UNISINOS

Estruturas Avançadas de Dados I

(B-Tree e B^+-Tree)

Prof. Gilberto Irajá Müller

Última atualização 28/08/2018

Motivação

- Toda a análise e discussão sobre árvores BST e AVL assume que tais árvores estejam todas em **memória**;
- Quando o assunto é banco de dados, não é realístico pensar neste cenário, pois os dados são exponencialmente maiores que a capacidade de memória;
- Um pequeno exemplo:
  - 10 milhões de registros $\cong$ **$10 * 2^{20}$ registros**
  - Suponha que cada registro tenha um total de 512 bytes = **2^9**
  - Total necessário seria: $10 * 2^{20} * 2^9 \cong$ **5 Gbytes**
- Imagine apenas 1 usuário de um total de 20, sendo que a aplicação destina 1/20 da memória de um total de 32GB RAM = 1,6GB < 5GB.

Motivação (cont.)

- Vamos assumir agora que o **número de acessos** seja igual a N = 10 milhões (**$10 * 2^{20}$**):
  - BST (pior caso) -> **O(N)** = 10 milhões de acessos
  - BST (caso médio) -> **O(1.38 * log(N))** ≅ 32 acessos
  - AVL (melhor caso) -> **O(log(N))** ≅ 23 acessos

- Considerando o tempo de acesso ao disco de 160ms (tempo de busca + latência + tempo transmissão do dado):
  - BST (pior caso) = $10 * 2^{20} * 160ms$ = **19,4 dias**
  - BST (caso médio) = **5,15 segundos**
  - AVL (melhor caso) = **3,73 segundos**

- Tempo de acesso ideal: 4/5 acessos (0,64 – 0,8 seg)

B-Tree – Introdução

- É uma árvore N-ária, ou seja, uma árvore com um número máximo de subárvores;

- Árvores B (ou B-Trees) foram projetadas para:
  - Manter a árvore balanceada e evitar o desperdício de espaço dentro de um nó à custa de maior complexidade nos algoritmos de inserção e remoção. Feita para gerenciamento de blocos do disco rígido.

- Introduzida por: R. Bayer e E. McCreight em:
- "Organization and Maintenance of Large Ordered Indexes", Acta Informatica, 1(3), Feb 1972.
- Revista por: D. Comer: "The Ubiquitous B-tree", ACM Computing Surveys, 11(2), 1979.

- Os autores nunca explicaram o "B" de "B"-trees.

  Boeing? Bayer-trees? Balanced-trees?

B-Tree – Introdução (cont.)

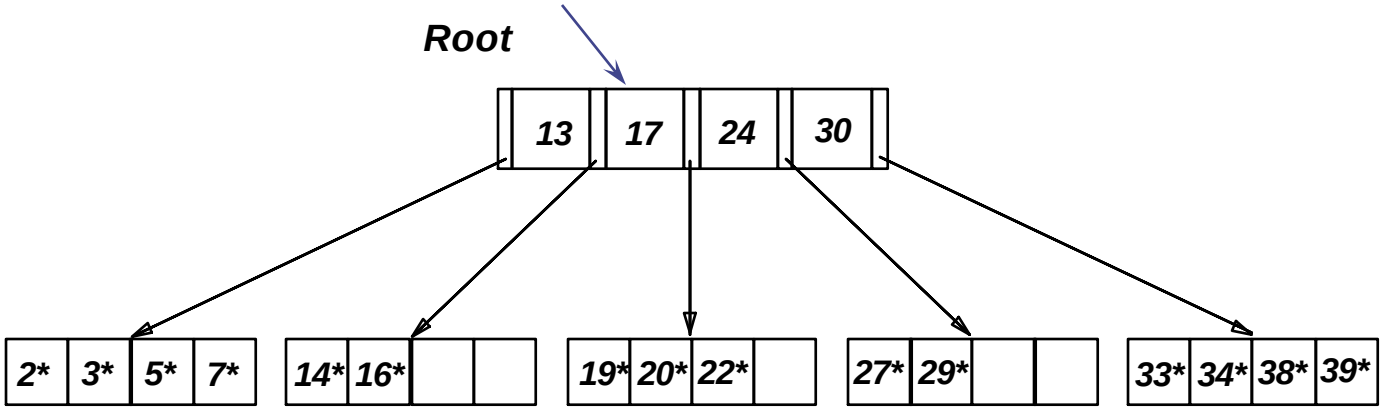
- Vídeo de Edward McCreight https://vimeo.com/73481096 explicando o "B" a partir de 16:20;

- Tradução minha: "*Bayer e eu estávamos no horário do almoço pensando sobre o nome... B, você sabe... Estávamos trabalhando na Boeing naquela época... Não podíamos usar esse nome sem falar com advogados. Então, há este B. Tem haver com equilíbrio, outro B. Bayer foi o autor sênior que tinha mais anos do que eu e muito mais publicações do que eu tinha. Então, há outro B. E, então, na "mesa" do almoço, nunca decidimos se existia um daquelas que fazia mais sentido do que as demais. O que realmente quer dizer é: quanto mais você pensa o que o B significa na B-Tree, melhor você entende as B-Trees*".

B-Tree – Introdução (cont.)

- A disk-based multi-level balanced tree index structure

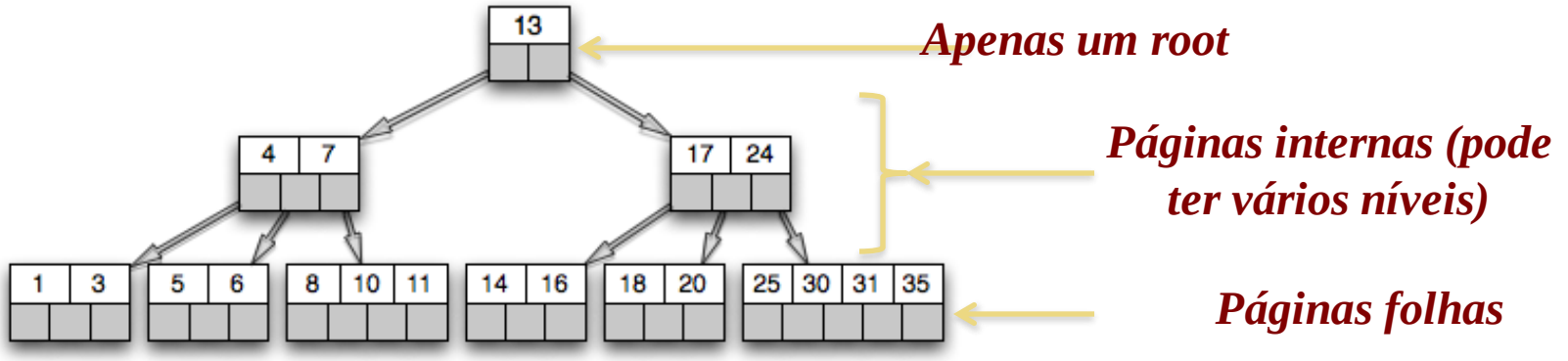
  - *Cada nó na árvore é uma página de disco*
  - *Os ponteiros apontam para os blocos no disco*



B-Tree – Introdução (cont.)

- A disk-based multi-level balanced tree index structure

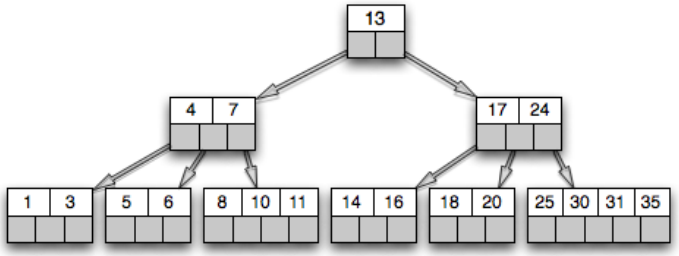
  - ***Apenas um root***
  - ***A Altura da árvore cresce ou diminui dinamicamente***
  - ***Toda (busca, inserção e exclusão) inicia do root até a folha***



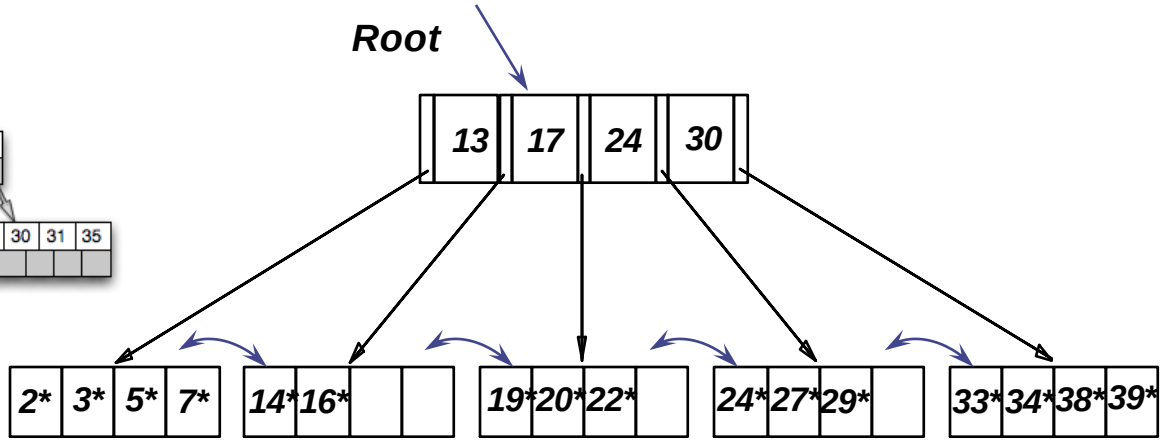
B-Tree – Introdução (cont.)

