Skip Lists

Sergios Anestis Kefalidis
Konstantinos Nikoletos
Kostas Plas &
Manolis Koubarakis

What is a Skip List?

A Skip List is a tree-like efficient **probabilistic** data structure that provides O(log(n)) average case for insertion, search and deletion.

Skip Lists are in their simplest iteration a sorted list, with many levels. Each level is sparser than the one below it and can be used to “skip” nodes when searching for elements.

Why do we use Skip Lists?

It’s an effort to get the best of both worlds! Sorted Arrays and Sorted Lists!

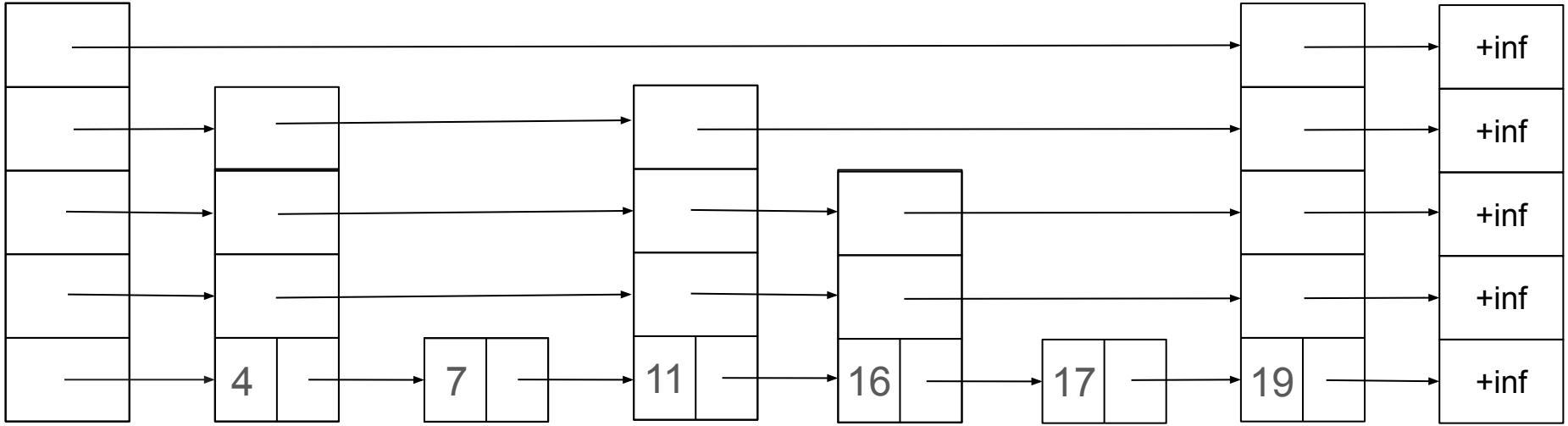
- Sorted Arrays:
  - We can search in sorted arrays in O(log(n)) using binary search
  - Insertion in worst case is O(n)
- Sorted Lists:
  - Insertion is O(n), but update is only O(1)
  - Worst case search in sorted lists is O(n)

By combining the properties of these two data structures we can get a faster structure that guarantees with high probability the average case complexity for the above operations.

How does a Skip List function?

- The lowest level of a skip list is always an order linked list
- Levels above the lower level can “skip” elements
- How do we decide the level and which elements to skip?
  - We leave it up to chance! To be more precise, we leave it up to **coin flip**
  - We can think of skip lists node as having an array of “next” pointers, instead of a single next pointer
  - When we insert an element to the skip list, we toss a coin until we get tails. The node’s level will be equal to the number of heads we got in a row!
  - By connecting the random levels of each node, we create a tree like structure
- Why is it called skip list?
  - Because the number of levels is random. Elements might be missing from the higher levels (they are skipped).
  - We can use this property to efficiently insert and search for elements!

How does a Skip List function?

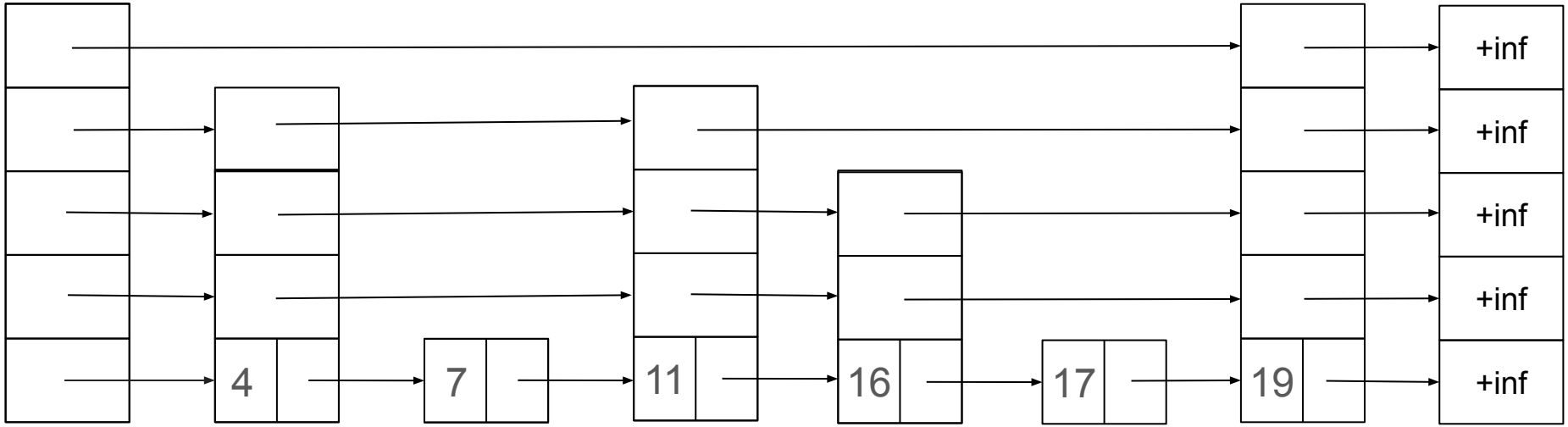


Insertion in Skip Lists (1)

- To insert an element in the skip list we first have to find where this element belongs.
  - We do that by searching each level, starting from the top. When the next node contains a key larger than our element we drop a level
  - In the lowest level, we basically search in a simple ordered linked list
- When we find the correct position to insert the element, we create a new node and fix its level.
  - We toss a coin till we get tails. The number of consecutive heads is the node's level
  - **Be careful:** We have to remember the previous nodes from each level in order to properly connect our new node to the list. We do that by maintaining an array of pointers
  - We connect our node with each level and we are finished!

