Interval BST: BST Extension 03

Gianpiero Cabodi and Paolo Camurati
Dip. Automatica e Informatica
Politecnico di Torino

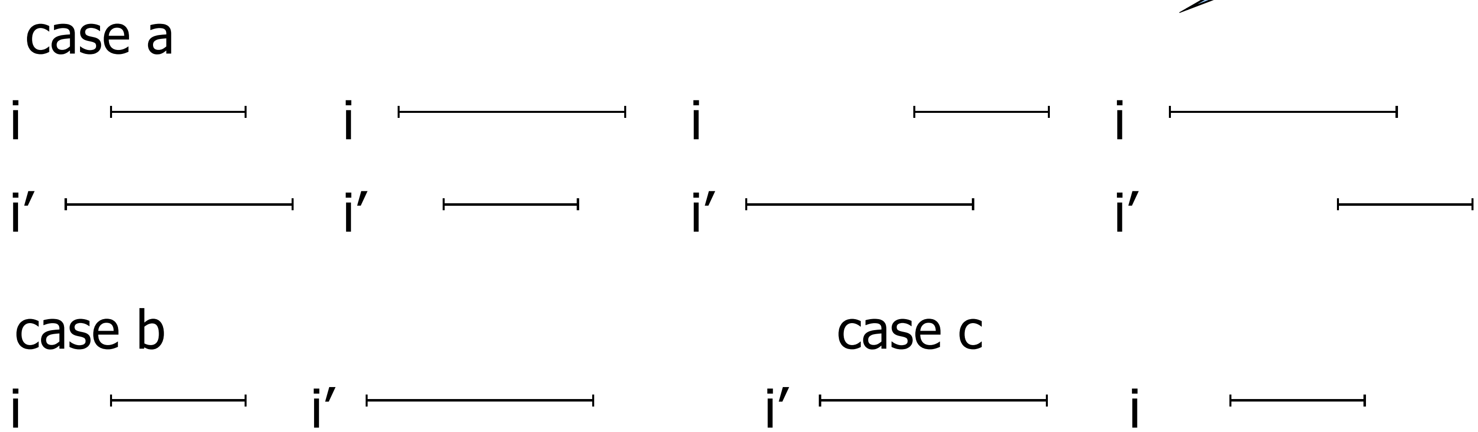
Interval BST

- BST used to store close intervals
  - Open and half-open intervals omit both or one of the endpoints from the set
  - Extending our result to those intervals would be straighforward
- A close interval is
  - An ordered real couple $[t_1, t_2]$, where $t_1 \leq t_2$ and $[t_1, t_2] = \{t \in \Re : t_1 \leq t \leq t_2\}$
  - The interval item $[t_1, t_2]$ can be realized with a `struct` with fields `low` $= t_1$ and `high` $= t_2$.

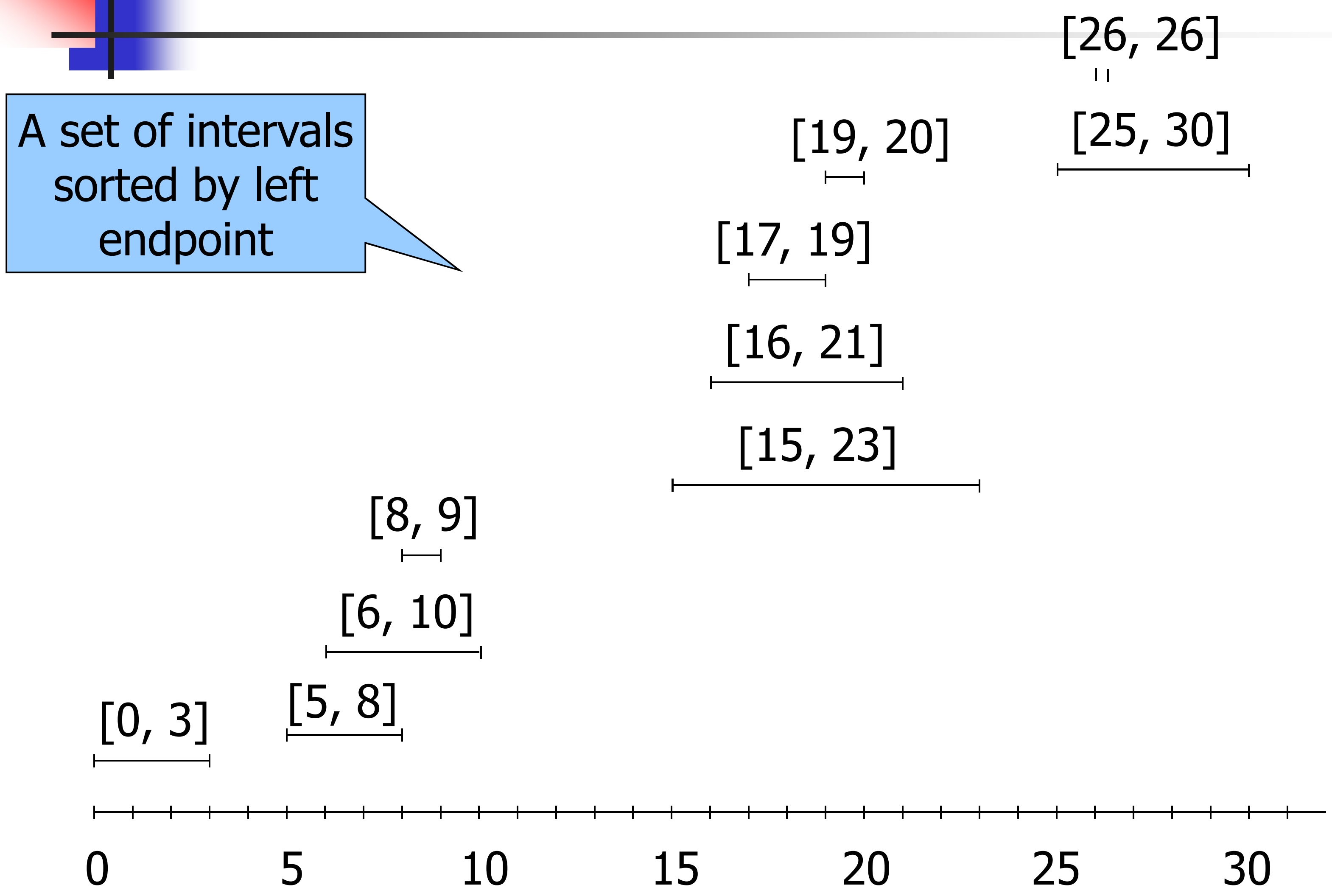
Interval BST

- Intervals i and i’ have intersection iff
  - `low[i] ≤ high[i’] && low[i’] ≤ high[i]`
- ∀ i, i’ the following conditions stand
  a. i and i’ have an intersection
  b. `high[i] ≤ low[i’]`
  c. `high[i’] ≤ low[i]`

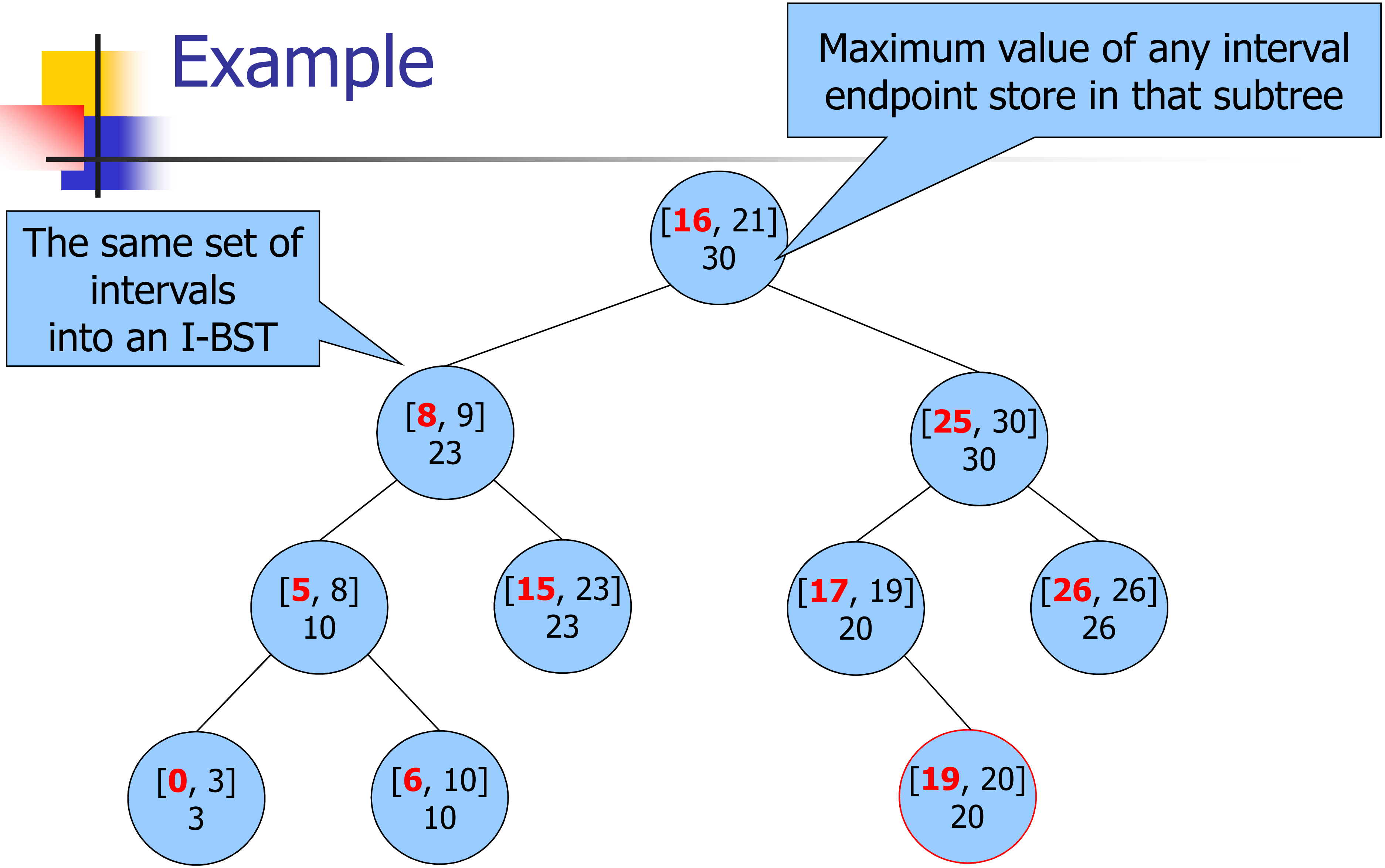
Interval trichotomy



Example


A set of intervals sorted by left endpoint
[26, 26]
[19, 20]
[25, 30]
[17, 19]
[16, 21]
[15, 23]
[8, 9]
[6, 10]
[5, 8]
[0, 3]
0
5
10
15
20
25
30

