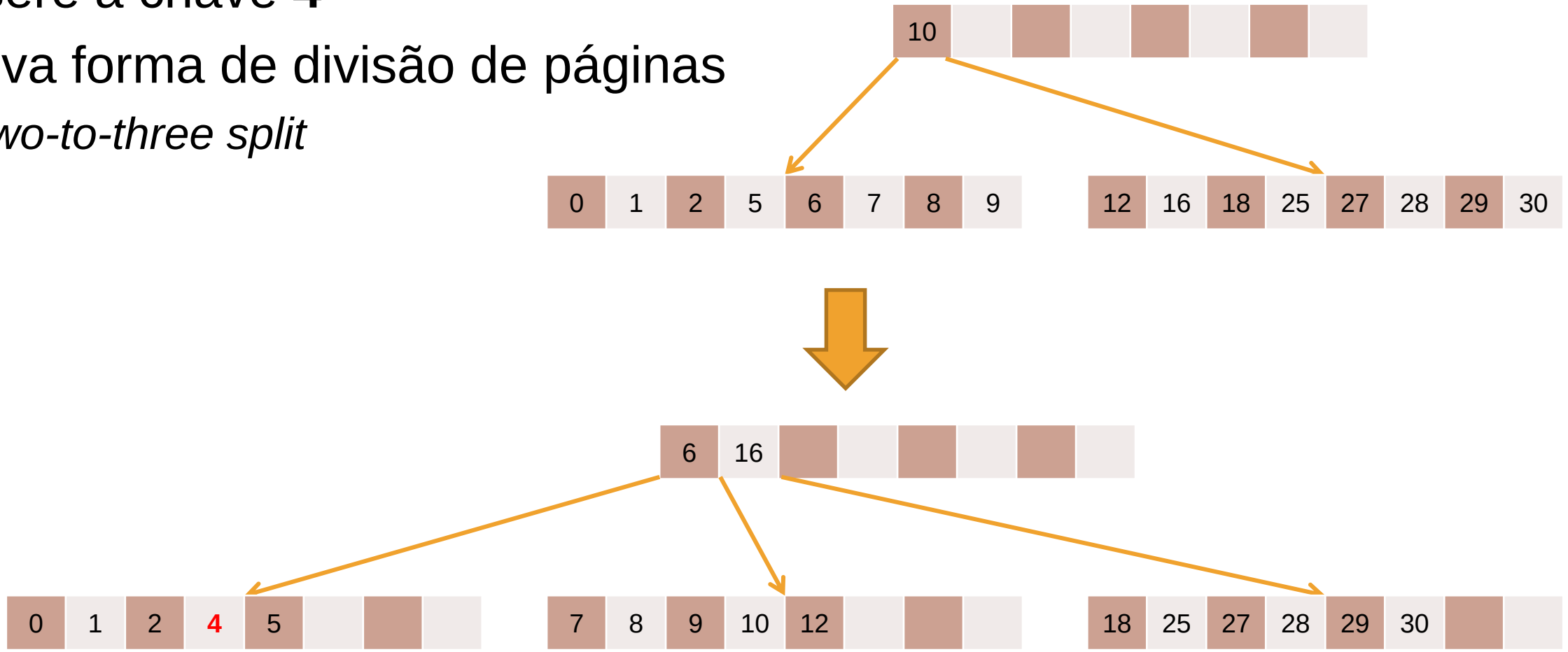


Árvore B*

- Nova forma de divisão de páginas, de modo a garantir a ocupação mínima
  - Divisão do conteúdo de duas páginas irmãs em três páginas
  - *two-to-three split*
- Funcionamento
  - Árvore B: split 1-to-2
  - Árvore B*: split 2-to-3, pelo menos um nó irmão está cheio

Árvore B*

- Insere a chave **4**
- Nova forma de divisão de páginas
  - *two-to-three split*



Árvore B*

- Mudança na taxa de ocupação afeta as rotinas de remoção e redistribuição
- Particionamento da raiz é um problema
  - Raiz não possui nó irmão para fazer divisão 2 para 3 páginas
  - Se dividir, os filhos da raiz não terão a taxa de ocupação mínima de 2/3

Árvore B*

- Soluções possíveis
- Permitir que a raiz seja maior (tamanho de página diferente)
  - Assim, quando dividir, terá 2 filhos com 2/3 de ocupação
- Fazer uma divisão convencional na raiz
  - Dividir a raiz usando a divisão 1-to-2 split
  - Permitir os filhos da raiz com uma taxa de ocupação menor que 2/3, como uma exceção

