BST Rubro Negra (Red Black)

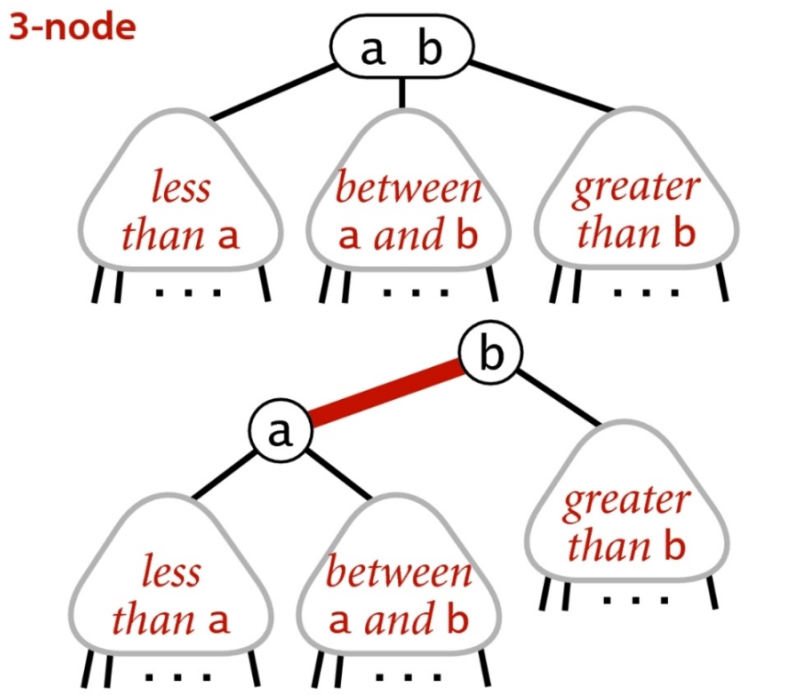
→ Uma maneira de Implementar árvores 2-3.

→ Cada nó duplo da árvore 2-3 é representado por dois nós simples ligados por um link rubro.

→ Nossas BST são esquerdistas (left-leaning), pois os links rubros estão sempre inclinados para a esquerda.

→ ARNE (Árvore Rubro-Negra esquerdista)

→ Representação de um nó duplo por meio de dois nós simples ligados por um link rubro:

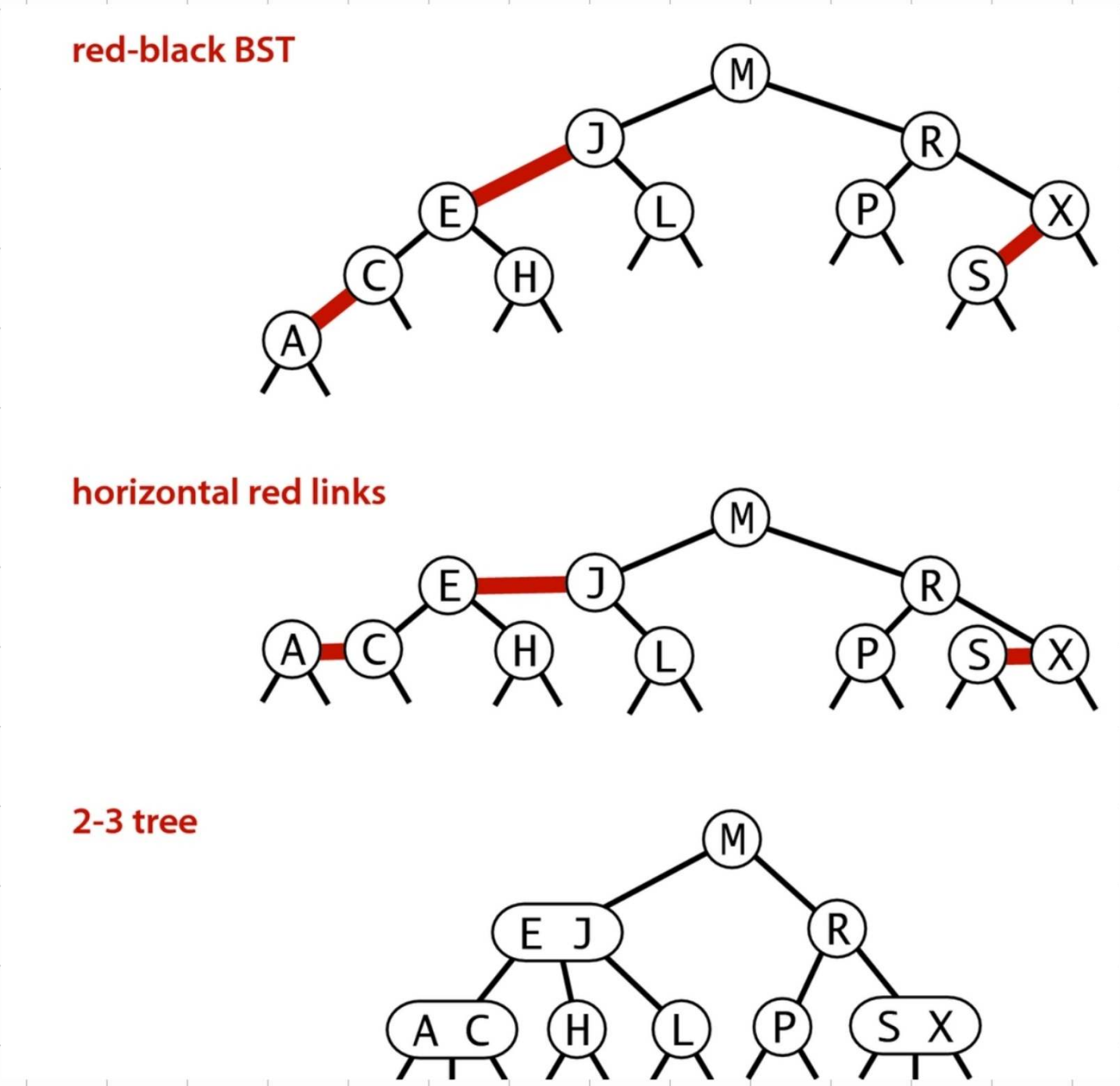


Encoding a 3-node with two 2-nodes connected by a left-leaning red link

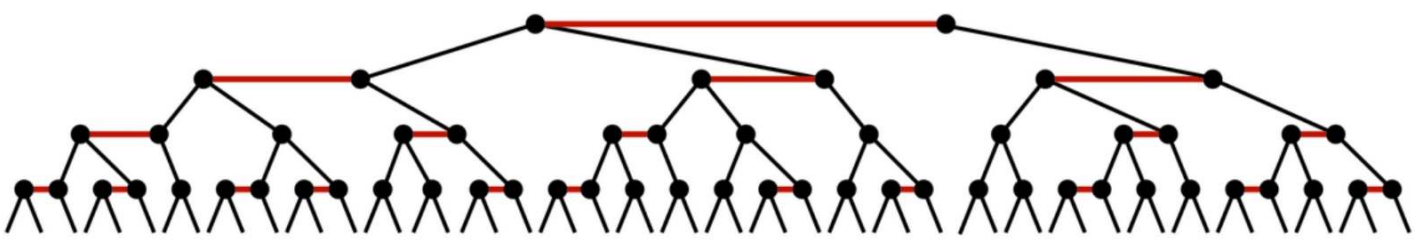
⇒ Definição: Uma BST rubro-negra é uma BST cujos links são negros e rubros e têm as seguintes propriedades:

- links rubros se inclinem para a esquerda;
- nenhum nó incide em dois links rubros;
- balanceamento negro perfeito: todo caminho da raiz até "z" tem o mesmo número de links negros

- Se os links rubros forem desenhados horizontalmente e depois contraídos, teremos uma árvore 2-3:



- Todos os links "j" estão à mesma profundidade negra:



A red-black tree with horizontal red links is a 2-3 tree

- A profundidade negra de um nó "x" é o número de links negros no caminho da raiz até "x".

- A altura negra de árvore é o máximo da profundidade negra de todos os nós

- Como armazenar a informação de link Rubro e Negra?

→ Da BST ORIGINAL

- typedef struct STNode * link;
  struct STNode { Item item; link l, r; int N; int color; };

enum tipo { RED, BLACK; };
#define RED 0
#define BLACK 1

✓ link h, z;

✓ link NEW(Item item, link l, link r, int N)
{ link x = malloc(sizeof(*x)); x->color = RED;
x->Item = item; x->l = l; x->r = r; x->N = N;
return x;
}

✓ Void STinit( )
{ h = (z = NEW(NULLItem, 0, 0, 0)); z->color = BLACK; }

int STcount( )
{ return h->N; }

- Como fazer Busca?

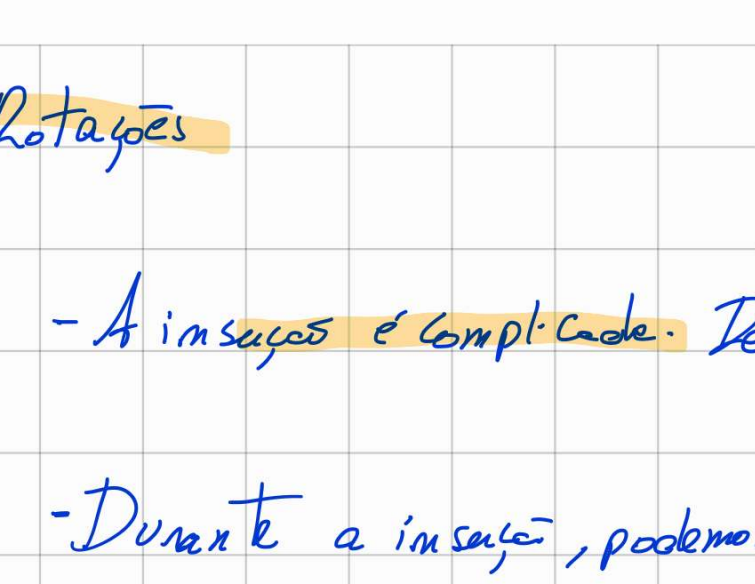
Rotações

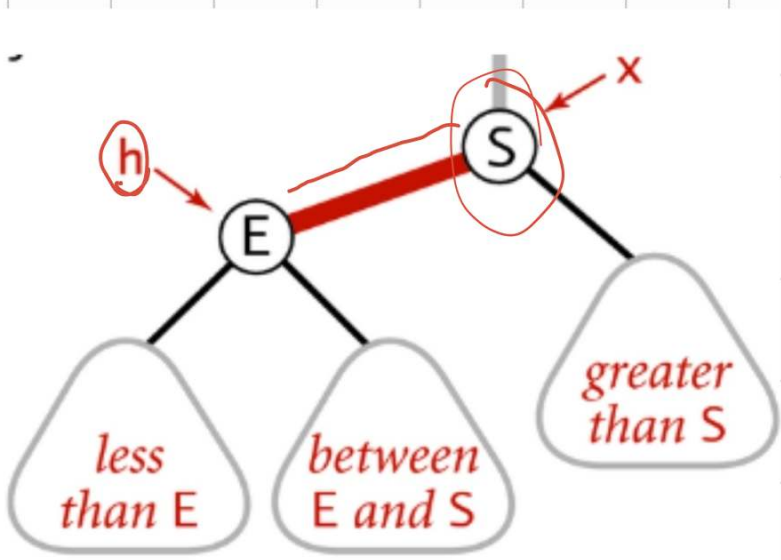
- A inserção é complicada. Depende de rotações

- Durante a inserção, podemos ter, temporariamente, um link rubro inclinado para a direita ou dois links rubros incidindo no mesmo nó.

