

Chapter 16

Simplex Range Searching

or:

Windowing Revisited

Range-Counting Query

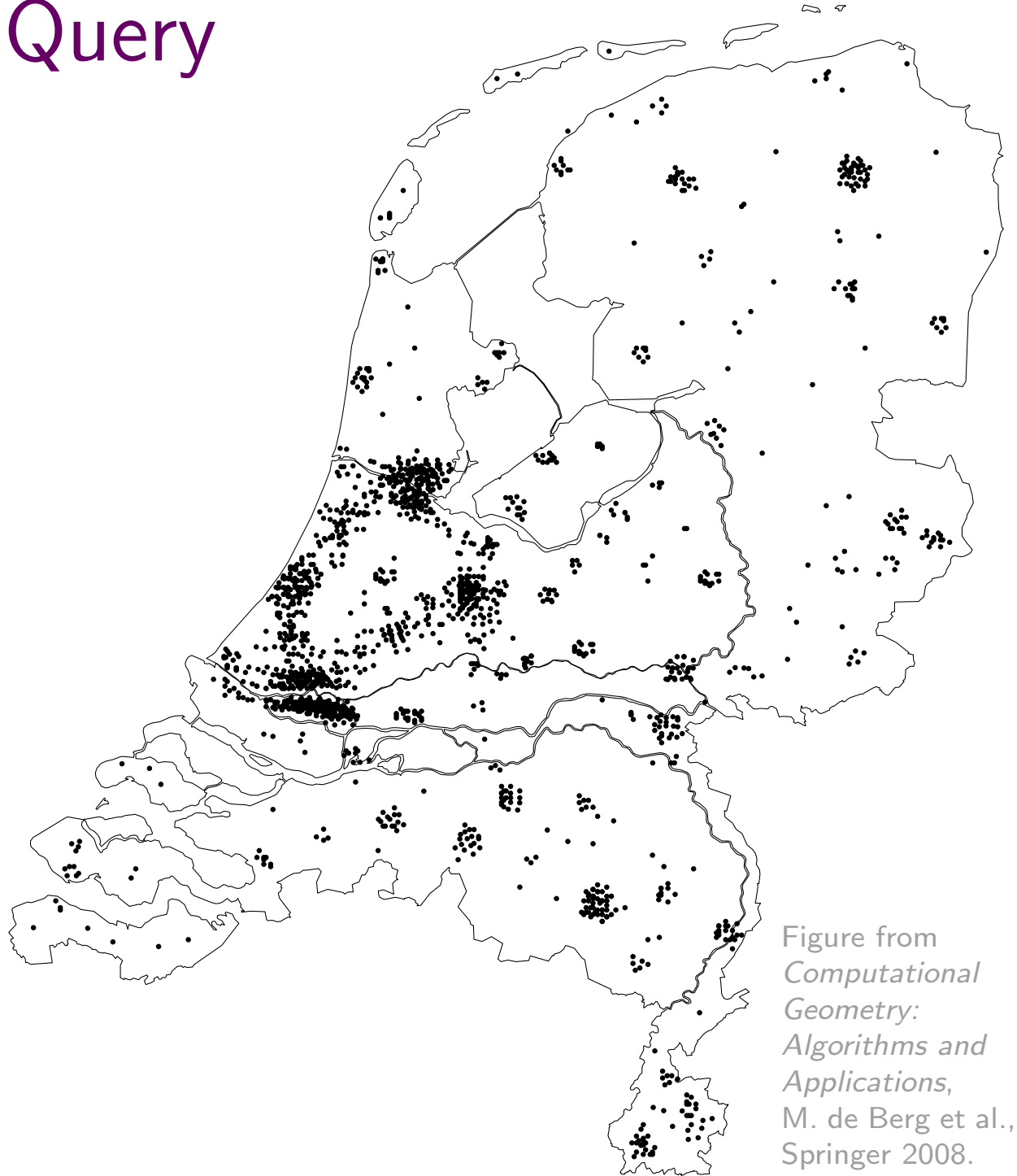


Figure from
*Computational
Geometry:
Algorithms and
Applications*,
M. de Berg et al.,
Springer 2008.

Range-Counting Query

area affected by
the construction
of a new airport

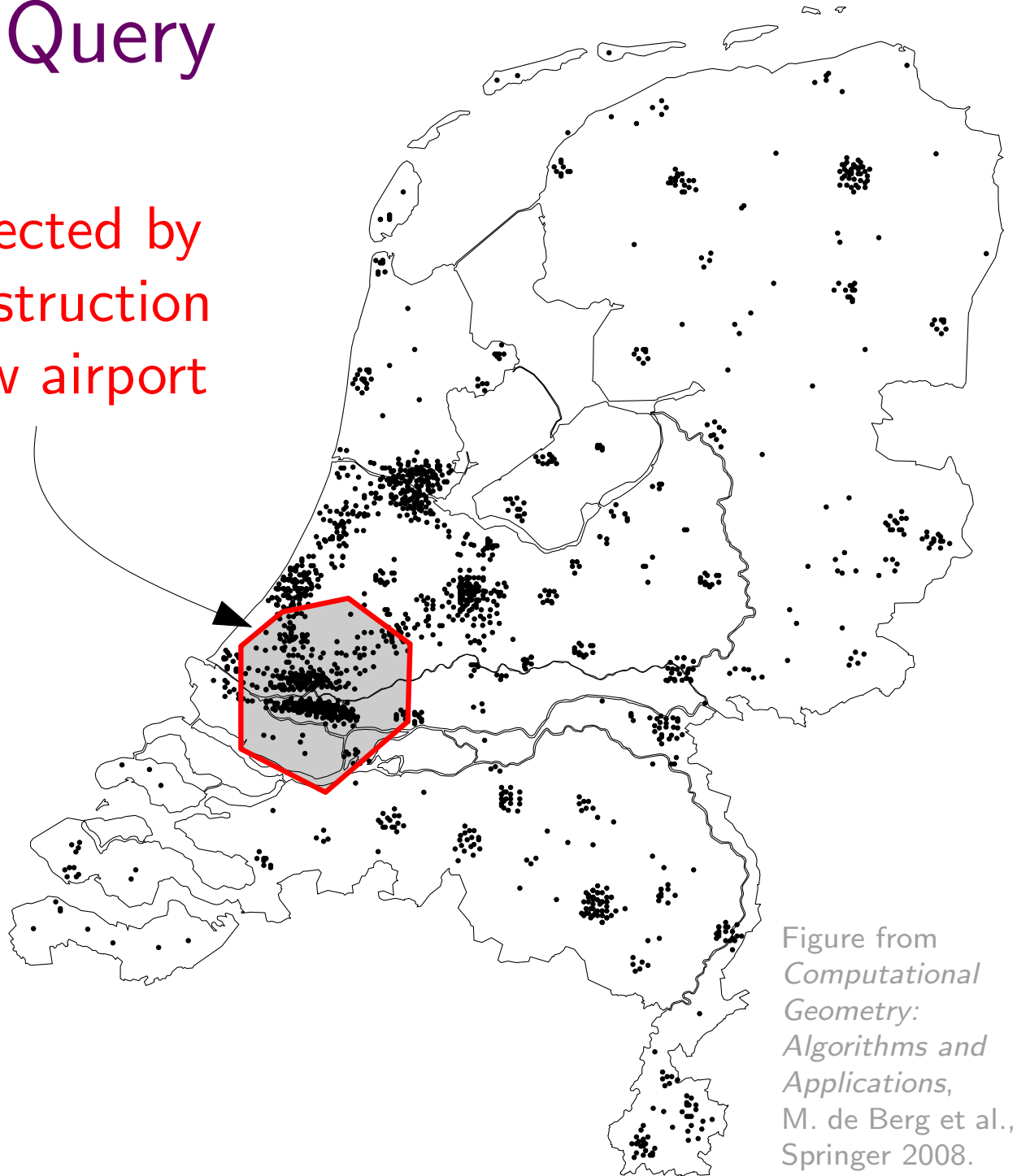


Figure from
*Computational
Geometry:
Algorithms and
Applications*,
M. de Berg et al.,
Springer 2008.

Range-Counting Query

area affected by
the construction
of a new airport

Observation:

Query range
depends on, e.g.,
dominant wind
directions

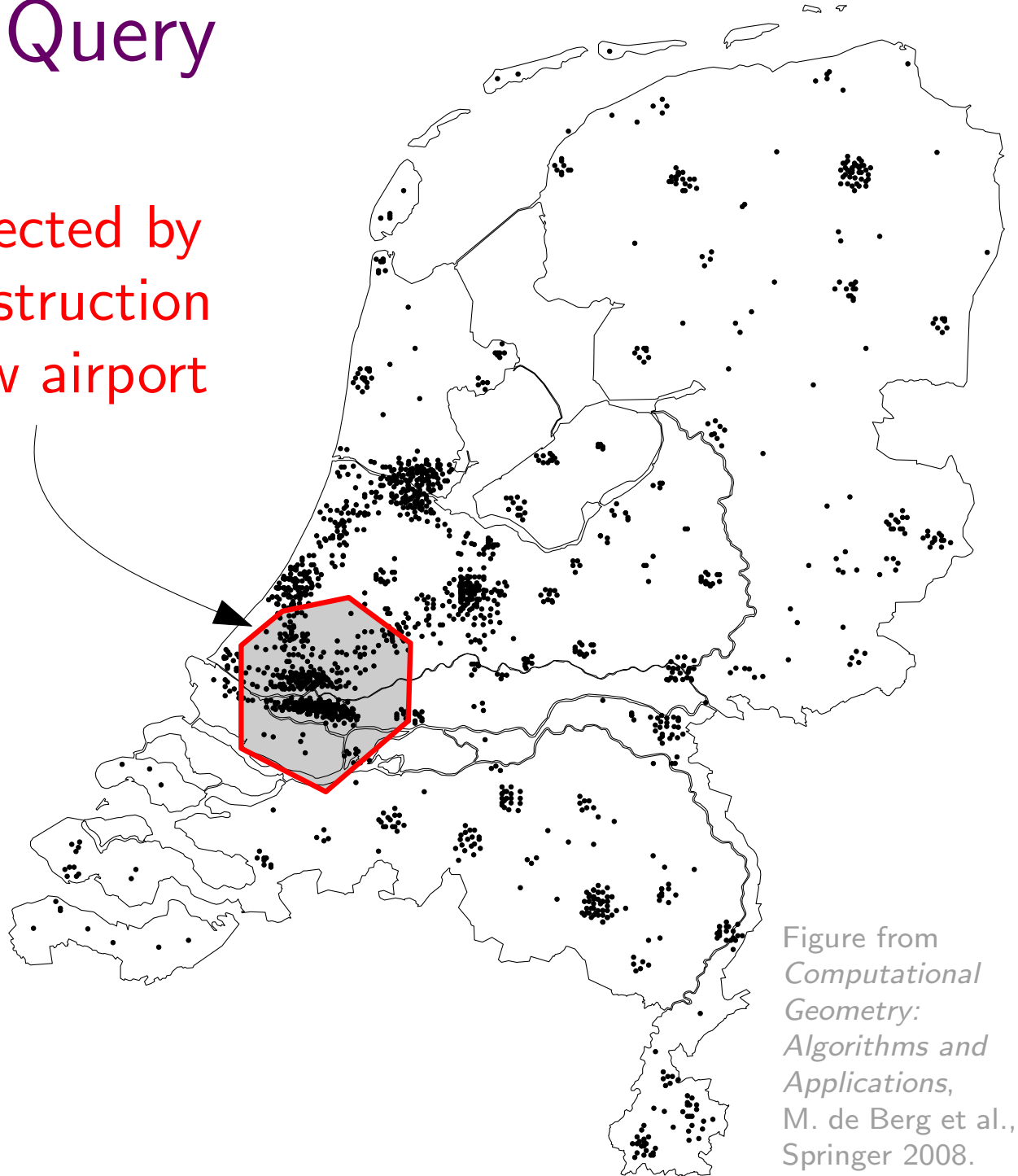
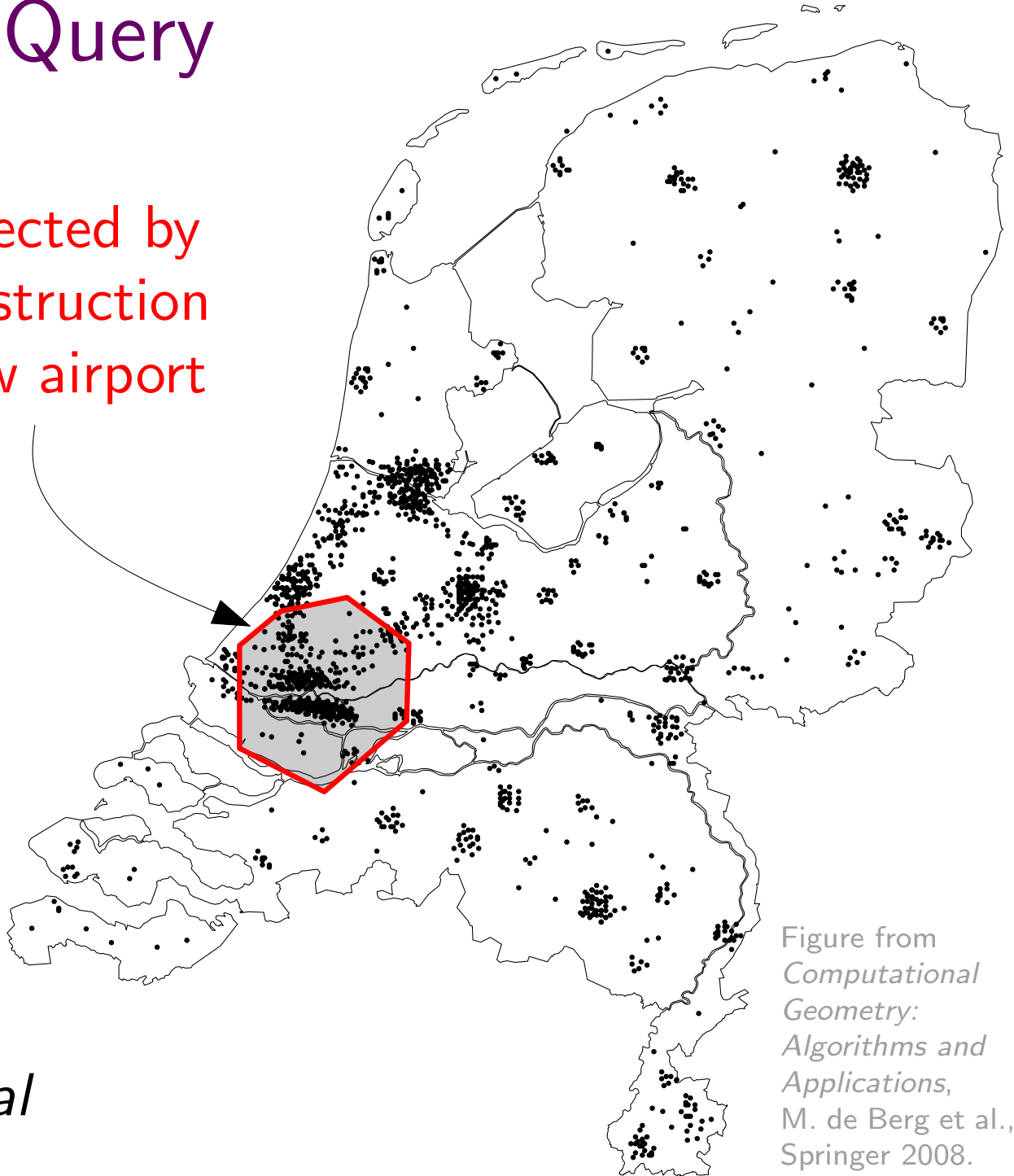


Figure from
*Computational
Geometry:
Algorithms and
Applications*,
M. de Berg et al.,
Springer 2008.

Range-Counting Query

area affected by
the construction
of a new airport



Observation:

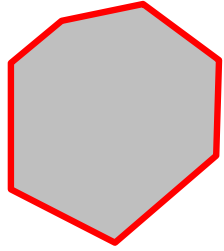
Query range
depends on, e.g.,
dominant wind
directions

⇒ *non-orthogonal*

Figure from
*Computational
Geometry:
Algorithms and
Applications*,
M. de Berg et al.,
Springer 2008.

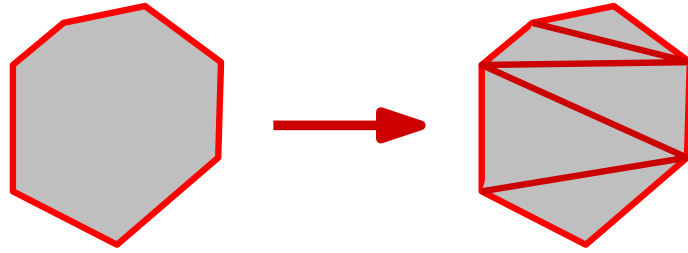
Non-orthogonal range queries

Query range:



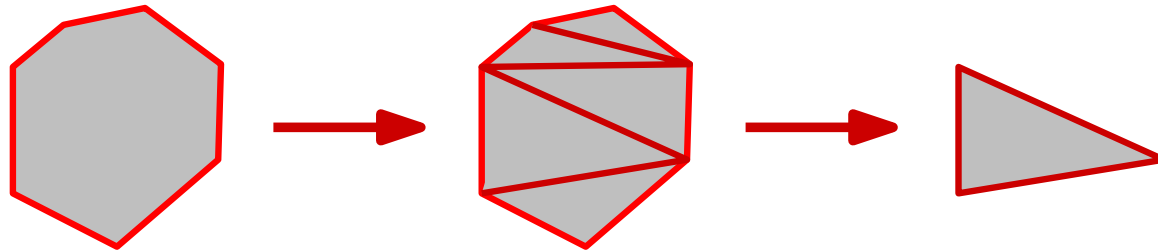
Non-orthogonal range queries

Query range:



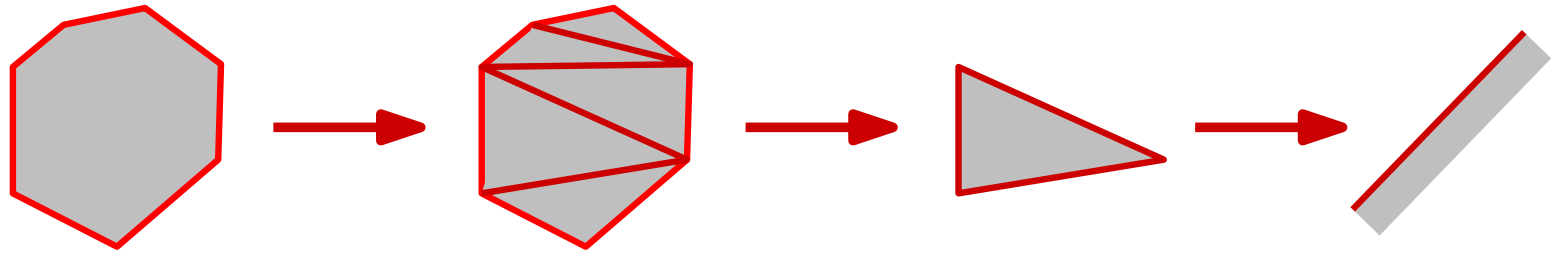
Non-orthogonal range queries

Query range:



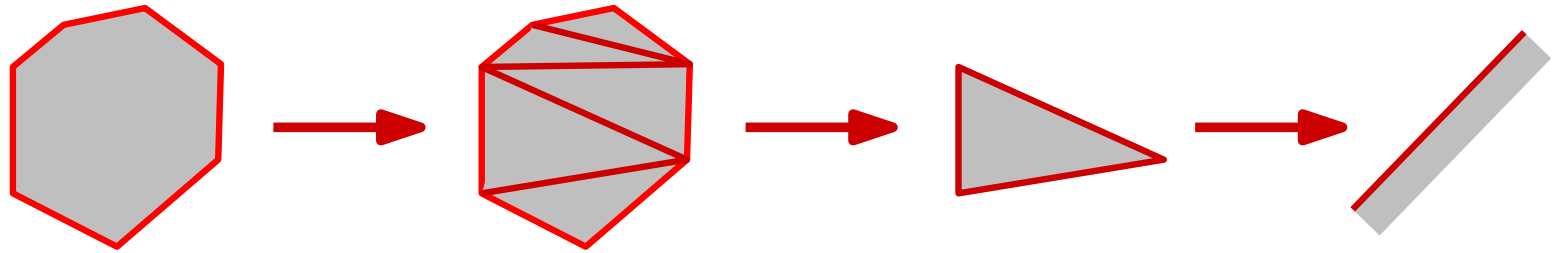
Non-orthogonal range queries

Query range:



Non-orthogonal range queries

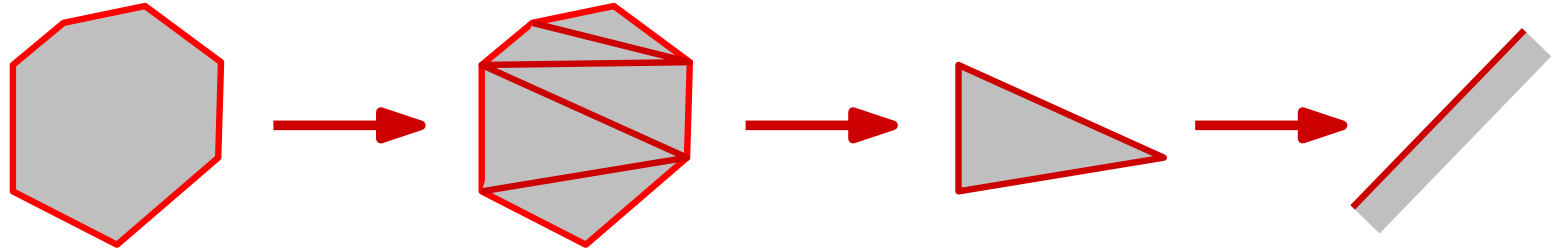
Query range:



Task: Design a data structure for the one-dimensional case!

