Julius-Maximilians-
UNIVERSITÄT WÜRZBURG

Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems

Institute for Informatics

Computational Geometry

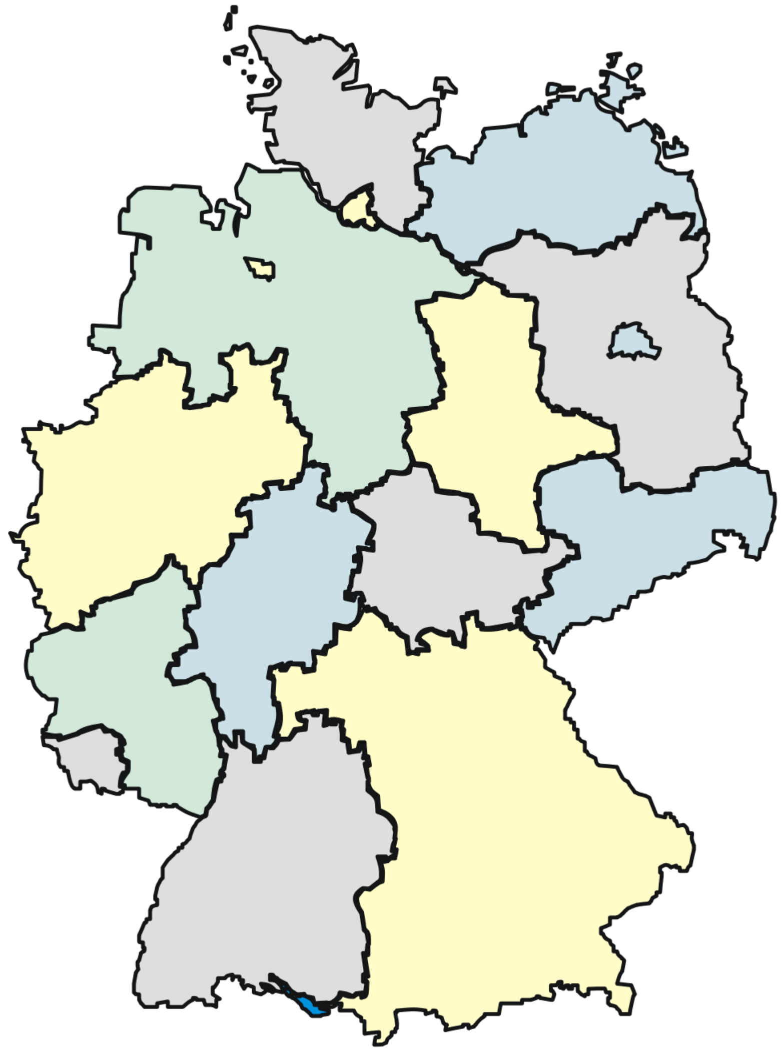
Point Localization
or
Where am I?

Lecture #5

Thomas van Dijk

Winter Semester 2019/20

What's the Problem?

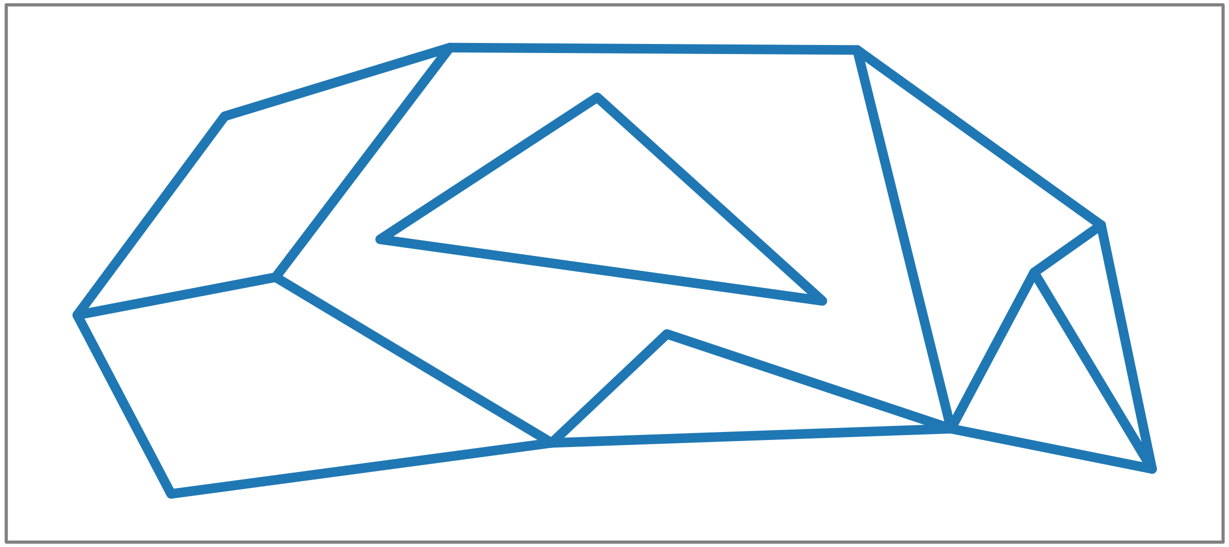


[Stefan-Xp, CC BY-SA 3.0, via wikipedia]

What's the Problem?

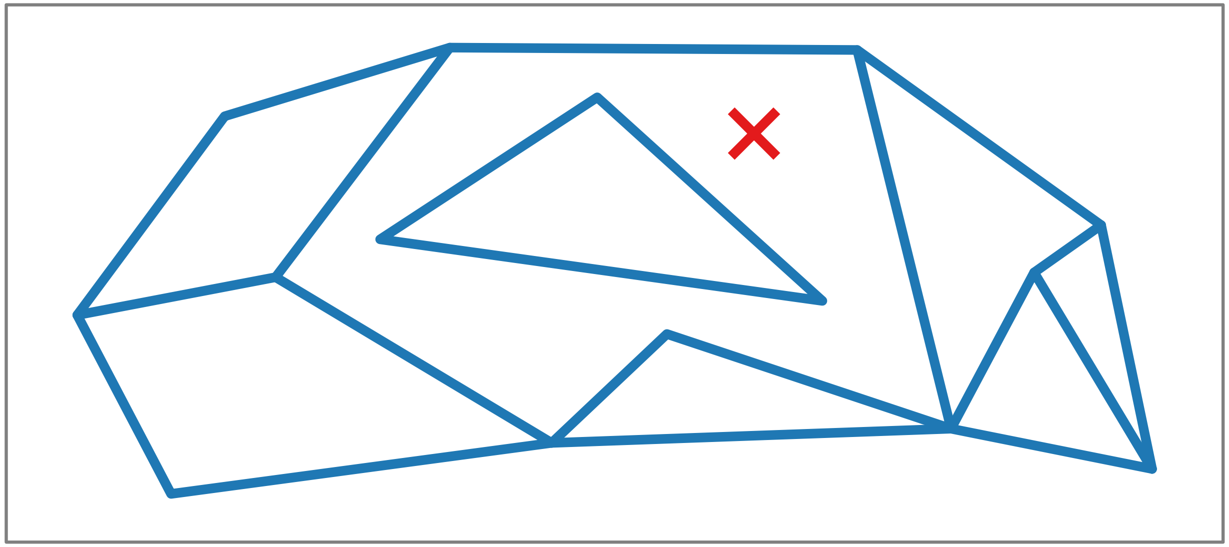
[Stefan-Xp, CC BY-SA 3.0, via wikipedia]

What's the Problem?



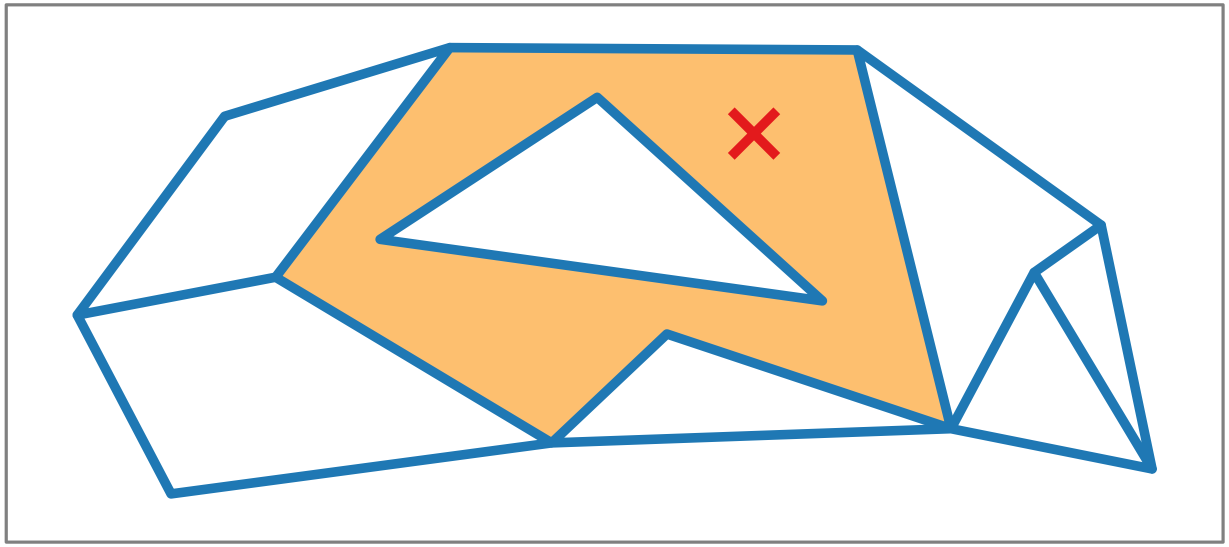
Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

What's the Problem?



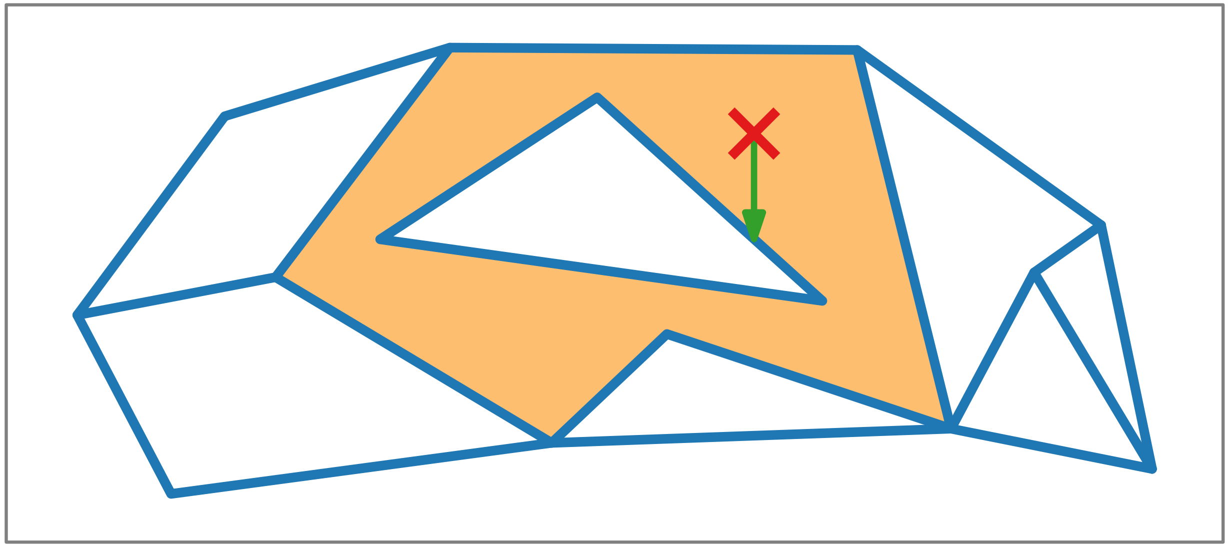
Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

What's the Problem?



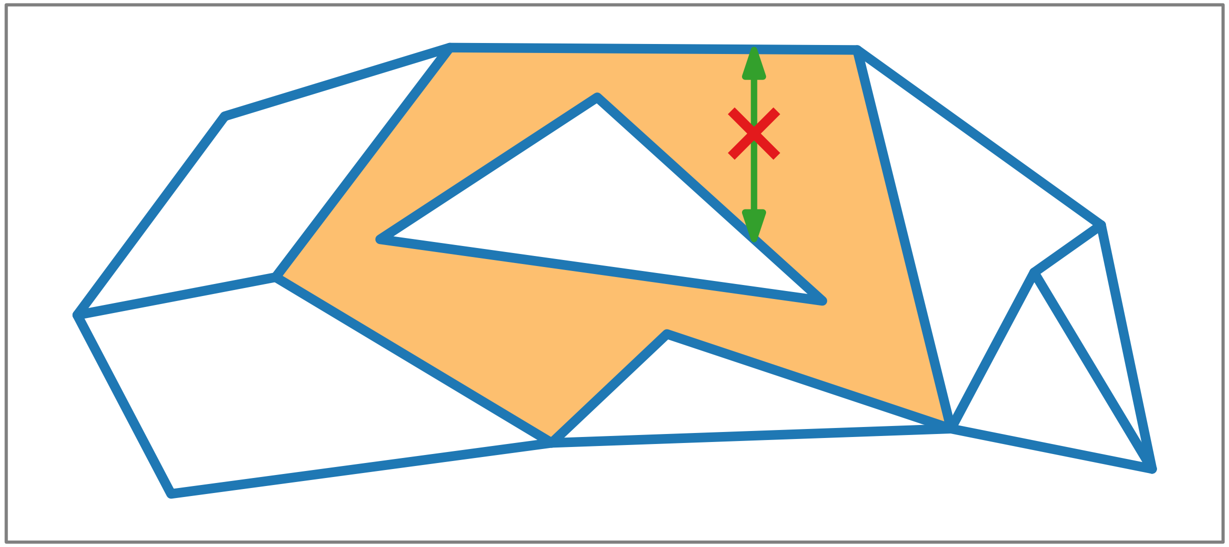
Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

What's the Problem?



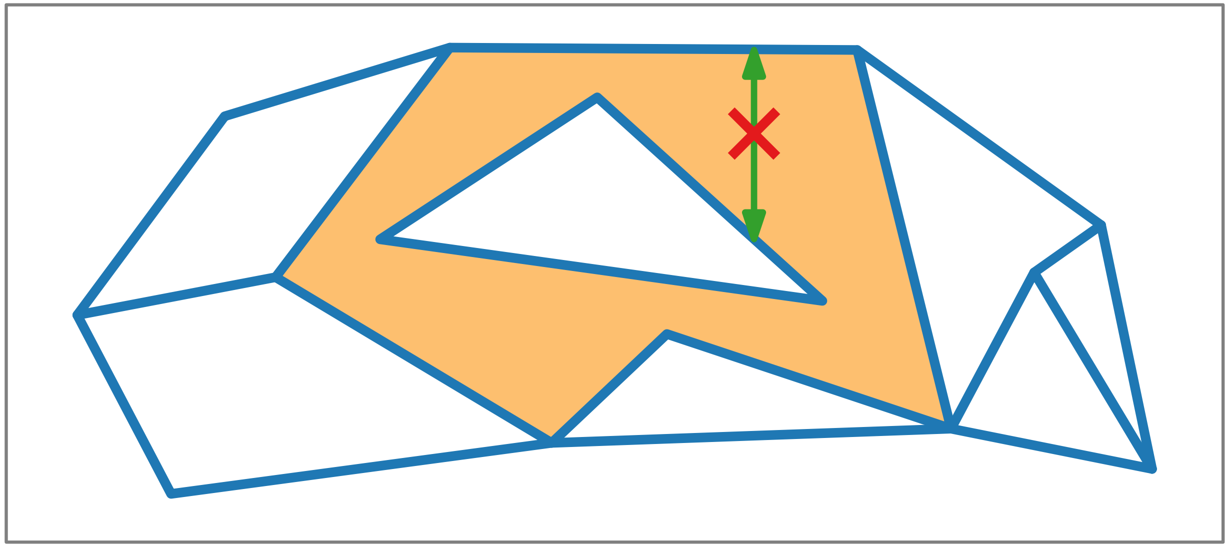
Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

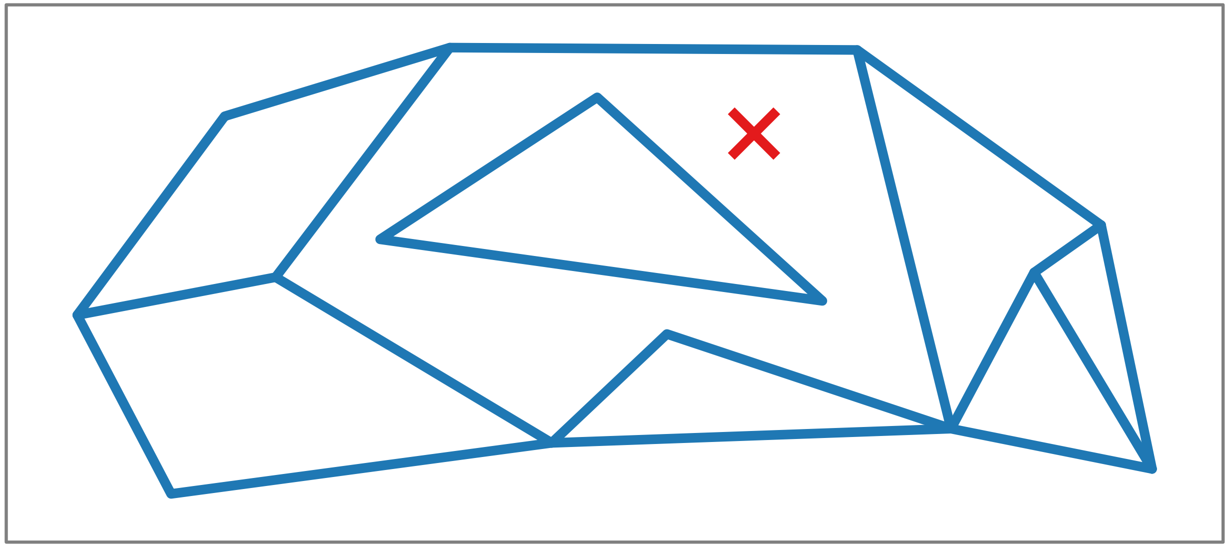
What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

[2 min]

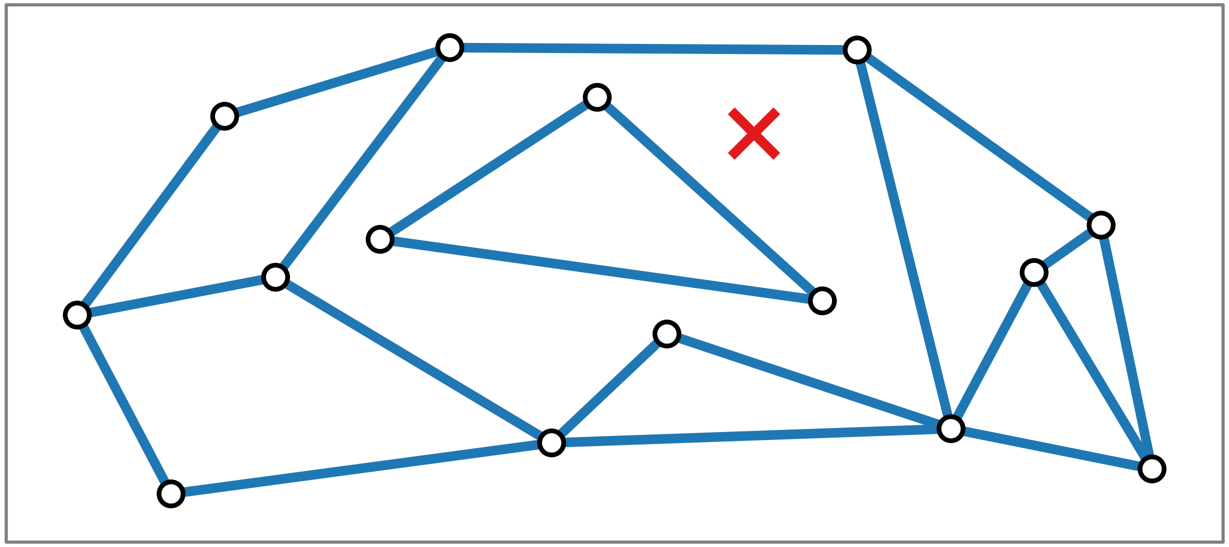
What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

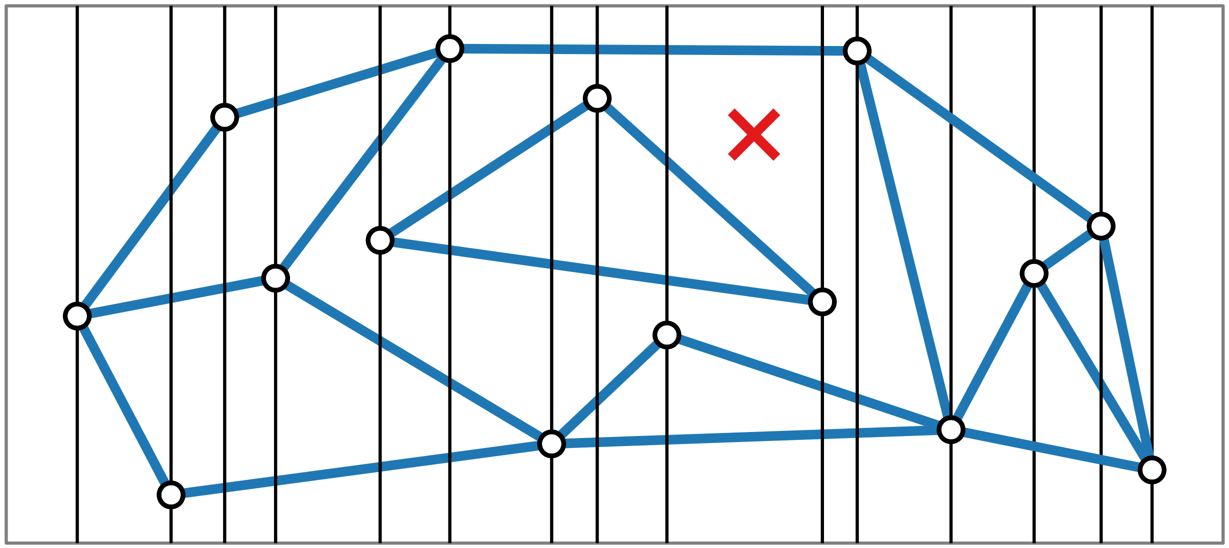
What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

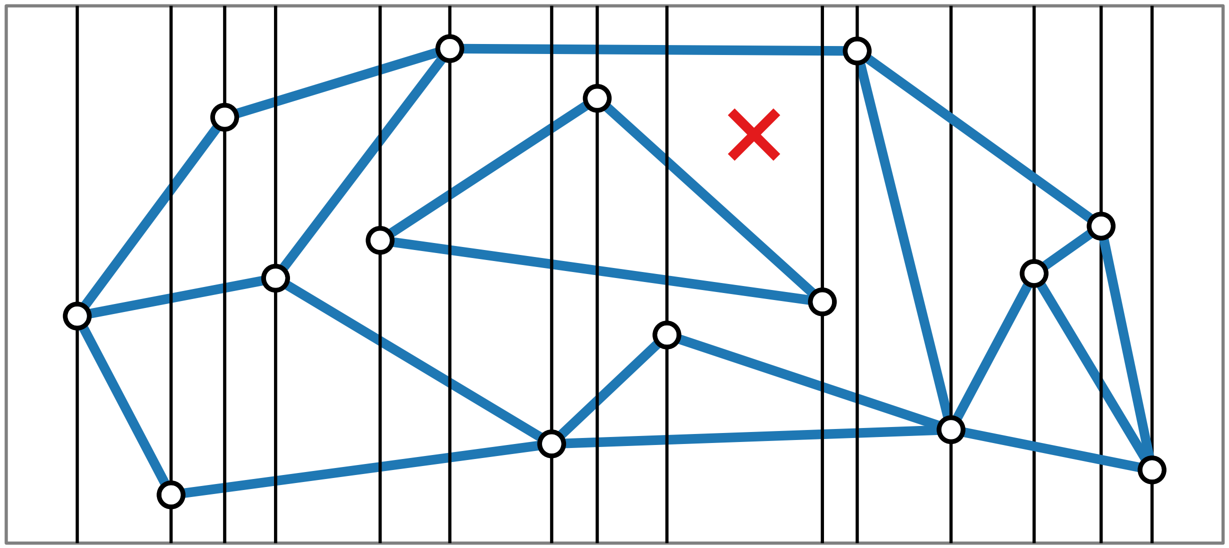
What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

What's the Problem?

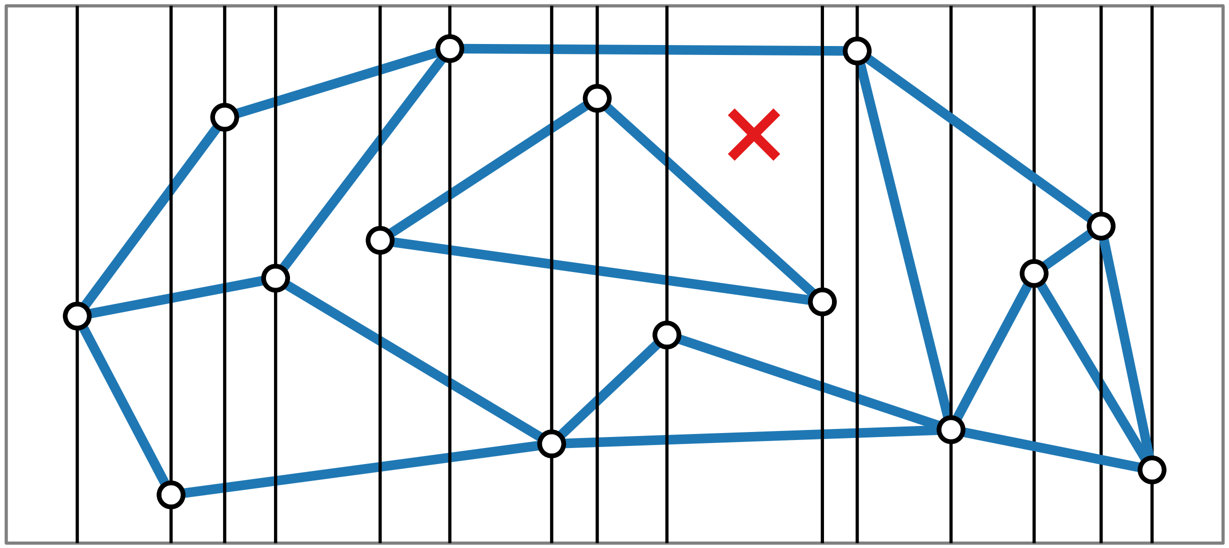


Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

Query:

What's the Problem?

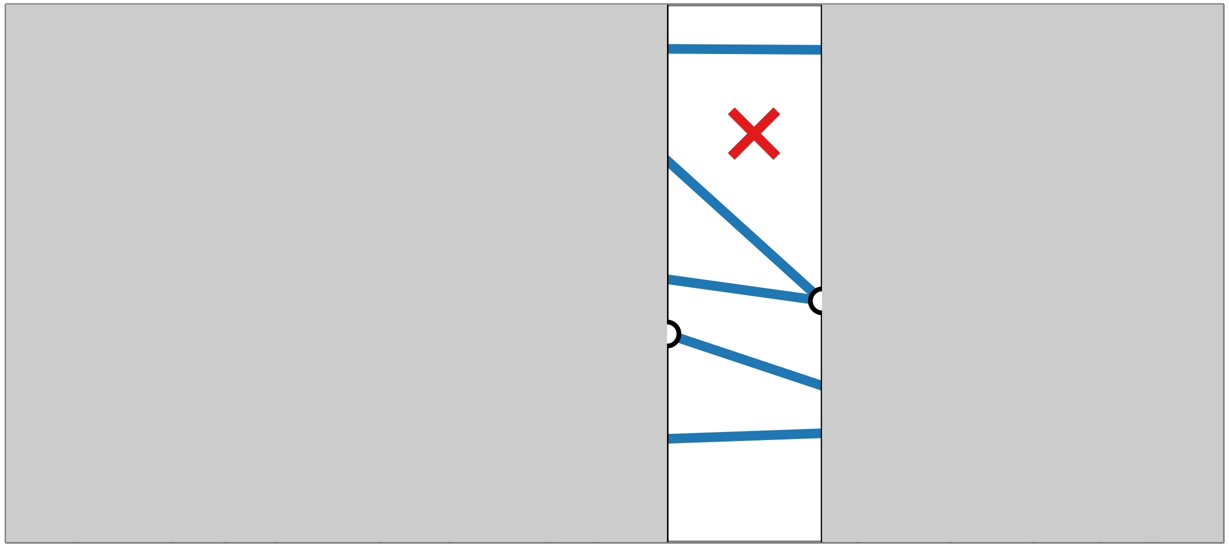


Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

Query: – find correct slab

What's the Problem?

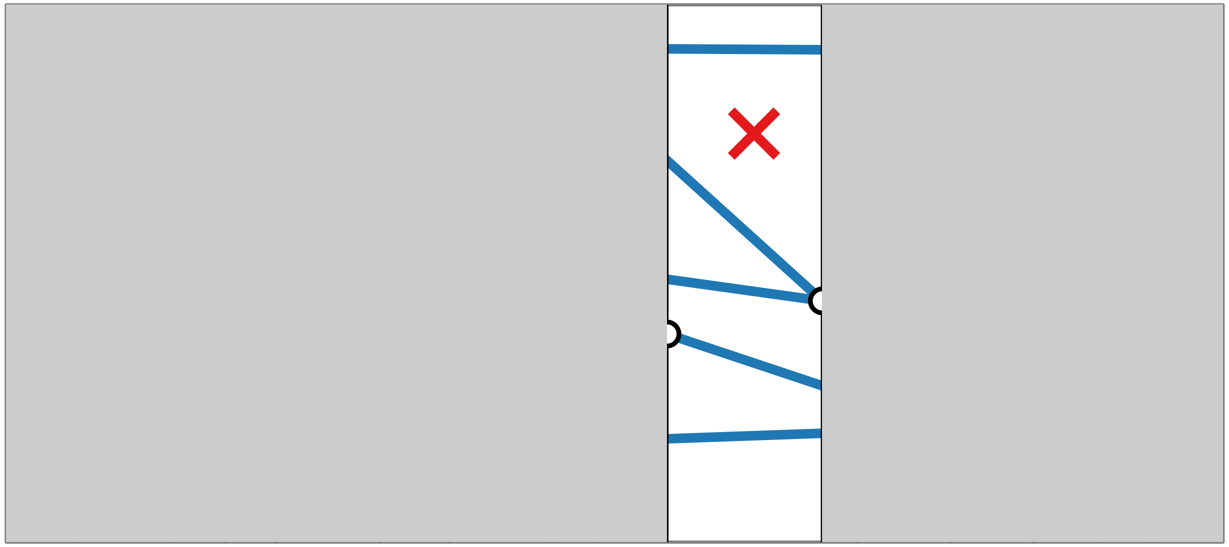


Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

Query: – find correct slab

What's the Problem?

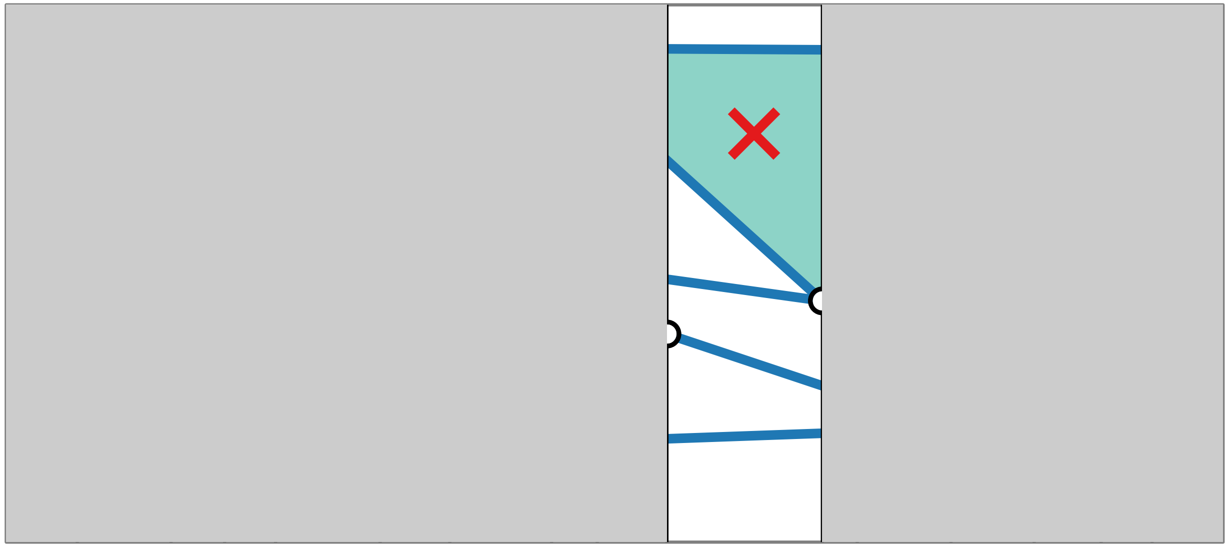


Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

Query: – find correct slab
– search in slab

What's the Problem?

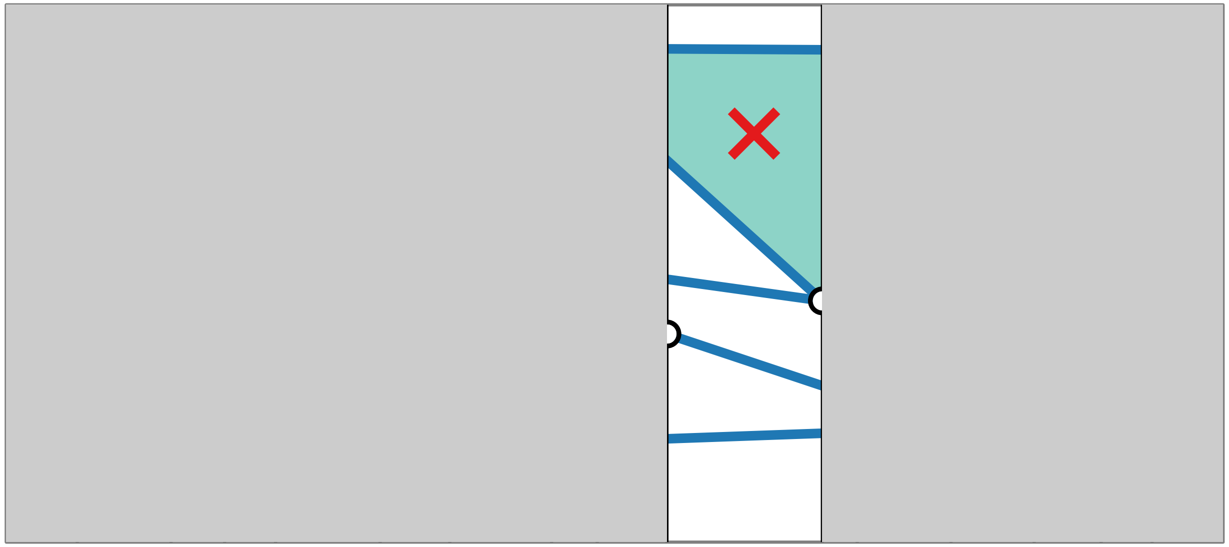


Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

Query: – find correct slab
– search in slab

What's the Problem?



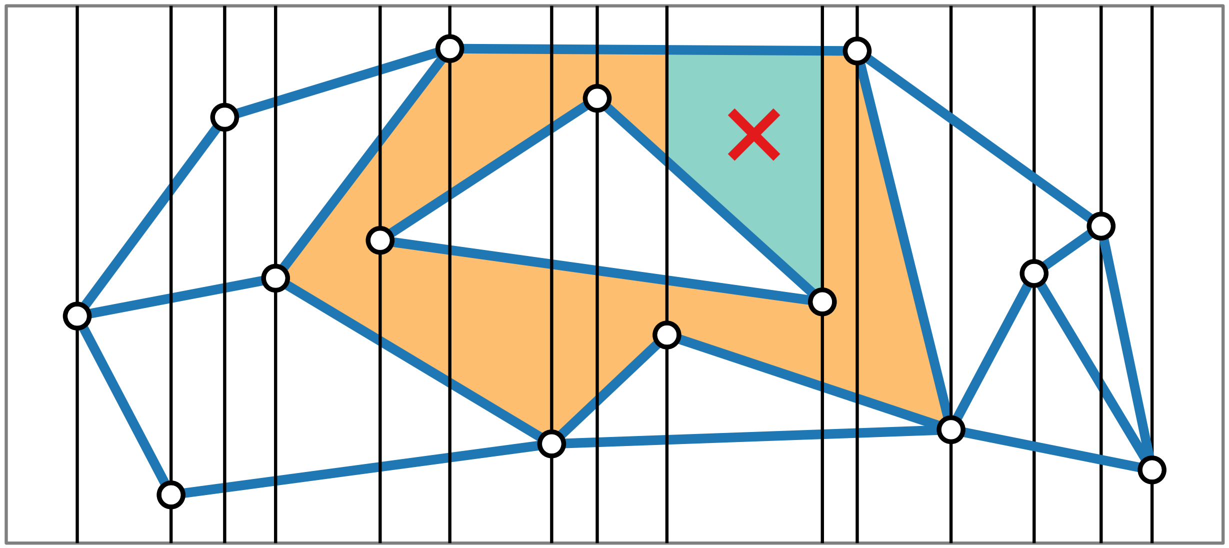
Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

Query: – find correct slab
– search in slab

} 2 bin. searches!

What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

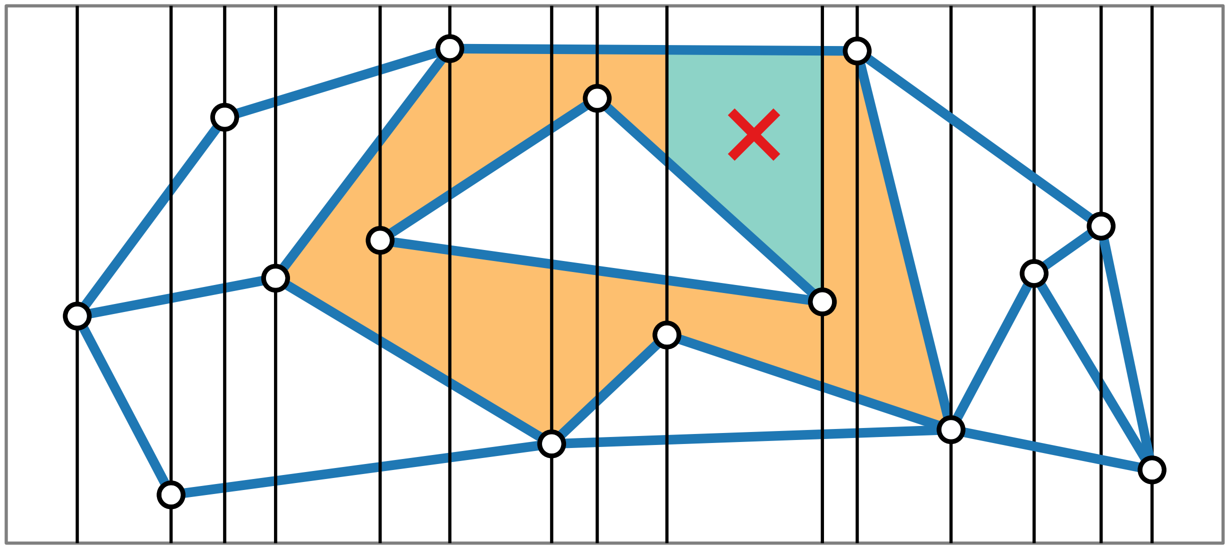
Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

Query: – find correct slab
– search in slab

} 2 bin. searches!

$O(\log n)$ time!

What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

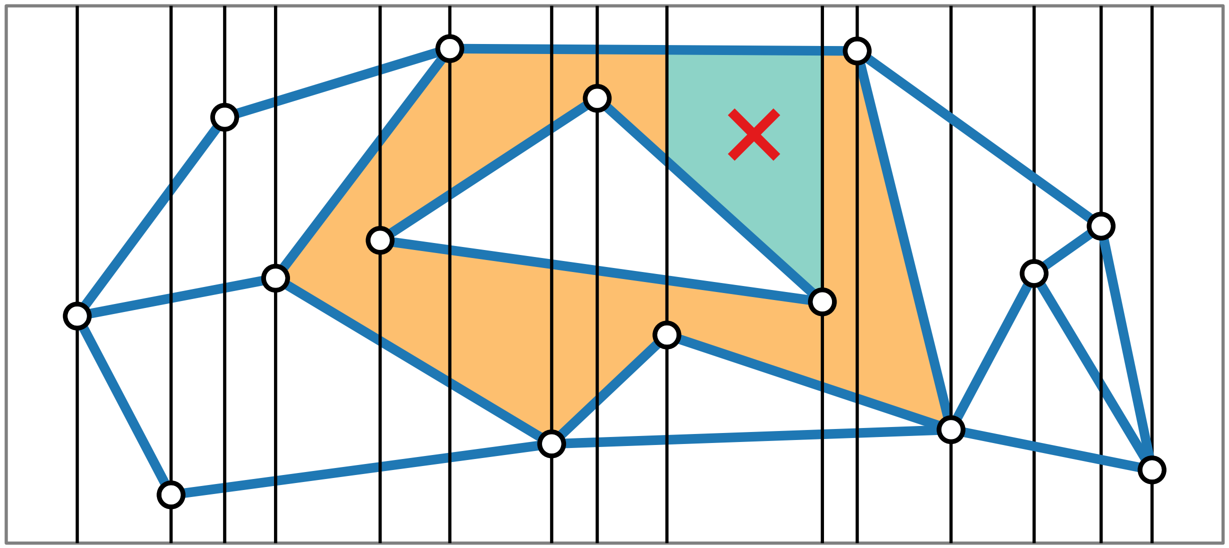
Query: – find correct slab
– search in slab

} 2 bin. searches!

$O(\log n)$ time!