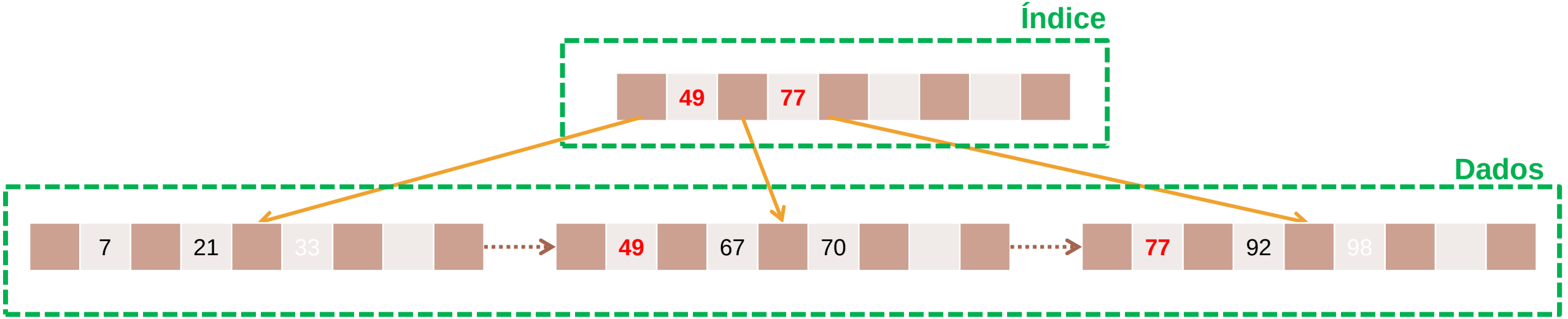
ÁRVORE B+

Árvore B+

- Semelhante à árvore B, exceto por duas características muito importantes:
  - Armazena dados somente nas folhas – os nós internos servem apenas de ponteiros
  - As folhas são encadeadas. Cada página folha aponta para sua próxima, para permitir acesso sequencial

Árvore B+

- Isso permite o armazenamento dos dados em um arquivo, e do índice em outro arquivo separado
  - Diferente do que acontece nas árvores B

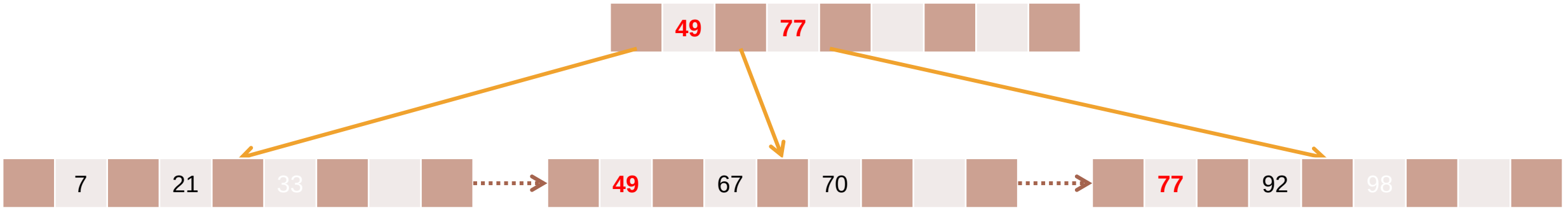


Árvore B+

- 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, MySQL e SQLite permitem o uso deste tipo de árvore para indexar tabelas
  - Outros sistemas de gerência de dados como o CouchDB, Tokyo Cabinet e Tokyo Tyrant permitem o uso deste tipo de árvore para acesso a dados

Árvore B+ | Busca

- Só se pode ter certeza de que o registro foi encontrado quando se chega em uma folha
  - Não existem dados nos nós intermediários, apenas nas folhas
  - Os dados dá chave **77** estão na folha, não no nó raiz
- As comparações agora não são apenas >, mas ≥
  - Índices **repetem valores** de chave que aparecem nas folhas

