DEPARTAMENTO DE INFORMÁTICA
PUC·RIO

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

INF 1010 – Estrutura de Dados Avançadas

Aula 12 – Árvores 2-3

2021.1

Prof. Augusto Baffa
<abaffa@inf.puc-rio.br>

PUC-Rio ExACTa

iCAN GAMES

visionlab
visualization laboratory

Árvores 2-3

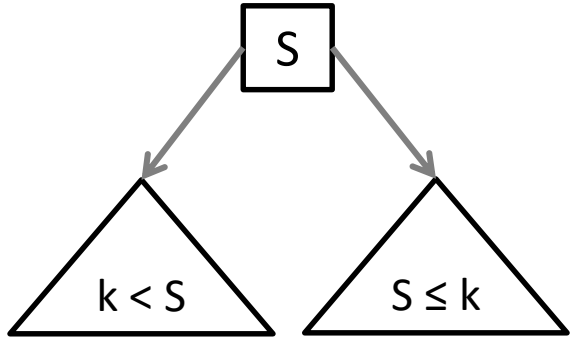
- Criada por John Hopcroft em 1970

- Cada nó armazena 1 ou 2 chaves
- Cada nó interno possui 2 ou 3 filhos

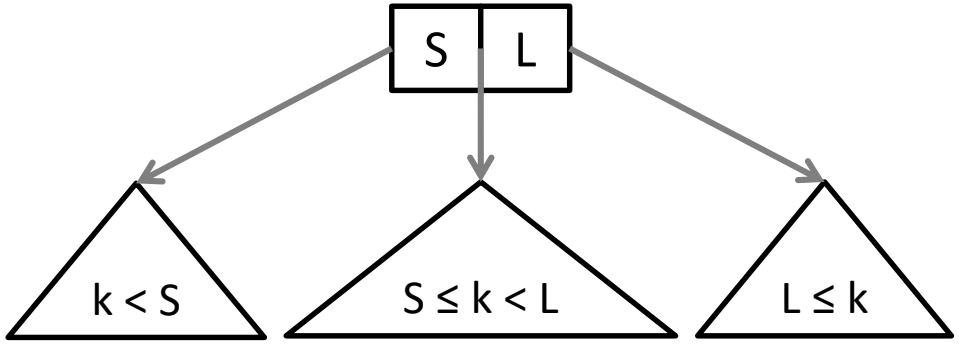
- São balanceadas
  - Não dependem de operações de rotação
  - Balanceamento é garantido diretamente pelas operações de inserção e remoção

Árvores 2-3

- São árvores de busca
  - Nó interno com 2 filhos
  - Nó interno com 3 filhos



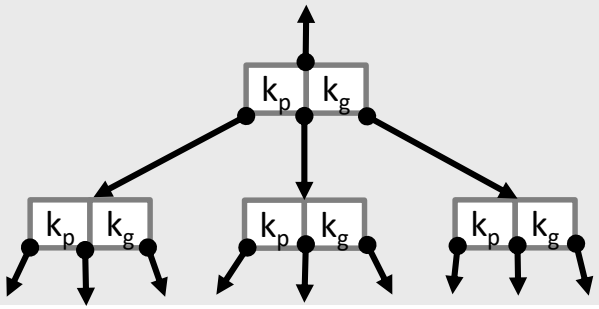
- Exemplo, buscar elemento ***k***



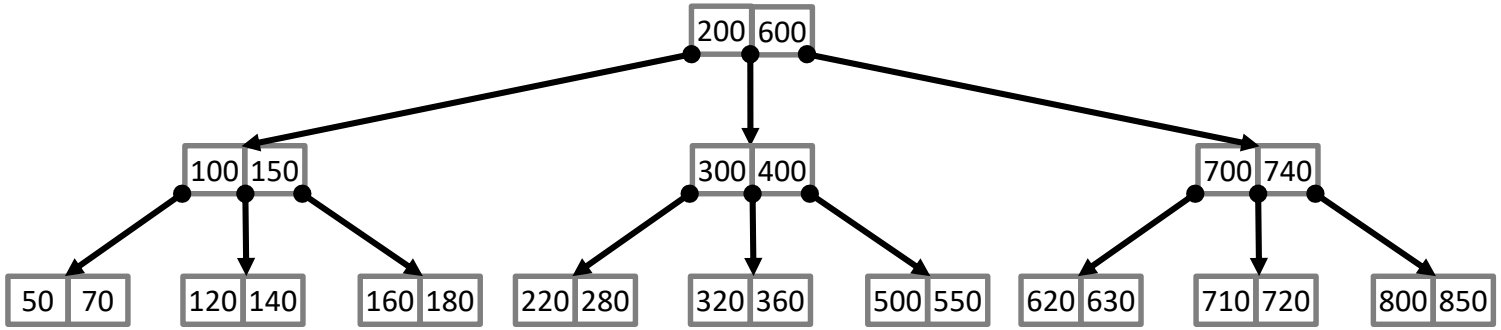
Árvores 2-3

- Árvores B de ordem 3: árvores 2-3
  - cada nó não folha tem no mínimo uma chave e dois filhos
  - cada nó não folha tem no máximo duas chaves e três filhos

```
struct arv_23 {
   int kp, kg; /* chaves: kp<kg, se kg existir.
                          se kg = -1, chave não existe */
   struct arv_23 *pai;
   struct arv_23 *esq;
   struct arv_23 *meio;
   struct arv_23 *dir;
};
```



Exemplo de Árvore 2-3

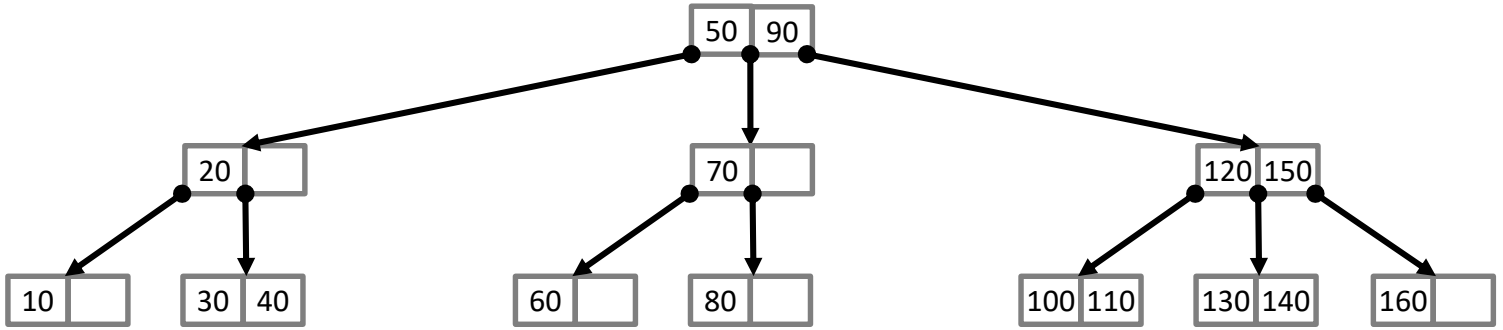


Percurso

- Percurso em Ordem Infixa
  - Caminhe pela subárvore da esquerda.
  - Imprima a menor chave (que pode ser a única).
  - Caminhe pela subárvore do meio.
  - Imprima a maior chave, se existir.
  - Caminhe pela subárvore da direita, se existir.

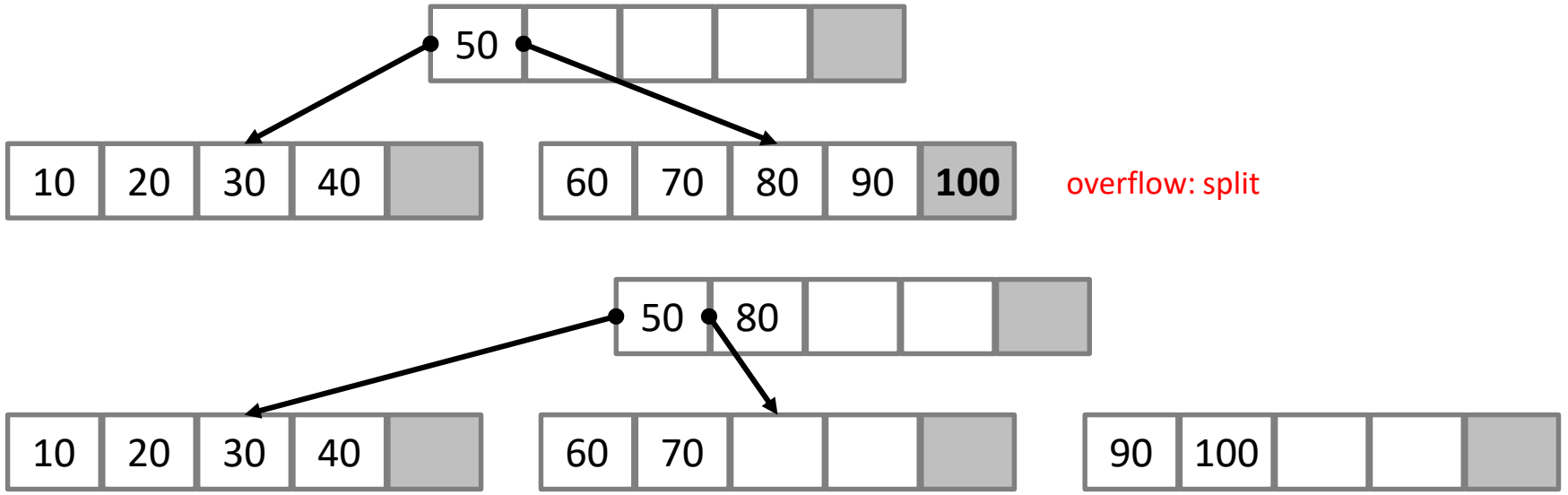
Busca

- Exemplo: Busca por “130”

