



Estruturas de dados II

Árvore B

André Tavares da Silva
andre.silva@udesc.br

Árvores B

- Estrutura de árvore autobalanceada para dados classificados
 - Acesso sequencial
 - Acesso aleatório (pesquisa)
- Possui complexidade de tempo logarítmo
 - Acesso aleatório: $O(\log n)$
 - Inserção: $O(\log n)$
 - Remoção: $O(\log n)$
- Idealizada por Rudolf Bayer e Edward McCreight (1971)
 - Muito utilizada atualmente em banco de dados e sistemas de arquivos

Relação entre árvores binárias e árvores B

- Árvore binária
 - Recomendado para **classificação interna**
 - Trabalha com dados em memória principal
 - Estrutura de árvore autobalanceada para dados classificados
 - Possui complexidade de tempo logarítmo
- Árvore B
 - Recomendado para **classificação externa**
 - Trabalha com dados em memória secundária
 - Objetiva reduzir o acesso aos dispositivos de memória secundária
 - É uma generalização de árvores binárias
 - Porém, os nós podem ter mais de dois filhos

Conceitos da árvores B

- Nó da árvore
 - Armazena um conjunto não vazio de chaves
 - As chaves são armazenadas em ordem crescente
 - Os nós também são chamados de páginas

- Páginas
 - Refere-se à organização de dados em blocos
 - Visa reduzir o acesso de E/S a partir técnica de paginação de dados
 - Exemplos
 - Paginação de memória principal
 - Sistema de arquivos
 - Gerenciadores de bancos de dados

Conceitos da árvores B

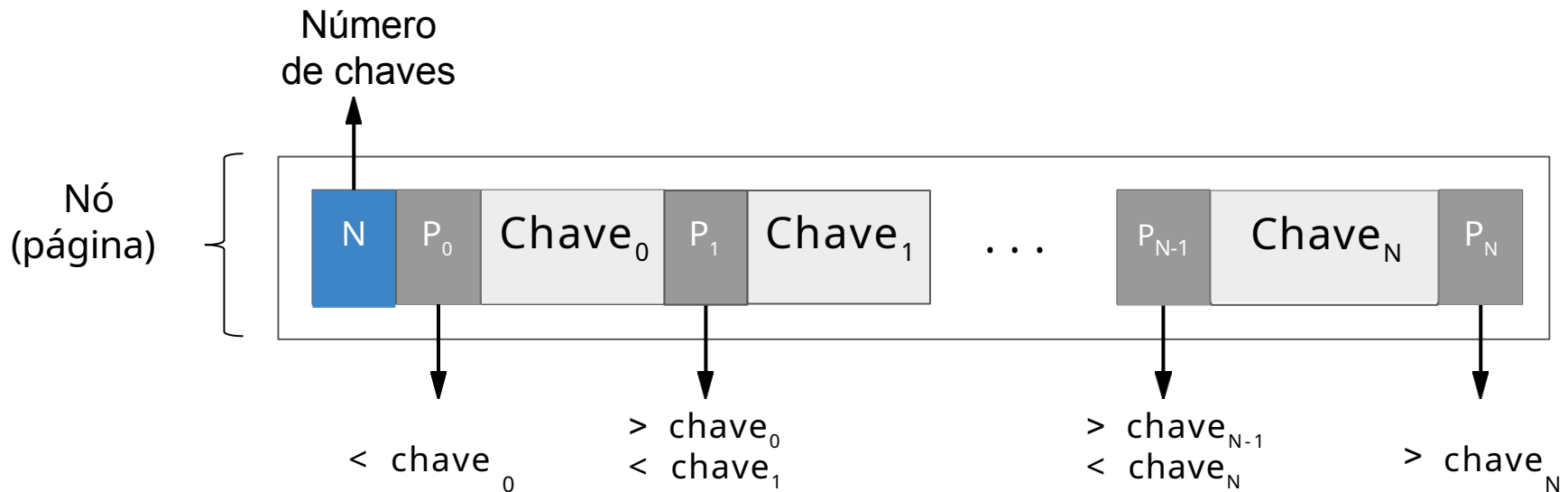
- Quantidade de chaves em um nó
 - Todo nó possui uma quantidade mínima e máxima de chaves
 - Exceto o nó raiz que não possui quantidade mínima
 - Para cada chave há um ponteiro a esquerda e direita para os nós filhos
 - A quantidade de filhos de um nó será o número de chaves mais um
- Ordem da árvore (Bayer e McCreight, 1972)
 - O número mínimo de chaves representa ordem da árvore
 - Todo nó (exceto a raiz) tem no máximo $M-1$ chaves e no mínimo $M/2$ (50%) de ocupação
 - Nós intermediários e nós folhas devem seguir esta taxa de ocupação
 - Essa característica é válida para $M > 4$, já que se com $M \leq 4$, quando dividir sempre ficará uma chave única em mais de um nó.

Conceitos da árvores B

- Propriedades da árvore
 - Todos os nós folhas estão no mesmo nível
 - A árvore cresce para a raiz, como a árvore 2-3
 - Uma árvore binária típica cresce nas folhas
 - As chaves de um nó devem estar ordenados de forma crescente
- Operações em uma árvore B
 - Adicionar uma chave
 - Remover uma chave
 - Localizar uma chave
 - Percorrer a árvore

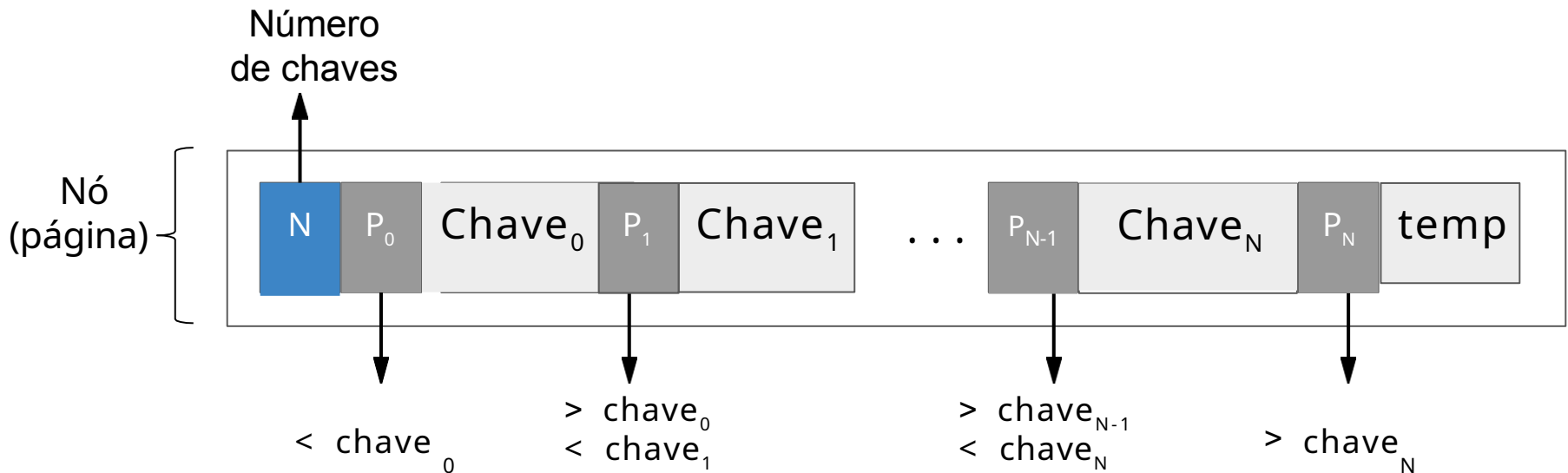
Estrutura da árvores B

- Estrutura do nó
 - Conjunto não vazio de chaves
 - Cada chave é precedida por um ponteiro para um nó filho
 - As chaves de registros são usualmente códigos numéricos



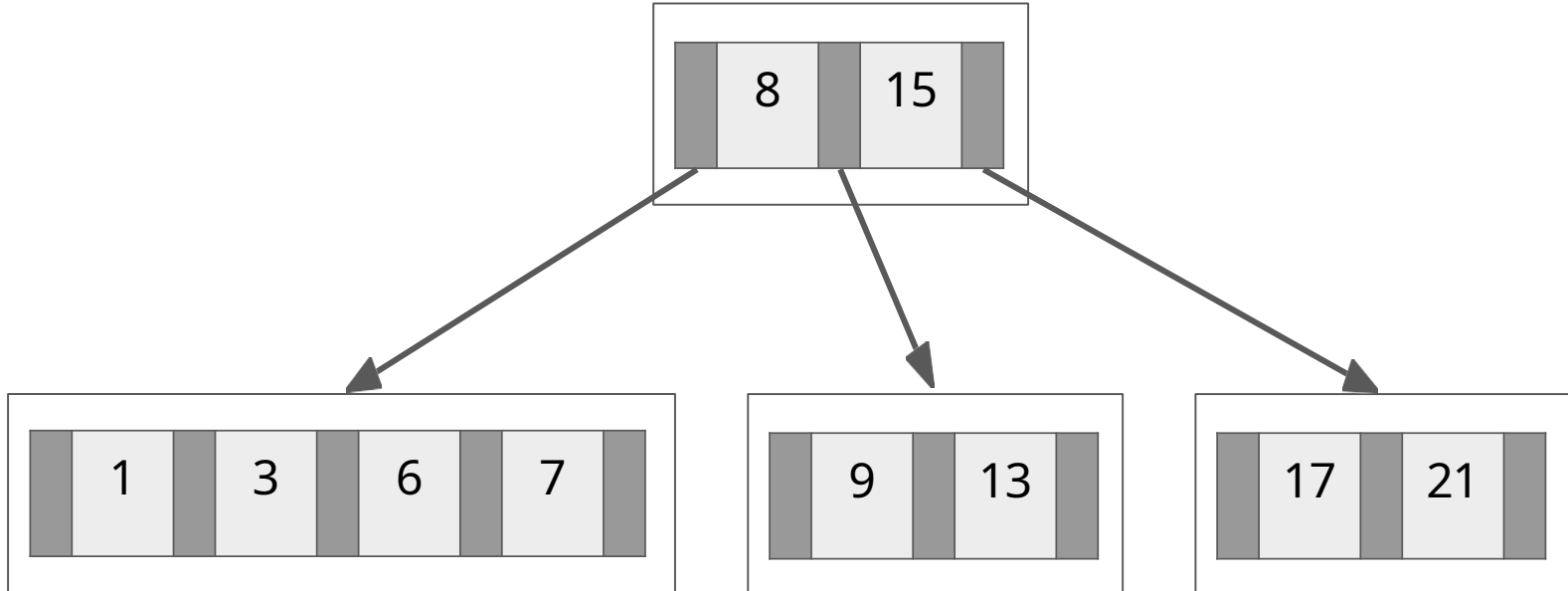
Estrutura da árvores B

- Estrutura do nó
 - Conjunto não vazio de chaves
 - Cada chave é precedida por um ponteiro para um nó filho
 - As chaves de registros são usualmente códigos numéricos
 - Pode conter um nó temporário (overflow) para uso na divisão



Estrutura da árvores B

- Estrutura da árvore
 - Nó raiz é a exceção para quantidade mínima de chaves
 - Os filhos de um nó é igual ao número de chaves mais um
 - Todos os nós folhas estão no mesmo nível da árvore



Operações em árvores B

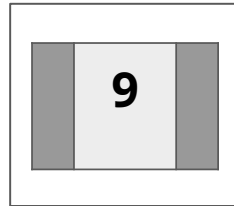
- Adicionar novas chaves
 - Inicia localizando o nó que deverá ser adicionada a chave
 - Use uma busca tradicional, porém, retorna o nó ao invés da chave
 - Insere a chave no nó localizado anteriormente
 - Deve prever os limites de chaves em um nó
 - Transbordo ou *overflow*: número de chaves excede a capacidade
 - Se ocorrer transbordo, deve ser realizado uma divisão do nó
 - Divisão ou *split*: separa as chaves contidas em um nó em dois
 - Cada nó (atual e o novo) possuirão 50% de ocupação
 - Após a divisão, a chave central da divisão deve ser promovida
 - Esta adição da chave promovida deve ser recursiva
 - Caso a divisão ocorrer na raiz, será adicionado um novo nível na árvore

Estrutura da árvores B

- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, 7, 8, 5, 1

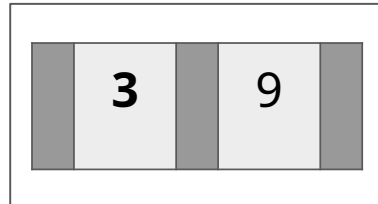
Estrutura da árvores B

- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: **9**, 3, 11, 7, 8, 5, 1



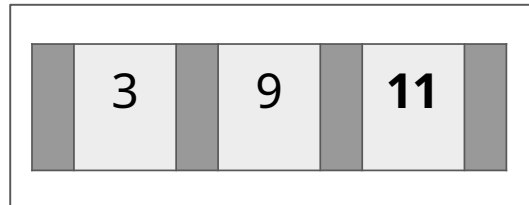
Estrutura da árvores B

- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, **3**, 11, 7, 8, 5, 1



Estrutura da árvores B

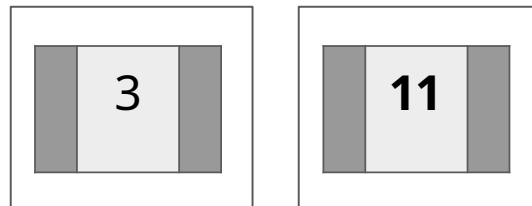
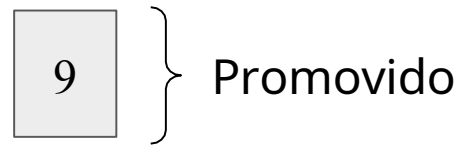
- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, **11**, 7, 8, 5, 1



Overflow

Estrutura da árvores B

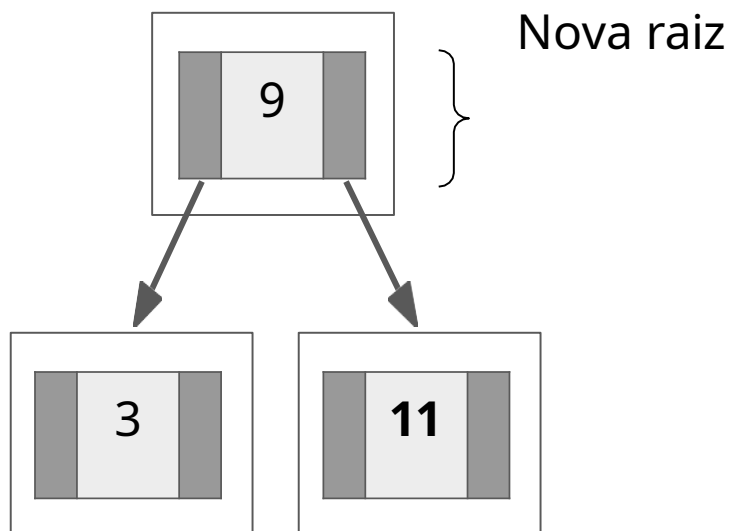
- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, **11**, 7, 8, 5, 1



