



**DCGI**

DEPARTMENT OF COMPUTER GRAPHICS AND INTERACTION

# INTERSECTIONS OF LINE SEGMENTS AND POLYGONS

**PETR FELKEL**

FEL CTU PRAGUE

felkel@fel.cvut.cz

<https://cw.felk.cvut.cz/doku.php/courses/a4m39vg/start>

Based on [Berg], [Mount], [Kukral], and [Drtina]

Version from 19.11.2015

# Talk overview

---

- Intersections of line segments (Bentley-Ottmann)
  - Motivation
  - Sweep line algorithm recapitulation
  - Sweep line intersections of line segments
- Intersection of polygons or planar subdivisions
  - See assignment [21] or [Berg, Section 2.3]
- Intersection of axis parallel rectangles
  - See assignment [26]



# Geometric intersections – what are they for?

---

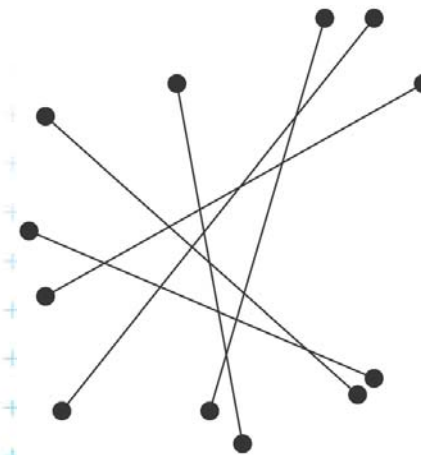
One of the most basic problems in computational geometry

- Solid modeling
  - Intersection of object boundaries in CSG
- Overlay of subdivisions, e.g. layers in GIS
  - Bridges on intersections of roads and rivers
  - Maintenance responsibilities (road network X county boundaries)
- Robotics
  - Collision detection and collision avoidance
- Computer graphics
  - Rendering via ray shooting (intersection of the ray with objects)
- ...



# Line segment intersection

- Intersection of complex shapes is often reduced to simpler and simpler intersection problems
- **Line segment intersection** is the most basic intersection algorithm
- **Problem statement:**  
Given  $n$  line segments in the plane, report all points where a pair of line segments intersect.
- **Problem complexity**
  - Worst case –  $I = O(n^2)$  intersections
  - Practical case – only some intersections
  - Use an **output sensitive algorithm**
    - $O(n \log n + I)$  optimal randomized algorithm
    - $O(n \log n + I \log n)$  **sweep line algorithm** - %



[Berg]



# Plane sweep line algorithm recapitulation

---

- Horizontal line (**sweep line**, *scan line*)  $\ell$  moves top-down (or vertical line: left to right) over the set of objects
  - The move is not continuous, but  $\ell$  **jumps from one event point to another**
    - Event points are in **priority queue** or sorted list ( $\sim y$ )
    - The (left) top-most event point is removed first
    - **New event points** may be created (usually as interaction of **neighbors** on the sweep line) and **inserted in the queue**
  - **Scan-line status**
    - Stores information about the objects intersected by  $\ell$
- It is updated while stopping on event point



# Line segment intersection - Sweep line alg.

---

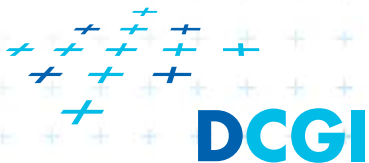
- Avoid testing of pairs of segments far apart
- Compute **intersections of neighbors** on the sweep line only
- $O(n \log n + I \log n)$  time in  $O(n)$  memory
  - $2n$  steps for end points,
  - $I$  steps for intersections,
  - $\log n$  search the status tree
- Ignore “nasty cases” (most of them will be solved later on)
  - No segment is parallel to the sweep line
  - Segments intersect in one point and do not overlap
  - No three segments meet in a common point



# Line segment intersections

---

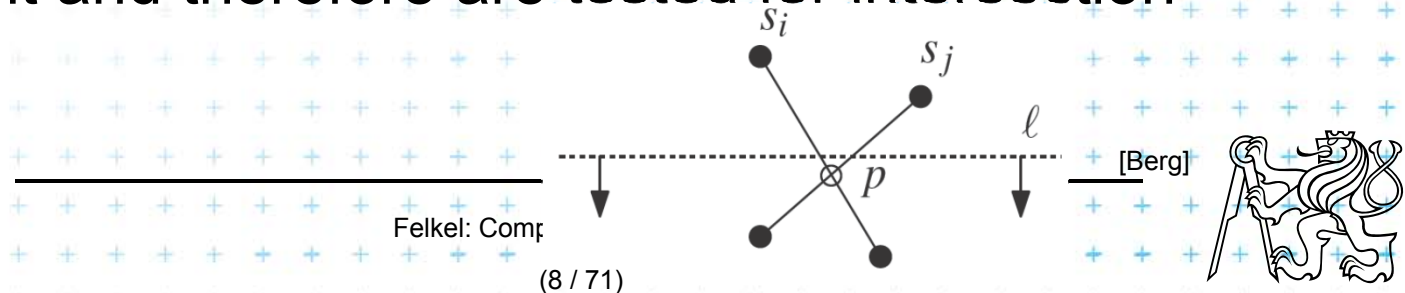
- *Status* = ordered sequence of segments intersecting the sweep line  $\ell$
- *Events* (waiting in the priority queue)
  - = points, where the algorithm actually does something
  - Segment *end-points*
    - known at algorithm start
  - Segment *intersections* between neighboring segments along SL
    - Discovered as the sweep executes



# Detecting intersections

- Intersection events must be **detected** and inserted to the event queue **before they occur**
- Given two segments  $a, b$  intersecting in a point  $p$ , there must be a placement of sweep line  $\ell$  prior to  $p$ , such that segments  $a, b$  are **adjacent along  $\ell$**  (only adjacent will be tested for intersection)
  - segments  $a, b$  are not adjacent when the alg. starts
  - segments  $a, b$  are adjacent just before  $p$

=> there must be an event point when  $a, b$  become adjacent and therefore are tested for intersection

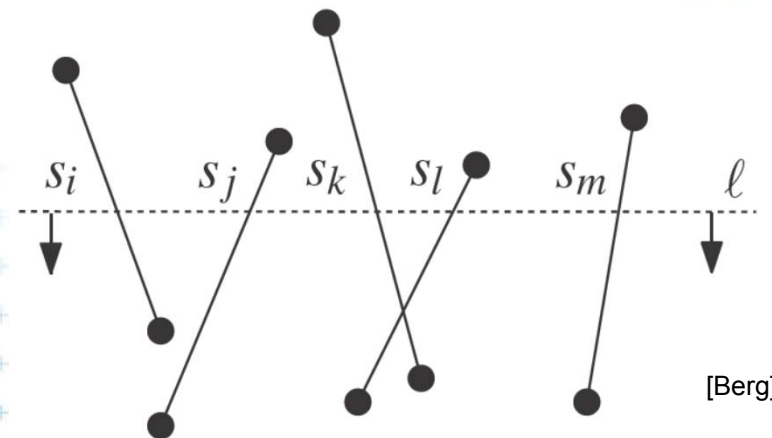
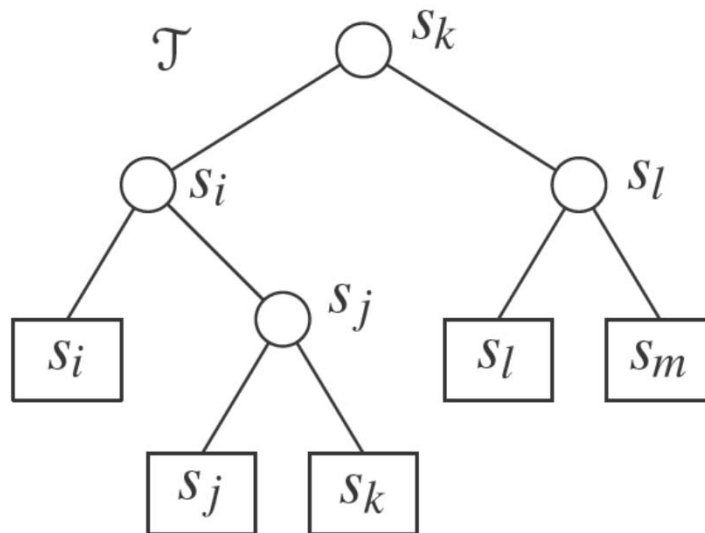




# Data structures

Sweep line  $\ell$  **status** = order of segments along  $\ell$

- Balanced binary search tree of segments
- Coords of intersections with  $\ell$  vary as  $\ell$  moves  
=> store pointers to line segments in tree nodes
  - Position of  $\ell$  is plugged in the  $y=mx+b$  to get the x-key



[Berg]



# Data structures

---

**Event queue** (postupový plán, časový plán)

- Define: **Order**  $<$  (top-down, lexicographic)

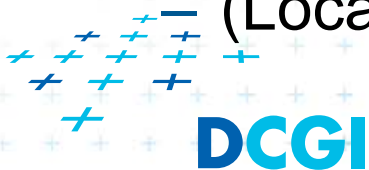
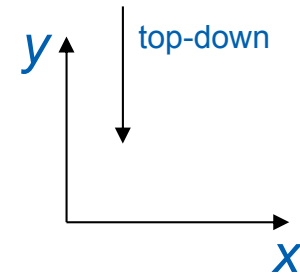
$p < q$  **iff**  $p_y > q_y$  **or**  $p_y = q_y$  and  $p_x < q_x$

top-down, left-right approach

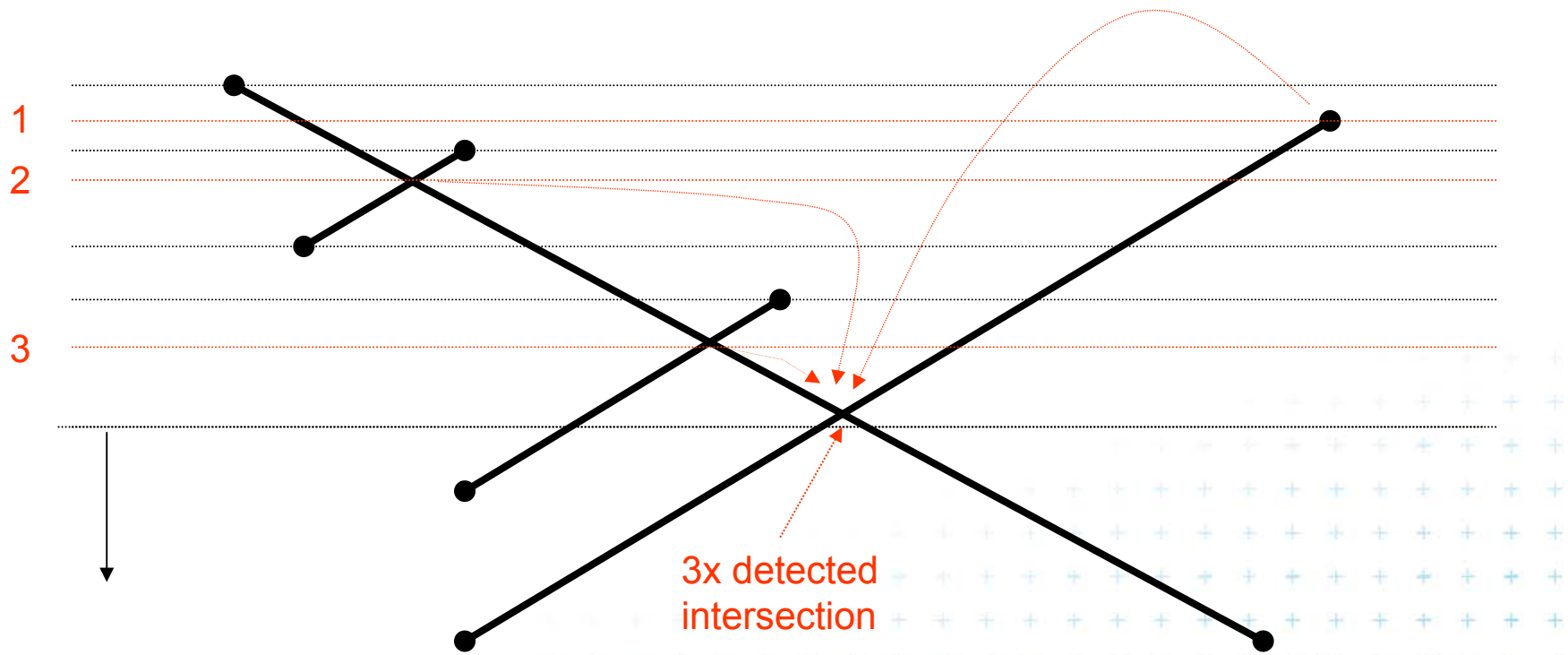
(points on  $\ell$  treated left to right)

- Operations

- **Insertion** of computed intersection points
- Fetching the **next event** (highest  $y$  below  $\ell$ )
- **Test**, if the segment is already **present in the queue**
- (Locate and **delete** intersection event in the queue)



# Problem with duplicities of intersections



# Data structures

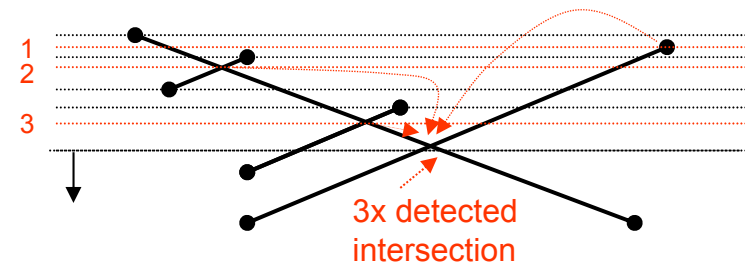
## Event queue data structure

### ■ Heap

- Problem: can not check **duplicated intersection events** (reinvented more than once)
- Intersections processed twice or even more
- Memory complexity up to  $O(n^2)$

### ■ Ordered dictionary (balanced binary tree)

- Can check duplicated events (adds just constant factor)
- Nothing inserted twice
- If non-neighbor intersections are deleted  
i.e., if only intersections of neighbors along  $\ell$  are stored  
then memory complexity just  $O(n)$



# Line segment intersection algorithm

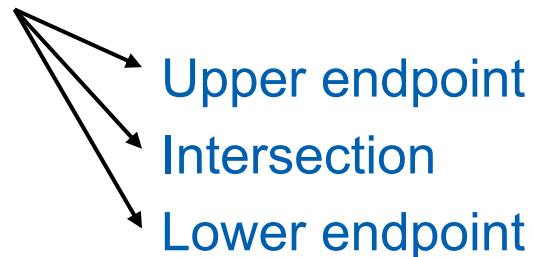
---

## FindIntersections( $S$ )

*Input:* A set  $S$  of line segments in the plane

*Output:* The set of intersection points + pointers to segments in each

1. init an empty event queue  $Q$  and insert the segment endpoints
2. init an empty status structure  $T$
3. **while**  $Q$  in not empty
4.     remove next event  $p$  from  $Q$
5.     handleEventPoint( $p$ )

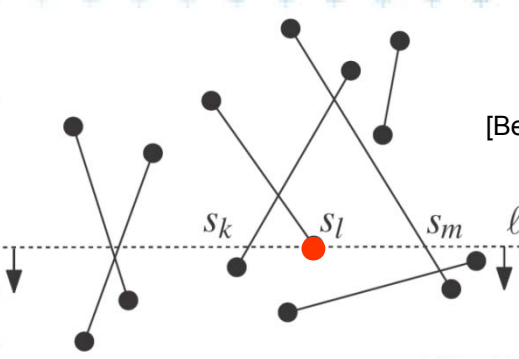
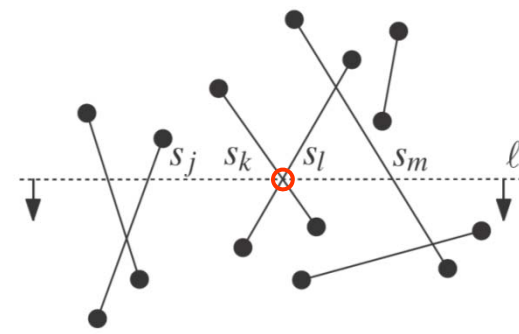
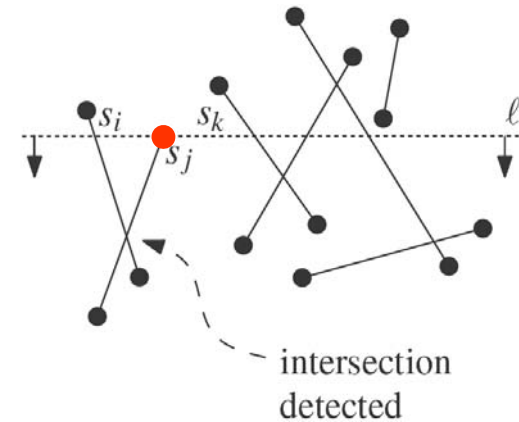


Note: Upper-end-point events store info about the segment

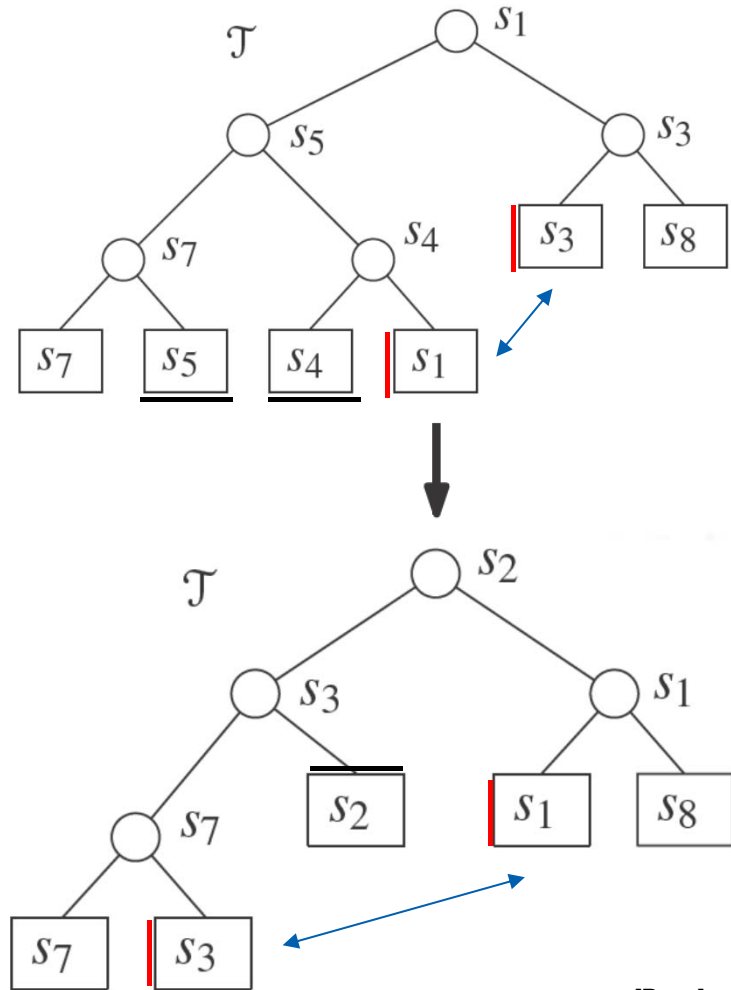
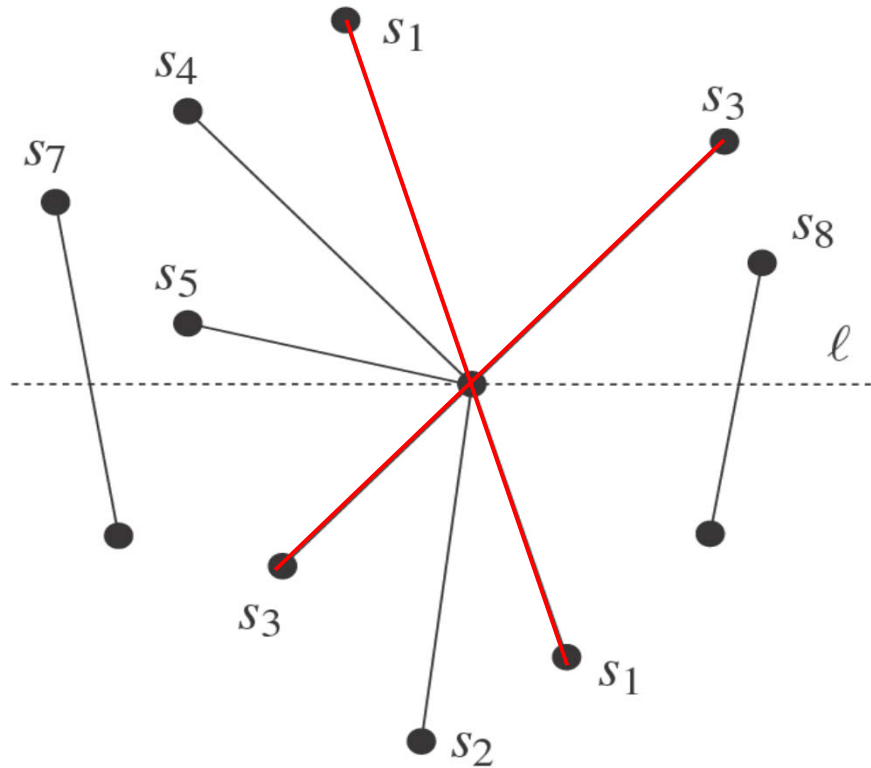


# handleEventPoint() principle

- Upper endpoint  $U(p)$ 
  - insert  $p$  (on  $s_j$ ) to status  $T$
  - add intersections with left and right neighbors to  $Q$
- Intersection  $C(p)$ 
  - switch order of segments in  $T$
  - add intersections of left and right neighbors to  $Q$
- Lower endpoint  $L(p)$ 
  - remove  $p$  (on  $s_l$ ) from  $T$
  - add intersections of left and right neighbors to  $Q$






# More than two segments incident



$$U(p) = \{s_2\}$$

$$C(p) = \{s_1, s_3\}$$

$$L(p) = \{s_4, s_5\}$$

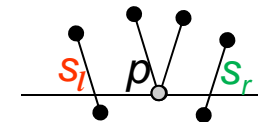
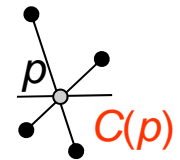
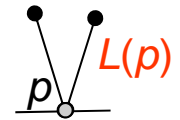
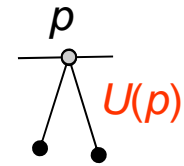
-  start here
-  cross on  $l$
-  end here



# Handle Events [Berg, page 25]

## handleEventPoint(p)

1. Let  $U(p)$  = set of segments whose **Upper endpoint is  $p$** .  
These segments are stored with the event point  $p$  (will be added to  $T$ )
2. **Search  $T$**  for all segments  $S(p)$  that contain  $p$  (are adjacent in  $T$ ):  
Let  $L(p) \subset S(p)$  = segments whose **Lower endpoint is  $p$**   
Let  $C(p) \subset S(p)$  = segments that **Contain  $p$  in interior**
3. **if** ( $L(p) \cup U(p) \cup C(p)$  contains more than one segment )
4.     **report  $p$  as intersection** together with  $L(p), U(p), C(p)$
5. Delete the segments in  $L(p) \cup C(p)$  from  $T$
6. Insert the segments in  $U(p) \cup C(p)$  into  $T$  } Reverse order of  $C(p)$  in  $T$   
(order as below  $\ell$ , horizontal segment as the last)
7. **if** ( $U(p) \cup C(p) = \emptyset$  ) then **findNewEvent**( $s_l, s_r, p$ ) // left & right neighbors
8. **else**  $s'$  = leftmost segment of  $U(p) \cup C(p)$ ; **findNewEvent**( $s_l, s', p$ )  
 $s''$  = rightmost segment of  $U(p) \cup C(p)$ ; **findNewEvent**( $s'', s_r, p$ )





# Detection of new intersections

**findNewEvent( $s_l, s_r, p$ )** // with handling of horizontal segments

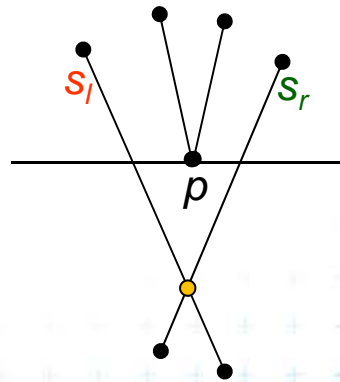
*Input:* two segments (left & right from  $p$  in  $T$ ) and a current event point  $p$

*Output:* updated event queue  $Q$  with new intersection

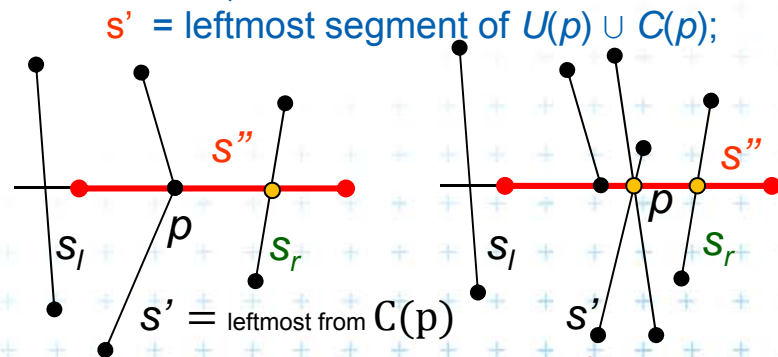
1. if [ (  $s_l$  and  $s_r$  intersect below the sweep line  $\ell$  ) or  
 ( intersect on  $\ell$  and to the right of  $p$  ) ] and // horizontal segments  
 ( the intersection  $\bullet$  is not present in  $Q$  )

2. then

insert intersection  $\bullet$  as a new event into  $Q$

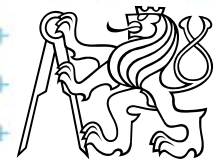


$s_l$  and  $s_r$  intersect below



$s_l$  and  $s_r$  intersect on  $\ell$

and to the right of  $p$



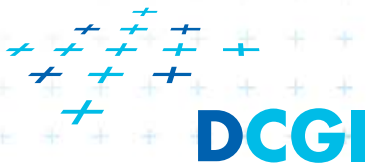
# Line segment intersections

---

- Memory  $O(I) = O(n^2)$  with duplicities in  $Q$   
or  $O(n)$  with duplicities in  $Q$  deleted
- Operational complexity
  - $n + I$  stops
  - $\log n$  each
  - $\Rightarrow O(I + n) \log n$  total
- The algorithm is by Bentley-Ottmann

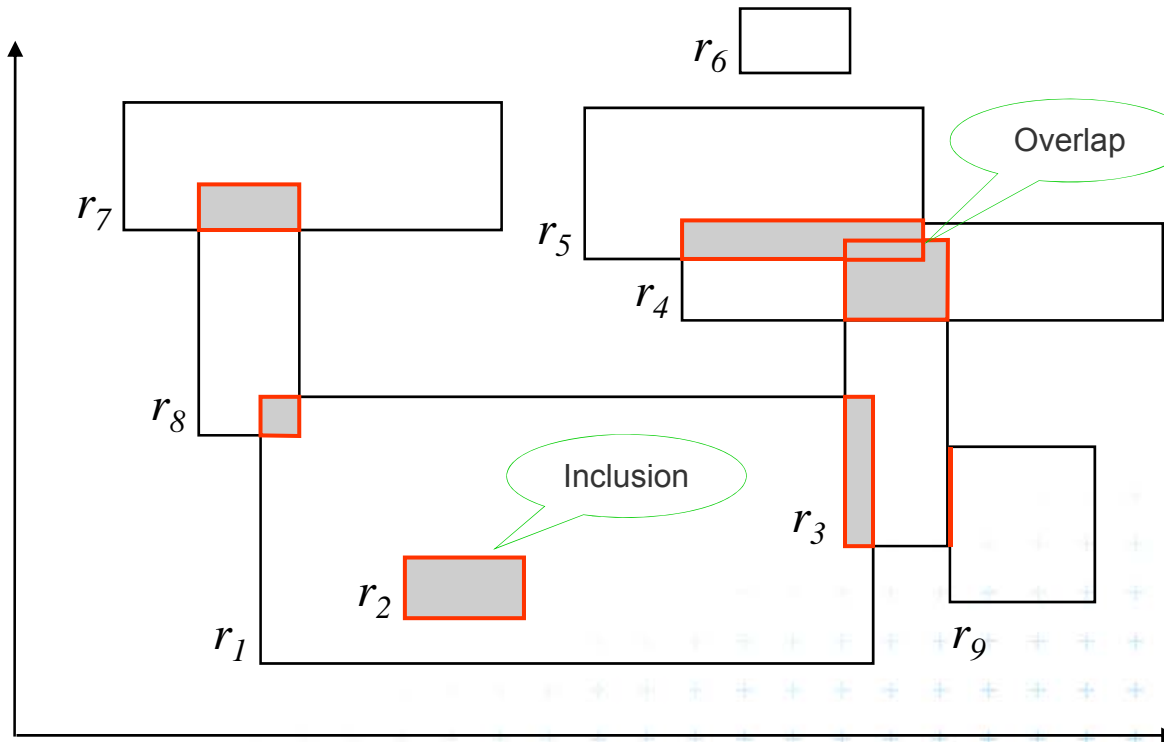
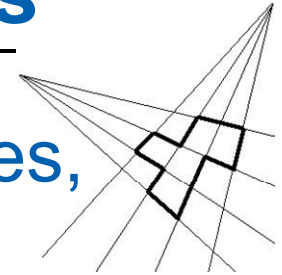
Bentley, J. L.; Ottmann, T. A. (1979), "Algorithms for reporting and counting geometric intersections", *IEEE Transactions on Computers* **C-28** (9): 643-647, doi:10.1109/TC.1979.1675432 .

See also [http://wopedia.mobi/en/Bentley%E2%80%93Ottmann\\_algorithm](http://wopedia.mobi/en/Bentley%E2%80%93Ottmann_algorithm)



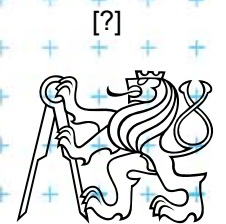
# Intersection of axis parallel rectangles

- Given the collection of  $n$  *isothetic* rectangles, report all intersecting parts



Alternate sides belong to two pencils of lines (trsy přímek) (often used with points in infinity = axis parallel)

Answer:  $(r_1, r_2) (r_1, r_3) (r_1, r_8) (r_3, r_4) (r_3, r_5) (r_3, r_9) (r_4, r_5) (r_7, r_8)$



# Brute force intersection

---

## Brute force algorithm

*Input:* set  $S$  of axis parallel rectangles

*Output:* pairs of intersected rectangles

1. For every pair  $(r_i, r_j)$  of rectangles  $\in S, i \neq j$
2.     if  $(r_i \cap r_j \neq \emptyset)$  then
3.         report  $(r_i, r_j)$

## Analysis

Preprocessing: None.

Query:  $O(N^2)$       $\binom{N}{2} = \frac{N(N-1)}{2} \in O(N^2)$ .

Storage:  $O(N)$



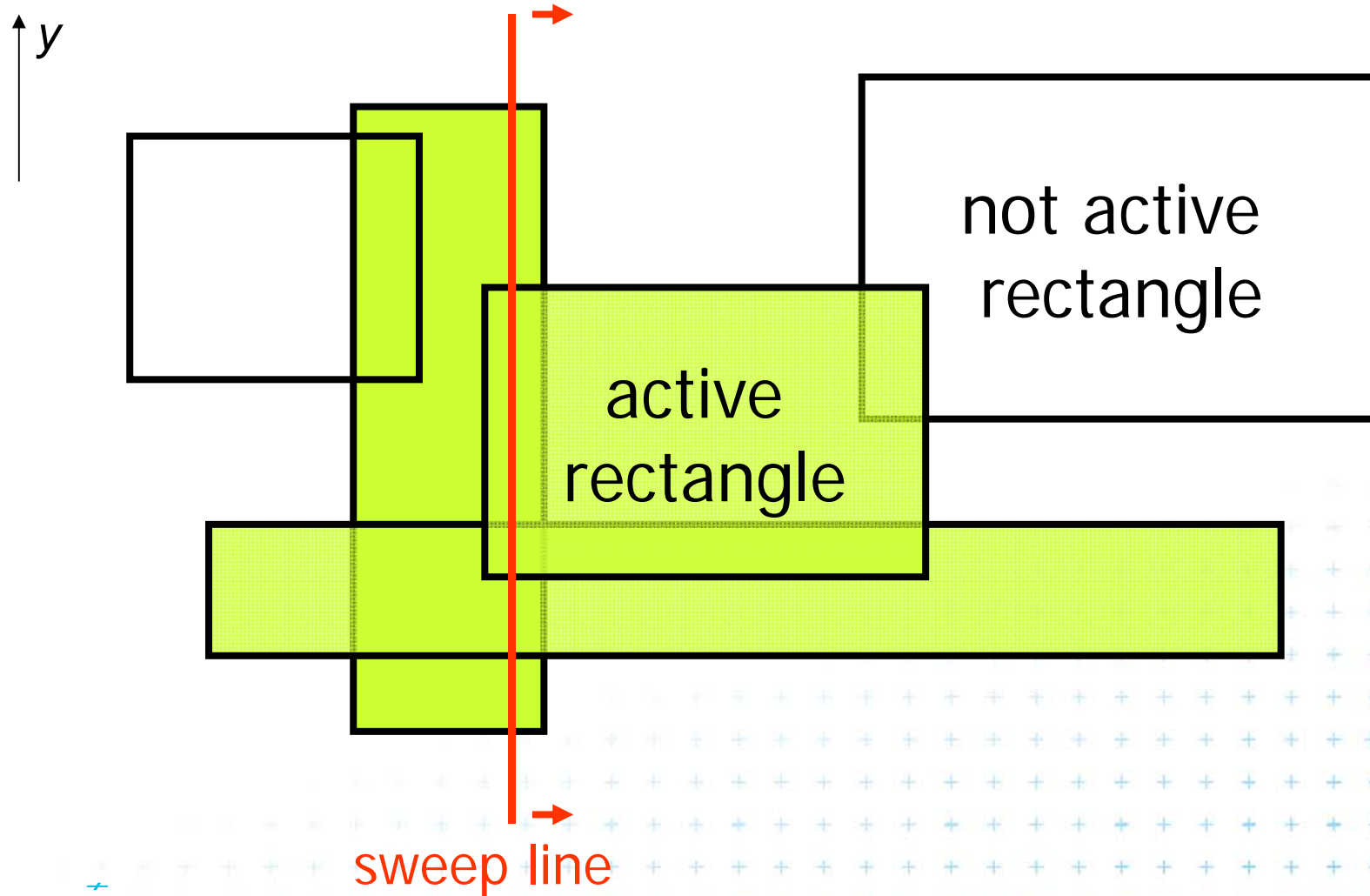
# Plane sweep intersection algorithm

---

- Vertical sweep line moves from left to right
- Stops at every x-coordinate of a rectangle (either its left side or its right side).
- **active rectangles** – a set  
= rectangles currently intersecting the sweep line
  - **left side** event of a rectangle  
=> the rectangle is **added** to the active set.
  - **right side**  
=> the rectangle is **deleted** from the active set.
- The active set used to detect rectangle intersection

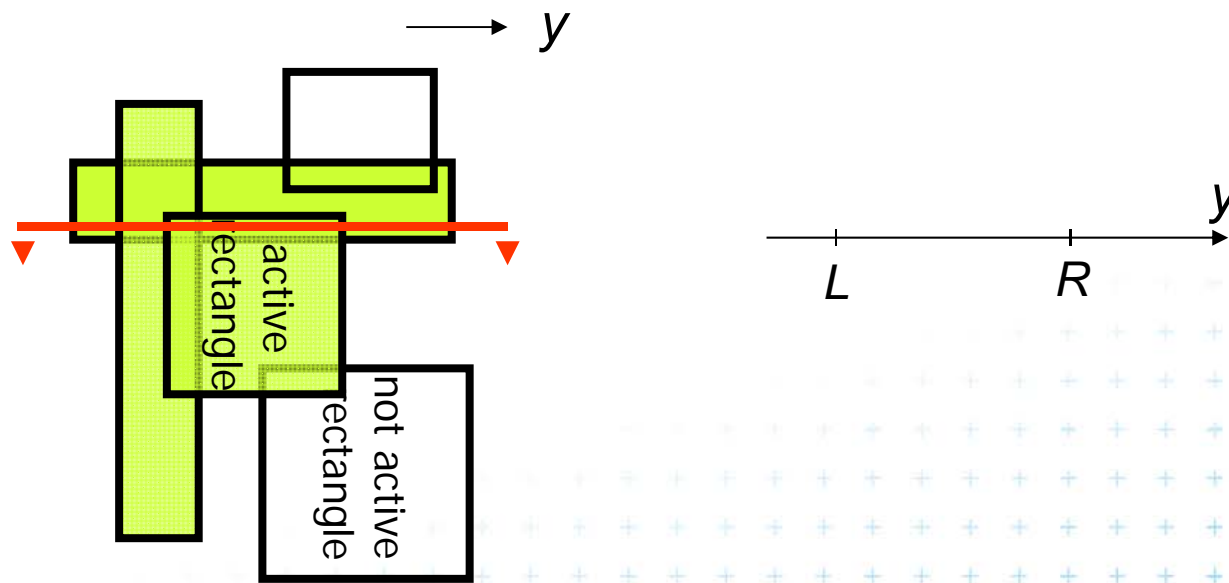


# Example rectangles and sweep line



# Interval tree as sweep line status structure

- Vertical sweep-line  $\Rightarrow$  Only  $y$ -coordinates along it
- Turn our view in slides  $90^\circ$  right
- Sweep line ( $y$ -axis) will be drawn as **horizontal**



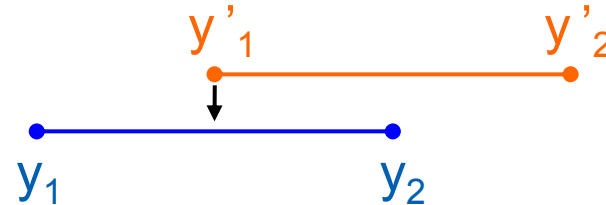
sweep line [Drtina]



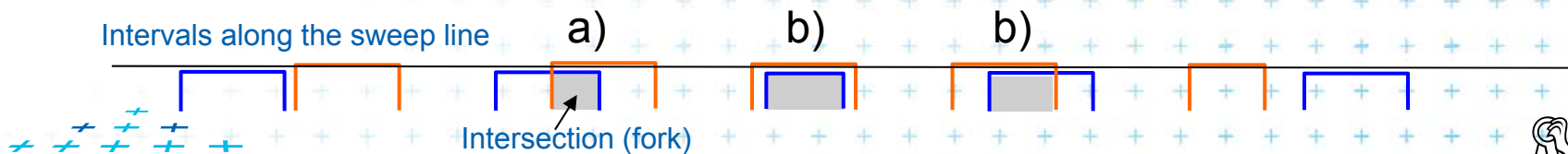
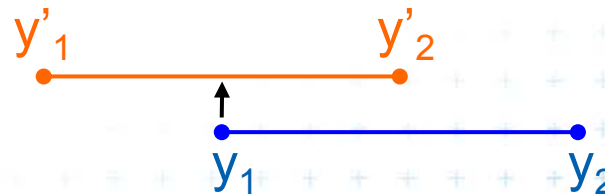
# Intersection test – between pair of intervals

- Given two intervals  $R = [y_1, y_2]$  and  $R' = [y'_1, y'_2]$  the condition  $R \cap R'$  is equivalent to one of these mutually exclusive conditions:

a)  $y_1 \leq y'_1 \leq y_2$



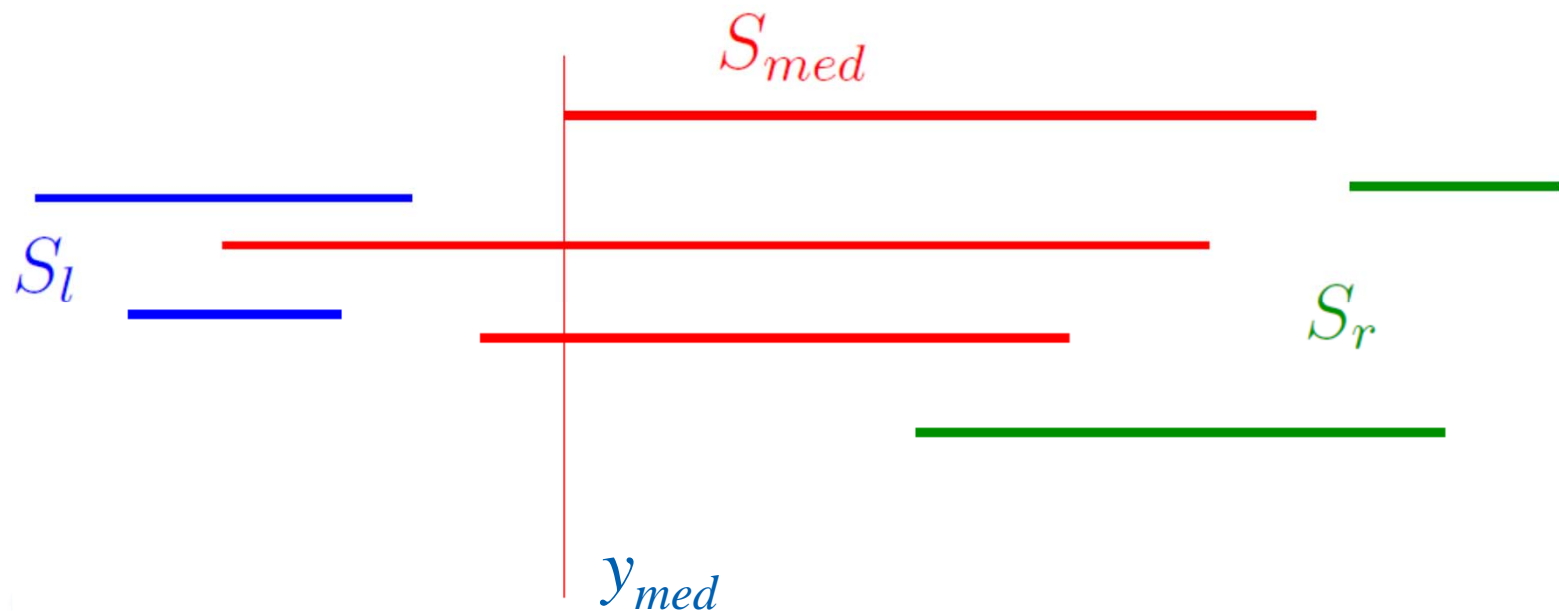
b)  $y'_1 \leq y_1 \leq y'_2$



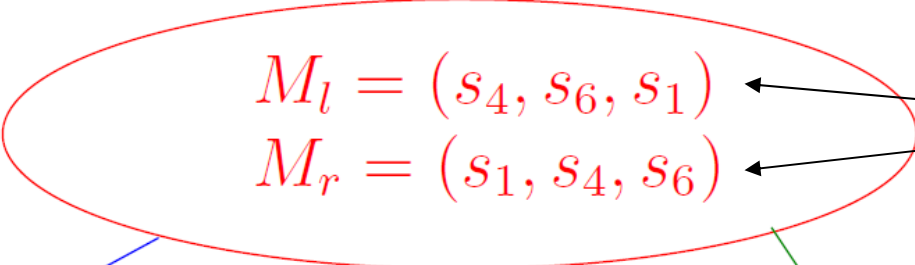
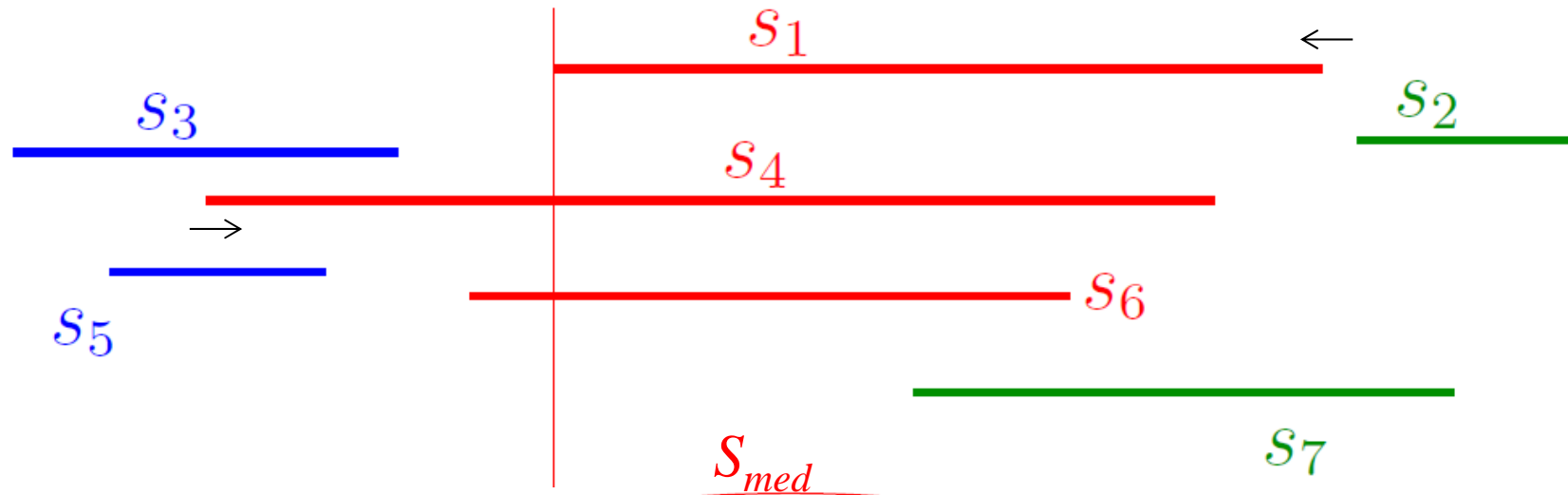


# Static interval tree – stores all end points

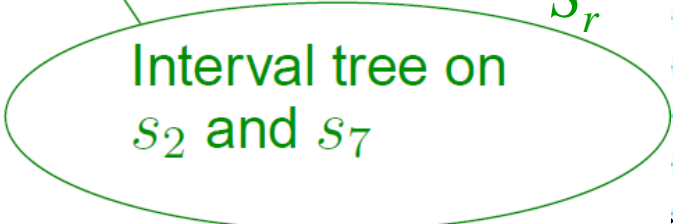
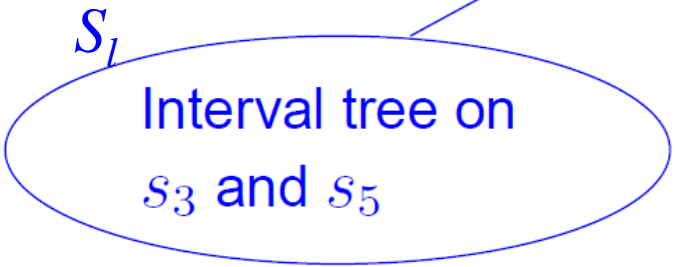
- Let  $v = y_{med}$  be the **median of end-points** of segments
- $S_l$  : segments of  $S$  that are completely to the **left of**  $y_{med}$
- $S_{med}$  : segments of  $S$  that **contain**  $y_{med}$
- $S_r$  : segments of  $S$  that are completely to the **right of**  $y_{med}$



# Static interval tree – Example

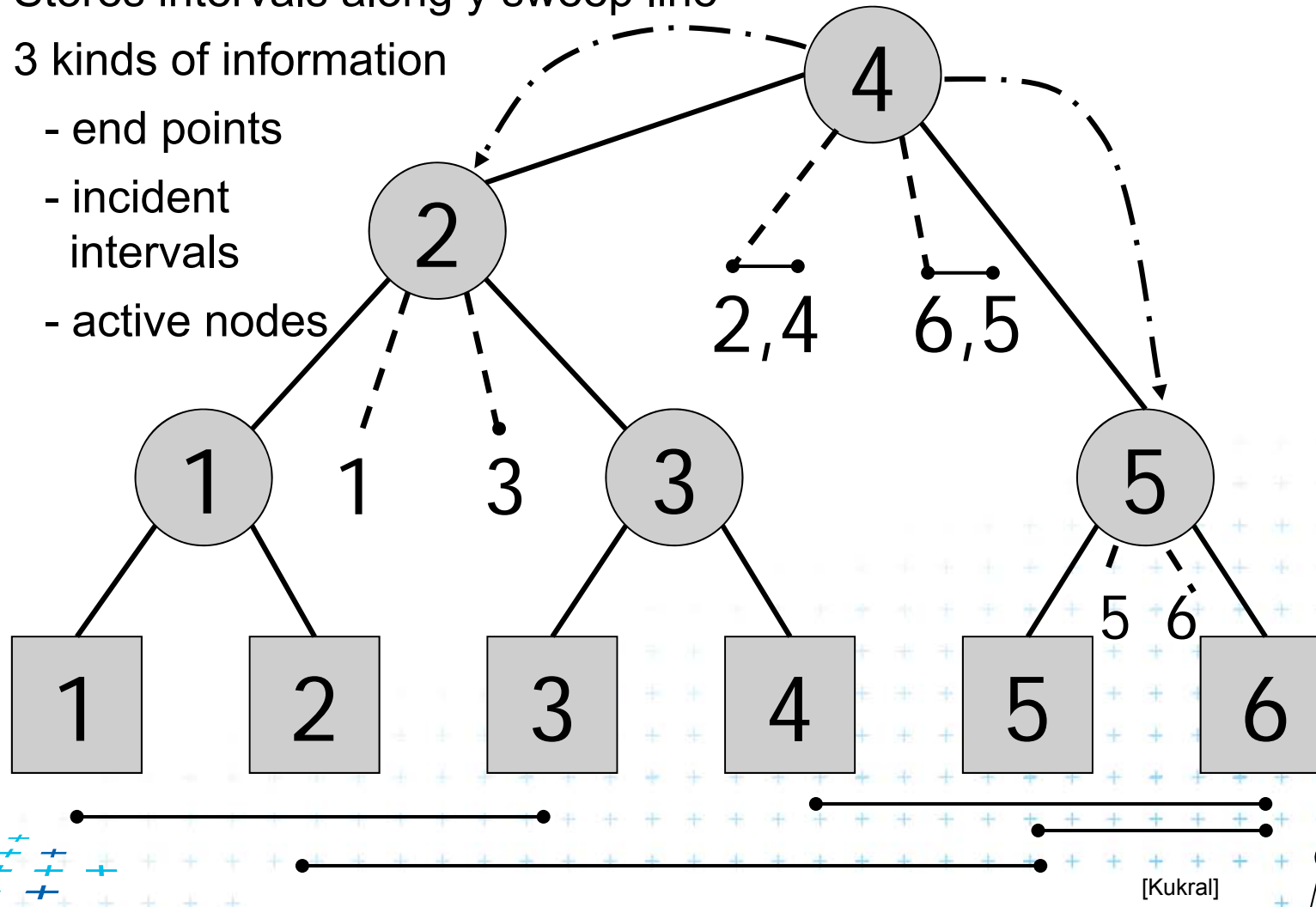


Left ends – ascending →  
 Right ends – descending ←



# Static interval tree [Edelsbrunner80]

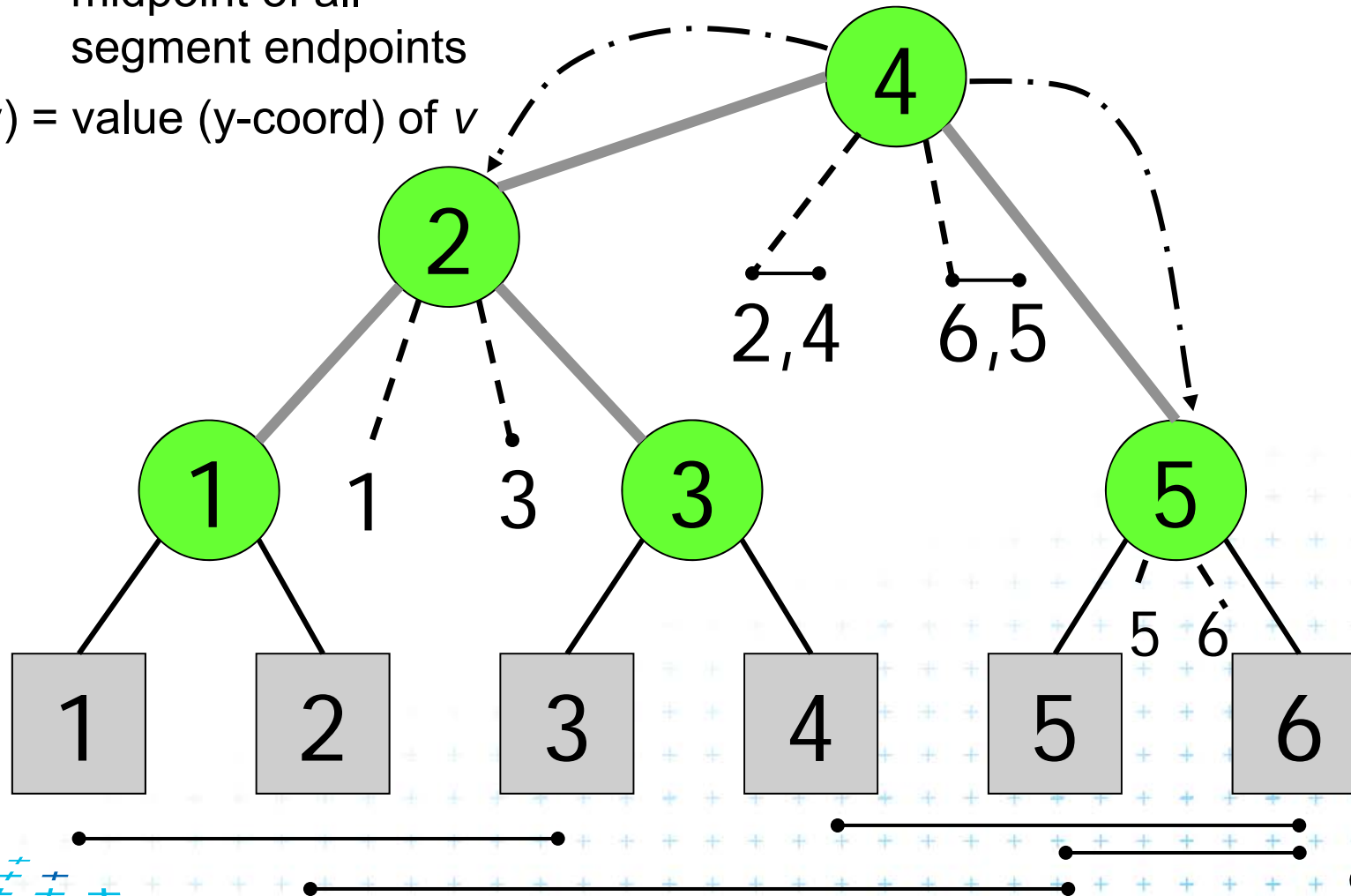
- Stores intervals along y sweep line
- 3 kinds of information
  - end points
  - incident intervals
  - active nodes



# Primary structure – static tree for endpoints

$v$  = midpoint of all  
segment endpoints

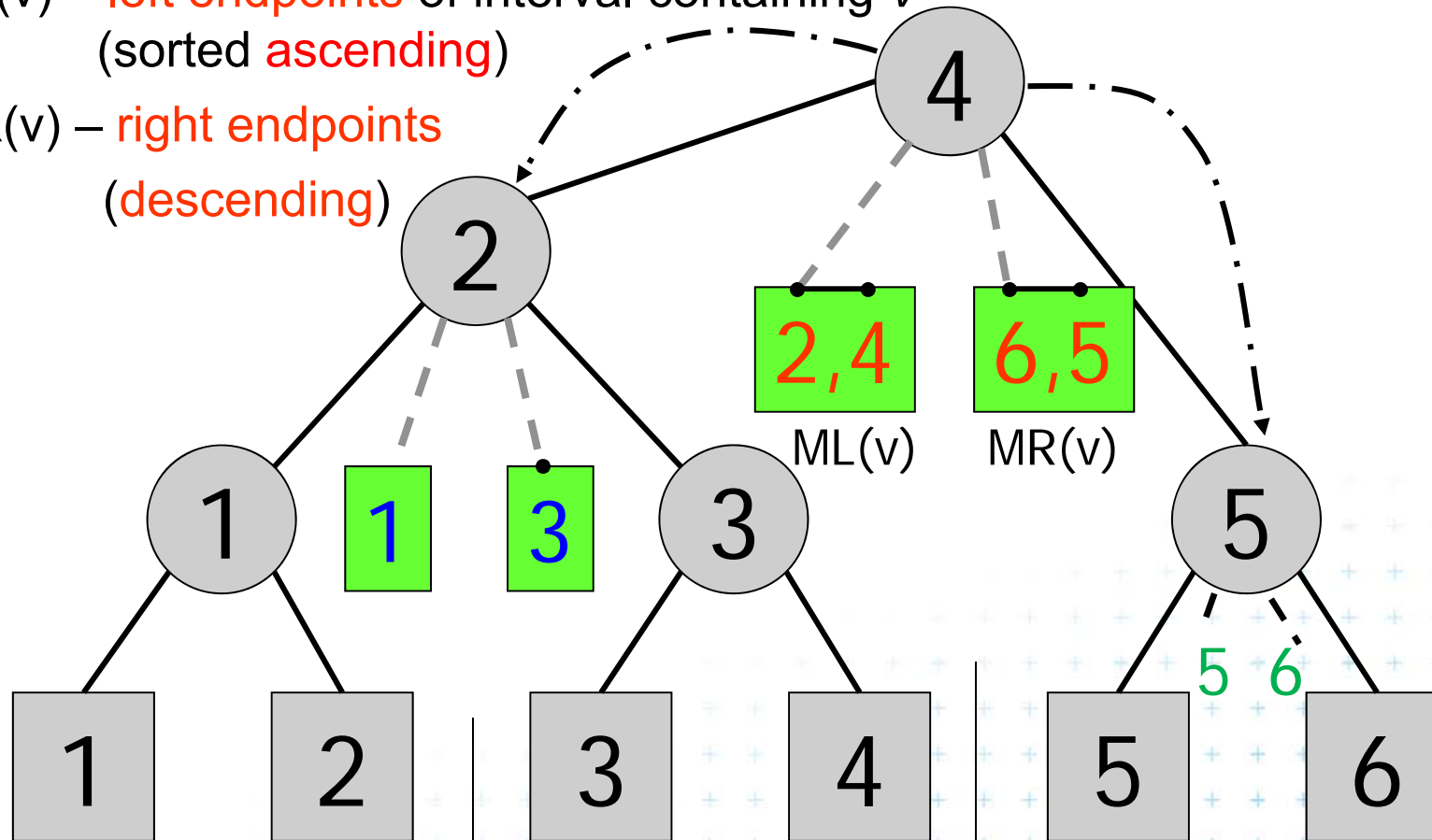
$H(v)$  = value (y-coord) of  $v$



# Secondary lists of incident interval end-pts.

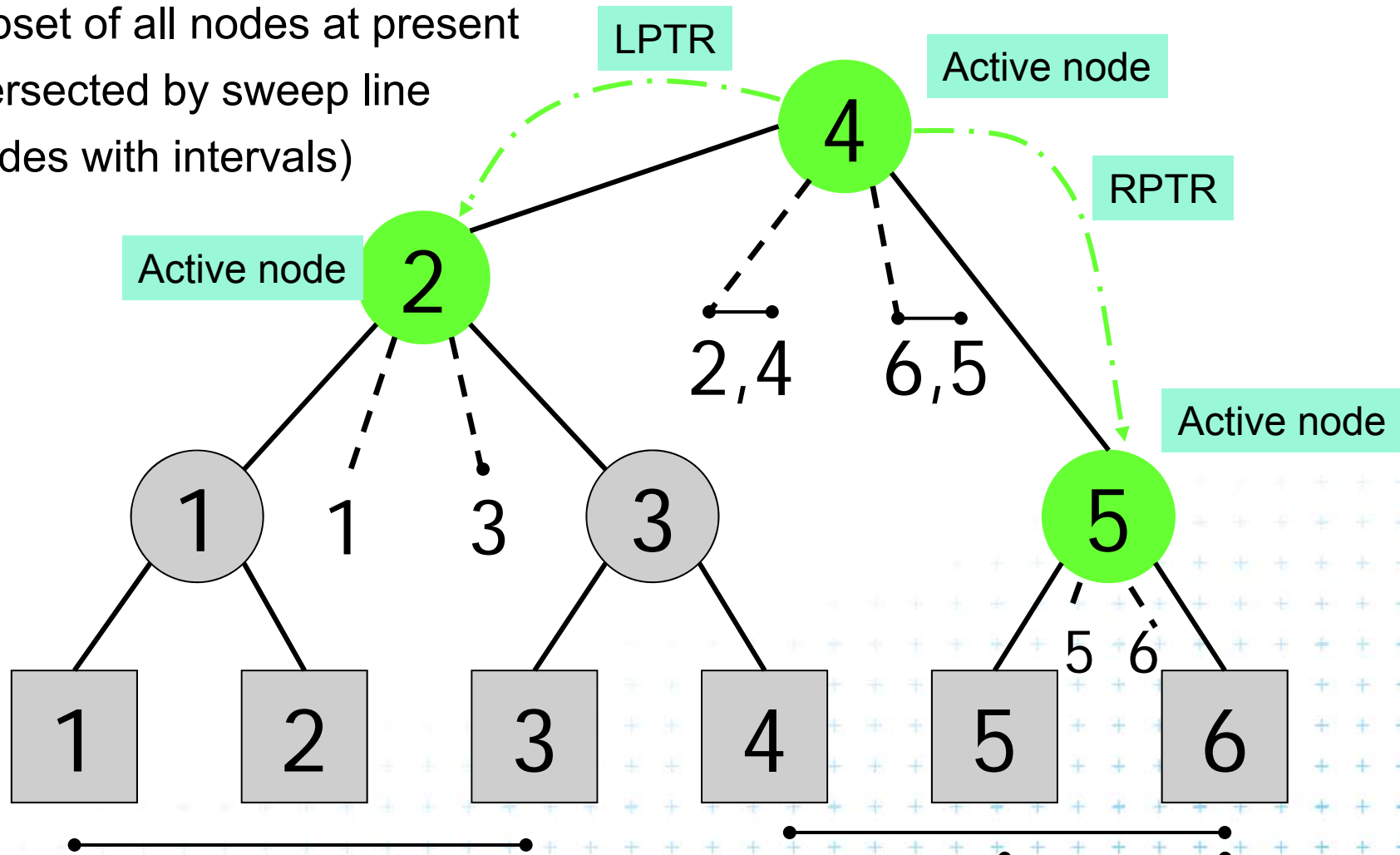
ML( $v$ ) – left endpoints of interval containing  $v$   
(sorted ascending)

MR( $v$ ) – right endpoints  
(descending)



# Active nodes – intersected by the sweep line

Subset of all nodes at present intersected by sweep line (nodes with intervals)





# Query = sweep and report intersections

---

## RectangleIntersections( S )

Input: Set S of rectangles

Output: Intersected rectangle pairs

1. Preprocess( S ) // create the interval tree  $T$  (for y-coords)  
// and event queue  $Q$  (for x-coords)
2. while (  $Q \neq \emptyset$  ) do
3. Get next entry  $(x_i, y_{il}, y_{ir}, t)$  from  $Q$  //  $t \in \{ left | right \}$
4. if (  $t = left$  ) // left edge 
5. a) QueryInterval (  $y_{il}, y_{ir}, root(T)$  ) // report intersections
6. b) InsertInterval (  $y_{il}, y_{ir}, root(T)$  ) // insert new interval
7. else // right edge 
8. c) DeleteInterval (  $y_{il}, y_{ir}, root(T)$  )



# Preprocessing

---

## Preprocess( S )

*Input:* Set  $S$  of rectangles

*Output:* Primary structure of the interval tree  $T$  and the event queue  $Q$

1.  $T = \text{PrimaryTree}(S)$  // Construct the static primary structure  
// of the interval tree -> sweep line STATUS  $T$
2. // Init event queue  $Q$  with vertical rectangle edges in ascending order  $\sim x$   
// Put the left edges with the same  $x$  ahead of right ones
3. for  $i = 1$  to  $n$
4.     insert( (  $x_{il}, y_{il}, y_{ir}, \text{left}$  ),  $Q$ )     // left edges of  $i$ -th rectangle
5.     insert( (  $x_{ir}, y_{il}, y_{ir}, \text{right}$  ),  $Q$ )     // right edges





# Interval tree – primary structure construction

---

**PrimaryTree(S)** // only the y-tree structure, without intervals

*Input:* Set  $S$  of rectangles

*Output:* Primary structure of an interval tree  $T$

1.  $S_y = \text{Sort endpoints of all segments in } S \text{ according to } y\text{-coordinate}$
2.  $T = \text{BST}( S_y )$
3. **return**  $T$

**BST(  $S_y$  )**

1. **if**(  $|S_y| = 0$  ) **return** null
2.  $yMed = \text{median of } S_y$
3.  $L = \text{endpoints } p_y \leq yMed$
4.  $R = \text{endpoints } p_y > yMed$
5.  $t = \text{new IntervalTreeNode}( yMed )$
6.  $t.\text{left} = \text{BST}(L)$
7.  $t.\text{right} = \text{BST}(R)$
8. **return**  $t$



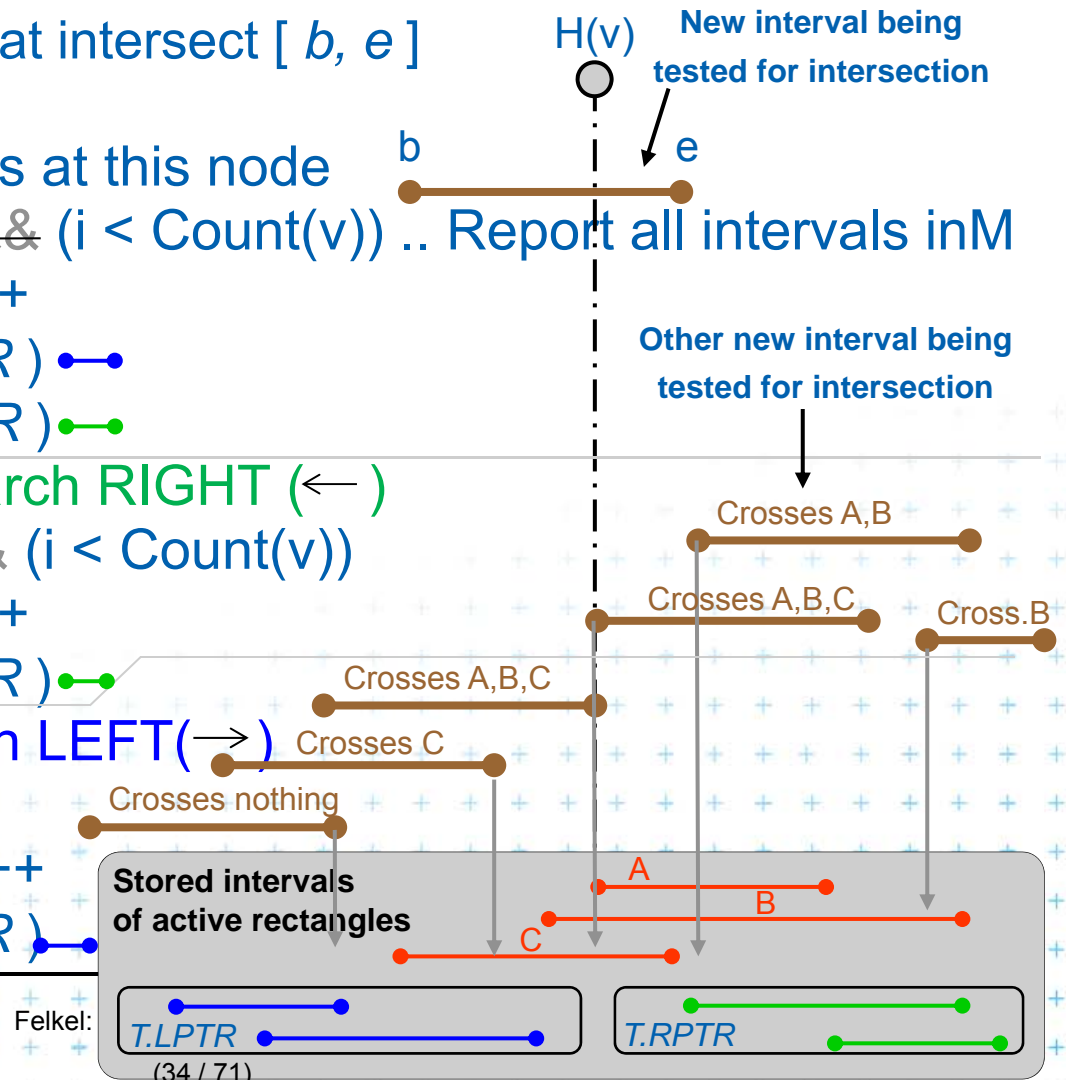
# Interval tree – search the intersections