Example



Maximum value of any interval endpoint store in that subtree
The same set of intervals into an I-BST
[16, 21]
30
[8, 9]
23
[25, 30]
30
[5, 8]
10
[15, 23]
23
[17, 19]
20
[26, 26]
26
[0, 3]
3
[6, 10]
10
[19, 20]
20

Operations

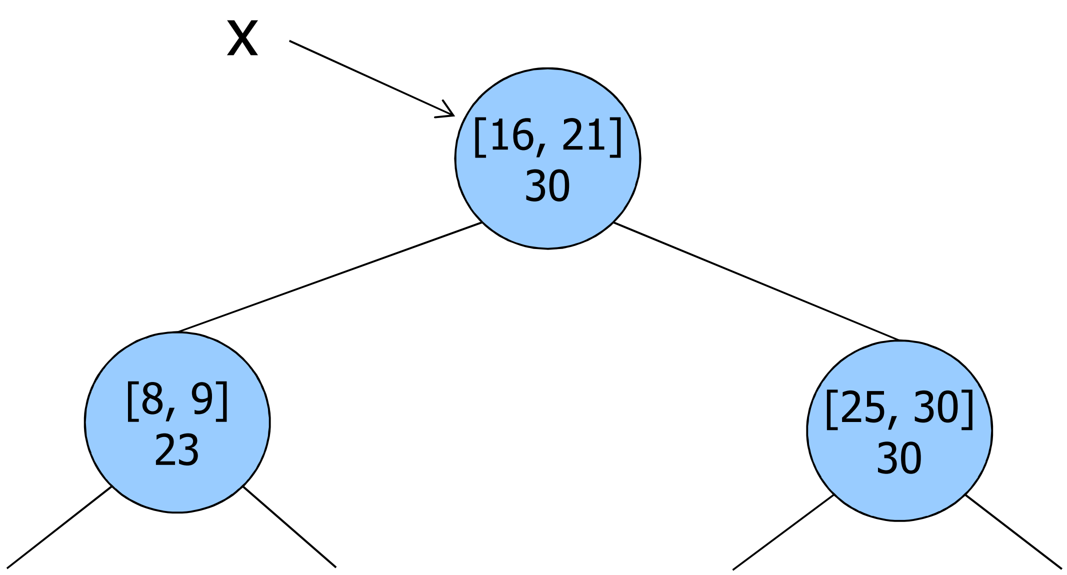
- **Insert** an item (interval) into the Interval BST
  - void IBSTinsert (IBST, Item) ;
- **Delete** an item (interval) from the Interval BST
  - void IBSTdelete (IBST, Item) ;
- **Search** an item (interval) into the Interval BST and **return the first interval** with an intersection
  - Item IBSTsearch (IBST, Item) ;

Insert

- To insert a new node into an I-BST
  - It is sufficient to use a "standard" BST insertion procedure "working" on the left endpoint
  - It is necessary to determine the maximum value for each new node
- An inorder tree walk of the tree lists the nodes in sorted order by left endpoint

Maximum evaluation

- The evaluation of the maximum has complexity $\Theta(1)$
  - `x->max =`
    `max (high(x), x->left->max, x->right->max)`



Delete

- It requires a search and then a delete
- Seach is the only new operation we have to develop
- Delete, once the element has been found, can be performed using the "standard" BST approach

Search

- Search a node n with an interval having an intersection with interval i
  - Visit the tree from root
  - Termination
    - Find an interval with an intersection with i or
    - An empty tree has been reached
  - Recursion from node n
    - On the right sub-tree
    - On the left sub-tree

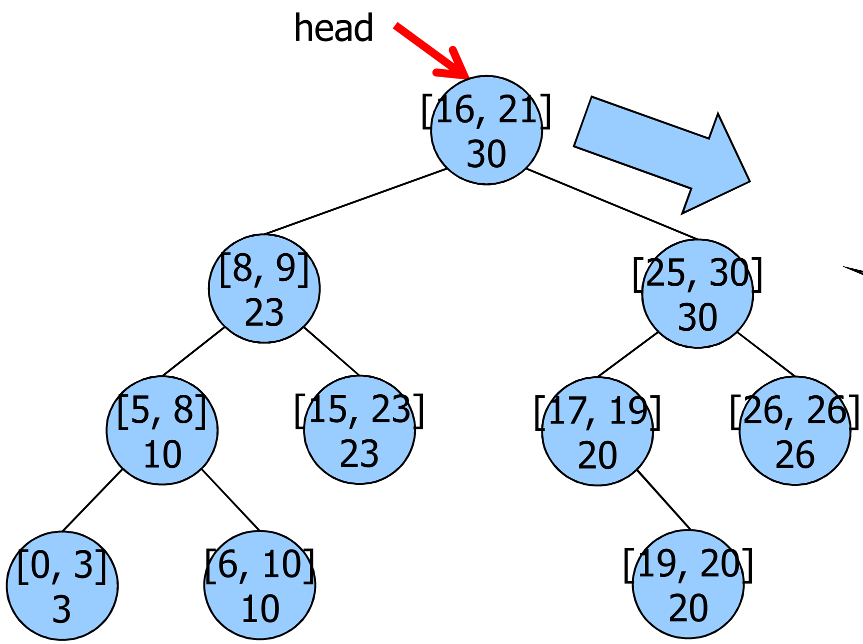
Search

- Recur on the right sub-tree if
  - n->l==NULL OR low[i] > n->l->max

current node

interval

If this condition holds, then it is not possibile to have an intersection on the left sub-tree



Search an interval with an intersection with [25, 30]

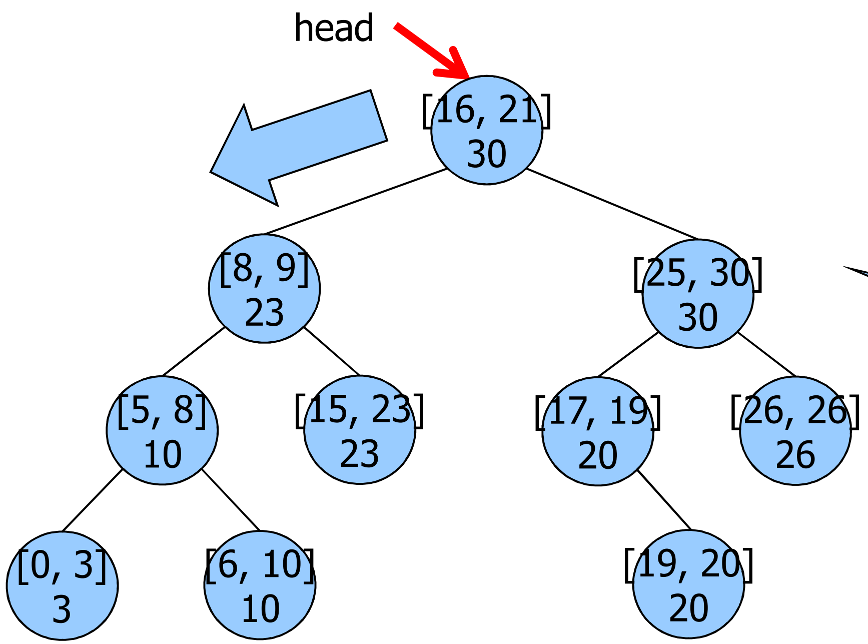
Search

- Recur on the left sub-tree if
  - low[i] ≤ n->l->max

interval

current node

If this condition holds, then if there is no intersaction on the left subtree there is no intersaction on the right one



Search an interval with an intersaction with [21, 27]

Search

- Recur on the left sub-tree if
  - low[i] ≤ n->l->max

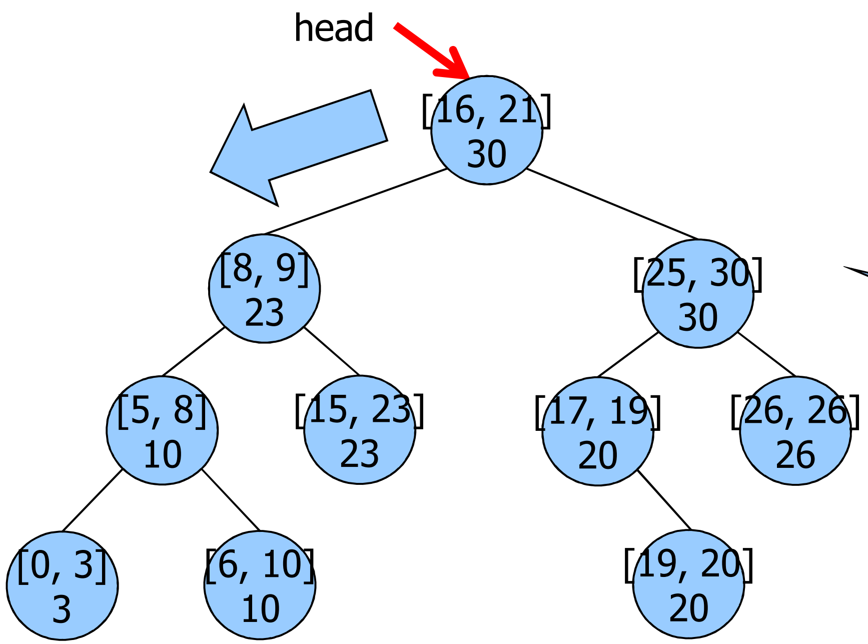
interval

current node

There is no intersaction on the left when

low[i] high[i]

