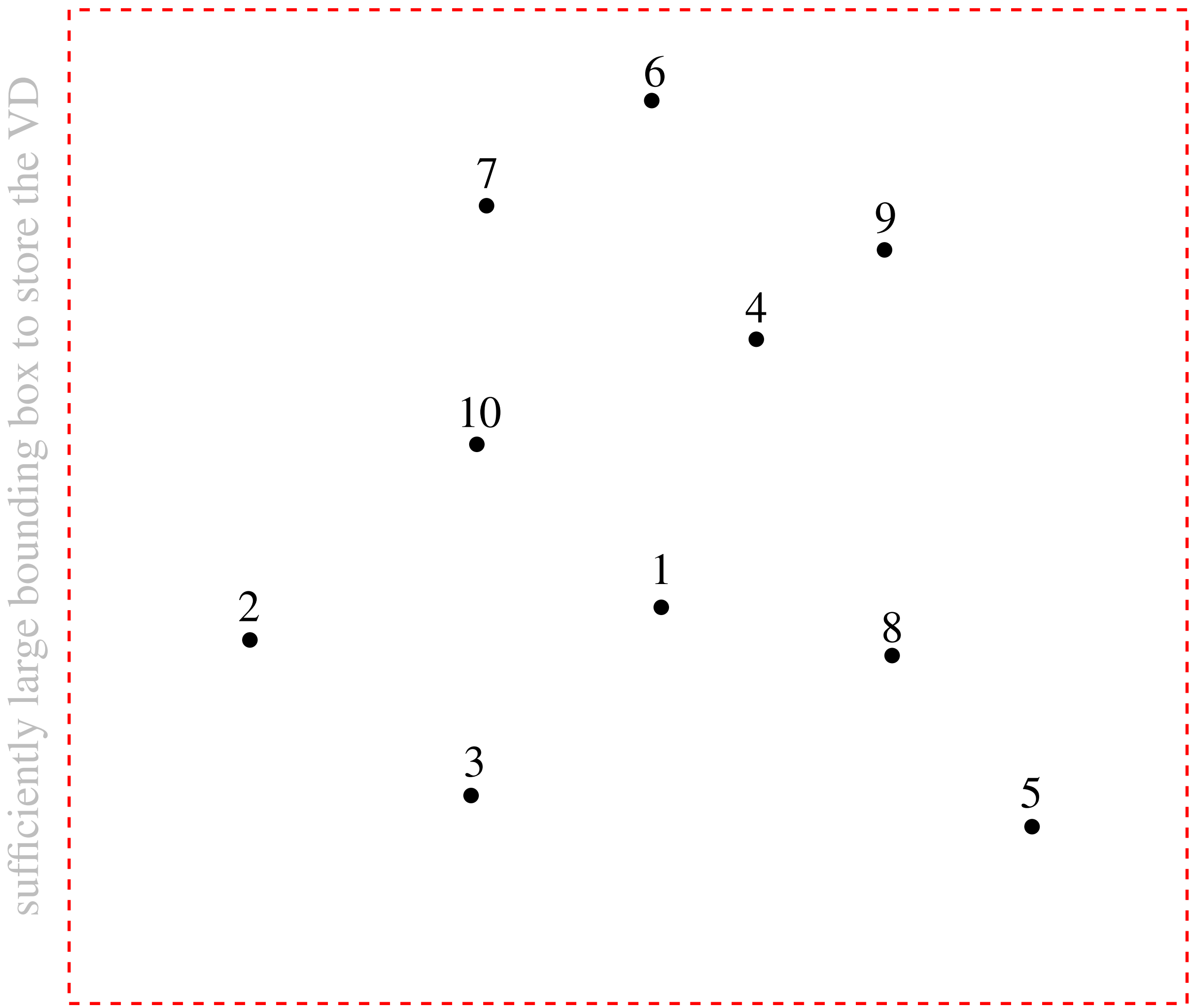
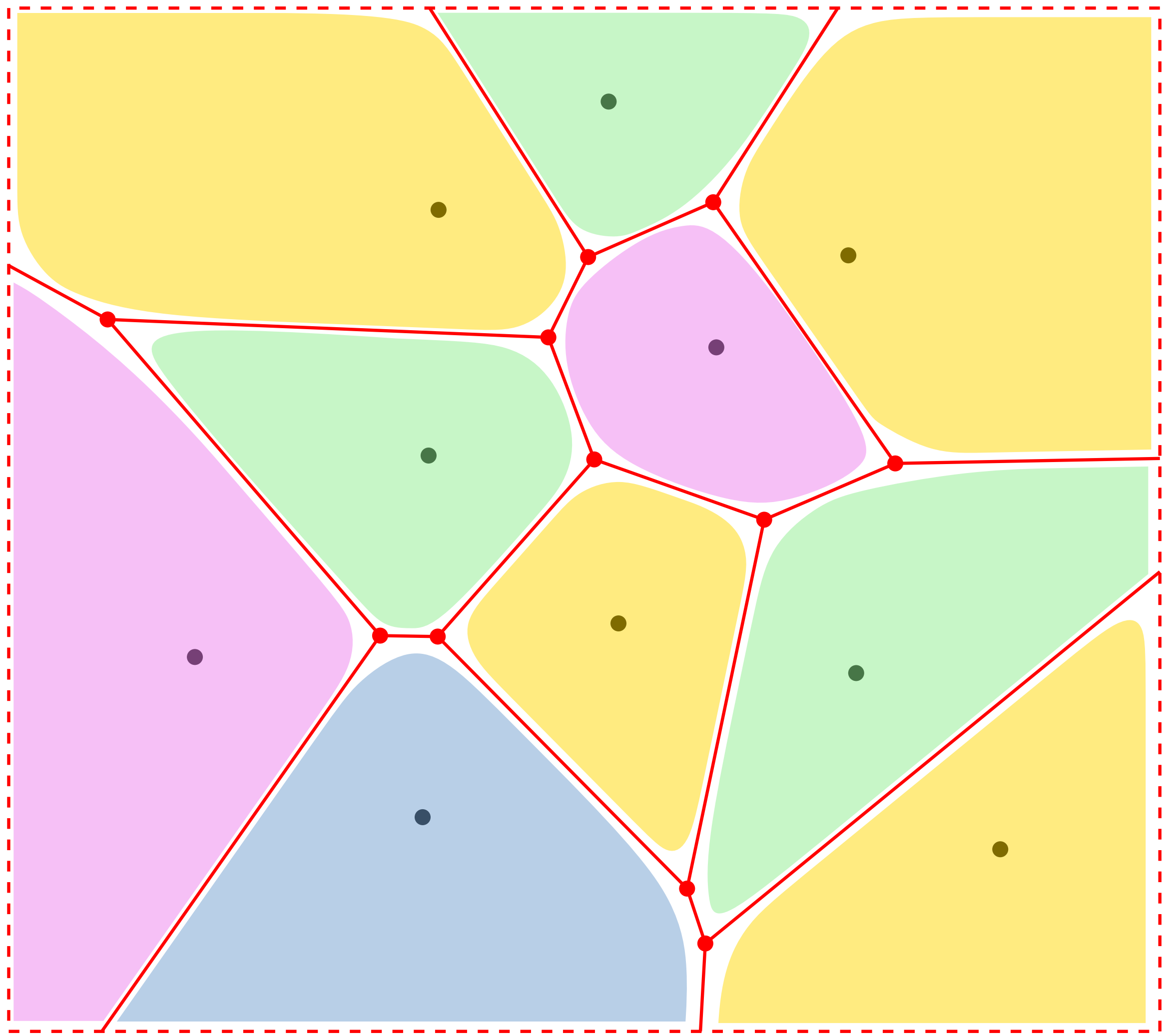
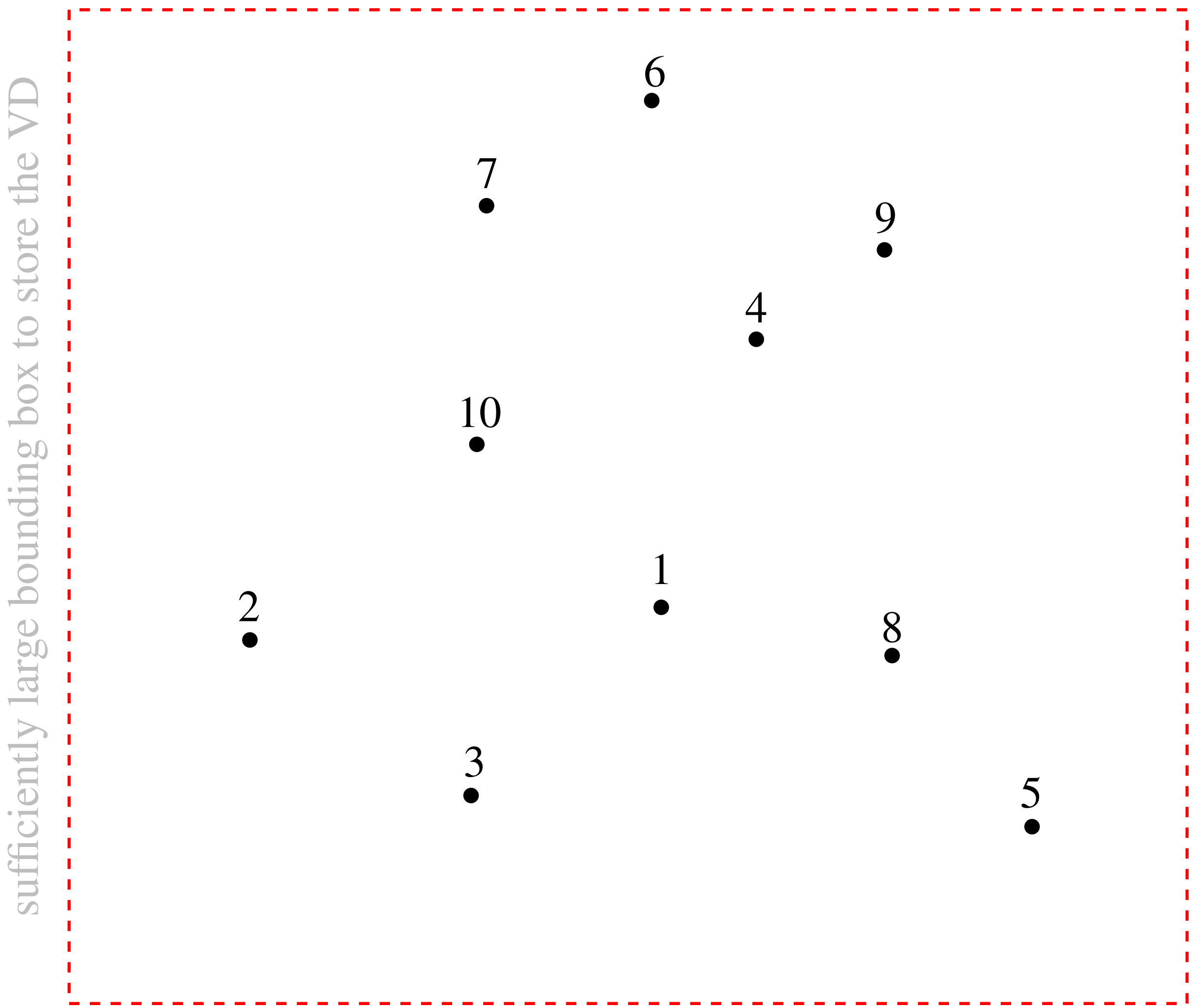