Split

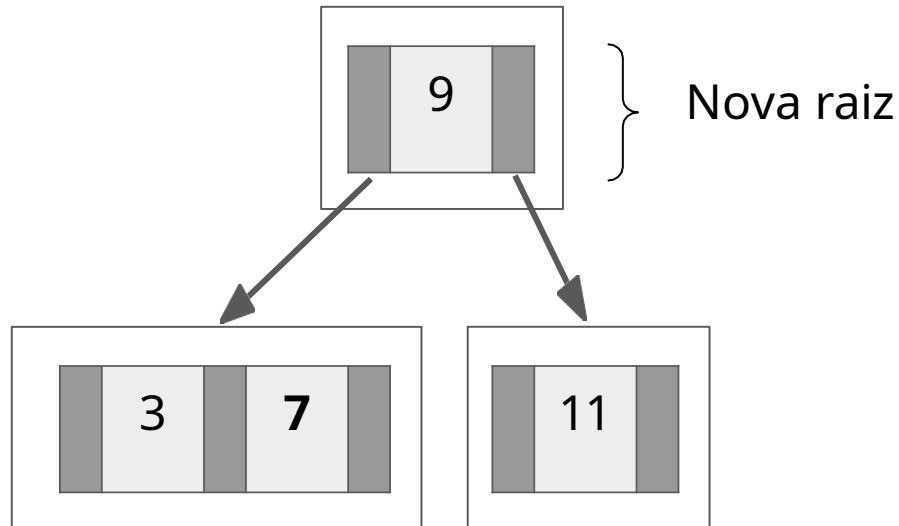
Estrutura da árvores B

- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, **11**, 7, 8, 5, 1



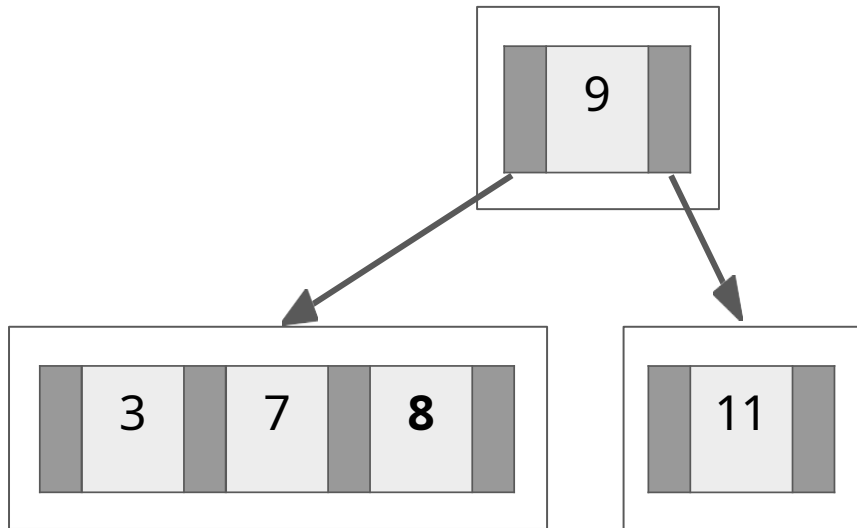
Estrutura da árvores B

- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, **7**, 8, 5, 1



Estrutura da árvores B

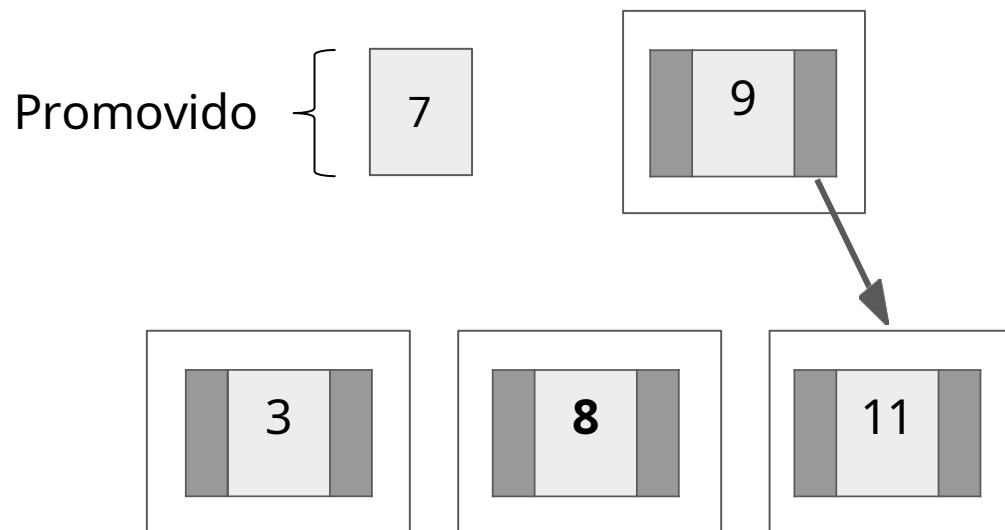
- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, 7, **8**, 5, 1



Overflow

Estrutura da árvores B

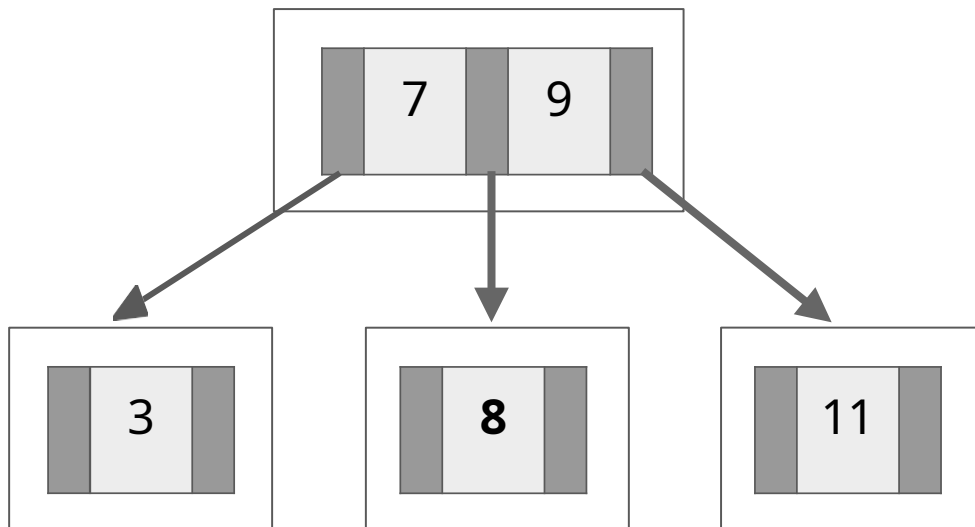
- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, 7, **8**, 5, 1



Split

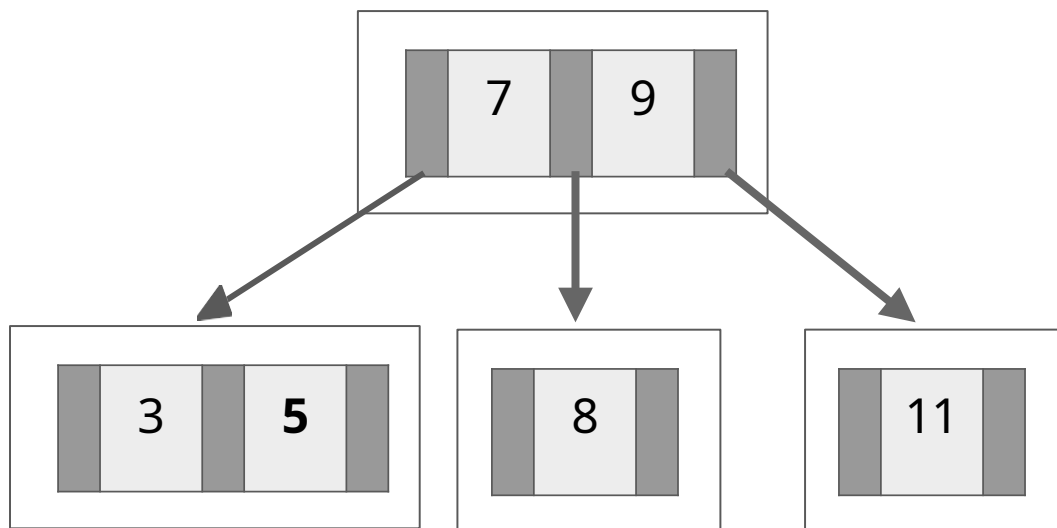
Estrutura da árvores B

- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, 7, **8**, 5, 1



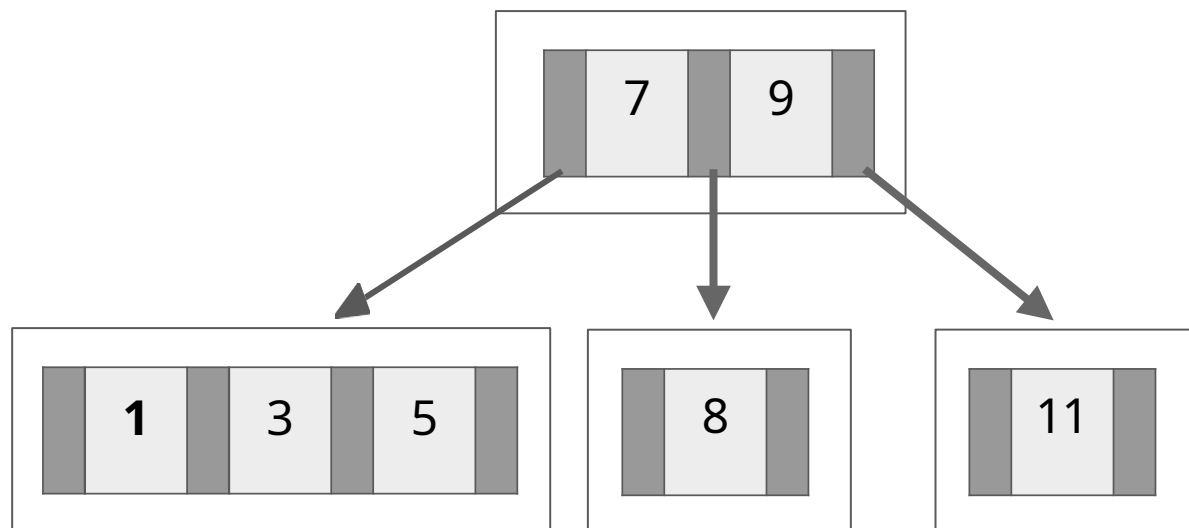
Estrutura da árvores B

- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, 7, 8, **5**, 1



Estrutura da árvores B

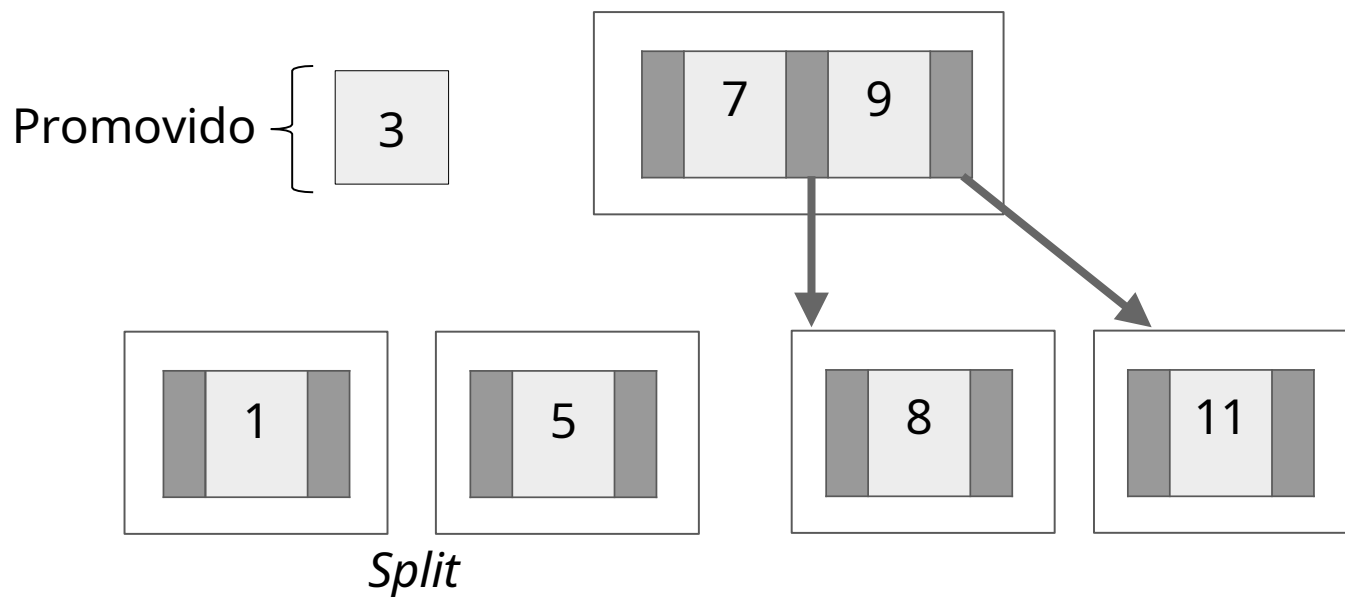
- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, 7, 8, 5, **1**