Insertion in Skip Lists (2)

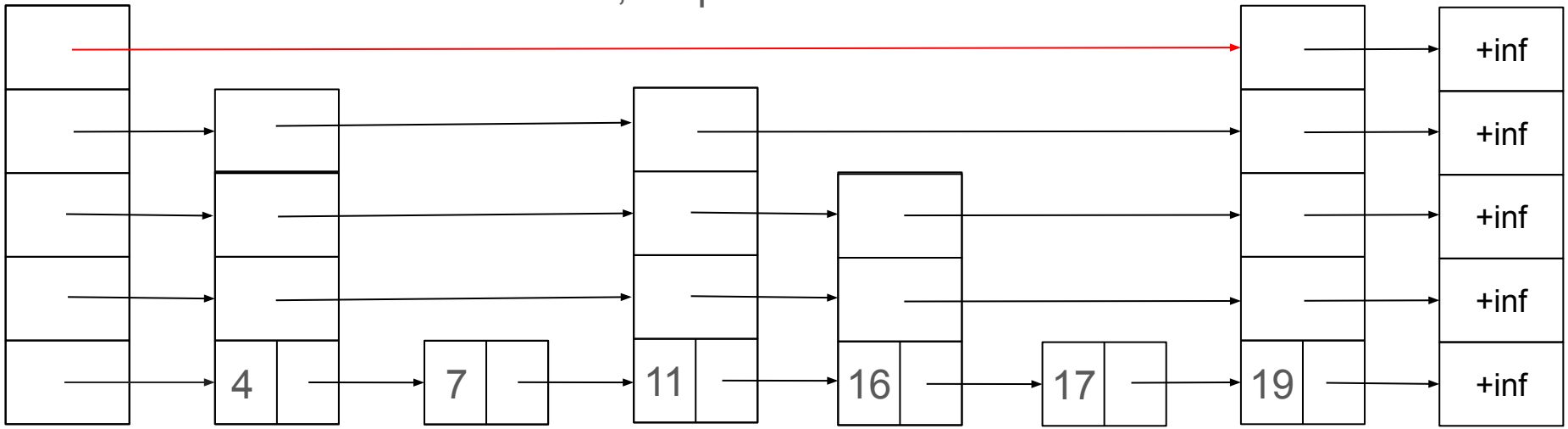
We want to insert 18



Insertion in Skip Lists (3)

We first search for 18's proper position

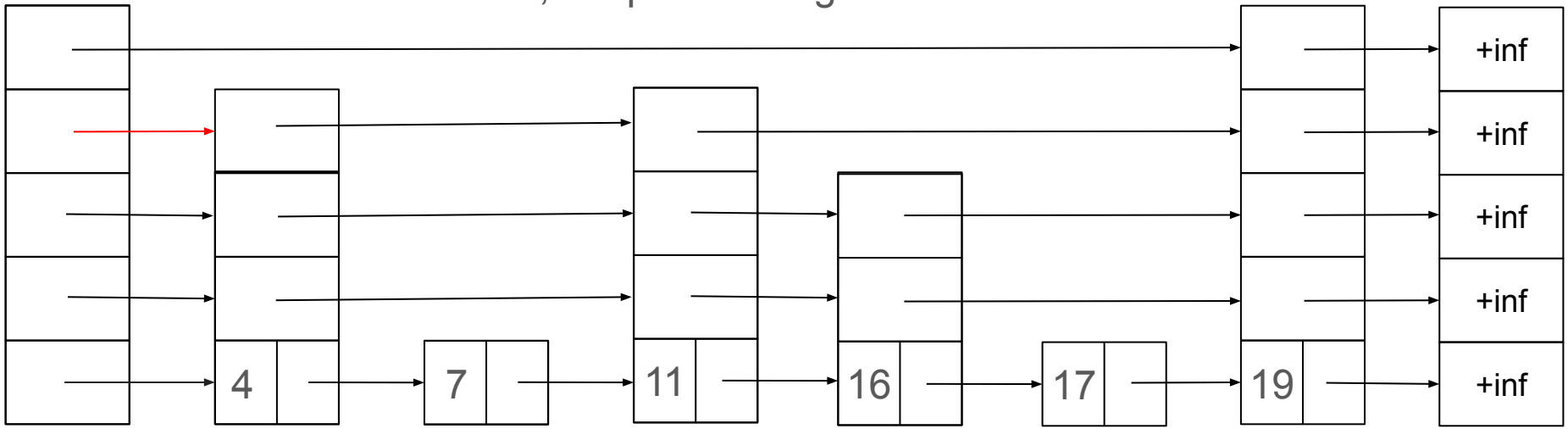
19>**18**, drop a level



Insertion in Skip Lists (4)

We first search for 18's proper position

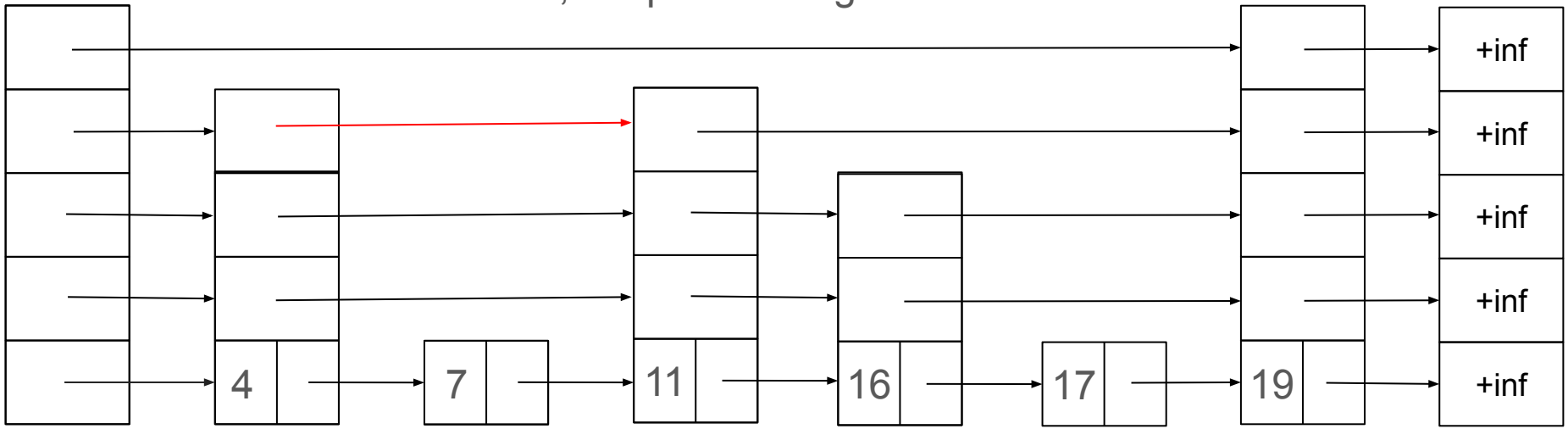
4<**18**, keep searching on this level



Insertion in Skip Lists (5)

