Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems

Institute for Informatics

Computational Geometry

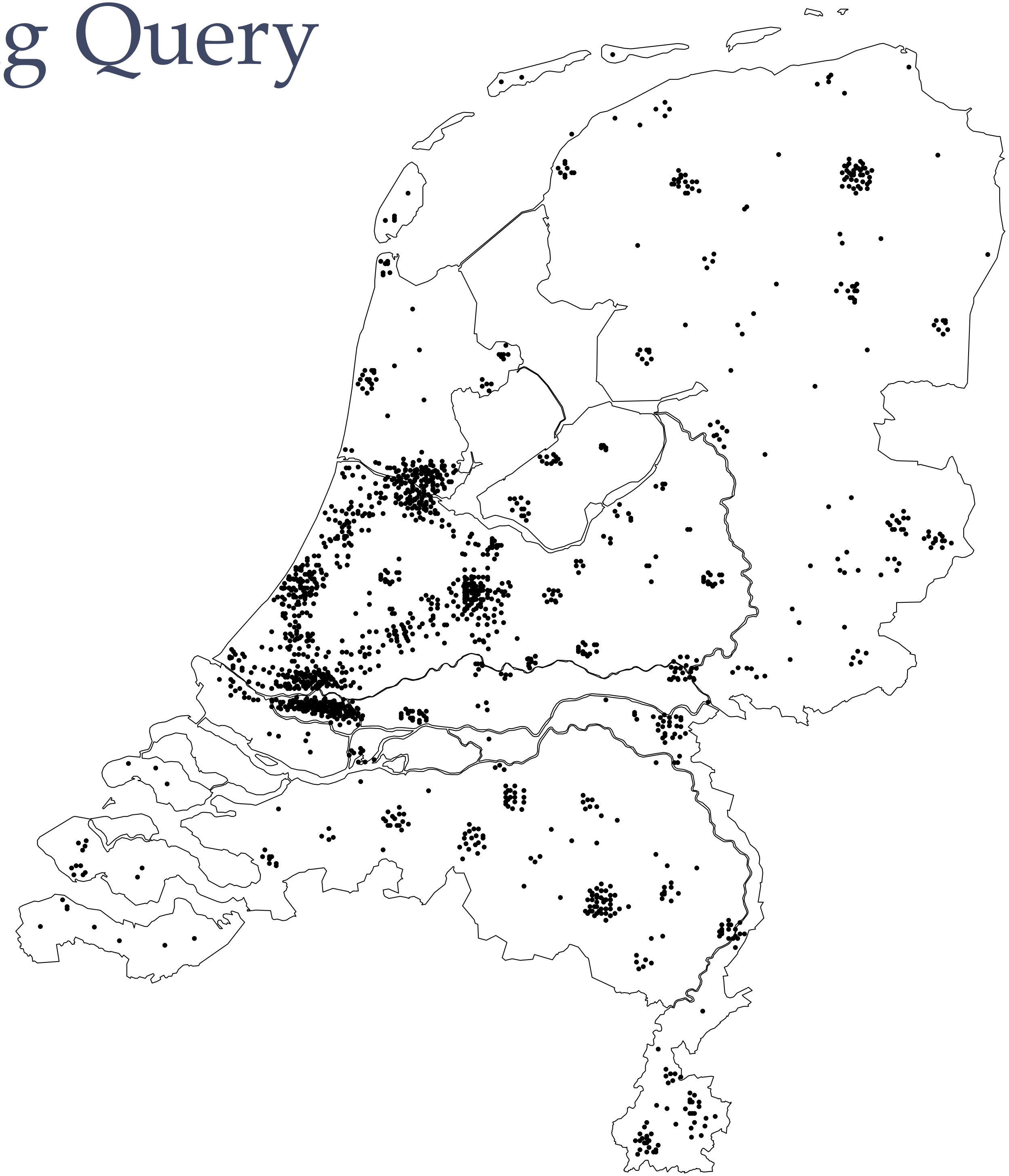
Simple Range Searching

Lecture #11

Dr. Philipp Kindermann

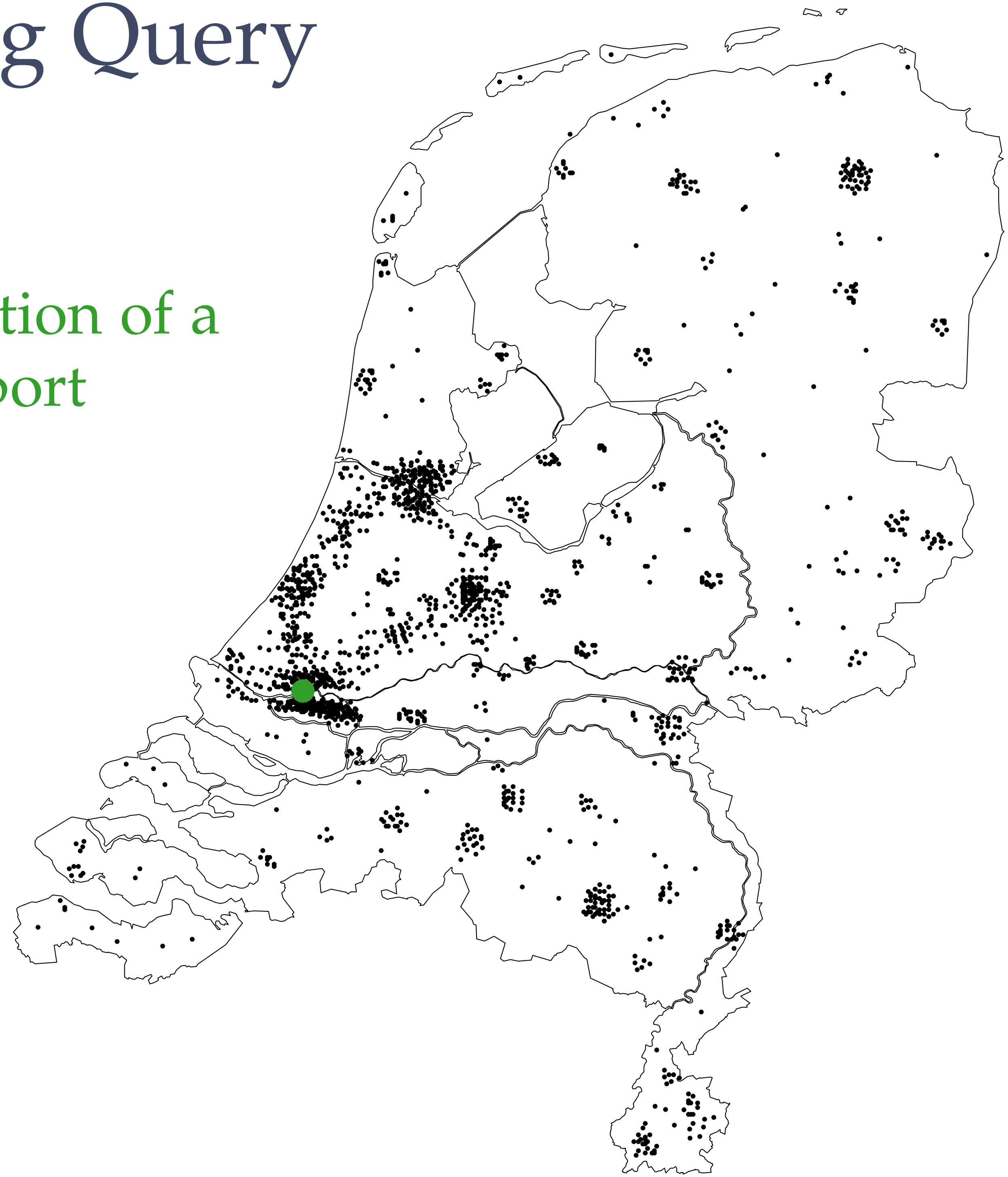
Winter Semester 2018/19

Range-Counting Query



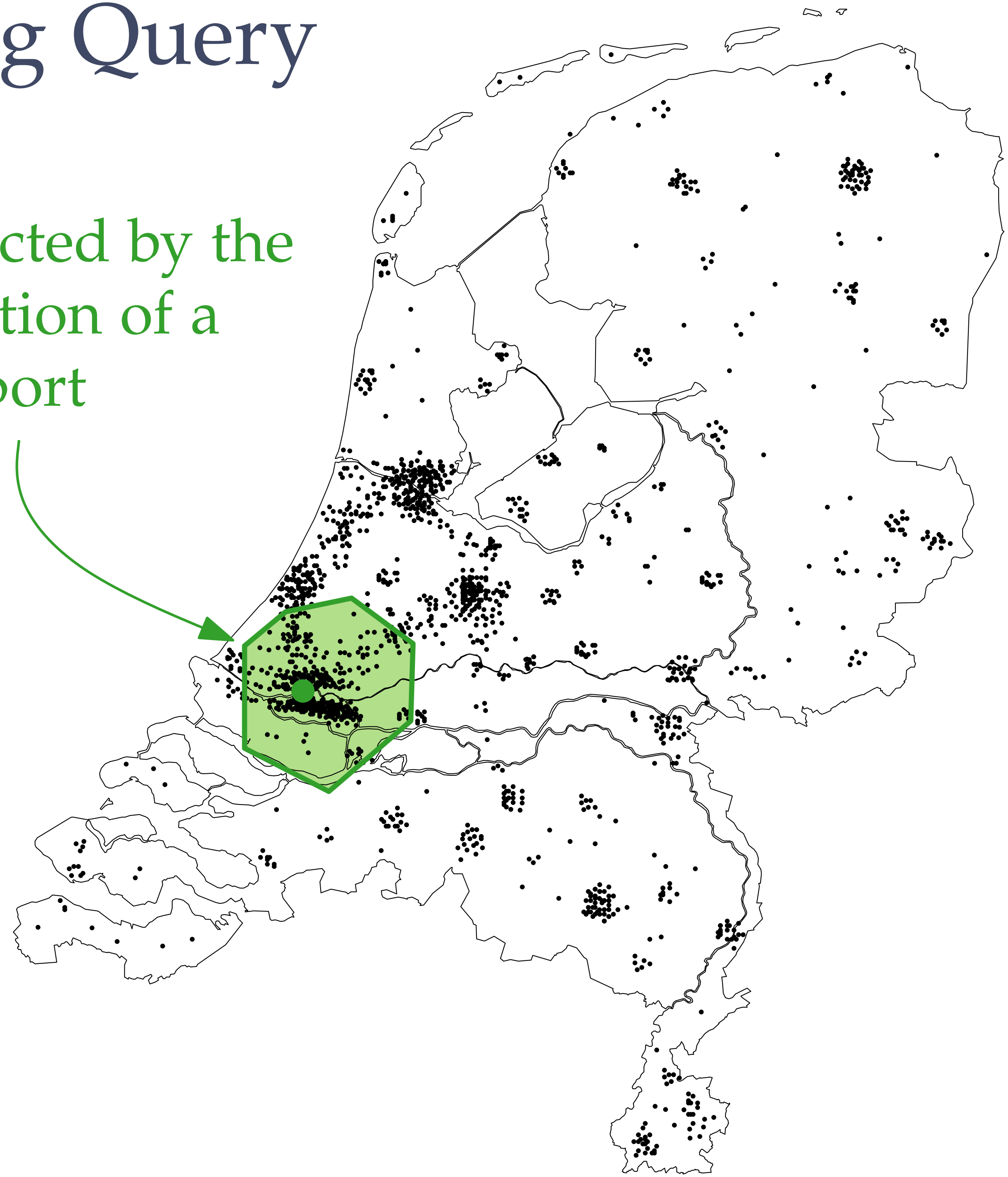
Range-Counting Query

construction of a
new airport



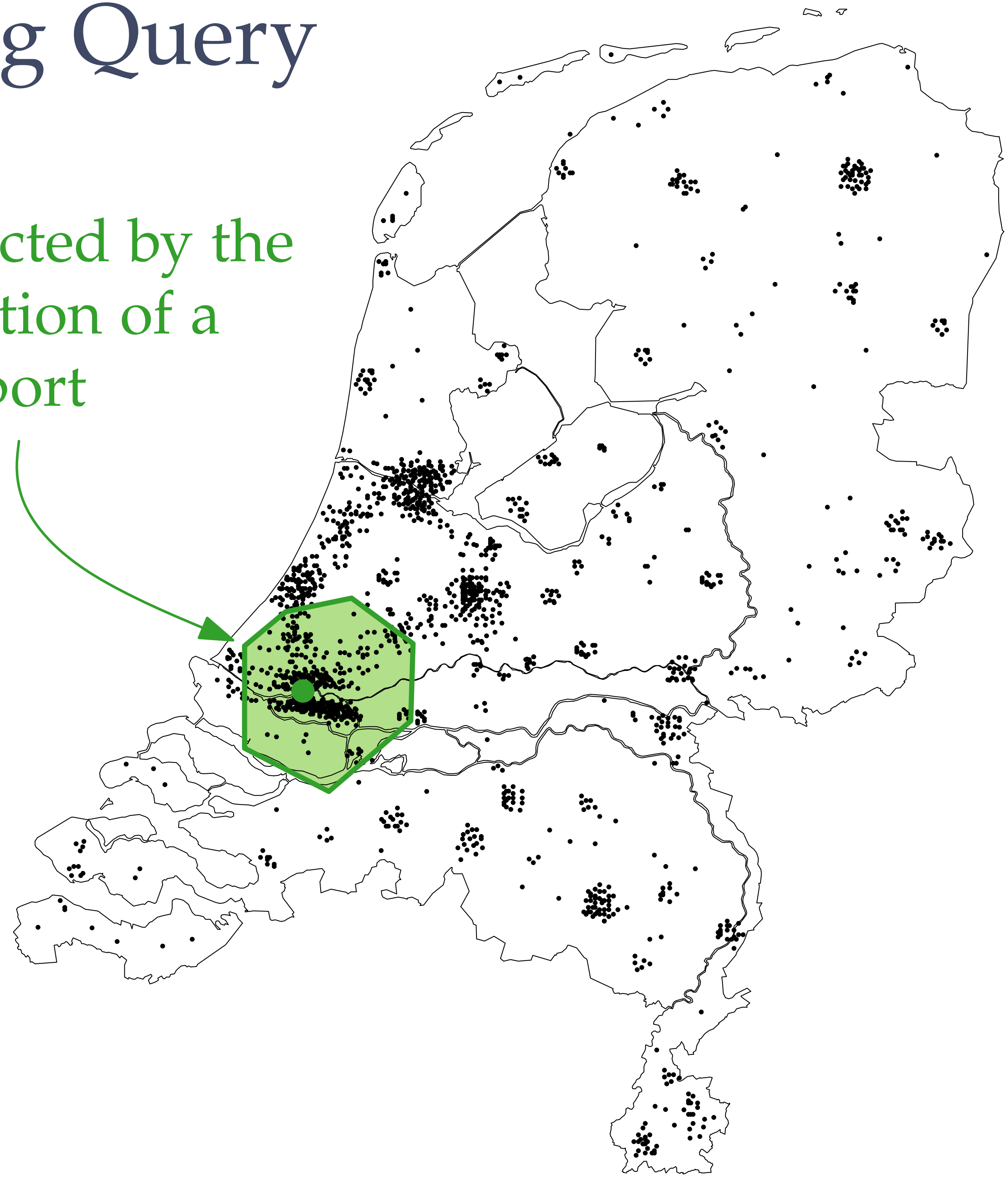
Range-Counting Query

area affected by the construction of a new airport



Range-Counting Query

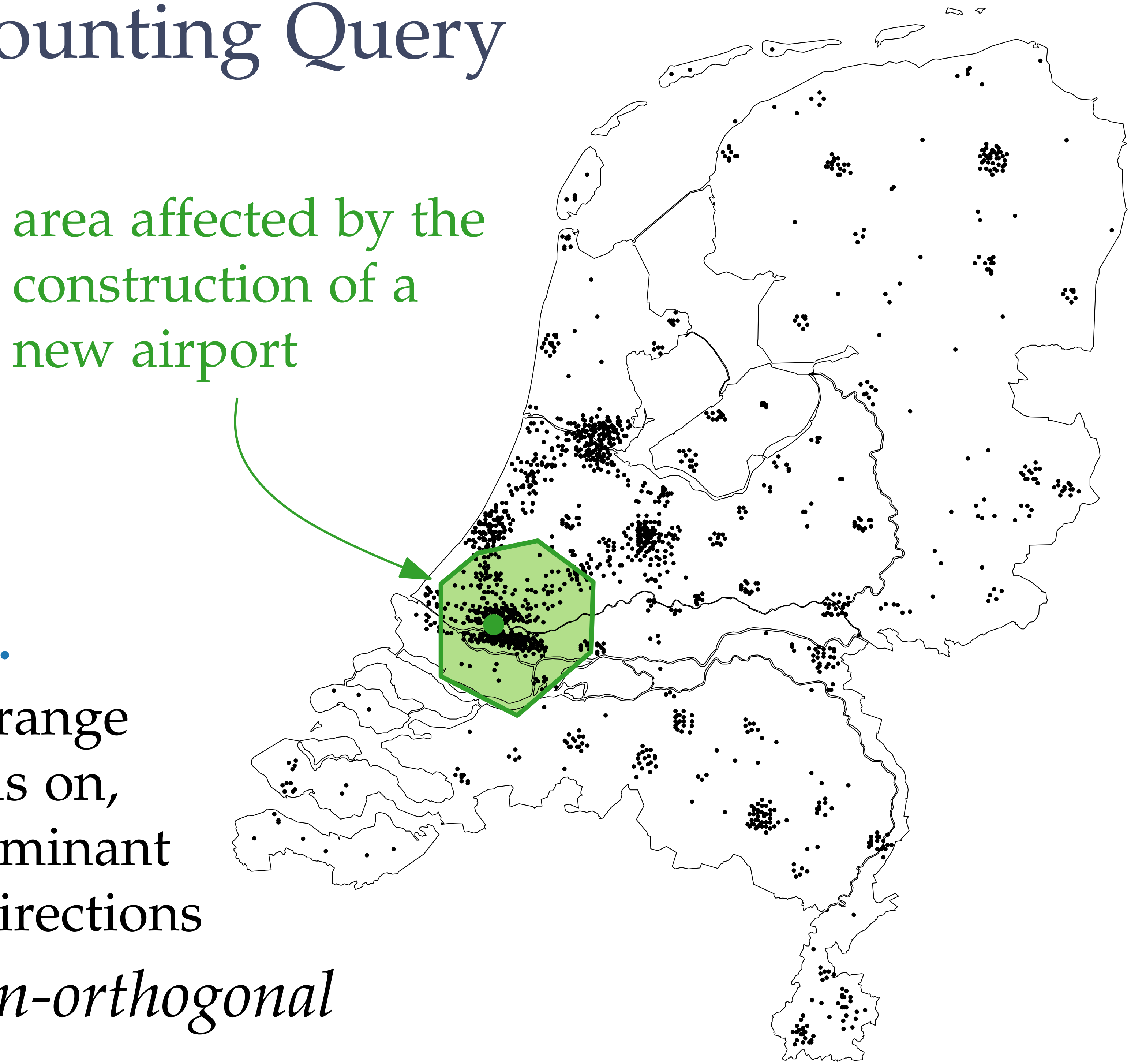
area affected by the construction of a new airport



Observation.

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

Range-Counting Query

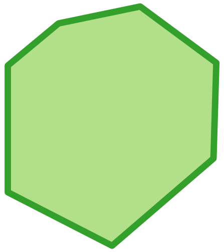


Observation.

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

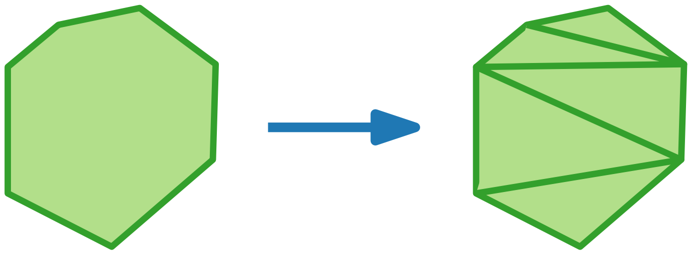
$\Rightarrow$ *non-orthogonal*

Non-orthogonal range queries

Query range: 

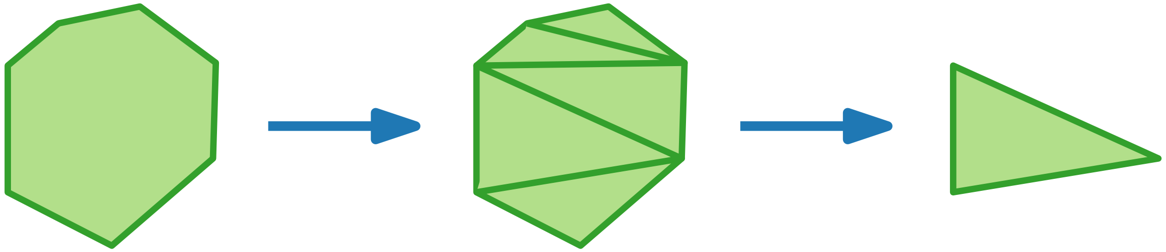
Non-orthogonal range queries

Query range:



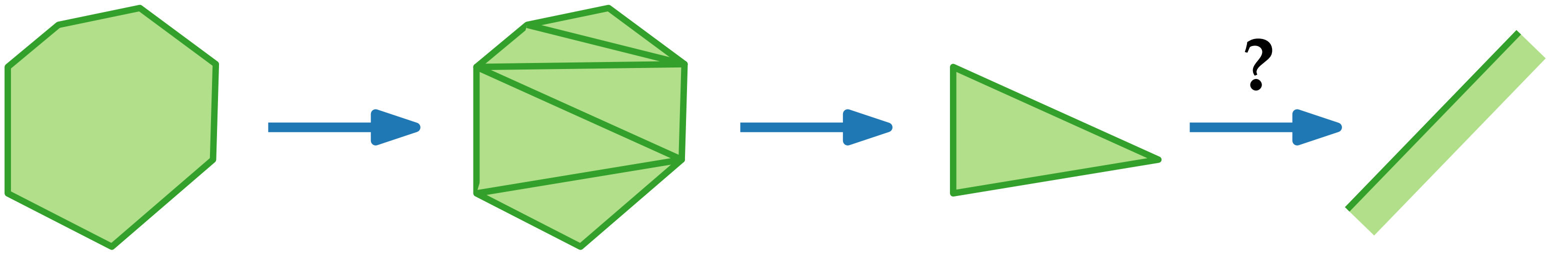
Non-orthogonal range queries

Query range:



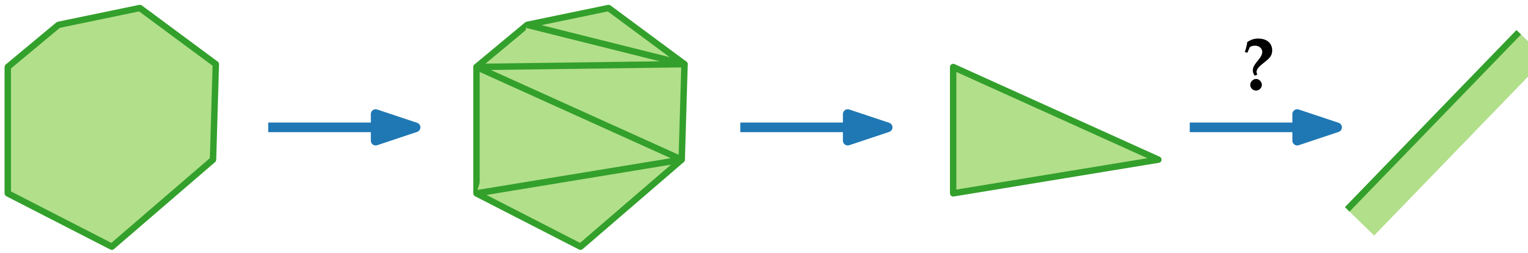
Non-orthogonal range queries

Query range:



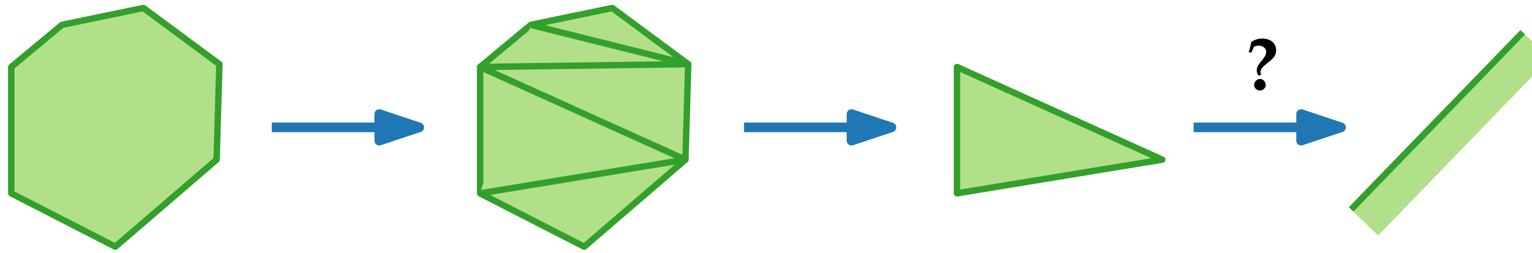
Non-orthogonal range queries

Query range:



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

Non-orthogonal range queries

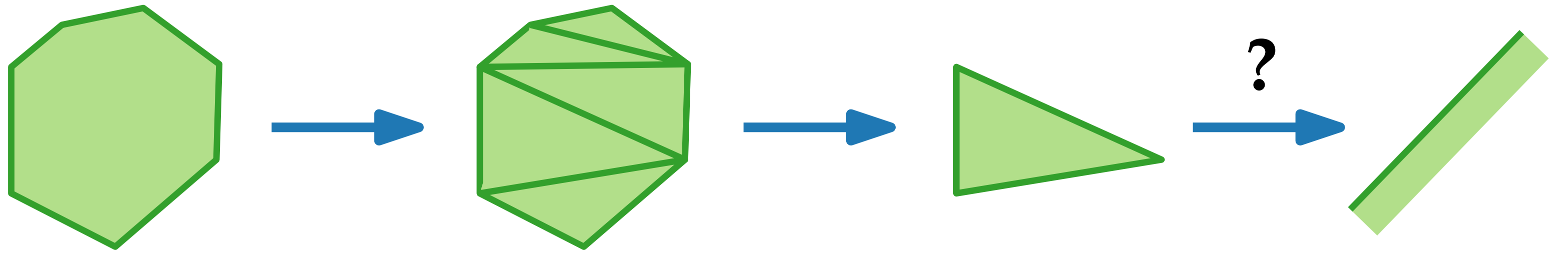
Query range: 

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

Task Design a data structure for the 1-dim. case:

Non-orthogonal range queries

Query range:

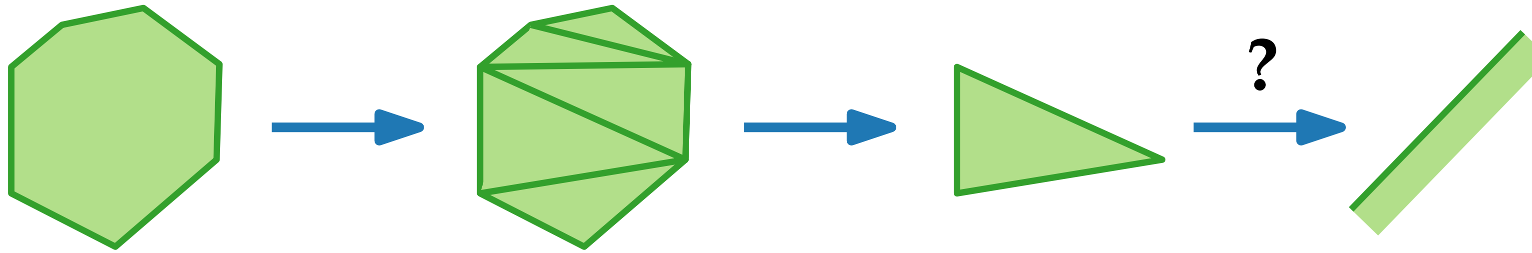


Problem. Given a set P of n points, preprocess P such that *half-space range-counting queries* can be answered quickly.

