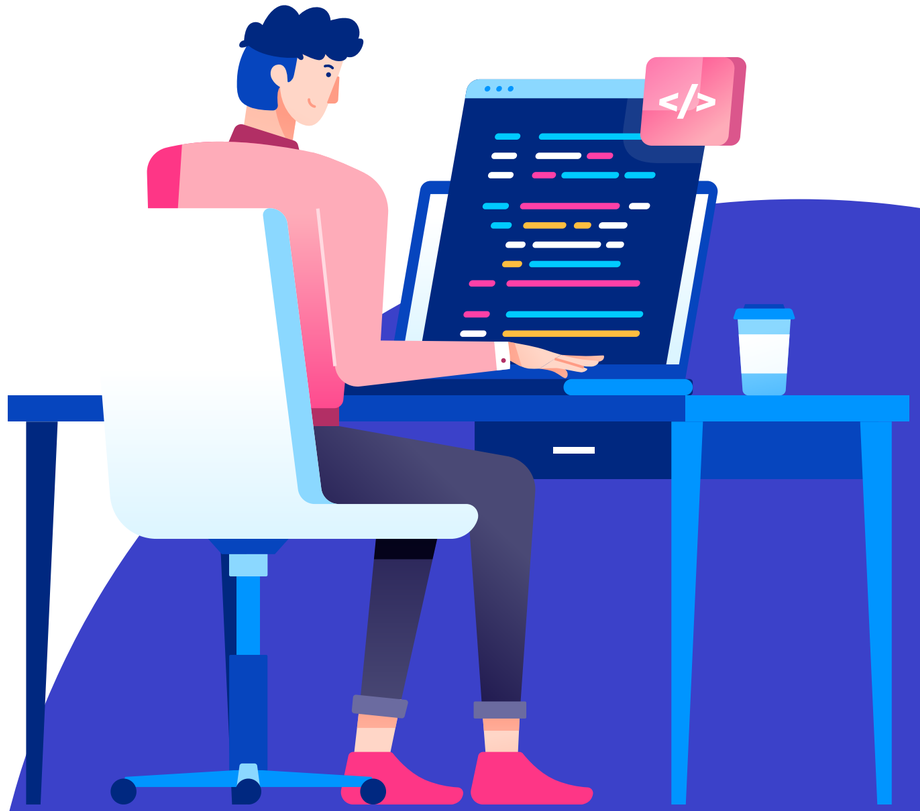
SKIP LIST

TEAM JC



TEAM JC

Search . . .

INTRODUCTION

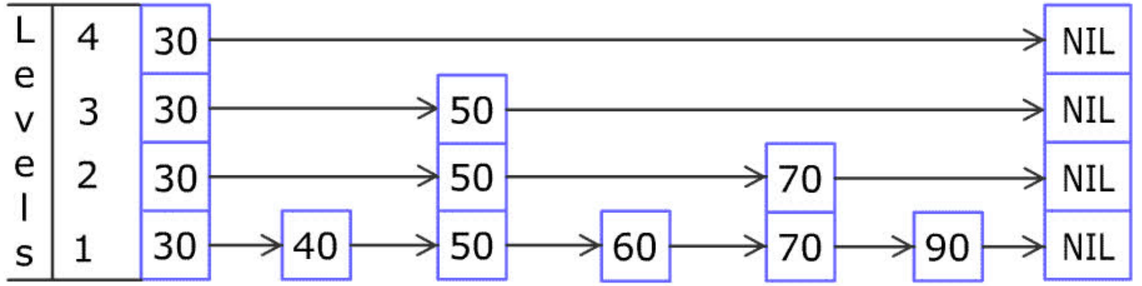
TEAM JC

Search . . .

WHAT IS A

TEAM JC

WHAT IS A SKIP LIST ?



DESCRIPTION

“Skip lists are a probabilistic data structure that seem likely to supplant balanced trees as the implementation method of choice for many applications. Skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, and faster.”

—**William Pugh**, Concurrent Maintenance of Skip Lists (**1989**)

ADDITIONAL INFORMATION

In computer science, a skip list is a probabilistic data structure that allows **O(log n)** search complexity as well as **O(log n)** insertion complexity within an ordered sequence of n elements. Thus, it can get the best features of a sorted array (for searching) while maintaining a linked list-like structure that allows insertion, which is not possible with a static array.

USAGE

Skip lists are used for efficient statistical computations of running medians (also known as moving medians).

Skip lists are also used in distributed applications (where the nodes represent physical computers, and pointers represent network connections) and for implementing highly scalable concurrent priority queues with less lock contention, or even without locking, as well as lock-free concurrent dictionaries.

WHO IS WILLIAM PUGH

TEAM JC

Search . . .

WILLIAM WORTHINGON PUGH JR.

An American Computer Scientist who invented the **Skip List** in **1989** at 29 years of age.

Also known as

"BILL" PUGH

TEAM JC

Search . . .

TIME AND SPACE COMPLEXITY

Search . . .

TIME COMPLEXITY

AVERAGE

ACCESS

O(log n)

SEARCH

O(log n)

INSERT

O(log n)

DELETE

O(log n)

TIME COMPLEXITY

WORST

ACCESS

O(n)

SEARCH

O(n)

INSERT

O(n)

DELETE

O(n)

SPACE COMPLEXITY

WORST

$O(n \log n)$

TEAM JC

Search . . .

VARIATIONS OF A SKIP LIST

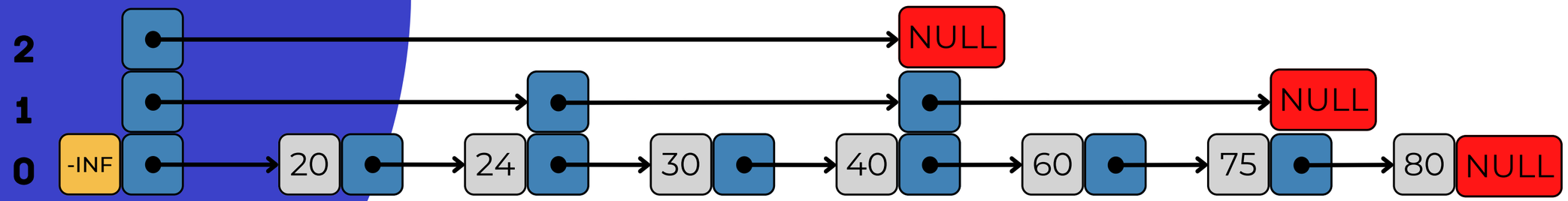
TEAM JC

Search . . .

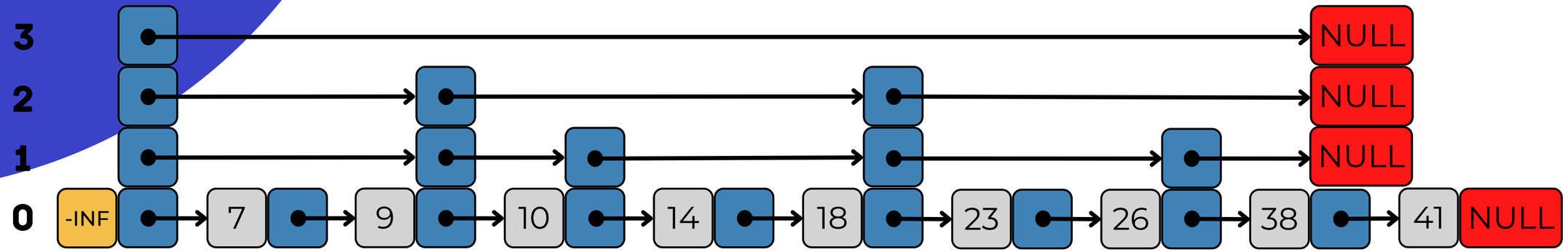
01 PERFECT SKIP LIST

02 RANDOMIZED SKIP LIST

TEAM JC



PERFECT SKIP LIST



RANDOMIZED SKIP LIST

PERFECT SKIP LIST

- **Keys** are in sorted order
- **O(log n)** levels
- **Header** node and element pointing to **NULL** are in every level
- Each higher level contains 1 / 2 the elements of the level below it
- Too structured to support **efficient** updates such as deletion and insertion

RANDOMIZED SKIP LIST

- **Keys** are in sorted order
- **O(log n)** levels
- **Header** node and element pointing to **NULL** are in every level
- Allows for some **imbalance** like in AVL trees
- randomized to support **efficient** updates (as it should be)

TEAM JC

Search . . .

WHY IS IT RANDOMIZED?

The level of an element is chosen in essence by flipping a coin. For example, we could think of flipping a coin until it comes up tails. We count the number of times the coin came up heads before coming up tails and add 1 to this number. This number represents the level of the element.

TEAM JC

Search . . .

DATA STRUCTURE OF SKIP LIST

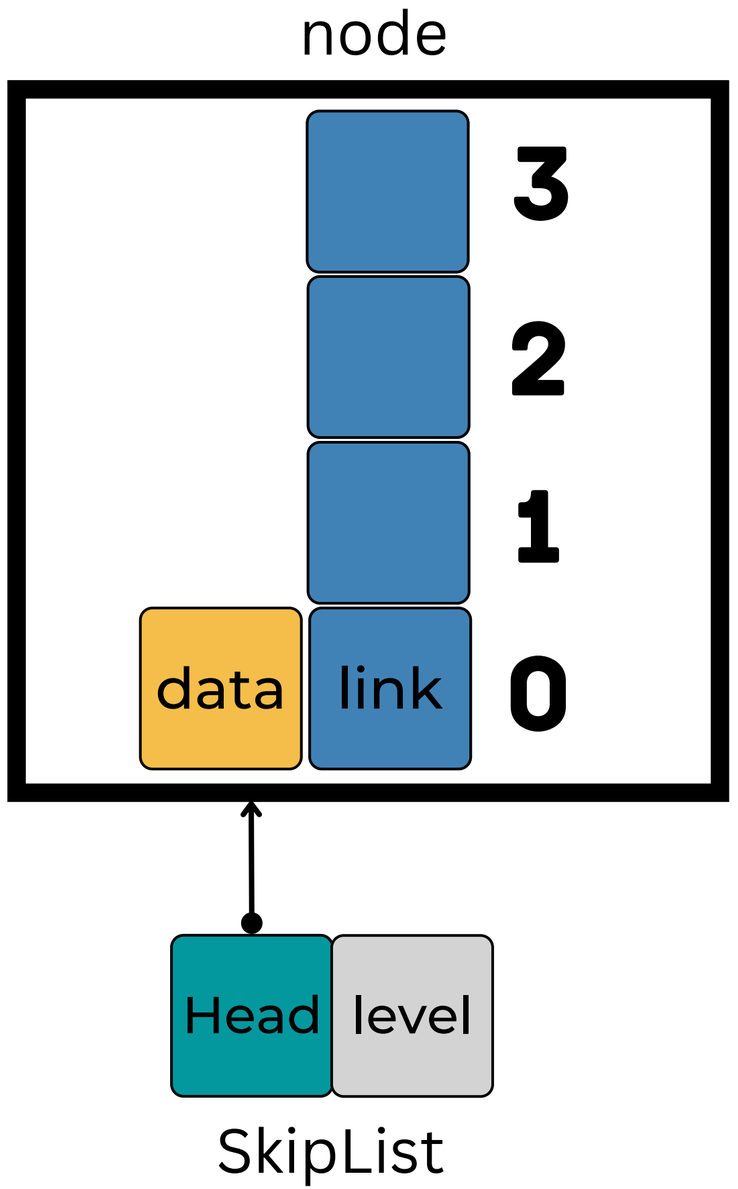
TEAM JC

DATA STRUCTURE

```
#define MAXLEVEL 3

typedef struct node{
 struct node **link;
 int data;
}*Node;

typedef struct {
 Node head;
 int level;
}SkipList;
```



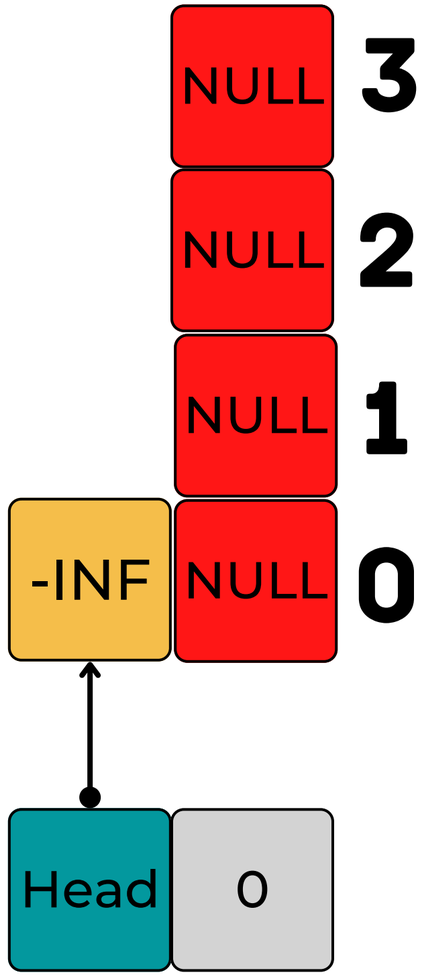
Search . . .

TEAM JC

INITIALIZATION

Search . . .

Set Level to 0;
Allocate memory to Head;
Allocate memory to Link array with the size based on max level;
Set data to -INF;
Set all value of Link to NULL;



Max Level = 3

TEAM JC

Search . . .

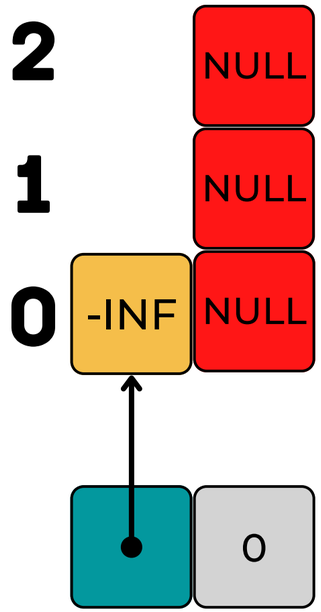
SIMULATION OF CODE

Randomized Skiplist

INSERTION

Max Level = 2

Level = 0

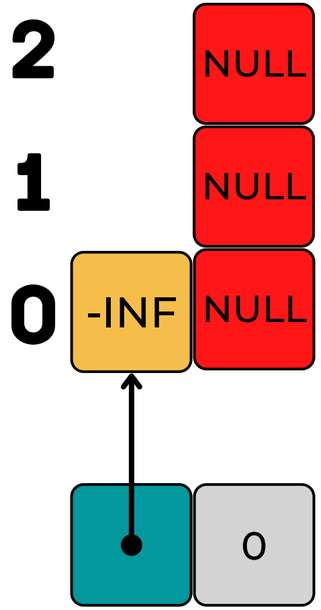


INSERTION

Max Level = 2

Level = 0

Update = 



INSERT(30)

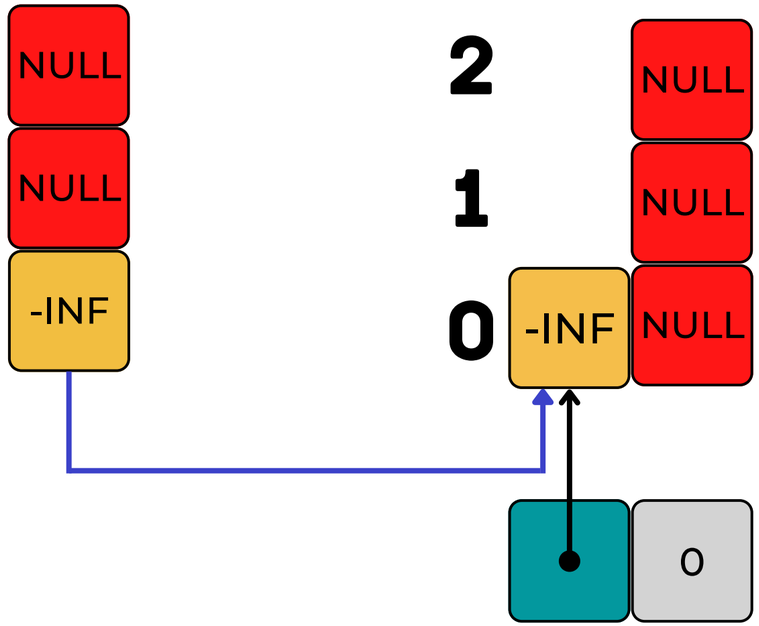
INSERTION

Max Level = 2

Level = 0

since link is NULL? **Move down a level**

Update =



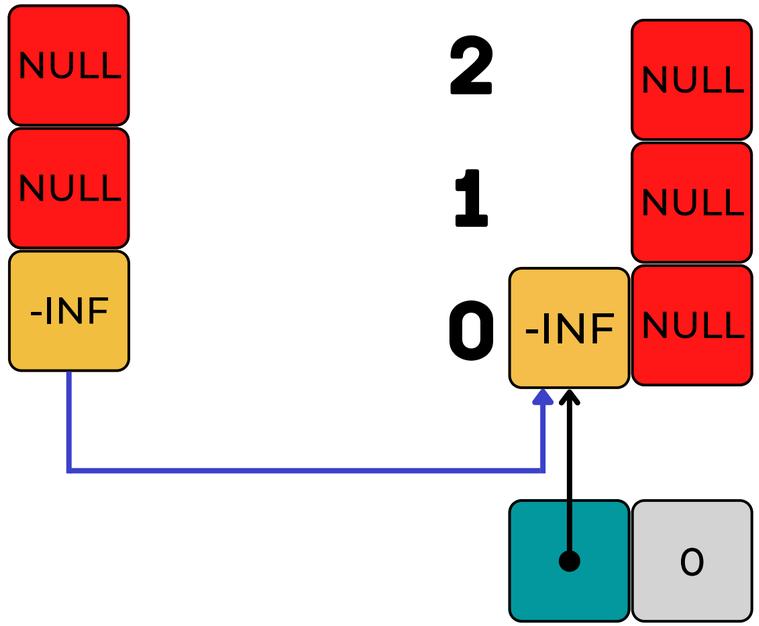
INSERT(30)

INSERTION

Max Level = 2

Level = -1

Update =



INSERT(30)

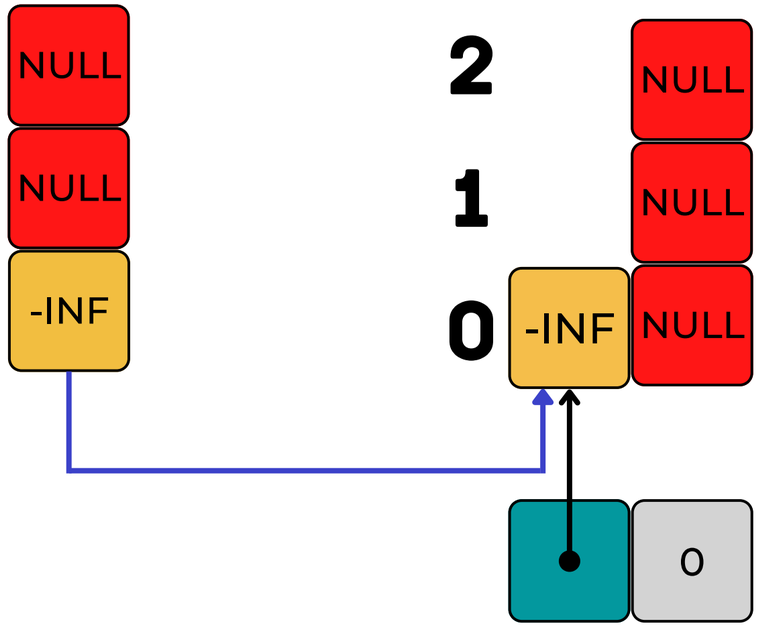
TEAM JC

Search . . .

INSERTION

Max Level = 2

Update =



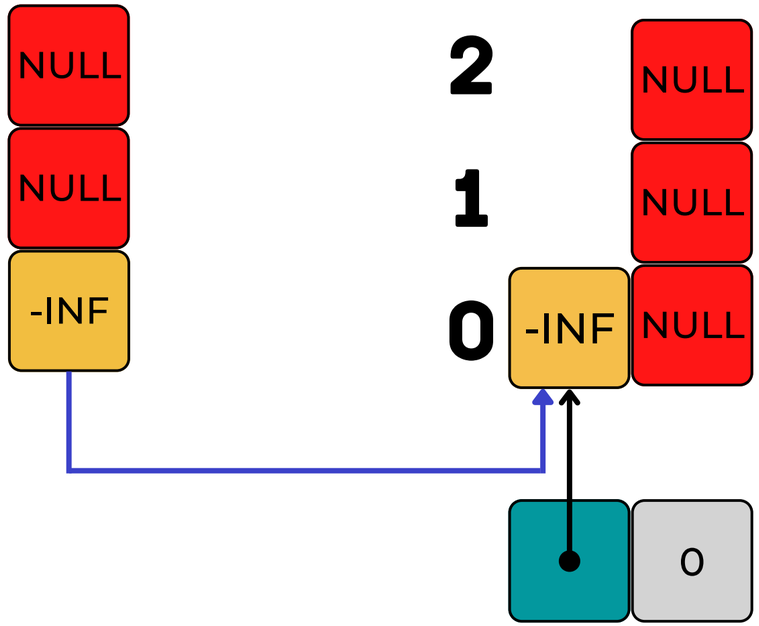
do coin flip;
while coin is heads;
  increment new level;
stop when coin is tails
  return new level;

INSERT(30)

INSERTION

New Level = 2

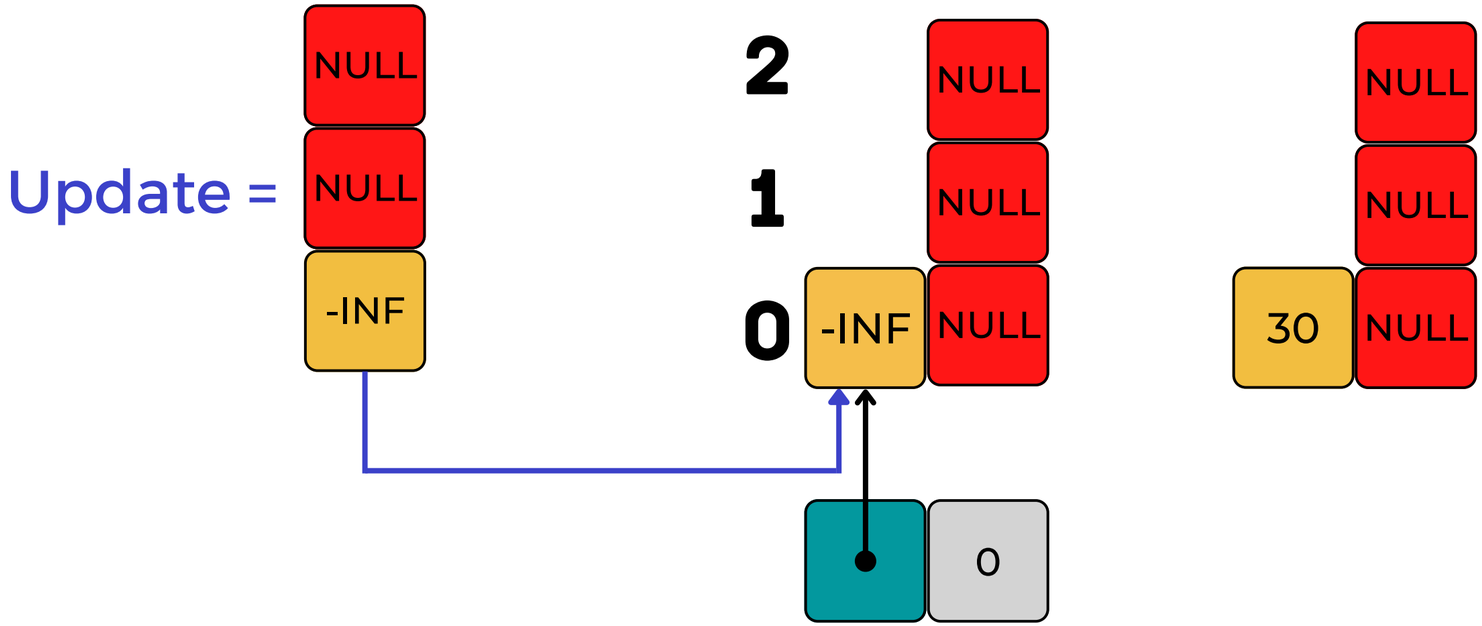
Update =



INSERT(30)

INSERTION

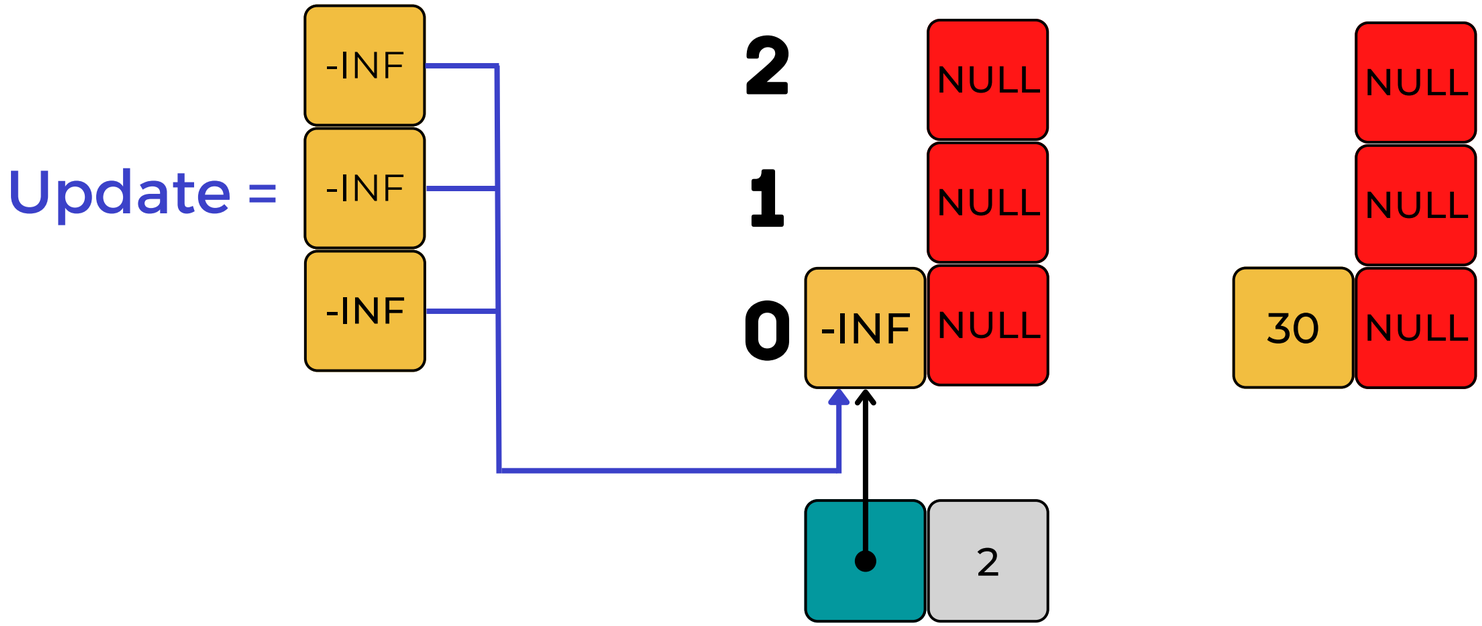
New Level = 2



INSERT(30)

INSERTION

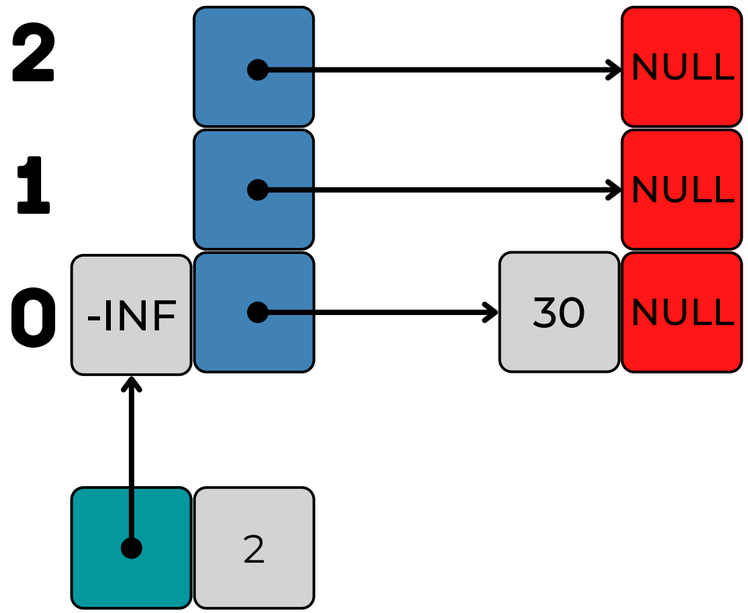
New Level = 2



INSERT(30)

INSERTION

New Level = 0

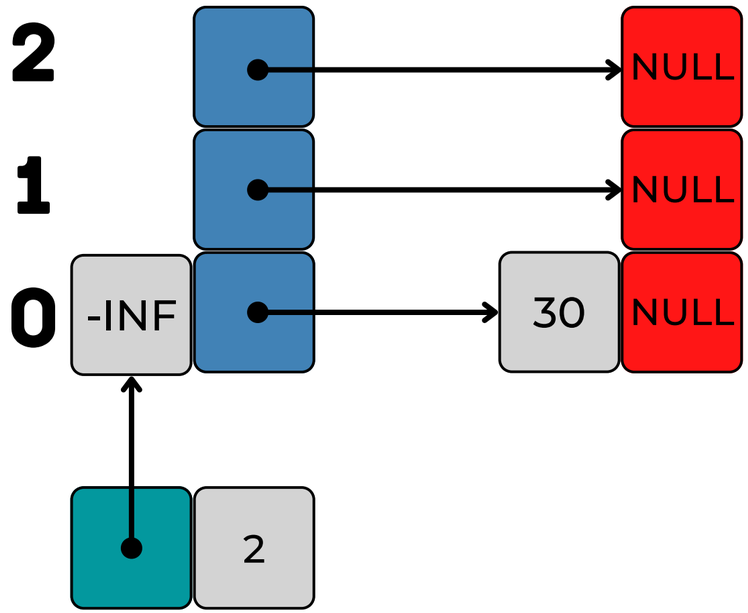


INSERT(30)

INSERTION

Max Level = 2

Level = 0



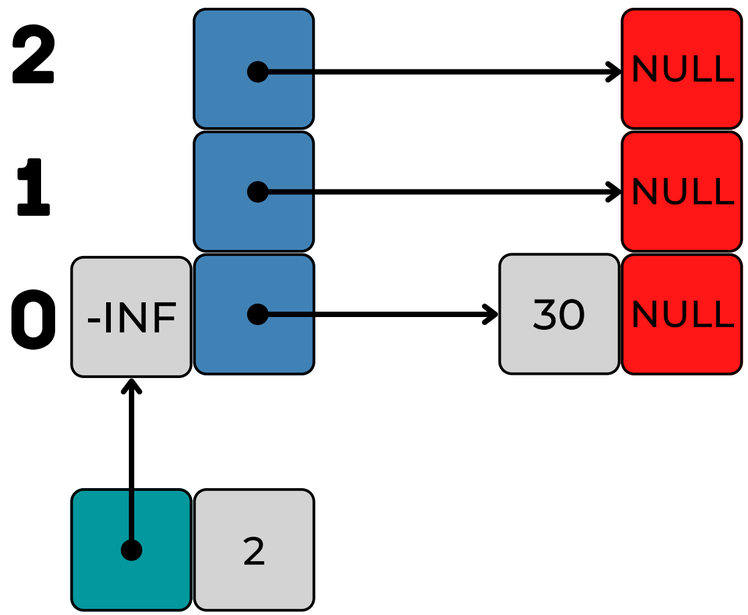
INSERT(75)

INSERTION

Max Level = 2

Level = 0

Update = NULL NULL NULL



INSERT(75)

INSERTION

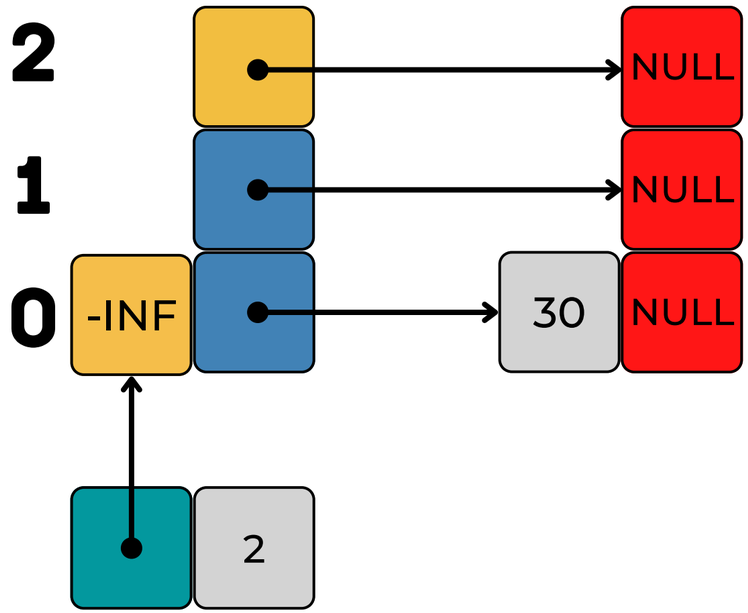
Max Level = 2

Level = 2

Update =

75 > 30 ?