- A disk-based multi-level balanced tree index structure

  - *Todos os nós folhas estão no mesmo nível*



Altura 2



Altura 1

B-Tree – Características

- Os nós são mais comumente chamados de **páginas**, pois, ao contrário das árvores binárias, podemos ter "n" chaves no nó;

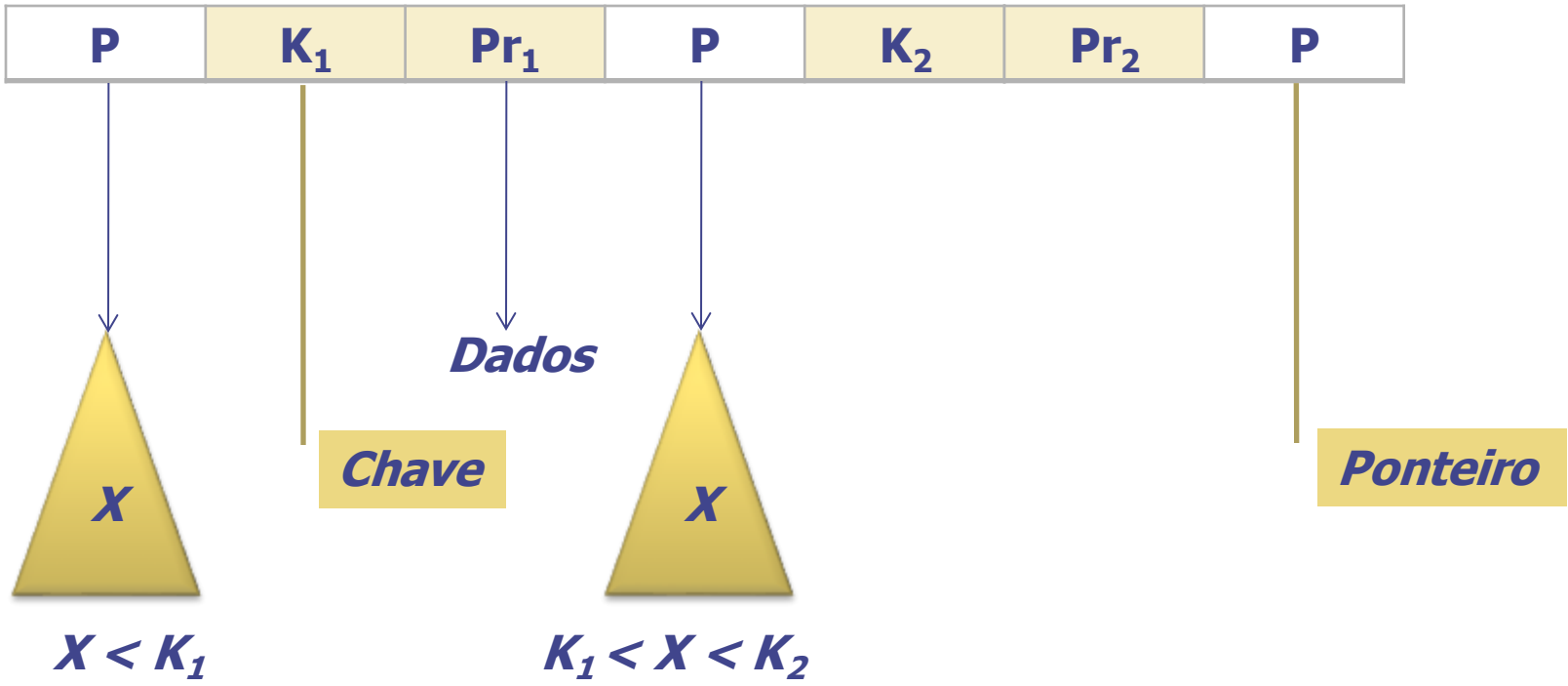
- Em uma árvore B de **ordem m**, cada página contém:
  - no mínimo m registros (e m + 1 subárvores);
  - no máximo 2m registros (e 2m + 1 subárvores);
  - exceto a página raiz, que pode conter entre 1 e 2m registros.

- Todas as páginas "folha" aparecem no mesmo nível.

B-Tree – Características (cont.)

- Operações de inserção e remoção são balanceadas de modo que cada nó tenha uma ocupação minima e máxima;
- Uma ocupação mínima de 50% é garantida em cada nó (exceto a raiz);
- Procura por um registro requer somente uma busca em profundidade da raiz até uma folha apropriada;
- Todos os caminhos da raiz até a folha têm o mesmo comprimento, ou seja, todas as folhas possuem a mesma profundidade;
- A página raiz tem ao menos duas páginas filhas (à exceção da página raiz ser uma folha);
- Uma página não folha com k páginas filhas terá k – 1 chaves.

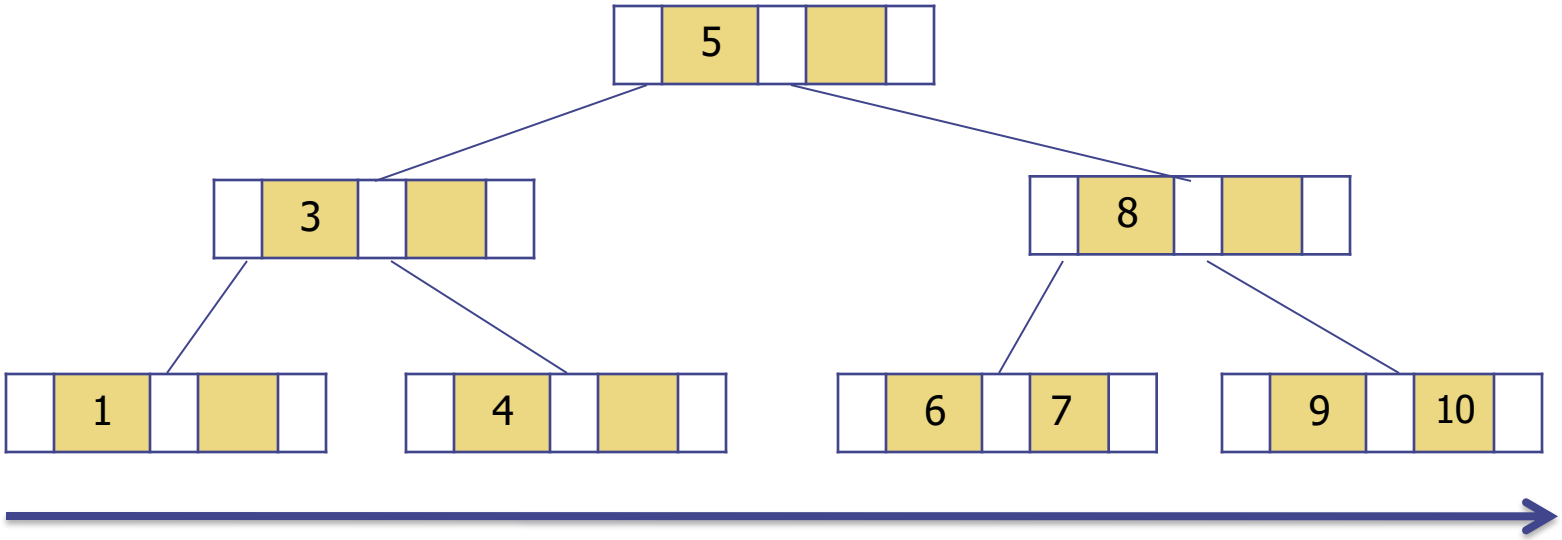
B-Tree – Página



Ordem m = 1

B-Tree – Exemplo

- *Árvore B de ordem 1 com três níveis.*



- *Todas as páginas contém 1 ou 2 chaves;*
- *A raiz poderá conter apenas um registro, quando a ordem for maior que 1;*
- *Os registros aparecem em ordem crescente da esquerda para a direita;*
- *É uma extensão natural da organização da árvore binária de pesquisa.*

B-Tree – Inserir

- **Primeiro passo:** pesquise a chave para ter a certeza de que esta não exista na árvore;
- **Segundo passo:** busque a posição onde esta será inserida (**uma folha**). Se a página não estiver cheia, insira o valor dentro dela (reordenando as suas chaves, caso seja necessário), senão execute uma subdivisão da página da seguinte forma:
  - Verifique se a página pai está cheia, se não:
    - Passe o elemento do meio da página para seu pai;
    - Divida a página em duas páginas iguais.
  - Se a página pai estiver cheia, repita as duas linhas acima recursivamente. Caso todas as páginas-pais estiverem cheias, inclusive a raiz, deve ser criada uma nova página aumentando assim a altura da árvore.
- Somente após satisfazer todas as divisões necessárias, insira a nova chave.

