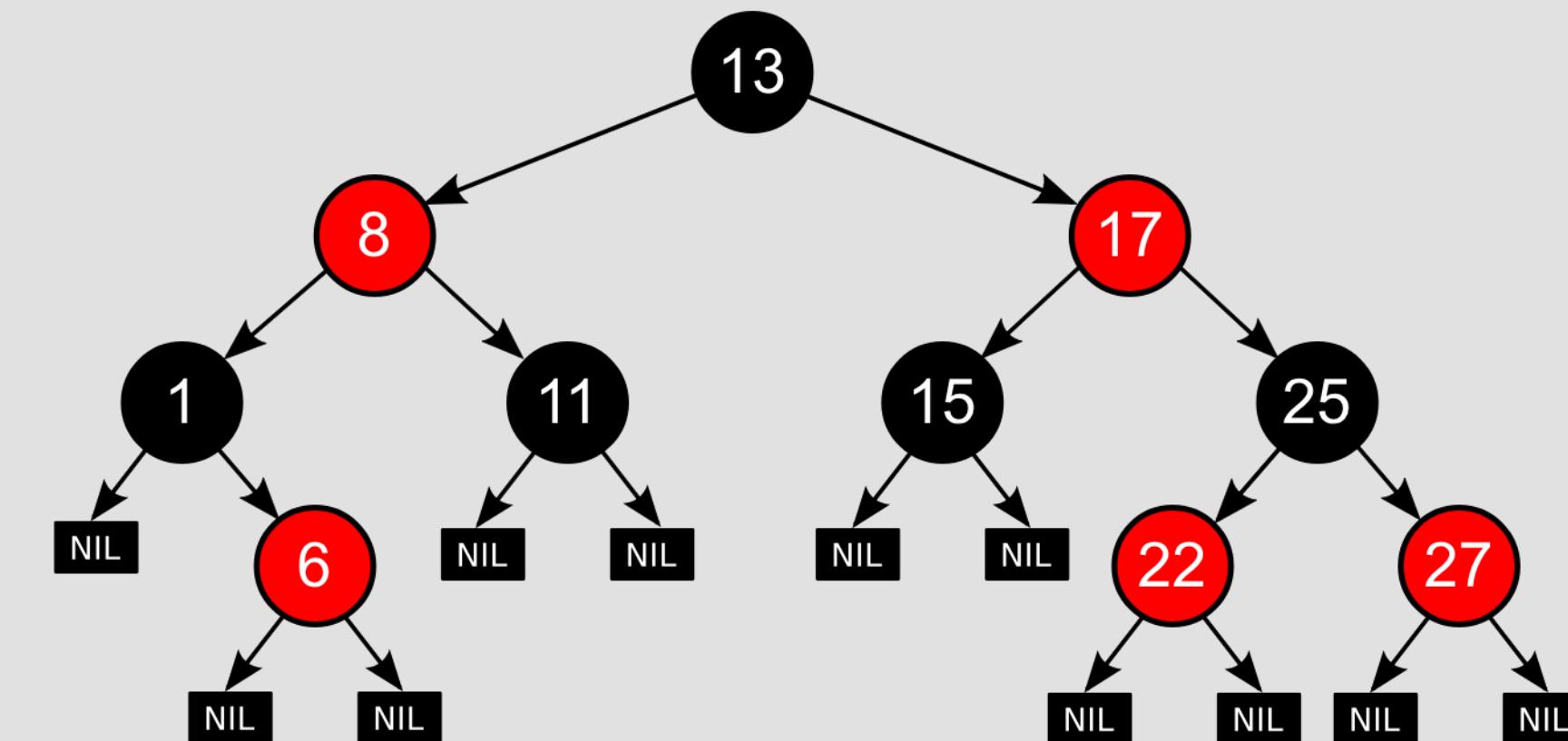


RED-BLACK TREE

Group 8:

ĐẶNG VÂN DUY - 23520361
HUỲNH DƯƠNG TIẾN - 22521465

GVHD: Nguyễn Thanh Sơn



nội dung TRÌNH BÀY

1	Giới thiệu tổng quan	4	So sánh Red-Black Tree và AVL Tree
2	Khái niệm	5	Ứng Dụng
3	Các thao tác trên cây đỏ đen	6	Kết Luận

01. Giới thiệu tổng quan

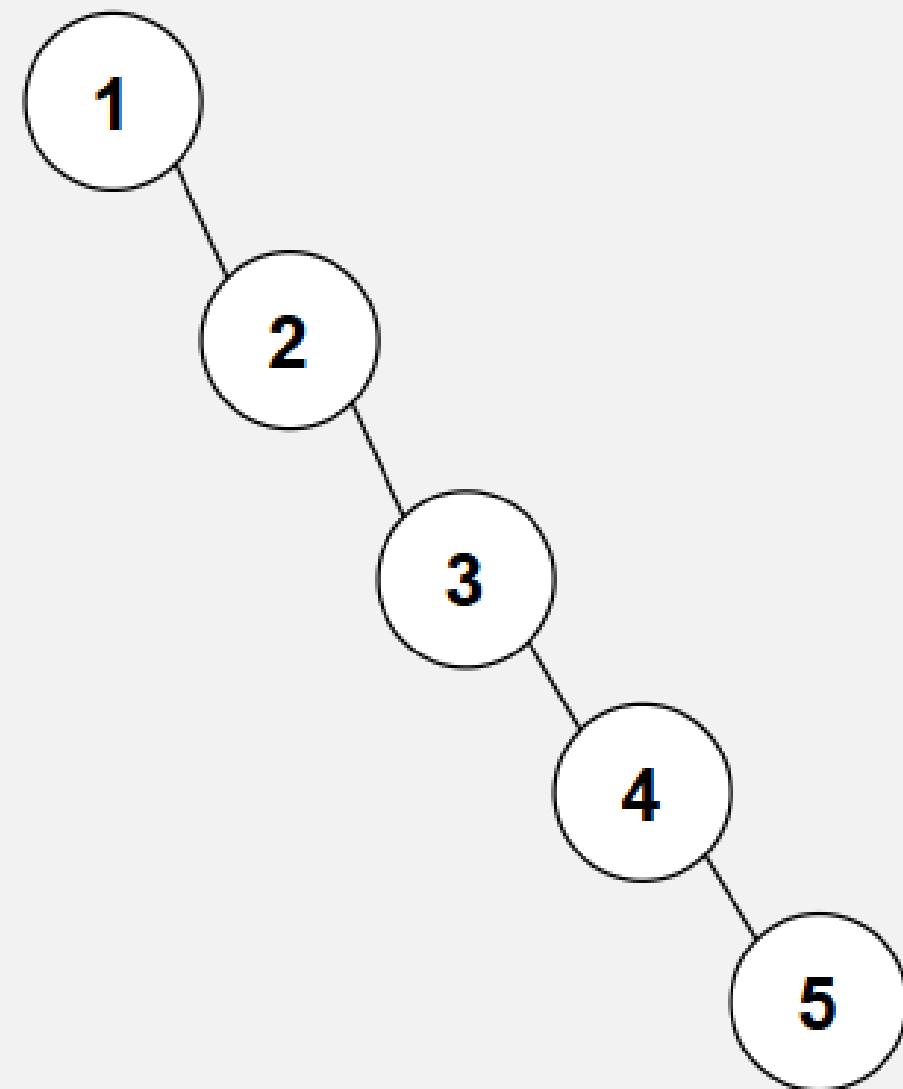


Cây đỗ đen là gì?

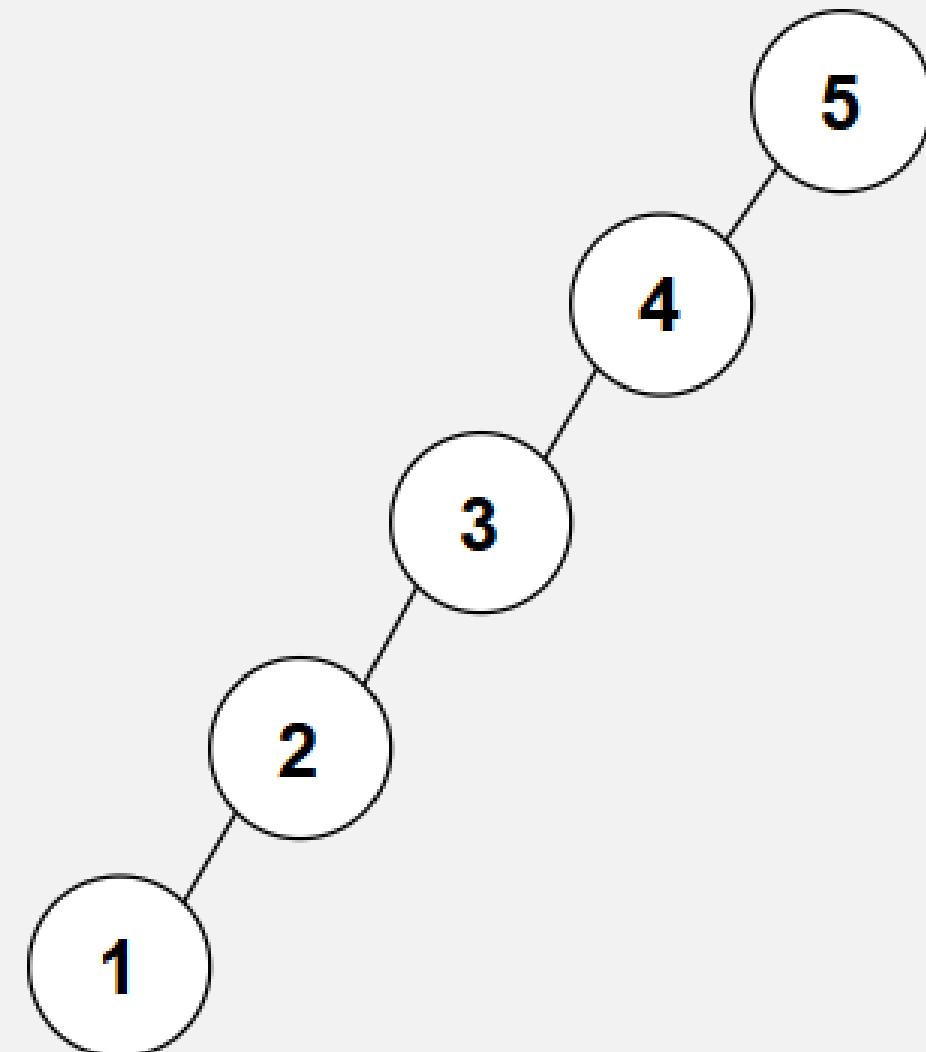
Tại sao cần cây đỗ đen ?

Cây tìm kiếm nhị phân (BST) ở dạng đã được sắp thứ tự (tăng dần hoặc giảm dần)
→ Cây suy biến thành danh sách liên kết, độ phức tạp tìm kiếm $O(n)$.

- 1,2,3,4,5

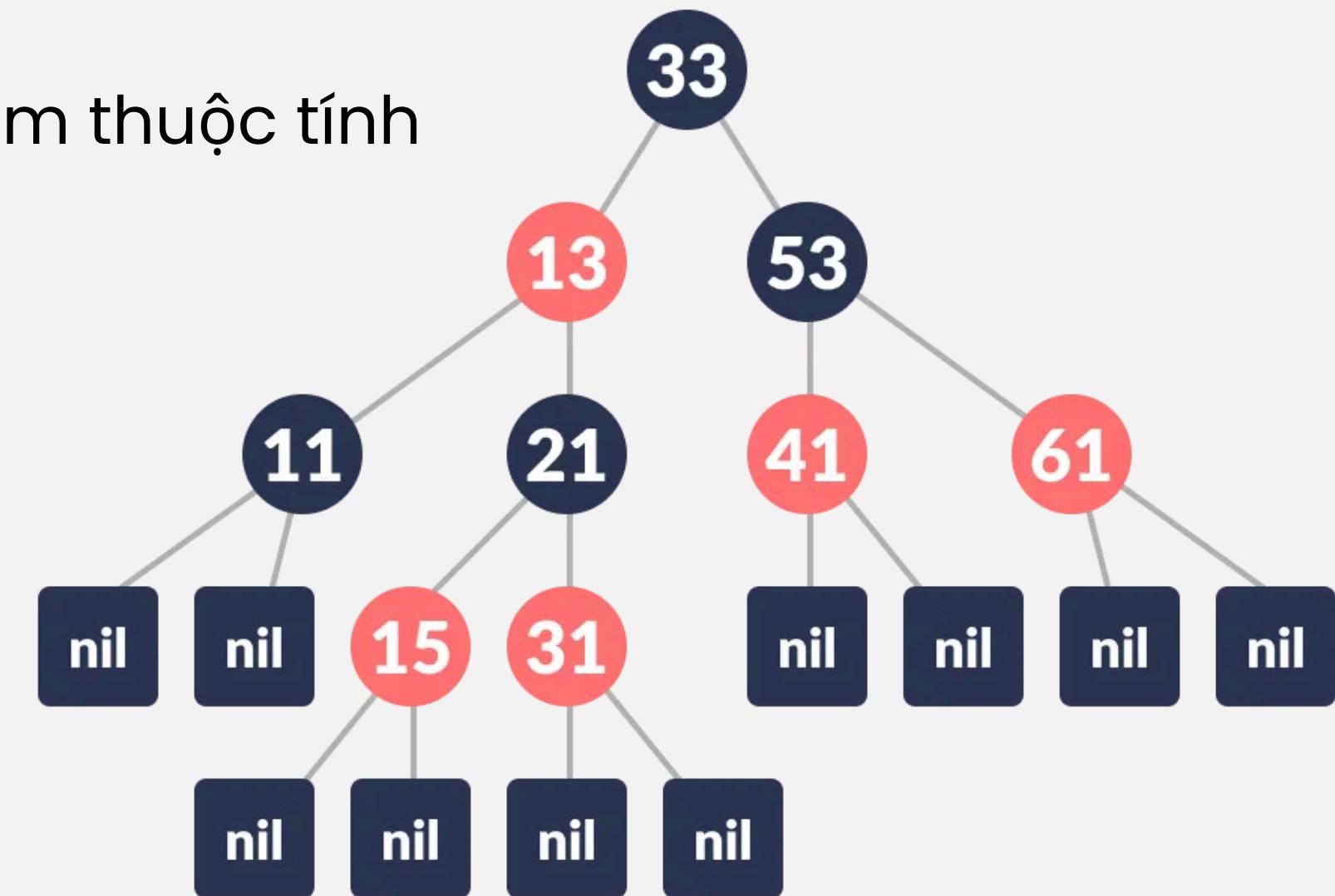


- 5,4,3,2,1



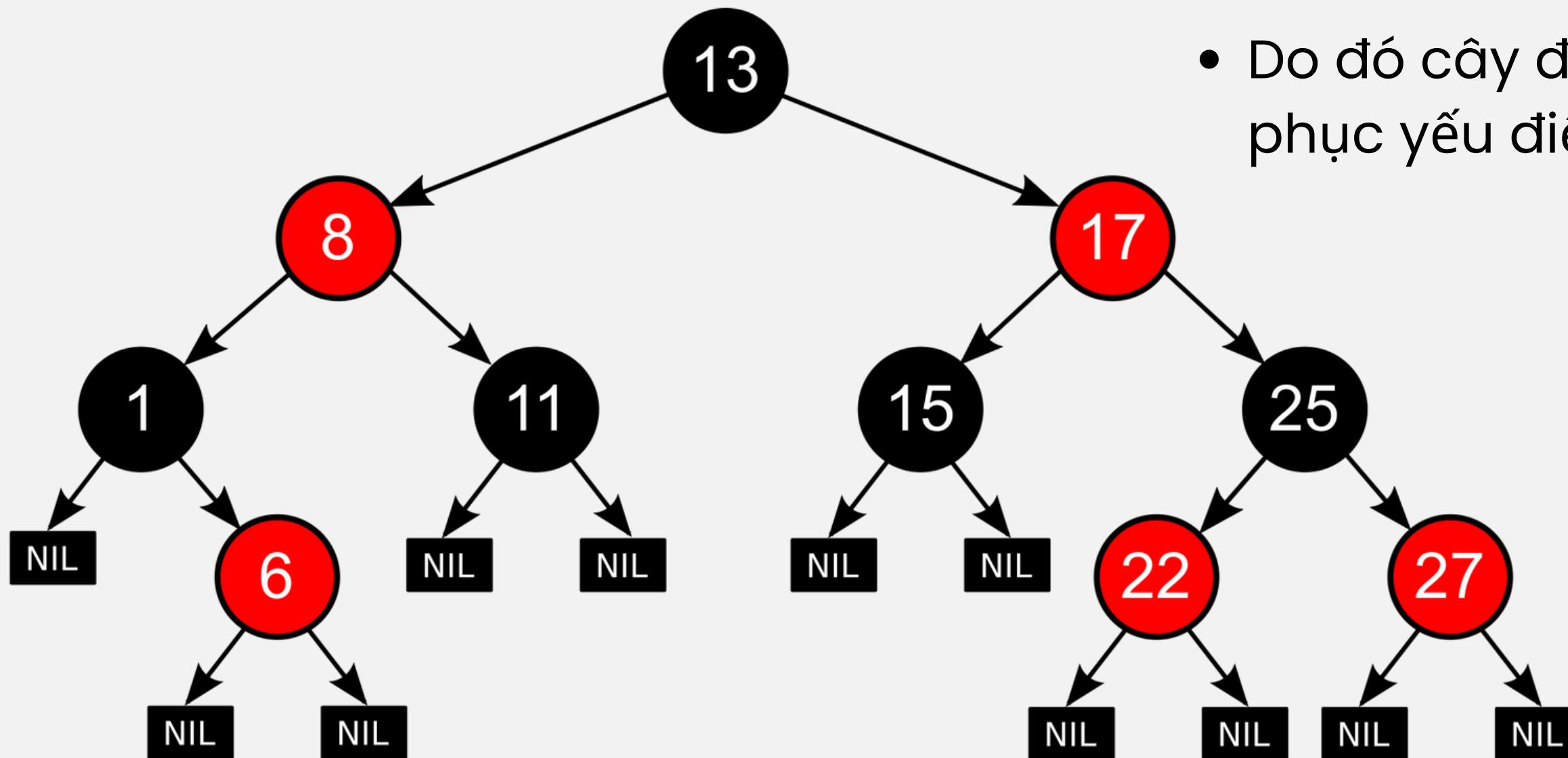
Cây đỏ đen là gì?

- Cây đỏ đen là cây nhị phân tìm kiếm tự cân bằng.
- Cấu trúc các node của cây đỏ đen có thêm thuộc tính để **quy ước màu** của node.



Tại sao cần cây đỏ đen?

- Nếu dữ liệu quá lớn thì việc cân bằng ở cây AVL sẽ mất rất nhiều công sức.
- Do đó cây đỏ đen ra đời để khắc phục yếu điểm này.



Lịch sử cây đỏ đen

- Năm 1972, Rudolf Bayer đã phát minh ra cấu trúc dữ liệu là trường hợp bậc 4 đặc biệt của B-tree. Tuy nhiên, chúng không phải là cây nhị phân tìm kiếm. Bayer gọi chúng là 'B-tree nhị phân đối xứng' trong bài báo của mình và sau đó chúng trở nên phổ biến như 2-3-4 cây.
- Trong bài báo năm 1978, Leonidas J. Guibas và Robert Sedgewick đã suy ra **cây đỏ đen** từ **cây B-tree nhị phân đối xứng**. Màu đỏ được chọn vì đây là màu đẹp nhất được tạo ra từ máy in laser màu nơi ông làm việc.

Lịch sử hình thành

Sự ảnh hưởng từ B-Tree

- Cây đỏ-đen **không phải ngẫu nhiên** mà có các quy tắc về màu sắc và cân bằng.
- Nó thực chất là **phiên bản nhị phân hóa** của B-tree (cụ thể là 2-3-4 Tree).
- Cách ánh xạ từ RBT sang B-tree:
 - Mỗi node đen trong RBT tương ứng với một phần tử trong node B-tree.
 - Node đỏ được coi như liên kết với node cha để tạo thành node B-tree lớn hơn (node 3 hoặc node 4).

Lịch sử hình thành

Sự ảnh hưởng từ B-Tree

- Các quy tắc Đỏ-Đen tương ứng với B-tree

Mỗi tính chất của RBT đều có ý nghĩa khi chuyển sang B-tree:

Tính chất RBT	Ý nghĩa trong B-tree
Gốc phải là đen	Nút gốc B-tree không thể bị "ép" thành nút con.
Không có 2 đỏ liên tiếp	Tránh tạo nút 5 (vượt quá bậc 4).
Mọi đường đi có cùng số đen	Đảm bảo cân bằng chiều cao trong B-tree.
Nút lá (null) là đen	Tương ứng với lá B-tree.

Lịch sử hình thành

Sự ảnh hưởng từ B-Tree

- Thao tác chèn/xóa và cân bằng
 - Khi thêm/xóa trong RBT, các phép quay (rotate) và đổi màu (recoloring) thực chất là mô phỏng quá trình **split/merge** trong B-tree:

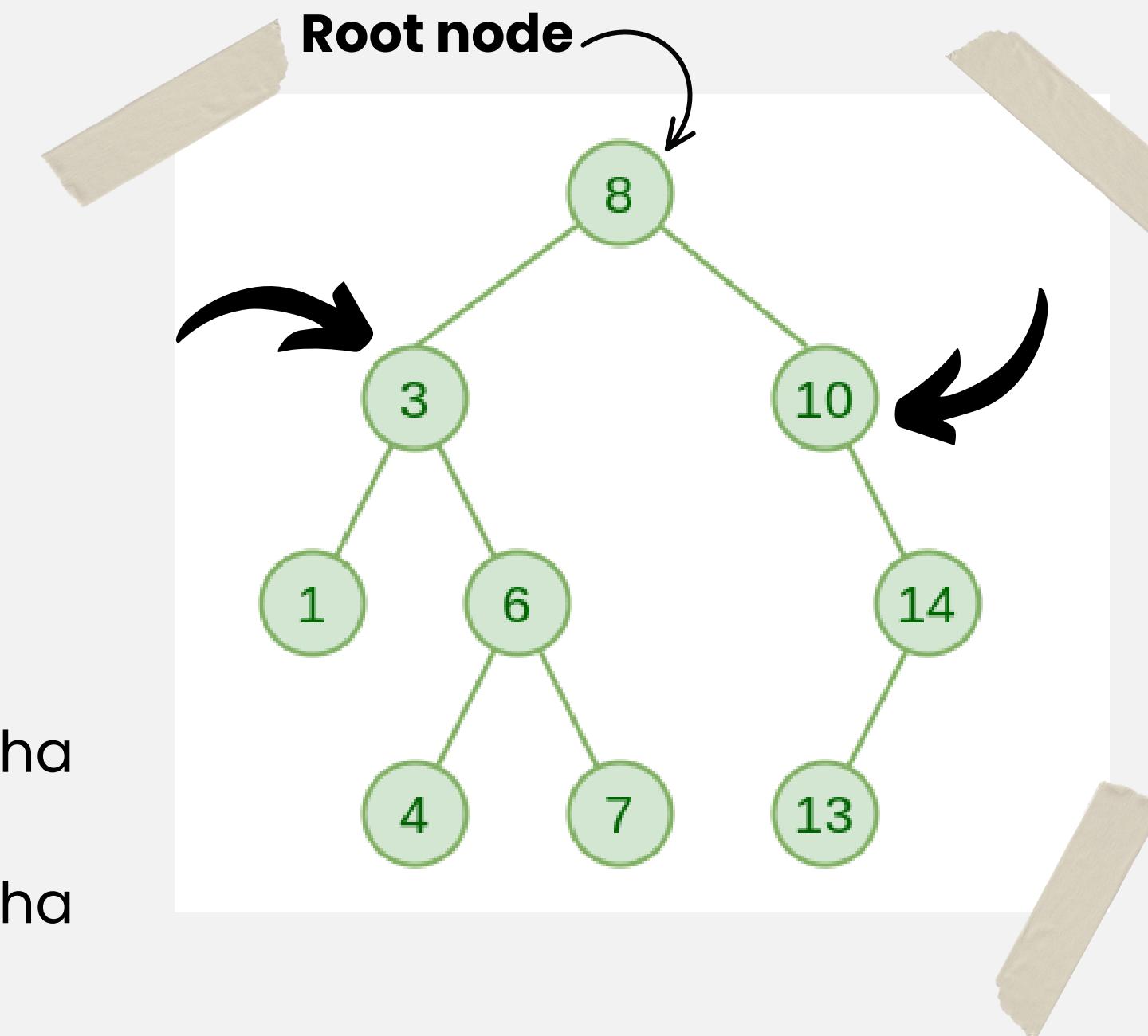
02. Khái niệm

Binary Search Tree

Dữ liệu được đưa vào các node (node ở trên cùng được gọi là Root node/ nút gốc)

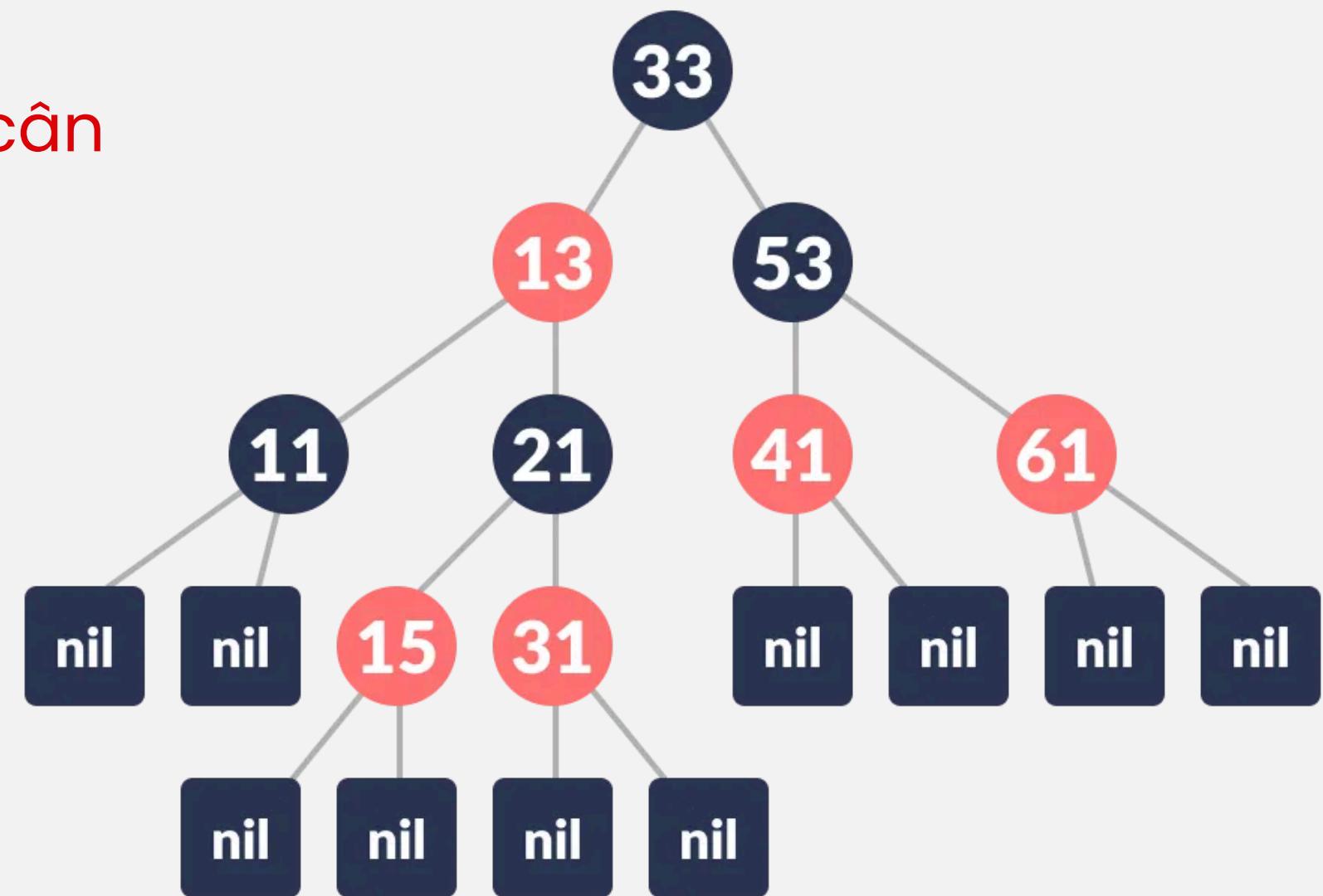
Với mỗi node đều thỏa mãn tính chất như sau:

- Node con bên trái sẽ có giá trị nhỏ hơn node cha
- Node con bên phải sẽ có giá trị lớn hơn node cha



02. Khái niệm

- Cây đỏ đen là **một cây nhị phân tìm kiếm tự cân bằng** (Self-Balancing Binary Search Tree).
=> Có các tính chất của cây nhị phân.
- Trong đó mỗi node có thêm thuộc tính màu: **đỏ hoặc đen**.
- Các node lá được gán khóa **null**.



Các quy tắc của cây đỏ đen

1. Mỗi node có màu đỏ hoặc đen.
2. Node gốc luôn là màu đen.
3. Tất cả các lá (Null hay NIL nodes) đều là màu đen.
4. Nếu một node là đỏ thì cả hai con của nó phải là đen.
5. Từ một node bất kỳ đến tất cả các lá của nó, số lượng node đen phải bằng nhau. *

Mở rộng

- Ban đầu để cây đỏ đen có thể giữ được cân bằng, người ta đặt ra các tính chất logic.

- Trong đó có tính chất số 5

"Từ một node bất kỳ đến tất cả các lá của nó, số lượng node đen trên các đường đi phải bằng nhau."

- Để tiện phân tích – người ta mới đặt tên cho "số lượng node đen trên một đường từ node đến lá" là "chiều cao đen (black-height)".