But:

What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

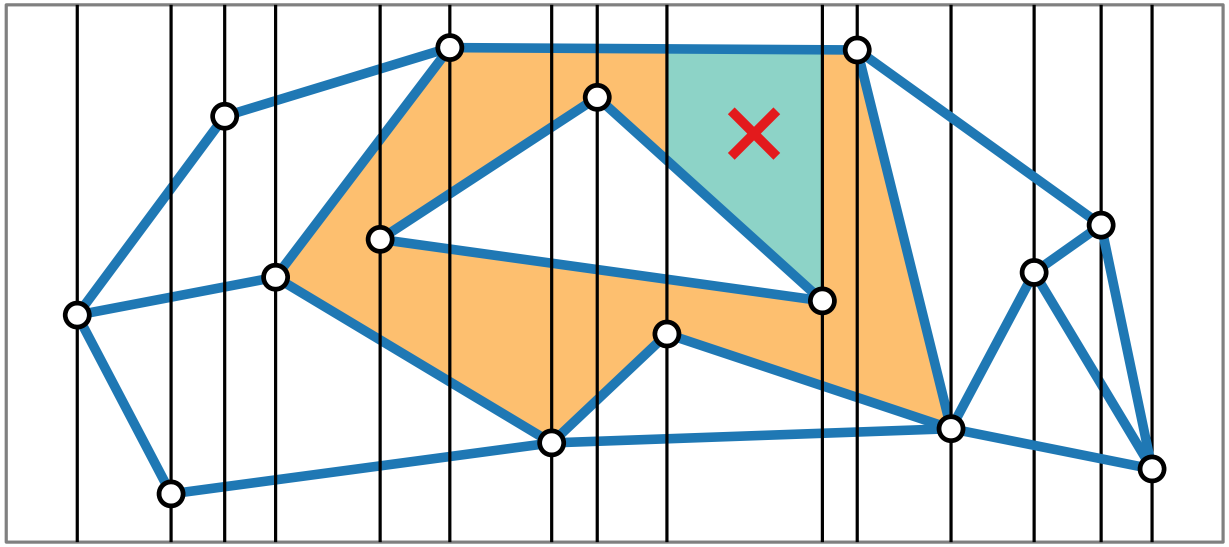
Query: – find correct slab
– search in slab

} 2 bin. searches!

$O(\log n)$ time!

But: Space?

What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

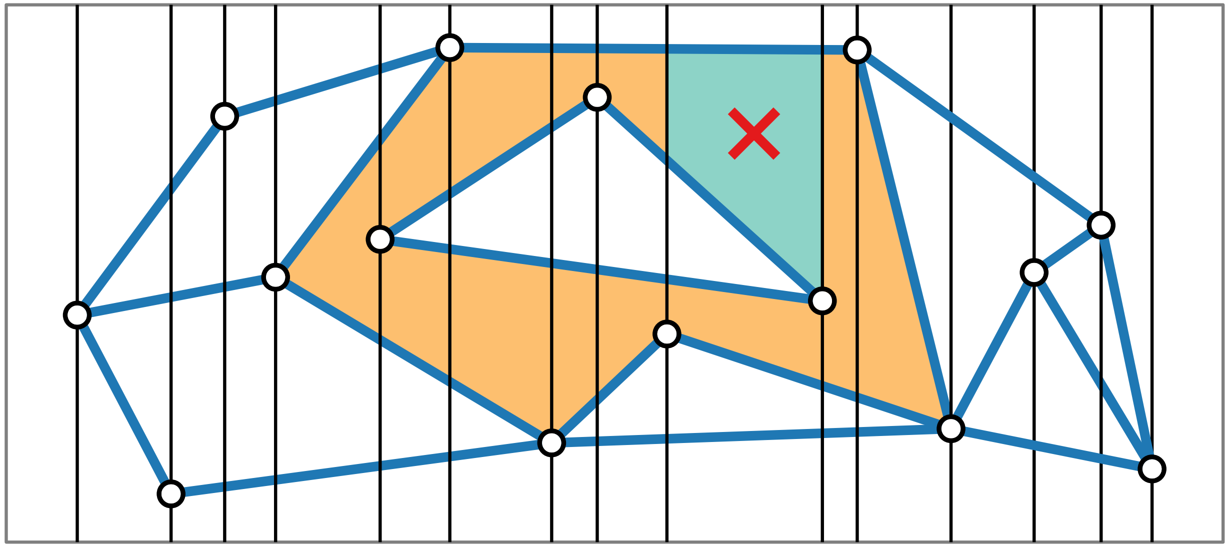
Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

Query: – find correct slab
– search in slab
} 2 bin. searches!

$O(\log n)$ time!

But: Space? $\Theta(n^2)$

What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

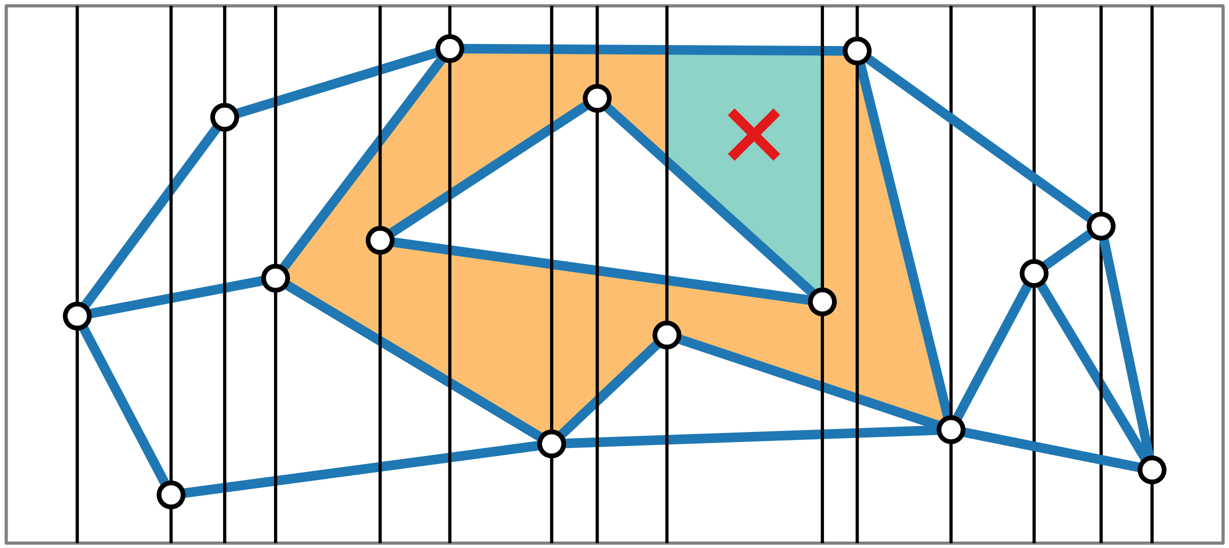
Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

Query: – find correct slab
– search in slab
} 2 bin. searches!

$O(\log n)$ time!

But: Space? $\Theta(n^2)$ **Task:** Tight example?

What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

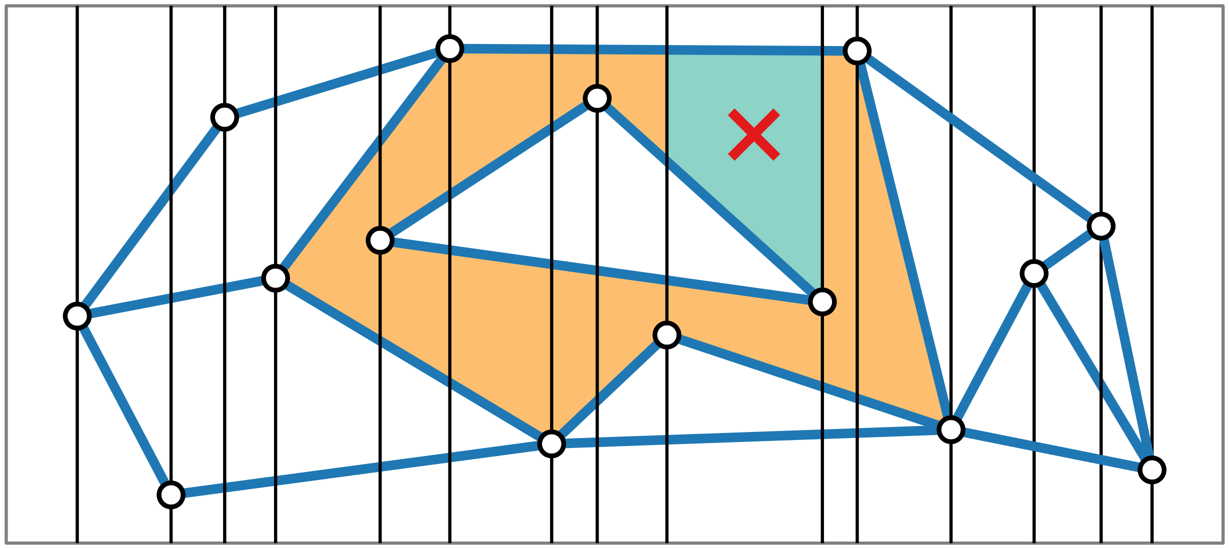
Query: – find correct slab
– search in slab

} 2 bin. searches!

$O(\log n)$ time!

But: Space? $\Theta(n^2)$ Preproc?

What's the Problem?



Task: Given a planar subdivision $\mathcal{S}$ with n segments, preprocess $\mathcal{S}$ to allow for fast pt. location queries!

Solution: Preproc.: Partition $\mathcal{S}$ into slabs induced by vertices.

Query: – find correct slab
– search in slab
} 2 bin. searches!

$O(\log n)$ time!

But: Space? $\Theta(n^2)$ Preproc? $O(n^2 \log n)$

Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

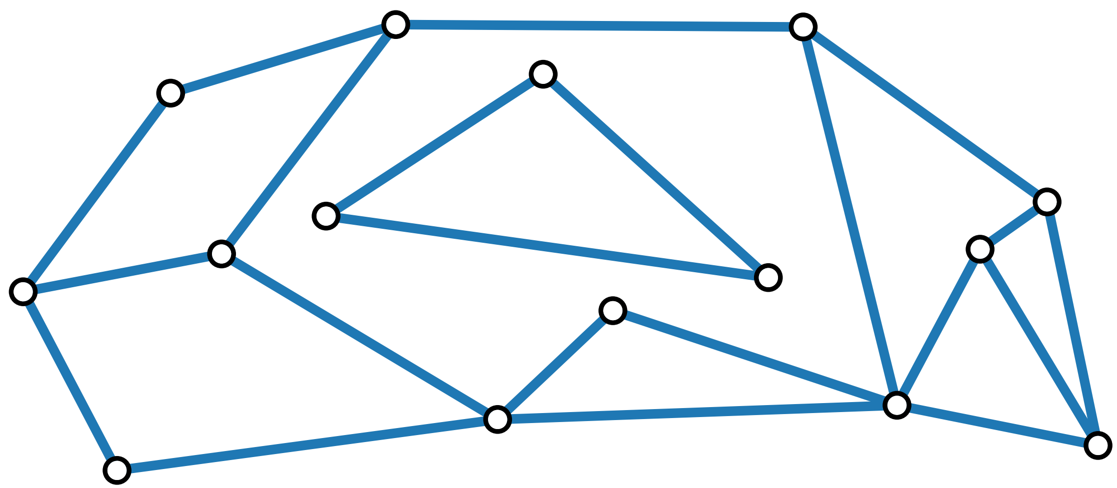
Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

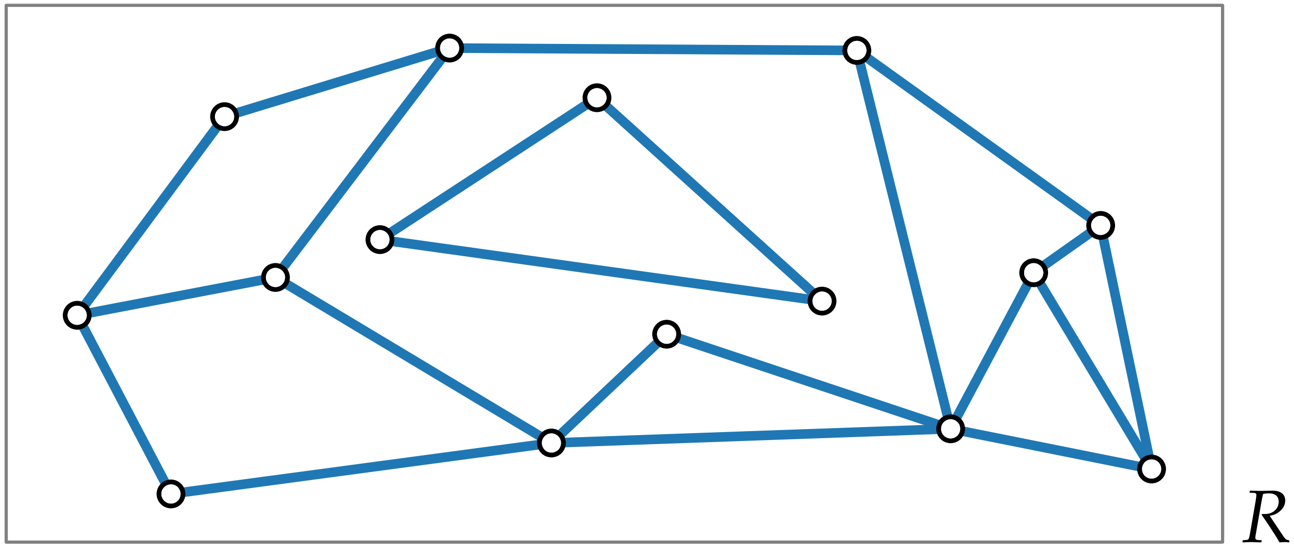


Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

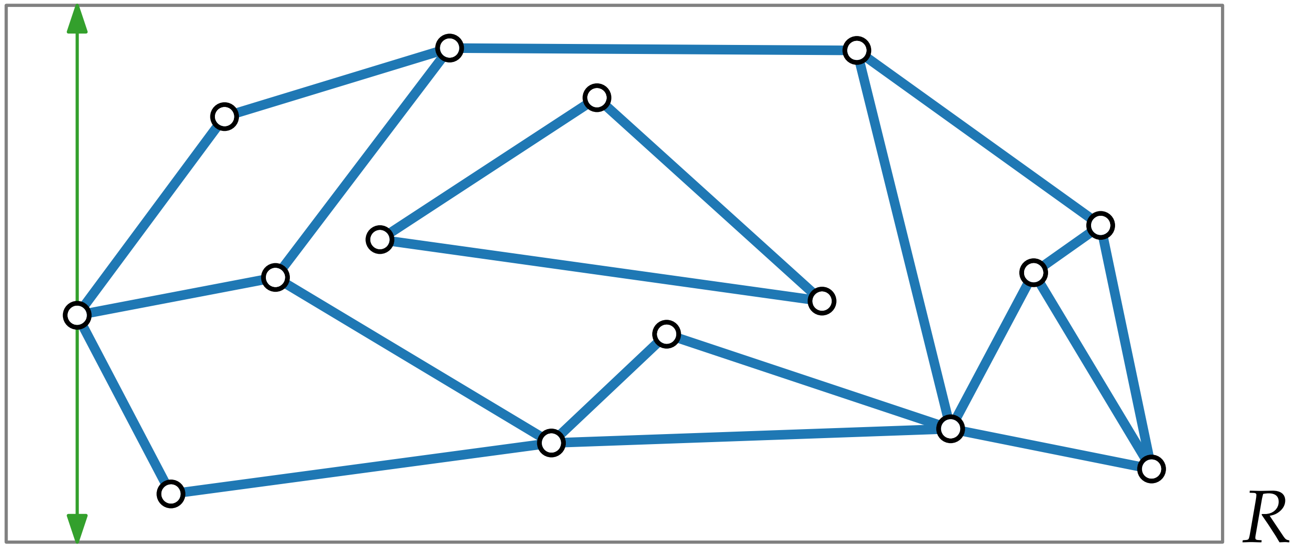


Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

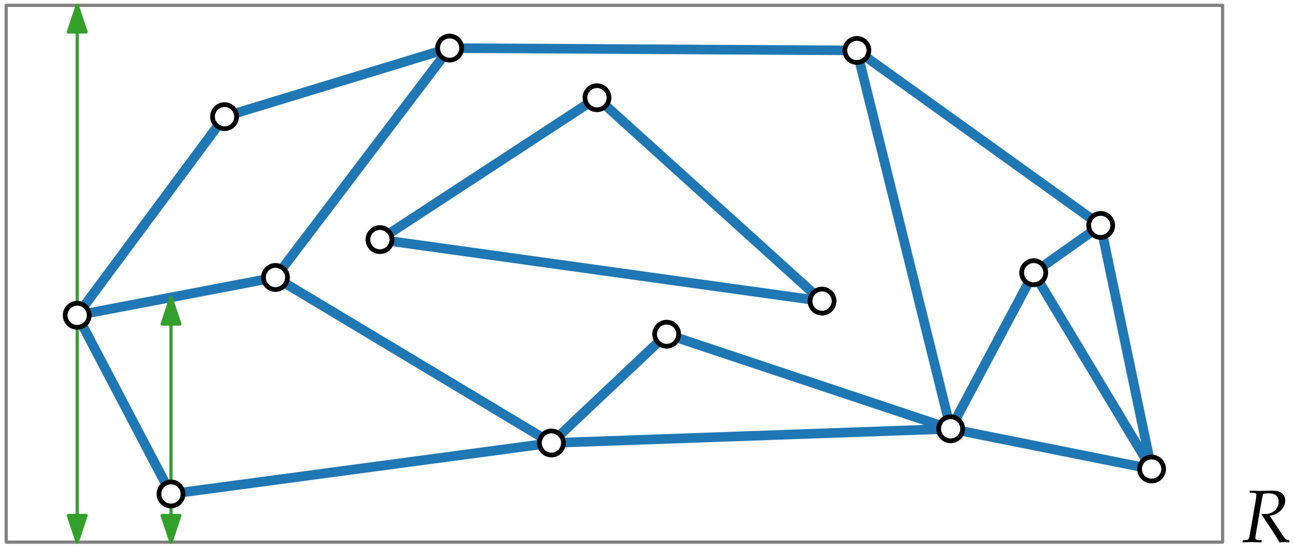


Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

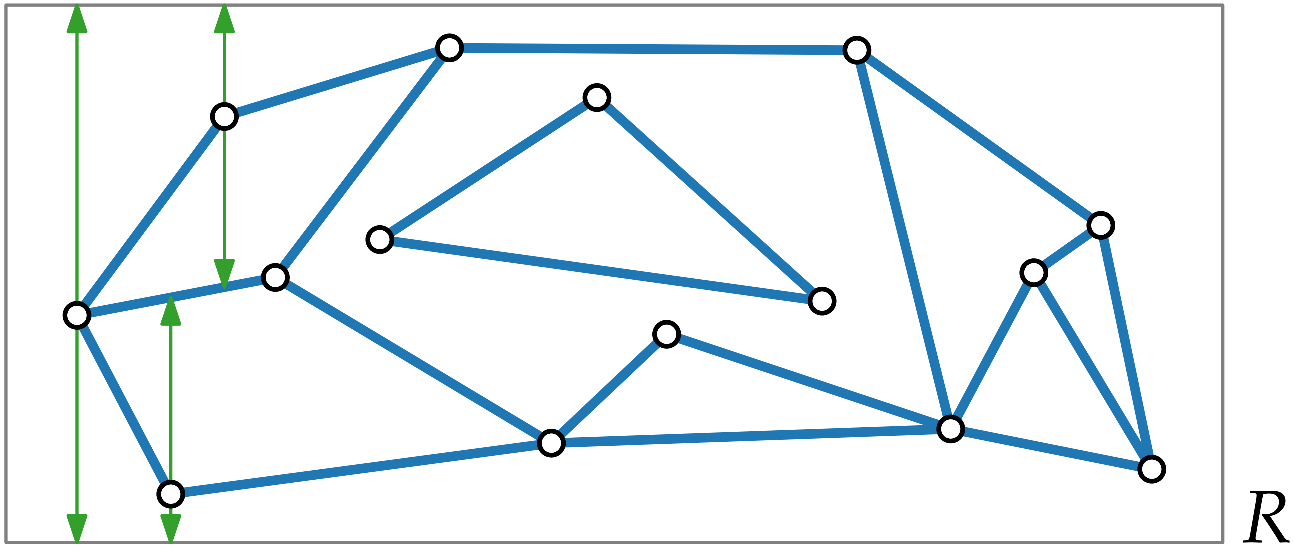


Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

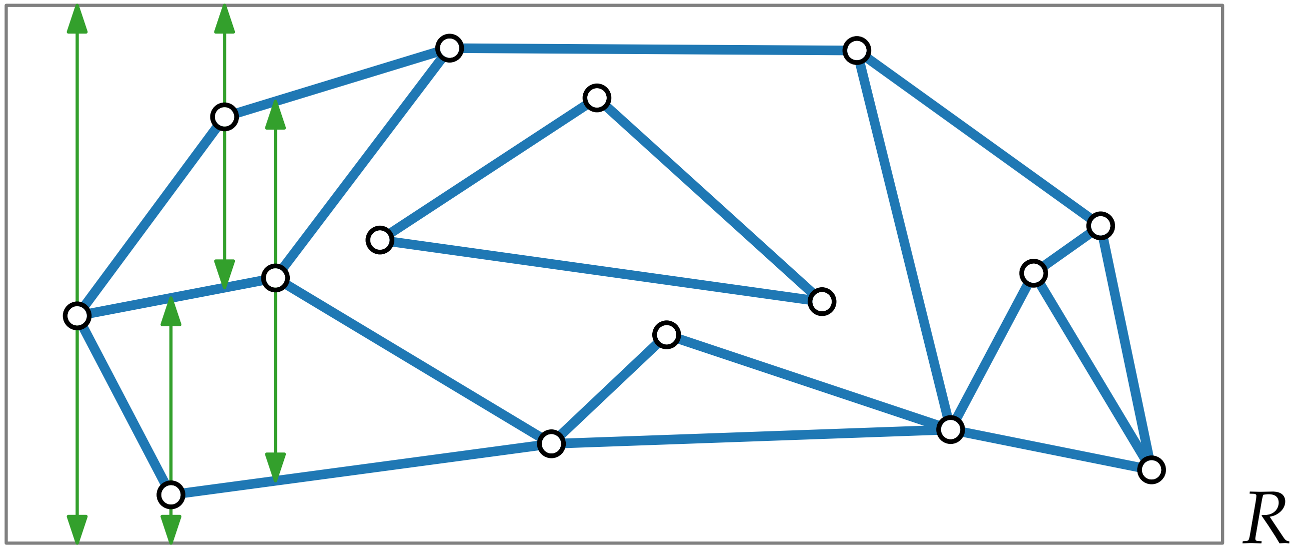


Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

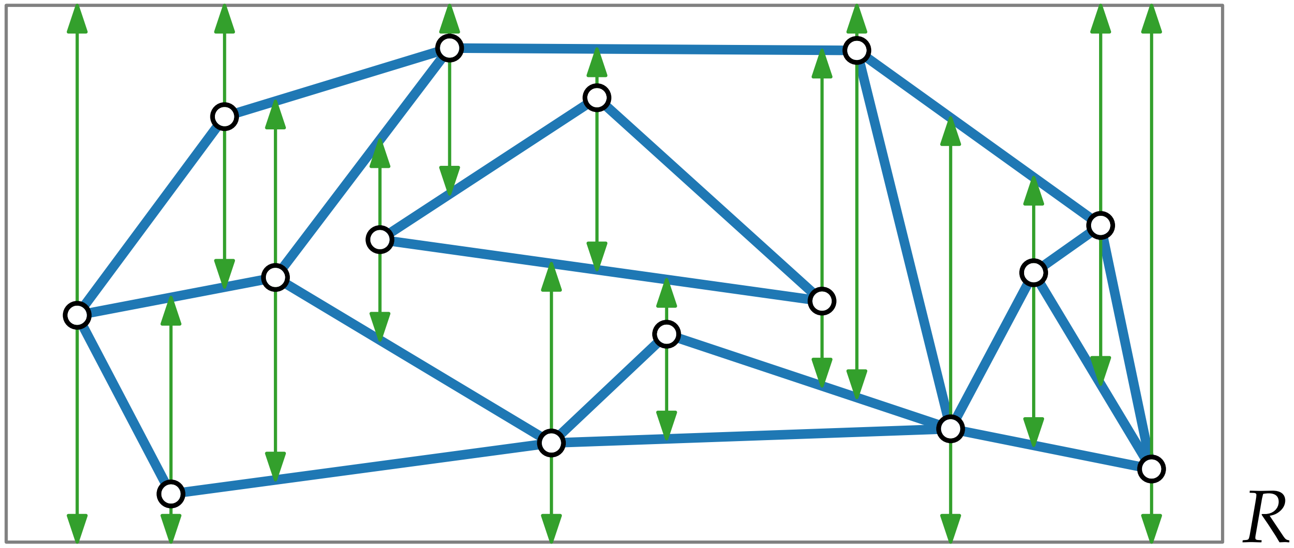


Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

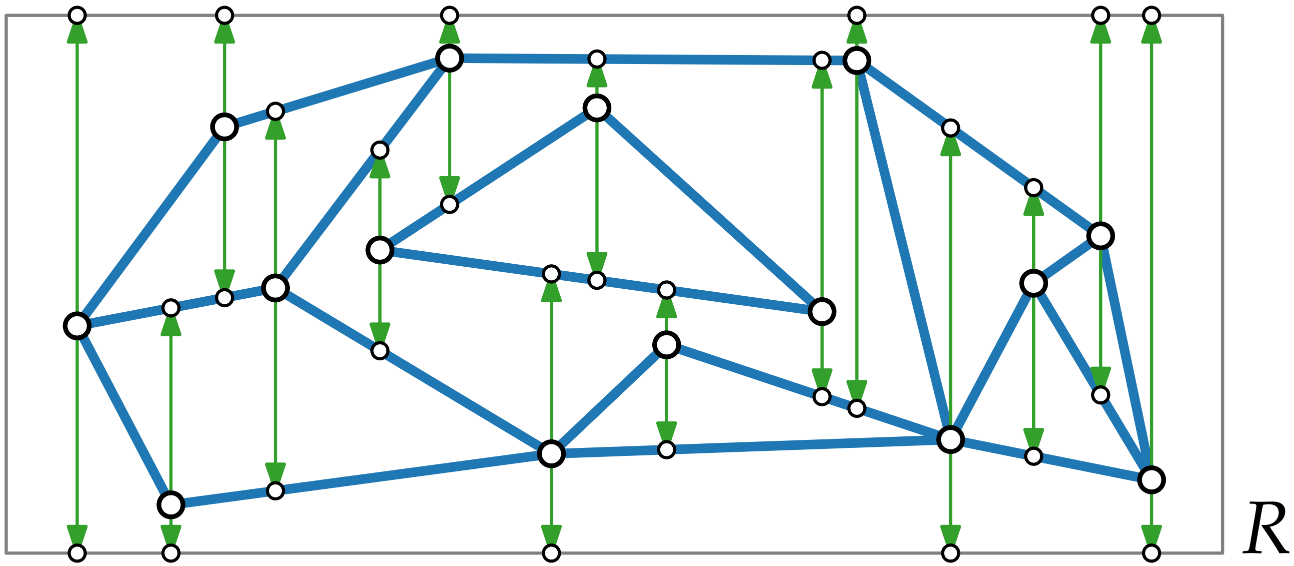


Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

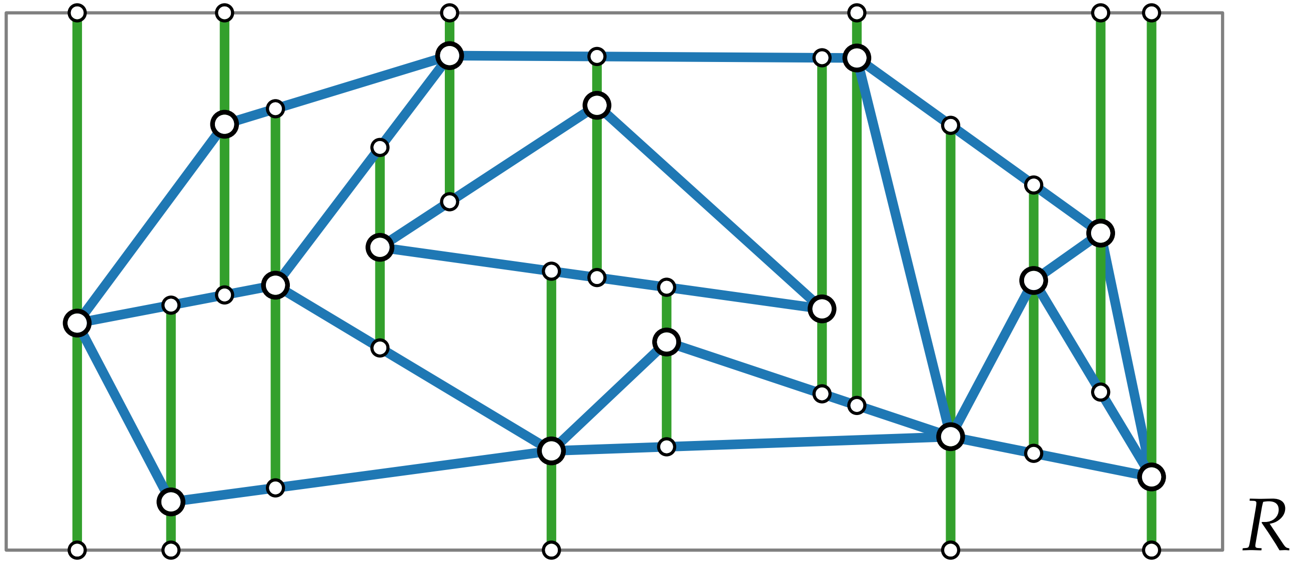


Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

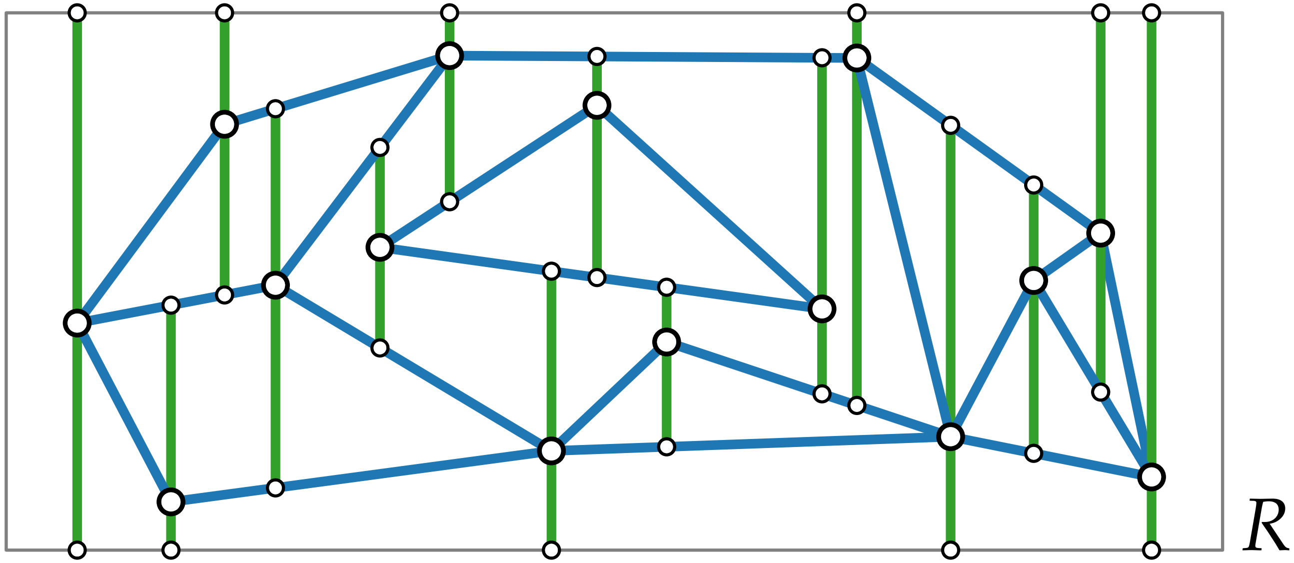


Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$



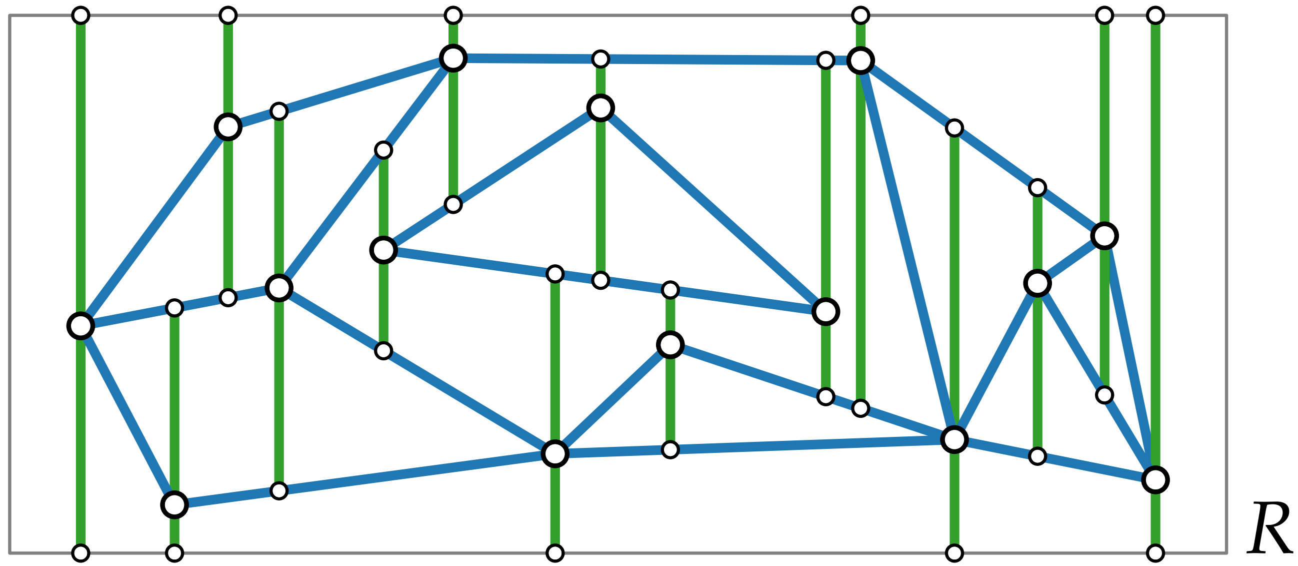
Assumption: $\mathcal{S}$ is in *general position*, that is, no two vertices have the same x-coordinates.

Decreasing the Complexity

Observation: The slab partition of $\mathcal{S}$ is a *refinement* $\mathcal{S}'$ of $\mathcal{S}$ that consists of (possibly degenerate) trapezoids.

Task: Find "good" refinement of $\mathcal{S}$ of low complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

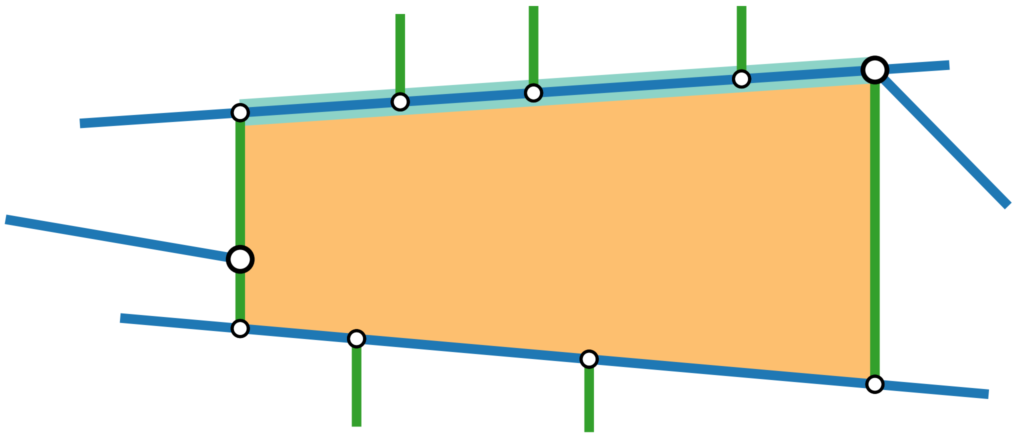


See Comp. Geom. A&A Ch. 6.3

Assumption: $\mathcal{S}$ is in *general position*, that is, no two vertices have the same x-coordinates.

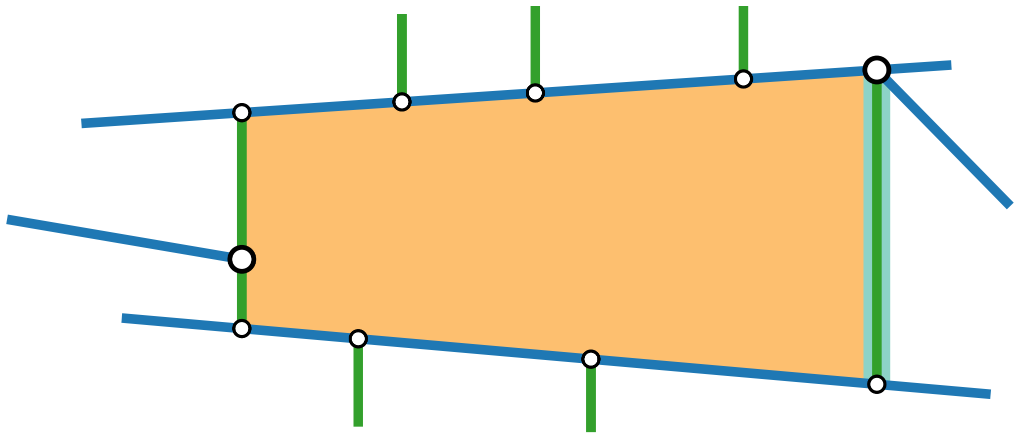
Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.



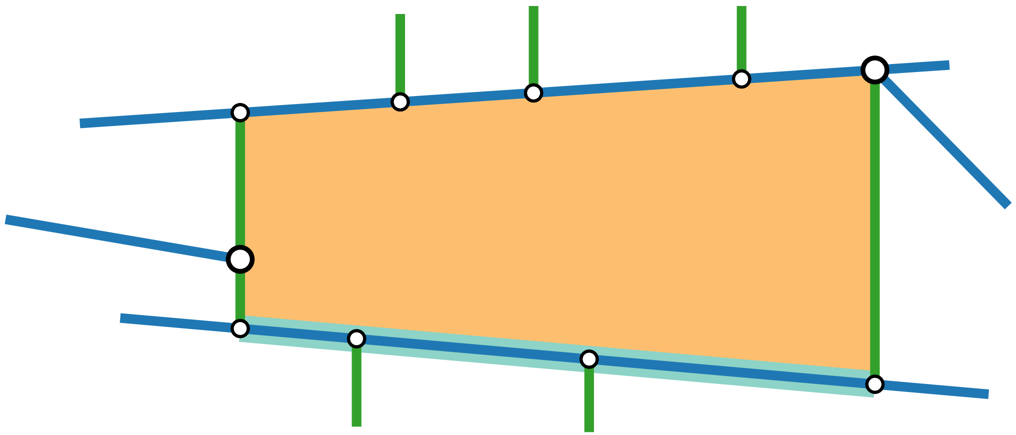
Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.



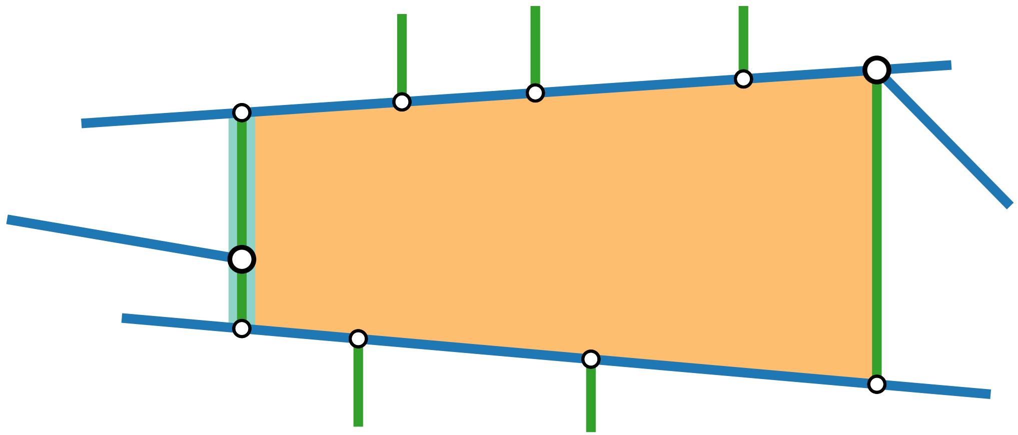
Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.



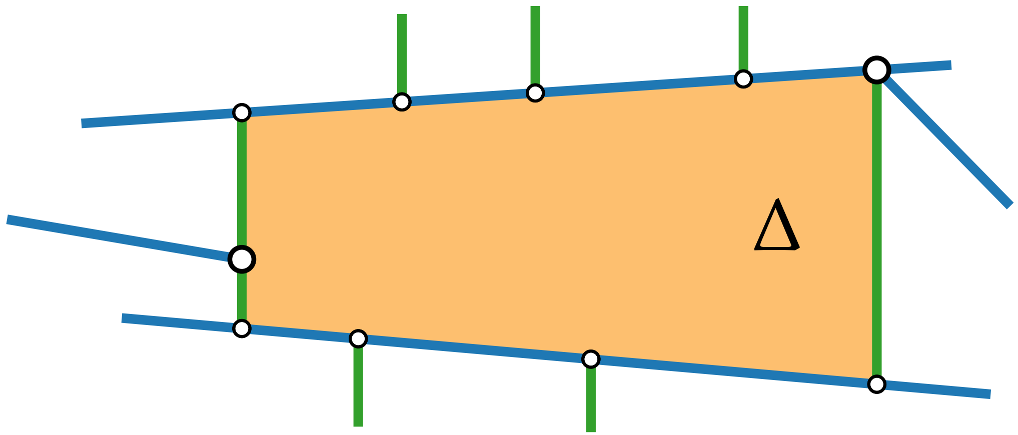
Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.



