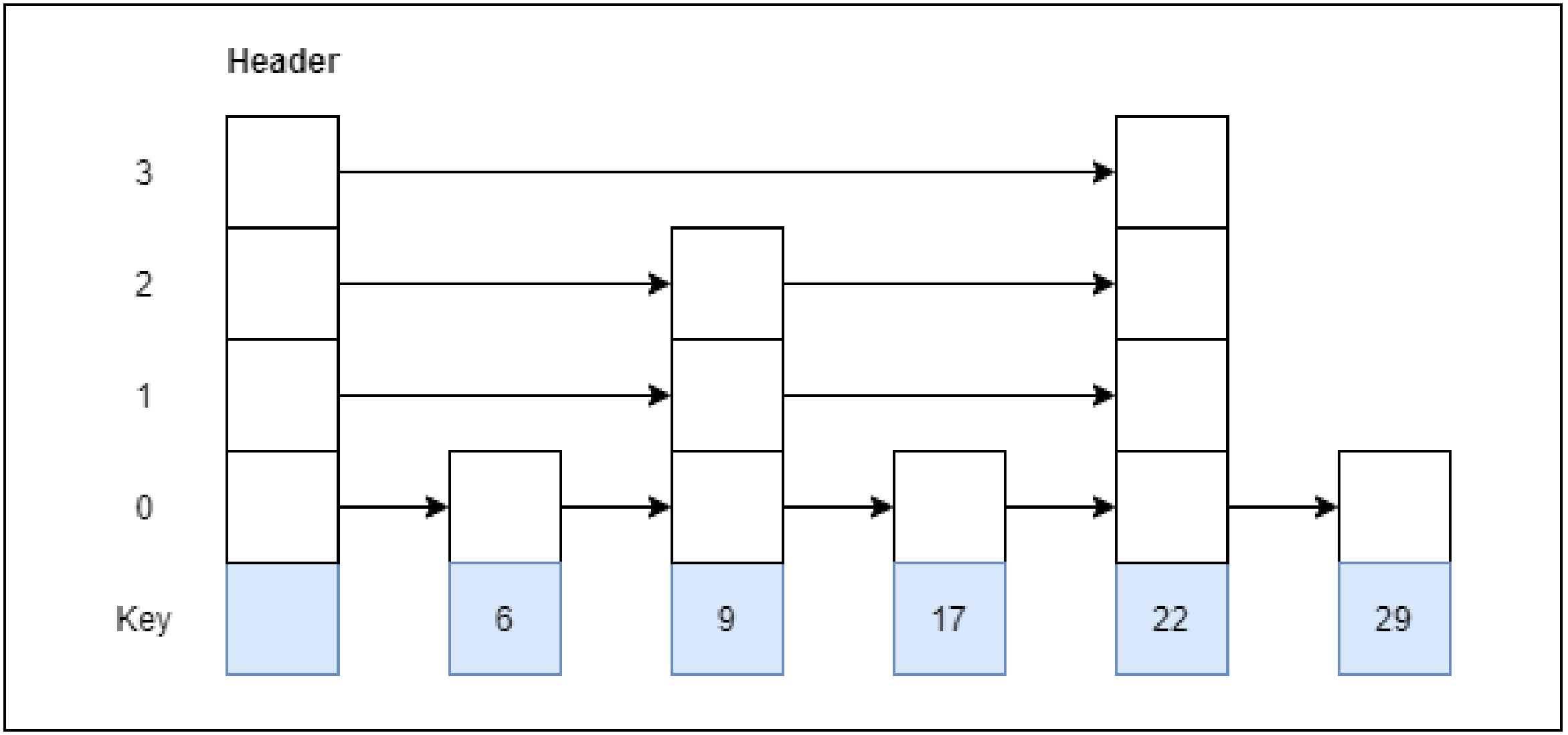
Skip List

Implementation

Agenda

- Skip List implementation in Python
- Insertion
- Search
- Deletion


Header
3
2
1
0
Key
6
9
17
22
29

Structure of Code

Class Node
//create Node

//__init__()
// which stores "Key"

and

// Forward List

Class SkipList
//create skip list object

//createNode

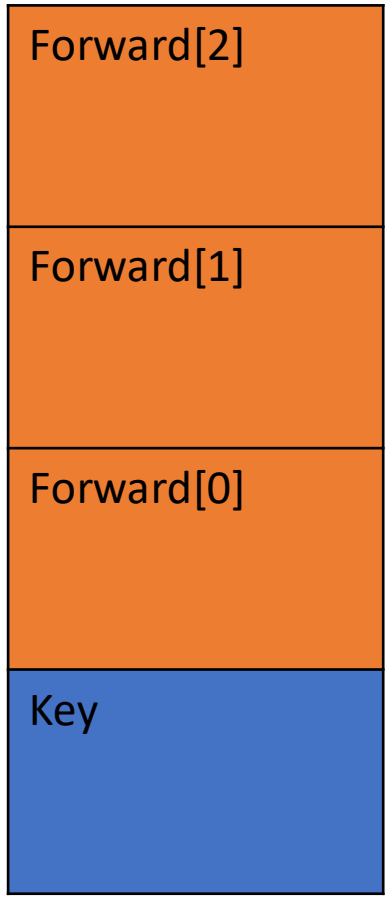
//randomLevel

//insertElement

//displayList

Node Class

```
5  class Node(object):
6      '''
7      Class to implement node
8      '''
9      def __init__(self, key, level):
10         self.key = key
11
12         # list to hold references to node of different level
13         self.forward = [None]*(level+1)
14
15
```



forward

Forward[0] None	Forward[1] None	Forward[2] None	Forward[3] None

randomLevel

```
Random Level:
    level := 1
    //random() that returns a random value in [0...1]

    while random() < p and level < MaxLevel do:
        level := level + 1

    return level
```