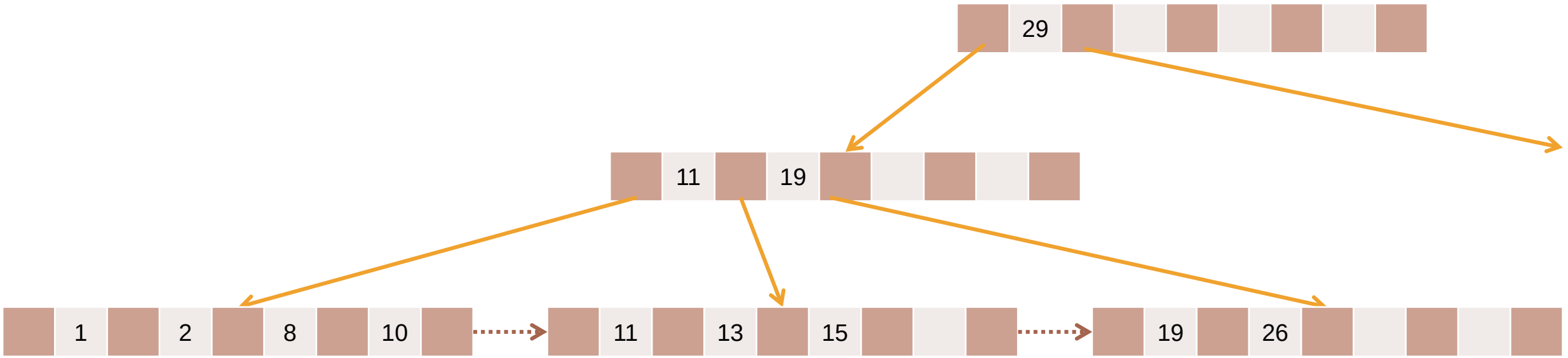


Árvore B+ | Inserção

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

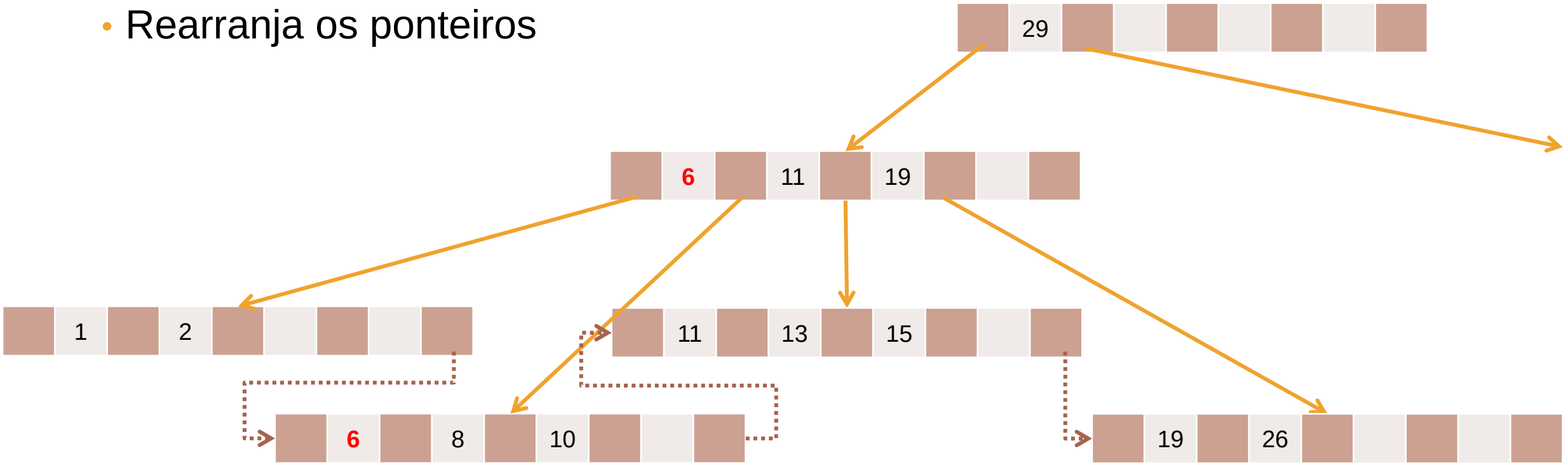
Árvore B+ | Inserção

- Insere a chave 6
  - Nó está cheio
  - Necessário particionar o nó



Árvore B+ | Inserção

- Insere a chave 6
  - Divide o nó, igual árvore B
  - Nova chave fica no novo nó
  - Rearranja os ponteiros