sufficiently large bounding box to store the VD
6
7
9
4
10
1
2
8
3
5

$P = 10$ sites



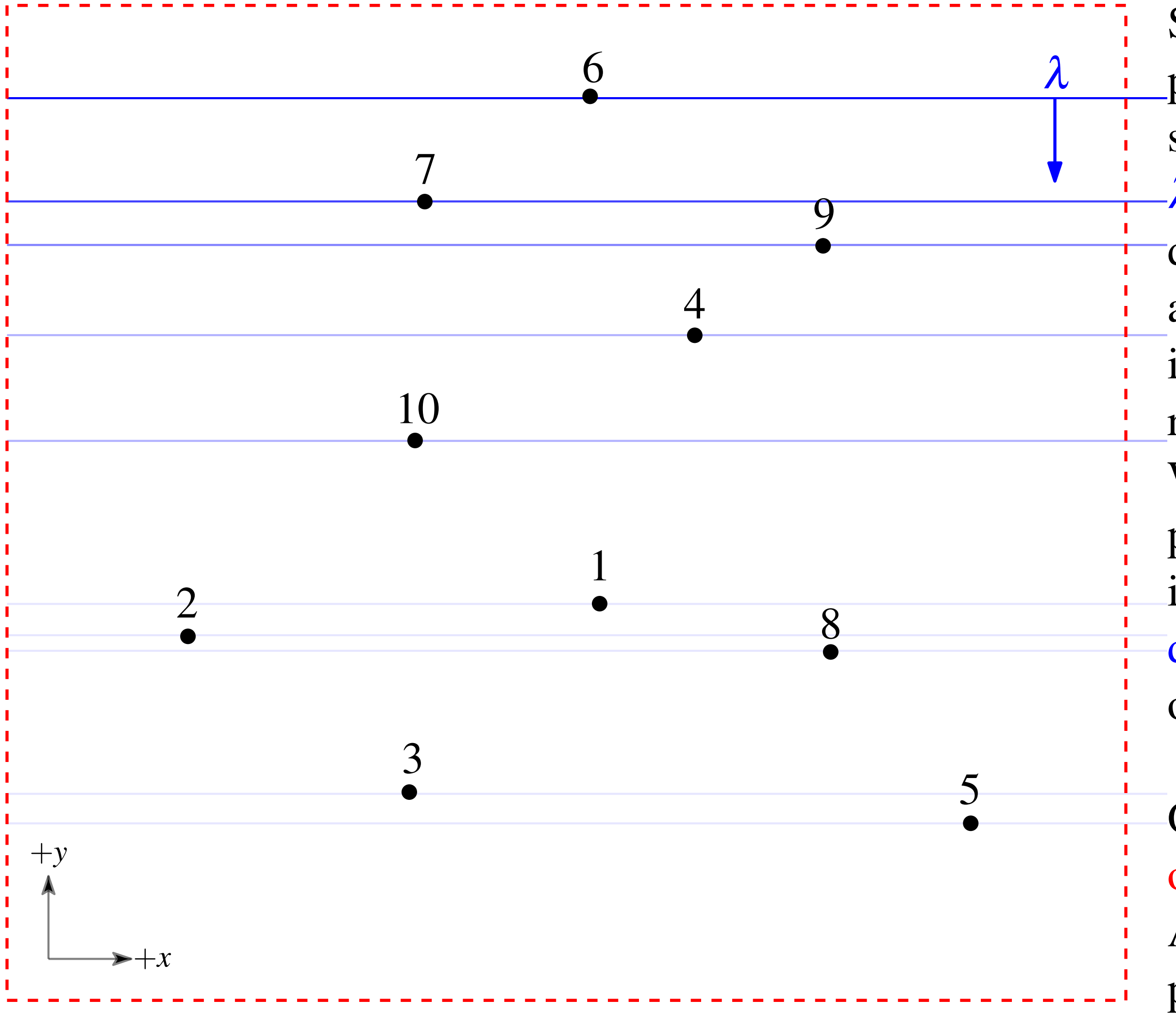
$\mathrm{VD}(P)$



sufficiently large bounding box to store the VD
6
7
9
4
10
1
2
8
3
5

$P = 10$ sites

The sweep line λ moves downward and halts only at the event points. At each event point some computation is done — will see next.



$P = 10$ sites

The sites, while given as input, may be arbitrarily ordered.

So, we need to store them in order, properly in a suitable data structure Q so that while moving the sweep line λ from top to bottom (say, along $-y$ direction), the sites are encountered and processed one by one (precessing of *site events*) . And accordingly related computation is done.

We shall also have to store more points (*circle events* — will see soon) in the same data structure, avoiding duplicate entry, in top-to-bottom order.

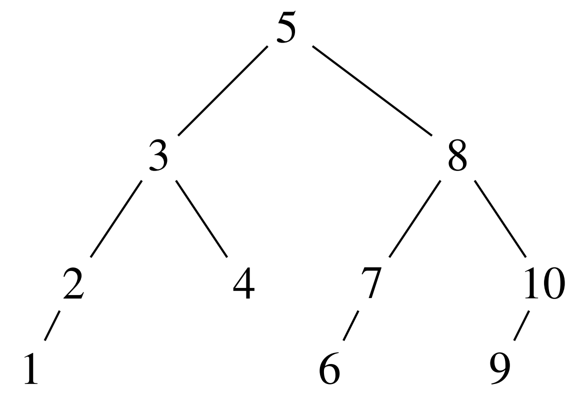
Q: Q should be efficient w.r.t. which operations?

A: insertion, deletion, stopping duplicate entry, ...

Q: Can you tell what will be this data structure?

A: Priority queue (see next slides)

It's not a heap but an AVL tree! Why?



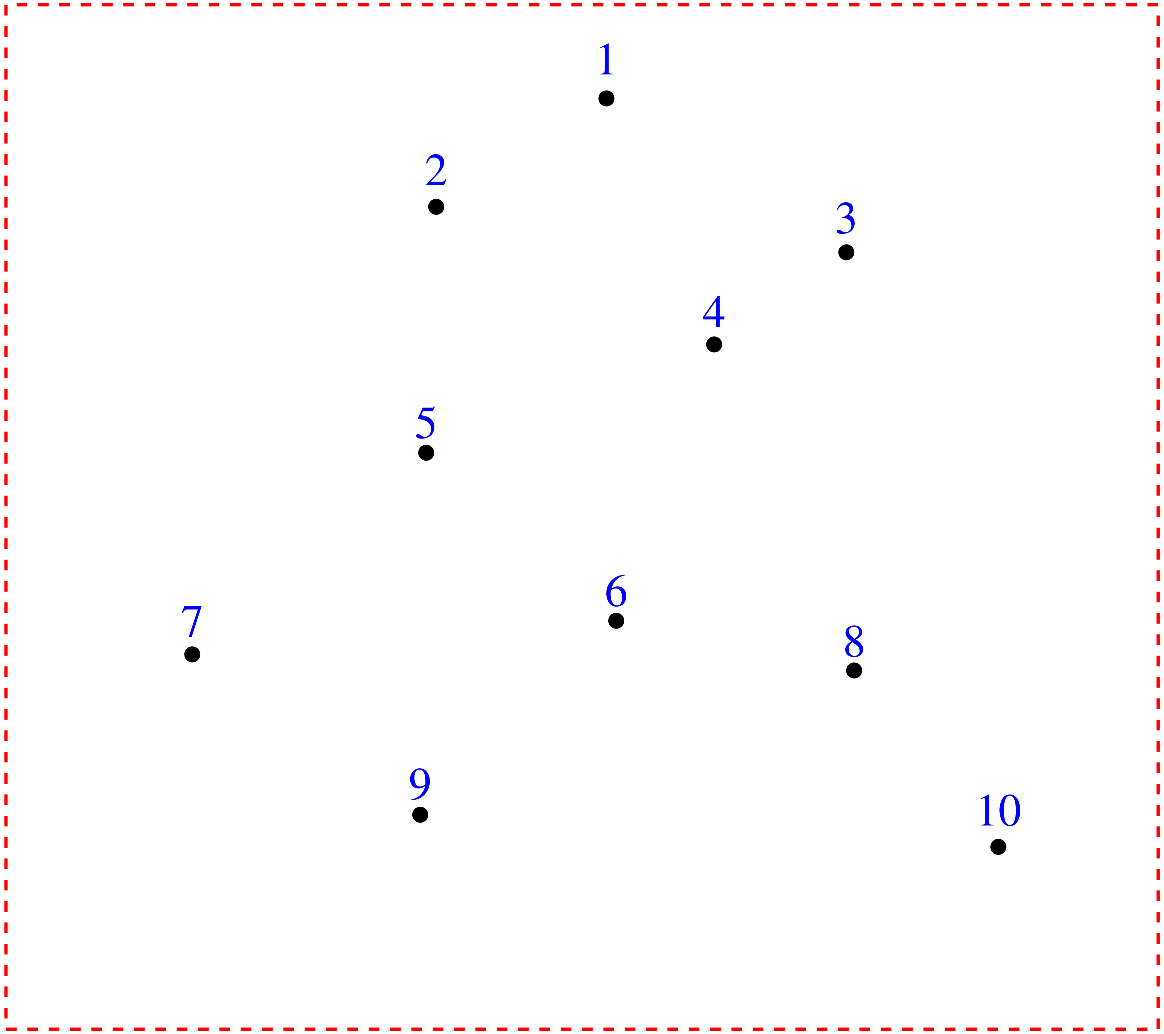
Event queue Q

(Its structure may vary)