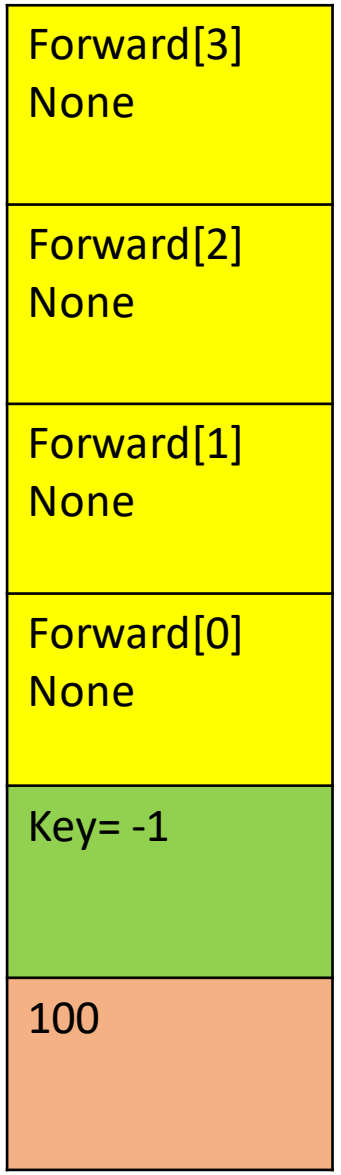
randomLevel

```
    # create random level for node
    def randomLevel(self):
        lvl = 0
        while random.random() < self.P and lvl < self.MAXLVL:
            lvl += 1
        return lvl
```

SkipList Class

```
class SkipList(object):
    '''
    Class for Skip list
    '''
    def __init__(self, max_lvl, P):
        # Maximum level for this skip list
        self.MAXLVL = max_lvl

        # P is the fraction of the nodes with level
        # i references also having level i+1 references
        self.P = P

        # create header node and initialize key to -1
        self.header = self.createNode(self.MAXLVL, -1)

        # current level of skip list
        self.level = 0
```

lst = SkipList(3, 0.5)



