Lecture Notes for *Introduction to Computational Geometry* (Com S 418/518)
Yan-Bin Jia, Iowa State University

Interval Trees

Outline:

I. Windowing query

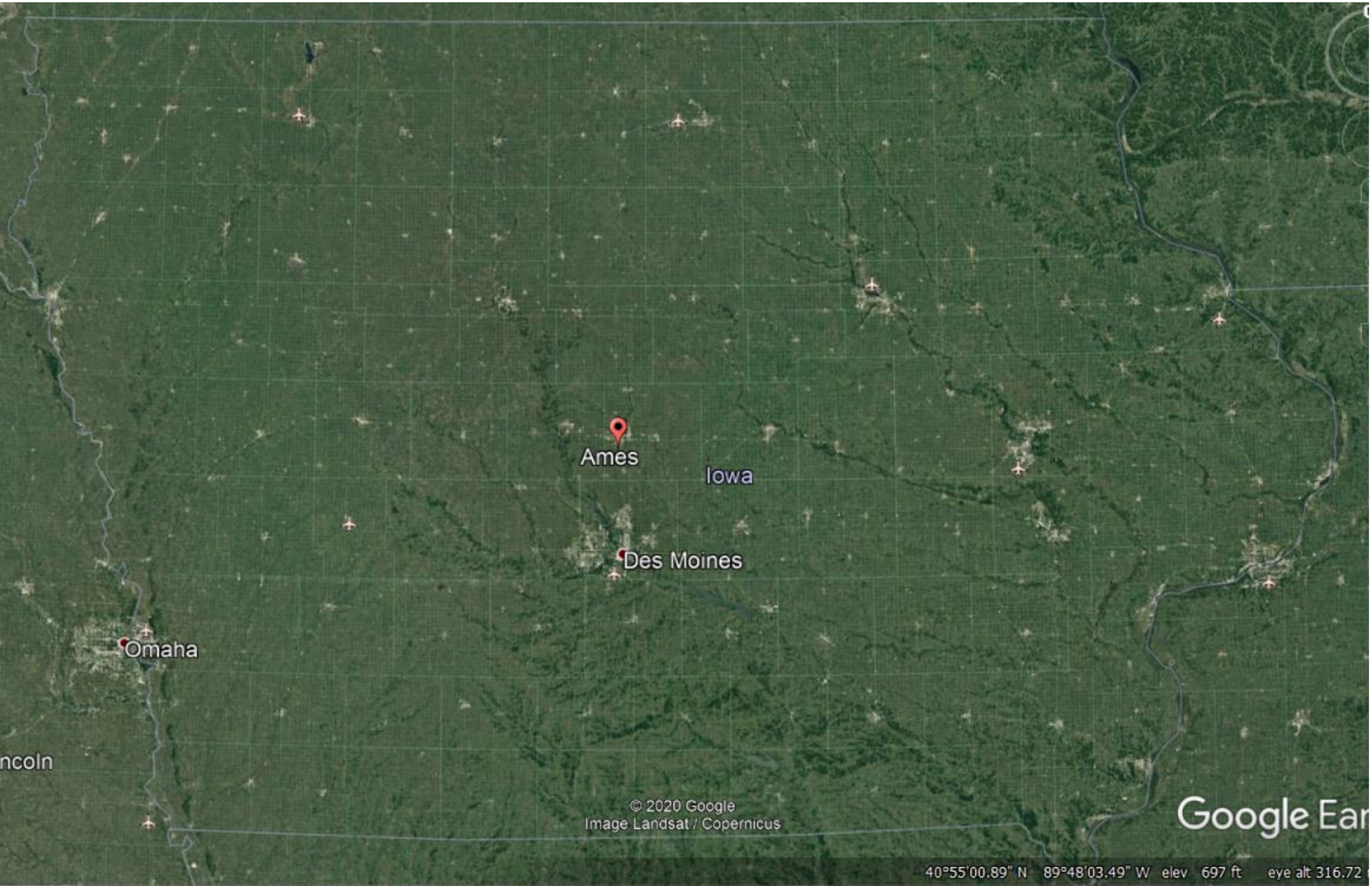
II. Interval tree

III. Construction and query

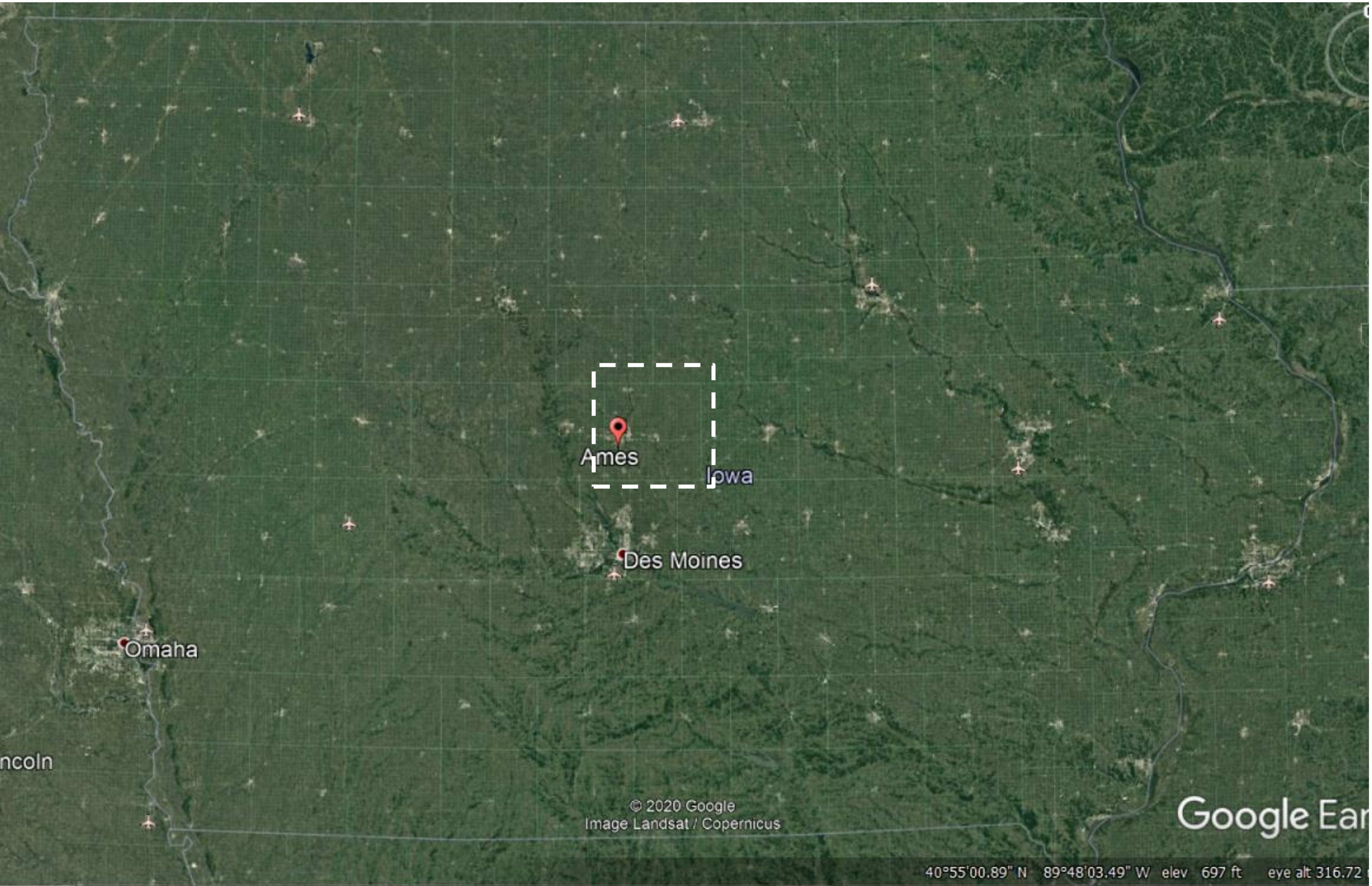
IV. Query with a line segment

Textbook: *Computational Geometry: Algorithms and Applications (*3rd ed.) by M. de Berg et al., Springer-Verlag, 2008.

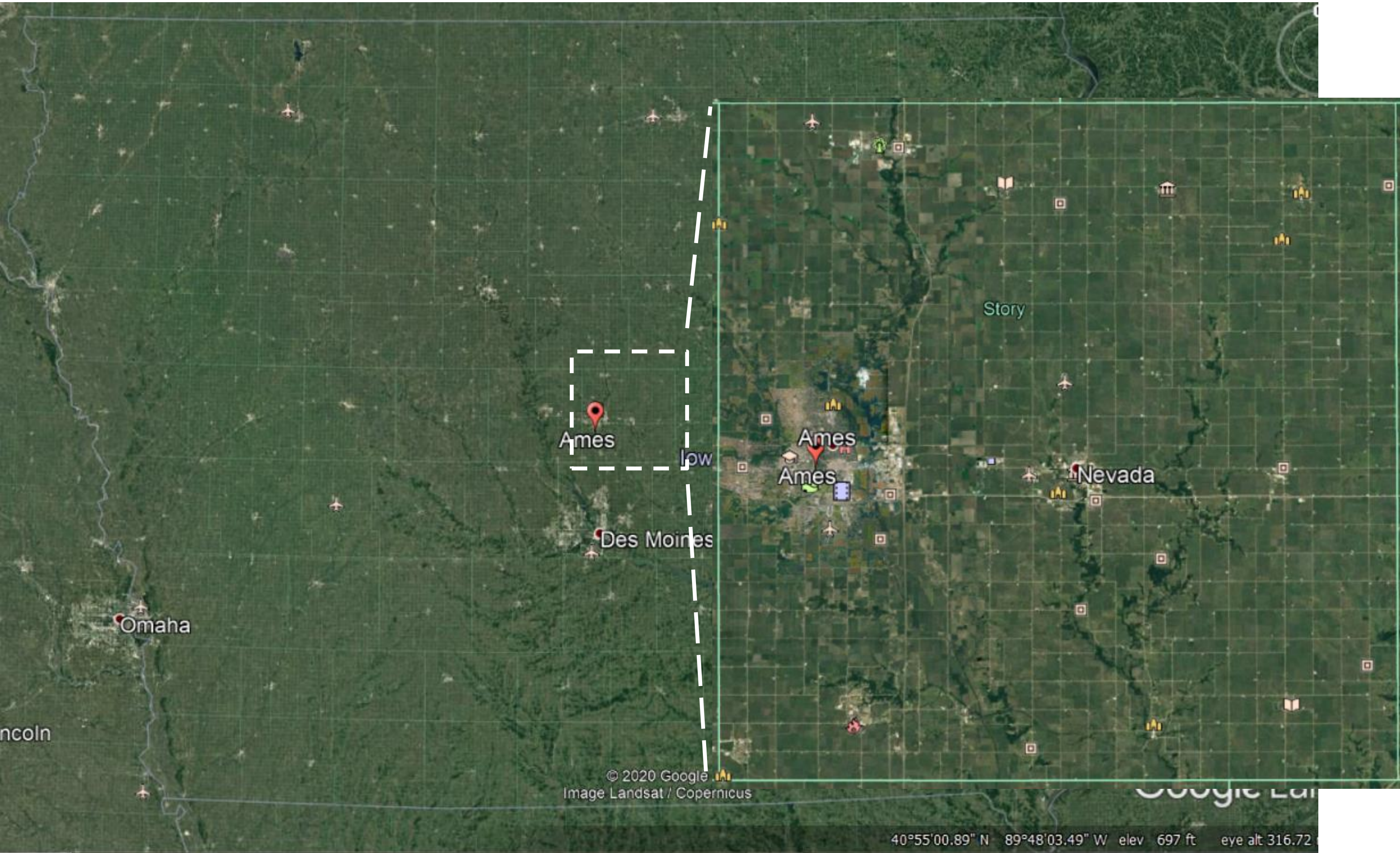
I. Windowing


Ames
Iowa
Des Moines
Omaha
ncoln
© 2020 Google
Image Landsat / Copernicus
Google Ear
40°55'00.89" N 89°48'03.49" W elev 697 ft eye alt 316.72

I. Windowing

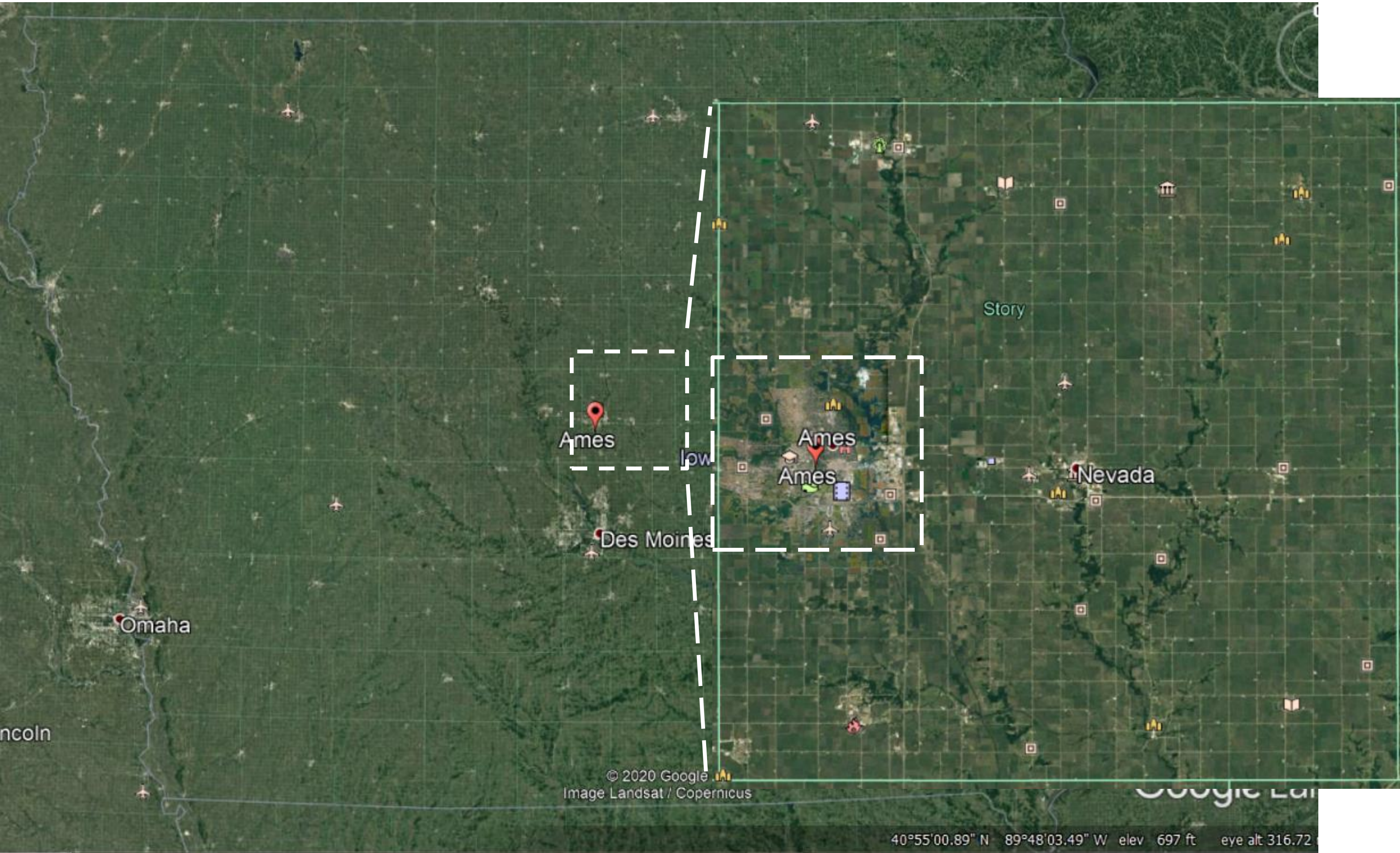

Ames
Iowa
Des Moines
Omaha
ncoln
© 2020 Google
Image Landsat / Copernicus
Google Ear
40°55'00.89" N 89°48'03.49" W elev 697 ft eye alt 316.72

I. Windowing



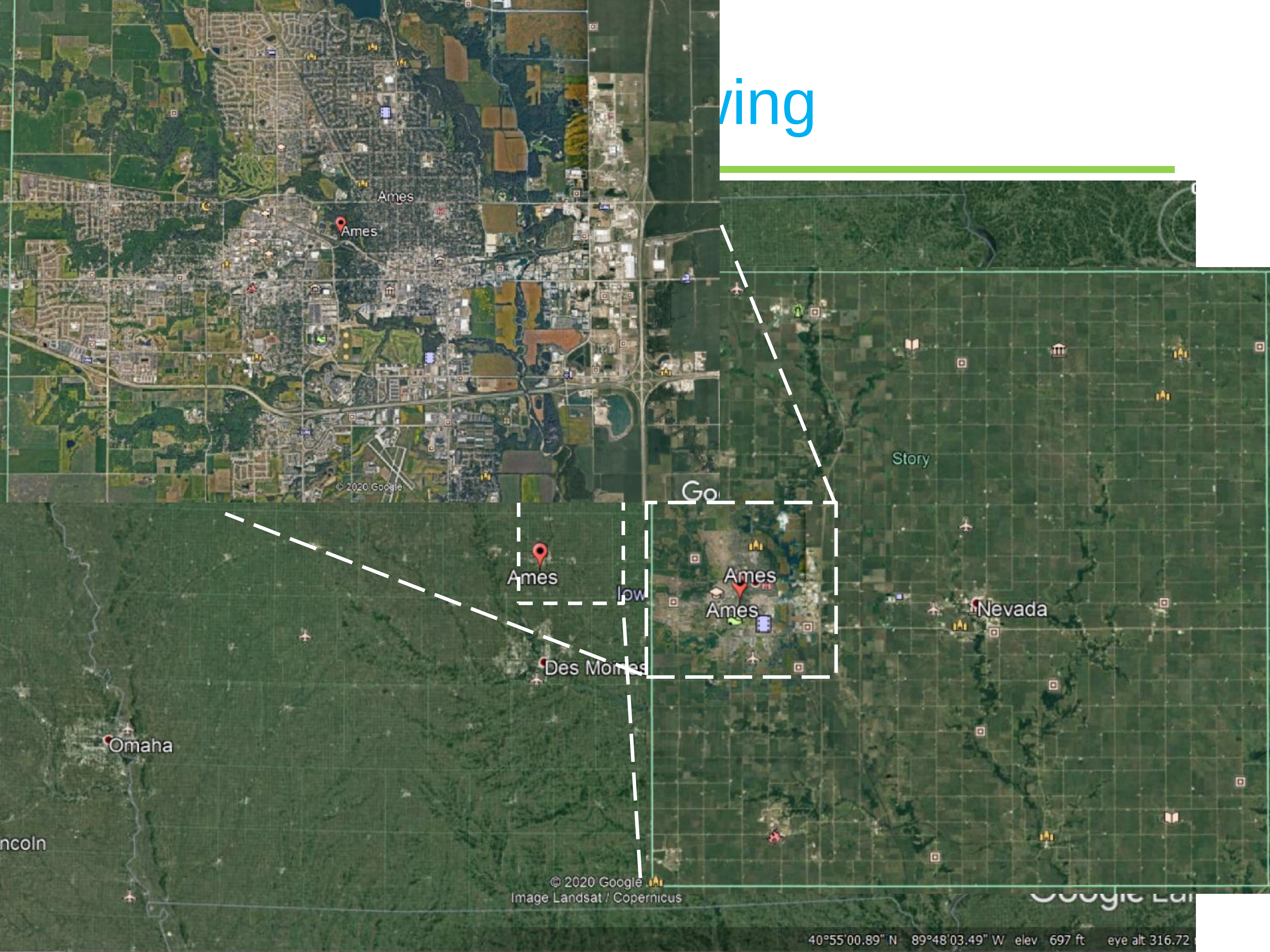
Ames
Des Moines
Omaha
ncoln
Iow
Story
Ames
Ames
Nevada
© 2020 Google
Image Landsat / Copernicus
40°55'00.89" N 89°48'03.49" W elev 697 ft eye alt 316.72

I. Windowing



Ames
Des Moines
Omaha
ncoln
Story
Ames
Ames
Nevada
© 2020 Google
Image Landsat / Copernicus
40°55'00.89" N 89°48'03.49" W elev 697 ft eye alt 316.72

ving


Ames
Ames
© 2020 Google
Ames
Story
Ames
Ames
Nevada
Omaha
© 2020 Google
Image Landsat / Copernicus
40°55'00.89" N 89°48'03.49" W elev 697 ft eye alt 316.72

Windowing Query

Determine the part of the map that lie in the window and display them.

Windowing Query

Determine the part of the map that lie in the window and display them.

Applications

- ♣ Geographic maps
- ♣ Computer graphics (e.g., flight simulation – part of landscape within sight)
- ♣ Design of circuit boards – zooming in onto a certain portion

Windowing Query

Determine the part of the map that lie in the window and display them.

Applications

- ♣ Geographic maps
- ♣ Computer graphics (e.g., flight simulation – part of landscape within sight)
- ♣ Design of circuit boards – zooming in onto a certain portion

Brute-force approach: Check every single feature to see if it is inside the window.

Windowing Query

Determine the part of the map that lie in the window and display them.

Applications

- ♣ Geographic maps
- ♣ Computer graphics (e.g., flight simulation – part of landscape within sight)
- ♣ Design of circuit boards – zooming in onto a certain portion

Brute-force approach: Check every single feature to see if it is inside the window.

Too slow because the amount of data is huge!

Windowing Query

Determine the part of the map that lie in the window and display them.

Applications

- ♣ Geographic maps
- ♣ Computer graphics (e.g., flight simulation – part of landscape within sight)
- ♣ Design of circuit boards – zooming in onto a certain portion

Brute-force approach: Check every single feature to see if it is inside the window.

Too slow because the amount of data is huge!