Then on the right low endpoints are even higher



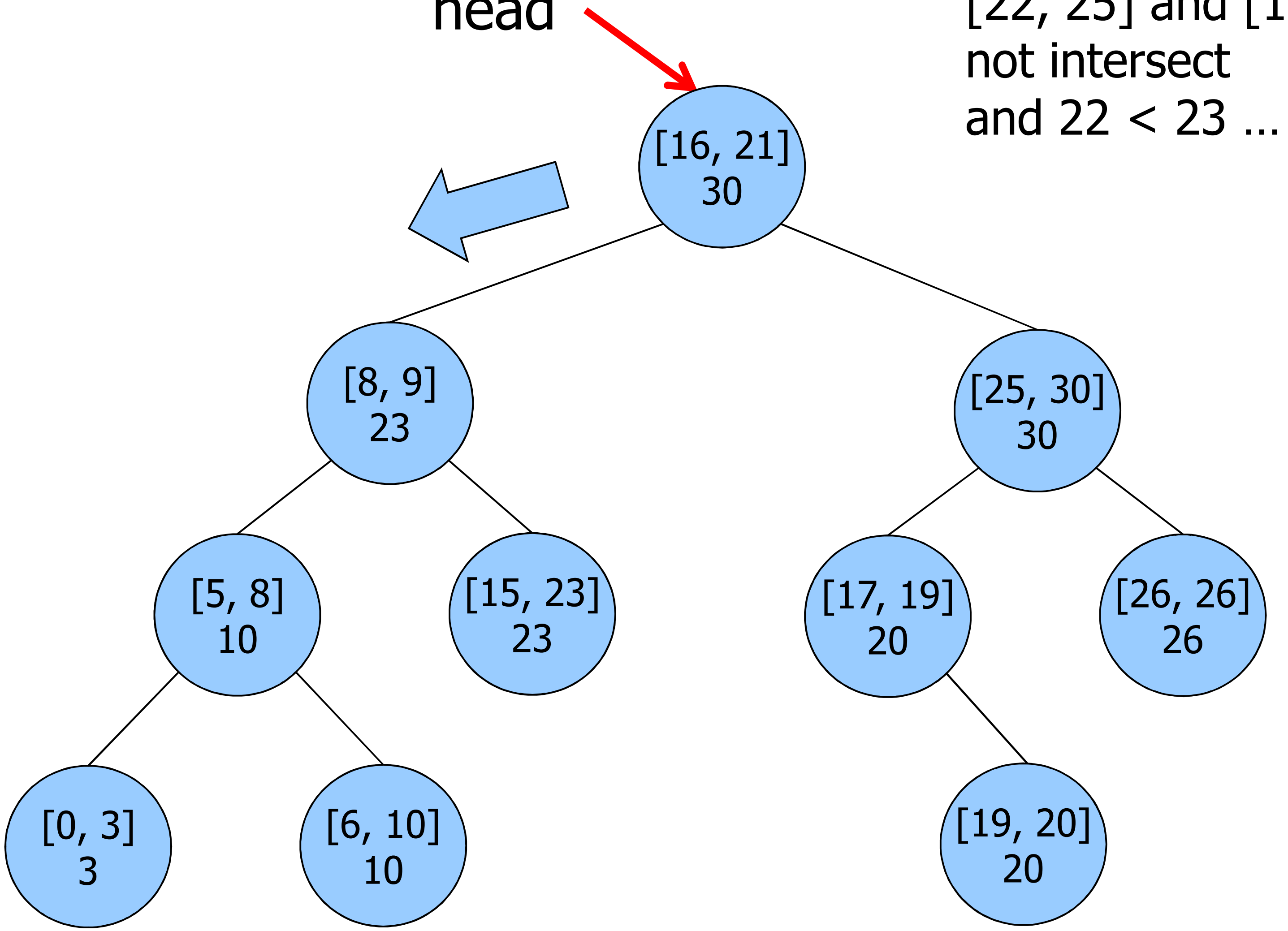
Search an interval with an intersection with [21, 27]

Example

Search an interval with an intersection with [22, 25]

head

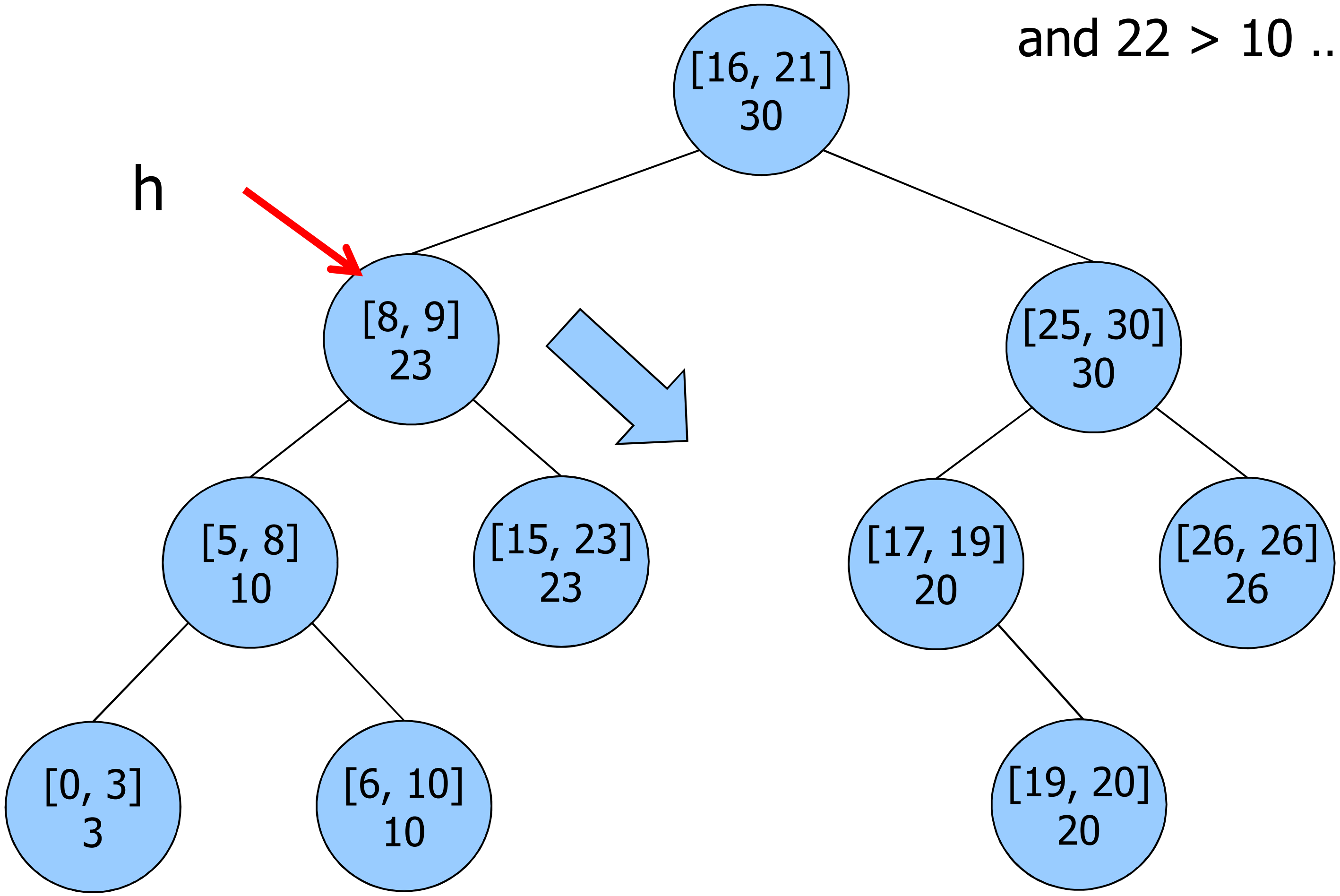
[22, 25] and [16, 21] do not intersect
and 22 < 23 … recur left



Example

Search an interval with an intersection with [22, 25]

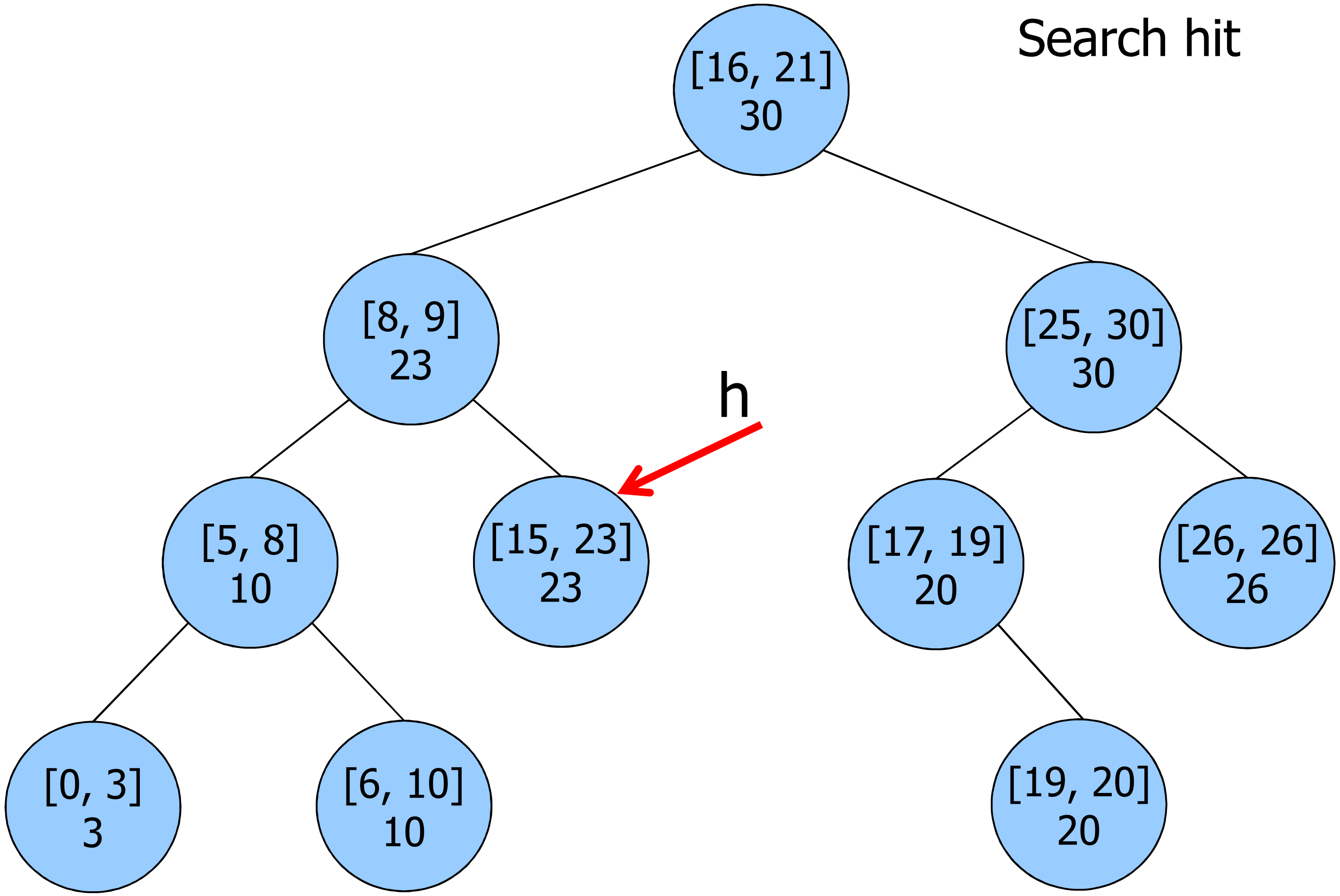
[22, 25] and [8, 9] do not intersect
and 22 > 10 … recur right



