CS468, Mon. Oct. 2nd, 2006

Quadtrees: Hierarchical Grids

Steve Oudot

From S. Har-Peled's notes, Chapter 2

Outline

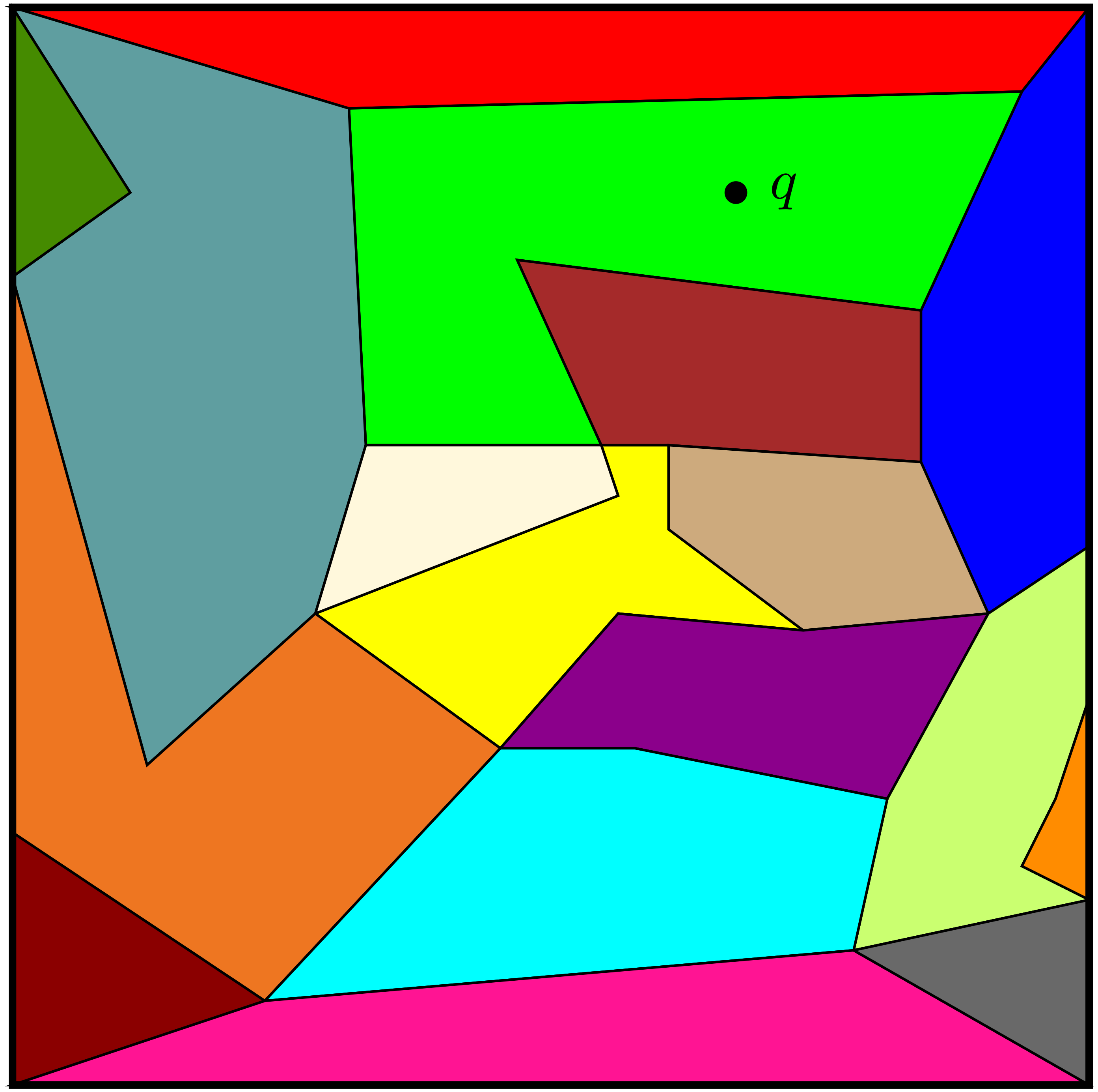
- Examples and preliminary results.
- Static setting: compressed quadtrees.

(deterministic)

- Dynamic setting: skip-quadtrees. (randomized)
- Adaptive meshing: balanced quadtrees. (deterministic)

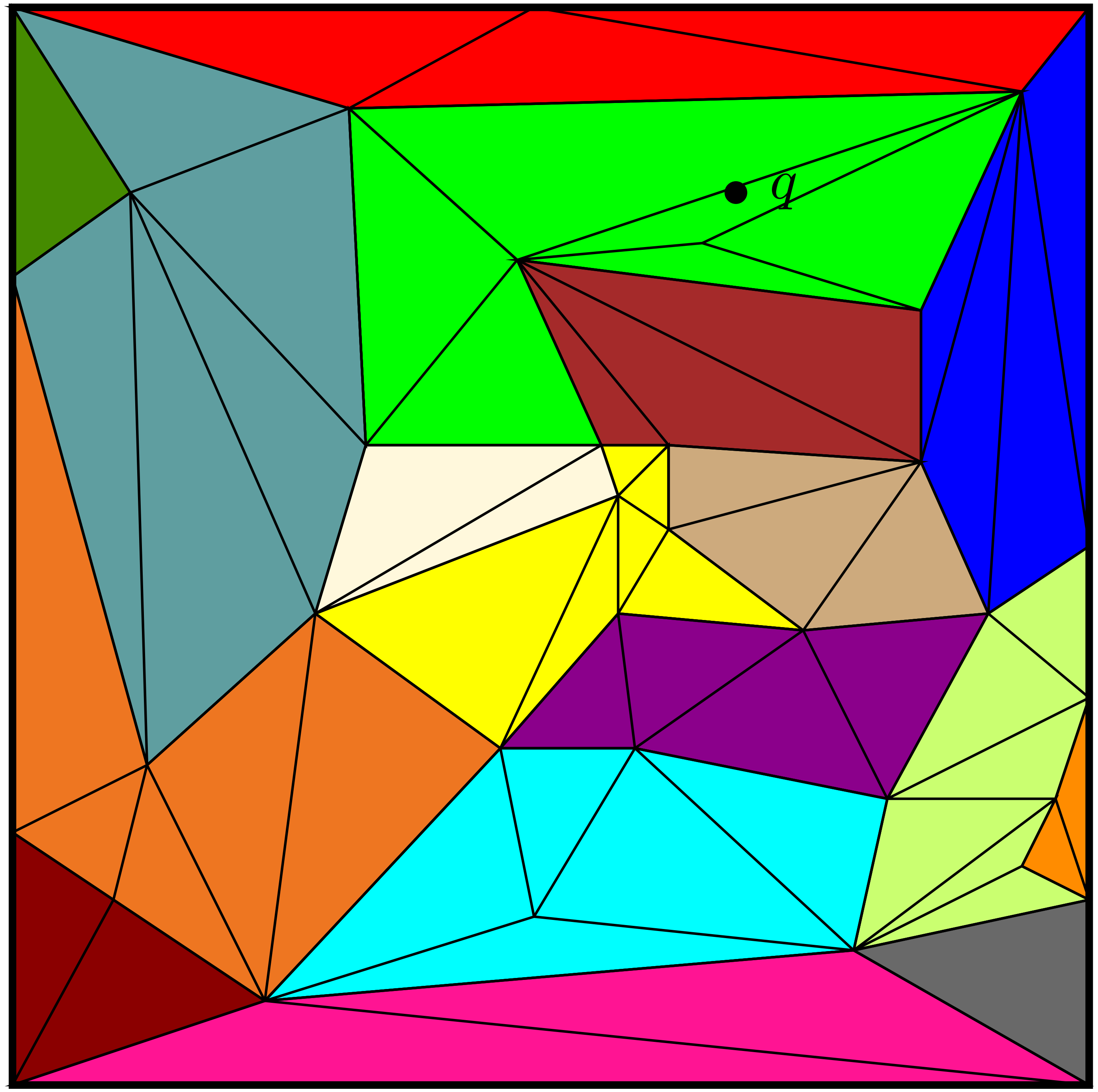
A first example (point location)

Goal: given a planar map M that partitions $[0,1[^2$, preprocess M such that, for any query point $q \in [0,1]^2$, the region containing q is found in sublinear time.



A first example (point location)

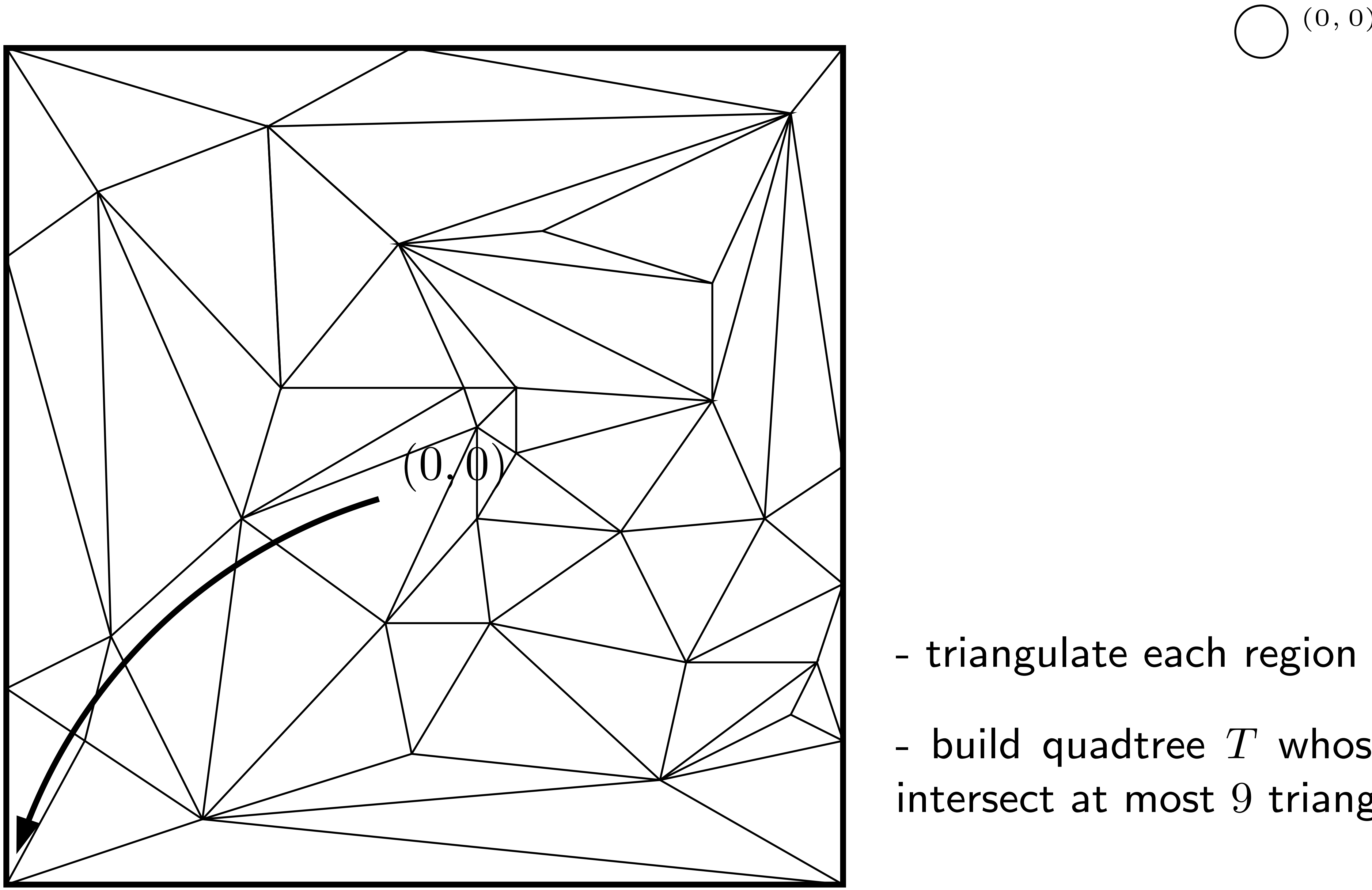
Goal: given a planar map M that partitions $[0,1[^2$, preprocess M such that, for any query point $q \in [0,1]^2$, the region containing q is found in sublinear time.



- triangulate each region

A first example (point location)

Goal: given a planar map M that partitions $[0,1[^2$, preprocess M such that, for any query point $q \in [0,1]^2$, the region containing q is found in sublinear time.

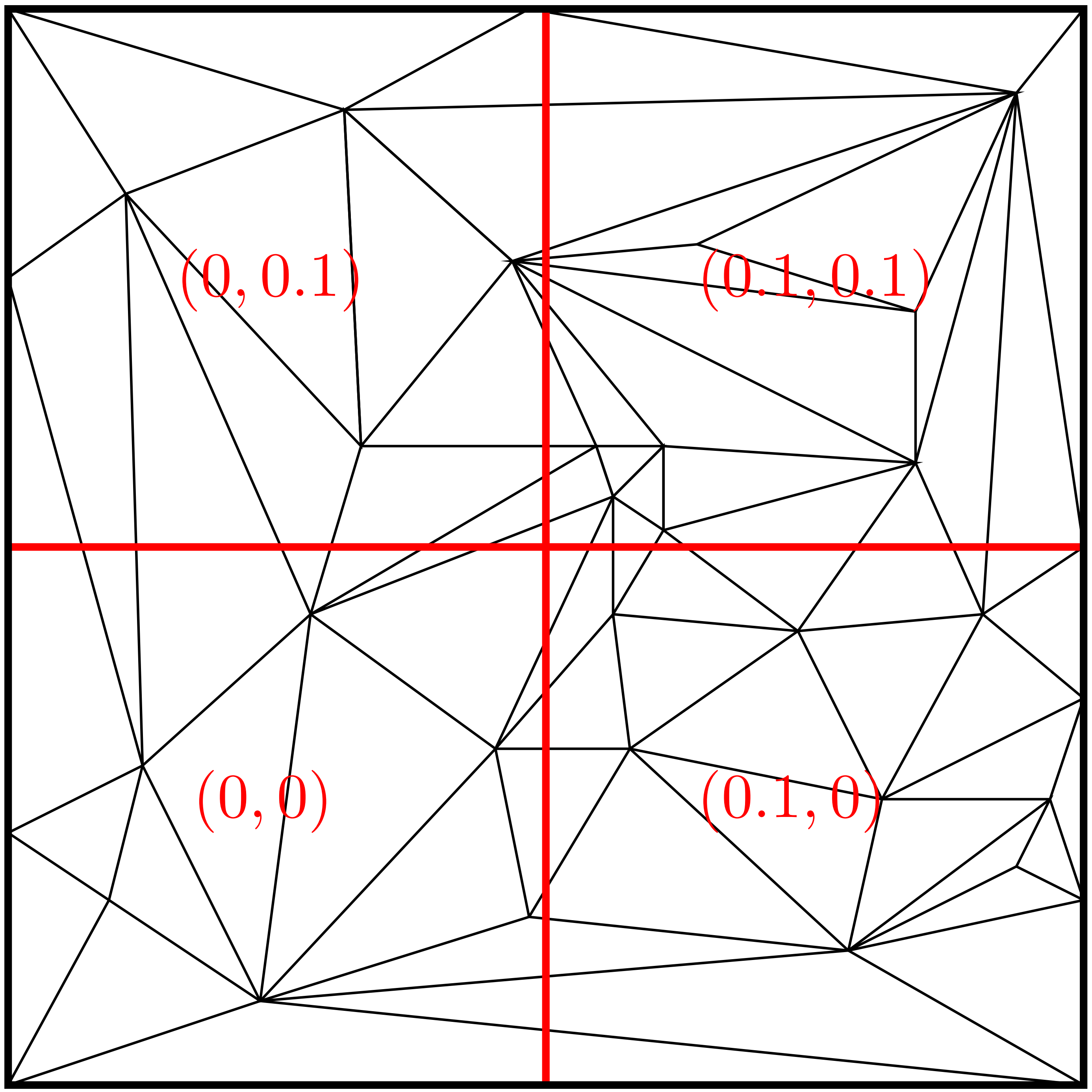


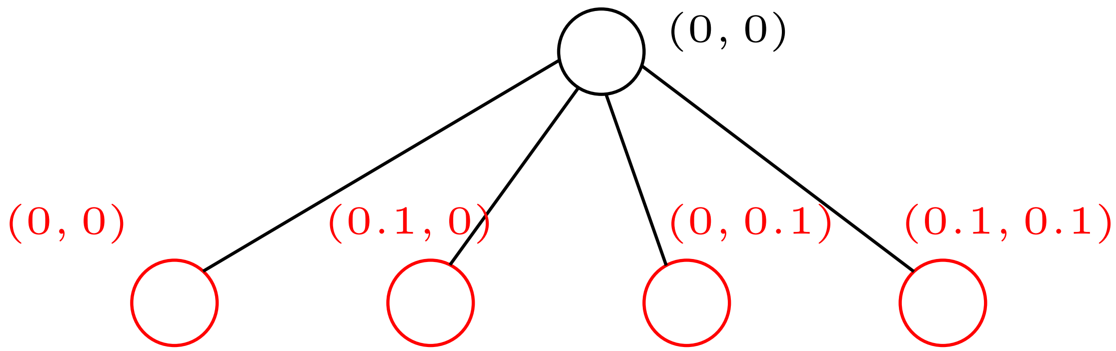
- triangulate each region

- build quadtree T whose leaves intersect at most 9 triangles

A first example (point location)

Goal: given a planar map M that partitions $[0,1[^2$, preprocess M such that, for any query point $q \in [0,1]^2$, the region containing q is found in sublinear time.



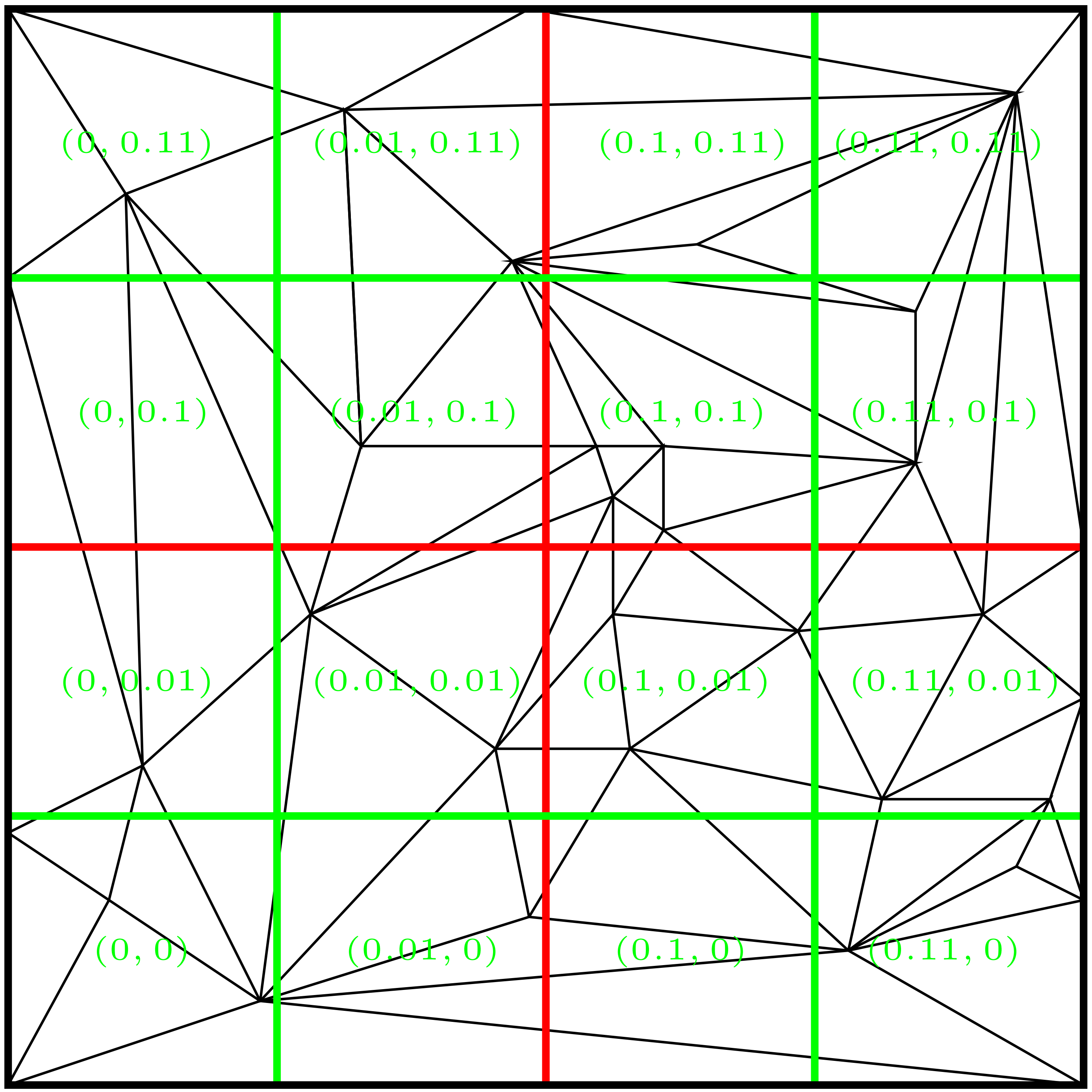


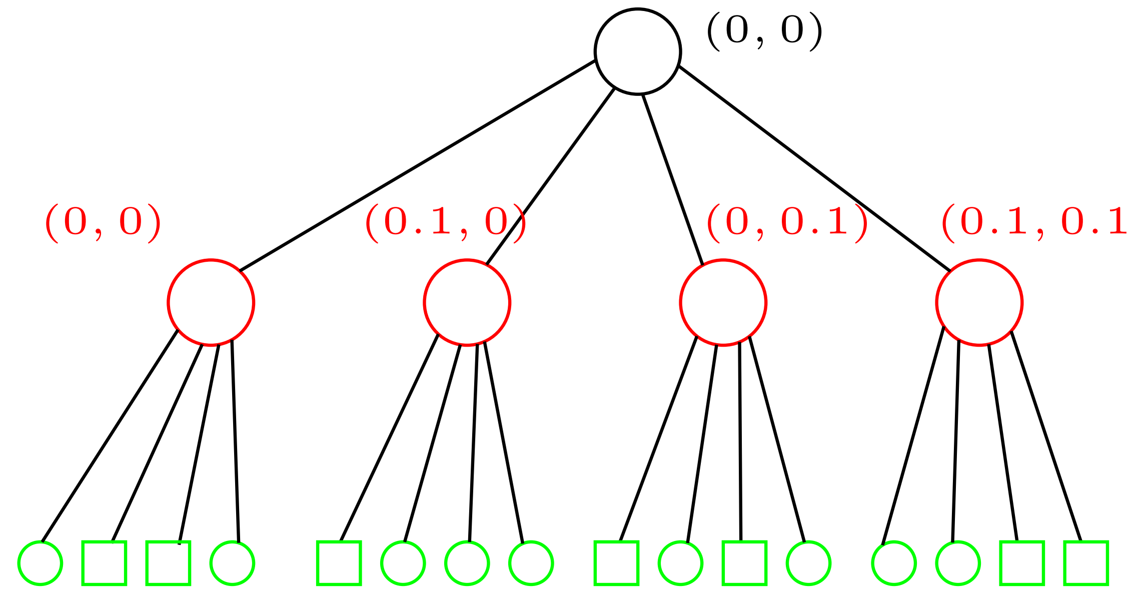
- triangulate each region

- build quadtree T whose leaves intersect at most 9 triangles

A first example (point location)

Goal: given a planar map M that partitions $[0,1[^2$, preprocess M such that, for any query point $q \in [0,1]^2$, the region containing q is found in sublinear time.



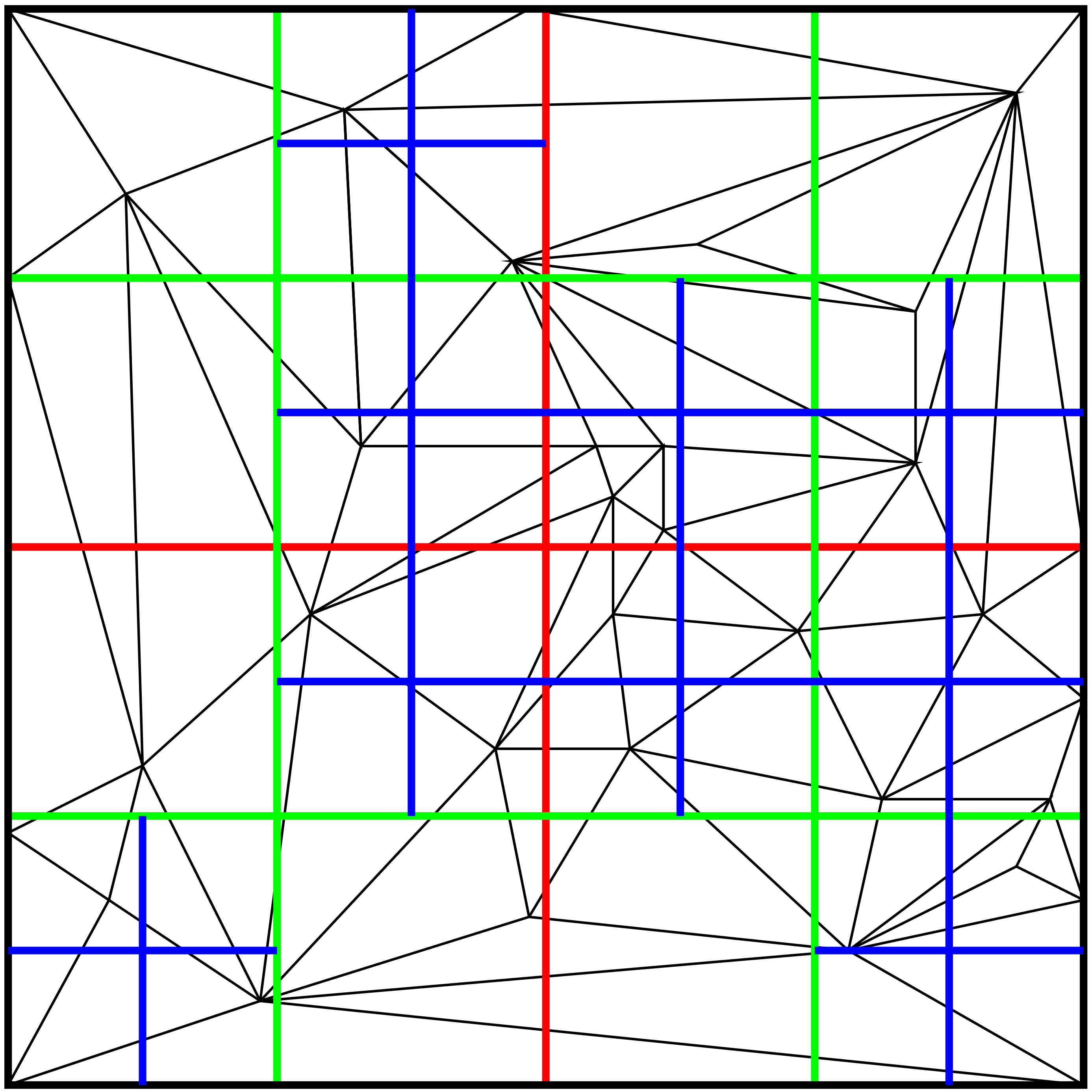


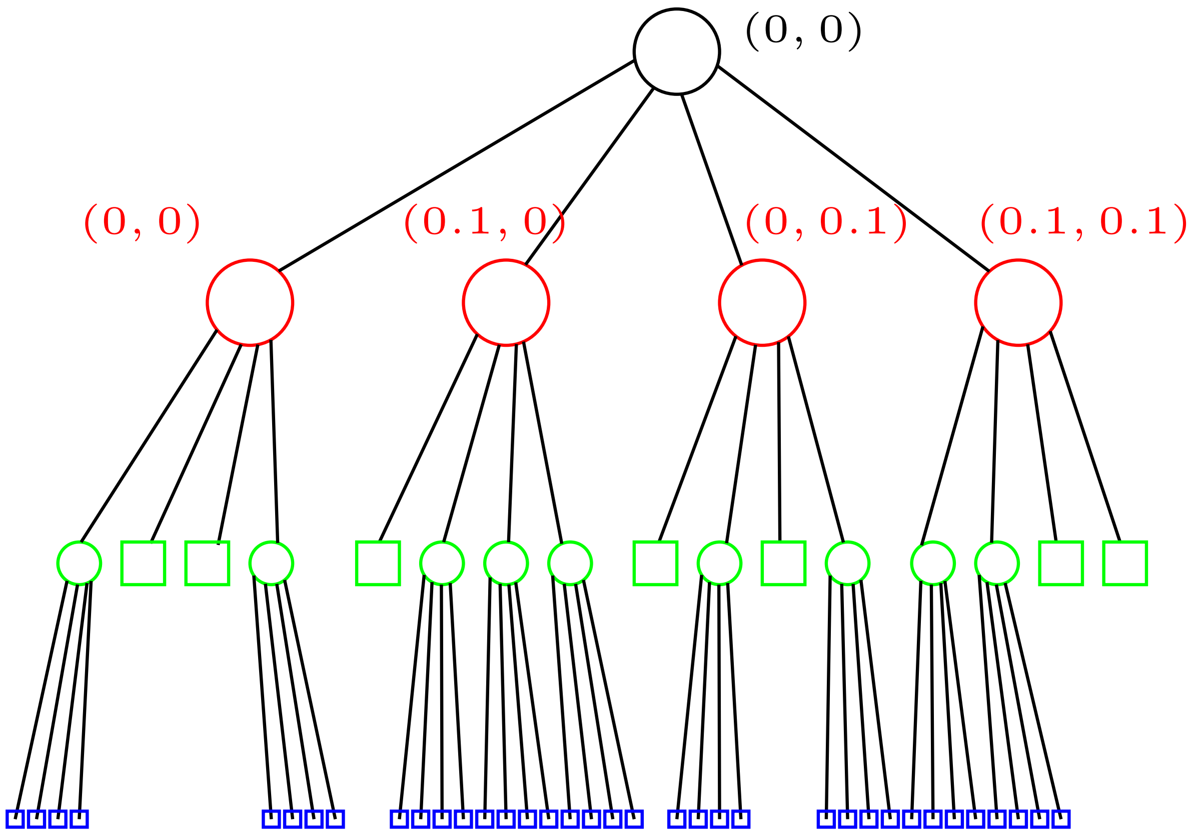
- triangulate each region

- build quadtree T whose leaves intersect at most 9 triangles

A first example (point location)

Goal: given a planar map M that partitions $[0,1[^2$, preprocess M such that, for any query point $q \in [0,1]^2$, the region containing q is found in sublinear time.