Overflow

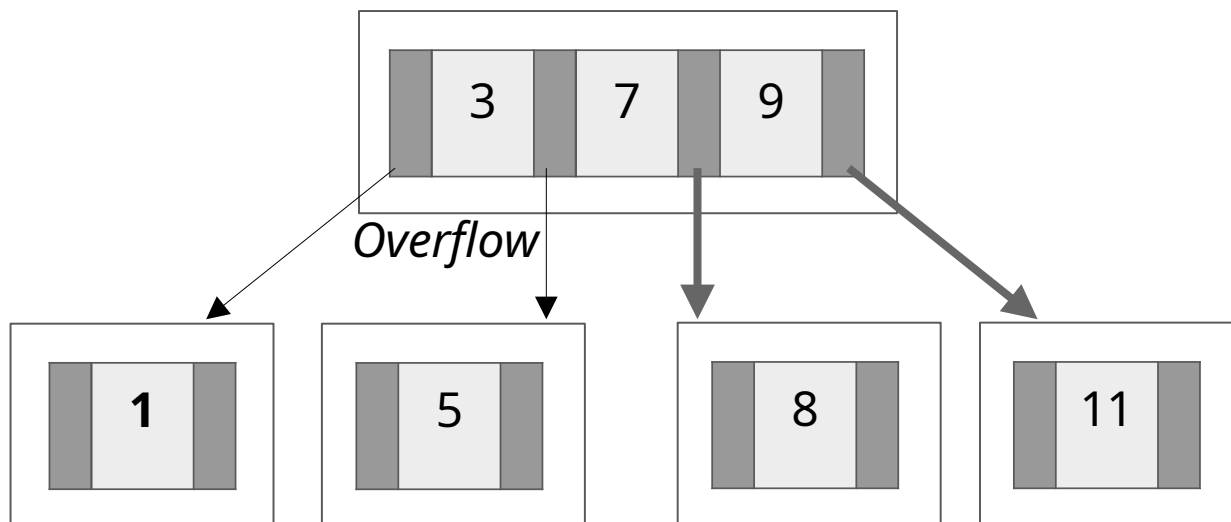
Estrutura da árvores B

- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, 7, 8, 5, **1**



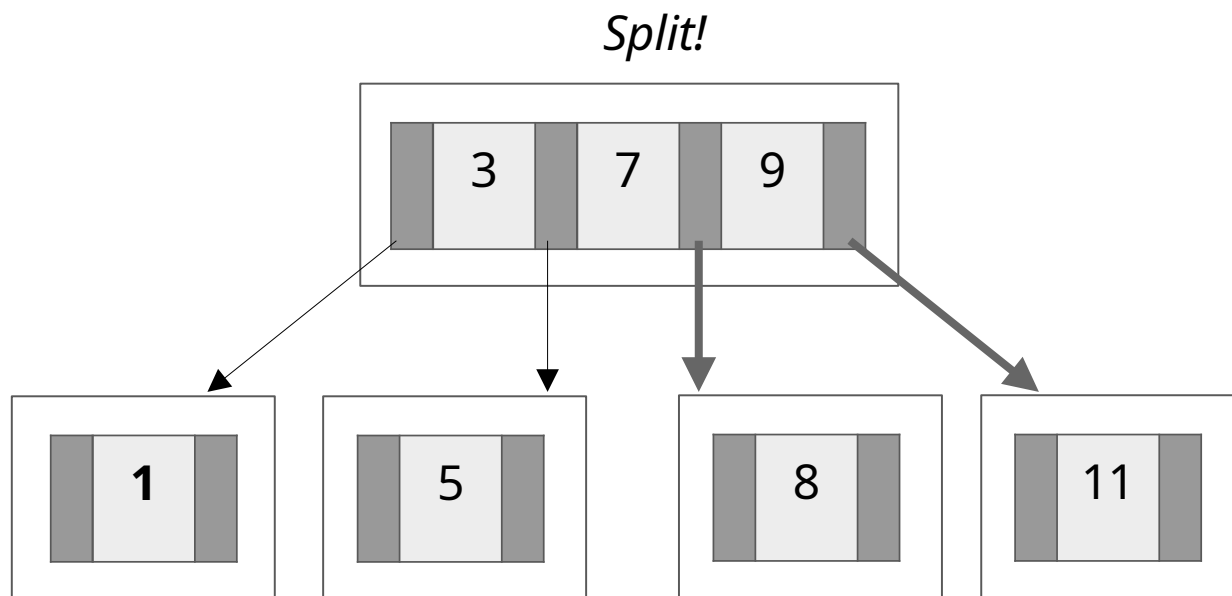
Estrutura da árvores B

- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, 7, 8, 5, **1**



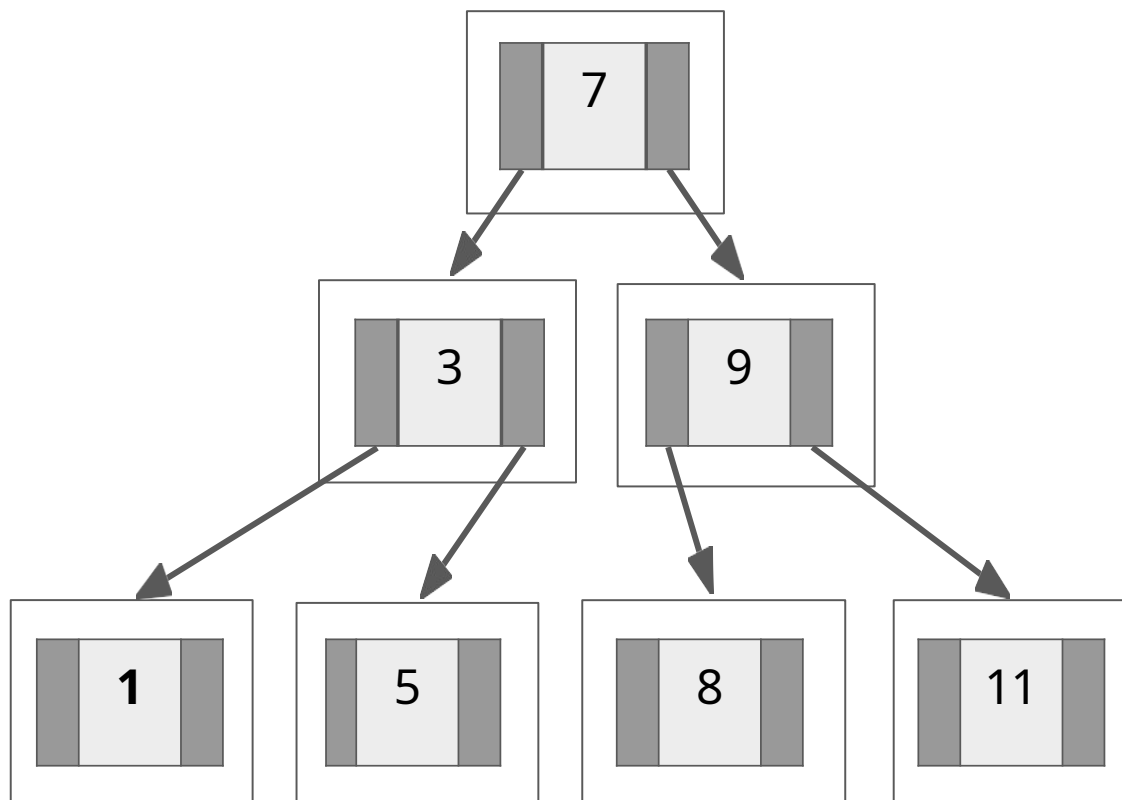
Estrutura da árvores B

- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, 7, 8, 5, **1**



Estrutura da árvores B

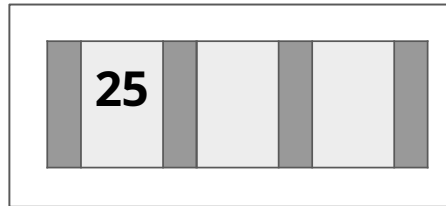
- Exemplo de adição de chaves em uma árvore de ordem $M=2$
 - Chaves: 9, 3, 11, 7, 8, 5, **1**



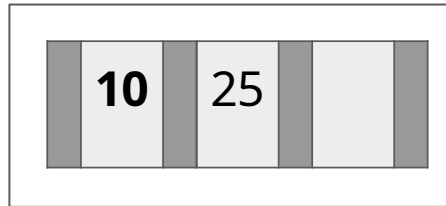
Exemplo

$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, 6, 28, 11, 1$

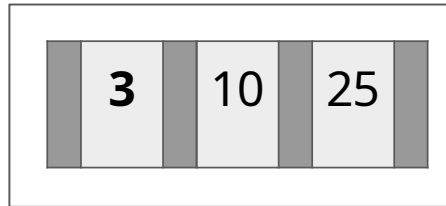
$M = 3 \rightarrow \mathbf{25}, 10, 3, 8, 14, 40, 20, 9, 2, 6, 28, 11, 1$



$M = 3 \rightarrow 25, \mathbf{10}, 3, 8, 14, 40, 20, 9, 2, 6, 28, 11, 1$

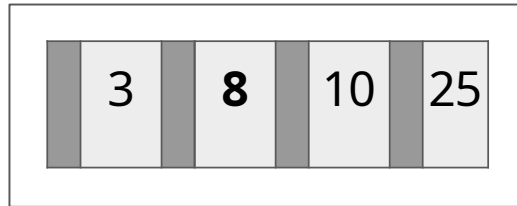


$M = 3 \rightarrow 25, 10, \mathbf{3}, 8, 14, 40, 20, 9, 2, 6, 28, 11, 1$

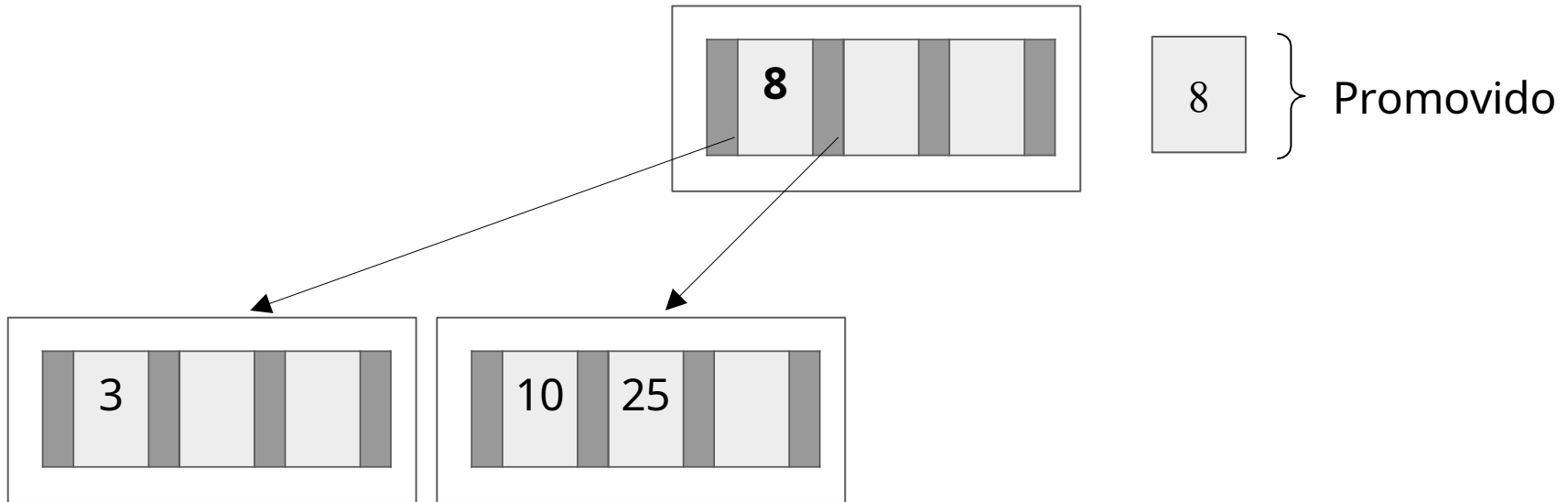


$M = 3 \rightarrow 25, 10, 3, \mathbf{8}, 14, 40, 20, 9, 2, 6, 28, 11, 1$

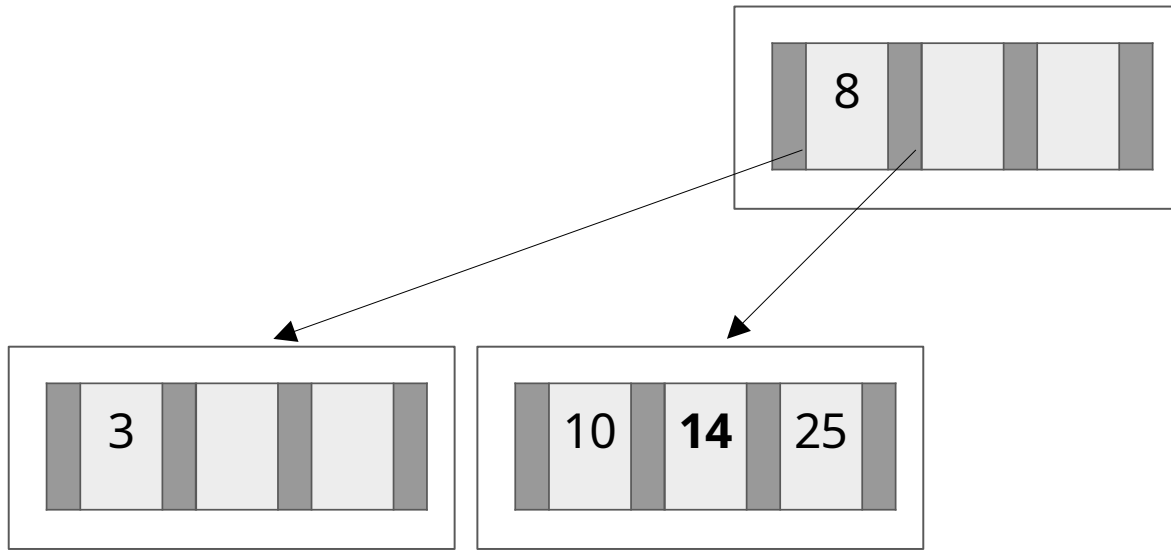
Split!



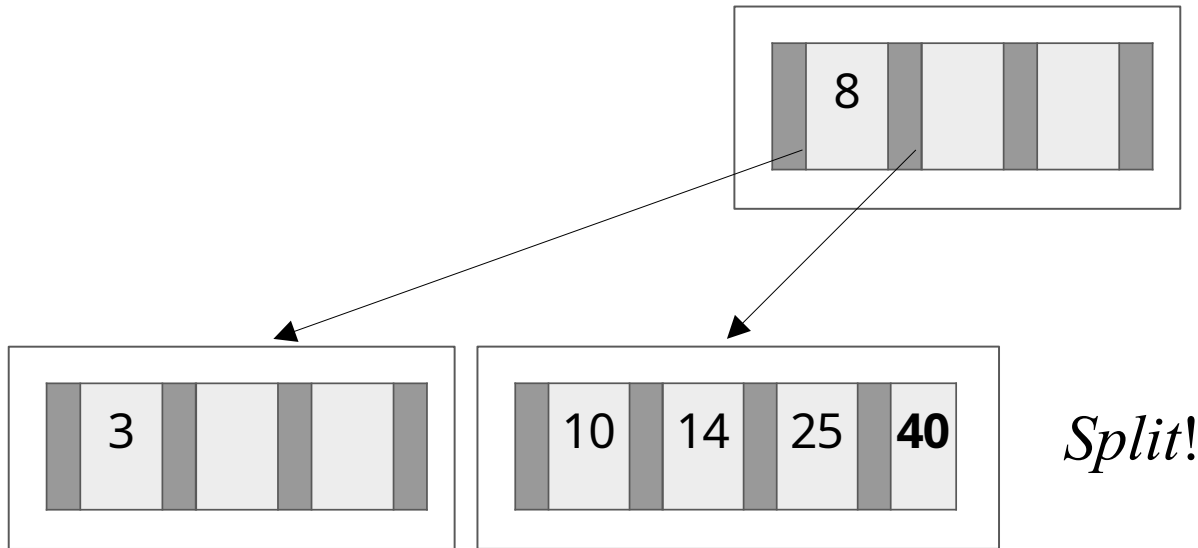
$M = 3 \rightarrow 25, 10, 3, \mathbf{8}, 14, 40, 20, 9, 2, 6, 28, 11, 1$



