Intelligence Of Things®

GLOBAL EDGE

•ALTRAN GROUP

RB Tree Deletion

Mariya Patel

Topics

- What are RB trees ?
- Why are they needed ?
- Rules/Properties
- Operation : Deletion
- Applications of RB tree

Definition

- Red black (RB) Tree is a **balanced** version of binary search tree.
- Every node is colored either red or black.
- In extended, NULL links are replaced by special nodes .

Motivation

- Let us take a sorted array 1 2 3 4 5 .
- Creating BST

Contd..

BST

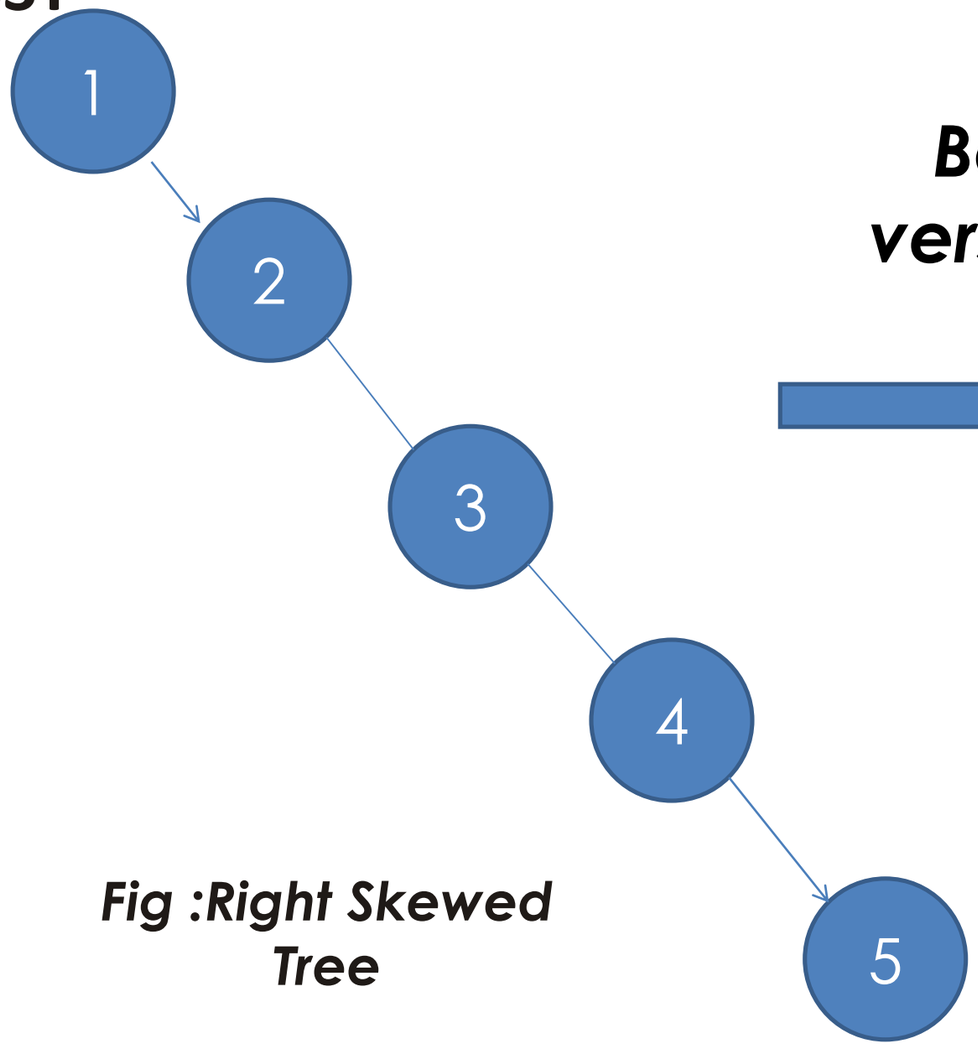
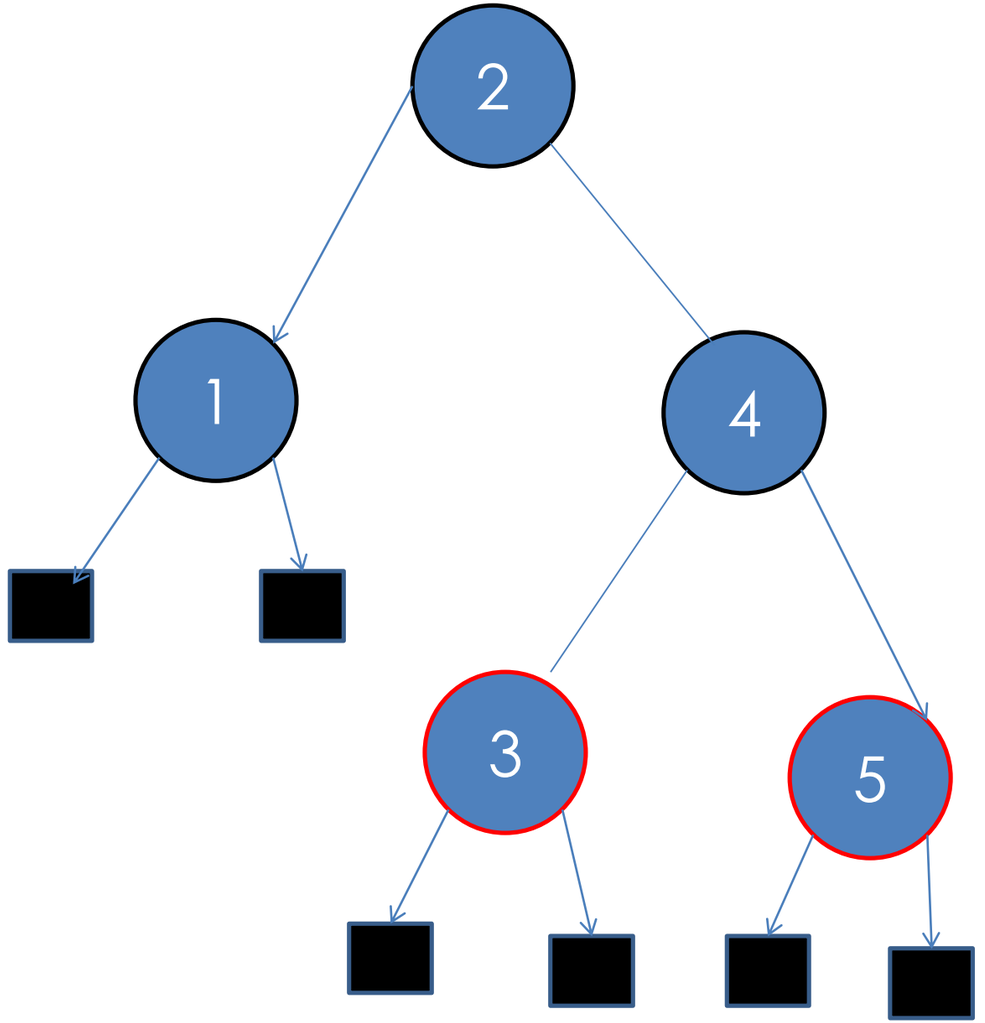