— Para corrigir usamos rotações

- Rotação Esquerda (anti-horária) em torno de um nó h: o filho direto de h "sobe" e adota h como seu filho esquerdo. Continuemos com uma BST, mas com raiz diferente:



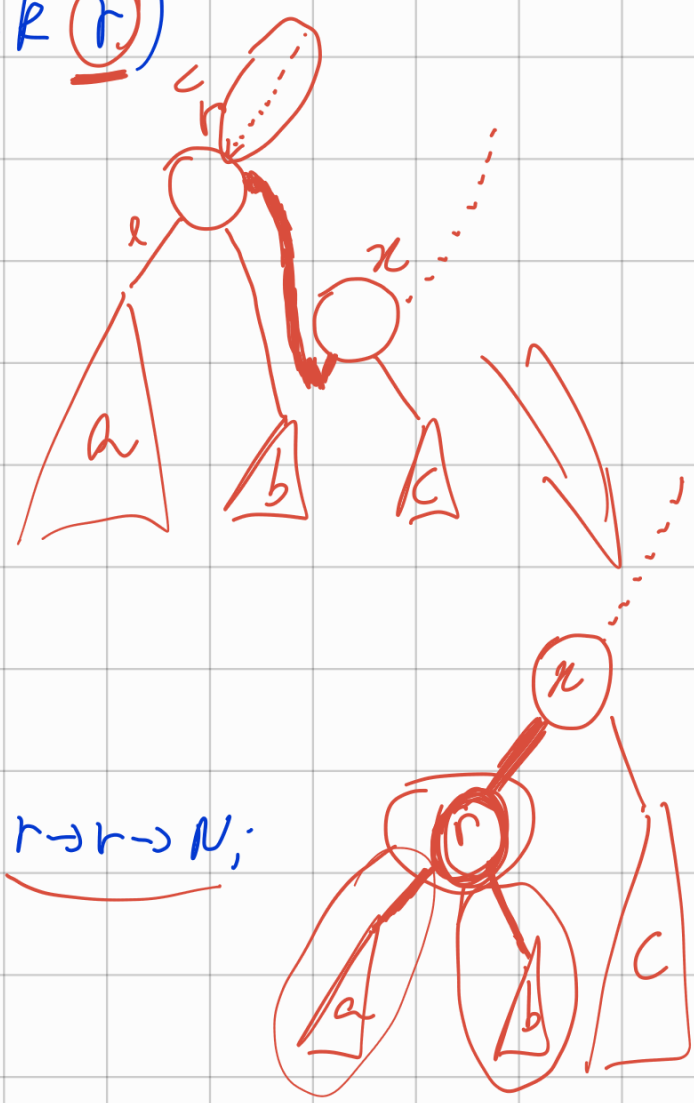


Left rotate (right link of h)

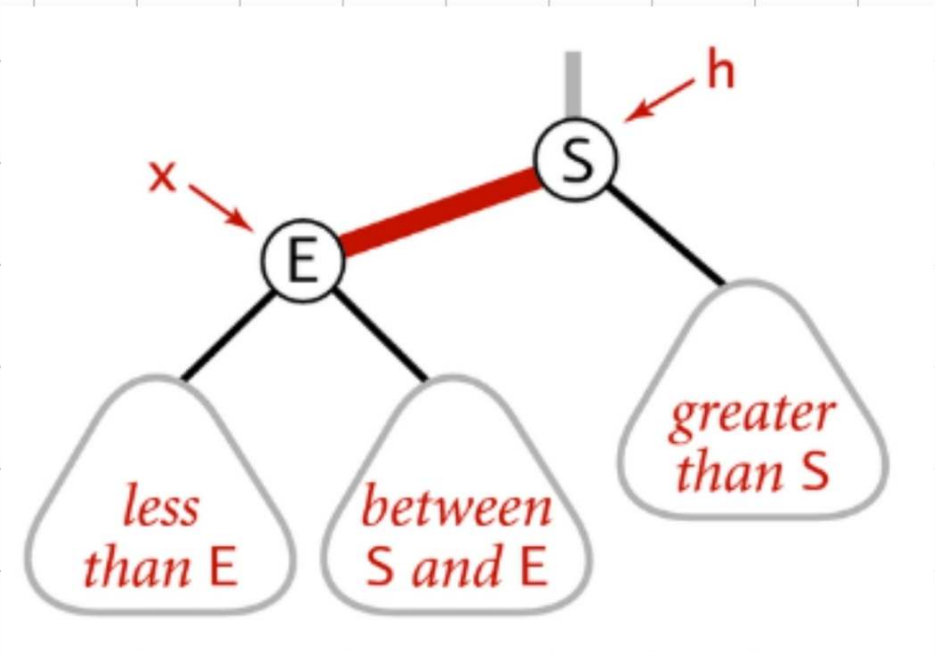
- Como implementar?