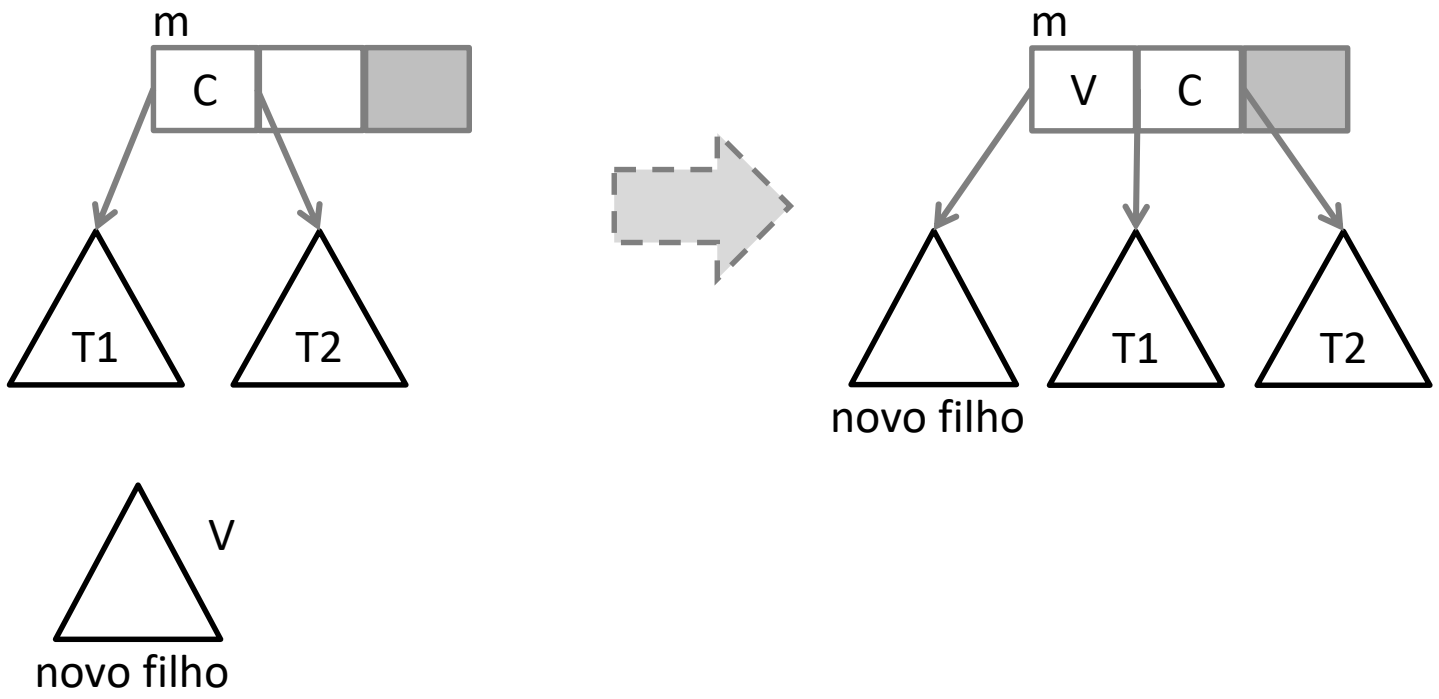


Ideia de Inserção

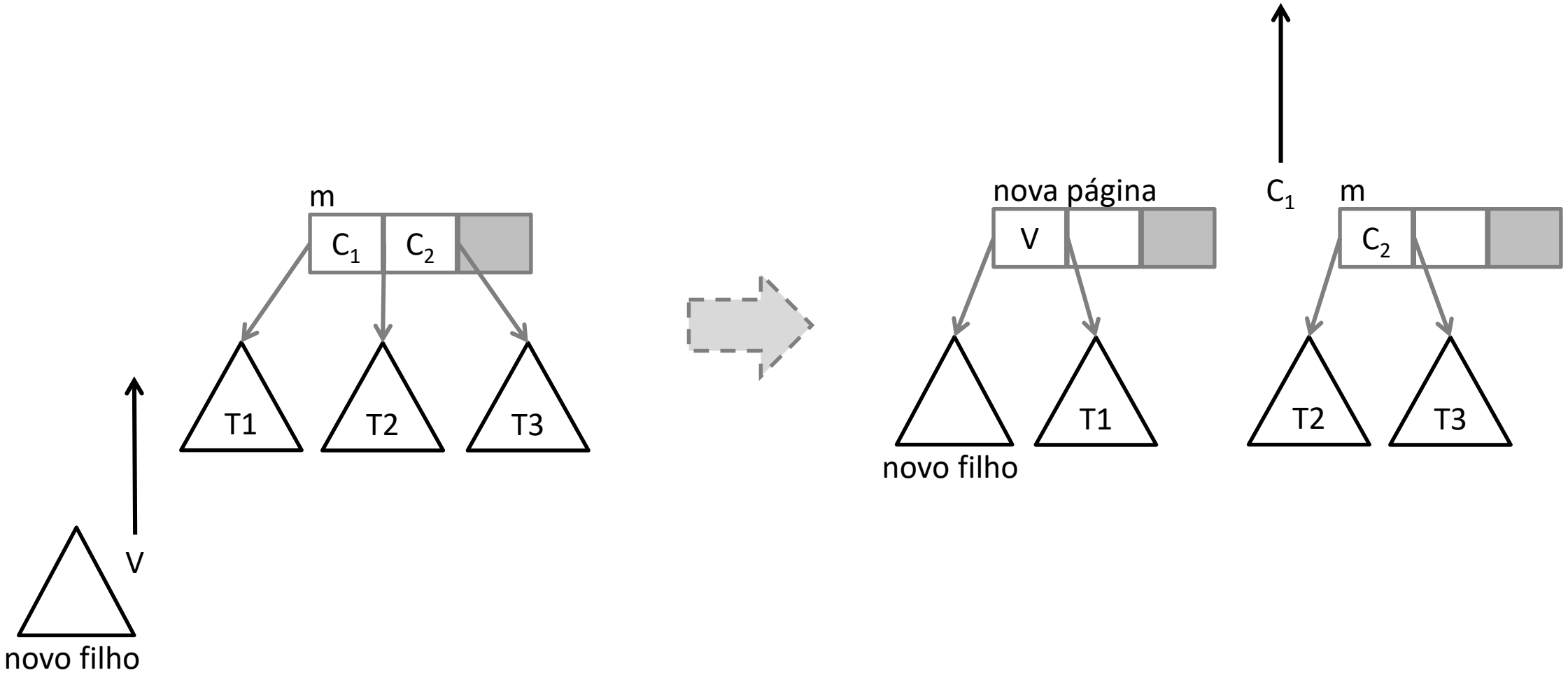
- Se há espaço no nó (não tem as duas chaves), coloca a chave neste nó
  - pode ser necessário “empurrar” a chave existente para a direita
- Se não há espaço, deve-se “quebrar” o nó em dois
  - neste caso o valor mediano deve “subir” (e o ponteiro para o novo nó)



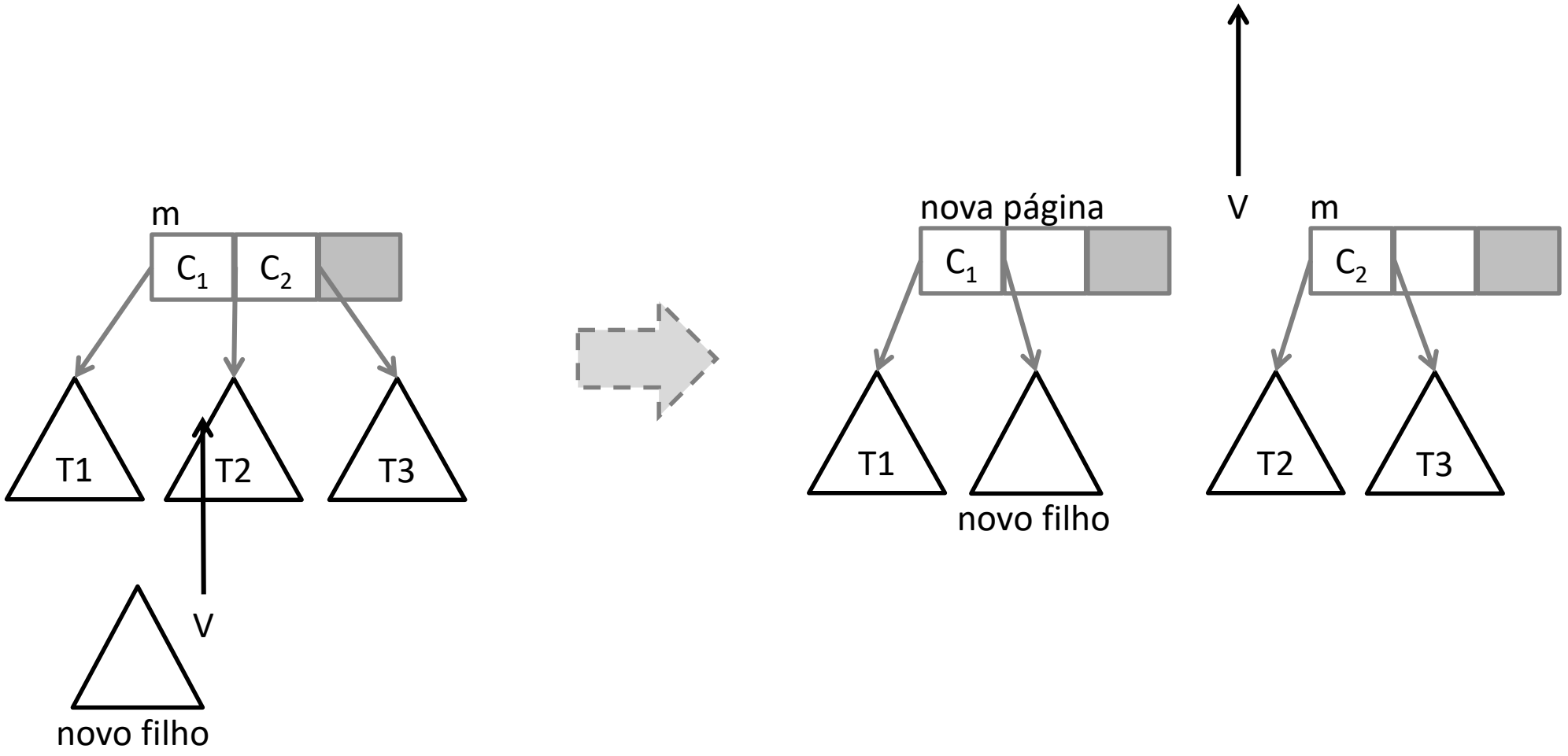
Quando há espaço no nó



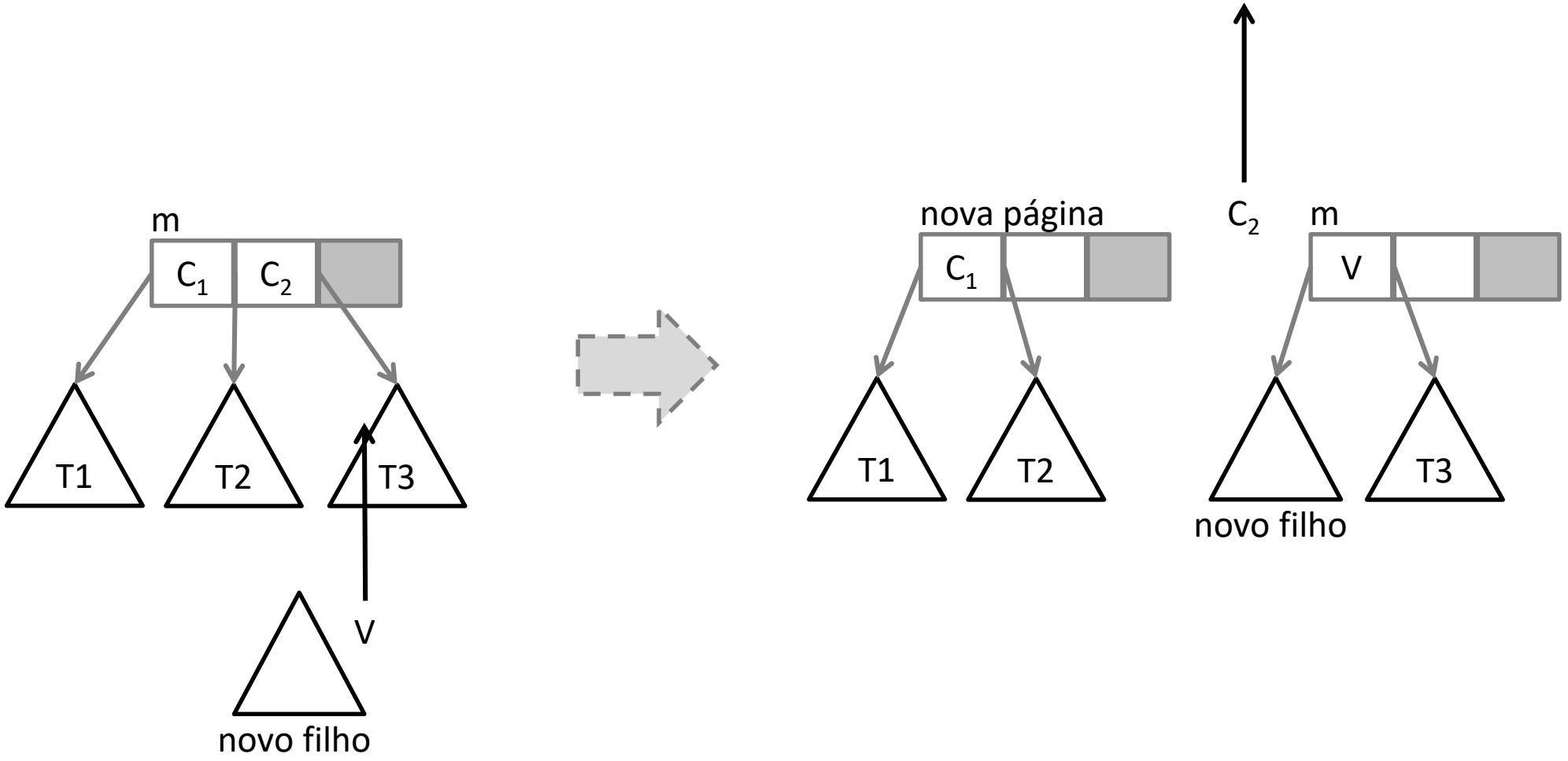
Propagação da mediana (1)



Propagação da mediana (2)



Propagação da mediana (3)



Inserção

- Algoritmo de Inserção
  - Localize a folha onde a nova chave será inserida.
  - Divida a folha, se necessário.
  - Divida nós internos, recursivamente, se necessário.