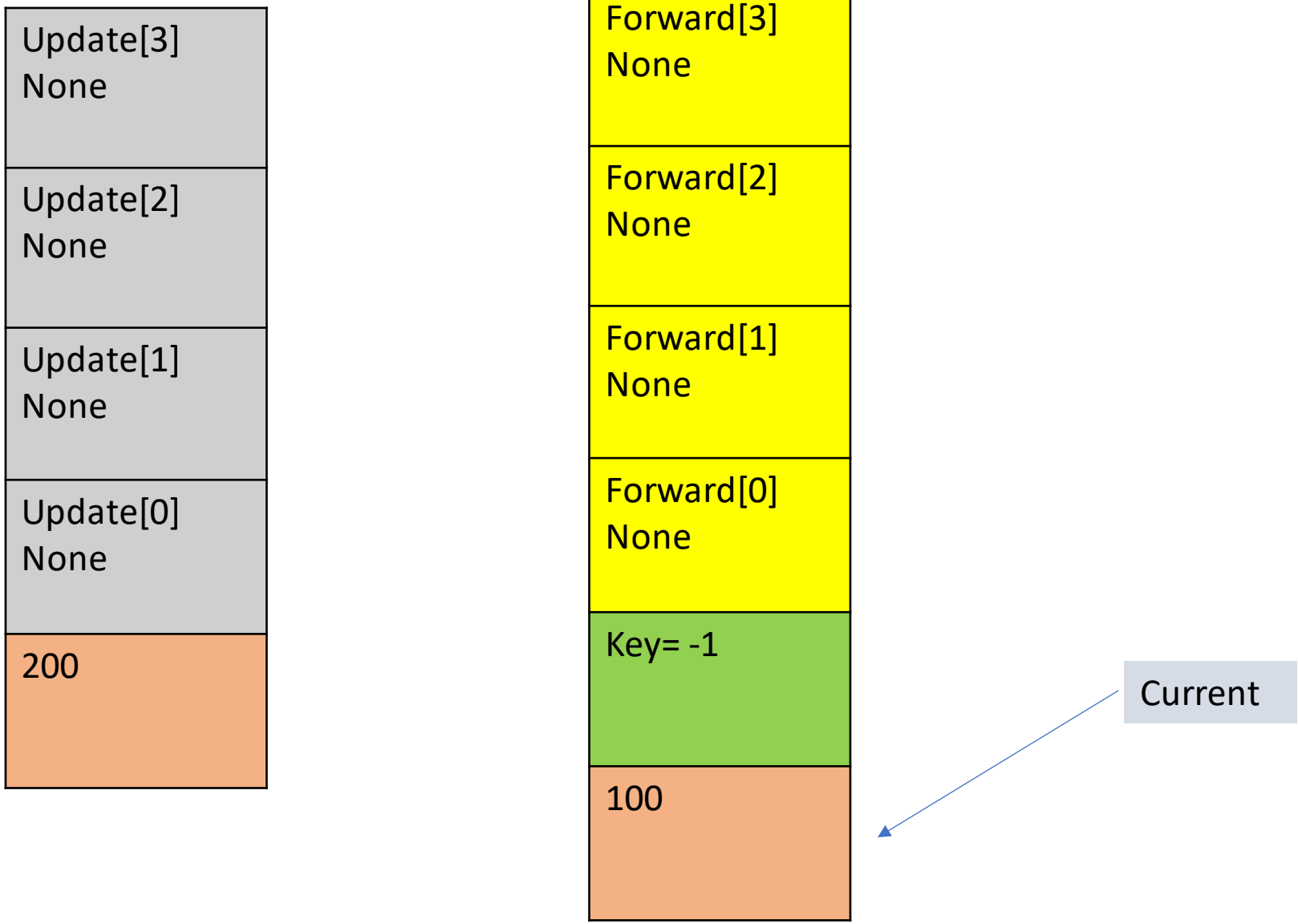
MAXLVL = 3

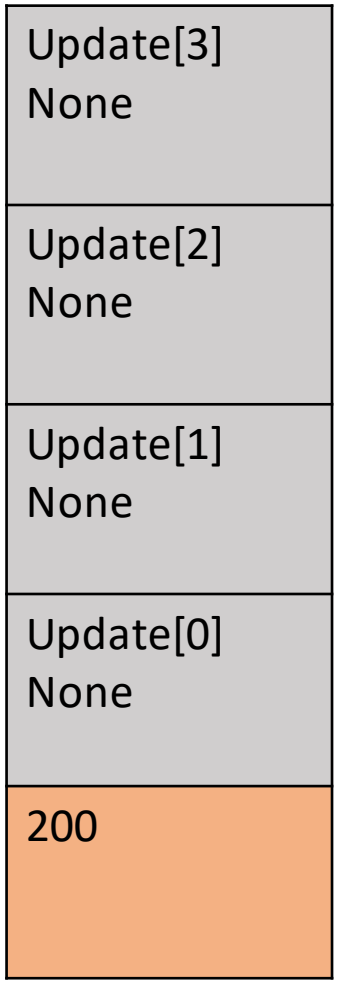
P=0.5

Level =0

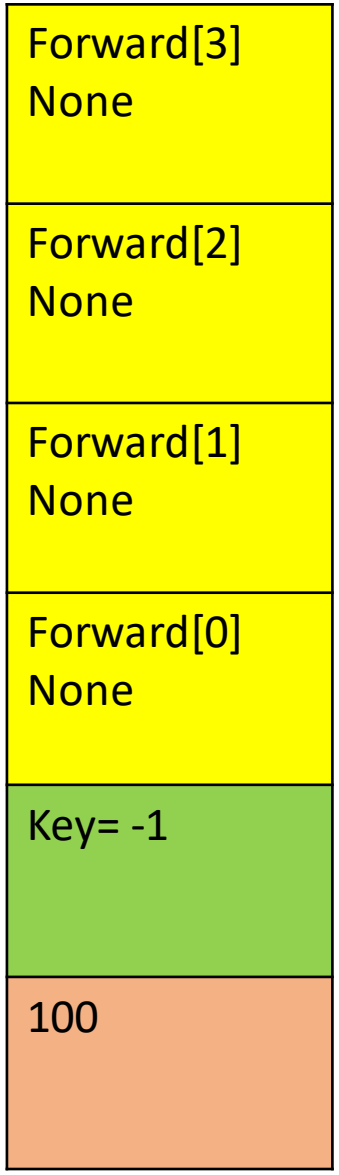
InsertElement(3)

```
    # insert given key in skip list
    def insertElement(self, key):
        # create update array and initialize it
        update = [None] * (self.MAXLVL + 1)
        current = self.header
```


Update[3]
None
Update[2]
None
Update[1]
None
Update[0]
None
200
Forward[3]
None
Forward[2]
None
Forward[1]
None
Forward[0]
None
Key= -1
100
Current



Update

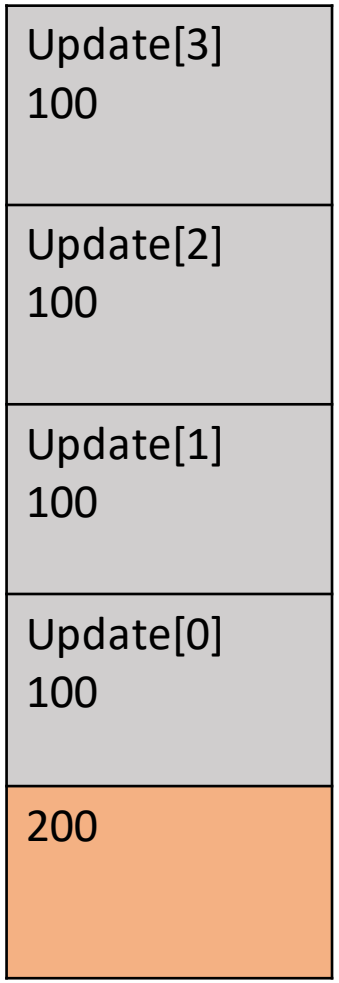


Header

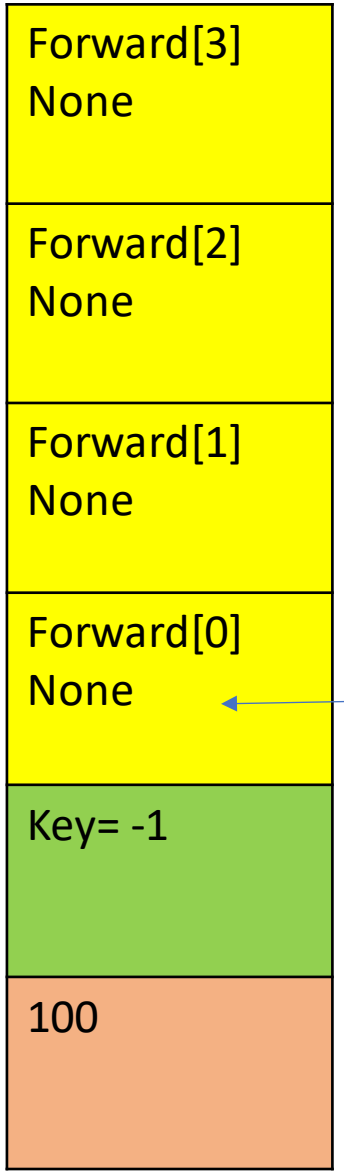
Current

```
for i in range(self.level, -1, -1):
    while current.forward[i] and current.forward[i].key < key:
        current = current.forward[i]
    update[i] = current
```

For = i in range(3, 2, 1, 0)