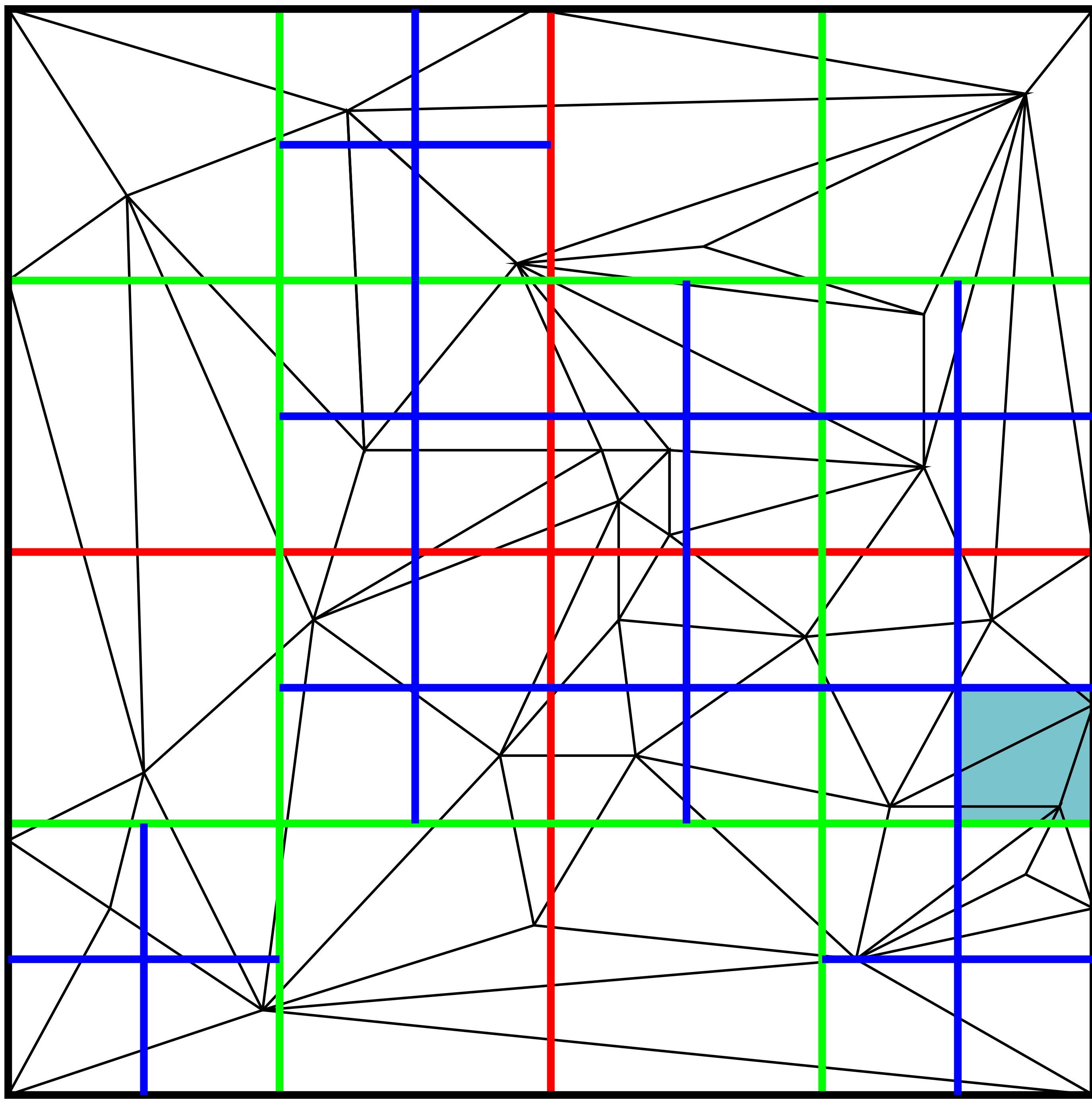


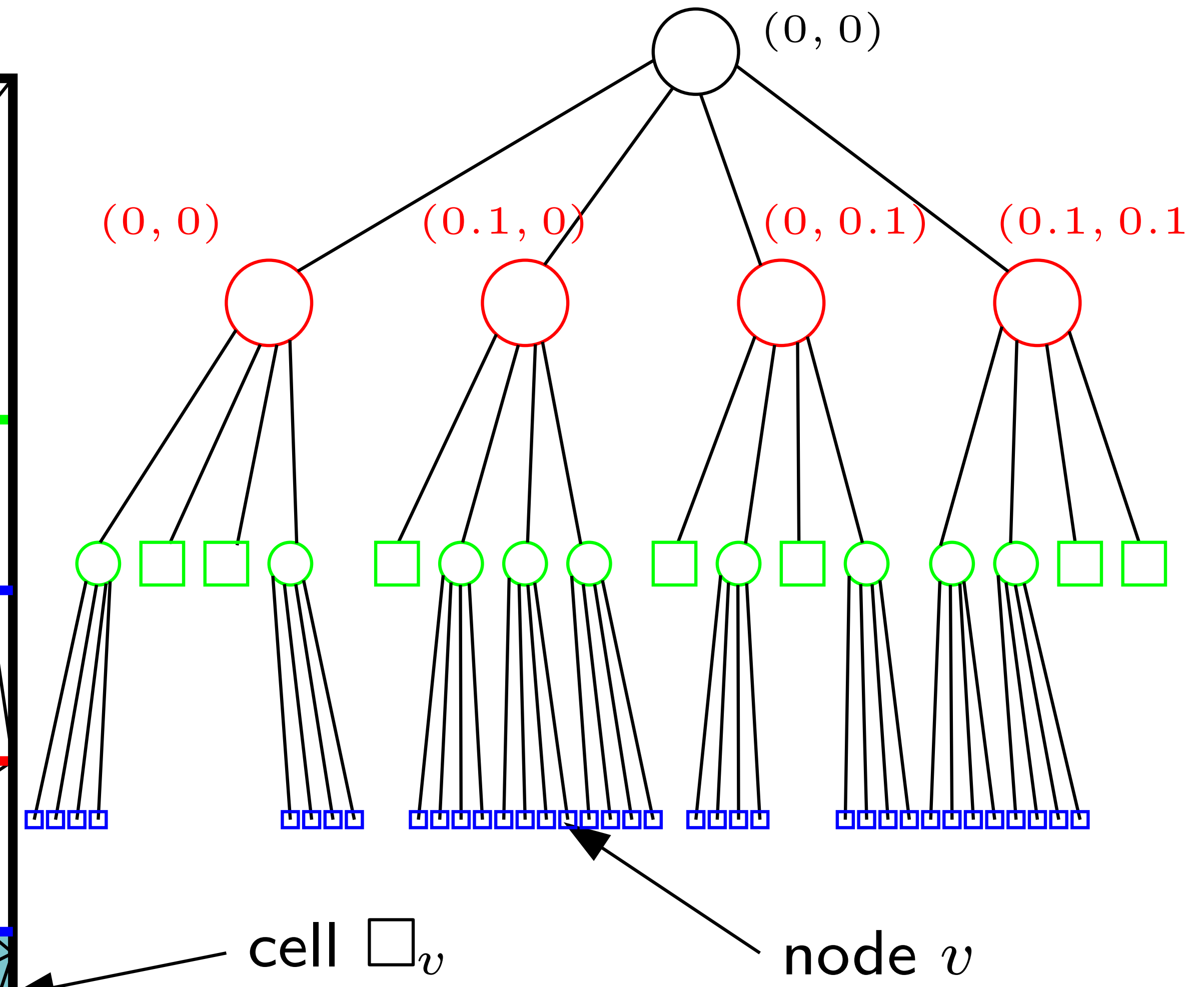
- triangulate each region

- build quadtree T whose leaves intersect at most 9 triangles

A first example (point location)

Goal: given a planar map M that partitions $[0,1[^2$, preprocess M such that, for any query point $q \in [0,1]^2$, the region containing q is found in sublinear time.



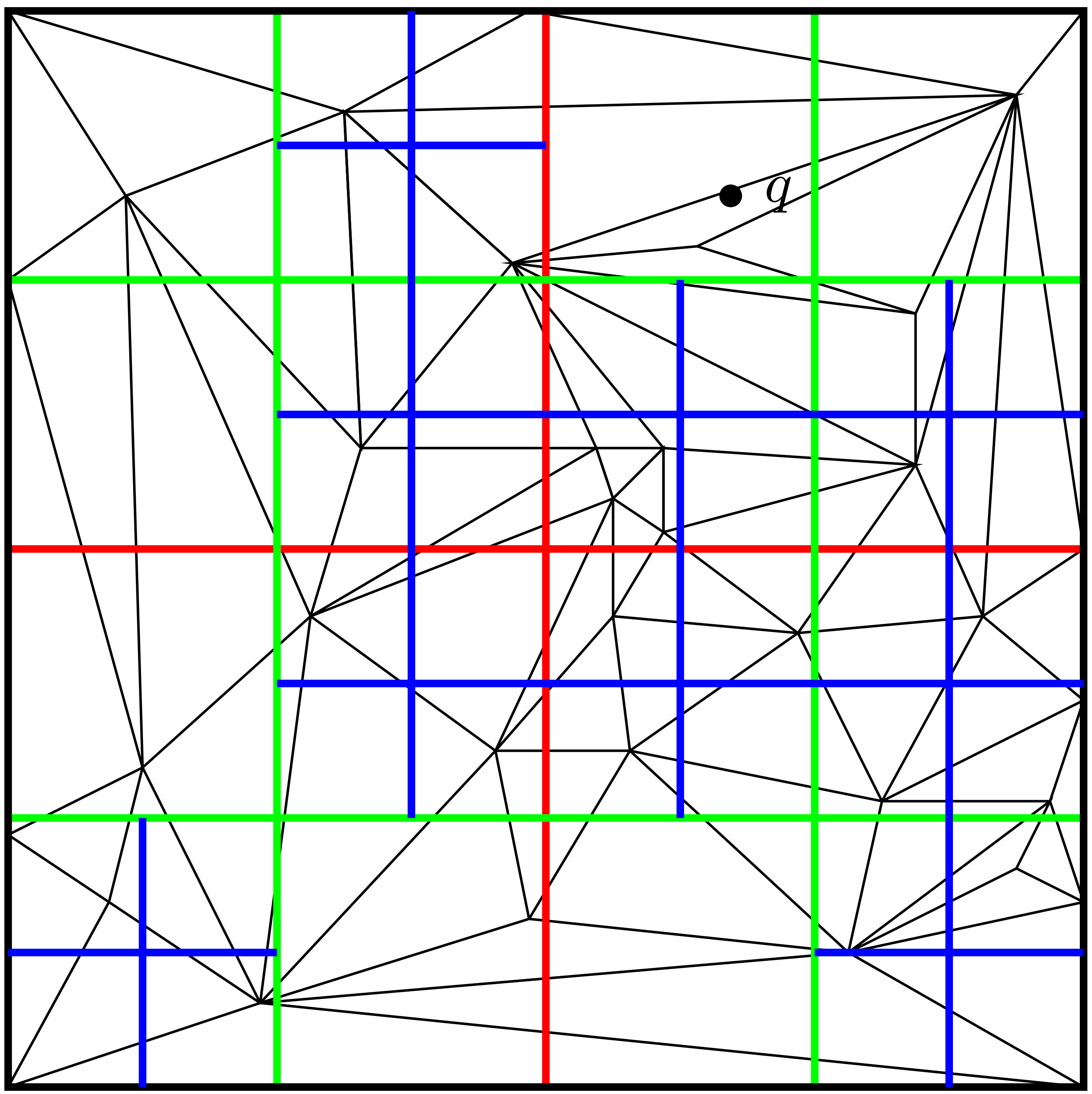


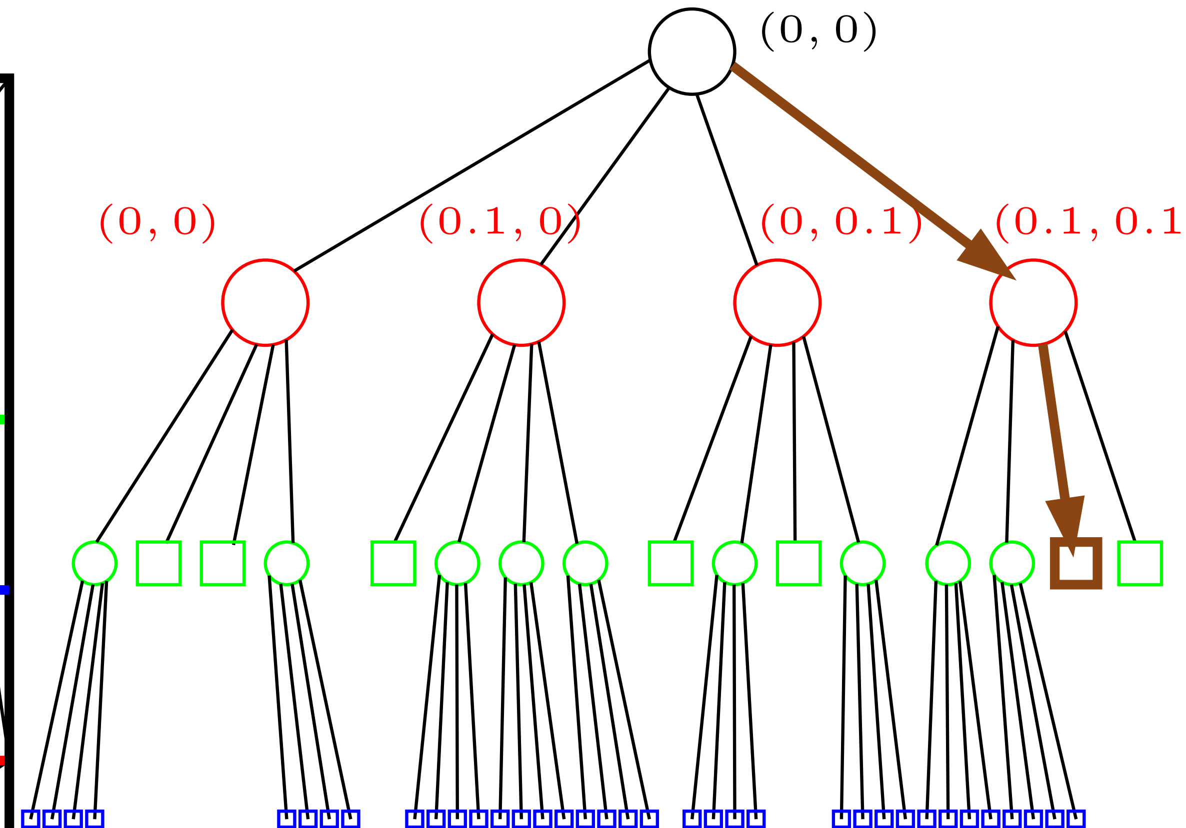
- triangulate each region

- build quadtree T whose leaves intersect at most 9 triangles

A first example (point location)

Goal: given a planar map M that partitions $[0,1[^2$, preprocess M such that, for any query point $q \in [0,1]^2$, the region containing q is found in sublinear time.





- $\forall q \in [0,1[^2$, walk down T to find leaf $v \in L$ that contains q, then check triangles that intersect $\square_v$.

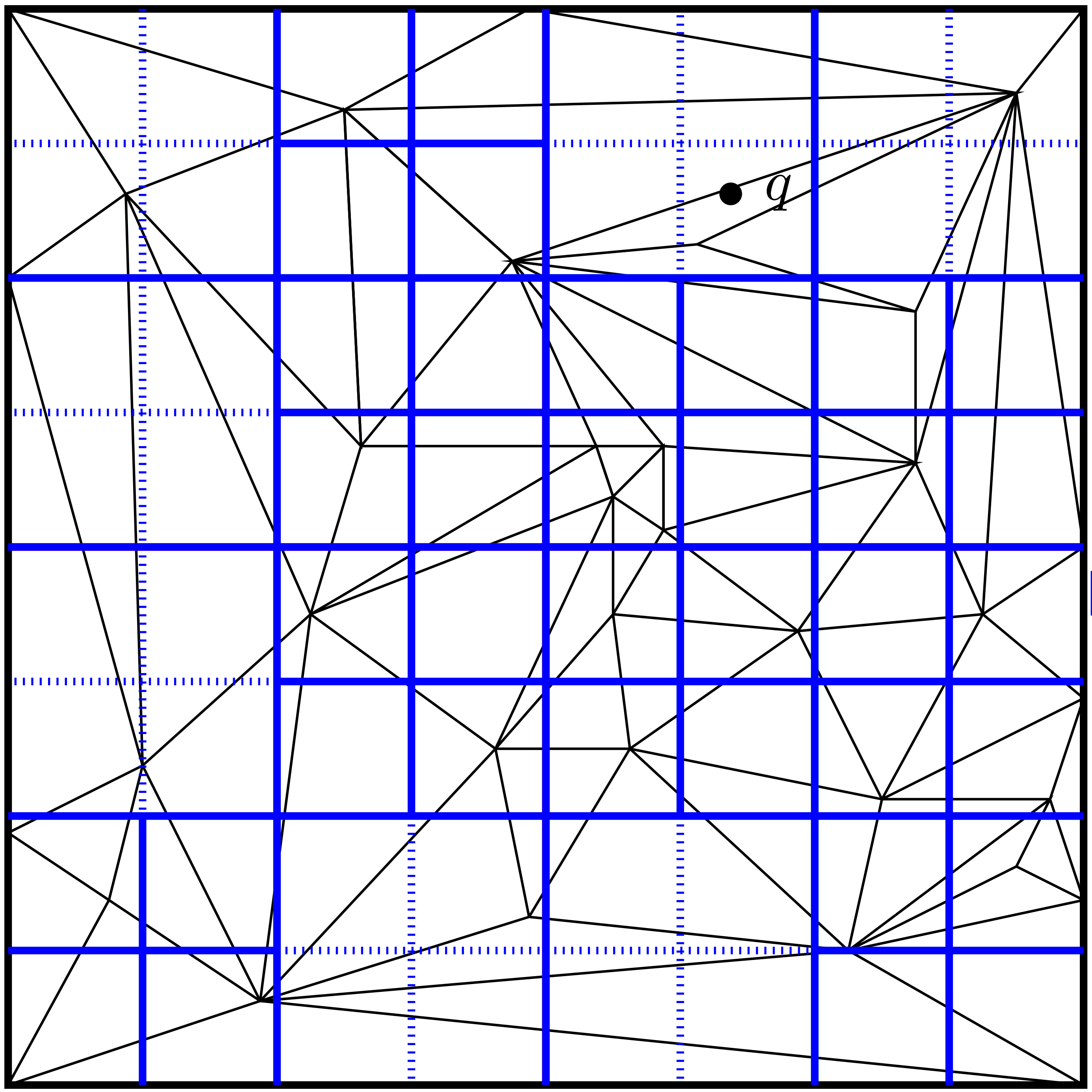
$\Rightarrow$ size $O(|L|)$, time $O(h)$

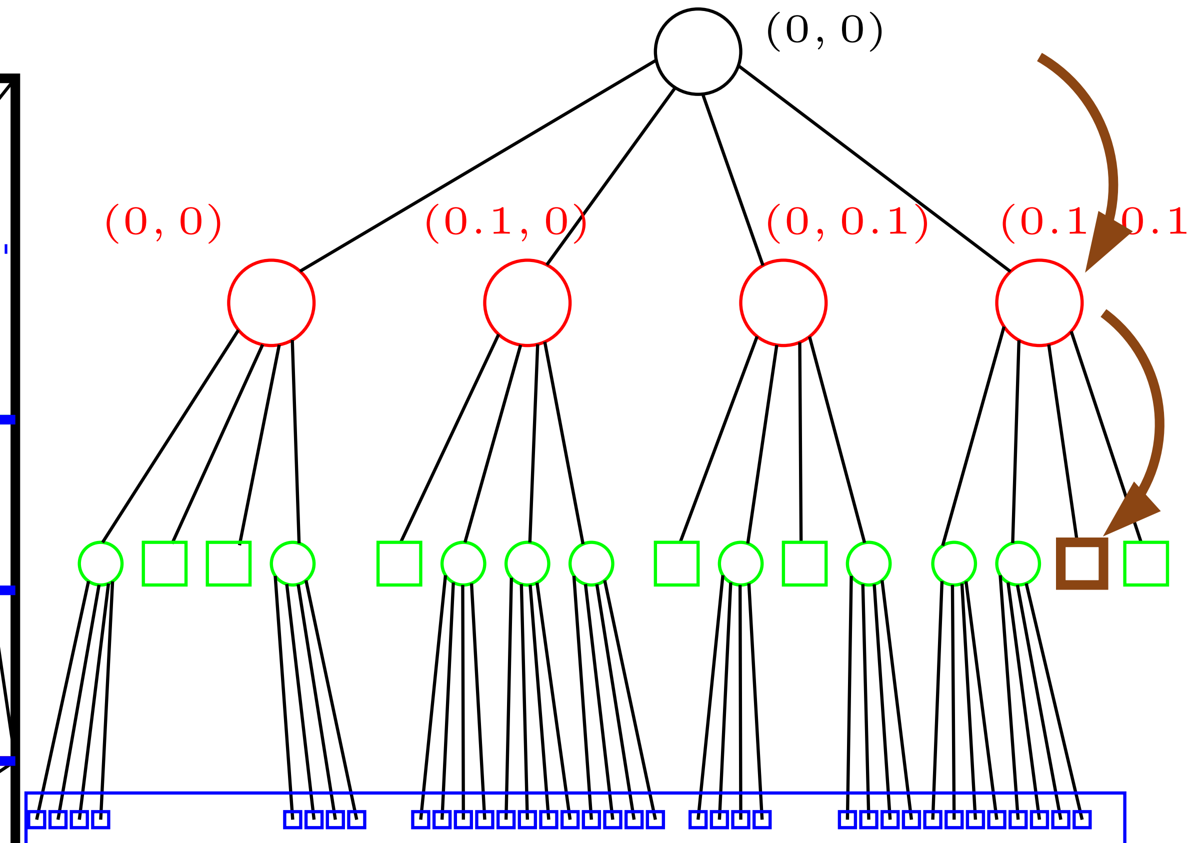
regular grid : size $\Theta(2^{2h})$, time $O(1)$

Q can we do better?

A first example (point location)

Goal: given a planar map M that partitions $[0,1[^2$, preprocess M such that, for any query point $q \in [0,1]^2$, the region containing q is found in sublinear time.





- level i forms a 2^{-i}-regular grid of $[0,1[^2$

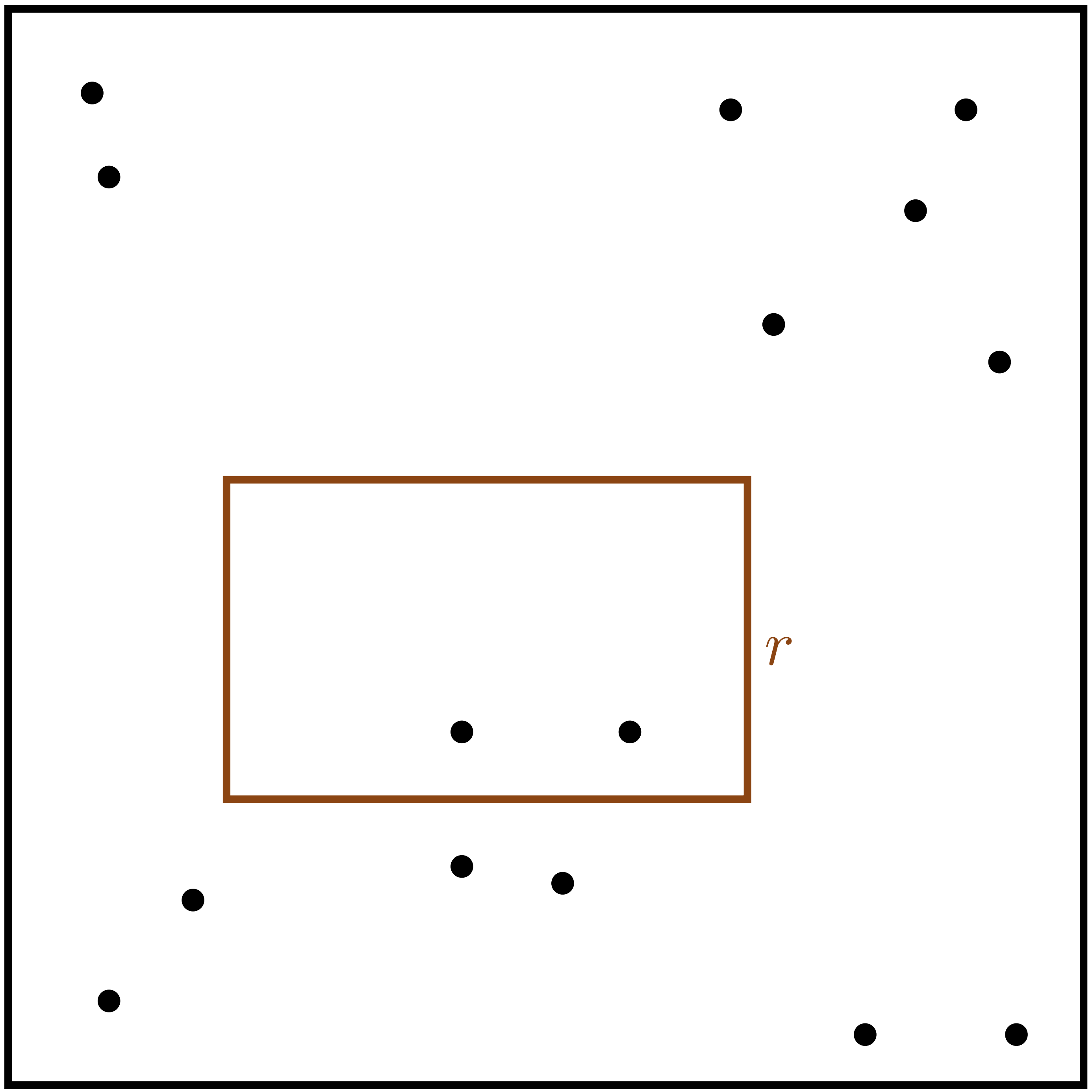
  $\Rightarrow \forall i,\ v_i(q) = \left(2^{-i}\lfloor 2^i q_x \rfloor, 2^{-i}\lfloor 2^i q_y \rfloor\right)$
- put nodes in hash-table
- $\forall q \in [0,1[^2$, binary search on height:

  Let $i = \mathrm{hmax}{+}\mathrm{hmin}/2$;
  if $v_i(q) \neq \mathrm{NULL}$, then search between i and hmax;
  else search between hmin and i;

  $\Rightarrow O(\log h)$

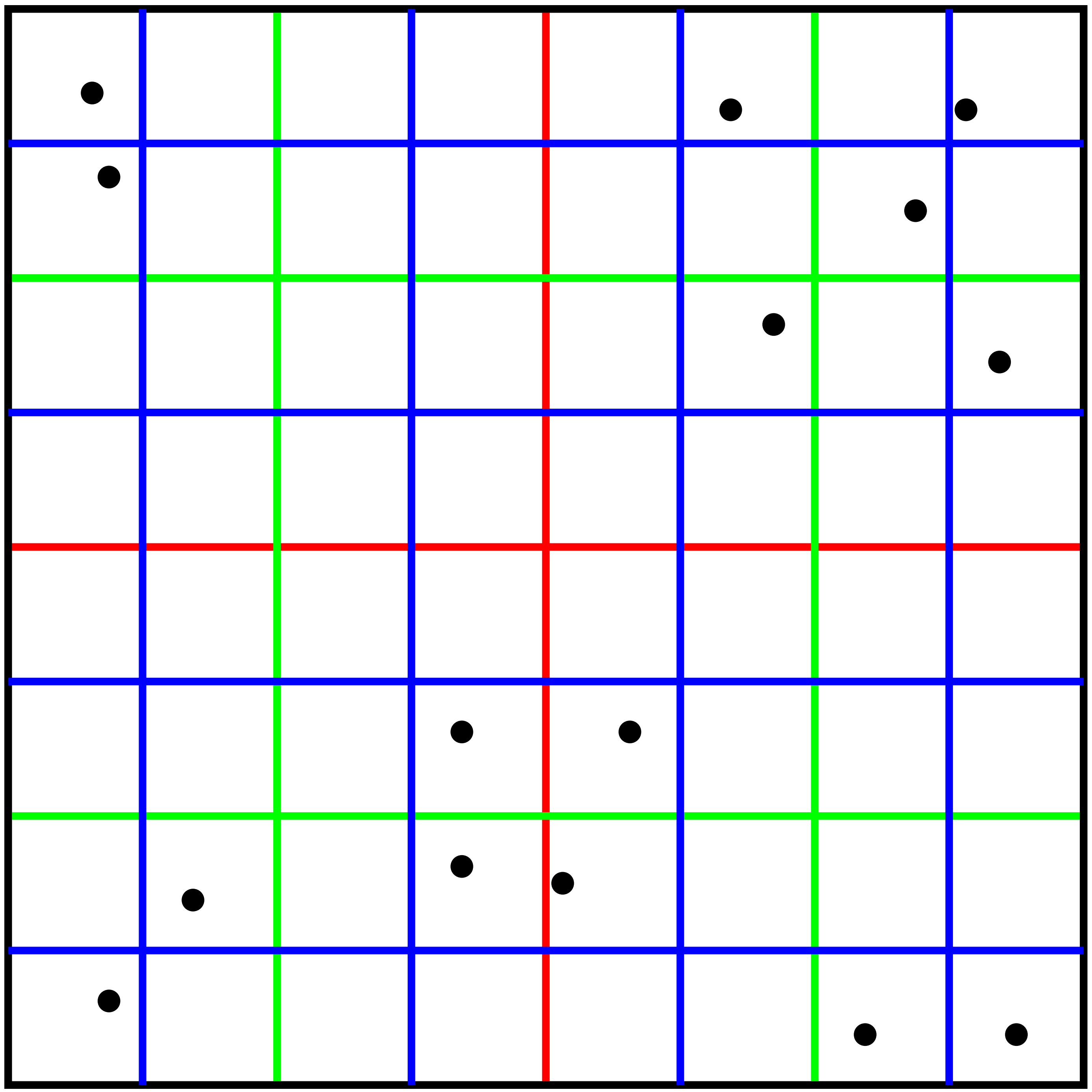
Another example (range searching)

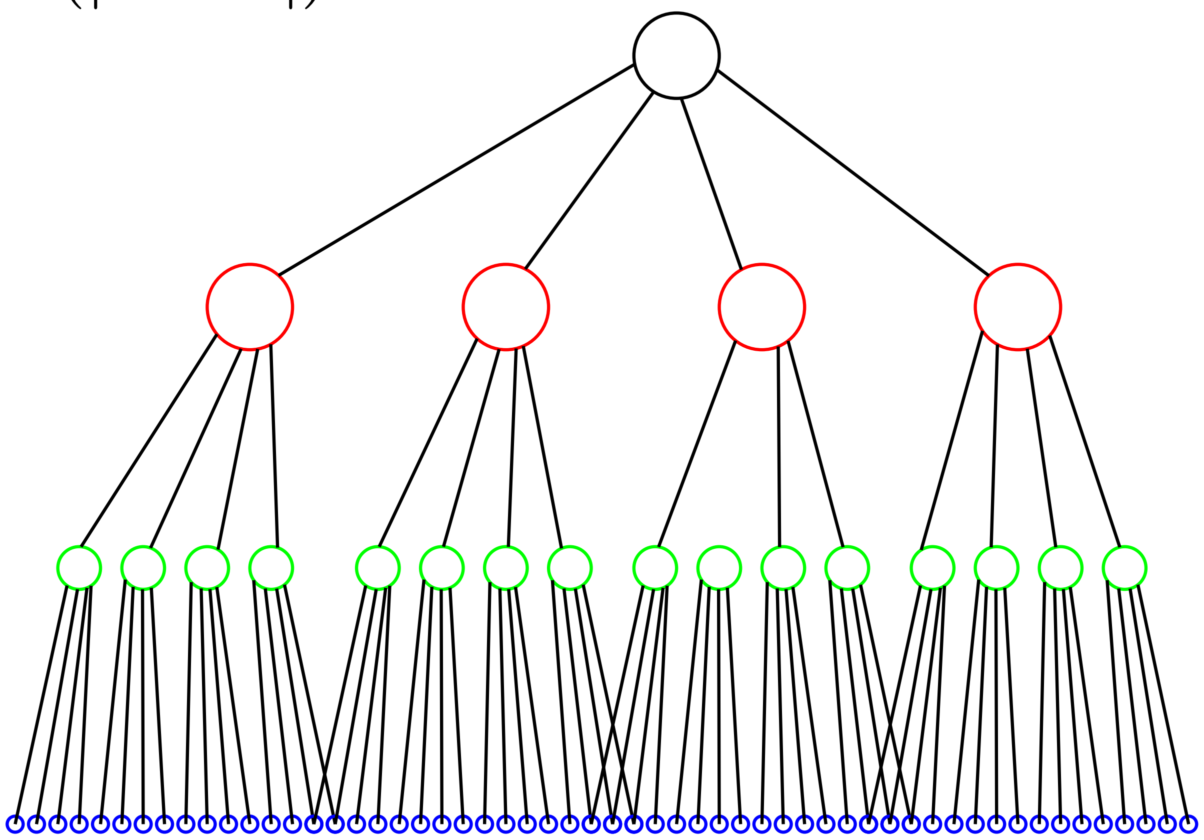
Goal: given a finite point set $P \subset [0,1[^2$, preprocess P such that, for any query rectangle $r \subseteq [0,1]^2$, $r \cap P$ is found in time $O(|r \cap P|)$.



Another example (range searching)

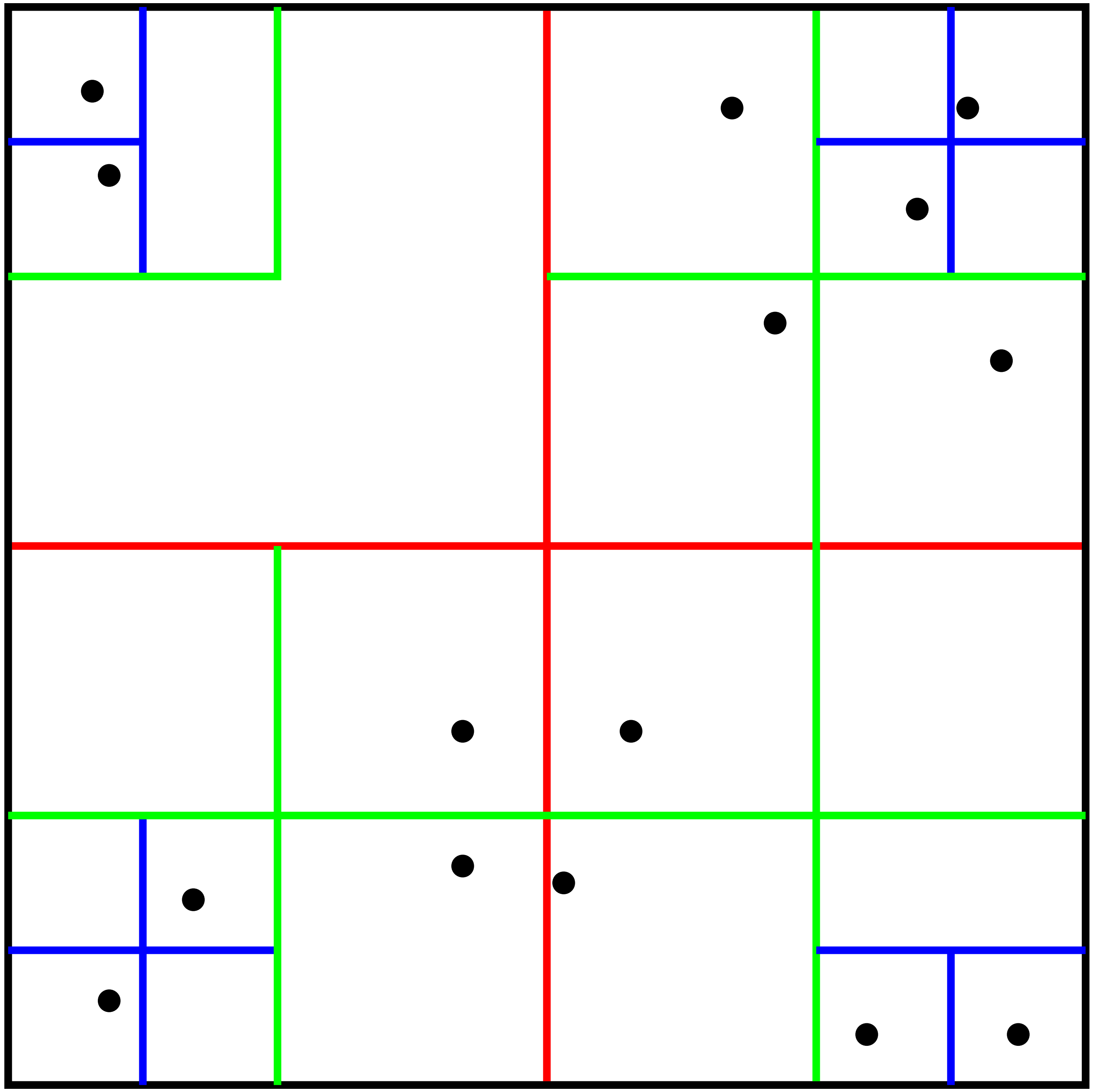
Goal: given a finite point set $P \subset [0,1[^2$, preprocess P such that, for any query rectangle $r \subseteq [0,1]^2$, $r \cap P$ is found in time $O(|r \cap P|)$.

