# COSE213: Data Structure

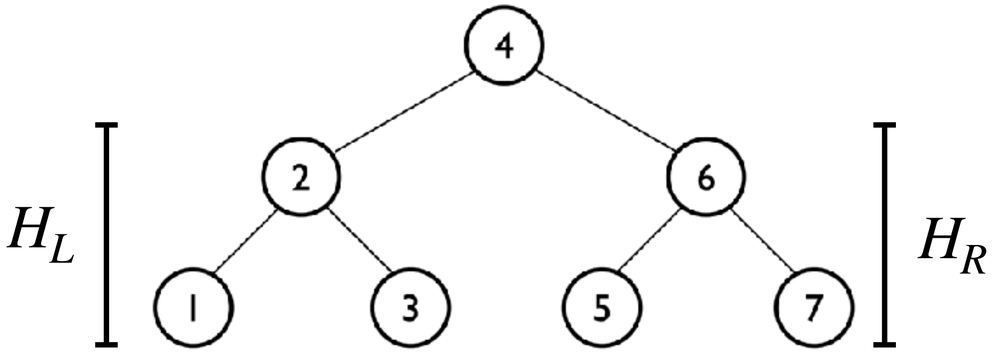
## Lecture 9 review

Minseok Jeon

2024 Fall

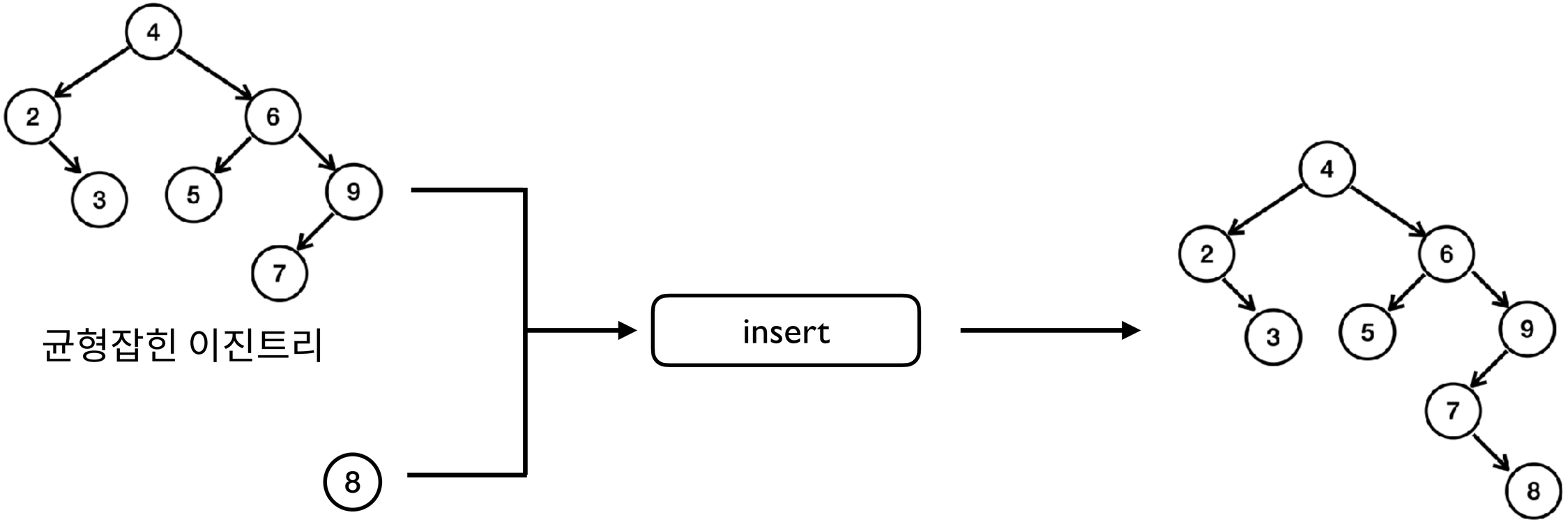
# AVL 트리

- AVL 트리: 항상 균형을 이루는 이진 탐색 트리
- AVL 트리가 만족해야 할 성질
  - 균형인수 $B$ ($H_L - H_R$)가 -1, 0, 또는 1
    - $B = 1$ : 왼쪽 서브트리의 높이가 1 더 큼
    - $B = 0$ : 두 서브트리의 높이가 같음
    - $B = -1$ : 오른쪽 서브트리의 높이가 1 더 큼
- AVL 탐색 트리의 **서브트리도 AVL 트리**임