Task Design a data structure for the 1-dim. case:

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

Non-orthogonal range queries

Query range: 

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

Task Design a data structure for the 1-dim. case:

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

– Consider P static / dynamic!

The 1-Dimensional Case

Task. Design a data structure for the 1-dim. case!

Solution.

The 1-Dimensional Case

Task. Design a data structure for the 1-dim. case!

Solution.
- use balanced binary search trees

The 1-Dimensional Case

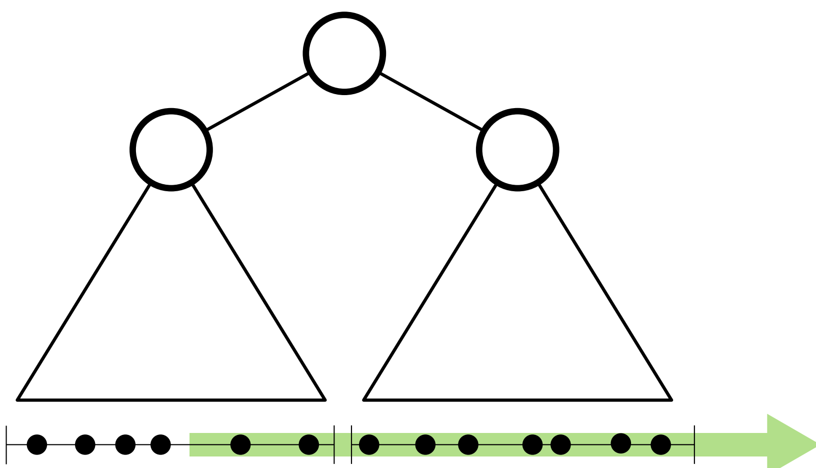
Task. Design a data structure for the 1-dim. case!

Solution.
- use balanced binary search trees
- augment each node with the number of nodes in its subtree [see Cormen et al., *Introduction to Algorithms*, MIT press, 3rd ed., 2009]

The 1-Dimensional Case

Task. Design a data structure for the 1-dim. case!

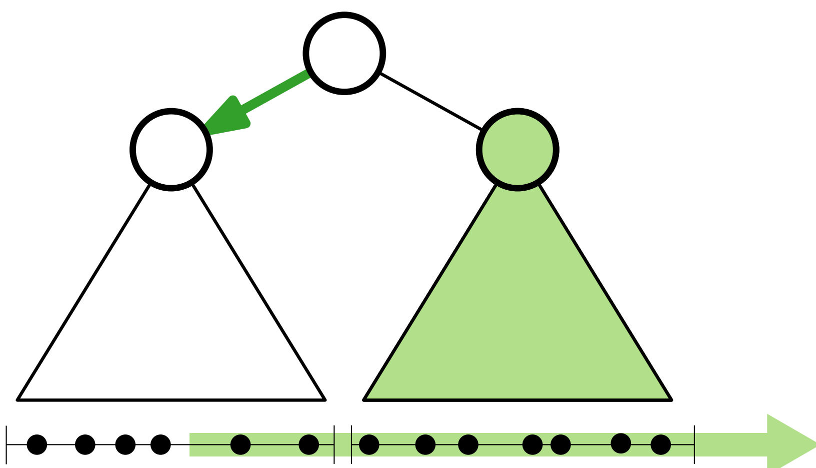
Solution.
- use balanced binary search trees
- augment each node with the number of nodes in its subtree [see Cormen et al., *Introduction to Algorithms*, MIT press, 3rd ed., 2009]



The 1-Dimensional Case

Task. Design a data structure for the 1-dim. case!

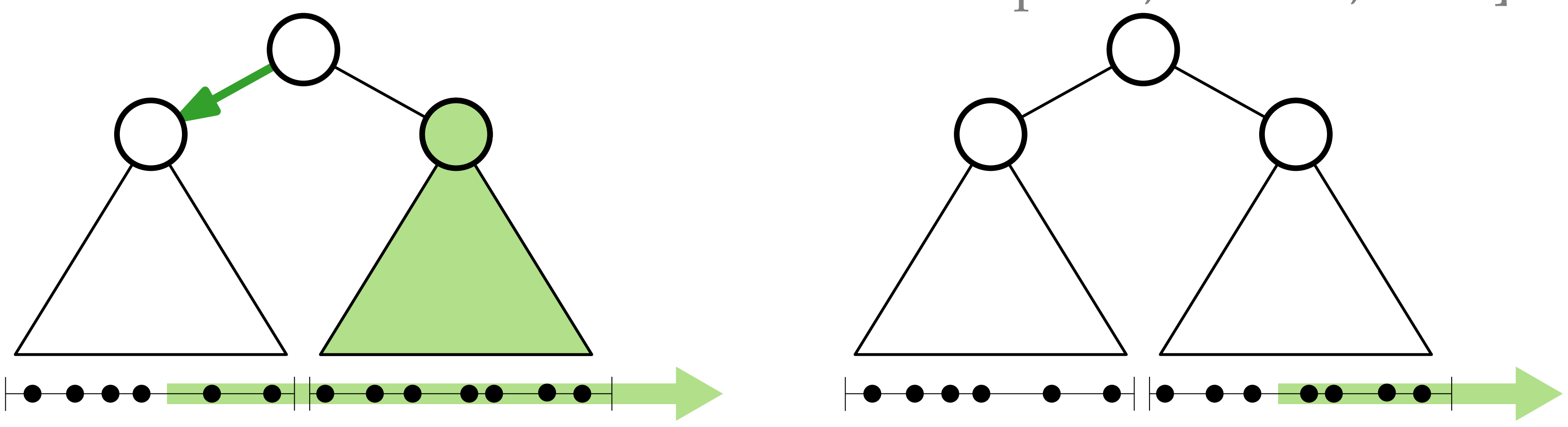
Solution.
- use balanced binary search trees
- augment each node with the number of nodes in its subtree [see Cormen et al., *Introduction to Algorithms*, MIT press, 3rd ed., 2009]



The 1-Dimensional Case

Task. Design a data structure for the 1-dim. case!

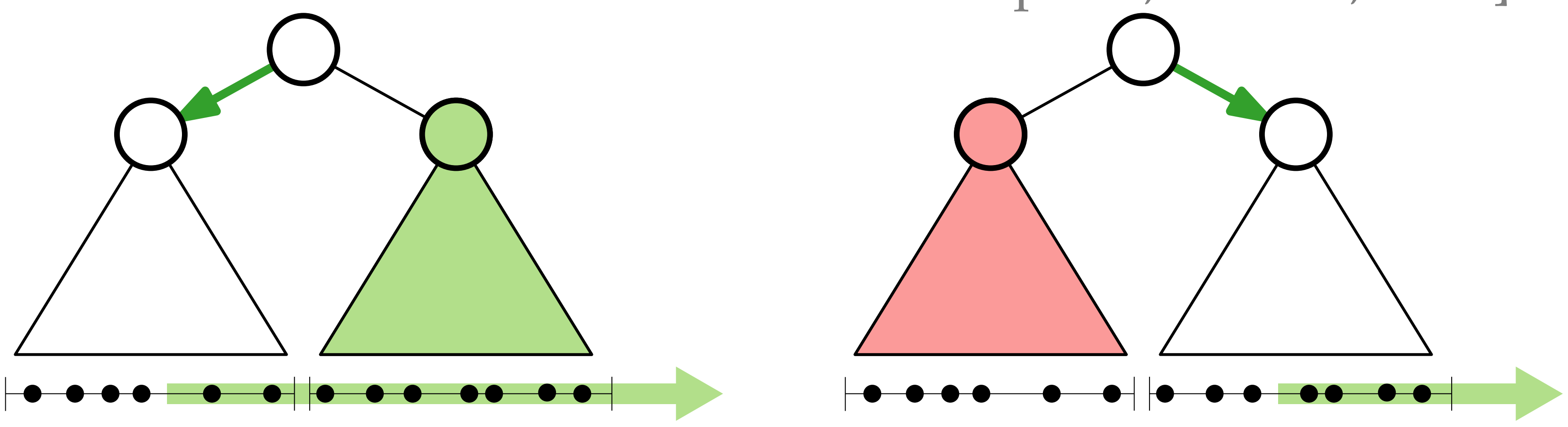
Solution.
- use balanced binary search trees
- augment each node with the number of nodes in its subtree [see Cormen et al., *Introduction to Algorithms*, MIT press, 3rd ed., 2009]



The 1-Dimensional Case

Task. Design a data structure for the 1-dim. case!

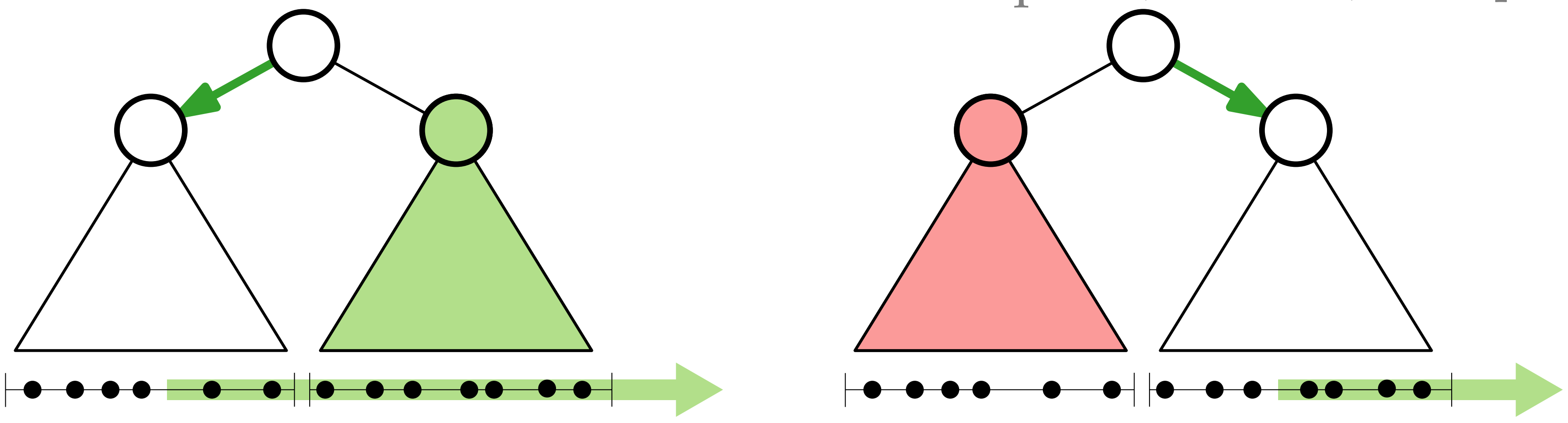
Solution.
- use balanced binary search trees
- augment each node with the number of nodes in its subtree [see Cormen et al., *Introduction to Algorithms*, MIT press, 3rd ed., 2009]



The 1-Dimensional Case

Task. Design a data structure for the 1-dim. case!

Solution.
- use balanced binary search trees
- augment each node with the number of nodes in its subtree [see Cormen et al., *Introduction to Algorithms*, MIT press, 3rd ed., 2009]

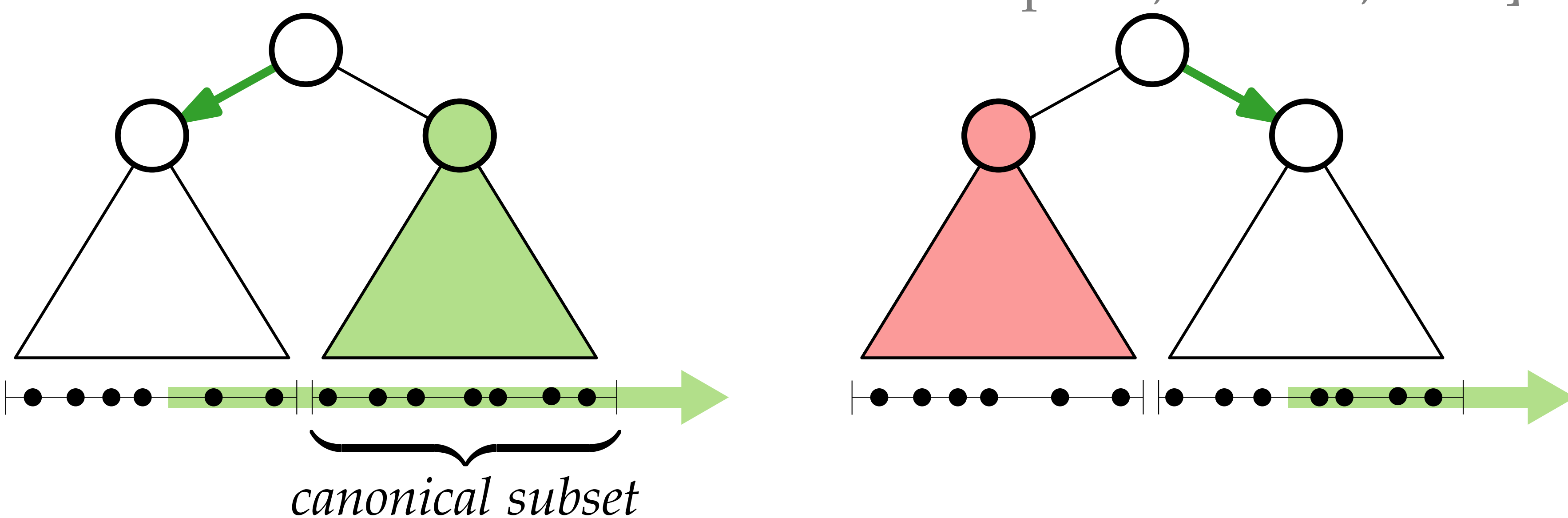


Lesson. On each level, visit ≤ 1 subtree recursively!

The 1-Dimensional Case

Task. Design a data structure for the 1-dim. case!

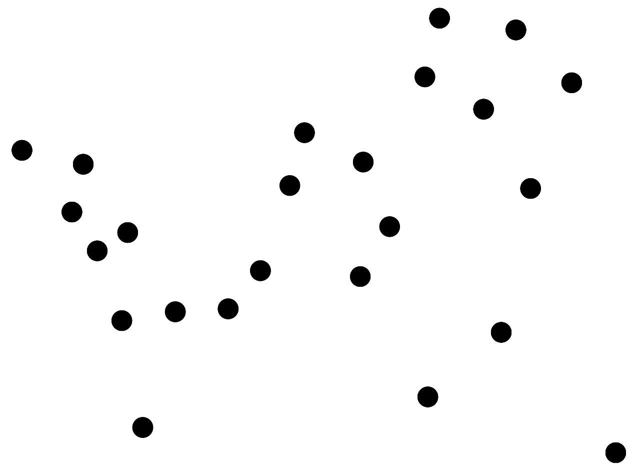
Solution.
- use balanced binary search trees
- augment each node with the number of nodes in its subtree [see Cormen et al., *Introduction to Algorithms*, MIT press, 3rd ed., 2009]



Lesson. On each level, visit ≤ 1 subtree recursively!

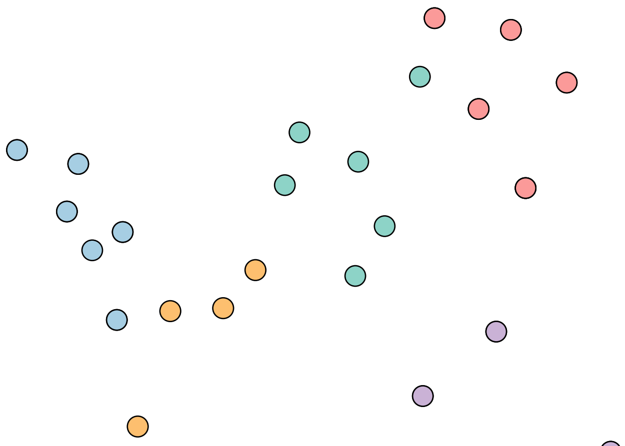
Generalizing to 2 Dimensions

Any ideas?



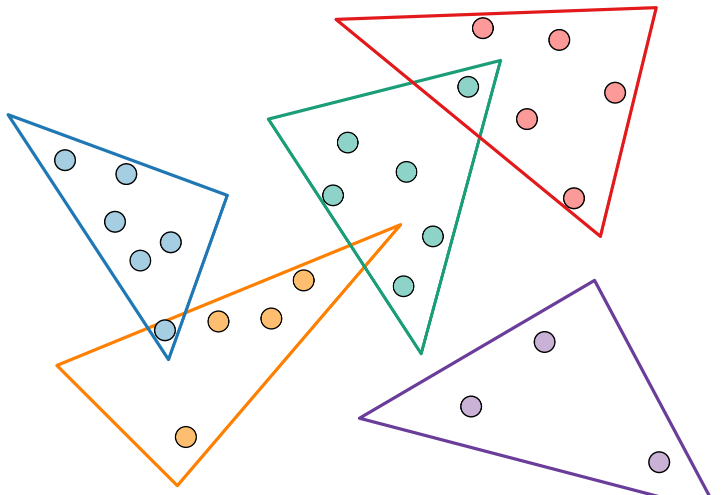
Generalizing to 2 Dimensions

Any ideas?



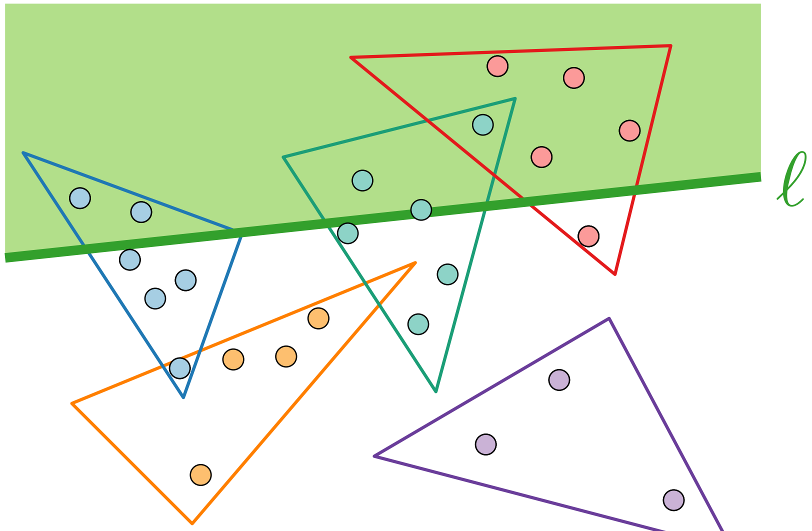
Generalizing to 2 Dimensions

Partition the input!



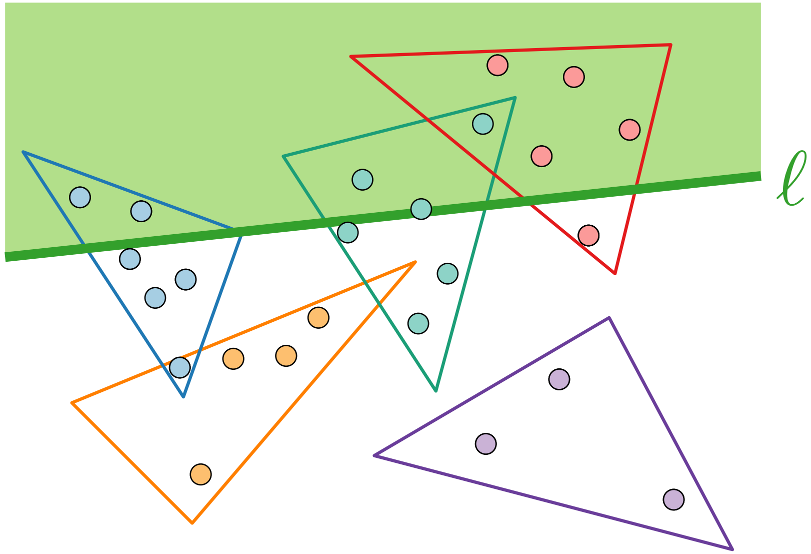
Generalizing to 2 Dimensions

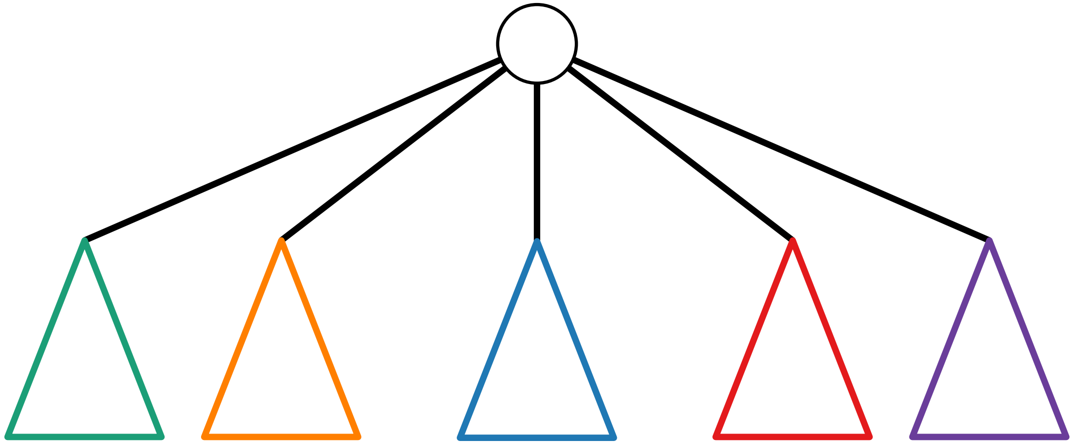
Partition the input! Query…



Generalizing to 2 Dimensions

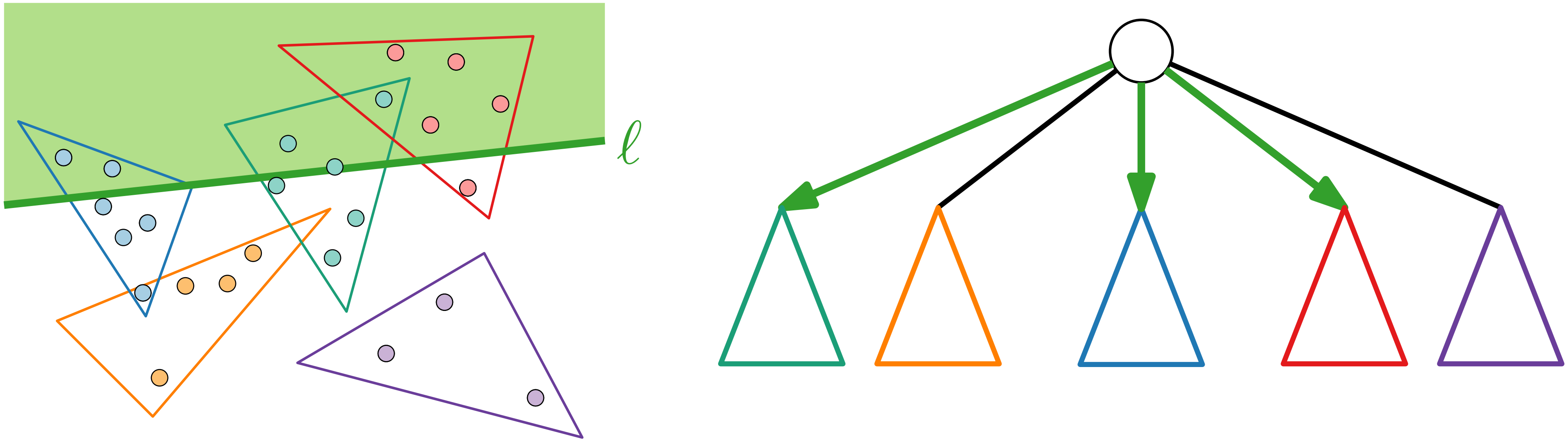
Partition the input! Query… in a *partition tree*





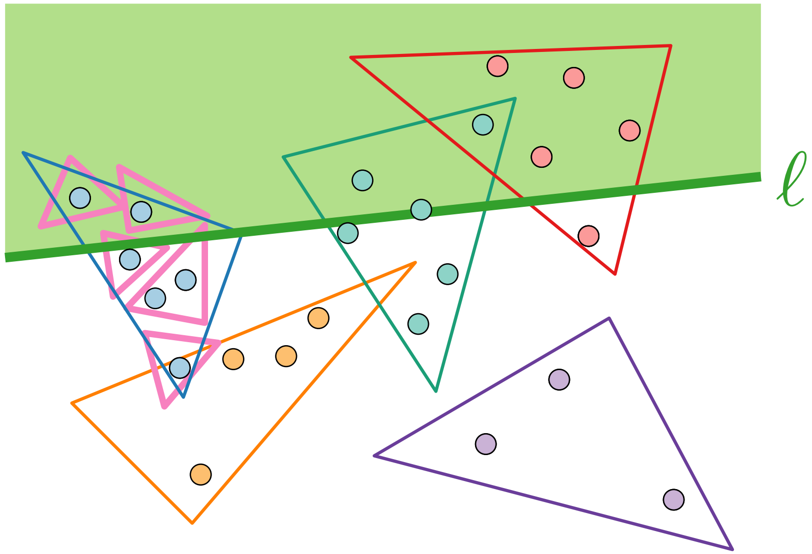
Generalizing to 2 Dimensions

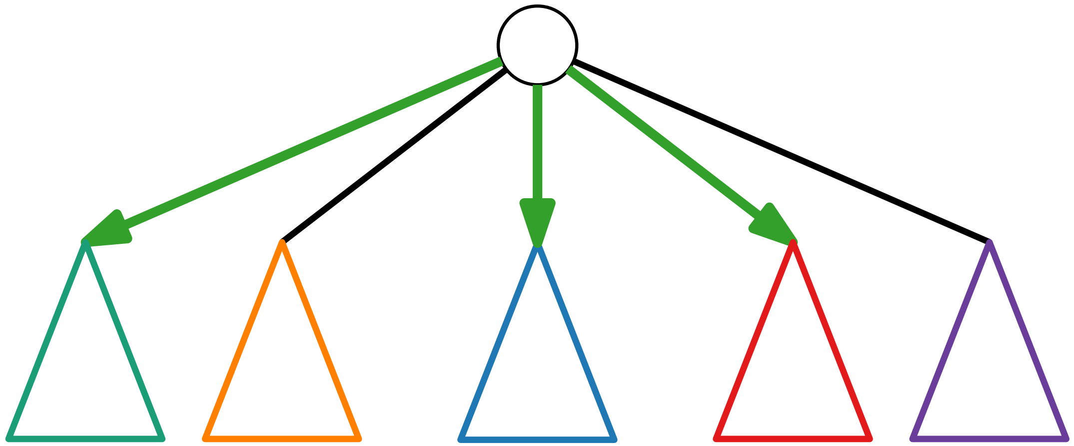
Partition the input! Query… in a *partition tree*



