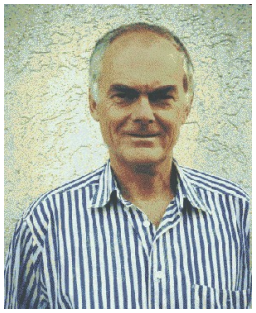
Estruturas de Dados Avançadas (INF1010)

Árvores Rubro-Negras

Árvores Rubro-Negras (red-black)



1972: “Symmetric Binary B-Trees”
Rudolf Bayes

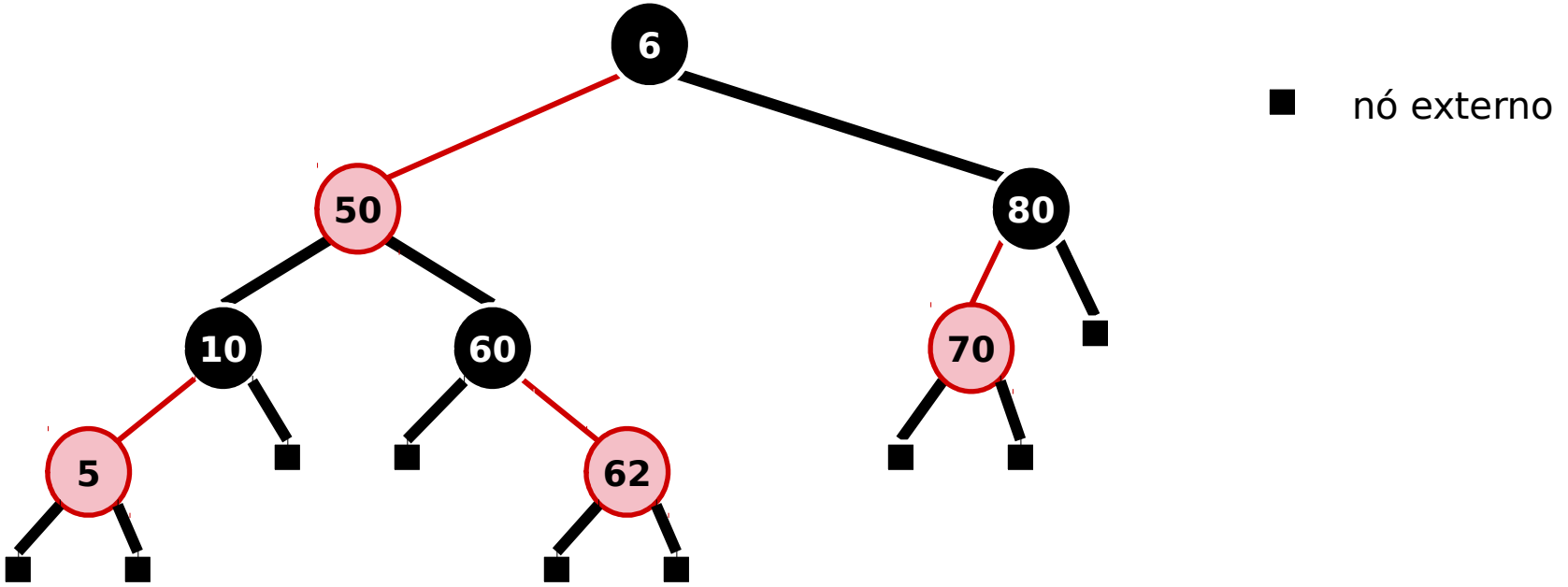
1978: "A Dichromatic Framework for Balanced Trees"
Leonidas J. Guibas e Robert Sedgewick

Árvore binária de busca auto-balanceável.

Buscas, inserções e remoções $O(\log_n)$.

“Complete fair scheduler” (Linux), aplicações de geometria computacional, estruturas auxiliares.