Chiều cao đen (Black-Height)

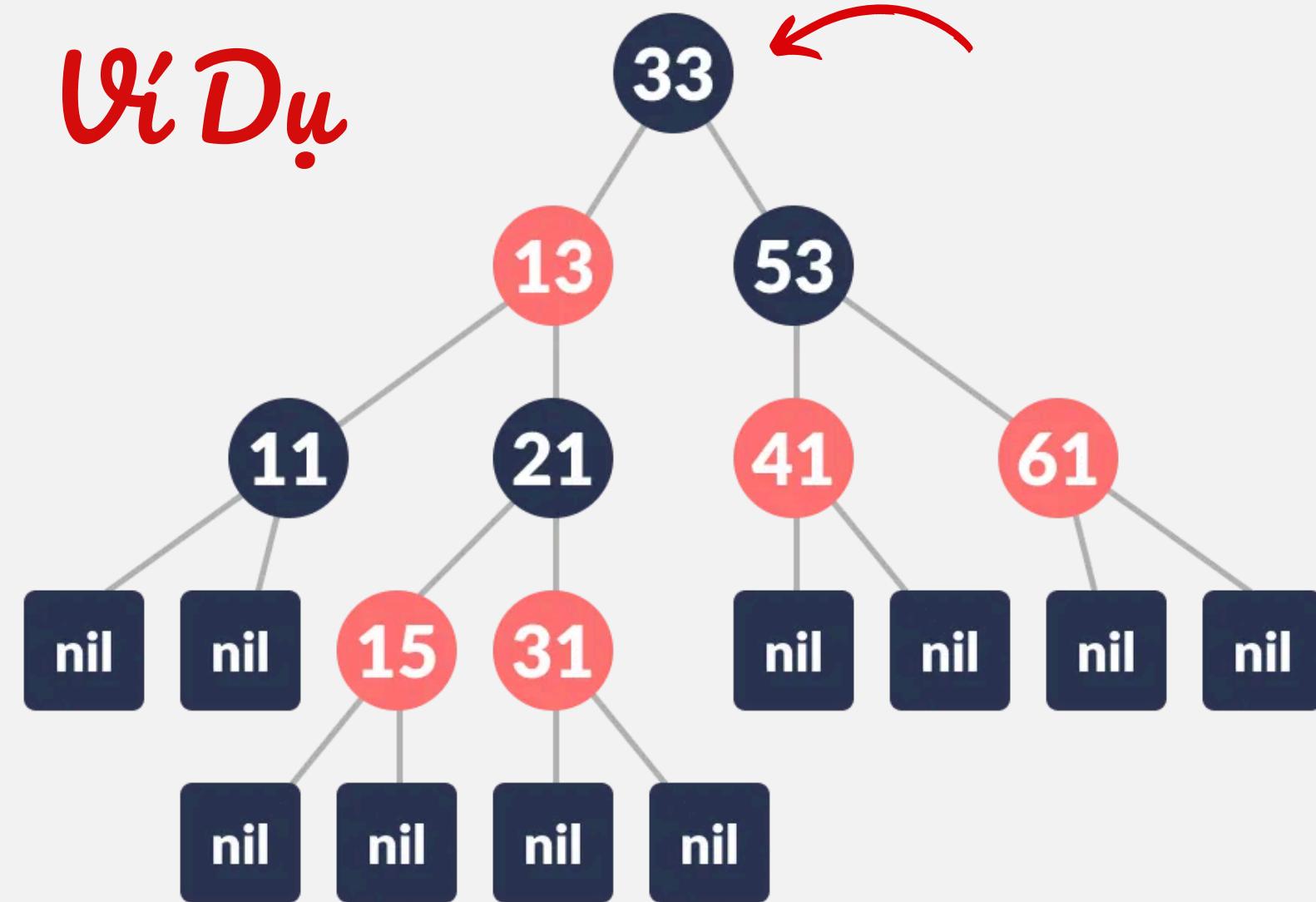
- Định nghĩa:

Chiều cao đen của một node là số lượng node màu đen trên đường đi đơn từ node đó đến bất kỳ lá nào (NIL), **không tính chính node đó**, nhưng **có tính các node NIL đen ở cuối**.

- Ký hiệu thường dùng:

$bh(x)$ là chiều cao đen của node x .

Ví Dụ



Bắt buộc phải bằng nhau trên mọi nhánh con của một node.

→ Đây là ràng buộc giúp đảm bảo cây không thể lệch nghiêm trọng về một bên.



Đường đi thứ nhất
là:

13-21-31-NIL

2 nút
đen

Đường đi thứ hai
là:

13-11-NIL

2 nút
đen

Đường đi thứ ba
là:

53-61-
NIL

2 nút

Tính chất về chiều cao cây:)

- chiều cao tổng thể của cây không vượt quá 2 lần chiều cao đen.

→ Tính chất này đảm bảo cây đỏ-đen **luôn cân bằng**, giúp các thao tác (tìm kiếm, chèn, xóa) có độ phức tạp **$O(\log n)$** trong trường hợp **tệ nhât**.

- Tại sao $\text{chiều cao cây} \leq 2 \times \text{chiều cao đen}$?

Theo tính chất cây đỏ-đen, không có 2 node đỏ liên tiếp trên một đường đi.

Trường hợp tệ nhât: Đường đi xen kẽ node đỏ và đen
(ví dụ: Đỏ → Đen → Đỏ → Đen).

- Số node đỏ \leq số node đen \rightarrow Tổng node $\leq 2 \times$ số node đen.
- Chiều cao cây (số node tối đa trên đường đi) $\leq 2 \times$ chiều cao đen.

Tính chất về chiều cao cây:)

Tại sao " $\text{height} \leq 2h$ " \rightarrow Cây cân bằng? (h là chiều cao đen)

- Giả sử cây có n node và có chiều cao đen h
- Thực tế, cây có thể có thêm node đỏ \rightarrow số node $n \geq h$.
- Do **không** có 2 node đỏ liên tiếp, chiều cao cây **không** vượt quá $2h$.
- Từ $\text{height} \leq 2h$ và $h \leq n \rightarrow \text{height} \leq 2n$.

Tính chất về chiều cao cây:)

Tại sao "height $\leq 2h$ " \rightarrow Cây cân bằng? (h là chiều cao đen)

Nhưng thực tế, h liên quan đến log n vì:

Cây có **h node đen** sẽ có ít nhất **$2^h - 1$ node** (nếu xem node đen tạo thành cây nhị phân hoàn chỉnh).

$$\Rightarrow h \leq \log_2(n + 1).$$

Kết hợp với height $\leq 2h \rightarrow$ height $\leq 2 \log_2(n + 1) = O(\log n)$.

 Điều này chứng tỏ cây không bao giờ bị lệch quá $O(\log n)$, dù trong trường hợp xấu nhất.

Hệ quả

- Trong cây đỏ-đen, độ dài đường đi dài nhất từ gốc đến lá (đường đi tệ nhất) **không dài quá gấp đôi** độ dài đường đi ngắn nhất (đường đi tốt nhất).

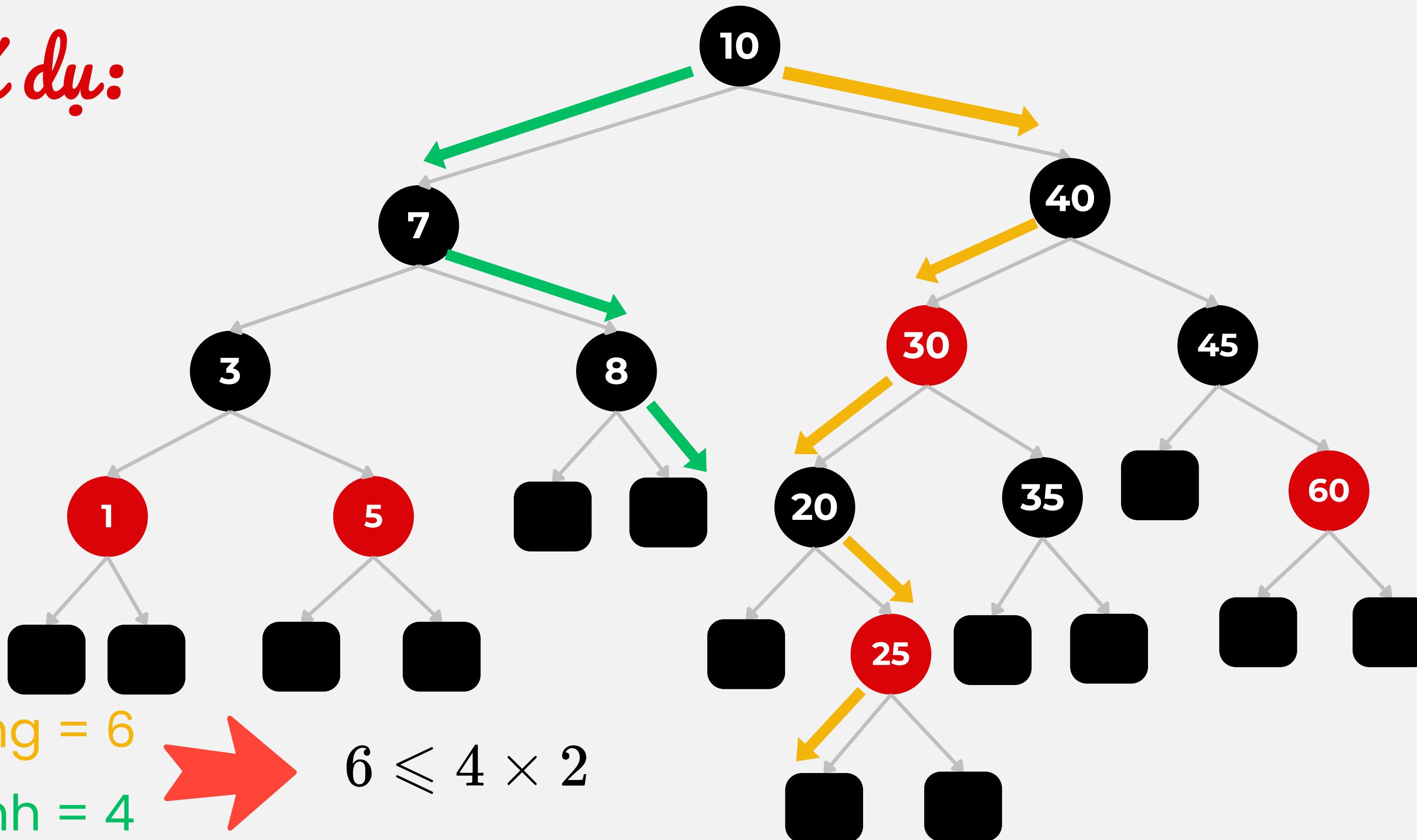
Đường đi "tệ nhất" xảy ra khi các nút đỏ và đen xen kẽ.

Đường đi "tốt nhất" chỉ chứa các nút đen.

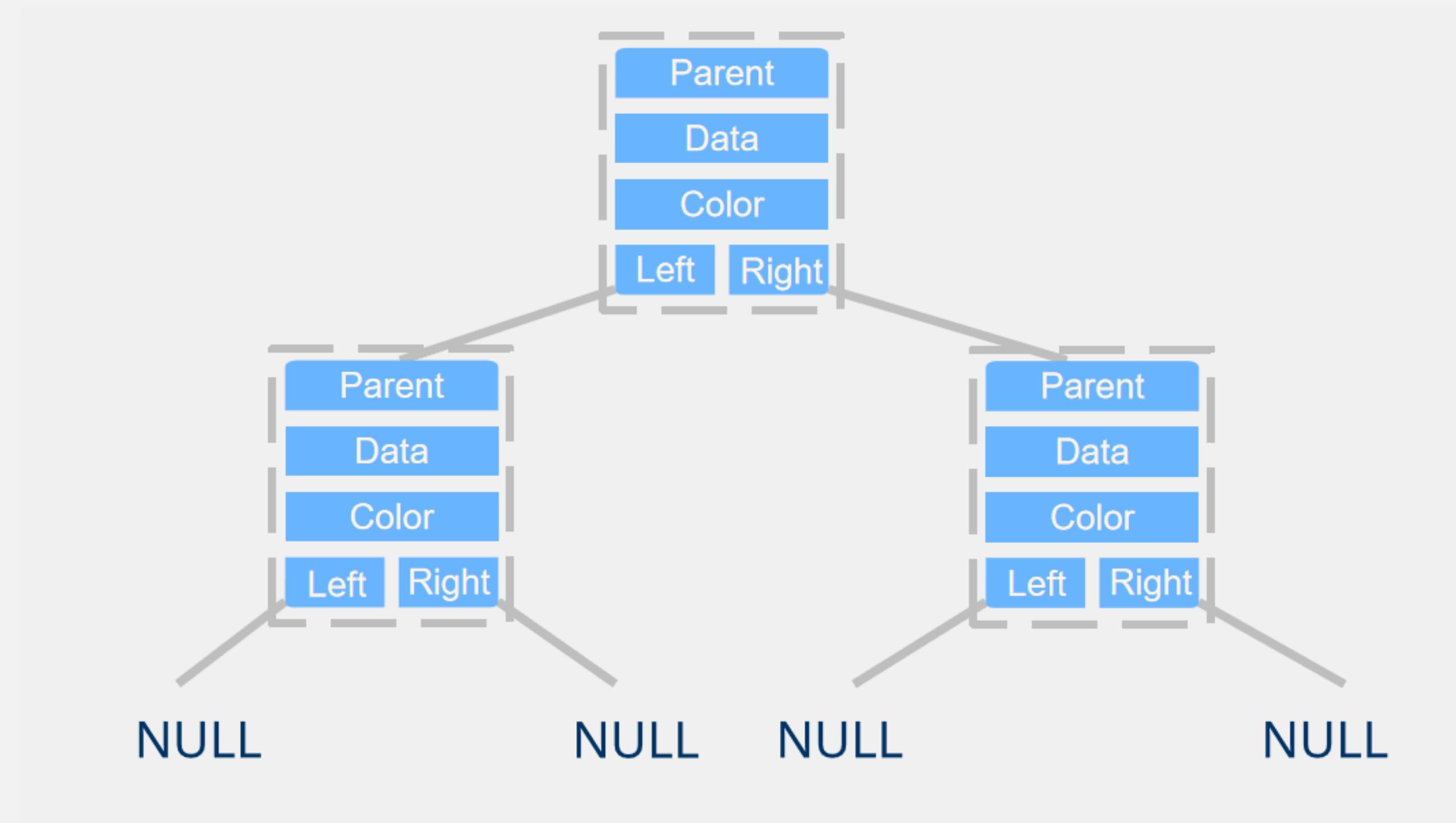


Đảm bảo các thao tác luôn có độ phức tạp $O(\log(n))$.

Ví dụ:



Cấu trúc của cây đỏ đen



Các thao tác với cây đở đen



Phép tìm kiếm

Phép chèn



Phép xoá



Phép tìm kiếm

- Tương tự như cây nhị phân tìm kiếm.

- So sánh khóa tại node hiện tại:

- Nếu nhỏ hơn \rightarrow đi trái.
- Nếu lớn hơn \rightarrow đi phải.
- Nếu bằng \rightarrow trả về node đó.

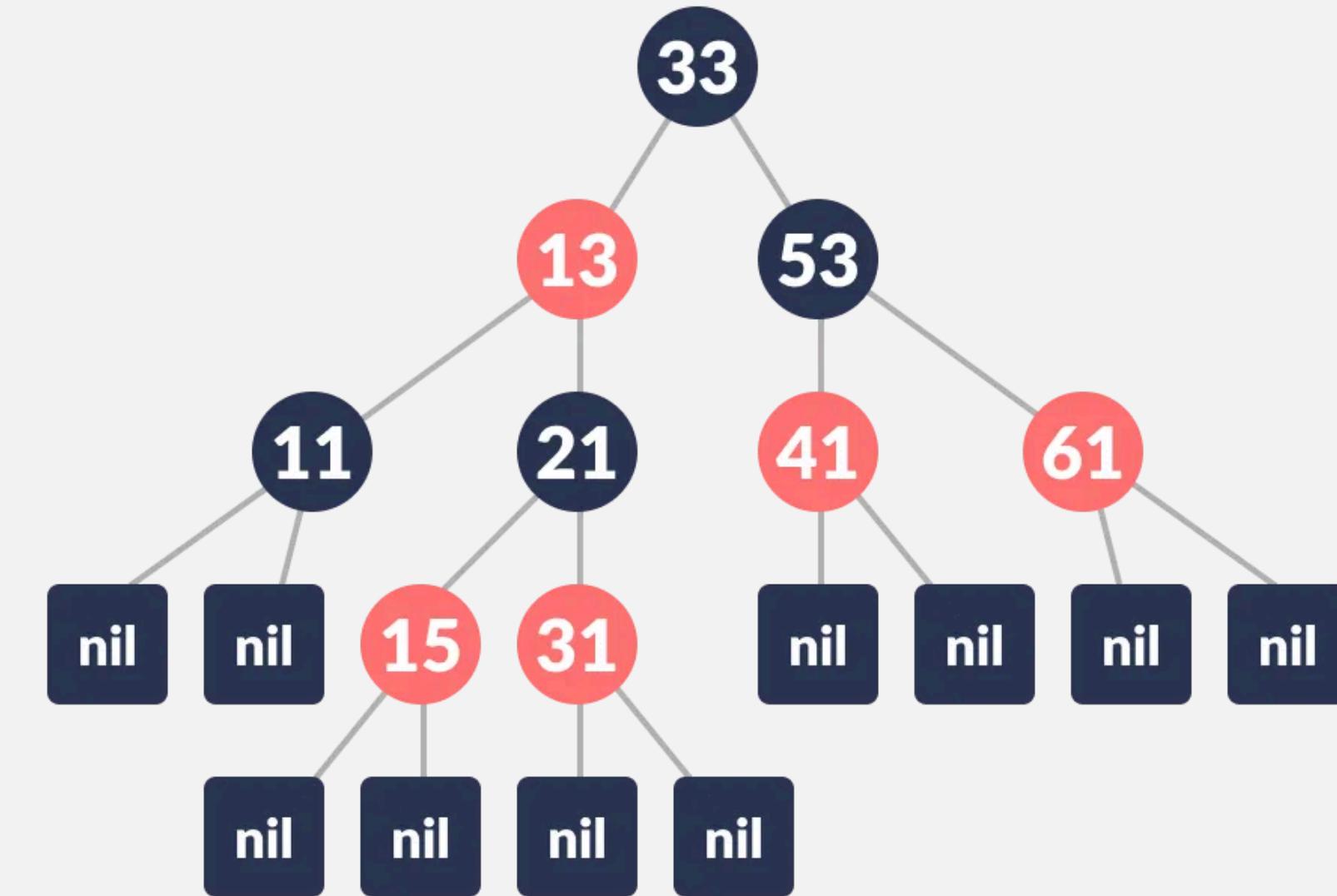
- Một cây đỏ đen có n node

Có: $\text{height} \leq 2\log(n+1)$

height: chiều cao cây

- Tính chất: $\text{height} \leq 2bh(x)$

- Thời gian tìm kiếm: $O(\log(n))$



Phép chèn

Việc thực hiện chèn một node N vào cây đỏ đen
được thực hiện theo các bước sau đây

1. Gán màu của node cần chèn là màu đỏ.
2. Ta thực hiện chèn một node vào cây đỏ đen giống như thao tác chèn trong cây nhị phân tìm kiếm.
3. Điều chỉnh lại nếu vi phạm các qui định của cây đỏ đen.

Phép chèn

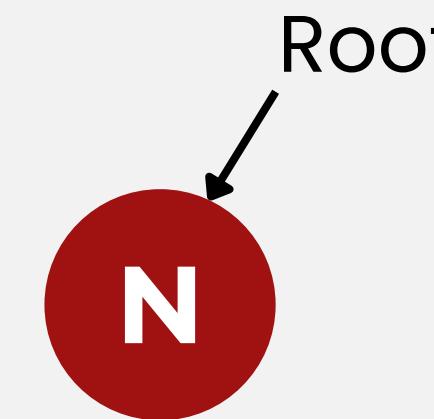
Một số qui ước gọi tên:

- Gọi **N** là node vừa được thêm vào cây
- Gọi **P** là node cha của N
- Gọi **U** là node chú bác của N (node cùng cha với P)
- Gọi **G** là node cha của nút P (node ông của N)

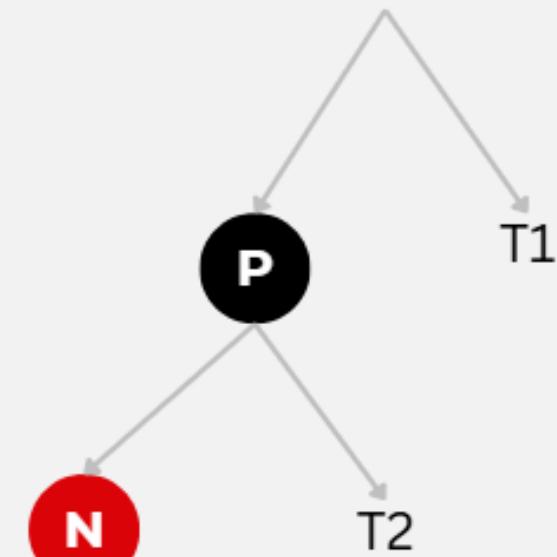
Phép chèn

Có 4 trường hợp có thể xảy ra khi chèn 1 node vào cây đỏ đen:

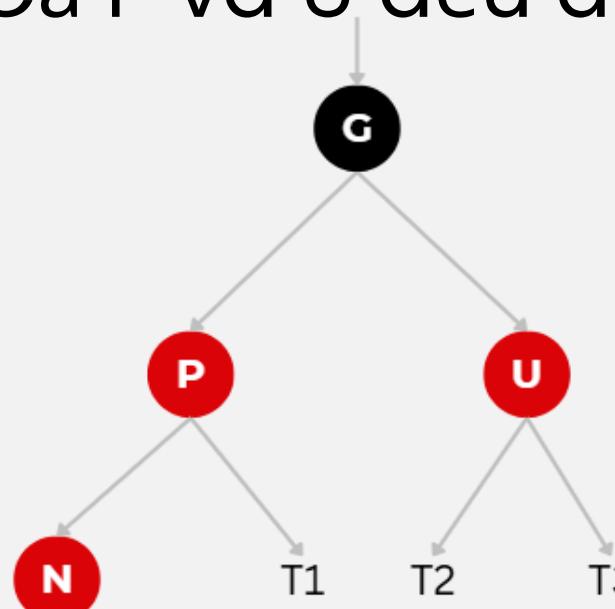
TH1: N được chèn vào vị trí node gốc của cây