## QueryInterval ( $b, e, T$ )

Input: Interval of the edge and current tree  $T$

Output: Report the rectangles that intersect  $[b, e]$

1. if(  $T = \text{null}$  ) return
2.  $i=0$ ; if(  $b < H(v) < e$  ) // forks at this node
3. while (  $MR(v).[i] \geq b$  ) && (  $i < \text{Count}(v)$  ) .. Report all intervals in  $M$
4. ReportIntersection;  $i++$
5. QueryInterval(  $b, e, T.LPTR$  )
6. QueryInterval(  $b, e, T.RPTR$  )
7. else if (  $H(v) \leq b < e$  ) // search RIGHT ( $\leftarrow$ )
8. while (  $MR(v).[i] \geq b$  ) && (  $i < \text{Count}(v)$  )
9. ReportIntersection;  $i++$
10. QueryInterval(  $b, e, T.RPTR$  )
11. else //  $b < e \leq H(v)$  //search LEFT ( $\rightarrow$ )
12. while (  $ML(v).[i] \leq e$  )
13. ReportIntersection;  $i++$
14. QueryInterval(  $b, e, T.LPTR$  )



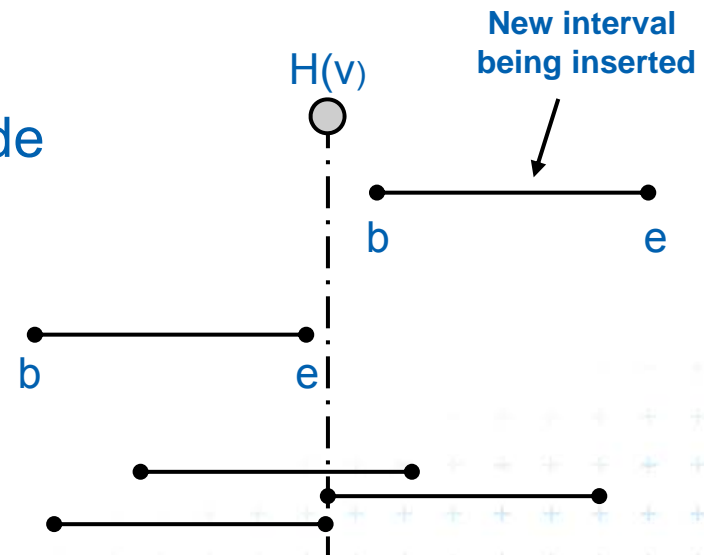
# Interval tree - interval insertion

## InsertInterval ( $b, e, T$ )

Input: Interval  $[b,e]$  and interval tree  $T$

Output:  $T$  after insertion of the interval

1.  $v = \text{root}(T)$
2. **while**(  $v \neq \text{null}$  ) // find the fork node
3.     **if** (  $H(v) < b < e$  )
4.          $v = v.\text{right}$  // continue right
5.     **else if** (  $b < e < H(v)$  )
6.          $v = v.\text{left}$  // continue left
7.     **else** //  $b \leq H(v) \leq e$  // insert interval
8.         set  $v$  node to *active*
9.         connect LPTR resp. RPTR to its parent
10.         insert  $[b,e]$  into list  $ML(v)$  – sorted in ascending order of  $b$ 's
11.         insert  $[b,e]$  into list  $MR(v)$  – sorted in descending order of  $e$ 's
12.         break
13. **endwhile**
14. **return**  $T$



+

DCGI

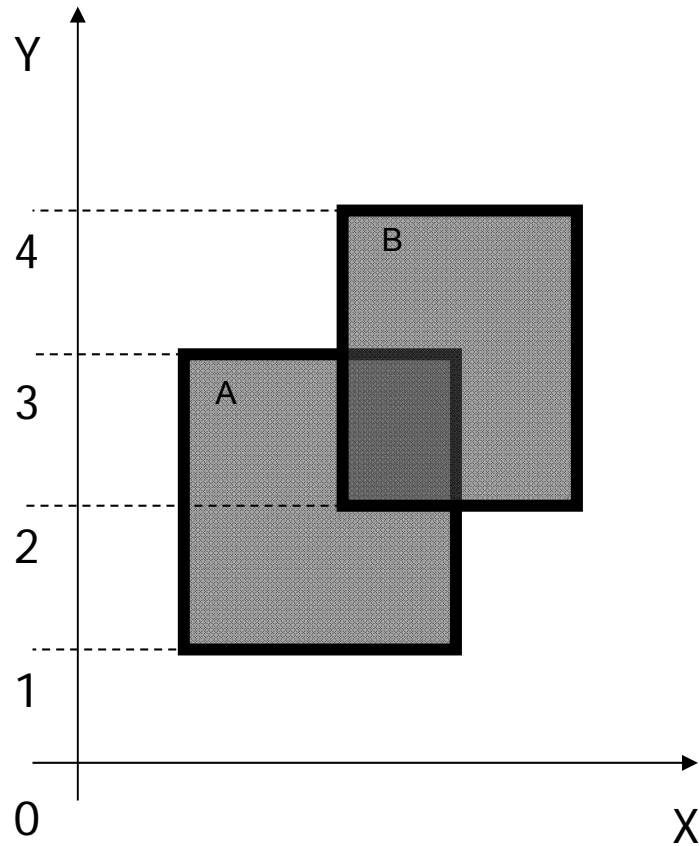


# Example 1

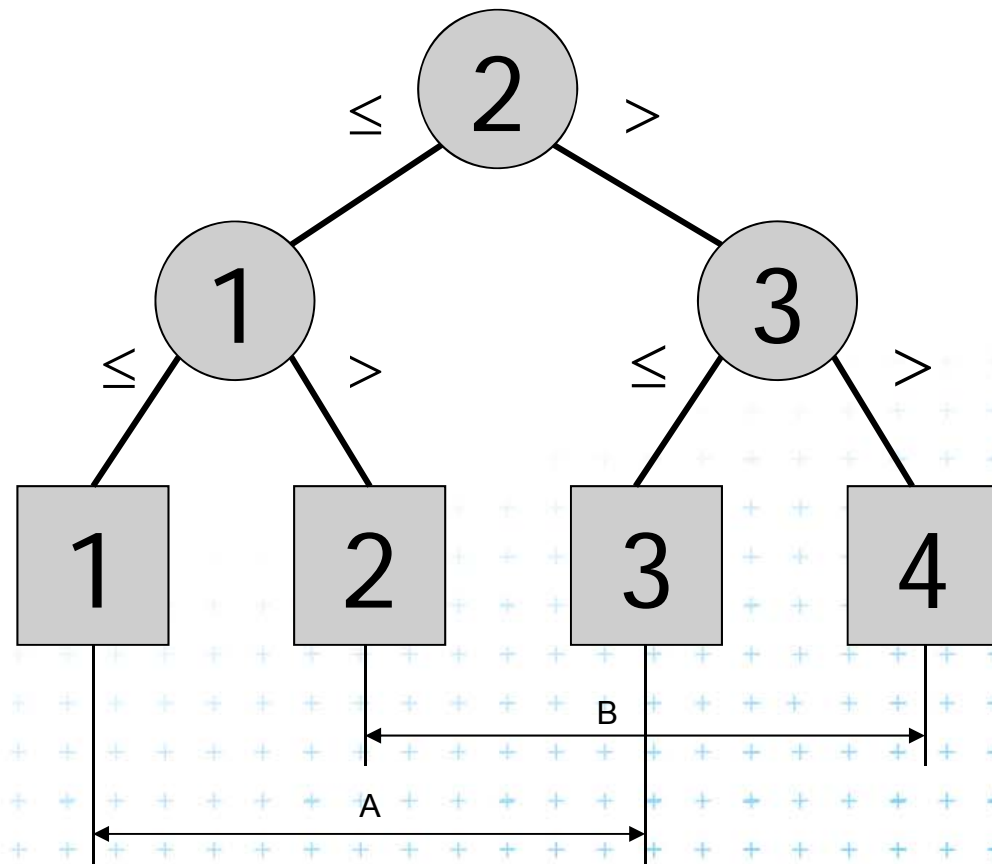
---



# Example 1 – static tree on endpoints



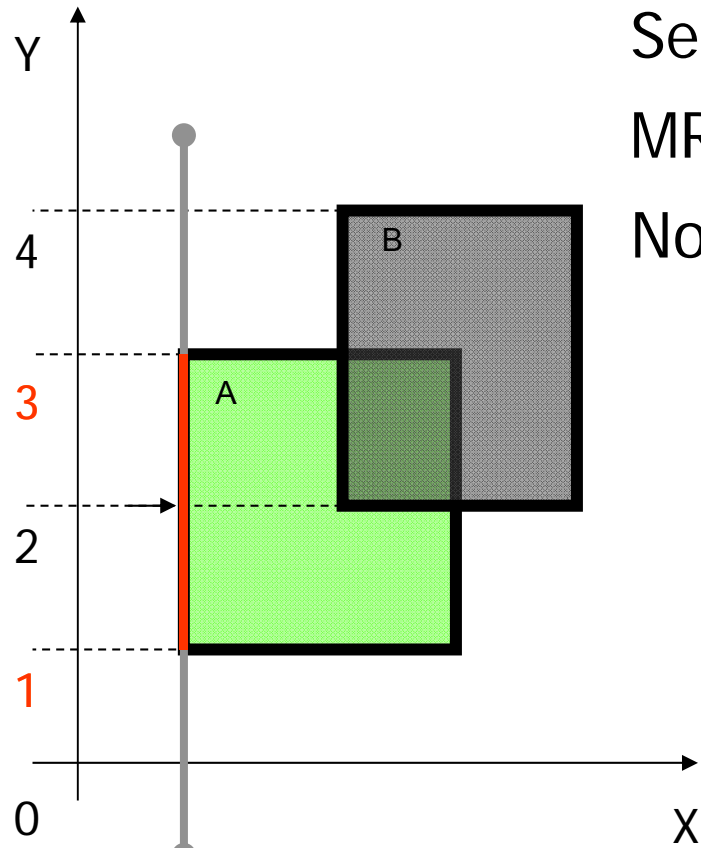
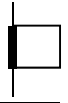
$H(v)$  – value of node  $v$






[Drtina]



# Interval insertion [1,3] a) Query Interval



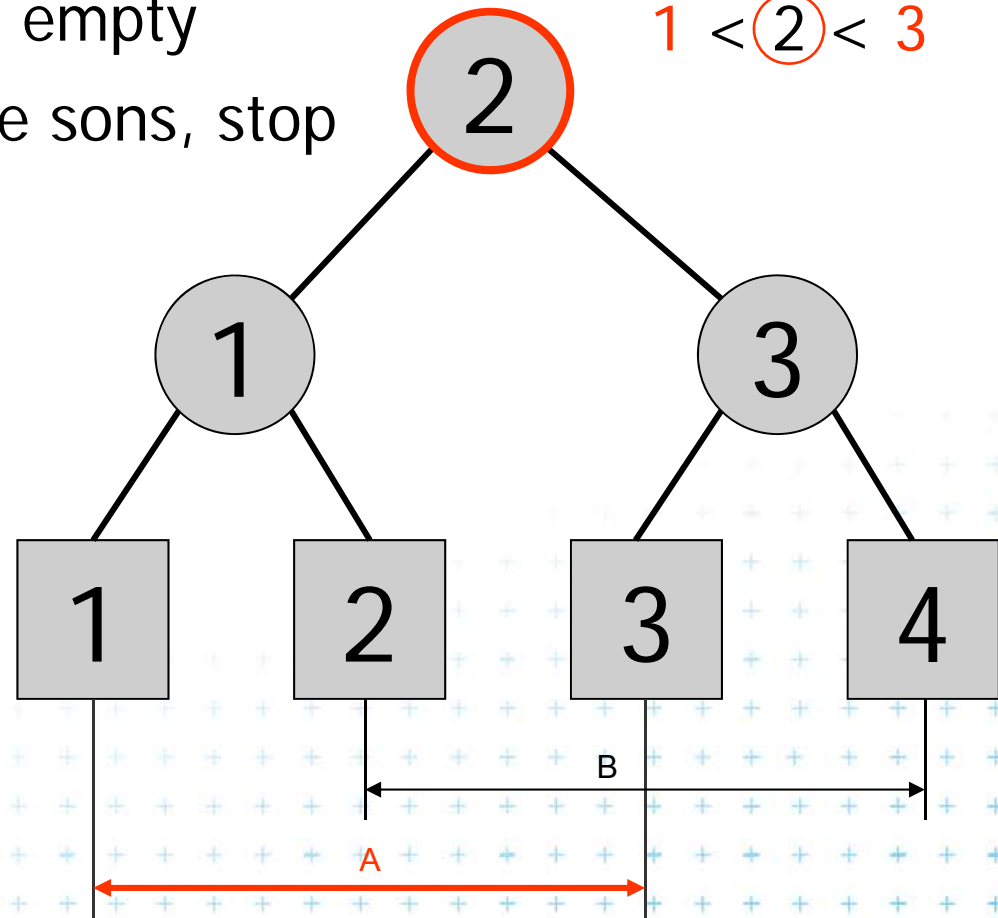
-  Active rectangle
-  Current node
-  Active node

Search  $MR(v)$  or  $ML(v)$ :  $\leftarrow b < H(v) < e$

$MR(v)$  is empty

No active sons, stop

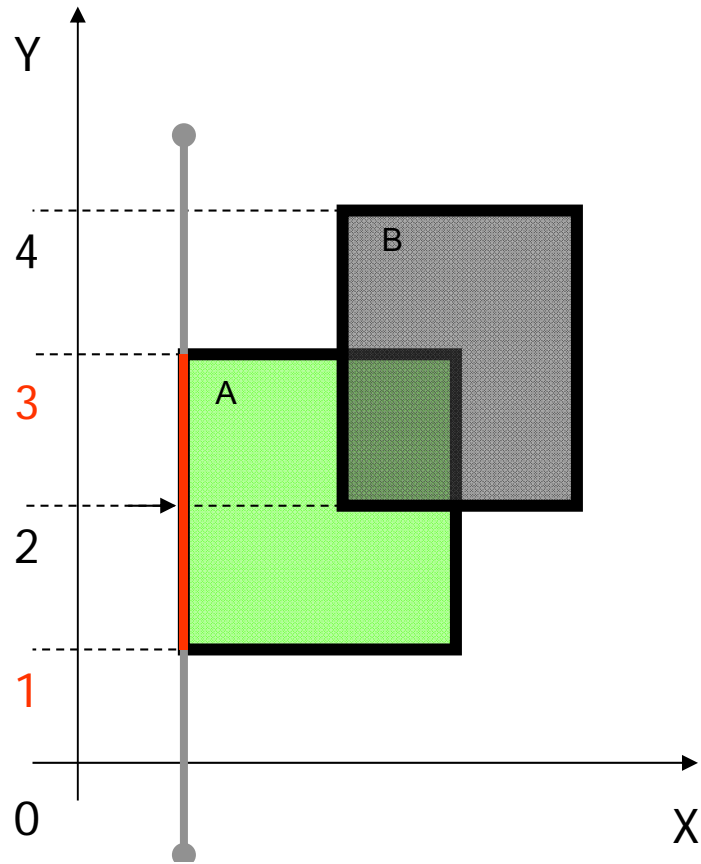
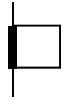
$1 < \textcircled{2} < 3$






[Drtina]



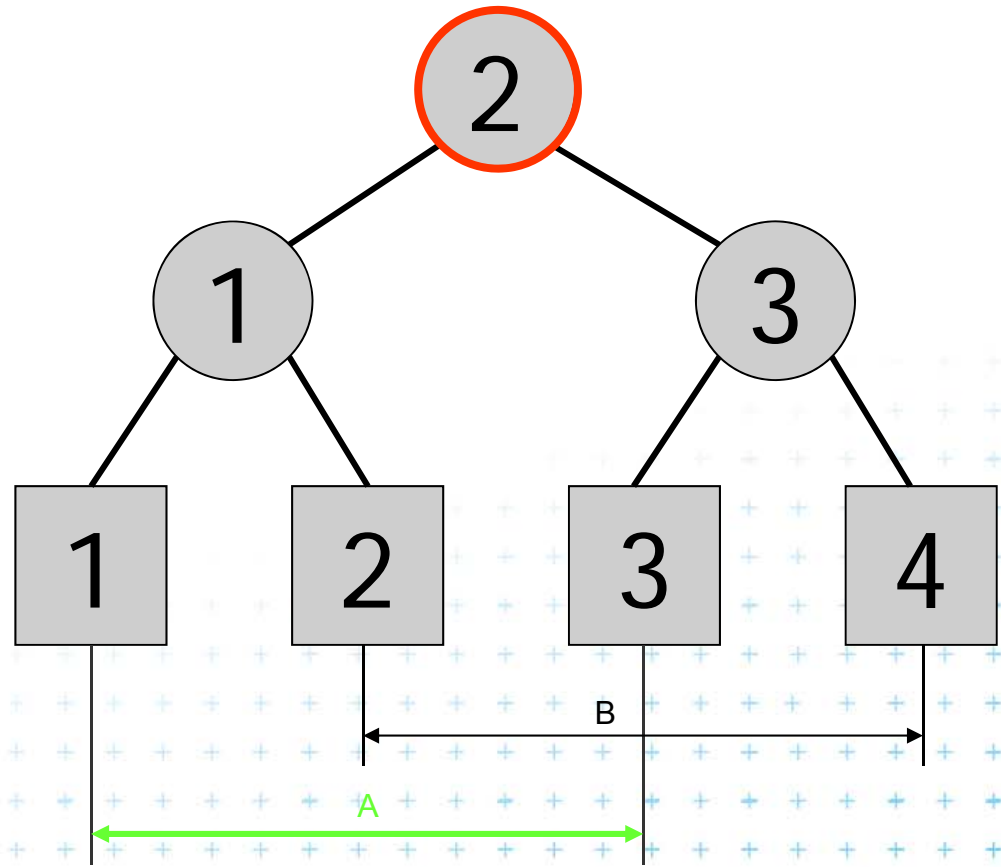
# Interval insertion [1,3]    b) Insert Interval



-  Active rectangle
-  Current node
-  Active node

$$b \leq H(v) \leq e$$

$$? 1 \leq \textcircled{2} \leq 3 ?$$

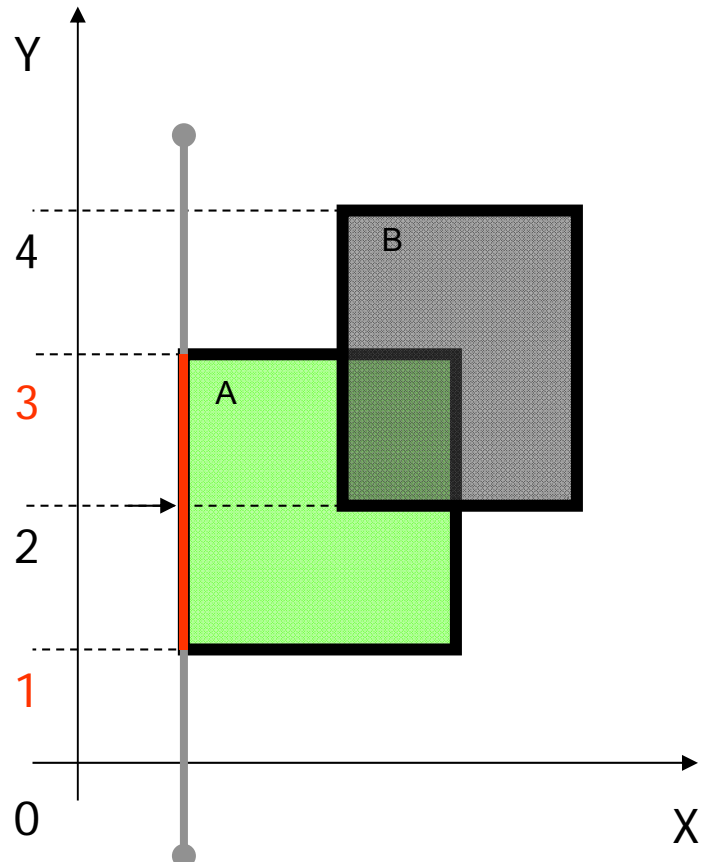


[Drtina]



# Interval insertion [1,3]

## b) Insert Interval

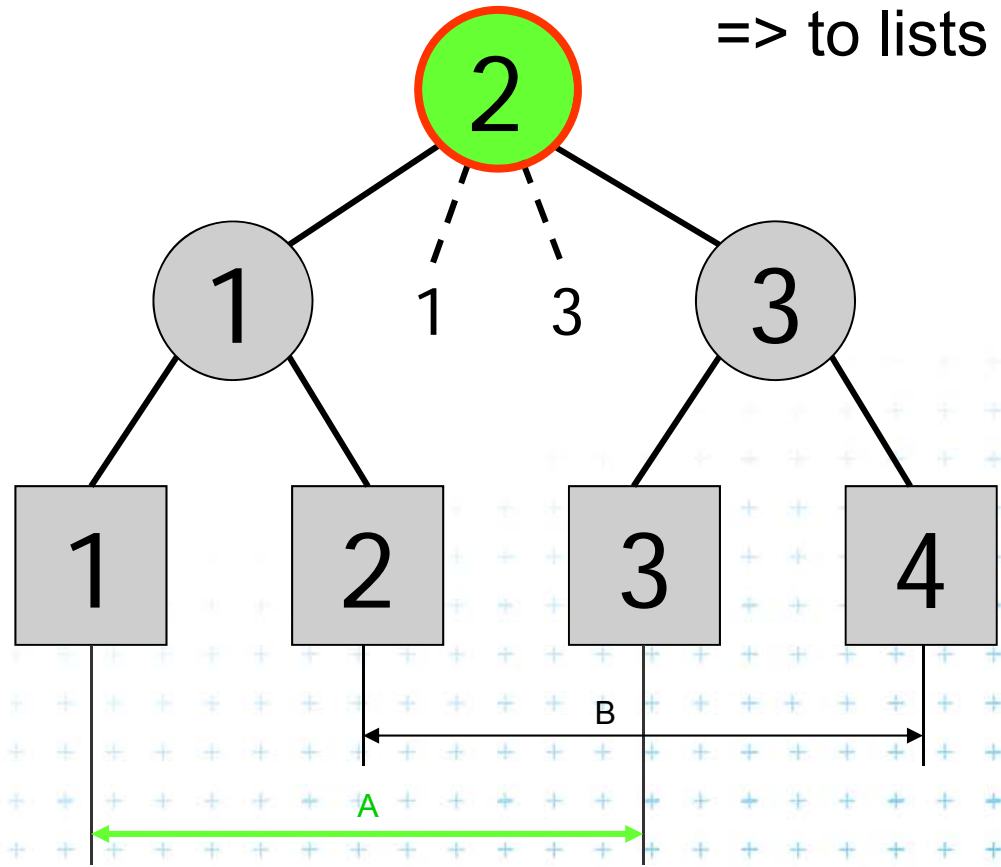


- Active rectangle
- Current node
- Active node

$$b \leq H(v) \leq e$$

$$1 \leq \textcircled{2} \leq 3$$

fork  
=> to lists

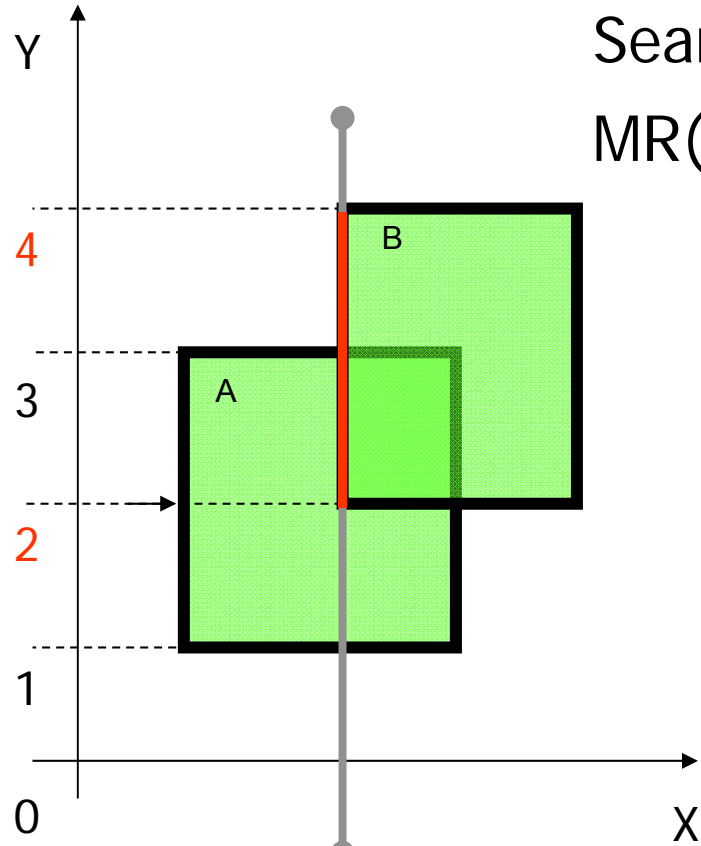
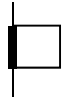





[Drtina]





# Interval insertion [2,4] a) Query Interval



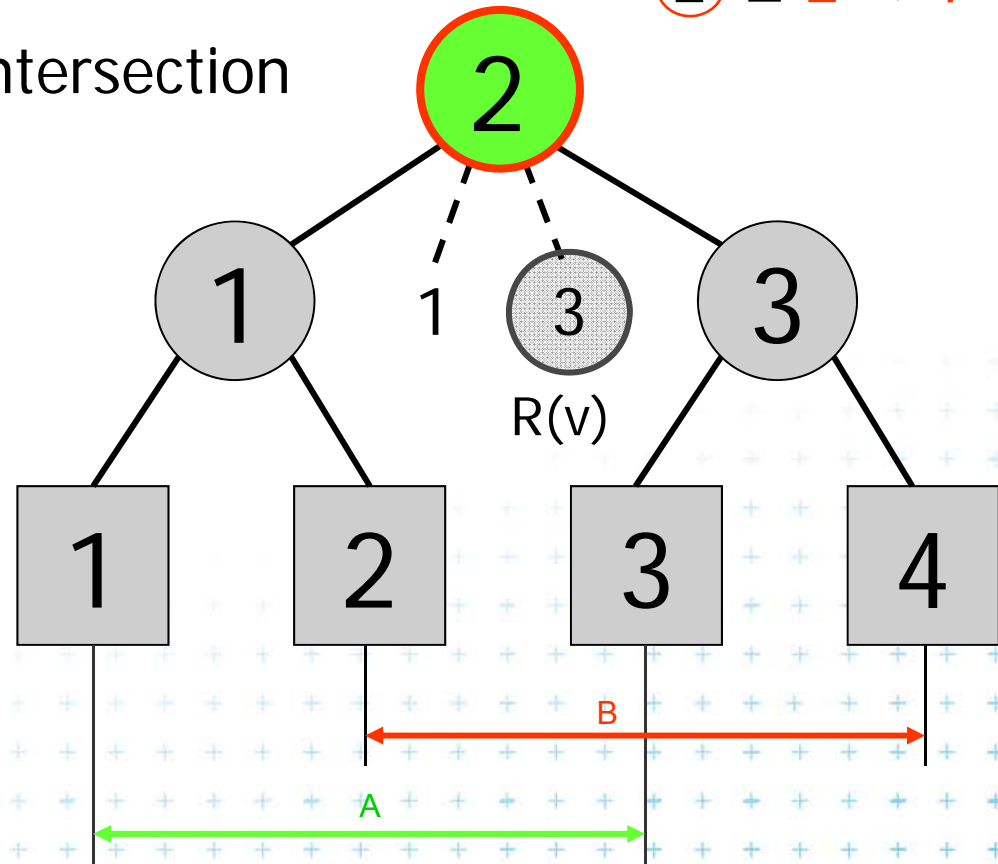
-  Active rectangle
-  Current node
-  Active node

Search MR(v) only:  $\leftarrow H(v) \leq b < e$

MR(v)[1] = 3  $\geq$  2?

$\textcircled{2} \leq 2 < 4$

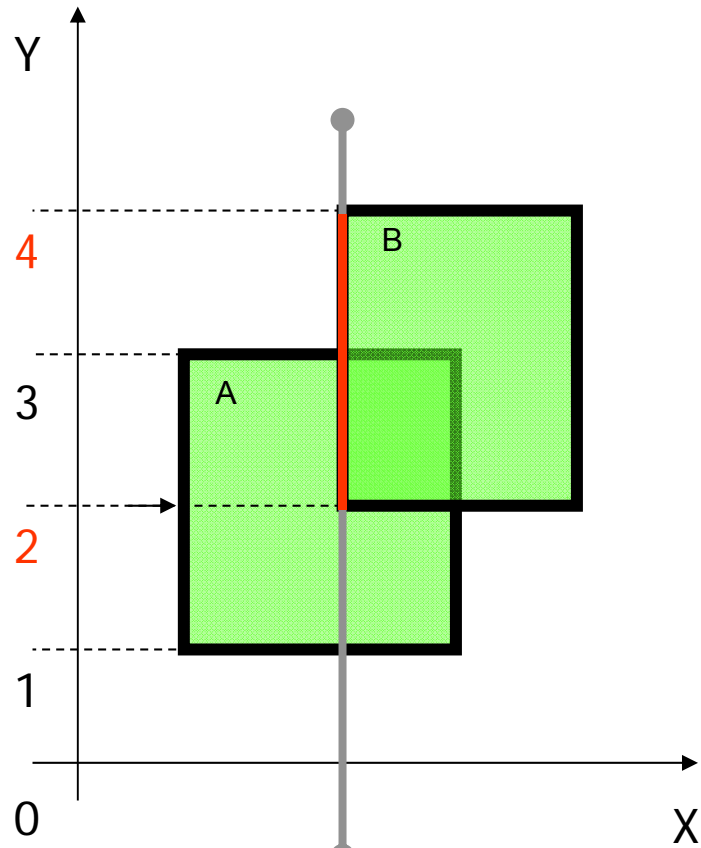
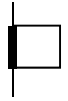
=> intersection






[Drtina]



# Interval insertion [2,4]    b) Insert Interval

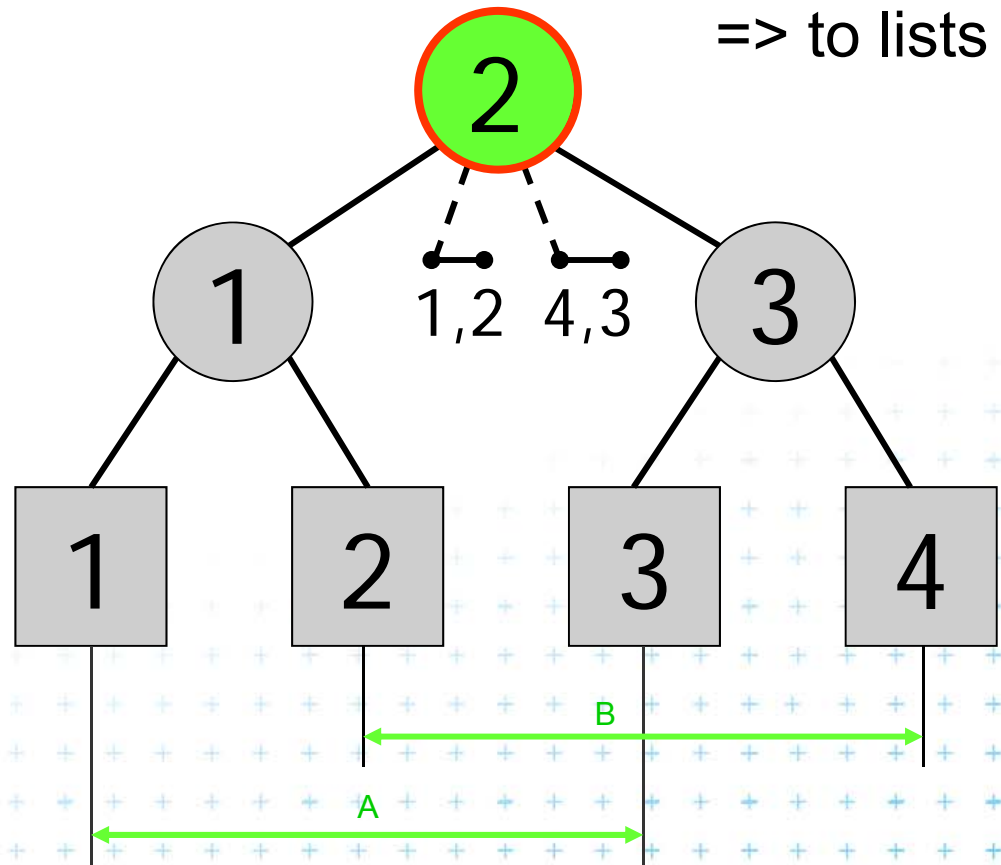


-  Active rectangle
-  Current node
-  Active node

$$b \leq H(v) \leq e$$

$$2 \leq \textcircled{2} \leq 4$$

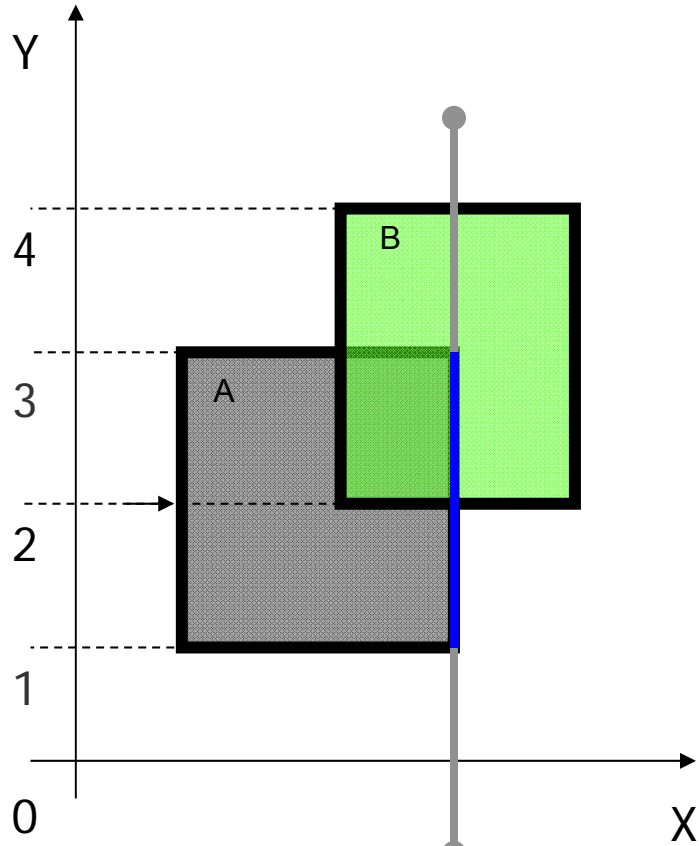
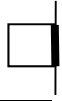
fork  
=> to lists






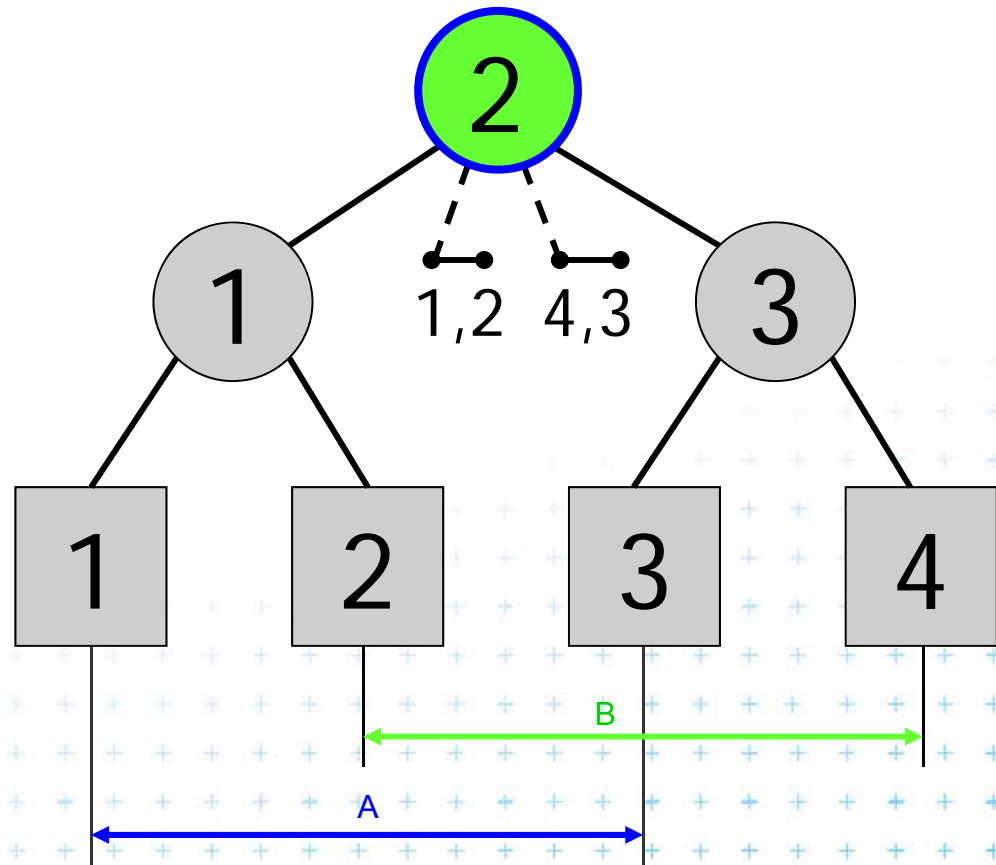
[Drtina]



# Interval delete [1,3]



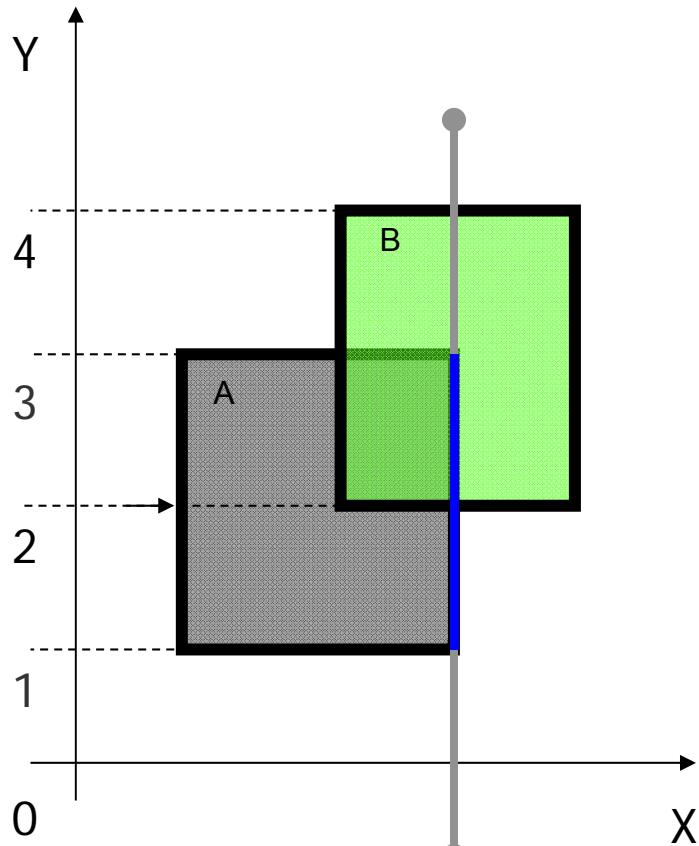
-  Active rectangle
-  Current node
-  Active node






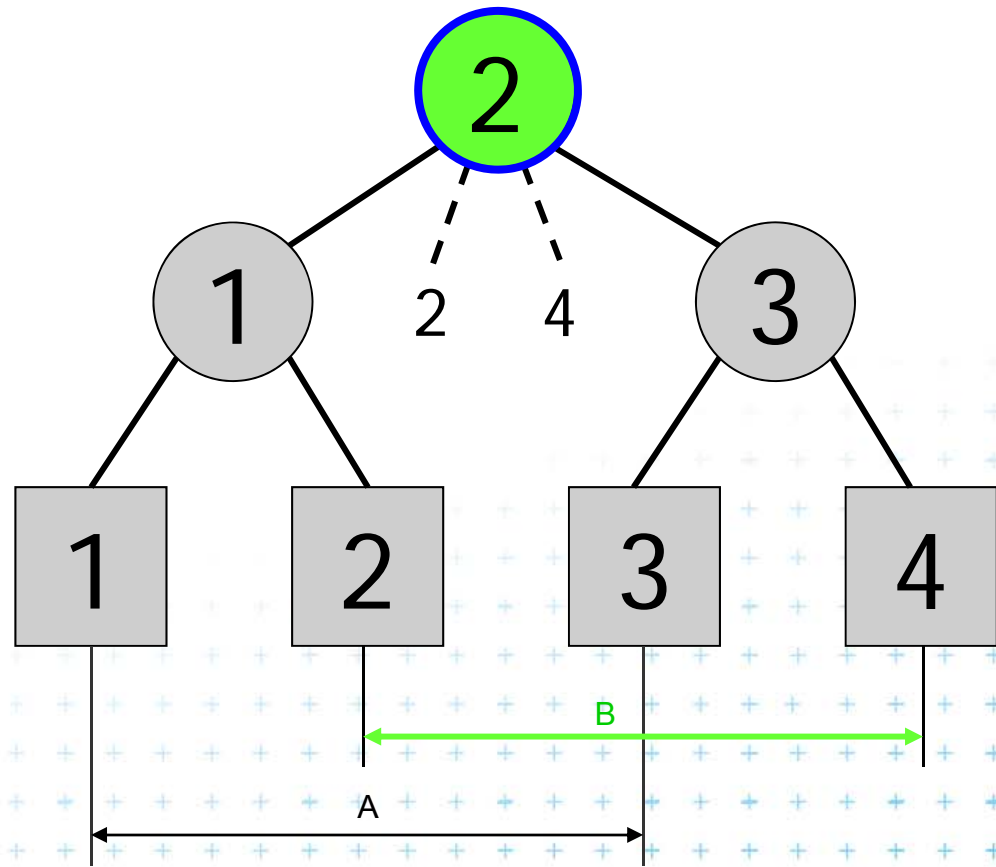
[Drtina]



# Interval delete [1,3]



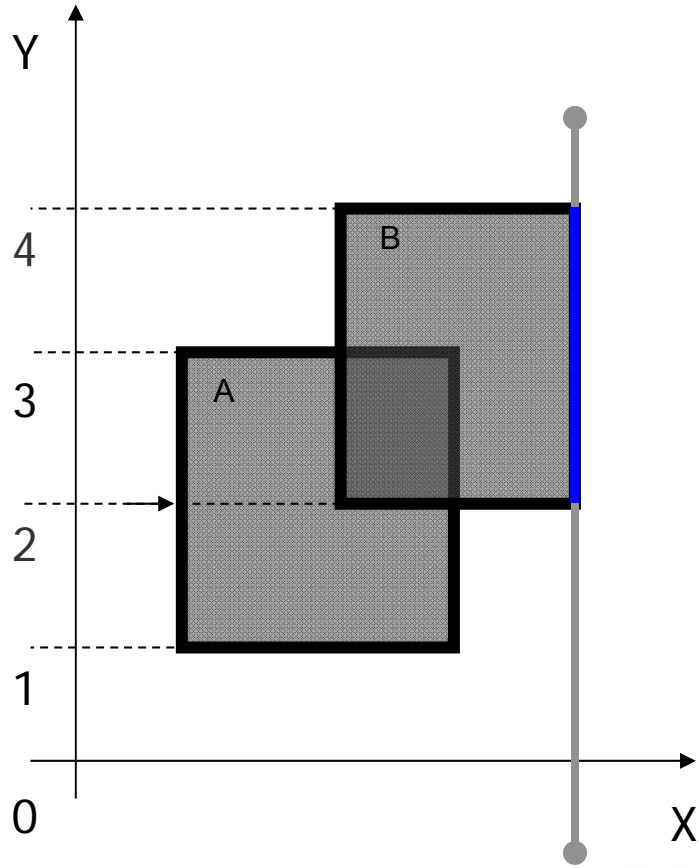
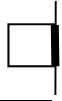
-  Active rectangle
-  Current node
-  Active node






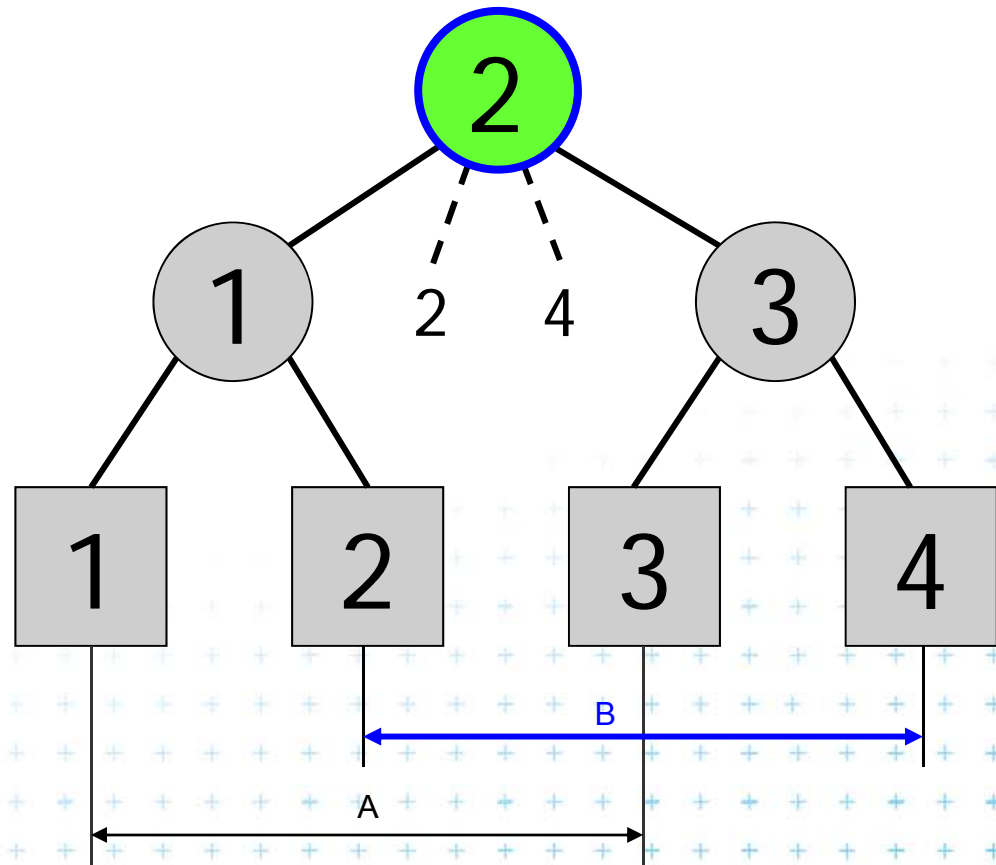
[Drtina]



# Interval delete [2,4]



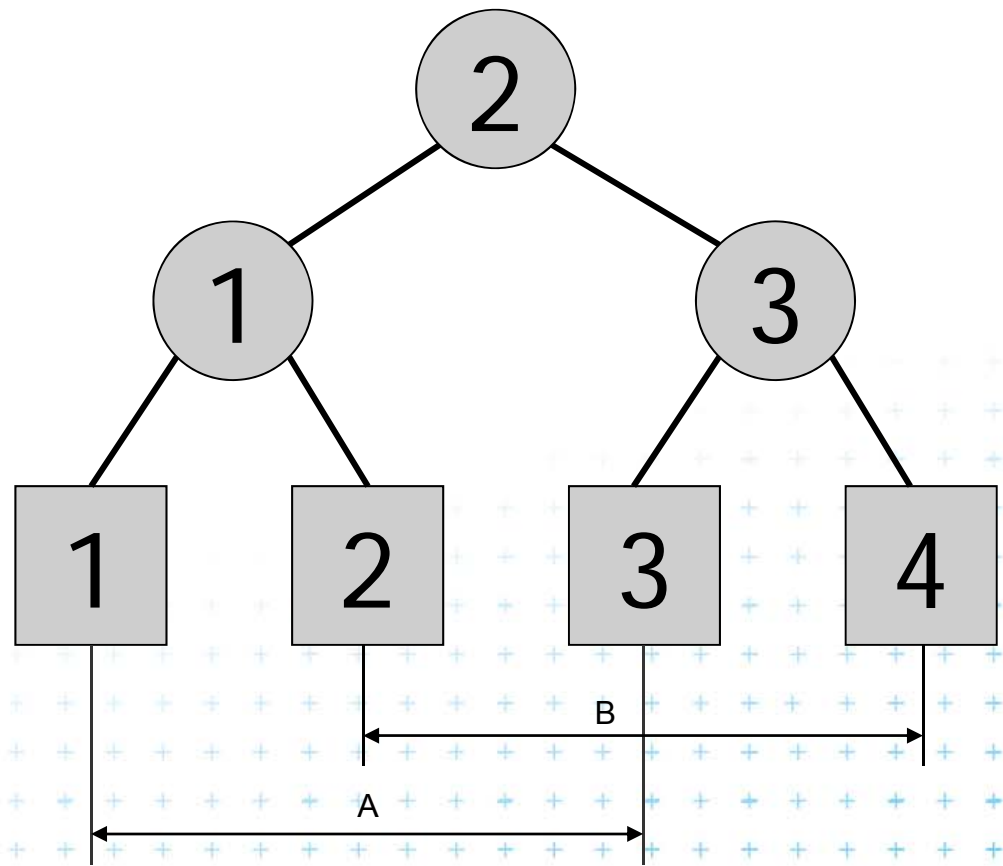
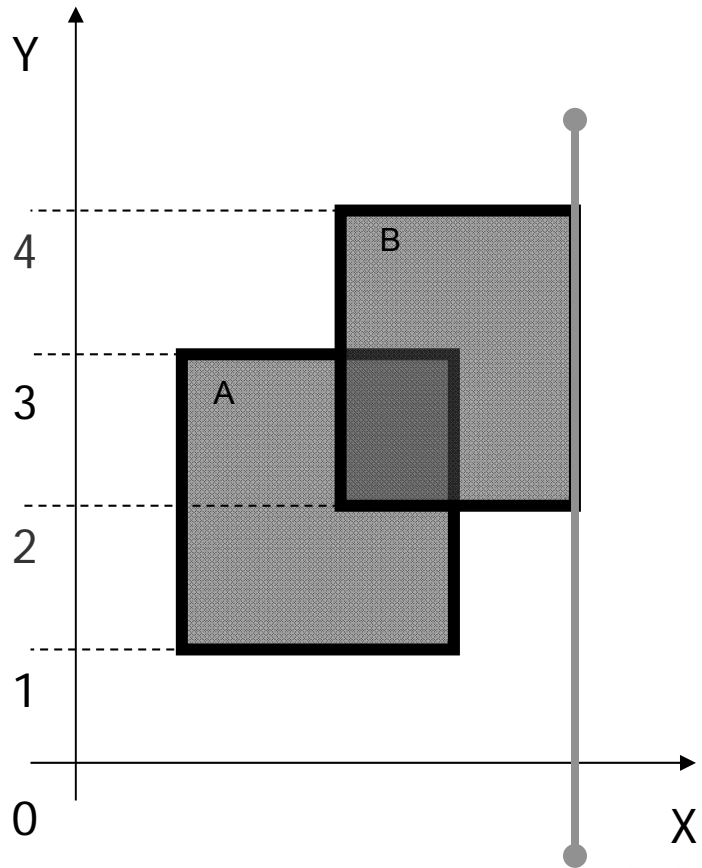
-  Active rectangle
-  Current node
-  Active node



[Drtina]



# Interval delete [2,4]



[Drtina]



# Example 2

---

## RectangleIntersections( S )

*Input:* Set S of rectangles

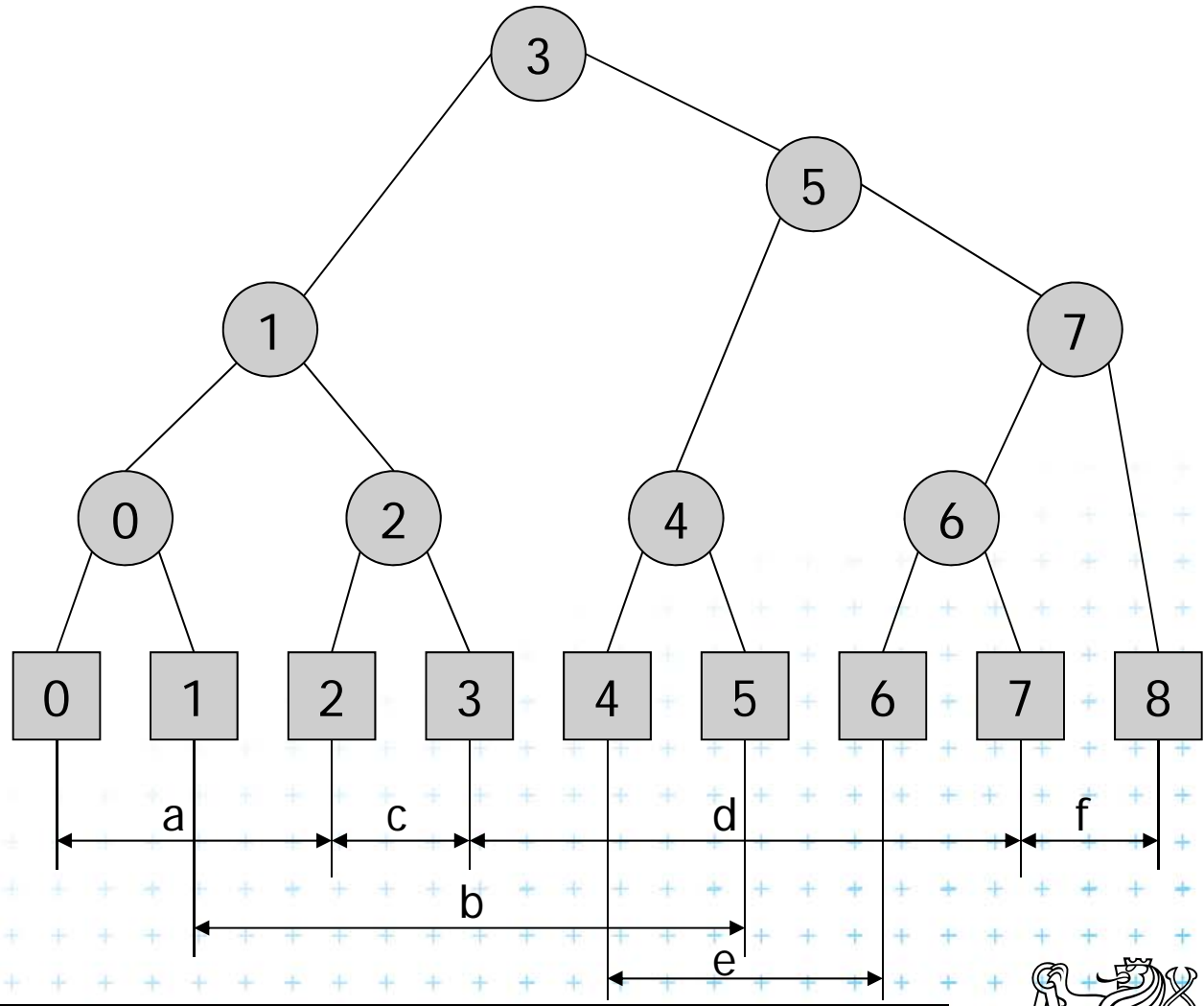
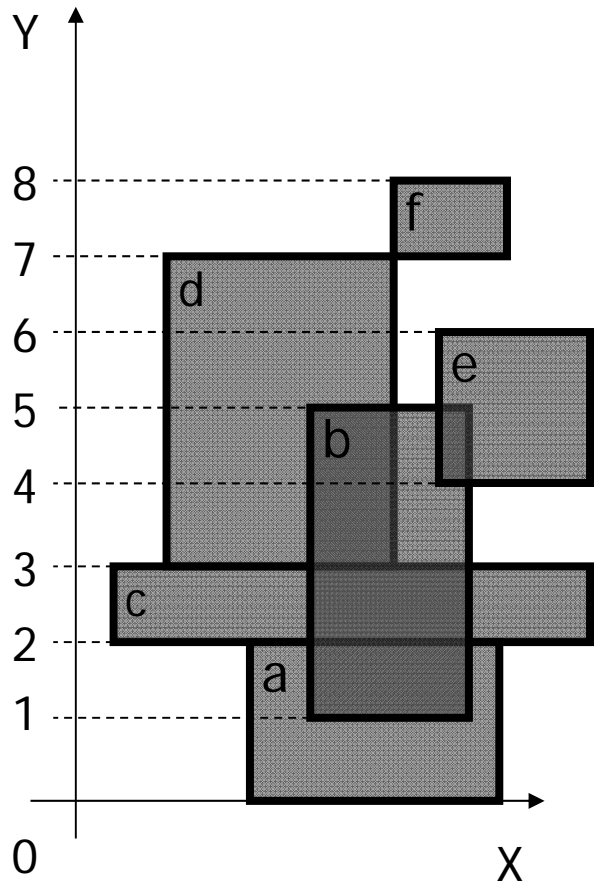
*Output:* Intersected rectangle pairs

// this is a copy of the slide before  
// just to remember the algorithm

1. Preprocess( S ) // create the interval tree T and event queue Q
2. while ( Q  $\neq \emptyset$  ) do
3.   Get next entry  $(x_{il}, y_{il}, y_{ir}, t)$  from Q //  $t \in \{ left | right \}$
4.   if (  $t = left$  ) // left edge
5.       a) QueryInterval (  $y_{il}, y_{ir}, root(T)$  ) // report intersections
6.       b) InsertInterval (  $y_{il}, y_{ir}, root(T)$  ) // insert new interval
7.   else // right edge
8.       c) DeleteInterval (  $y_{il}, y_{ir}, root(T)$  )



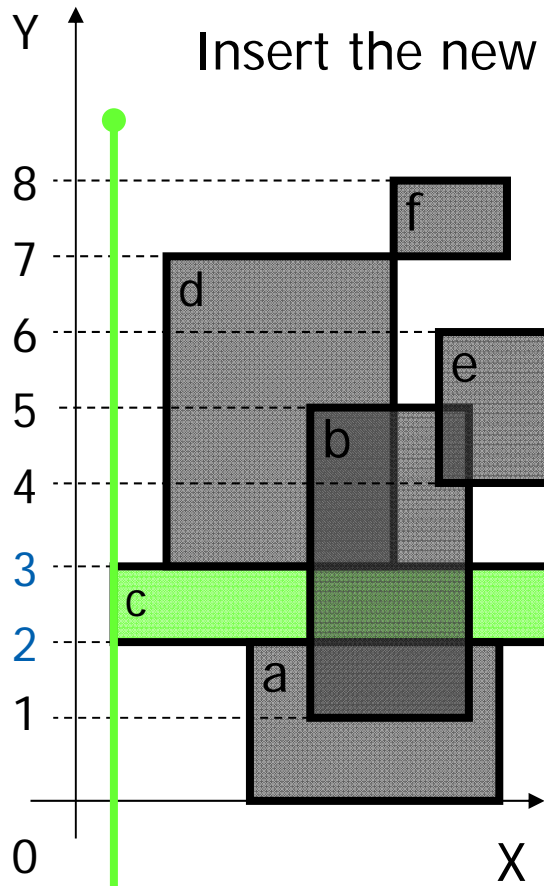
# Example 2



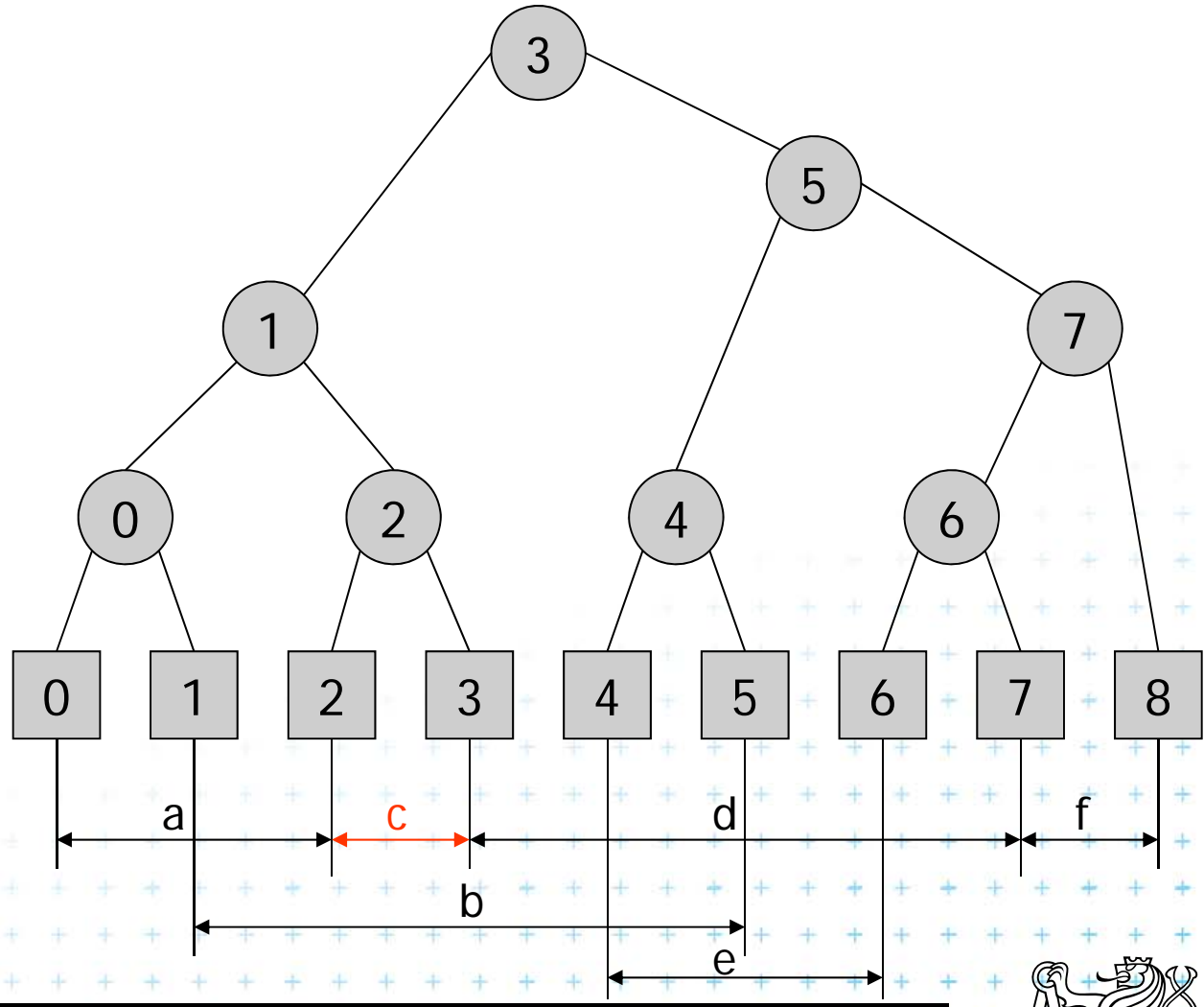





# Insert [2,3] – empty => b) Insert Interval

$$b \leq H(v) \leq e$$



Insert the new interval to secondary lists

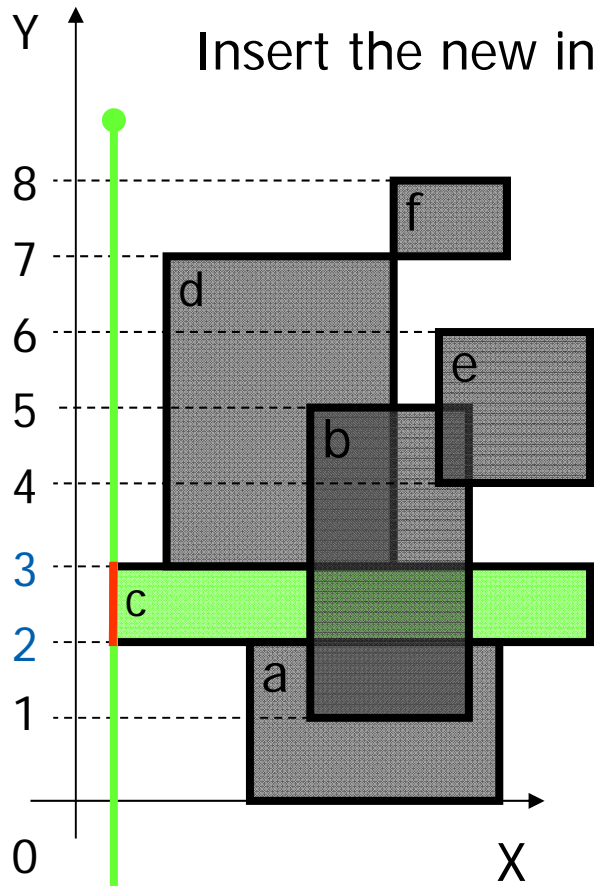


-  Active rectangle
-  Current node
-  Active node

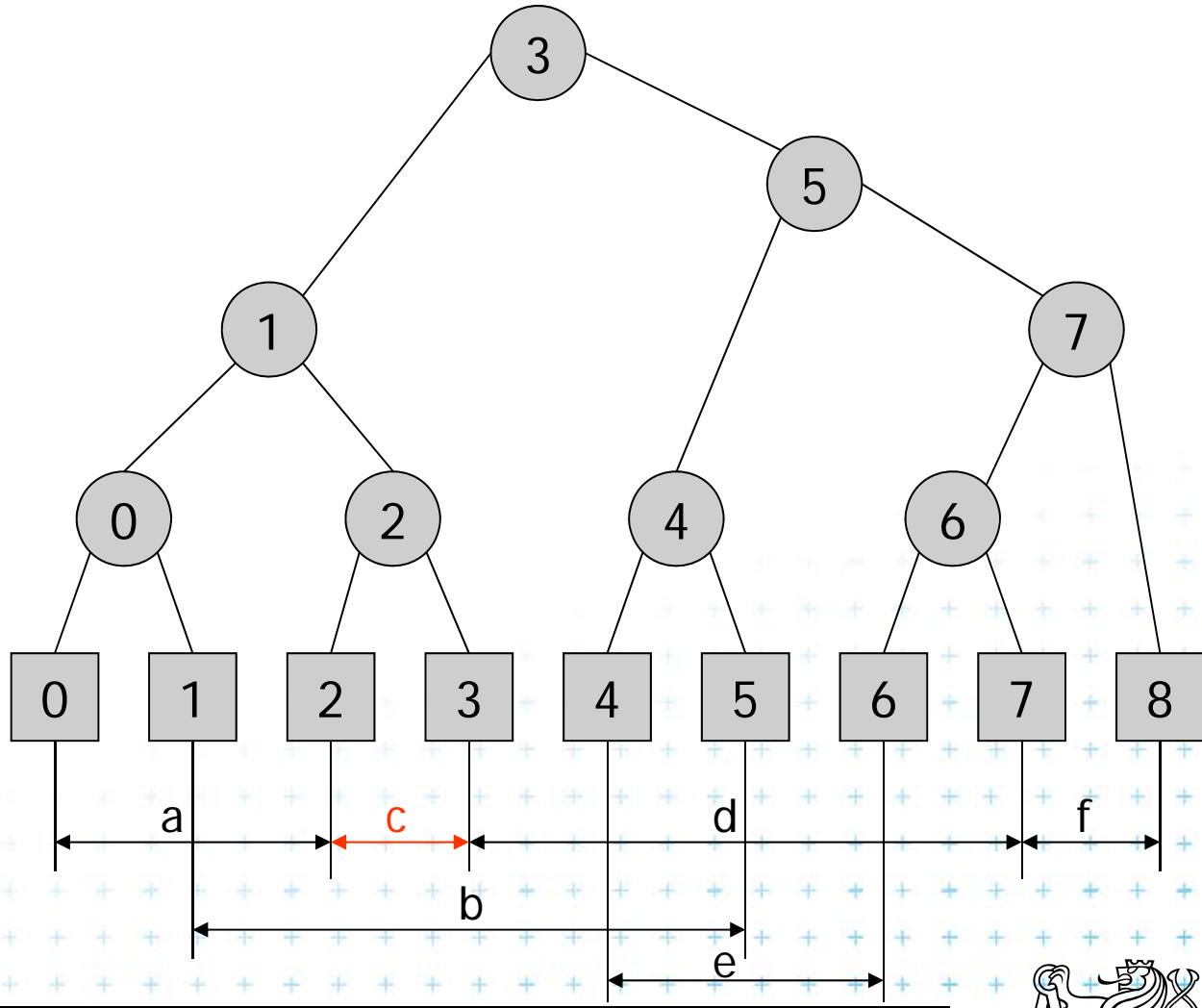





# Insert [2,3] – empty => b) Insert Interval

$$b \leq H(v) \leq e$$



Insert the new interval to secondary lists

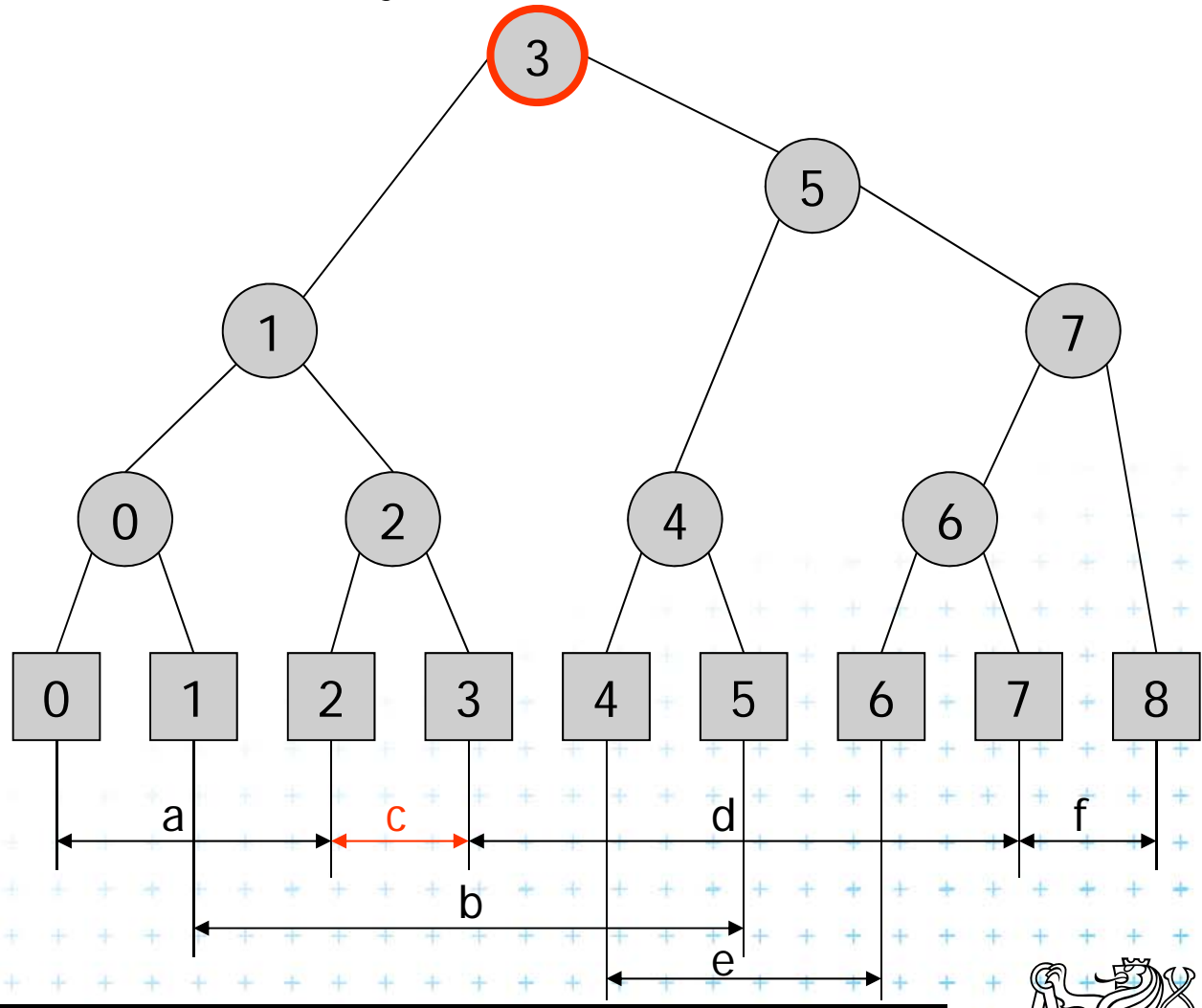
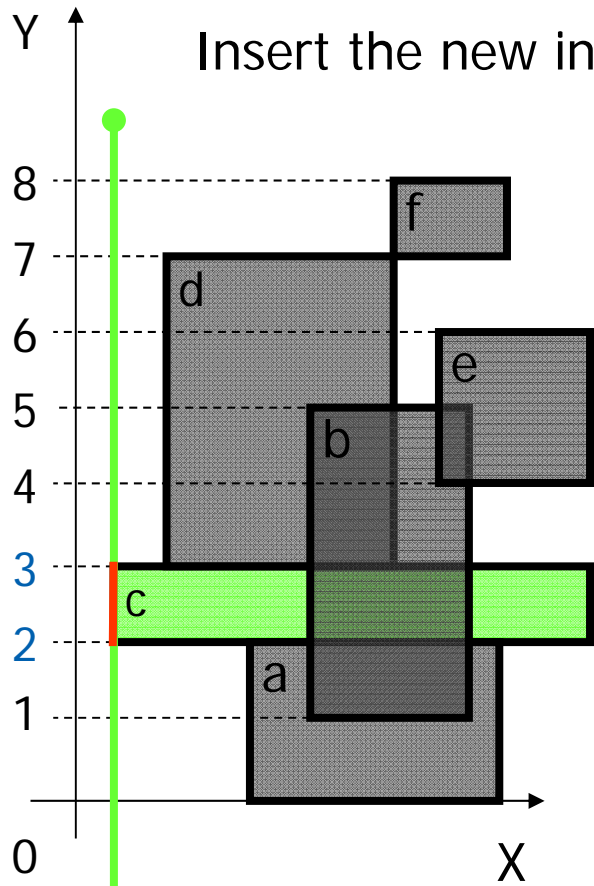