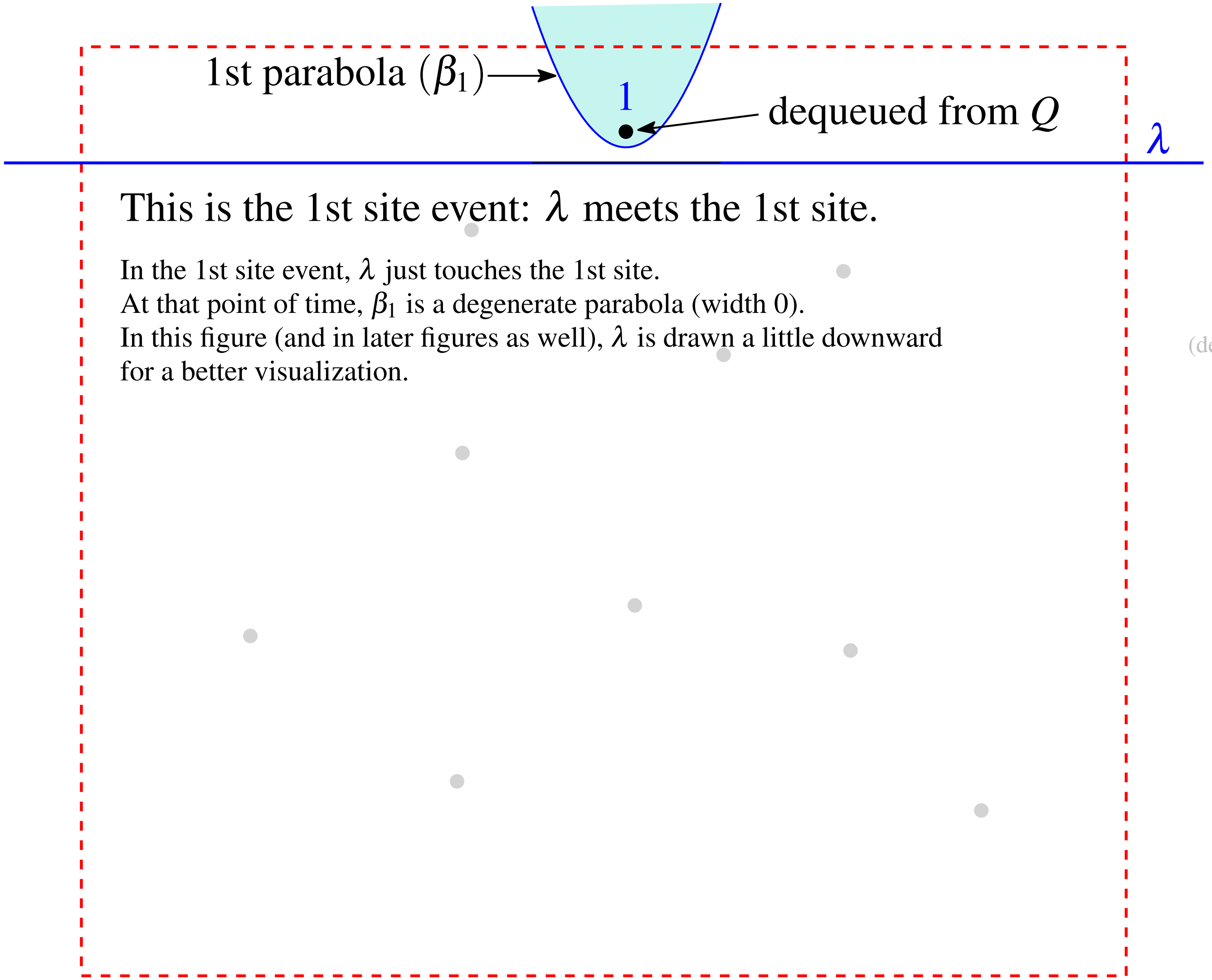
(You should work out the actual)

Each node stores the site ID. The site coordinates are in a linear array of points. You can think that the array index is the site ID.

Q will contain circle events when they occur (see later slides).

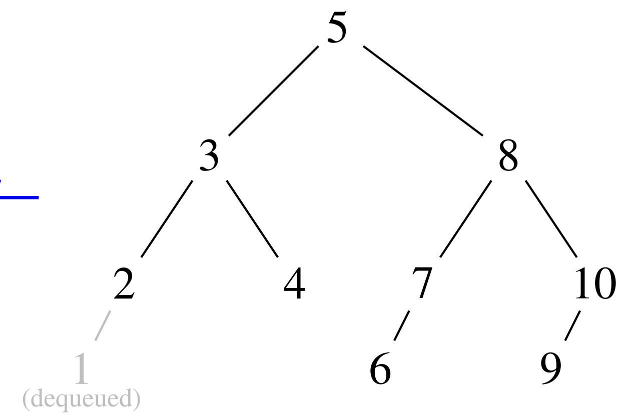


Event queue (priority queue) Q is initialized with P



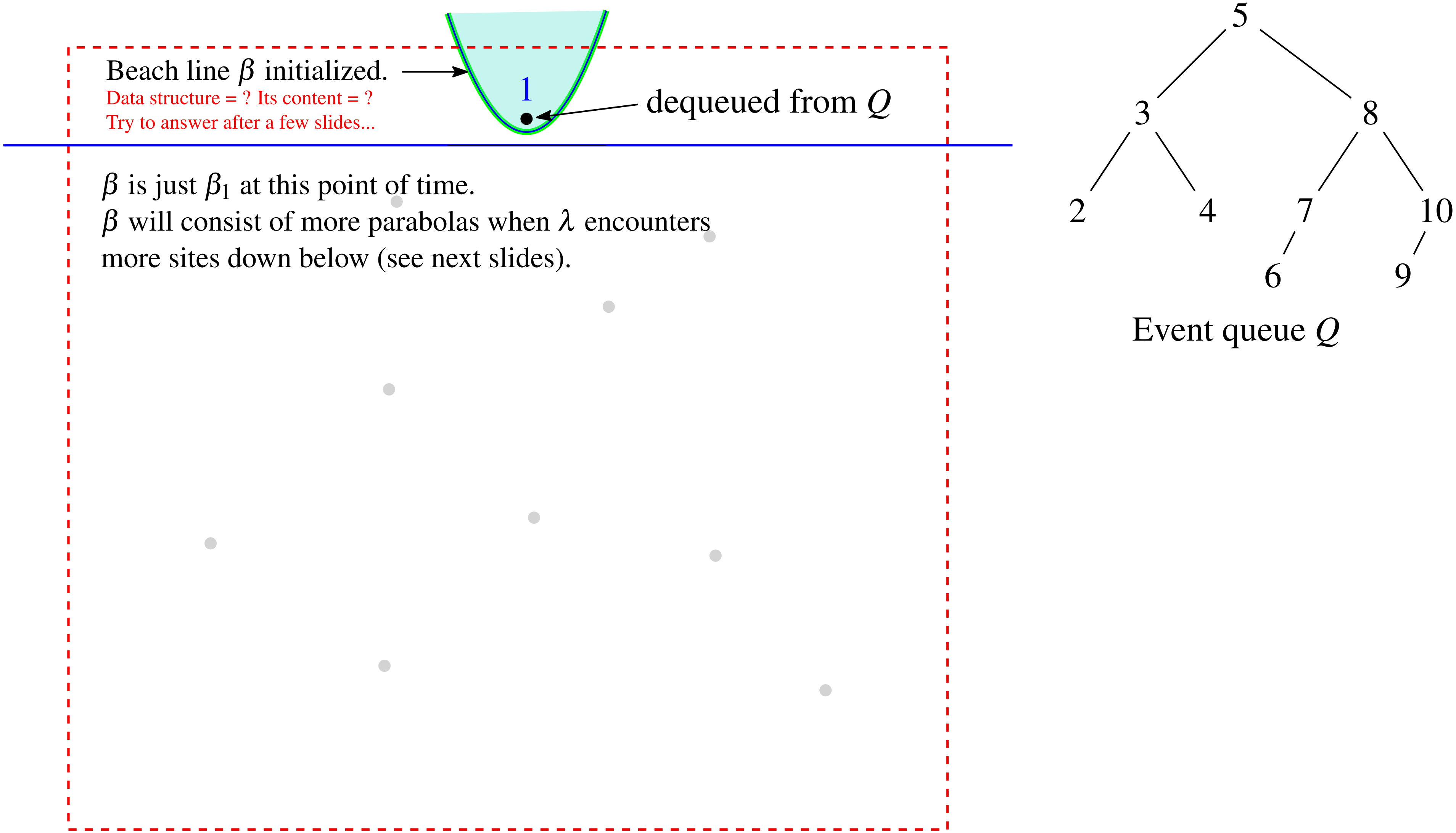
This is the 1st site event: λ meets the 1st site.