We first search for 18's proper position

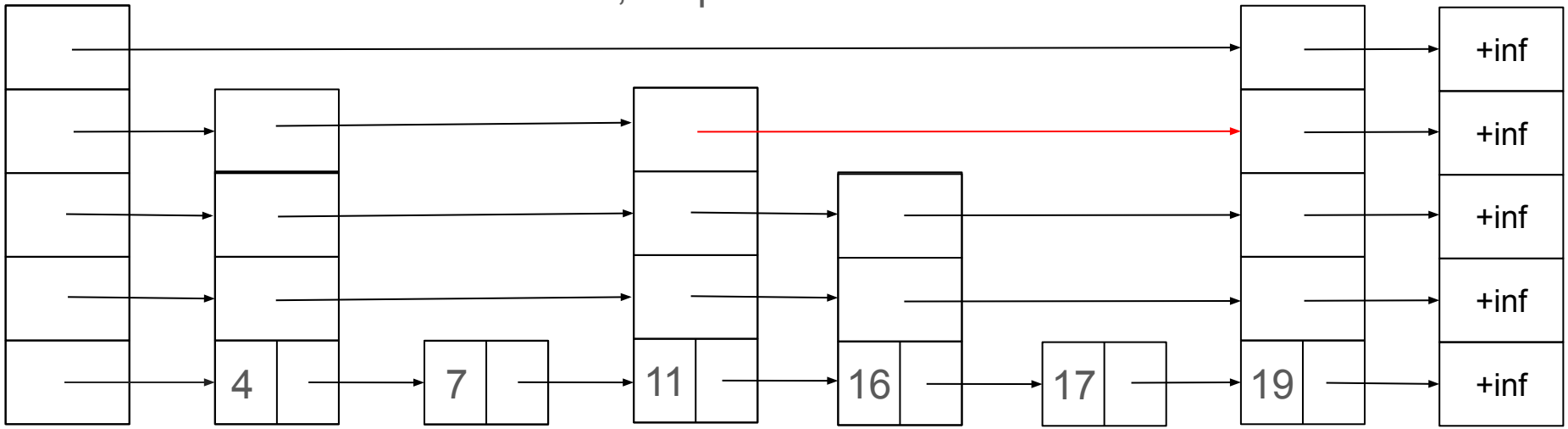
11<**18**, keep searching on this level



Insertion in Skip Lists (6)

We first search for 18's proper position

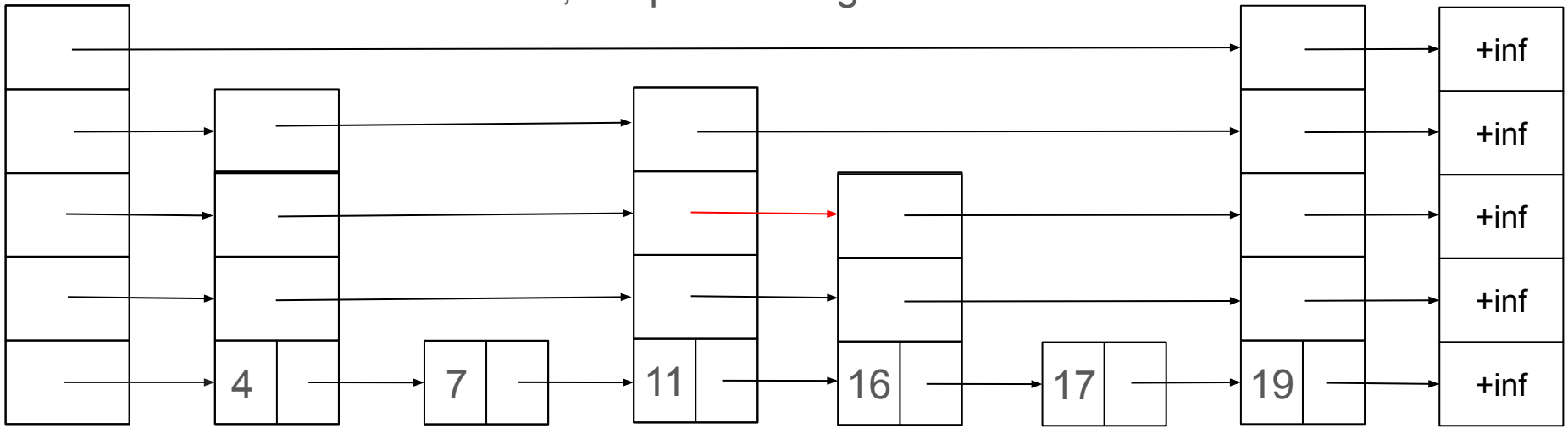
19>**18**, drop a level



Insertion in Skip Lists (7)

We first search for 18’s proper position

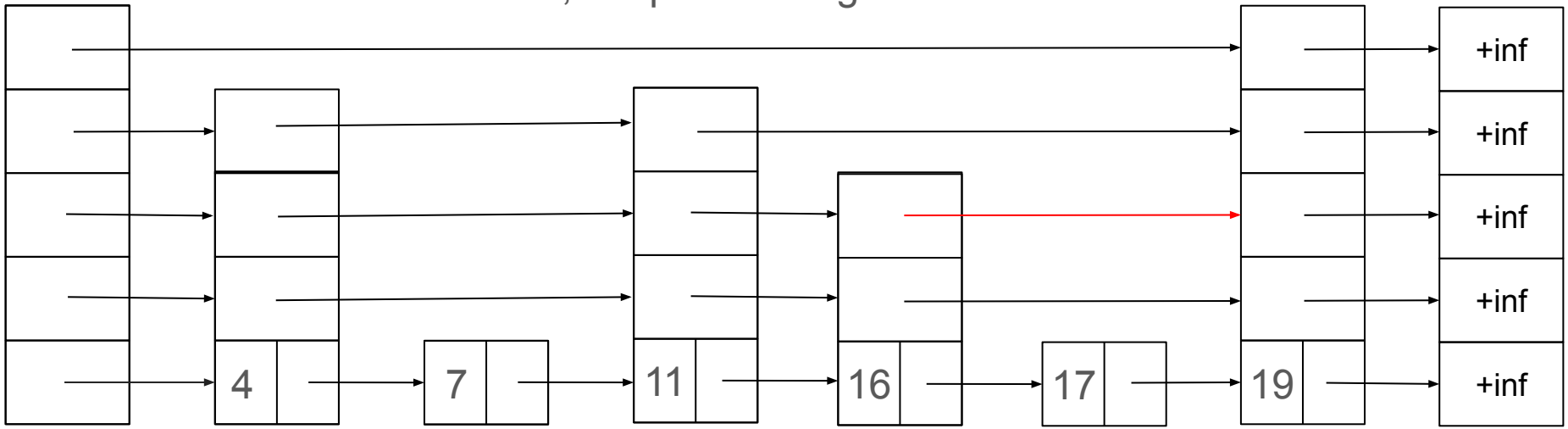
16<**18**, keep searching on this level



Insertion in Skip Lists (8)