$M = 3 \rightarrow 25, 10, 3, 8, \mathbf{14}, 40, 20, 9, 2, 6, 28, 11, 1$

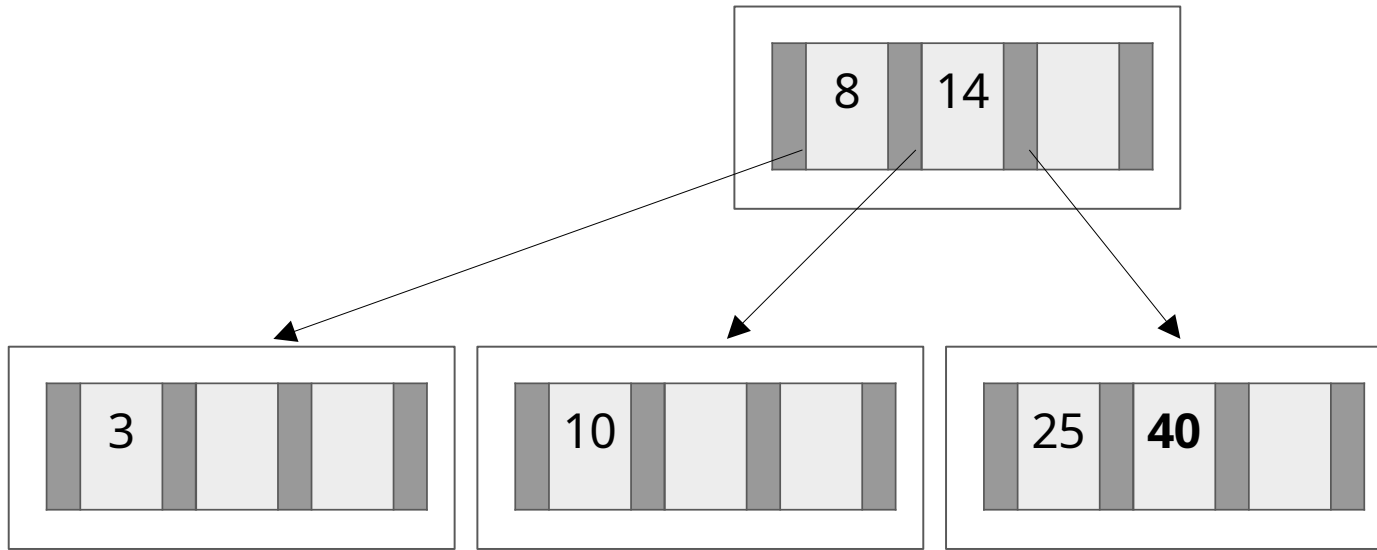


$M = 3 \rightarrow 25, 10, 3, 8, 14, \mathbf{40}, 20, 9, 2, 6, 28, 11, 1$

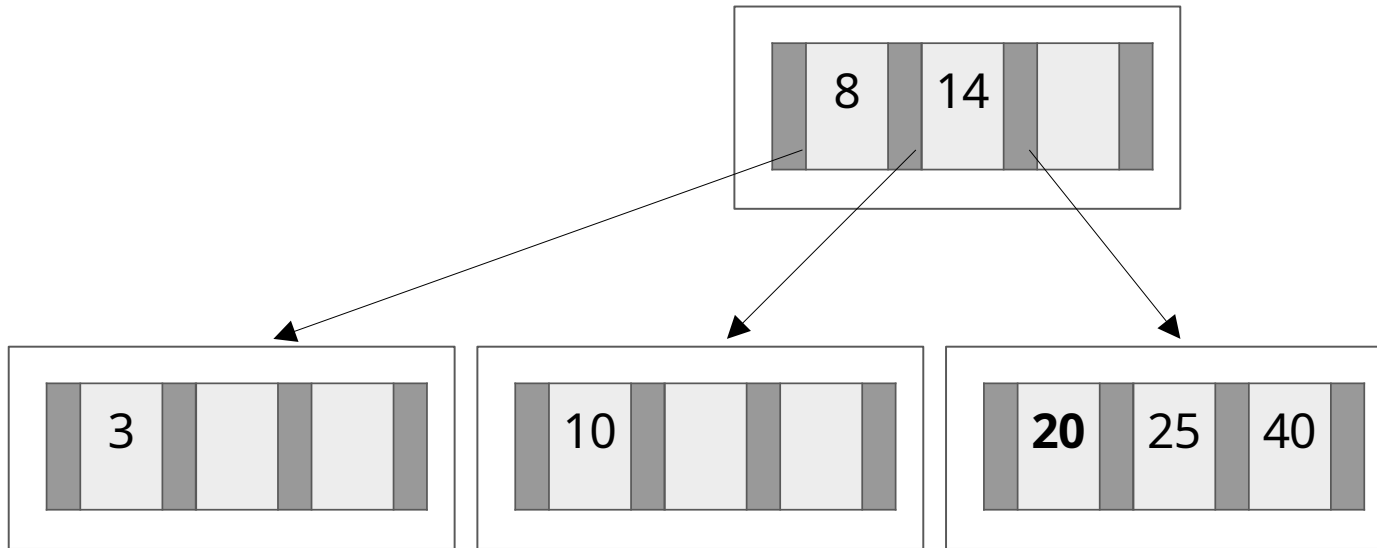


Split!

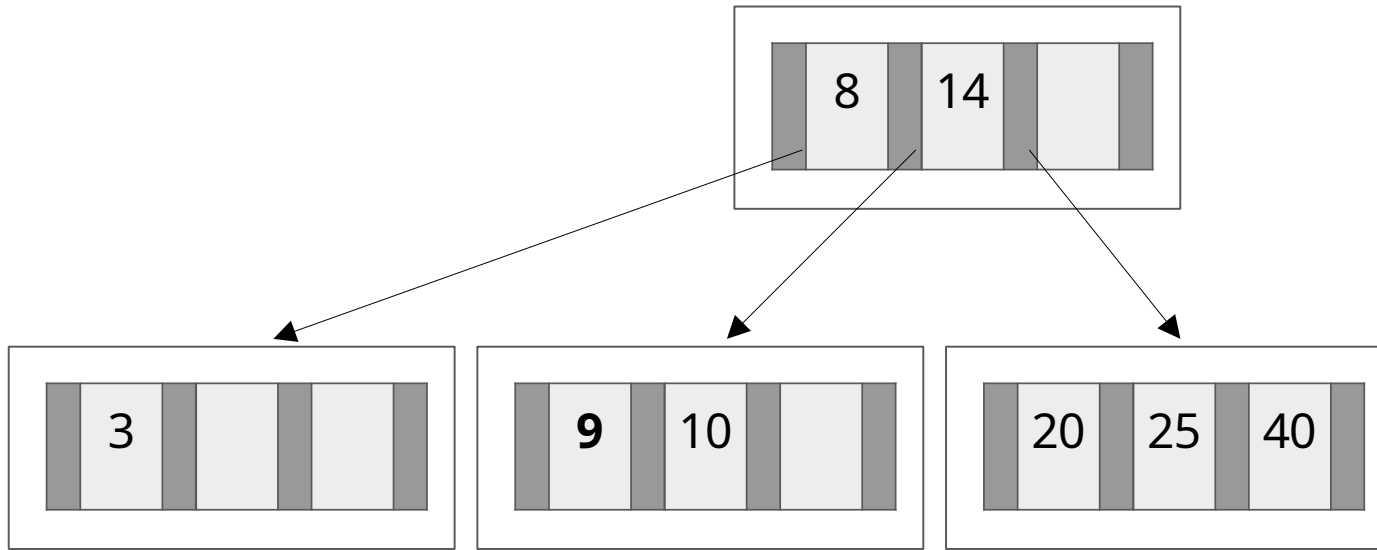
$M = 3 \rightarrow 25, 10, 3, 8, 14, \mathbf{40}, 20, 9, 2, 6, 28, 11, 1$



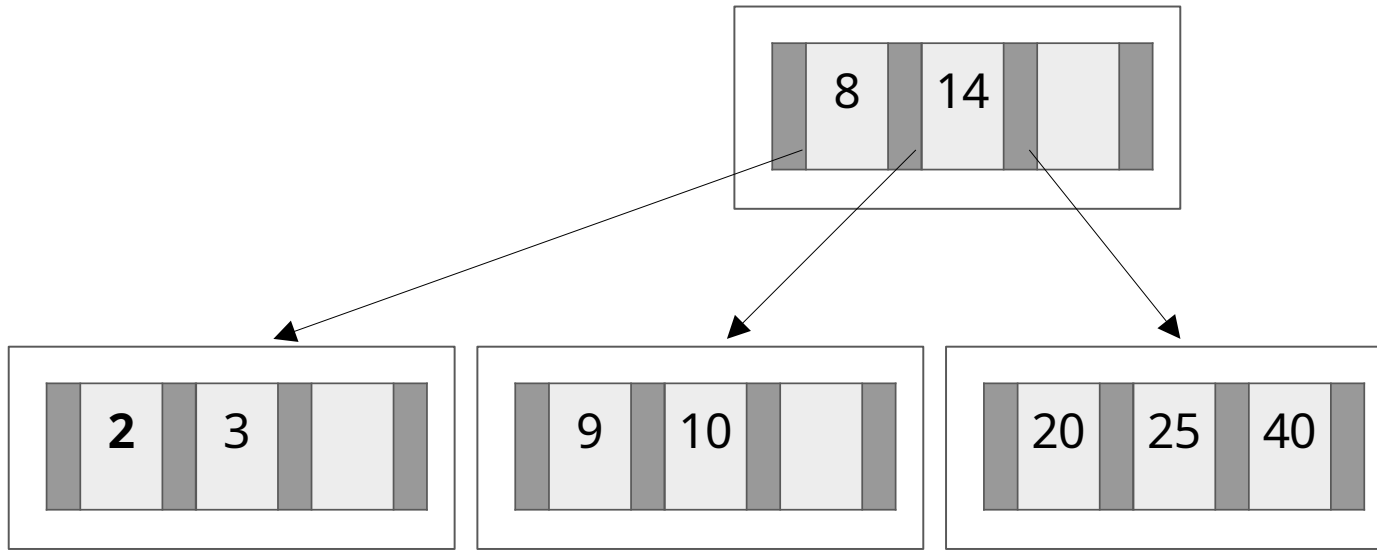
$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, \mathbf{20}, 9, 2, 6, 28, 11, 1$



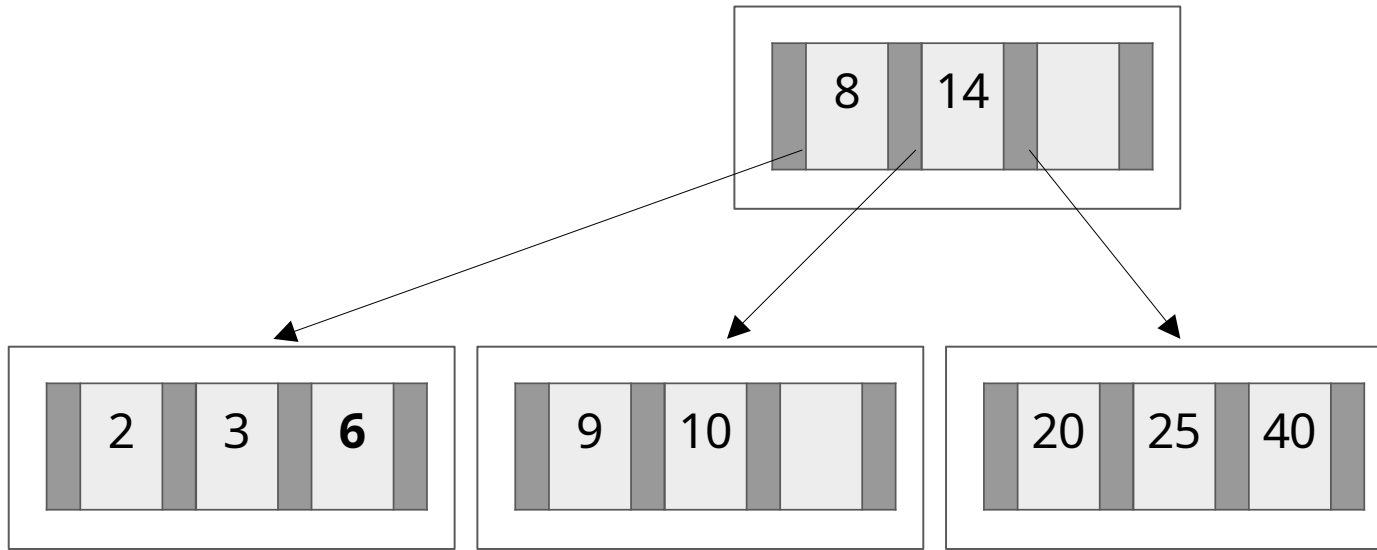
$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, \mathbf{9}, 2, 6, 28, 11, 1$



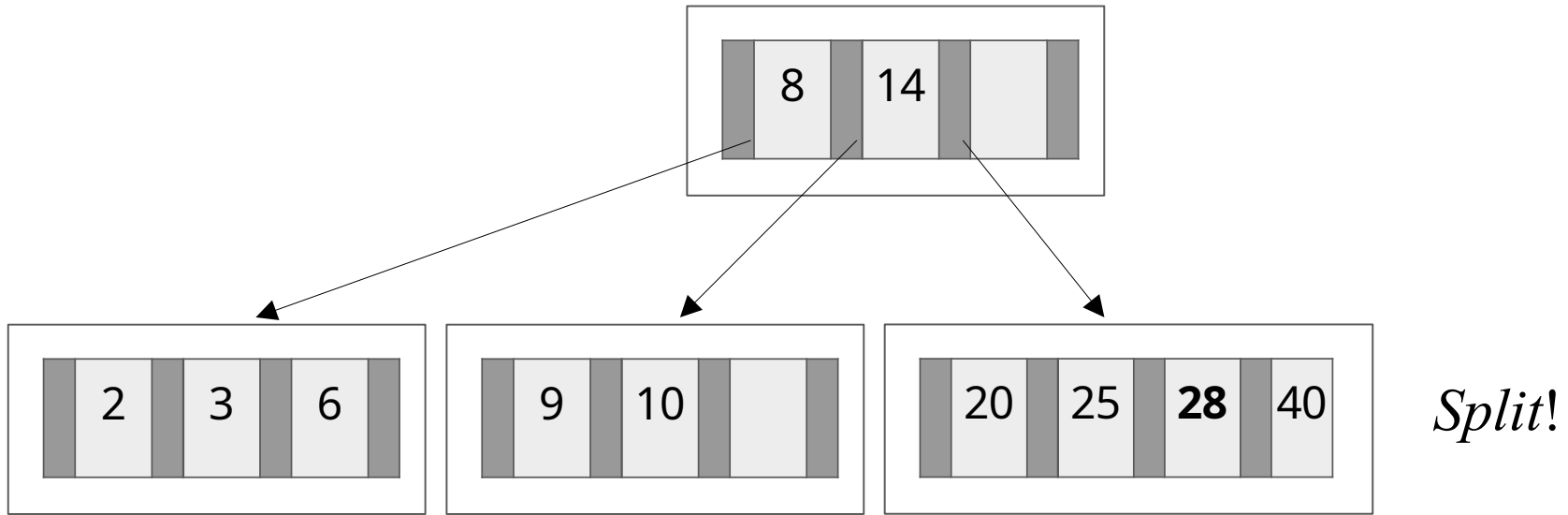
$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, \mathbf{2}, 6, 28, 11, 1$



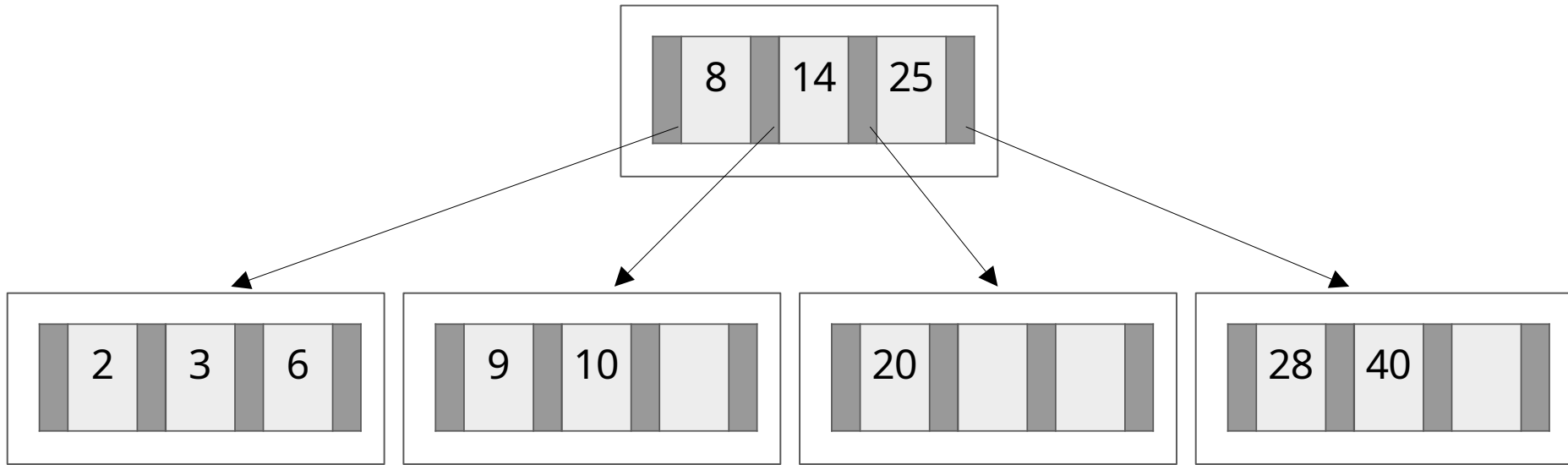
$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, \mathbf{6}, 28, 11, 1$



