Binomial Heaps

Chiranjeev Kumar

IIT(ISM), Dhanbad

Slides from Prof. Jim Anderson, [California State University, Northridge]

Binomial Heaps

- A way to implement mergeable heaps.
  - Useful in scheduling algorithms and graph algorithms.

- Operations:
  - Make-Heap().
  - Insert(H, x), where x is a node H.
  - Minimum(H).
  - Extract-Min(H).
  - Union(H_1, H_2): merge H_1 and H_2, creating a new heap.
  - Decrease-Key(H, x, k): decrease x.key (x is a node in H) to k. (It's assumed that $k \le$ x.key.)

Definitions

Binomial Heap: Collection of binomial trees (satisfying some properties).

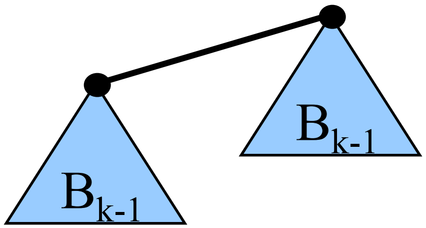
Binomial Trees

Definition is **inductive**.

Base Case: B_0 = single node is a binomial tree.

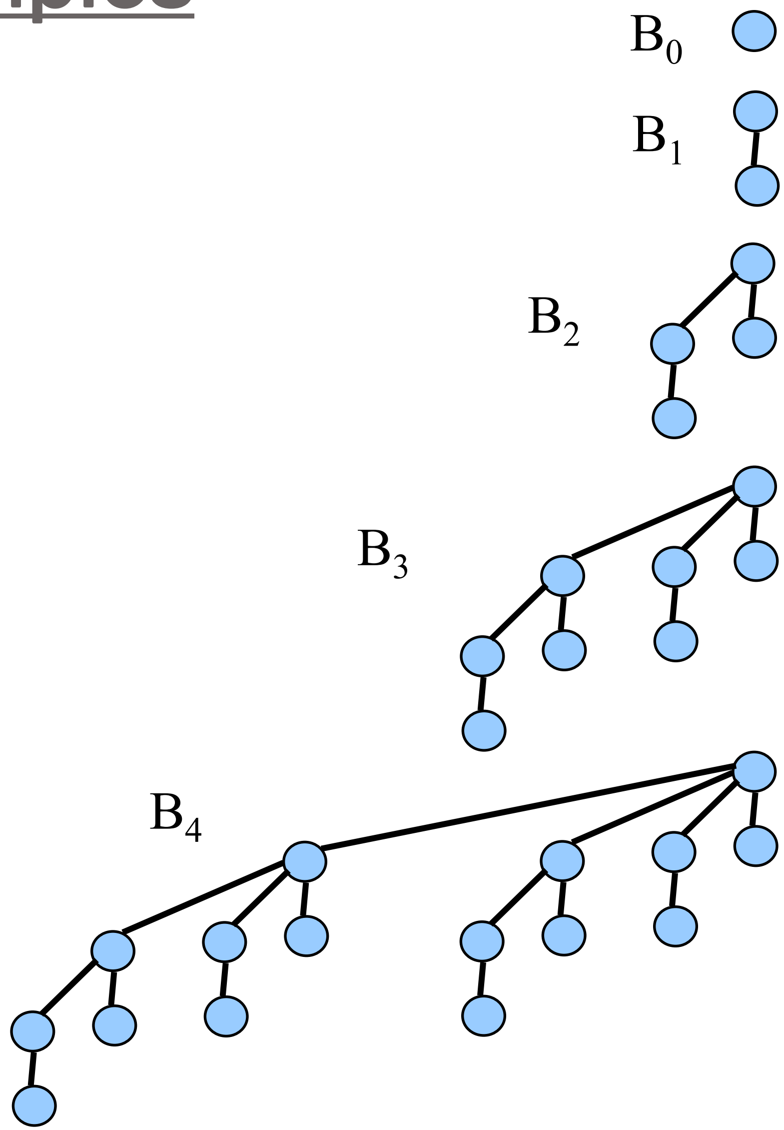
Inductive Step:

B_k =



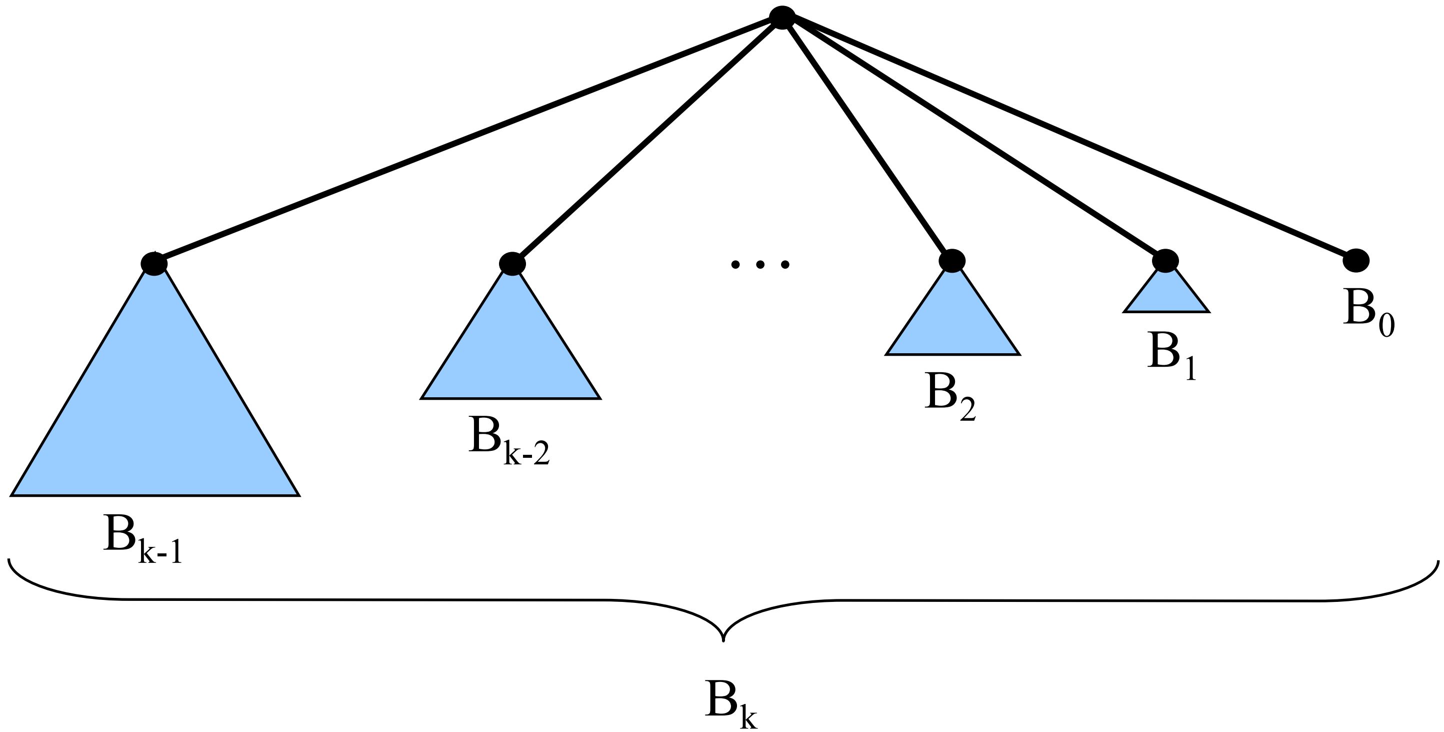
is a binomial tree.

Examples



depth	# nodes
0	1
1	4
2	6
3	4
4	1

Another Way to Look at B_k



Properties of Binomial Trees

Lemma: For the binomial tree B_k,

1. There are 2^k nodes.

2. Tree height is k.

3. $\binom{k}{i}$ nodes at depth i, i = 0, 1, …, k **[binomial coefficients]**.

4. Root has degree k, other nodes have smaller degree. i^{th} child of root is root of subtree B_i, where i = k–1, k–2, …, 0 [k–1 is LM, 0 is RM].

Proof:

1. Induction on k.

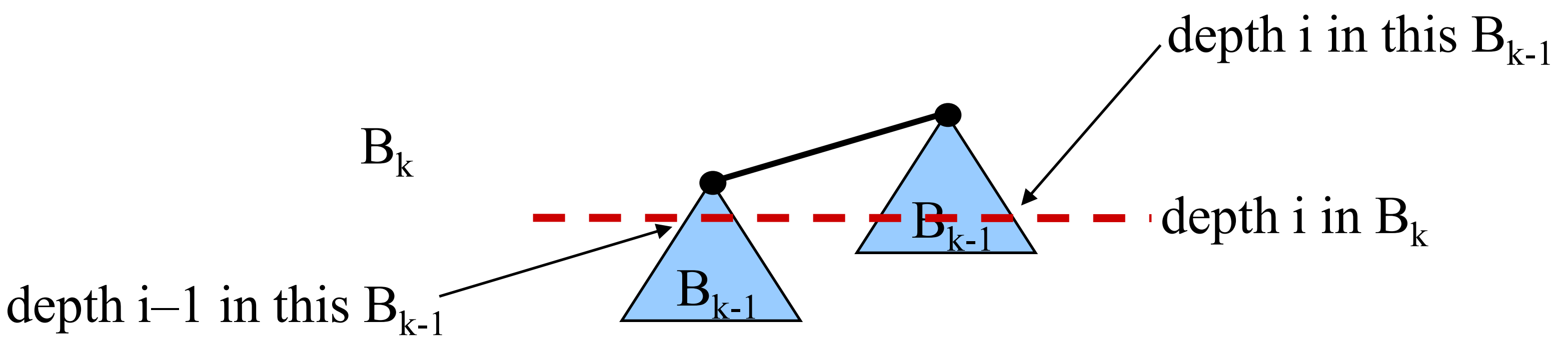
$$\begin{aligned} &\#\text{ nodes in } B_k \\ &= 2(\#\text{ nodes in } B_{k\text{-}1}) \\ &= 2(2^{k-1}) \\ &= 2^k \end{aligned}$$

2. Induction on k.

$$\begin{aligned} &\text{height of } B_k \\ &= 1 + \text{height of } B_{k\text{-}1} \\ &= 1 + (k-1) \\ &= k \end{aligned}$$

Proof (Continued)

3. Let $D(k,i)$ = # nodes at depth i in B_k. Want to show $D(k,i) = \binom{k}{i}$.



So,

$$
\begin{aligned}
& D(k,i) \\
&= D(k-1,i) + D(k-1,i-1) \\
&= \binom{k-1}{i} + \binom{k-1}{i-1} \quad \text{, by ind. hyp.} \\
&= \frac{(k-1)!}{i!\,(k-1-i)!} + \frac{(k-1)!}{(i-1)!(k-i)!}
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{(k-1)!(k-i) + i(k-1)!}{i!\,(k-i)!} \\
&= \frac{k! - i(k-1)! + i(k-1)!}{i!\,(k-i)!} \\
&= \binom{k}{i}
\end{aligned}
$$

Proof (Continued)

4. Root degree of B_k $= 1 +$ root degree of B_{k-1}

$= 1 + k-1$, induction hypothesis

$= k$

The maximum degree in an n-node binomial tree is lg n.

Binomial Heaps

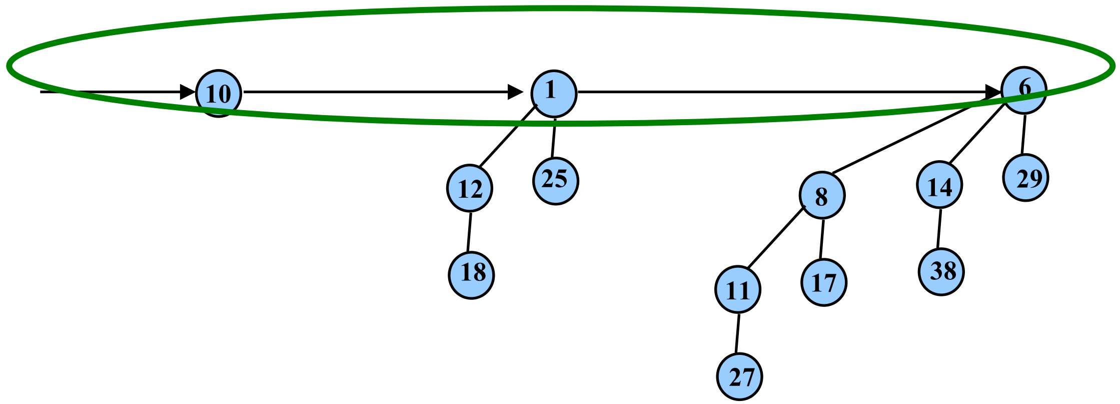
- Set of binomial trees satisfying **binomial-heap properties**:
  1. **Heap ordered:**
     - Implies root of a binomial tree has the smallest key in that tree.
  2. Set includes at most one binomial tree whose root is a given degree.
     - Implies B.H. with n nodes has at most $\lfloor \lg n \rfloor + 1$ B.T.'s.