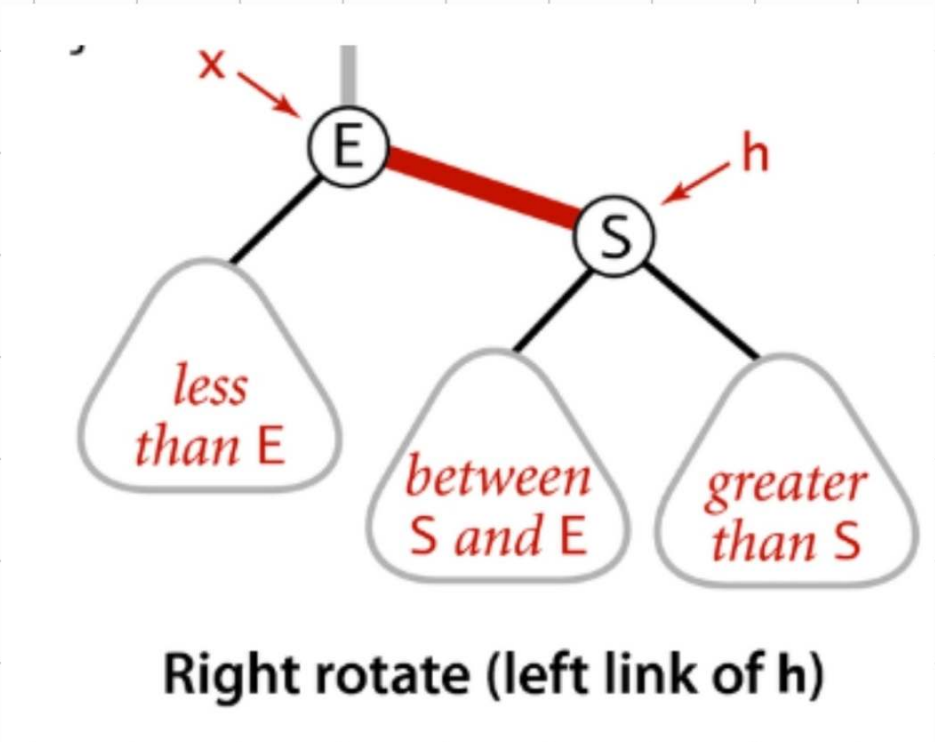
- Feito com:
  - Com dois links
  - Quantidade de chaves

```
link rotateLeft(link r)
{
   link x = r->r;
   r->r = x->l;
   x->l = r;
   x->color = r->color;
   r->color = RED;
   x->N = r->N;
   r->N = 1 + r->l->N + r->r->N;
   return x;
}
```



→ Rotação direita (ou horária) em torno de um nó h:
(O filho esquerdo de h "sobe" e adota h como seu filho direito. Continuamos com uma BST, mas raiz diferente

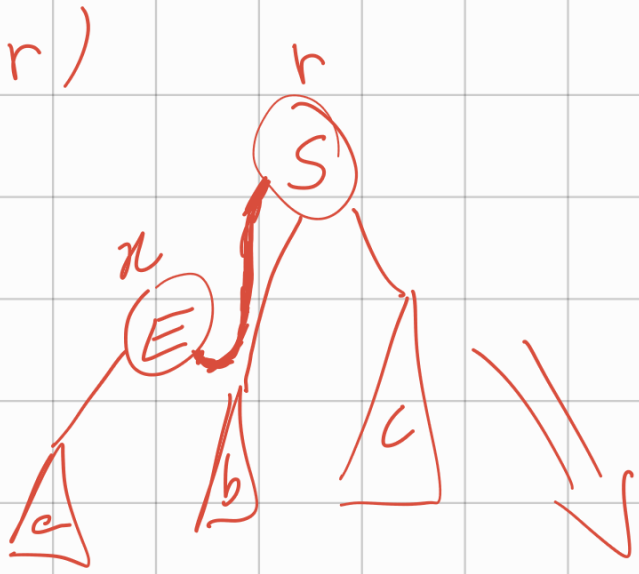


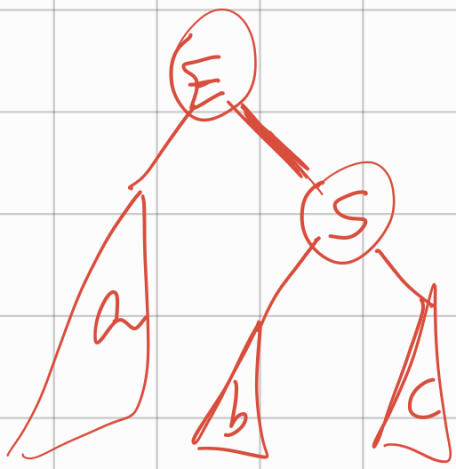


Right rotate (left link of h)

- Como implementar?

```
link rotateRight(link r)
{
    link x = r->l;
    r->l = x->r;
    x->r = r;
    x->color = r->color;
    r->color = RED;
    x->N = r->N;
    r->N = 1 + r->l->N + r->r->N;
    return x;
}
```