Update



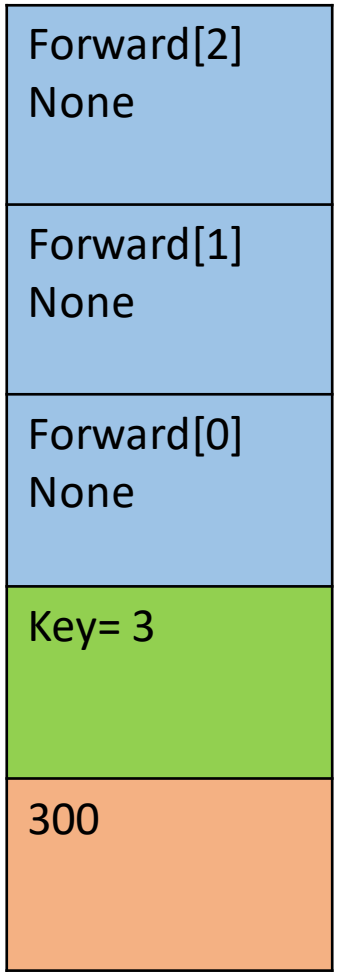
Header

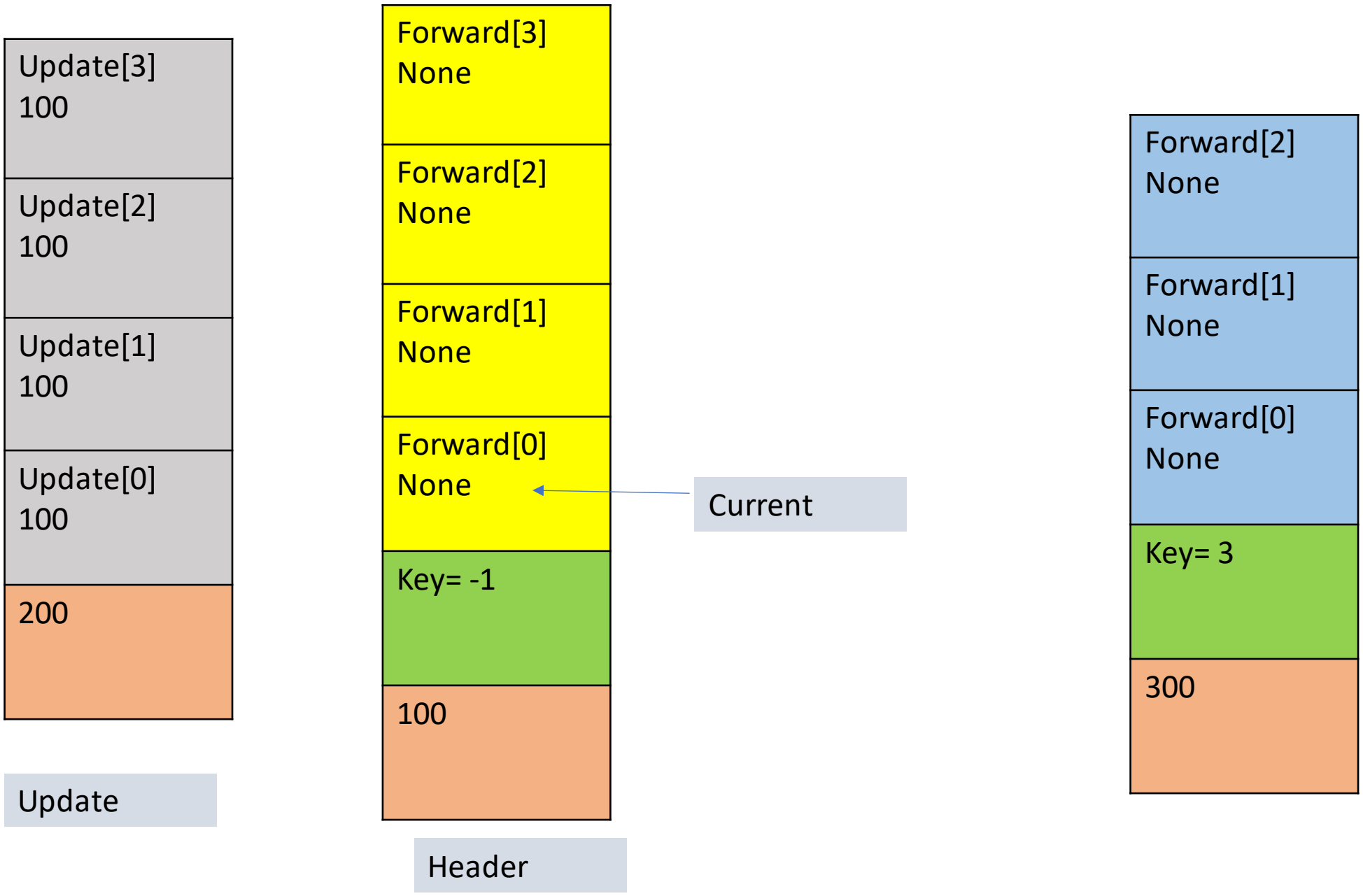
current = current.forward[0]

Current

```
if current == None or current.key != key:
    # Generate a random level for node
    rlevel = self.randomLevel()

    '''
    If random level is greater than list's current
    level (node with highest level inserted in
    list so far), initialize update value with reference
    to header for further use
    '''
    if rlevel > self.level:
        for i in range(self.level + 1, rlevel + 1):
            update[i] = self.header
        self.level = rlevel

    # create new node with random level generated
    n = self.createNode(rlevel, key)
```

rlevel = 2




Update[3]
100
Update[2]
100
Update[1]
100
Update[0]
100
200
Update
Forward[3]
None
Forward[2]
None
Forward[1]
None
Forward[0]
None
Key= -1
100
Header
Current
Forward[2]
None
Forward[1]
None
Forward[0]
None
Key= 3
300