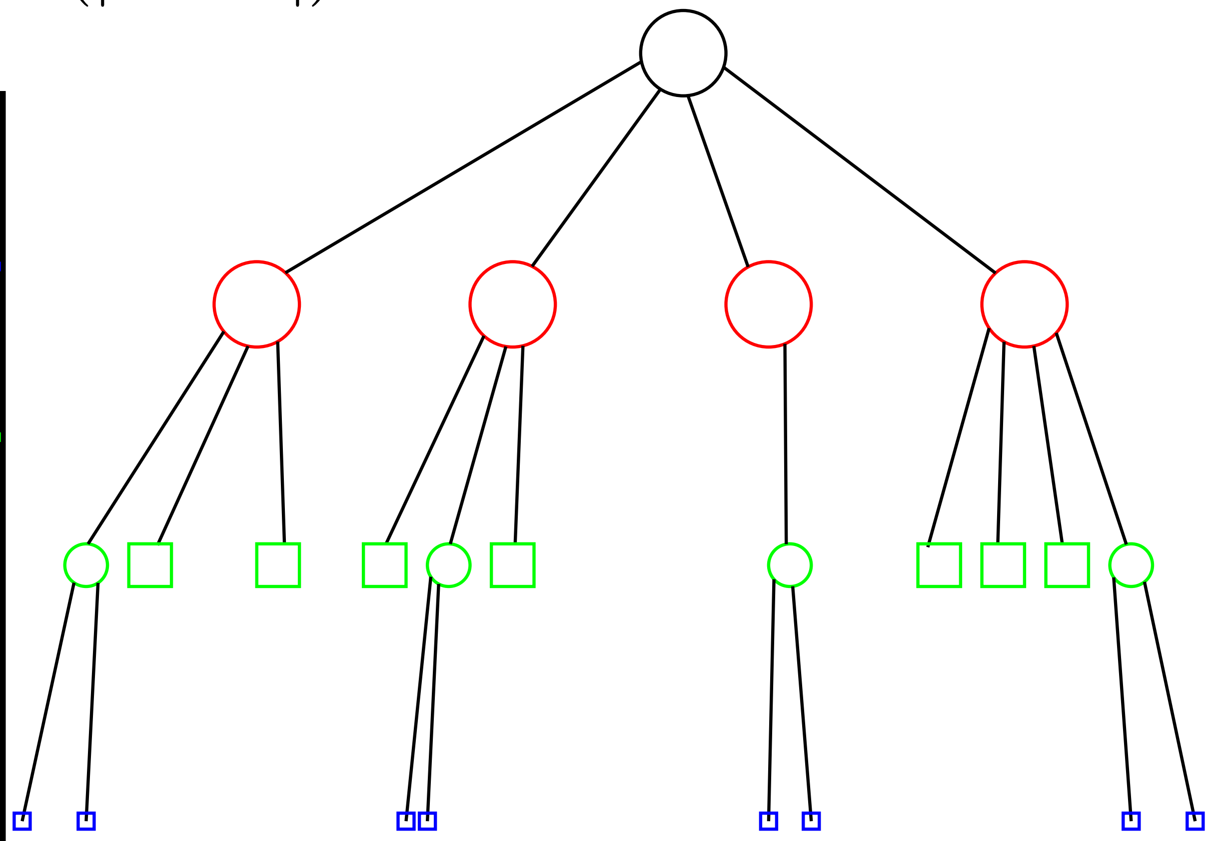




Another example (range searching)

Goal: given a finite point set $P \subset [0,1[^2$, preprocess P such that, for any query rectangle $r \subseteq [0,1]^2$, $r \cap P$ is found in time $O(|r \cap P|)$.

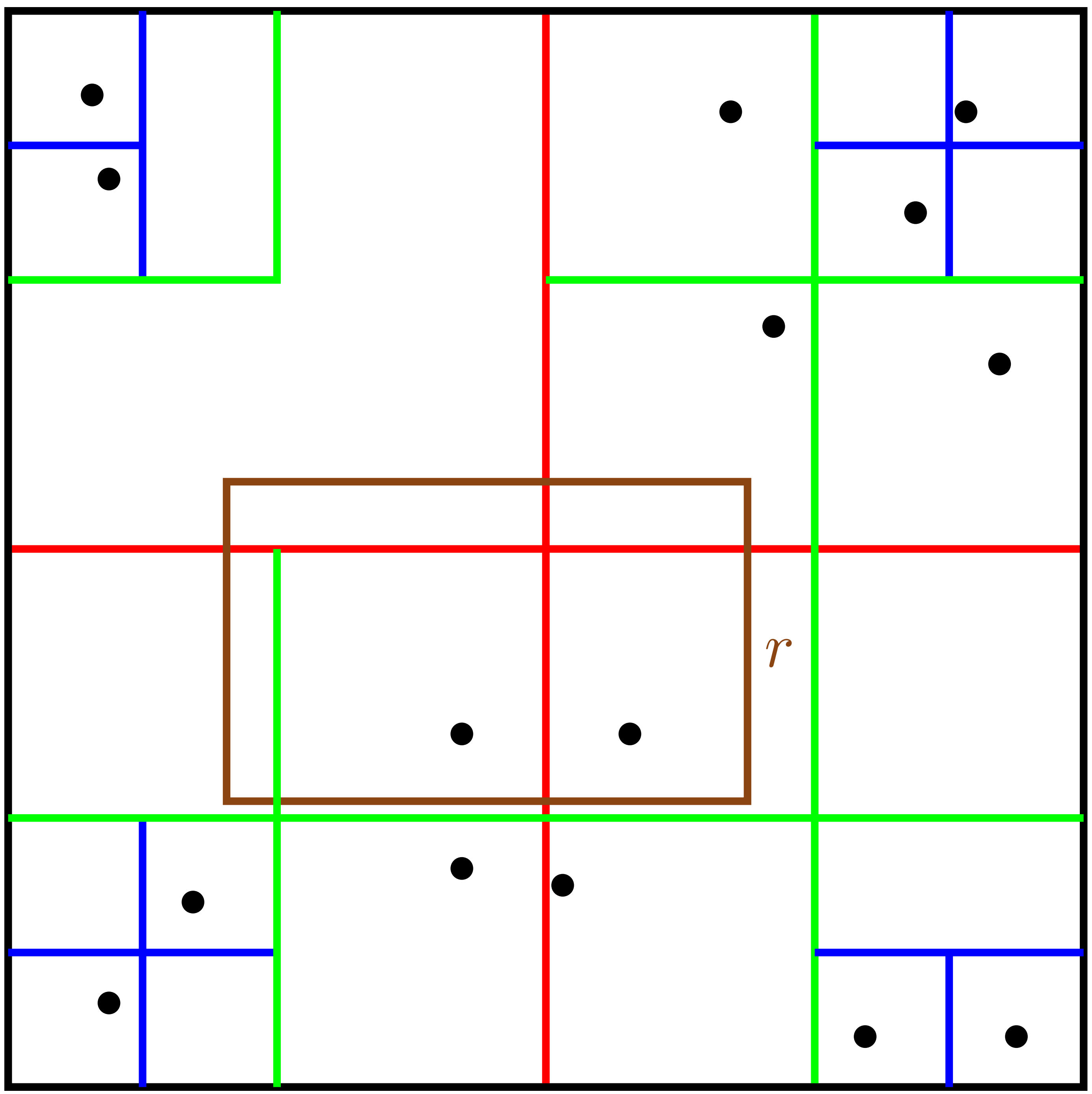


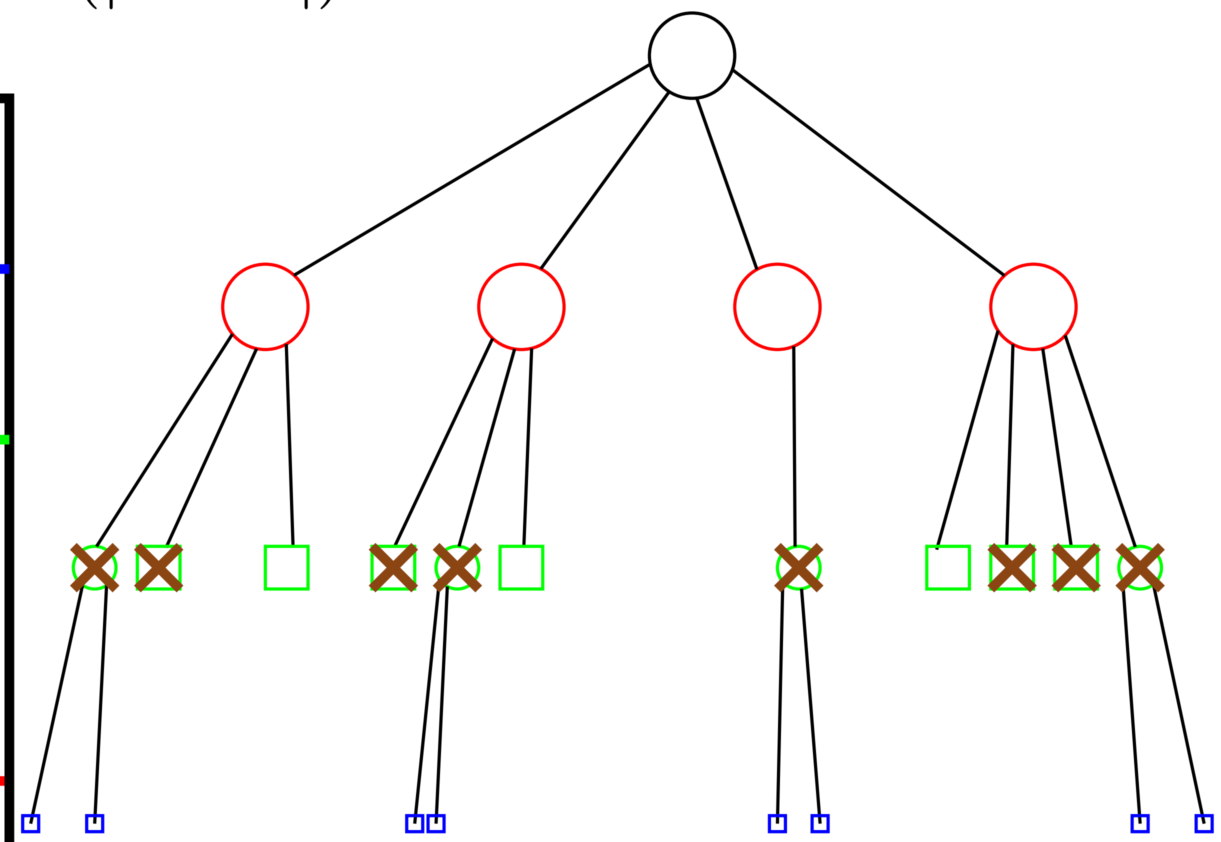


1 point per leaf ($\Rightarrow |P|$ leaves)
$\Rightarrow |T| \leq h|L| = h|P|$.

Another example (range searching)

Goal: given a finite point set $P \subset [0,1[^2$, preprocess P such that, for any query rectangle $r \subseteq [0,1]^2$, $r \cap P$ is found in time $O(|r \cap P|)$.





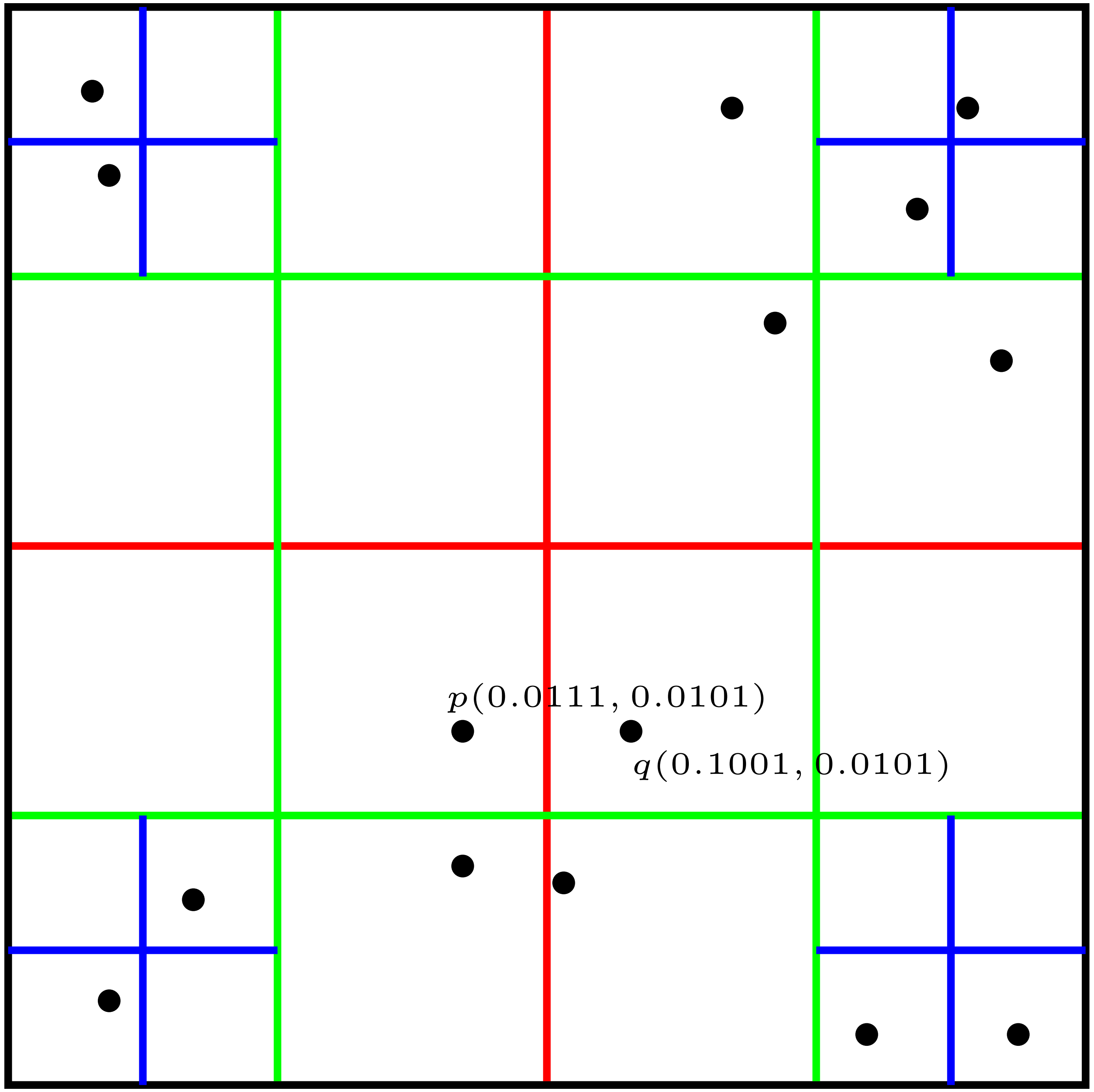
1 point per leaf ($\Rightarrow |P|$ leaves)
$\Rightarrow |T| \leq h|L| = h|P|$.

Query: go from root to leaves, stopping each time $r \cap \square_v = \emptyset$.
$\Rightarrow O(|r \cap T|) \leq O(h|r \cap L|)$

Q Bound on h?

Various bounds

Lemma Let $P \subset [0,1[^2$ be finite. Assume wlog $\mathrm{diam}(P) = \max\{\mathsf{d}(p,q|p,q \in P\} \geq 1/2$. Then, $h = O(\log \Phi(P))$, where $\Phi(P) = {}^{\mathrm{diam}(P)}/_{\min\{\mathsf{d}(p,q)|p,q\in P\}}$.



Def $\forall p,q \in P$, let $h(p,q)$ be the smallest i s.t. $v_i(p) \neq v_i(q)$.

$\forall i,\ v_i(p) = \left(2^{-i}\lfloor 2^i p_x \rfloor, 2^{-i}\lfloor 2^i p_y \rfloor\right)$

Prop $\forall p,q \in [0,1]^2,\ h(p,q) =$
$\min\{-\lceil\log(p_x \dot{\vee} q_x)\rceil, -\lceil\log(p_y \dot{\vee} q_y)\rceil\}$
$\leq \min\{-\lceil\log|p_x - q_x|, -\lceil\log|p_y - q_y|\}$
$= -\lceil\log\max\{|p_x - q_x|, |p_y - q_y|\}\rceil$
$\leq -\lceil\log\frac{1}{\sqrt{2}}\mathsf{d}(p,q)\rceil = \frac{1}{2} - \log\mathsf{d}(p,q)$

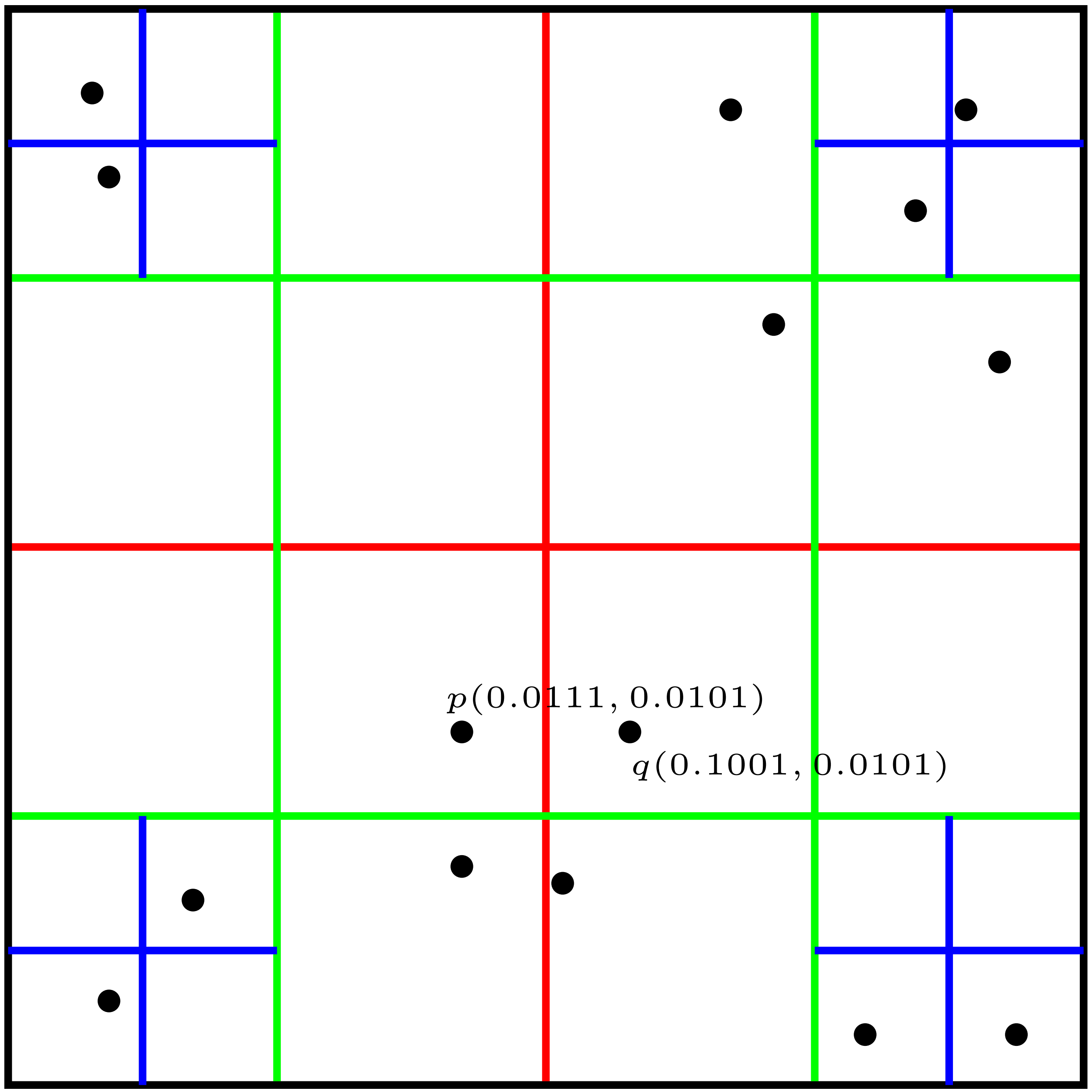
Observation: for every internal node v of T, $|\Box_v \cap P| \geq 2$

$\Rightarrow l(v) \leq h(p,q) - 1,\ \forall p,q \in \Box_v$

$\Rightarrow h \leq \frac{1}{2} - \log\min\{\mathsf{d}(p,q)|p,q \in P\}$
$\quad \leq \frac{3}{2} + \log\Phi(P)$

Various bounds

Lemma Let $P \subset [0,1[^2$ be finite. Assume wlog $\operatorname{diam}(P) = \max\{\mathsf{d}(p,q|p,q \in P\} \geq 1/2$. Then, $h = O(\log \Phi(P))$, where $\Phi(P) = {}^{\operatorname{diam}(P)}/_{\min\{\mathsf{d}(p,q)|p,q \in P\}}$.

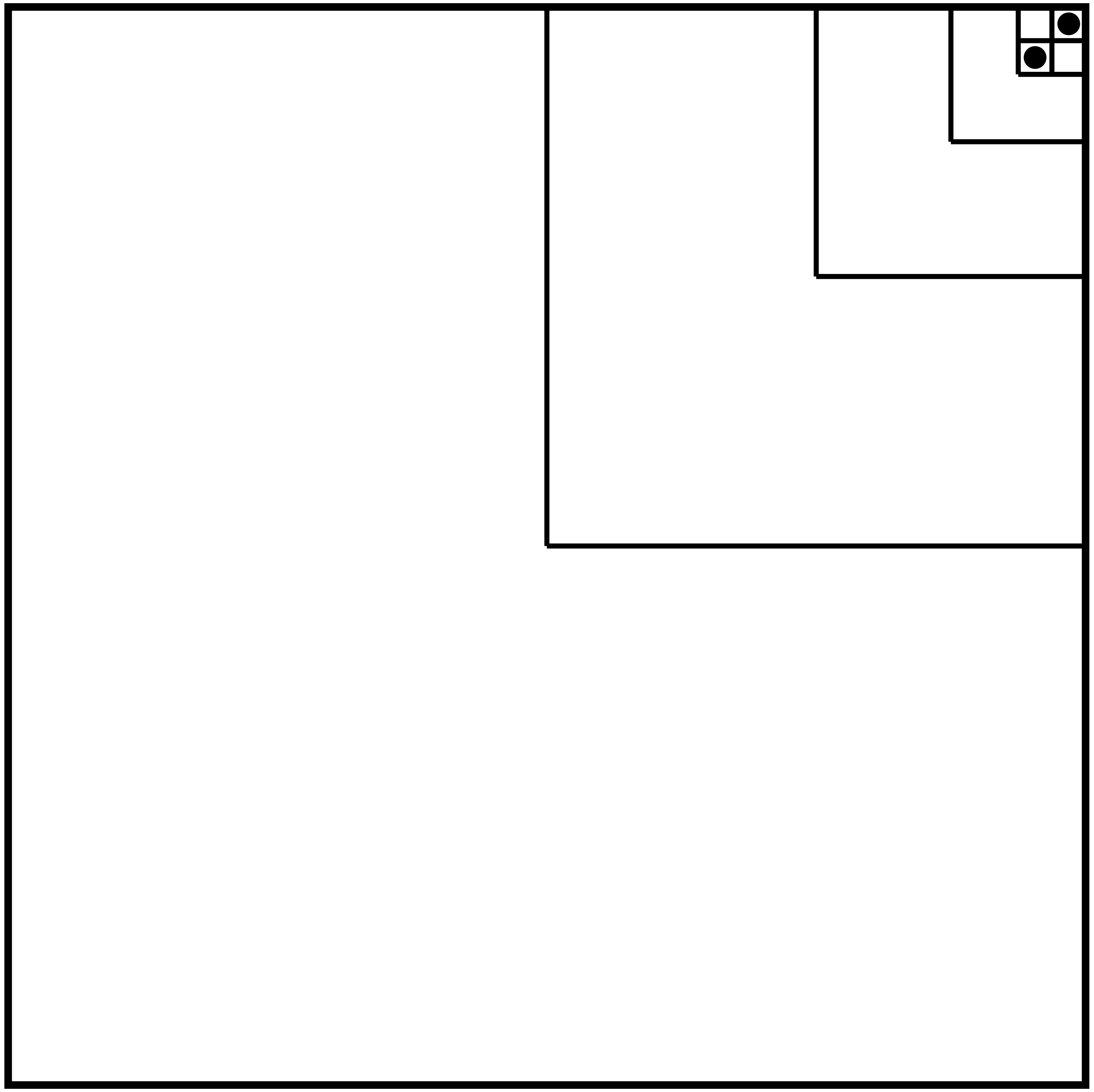


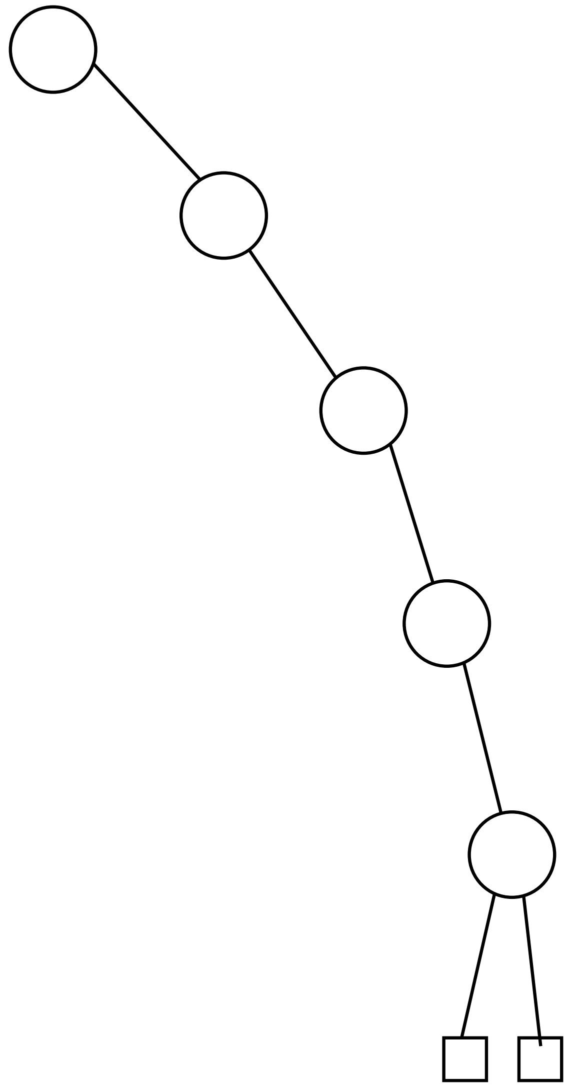
Corollary
data structure size: $O(|P| \log \Phi(P))$
construction time: $O(|P| \log \Phi(P))$
query time: $O(\log \log \Phi(P))$

Q Can we do better?

Compressed quadtrees

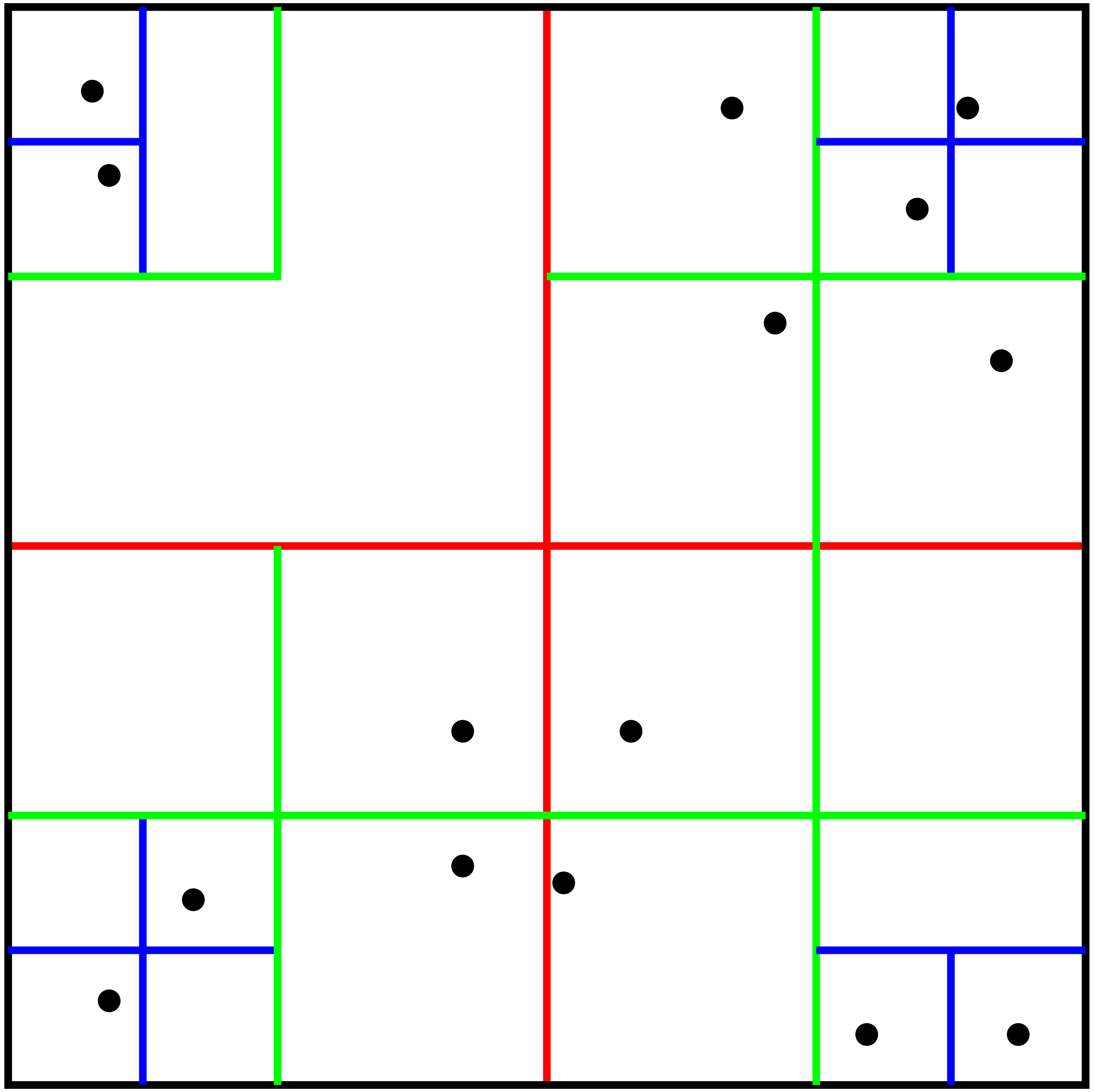
Pb Bounds on complexity depend on $\Phi(P)$.