Generalizing to 2 Dimensions

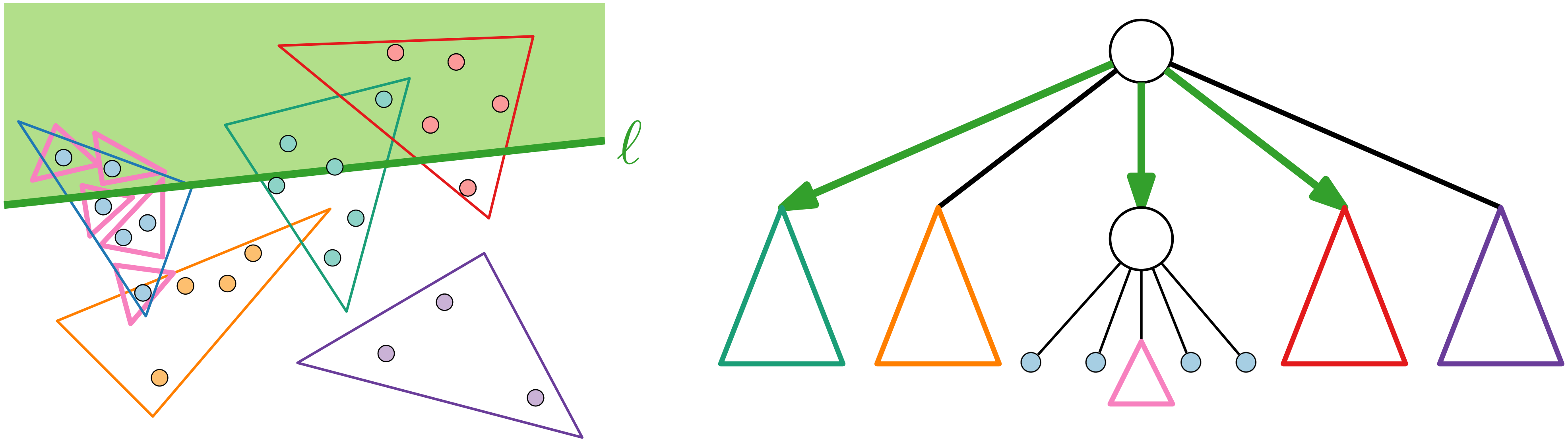
Partition the input! Query… in a *partition tree*





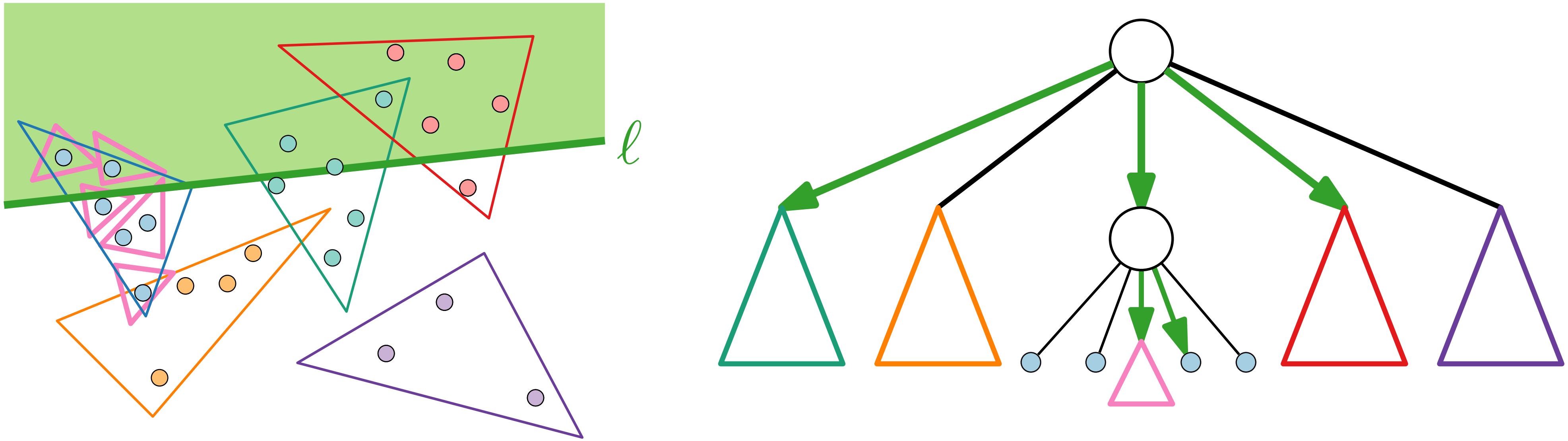
Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!



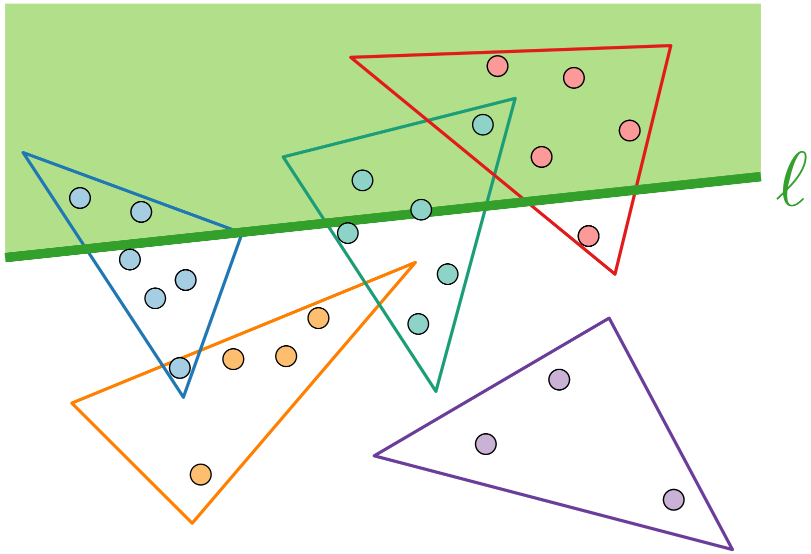
Generalizing to 2 Dimensions

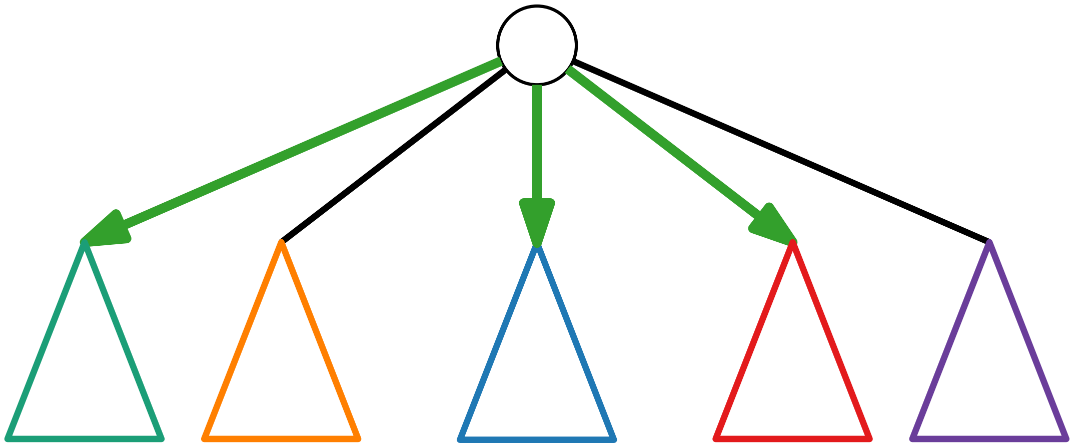
Partition the input! Query… in a *partition tree* …recursively!



Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!

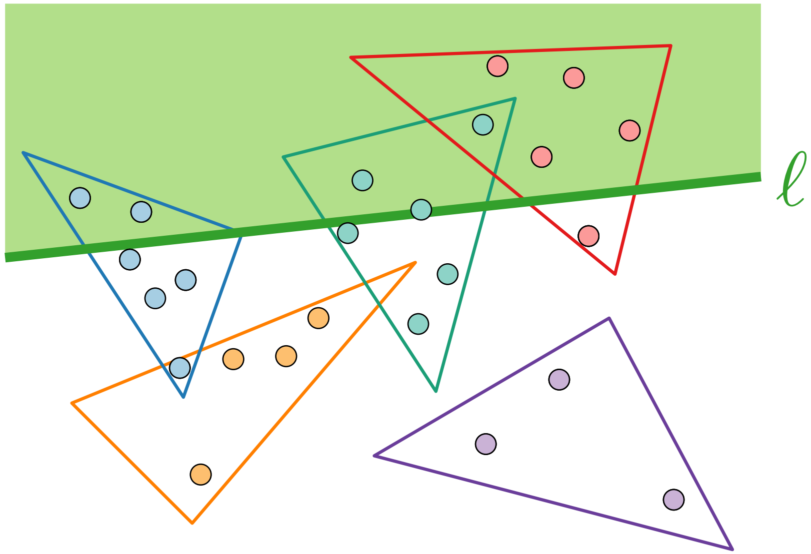


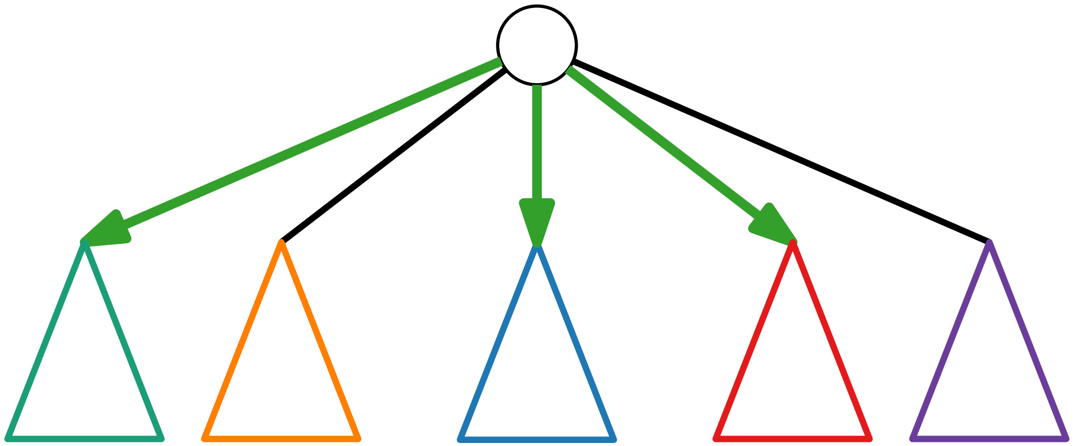


Definition. $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\}$ is a *simplicial partition* (of size r) for S if

Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree* ...recursively!



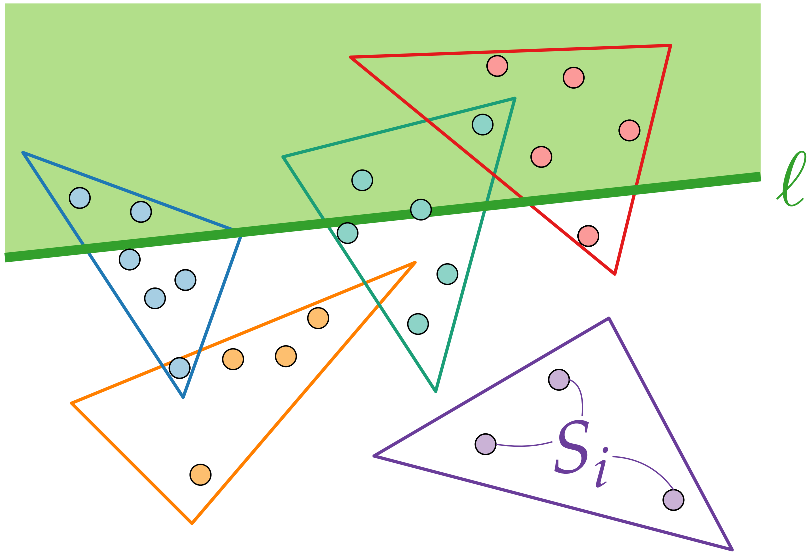


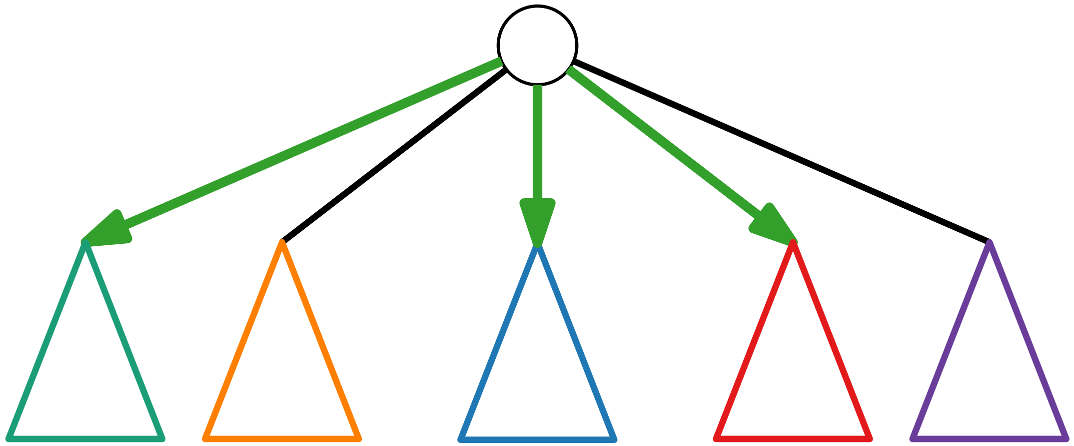
Definition. $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\}$ is a *simplicial partition* (of size r) for S if

– S is partitioned by $S_1, \ldots, S_r$ and

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!



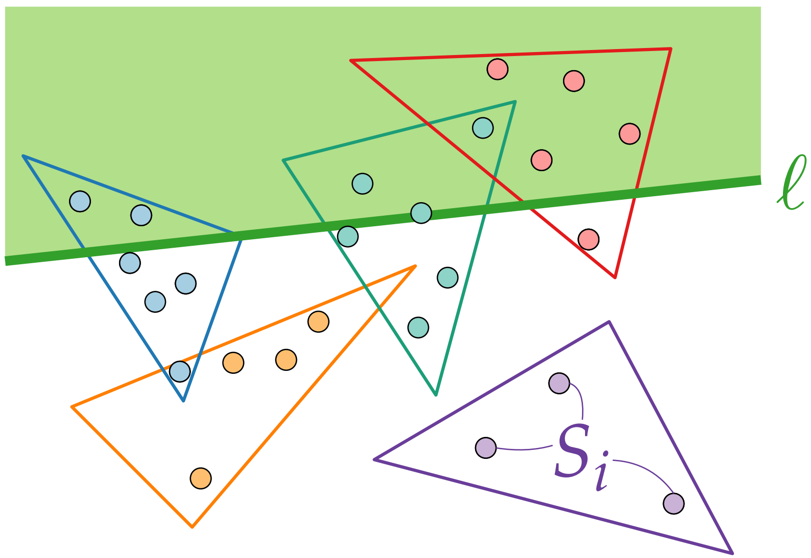


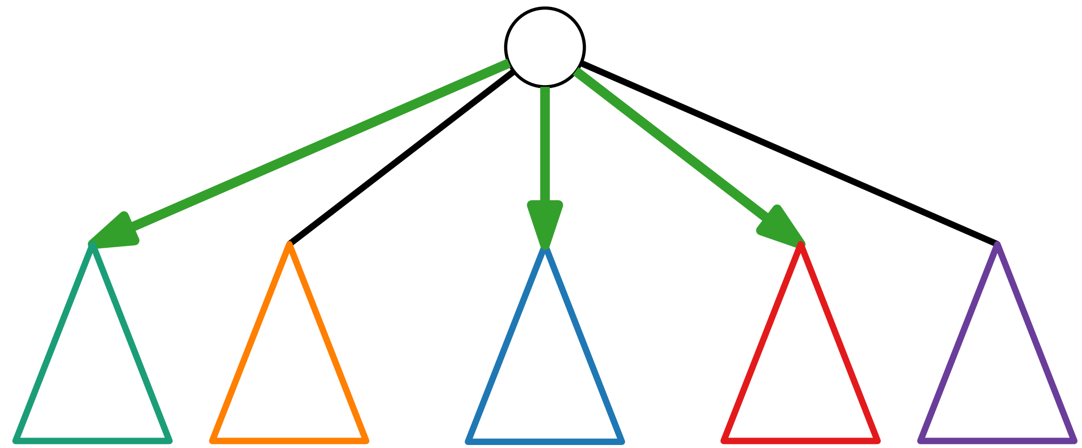
Definition. $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\}$ is a *simplicial partition* (of size r) for S if

– S is partitioned by $S_1, \ldots, S_r$ and

Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree* ...recursively!





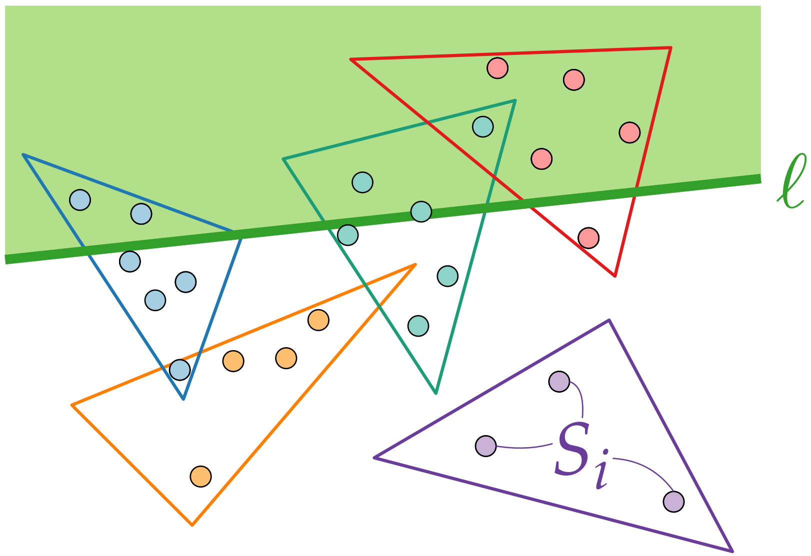
Definition. $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\}$ is a *simplicial partition* (of size r) for S if

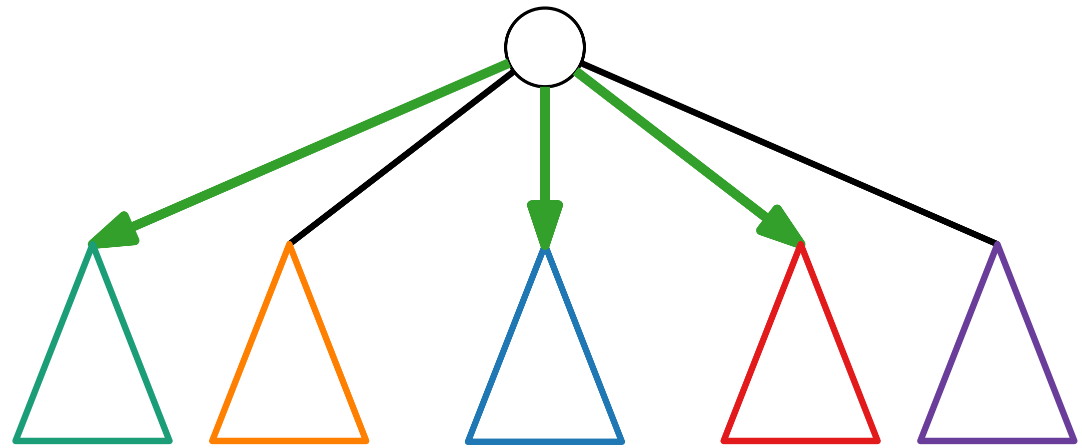
– S is partitioned by $S_1, \ldots, S_r$ and

classes of S

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!





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

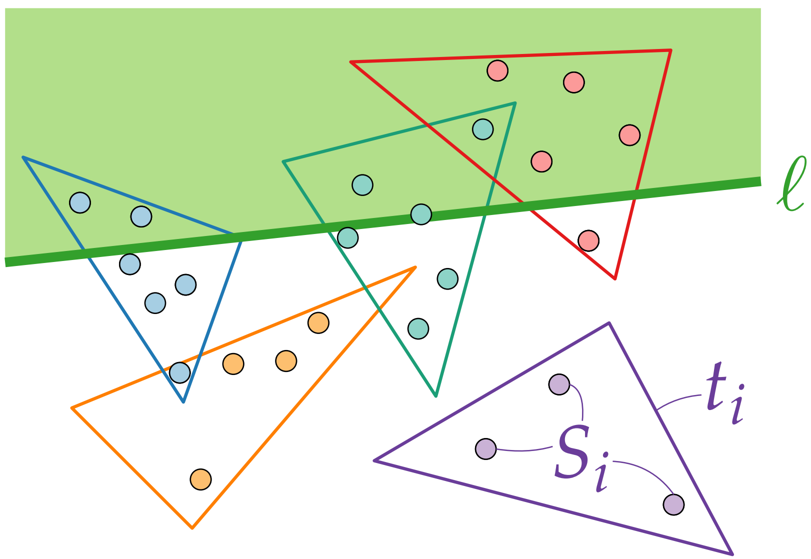
classes of S

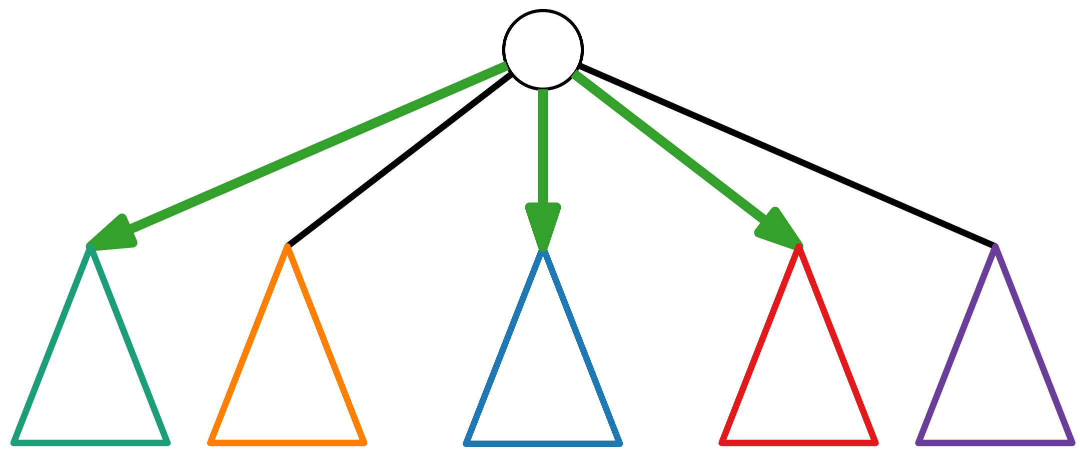
– S is partitioned by $S_1, \ldots, S_r$ and

– for $1 \leq i \leq r$, t_i is a triangle and $S_i \subset t_i$.

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!





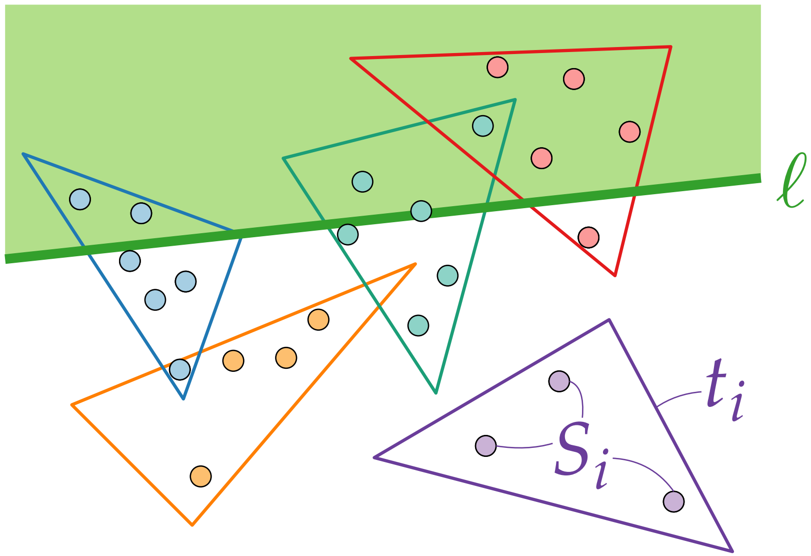
Definition. $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\}$ is a *simplicial partition* (of size r) for S if

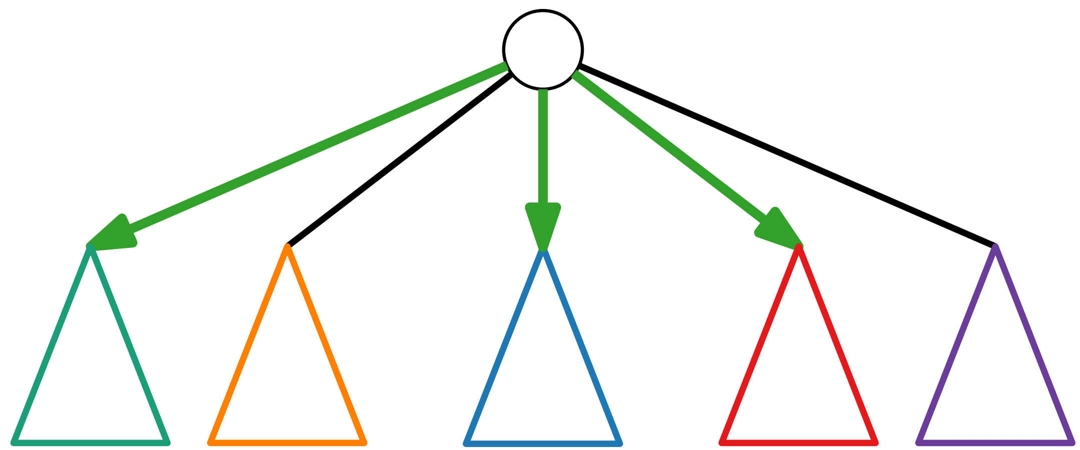
classes of S

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

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!