       Think of n in binary: $\langle b_{\lfloor \lg n \rfloor}, \ldots, b_0 \rangle$, i.e.,

       B.H contains B_i iff b_i is 1.

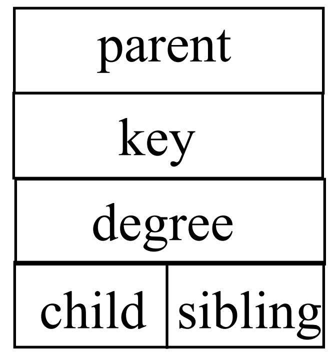
$$n = \sum_{i=1}^{\lfloor \lg n \rfloor} b_i 2^i$$

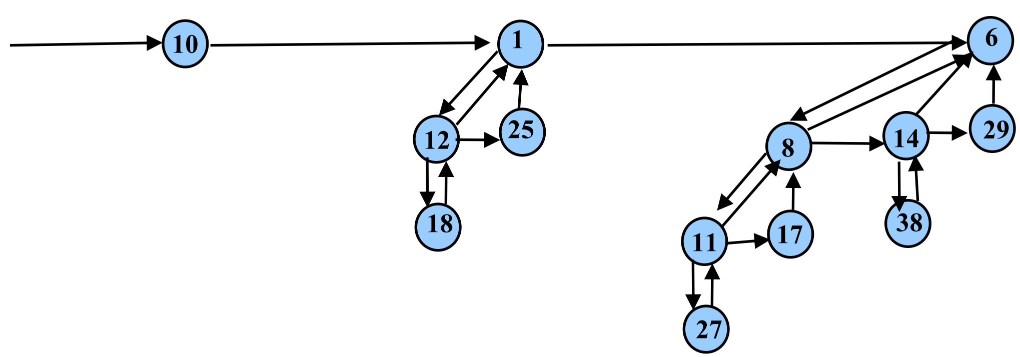
Representing Binomial Heaps



called the **root list**

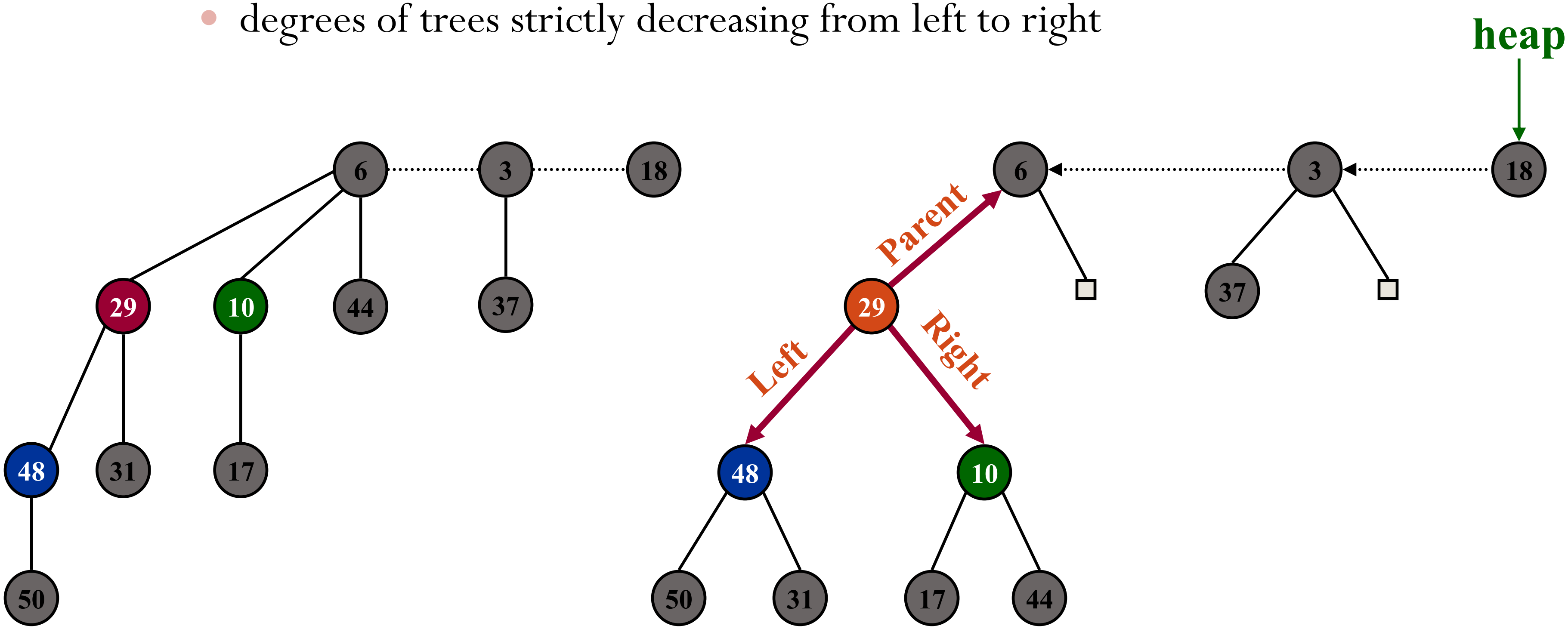
Each node is represented by a structure like this





Binomial Heap: Implementation

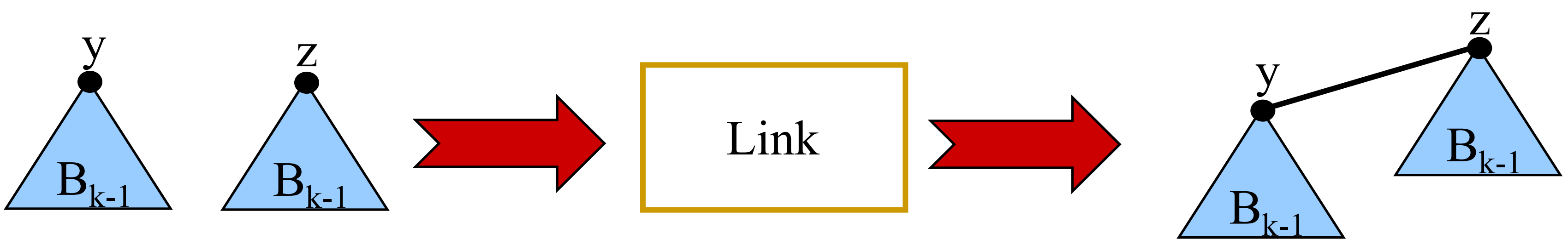
- Implementation.
  - Represent trees using left-child, right sibling pointers.
    - three links per node (parent, left, right)
  - Roots of trees connected with singly linked list.
    - degrees of trees strictly decreasing from left to right



Binomial Heap

Linking Two Binomial Heaps

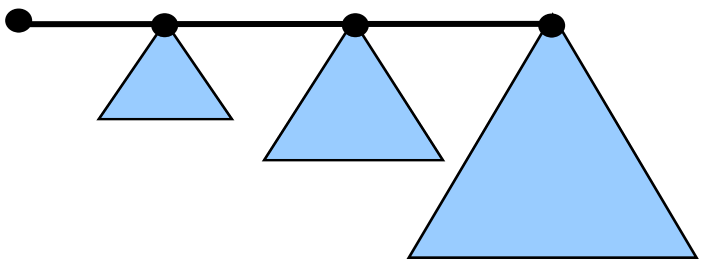
```
Link(y,z)
    p[y] := z;
    sibling[y] := child[z];
    child[z] := y;
    degree[z] := degree[z] + 1
```



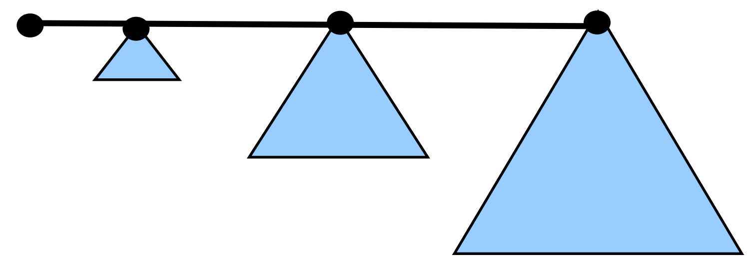
Union

H_1, H_2 → **Union** → $H_1 \cup H_2$

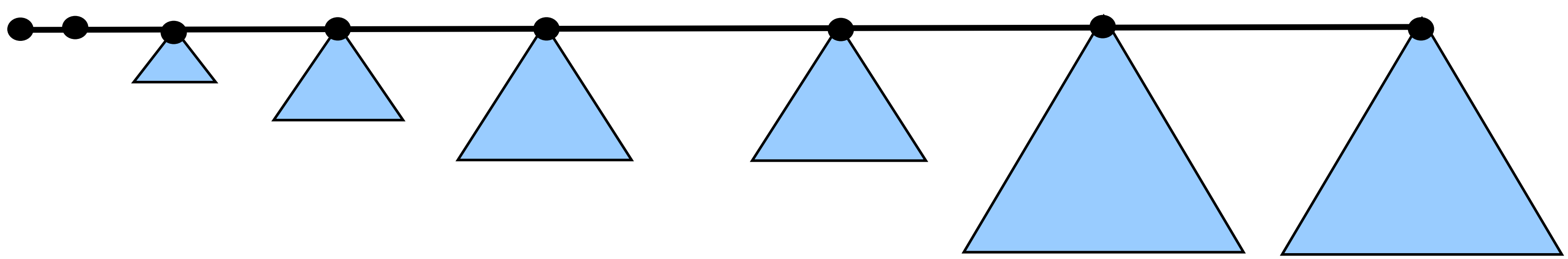
$H_1 =$



$H_2 =$



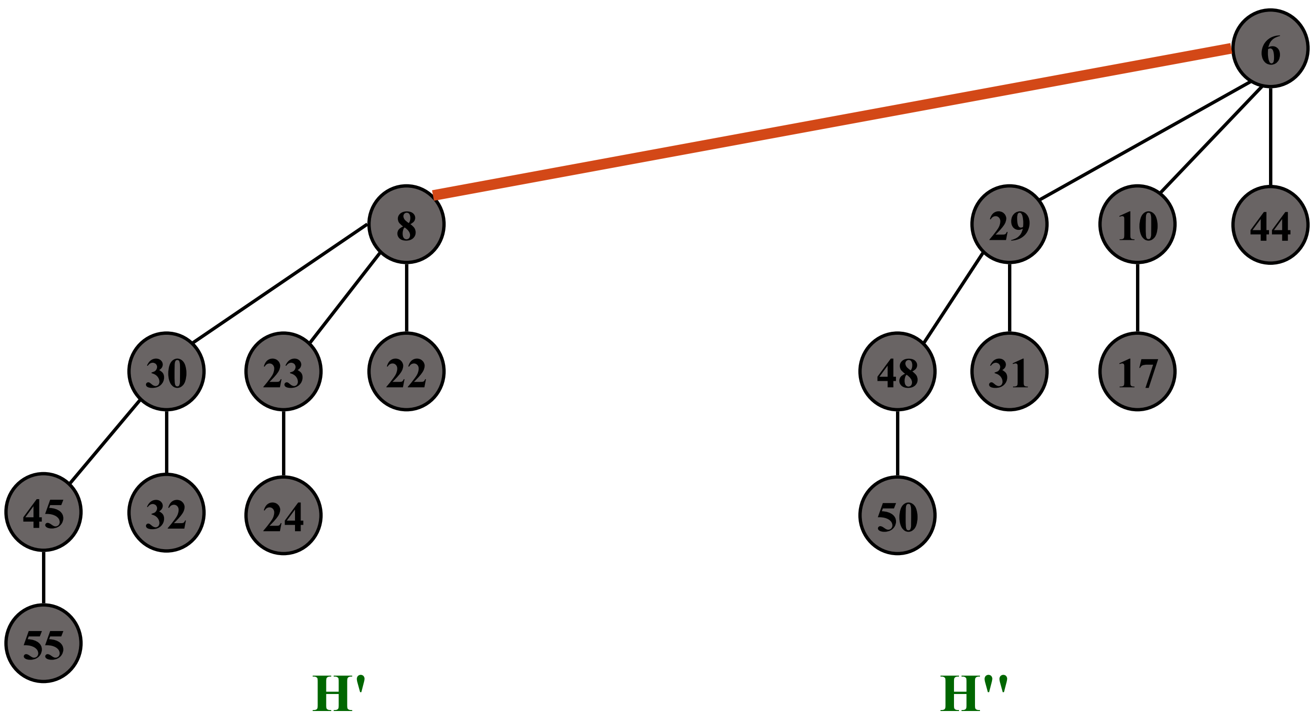
First, simply merge the two root lists by root degree (like merge sort).