Fig :Right Skewed Tree

Balanced version of BST

RB Tree



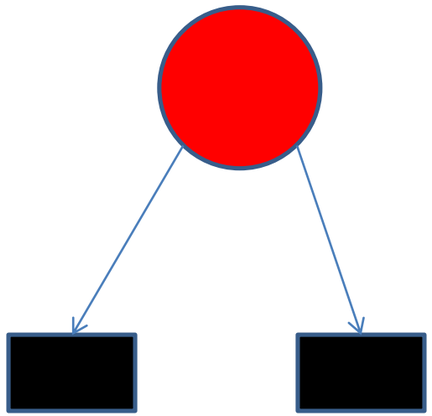
Properties :

- Root is always black
- All extended nodes are black
- A red node cannot have red children, it can have only black children; A black node can have black or red children
- For each node N, all paths from node N to external nodes contain the same number of black nodes (same black height)

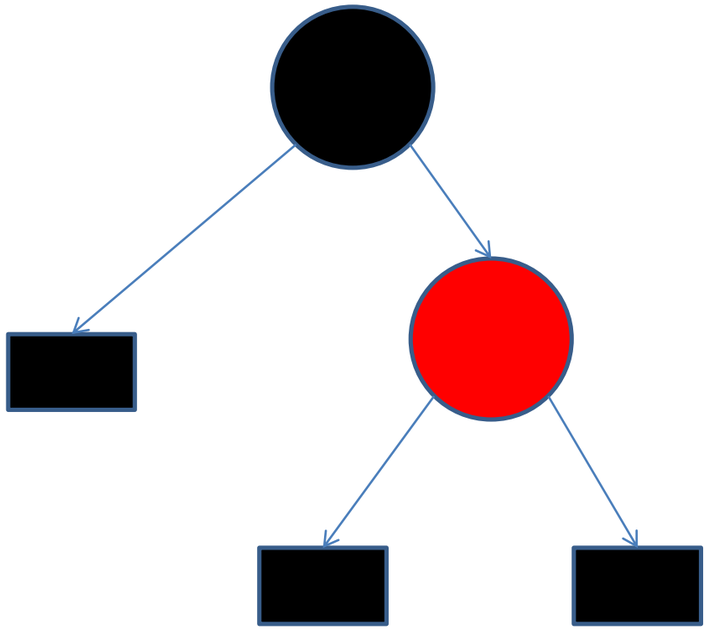
Deletion

- In BST, to delete a node with two children
- Identify inorder successor of the node
- copy the inorder successor information into the node
- delete the inorder successor.
- Inorder successor will have either no child or only right child.
- That leads us to 6 cases for deletion.

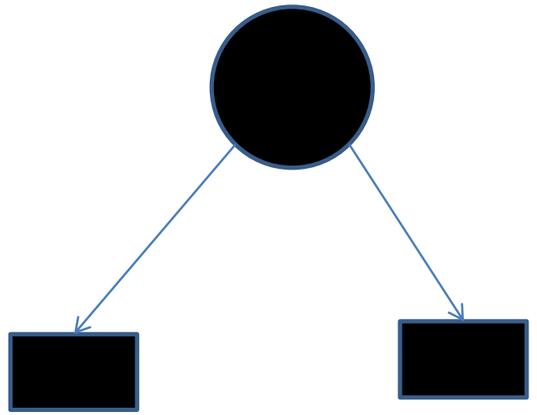
3 Valid Cases :



Case (1)



Case (2)