True = **Move to next node**



INSERT(75)

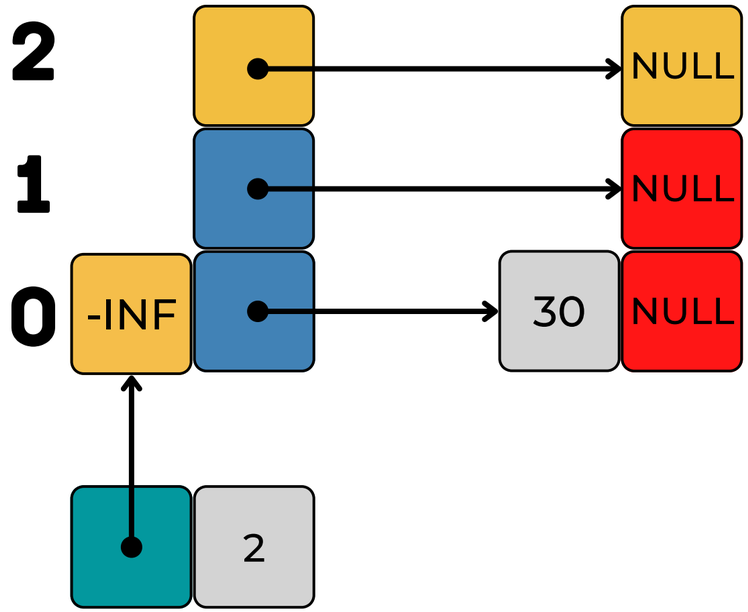
INSERTION

Max Level = 2

Level = 2

Update =

since link is NULL? **Move down a level**



INSERT(75)

TEAM JC

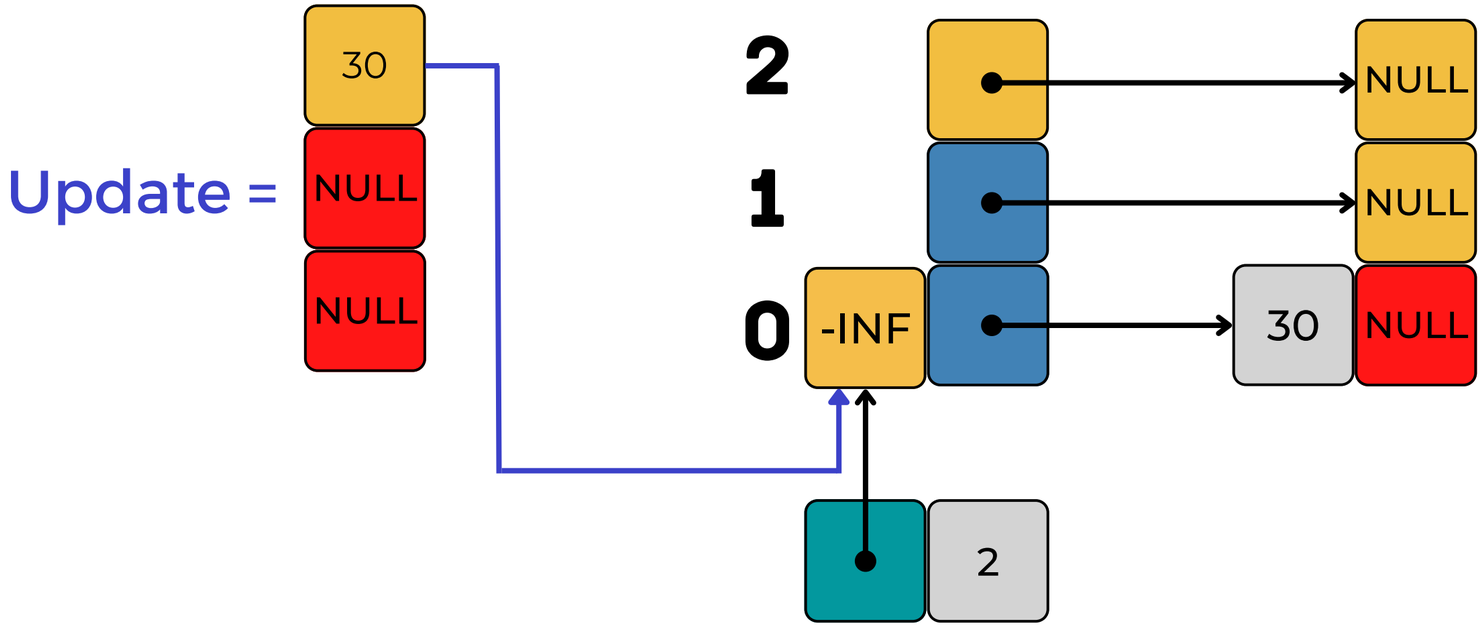
Search . . .

INSERTION

Max Level = 2

Level = 1

since link is NULL? **Move down a level**



INSERT(75)

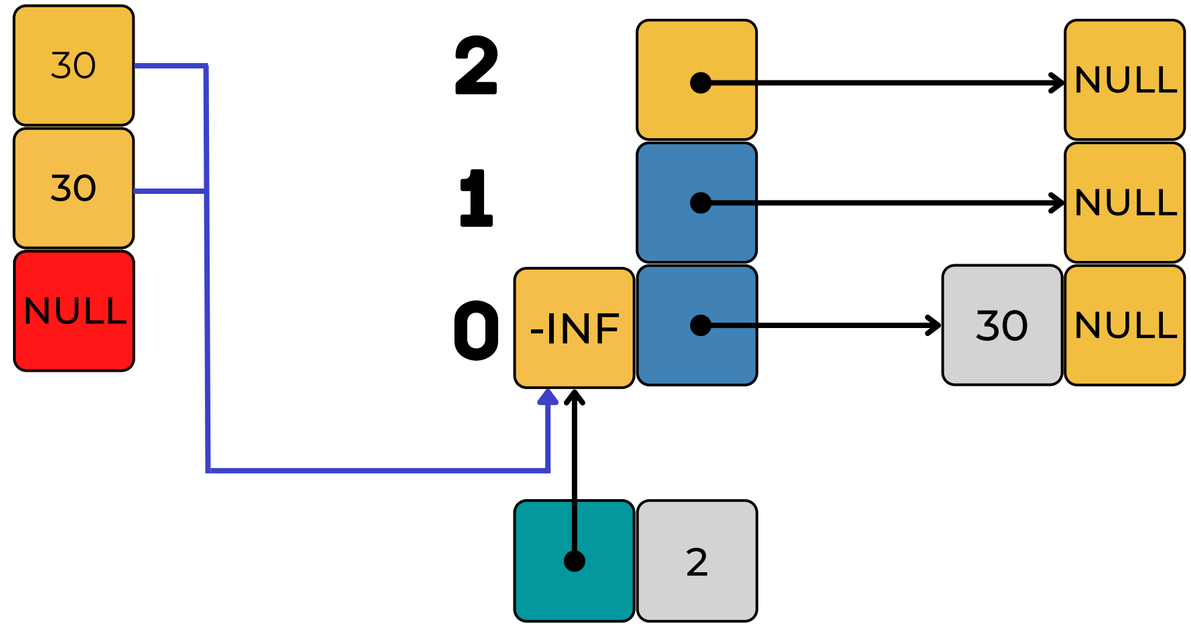
Search . . .

INSERTION

Max Level = 2

Level = 0

since link is NULL? **Move down a level**



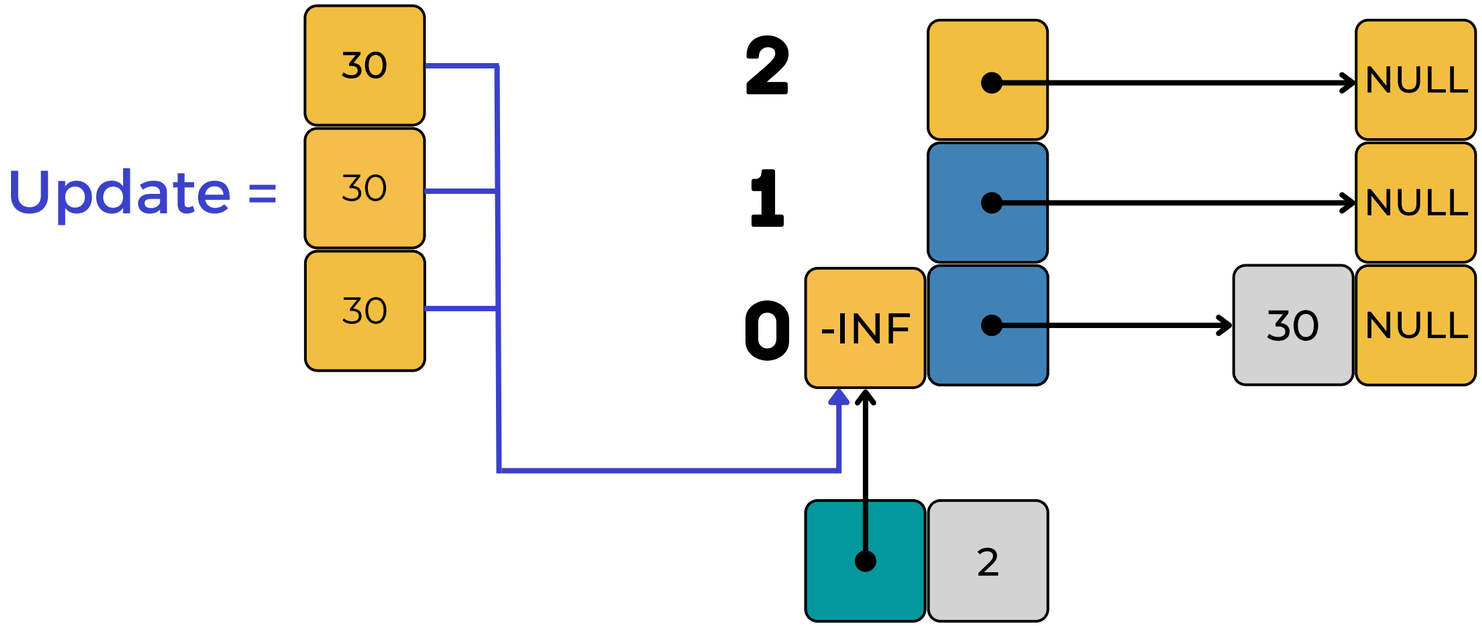
INSERT(75)

Search . . .

INSERTION

Max Level = 2

Level = -1

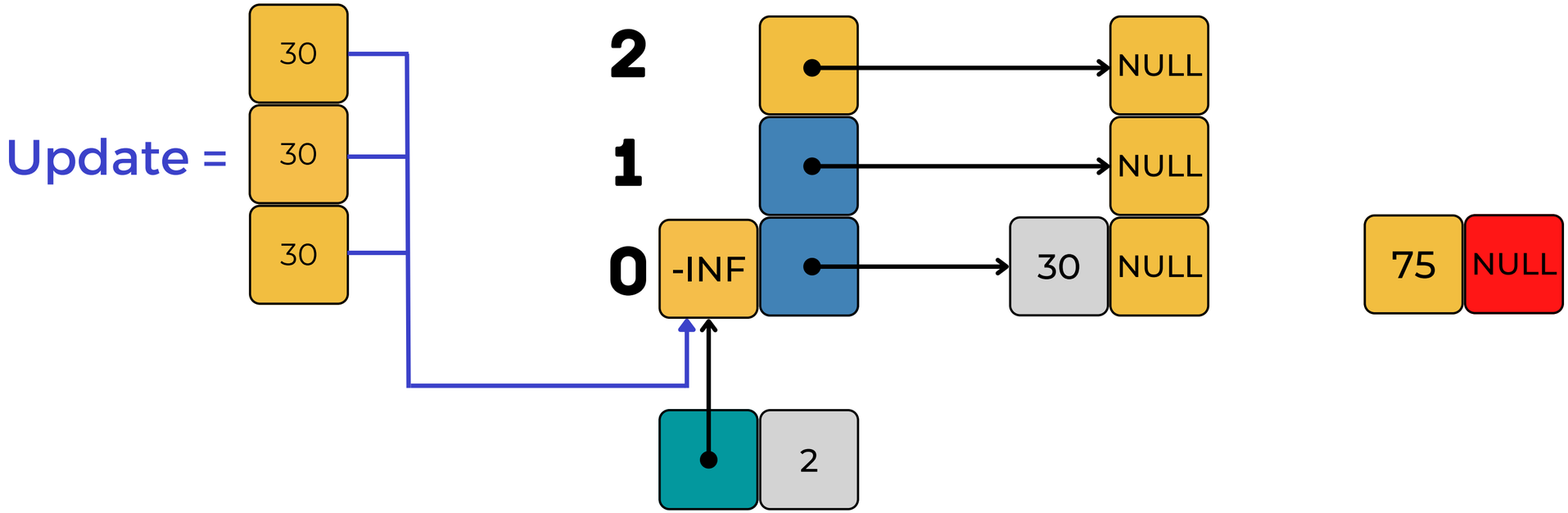


INSERT(75)

Search . . .

INSERTION

New Level = 0



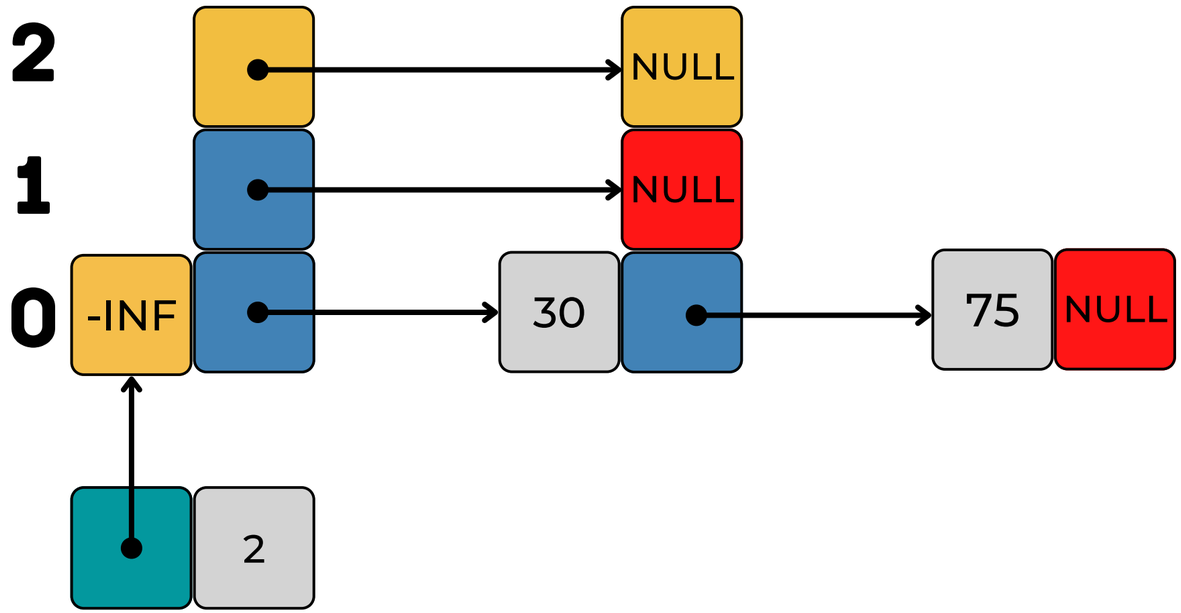
INSERT(75)

TEAM JC

Search . . .

INSERTION

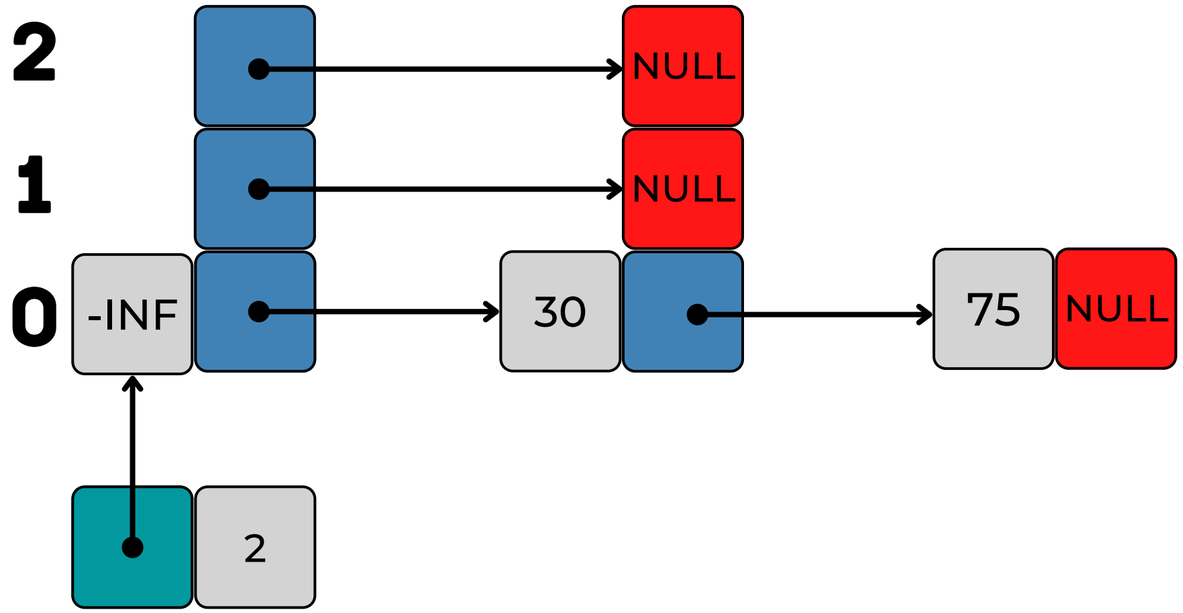
New Level = 0



INSERT(75)

Search . . .

INSERTION

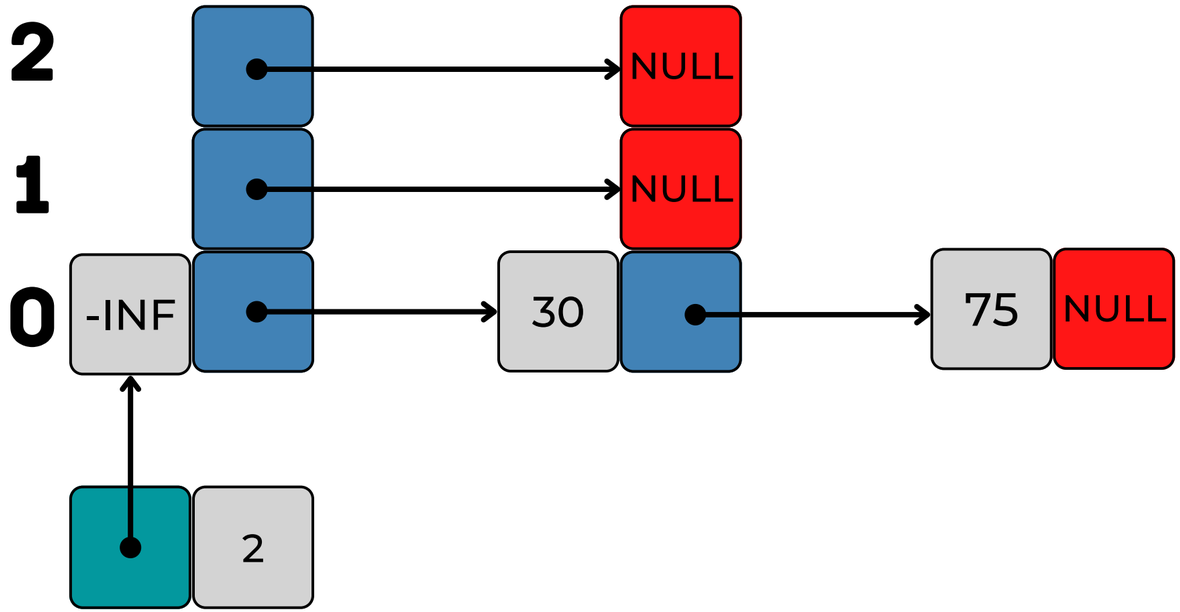


INSERT(20)

TEAM JC

Search . . .

INSERTION



Update

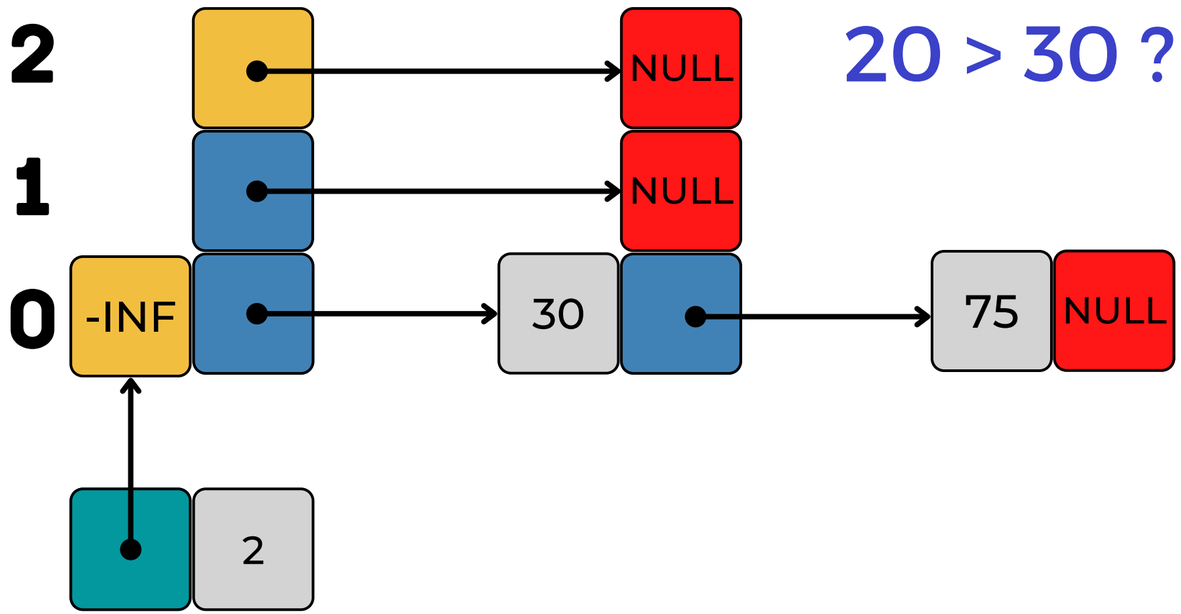
INSERT(20)

Search . . .

INSERTION

Max Level = 2

Level = 2



Update

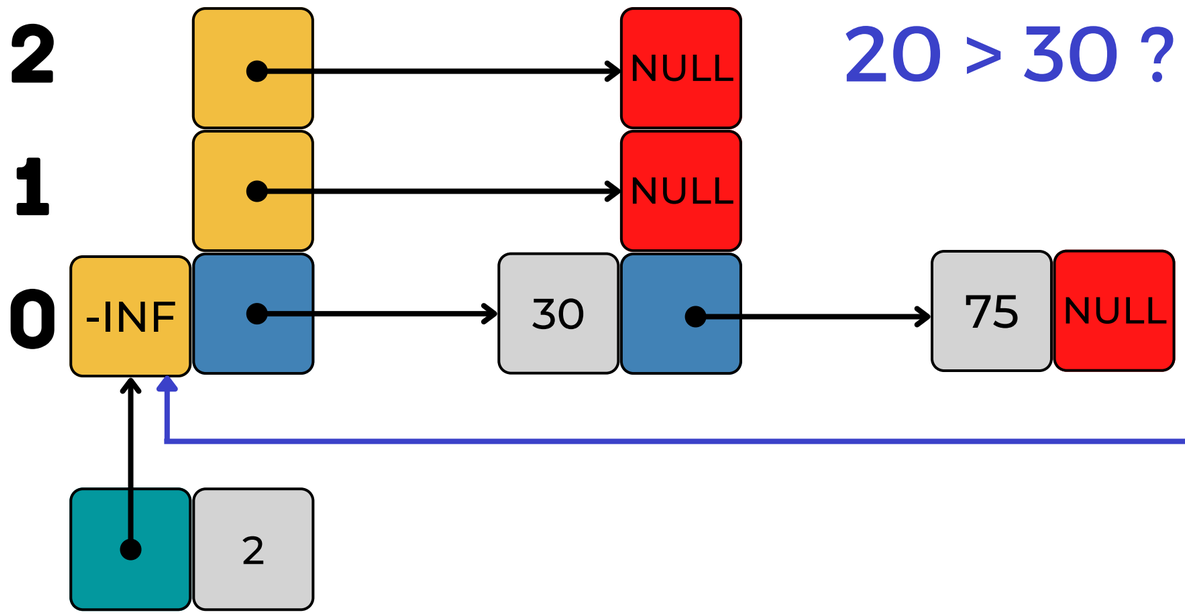
INSERT(20)

Search . . .

INSERTION

Max Level = 2

Level = 1



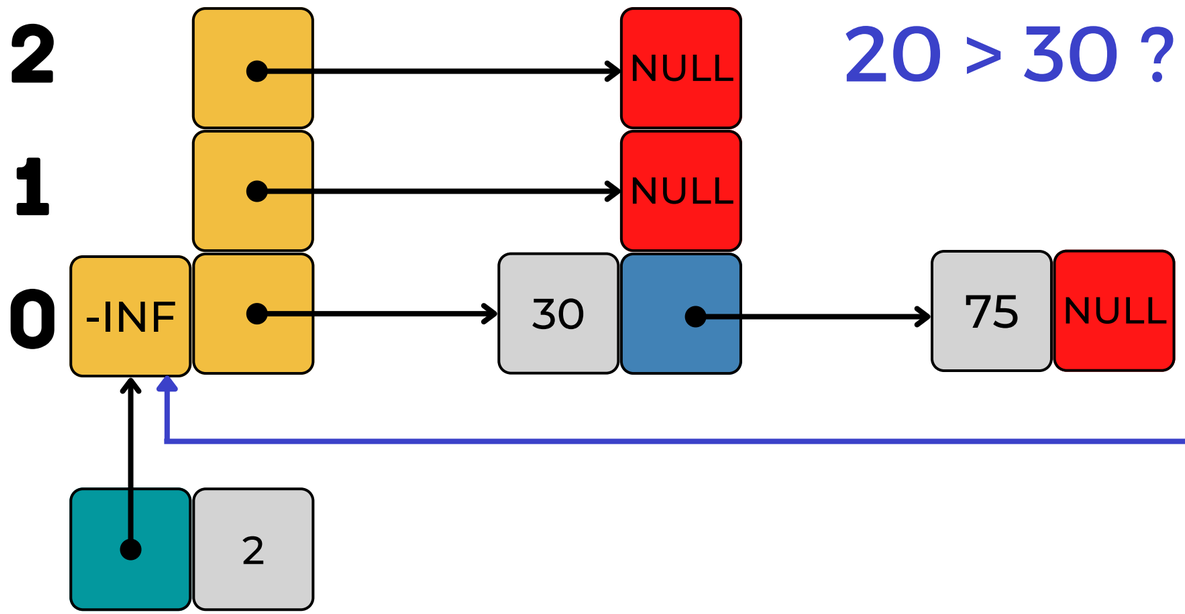
INSERT(20)

Search . . .

INSERTION

Max Level = 2

Level = 0

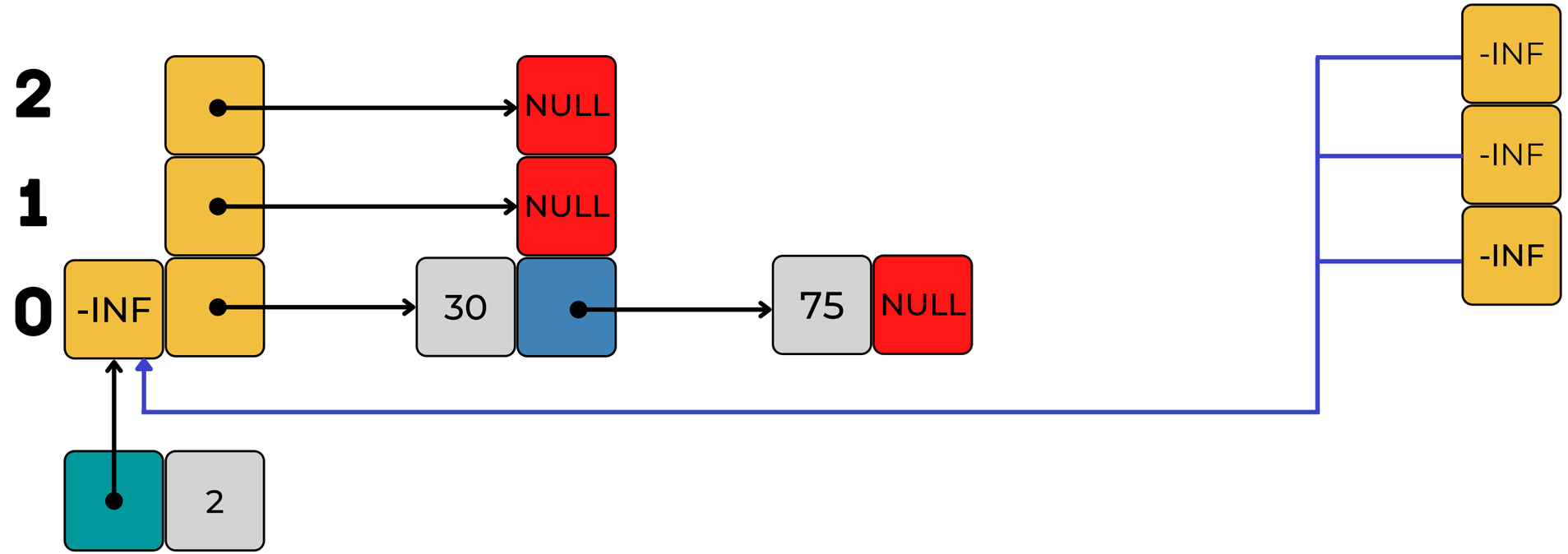


INSERT(20)

INSERTION

Max Level = 2

Level = -1



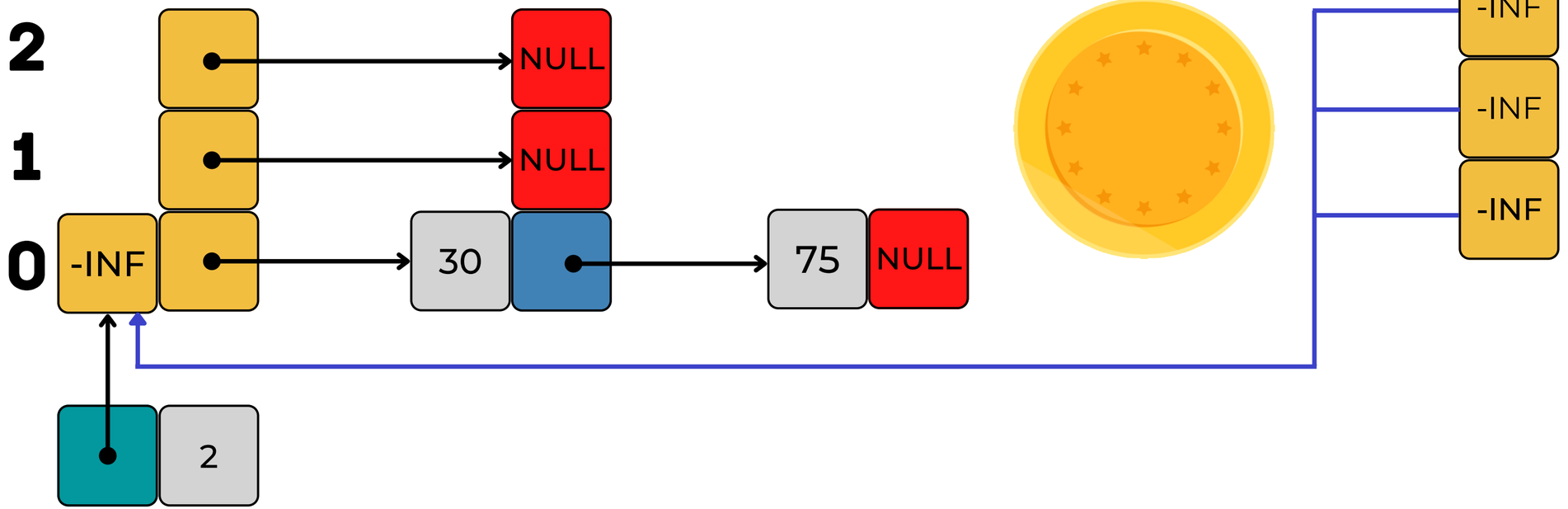
INSERT(20)

Search . . .