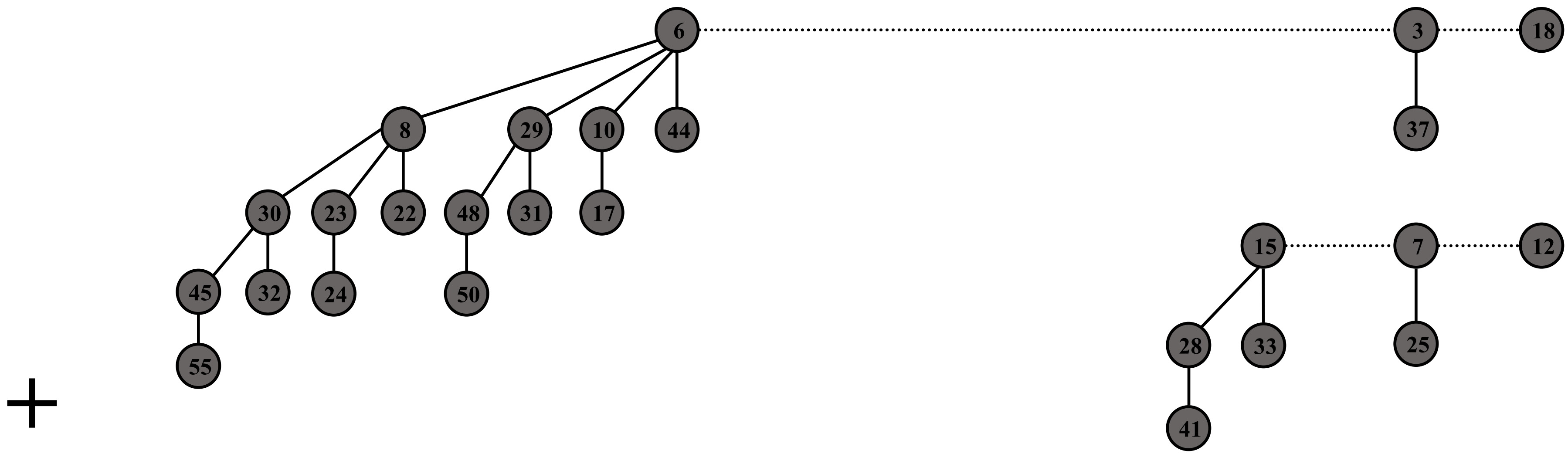
Remaining Problem: Can have two trees with the same root degree.

Binomial Heap: Union

- Create heap H that is union of heaps H' and H''.
  - "Mergeable heaps."
  - Easy if H' and H'' are each order k binomial trees.
    - connect roots of H' and H''
    - choose smaller key to be root of H



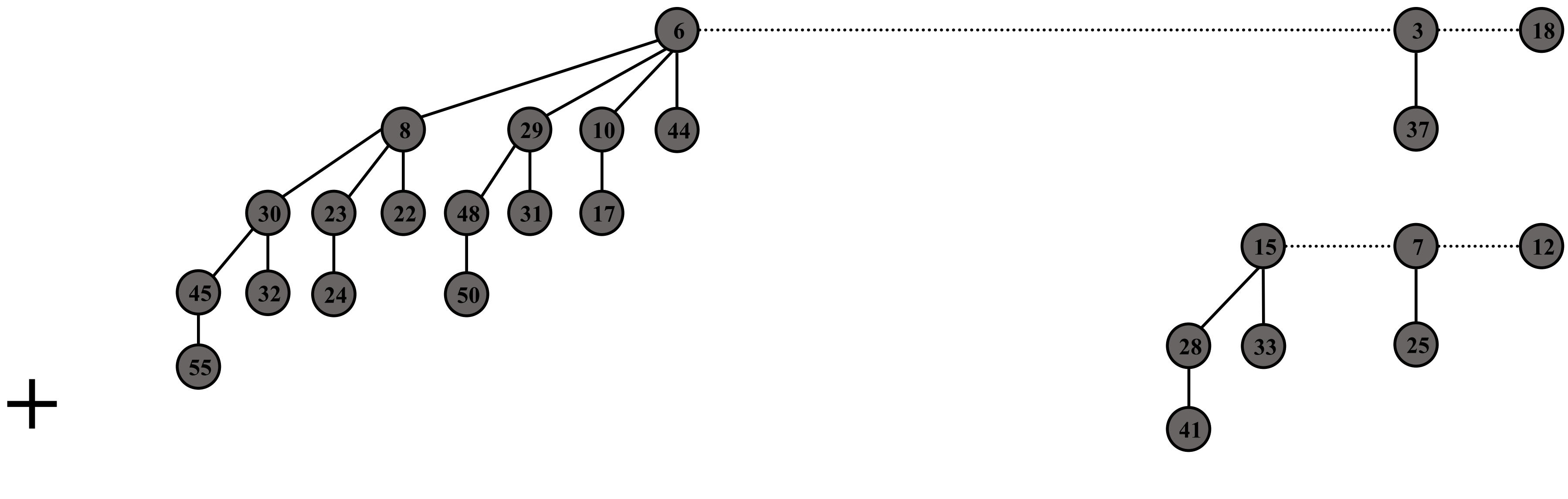
Binomial Heap: Union



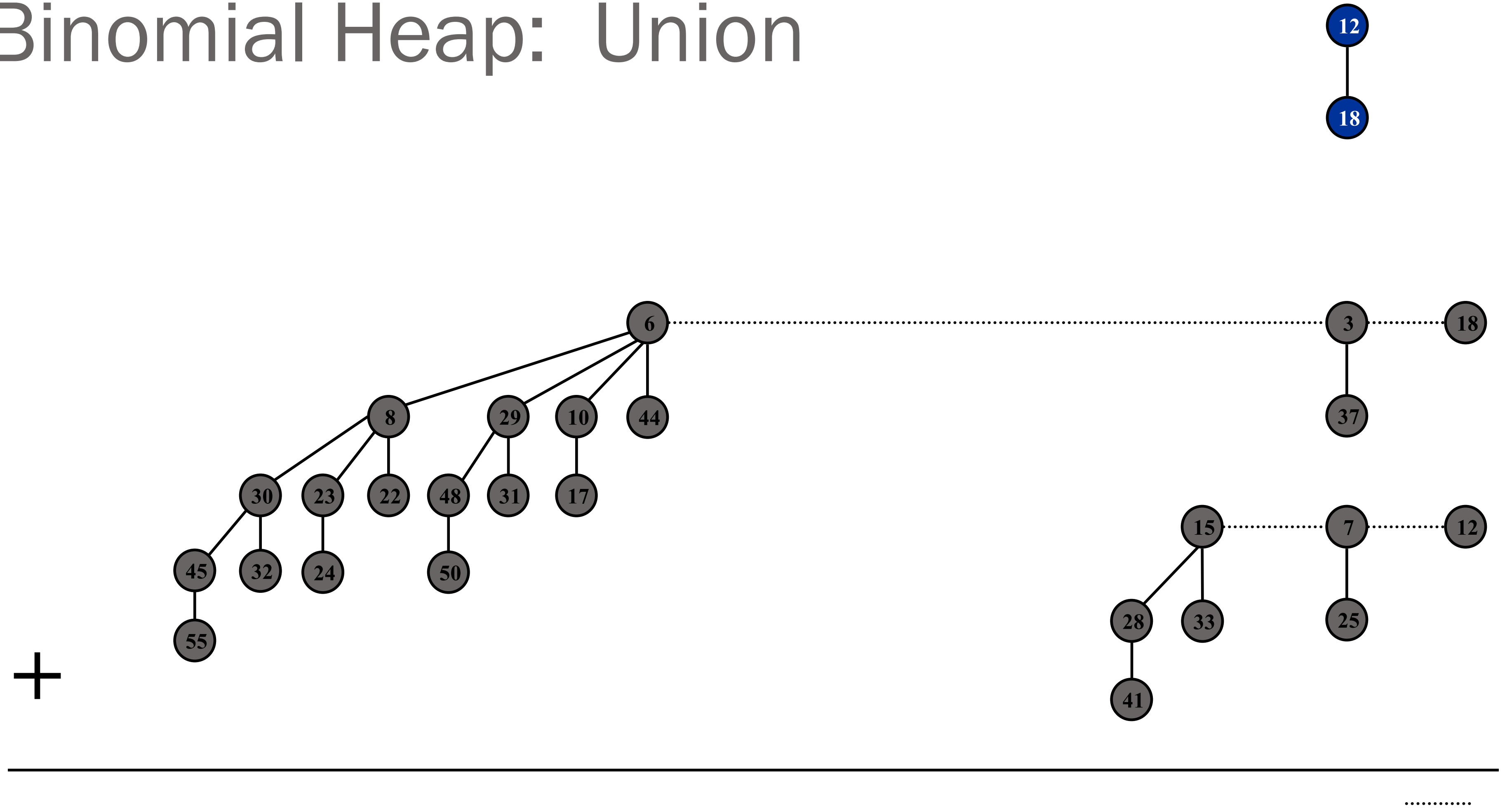
		1	1	1	
	1	0	0	1	1
+	0	0	1	1	1
	1	1	0	1	0