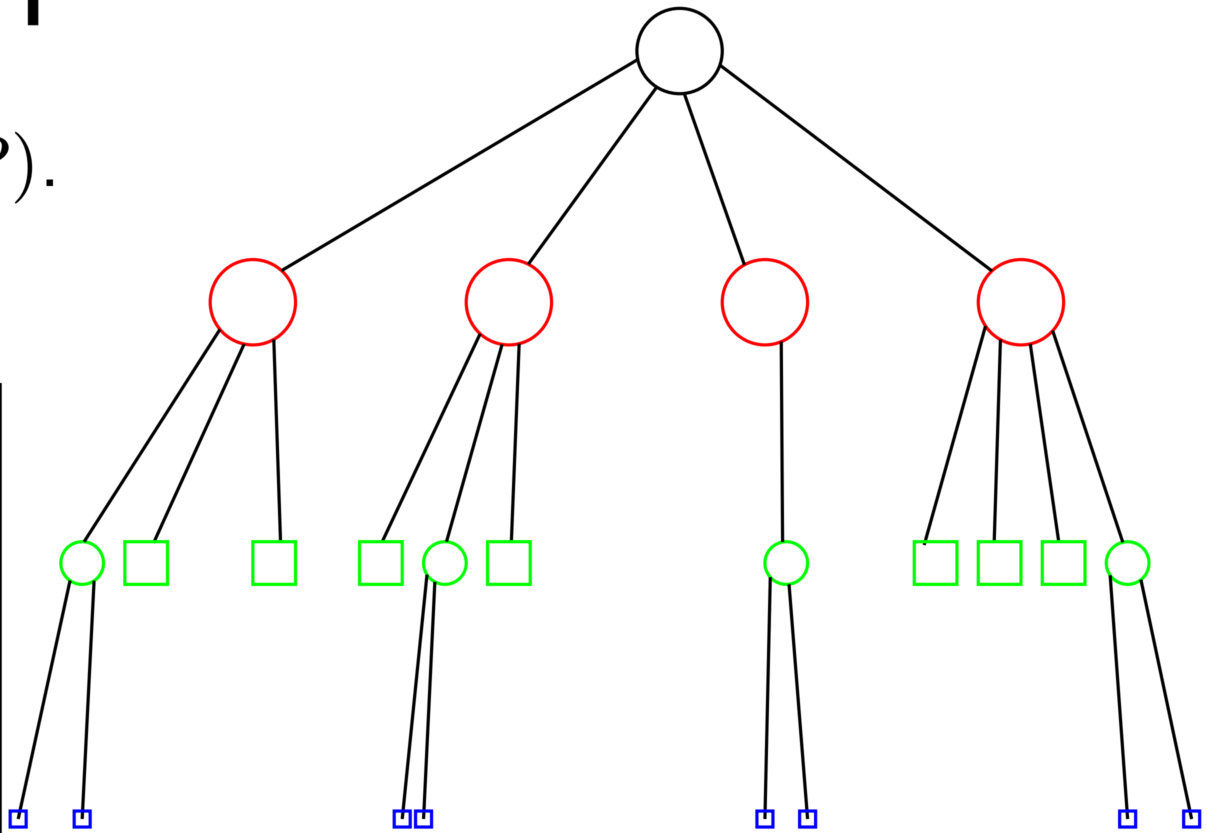




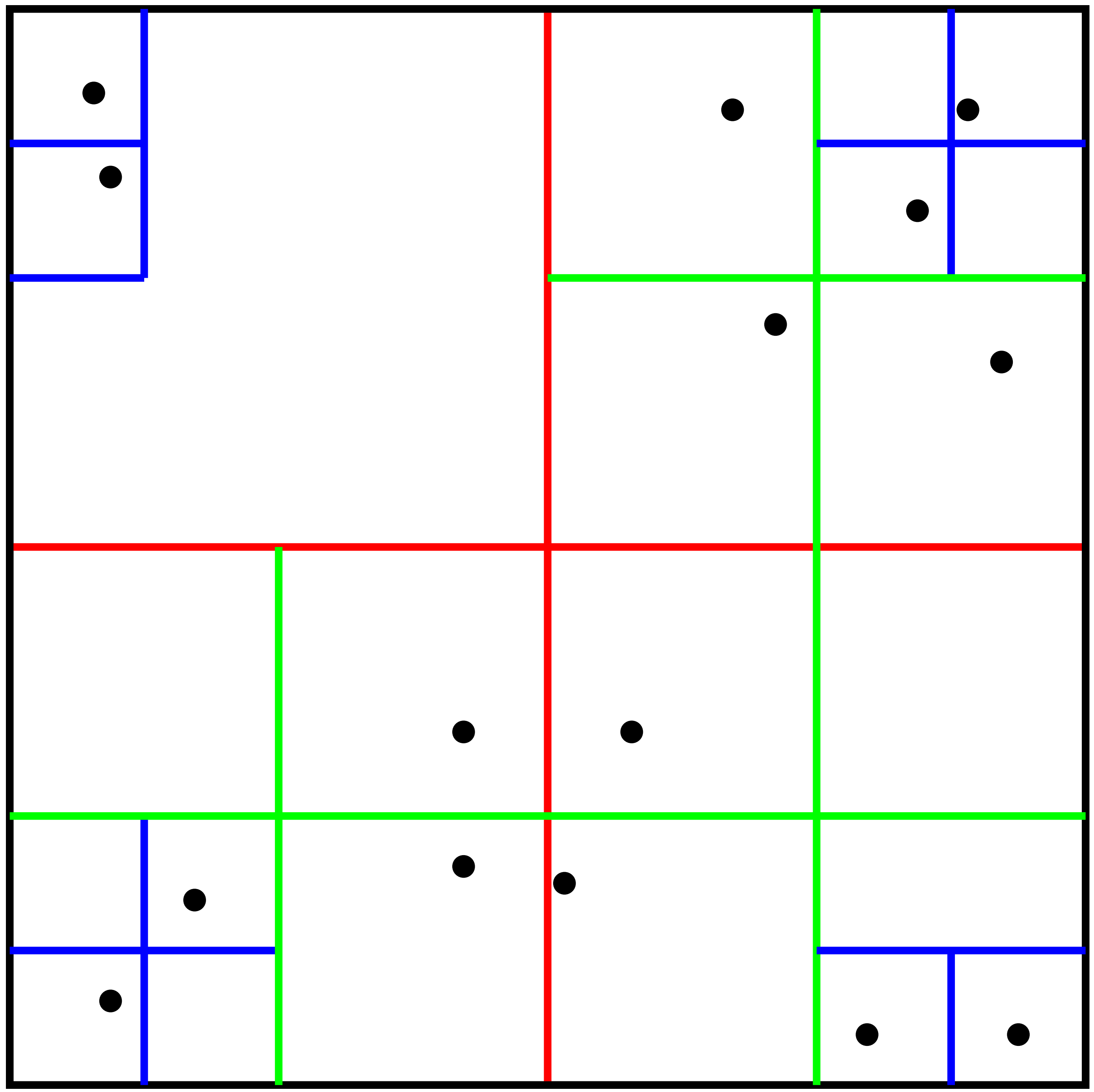
Compressed quadtrees

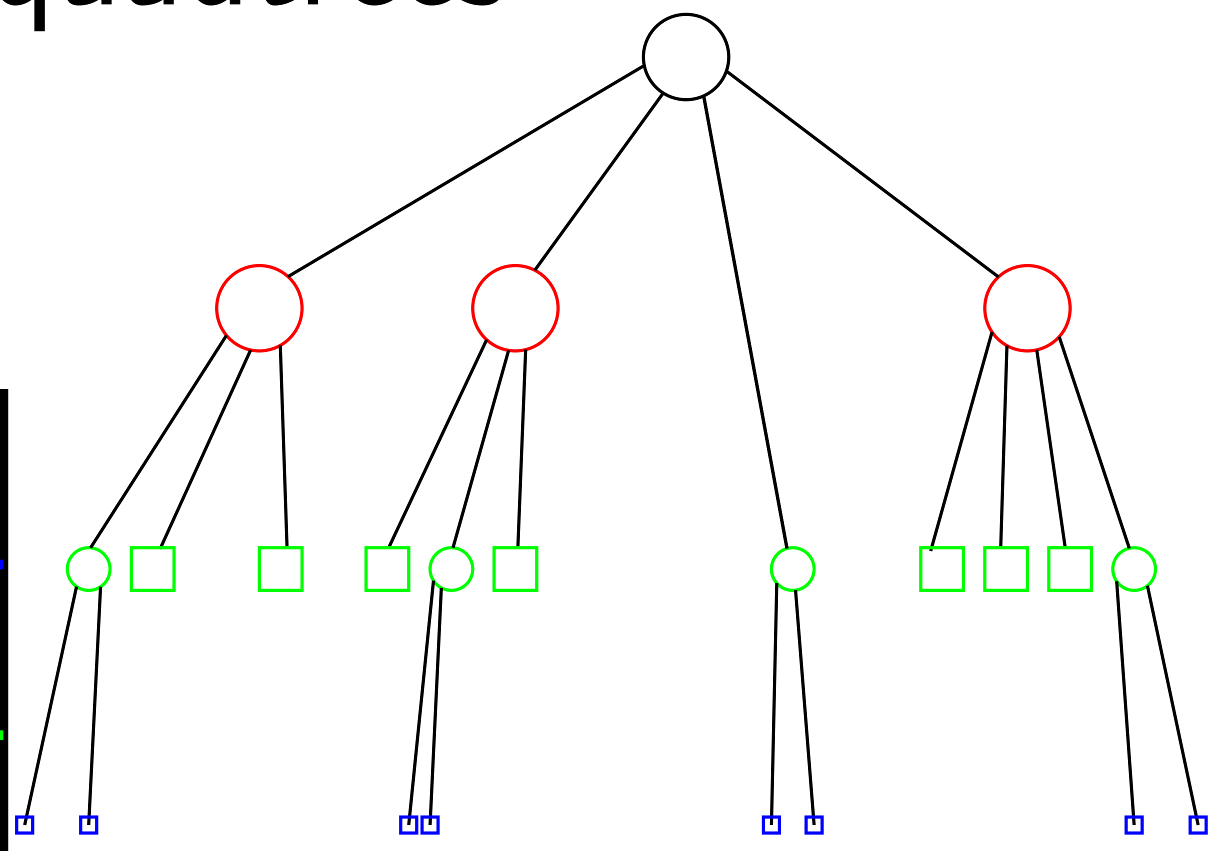
Pb Bounds on complexity depend on $\Phi(P)$.





Compressed quadtrees





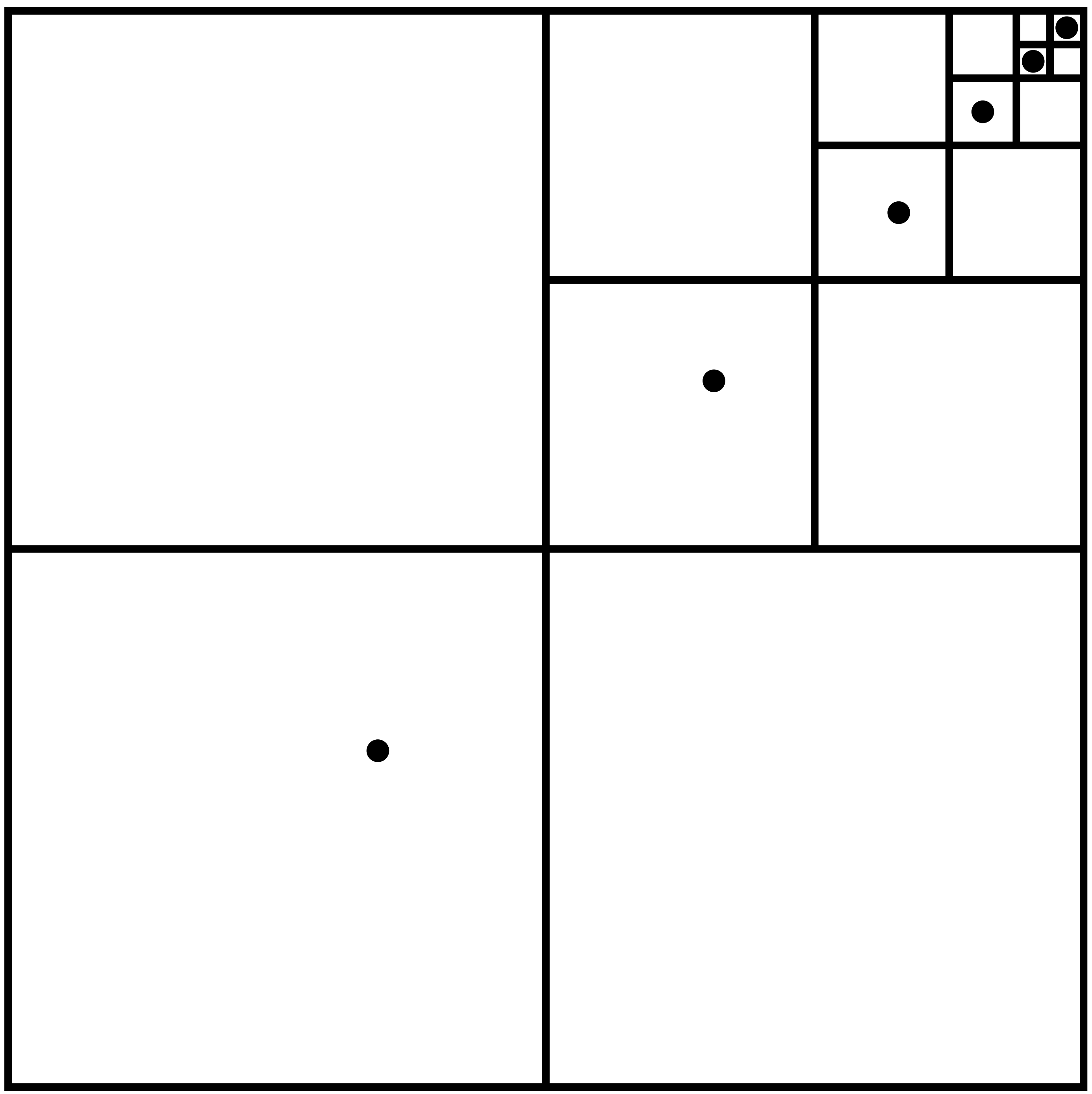
every internal node has ≥ 2 sons

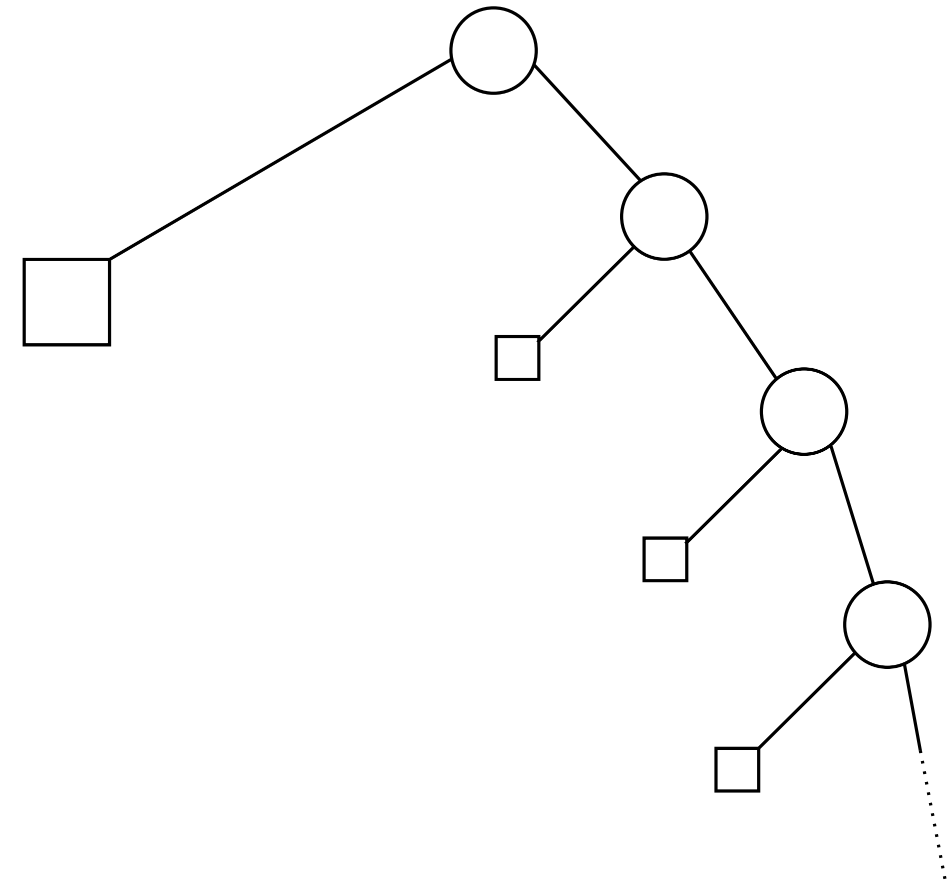
$$\Rightarrow |T| \leq 2|L| - 1 = 2|P| - 1$$

Q how to construct T efficiently?

Q how to locate a point efficiently?

Compressed quadtrees





every internal node has ≥ 2 sons

$$\Rightarrow |T| \leq 2|L| - 1 = 2|P| - 1$$

Q how to construct T efficiently?

Q how to locate a point efficiently?

Compressed quadtrees (construction)

Note Computing the uncompressed quadtree can take unbounded time

Quadratic algorithm:

1. For all pairs of points $(p, q) \in P^2$, find $\square_{v_{pq}} = \square_{v_i(p)} = \square_{v_i(q)}$, where $i = h(p, q) - 1$.

   $\rightarrow v_{pq}$ must be a node of compressed quadtree T

   $\rightarrow$ every node of T is a v_{pq} for some pair $(p, q) \in P^2$

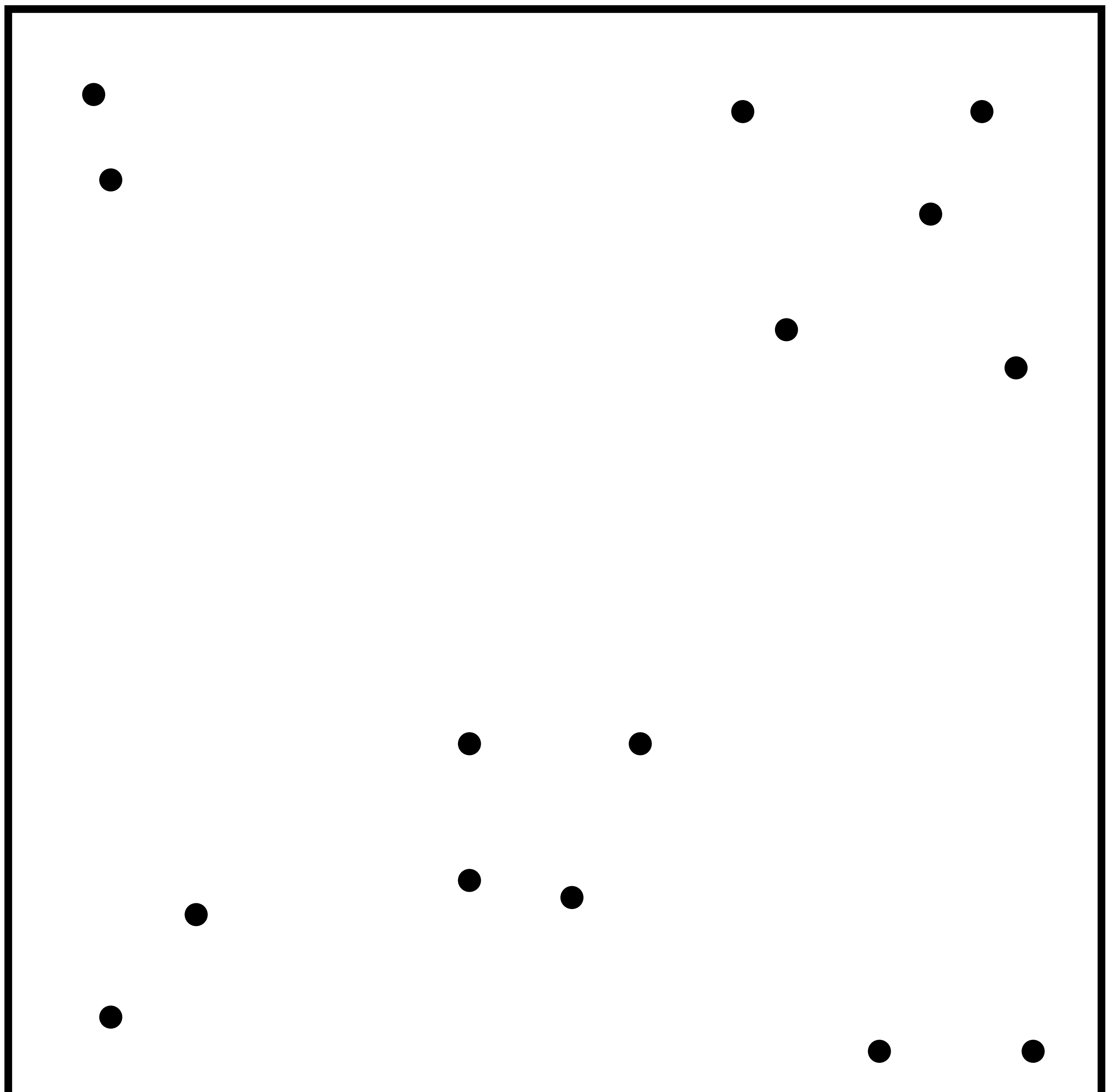
   $\Rightarrow$ this step computes the exact list of the nodes of T

2. For each node v in the list, find its most recent ancestor (in the list) and connect v to it.

   Note a node is stored only once in the list, although it may have been found multiple times in step 1 (use hash-table).

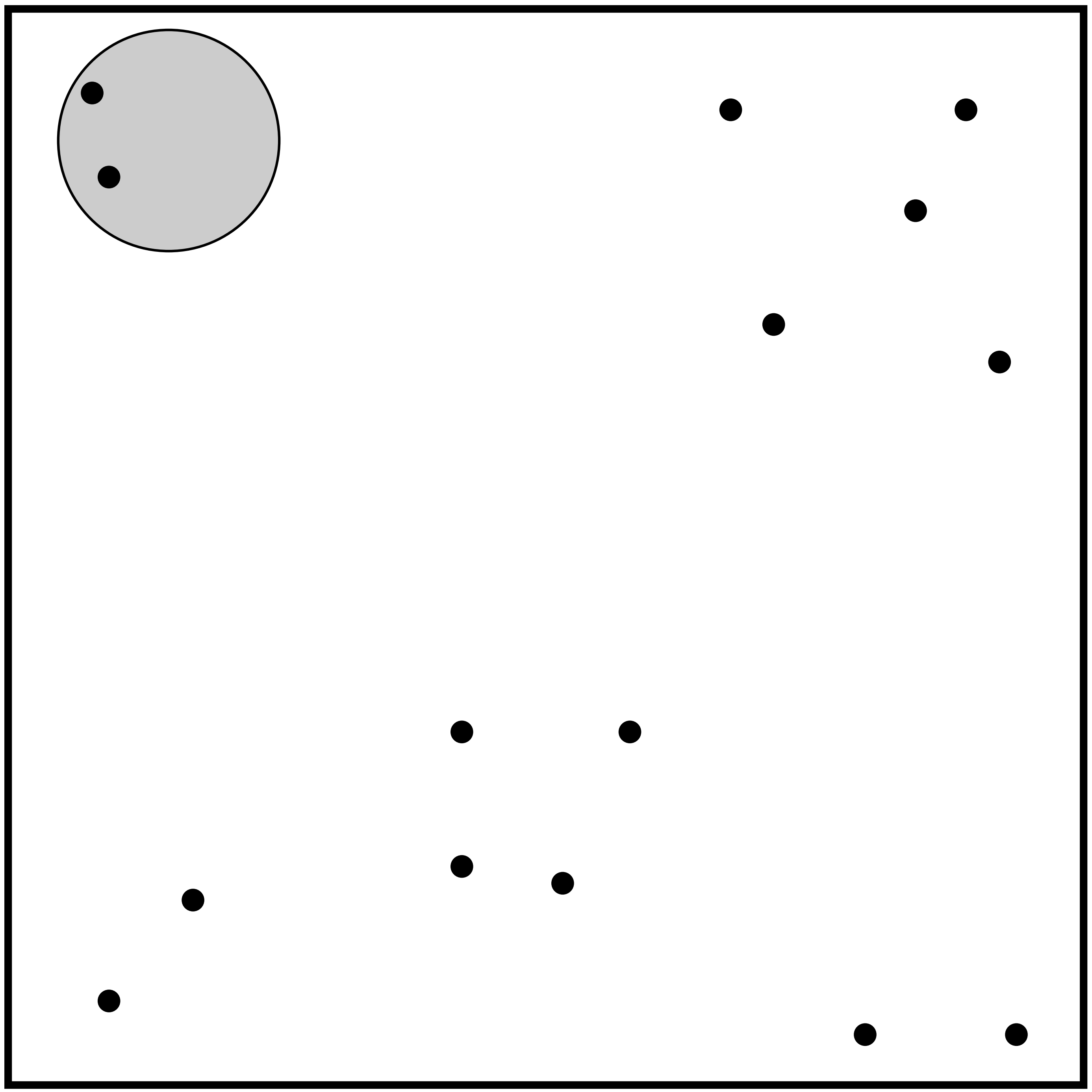
Compressed quadtrees (construction)

More subtle algorithm: let $k = |P|/10$.



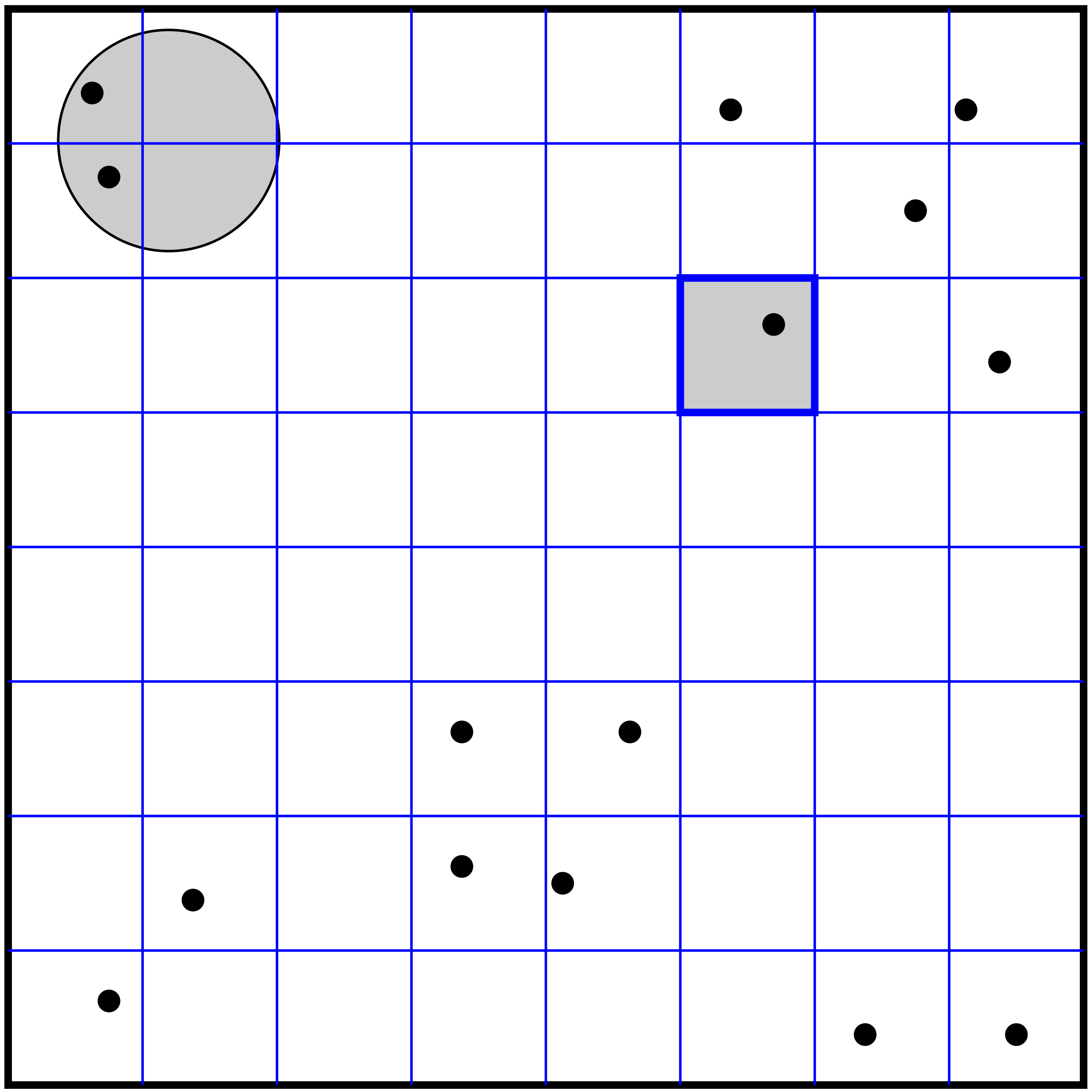
Compressed quadtrees (construction)

More subtle algorithm: let $k = |P|/10$. - Compute D_r s.t. $r_{\mathrm{opt}}(P, k) \leq r \leq 2\, r_{\mathrm{opt}}(P, k)$.



Compressed quadtrees (construction)

More subtle algorithm: let $k = |P|/10$.



- Compute D_r s.t. $r_{\text{opt}}(P,k) \leq r \leq 2\, r_{\text{opt}}(P,k)$.

- Let $l = 2^{\lfloor \log r \rfloor} \geq r/2$. Place the pts of P on UG_l, and find cell c with max number of points.

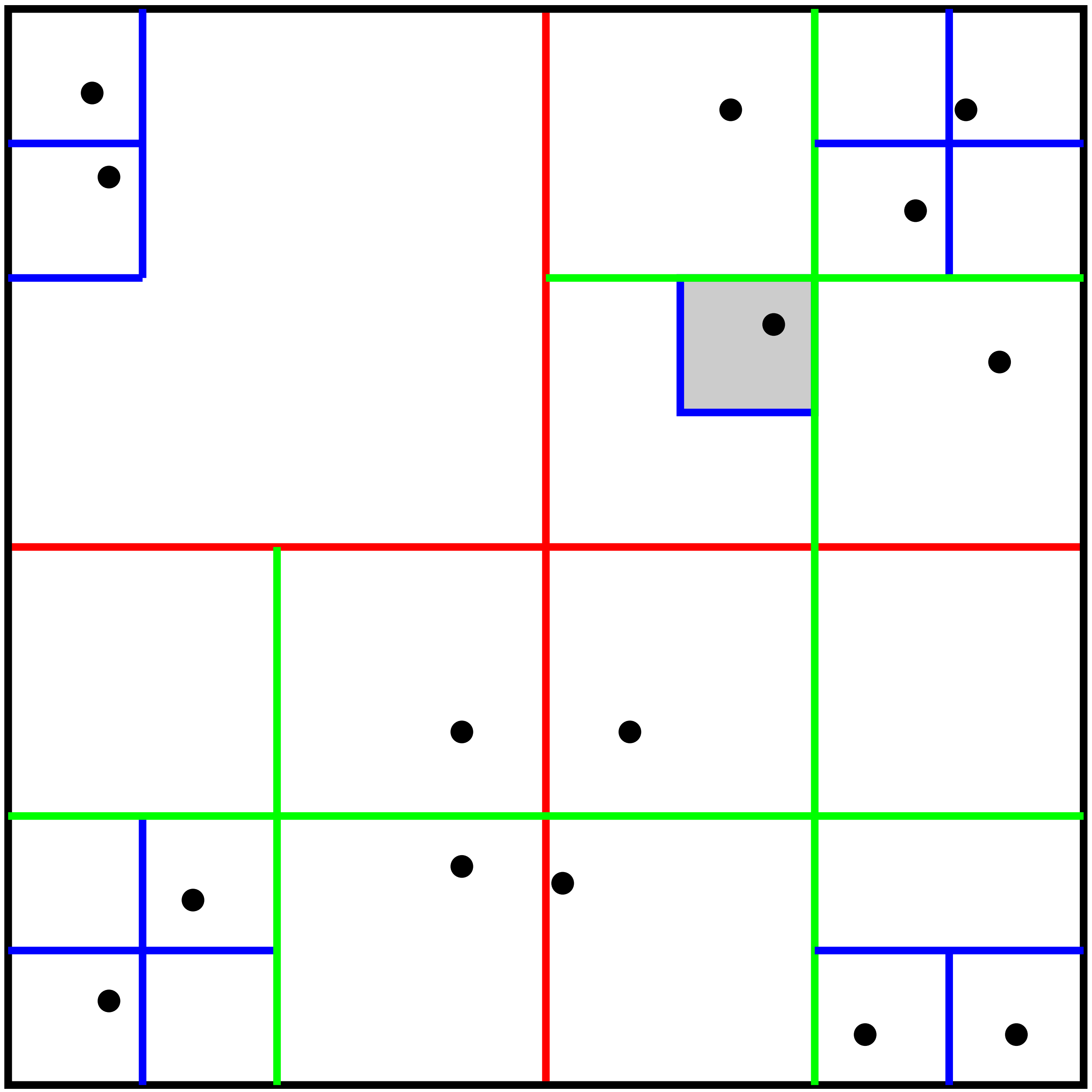
$P_{\text{in}} = P \cap c$, $P_{\text{out}} = P \setminus c$.

$l \geq \frac{r}{2} \Rightarrow |P_{\text{in}}| \geq \frac{k}{25} = \frac{|P|}{250}$.

$l \leq 2\, r_{\text{opt}}(P,k) \Rightarrow |P_{\text{in}}| \leq \frac{4|P|}{5}$.