In the 1st site event, λ just touches the 1st site.
At that point of time, β_1 is a degenerate parabola (width 0).
In this figure (and in later figures as well), λ is drawn a little downward
for a better visualization.

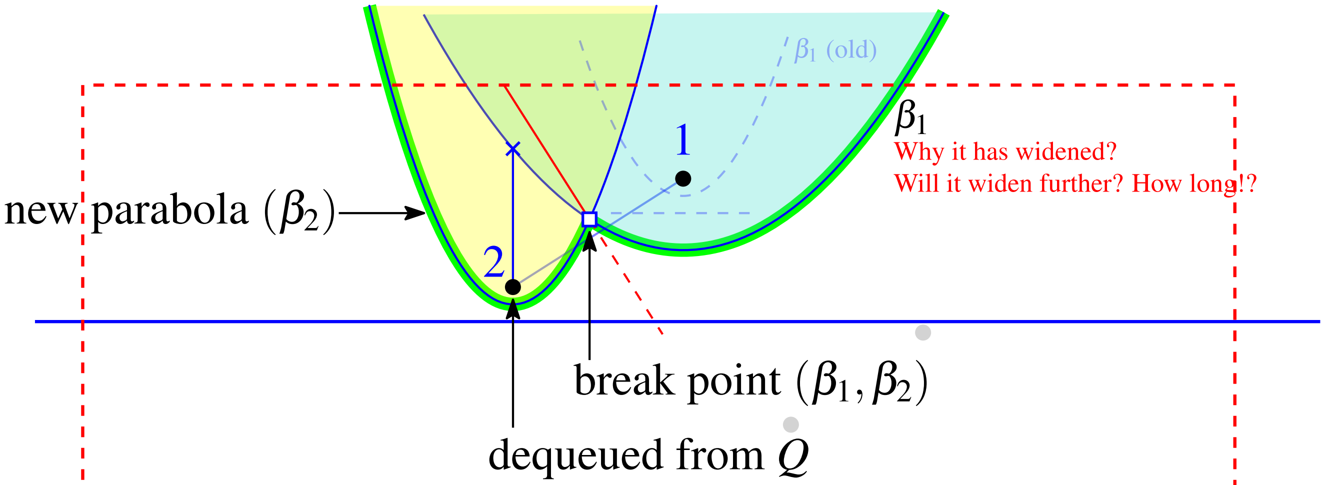


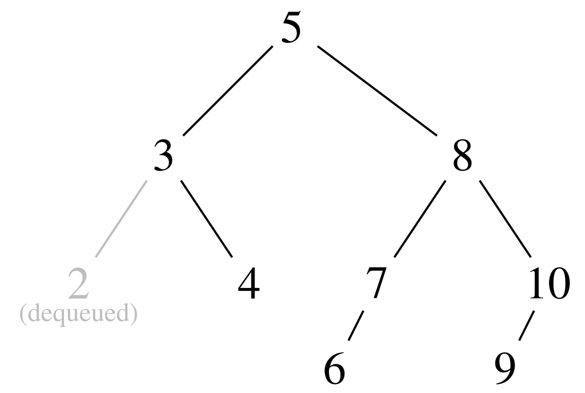
Event queue Q

(it's balanced after the dequeue opr.)



Beach line β initialized.
Data structure = ? Its content = ?
Try to answer after a few slides...
1
dequeued from Q
β is just β_1 at this point of time.
β will consist of more parabolas when λ encounters
more sites down below (see next slides).
5
3
8
2
4
7
10
6
9
Event queue Q





Event queue Q

It's still balanced after the 2nd dequeue opr.

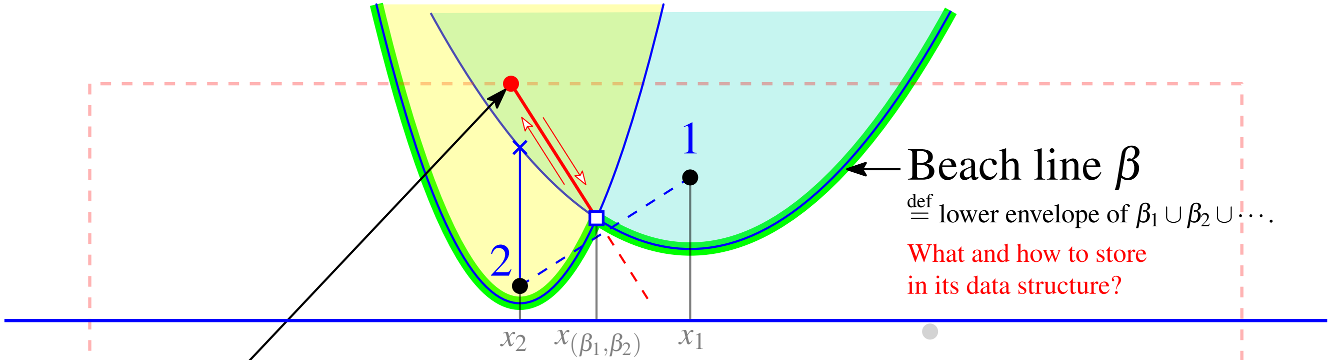
How to check it's balanced?