INSERTION

Max Level = 2

Level = -1

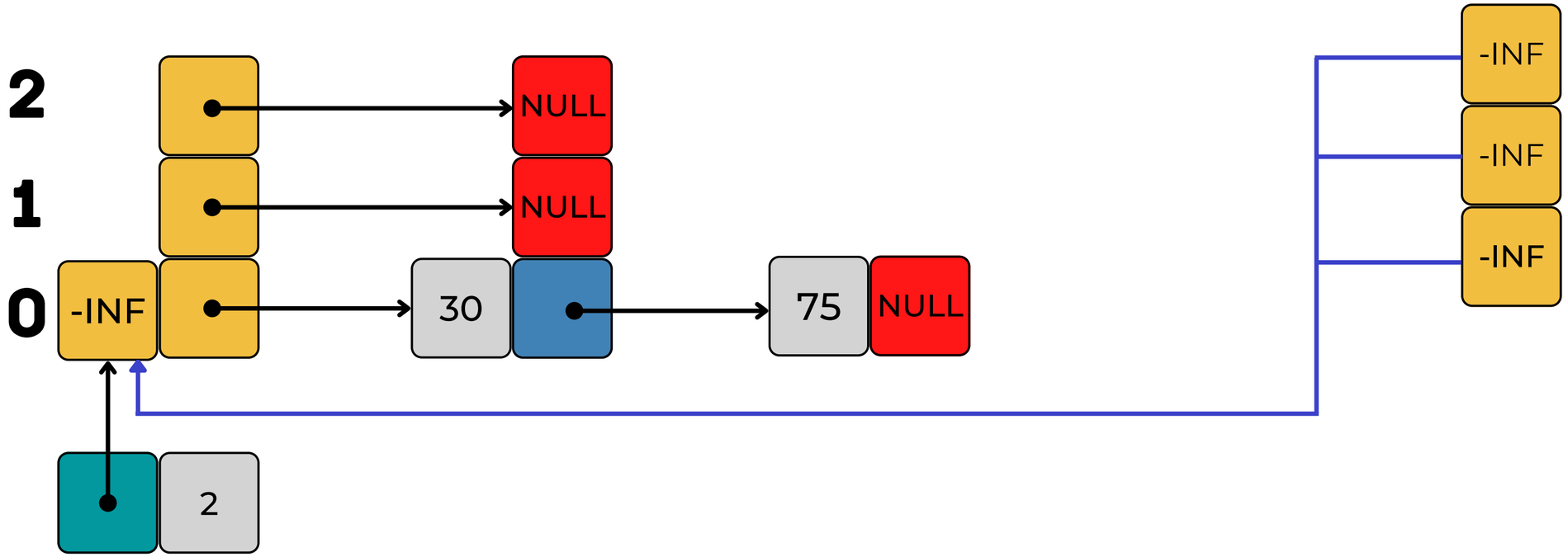


INSERT(20)

INSERTION

New Level = 1

Update



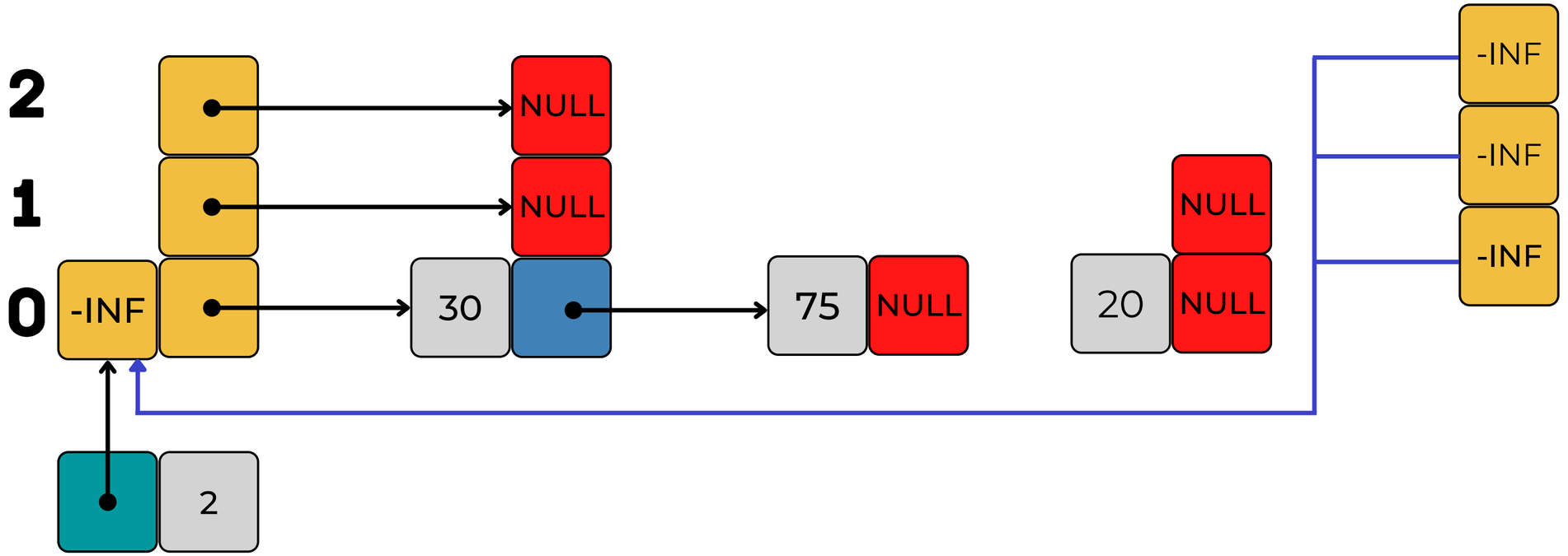
INSERT(20)

Search . . .

INSERTION

New Level = 1

Update



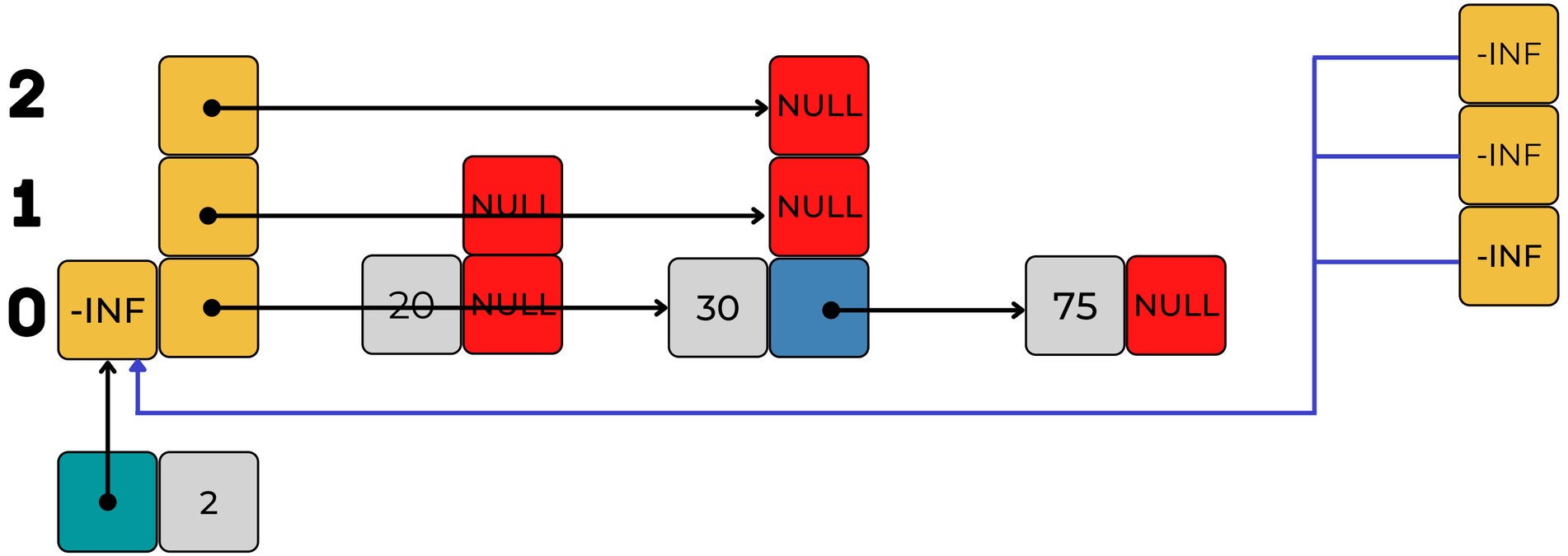
INSERT(20)

Search . . .

INSERTION

New Level = 1

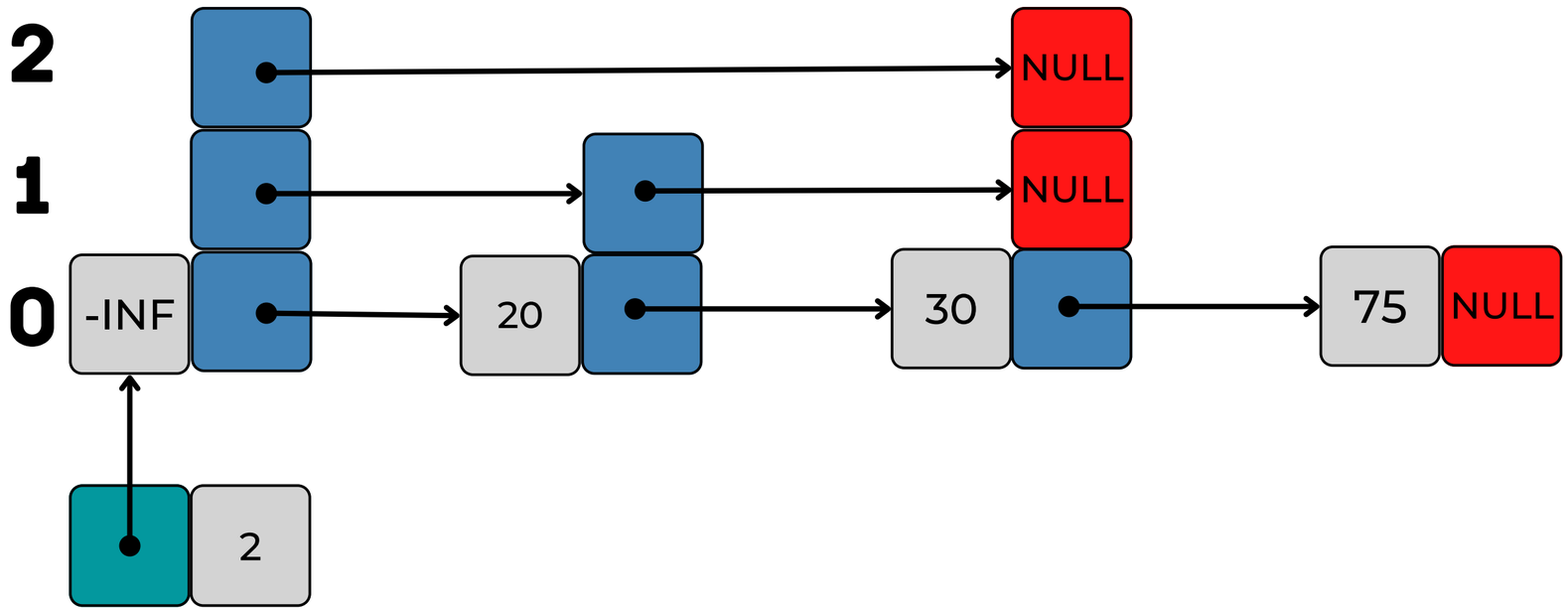
Update



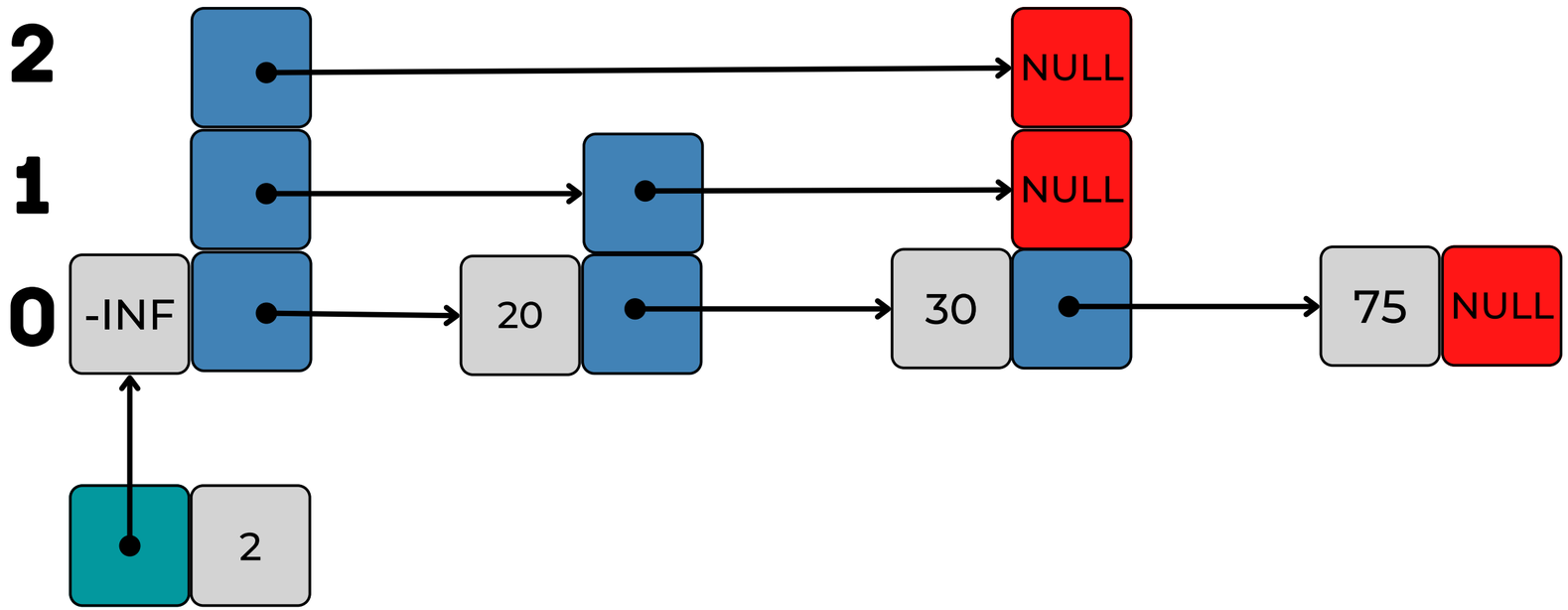
INSERT(20)

Search . . .

INSERTION



INSERTION

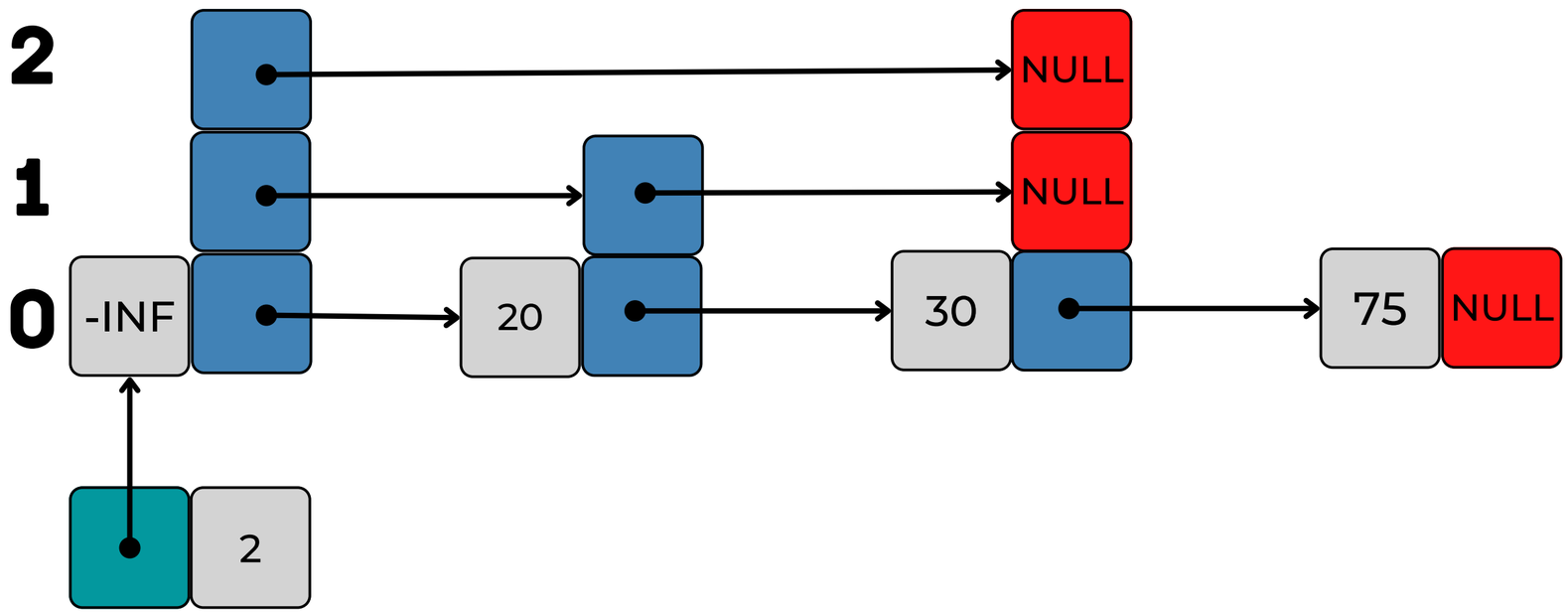


INSERT(50)

TEAM JC

Search . . .

INSERTION



Update =

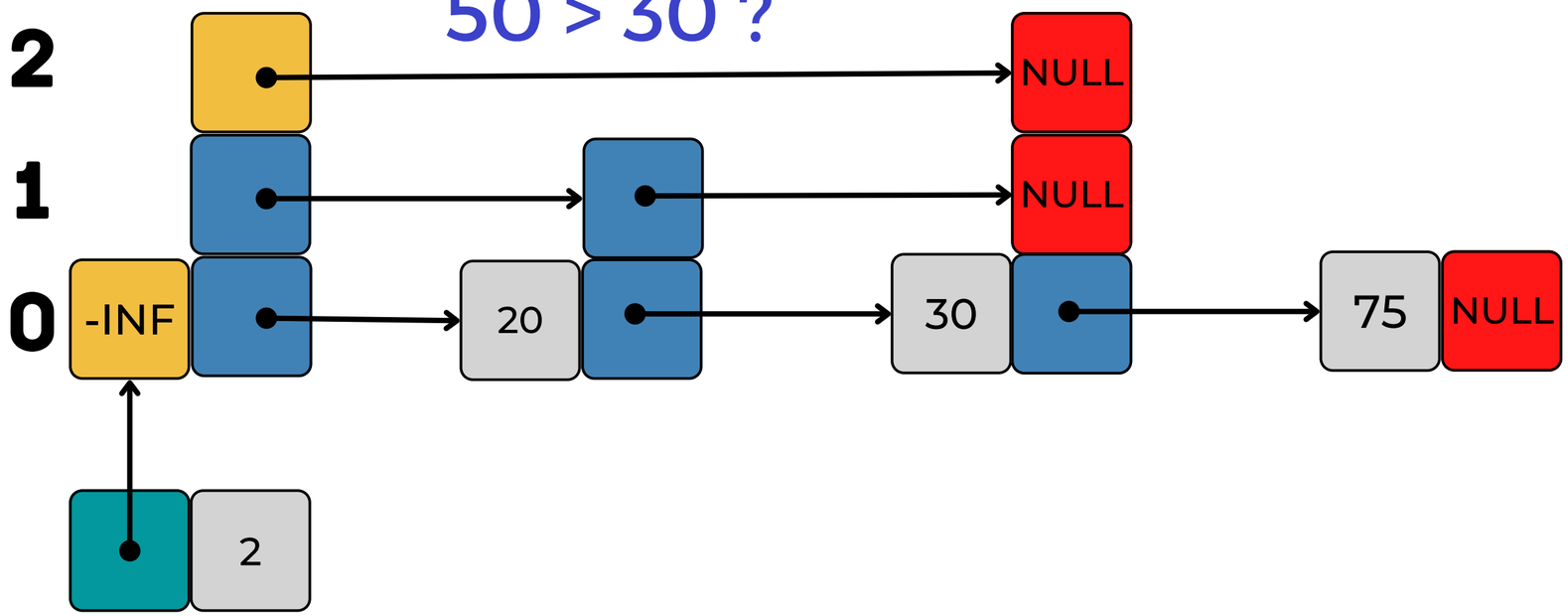
INSERT(50)

INSERTION

Max Level = 2

Level = 2

True = **Move to next node**



Update =

INSERT(50)

TEAM JC

Search . . .

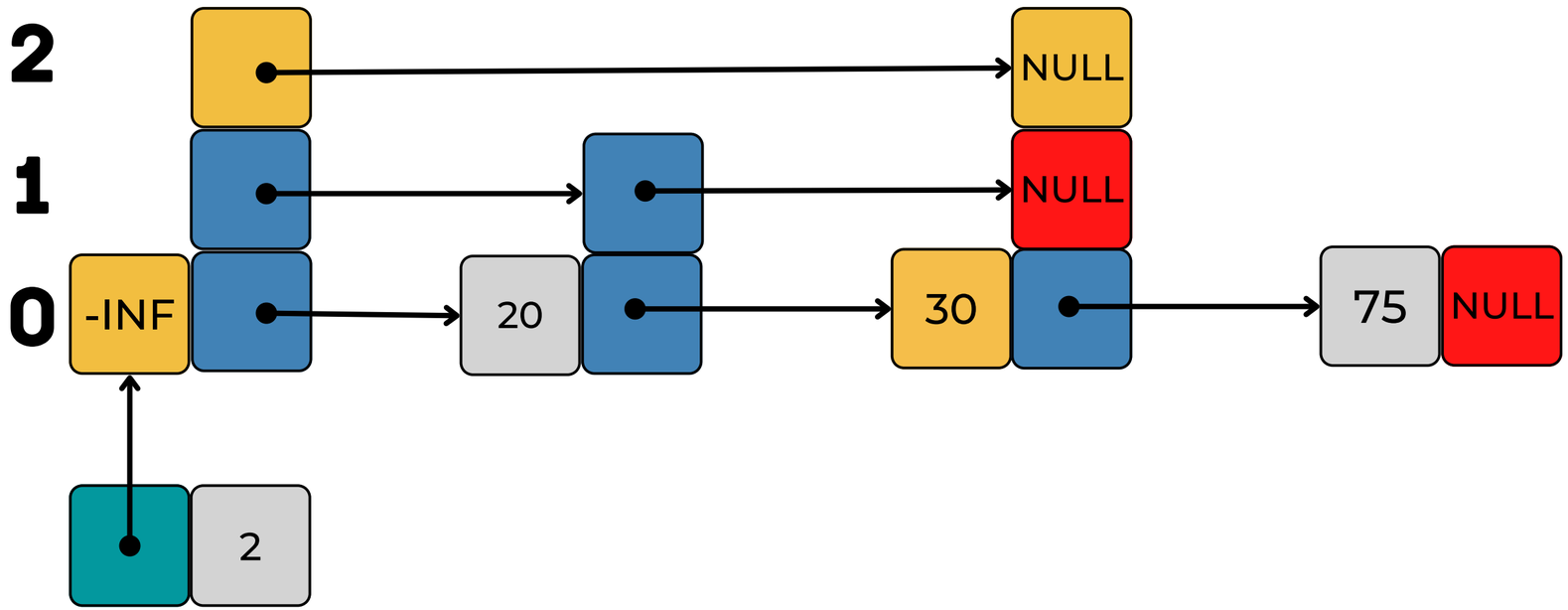
INSERTION

Max Level = 2

Level = 2

since link is NULL? **Move down a level**

Update



INSERT(50)

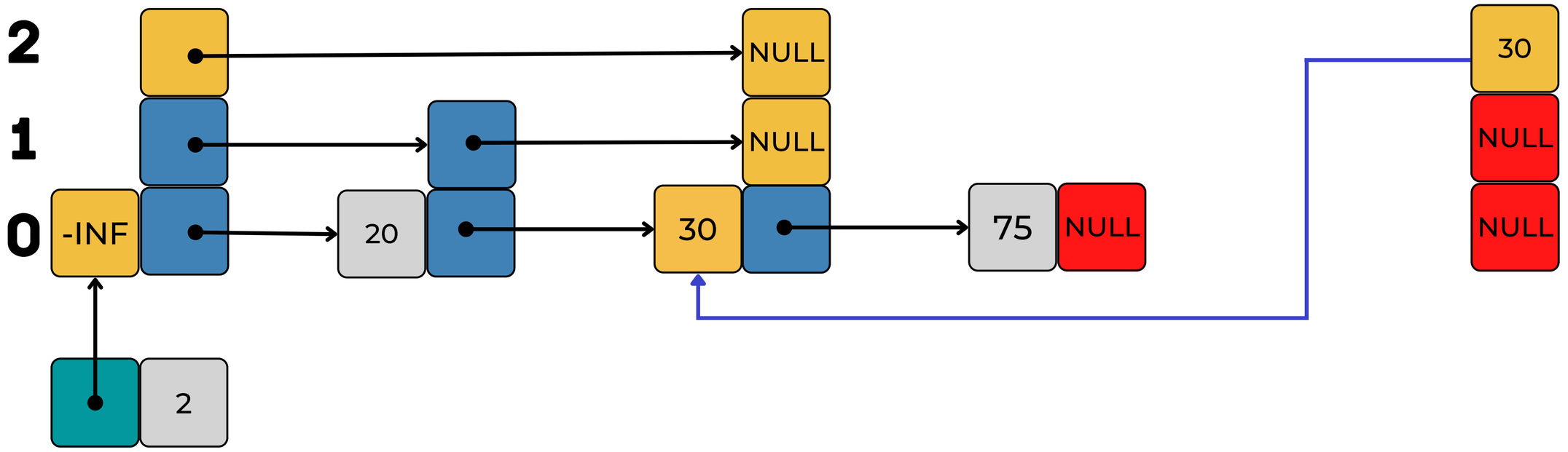
INSERTION

Max Level = 2

Level = 1

since link is NULL? **Move down a level**

Update

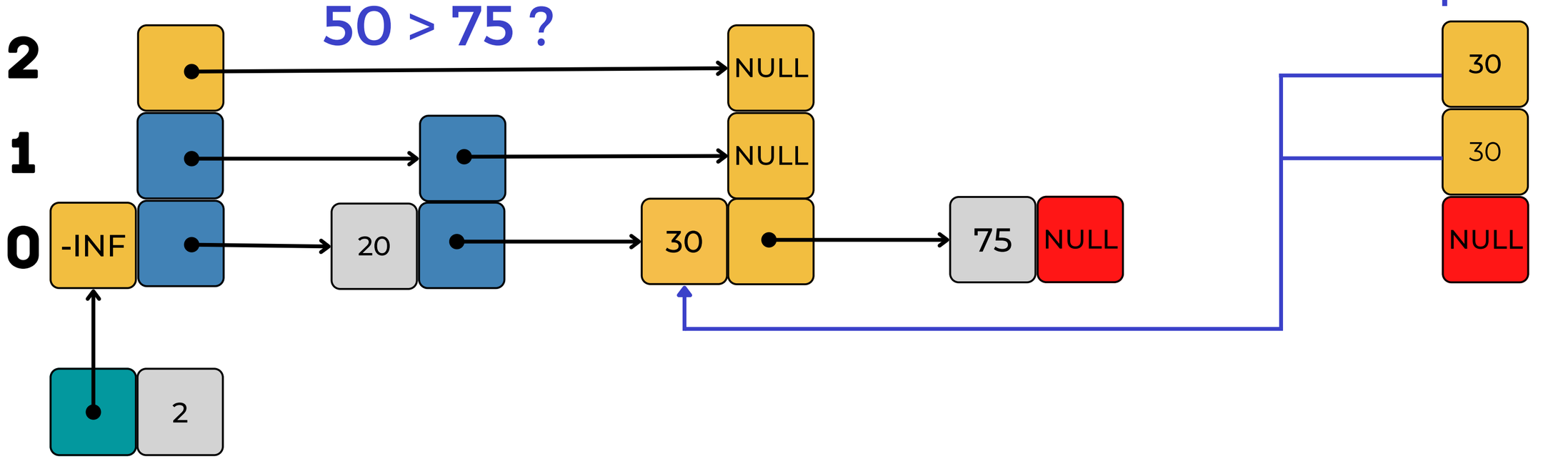


INSERT(50)