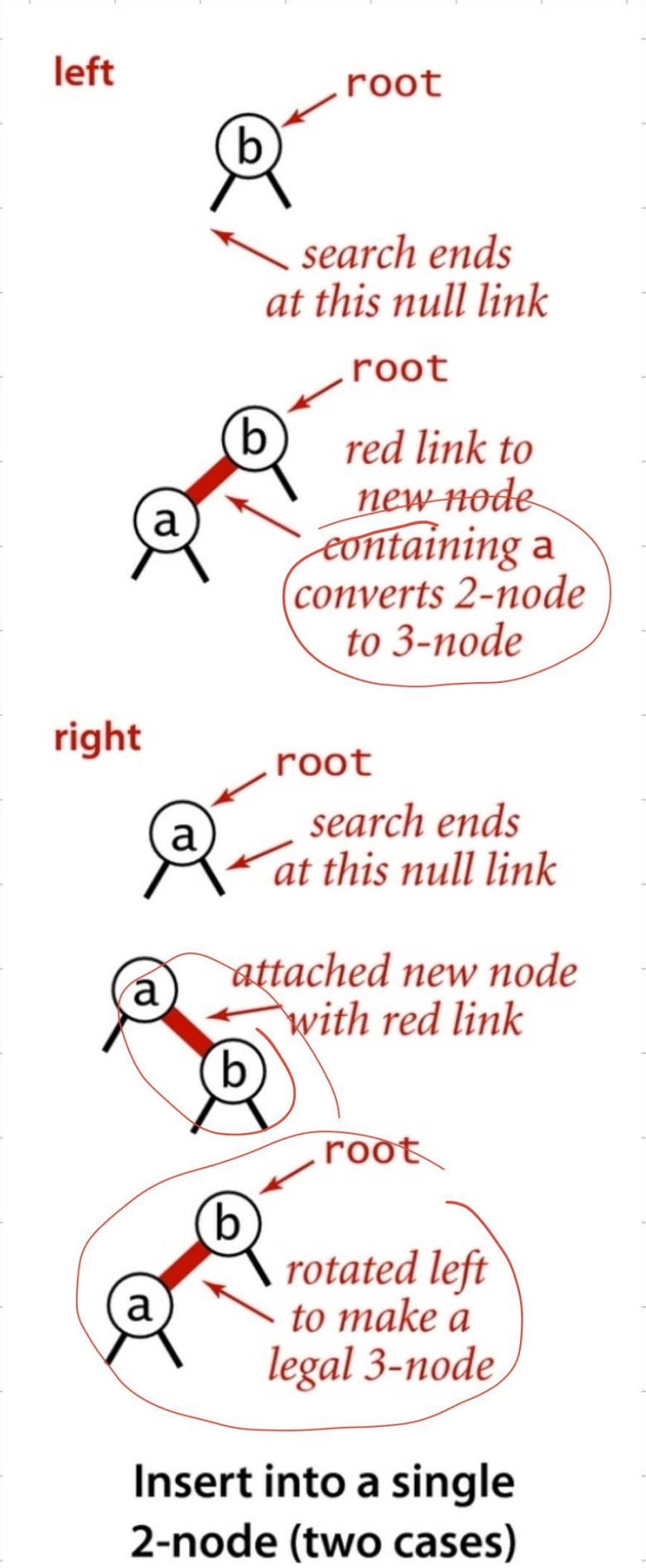




- ~~Após~~ as operações de rotação SE locais.
- Depois de uma rotação, continuamos tendo uma BST com balenceamento negro perfeito.
  - A operação pode criar um link rubro inclinado para o lado errado ou dois links rubros seguidos. (!!)

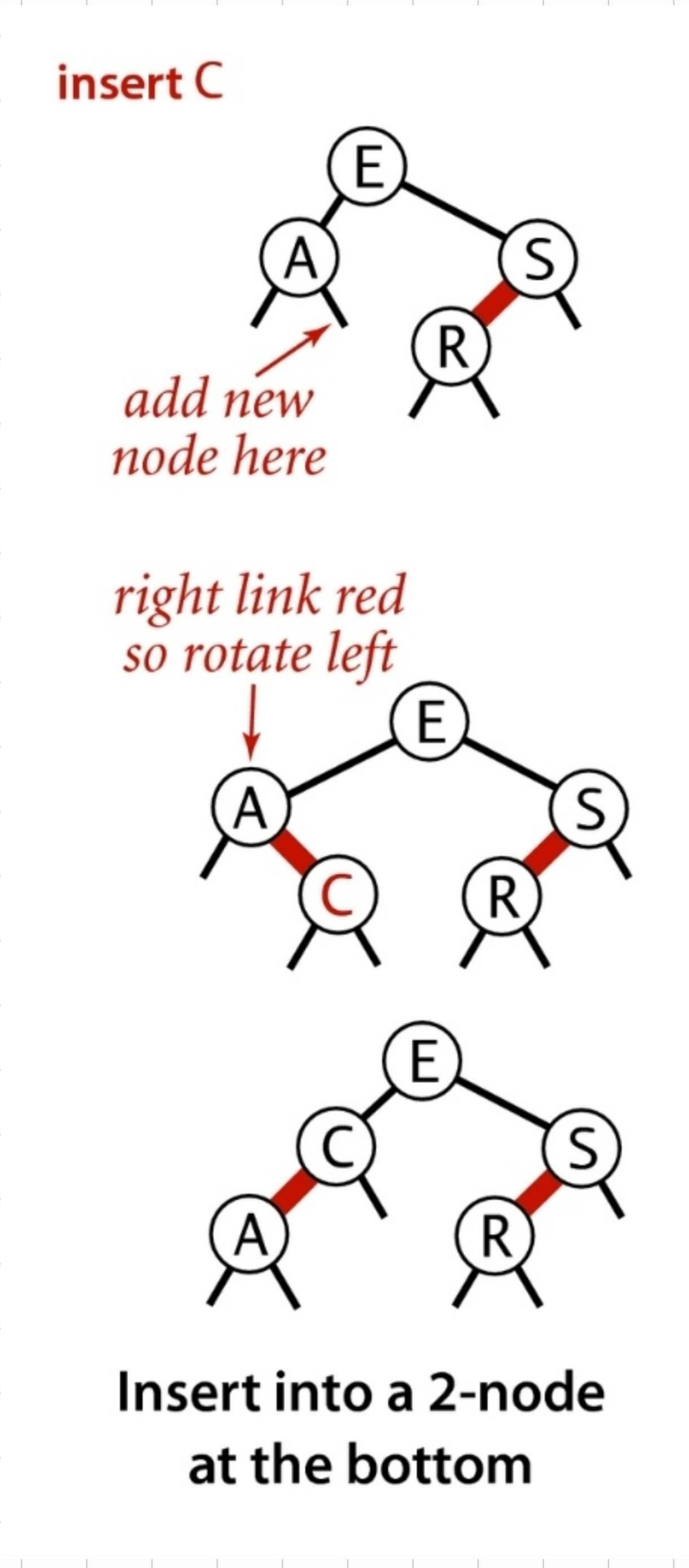
Inserção

- Um novo nó se liga a árvore por um link rubro.