We first search for 18's proper position

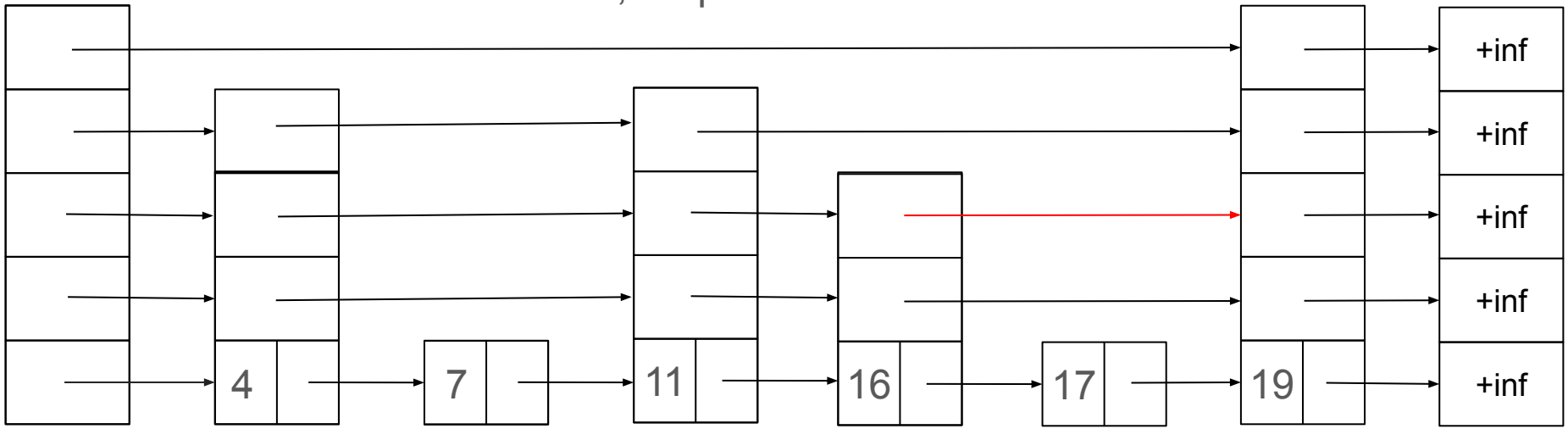
16<**18**, keep searching on this level



Insertion in Skip Lists (9)

We first search for 18's proper position

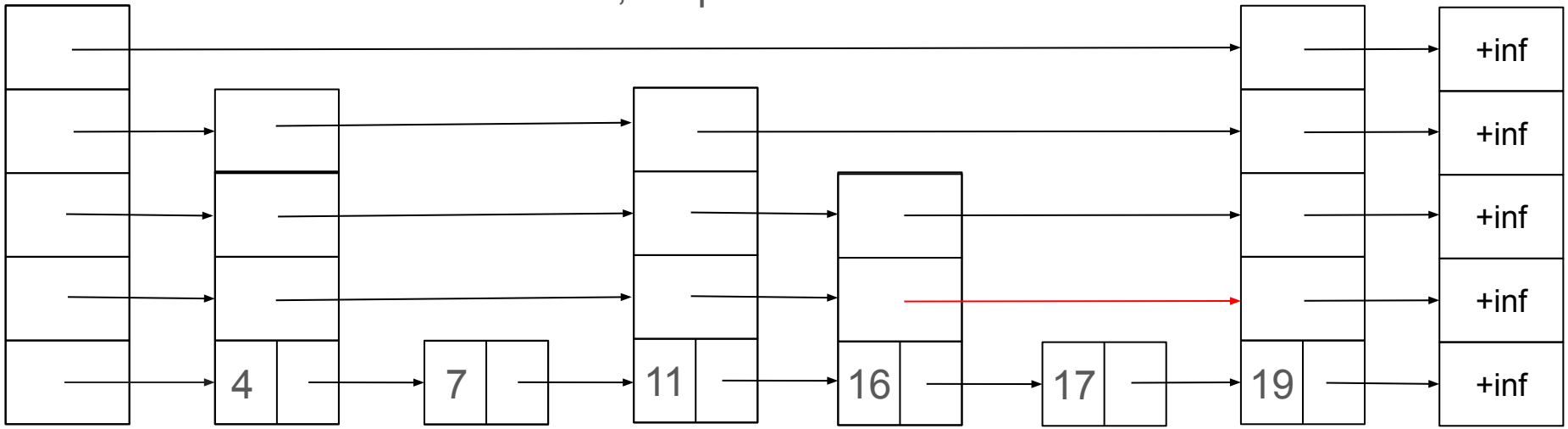
19>**18**, drop a level



Insertion in Skip Lists (10)

We first search for 18's proper position

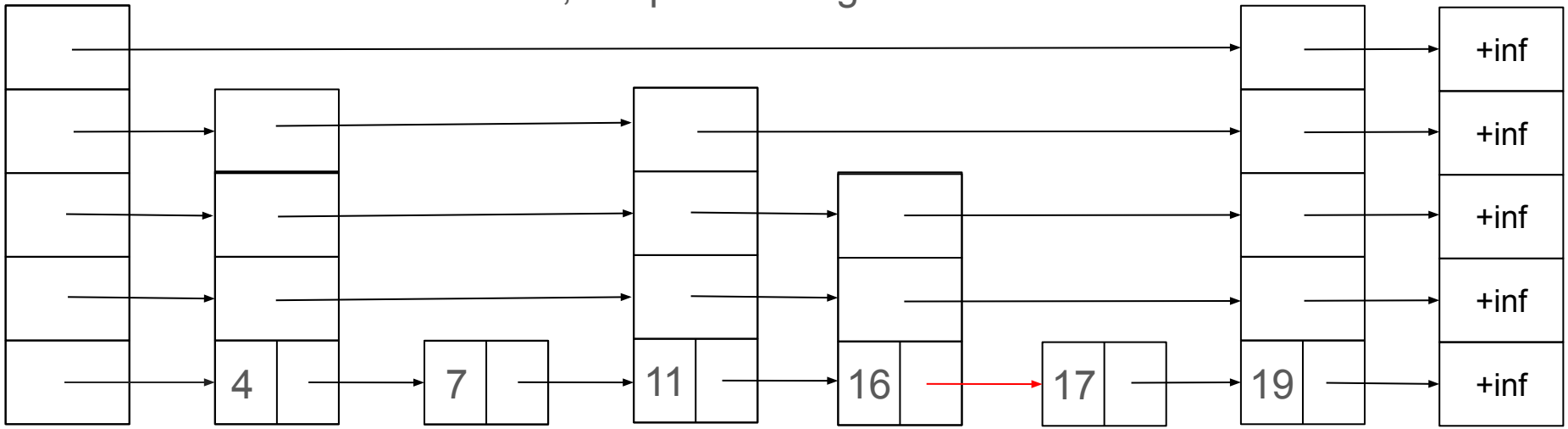
19>**18**, drop a level



Insertion in Skip Lists (11)