Árvore B+ | Remoção

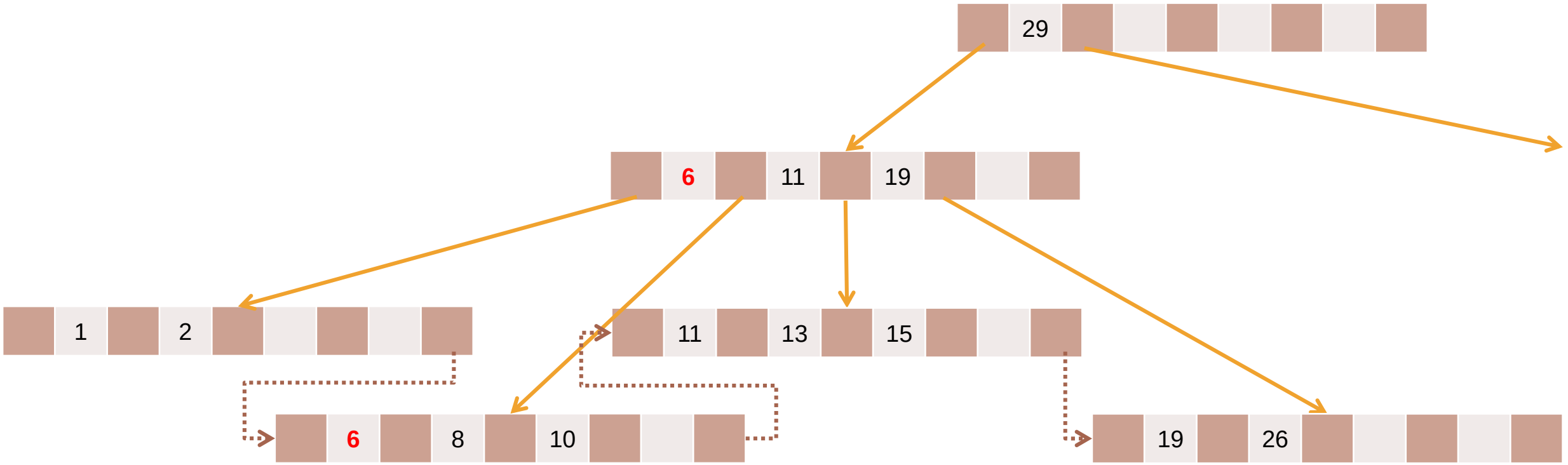
- A remoção ocorre apenas no nó folha
  - Similar a árvore B
  - Pode ser necessário fazer concatenação dos nós ou redistribuição das chaves
- As chaves excluídas continuam nos nós intermediários

Árvore B+ | Remoção

- Remoções que causem concatenação de folhas podem se propagar para os nós internos da árvore
  - 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
- Remoções que causem redistribuição dos registros nas folhas provocam mudanças no conteúdo do índice, mas não na estrutura (não se propagam)