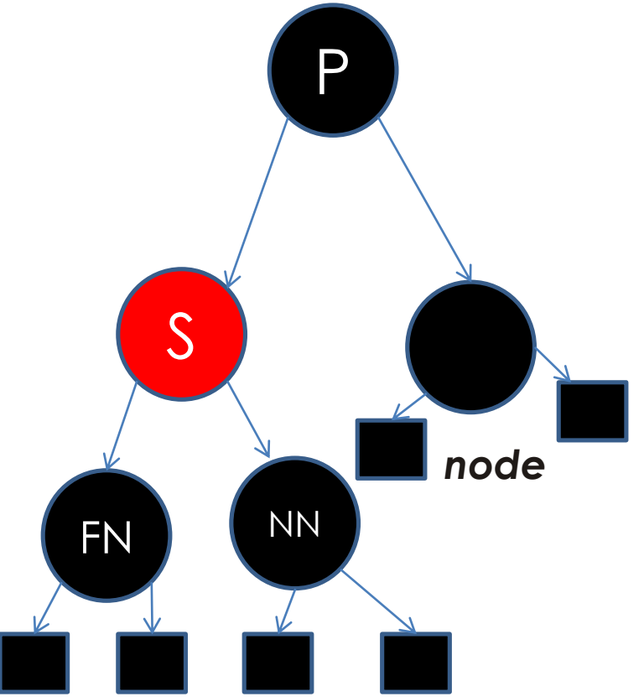


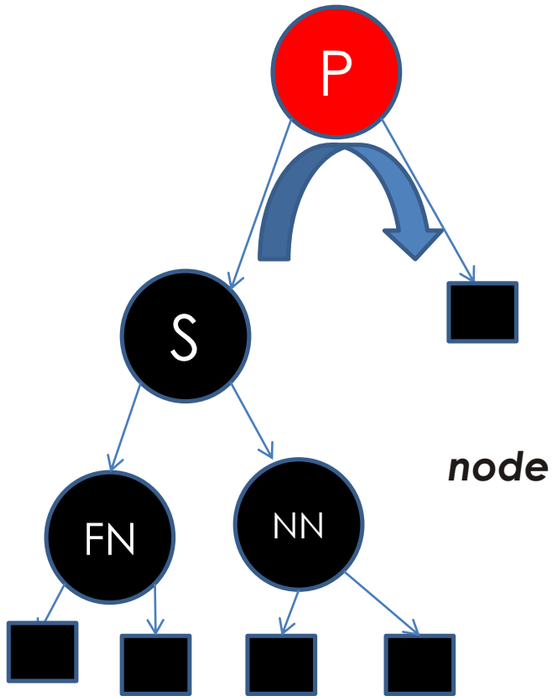
Case (3)

Case R_1 :

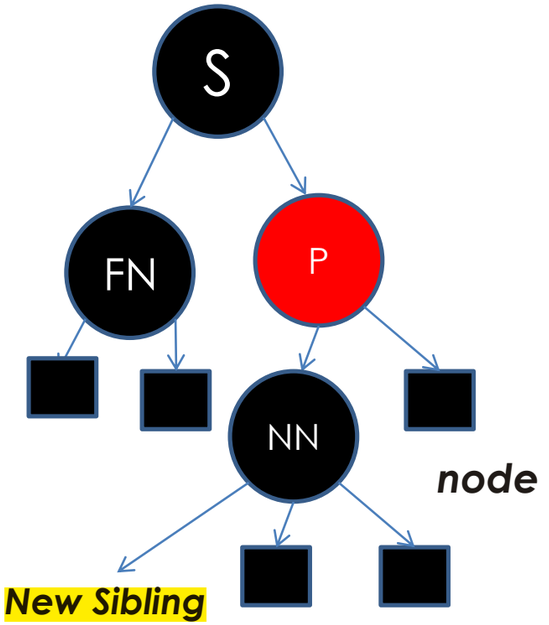
Sibling = RED , Far Nephew = BLACK , Near Nephew = BLACK , Parent = BLACK