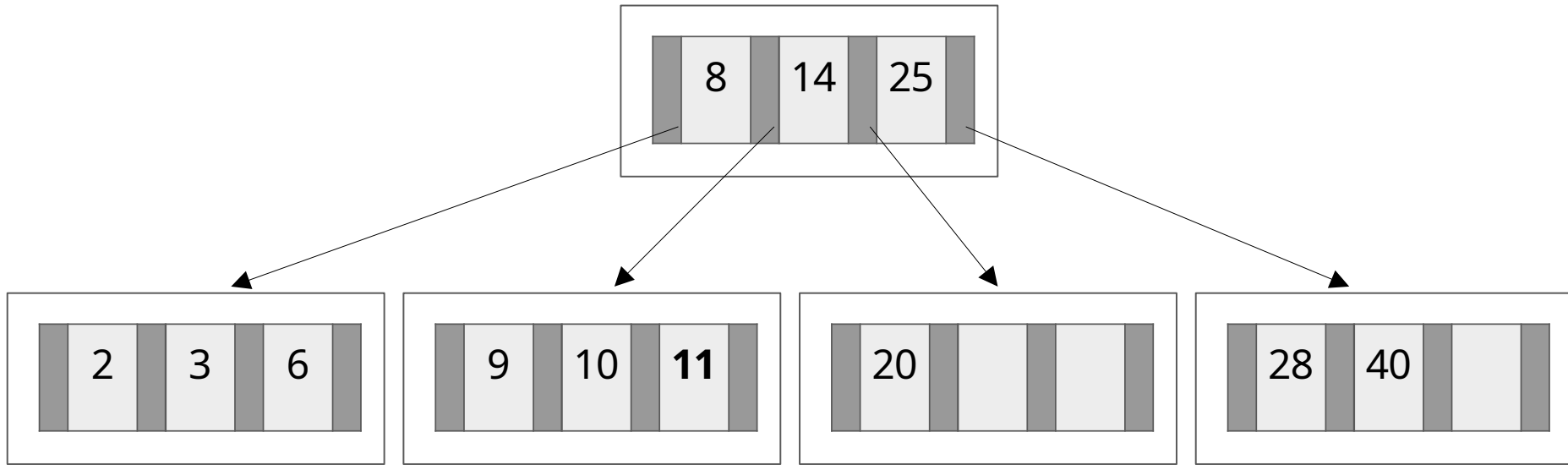
$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, 6, \mathbf{28}, 11, 1$



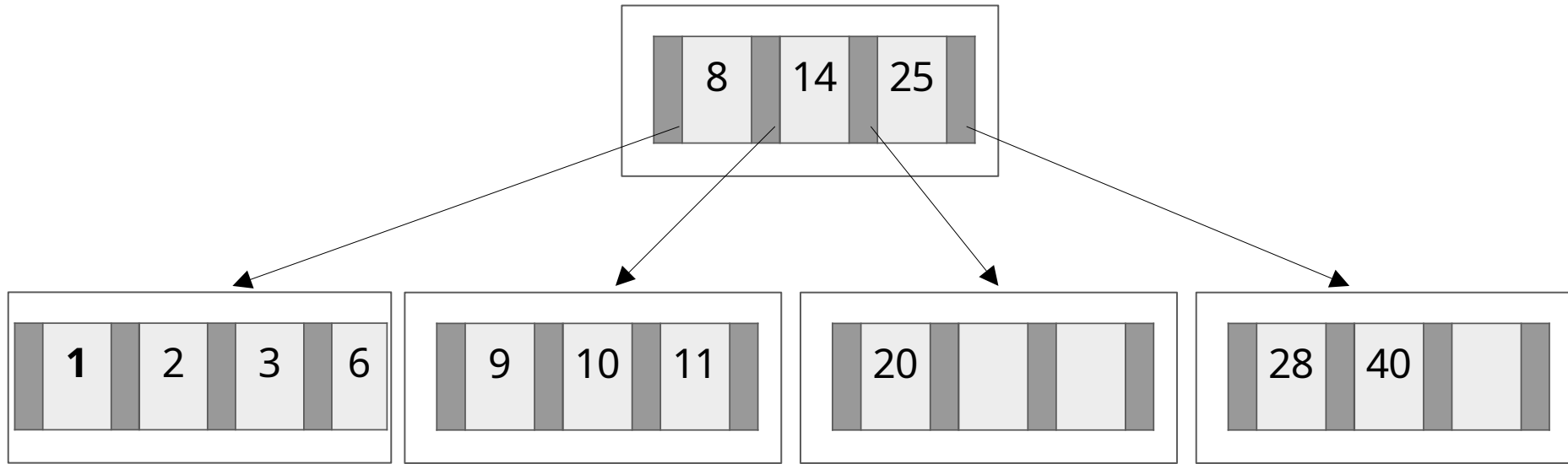
$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, 6, \mathbf{28}, 11, 1$



$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, 6, 28, \mathbf{11}, 1$




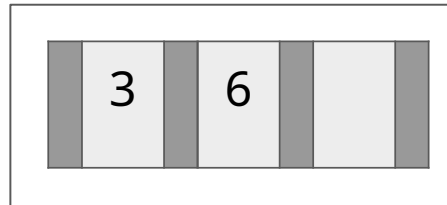
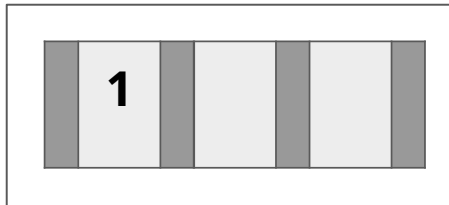
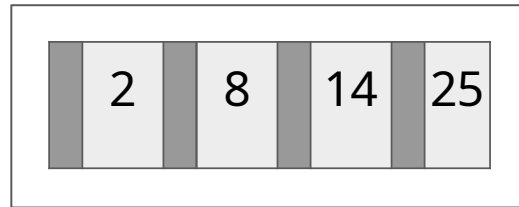
$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, 6, 28, 11, 1$



Split!

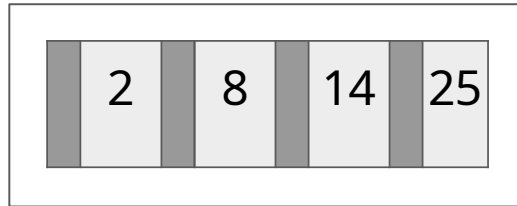
$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, 6, 28, 11, 1$


 } Promovido

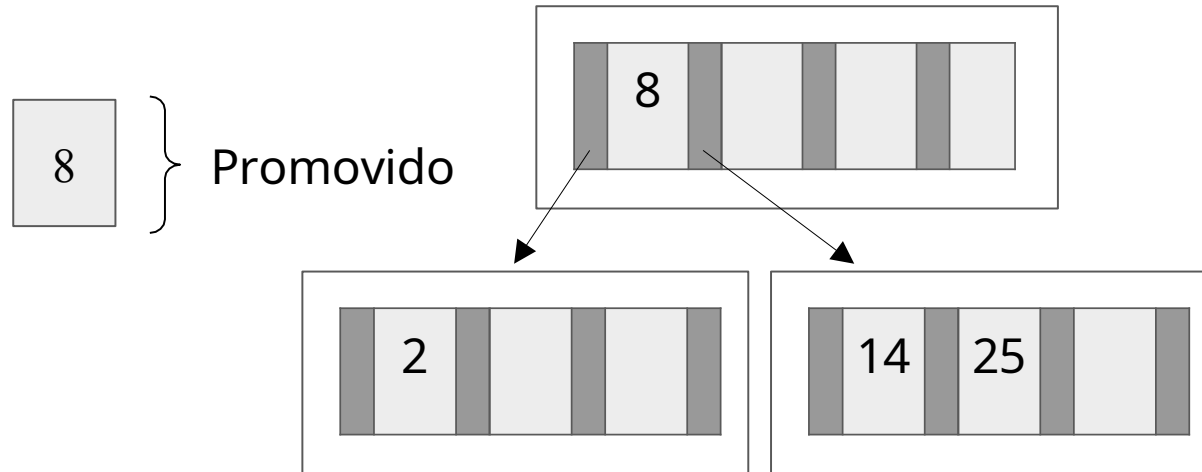


$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, 6, 28, 11, 1$

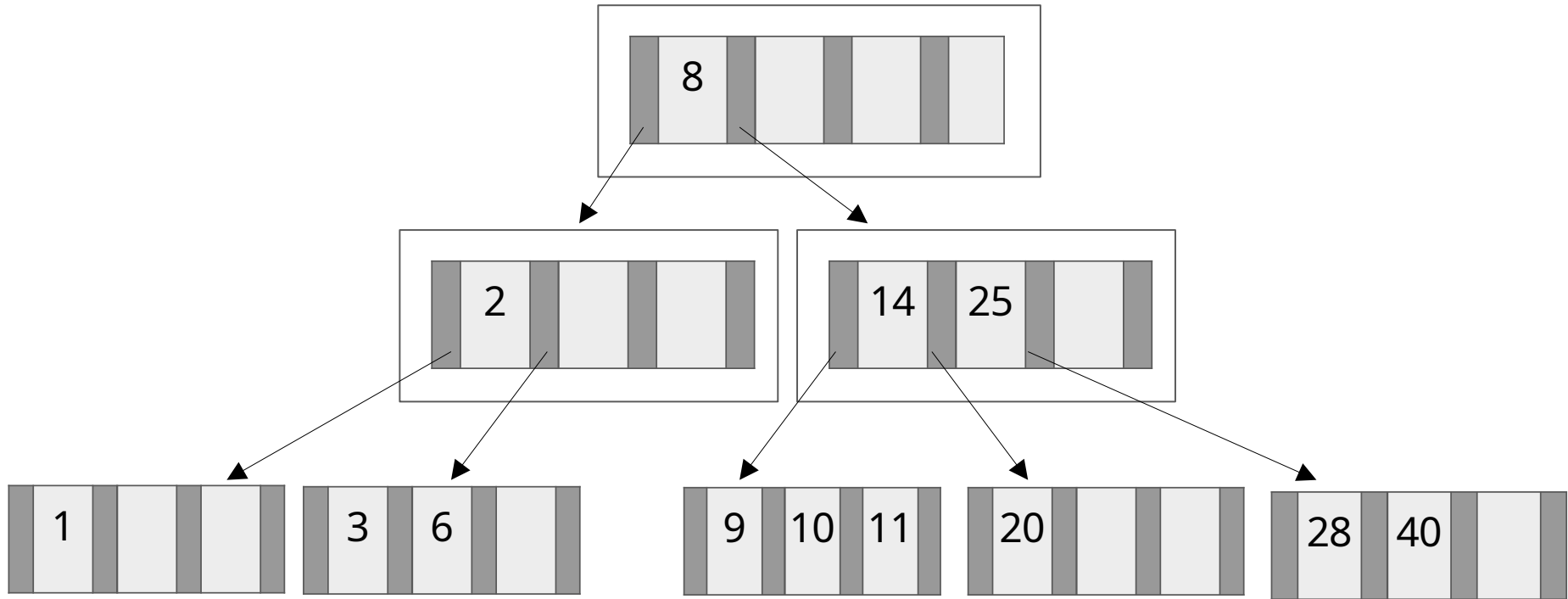
Split!



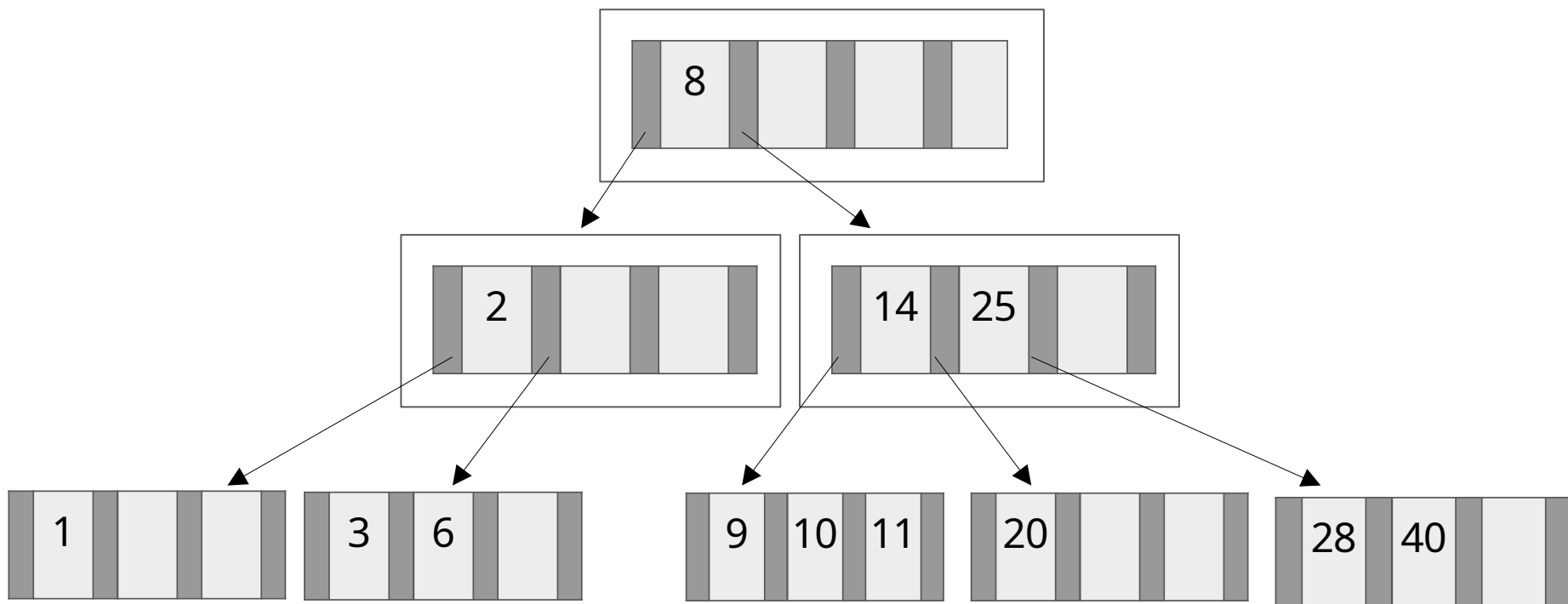
$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, 6, 28, 11, 1$



$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, 6, 28, 11, 1$



$M = 3 \rightarrow 25, 10, 3, 8, 14, 40, 20, 9, 2, 6, 28, 11, 1$



Divididos anteriormente

Estrutura da árvores B

- Estrutura da árvore

```
typedef struct _no {  
    int total;  
    int* chaves;  
    struct _no** filhos;  
    struct _no* pai;  
} No;
```

```
struct arvoreB {  
    No* raiz;  
    int ordem;  
} arvore;
```

Operações em árvores B

- Criar uma árvore B com uma ordem predefinida (cont.)

```
No* criaNo() {  
    No* no = malloc(sizeof(No));  
    int max = arvore->ordem * 2;  
    no->pai = NULL;  
    no->chaves = malloc(sizeof(int) * max);  
    no->filhos = malloc(sizeof(No) * max);  
    no->total = 0;  
    for (int i = 0; i < max; i++)  
        no->filhos[i] = NULL;  
    return no;  
}
```

Operações em árvores B

- Percorrer uma árvore a partir de um nó

```
void percorreArvore(No* no, void(visita)(int chave)) {  
    if (no != NULL) {  
        for (int i=0; i<no->total; i++) {  
            percorreArvore(no->filhos[i], visita);  
            visita(no->chaves[i]);  
        }  
        percorreArvore(no->filhos[no->total], visita);  
    }  
}
```

Operações em árvores B

- Localizar uma chave em uma árvore

```
int localizaChave(int chave) {  
    No *no = arvore->raiz;  
    while (no != NULL) {  
        int i= pesquisaBinaria(no, chave);  
        if (i < no->total && no->chaves[i]==chave) {  
            return 1; // encontrou  
        } else {  
            No = no->filhos[i];  
        }  
    }  
    return 0; // não encontrou  
}
```

Operações em árvores B

- Localizar um nó a partir de uma chave

```
No* localizaNo(int chave)  {
    No *no = arvore->raiz;
    while (no != NULL)  {
        int i = pesquisaBinaria(no, chave);
        if (no->filhos[i] == NULL)
            return no; //encontrou nó
        else
            no = no->filhos[i];
    }
    return NULL; //não encontrou nenhum nó
}
```

Operações em árvores B

- Localizar uma chave em uma árvore (cont.)

```
int pesquisaBinaria(No* no, int chave) {
    int inicio = 0, fim = no->total - 1, meio;
    while (inicio <= fim) {
        meio = (inicio + fim) / 2;
        if (no->chaves[meio] == chave) {
            return meio; //encontrou
        } else if (no->chaves[meio] > chave) {
            fim = meio - 1;
        } else {
            inicio = meio + 1;
        }
    }
    return inicio; //não encontrou
}
```

Operações em árvores B

- Adicionar chave em um nó

```
void adicionaChaveNo(No* no, No* direita, int chave) {  
    int i = pesquisaBinaria(no, chave);  
    for (int j = no->total - 1; j >= i; j--) {  
        no->chaves[j + 1] = no->chaves[j];  
        no->filhos[j + 2] = no->filhos[j + 1];  
    }  
    no->chaves[i] = chave;  
    no->filhos[i + 1] = direita;  
    no->total++;  
}
```

Operações em árvores B

- Verificar se houve transbordo (*overflow*) em um nó

```
int transbordo(No *no) {  
    return no->total >= 2 * arvore->ordem;  
}
```


Operações em árvores B

- Dividir chaves (*split*) de um nó em um novo nó

```

No* divideNo(No* no) {
    int meio = no->total / 2;
    No* novo = criaNo();
    novo->pai = no->pai;
    for (int i = meio + 1; i < no->total; i++) {
        novo->filhos[novo->total] = no->filhos[i];
        novo->chaves[novo->total] = no->chaves[i];
        if (novo->filhos[novo->total] != NULL)
            novo->filhos[novo->total]->pai = novo;
        novo->total++;
    }
    novo->filhos[novo->total] = no->filhos[no->total];
    if (novo->filhos[novo->total] != NULL)
        novo->filhos[novo->total]->pai = novo;
    no->total = meio;
    return novo;
}

```

Operações em árvores B

- Adicionar nova chave na árvore

```
void adicionaChave(int chave) {  
    No* no = localizaNo(chave);  
    adicionaChaveRecursivo(no, NULL, chave);  
}
```

Operações em árvores B

- Adicionar nova chave na árvore (cont.)