Can be checked & re-balanced in $O(\log n)$ time.

(Find out why)

Q: How to know which one of the existing parabolas will be intersected by the new parabola (here, β_2)?

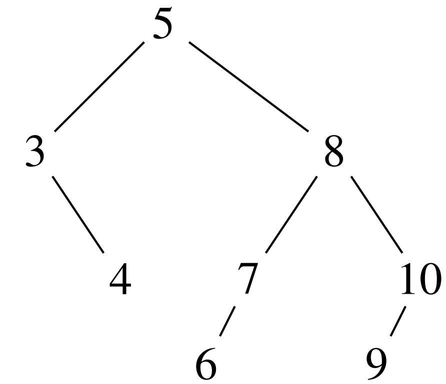
A: From the data structure B of beach line.
B will contain the break points and the foci of the parabolas in β, in left-to-right order. So, B is also a height-balanced BST.
For the new site, its x-coordinate acts as the search-key to find that parabola, using binary search in B (discussed later).



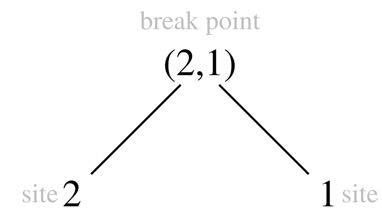
Vertex is inserted in DCEL.
Each incident half-edge is also inserted in DCEL.
The other endpoint, when found, is updated for the half-edge.
Its corresponding incident face is inserted in DCEL.
The half-edges on the boundary of the bounding box can be inserted now or later when the algorithm ends.

β is x-monotone.

That is, if you once start traversing from the leftmost end of β, you sometimes move up and sometimes down, but you never move left.
That means, there will be exactly one point of intersection between a vertical line and β.

