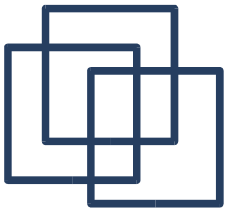
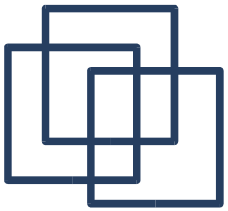
Red-Black Trees



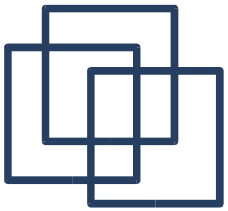
Why?

- Binary search tree perform in O(height) so:
  - if the height is large, this is bad
  - large height if unbalanced tree
  - keep the tree balanced
- Red-black trees = binary search tree with a color per node (red/black) that is approximately balanced.



Height

- **Height ≤ *2*lg*(n+1)***.
- Proof:
  - subtrees of x contain at least $2^{bh(x)}-1$ nodes (induction on the height of x).
  - *bh(root)* ≥ $h/2$ so $n \geq 2^{h/2}-1$
  - log: $h \leq 2\lg(n+1)$

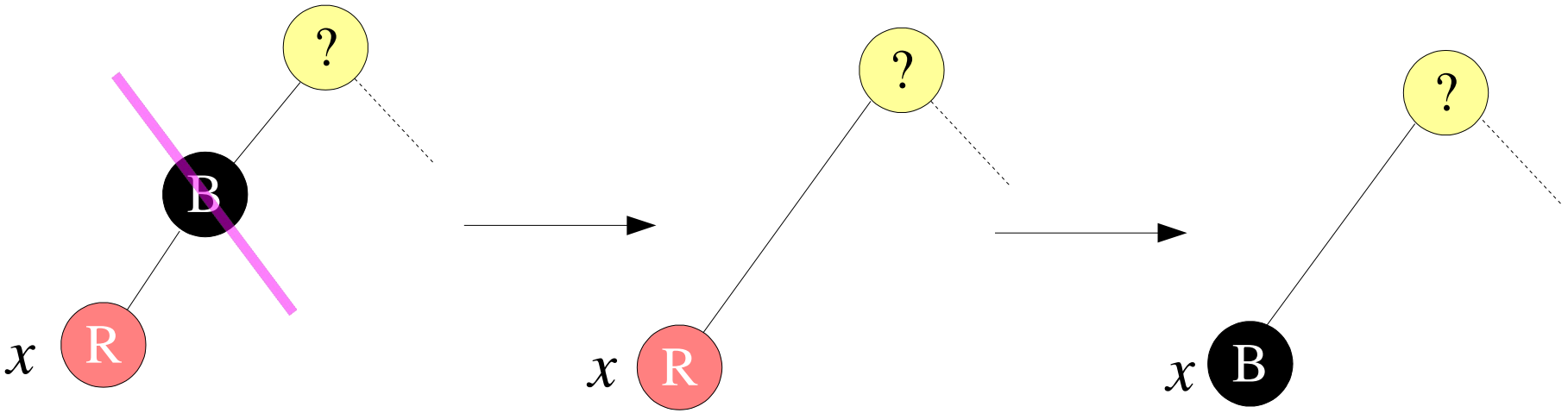


Deletion

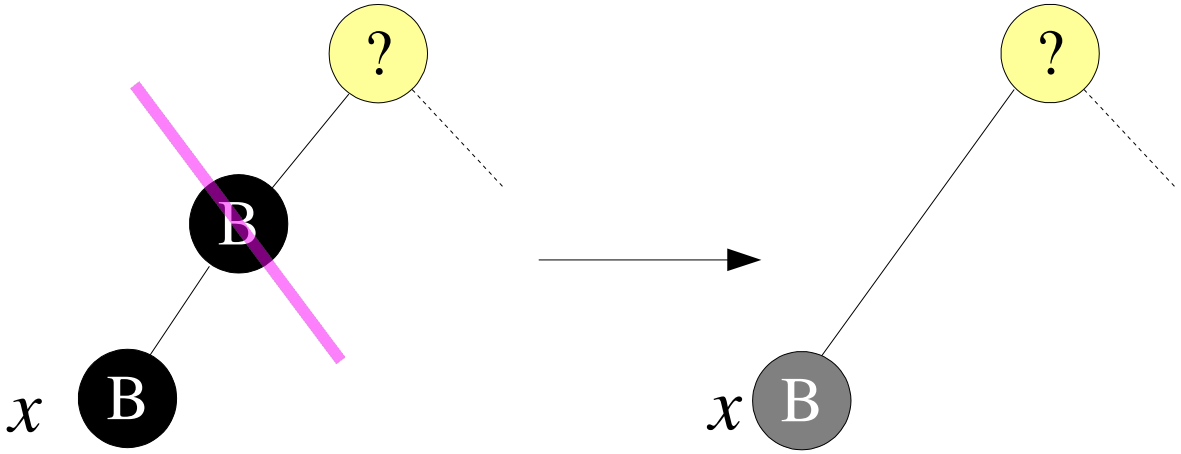
- Deletion of a red node is easy.
- Fix deletion of a black node.
- Algorithm:
  - find node to delete
  - delete it as in binary search trees
  - node to be deleted has at most one child
  - if we delete a **red node**, the tree is still a RBT
  - assume we delete a **black node**
  - x: child of the deleted node