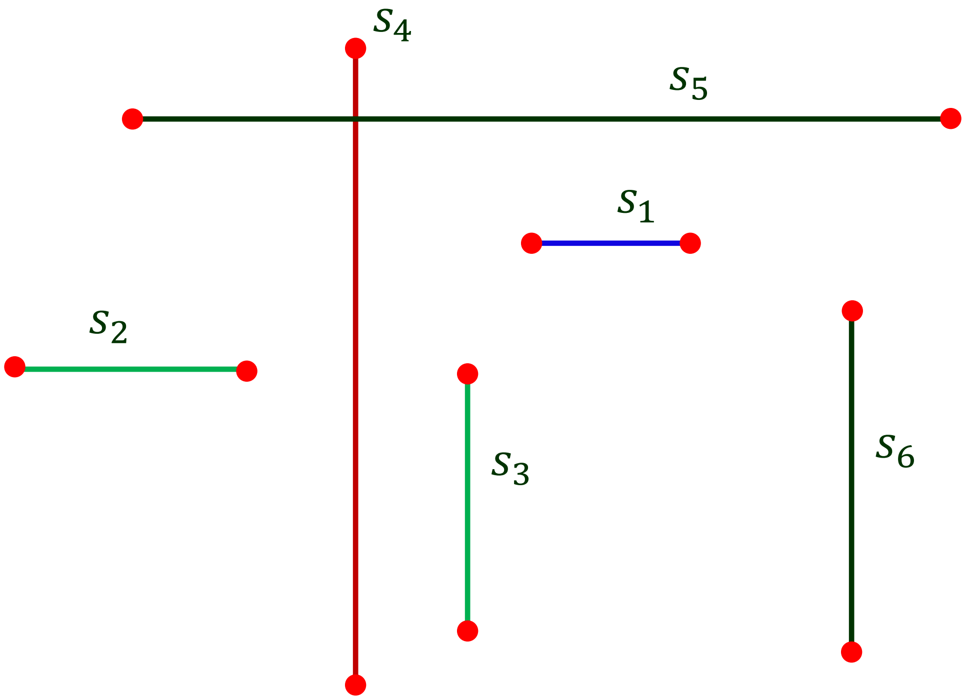
Use some data structure to store the map and allow quick retrieval of its portion inside a window.

vs. Range Query

	Windowing Query	Range Query
data	segments, polygons, curves	points
search space	2D or 3D	k-D

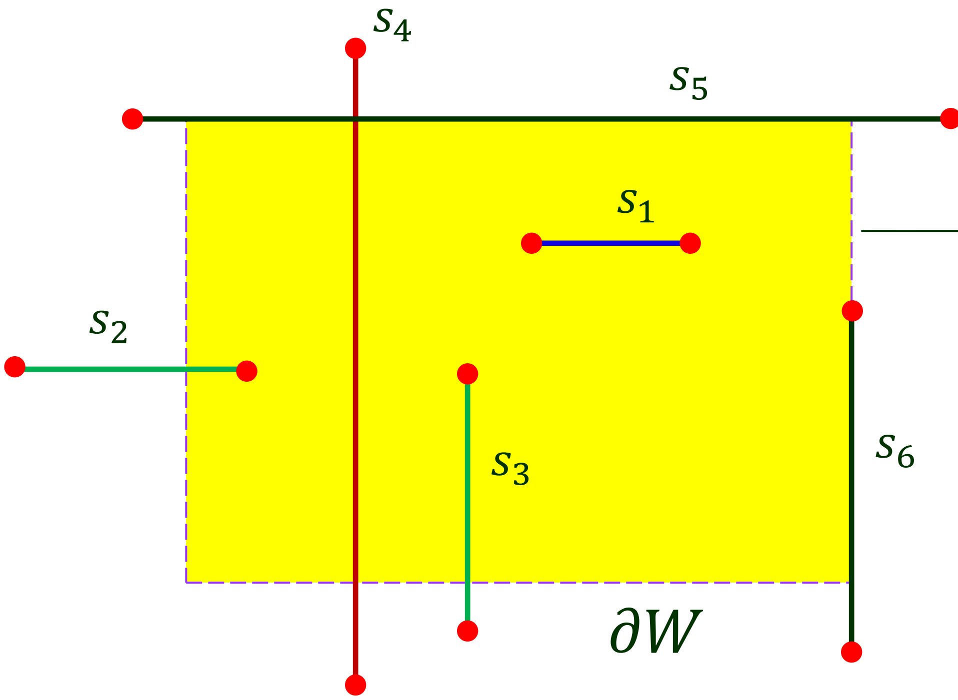
Formulating the Problem

$S = \{s_1, s_2, \ldots, s_n\}$: set of n axis-parallel segments



Formulating the Problem

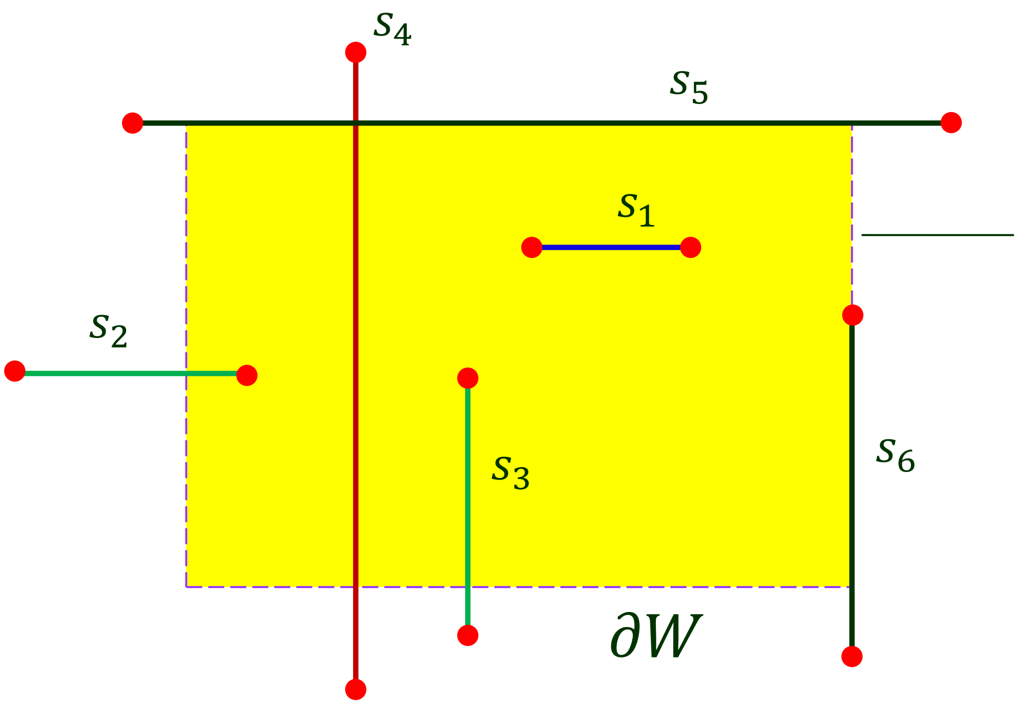
$S = \{s_1, s_2, \dots, s_n\}$: set of n axis-parallel segments



Query window: $W = [x, x'] \times [y, y']$
with boundary ∂W

Formulating the Problem

$S = \{s_1, s_2, \ldots, s_n\}$: set of n axis-parallel segments



Query window: $W = [x, x'] \times [y, y']$ with boundary ∂W

Possible cases of a segment s:

- s inside W;
- s intersects ∂W once;
- s intersects ∂W twice;
- s overlaps ∂W.

Segments with ≥ 1 Endpoints in W

Range query over $2n$ endpoints

$O(n \log n)$ storage

$O(n \log^2 n + k)$ query time

Segments with ≥ 1 Endpoints in W

Range query over $2n$ endpoints

$O(n \log n)$ storage

$O(n \log^2 n + k)$ query time

$\Downarrow$ fractional cascading

$O(n \log n)$

Segments with ≥ 1 Endpoints in W

Range query over $2n$ endpoints

$O(n \log n)$ storage

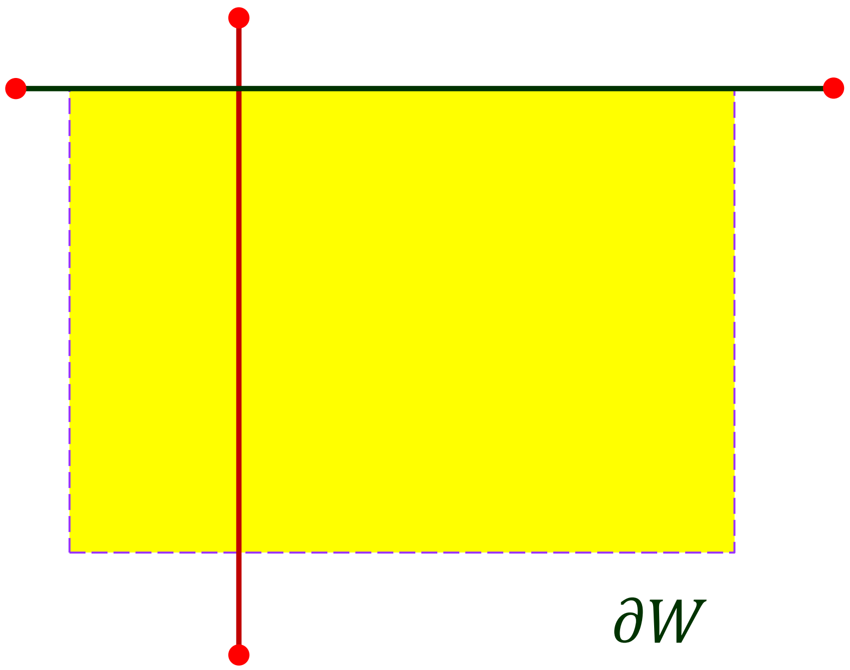
$O(n \log^2 n + k)$ query time

$\Downarrow$ fractional cascading

$O(n \log n)$

♣ Not to report a segment twice.

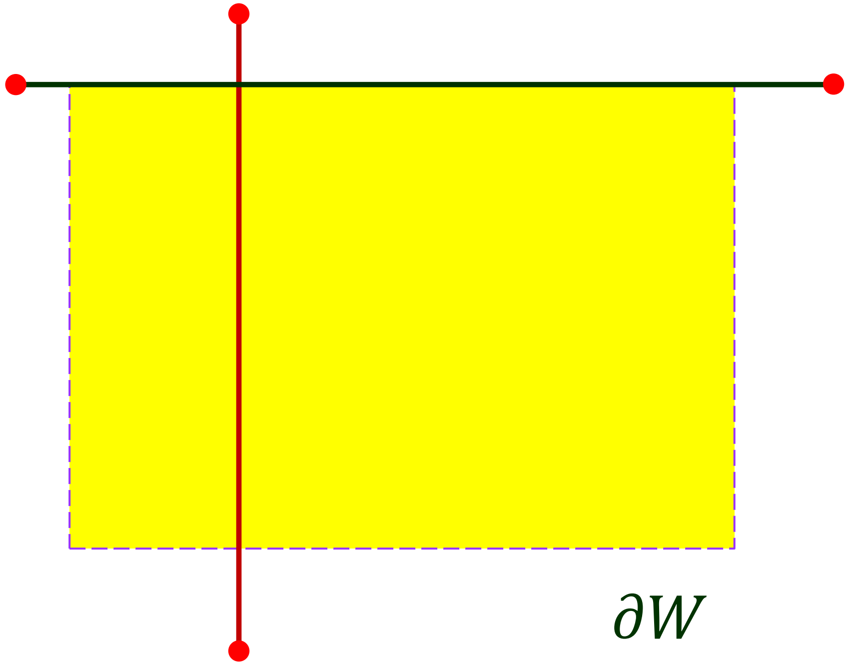
Segments with No Endpoint Inside W



Segments that pass through W may

- crosses W twice;
- contains one boundary edge.

Segments with No Endpoint Inside W

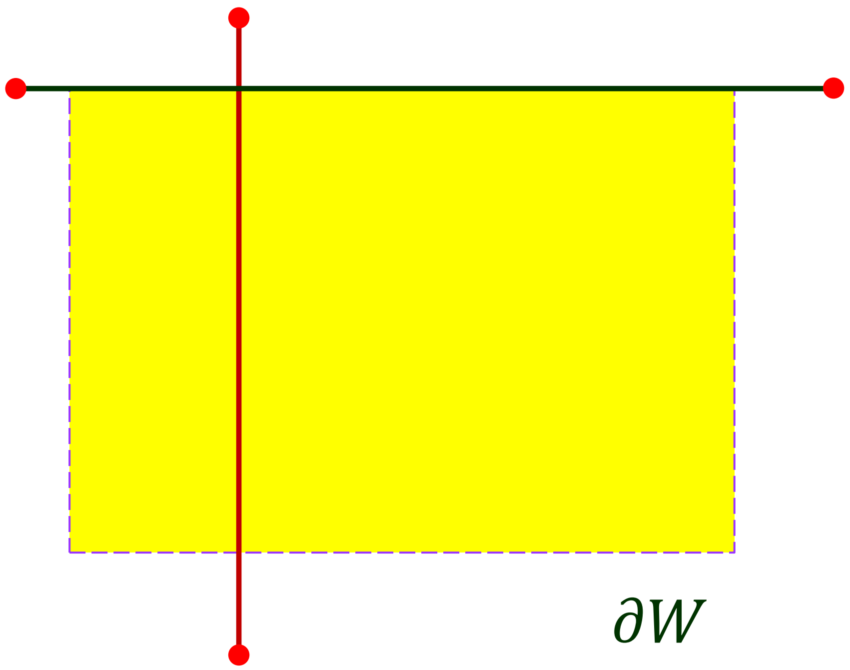


Segments that pass through W may

- crosses W twice;
- contains one boundary edge.

♦ Need only consider horizontal segments.

Segments with No Endpoint Inside W



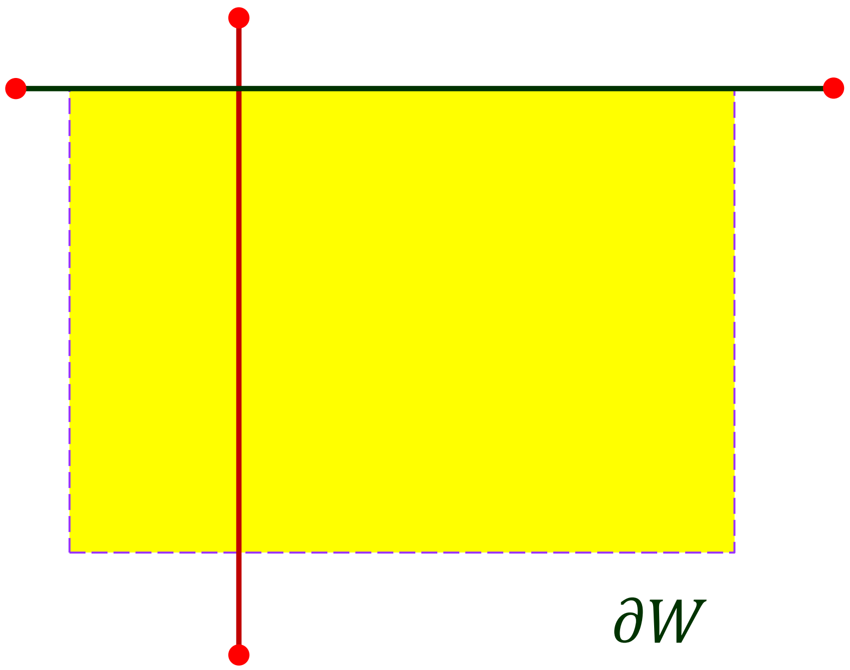
Segments that pass through W may

- crosses W twice;
- contains one boundary edge.

♦ Need only consider horizontal segments.

The case of vertical segments is similar.

Segments with No Endpoint Inside W



Segments that pass through W may

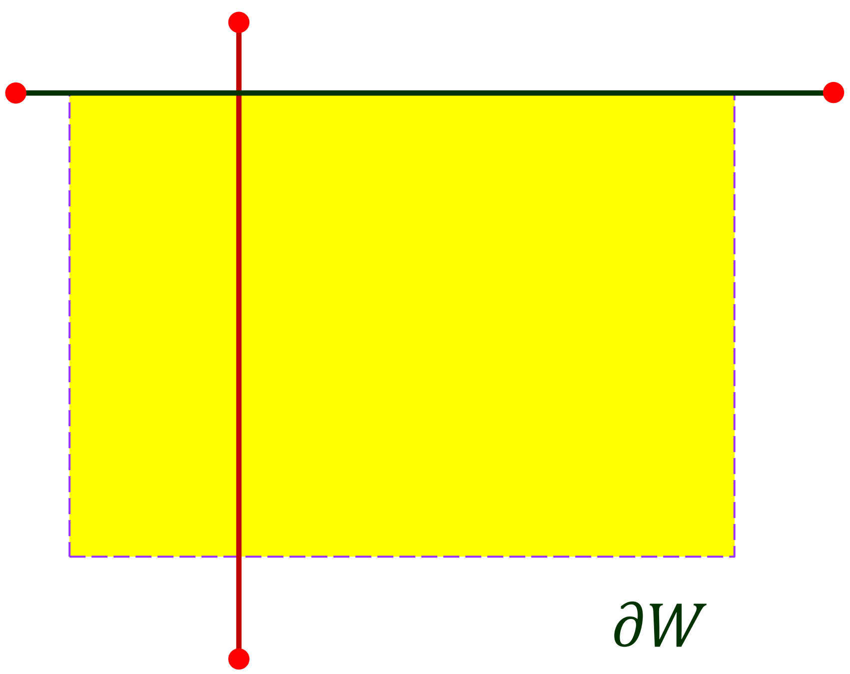
- crosses W twice;
- contains one boundary edge.

♦ Need only consider horizontal segments.

The case of vertical segments is similar.

♦ Need only know how to efficiently check if such a segment intersects the left boundary edge.

Segments with No Endpoint Inside W



Segments that pass through W may

- crosses W twice;
- contains one boundary edge.

♦ Need only consider horizontal segments.

The case of vertical segments is similar.

♦ Need only know how to efficiently check if such a segment intersects the left boundary edge.

It's similar to check if it intersects the right boundary edge.

II. Query Segment as a Full Line

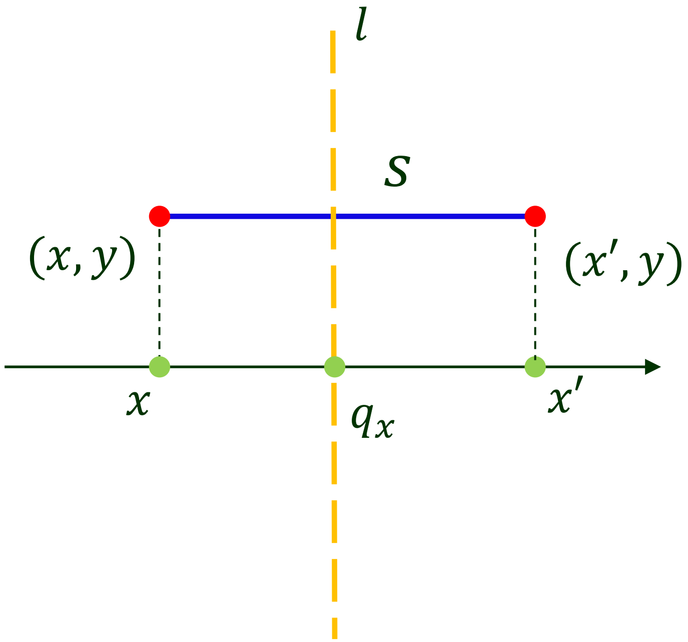
We first consider the *simpler* version of query when the left boundary edge of W is a *full line*.

$$l\text{:}\ \ x = q_x$$

II. Query Segment as a Full Line

We first consider the *simpler* version of query when the left boundary edge of W is a *full line*.

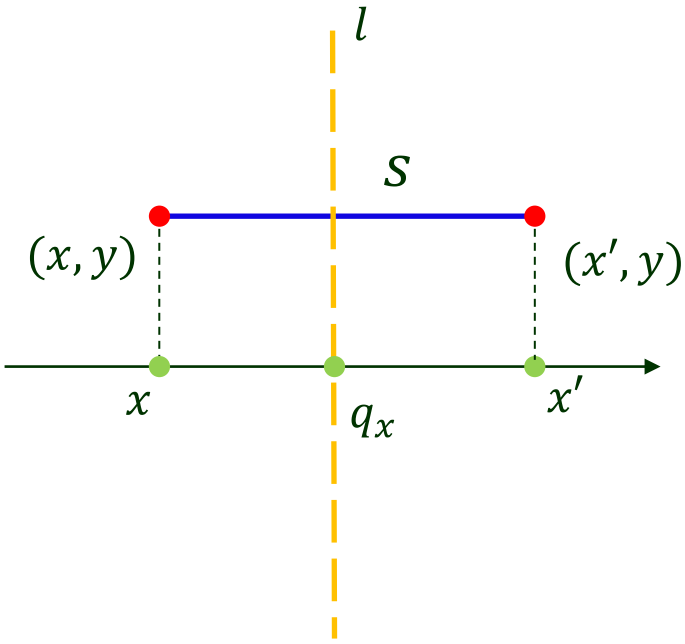
$$l\text{:}\ \ x = \ q_x$$



II. Query Segment as a Full Line

We first consider the *simpler* version of query when the left boundary edge of W is a *full line*.

$$l\colon\ x = q_x$$



Segment s: $[x, x'] \times y$ intersects l.

$$\Updownarrow$$

$$x \leq q_x \leq x'$$

II. Query Segment as a Full Line

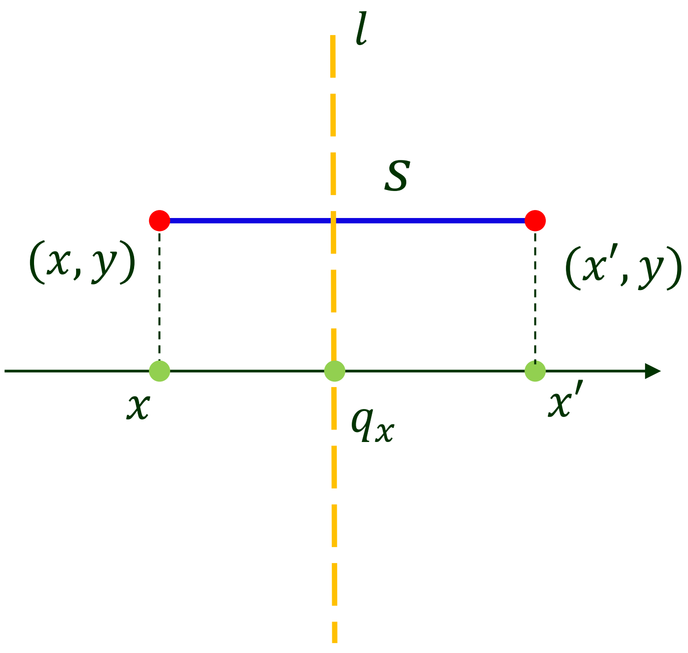
We first consider the *simpler* version of query when the left boundary edge of W is a *full line*.

$$l\colon\ x = q_x$$

Segment s: $[x, x'] \times y$ intersects l.

$$\Updownarrow$$

$$x \le q_x \le x'$$

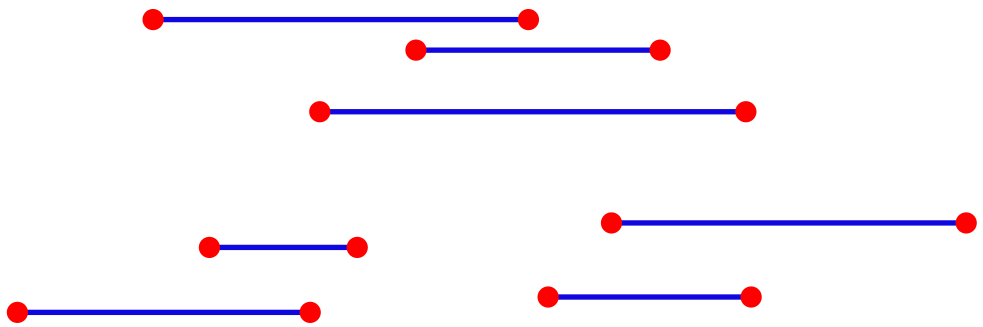


1D problem

Given a set of intervals $I = \{[x_1, y_1], [x_2, y_2] \ldots, [x_n, y_n]\}$, report those that contain the query point q_x.

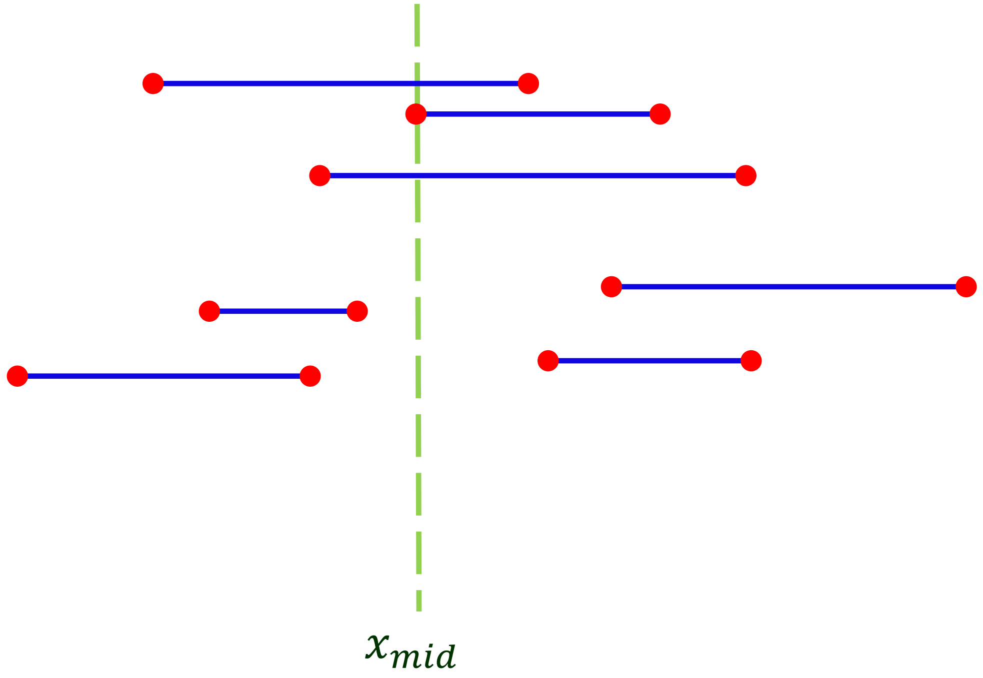
Partitioning the Interval Set

x_{mid}: median of $2n$ interval endpoints.