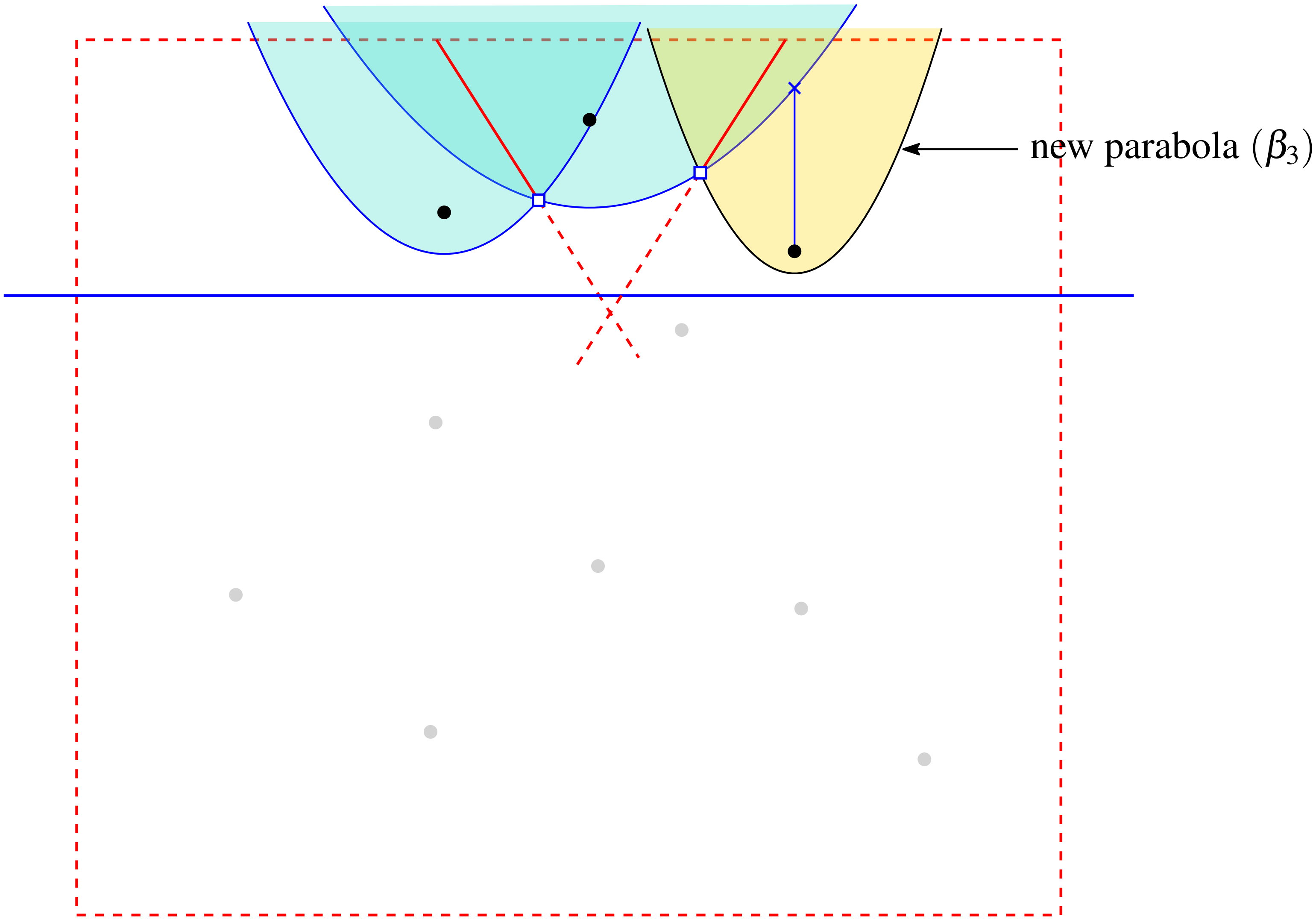


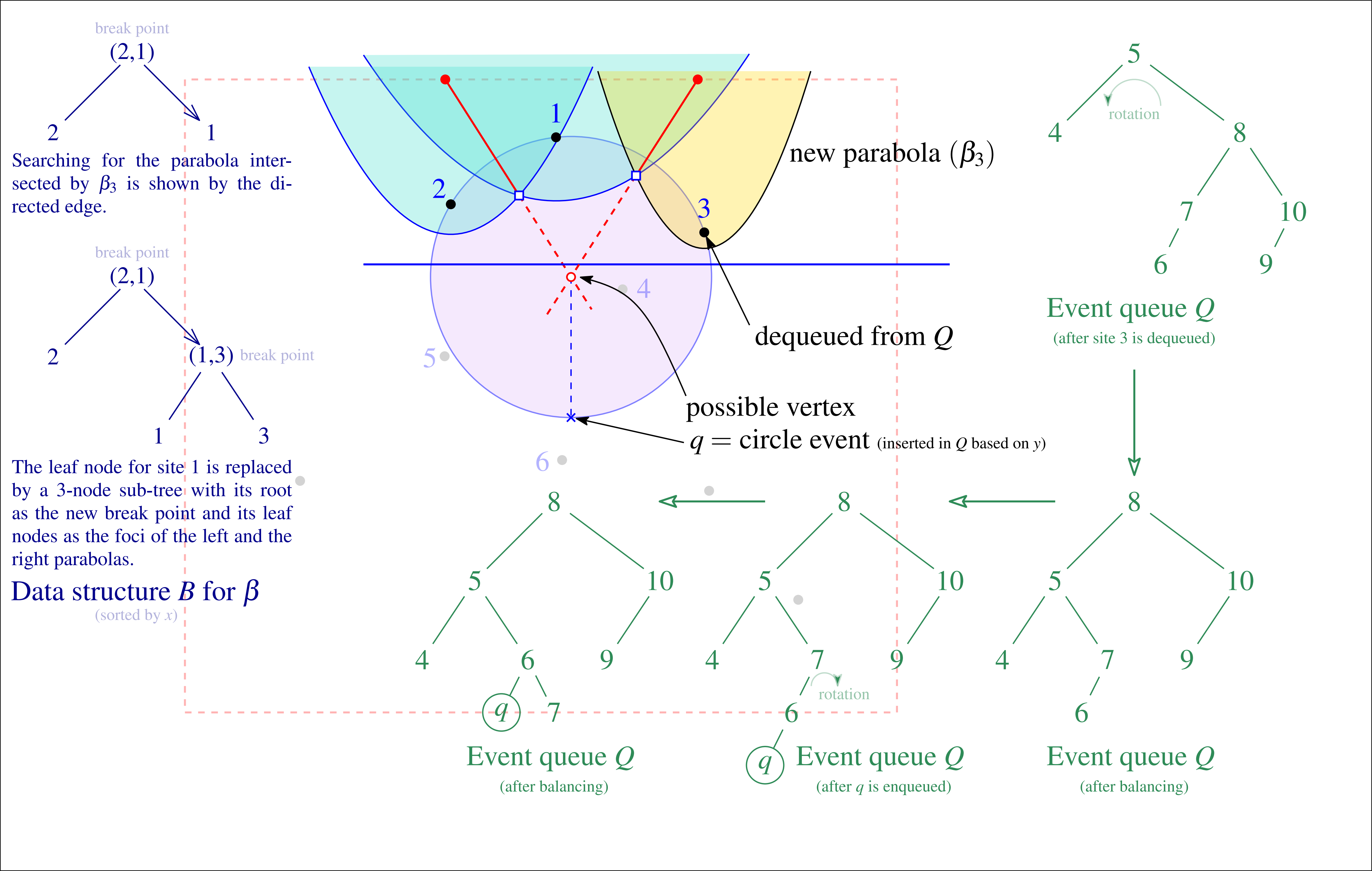
Event queue Q



Data structure B for β

B contains break points (as non-leaf nodes) and foci (i.e., site ID's, as leaf nodes) of parabolas in β, in left-to-right order.
So, in the inorder traversal of B, the x-coordinates of the sites and the break points stored in B will be in increasing order. For example, here $x_2 < x_{(\beta_1,\beta_2)} < x_1$.
For the current position of λ, you know its y-coordinate, and so you can compute everything related with β—that's the trick :-)

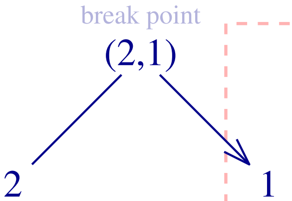

new parabola (β_3)



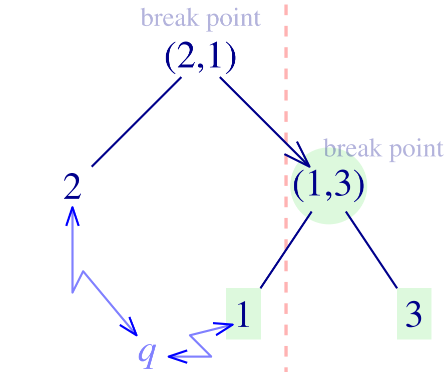
break point
(2,1)
2
1
Searching for the parabola intersected by β3 is shown by the directed edge.