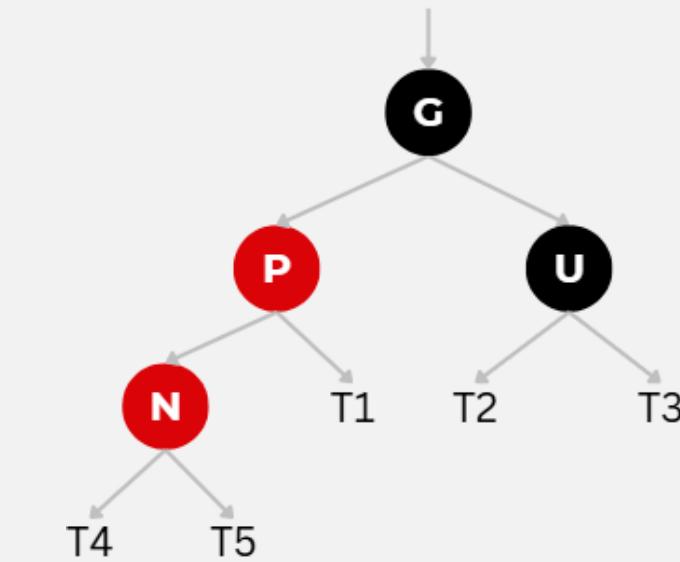
TH2: P là node đen



TH3: Cả P và U đều đỏ

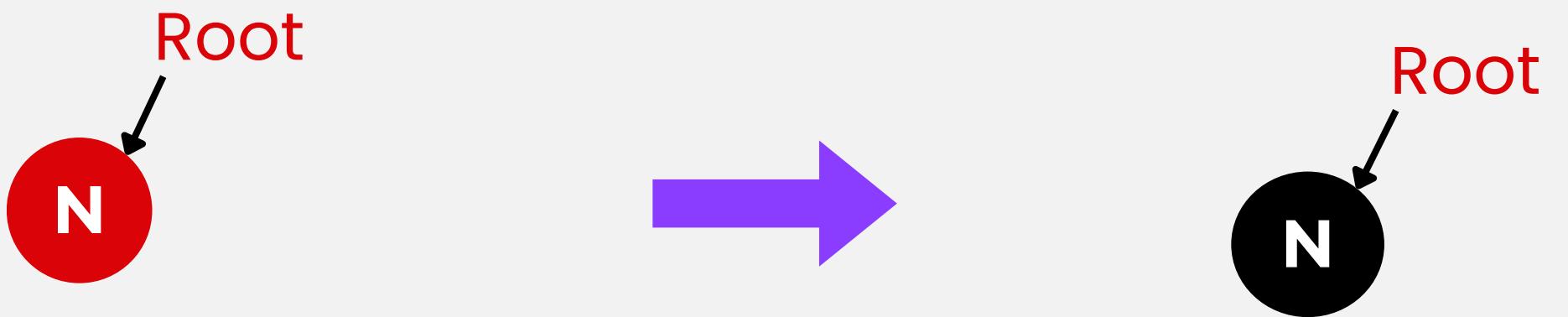


TH4: P là node đỏ, U là node đen



Phép chèn

TH1: N được chèn vào vị trí node gốc của cây

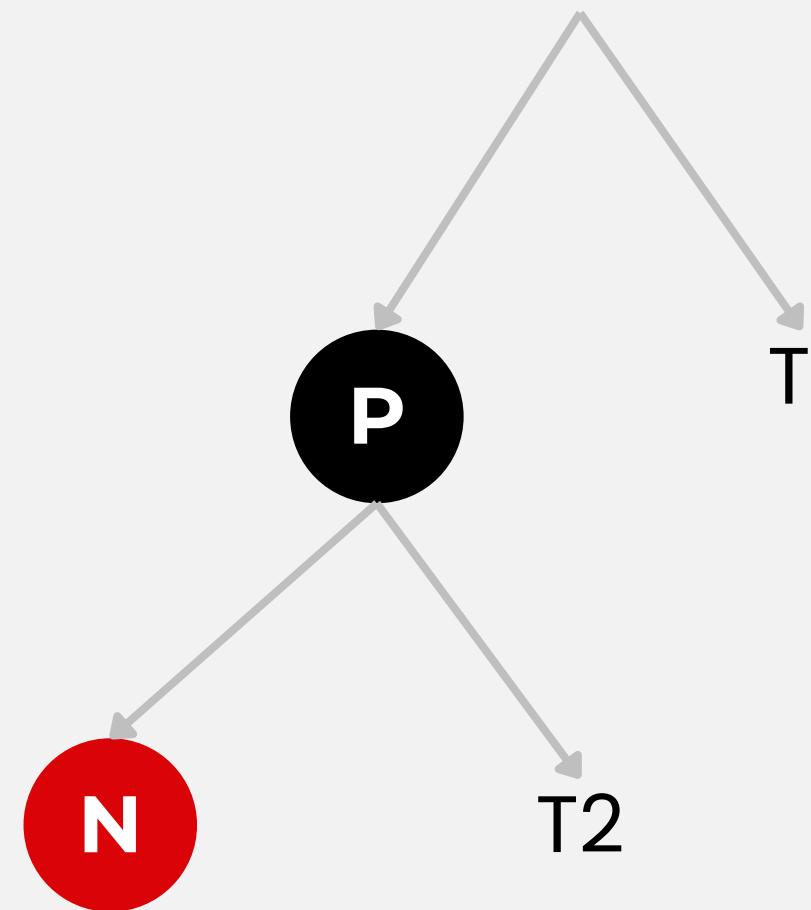


Vi phạm yêu cầu: node gốc phải là node đen

→ Đổi màu node N thành màu đen

Phép chèn

TH2: P là node đen



Không vi phạm định nghĩa của cây đỏ đen

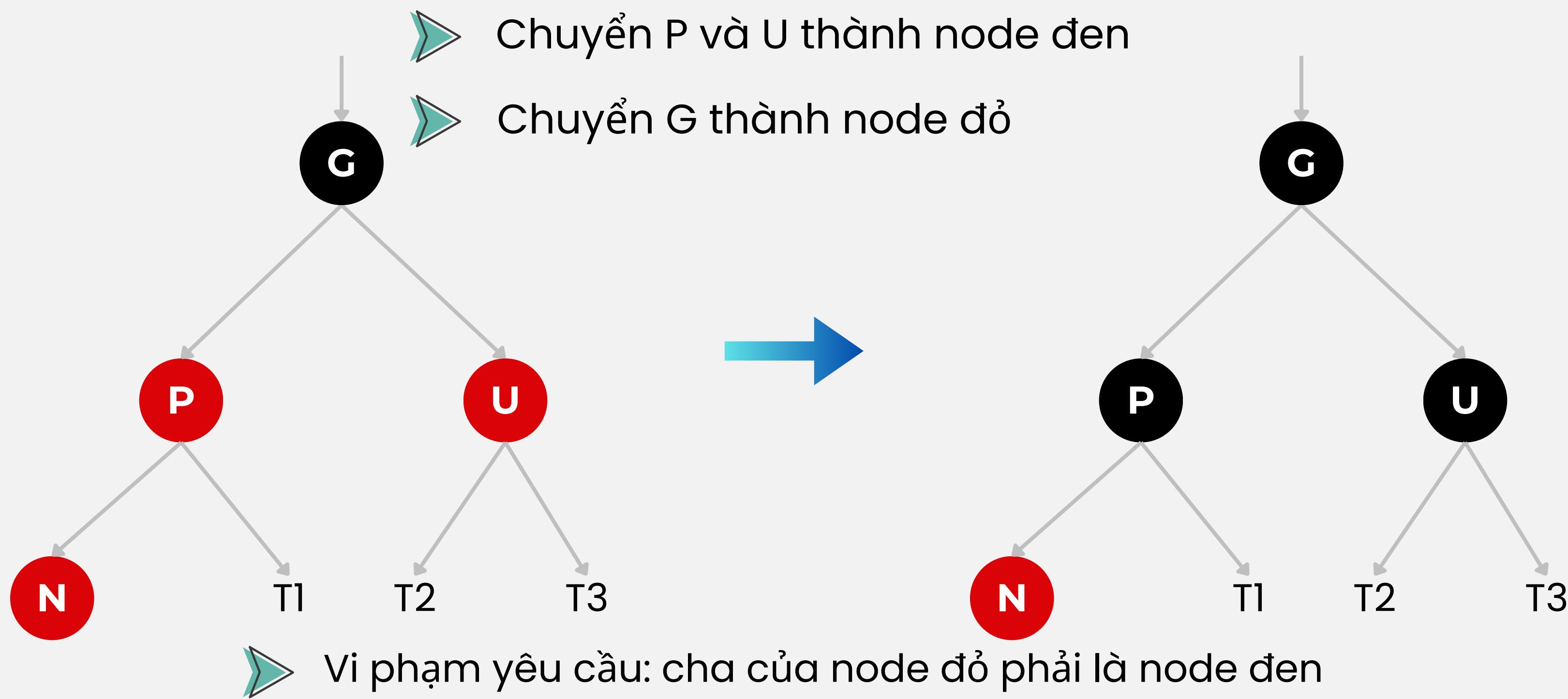
Không vi phạm tính chất của cây đỏ đen



Không cần phải điều chỉnh

Phép chèn

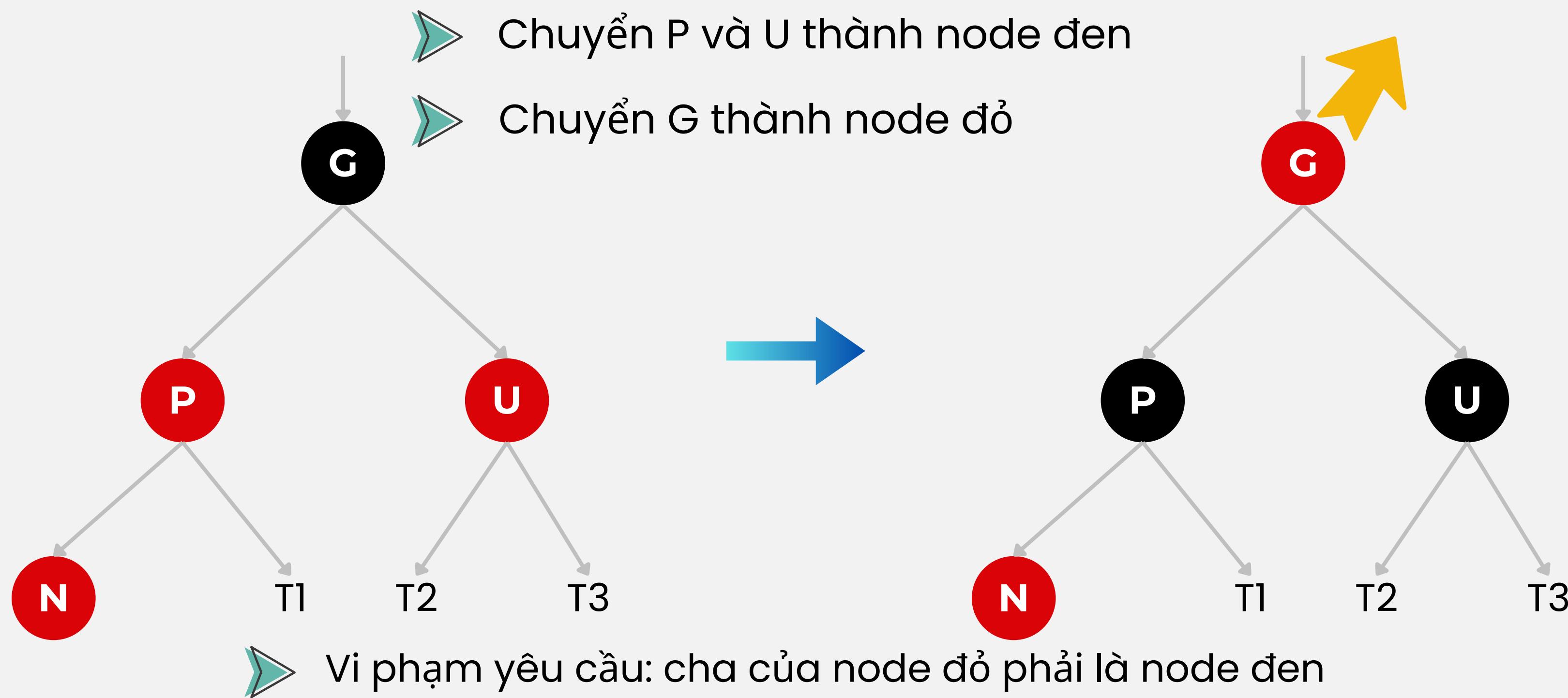
TH3: P là node đỏ và U là node đỏ



Phép chèn

TH3: P là node đő và U là node đő

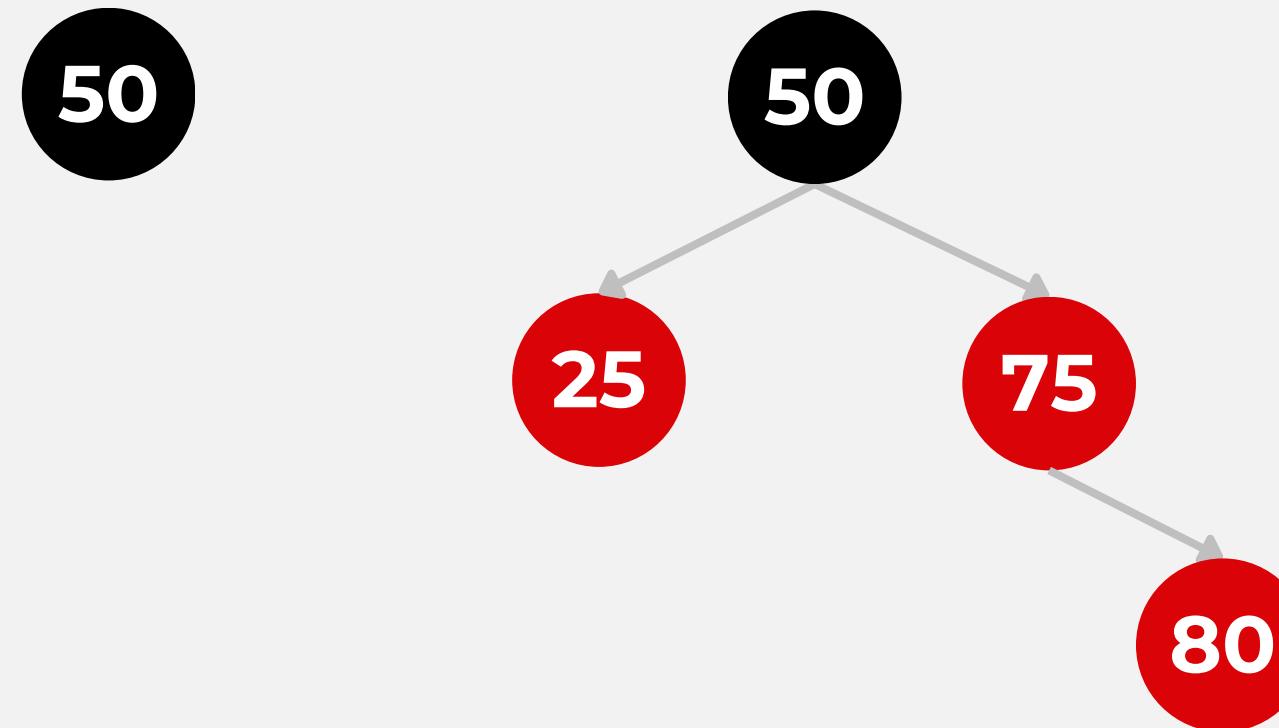
Lưu ý: Có thể vi phạm dây chuyền



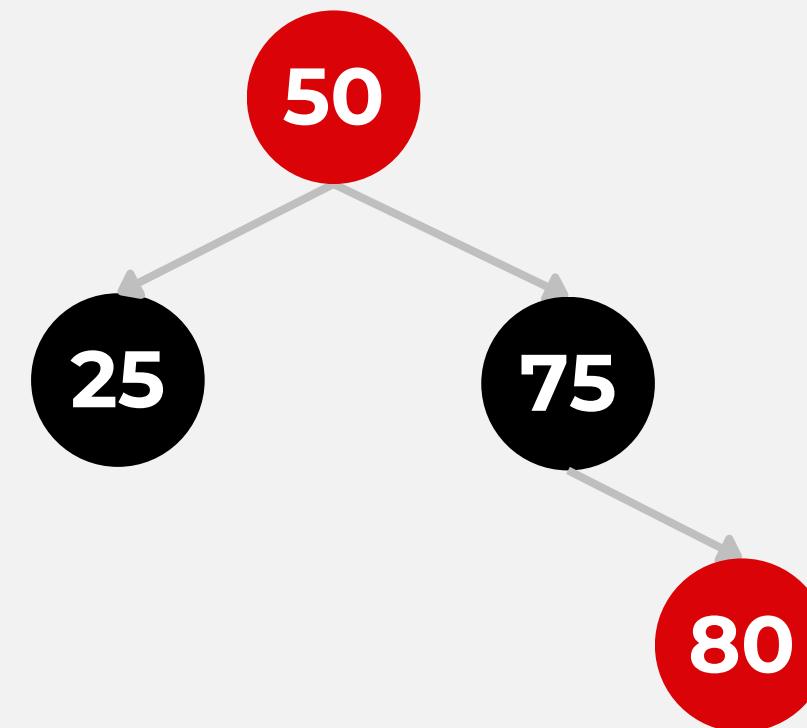
Ví dụ: Tạo cây đỏ đen

Tạo cây đỏ đen từ dãy số sau đây: 50, 75, 25, 80

Thêm 50



Thêm 75 → 25 → 80



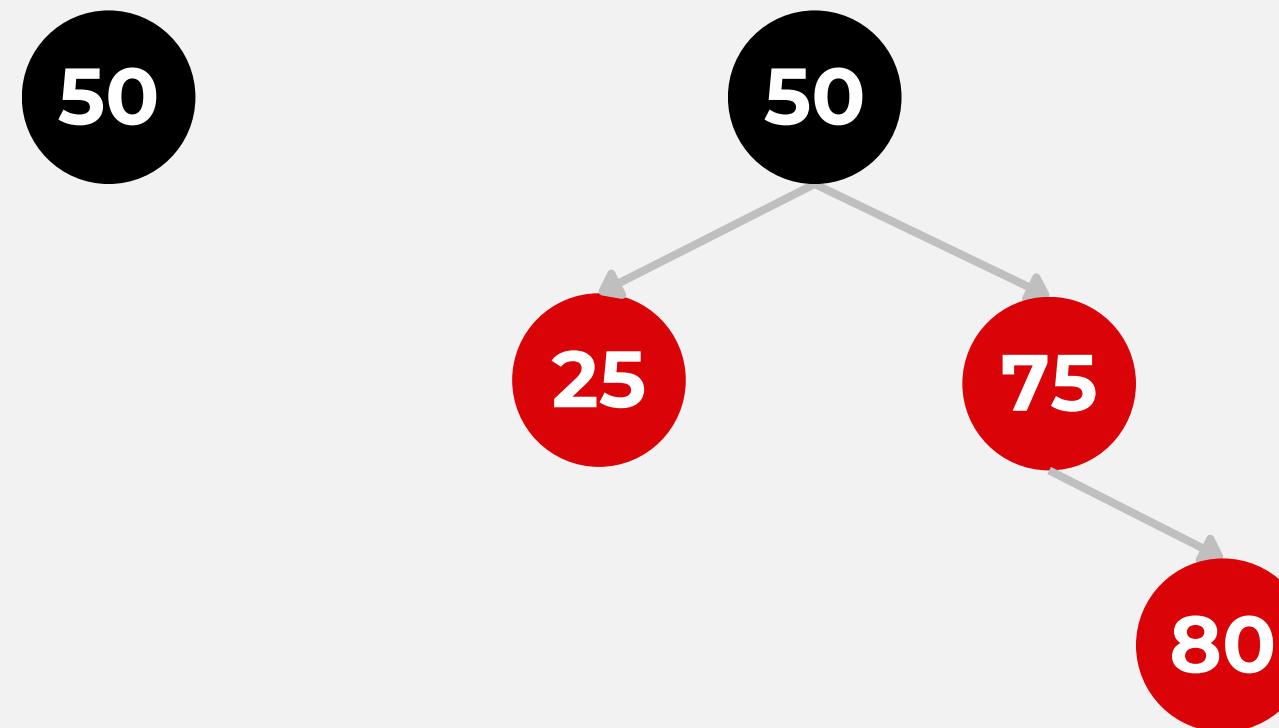
- TH3: P là node đỏ và U là node đen
- Đổi 75 và 25 thành đen
- 50 thành đỏ

- TH1: Nút gốc là màu đen
- Đổi 50 thành màu đen

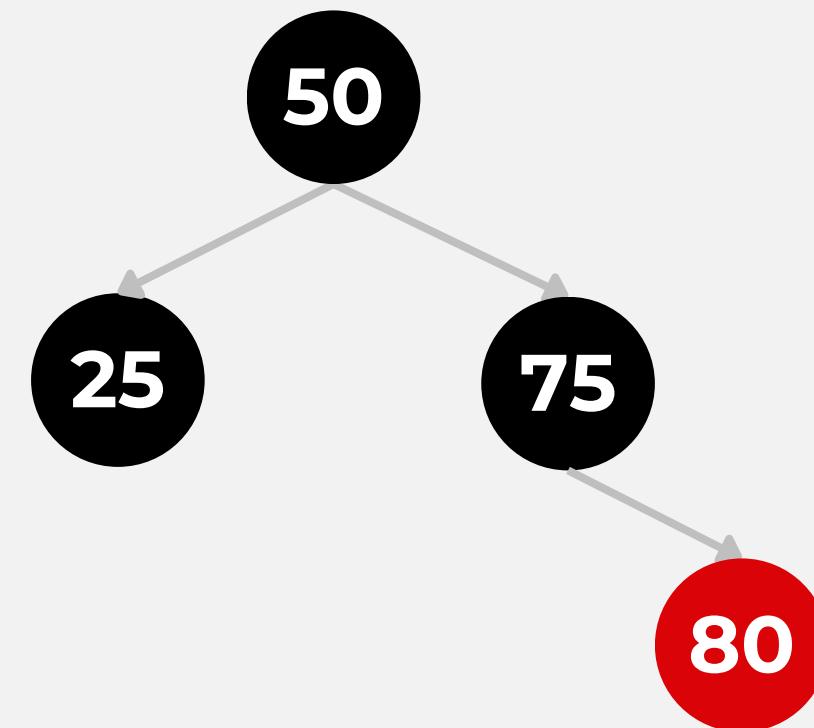
Ví dụ: Tạo cây đỏ đen

Tạo cây đỏ đen từ dãy số sau đây: 50, 75, 25, 80

Thêm 50



Thêm 75 → 25 → 80



TH3: P là node đỏ và U là node đen
- Đổi 75 và 25 thành đen
- 50 thành đỏ

TH1: Nút gốc là màu đen
- Đổi 50 thành màu đen

Phép chèn

TH4: P là node đỏ, U là node đen

P là **con trái** của G, N là **con trái** của P (Left-Left)

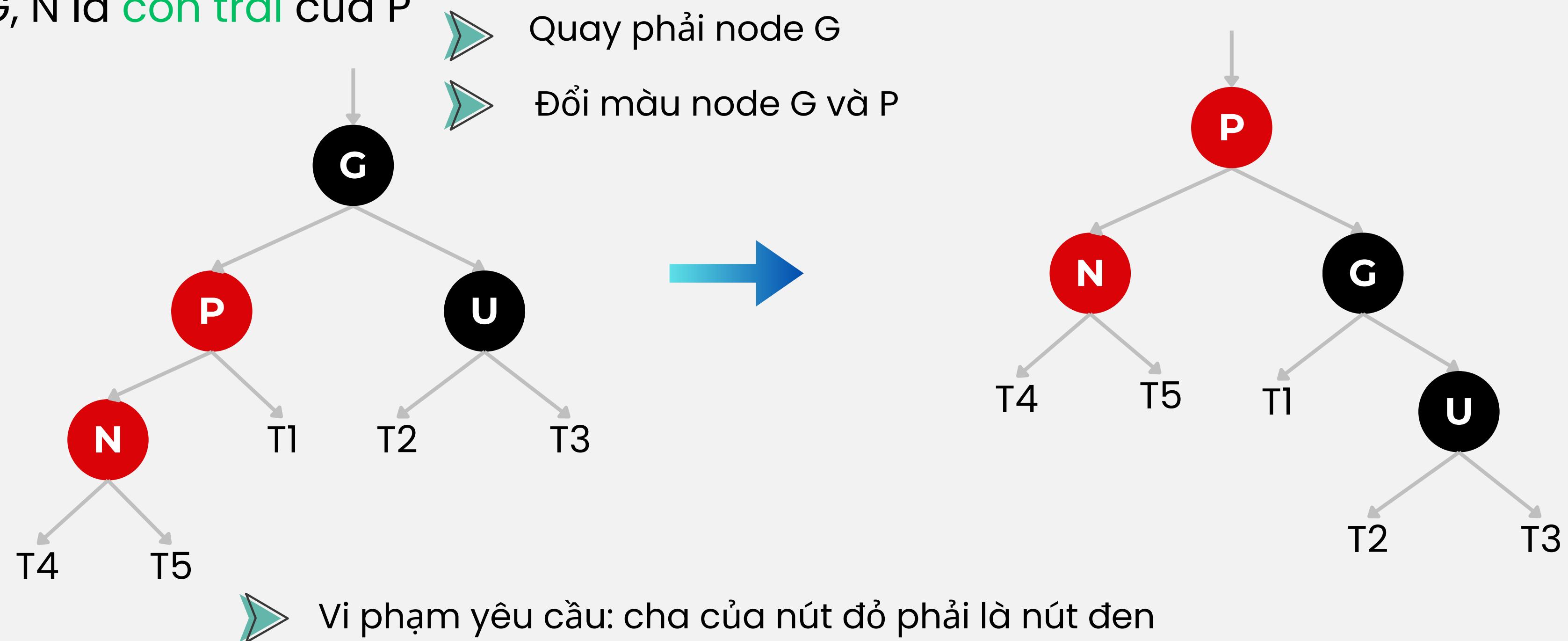
P là **con trái** của G, N là **con phải** của P (Left-Right)

P là **con phải** của G, N là **con phải** của P (Right-Right)

P là **con phải** của G, N là **con trái** của P (Right-Left)

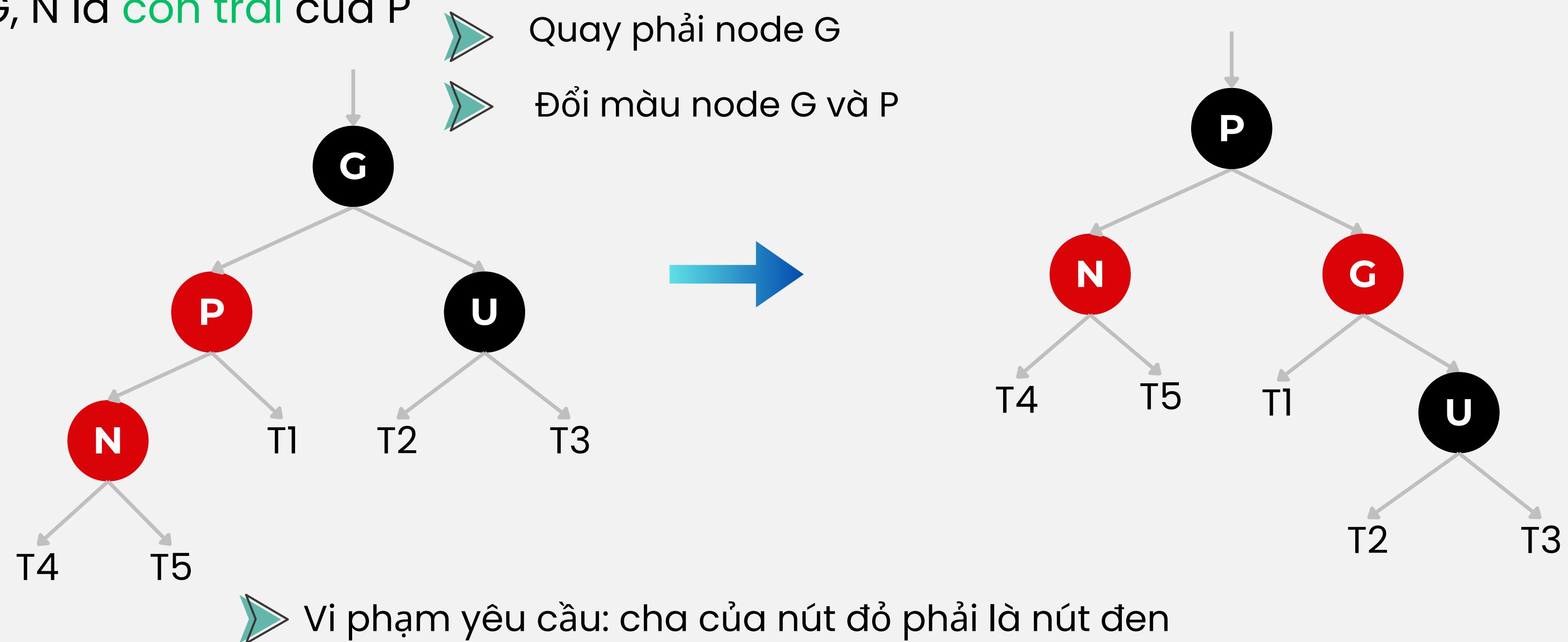
Phép chèn

Left-Left: P là node đỏ và U là node đen, P là **con trái** của G, N là **con trái** của P



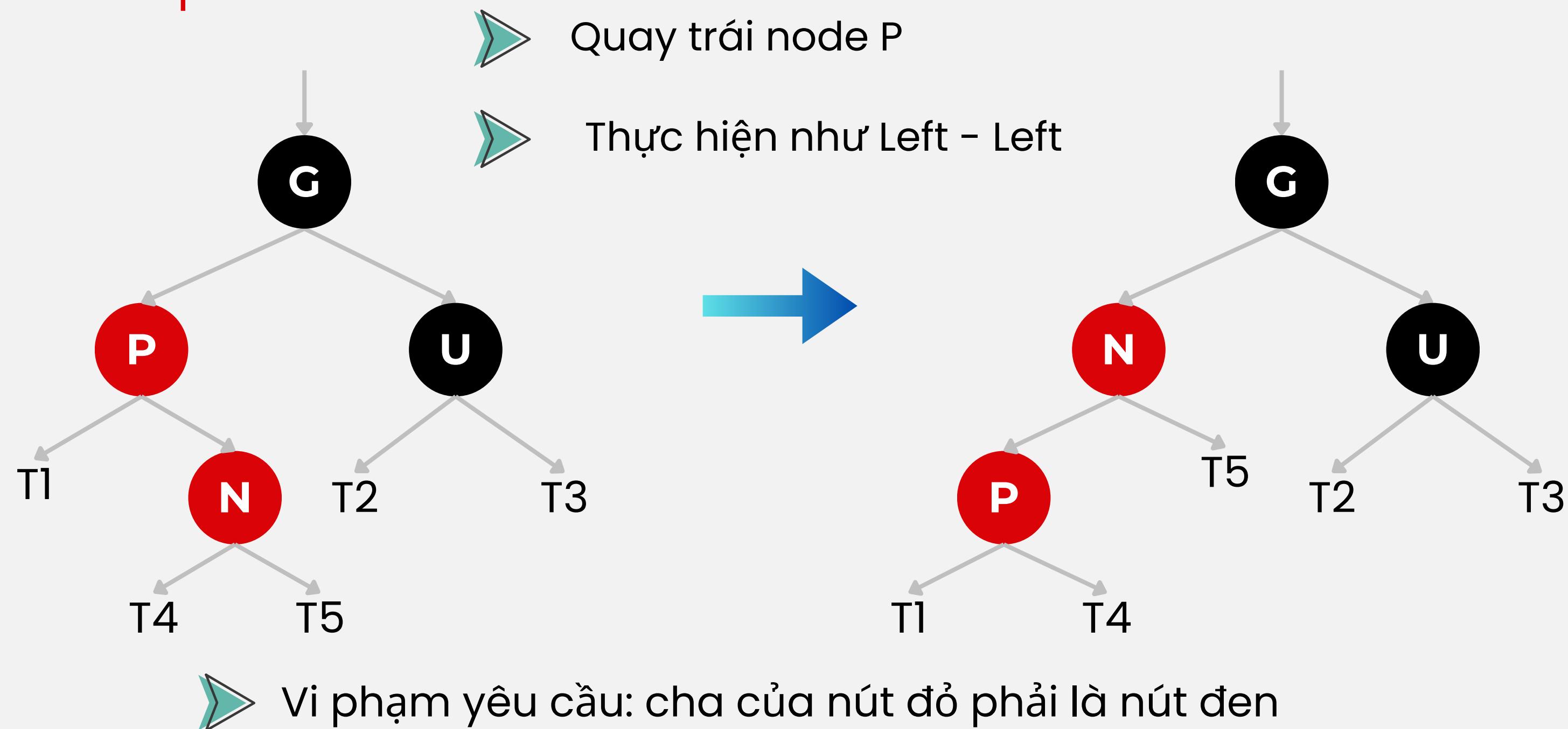
Phép chèn

Left-Left: P là node đỏ và U là node đen, P là **con trái** của G, N là **con trái** của P



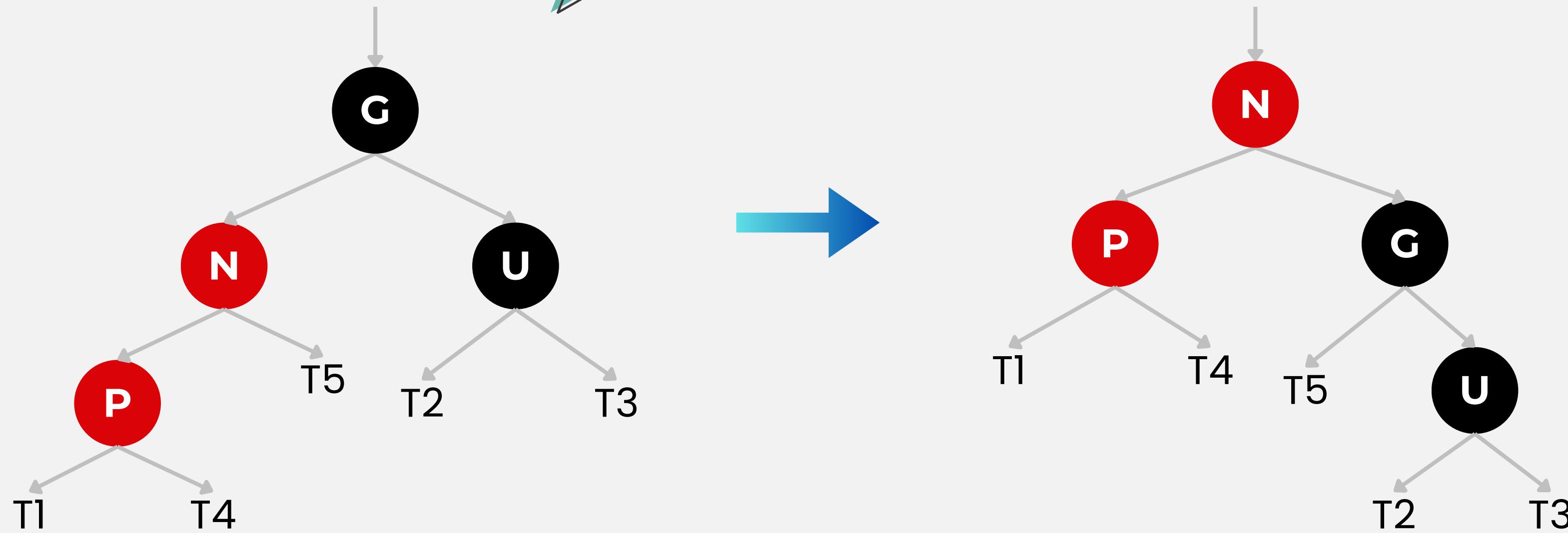
Phép chèn

Left-Right: P là node đỏ và U là node đen, P là **con trái** của G, N là **con phải** của P