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

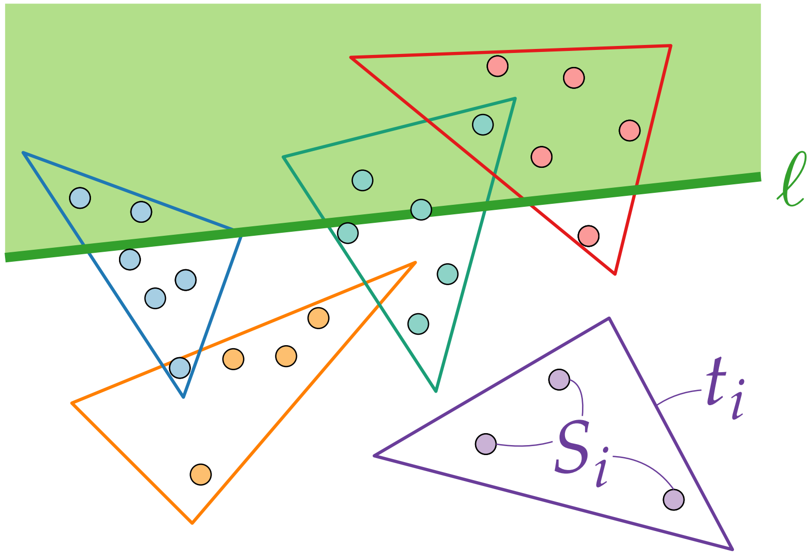
classes of S

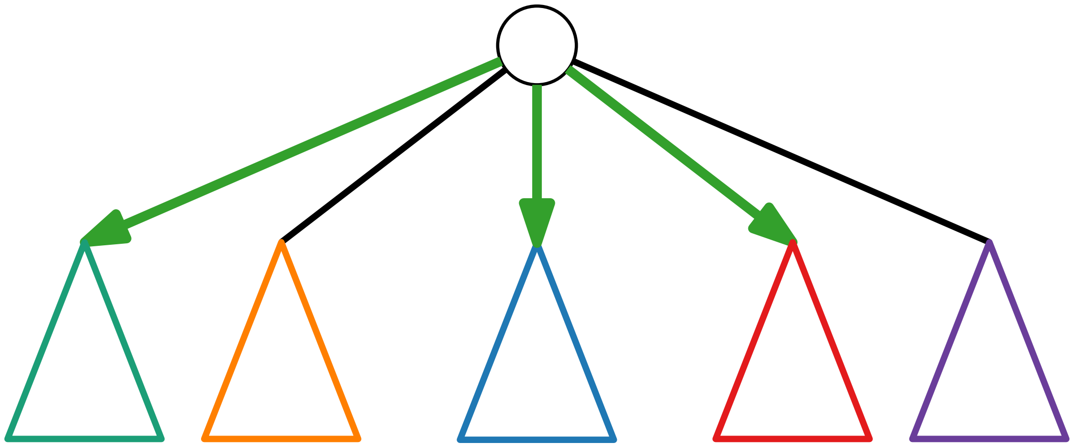
– S is partitioned by $S_1, \ldots, S_r$ and
– for $1 \le i \le r$, t_i is a triangle and $S_i \subset t_i$.

$\Psi(S)$ is *fine* if $|S_i| \le 2\frac{|S|}{r}$ for every $1 \le i \le r$.

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!

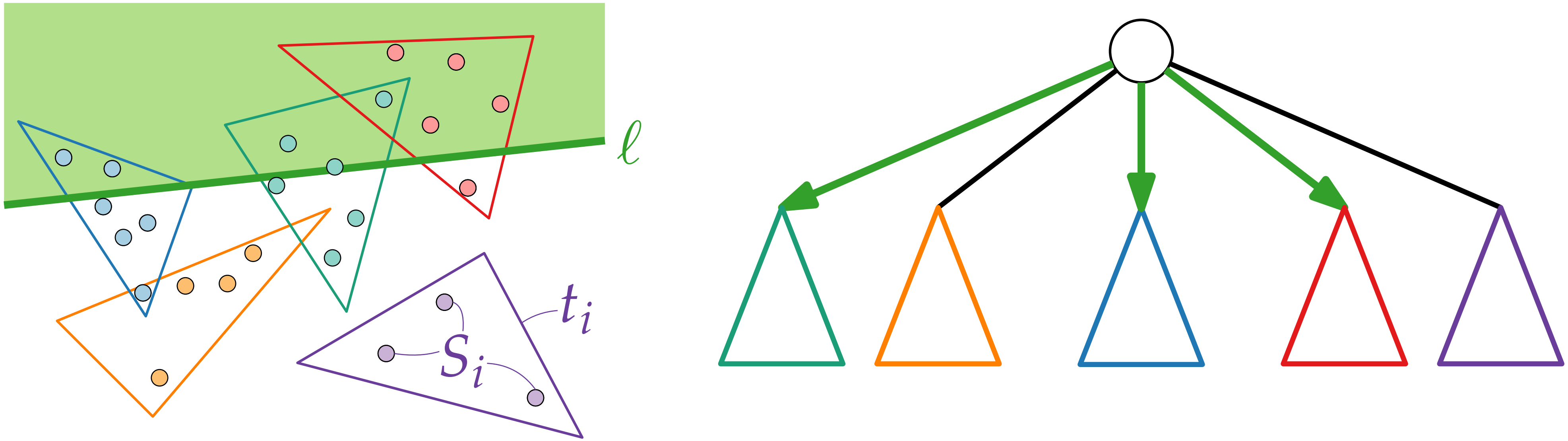




Definition. The *crossing number* of ℓ (w.r.t. $\Psi(S)$) is the number of triangles $t_1, \ldots, t_r$ *crossed* by ℓ.

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!

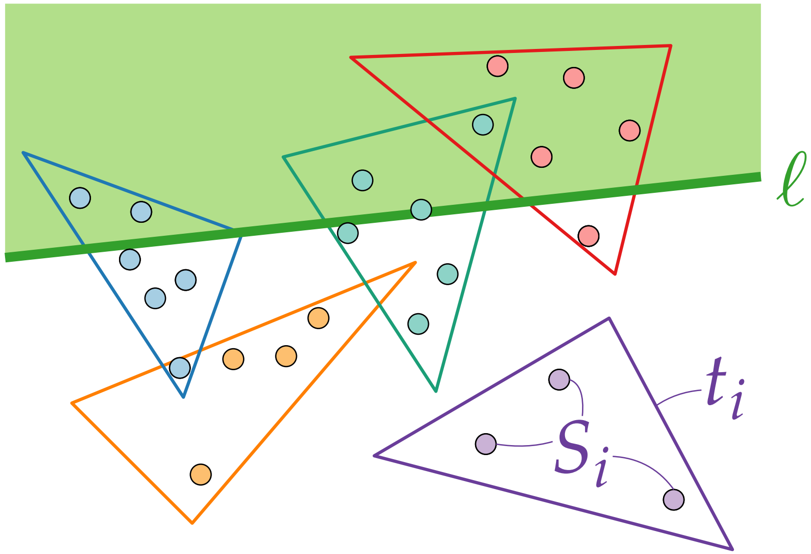


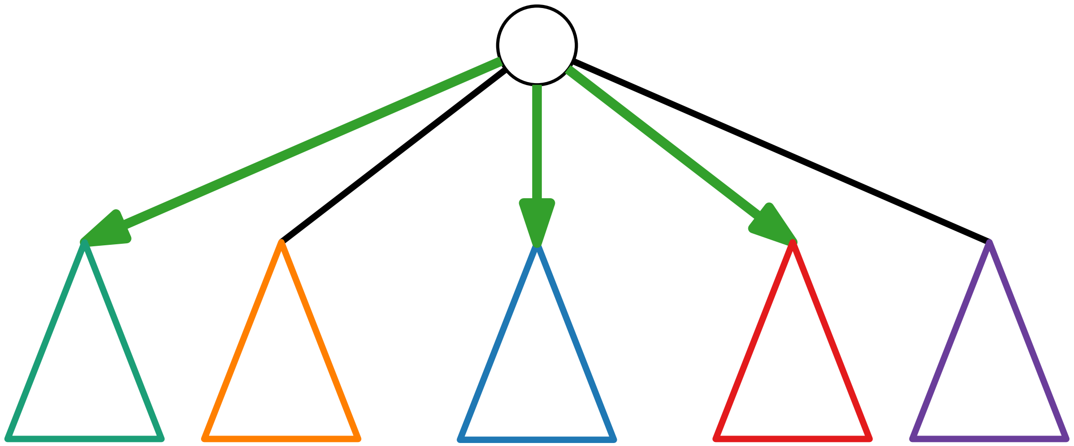
Definition. The *crossing number* of ℓ (w.r.t. $\Psi(S)$) is the number of triangles $t_1, \ldots, t_r$ *crossed* by ℓ.

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

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!

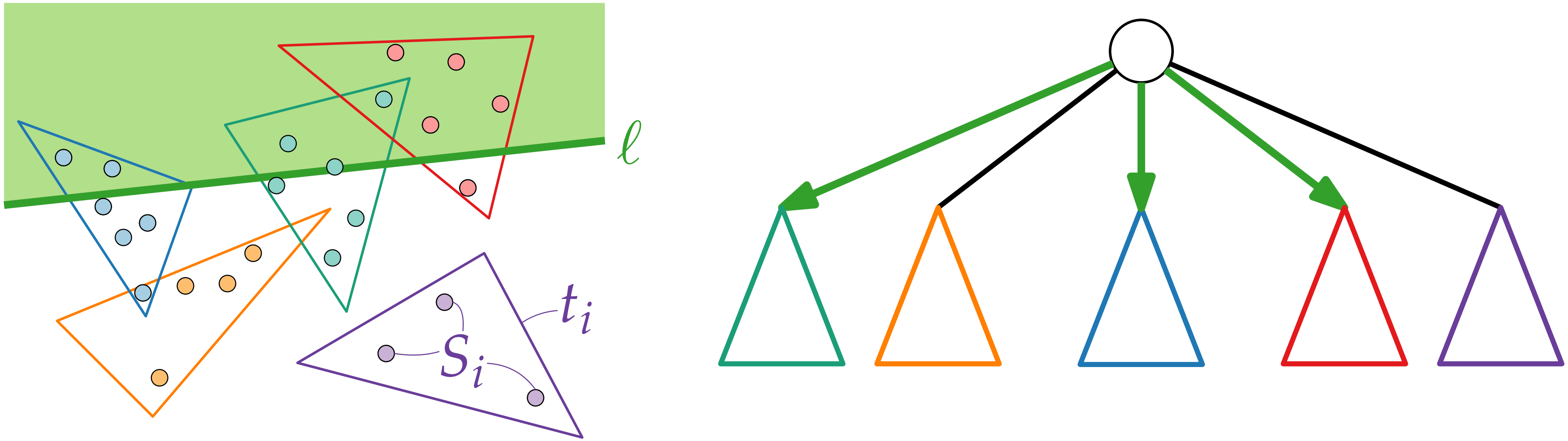




Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\quad)$ exists.

Generalizing to 2 Dimensions

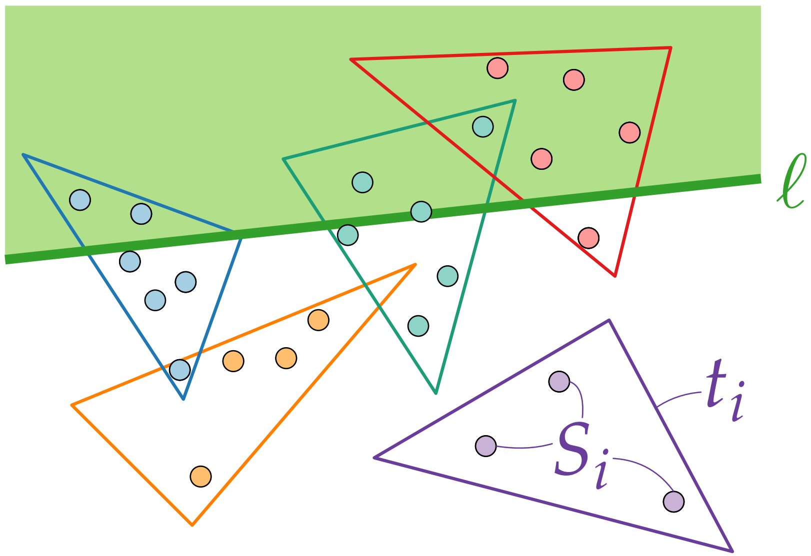
Partition the input! Query… in a *partition tree* …recursively!

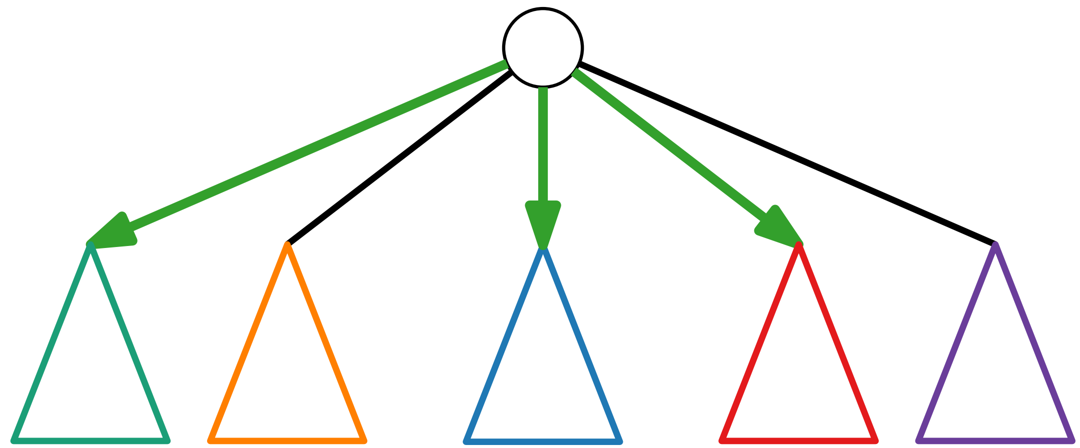


Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \le r \le n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists.

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!

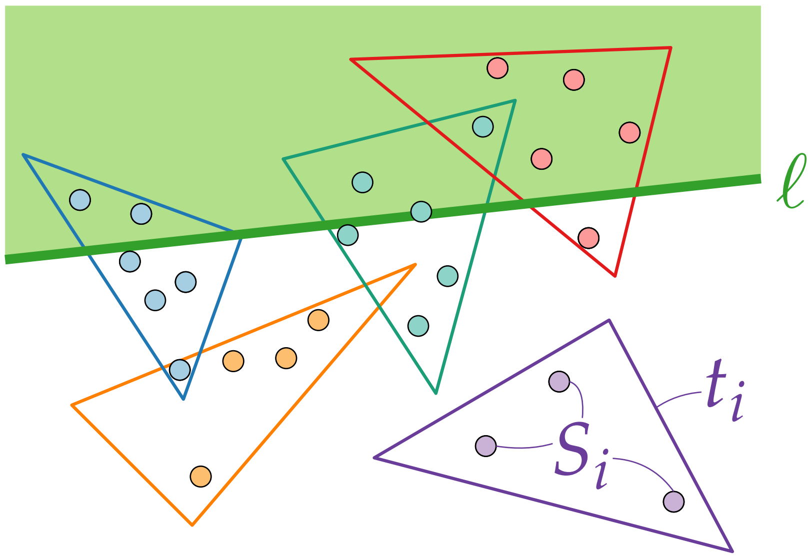


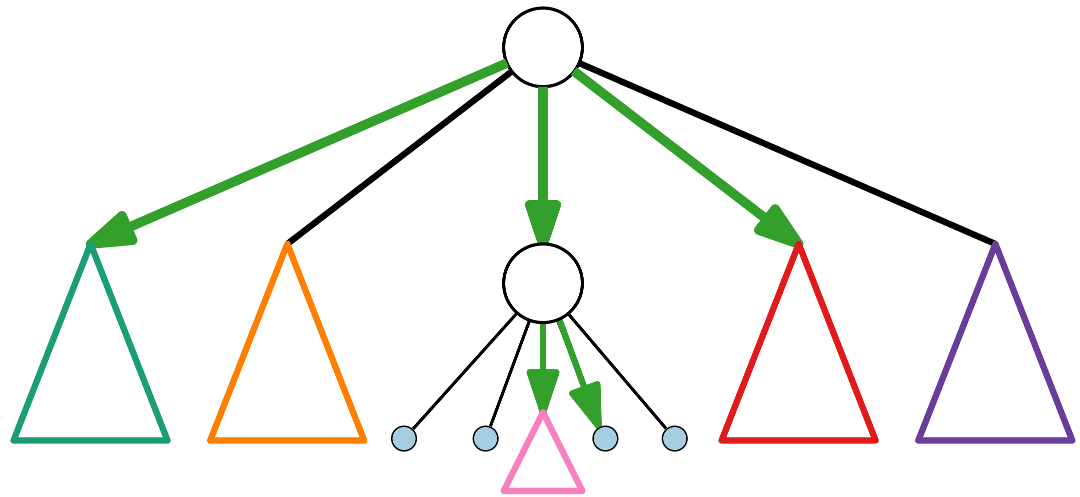


Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!

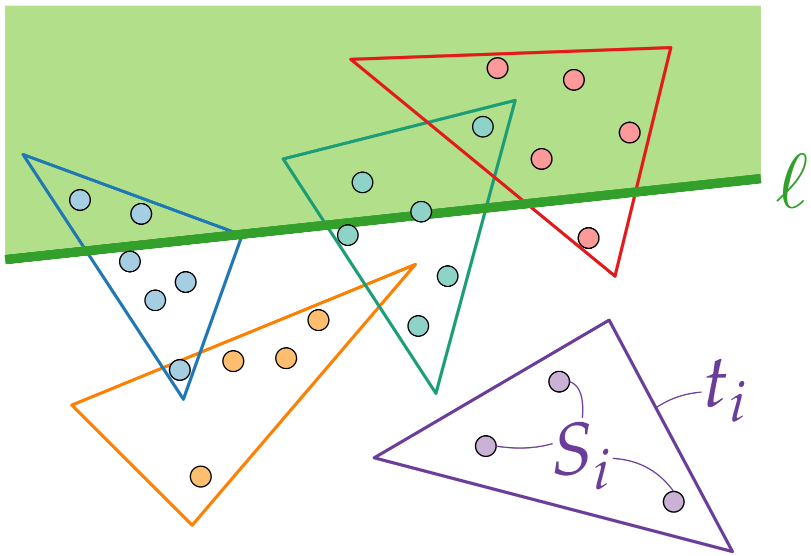


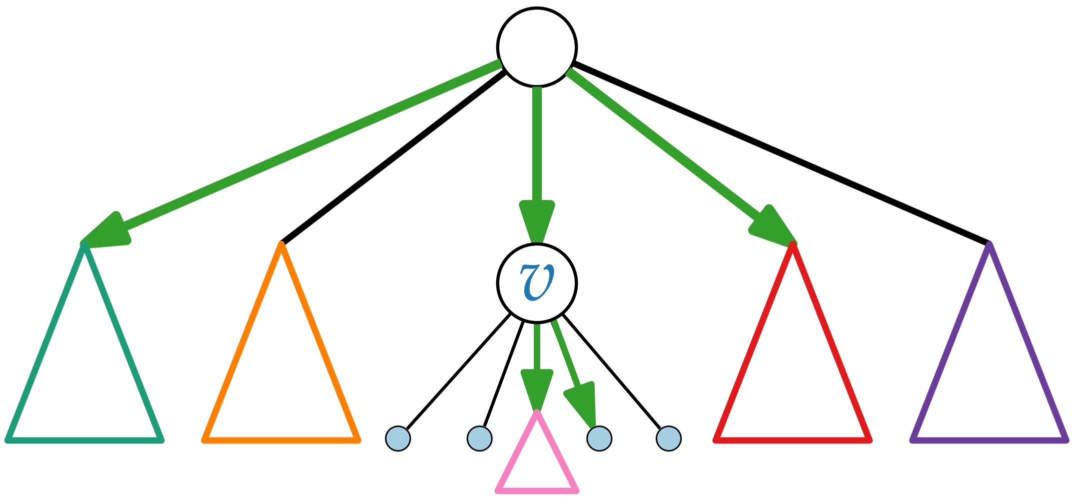


Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!

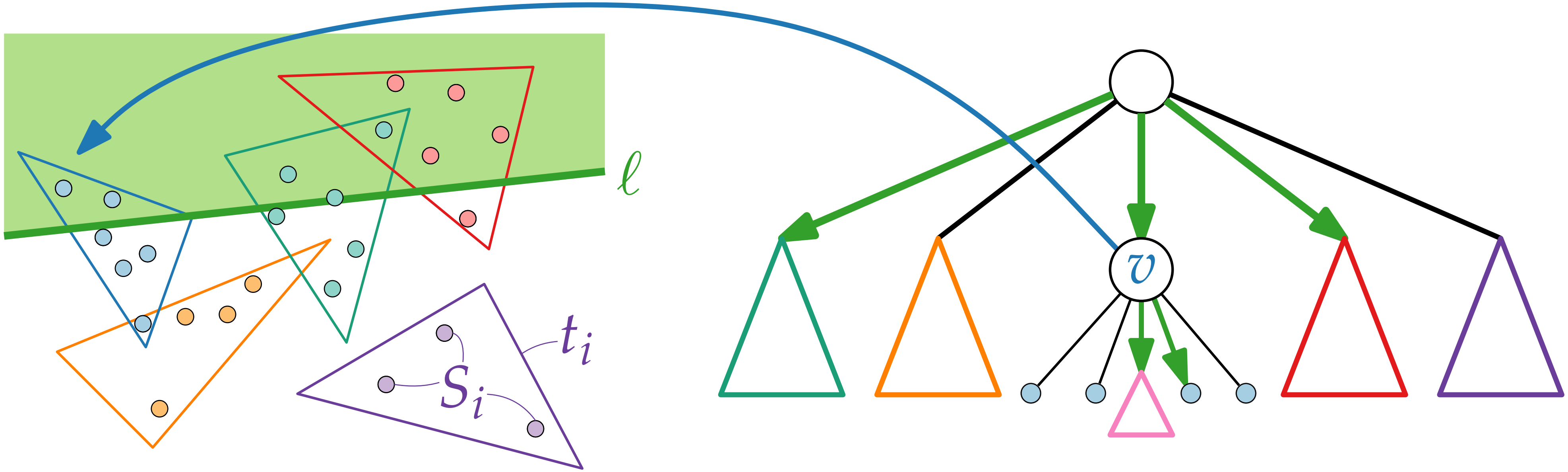




Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Generalizing to 2 Dimensions

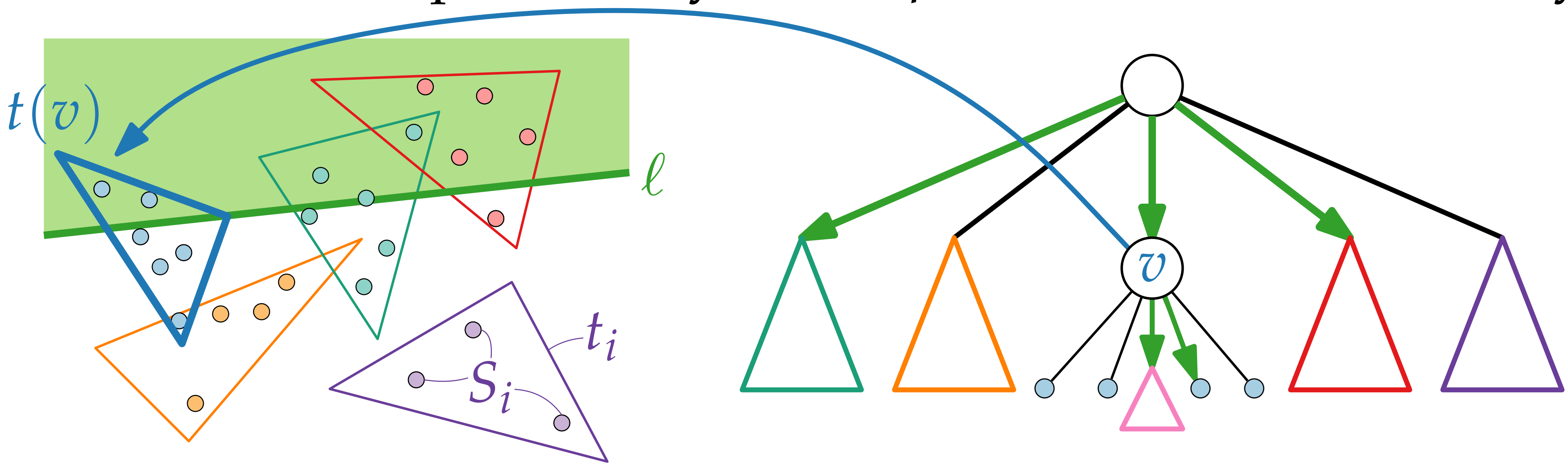
Partition the input! Query… in a *partition tree* …recursively!



Theorem. [Matoušek, DCG 1992] 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

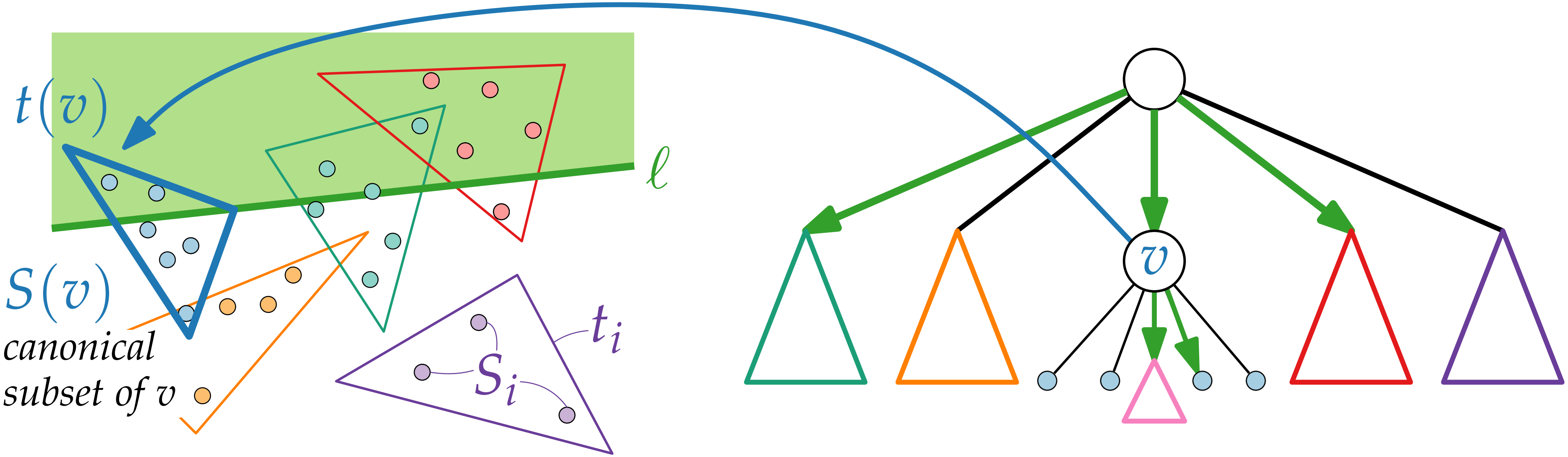
Partition the input! Query… in a *partition tree* …recursively!