Deletion – case trivial

- If x is red, color it black and stop,

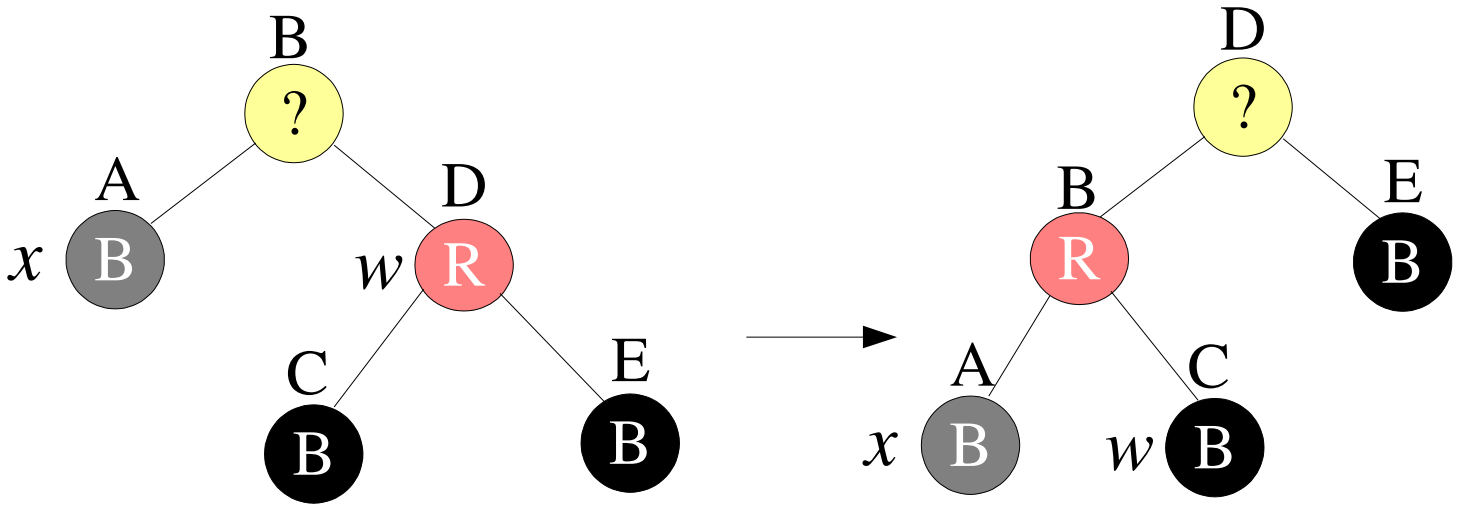


- else x is black, mark it double black.



Deletion – case 1

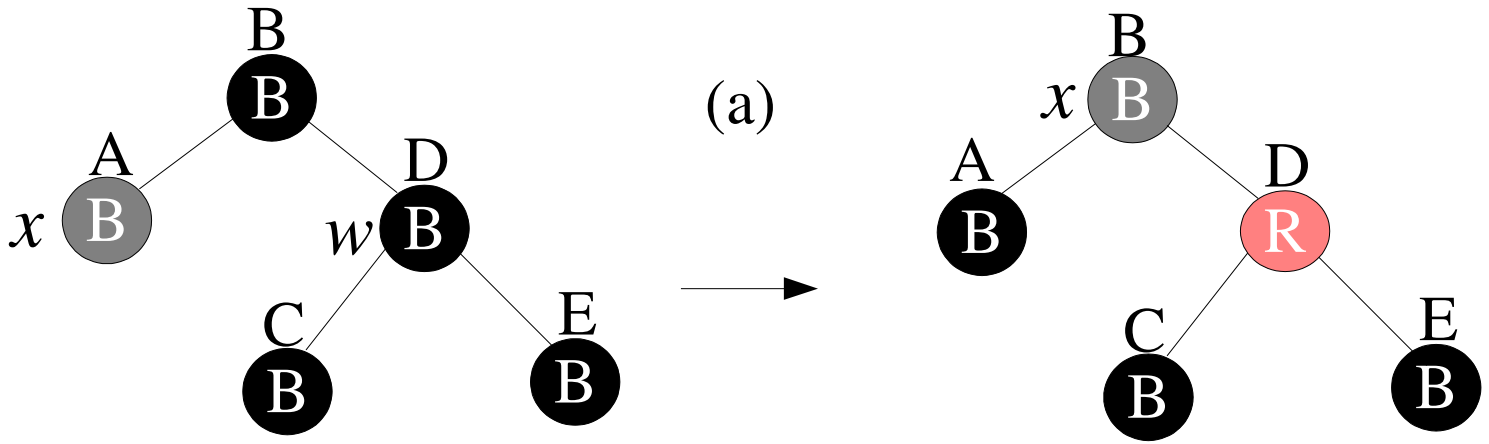
- If x's sibling is red.