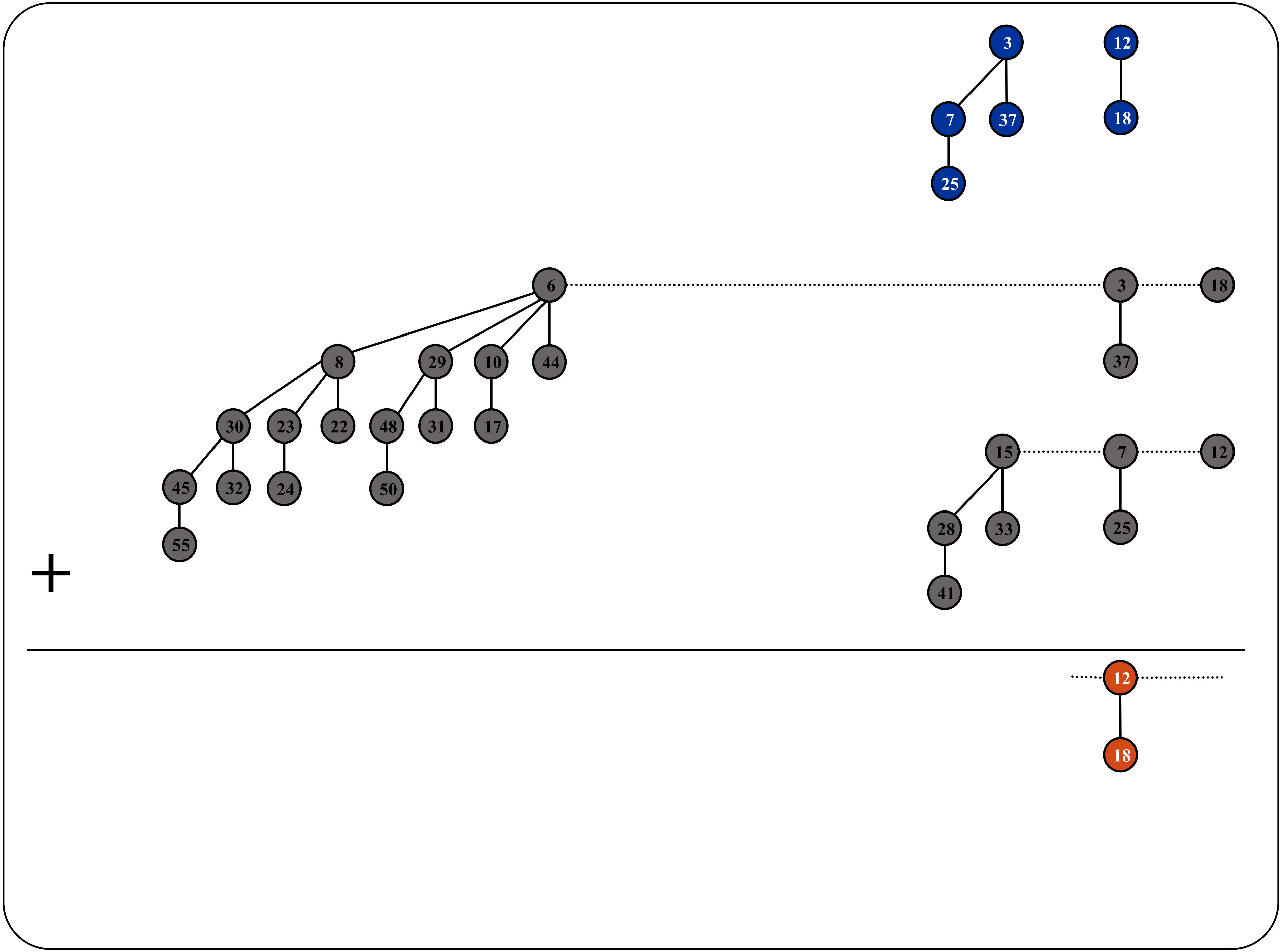
19 + 7 = 26

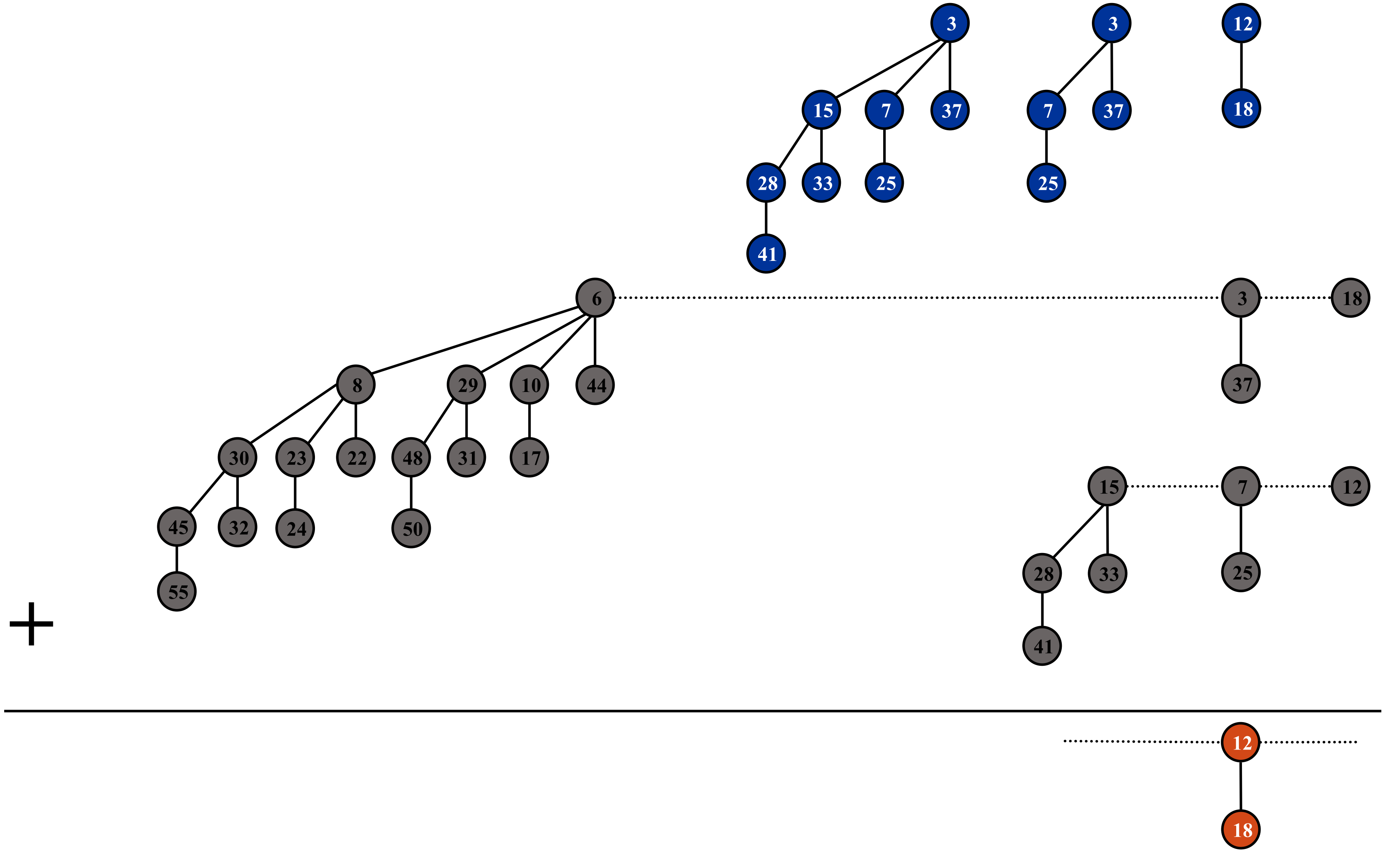
Binomial Heap: Union

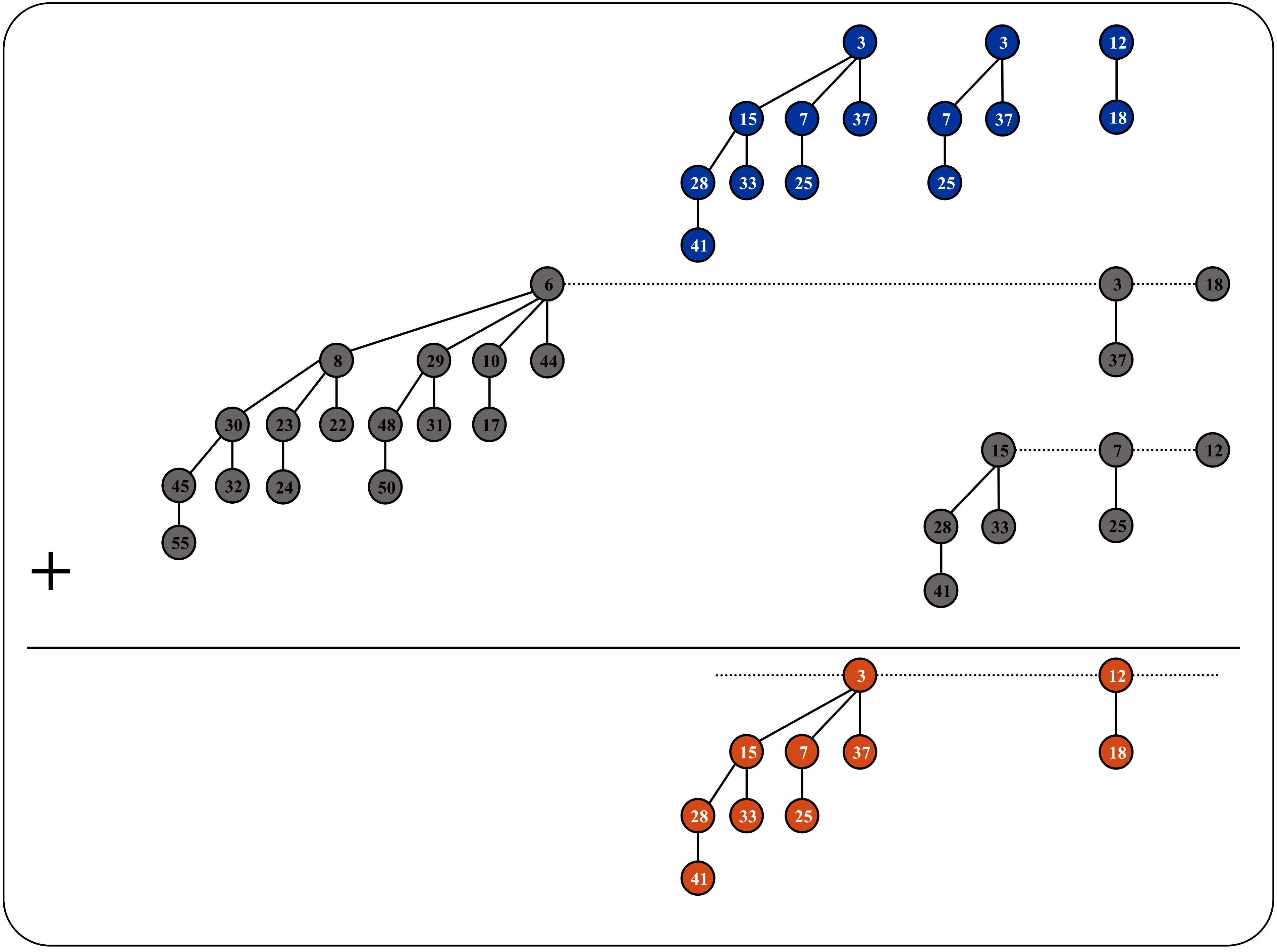


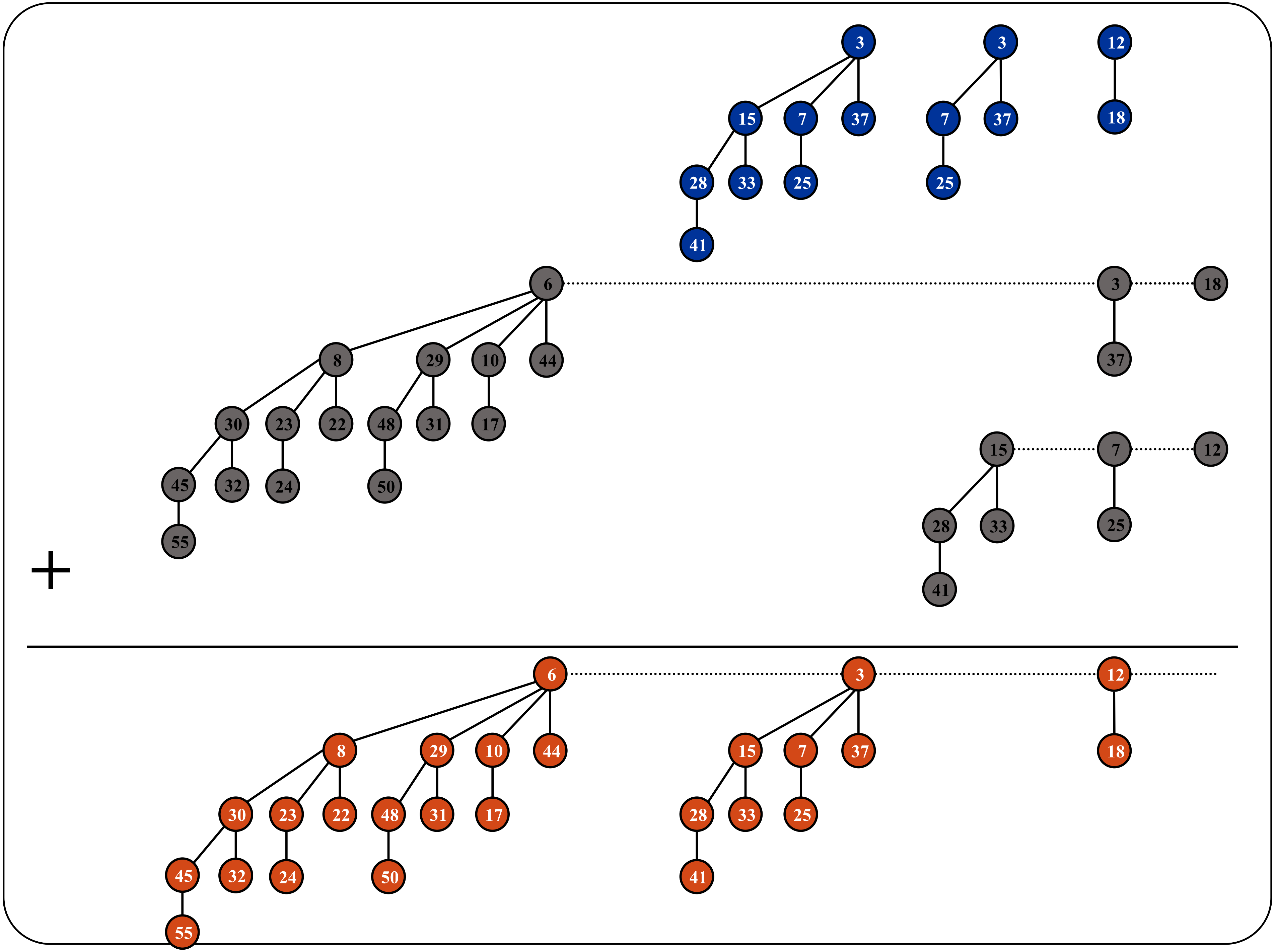
Binomial Heap: Union




3
12
7
37
18
25
6
3
18
8
29
10
44
37
30
23
22
48
31
17
15
7
12
45
32
24
50
28
33
25
55
+
41
12
18