Theorem. [Matoušek, DCG 1992] 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

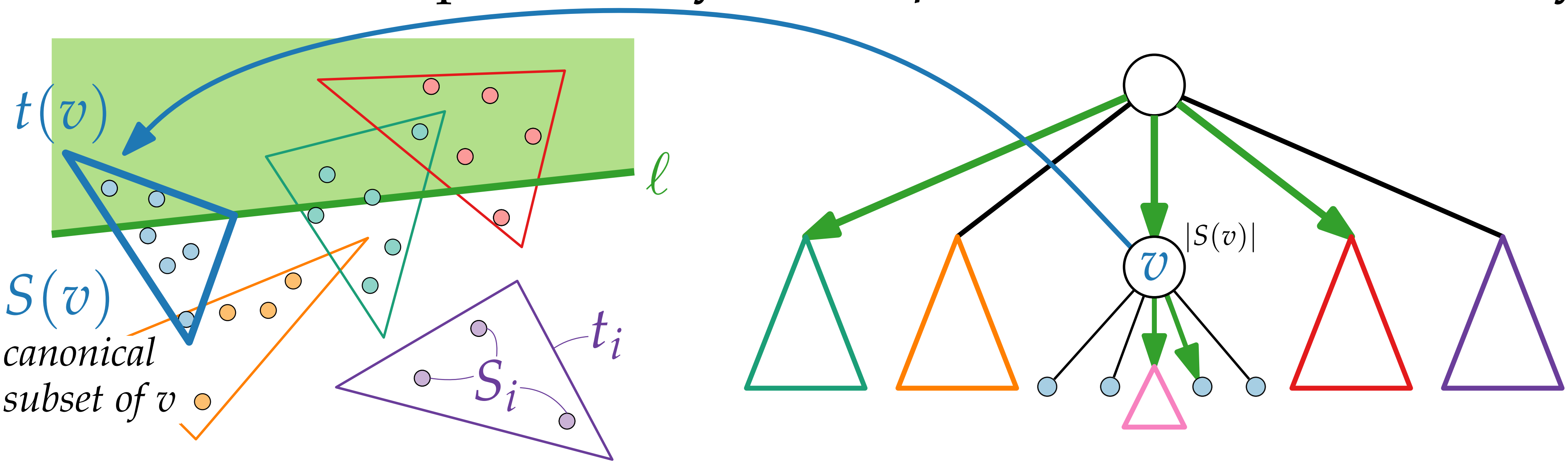
Partition the input! Query… in a *partition tree* …recursively!



Theorem. [Matoušek, DCG 1992] 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

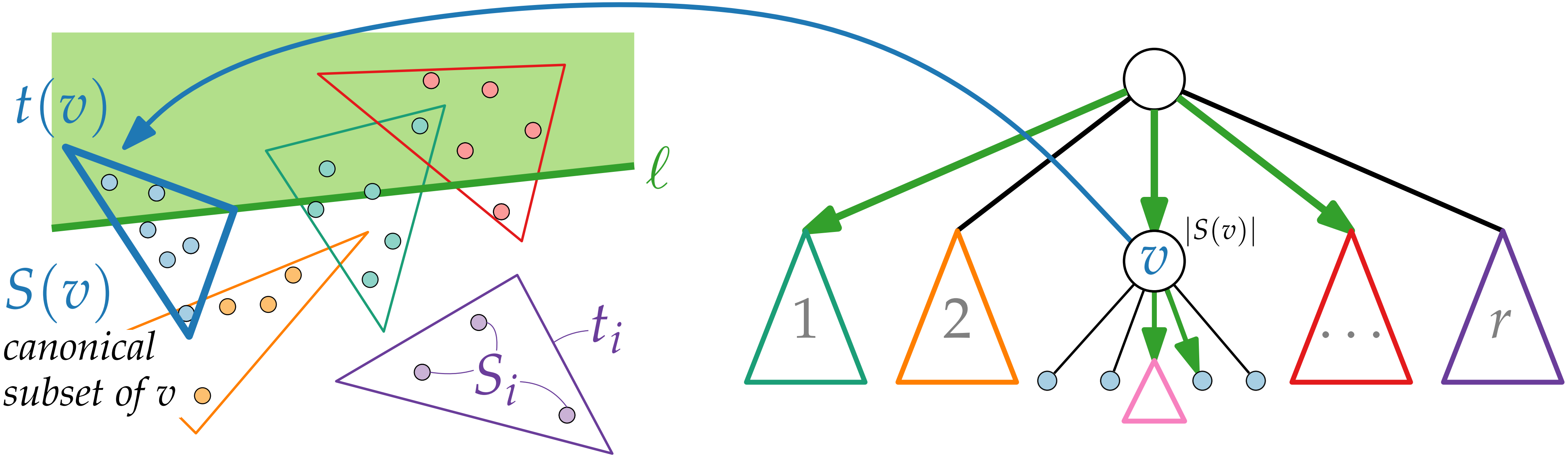
Partition the input! Query… in a *partition tree* …recursively!



Theorem. [Matoušek, DCG 1992] 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

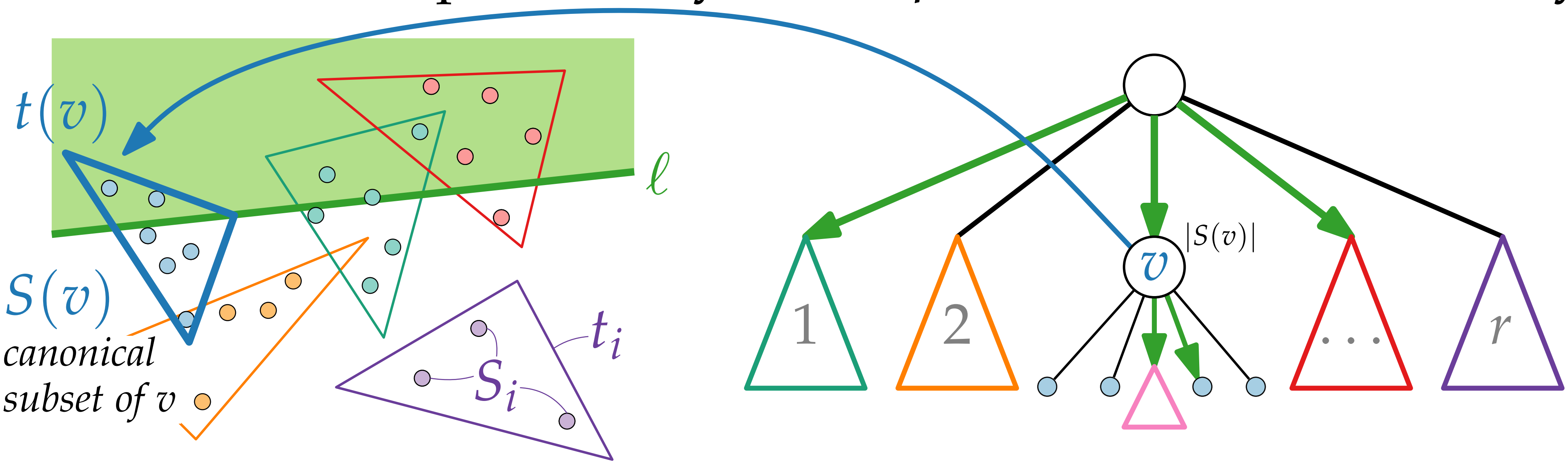
Partition the input! Query... in a *partition tree* ...recursively!



Theorem. [Matoušek, DCG 1992] 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

Partition the input! Query… in a *partition tree* …recursively!

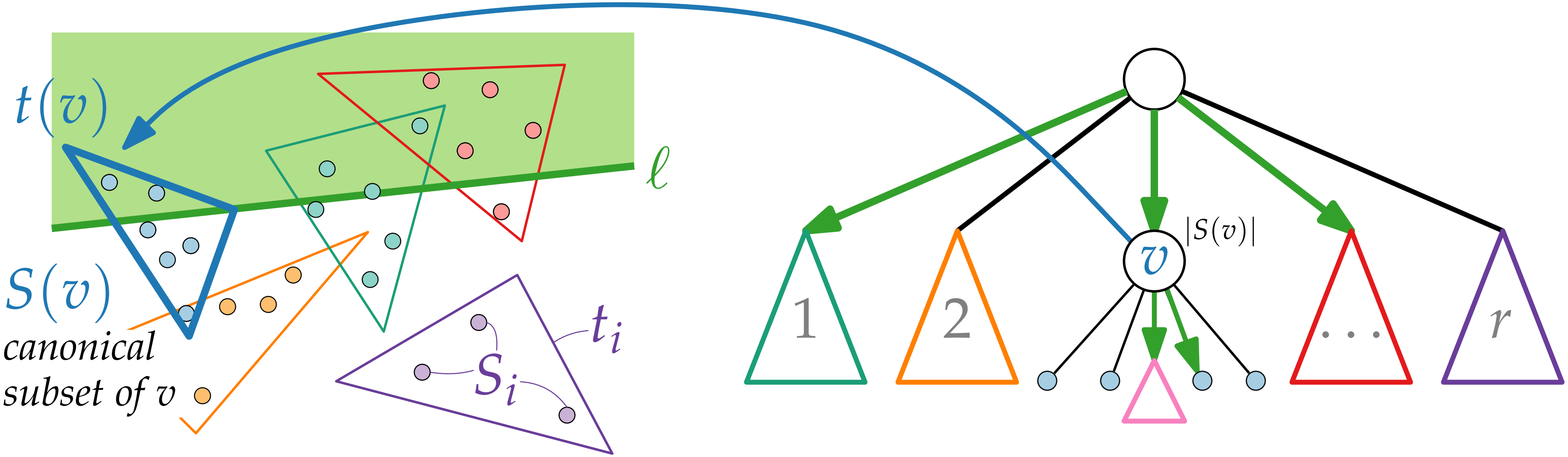


Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Lemma. A partition tree for S can be constructed in $O(n^{1+\varepsilon})$ time. The tree uses $O(n)$ storage.

Generalizing to 2 Dimensions

Partition the input! Query… in a *partition tree* …recursively!

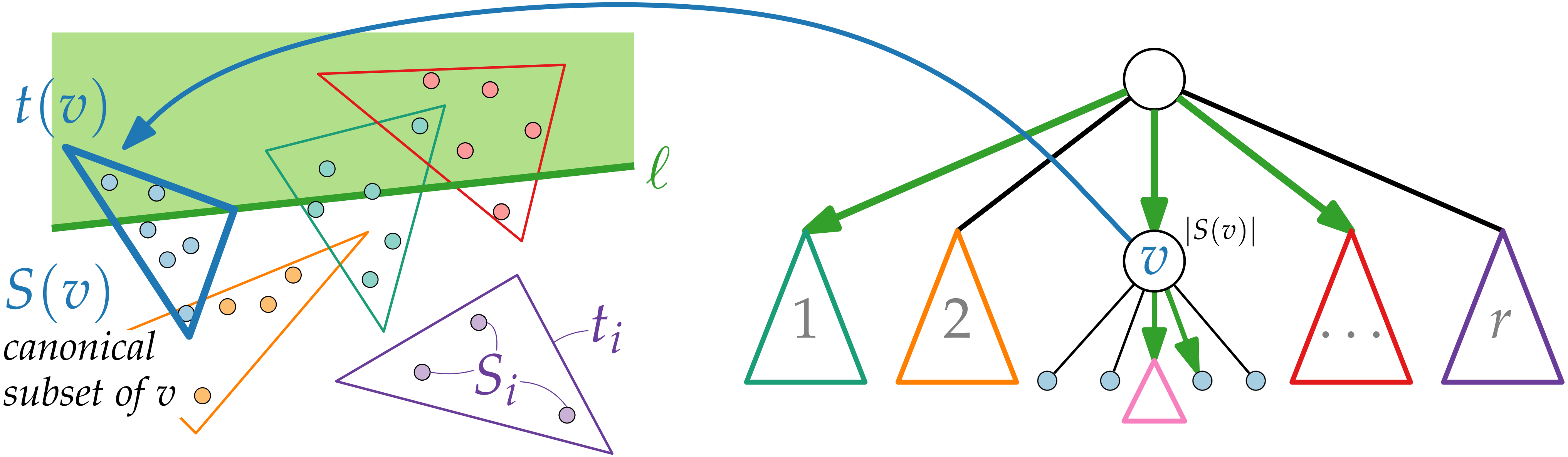


Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \le r \le n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Lemma. A partition tree for S can be constructed in $O(n^{1+\varepsilon})$ time. The tree uses $O(n)$ storage.

Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree* ... recursively!



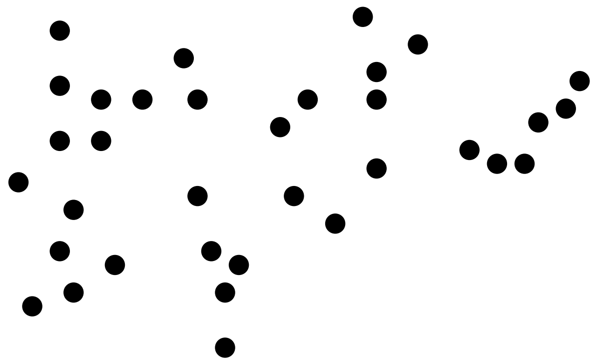
Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $O(\sqrt{r})$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Lemma. A partition tree for S can be constructed in $O(n^{1+\varepsilon})$ time. The tree uses $O(n)$ storage.