break point
(2,1)
2
(1,3) break point
1
3
The leaf node for site 1 is replaced by a 3-node sub-tree with its root as the new break point and its leaf nodes as the foci of the left and the right parabolas.
Data structure B for β
(sorted by x)
1
2
3
4
5
6
new parabola (β3)
dequeued from Q
possible vertex
q = circle event (inserted in Q based on y)
5
rotation
4
8
7
10
6
9
Event queue Q
(after site 3 is dequeued)
8
5
10
4
6
9
q
7
Event queue Q
(after balancing)
8
5
10
4
7
9
rotation
6
q
Event queue Q
(after q is enqueued)
8
5
10
4
7
9
6
Event queue Q
(after balancing)

Data structure B for β

(sorted by x)

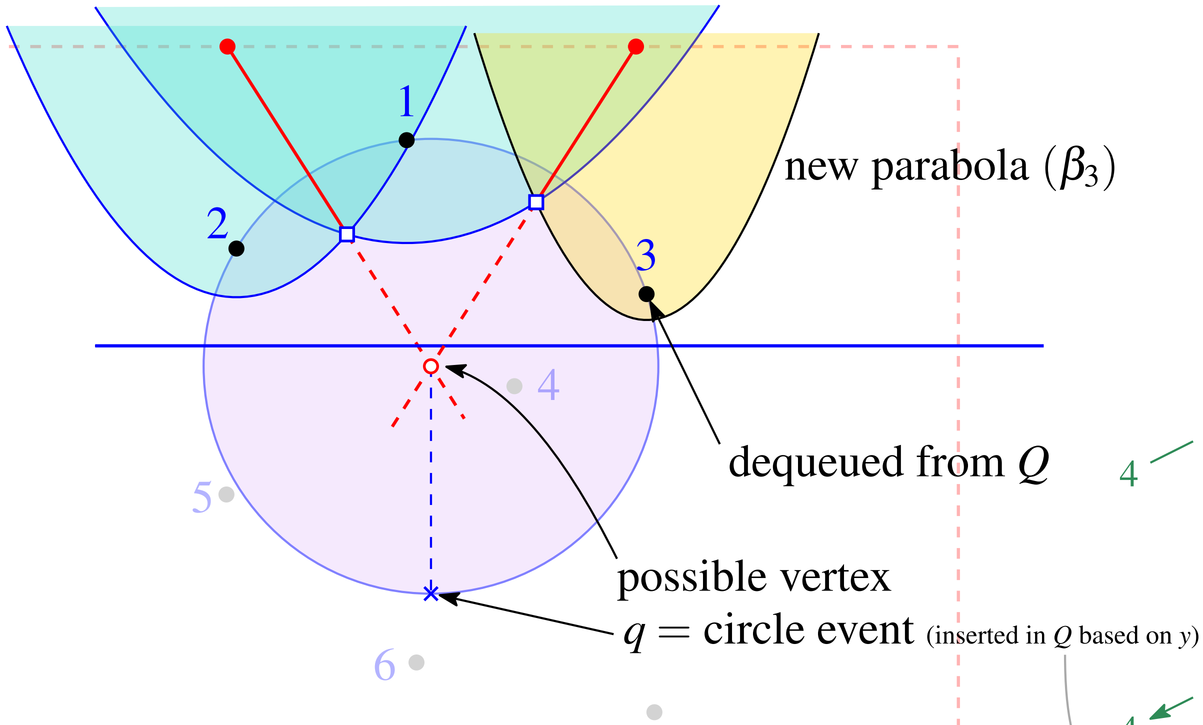


Searching for the parabola(s) intersected by β_3 is shown by the directed edge.