B-Tree – Exemplo m = 1

- Inserindo Danrlei

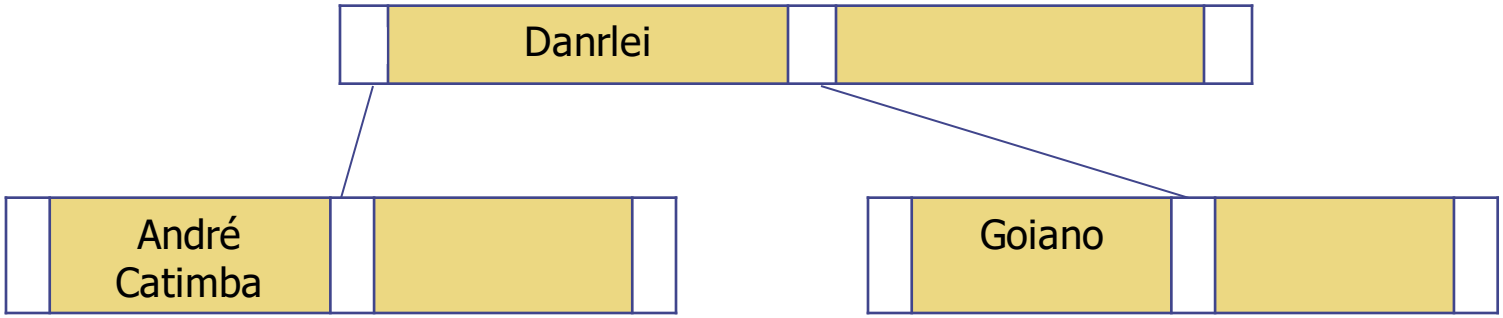
B-Tree – Exemplo m = 1

- Inserindo André Catimba

- *Ordenar as duas chaves de forma a manter a classificação.*

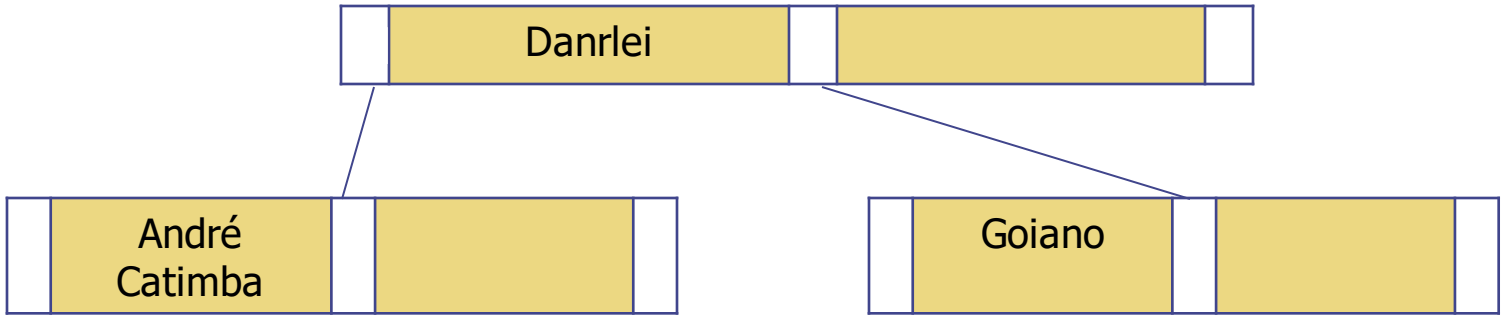
B-Tree – Exemplo m = 1

- Inserindo Goiano

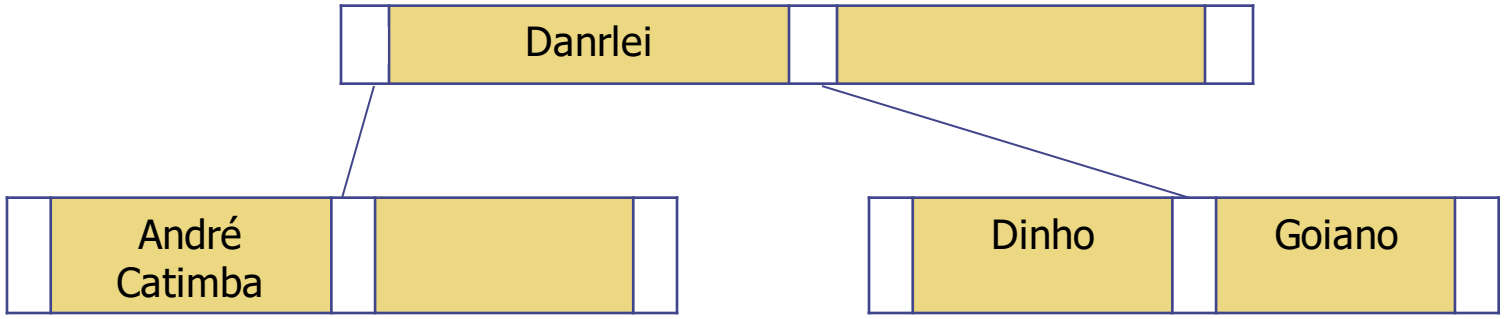


- Ocorrerá overflow, ou seja, a página está cheia;
- A chave do meio (elementos ordenados) é Danrlei que será promovida como pai (nova página) da página atual. Ocorrerá a divisão (split) da página.

B-Tree – Exemplo m = 1



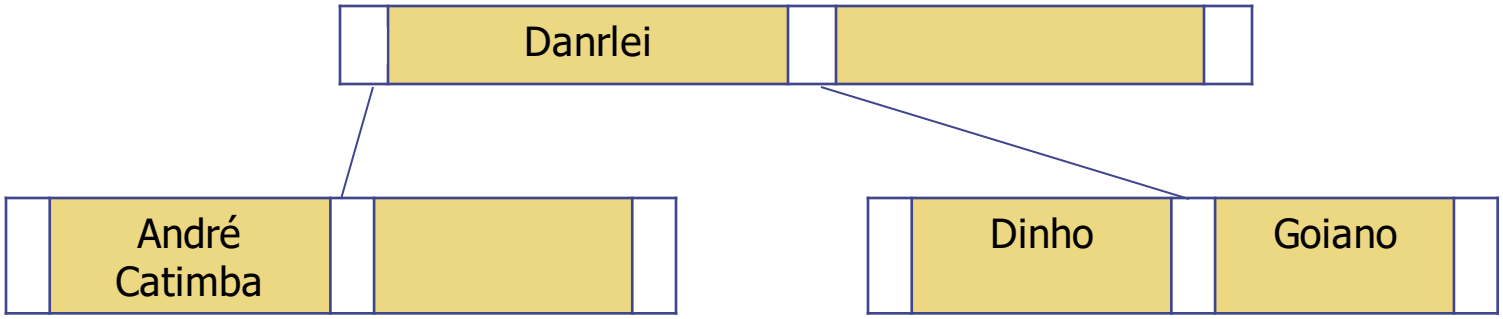
- Inserindo Dinho



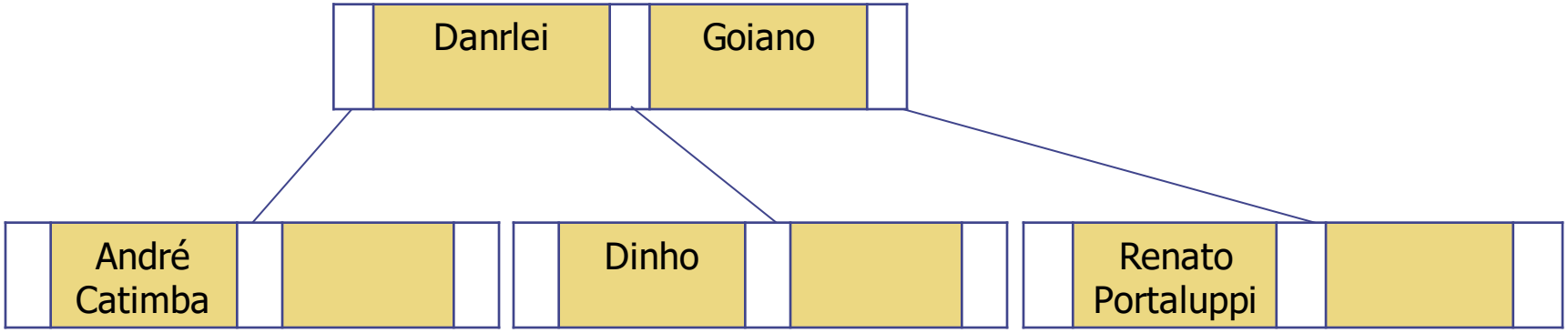
- Navegar na árvore até achar a página folha (página com o Goiano);
- Inserir a chave reorganizando a página.

Momento lúdico: Observem aqui o esforço que o professor fez para que o Dinho e Goiano ficassem juntos! Melhor meio-campo da história Gremista!

B-Tree – Exemplo m = 1

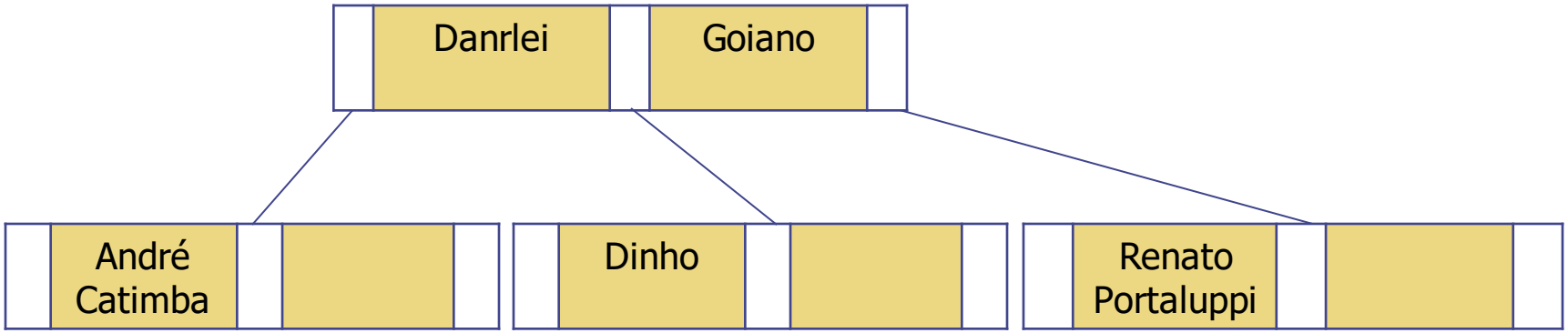


- Inserindo Renato Portaluppi

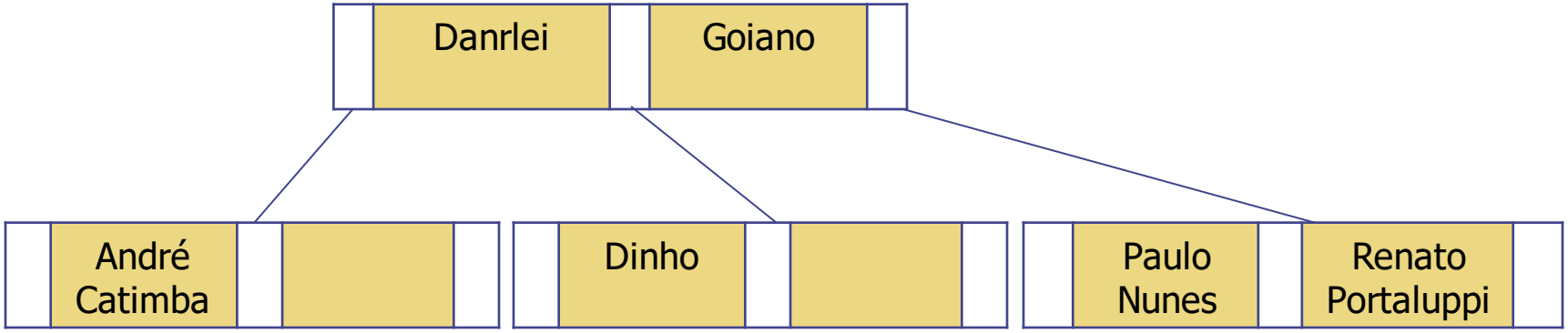


- Navegar até a página folha (página com o Dinho e Goiano);
- Como a página está cheia é necessário fazer a divisão (split);
- A chave do meio é promovida para a raiz;
- Como há espaço na raiz, ela não terá divisão.