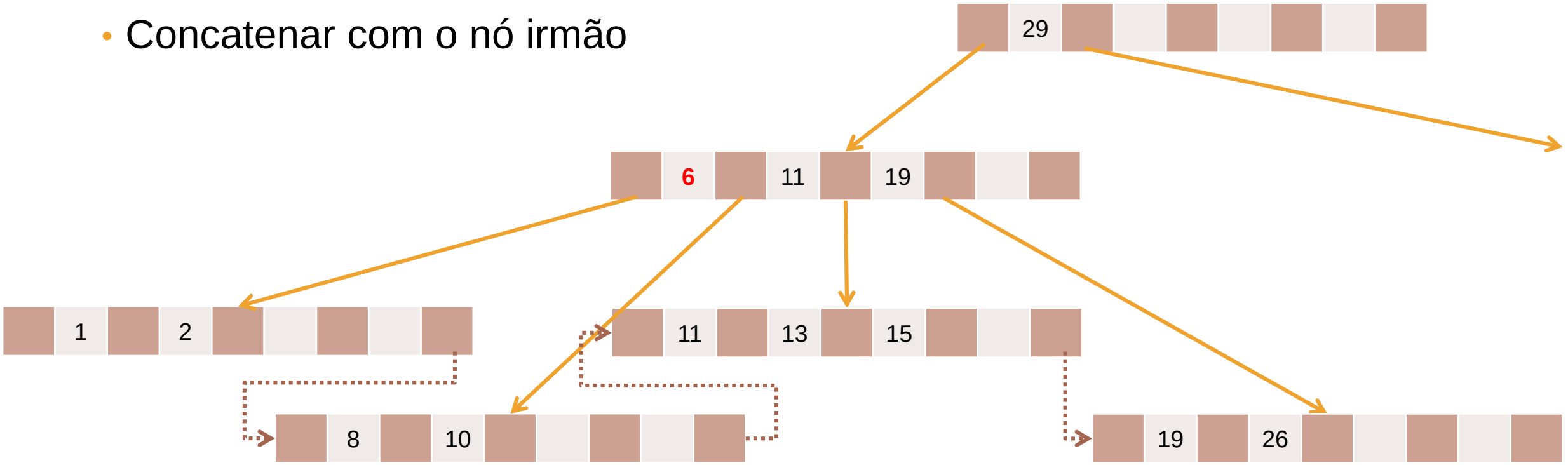
Árvore B+ | Remoção

- Remove a chave 6
  - Já está na folha
  - Apenas remover a chave da folha



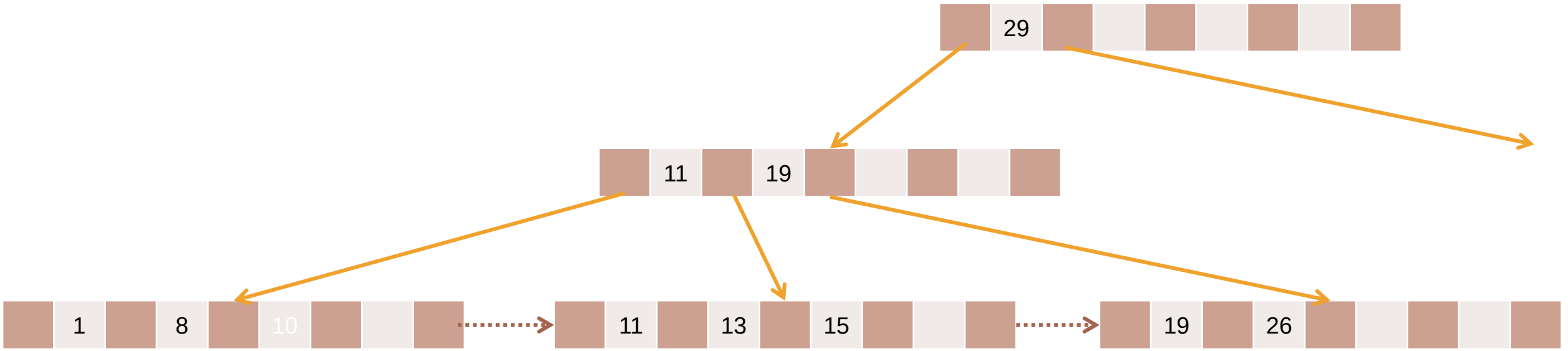
Árvore B+ | Remoção

- Resultado da remoção da chave 6
- Remove a chave 2
  - Remoção causa *underflow*
  - Concatenar com o nó irmão



Árvore B+ | Remoção

- Resultado da remoção da chave 2



ÁRVORE B+ DE PREFIXO SIMPLES (OU PRÉ-FIXADA)

Acessando um arquivo

- Basicamente, podemos acessar um arquivo de duas maneiras:
  - Acesso indexado: arquivo é um conjunto de registros que são indexados por uma chave
  - Acesso sequencial: arquivo é acessado sequencialmente (i.e., registros fisicamente contínuos)

- Queremos que os arquivos suportem acesso indexado eficiente, e também acesso sequencial

Acessando um arquivo

- Arquivo indexado por um índice árvore-B
  - Acesso indexado pela chave
    - Desempenho excelente
    - Ordem logarítmica
  - Acesso sequencial aos registros ordenados pela chave
    - Desempenho péssimo
    - Ordem linear

Acessando um arquivo

- Arquivo com registros ordenados pela chave
  - Processamento sequencial (acessar todos registros)
    - Desempenho apropriado
    - Buferização
  - Processamento randômico
    - Desempenho inapropriado
    - Logarítmico (ordem 2)

Acessando um arquivo

- Solução: usar um modelo híbrido
  - Organizar um arquivo de modo que seja eficiente tanto para processamento sequencial quanto aleatório

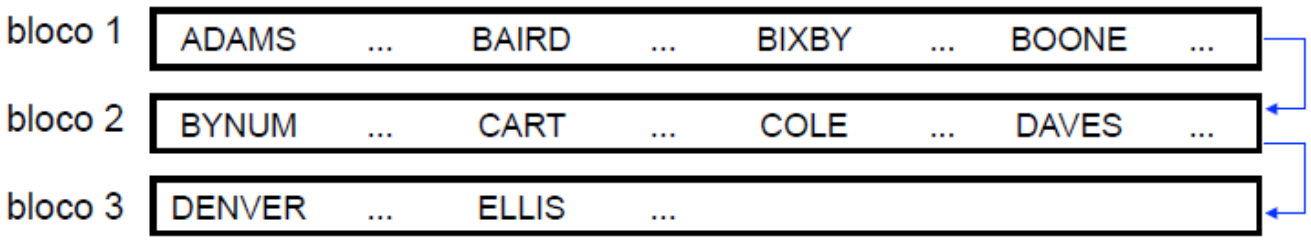
- Estrutura híbrida
  - Chaves: organizadas como árvore B (i.e., *index set*)
  - Nós folhas: consistem em blocos de dados (*sequence set*)

Árvore B+ de prefixo simples

- Variação da árvore B+ que utiliza um prefixo comum para armazenar e pesquisar chaves de forma mais eficiente
  - Armazena na árvore as cadeias separadoras mínimas entre cada par de blocos
  - Se as chaves forem ***strings***, um prefixo comum pode ser uma sequência de caracteres iniciais compartilhados entre as chaves
  - Usar separadores mínimos faz com que os nós possam ser maiores
  - Necessidade de maior controle do tamanho do nó e de onde começa e termina cada cadeia separadora