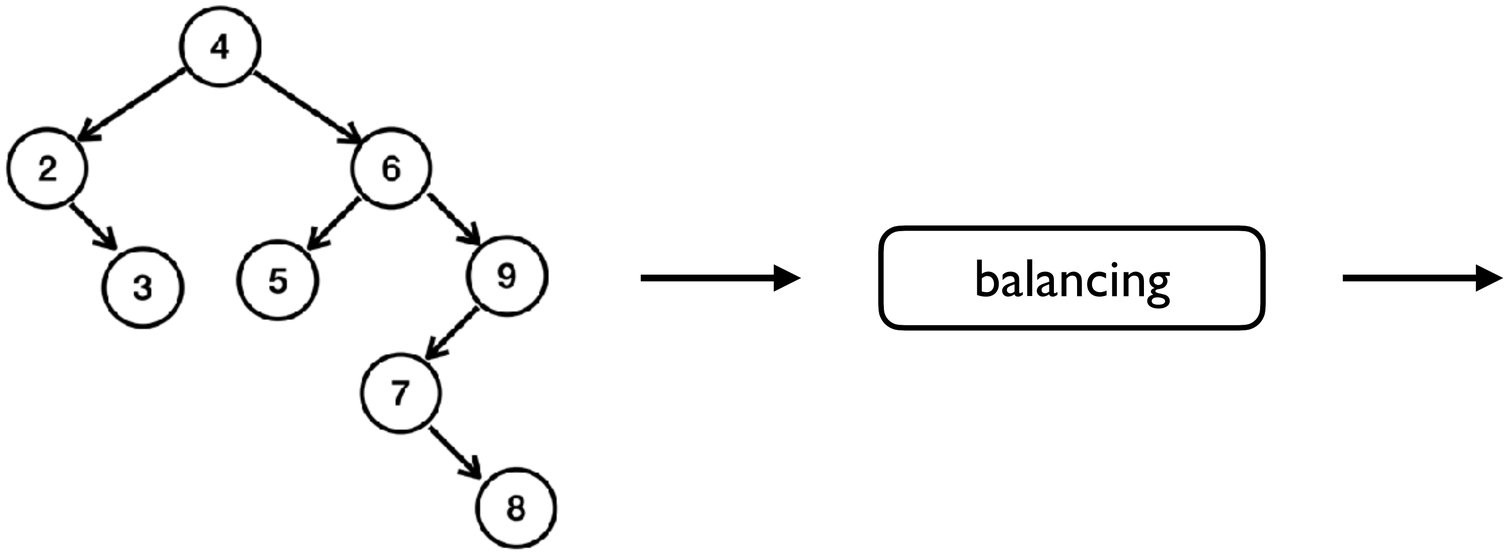
# 이진 탐색 트리에서의 삽입(insert)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음