pointers to and from q

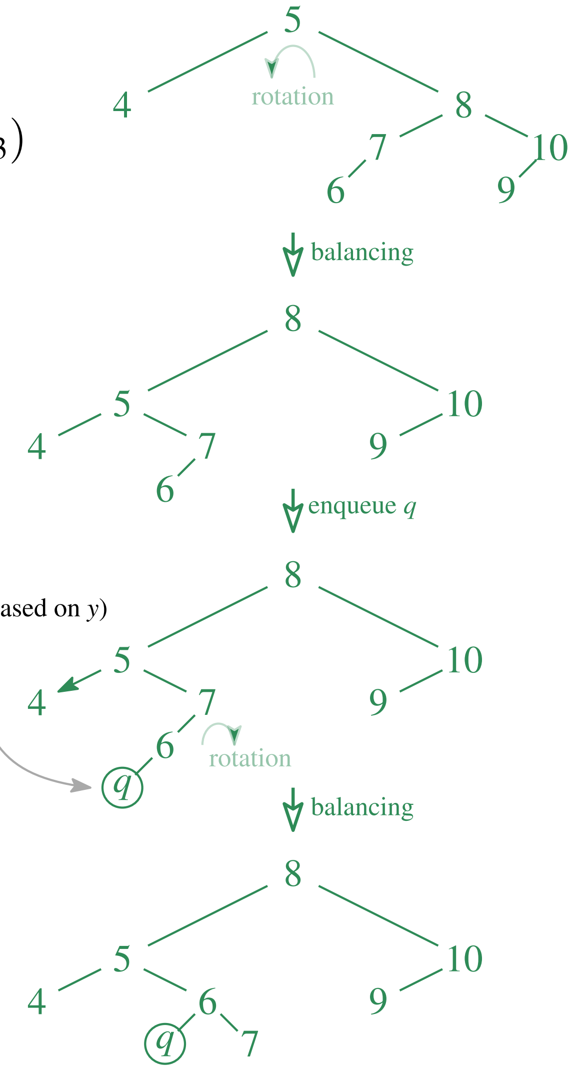
The leaf node for site 1 is replaced by a 3-node sub-tree with its root as the new break point and its leaf nodes as the foci of the left and the right parabolas.
Balancing is done if needed.



A *circle event* q is the lowest point of the circumcircle of three foci forming three consecutive parabolas in β.
If some site below λ is later found to lie in the interior of that circumcircle, then the circle event is a "false alarm".
How and when the false alarm is discovered?
[In this example, site 4 will throw "false alarm" when it is encountered by λ]

Event queue Q

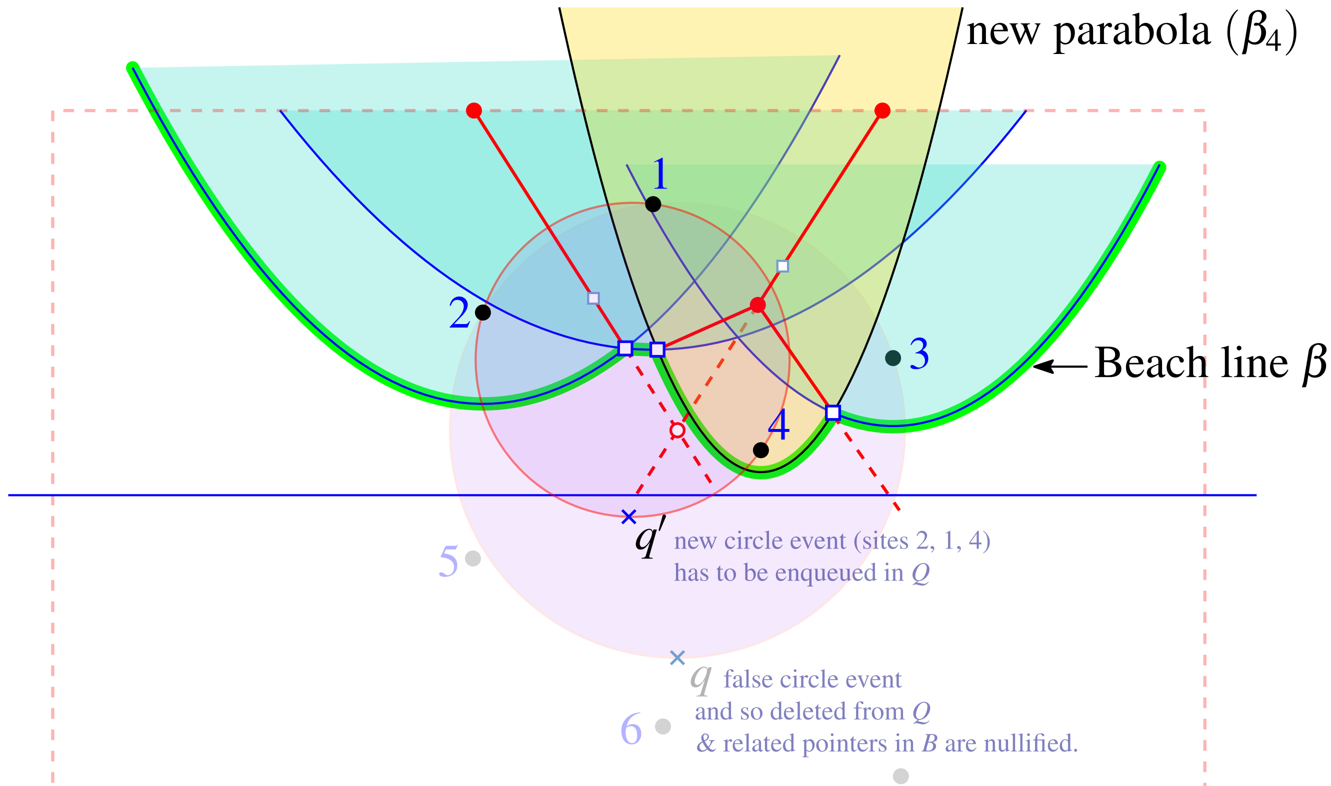
(after site 3 is dequeued)



Event queue Q

(after all operations)

As mentioned above, a circle event q is formed by three consecutive parabolas in β.
That means, it is formed by two perpendicular bisectors—one between the left and the middle parabolas, and the other between the middle and the right parabolas.
In other words, q is formed between two consecutive break points in β, and so they are two consecutive non-leaf nodes (think about inorder traversal) in B.



Two new break points — why?

Check for new circle event (created by two newly-consecutive break points).

What are the changes in B?

Updates in DCEL?