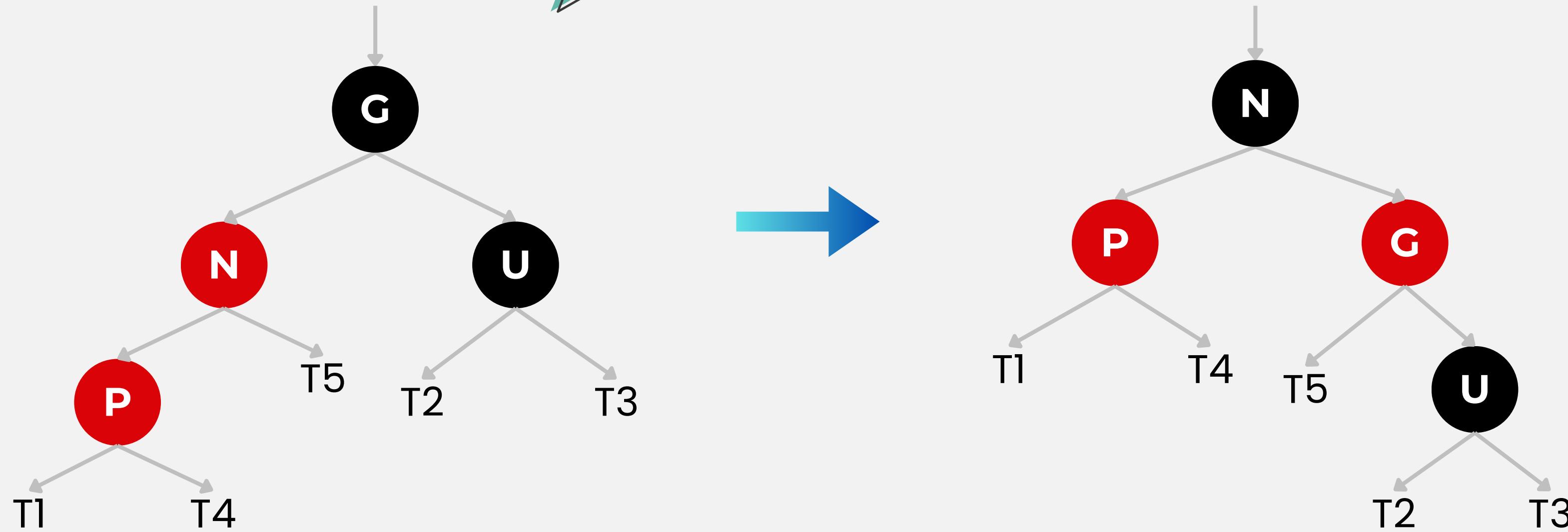
Phép chèn

- Quay phải node G
- Đổi màu node G và P



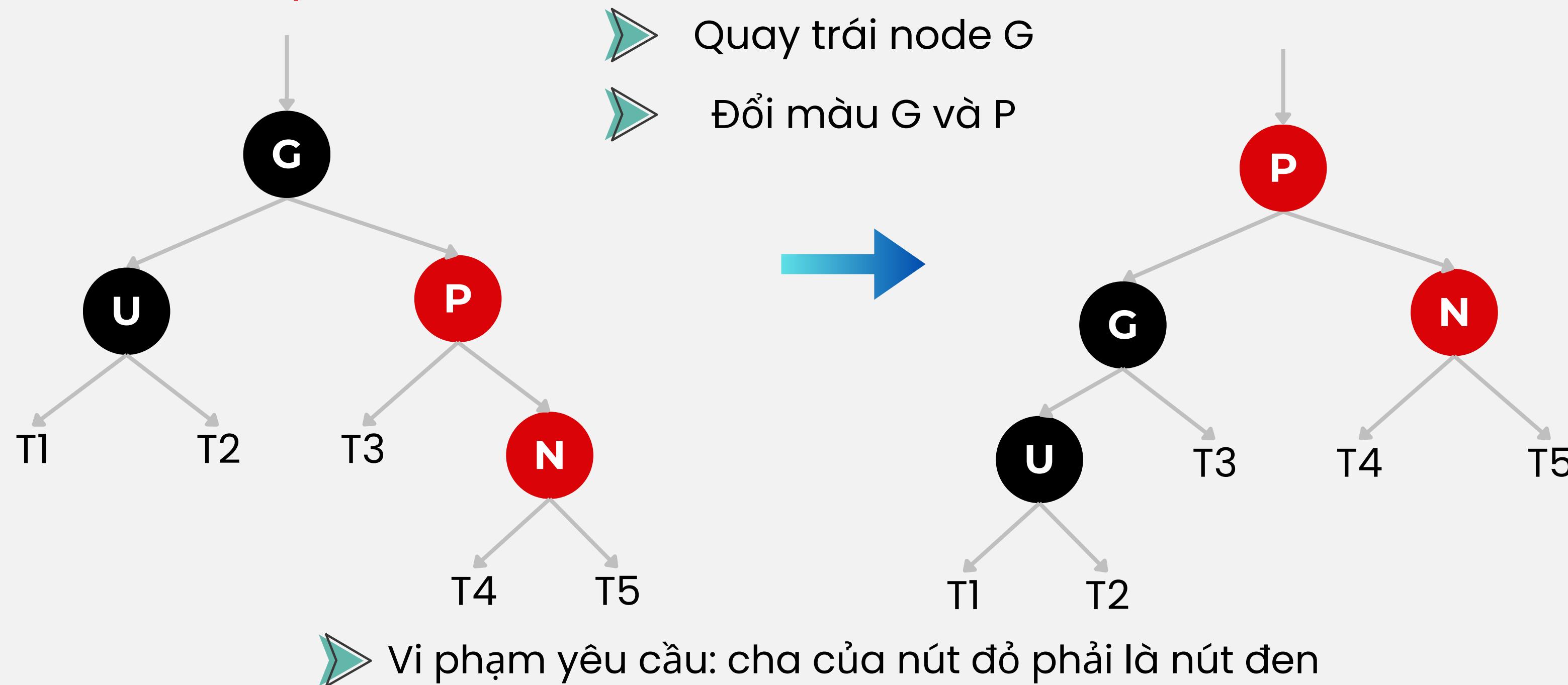
Phép chèn

- Quay phải node G
- Đổi màu node G và P



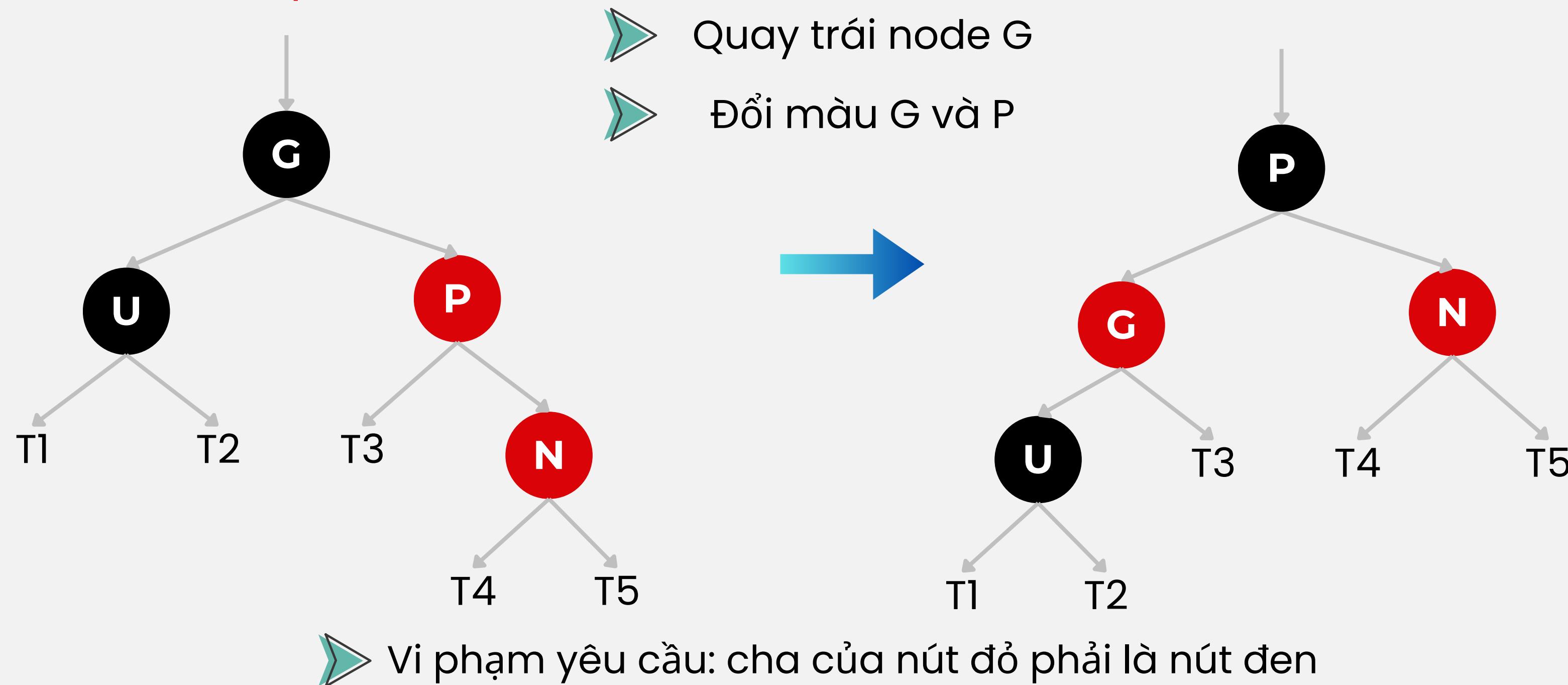
Phép chèn

Right-Right: P là node đỏ và U là node đen, P là **con phải** của G, N là **con phải** của P



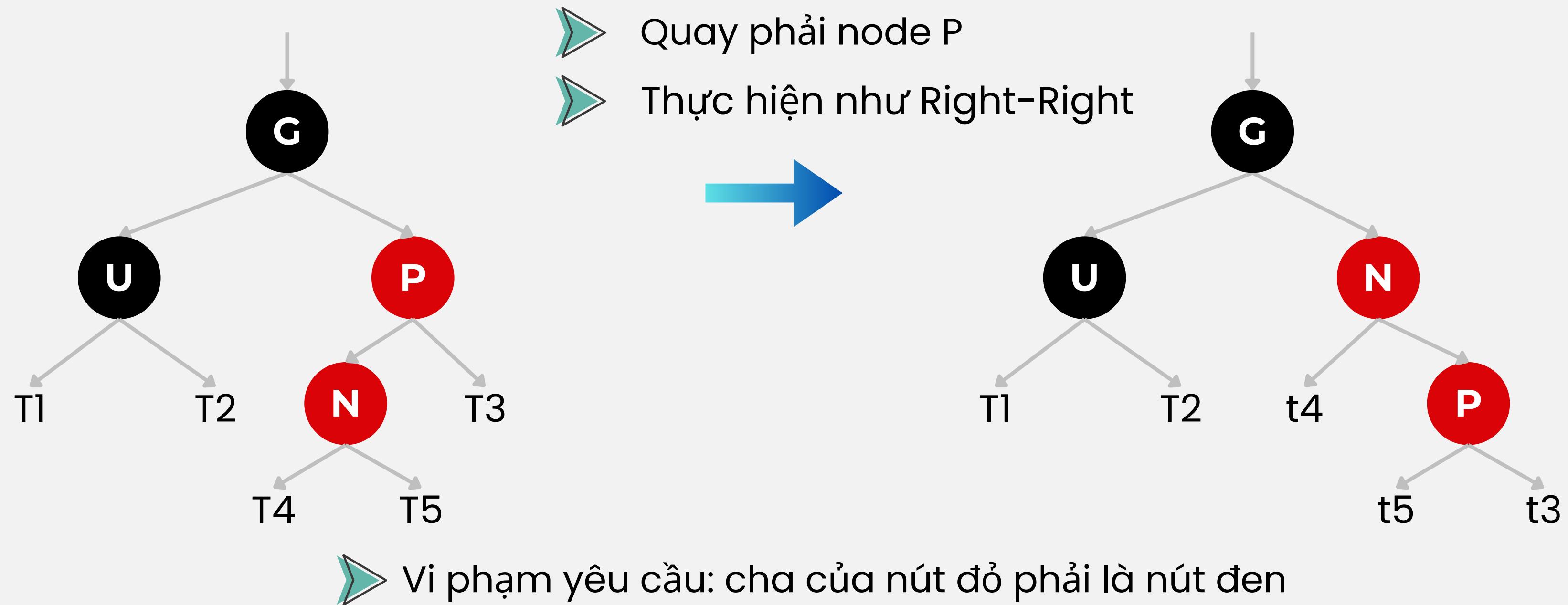
Phép chèn

Right-Right: P là node đỏ và U là node đen, P là **con phải** của G, N là **con phải** của P

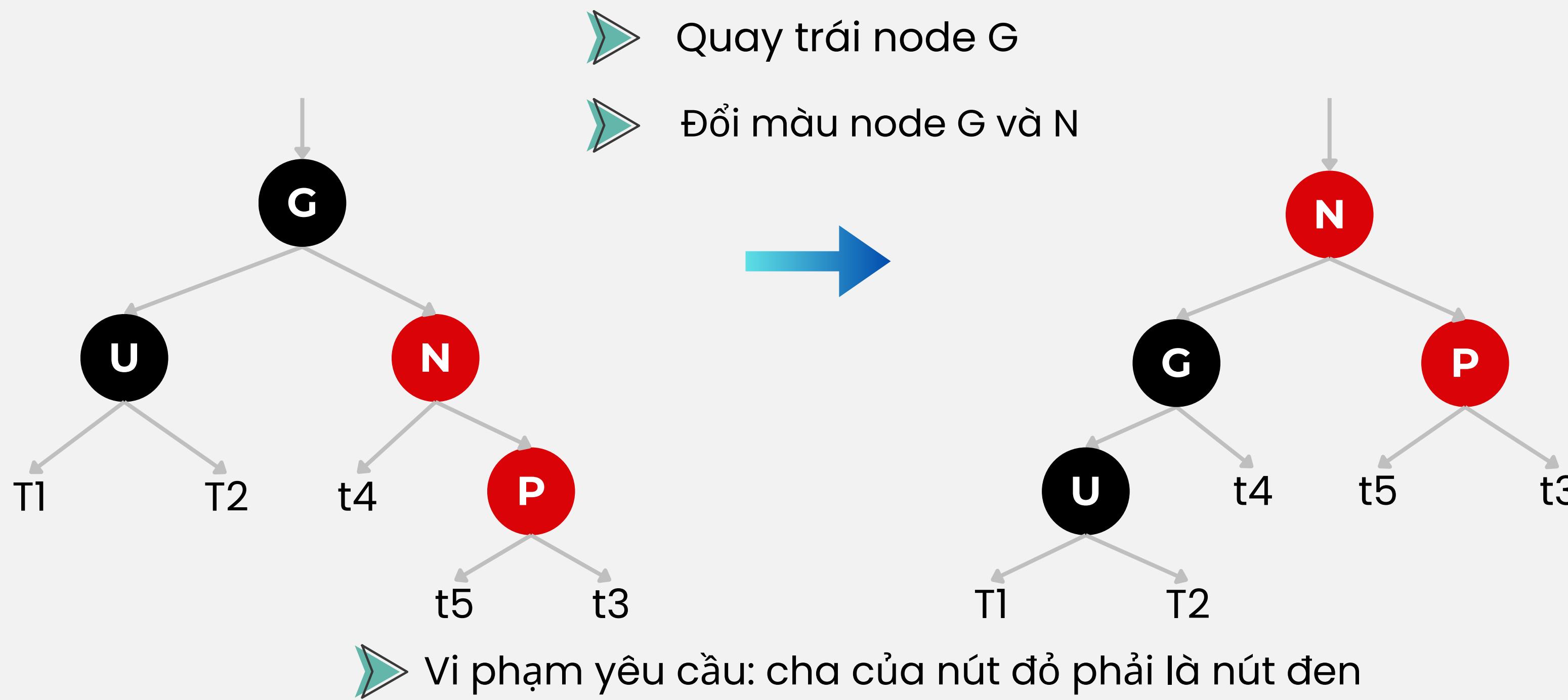


Phép chèn

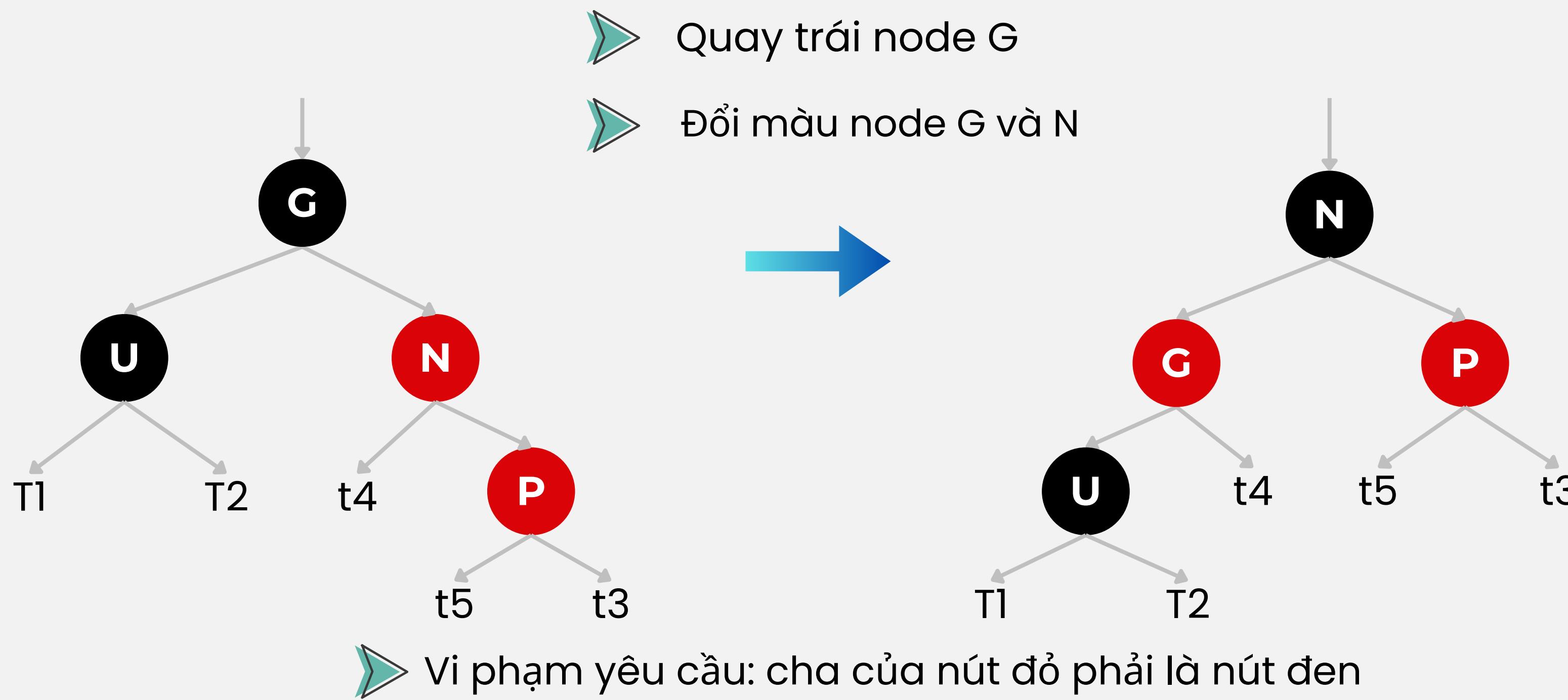
Right-Left: P là node đỏ và U là node đen, P là **con phải** của G, N là **con trái** của P



Phép chèn



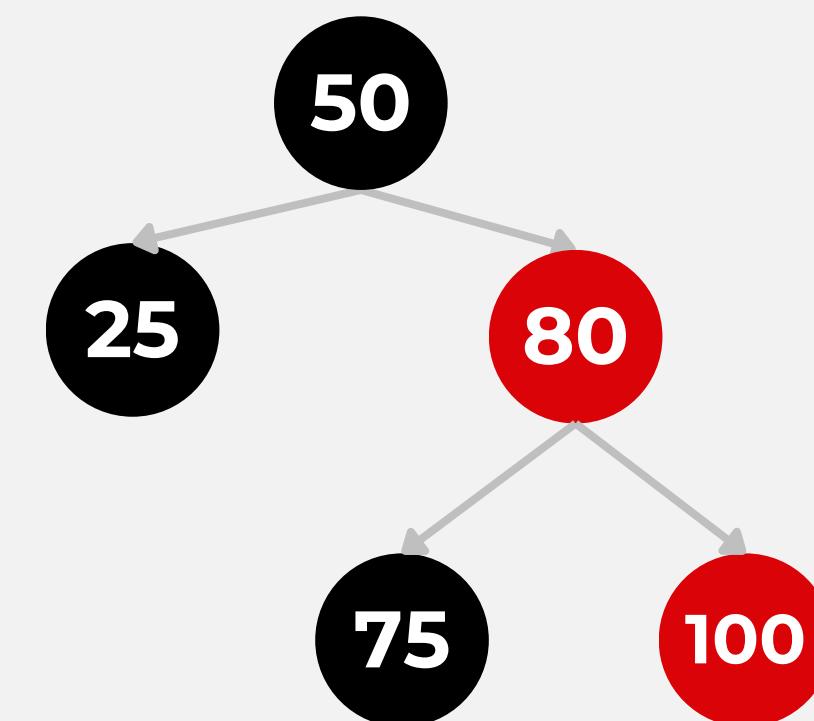
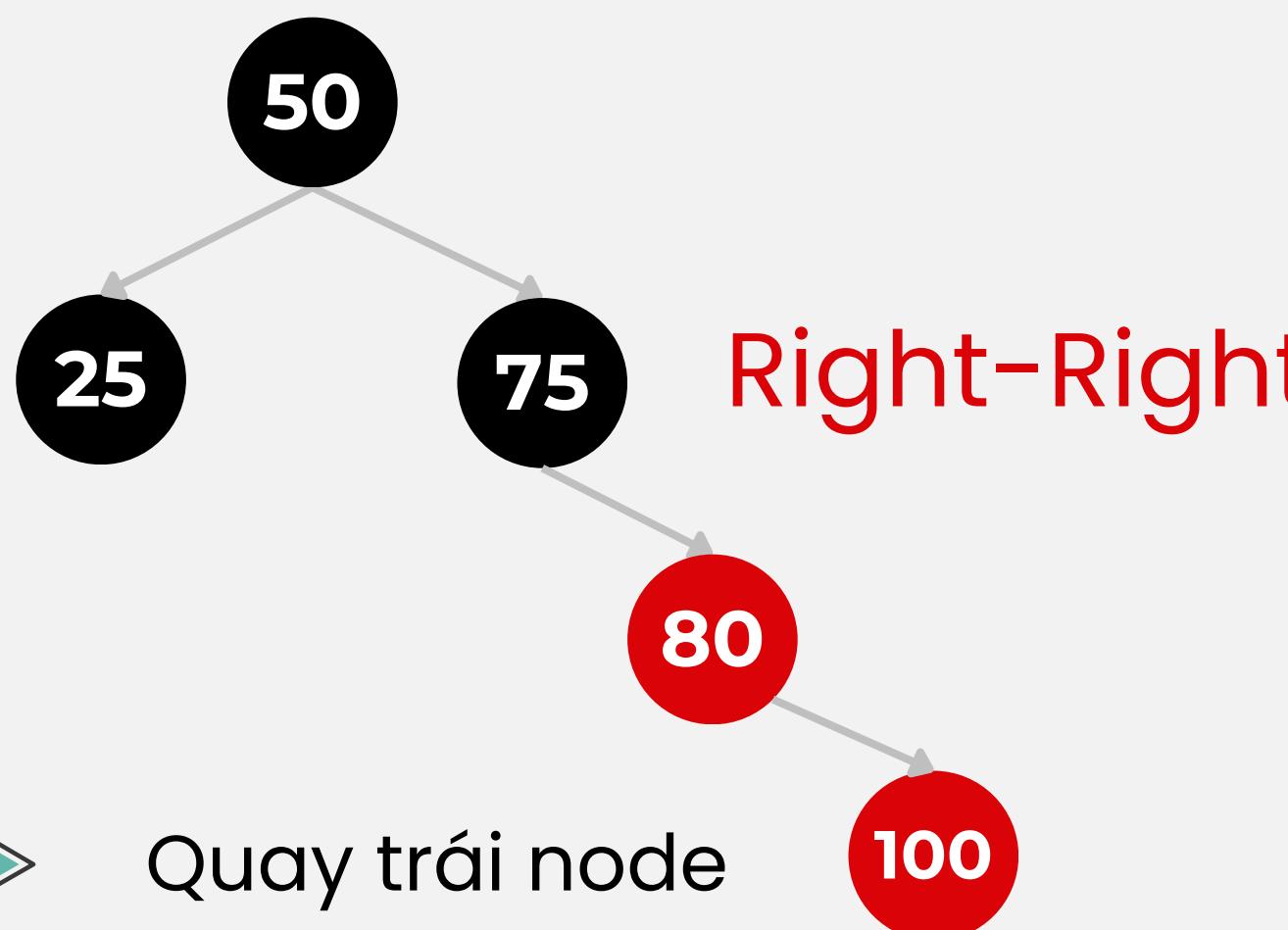
Phép chèn