# 트리 균형 잡기 (Balancing)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음
  - 균형이 깨질 경우 노드 회전(rotation)을 통해 균형을 잡아야 함

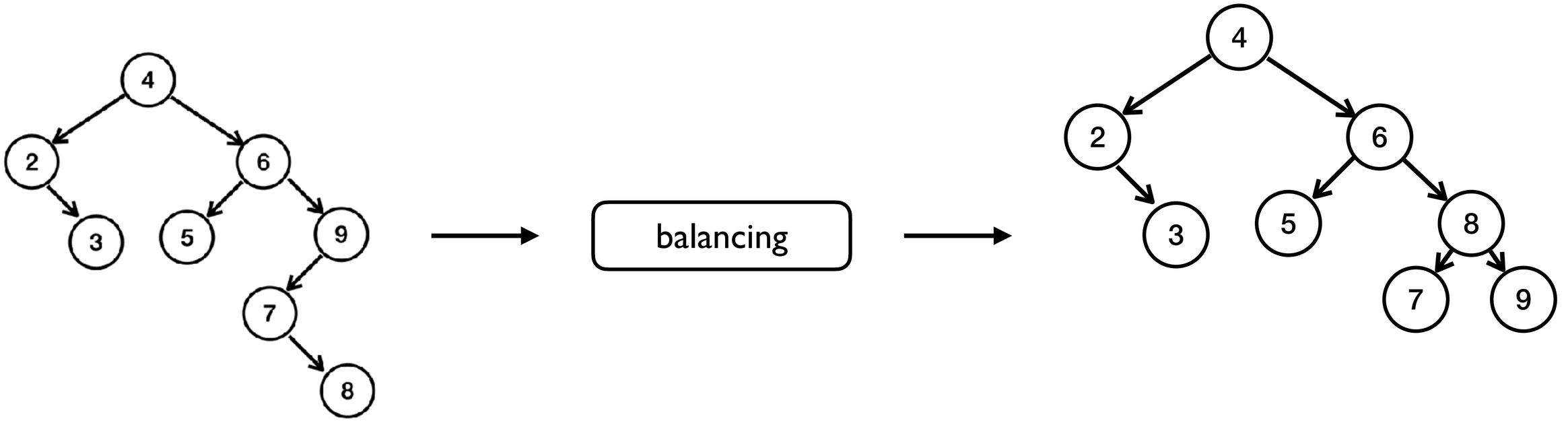



불균형한 이진트리

균형잡힌 이진트리

# 트리 균형 잡기 (Balancing)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음
  - 균형이 깨질 경우 노드 회전(rotation)을 통해 균형을 잡아야 함

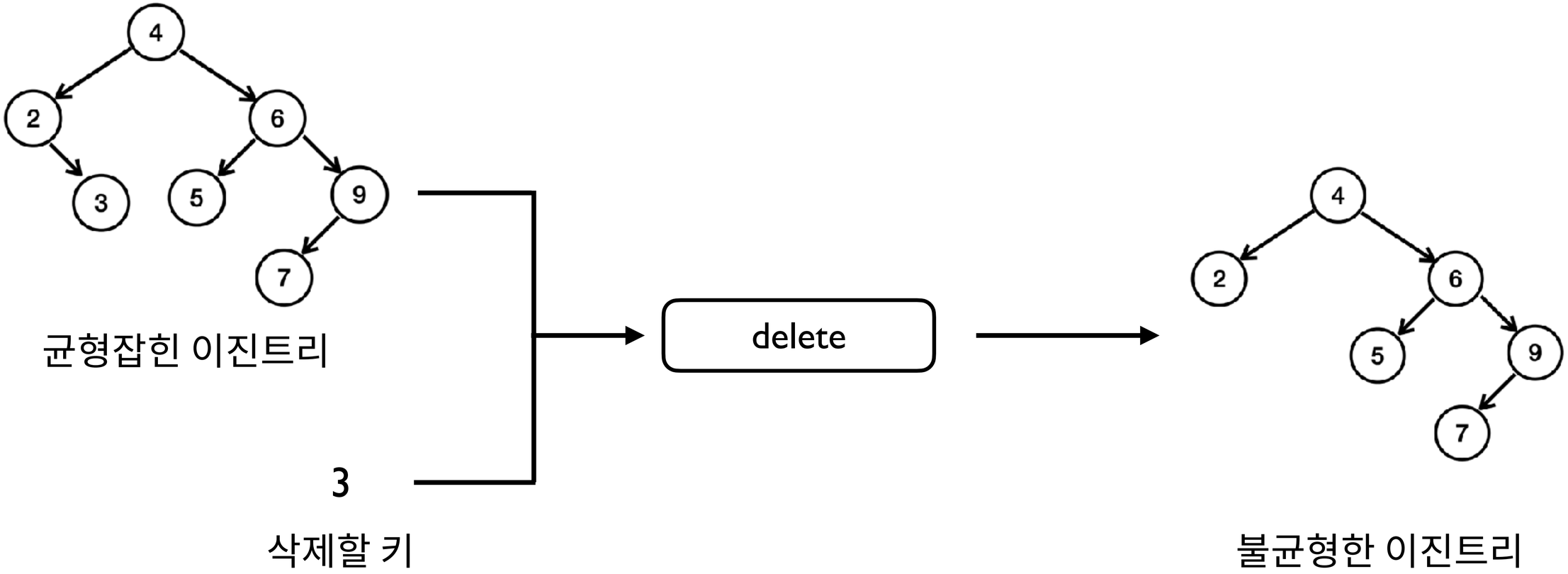



불균형한 이진트리

균형잡힌 이진트리

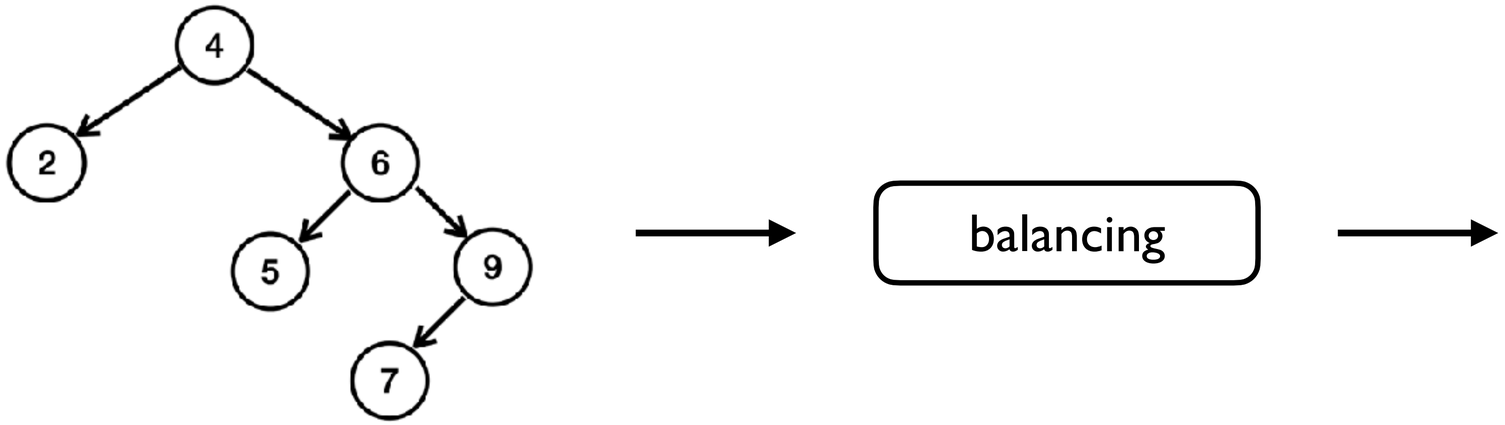