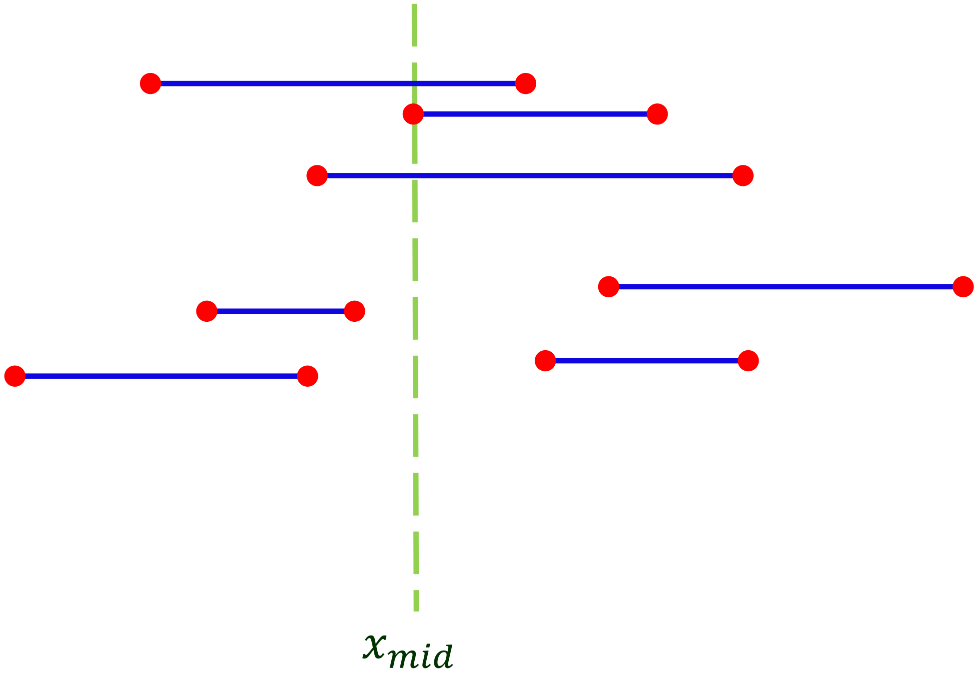
Partitioning the Interval Set

x_{mid}: median of $2n$ interval endpoints.



Partitioning the Interval Set

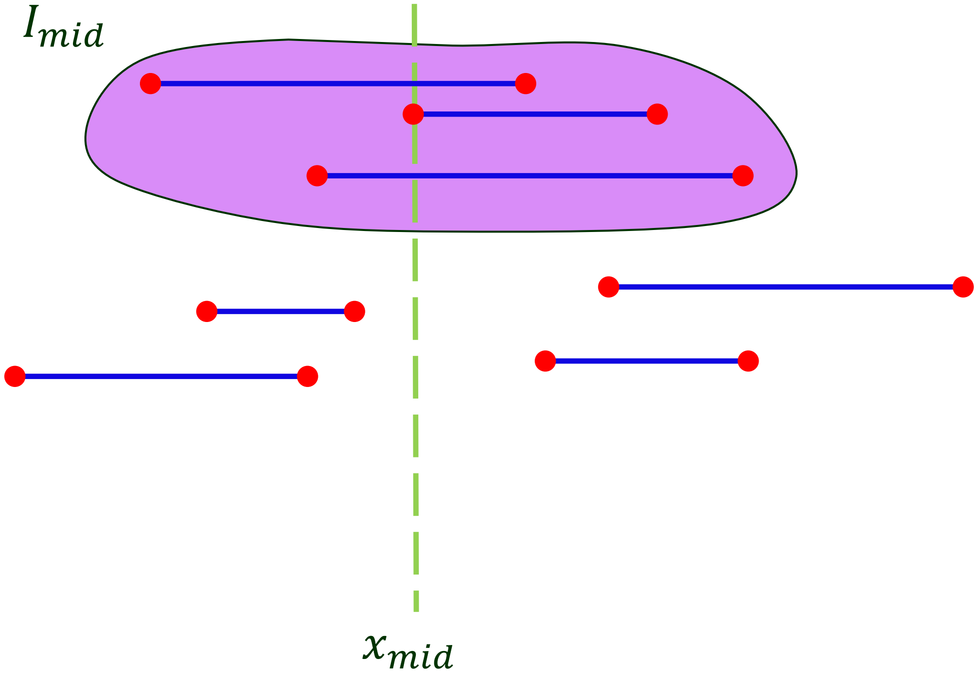
x_{mid}: median of $2n$ interval endpoints.



$$I_{mid} = \{[x_j, x_j'] \in I \mid x_j \le x_{mid} \le x_j'\}$$

Partitioning the Interval Set

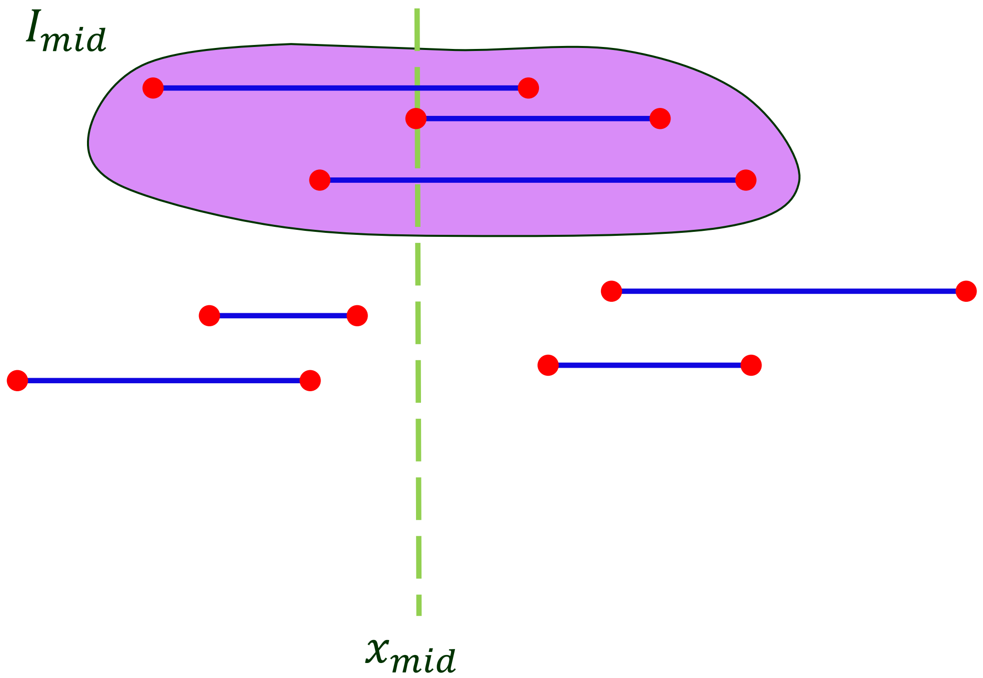
x_{mid}: median of $2n$ interval endpoints.



$$I_{mid} = \{[x_j, x_j'] \in I \mid x_j \le x_{mid} \le x_j'\}$$

Partitioning the Interval Set

x_{mid}: median of $2n$ interval endpoints.

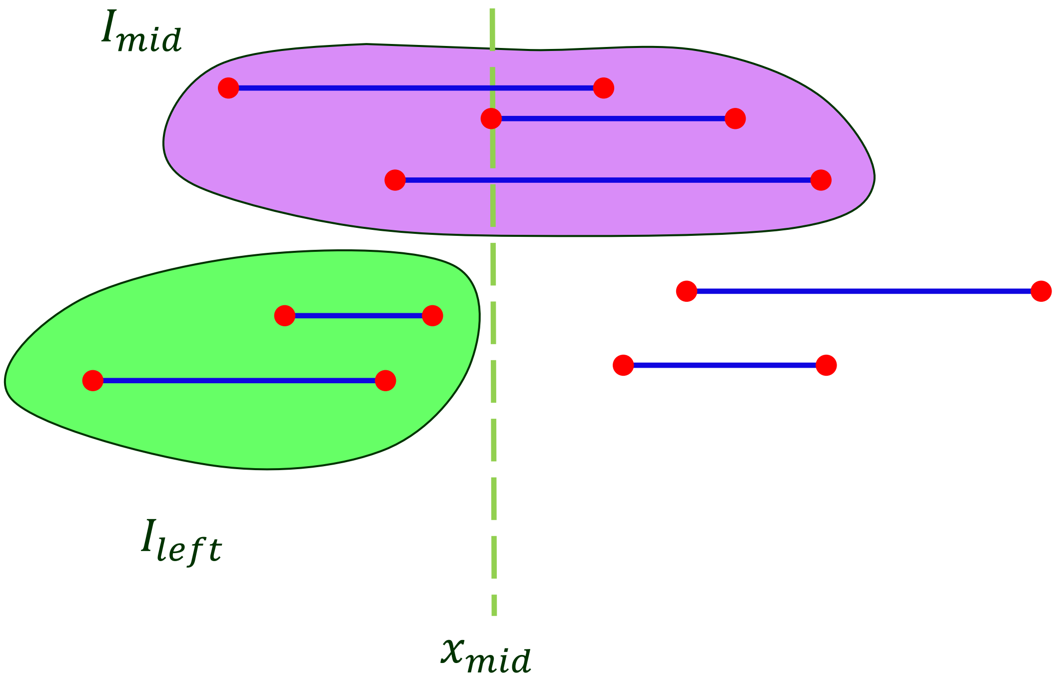


$$I_{mid} = \{[x_j, x_j'] \in I \mid x_j \le x_{mid} \le x_j'\}$$

$$I_{left} = \{[x_j, x_j'] \in I \mid x_j' < x_{mid}\}$$

Partitioning the Interval Set

x_{mid}: median of $2n$ interval endpoints.

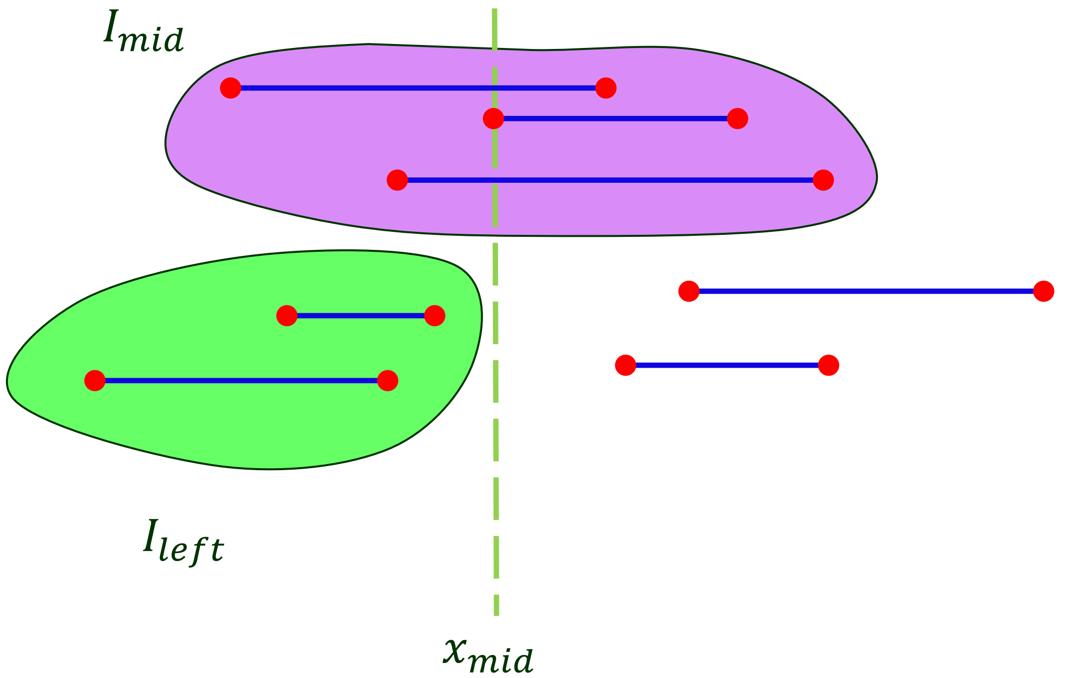


$$I_{mid} = \{[x_j, x_j'] \in I \mid x_j \le x_{mid} \le x_j'\}$$

$$I_{left} = \{[x_j, x_j'] \in I \mid x_j' < x_{mid}\}$$

Partitioning the Interval Set

x_{mid}: median of $2n$ interval endpoints.



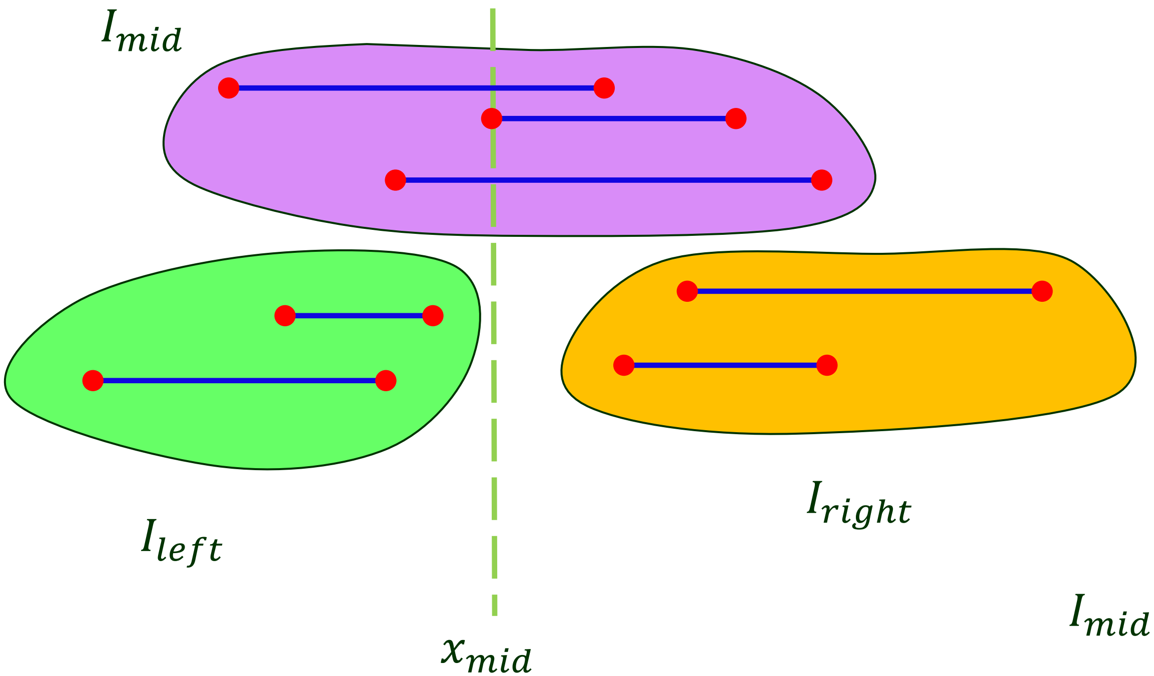
$$I_{mid} = \{[x_j, x_j'] \in I \mid x_j \le x_{mid} \le x_j'\}$$

$$I_{left} = \{[x_j, x_j'] \in I \mid x_j' < x_{mid}\}$$

$$I_{right} = \{[x_j, x_j'] \in I \mid x_j > x_{mid}\}$$

Partitioning the Interval Set

x_{mid}: median of $2n$ interval endpoints.

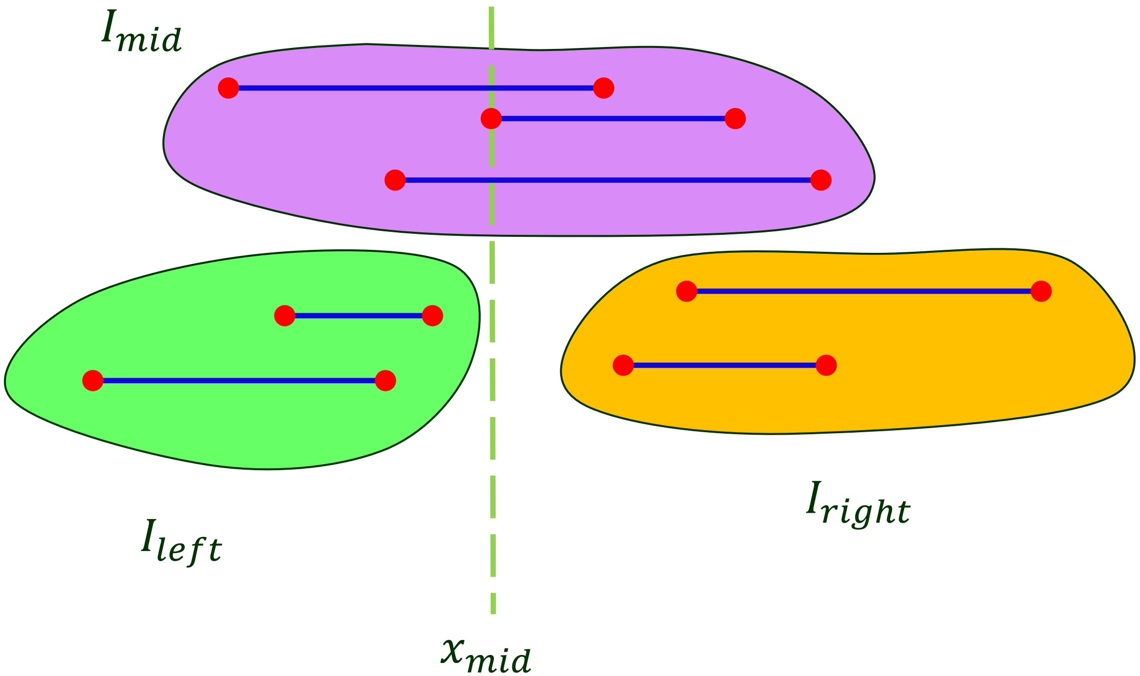


$$I_{mid} = \{[x_j, x_j'] \in I \mid x_j \le x_{mid} \le x_j'\}$$

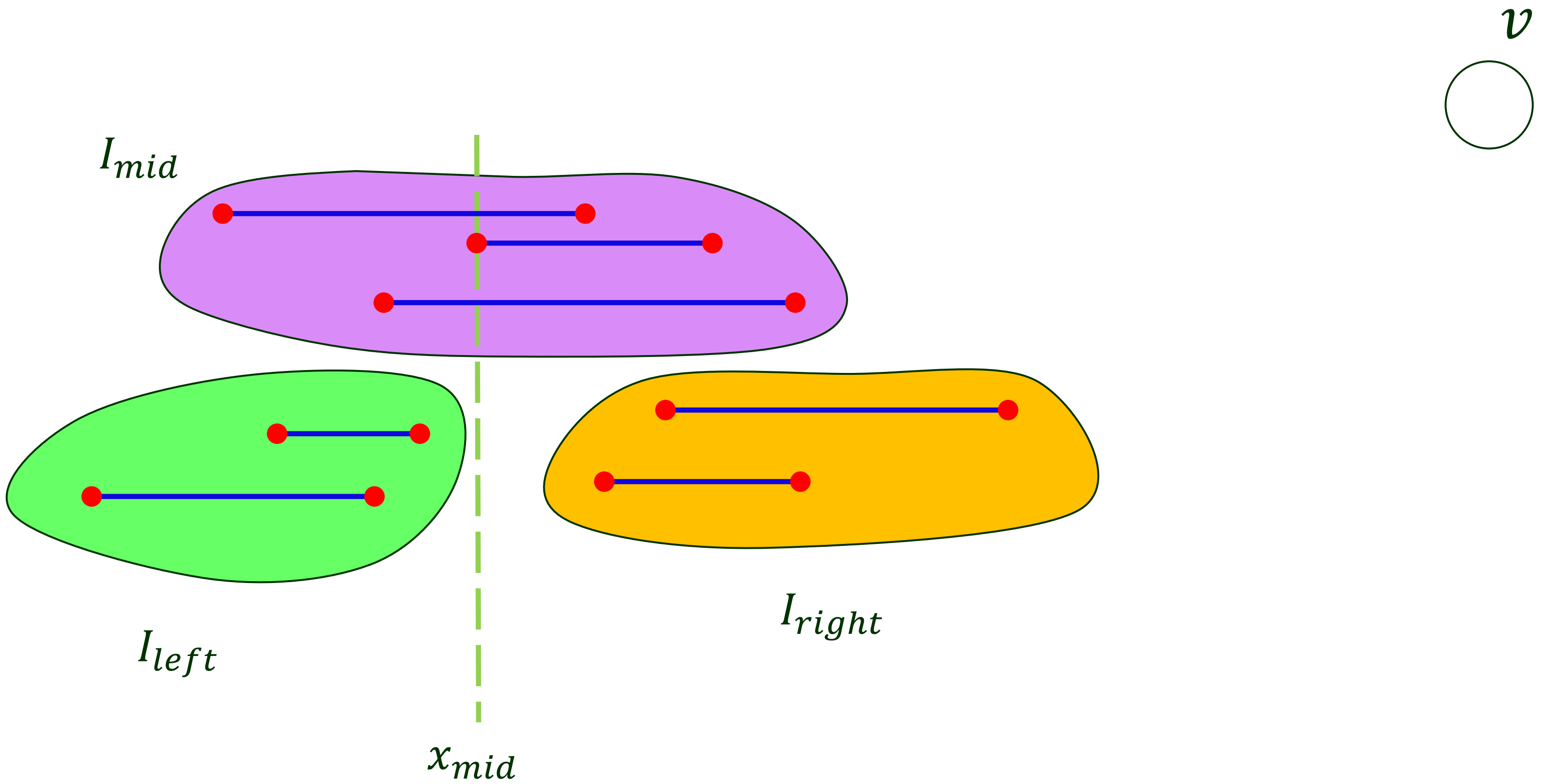
$$I_{left} = \{[x_j, x_j'] \in I \mid x_j' < x_{mid}\}$$