Implementação de Inserção

```
Arv23* insere (Arv23* m, int chave) {
   int valorquesubiu;
   Arv23* novofilho, novaraiz;
   if (m==NULL) {
      m = novoNo (chave);
      m->pai = novoNo (-1);
   }
   else {
      if (insere2 (m, chave, &valorquesubiu, &novofilho)) {
         /* cria nova raiz */
         novaraiz = novoNo (valorquesubiu);
         novaraiz->pai = m->pai;
         novaraiz->esq = novofilho;
         novaraiz->esq->pai = novaraiz;
         novaraiz->meio = m;
         novaraiz->meio->pai = novaraiz;
         m = novaraiz;
      }
   }
   return m;
}
```

```
static Arv23* novoNo (int chave) {
   Arv23 *m = (Arv23*)malloc(sizeof(struct arv_23));
   if (m==NULL) return NULL;

   m->pai = NULL;
   m->kp = chave;
   m->kg = -1;
   m->esq = m->meio = m->dir = NULL;
   return m;
}
```

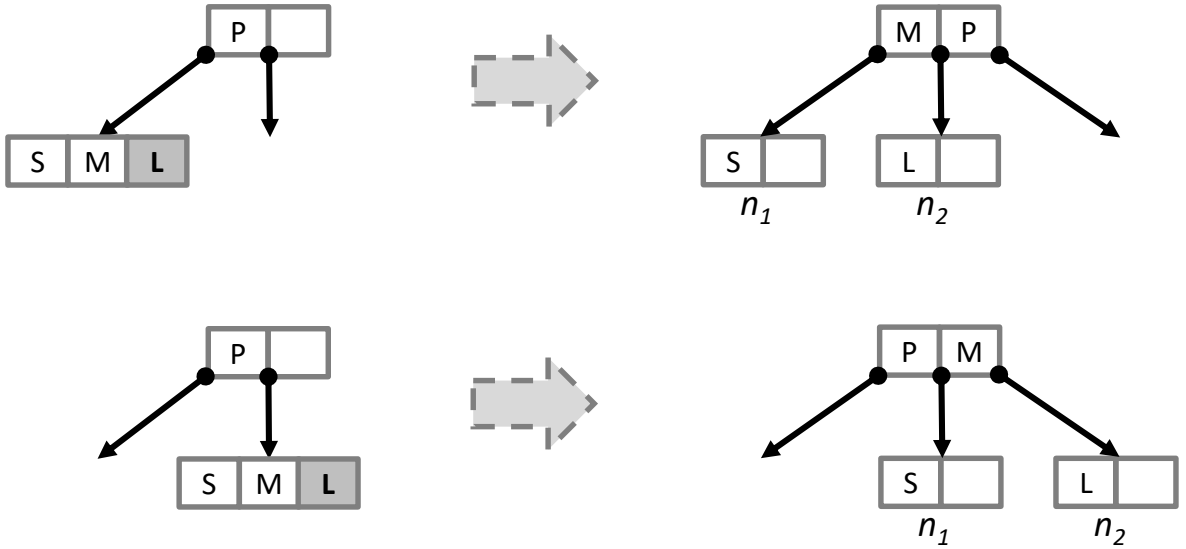
Implementação de Inserção

```
static int insere2 (Arv23* m, int chave, int* valorainserir,
                                        Arv23** novofilho) {
   /* indica se deve inserir neste nó */
   int inseriraqui = 0;
   if (m==NULL) { printf("erro! subárvore nula! \n"); exit (1);}

   if (m->esq != NULL) {

      /* não é folha, só insere aqui se subir um valor */
      if (chave < m->kp)
         inseriraqui = insere2(m->esq, chave, valorainserir,
                                                 novofilho);

      else if (((m->kg != -1) && (chave < m->kg)) || (m->kg == -1))
         /* ou está entre as duas chaves ou só tem uma chave no nó */
         inseriraqui = insere2(m->meio, chave,  valorainserir,
                                                 novofilho);

      else /* chave > m->kg */
         inseriraqui = insere2(m->dir, chave, valorainserir,
                                                 novofilho);
   }
   else {
      /* este nó é folha, tem que inserir nele de qq jeito */
      *valorainserir = chave;
      inseriraqui = 1;
      *novofilho = NULL;
   }
...
```

```
...
   if (!inseriraqui) return 0; /* inserção já
                                  está completa */

   /* procura espaço no nó */
   if (m->kg==-1) { /* tem espaço no nó */
      if (*valorainserir < m->kp) {
         /* empurra chave kp */
         …
      }
      else {
      /* é maior que a chave que já está lá */
         …
      }
      return 0; /* como havia espaço,
                   não sobem valores */
   }
   *novofilho = overflowQuebra (m, valorainserir,
                                        *novofilho);
   return 1; /* se há quebra, sobe mediana
                para nova inserção */
}
```

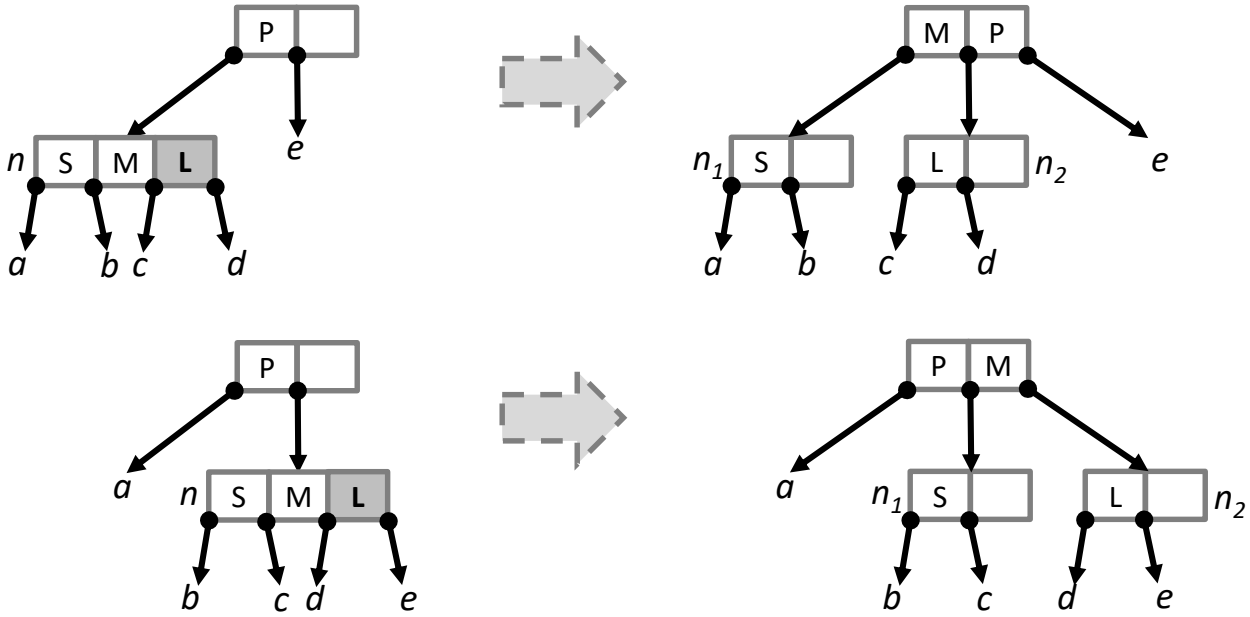
Inserção

- Algoritmo de Inserção (cont.)
  - Se a folha já tem 2 chaves, divida a folha
  - Mova a “chave do meio” para o pai



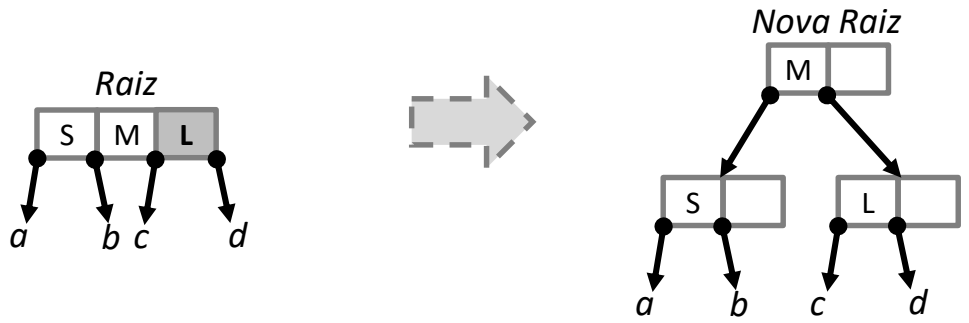
Inserção

- Algoritmo de Inserção (cont.)
  - Se o pai já tem 2 chaves, divida o pai
  - Mova a “chave do meio” para o pai do pai



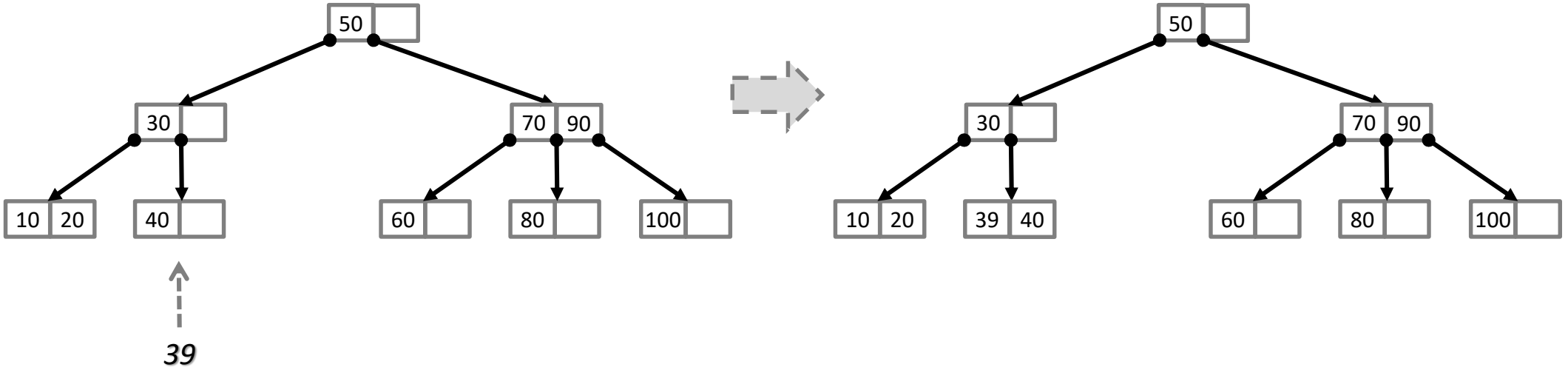
Inserção

- Algoritmo de Inserção (cont.)
  - Se a raiz já tem 2 chaves, crie nova raiz
  - Mova a “chave do meio” para a nova raiz