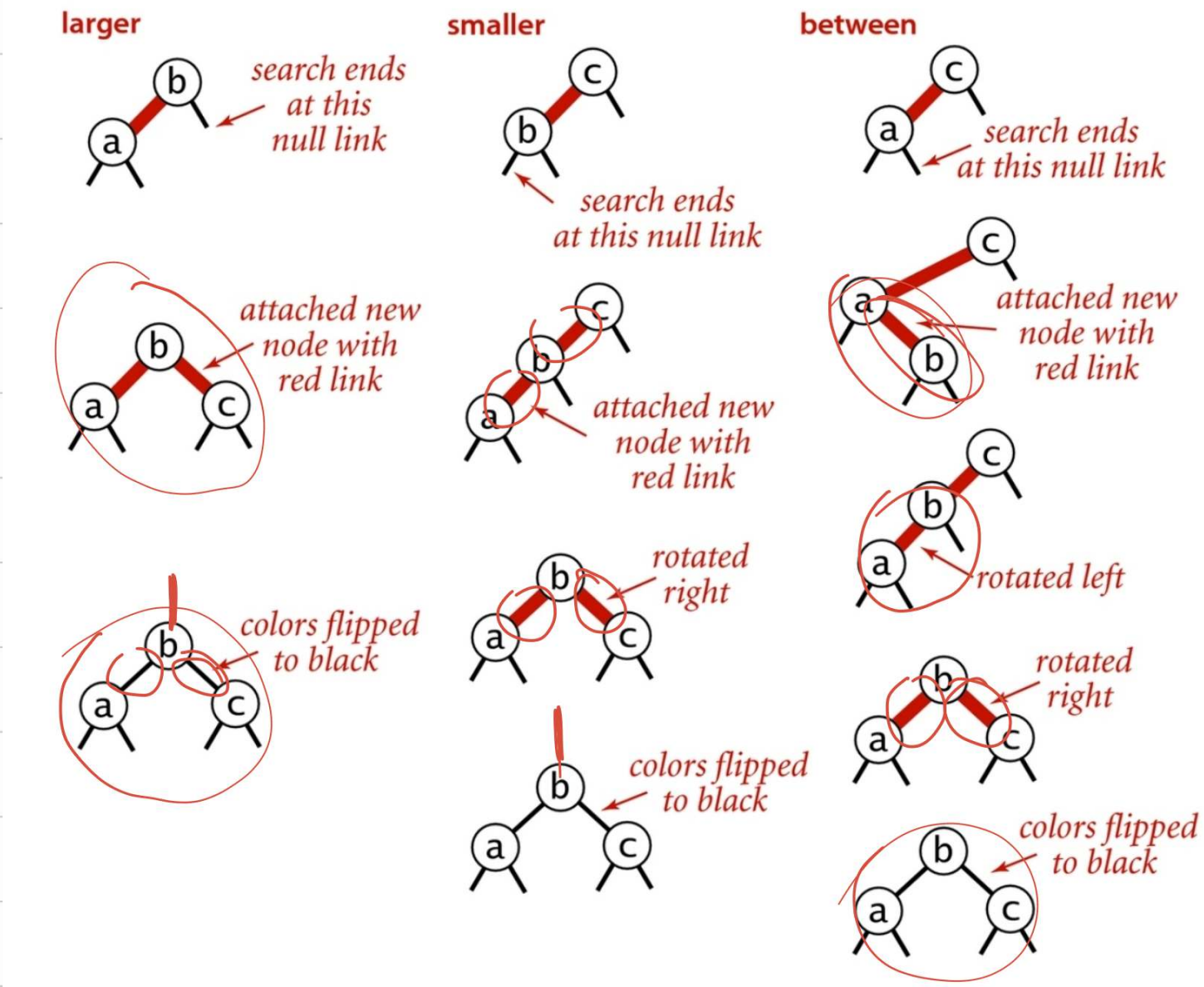


Insert into a single 2-node (two cases)

- Caso 1: Pendurar um novo nó em um nó simples



→ Caso 2: Pendurar um novo nó em um nó duplo

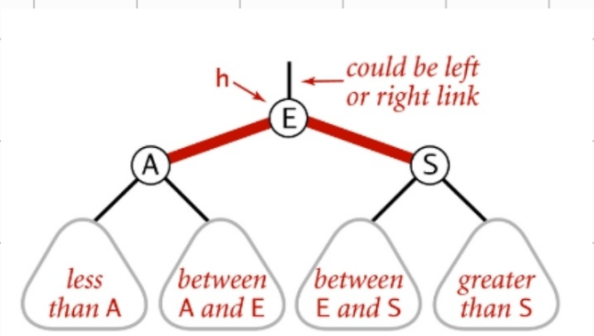


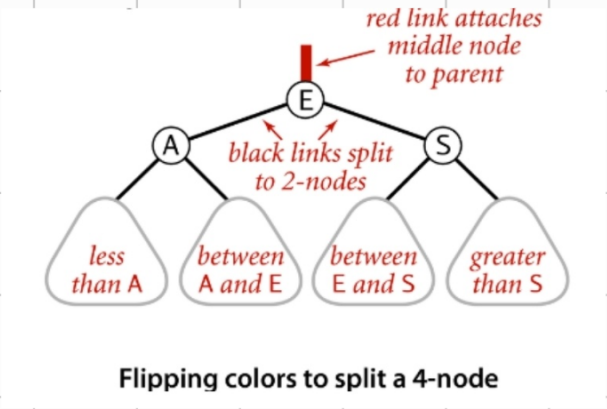
- Toda inserção de um novo nó precisa de zero, uma, ou duas rotações seguidas de uma inversão de cores