3
15
7
37
28
33
25
41
3
7
37
25
12
18
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
3
18
37
15
7
12
28
33
25
41
+
12
18

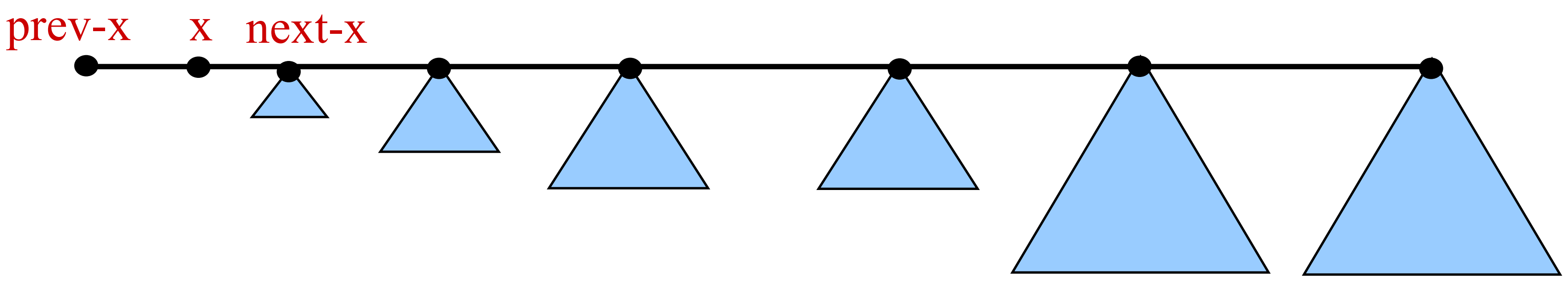

3
15
7
37
28
33
25
41
3
7
37
25
12
18
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
3
18
37
15
7
12
28
33
25
41
+
3
12
15
7
37
18
28
33
25
41



3 3 12
15 7 37 7 37 18
28 33 25 25
41
6 3 18
8 29 10 44 37
30 23 22 48 31 17
45 32 24 50
55
15 7 12
28 33 25
41
+
6 3 12
8 29 10 44 15 7 37 18
30 23 22 48 31 17 28 33 25
45 32 24 50 41
55

Union (Continued)

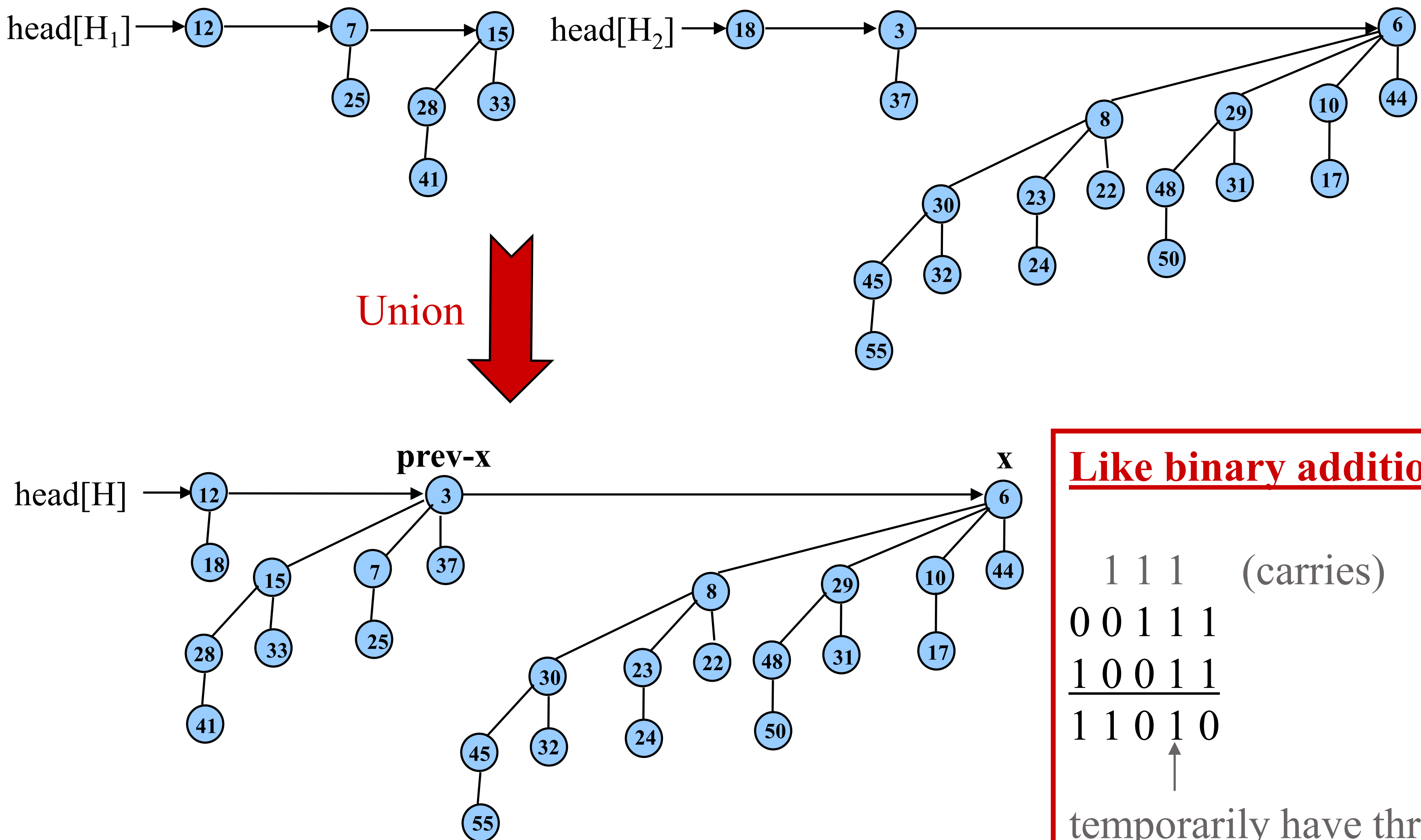
Union traverses the new root list like this:



Depending on what x, next-x, and sibling[next-x] point to, Union links trees with the same root degree.

Note: We may temporarily create three trees with the same root degree.

Analogy

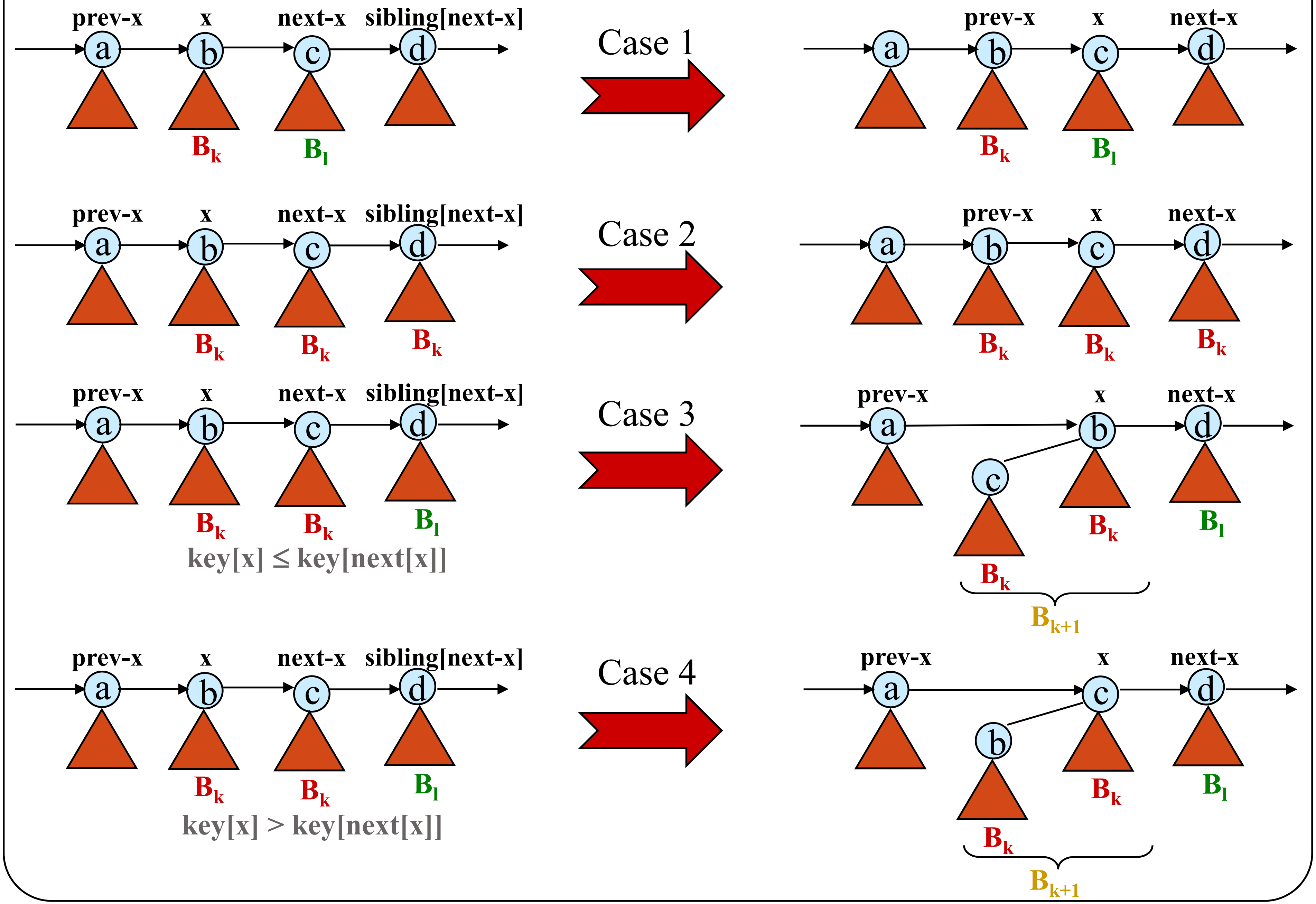


Like binary addition:

```
  1 1 1     (carries)
0 0 1 1 1
1 0 0 1 1
---------
1 1 0 1 0
```

temporarily have three trees of this degree

Cases

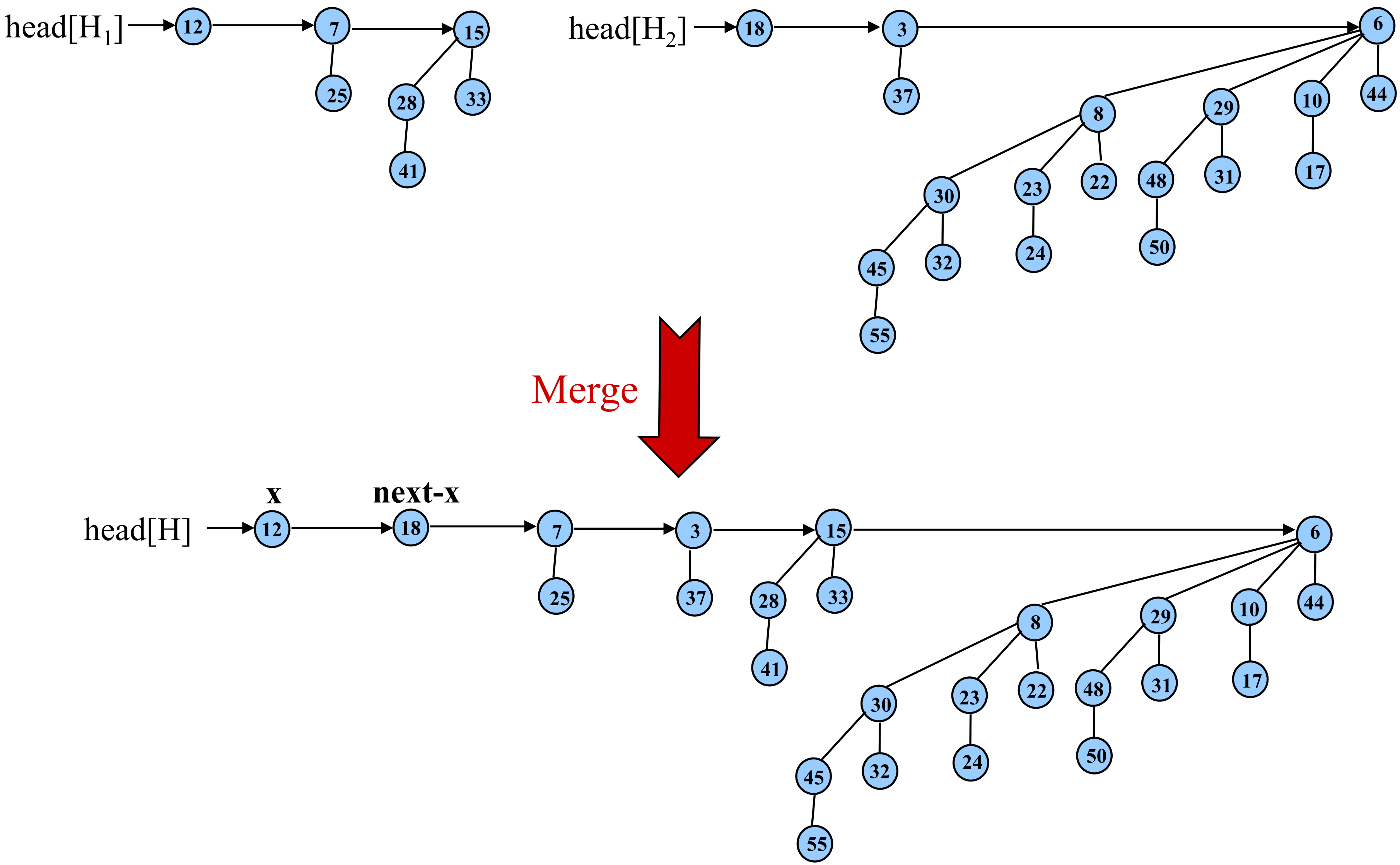


prev-x
x
next-x
sibling[next-x]
a
b
c
d
B
k
B
l
Case 1
Case 2
Case 3
Case 4
key[x] ≤ key[next[x]]
key[x] > key[next[x]]
B
k+1