Example

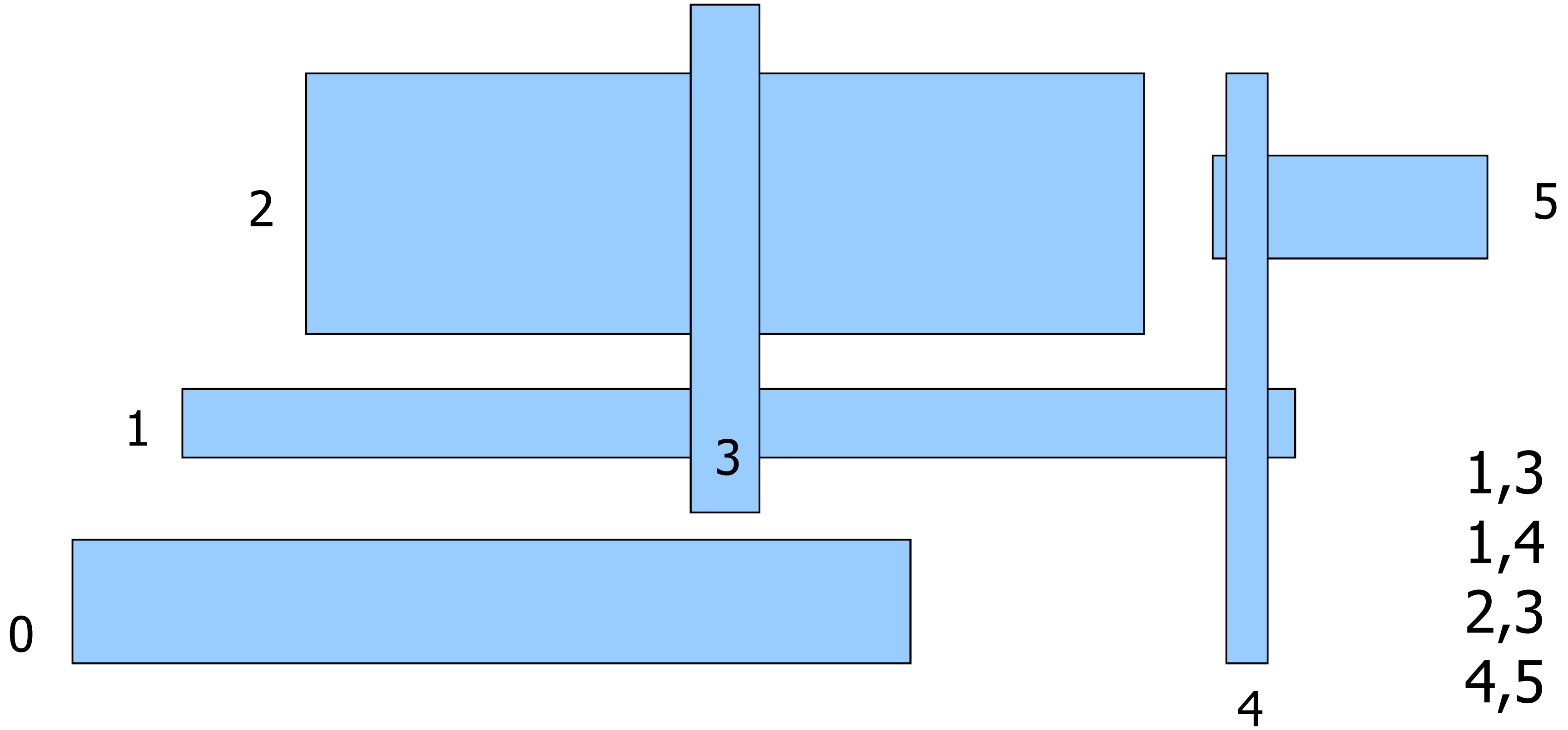
Search an interval with an intersection with [22, 25]

[22, 25] and [15, 23] do intersect
Search hit



Application

Given N rectangles placed with sides parallel to the Cartesian axis, check whether a new rectangle has an intersection with another rectangle



1,3
1,4
2,3
4,5

Application

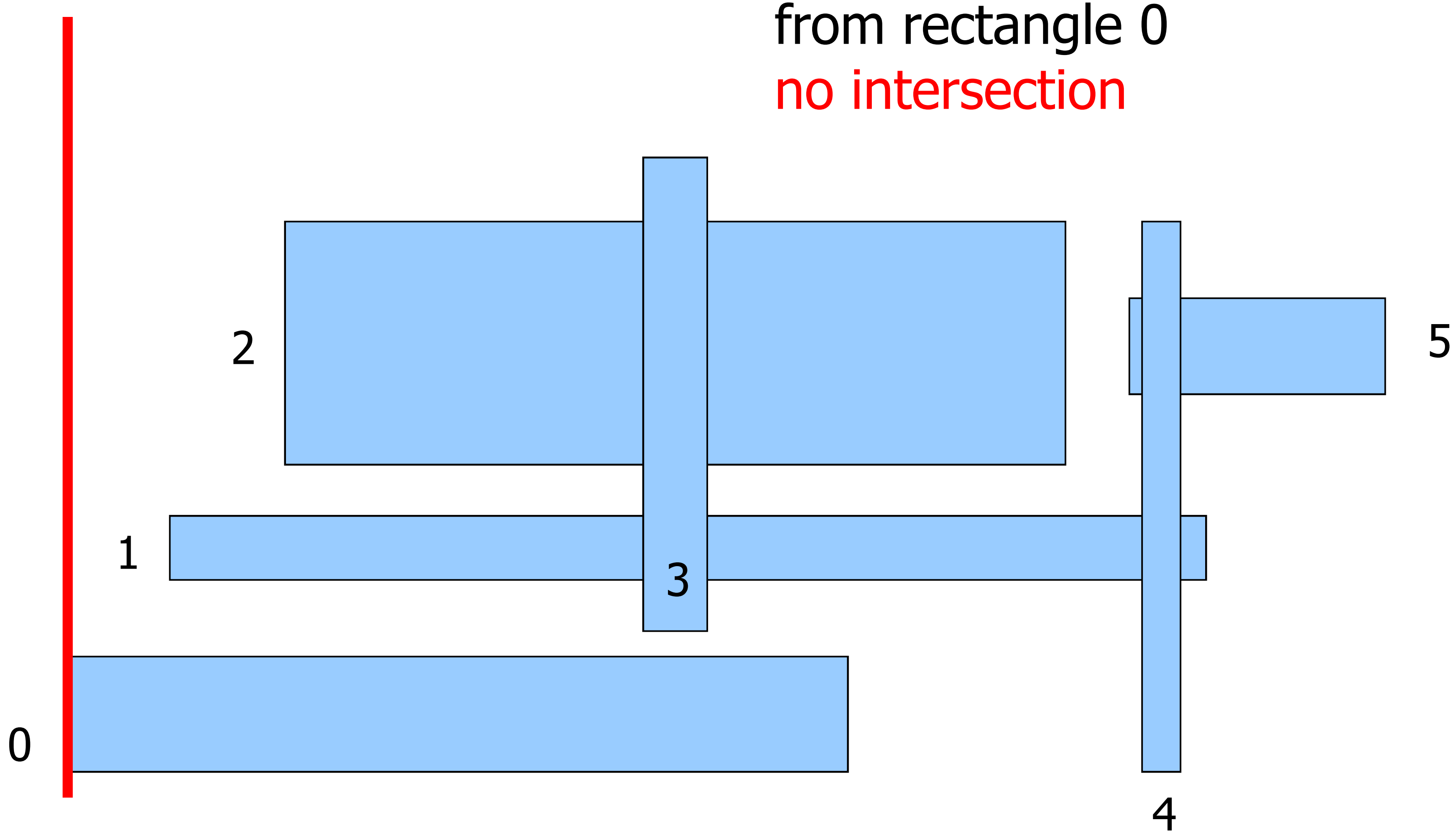
- Electronic CAD Application
  - Verify if the there are connections with an intersection on an electronic circuit
- Basic Algorithm
  - Check the intersection among all rectangle couples
  - Complexity $O(N^2)$

Application

- Algorithms using IBST
  - Order rectangles based on ascending left extreme x-values
  - Iterate on rectangles for ascending x-values
    - When a left extreme in encountered, insert the y-value range into an I-BST and check for intersactions
    - When a right extreme is found, remove the interval from the I-BST y-values
- Efficient algorithm
  - Complexity O(N·logN)
  - Applicability to VLSI and beyond