Non-orthogonal range queries

Query range:

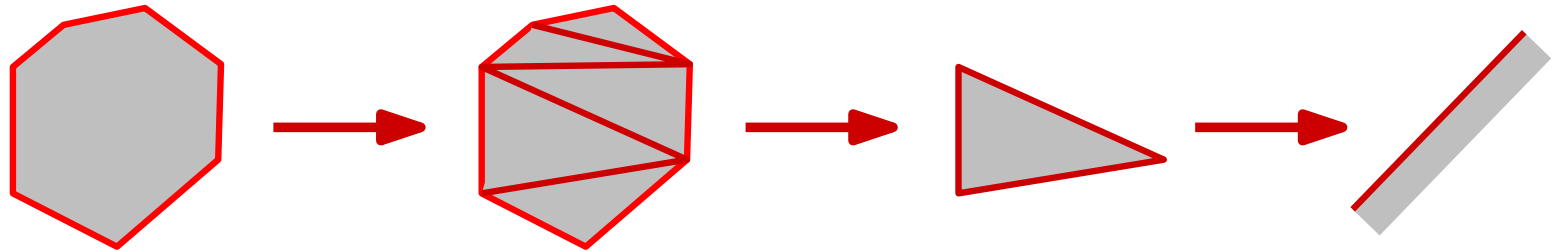


Task: Design a data structure for the one-dimensional case!

Given a set P of n numbers, preprocess P such that *half-space range-counting queries* can be answered quickly.

Non-orthogonal range queries

Query range:



Task: Design a data structure for the one-dimensional case!

Given a set P of n numbers, preprocess P such that *half-space range-counting queries* can be answered quickly.

Given a number x , return $|P \cap [x, \infty)|$.

The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution:

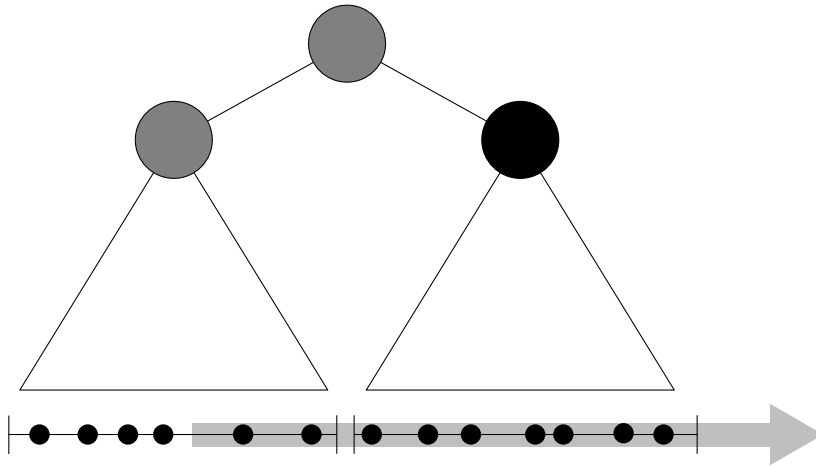
- use balanced binary search trees
- augment each node with the number of nodes in its subtree

The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution:

- use balanced binary search trees
- augment each node with the number of nodes in its subtree

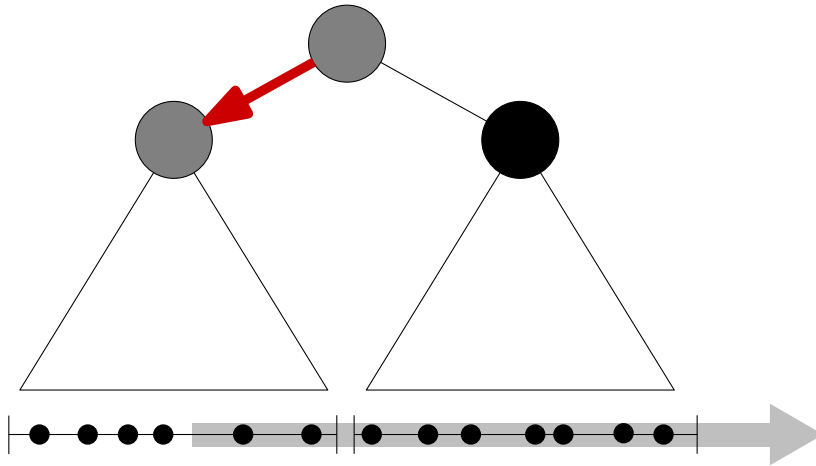


The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution:

- use balanced binary search trees
- augment each node with the number of nodes in its subtree

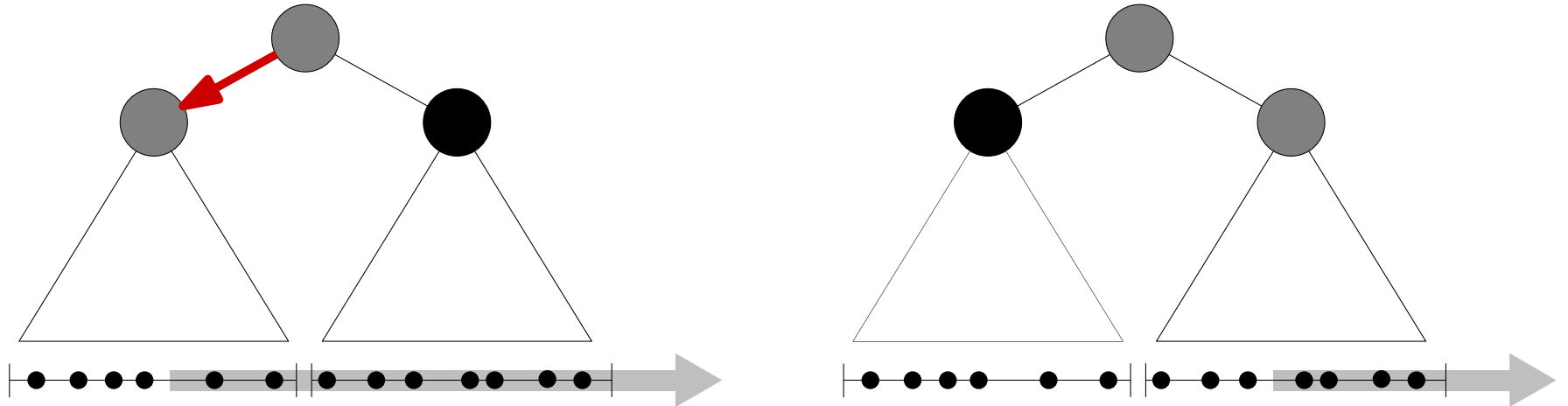


The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution:

- use balanced binary search trees
- augment each node with the number of nodes in its subtree

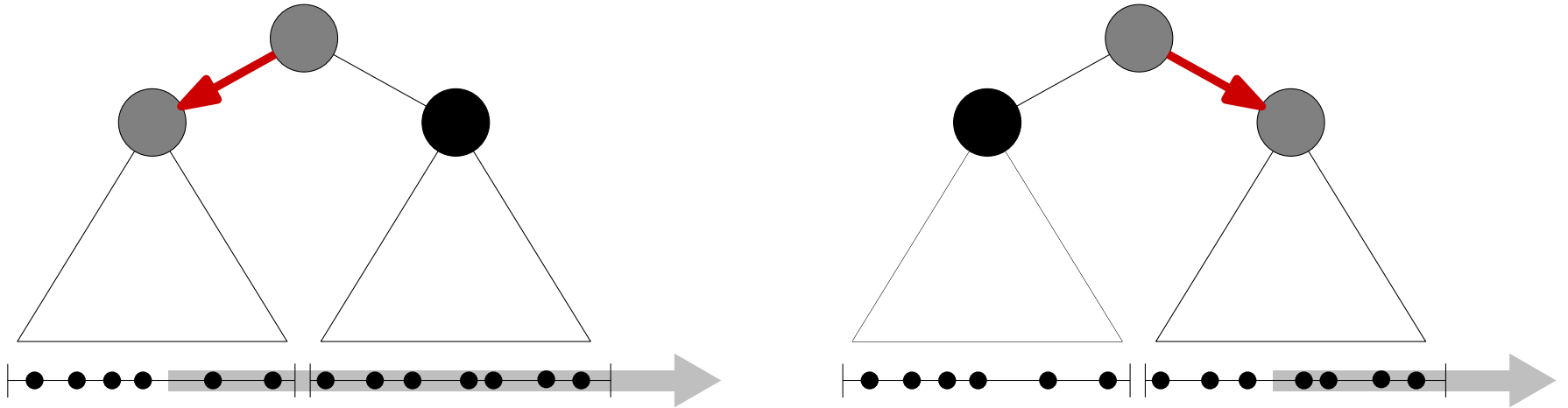


The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution:

- use balanced binary search trees
- augment each node with the number of nodes in its subtree

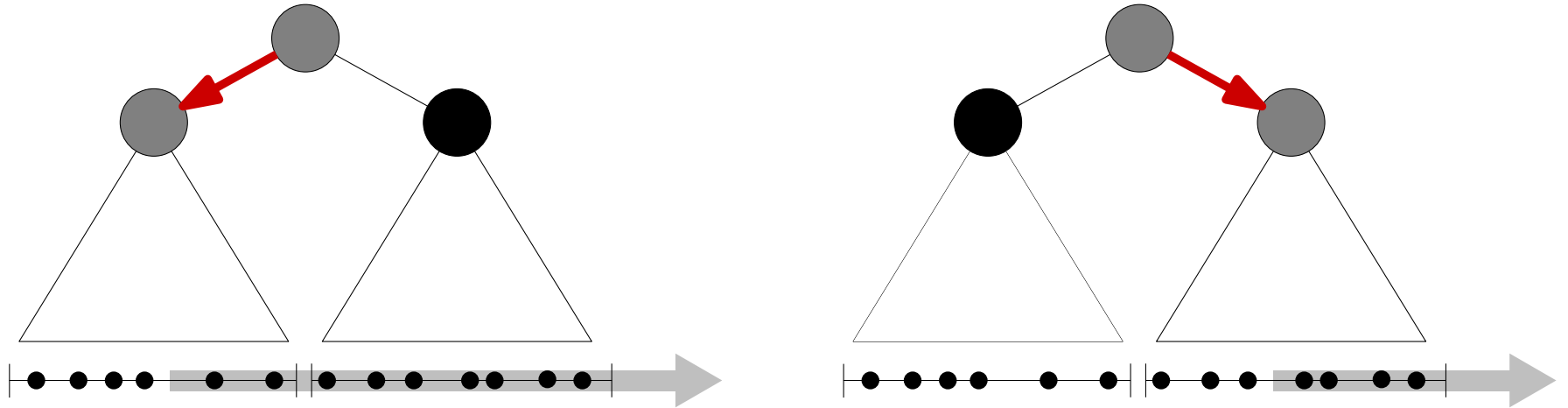


The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution:

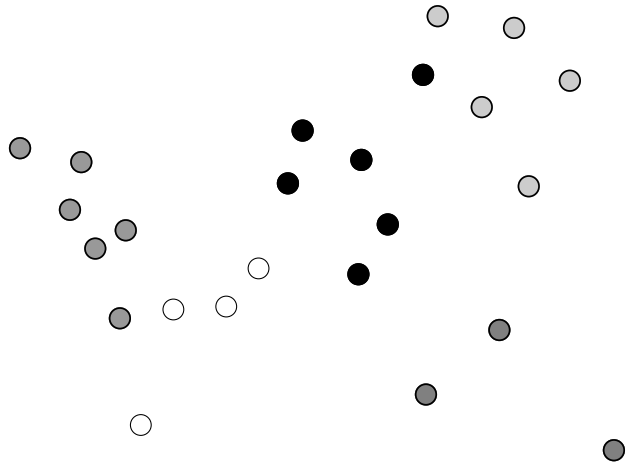
- use balanced binary search trees
- augment each node with the number of nodes in its subtree



Lesson: On each level, must visit ≤ 1 subtree recursively!

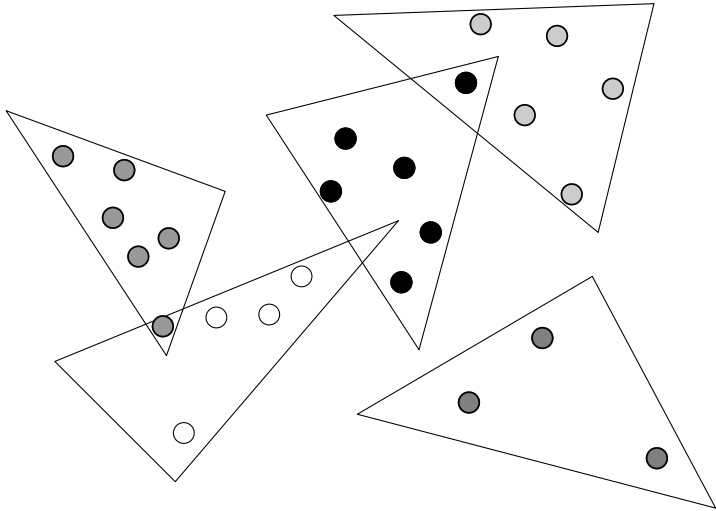
Generalizing to 2 Dimensions

Any ideas?



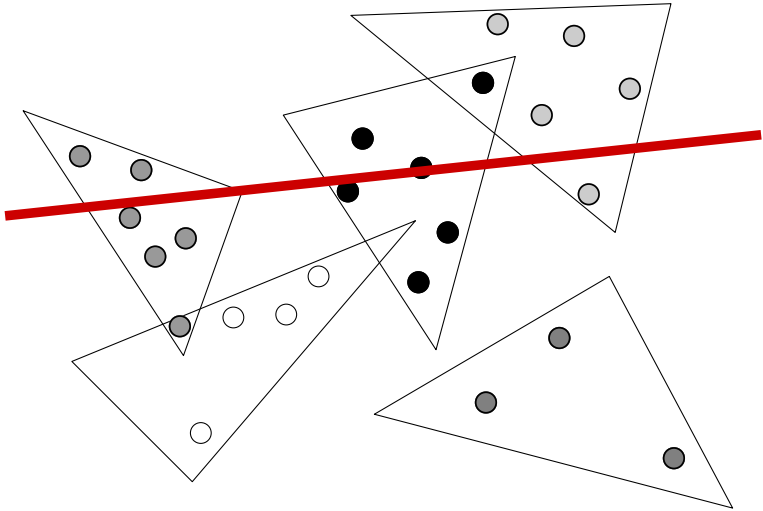
Generalizing to 2 Dimensions

Any ideas?



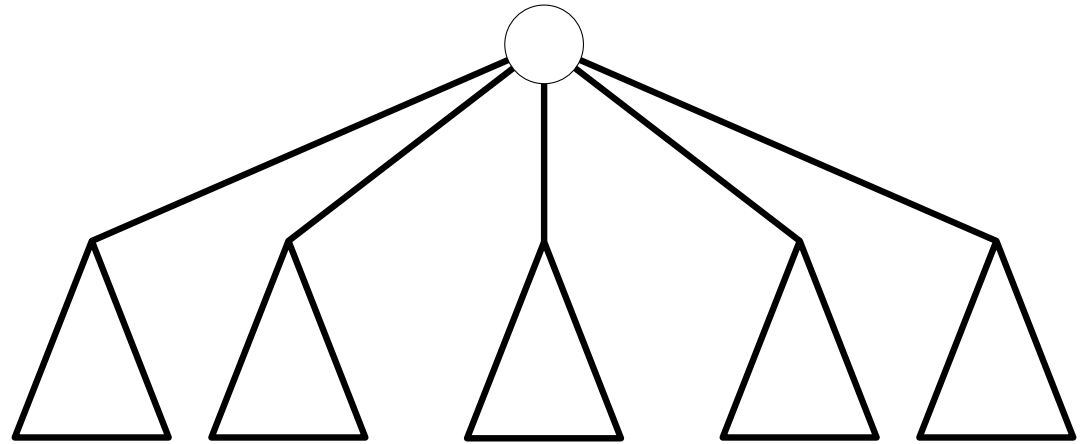
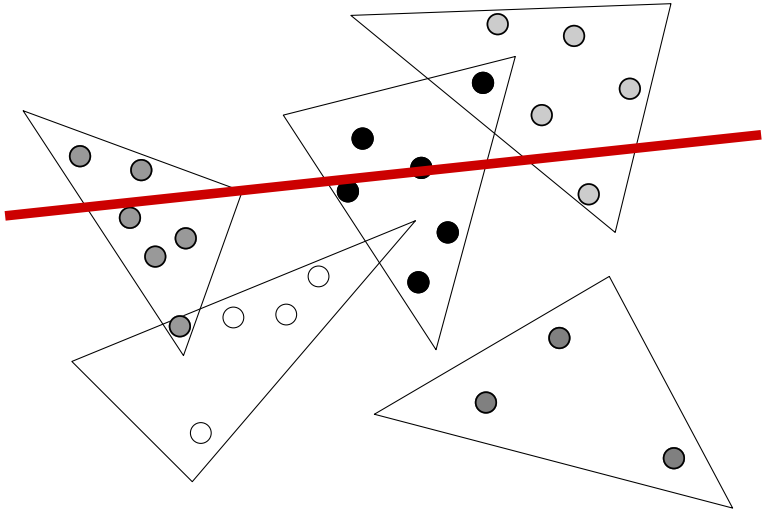
Generalizing to 2 Dimensions

Any ideas?



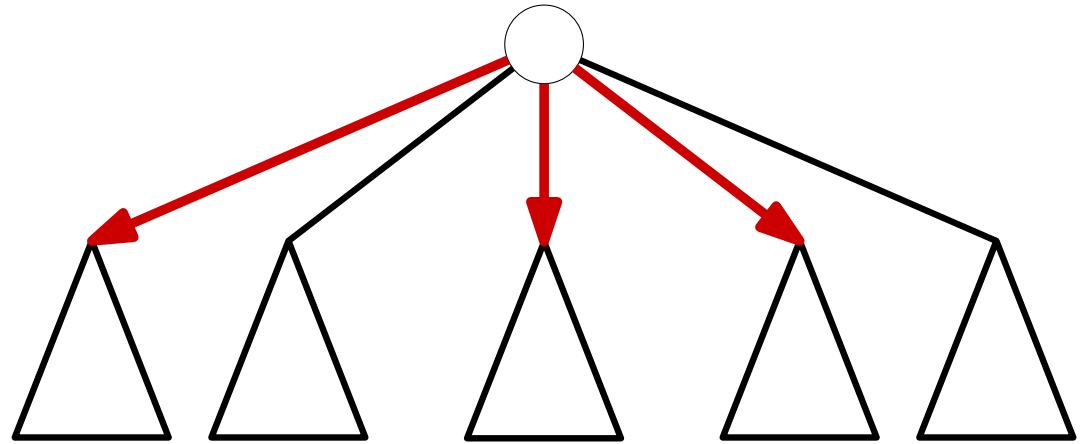
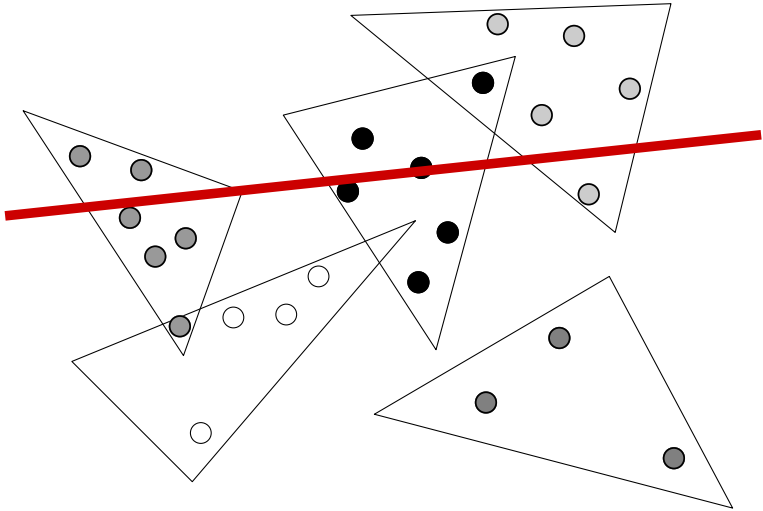
Generalizing to 2 Dimensions

Any ideas?