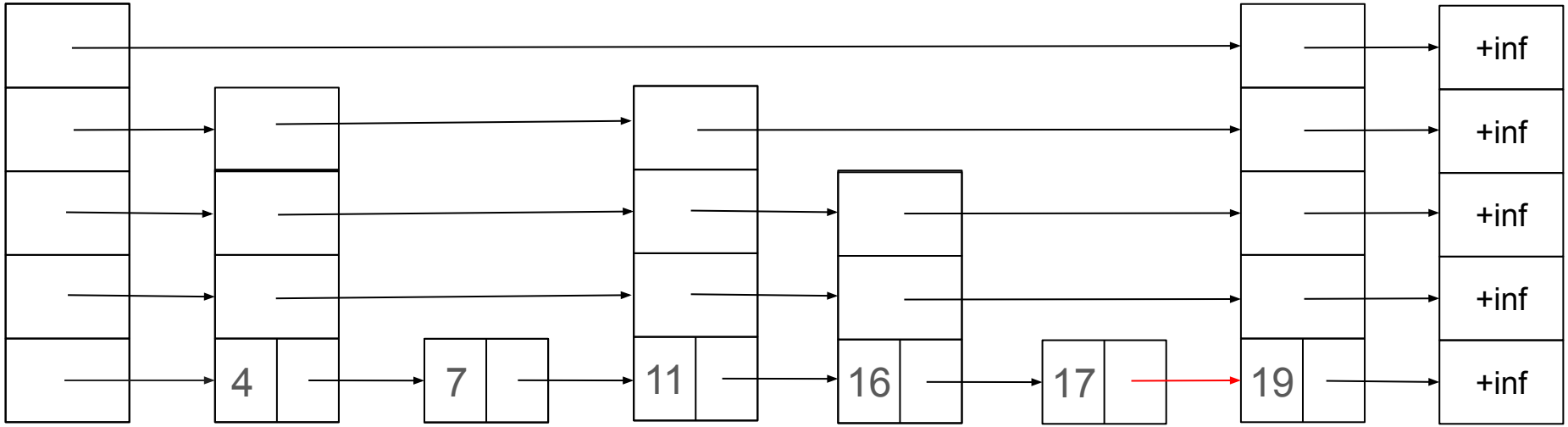
We first search for 18's proper position

17<**18**, keep searching on this level



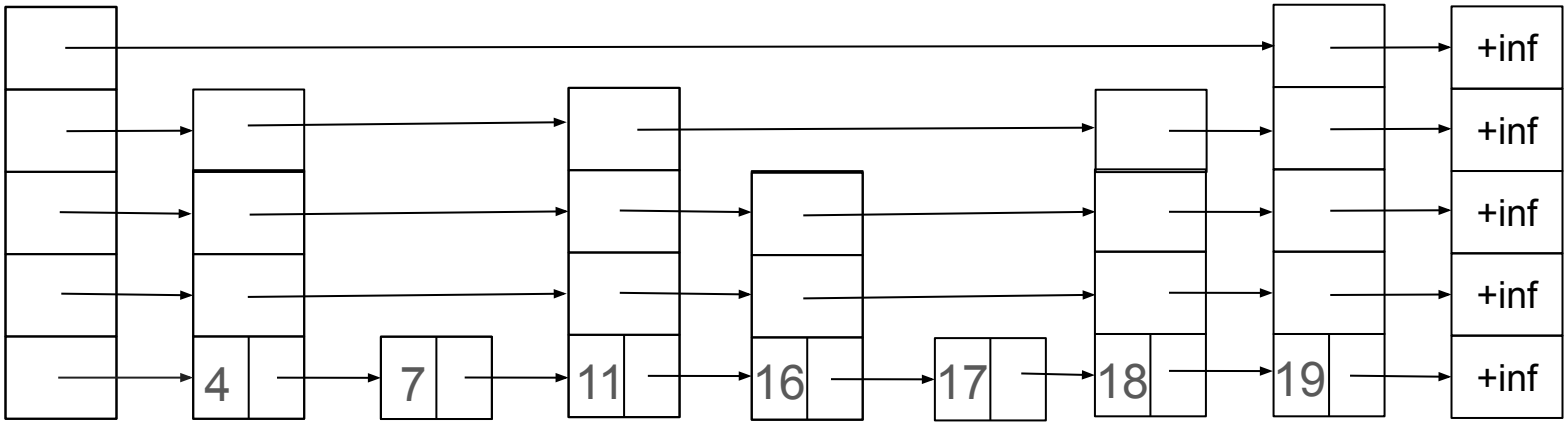
Insertion in Skip Lists (12)

We are in the lowest level. When we find an element larger than 18 we can stop searching. We insert 18 between 17 and 19.



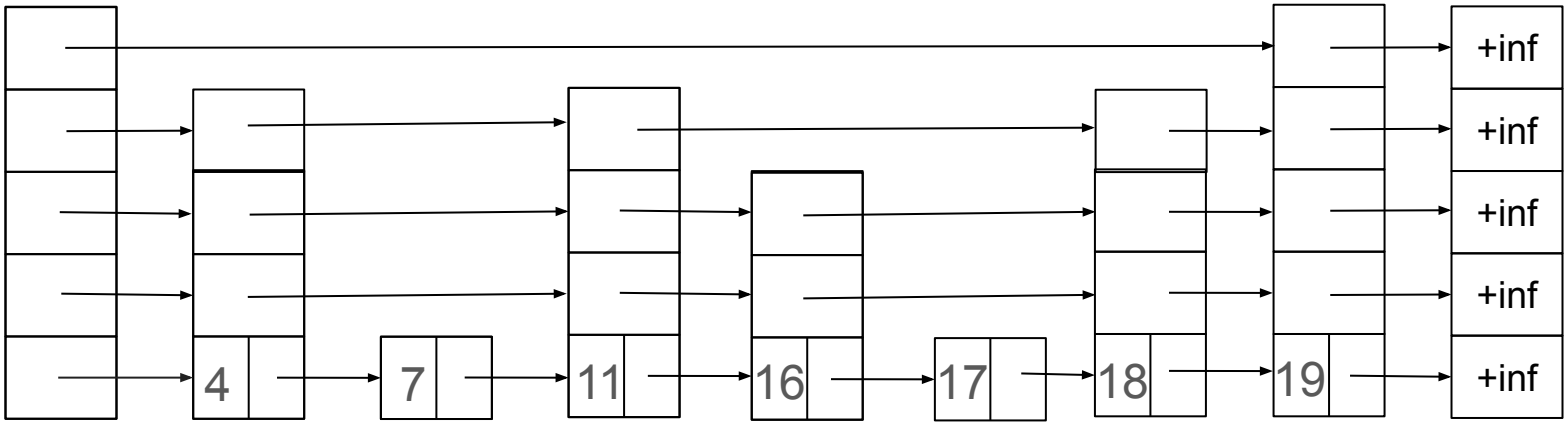
Insertion in Skip Lists (12)