B-Tree – Exemplo m = 1

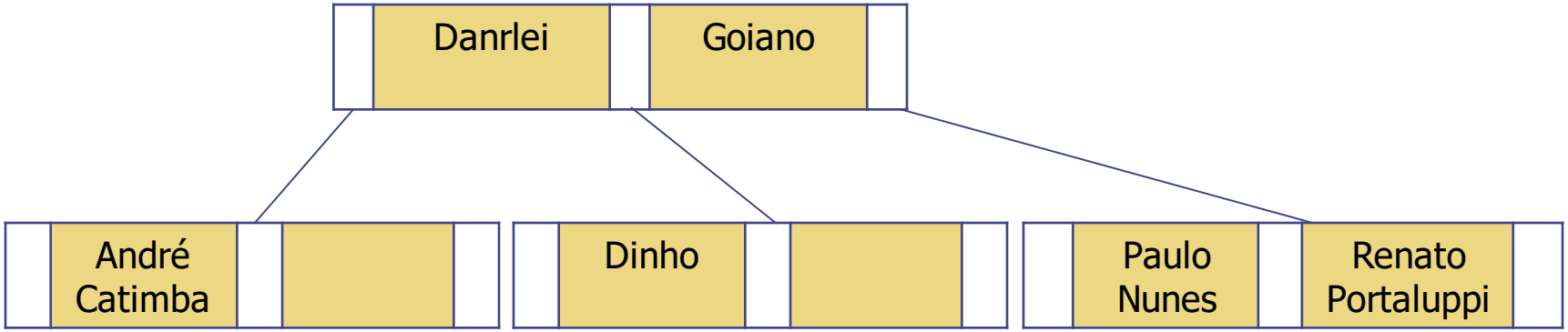


- Inserindo Paulo Nunes

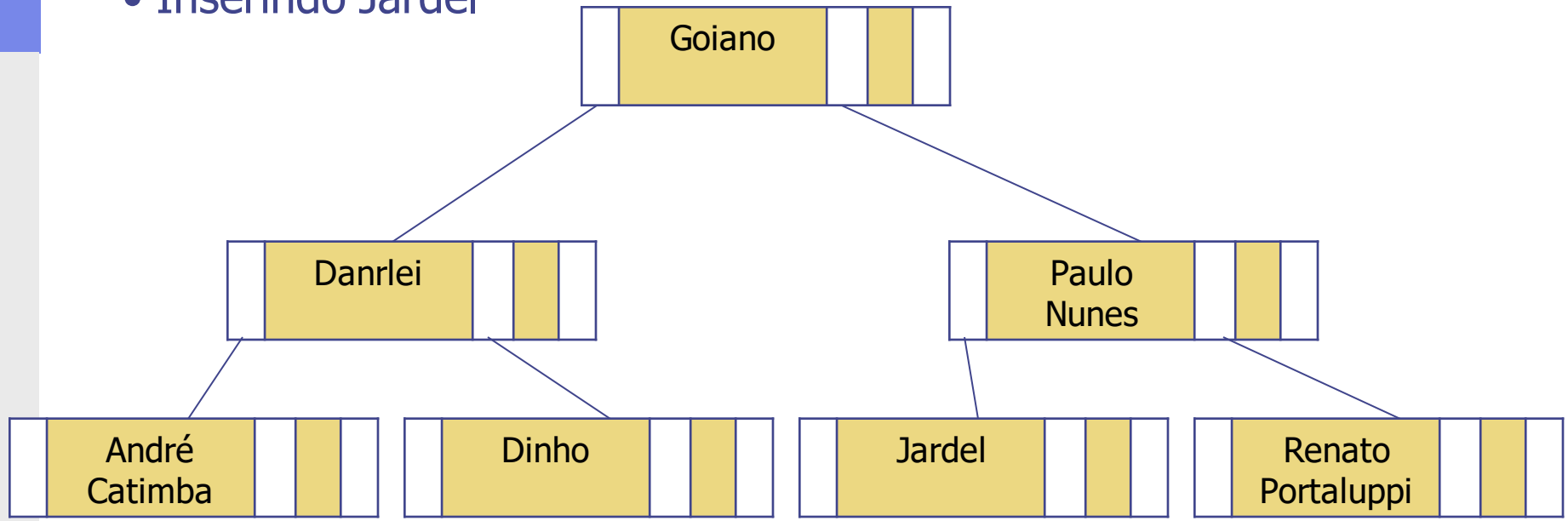


- Navegar até a página folha (página com o Renato Portaluppi);
- Como a página não está cheia, apenas inserir.

B-Tree – Exemplo m = 1



- Inserindo Jardel



- Navegar até a página folha (página com o Paulo e Renato);
- A página está cheia, então, é dividido;
- A chave Paulo Nunes é movida para a página raiz;
- A página raiz está cheia, então, é dividido e uma nova página criada.

B-Tree – Desempenho

- São necessários 3 acessos para encontrar o Renato Portaluppi
- Em relação à utilização:

$$\frac{\text{Registros Utilizados}}{\text{Registros Totais}} = \frac{7}{14} = 50\%$$

- Como a Árvore B é balanceada, no pior caso, o número total de acessos é igual a altura da árvore para a ordem "m":

$$h \leq \log_{m+1} \frac{n+1}{2} + 1$$

Onde "n" é o número de registros armazenados.
Portanto, no pior caso, a complexidade é $O(\log_m n)$.

B-Tree – Considerações

- Ordem m = 1
- n = 7

$$h \leq \log_{1+1} \frac{7+1}{2} + 1$$

$$h \leq 3$$

- Ou seja, no máximo 3 acessos.

- Ordem m = 256
- n = 10.000.000

$$h \leq \log_{256+1} \frac{10.000.000+1}{2} + 1$$

$$h \leq \approx 2{,}8$$

- Se fizéssemos com um AVL teríamos: ~23 acessos $\log_2(10.000.000)$

B-Tree – Complexidade

- Na inserção, cada cisão (split ou divisão) opera sobre um número fixo de páginas, portanto, sua complexidade é de O(1) acesso. No pior caso, overflows se propagarão até a raiz, resultando em complexidade $O(\log_m n)$ splits com O(1) acesso cada;

- Na exclusão, cada concatenação opera sobre um número fixo de páginas, logo, sua complexidade é de O(1) acesso. No pior caso, underflows se propagarão até a raiz, resultando em complexidade $O(\log_m n)$ concatenações com O(1) acesso cada;

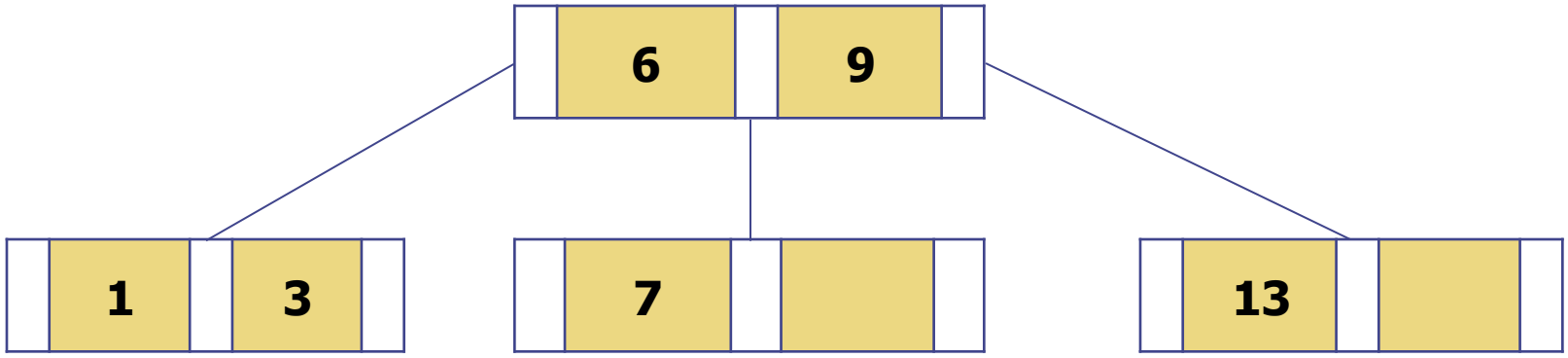
- Na busca, o pior caso será o maior nível (ou a altura) da árvore, que, neste caso é $O(\log_m n)$ acessos.

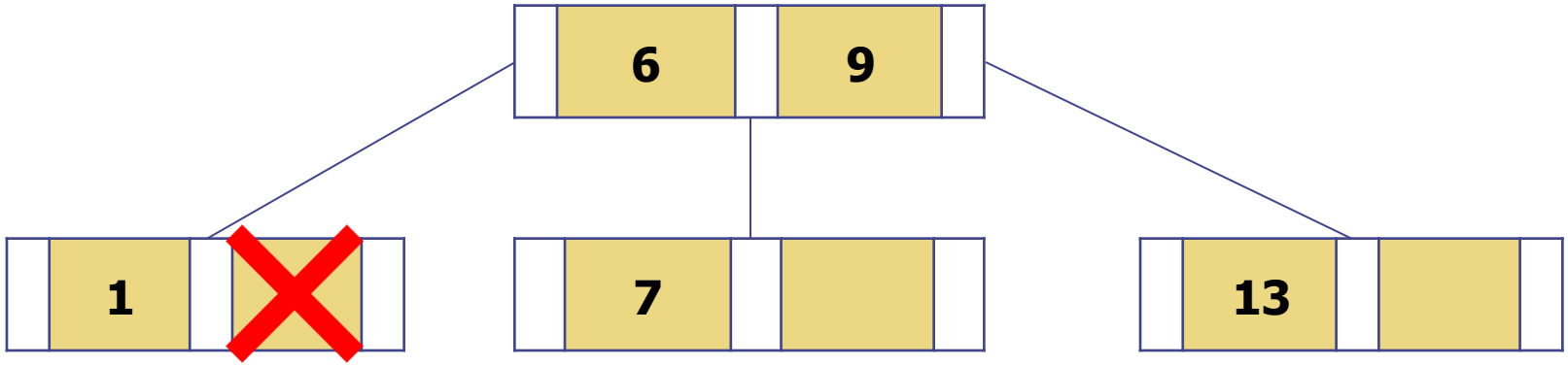
B-Tree – Exclusão

- Há 4 casos principais de exclusão:
  - Caso 1: exclusão de uma chave na página folha – sem underflow (capacidade mínima da página);
  - Caso 2: exclusão de uma chave não-folha – sem underflow;
  - Caso 3: exclusão de uma chave na folha; underflow com "irmão rico";
  - Caso 4: exclusão de uma chave na folha; underflow com "irmão pobre".