- x stays at same black height.
- Case 2b, B will be colored to black.

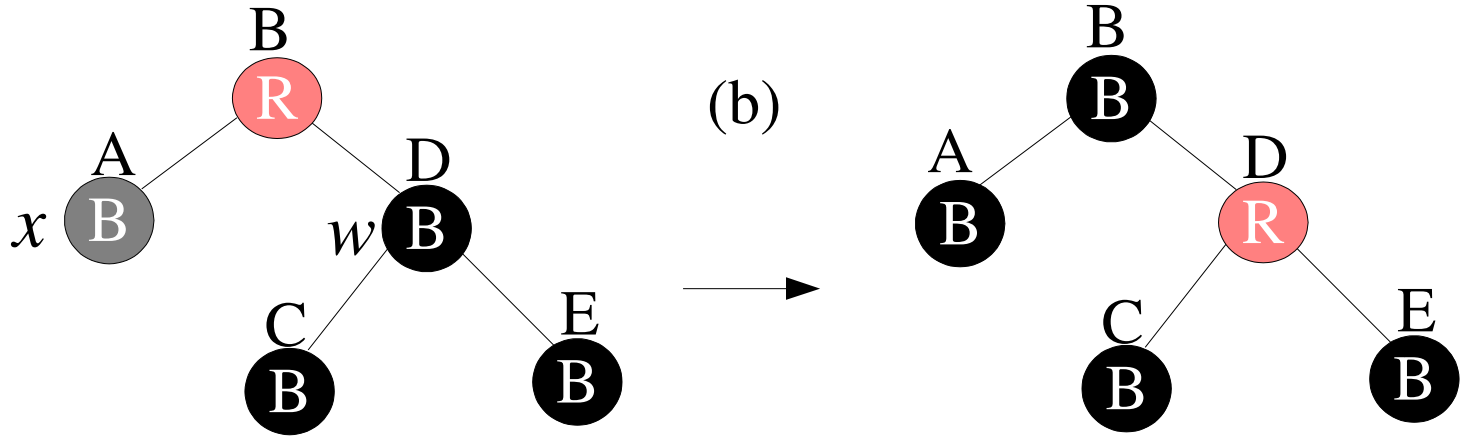
Deletion – case 2

- If x's sibling is black, x's parent is black, ...



  - Decrease x black height.