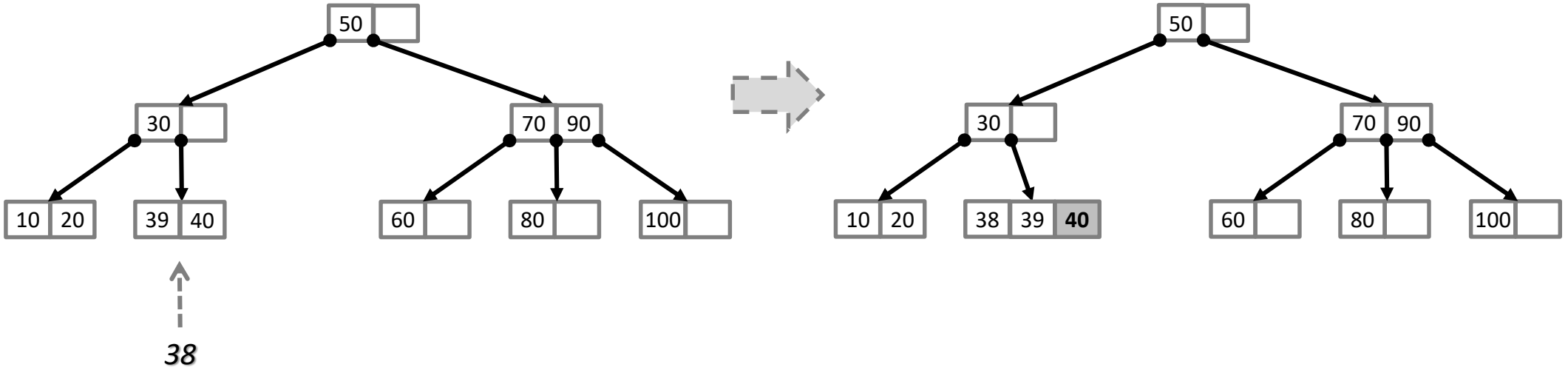
Inserção

- Inserção de “39”
  - Localize a folha onde “39” deve ser inserida
  - A folha só tem 1 chave: insira “39”



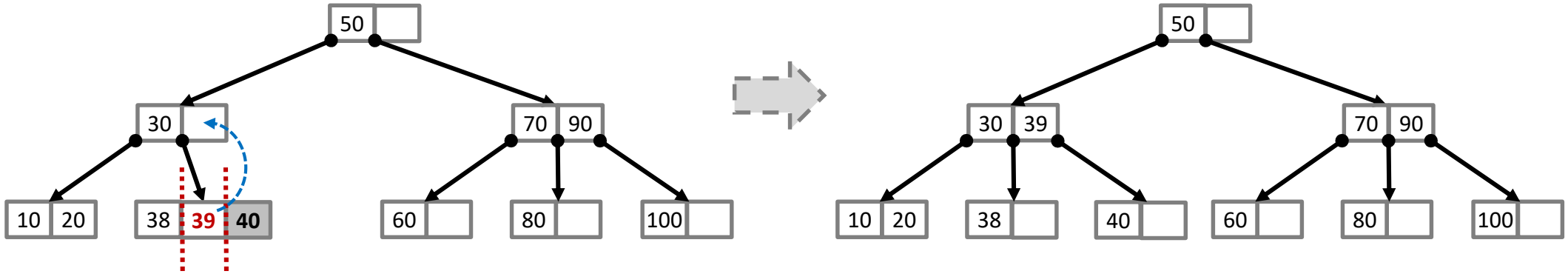
Inserção

- Inserção de “38”
  - Localize a folha onde “38” deve ser inserida
  - A folha tem 2 chaves: simule a inserção de “38”



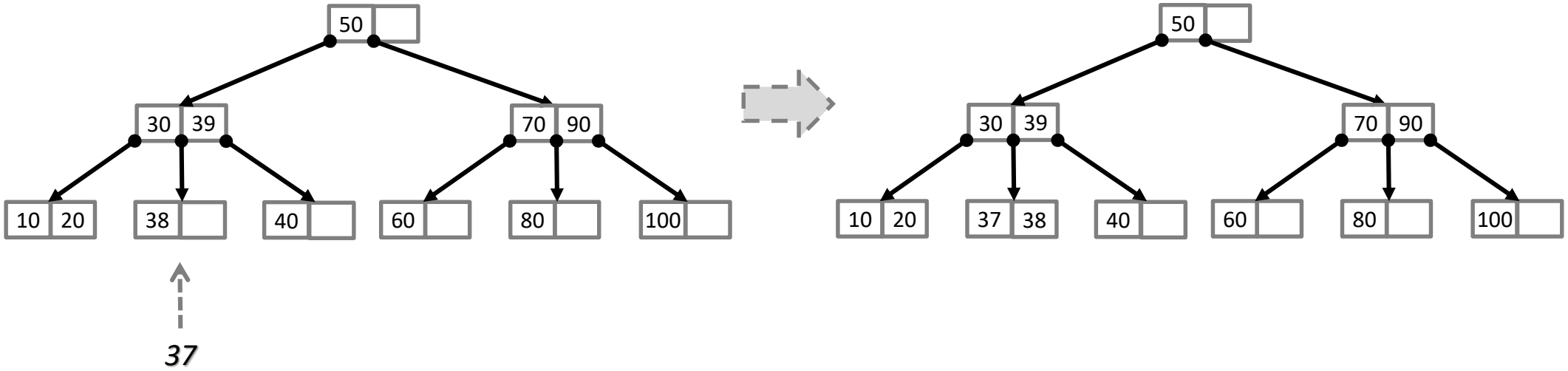
Inserção

- Inserção de “38”
  - Mova a chave do meio para o pai **p**
  - Separe o menor e o maior valores em 2 nós, que serão filhos de **p**



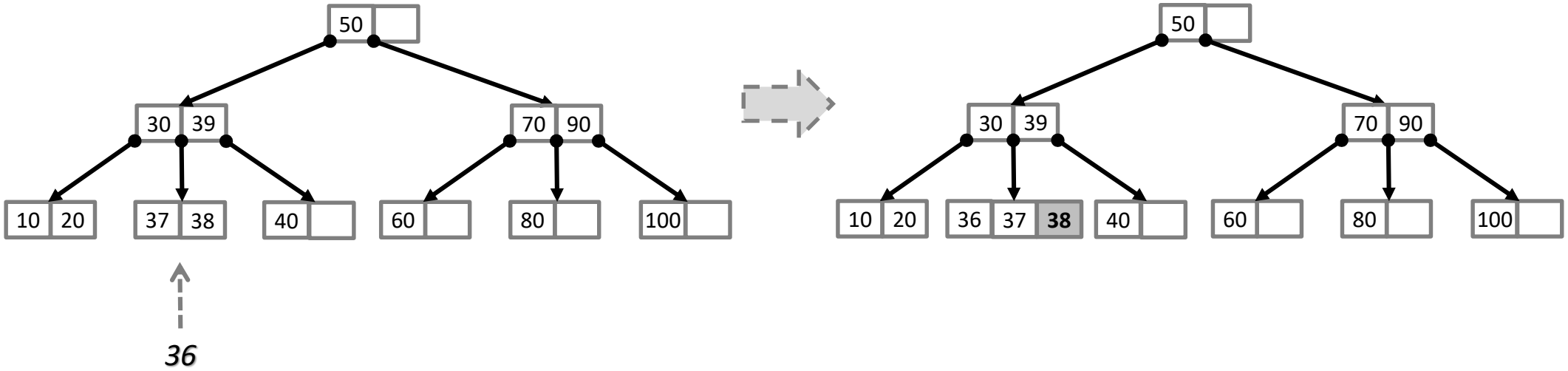
Inserção

- Inserção de “37”
  - Localize a folha onde “37” deve ser inserida
  - A folha só tem 1 chave: insira “37”



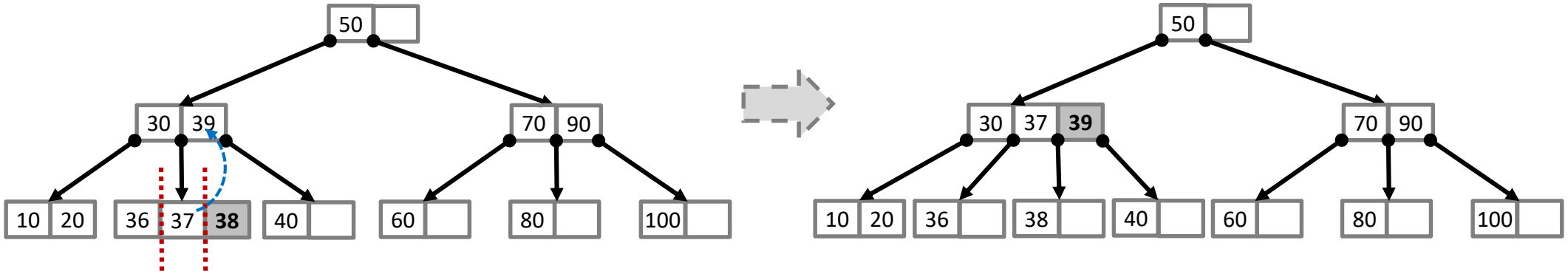
Inserção

- Inserção de “36”
  - Localize a folha onde “36” deve ser inserida
  - A folha já tem 2 chave: Simule a inserção de “36”



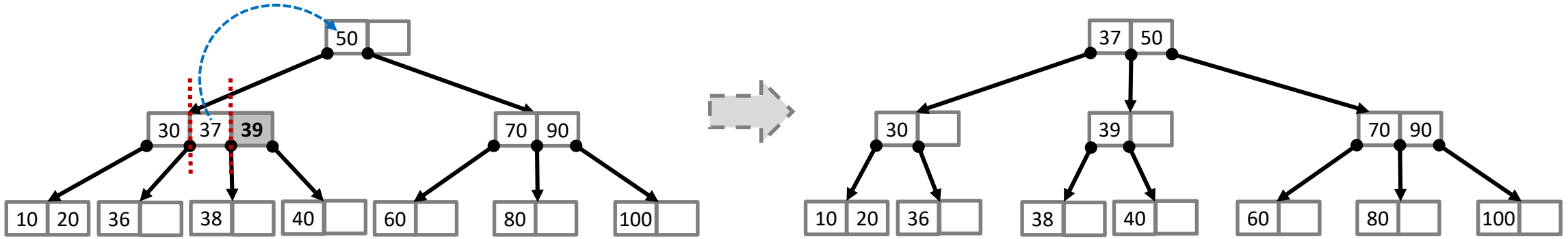
Inserção

- Inserção de “36”
  - Simule mover a chave do meio para o pai **p**
  - Simule criar 2 novos filhos de **p**