We flip a coin to find 18's level! Let's say we got 3 heads before we got tails. Our node's level will be 1+3 = 4



Insertion in Skip Lists (13)

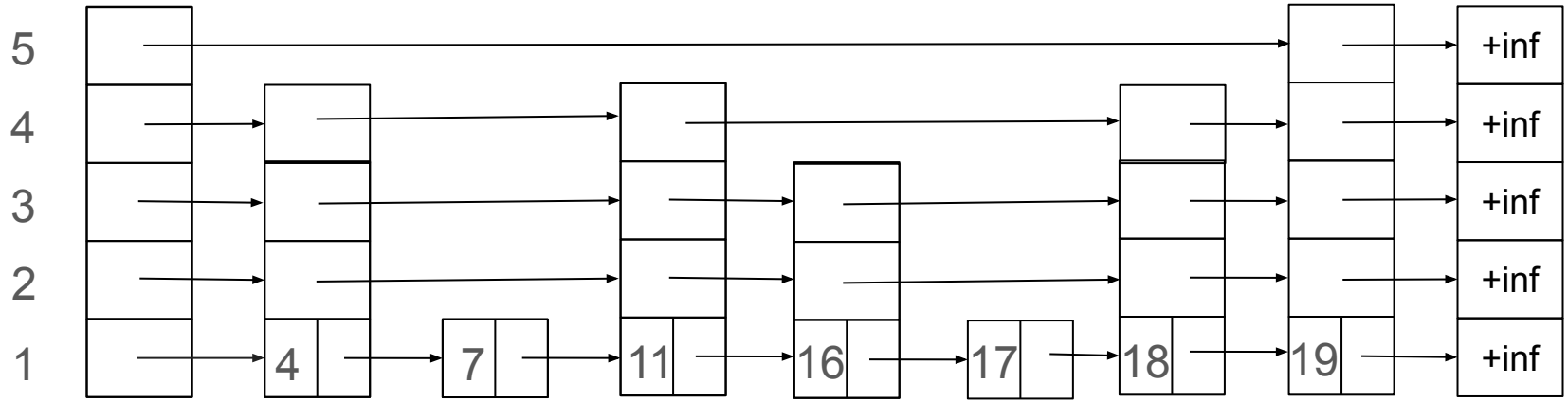
Remember we need to keep an array of pointers from previous levels so we can connect 18 with the rest of the list! We keep a “previous” pointer each time we drop down by a level



Insertion in Skip Lists (14)

So our previous array would look something like that:

1	2	3	4	5
17	16	16	11	S



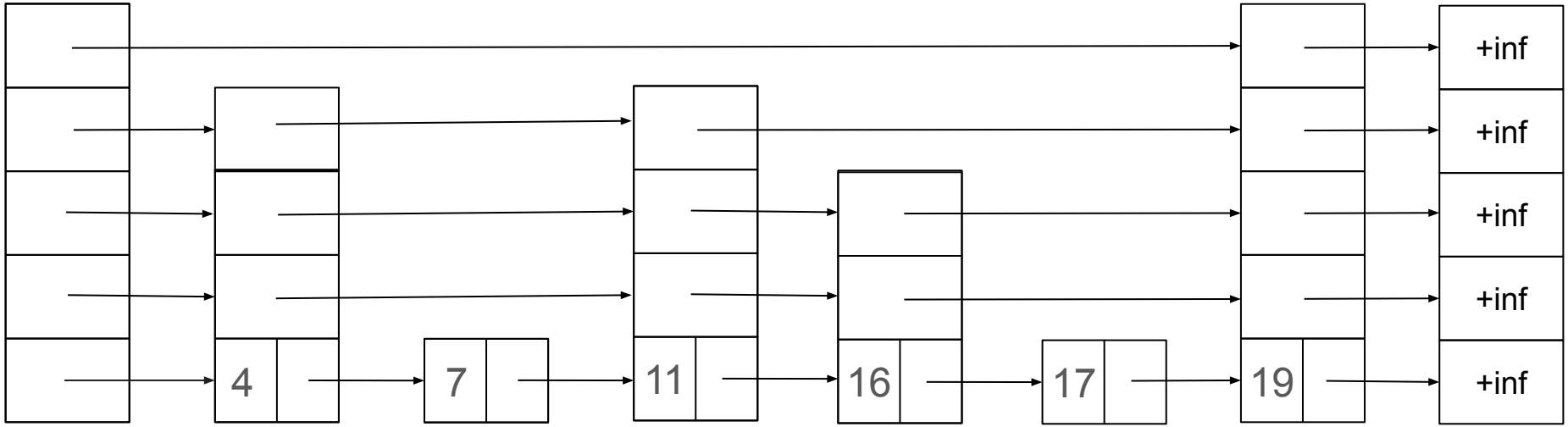
Searching in Skip Lists (1)

- Search in Skip Lists is almost the same as inserting elements!
  - We start from the highest level
  - Every time we find a value larger than the value we are searching for we drop a level

- When we reach the last level, we take an additional step to the next node
  - This node is either the value we are looking for or the value does not exist at all

Searching in Skip Lists (2)