search tree with n leaves

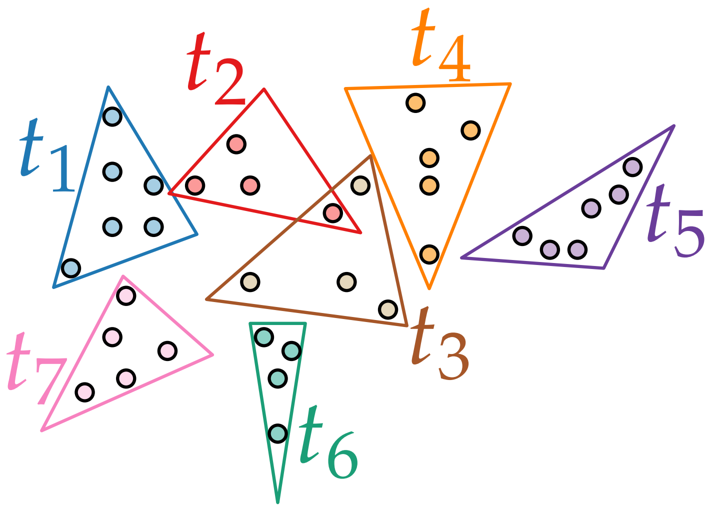
Example for a Query

point set S



Example for a Query

point set S

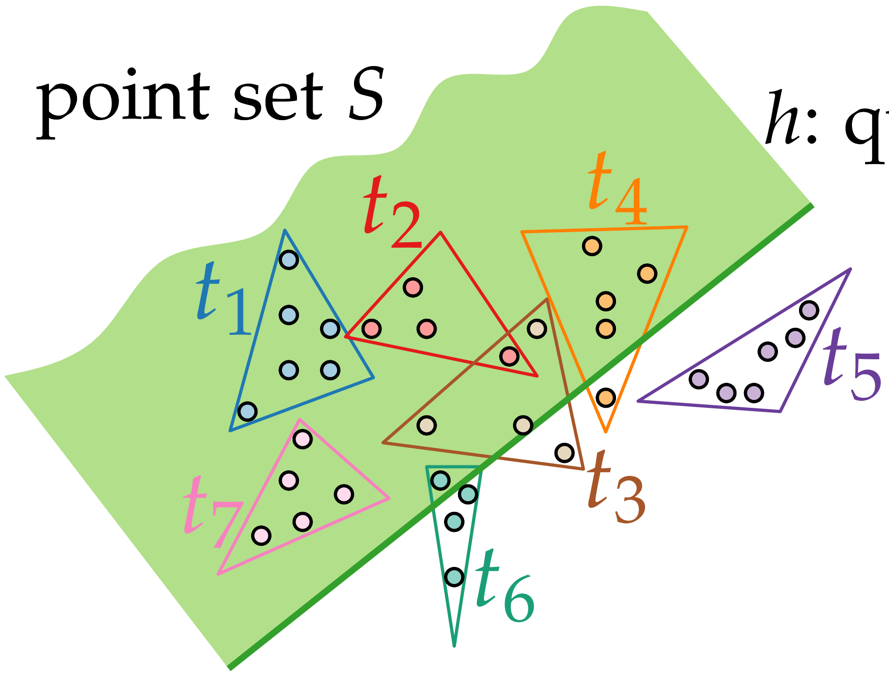


partition by triangles

Example for a Query

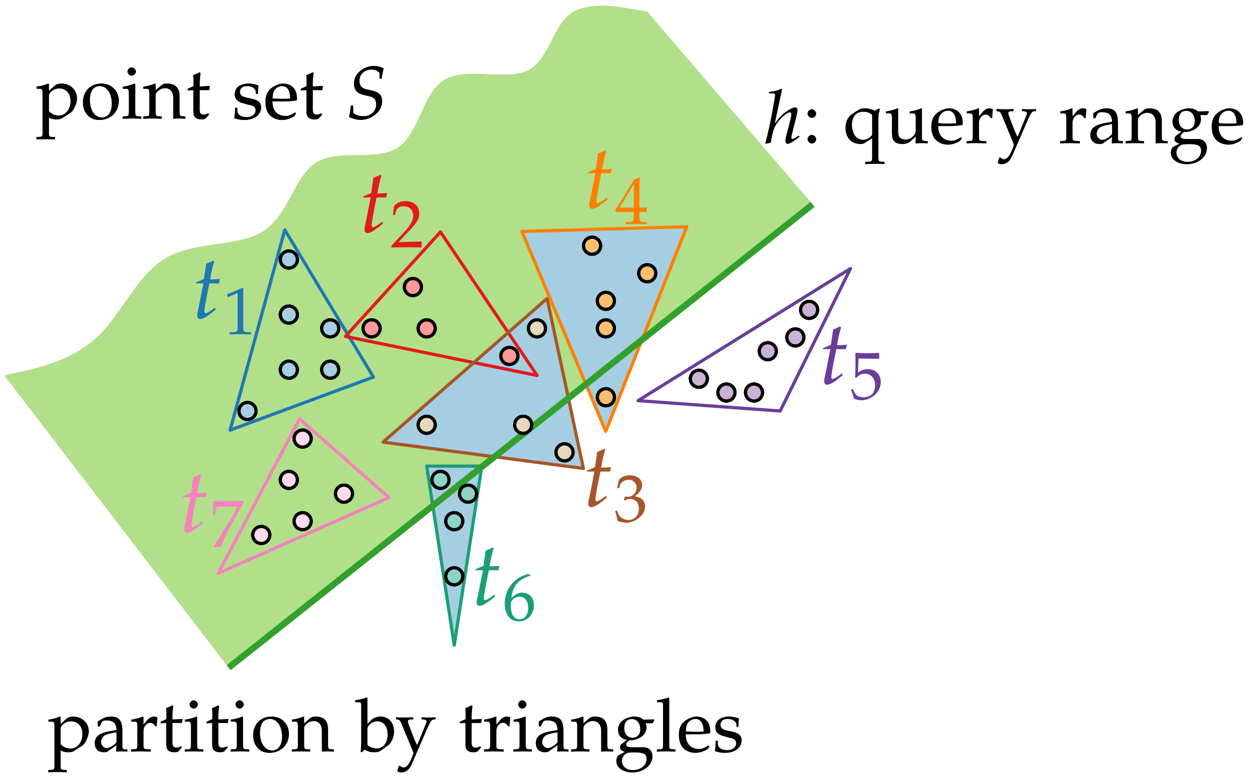
point set S

h: query range

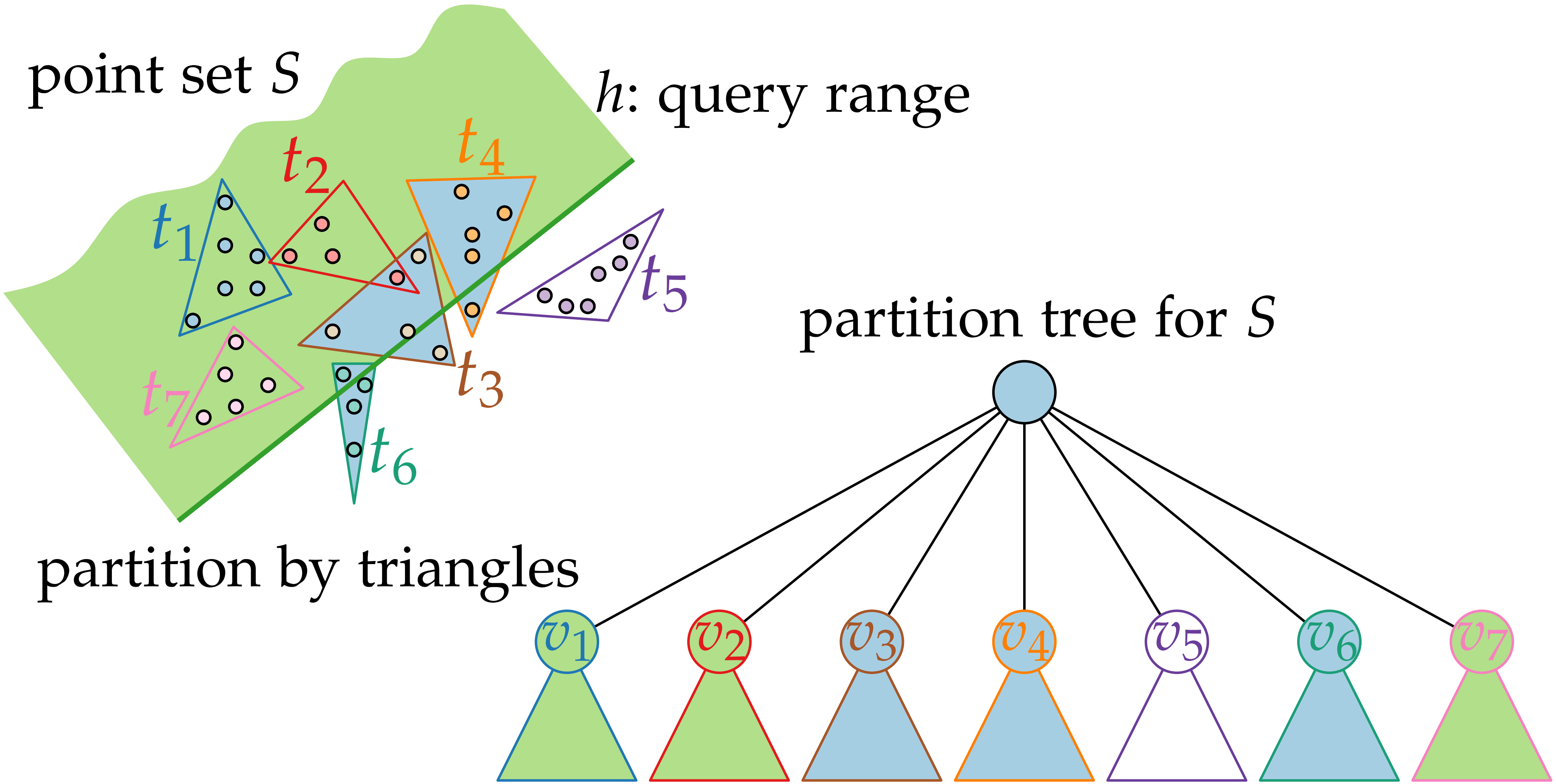


partition by triangles

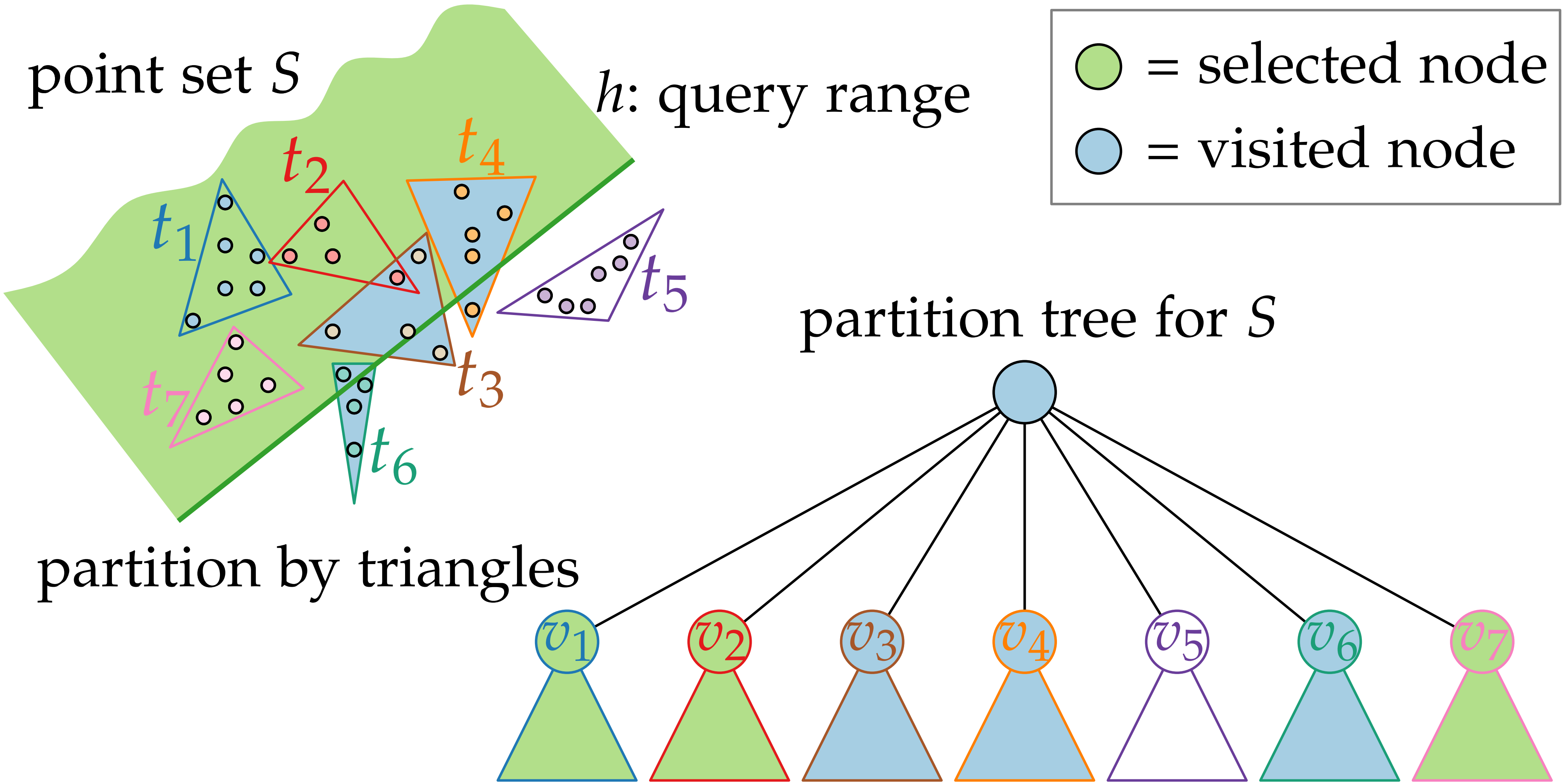
Example for a Query



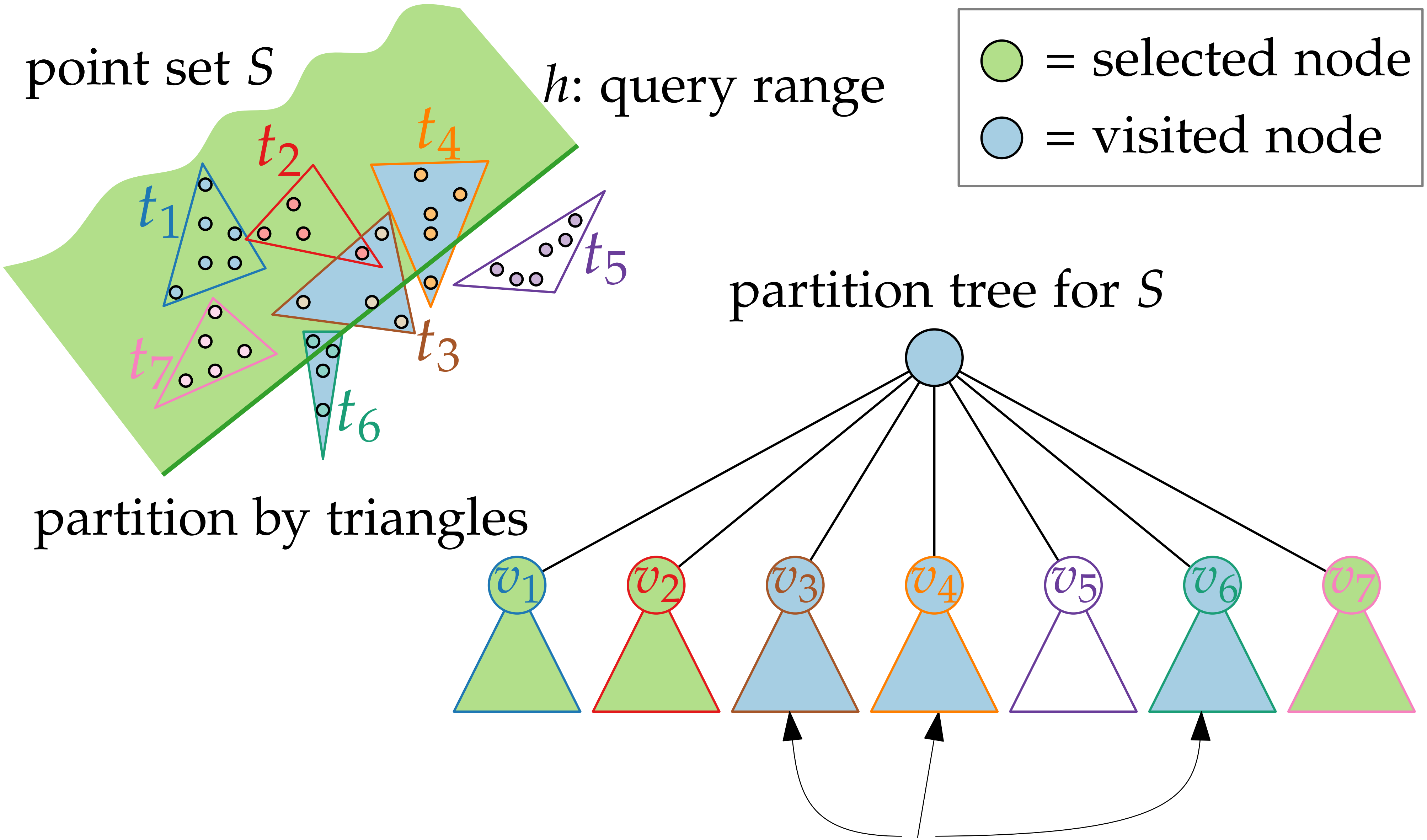
Example for a Query



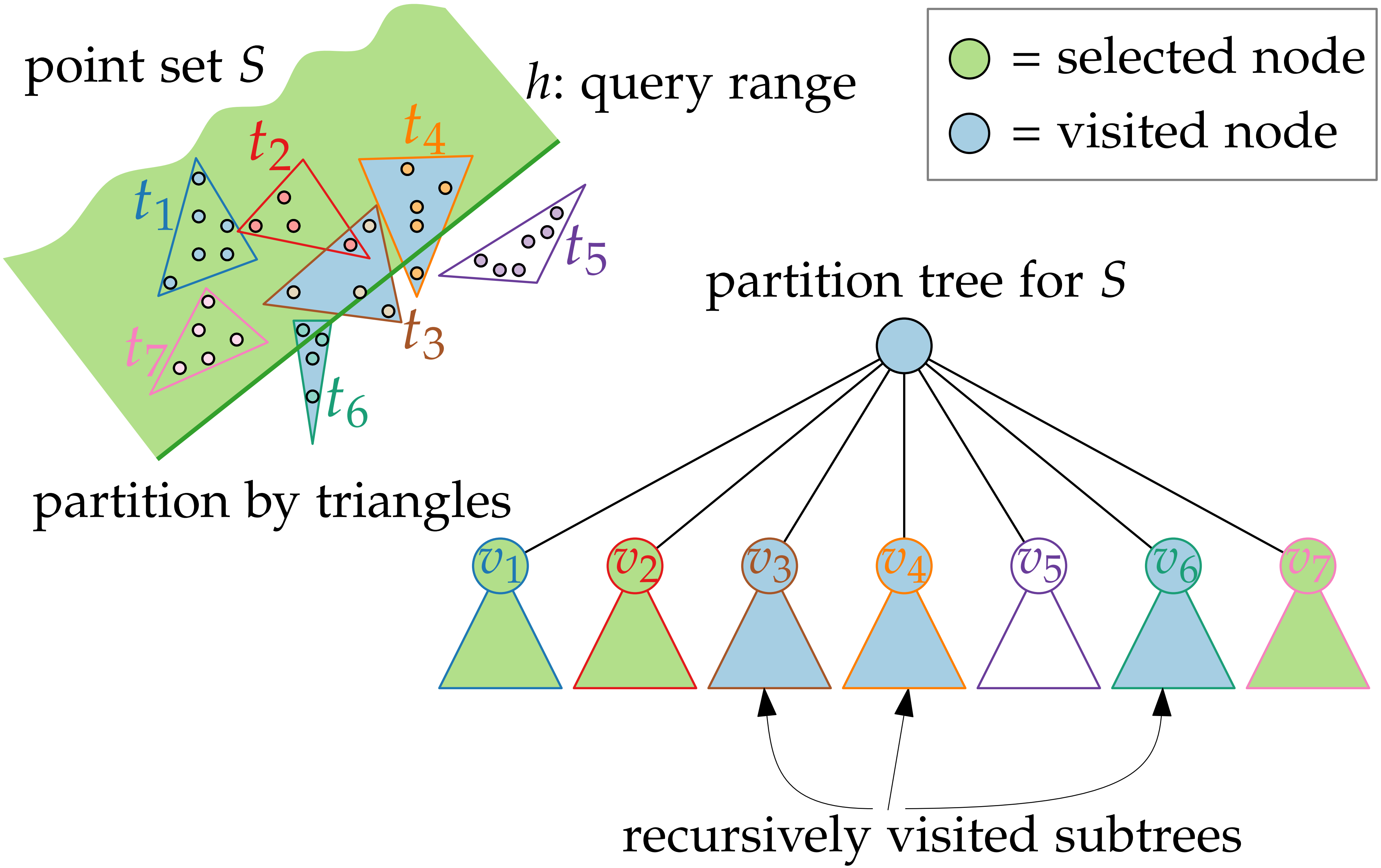
Example for a Query



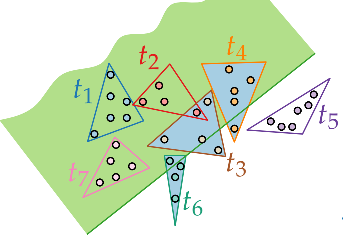
Example for a Query

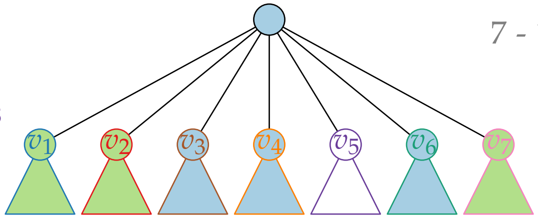


Example for a Query



Query Algorithm

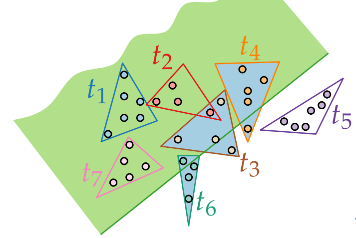


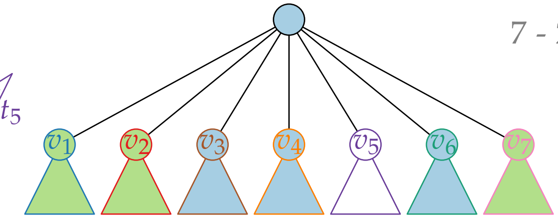


SelectInHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

$N \leftarrow \emptyset$ // set of *selected* nodes

Query Algorithm





SelectInHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

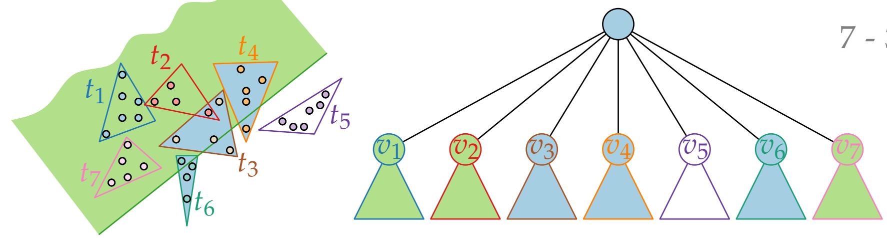
$N \leftarrow \emptyset$ // set of *selected* nodes

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

else

return N // with $S \cap h = \bigcup_{\nu \in N} S(\nu)$

Query Algorithm



SelectInHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

$N \leftarrow \emptyset$ // set of *selected* nodes

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

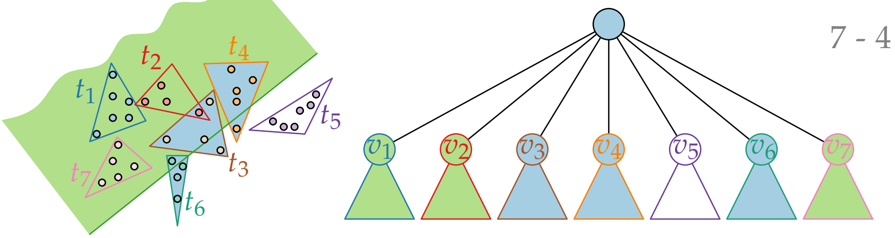
  if point stored at μ lies in h **then**

    $N \leftarrow \{\mu\}$

else

return N // with $S \cap h = \bigcup_{\nu \in N} S(\nu)$

Query Algorithm



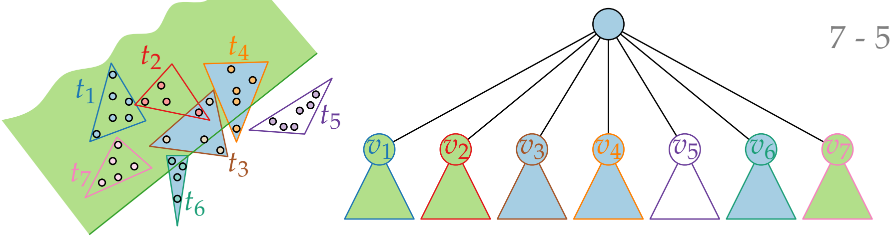
SelectInHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

$N \leftarrow \emptyset$ // set of *selected* nodes

if $\mathcal{T} = \{\mu\}$ **then**
  if point stored at μ lies in h **then**
    $N \leftarrow \{\mu\}$
else
  foreach child v of the root of $\mathcal{T}$ **do**

return N // with $S \cap h = \bigcup_{v \in N} S(v)$

Query Algorithm



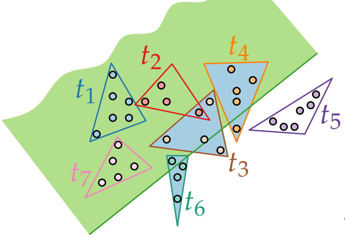
SelectInHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

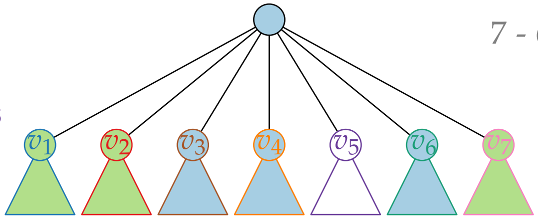
$N \leftarrow \emptyset$ // set of *selected* nodes

if $\mathcal{T} = \{\mu\}$ **then**
    if point stored at μ lies in h **then**
        $N \leftarrow \{\mu\}$
else
    foreach child ν of the root of $\mathcal{T}$ **do**
        if $t(\nu) \subset h$ **then**
        else

return N // with $S \cap h = \bigcup_{\nu \in N} S(\nu)$

Query Algorithm





SelectInHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

$N \leftarrow \emptyset$ // set of *selected* nodes

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

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

    $N \leftarrow \{\mu\}$

else

  foreach child v of the root of $\mathcal{T}$ **do**

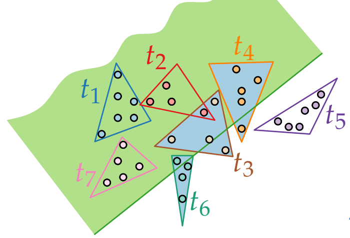
    if $t(v) \subset h$ **then**

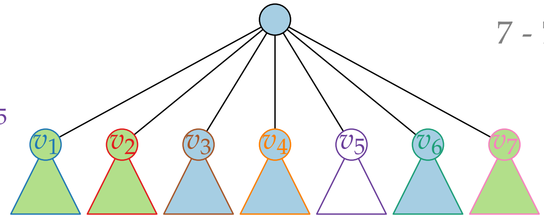
      $N \leftarrow N \cup \{v\}$

    else

return N // with $S \cap h = \bigcup_{v \in N} S(v)$

Query Algorithm





SelectInHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

$N \leftarrow \emptyset$ // set of *selected* nodes

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

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

    $N \leftarrow \{\mu\}$

else

  foreach child v of the root of $\mathcal{T}$ **do**

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

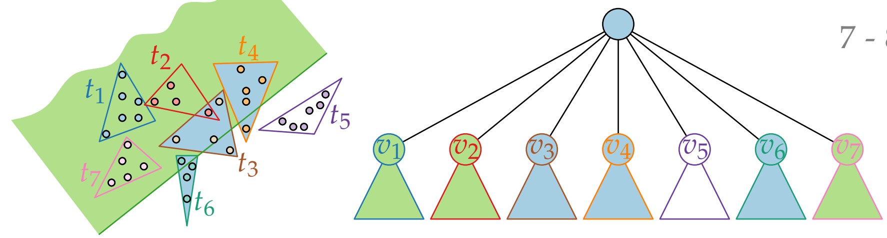
      $N \leftarrow N \cup \{v\}$

    else

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

return N // with $S \cap h = \bigcup_{v \in N} S(v)$

Query Algorithm



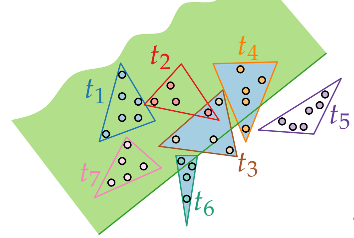
SelectInHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

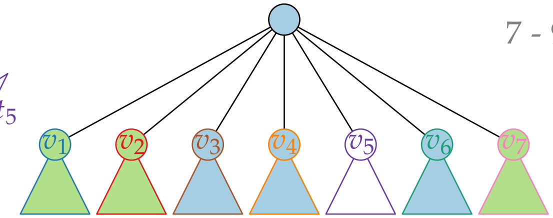
$N \leftarrow \emptyset$ // set of *selected* nodes

if $\mathcal{T} = \{\mu\}$ **then**
  if point stored at μ lies in h **then**
    $N \leftarrow \{\mu\}$
else
  foreach 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**
        $N \leftarrow N \cup$ SelectInHalfplane$(h, \mathcal{T}_\nu)$

return N // with $S \cap h = \bigcup_{\nu \in N} S(\nu)$

Query Algorithm





SelectInHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

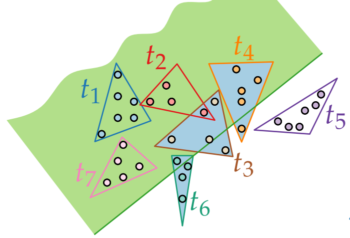
$N \leftarrow \emptyset$ // set of *selected* nodes

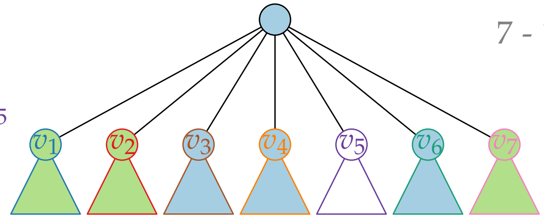
if $\mathcal{T} = \{\mu\}$ **then**
    if point stored at μ lies in h **then**
        $N \leftarrow \{\mu\}$
else
    foreach child v of the root of $\mathcal{T}$ **do**
        if $t(v) \subset h$ **then**
            $N \leftarrow N \cup \{v\}$
        else
            if $t(v) \cap h \neq \emptyset$ **then**
                $N \leftarrow N \cup$ SelectInHalfplane$(h, \mathcal{T}_v)$
return N // with $S \cap h = \bigcup_{v \in N} S(v)$

Task:

Turn this into a range *counting* query algorithm!

Query Algorithm





Count
~~Select~~InHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

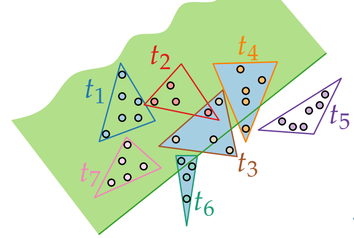
$N \leftarrow \emptyset$ // set of *selected* nodes

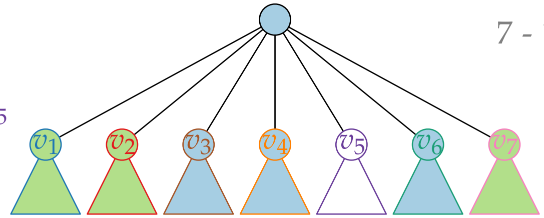
if $\mathcal{T} = \{\mu\}$ **then**
  if point stored at μ lies in h **then**
    $N \leftarrow \{\mu\}$
else
  foreach child ν of the root of $\mathcal{T}$ **do**
    if $t(\nu) \subset h$ **then**
      $N \leftarrow N \cup \{\nu\}$
    else
      if $t(\nu) \cap h \neq \emptyset$ **then**
        $N \leftarrow N \cup$ SelectInHalfplane$(h, \mathcal{T}_\nu)$
return N // with $S \cap h = \bigcup_{\nu \in N} S(\nu)$

Task:

Turn this into a range *counting* query algorithm!

Query Algorithm





COUNT

~~SELECT~~INHALFPLANE(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

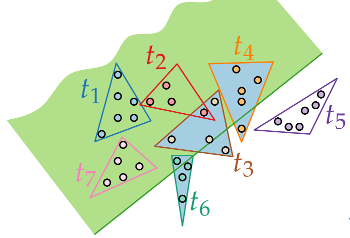
$N \leftarrow \emptyset$ // ~~set~~ number of *selected* nodes

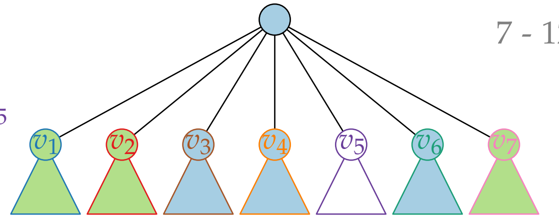
if $\mathcal{T} = \{\mu\}$ **then**
    if point stored at μ lies in h **then**
        $N \leftarrow \{\mu\}$
else
    foreach child v of the root of $\mathcal{T}$ **do**
        if $t(v) \subset h$ **then**
            $N \leftarrow N \cup \{v\}$
        else
            if $t(v) \cap h \neq \emptyset$ **then**
                $N \leftarrow N \cup \textsc{SelectInHalfplane}(h, \mathcal{T}_v)$
return N // with $S \cap h = \bigcup_{v \in N} S(v)$

Task:

Turn this into a range *counting* query algorithm!

Query Algorithm





Count

~~Select~~InHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

$N \leftarrow \cancel{\emptyset}$ 0 // ~~set~~ of *selected* nodes

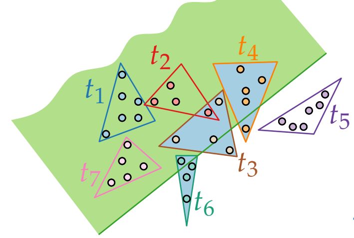
number

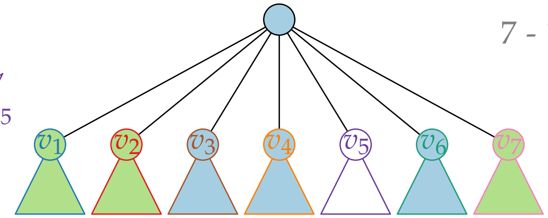
if $\mathcal{T} = \{\mu\}$ **then**
 if point stored at μ lies in h **then**
  $N \leftarrow \{\mu\}$
else
 foreach child ν of the root of $\mathcal{T}$ **do**
  if $t(\nu) \subset h$ **then**
   $N \leftarrow N \cup \{\nu\}$
  else
   if $t(\nu) \cap h \neq \emptyset$ **then**
    $N \leftarrow N \cup$ SelectInHalfplane$(h, \mathcal{T}_\nu)$
return N // with $S \cap h = \bigcup_{\nu \in N} S(\nu)$

Task:

Turn this into a range *counting* query algorithm!

Query Algorithm





Count ~~Select~~InHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

$N \leftarrow \cancel{\emptyset}$ 0 // ~~set~~ number of *selected* nodes

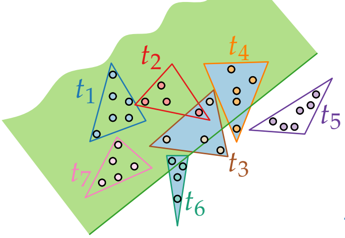
if $\mathcal{T} = \{\mu\}$ **then**
- **if** point stored at μ lies in h **then**
  - $N \leftarrow \cancel{\{\mu\}}$ 1

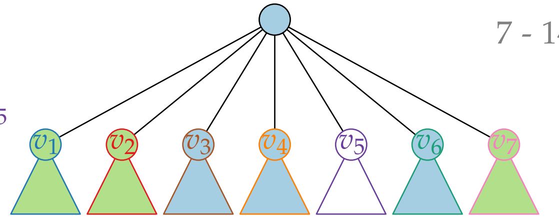
else
- **foreach** child v of the root of $\mathcal{T}$ **do**
  - **if** $t(v) \subset h$ **then**
    - $N \leftarrow N \cup \{v\}$
  - **else**
    - **if** $t(v) \cap h \neq \emptyset$ **then**
      - $N \leftarrow N \cup \textsc{SelectInHalfplane}(h, \mathcal{T}_v)$

return N // with $S \cap h = \bigcup_{v \in N} S(v)$

Task:

Turn this into a range *counting* query algorithm!





Query Algorithm

COUNT

~~SELECT~~INHALFPLANE(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

$N \leftarrow \cancel{\emptyset}$ 0 // ~~set~~ number of *selected* nodes

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

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

    $N \leftarrow \cancel{\{\mu\}}$ 1

else

  foreach child v of the root of $\mathcal{T}$ **do**

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

      $N \leftarrow N \cancel{\cup \{v\}} + |S(v)|$

    else

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

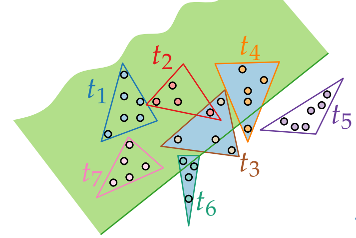
        $N \leftarrow N \cup$ SELECTINHALFPLANE$(h, \mathcal{T}_v)$

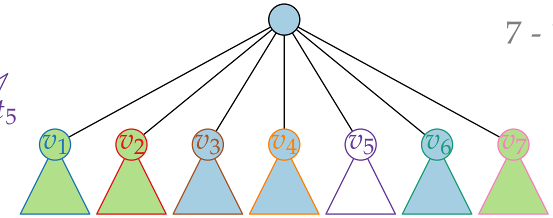
return N // with $S \cap h = \bigcup_{v \in N} S(v)$

Task:

Turn this into a range *counting* query algorithm!

Query Algorithm





COUNT
~~SELECT~~InHalfplane(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

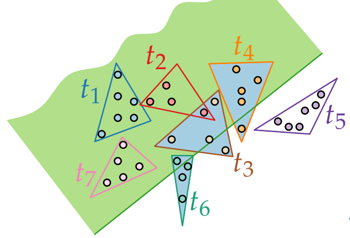
$N \leftarrow \cancel{\emptyset}$ 0   // ~~set~~ of *selected* nodes
number

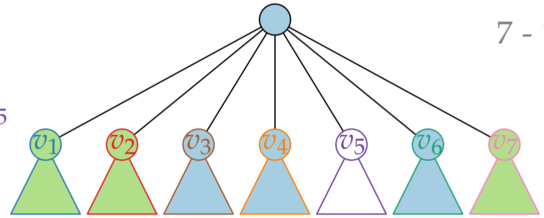
if $\mathcal{T} = \{\mu\}$ **then**
  if point stored at μ lies in h **then**
    $N \leftarrow \cancel{\{\mu\}}$ 1
else
  foreach child v of the root of $\mathcal{T}$ **do**
    if $t(v) \subset h$ **then**
      $N \leftarrow N \cancel{\cup \{v\}} + |S(v)|$
    else
      if $t(v) \cap h \neq \emptyset$ **then**
        $N \leftarrow N \cancel{\cup}$ ~~SELECT~~InHalfplane$(h, \mathcal{T}_v)$
        + COUNT
return N // with $S \cap h = \bigcup_{v \in N} S(v)$

Task:

Turn this into a range *counting* query algorithm!