-  Active rectangle
-  Current node
-  Active node



# Insert [2,3] – empty => b) Insert Interval

$$b \leq H(v) \leq e$$

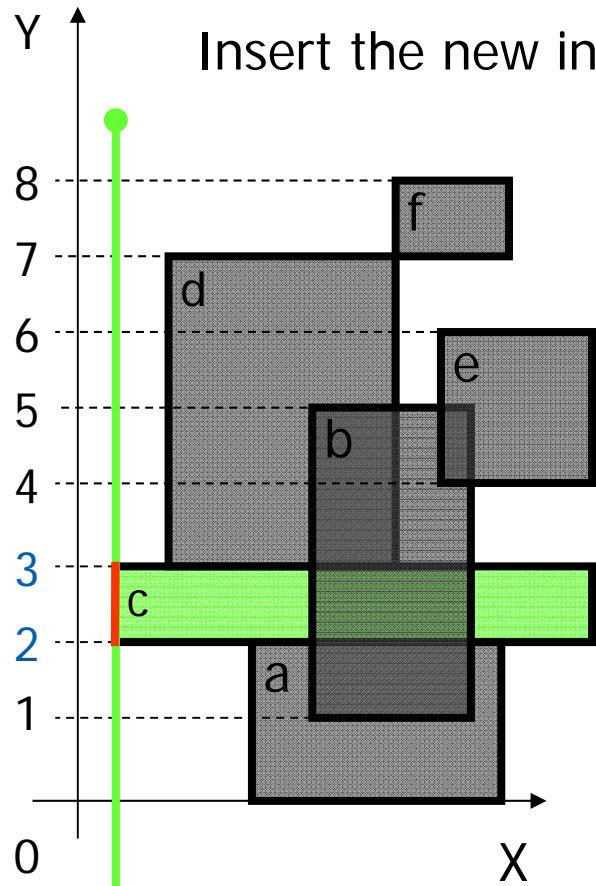


- Active rectangle
- Current node
- Active node



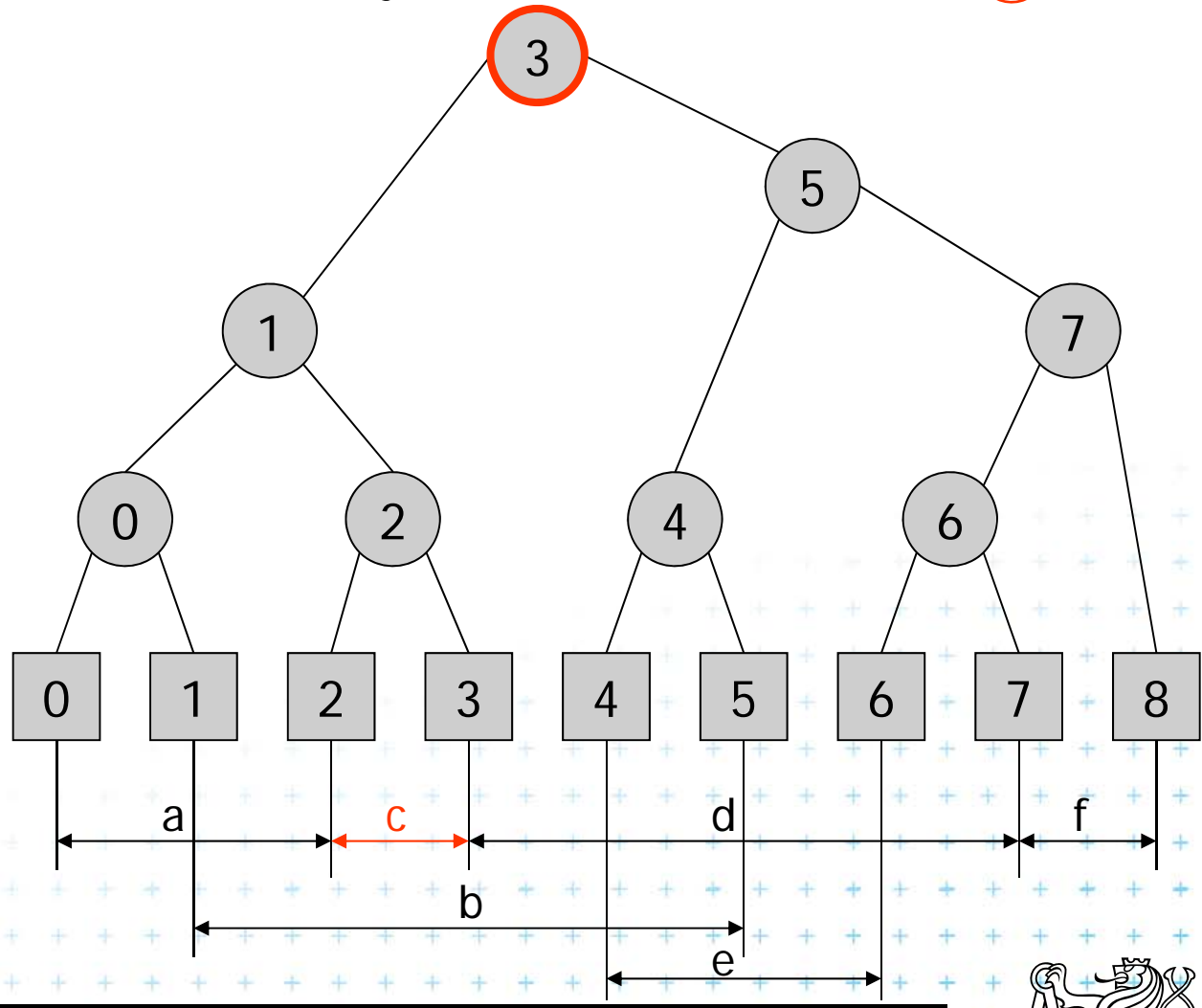
# Insert [2,3] – empty => b) Insert Interval




$$b \leq H(v) \leq e$$



Insert the new interval to secondary lists

$$? 2 \leq 3 \leq 3 ?$$

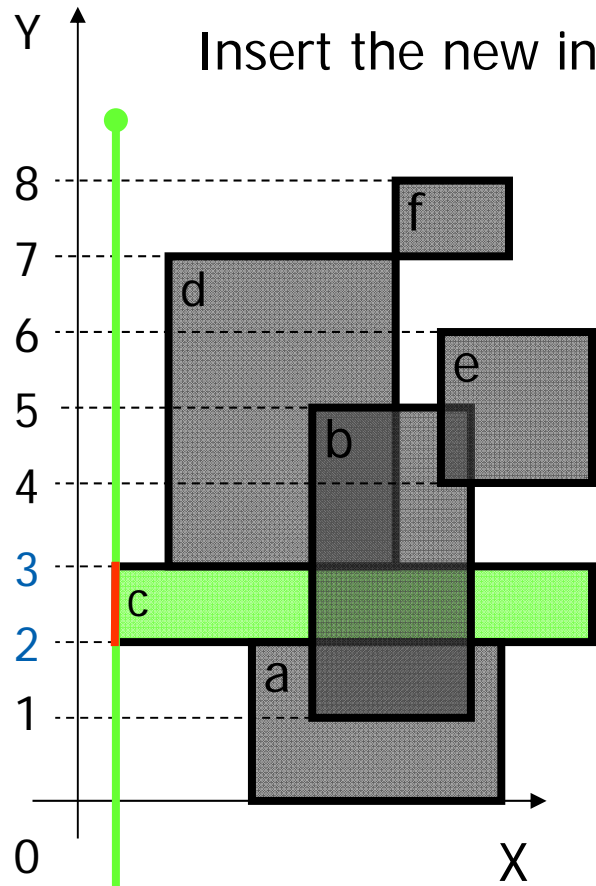


-  Active rectangle
-  Current node
-  Active node



# Insert [2,3] – empty => b) Insert Interval

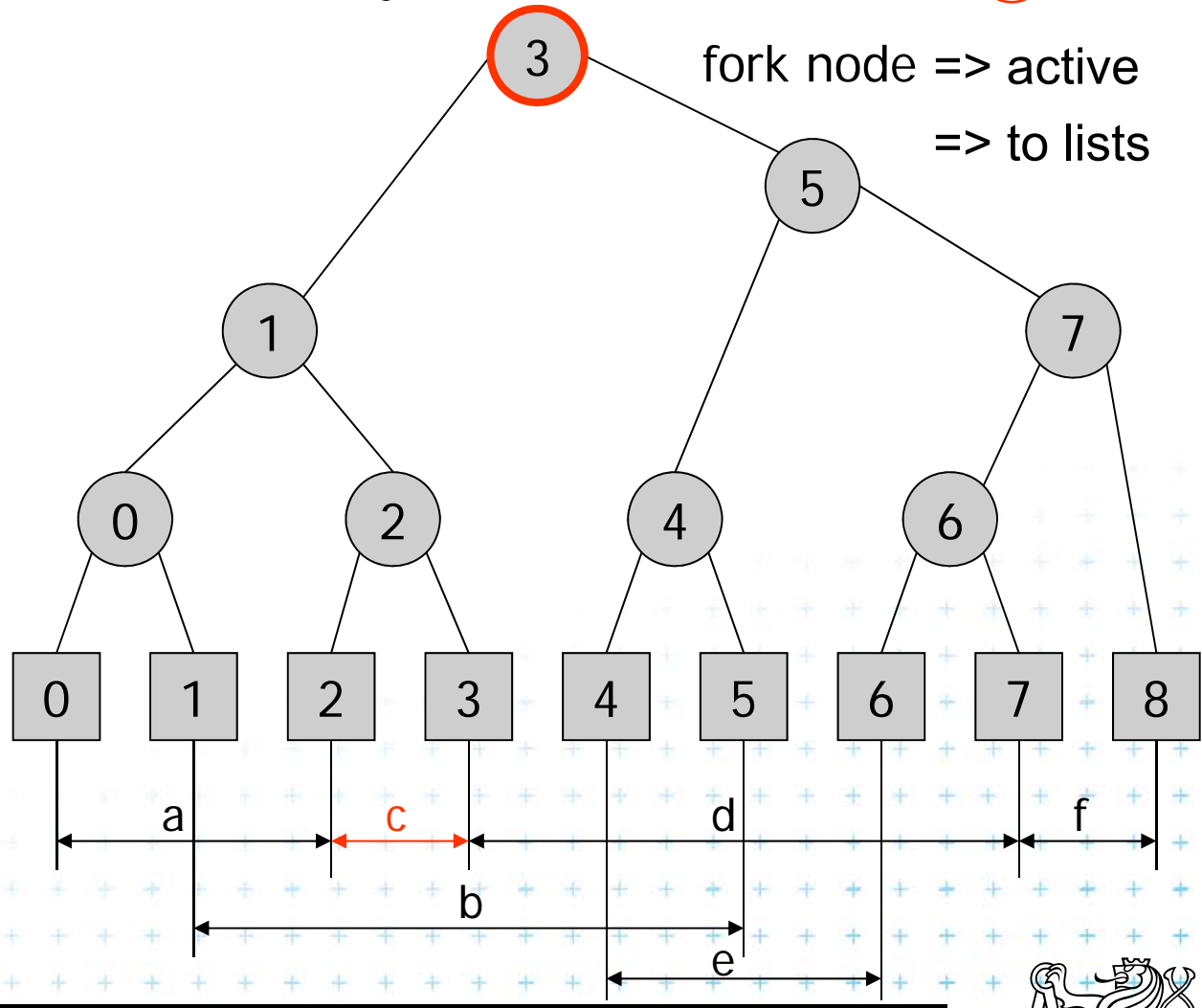
$$b \leq H(v) \leq e$$






Insert the new interval to secondary lists

$$? 2 \leq 3 \leq 3 ?$$

fork node => active  
=> to lists

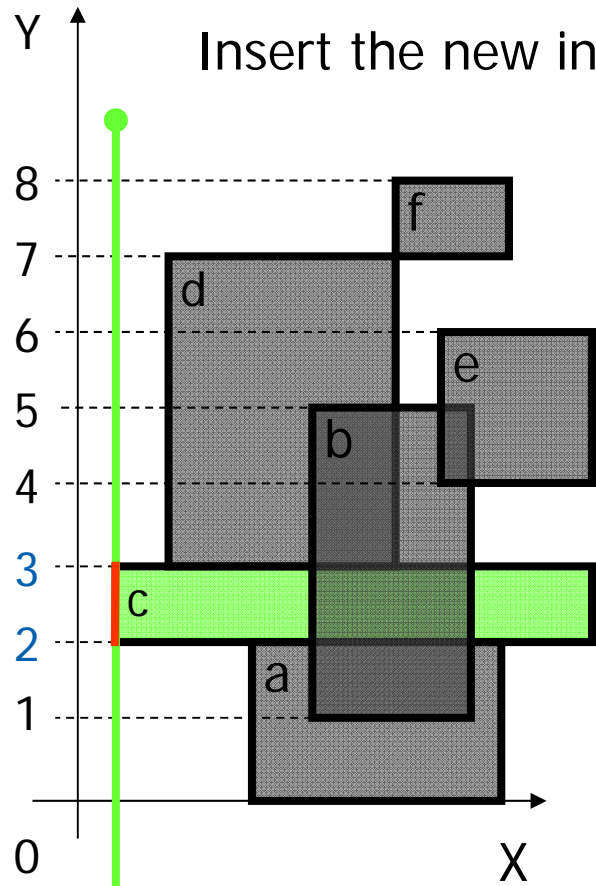


-  Active rectangle
-  Current node
-  Active node



# Insert [2,3] – empty => b) Insert Interval

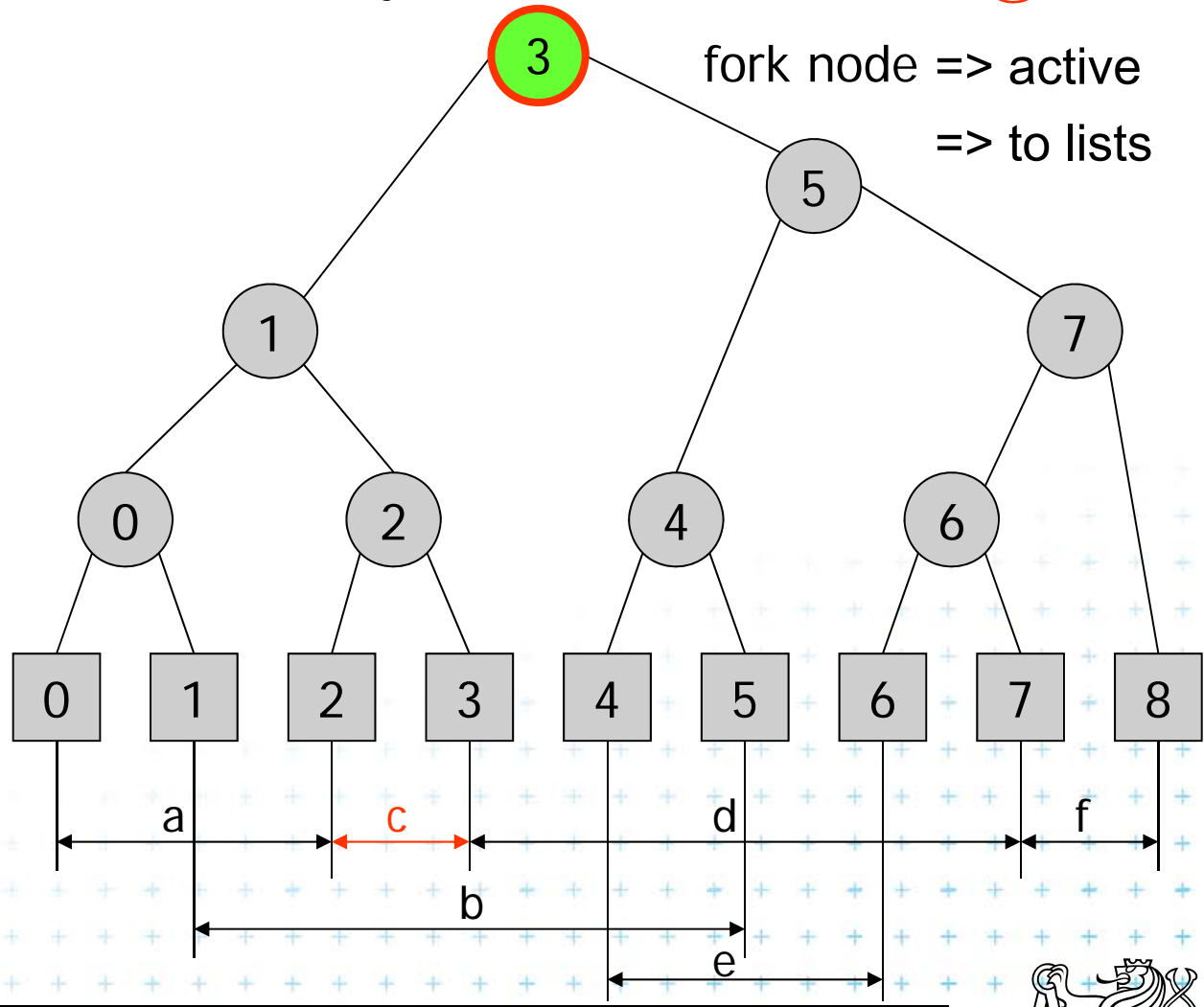
$$b \leq H(v) \leq e$$






Insert the new interval to secondary lists

$$? 2 \leq 3 \leq 3 ?$$

fork node => active  
=> to lists

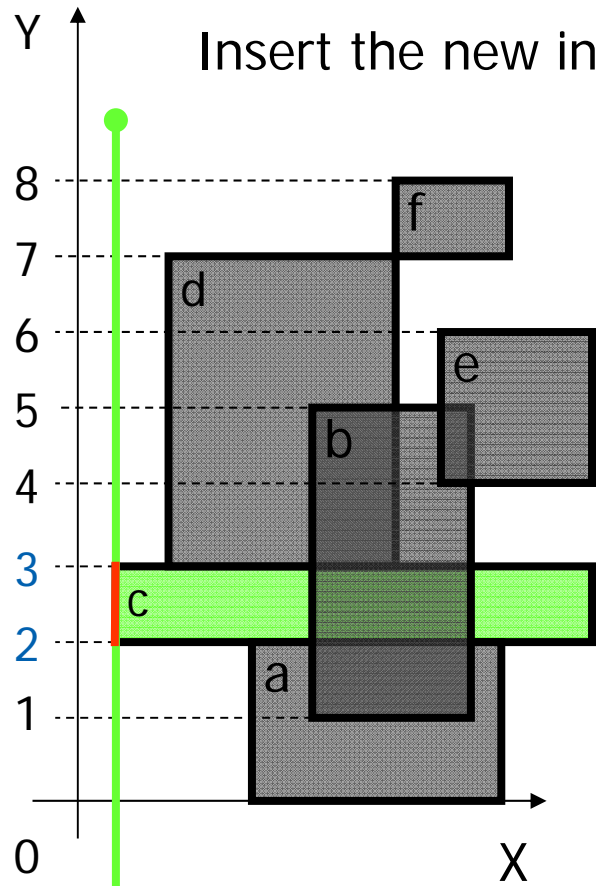


-  Active rectangle
-  Current node
-  Active node



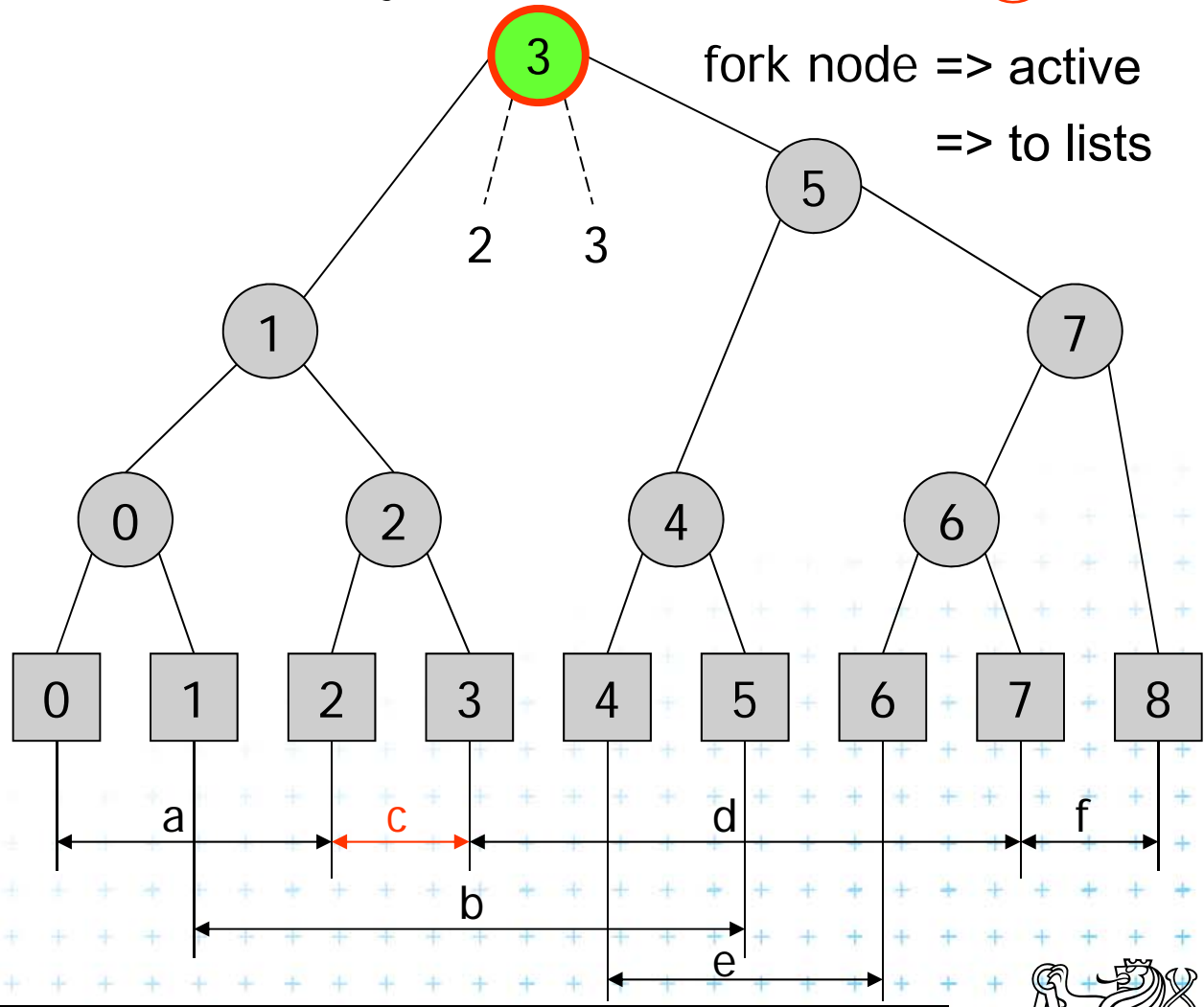
# Insert [2,3] – empty => b) Insert Interval

$$b \leq H(v) \leq e$$



Insert the new interval to secondary lists

$$? 2 \leq \textcircled{3} \leq 3 ?$$



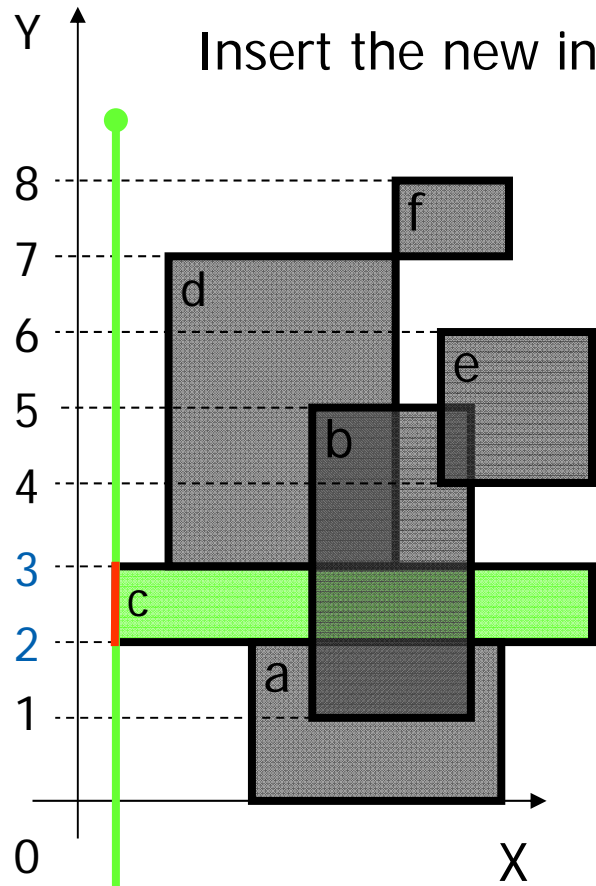
fork node => active  
=> to lists

- Active rectangle
- Current node
- Active node



# Insert [2,3] – empty => b) Insert Interval

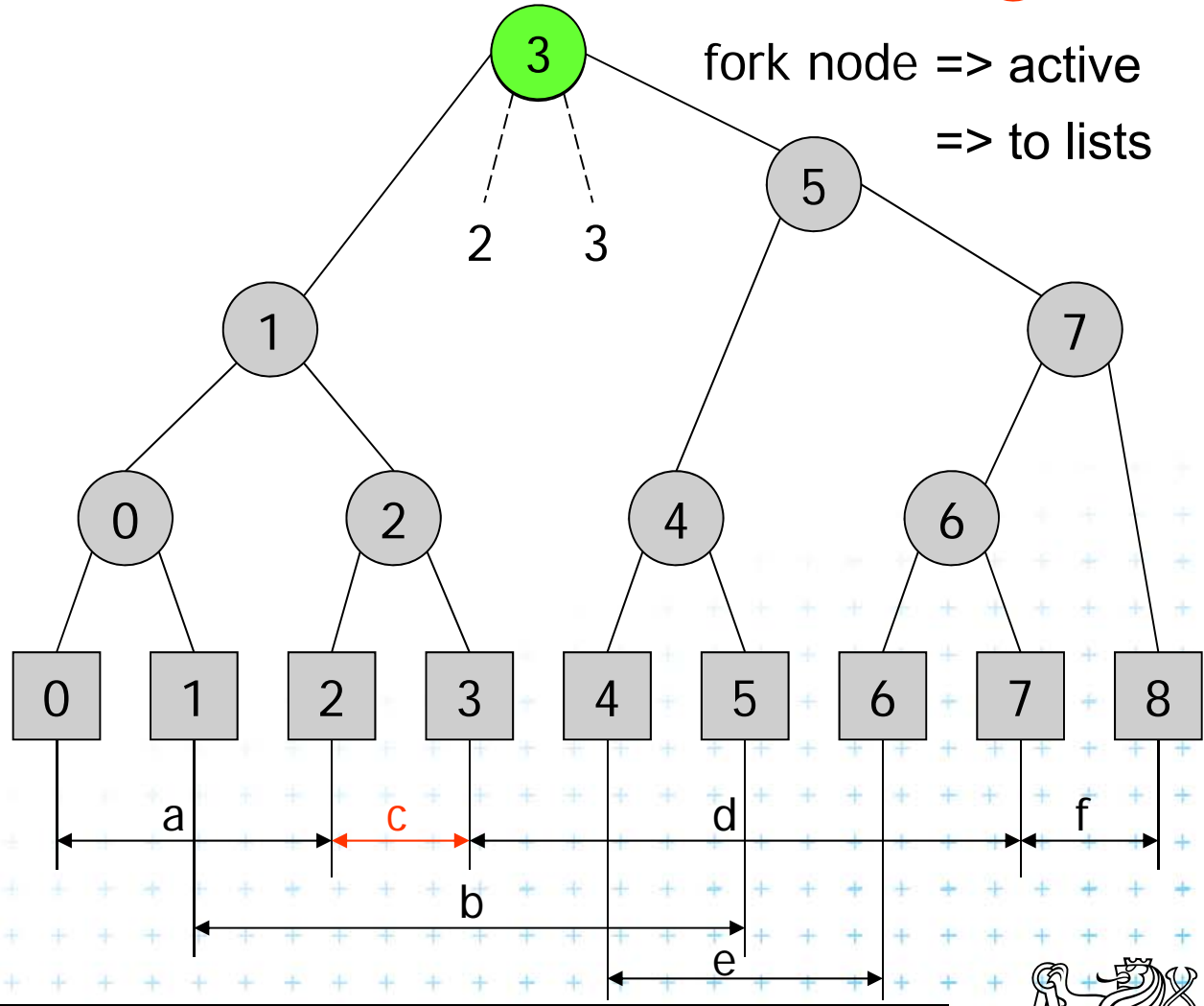
$$b \leq H(v) \leq e$$






Insert the new interval to secondary lists

$$? 2 \leq \textcircled{3} \leq 3 ?$$

fork node => active  
=> to lists



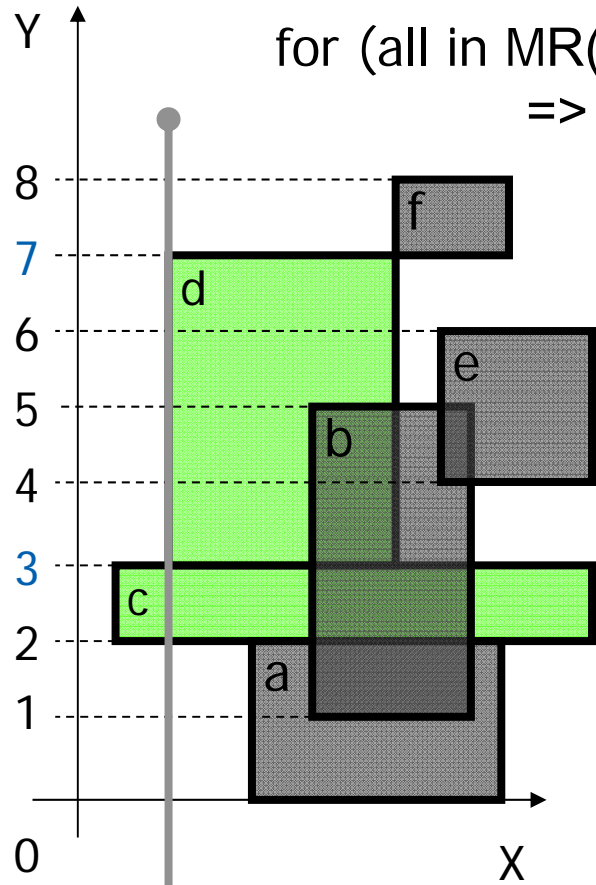
-  Active rectangle
-  Current node
-  Active node



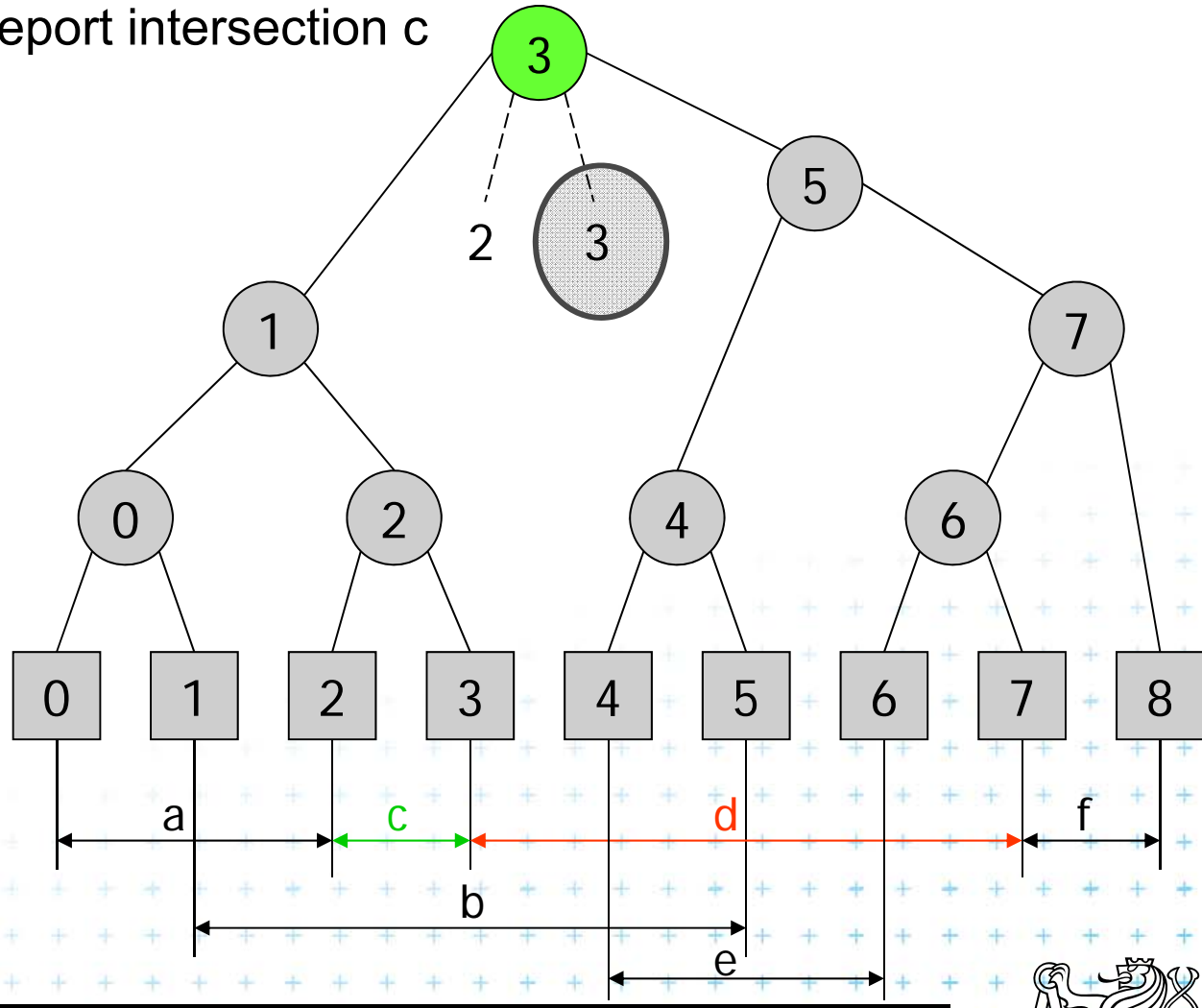


# Insert [3,7] a) Query Interval

$$H(v) \leq b < e$$

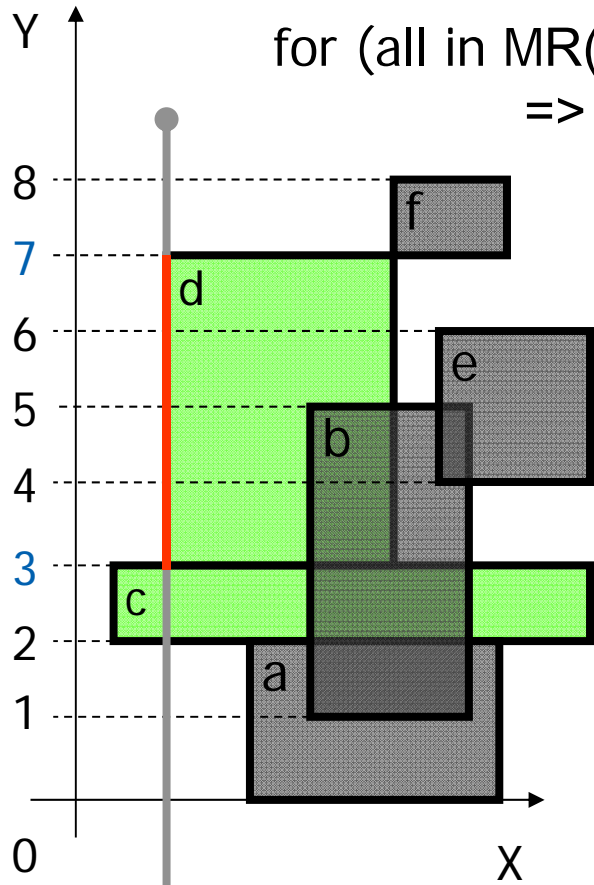


for (all in MR(v)) test  $MR(v)[i] \geq 3$   
 $\Rightarrow$  report intersection c

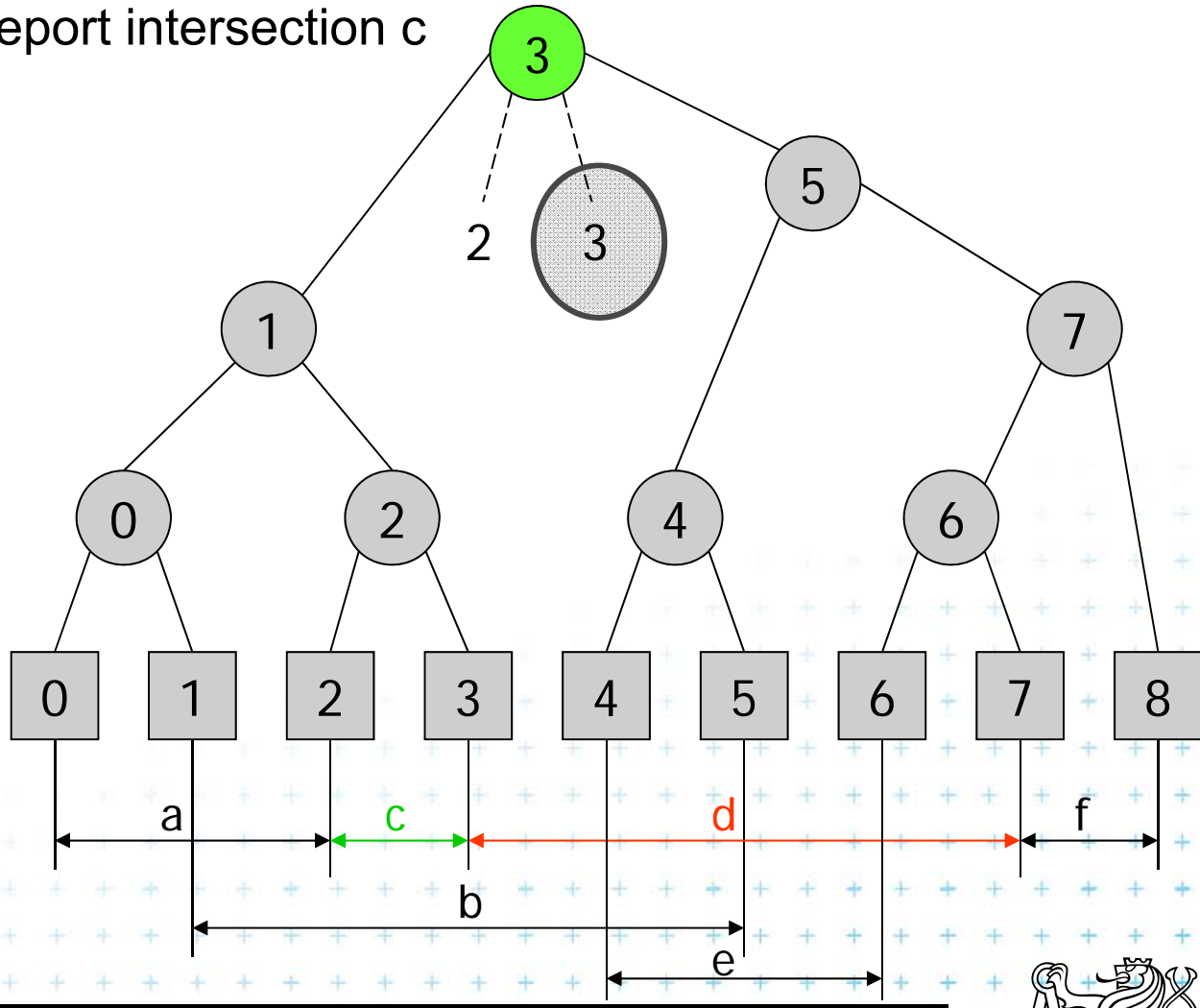





# Insert [3,7] a) Query Interval

$$H(v) \leq b < e$$



for (all in MR(v)) test  $MR(v)[i] \geq 3$   
 $\Rightarrow$  report intersection c

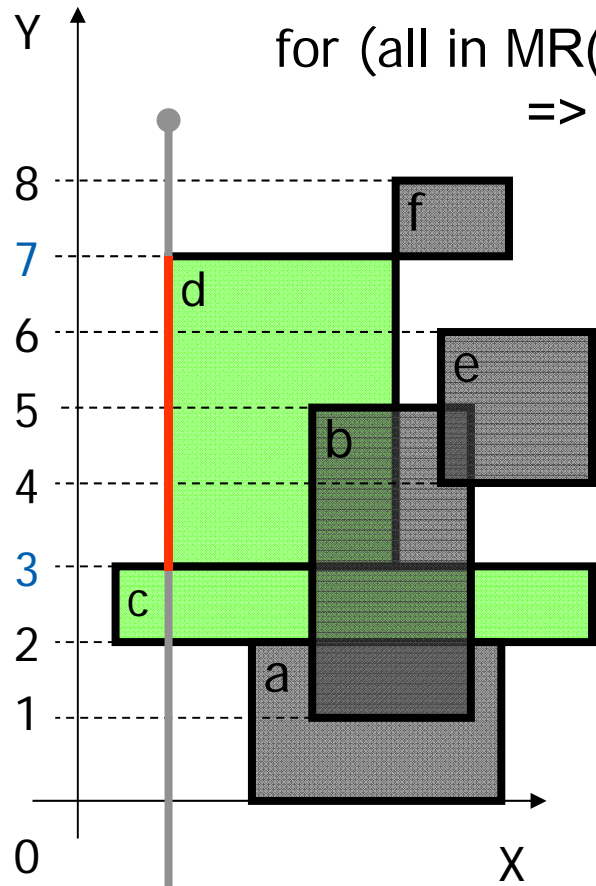


-  Active rectangle
-  Current node
-  Active node

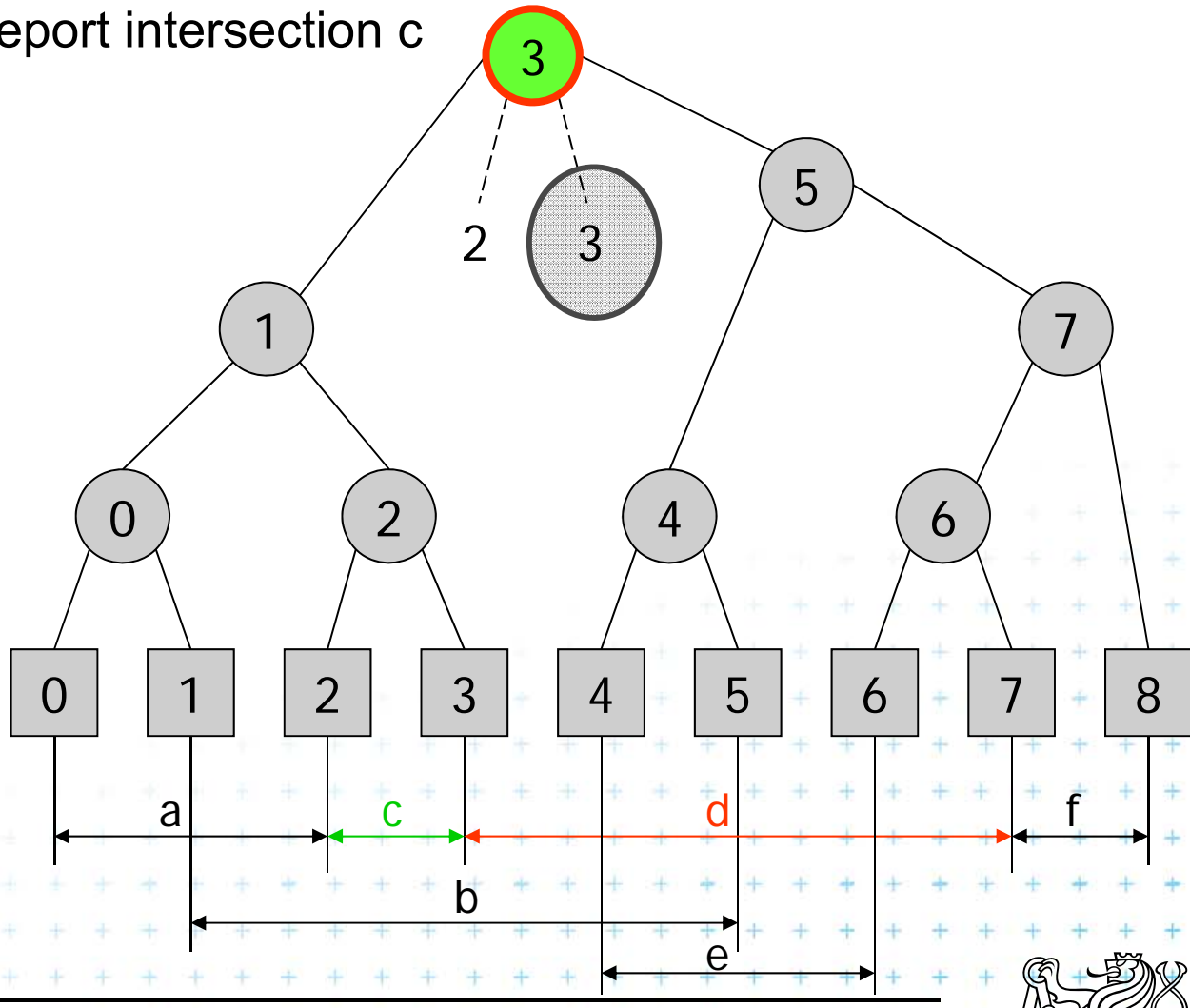


# Insert [3,7] a) Query Interval

$$H(v) \leq b < e$$

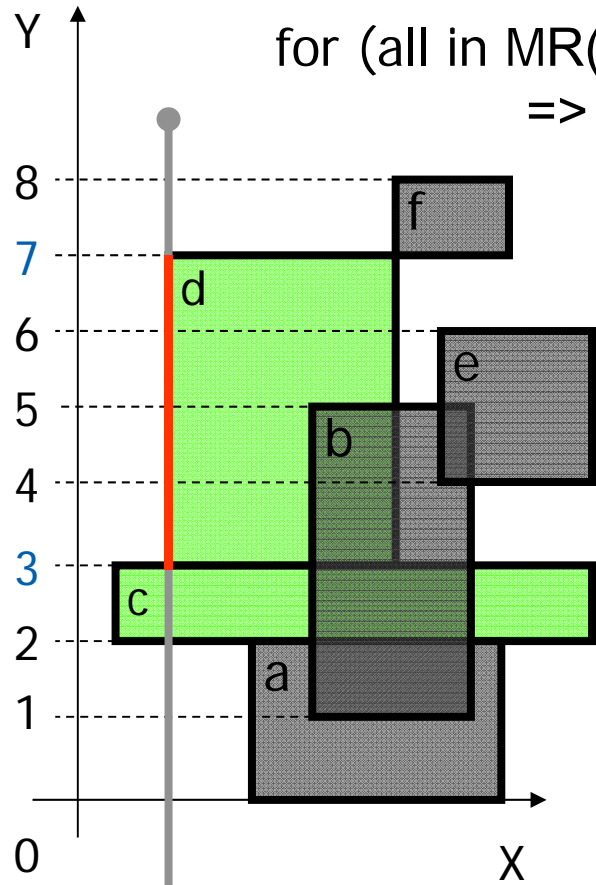


for (all in MR(v)) test  $MR(v)[i] \geq 3$   
 $\Rightarrow$  report intersection c



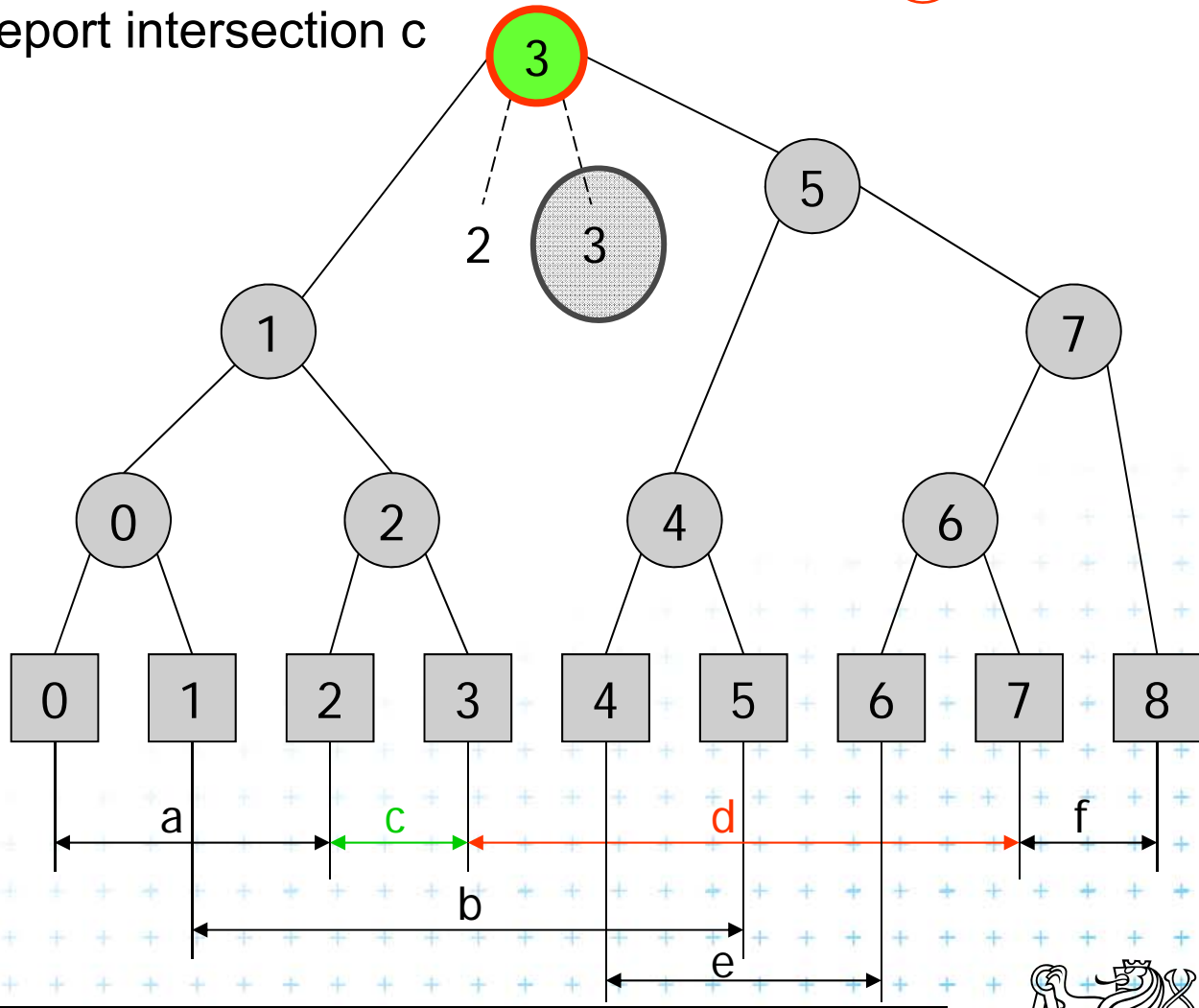
# Insert [3,7] a) Query Interval

$$H(v) \leq b < e$$



for (all in MR(v)) test  $MR(v)[i] \geq 3$   
 $\Rightarrow$  report intersection c

$$? 3 \leq 3 < 7 ?$$

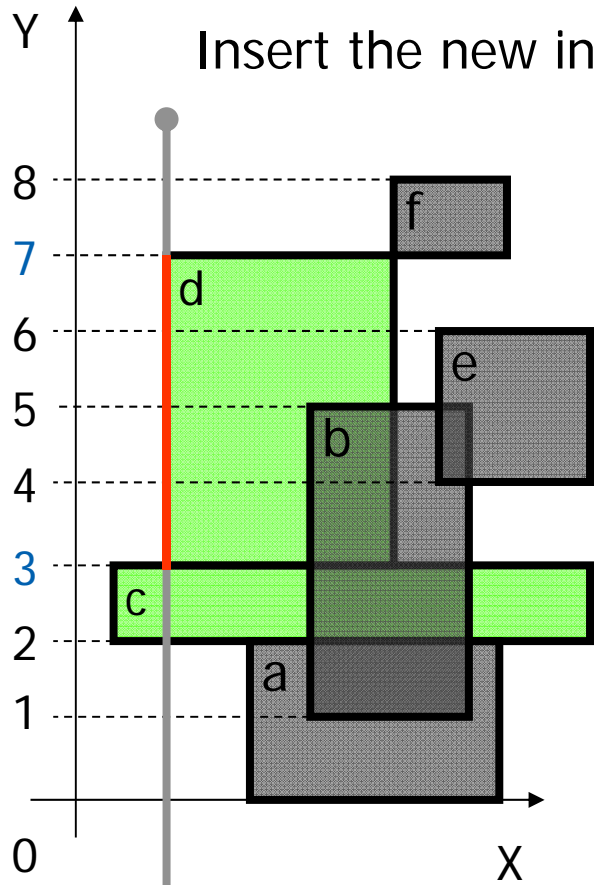


- Active rectangle
- Current node
- Active node



# Insert [3,7] b) Insert Interval

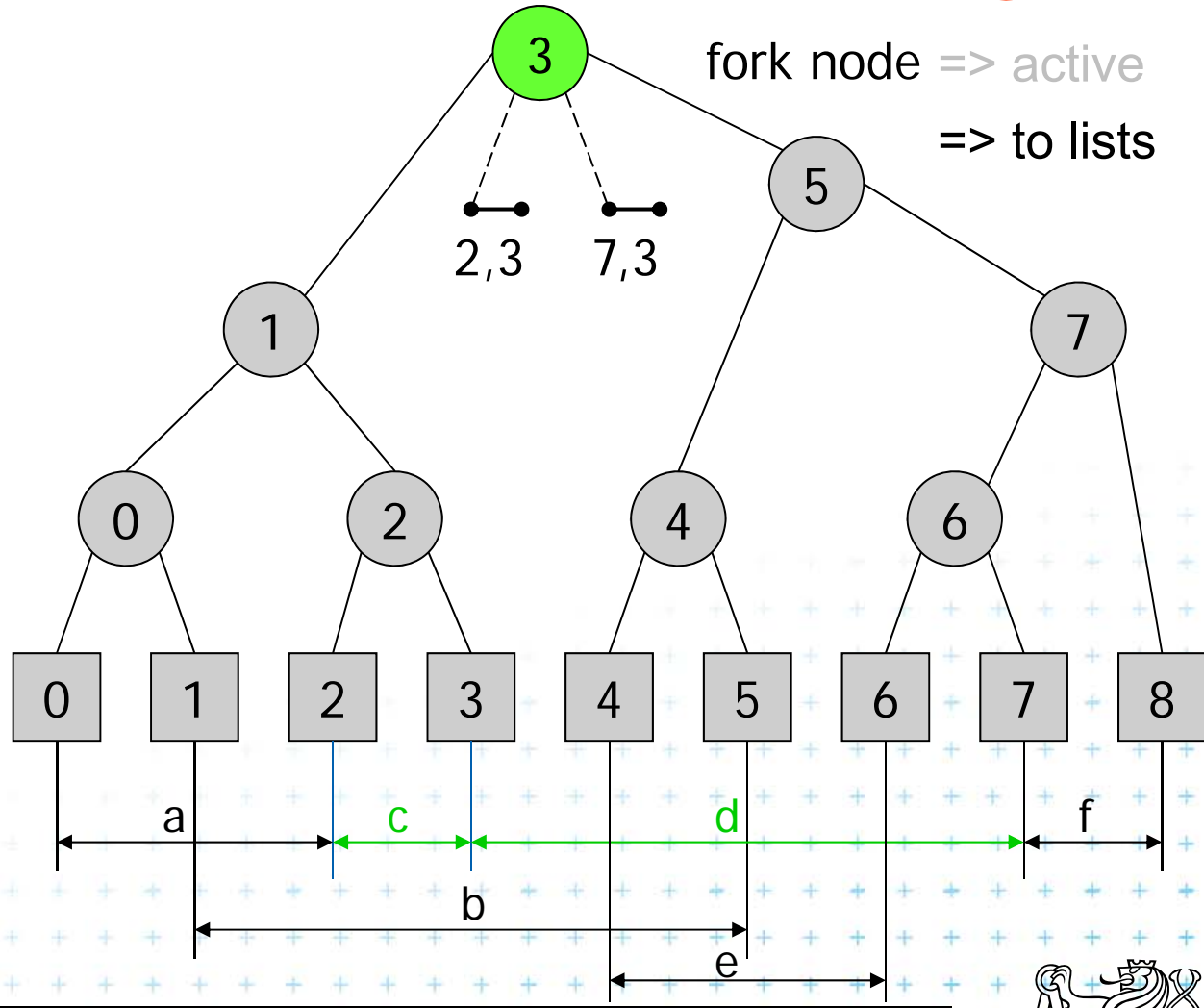
$$b \leq H(v) \leq e$$






Insert the new interval to secondary lists

$$3 \leq \textcircled{3} \leq 7$$

fork node => active  
=> to lists



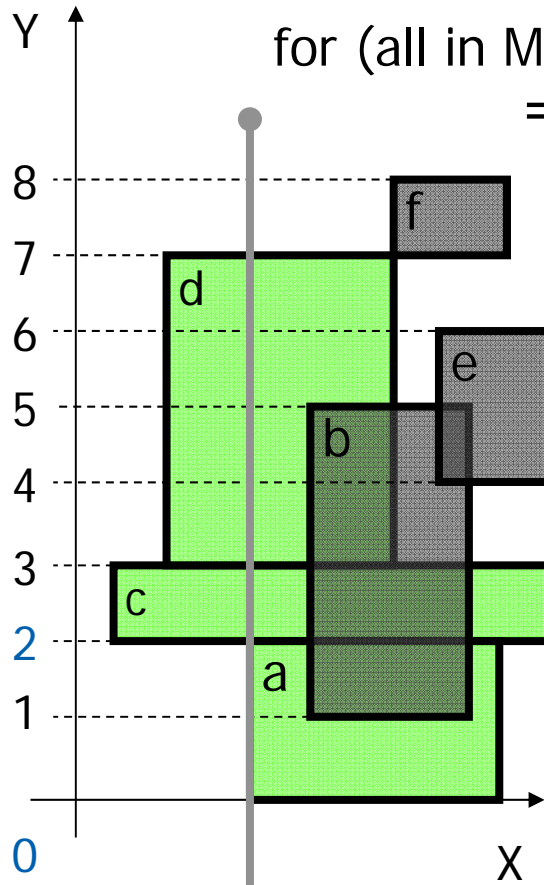
-  Active rectangle
-  Current node
-  Active node



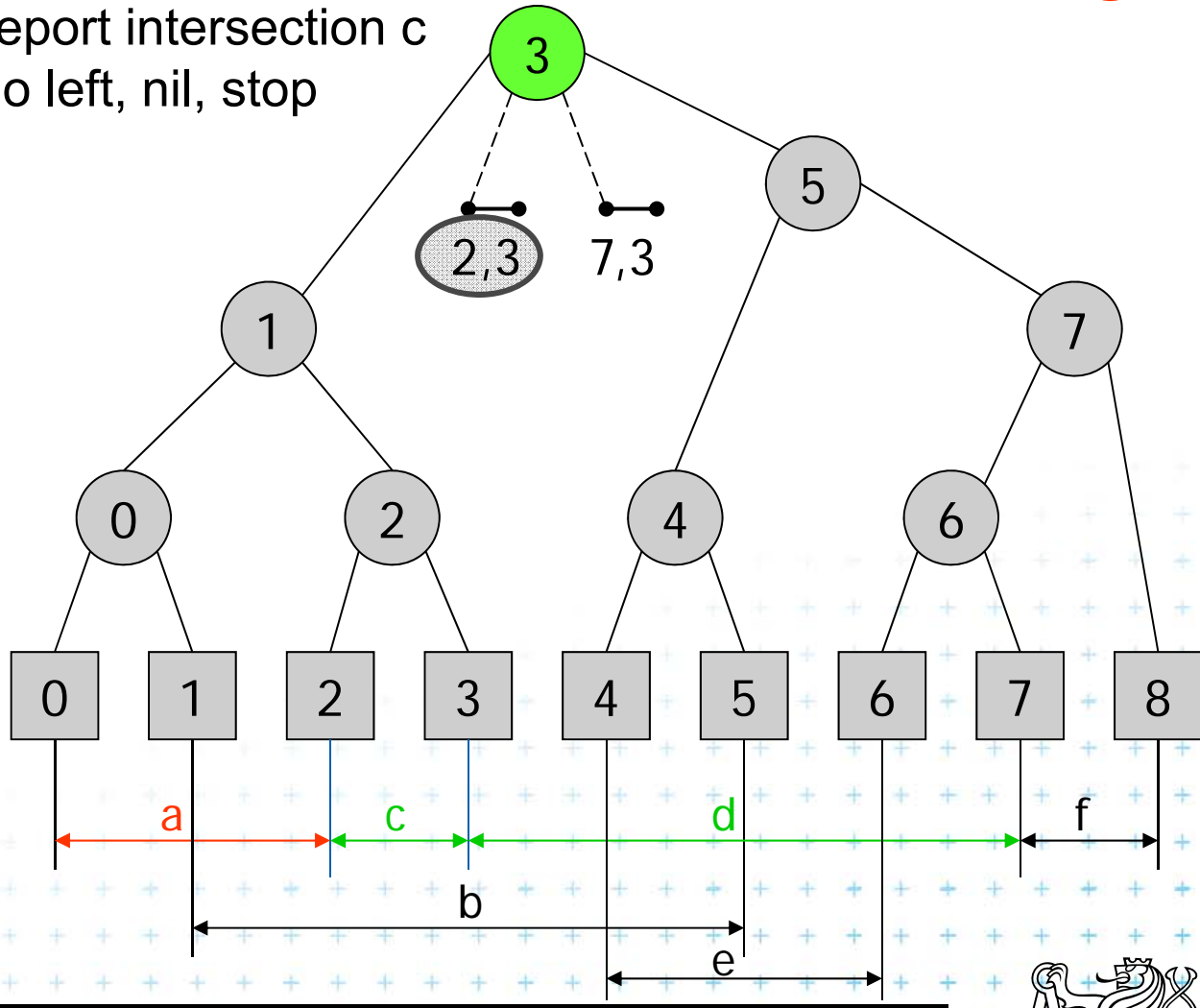
# Insert [0,2] a) Query Interval

$$b < e \leq H(v)$$

$$? 0 < 2 \leq 3 ?$$



for (all in ML(v)) test  $ML(v).[i] \leq 2$   
 $\Rightarrow$  report intersection c  
 go left, nil, stop