INSERTION

Max Level = 2

Level = 0

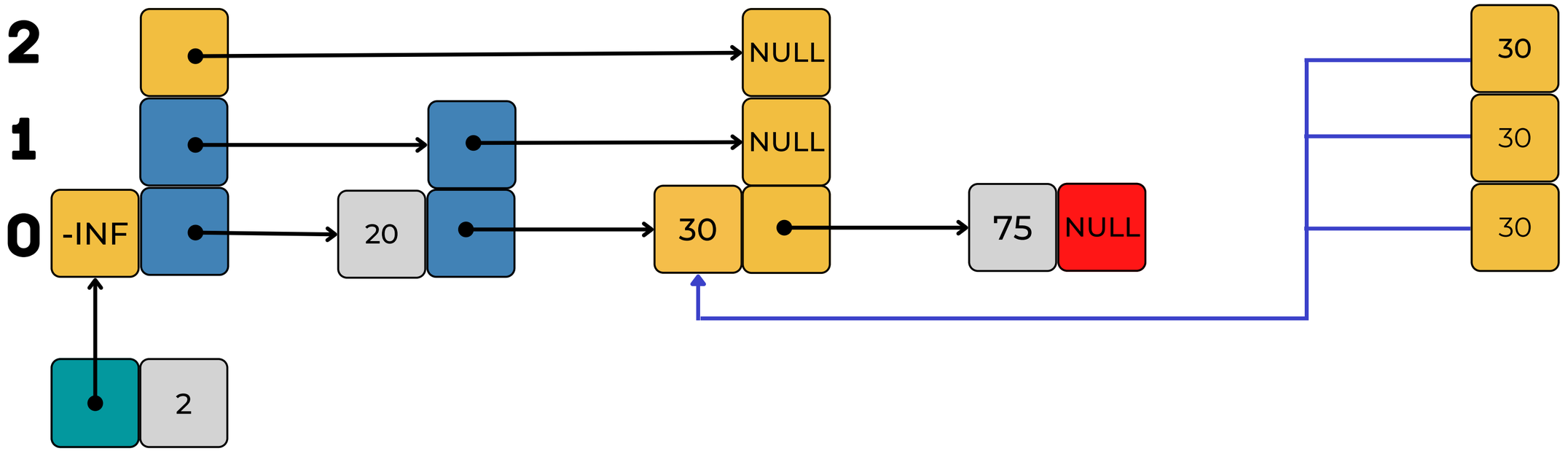


INSERT(50)

INSERTION

Max Level = 2

Level = -1

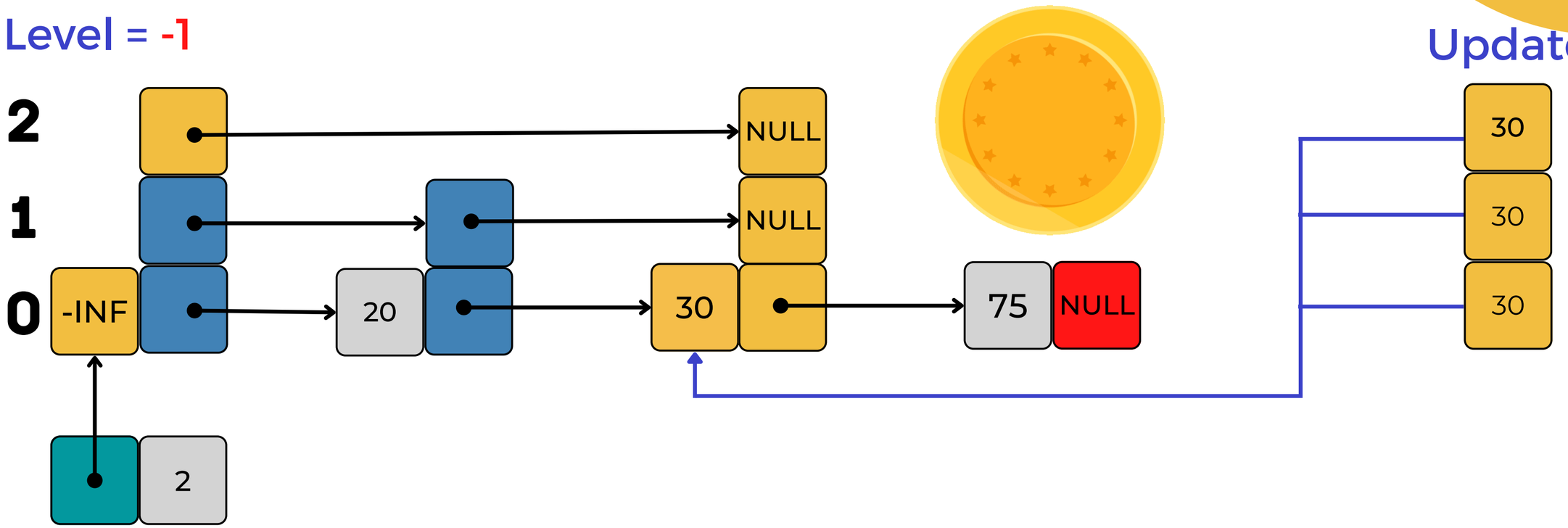


INSERT(50)

INSERTION

Max Level = 2

Level = -1



INSERT(50)

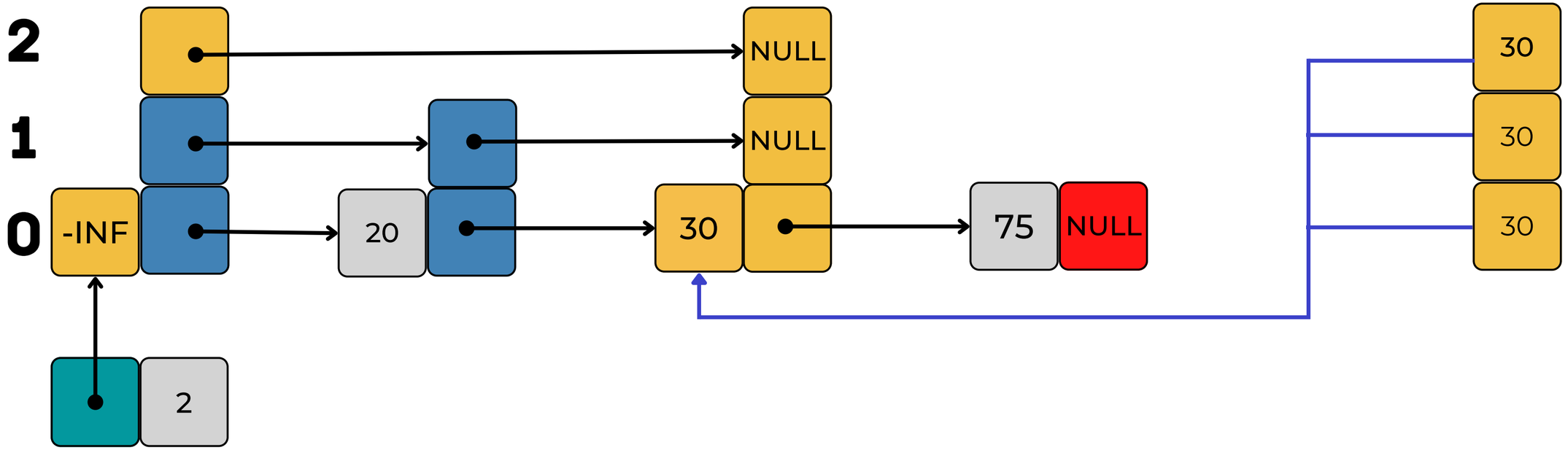
TEAM JC

Search . . .

INSERTION

New Level = 1

Update



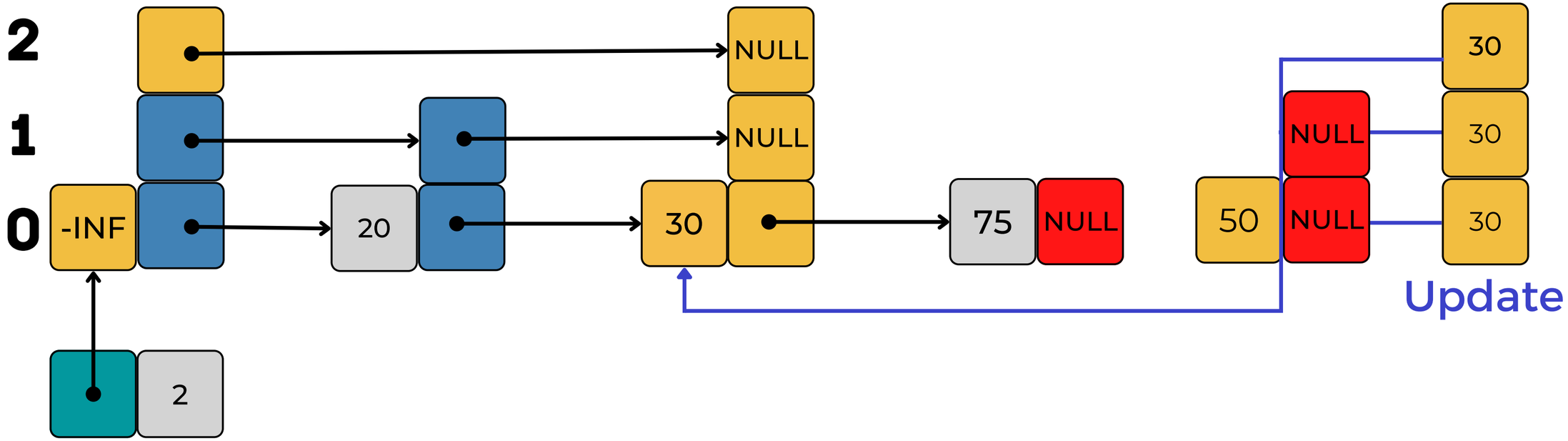
INSERT(50)

TEAM JC

Search . . .

INSERTION

New Level = 1



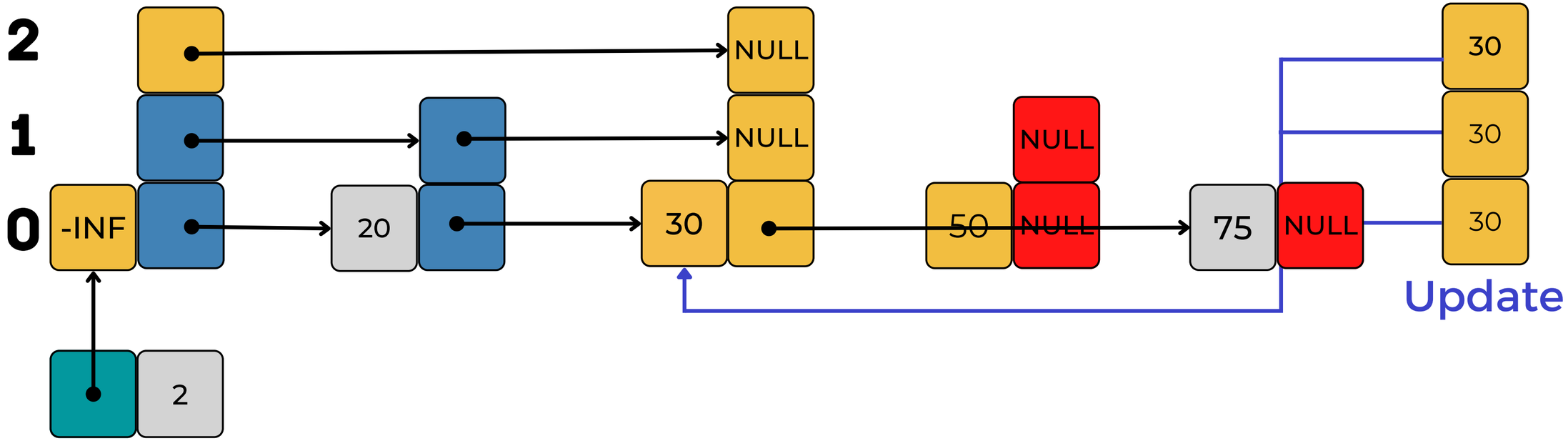
INSERT(50)

TEAM JC

Search . . .

INSERTION

New Level = 1



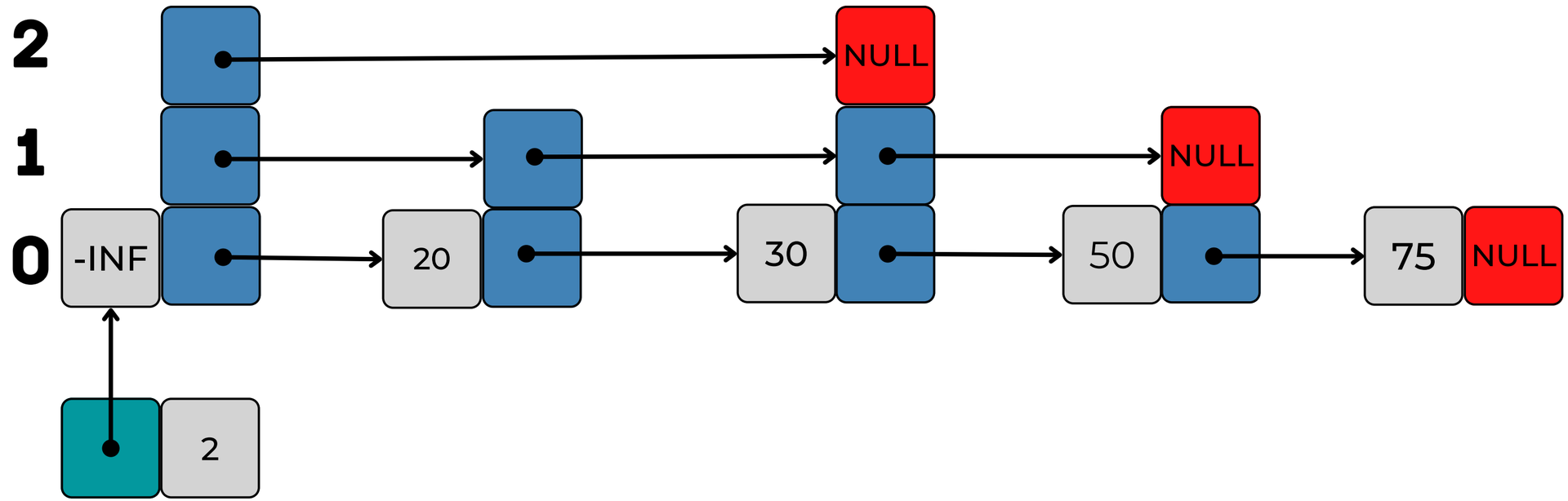
INSERT(50)

TEAM JC

Search . . .

SEARCH

Level = 2



SEARCH(75)

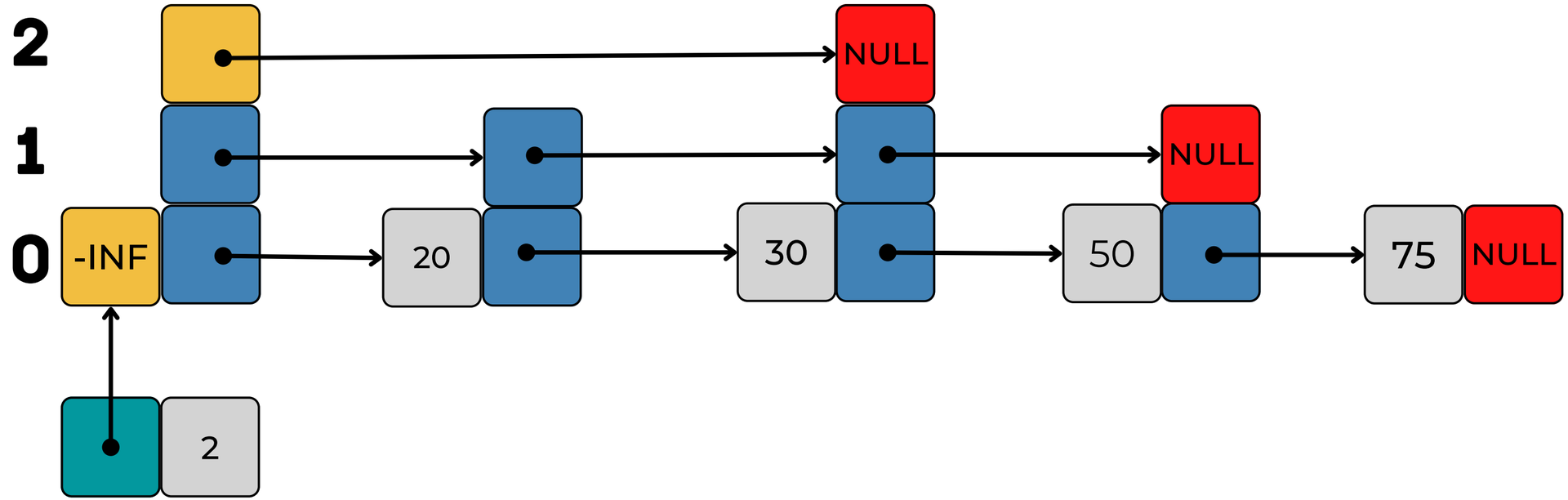
TEAM JC

Search . . .

SEARCH

Level = 2

75>30? **True** = Move to next node



SEARCH(75)

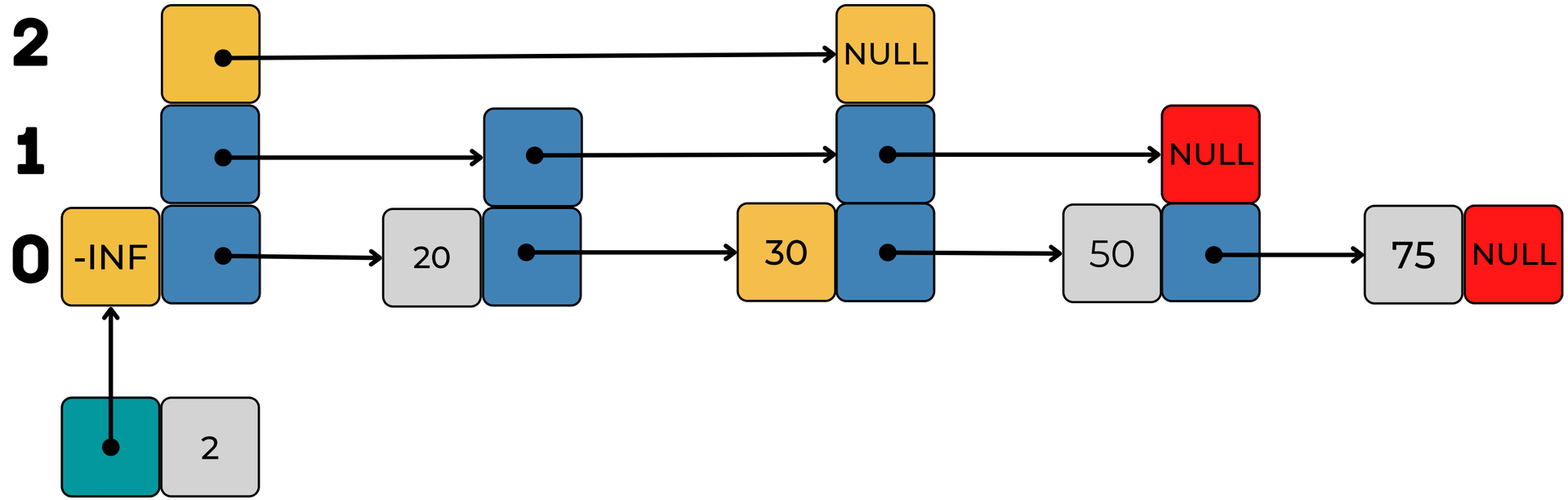
TEAM JC

Search . . .

SEARCH

Level = 2

since link is NULL? **Move down a level**



SEARCH(75)

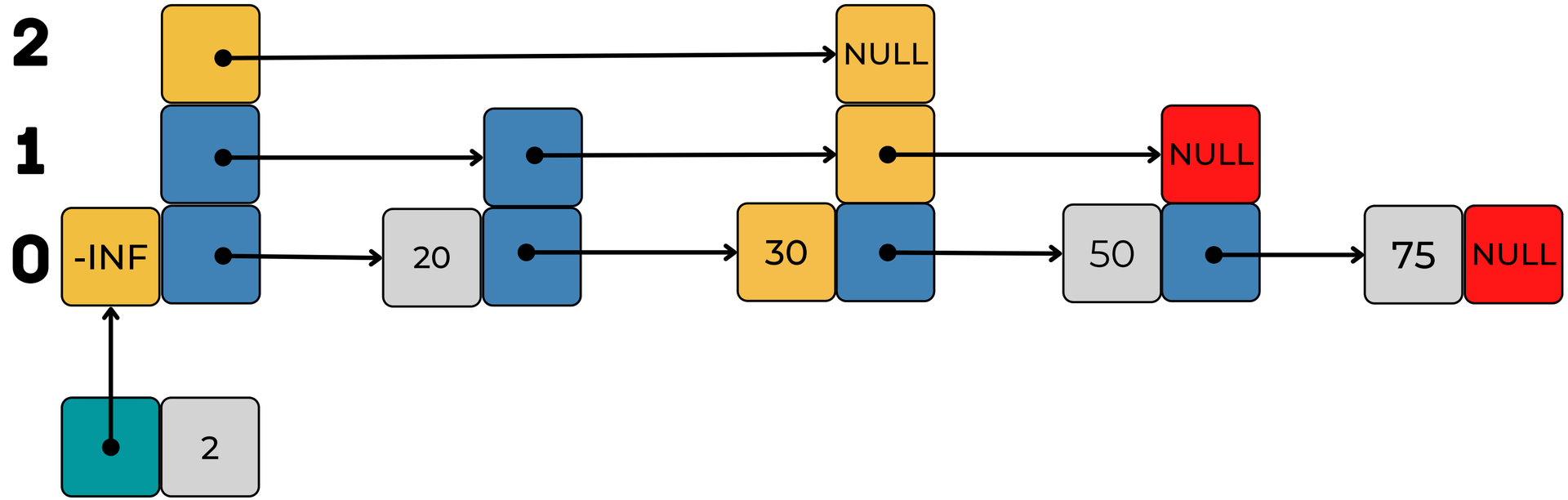
TEAM JC

Search . . .

SEARCH

Level = 1

75>50? **True** = Move to next node



SEARCH(75)

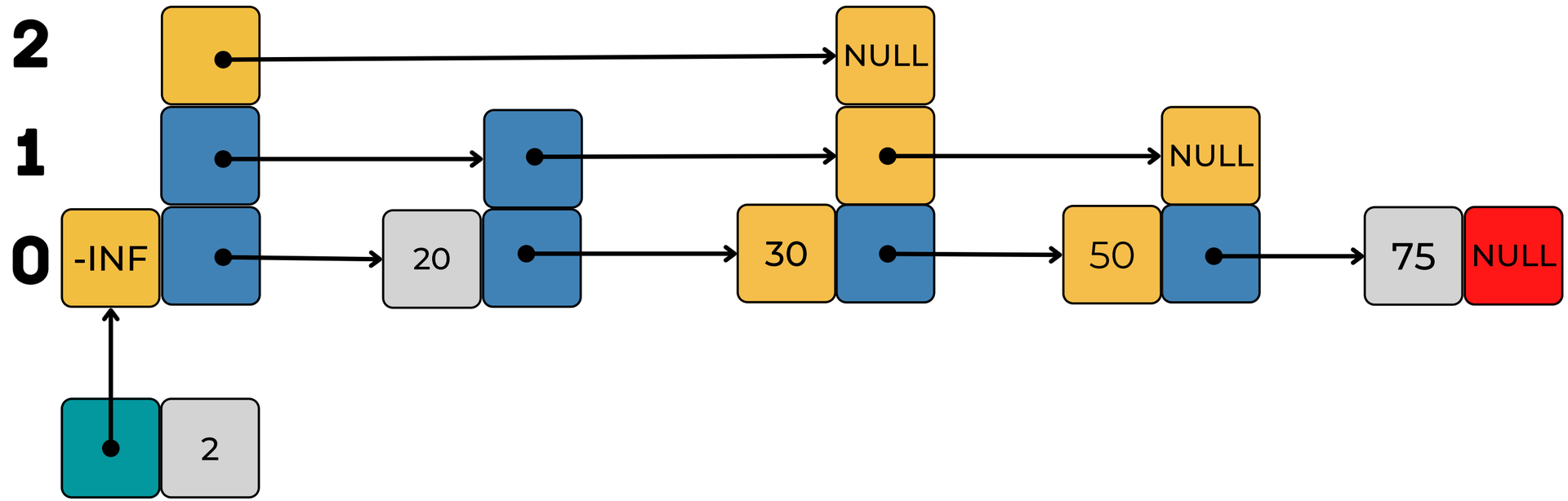
TEAM JC

Search . . .

SEARCH

Level = 1

since link is NULL? **Move down a level**



SEARCH(75)

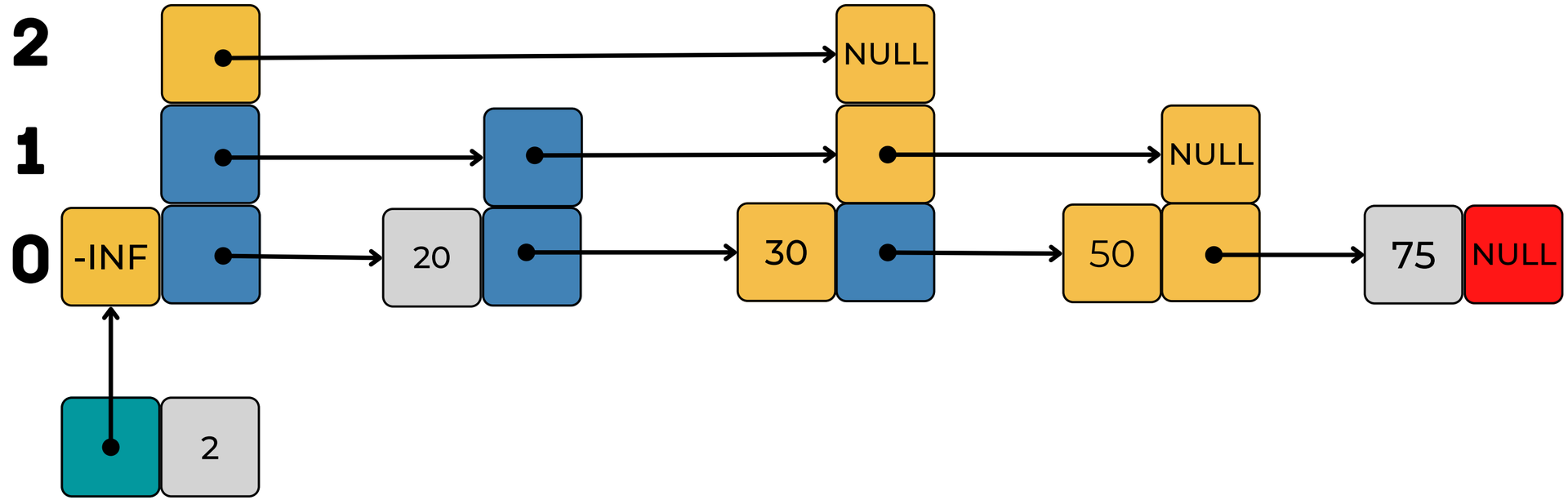
TEAM JC

Search . . .

SEARCH

Level = 0

75>75? **False** = **Move down a level**



SEARCH(75)

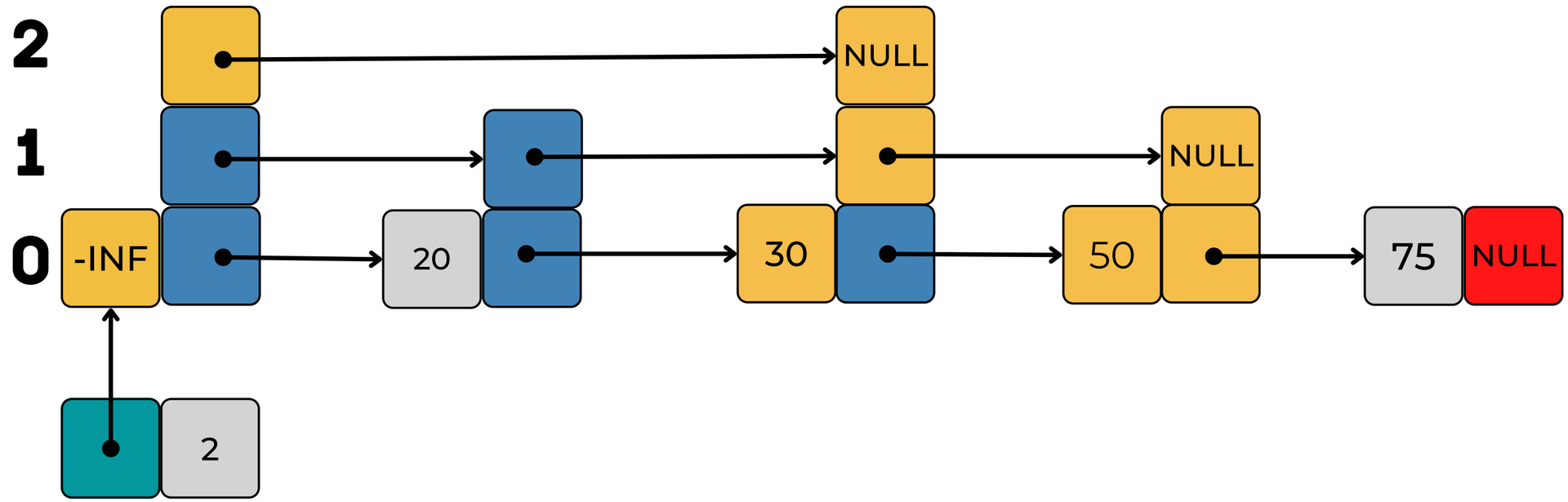
TEAM JC

Search . . .

SEARCH

Level = -1

since level < 0 then return the next node



SEARCH(75)

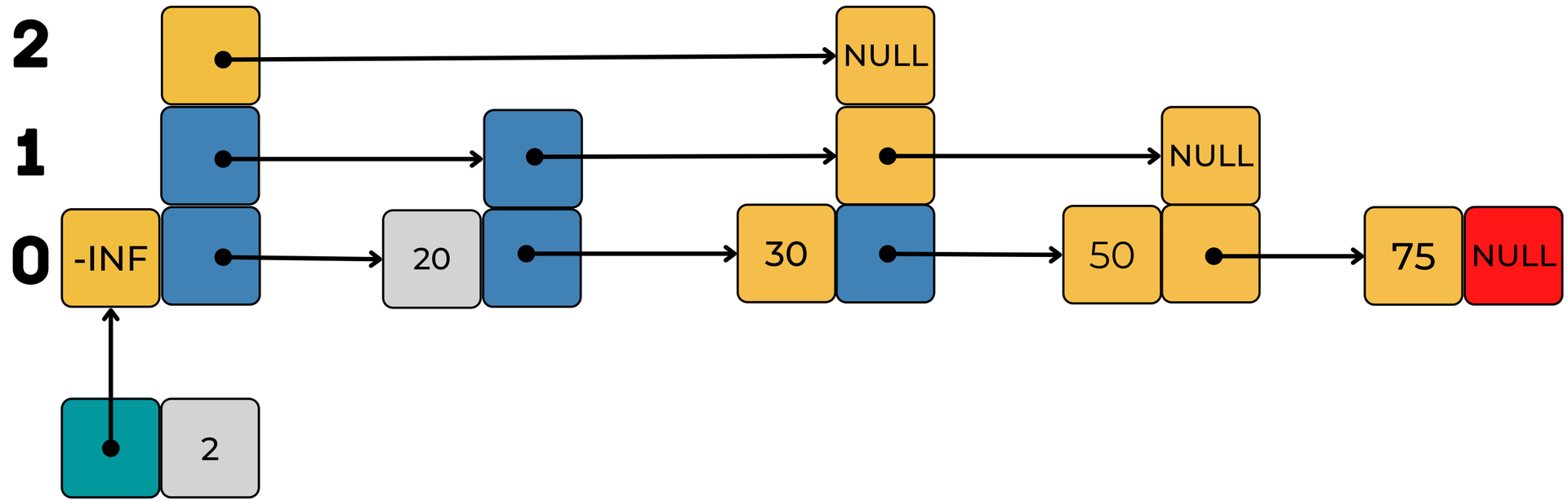
TEAM JC

Search . . .