Ví dụ: Tạo cây đỏ đen

Tạo cây đỏ đen từ dãy số sau đây: 50, 75, 25, 80 , 100, 110, 105

Thêm 100



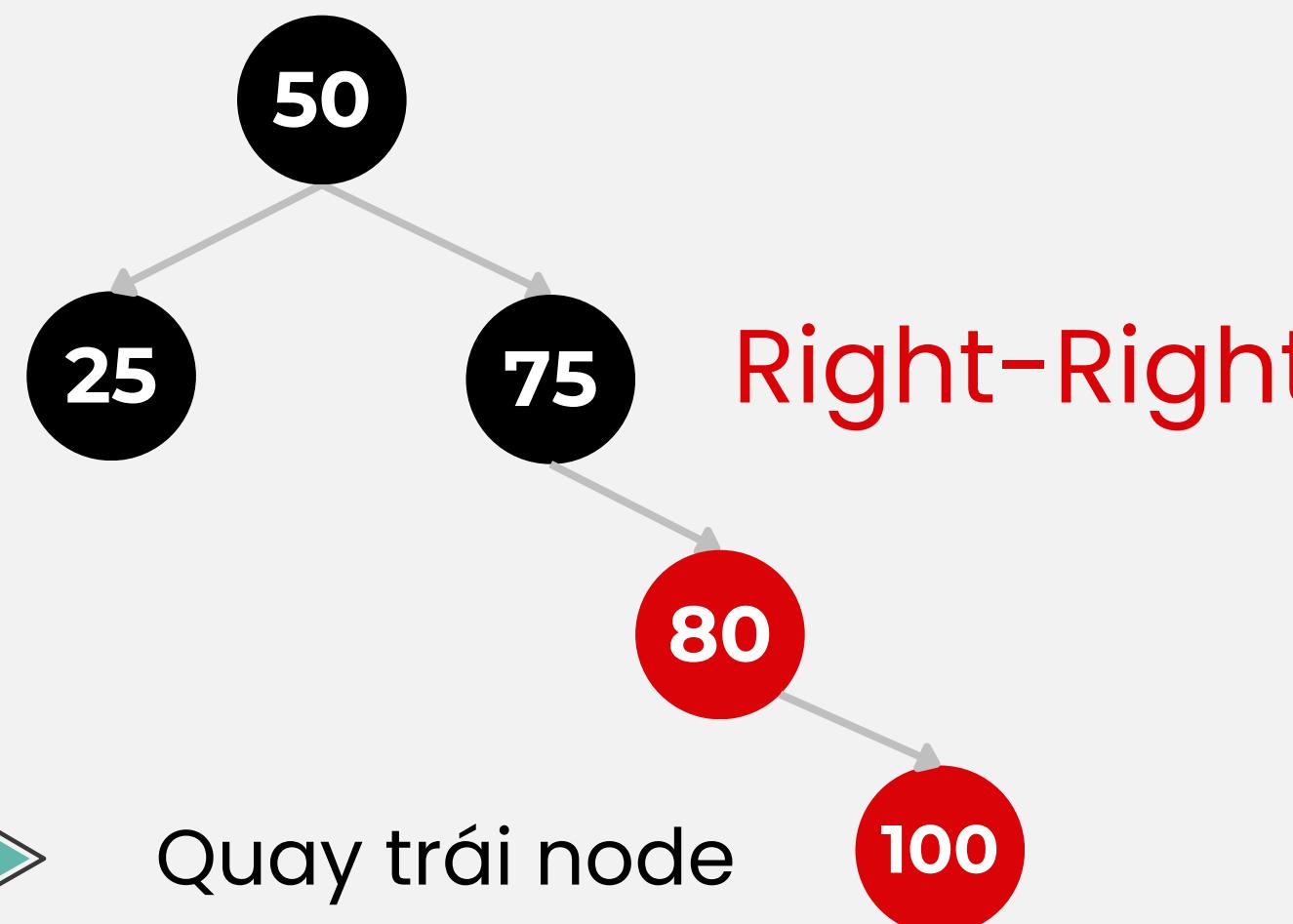
Quay trái node

Đổi màu node 75 và

Ví dụ: Tạo cây đỏ đen

Tạo cây đỏ đen từ dãy số sau đây: 50, 75, 25, 80, 100, 110, 105

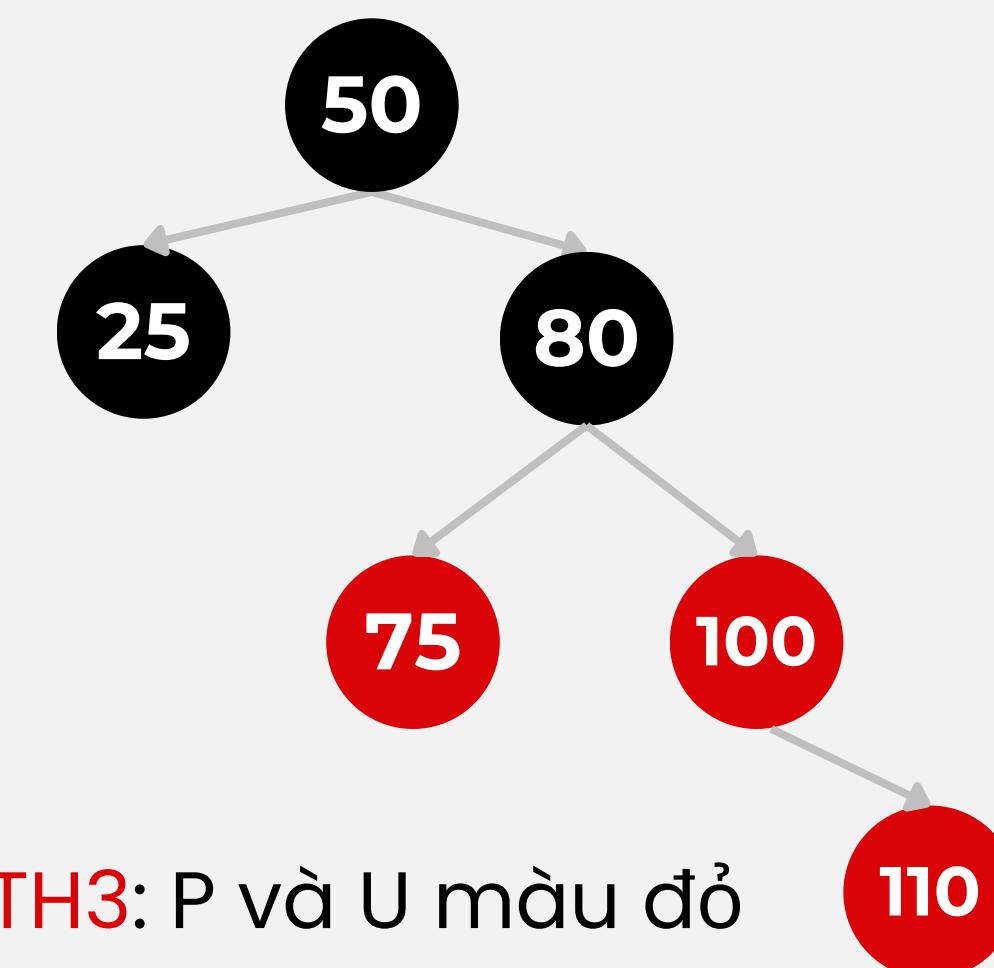
Thêm 100



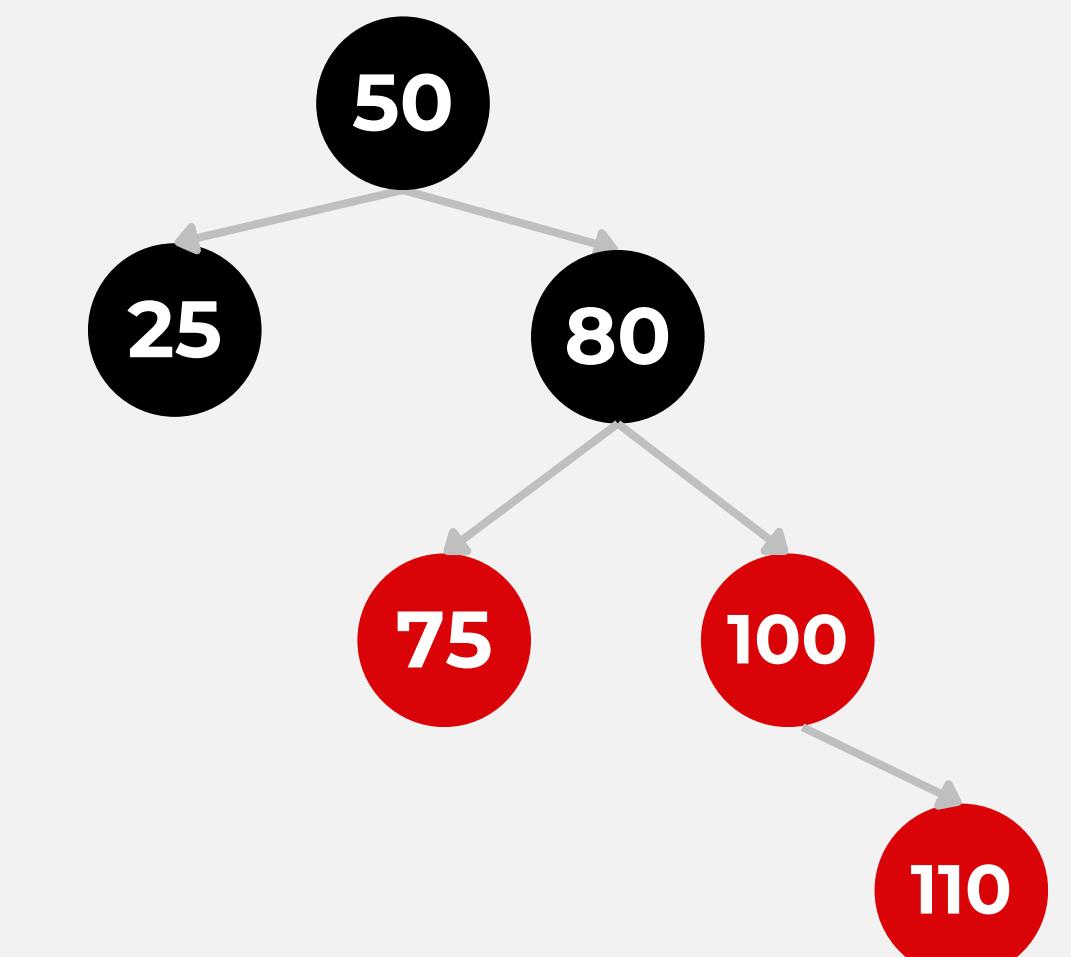
Quay trái node

Đổi màu node 75 và 80

Thêm 110



TH3: P và U màu đỏ



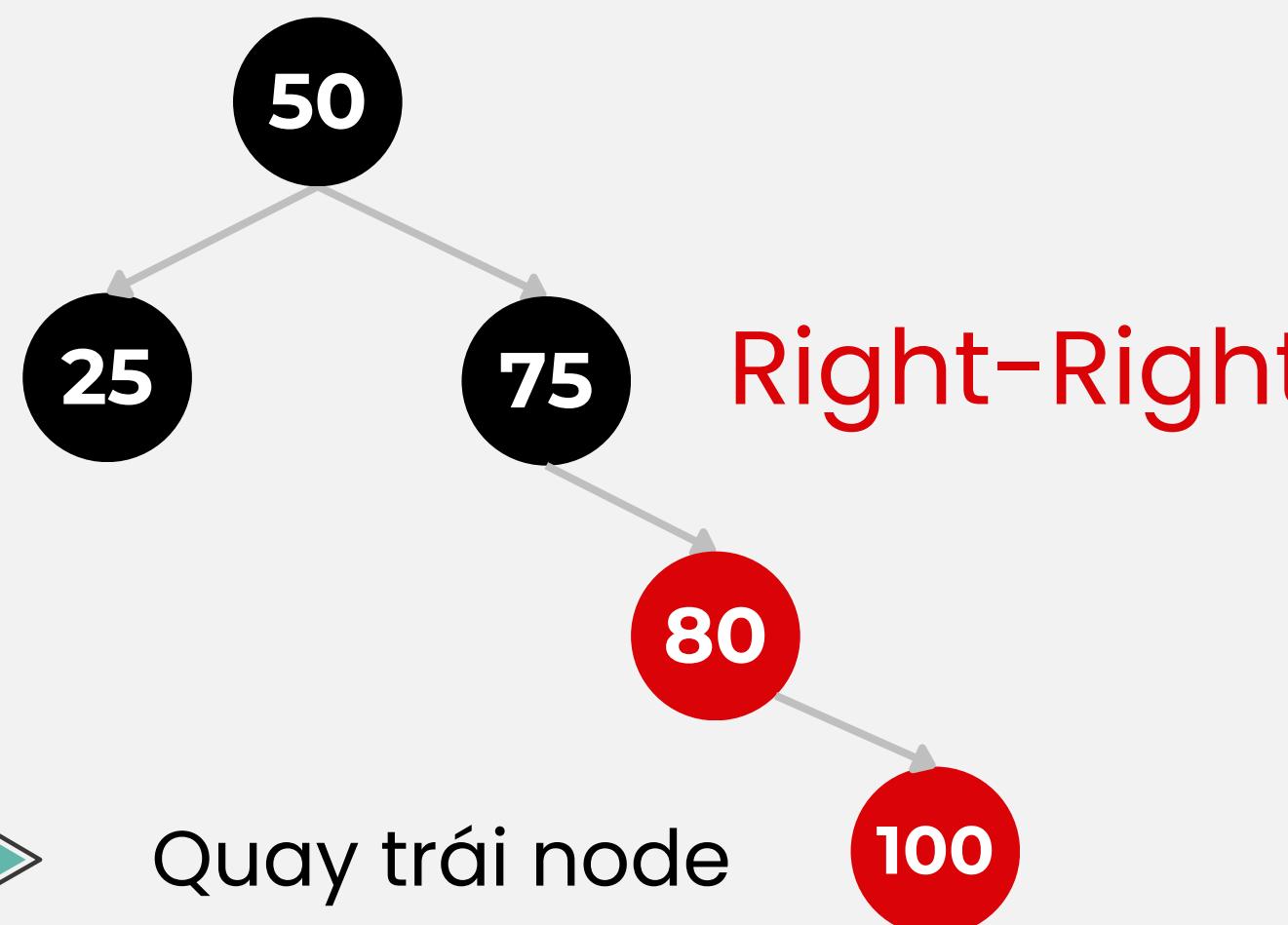
Đổi 75 và 100 thành đen

Đổi 80 thành đỏ

Ví dụ: Tạo cây đỏ đen

Tạo cây đỏ đen từ dãy số sau đây: 50, 75, 25, 80, 100, 110, 105

Thêm 100

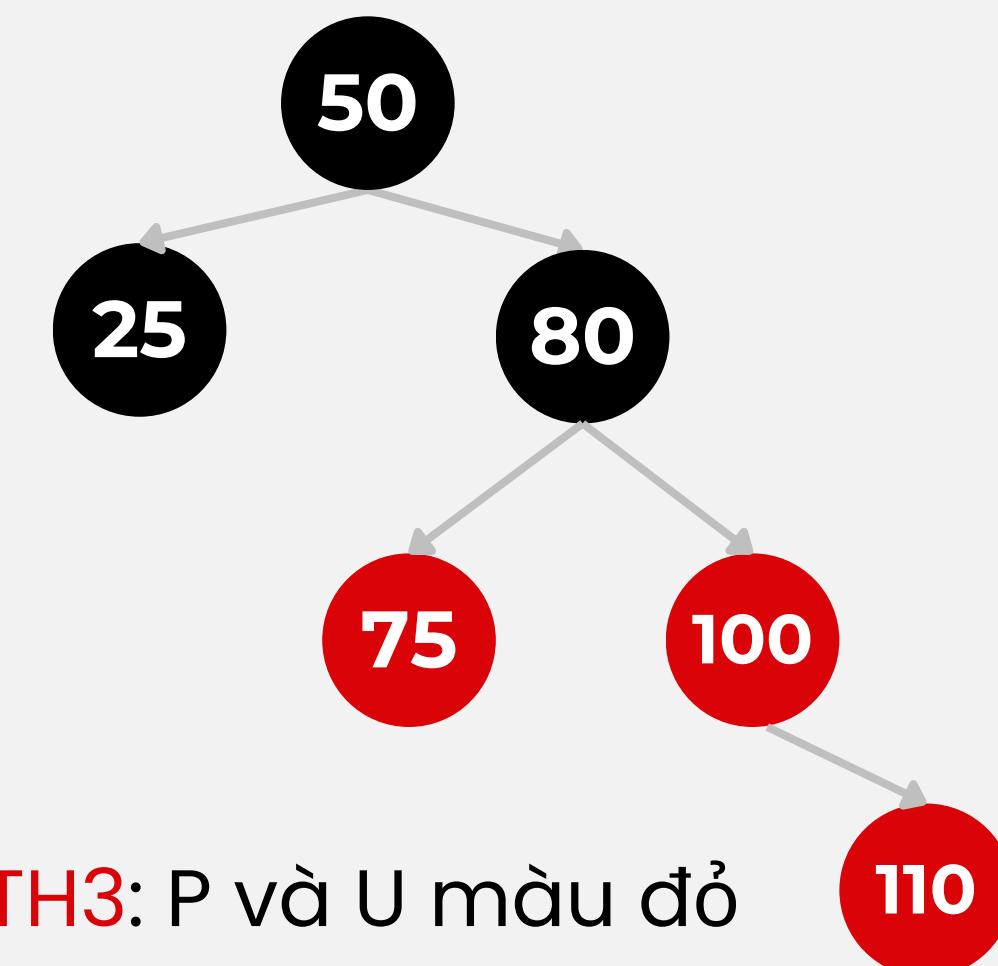


Quay trái node

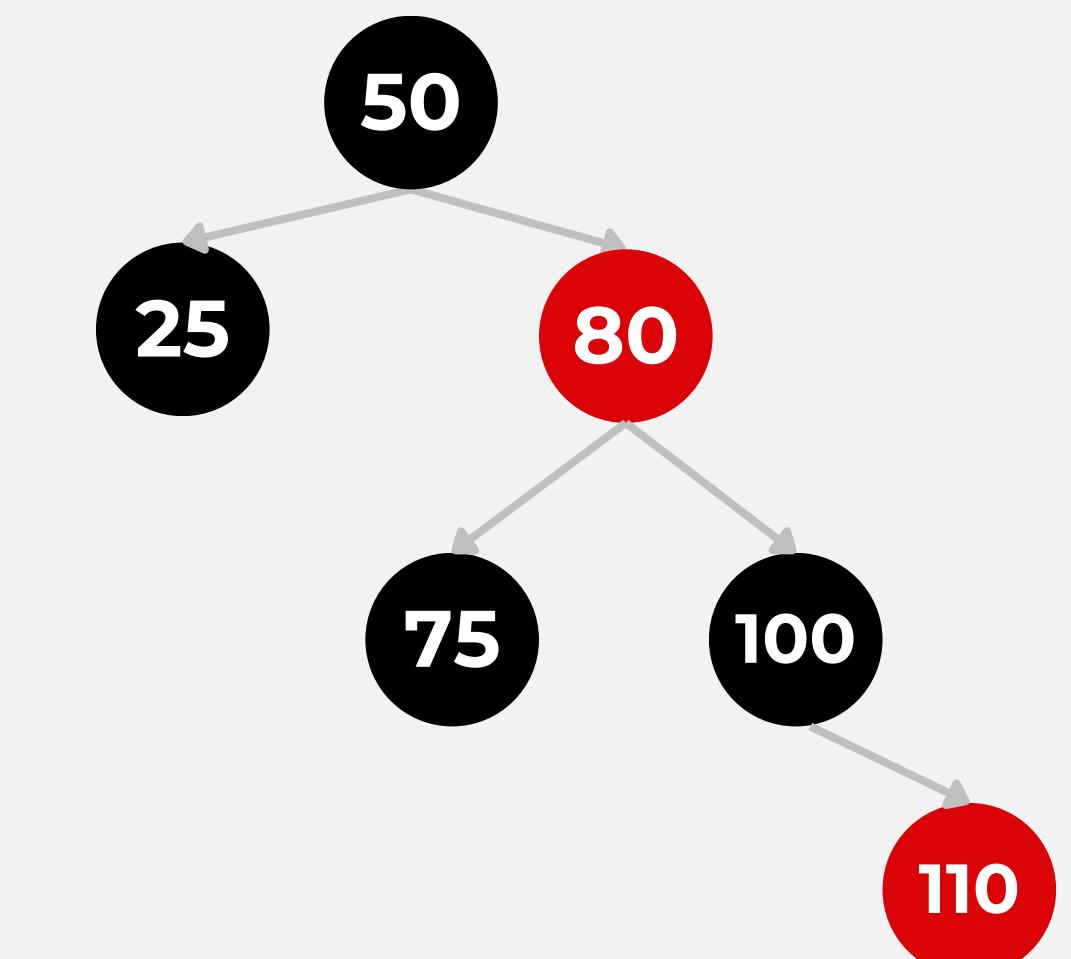
Đổi màu node 75 và

80

Thêm 110



Đổi 75 và 100 thành đen

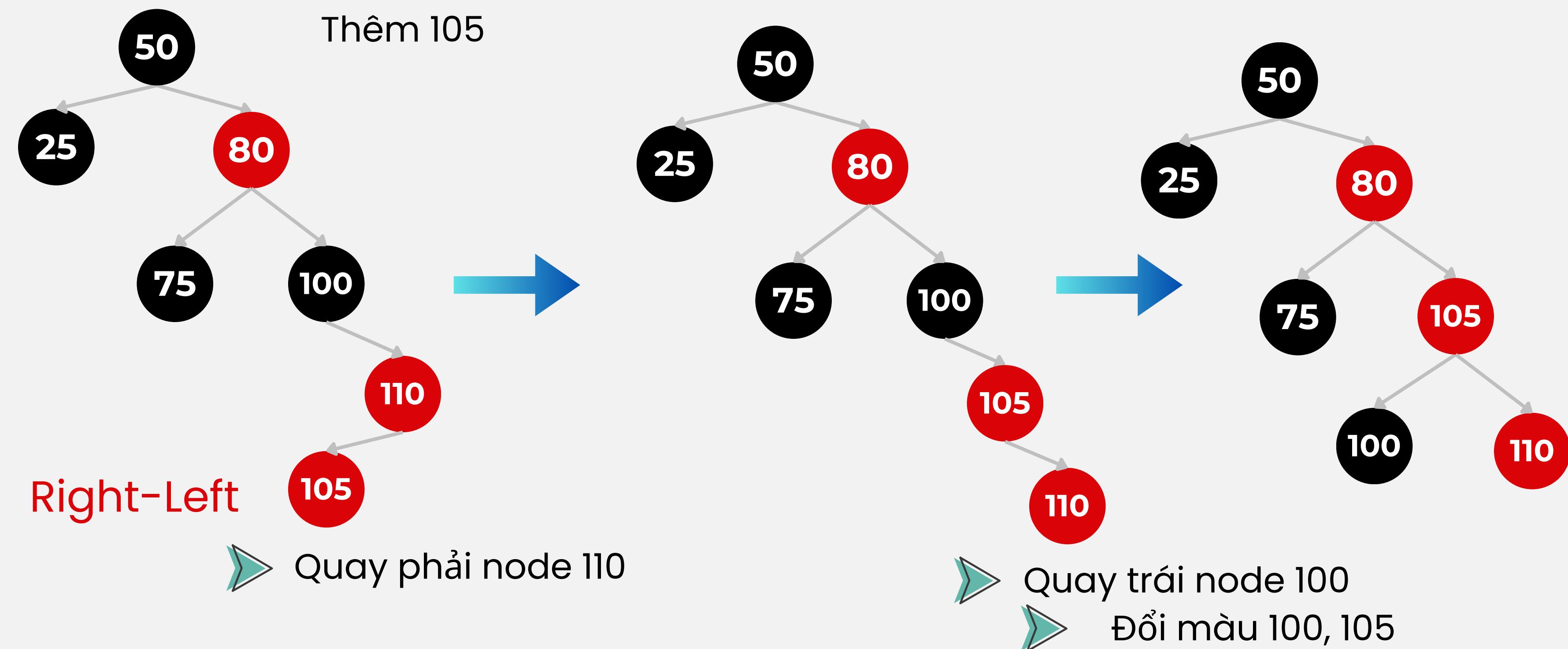


Đổi 75 và 100 thành đen

Đổi 80 thành đỏ

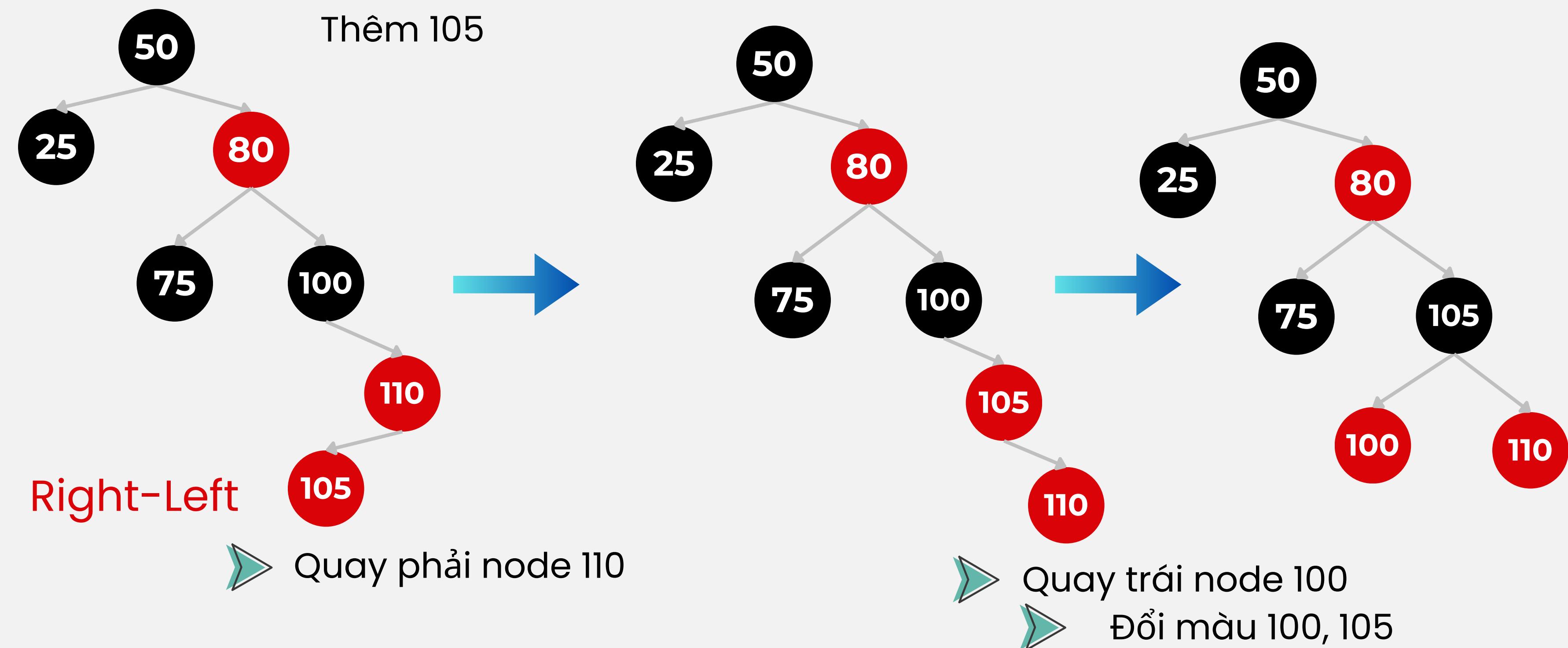
Ví dụ: Tạo cây đỏ đen

Tạo cây đỏ đen từ dãy số sau đây: 50, 75, 25, 80 , 100, 110, 105



Ví dụ: Tạo cây đỏ đen

Tạo cây đỏ đen từ dãy số sau đây: 50, 75, 25, 80 , 100, 110, 105



Phép xóa

Việc xóa một node X của cây đỏ đen được thực hiện theo các bước sau:

1. Tìm kiếm node cần xóa. Nếu không tìm thấy thì dừng
2. Nếu tìm thấy, ta tiến hành xóa
 - Node cần xóa là **nút lá**
 - Node cần xóa là **nút 1 con**
 - Node cần xóa là **nút 2 con** (tìm node thay thế)
3. Điều chỉnh lại nếu vi phạm các qui định của cây đỏ đen

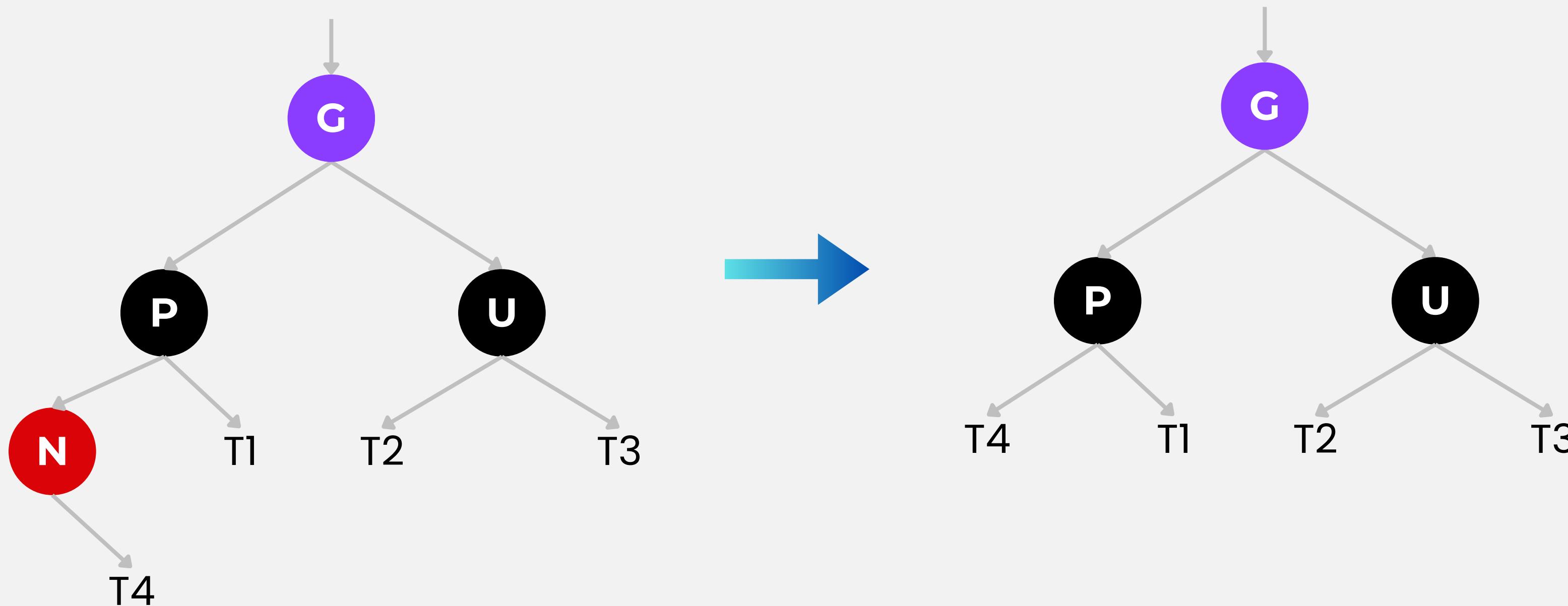
Phép xóa

Các trường hợp có thể xảy ra khi xóa node X của cây đỏ đen

1. Node cần xóa là node đỏ
2. Node cần xóa là node đen
 - Node con của X là node đỏ (có 1 con)
 - 2 con của X (kể cả code NIL) đều là node đen

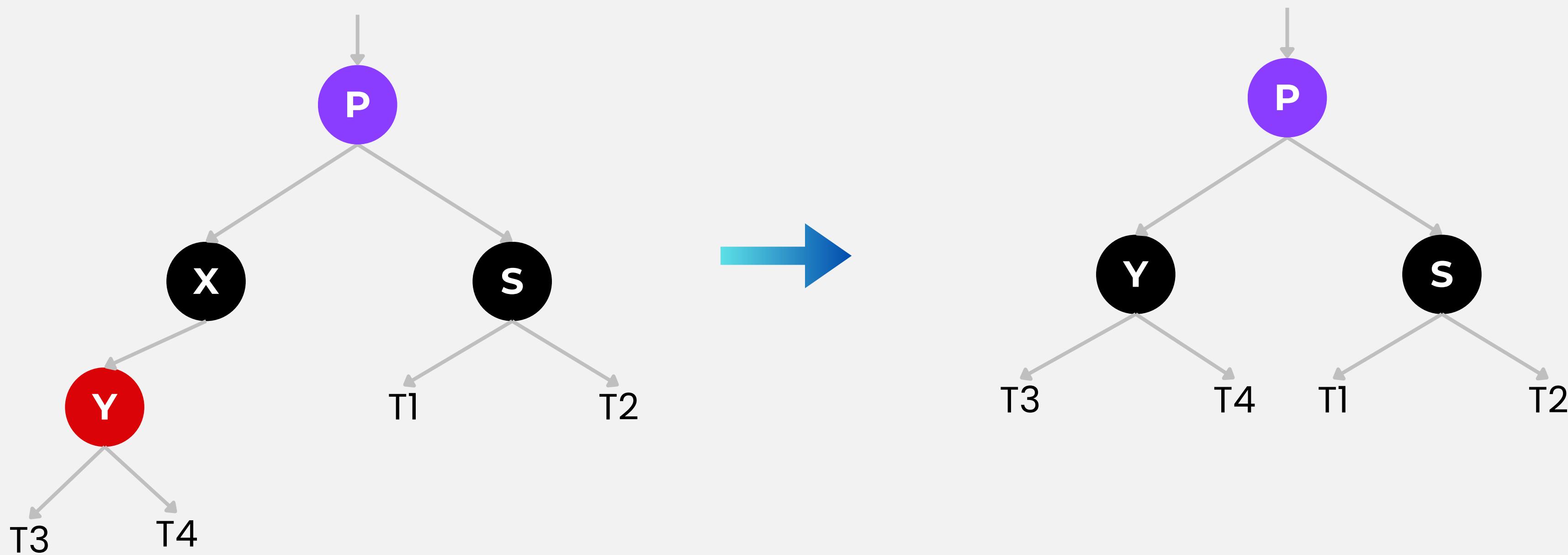
Phép xóa

X màu đỏ hoặc nút gốc: Việc node X sẽ không ảnh hưởng tới đặc điểm và tính chất của cây đỏ đen nên không cần điều chỉnh



Phép xóa

X là node màu **đen** và có 1 con là node **đỏ**: Điều chỉnh màu của node con của X thành màu **đen**