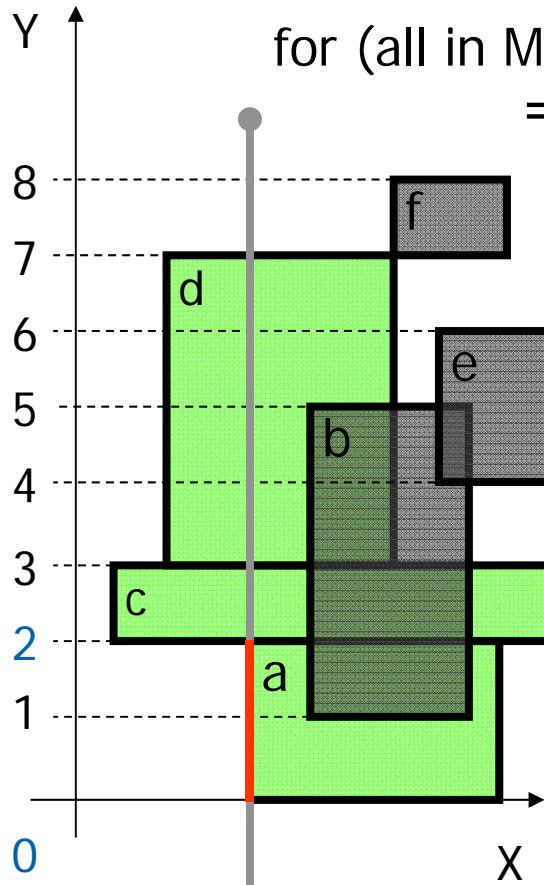
- Active rectangle
- Current node
- Active node



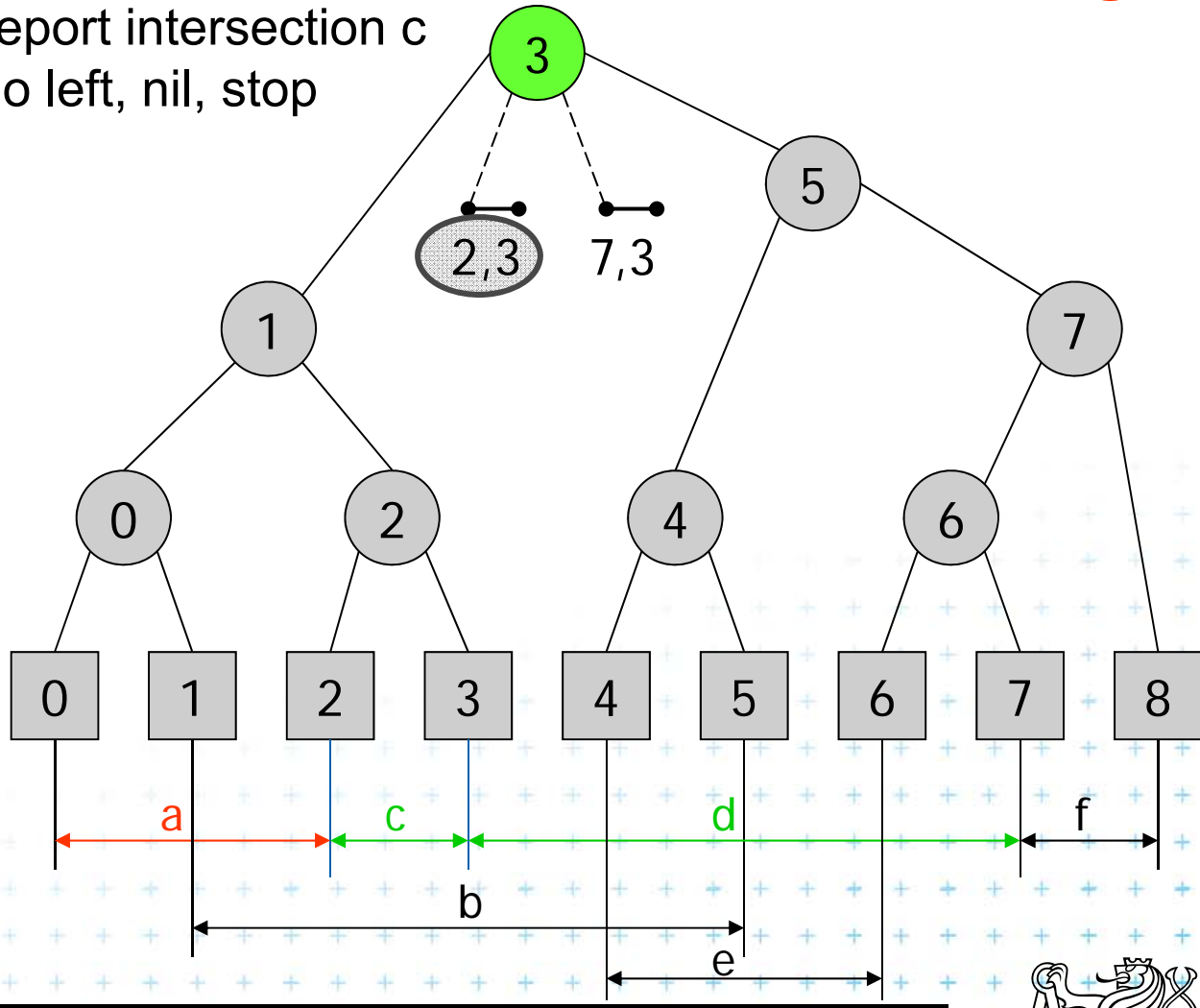
# Insert [0,2] a) Query Interval

$$b < e \leq H(v)$$

$$? 0 < 2 \leq 3 ?$$



for (all in ML(v)) test  $ML(v).[i] \leq 2$   
 $\Rightarrow$  report intersection c  
 go left, nil, stop



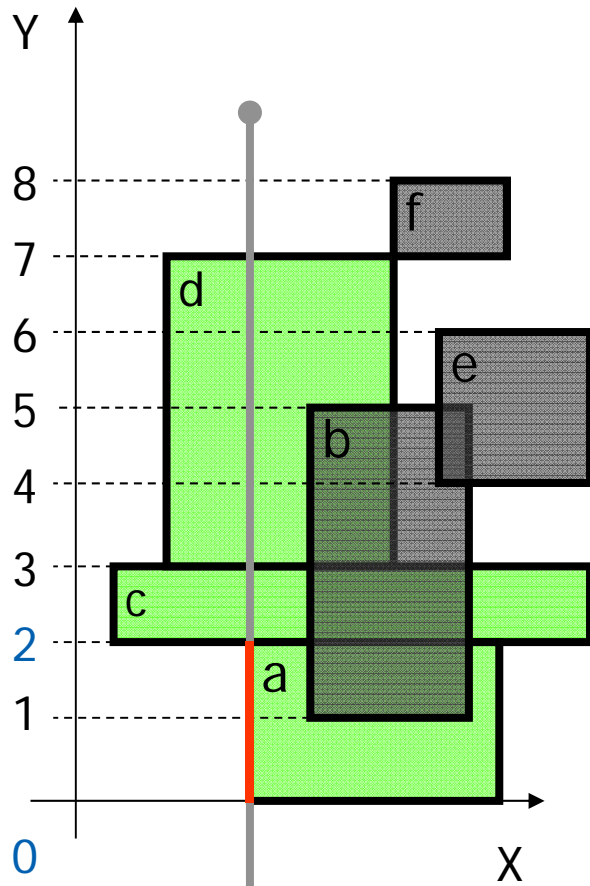
- Active rectangle
- Current node
- Active node






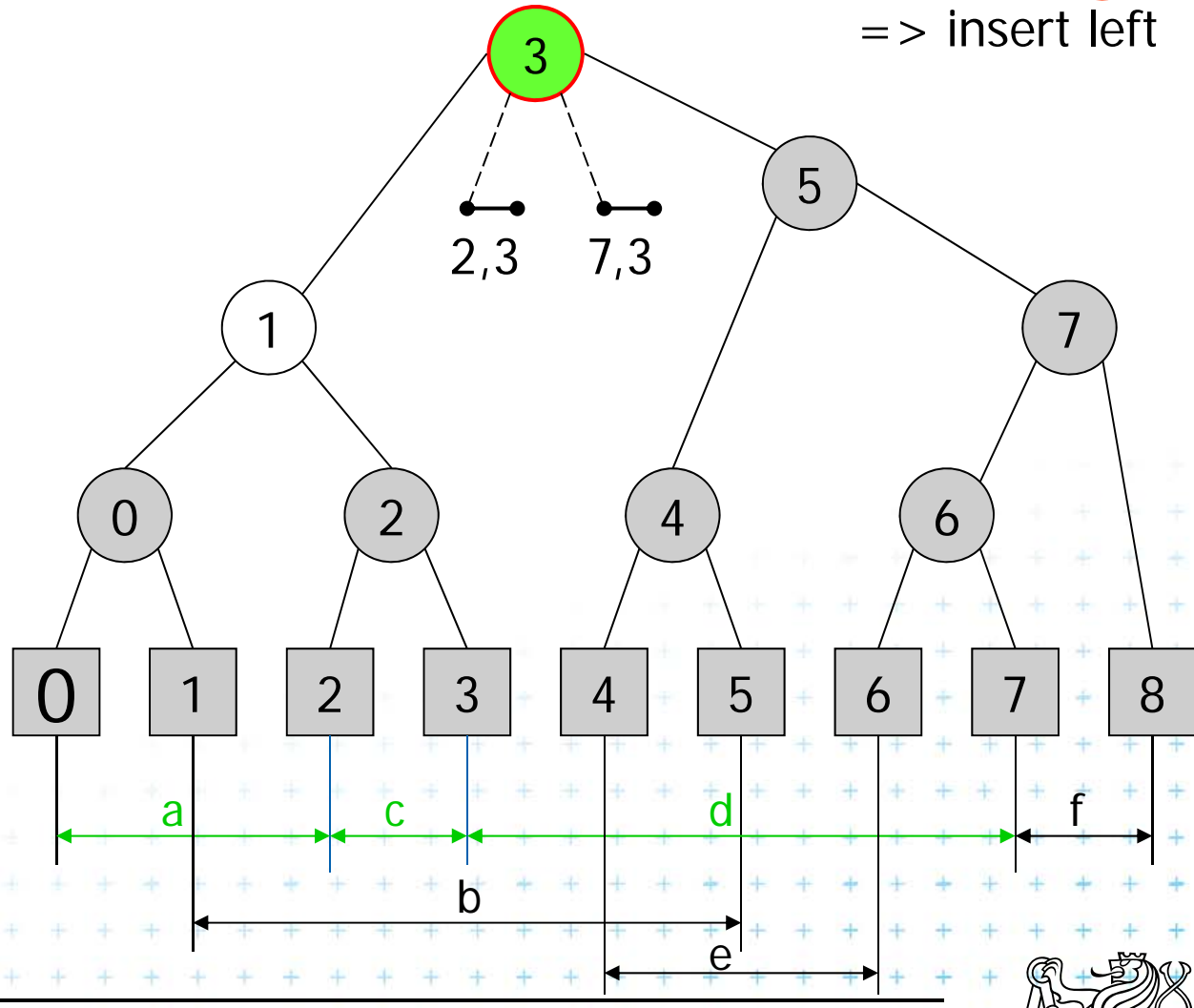
# Insert [0,2] b) Insert Interval 1/2

$$b < e < H(v)$$

?  $0 < 2 < 3$ ?  
 $\Rightarrow$  insert left



-  Active rectangle
-  Current node
-  Active node

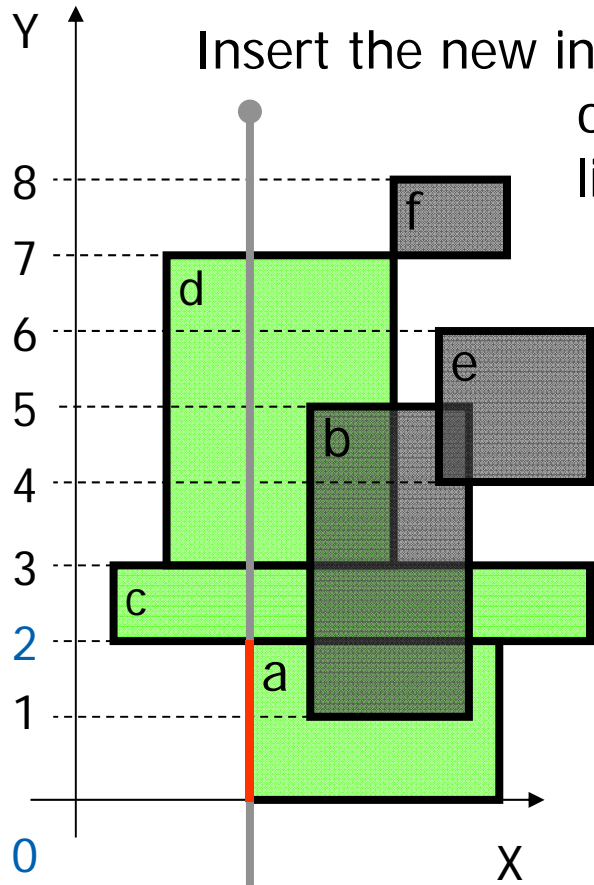




# Insert [0,2] b) Insert Interval 2/2

$$b \leq H(v) \leq e$$

$$? 0 \leq 1 \leq 2 ?$$

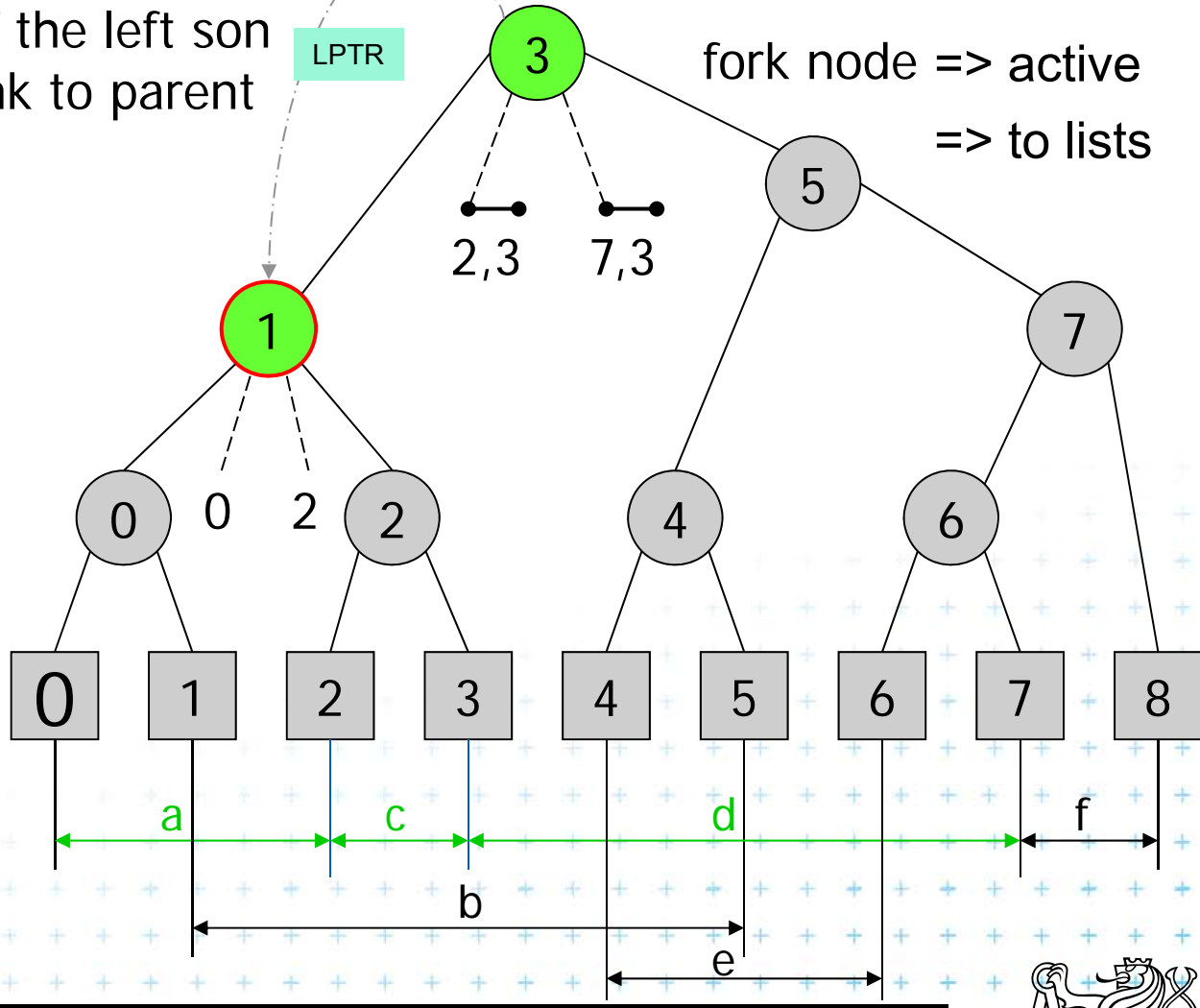


Insert the new interval to secondary lists

of the left son  
link to parent

LPTR

fork node => active  
=> to lists

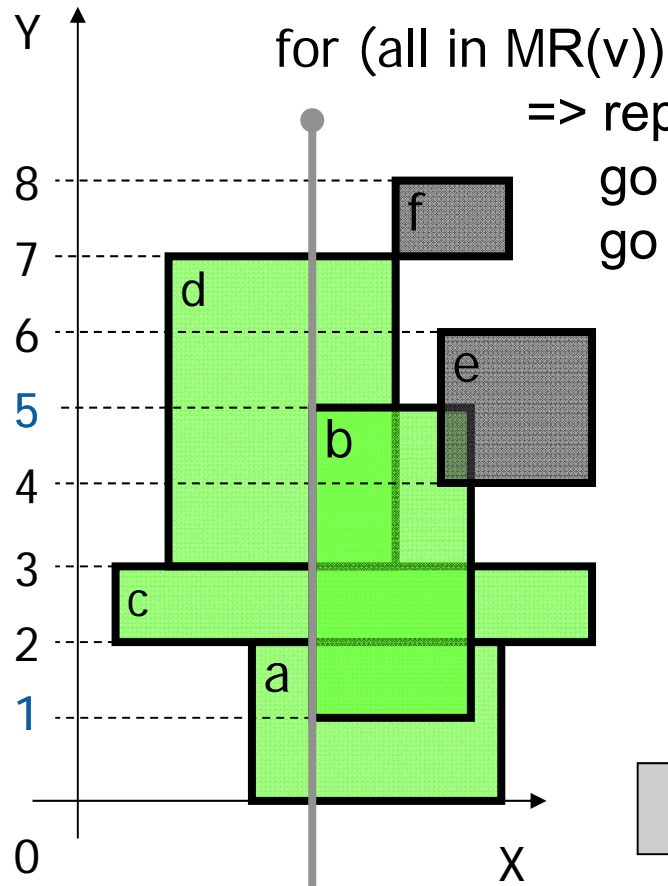


- Active rectangle
- Current node
- Active node



# Insert [1,5] a) Query Interval 1/2

$$b < H(v) < e$$

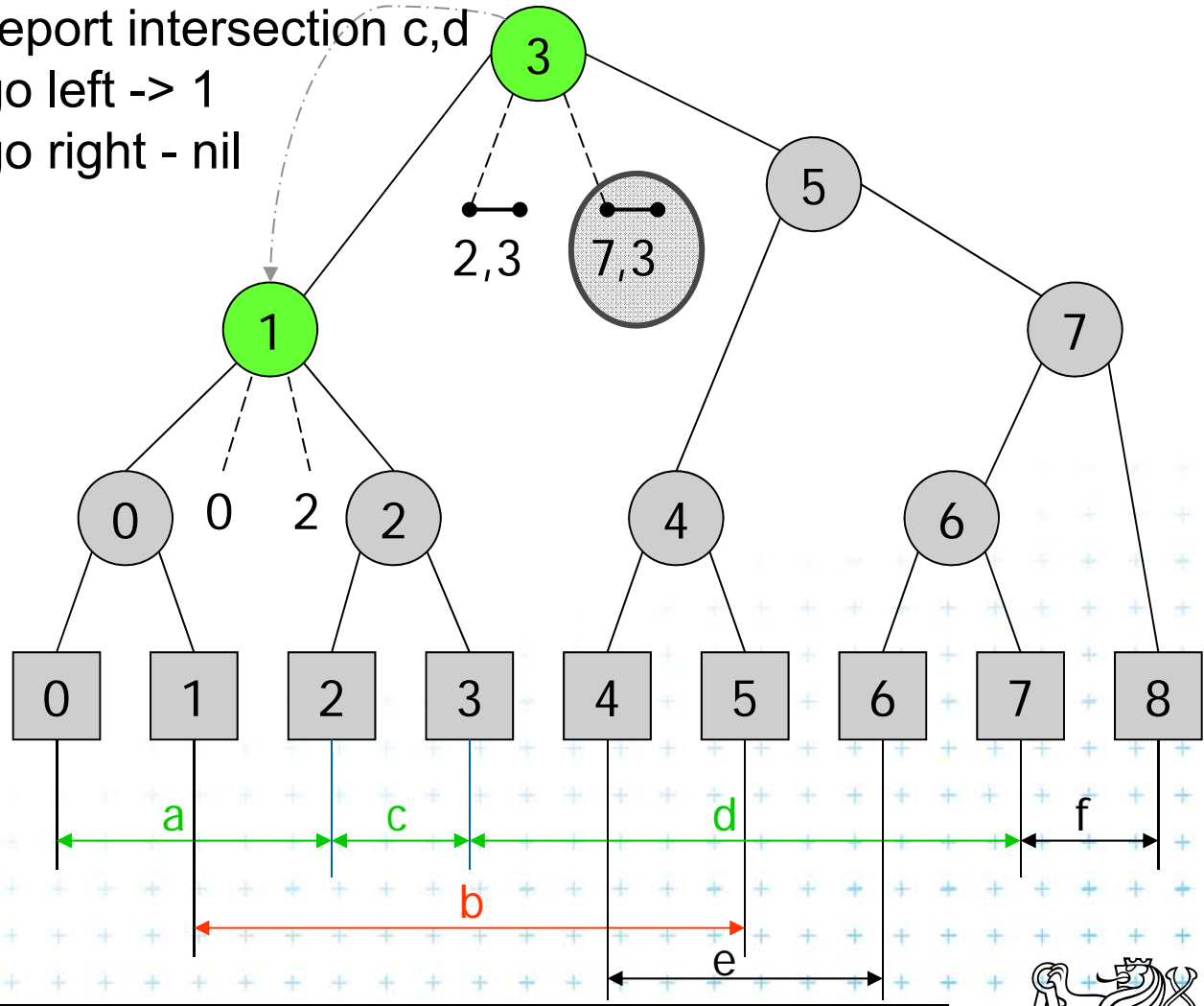


for (all in MR(v))

=> report intersection c,d

go left -> 1

go right - nil

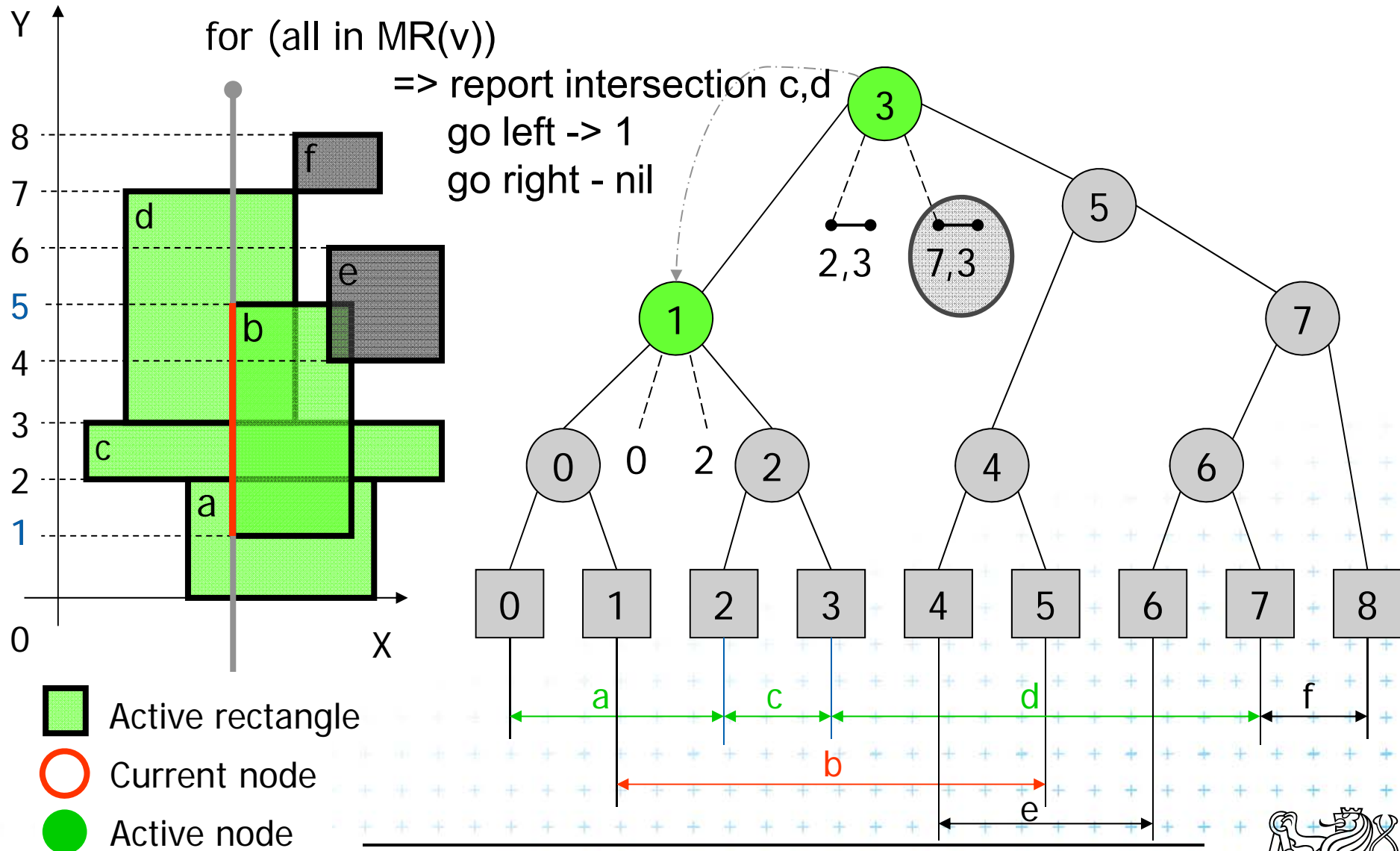


- Active rectangle
- Current node
- Active node



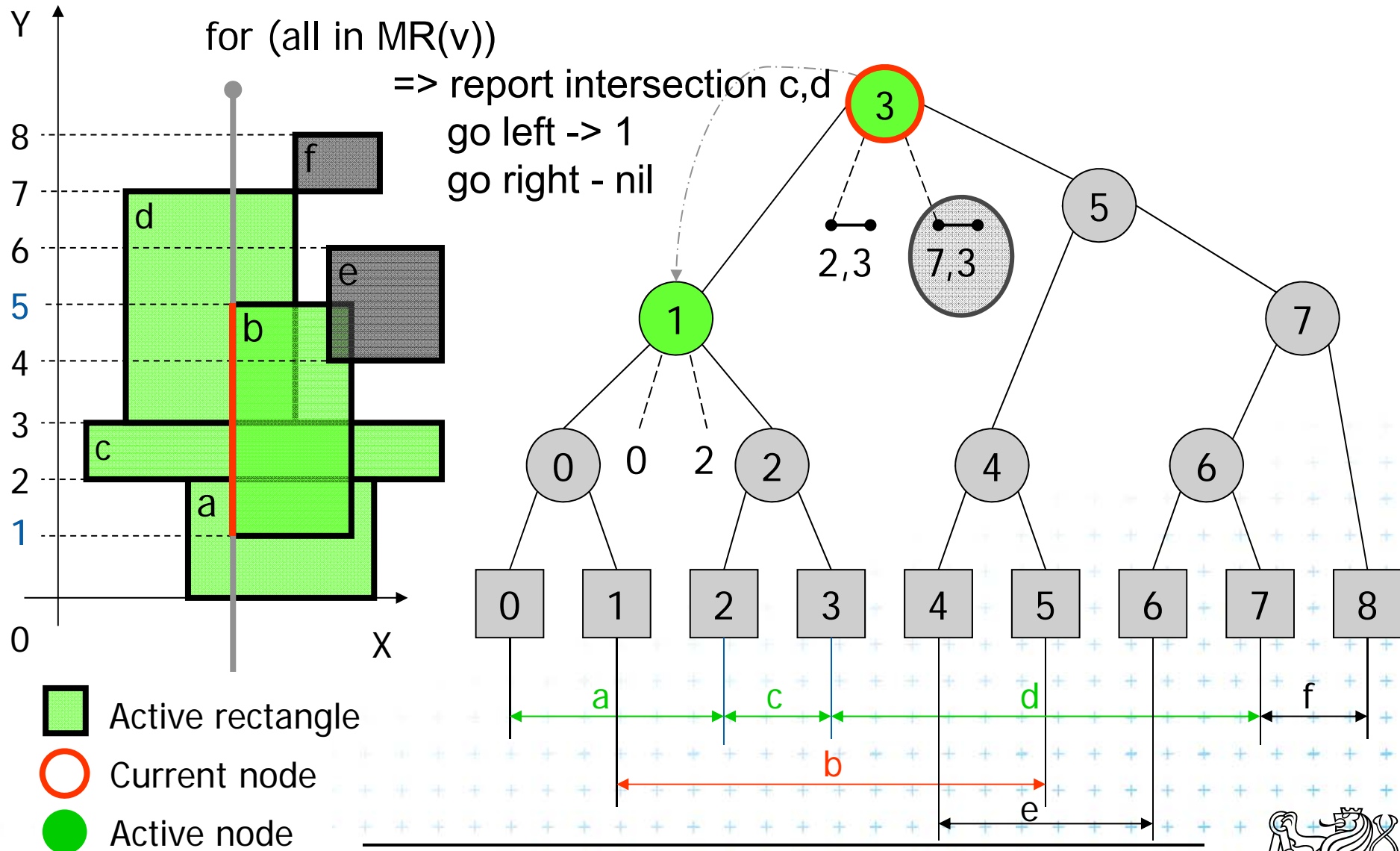
# Insert [1,5] a) Query Interval 1/2

$$b < H(v) < e$$



# Insert [1,5] a) Query Interval 1/2

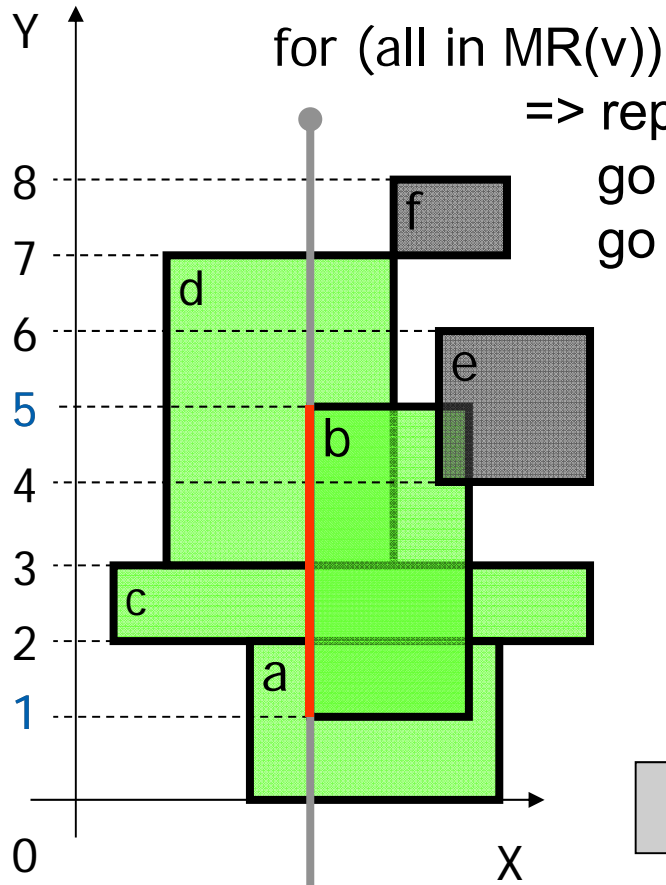
$$b < H(v) < e$$



# Insert [1,5] a) Query Interval 1/2

$$b < H(v) < e$$

$$? 1 < \textcircled{3} < 5 ?$$

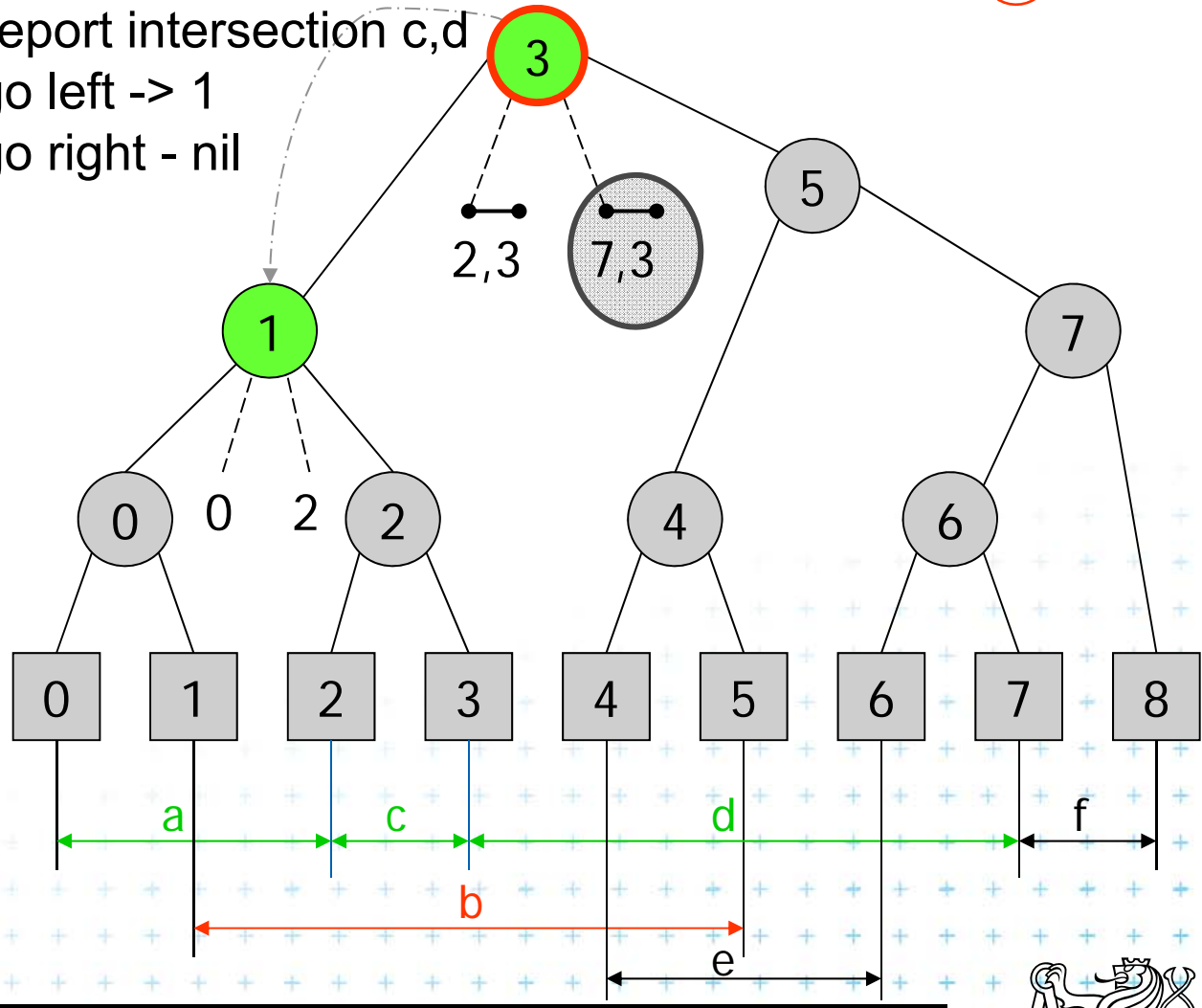





for (all in MR(v))

=> report intersection c,d

go left -> 1

go right - nil



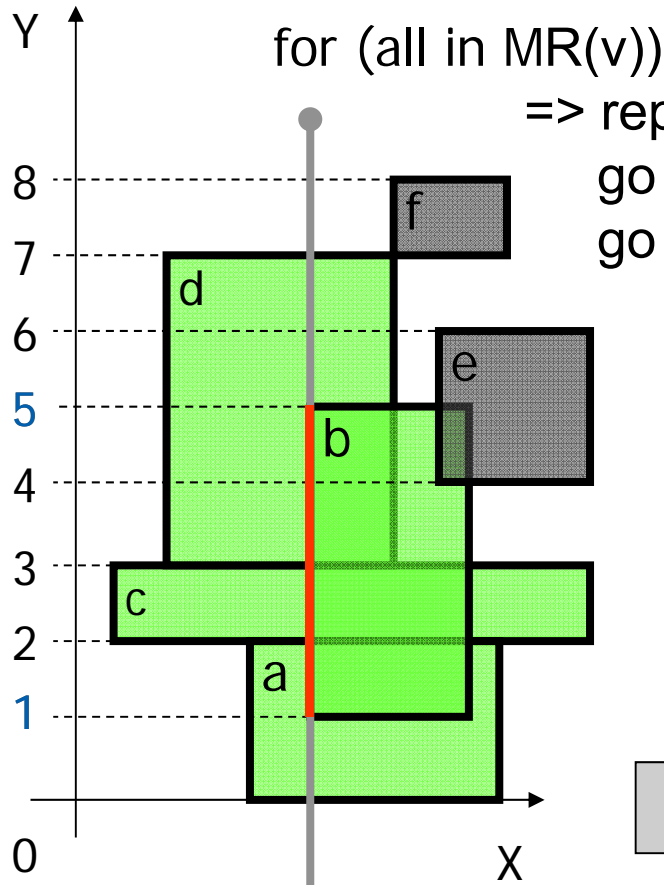
-  Active rectangle
-  Current node
-  Active node



# Insert [1,5] a) Query Interval 1/2

$$b < H(v) < e$$

$$? 1 < \textcircled{3} < 5 ?$$

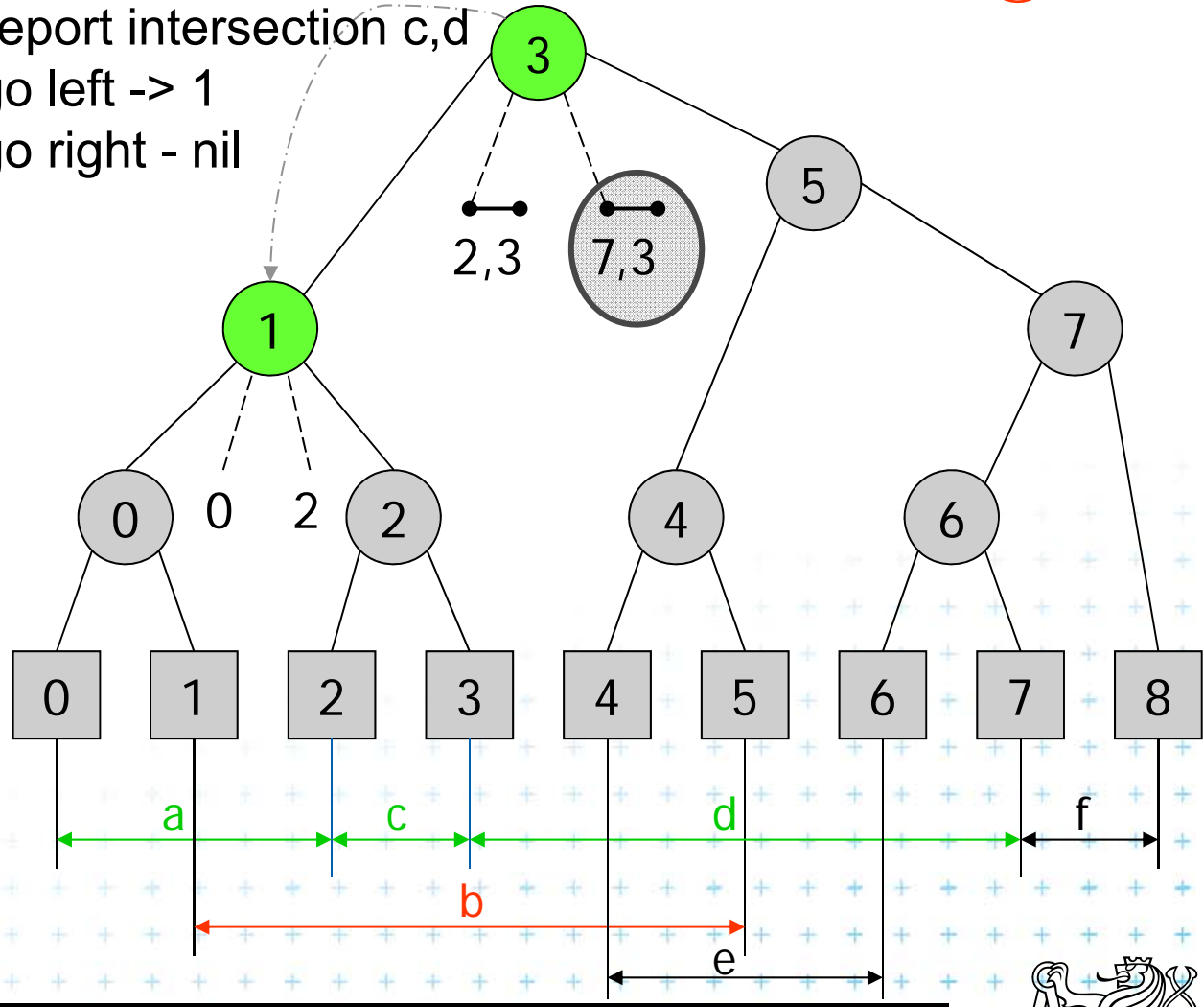


for (all in MR(v))

=> report intersection c,d

go left -> 1

go right - nil



Active rectangle

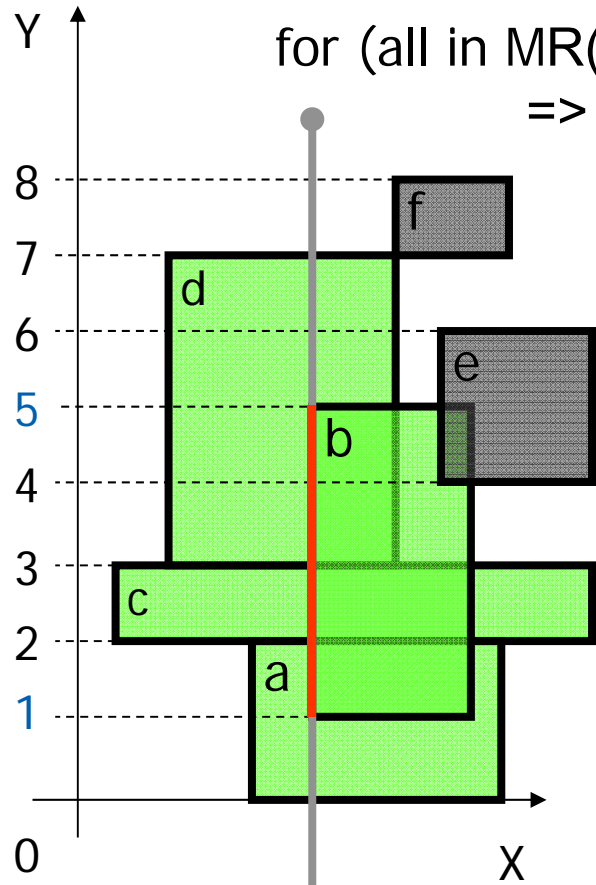
Current node

Active node



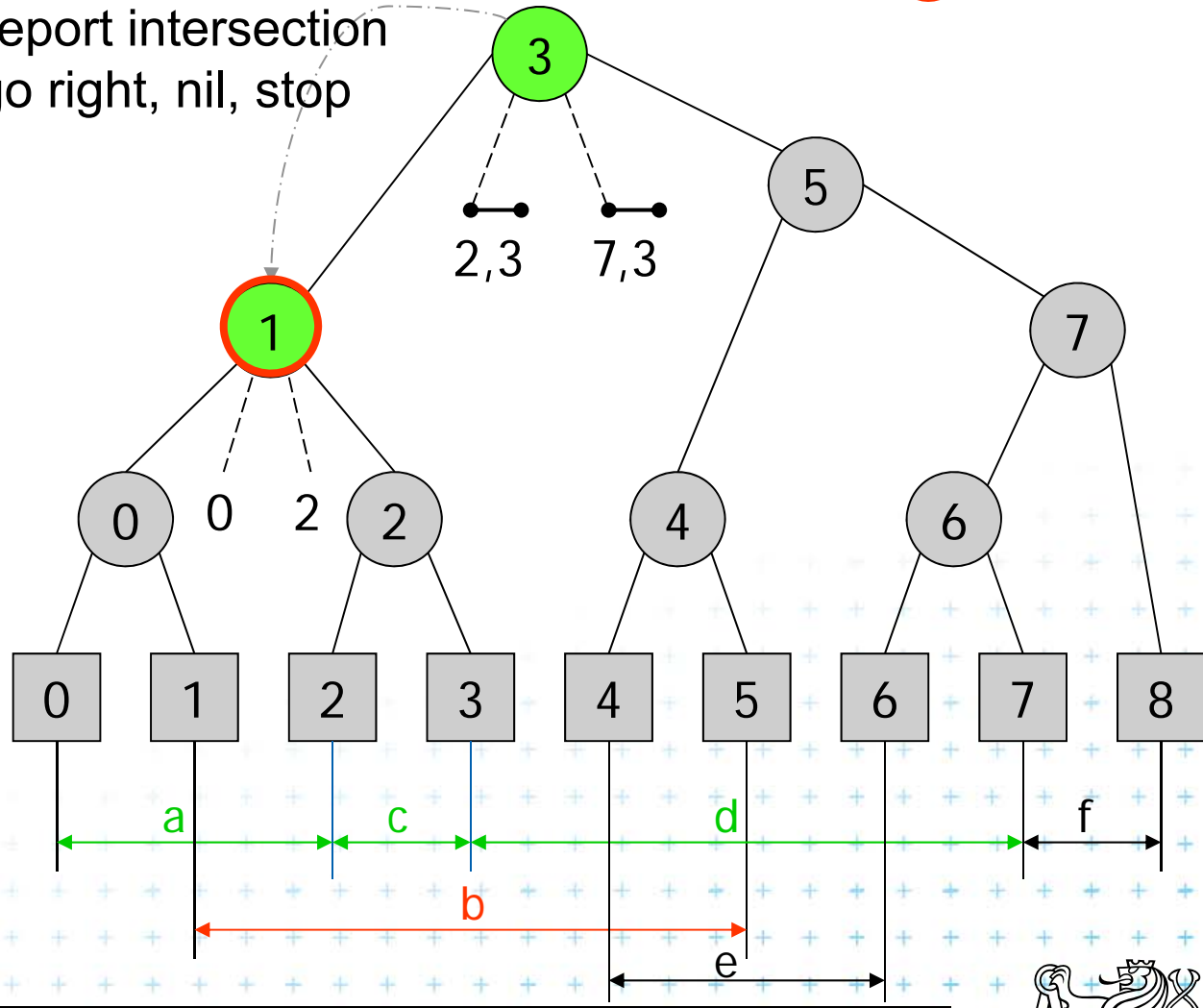
# Insert [1,5] a) Query Interval 2/2

$$H(v) \leq b < e$$



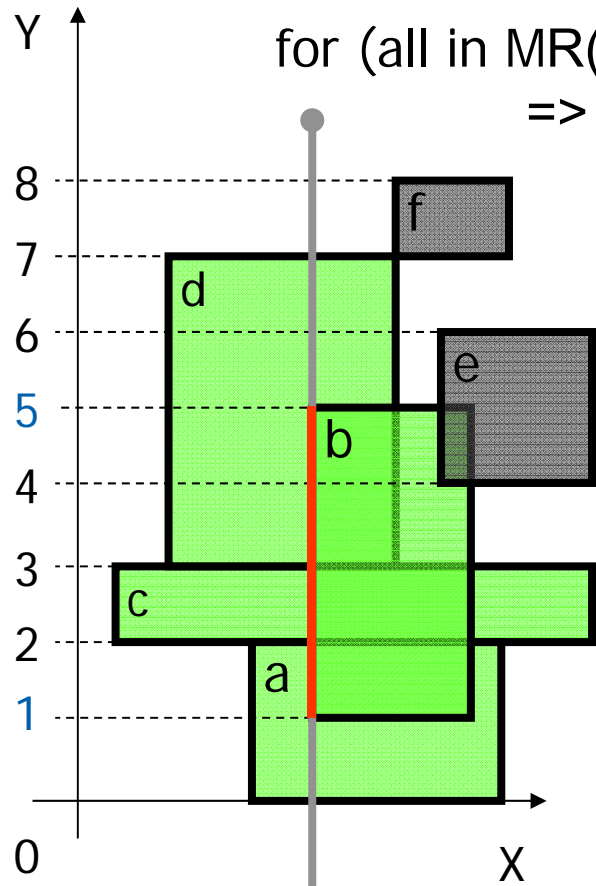
for (all in MR(v)) test  $MR(v)[i] \geq 1$   
 $\Rightarrow$  report intersection  
 go right, nil, stop

$$? 1 \leq 1 < 5 ?$$



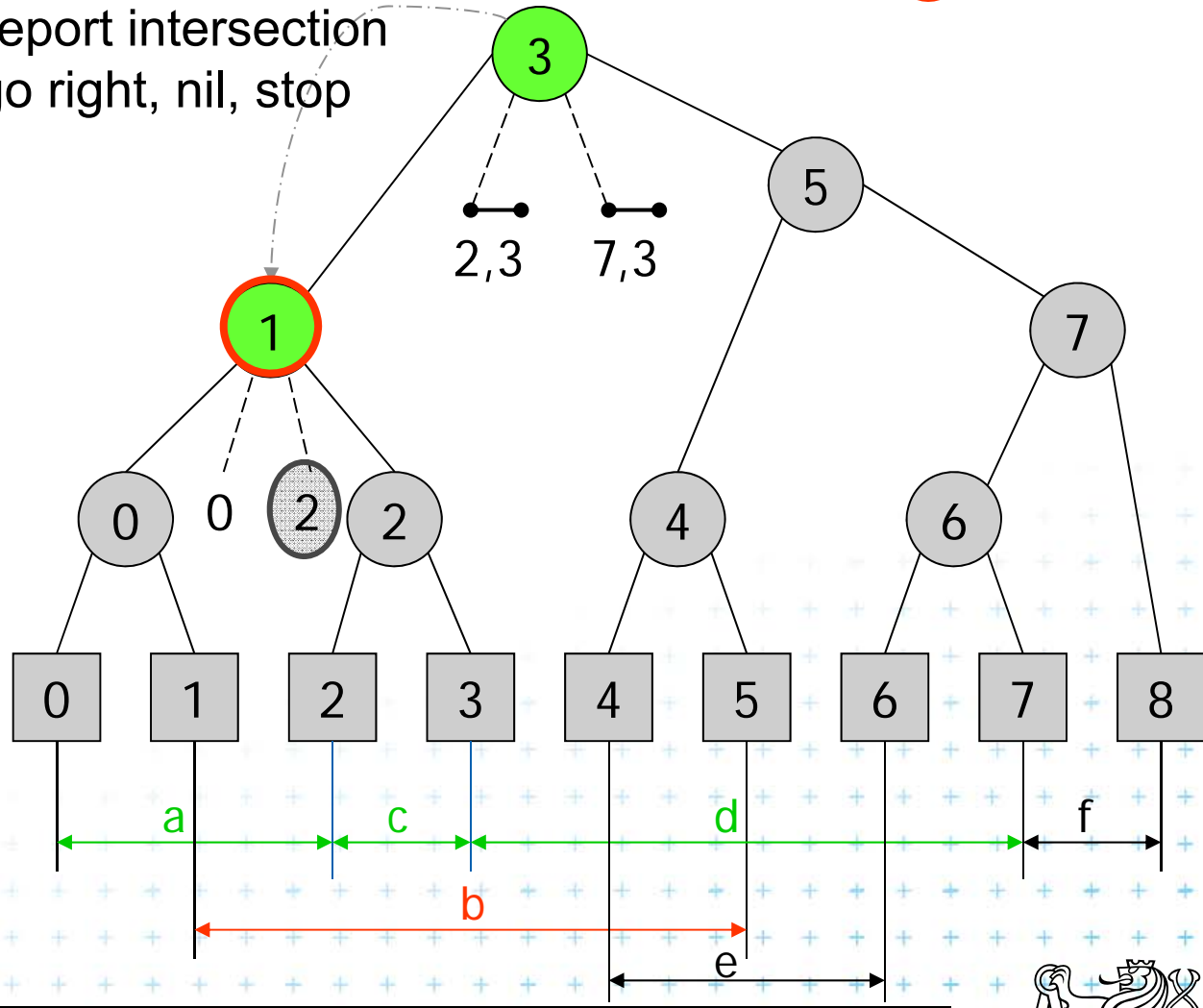
# Insert [1,5] a) Query Interval 2/2

$$H(v) \leq b < e$$



for (all in MR(v)) test  $MR(v)[i] \geq 1$   
 $\Rightarrow$  report intersection  
 go right, nil, stop

$$? 1 \leq 1 < 5 ?$$



- Active rectangle
- Current node
- Active node

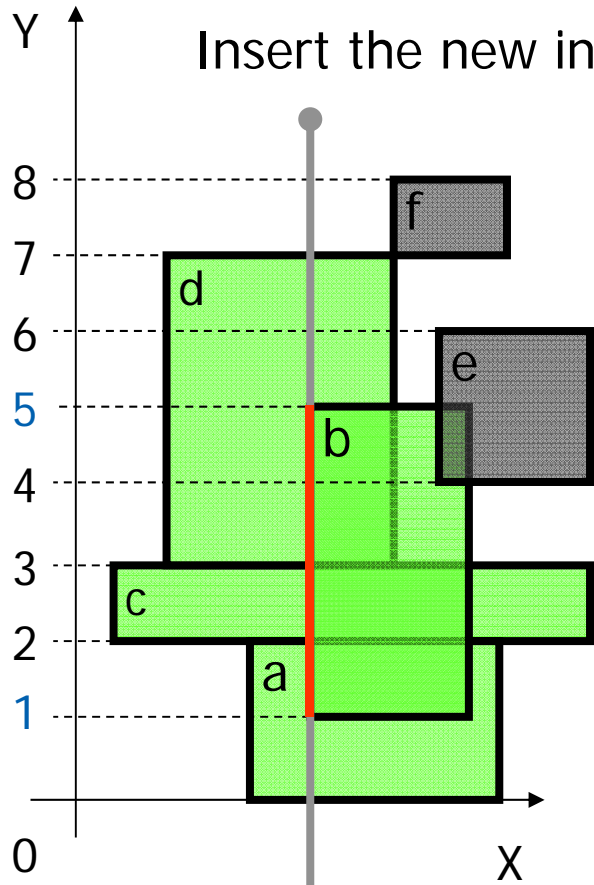




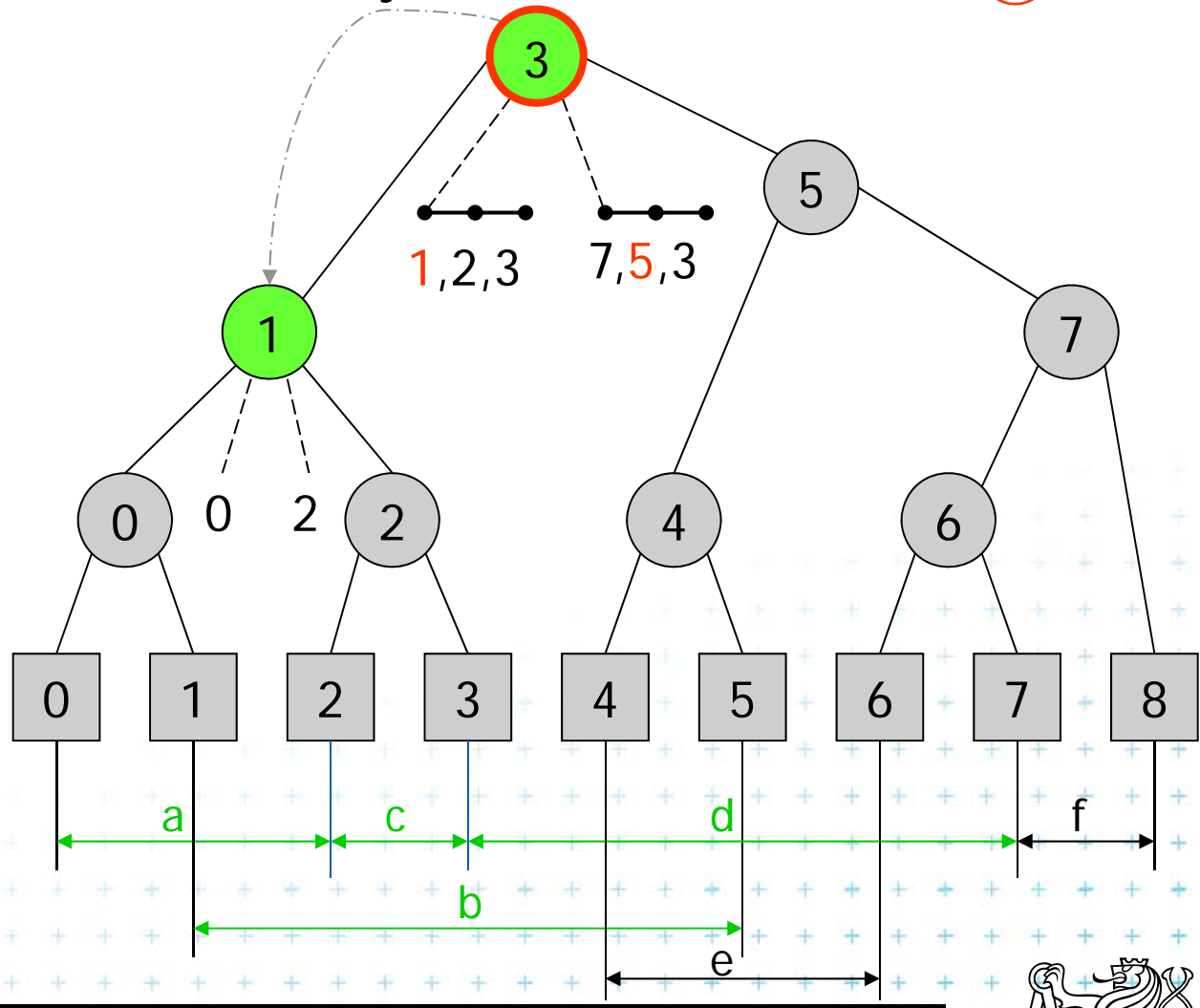
# Insert [1,5] b) Insert Interval




$$b \leq H(v) \leq e$$

$$? 1 \leq 3 \leq 5 ?$$



Insert the new interval to secondary lists

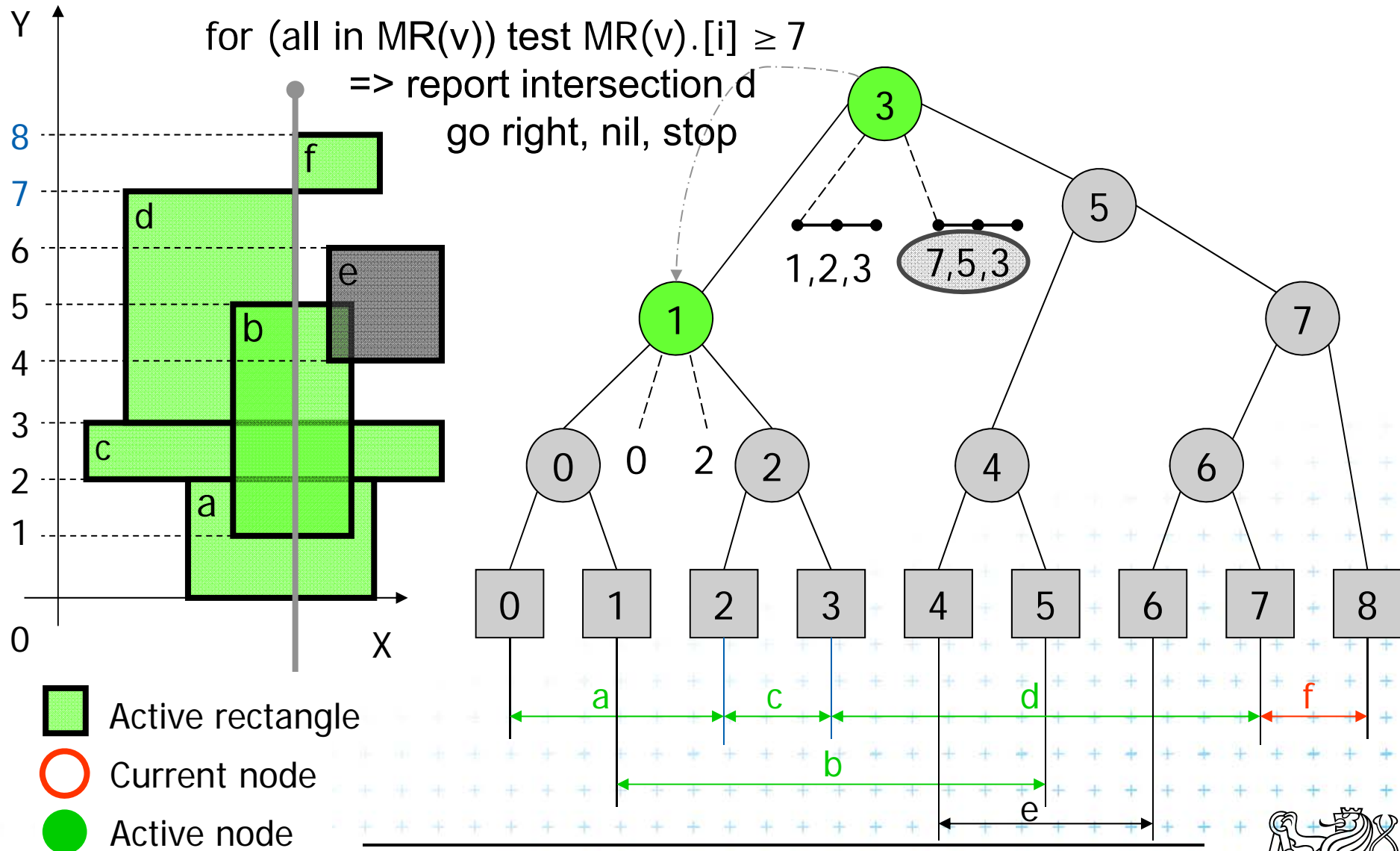


-  Active rectangle
-  Current node
-  Active node



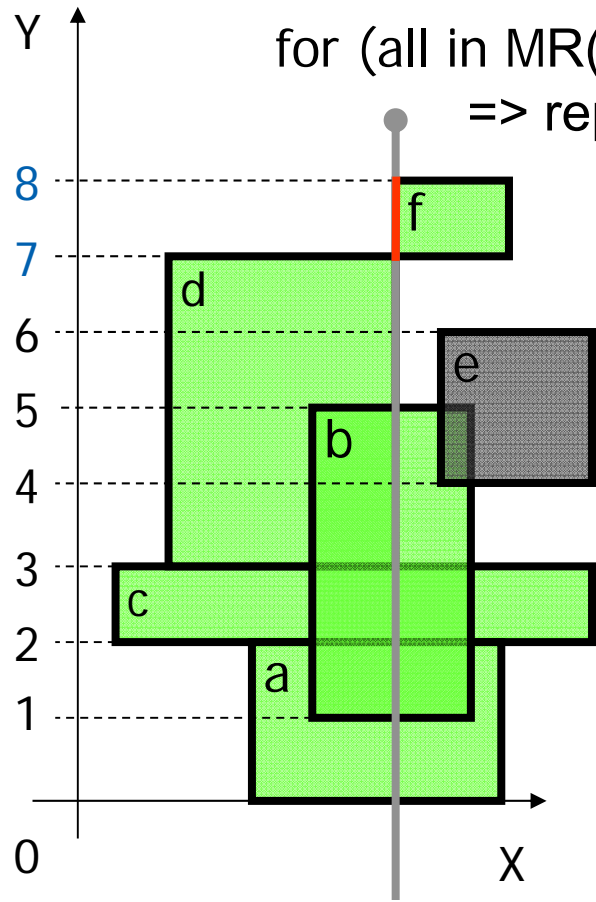
# Insert [7,8] a) Query Interval

$$H(v) \leq b < e$$

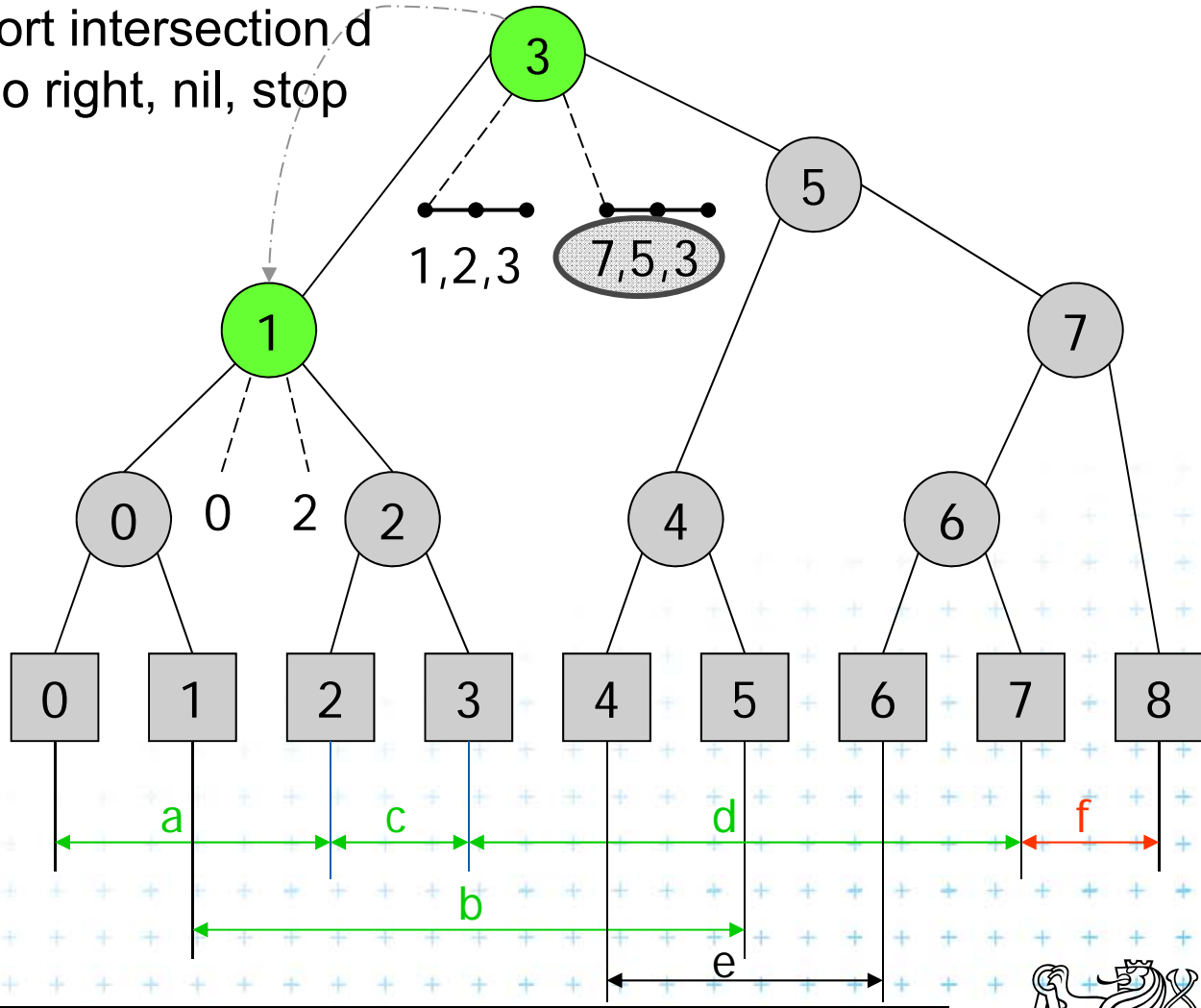


# Insert [7,8] a) Query Interval

$$H(v) \leq b < e$$



for (all in MR(v)) test MR(v).[i]  $\geq 7$   
 $\Rightarrow$  report intersection d  
 go right, nil, stop