$$I_{right} = \{[x_j, x_j'] \in I \mid x_j > x_{mid}\}$$

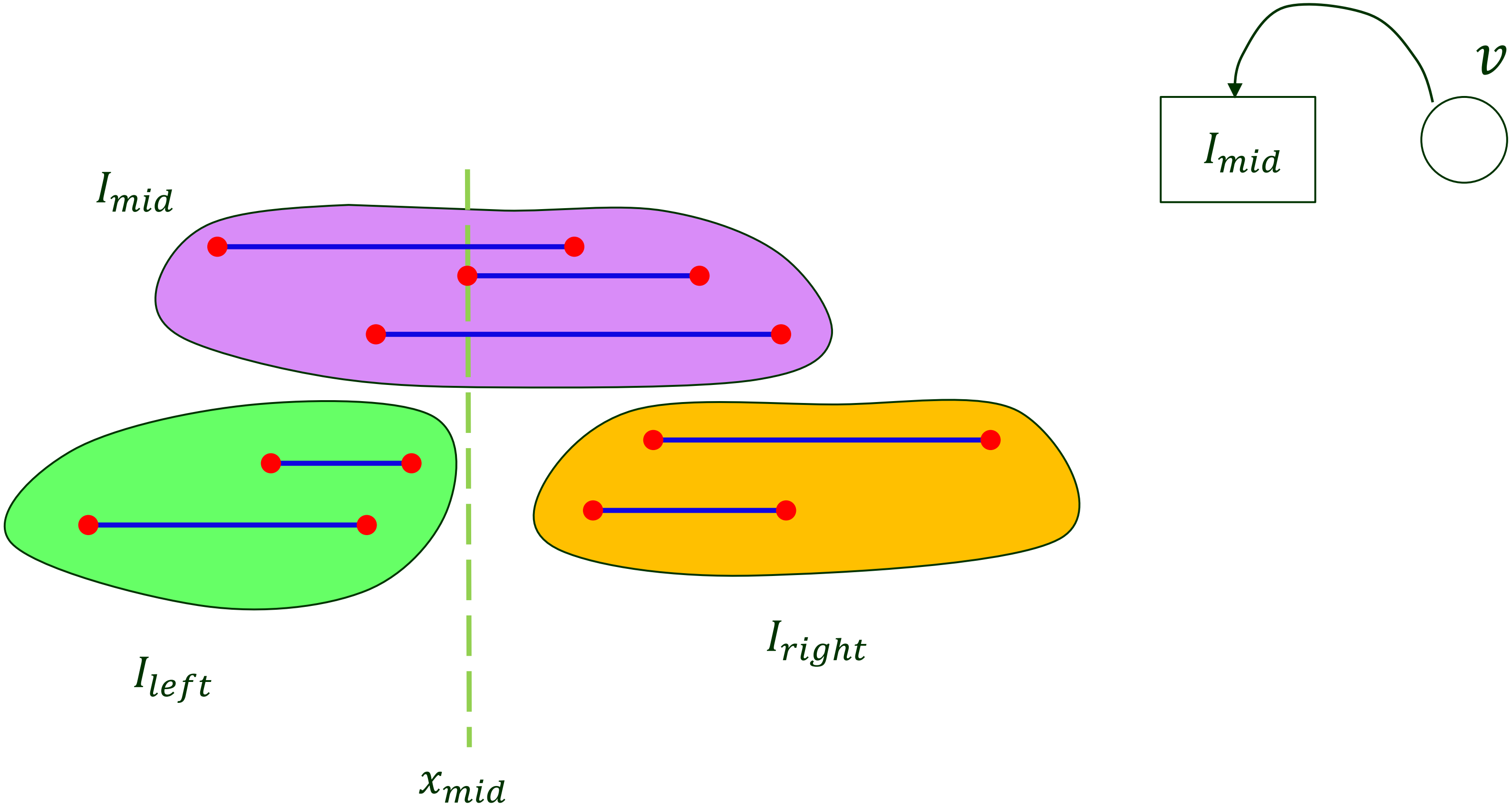
Interval Tree



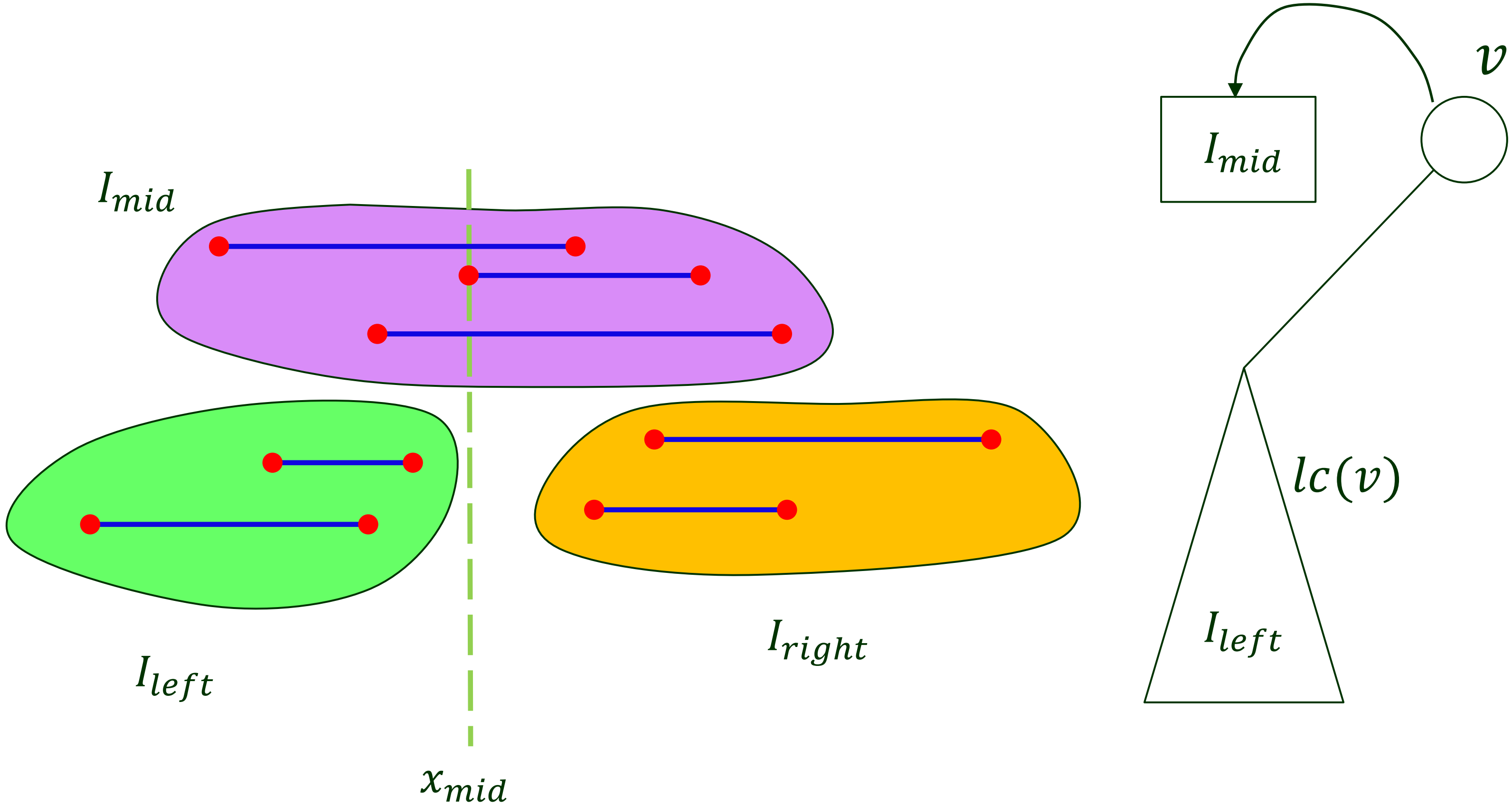
Interval Tree



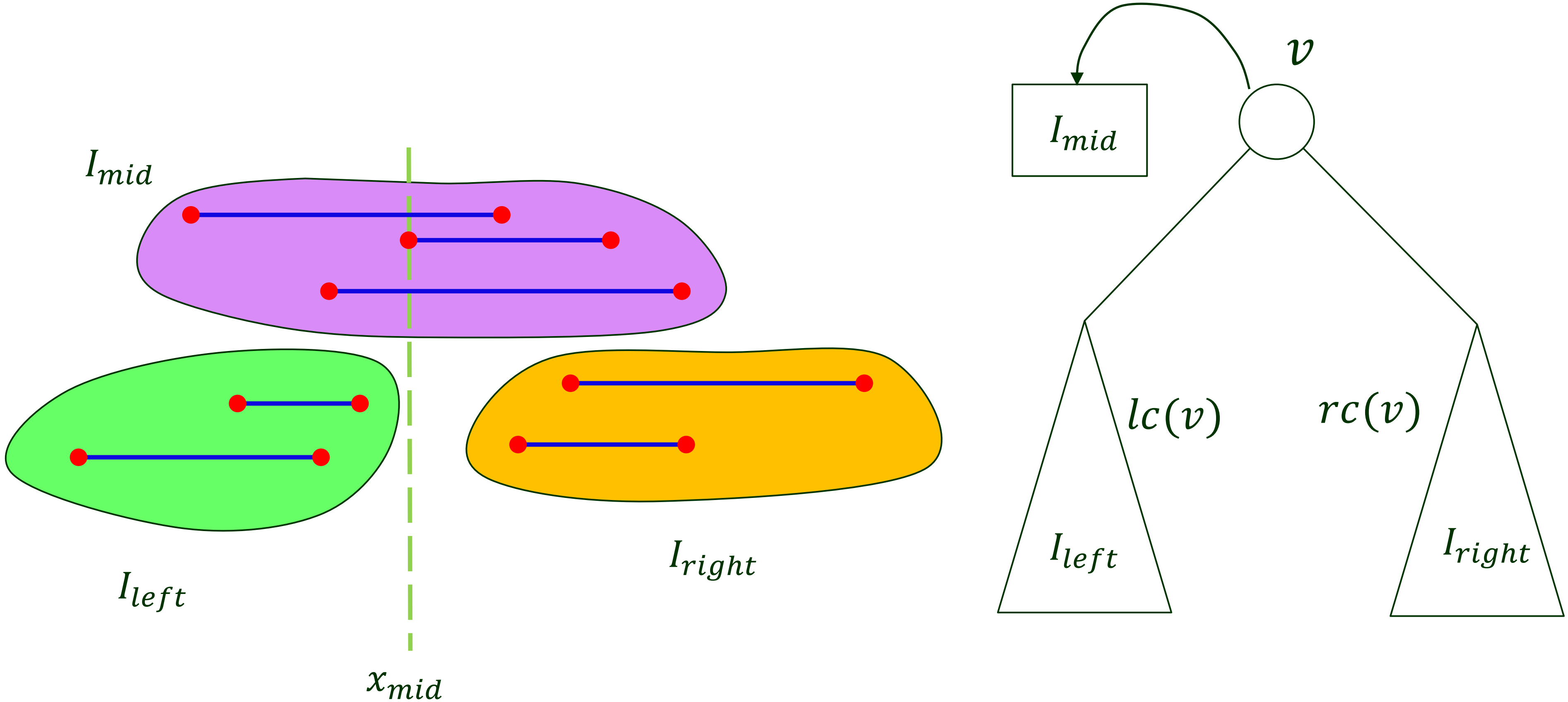
Interval Tree



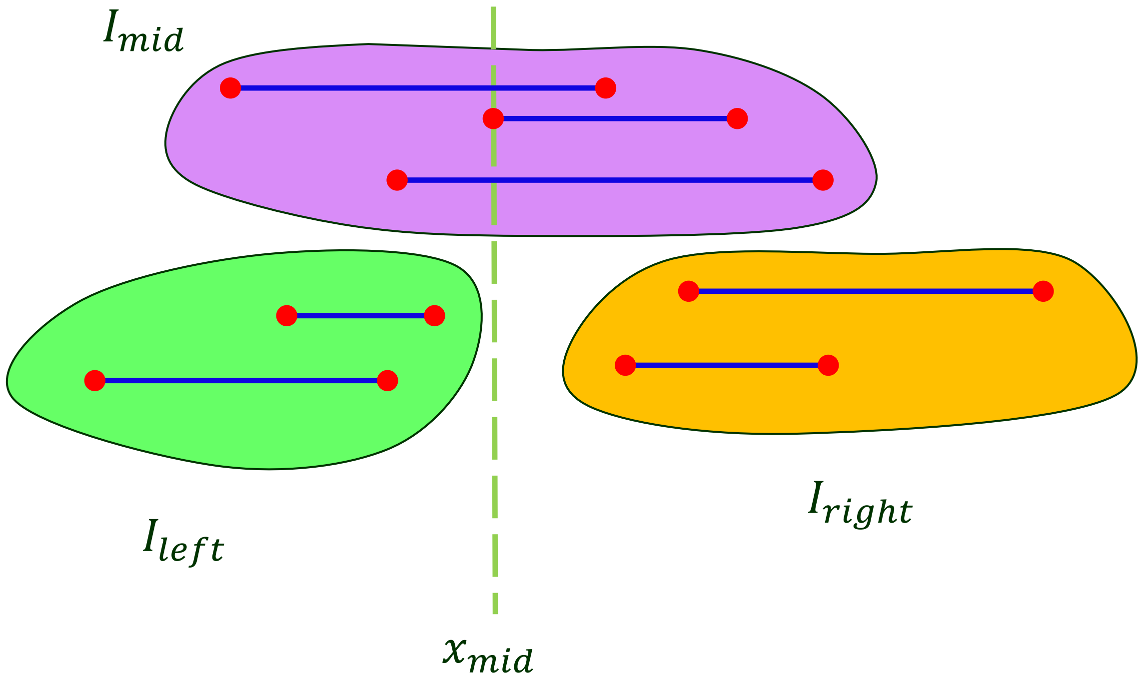
Interval Tree

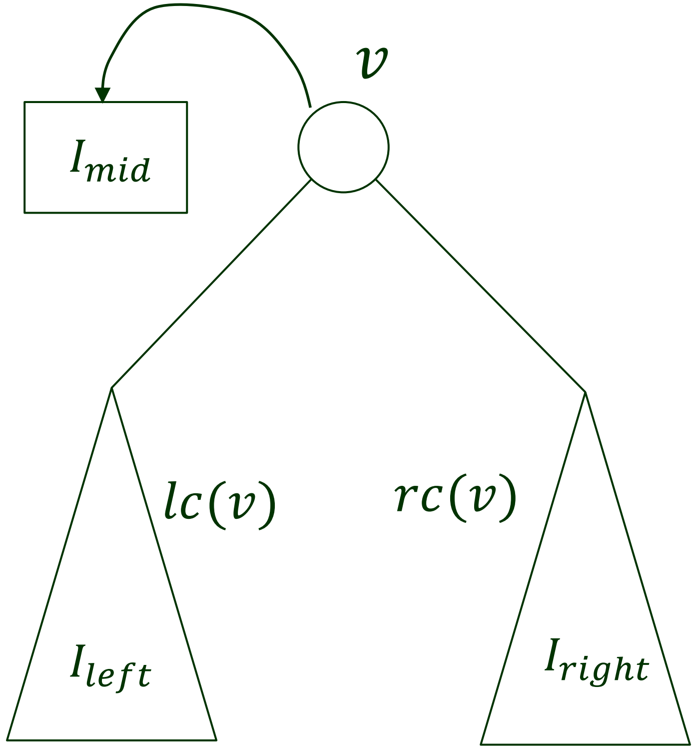


Interval Tree



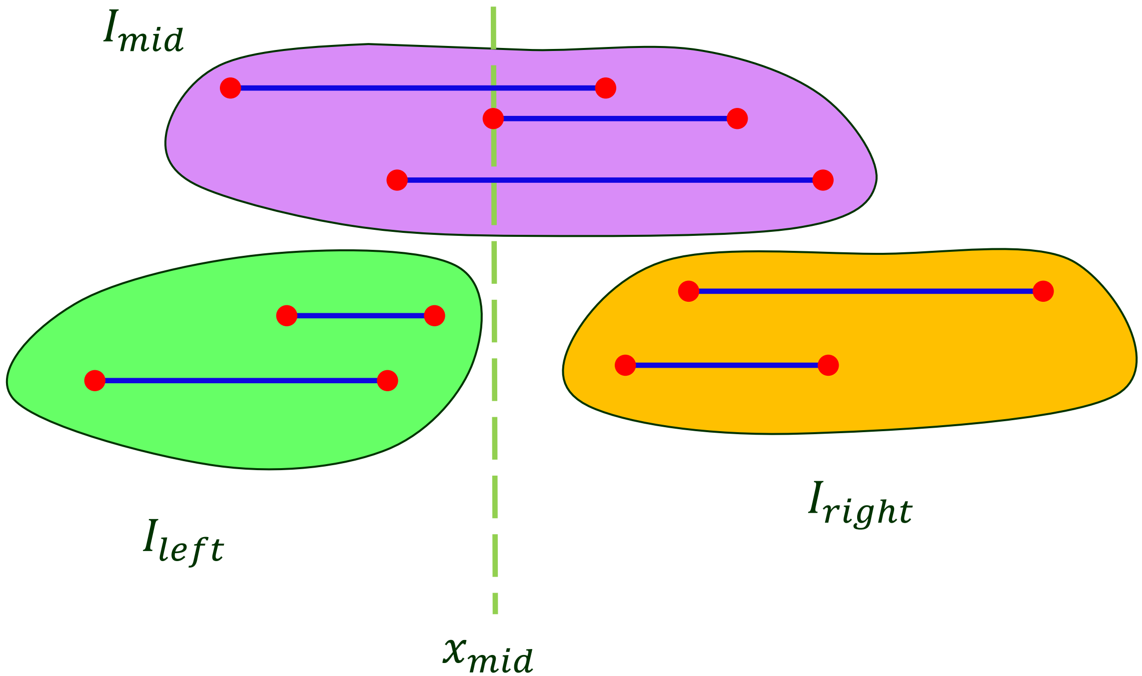
Interval Tree

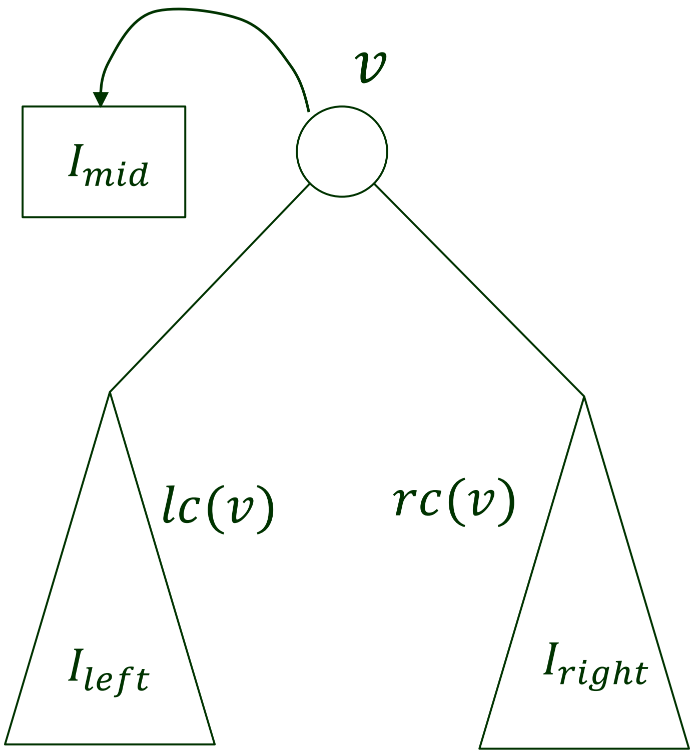




Subtrees $lc(v)$ and $rc(v)$ are recursively defined.

Interval Tree





Subtrees $lc(v)$ and $rc(v)$ are recursively defined.

$O(\log n)$ depth

Storage of I_{mid}

I_{mid} is stored in two lists at v:

Storage of I_{mid}

I_{mid} is stored in two lists at v:

- $\mathcal{L}_{left}(v)$: sorted in the increasing order on the left endpoints of the intervals.

Storage of I_{mid}

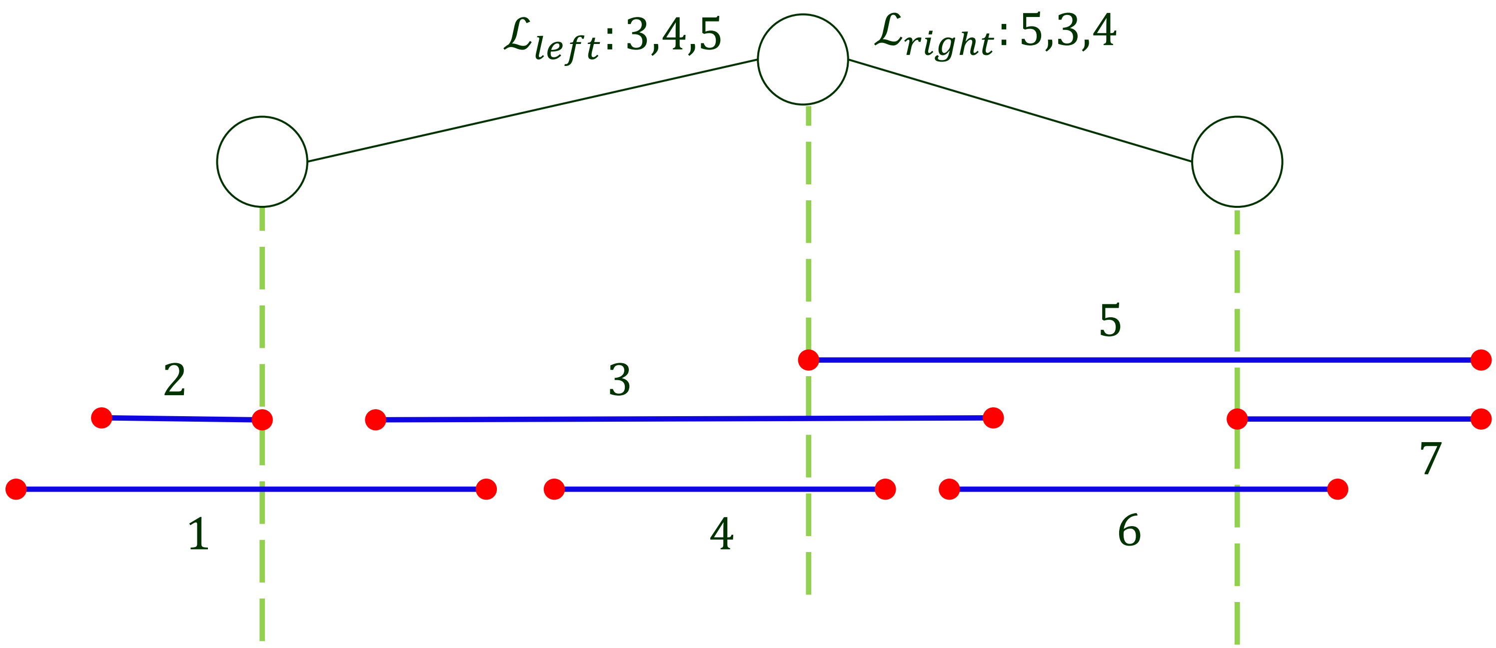
I_{mid} is stored in two lists at v:

- $\mathcal{L}_{left}(v)$: sorted in the increasing order on the left endpoints of the intervals.
- $\mathcal{L}_{right}(v)$: sorted in the decreasing order on their right endpoints.

Storage of I_{mid}

I_{mid} is stored in two lists at v:

- $\mathcal{L}_{left}(v)$: sorted in the increasing order on the left endpoints of the intervals.
- $\mathcal{L}_{right}(v)$: sorted in the decreasing order on their right endpoints.



Total Storage

$I_{left}, I_{mid}, I_{right}$ are disjoint.

Every interval appears in only one set I_{mid} (stored once in each of the two sorted lists).

- All associated lists require $O(n)$ storage.
- The tree uses $O(n)$ storage.
- The tree has height $O(\log n)$.

III. Construction of the Interval Tree

ConstructInveralTree(I)

if $I = \emptyset$

    then return an empty leaf

    else compute x_{mid}

        store it in a new node v

        compute I_{mid}, I_{left}, I_{right}

        construct sorted lists $\mathcal{L}_{left}(v)$ and $\mathcal{L}_{right}(v)$

        store them at v

        $lc(v) \leftarrow$ ConstructIntervalTree(I_{left})

        $rc(v) \leftarrow$ ConstructIntervalTree(I_{right})

        return v

III. Construction of the Interval Tree

ConstructInveralTree(I)

if $I = \emptyset$
    then return an empty leaf
    else compute x_{mid}
        store it in a new node v
        compute I_{mid}, I_{left}, I_{right}
        construct sorted lists $\mathcal{L}_{left}(v)$ and $\mathcal{L}_{right}(v)$
        store them at v
        $lc(v) \leftarrow$ ConstructIntervalTree(I_{left})
        $rc(v) \leftarrow$ ConstructIntervalTree(I_{right})
        return v

Presort the points and maintain ordered lists through recursive calls.

III. Construction of the Interval Tree

ConstructInveralTree(I)

if $I = \emptyset$
    then return an empty leaf
    else compute x_{mid} // $O(n)$
        store it in a new node v
        compute I_{mid}, I_{left}, I_{right}
        construct sorted lists $\mathcal{L}_{left}(v)$ and $\mathcal{L}_{right}(v)$
        store them at v
        $lc(v) \leftarrow$ ConstructIntervalTree(I_{left})
        $rc(v) \leftarrow$ ConstructIntervalTree(I_{right})
        return v

Presort the points and maintain ordered lists through recursive calls.

III. Construction of the Interval Tree

ConstructInveralTree(I)

if $I = \emptyset$
    then return an empty leaf
    else compute x_{mid} // $O(n)$
        store it in a new node v
        compute I_{mid}, I_{left}, I_{right} // $O(n)$
        construct sorted lists $\mathcal{L}_{left}(v)$ and $\mathcal{L}_{right}(v)$
        store them at v
        $lc(v) \leftarrow$ ConstructIntervalTree(I_{left})
        $rc(v) \leftarrow$ ConstructIntervalTree(I_{right})
        return v

Presort the points and maintain ordered lists through recursive calls.

III. Construction of the Interval Tree

ConstructInveralTree(I)

if $I = \emptyset$
  then return an empty leaf
  else compute x_{mid} // $O(n)$
    store it in a new node v
    compute I_{mid}, I_{left}, I_{right} // $O(n)$
    construct sorted lists $\mathcal{L}_{left}(v)$ and $\mathcal{L}_{right}(v)$
    store them at v // $O(n)$
    $lc(v) \leftarrow$ ConstructIntervalTree(I_{left})
    $rc(v) \leftarrow$ ConstructIntervalTree(I_{right})
    return v

Presort the points and maintain ordered lists through recursive calls.

III. Construction of the Interval Tree

ConstructInveralTree(I)

1. **if** $I = \emptyset$
2.   **then** return an empty leaf
3.   **else** compute x_{mid} // $O(n)$
4.   store it in a new node v
5.   compute I_{mid}, I_{left}, I_{right} // $O(n)$
6.   construct sorted lists $\mathcal{L}_{left}(v)$ and $\mathcal{L}_{right}(v)$
7.   store them at v // $O(n)$
8. $lc(v) \leftarrow$ ConstructIntervalTree(I_{left})
9. $rc(v) \leftarrow$ ConstructIntervalTree(I_{right})
10.   **return** v

Presort the points and maintain ordered lists through recursive calls.

$\Longrightarrow$ $O(n)$ time before recursions on lines 8 and 9

III. Construction of the Interval Tree

ConstructInveralTree(I)

1. **if** $I = \emptyset$
2.   **then** return an empty leaf
3.   **else** compute x_{mid} // $O(n)$
4.   store it in a new node v
5.   compute I_{mid}, I_{left}, I_{right} // $O(n)$
6.   construct sorted lists $\mathcal{L}_{left}(v)$ and $\mathcal{L}_{right}(v)$
7.   store them at v // $O(n)$
8. $lc(v) \leftarrow$ ConstructIntervalTree(I_{left})
9. $rc(v) \leftarrow$ ConstructIntervalTree(I_{right})
10.   **return** v

Presort the points and maintain ordered lists through recursive calls.

$\Rightarrow$ $O(n)$ time before recursions on lines 8 and 9

$\Rightarrow$ Total construction time: $O(n \log n)$

Query

Find all intervals containing q_x.

- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.