- Inversão de cores

- Implemeta a divisão de um nó triplo temporário na árvore 2-3

- Inversão de cores em um nó h: a cor rubro é retirada dos dois links que partem de h e passa o link que entra em h.
  - A cor rubra "passa para cima"

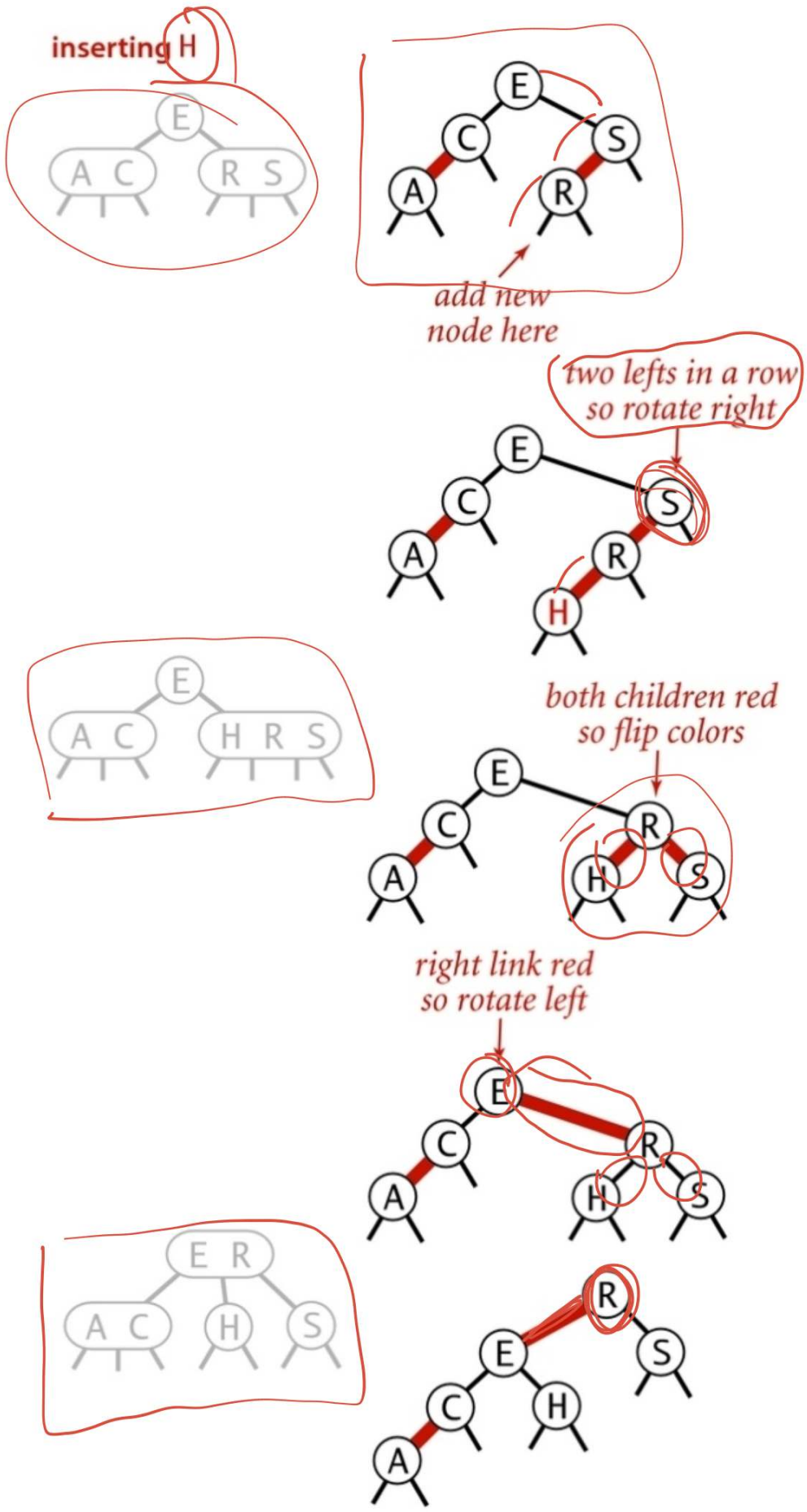




Flipping colors to split a 4-node

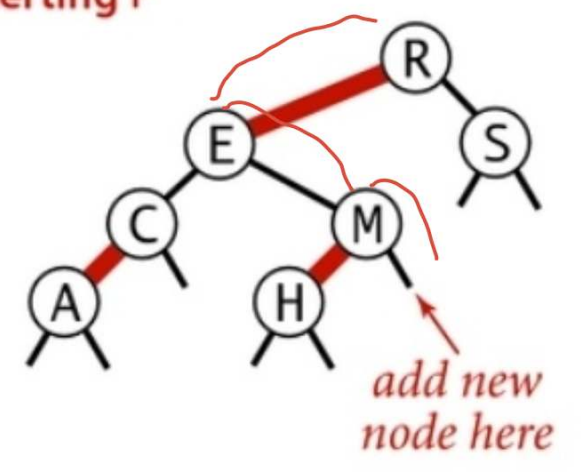
```
Void flipColors(link r)
{
    r->l->color = BLACK;
    r->r->color = BLACK;
    r->color = RED;
}
```