SEARCH

Level = -1

75 == 75? **True then 75 is an element from the skiplist**

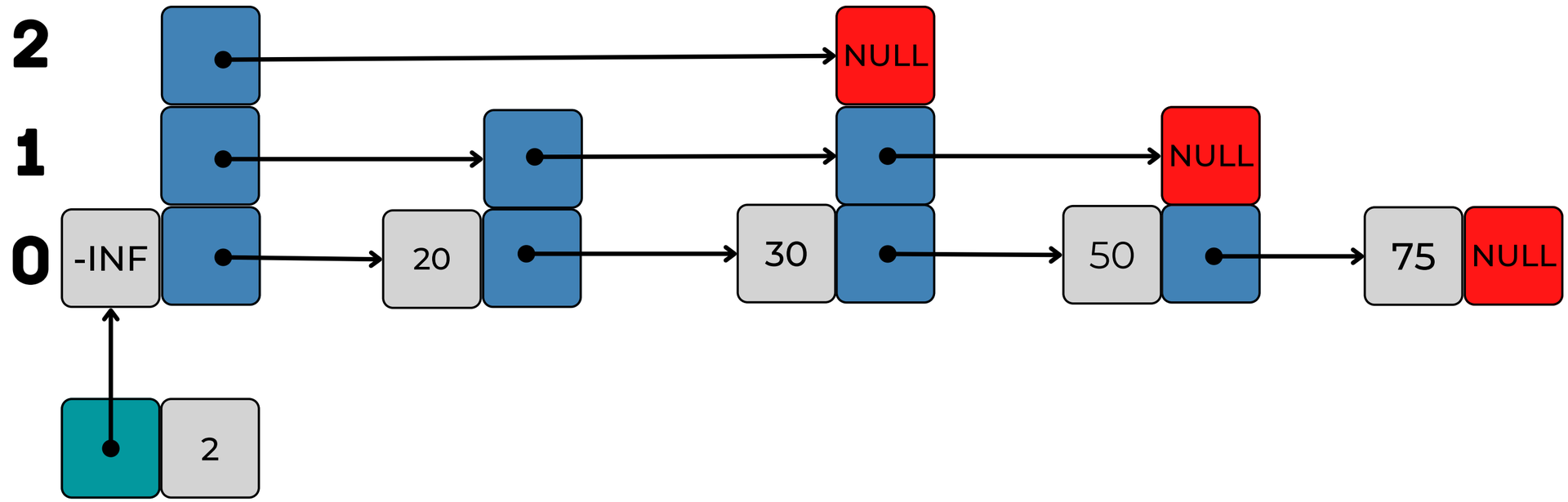


SEARCH(75)

Search . . .

SEARCH

Level = 2



SEARCH(40)

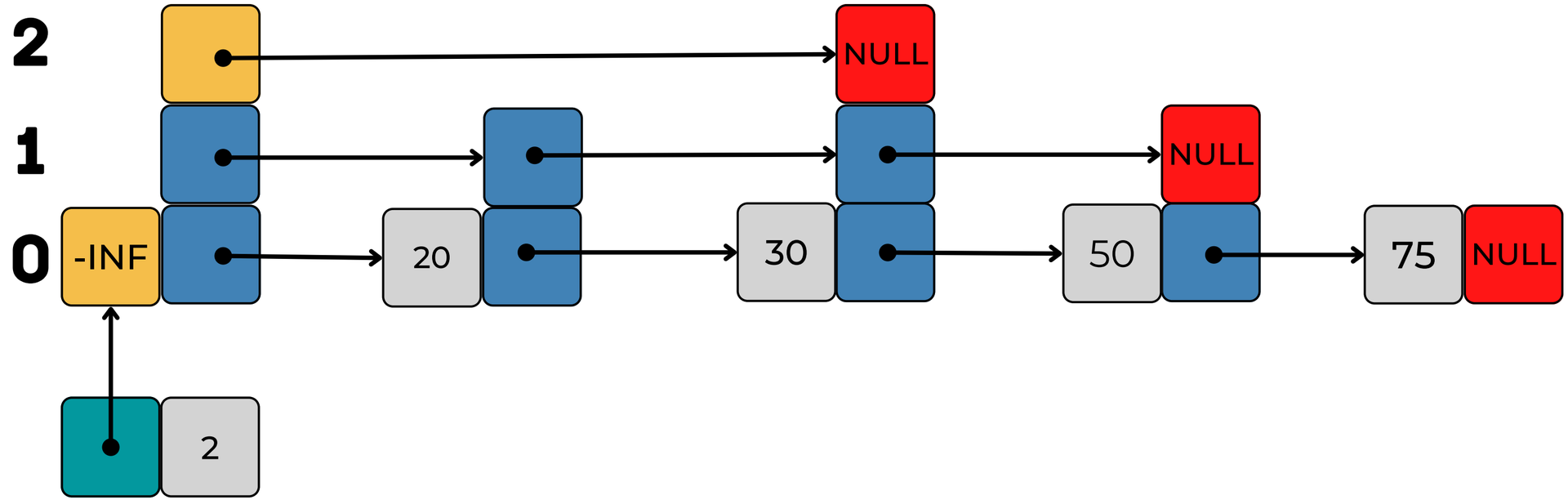
TEAM JC

Search . . .

SEARCH

Level = 2

40>30? **True** = Move to next node



SEARCH(40)

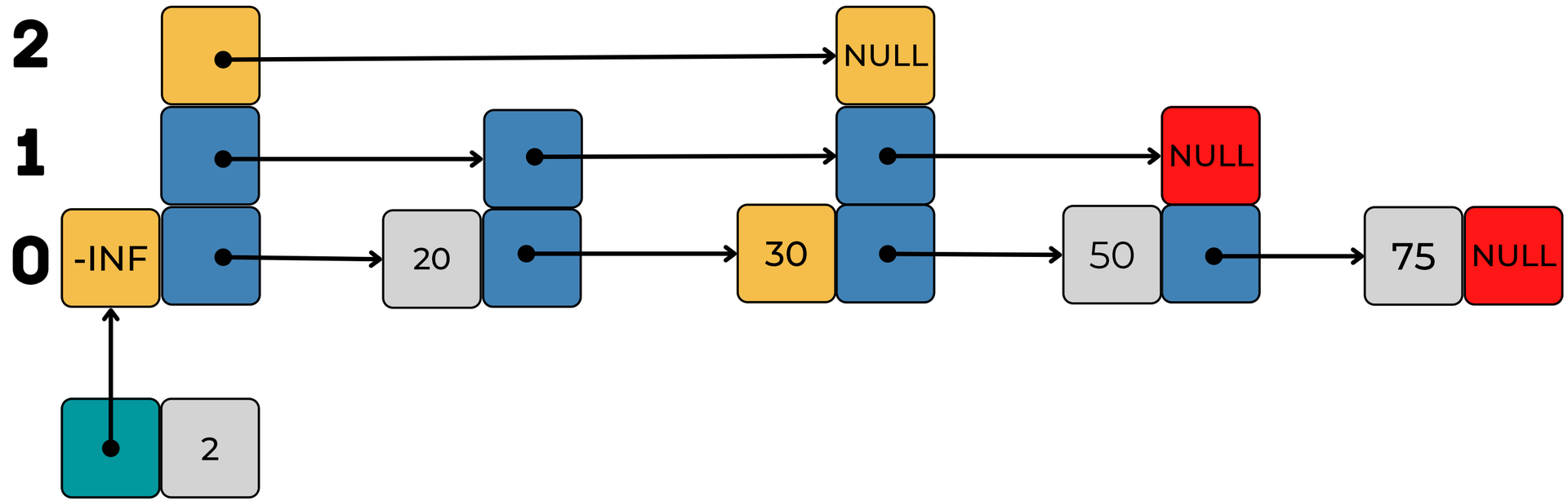
TEAM JC

Search . . .

SEARCH

Level = 2

since link is NULL? **Move down a level**



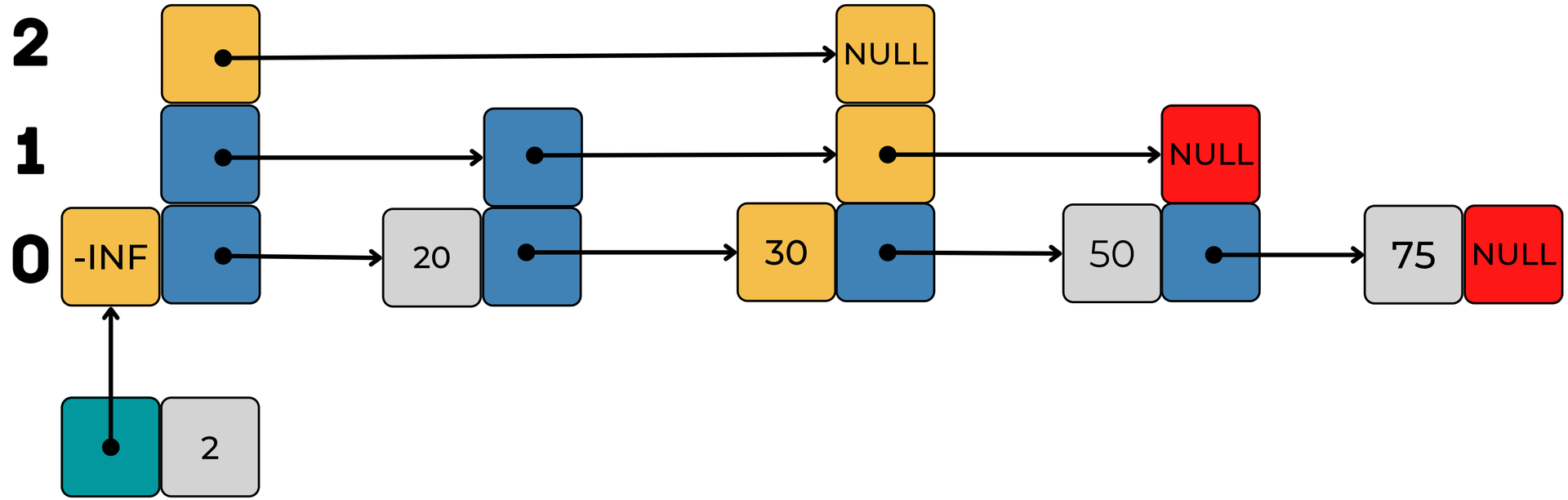
SEARCH(40)

Search . . .

SEARCH

Level = 1

40>50? **False** = **Move down a level**



SEARCH(40)

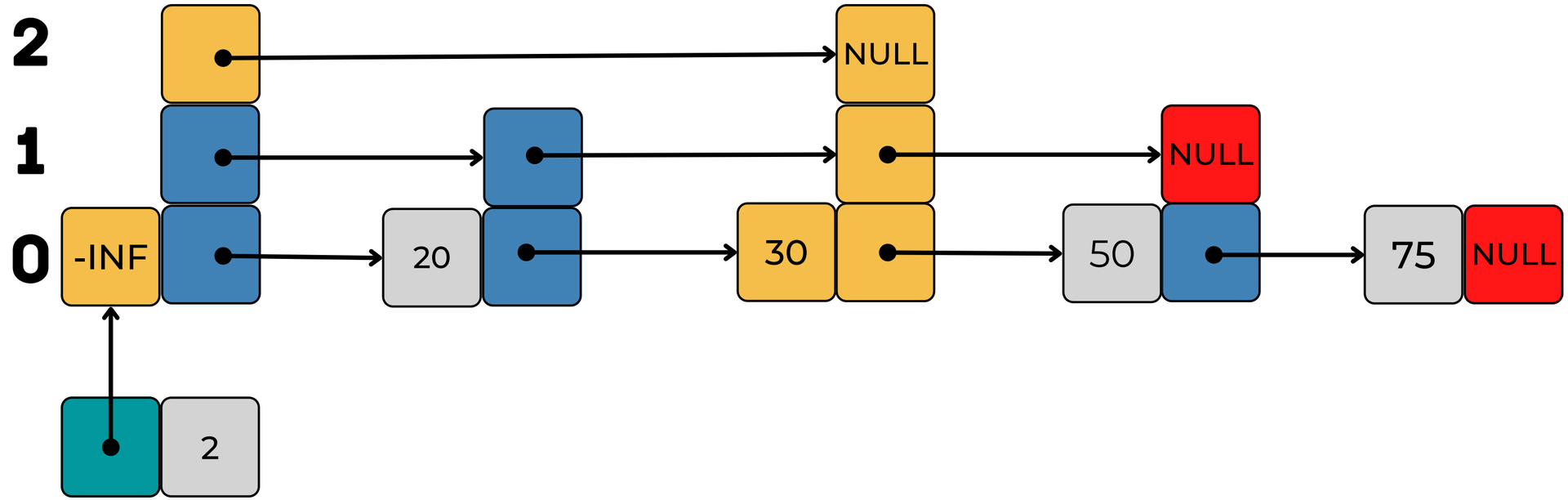
TEAM JC

Search . . .

SEARCH

Level = 0

40>50? **False** = **Move down a level**



SEARCH(40)

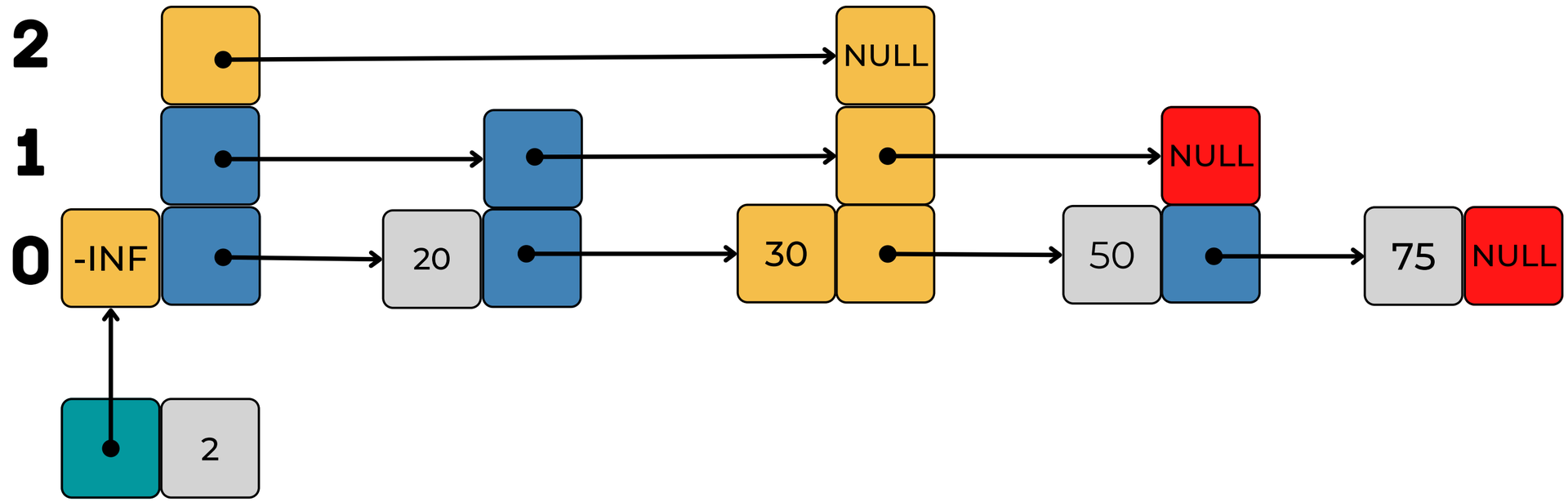
TEAM JC

Search . . .

SEARCH

Level = -1

since level < 0 then return the next node



SEARCH(40)

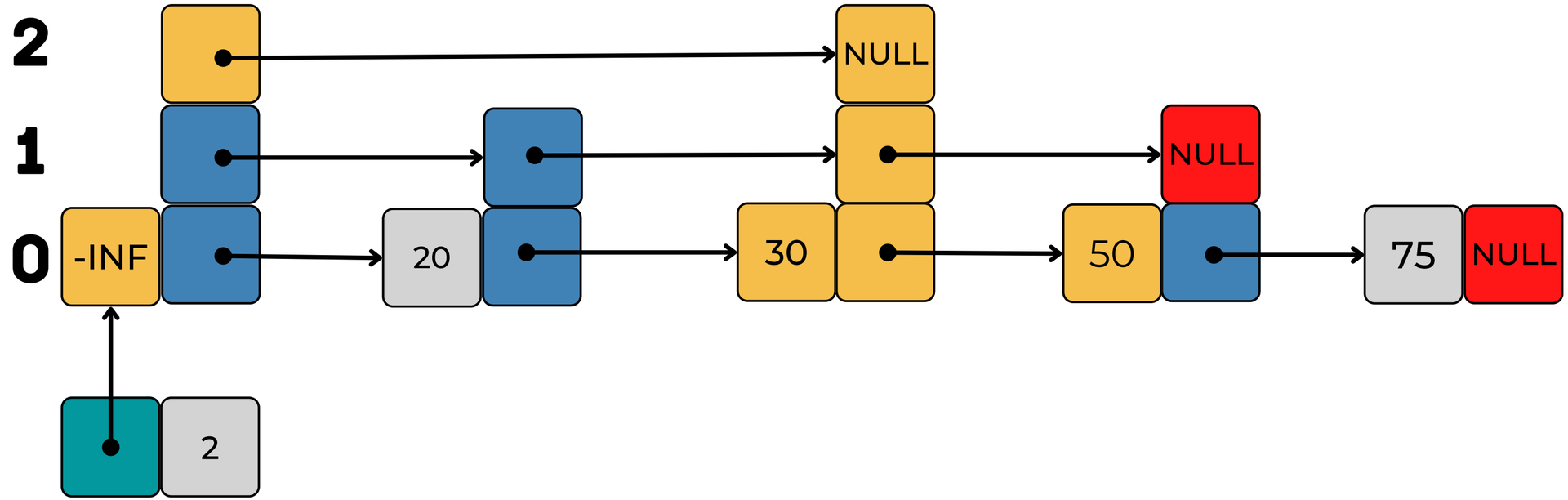
TEAM JC

Search . . .

SEARCH

Level = -1

40 == 50? **False then 40 is not an element from the skiplist**



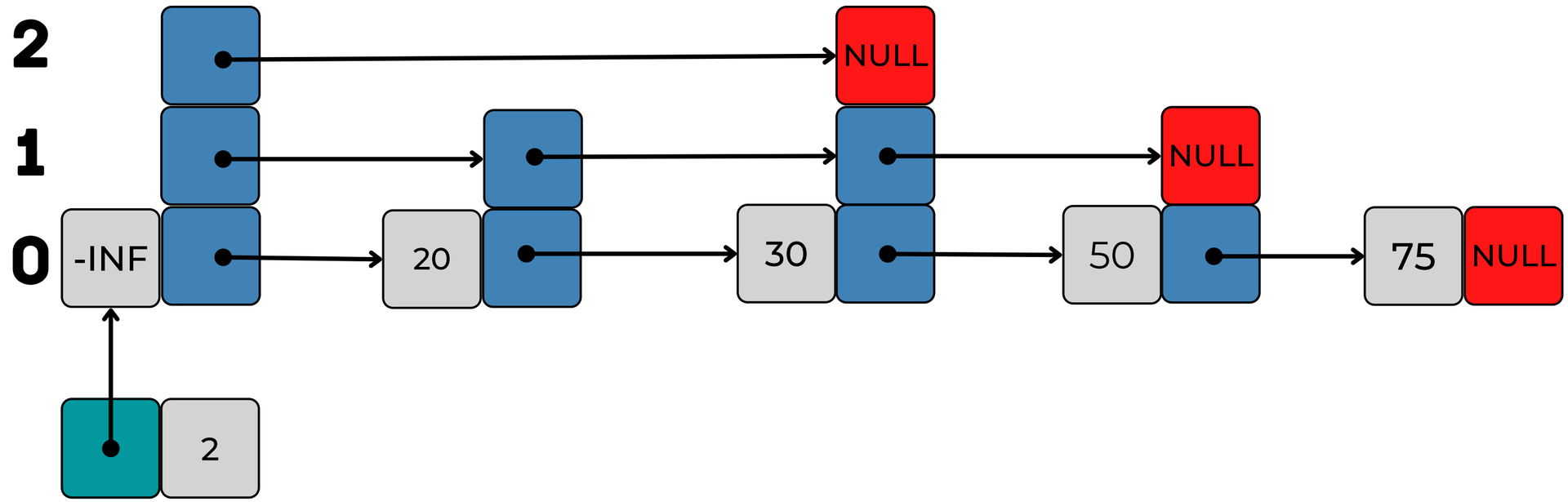
SEARCH(40)

TEAM JC

Search . . .

DELETION

Level = 2

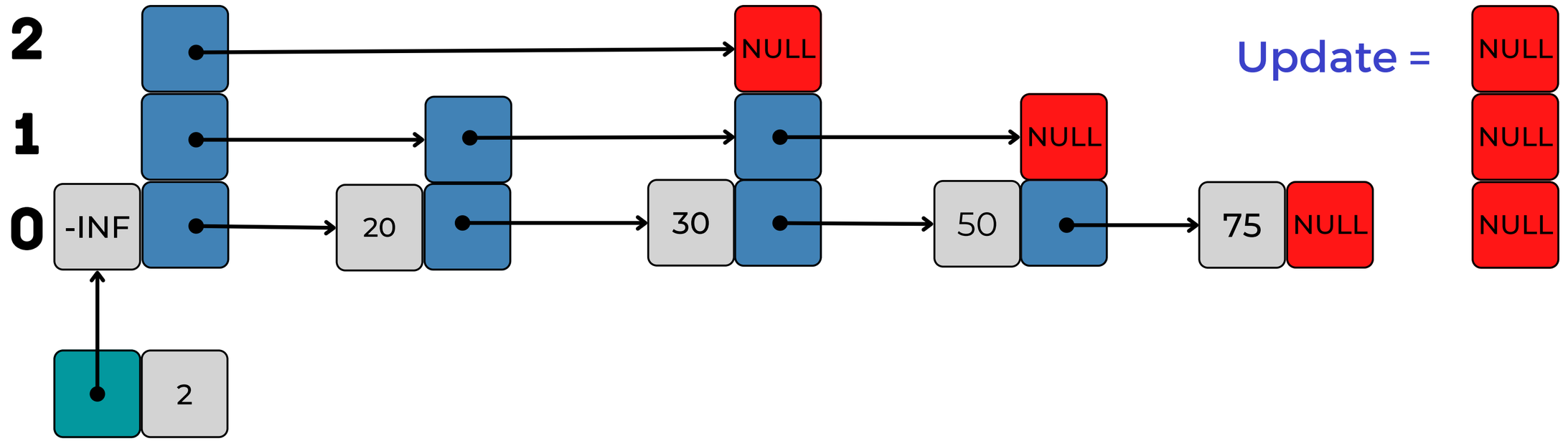


TEAM JC

Search . . .

DELETION

Level = 2



DELETE(50)

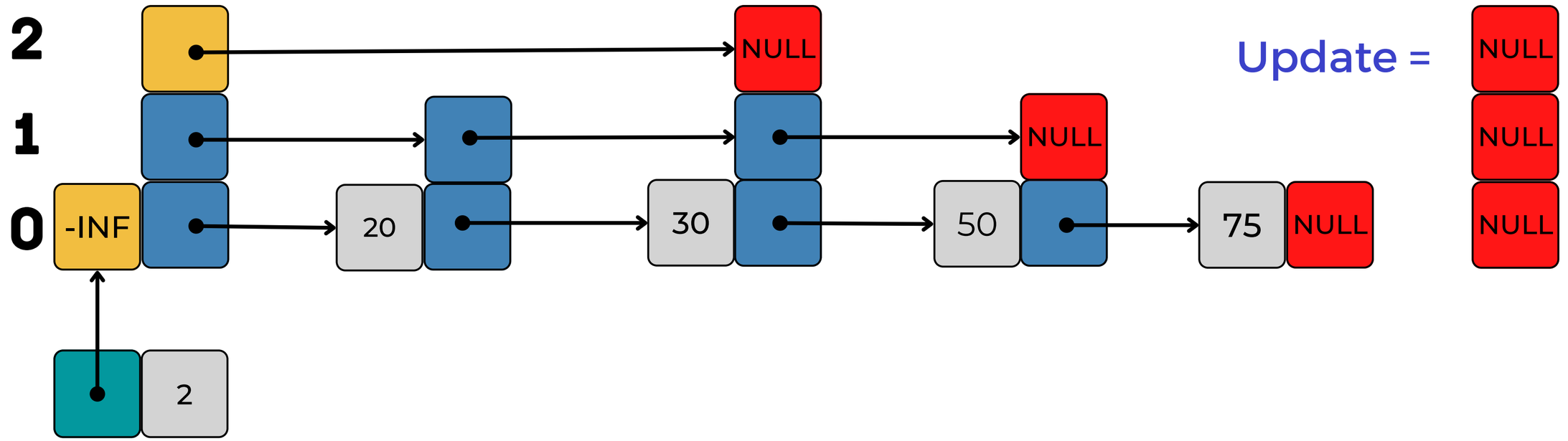
TEAM JC

Search . . .

DELETION

Level = 2

50>30?



Update =

DELETE(50)

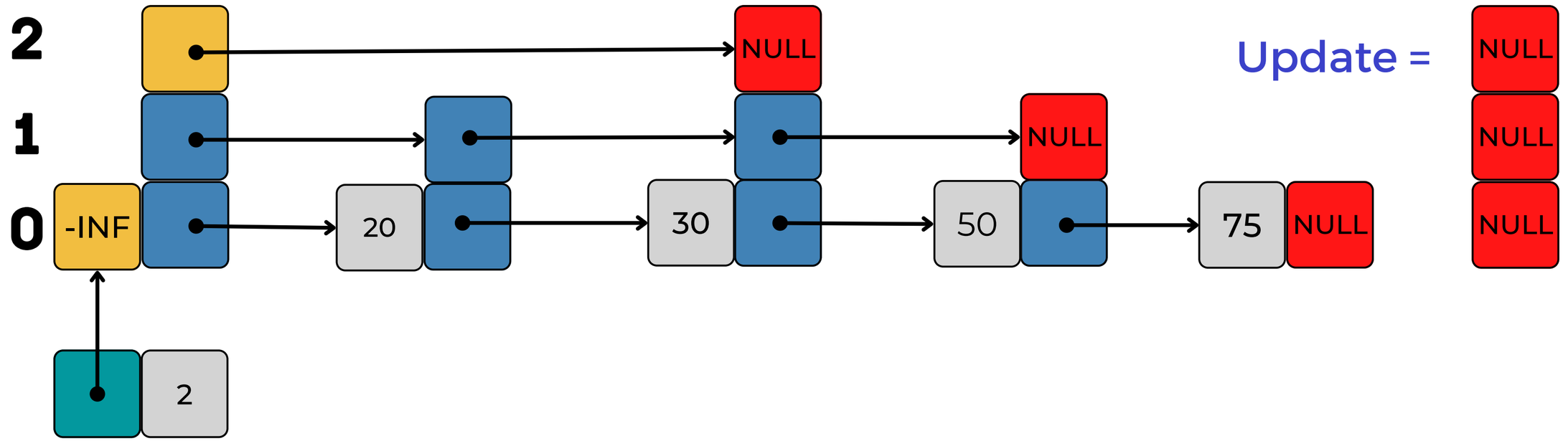
TEAM JC

Search . . .

DELETION

Level = 2

50>30? **True** = **Move to next node**



Update =

DELETE(50)

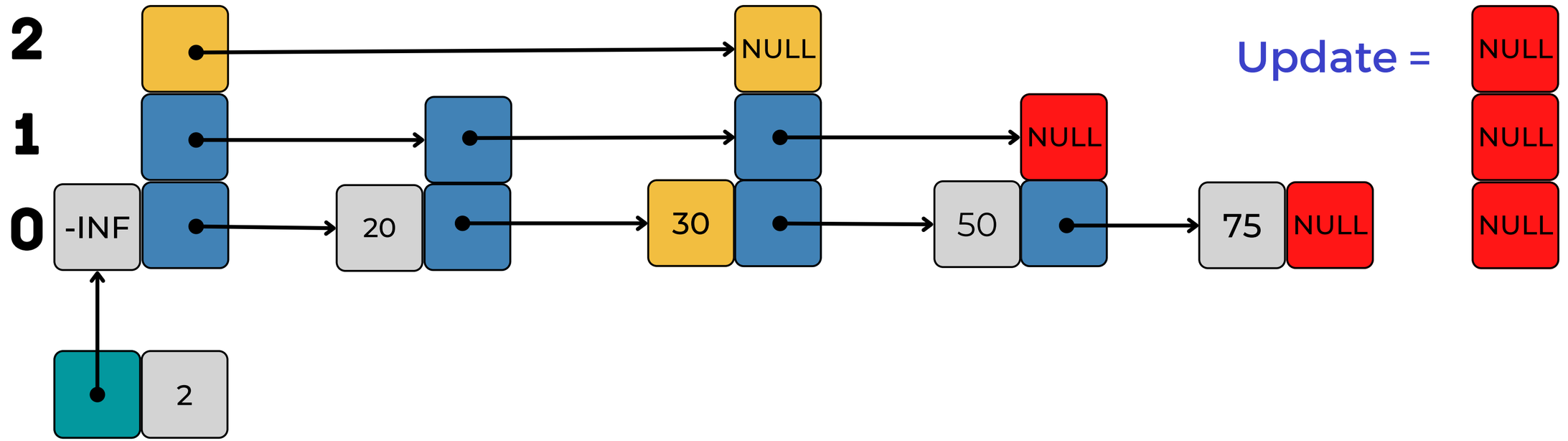
TEAM JC

Search . . .

DELETION

Level = 2

NULL?



Update =

DELETE(50)

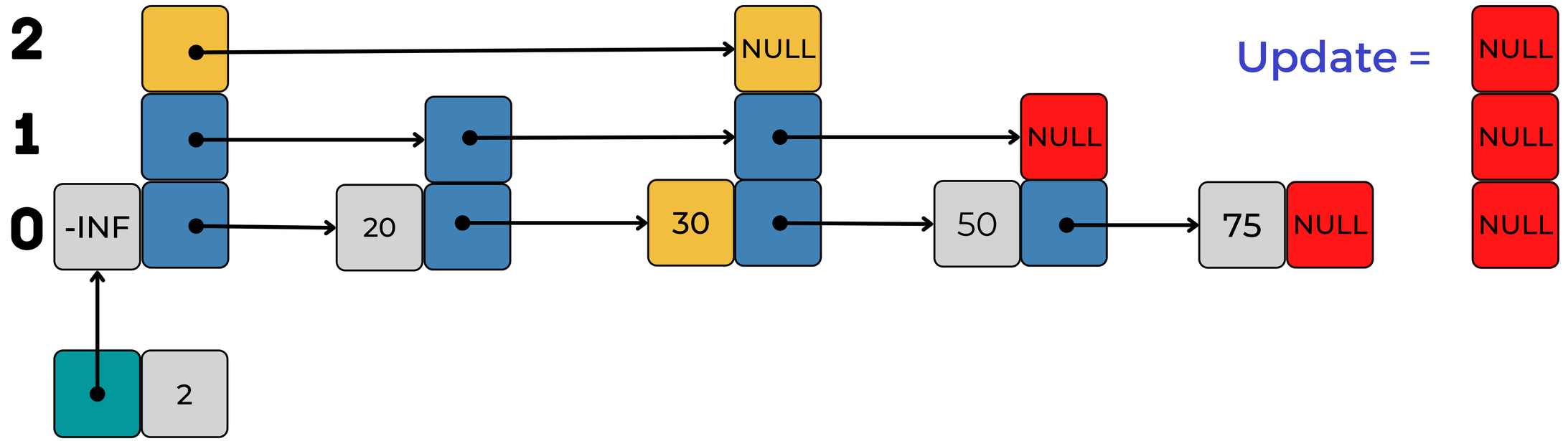
TEAM JC

Search . . .