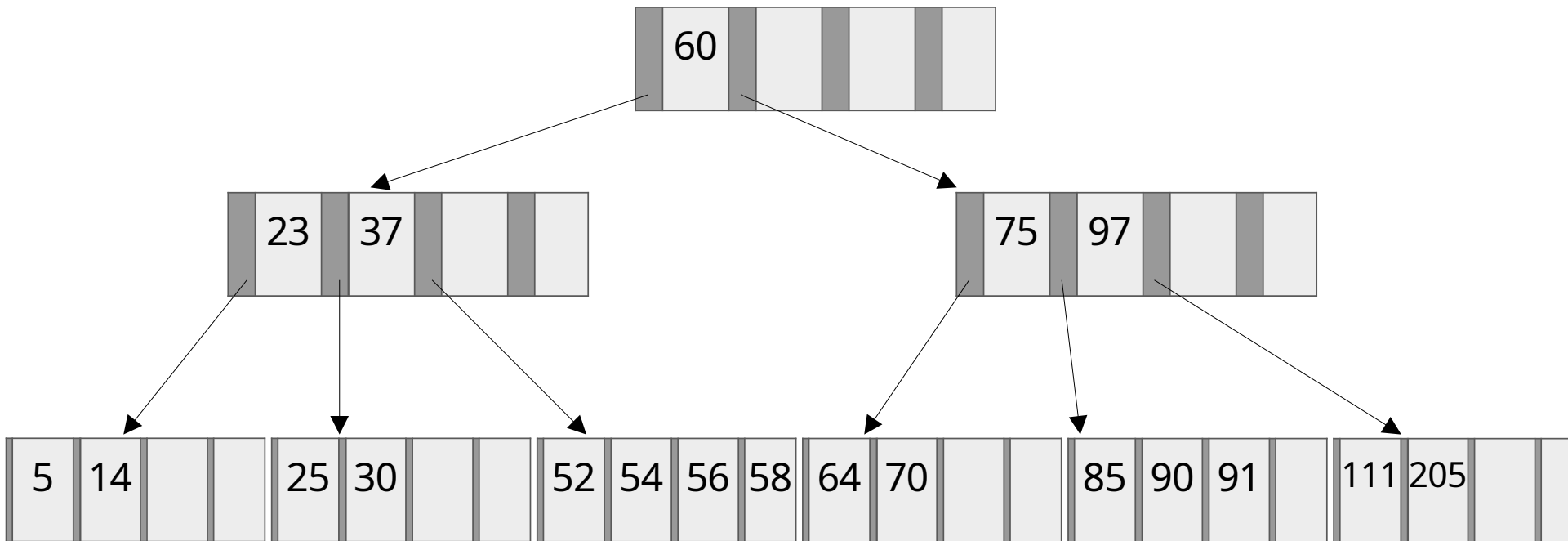
```
void adicionaChaveRecursivo(No* no, No* novo, int chave) {
    adicionaChaveNo(no, novo, chave);
    if (transbordo(no)) {
        int promovido = no->chaves[arvore->ordem];
        No* novo = divideNo(no);
        if (no->pai == NULL) {
            No* raiz = criaNo(); raiz->filhos[0] = no;
            adicionaChaveNo(pai, novo, promovido);
            no->pai = raiz; novo->pai = raiz;
            arvore->raiz = raiz;
        } else
            adicionaChaveRecursivo(no->pai, novo, promovido);
    }
}
```

Operações em árvores B

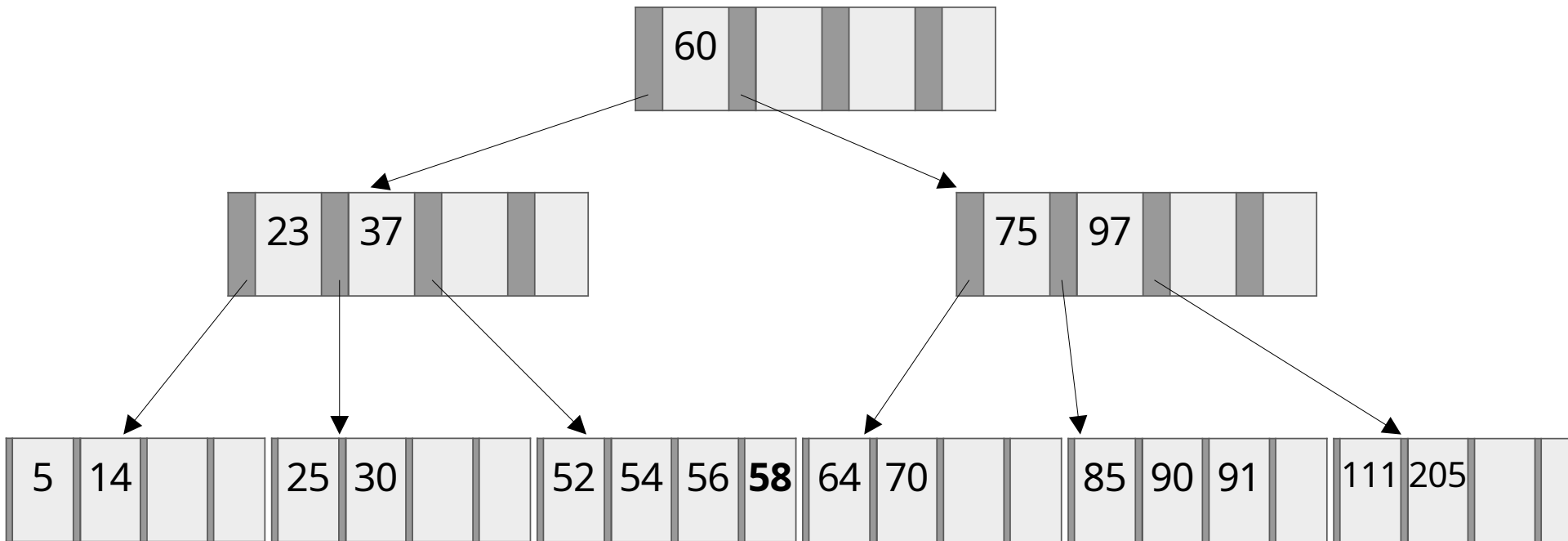
- Remoção

- Caso 1: se elemento estiver na folha e a folha mantém 50% de ocupação, basta remover a chave;
- Caso 2: se o elemento NÃO estiver em folha, tem que ser trocado pelo seu sucessor
 - Não se faz remoção em nós superiores, senão teria que subir algum valor para substituir e a árvore pode ficar desbalanceada
 - Nesse caso, substitui-se pelo valor máximo à esquerda (ou valor mínimo à direita)
- Caso 3: se a folha ficar com menos de 50% de ocupação e a página irmã puder ceder uma chave
 - Normalmente necessário uma rotação para manter os índices válidos
 - Transbordo ou *overflow*: número de chaves excede a capacidade
- Caso 4: se folha ficar com menos de 50% de ocupação e as páginas irmãs não puderem ceder uma chave
 - Faz-se a fusão com o irmão à esquerda ou o irmão à direita

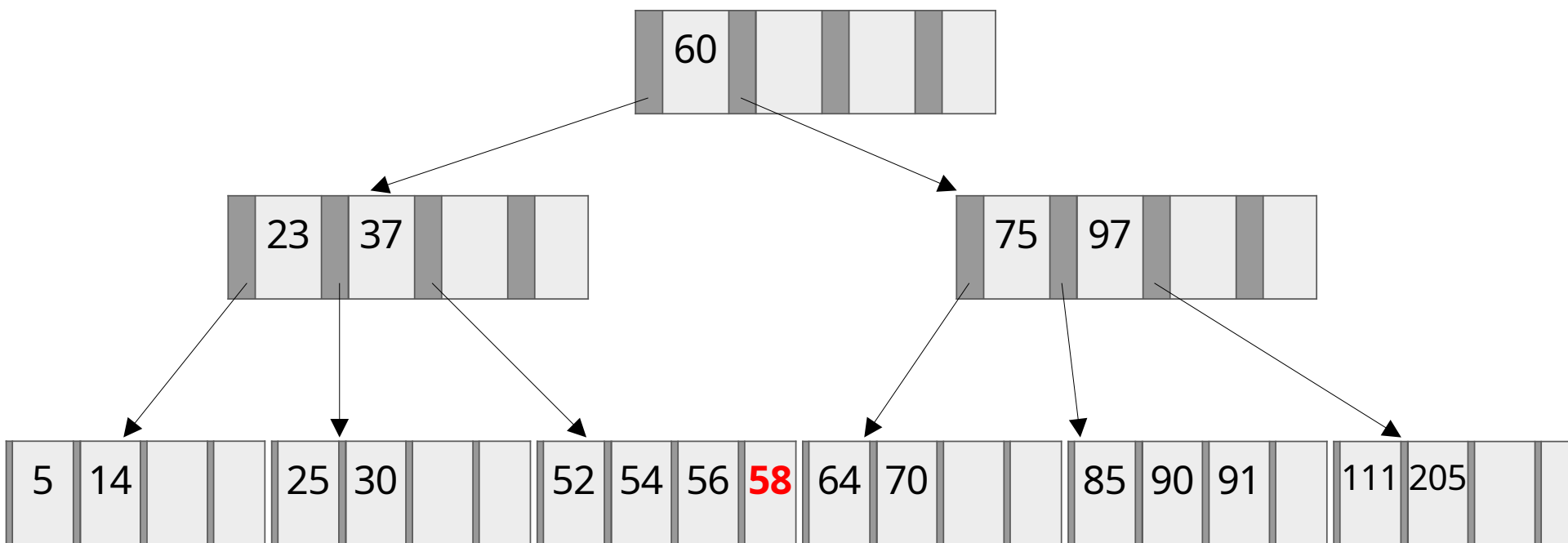
$M = 4$ – ordem 2



$M = 4 \rightarrow$ **Remover 58**

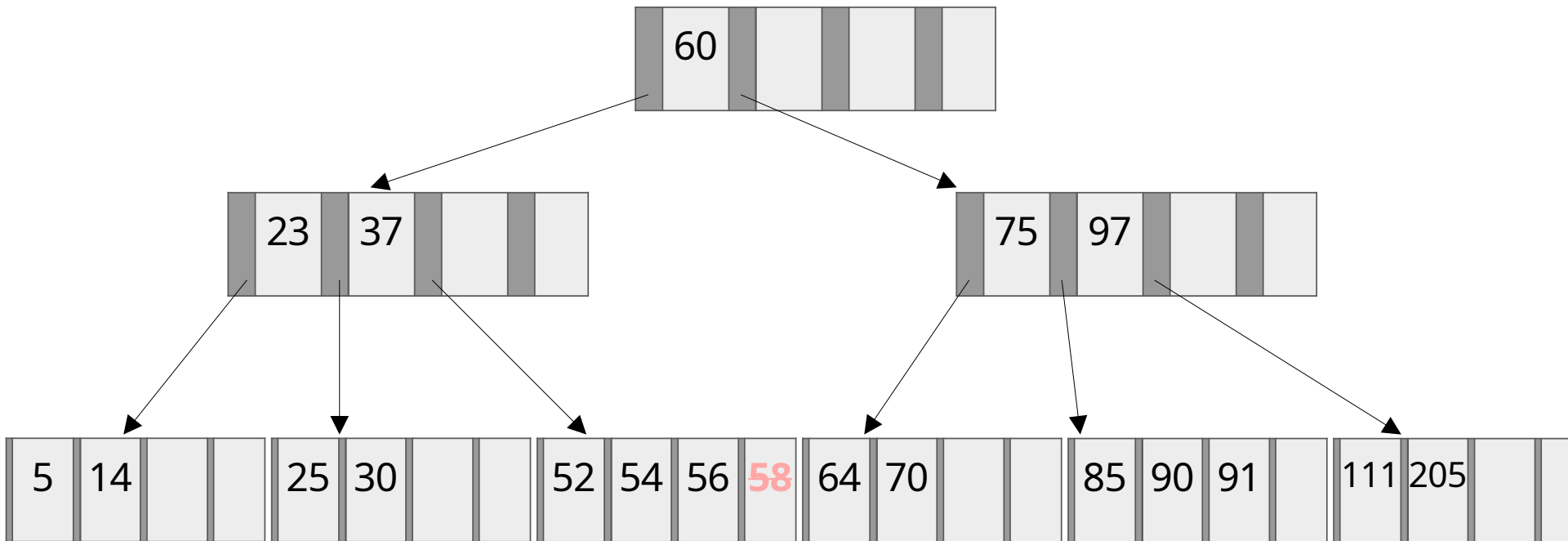


$M = 4 \rightarrow$ Remove 58



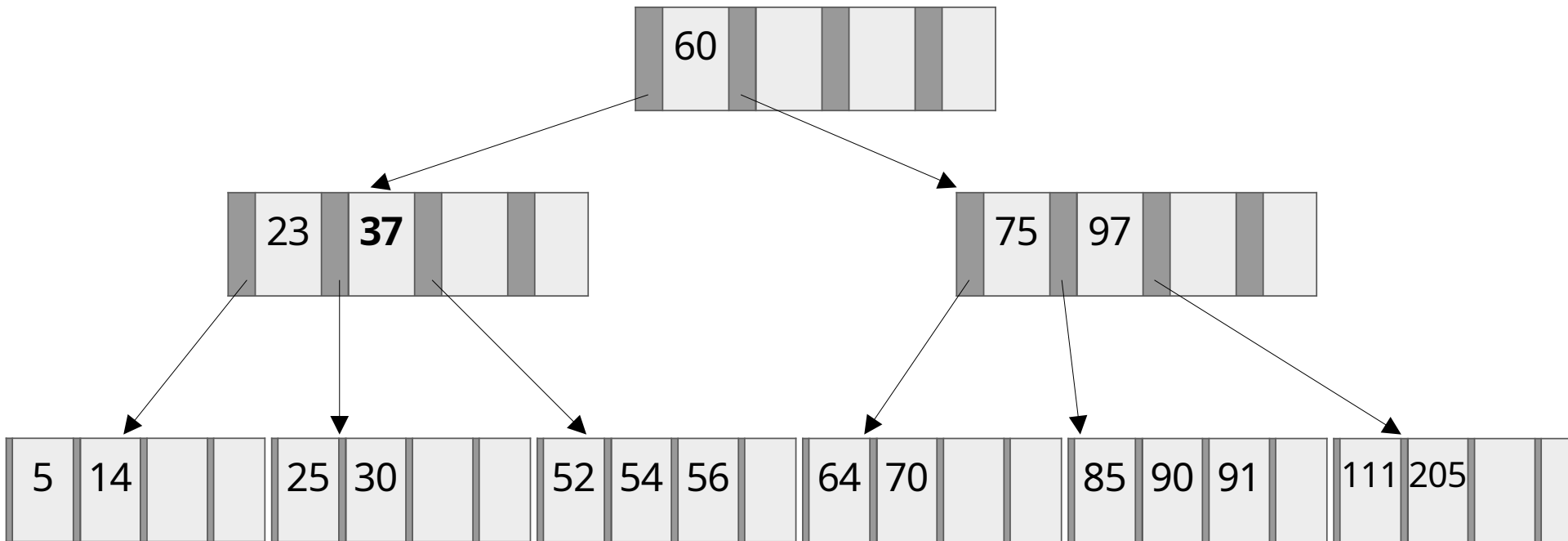
Caso 1: se elemento estiver na folha e a folha mantém 50% de ocupação, basta remover a chave;