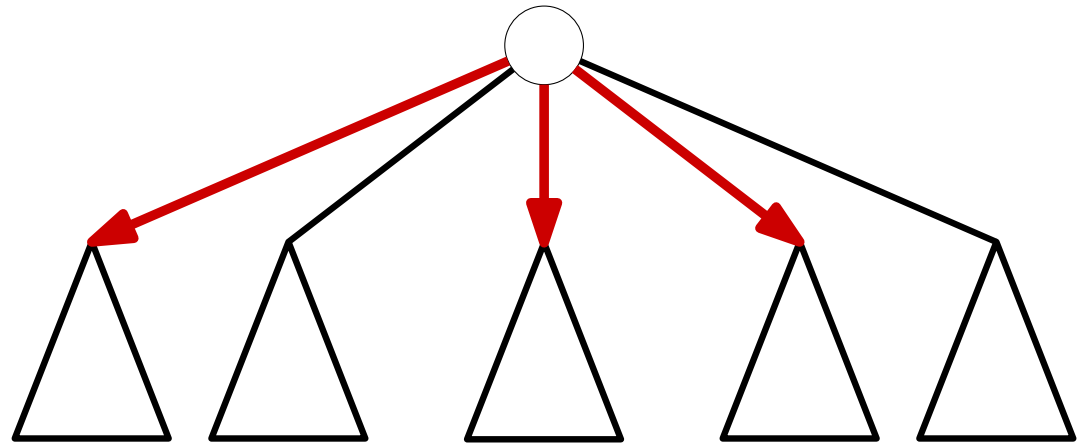
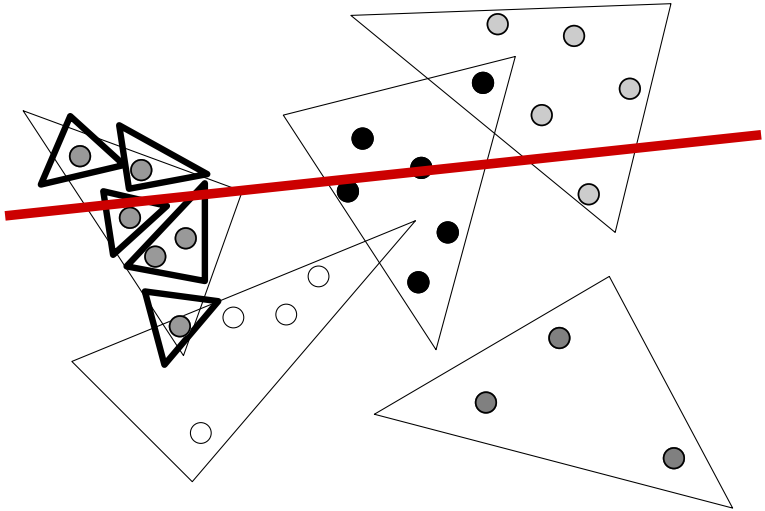
Generalizing to 2 Dimensions

Any ideas?



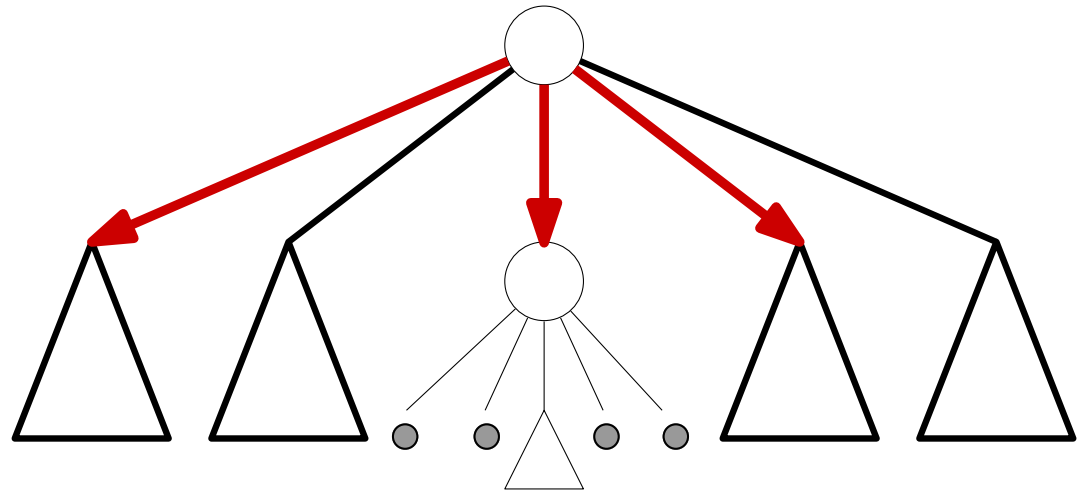
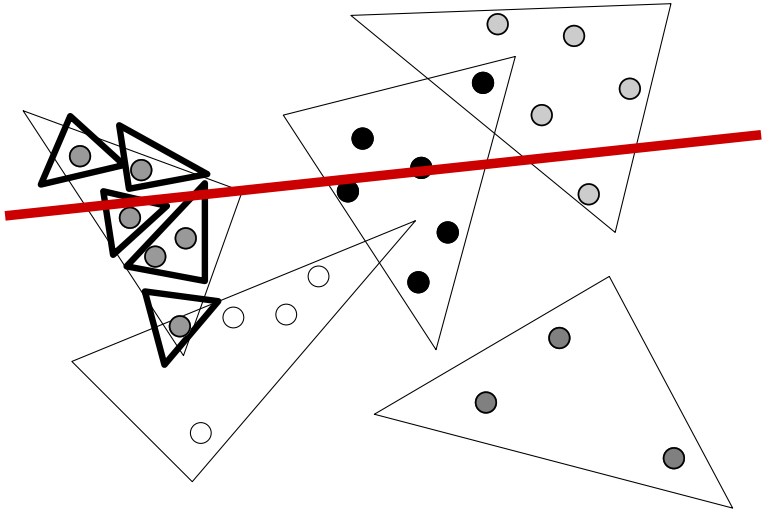
Generalizing to 2 Dimensions

Any ideas?



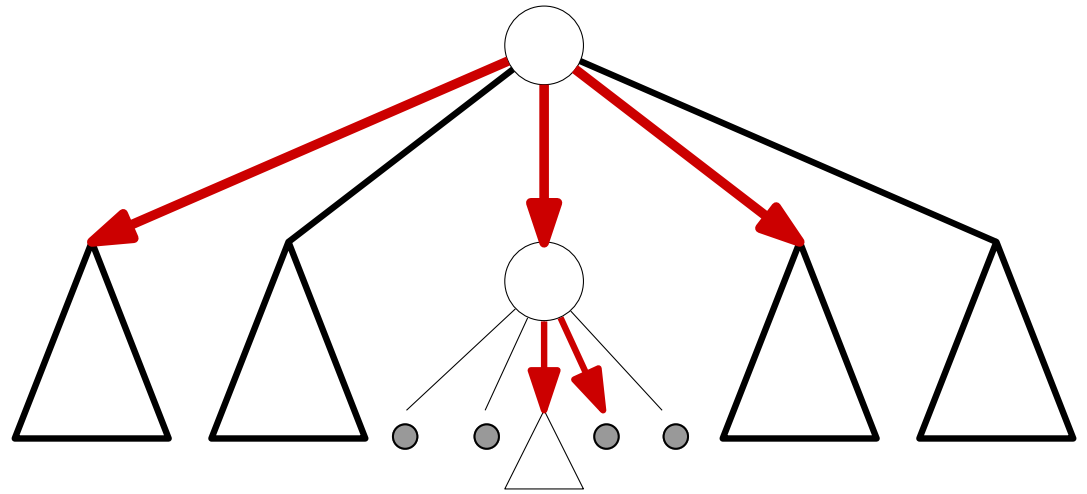
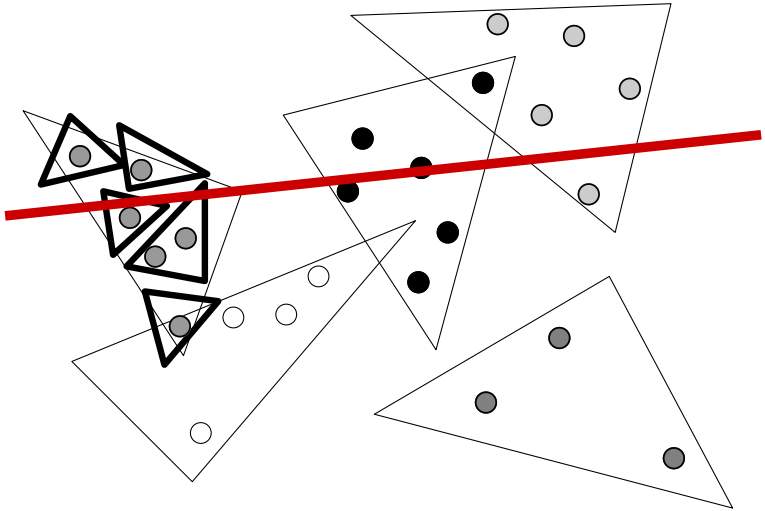
Generalizing to 2 Dimensions

Any ideas?



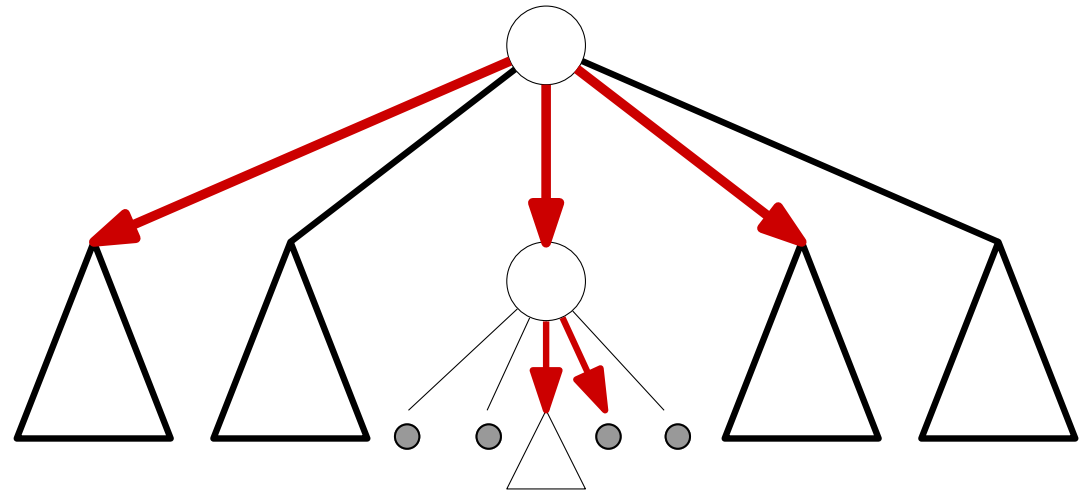
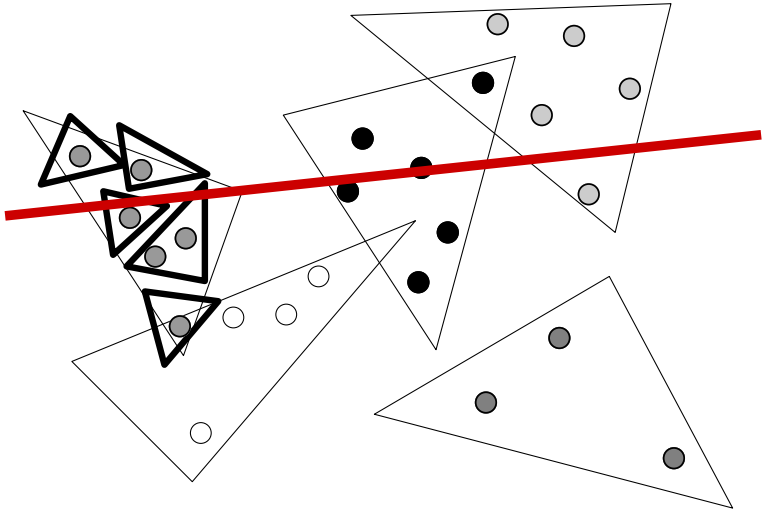
Generalizing to 2 Dimensions

Any ideas?



Generalizing to 2 Dimensions

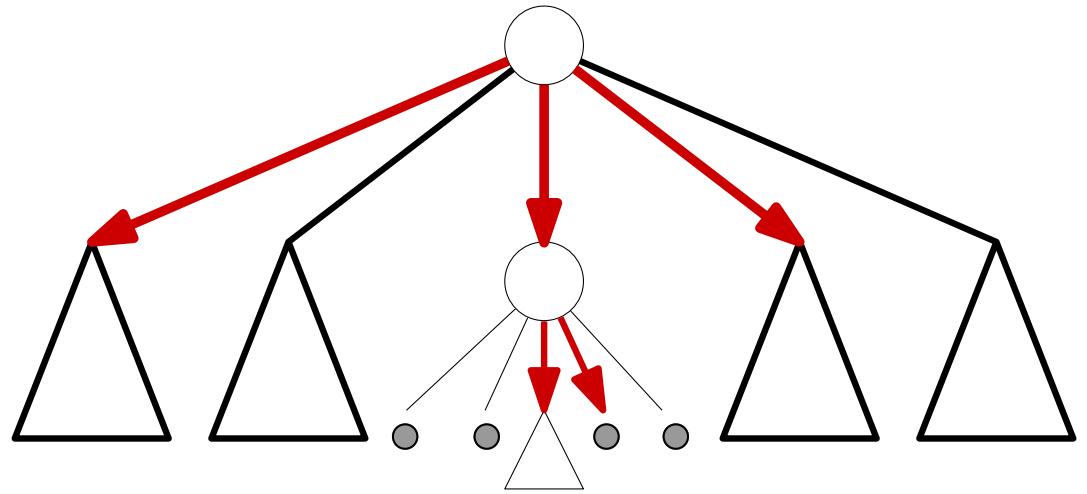
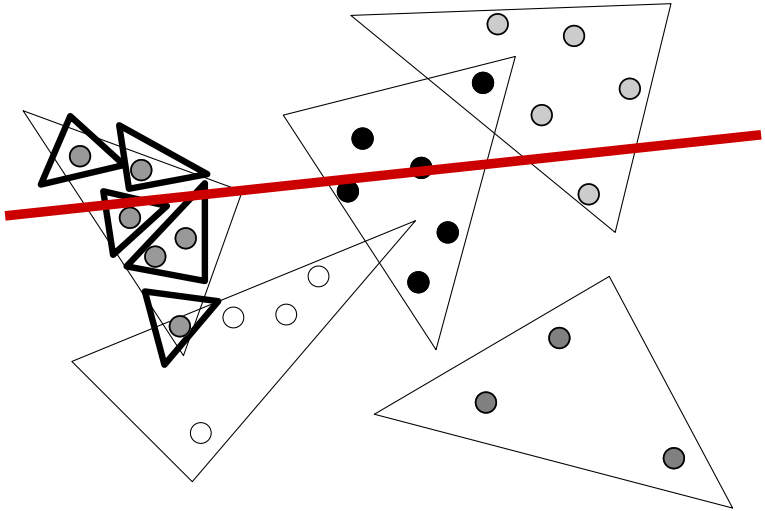
Any ideas?



Definition: $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \dots, (S_r, t_r)\}$ is a *simplicial partition* (of size r) for S if

Generalizing to 2 Dimensions

Any ideas?

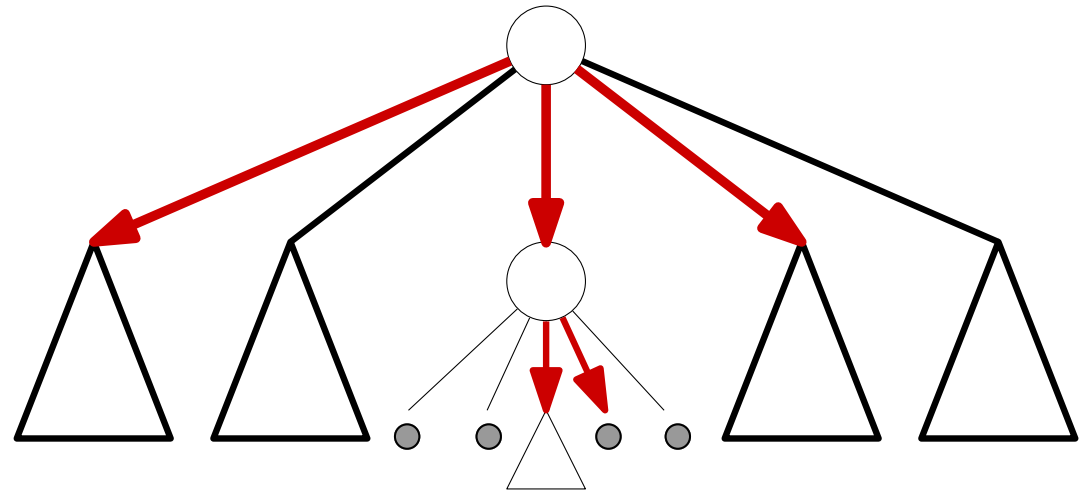
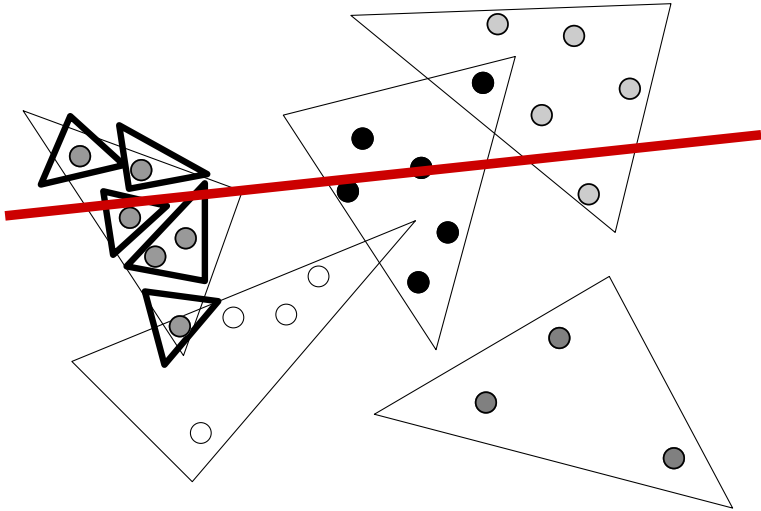


Definition: $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \dots, (S_r, t_r)\}$ is a *simplicial partition* (of size r) for S if

- S is partitioned by S_1, \dots, S_r and
- for $1 \leq i \leq r$, t_i is a triangle and $S_i \subset t_i$.

Generalizing to 2 Dimensions

Any ideas?

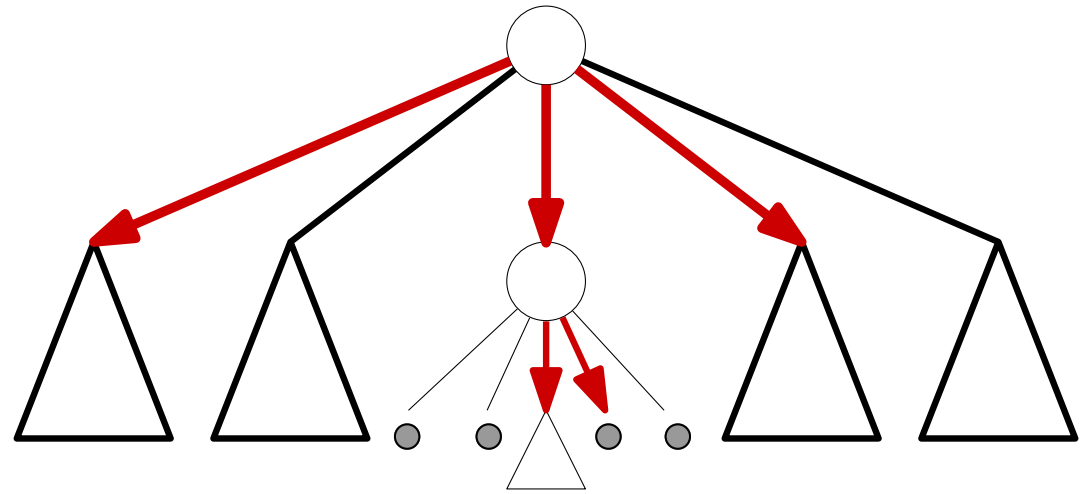
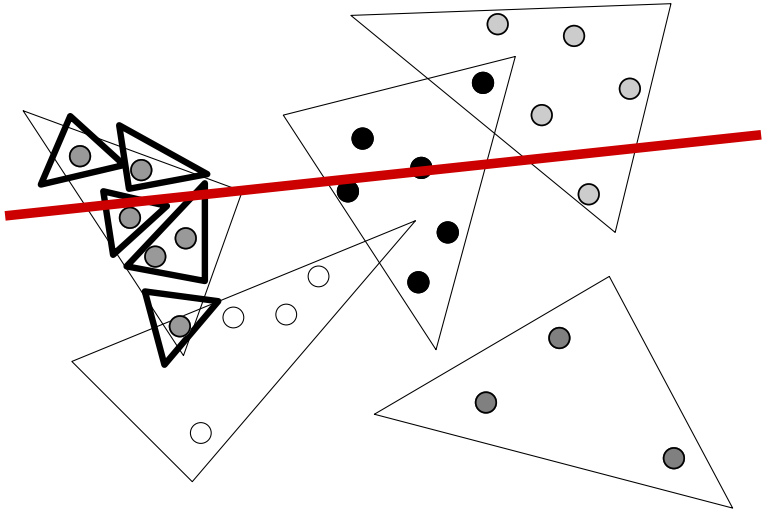


Definition: $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \dots, (S_r, t_r)\}$ is a *simplicial partition* (of size r) for S if *classes of S*

- S is partitioned by S_1, \dots, S_r and
- for $1 \leq i \leq r$, t_i is a triangle and $S_i \subset t_i$.