Notation

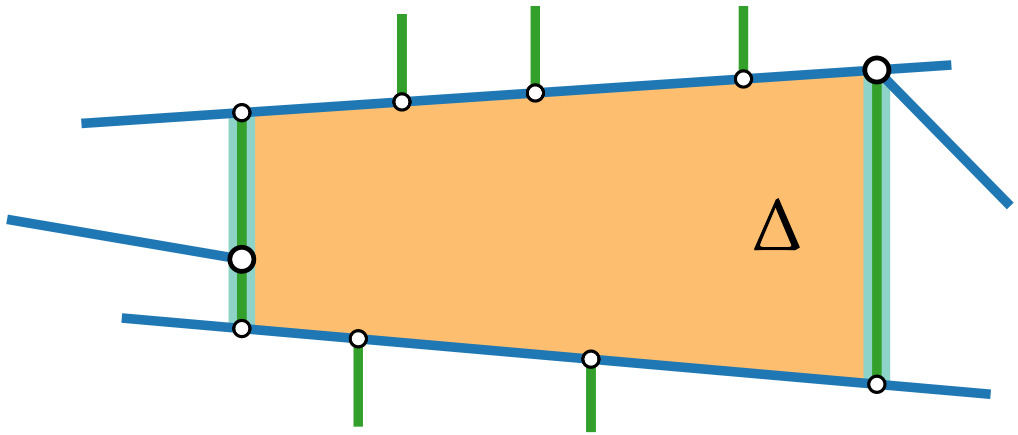
Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.



Observation: $\mathcal{S}$ in gen. pos. $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.

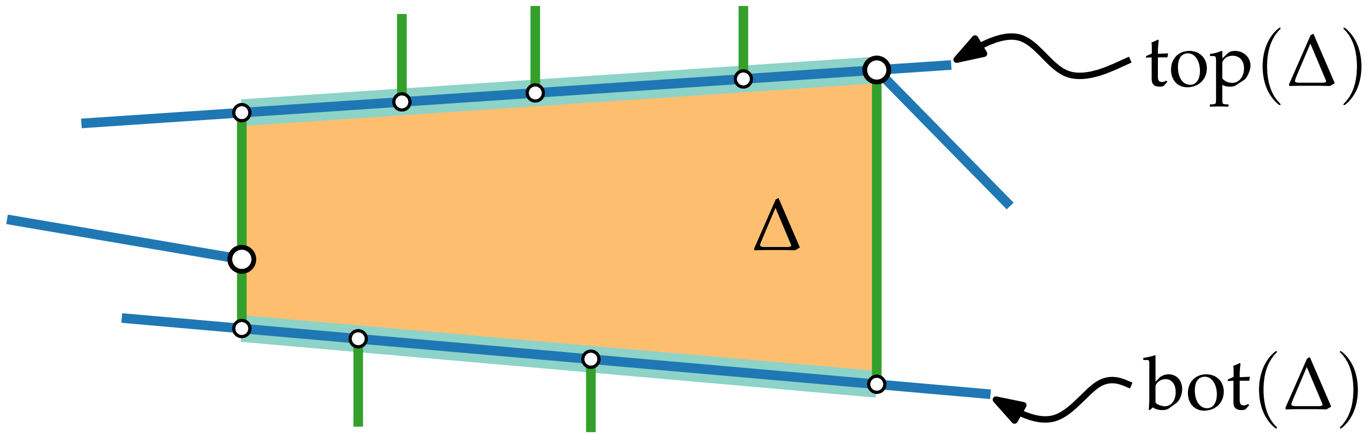


Observation: $\mathcal{S}$ in gen. pos. $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

– one or two vertical sides

Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.



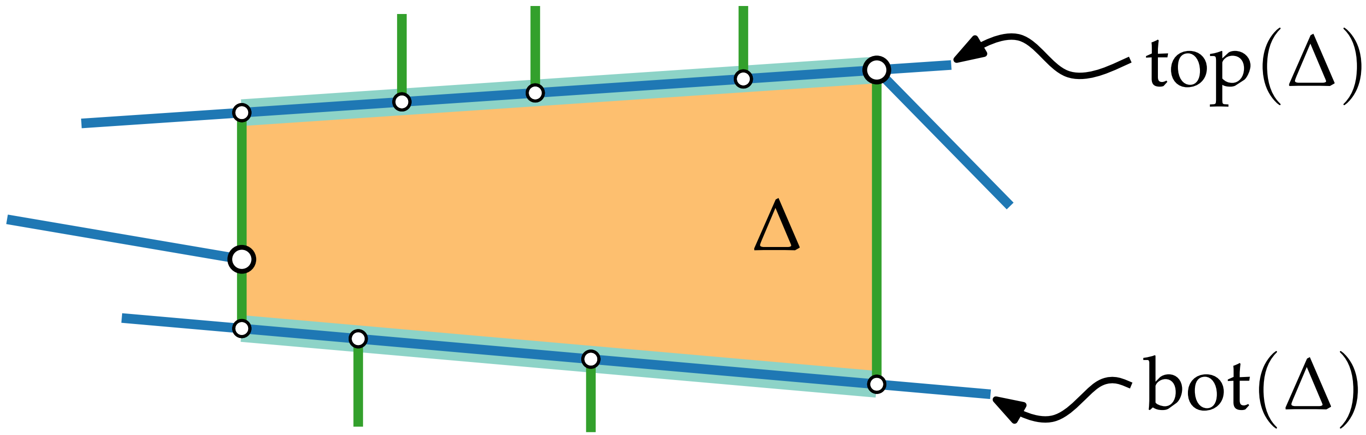
Observation: $\mathcal{S}$ in gen. pos. $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

– one or two vertical sides

– exactly 2 non-vertical sides

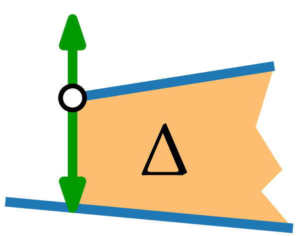
Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.



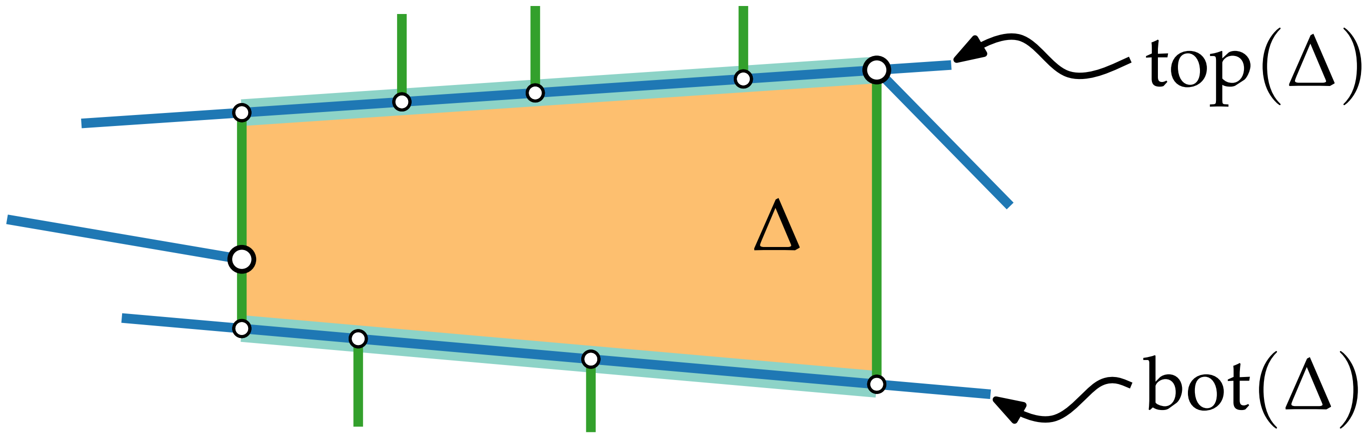
Observation: $\mathcal{S}$ in gen. pos. $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:
– one or two vertical sides
– exactly 2 non-vertical sides

Left side:



Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.

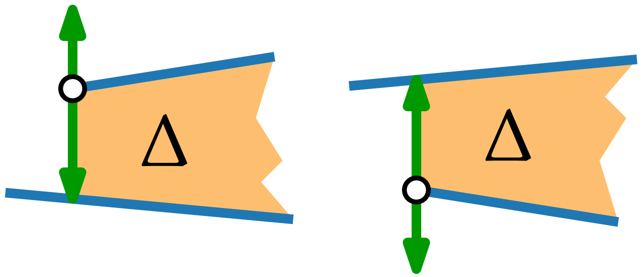


Observation: $\mathcal{S}$ in gen. pos. $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

– one or two vertical sides

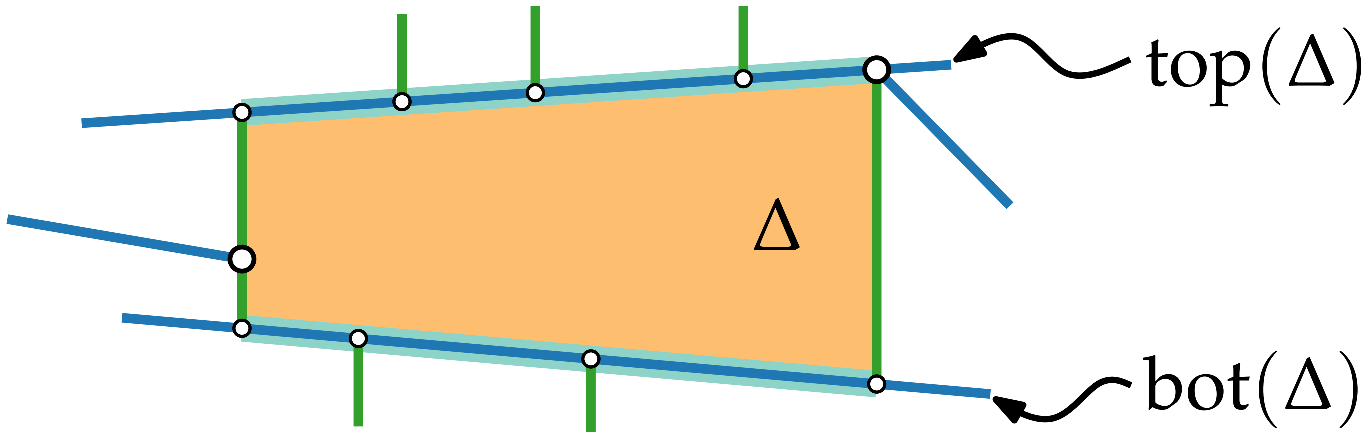
– exactly 2 non-vertical sides

Left side:



Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.

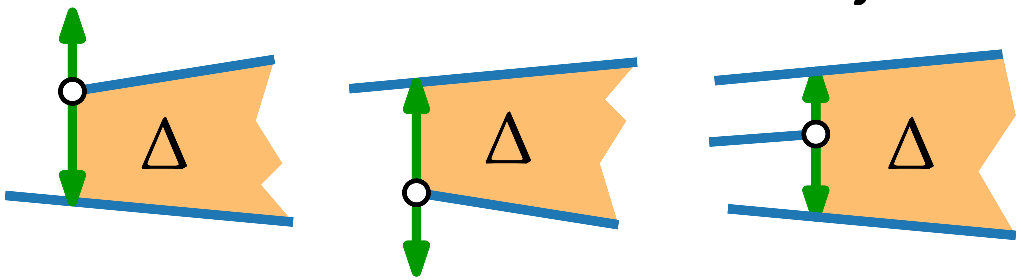


Observation: $\mathcal{S}$ in gen. pos. $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

– one or two vertical sides

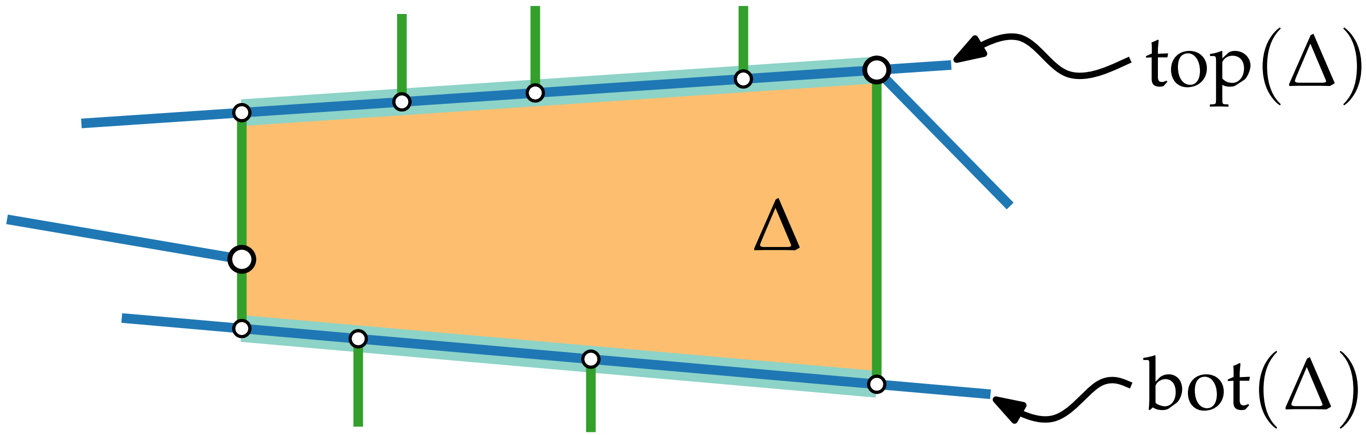
– exactly 2 non-vertical sides

Left side:



Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.

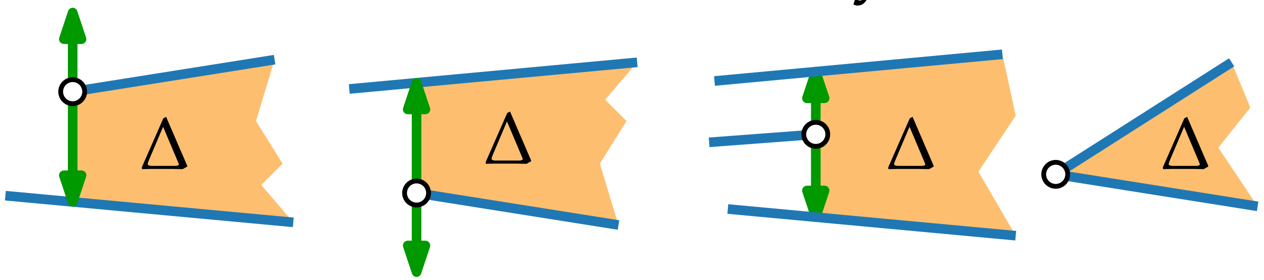


Observation: $\mathcal{S}$ in gen. pos. $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

– one or two vertical sides

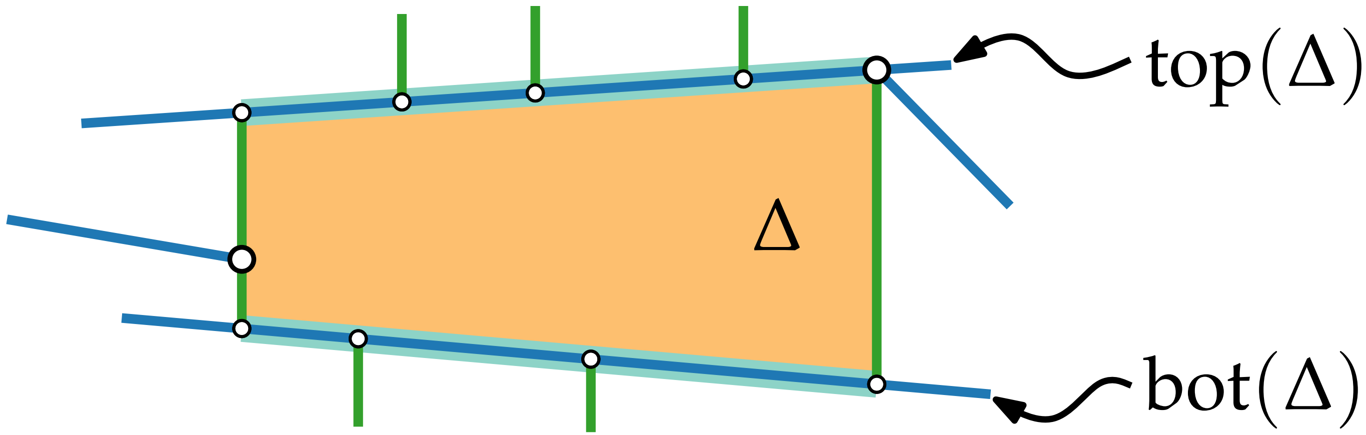
– exactly 2 non-vertical sides

Left side:



Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.

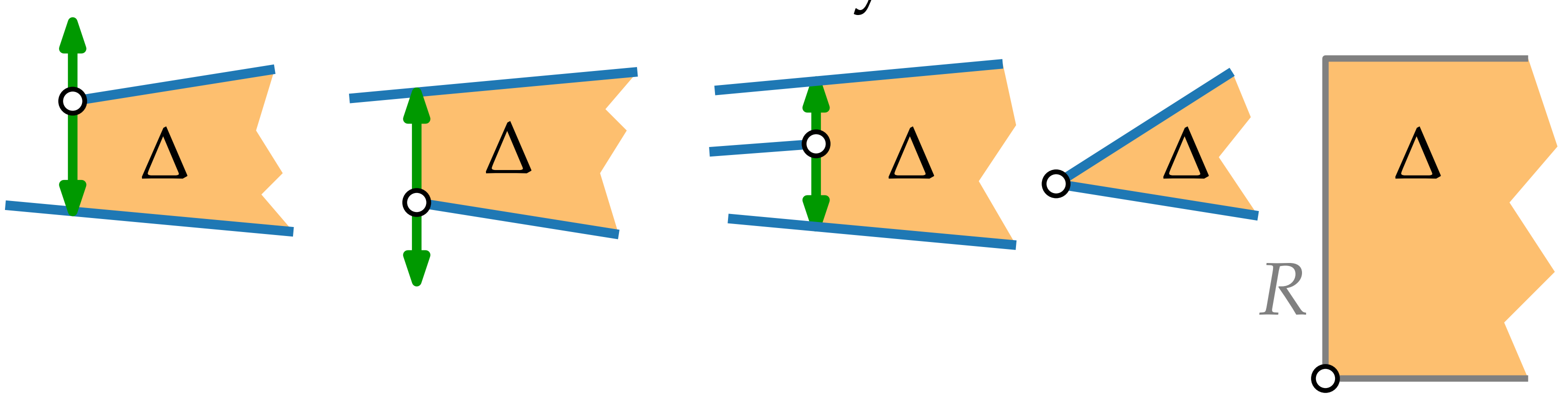


Observation: $\mathcal{S}$ in gen. pos. $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

– one or two vertical sides

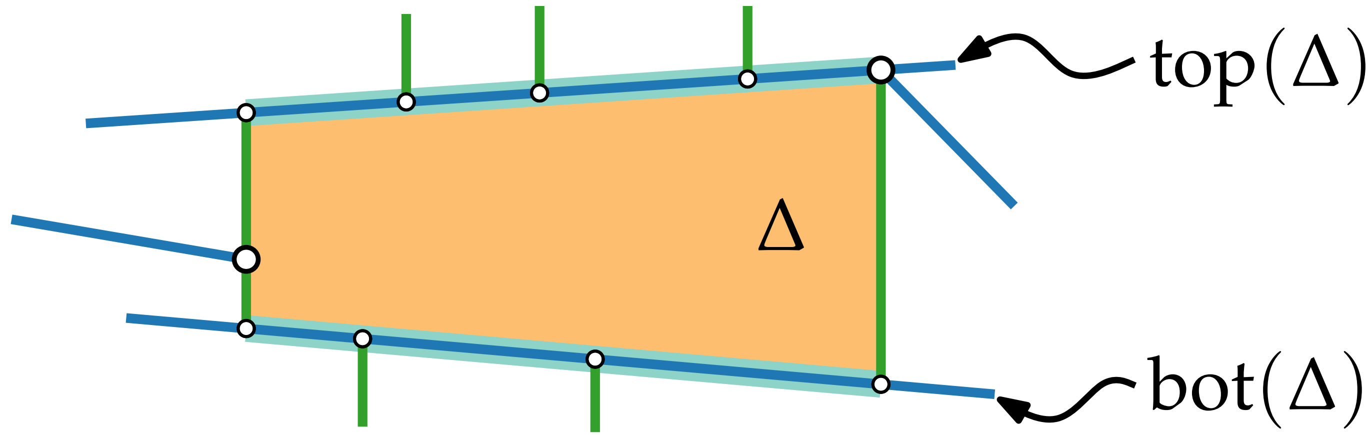
– exactly 2 non-vertical sides

Left side:



Notation

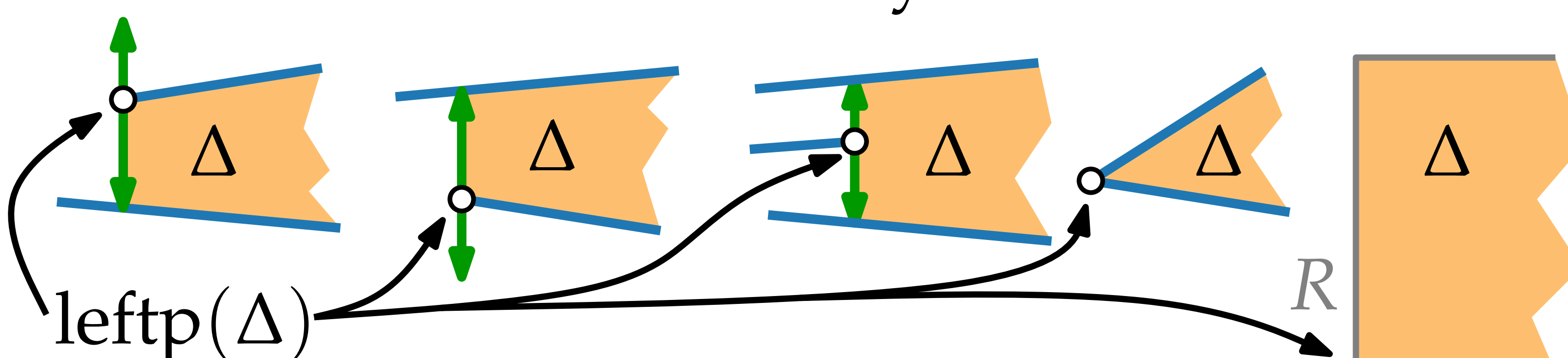
Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of max. length contained in the boundary of the face.



Observation: $\mathcal{S}$ in gen. pos. $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

– one or two vertical sides
– exactly 2 non-vertical sides

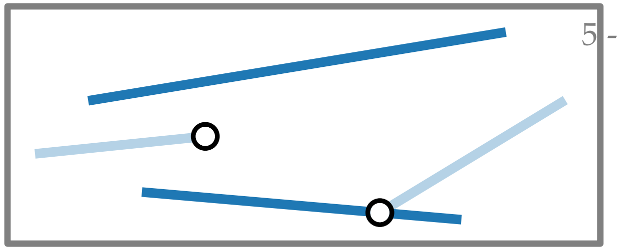
Left side:



Complexity of $\mathcal{T}(\mathcal{S})$

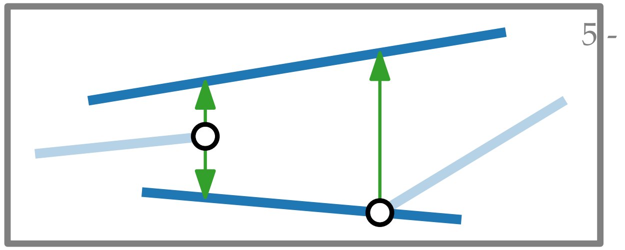
Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Complexity of $\mathcal{T}(\mathcal{S})$



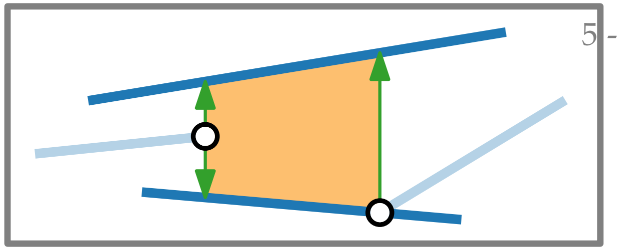
Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Complexity of $\mathcal{T}(\mathcal{S})$



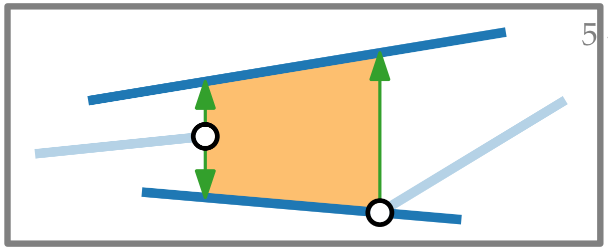
Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Complexity of $\mathcal{T}(\mathcal{S})$



Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

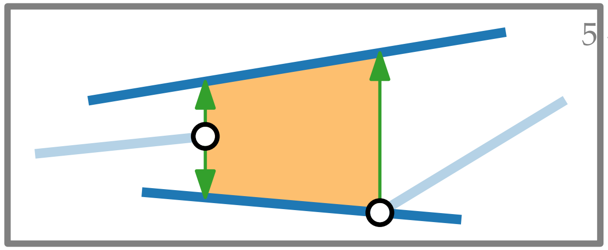
Complexity of $\mathcal{T}(\mathcal{S})$



Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq$ vtc and $\leq$ trapezoids.

Complexity of $\mathcal{T}(\mathcal{S})$



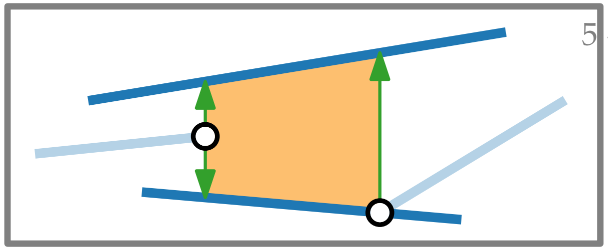
Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by $\mathrm{top}(\Delta)$, $\mathrm{bot}(\Delta)$, $\mathrm{leftp}(\Delta)$, and $\mathrm{rightp}(\Delta)$.

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq$ vtc and $\leq$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

– endpts of segments in $\mathcal{S}$

Complexity of $\mathcal{T}(\mathcal{S})$



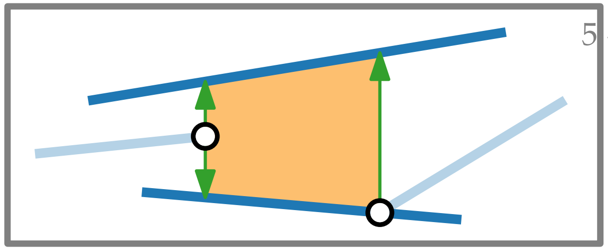
Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq$ vtc and $\leq$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

– endpts of segments in $\mathcal{S}$ $\quad \leq 2n$

Complexity of $\mathcal{T}(\mathcal{S})$



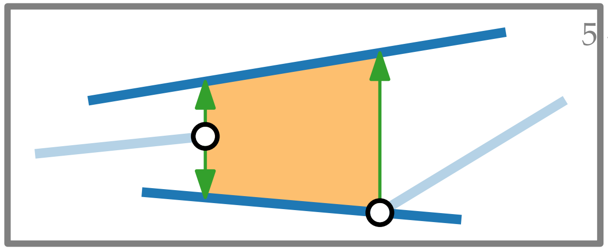
Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq$ vtc and $\leq$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

– endpts of segments in $\mathcal{S}$ $\leq 2n$
– endpts of vertical extensions

Complexity of $\mathcal{T}(\mathcal{S})$



Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

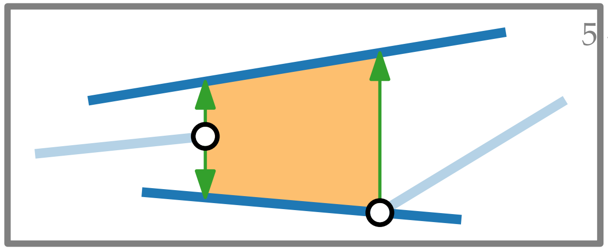
Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq$ vtc and $\leq$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

– endpts of segments in $\mathcal{S}$ $\leq 2n$

– endpts of vertical extensions $\leq 2 \cdot 2n$

Complexity of $\mathcal{T}(\mathcal{S})$



Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq$ vtc and $\leq$ trapezoids.

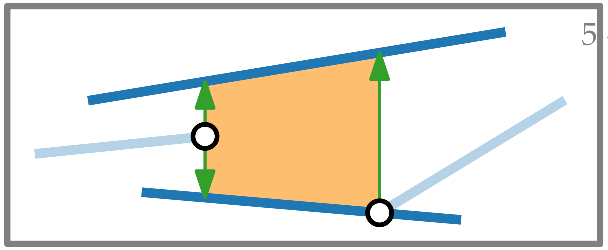
Proof. The vertices of $\mathcal{T}(S)$ are

– endpts of segments in $\mathcal{S}$ $\leq 2n$

– endpts of vertical extensions $\leq 2 \cdot 2n$

– vertices of R

Complexity of $\mathcal{T}(\mathcal{S})$



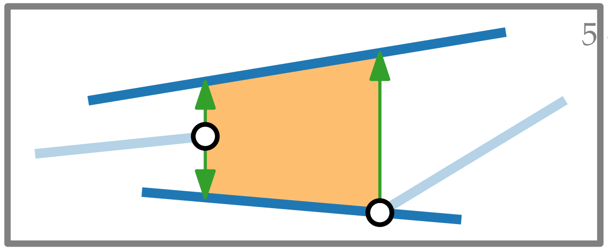
Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq$ vtc and $\leq$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

- endpts of segments in $\mathcal{S}$ $\leq 2n$
- endpts of vertical extensions $\leq 2 \cdot 2n$
- vertices of R 4

Complexity of $\mathcal{T}(\mathcal{S})$



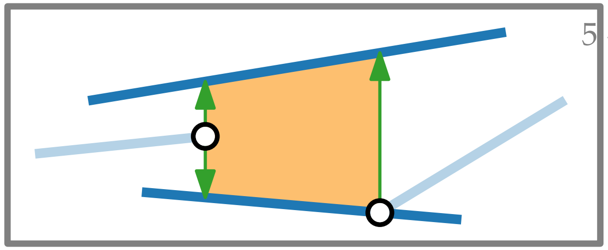
Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq$ vtc and $\leq$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

$$\left.\begin{array}{ll} \text{– endpts of segments in } \mathcal{S} & \leq 2n \\ \text{– endpts of vertical extensions} & \leq 2 \cdot 2n \\ \text{– vertices of } R & 4 \end{array}\right\}$$

Complexity of $\mathcal{T}(\mathcal{S})$



Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

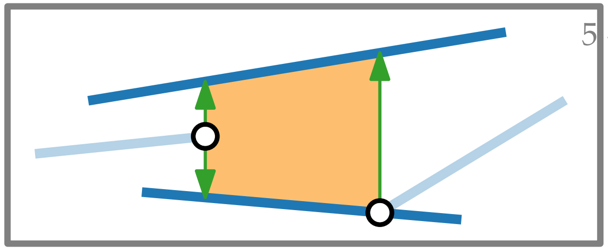
Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq$ vtc and $\leq$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

– endpts of segments in $\mathcal{S}$ $\leq 2n$
– endpts of vertical extensions $\leq 2 \cdot 2n$
– vertices of R 4

$\Bigg\} \leq 6n + 4$

Complexity of $\mathcal{T}(\mathcal{S})$



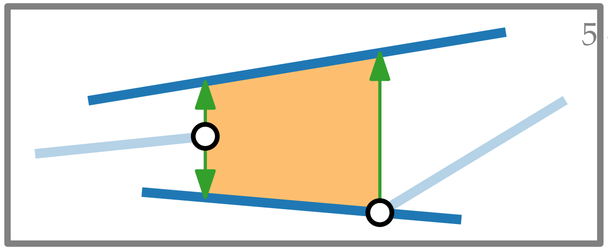
Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq 6n+4$ vtc and $\leq 3n+1$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

$$\left.\begin{array}{ll} \text{– endpts of segments in } \mathcal{S} & \leq 2n \\ \text{– endpts of vertical extensions} & \leq 2 \cdot 2n \\ \text{– vertices of } R & 4 \end{array}\right\} \leq 6n+4$$

Complexity of $\mathcal{T}(\mathcal{S})$



Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

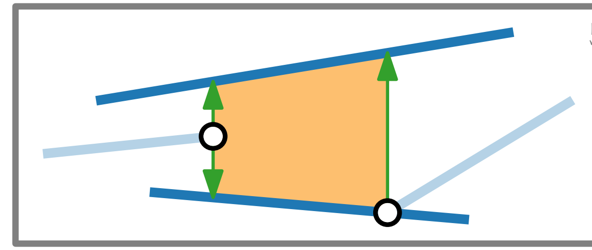
Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq 6n + 4$ vtc and $\leq 3n + 1$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

$$\left.\begin{array}{ll} \text{– endpts of segments in } \mathcal{S} & \leq 2n \\ \text{– endpts of vertical extensions} & \leq 2 \cdot 2n \\ \text{– vertices of } R & 4 \end{array}\right\} \leq 6n + 4$$

Bound #trapezoids via Euler or directly (segments/leftp).

Complexity of $\mathcal{T}(\mathcal{S})$



Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq 6n + 4$ vtc and $\leq 3n + 1$ trapezoids.

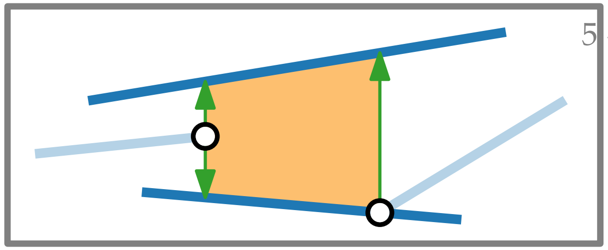
Proof. The vertices of $\mathcal{T}(S)$ are

$$\left.\begin{array}{ll} \text{– endpts of segments in } \mathcal{S} & \leq 2n \\ \text{– endpts of vertical extensions} & \leq 2 \cdot 2n \\ \text{– vertices of } R & 4 \end{array}\right\} \leq 6n + 4$$

Bound #trapezoids via Euler or directly (segments/leftp).

Approach:

Complexity of $\mathcal{T}(\mathcal{S})$



Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq 6n+4$ vtc and $\leq 3n+1$ trapezoids.

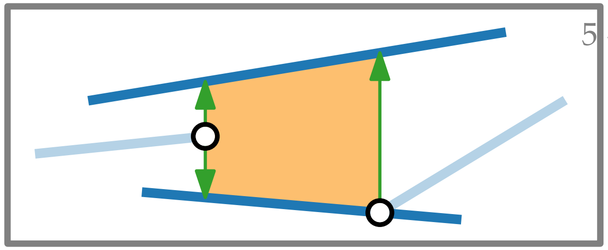
Proof. The vertices of $\mathcal{T}(S)$ are

$$\left.\begin{array}{ll} \text{– endpts of segments in } \mathcal{S} & \leq 2n \\ \text{– endpts of vertical extensions} & \leq 2 \cdot 2n \\ \text{– vertices of } R & 4 \end{array}\right\} \leq 6n+4$$

Bound #trapezoids via Euler or directly (segments/leftp).

Approach: Construct trapezoidal map $\mathcal{T}(\mathcal{S})$ and point-location data structure $\mathcal{D}(\mathcal{S})$ for $\mathcal{T}(\mathcal{S})$ *incrementally*!

Complexity of $\mathcal{T}(\mathcal{S})$



Observe: A face Δ of $\mathcal{T}(\mathcal{S})$ is uniquely defined by top(Δ), bot(Δ), leftp(Δ), and rightp(Δ).

Lemma. $\mathcal{S}$ planar subdivision in gen. pos. with n segments $\Rightarrow \mathcal{T}(\mathcal{S})$ has $\leq 6n+4$ vtc and $\leq 3n+1$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

$$\left.\begin{array}{ll} \text{– endpts of segments in } \mathcal{S} & \leq 2n \\ \text{– endpts of vertical extensions} & \leq 2 \cdot 2n \\ \text{– vertices of } R & 4 \end{array}\right\} \leq 6n+4$$

Bound #trapezoids via Euler or directly (segments/leftp).

Approach: Construct trapezoidal map $\mathcal{T}(\mathcal{S})$ and point-location data structure $\mathcal{D}(\mathcal{S})$ for $\mathcal{T}(\mathcal{S})$ *incrementally*!

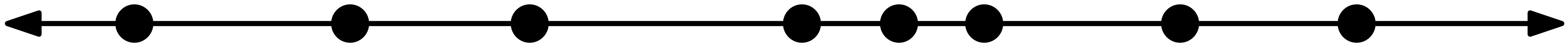
algorithm-design paradigm!

The 1D Problem

Given a set S of n real numbers...

The 1D Problem

Given a set S of n real numbers...