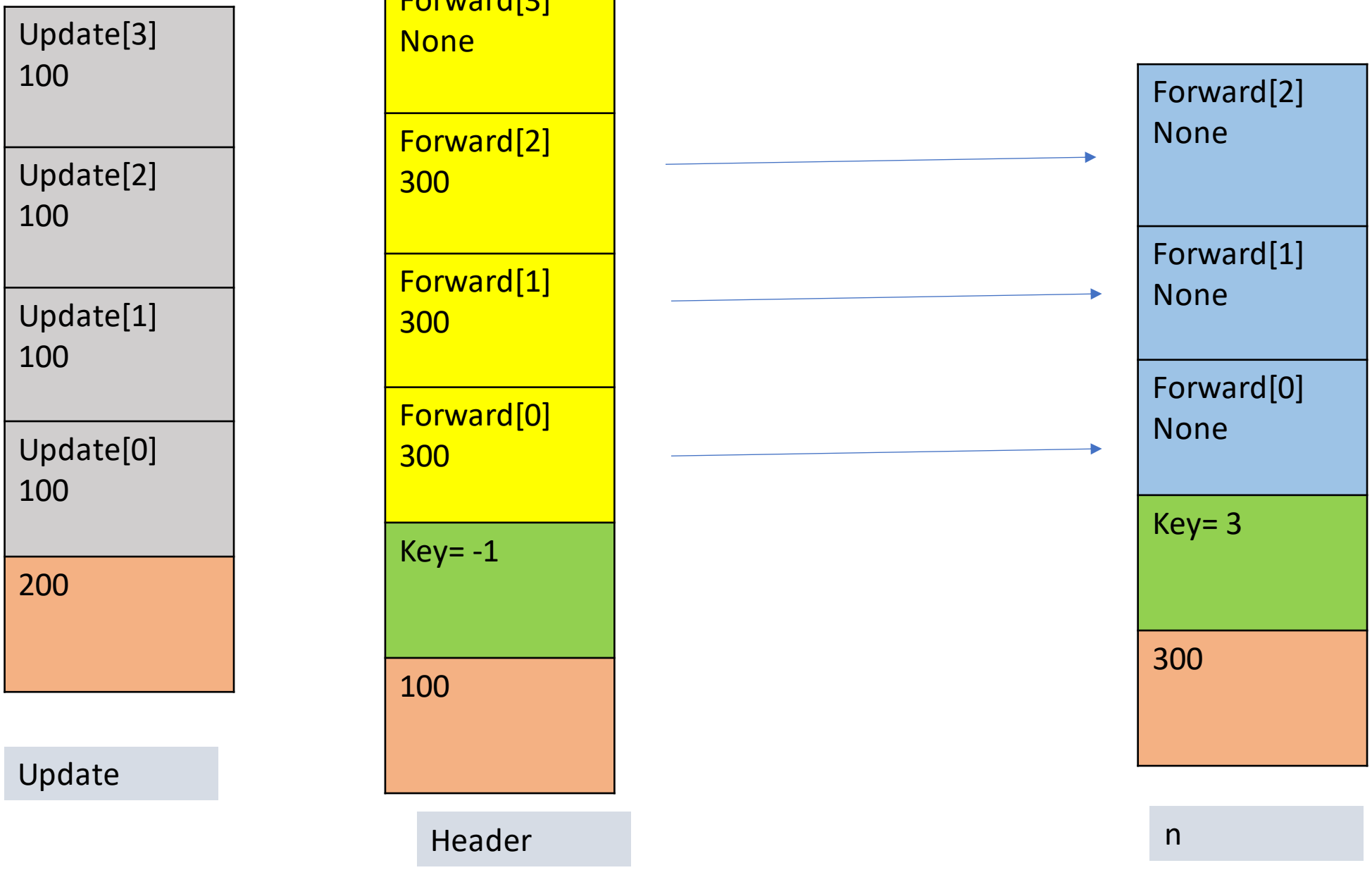
```
    # insert node by rearranging references
    for i in range(rlevel + 1):
        n.forward[i] = update[i].forward[i]
        update[i].forward[i] = n
```

For i in range (2 +1),

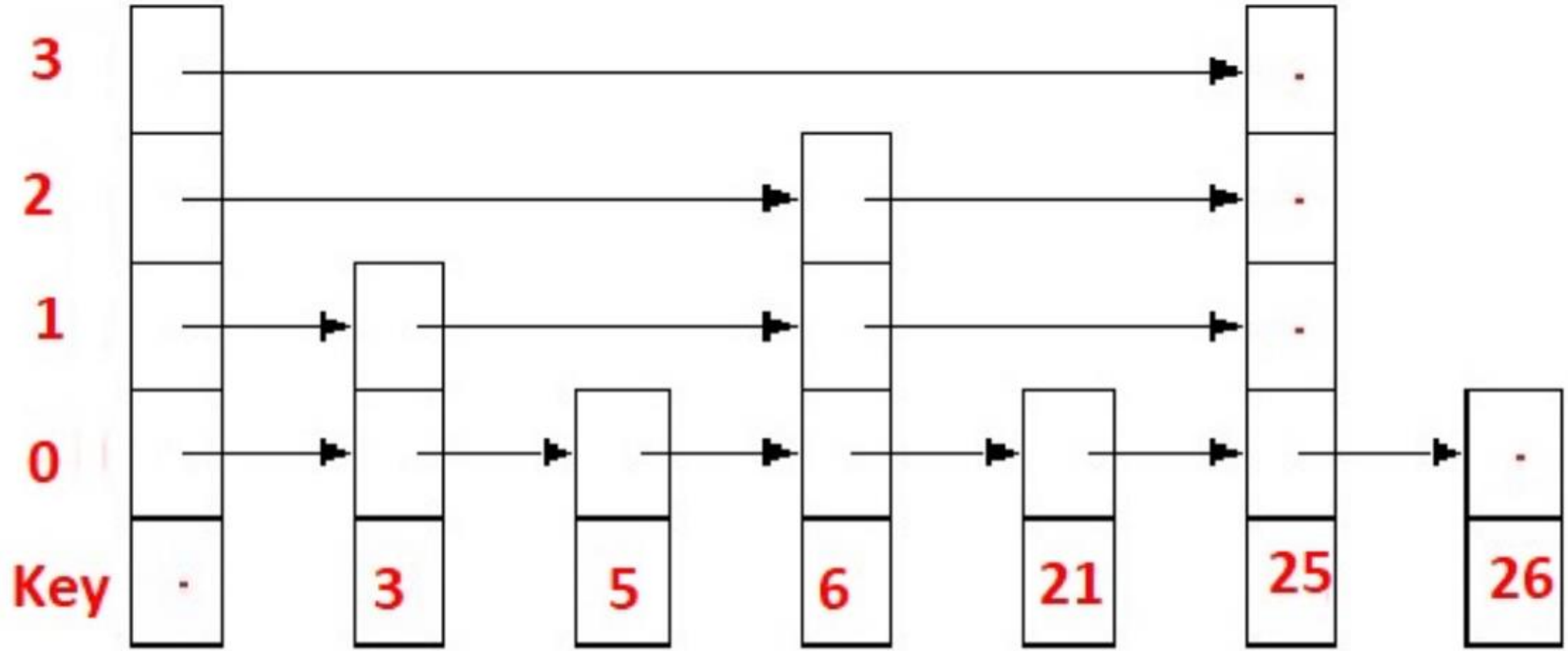
For i in range (0, 1, 2):