Query Algorithm





COUNT
~~SELECT~~INHALFPLANE(half-plane h, partit. tree $\mathcal{T}$ for pt set S)

$N \leftarrow \cancel{\emptyset}$ 0 // ~~set~~ of *selected* nodes
number

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

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

    $N \leftarrow \cancel{\{\mu\}}$ 1

else

  foreach child v of the root of $\mathcal{T}$ **do**

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

      $N \leftarrow N \cancel{\cup \{v\}} + |S(v)|$

    else

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

        $N \leftarrow N \cancel{\cup}$ ~~SELECT~~INHALFPLANE$(h, \mathcal{T}_v)$
+ COUNT

return N // with $|S \cap h| = |\bigcup_{v \in N} S(v)|$

Task:

Turn this into a range *counting* query algorithm!

Analysis of the Partition Tree

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

Analysis of the Partition Tree

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

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

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 $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

Proof. Let $\varepsilon > 0$.

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$.

Theorem. [Matoušek, DCG 1992] 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.

Analysis of the Partition Tree

Lemma. For any $\varepsilon > 0$, there is a partition tree $\mathcal{T}$ for S s.t.:
for a query half-plane h,
$\textsc{SelectInHalfplane}$ 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 $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

Proof. Let $\varepsilon > 0$.

Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $c\sqrt{r}$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \le r \le n$, a fine simplicial partition of size r and crossing number $c\sqrt{r}$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ & \text{if } n > 1. \end{cases}$$

Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $c\sqrt{r}$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + & \text{if } n > 1. \end{cases}$$

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 $c\sqrt{r}$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.
[Matoušek, DCG 1992]

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \sum_{v \in C(h)} & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $c\sqrt{r}$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \sum_{v \in C(h)} Q(|S(v)|) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $c\sqrt{r}$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \sum_{v \in C(h)} Q(|S(v)|) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $c\sqrt{r}$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \sum_{v \in C(h)} Q(\overset{2n/r}{\cancel{|S(v)|}}) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \leq r \leq n$, a fine simplicial partition of size r and crossing number $c\sqrt{r}$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \le \begin{cases} 1 & \text{if } n = 1, \\ r + \overset{c\sqrt{r}}{\cancel{\sum_{v \in C(h)}}} Q(|\overset{2n/r}{\cancel{S(v)}}|) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

Theorem. [Matoušek, DCG 1992] For any set S of n pts and any $1 \le r \le n$, a fine simplicial partition of size r and crossing number $c\sqrt{r}$ exists. For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \overbrace{\cancel{\sum_{v \in C(h)}}}^{c\sqrt{r}} Q(|\overbrace{\cancel{S(v)}}^{2n/r}|) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

$$\Rightarrow Q(n) \leq r + c\sqrt{2(\sqrt{2}c)^{1/\varepsilon}} Q(2n/2(\sqrt{2}c)^{1/\varepsilon})$$

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \overset{c\sqrt{r}}{\cancel{\sum_{v \in C(h)}}} Q(\overset{2n/r}{\cancel{|S(v)|}}) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

$$\Rightarrow Q(n) \leq r + c\sqrt{2(\sqrt{2}c)^{1/\varepsilon}}\, Q(2n/2(\sqrt{2}c)^{1/\varepsilon})$$

$$= 2d^{1/\varepsilon} + d^{1+1/2\varepsilon} Q(n/d^{1/\varepsilon}) \quad \text{for } d = \sqrt{2}c$$

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \overset{c\sqrt{r}}{\cancel{\sum_{v \in C(h)}}} Q(\overset{2n/r}{\cancel{|S(v)|}}) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

$$\Rightarrow Q(n) \leq r + c\sqrt{2(\sqrt{2}c)^{1/\varepsilon}}\, Q(2n/2(\sqrt{2}c)^{1/\varepsilon})$$

$$= 2d^{1/\varepsilon} + d^{1+1/2\varepsilon} Q(n/d^{1/\varepsilon}) \quad \text{for } d = \sqrt{2}c$$

Master Theorem: $f(n) \in O(n^{\log_b a - \varepsilon'}) \Rightarrow Q(n) \in O(n^{\log_b a})$

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \overbrace{\cancel{\textstyle\sum_{v \in C(h)}}}^{c\sqrt{r}} Q(|\overbrace{\cancel{S(v)}}^{2n/r}|) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

$$\Rightarrow Q(n) \leq r + c\sqrt{2(\sqrt{2}c)^{1/\varepsilon}}\, Q(2n/2(\sqrt{2}c)^{1/\varepsilon})$$
$$= 2d^{1/\varepsilon} + d^{1+1/2\varepsilon} Q(n/d^{1/\varepsilon}) \quad \text{for } d = \sqrt{2}c$$

Master Theorem: $f(n) \in O(n^{\log_b a - \varepsilon'}) \Rightarrow Q(n) \in O(n^{\log_b a})$

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \le \begin{cases} 1 & \text{if } n = 1, \\ r + \overset{c\sqrt{r}}{\cancel{\sum_{v \in C(h)}}} Q(|\overset{2n/r}{\cancel{S(v)}}|) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

$$\Rightarrow Q(n) \le r + c\sqrt{2(\sqrt{2}c)^{1/\varepsilon}}\, Q(2n/2(\sqrt{2}c)^{1/\varepsilon})$$

$$= 2d^{1/\varepsilon} + d^{1+1/2\varepsilon} Q(n/d^{1/\varepsilon}) \quad \text{for } d = \sqrt{2}c$$

Master Theorem: $f(n) \in O(n^{\log_b a - \varepsilon'}) \Rightarrow Q(n) \in O(n^{\log_b a})$

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \overset{c\sqrt{r}}{\cancel{\sum_{v \in C(h)}}} Q(\overset{2n/r}{\cancel{|S(v)|}}) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

$$\Rightarrow Q(n) \leq r + c\sqrt{2(\sqrt{2}c)^{1/\varepsilon}}\, Q(2n/2(\sqrt{2}c)^{1/\varepsilon})$$

$$= 2d^{1/\varepsilon} + d^{1+1/2\varepsilon} Q(n/d^{1/\varepsilon}) \quad \text{for } d = \sqrt{2}c$$

Master Theorem: $f(n) \in O(n^{\log_b a - \varepsilon'}) \Rightarrow Q(n) \in O(n^{\log_b a})$

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \overbrace{\cancel{\sum_{v \in C(h)}}}^{c\sqrt{r}} Q(\overbrace{\cancel{|S(v)|}}^{2n/r}) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

$$\Rightarrow Q(n) \leq r + c\sqrt{2(\sqrt{2}c)^{1/\varepsilon}}\, Q(2n/2(\sqrt{2}c)^{1/\varepsilon})$$

$$= 2d^{1/\varepsilon} + d^{1+1/2\varepsilon} Q(n/d^{1/\varepsilon}) \quad \text{for } d = \sqrt{2}c$$

Master Theorem: $f(n) \in O(n^{\log_b a - \varepsilon'}) \Rightarrow Q(n) \in O(n^{\log_b a})$

$\log_b a =$

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \overset{c\sqrt{r}}{\cancel{\sum_{v \in C(h)}}} Q(\overset{2n/r}{\cancel{|S(v)|}}) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

$$\Rightarrow Q(n) \leq r + c\sqrt{2(\sqrt{2}c)^{1/\varepsilon}}\, Q(2n/2(\sqrt{2}c)^{1/\varepsilon})$$

$$= 2d^{1/\varepsilon} + d^{1+1/2\varepsilon} Q(n/d^{1/\varepsilon}) \quad \text{for } d = \sqrt{2}c$$

Master Theorem: $f(n) \in O(n^{\log_b a - \varepsilon'}) \Rightarrow Q(n) \in O(n^{\log_b a})$

$$\log_b a = \frac{1+1/2\varepsilon}{1/\varepsilon} =$$

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \overset{c\sqrt{r}}{\cancel{\sum_{v \in C(h)}}} Q(\overset{2n/r}{\cancel{|S(v)|}}) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

$$\Rightarrow Q(n) \leq r + c\sqrt{2(\sqrt{2}c)^{1/\varepsilon}}\, Q(2n/2(\sqrt{2}c)^{1/\varepsilon})$$

$$= 2d^{1/\varepsilon} + d^{1+1/2\varepsilon} Q(n/d^{1/\varepsilon}) \quad \text{for } d = \sqrt{2}c$$

Master Theorem: $f(n) \in O(n^{\log_b a - \varepsilon'}) \Rightarrow Q(n) \in O(n^{\log_b a})$

$$\log_b a = \frac{1 + 1/2\varepsilon}{1/\varepsilon} = \varepsilon + 1/2$$

Analysis of the Partition Tree

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 $h \cap S = \bigcup_{v \in N} S(v)$.

Proof. Let $\varepsilon > 0$. Let $r = 2(\sqrt{2}c)^{1/\varepsilon}$.

$$\Rightarrow Q(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ r + \overset{c\sqrt{r}}{\cancel{\sum_{v \in C(h)}}} Q(\overset{2n/r}{\cancel{|S(v)|}}) & \text{if } n > 1. \end{cases}$$

$C(h)$: all children v of the root s.t. h crosses $t(v)$

$$\Rightarrow Q(n) \leq r + c\sqrt{2(\sqrt{2}c)^{1/\varepsilon}}\, Q(2n/2(\sqrt{2}c)^{1/\varepsilon})$$

$$= 2d^{1/\varepsilon} + d^{1+1/2\varepsilon} Q(n/d^{1/\varepsilon}) \quad \text{for } d = \sqrt{2}c$$

Master Theorem: $f(n) \in O(n^{\log_b a - \varepsilon'}) \Rightarrow Q(n) \in O(n^{\log_b a})$

$$\log_b a = \frac{1+1/2\varepsilon}{1/\varepsilon} = \varepsilon + 1/2 \Rightarrow Q(n) \in O(n^{1/2+\varepsilon}) \quad \square$$

Analysis of the Partition Tree

Lemma. A partition tree for S can be constructed in $O(n^{1+\varepsilon})$ time. The tree uses $O(n)$ storage.

Analysis of the Partition Tree

Lemma. A partition tree for S can be constructed in $O(n^{1+\varepsilon})$ time. The tree 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

Lemma. A partition tree for S can be constructed in $O(n^{1+\varepsilon})$ time. The tree 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
using $O(n)$ space and $O(n^{1+\varepsilon})$ prep.

Back to *Triangular* Range Queries

Any ideas?

Back to *Triangular* Range Queries

Any ideas? Just use SelectInHalfplane!

Back to *Triangular* Range Queries

Any ideas? Just use SELECTINHALFPLANE!

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? Just use SELECTINHALFPLANE!

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? Just use SELECTINHALFPLANE!

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? Just use SELECTINHALFPLANE!

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? Just use SelectInHalfplane!

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? Just use SELECTINHALFPLANE!

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

$|S(v)|$

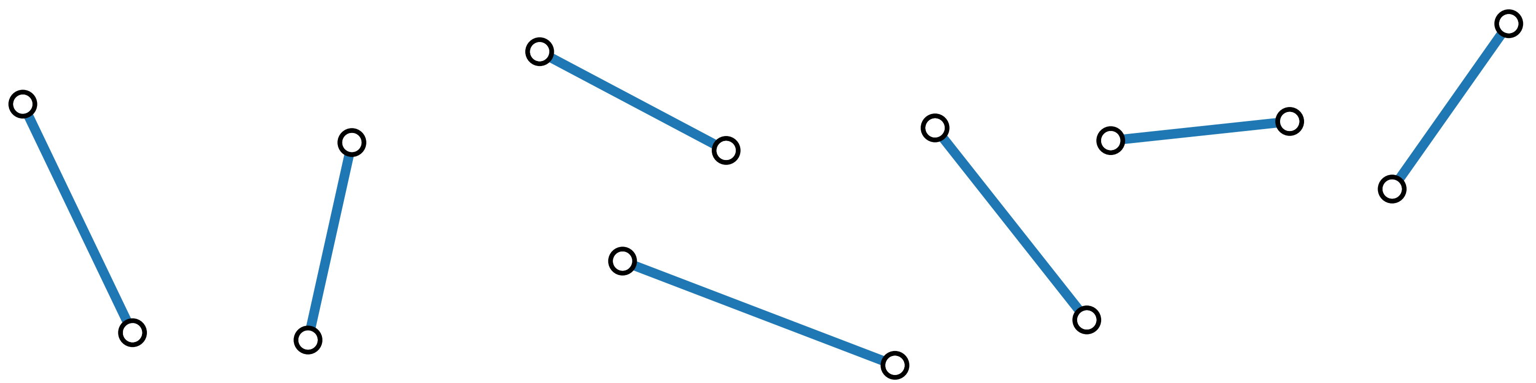
Idea. Store with each internal node not just a number, but another data structure!

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.

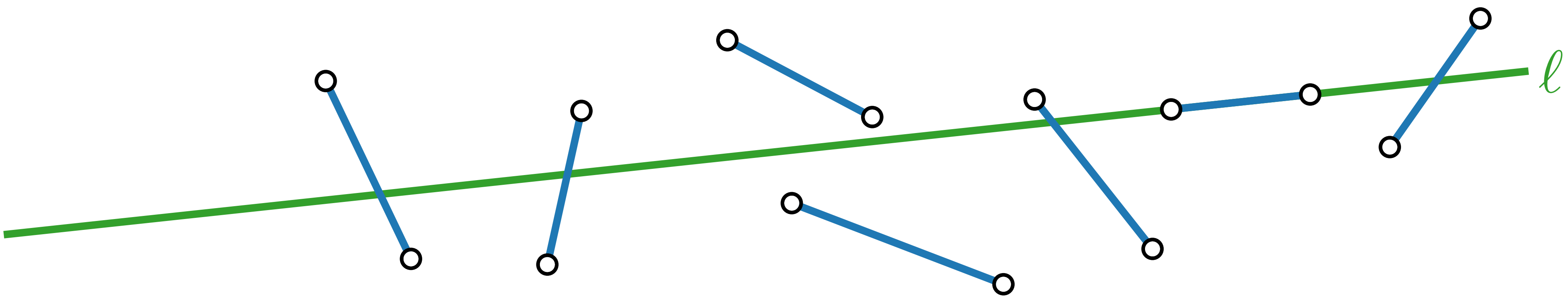


Multi-Level Partition Trees

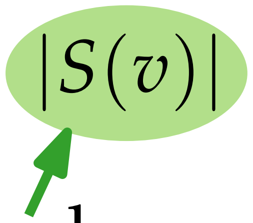
$|S(v)|$

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.



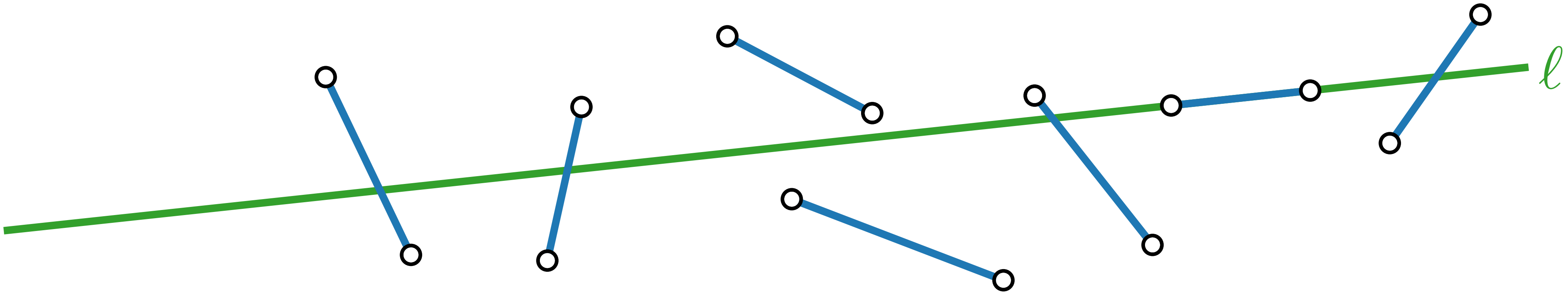
Multi-Level Partition Trees



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:



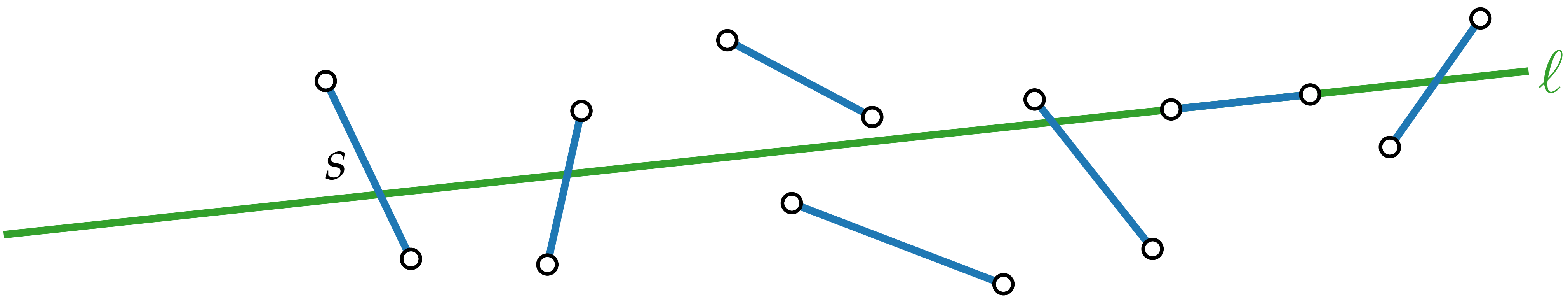
Multi-Level Partition Trees

$|S(v)|$

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:



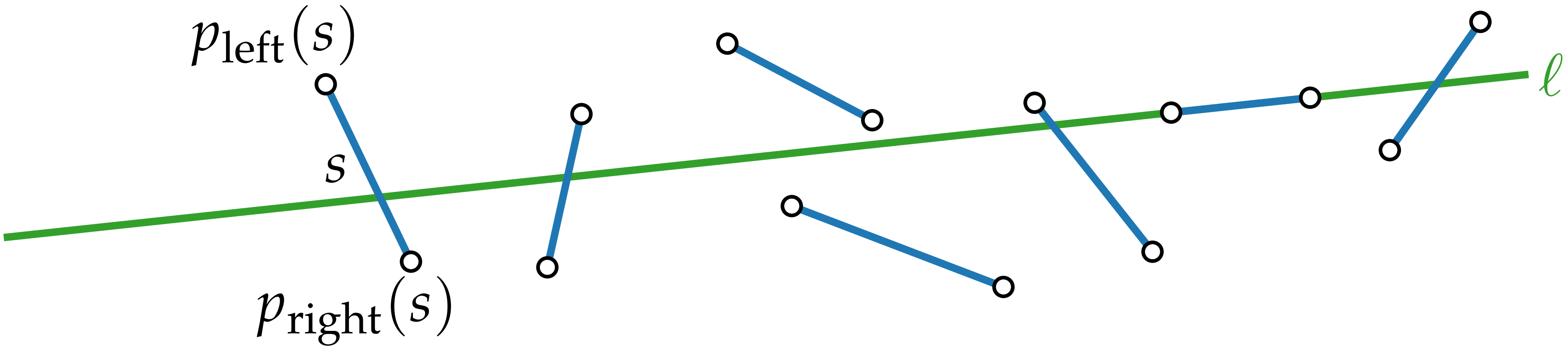
Multi-Level Partition Trees

$|S(v)|$

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:



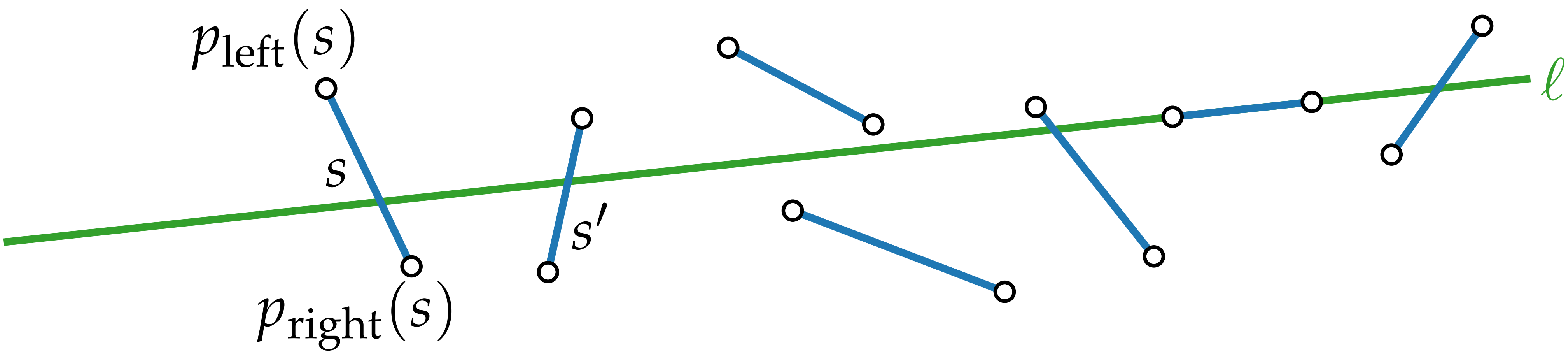
Multi-Level Partition Trees

$|S(v)|$

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:



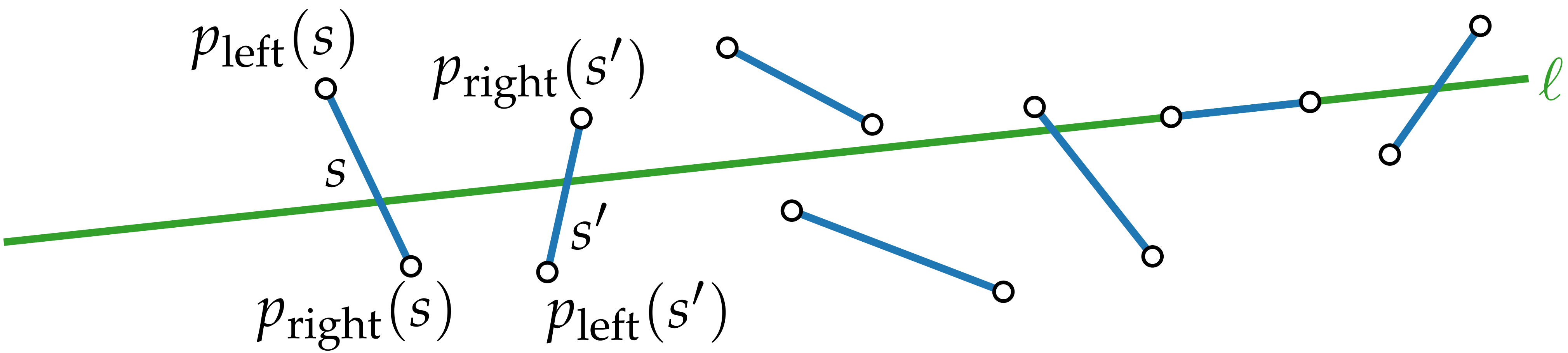
Multi-Level Partition Trees

$|S(v)|$

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:



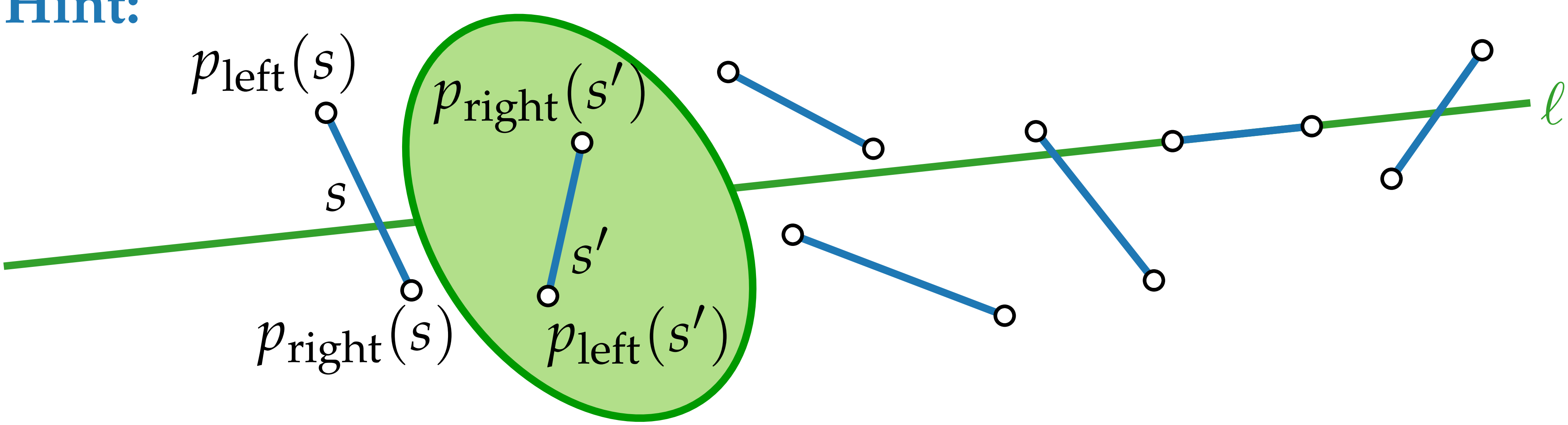
Multi-Level Partition Trees

$|S(v)|$

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:



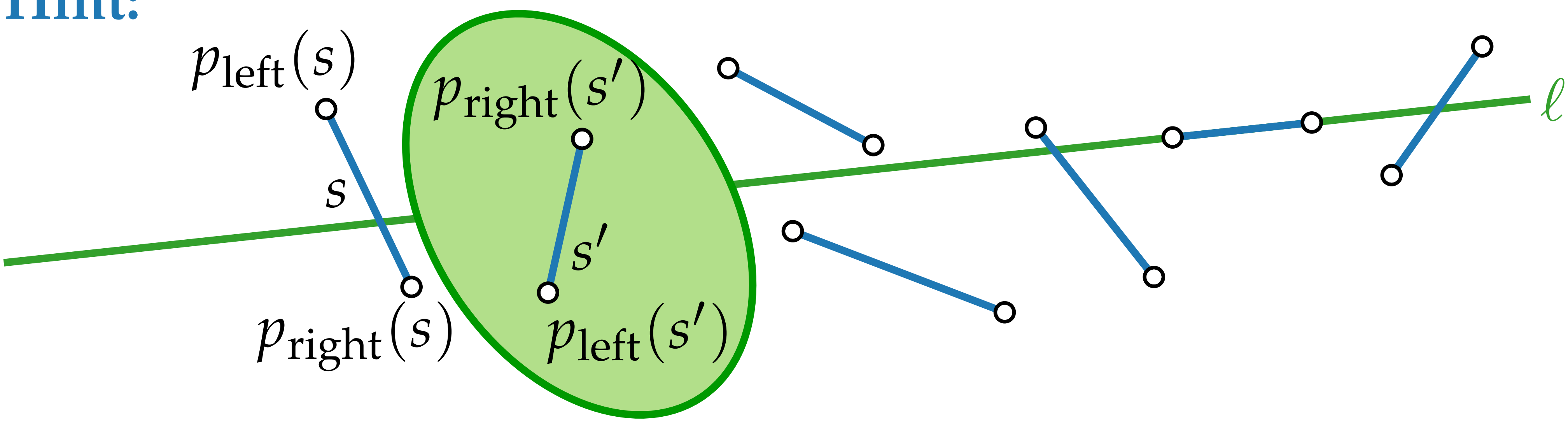
Multi-Level Partition Trees

$|S(v)|$

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:



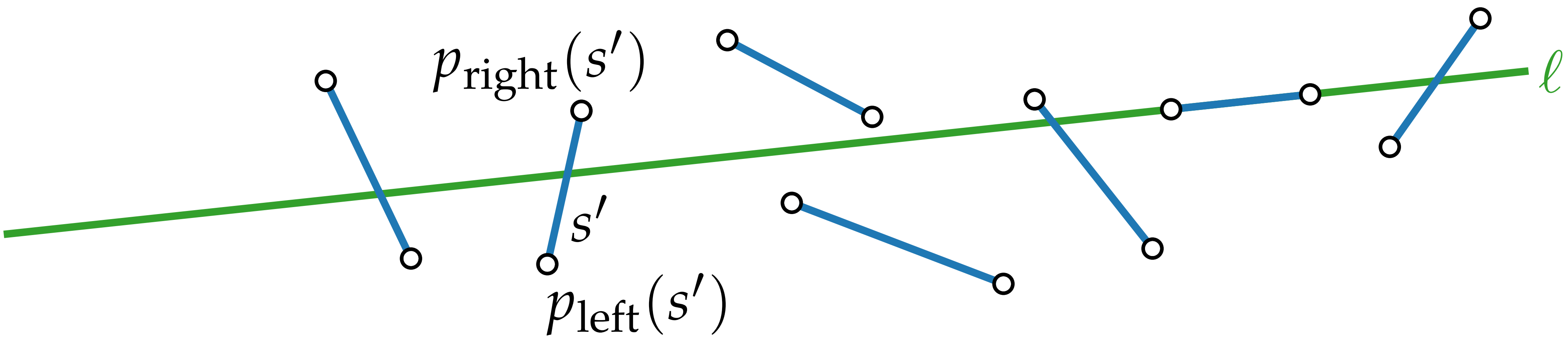
[3 min]

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.