Árvore Rubro-Negra: restrições



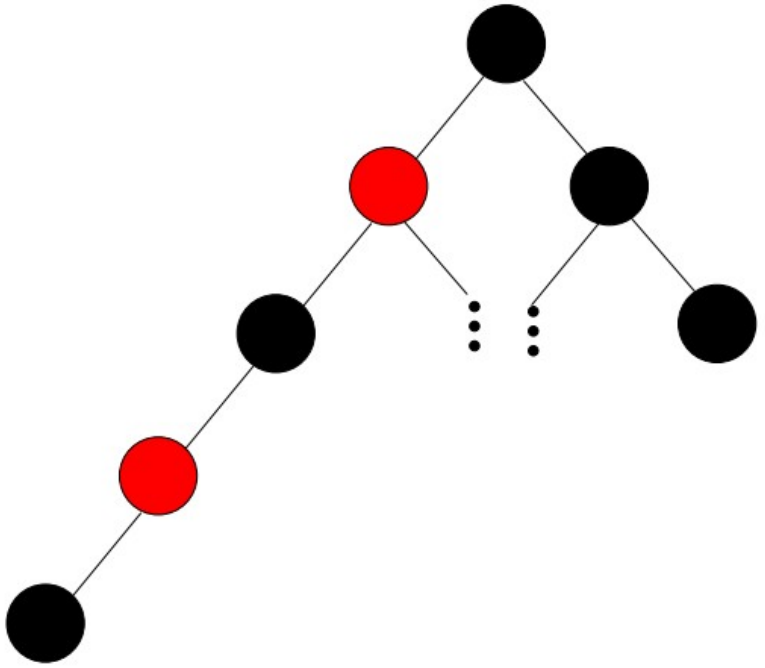
Todos os nós são vermelhos ou pretos

RB1: A raiz e as folhas (nós vazios/NULL) são pretos

RB2: Nós vermelhos têm apenas filhos pretos

RB3: Todos os caminhos da raiz até suas folhas têm o mesmo número de nós pretos

Árvore Rubro-Negra: propriedade

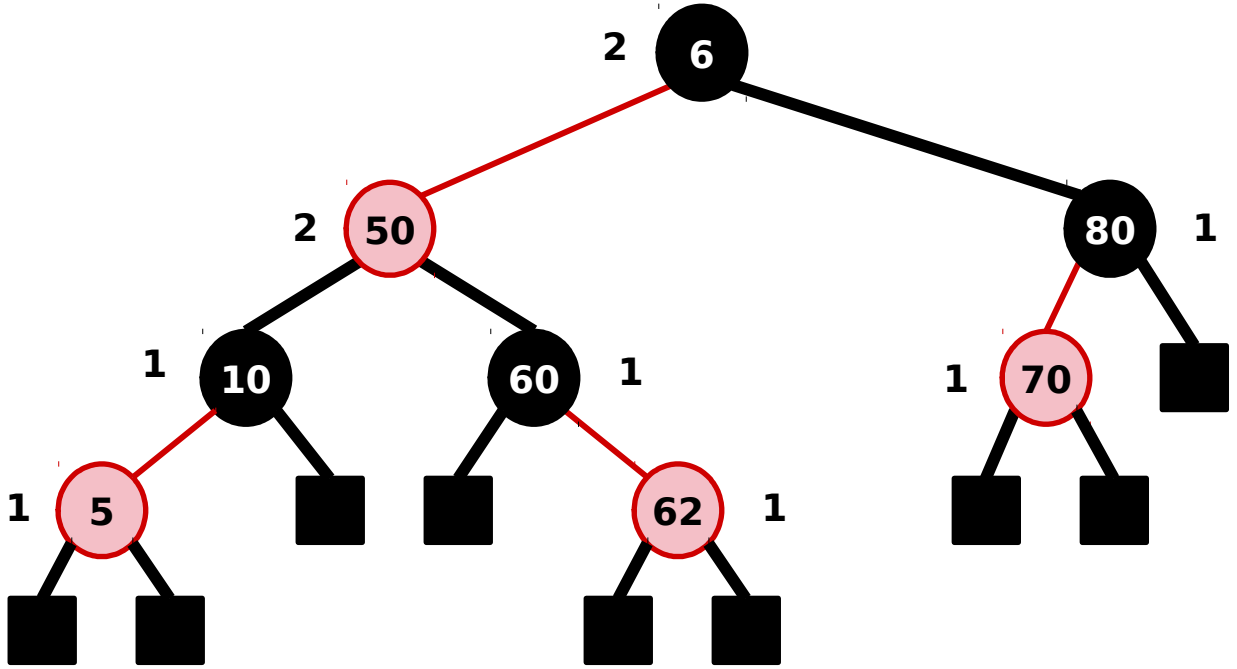


balanceamento **razoável**

o maior caminho entre a raiz e uma folha é **no máximo** o dobro do menor caminho

- um caminho mais longo terá nós vermelhos, que não serão consecutivos
- os caminhos da raiz até suas folhas têm o mesmo número de nós pretos

Árvore Rubro-Negra: conceitos



$h \leq 2r$
$n \geq 2^r - 1$
$h \leq 2 \log_2(n+1)$

rank (bh, black height) de um nó: número de arestas pretas até um nó externo (número de nós pretos – 1)

altura (h) da árvore, sem contar os nós externos

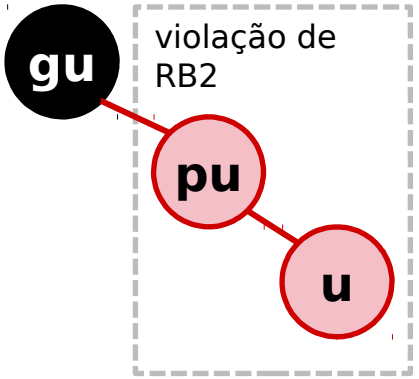
Árvore Rubro-Negra: Inserção

Árvore vazia: novo nó será preto

- é uma raiz → para não violar **RB1**

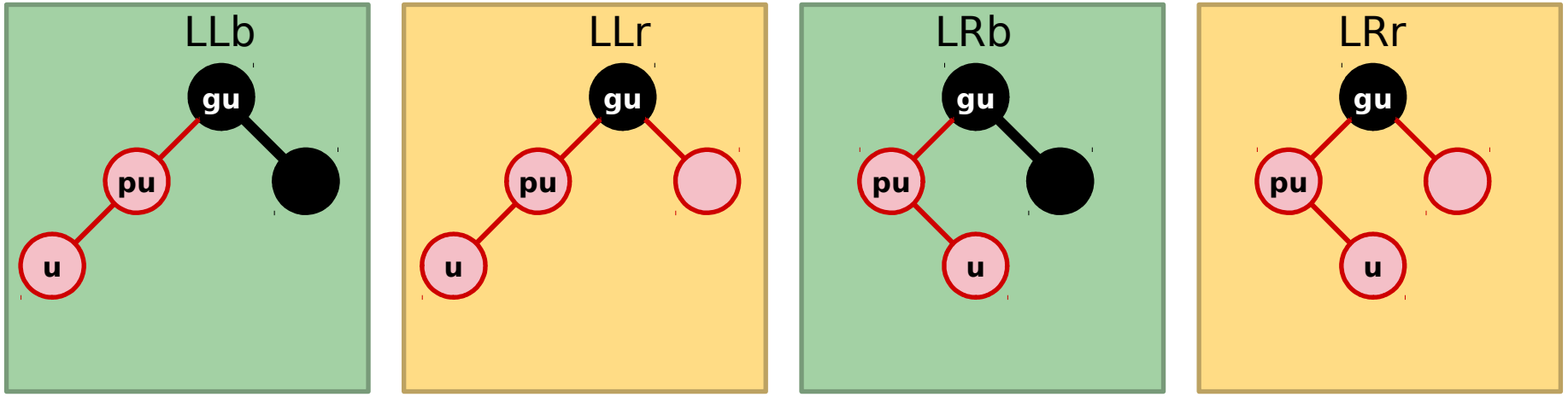
Árvore não vazia: novo nó será vermelho

- se fosse preto, violaria **RB3**, pois os filhos do novo nó seriam pretos (aumentando o número de nós pretos no caminho
- pode ou não violar **RB2** (se o pai for vermelho)