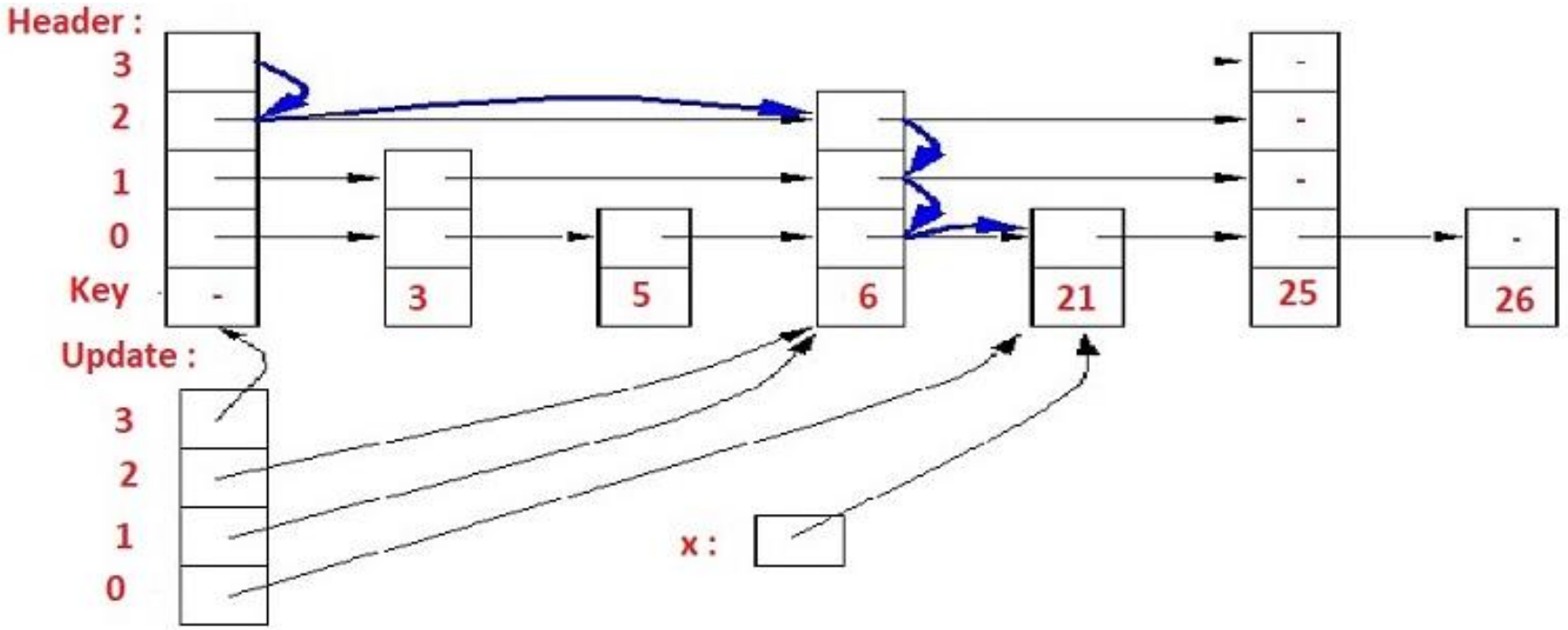
n.f(0) = u(0).f(0)
= 100.f(0)
= None

U(0).f(0) = n


Update[3]
100
Update[2]
100
Update[1]
100
Update[0]
100
200
Update
Forward[3]
None
Forward[2]
300
Forward[1]
300
Forward[0]
300
Key= -1
100
Header
Forward[2]
None
Forward[1]
None
Forward[0]
None
Key= 3
300
n

Header :



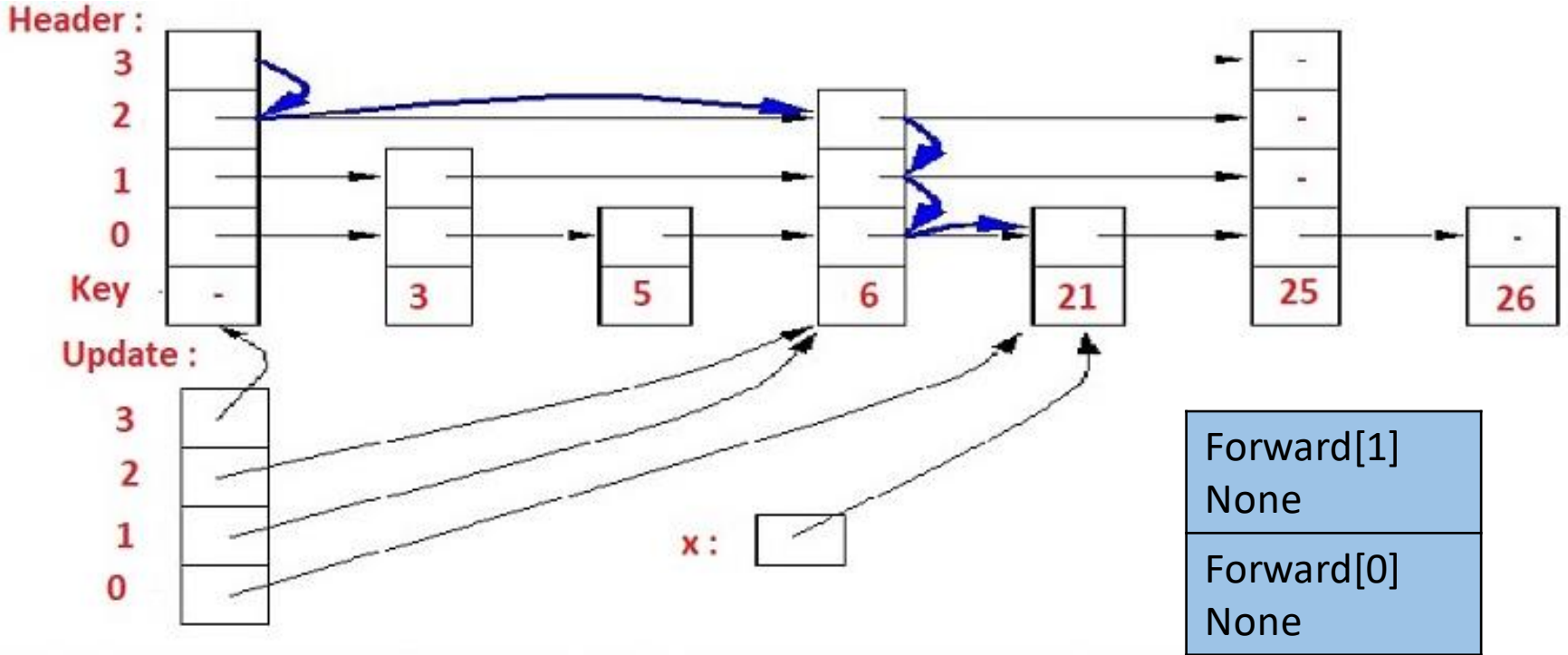


Update[3]
Update[2]
Update[1]
Update[0]
200

```
for i in range(self.level, -1, -1):
    while current.forward[i] and current.forward[i].key < key:
        current = current.forward[i]
    update[i] = current
```

```
if rlevel > self.level:
    for i in range(self.level + 1, rlevel + 1):
        update[i] = self.header
    self.level = rlevel
```