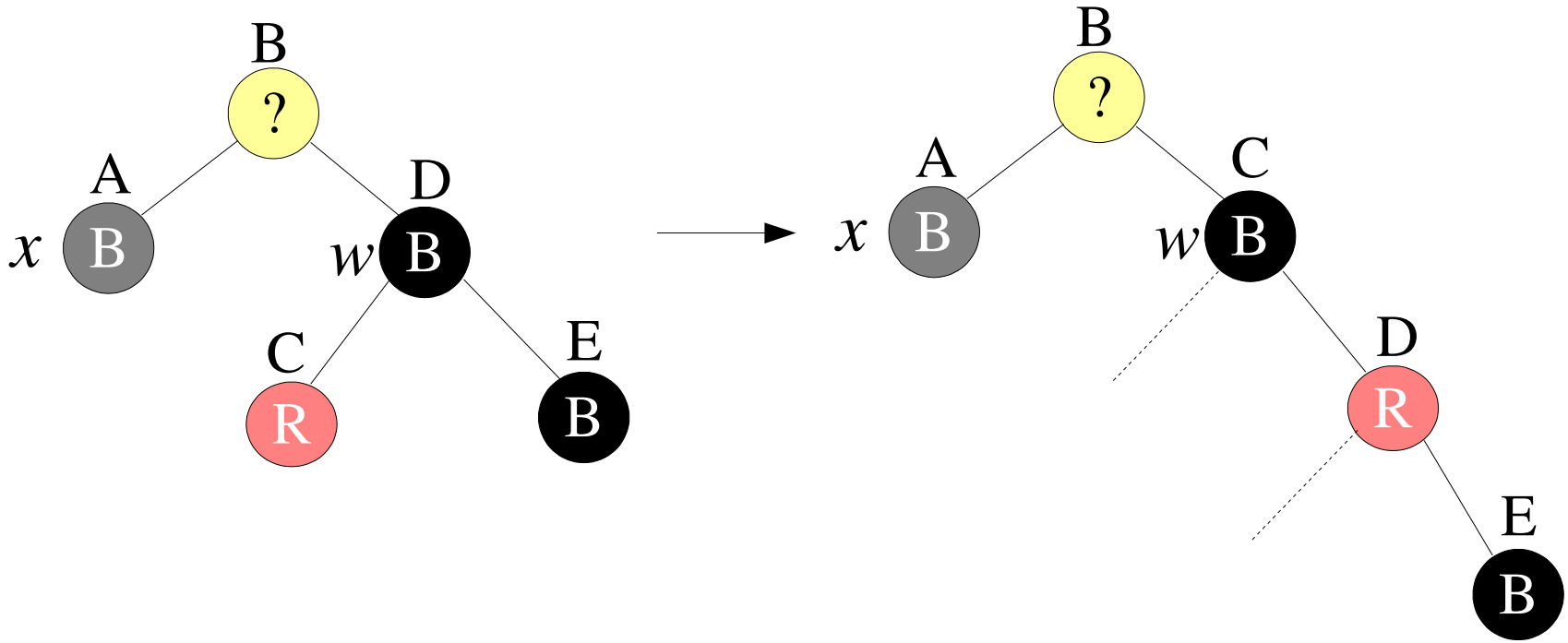
- If x's sibling is black, x's parent is red, ...



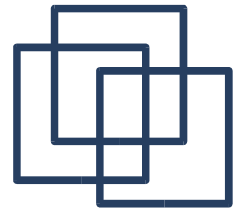
  - Stop.

Deletion – case 3

- If x's sibling is black, ...

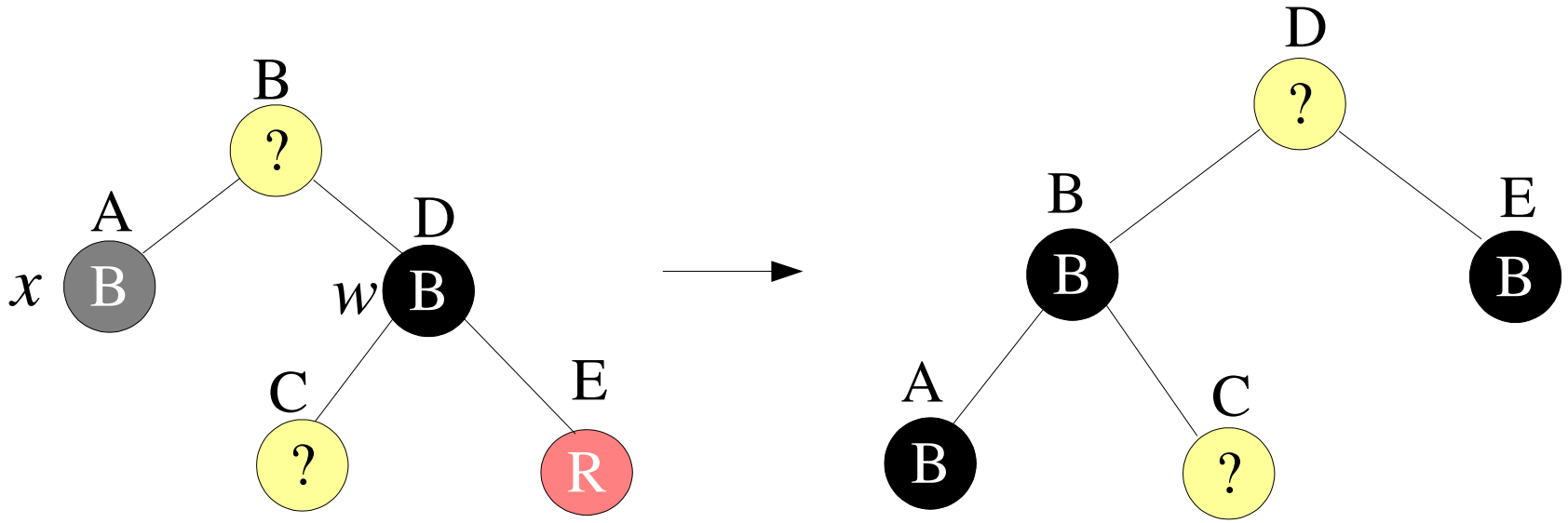


- x stays at same black height.
- Case 4.



Deletion – case 4

- If x's sibling is black, ...



- Stop.