The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, $I_{i-1} :=$ set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

The 1D Problem

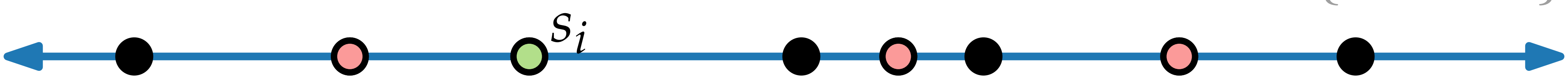
Given a set S of n real numbers... $i \in \{1, \ldots, n\}$

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

– pick an arbitrary point s_i from $S \setminus S_{i-1}$

The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$

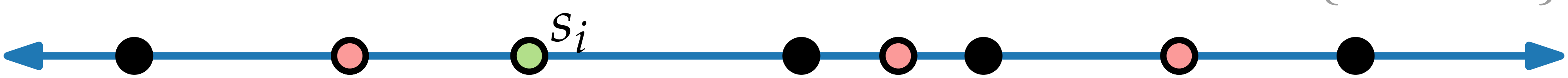


$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

– pick an arbitrary point s_i from $S \setminus S_{i-1}$

The 1D Problem

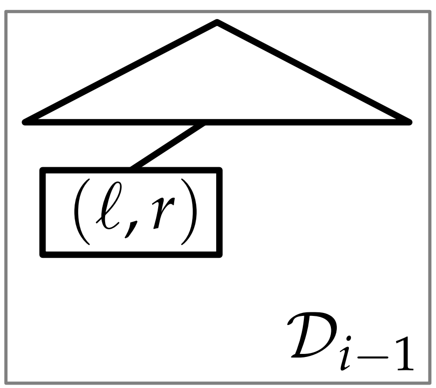
Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

– pick an arbitrary point s_i from $S \setminus S_{i-1}$

– locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}



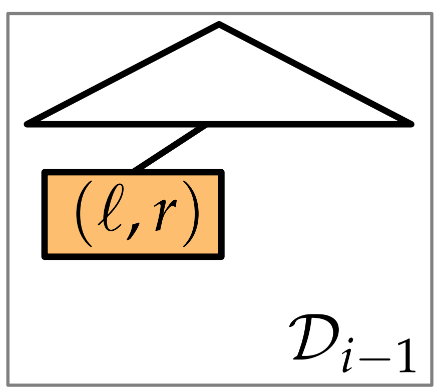
The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

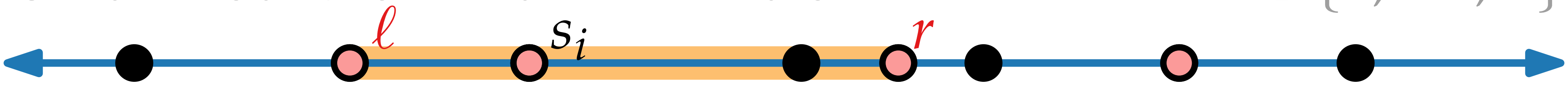
– pick an arbitrary point s_i from $S \setminus S_{i-1}$

– locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}



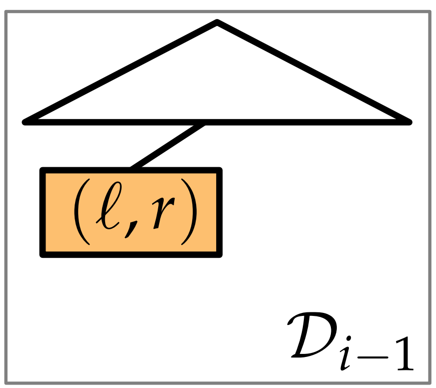
The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



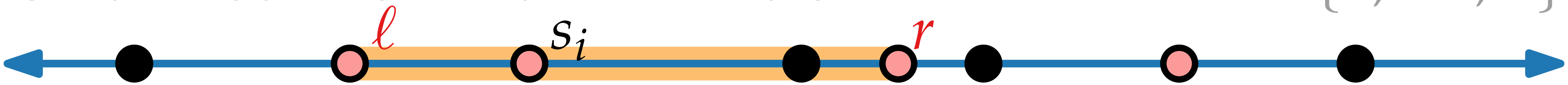
$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

– pick an arbitrary point s_i from $S \setminus S_{i-1}$
– locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}
– split interval (ℓ, r) of I_{i-1} containing s_i



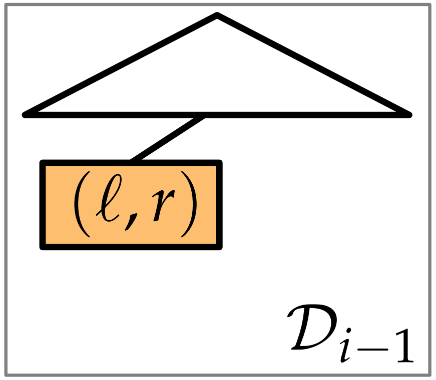
The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



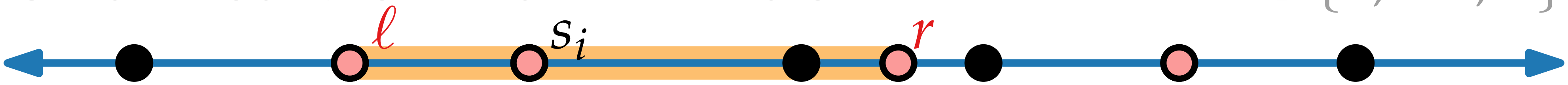
$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

– pick an arbitrary point s_i from $S \setminus S_{i-1}$
– locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}
– split interval (ℓ, r) of I_{i-1} containing s_i
– build $\mathcal{D}_i$:



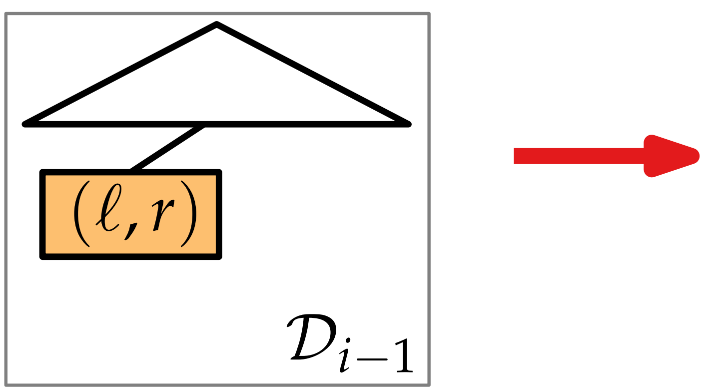
The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



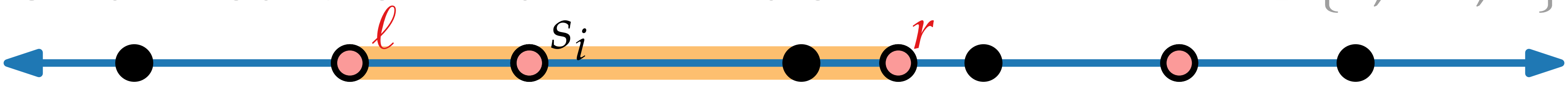
$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, $I_{i-1} :=$ set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

– pick an arbitrary point s_i from $S \setminus S_{i-1}$
– locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}
– split interval (ℓ, r) of I_{i-1} containing s_i
– build $\mathcal{D}_i$:



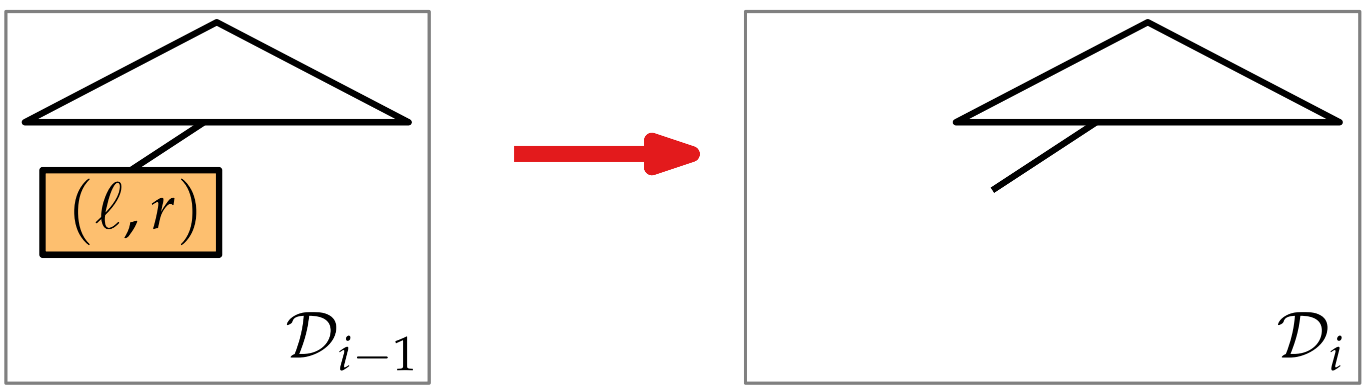
The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



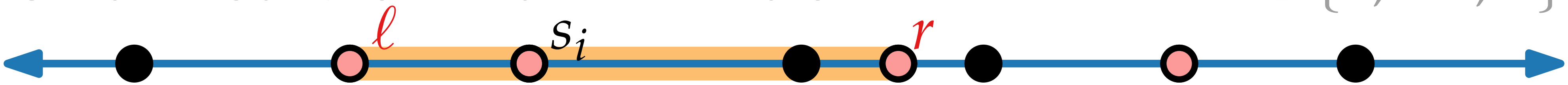
$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, $I_{i-1} :=$ set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

- pick an arbitrary point s_i from $S \setminus S_{i-1}$
- locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}
- split interval (ℓ, r) of I_{i-1} containing s_i
- build $\mathcal{D}_i$:



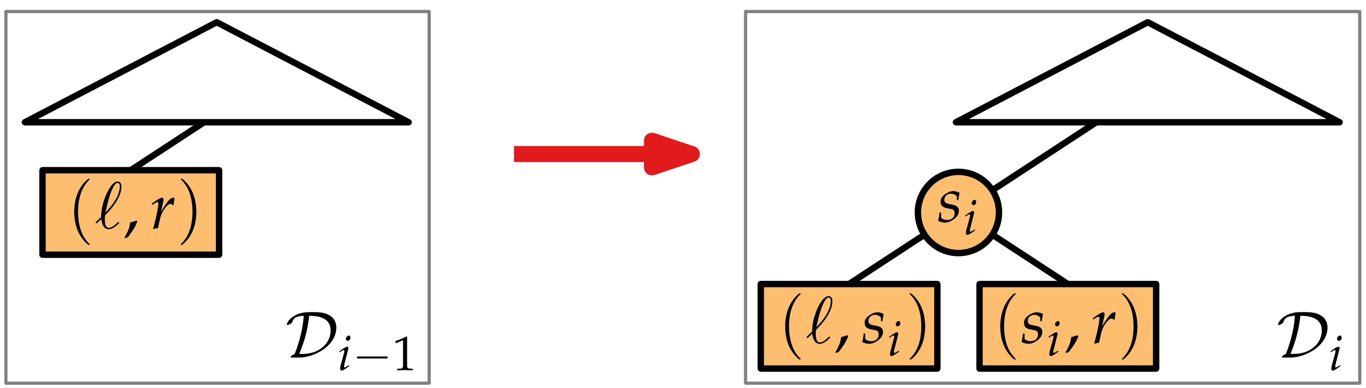
The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



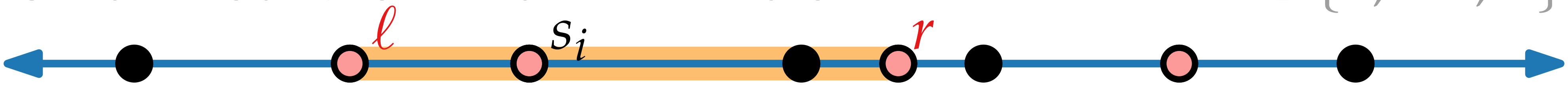
$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

– pick an arbitrary point s_i from $S \setminus S_{i-1}$
– locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}
– split interval (ℓ, r) of I_{i-1} containing s_i
– build $\mathcal{D}_i$:



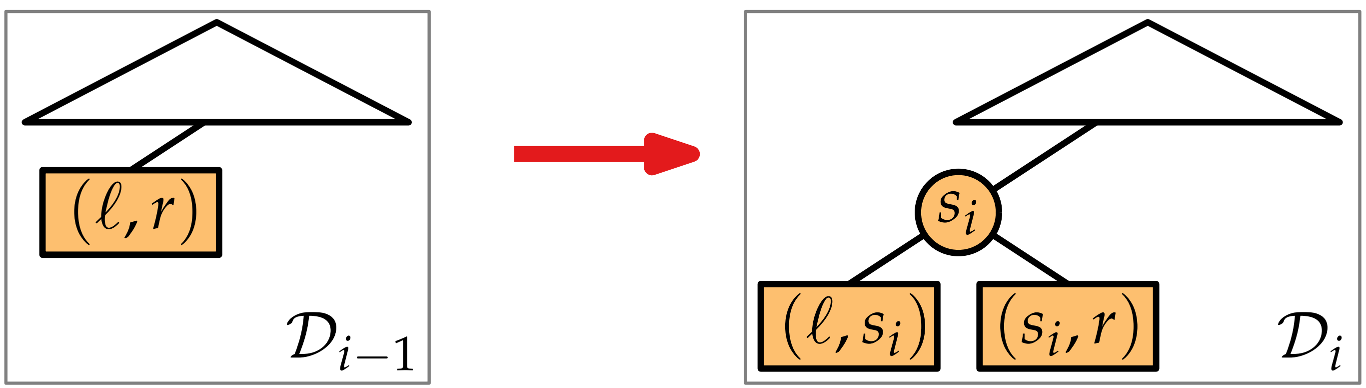
The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, $I_{i-1} :=$ set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

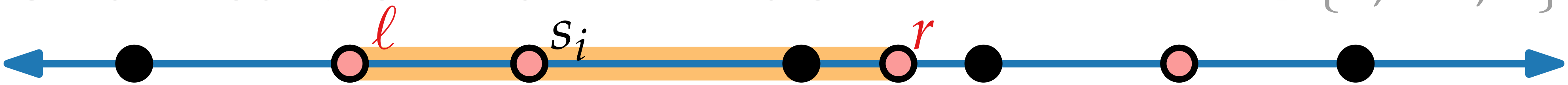
- pick an arbitrary point s_i from $S \setminus S_{i-1}$
- locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}
- split interval (ℓ, r) of I_{i-1} containing s_i
- build $\mathcal{D}_i$:



Problem:

The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



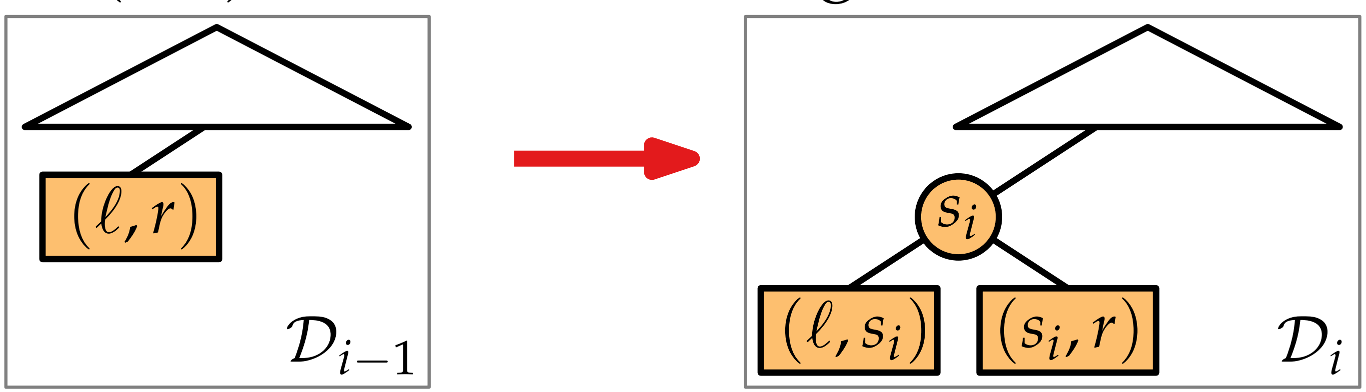
$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

– pick an arbitrary point s_i from $S \setminus S_{i-1}$

– locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}

– split interval (ℓ, r) of I_{i-1} containing s_i

– build $\mathcal{D}_i$:



Problem: *looong* search paths!

The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

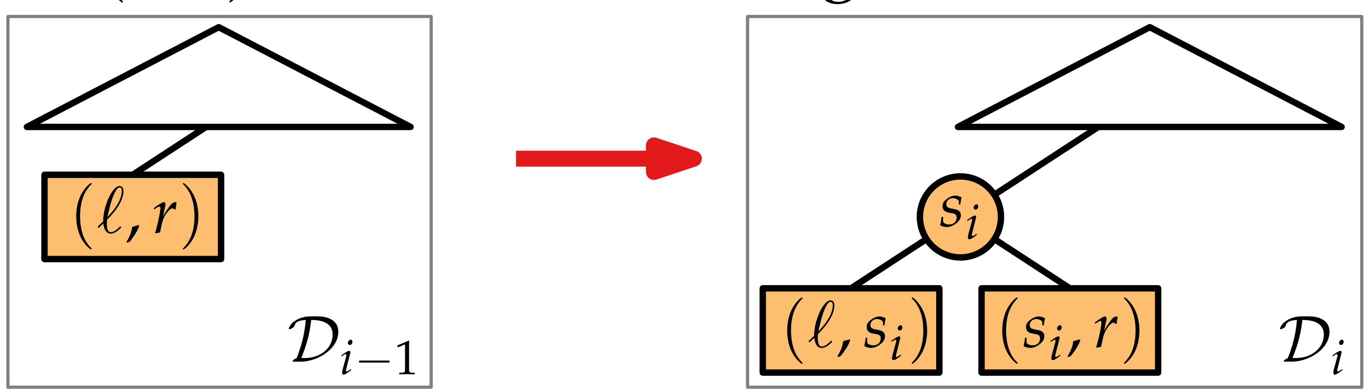
Solution:

– pick an arbitrary point s_i from $S \setminus S_{i-1}$

– locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}

– split interval (ℓ, r) of I_{i-1} containing s_i

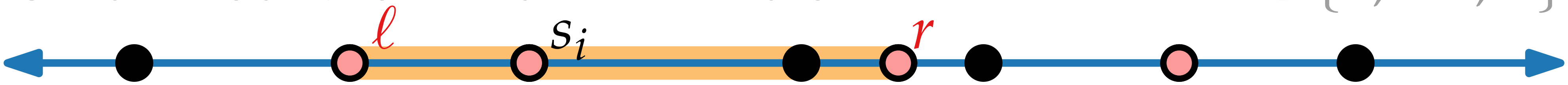
– build $\mathcal{D}_i$:



Problem: *looong* search paths!

The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

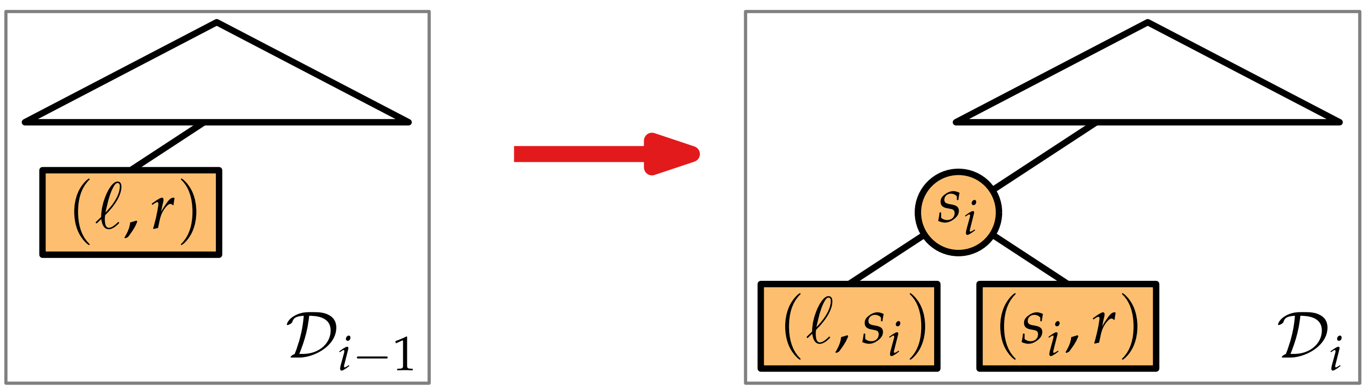
Solution:

– pick ~~an arbitrary~~ point s_i from $S \setminus S_{i-1}$

– locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}

– split interval (ℓ, r) of I_{i-1} containing s_i

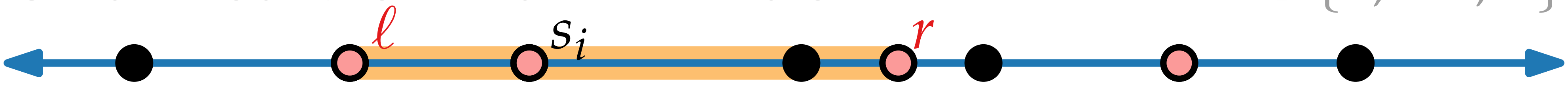
– build $\mathcal{D}_i$:



Problem: *looong* search paths!

The 1D Problem

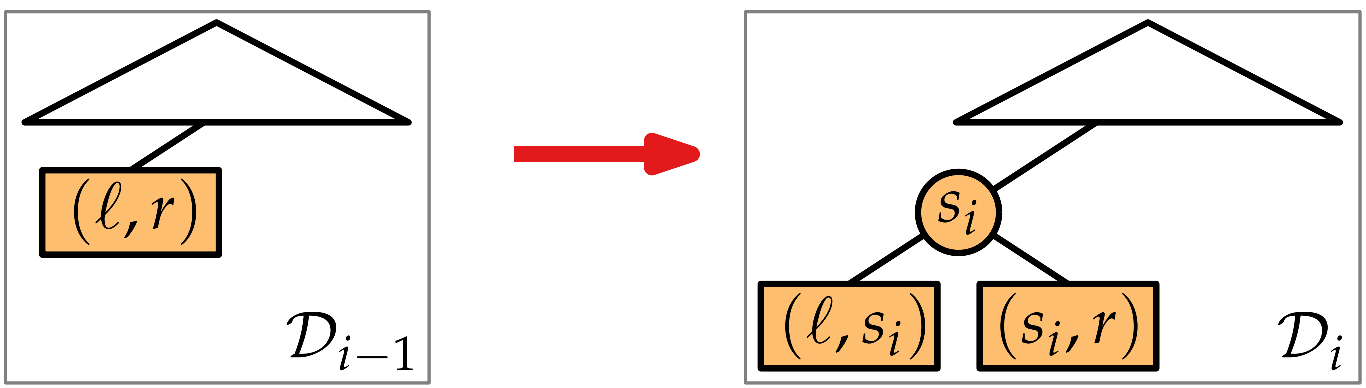
Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

Solution: *random!*

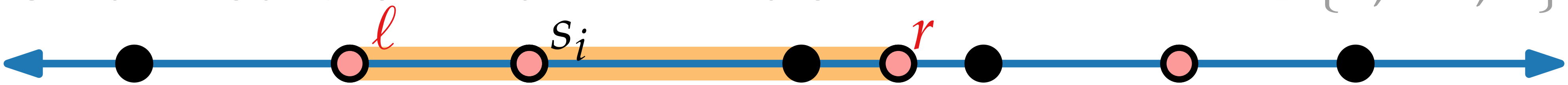
- pick ~~an arbitrary~~ point s_i from $S \setminus S_{i-1}$
- locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}
- split interval (ℓ, r) of I_{i-1} containing s_i
- build $\mathcal{D}_i$:



Problem: *looong* search paths!

The 1D Problem

Given a set S of n real numbers... $i \in \{1, \ldots, n\}$



$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

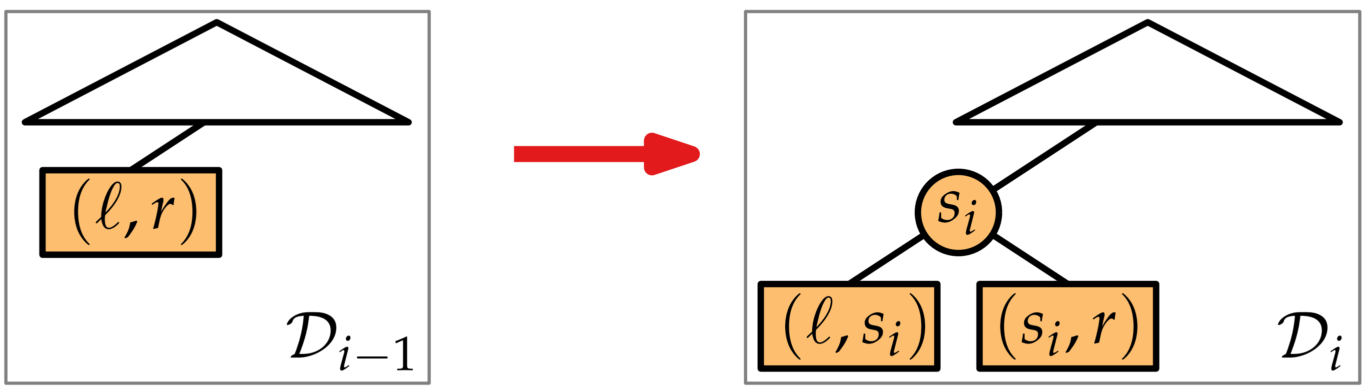
Solution: *random!*

– pick ~~an arbitrary~~ point s_i from $S \setminus S_{i-1}$

– locate s_i in the search structure $\mathcal{D}_{i-1}$ of S_{i-1}

– split interval (ℓ, r) of I_{i-1} containing s_i

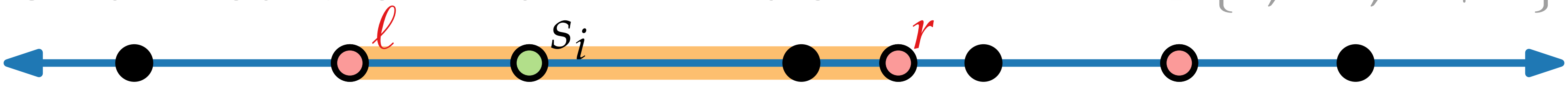
– build $\mathcal{D}_i$:



~~**Problem:** *looong* search paths!~~

The 1D Result

Given a set S of n real numbers... $i \in \{1, \ldots, n+1\}$

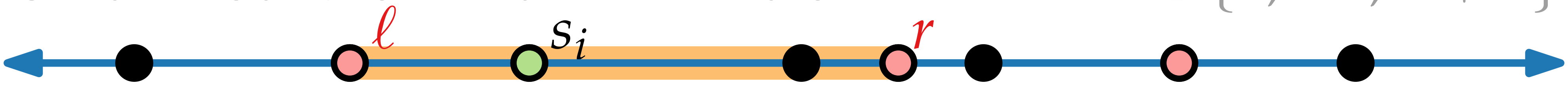


$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

Thm. The randomized-incremental algorithm preproc. a set S of n reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

The 1D Result

Given a set S of n real numbers... $i \in \{1, \ldots, n+1\}$



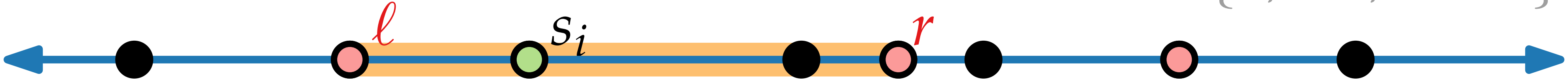
$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

Thm. The randomized-incremental algorithm preproc. a set S of n reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

Proof. Let $q \in \mathbb{R}$ (wlog. $q \notin S$) and $I_i(q) = \arg\{I \in I_i : q \in I\}$.

The 1D Result

Given a set S of n real numbers... $i \in \{1, \ldots, n+1\}$



$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

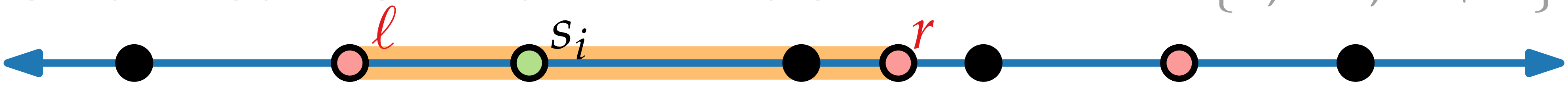
Thm. The randomized-incremental algorithm preproc. a set S of n reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

Proof. Let $q \in \mathbb{R}$ (wlog. $q \notin S$) and $I_i(q) = \arg\{I \in I_i : q \in I\}$.

$E[\text{query time in } \mathcal{D}_n] =$

The 1D Result

Given a set S of n real numbers... $i \in \{1, \ldots, n+1\}$



$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

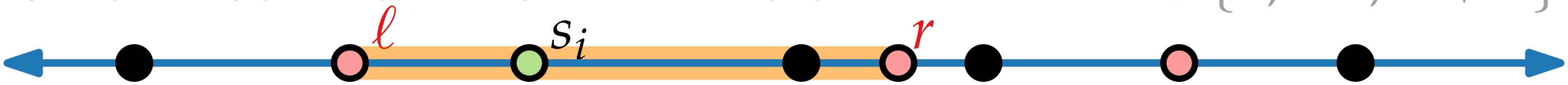
Thm. The randomized-incremental algorithm preproc. a set S of n reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

Proof. Let $q \in \mathbb{R}$ (wlog. $q \notin S$) and $I_i(q) = \arg\{I \in I_i : q \in I\}$.

E[query time in $\mathcal{D}_n$] $=$ E[length search path in $\mathcal{D}_n$] $=$

The 1D Result

Given a set S of n real numbers... $i \in \{1, \dots, n+1\}$



$S_{i-1} := \{s_1, \dots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

Thm. The randomized-incremental algorithm preproc. a set S of n reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

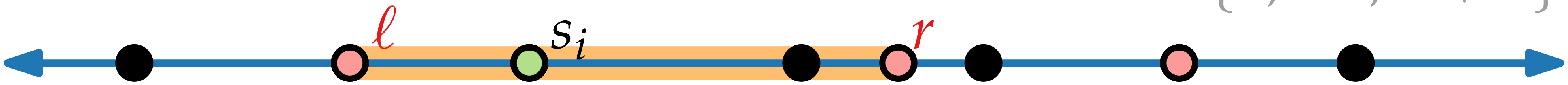
Proof. Let $q \in \mathbb{R}$ (wlog. $q \notin S$) and $I_i(q) = \arg\{I \in I_i : q \in I\}$.

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

The 1D Result

Given a set S of n real numbers... $i \in \{1, \ldots, n+1\}$



$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

Thm. The randomized-incremental algorithm preproc. a set S of n reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

Proof. Let $q \in \mathbb{R}$ (wlog. $q \notin S$) and $I_i(q) = \arg\{I \in I_i : q \in I\}$.

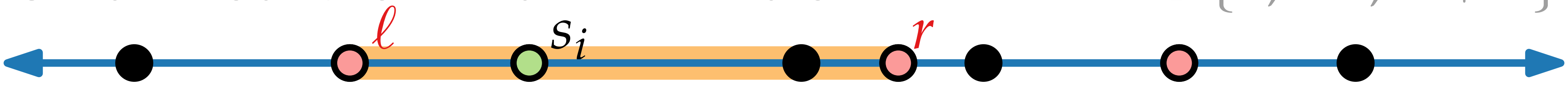
Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] =$

The 1D Result

Given a set S of n real numbers... $i \in \{1, \dots, n+1\}$



$S_{i-1} := \{s_1, \dots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

Thm. The randomized-incremental algorithm preproc. a set S of n reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

Proof. Let $q \in \mathbb{R}$ (wlog. $q \notin S$) and $I_i(q) = \arg\{I \in I_i : q \in I\}$.

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \textbf{?}$

Expected Query Time of $\mathcal{D}_n$

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] =$ **?**

Expected Query Time of $\mathcal{D}_n$

$E[X_i] =$

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] =$ **?**

Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] =$

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] =$ **?**

Expected Query Time of $\mathcal{D}_n$

$$E[X_i] = P[X_i = 1] =$$
$$= \text{probability that } I_i(q) \neq I_{i-1}(q)$$

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$$
$$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \textbf{?}$$

Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] =$

$= \text{probability that } I_i(q) \neq I_{i-1}(q)\text{, i.e., } s_i \in I_{i-1}(q).$

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \textbf{?}$

Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] =$

$= \text{probability that } I_i(q) \neq I_{i-1}(q) \text{, i.e., } s_i \in I_{i-1}(q).$

Backwards analysis:

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] =$?

Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] =$

$= \text{probability that } I_i(q) \neq I_{i-1}(q)$, i.e., $s_i \in I_{i-1}(q)$.

Backwards analysis:

Consider S_i fixed.

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] =$ **?**

Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] =$

$= \text{probability that } I_i(q) \neq I_{i-1}(q)\text{, i.e., } s_i \in I_{i-1}(q).$

Backwards analysis:

Consider S_i fixed.

If we *remove* a randomly chosen pt from S_i,

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$

Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] =$

$= \text{probability that } I_i(q) \neq I_{i-1}(q)\text{, i.e., } s_i \in I_{i-1}(q).$

Backwards analysis:

Consider S_i fixed.
If we *remove* a randomly chosen pt from S_i, what's the probability that the interval containing q changes?

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$

Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] =$

$= \text{probability that } I_i(q) \neq I_{i-1}(q)\text{, i.e., } s_i \in I_{i-1}(q).$

Backwards analysis:

Consider S_i fixed.
If we *remove* a randomly chosen pt from S_i, what's the probability that the interval containing q changes?
– we have i choices, identically distributed

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$

Expected Query Time of $\mathcal{D}_n$

$$E[X_i] = P[X_i = 1] =$$
$$= \text{probability that } I_i(q) \neq I_{i-1}(q)\text{, i.e., } s_i \in I_{i-1}(q).$$

Backwards analysis:

Consider S_i fixed.
If we *remove* a randomly chosen pt from S_i, what's the probability that the interval containing q changes?
– we have i choices, identically distributed
– at most two of these change the interval

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$$
$$= E[\textstyle\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \textbf{?}$$

Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] =$

$= \text{probability that } I_i(q) \neq I_{i-1}(q)\text{, i.e., } s_i \in I_{i-1}(q).$

Backwards analysis:

Consider S_i fixed.
If we *remove* a randomly chosen pt from S_i,
what's the probability that the interval
containing q changes?
– we have i choices, identically distributed
– at most two of these change the interval

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] =$ **?**

Expected Query Time of $\mathcal{D}_n$

$$E[X_i] = P[X_i = 1] = 2/i$$

$$= \text{probability that } I_i(q) \neq I_{i-1}(q), \text{ i.e., } s_i \in I_{i-1}(q).$$

Backwards analysis:

Consider S_i fixed.
If we *remove* a randomly chosen pt from S_i, what's the probability that the interval containing q changes?
– we have i choices, identically distributed
– at most two of these change the interval

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$$

$$= E[\textstyle\sum_{i=1}^n X_i] = \sum_{i=1}^n E[X_i] = \textbf{?}$$

Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] = 2/i$
$= \text{probability that } I_i(q) \neq I_{i-1}(q)\text{, i.e., } s_i \in I_{i-1}(q).$

Backwards analysis:

Consider S_i fixed.
If we *remove* a randomly chosen pt from S_i, what's the probability that the interval containing q changes?
– we have i choices, identically distributed
– at most two of these change the interval

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$
$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] =$?

Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] = 2/i$

$= \text{probability that } I_i(q) \neq I_{i-1}(q)\text{, i.e., } s_i \in I_{i-1}(q).$

Backwards analysis:

Consider S_i fixed.
If we *remove* a randomly chosen pt from S_i, what's the probability that the interval containing q changes?
– we have i choices, identically distributed
– at most two of these change the interval

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

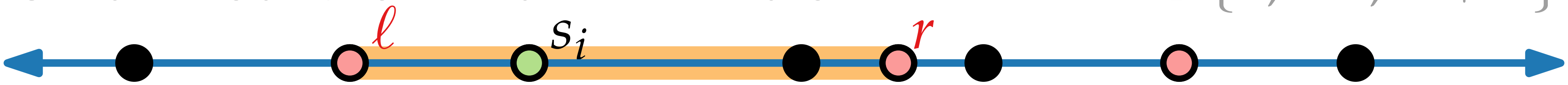
$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$

$O(\log n)$

The 1D Result

Given a set S of n real numbers... $i \in \{1, \ldots, n+1\}$



$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, I_{i-1} := set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

Thm. The randomized-incremental algorithm preproc. a set S of n reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

The 2D Problem

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

The 2D Problem

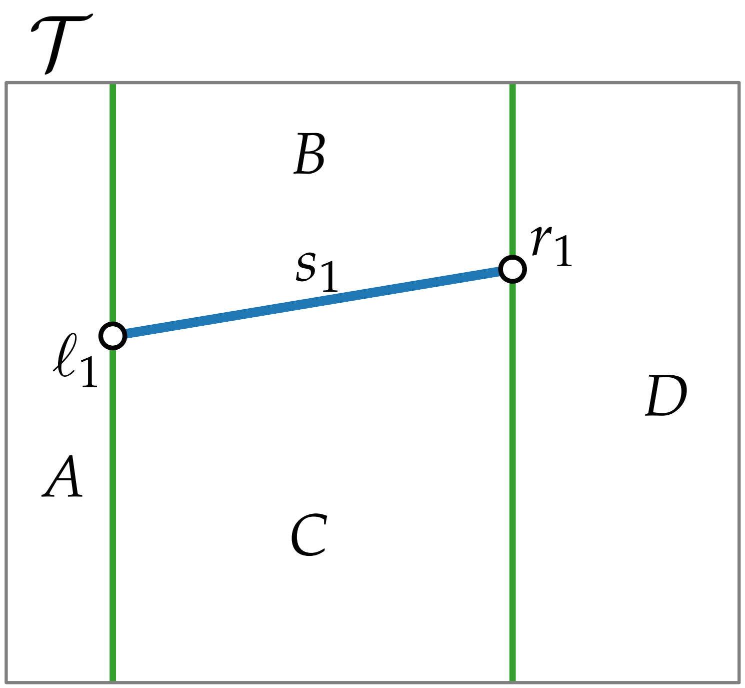
trapezoidal map →

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

The 2D Problem

trapezoidal map

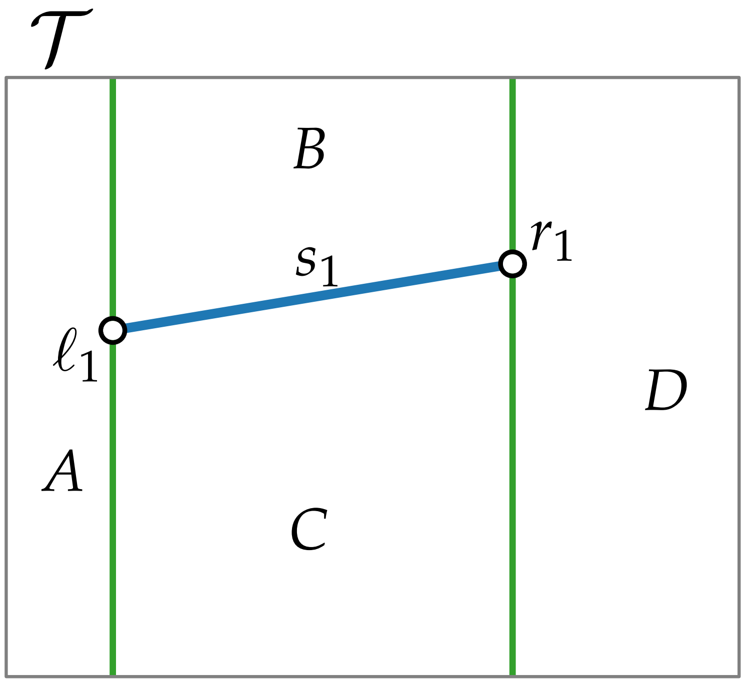
Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$



The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

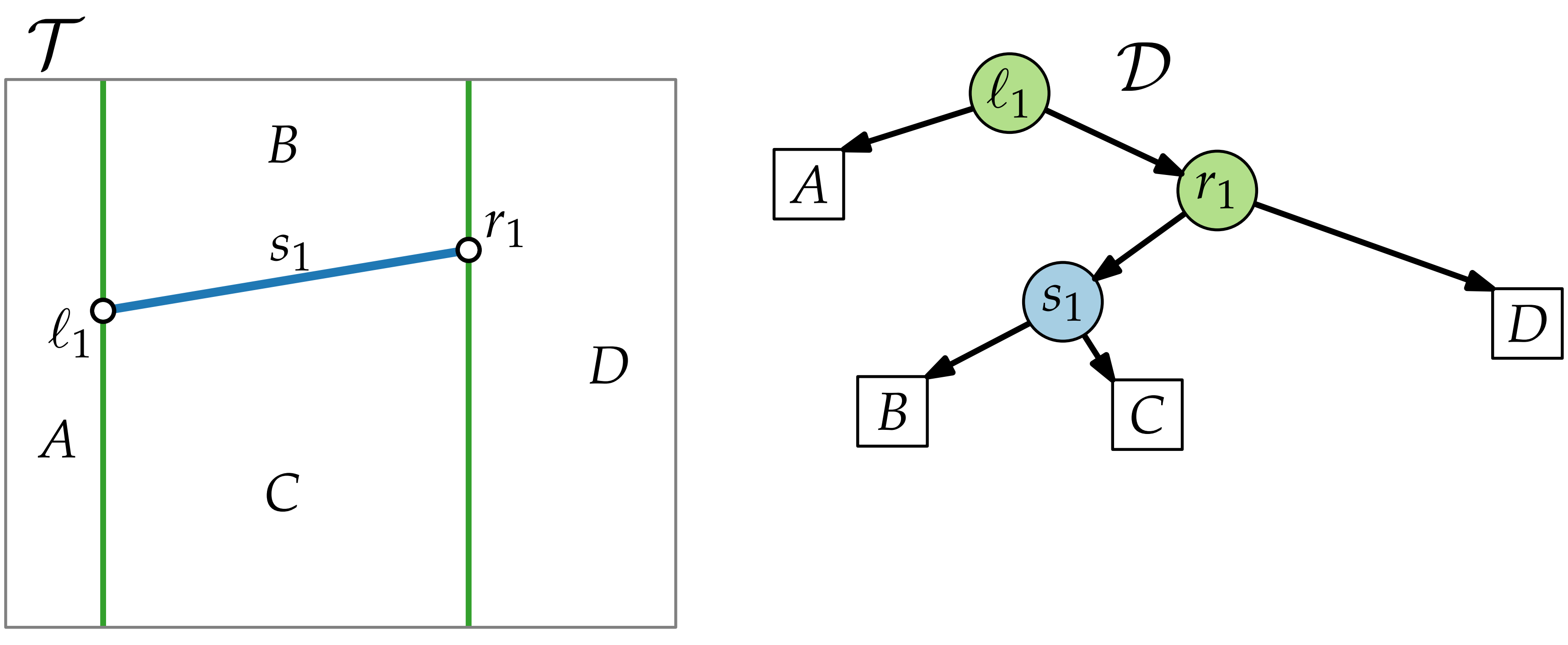


The 2D Problem

point-location data structure (DAG)

trapezoidal map

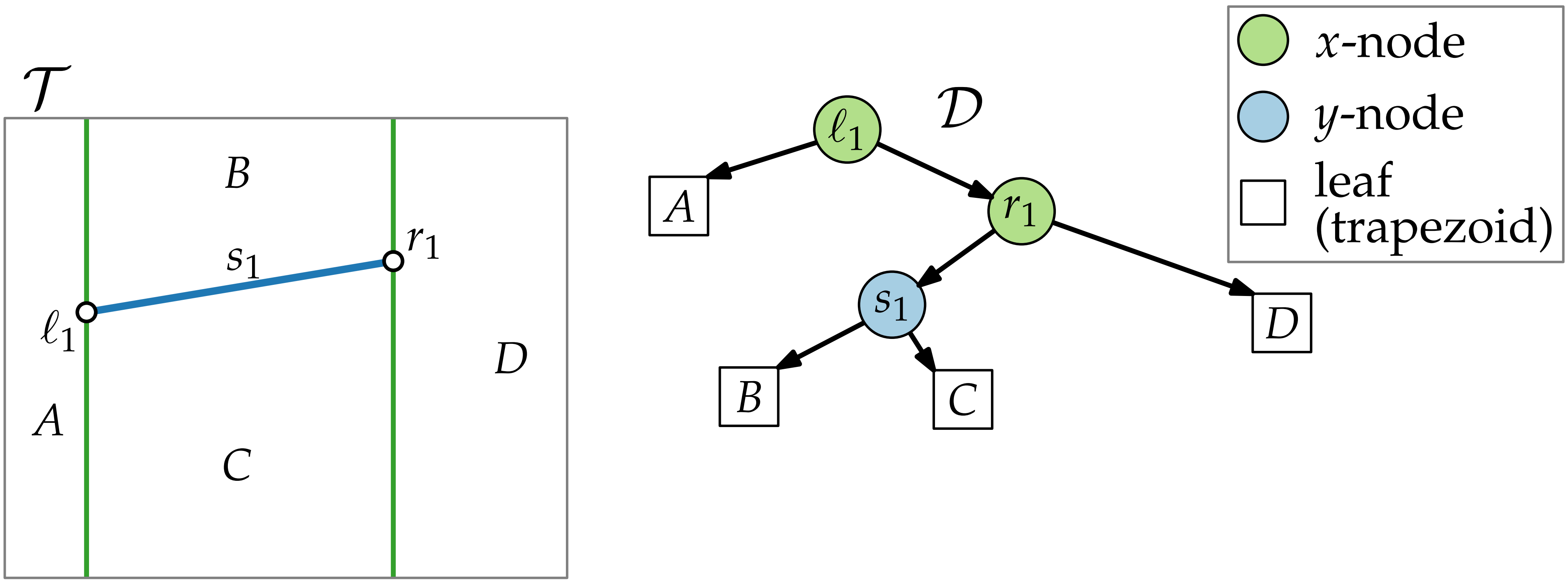
Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$