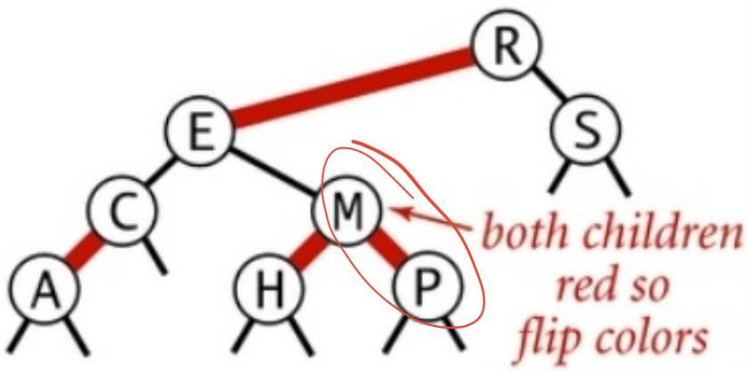
- A operação é local e preserva o balanceamento negro perfeito. A altura aumenta em 1 se e somente se h é a raiz de árvore

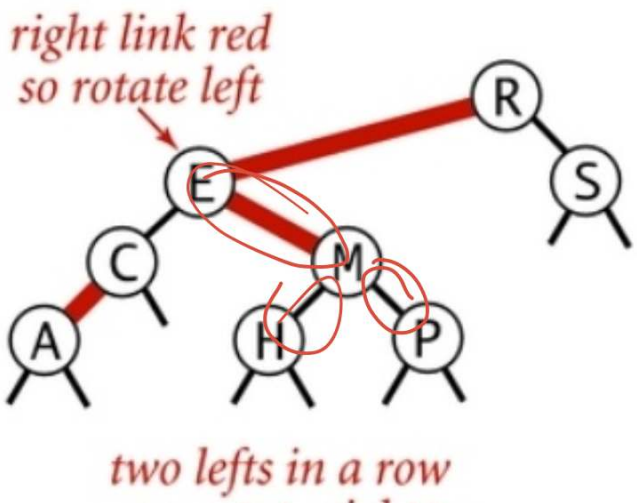


Insert into a 3-node at the bottom

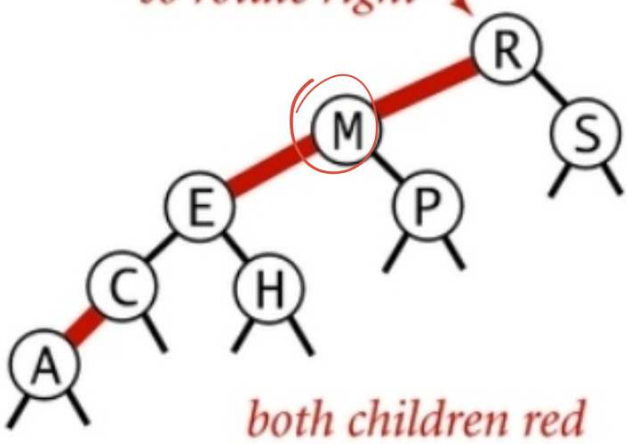
inserting P



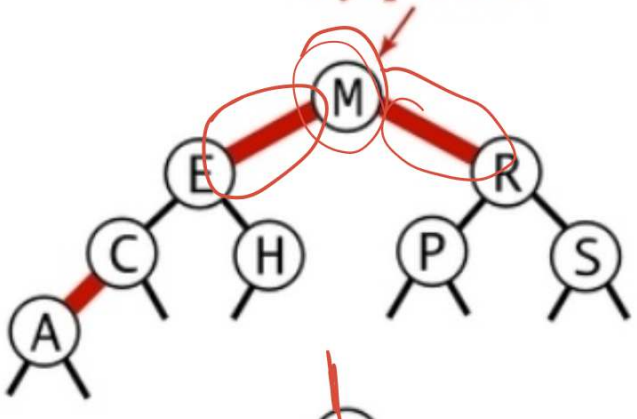


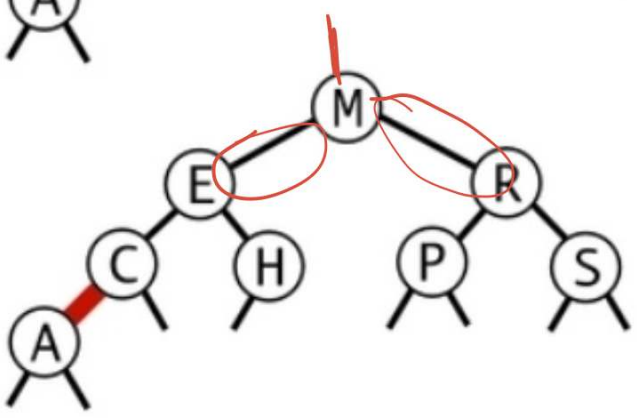


two lefts in a row
so rotate right



both children red
so flip colors





→ Como implementa a inserção?

- tem ordem das operações?

```
link InsertR(link r, Item item)
{
  if(r == z) return NEW(item, z, z, 1);
  Key k = Key(item), t = Key(r->item);
  if(less(k, t)) r->l = insertR(r->l, item);
  else           r->r = insertR(r->r, item);

  if(isRed(r->r) && !isRed(r->l)) r = rotateLeft(r)
  if(isRed(r->l) && isRed(r->left->left)) r = rotateRight(r)
  if(isRed(r->l) && isRed(r->r)) flipColors(r);
  r->N = r->l->N + r->r->N + 1;
  return r;
}
```