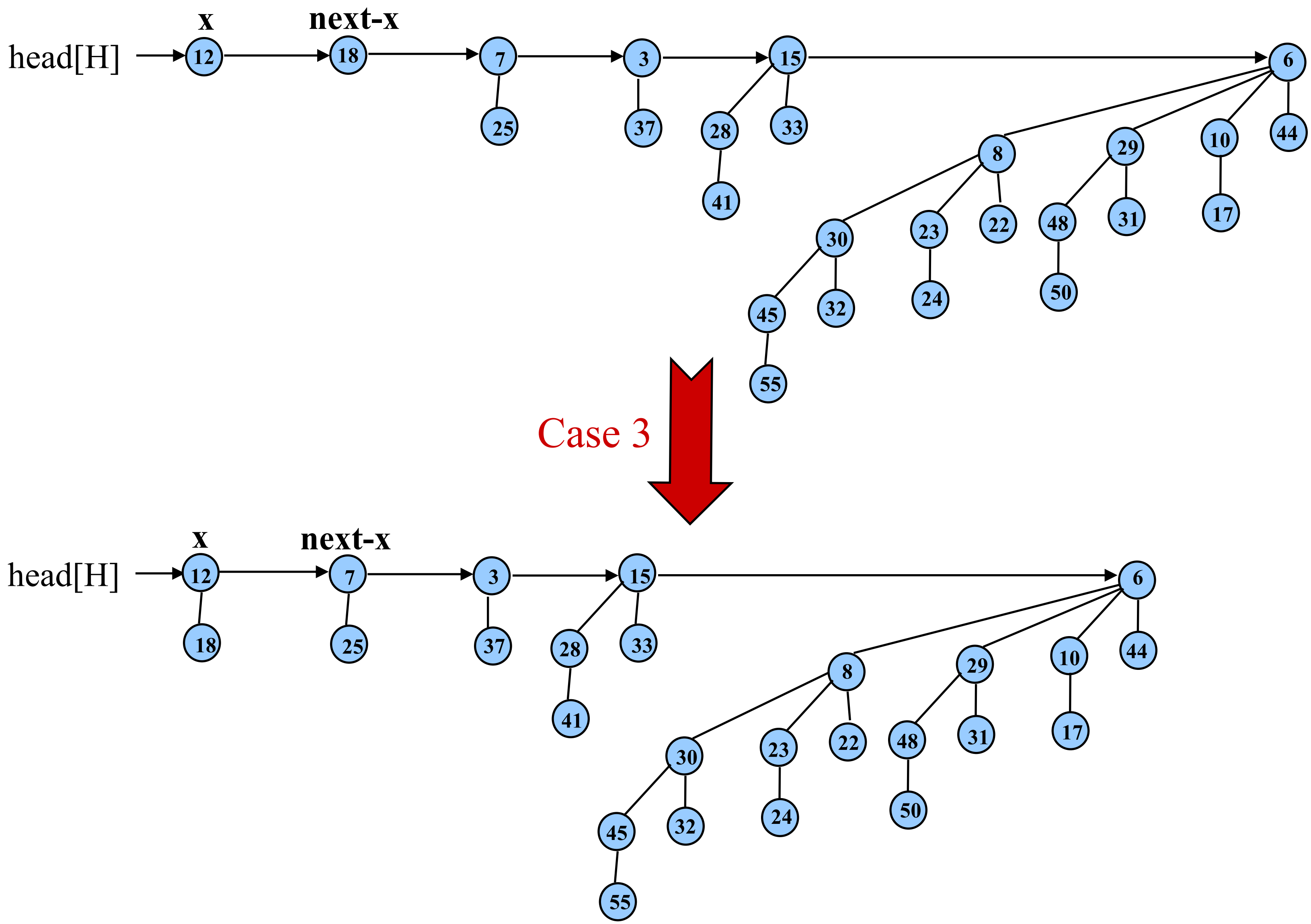
Code for Union

```
Union(H1, H2)
    H := new heap;
    head[H] := merge(H1, H2);          /* simple merge of root lists */
    if head[H] = NIL then return H fi;
    prev-x := NIL;
    x := head[H];
    next-x := sibling[x];
    while next-x ≠ NIL do
        if (degree[x] ≠ degree[next-x]) or
          (sibling[next-x] ≠ NIL and degree[sibling[next-x]] = degree[x]) then
Cases 1,2  prev-x := x;
           x := next-x;
        else
           if key[x] ≤ key[next-x] then
Case 3         sibling[x] := sibling[next-x];
               Link(next-x, x)
           else
Case 4         if prev-x = NIL then head[H] := next-x else sibling[prev-x] := next-x fi
               Link(x, next-x);
               x := next-x
           fi
        fi;
        next-x := sibling[x]
    od;
    return H
```

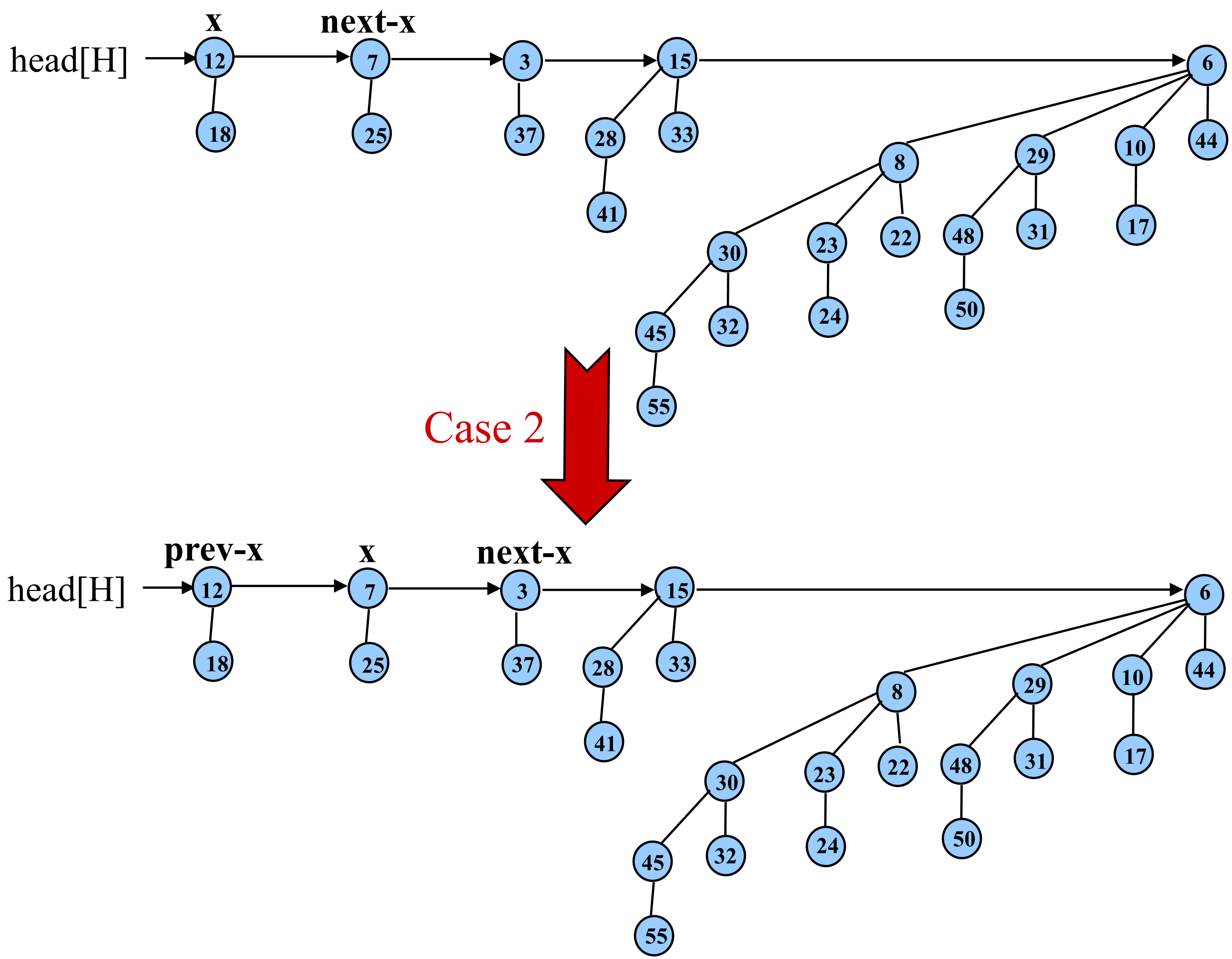
Union Example



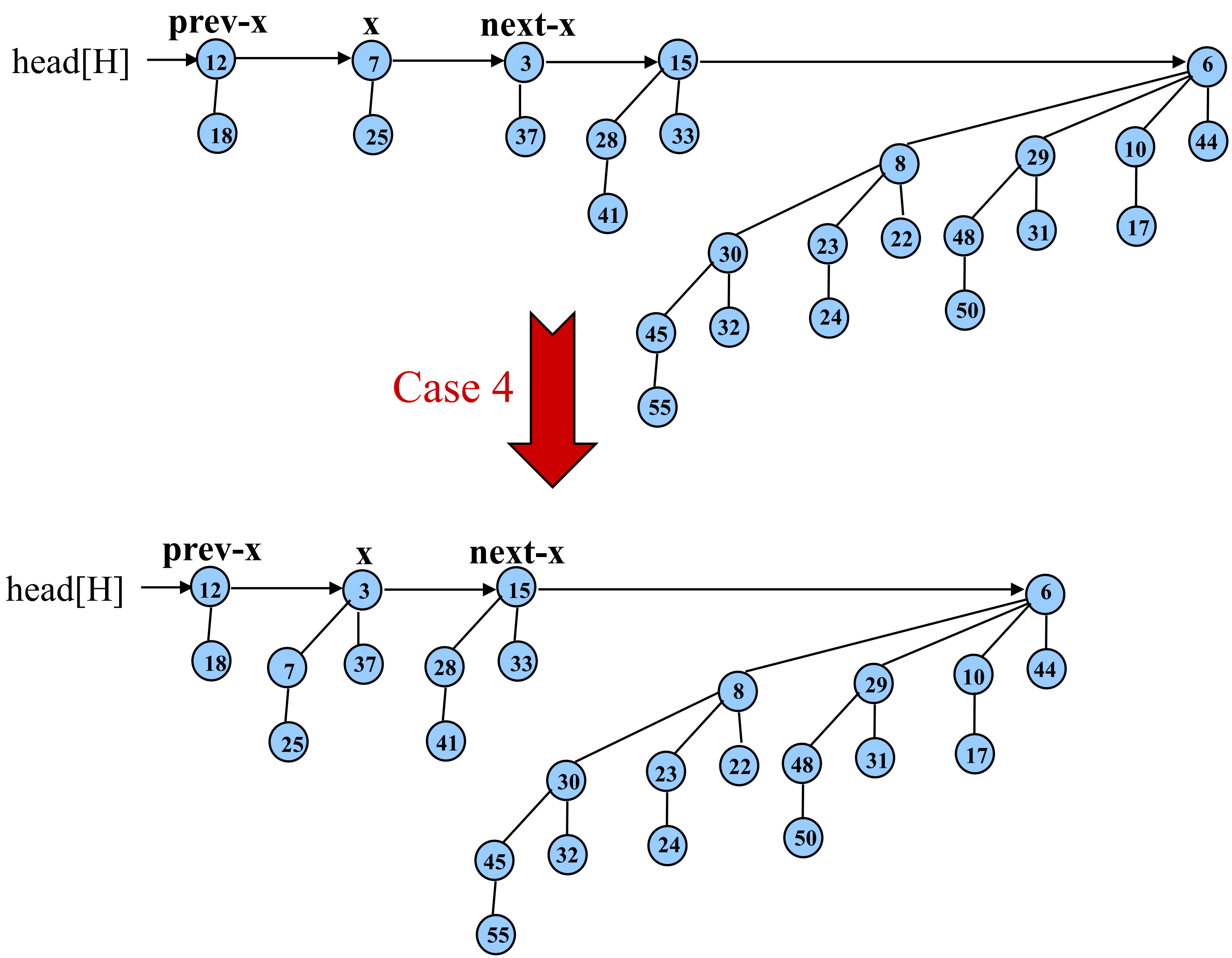
Union Example (Continued)