Compressed quadtrees (construction)

More subtle algorithm: let $k = |P|/10$.



- Compute D_r s.t. $r_{\text{opt}}(P,k) \le r \le 2\, r_{\text{opt}}(P,k)$.
- Let $l = 2^{\lfloor \log r \rfloor} \ge r/2$. Place the pts of P on UG_l, and find cell c with max number of points.

$P_{\text{in}} = P \cap c$, $P_{\text{out}} = P \setminus c$.

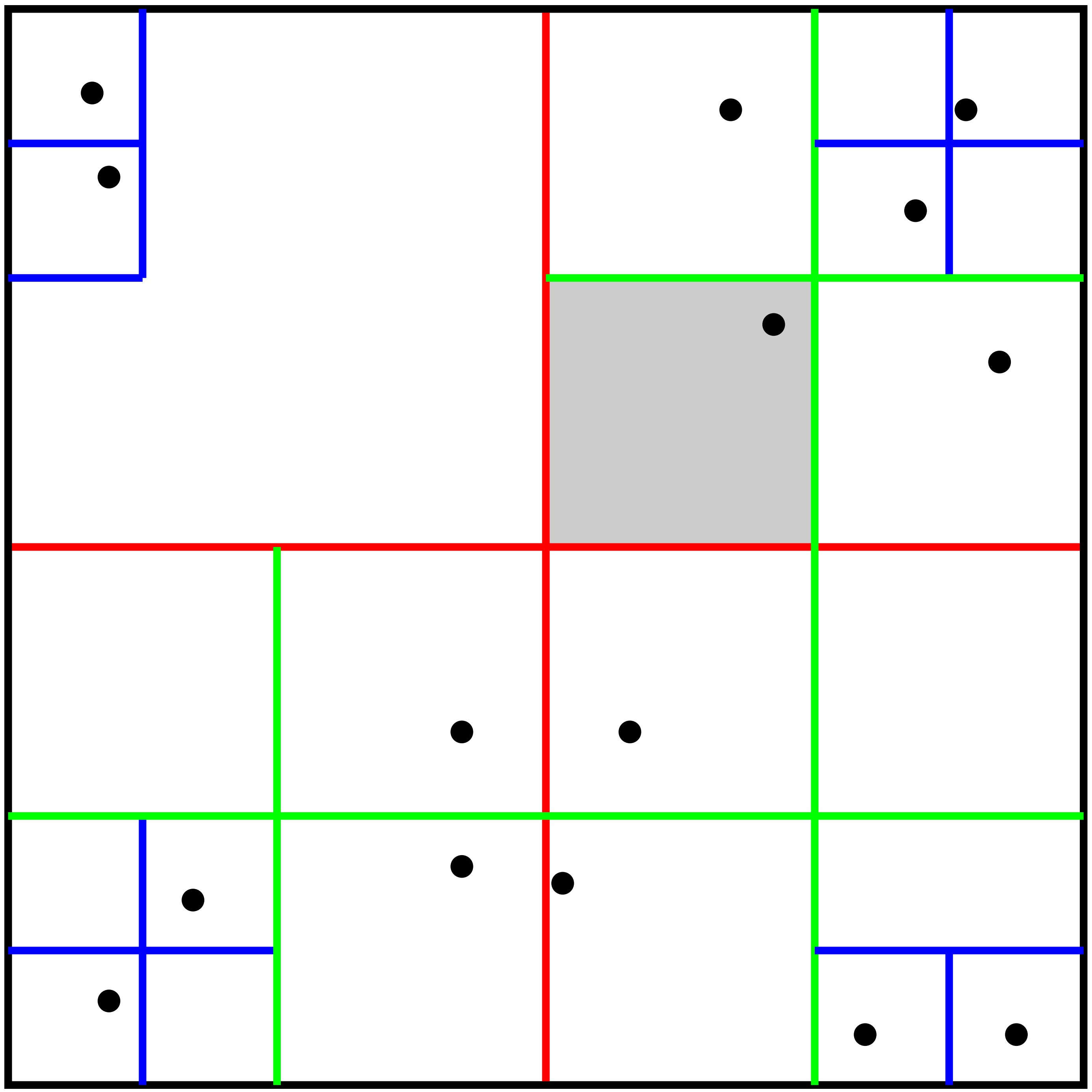
$l \ge \frac{r}{2} \Rightarrow |P_{\text{in}}| \ge \frac{k}{25} = \frac{|P|}{250}$.

$l \le 2\, r_{\text{opt}}(P,k) \Rightarrow |P_{\text{in}}| \le \frac{4|P|}{5}$.

- Recursive call on P_{in} and P_{out}.

Compressed quadtrees (construction)

More subtle algorithm: let $k = |P|/10$.



- Compute D_r s.t. $r_{\text{opt}}(P,k) \le r \le 2\, r_{\text{opt}}(P,k)$.

- Let $l = 2^{\lfloor \log r \rfloor} \ge r/2$. Place the pts of P on UG_l, and find cell c with max number of points.

$P_{\text{in}} = P \cap c$, $P_{\text{out}} = P \setminus c$.

$l \ge \frac{r}{2} \Rightarrow |P_{\text{in}}| \ge \frac{k}{25} = \frac{|P|}{250}$.

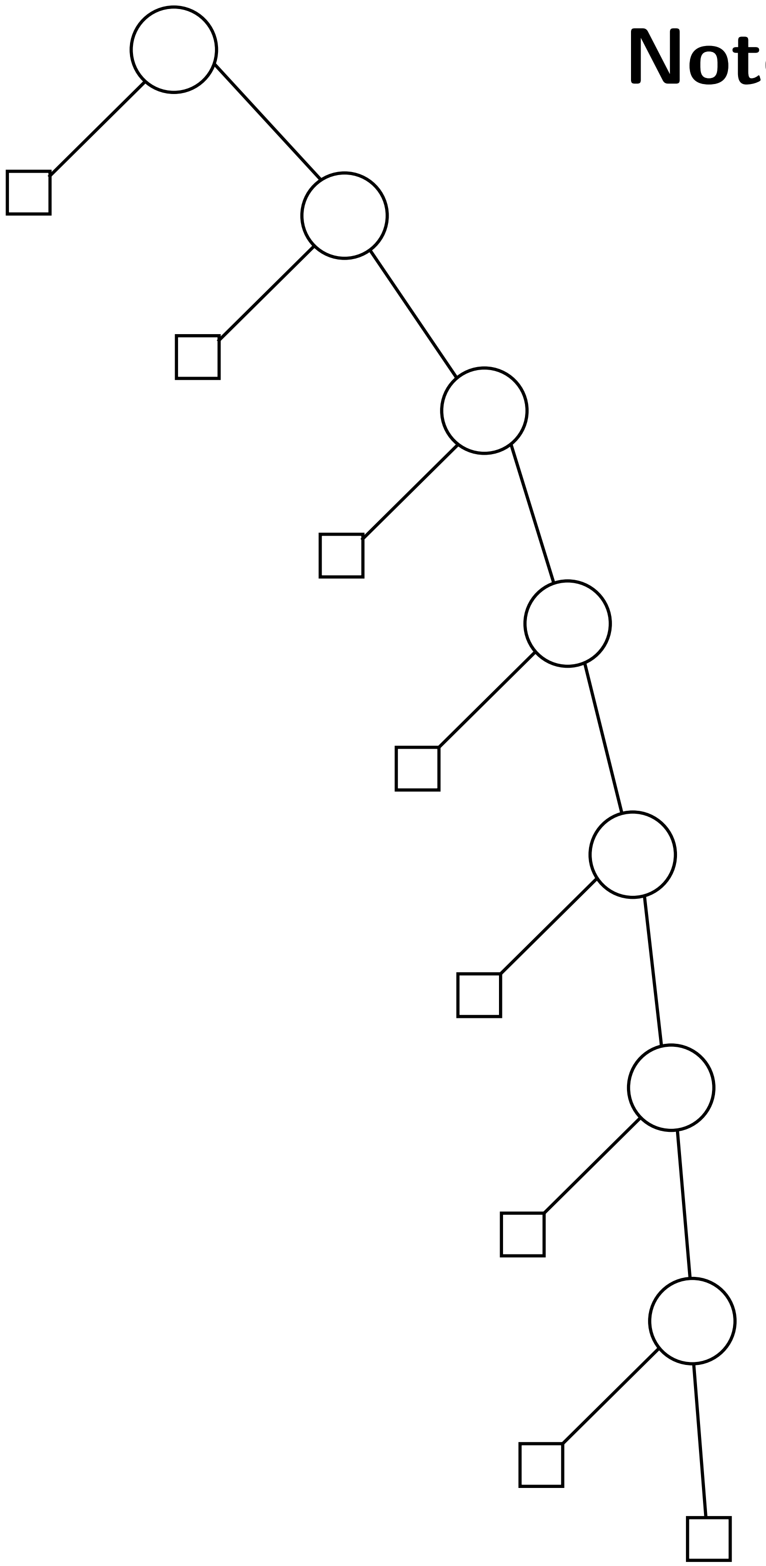
$l \le 2\, r_{\text{opt}}(P,k) \Rightarrow |P_{\text{in}}| \le \frac{4|P|}{5}$.

- Recursive call on P_{in} and P_{out}.

Locate any $p \in P_{\text{in}}$ in T_{out}, and hang root of T_{in} onto the node.

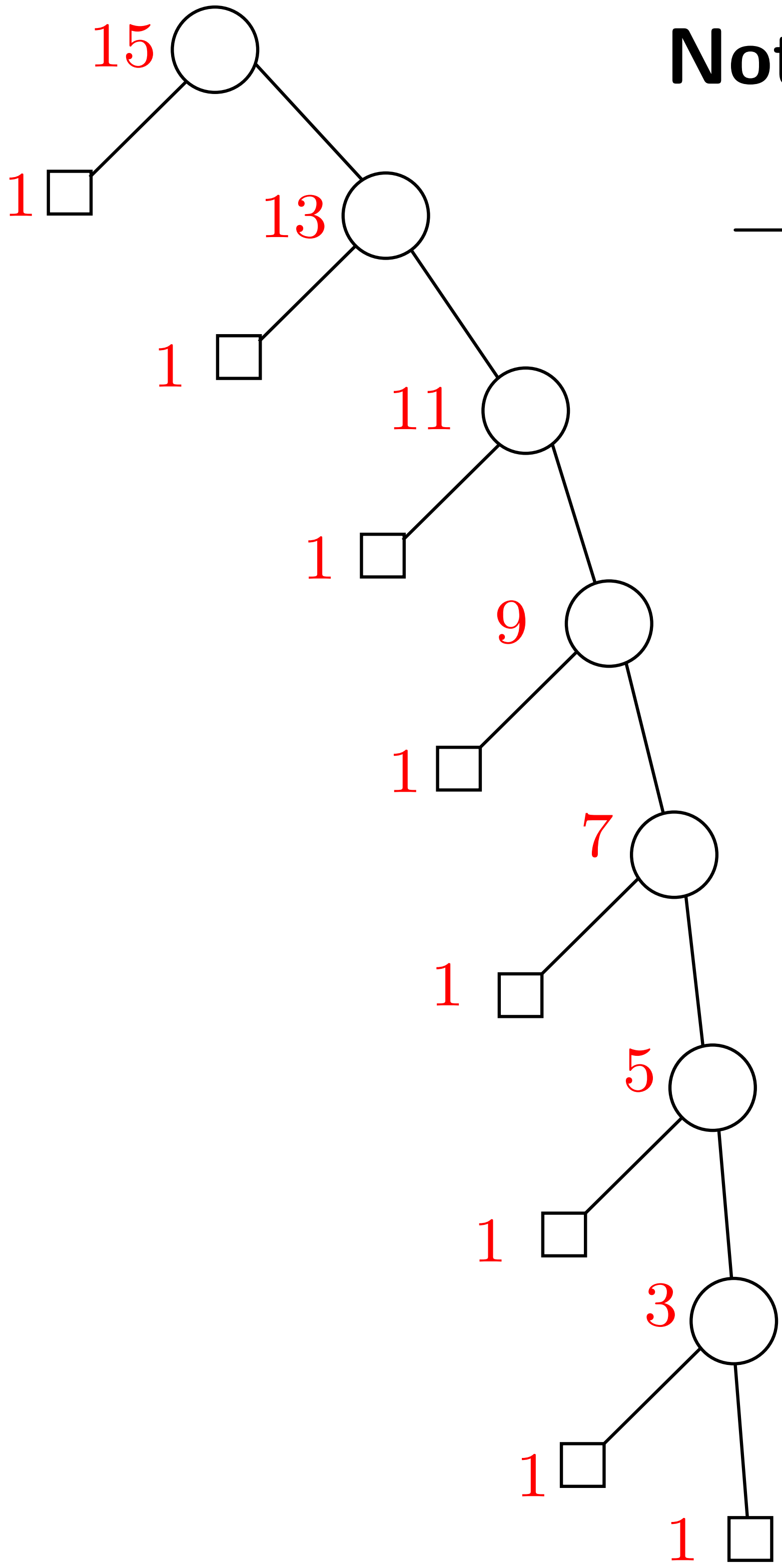
$\Rightarrow O(|P| \log |P|)$

Compressed quadtrees (pt location)



Note If T unbalanced, then query time $= \Omega(|P|)$.

Compressed quadtrees (pt location)

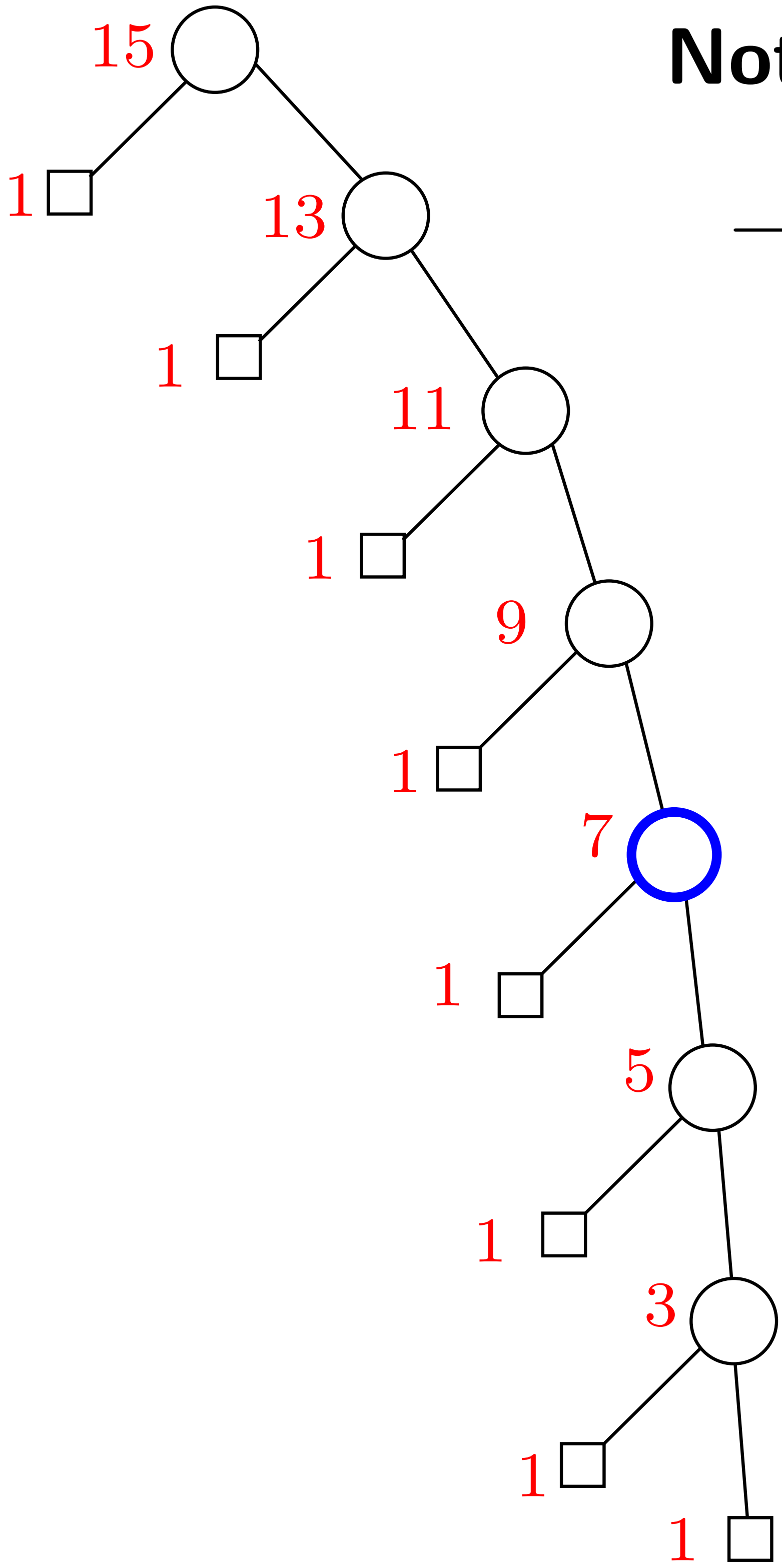


Note If T unbalanced, then query time $= \Omega(|P|)$.

$\rightarrow$ Finger tree construction:

Preprocessing: compute sizes of subtrees of T.

Compressed quadtrees (pt location)



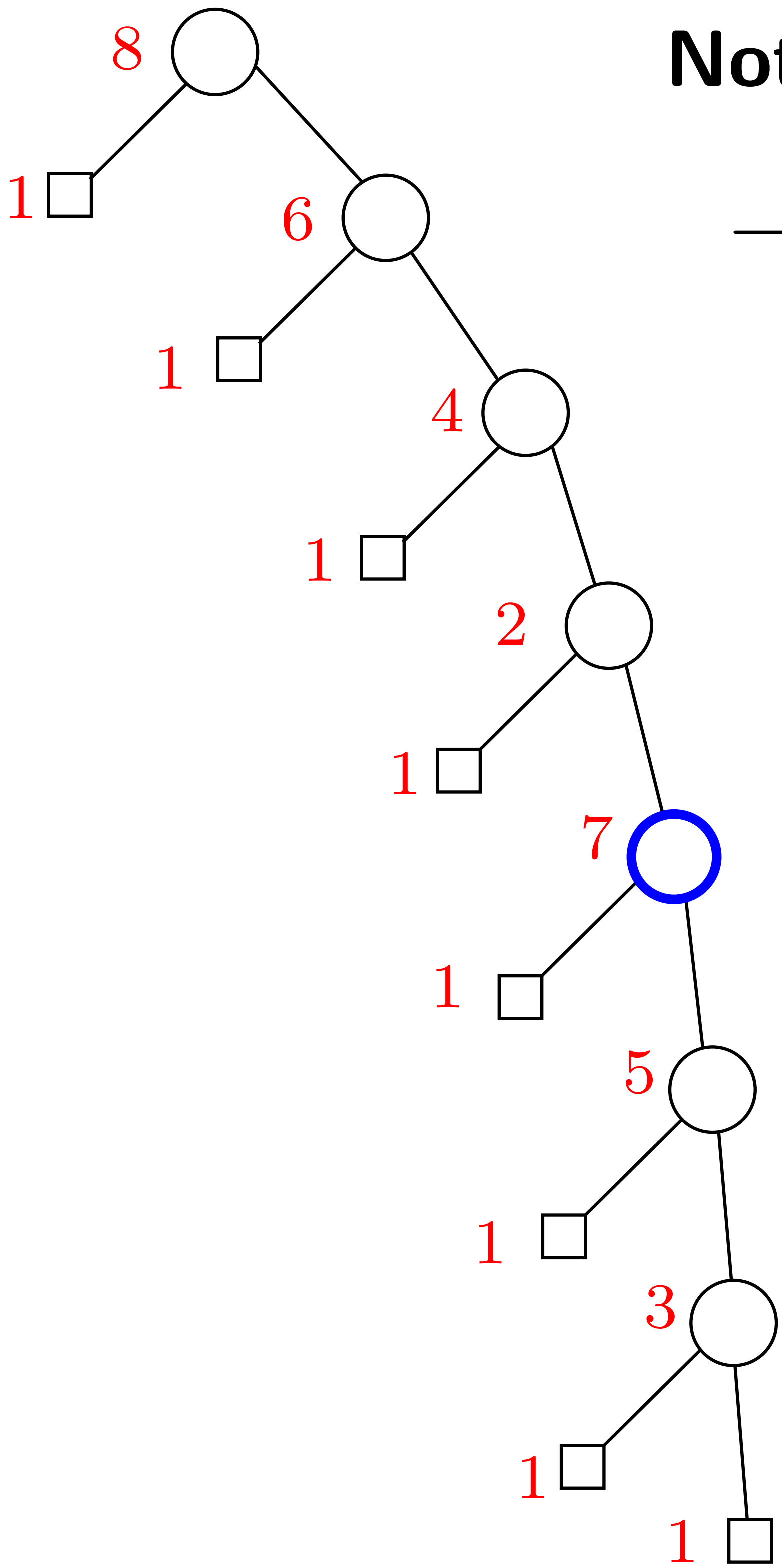
Note If T unbalanced, then query time $= \Omega(|P|)$.

$\rightarrow$ Finger tree construction:

Preprocessing: compute sizes of subtrees of T.

- starting from root of T, keep going to son with highest number, until subtree has size at most $^{|T|}/_{2}$.

Compressed quadtrees (pt location)



Note If T unbalanced, then query time $= \Omega(|P|)$.

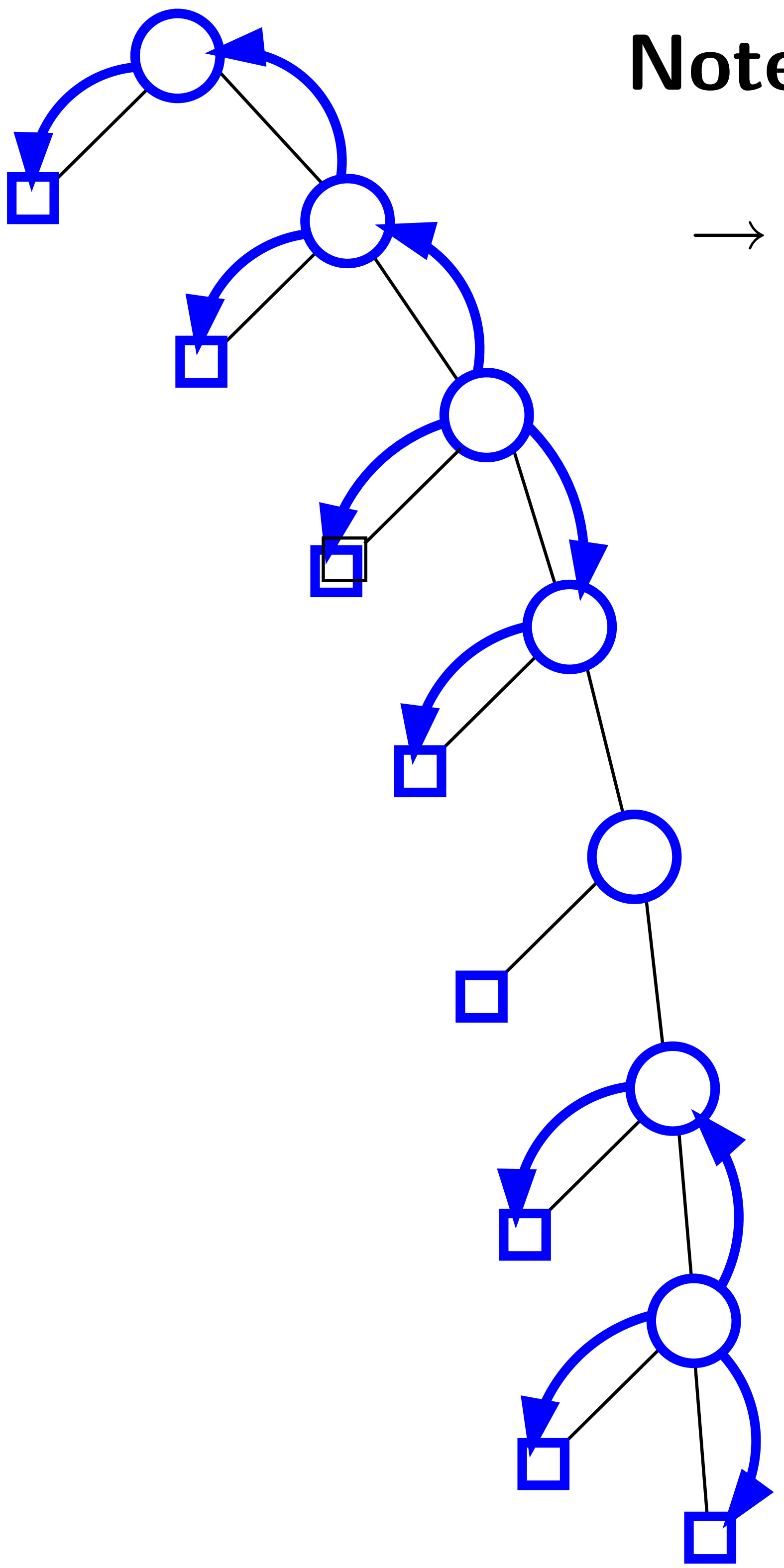
$\rightarrow$ Finger tree construction:

Preprocessing: compute sizes of subtrees of T.

- starting from root of T, keep going to son with highest number, until subtree has size at most $|T|/2$.

- Update numbers of parents of the *separator* node.

Compressed quadtrees (pt location)



Note If T unbalanced, then query time $= \Omega(|P|)$.

$\rightarrow$ Finger tree construction:

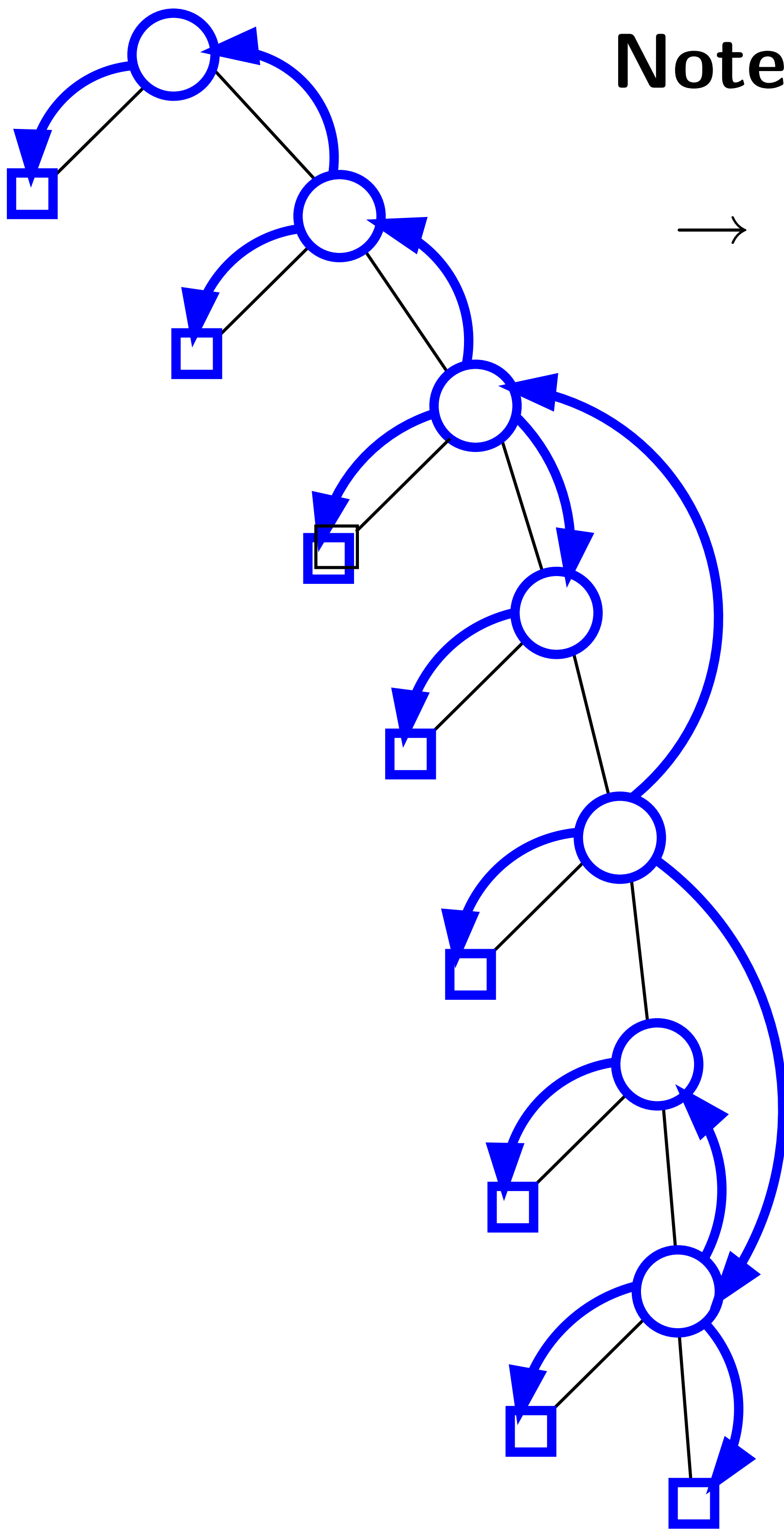
Preprocessing: compute sizes of subtrees of T.

- starting from root of T, keep going to son with highest number, until subtree has size at most $|T|/2$.

- Update numbers of parents of the *separator* node.

- Recursive call on all subtrees.

Compressed quadtrees (pt location)



Note If T unbalanced, then query time $= \Omega(|P|)$.

$\rightarrow$ Finger tree construction:

Preprocessing: compute sizes of subtrees of T.

- starting from root of T, keep going to son with highest number, until subtree has size at most $^{|T|}/_{2}$.

- Update numbers of parents of the *separator* node.

- Recursive call on all subtrees.

- Hang finger trees of subtrees to separator node.

$\rightarrow$ Construction time: $O(|T| \log |T|) = O(|P| \log |P|)$.

$\rightarrow$ Finger tree has same size as T and is balanced, hence its height is $O(\log |T|)$.

$\Rightarrow$ pt location time: $O(\log |T|)$.

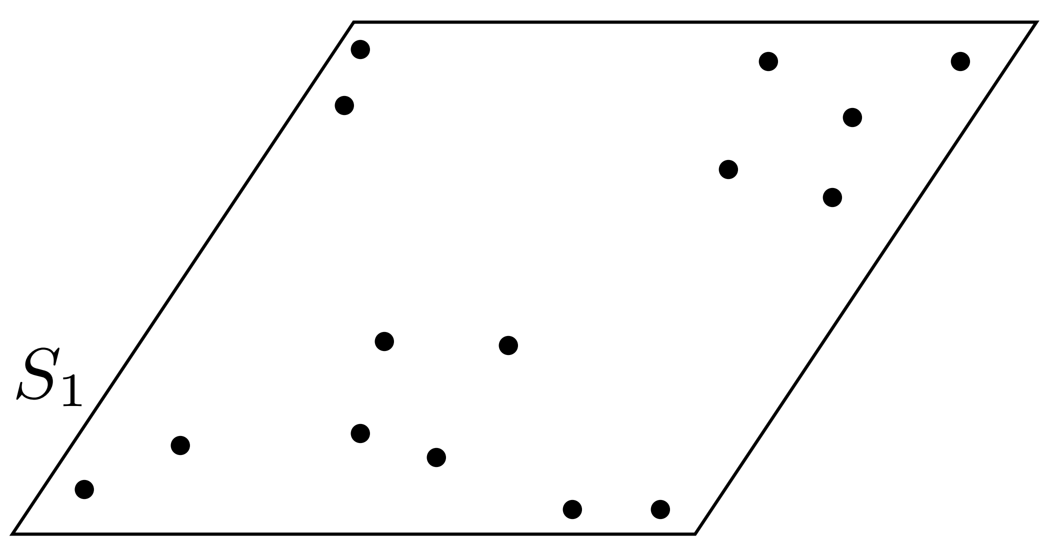
Dynamic quadtrees

Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.

Dynamic quadtrees

Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.