The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

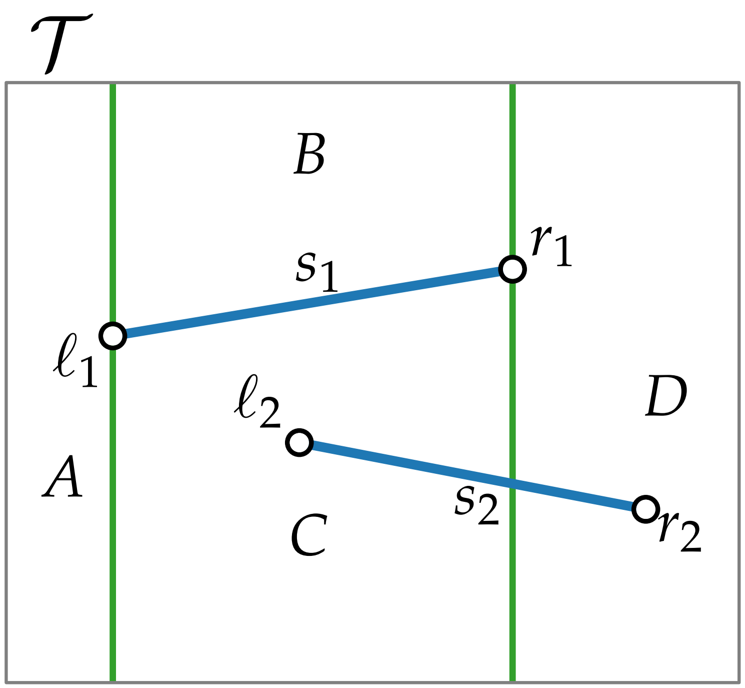


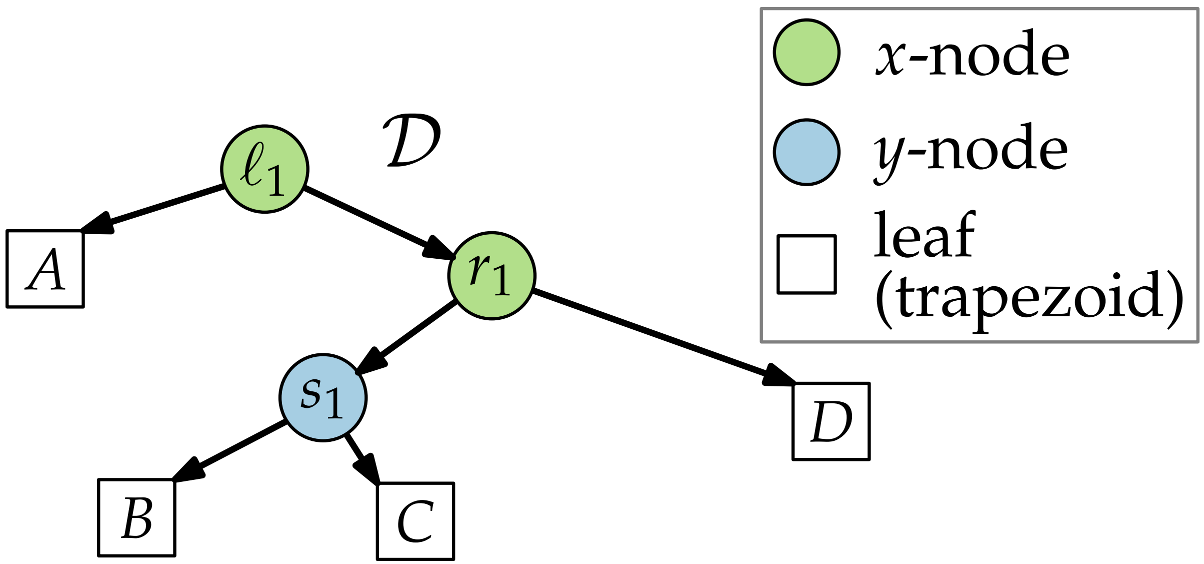
The 2D Problem

point-location data structure (DAG)

trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

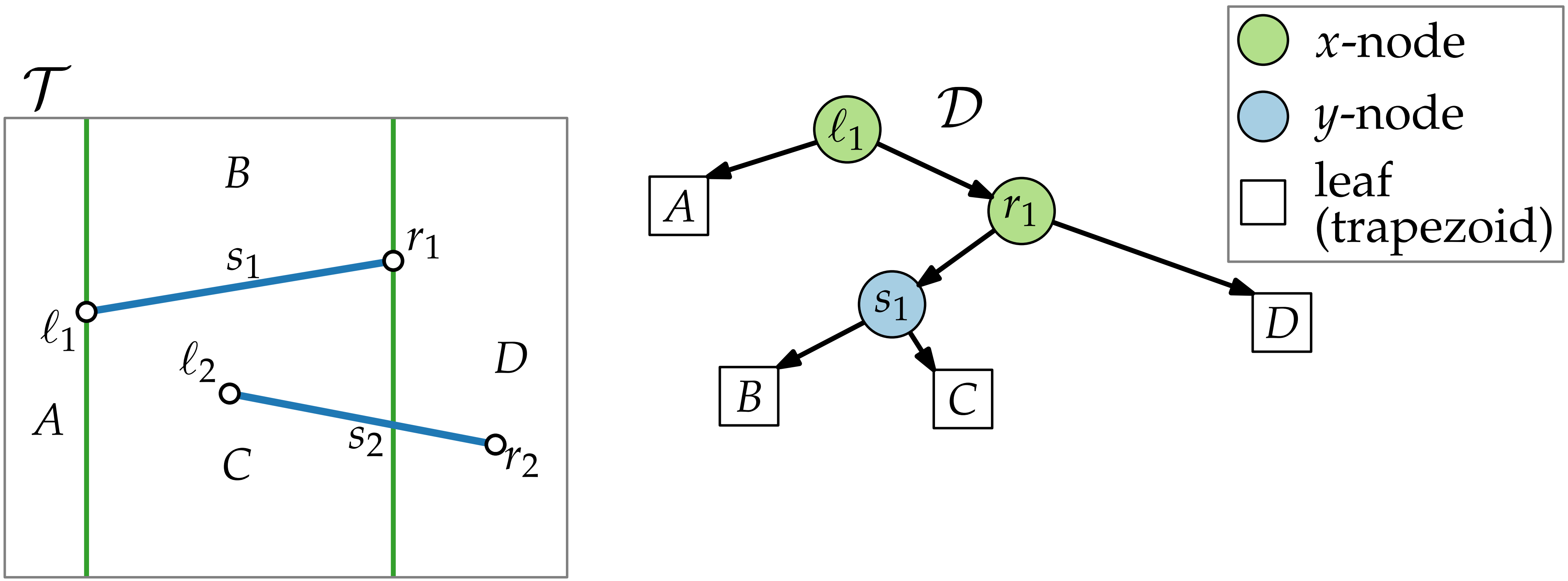




The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$
– use $\mathcal{D}$ to locate left endpoint of next segment s



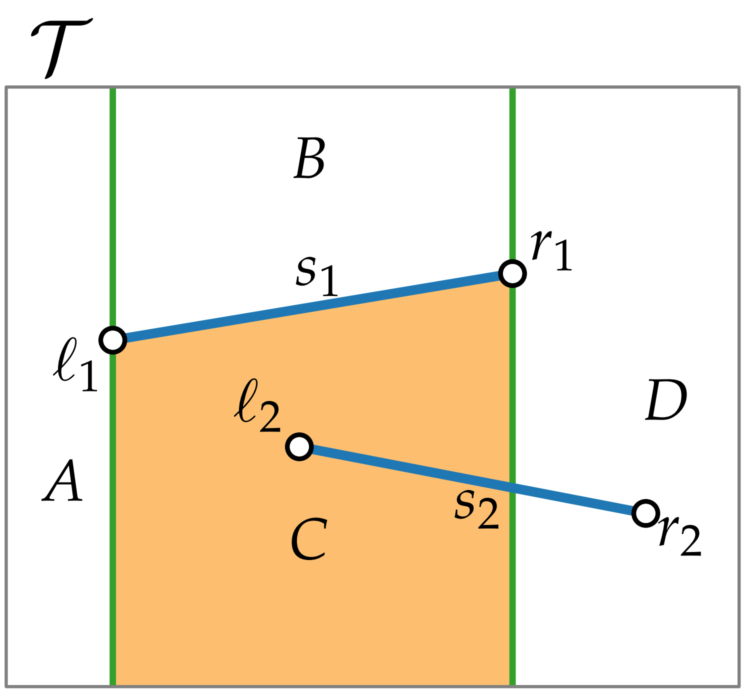
The 2D Problem

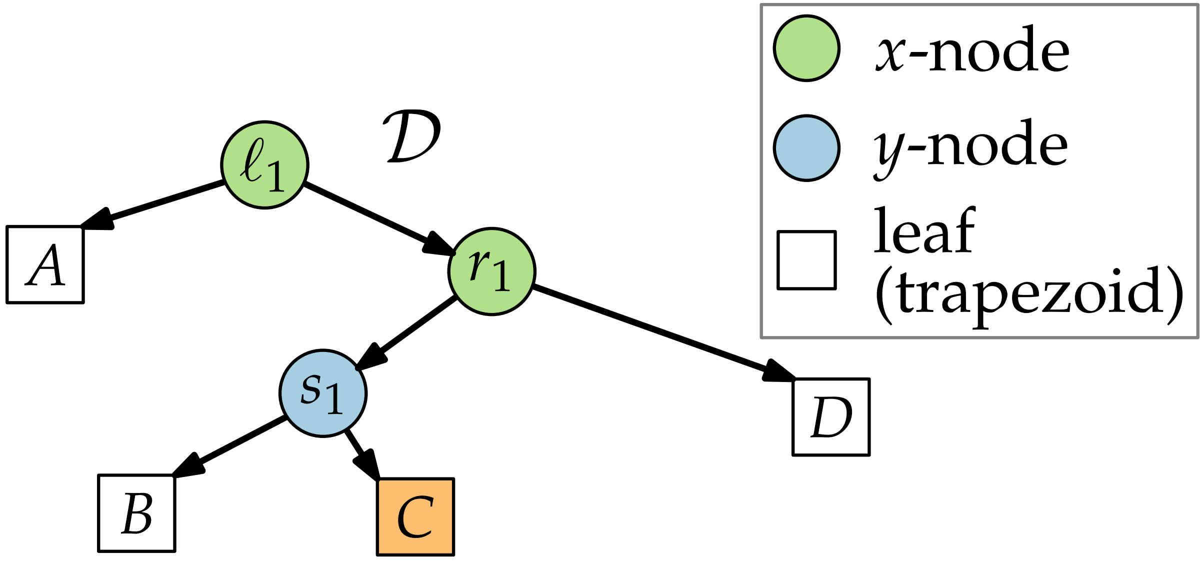
point-location data structure (DAG)

trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

– use $\mathcal{D}$ to locate left endpoint of next segment s

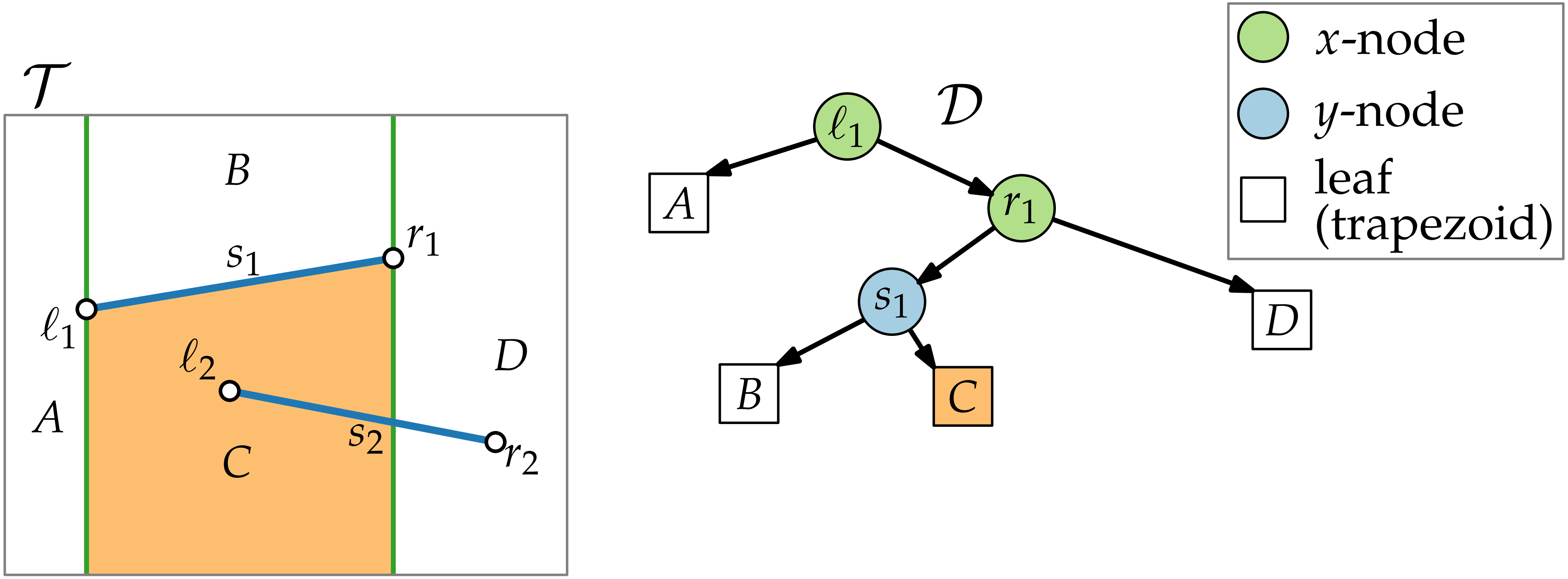




The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$
– use $\mathcal{D}$ to locate left endpoint of next segment s
– "walk" along s through $\mathcal{T}$



The 2D Problem

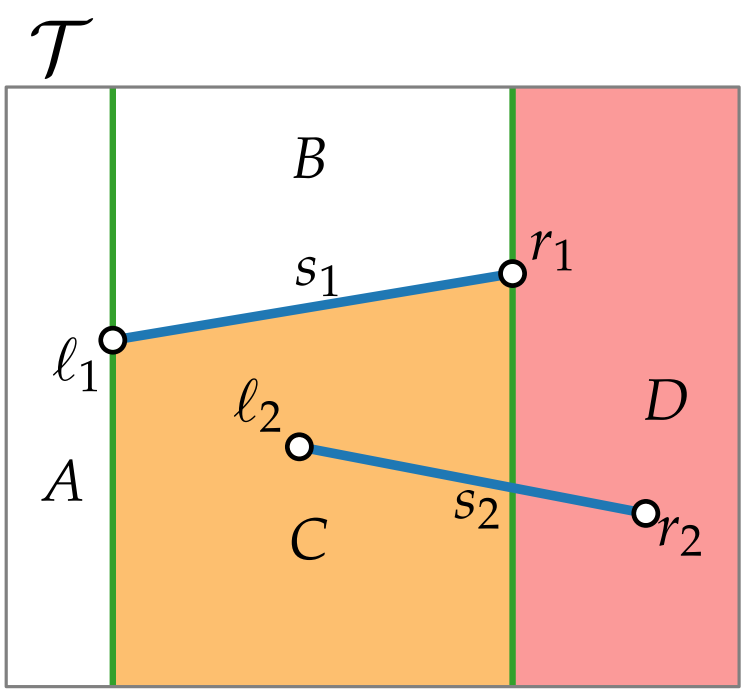
point-location data structure (DAG)

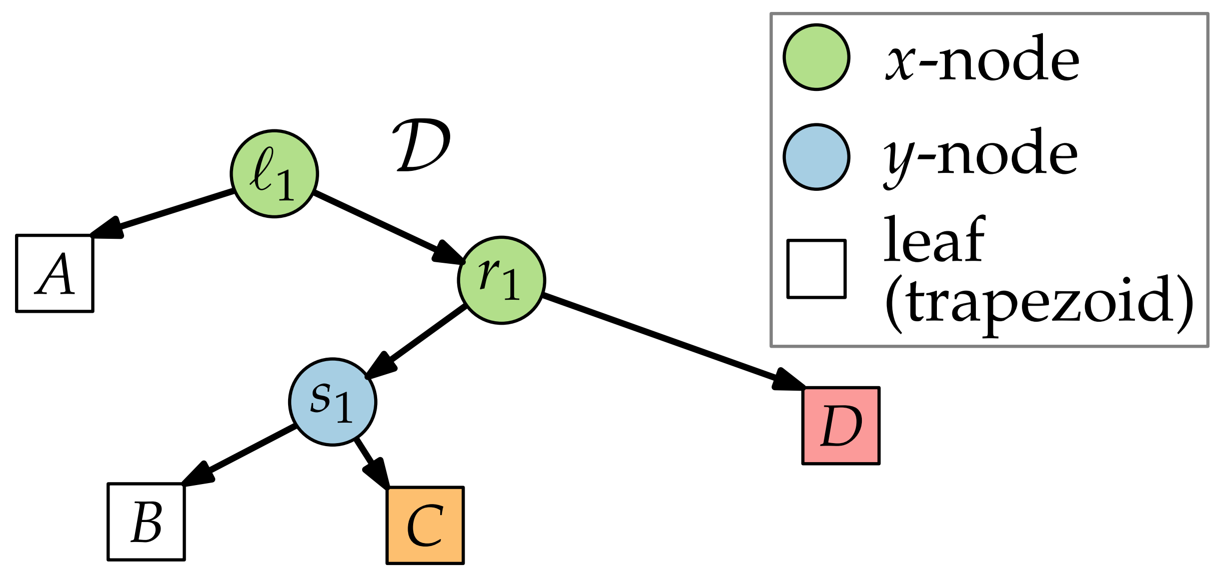
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

– use $\mathcal{D}$ to locate left endpoint of next segment s

– "walk" along s through $\mathcal{T}$



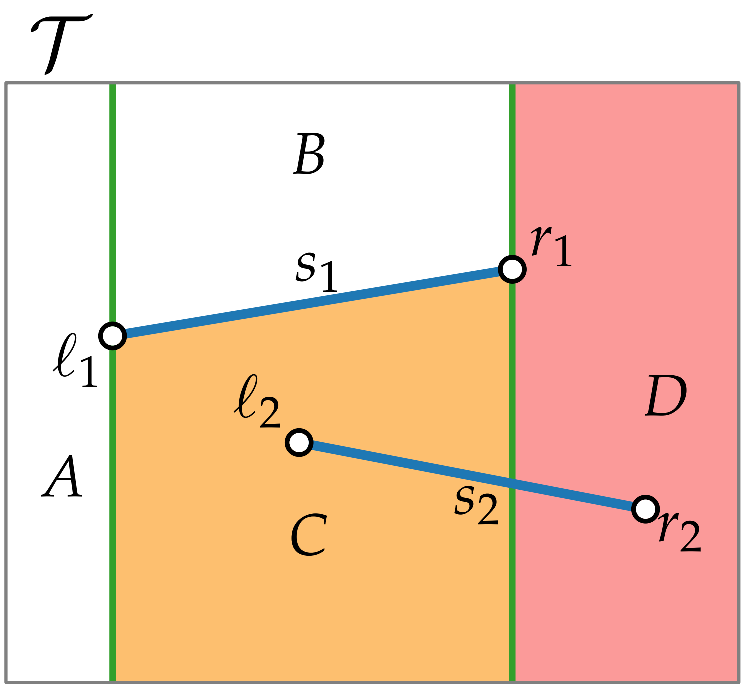


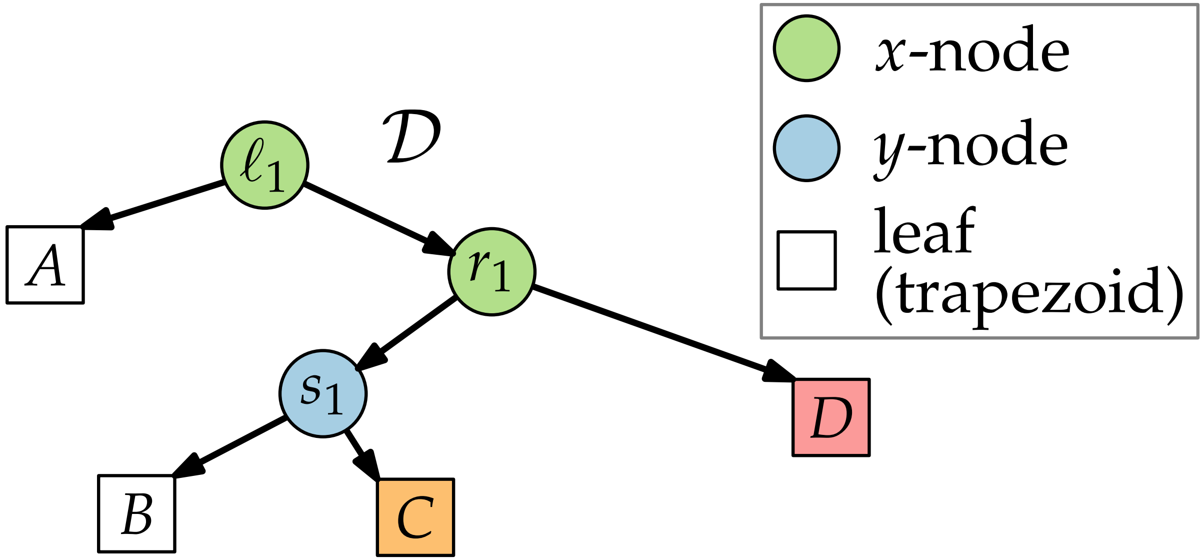
The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

– use $\mathcal{D}$ to locate left endpoint of next segment s
– "walk" along s through $\mathcal{T}$
– destroy all trapezoids of $\mathcal{T}$ intersecting s



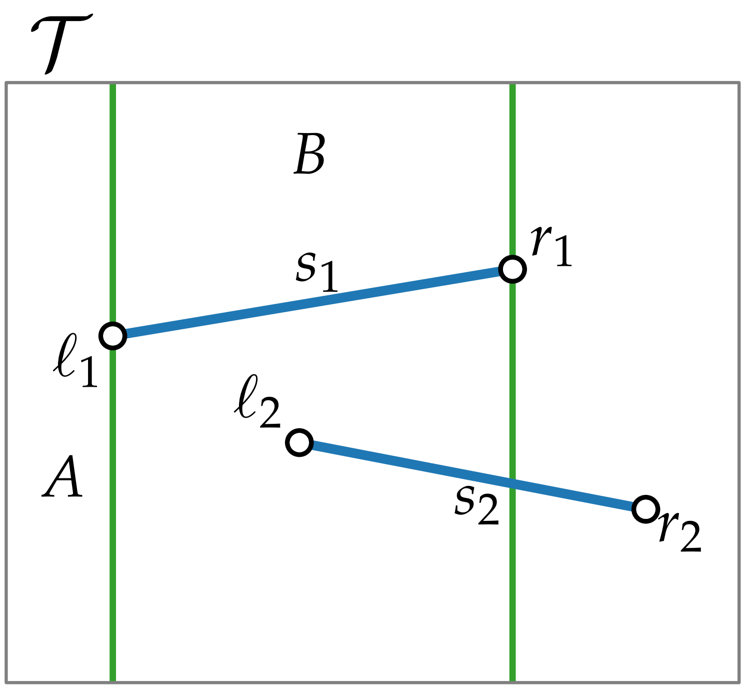


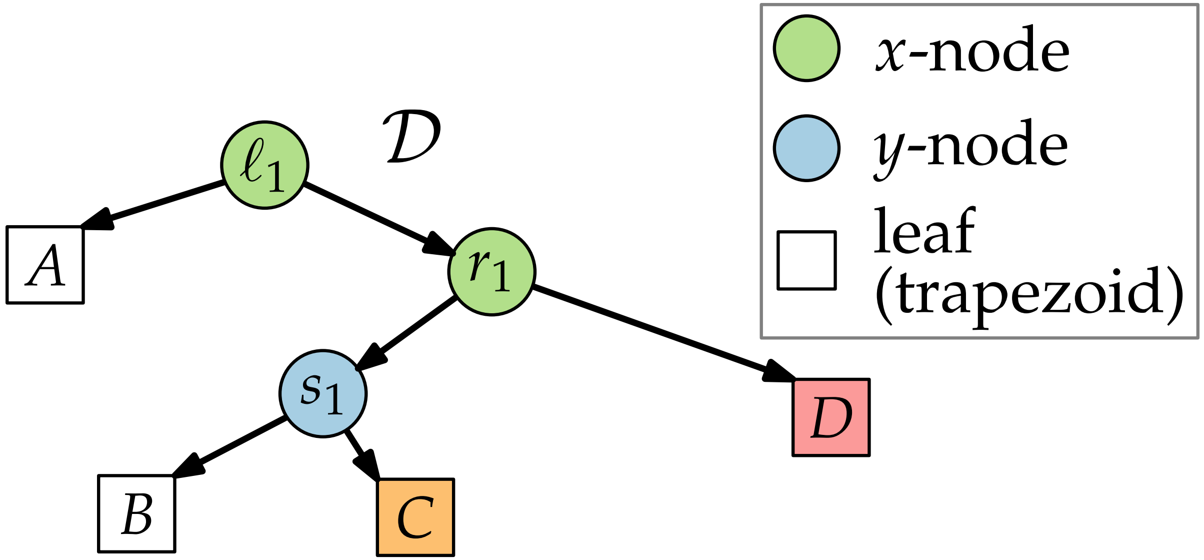
The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s



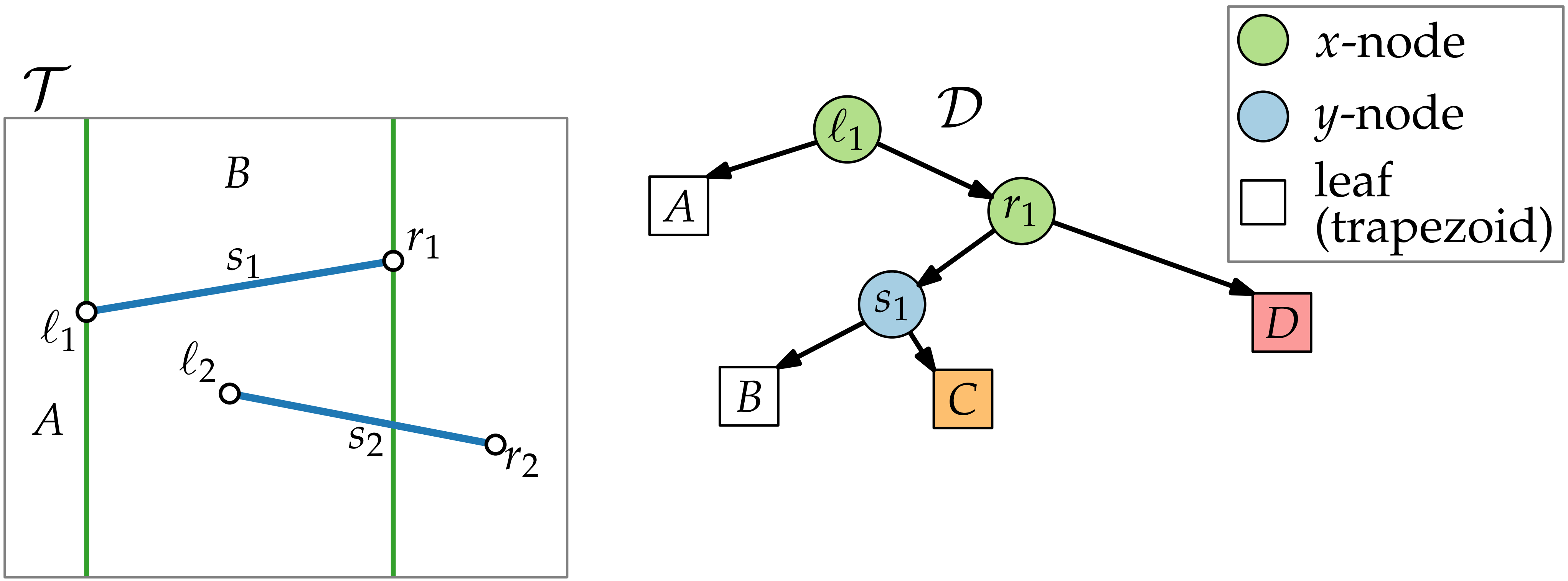


The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)

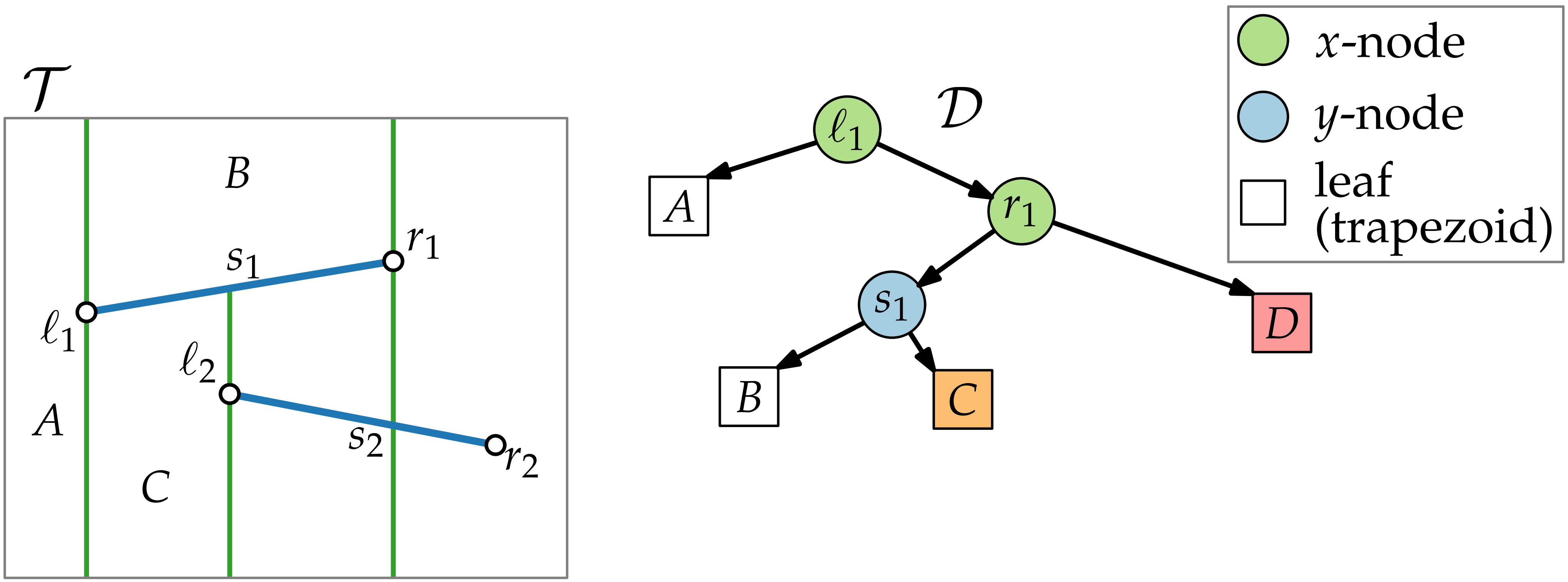


The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)

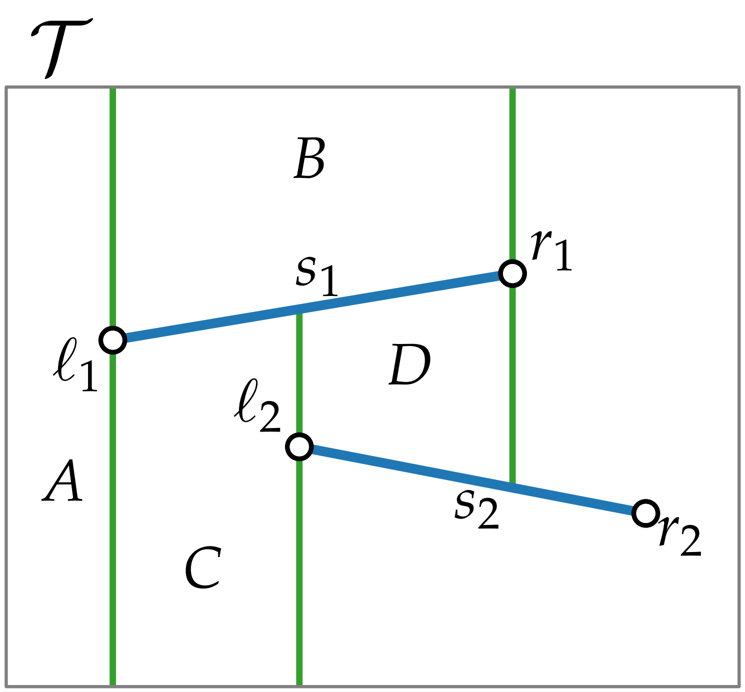


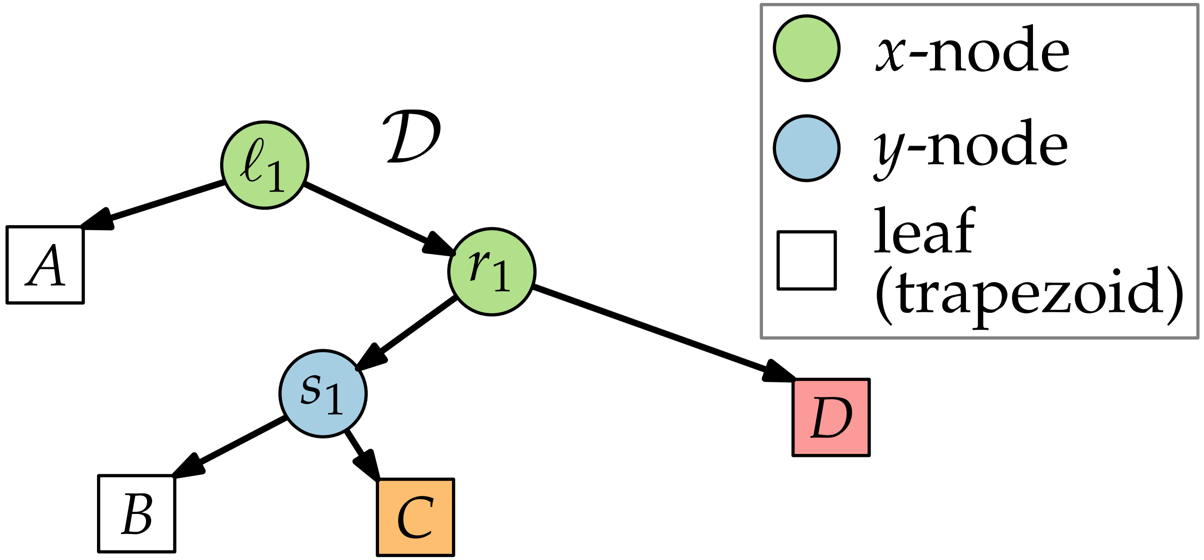
The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)



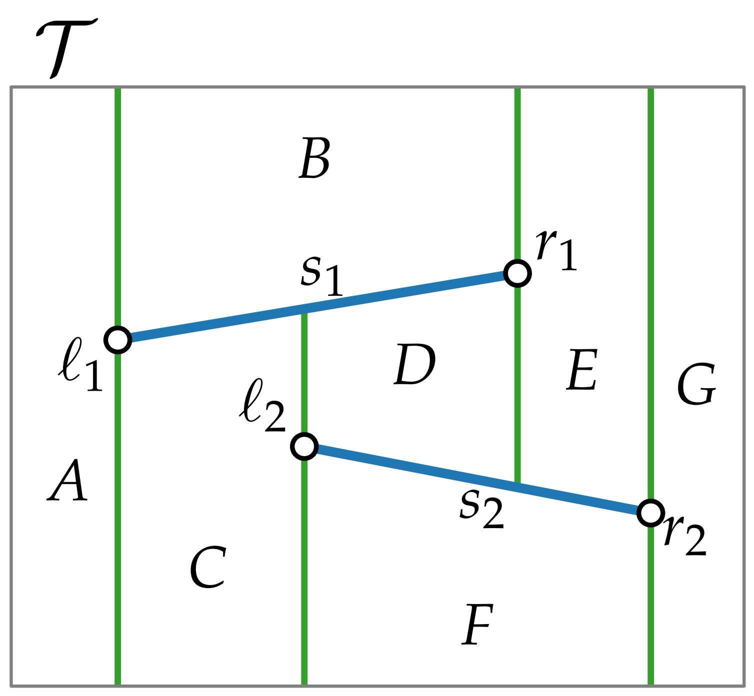


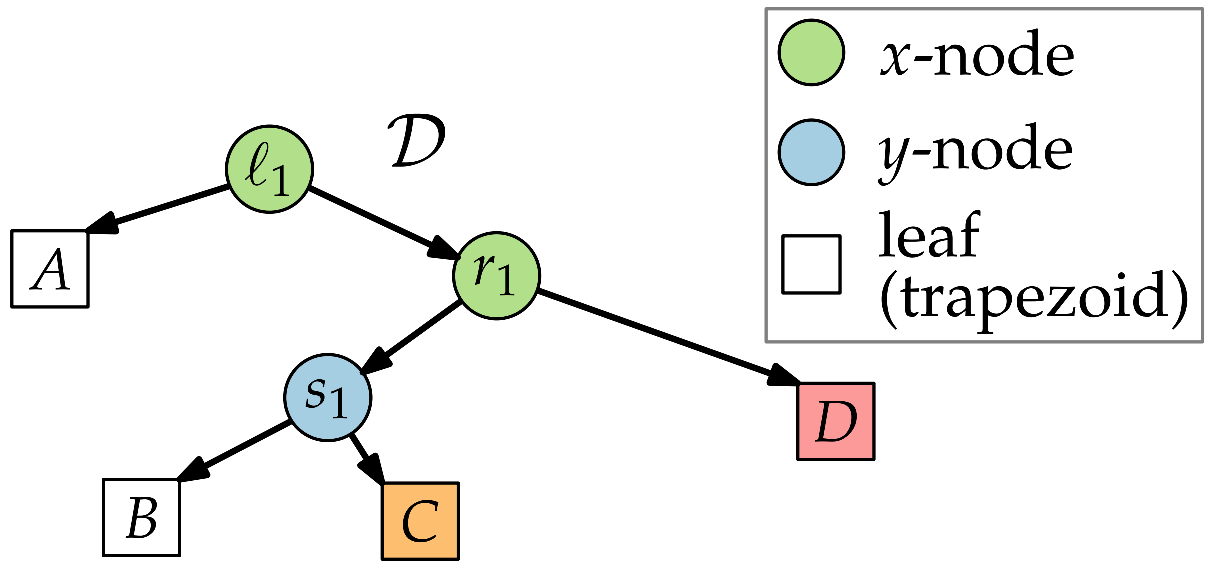
The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)



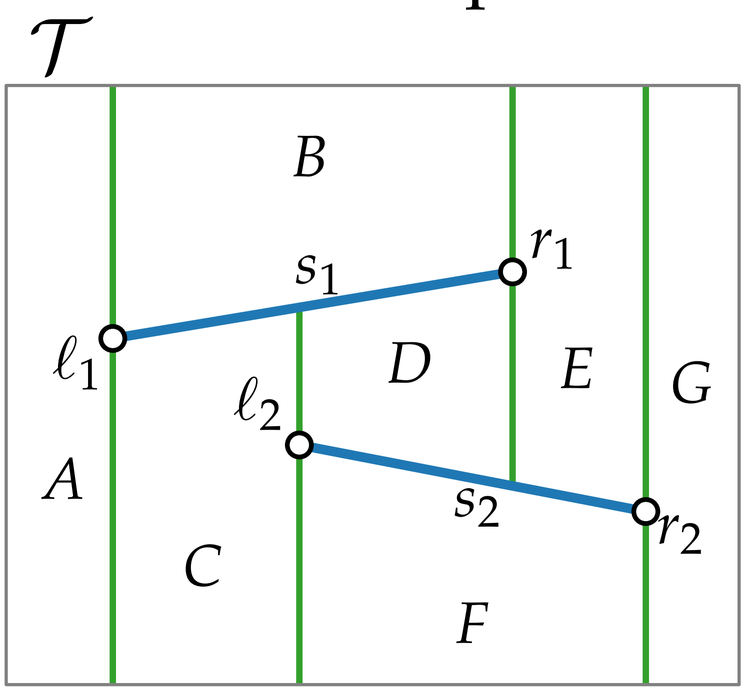


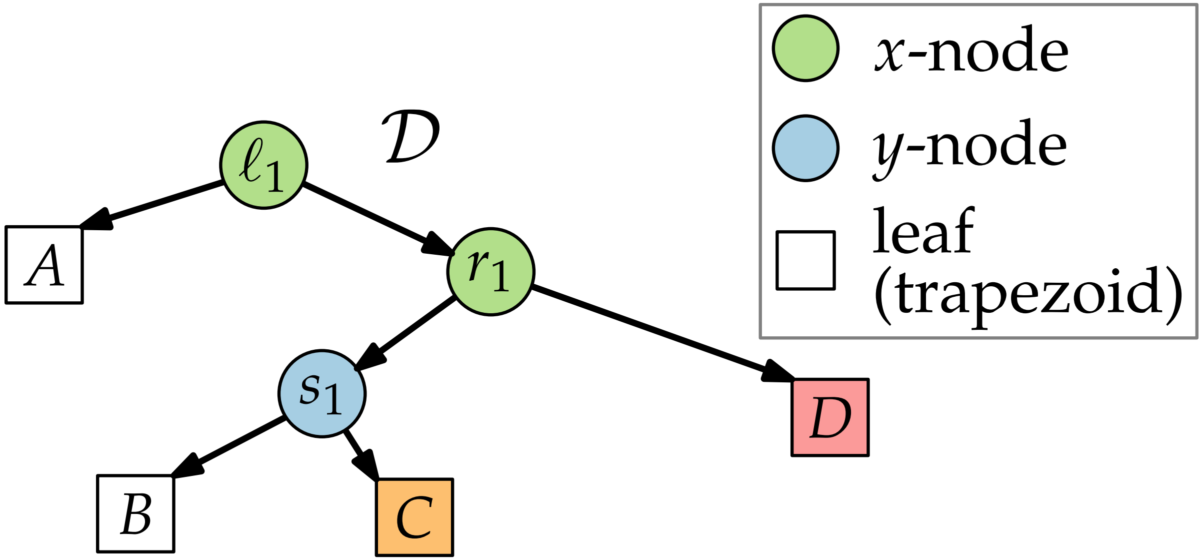
The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$