Generalizing to 2 Dimensions

Any ideas?



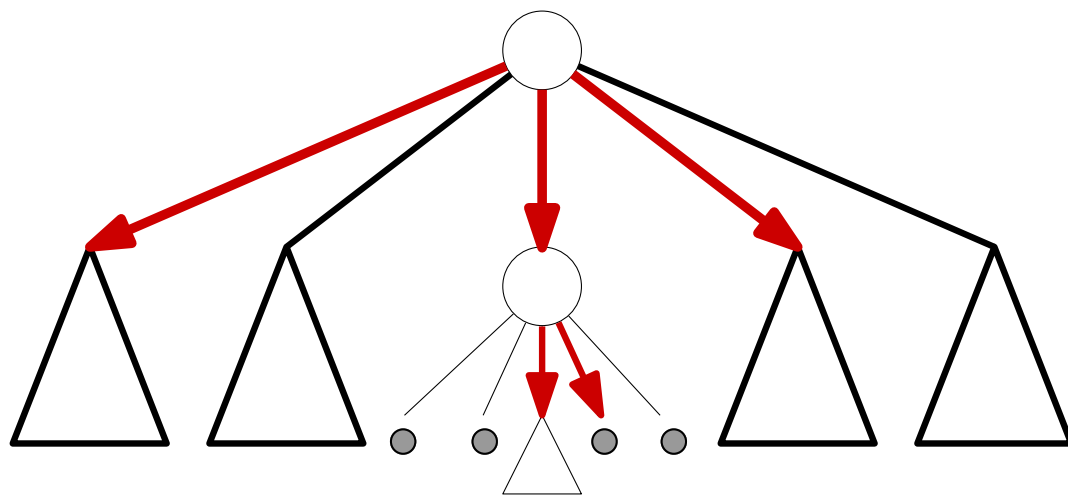
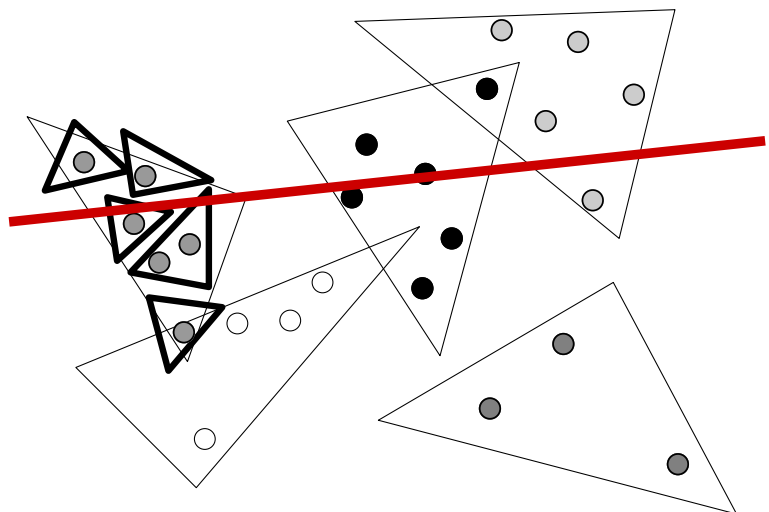
Definition: $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \dots, (S_r, t_r)\}$ is a *simplicial partition* (of size r) for S if

- S is partitioned by S_1, \dots, S_r and
- for $1 \leq i \leq r$, t_i is a triangle and $S_i \subset t_i$.

$\Psi(S)$ is *fine* if $|S_i| \leq 2|S|/r$ for every $1 \leq i \leq r$.

Generalizing to 2 Dimensions

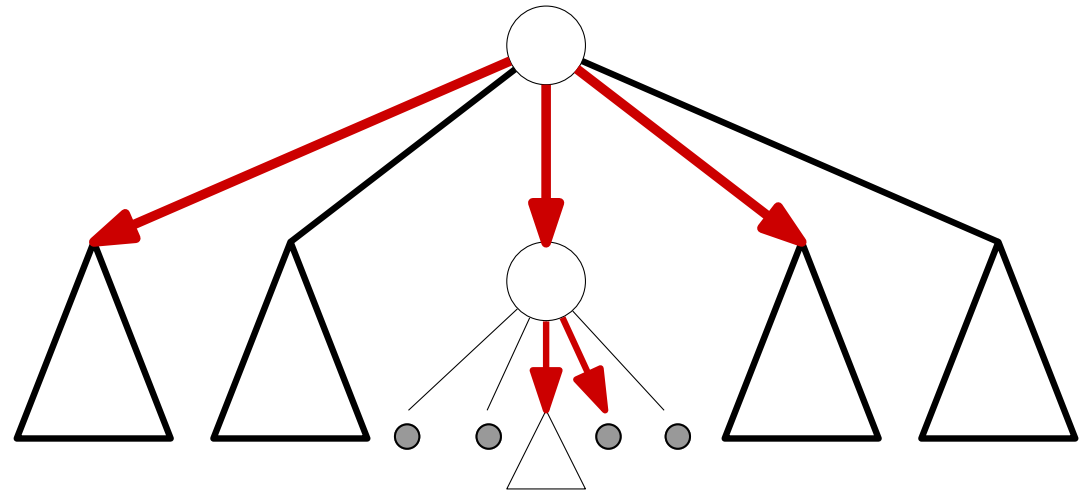
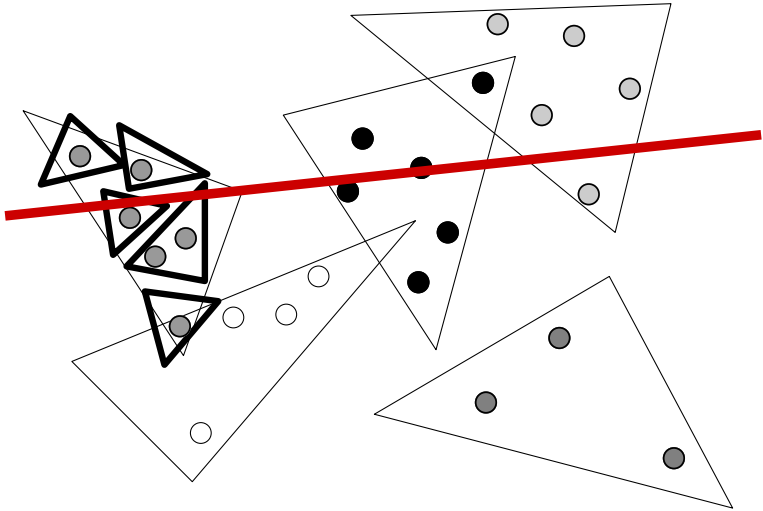
Any ideas?



Definition: The *crossing number* of ℓ (w.r.t. $\Psi(S)$) is the number of triangles t_1, \dots, t_r intersected by ℓ .

Generalizing to 2 Dimensions

Any ideas?

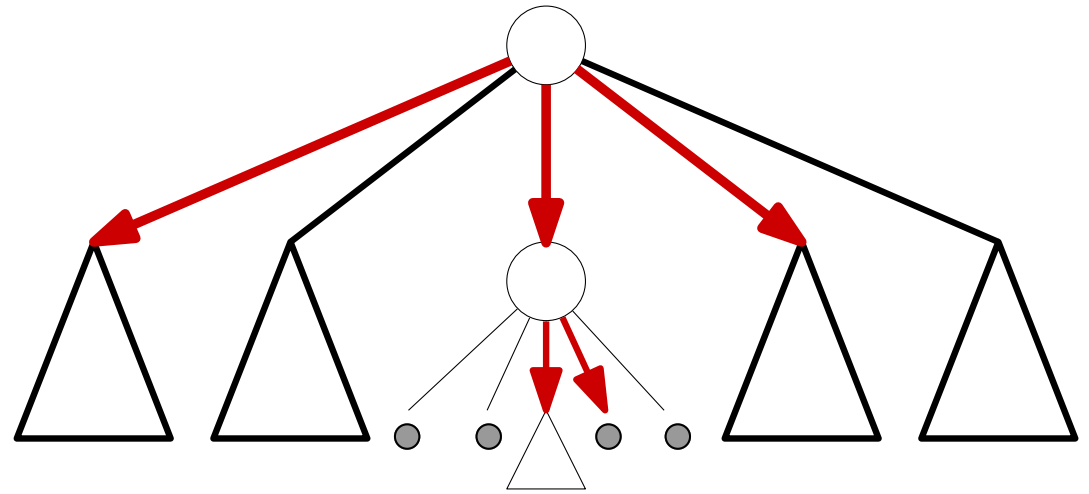
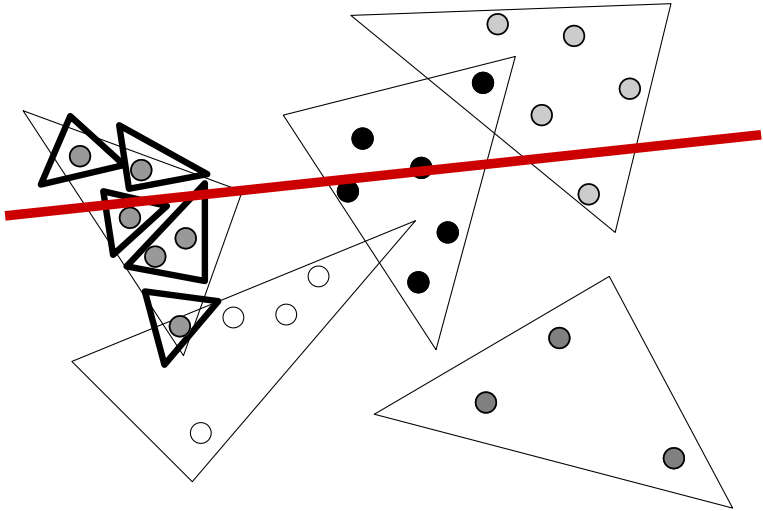


Definition: The *crossing number* of ℓ (w.r.t. $\Psi(S)$) is the number of triangles t_1, \dots, t_r intersected by ℓ .

The *crossing number* of $\Psi(S)$ is the maximum crossing number over all possible lines.

Generalizing to 2 Dimensions

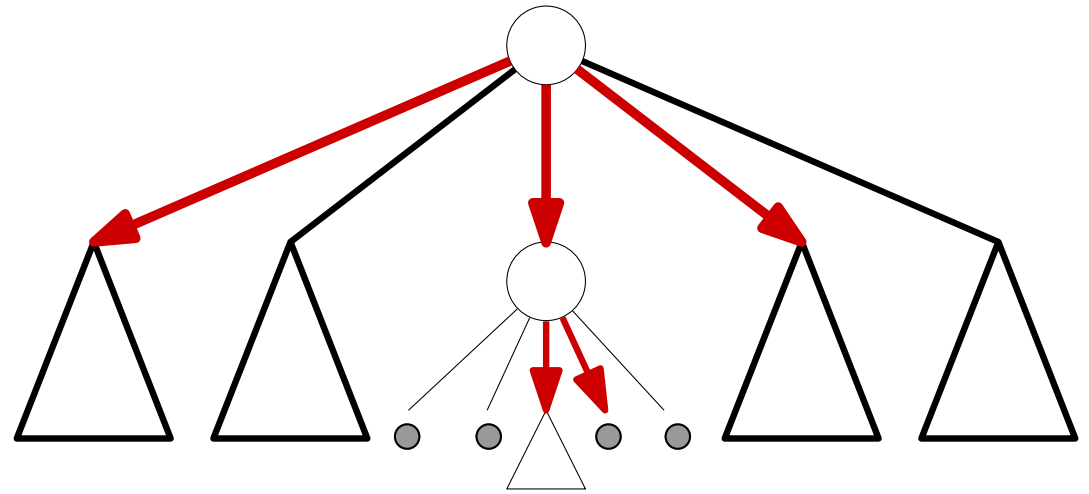
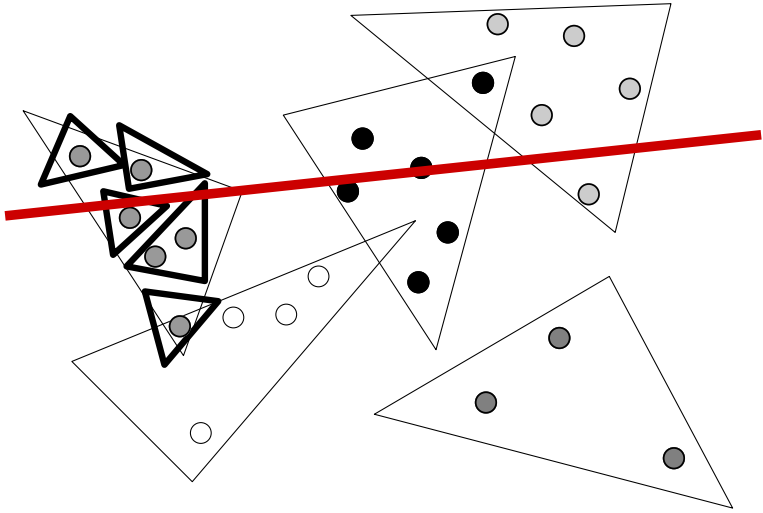
Any ideas?



Theorem: For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\blacksquare)$ exists.

Generalizing to 2 Dimensions

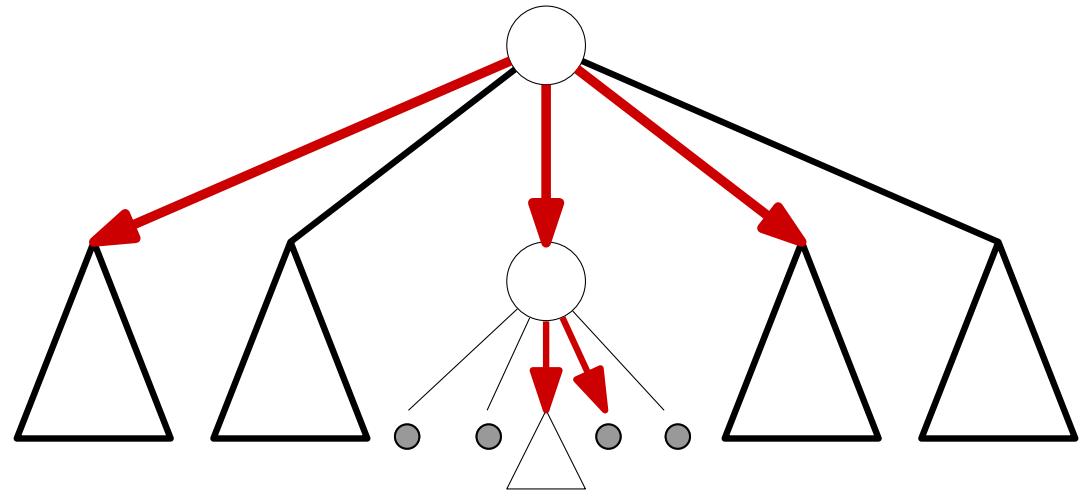
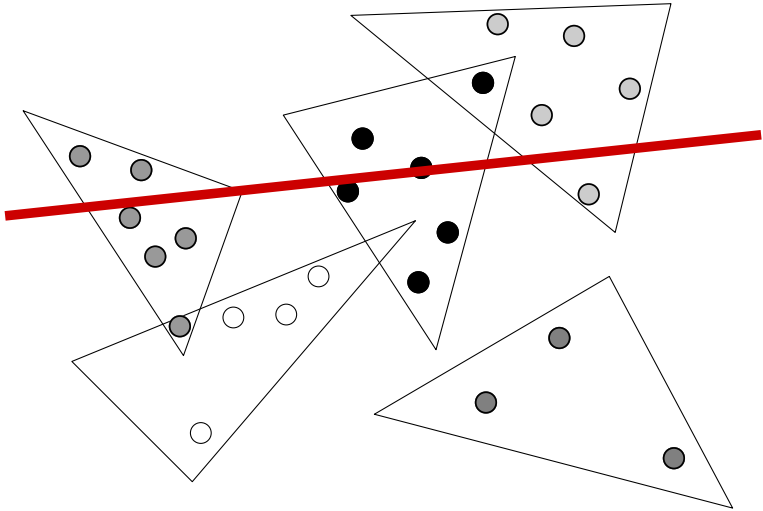
Any ideas?



Theorem: For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists.

Generalizing to 2 Dimensions

Any ideas?

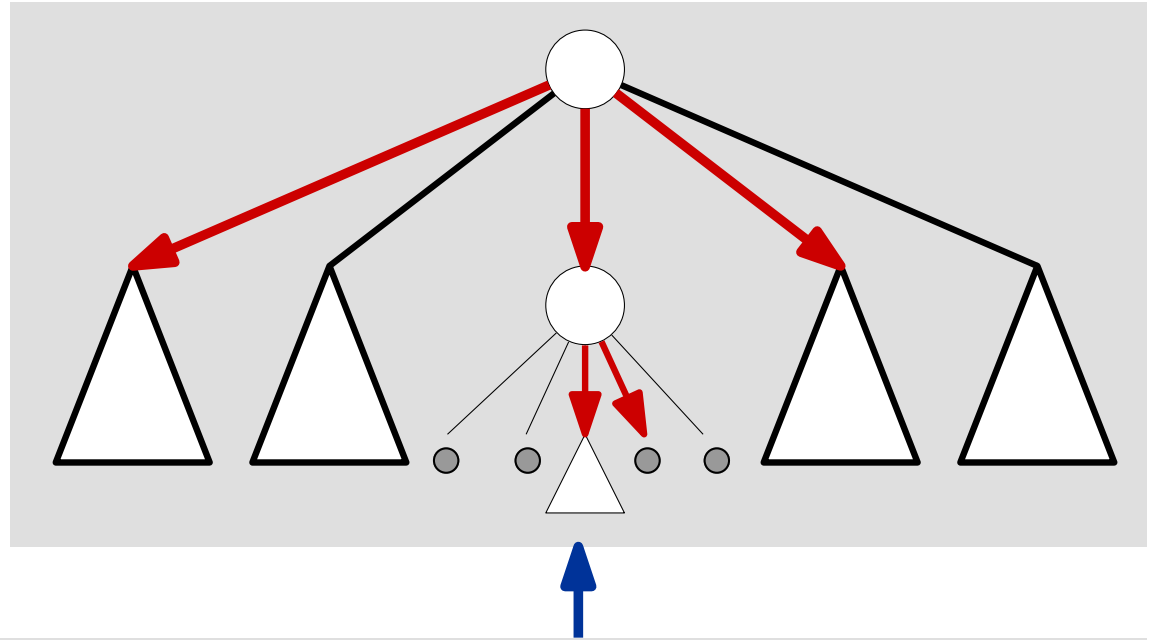
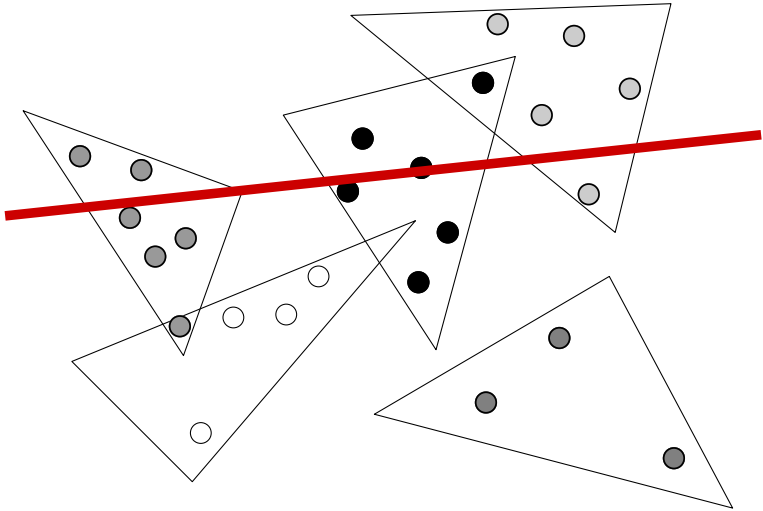


Theorem: For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Generalizing to 2 Dimensions

Any ideas?