# 이진 탐색 트리에서의 삭제(delete)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음



# 트리 균형 잡기 (Balancing Trees)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음
  - 균형이 깨질 경우 노드 회전(rotation)을 통해 균형을 잡아야 함

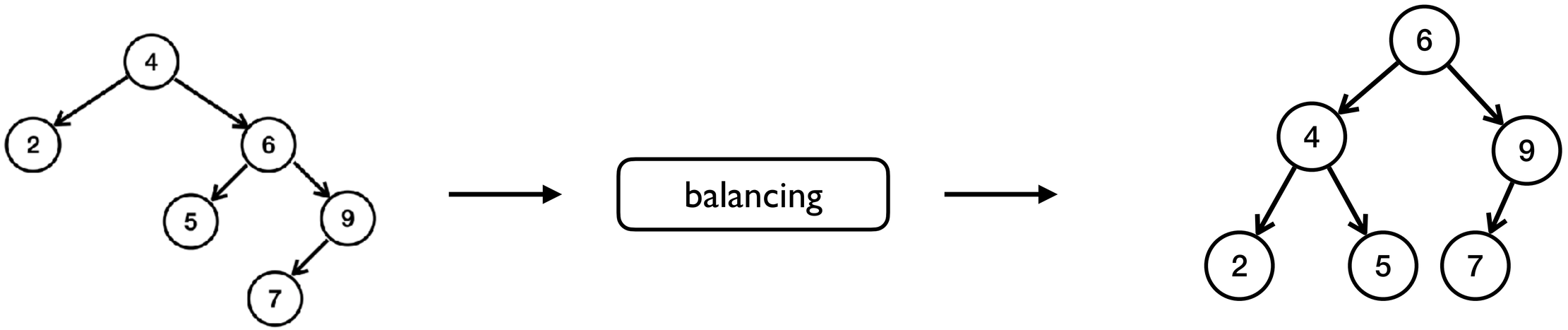



불균형한 이진트리

균형잡힌 이진트리

# 트리 균형 잡기 (Balancing Trees)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음
  - 균형이 깨질 경우 노드 회전(rotation)을 통해 균형을 잡아야 함