DELETION

Level = 2

NULL? **True** = **Move down a level**



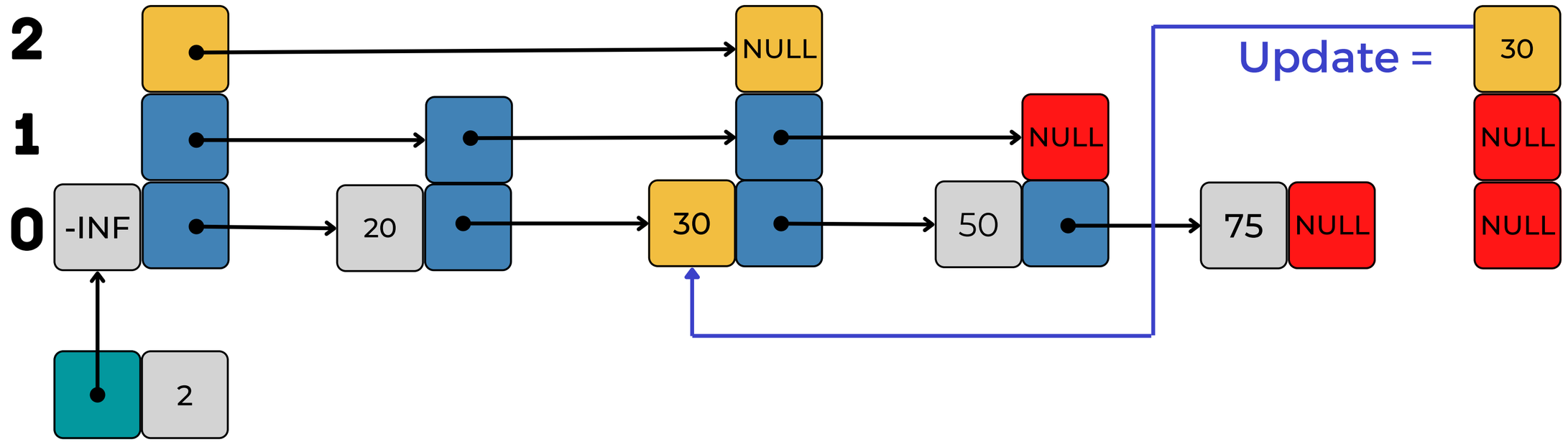
DELETE(50)

TEAM JC

Search . . .

DELETION

Level = 2



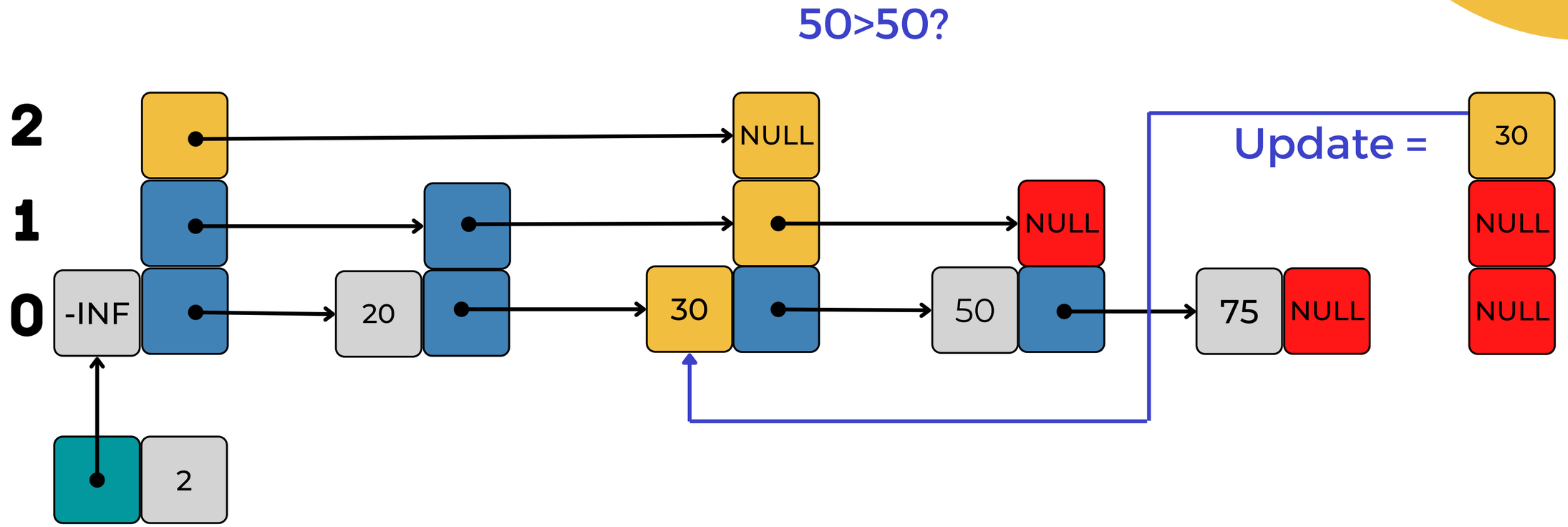
DELETE(50)

TEAM JC

Search . . .

DELETION

Level = 1



DELETE(50)

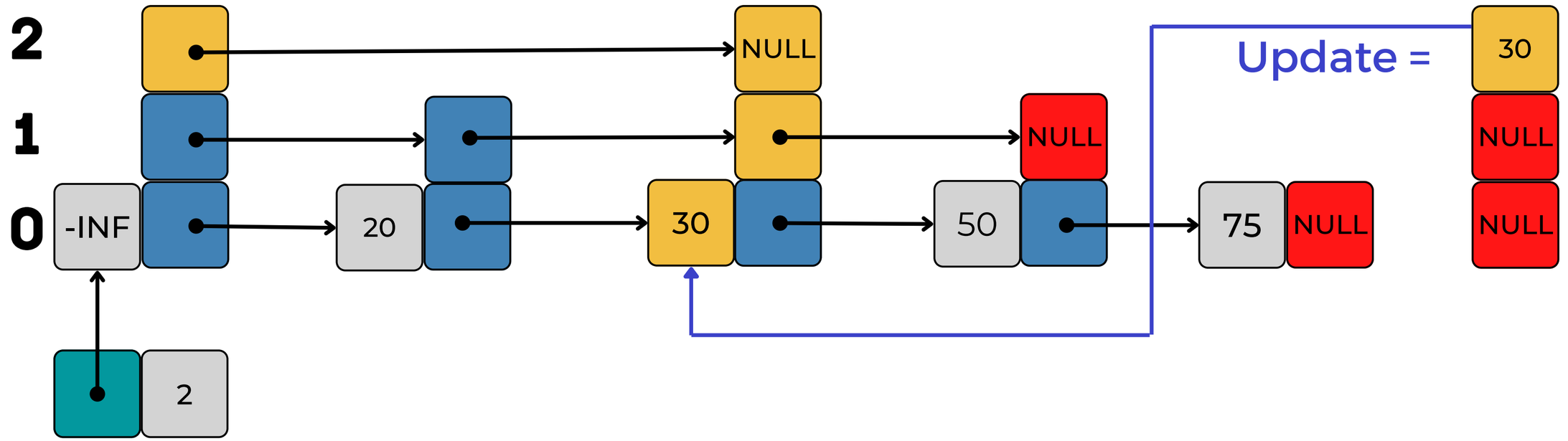
TEAM JC

Search . . .

DELETION

Level = 1

50>50? **False** = **Move down a level**



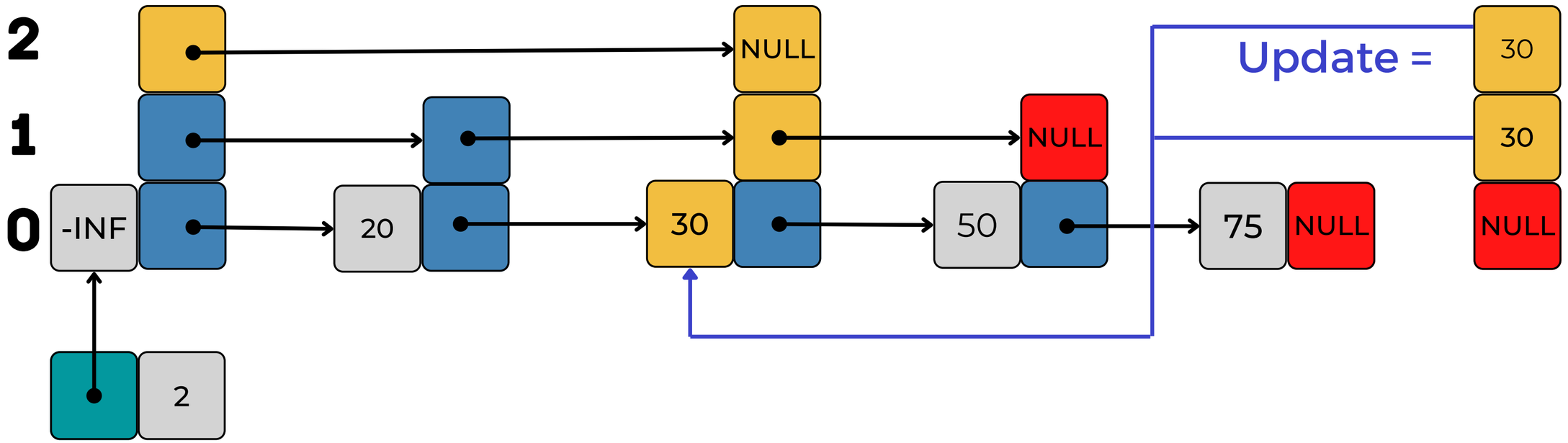
DELETE(50)

TEAM JC

Search . . .

DELETION

Level = 1



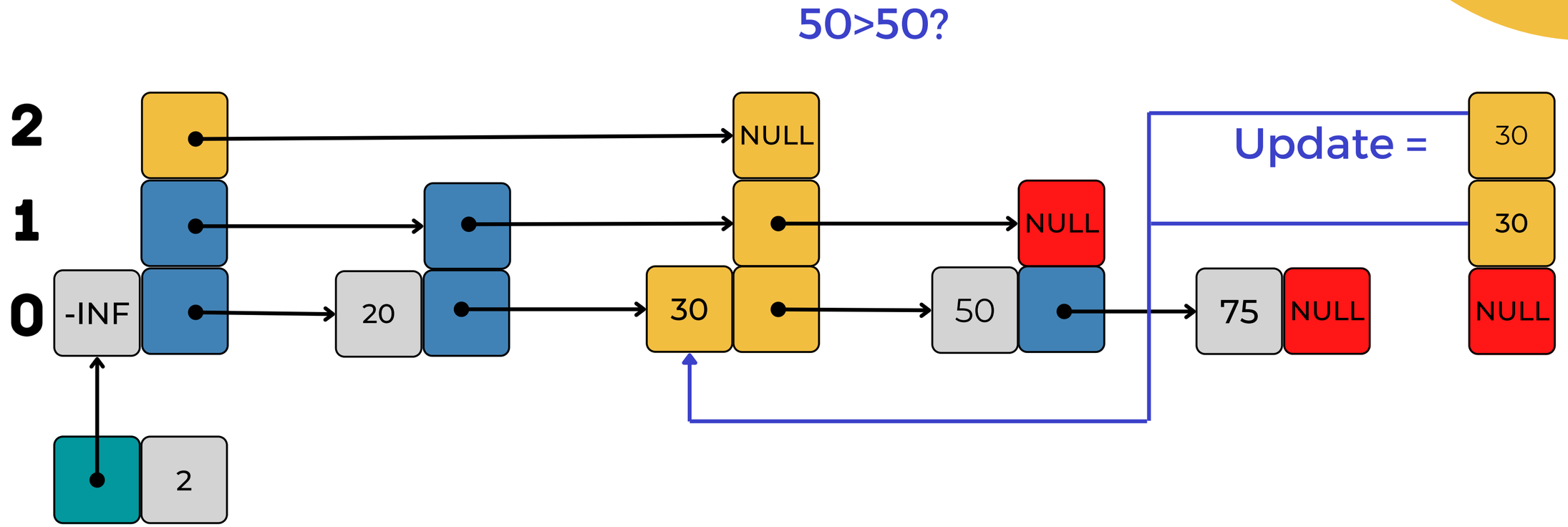
DELETE(50)

TEAM JC

Search . . .

DELETION

Level = 0



DELETE(50)

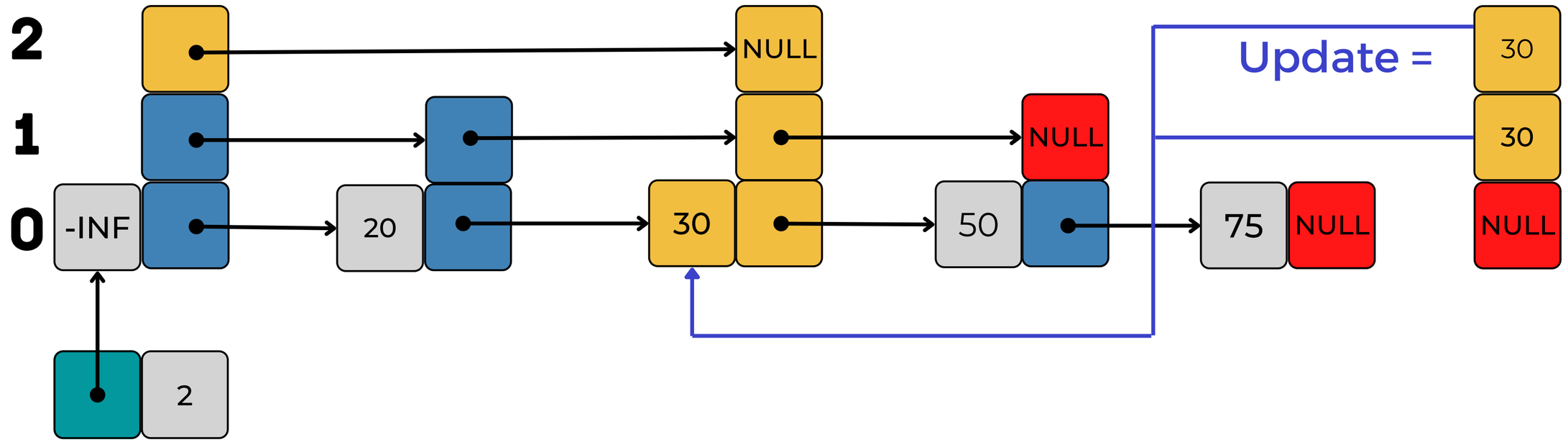
TEAM JC

Search . . .

DELETION

Level = 0

50>50? **False** = **Move down a level**



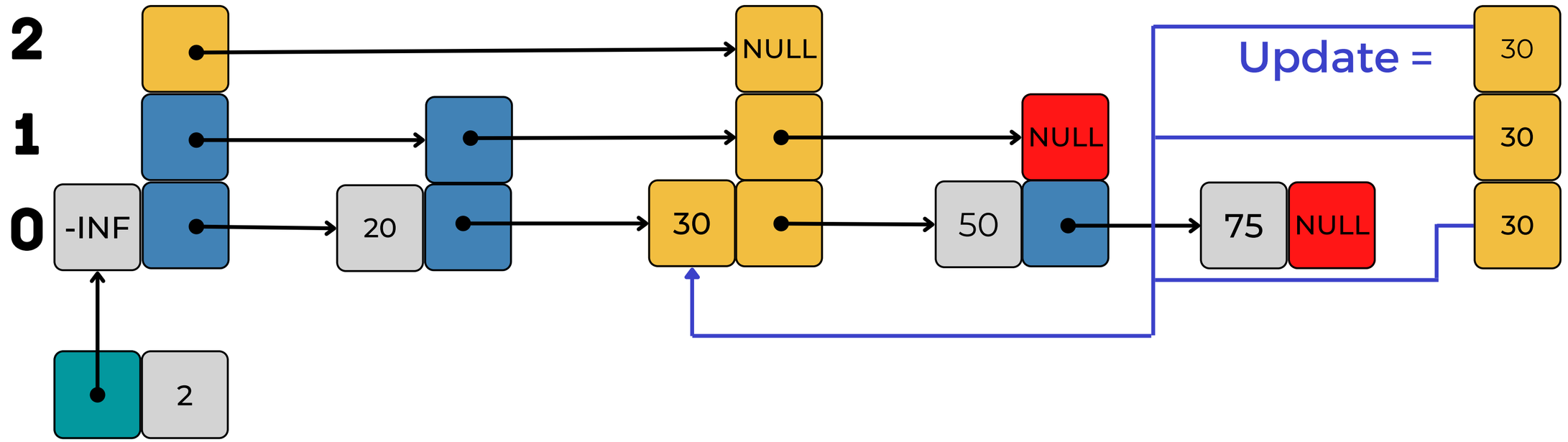
DELETE(50)

TEAM JC

Search . . .

DELETION

Level = -1



DELETE(50)

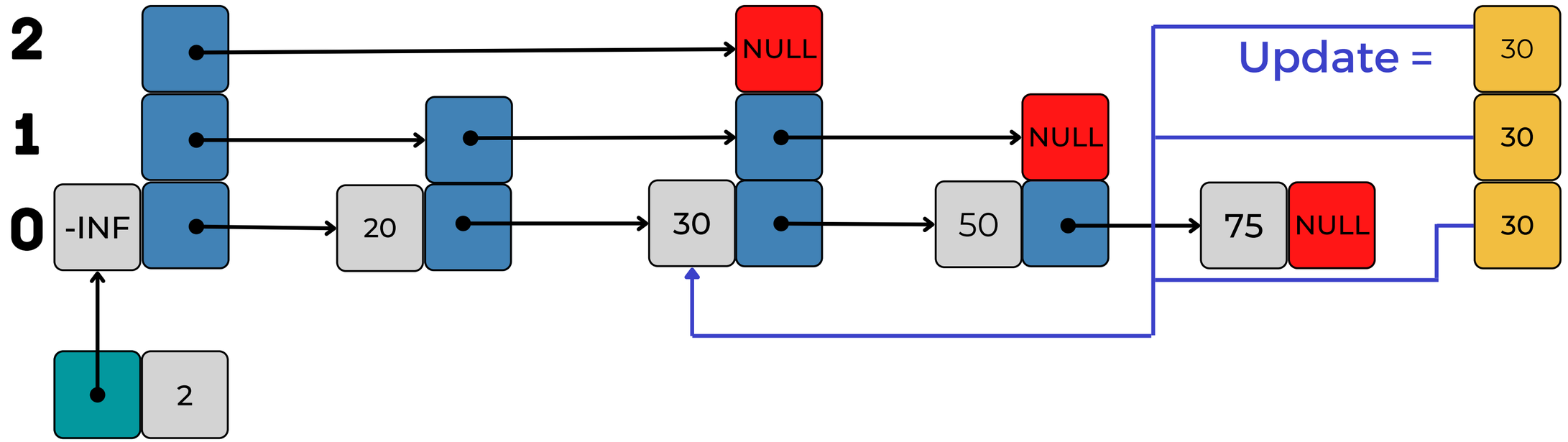
TEAM JC

Search . . .

DELETION

Level = -1

Level -1 meaning end of traversal

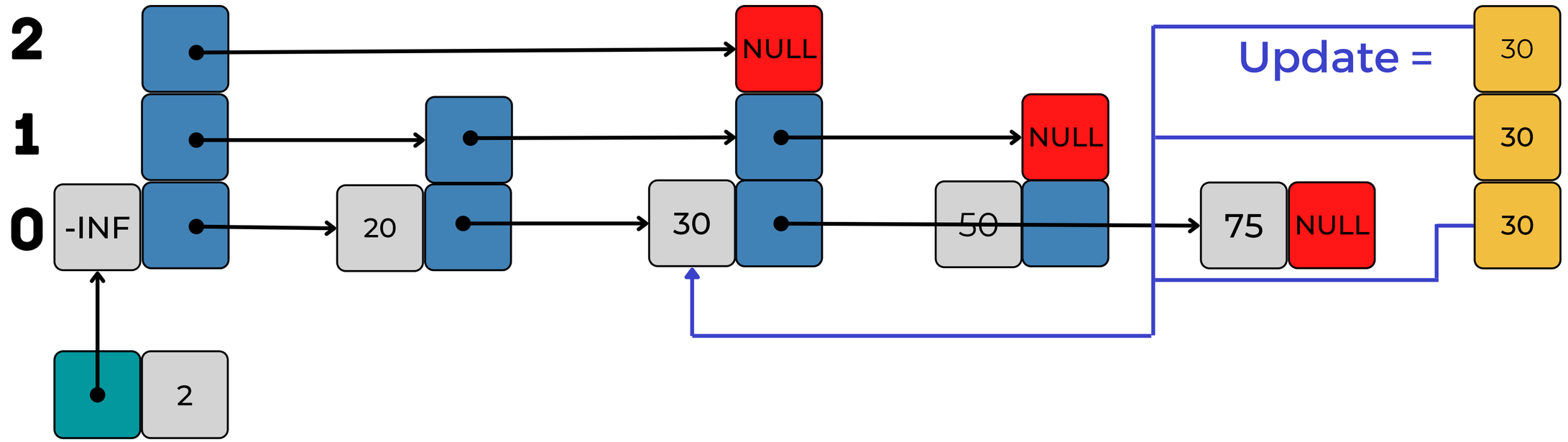


DELETE(50)

DELETION

Level = -1

Connect update array node to node after



DELETE(50)

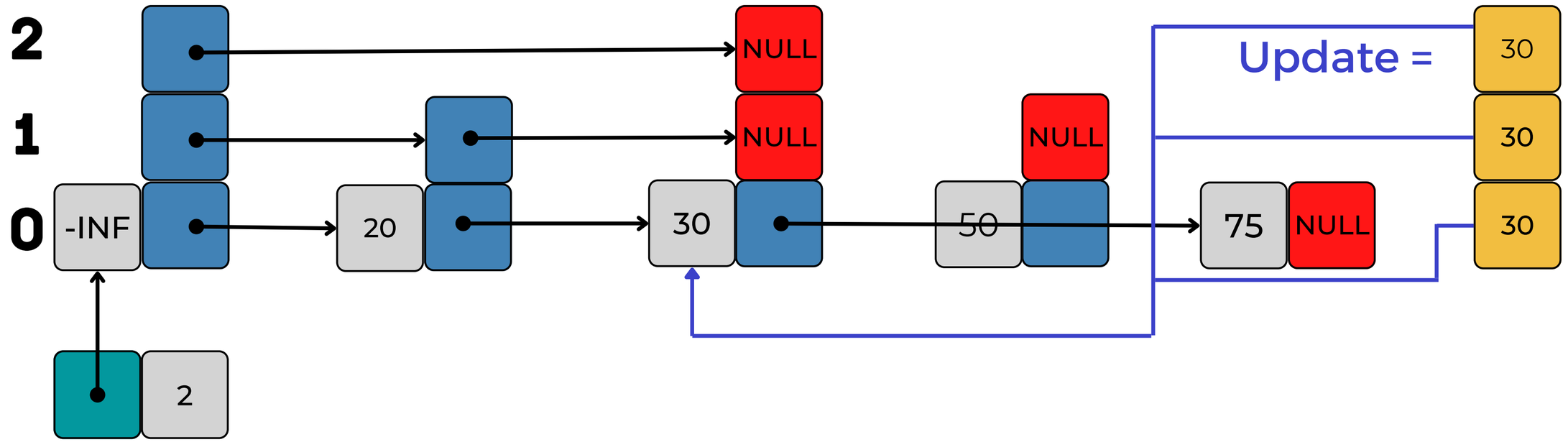
TEAM JC

Search . . .

DELETION

Level = -1

Connect update array node to node after



DELETE(50)

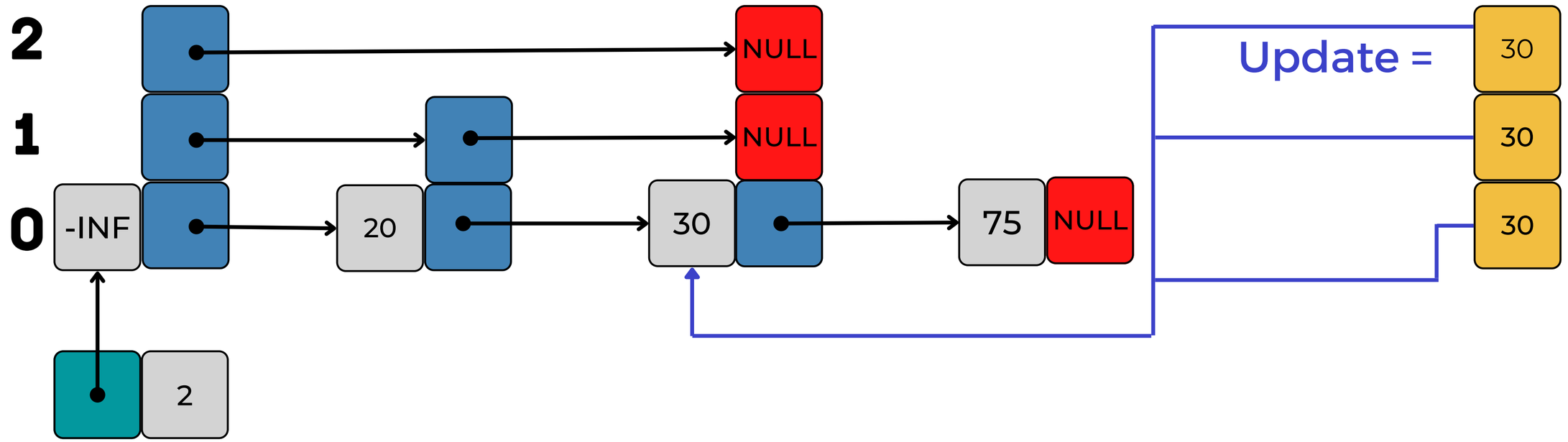
TEAM JC

Search . . .

DELETION

Level = -1

node 50 is now deleted



DELETE(50)

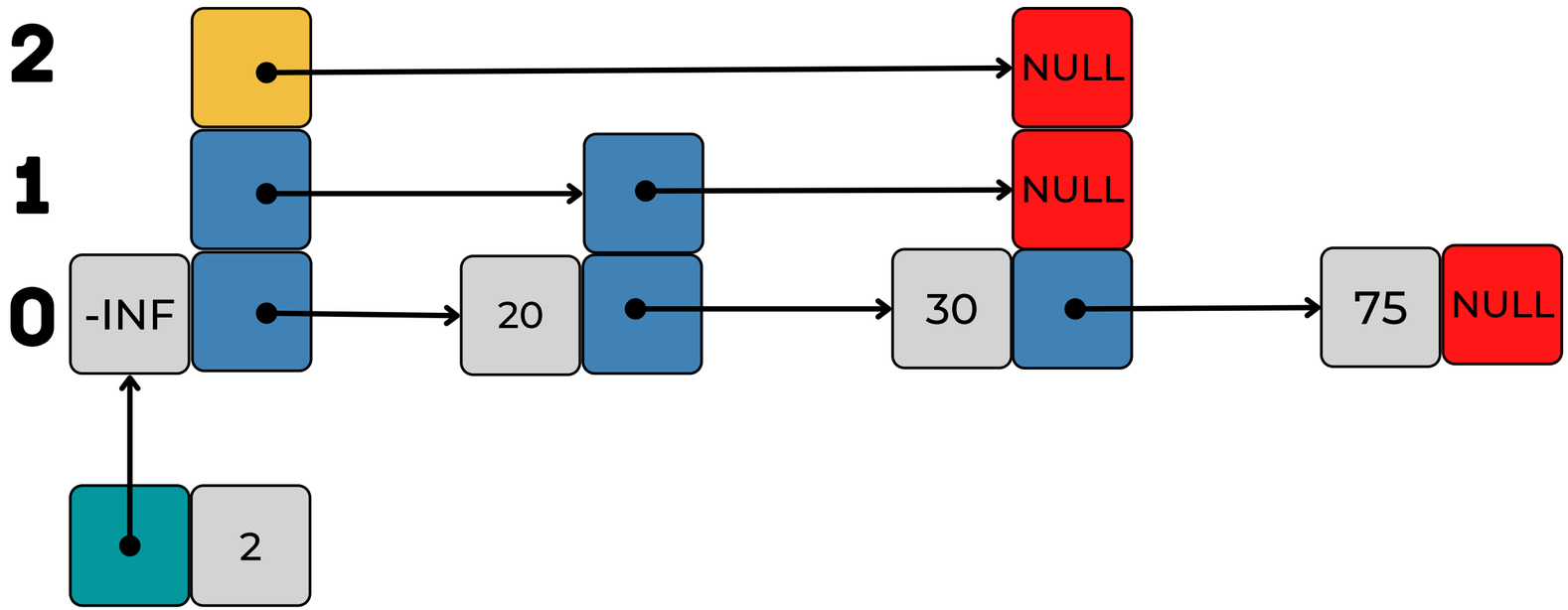
TEAM JC

Search . . .

DELETION

Level = -1

Is head link (lvl 2) null?



DELETE(50)

TEAM JC

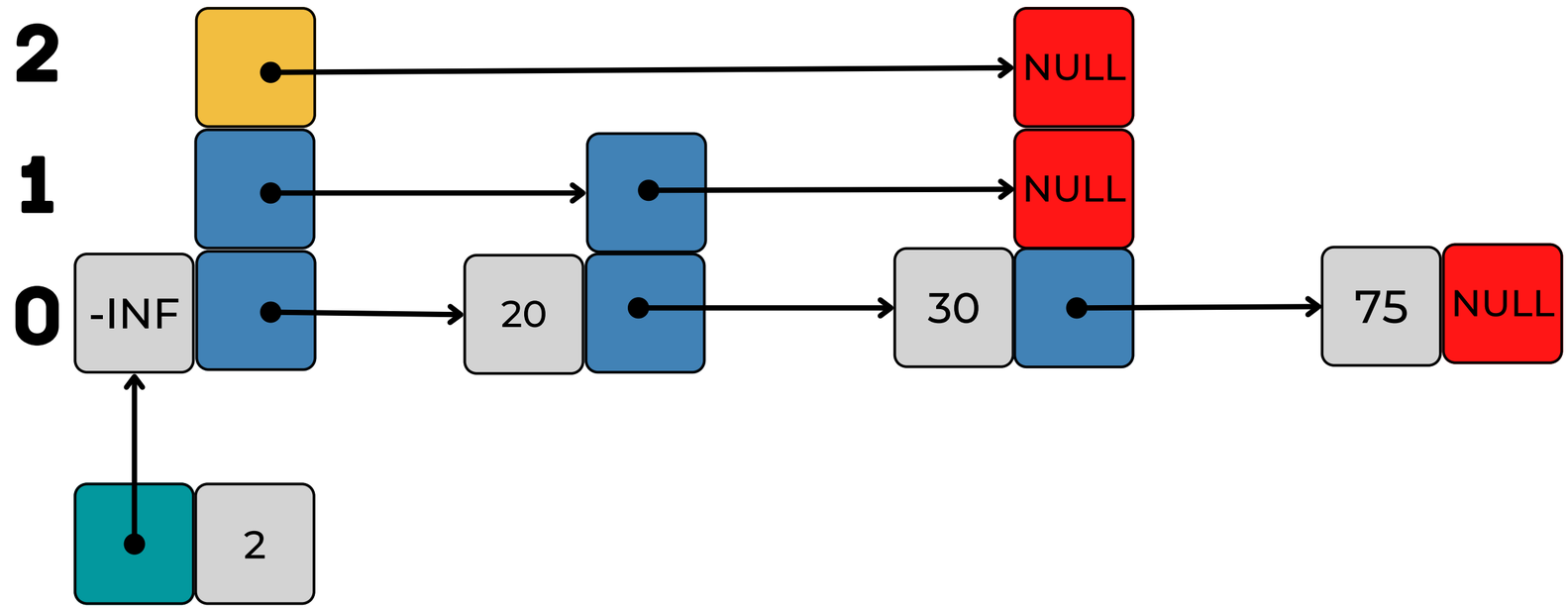
Search . . .

DELETION

Level = -1

Is head link (lvl 2) null?