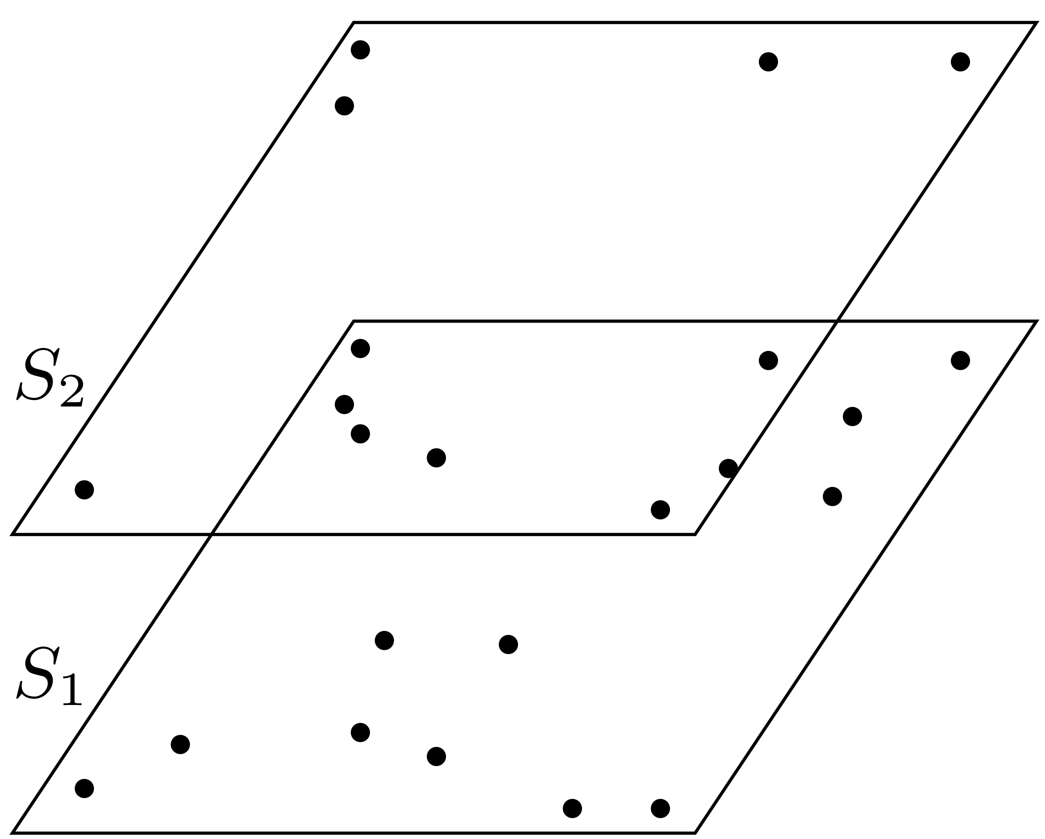
Def A *gradation* of P is a subsampling sequence $(S_m, S_{m-1}, \cdots, S_2, S_1)$ such that:
(i) $S_1 = P$,
(ii) S_i = pts of S_{i-1} picked with proba. $1/2$,
(iii) $|S_m| = 1 < |S_{m-1}|$.



Dynamic quadtrees

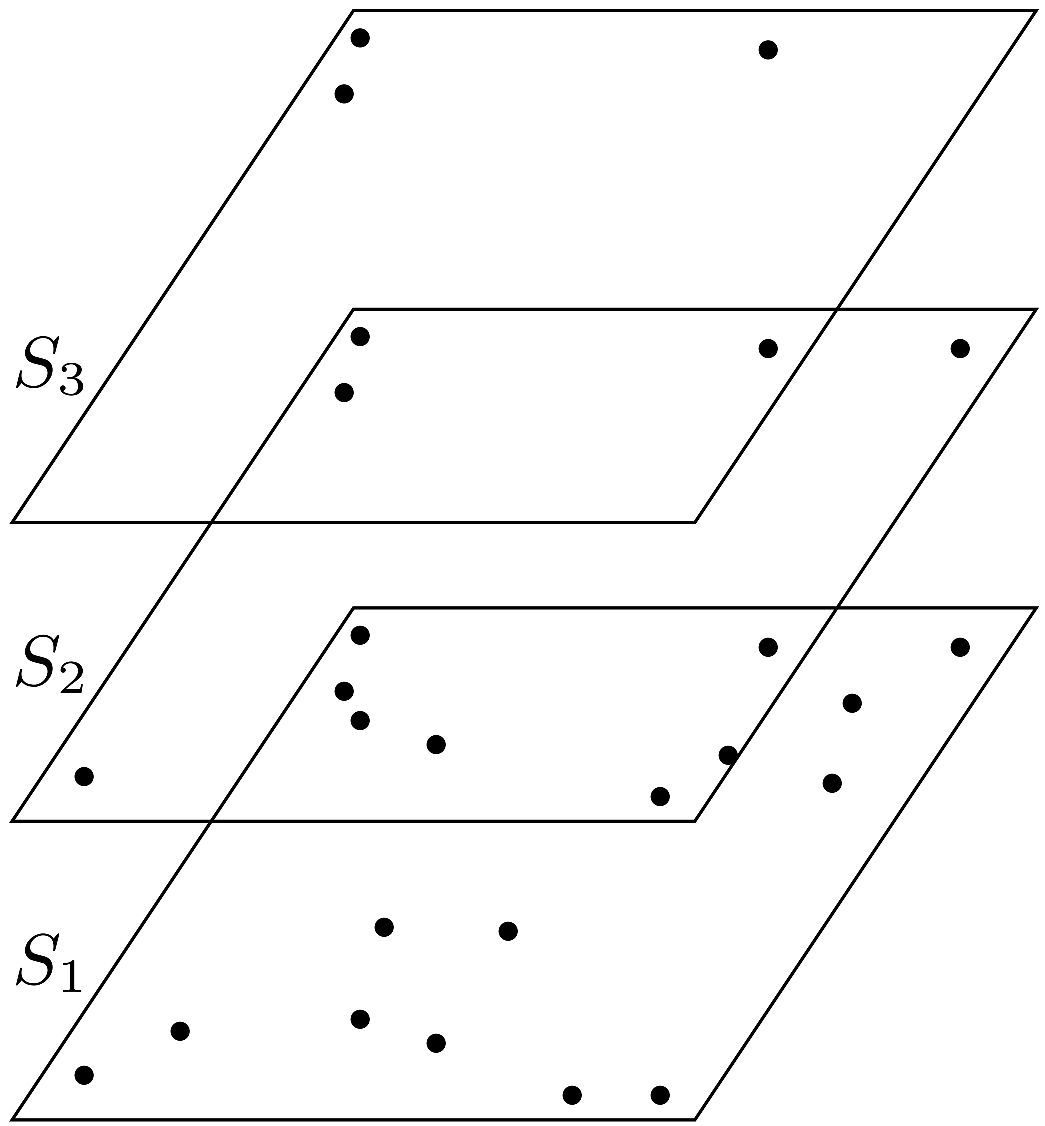
Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.

Def A *gradation* of P is a subsampling sequence $(S_m, S_{m-1}, \cdots, S_2, S_1)$ such that:
(i) $S_1 = P$,
(ii) $S_i =$ pts of S_{i-1} picked with proba. $1/2$,
(iii) $|S_m| = 1 < |S_{m-1}|$.



Dynamic quadtrees

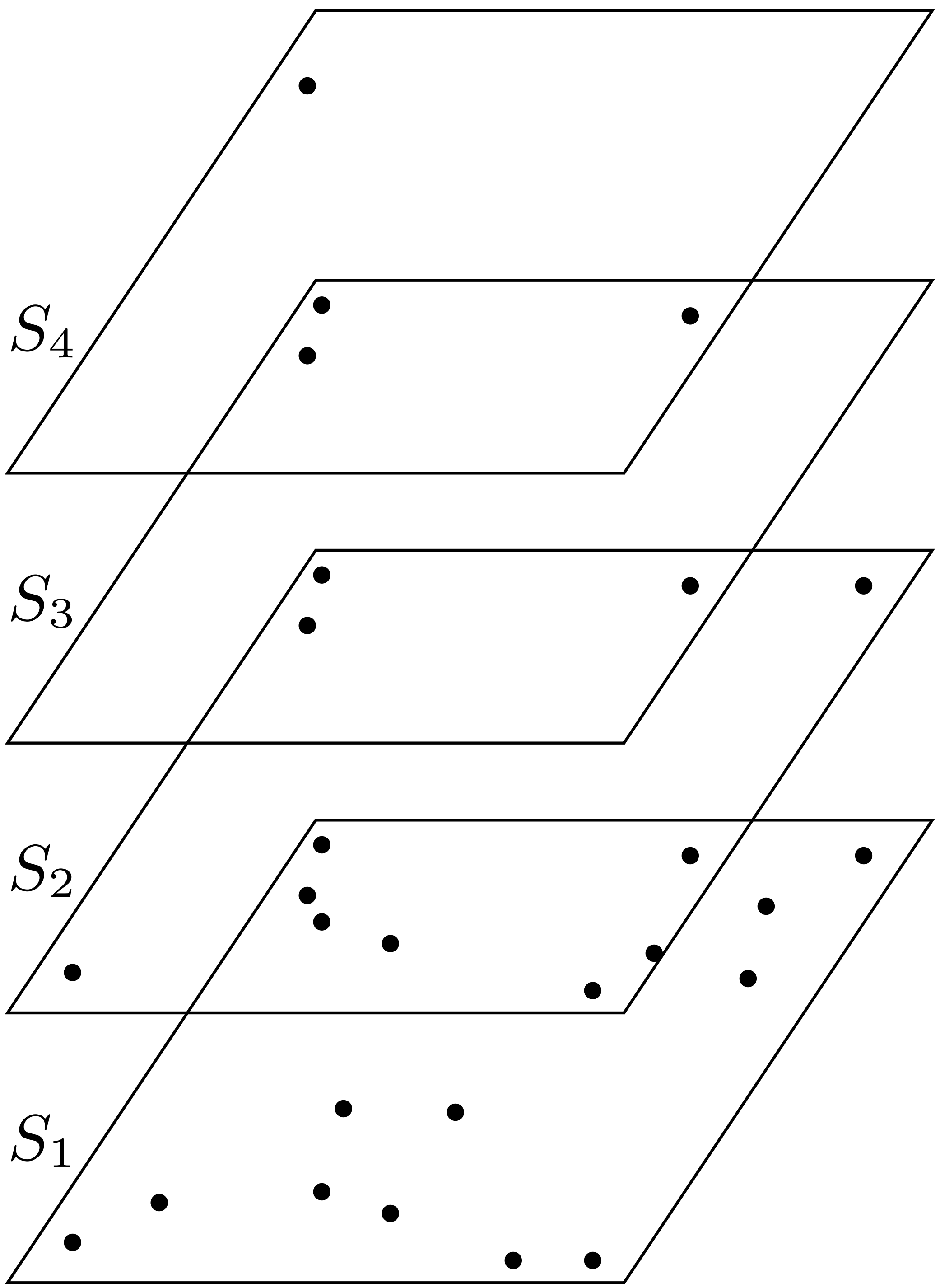
Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.



Def A *gradation* of P is a subsampling sequence $(S_m, S_{m-1}, \cdots, S_2, S_1)$ such that:
(i) $S_1 = P$,
(ii) $S_i =$ pts of S_{i-1} picked with proba. $1/2$,
(iii) $|S_m| = 1 < |S_{m-1}|$.

Dynamic quadtrees

Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.



Def A *gradation* of P is a subsampling sequence $(S_m, S_{m-1}, \cdots, S_2, S_1)$ such that:
(i) $S_1 = P$,
(ii) $S_i =$ pts of S_{i-1} picked with proba. $1/2$,
(iii) $|S_m| = 1 < |S_{m-1}|$.

Prop $\forall i$, $\mathbf{E}[|S_i|] = \frac{|S_{i-1}|}{2}$
$\Rightarrow \mathbf{E}[|S_i|] = \mathbf{E}[\mathbf{E}[|S_i|]] = \frac{1}{2}\mathbf{E}[|S_{i-1}|] = \cdots = \frac{1}{2^{i-1}}\mathbf{E}[|S_1|] = \frac{|P|}{2^{i-1}}$.

In particular, for $k = \lceil 11 \log |P| \rceil$, we have $\mathbf{E}[|S_k|] = \frac{|P|}{2^k} \leq \frac{|P|}{2^{11 \log |P|}} = \frac{1}{|P|^{10}}$

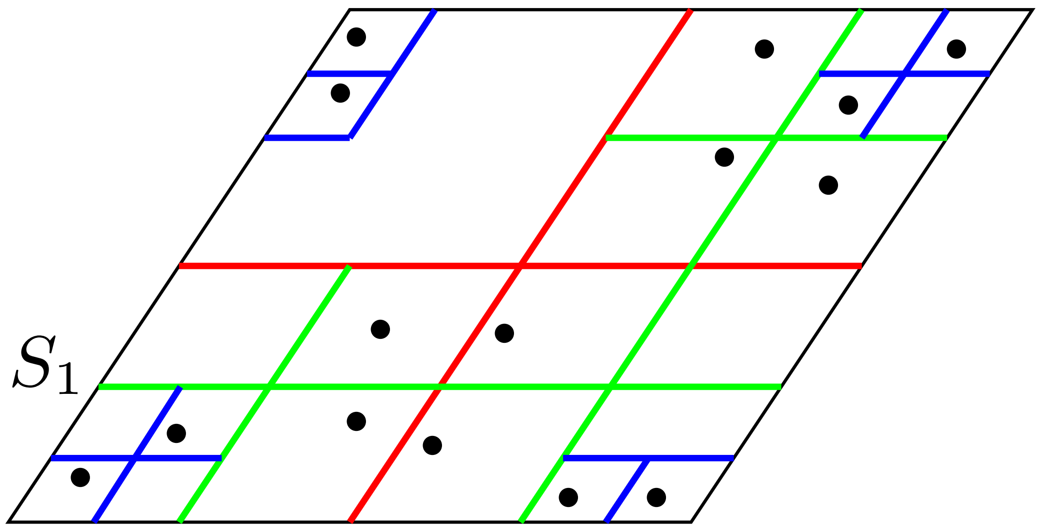
$\Rightarrow$ By Markov's inequality, $\mathbf{Pr}(m \geq k) = \mathbf{Pr}(|S_k| \geq 1) \leq \frac{\mathbf{E}(|S_k|)}{1} \leq \frac{1}{|P|^{10}}$.

$\Rightarrow$ with high proba., $m = O(\log |P|)$.

Dynamic quadtrees

Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.

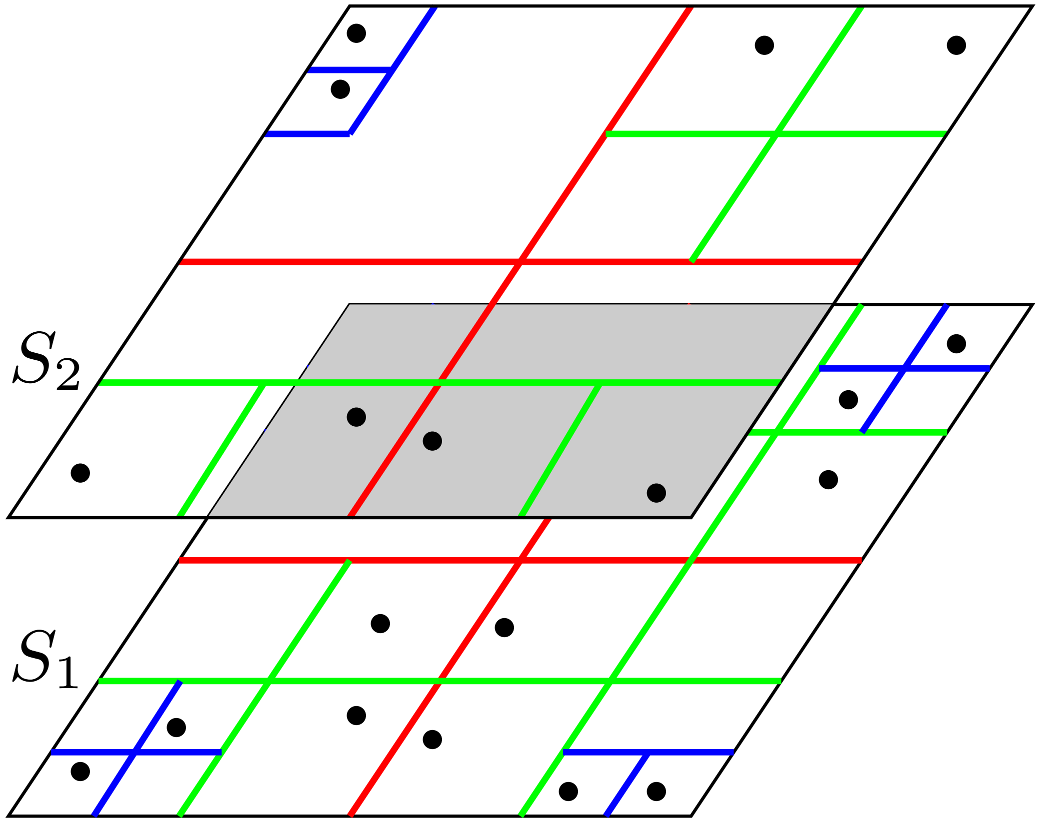
Def Given gradation $(S_m, \cdots, S_1 = P)$, build compressed quadtrees $T_i(S_i)$



Dynamic quadtrees

Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.

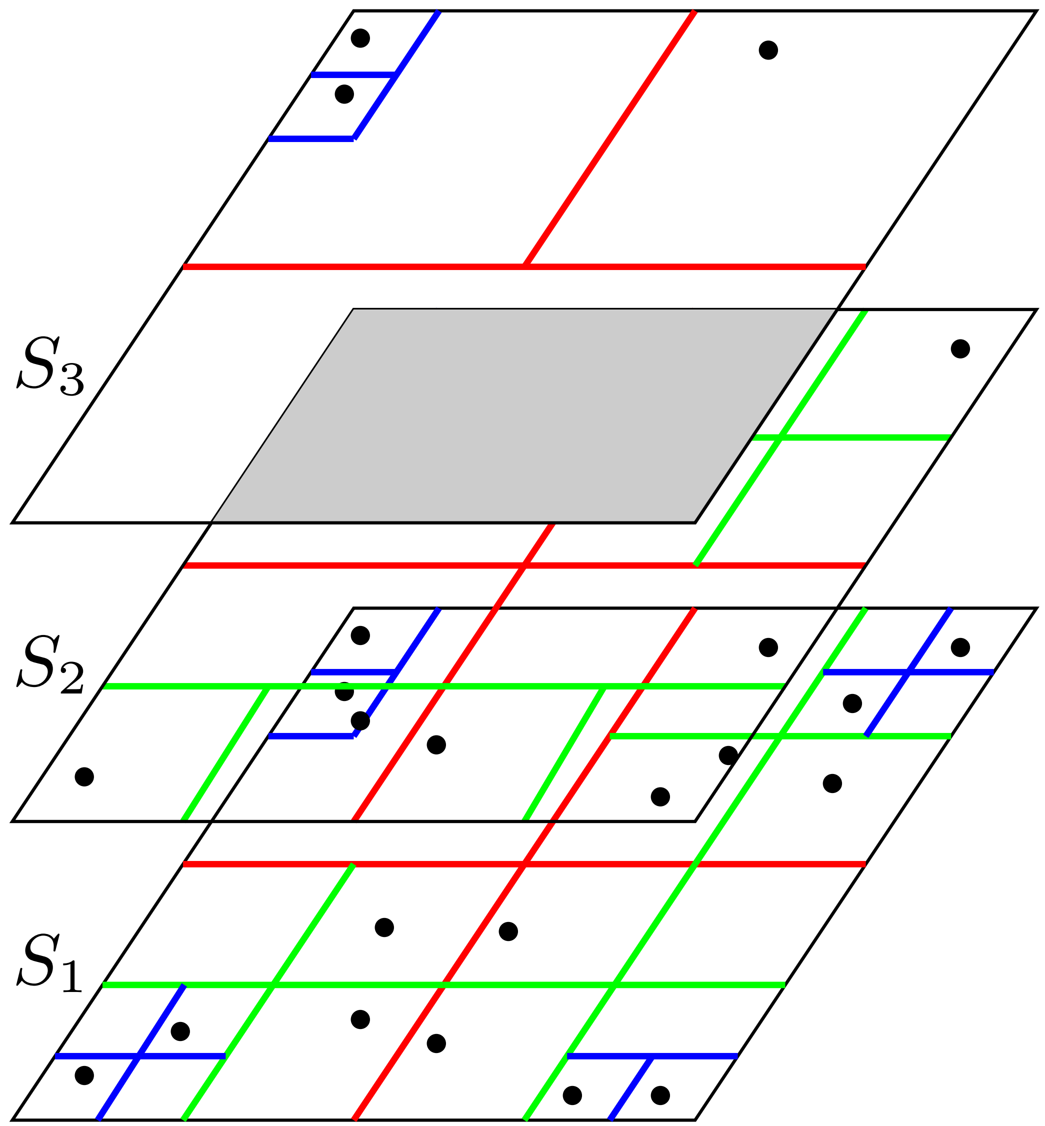
Def Given gradation $(S_m, \cdots, S_1 = P)$, build compressed quadtrees $T_i(S_i)$



Dynamic quadtrees

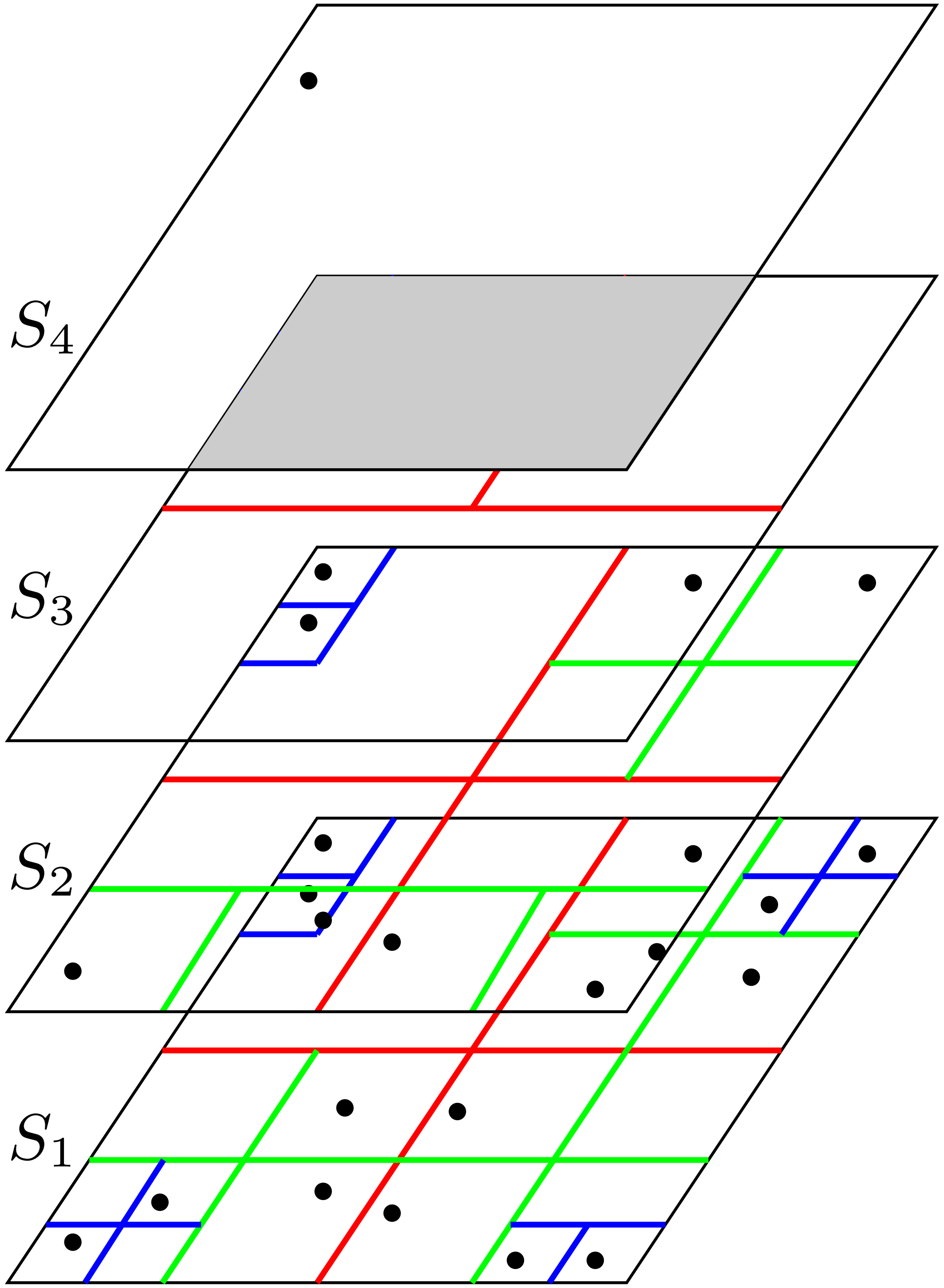
Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.

Def Given gradation $(S_m, \cdots, S_1 = P)$, build compressed quadtrees $T_i(S_i)$



Dynamic quadtrees

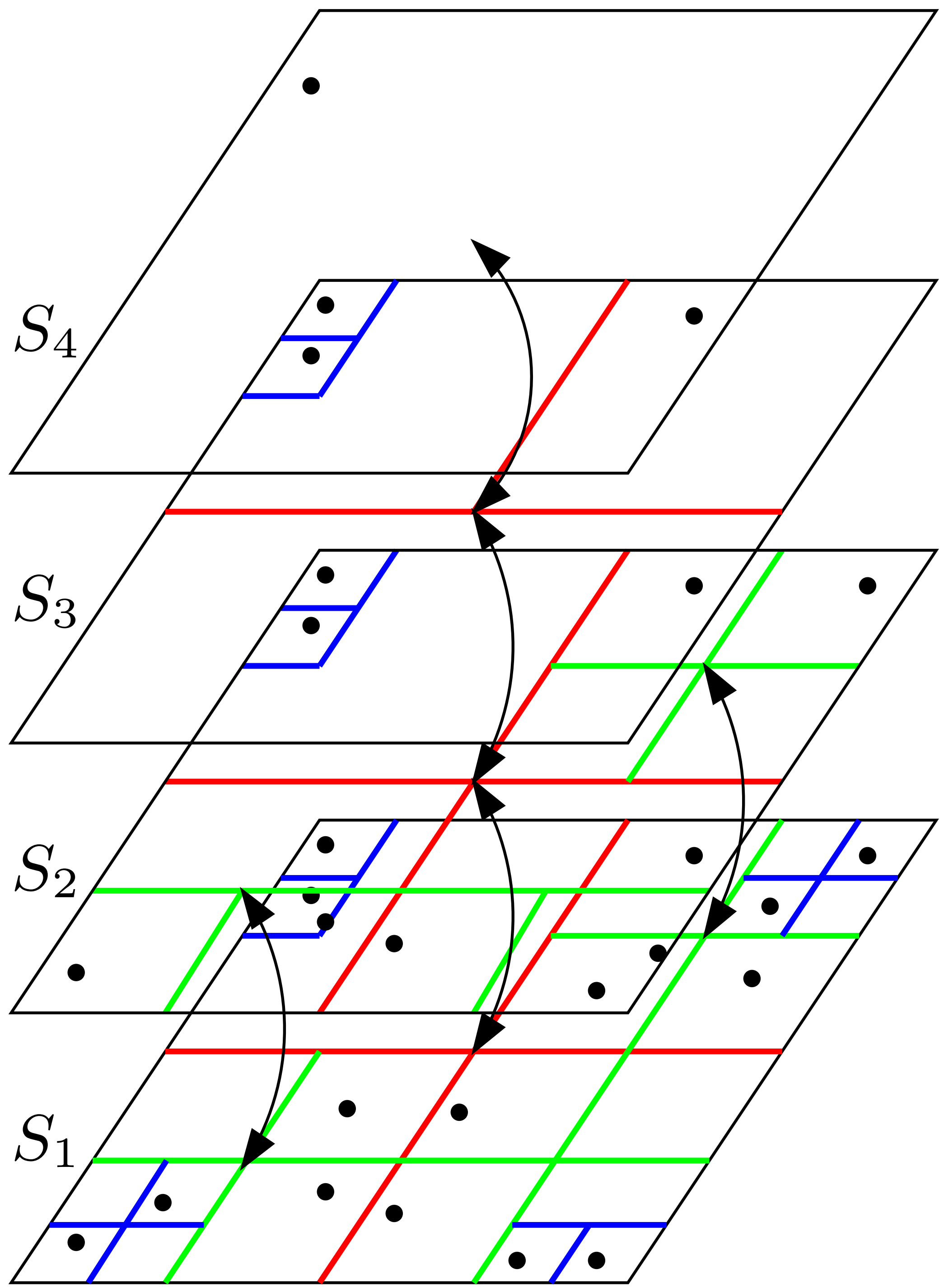
Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.



Def Given gradation $(S_m, \cdots, S_1 = P)$, build compressed quadtrees $T_i(S_i)$

Dynamic quadtrees

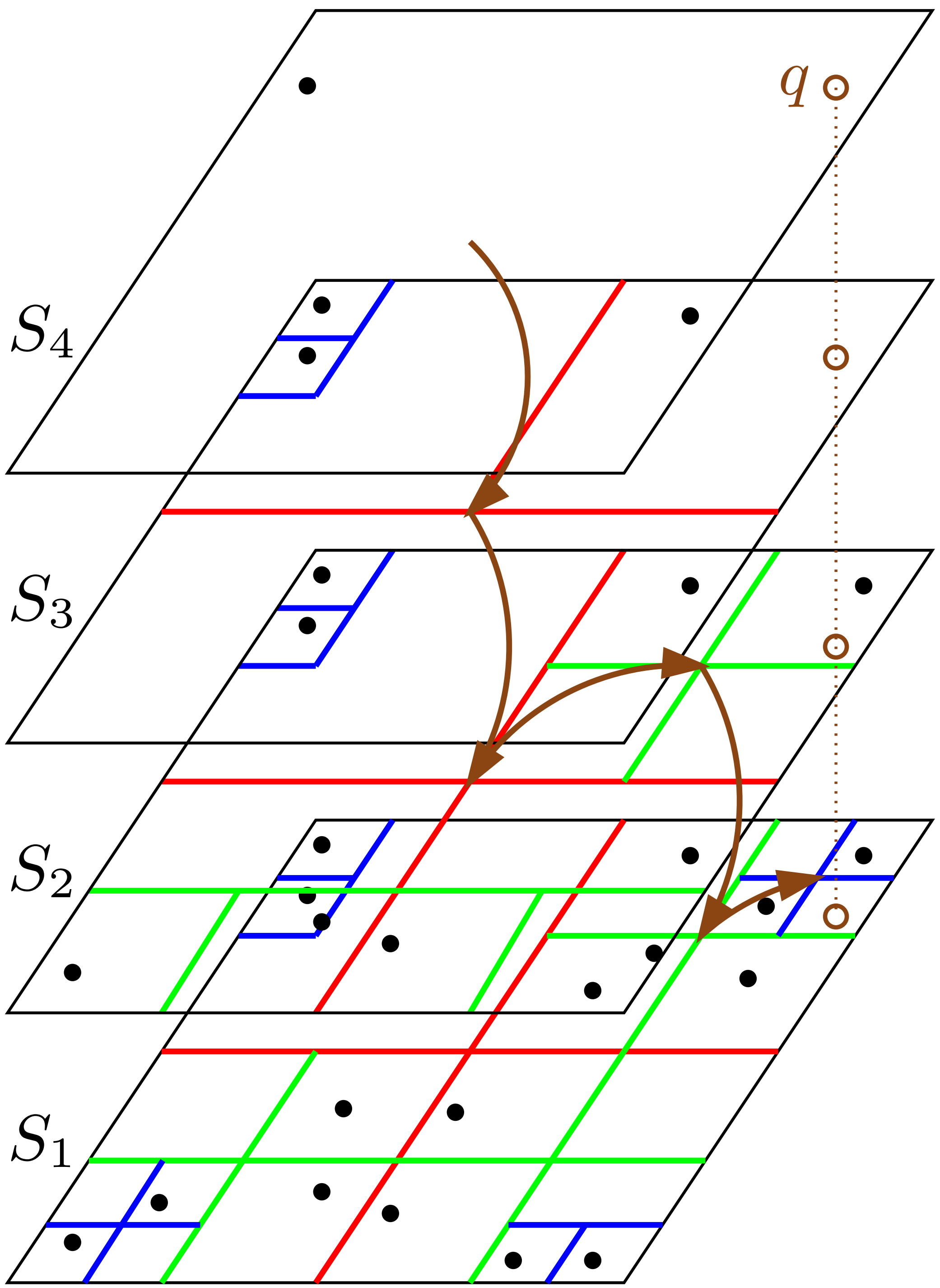
Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.



Def Given gradation $(S_m, \cdots, S_1 = P)$, build compressed quadtrees $T_i(S_i)$ and connect the internal nodes of S_i to their instances in S_{i-1}.
$\Rightarrow$ hierarchy of compressed quadtrees.