B-Tree – Exclusão – Caso 1

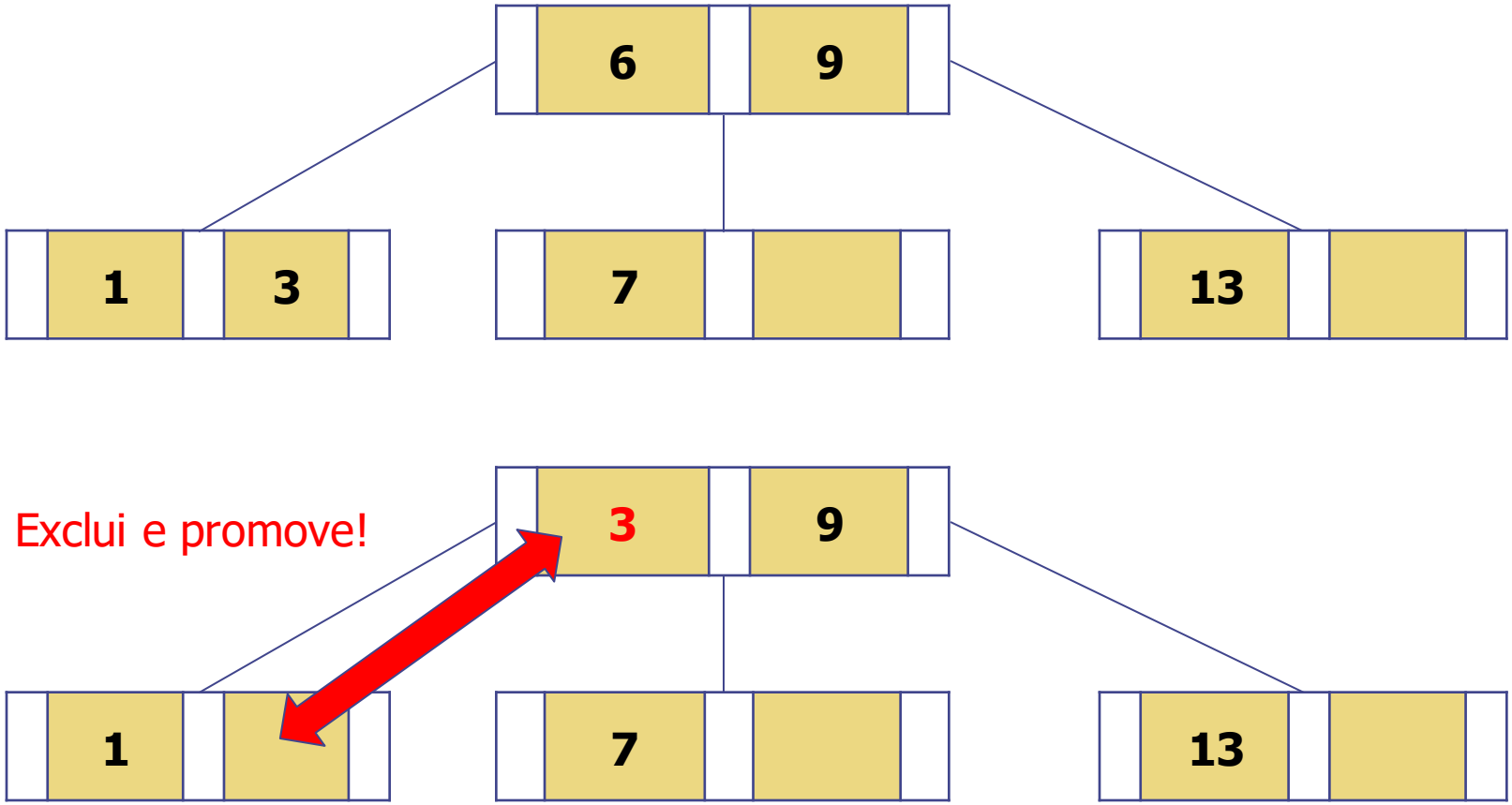
- Caso 1: exclusão de uma chave na página folha – sem underflow. Ex.: 3.





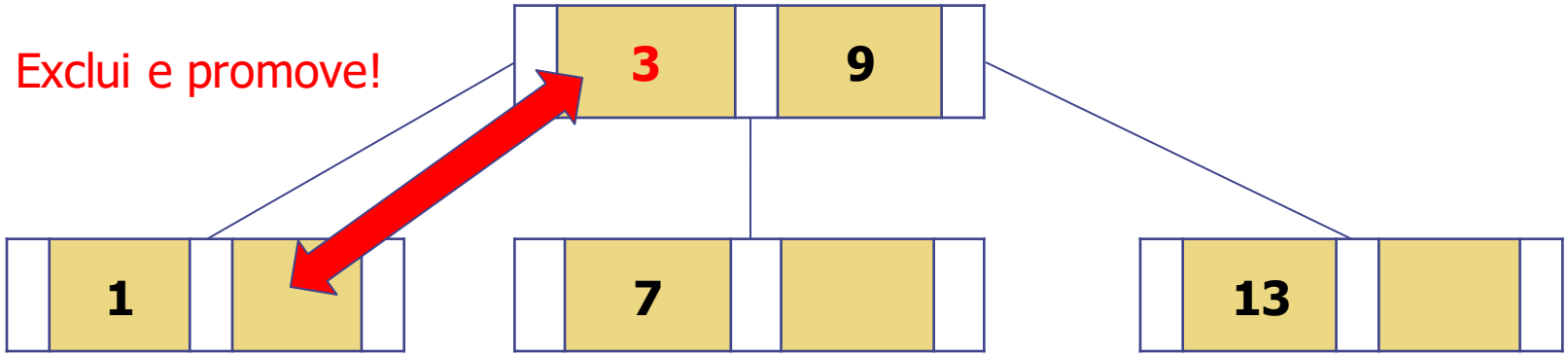
B-Tree – Exclusão – Caso 2

- Caso 2: exclusão de uma chave não-folha – sem underflow. Ex.: 6.



B-Tree – Exclusão – Caso 2

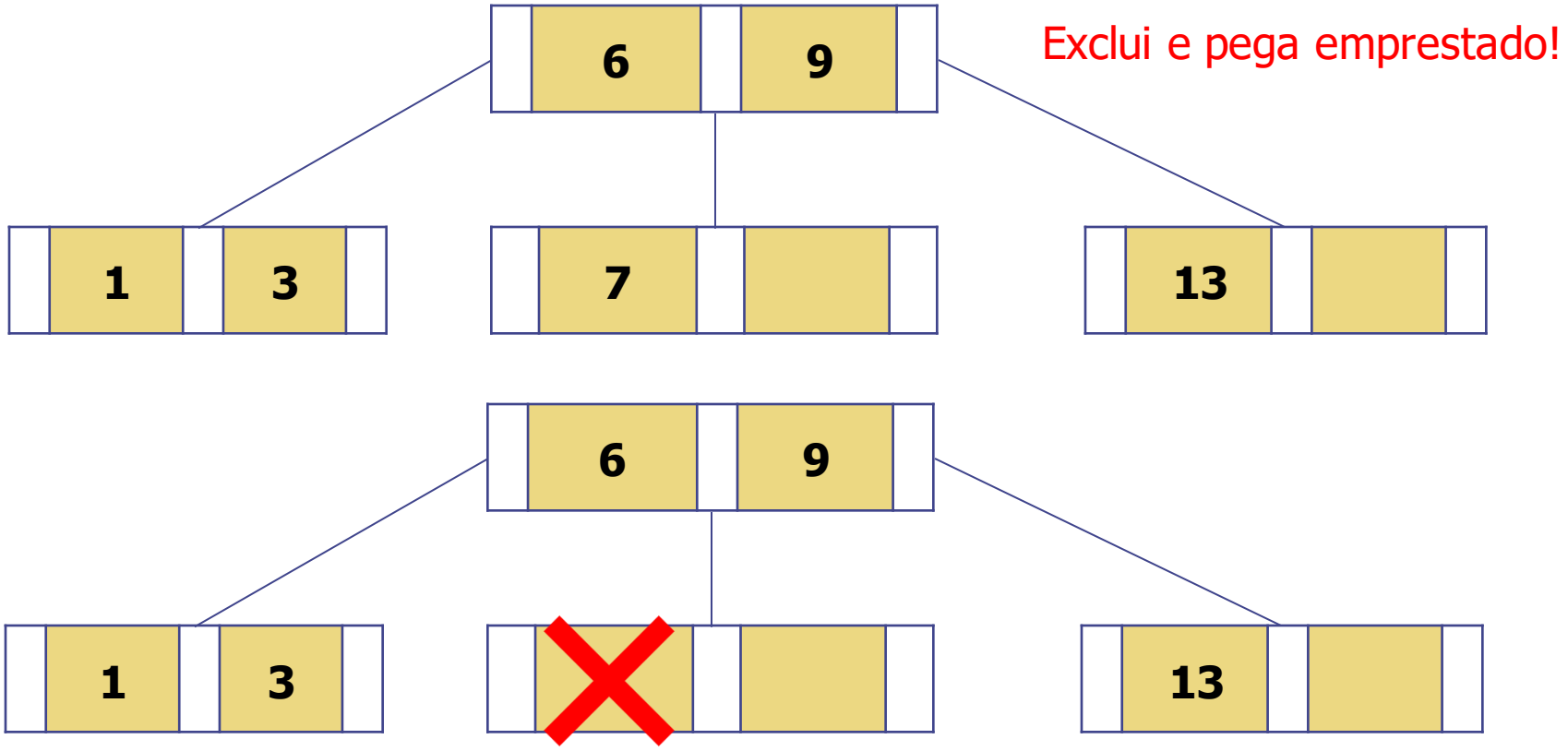
- Caso 2: exclusão de uma chave não-folha – sem underflow. Ex.: 6.



- Qual chave promover? Busca-se a maior chave da subárvore à esquerda ou a menor chave da subárvore à direita.
- Observação: Com a exclusão de uma chave não-folha, exclui-se uma chave na folha.