- Rlevel =1
- 1 > 3
- Ignore



Forward[1] None
Forward[0] None
Key= 3
2100

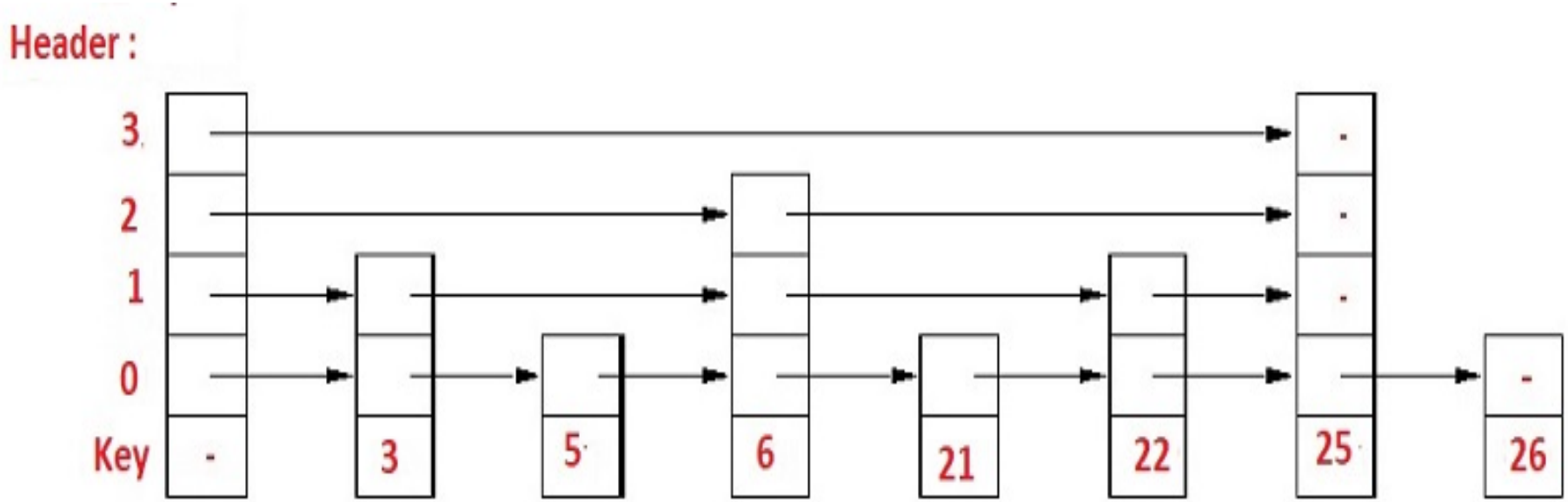
For i in range (1 +1),

For i in range (0, 1):

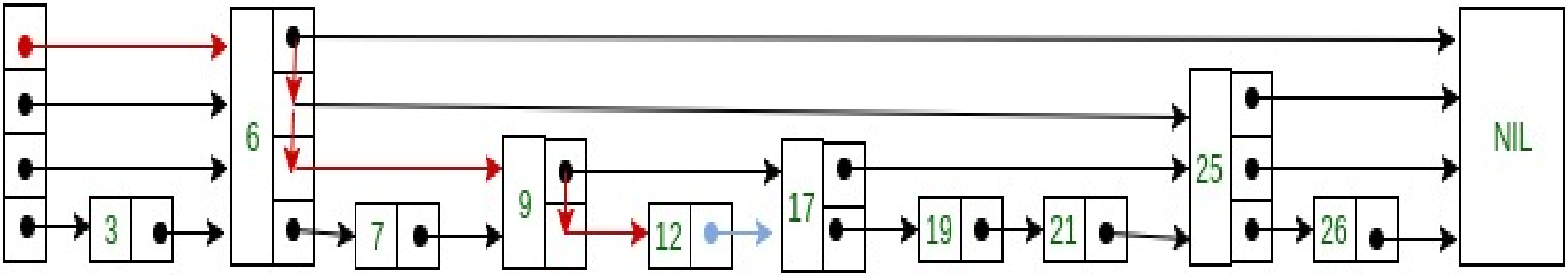
i=0
n.f(0) = u(0).f(0)
U(0).f(0) = n

i = 1
n.f(1) = u(1).f(1)
U(1).f(1) = n

```
    # insert node by rearranging references
    for i in range(rlevel + 1):
        n.forward[i] = update[i].forward[i]
        update[i].forward[i] = n
```