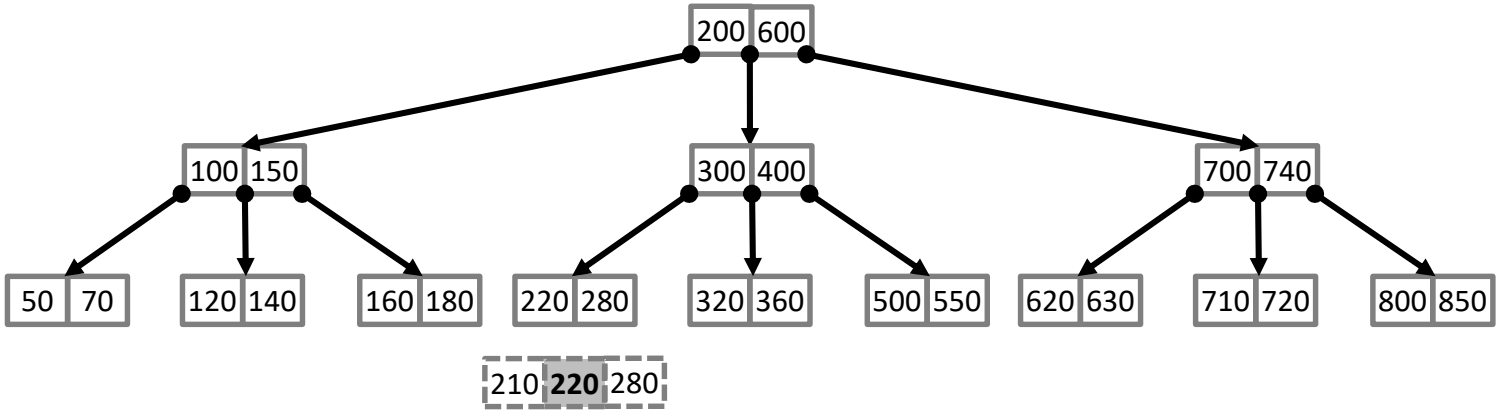


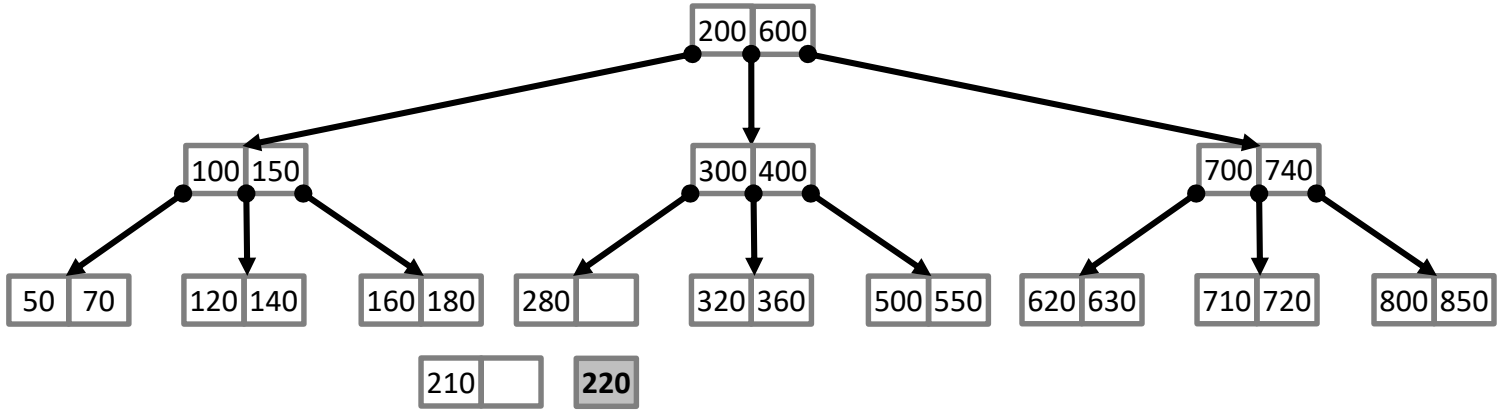
Inserção

- Inserção de “36”
  - Divida o nó **p**
    - Mova a chave do meio para o pai de **p**
    - Os nós com as chaves menor e maior formam novos nós
    - Redistribua os 4 filhos de **p** entre os novos nós



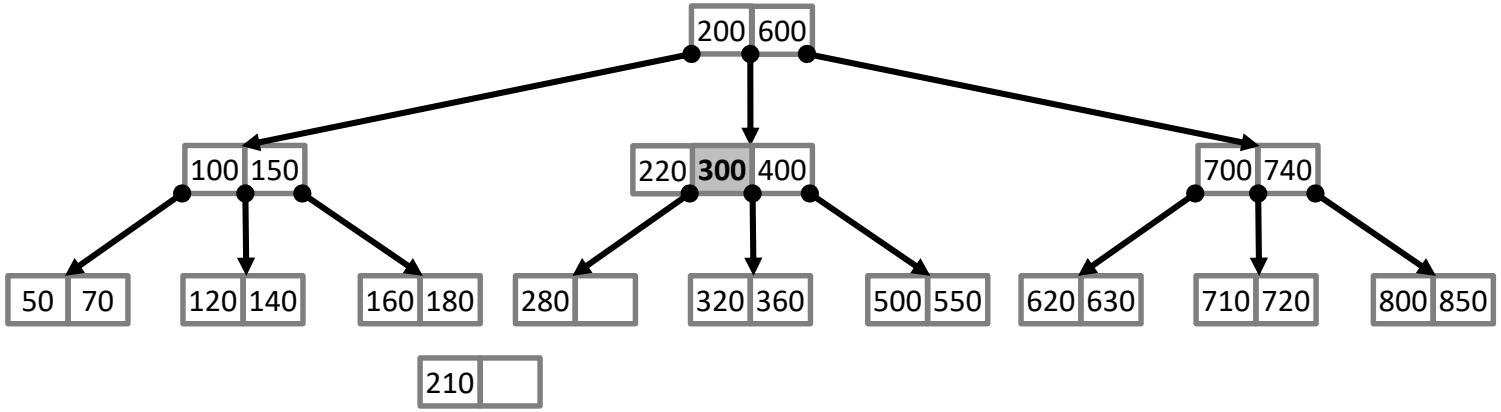
Exemplo de Árvore 2-3
Inserir 210



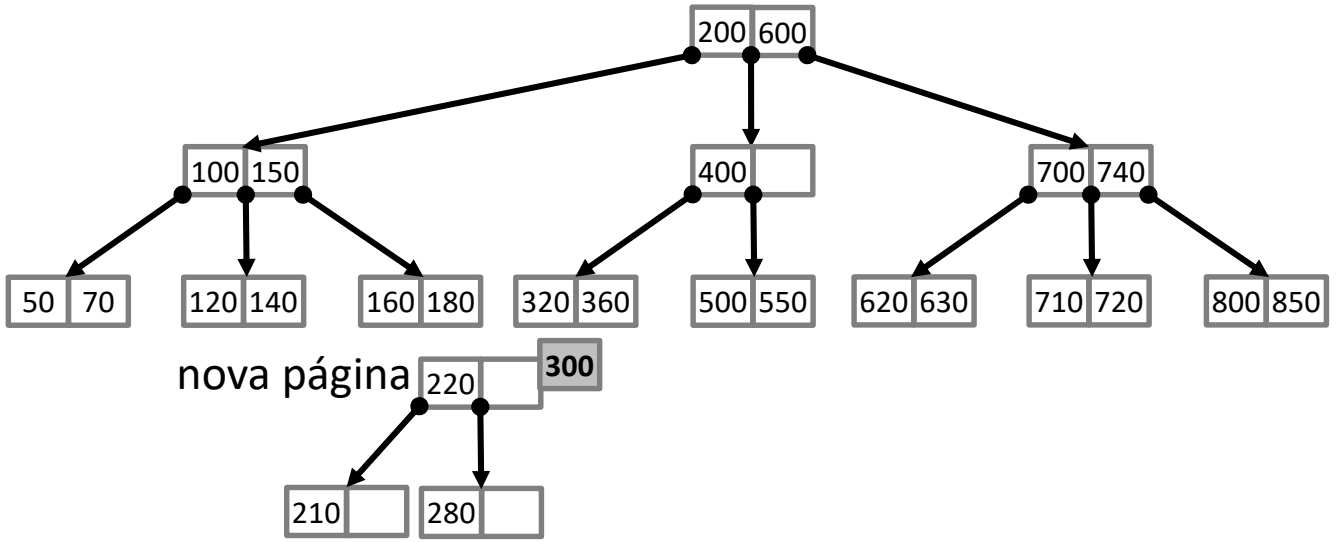


nova página

Exemplo de Árvore 2-3
Inserir 210

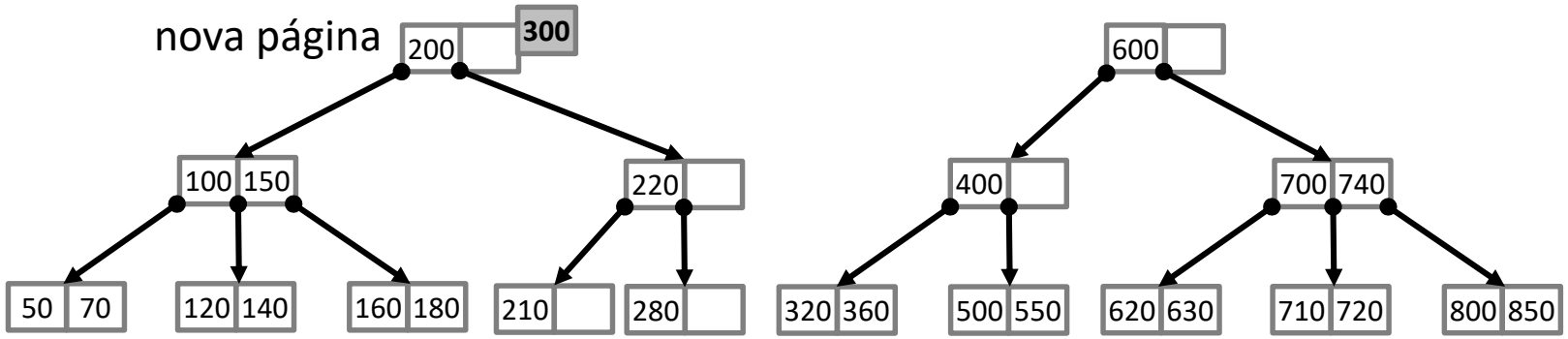


nova página

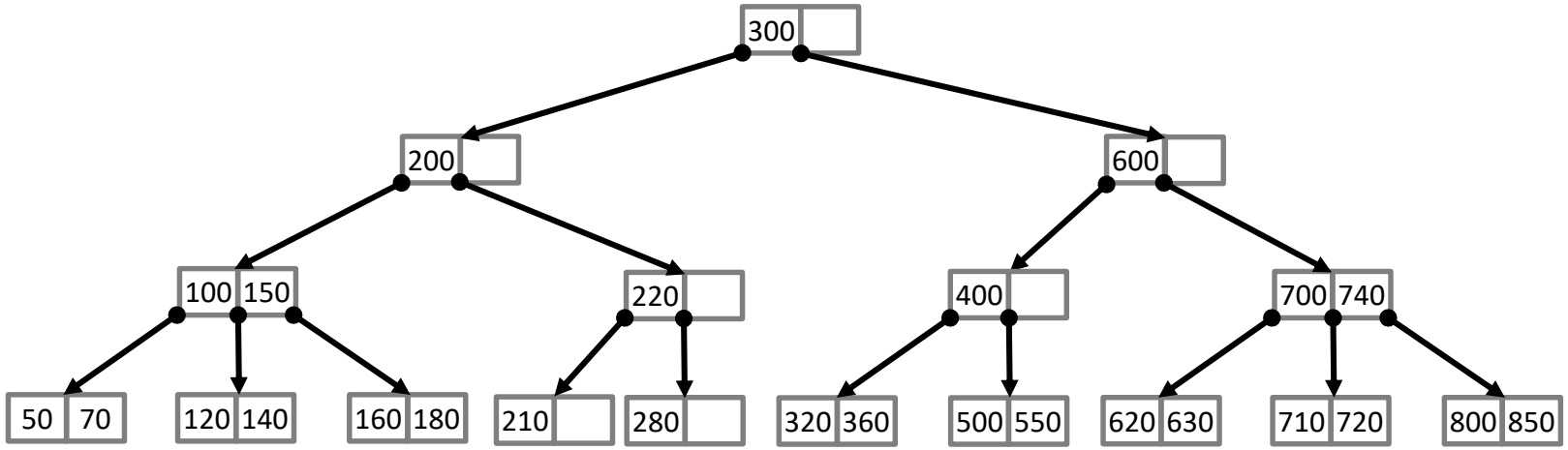


nova página

Exemplo de Árvore 2-3
Inserir 210



Exemplo de Árvore 2-3
Inserir 210



Remoção

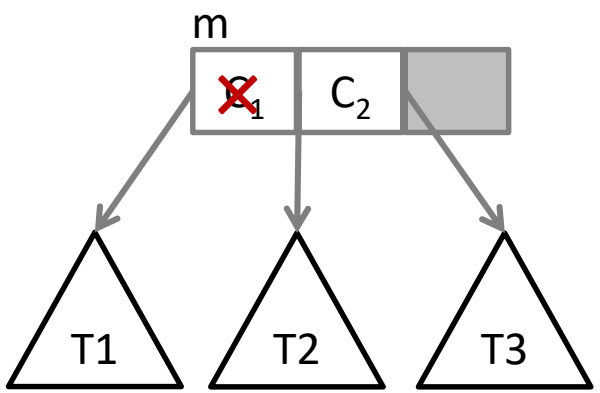
- Algoritmo de Remoção
  - Se a chave a ser removida pertencer a uma folha,
    - Remova a chave.
  - Se a chave a ser removida pertencer a um nó interno
    - Troque a chave com a sua sucessora (que necessariamente está em uma folha).
    - Remova a chave (da folha).
  - Redistribua e combine nós.