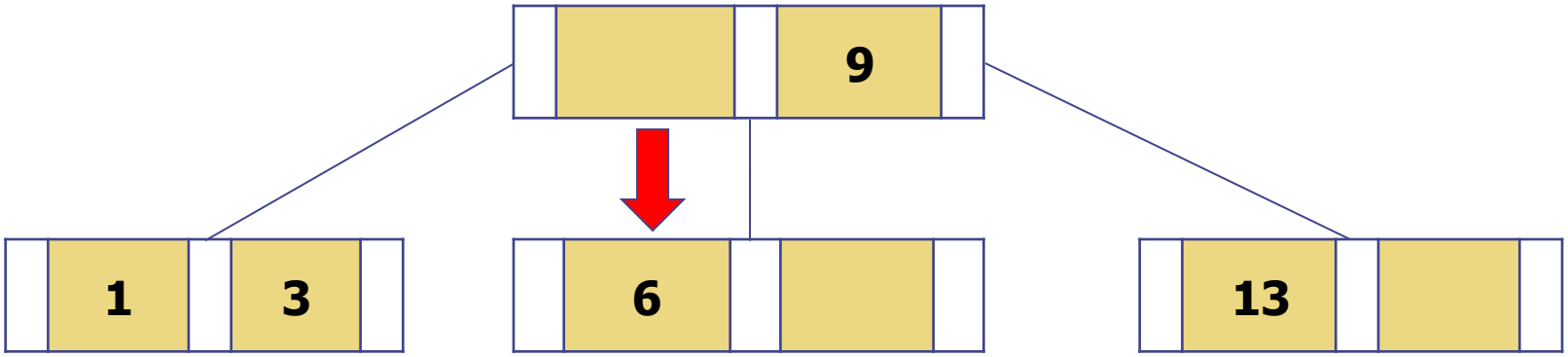
B-Tree – Exclusão – Caso 3

- Caso 3: exclusão de uma chave na folha; underflow com "irmão rico". Ex.: 7.



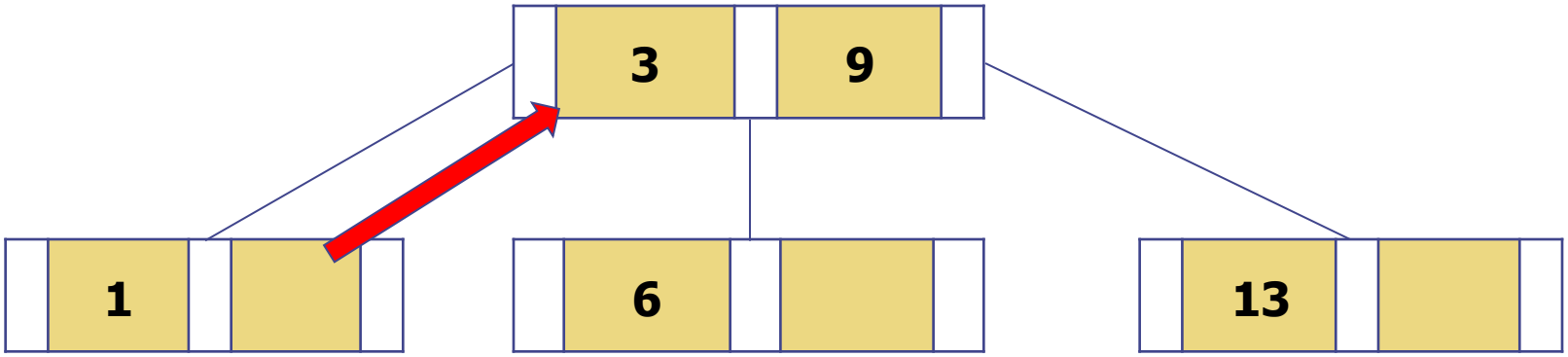
B-Tree – Exclusão – Caso 3

- Caso 3: exclusão de uma chave na folha; underflow com "irmão rico". Ex.: 7.



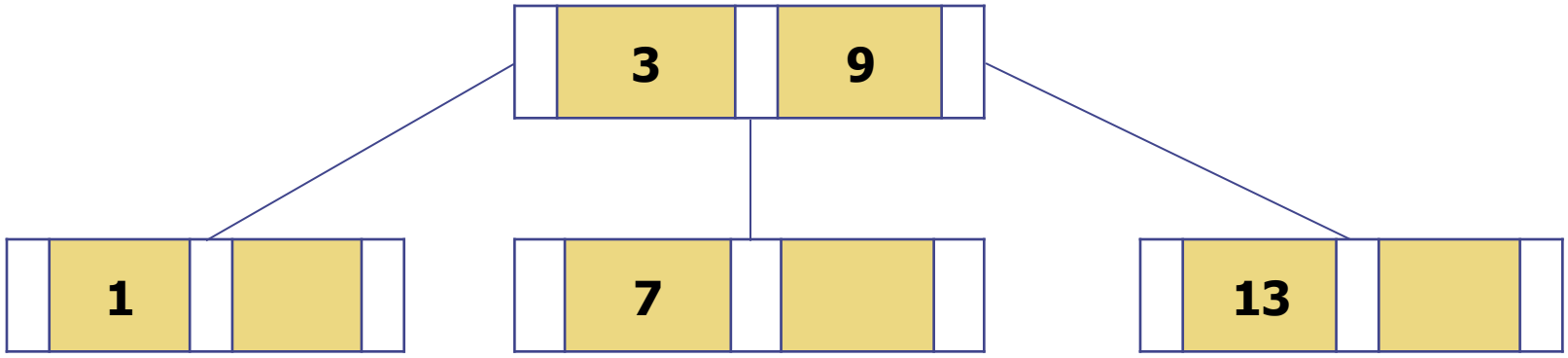
Pega-se emprestado: usa-se a chave do parente.

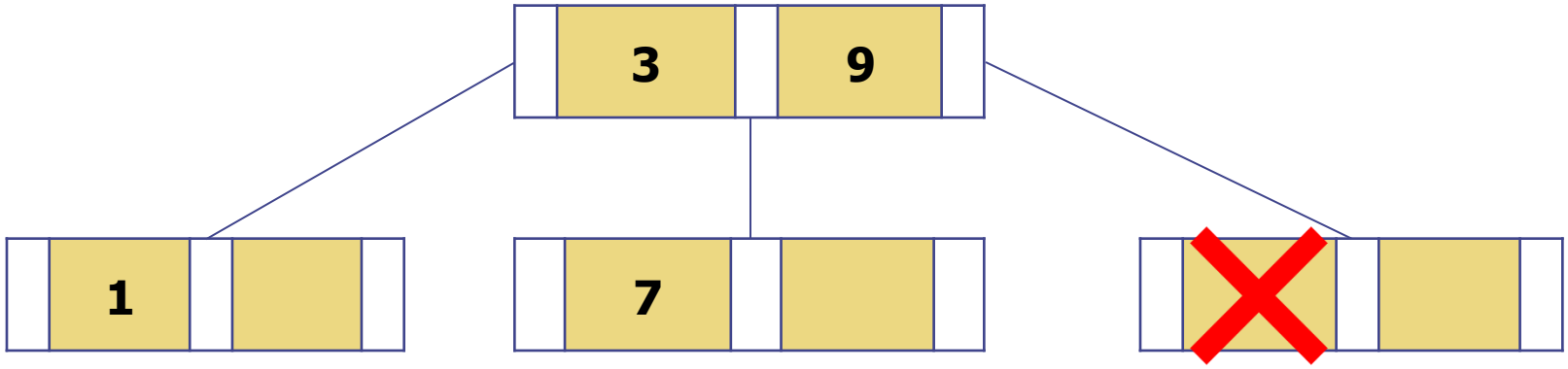
Irmão rico: é aquele que pode pegar uma chave sem ocorrer underflow.



B-Tree – Exclusão – Caso 4

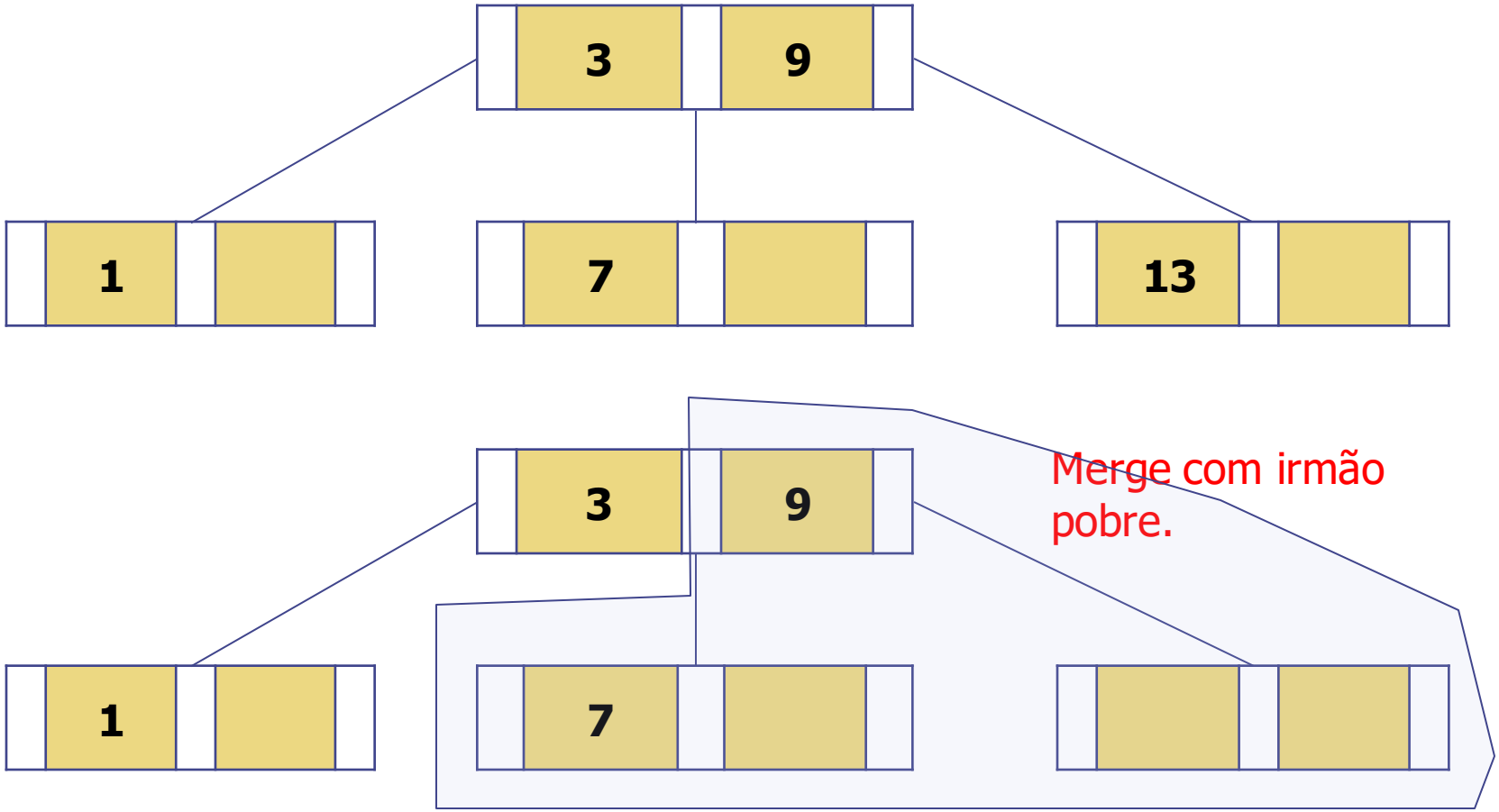
- Caso 4: exclusão de uma chave na folha; underflow com "irmão pobre". Ex.: 13.