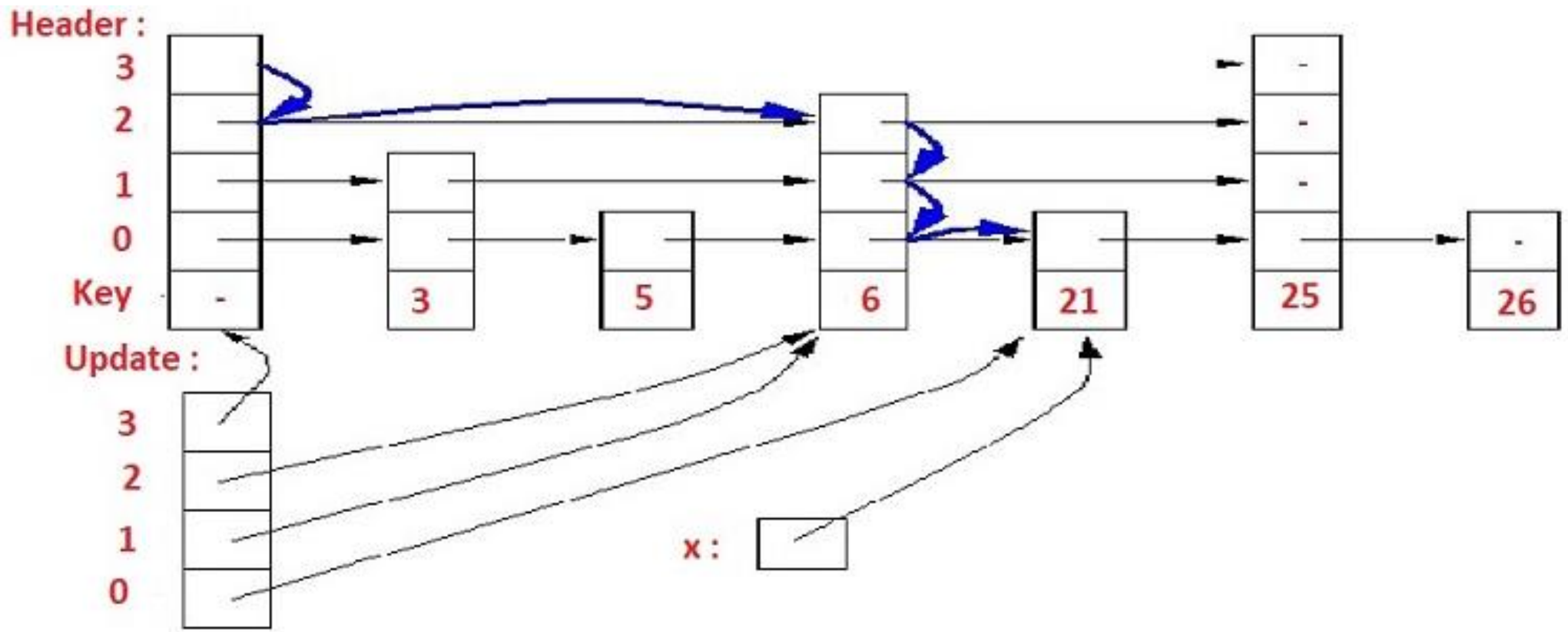

Header :
3
2
1
0
Key
-
3
5
6
21
22
25
26

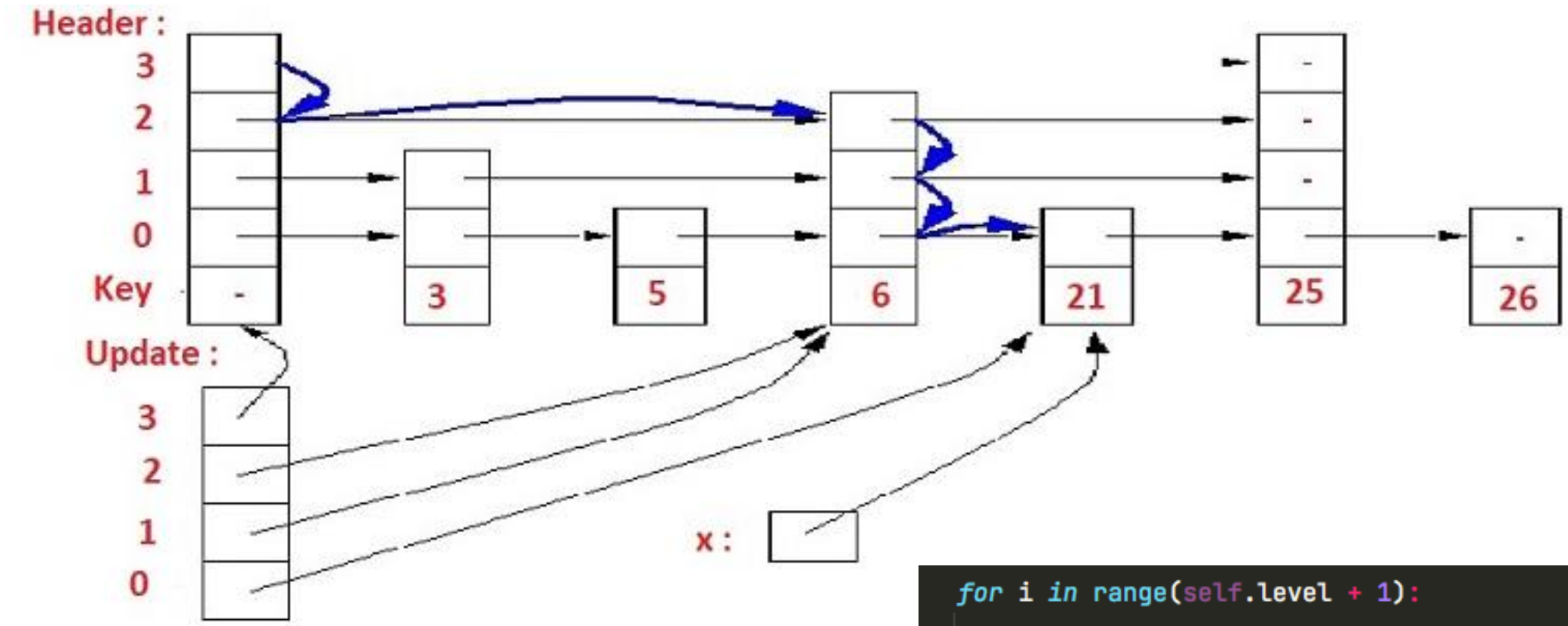
Search



```
def searchElement(self, key):
    current = self.header

    for i in range(self.level, -1, -1):
        while (current.forward[i] and current.forward[i].key < key):
            current = current.forward[i]

    # reached level 0 and advance reference to
    # right, which is prssibly our desired node
    current = current.forward[0]

    # If current node have key equal to
    # search key, we have found our target node
    if current and current.key == key:
        print("Found key ", key)
```

Delete






```
for i in range(self.level + 1):

    '''
    If at level i, next node is not target
    node, break the loop, no need to move
    further level
    '''
    if update[i].forward[i] != current:
        break
    update[i].forward[i] = current.forward[i]
```

Thank you