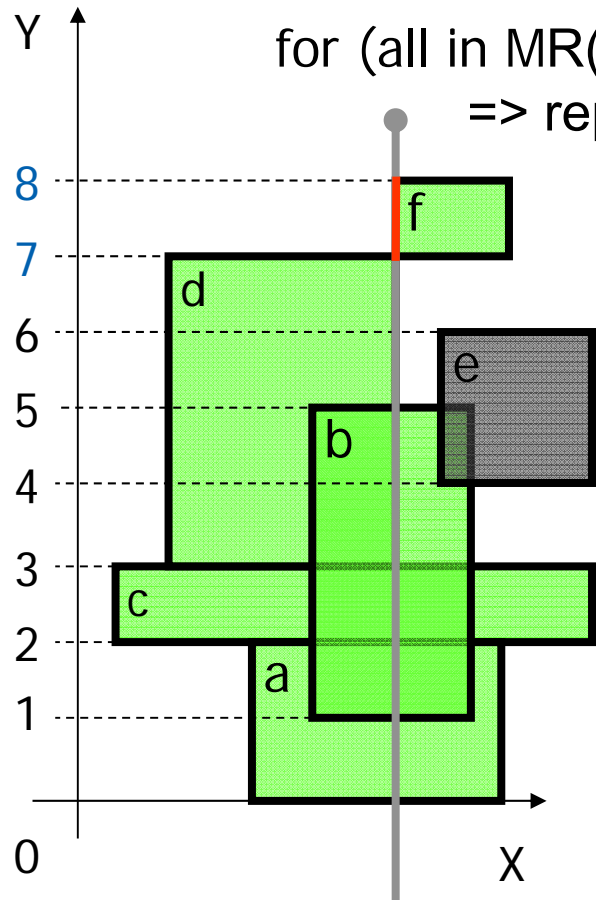


- Active rectangle
- Current node
- Active node

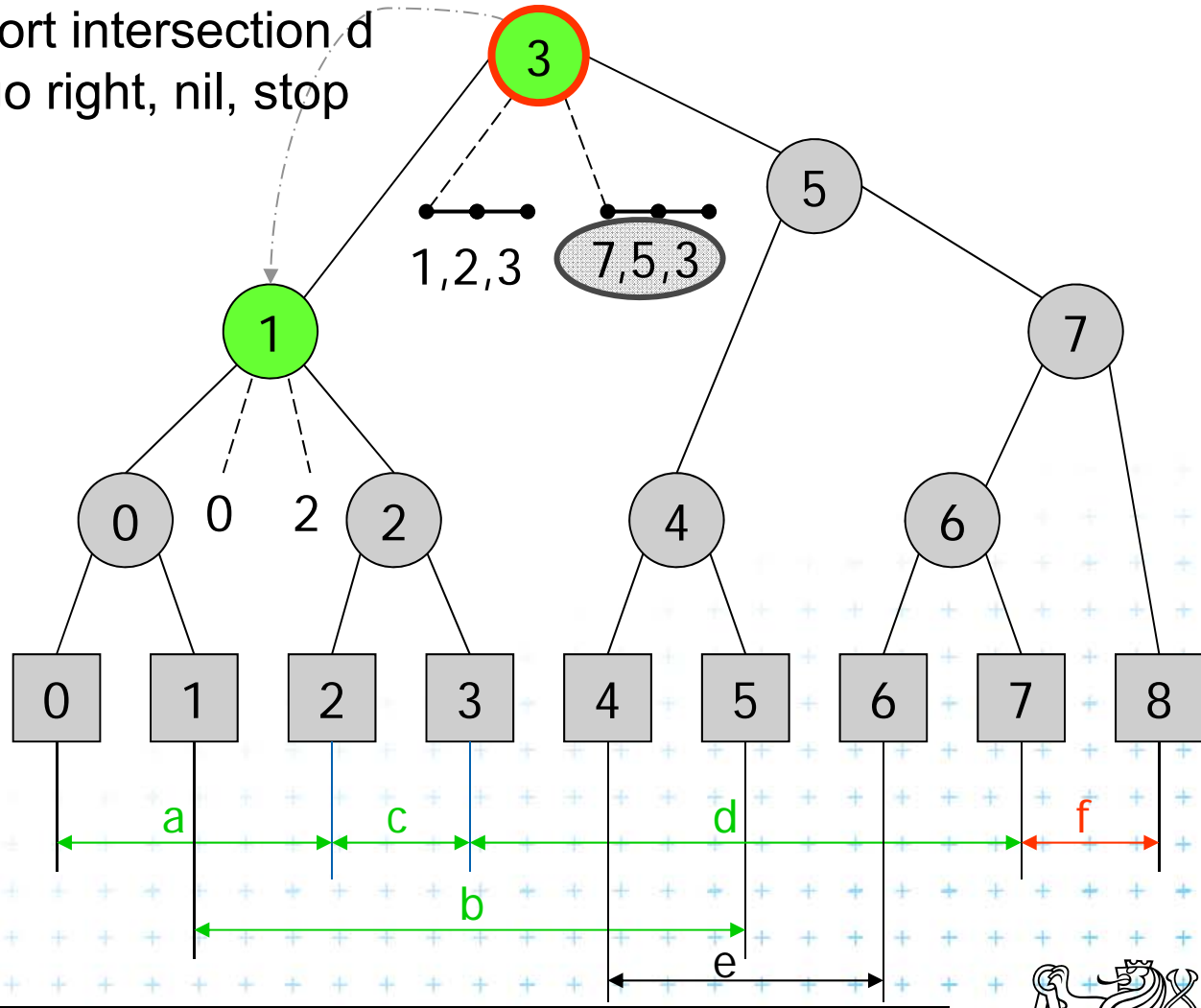





# Insert [7,8] a) Query Interval

$$H(v) \leq b < e$$



for (all in MR(v)) test MR(v).[i]  $\geq 7$   
 $\Rightarrow$  report intersection d  
 go right, nil, stop

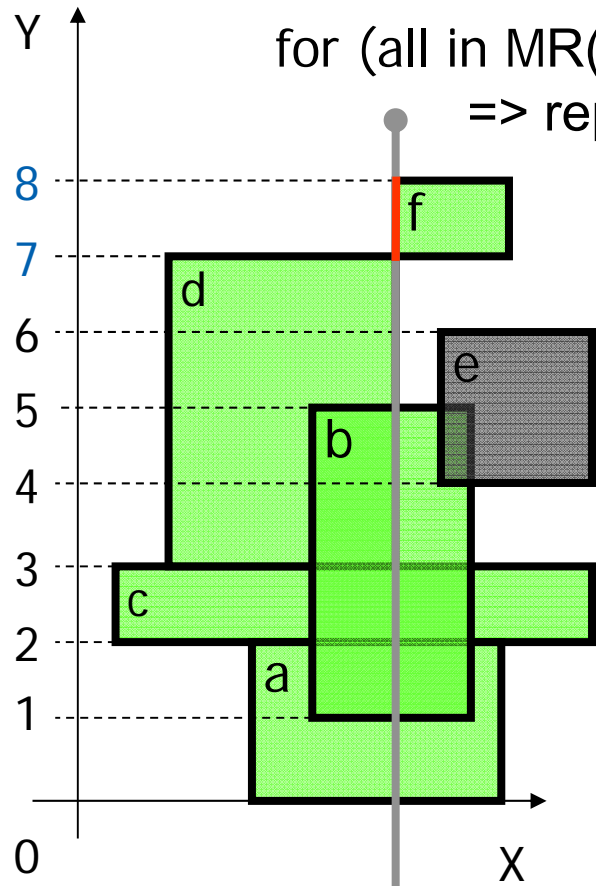


-  Active rectangle
-  Current node
-  Active node



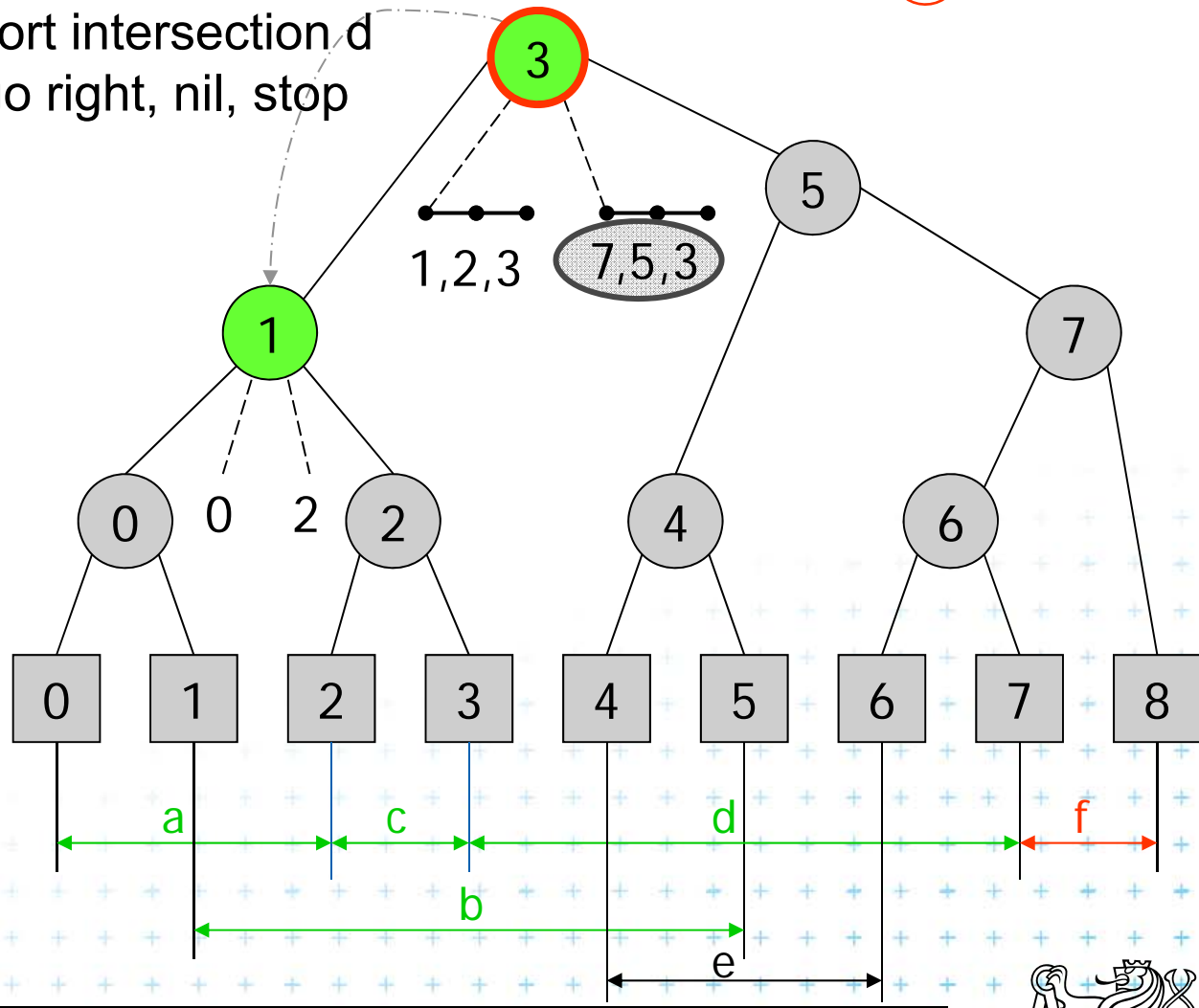
# Insert [7,8] a) Query Interval

$$H(v) \leq b < e$$



for (all in MR(v)) test  $MR(v).[i] \geq 7$   
 $\Rightarrow$  report intersection d  
 go right, nil, stop

$$? 3 \leq 7 < 8 ?$$

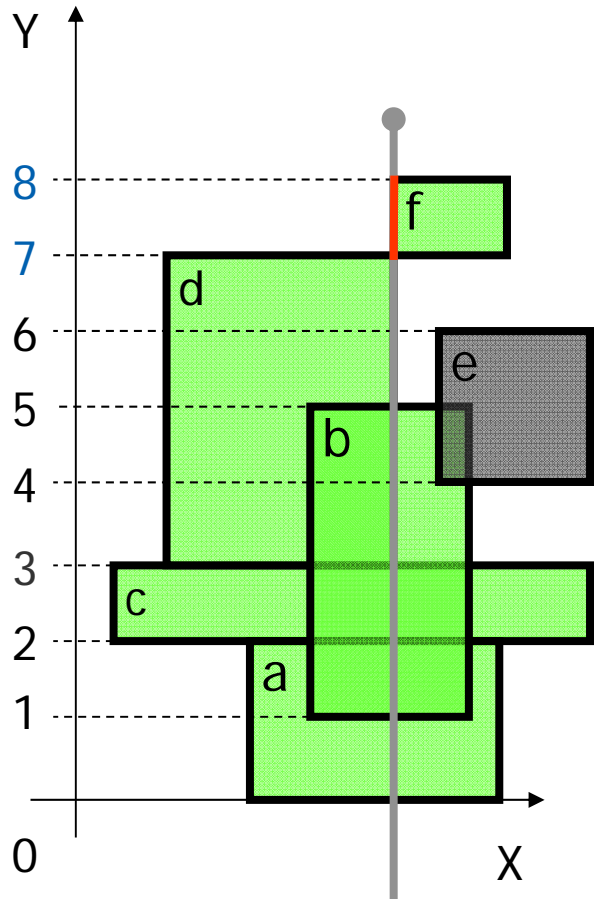





- Active rectangle
- Current node
- Active node

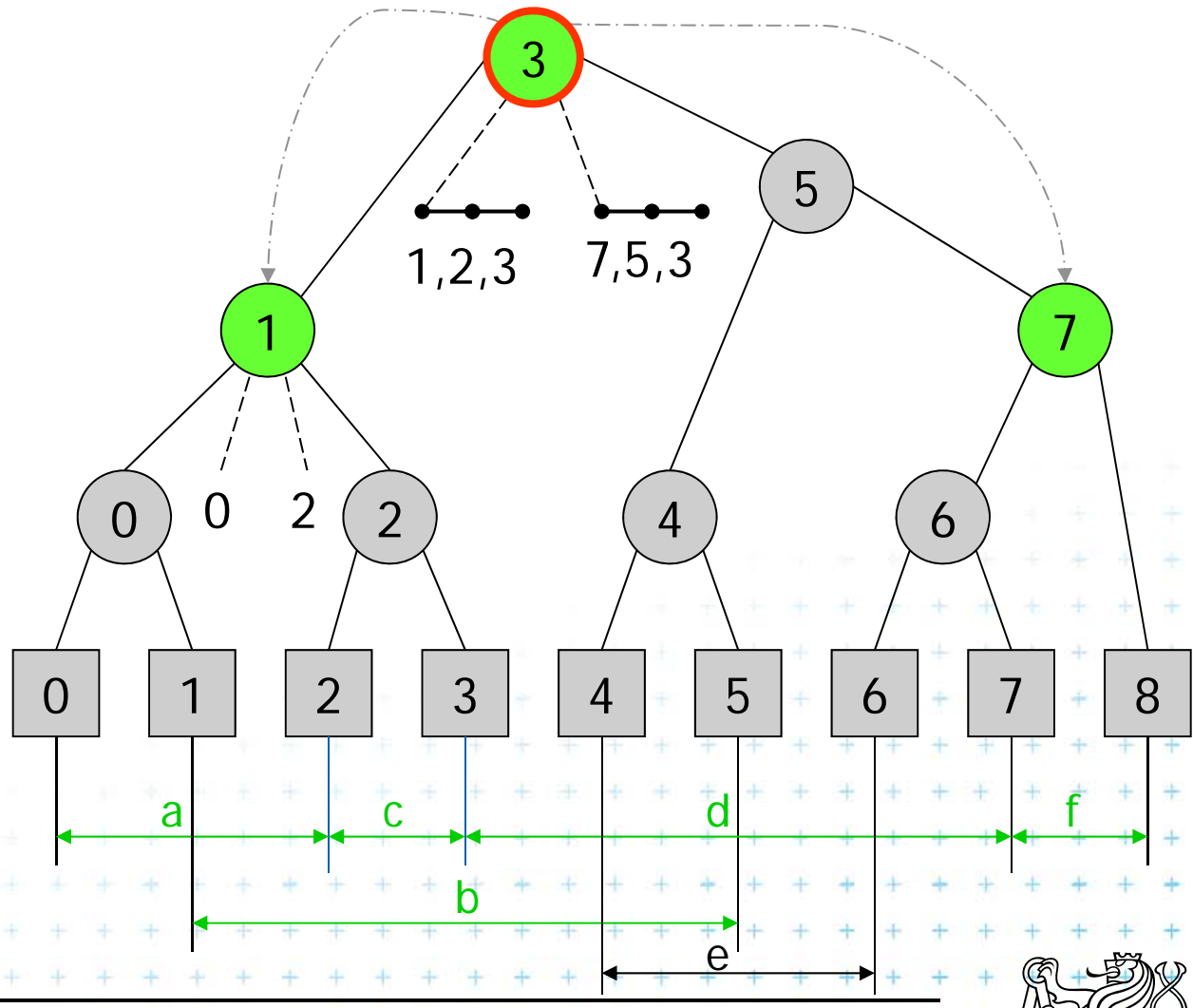


# Insert [7,8] b) Insert Interval

$$b \leq H(v) \leq e$$

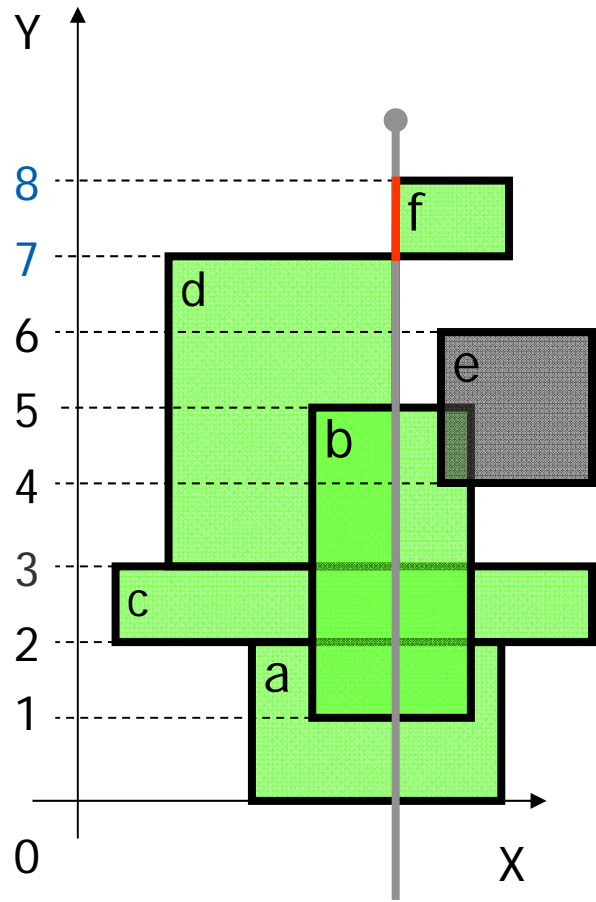


-  Active rectangle
-  Current node
-  Active node

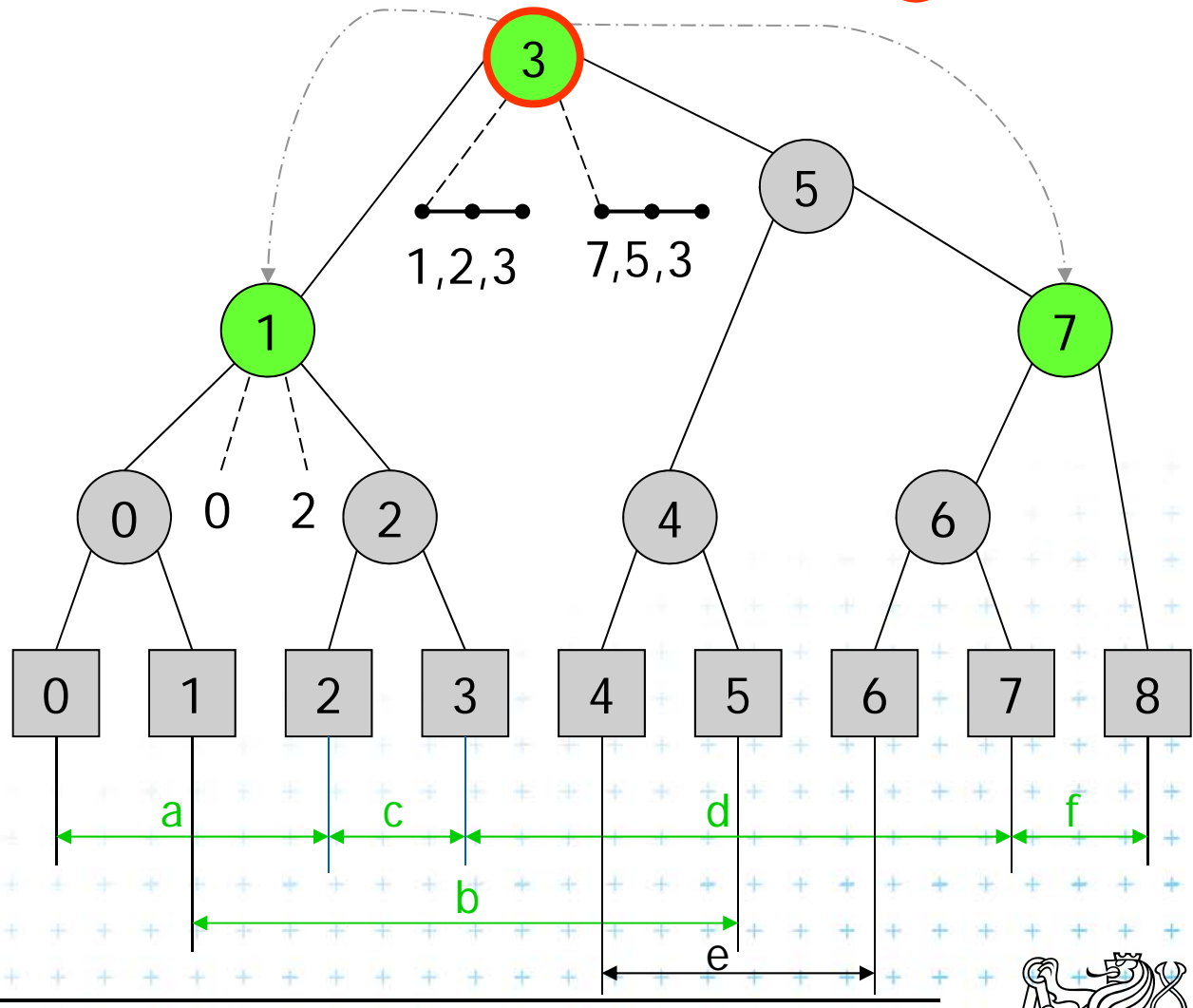


# Insert [7,8] b) Insert Interval

$$b \leq H(v) \leq e$$



right  $\leq$  ? 3  $\leq$  7 < 8 ?

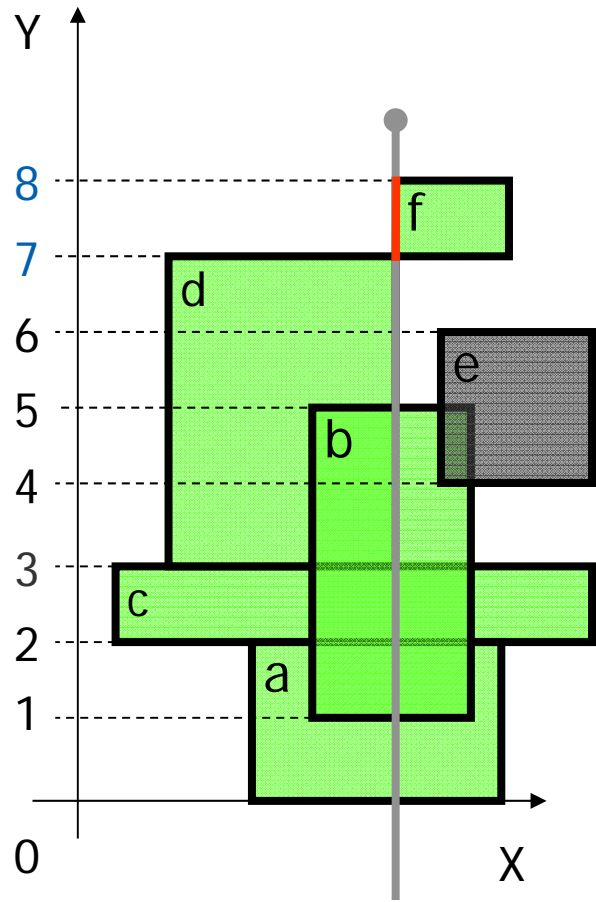


- Active rectangle
- Current node
- Active node

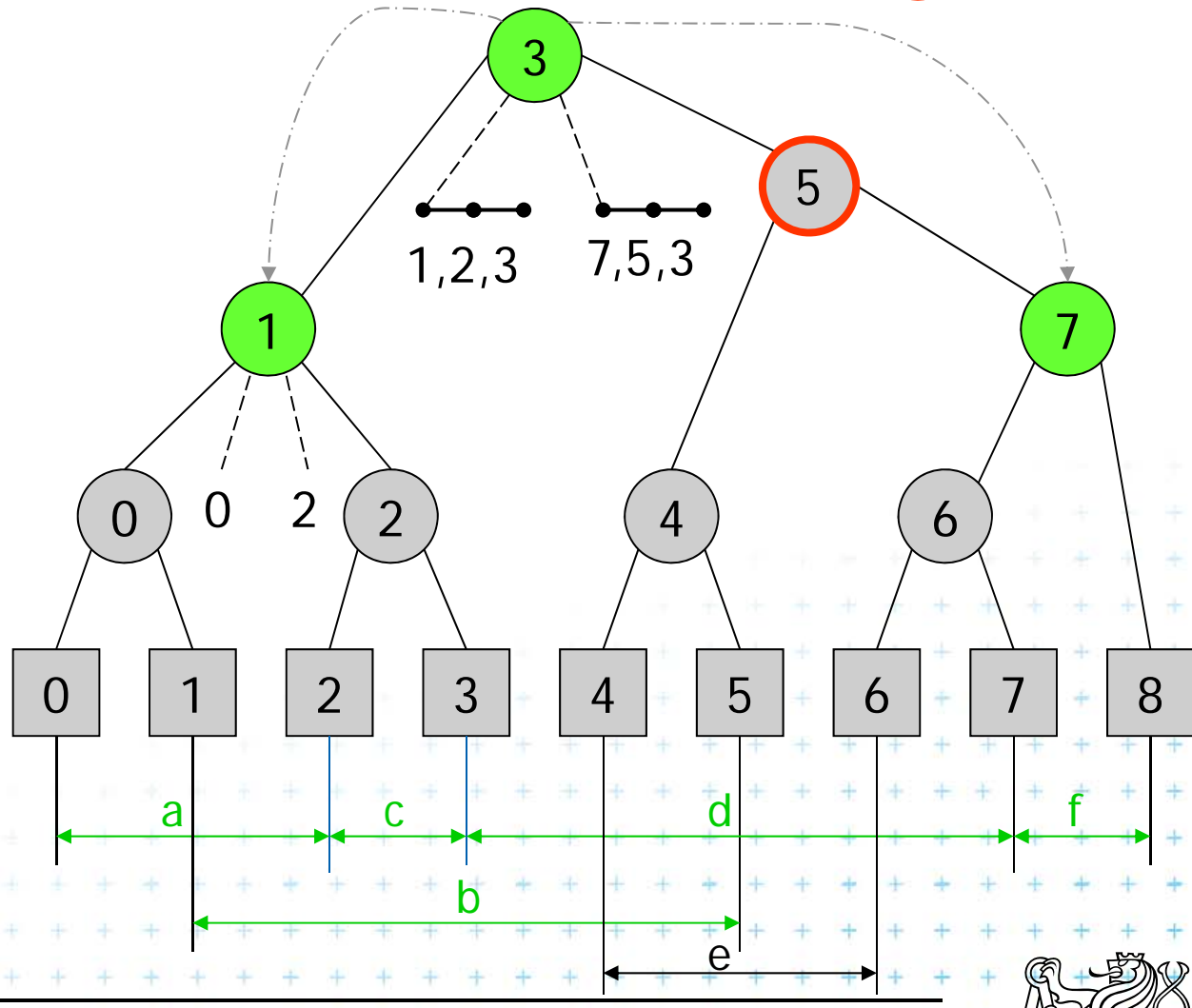


# Insert [7,8] b) Insert Interval

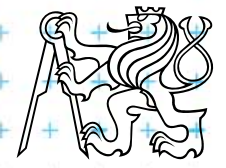
$$b \leq H(v) \leq e$$



right  $\leq$  ? 3  $\leq$  7 < 8 ?



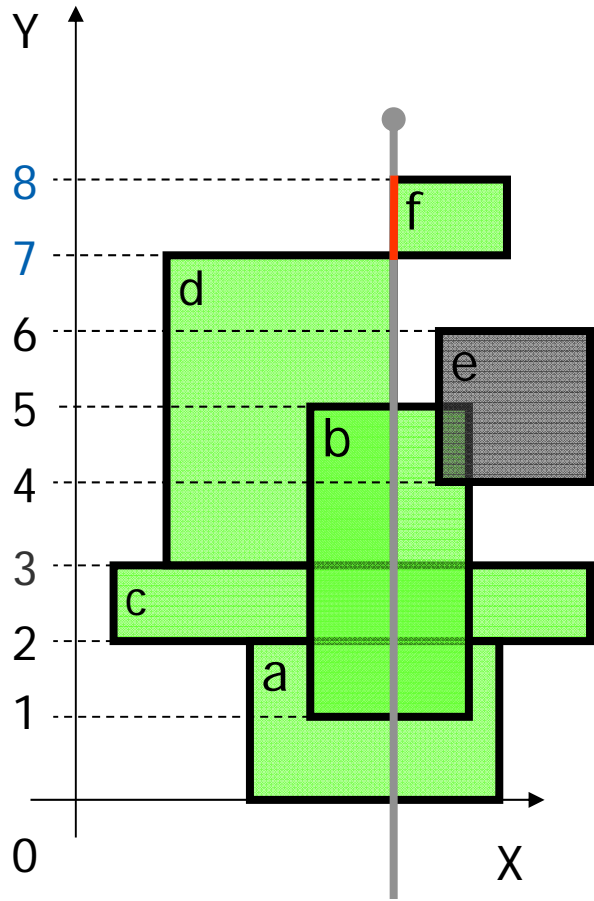
- Active rectangle
- Current node
- Active node






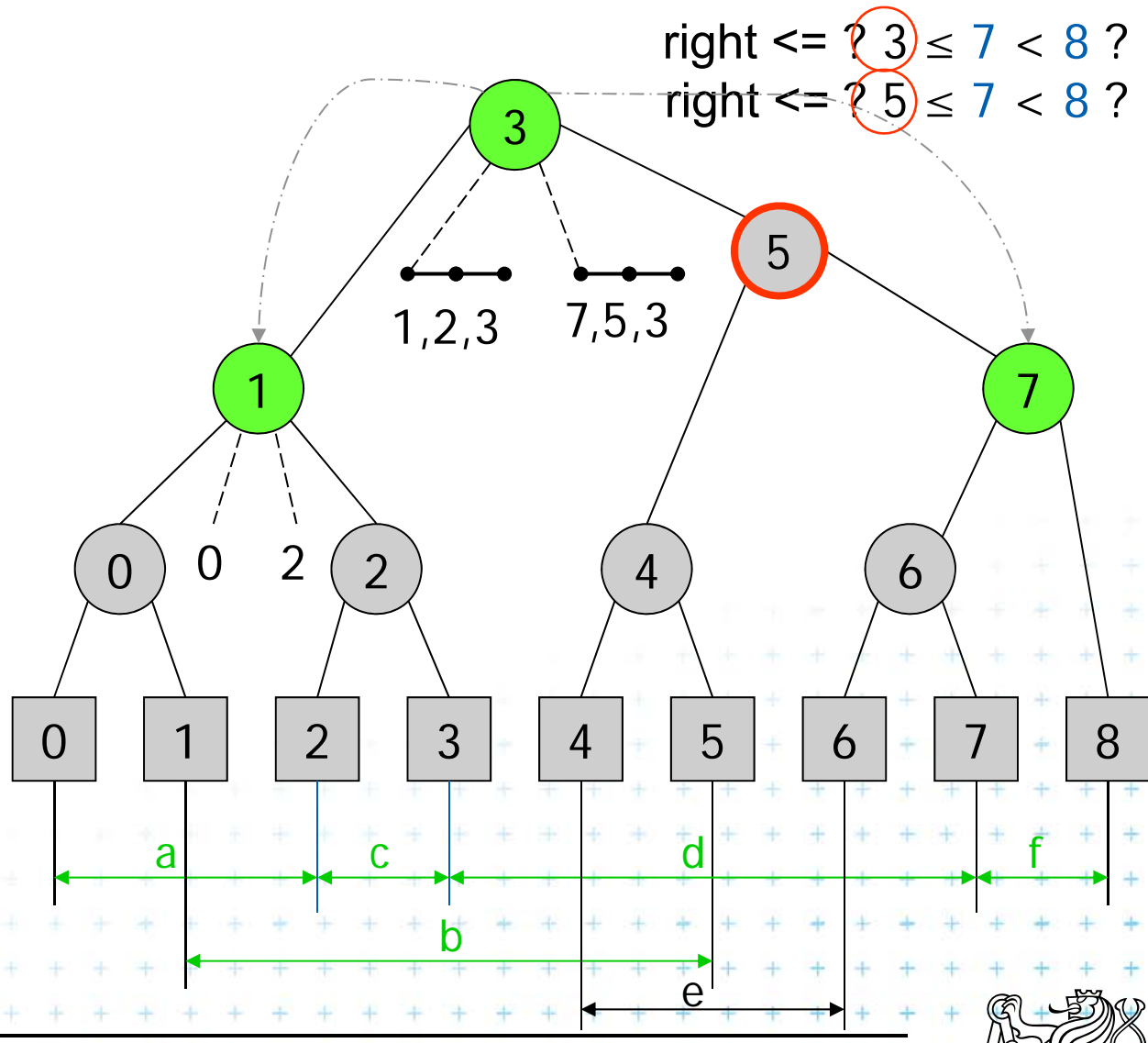


# Insert [7,8] b) Insert Interval

$$b \leq H(v) \leq e$$

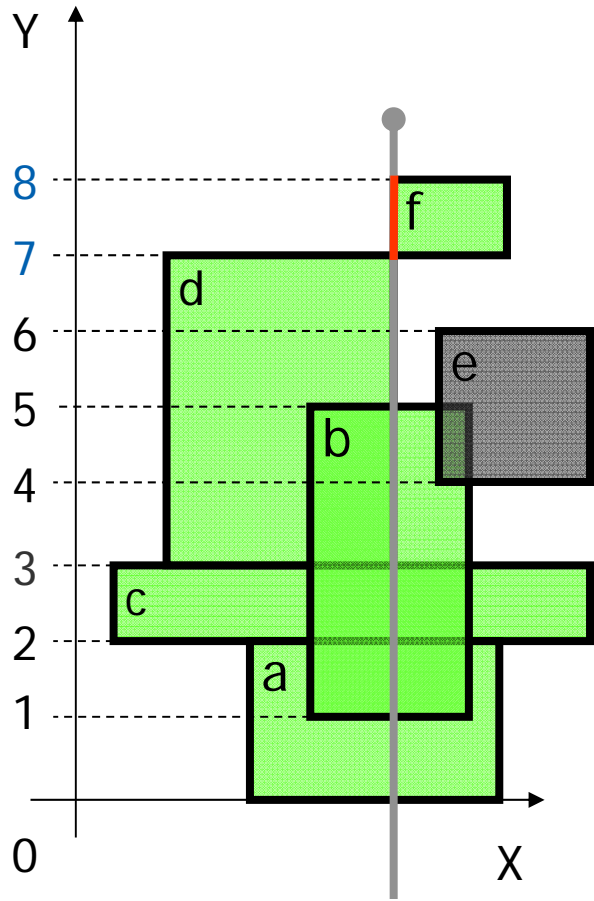


-  Active rectangle
-  Current node
-  Active node

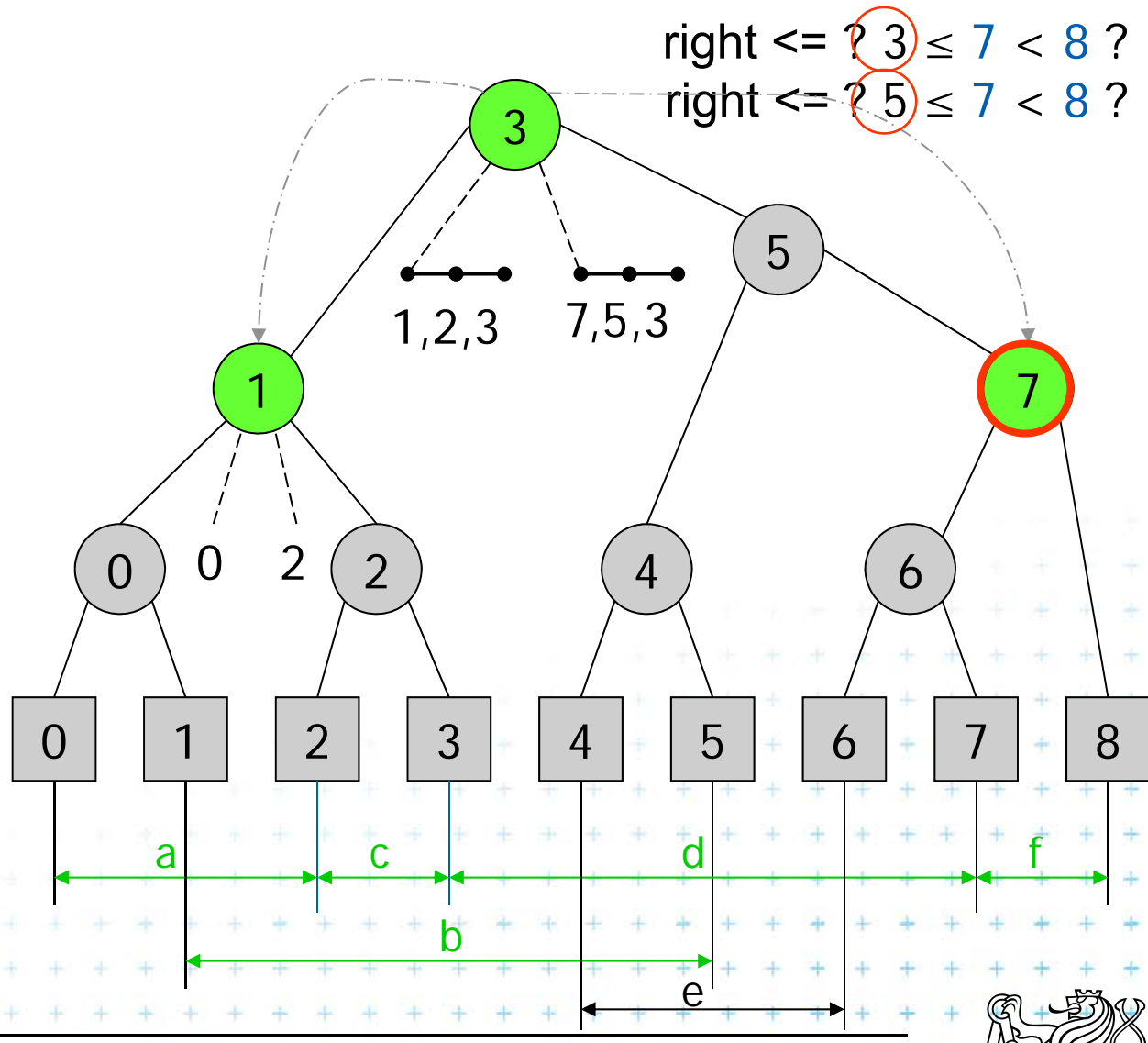


# Insert [7,8] b) Insert Interval

$$b \leq H(v) \leq e$$

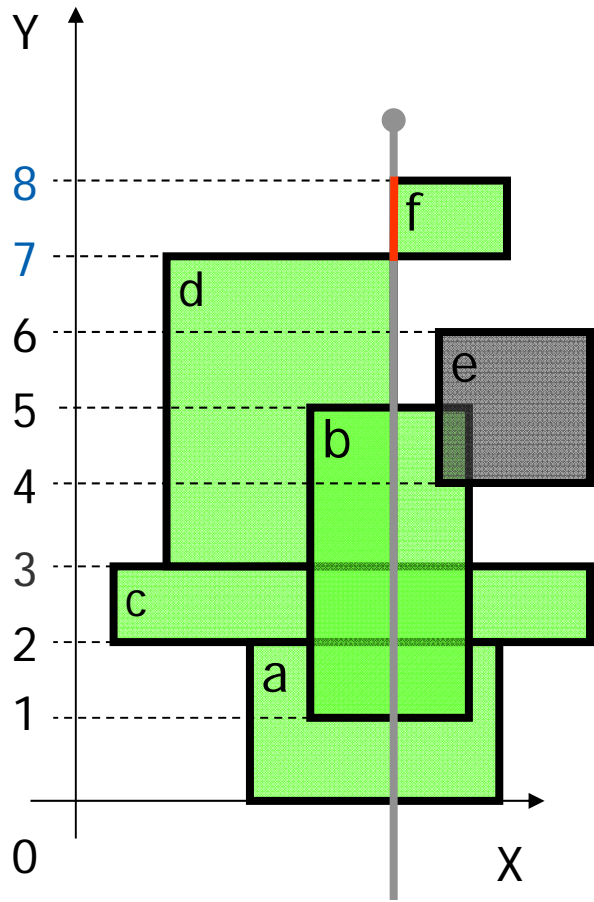


- Active rectangle
- Current node
- Active node

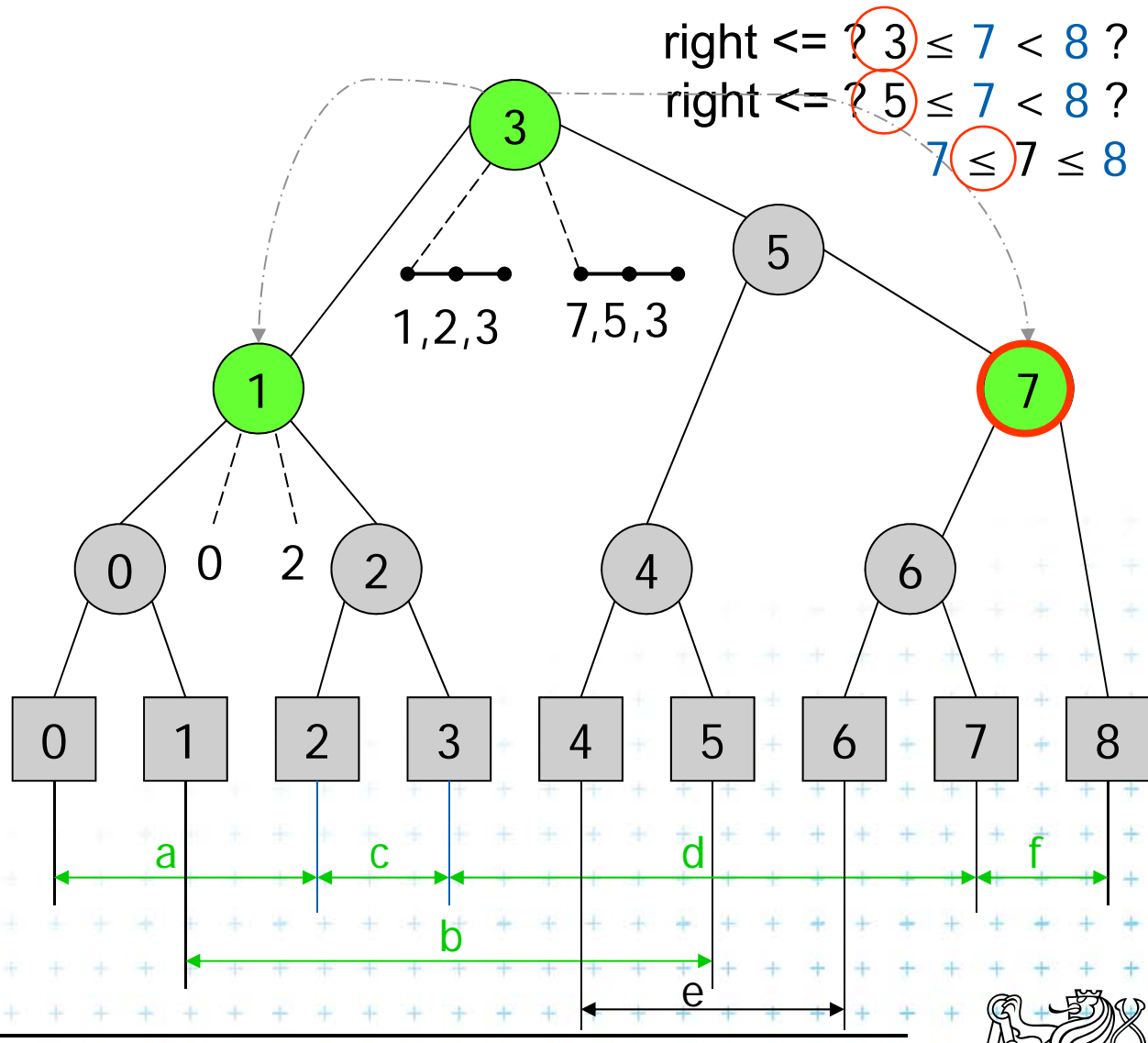


# Insert [7,8] b) Insert Interval

$$b \leq H(v) \leq e$$

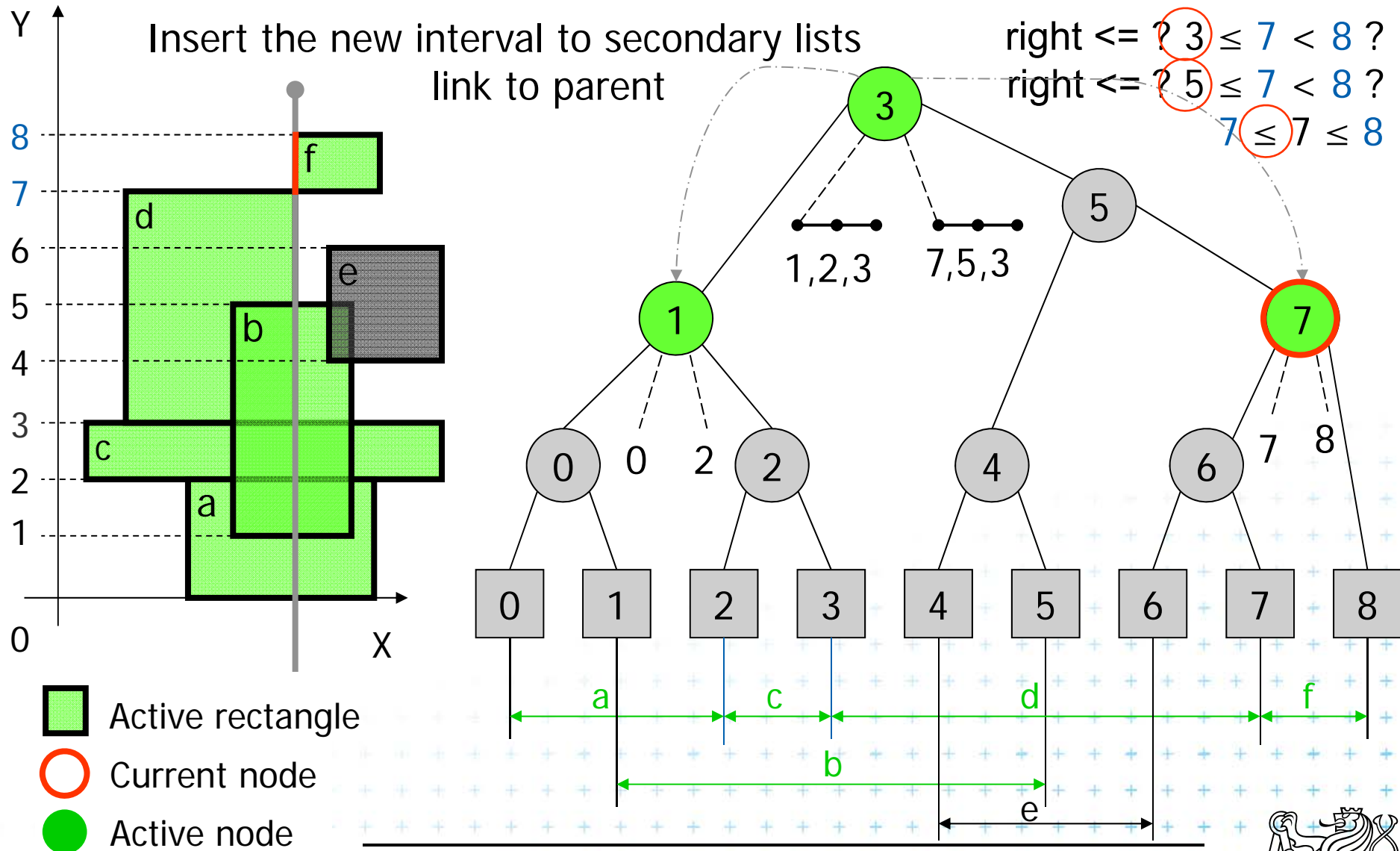


- Active rectangle
- Current node
- Active node



# Insert [7,8] b) Insert Interval

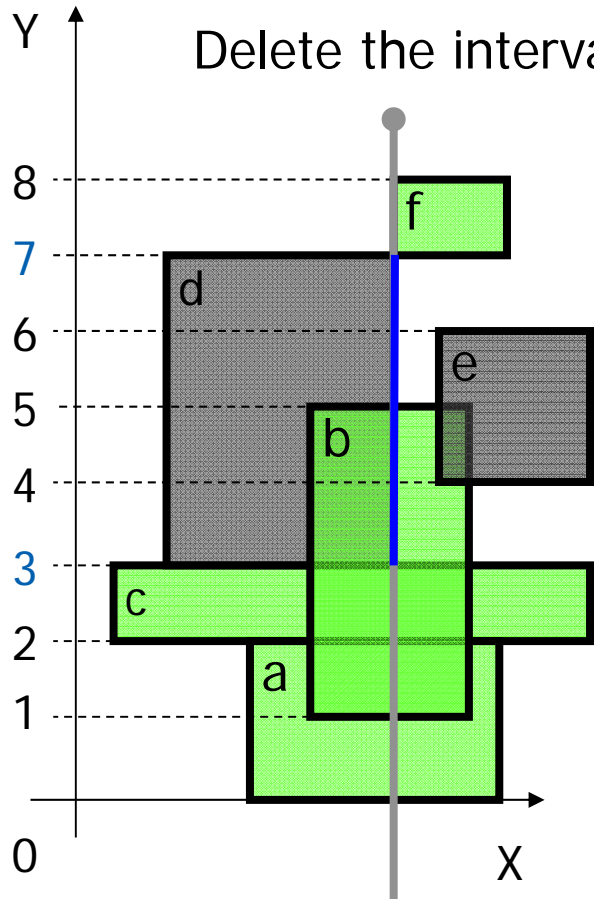
$$b \leq H(v) \leq e$$



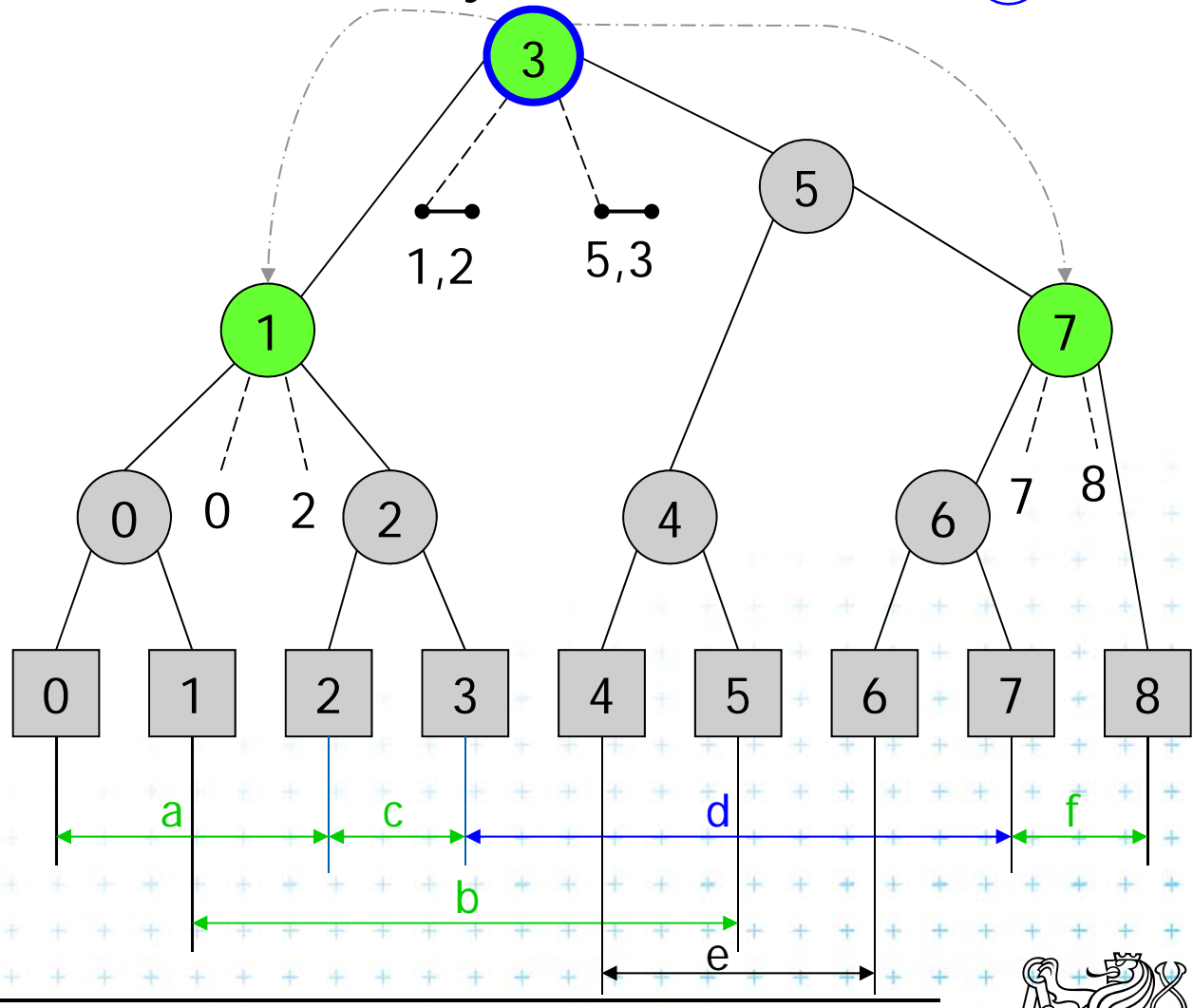
# Delete [3,7] Delete Interval

$$b \leq H(v) \leq e$$

$$? 3 \leq 7 \leq 8 ?$$



Delete the interval [3,7] from secondary lists

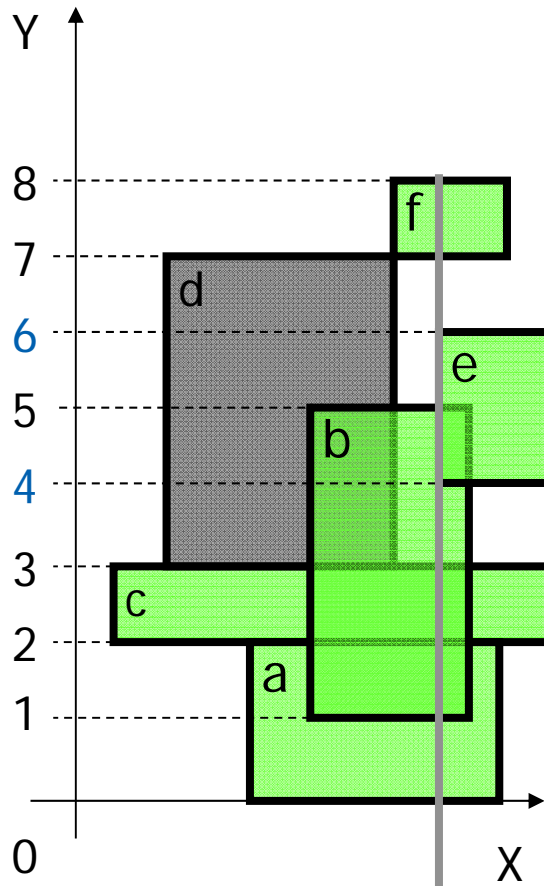





- Active rectangle
- Current node
- Active node

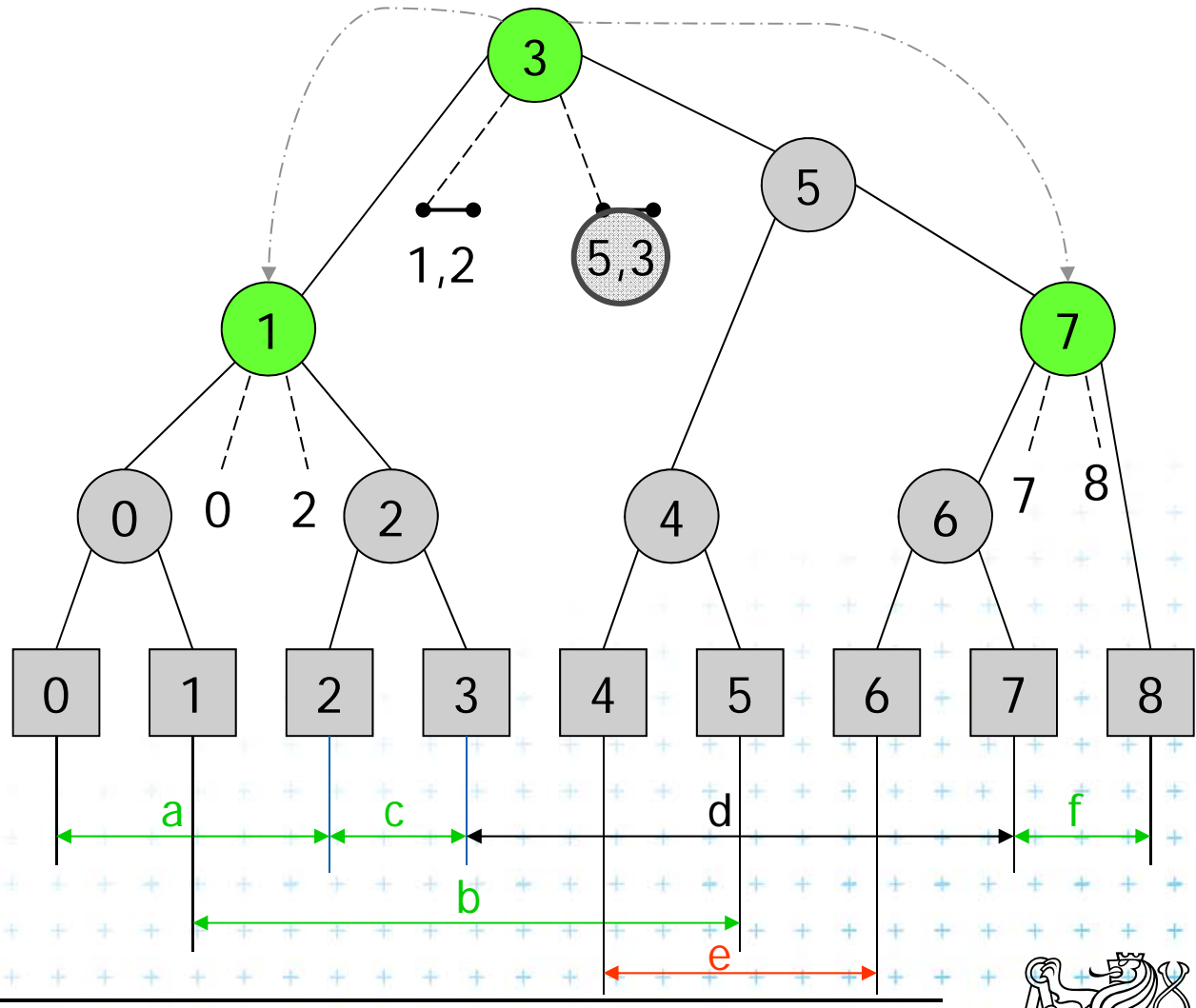


# Insert [4,6] a) Query Interval

$$H(v) \leq b < e$$

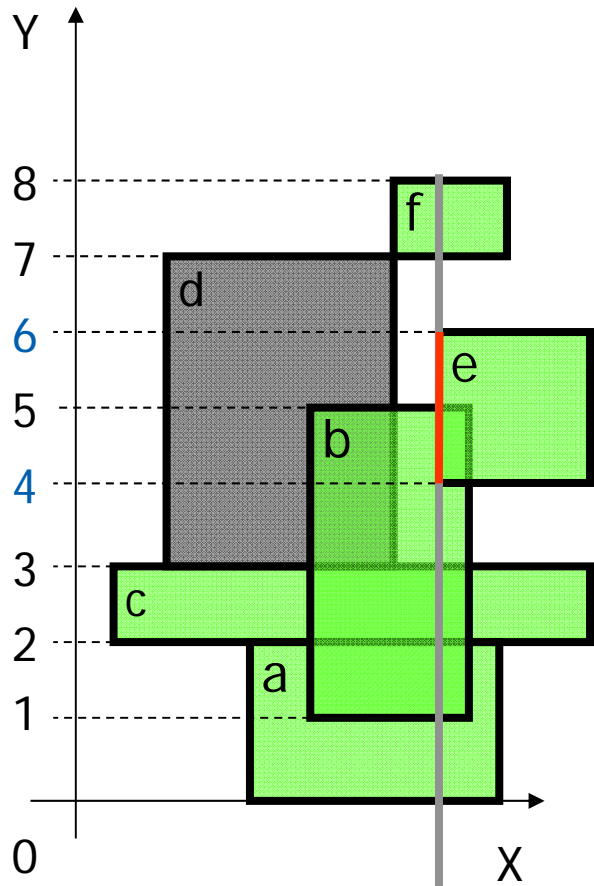





-  Active rectangle
-  Current node
-  Active node

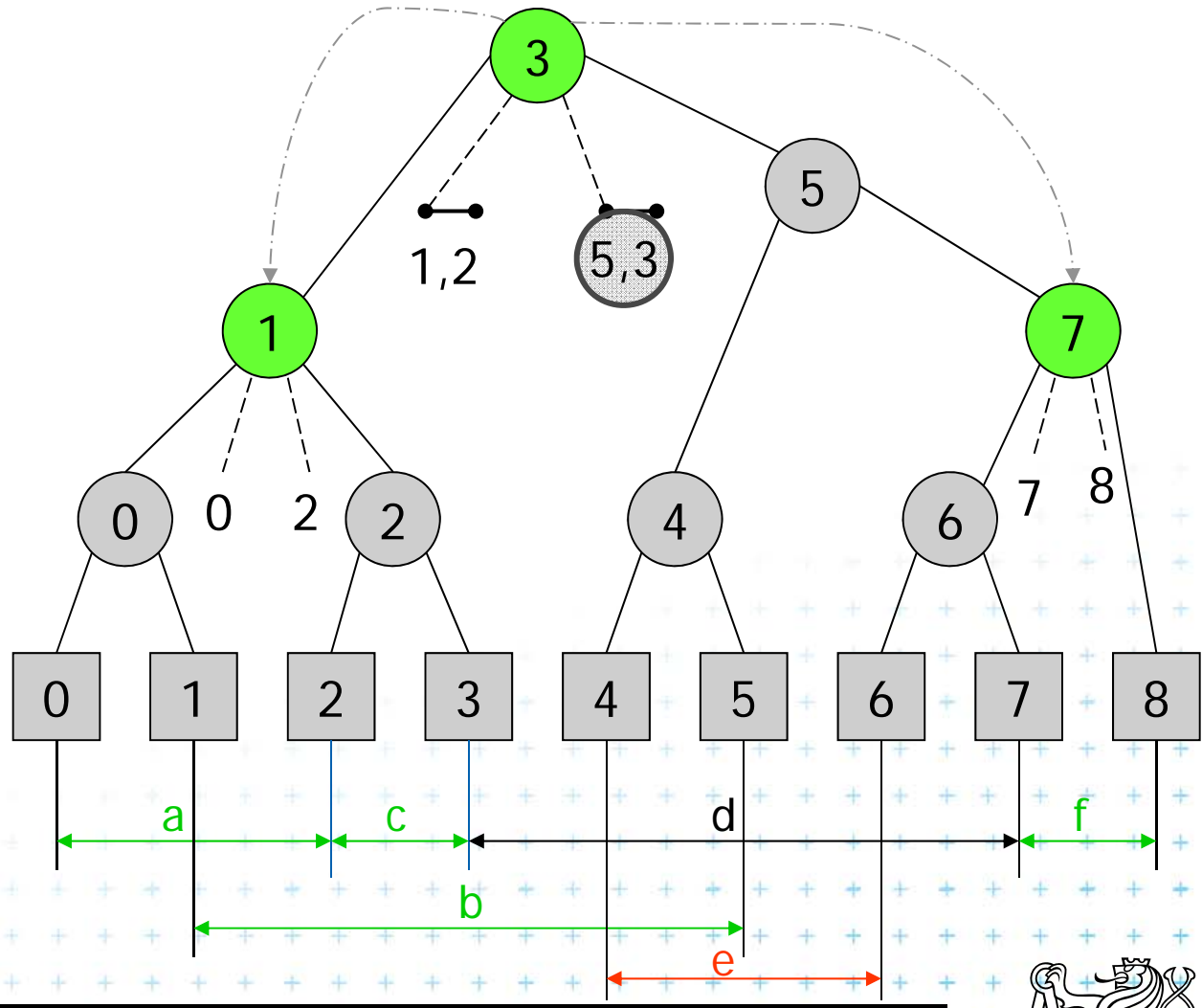


# Insert [4,6] a) Query Interval

$$H(v) \leq b < e$$

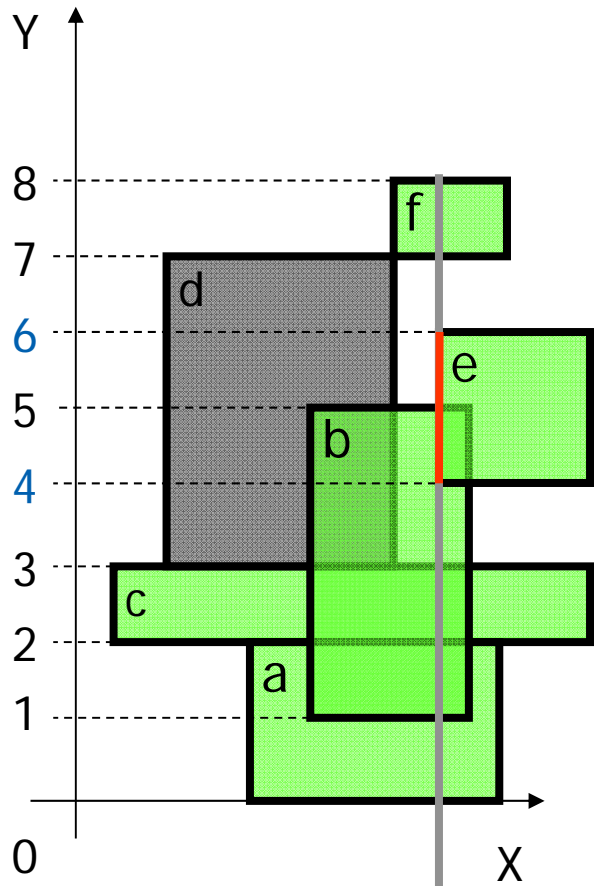





-  Active rectangle
-  Current node
-  Active node

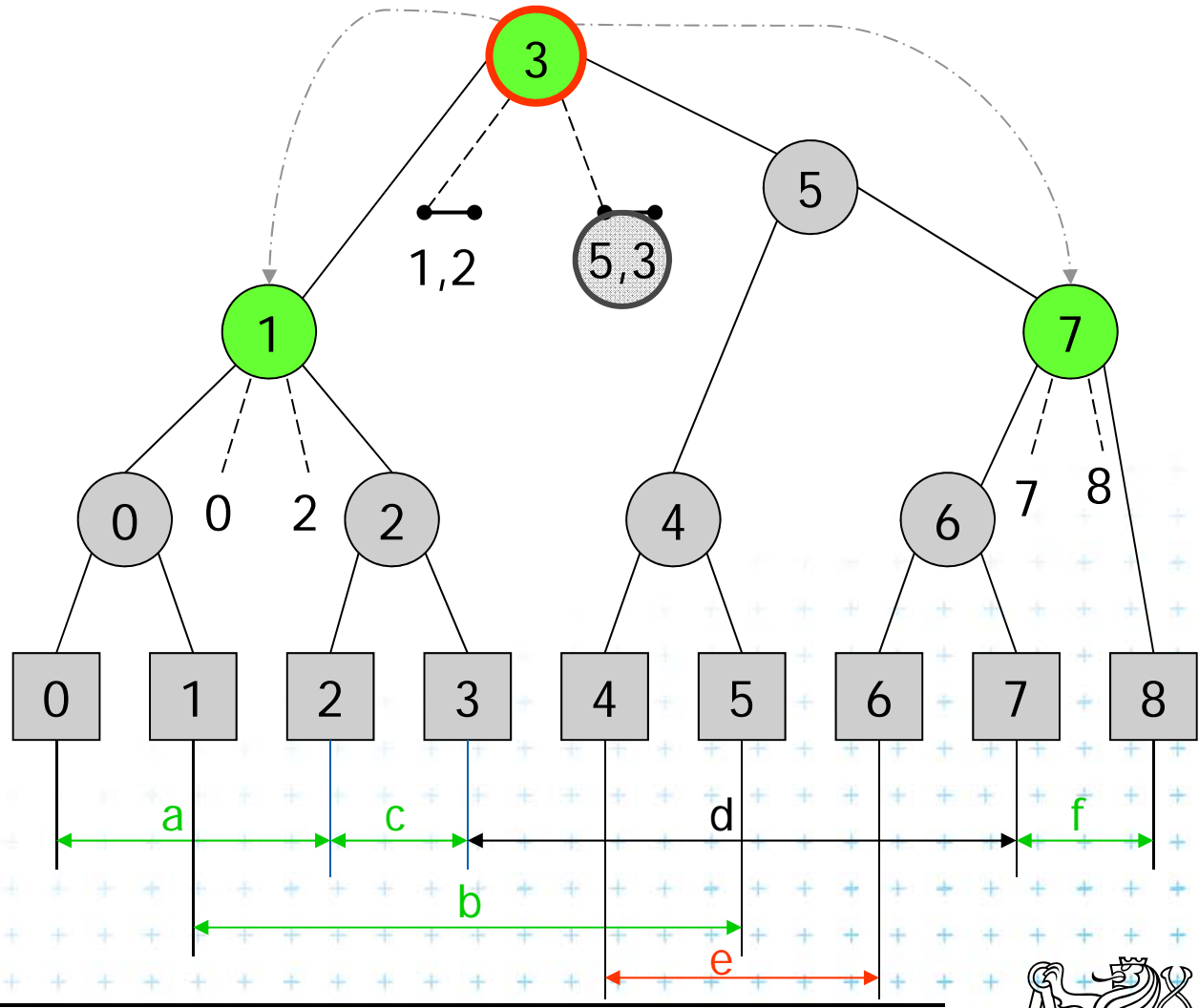


# Insert [4,6] a) Query Interval

$$H(v) \leq b < e$$



-  Active rectangle
-  Current node
-  Active node

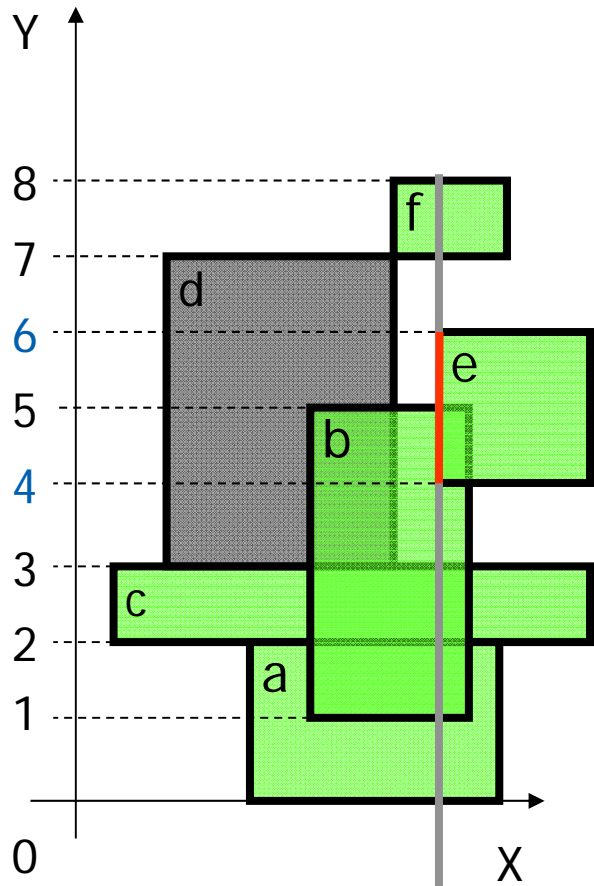







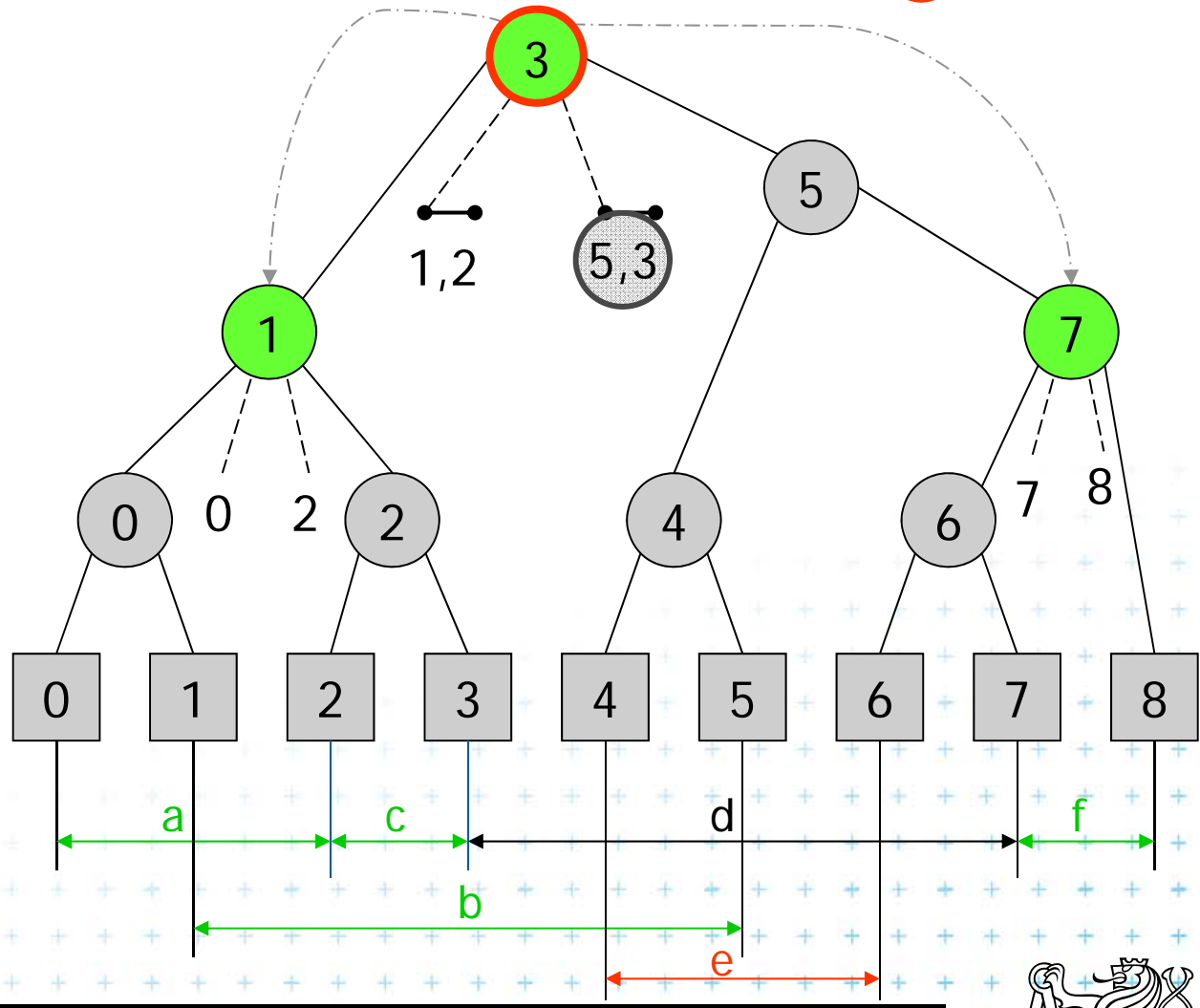
# Insert [4,6] a) Query Interval

$$H(v) \leq b < e$$

$$3 \leq 4 < 6 ?$$



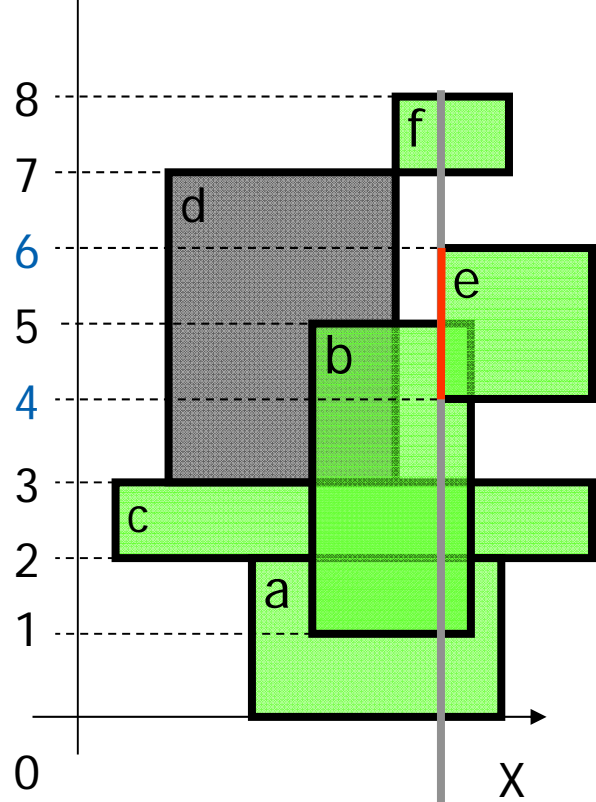
-  Active rectangle
-  Current node
-  Active node