불균형한 이진트리

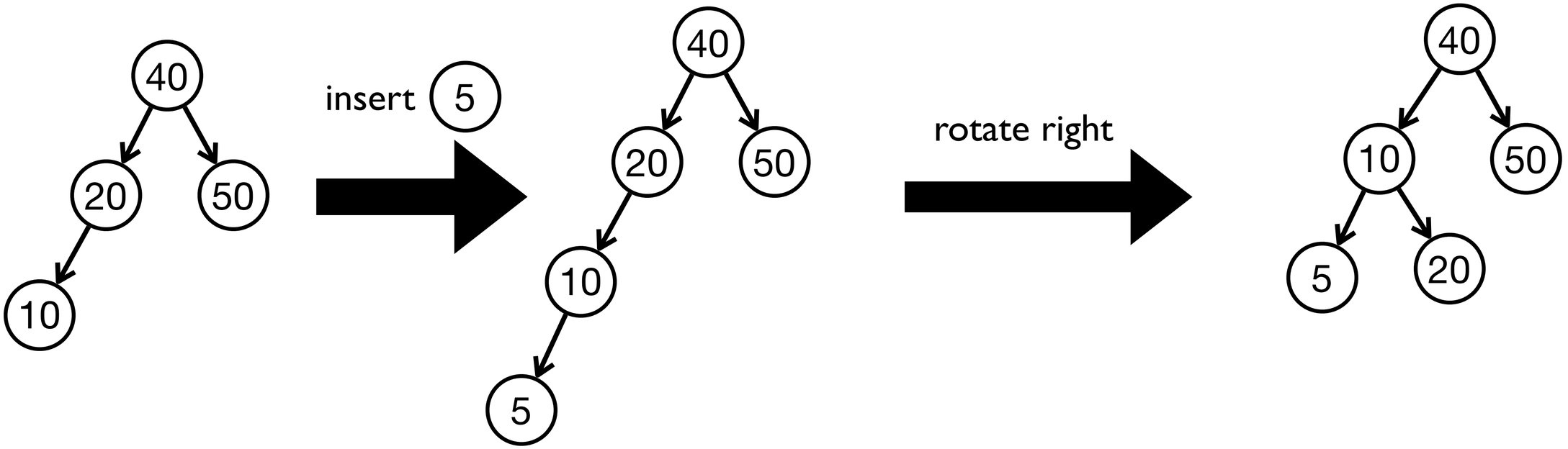
불균형한 이진트리

# 트리 균형 잡기 (Balancing Trees)

- 노드 삽입, 삭제 후 트리 균형잡기가 필요한 네가지 경우
  - LL(Left-Left) 케이스: 불균형 노드의 왼쪽(L) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
  - RR(Right-Right) 케이스: 불균형 노드의 오른쪽(R) 자식의 오른쪽(R) 서브트리가 너무 깊어 불균형이 발생한 경우
  - LR(Left-Right) 케이스: 불균형 노드의 왼쪽(L) 자식의 오른쪽(R) 서브트리가 너무 깊어 불균형이 발생한 경우
  - RL(Right-Left) 케이스: 불균형 노드의 오른쪽(R) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우

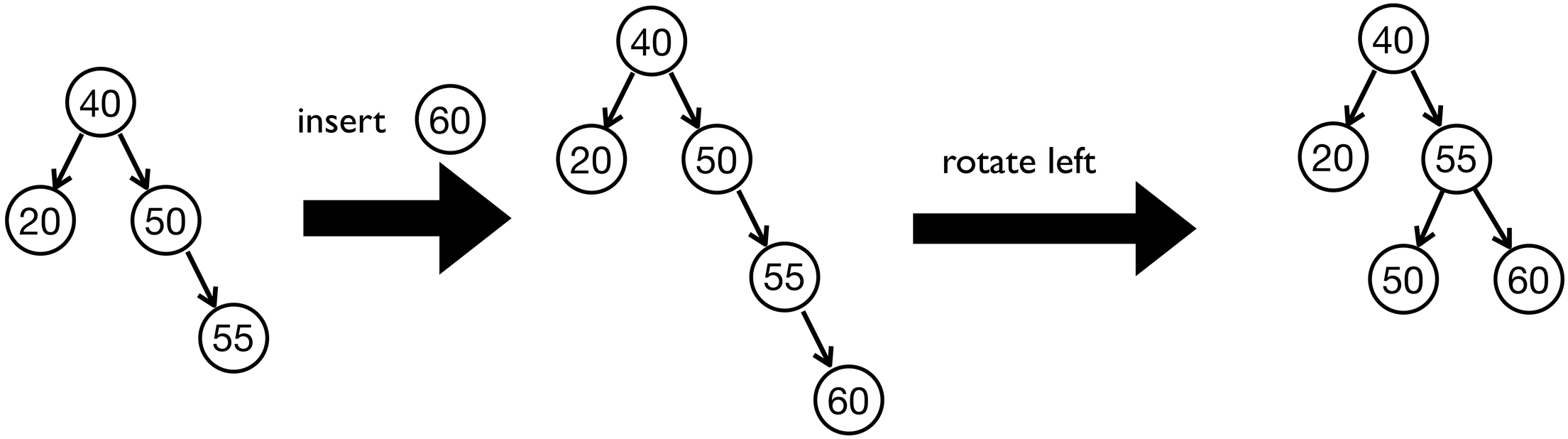
# 트리 균형잡기가 필요한 네가지 경우

- LL(Left - Left) 케이스: 불균형 노드의 왼쪽(L) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드를 오른쪽으로 회전