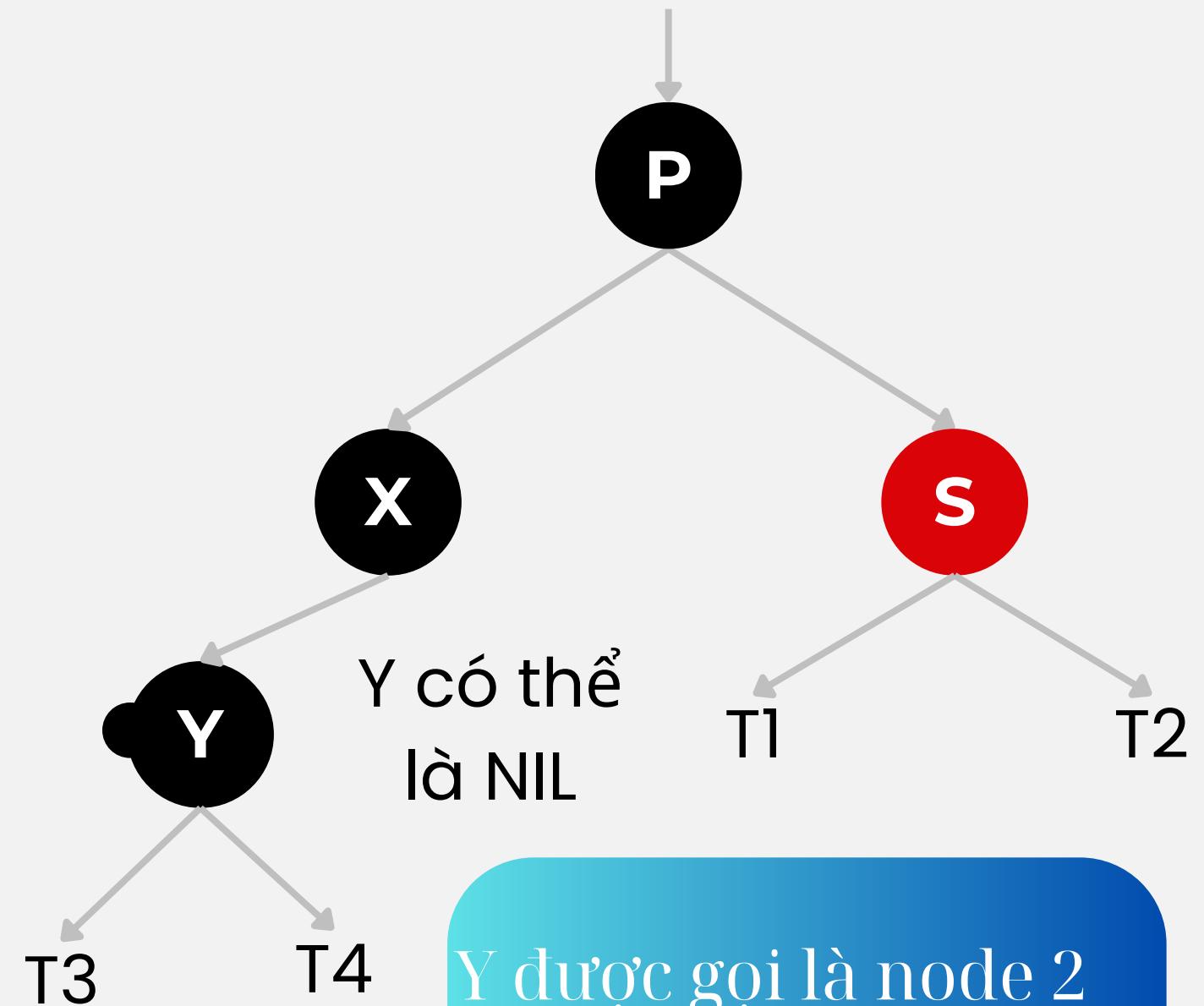


Phép xóa

X là node đen và có 2 con là node đen. Gọi S là node anh em của X, có 4 trường hợp xảy ra

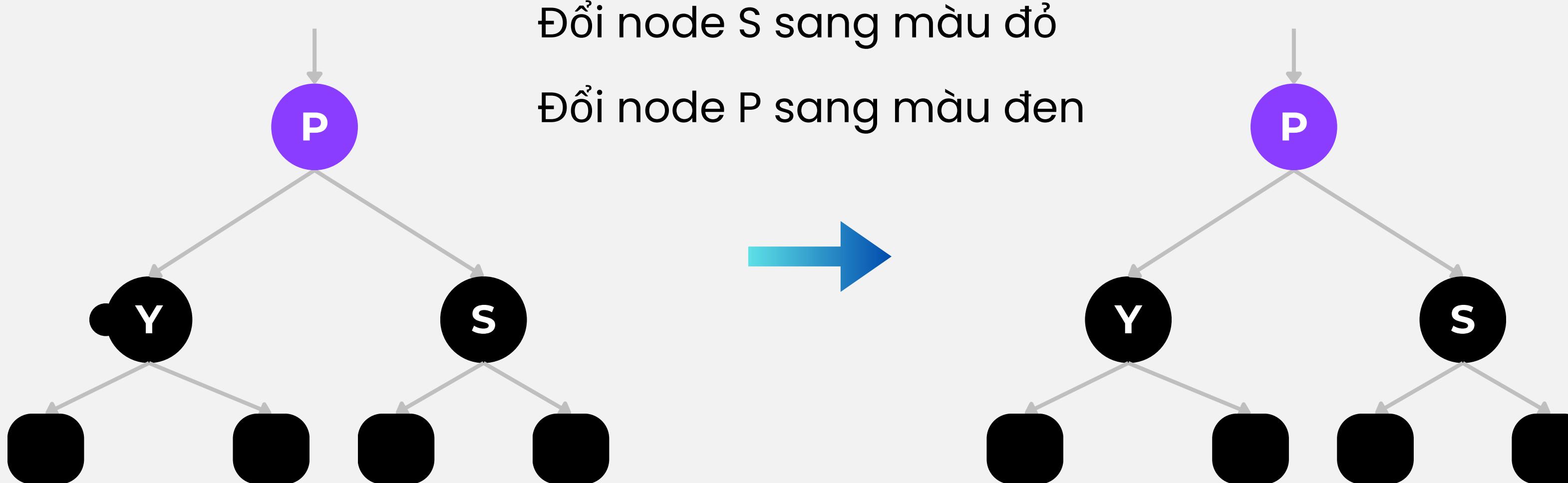
1. S màu đen và 2 con của S màu đen
2. S màu đen và con phải của S màu đỏ
3. S màu đen và con trái của S màu đỏ
4. S màu đỏ

Tất cả các đường đi qua node X sẽ bị ít hơn 1 node đen



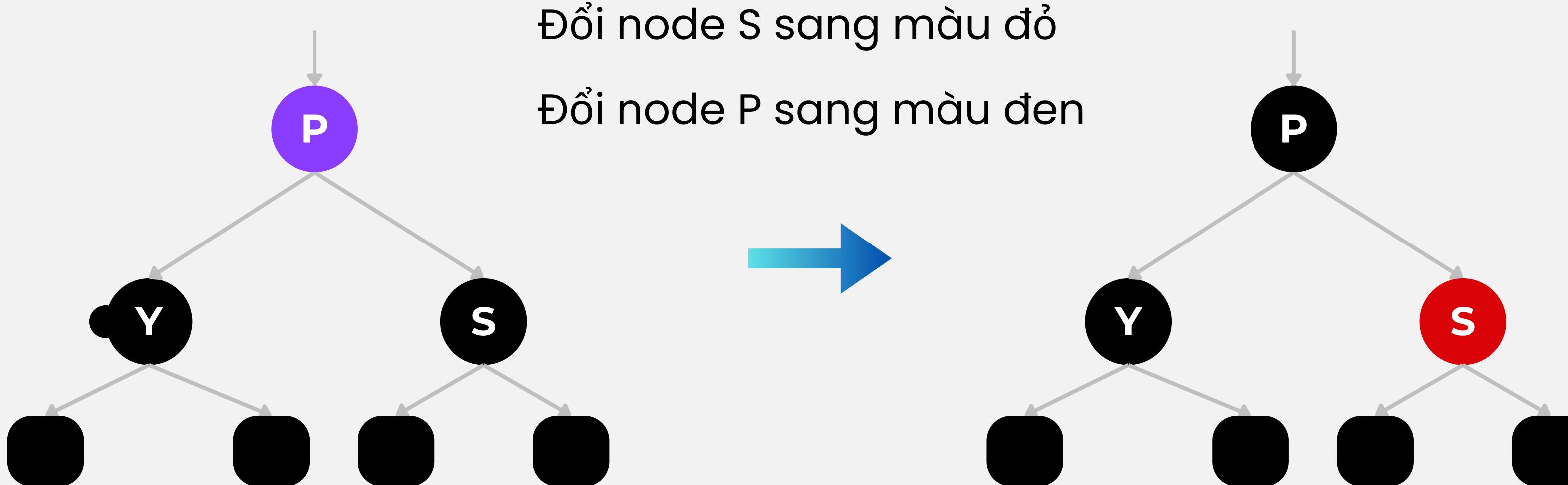
Phép xóa

TH1: X là node đen, X có 2 con là node đen và S màu đen và 2 con của S màu đen



Phép xóa

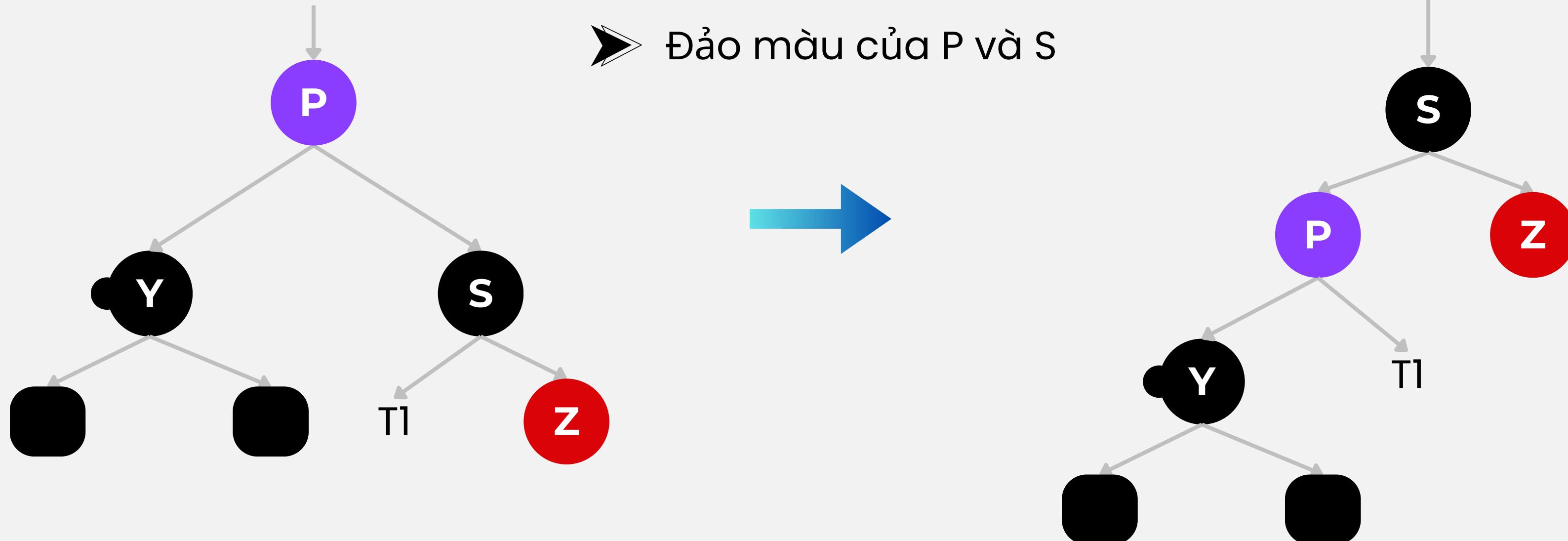
TH1: X là node đen, X có 2 con là node đen và S màu đen và 2 con của S màu đen



Phép xóa

TH2: X là node đen, X có 2 con là node đen, S màu đen
và con phải của S màu đỏ

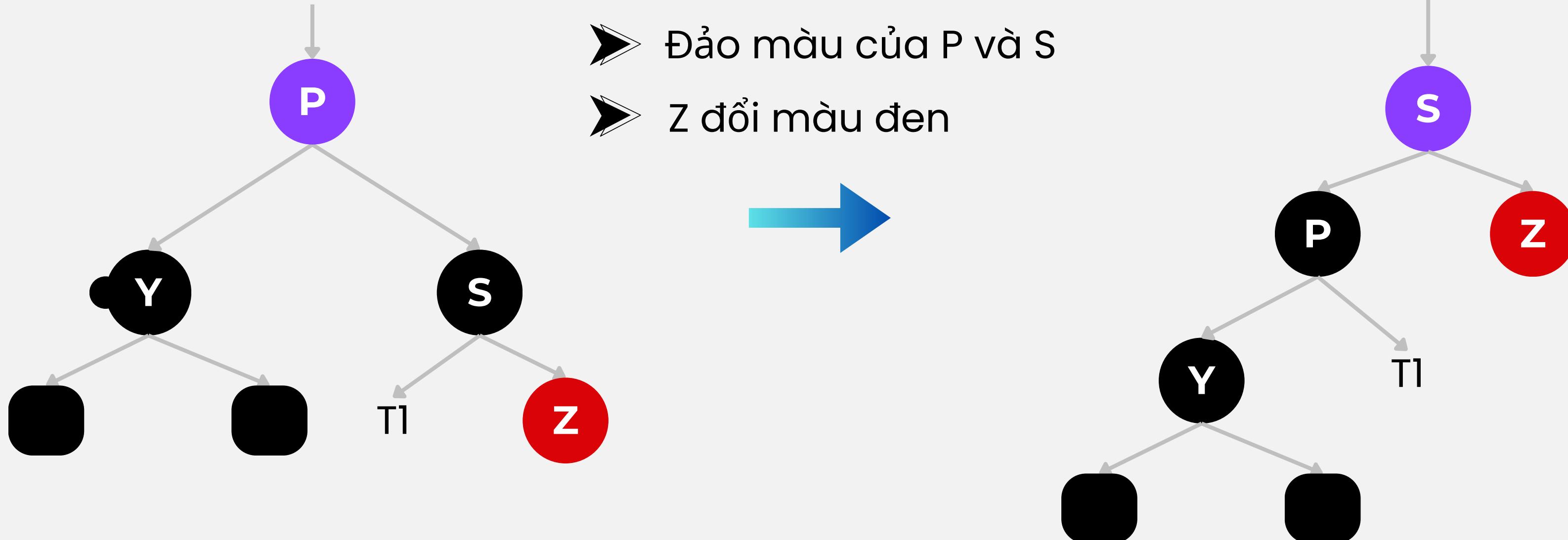
- Quay trái tại P
- Đảo màu của P và S



Phép xóa

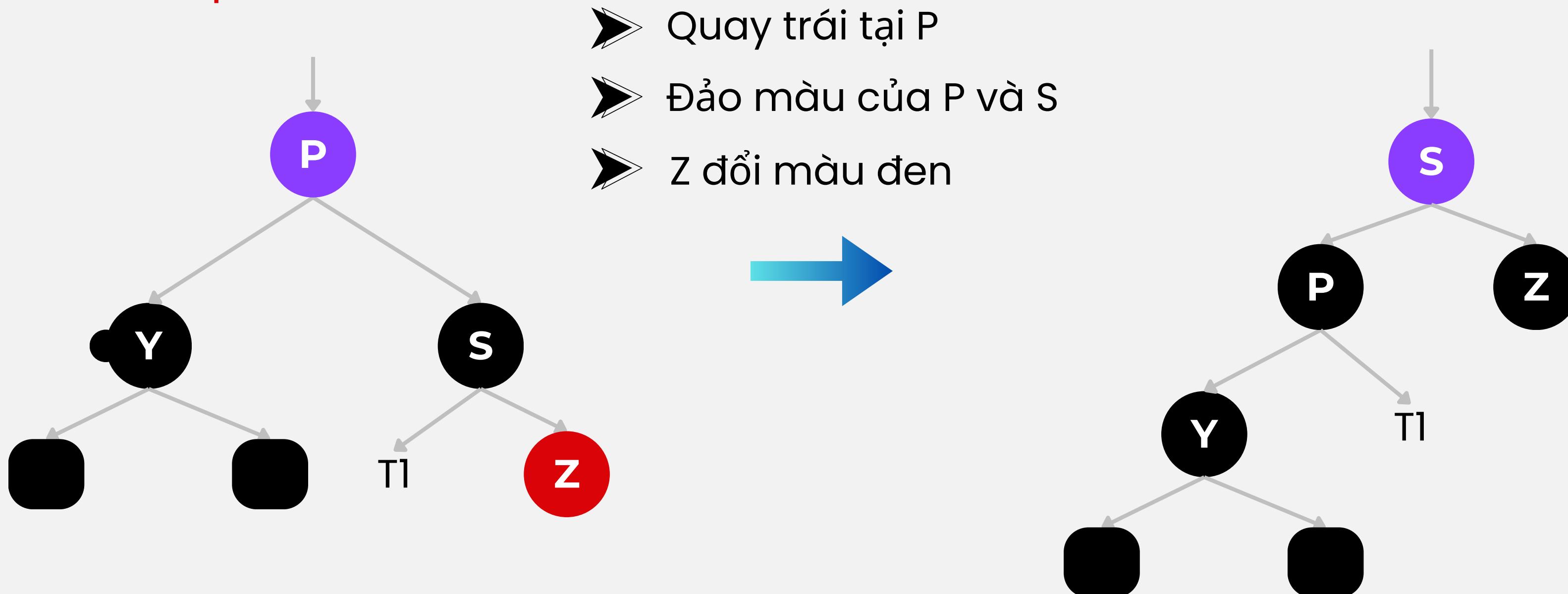
TH2: X là node đen, X có 2 con là node đen, S màu đen
và con phải của S màu đỏ

- Quay trái tại P
- Đảo màu của P và S
- Z đổi màu đen



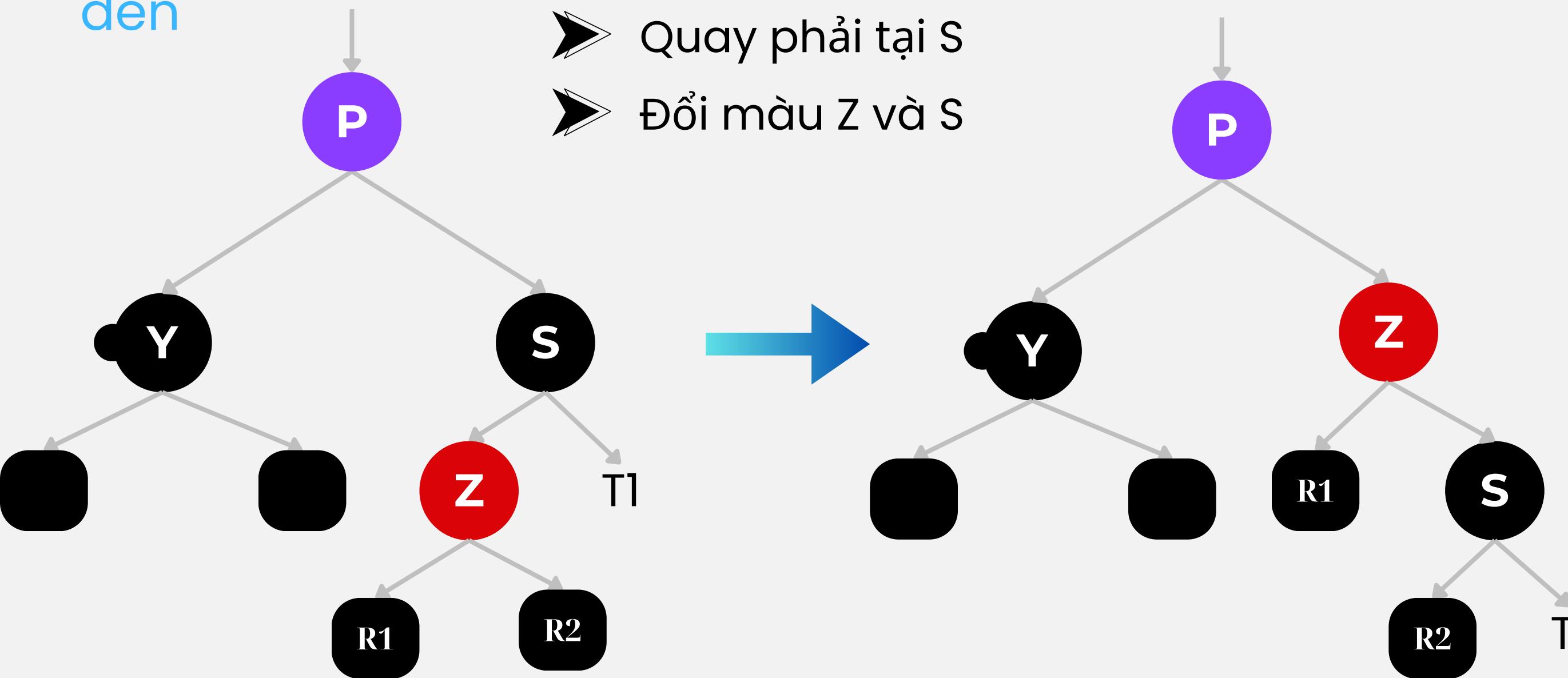
Phép xóa

TH2: X là node đen, X có 2 con là node đen, S màu đen
và con phải của S màu đỏ



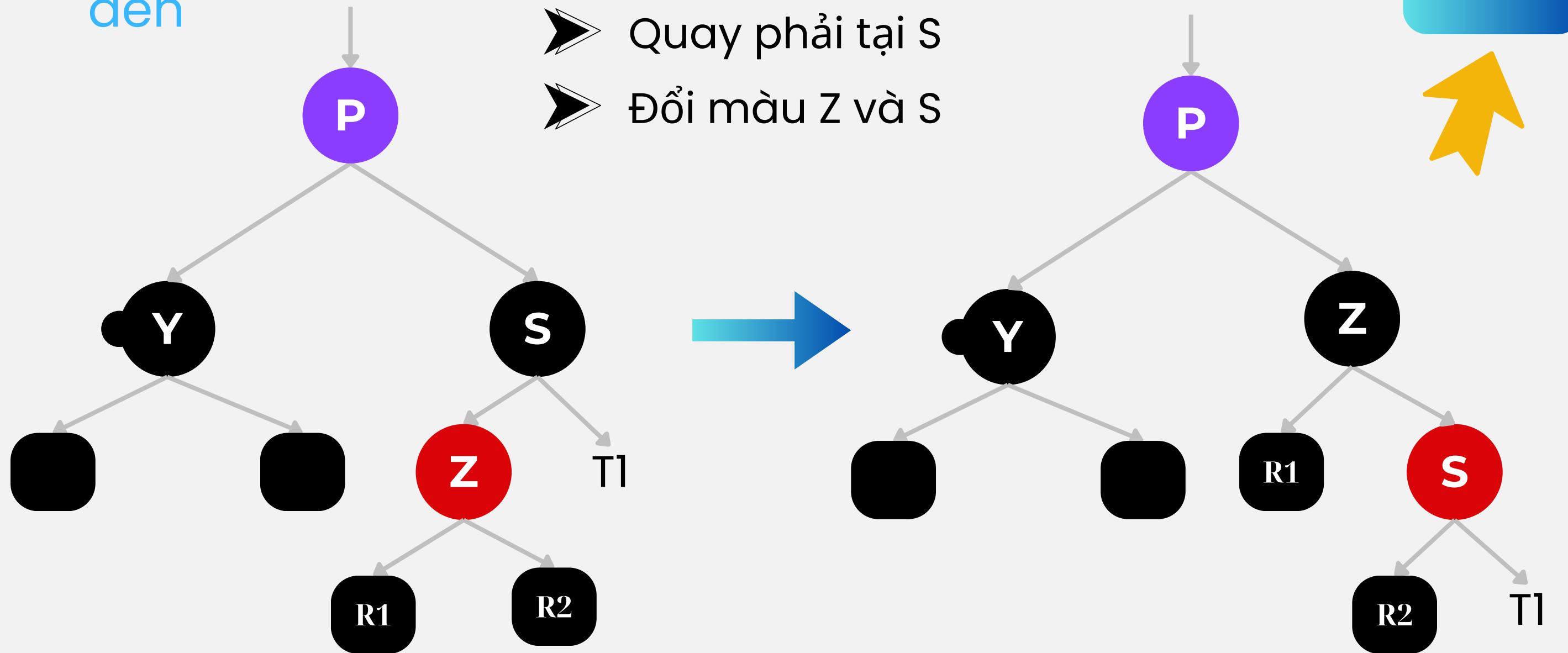
Phép xóa

TH3: X là node đen, X có 2 con là node đen, S màu
đen và con trái của S màu đỏ, con phải của S màu
đen



Phép xóa

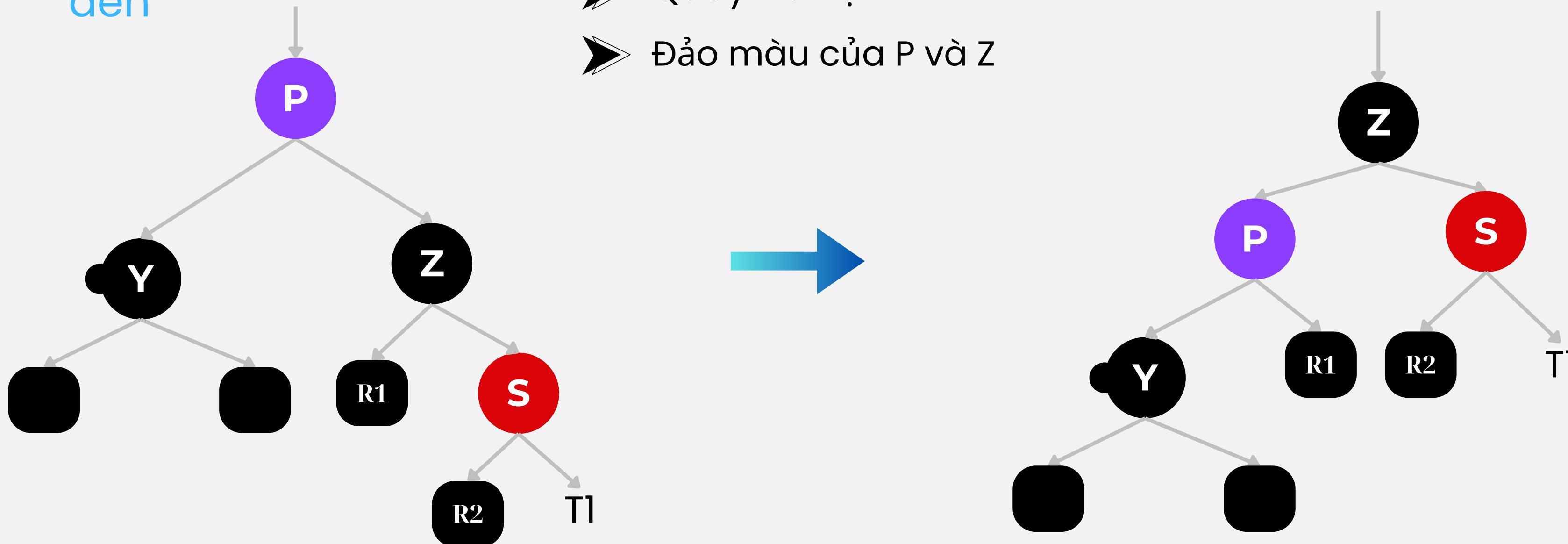
TH3: X là node đen, X có 2 con là node đen, S màu
đen và con trái của S màu đỏ, con phải của S màu
đen



Phép xóa

TH3: X là node đen, X có 2 con là node đen, S màu
đen và con trái của S màu đỏ, con phải của S màu
đen

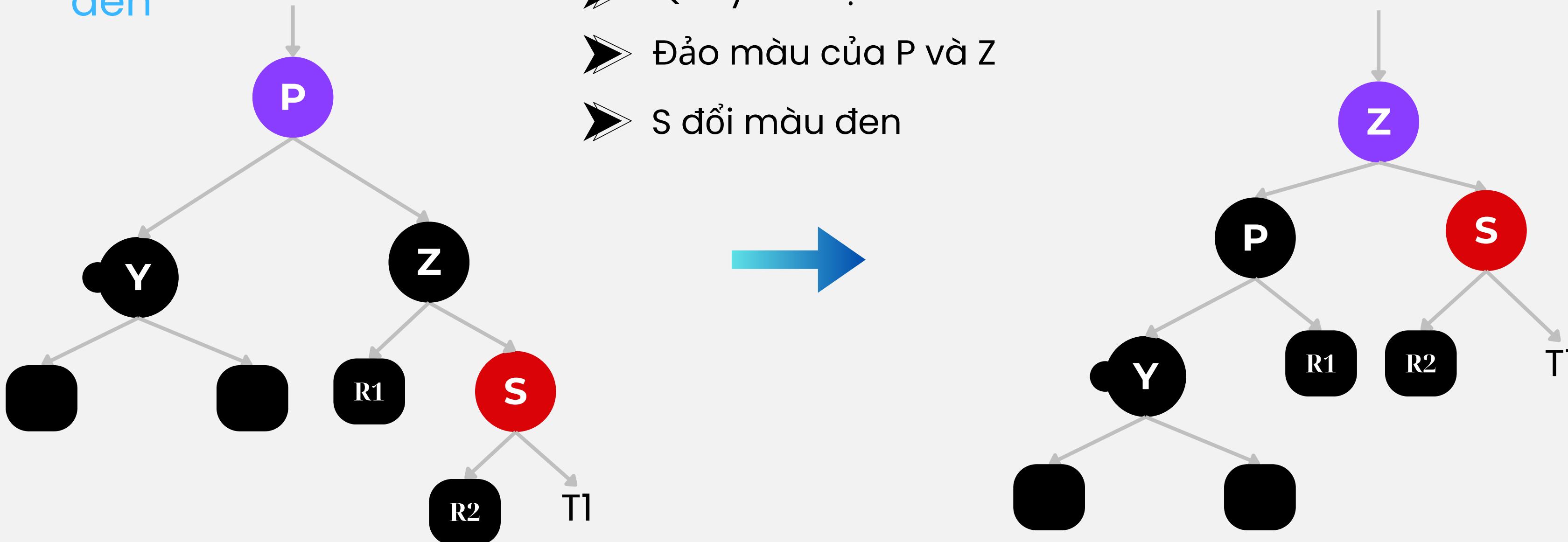
- Quay trái tại P
- Đảo màu của P và Z



Phép xóa

TH3: X là node đen, X có 2 con là node đen, S màu
đen và con trái của S màu đỏ, con phải của S màu
đen