The 2D Problem

point-location data structure (DAG)

trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

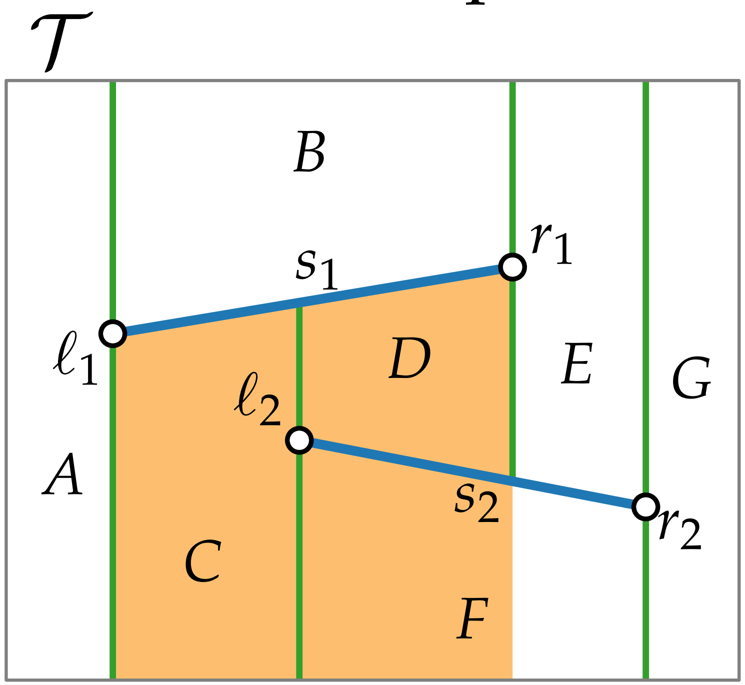
– use $\mathcal{D}$ to locate left endpoint of next segment s

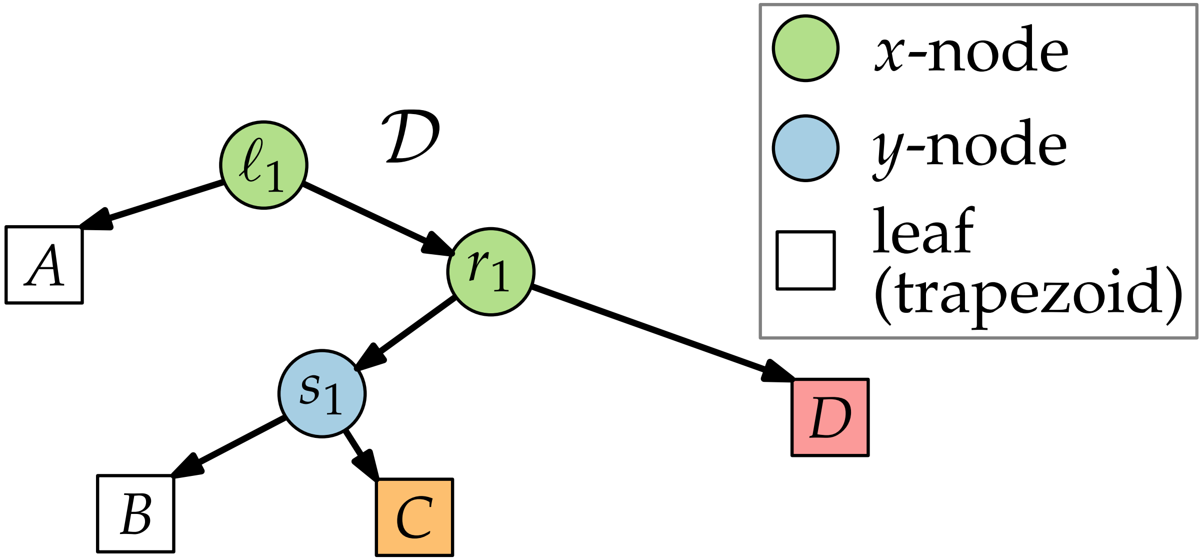
– "walk" along s through $\mathcal{T}$

– destroy all trapezoids of $\mathcal{T}$ intersecting s

– construct new trapezoids of $\mathcal{T}$ (adjacent to s)

– update $\mathcal{D}$





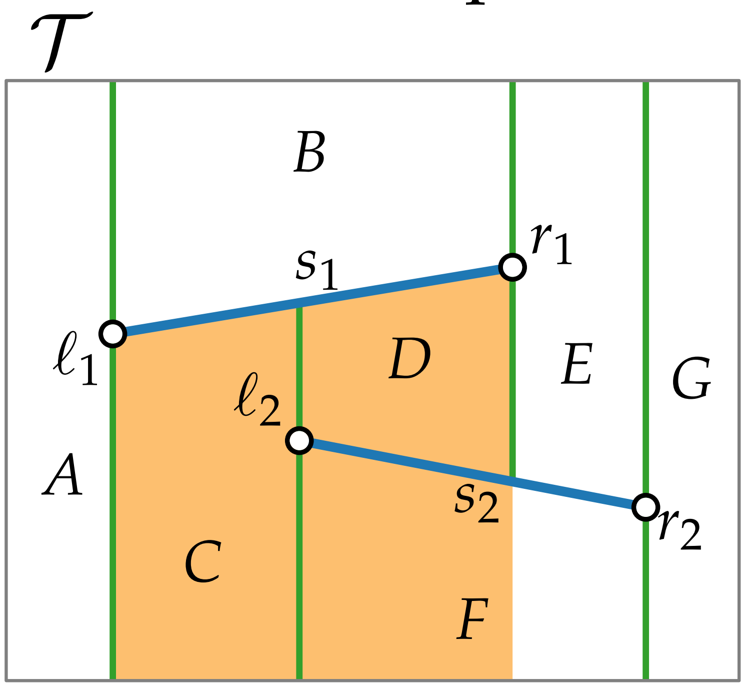
The 2D Problem

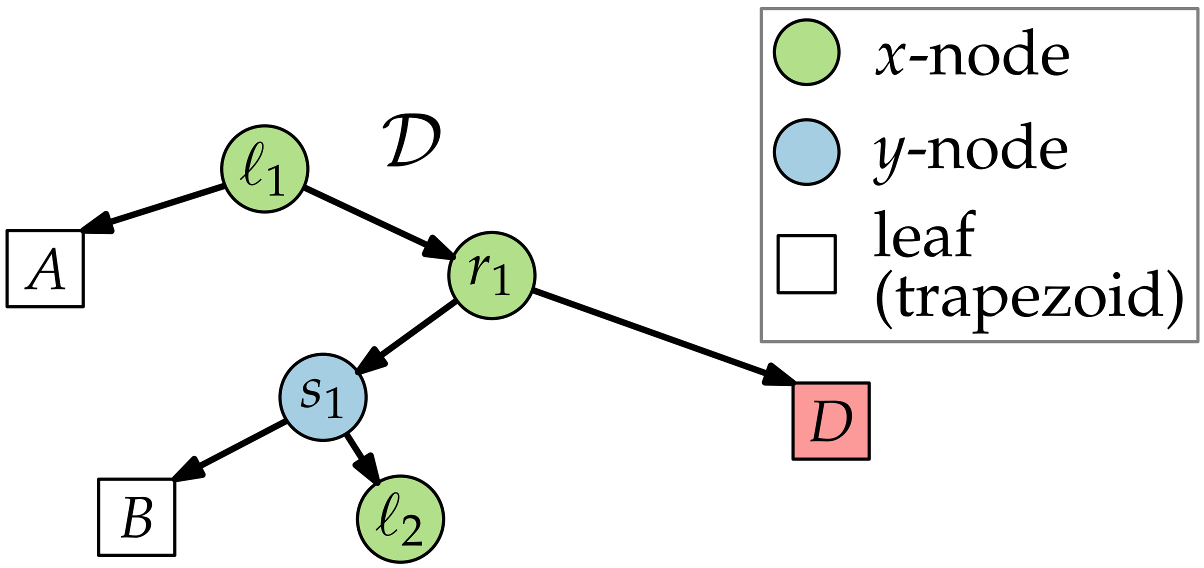
point-location data structure (DAG)

trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$





The 2D Problem

point-location data structure (DAG)

trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

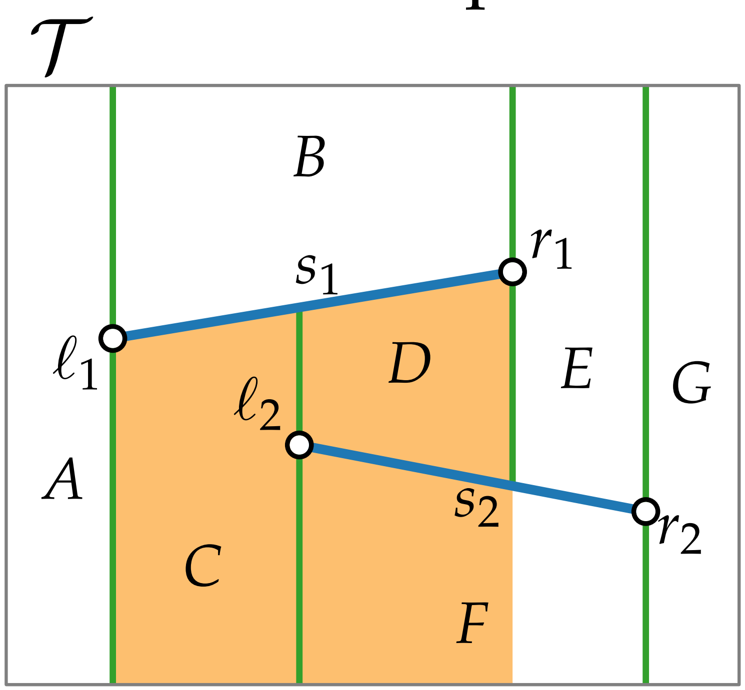
– use $\mathcal{D}$ to locate left endpoint of next segment s

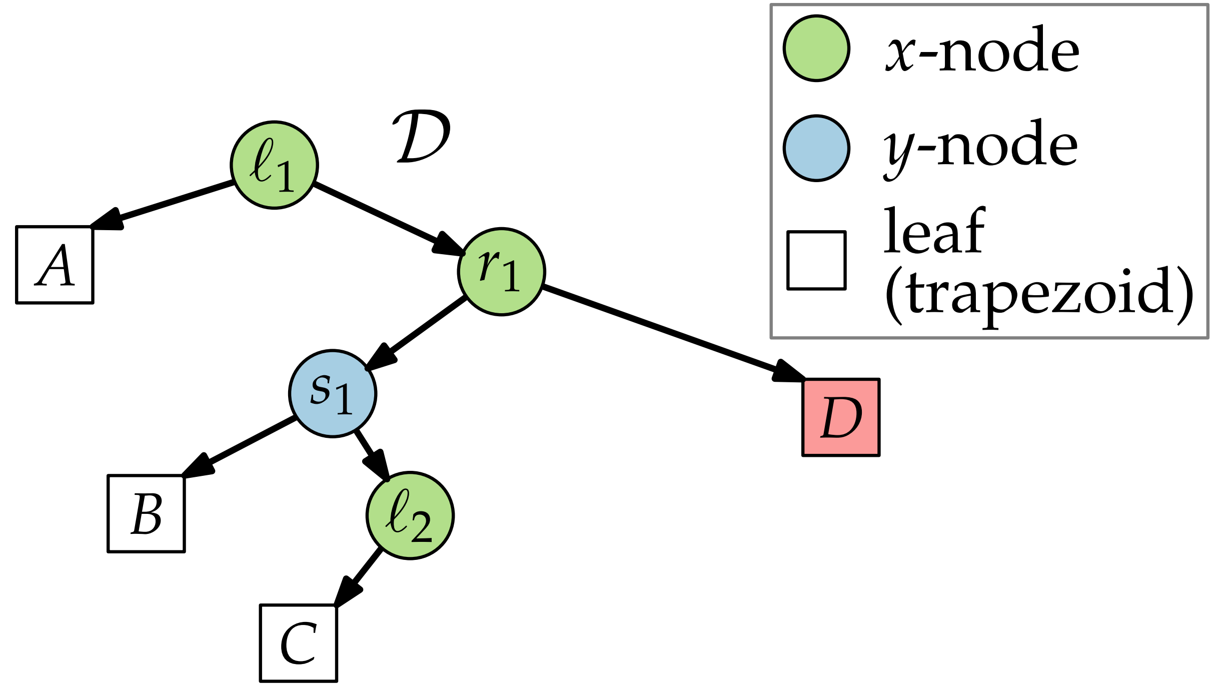
– "walk" along s through $\mathcal{T}$

– destroy all trapezoids of $\mathcal{T}$ intersecting s

– construct new trapezoids of $\mathcal{T}$ (adjacent to s)

– update $\mathcal{D}$



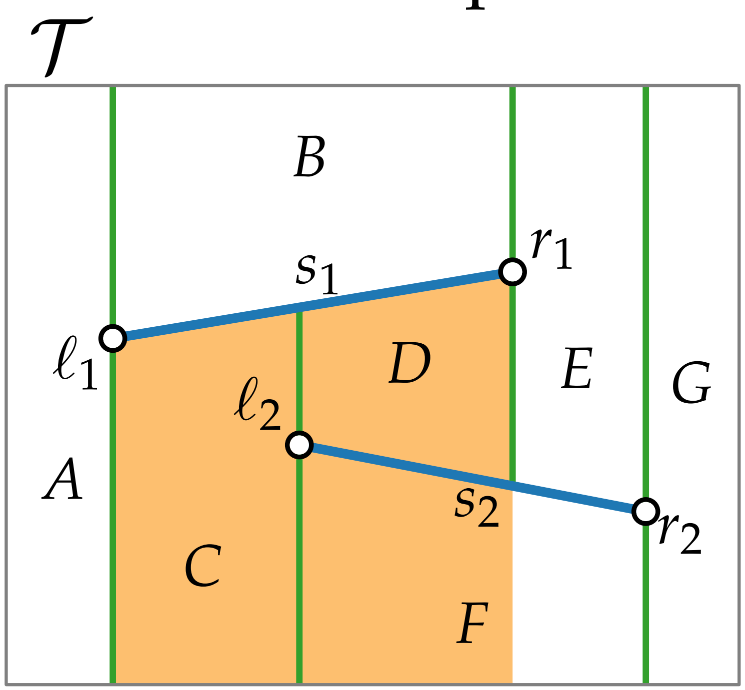


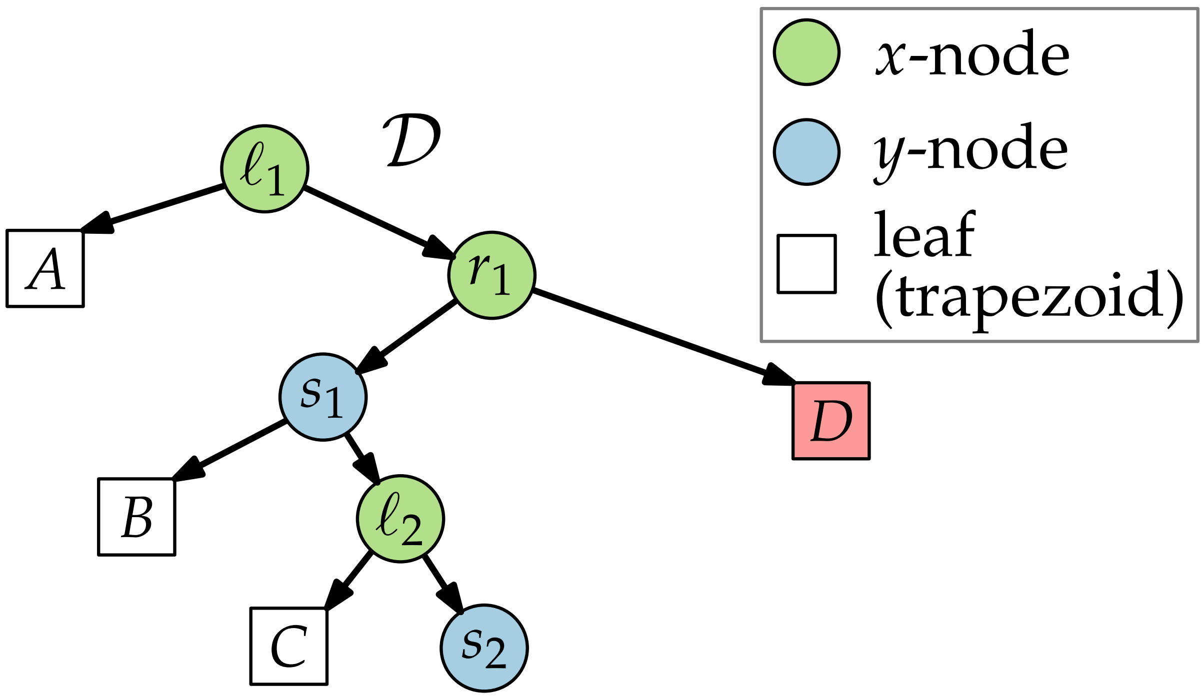
The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$





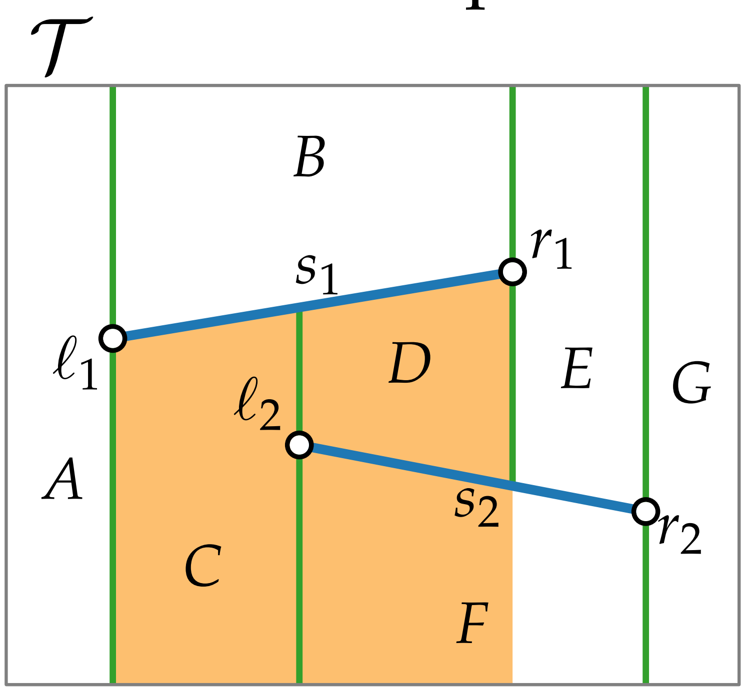
The 2D Problem

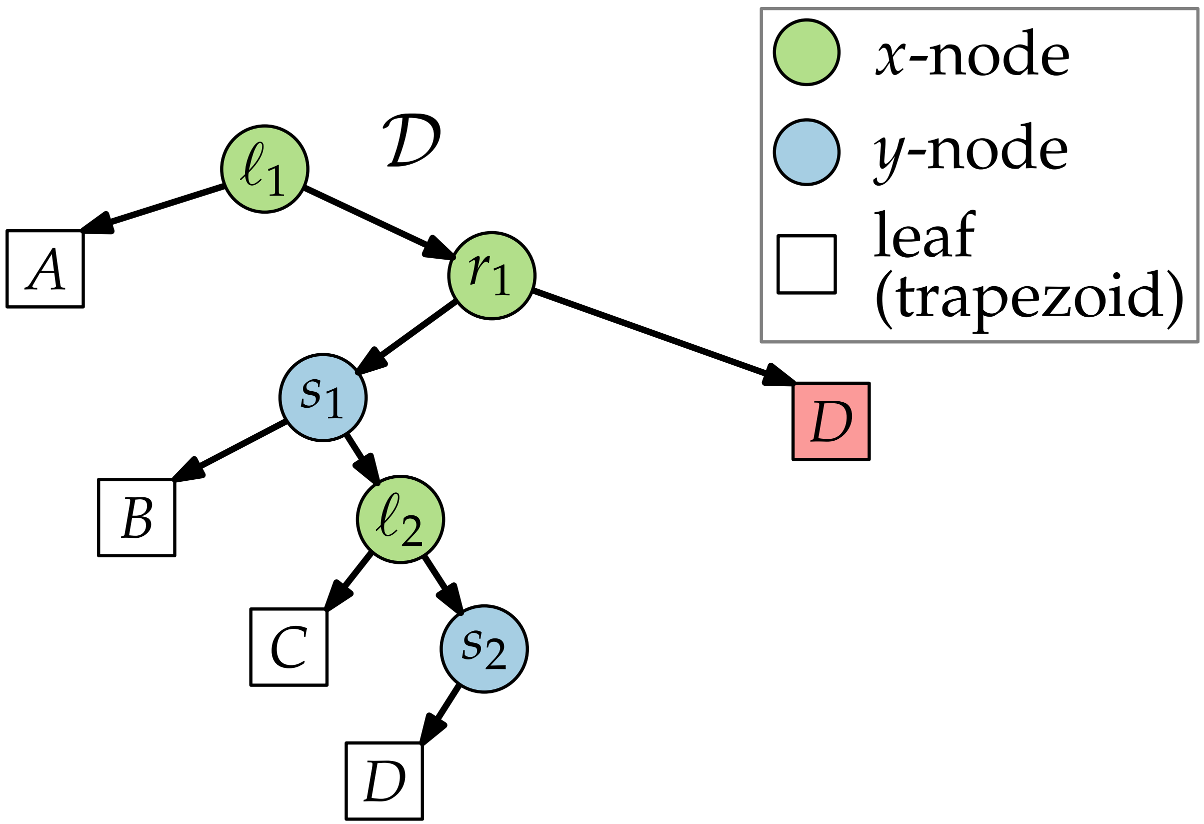
point-location data structure (DAG)

trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$



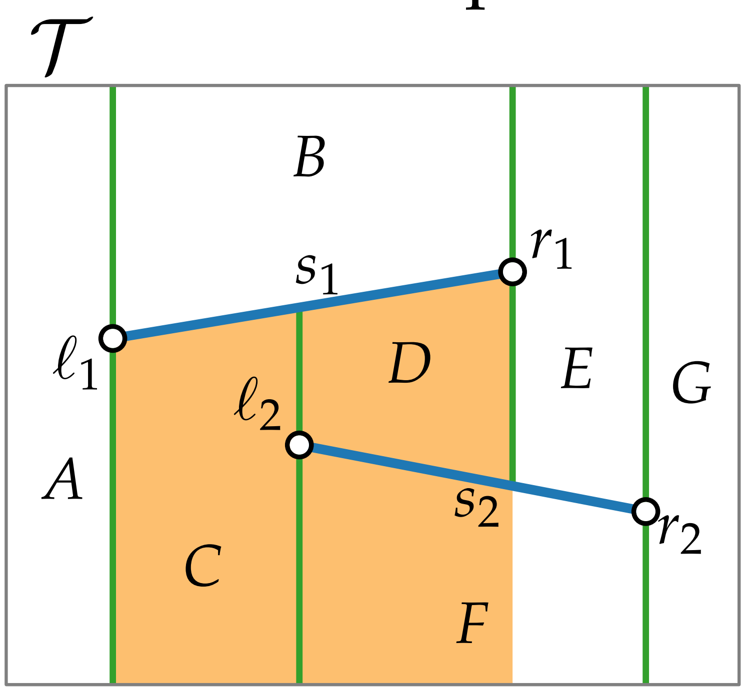


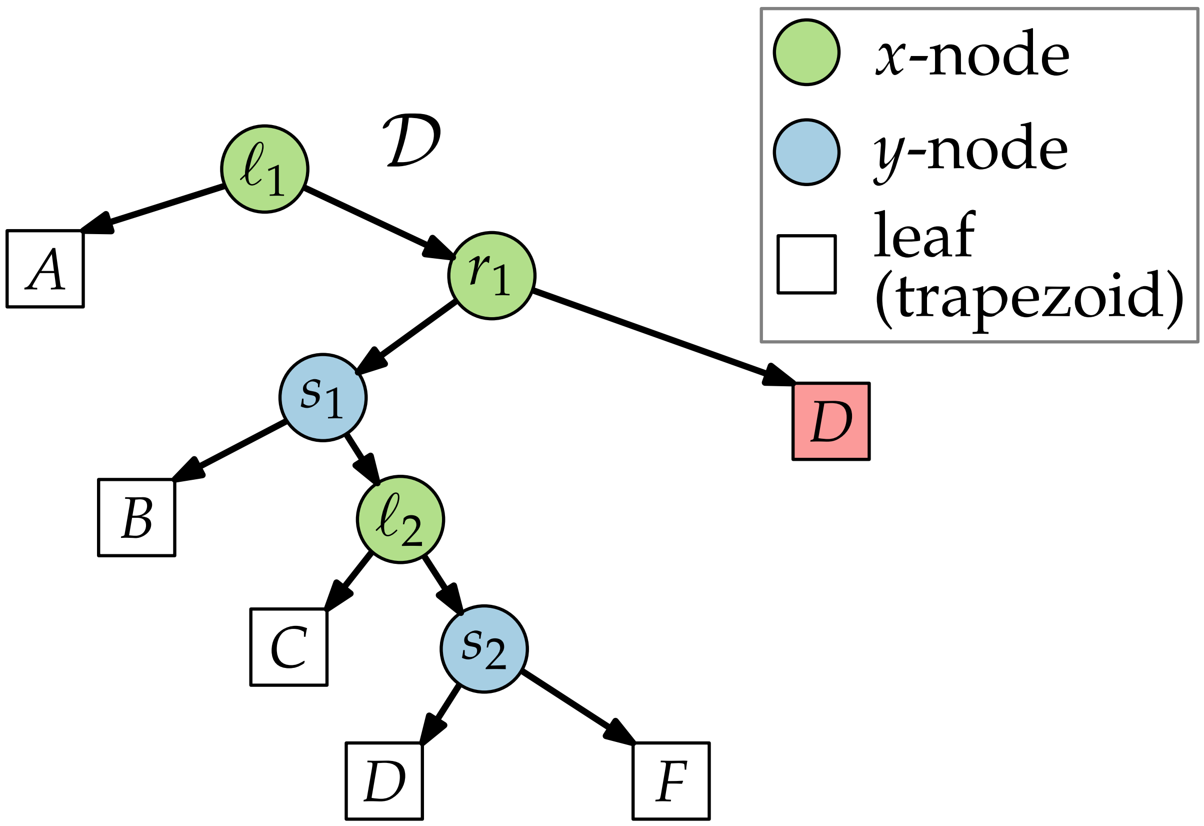
The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$



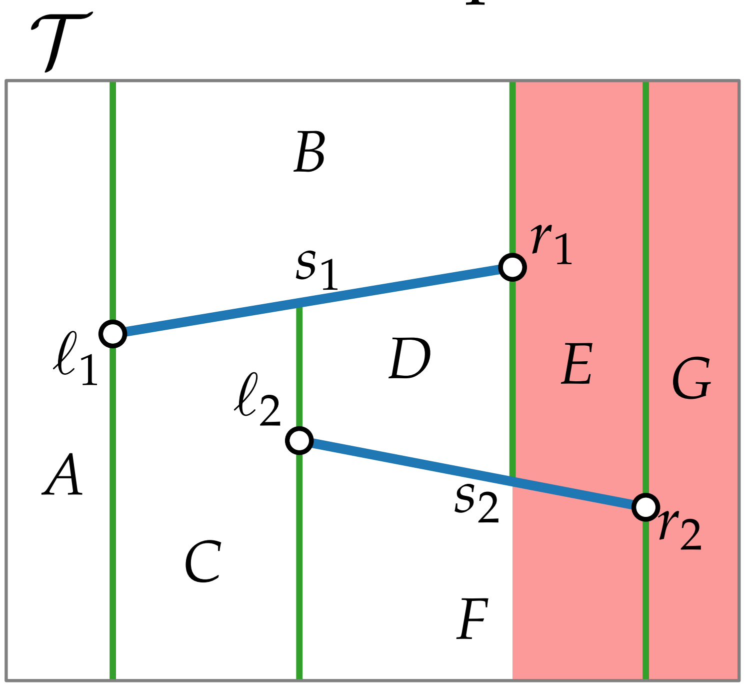


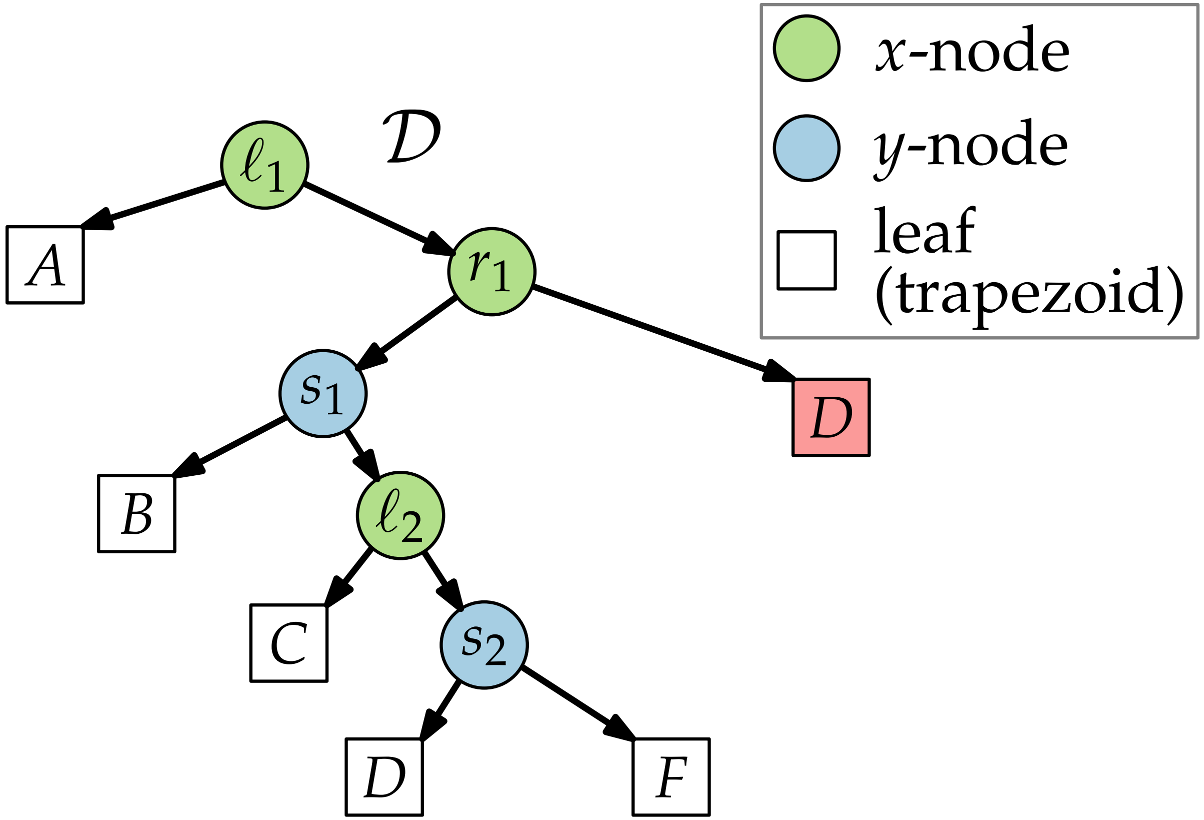
The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$

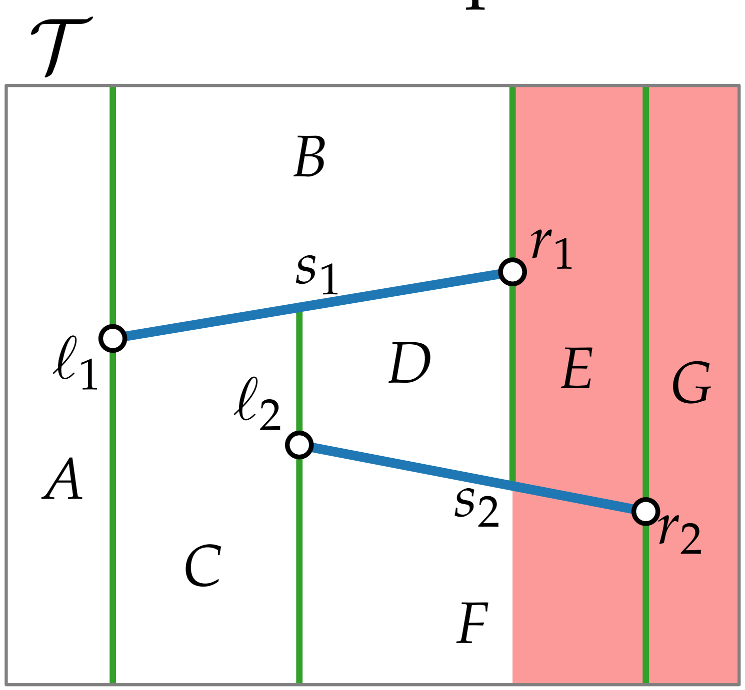


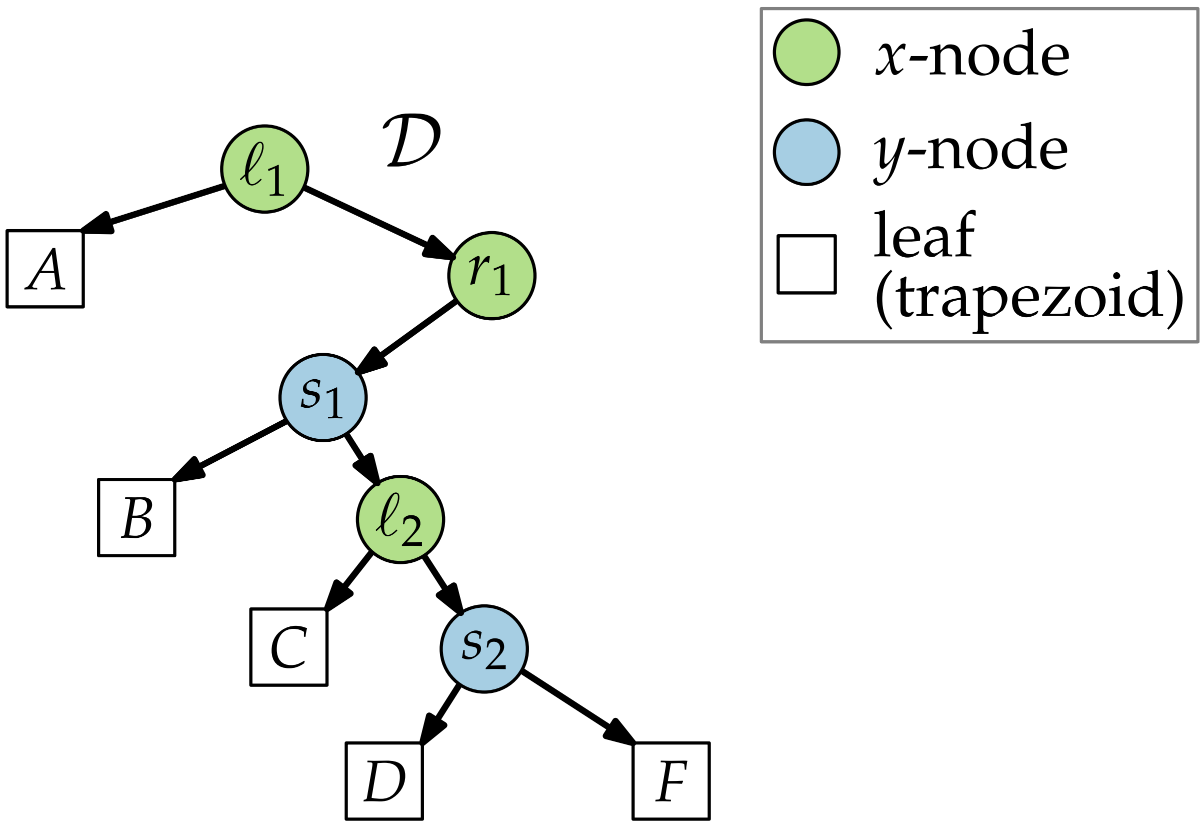


The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$
- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$





The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

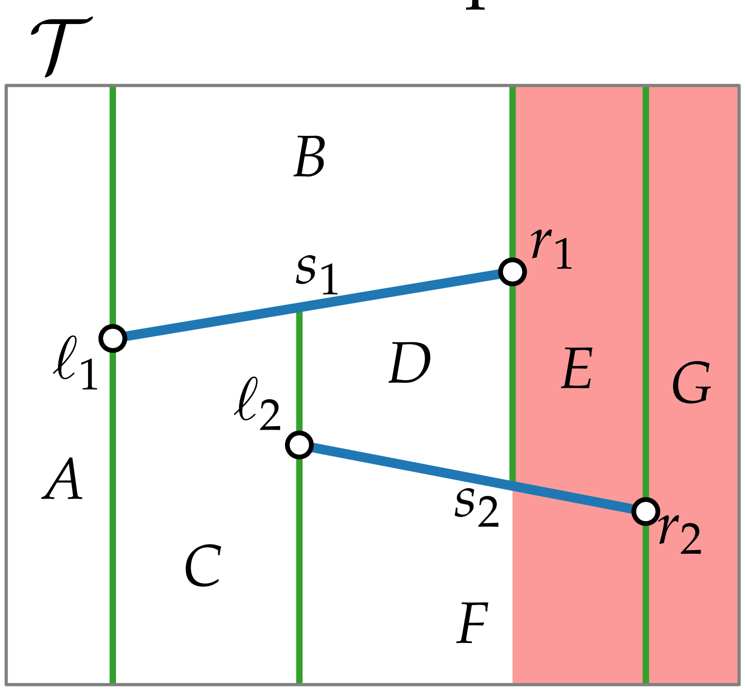
– use $\mathcal{D}$ to locate left endpoint of next segment s

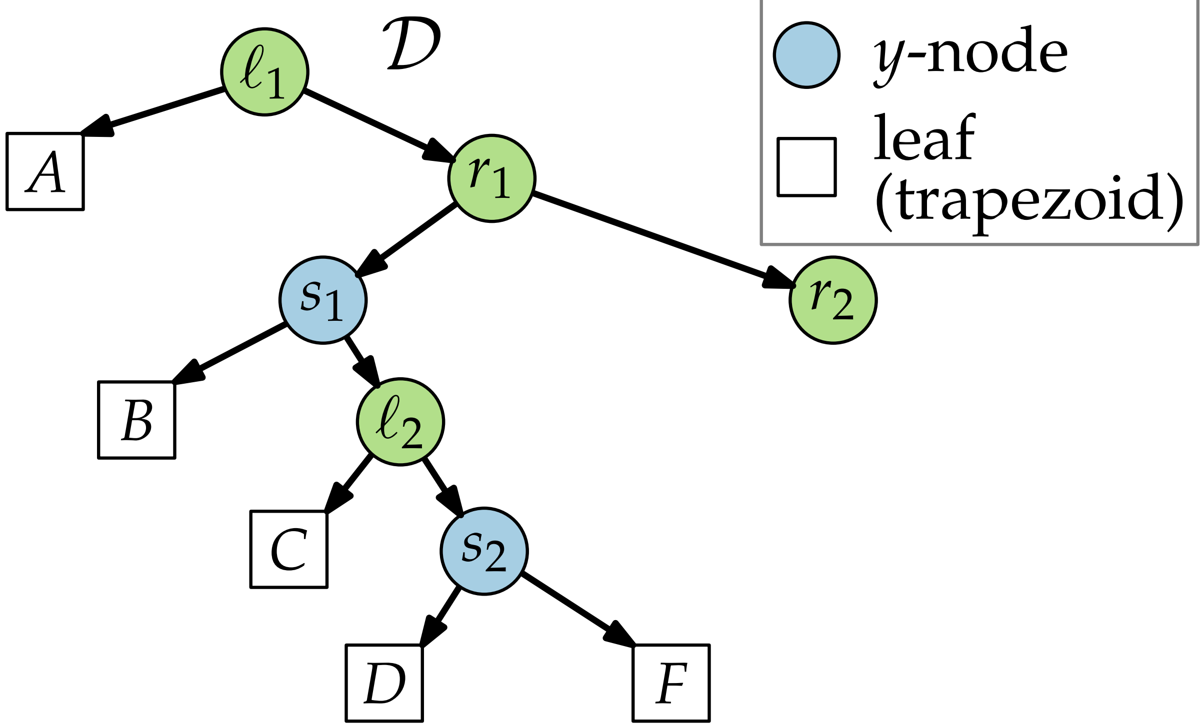
– "walk" along s through $\mathcal{T}$

– destroy all trapezoids of $\mathcal{T}$ intersecting s

– construct new trapezoids of $\mathcal{T}$ (adjacent to s)