Query

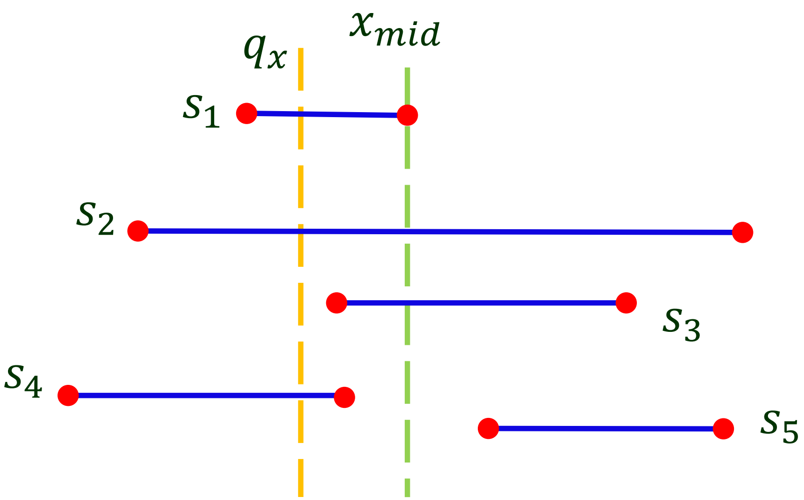
Find all intervals containing q_x.

- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.
  - $q_x < x_{mid}(v)$

Query

Find all intervals containing q_x.

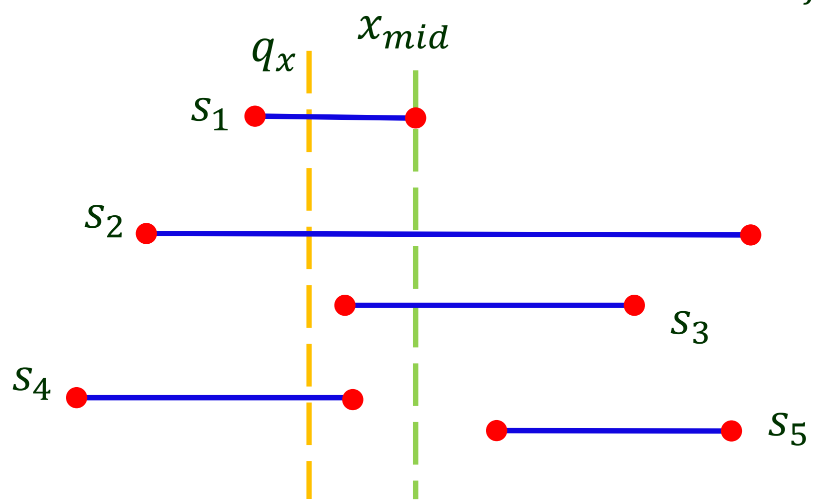
- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.
  - $q_x < x_{mid}(v)$



Query

Find all intervals containing q_x.

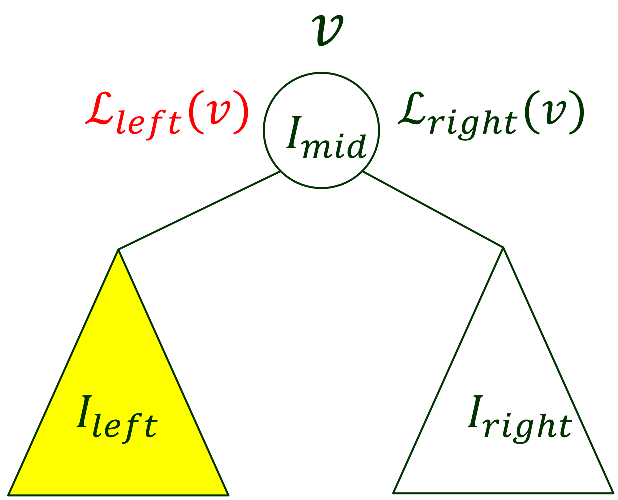
- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.
- $q_x < x_{mid}(v) \implies$ Search I_{mid} and I_{left}.

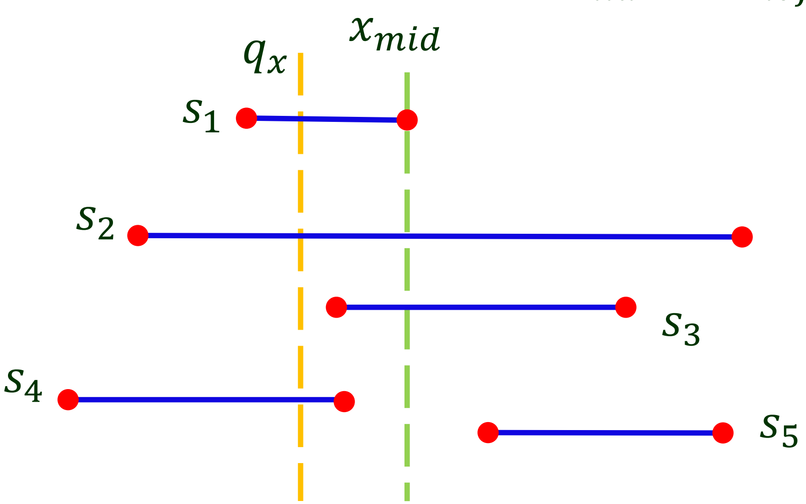


Query

Find all intervals containing q_x.

- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.
- $q_x < x_{mid}(v)$ $\Longrightarrow$ Search I_{mid} and I_{left}.

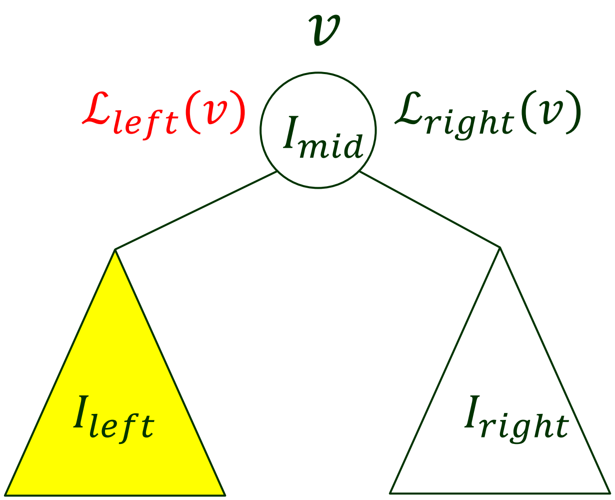


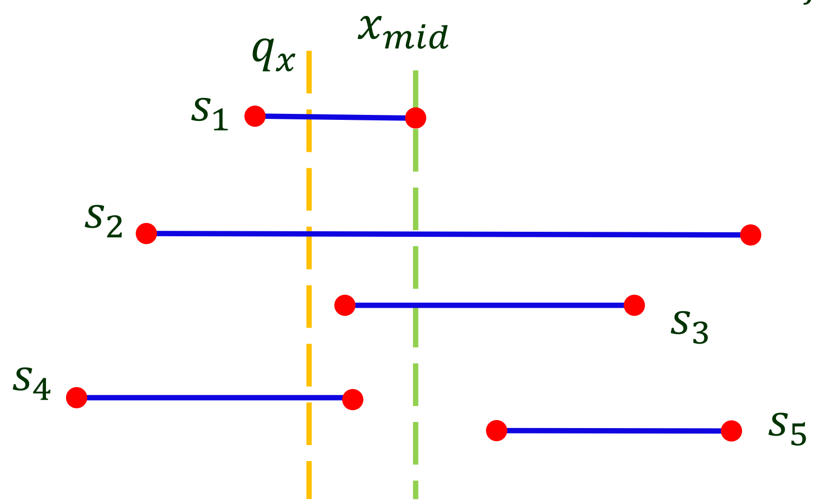


Query

Find all intervals containing q_x.

- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.
- $q_x < x_{mid}(v) \implies$ Search I_{mid} and I_{left}.



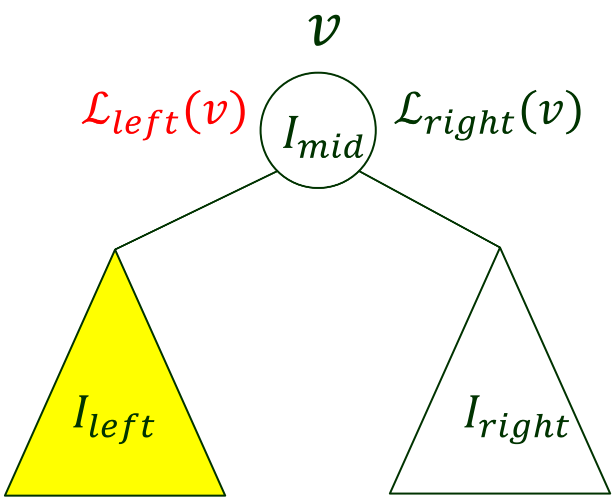


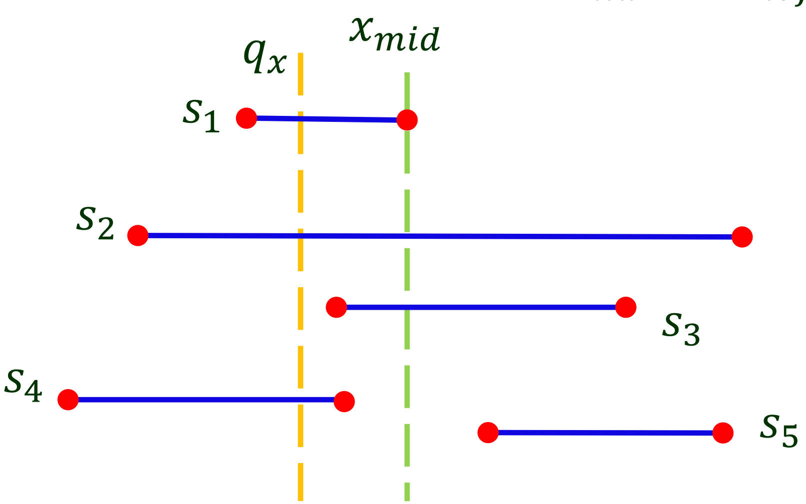
$\mathcal{L}_{left}(v) = \langle s_2, s_1, s_3 \rangle$

Query

Find all intervals containing q_x.

- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.
- $q_x < x_{mid}(v) \implies$ Search I_{mid} and I_{left}.





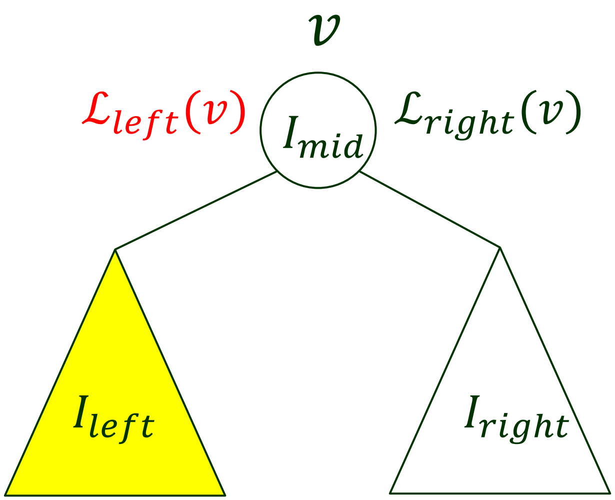
$\mathcal{L}_{left}(v) = \langle s_2, s_1, s_3 \rangle$

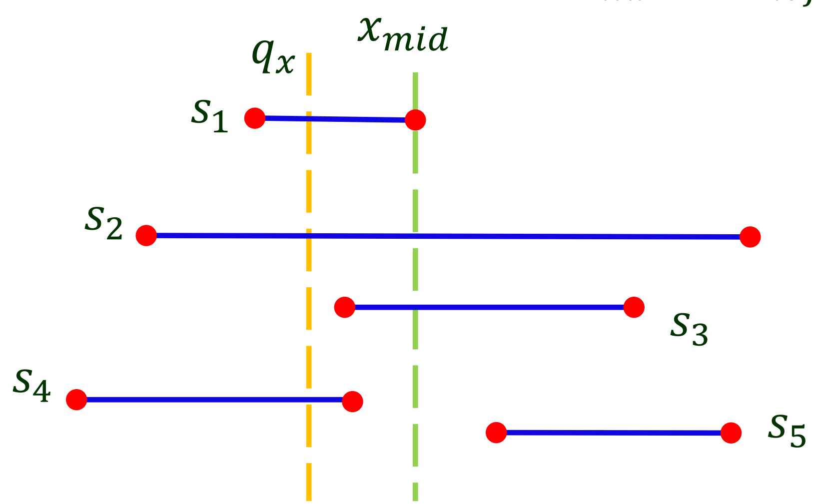
- At v, start at the leftmost endpoint of $\mathcal{L}_{left}$.

Query

Find all intervals containing q_x.

- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.
- $q_x < x_{mid}(v) \implies$ Search I_{mid} and I_{left}.





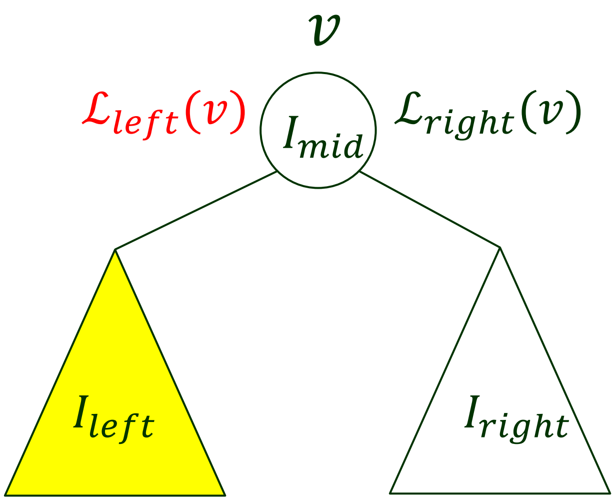
$\mathcal{L}_{left}(v) = \langle s_2, s_1, s_3 \rangle$

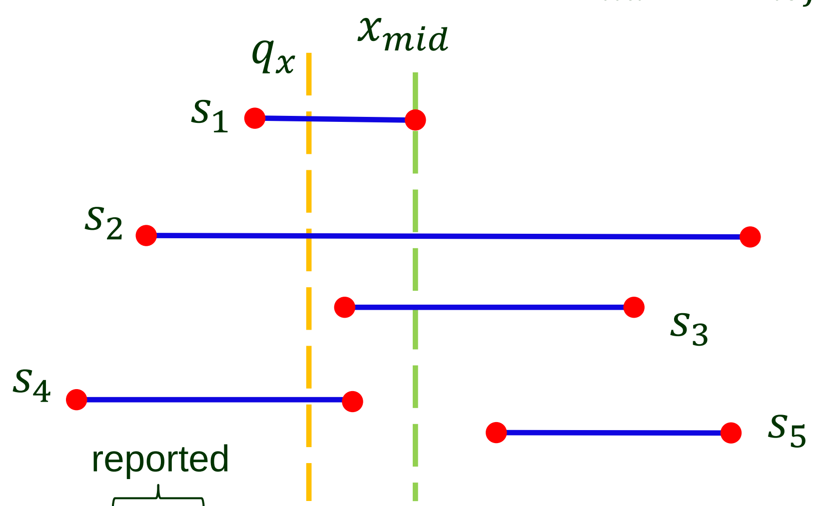
- At v, start at the leftmost endpoint of $\mathcal{L}_{left}$.
- Report all the intervals containing q_x.

Query

Find all intervals containing q_x.

- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.
- $q_x < x_{mid}(v) \implies$ Search I_{mid} and I_{left}.





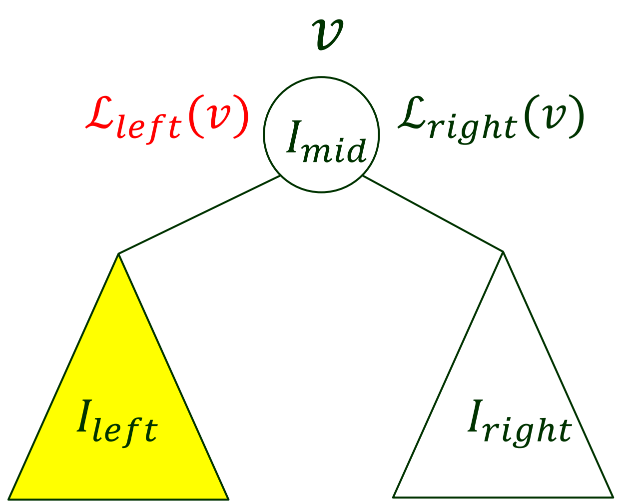
$$\mathcal{L}_{left}(v) = \langle \overbrace{s_2, s_1}^{\text{reported}}, s_3 \rangle$$

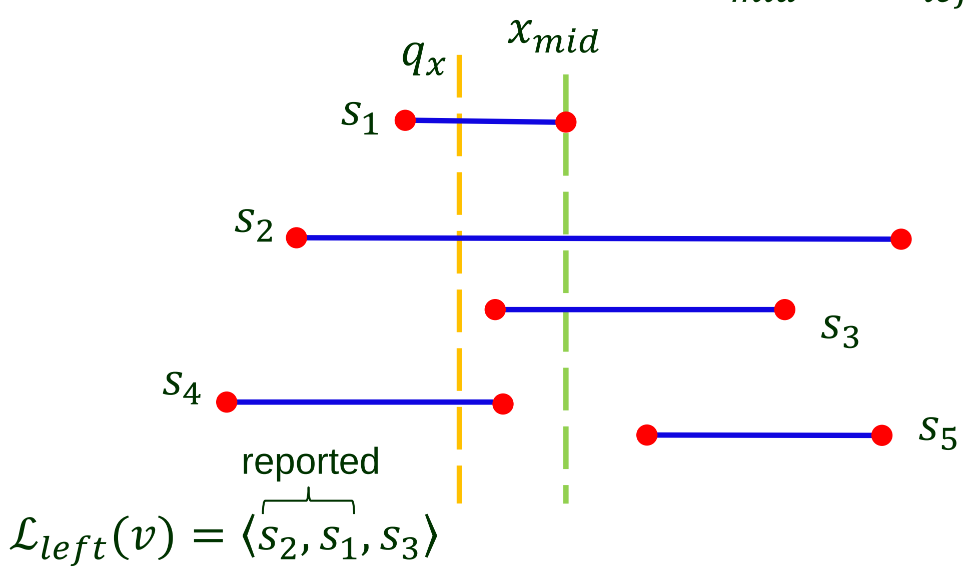
- At v, start at the leftmost endpoint of $\mathcal{L}_{left}$.
- Report all the intervals containing q_x.

Query

Find all intervals containing q_x.

- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.
- $q_x < x_{mid}(v) \implies$ Search I_{mid} and I_{left}.



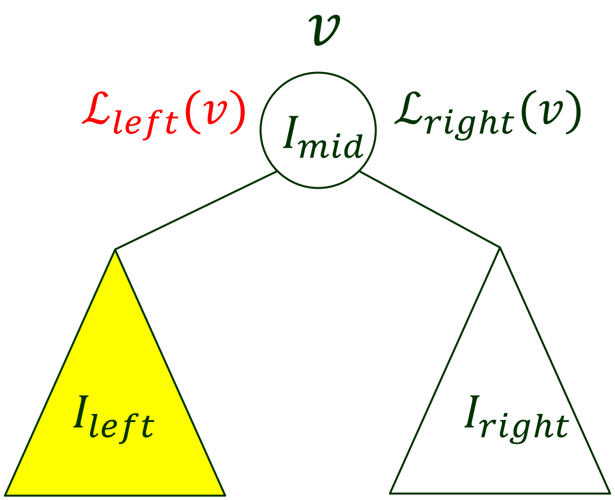


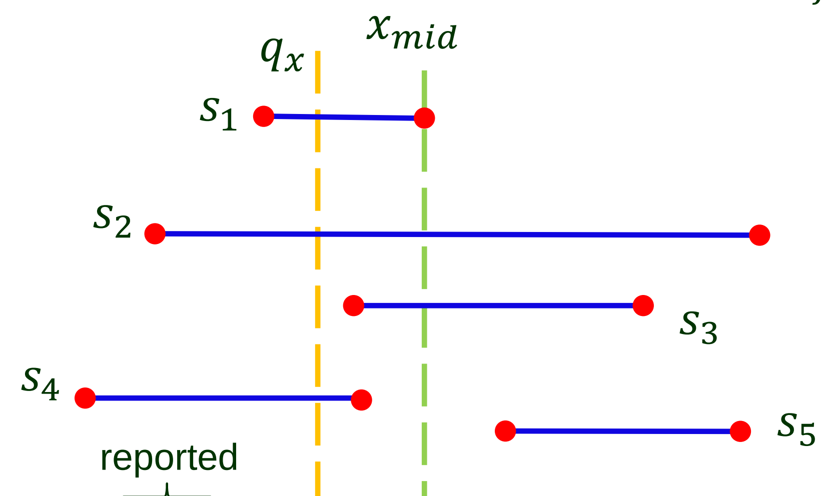
- At v, start at the leftmost endpoint of $\mathcal{L}_{left}$.
- Report all the intervals containing q_x.
- Stop at an interval that does not contain q_x (i.e., its left endpoint is to the right of q_x).

Query

Find all intervals containing q_x.

- $q_x = x_{mid}$ ends the query after reporting all intervals in I_{mid}.
- $q_x < x_{mid}(v) \implies$ Search I_{mid} and I_{left}.



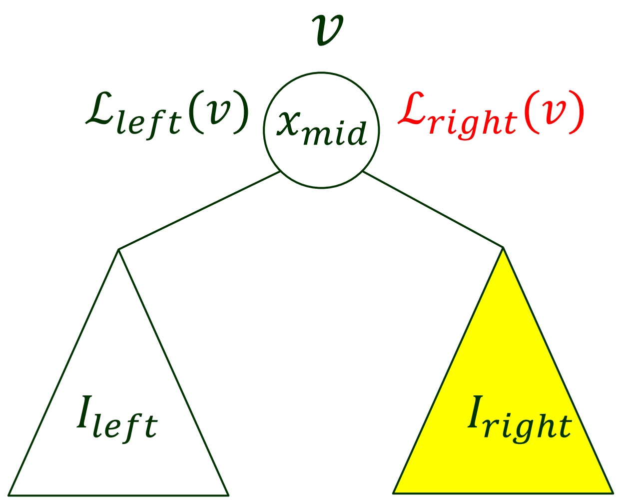


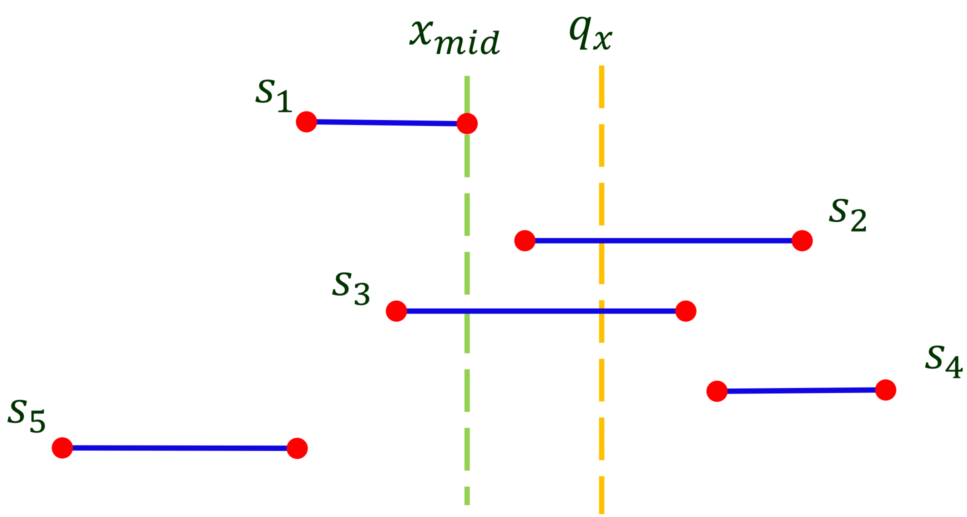
$\mathcal{L}_{left}(v) = \langle \overbrace{s_2, s_1}^{\text{reported}}, s_3 \rangle$

- At v, start at the leftmost endpoint of $\mathcal{L}_{left}$.
- Report all the intervals containing q_x.
- Stop at an interval that does not contain q_x (i.e., its left endpoint is to the right of q_x).
- Query the left subtree.

Query (cont'd)

- $q_x > x_{mid}(v) \quad \Rightarrow \quad$ Search I_{mid} and I_{right}.