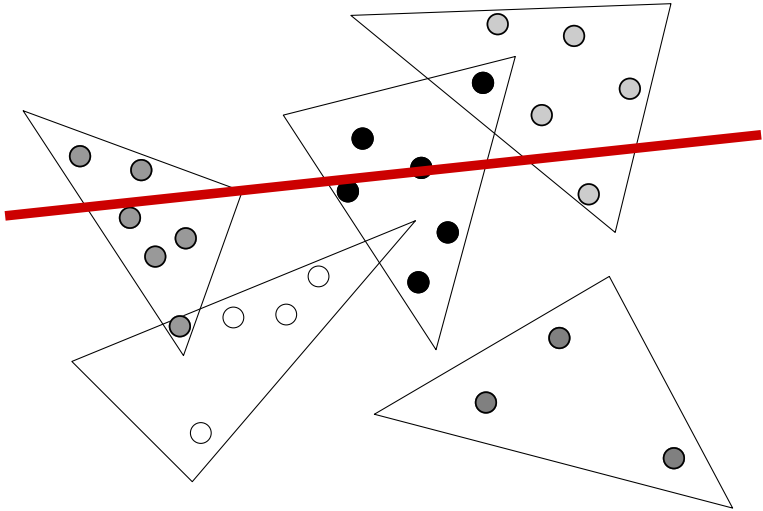


Theorem: For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists.

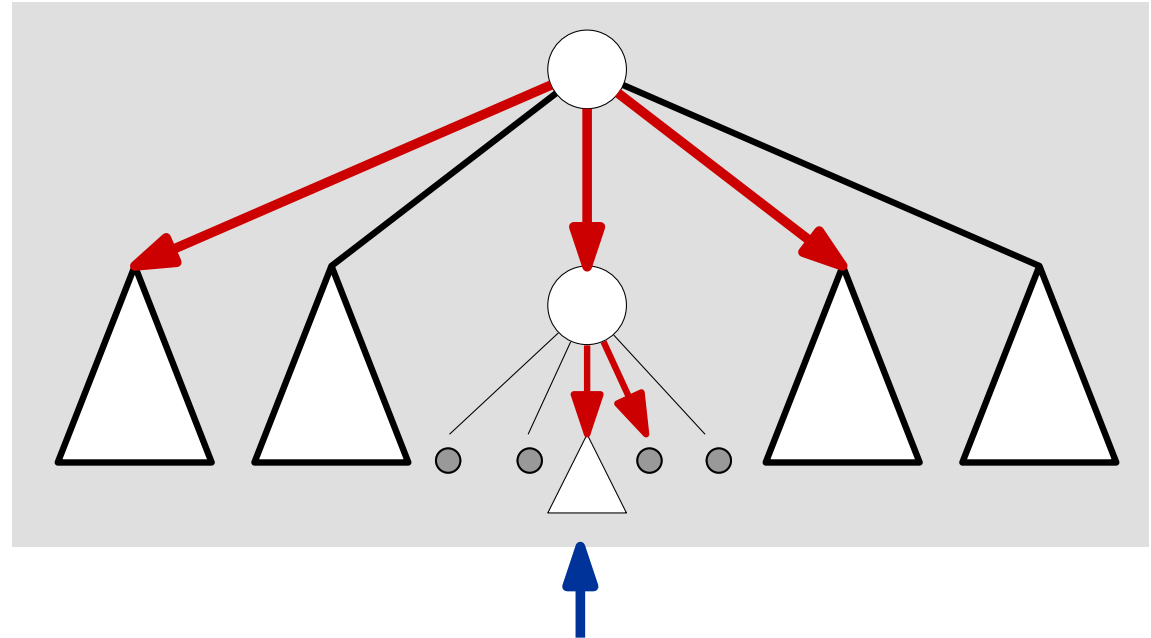
For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Generalizing to 2 Dimensions

Any ideas?



partition tree

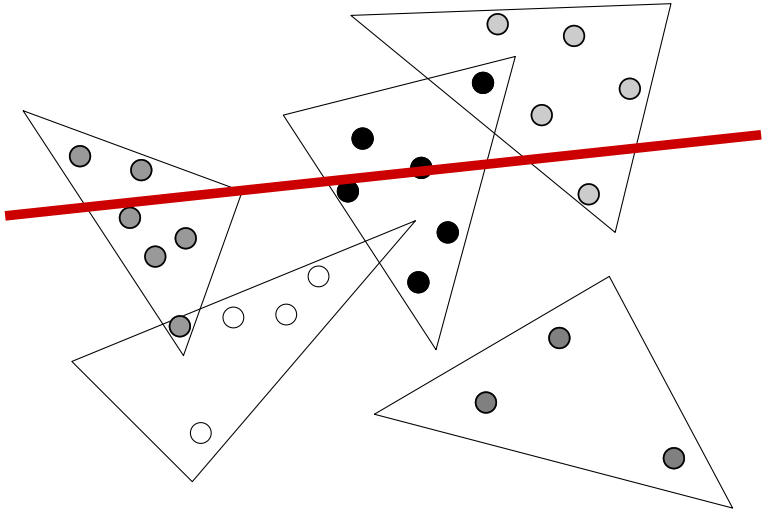


Theorem: For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists.

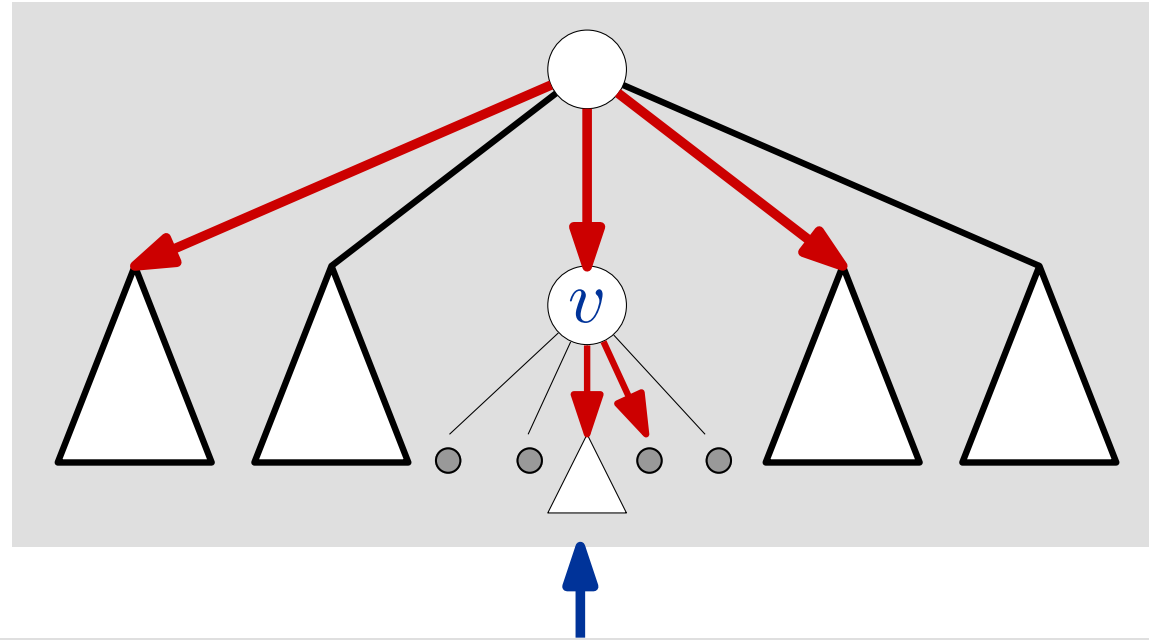
For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Generalizing to 2 Dimensions

Any ideas?



partition tree

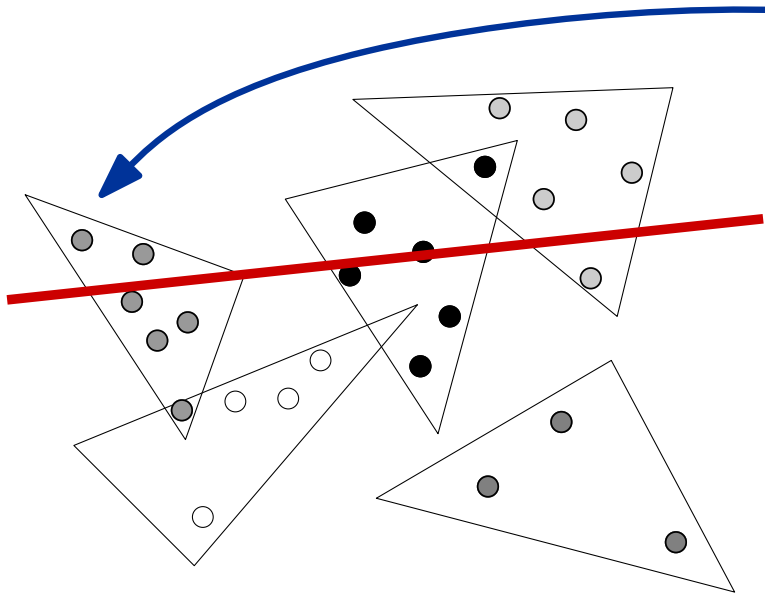


Theorem: For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists.

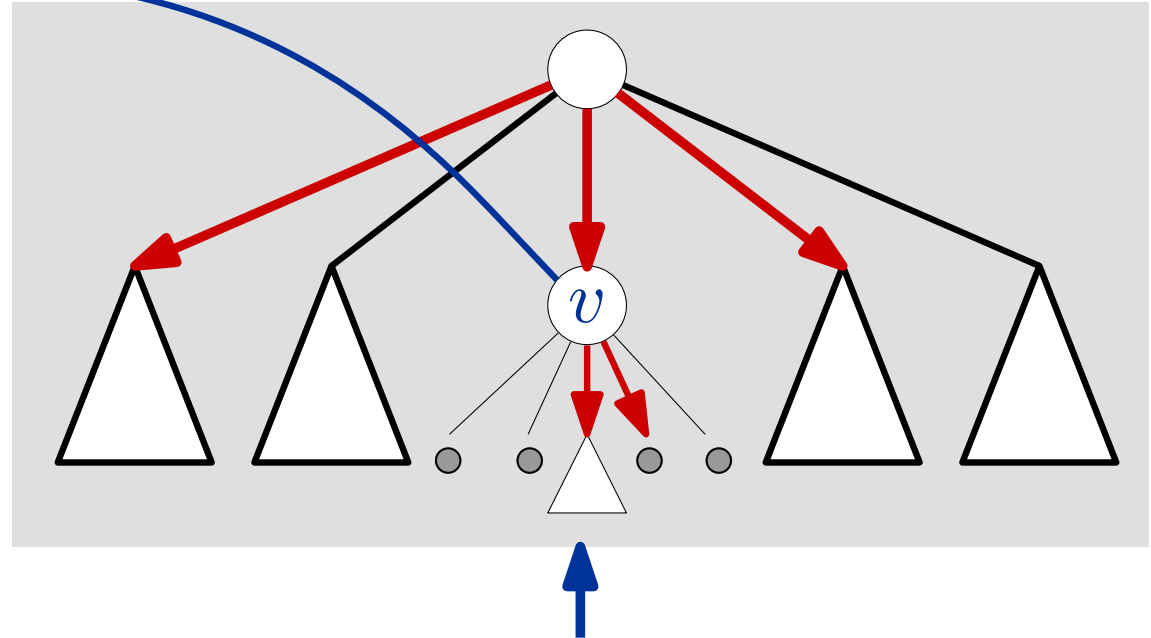
For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Generalizing to 2 Dimensions

Any ideas?



partition tree

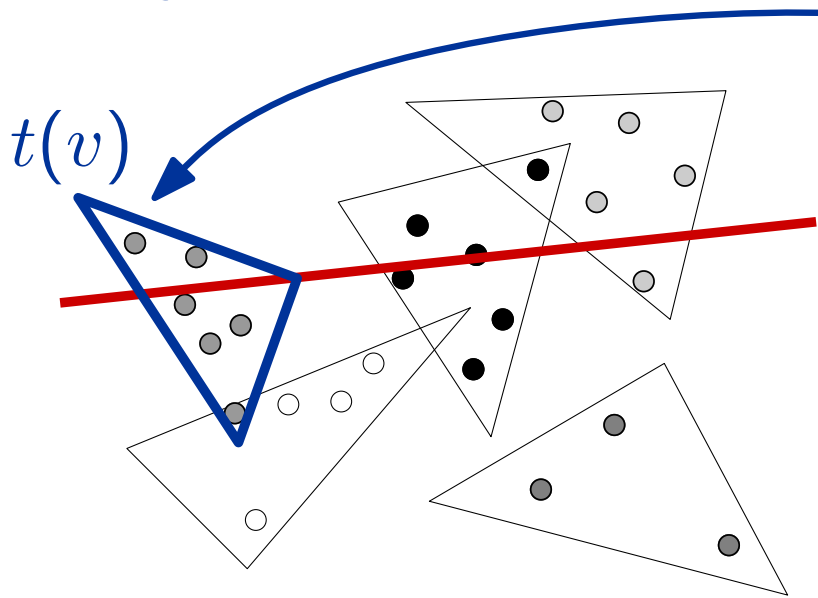


Theorem: For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists.

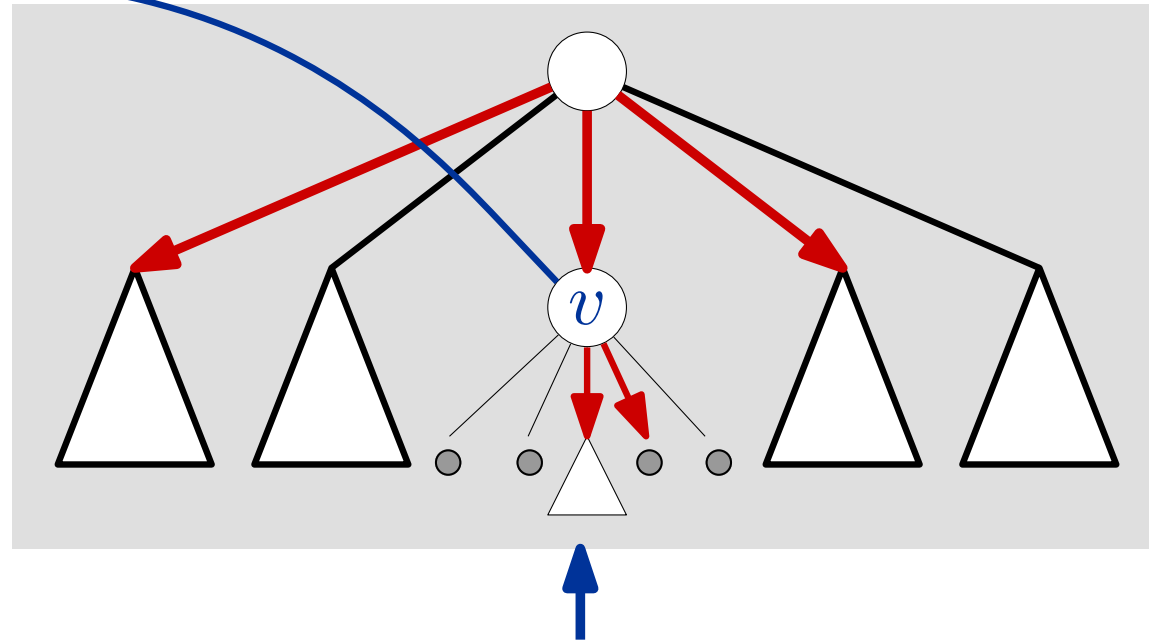
For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Generalizing to 2 Dimensions

Any ideas?



partition tree

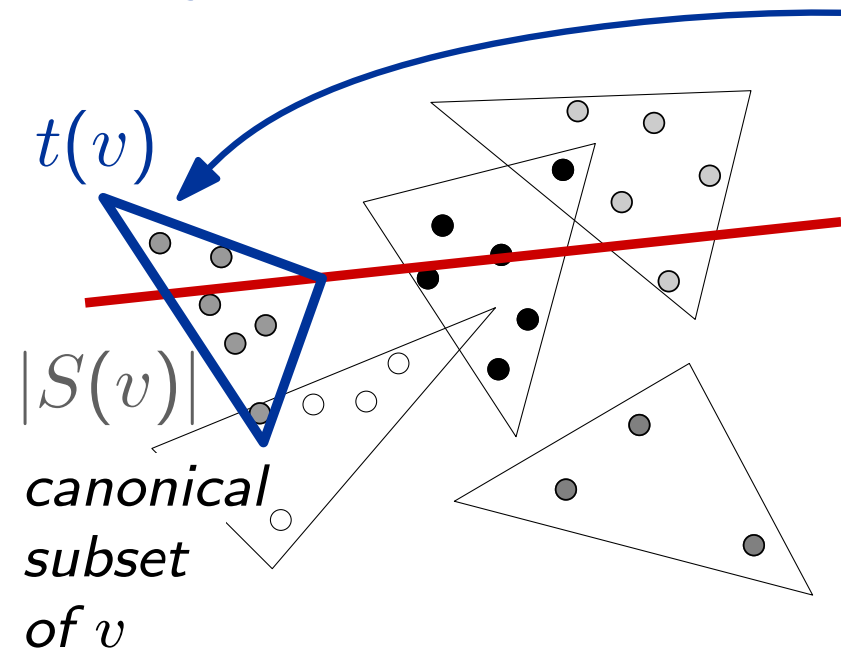


Theorem: For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists.

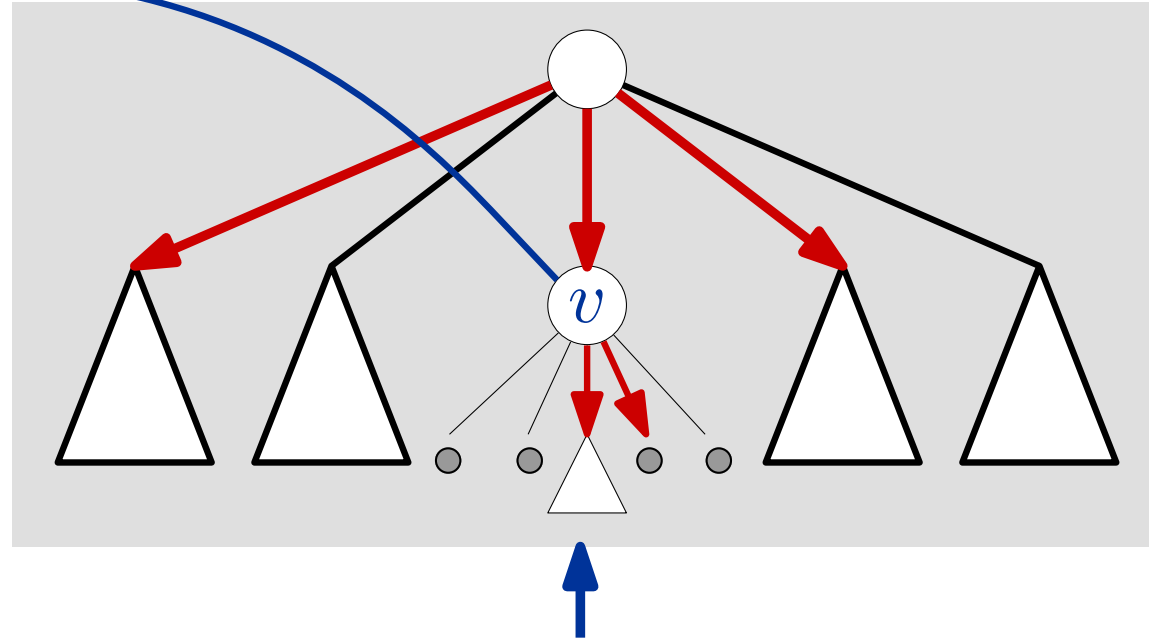
For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Generalizing to 2 Dimensions

Any ideas?



partition tree

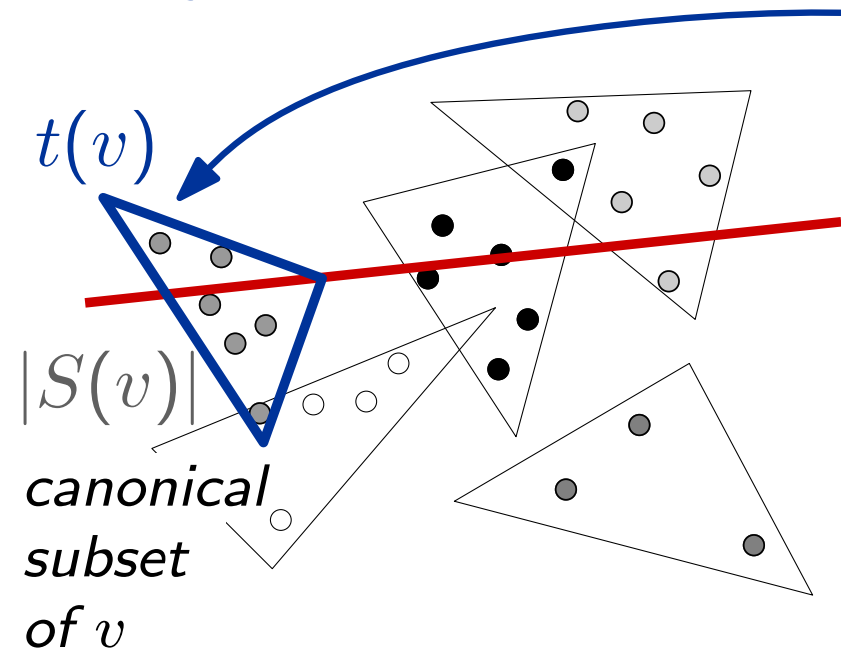


Theorem: For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists.