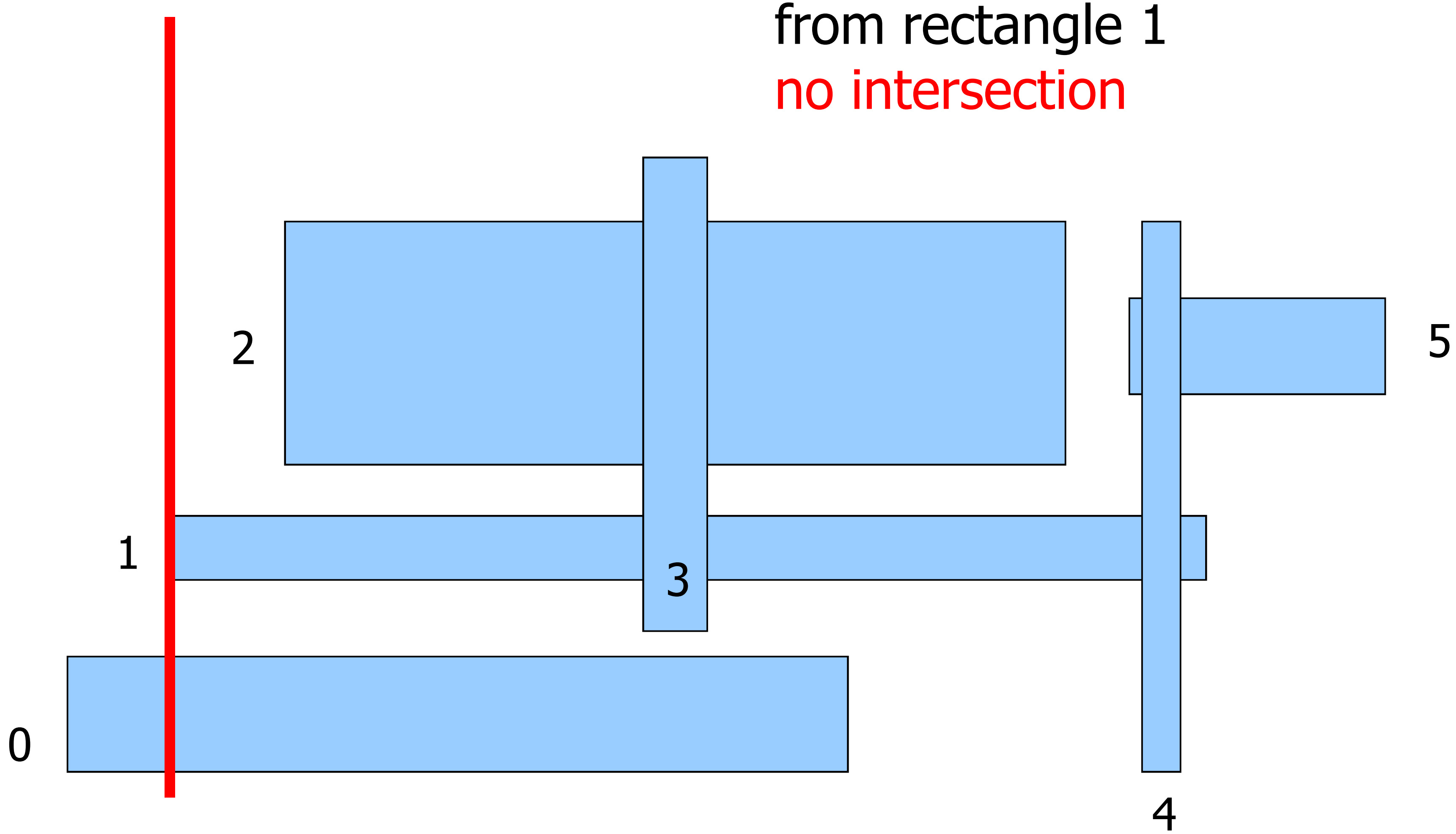
Example

Insert interval on y-values from rectangle 0
no intersection



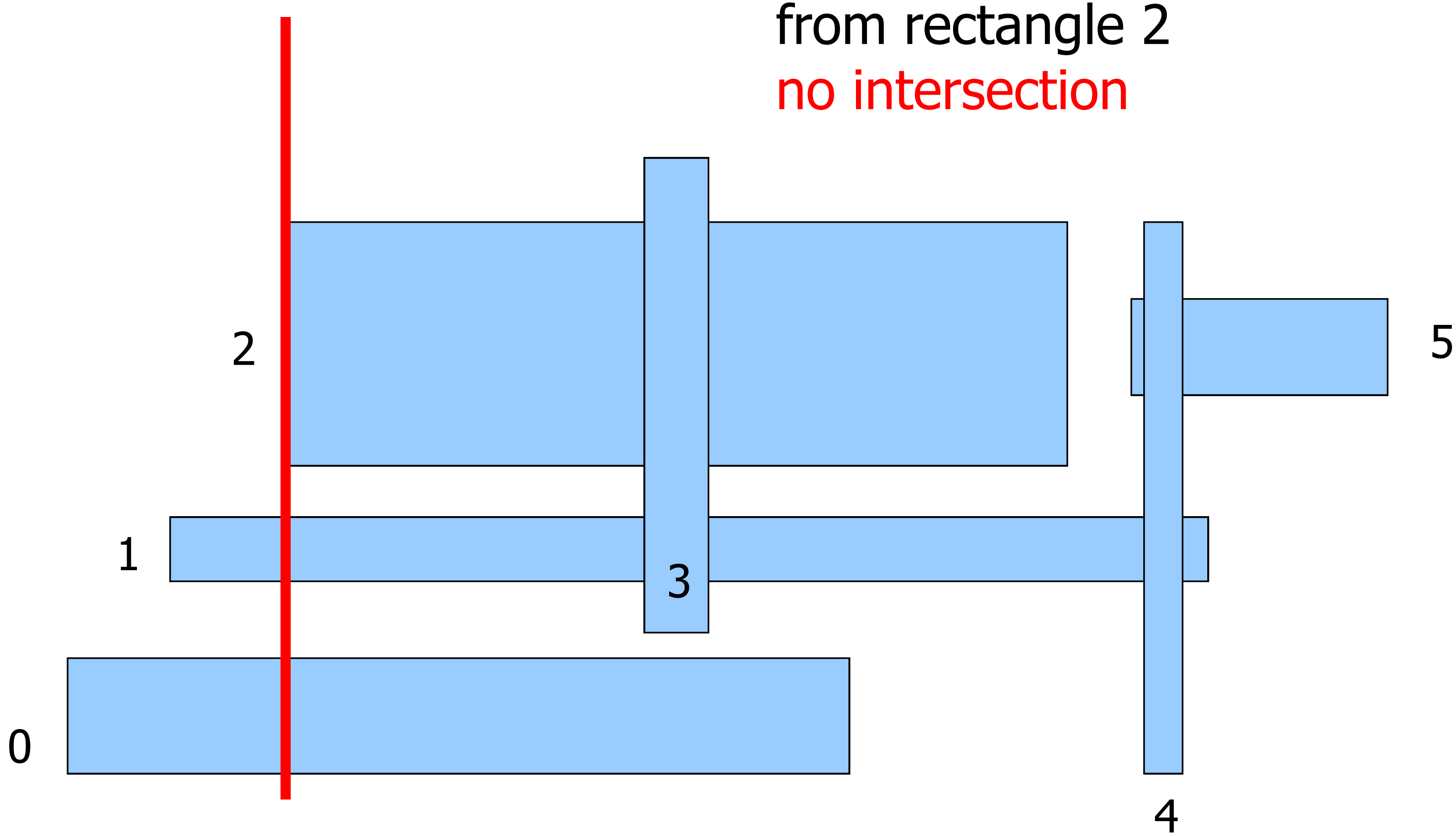
Example

Insert interval on y-values
from rectangle 1
no intersection



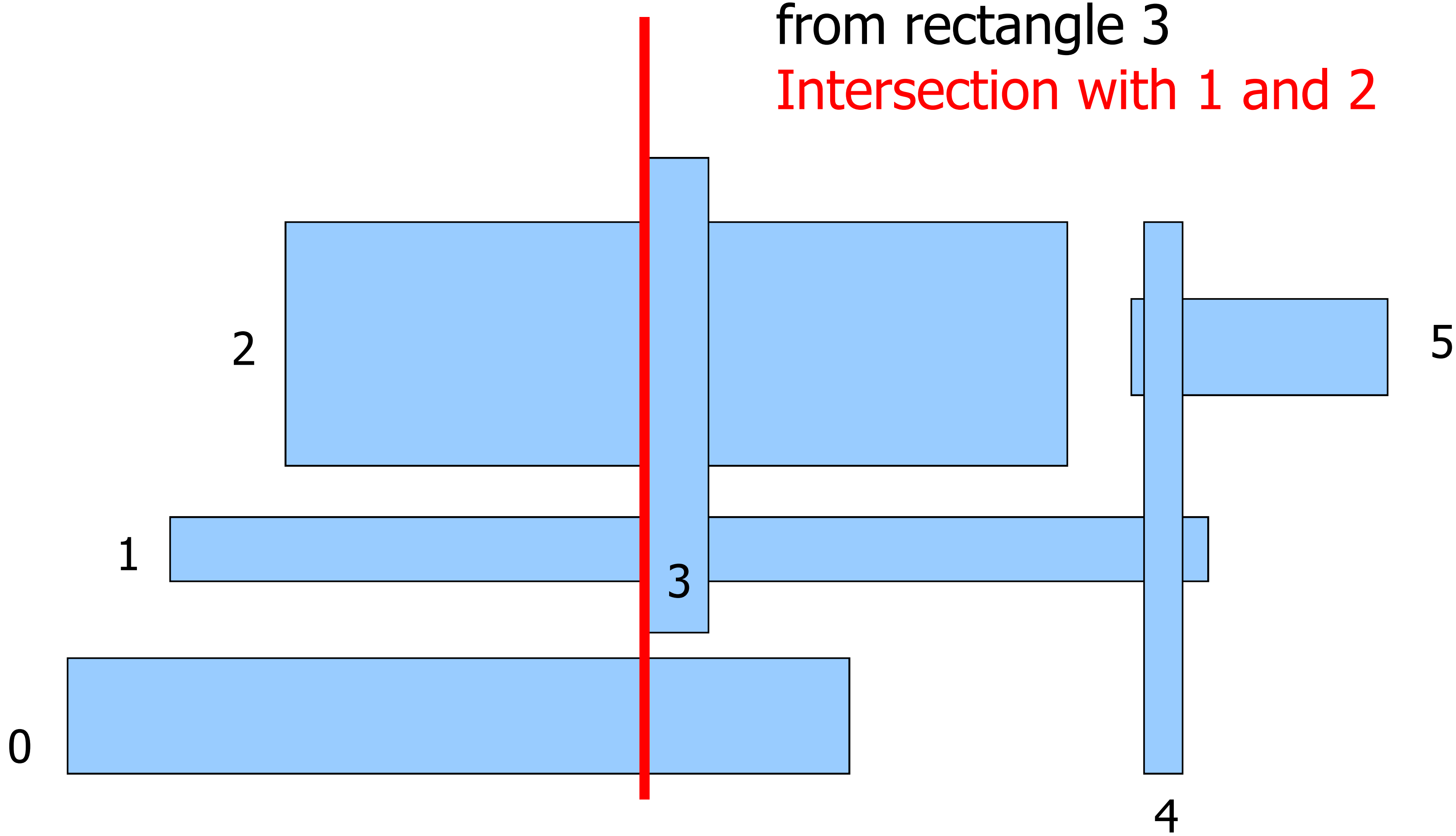
Example

Insert interval on y-values from rectangle 2
no intersection



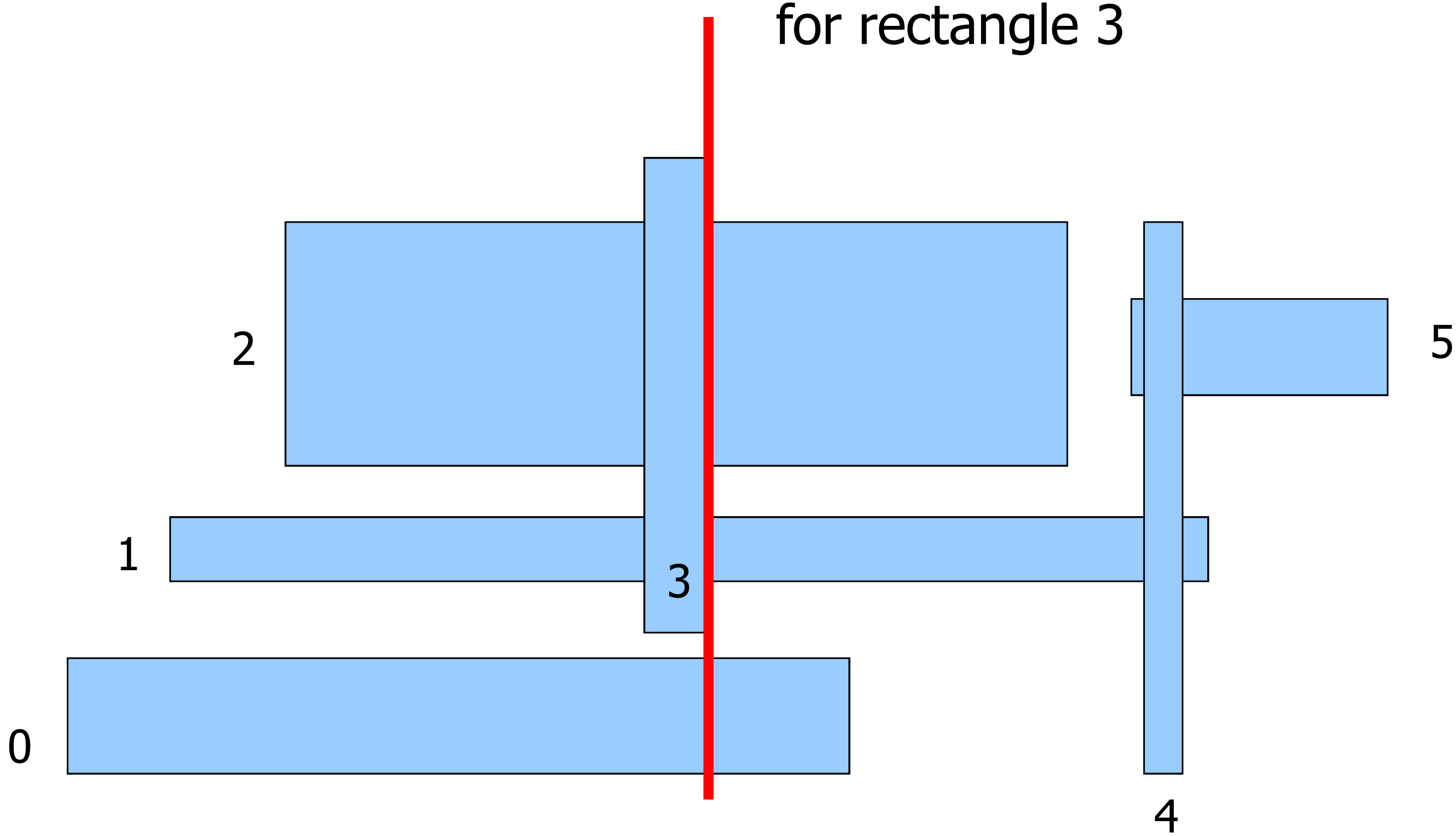