– update $\mathcal{D}$



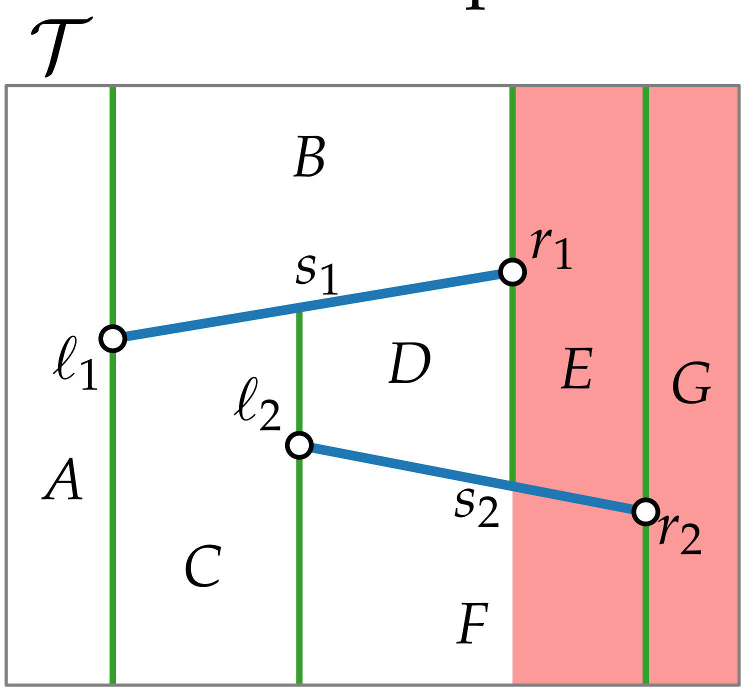


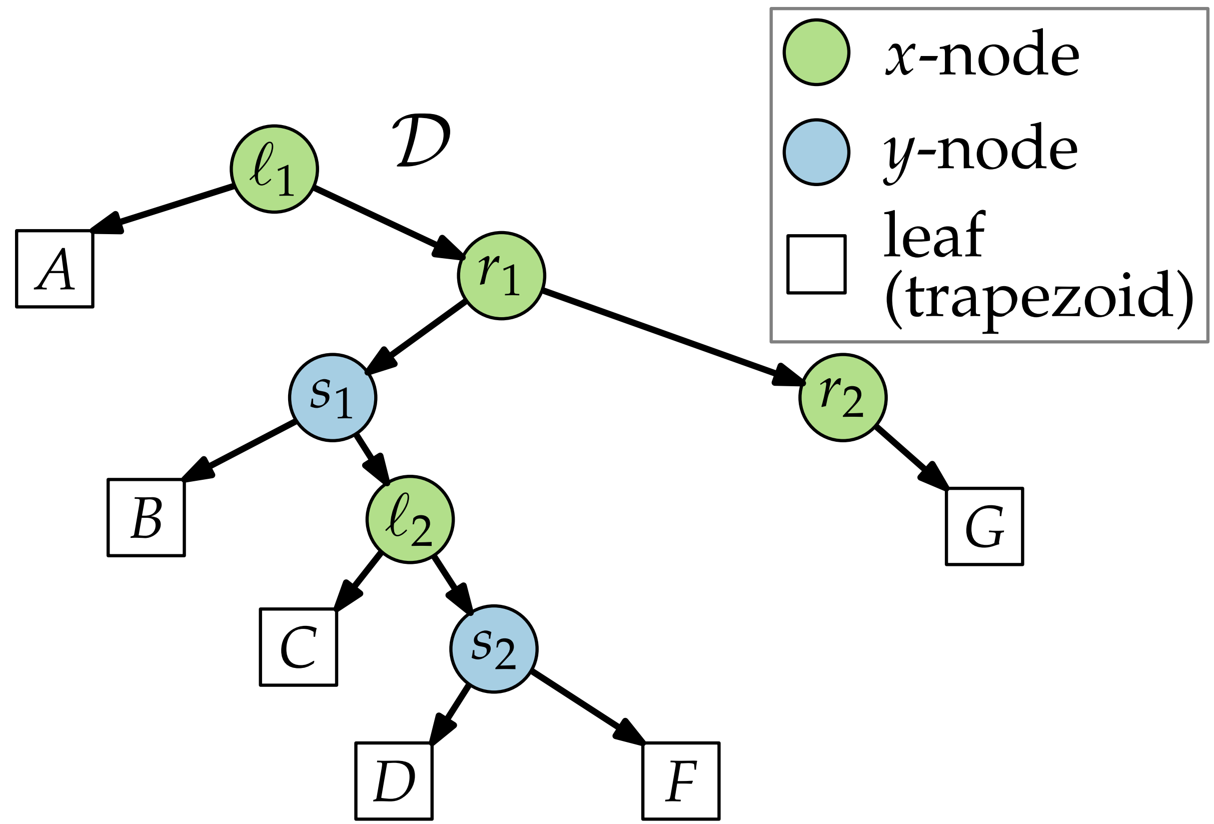
The 2D Problem

point-location data structure (DAG)
trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

– use $\mathcal{D}$ to locate left endpoint of next segment s
– "walk" along s through $\mathcal{T}$
– destroy all trapezoids of $\mathcal{T}$ intersecting s
– construct new trapezoids of $\mathcal{T}$ (adjacent to s)
– update $\mathcal{D}$





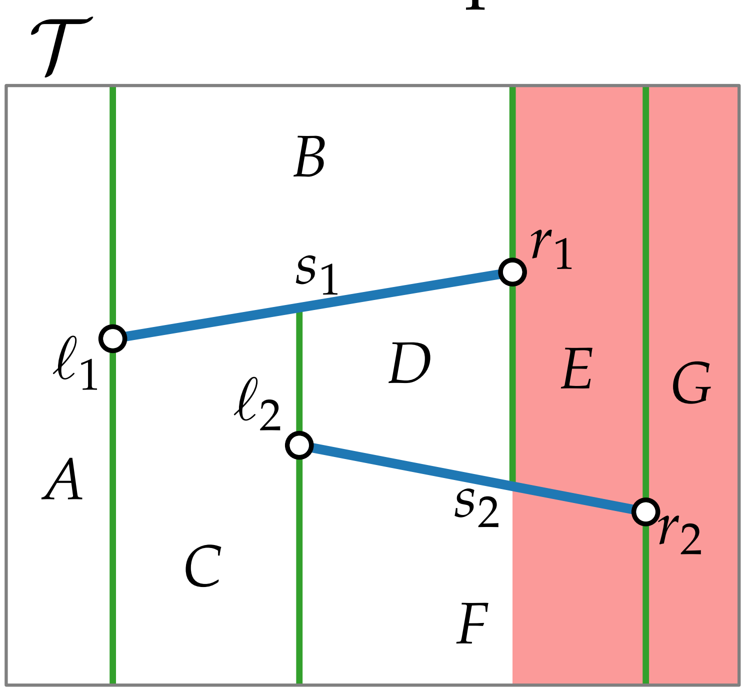
The 2D Problem

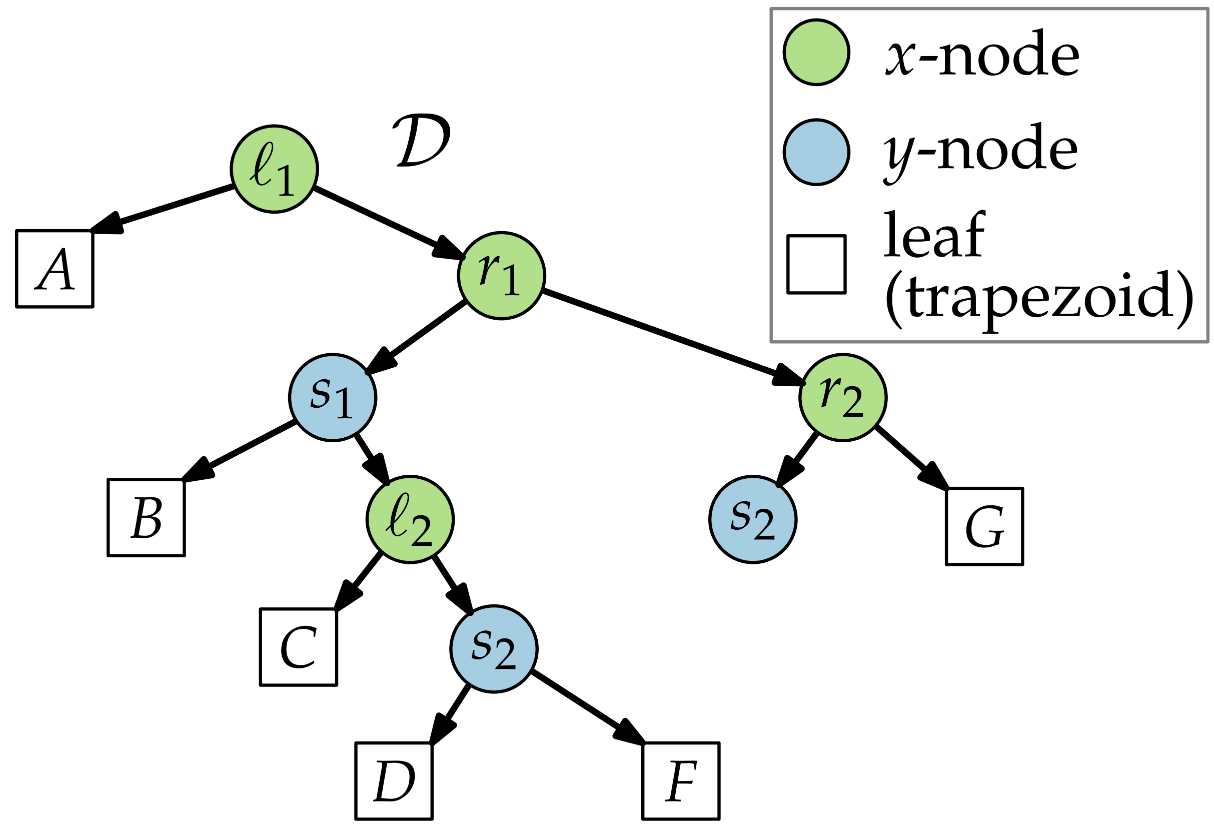
point-location data structure (DAG)

trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$





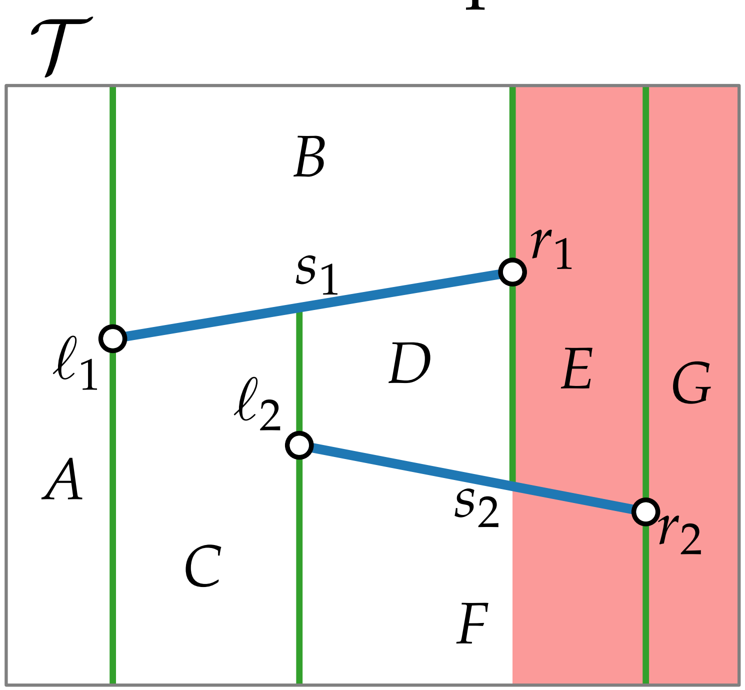
The 2D Problem

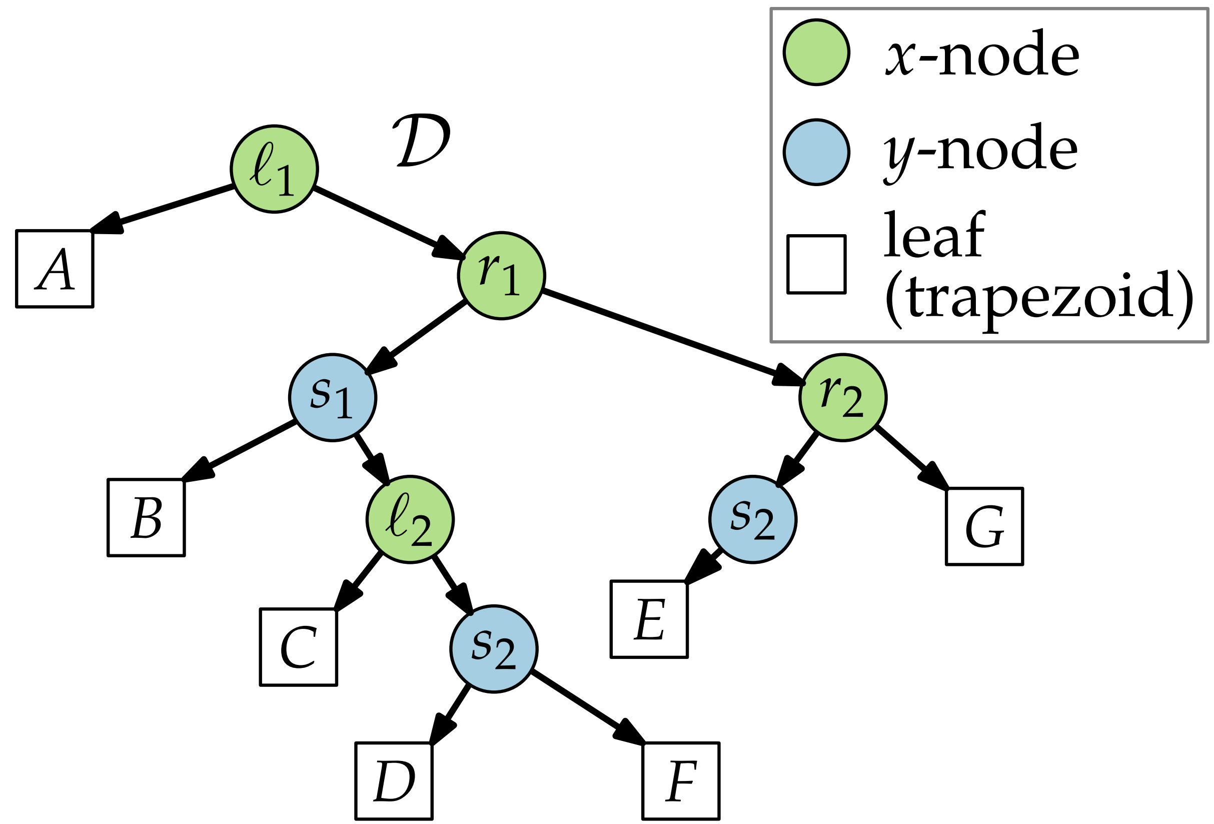
point-location data structure (DAG)

trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$





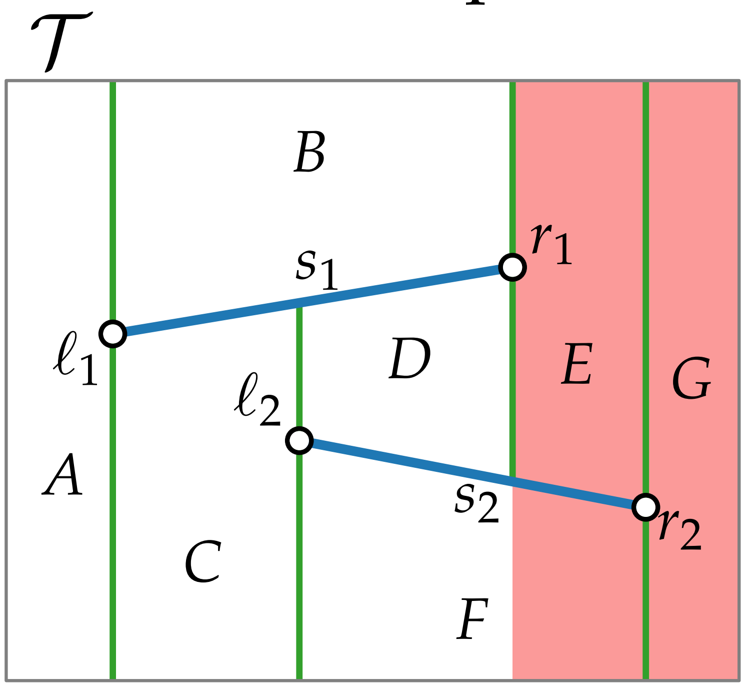
The 2D Problem

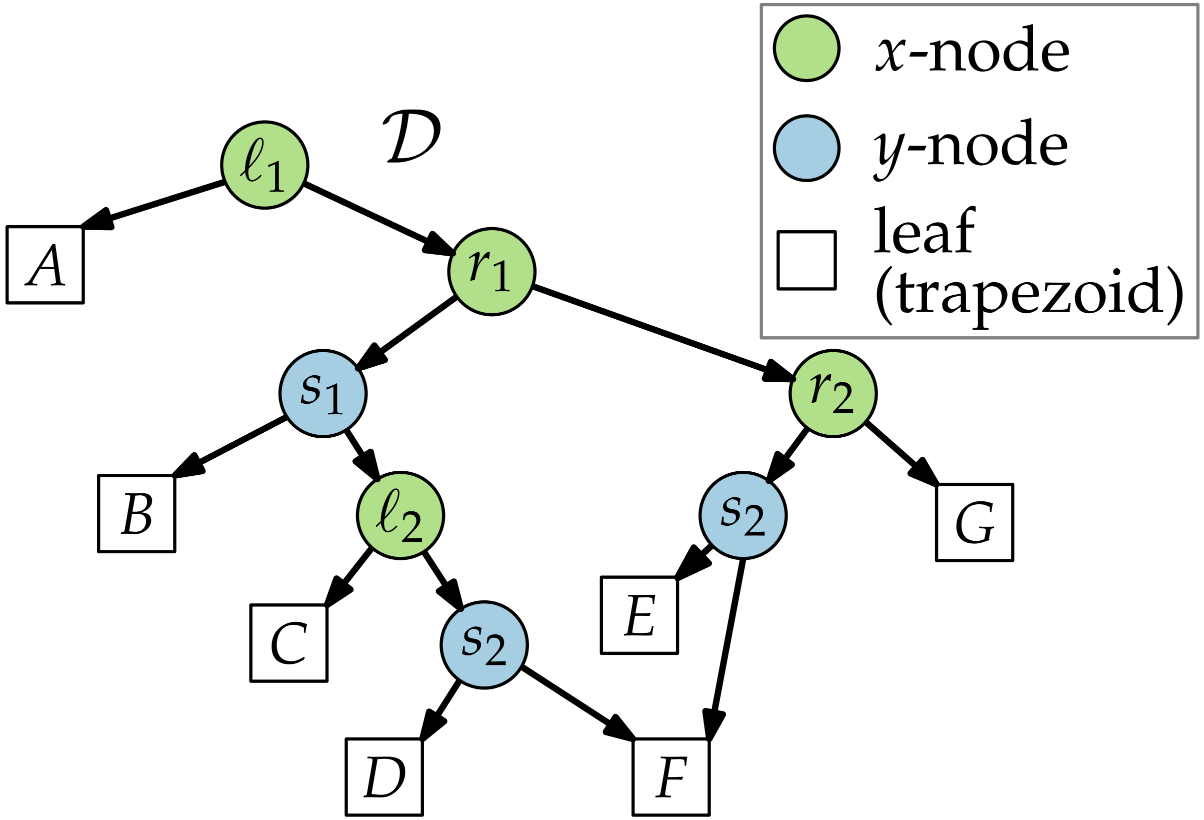
point-location data structure (DAG)

trapezoidal map

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$





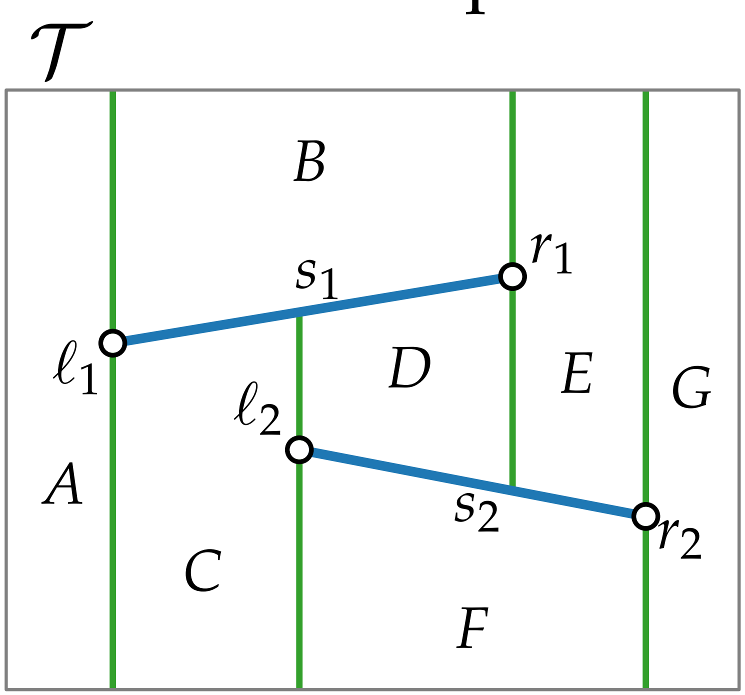
The 2D Problem

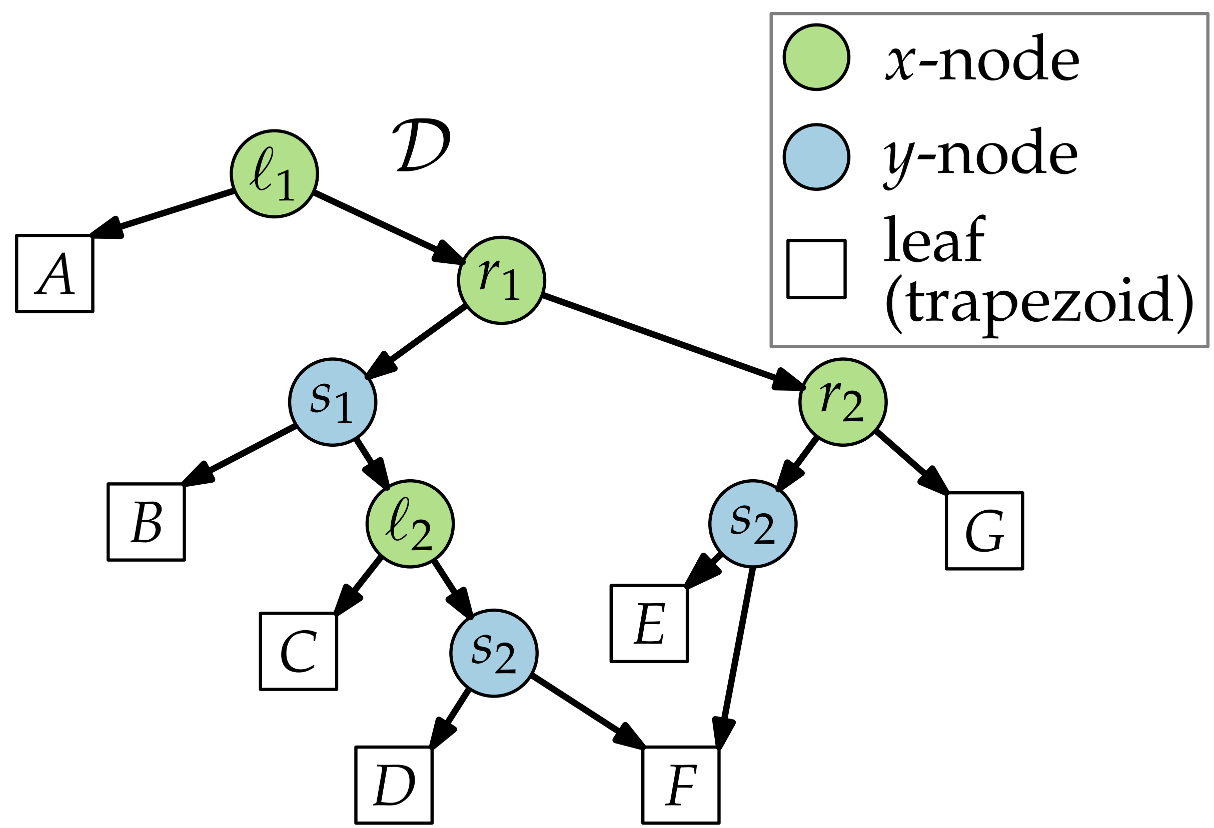
point-location data structure (DAG)

trapezoidal map

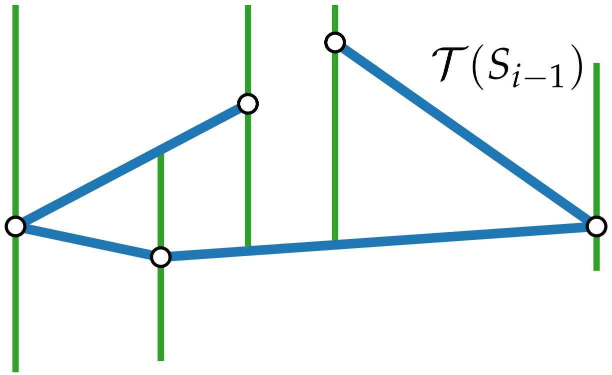
Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment s
- "walk" along s through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting s
- construct new trapezoids of $\mathcal{T}$ (adjacent to s)
- update $\mathcal{D}$



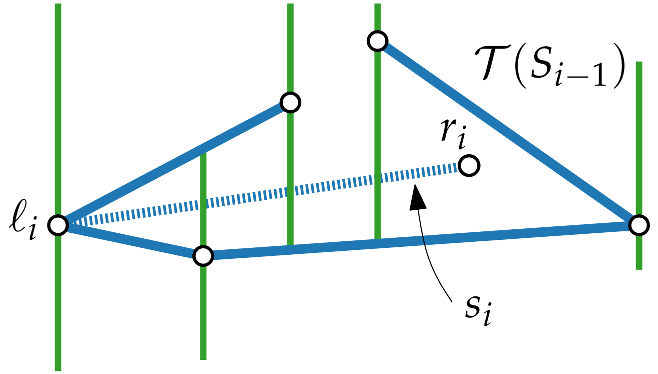


Walking through $\mathcal{T}$ and Updating $\mathcal{D}$



TrapezoidalMap(set S of n non-crossing segments)
$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()
$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$
for $i = 1$ **to** n **do**

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$



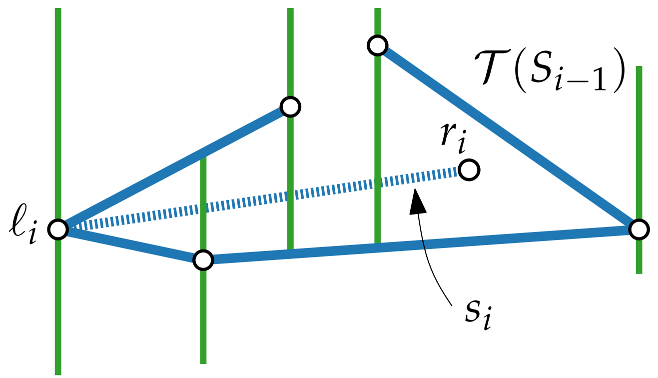
TrapezoidalMap(set S of n non-crossing segments)

$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1$ **to** n **do**

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$



TrapezoidalMap(set S of n non-crossing segments)

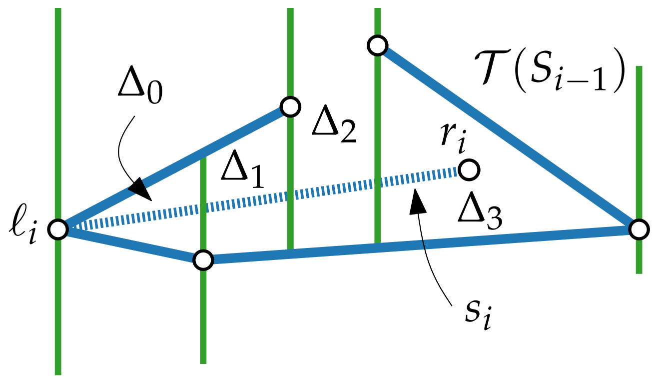
R = BBox(S); $\mathcal{T}$.init(); $\mathcal{D}$.init()

$(s_1, s_2, \ldots, s_n)$ = RandomPermutation(S)

for $i = 1$ **to** n **do**

$(\Delta_0, \ldots, \Delta_k)$ = FollowSegment($\mathcal{T}$, $\mathcal{D}$, s_i)

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$



TrapezoidalMap(set S of n non-crossing segments)

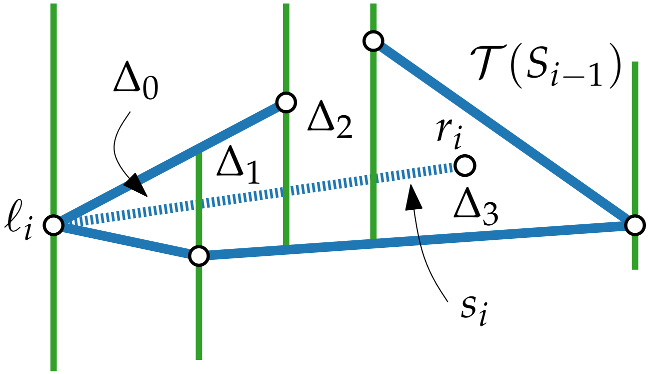
$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1$ **to** n **do**

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$



TrapezoidalMap(set S of n non-crossing segments)

$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()

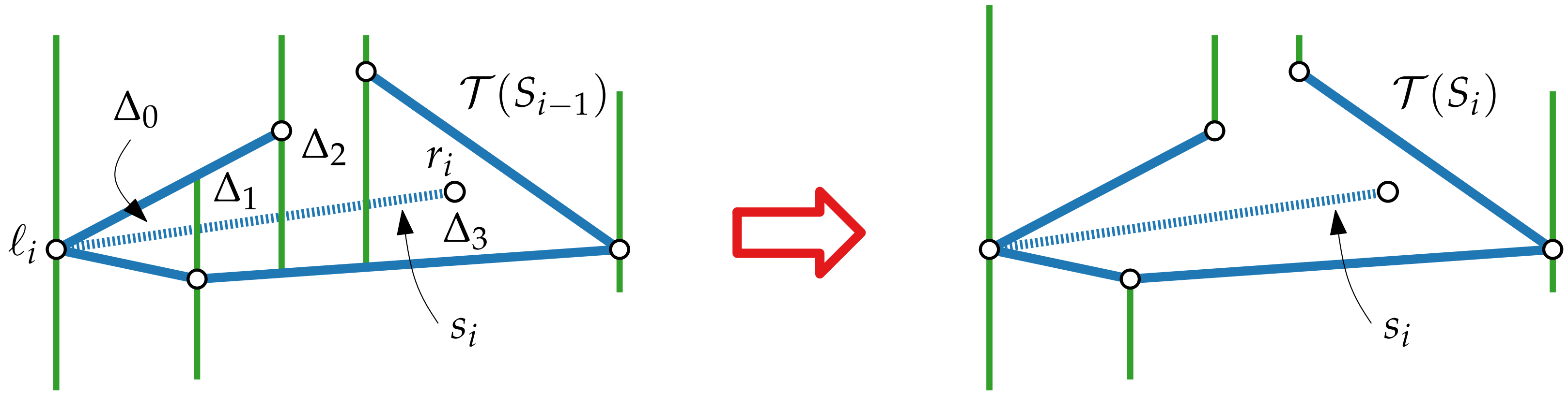
$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1$ **to** n **do**

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

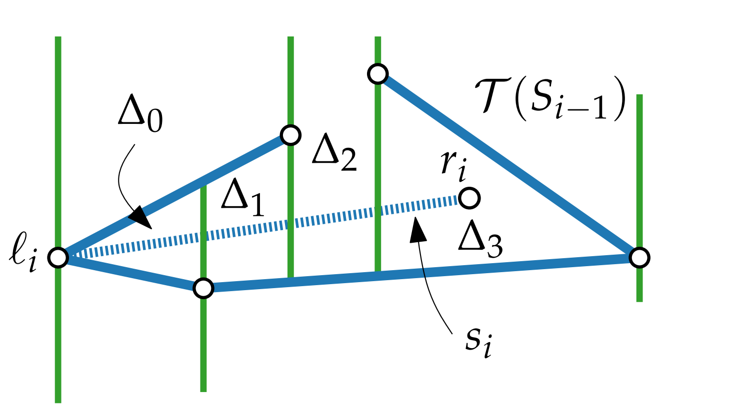
$\mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k)$

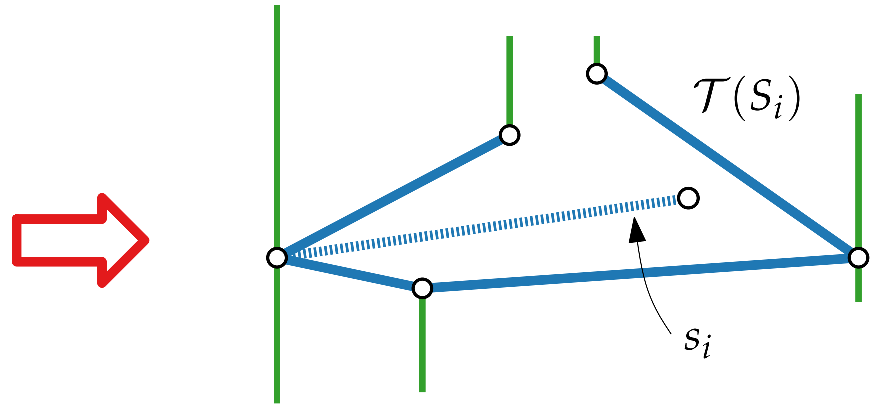
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$



TrapezoidalMap(set S of n non-crossing segments)
$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()
$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$
for $i = 1$ **to** n **do**
$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$
$\mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k)$

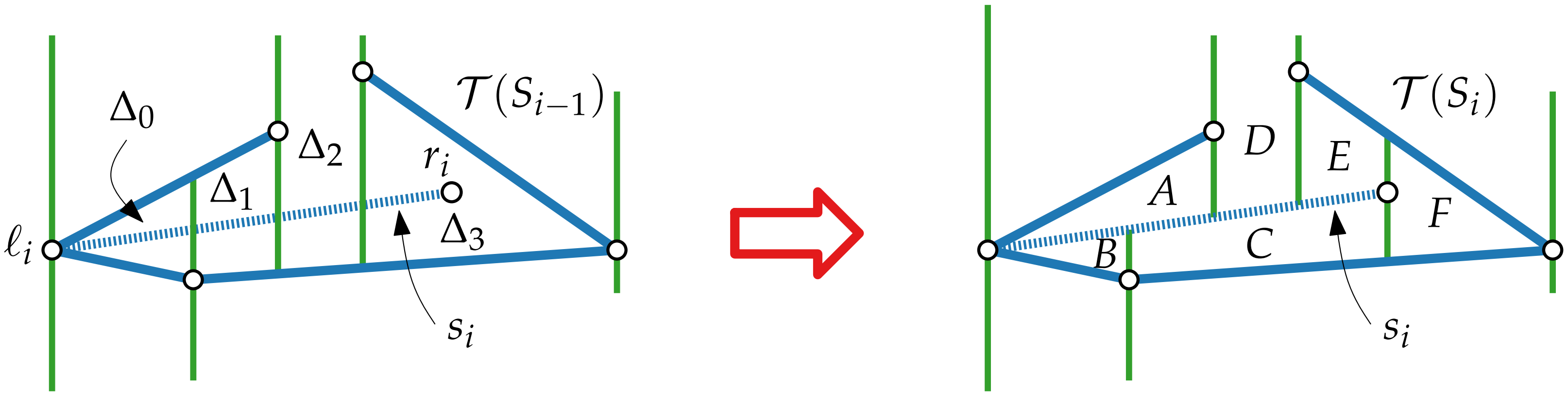
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$





TrapezoidalMap(set S of n non-crossing segments)
$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()
$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$
for $i = 1$ **to** n **do**
$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$
$\mathcal{T}$.remove$(\Delta_0, \ldots, \Delta_k)$
$\mathcal{T}$.add(new trapezoids incident to s_i)

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$



TrapezoidalMap(set S of n non-crossing segments)

$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

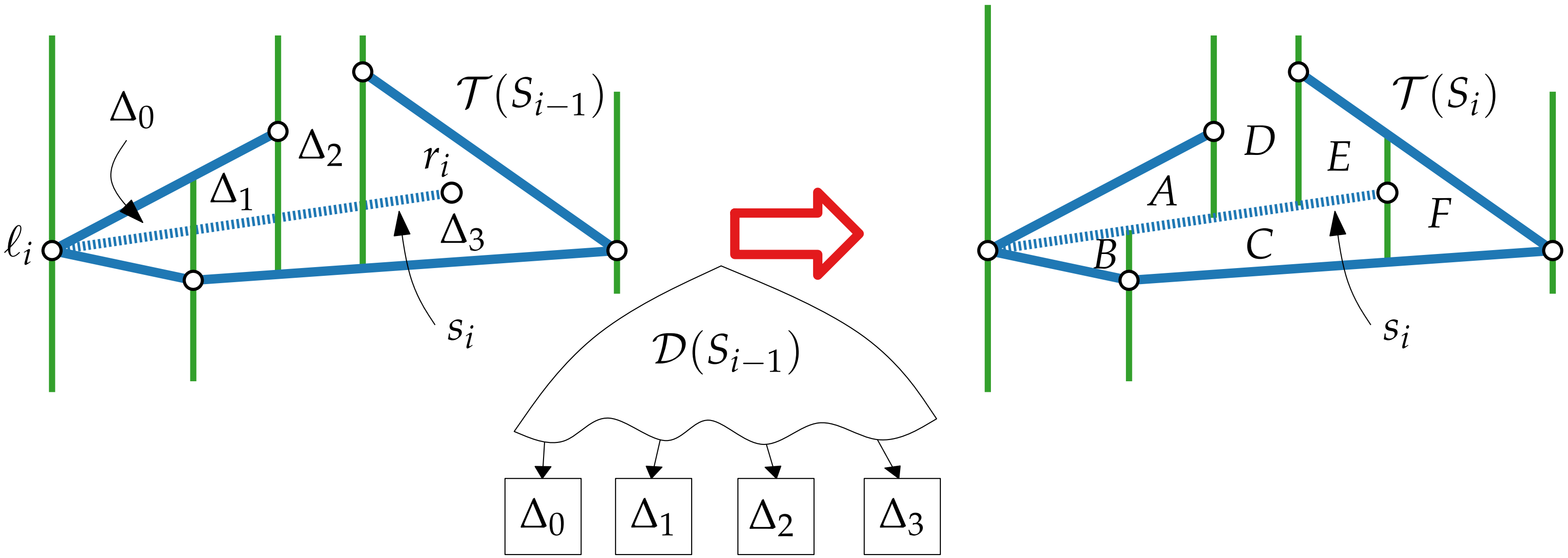
for $i = 1$ **to** n **do**

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

$\mathcal{T}$.remove$(\Delta_0, \ldots, \Delta_k)$

$\mathcal{T}$.add(new trapezoids incident to s_i)

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$



TrapezoidalMap(set S of n non-crossing segments)

$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

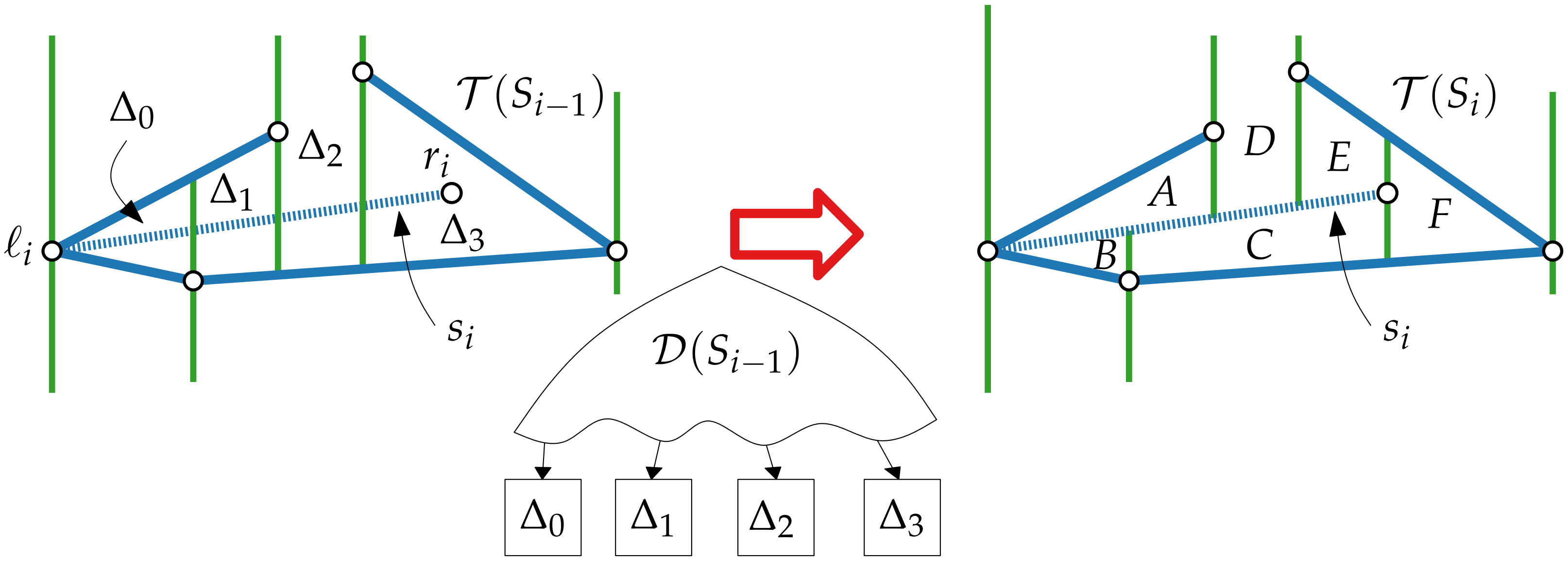
for $i = 1$ **to** n **do**

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

$\mathcal{T}$.remove$(\Delta_0, \ldots, \Delta_k)$

$\mathcal{T}$.add(new trapezoids incident to s_i)

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$



TrapezoidalMap(set S of n non-crossing segments)

$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1$ **to** n **do**

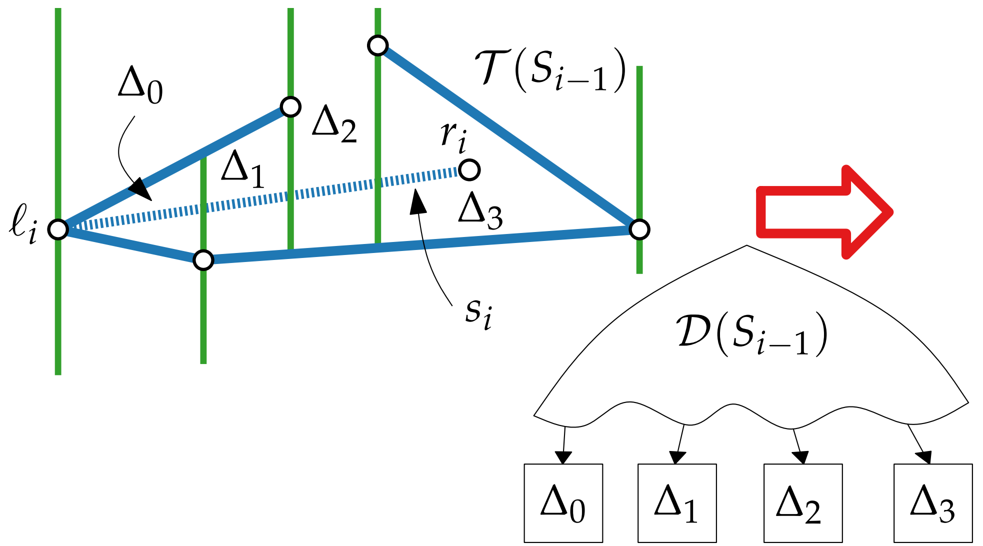
  $(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