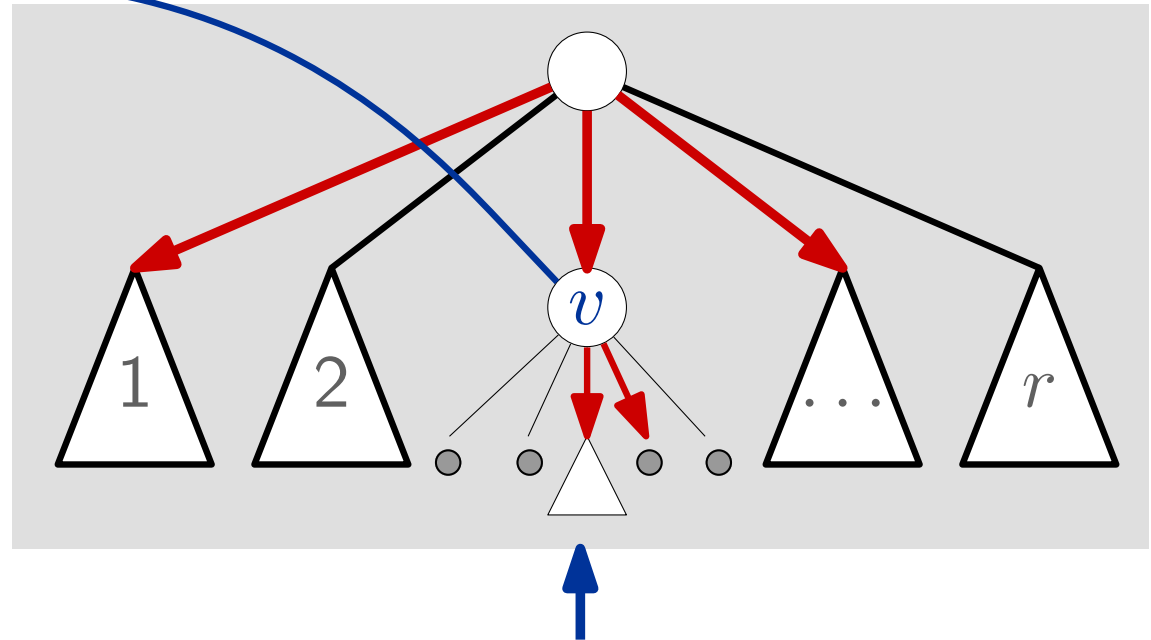
For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Generalizing to 2 Dimensions

Any ideas?



partition tree



Theorem: For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Query Algorithm

SELECTINHALFPLANE(half-plane h , partition tree \mathcal{T})

$N \leftarrow \emptyset$ { set of *selected* nodes }

Query Algorithm

SELECTINHALFPLANE(half-plane h , partition tree \mathcal{T})

$N \leftarrow \emptyset$ { set of *selected* nodes }

if $\mathcal{T} = \{\mu\}$ **then**



else



return N

Query Algorithm

SELECTINHALFPLANE(half-plane h , partition tree \mathcal{T})

$N \leftarrow \emptyset$ { set of *selected* nodes }

if $\mathcal{T} = \{\mu\}$ **then**

if point stored at μ lies in h **then**

$N \leftarrow \{\mu\}$

else

return N

Query Algorithm

SELECTINHALFPLANE(half-plane h , partition tree \mathcal{T})

$N \leftarrow \emptyset$ { set of *selected* nodes }

if $\mathcal{T} = \{\mu\}$ **then**

if point stored at μ lies in h **then**

$N \leftarrow \{\mu\}$

else

for each child ν of the root of \mathcal{T} **do**



return N

Query Algorithm

SELECTINHALFPLANE(half-plane h , partition tree \mathcal{T})

$N \leftarrow \emptyset$ { set of *selected* nodes }

if $\mathcal{T} = \{\mu\}$ **then**

if point stored at μ lies in h **then**

$N \leftarrow \{\mu\}$

else

for each child ν of the root of \mathcal{T} **do**

if $t(\nu) \subset h$ **then**

else

return N

Query Algorithm

SELECTINHALFPLANE(half-plane h , partition tree \mathcal{T})

$N \leftarrow \emptyset$ { set of *selected* nodes }

if $\mathcal{T} = \{\mu\}$ **then**

if point stored at μ lies in h **then**

$N \leftarrow \{\mu\}$

else

for each child ν of the root of \mathcal{T} **do**

if $t(\nu) \subset h$ **then**

$N \leftarrow N \cup \{\nu\}$

else

return N

Query Algorithm

SELECTINHALFPLANE(half-plane h , partition tree \mathcal{T})

$N \leftarrow \emptyset$ { set of *selected* nodes }

if $\mathcal{T} = \{\mu\}$ **then**

if point stored at μ lies in h **then**

$N \leftarrow \{\mu\}$

else

for each child ν of the root of \mathcal{T} **do**

if $t(\nu) \subset h$ **then**

$N \leftarrow N \cup \{\nu\}$

else

if $t(\nu) \cap h \neq \emptyset$ **then**

return N

Query Algorithm

SELECTINHALFPLANE(half-plane h , partition tree \mathcal{T})

$N \leftarrow \emptyset$ { set of *selected* nodes }

if $\mathcal{T} = \{\mu\}$ **then**

if point stored at μ lies in h **then**

$N \leftarrow \{\mu\}$

else

for each child ν of the root of \mathcal{T} **do**

if $t(\nu) \subset h$ **then**

$N \leftarrow N \cup \{\nu\}$

else

if $t(\nu) \cap h \neq \emptyset$ **then**

$N \leftarrow N \cup \text{SELECTINHALFPLANE}(h, \mathcal{T}_\nu)$

return N

Query Algorithm

SELECTINHALFPLANE(half-plane h , partition tree \mathcal{T})

$N \leftarrow \emptyset$ { set of *selected* nodes }

if $\mathcal{T} = \{\mu\}$ **then**

if point stored at μ lies in h **then**

$N \leftarrow \{\mu\}$

else

for each child ν of the root of \mathcal{T} **do**

if $t(\nu) \subset h$ **then**

$N \leftarrow N \cup \{\nu\}$

else

if $t(\nu) \cap h \neq \emptyset$ **then**

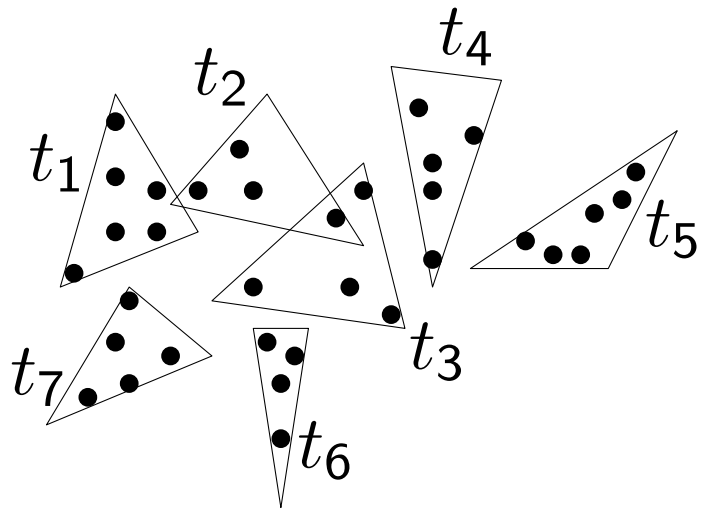
$N \leftarrow N \cup \text{SELECTINHALFPLANE}(h, \mathcal{T}_\nu)$

return N

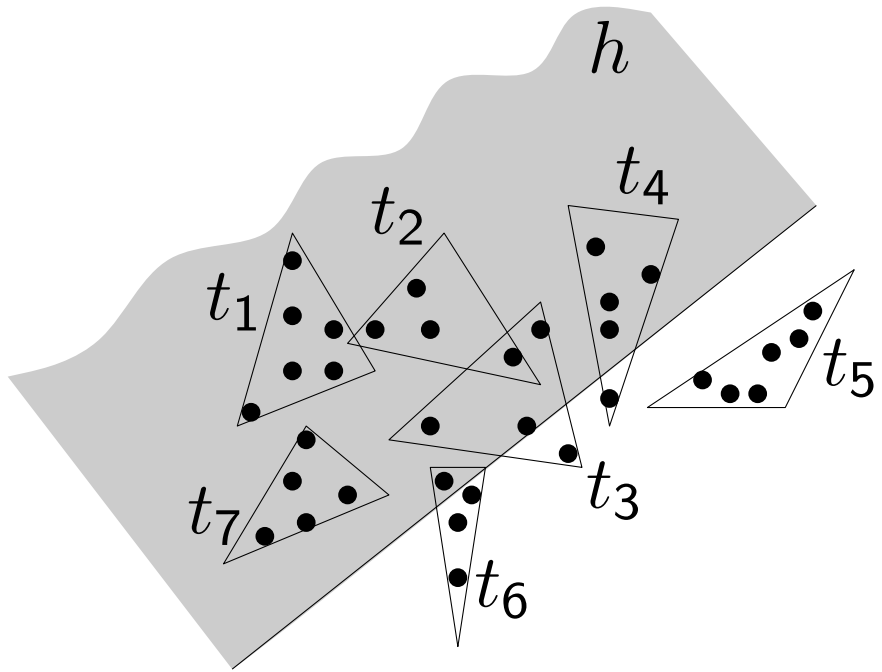
Task:

Turn this into a
range counting
query algorithm!

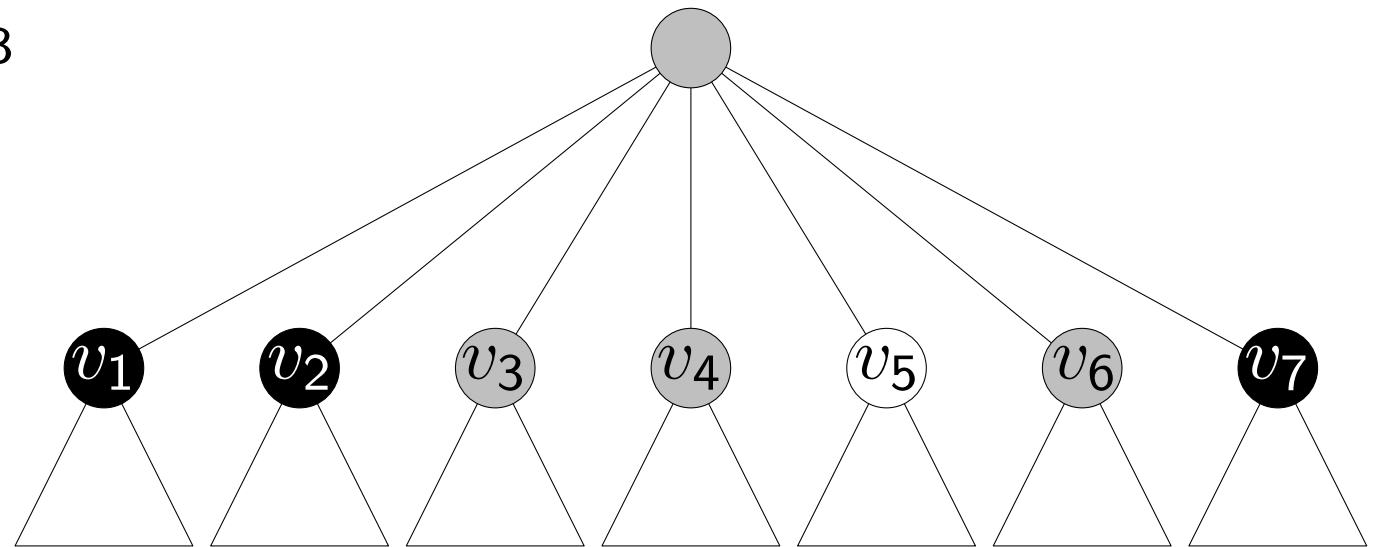
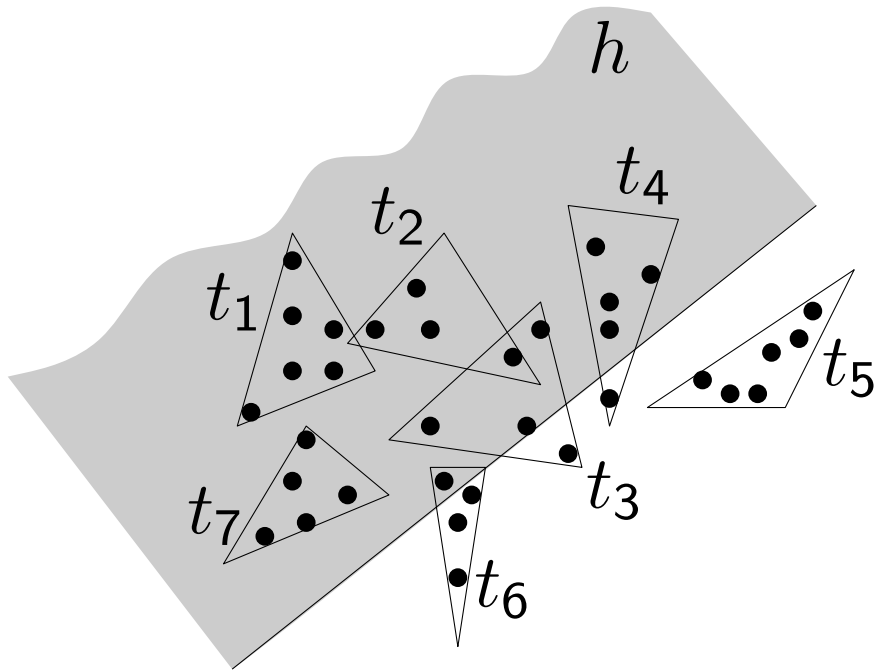
Example



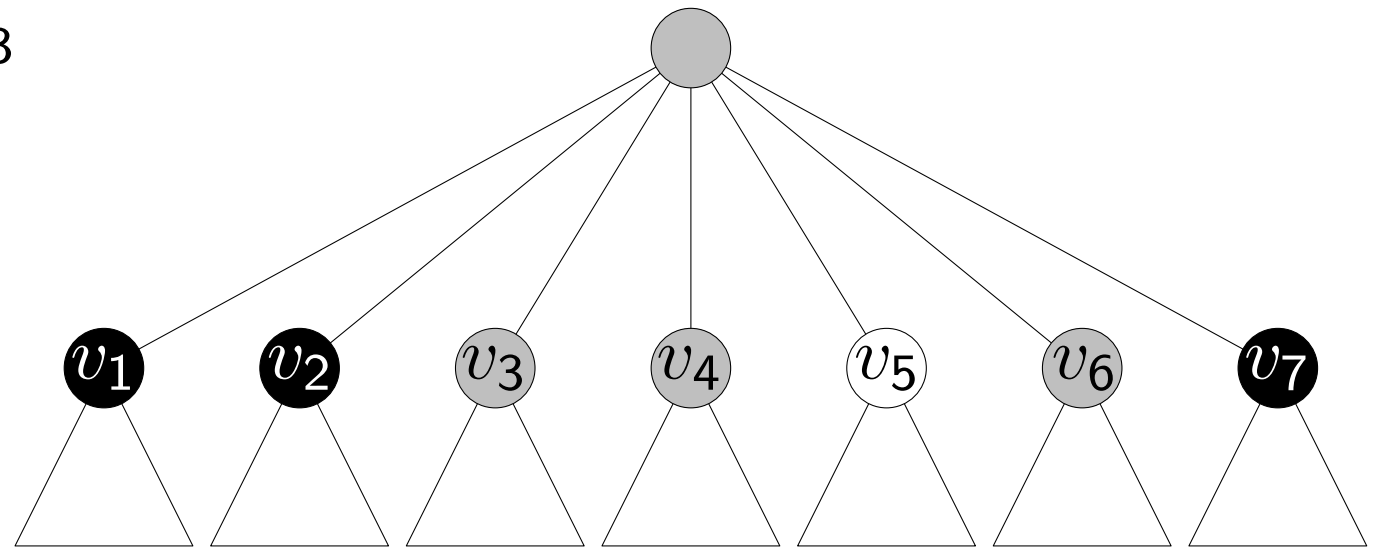
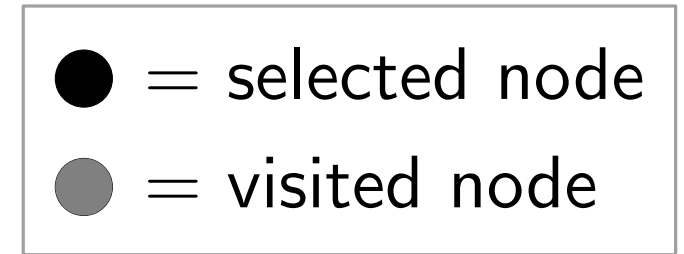
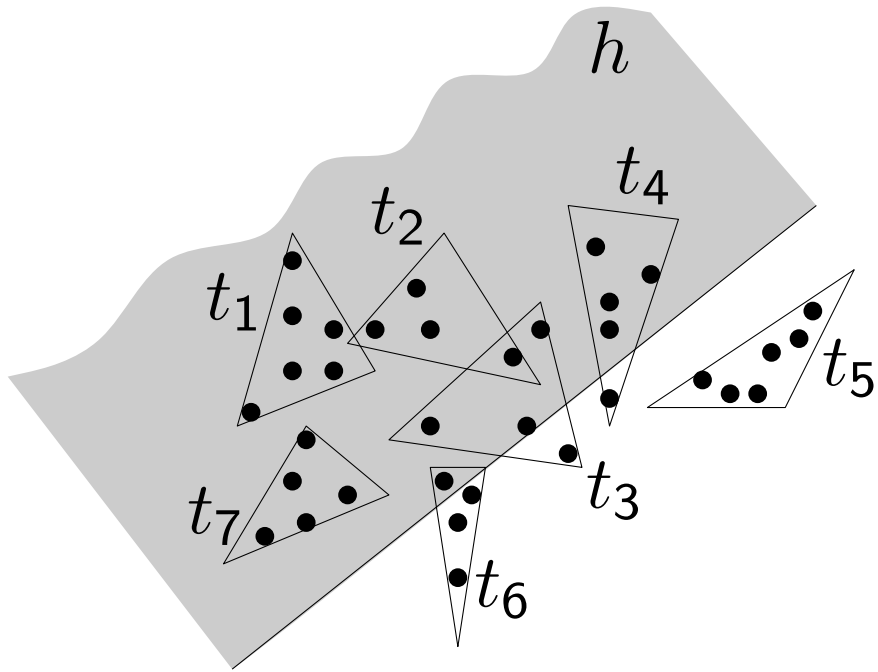
Example