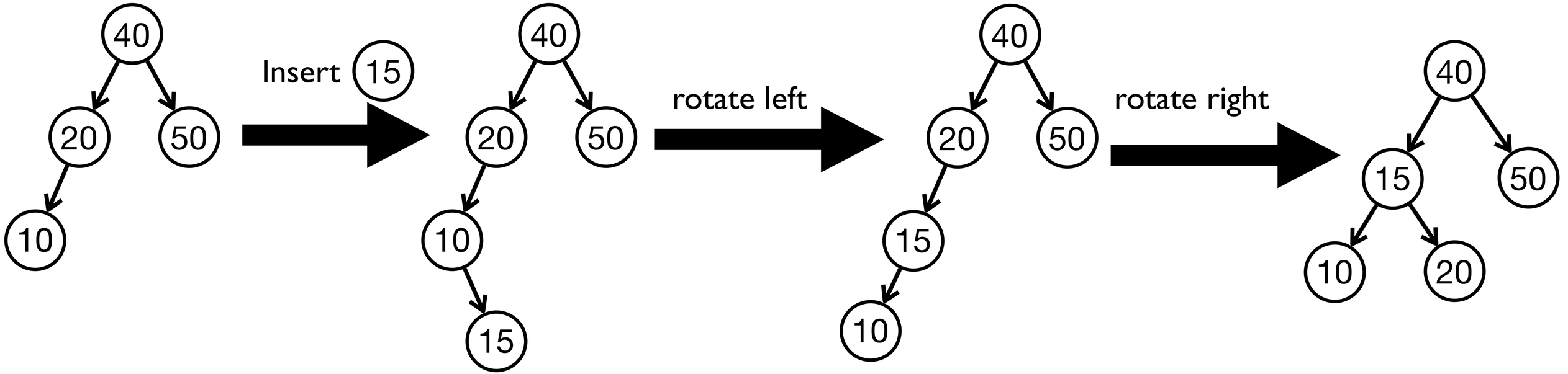
# 트리 균형잡기가 필요한 네가지 경우

- RR(Right - Right) 케이스: 불균형 노드의 오른쪽(R) 자식의 오른(R) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드를 왼쪽으로 회전



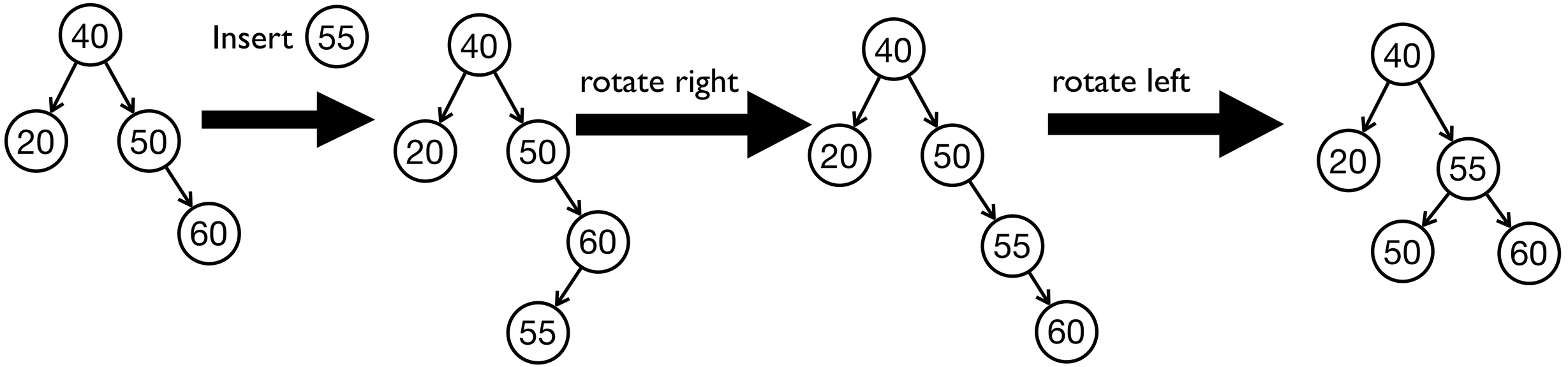
# 트리 균형잡기가 필요한 네가지 경우

- LR(Left - Right) 케이스: 불균형 노드의 왼쪽(L) 자식의 오른쪽(R) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드의 왼쪽 자식노드를 왼쪽으로 회전 후 분균형이 발생한 노드를 오른쪽으로 회전