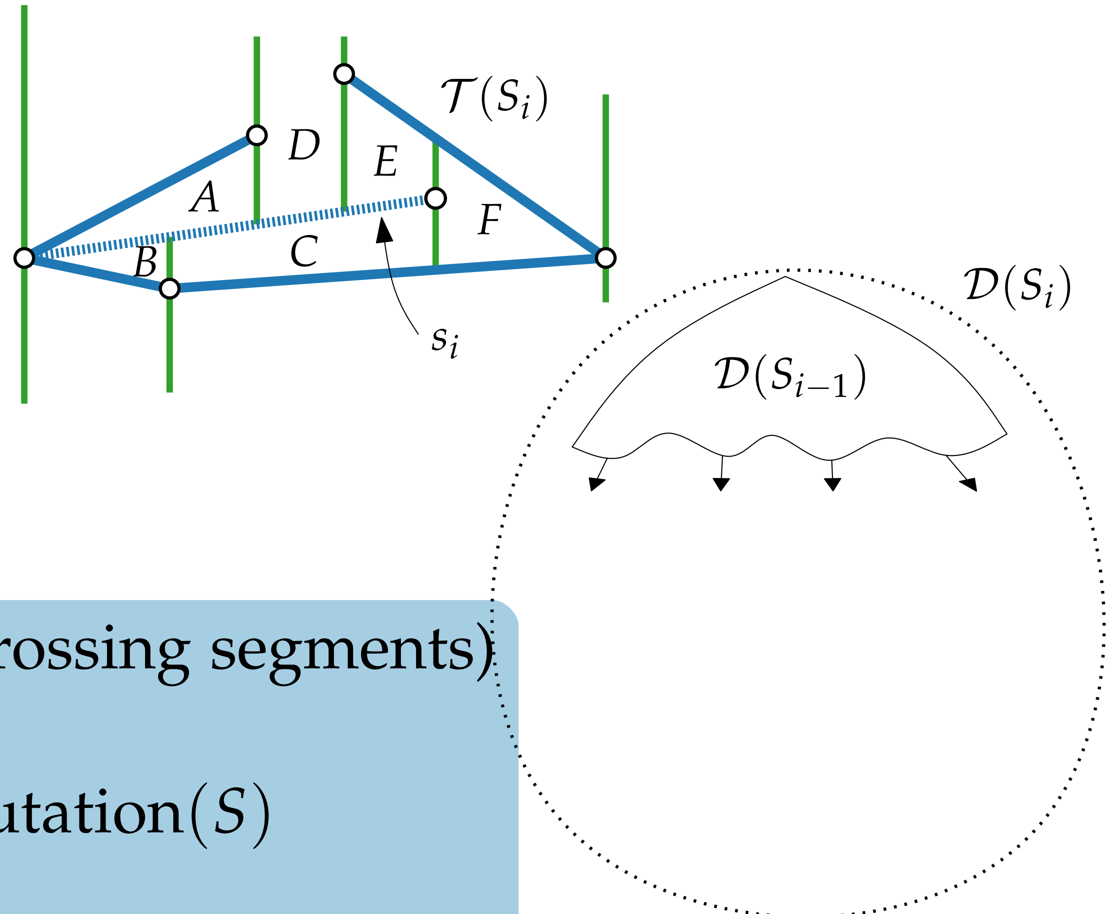
  $\mathcal{T}$.remove$(\Delta_0, \ldots, \Delta_k)$

  $\mathcal{T}$.add(new trapezoids incident to s_i)

  $\mathcal{D}$.remove_leaves$(\Delta_0, \ldots, \Delta_k)$

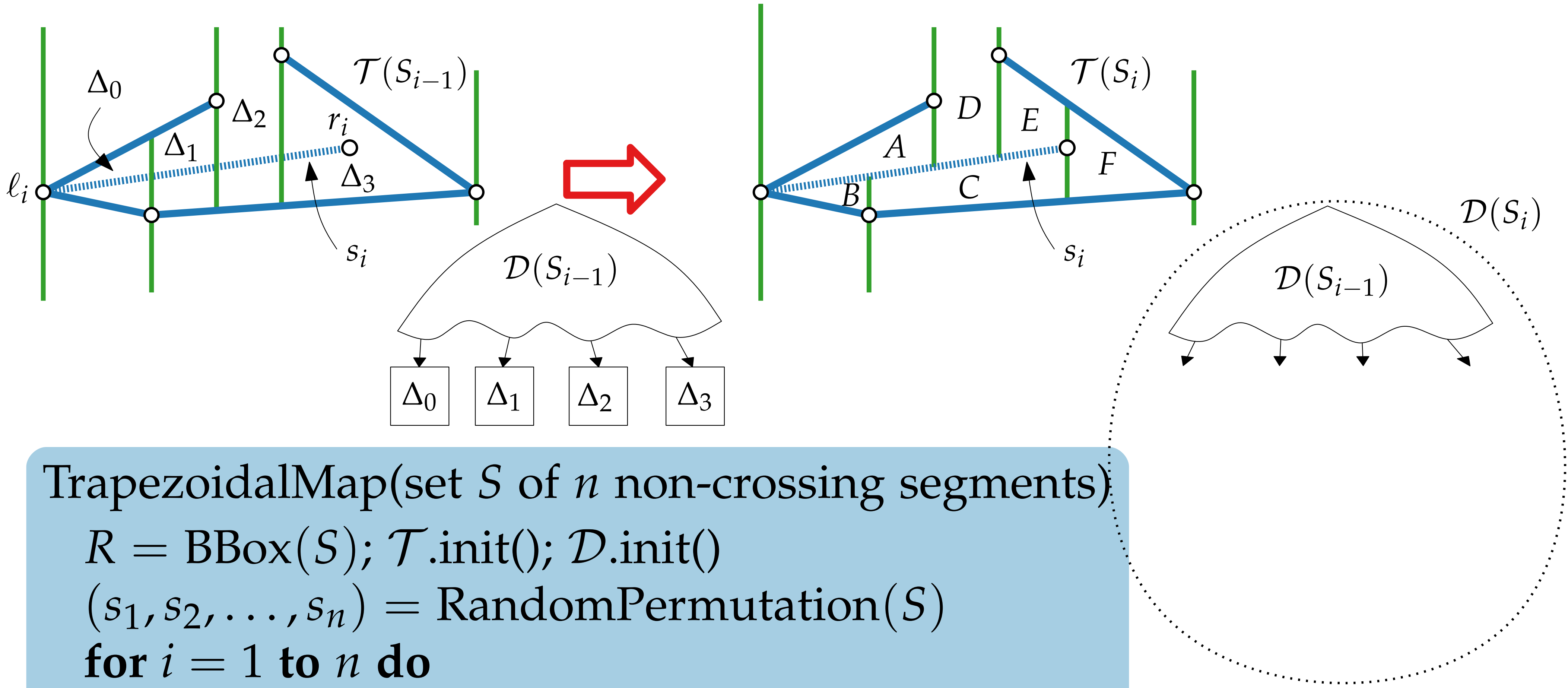
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$





TrapezoidalMap(set S of n non-crossing segments)
  $R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()
  $(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$
  for $i = 1$ **to** n **do**
    $(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$
    $\mathcal{T}$.remove$(\Delta_0, \ldots, \Delta_k)$
    $\mathcal{T}$.add(new trapezoids incident to s_i)
    $\mathcal{D}$.remove_leaves$(\Delta_0, \ldots, \Delta_k)$

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$



TrapezoidalMap(set S of n non-crossing segments)

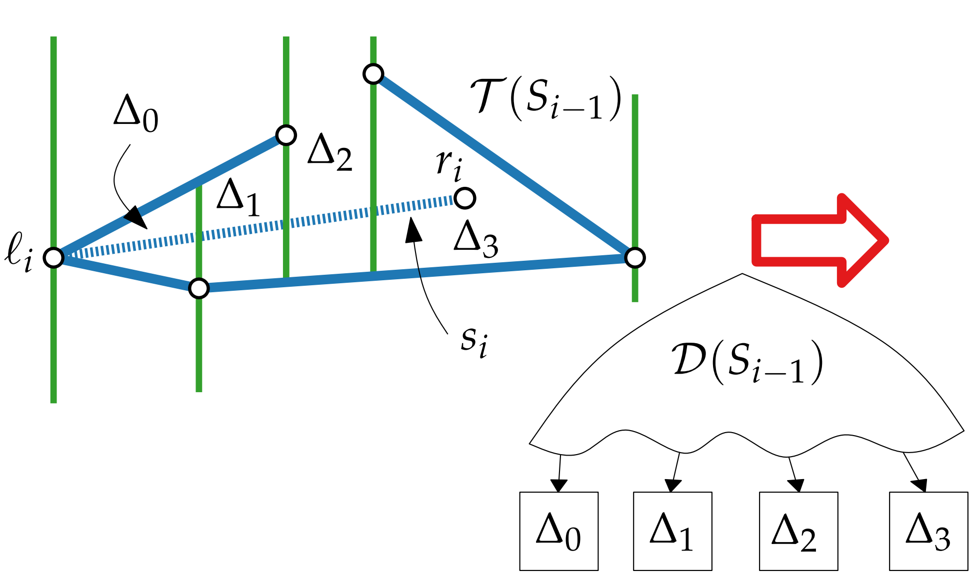
$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()

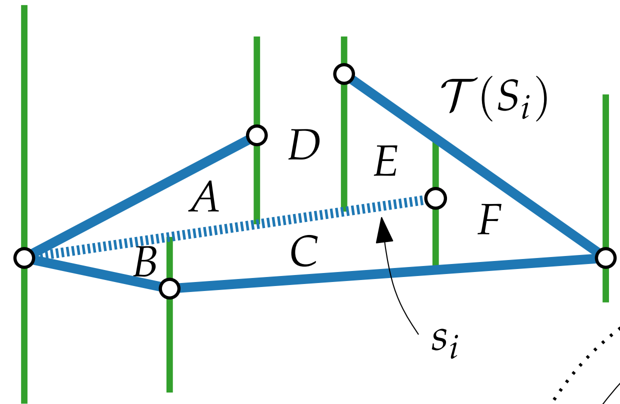
$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

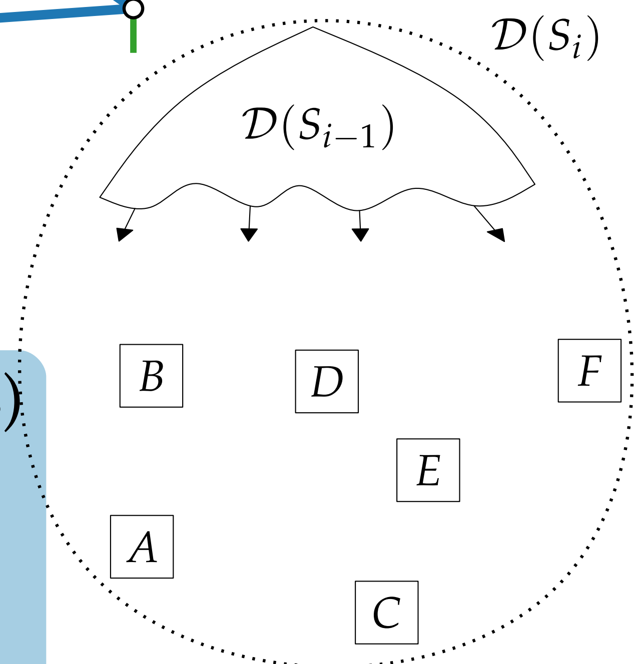
for $i = 1$ **to** n **do**

- $(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$
- $\mathcal{T}$.remove$(\Delta_0, \ldots, \Delta_k)$
- $\mathcal{T}$.add(new trapezoids incident to s_i)
- $\mathcal{D}$.remove_leaves$(\Delta_0, \ldots, \Delta_k)$
- $\mathcal{D}$.add_leaves(new trapezoids incident to s_i)

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

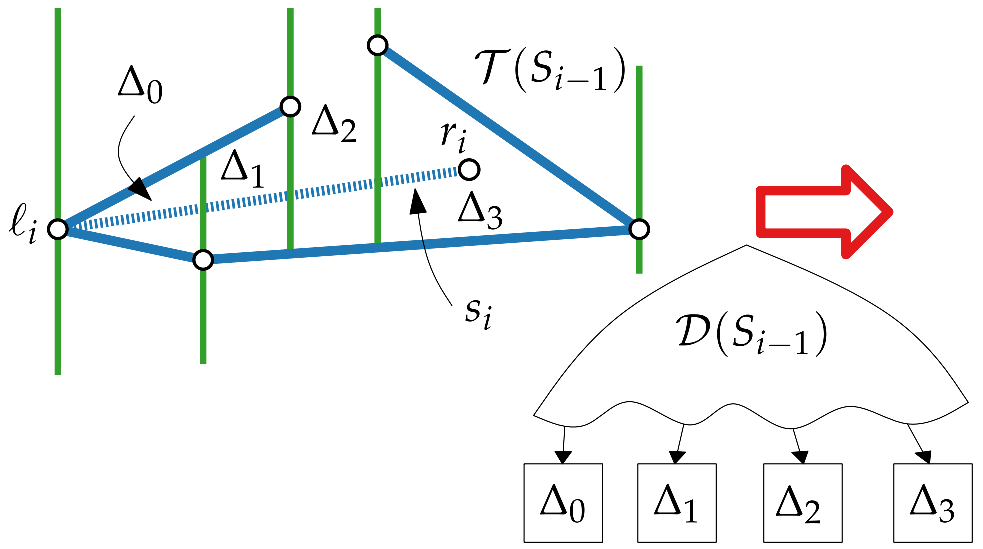


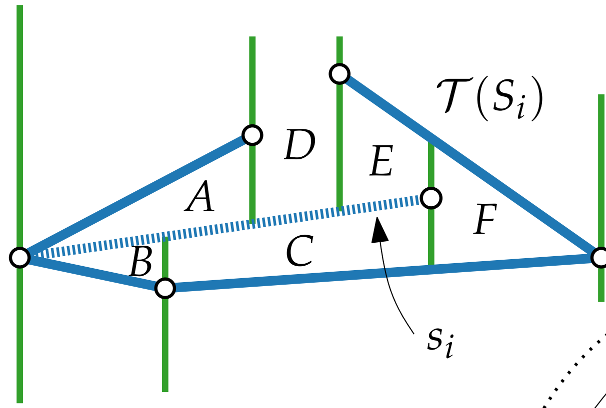


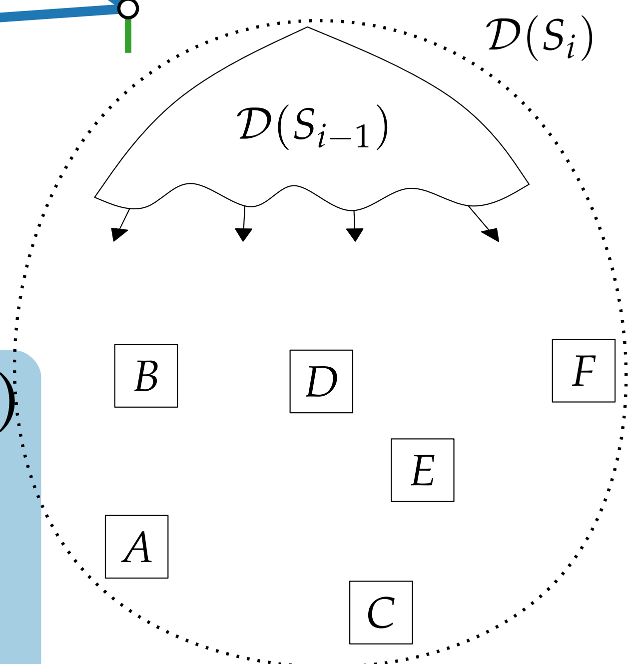


TrapezoidalMap(set S of n non-crossing segments)
  $R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()
  $(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$
  for $i = 1$ **to** n **do**
    $(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$
    $\mathcal{T}$.remove$(\Delta_0, \ldots, \Delta_k)$
    $\mathcal{T}$.add(new trapezoids incident to s_i)
    $\mathcal{D}$.remove_leaves$(\Delta_0, \ldots, \Delta_k)$
    $\mathcal{D}$.add_leaves(new trapezoids incident to s_i)

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

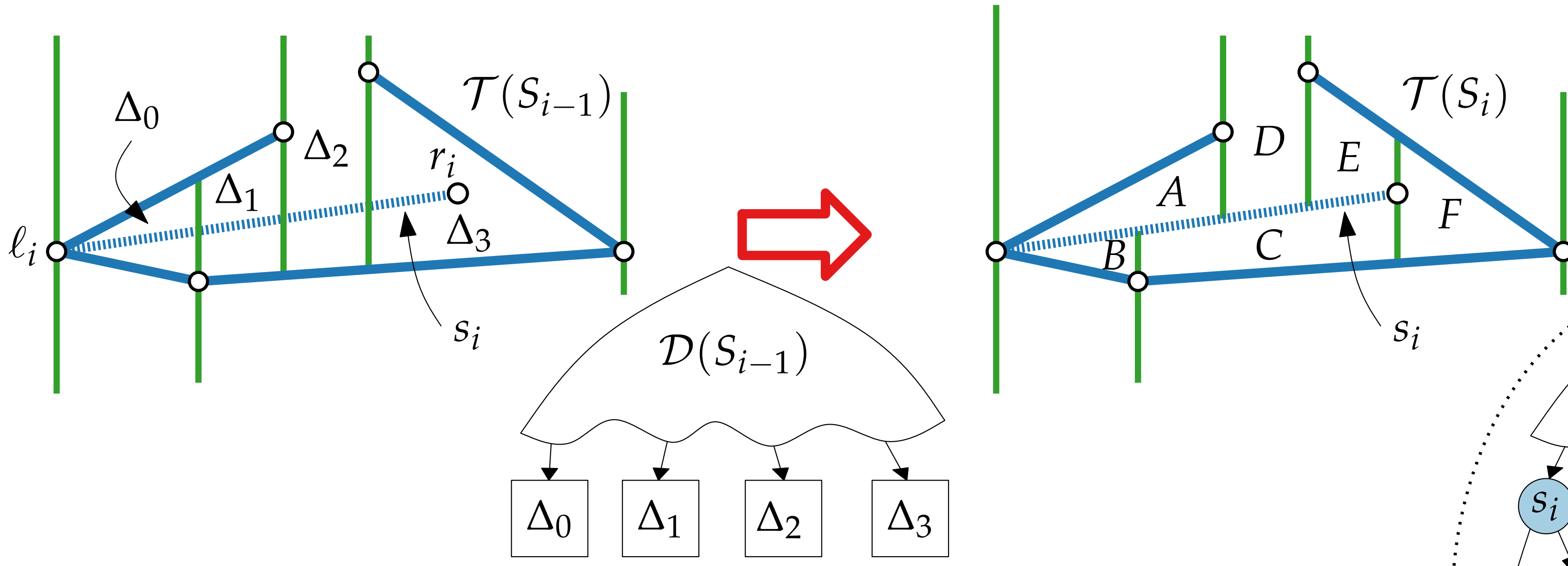


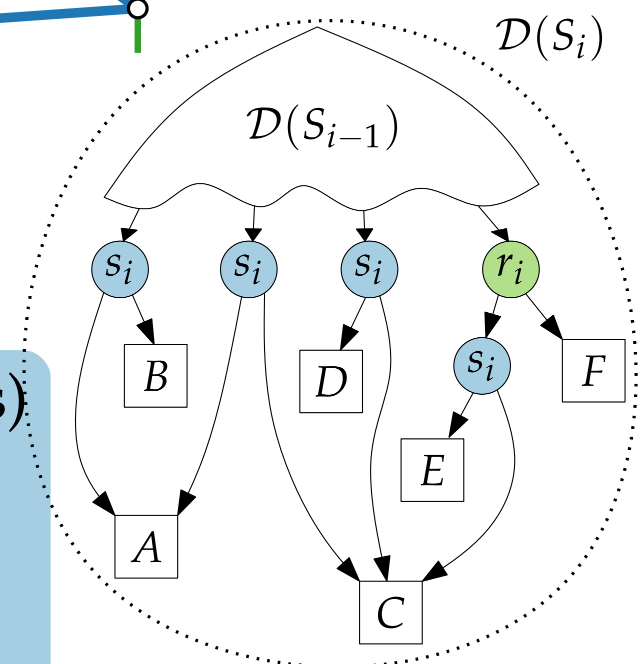




TrapezoidalMap(set S of n non-crossing segments)
  $R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()
  $(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$
  for $i = 1$ **to** n **do**
    $(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$
    $\mathcal{T}$.remove$(\Delta_0, \ldots, \Delta_k)$
    $\mathcal{T}$.add(new trapezoids incident to s_i)
    $\mathcal{D}$.remove_leaves$(\Delta_0, \ldots, \Delta_k)$
    $\mathcal{D}$.add_leaves(new trapezoids incident to s_i)
    $\mathcal{D}$.add_new_inner_nodes()

Walking through $\mathcal{T}$ and Updating $\mathcal{D}$





TrapezoidalMap(set S of n non-crossing segments)

$R = \text{BBox}(S)$; $\mathcal{T}$.init(); $\mathcal{D}$.init()

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1$ **to** n **do**

- $(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$
- $\mathcal{T}$.remove$(\Delta_0, \ldots, \Delta_k)$
- $\mathcal{T}$.add(new trapezoids incident to s_i)
- $\mathcal{D}$.remove_leaves$(\Delta_0, \ldots, \Delta_k)$
- $\mathcal{D}$.add_leaves(new trapezoids incident to s_i)
- $\mathcal{D}$.add_new_inner_nodes()

The 2d-Result

Theorem. TrapezoidalMap(S) computes $\mathcal{T}(S)$ for a set of n line segments in general position and a search structure $\mathcal{D}$ for $\mathcal{T}(S)$ in $O(n \log n)$ expected time.

The 2d-Result

Theorem. TrapezoidalMap(S) computes $\mathcal{T}(S)$ for a set of n line segments in general position and a search structure $\mathcal{D}$ for $\mathcal{T}(S)$ in $O(n \log n)$ expected time. The expected size of $\mathcal{D}$ is $O(n)$ and the expected query time is $O(\log n)$.

The 2d-Result

Theorem. TrapezoidalMap(S) computes $\mathcal{T}(S)$ for a set of n line segments in general position and a search structure $\mathcal{D}$ for $\mathcal{T}(S)$ in $O(n \log n)$ expected time. The expected size of $\mathcal{D}$ is $O(n)$ and the expected query time is $O(\log n)$.

Invariant: Before step i, $\mathcal{T}$ is a trapezoidal map for S_{i-1} and $\mathcal{D}$ is a valid search structure for $\mathcal{T}$.

The 2d-Result

Theorem. TrapezoidalMap(S) computes $\mathcal{T}(S)$ for a set of n line segments in general position and a search structure $\mathcal{D}$ for $\mathcal{T}(S)$ in $O(n \log n)$ expected time. The expected size of $\mathcal{D}$ is $O(n)$ and the expected query time is $O(\log n)$.

Invariant: Before step i, $\mathcal{T}$ is a trapezoidal map for S_{i-1} and $\mathcal{D}$ is a valid search structure for $\mathcal{T}$.

Proof. – Correctness by loop invariant.

The 2d-Result

Theorem. TrapezoidalMap(S) computes $\mathcal{T}(S)$ for a set of n line segments in general position and a search structure $\mathcal{D}$ for $\mathcal{T}(S)$ in $O(n \log n)$ expected time. The expected size of $\mathcal{D}$ is $O(n)$ and the expected query time is $O(\log n)$.

Invariant: Before step i, $\mathcal{T}$ is a trapezoidal map for S_{i-1} and $\mathcal{D}$ is a valid search structure for $\mathcal{T}$.

Proof.

– Correctness by loop invariant.

– Query time similar to 1D analysis.

The 2d-Result

Theorem. TrapezoidalMap(S) computes $\mathcal{T}(S)$ for a set of n line segments in general position and a search structure $\mathcal{D}$ for $\mathcal{T}(S)$ in $O(n \log n)$ expected time. The expected size of $\mathcal{D}$ is $O(n)$ and the expected query time is $O(\log n)$.

Invariant: Before step i, $\mathcal{T}$ is a trapezoidal map for S_{i-1} and $\mathcal{D}$ is a valid search structure for $\mathcal{T}$.

Proof.
– Correctness by loop invariant.
– Query time similar to 1D analysis.
$\Rightarrow$ construction time

Query Time

Let $T(q)$ be the query time for a fixed query pt q.

Query Time

Let $T(q)$ be the query time for a fixed query pt q.

$\Rightarrow T(q) = O(\qquad)$.

Query Time

Let $T(q)$ be the query time for a fixed query pt q.

$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

Query Time

Let $T(q)$ be the query time for a fixed query pt q.
$\Rightarrow T(q) = O($length of the path from $\mathcal{D}$.root to $q)$.

height($\mathcal{D}$) increases by at most 3 in each step.

Query Time

Let $T(q)$ be the query time for a fixed query pt q.
$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq$

Query Time

Let $T(q)$ be the query time for a fixed query pt q.

$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

Query Time

Let $T(q)$ be the query time for a fixed query pt q.
$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the *expected* behaviour of $\mathcal{D}$:

Query Time

Let $T(q)$ be the query time for a fixed query pt q.

$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the *expected* behaviour of $\mathcal{D}$:

$\Rightarrow$ average of $T(q)$ over

Query Time

Let $T(q)$ be the query time for a fixed query pt q.

$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the *expected* behaviour of $\mathcal{D}$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders

Query Time

Let $T(q)$ be the query time for a fixed query pt q.

$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the *expected* behaviour of $\mathcal{D}$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of S)

Query Time

Let $T(q)$ be the query time for a fixed query pt q.

$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the *expected* behaviour of $\mathcal{D}$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of S)

$X_i :=$ # nodes that are added to the query path in iteration i.

Query Time

Let $T(q)$ be the query time for a fixed query pt q.
$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the *expected* behaviour of $\mathcal{D}$:
$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of S)

$X_i :=$ # nodes that are added to the query path in iteration i.
S and q are fixed.

Query Time

Let $T(q)$ be the query time for a fixed query pt q.
$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the *expected* behaviour of $\mathcal{D}$:
$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of S)

$X_i :=$ # nodes that are added to the query path in iteration i.
S and q are fixed.
$\Rightarrow X_i$ random var. that depends only on insertion order of S.

Query Time

Let $T(q)$ be the query time for a fixed query pt q.

$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the *expected* behaviour of $\mathcal{D}$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of S)

$X_i :=$ # nodes that are added to the query path in iteration i.

S and q are fixed.

$\Rightarrow X_i$ random var. that depends only on insertion order of S.

$\Rightarrow$ expected path length from $\mathcal{D}$.root to q is

Query Time

Let $T(q)$ be the query time for a fixed query pt q.

$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the *expected* behaviour of $\mathcal{D}$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of S)

$X_i :=$ # nodes that are added to the query path in iteration i.

S and q are fixed.

$\Rightarrow X_i$ random var. that depends only on insertion order of S.

$\Rightarrow$ expected path length from $\mathcal{D}$.root to q is

$\mathbf{E}[\sum_{i=1}^{n} X_i] =$

Query Time

Let $T(q)$ be the query time for a fixed query pt q.
$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$.

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \le 3n$.

We are interested in the *expected* behaviour of $\mathcal{D}$:
$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of S)

$X_i :=$ # nodes that are added to the query path in iteration i.
S and q are fixed.
$\Rightarrow X_i$ random var. that depends only on insertion order of S.
$\Rightarrow$ expected path length from $\mathcal{D}$.root to q is

$$\mathbf{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbf{E}[X_i] = ?$$

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] =$

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq$

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq \; 3 \cdot \mathbf{P}[X_i \geq 1] =$

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq 3 \cdot \mathbf{P}[X_i \geq 1] = 3p_i$

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq 3 \cdot \mathbf{P}[X_i \geq 1] = 3p_i$

$\Delta_q(S_i) :=$ trapezoid in $\mathcal{T}(S_i)$ that contains q.

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq 3 \cdot \mathbf{P}[X_i \geq 1] = 3p_i$

$\Delta_q(S_i) :=$ trapezoid in $\mathcal{T}(S_i)$ that contains q.

Key idea: Iteration i contributes a node to Π_q iff

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq 3 \cdot \mathbf{P}[X_i \geq 1] = 3p_i$

$\Delta_q(S_i) :=$ trapezoid in $\mathcal{T}(S_i)$ that contains q.

Key idea: Iteration i contributes a node to Π_q iff $\Delta_q(S_{i-1}) \neq \Delta_q(S_i)$.

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq 3 \cdot \mathbf{P}[X_i \geq 1] = 3p_i$

$\Delta_q(S_i) :=$ trapezoid in $\mathcal{T}(S_i)$ that contains q.

Key idea: Iteration i contributes a node to Π_q iff $\Delta_q(S_{i-1}) \neq \Delta_q(S_i)$.

In this case $\Delta_q(S_i)$ must have been created in iteration i.

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq 3 \cdot \mathbf{P}[X_i \geq 1] = 3p_i$

$\Delta_q(S_i) :=$ trapezoid in $\mathcal{T}(S_i)$ that contains q.

Key idea: Iteration i contributes a node to Π_q iff $\Delta_q(S_{i-1}) \neq \Delta_q(S_i)$.

In this case $\Delta_q(S_i)$ must have been created in iteration i.

$\Rightarrow \Delta := \Delta_q(S_i)$ is adjacent to the new segment s_i.

Query Time (cont'd)

p_i = prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq 3 \cdot \mathbf{P}[X_i \geq 1] = 3p_i$

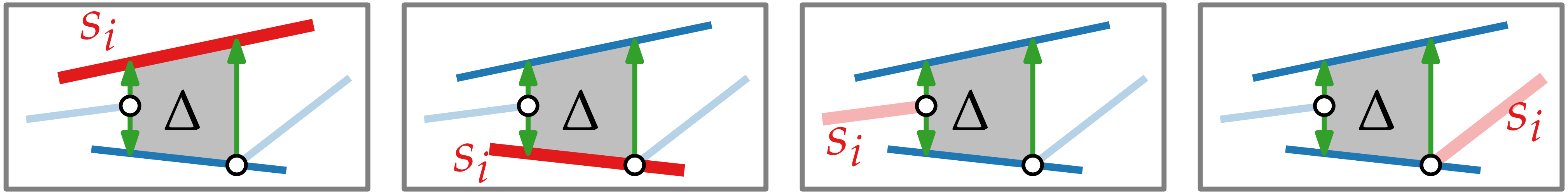
$\Delta_q(S_i) :=$ trapezoid in $\mathcal{T}(S_i)$ that contains q.

Key idea: Iteration i contributes a node to Π_q iff $\Delta_q(S_{i-1}) \neq \Delta_q(S_i)$.

In this case $\Delta_q(S_i)$ must have been created in iteration i.

$\Rightarrow \Delta := \Delta_q(S_i)$ is adjacent to the new segment s_i.

$\Rightarrow \text{top}(\Delta) = s_i$, $\text{bot}(\Delta) = s_i$, $\text{leftp}(\Delta) \in s_i$, or $\text{rightp}(\Delta) \in s_i$.



Query Time (cont'd)

p_i = prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq 3 \cdot \mathbf{P}[X_i \geq 1] = 3p_i$

$\Delta_q(S_i) :=$ trapezoid in $\mathcal{T}(S_i)$ that contains q.

Key idea: Iteration i contributes a node to Π_q iff $\Delta_q(S_{i-1}) \neq \Delta_q(S_i)$.

In this case $\Delta_q(S_i)$ must have been created in iteration i.

$\Rightarrow \Delta := \Delta_q(S_i)$ is adjacent to the new segment s_i.

$\Rightarrow \text{top}(\Delta) = s_i$, $\text{bot}(\Delta) = s_i$, $\text{leftp}(\Delta) \in s_i$, or $\text{rightp}(\Delta) \in s_i$.

Trick: $\mathcal{T}(S_i)$ (and thus Δ) is uniquely determined by S_i.

Query Time (cont'd)

p_i = prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq 3 \cdot \mathbf{P}[X_i \geq 1] = 3p_i$

$\Delta_q(S_i) :=$ trapezoid in $\mathcal{T}(S_i)$ that contains q.

Key idea: Iteration i contributes a node to Π_q iff $\Delta_q(S_{i-1}) \neq \Delta_q(S_i)$.

In this case $\Delta_q(S_i)$ must have been created in iteration i.

$\Rightarrow \Delta := \Delta_q(S_i)$ is adjacent to the new segment s_i.

$\Rightarrow \text{top}(\Delta) = s_i$, $\text{bot}(\Delta) = s_i$, $\text{leftp}(\Delta) \in s_i$, or $\text{rightp}(\Delta) \in s_i$.

Trick: $\mathcal{T}(S_i)$ (and thus Δ) is uniquely determined by S_i. Consider $S_i \subseteq S$ fixed.

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.

$\Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbf{P}[X_i = j] \leq \ 3 \cdot \mathbf{P}[X_i \geq 1] = \ 3p_i$

$\Delta_q(S_i) :=$ trapezoid in $\mathcal{T}(S_i)$ that contains q.

Key idea: Iteration i contributes a node to Π_q iff $\Delta_q(S_{i-1}) \neq \Delta_q(S_i)$.

In this case $\Delta_q(S_i)$ must have been created in iteration i.

$\Rightarrow \Delta := \Delta_q(S_i)$ is adjacent to the new segment s_i.

$\Rightarrow$ top$(\Delta) = s_i$, bot$(\Delta) = s_i$, leftp$(\Delta) \in s_i$, or rightp$(\Delta) \in s_i$.

Trick: $\mathcal{T}(S_i)$ (and thus Δ) is uniquely determined by S_i.
Consider $S_i \subseteq S$ fixed.
$\Rightarrow \Delta$ does *not* depend on insertion order.

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.
i.e., prob. that Δ changes when inserting s_i.

Aim: bound p_i.

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.
i.e., prob. that Δ changes when inserting s_i.

Aim: bound p_i.

Tool:

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.
i.e., prob. that Δ changes when inserting s_i.

Aim: bound p_i.
Tool: *Backwards analysis!*

Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.
i.e., prob. that Δ changes when inserting s_i.

Aim: bound p_i.
Tool: *Backwards analysis!*

$p_i =$ prob. that Δ changes when s_i is removed

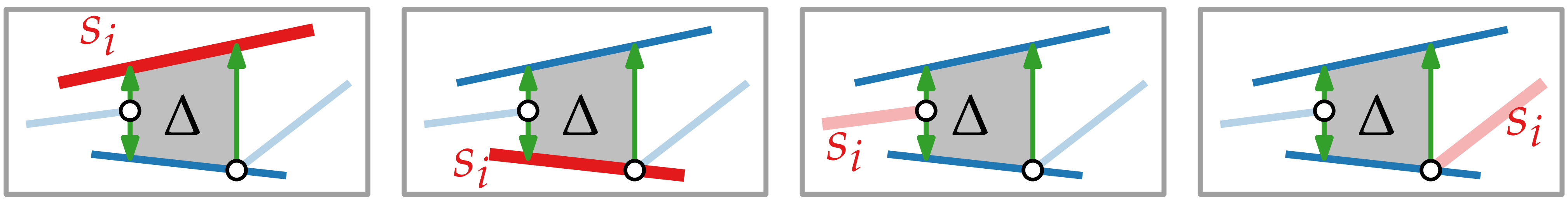
Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.
i.e., prob. that Δ changes when inserting s_i.

Aim: bound p_i.
Tool: *Backwards analysis!*

$p_i =$ prob. that Δ changes when s_i is removed

Four cases:



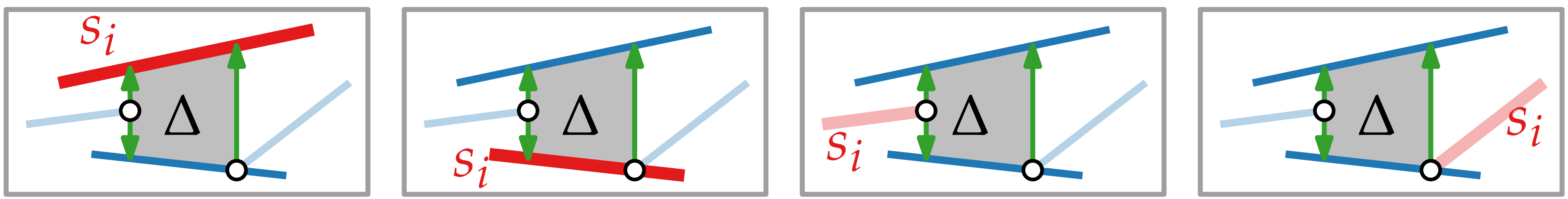
Query Time (cont'd)

$p_i =$ prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.
i.e., prob. that Δ changes when inserting s_i.

Aim: bound p_i.
Tool: *Backwards analysis!*

$p_i =$ prob. that Δ changes when s_i is removed

Four cases:



$\mathbf{P}(\text{top}(\Delta) = s_i)$ =?

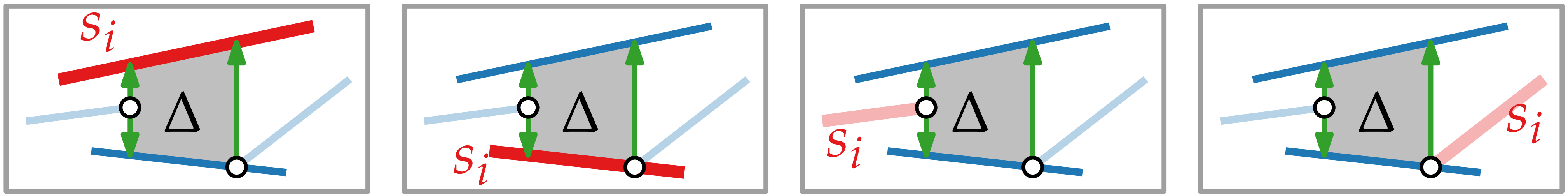
Query Time (cont'd)

p_i = prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.
i.e., prob. that Δ changes when inserting s_i.

Aim: bound p_i.
Tool: *Backwards analysis!*

p_i = prob. that Δ changes when s_i is removed

Four cases:



$\mathbf{P}(\mathrm{top}(\Delta) = s_i) = 1/i$ (since exactly 1 of i segments is $\mathrm{top}(\Delta)$).

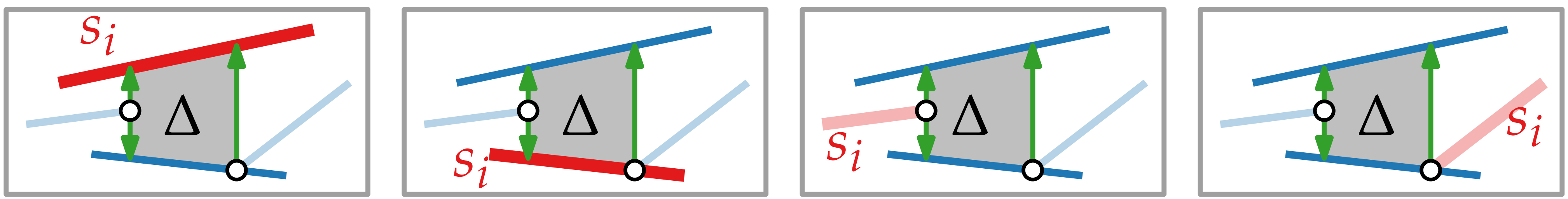
Query Time (cont'd)

p_i = prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.
i.e., prob. that Δ changes when inserting s_i.

Aim: bound p_i.
Tool: *Backwards analysis!*

p_i = prob. that Δ changes when s_i is removed

Four cases:



$\mathbf{P}(\text{top}(\Delta) = s_i) = 1/i$ (since exactly 1 of i segments is $\text{top}(\Delta)$).

$\Rightarrow p_i \leq 4/i$

$$\Rightarrow \mathbf{E}[\textstyle\sum_{i=1}^n X_i] = \sum_{i=1}^n \mathbf{E}[X_i] \leq \sum_{i=1}^n 3 \cdot p_i$$

$$= 12 \textstyle\sum_{i=1}^n 1/i \in O(\log n)$$

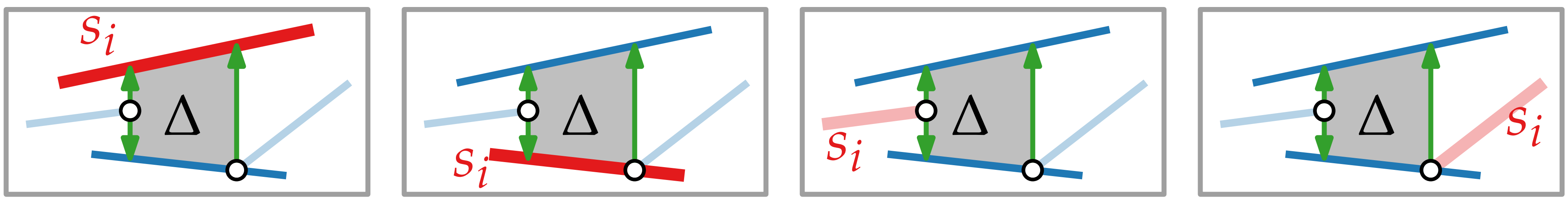
Query Time (cont'd)

p_i = prob. that the search path Π_q of q in $\mathcal{D}$ contains a node that was created in iteration i.
i.e., prob. that Δ changes when inserting s_i.

Aim: bound p_i.
Tool: *Backwards analysis!*

p_i = prob. that Δ changes when s_i is removed

Four cases:



$\mathbf{P}(\text{top}(\Delta) = s_i) = 1/i$ (since exactly 1 of i segments is top(Δ)).

$\Rightarrow p_i \leq 4/i$

$$\Rightarrow \mathbf{E}[\textstyle\sum_{i=1}^n X_i] = \sum_{i=1}^n \mathbf{E}[X_i] \leq \sum_{i=1}^n 3 \cdot p_i$$

$$= 12 \textstyle\sum_{i=1}^n 1/i \in O(\log n)$$

$\square$