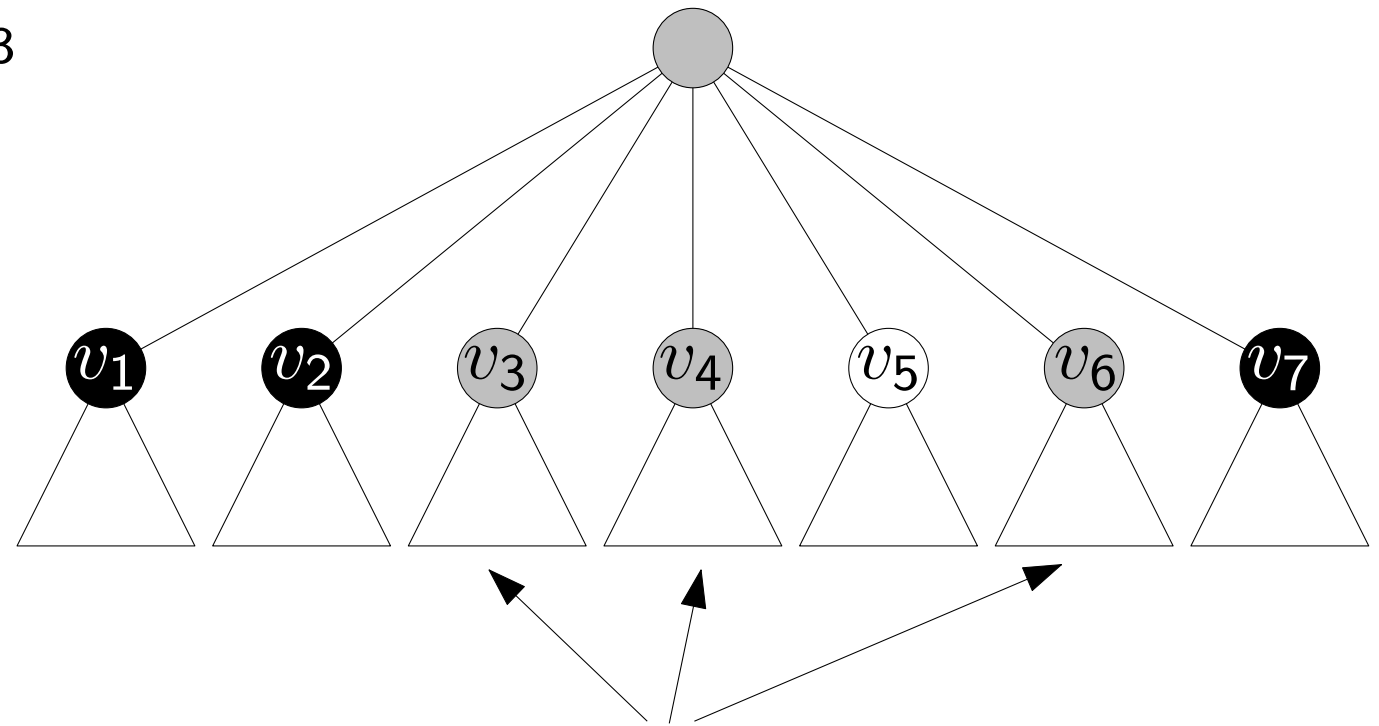
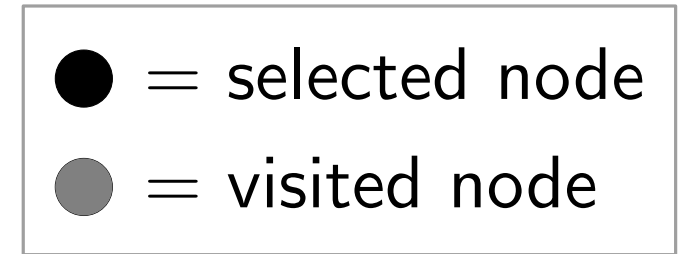
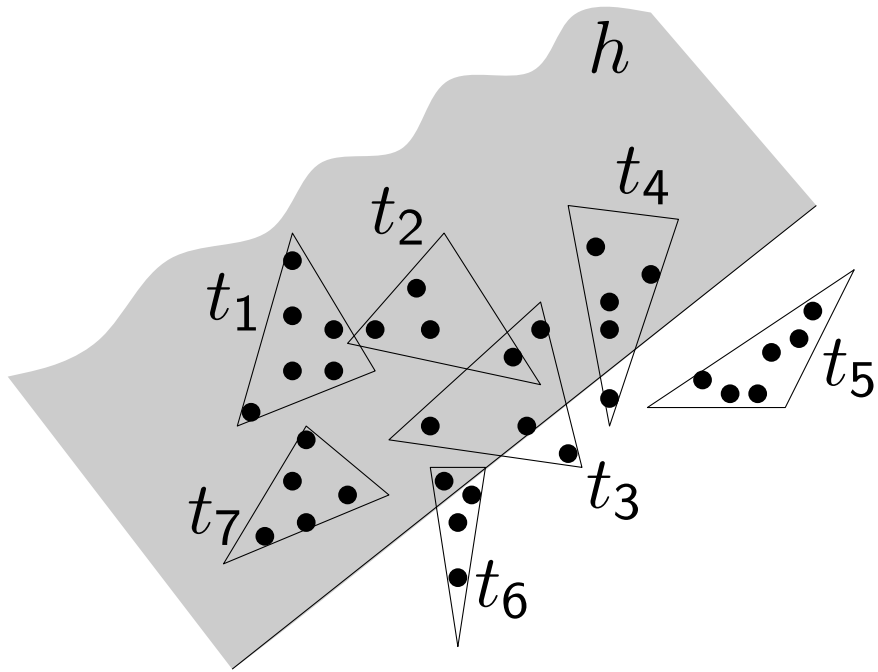
Example



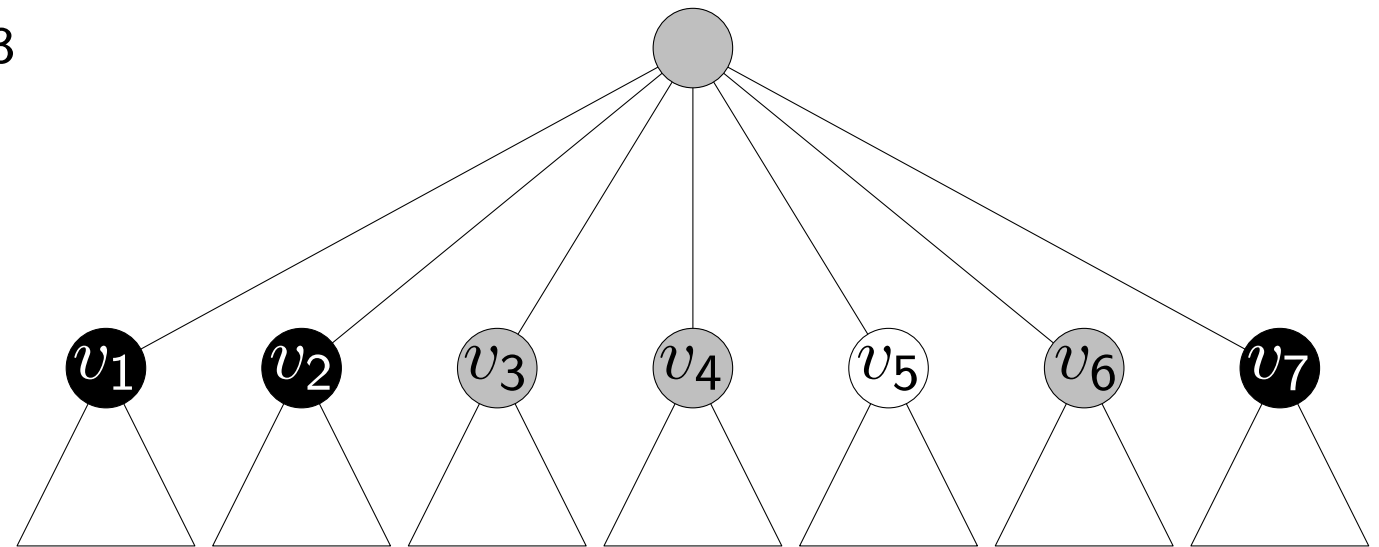
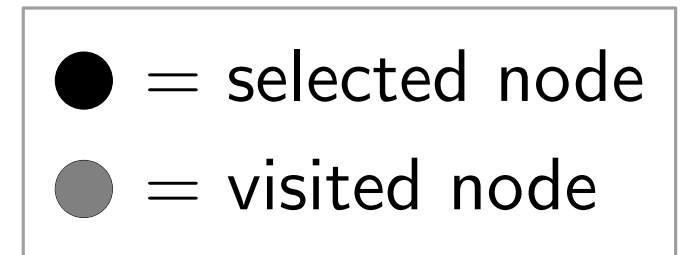
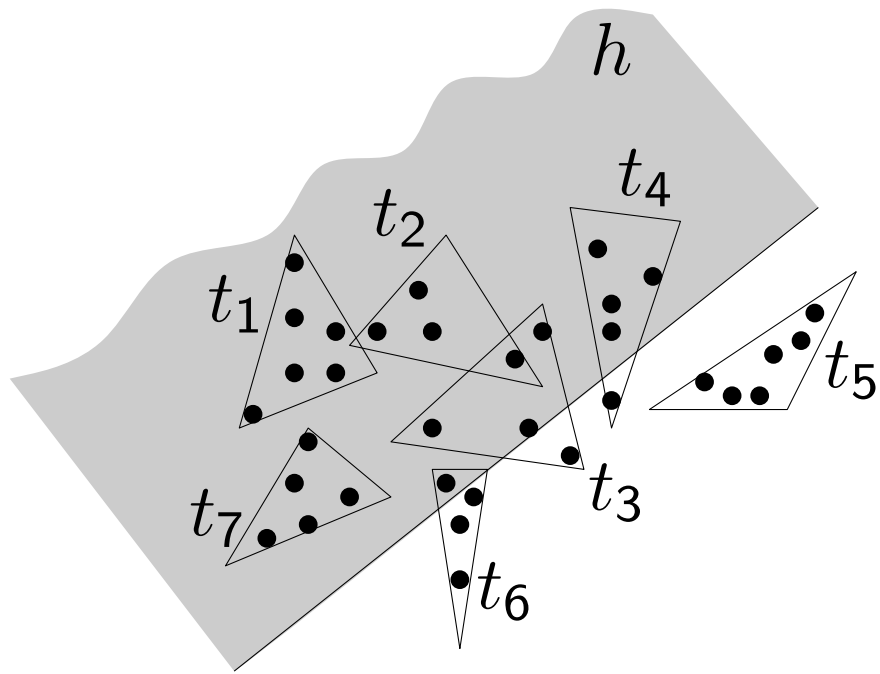
Example



Example



Example



recursively visited subtrees

Analysis of the Partition Tree

Let S be a set of n points in the plane.

Lemma: A partition tree for S uses $O(n)$ storage.

Analysis of the Partition Tree

Let S be a set of n points in the plane.

Lemma: A partition tree for S uses $O(n)$ storage.

Lemma: For any $\varepsilon > 0$, there is a partition tree \mathcal{T} for S s.t.:

Analysis of the Partition Tree

Let S be a set of n points in the plane.

Lemma: A partition tree for S uses $O(n)$ storage.

Lemma: For any $\varepsilon > 0$, there is a partition tree \mathcal{T} for S s.t.:
for a query half-plane h , SELECTINHALFPLANE
selects in $O(n^{1/2+\varepsilon})$ time
a set N of $O(n^{1/2+\varepsilon})$ nodes of \mathcal{T}

Analysis of the Partition Tree

Let S be a set of n points in the plane.

Lemma: A partition tree for S uses $O(n)$ storage.

Lemma: For any $\varepsilon > 0$, there is a partition tree \mathcal{T} for S s.t.:

- for a query half-plane h , `SELECTINHALFPLANE` selects in $O(n^{1/2+\varepsilon})$ time
- a set N of $O(n^{1/2+\varepsilon})$ nodes of \mathcal{T}
- with the property that $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

Analysis of the Partition Tree

Let S be a set of n points in the plane.

Lemma: A partition tree for S uses $O(n)$ storage.

Lemma: For any $\varepsilon > 0$, there is a partition tree \mathcal{T} for S s.t.:

- for a query half-plane h , `SELECTINHALFPLANE` selects in $O(n^{1/2+\varepsilon})$ time
- a set N of $O(n^{1/2+\varepsilon})$ nodes of \mathcal{T}
- with the property that $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

Corollary: Half-plane range counting queries can be answered in $O(n^{1/2+\varepsilon})$ time.

Back to Triangular Range Queries

Any ideas?

Back to Triangular Range Queries

Any ideas?

Theorem: Given a set S of n pts in the plane, for any $\varepsilon > 0$, a triangular range-counting query can be answered in $O(n^{1/2+\varepsilon})$ time using a partition tree.

Back to Triangular Range Queries

Any ideas?

Theorem: Given a set S of n pts in the plane, for any $\varepsilon > 0$, a triangular range-counting query can be answered in $O(n^{1/2+\varepsilon})$ time using a partition tree.

The tree can be built in $O(n^{1+\varepsilon})$ time and uses $O(n)$ space.

Back to Triangular Range Queries

Any ideas?

Theorem: Given a set S of n pts in the plane, for any $\varepsilon > 0$, a triangular range-counting query can be answered in $O(n^{1/2+\varepsilon})$ time using a partition tree.

The tree can be built in $O(n^{1+\varepsilon})$ time and uses $O(n)$ space.

The points inside the query range can be reported in $O(k)$ additional time, where k is the number of reported pts.

Back to Triangular Range Queries

Any ideas?

Theorem: Given a set S of n pts in the plane, for any $\varepsilon > 0$, a triangular range-counting query can be answered in $O(n^{1/2+\varepsilon})$ time using a partition tree.

The tree can be built in $O(n^{1+\varepsilon})$ time and uses $O(n)$ space.

The points inside the query range can be reported in $O(k)$ additional time, where k is the number of reported pts.

Can we do better?

Back to Triangular Range Queries

Any ideas?

Theorem: Given a set S of n pts in the plane, for any $\varepsilon > 0$, a triangular range-counting query can be answered in $O(n^{1/2+\varepsilon})$ time using a partition tree.

The tree can be built in $O(n^{1+\varepsilon})$ time and uses $O(n)$ space.

The points inside the query range can be reported in $O(k)$ additional time, where k is the number of reported pts.

Can we do better?

Use cutting trees! (Chapter 16.3)

Back to Triangular Range Queries

Any ideas?

Theorem: Given a set S of n pts in the plane, for any $\varepsilon > 0$, a triangular range-counting query can be answered in $O(n^{1/2+\varepsilon})$ time using a partition tree.

The tree can be built in $O(n^{1+\varepsilon})$ time and uses $O(n)$ space.

The points inside the query range can be reported in $O(k)$ additional time, where k is the number of reported pts.

Can we do better?

Use cutting trees! (Chapter 16.3)

Query time $O(\log^3 n)$, prep. & storage $O(n^{2+\varepsilon})$.

Multi-Level Partition Trees

Idea: Store with each internal node not just a number,

Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number,

Multi-Level Partition Trees

$|S(v)|$

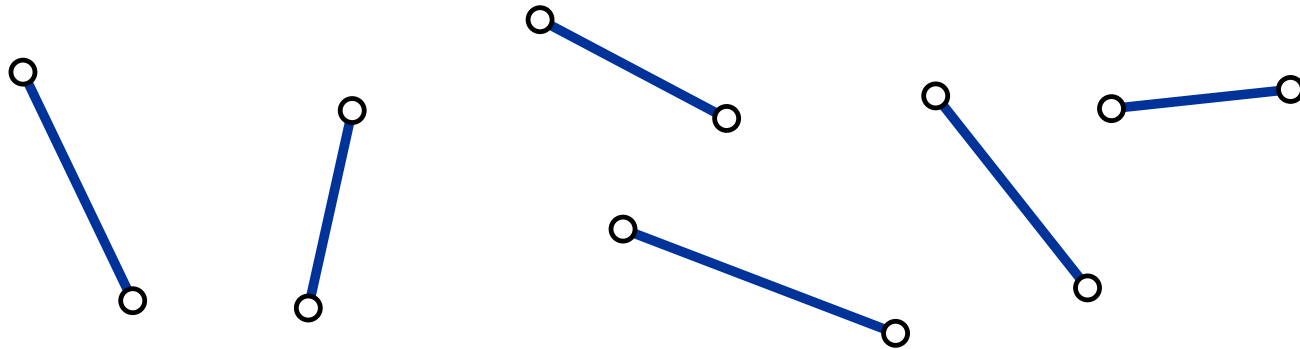
Idea: Store with each internal node not just a number, but another data structure!

Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

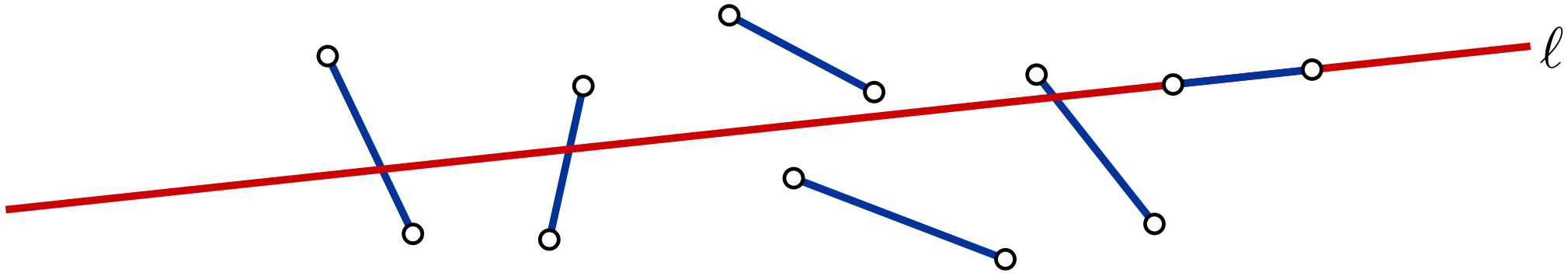


Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .



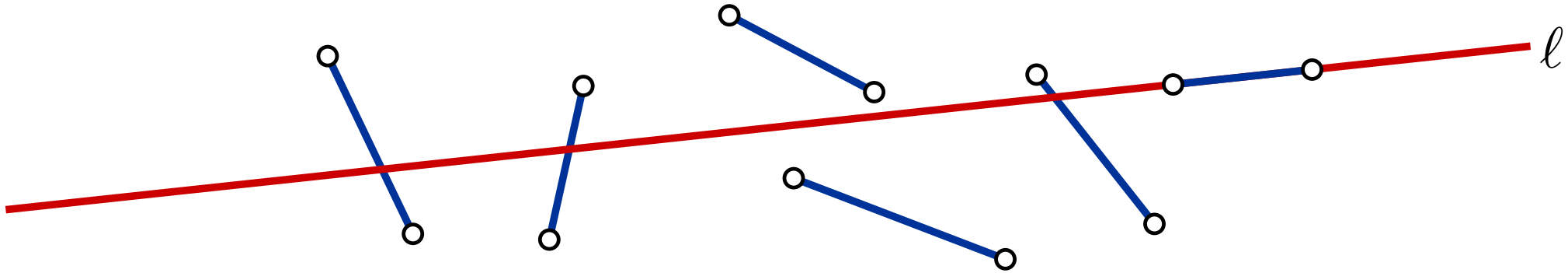
Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

Hint:



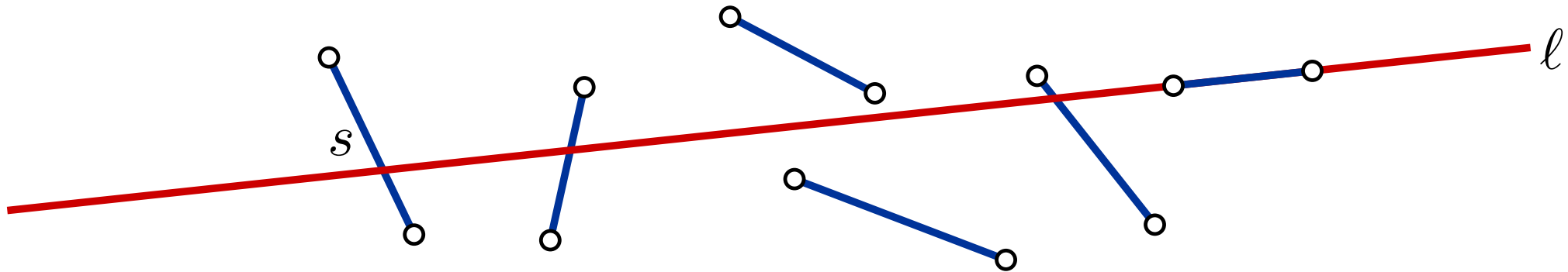
Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

Hint:



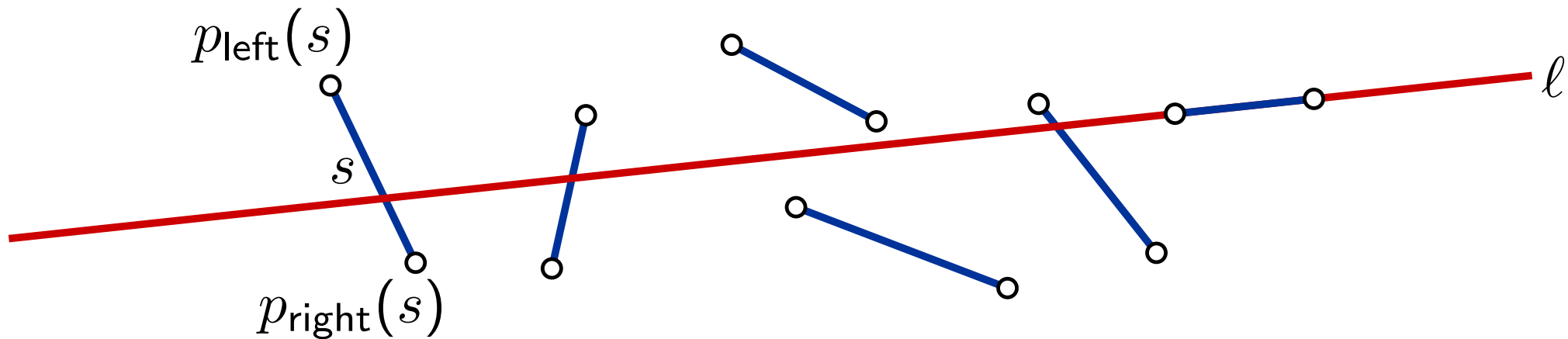
Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

Hint:



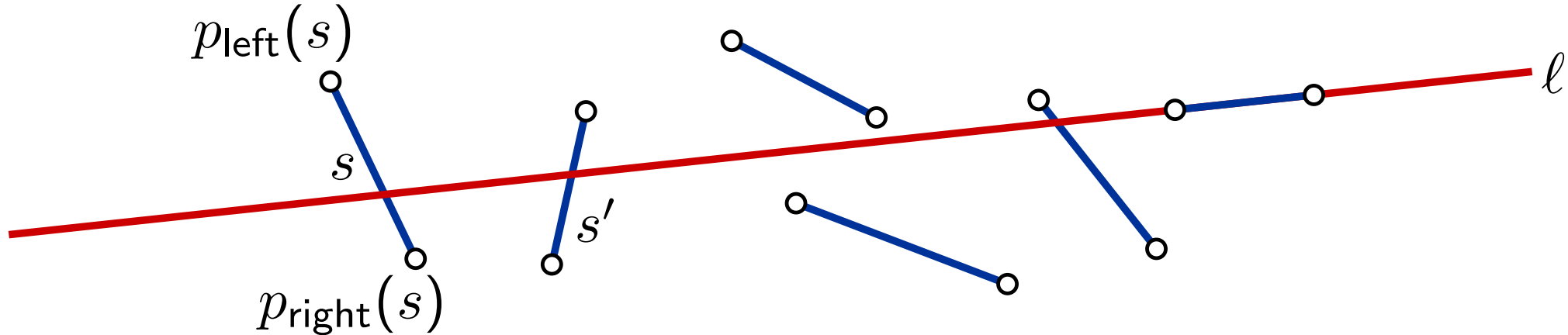
Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