False = **End traversal**



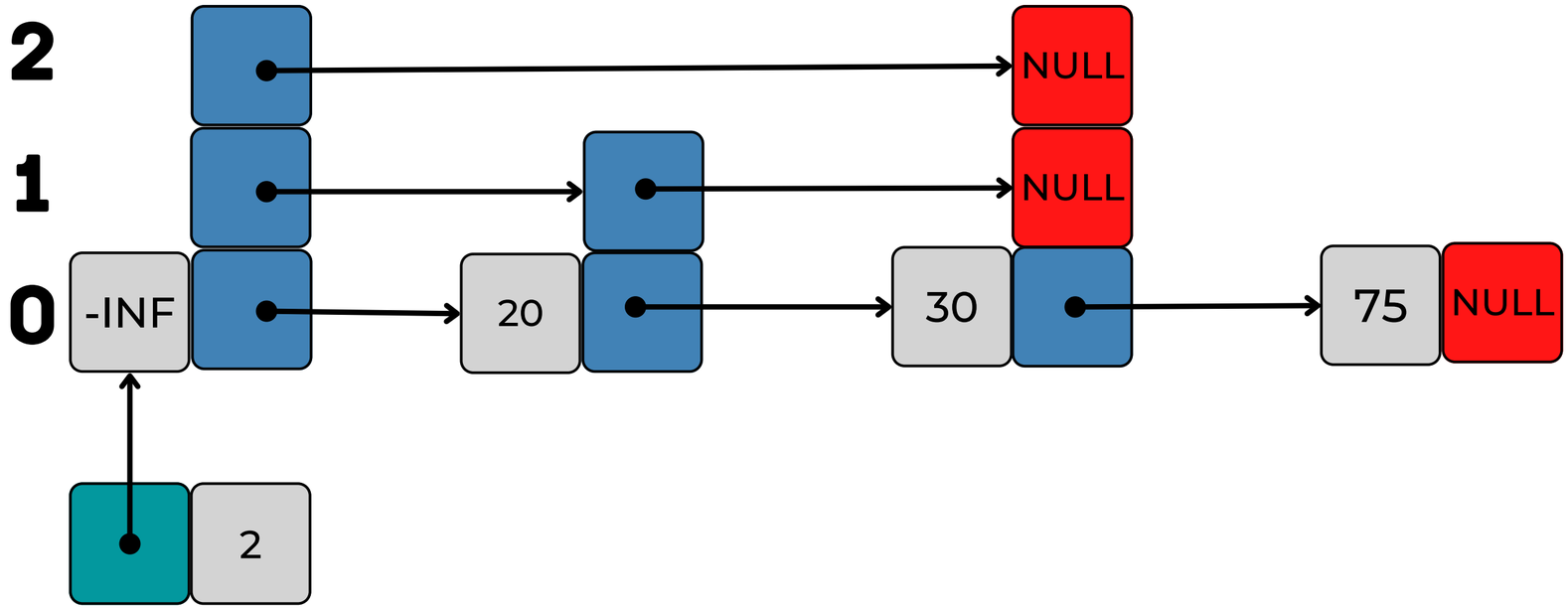
DELETE(50)

TEAM JC

Search . . .

DELETION

Level = 2

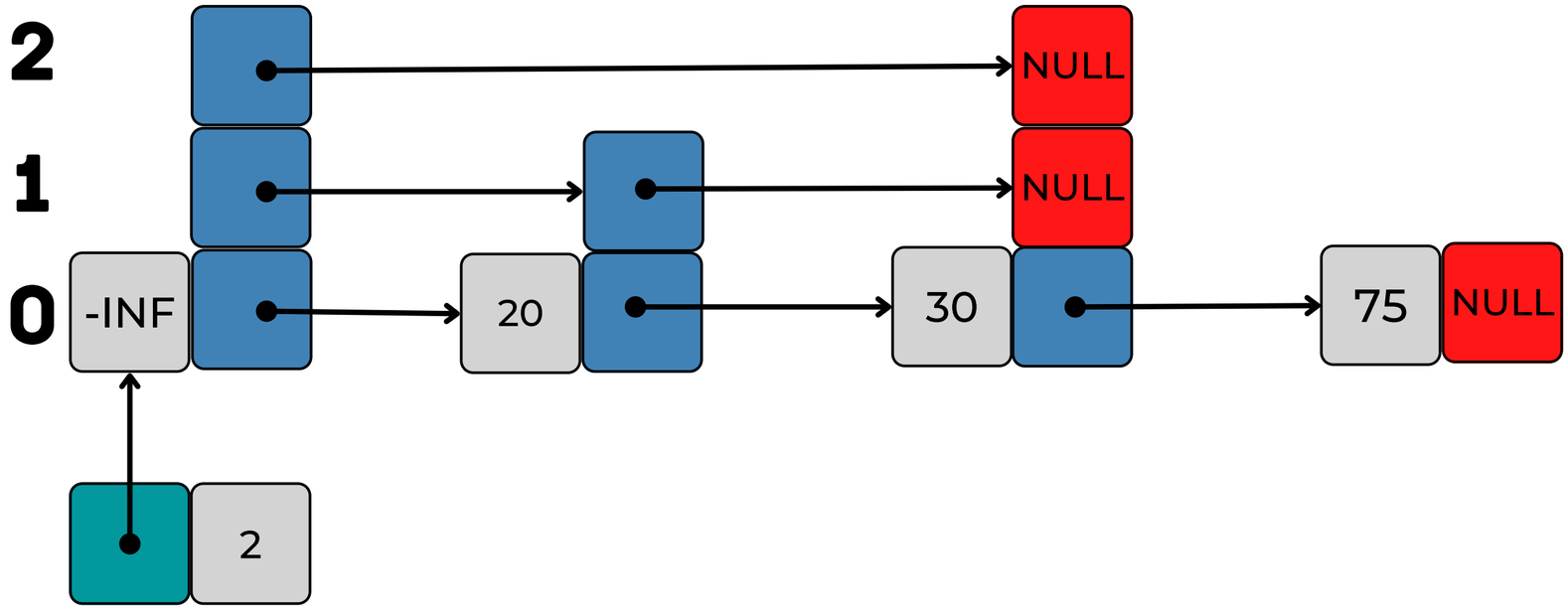


TEAM JC

Search . . .

DELETION

Level = 2



Update =

DELETE(30)

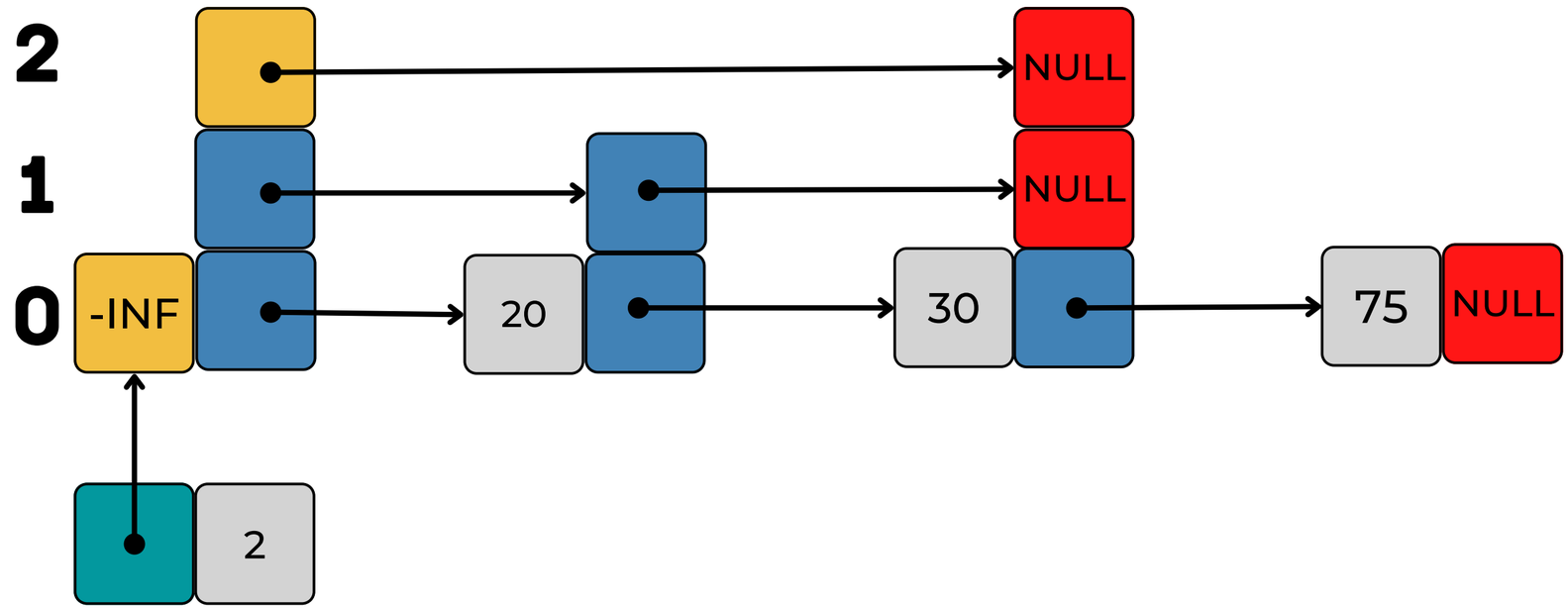
TEAM JC

Search . . .

DELETION

Level = 2

30>30?



Update =

DELETE(30)

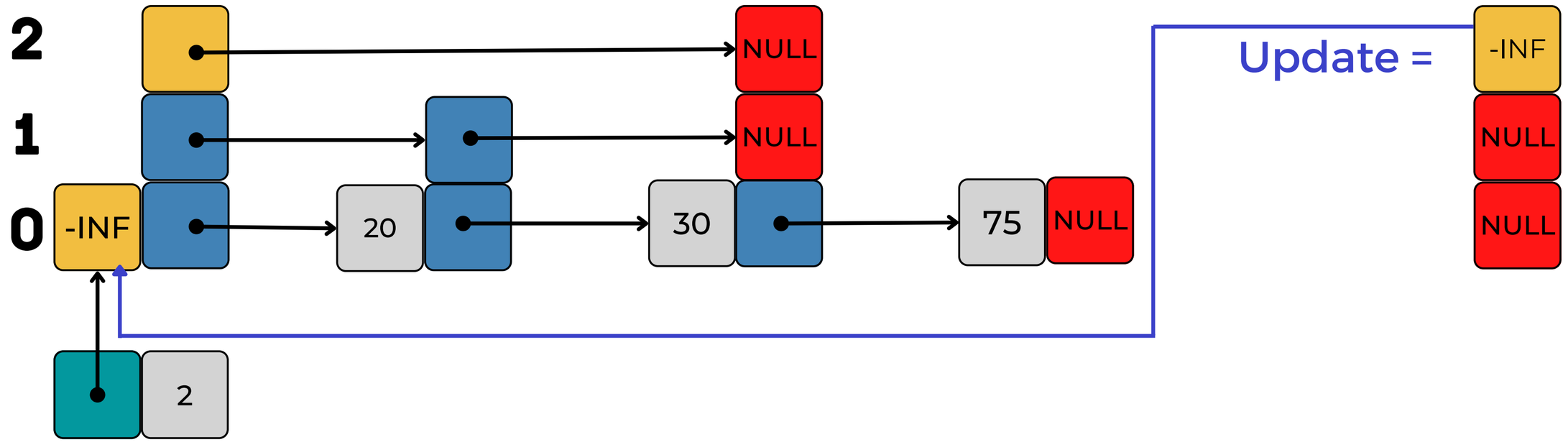
TEAM JC

Search . . .

DELETION

Level = 2

30>30? **True** = **Move to next node**



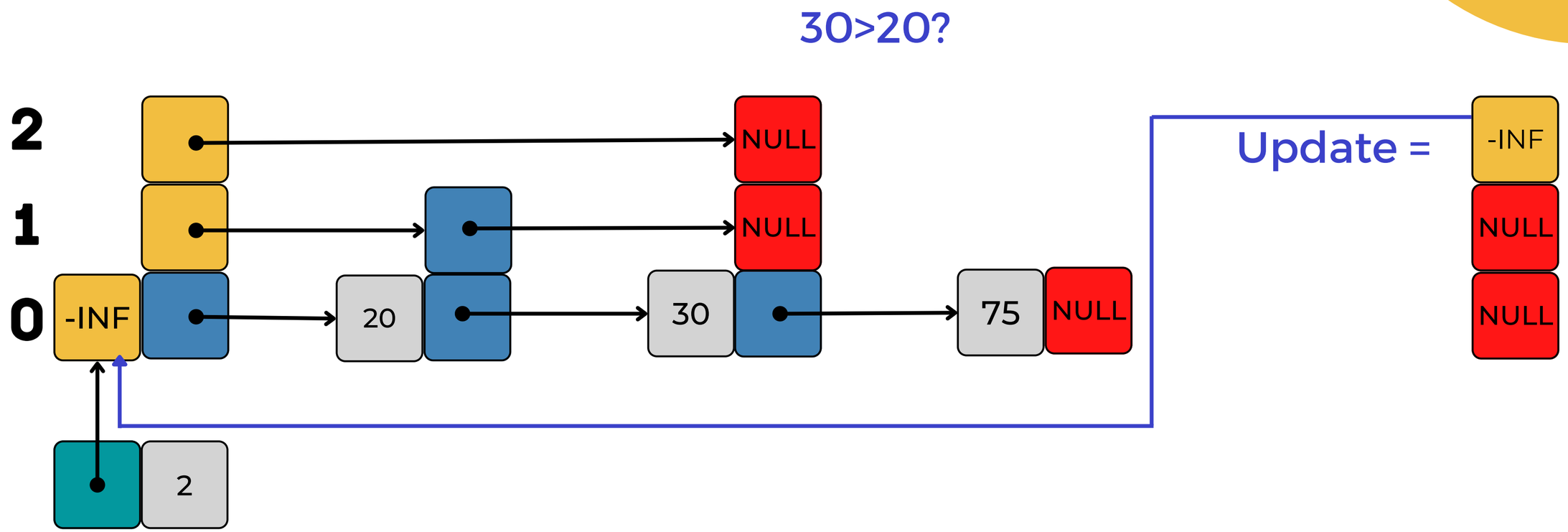
DELETE(30)

TEAM JC

Search . . .

DELETION

Level = 1



DELETE(30)

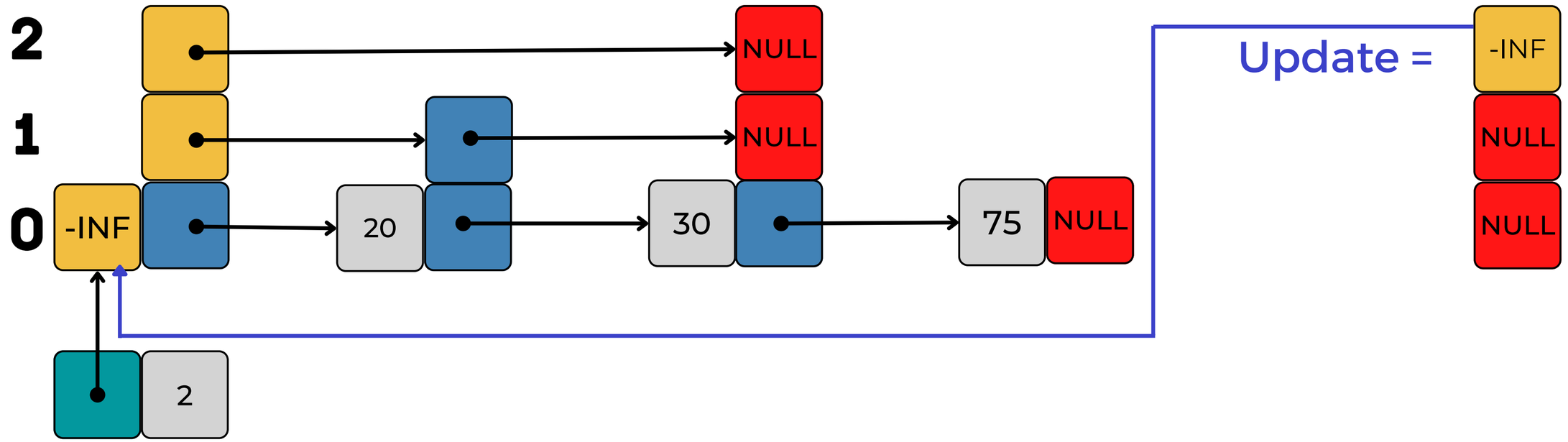
TEAM JC

Search . . .

DELETION

Level = 1

30>20? **True** = **Move to next node**



DELETE(30)

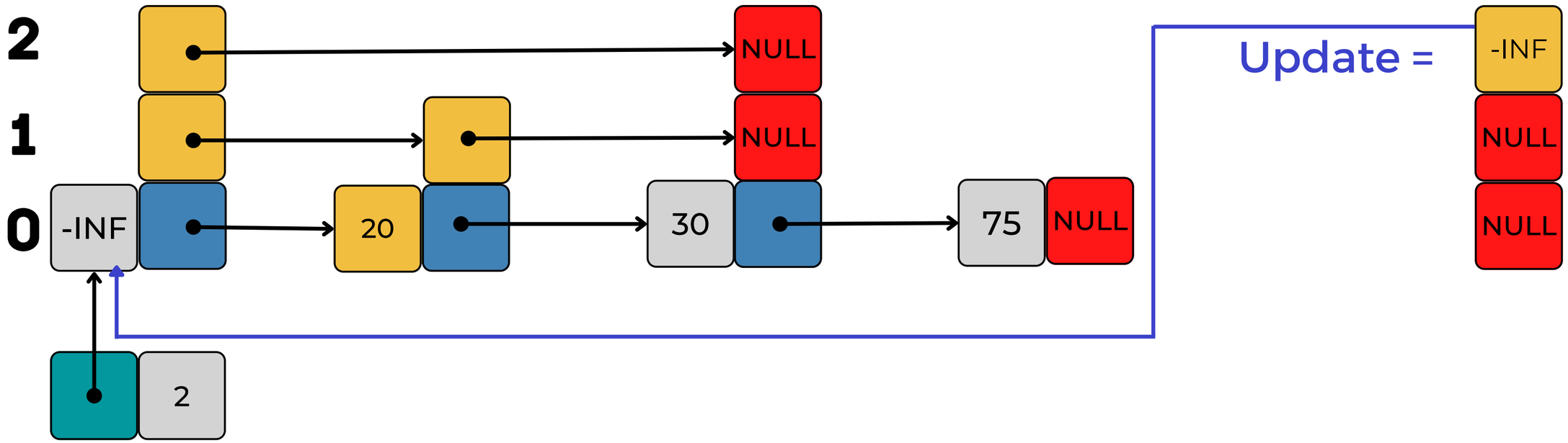
TEAM JC

Search . . .

DELETION

Level = 1

30>30?



DELETE(30)

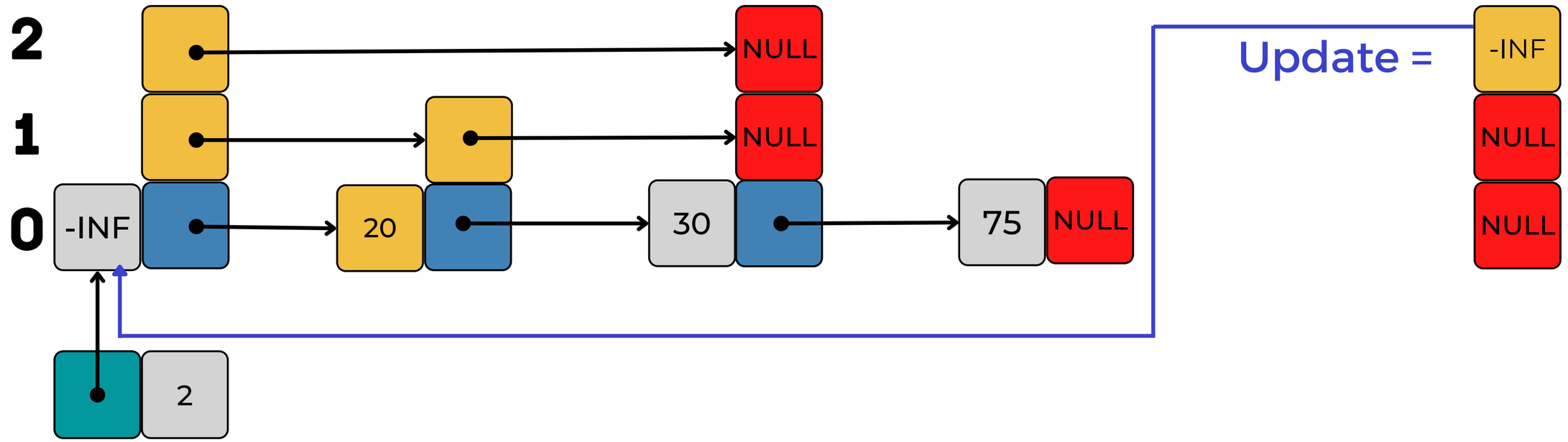
TEAM JC

Search . . .

DELETION

Level = 1

30>30? **False** = **Move down a level**



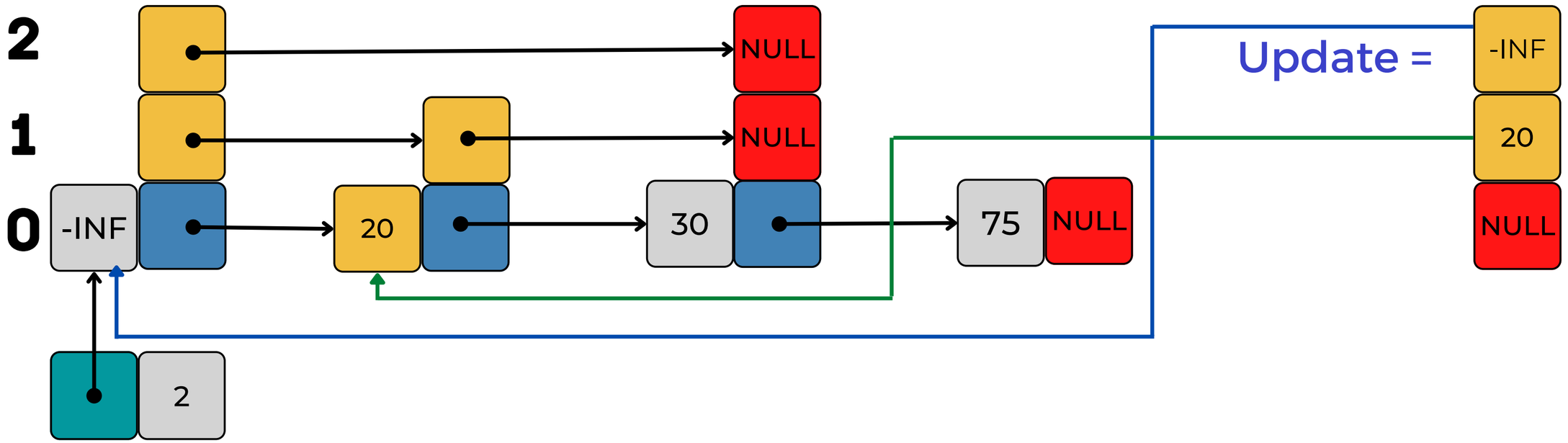
DELETE(30)

TEAM JC

Search . . .

DELETION

Level = 1



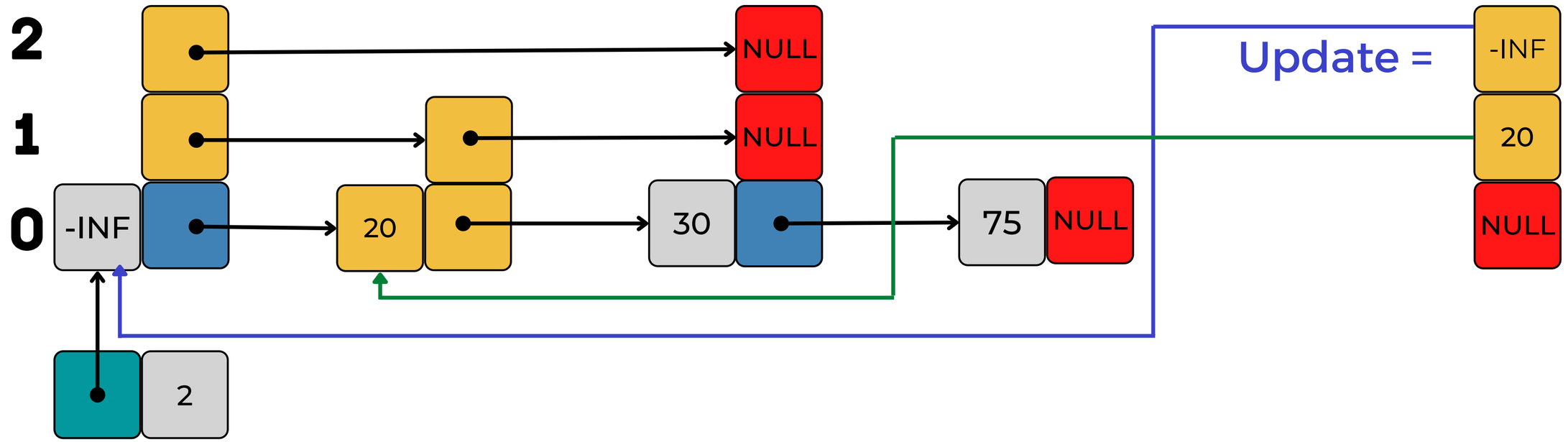
DELETE(30)

Search . . .

DELETION

Level = 0

30>30?



DELETE(30)

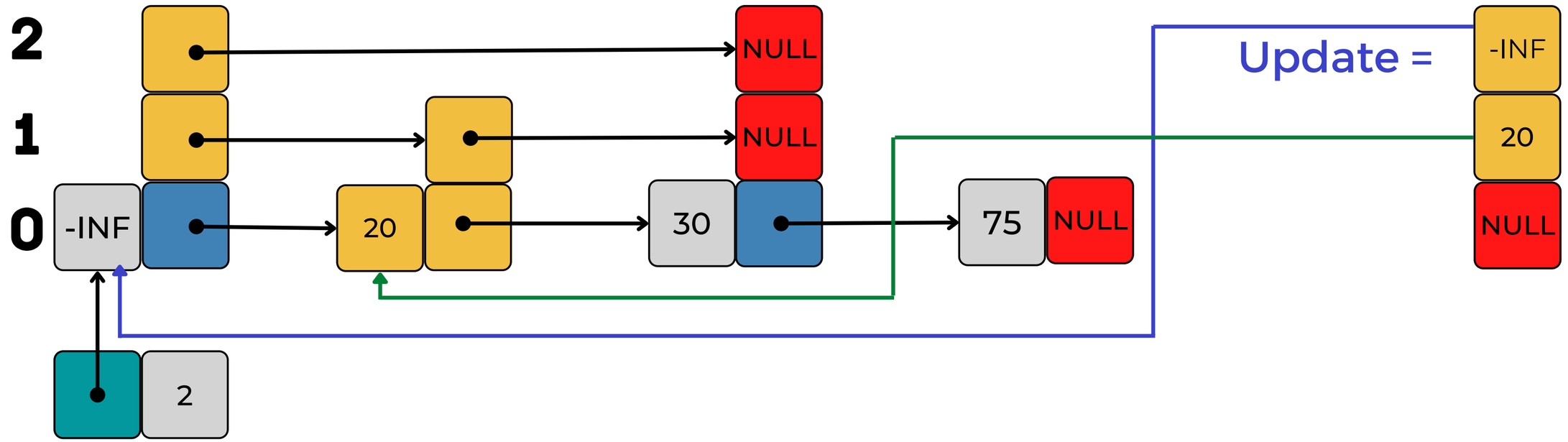
TEAM JC

Search . . .

DELETION

Level = 0

30>30? **False** = **Move down a level**



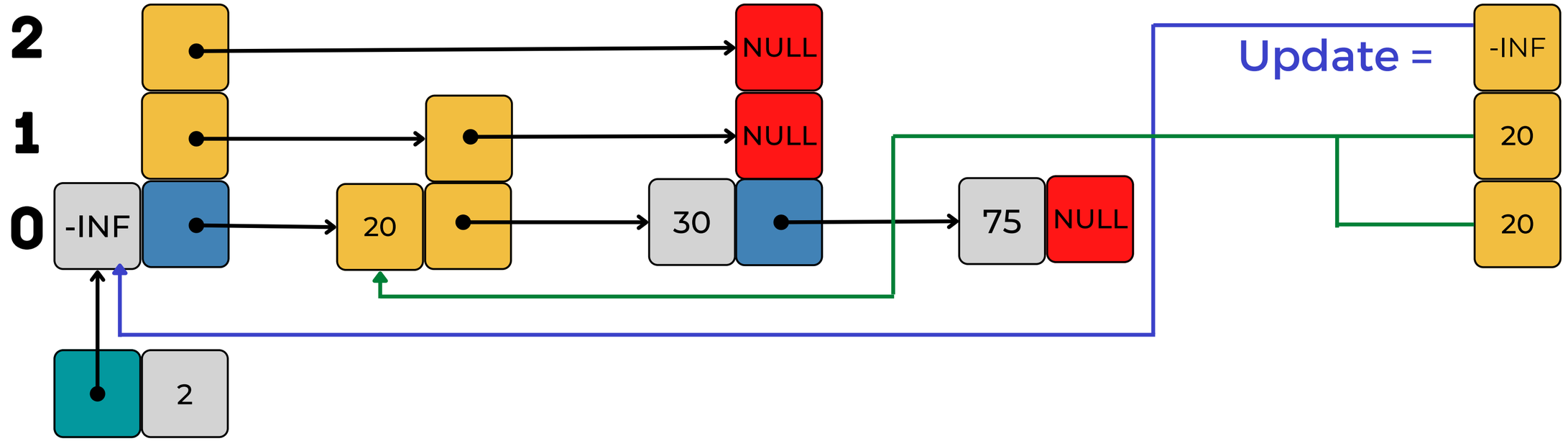
DELETE(30)

TEAM JC

Search . . .

DELETION

Level = 0



DELETE(30)

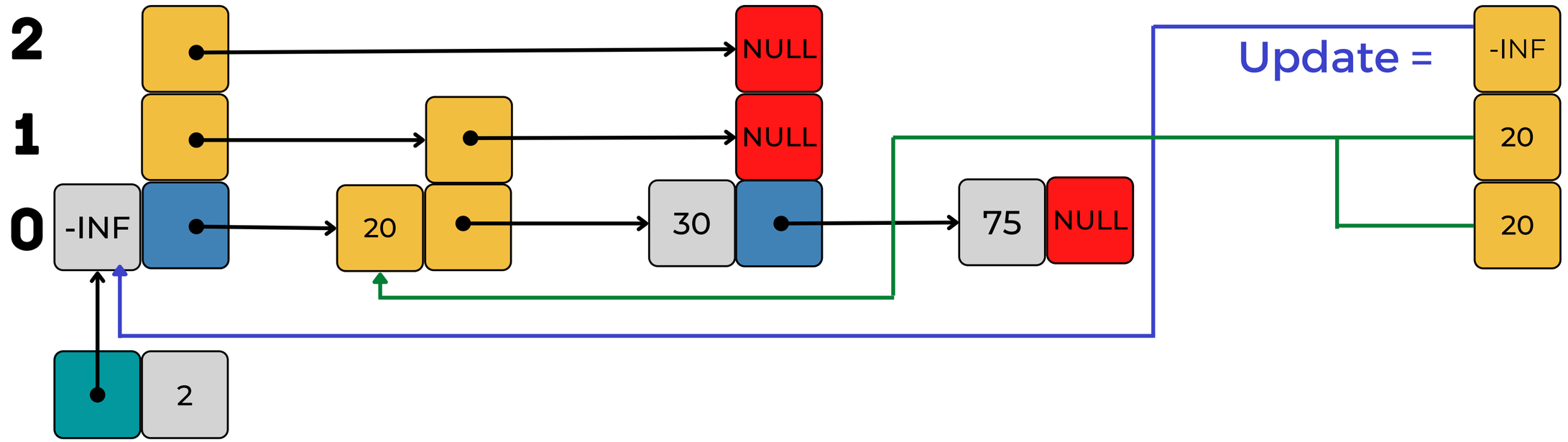
TEAM JC

Search . . .