$M = 4 \rightarrow$ Remove 58

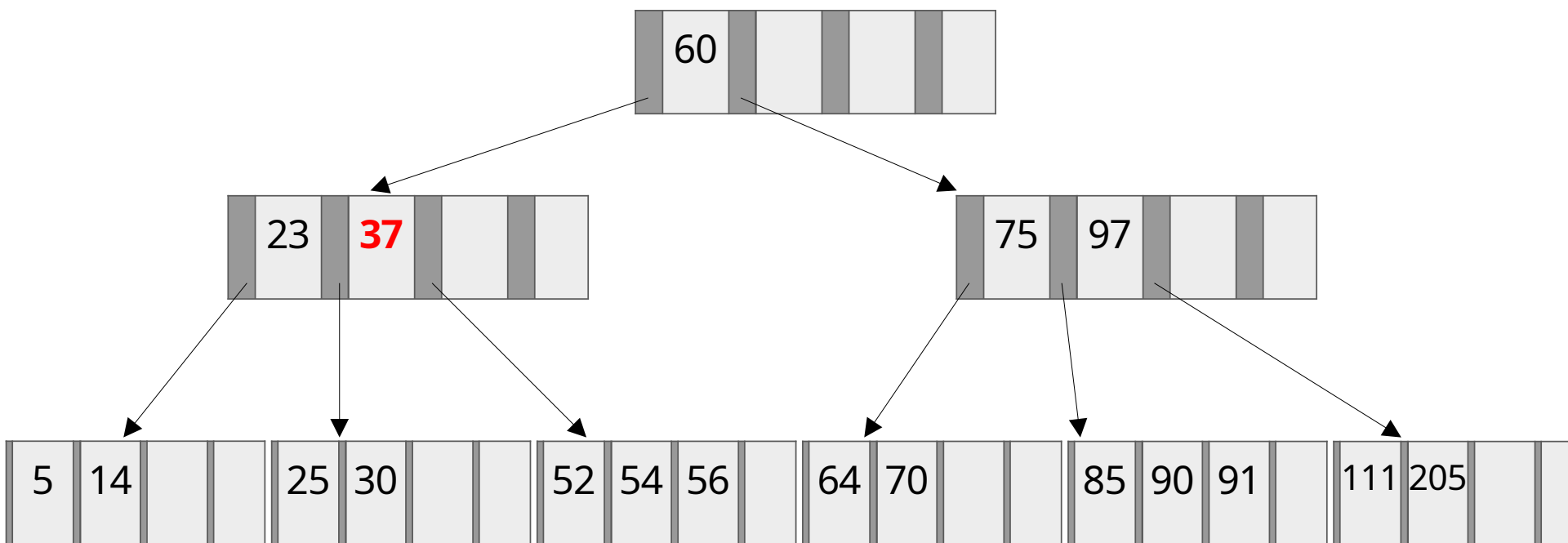


Caso 1: se elemento estiver na folha e a folha mantém 50% de ocupação, basta remover a chave;

$M = 4 \rightarrow$ **Remove 37**



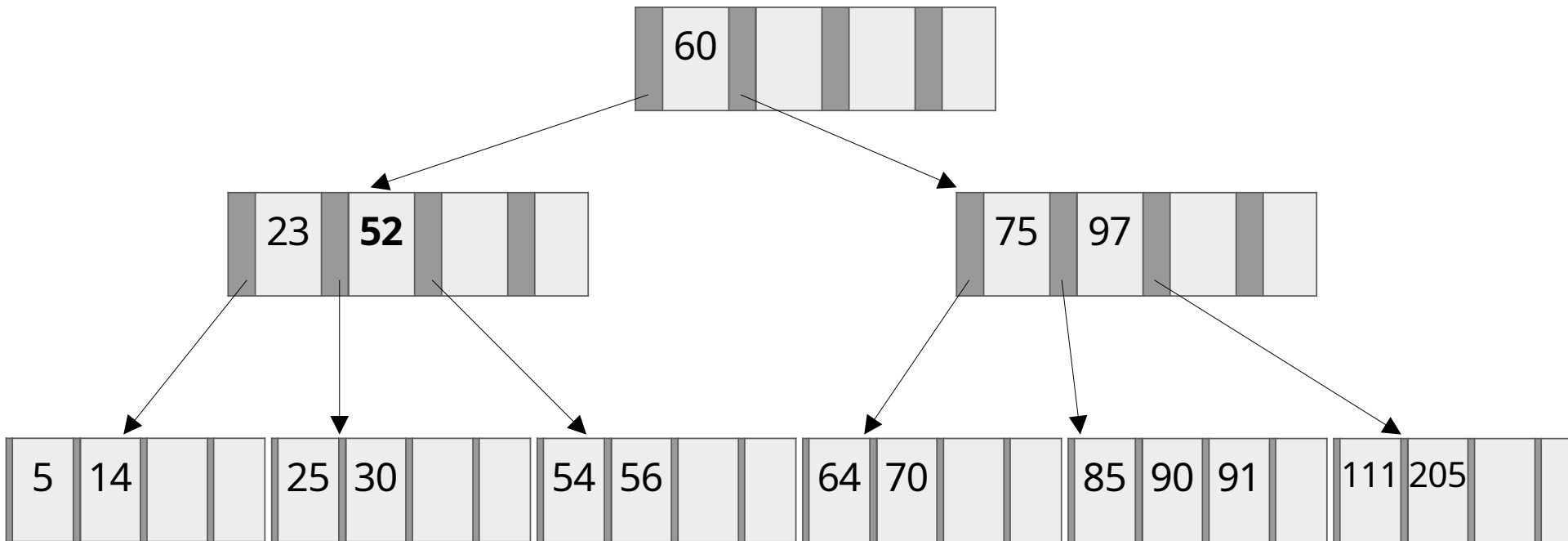
M = 4 → Remove 37



Caso 2: se o elemento NÃO estiver em folha, tem que ser trocado pelo seu sucessor

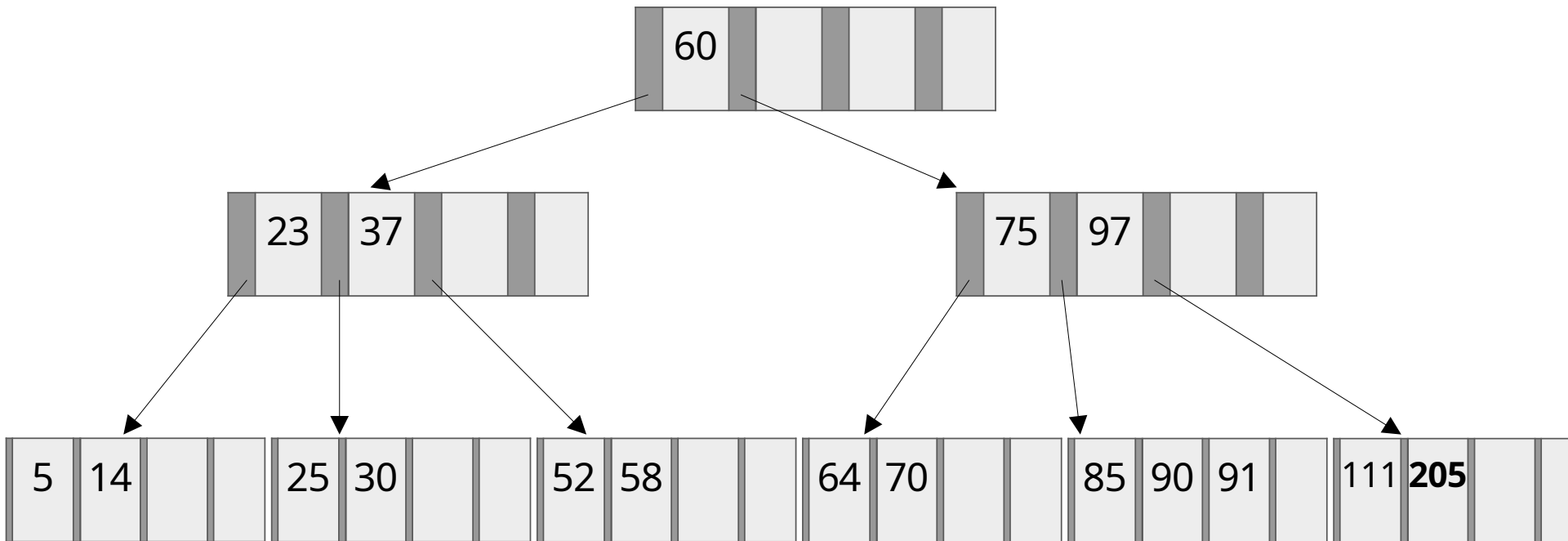
- Não se faz remoção em nós superiores, senão teria que subir algum valor para substituir e a árvore pode ficar desbalanceada
- Nesse caso, substitui-se pelo valor mínimo à direita

$M = 4 \rightarrow$ Remove 37

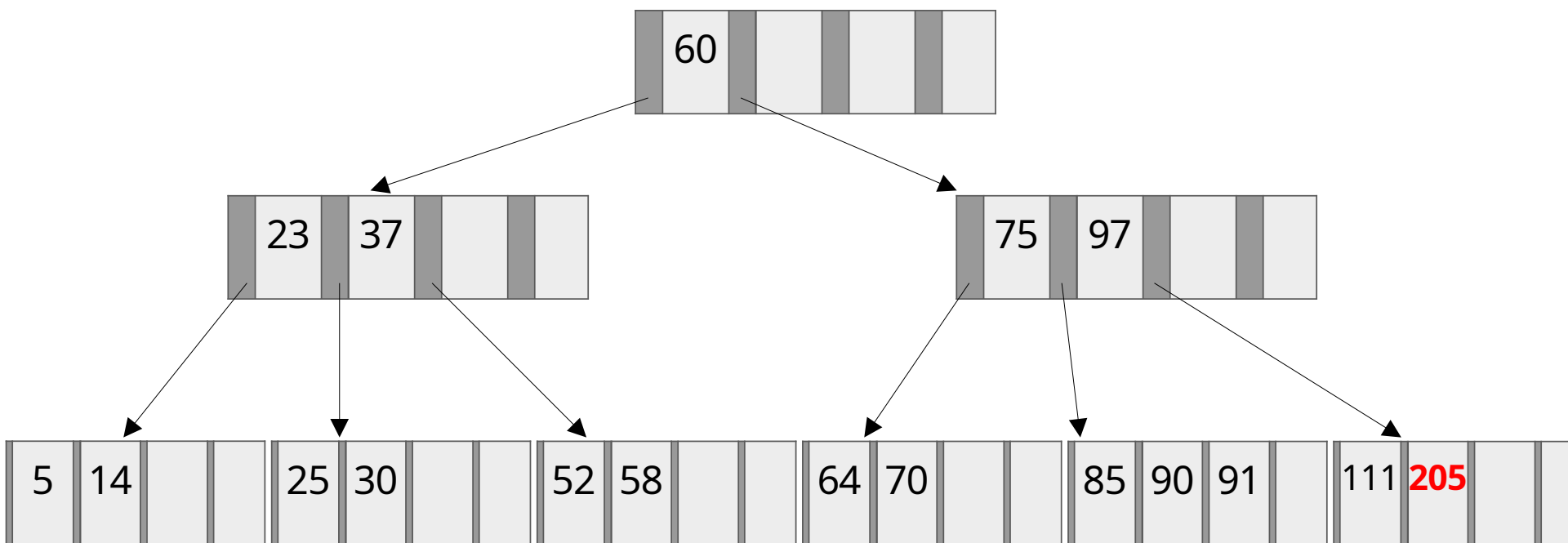


- Caso 2: se o elemento NÃO estiver em folha, tem que ser trocado pelo seu sucessor
- Não se faz remoção em nós superiores, senão teria que subir algum valor para substituir e a árvore pode ficar desbalanceada
 - Nesse caso, substitui-se pelo valor mínimo à direita (pela esquerda ficaria abaixo de 50%)

$M = 4 \rightarrow$ **Remover 205**



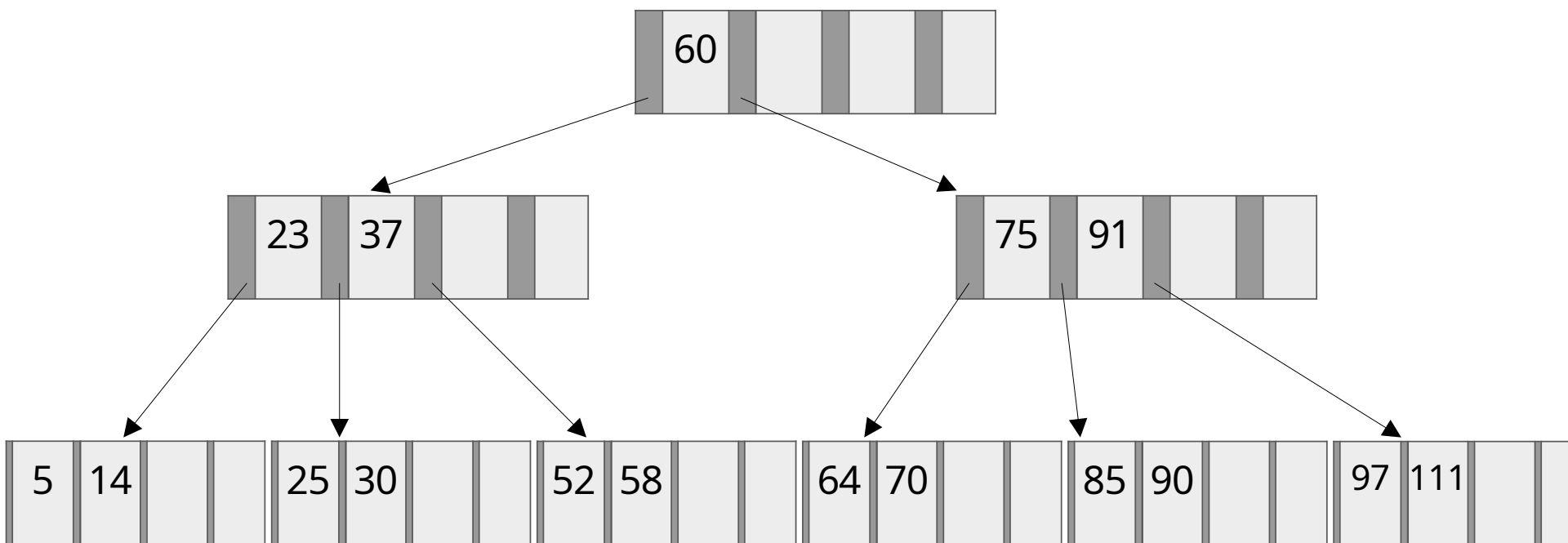
$M = 4 \rightarrow$ Remove 205



Caso 3: se a folha ficar com menos de 50% de ocupação e a página irmã puder ceder uma chave