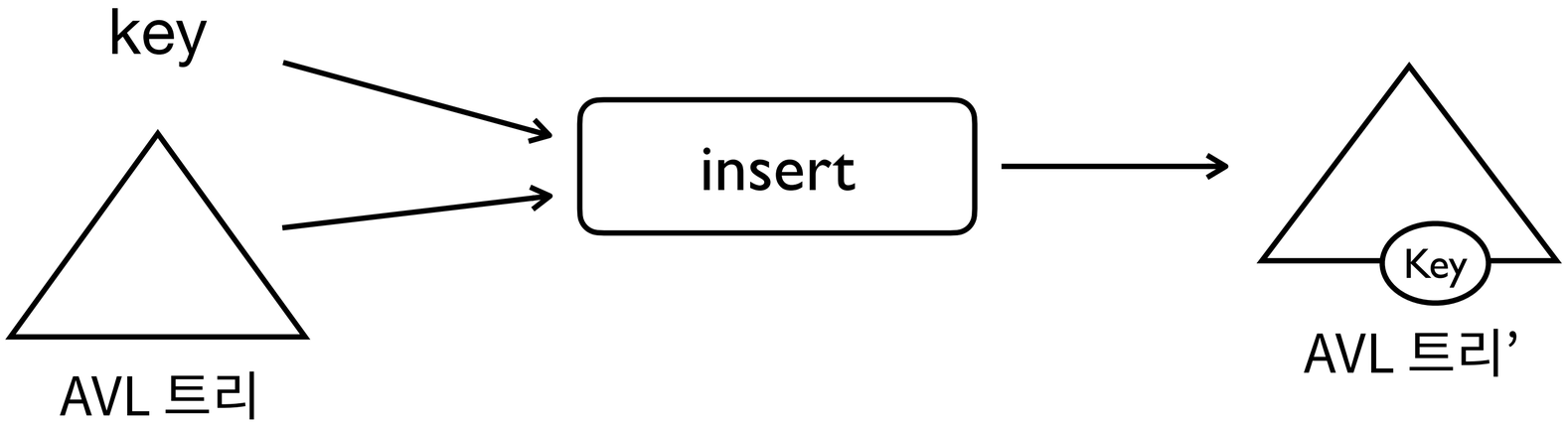
# 트리 균형잡기가 필요한 네가지 경우

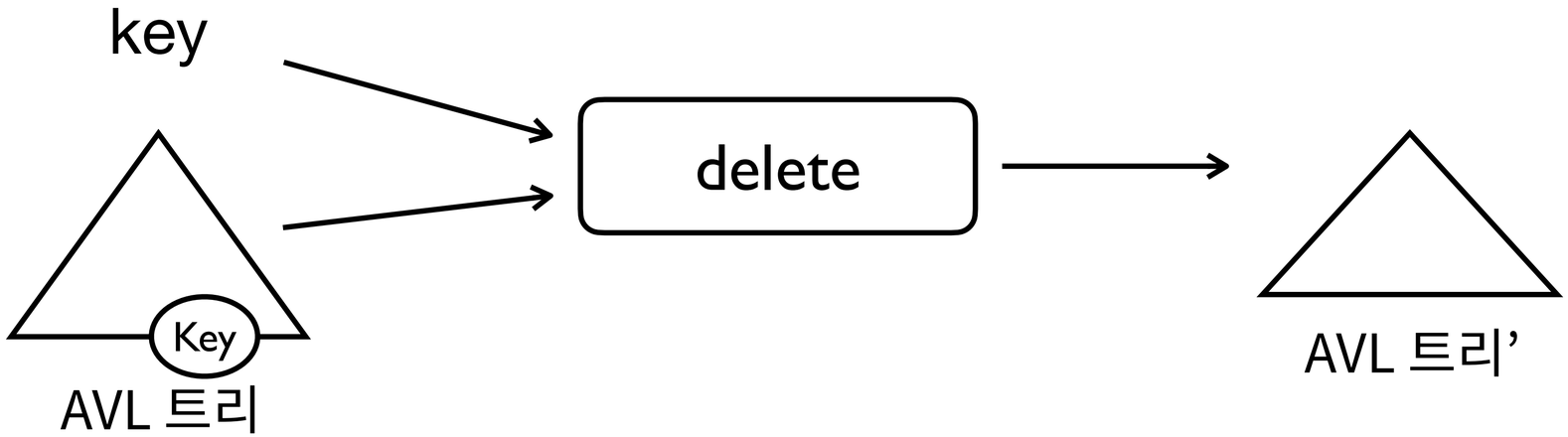
- RL(Right - Left) 케이스: 불균형 노드의 오른쪽(R) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드의 오른쪽  자식노드를 오른쪽으로 회전 후 분균형이 발생한 노드를 왼쪽으로 회전