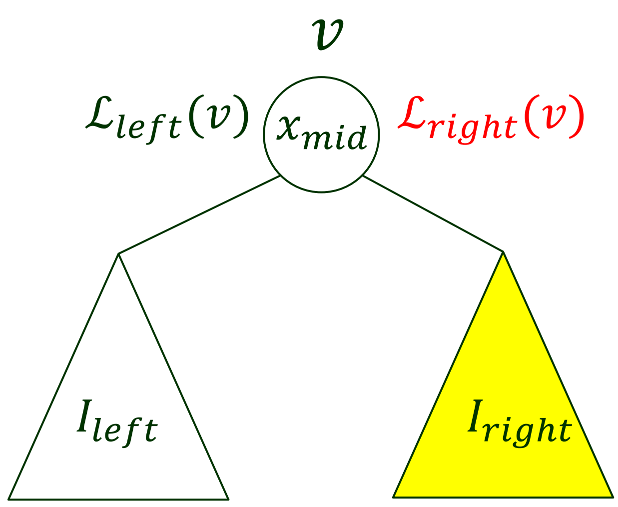


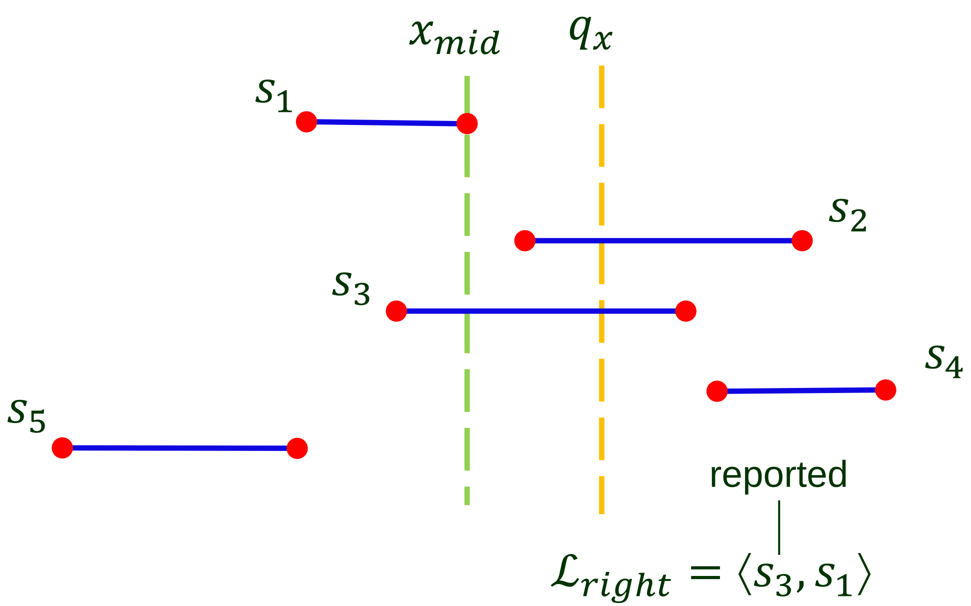
$\mathcal{L}_{right} = \langle s_3, s_1 \rangle$

- At v, start at the rightmost endpoint of $\mathcal{L}_{right}$.
- Report all the intervals containing q_x.
- Stop at an interval that does not contain q_x.
- Query the right subtree.

Query (cont'd)

- $q_x > x_{mid}(v)$ $\Longrightarrow$ Search I_{mid} and I_{right}.

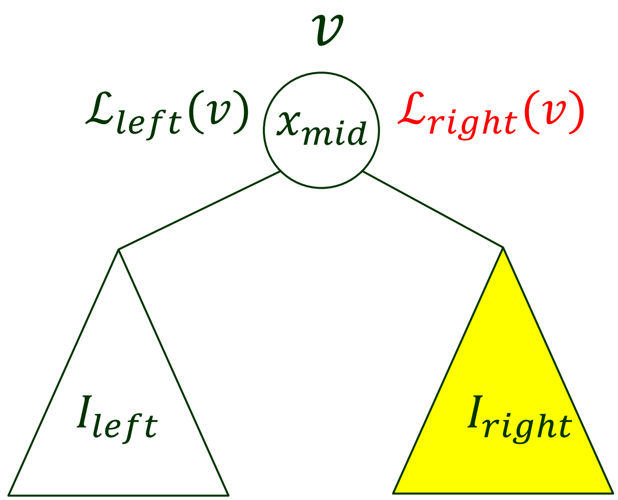


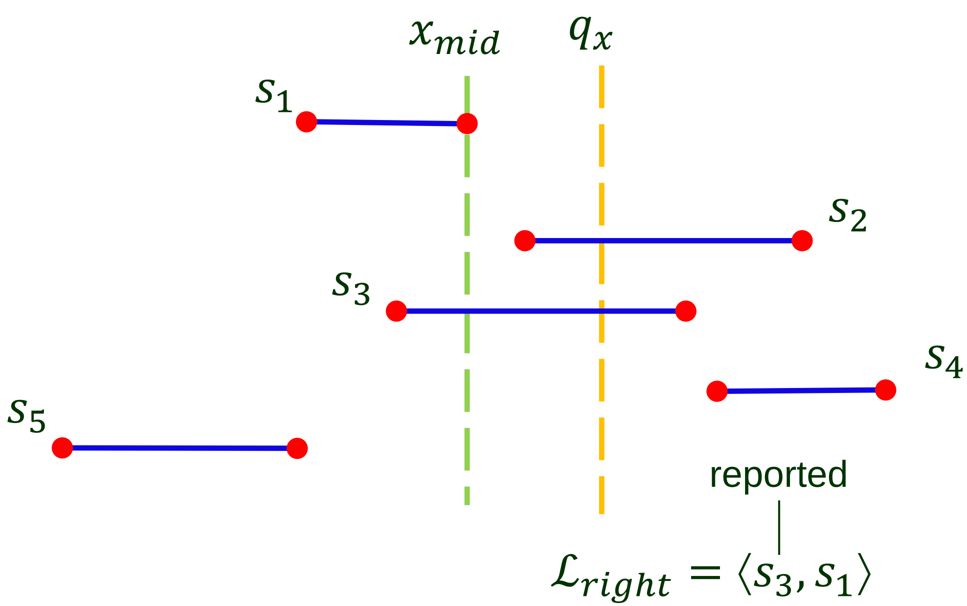


- At v, start at the rightmost endpoint of $\mathcal{L}_{right}$.
- Report all the intervals containing q_x.
- Stop at an interval that does not contain q_x.
- Query the right subtree.

Query (cont'd)

- $q_x > x_{mid}(v)$ $\Longrightarrow$ Search I_{mid} and I_{right}.



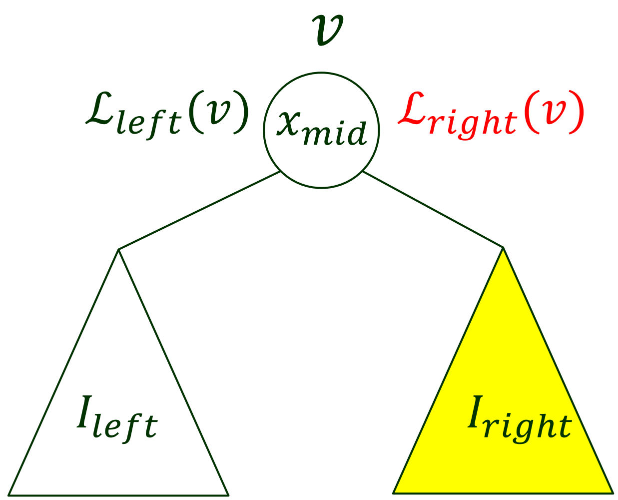


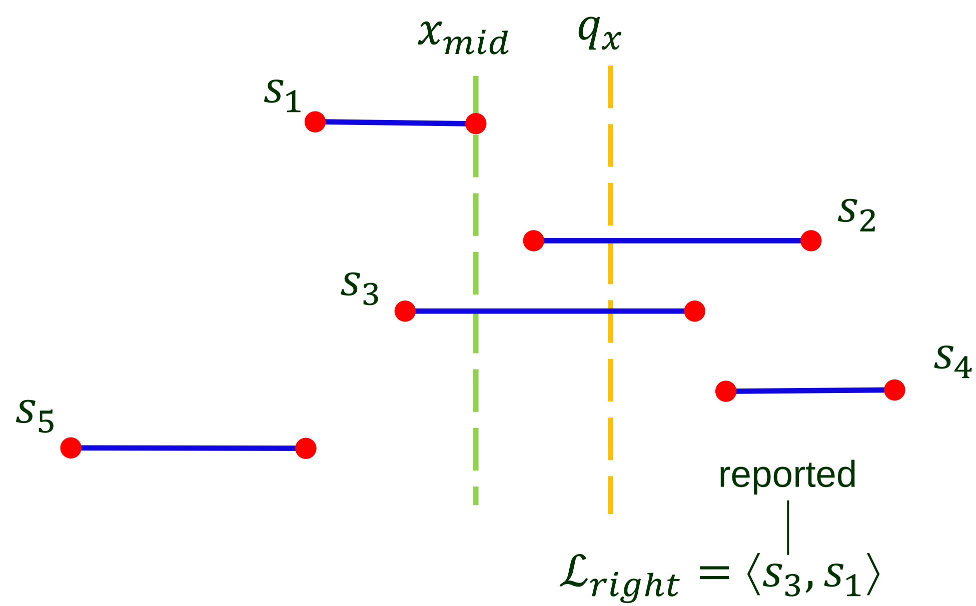
- At v, start at the rightmost endpoint of $\mathcal{L}_{right}$.
- Report all the intervals containing q_x.
- Stop at an interval that does not contain q_x.
- Query the right subtree.

Query time: $O(\log n + k)$

Query (cont'd)

◆ $q_x > x_{mid}(v)$ $\Longrightarrow$ Search I_{mid} and I_{right}.





- At v, start at the rightmost endpoint of $\mathcal{L}_{right}$.
- Report all the intervals containing q_x.
- Stop at an interval that does not contain q_x.
- Query the right subtree.

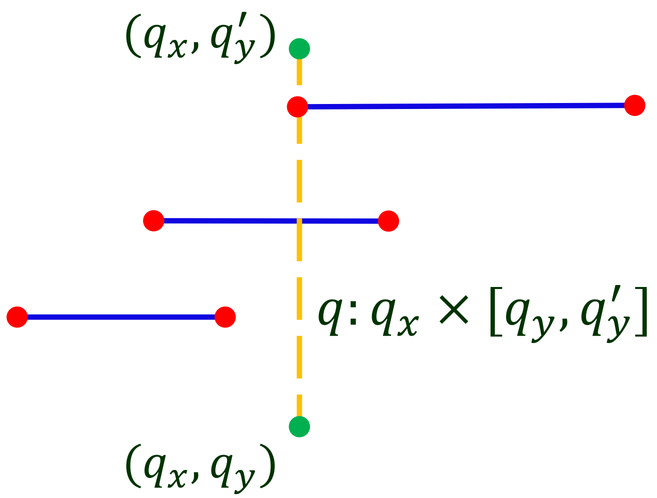
Query time: $O(\log n + k)$

#reported intervals

IV. Query Segment with Finite Length

Query object: a vertical line segment q: $q_x \times [q_y, q'_y]$

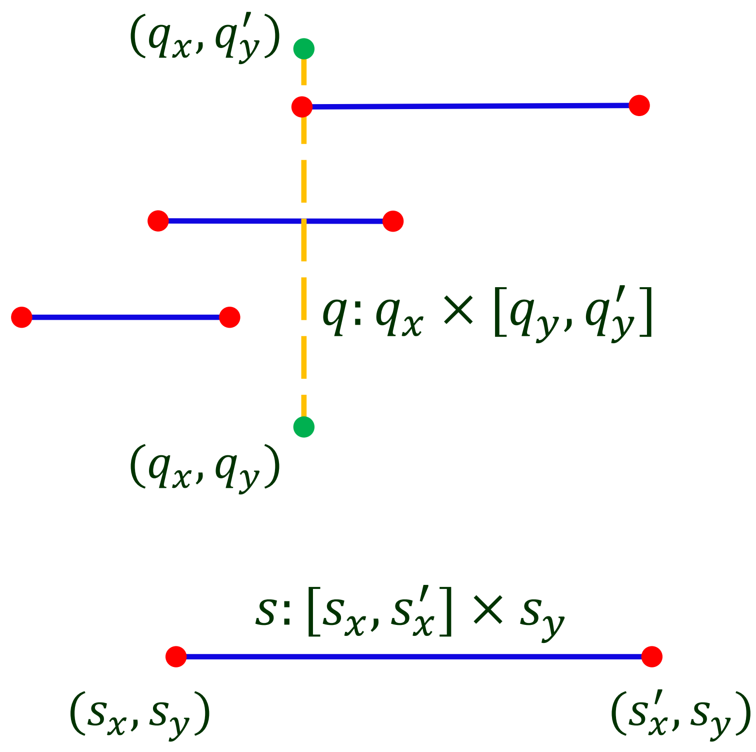
$S = \{s_1, s_2, \dots, s_n\}$: set of n horizontal segments



IV. Query Segment with Finite Length

Query object: a vertical line segment q: $q_x \times [q_y, q'_y]$

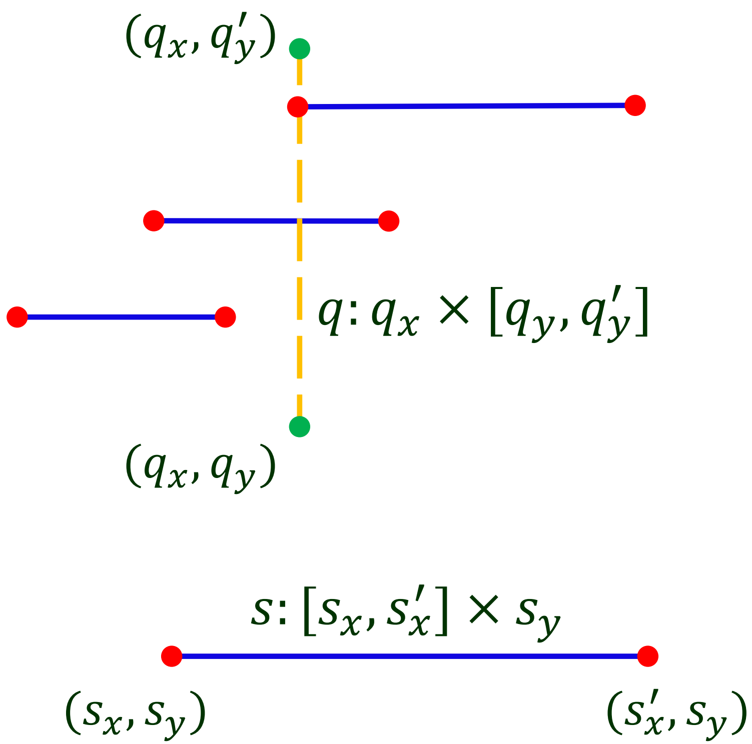
$S = \{s_1, s_2, \ldots, s_n\}$: set of n horizontal segments



IV. Query Segment with Finite Length

Query object: a vertical line segment q: $q_x \times [q_y, q_y']$

$S = \{s_1, s_2, \dots, s_n\}$: set of n horizontal segments



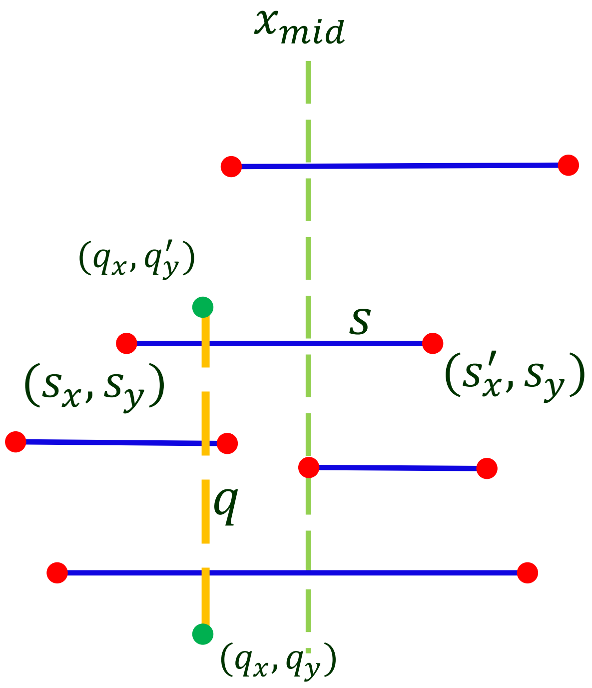
s *intersects* q iff

i) $s_x \le q_x \le s_x'$

ii) $q_y \le s_y \le q_y'$

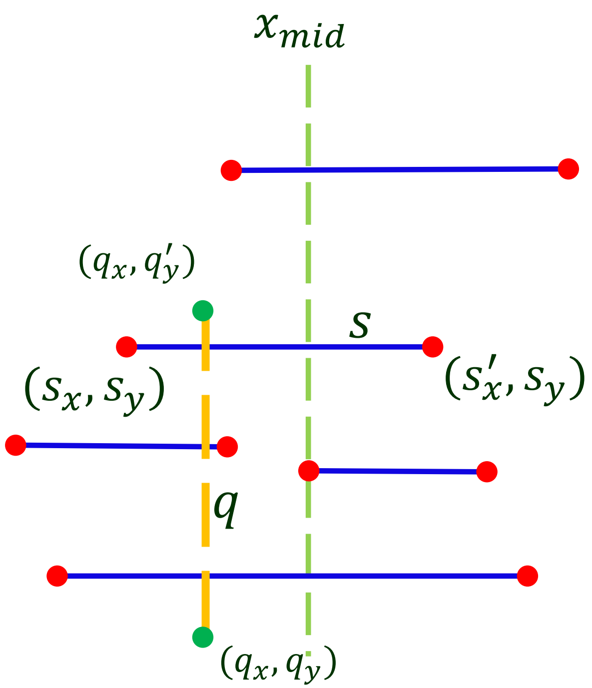
Modified Query

Modify the (earlier described) query with a vertical line object.



Modified Query

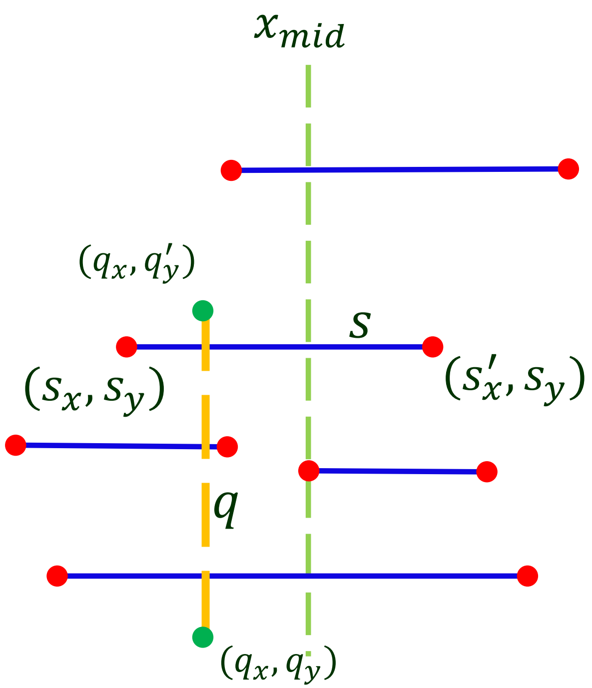
Modify the (earlier described) query with a vertical line object.



- q to the left of x_{mid}, i.e., $q_x \leq x_{mid}$.

Modified Query

Modify the (earlier described) query with a vertical line object.

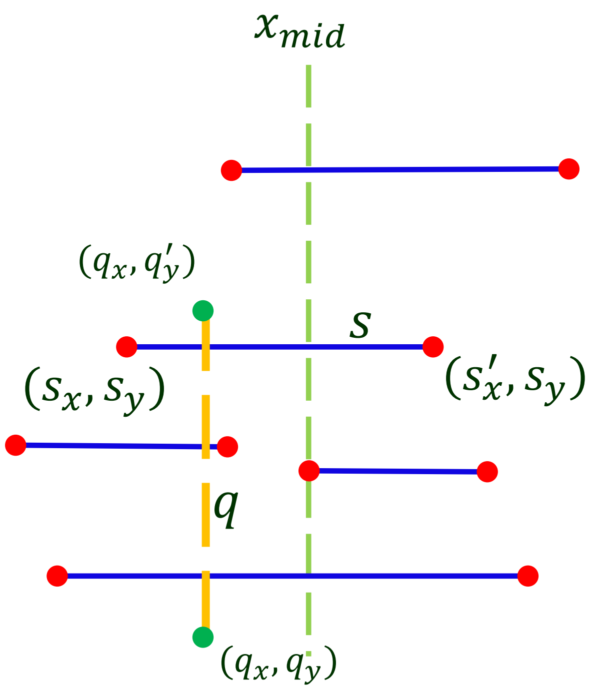


- q to the left of x_{mid}, i.e., $q_x \leq x_{mid}$.
  - Query I_{mid}. We know that every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query iff

$$s_x \leq q_x \leq s'_x \text{ and } q_y \leq s_y \leq q'_y$$

Modified Query

Modify the (earlier described) query with a vertical line object.



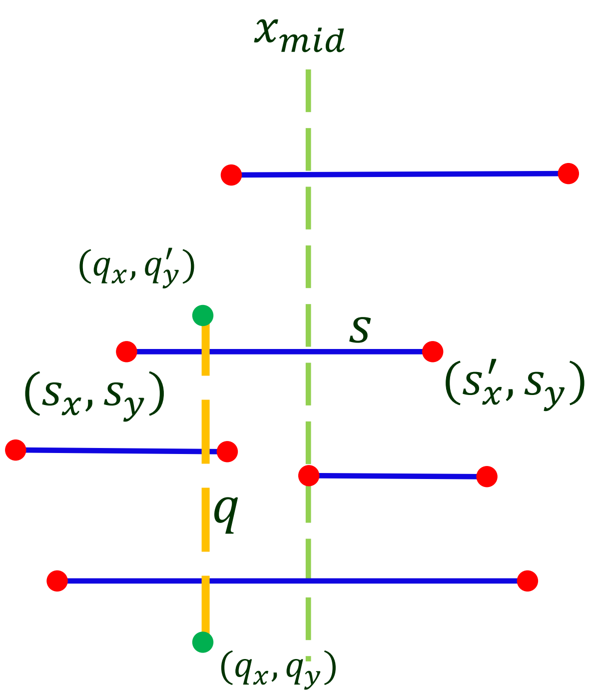
- q to the left of x_{mid}, i.e., $q_x \le x_{mid}$.
  - Query I_{mid}. We know that every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query iff

$$s_x \le q_x \le s'_x \text{ and } q_y \le s_y \le q'_y$$

$$x_{mid} \le s'_x$$

Modified Query

Modify the (earlier described) query with a vertical line object.



- q to the left of x_{mid}, i.e., $q_x \leq x_{mid}$.
  - Query I_{mid}. We know that every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query iff

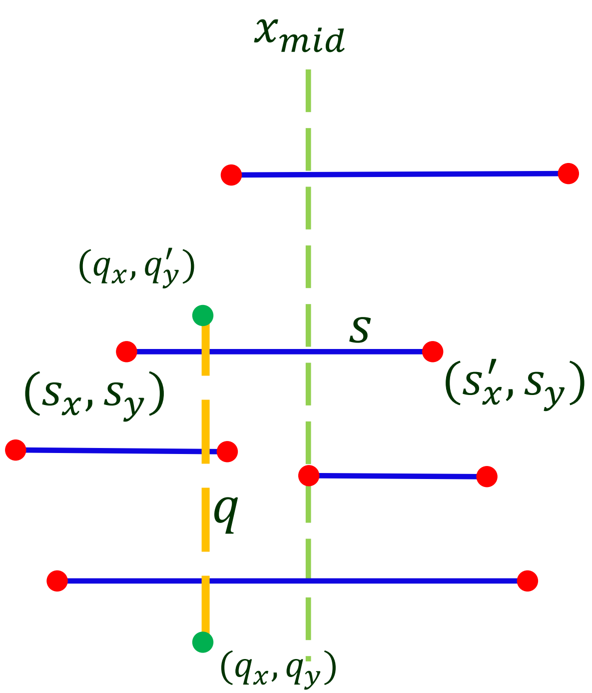
$$s_x \leq q_x \leq s'_x \text{ and } q_y \leq s_y \leq q'_y$$

$x_{mid} \leq s'_x$

$q_x \leq x_{mid}$

Modified Query

Modify the (earlier described) query with a vertical line object.