- Normalmente necessário uma rotação para manter os índices válidos

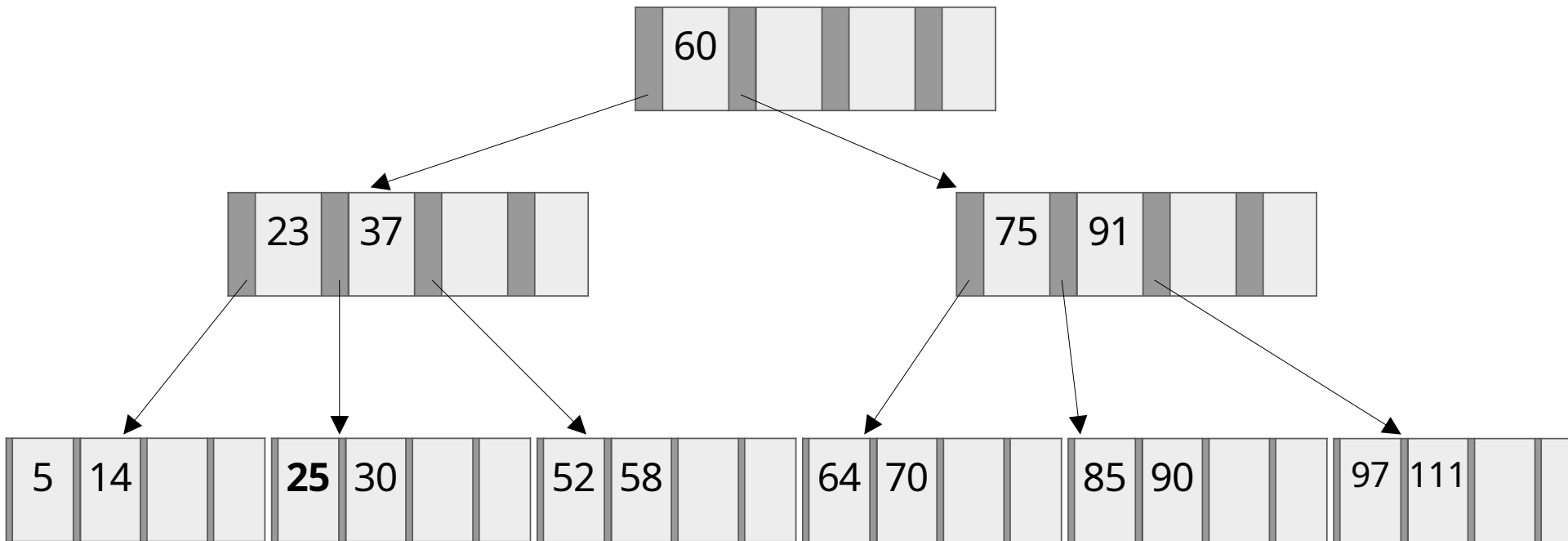
$M = 4 \rightarrow$ Remover 205



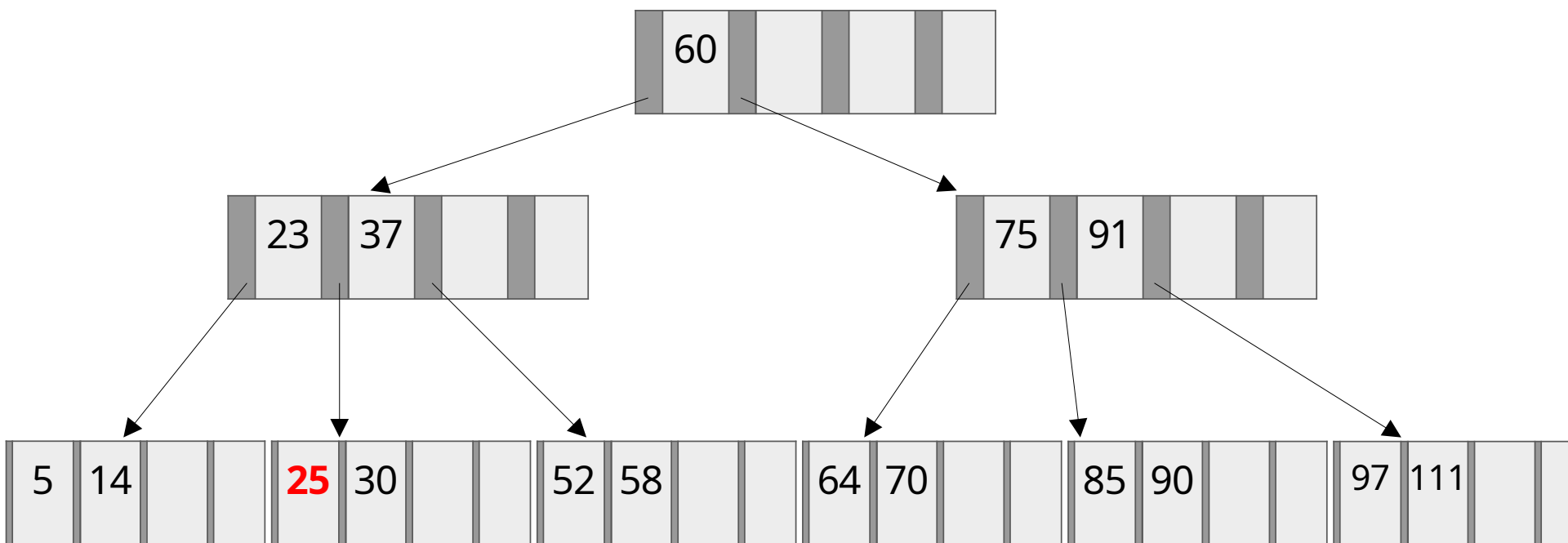
Caso 3: se a folha ficar com menos de 50% de ocupação e a página irmã puder ceder uma chave

- Normalmente necessário uma rotação para manter os índices válidos

$M = 4 \rightarrow$ **Remove 25**



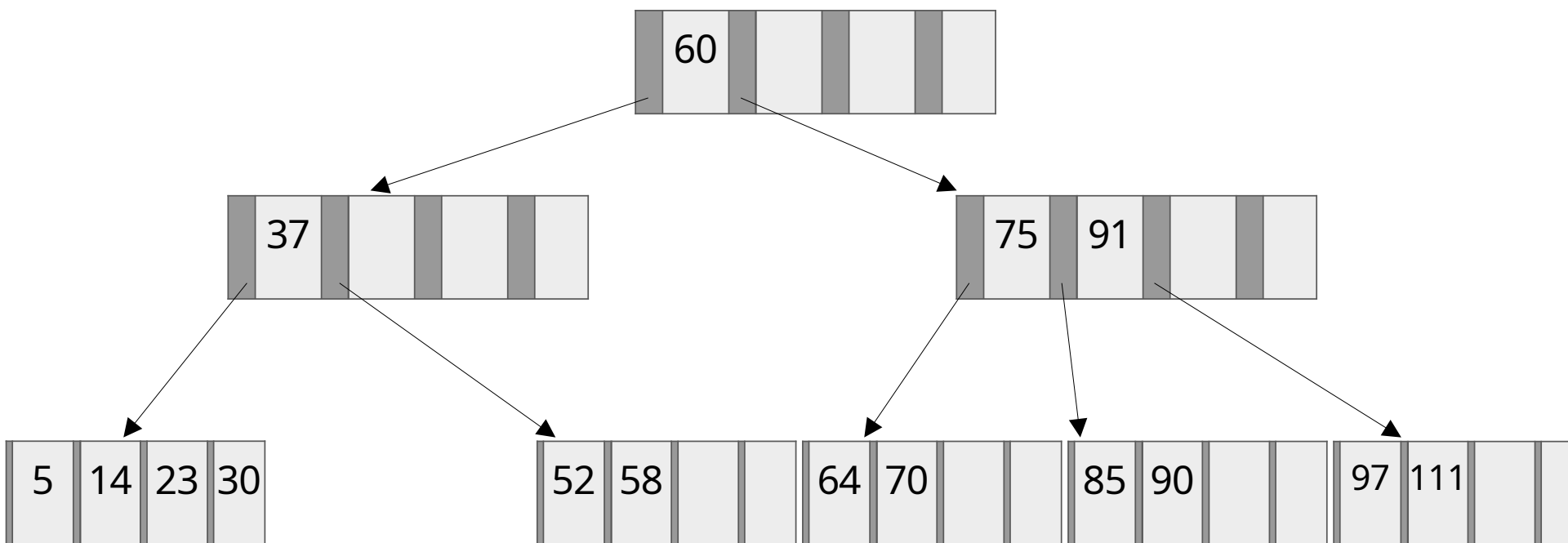
$M = 4 \rightarrow \text{Remover } 25$



Caso 4: se folha ficar com menos de 50% de ocupação e as páginas irmãs não puderem ceder uma chave

- *Faz-se a fusão com o irmão à esquerda ou o irmão à direita*

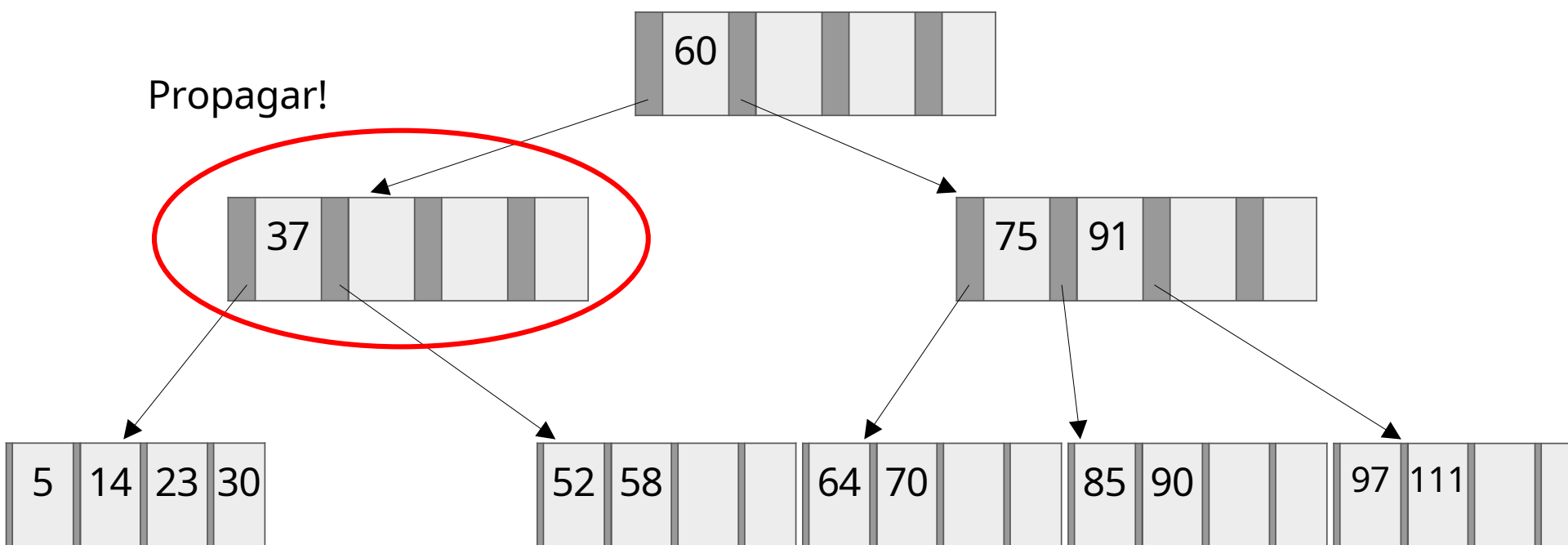
$M = 4 \rightarrow$ Remove 25



Caso 4: se folha ficar com menos de 50% de ocupação e as páginas irmãs não puderem ceder uma chave

- *Faz-se a fusão com o irmão à esquerda ou o irmão à direita*

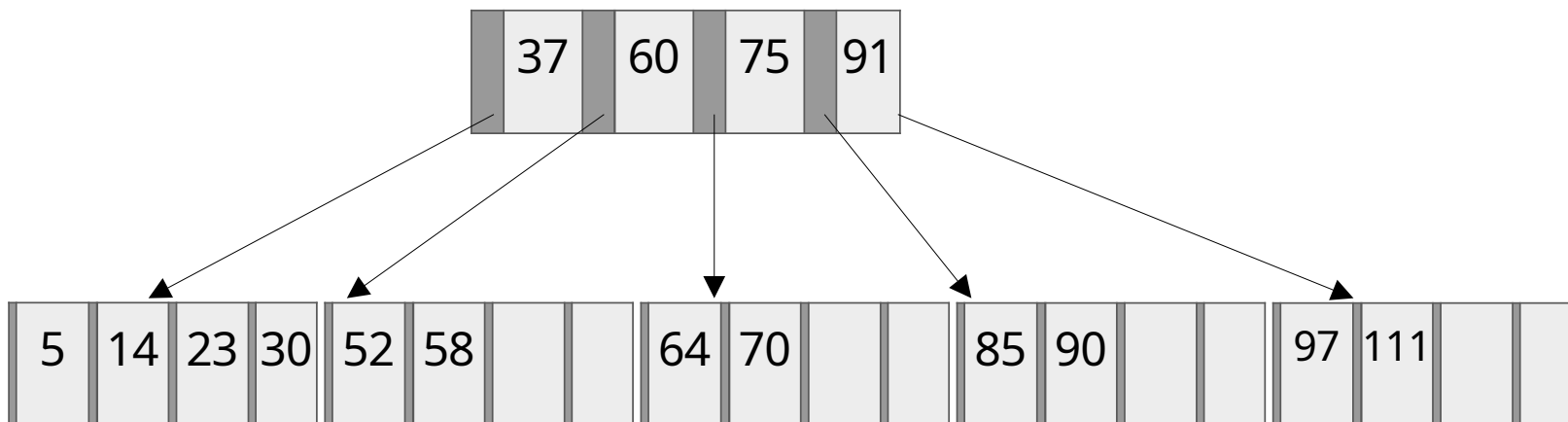
$M = 4 \rightarrow$ Remover 25



Caso 4: se folha ficar com menos de 50% de ocupação e as páginas irmãs não puderem ceder uma chave

- *Faz-se a fusão com o irmão à esquerda ou o irmão à direita*

$M = 4 \rightarrow \text{Remover } 25$



Caso 4 (novamente): se folha ficar com menos de 50% de ocupação e as páginas irmãs não puderem ceder uma chave

- *Faz-se a fusão com o irmão à esquerda ou o irmão à direita*

Exercícios

1. Mostre todas as árvores B válidas com grau mínimo 2 ($M=4$) que incluem as chaves 1, 2, 3, 4 e 5.
2. Mostre o resultado da inserção (nesta ordem) das chaves 16, 29, 27, 21, 13, 22, 18, 30, 32, 33, 23, 28, 24, 26, 11, 12, 34, 35, 14, 36, 15 em uma árvore B de grau 3 ($M=6$). Mostre a configuração da árvore após cada inserção e imediatamente antes de cada *split*.
3. Quantas leituras são feitas no pior caso para buscar um elemento em uma árvore B. Quantas leituras e escritas são necessárias para inserir um elemento, no pior caso?



Estruturas de dados II

Árvore B

André Tavares da Silva
andre.silva@udesc.br