nova árvore está **desbalanceada**

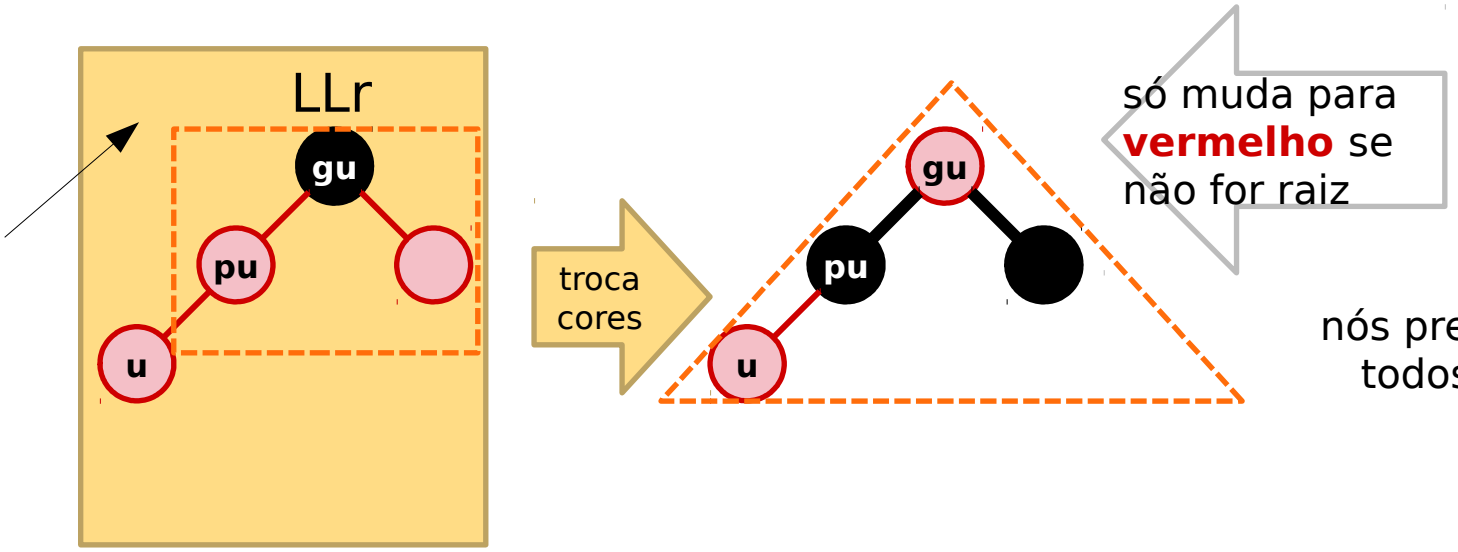
Tipos de Desbalanceamento



- Desbalanceamento do tipo XYr é tratado apenas por mudança de cores
  - a violação de **RB2** pode propagar dois níveis acima!

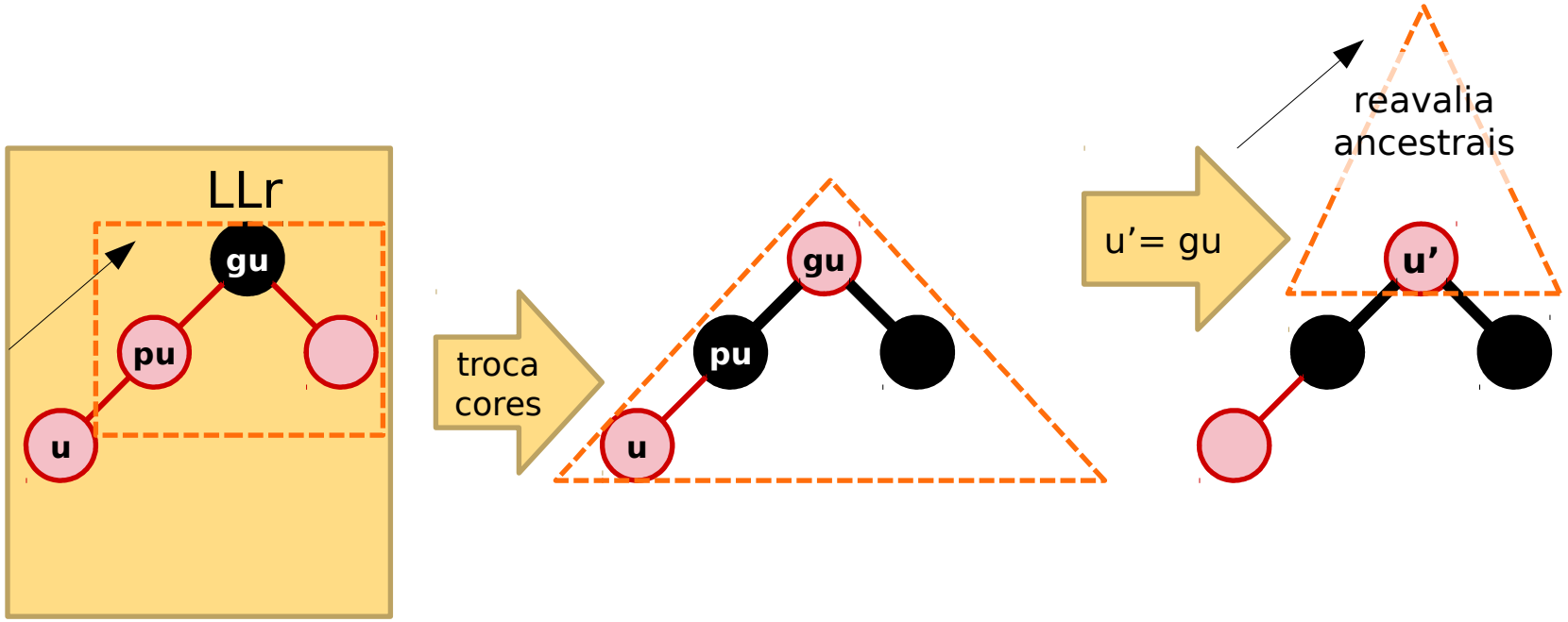
- Desbalanceamento do tipo XYb requer rotação
  - não é necessário propagar