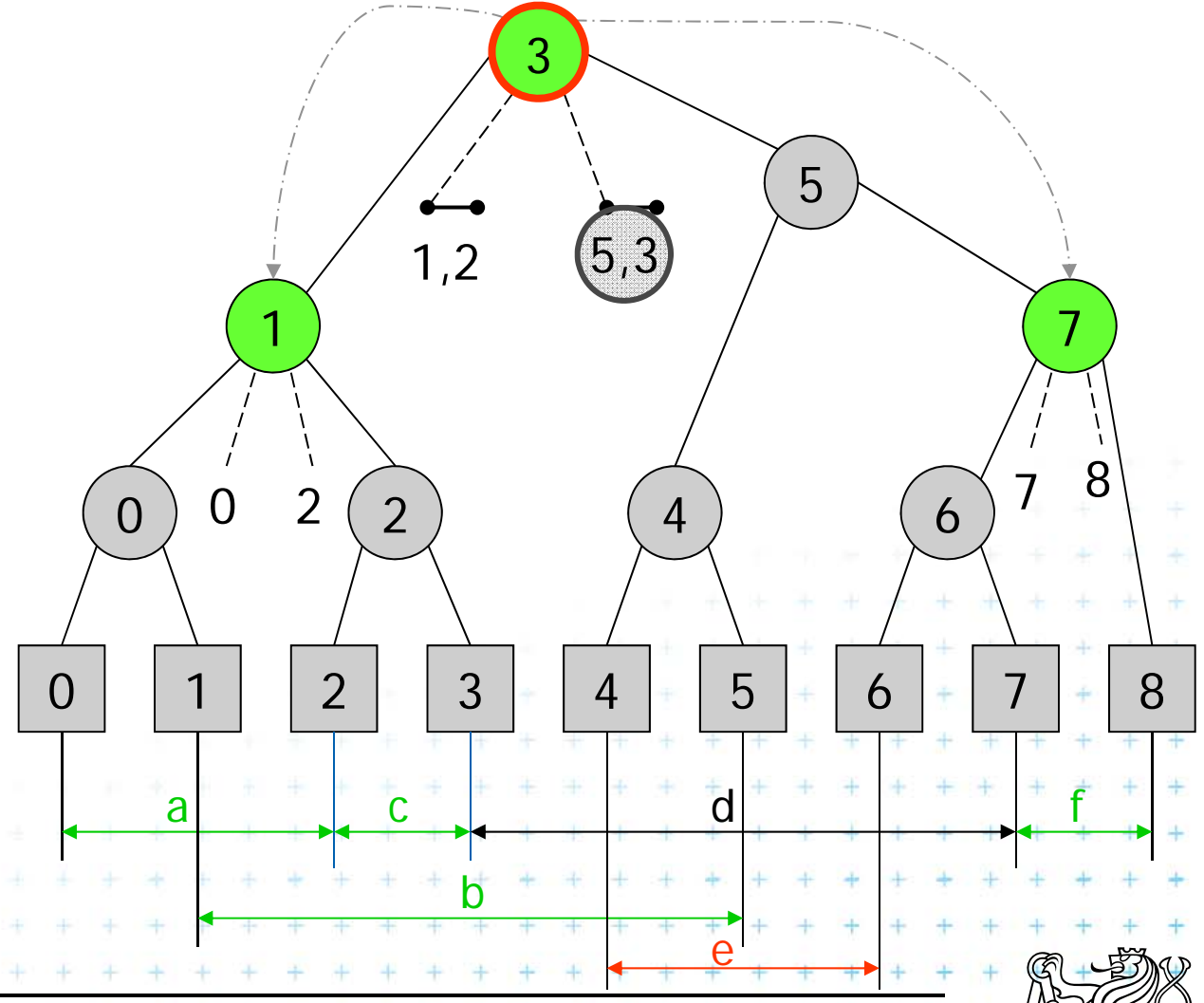
# Insert [4,6] a) Query Interval

$$H(v) \leq b < e$$

Y for (all in MR(v)) test  $MR(v).[i] \geq 4 \Rightarrow$  report intersection b  $3 \leq 4 < 6$  ?



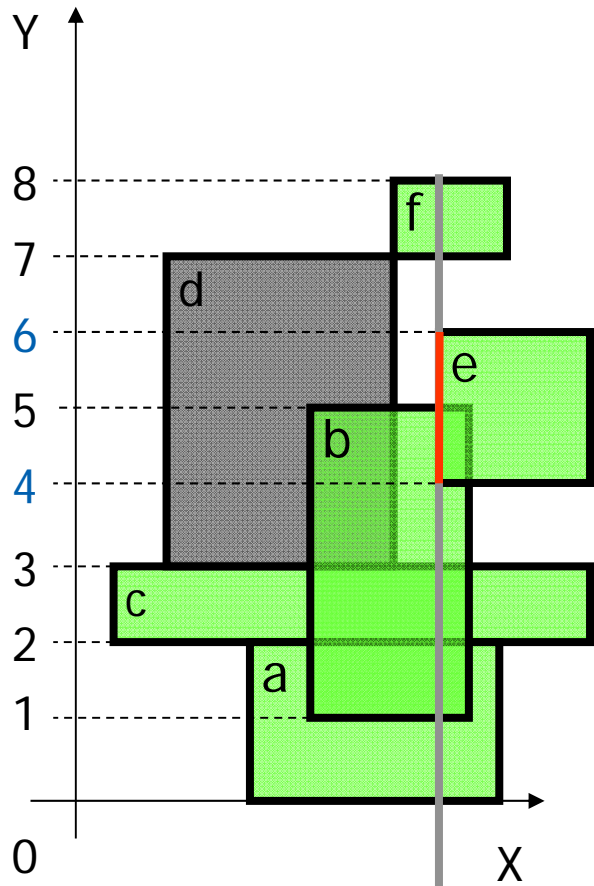
-  Active rectangle
-  Current node
-  Active node






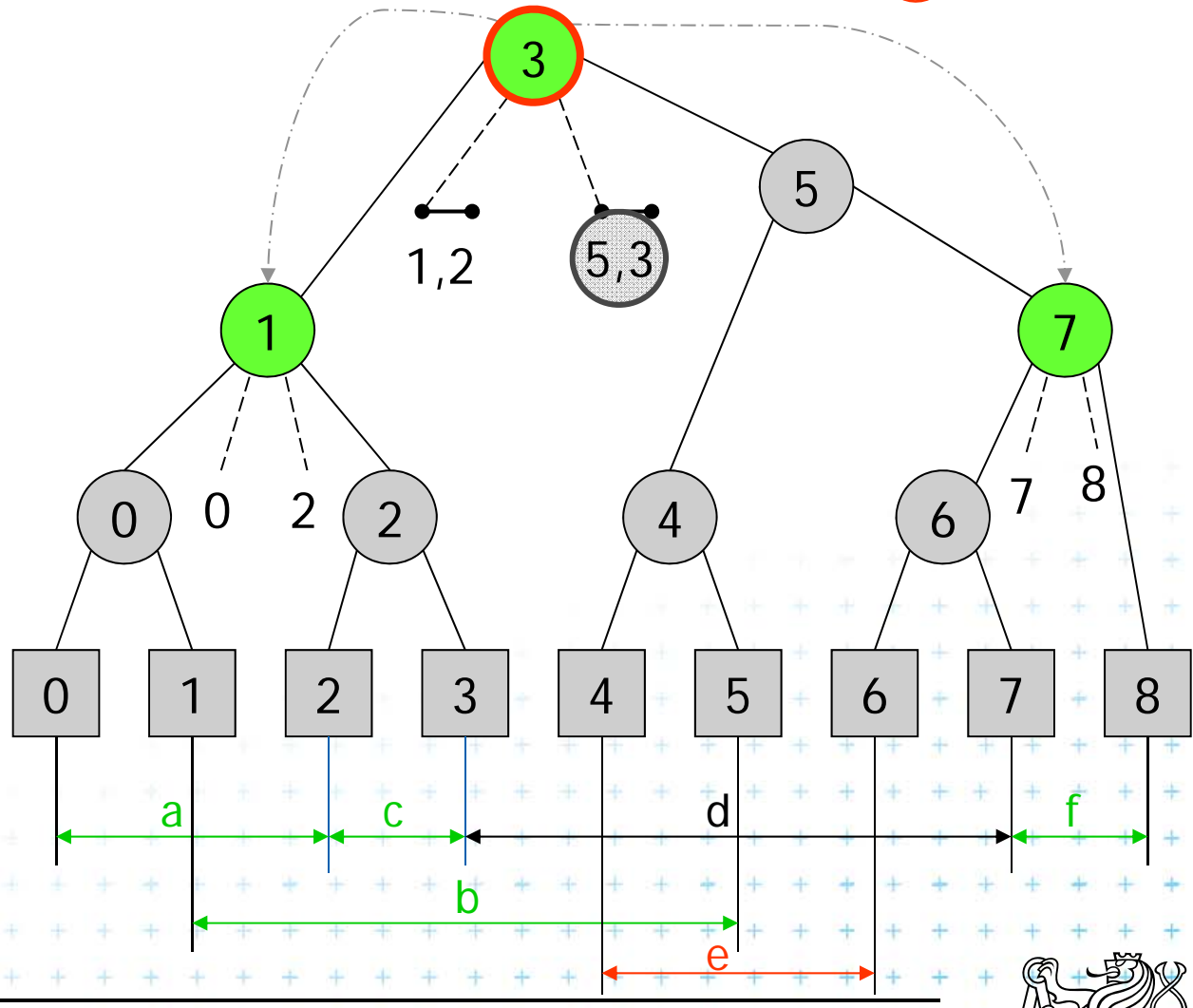
# Insert [4,6] a) Query Interval

$$H(v) \leq b < e$$

$$3 \leq 4 < 6 ?$$

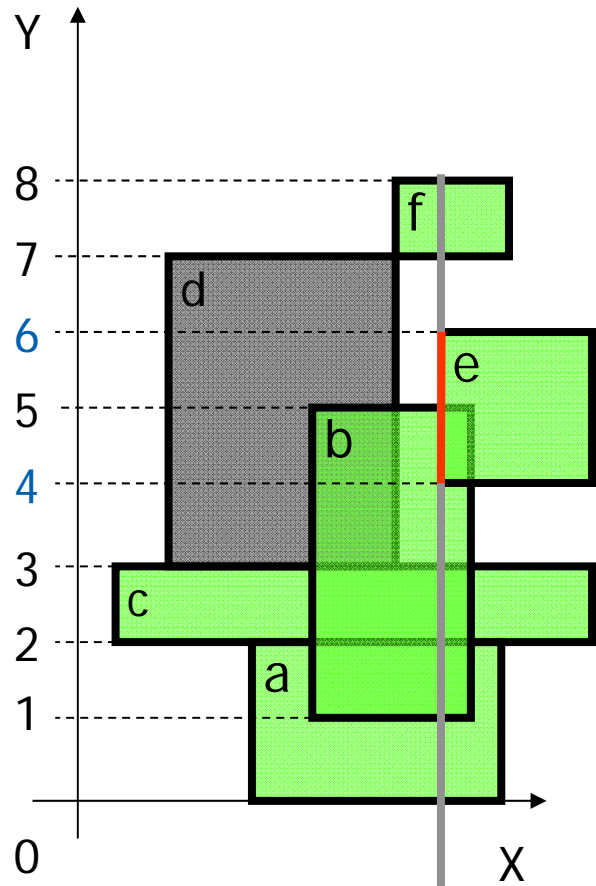


-  Active rectangle
-  Current node
-  Active node

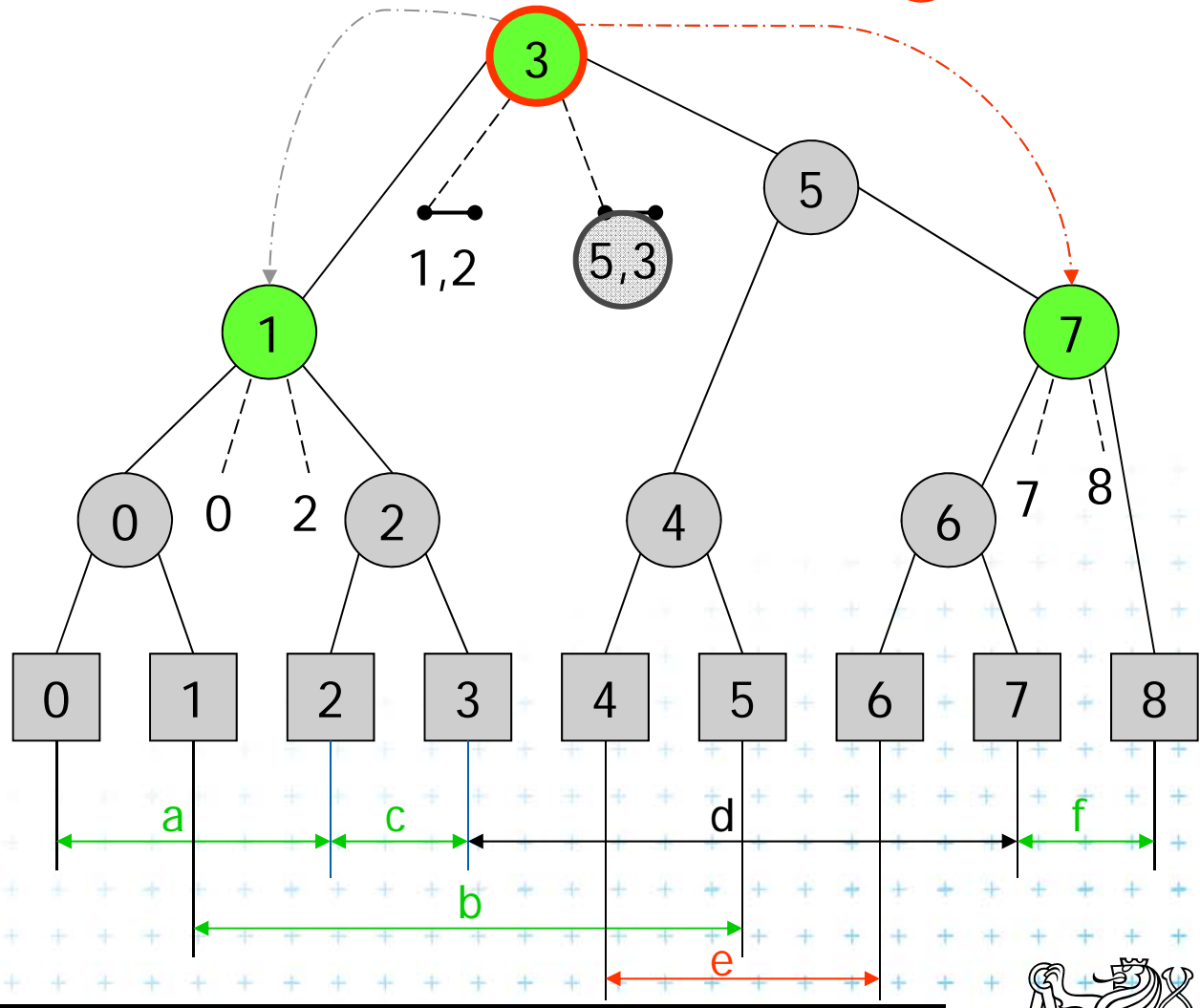


# Insert [4,6] a) Query Interval

$$H(v) \leq b < e$$



$$3 \leq 4 < 6 ?$$



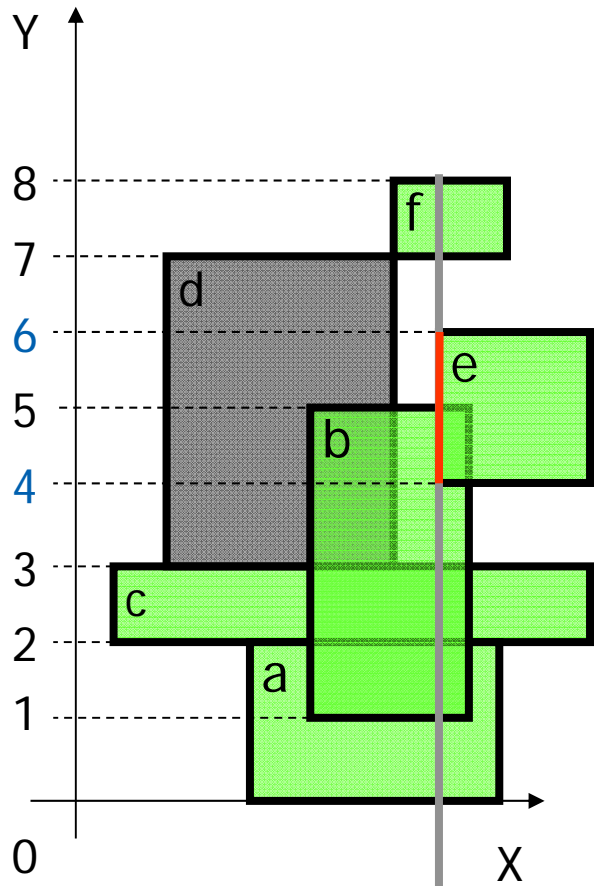
- Active rectangle
- Current node
- Active node



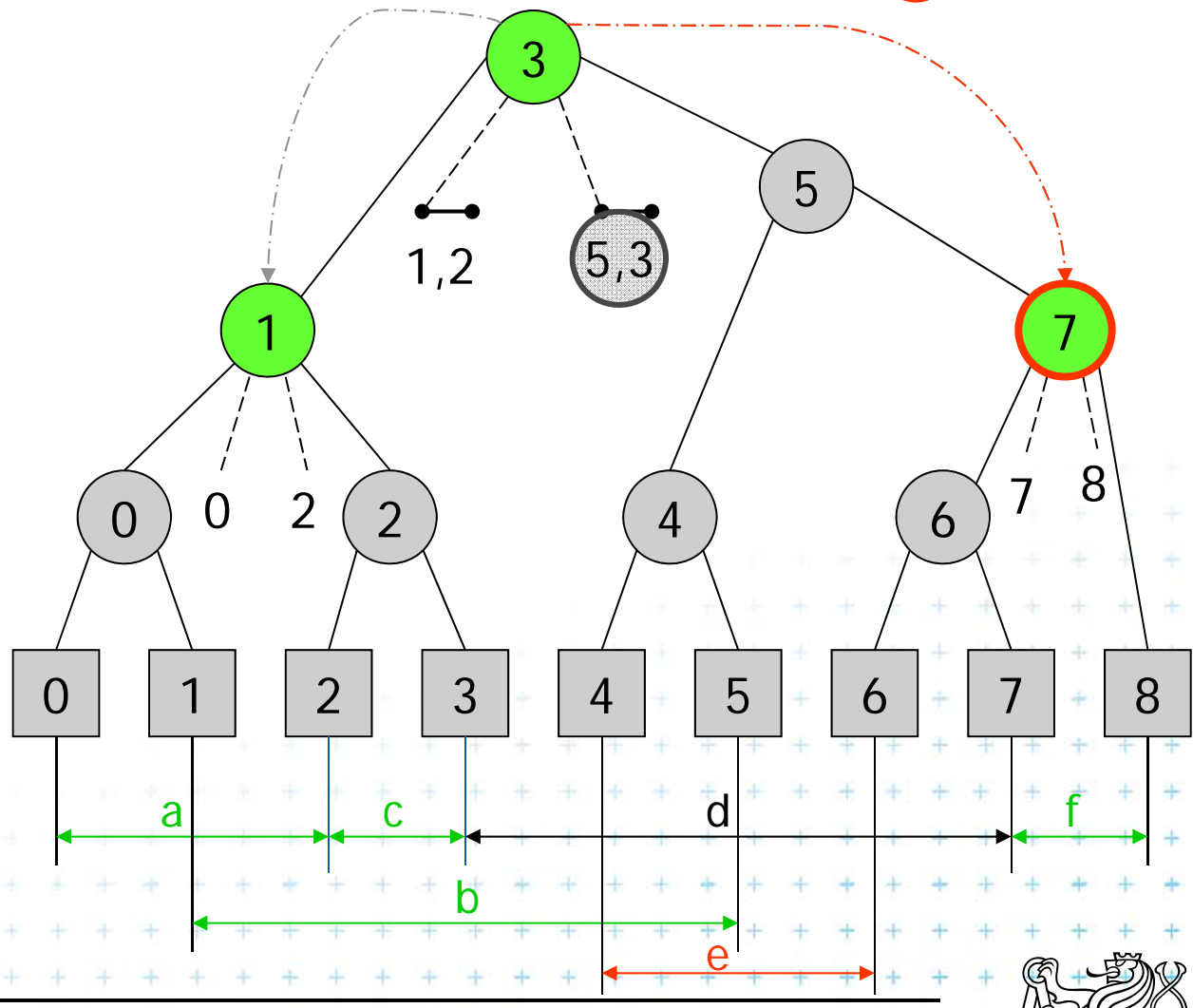
# Insert [4,6] a) Query Interval

$$H(v) \leq b < e$$

$$3 \leq 4 < 6 ?$$



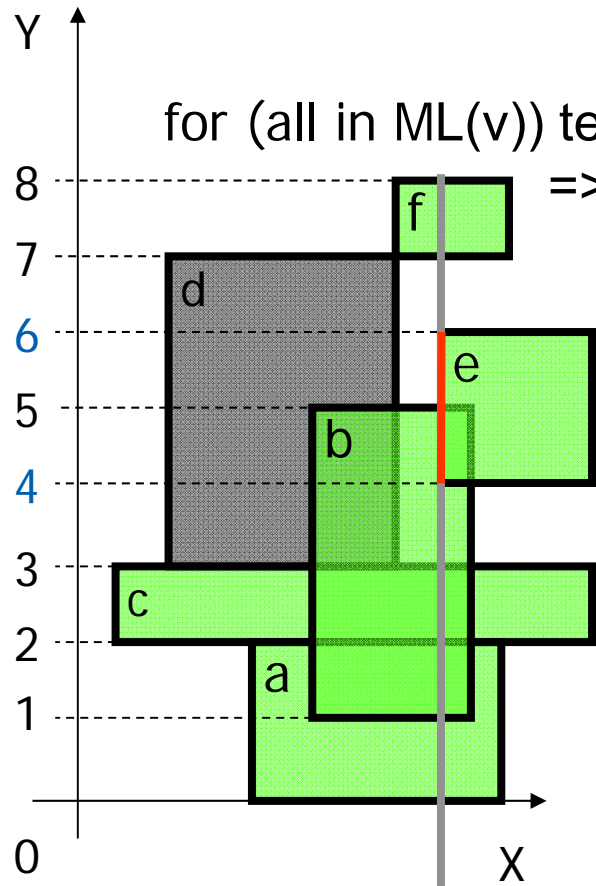
- Active rectangle
- Current node
- Active node





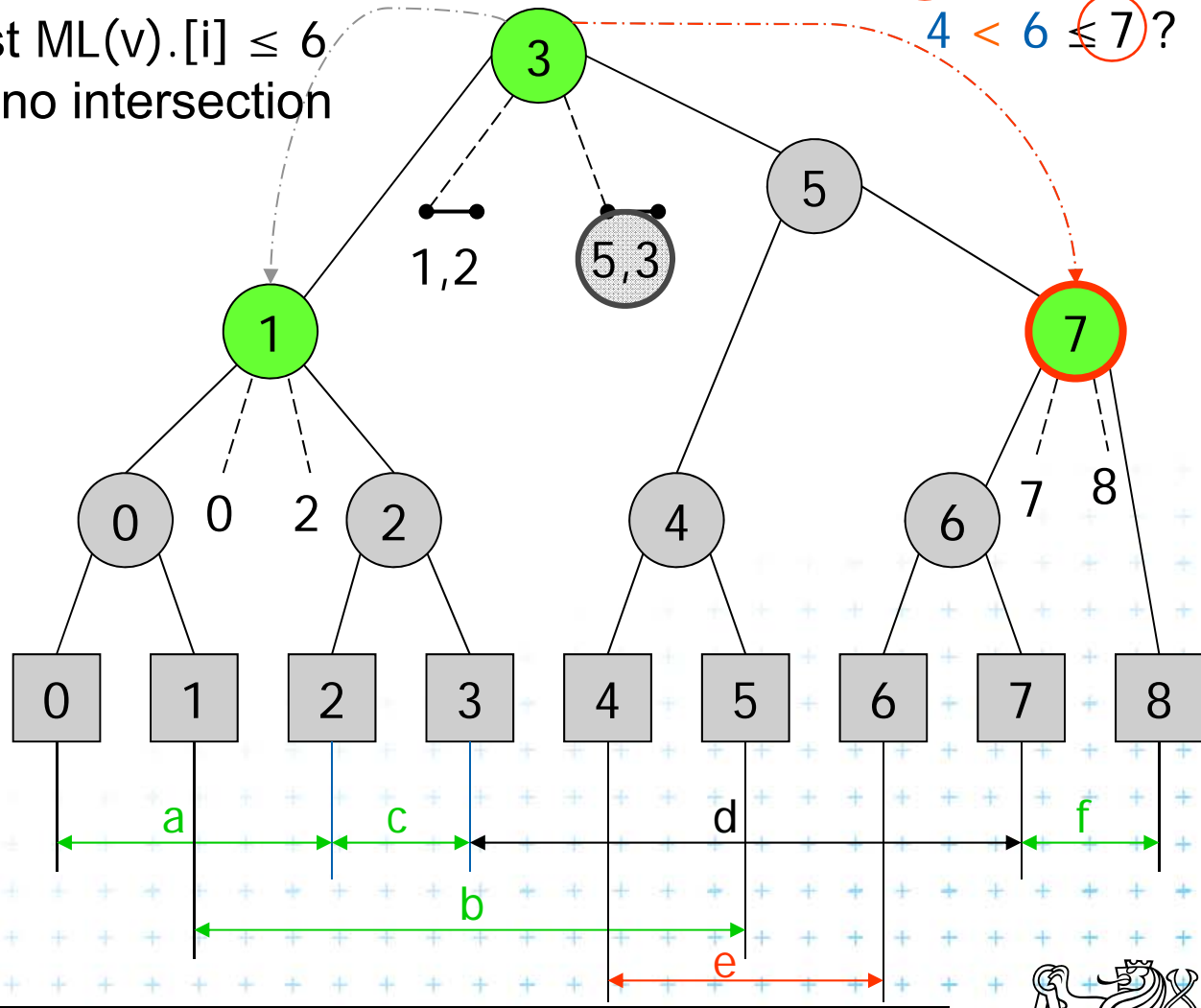
# Insert [4,6] a) Query Interval

$$H(v) \leq b < e$$



for (all in ML(v)) test  $ML(v).[i] \leq 6$   
 $\Rightarrow$  no intersection

$$\begin{aligned} & \textcircled{3} \leq 4 < 6 ? \\ & 4 < 6 \leq \textcircled{7} ? \end{aligned}$$

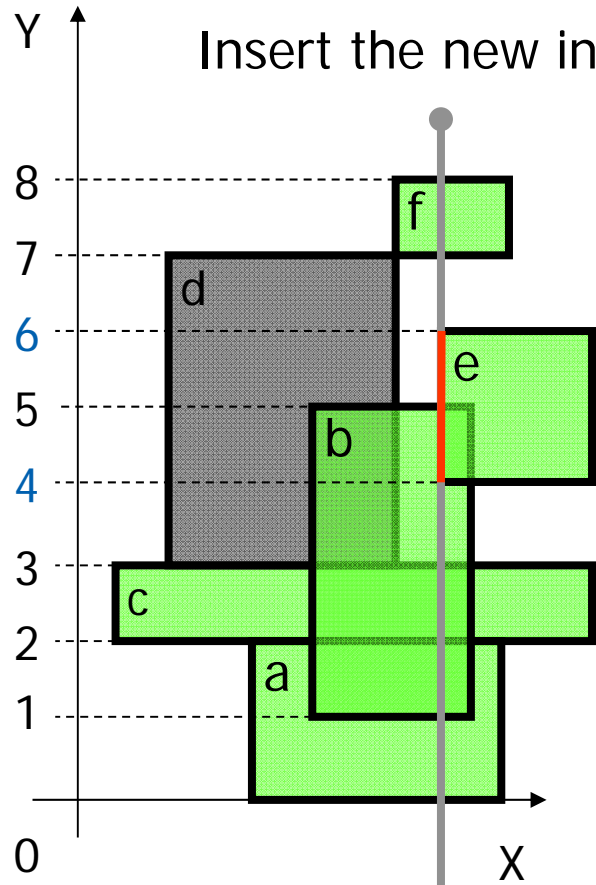


- Active rectangle
- Current node
- Active node

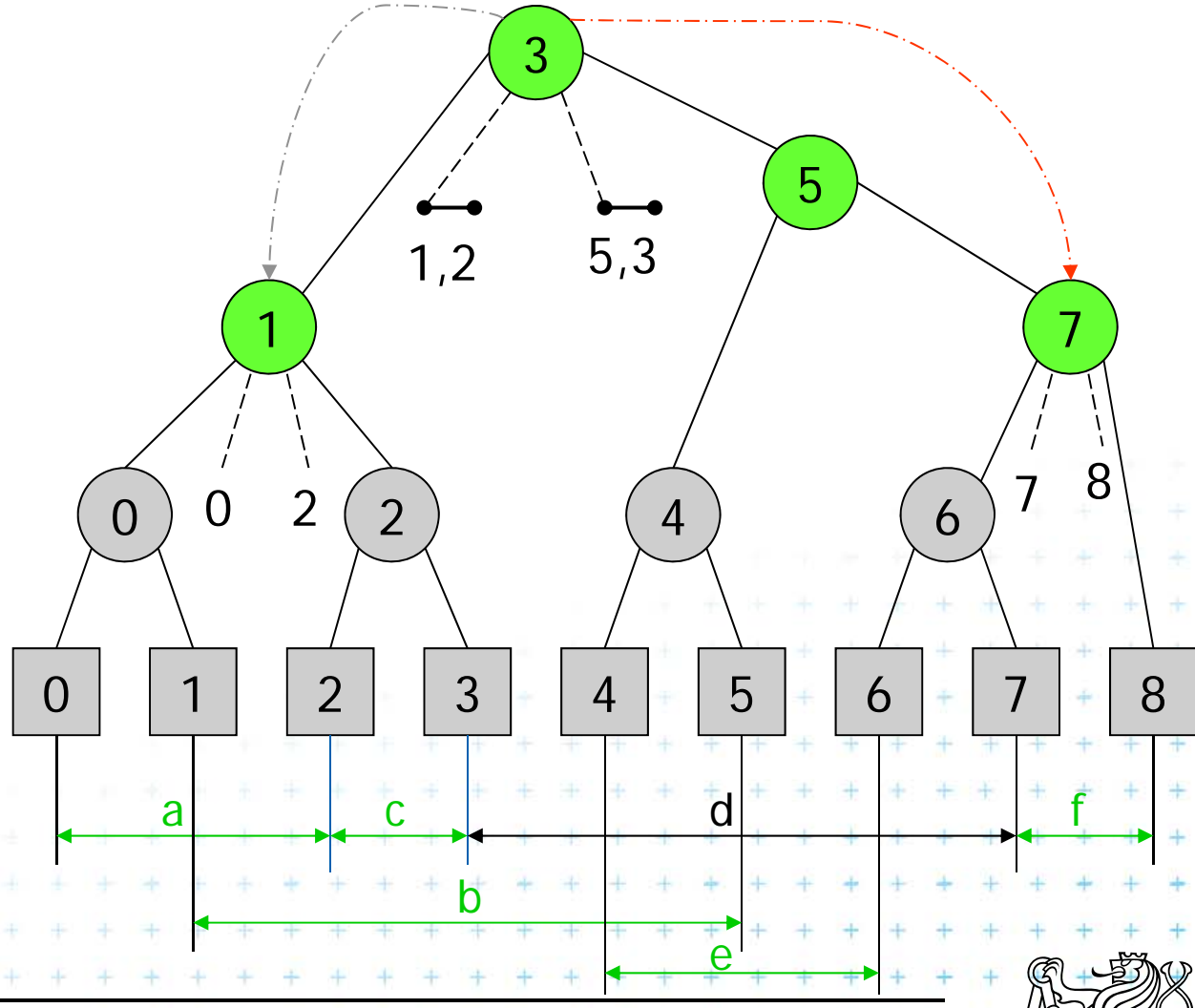





# Insert [4,6] b) Insert Interval

$$H(v) \leq b < e$$



Insert the new interval to secondary lists



-  Active rectangle
-  Current node
-  Active node

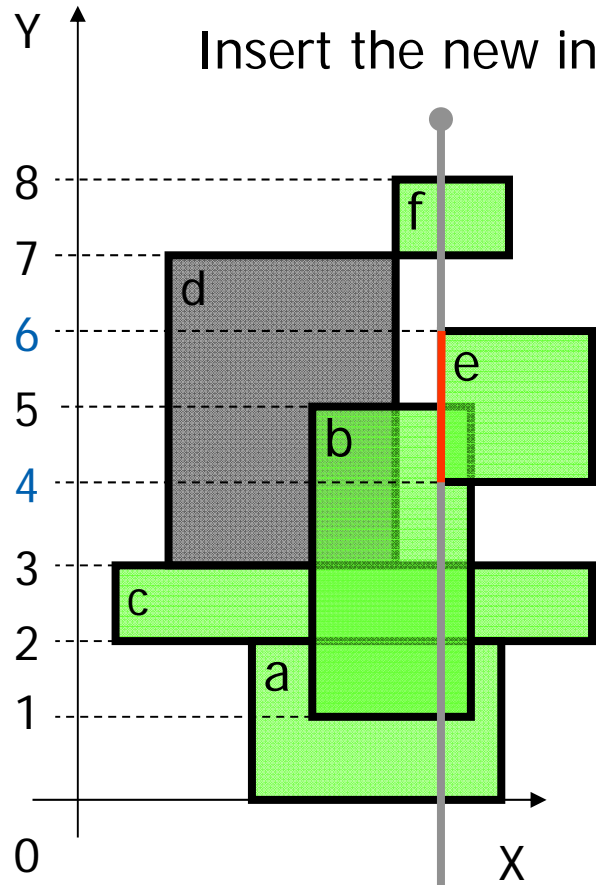






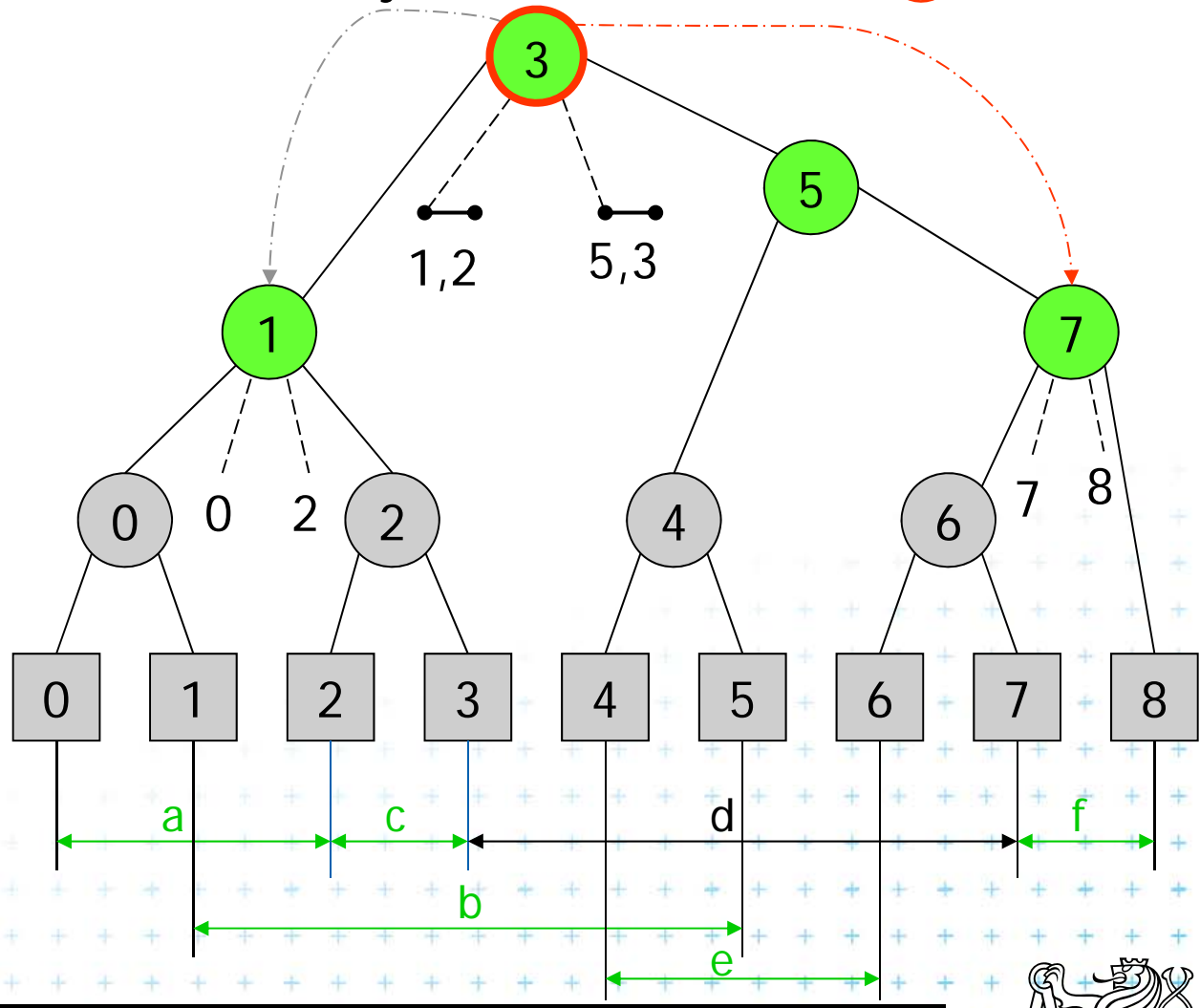
# Insert [4,6] b) Insert Interval

$$H(v) \leq b < e$$



Insert the new interval to secondary lists

$$? 3 \leq 4 < 6 ?$$



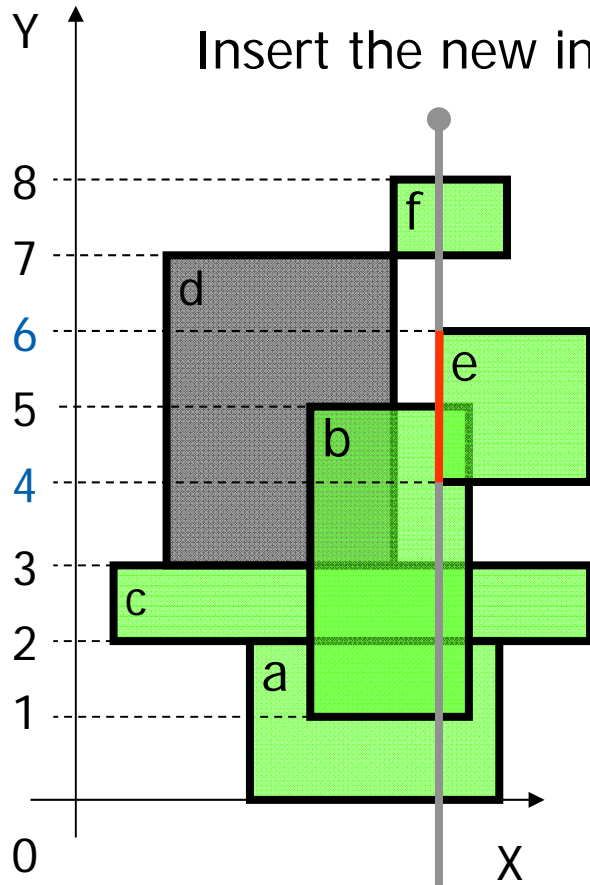
- Active rectangle
- Current node
- Active node