Árvore B+ de prefixo simples

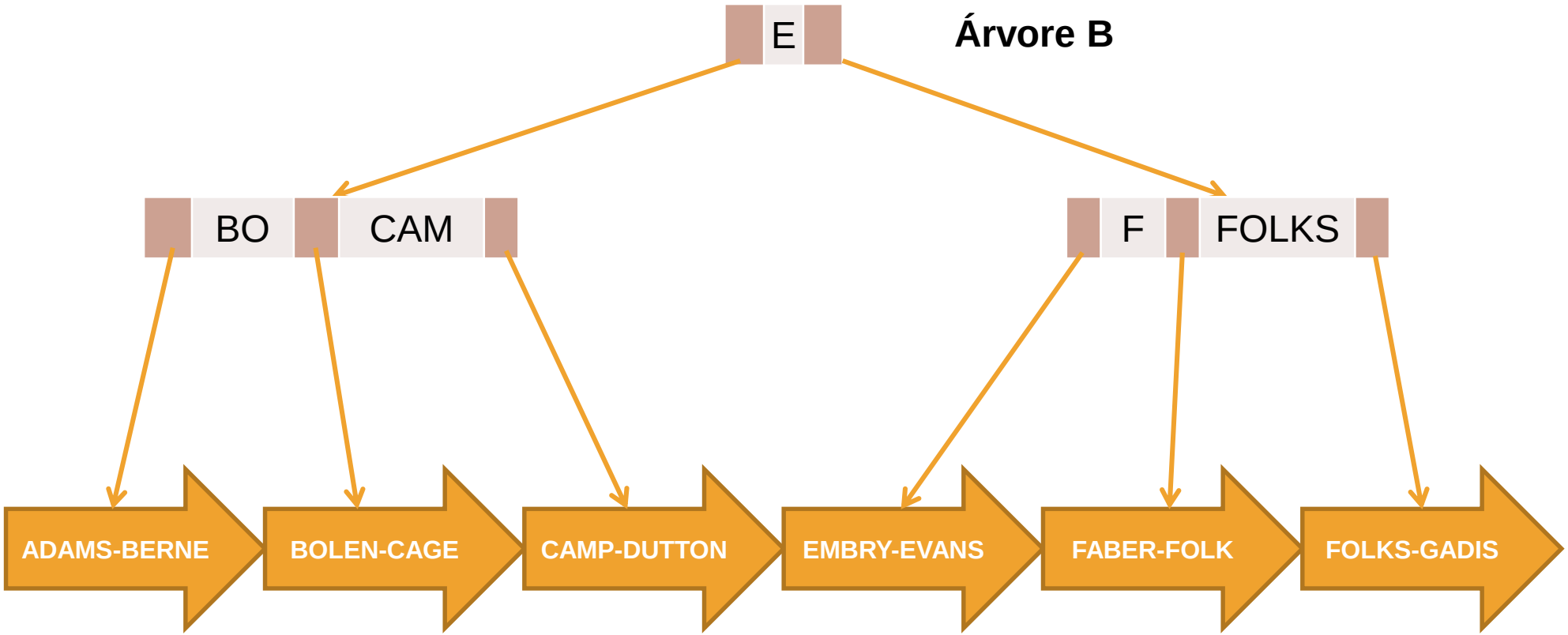
- *Sequence Set*
  - **A**rquivo de dados é organizado em blocos de tamanho fixo, de registros sequenciais, ordenados pelas chaves, e encadeados
  - Privilegia o acesso sequencial do arquivo



Árvore B+ de prefixo simples

- Arquivo de índices (*index set*) é organizado como uma Árvore B
  - As folhas são os blocos de registros sequenciais
  - Privilegia a busca aleatória no arquivo
  - Páginas não folhas contêm chaves ou partes de chaves separadoras para os filhos

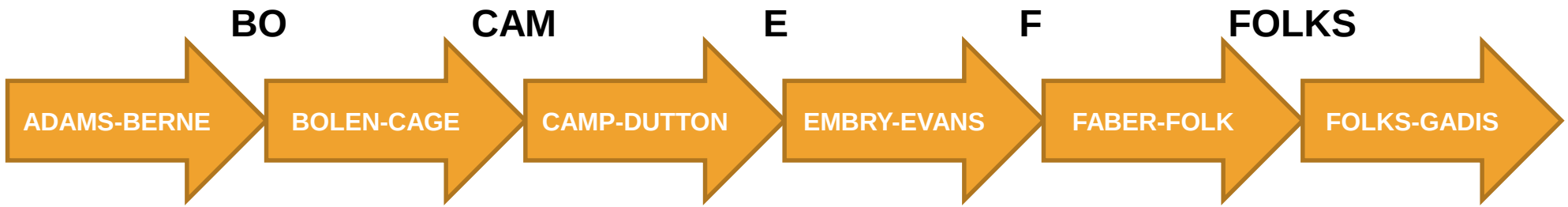
Árvore B+ de prefixo simples



Blocos de registros de dados, ordenados pelas chaves

Árvore B+ de prefixo simples

- Uso de separadores de blocos no lugar das chaves de busca
  - Os separadores são mantidos no índice, ao invés das chaves de busca
  - Escolhemos o menor separador possível
  - Possuem tamanho variável
  - Estruturas semelhantes podem ser consideradas como alternativas
    - Para chaves muito grandes e / ou repetitivas