Correção por Mudança de Cor

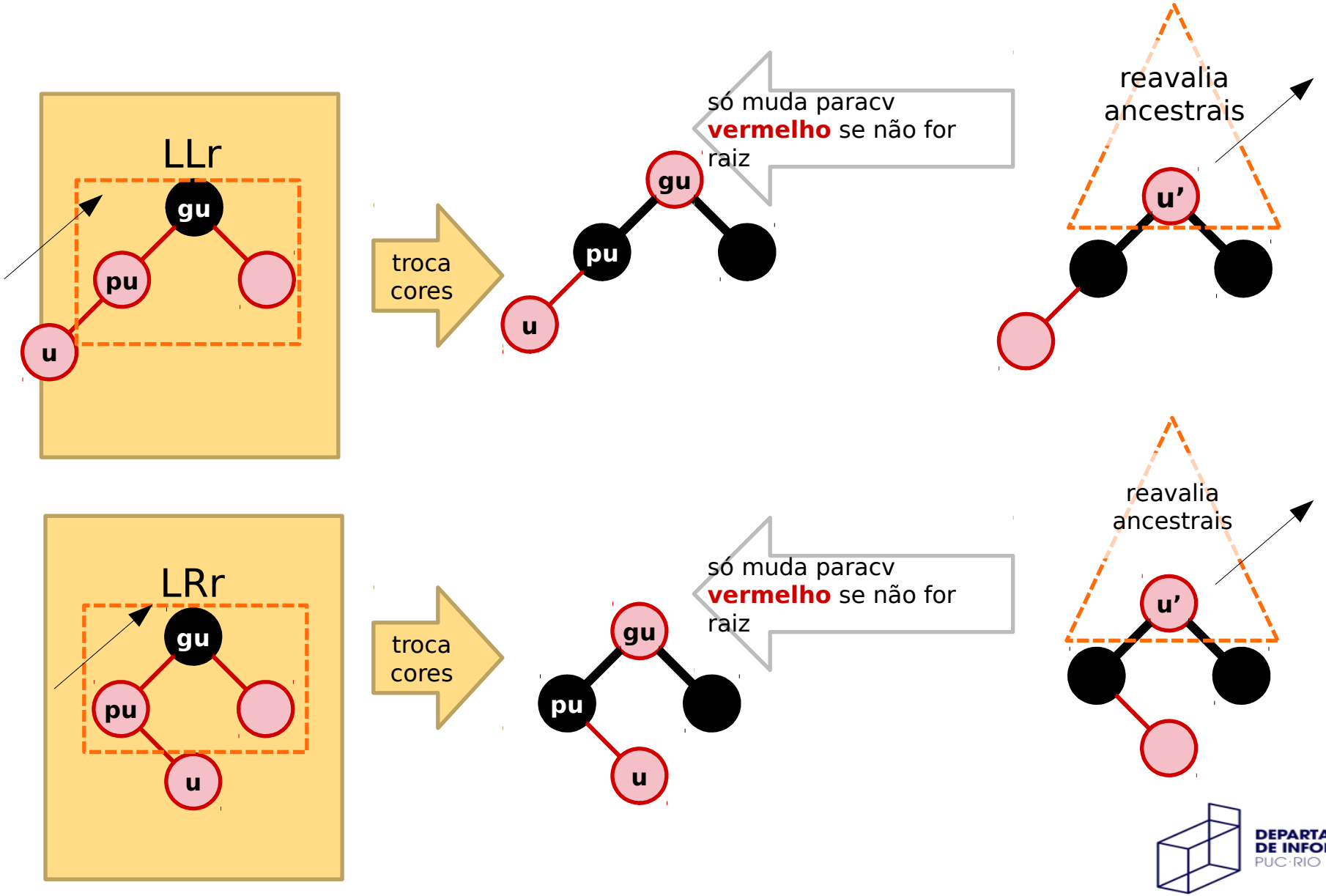


aqui correto mas e o pai de **gu**?

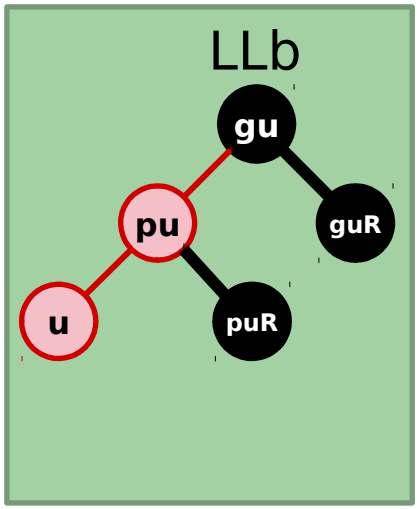
Correção por Mudança de Cor



Correção por Mudança de Cor

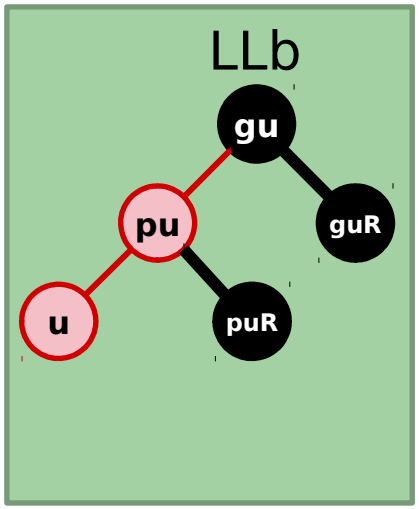


Correção por Rotação

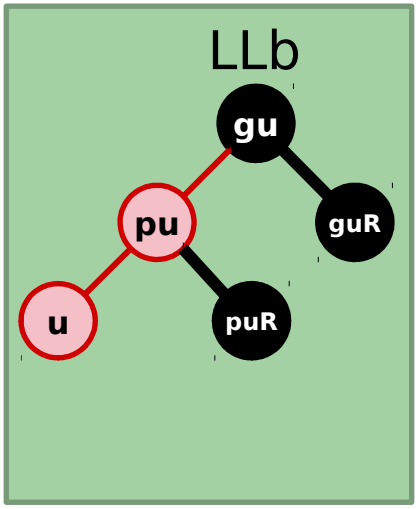


aqui não tem como mudar as cores sem alterar a contagem de nós negros em caminhos!!!

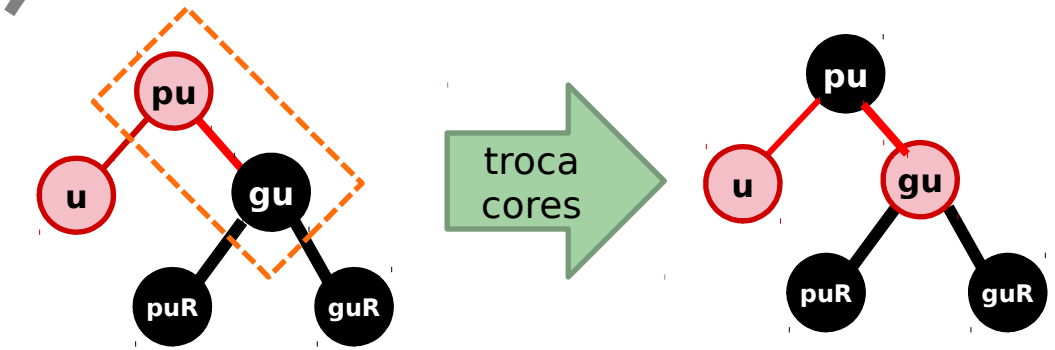
Correção por Rotação



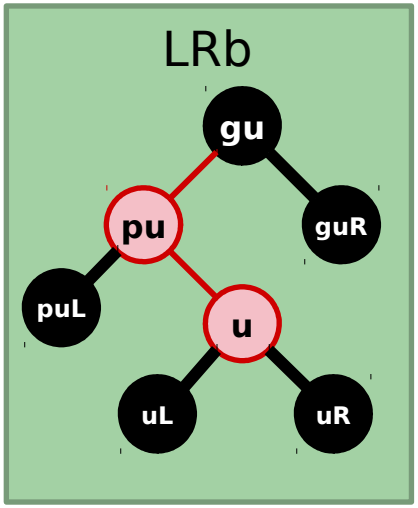
aqui não tem como mudar as cores sem alterar a contagem de nós negros em caminhos!!!