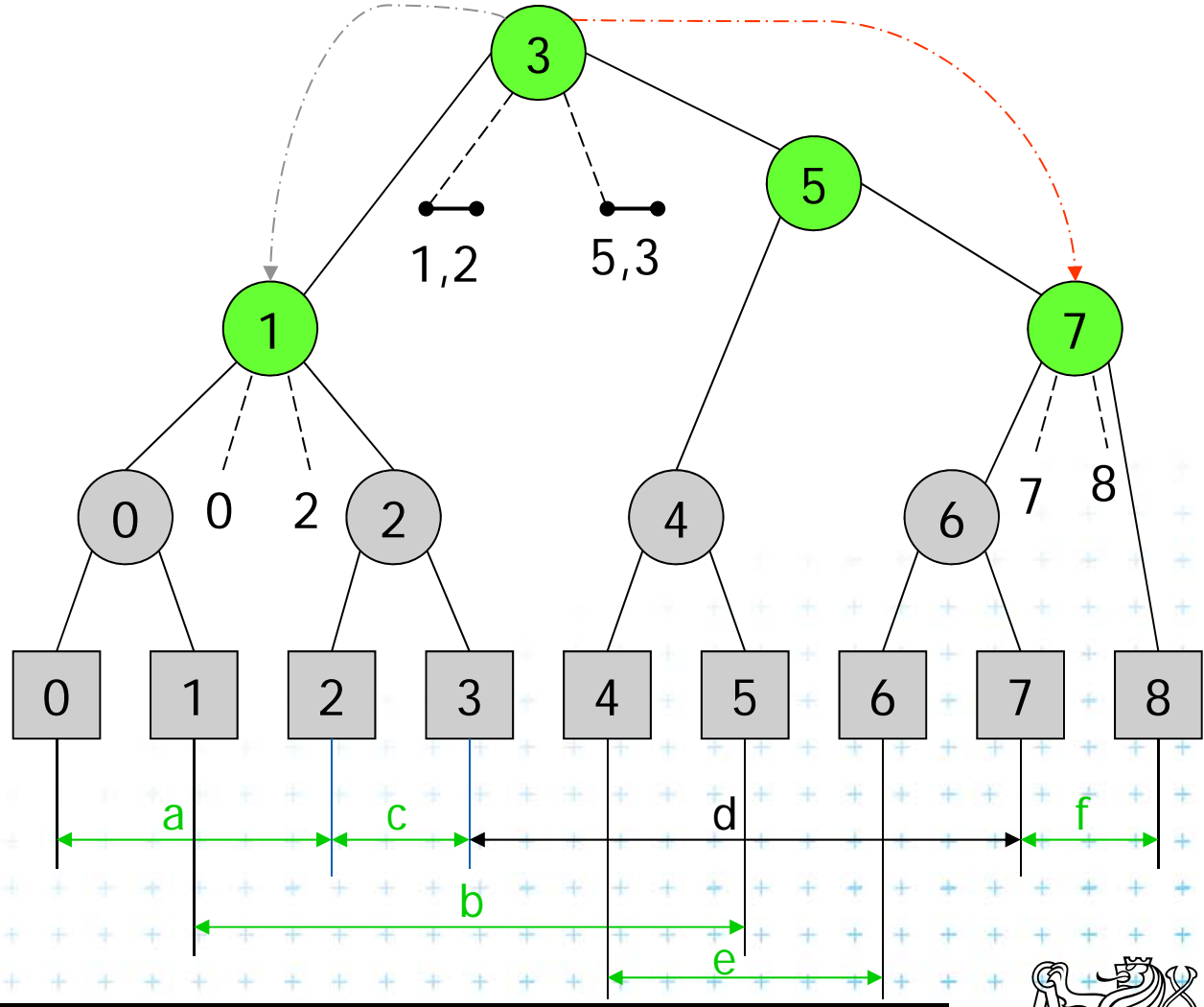
# Insert [4,6] b) Insert Interval




$$H(v) \leq b < e$$

$$? 3 \leq 4 < 6 ?$$



Insert the new interval to secondary lists

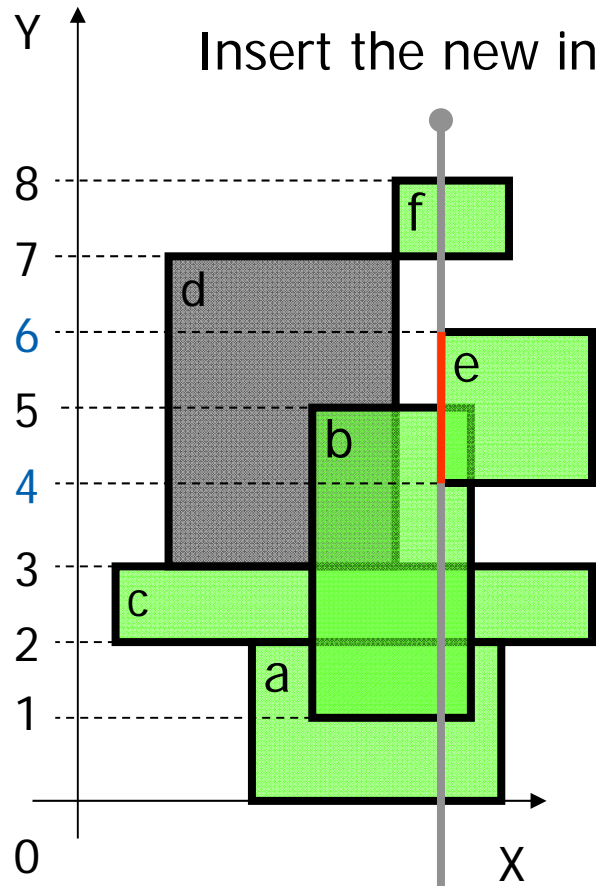


-  Active rectangle
-  Current node
-  Active node



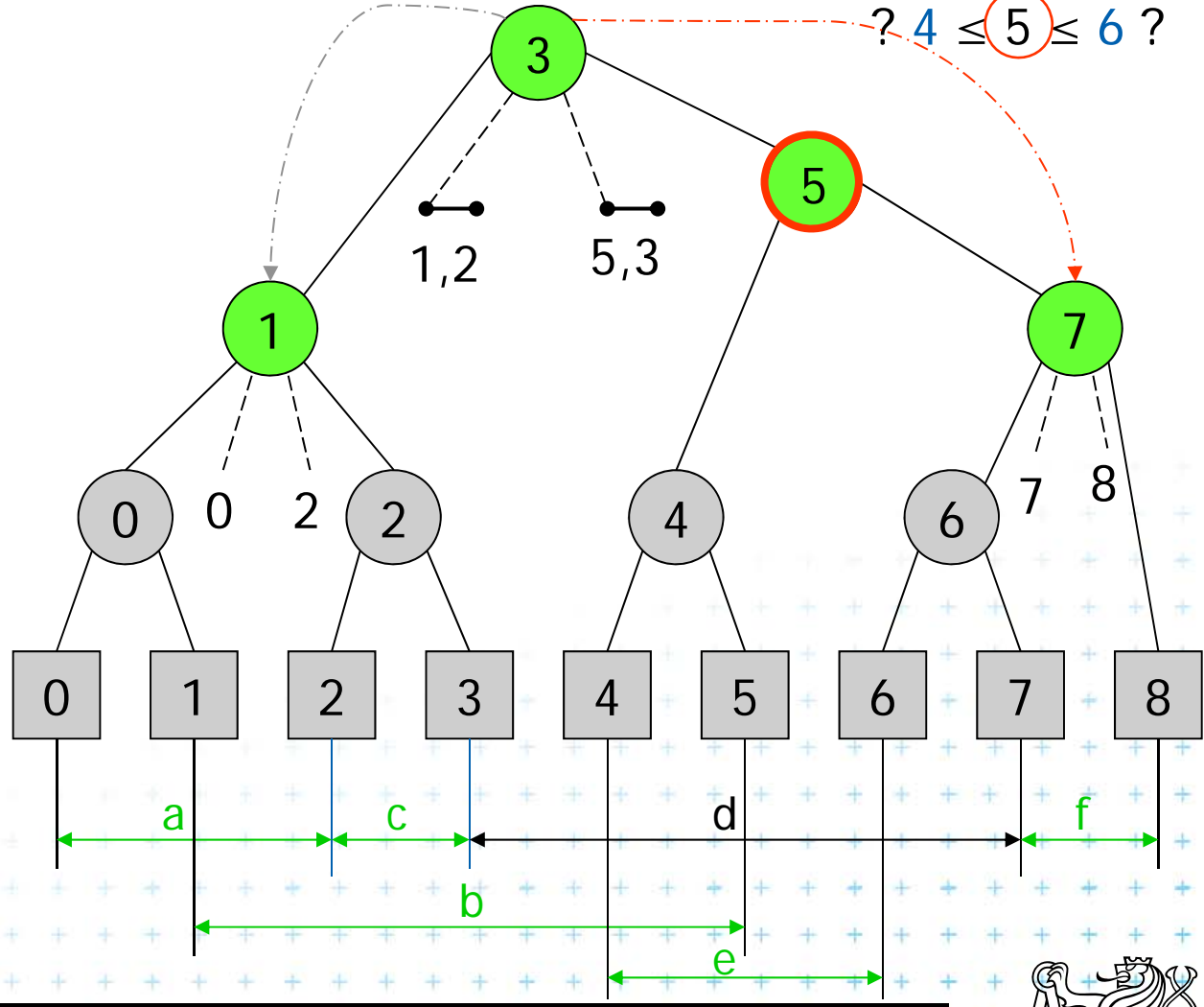
# Insert [4,6] b) Insert Interval

$$H(v) \leq b < e$$



Insert the new interval to secondary lists

$$\begin{aligned} &? 3 \leq 4 < 6 ? \\ &? 4 \leq 5 \leq 6 ? \end{aligned}$$

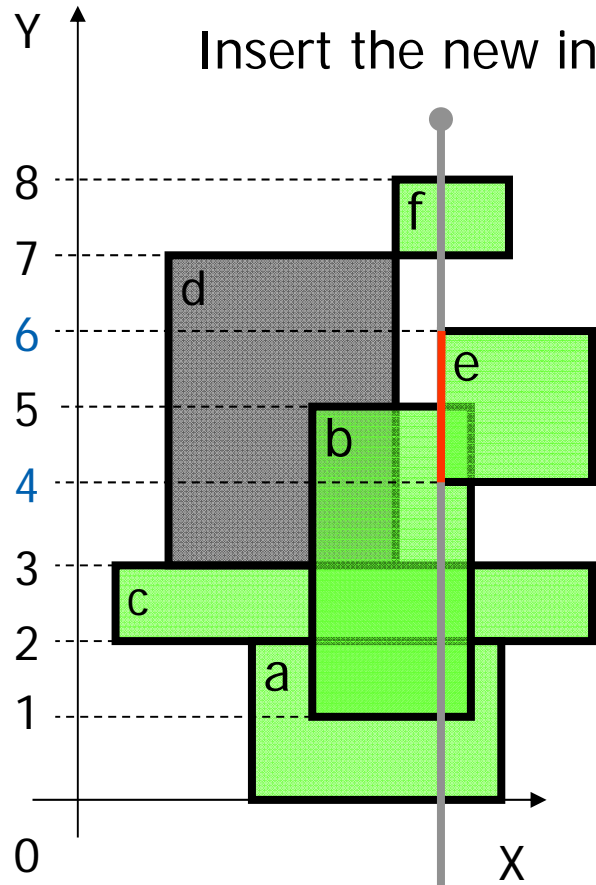


- Active rectangle
- Current node
- Active node



# Insert [4,6] b) Insert Interval

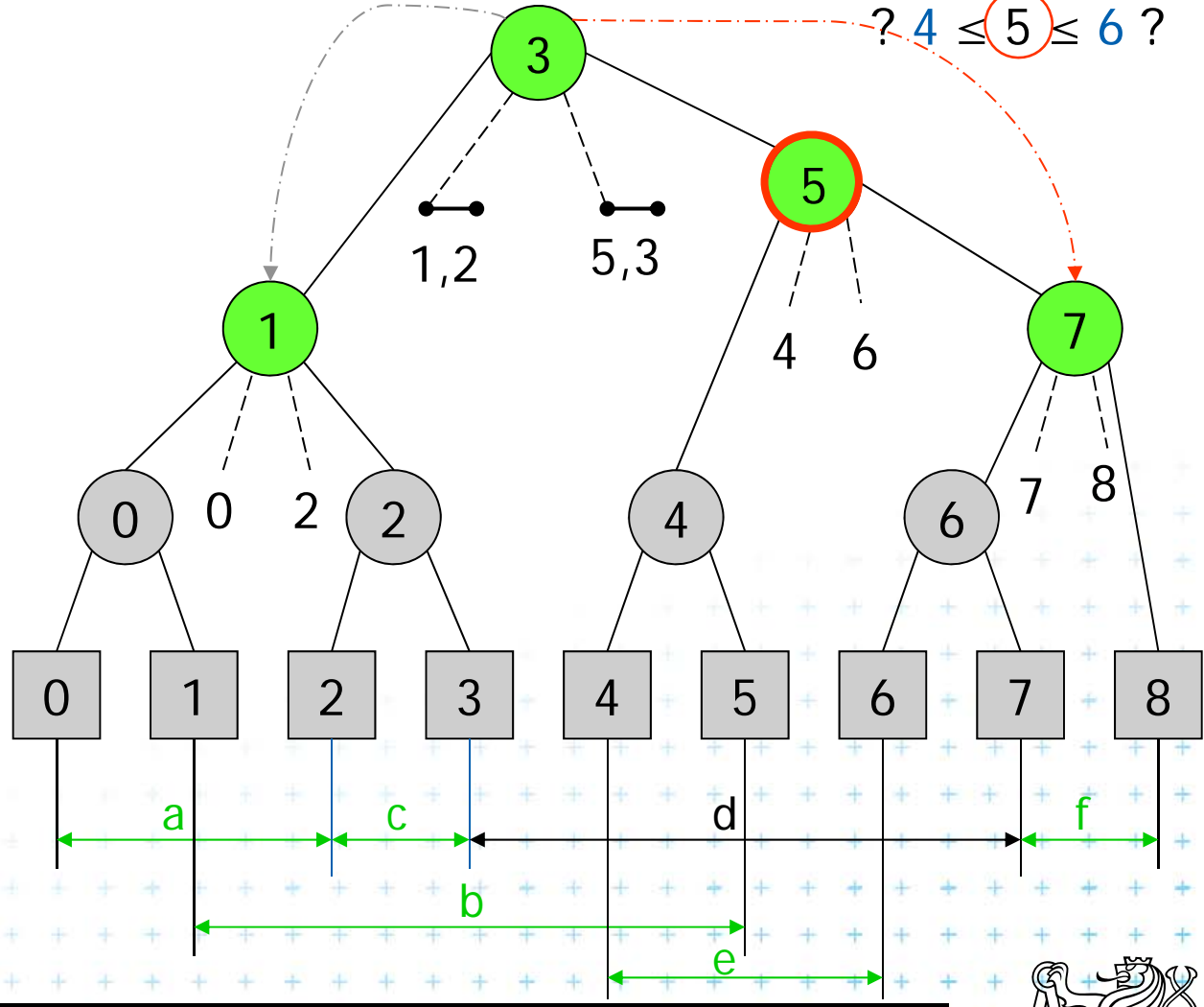
$$H(v) \leq b < e$$



Insert the new interval to secondary lists

$$? 3 \leq 4 < 6 ?$$

$$? 4 \leq 5 \leq 6 ?$$

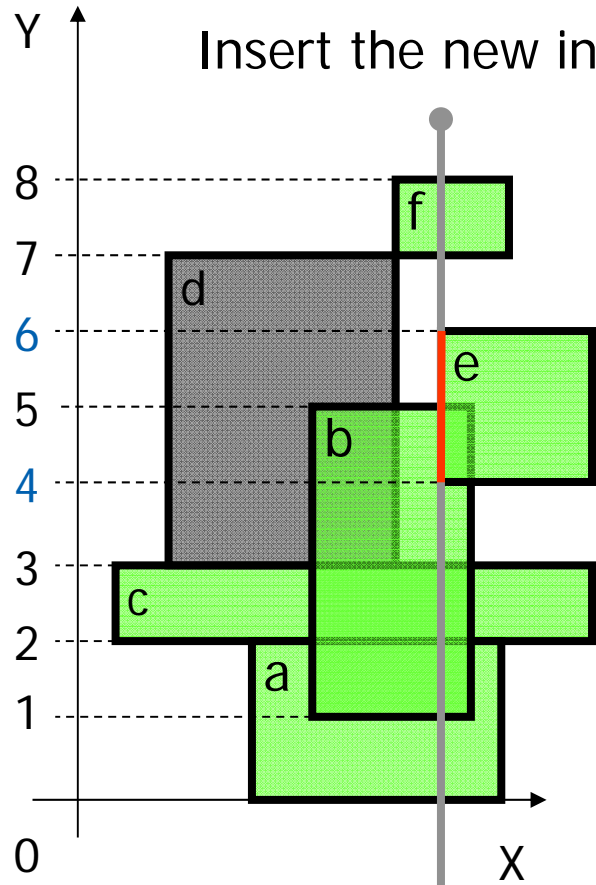


- Active rectangle
- Current node
- Active node



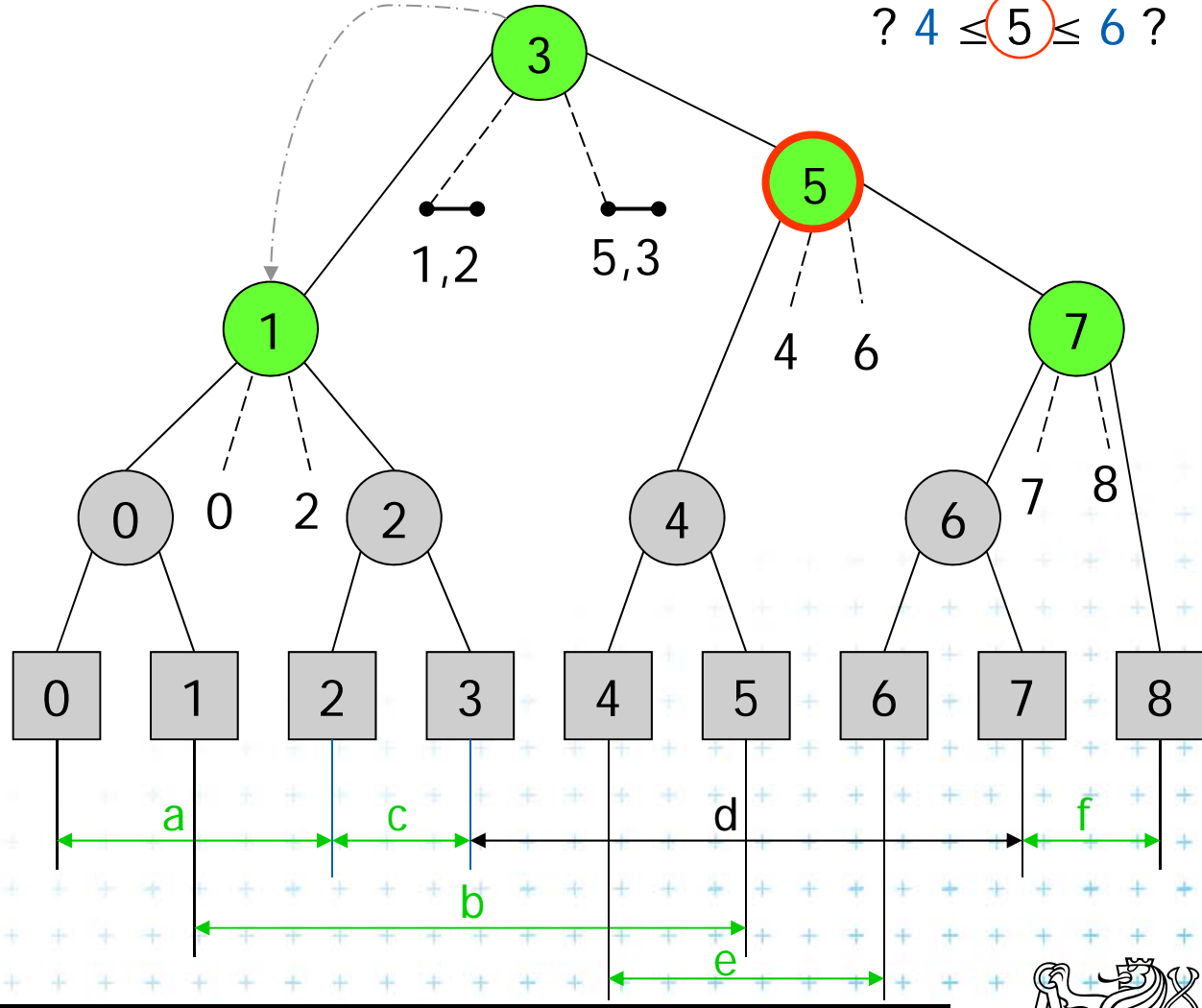
# Insert [4,6] b) Insert Interval

$$H(v) \leq b < e$$



Insert the new interval to secondary lists

$$\begin{aligned} &? 3 \leq 4 < 6 ? \\ &? 4 \leq 5 \leq 6 ? \end{aligned}$$

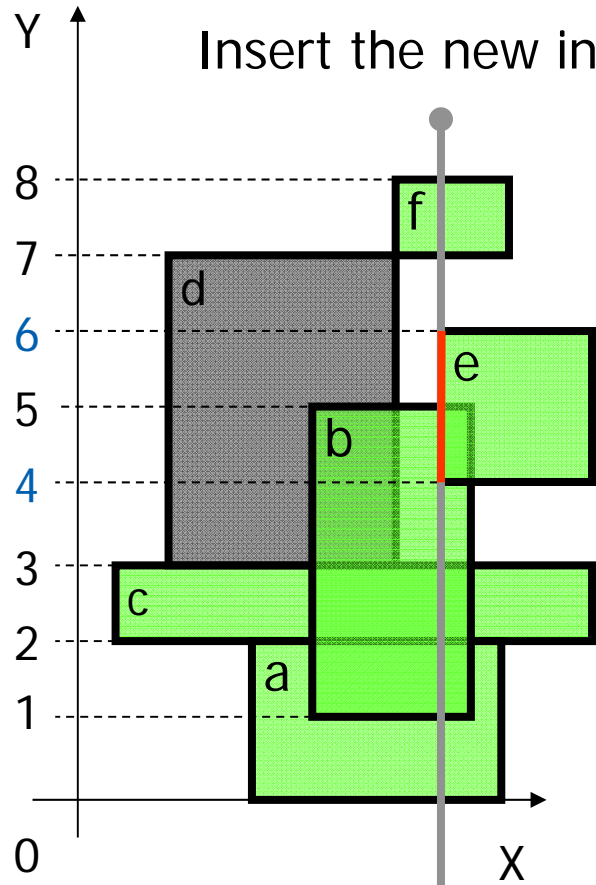


- Active rectangle
- Current node
- Active node



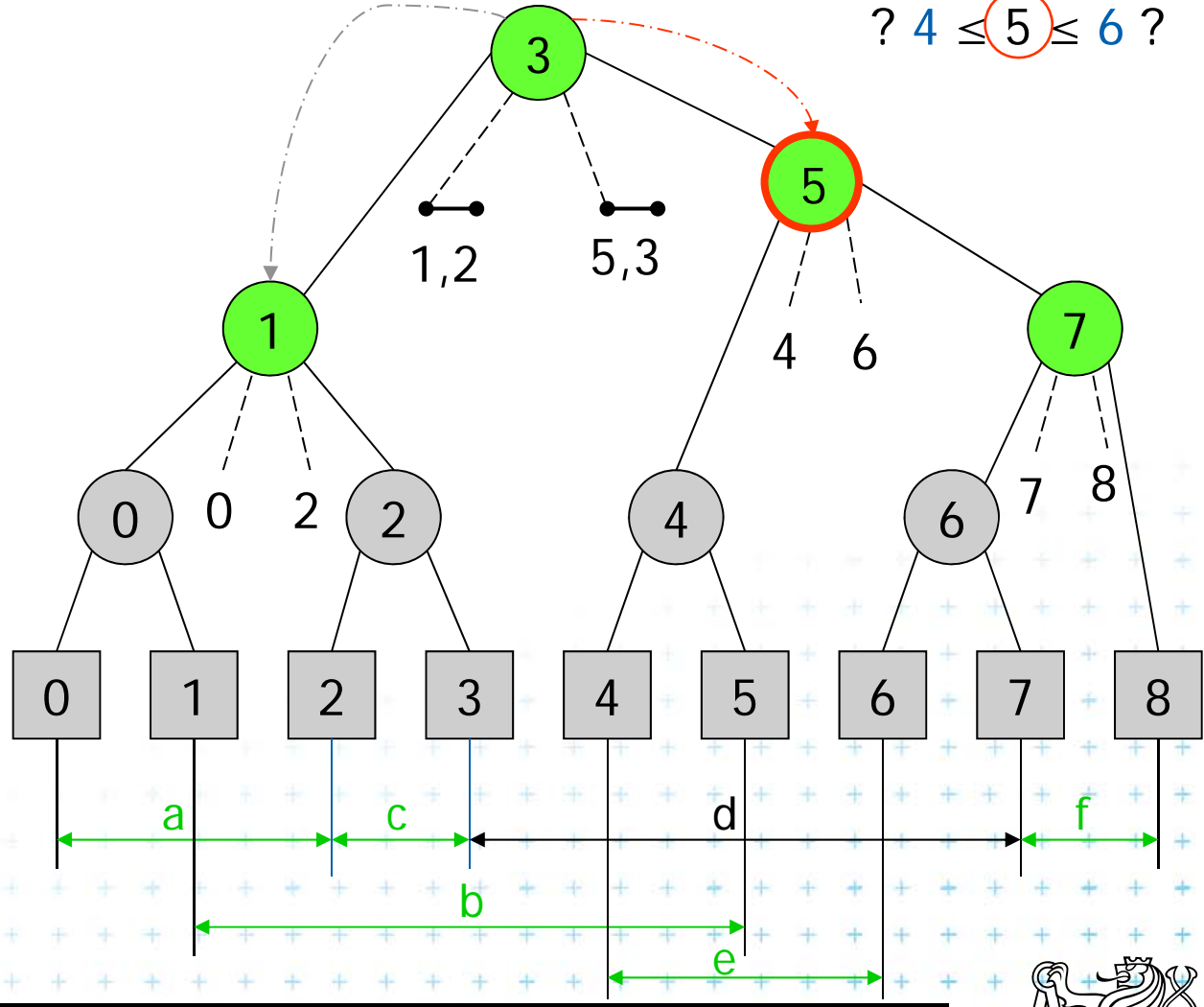
# Insert [4,6] b) Insert Interval

$$H(v) \leq b < e$$



Insert the new interval to secondary lists

$$\begin{aligned} &? 3 \leq 4 < 6 ? \\ &? 4 \leq 5 \leq 6 ? \end{aligned}$$

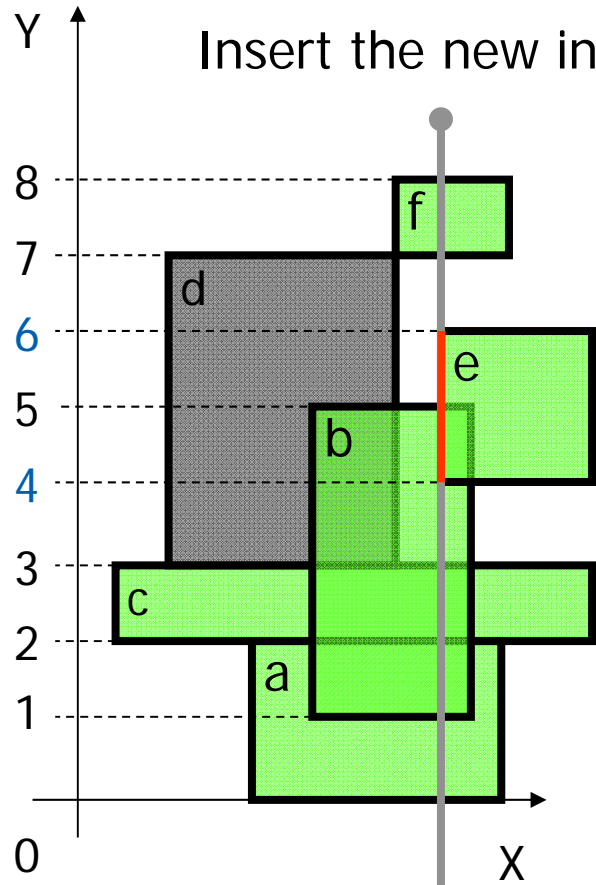


- Active rectangle
- Current node
- Active node



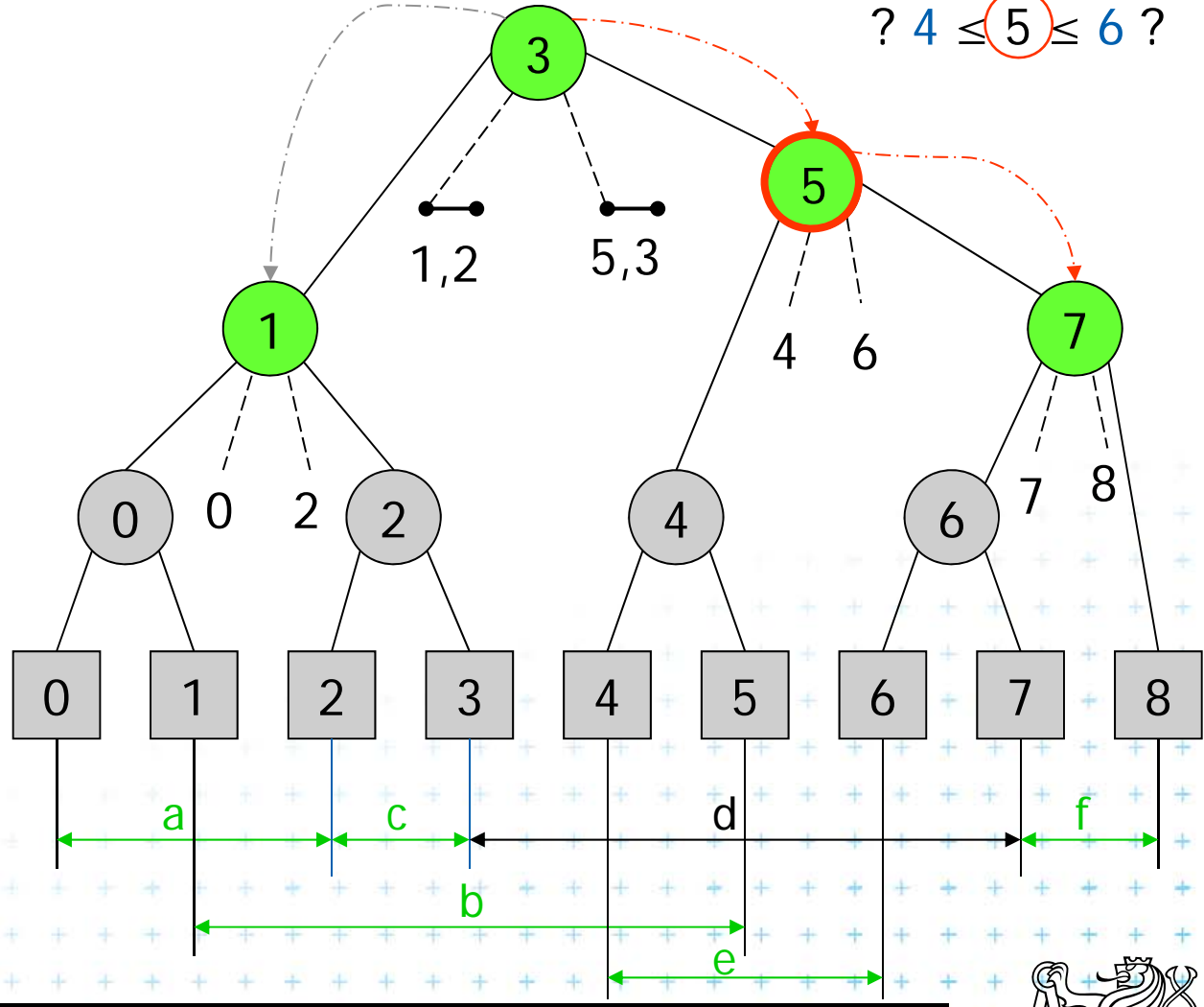
# Insert [4,6] b) Insert Interval

$$H(v) \leq b < e$$



Insert the new interval to secondary lists

$$\begin{aligned} &? 3 \leq 4 < 6 ? \\ &? 4 \leq 5 \leq 6 ? \end{aligned}$$



- Active rectangle
- Current node
- Active node

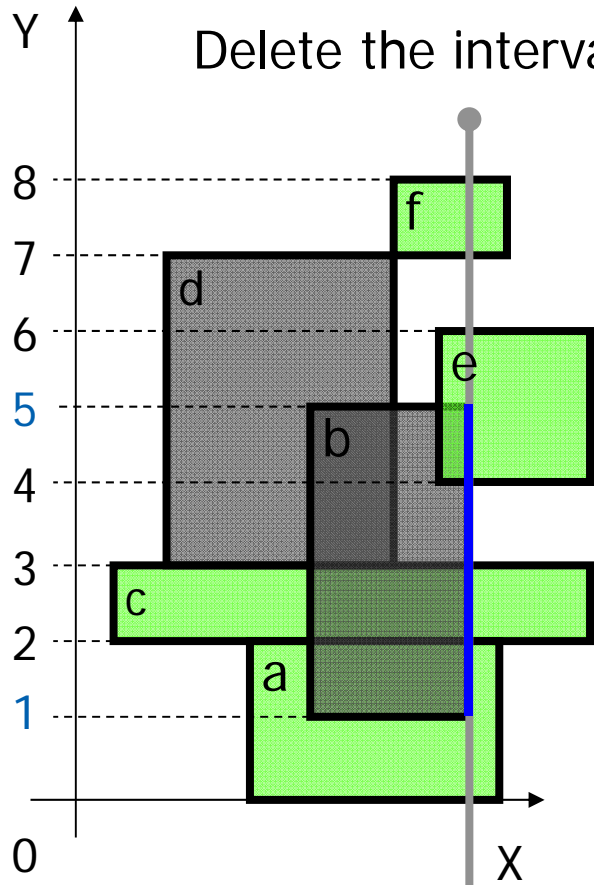




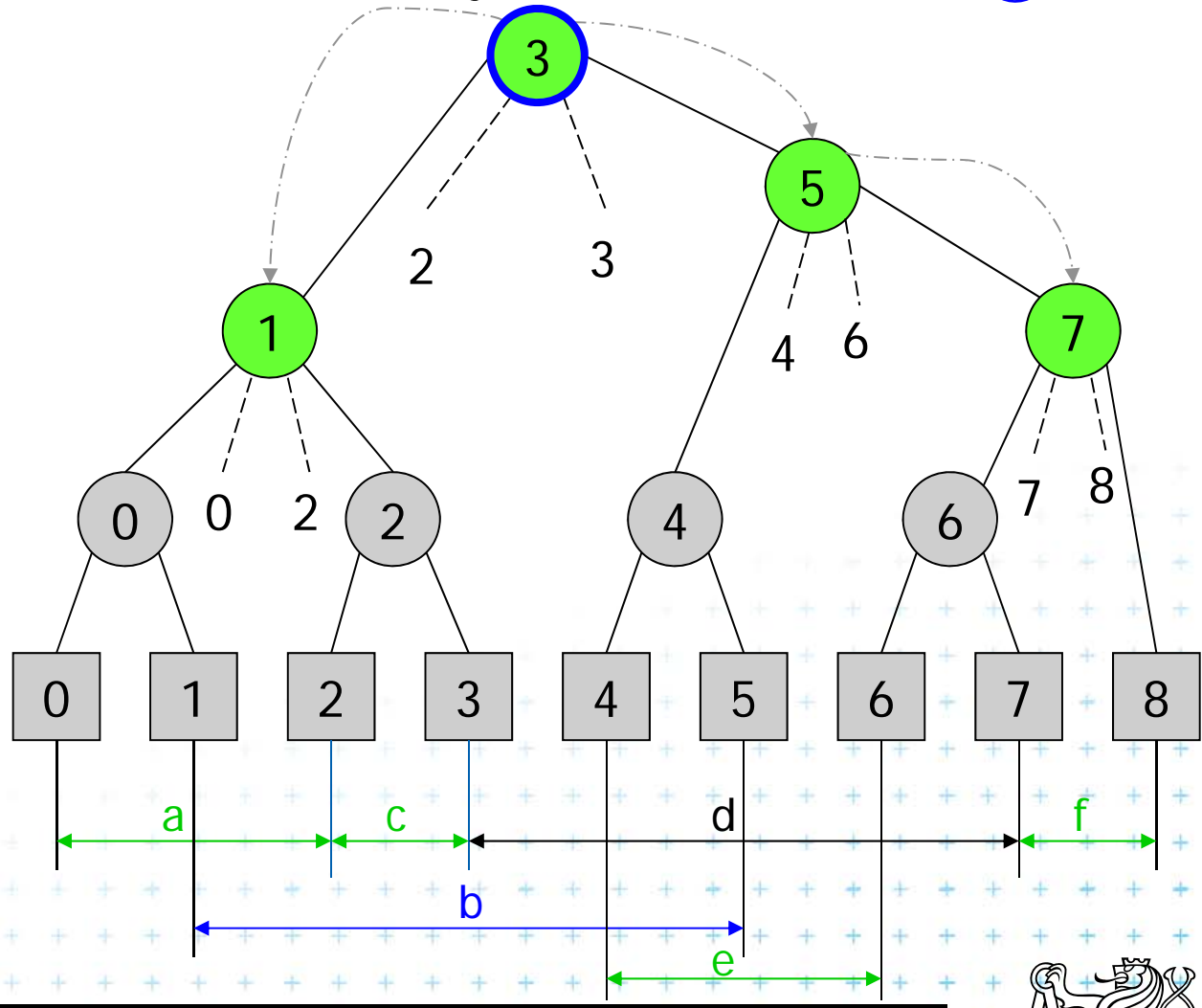
# Delete [1,5] Delete Interval




$$b \leq H(v) \leq e$$

$$? 1 \leq 3 \leq 5 ?$$



Delete the interval [1,5] from secondary lists



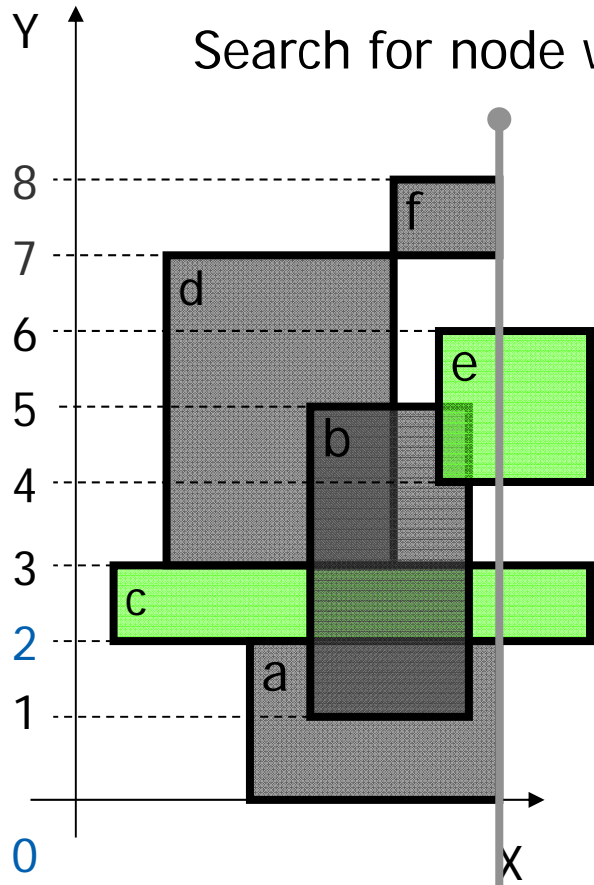
-  Active rectangle
-  Current node
-  Active node



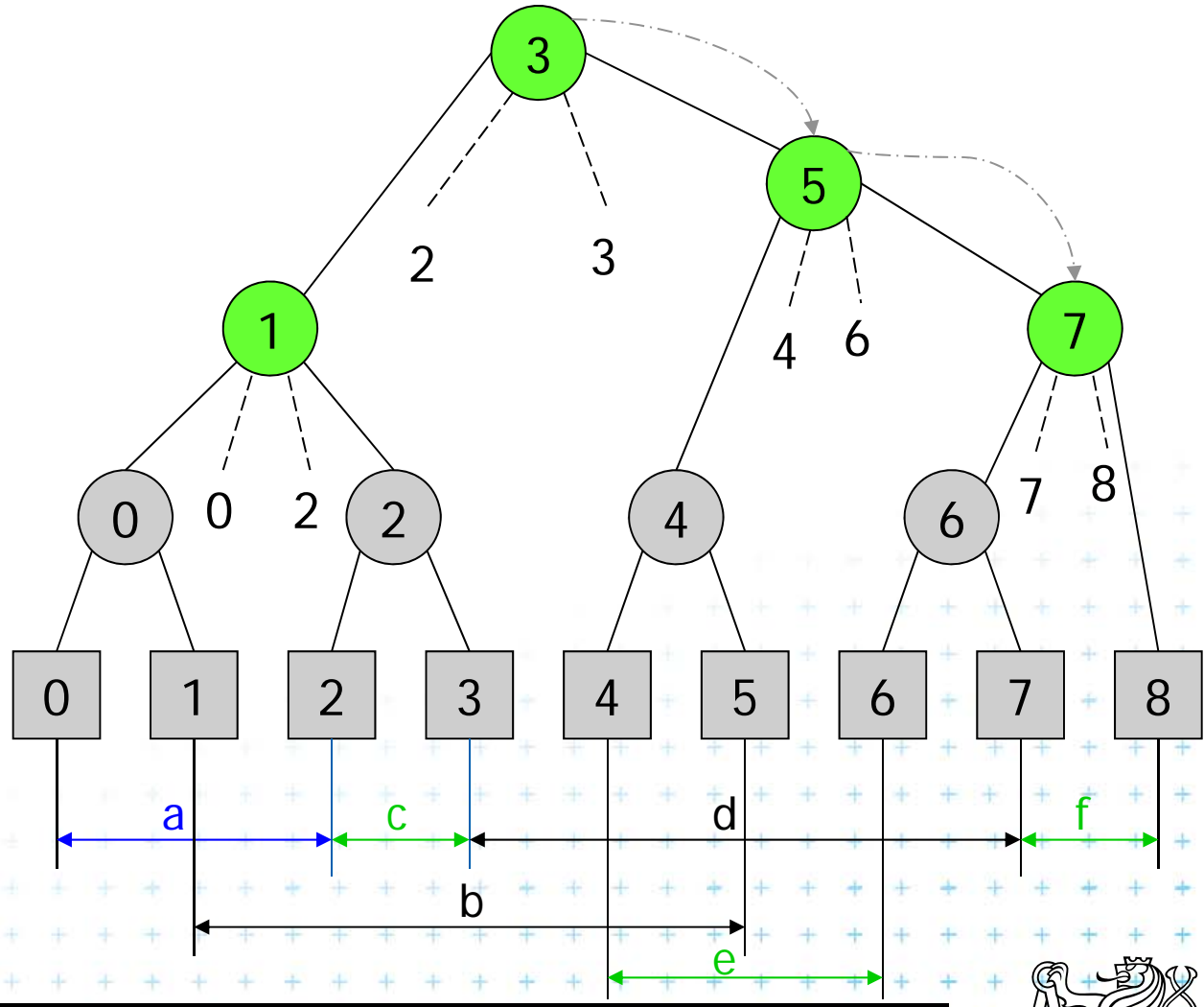
# Delete [0,2] Delete Interval 1/2

$$b < e \leq H(v)$$

$$? 0 < 2 \leq 3?$$



Search for node with interval [0,2]



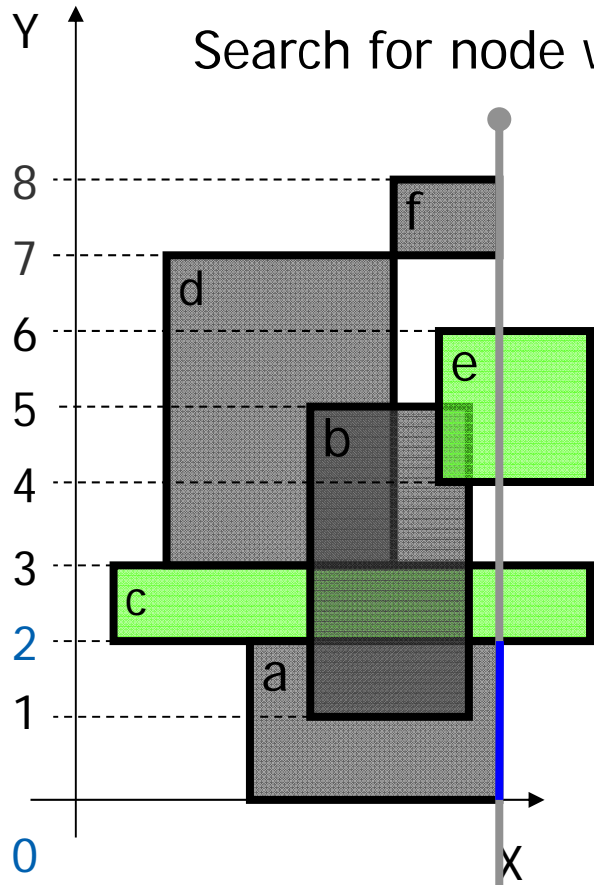
- Active rectangle
- Current node
- Active node



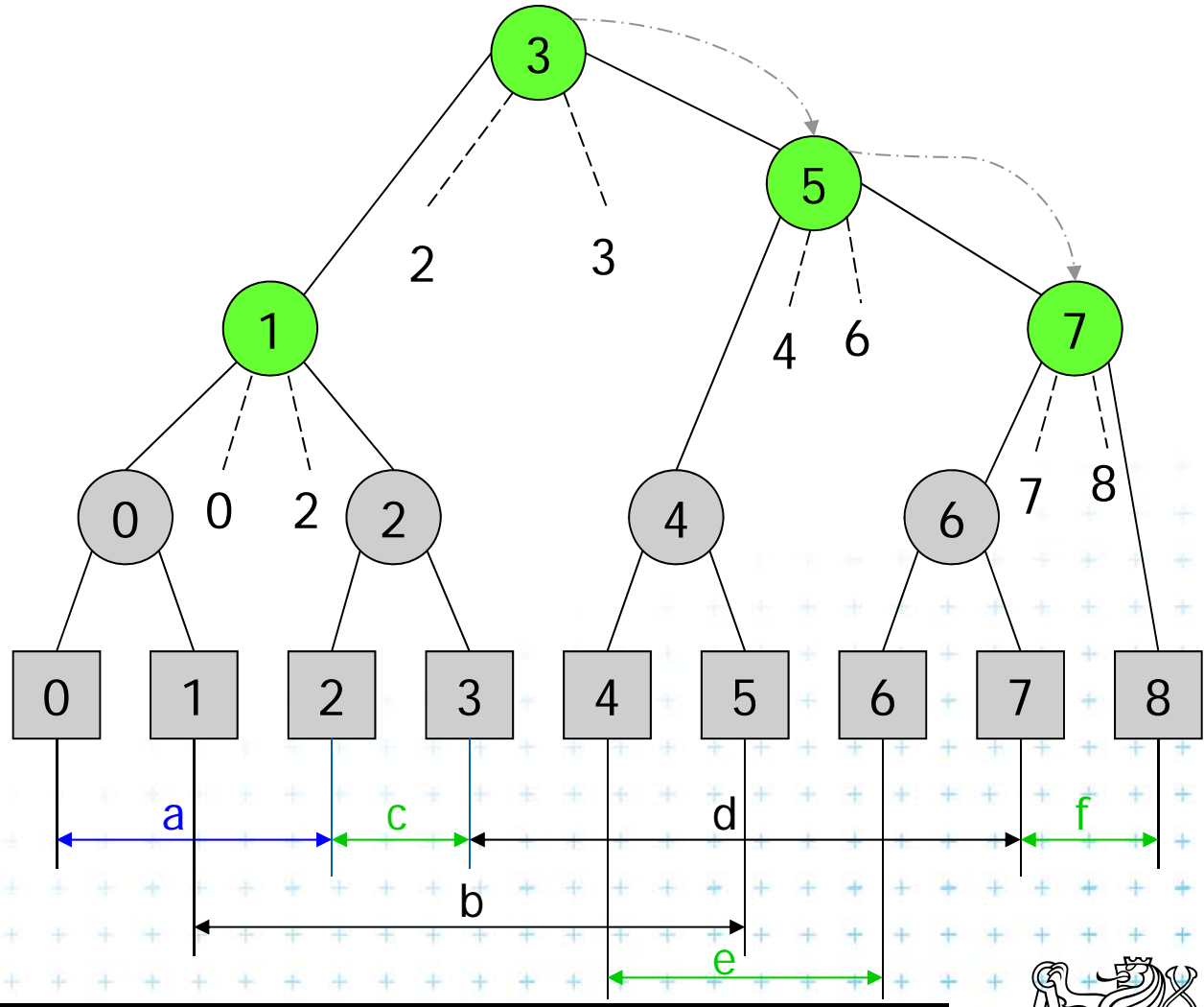
# Delete [0,2] Delete Interval 1/2




$$b < e \leq H(v)$$

$$? 0 < 2 \leq 3?$$



Search for node with interval [0,2]



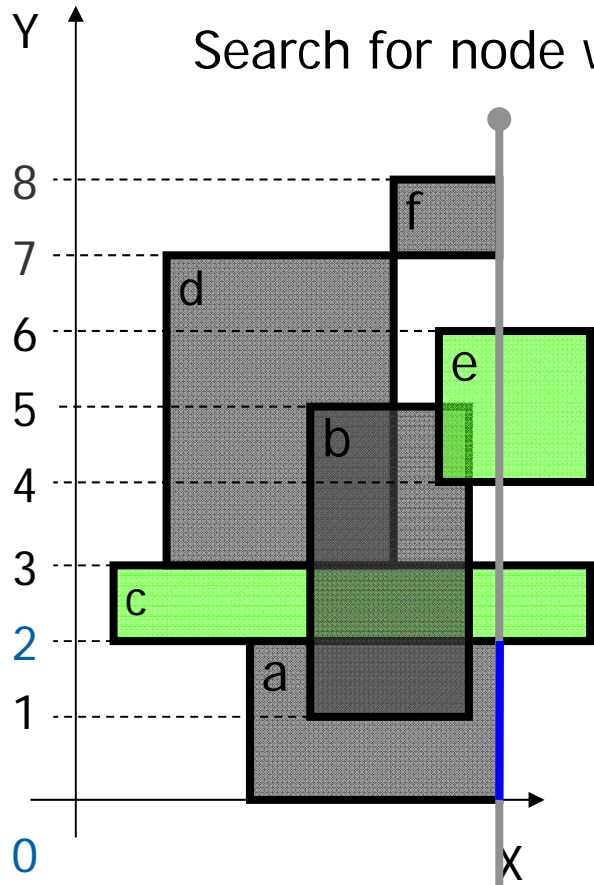
-  Active rectangle
-  Current node
-  Active node



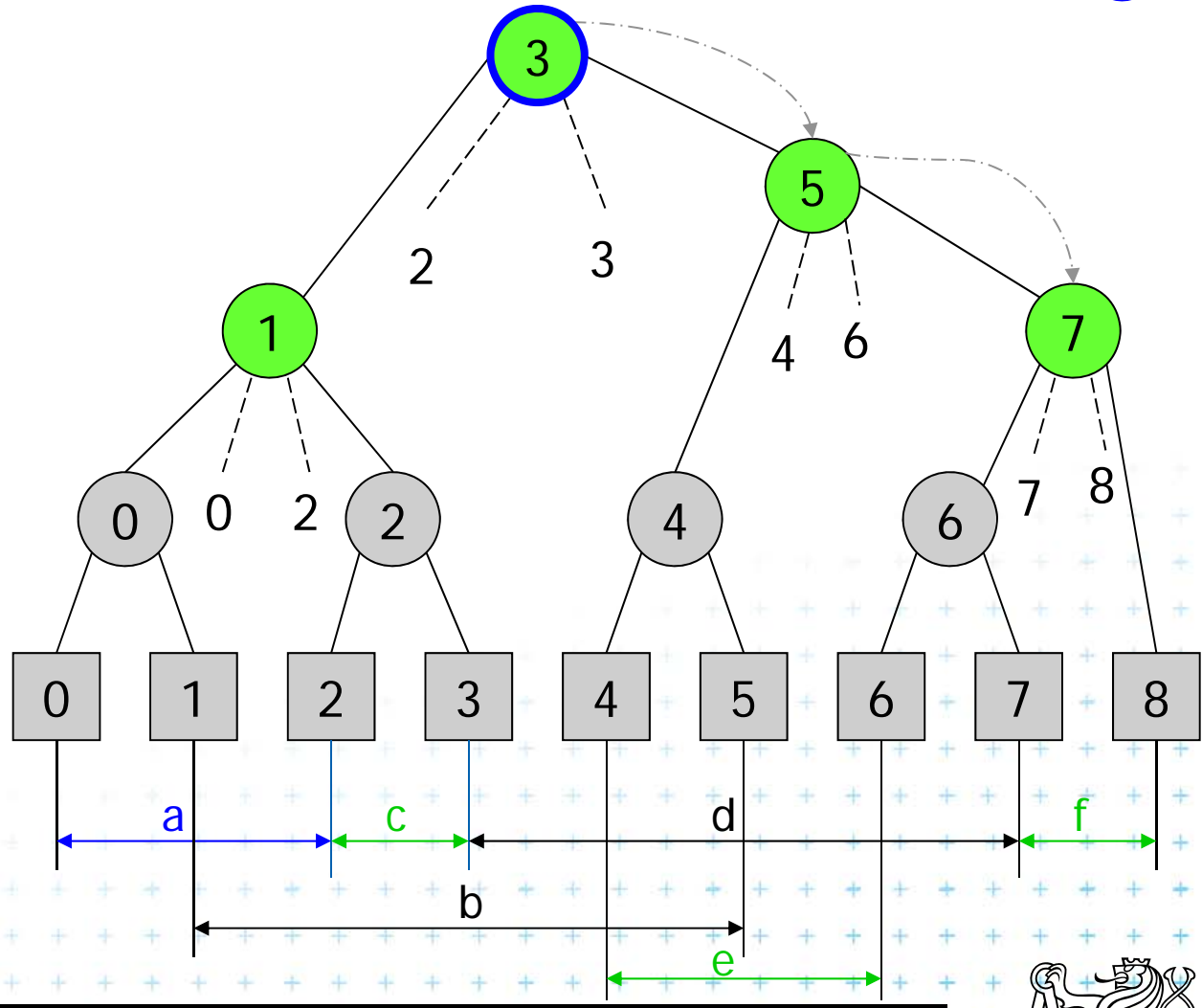
# Delete [0,2] Delete Interval 1/2

$$b < e \leq H(v)$$

$$? 0 < 2 \leq 3?$$

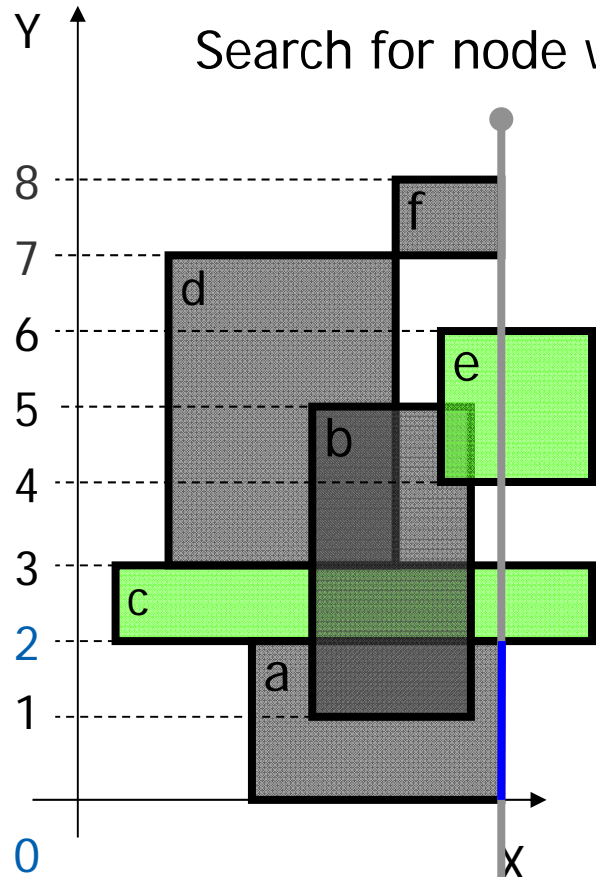


Search for node with interval [0,2]



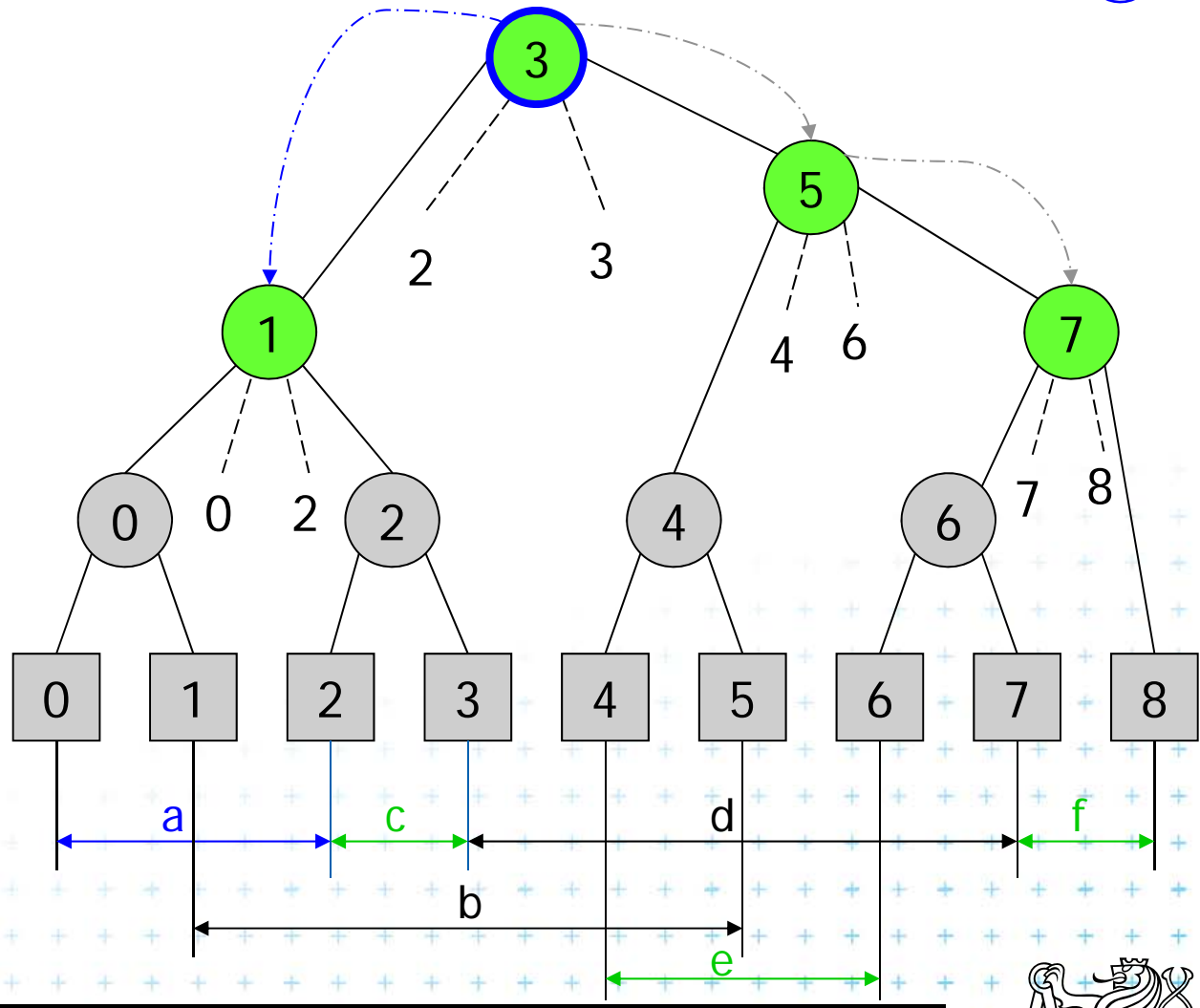
# Delete [0,2] Delete Interval 1/2




$$b < e \leq H(v)$$



Search for node with interval [0,2]

$$? 0 < 2 \leq 3?$$

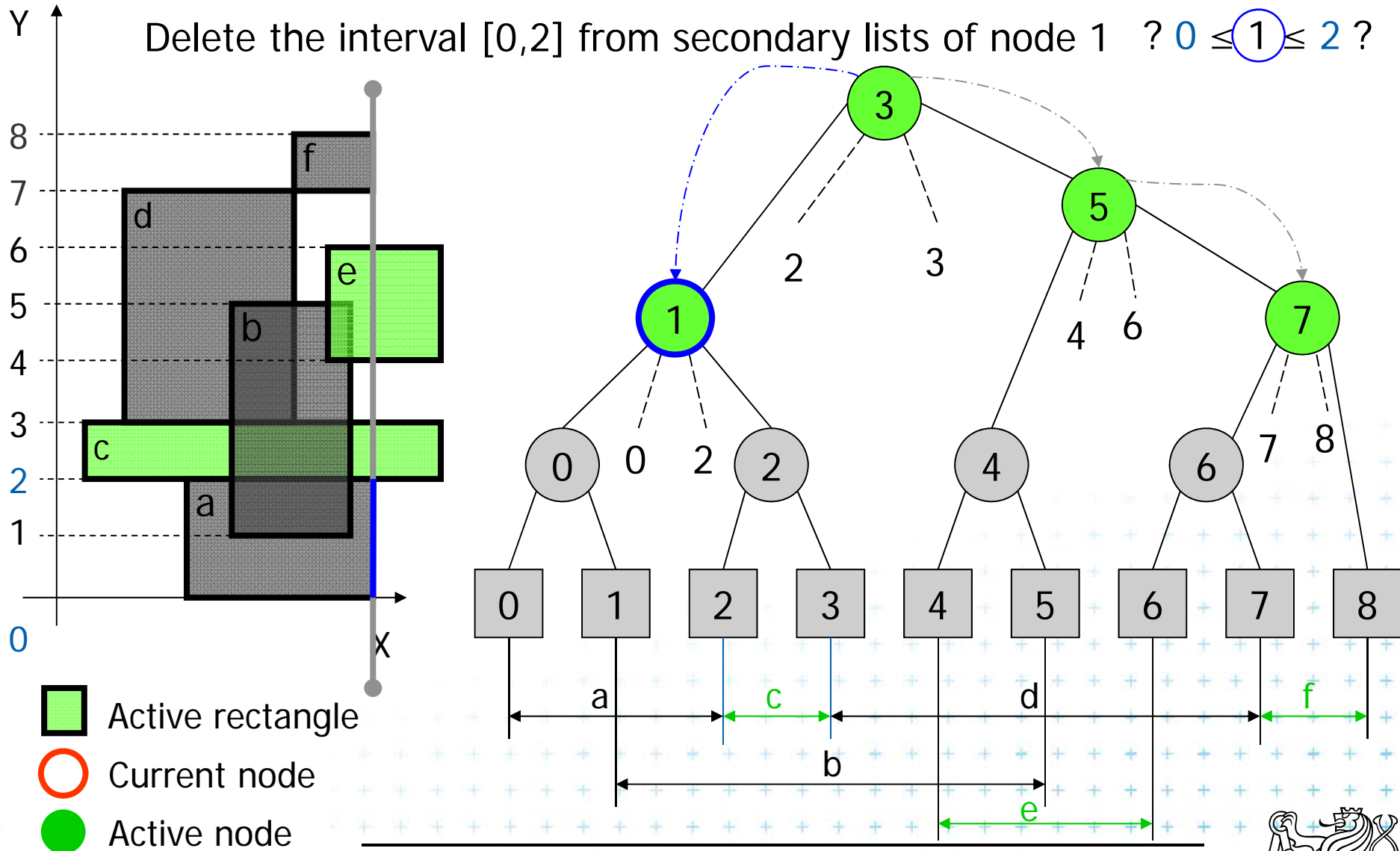


-  Active rectangle
-  Current node
-  Active node



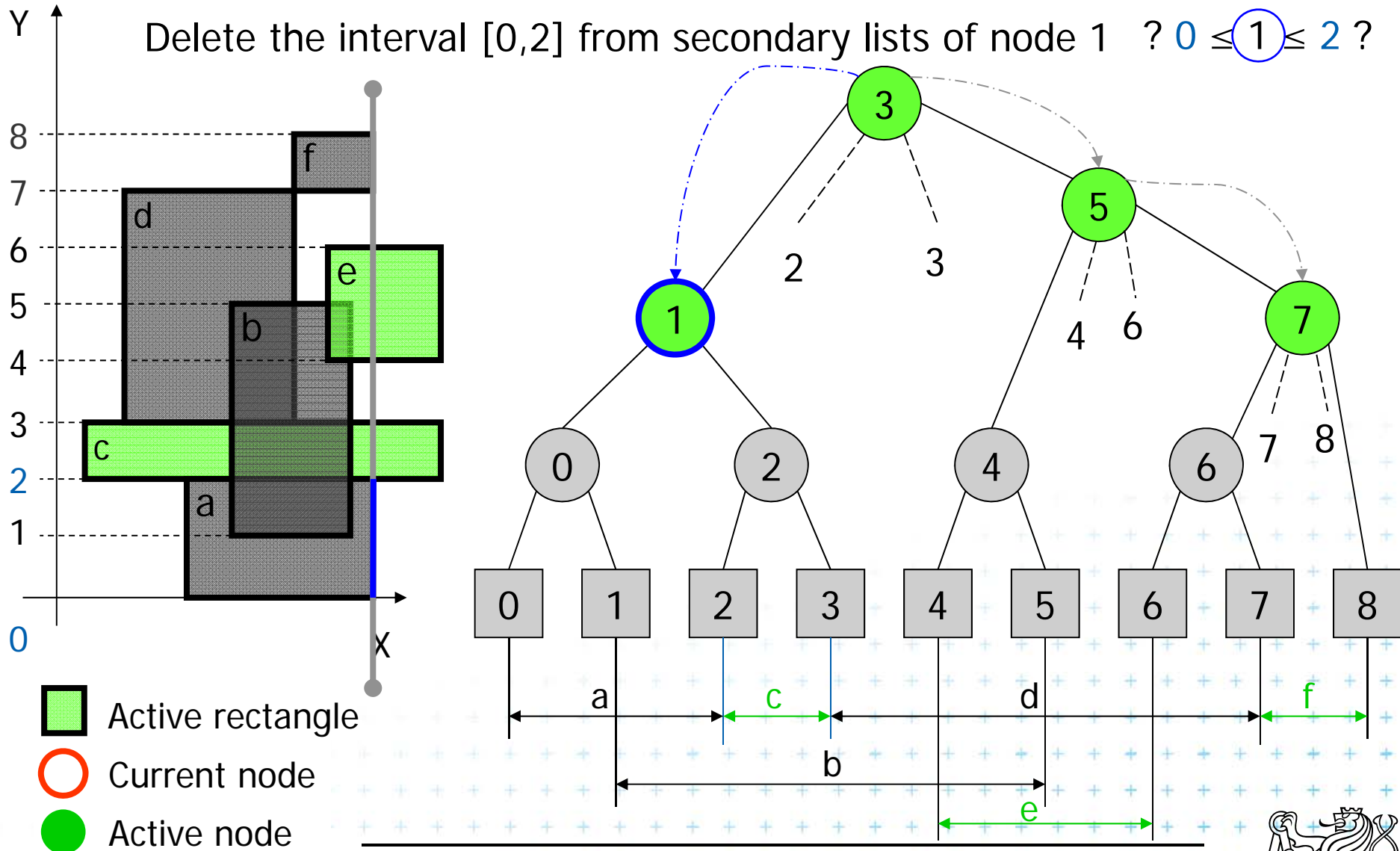
# Delete [0,2] Delete Interval 2/2

$$b \leq H(v) \leq e$$



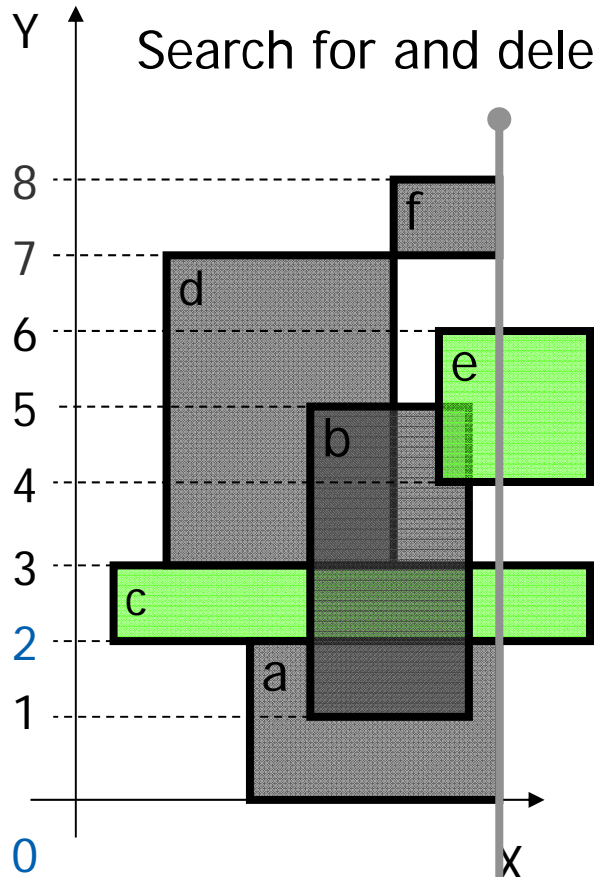
# Delete [0,2] Delete Interval 2/2

$$b \leq H(v) \leq e$$

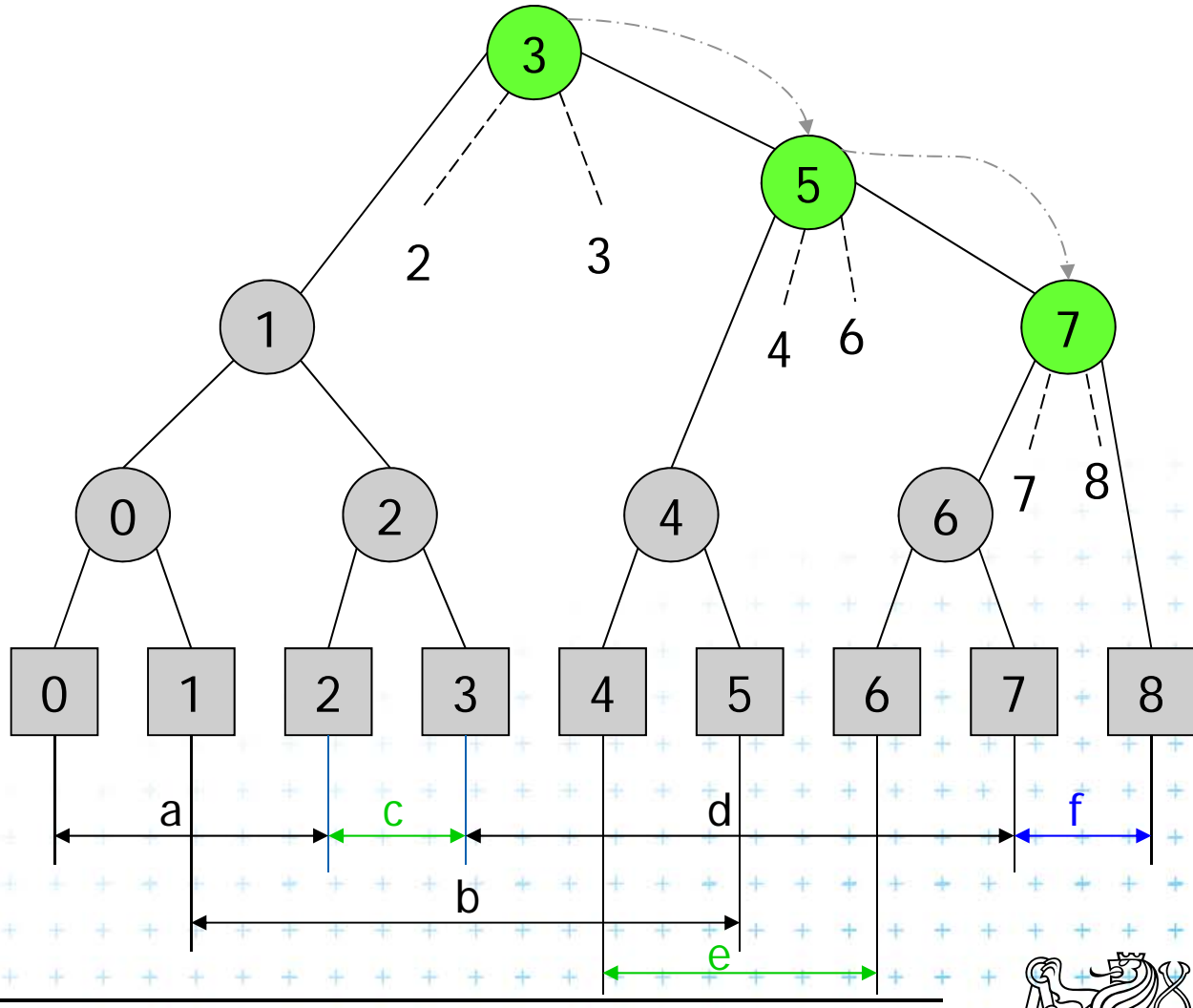





# Delete [7,8] Delete Interval

$$b \leq H(v) \leq e$$



Search for and delete node with interval [7,8]



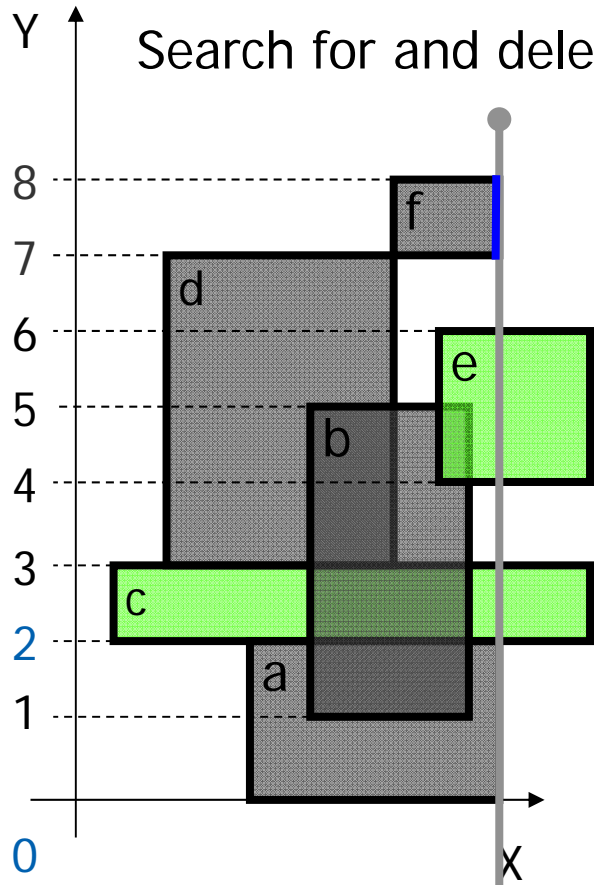
-  Active rectangle
-  Current node
-  Active node



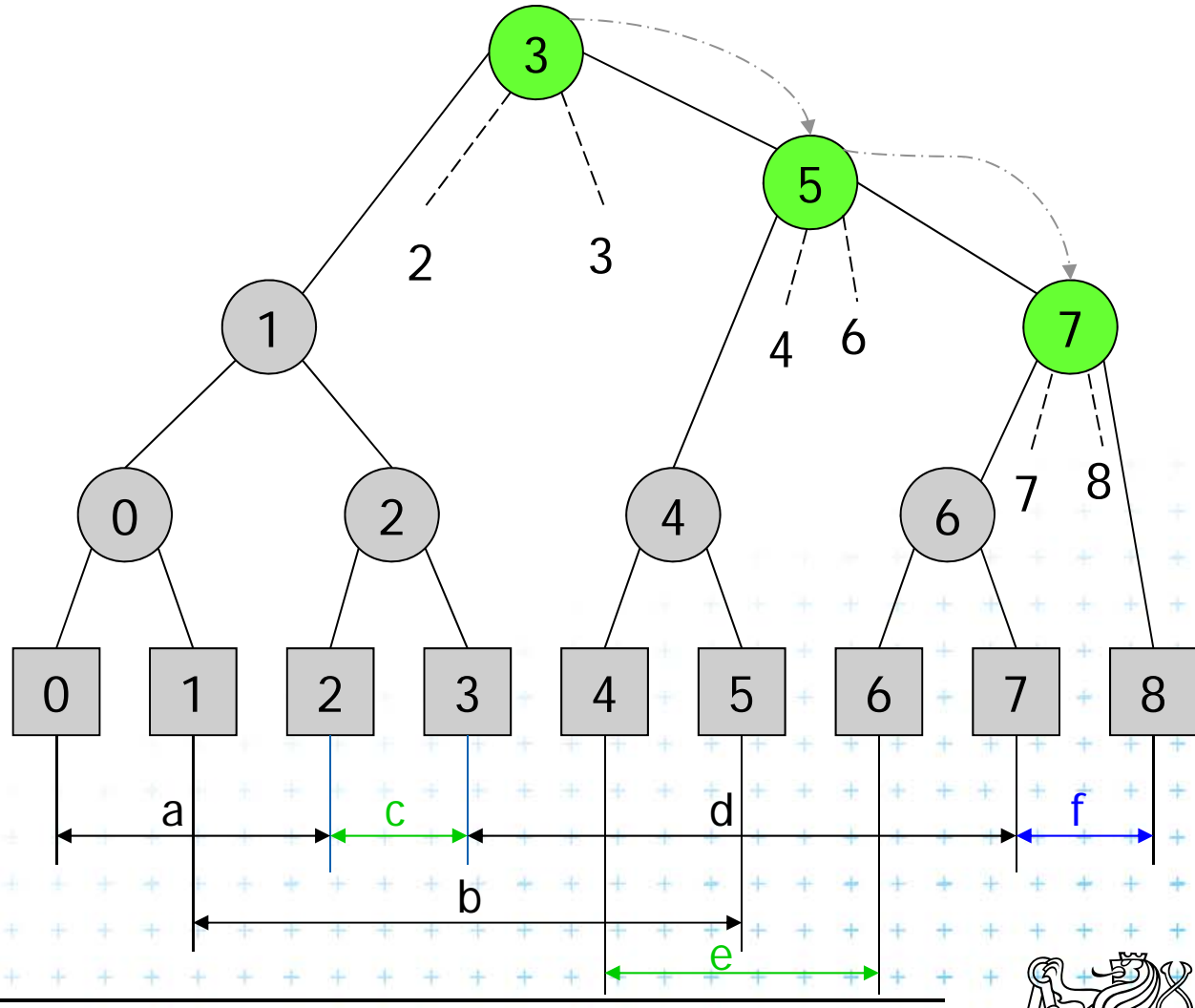


# Delete [7,8] Delete Interval

$$b \leq H(v) \leq e$$



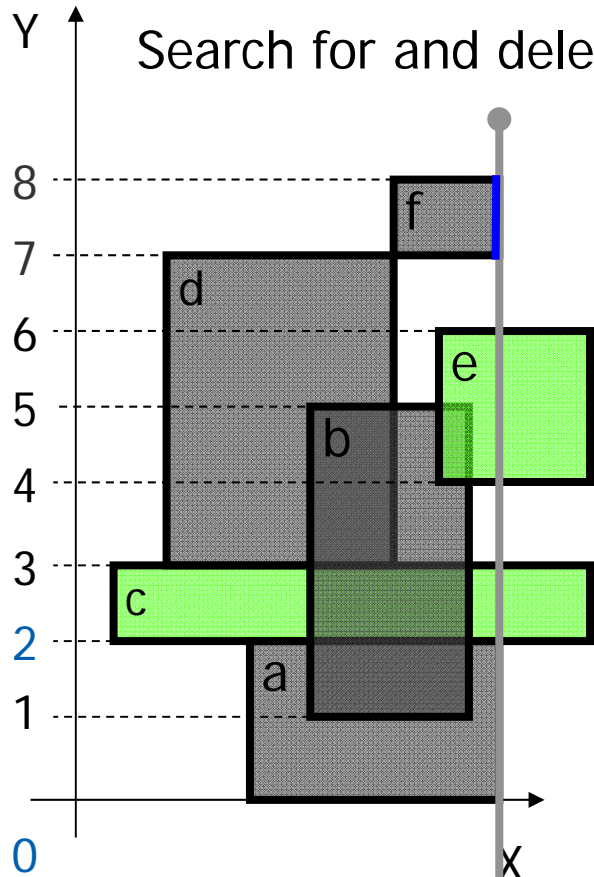
Search for and delete node with interval [7,8]



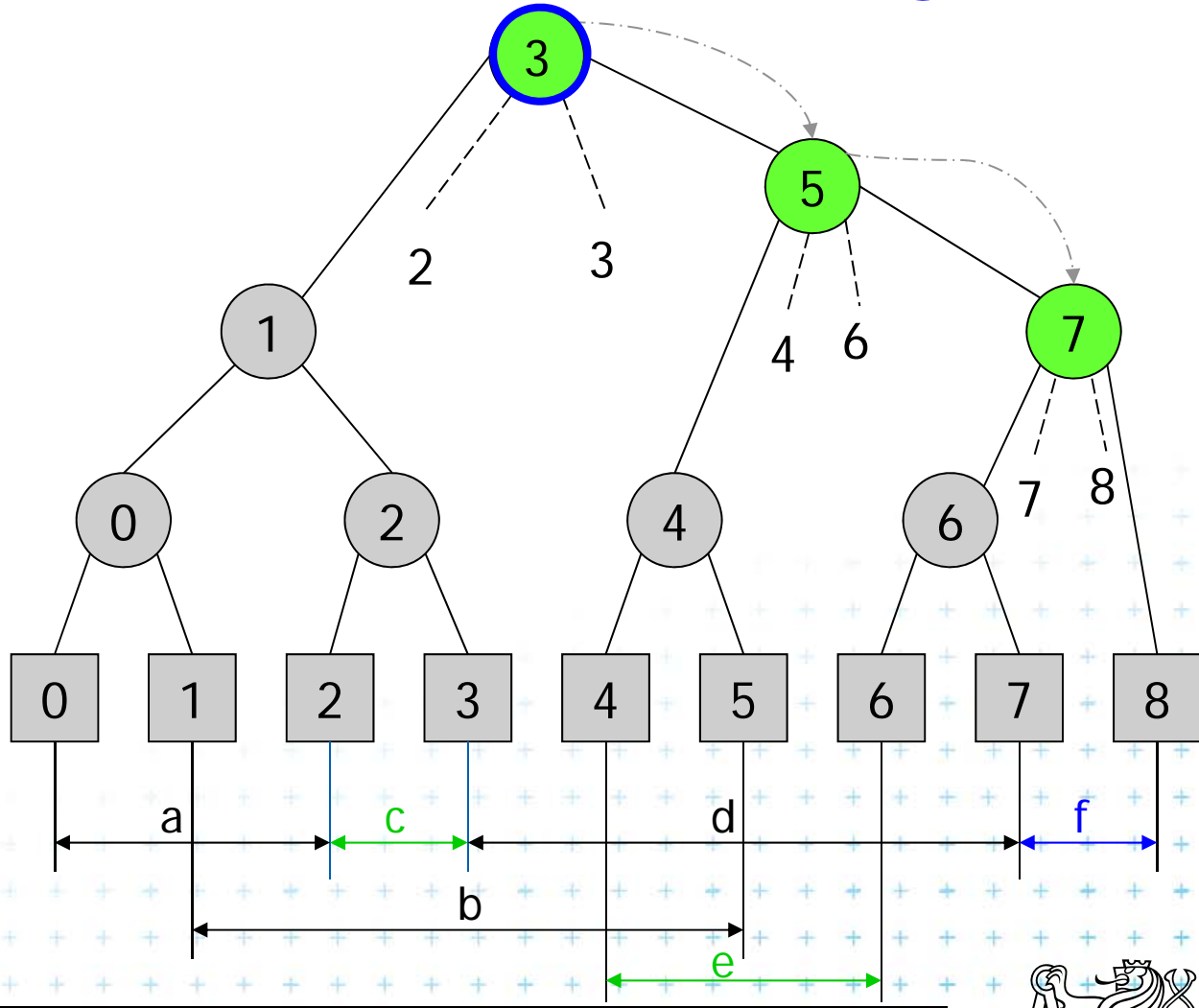
# Delete [7,8] Delete Interval

$$b \leq H(v) \leq e$$

$$? 3 \leq 7 < 8 ?$$



Search for and delete node with interval [7,8]

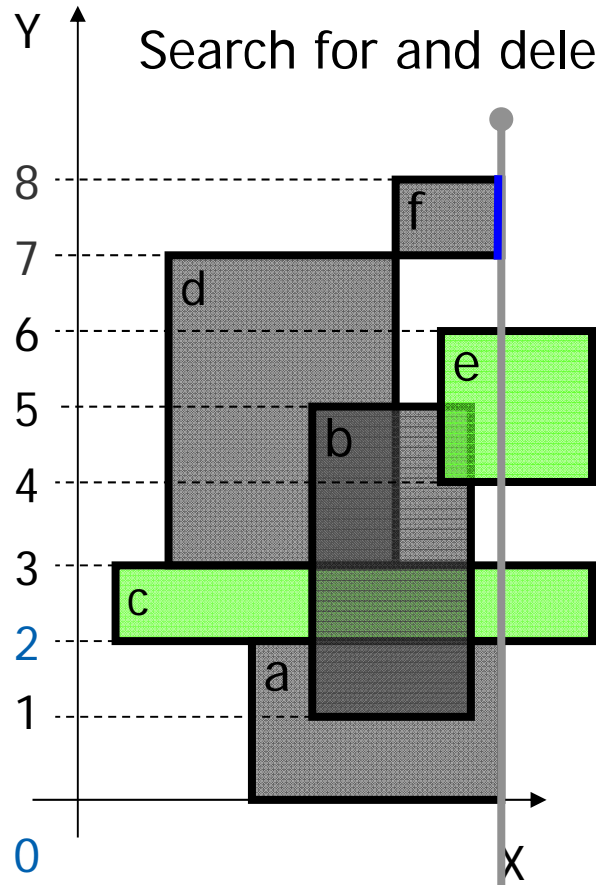


- Active rectangle
- Current node
- Active node



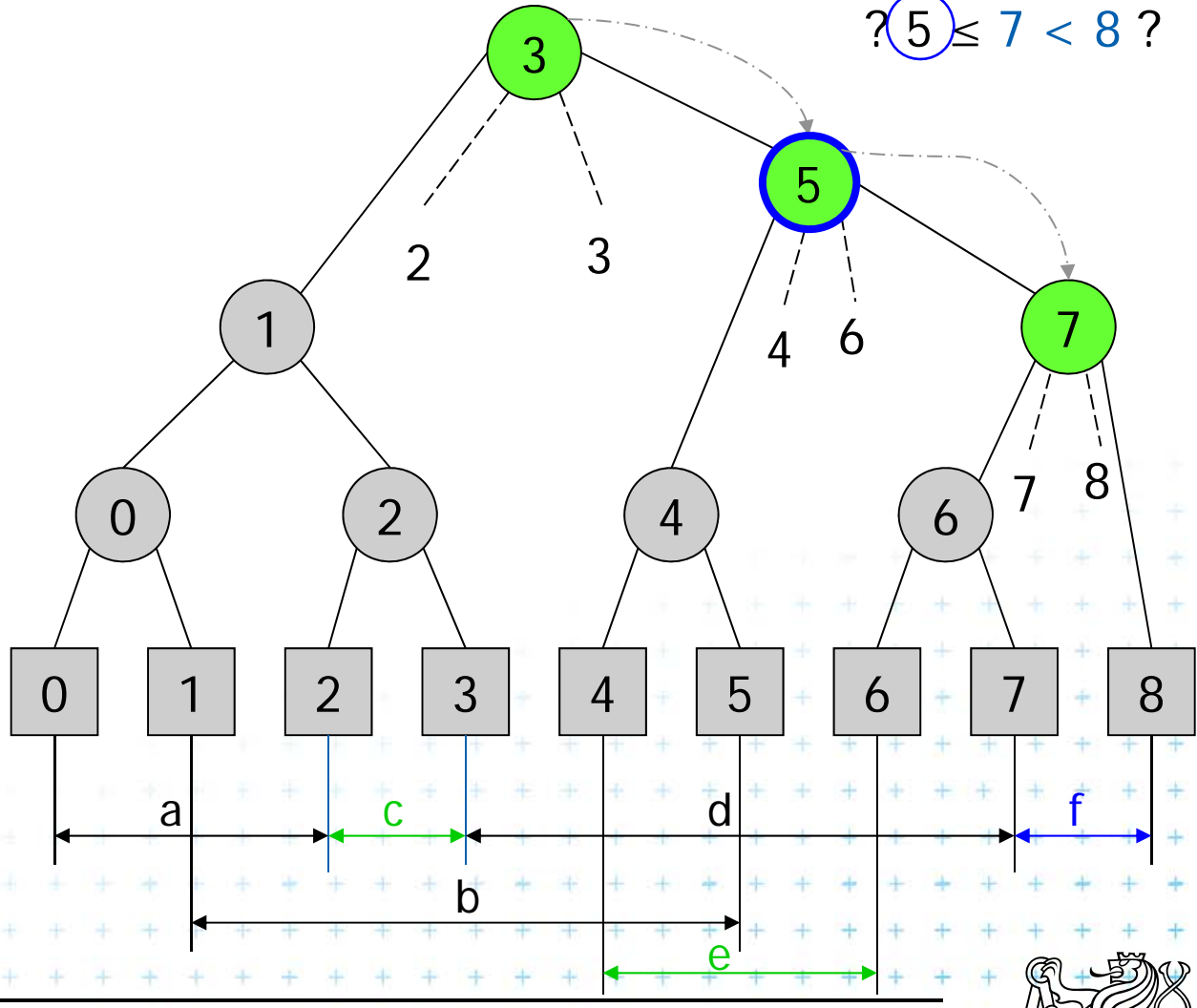
# Delete [7,8] Delete Interval




$$b \leq H(v) \leq e$$



Search for and delete node with interval [7,8]

?  $3 \leq 7 < 8$  ?  
 ?  $5 \leq 7 < 8$  ?