Example

Insert interval on y-values
from rectangle 3
Intersection with 1 and 2



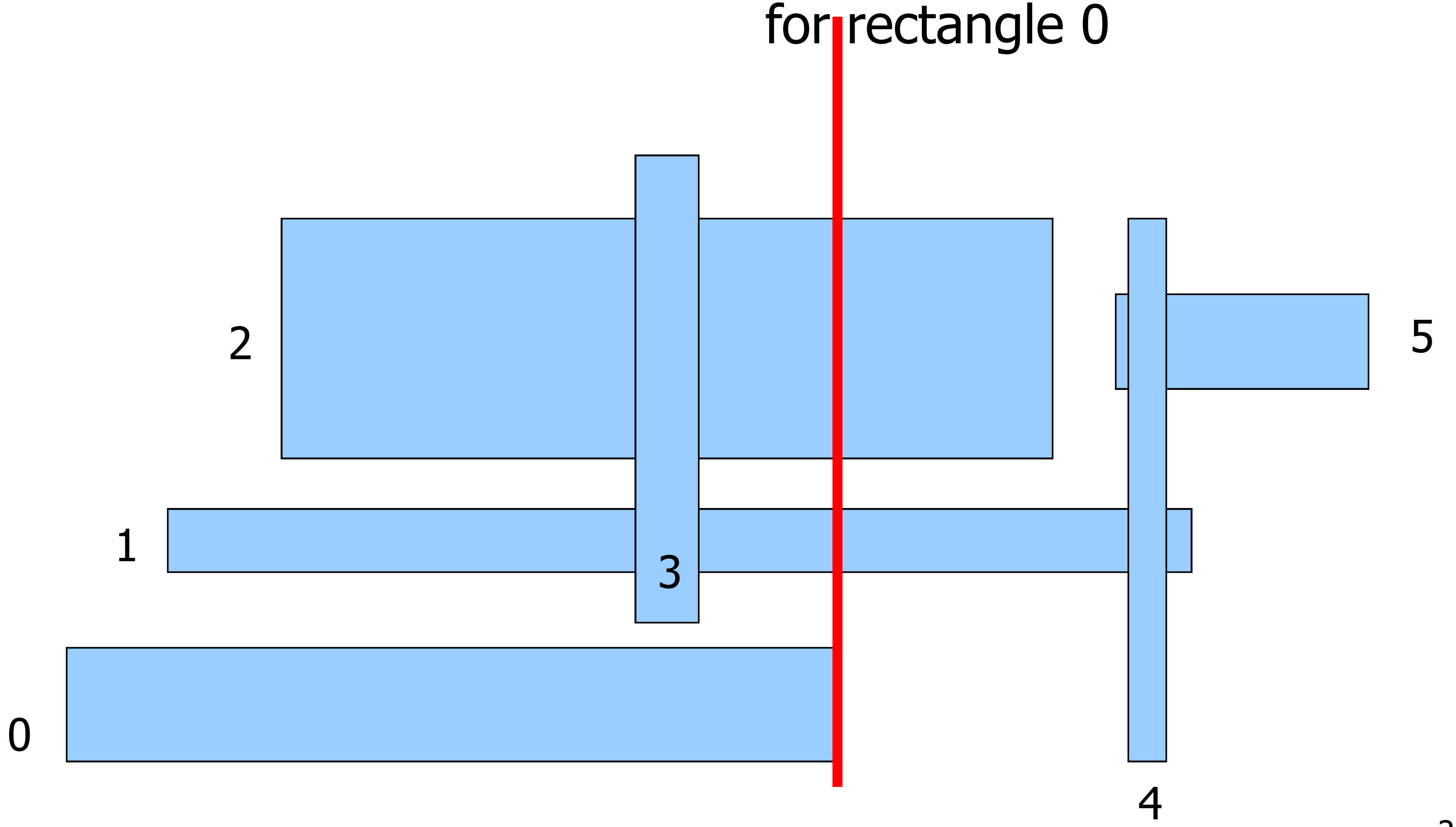
Example

Remove y-value interval for rectangle 3



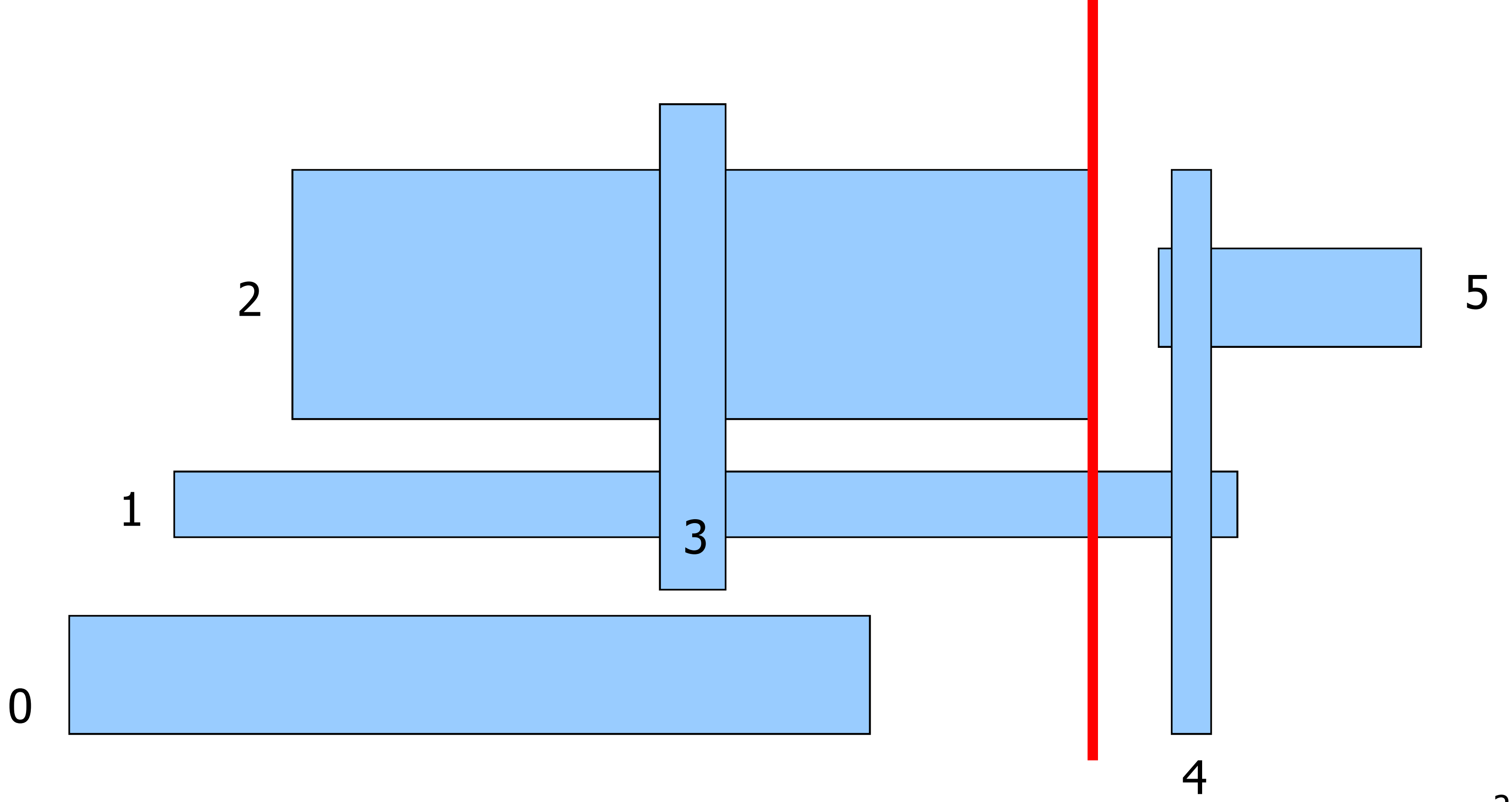
Example

Remove y-value interval for rectangle 0



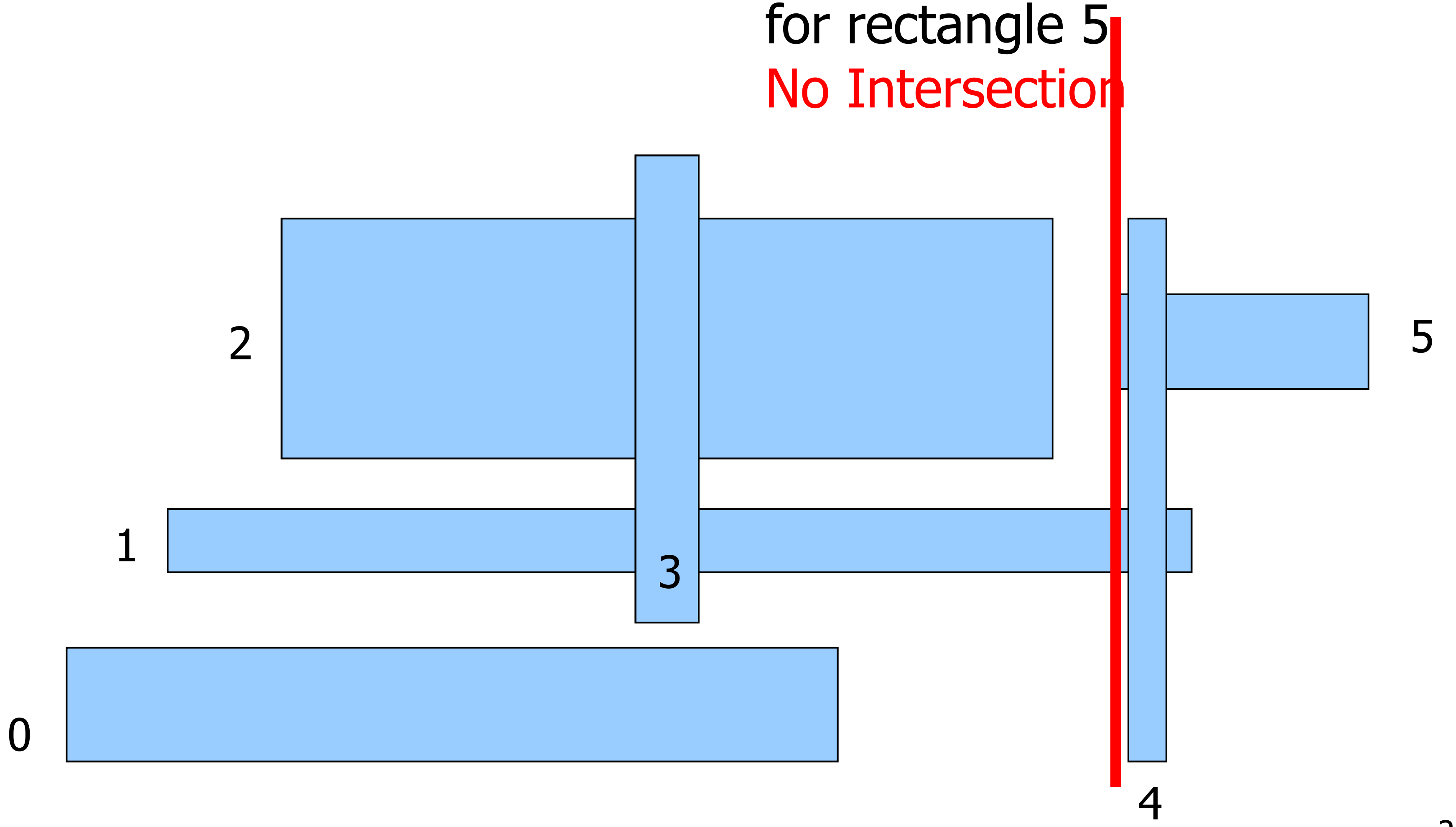
Example

Remove y-value interval for rectangle 2



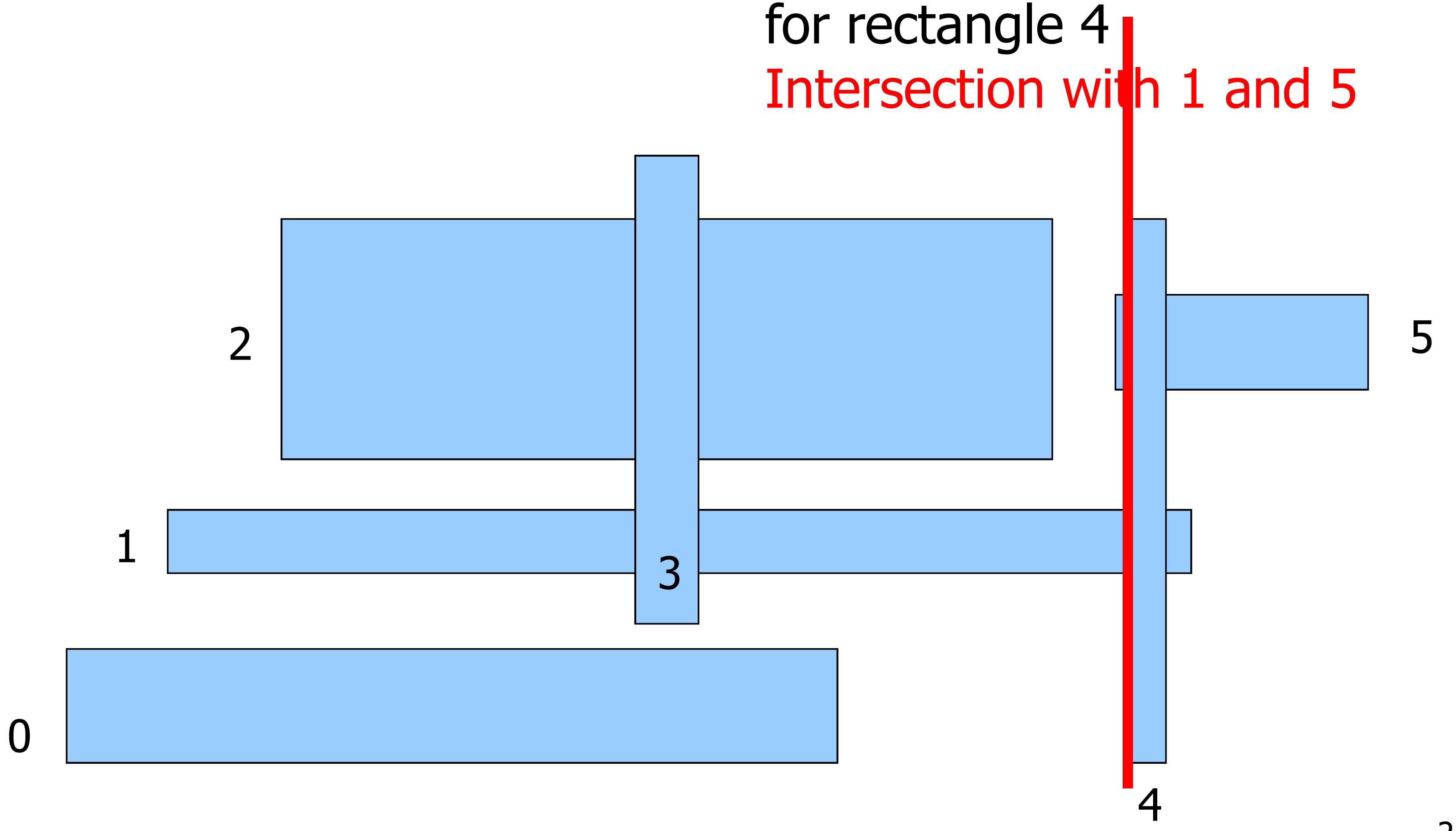