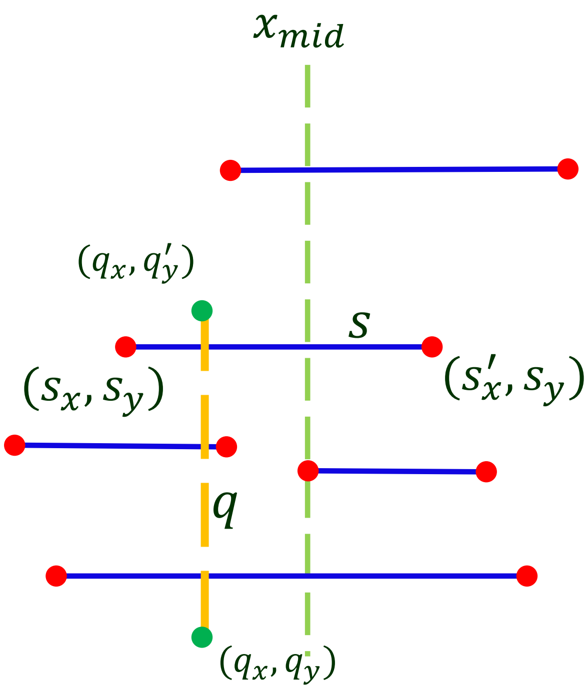
- q to the left of x_{mid}, i.e., $q_x \le x_{mid}$.
  - Query I_{mid}. We know that every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query iff

$$s_x \le \boxed{q_x \le s'_x} \text{ and } q_y \le s_y \le q'_y$$

$$\left.\begin{array}{l} x_{mid} \le s'_x \\ q_x \le x_{mid} \end{array}\right\}$$

Modified Query

Modify the (earlier described) query with a vertical line object.



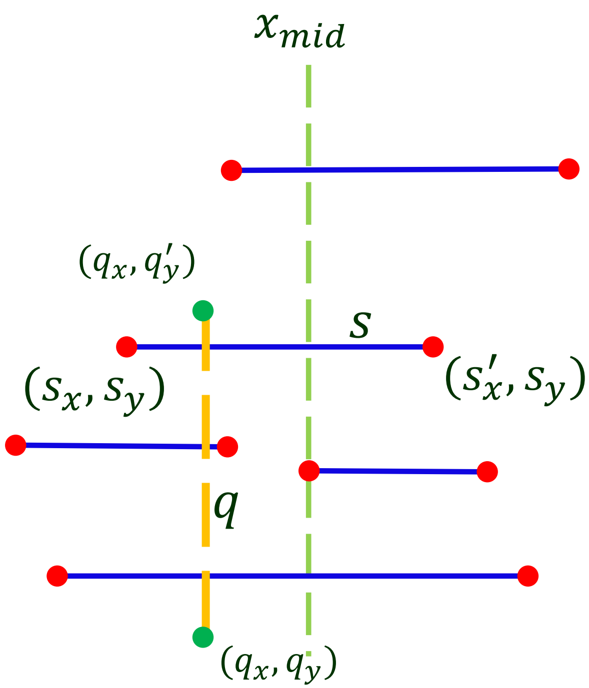
- q to the left of x_{mid}, i.e., $q_x \le x_{mid}$.
  - Query I_{mid}. We know that every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query iff

$$\boxed{s_x \le q_x} \le s'_x \text{ and } \boxed{q_y \le s_y \le q'_y}$$

$$\left.\begin{array}{l} x_{mid} \le s'_x \\ q_x \le x_{mid} \end{array}\right\} \Rightarrow q_x \le s'_x$$

Modified Query

Modify the (earlier described) query with a vertical line object.



- q to the left of x_{mid}, i.e., $q_x \le x_{mid}$.
  - Query I_{mid}. We know that every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query iff

$$\boxed{s_x \le \boxed{q_x}} \le s'_x \text{ and } \boxed{q_y \le s_y \le q'_y}$$

$$\left.\begin{array}{l} x_{mid} \le s'_x \\ q_x \le x_{mid} \end{array}\right\}$$

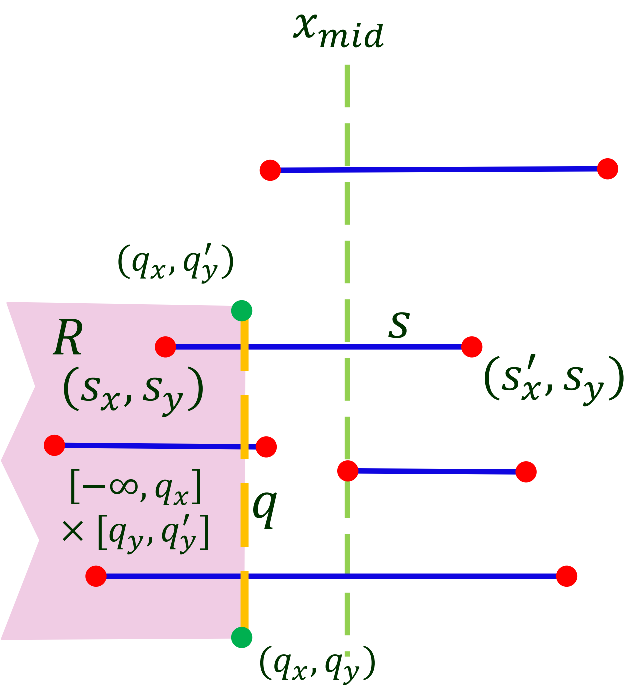
$$\Updownarrow$$

$$(s_x, s_y) \in R\colon [-\infty, q_x] \times [q_y, q'_y]$$

Left endpoint

Modified Query

Modify the (earlier described) query with a vertical line object.



- q to the left of x_{mid}, i.e., $q_x \le x_{mid}$.
  - Query I_{mid}. We know that every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query iff

$$\boxed{s_x \le q_x} \le s'_x \text{ and } \boxed{q_y \le s_y \le q'_y}$$

$$\left.\begin{array}{l} x_{mid} \le s'_x \\ q_x \le x_{mid} \end{array}\right\}$$

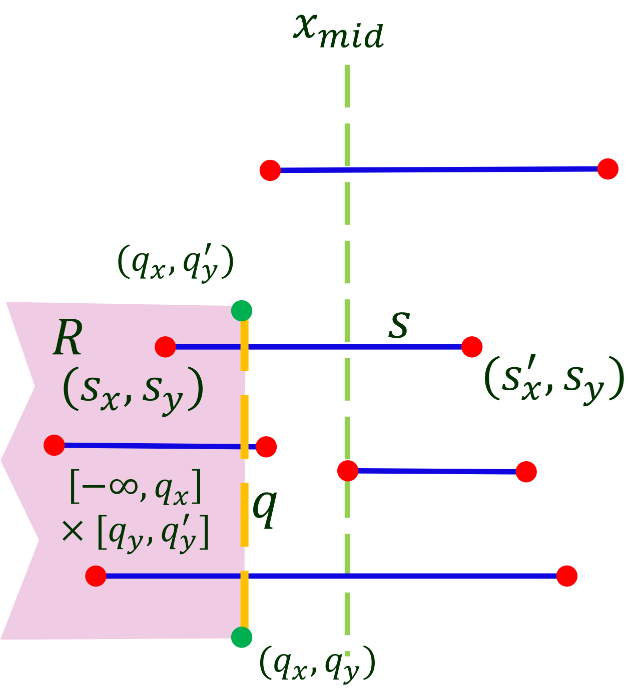
$$\Updownarrow$$

$$(s_x, s_y) \in R: [-\infty, q_x] \times [q_y, q'_y]$$

Left endpoint

Modified Query

Modify the (earlier described) query with a vertical line object.



- q to the left of x_{mid}, i.e., $q_x \le x_{mid}$.
  - Query I_{mid}. We know that every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query iff

$$\boxed{s_x \le \boxed{q_x}} \le s'_x \text{ and } \boxed{q_y \le s_y \le q'_y}$$

$$\left.\begin{array}{l} x_{mid} \le s'_x \\ q_x \le x_{mid} \end{array}\right\} \Rightarrow q_x \le s'_x$$

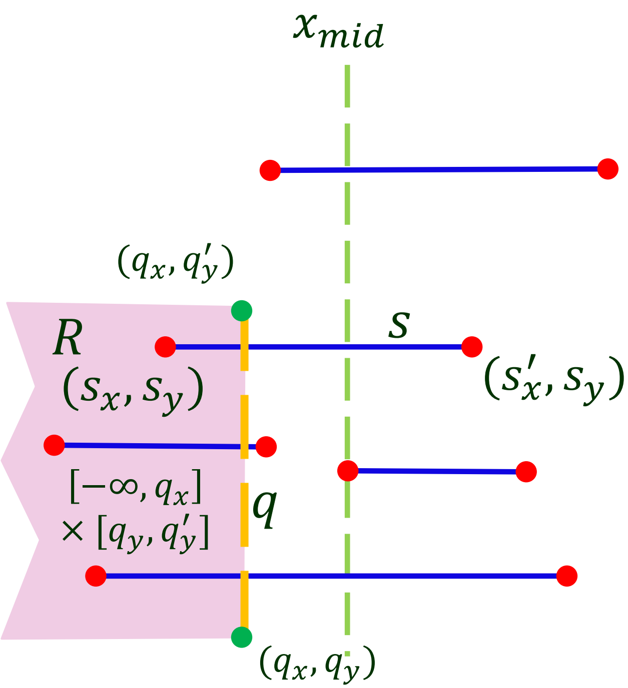
$$\Updownarrow$$

$$\underbrace{\underbrace{(s_x, s_y)}_{\text{Left endpoint}} \in R: [-\infty, q_x] \times [q_y, q'_y]}$$

Rectangular range query – 2D range tree (constructed over left endpoints from I_{mid})

Modified Query

Modify the (earlier described) query with a vertical line object.



- q to the left of x_{mid}, i.e., $q_x \le x_{mid}$.
  - Query I_{mid}. We know that every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query iff

$$\boxed{s_x \le \boxed{q_x}} \le s'_x \text{ and } \boxed{q_y \le s_y \le q'_y}$$

$$\left.\begin{array}{l} x_{mid} \le s'_x \\ q_x \le x_{mid} \end{array}\right\} \Rightarrow q_x \le s'_x$$

$$\Updownarrow$$

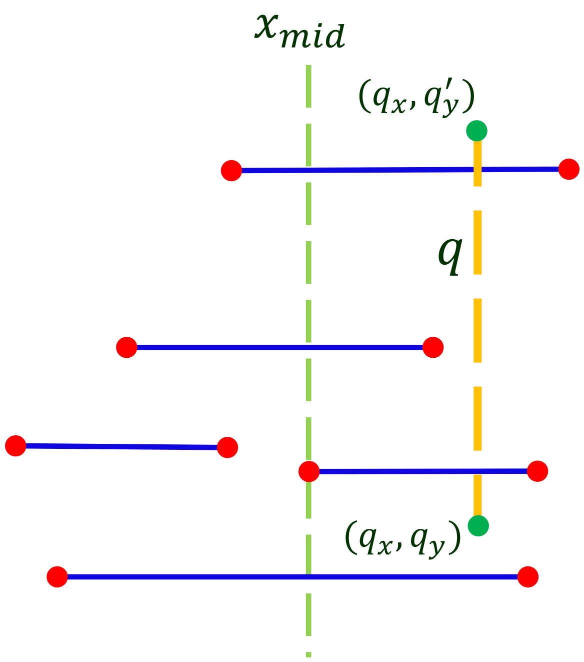
$$\underbrace{\underbrace{(s_x, s_y)}_{\text{Left endpoint}} \in R : [-\infty, q_x] \times [q_y, q'_y]}$$

Rectangular range query – 2D range tree
(constructed over left endpoints from I_{mid})

  - Query the left subtree (storing I_{left}).

Modified Query (cont'd)

- q to the right of x_{mid}, i.e., $q_x > x_{mid}$.

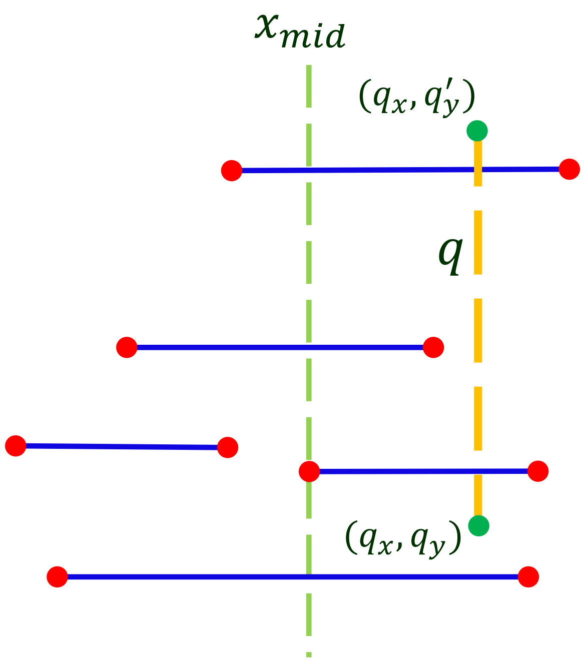


Modified Query (cont'd)

- q to the right of x_{mid}, i.e., $q_x > x_{mid}$.

  - Query I_{mid}. Every segment $s \in I_{mid}$ satisfies the query iff

$$s_x \le q_x \le s'_x \text{ and } q_y \le s_y \le q'_y$$

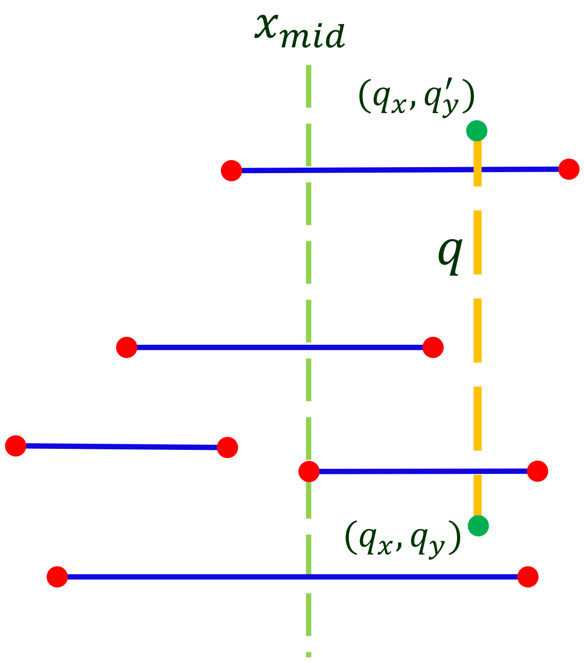


Modified Query (cont'd)

- q to the right of x_{mid}, i.e., $q_x > x_{mid}$.
  - Query I_{mid}. Every segment $s \in I_{mid}$ satisfies the query iff

$$s_x \leq q_x \leq s'_x \text{ and } q_y \leq s_y \leq q'_y$$

$$s_x \leq x_{mid}$$

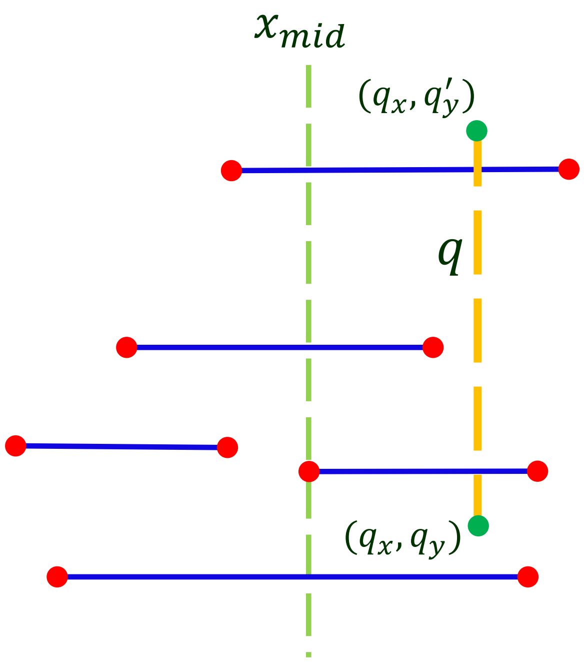


Modified Query (cont'd)

- q to the right of x_{mid}, i.e., $q_x > x_{mid}$.

  - Query I_{mid}. Every segment $s \in I_{mid}$ satisfies the query iff

$$s_x \le q_x \le s'_x \text{ and } q_y \le s_y \le q'_y$$

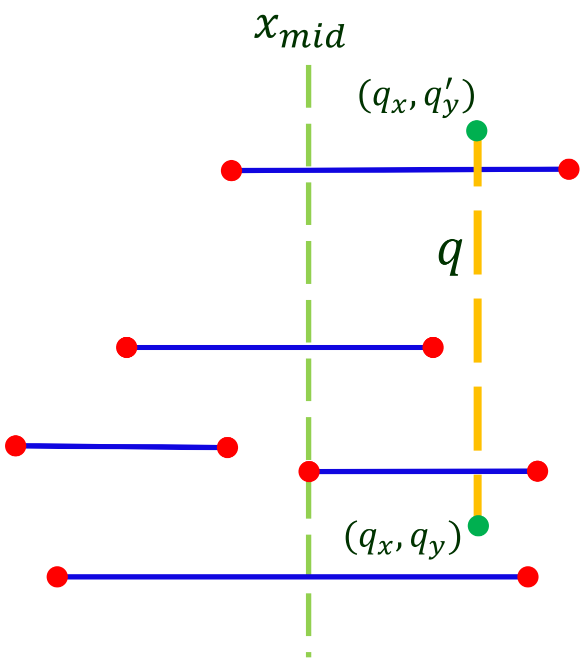


$$s_x \le x_{mid}$$
$$x_{mid} < q_x$$

Modified Query (cont'd)

- q to the right of x_{mid}, i.e., $q_x > x_{mid}$.

  - Query I_{mid}. Every segment $s \in I_{mid}$ satisfies the query iff

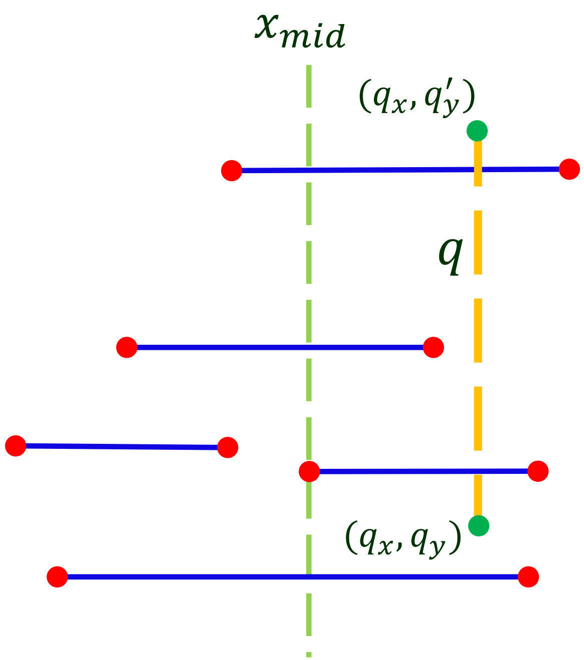


$$\boxed{s_x \le q_x} \le s'_x \text{ and } q_y \le s_y \le q'_y$$

$$\left.\begin{array}{l} s_x \le x_{mid} \\ x_{mid} < q_x \end{array}\right\} \Rightarrow s_x \le q_x$$

Modified Query (cont'd)

- q to the right of x_{mid}, i.e., $q_x > x_{mid}$.
  - Query I_{mid}. Every segment $s \in I_{mid}$ satisfies the query iff



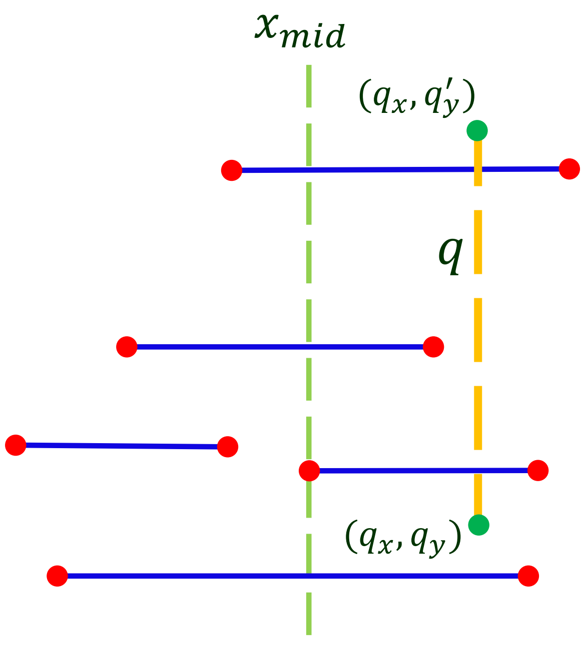
$$\boxed{s_x \le q_x} \le s'_x \text{ and } \boxed{q_y \le s_y \le q'_y}$$

$$\left.\begin{array}{l} s_x \le x_{mid} \\ x_{mid} < q_x \end{array}\right\}$$

Modified Query (cont'd)

- q to the right of x_{mid}, i.e., $q_x > x_{mid}$.

  - Query I_{mid}. Every segment $s \in I_{mid}$ satisfies the query iff



$$s_x \le q_x \le s'_x \text{ and } q_y \le s_y \le q'_y$$

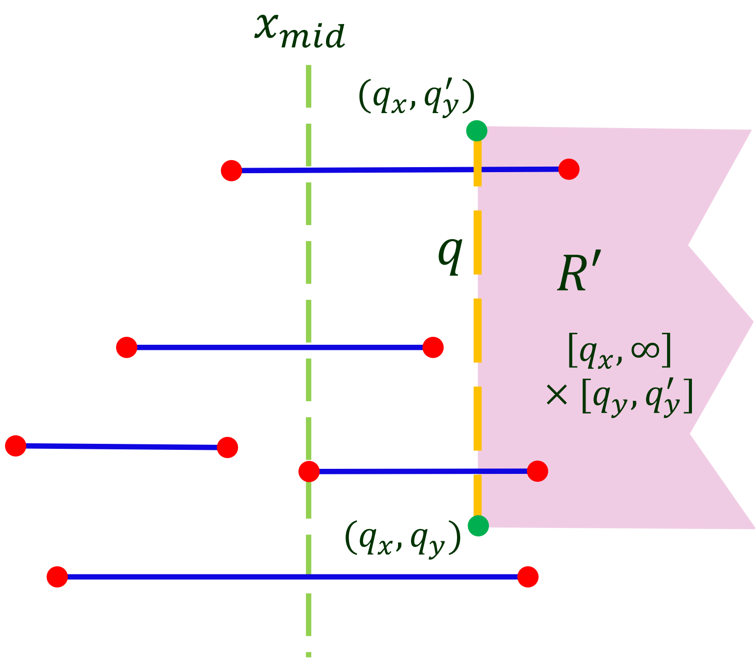
$$\left.\begin{array}{l} s_x \le x_{mid} \\ x_{mid} < q_x \end{array}\right\}$$

$$(s'_x, s_y) \in R' : [q_x, \infty] \times [q_y, q'_y]$$

Right endpoint

Modified Query (cont'd)

- q to the right of x_{mid}, i.e., $q_x > x_{mid}$.



- Query I_{mid}. Every segment $s \in I_{mid}$ satisfies the query iff

$$\left.\begin{array}{l} s_x \le x_{mid} \\ x_{mid} < q_x \end{array}\right\} \Rightarrow s_x \le q_x$$

$$s_x \le q_x \le s'_x \text{ and } q_y \le s_y \le q'_y$$

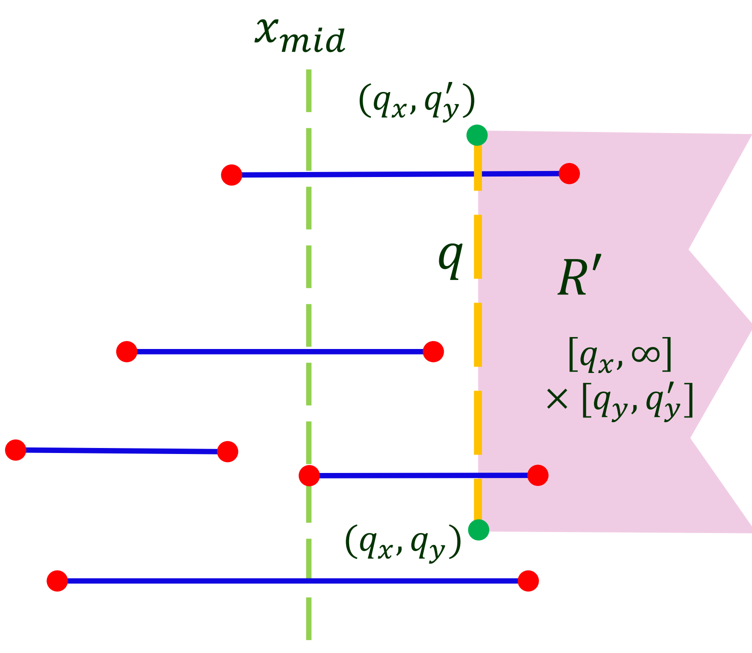
$$\Updownarrow$$

$$(s'_x, s_y) \in R' : [q_x, \infty] \times [q_y, q'_y]$$

Right endpoint

Modified Query (cont'd)

- q to the right of x_{mid}, i.e., $q_x > x_{mid}$.