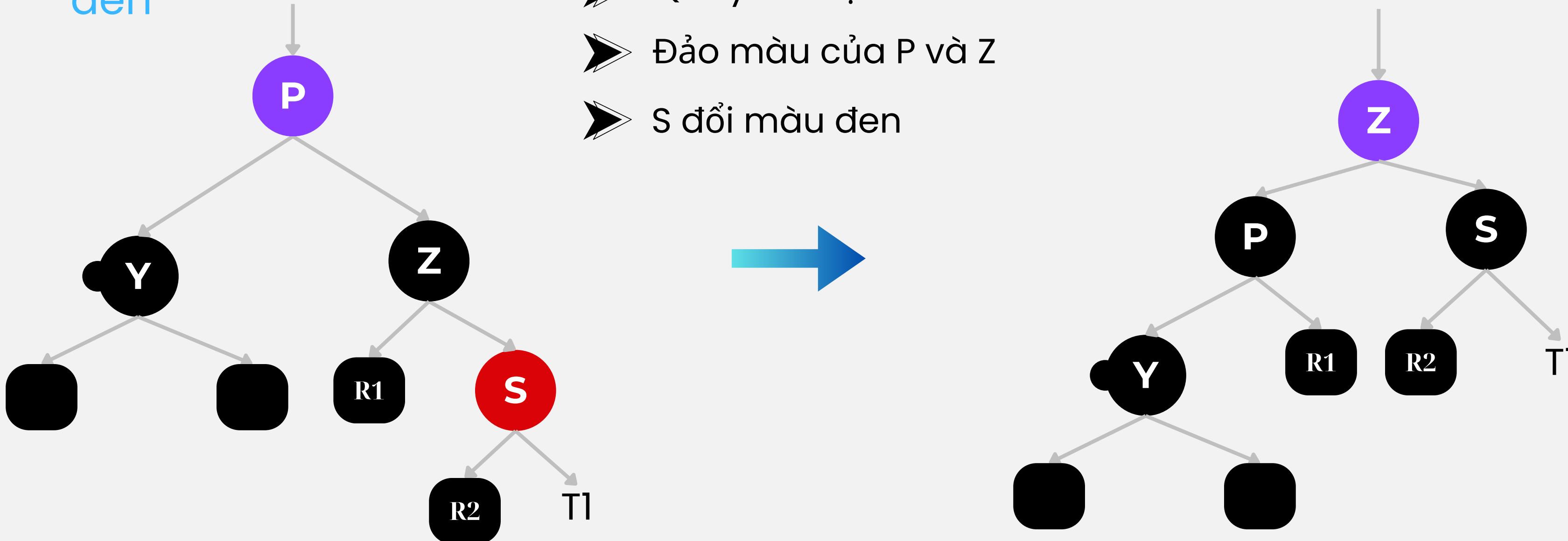
- Quay trái tại P
- Đảo màu của P và Z
- S đổi màu đen



Phép xóa

TH3: X là node đen, X có 2 con là node đen, S màu
đen và con trái của S màu đỏ, con phải của S màu
đen

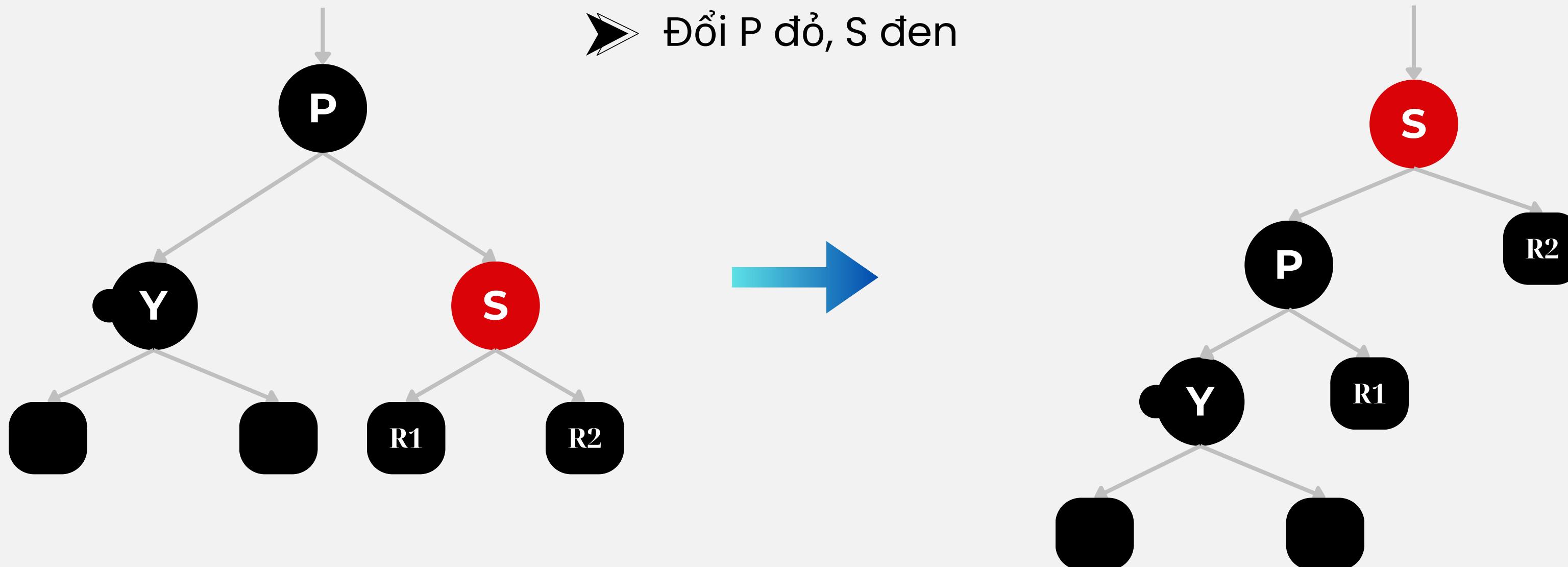
- Quay trái tại P
- Đảo màu của P và Z
- S đổi màu đen



Phép xóa

TH4: X là node đen, X có 2 con là node đen, S màu đỏ.

- Quay trái tại P
- Đổi P đỏ, S đen



Phép xóa

TH4: X là node đen, X có 2 con là node đen, S màu đỏ.

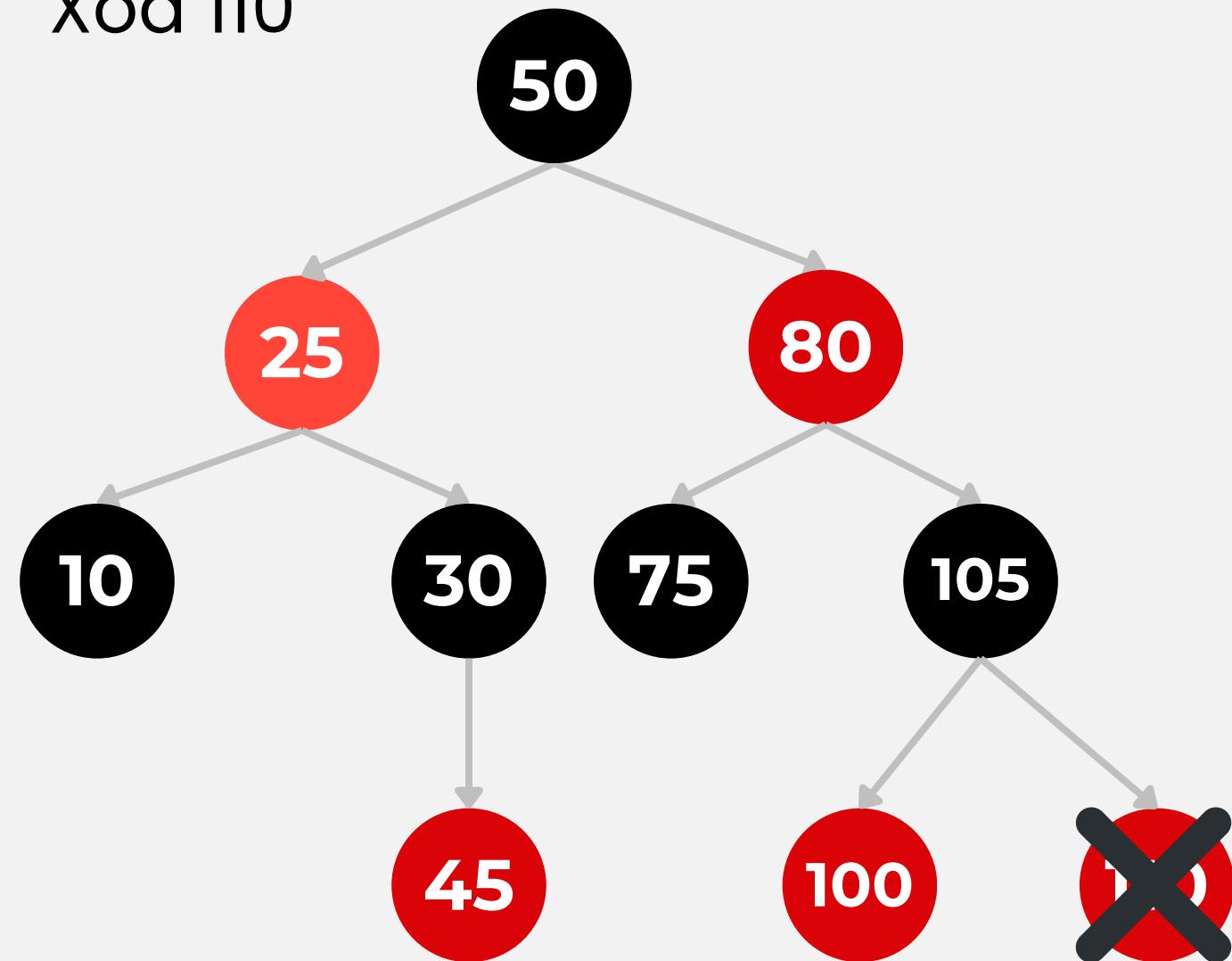
- Quay trái tại P
- Đổi P đỏ, S đen
- Dấu hiệu đen vẫn ở Y



Ví dụ: Xóa nút

Xóa node 110, 105, 45, 50, 30

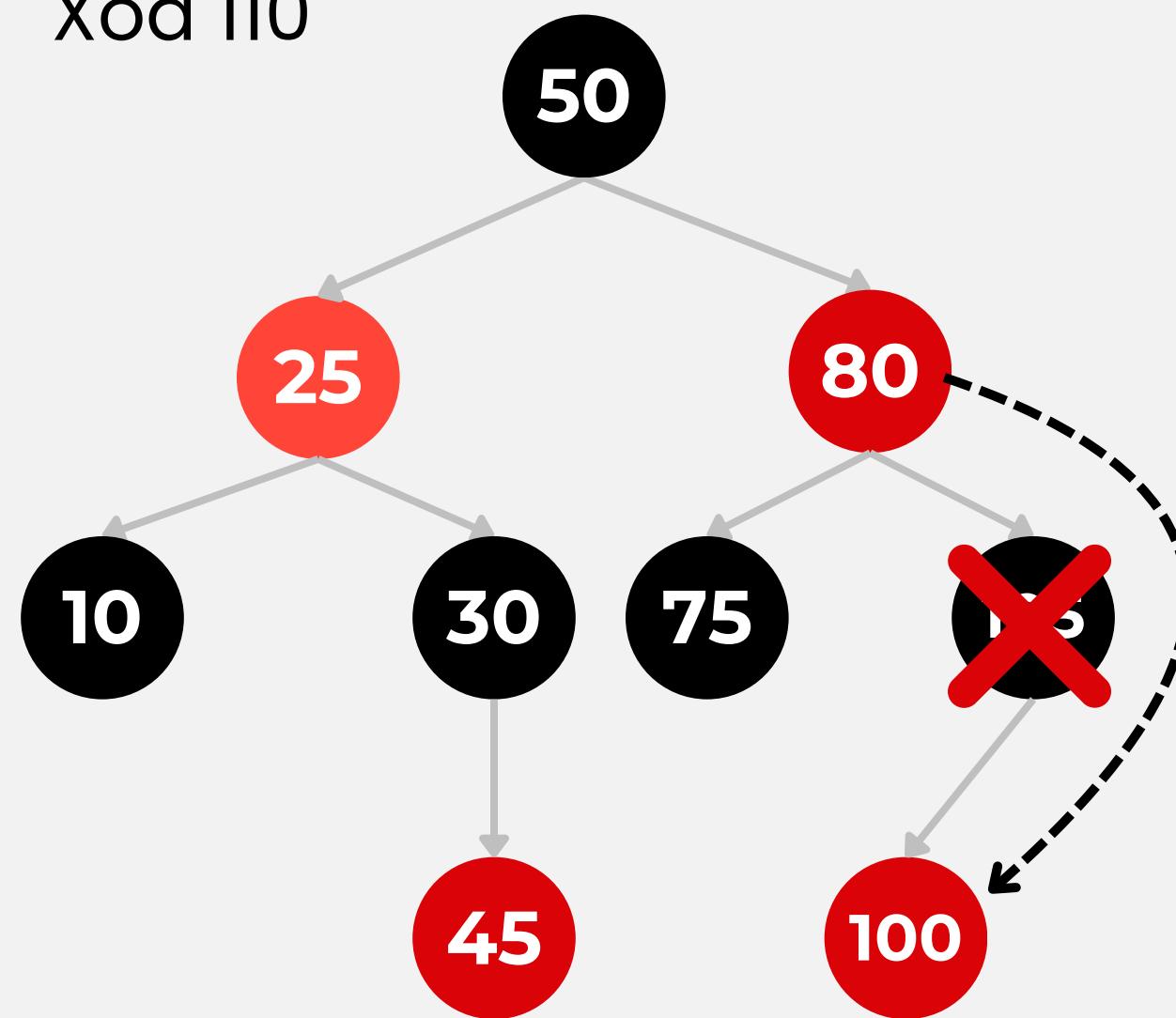
Xóa 110



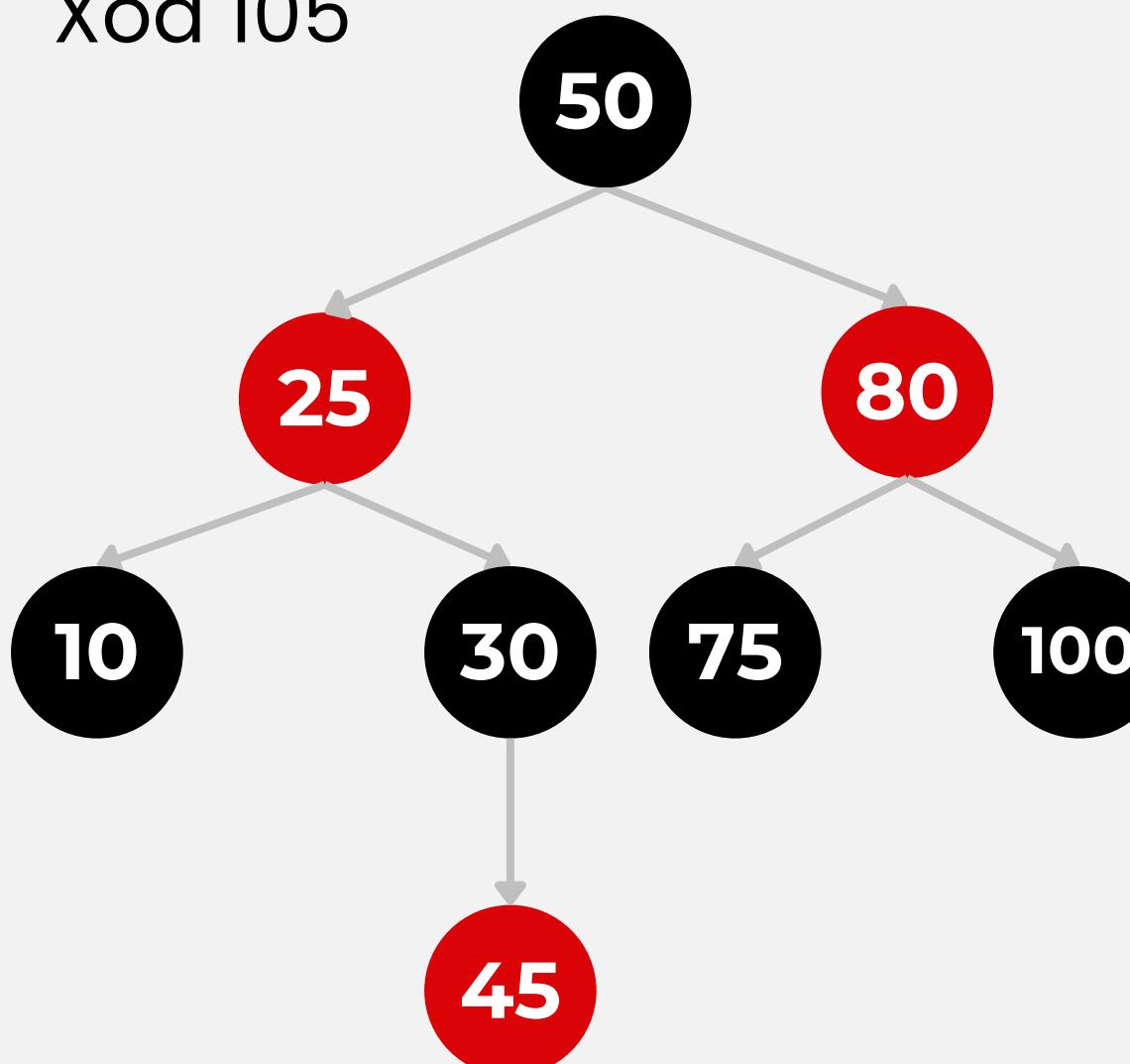
Ví dụ: Xóa nút

Xóa node 110, 105, 45, 50, 30

Xóa 110



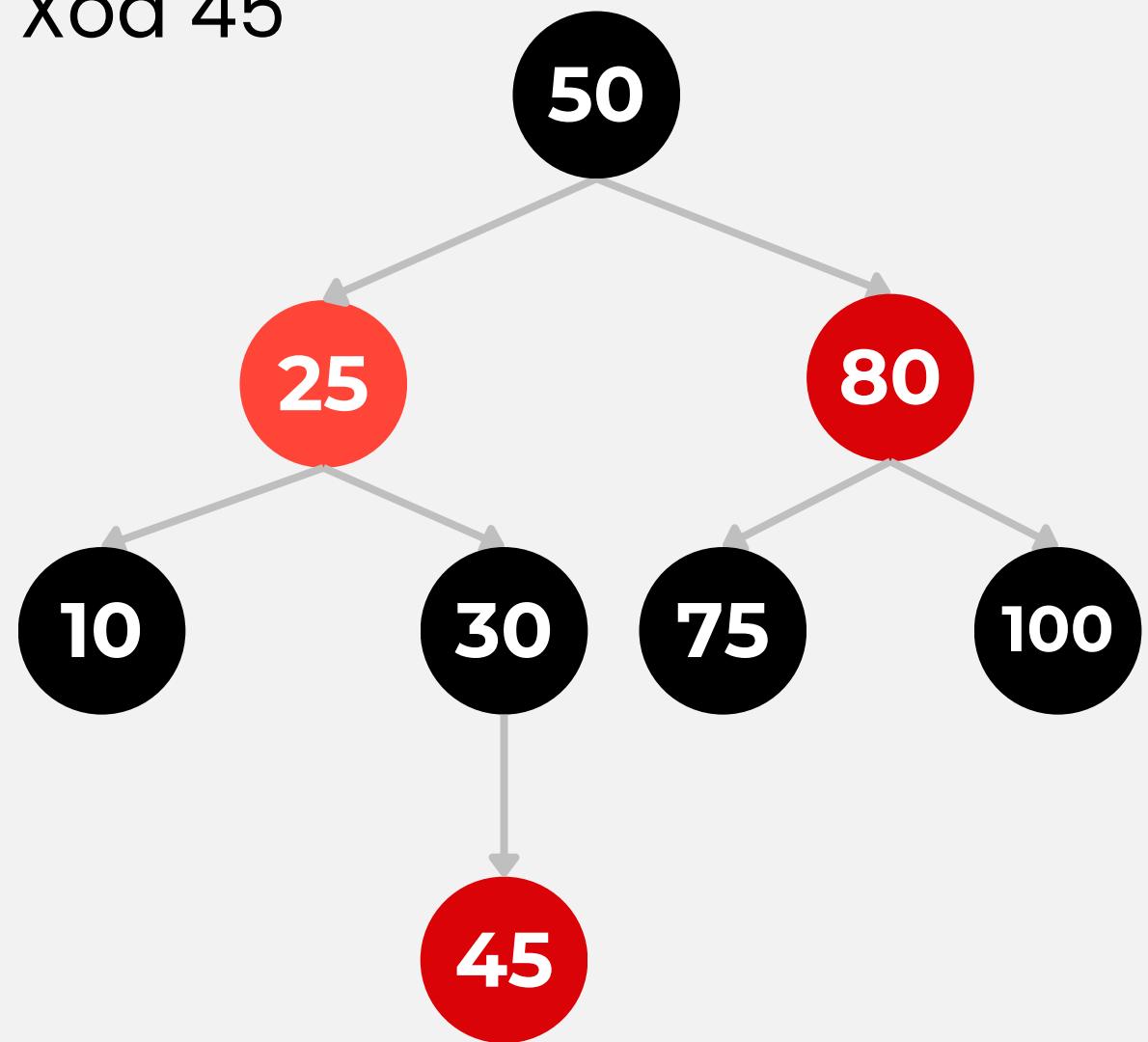
Xóa 105



Ví dụ: Xóa nút

Xóa node 110, 105, 45,50,30

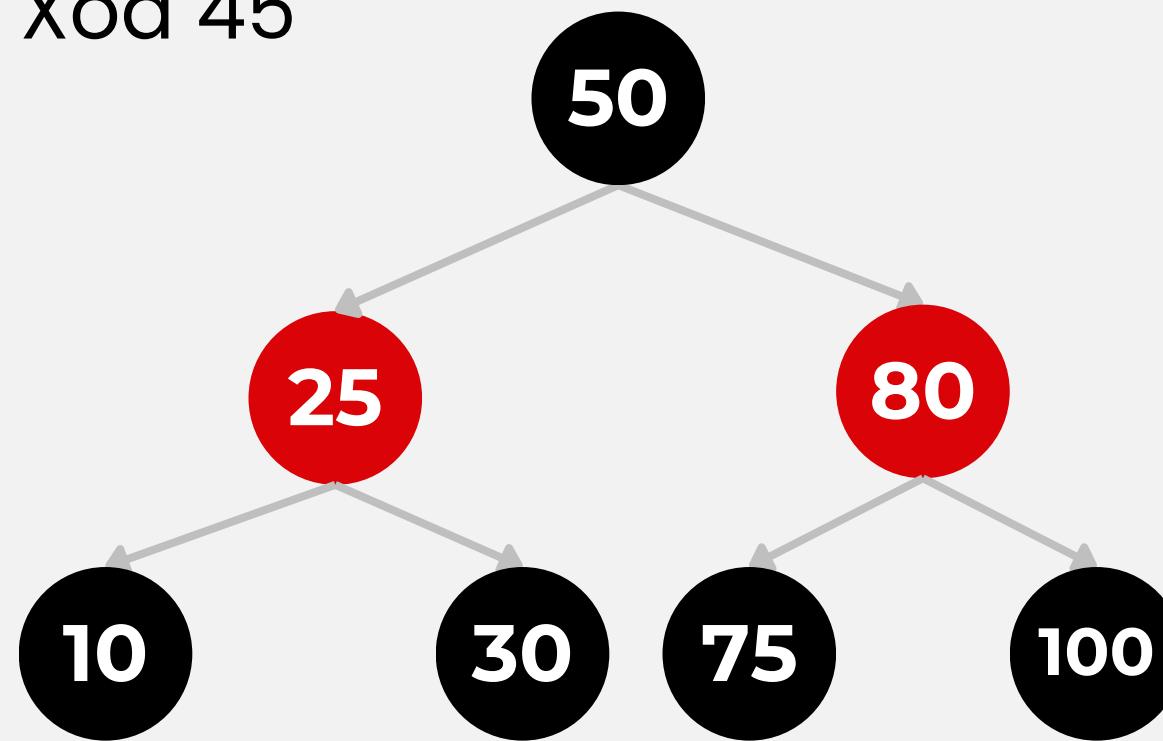
Xóa 45



Ví dụ: Xóa nút

Xóa node 110, 105, 45, 50, 30

Xóa 45

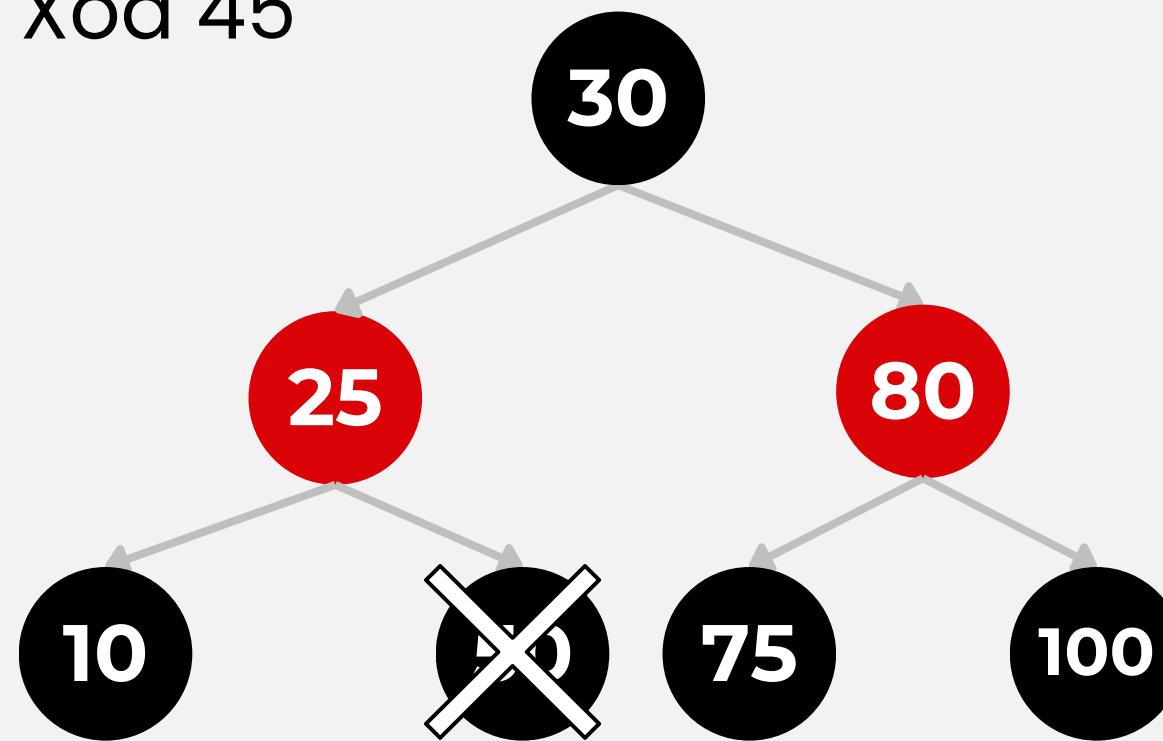


Xóa 50
Node 50 là node có 2 con
➤ Tìm node thay thế
(node lớn nhất của cây con bên trái)

Ví dụ: Xóa nút

Xóa node 110, 105, 45,50,30

Xóa 45

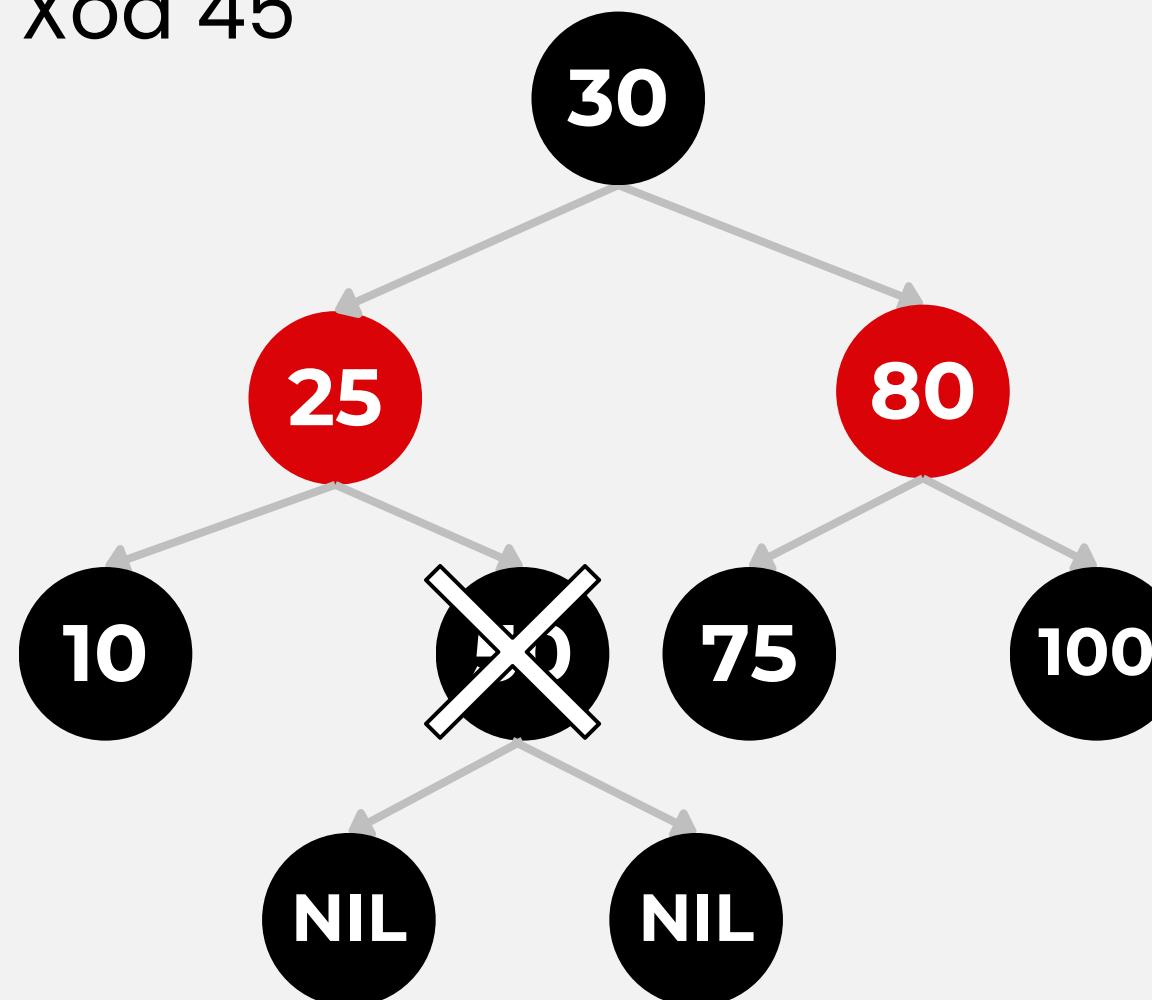


Xóa 50
Node 50 là node có 2 con
➤ Tìm node thay thế
(node lớn nhất của cây con bên trái)