Remoção em árvore 2-3

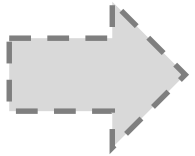
- Se não é nó folha, substitua a chave pela maior chave da sua subárvore à esquerda ou a menor chave da sua sub-árvore à direita

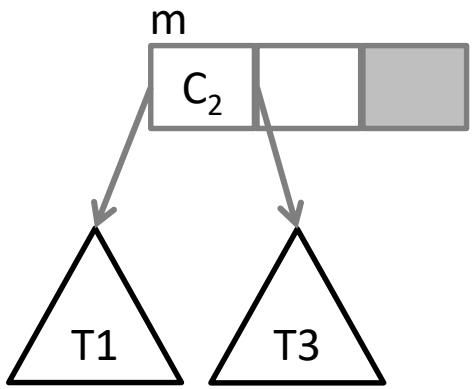
```
if (m->esq != NULL) { /* não é folha */
   if (chave < m->kp)
      res = retirarec (m->esq, chave);

   else if (m->kp == chave) { /* achou - troca por succ */
      m->kp = maisaesquerda (m->meio);
      res = retirarec (m->meio, m->kp);
   }
   else if (((m->kg != -1) && (chave < m->kg)) || (m->kg == -1))
   /* ou está entre as duas chaves ou só tem uma chave no nó */
      res = retirarec(m->meio, chave);

   else if (m->kg == chave) { /* achou - troca por succ */
      m->kg = maisaesquerda (m->dir);
      res = retirarec (m->dir, m->kg);
   }
   else  /* chave > m->kg */
      res = retirarec(m->dir, chave);

   if (res==OK) return OK;
}
...
```

Remoção em árvore 2-3: casos simples



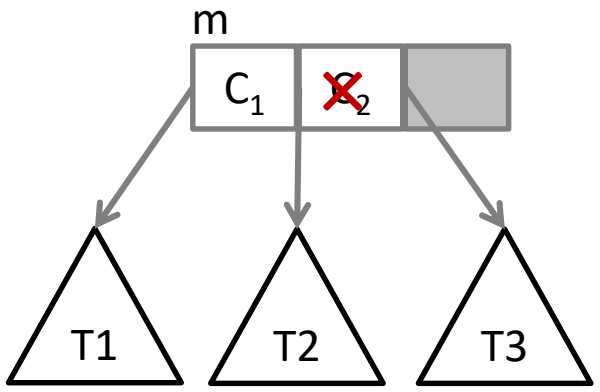
T1 ou T2
"acabaram"



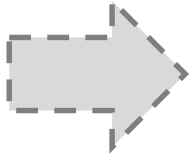


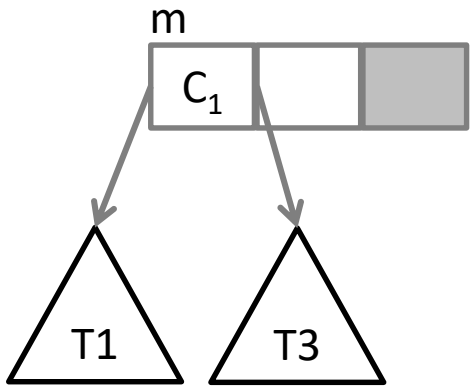
ou T2

RETIRA_MENOR



T2 ou T3
"acabaram"





ou T2

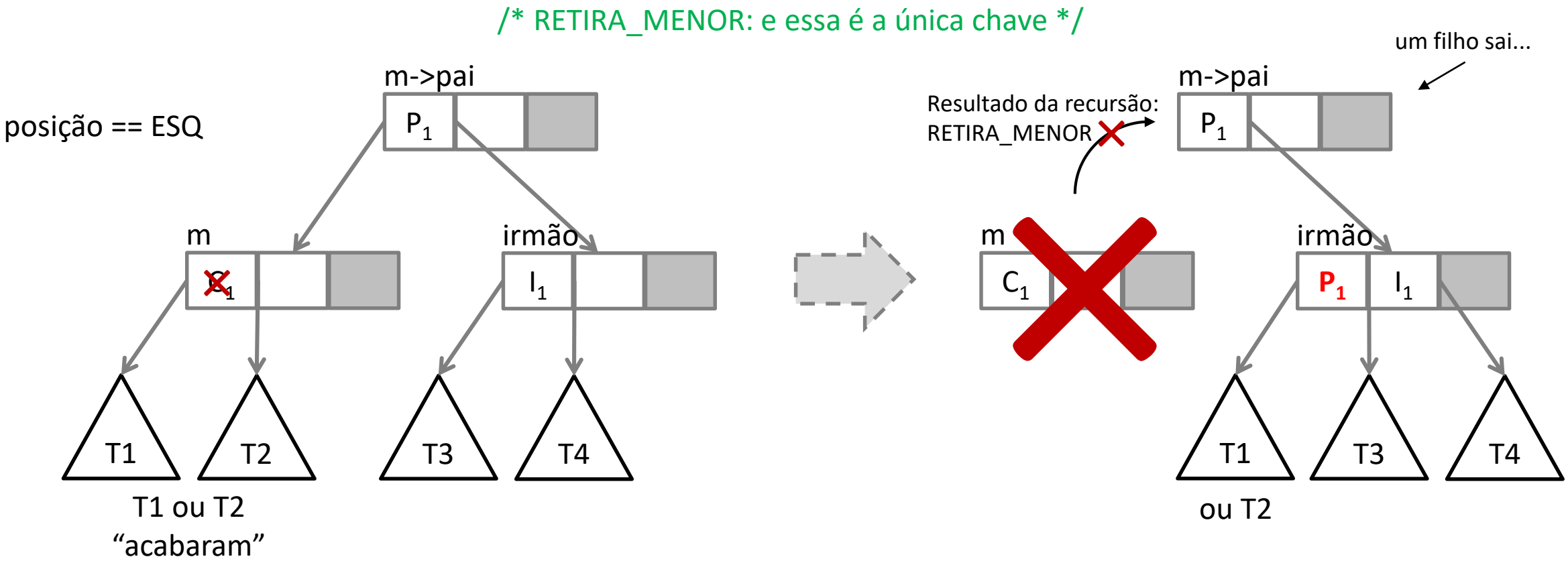
RETIRA_MAIOR

Remoção em árvore 2-3: casos simples

```
...
else { /* este nó é folha, ou chave está aqui ou não está na árvore*/
   if (chave==m->kp) res = RETIRA_MENOR;
   else if (chave == m->kg) res = RETIRA_MAIOR;
   else /* chave não está na árvore!!! */
      return OK;
}
/* retirada */
/* pode ser porque estamos em uma folha ou "desceu" uma das chaves */
if (res == RETIRA_MAIOR) { /* caso mais simples */
   preenche (m, m->esq, m->kp, (m->meio ? m->meio : m->dir), -1, NULL);
   return OK;
}

/* RETIRA_MENOR */
if (m->kg != -1) {
   /* ainda vai ficar um no nó, tb simples */
   preenche (m, (m->esq ? m->esq : m->meio), m->kg, m->dir, -1, NULL);
   return OK;
}
...
```

Remoção em árvore 2-3: combinação

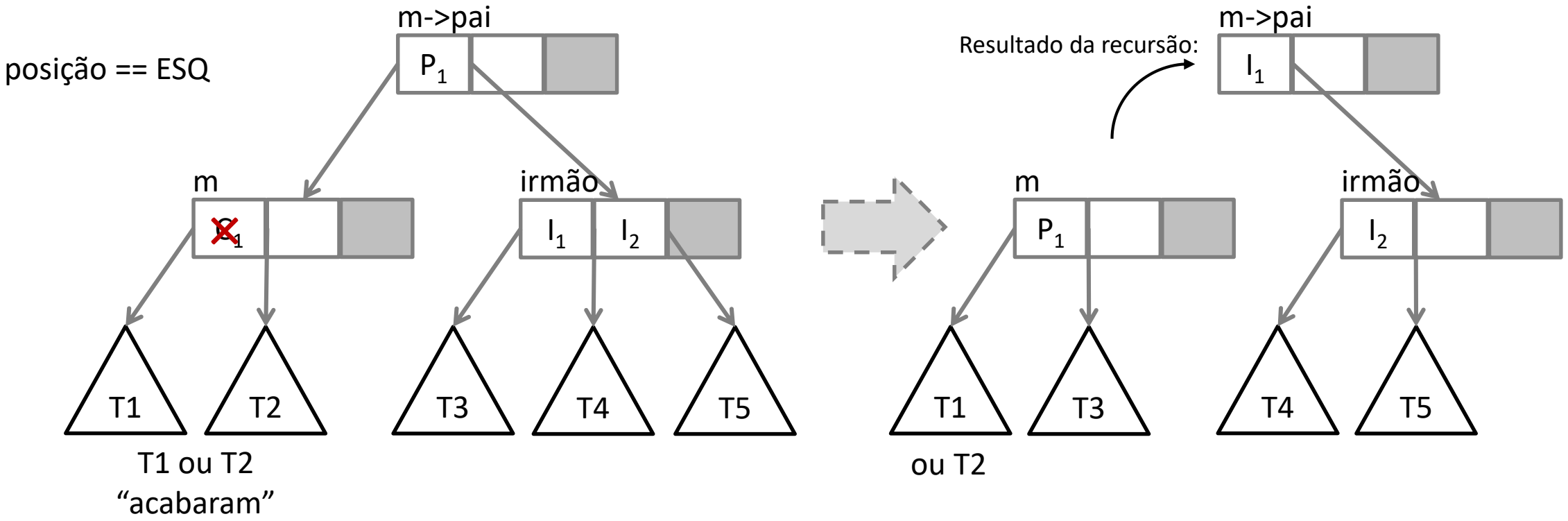
- O nó adjacente tem, **menos de m-1 (só 1 chave)**: a chave do "pai" desce para a combinação