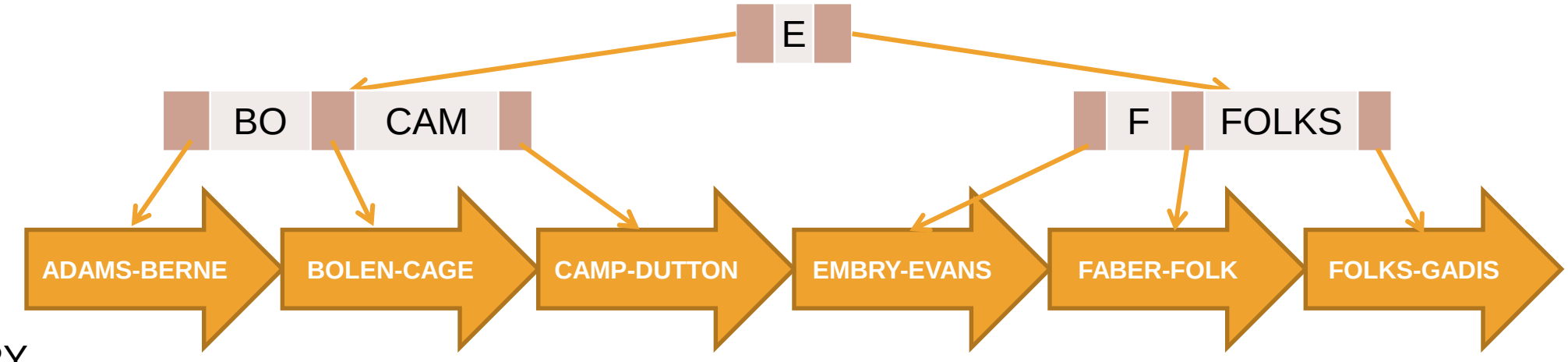
Manutenção

- É importante notar que as operações são realizadas dentro do *Sequence Set*, pois é lá que os registros estão
- Mudanças no índice são consequências das operações fundamentais aplicadas ao *Sequence Set*
  - Adicionamos um novo separador no índice apenas se um novo bloco é formado no *Sequence Set* como consequência de uma operação de divisão
  - Um separador é eliminado do índice apenas se um bloco é removido do *Sequence Set*, como consequência de uma concatenação

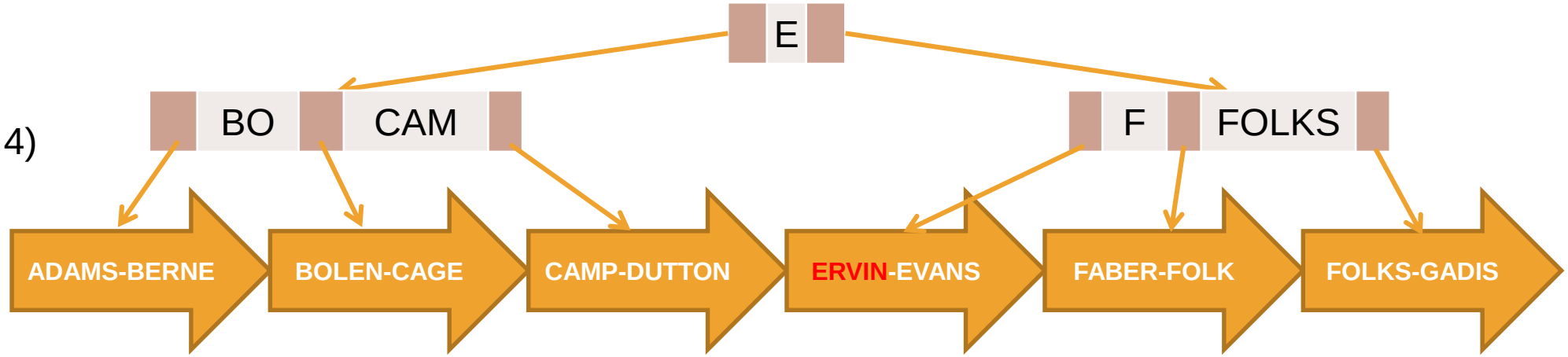
Manutenção

- A ocorrência de *overflow* e *underflow* dos nós do índice não acompanha a ocorrência de *overflow/underflow* no *Sequence Set*
- E uma divisão/concatenação de blocos no *Sequence Set* não resulta necessariamente em uma divisão/concatenação de nós do índice