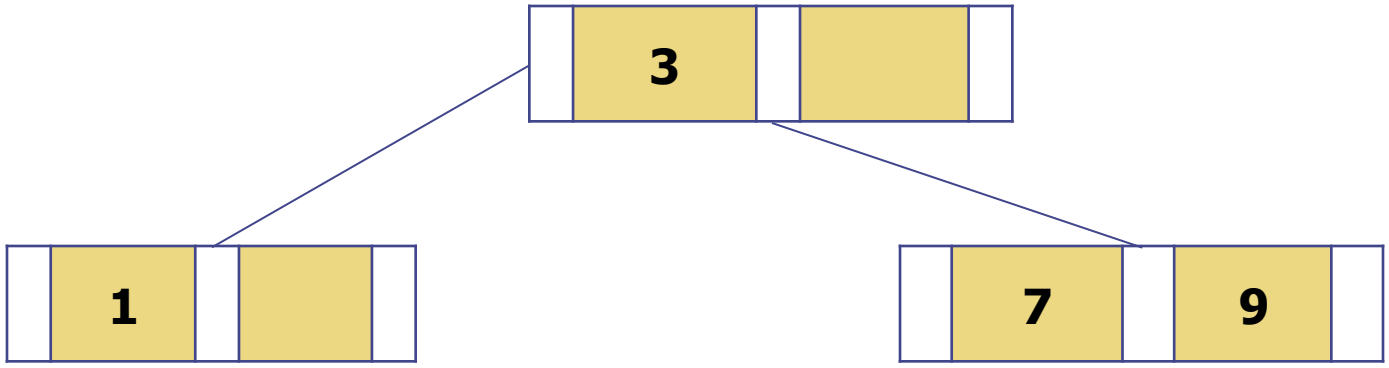
B-Tree – Exclusão – Caso 4

- Caso 4: exclusão de uma chave na folha; underflow com "irmão pobre". Ex.: 13.



B-Tree – Exclusão – Caso 4

- Caso 4: exclusão de uma chave na folha; underflow com "irmão pobre". Ex.: 13.



- A união ocorre pegando a chave do pai fazendo exatamente o contrário da inserção;
- **Caso ocorra underflow no pai, então, faz o processo (merge) recursivamente.**

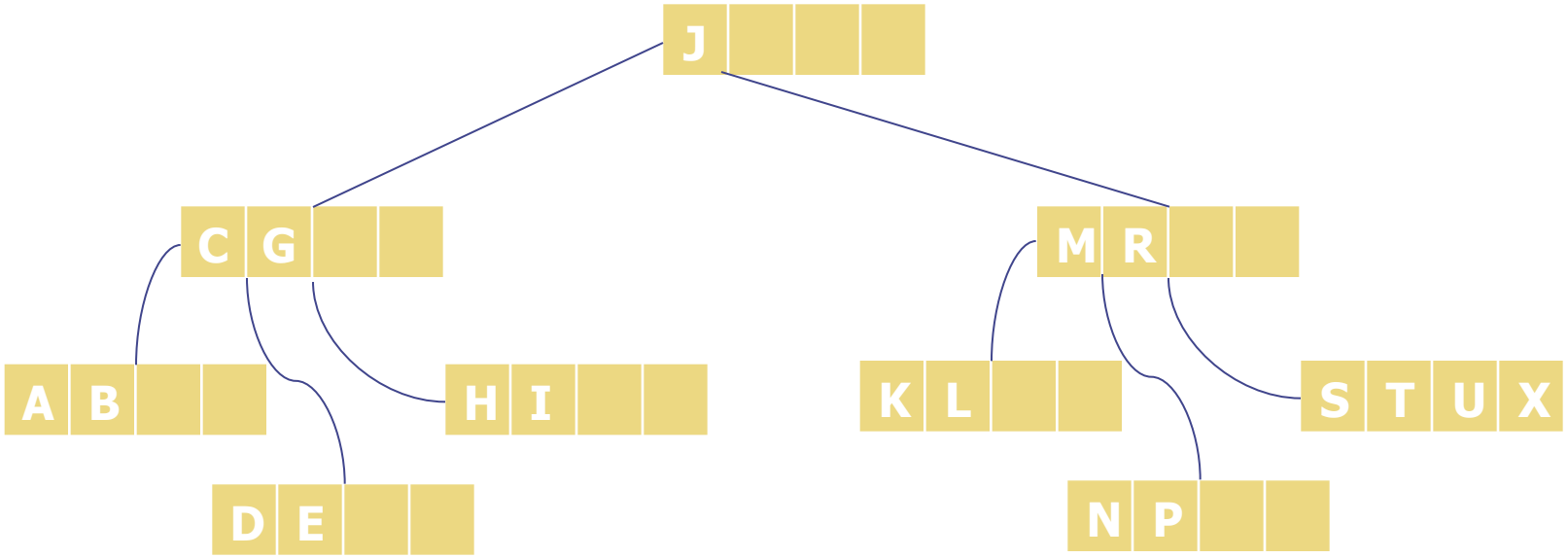
B-Tree - Exercício

- **Exercício 1.** Mostre passo-a-passo a inserção dos valores de "A" a "G" em uma árvore B. m = 1.

- **Exercício 2.** Mostre a árvore passo-a-passo. m = 2. Chaves: 20, 10, 40, 50, 30, 55, 3, 11, 4, 28, 36, 33, 52, 17

B-Tree - Exercício

- **Exercício 3.** Exclua a chave **E** para a árvore de ordem m = 2. Qual caso (1, 2, 3 ou 4) se encaixa essa exclusão?



B+-Tree

- É uma variação das Árvores B, porém, os **dados são gravados nas folhas** de forma a melhorar o desempenho na recuperação do dado;

- Os nós internos (ramos) possuem apenas as chaves e apontadores para as folhas;

- As folhas estão ligadas entre si com uma lista encadeada de forma a gerar uma consulta eficiente;

- O número máximo de "apontadores" de uma página é chamado de ordem m (pouco diferente ao das Árvores B).

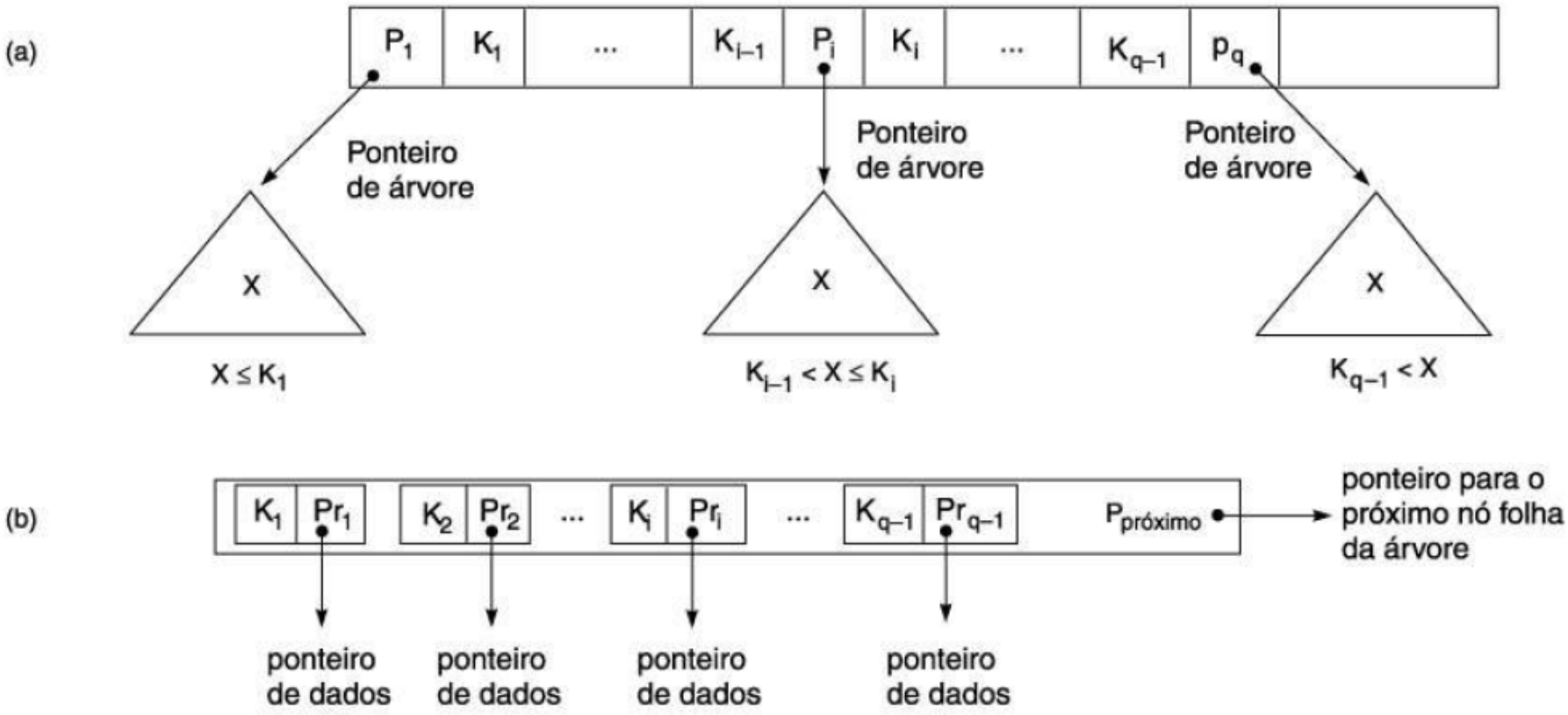
B^+-Tree - Características

- A raiz ou será uma folha ou terá 2 ou m subárvores;
- As chaves/dados são armazenados nas folhas;
- Os nós internos armazenam m – 1 chaves de forma a guiar na busca;
- Cada nó interno terá entre $\lceil m/2 \rceil$ e m filhos; exclui-se a raiz;
- Cada folha possui a mesma profundidade e terá entre $\lceil L/2 \rceil$ e m itens (dados).