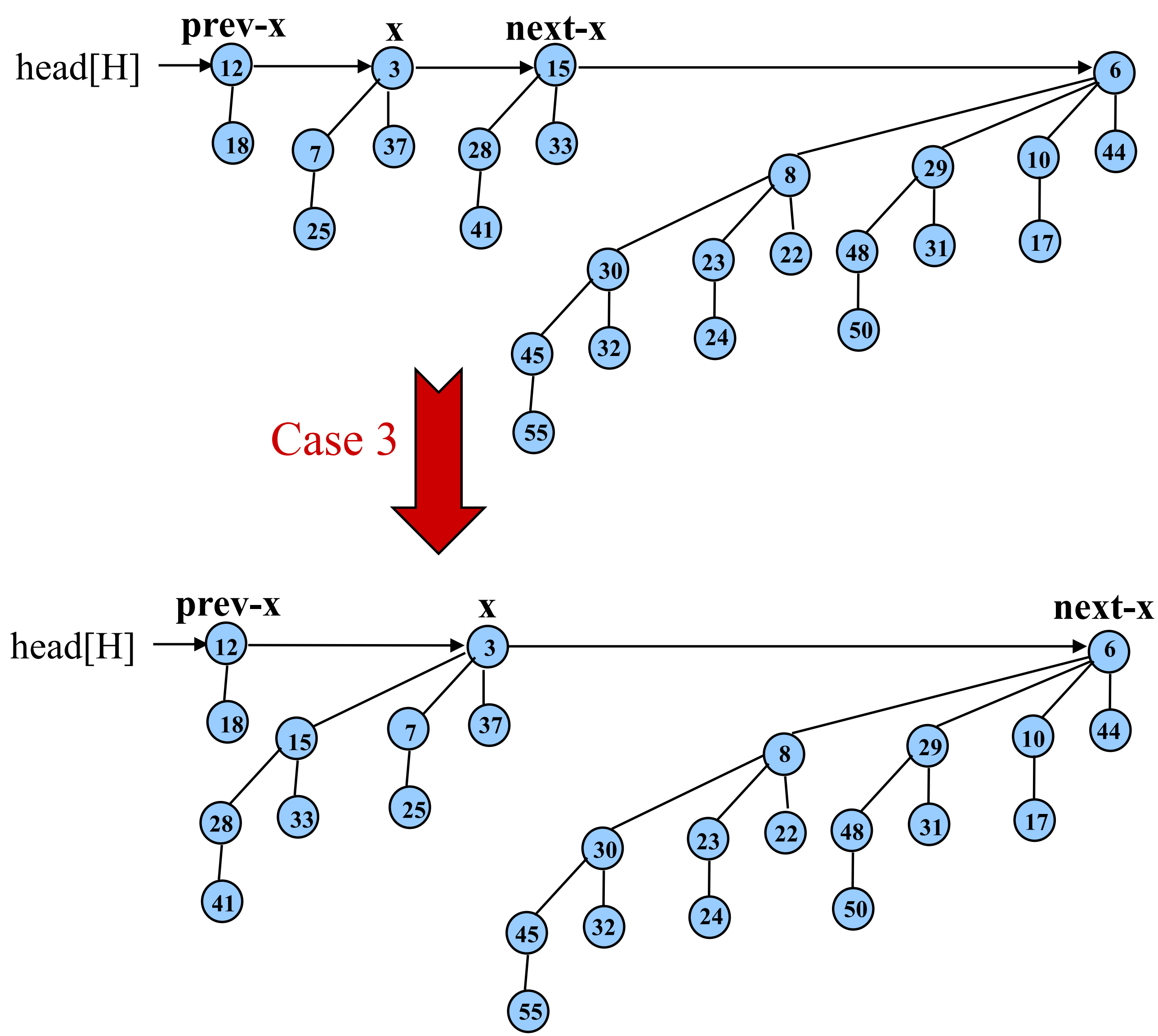
Union Example (Continued)



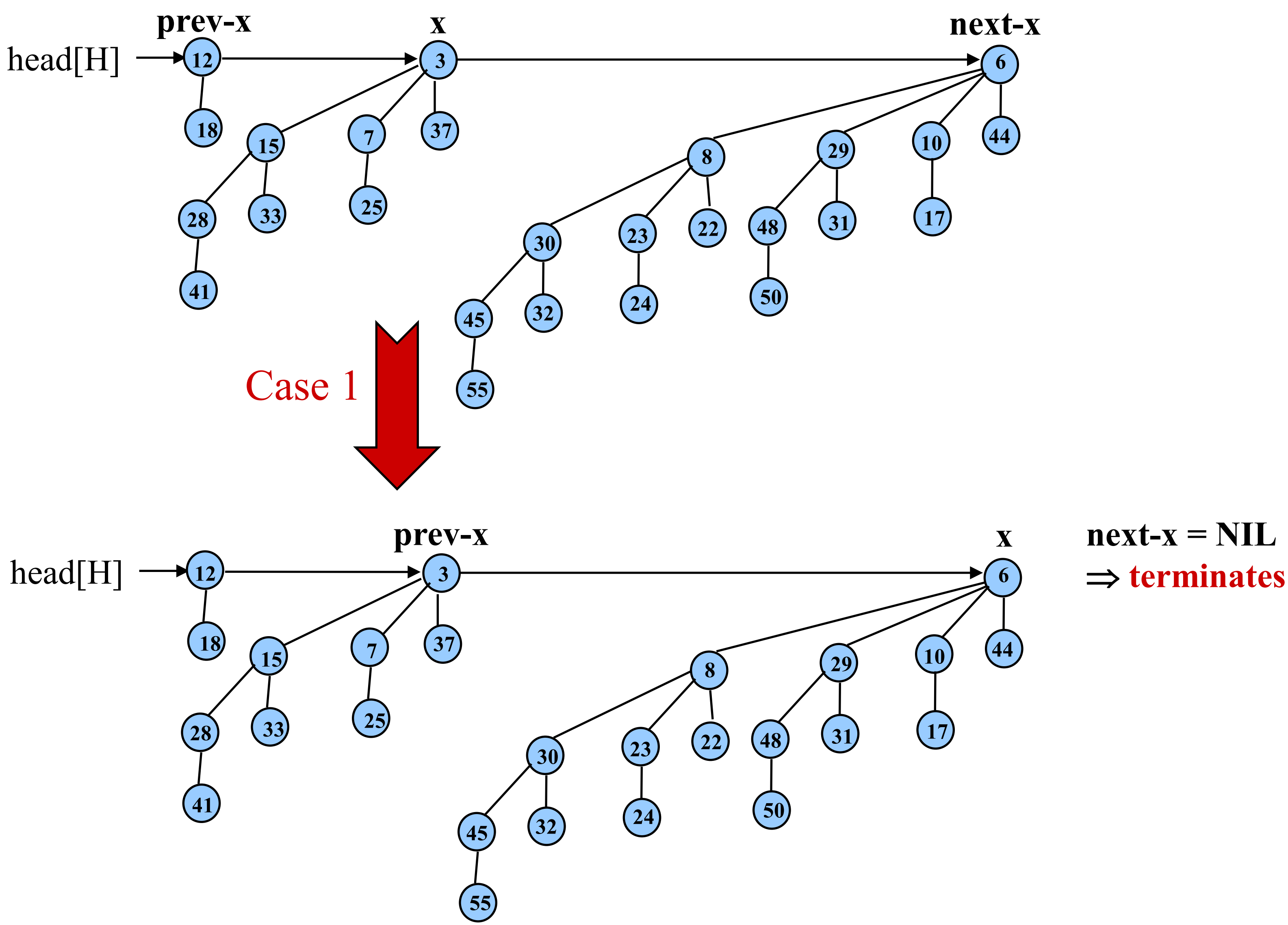
Union Example (Continued)



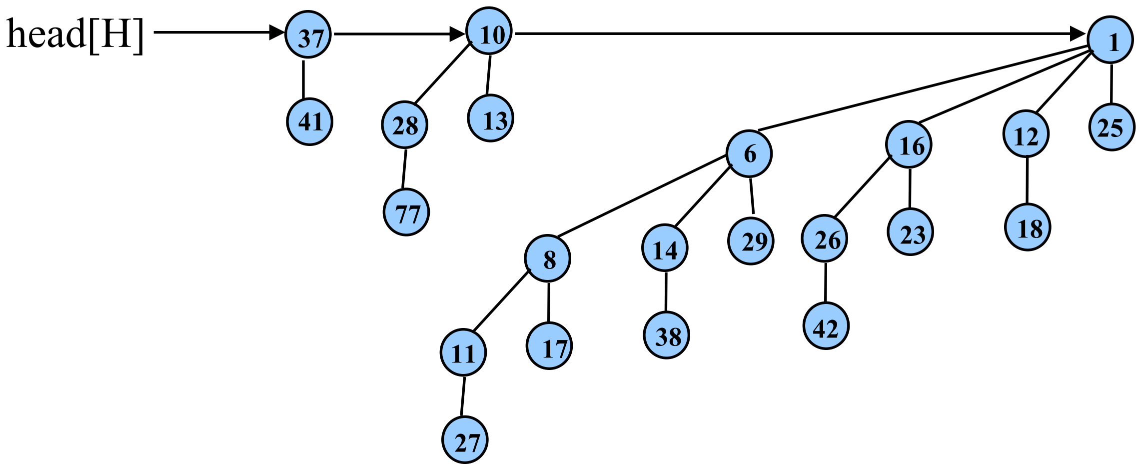
Union Example (Continued)

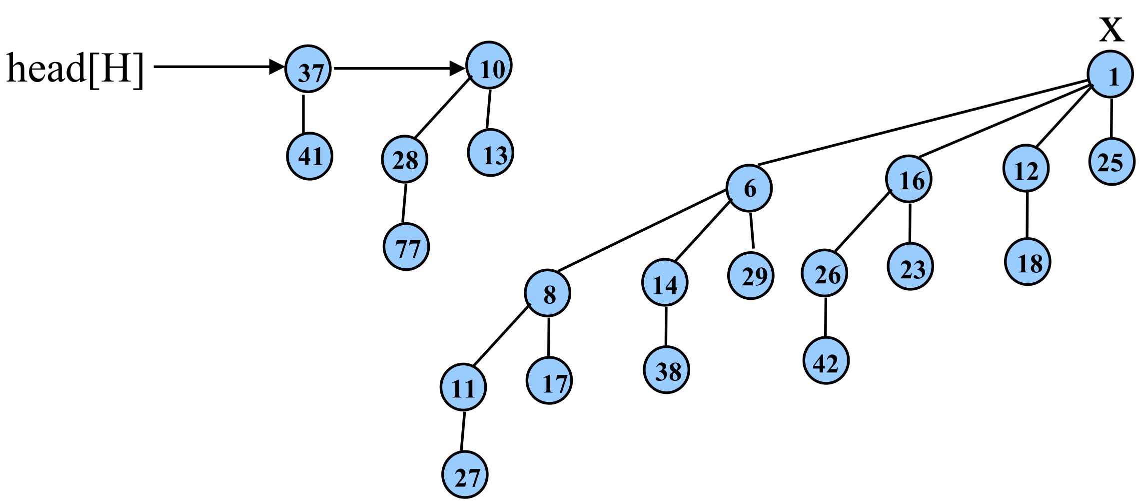


Union Example (Continued)