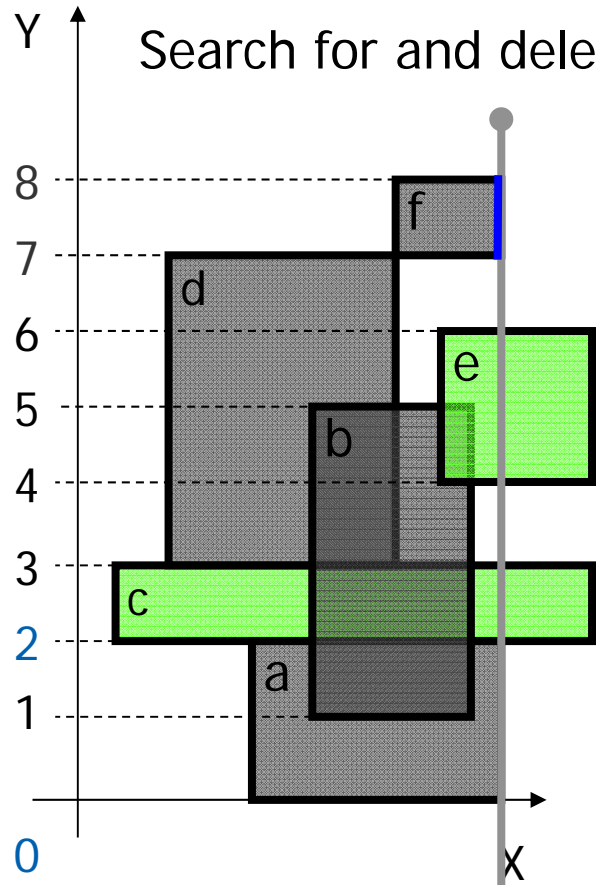


-  Active rectangle
-  Current node
-  Active node



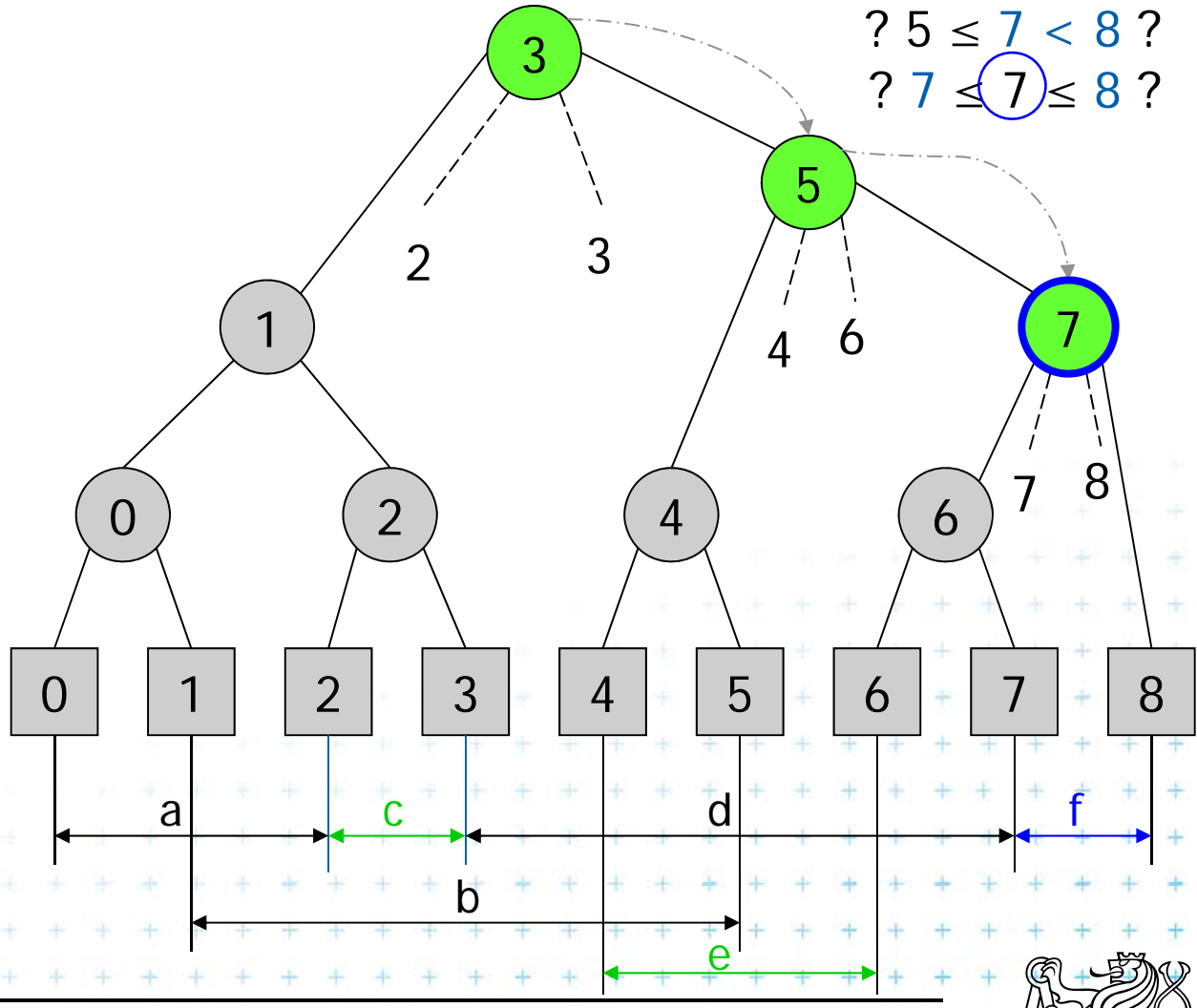
# Delete [7,8] Delete Interval




$$b \leq H(v) \leq e$$



Search for and delete node with interval [7,8]

?  $3 \leq 7 < 8$  ?  
 ?  $5 \leq 7 < 8$  ?  
 ?  $7 \leq 7 \leq 8$  ?

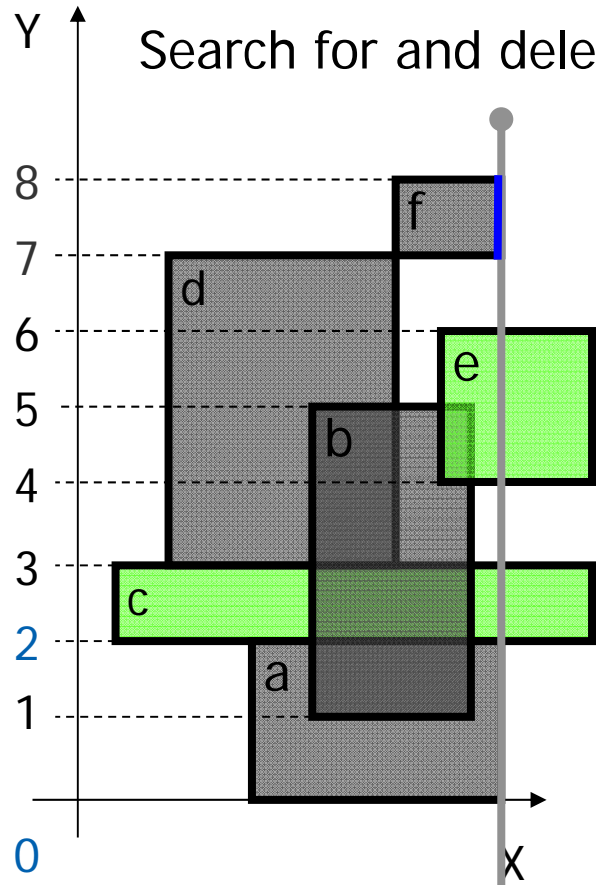


-  Active rectangle
-  Current node
-  Active node



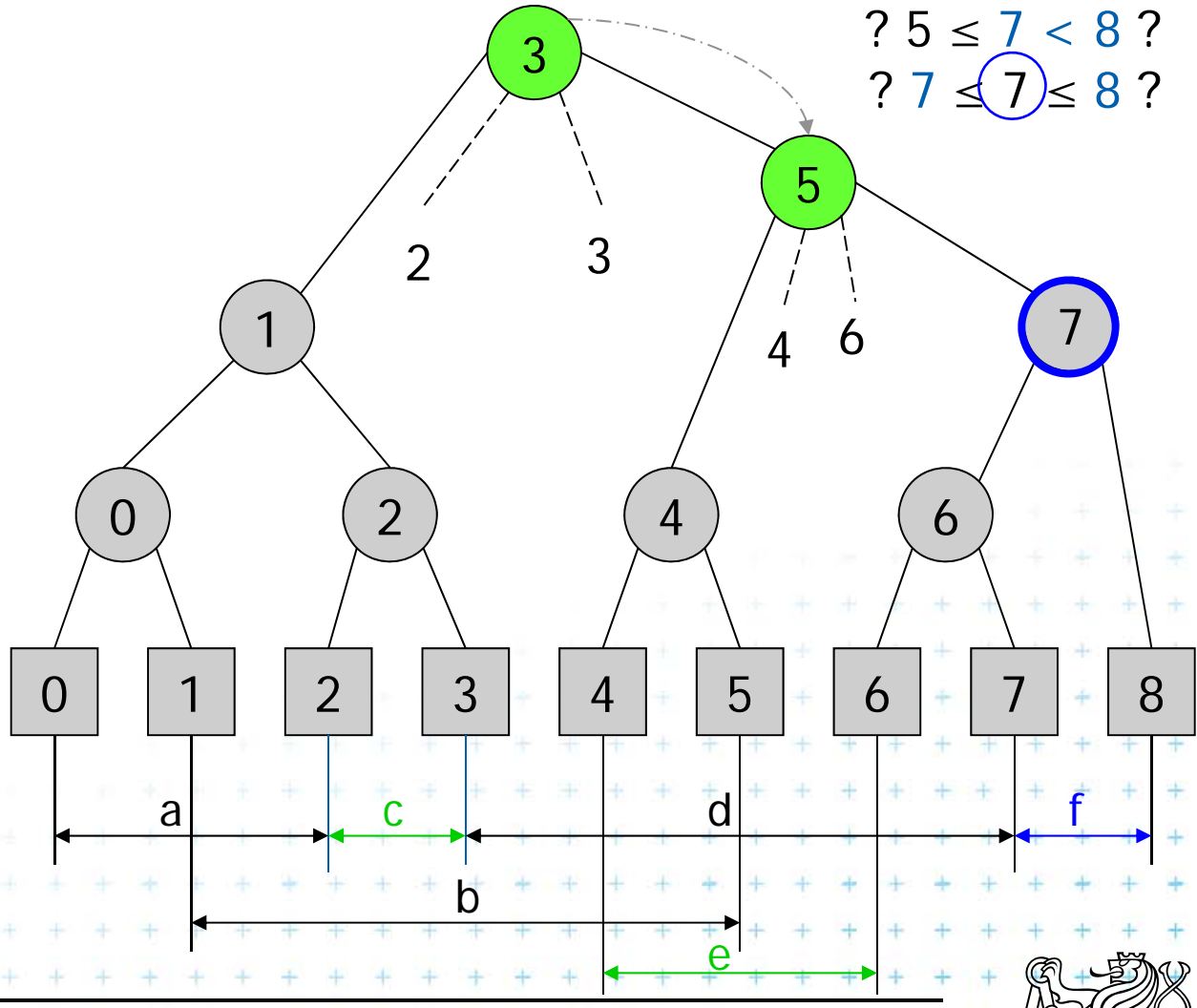
# Delete [7,8] Delete Interval

$$b \leq H(v) \leq e$$



Search for and delete node with interval [7,8]

?  $3 \leq 7 < 8$  ?  
 ?  $5 \leq 7 < 8$  ?  
 ?  $7 \leq 7 \leq 8$  ?



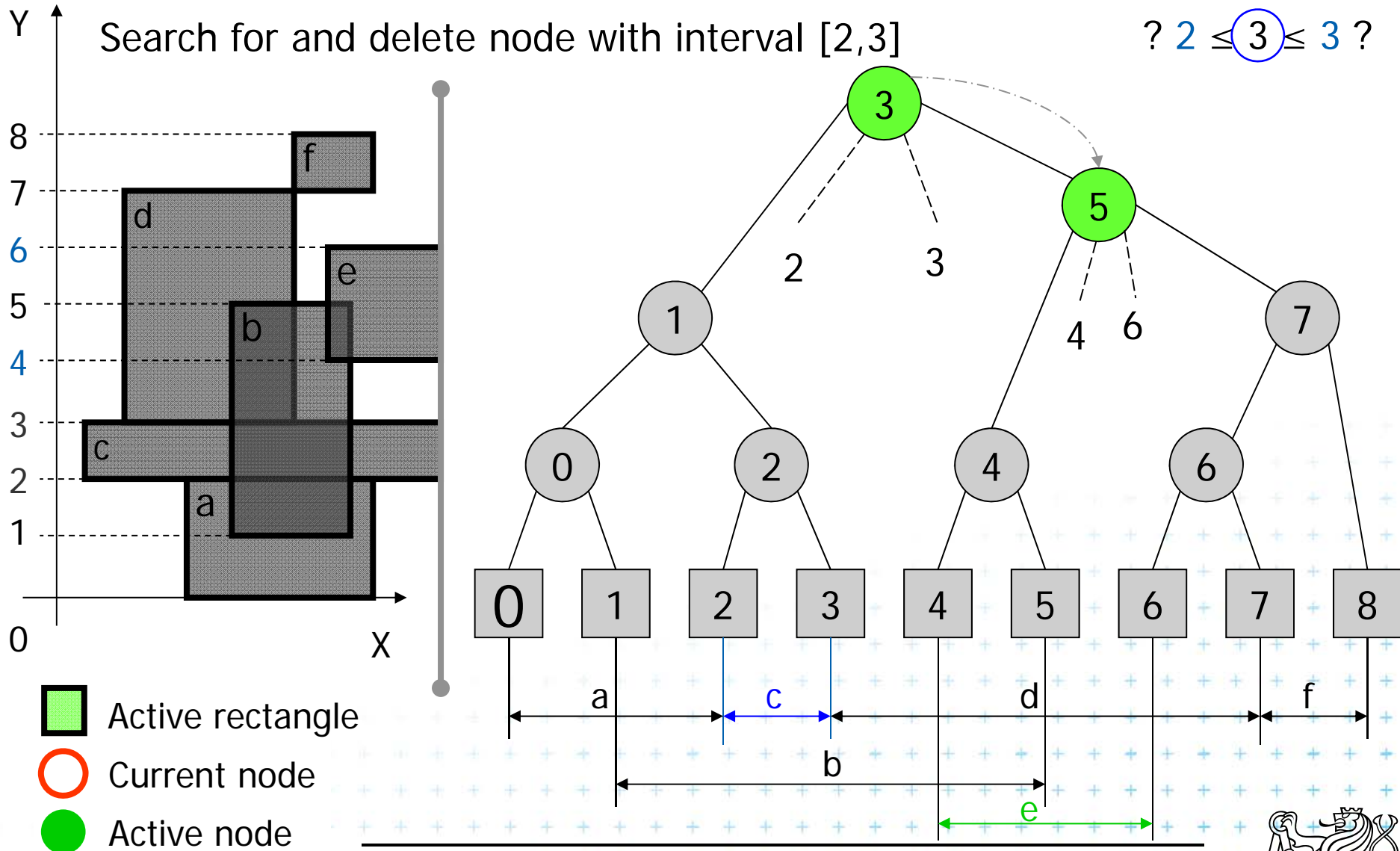
- Active rectangle
- Current node
- Active node



# Delete [2,3] Delete Interval

$$b \leq H(v) \leq e$$

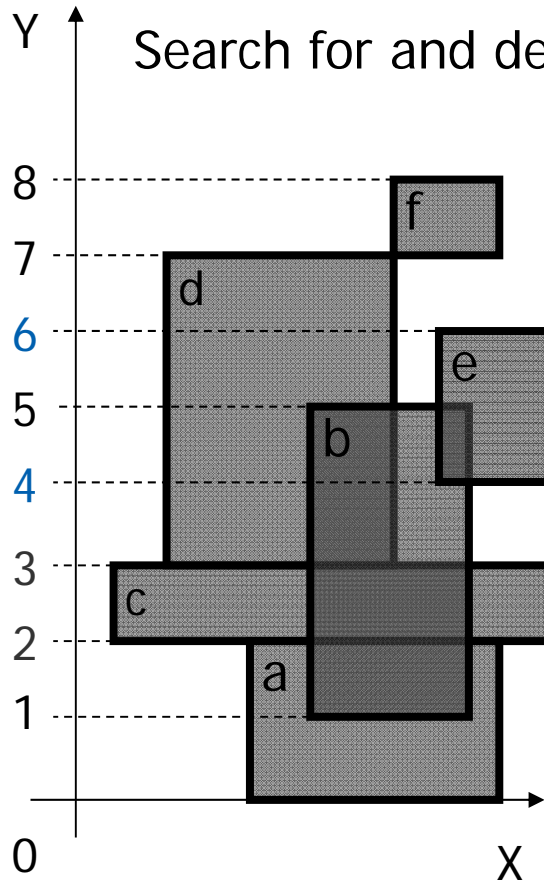
$$? 2 \leq \textcircled{3} \leq 3 ?$$



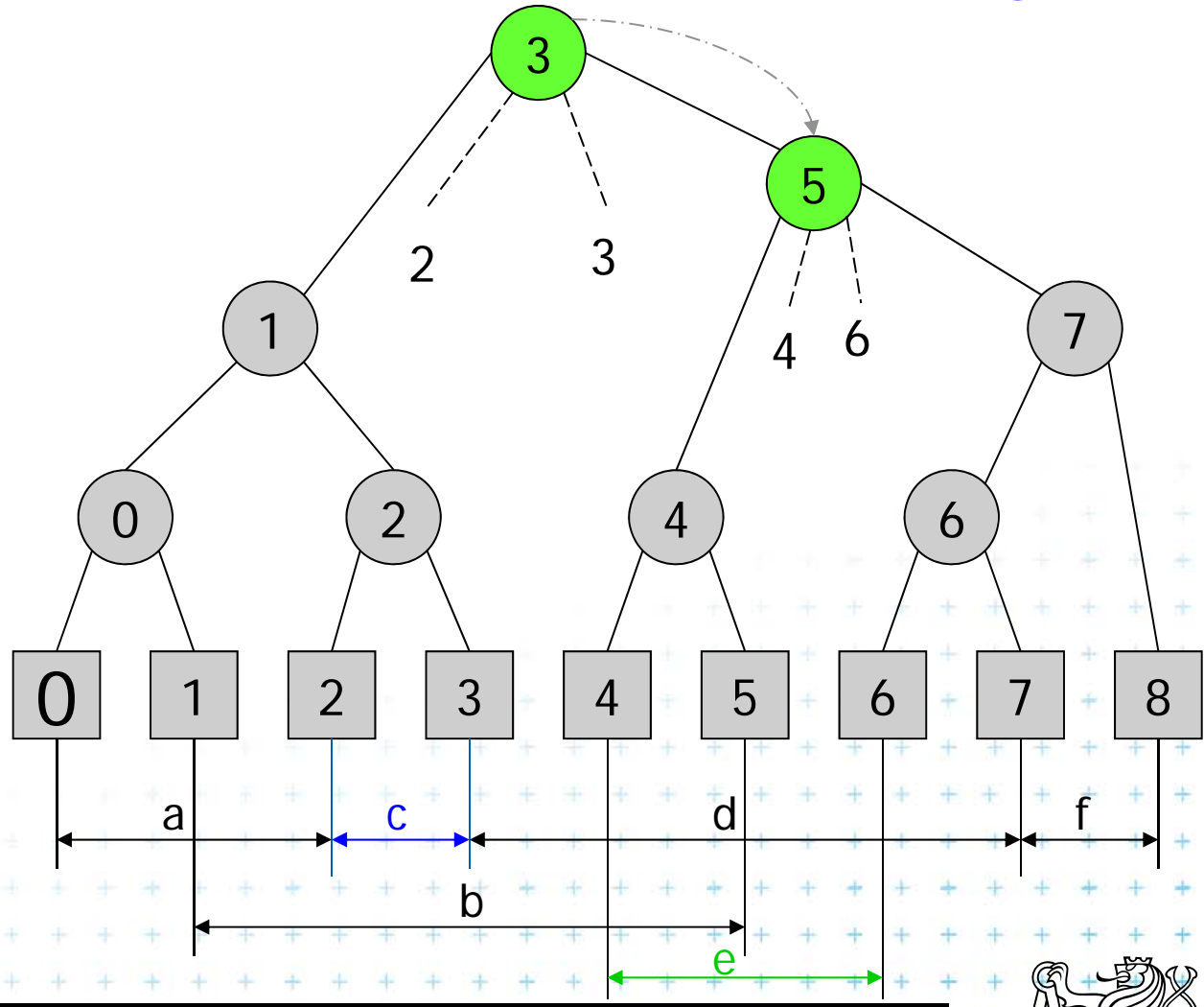
# Delete [2,3] Delete Interval

$$b \leq H(v) \leq e$$

$$? 2 \leq 3 \leq 3 ?$$



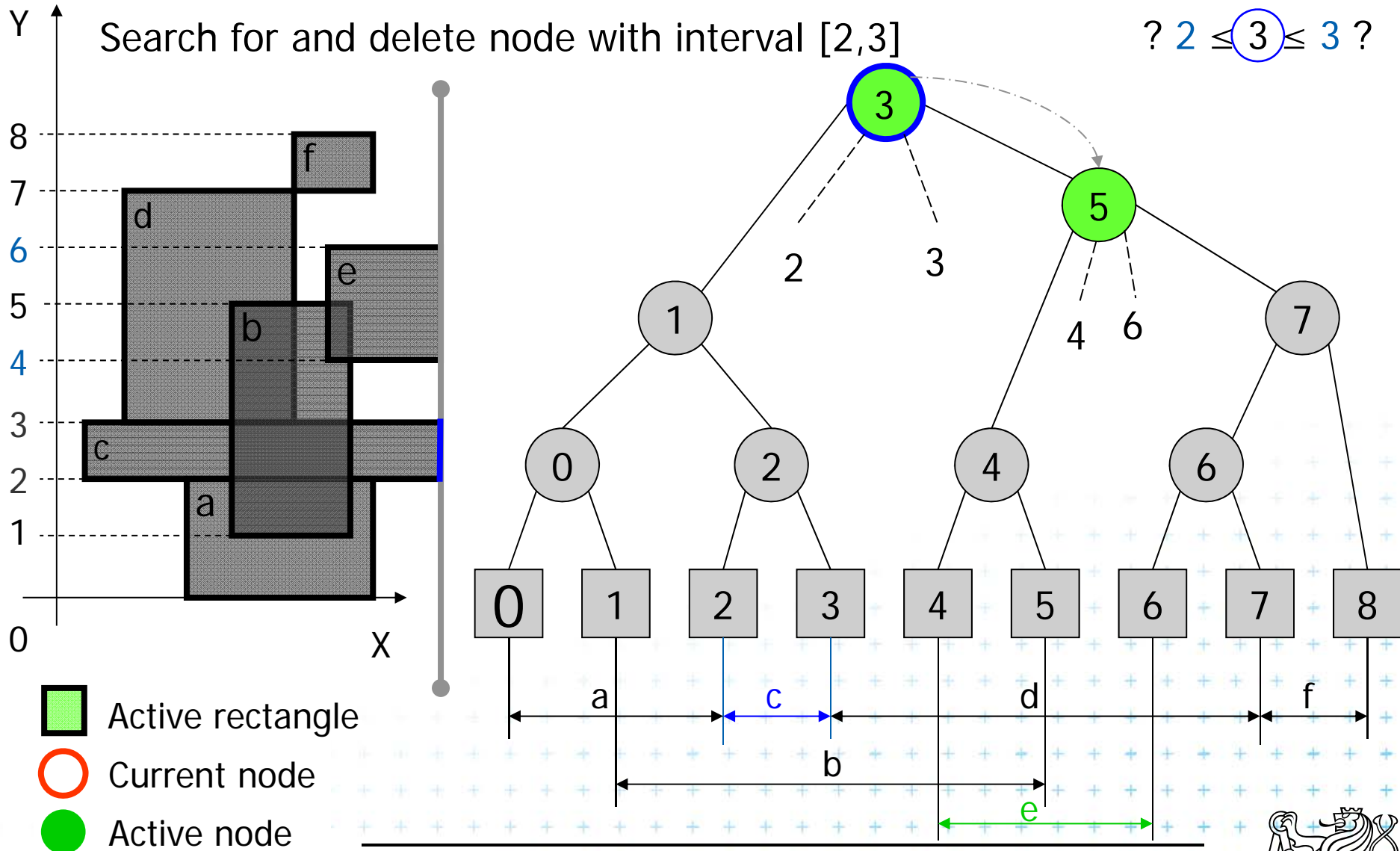
Search for and delete node with interval [2,3]



# Delete [2,3] Delete Interval

$$b \leq H(v) \leq e$$

$$? 2 \leq 3 \leq 3 ?$$

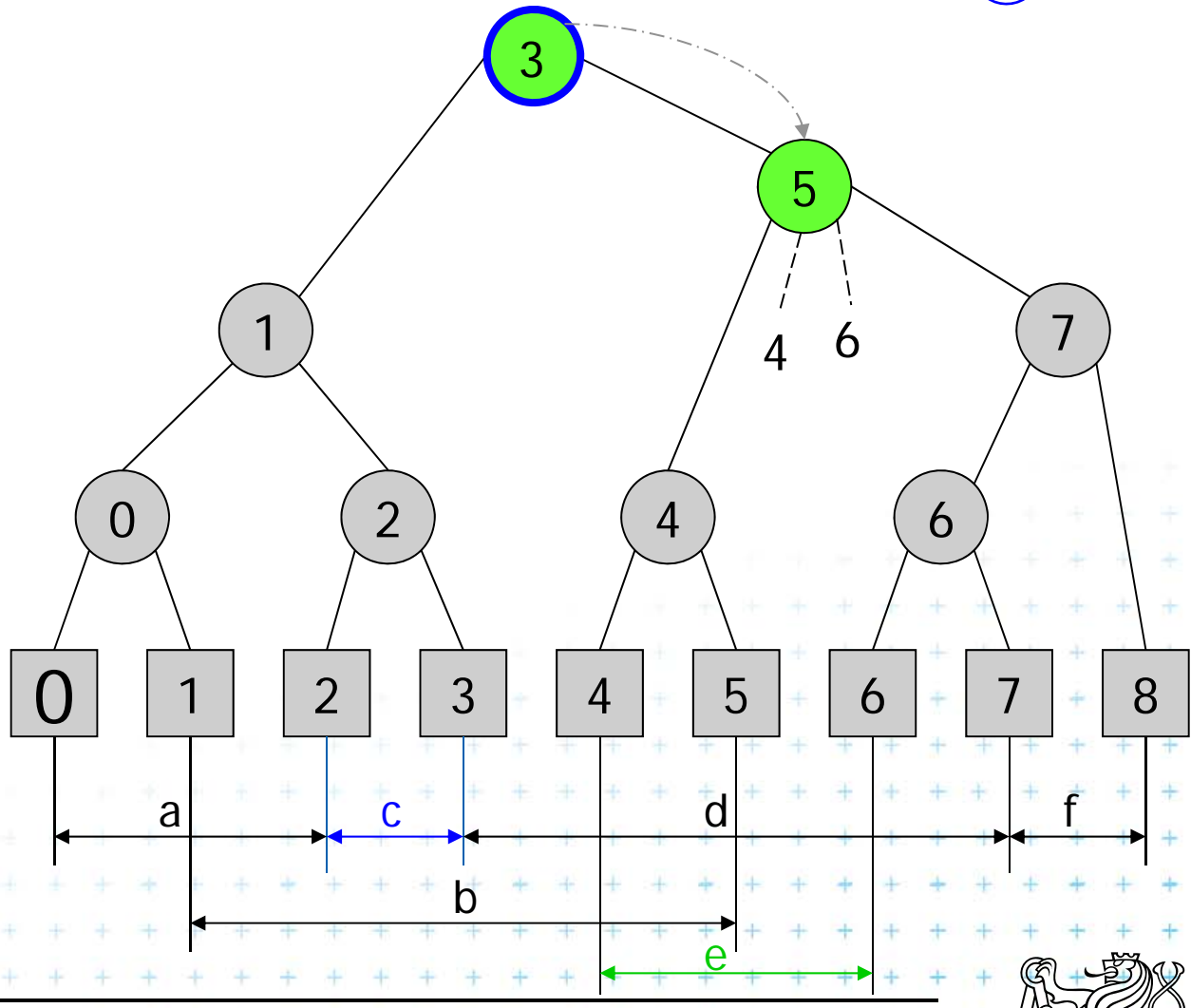
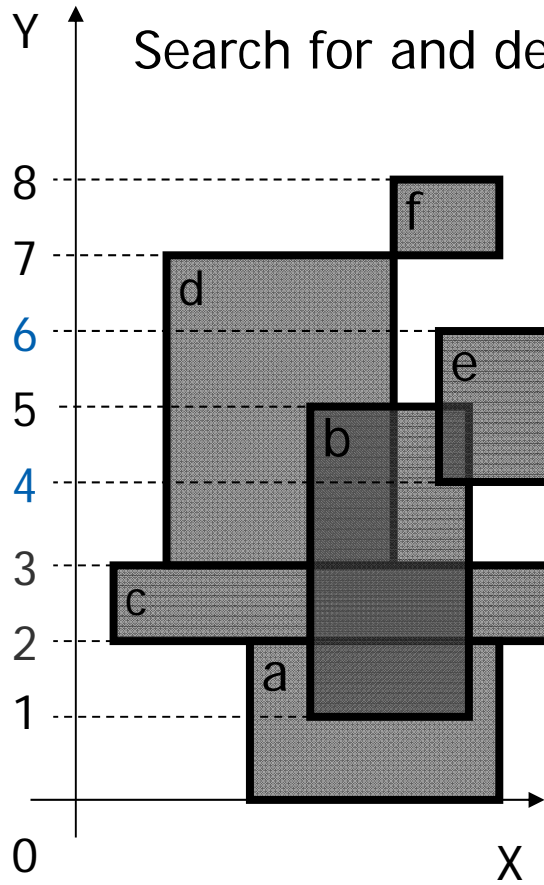




# Delete [2,3] Delete Interval

$$b \leq H(v) \leq e$$

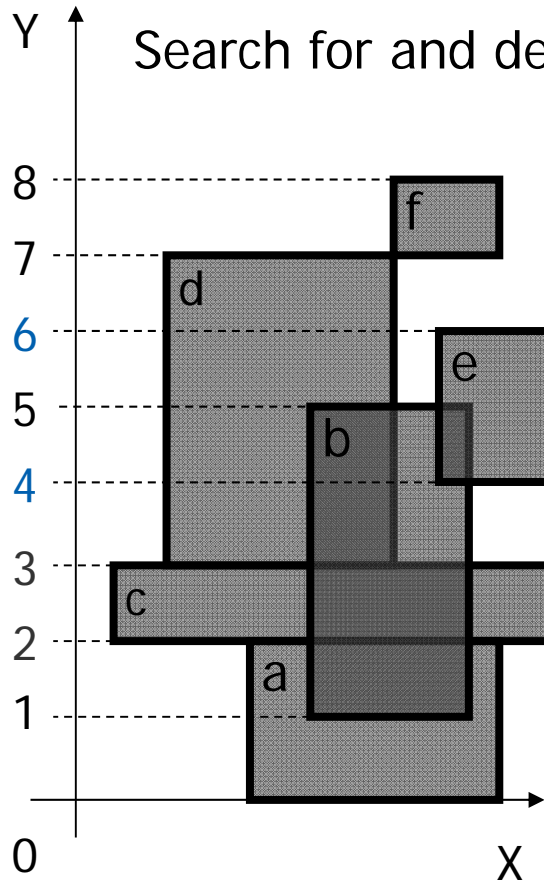
$$? 2 \leq 3 \leq 3 ?$$



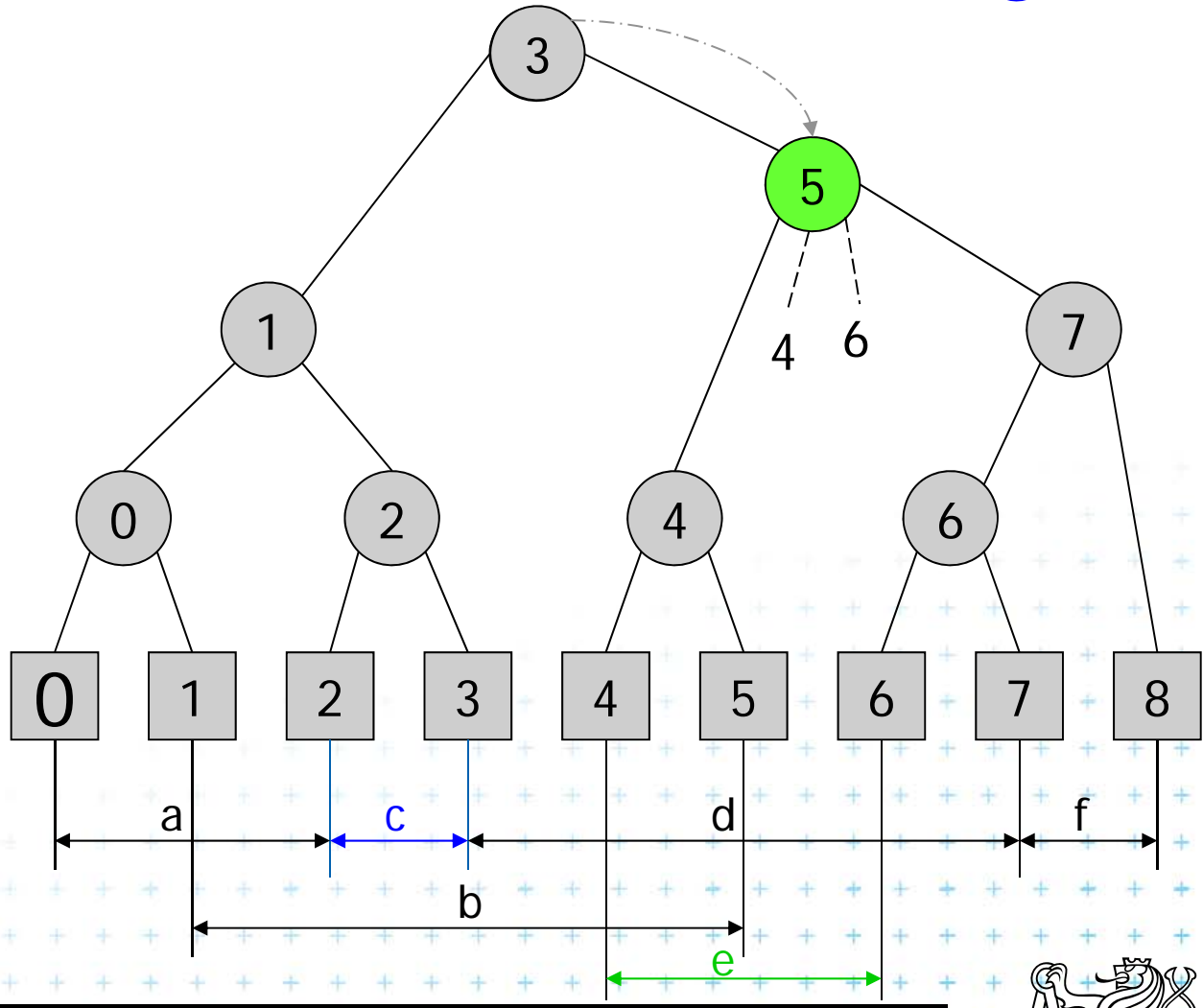
# Delete [2,3] Delete Interval

$$b \leq H(v) \leq e$$

$$? 2 \leq 3 \leq 3 ?$$



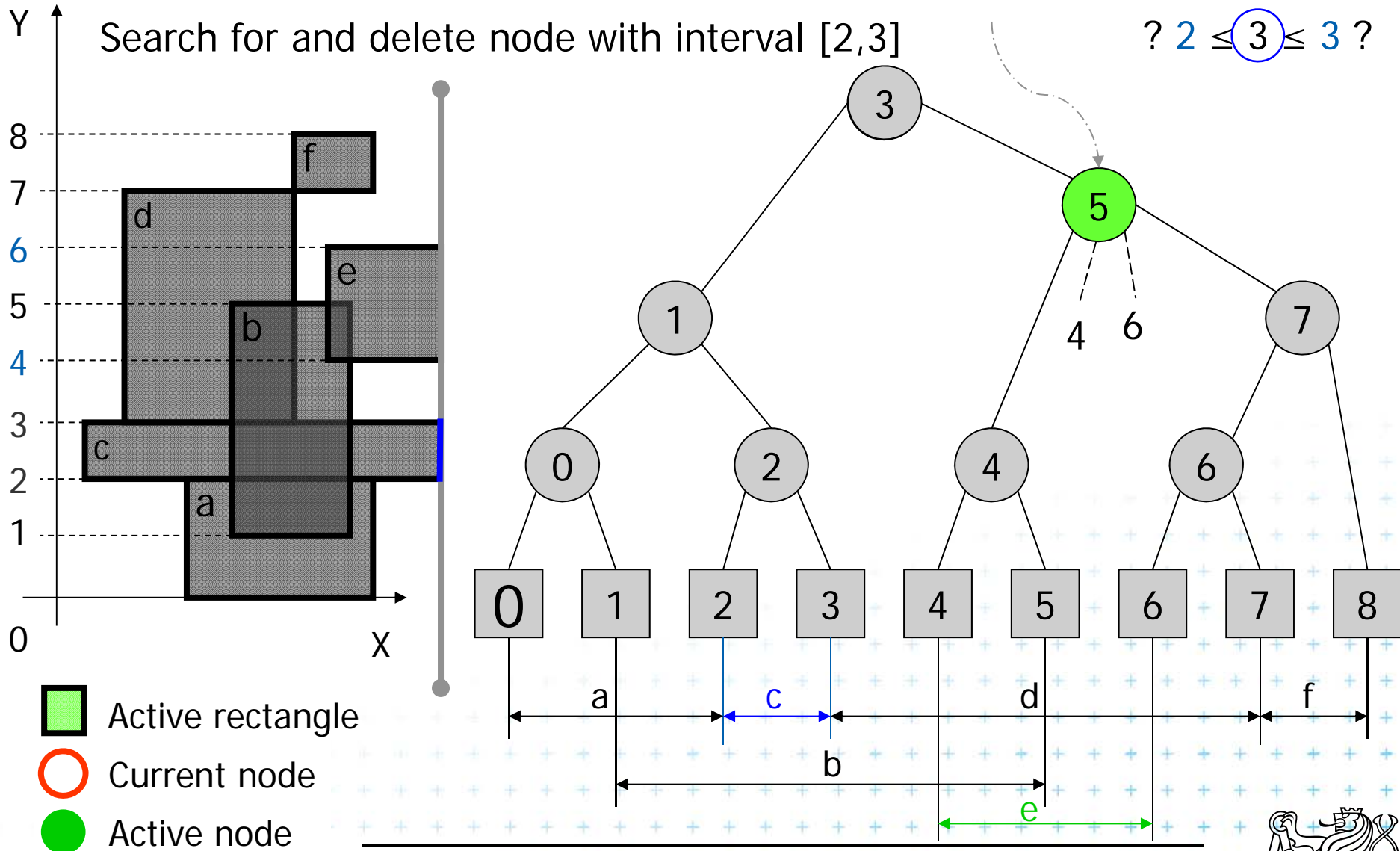
Search for and delete node with interval [2,3]






# Delete [2,3] Delete Interval

$$b \leq H(v) \leq e$$

$$? 2 \leq 3 \leq 3 ?$$

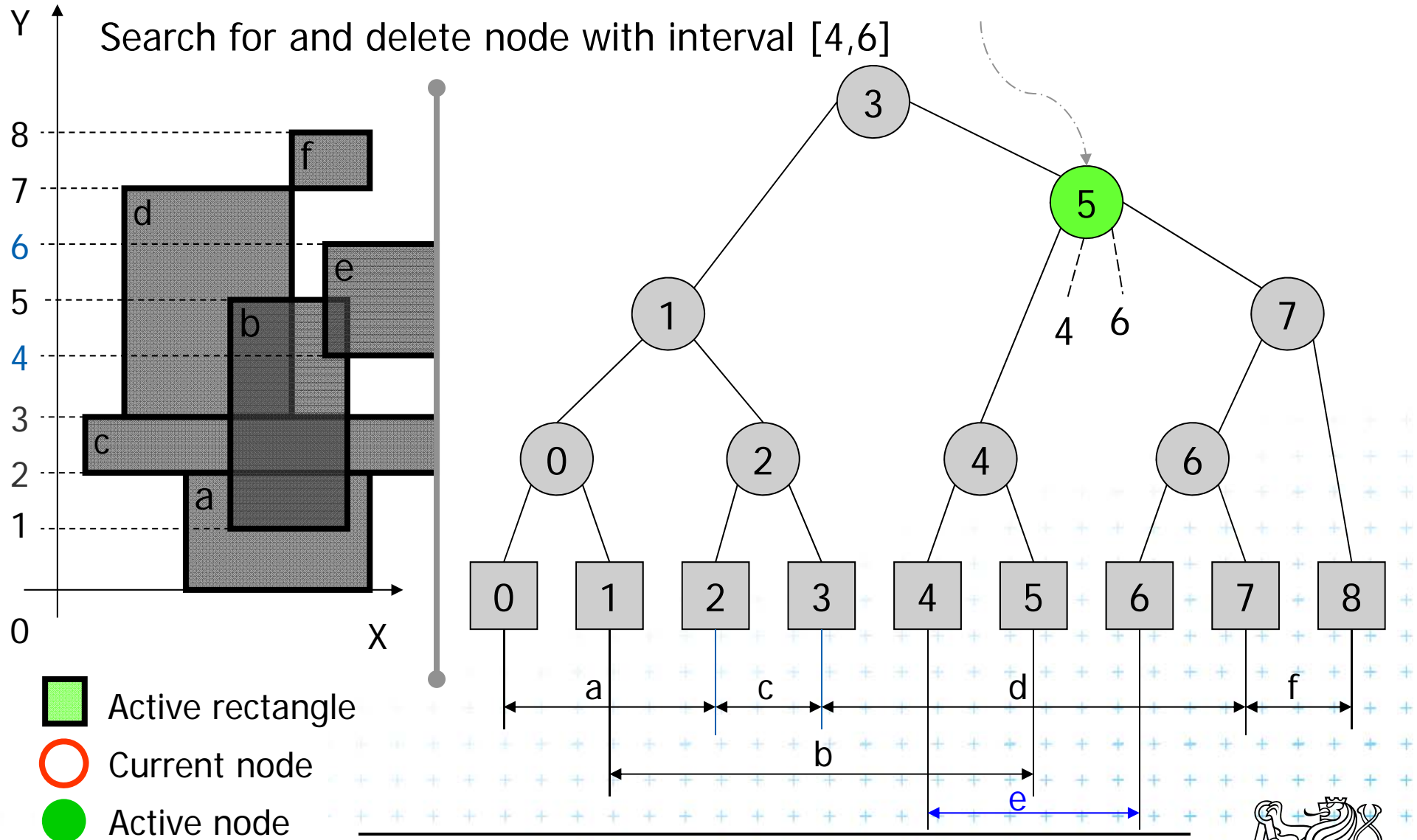


-  Active rectangle
-  Current node
-  Active node



# Delete [4,6] Delete Interval

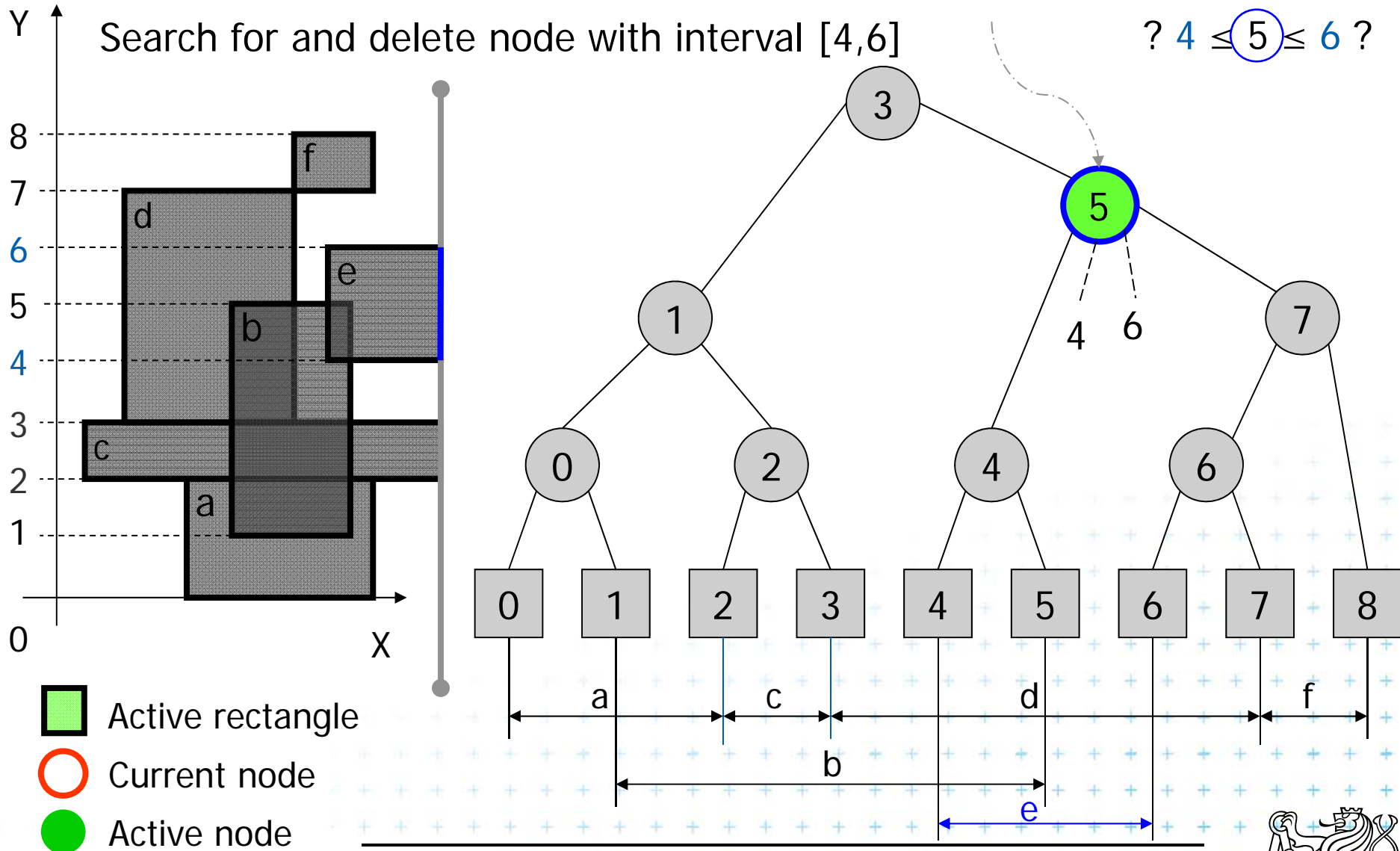
$$b \leq H(v) \leq e$$






# Delete [4,6] Delete Interval

$$b \leq H(v) \leq e$$

$$? 4 \leq 5 \leq 6 ?$$



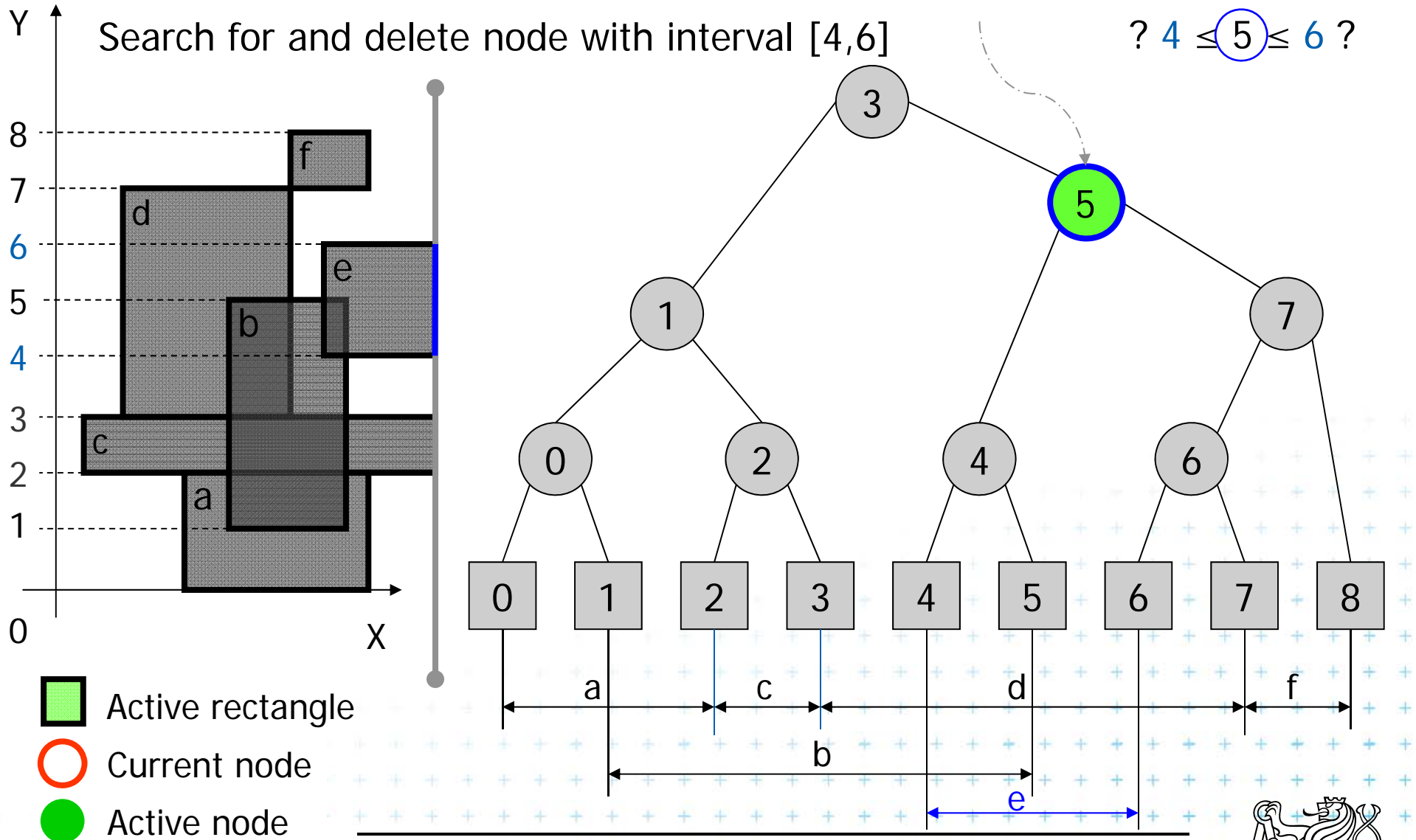
-  Active rectangle
-  Current node
-  Active node



# Delete [4,6] Delete Interval

$$b \leq H(v) \leq e$$

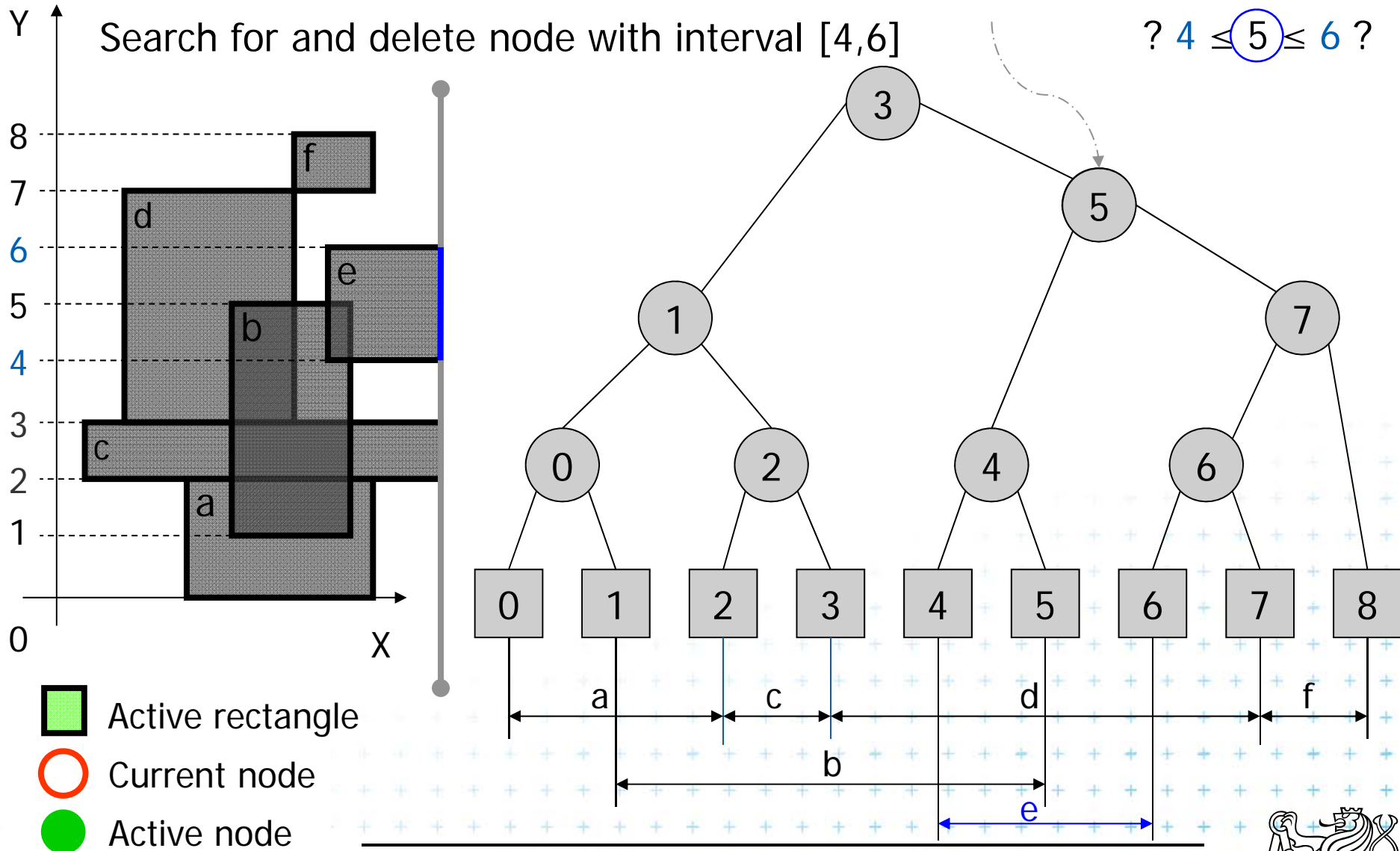
$$? 4 \leq 5 \leq 6 ?$$



# Delete [4,6] Delete Interval

$$b \leq H(v) \leq e$$

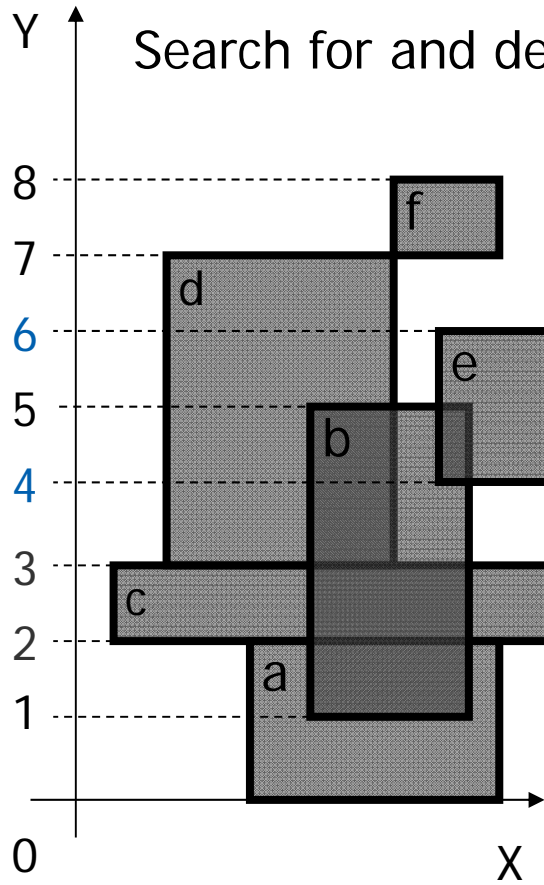
$$? 4 \leq 5 \leq 6 ?$$



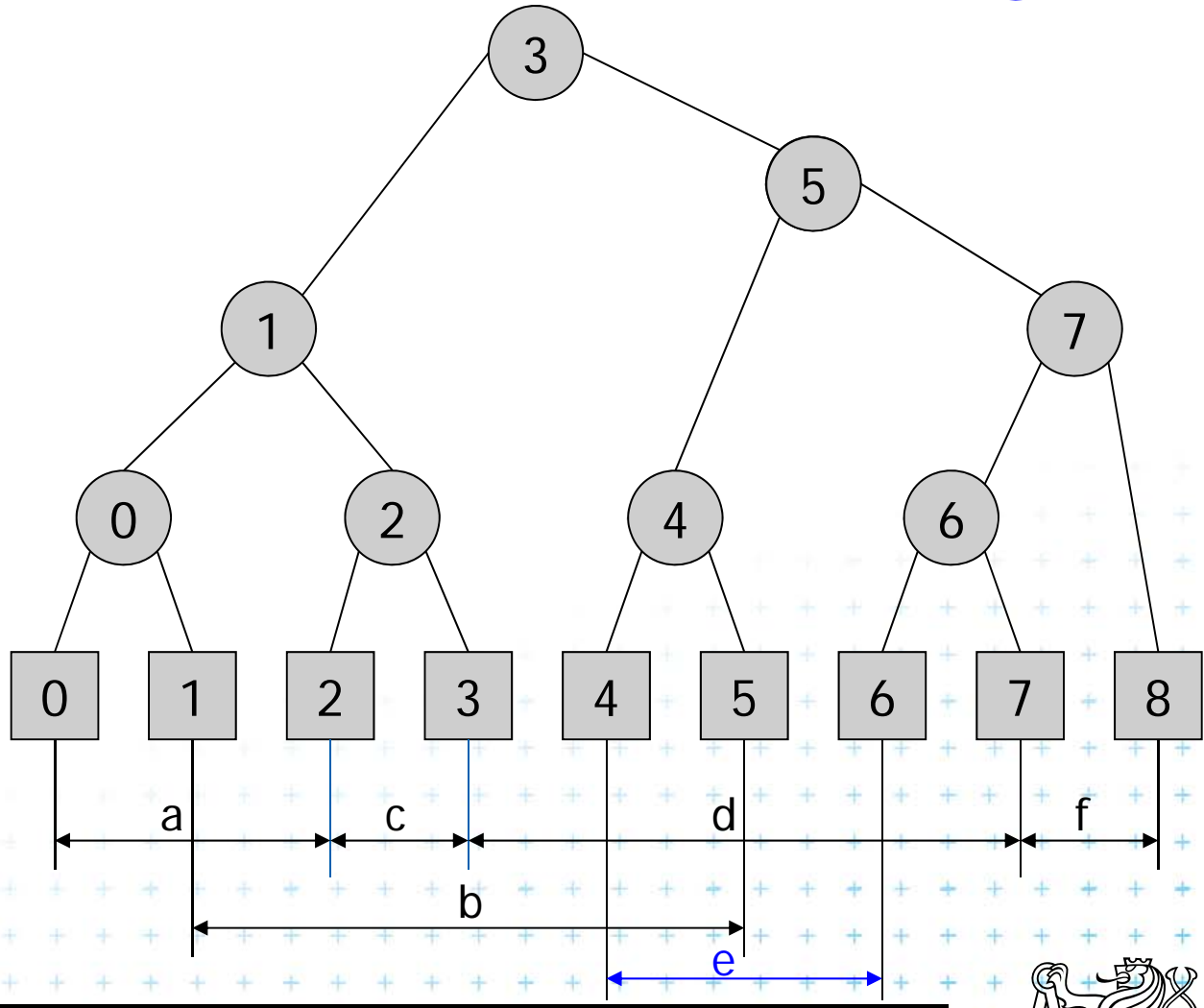
# Delete [4,6] Delete Interval

$$b \leq H(v) \leq e$$

$$? 4 \leq 5 \leq 6 ?$$

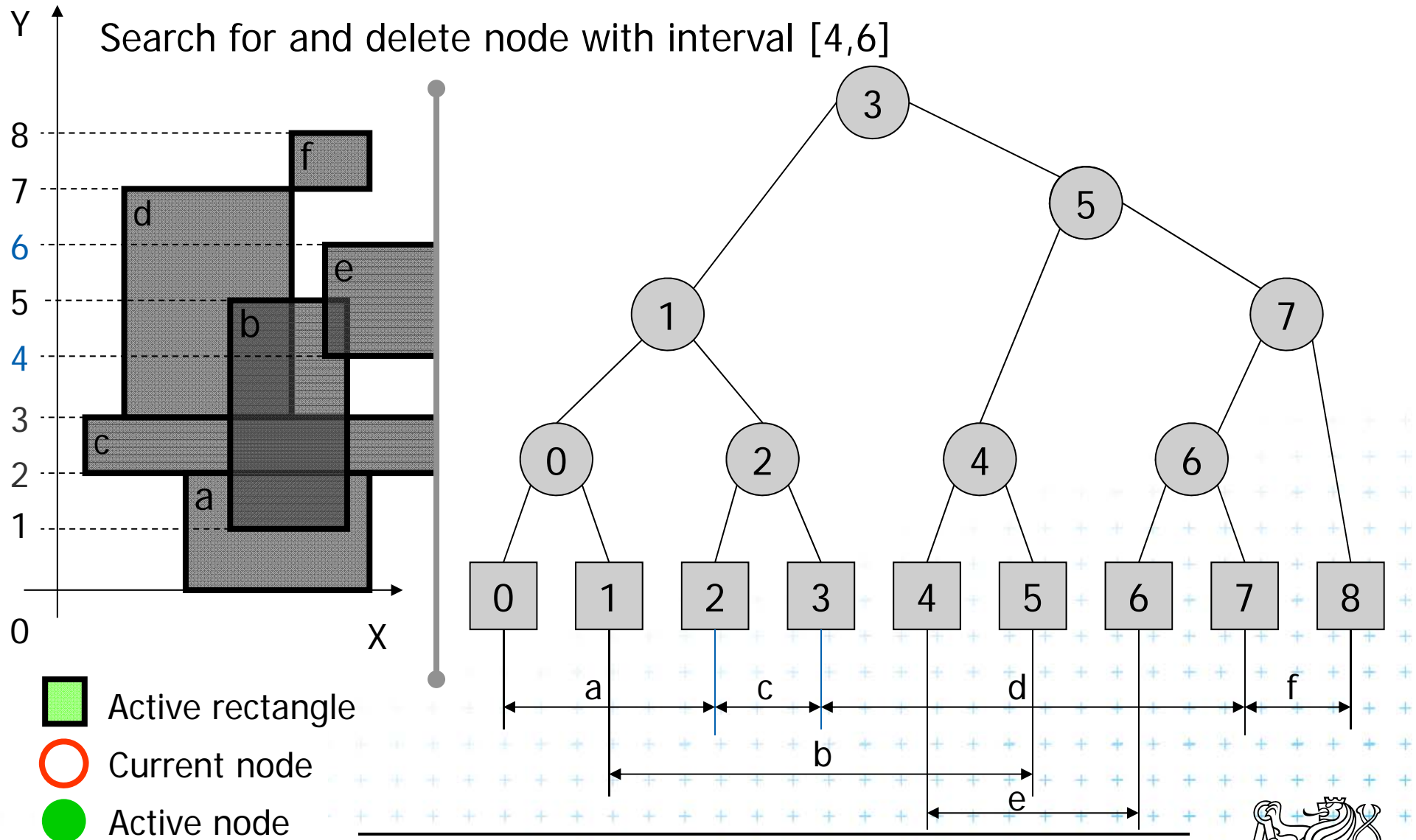


Search for and delete node with interval [4,6]





# Empty tree



# Complexities of rectangle intersections

---

- $n$  rectangles,  $s$  intersected pairs found
- $O(n \log n)$  preprocessing time to separately sort
  - x-coordinates of the rectangles for the plane sweep
  - the y-coordinates for initializing the interval tree.
- The plane sweep itself takes  $O(n \log n + s)$  time, so the overall time is  $O(n \log n + s)$
- $O(n)$  space
- This time is optimal for a decision-tree algorithm (i.e., one that only makes comparisons between rectangle coordinates).



# References

---

- [Berg] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars: **Computational Geometry: Algorithms and Applications**, Springer-Verlag, 3rd rev. ed. 2008. 386 pages, 370 fig. ISBN: 978-3-540-77973-5, Chapters 3 and 9, <http://www.cs.uu.nl/geobook/>
- [Mount] David Mount, - **CMSC 754: Computational Geometry**, Lecture Notes for Spring 2007, University of Maryland, Lectures 7,22, 13,14, and 30.  
<http://www.cs.umd.edu/class/spring2007/cmsc754/lectures.shtml>
- [Rourke] Joseph O'Rourke: **Computational Geometry in C**, Cambridge University Press, 1993, ISBN 0-521- 44592-2  
<http://maven.smith.edu/~orourke/books/compgeom.html>
- [Drtina] Tomáš Drtina: **Intersection of rectangles. Semestral Assignment. Computational Geometry course, FEL CTU Prague, 2006**
- [Kukral] Petr Kukrál: **Intersection of rectangles. Semestral Assignment. Computational Geometry course, FEL CTU Prague, 2006**
- [Vigneron] **Segment trees and interval trees, presentation, INRA, France,**  
<http://w3.jouy.inra.fr/unites/miaj/public/vigneron/cs4235/slides.html>