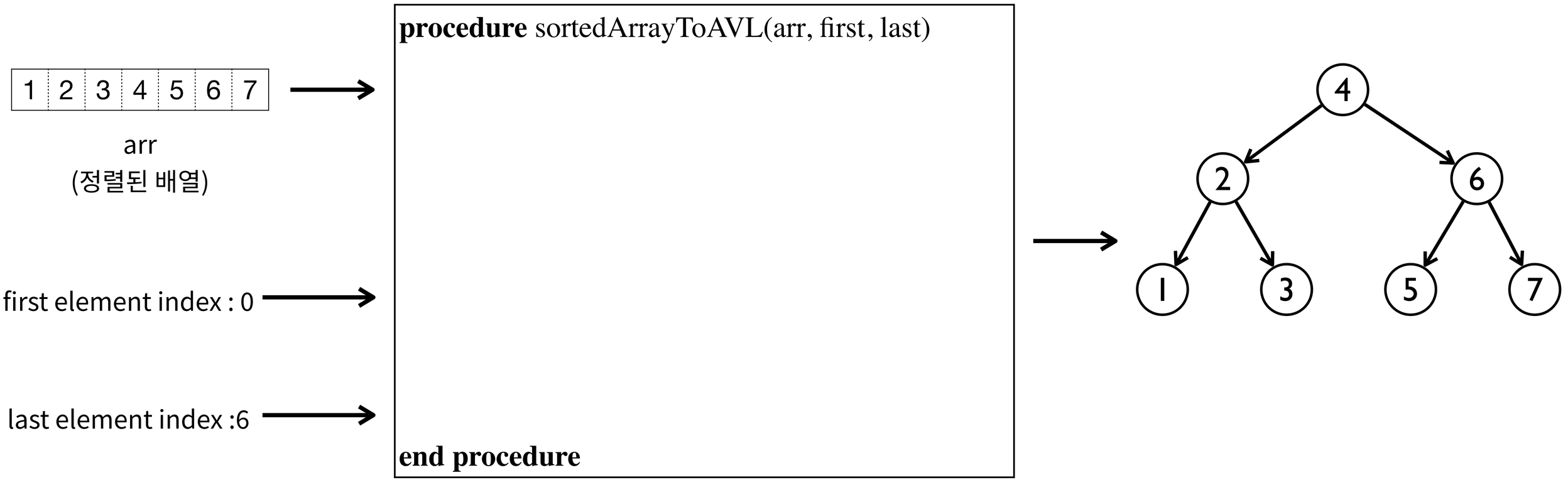
# Quiz

- AVL트리에서 삽입(insert) 또는 삭제(delete)에서 밸런싱(또는 회전)은 최대 몇 회 발생할까?



# 주어진 배열을 AVL트리로 만들기

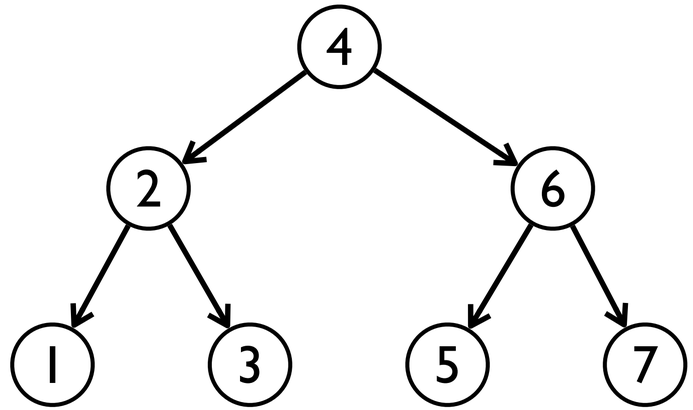


# 주어진 배열을 AVL트리로 만들기

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|

arr
(정렬된 배열)

first element index : 0

last element index :6

```
procedure sortedArrayToAVL(arr, first, last)
  if start > end then
    return NULL
  end if
  mid ← first + ⌊(last - first)/2⌋
  root ← allocateNode()
  root.key ← arr[mid]
  root.left ← NULL
  root.right ← NULL
  root.left ← sortedArrayToAVL(arr, first, mid - 1)
  root.right ← sortedArrayToAVL(arr, mid+1, last)
  return root
end procedure
```