Hint:



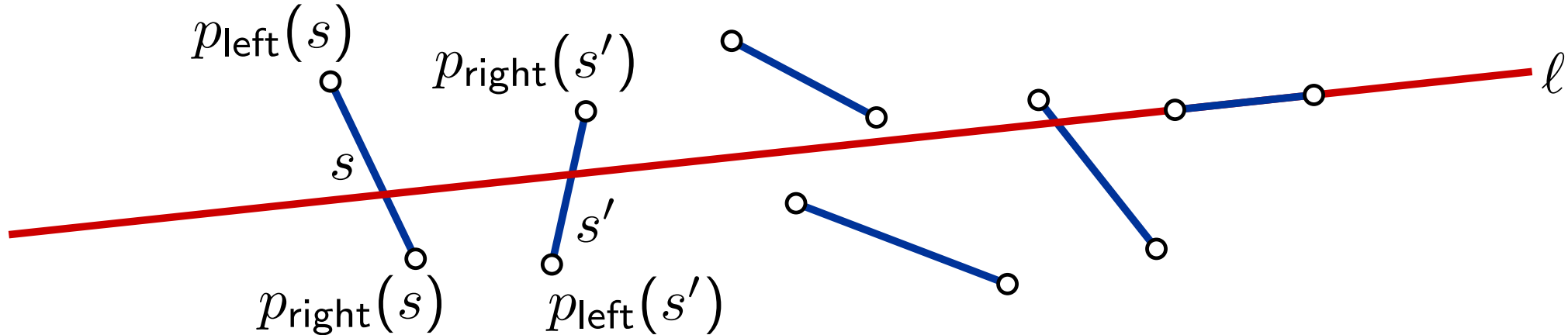
Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

Hint:



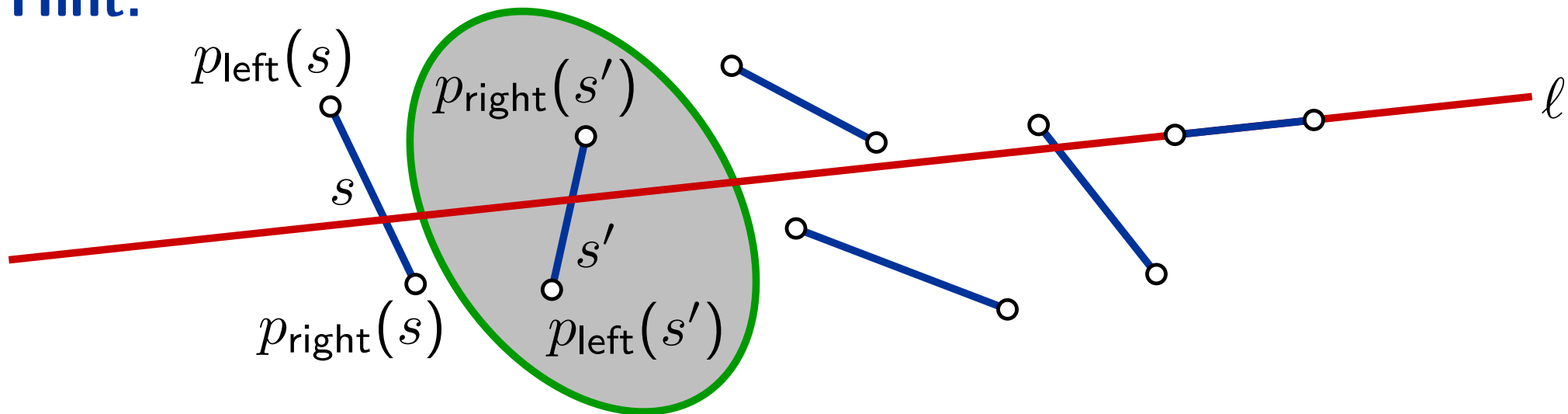
Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

Hint:



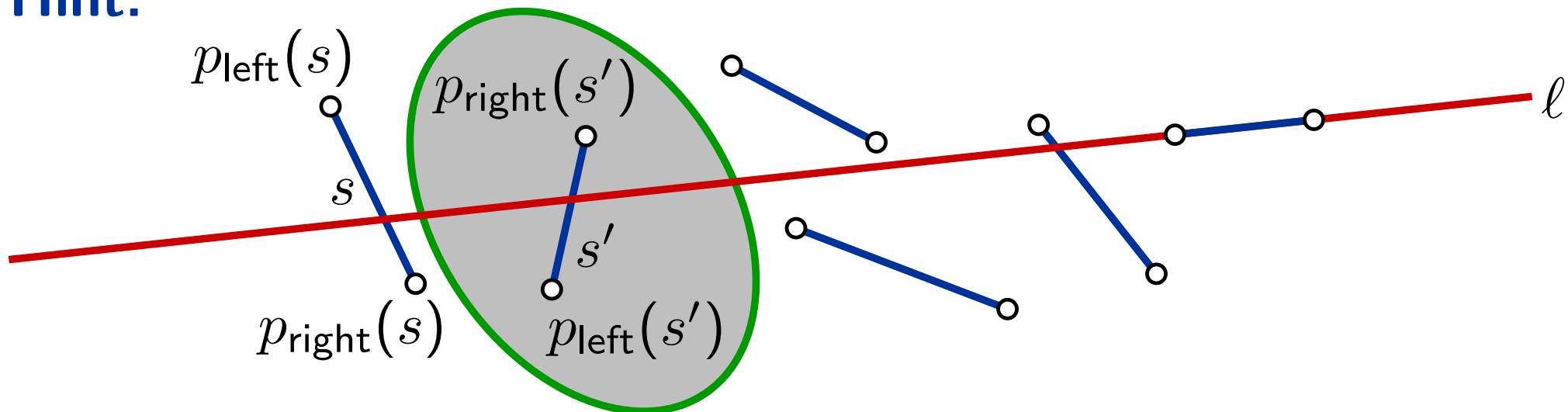
Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

Hint:



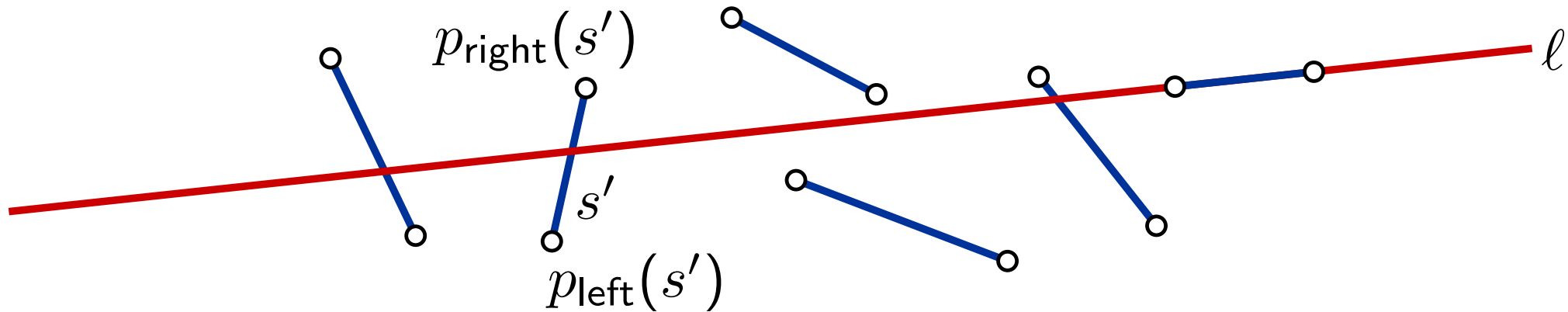
[3 min]

Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

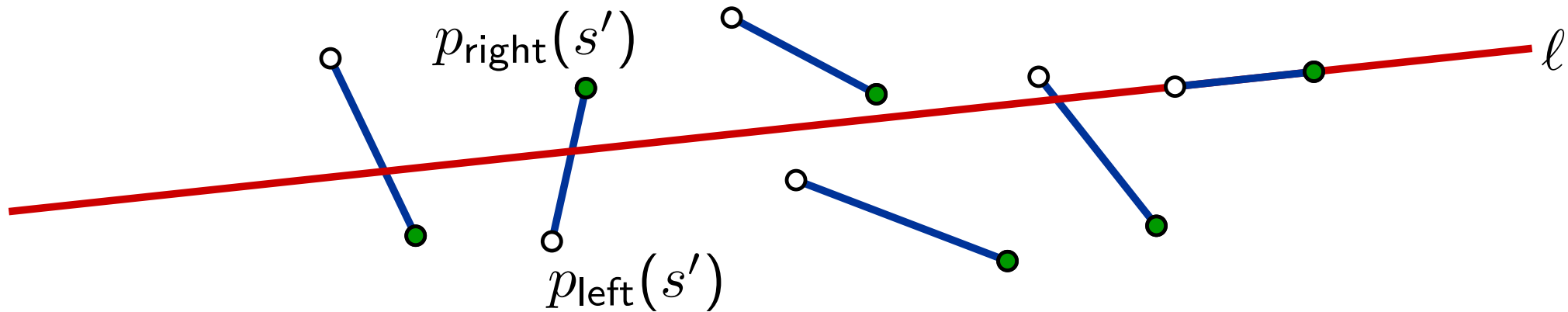


Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

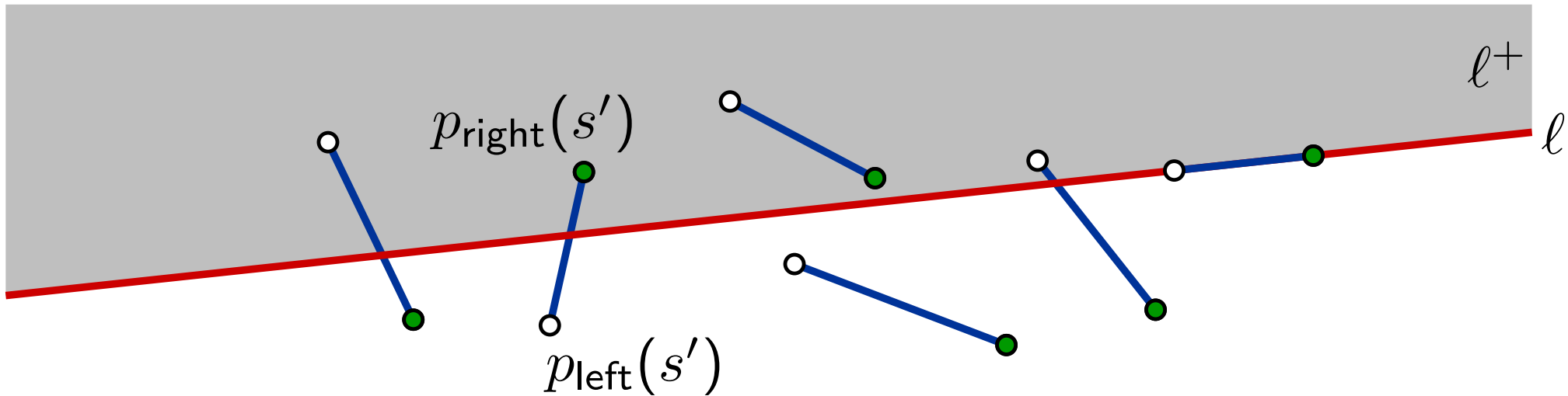


Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

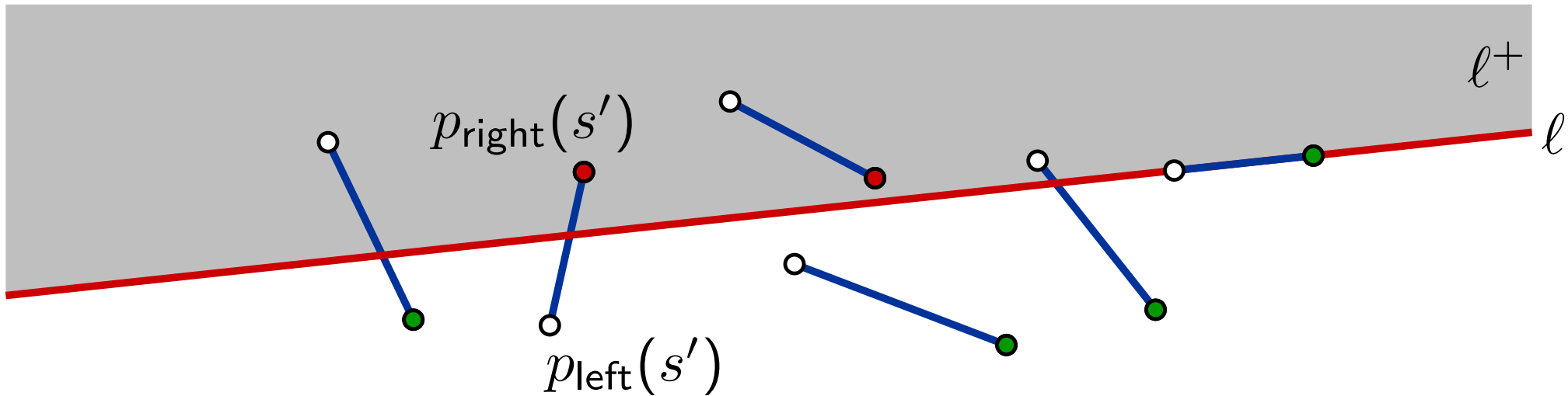


Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

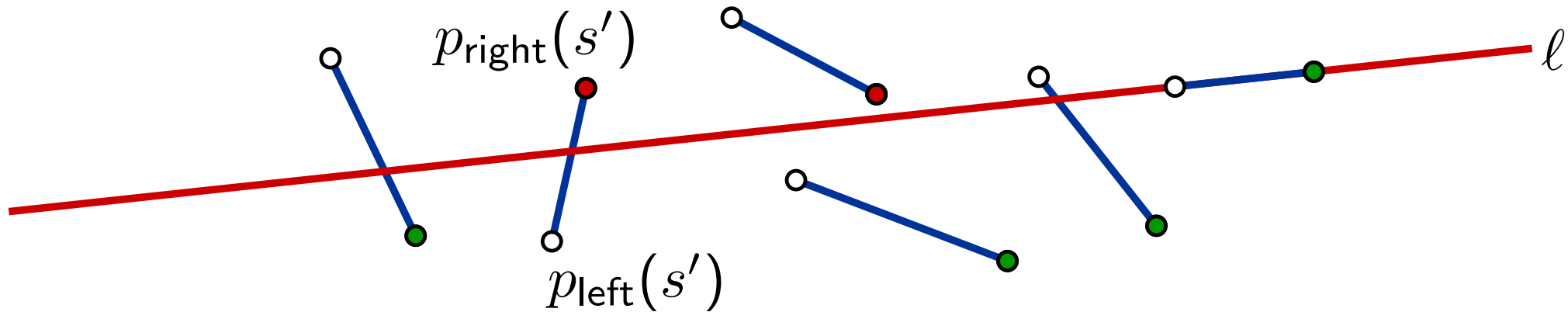


Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

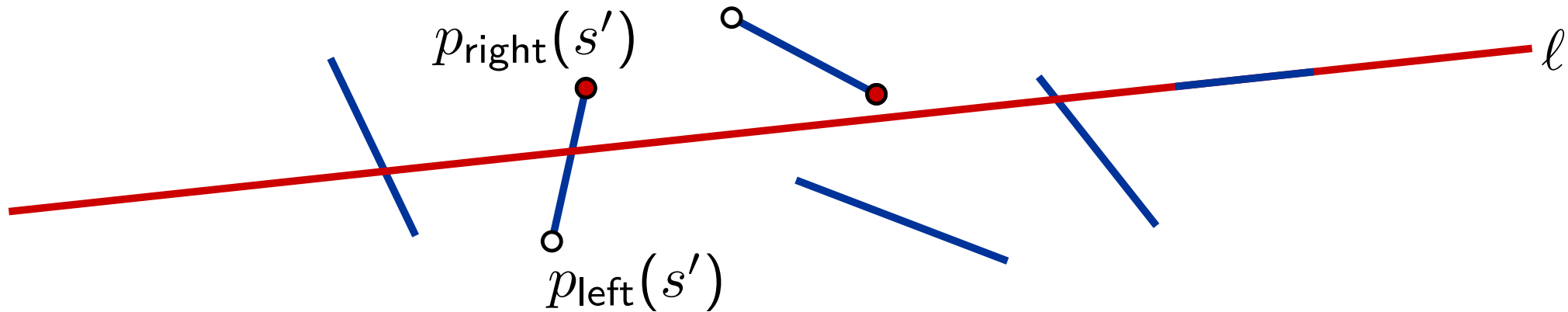


Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .

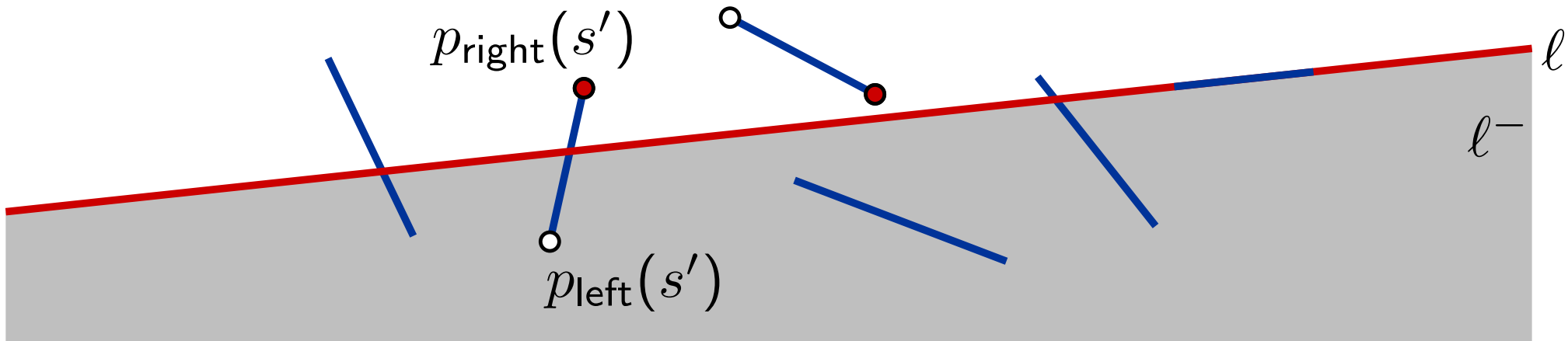


Multi-Level Partition Trees

$|S(v)|$

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line ℓ .



Results

Lemma: A 2-level partition tree for line-intersection queries among a set of n segments uses $O(n \log n)$ storage.

Results

Lemma: A 2-level partition tree for line-intersection queries among a set of n segments uses $O(n \log n)$ storage.

Lemma: Let S be a set of n segments in the plane. For any $\varepsilon > 0$, there is a 2-level partition tree \mathcal{T} for S s.t.:

Results

Lemma: A 2-level partition tree for line-intersection queries among a set of n segments uses $O(n \log n)$ storage.

Lemma: Let S be a set of n segments in the plane. For any $\varepsilon > 0$, there is a 2-level partition tree \mathcal{T} for S s.t.:

- given a query line ℓ , we can select $O(n^{1/2+\varepsilon})$ nodes from \mathcal{T} whose canonical subsets represent the segments intersected by ℓ .

Results

Lemma: A 2-level partition tree for line-intersection queries among a set of n segments uses $O(n \log n)$ storage.

Lemma: Let S be a set of n segments in the plane. For any $\varepsilon > 0$, there is a 2-level partition tree \mathcal{T} for S s.t.:

- given a query line ℓ , we can select $O(n^{1/2+\varepsilon})$ nodes from \mathcal{T} whose canonical subsets represent the segments intersected by ℓ .
- The selection takes $O(n^{1/2+\varepsilon})$ time.

Results

Lemma: A 2-level partition tree for line-intersection queries among a set of n segments uses $O(n \log n)$ storage.

Lemma: Let S be a set of n segments in the plane. For any $\varepsilon > 0$, there is a 2-level partition tree \mathcal{T} for S s.t.:

- given a query line ℓ , we can select $O(n^{1/2+\varepsilon})$ nodes from \mathcal{T} whose canonical subsets represent the segments intersected by ℓ .
- The selection takes $O(n^{1/2+\varepsilon})$ time.

Corollary: Let S be a set of n segments in the plane. After $O(\dots)$ -time preprocessing, we can count the number of segments in S intersected by a query line in $O(n^{1/2+\varepsilon})$ time.