Remoção em árvore 2-3: combinação

```
...

/* RETIRA_MENOR */
if (m->kg != -1) {
   /* ainda vai ficar um no nó, tb simples */
   preenche (m, (m->esq ? m->esq : m->meio), m->kg, m->dir, -1, NULL);
   return OK;
}

/* RETIRA_MENOR: essa é a única chave! combinar ou distribuir */
minhapos = minhaposnopai (m->pai, m);

/* se ainda tiver algum filho pegá-lo para passar para outro */
filhoqueficou = m->esq ? m->esq : m->meio;
if (minhapos == ESQ) {
   irmao = m->pai->meio
   if (irmao->kg == -1) { /* combinar */
      preenche (irmao, filhoqueficou, m->pai->kp, irmao->esq, irmao->kp, irmao->meio);
      if (irmao->esq) irmao->esq->pai = irmao;
      m->pai->esq = NULL;
      free(m);
      res = RETIRA_MENOR;
   }
   ...
```

Remoção em árvore 2-3: redistribuição

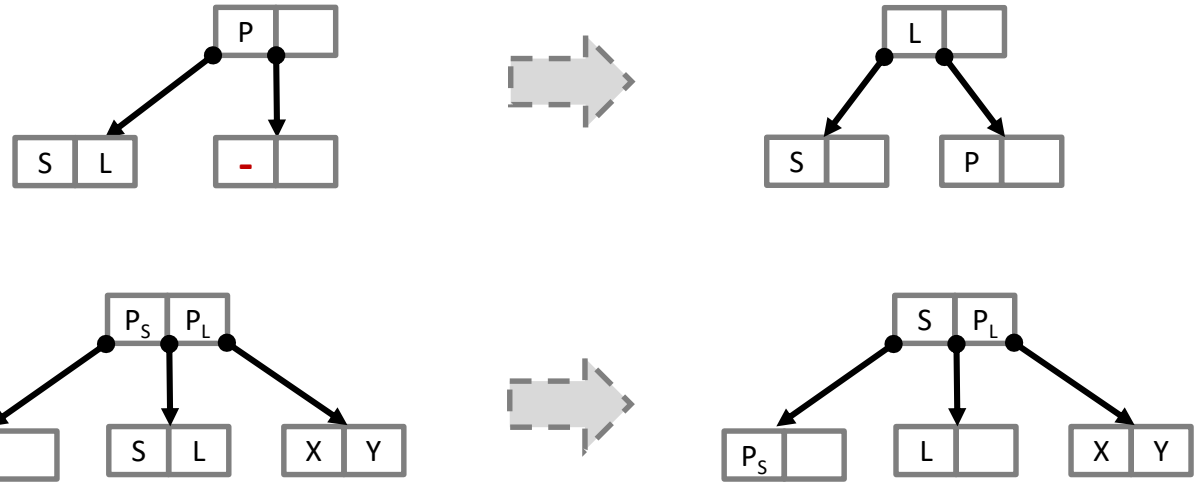


Remoção em árvore 2-3: redistribuição

```
...
/* RETIRA_MENOR: essa é a única chave! combinar ou distribuir */
minhapos = minhaposnopai (m->pai, m);

/* se ainda tiver algum filho pegá-lo para passar para outro */
filhoqueficou = m->esq ? m->esq : m->meio;
if (minhapos == ESQ) {
   irmao = m->pai->meio;
   if (irmao->kg == -1) { /* combinar */
      ...
   }
   else { /* irmão tem duas chaves, redistribuir */
      preenche (m, filhoqueficou, m->pai->kp, irmao->esq, -1, NULL);
      if (m->esq) m->esq->pai = m;
      if (m->meio) m->meio->pai = m;
      preenche (m->pai, m->pai->esq, irmao->kp, m->pai->meio, m->pai->kg, m->pai->dir);
      preenche (irmao, irmao->meio, irmao->kg, irmao->dir, -1, NULL);
      res = OK;
   }
}
...
```