- Query I_{mid}. Every segment $s \in I_{mid}$ satisfies the query iff

$$\boxed{s_x \le q_x \le s'_x} \text{ and } \boxed{q_y \le s_y \le q'_y}$$

$$\left.\begin{array}{l} s_x \le x_{mid} \\ x_{mid} < q_x \end{array}\right\} \Rightarrow s_x \le q_x$$

$$\Updownarrow$$

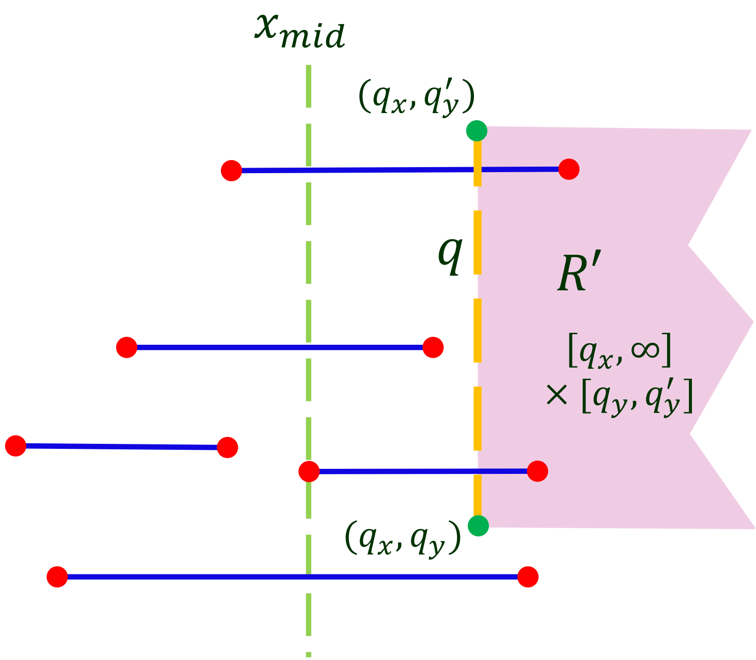
$$(s'_x, s_y) \in R' : [q_x, \infty] \times [q_y, q'_y]$$

Right endpoint

Rectangular range query – 2D range tree (constructed over right endpoints from I_{mid})

Modified Query (cont'd)

- q to the right of x_{mid}, i.e., $q_x > x_{mid}$.



- Query I_{mid}. Every segment $s \in I_{mid}$ satisfies the query iff

$$\boxed{s_x \le q_x} \le s'_x \text{ and } \boxed{q_y \le s_y \le q'_y}$$

$$\left.\begin{array}{l} s_x \le x_{mid} \\ x_{mid} < q_x \end{array}\right\} \Rightarrow s_x \le q_x$$

$$q_x \le s'_x \text{ and } q_y \le s_y \le q'_y \iff (s'_x, s_y) \in R' : [q_x, \infty] \times [q_y, q'_y]$$

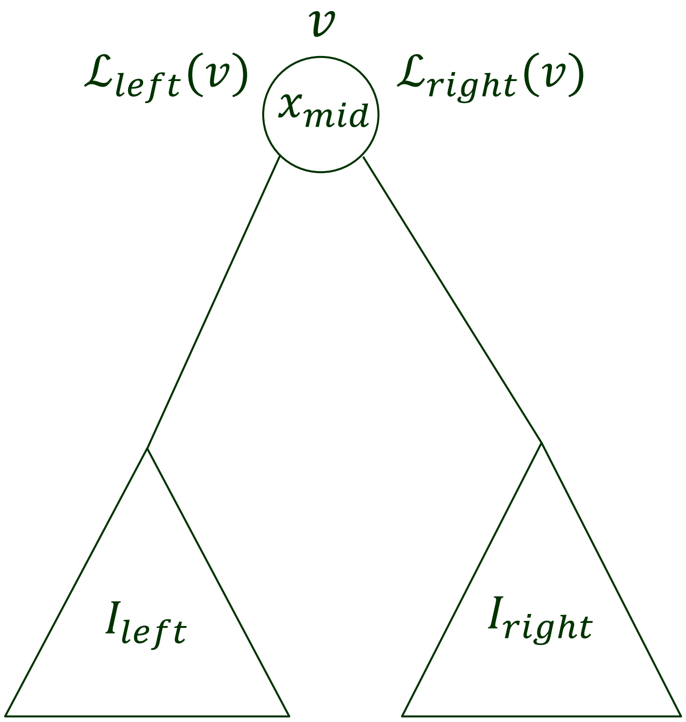
Right endpoint

Rectangular range query – 2D range tree (constructed over right endpoints from I_{mid})

- Query the right subtree (storing I_{right}).

Data Structure

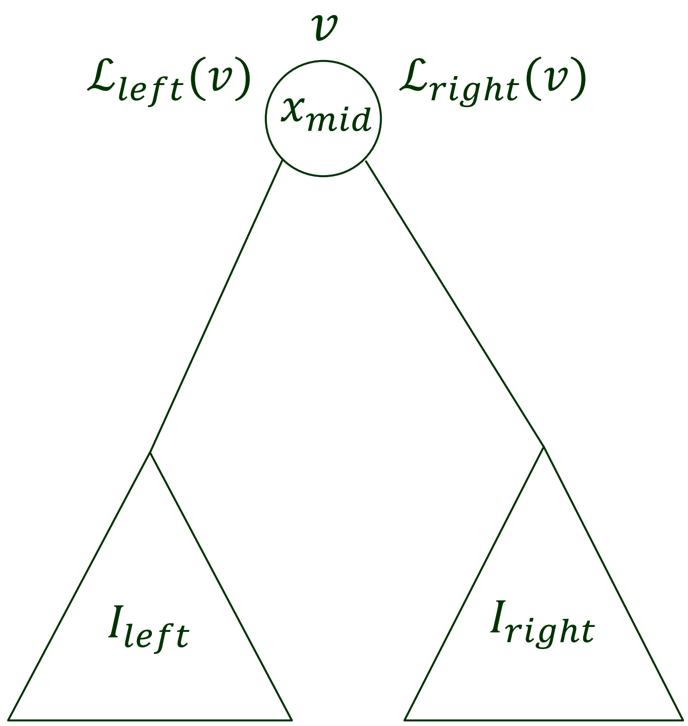
Main structure: interval tree $\mathcal{T}$ on x-coordinates.



At each node v, replace

Data Structure

Main structure: interval tree $\mathcal{T}$ on x-coordinates.

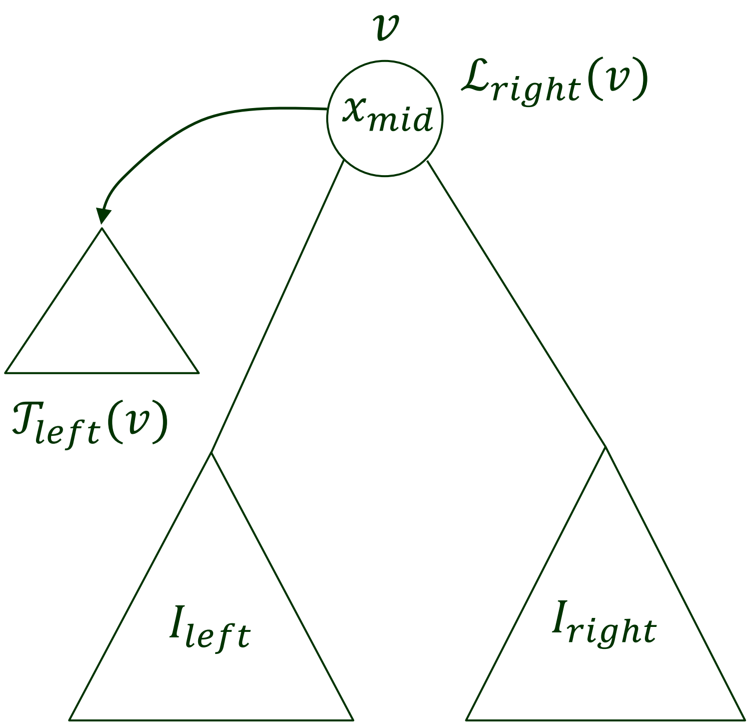


At each node v, replace

- $\mathcal{L}_{left}(v)$ with a range tree $\mathcal{T}_{left}(v)$ on the left endpoints of $I_{mid}(v)$.

Data Structure

Main structure: interval tree $\mathcal{T}$ on x-coordinates.

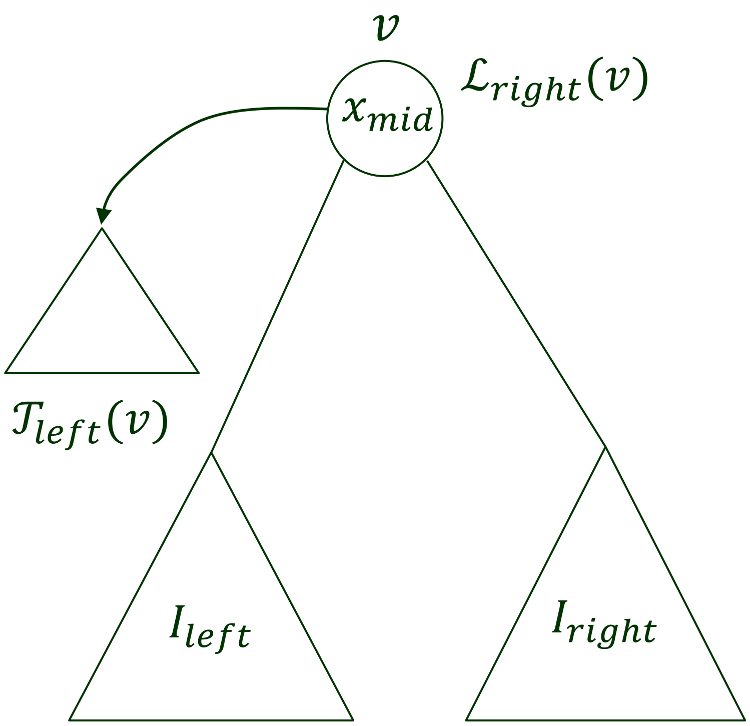


At each node v, replace

- $\mathcal{L}_{left}(v)$ with a range tree $\mathcal{T}_{left}(v)$ on the left endpoints of $I_{mid}(v)$.

Data Structure

Main structure: interval tree $\mathcal{T}$ on x-coordinates.

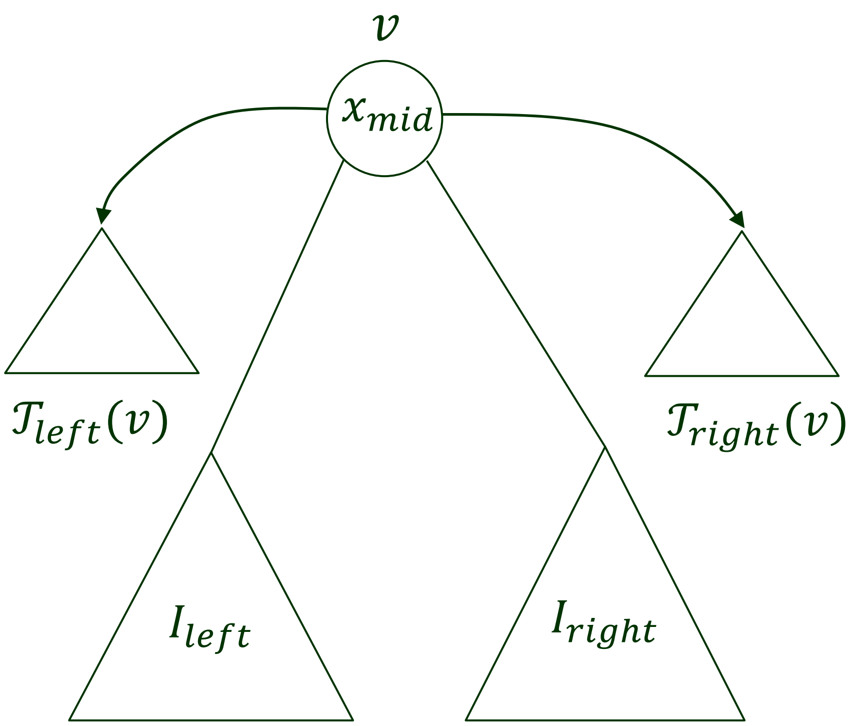


At each node v, replace

- $\mathcal{L}_{left}(v)$ with a range tree $\mathcal{T}_{left}(v)$ on the left endpoints of $I_{mid}(v)$.
- $\mathcal{L}_{right}(v)$ with a range tree $\mathcal{T}_{right}(v)$ on the right endpoints of $I_{mid}(v)$.

Data Structure

Main structure: interval tree $\mathcal{T}$ on x-coordinates.



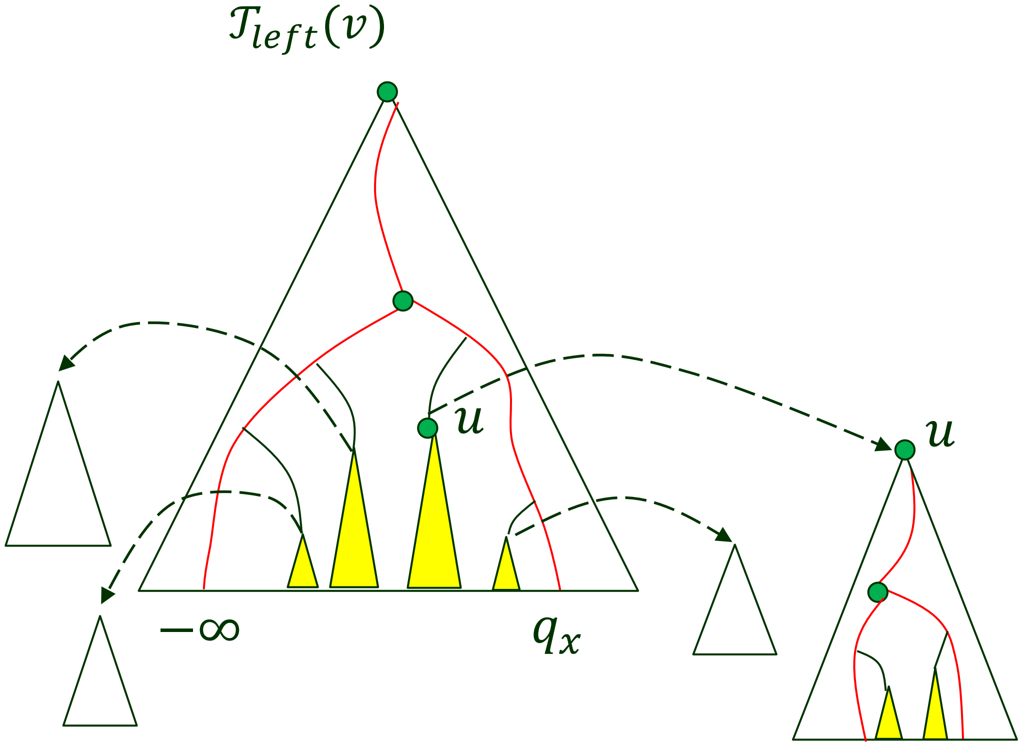
At each node v, replace

- $\mathcal{L}_{left}(v)$ with a range tree $\mathcal{T}_{left}(v)$ on the left endpoints of $I_{mid}(v)$.

- $\mathcal{L}_{right}(v)$ with a range tree $\mathcal{T}_{right}(v)$ on the right endpoints of $I_{mid}(v)$.

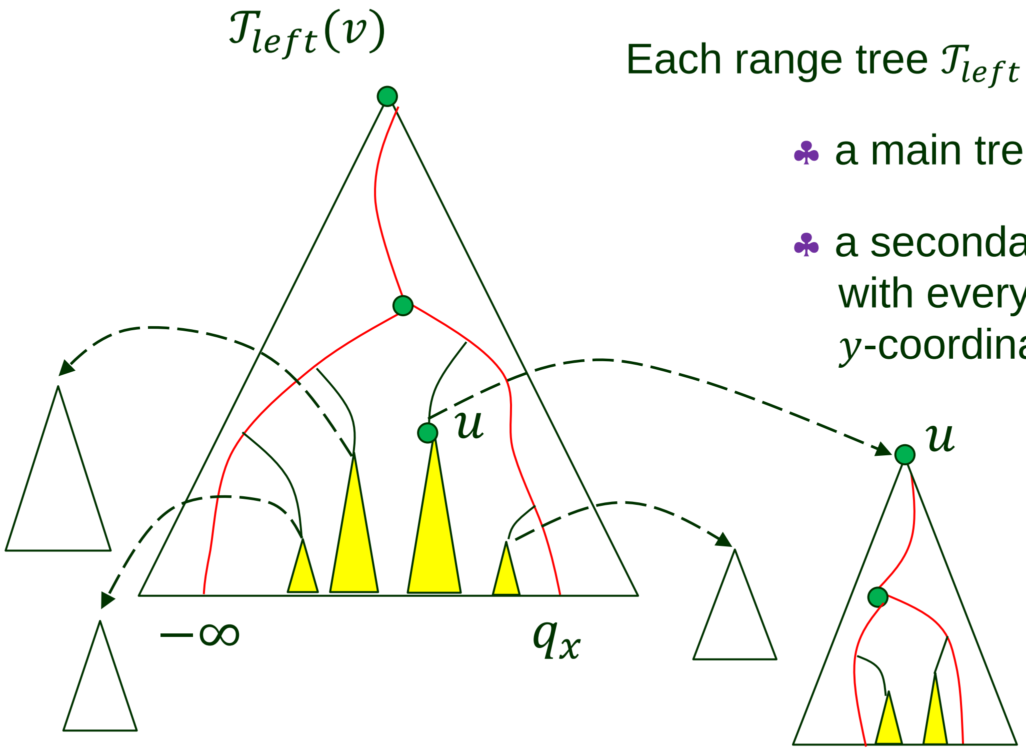
Range Queries

- Query with $[-\infty, q_x] \times [q_y, q'_y]$ performed on $\mathcal{T}_{left}(v)$.



Range Queries

- Query with $[-\infty, q_x] \times [q_y, q_y']$ performed on $\mathcal{T}_{left}(v)$.

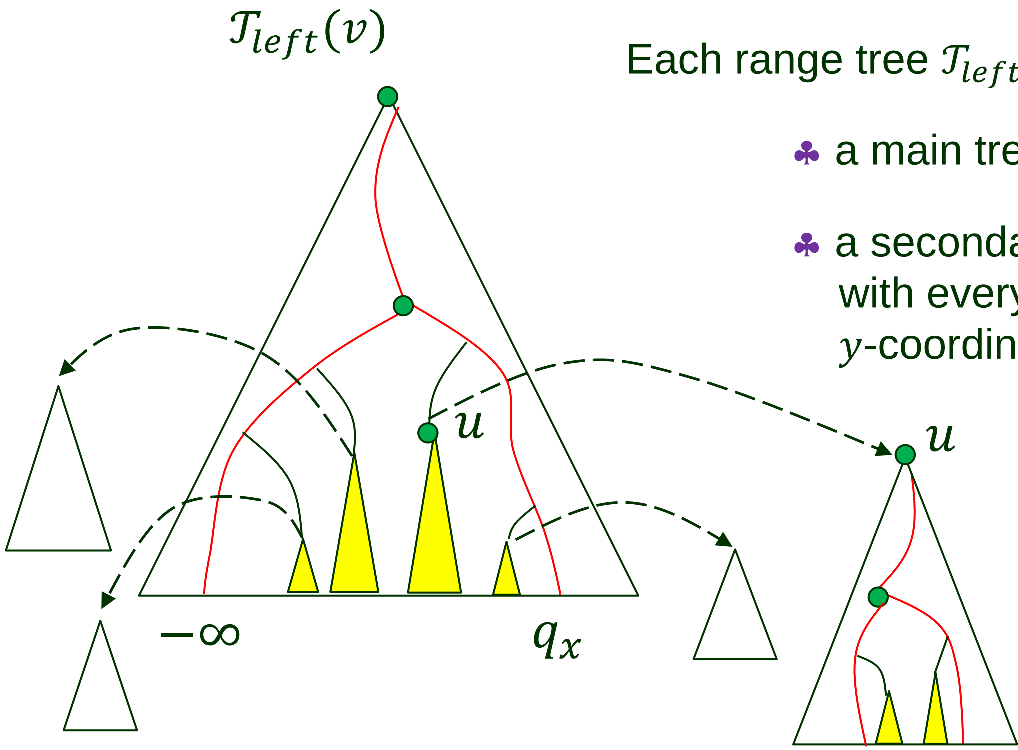


Each range tree $\mathcal{T}_{left}(v)$ (or $\mathcal{T}_{right}(v)$) has

- a main tree on the x-coordinate
- a secondary structure associated with every internal node on the y-coordinate.

Range Queries

- Query with $[-\infty, q_x] \times [q_y, q_y']$ performed on $\mathcal{T}_{left}(v)$.
- Query with $[q_x, \infty] \times [q_y, q_y']$ performed on $\mathcal{T}_{right}(v)$.



Each range tree $\mathcal{T}_{left}(v)$ (or $\mathcal{T}_{right}(v)$) has

- a main tree on the x-coordinate
- a secondary structure associated with every internal node on the y-coordinate.

Query Time

Perform a query on the range tree $\mathcal{T}_{left}(v)$ instead of traversing the sorted list $\mathcal{L}_{left}(v)$.

Query Time

Perform a query on the range tree $\mathcal{T}_{left}(v)$ instead of traversing the sorted list $\mathcal{L}_{left}(v)$.

- $O(\log n)$ nodes on the search path in the interval tree.

Query Time

Perform a query on the range tree $\mathcal{T}_{left}(v)$ instead of traversing the sorted list $\mathcal{L}_{left}(v)$.

- $O(\log n)$ nodes on the search path in the interval tree.
- at each such node v, $O(\log^2 n + k_v)$ time spent on the range tree $\mathcal{T}_{left}(v)$ or $\mathcal{T}_{right}(v)$.

Query Time

Perform a query on the range tree $\mathcal{T}_{left}(v)$ instead of traversing the sorted list $\mathcal{L}_{left}(v)$.

- $O(\log n)$ nodes on the search path in the interval tree.
- at each such node v, $O(\log^2 n + k_v)$ time spent on the range tree $\mathcal{T}_{left}(v)$ or $\mathcal{T}_{right}(v)$.

reported segments
in the range tree

Query Time

Perform a query on the range tree $\mathcal{T}_{left}(v)$ instead of traversing the sorted list $\mathcal{L}_{left}(v)$.

- $O(\log n)$ nodes on the search path in the interval tree.
- at each such node v, $O(\log^2 n + k_v)$ time spent on the range tree $\mathcal{T}_{left}(v)$ or $\mathcal{T}_{right}(v)$.

reported segments
in the range tree

Total query time: $O(\log^3 n + k_v \log n)$

Query Time

Perform a query on the range tree $\mathcal{T}_{left}(v)$ instead of traversing the sorted list $\mathcal{L}_{left}(v)$.

- $O(\log n)$ nodes on the search path in the interval tree.
- at each such node v, $O(\log^2 n + k_v)$ time spent on the range tree $\mathcal{T}_{left}(v)$ or $\mathcal{T}_{right}(v)$.

\# reported segments in the range tree

Total query time: $O(\log^3 n + k_v \log n)$

reduced to (using fractional cascading)

$O(\log^2 n + k_v)$

Query Time

Perform a query on the range tree $\mathcal{T}_{left}(v)$ instead of traversing the sorted list $\mathcal{L}_{left}(v)$.

- $O(\log n)$ nodes on the search path in the interval tree.
- at each such node v, $O(\log^2 n + k_v)$ time spent on the range tree $\mathcal{T}_{left}(v)$ or $\mathcal{T}_{right}(v)$.

reported segments
in the range tree

Total query time: $O(\log^3 n + k_v \log n)$

reduced to (using
fractional cascading)

$O(\log^2 n + k_v)$

Storage: $O(n \log n)$

Query Time

Perform a query on the range tree $\mathcal{T}_{left}(v)$ instead of traversing the sorted list $\mathcal{L}_{left}(v)$.

- $O(\log n)$ nodes on the search path in the interval tree.
- at each such node v, $O(\log^2 n + k_v)$ time spent on the range tree $\mathcal{T}_{left}(v)$ or $\mathcal{T}_{right}(v)$.

reported segments
in the range tree

Total query time: $O(\log^3 n + k_v \log n)$

reduced to (using
fractional cascading)

$O(\log^2 n + k_v)$

Storage: $O(n \log n)$

Construction time: $O(n \log n)$