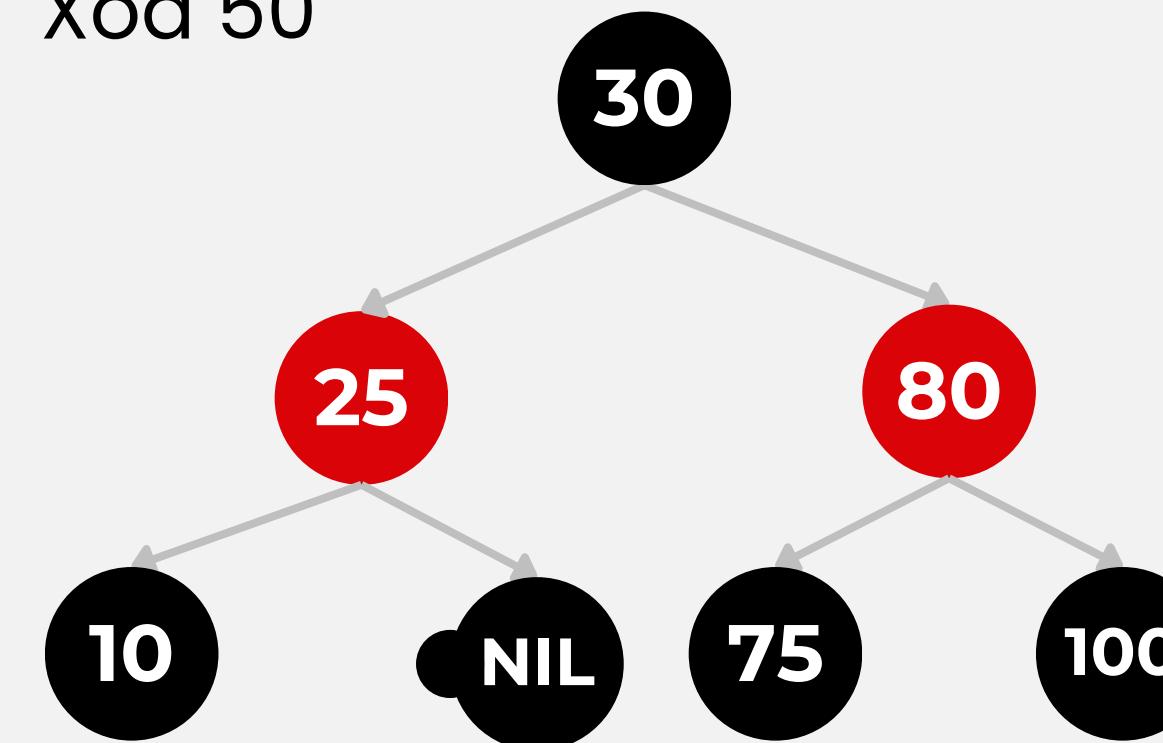
Ví dụ: Xóa nút

Xóa node 110, 105, 45,50,30

Xóa 45



Xóa 50



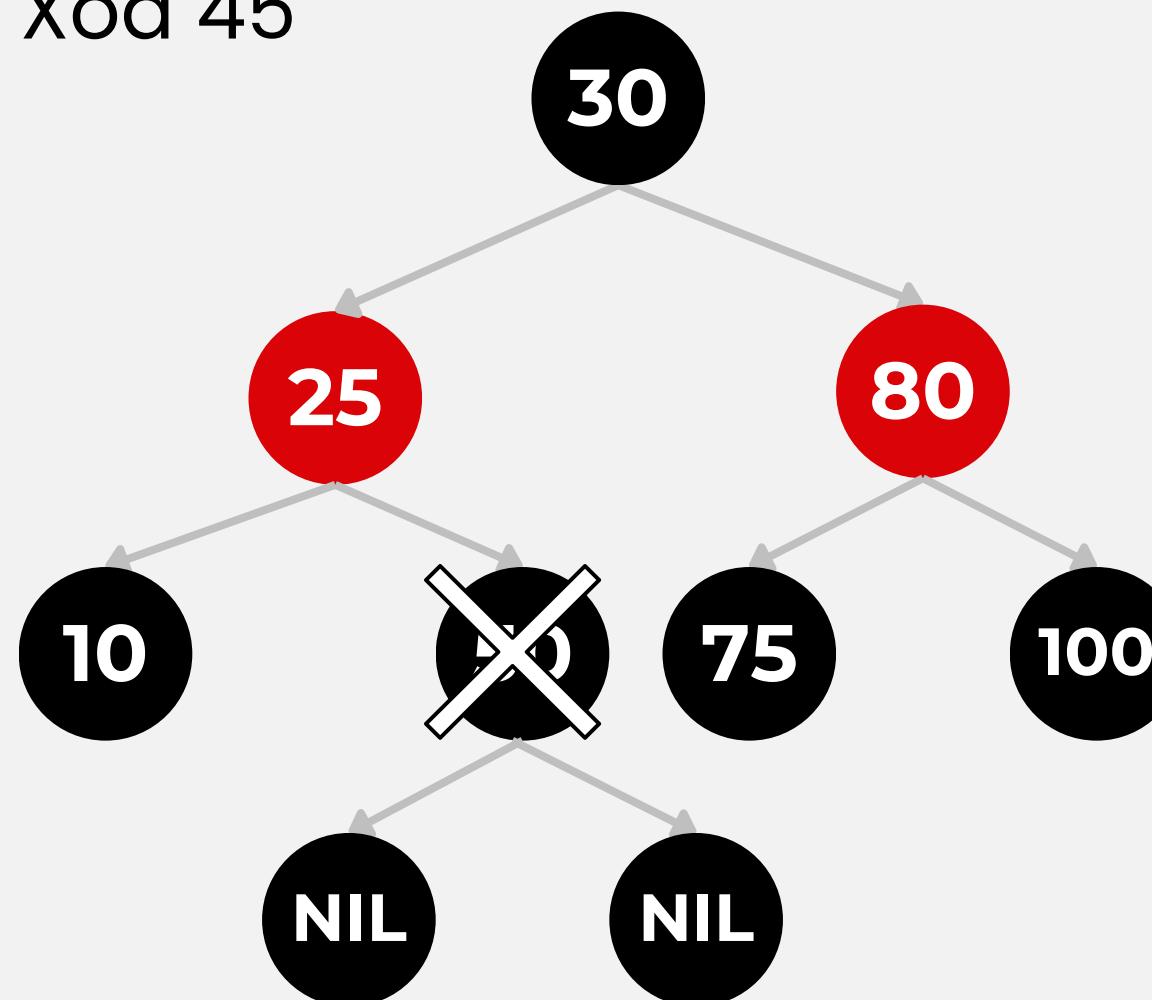
**Node cần xóa có 2 con màu
đen và node anh em có
màu đen**

Đổi node 10 sang màu đỏ
Đổi 25 sang màu đen

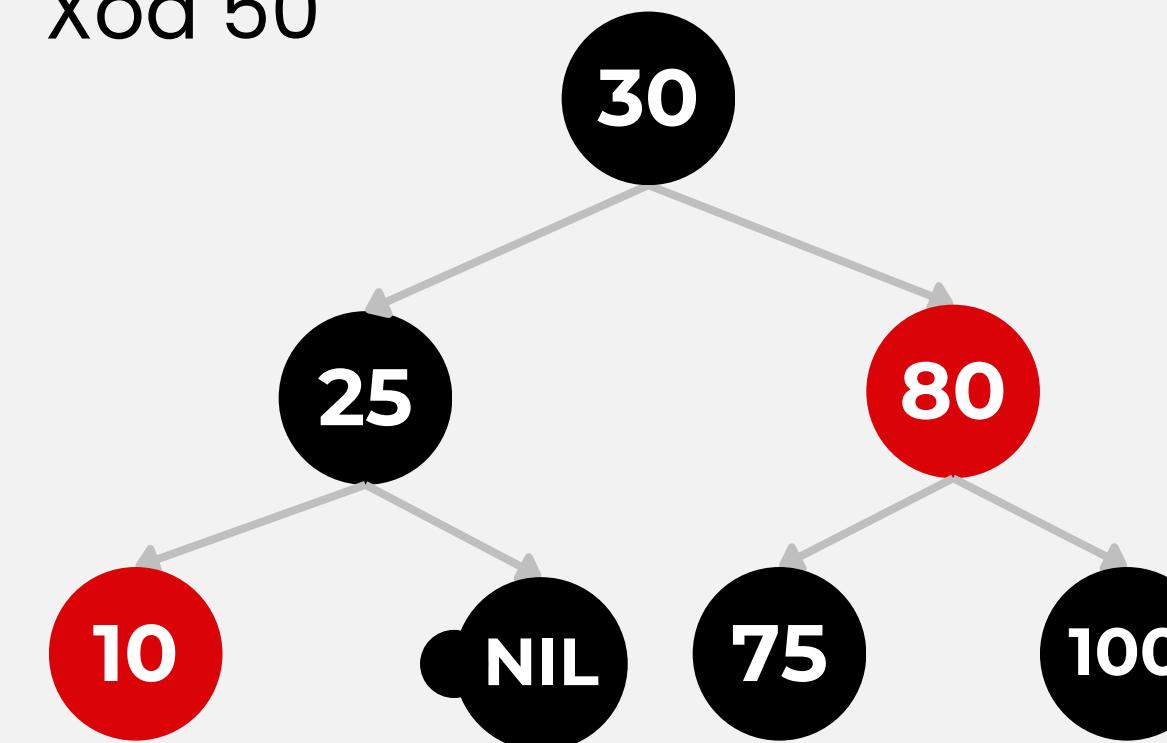
Ví dụ: Xóa nút

Xóa node 110, 105, 45,50,30

Xóa 45



Xóa 50



**Node cần xóa có 2 con màu
đen và node anh em có
màu đen**

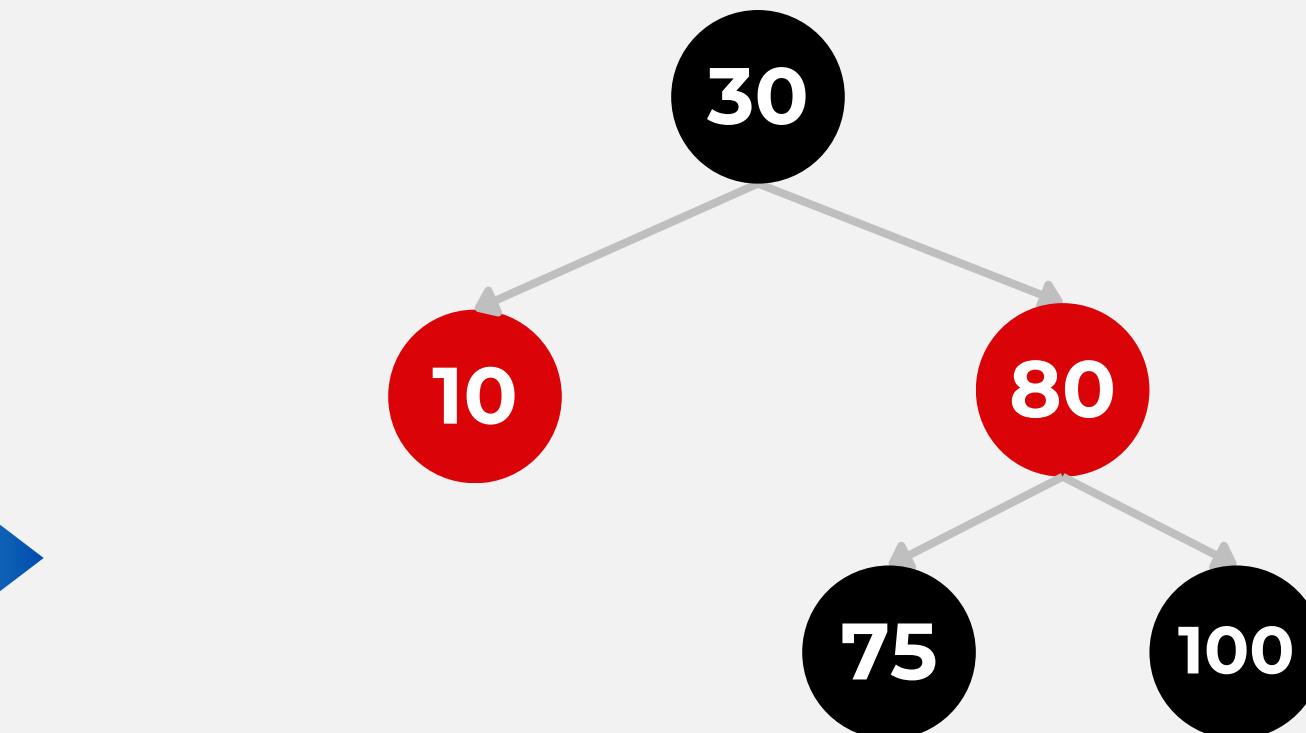
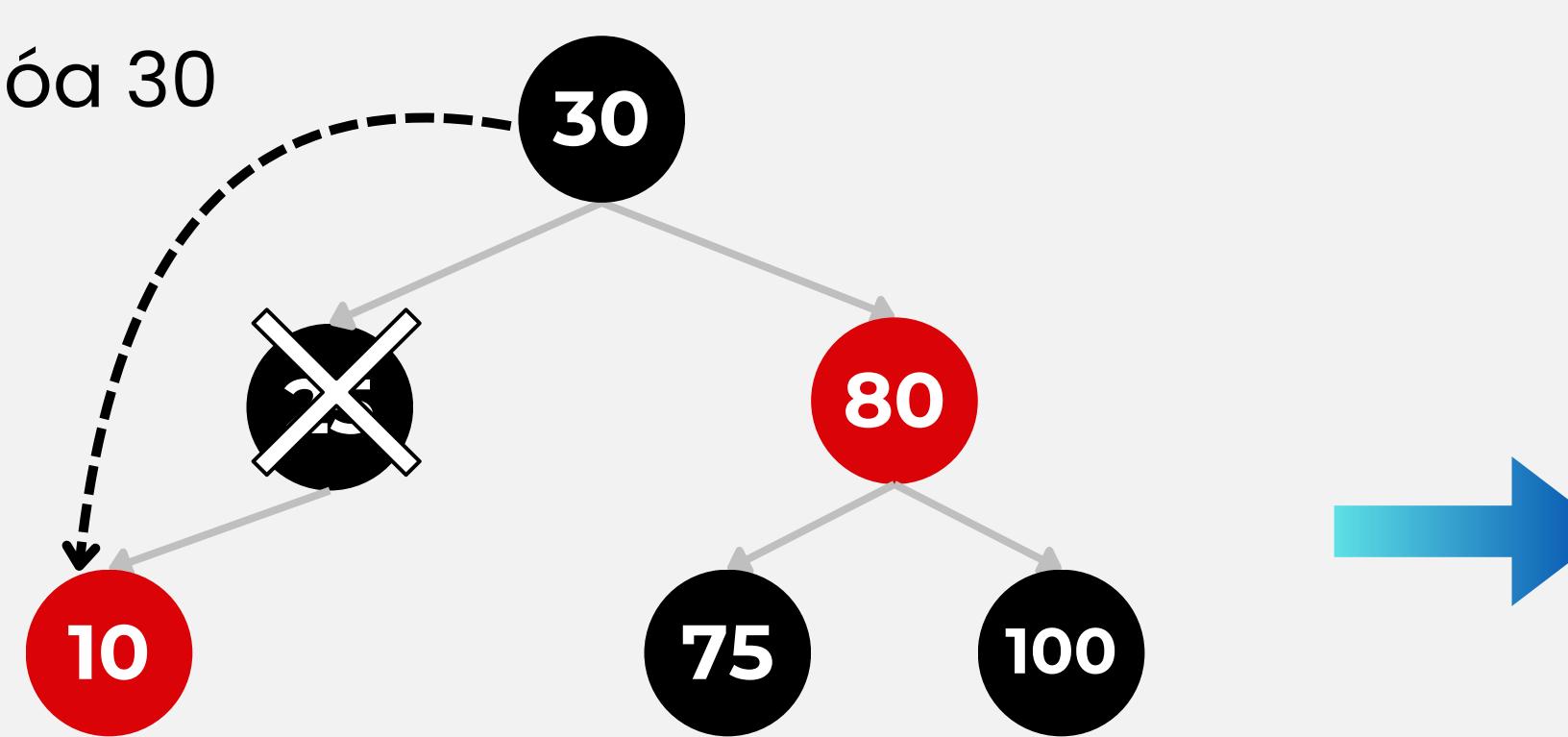
Đổi node 10 sang màu đỏ

Đổi 25 sang màu đen

Ví dụ: Xóa nút

Xóa node 110, 105, 45,50,30

Xóa 30

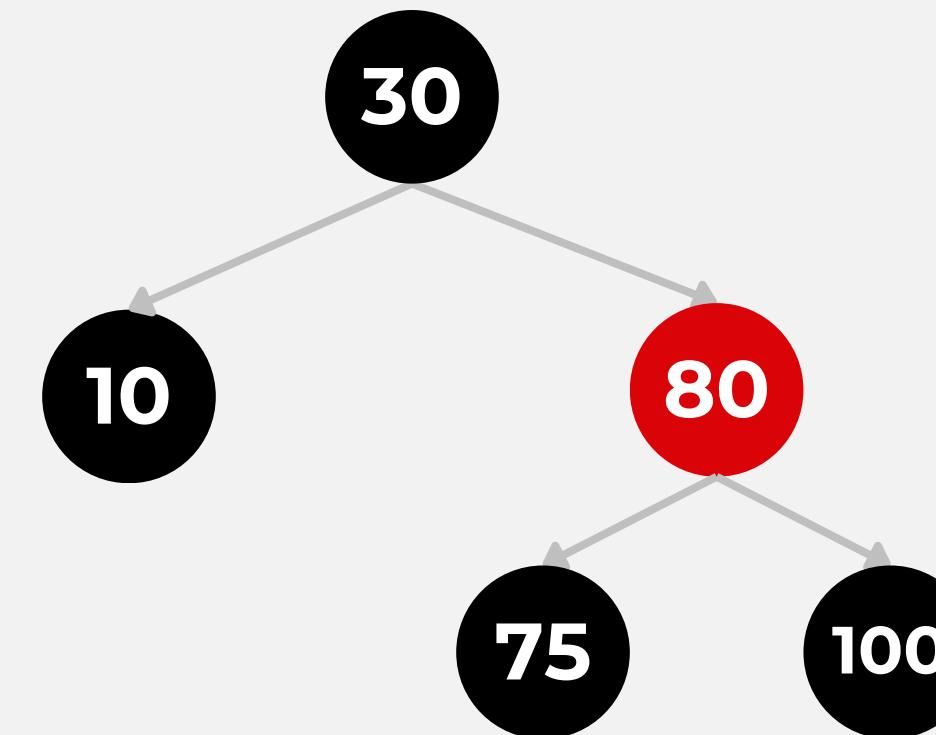
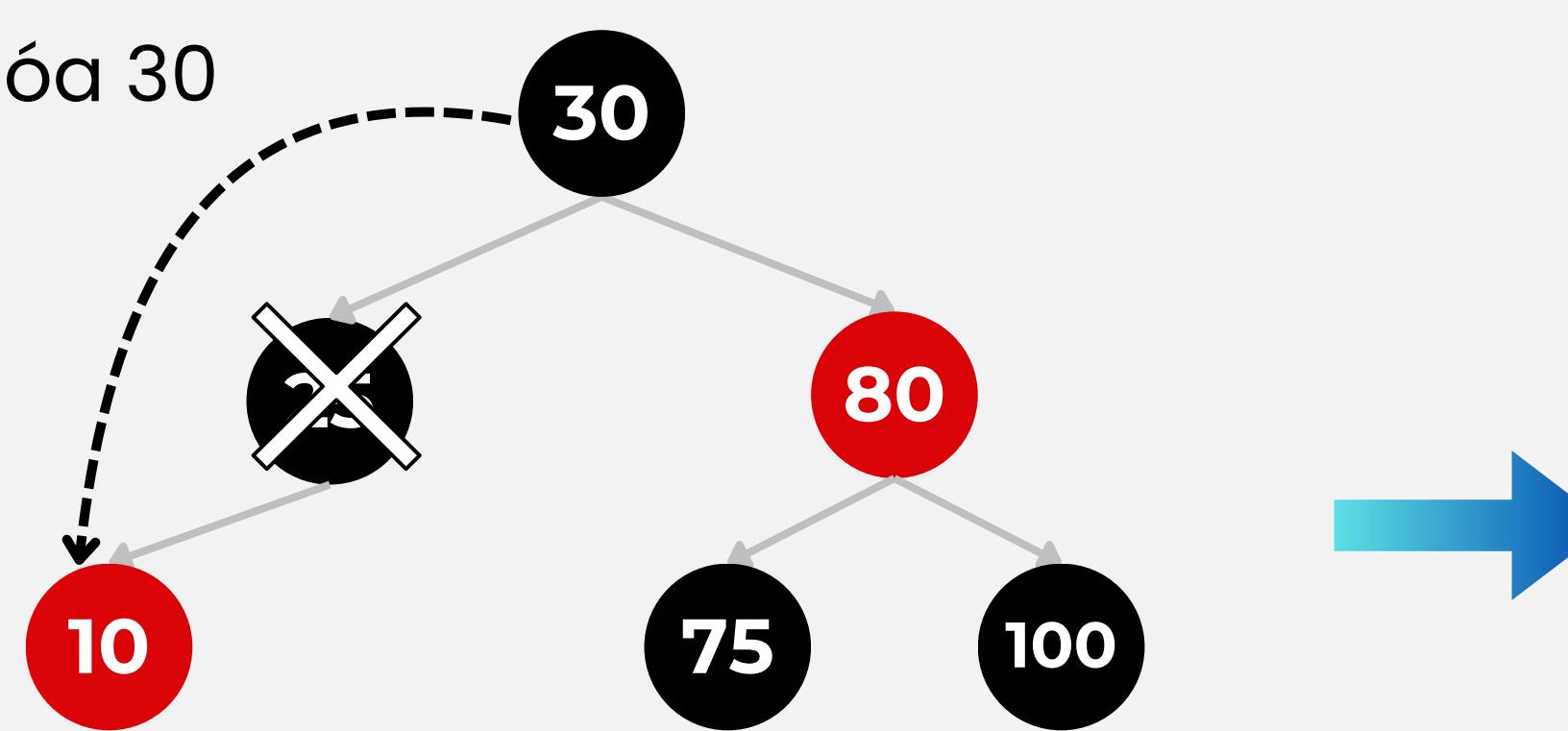


➤ Đổi nút 10 sang màu đen

Ví dụ: Xóa nút

Xóa node 110, 105, 45,50,30

Xóa 30



➤ Đổi nút 10 sang màu đen

04. So sánh cây AVL và cây đỏ đen

Tiêu chí	AVL Tree	Red-Black Tree
Mức độ cân bằng	Rất chặt chẽ. Luôn đảm bảo hiệu số chiều cao (balance factor) giữa 2 cây con của mỗi node là -1, 0 hoặc 1.	Lỏng hơn. Chỉ đảm bảo số node đen trên mọi đường đi từ gốc đến lá là như nhau.
Chiều cao tối đa	$\leq 1.44 \times \log_2(n)$	$\leq 2 \times \log_2(n)$
Tái cân bằng	Insert cần tối đa 1 rotation (single hoặc double) Delete cần có thể lên đến $O(\log n)$	Insert tối đa 2 rotation Delete tối đa 3 rotation
Tìm kiếm	Nhanh hơn nhờ chiều cao thấp $O(\log n)$	Chậm hơn chút do chiều cao lớn hơn $O(\log n)$
Độ phức tạp	Phức tạp. Cần quản lý balance factor (-1, 0, 1) và xử lý 4 trường hợp quay (LL, LR, RR, RL). Đặc biệt phức tạp khi xóa node.	Cũng phức tạp, nhưng việc xóa và tái cân bằng được thiết kế dễ lập trình hơn nhờ quy tắc màu và giới hạn chiều cao.

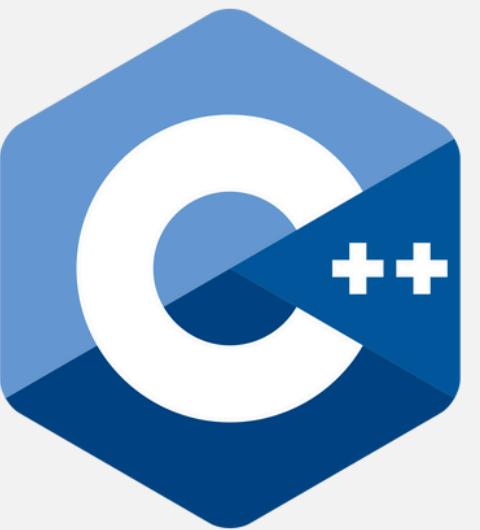
Độ phức tạp

	Trung bình	Xấu nhất
Không gian	$O(n)$	$O(n)$
Tìm kiếm	$O(\log n)$	$O(\log n)$
Thêm	$O(\log n)$	$O(\log n)$
Xóa	$O(\log n)$	$O(\log n)$

05. Ứng dụng



Lập tiến trình CPU cho hệ điều hành linux



Hầu hết các chức năng của thư viện cây BST tự cân bằng trong C++(map, set) và Java(TreeMap, TreeSet) đều sử dụng cấu trúc cây đỏ đen



05. Ứng dụng

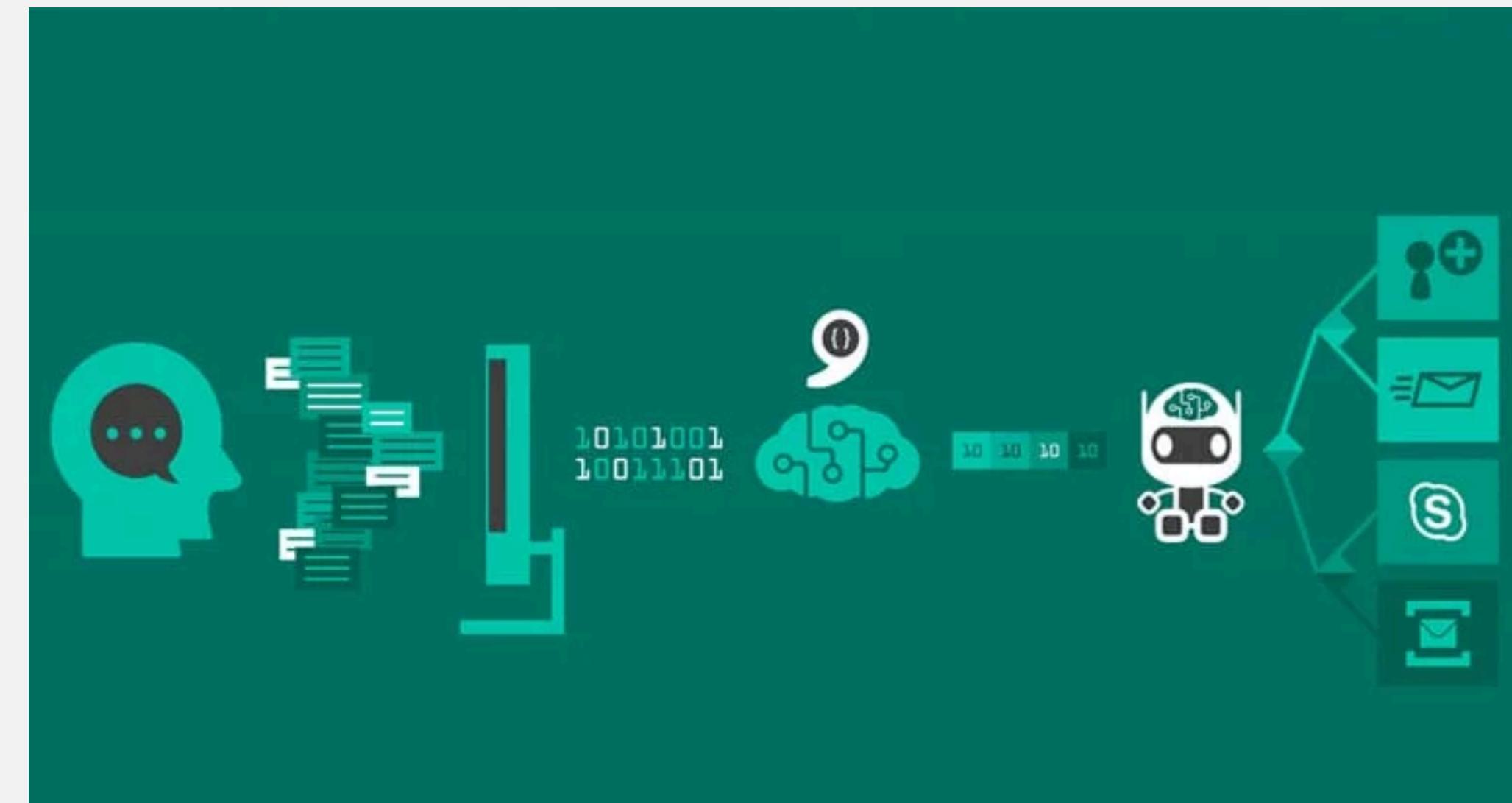
Bộ chỉ mục

Bộ gõ dự đoán



Ngôn ngữ tự động hóa thiết
kế

Trí tuệ nhân tạo / NLP – Cấu trúc dữ liệu cho
mô hình xử lý văn bản, từ điển token có thứ
tự



06. Kết luận

Tóm tắt đặc điểm

- Là cây nhị phân tìm kiếm cân bằng
- Đảm bảo thời gian thao tác $O(\log(n))$ với:
 - Tìm kiếm, chèn, xóa
- Sử dụng quy tắc đỏ đen để duy trì tính cân bằng

06. Kết luận

Ưu điểm nổi bật

- Hiệu suất ổn định, kể cả với dữ liệu thay đổi liên tục
- Được ứng dụng rộng rãi
 - STL
 - Cơ sở dữ liệu
 - Hệ điều hành
 - Trí tuệ nhân tạo



Quizizz

Demo.



Nội dung trình bày

Red-Black
Tree
Visualizer

Demo
Dictionary