Example

Insert interval on y-values
for rectangle 5
No Intersection



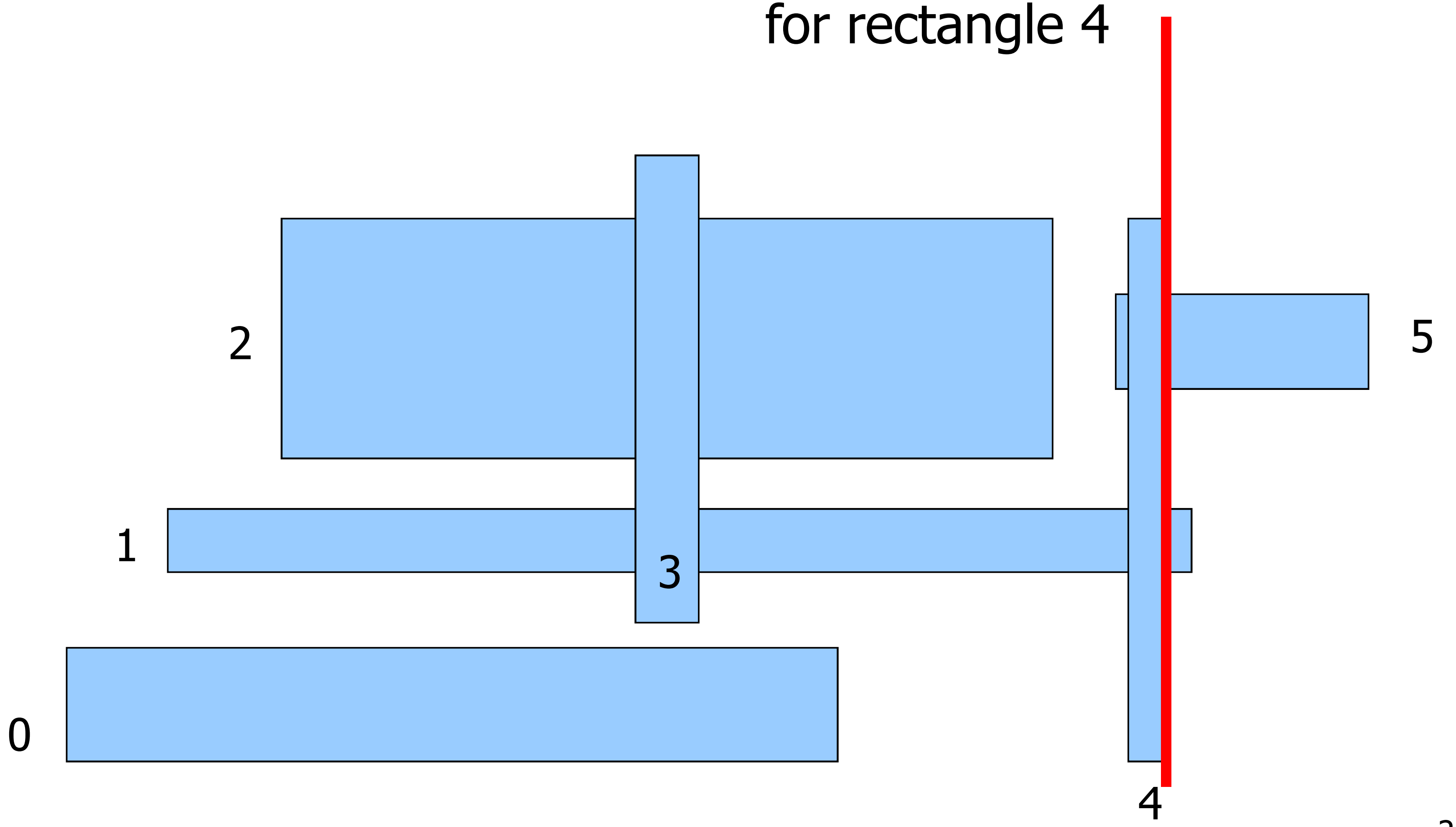
Example

Insert interval on y-values
for rectangle 4
Intersection with 1 and 5



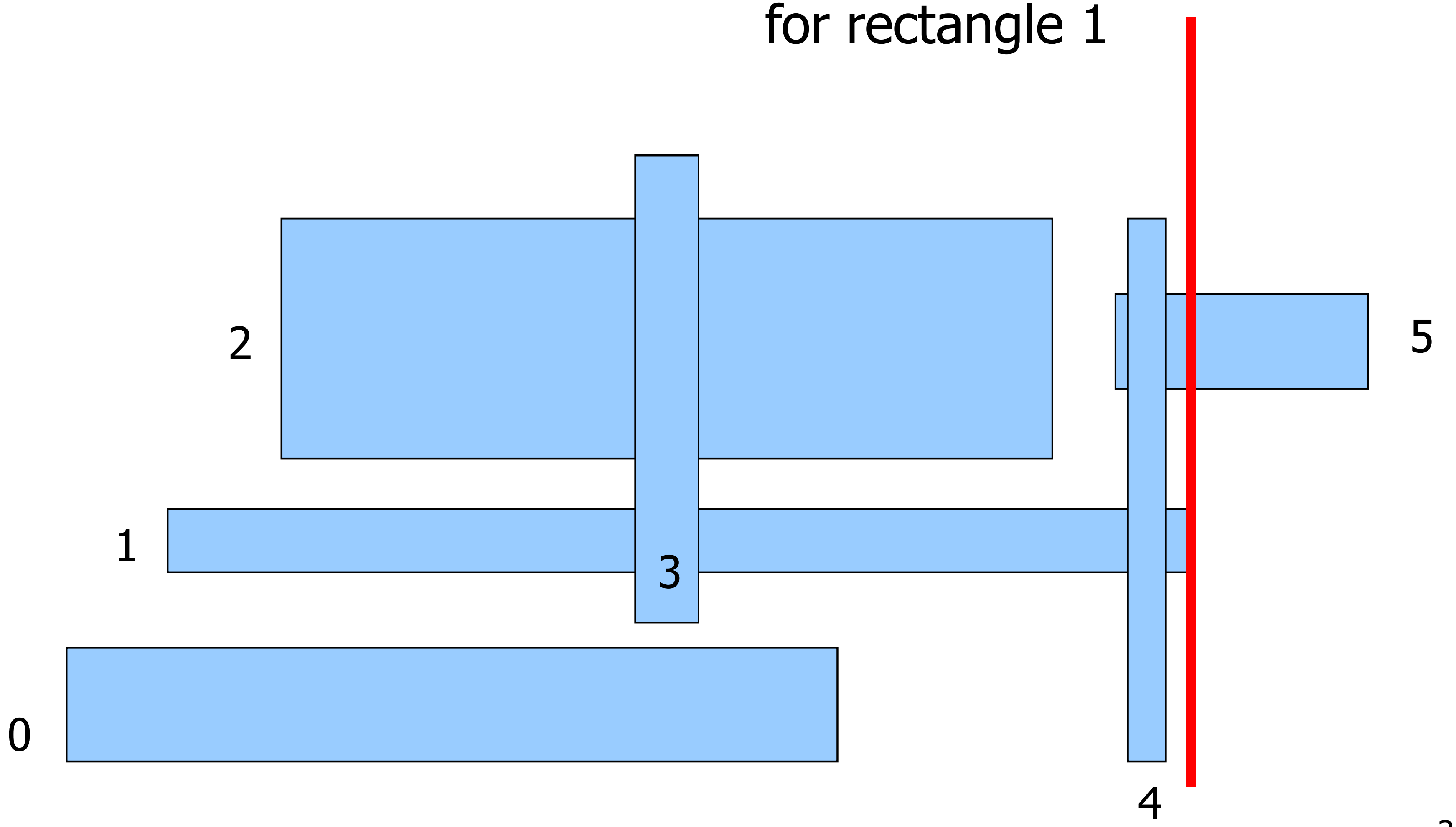
Example

Remove y-value interval for rectangle 4



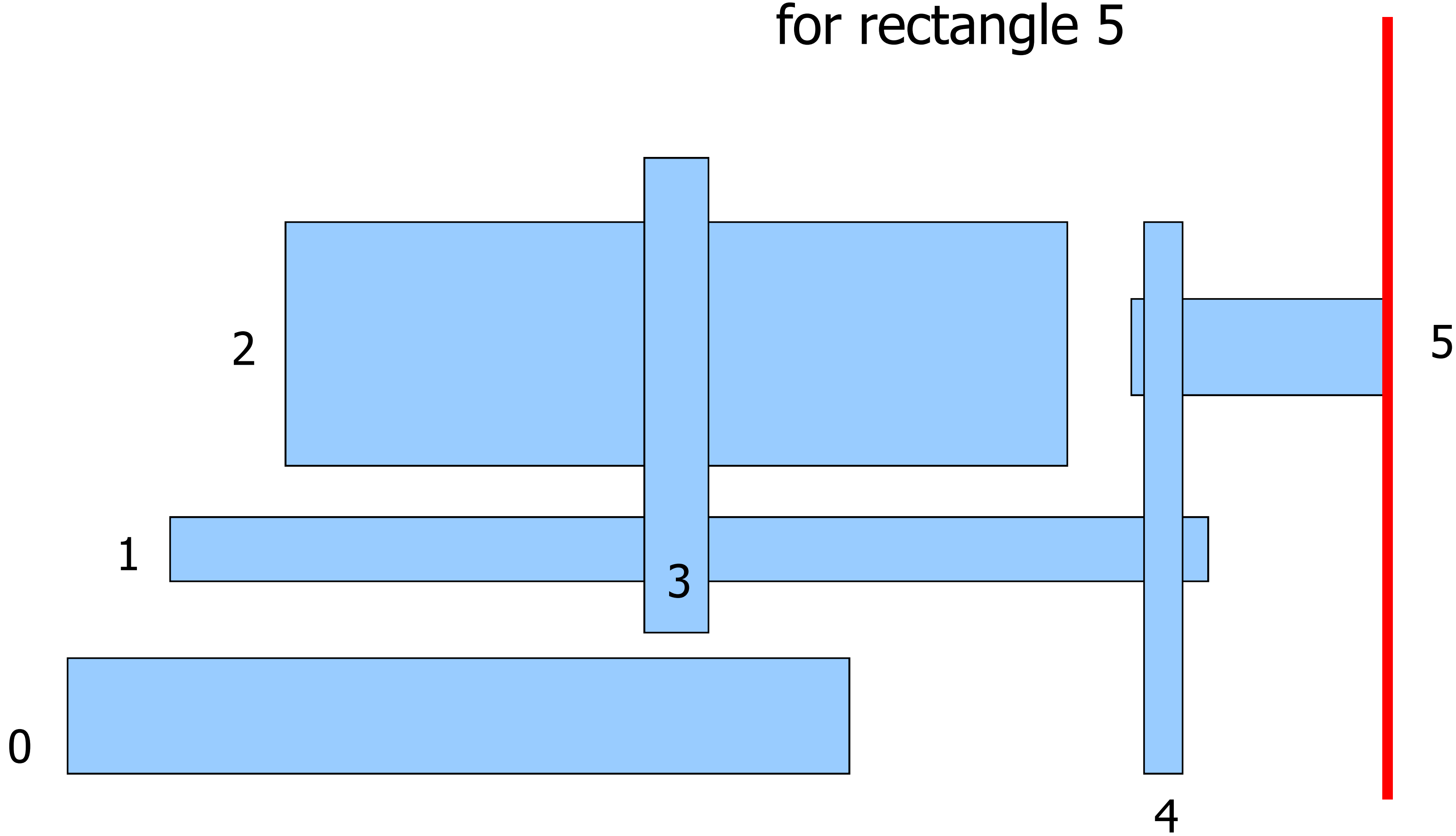
Example

Remove y-value interval for rectangle 1



Example

Remove y-value interval for rectangle 5



Analysis

- Sorting
  - $O(N \cdot \log N)$
- If the IBST is balanced
  - Each insertion/deletion of an interval or seach of the first interval that intersects a given one has cost $O(\log N)$
  - Searching for all intervals that intersect a given one has cost $O(R\log N)$, where R is the number of intersections