DELETION

Level = -1

Level -1 meaning end of traversal



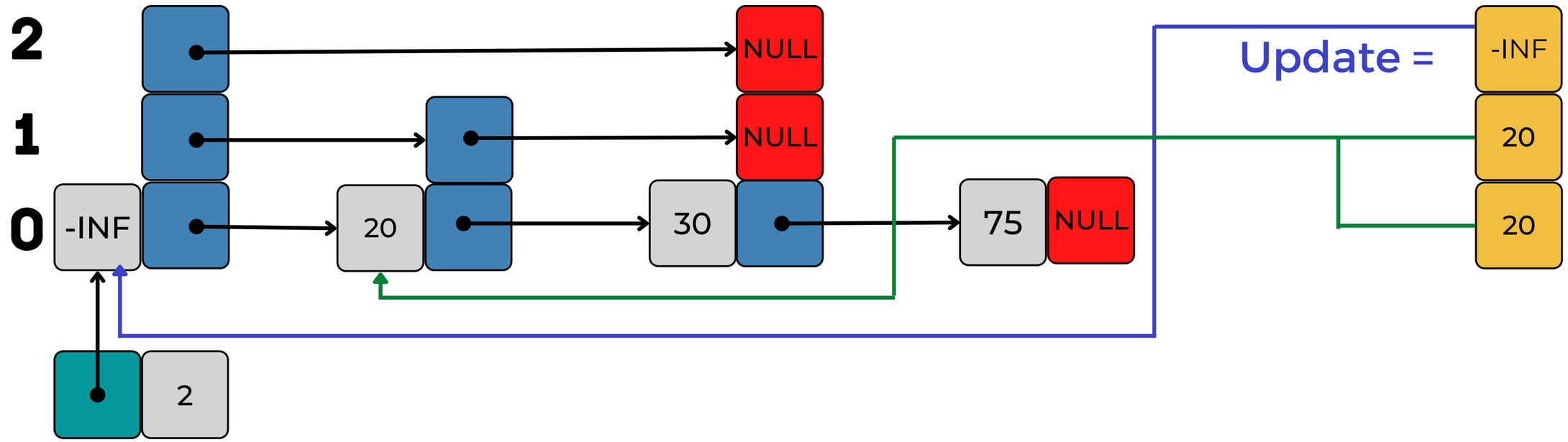
DELETE(30)

TEAM JC

Search . . .

DELETION

Level = -1



DELETE(30)

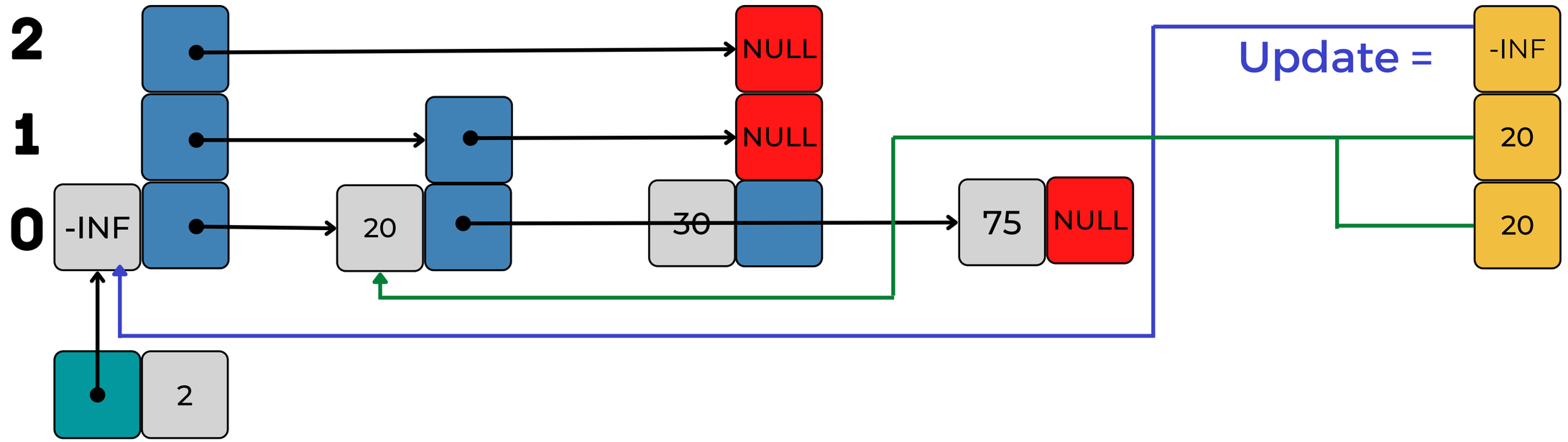
TEAM JC

Search . . .

DELETION

Level = -1

Connect update array node to node after



DELETE(30)

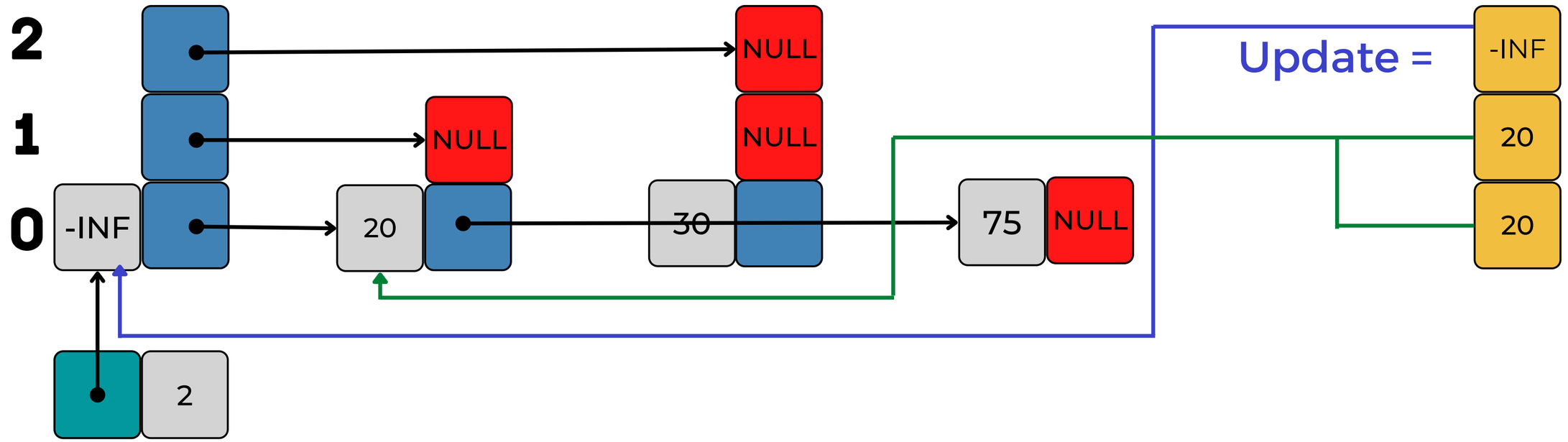
TEAM JC

Search . . .

DELETION

Level = -1

Connect update array node to node after



DELETE(30)

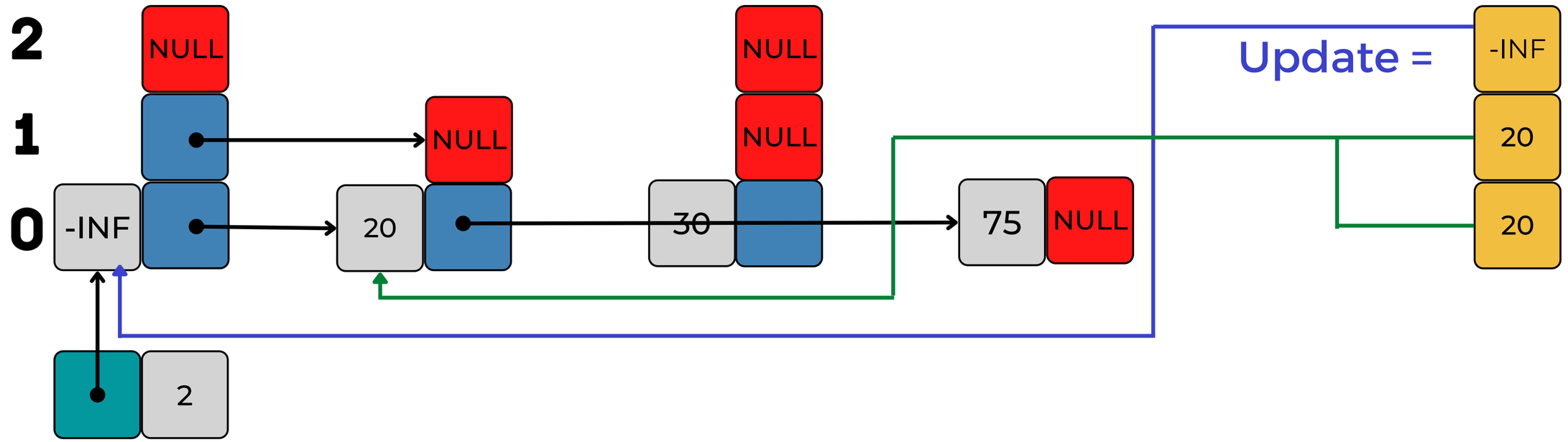
TEAM JC

Search . . .

DELETION

Level = -1

Connect update array node to node after



DELETE(30)

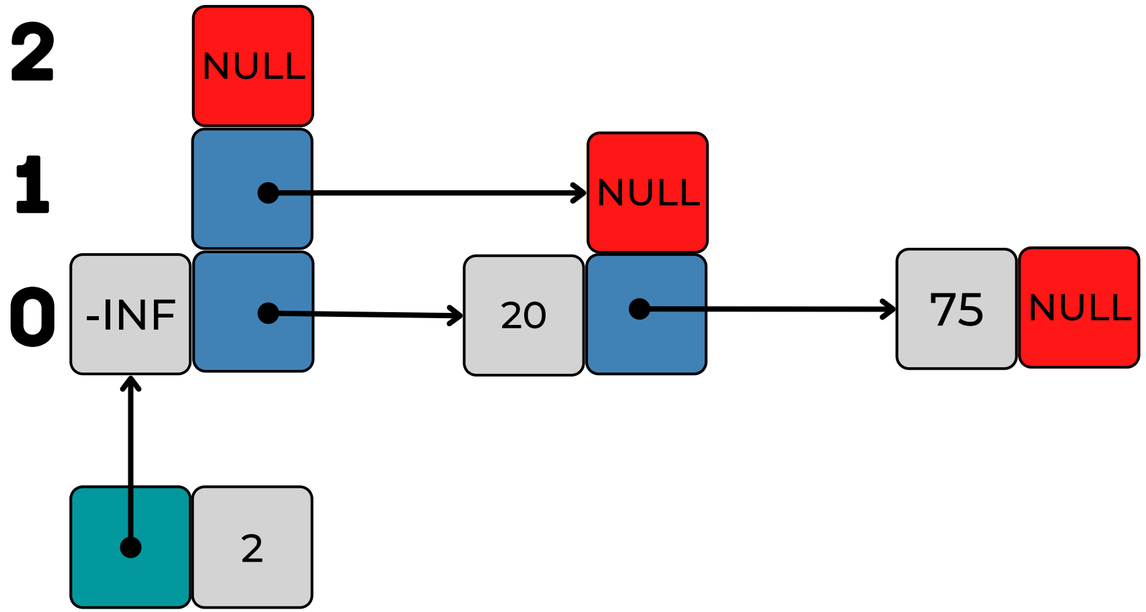
TEAM JC

Search . . .

DELETION

Level = -1

node 30 is now deleted



DELETE(30)

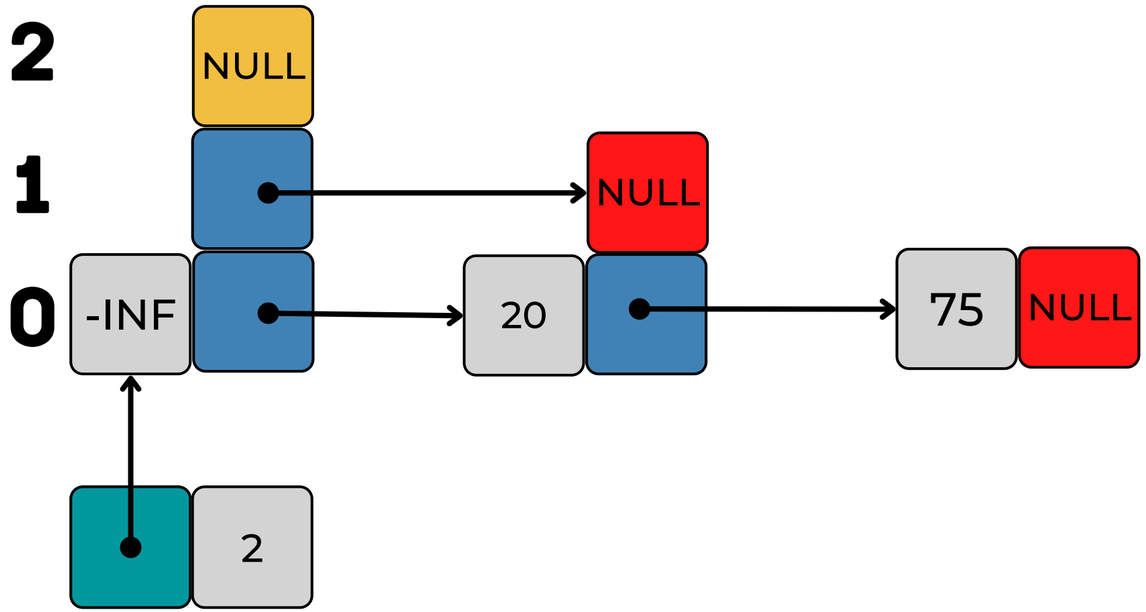
TEAM JC

Search . . .

DELETION

Level = -1

Is head link (lvl 2) null?



DELETE(30)

TEAM JC

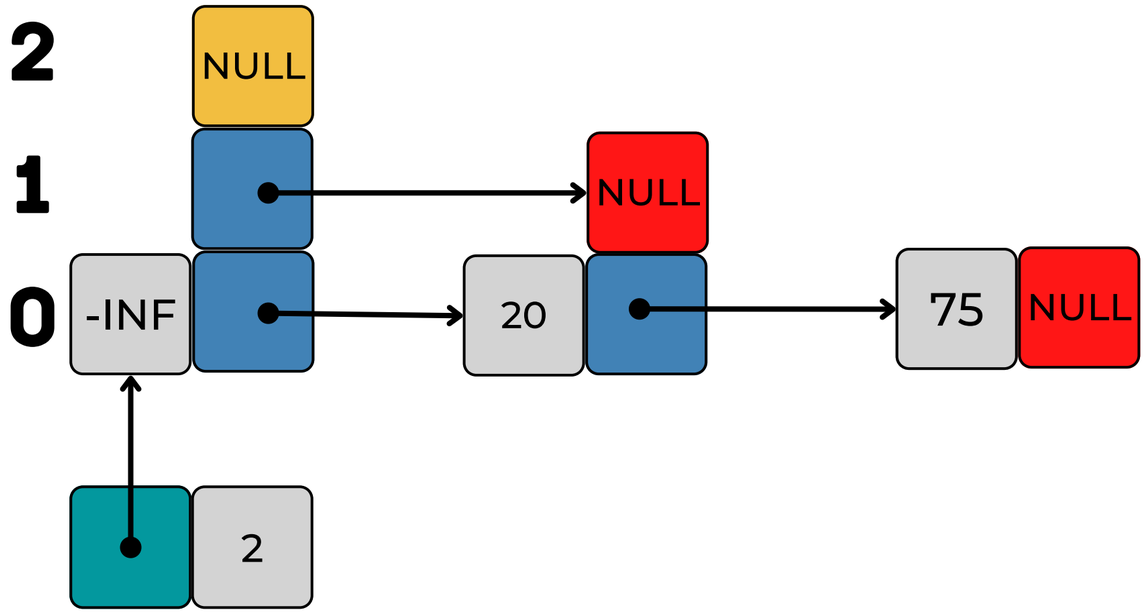
Search . . .

DELETION

Level = -1

Is head link (lvl 2) null?

True = Subtract level count



DELETE(30)

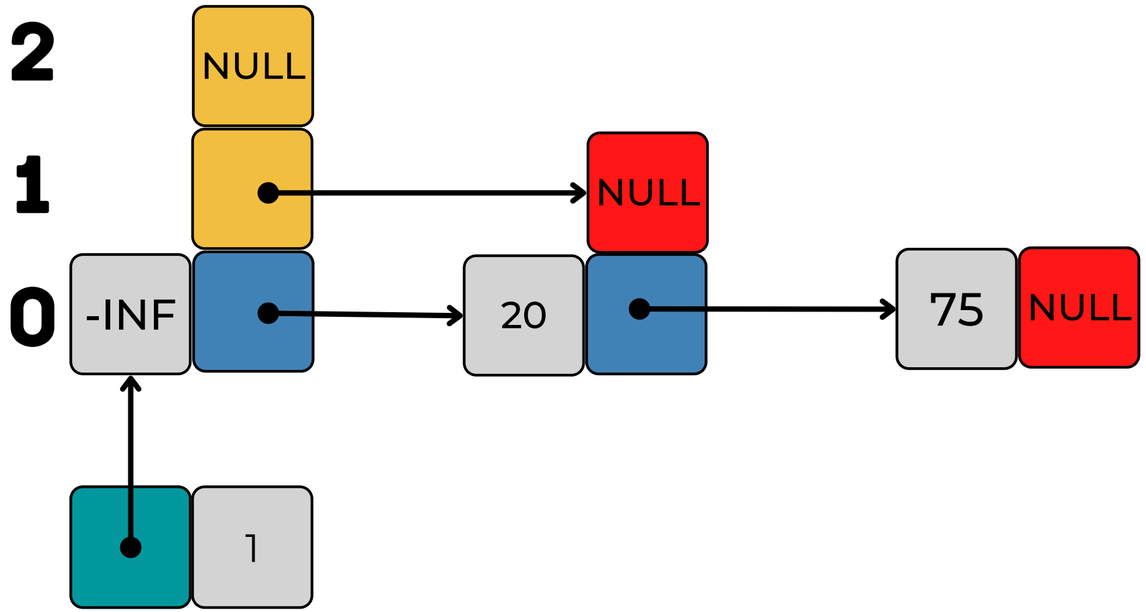
TEAM JC

Search . . .

DELETION

Level = -1

Is head link (lvl 1) null?



DELETE(30)

TEAM JC

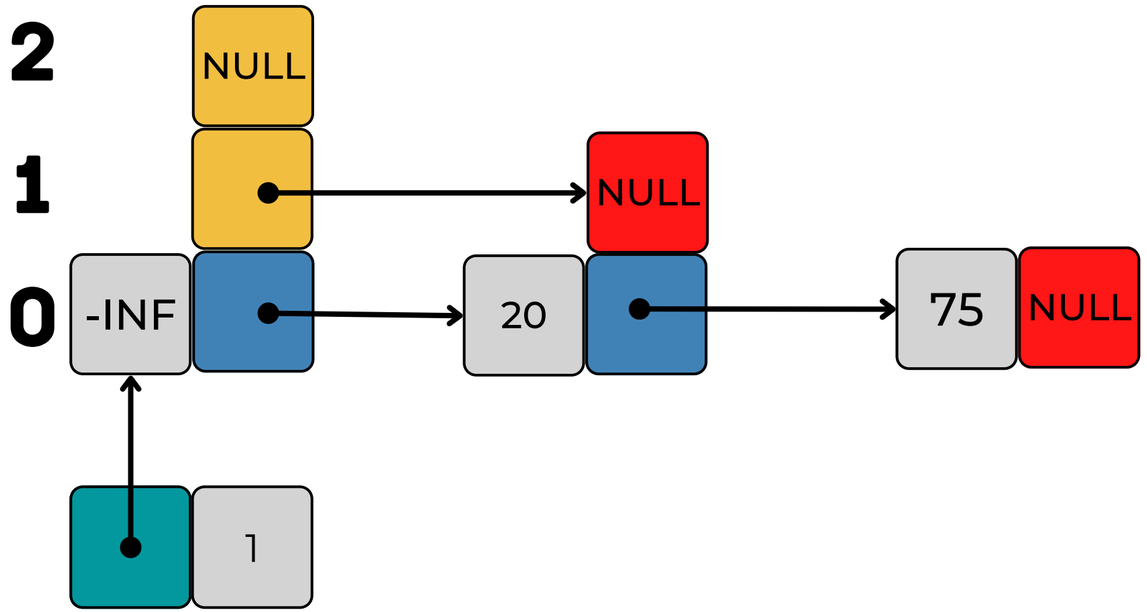
Search . . .

DELETION

Level = -1

Is head link (lvl 1) null?

False = **End traversal**



DELETE(30)

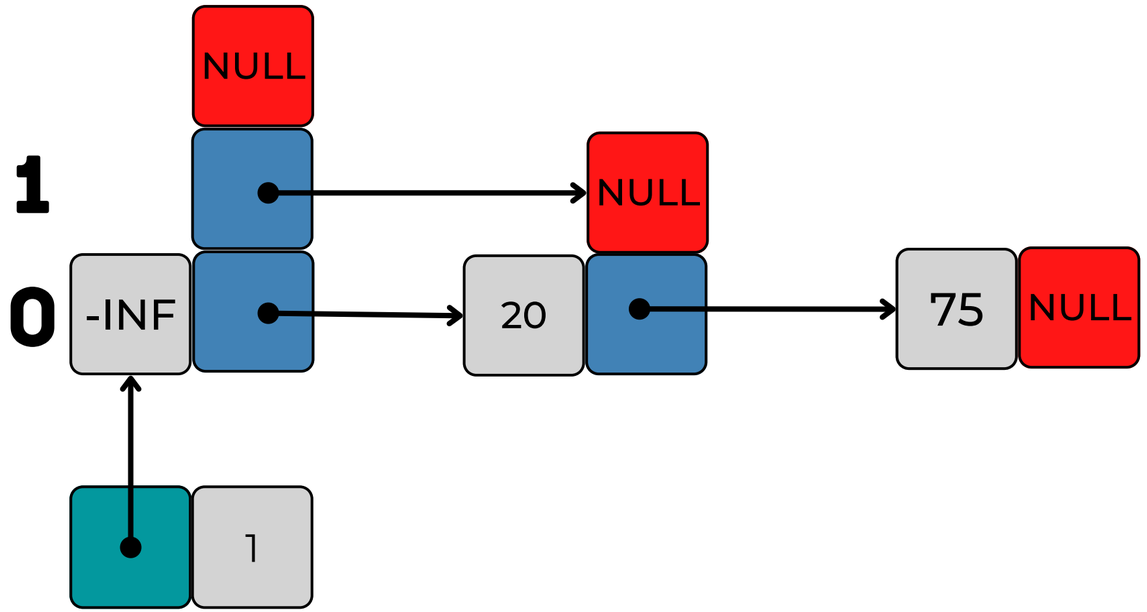
TEAM JC

Search . . .

DELETION

Level = 1

node 30 is now deleted



DELETE(30)

CODE COMPARISON

INTERNET

STRUCTURE

```
class Node
{
public:
    int key;

    Node **forward;
    Node(int, int);
};

class SkipList
{
    int MAXLVL;

    float P;

    int level;

    Node *header;
public:
    SkipList(int, float);
    int randomLevel();
    Node* createNode(int, int);
    void insertElement(int);
    void displayList();
};
```

STREAMLINED

```
#define MAXLEVEL 3

typedef struct node{
  struct node **link;
  int data;
}*Node;

typedef struct {
  Node head;
  int level;
}SkipList;
```

HTTPS://WWW.GEEKSFORGEEKS.ORG/SKIP-LIST/?REF=LBP

INTERNET

INITIALIZATION

```
SkipList::SkipList(int MAXLVL, float P)
{
    this->MAXLVL = MAXLVL;
    this->P = P;
    level = 0;

    // create header node and initialize key to -1
    header = new Node(-1, MAXLVL);
};

Node::Node(int key, int level)
{
    this->key = key;
    // Allocate memory to forward
    forward = new Node*[level+1];
    // Fill forward array with 0(NULL)
    memset(forward, 0, sizeof(Node*)*(level+1));
};
```

STREAMLINED

```
void initList(SkipList *list)
{
  list->level = 0;

  // create head
  list->head = (struct node*)calloc(sizeof(struct
node),1);
  // initialize data to INT_MIN
  list->head->data = -2147483647;
  // initalize pointer of arrays
  list->head->link = (struct
node**)calloc(sizeof(struct node*),MAXLEVEL+1);
}
```

HTTPS://WWW.GEEKSFORGEEKS.ORG/SKIP-LIST/?REF=LBP

INTERNET

INSERTION

```
void SkipList::insertElement(int key)
{
    Node *current = header;
    // create update array and initialize it
    Node *update[MAXLVL+1];
    memset(update, 0, sizeof(Node*)*(MAXLVL+1));
    for (int i = level; i >= 0; i--){
        while (current->forward[i] != NULL &&
              current->forward[i]->key < key)
            current = current->forward[i];
        update[i] = current;
    }
    current = current->forward[0];
    if (current == NULL || current->key != key)
    {
        // Generate a random level for node
        int rlevel = randomLevel();
        if (rlevel > level){
            for (int i=level+1;i<rlevel+1;i++)
                update[i] = header;
            level = rlevel;
        }
        // create new node with random level generated
        Node* n = createNode(key, rlevel);
        // insert node by rearranging pointers
        for (int i=0;i<=rlevel;i++){
            n->forward[i] = update[i]->forward[i];
            update[i]->forward[i] = n;
        }
    }
};
```

HTTPS://WWW.GEEKSFORGEEKS.ORG/SKIP-LIST/?REF=LBP

STREAMLINED

```
void insertData(SkipList *list, int data)
{
    // initialize update array
    Node *update = (struct node**)calloc(sizeof(struct
node*),MAXLEVEL+1);
    Node node = updateInsert(list->head, data, update,
list->level);
    if(node == NULL || node->data != data) {
        // Generate random level
        int nlevel = coinFlipLevel();
        // Create new node
        Node newNode = (struct node*)calloc(sizeof(struct
node),1);
        newNode->data = data;
        newNode->link = (struct
node**)calloc(sizeof(struct node*),nlevel+1);
        int i;
        for(i = list->level+1; i <= nlevel; i++) {
            update[i] = list->head;
            list->level = i;
        }
        // insert node by rearranging pointers
        for(i = 0; i <= nlevel; i++) {
            newNode->link[i] = update[i]->link[i];
            update[i]->link[i] = newNode;
        }
    }
    free(update);
}
```

INTERNET

INSERTION

```
void SkipList::insertElement(int key)
{
    Node *current = header;
    // create update array and initialize it
    Node *update[MAXLVL+1];
    memset(update, 0, sizeof(Node*)*(MAXLVL+1));
    for (int i = level; i >= 0; i--){
        while (current->forward[i] != NULL &&
              current->forward[i]->key < key)
            current = current->forward[i];
        update[i] = current;
    }
    current = current->forward[0];
    if (current == NULL || current->key != key)
    {
        // Generate a random level for node
        int rlevel = randomLevel();
        if (rlevel > level){
            for (int i=level+1;i<rlevel+1;i++)
                update[i] = header;
            level = rlevel;
        }
        // create new node with random level generated
        Node* n = createNode(key, rlevel);
        // insert node by rearranging pointers
        for (int i=0;i<=rlevel;i++){
            n->forward[i] = update[i]->forward[i];
            update[i]->forward[i] = n;
        }
    }
};
```

HTTPS://WWW.GEEKSFORGEEKS.ORG/SKIP-LIST/?REF=LBP

STREAMLINED

```
Node updateInsert(Node head, int data, Node
*update, int level)
{
  if(head == NULL || head->data >= data ) {
    return head;
  }

  update[level] = head;

  if (head->link[level] != NULL && data >= head-
>link[level]->data || level == 0) {
    updateInsert(head->link[level], data, update,
level);
  } else {
    updateInsert(head, data, update, level-1);
  }
}
```

INTERNET

INSERTION

```
int SkipList::randomLevel()
{
    float r = (float)rand()/RAND_MAX;
    int lvl = 0;
    while (r < P && lvl < MAXLVL)
    {
        lvl++;
        r = (float)rand()/RAND_MAX;
    }
    return lvl;
};
```

HTTPS://WWW.GEEKSFORGEEKS.ORG/SKIP-LIST/?REF=LBP

STREAMLINED

```
int coinFlipLevel()
{
  int newLevel, coin;
  for(newLevel = 0, coin = rand() % 2; coin &&
newLevel < MAXLEVEL; newLevel++, coin = rand() % 2)
{}
  return newLevel;
}
```

INTERNET

DELETION

```
void SkipList::deleteElement(int key)
{
    Node *current = header;
    Node *update[MAXLVL+1];
    memset(update, 0, sizeof(Node*)*(MAXLVL+1));
    for(int i = level; i >= 0; i--) {
        while(current->forward[i] != NULL  &&
              current->forward[i]->key < key)
            current = current->forward[i];
        update[i] = current;
    }
    current = current->forward[0];
    if(current != NULL and current->key == key)
    {
        for(int i=0;i<=level;i++)
        {
            if(update[i]->forward[i] != current)
                break;
                  update[i]->forward[i]  =  current-
>forward[i];
        }

        // Remove levels having no elements
        while(level>0 &&
              header->forward[level] == 0)
            level--;
    }
};
```

HTTPS://WWW.GEEKSFORGEEKS.ORG/SKIP-LIST/?REF=LBP

STREAMLINED

```
void deleteData(SkipList *list, int data) {
  Node *update = (struct
node**)calloc(sizeof(struct node*),MAXLEVEL+1);

  Node node = updateDelete(list->head, data,
update, list->level);

  if(node != NULL && node->data == data) {
    int i;
    for(i = 0; i <= list->level && update[i]-
>link[i] == node; i++) {
      update[i]->link[i] = node->link[i];
    }
    for(; list->level > 0 && list->head->link[list-
>level] == NULL; list->level--) {}
    free(node);
  }

  free(update);
}
```

INTERNET

DELETION

```
void SkipList::deleteElement(int key)
{
    Node *current = header;
    Node *update[MAXLVL+1];
    memset(update, 0, sizeof(Node*)*(MAXLVL+1));
    for(int i = level; i >= 0; i--) {
        while(current->forward[i] != NULL  &&
              current->forward[i]->key < key)
            current = current->forward[i];
        update[i] = current;
    }
    current = current->forward[0];
    if(current != NULL and current->key == key)
    {
        for(int i=0;i<=level;i++)
        {
            if(update[i]->forward[i] != current)
                break;
                  update[i]->forward[i]  =  current-
>forward[i];
        }

        // Remove levels having no elements
        while(level>0 &&
              header->forward[level] == 0)
            level--;
    }
};
```

HTTPS://WWW.GEEKSFORGEEKS.ORG/SKIP-LIST/?REF=LBP

STREAMLINED

```
Node updateDelete(Node head, int data, Node
*update, int level) {
  if(head == NULL || head->data >= data){
    return head;
  }

  update[level] = head;

  if (head->link[level] != NULL && data > head-
>link[level]->data || level == 0) {
    updateDelete(head->link[level], data, update,
level);
  } else {
    updateDelete(head, data, update, level-1);
  }
}
```

INTERNET

SEARCH

```
void SkipList::searchElement(int key)
{
    Node *current = header;

    for(int i = level; i >= 0; i--)
    {
        while(current->forward[i] &&
                current->forward[i]->key < key)
            current = current->forward[i];
    }
    current = current->forward[0];

    if(current and current->key == key)
        cout<<"Found key: "<<key<<"\n";
};
```

HTTPS://WWW.GEEKSFORGEEKS.ORG/SKIP-LIST/?REF=LBP

STREAMLINED

```
Node searchData(Node head, int level, int data) {
  if(head == NULL || head->data >= data ){
    return head;
  }

  if (head->link[level] != NULL && data >= head-
>link[level]->data || level == 0) {
    searchData(head->link[level], level, data);
  } else {
    searchData(head, level-1, data);
  }
}
```

THANK YOU

END SLIDE