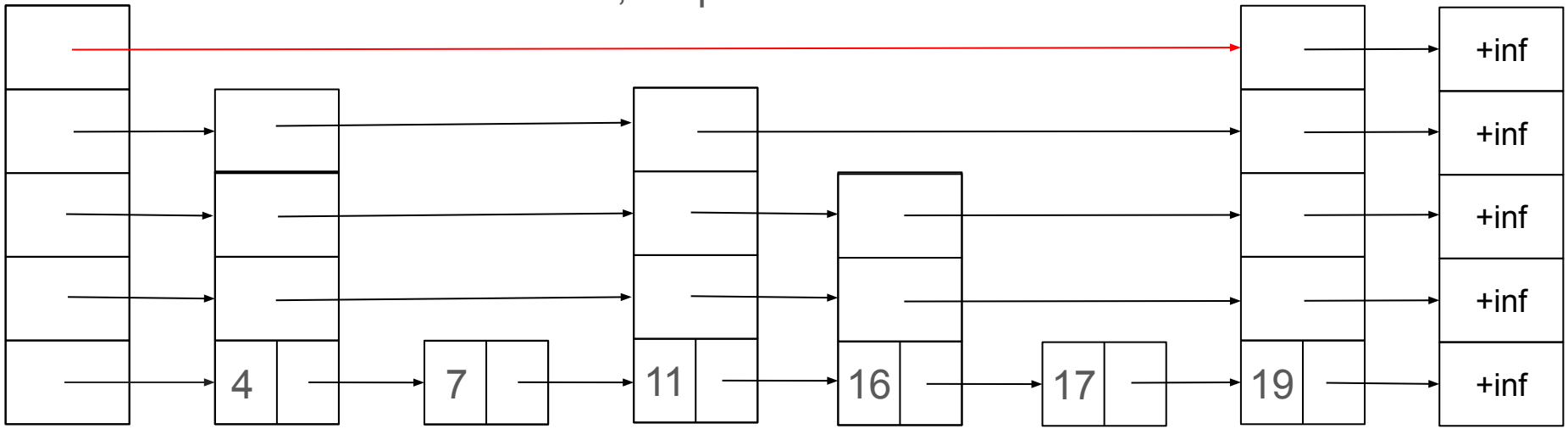
We want to find 17



Searching in Skip Lists (3)

We want to find 17

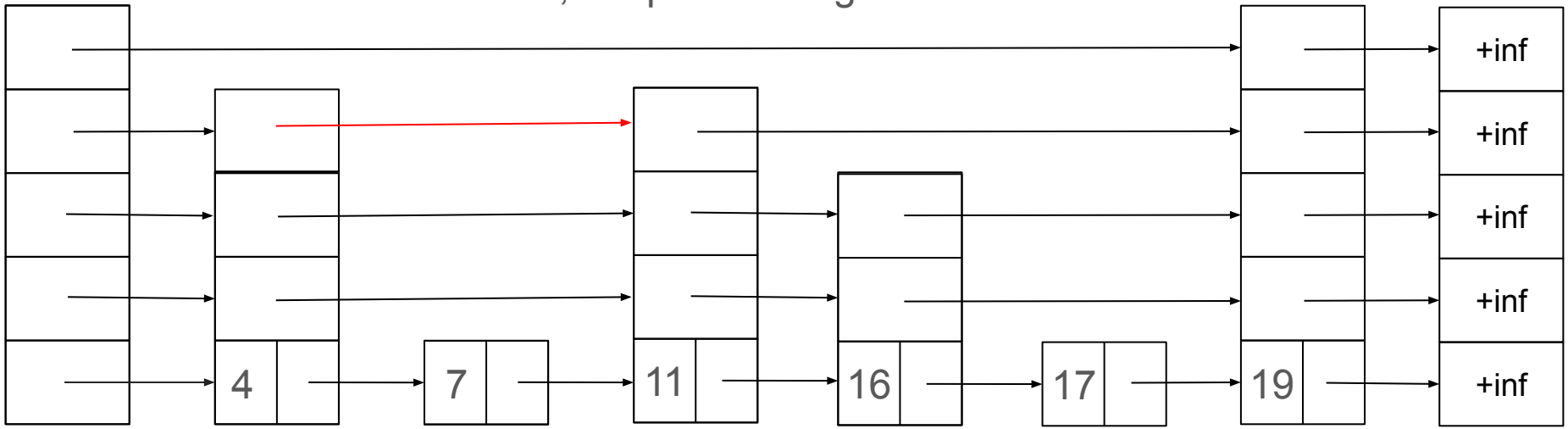
19>**17**, drop a level



Searching in Skip Lists (4)

We want to find 17

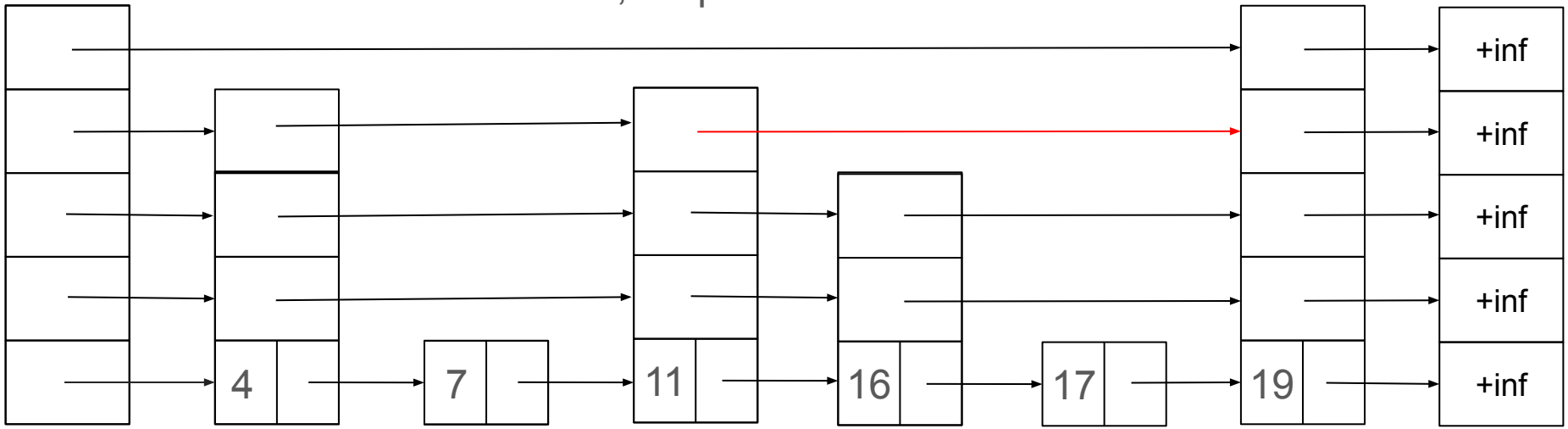
11<**17**, keep searching on this level



Searching in Skip Lists (5)