Dynamic quadtrees

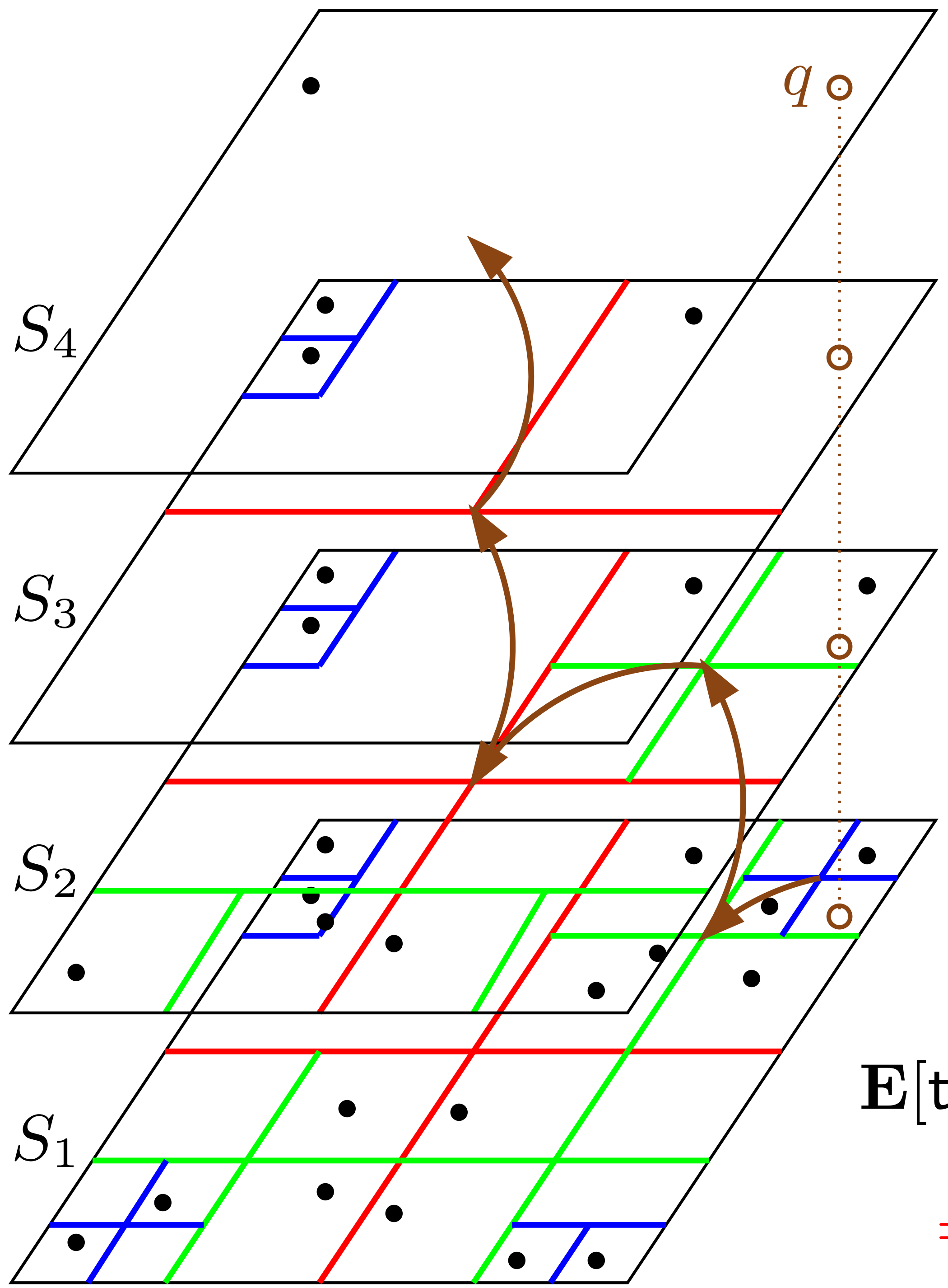
Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.



Pt location Given $q \in [0,1[^2$, locate q in T_m, then follow link of latest internal node v_m to T_{m-1}, then locate q in T_{m-1} from v_m, $\cdots$, locate q in T_1 from v_2.

Dynamic quadtrees

Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.



Pt location Given $q \in [0,1[^2$, locate q in T_m, then follow link of latest internal node v_m to T_{m-1}, then locate q in T_{m-1} from v_m, $\cdots$, locate q in T_1 from v_2.

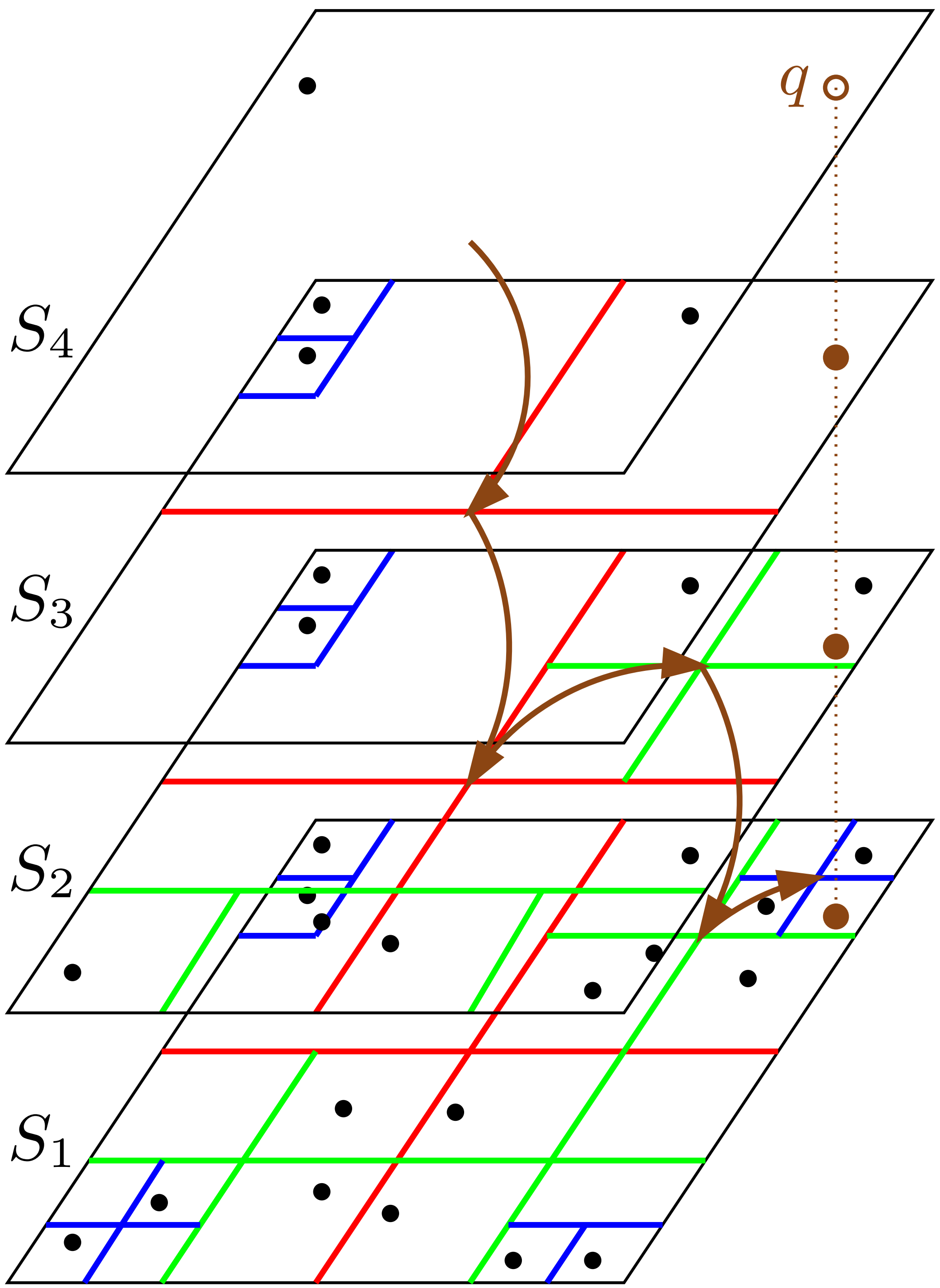
Backward analysis Let v be last node visited in T_i. Let $v_1 = v, v_2, \cdots, v_r$ be path to root. $\forall j$, $U_j := S_i \cap \square_{v_j} \rightarrow |U_j| \geq j$, and $[|U_j \cap S_{i+1}| \leq 1 \Leftrightarrow v_j \in T_i]$.
Let $V_j = 1$ iff $v_j \notin T_{i+1} \rightarrow$
$\mathbf{E}[V_j] = \mathbf{Pr}(V_j = 1) = \mathbf{Pr}(|U_j \cap S_{i+1}| \leq 1) = \frac{1}{2^{|U_j|}} + \frac{|U_j|}{2^{|U_j|-1}}\frac{1}{2} = \frac{1+|U_j|}{2^{|U_j|}} \leq \frac{1+j}{2^j}$.

$\mathbf{E}[\text{time spent in } T_i] \leq \sum_j \mathbf{E}[V_j] = \sum_j \frac{j+1}{2^j} = O(1)$.

$\Rightarrow$ $\mathbf{E}[\text{location time}] = O(\log |P|)$.

Dynamic quadtrees

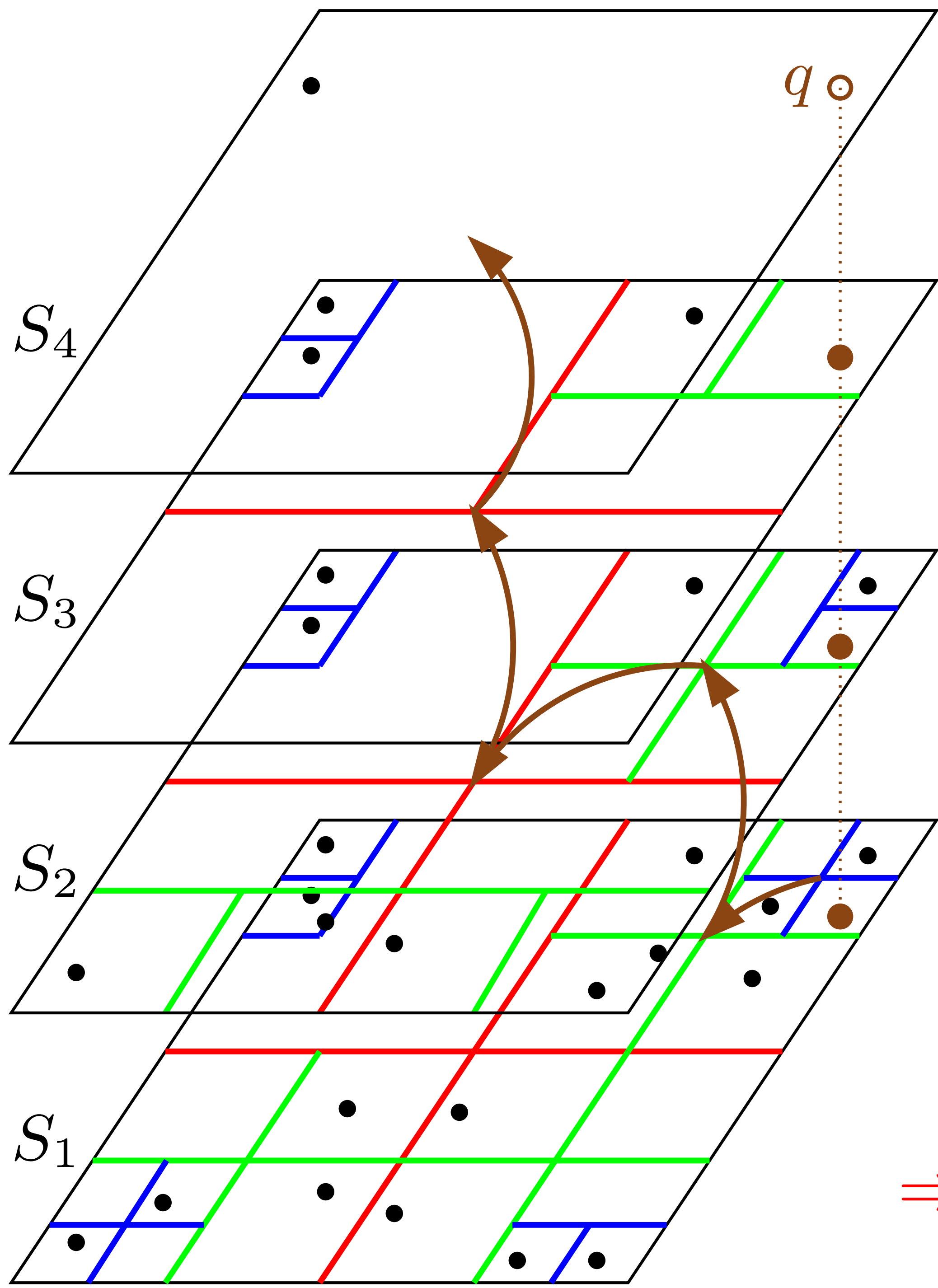
Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.



Pt insertion Given $q \in [0,1[^2$,
- locate q in $(T_m, \cdots, T_1)$ and store path.
- insert q in T_1 by splitting last node of path.
- toss coin: if neg. result, then done. Else, add q to S_2 and insert it in T_2 using last node of location path in T_2.
- iterate process, until coin toss gives neg. result (create new layers if necessary).

Dynamic quadtrees

Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.



Pt insertion Given $q \in [0,1[^2$,

- locate q in $(T_m, \cdots, T_1)$ and store path.
- insert q in T_1 by splitting last node of path.
- toss coin: if neg. result, then done. Else, add q to S_2 and insert it in T_2 using last node of location path in T_2.
- iterate process, until coin toss gives neg. result (create new layers if necessary).

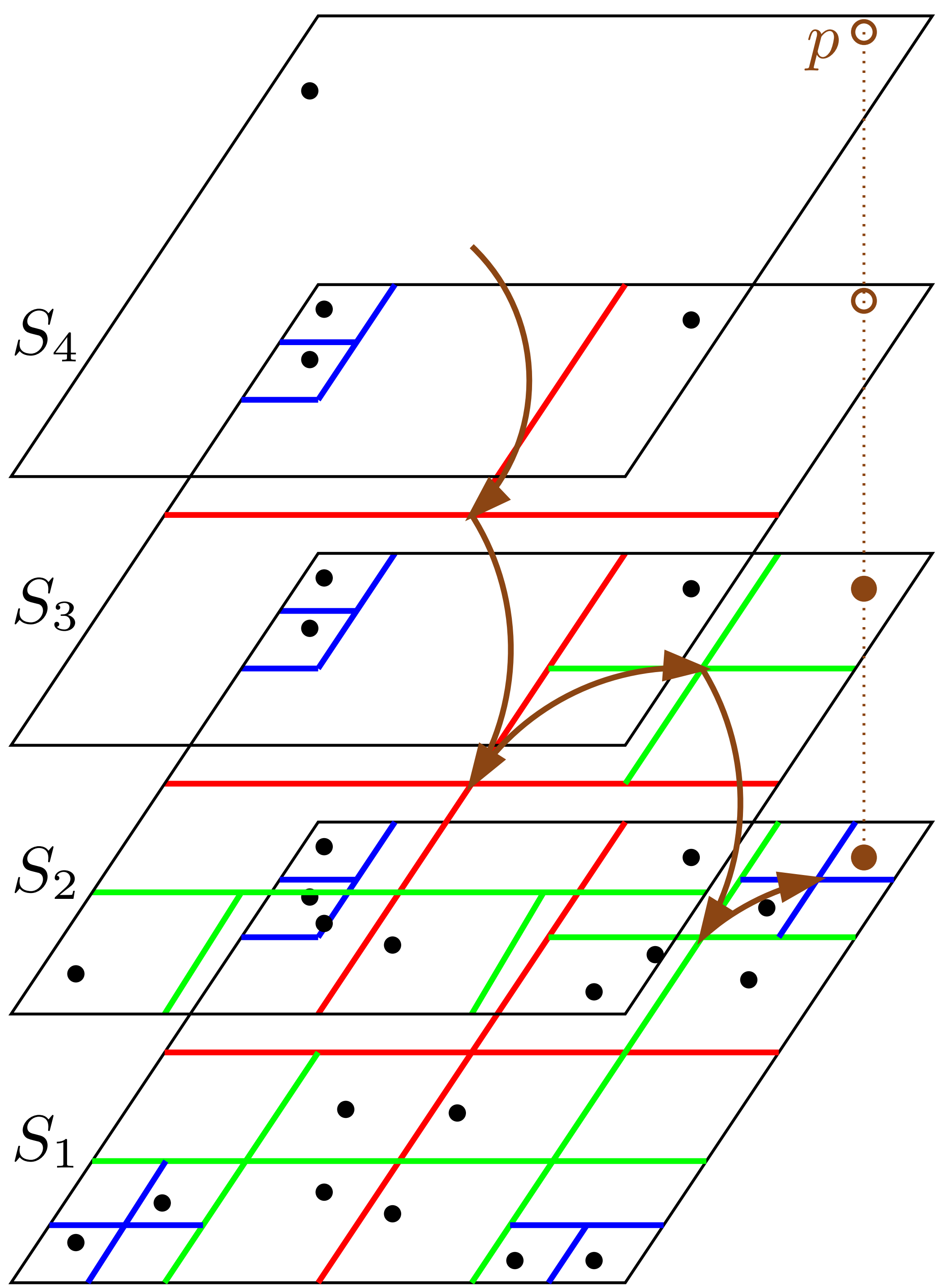
Analysis Outside location path, time spent per layer is $O(1)$. Since q raised w/ proba. $\frac{1}{2}$ per layer, $\mathbf{E}$[max. layer reached] $= \sum_i \frac{i}{2^i} = O(1)$.

$\Rightarrow \mathbf{E}[\text{insertion time}] = O(\log |P|)$.

$\Rightarrow \mathbf{E}[\text{iterative construction time}] = O(|P| \log |P|)$.

Dynamic quadtrees

Note Compressed quadtrees can be updated efficiently under point insertion/deletion, but not finger trees.



Pt deletion Given $p \in P$,
- locate p in $(T_m, \cdots, T_1)$ and store path.
- delete q from leaves of the T_i.
- recursively remove empty nodes from the T_i and transform internal nodes with only 1 pt into leaves (plus remove empty layers).

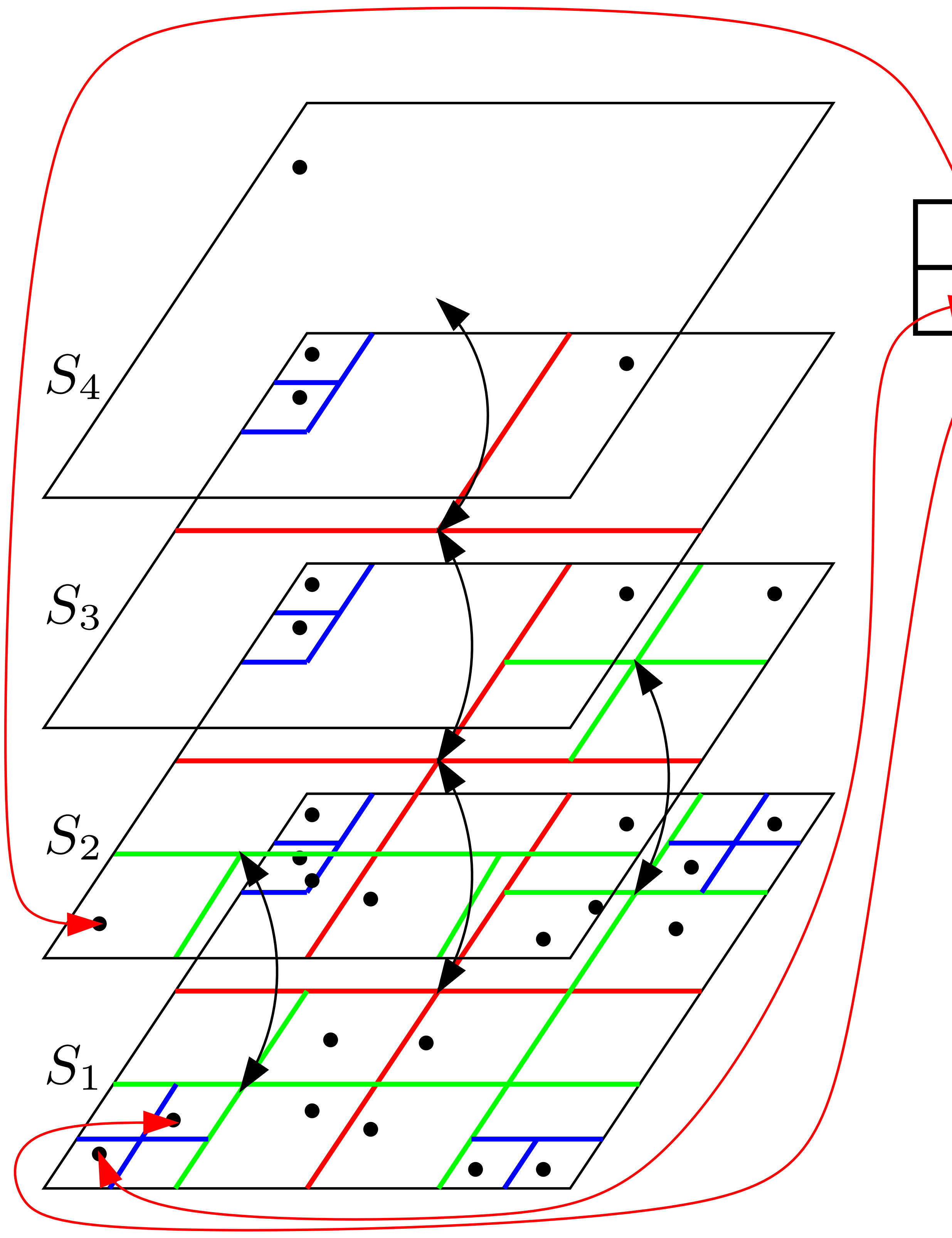
Analysis Only the nodes of the location path are to be considered for deletion or status change. If the instance of v in T_i is deleted, then its parent node in T_i is still non-empty, hence only v and its parent have to be updated $\Rightarrow O(1)$ update time per layer.

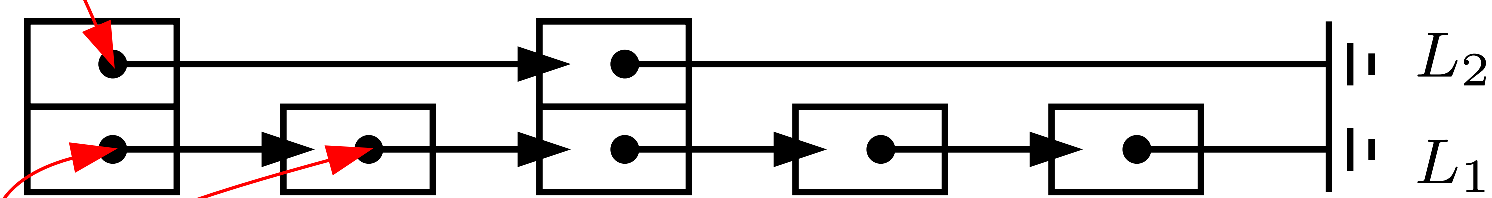
$\Rightarrow \mathbf{E}[\text{deletion time}] = O(\log |P|)$.

Dynamic quadtrees (derandomization)

[D. Eppstein, M. Goodrich, J. Sun, SCG 2005] (deterministic quadtrees)

[J. Munro, T. Papadakis, R. Sedgewick, SODA 1992] (deterministic skip-lists)





Idea:

- Put $S_1 = P$ in list L_1, ordered according to T_1.

- build 1-2-3 deterministic skip-list for L_1: $\forall i > 1$, there are ≥ 1 and ≤ 3 cells in L_{i-1} between any consecutive cells of L_i. $\Rightarrow O(\log |P|)$ layers.

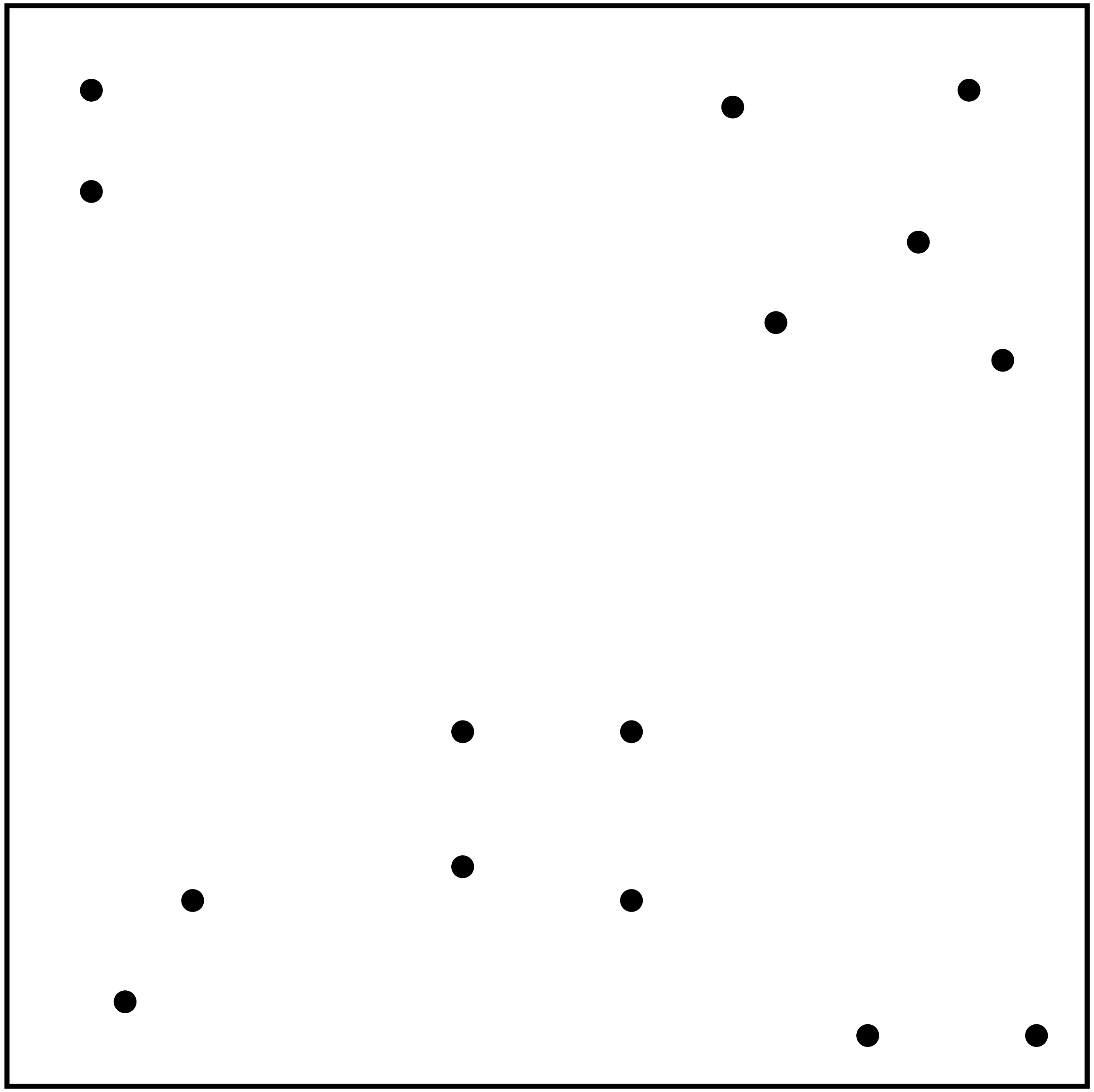
- $\forall i$, $S_i =" P \cap L_i"$. Build compressed quadtree T_i for S_i $\rightarrow$ same order of S_i in L_i and T_i.

- add bi-directed pointers between pts of S_i in L_i and T_i.

$\Rightarrow$ search, insertion, deletion times: $O(\log(n))$.

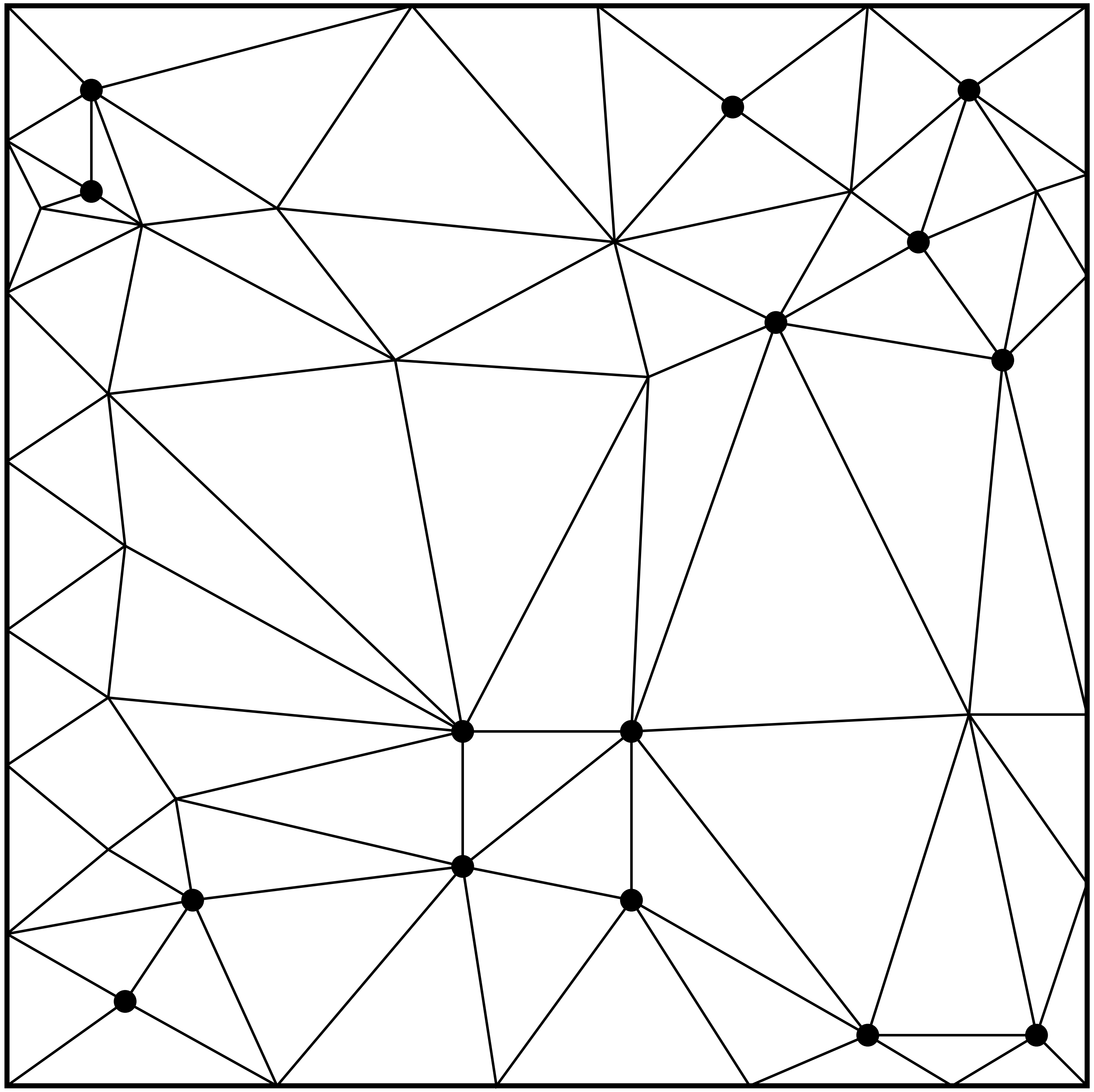
Balanced quadtrees

Adaptive mesh generation Given $P \subset]0,1[^2$ finite, construct the smallest possible triangulation T of $]0,1[^2$, with bounded minimum angle, s.t. every point of P is a vertex of T.



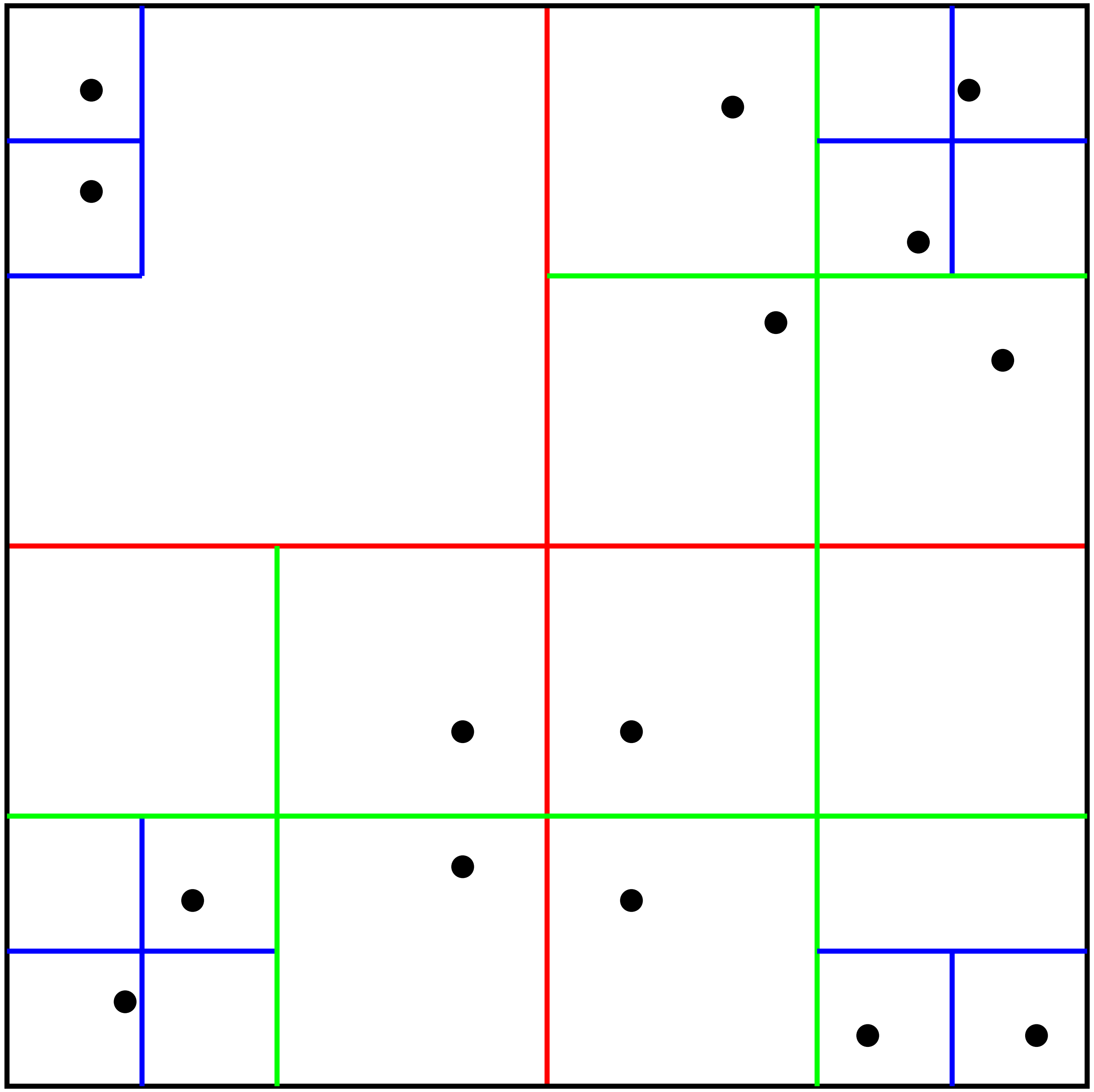
Balanced quadtrees

Adaptive mesh generation Given $P \subset]0,1[^2$ finite, construct the smallest possible triangulation T of $]0,1[^2$, with bounded minimum angle, s.t. every point of P is a vertex of T.