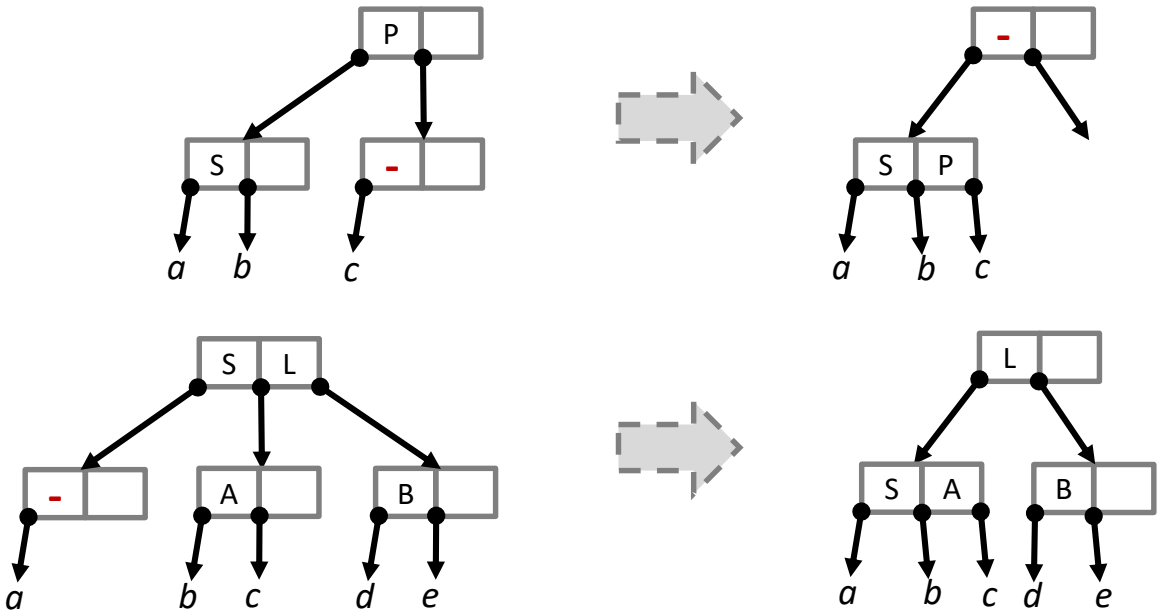
Remoção

- Algoritmo de Remoção (cont)
  - Remova e Redistribua – Casos 1 e 2



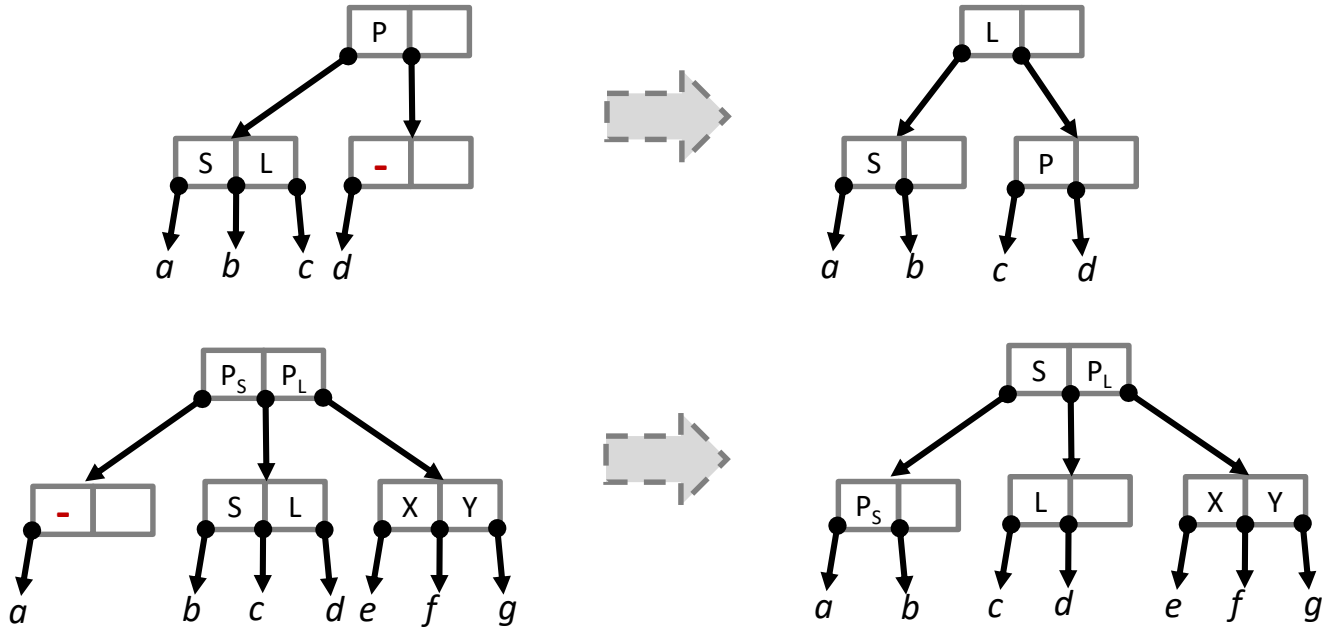
Remoção

- Algoritmo de Remoção (cont)
  - Remova e Combine – Casos 3 e 4



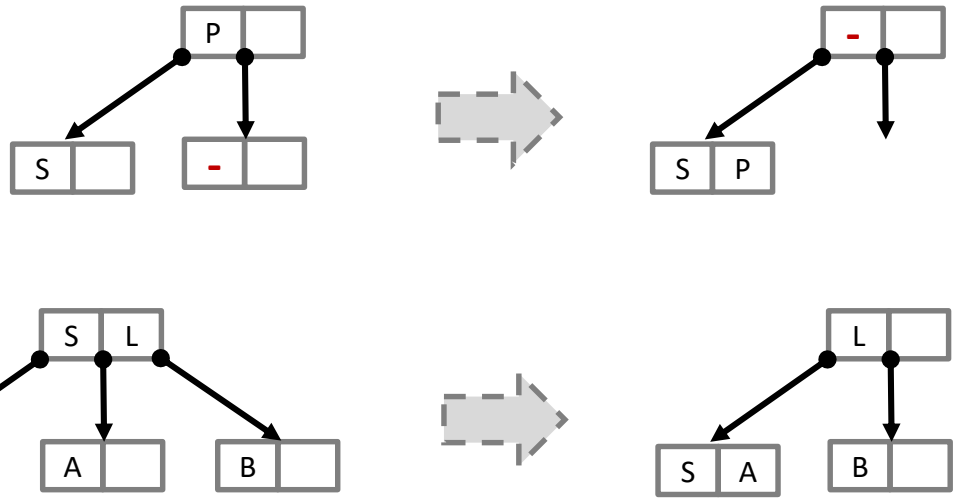
Remoção

- Algoritmo de Remoção (cont)
  - Remova e Redistribua – Casos 5 e 6



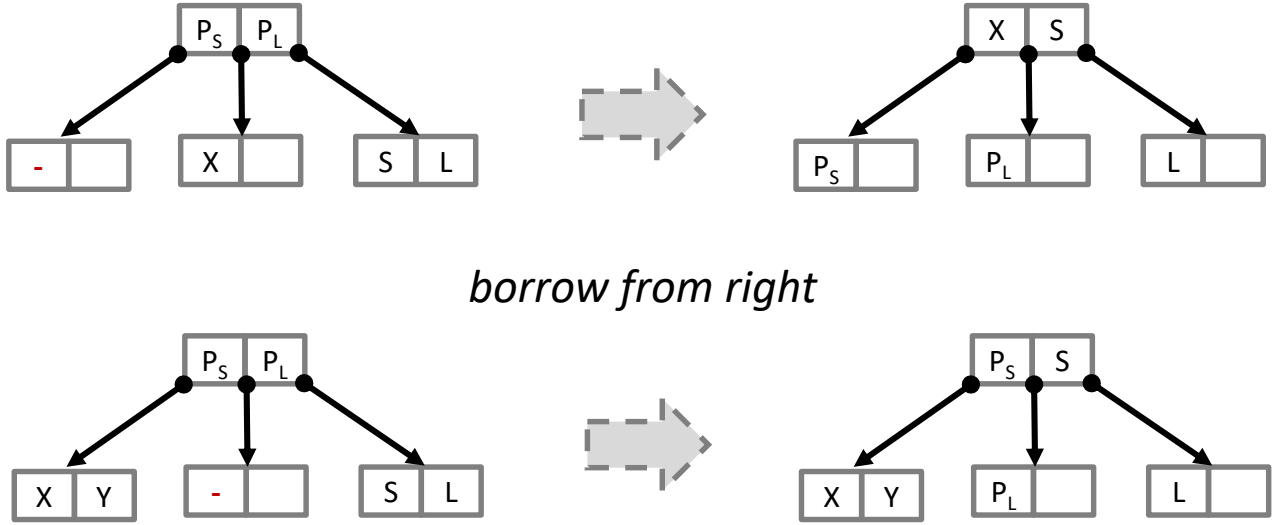
Remoção

- Algoritmo de Remoção (cont)
  - Remova e Combine – Casos 7 e 8



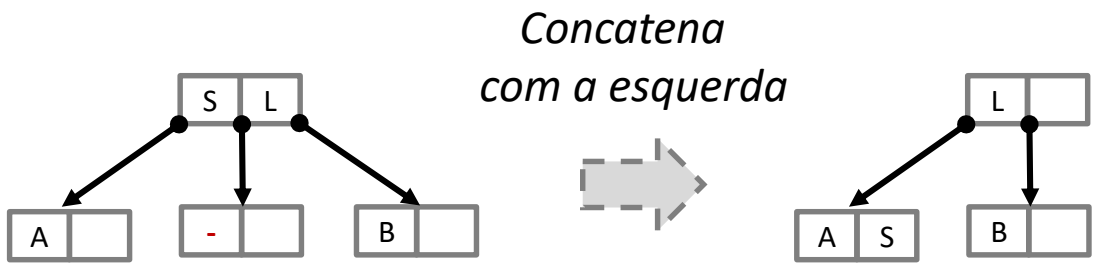
Remoção

- Algoritmo de Remoção (cont)
  - Remova e Redistribua – Casos 9 e 10



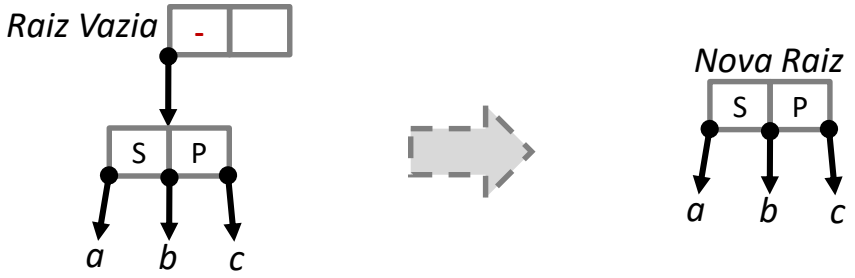
Remoção

- Algoritmo de Remoção (cont)
  - Remova e Combine – Caso 11



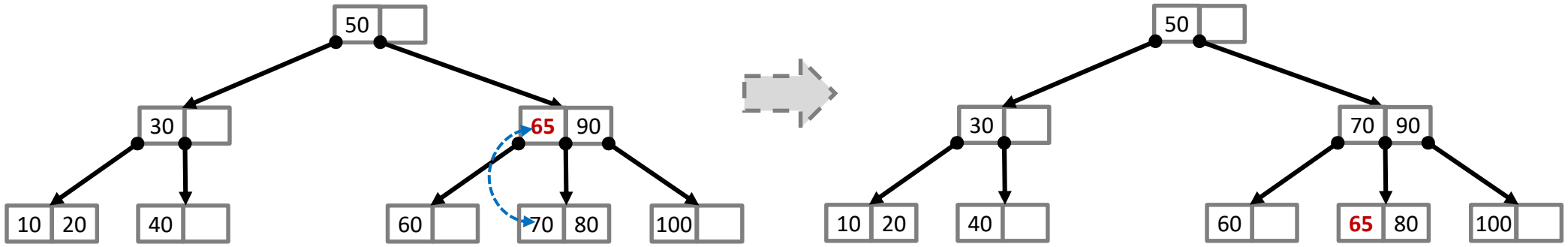
Remoção

- Algoritmo de Remoção (cont)
  - Remova a raiz – Caso 12



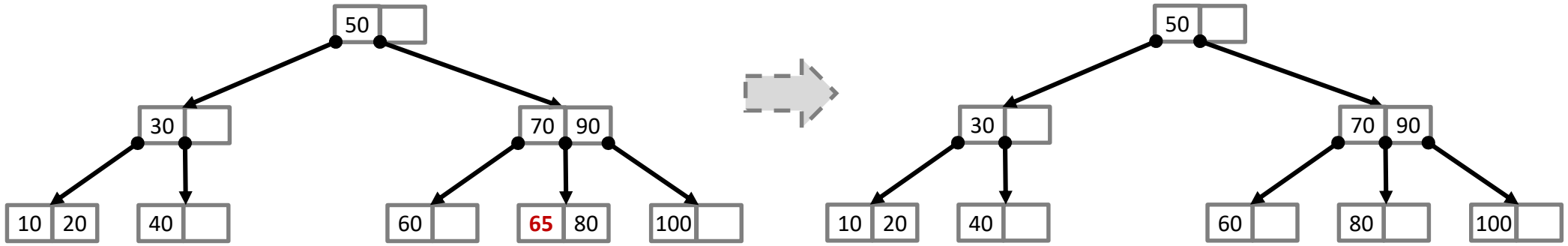
Remoção

- Remoção de “65”
  - O nó que contém “65” é interior
  - Troque “65” com a sua sucessora



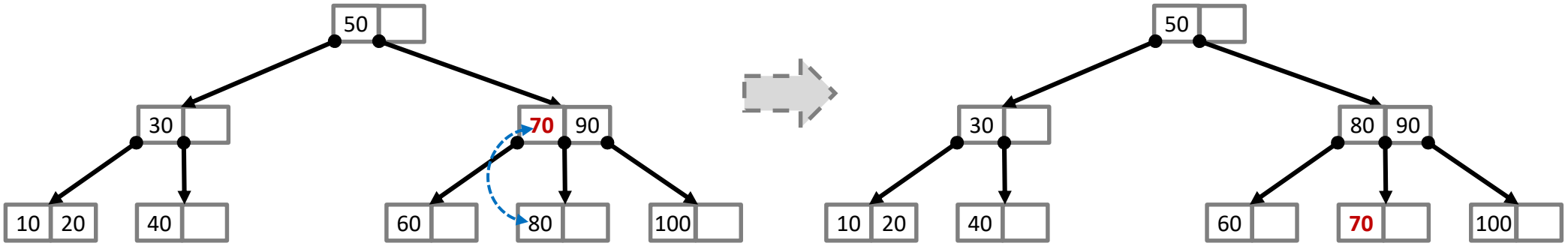
Remoção

- Remoção de “65”
  - Remova “65”
    - Como a folha possui 2 chaves, é possível remover “65”



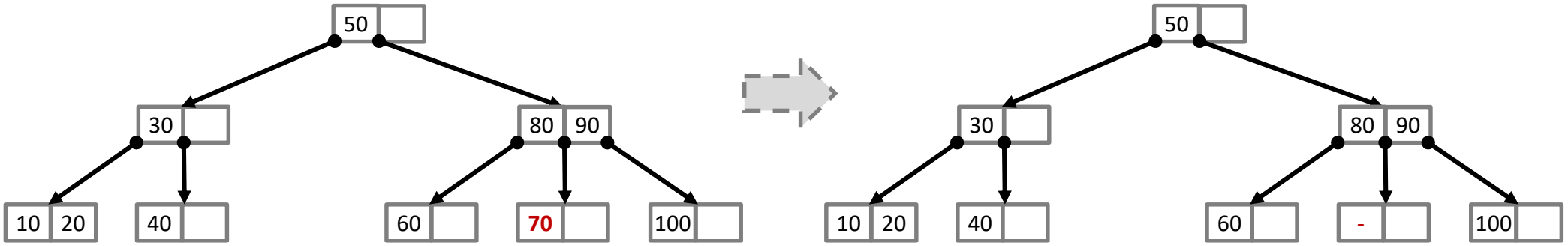
Remoção

- Remoção de “70”
  - O nó que contém “70” é interior
  - Troque “70” com a sua sucessora



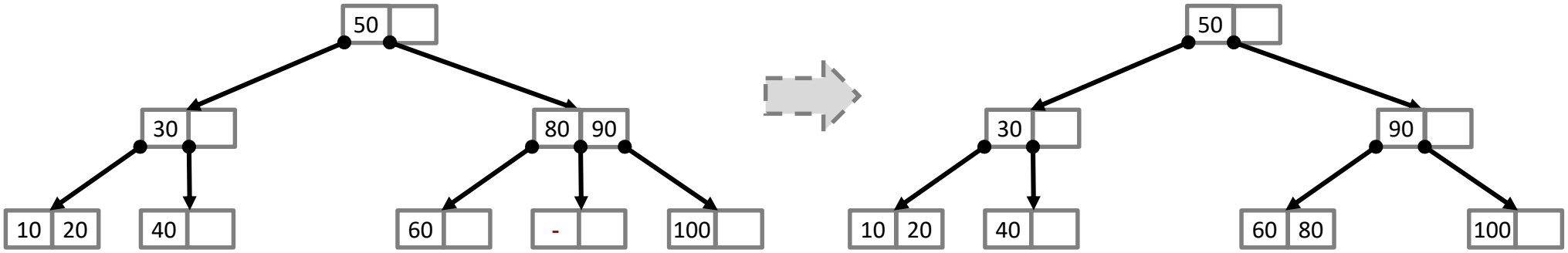
Remoção

- Remoção de “70”
  - Remova “70”



Remoção

- Remoção de “70”
  - A árvore torna-se inválida
  - Combine nós



Resumo

- Árvores 2-3
  - Cada nó armazena 1 ou 2 chaves
  - Cada nó interno possui 2 ou 3 filhos
  - Algoritmos de inserção e remoção mantém as árvores balanceadas
  - Caso especial de Árvores-B

Complexidade da Árvore 2-3

- Busca tem custo O(log n)
- Inserção tem custo O(log n)
- Remoção tem custo O(log n)

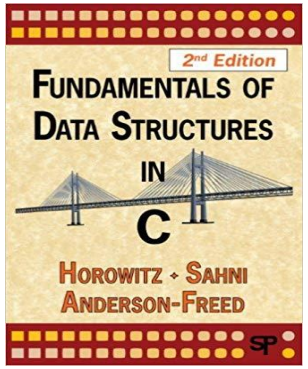
Árvores 2-3

VAMOS EXERCITAR?

Exercício Aula 12
Árvores B

- Explique como ficaria uma árvore B de ordem 3 após cada uma das operações abaixo.
  - Desenhe a árvore em uma folha de papel e explique os passos para cada operação
  - Insira os elementos 30, 60, 10, 50, 20, 40, 70, 90, 80
  - Remova os elementos 20, 60, 70, 50, 80, 30
  - Utilize seu celular para fotografar suas respostas e enviar ao site do EAD.

Leitura Complementar



- Horowitz. E.; Sahni, S.; Anderson-Freed, S. **Fundamentals of Data Structures in C**, 2nd edition. Silicon Press, 2008.
  - Capítulo 11: Multiways Search Trees – 11.2 B-Trees; 11.3 B^+-Trees;

- Kruse, R.; Tondo, C.; Leung, B.; Mogalla, S.; **Data Structures and Program Design in C**, 2nd edition. Pearson, 1996.
  - Capítulo 10: Multiway Trees – 10.3 External Searching: B-Trees