We want to find 17

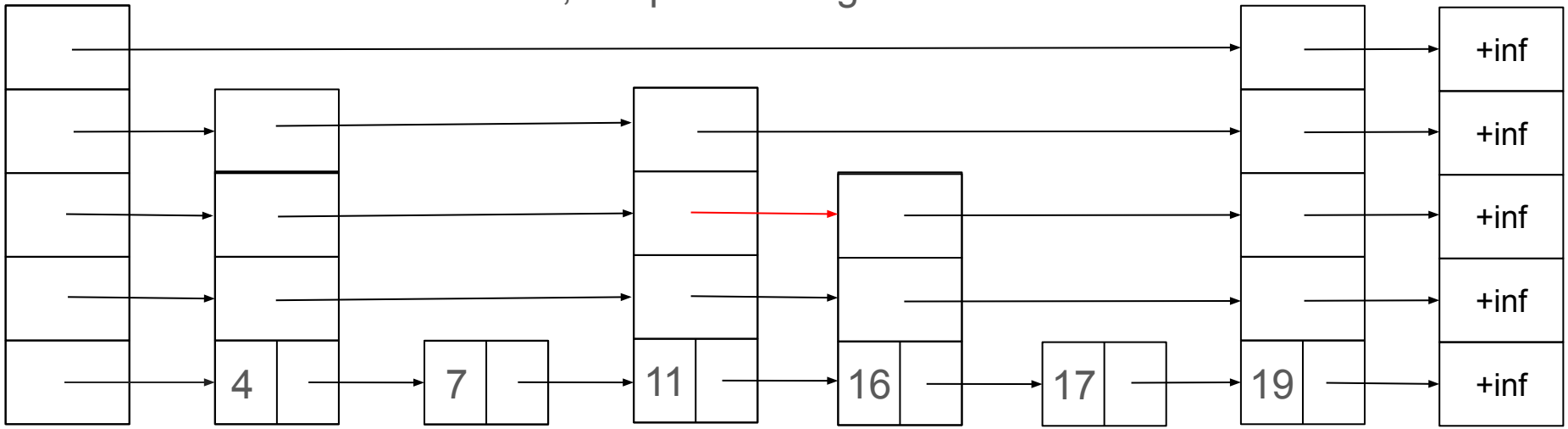
19>**17**, drop a level



Searching in Skip Lists (6)

We want to find 17

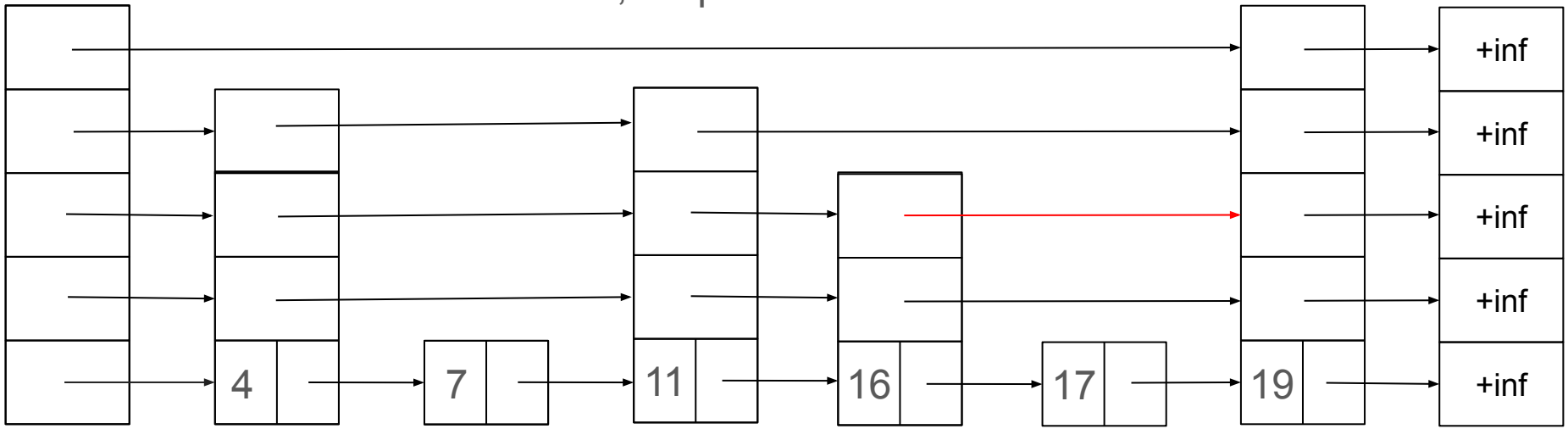
16<**17**, keep searching on this level



Searching in Skip Lists (7)

We want to find 17

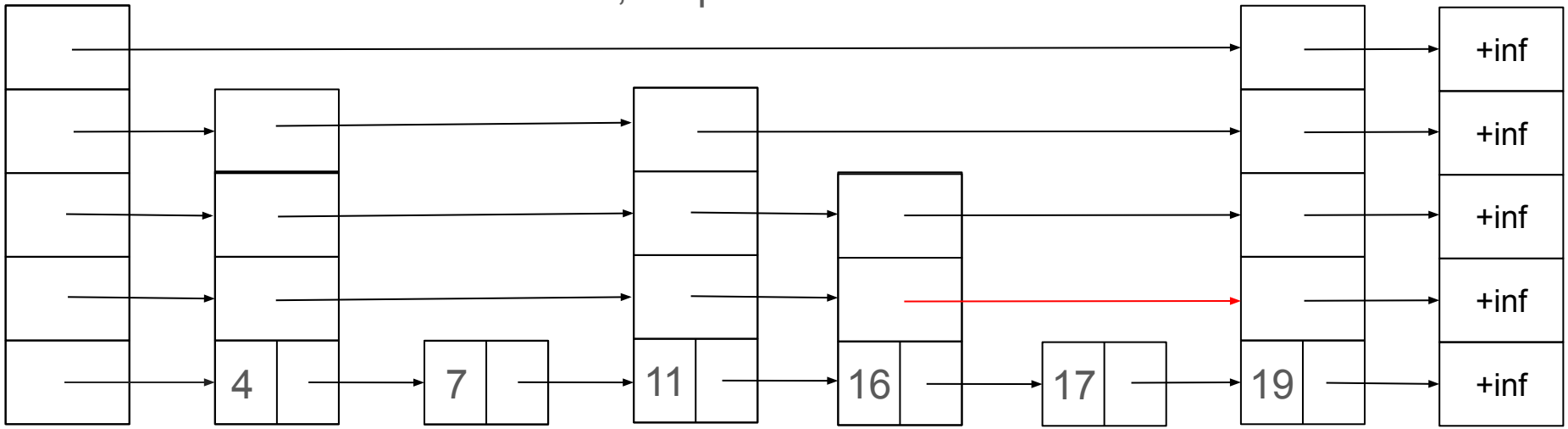
19>**17**, drop a level



Searching in Skip Lists (8)

We want to find 17

19>**17**, drop a level



Searching in Skip Lists (9)

We want to find 17. 17>17 and we are on the lowest level so we stop. The node next to 16 in the lowest level is 17, the search concludes successfully