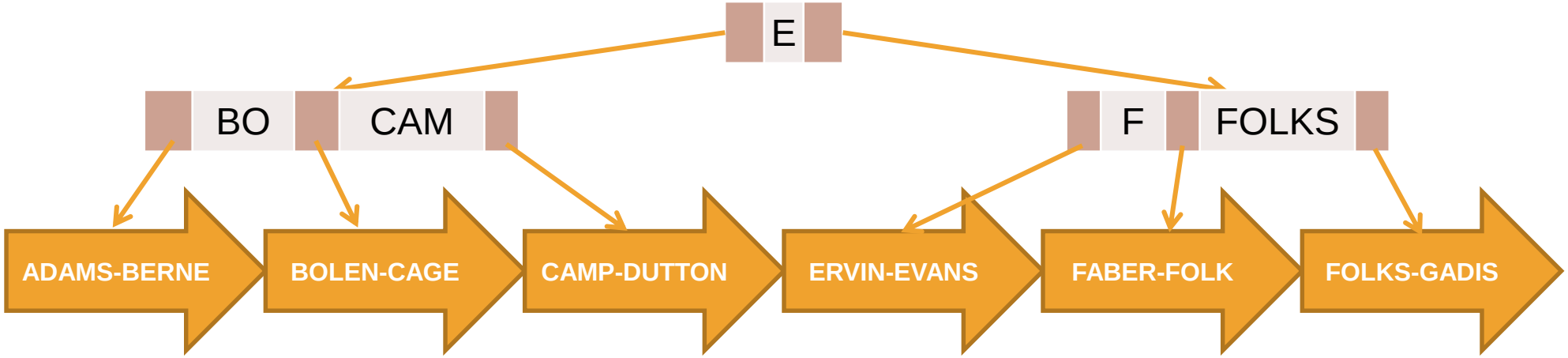
Exemplo: remoção



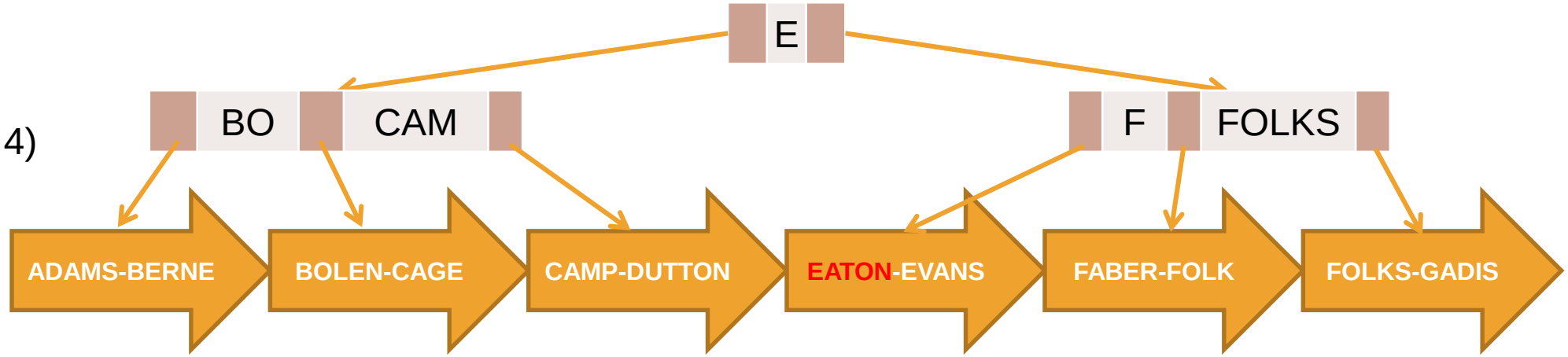
- Remoção de EMBRY
- Sem redistribuição ou concatenação
- Efeito é limitado ao *sequence set* (bloco 4)



Exemplo: inserção



- Remoção de EATON
- Considera espaço disponível no bloco
- Efeito é limitado ao *sequence set* (bloco 4)



Material Complementar | Vídeo Aulas

- Aula 151 – Árvore B
  - https://youtu.be/r6psF2NUMfg
- Aula 152 – Árvore B | Inserção
  - https://youtu.be/lmg_zIWw3RM
- Aula 153 – Árvore B | Remoção
  - https://youtu.be/IYn2D-Ake7U
- Aula 154 – Árvore B Virtual
  - https://youtu.be/3KbA1EgSH5I
- Aula 155 – Árvore B*
  - https://youtu.be/kMlszH2un80

Material Complementar | GitHub

- https://github.com/arbackes