Recolor the parent red, sibling black



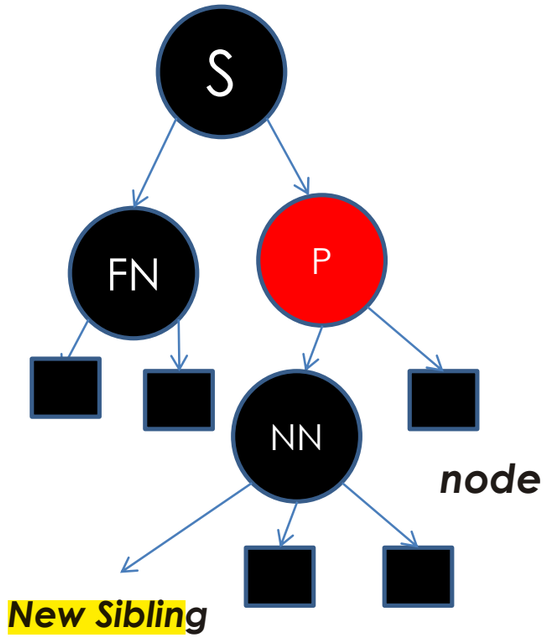


Right rotation is performed at about parent,

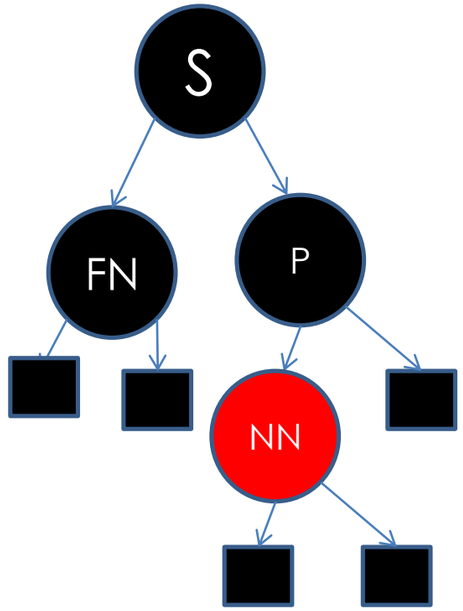


Case : R_2a

Sibling = BLACK , FarNephew = BLACK , NearNephew = BLACK , Parent = RED

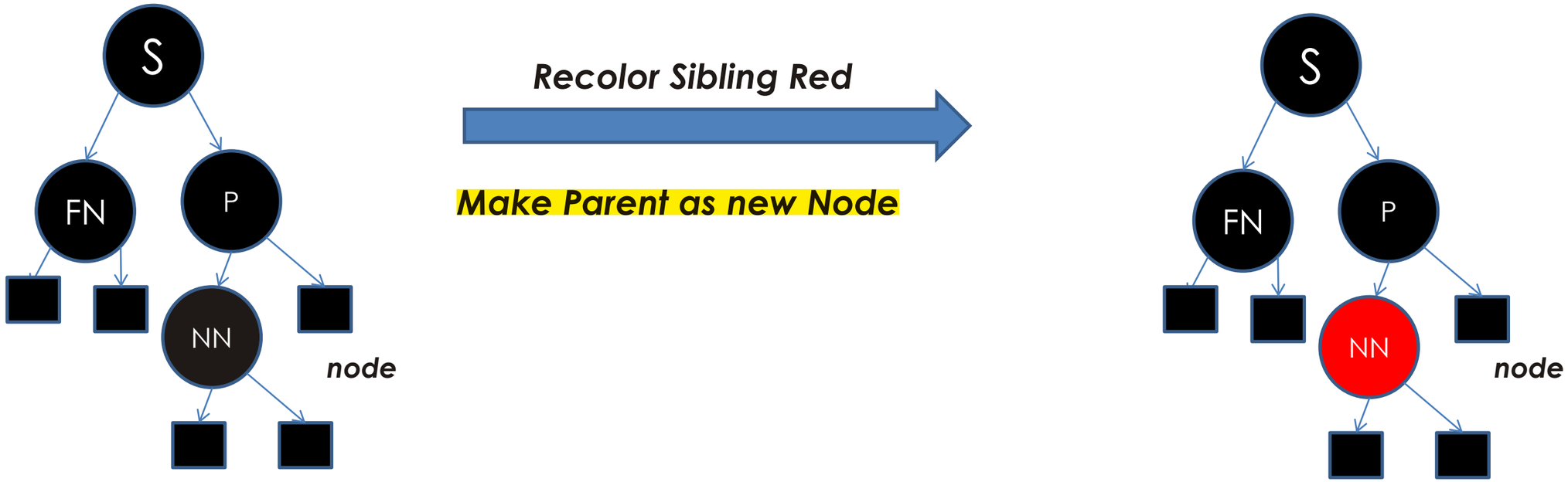


Recolor Sibling Red Parent Black. STOP



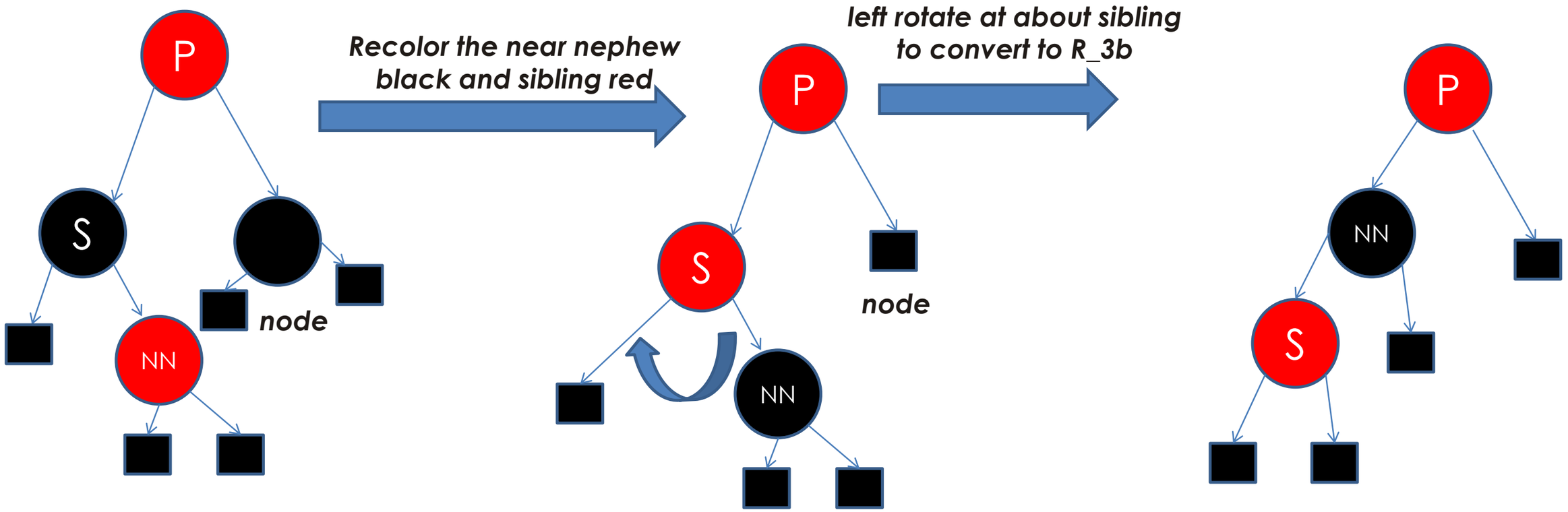
Case : R_2b

Sibling = BLACK , FarNephew = BLACK , NearNephew = BLACK , Parent = BLACK



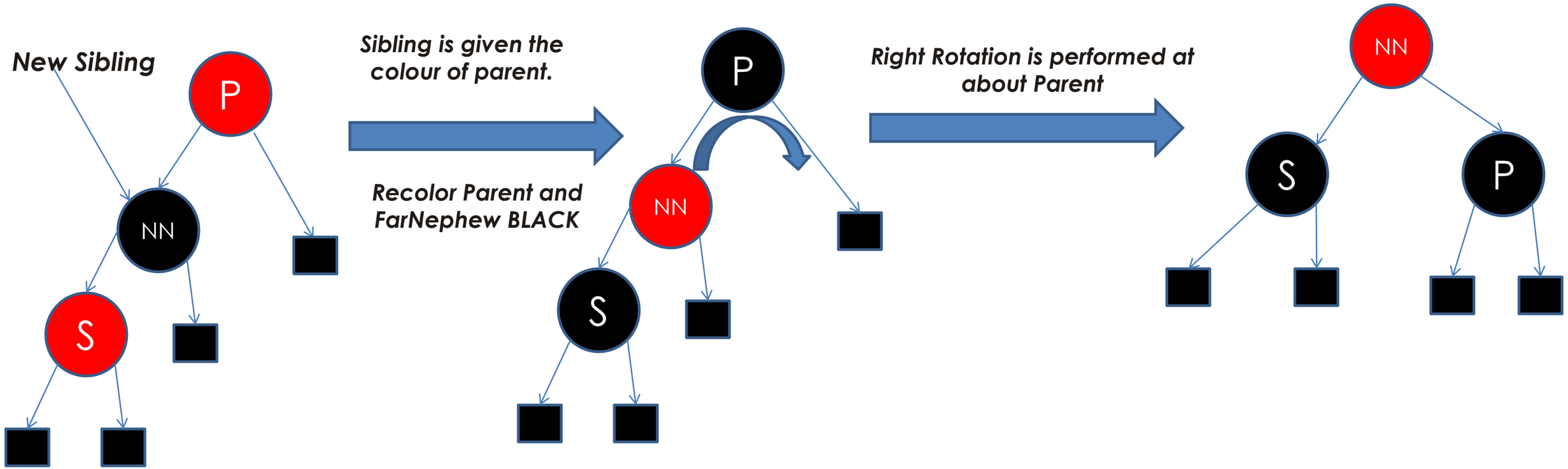
Case : R_3a

Sibling = BLACK , FarNephew = BLACK , NearNephew = RED , Parent = BLACK/RED



Case R_3b

Sibling = BLACK , FarNephew = RED , NearNephew = BLACK / RED , Parent = BLACK /RED