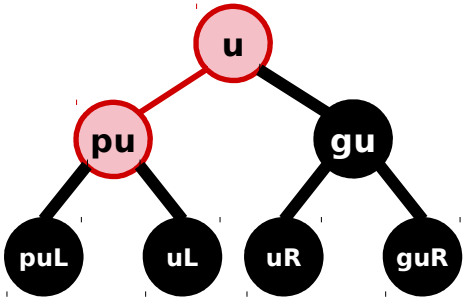
rotação para direita

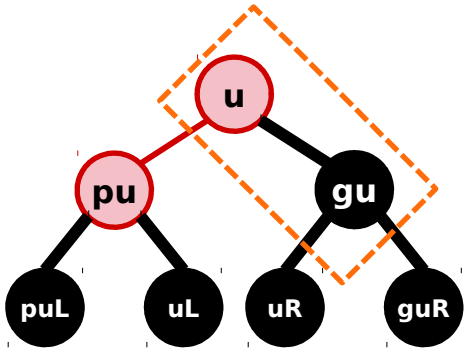


Correção por Rotação

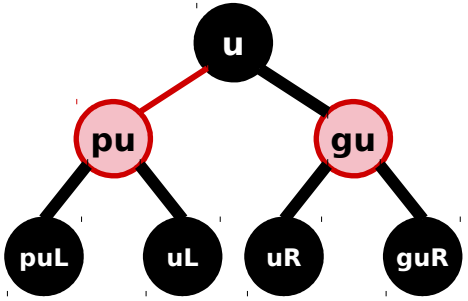


rotação dupla
esquerda-direita

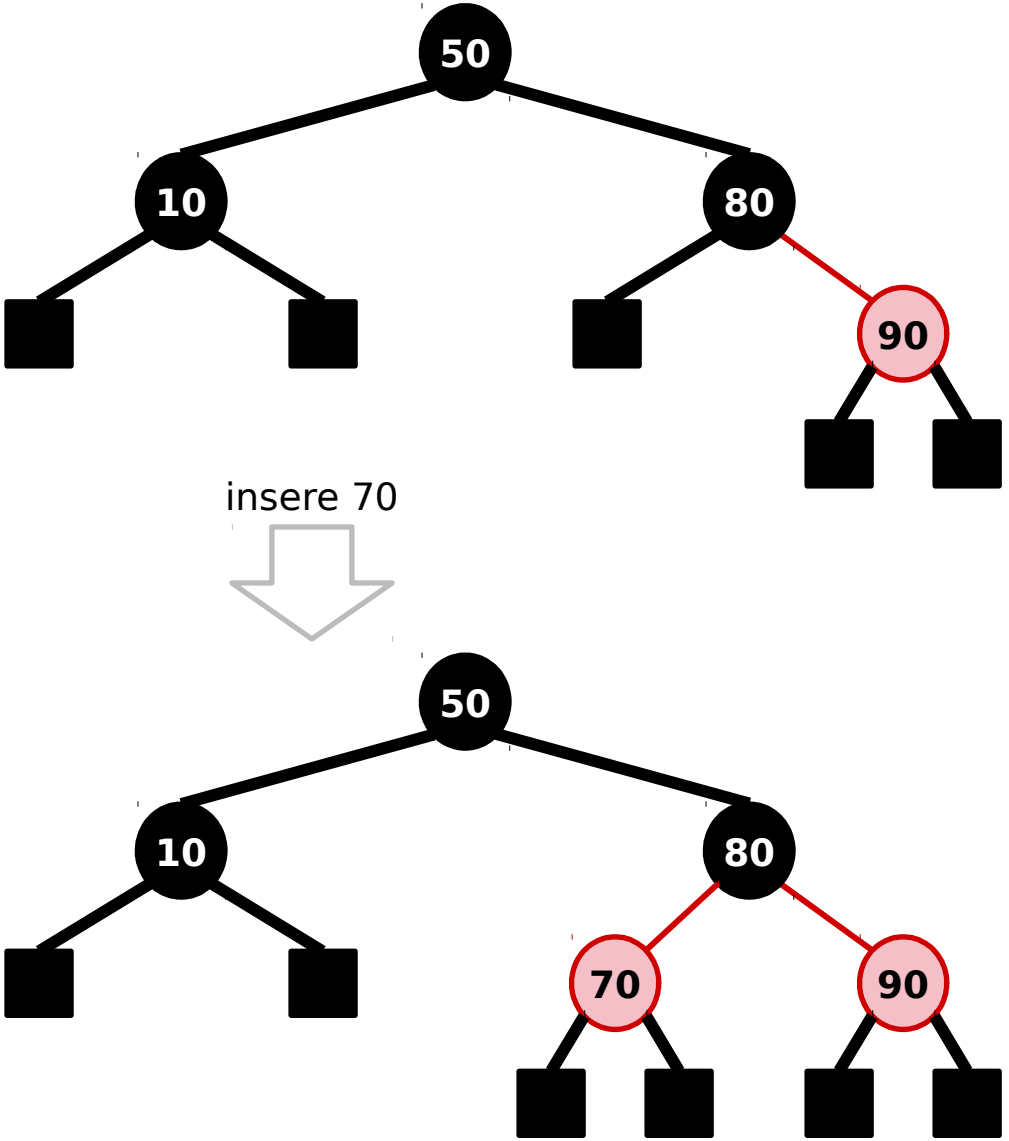




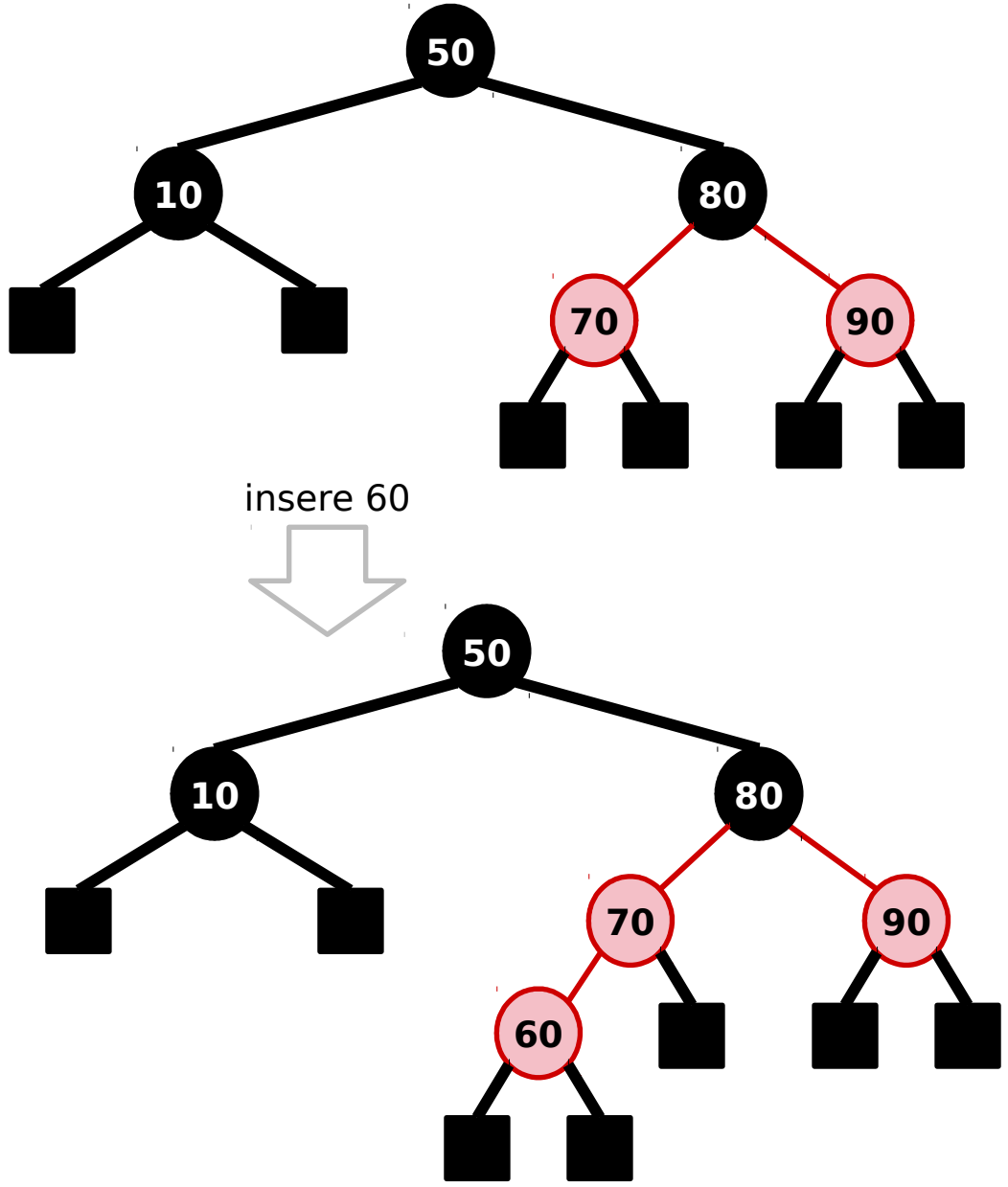
troca
cores



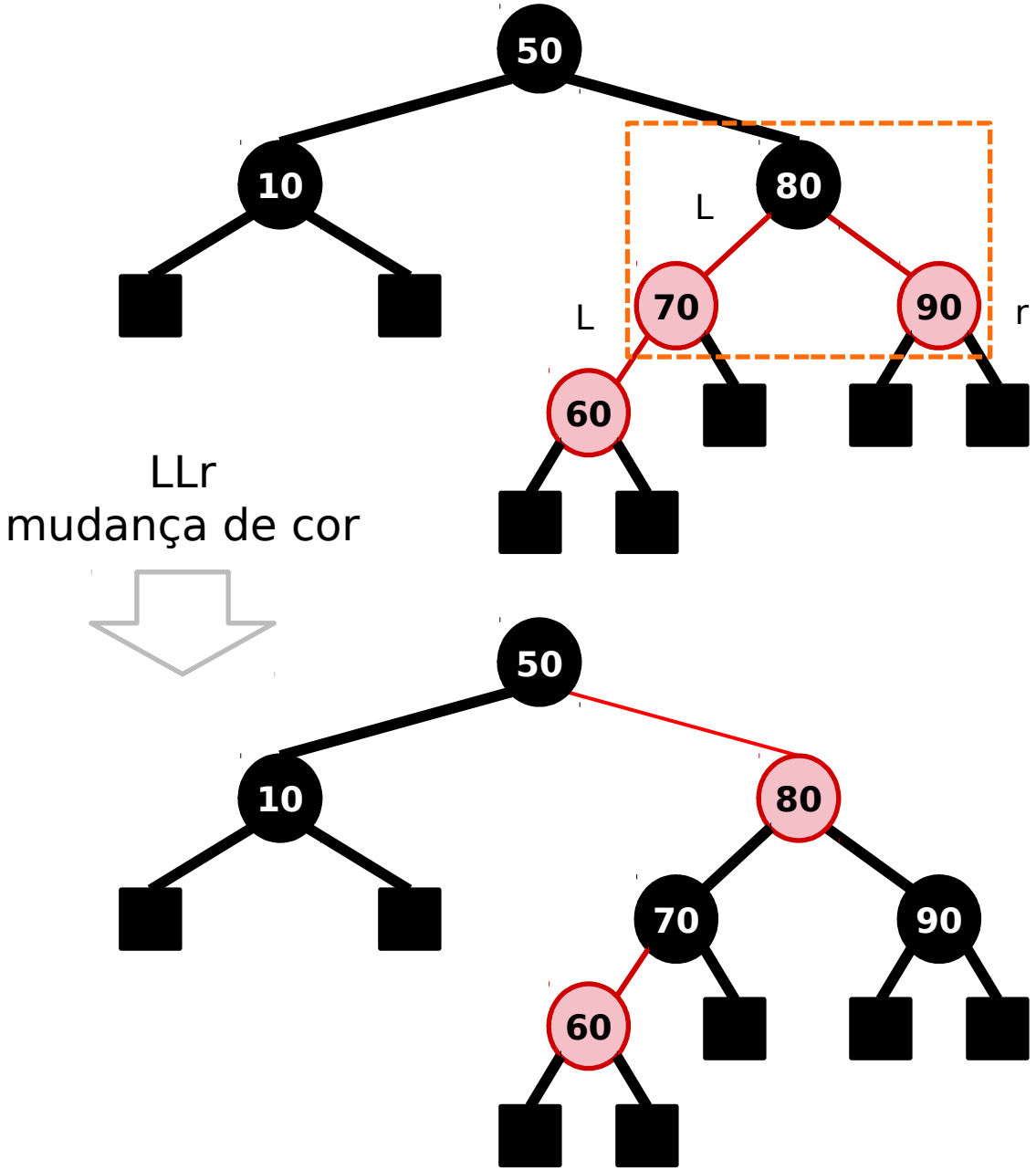
Exemplos de Inserção



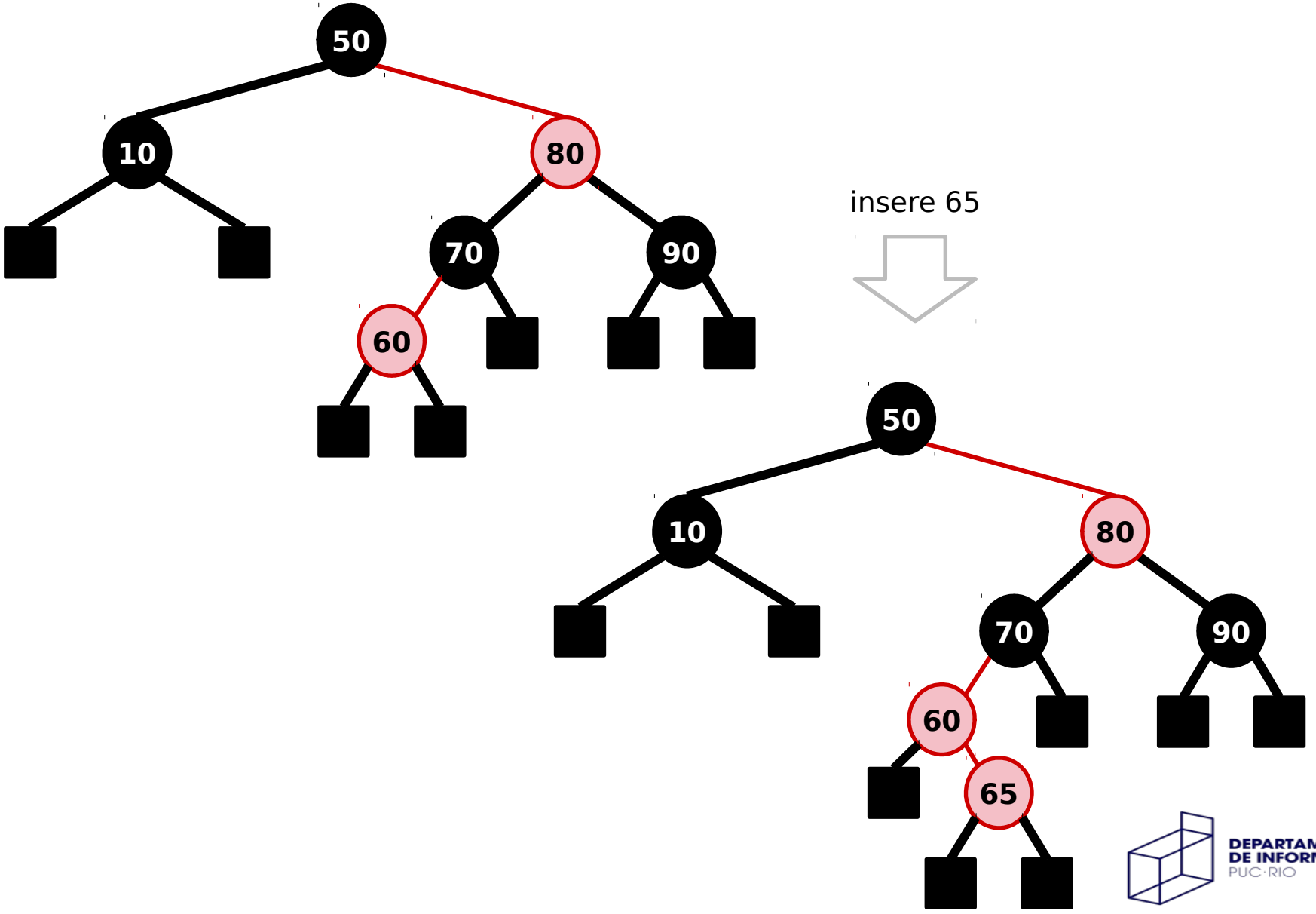
Exemplos de Inserção



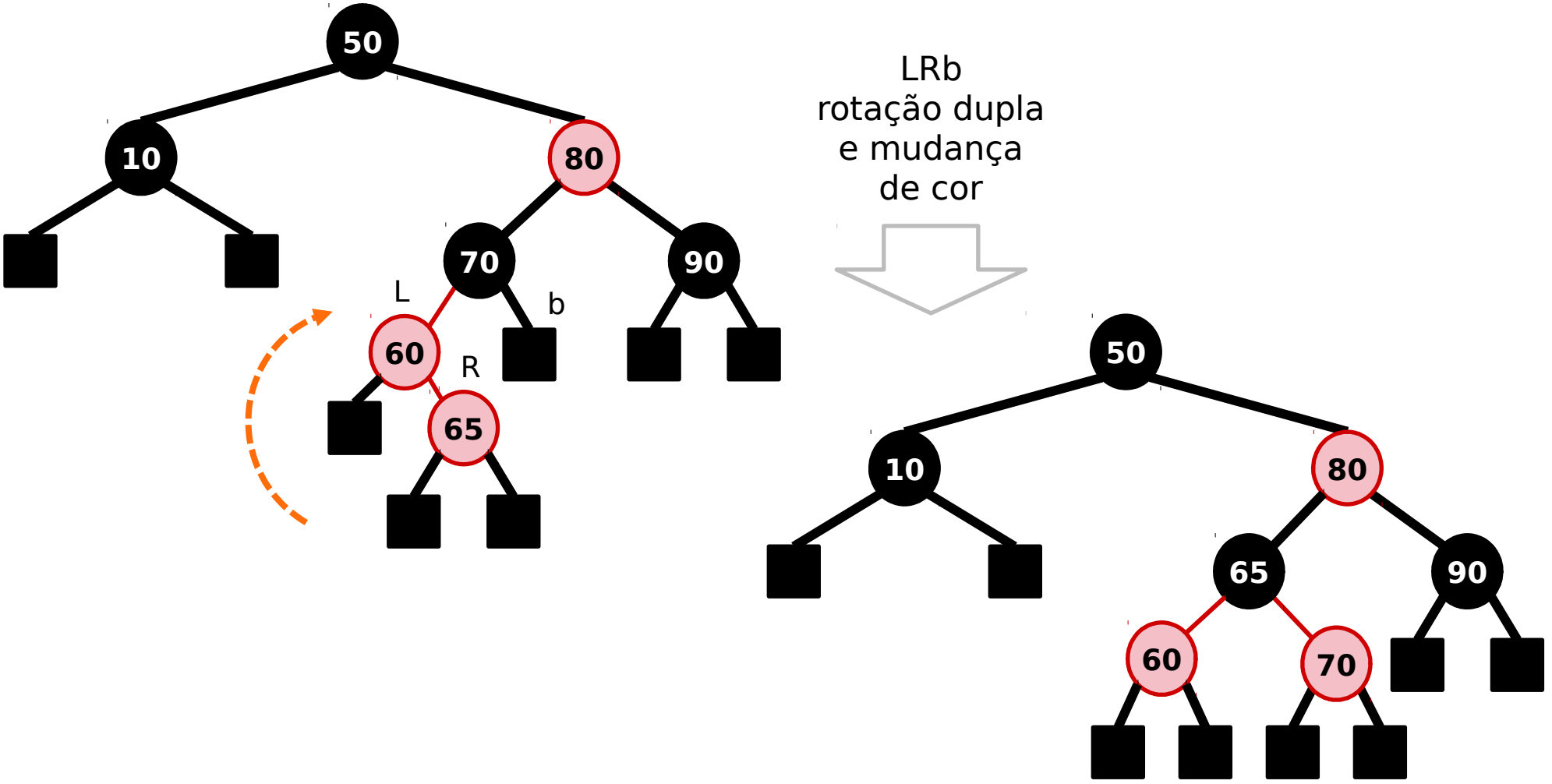
Exemplos de Inserção



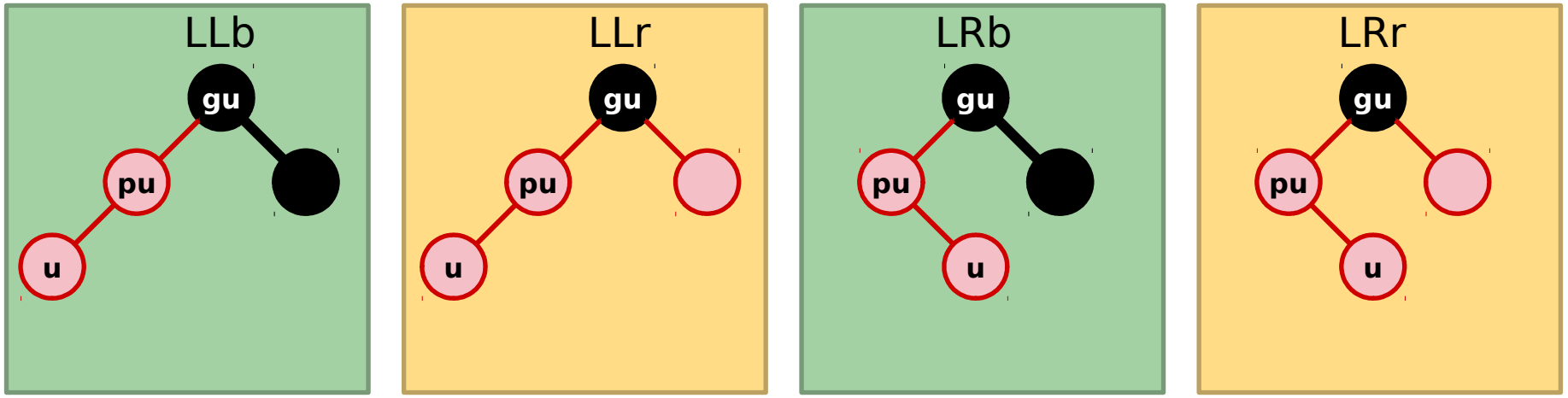
Exemplos de Inserção



Exemplos de Inserção



Inserção à Esquerda



Inserção de u propaga para o pai (“VERMELHO”)

Se pai é vermelho, propaga para o avô a inserção à ESQUERDA