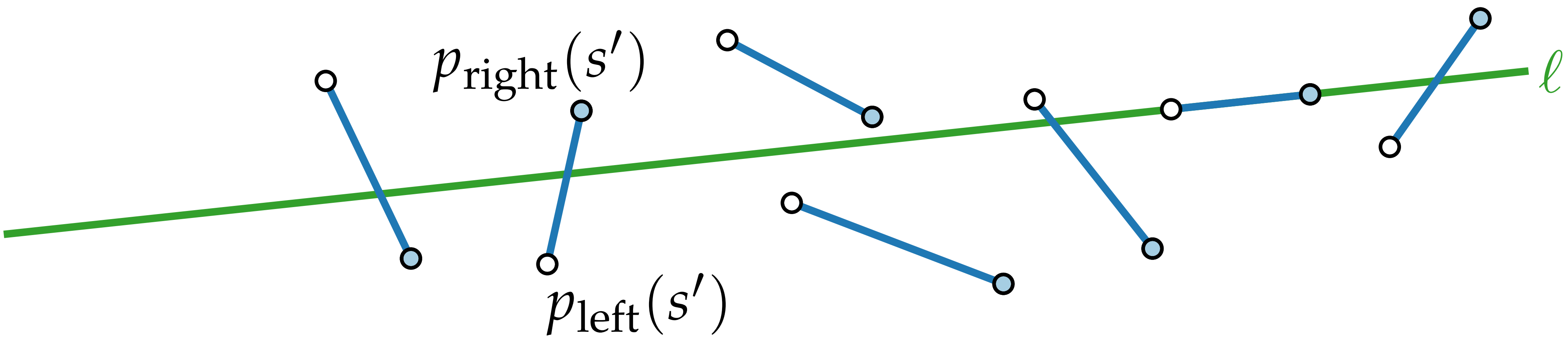


Multi-Level Partition Trees

$|S(v)|$

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.

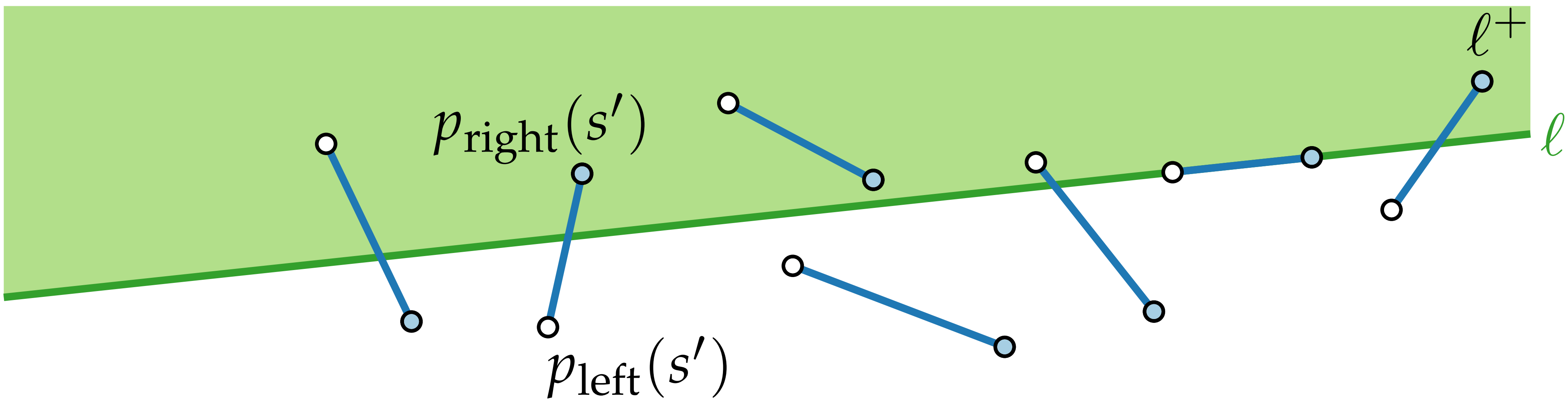


Multi-Level Partition Trees

$|S(v)|$

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.

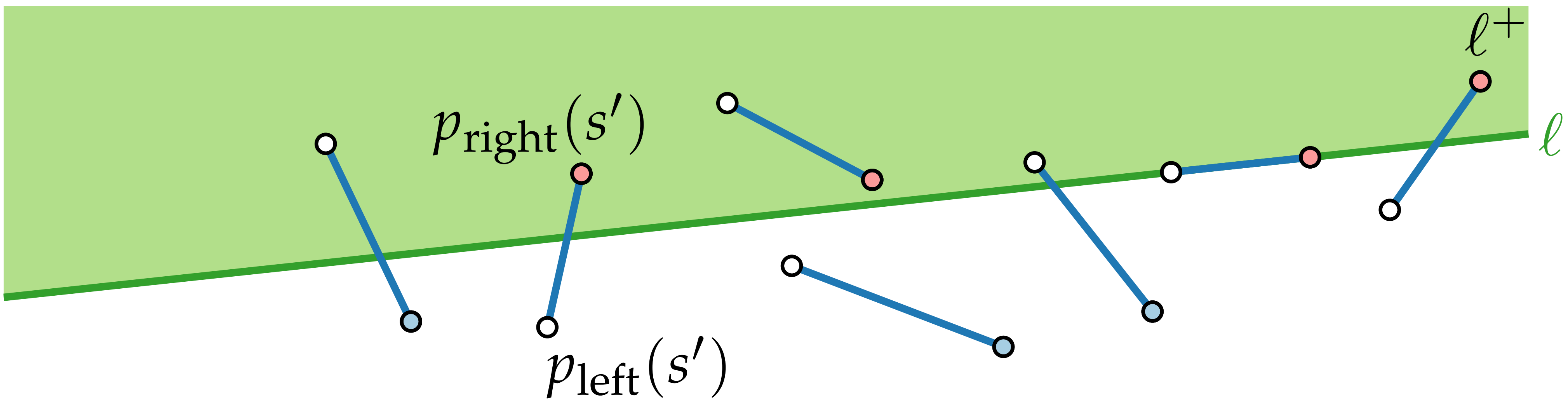


Multi-Level Partition Trees

$|S(v)|$

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.

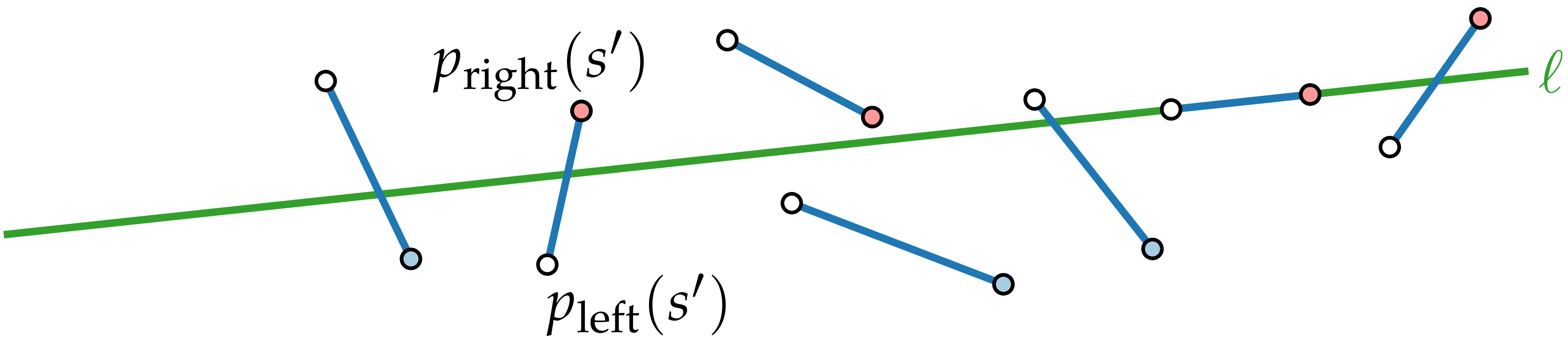


Multi-Level Partition Trees

$|S(v)|$

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.

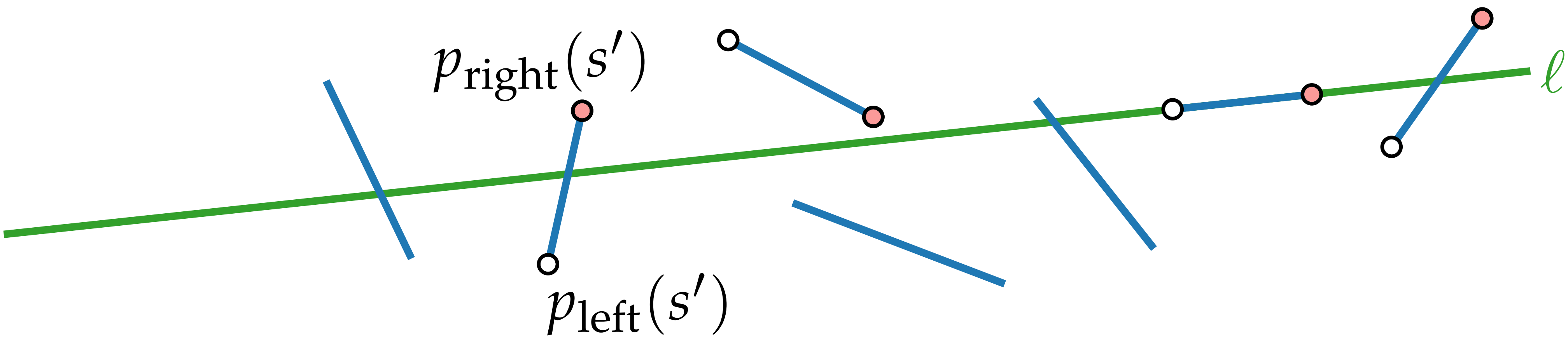


Multi-Level Partition Trees

$|S(v)|$

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.

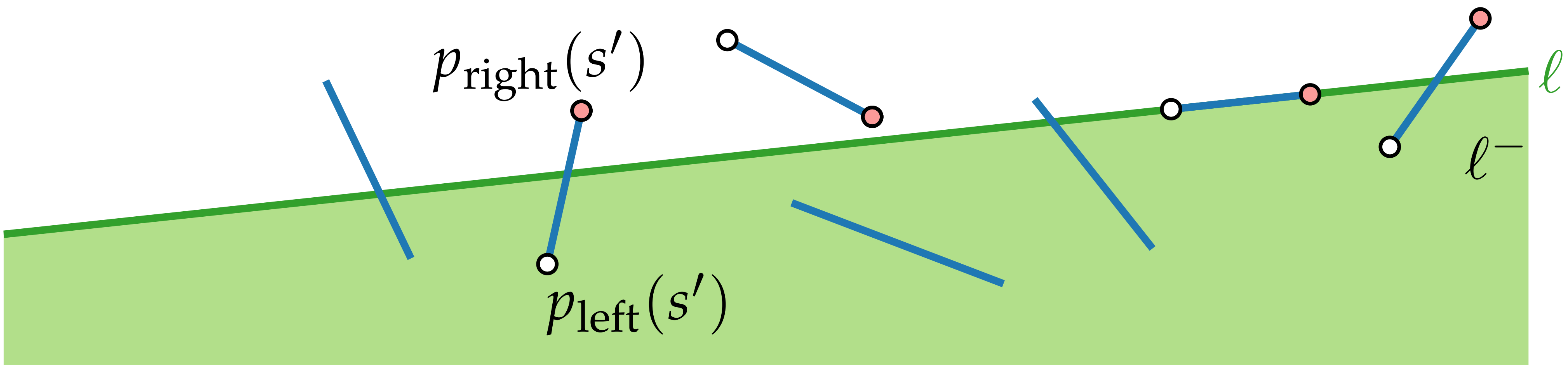


Multi-Level Partition Trees

$|S(v)|$

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.

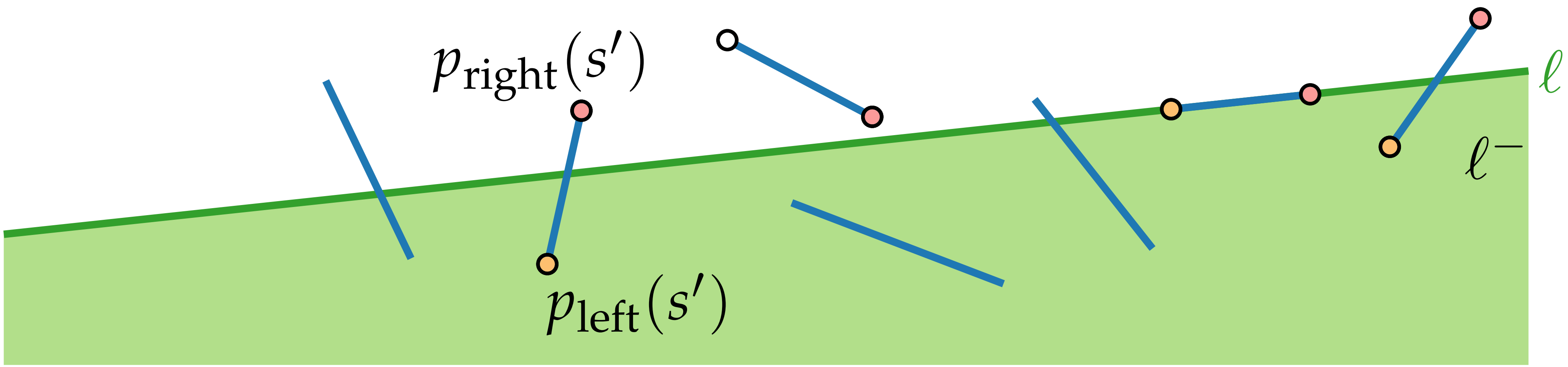


Multi-Level Partition Trees

$|S(v)|$

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.

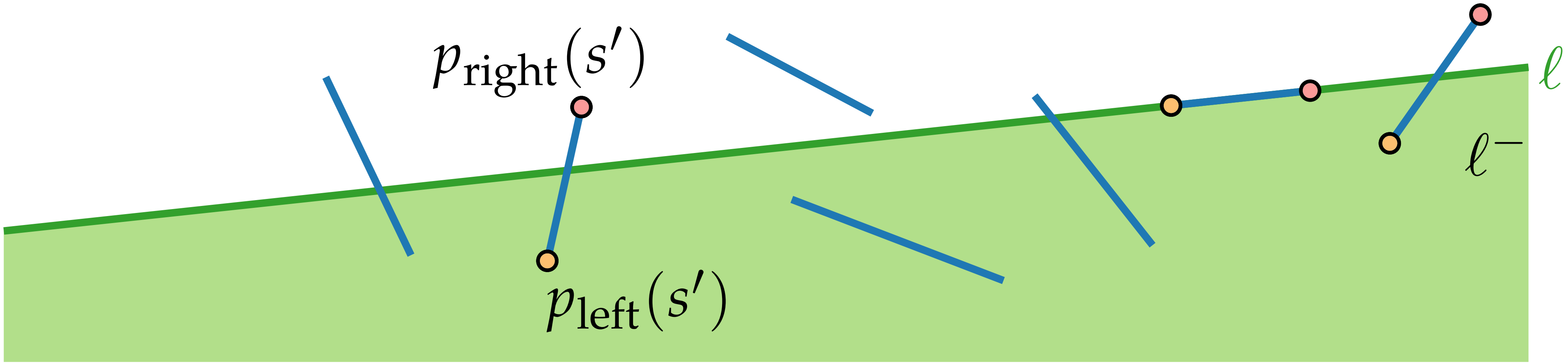


Multi-Level Partition Trees

$|S(v)|$

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.



Query Algorithm

SelectIntSegments(line ℓ, two-level partition tree $\mathcal{T}$ for S)

– first-level tree stores $P_{\text{right}}(S)$
– second-level trees store subsets of $P_{\text{left}}(S)$

$N \leftarrow \emptyset$
if $\mathcal{T} = \{\mu\}$ **then**
  if segment stored in μ intersects ℓ **then** $N \leftarrow \{\mu\}$
else
  foreach child ν of $\mathcal{T}$'s root **do**
    if $t(\nu) \subset \ell^+$ **then**
      $N \leftarrow N \cup \text{SelectInHalfplane}(\ell^-, \mathcal{T}_\nu^{\text{assoc}})$
    else
      if $t(\nu) \cap \ell \neq \emptyset$ **then**
        $N \leftarrow N \cup \text{SelectIntSegments}(\ell, \mathcal{T}_\nu)$
return N

Query Algorithm

For $S' \subseteq S$, let
$P_{\text{right}}^{\text{left}}(S') = \{p_{\text{right}}^{\text{left}}(s) \mid s \in S'\}$

SelectIntSegments(line ℓ, two-level partition tree $\mathcal{T}$ for S)

– first-level tree stores $P_{\text{right}}(S)$
– second-level trees store subsets of $P_{\text{left}}(S)$

```
N ← ∅
if 𝒯 = {μ} then
    if segment stored in μ intersects ℓ then N ← {μ}
else
    foreach child v of 𝒯's root do
        if t(v) ⊂ ℓ⁺ then
            N ← N ∪ SelectInHalfplane(ℓ⁻, 𝒯_v^assoc)
        else
            if t(v) ∩ ℓ ≠ ∅ then
                N ← N ∪ SelectIntSegments(ℓ, 𝒯_v)
return N
```

stores $P_{\text{left}}(S_{\text{seg}}(v))$, where
$S_{\text{seg}}(v) = \{s \mid p_{\text{right}}(s) \in S(v)\}$

Query Algorithm

For $S' \subseteq S$, let
$P^{\text{left}}_{\text{right}}(S') = \{p^{\text{left}}_{\text{right}}(s) \mid s \in S'\}$

SelectIntSegments(line ℓ, two-level partition tree $\mathcal{T}$ for S)

– first-level tree stores $P_{\text{right}}(S)$
– second-level trees store subsets of $P_{\text{left}}(S)$

$N \leftarrow \emptyset$
if $\mathcal{T} = \{\mu\}$ **then**
  if segment stored in μ intersects ℓ **then** $N \leftarrow \{\mu\}$
else
  foreach child ν of $\mathcal{T}$'s root **do**
    if $t(\nu) \subset \ell^+$ **then**
      $N \leftarrow N \cup \text{SelectInHalfplane}(\ell^-, \mathcal{T}_\nu^{\text{assoc}})$
    else
      if $t(\nu) \cap \ell \neq \emptyset$ **then**
        $N \leftarrow N \cup \text{SelectIntSegments}(\ell, \mathcal{T}_\nu)$
return N

stores $P_{\text{left}}(S_{\text{seg}}(\nu))$, where
$S_{\text{seg}}(\nu) = \{s \mid p_{\text{right}}(s) \in S(\nu)\}$

!!! $\bigcup_{\nu \in N} S(\nu) = \{s \in S \mid p_{\text{right}}(s) \text{ above } \ell \text{ and } p_{\text{left}}(s) \text{ below } \ell\}$.

Query Algorithm

For $S' \subseteq S$, let
$P_{\text{right}}^{\text{left}}(S') = \{p_{\text{right}}^{\text{left}}(s) \mid s \in S'\}$

SelectIntSegments(line ℓ, two-level partition tree $\mathcal{T}$ for S)

– first-level tree stores $P_{\text{right}}(S)$
– second-level trees store subsets of $P_{\text{left}}(S)$

$N \leftarrow \emptyset$
if $\mathcal{T} = \{\mu\}$ **then**
  if segment stored in μ intersects ℓ **then** $N \leftarrow \{\mu\}$
else
  foreach child v of $\mathcal{T}$'s root **do**
    if $t(v) \subset \ell^+$ **then**
      $N \leftarrow N \cup \text{SelectInHalfplane}(\ell^-, \mathcal{T}_v^{\text{assoc}})$
    else
      if $t(v) \cap \ell \neq \emptyset$ **then**
        $N \leftarrow N \cup \text{SelectIntSegments}(\ell, \mathcal{T}_v)$
return N

stores $P_{\text{left}}(S_{\text{seg}}(v))$, where
$S_{\text{seg}}(v) = \{s \mid p_{\text{right}}(s) \in S(v)\}$

below above ?

!!! $\bigcup_{v \in N} S(v) = \{s \in S \mid p_{\text{right}}(s)$ above ℓ and $p_{\text{left}}(s)$ below $\ell\}$.

Query Algorithm

For $S' \subseteq S$, let
$P_{\text{right}}^{\text{left}}(S') = \{p_{\text{right}}^{\text{left}}(s) \mid s \in S'\}$

SelectIntSegments(line ℓ, two-level partition tree $\mathcal{T}$ for S)

– first-level tree stores $P_{\text{right}}(S)$
– second-level trees store subsets of $P_{\text{left}}(S)$

$N \leftarrow \emptyset$
if $\mathcal{T} = \{\mu\}$ **then**
  if segment stored in μ intersects ℓ **then** $N \leftarrow \{\mu\}$
else
  foreach child v of $\mathcal{T}$'s root **do**
    if $t(v) \subset \ell^{+}$ **then**
      $N \leftarrow N \cup \text{SelectInHalfplane}(\ell^{-}, \mathcal{T}_v^{\text{assoc}})$
    else
      if $t(v) \cap \ell \neq \emptyset$ **then**
        $N \leftarrow N \cup \text{SelectIntSegments}(\ell, \mathcal{T}_v)$
return N

stores $P_{\text{left}}(S_{\text{seg}}(v))$, where
$S_{\text{seg}}(v) = \{s \mid p_{\text{right}}(s) \in S(v)\}$

below above ?

!!! $\bigcup_{v \in N} S(v) = \{s \in S \mid p_{\text{right}}(s)$ above ℓ and $p_{\text{left}}(s)$ below $\ell\}$.

Query Algorithm

For $S' \subseteq S$, let
$P_{\text{right}}^{\text{left}}(S') = \{p_{\text{right}}^{\text{left}}(s) \mid s \in S'\}$

SelectIntSegments(line ℓ, two-level partition tree $\mathcal{T}$ for S)

– first-level tree stores $P_{\text{right}}(S)$
– second-level trees store subsets of $P_{\text{left}}(S)$

$N \leftarrow \emptyset$
if $\mathcal{T} = \{\mu\}$ **then**
    if segment stored in μ intersects ℓ **then** $N \leftarrow \{\mu\}$
else
    foreach child ν of $\mathcal{T}$'s root **do**
        if $t(\nu) \subset \ell^{+}$ **then**
            $N \leftarrow N \cup \text{SelectInHalfplane}(\ell^{-}, \mathcal{T}_{\nu}^{\text{assoc}})$
        else
            if $t(\nu) \cap \ell \neq \emptyset$ **then**
                $N \leftarrow N \cup \text{SelectIntSegments}(\ell, \mathcal{T}_{\nu})$
return N

stores $P_{\text{left}}(S_{\text{seg}}(\nu))$, where
$S_{\text{seg}}(\nu) = \{s \mid p_{\text{right}}(s) \in S(\nu)\}$

below above ?

!!! $\bigcup_{\nu \in N} S(\nu) = \{s \in S \mid p_{\text{right}}(s)$ above ℓ and $p_{\text{left}}(s)$ below $\ell\}$.

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. We can count the number of segments in S intersected by a query line in $O(n^{1/2+\varepsilon})$ time using $O(n \log \quad n)$ space and $O(\quad)$ prep.

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. We can count the number of segments in S intersected by a query line in $O(n^{1/2+\varepsilon})$ time using $O(n \log n)$ space and $O(n^{1+\varepsilon})$ prep.

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.

δ-level objects

Corollary. Let S be a set of n ~~segments~~ in the plane. We can count the number of ~~segments~~ in S in a δ-level query ~~intersected by a query line~~ in $O(n^{1/2+\varepsilon})$ time using $O(n \log n)$ space and $O(n^{1+\varepsilon})$ prep.

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.

δ-level objects

Corollary. Let S be a set of n ~~segments~~ in the plane.
We can count the number of ~~segments~~ in S
in a δ-level ~~intersected by a query line~~ in $O(n^{1/2+\delta\varepsilon})$ time
query
using $O(n \log^{\delta-1} n)$ space and $O(n^{1+\delta\varepsilon})$ prep.