Deletion - (Case - Black node with no children)

Node	Sibling color	Far Nephew	Near Nephew	Parent Color	Case #	How to Balance?
Left case	Red Case L_1	Black	Black	Black	L_1	Recolor parent red,sibling black. Left rotation is performed at about parent. New sibling is black , apply case 2 or case 3 for node.
	Black Case L_2	Black	Black	Red	L_2a	Recolor the sibling and parent black stop
				Black	L_2b	Recolor sibling red, make parent new node and continue checking at parent.
	Black Case L_3	Black L_3a	Red	Black/ Red	L_3a	Recolor near nephew black and sibling red right rotate at about sibling to convert L_3b
		Red L_3b	Red	Black/ Red	L_3b	Sibling is given the color of parent . Recolor parent to red and far nephew black Left rotate at about parent.

Pseudo code :

```
RB-DELETE(T, z)
 if z->left = null or z->right = null
then y ← z
 else y ← TREE-SUCCESSOR(z)
 if y->left ≠ null
then x ← y->left
 else x ← y->right
 x->p ← y->p
 if y->p = nul
then T->root ← x
else if y = y->p->left
then y->p->left ← x
 else y->p->right ← x
if y≠ z
 then z->key ← y->key
copy y's data into z
 if y->color = BLACK
 then RB-DELETE-FIXUP(T, y)
return y
```

Applications of RB Tree :

- Widely used as system symbol tables
- Linux kernel: completely fair scheduler
- C++ STL: map, multimap, multiset
- Java: java.util.TreeMap, java.util.TreeSet
- Computational Geometry Data structures