B^+-Tree – Características (cont.)

Capacidade do m	Capacidade do L
Cada nó representa um bloco de disco	Cada nó representa um bloco de disco
Assumindo que um bloco de disco seja 8K = 2^{13} bytes	Assumindo que um bloco de disco seja 8K = 2^{13} bytes
Um nó interno pode armazenar m – 1 chaves e m referências ao disco	Um nó folha pode armazenar L registros
Assumindo que uma chave requer 32 bytes	Assumindo que um registro requer 256 bytes
Assumindo que uma referência ao disco requer 4 bytes	
$32(m - 1) + 4 * m \leq 2^{13}$	$256 * L \leq 2^{13}$
$36m - 32 \leq 8192$	$2^8 * L \leq 2^{13}$
$m \leq 228$	$L \leq 32$

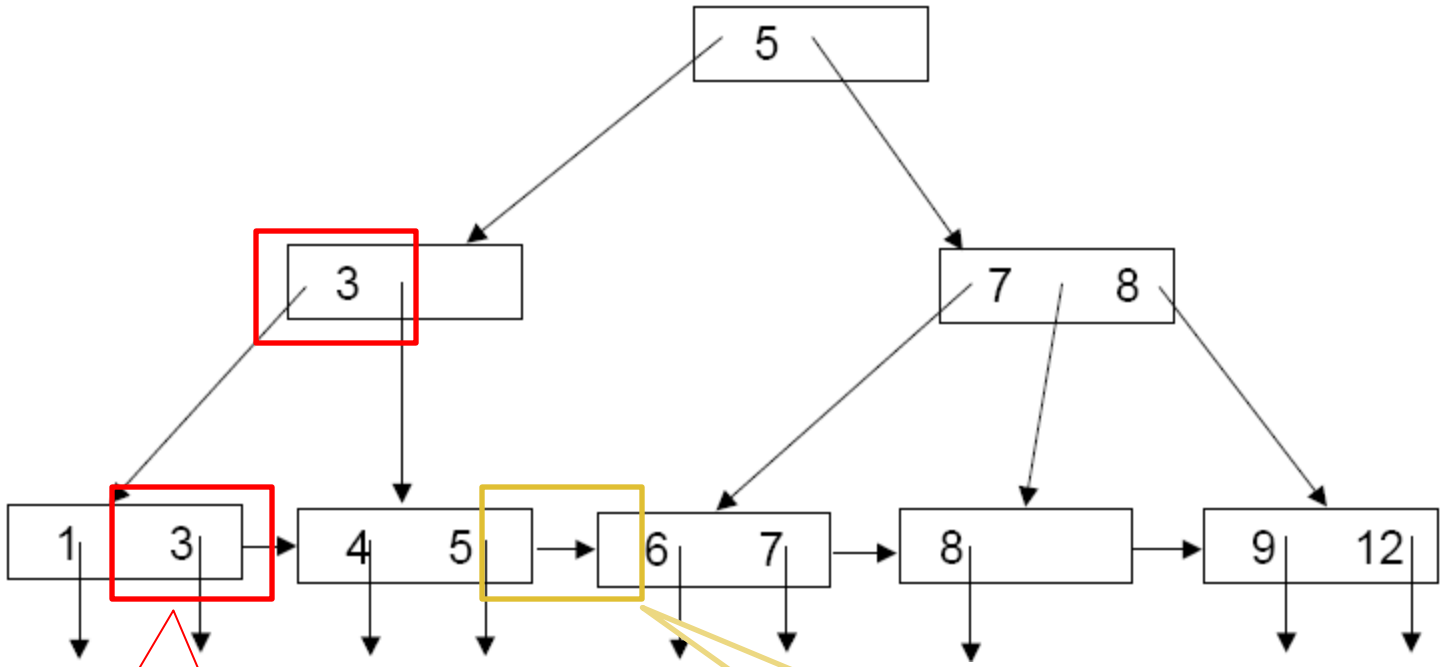
B+-Tree – Estrutura



Os nós de uma árvore-B+. (a) Nó interno de uma árvore-B+ com q - 1 valores de busca. (b) Nó folha de uma árvore-B+ com q - 1 valores de busca e q - 1 ponteiros de dados.

B+-Tree – Exemplo

Neste exemplo, o fator de igualdade é para a esquerda.



Todos os valores estão nas folhas

Um lista encadeada para aumentar o desempenho

B+-Tree – Código Fonte

```
public class BPlusTree<K extends Comparable<K>, V> {
   private static final int M = 4; // M - 1 (máximo de filhos) M >= 4
   private Node root;
   private int height; // Altura da árvore
   private int size; // Número de nós

   private static class Node {
       private Entry[] children = new Entry[M]; // Array com os filhos - 1
       private int size; // Número de filhos (M - 1)
       // Cria o nó com um tamanho k
       public Node(int k) {
          size = k;
       }
   }

   // Nós internos usam somente a key e o next
   // Nós extenos (filhos) usam somente a key e o value
   private static class Entry {
       private Comparable<?> key;
       private Object value;
       private Node next;

       public Entry(Comparable<?> key, Object value, Node next) {
          this.key = key;
          this.value = value;
          this.next = next;
       }
   }
   // Continua...
```

B+-Tree – Código Fonte

```
public BPlusTree() { root = new Node(0); }

public boolean isEmpty() {return size() == 0; }

public int size() { return size; }

public int height() { return height; }

public V search(K key) {
   return search(root, key, height);
}

private V search(Node node, K key, int height) {
   Entry[] children = node.children;

   // Nó folha
   if (height == 0) {
       for (int i = 0; i < node.size; i++)
         if (key.compareTo((K) children[i].key) == 0)
             return (V) children[i].value;
   } else {
   // Nó interno
      for (int i = 0; i < node.size; i++)
         if (i + 1 == node.size || key.compareTo((K) children[i + 1].key) < 0)
             return search(children[i].next, key, height - 1);
   }
   return null;
}
```

B+-Tree – Código Fonte

```
public void insert(K key, V value) {
   Node newNode = insert(root, key, value, height);
   size++;
   if (newNode == null) return;

   // Split do root
   Node temp = new Node(2);
   temp.children[0] = new Entry(root.children[0].key, null, root);
   temp.children[1] = new Entry(newNode.children[0].key, null, newNode);
   root = temp;
   height++;
}
```

B+-Tree – Código Fonte

```
private Node insert(Node node, K key, V value, int height) {
   int i;
   Entry newEntry = new Entry(key, value, null);
   // Nó folha
   if (height == 0) {
       for (i = 0; i < node.size; i++)
          if (key.compareTo((K) node.children[i].key) < 0)
              break;
   } else {
   // Nó interno
   for (i = 0; i < node.size; i++) {
     if (i + 1 == node.size || key.compareTo((K) node.children[i + 1].key) < 0) {
            Node temp = insert(node.children[i++].next, key, value, height - 1);
            if (temp == null) return null;
            newEntry.key = temp.children[0].key;
            newEntry.next = temp;
            break;
     }
   }
  }

  for (int j = node.size; j > i; j--)
     node.children[j] = node.children[j - 1];
  node.children[i] = newEntry;
  node.size++;
  if (node.size < M) return null;
  return split(node);
}
```

```
private Node split(Node h) {
   Node t = new Node(M / 2);
   h.size = M / 2;
   for (int j = 0; j < M / 2; j++)
     t.children[j] = h.children[M / 2 + j];
   return t;
}
```

B+-Tree – Código Fonte

```
public String toString() {
   return toString(root, height, "") + "\n";
}

private String toString(Node node, int height, String indent) {
   StringBuilder s = new StringBuilder();
   Entry[] children = node.children;

   if (height == 0) {
      for (int i = 0; i < node.size; i++) {
         s.append(indent + children[i].key + "\n");
      }
   } else {
      for (int i = 0; i < node.size; i++) {
        if (i > 0) s.append(indent + "(" + children[i].key + ")\n");
        s.append(toString(children[i].next, height - 1, indent + "     "));
     }
   }
   return s.toString();
}
```

Código extraído de: ***http://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/BTree.java***

- Observe que no site o nome está como BTree, porém, o comportamento é de uma BPlusTree.
- Além disso, o fator de igualdade está para a direita, ou seja, k - 1 < k <= k + 1.

B+-Tree – Exemplo Passo-a-passo

Fator de igualdade para a esquerda.

- Mostre a árvore (passo-a-passo). Ordem **m = 3.** Inserir os valores: 2, 1, 5, 7, 3, 4.

B+-Tree – Exercício

Fator de igualdade para a esquerda.

- **Exercício 4.** Mostre a árvore (passo-a-passo). Ordem **m = 3.** Inserir os valores: 20, 10, 40, 50, 30, 55, 3.

- **Exercício 5.** Mostre a árvore (passo-a-passo). Ordem **m = 5.** Inserir os valores: X, M, B, A, 55, 3, 11, 4, 28, 36, 33, 52, 17, 25, 13, 45, 9, 43, 8, 48.

Simuladores

- *B-Tree*
- *http://ysangkok.github.io/js-clrs-btree/btree.html*
- *https://www.cs.usfca.edu/~galles/visualization/BTree.html*
- *B^+-Tree*
- *https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html*
- *http://www.seanster.com/BplusTree/BplusTree.html*

Bibliografia

- Silberchatz, A; Korth, H. F., Sudarshan, S. Sistema de Banco de Dados. 3ª. Edição, Makron Books, 1999.
- Slides do Prof. Clodis Boscarioli. Unioeste. Aulas 1, 2 e 3.
- Slides do Prof. Casanova. PUC-Rio, Módulo 15- Estruturas de Indexação.
- Slides da Profa. Sandra de Amo. Gerenciamento de Banco de Dados.
- Slides do Prof. Ilmério Reis da Silva. Gerenciamento de Banco de Dados 2 – Armazenamento de Dados.