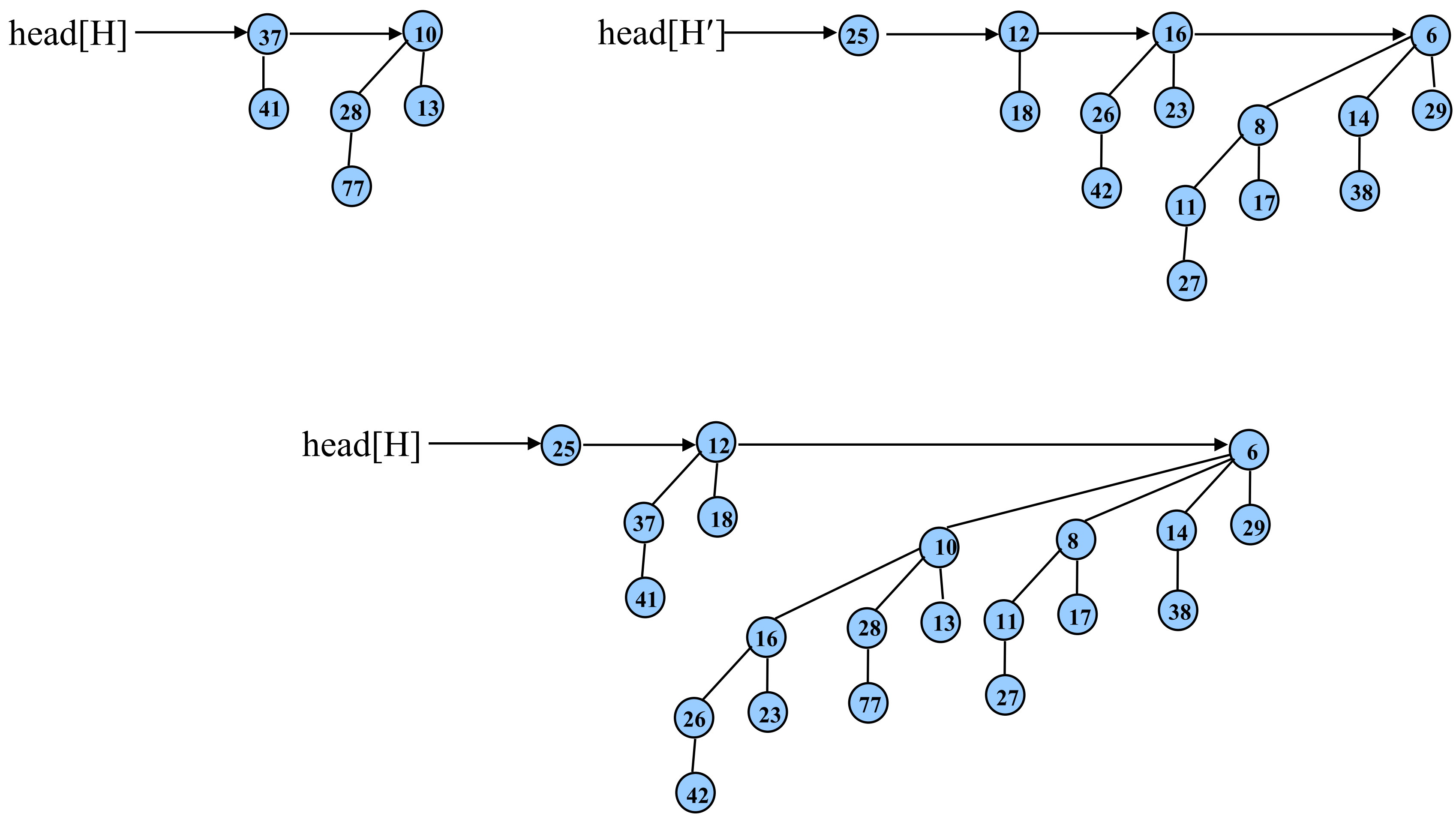


Extract-Min


head[H]
37
10
1
41
28
13
77
6
16
12
25
8
14
29
26
23
18
11
17
38
42
27


head[H]
37
10
x
1
41
28
13
77
6
16
12
25
8
14
29
26
23
18
11
17
38
42
27

Extract-Min (Continued)

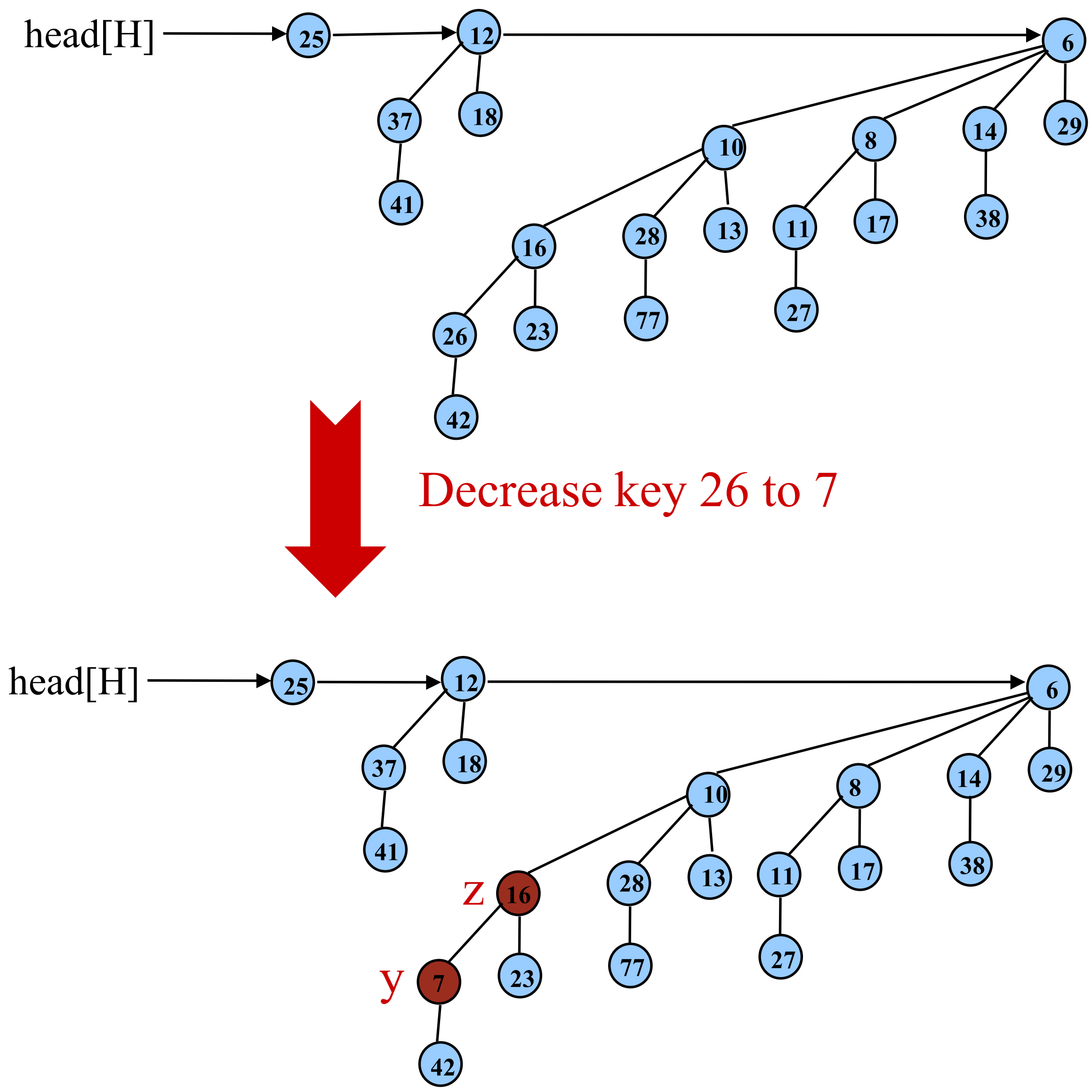


Insert and Extract-Min

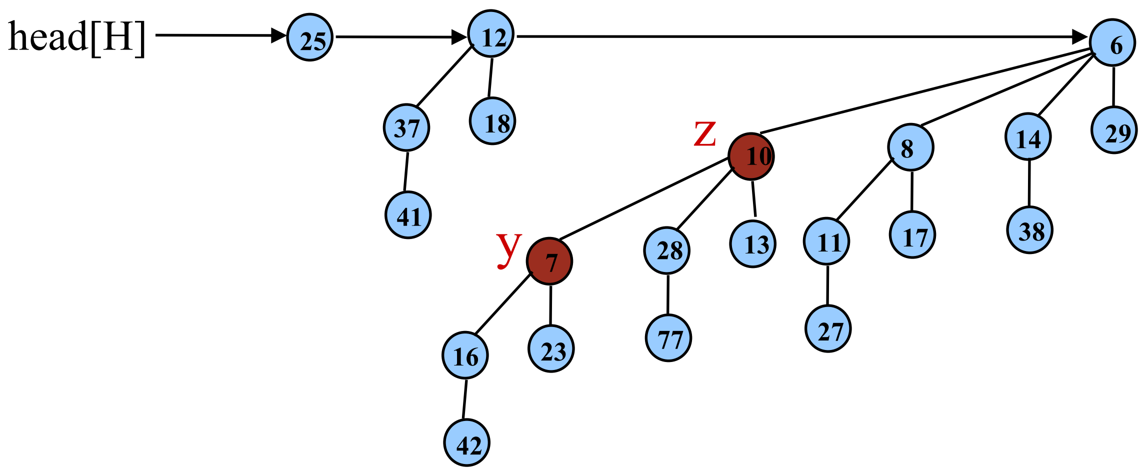
```
Insert(H, x)
        H′ := Make-B-H();
        p[x] := NIL;
        child[x] := NIL;
        sibling[x] := NIL;
        degree[x] := 0;
        head(H′) := x;
        H := Union(H, H′)
```

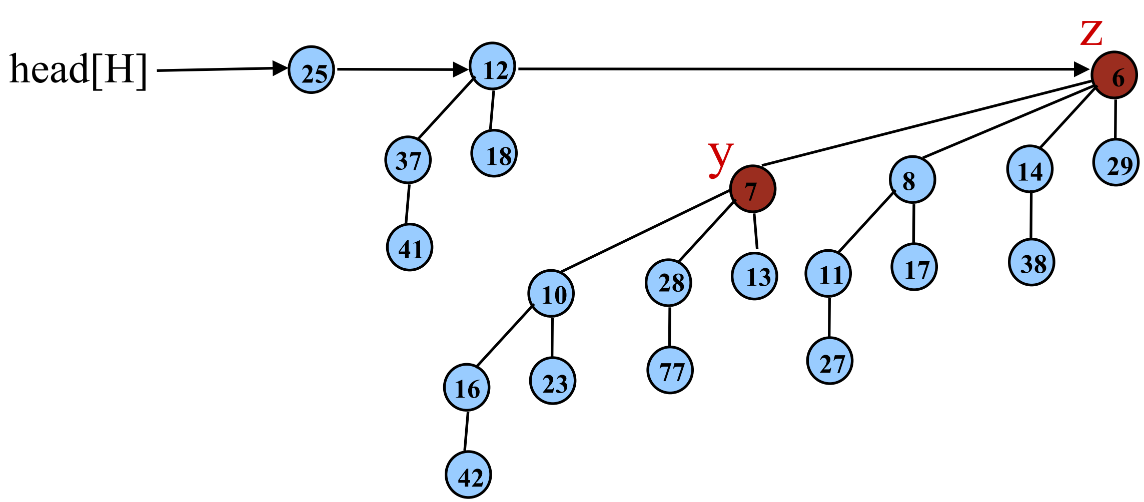
```
Extract-Min(H)
        remove minimum key root x from
          H's root list;
        H′ := Make-B-H();
        root list of H′ = x's children in
          reverse order;
        H := Union(H, H′);
        return x
```

Decrease-Key Example



Decrease-Key Example (Continue)





Decrease-Key

```
Decrease-Key(H, x, k)
    if k > key[x] then "error" fi;
    key[x] := k;
    y := x;
    z := p[y];
    while z ≠ NIL and key[y] < key[z] do
        exchange key[y] and key[z];
        y := z;
        z := p[y]
    od
```

Delete

```
Delete(H, x)
        Decrease-Key(H, x, –∞);
        Extract-Min(H)
```

Thank you