Avô verifica a cor do “tio”

- se tio é vermelho, troca cores (e propaga “VERMELHO”)
- se tio é preto, verifica de onde veio a inserção
  - pela ESQUERDA (LLb) → rotaciona à direita e troca cores
  - pela DIREITA (LRb) → rotação dupla esq/dir e troca cores

Inserção em Árvore Rubro-Negra

```
struct smapa {
   char vermelho;   /* 0 preto, 1 vermelho */
   int chave;
   int dado;
   struct smapa* esq;
   struct smapa *dir;
};

typedef enum result {OK, RED, LEFTRED, RIGHTRED} Result;

Mapa* insere (Mapa* m, int chave, int novodado) {
   Result status;
   m = insereRec (m, chave, novodado, &status);
   if (status == RED) m->vermelho = 0;
  return m;
}
```

Inserção em Árvore Rubro-Negra

```
static Mapa *cria_no (int c, int novodado) {
  Mapa *m = (Mapa *)malloc(sizeof(Mapa));
  if (m!=NULL) {
    m->esq = nn->dir = NULL;
    m->chave =c;
    m->vermelho = 1;
    m->dado = novodado;
  }
  return m;
}

static Mapa* insereRec (Mapa* m, int chave, int novodado, Result* status) {
  if (m==NULL) {
    m = cria_no (chave, novodado);
    *status = RED;
  }
  else if (chave < m->chave) {
    m->esq = insereRec (m->esq, chave, novodado, status);
    m = corrigeEsq (m, status);
  }
  else if (chave > m->chave) {
    m->dir = insereRec (m->dir, chave, novodado, status);
    m = corrigeDir (m, status);
  }
  return m;
}
```

Inserção em Árvore Rubro-Negra

```
static Mapa* corrigeEsq (Mapa *m, Result* status) {
   switch (*status) {
     case OK:    /* não há correção daqui para cima */
         break;
    case RED:
         if (m->vermelho) *status = LEFTRED;   /* propaga para avô */
         else *status = OK;  /* filho vermelho, pai preto */
         break;
    case LEFTRED:
         if (!m->dir || !m->dir->vermelho) {  /* tio vazio/preto */
             /* LLb – rotaciona à direita e troca cores  */
         }
         else  { /* tio é vermelho */
            /* LLr – troca cores  */
         }
      break;
    case RIGHTRED:
      if (!m->dir || !m->dir->vermelho) {  /* tio vazio/preto */
            /* LRb – rotação dupla esquerda-direita e troca cores  */
      else {  /* tio é vermelho */
           /* LRr – troca cores  */);
      }
      break;
  }
  return m;
}
```