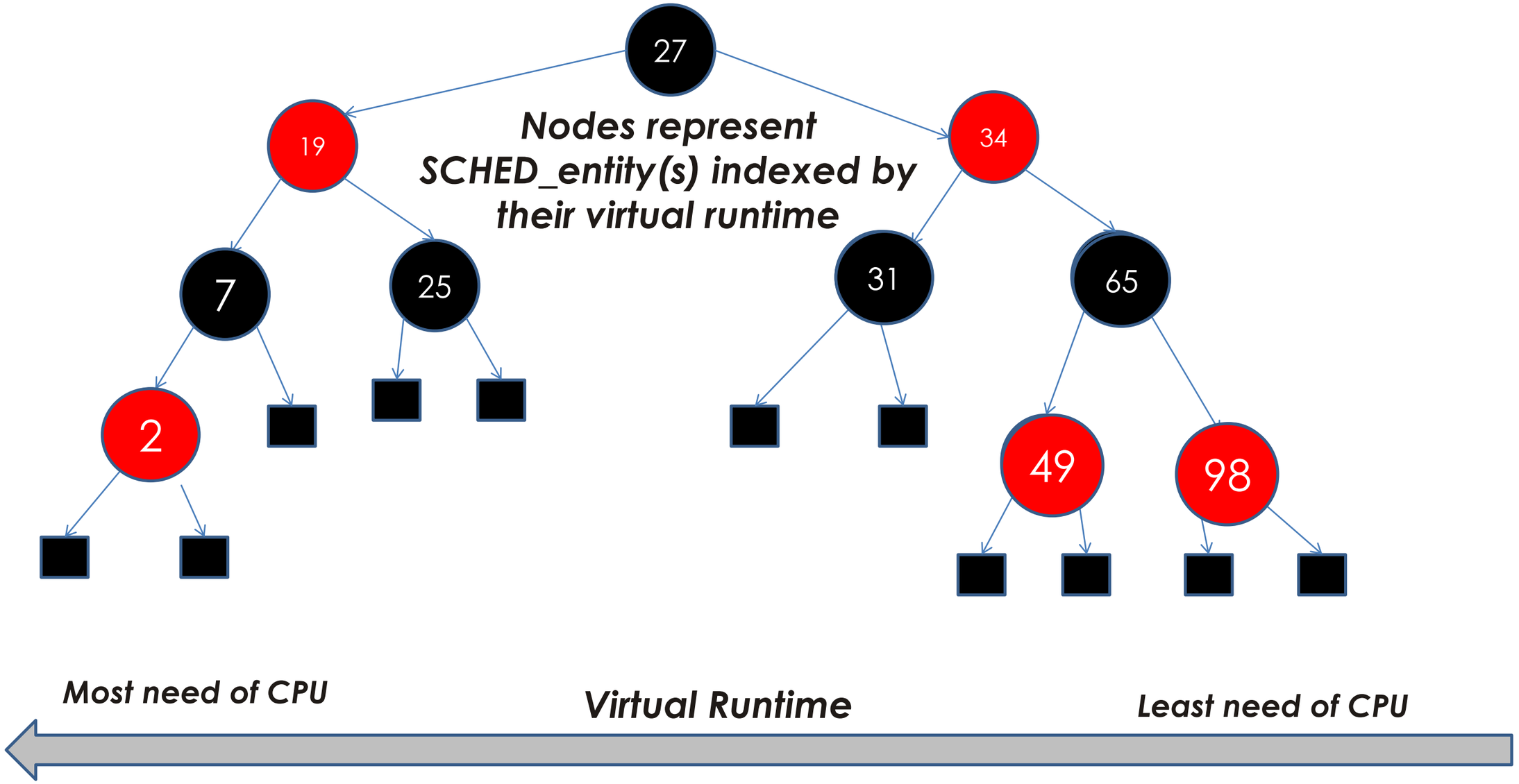
CFS Scheduler :

- The main idea of CFS is to maintain balance in providing processor time to tasks.
- When the time for tasks is out of balance , then those out-of-balance tasks should be given time to execute.
- To determine the balance , the CFS maintains the amount of time provided to given task in virtual time.
- Smaller the task's virtual time , the higher its need for the processor.
- CFS also includes sleeper fairness to ensure that tasks that are not currently runnable receive a comparable share of processor when they eventually need it.

Contd..

- But rather than maintain the tasks in a run queue , the CFS maintains a time-ordered red-black tree.

Fig : CFS Scheduler



Contd…

- With tasks (represented by sched_entity objects) stored in the time-ordered red-black tree.
- Tasks with the gravest need for the processor (lowest virtual runtime) are stored toward the left side of the tree.
- Tasks with the least need of the processor (highest virtual runtimes) are stored toward the right side of the tree.

References

- https://developer.ibm.com/technologies/linux/tutorials/l-completely-fair-scheduler/
- https://opensource.com/article/19/2/fair-scheduling-linux
- https://iq.opengenus.org/red-black-tree-deletion/

Thank You!

GLOBAL EDGE
Intelligence Of Things®
•ALTRAN GROUP

India: Bengaluru, Hyderabad | US: California

| www.globaledgesoft.com

FAIRNESS • LEARNING • RESPONSIBILITY • INNOVATION • RESPECT