Balanced quadtrees

Adaptive mesh generation Given $P \subset]0,1[^2$ finite, construct the smallest possible triangulation T of $]0,1[^2$, with bounded minimum angle, s.t. every point of P is a vertex of T.

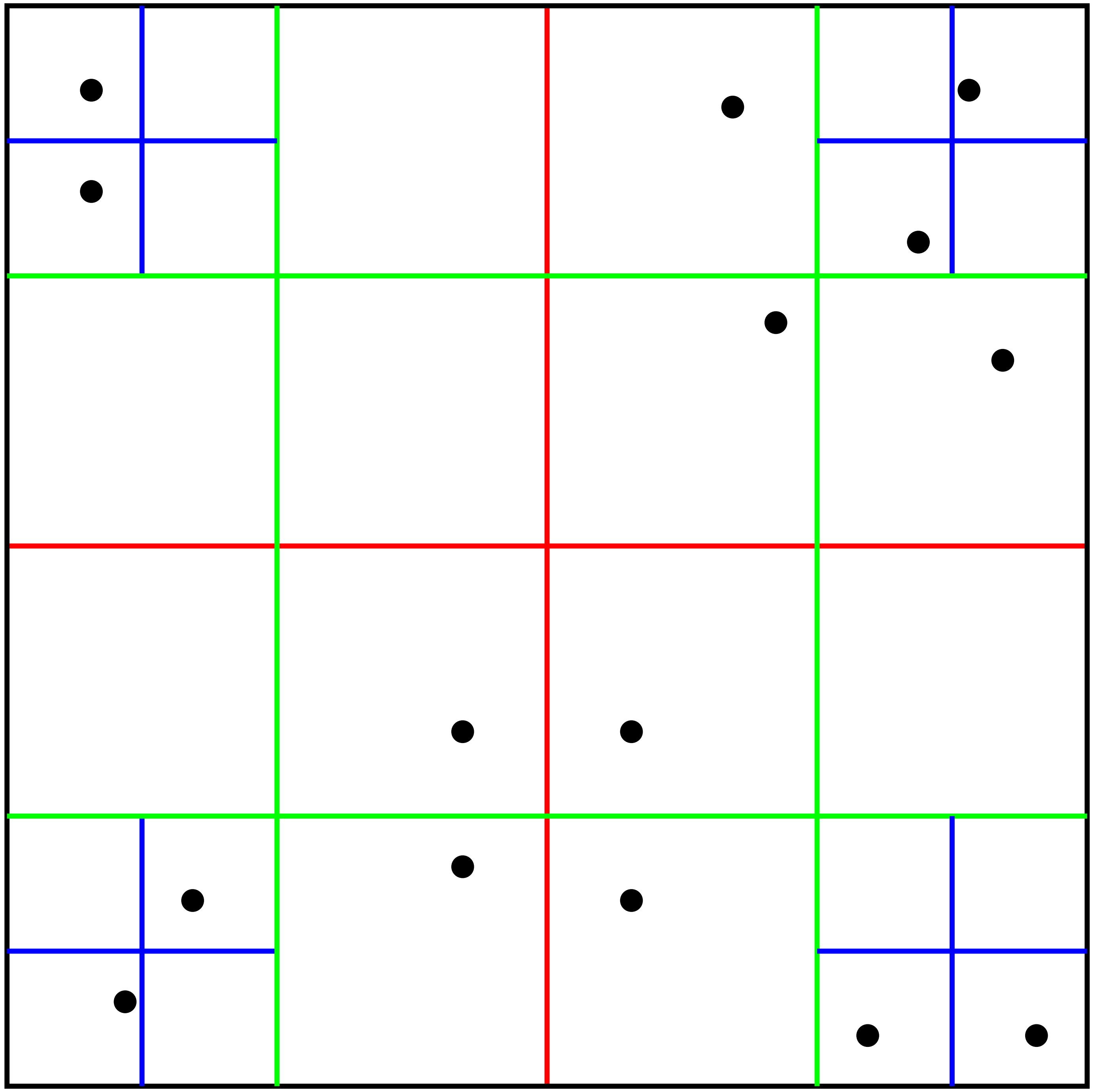


Strategy:

- compute compressed quadtree T_P of $P \rightarrow O(|P| \log |P|)$.

Balanced quadtrees

Adaptive mesh generation Given $P \subset]0,1[^2$ finite, construct the smallest possible triangulation T of $]0,1[^2$, with bounded minimum angle, s.t. every point of P is a vertex of T.



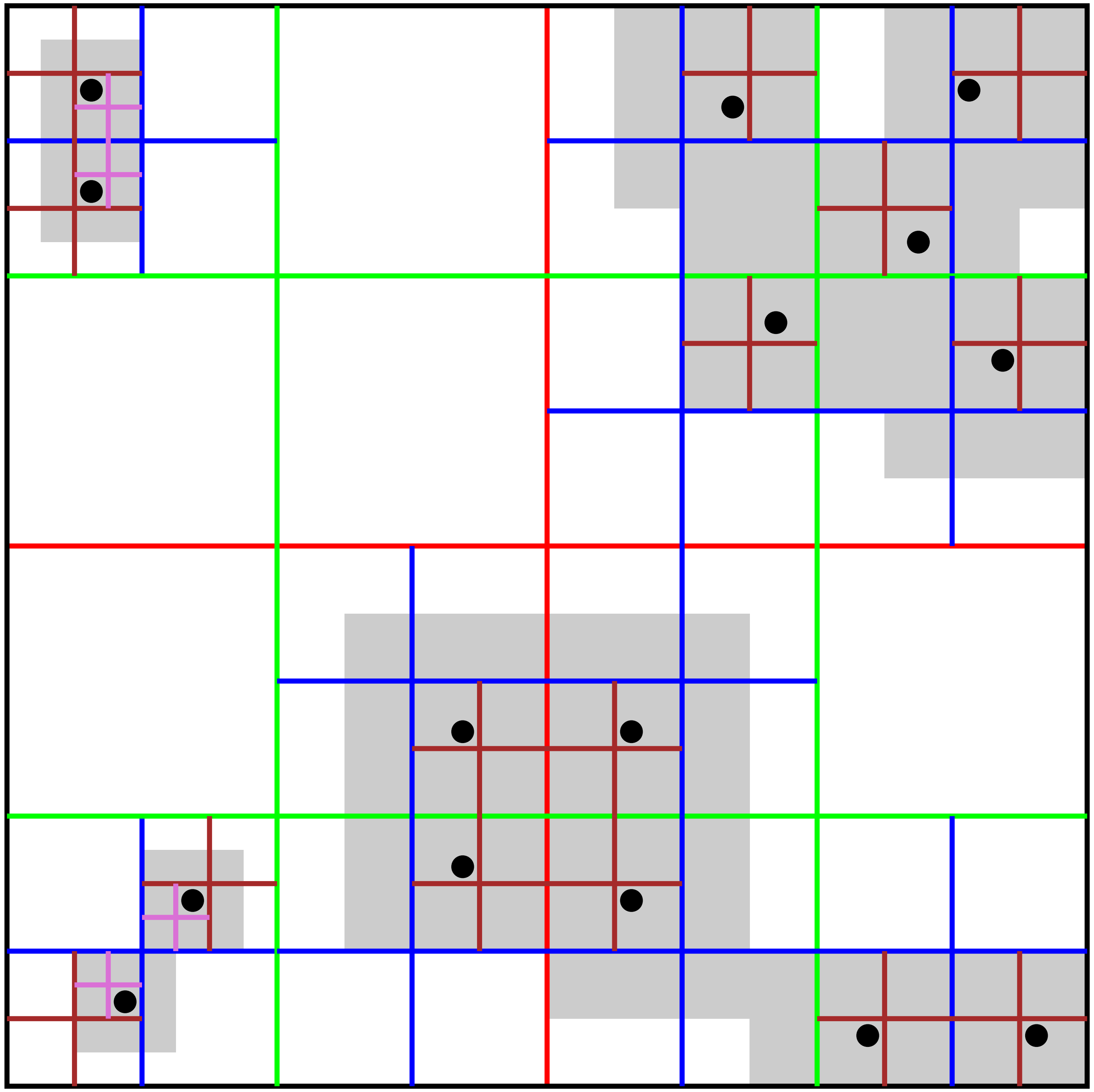
Strategy:

- compute compressed quadtree T_P of $P \rightarrow O(|P| \log |P|)$.

- uncompress T_P

Balanced quadtrees

Adaptive mesh generation Given $P \subset]0,1[^2$ finite, construct the smallest possible triangulation T of $]0,1[^2$, with bounded minimum angle, s.t. every point of P is a vertex of T.



Strategy:

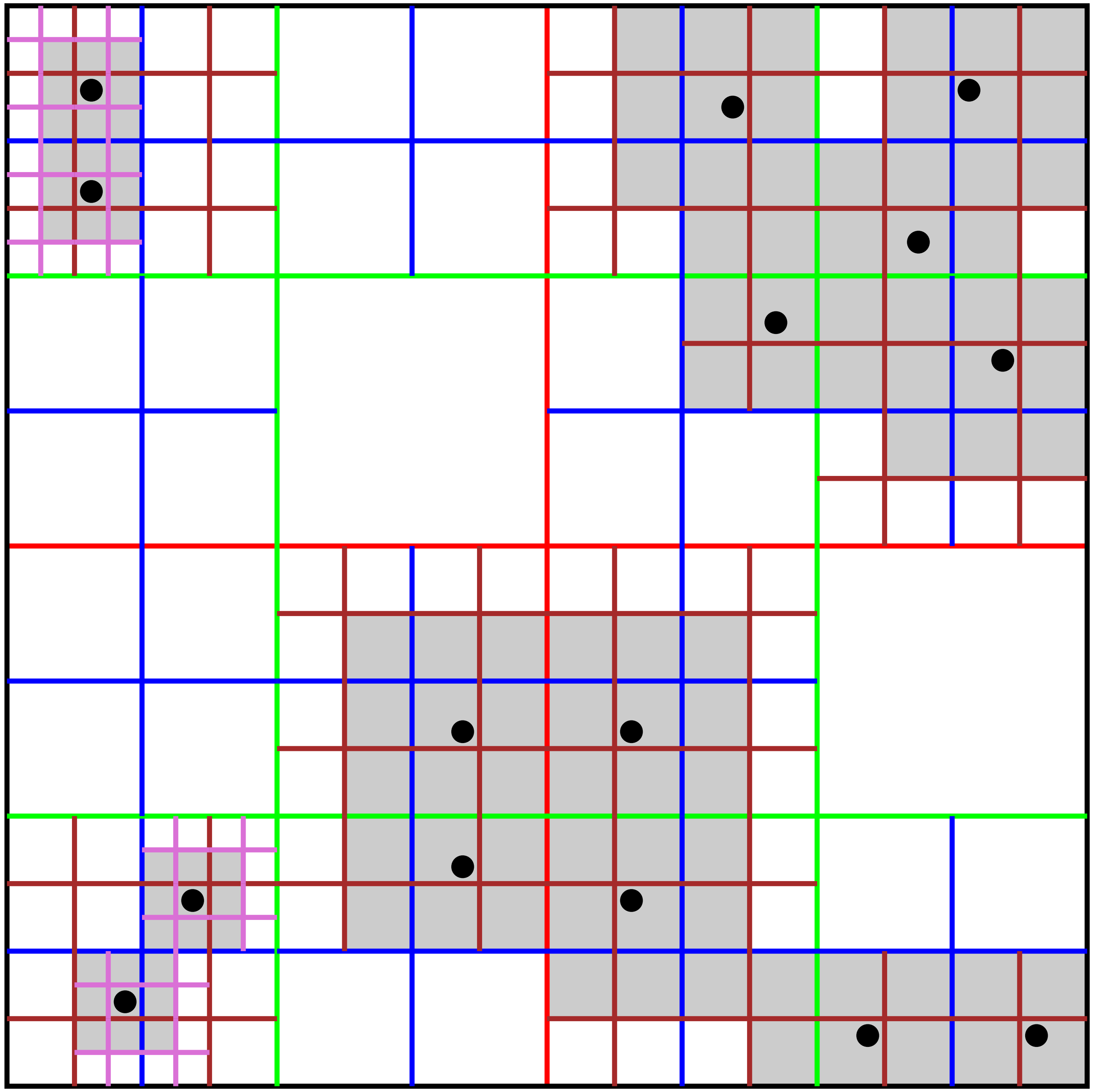
- compute compressed quadtree T_P of $P \rightarrow O(|P| \log |P|)$.

- uncompress T_P

- refine T_P so that, $\forall p \in P$, the 1-ring neighb. of p contains no pt of $P \setminus \{p\}$ ($\forall$ cell, use pointers to adjacent cells in 8-connectivity).

Balanced quadtrees

Adaptive mesh generation Given $P \subset]0,1[^2$ finite, construct the smallest possible triangulation T of $]0,1[^2$, with bounded minimum angle, s.t. every point of P is a vertex of T.



Strategy:

- compute compressed quadtree T_P of $P \rightarrow O(|P| \log |P|)$.

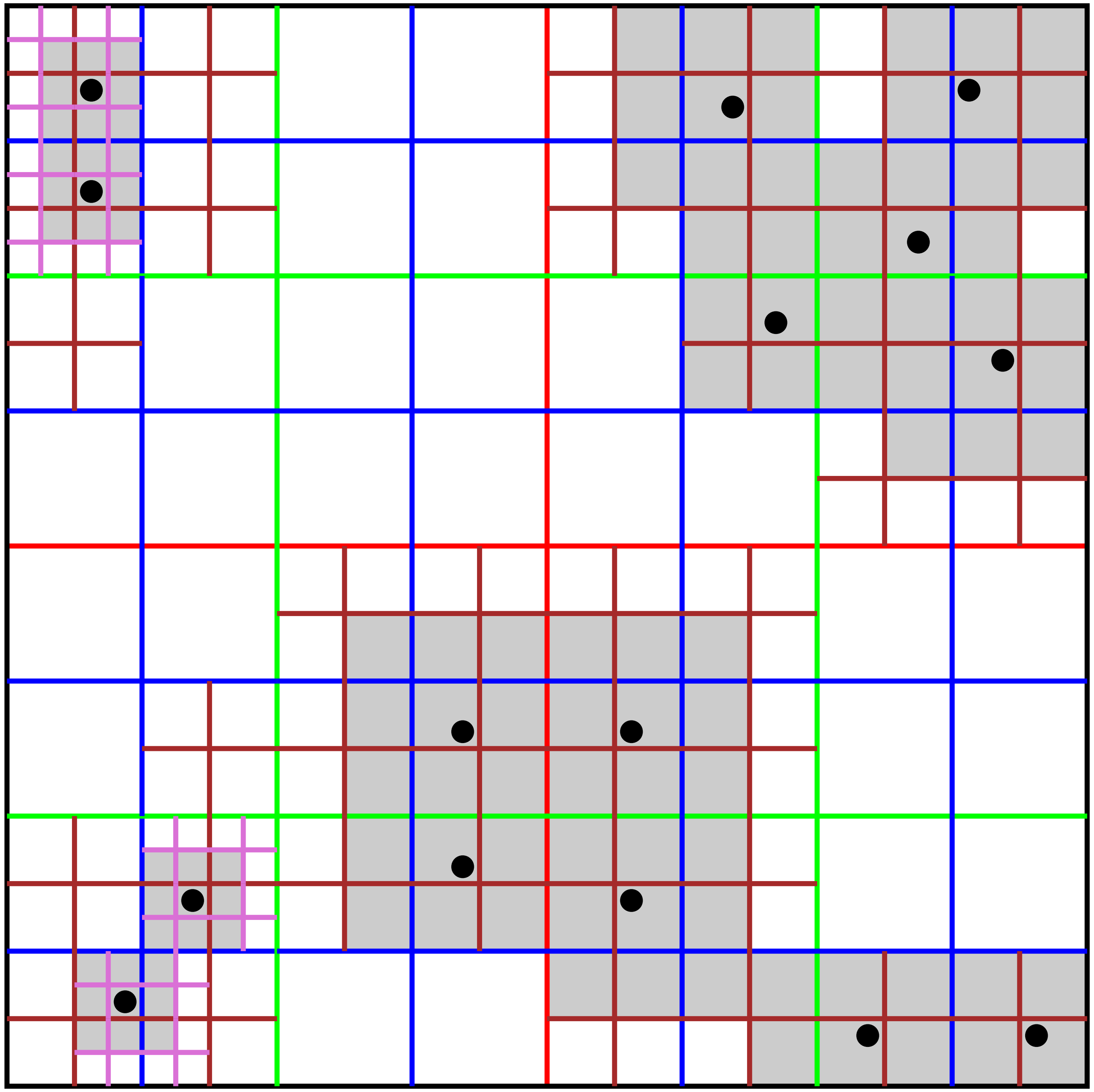
- uncompress T_P

- refine T_P so that, $\forall p \in P$, the 1-ring neighb. of p contains no pt of $P \setminus \{p\}$ ($\forall$ cell, use pointers to adjacent cells in 8-connectivity).

- insert 1-ring neighbs. in T_P

Balanced quadtrees

Adaptive mesh generation Given $P \subset]0,1[^2$ finite, construct the smallest possible triangulation T of $]0,1[^2$, with bounded minimum angle, s.t. every point of P is a vertex of T.



Strategy:

- compute compressed quadtree T_P of $P \to O(|P| \log |P|)$.

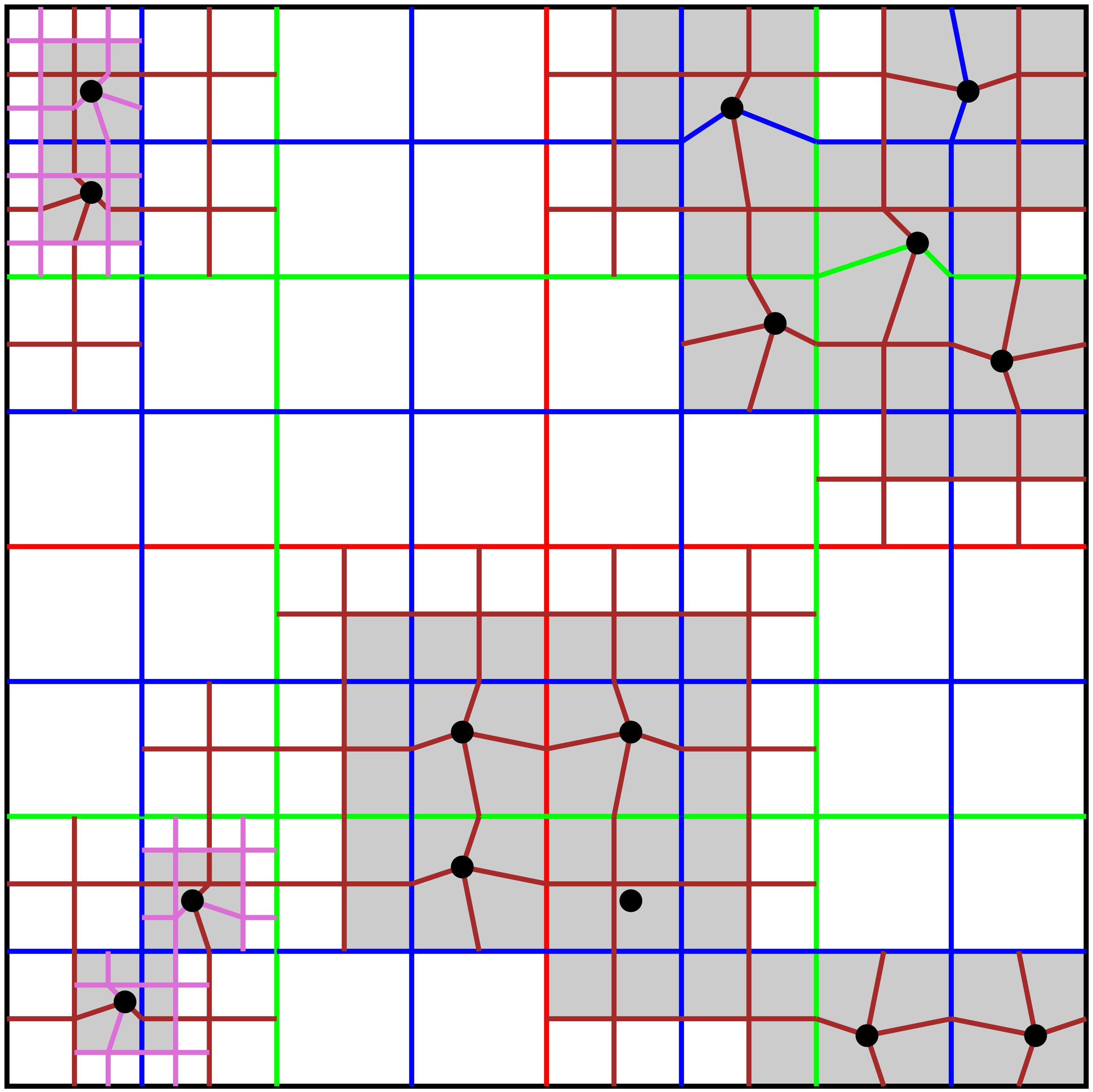
- uncompress T_P

- refine T_P so that, $\forall p \in P$, the 1-ring neighb. of p contains no pt of $P \setminus \{p\}$ ($\forall$ cell, use pointers to adjacent cells in 8-connectivity).

- insert 1-ring neighbs. in T_P and refine T_P so that it is balanced.

Balanced quadtrees

Adaptive mesh generation Given $P \subset]0,1[^2$ finite, construct the smallest possible triangulation T of $]0,1[^2$, with bounded minimum angle, s.t. every point of P is a vertex of T.



Strategy:

- compute compressed quadtree T_P of $P \rightarrow O(|P| \log |P|)$.

- uncompress T_P

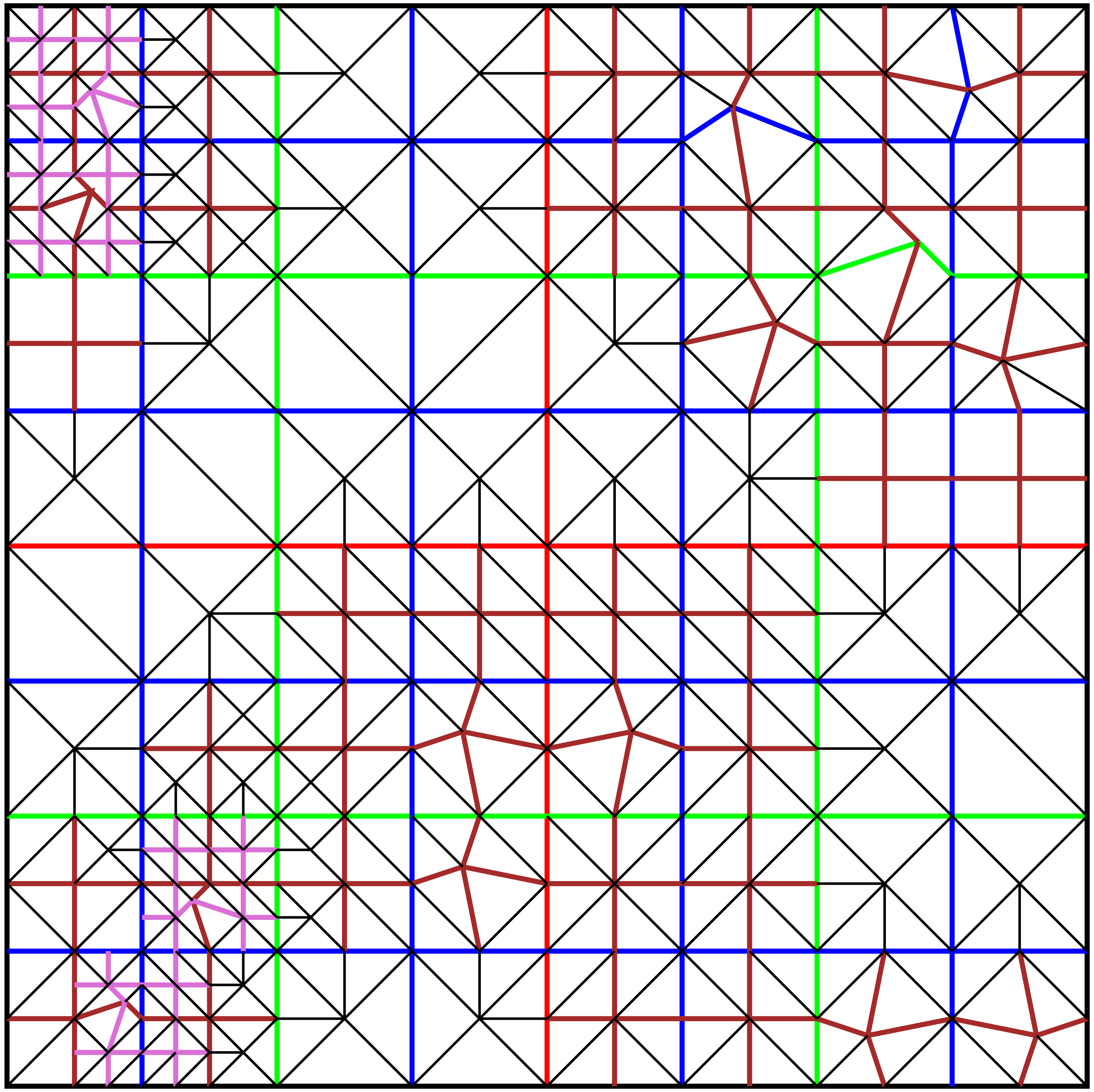
- refine T_P so that, $\forall p \in P$, the 1-ring neighb. of p contains no pt of $P \setminus \{p\}$ ($\forall$ cell, use pointers to adjacent cells in 8-connectivity).

- insert 1-ring neighbs. in T_P and refine T_P so that it is balanced.

- *snap* nearest vertices onto pts of P.

Balanced quadtrees

Adaptive mesh generation Given $P \subset]0,1[^2$ finite, construct the smallest possible triangulation T of $]0,1[^2$, with bounded minimum angle, s.t. every point of P is a vertex of T.



Strategy:

- compute compressed quadtree T_P of $P \rightarrow O(|P| \log |P|)$.

- uncompress T_P

- refine T_P so that, $\forall p \in P$, the 1-ring neighb. of p contains no pt of $P \setminus \{p\}$ ($\forall$ cell, use pointers to adjacent cells in 8-connectivity).

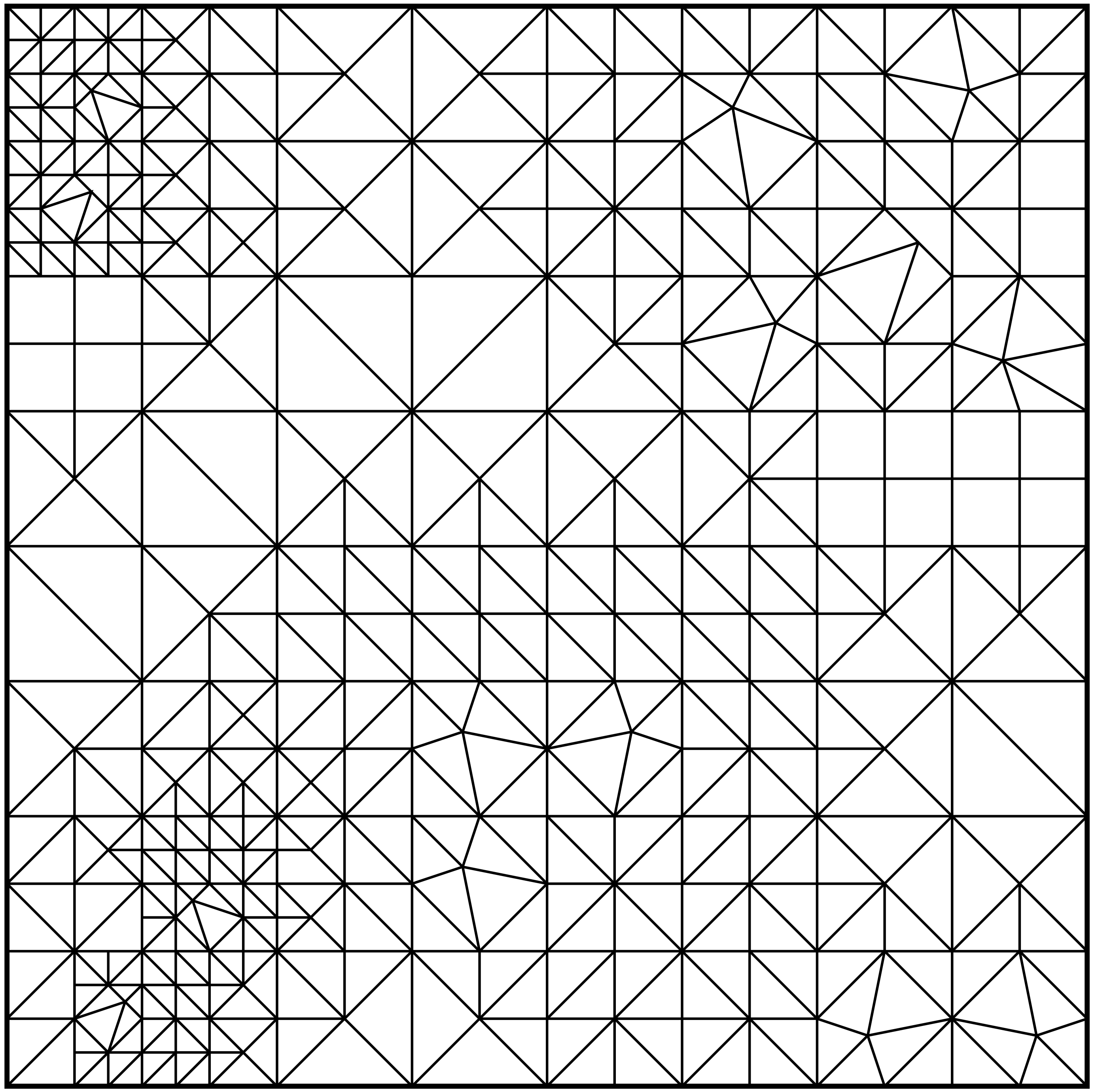
- insert 1-ring neighbs. in T_P and refine T_P so that it is balanced.

- *snap* nearest vertices onto pts of P.

- triangulate cells (3 cases: unsplit bound., split bound., moved vertex).

Balanced quadtrees

Adaptive mesh generation Given $P \subset]0,1[^2$ finite, construct the smallest possible triangulation T of $]0,1[^2$, with bounded minimum angle, s.t. every point of P is a vertex of T.



Strategy:

- compute compressed quadtree T_P of $P \rightarrow O(|P| \log |P|)$.

- uncompress T_P

- refine T_P so that, $\forall p \in P$, the 1-ring neighb. of p contains no pt of $P \setminus \{p\}$ ($\forall$ cell, use pointers to adjacent cells in 8-connectivity).

- insert 1-ring neighbs. in T_P and refine T_P so that it is balanced.

- *snap* nearest vertices onto pts of P.

- triangulate cells (3 cases: unsplit bound., split bound., moved vertex).

$\Rightarrow$ time: $O(|P| \log |P| + |\text{output}|)$

Take-home message

- Quadtrees vs. uniform grids: space-time trade-off.
- Effective location data structure in low dimensions, both in static (compressed quadtrees) and dynamic (skip-quadtrees) settings.
- Main advantages: easy to implement, good average behaviour in practice (time and space).
- Downside: fundamentally anisotropic (*cf.* point location among triangles, mesh generation, *etc.*).
- **Very useful for approximation** (*cf.* snap-rounding).