INF631 – Data Analysis: Geometry and Topology in Arbitrary Dimensions

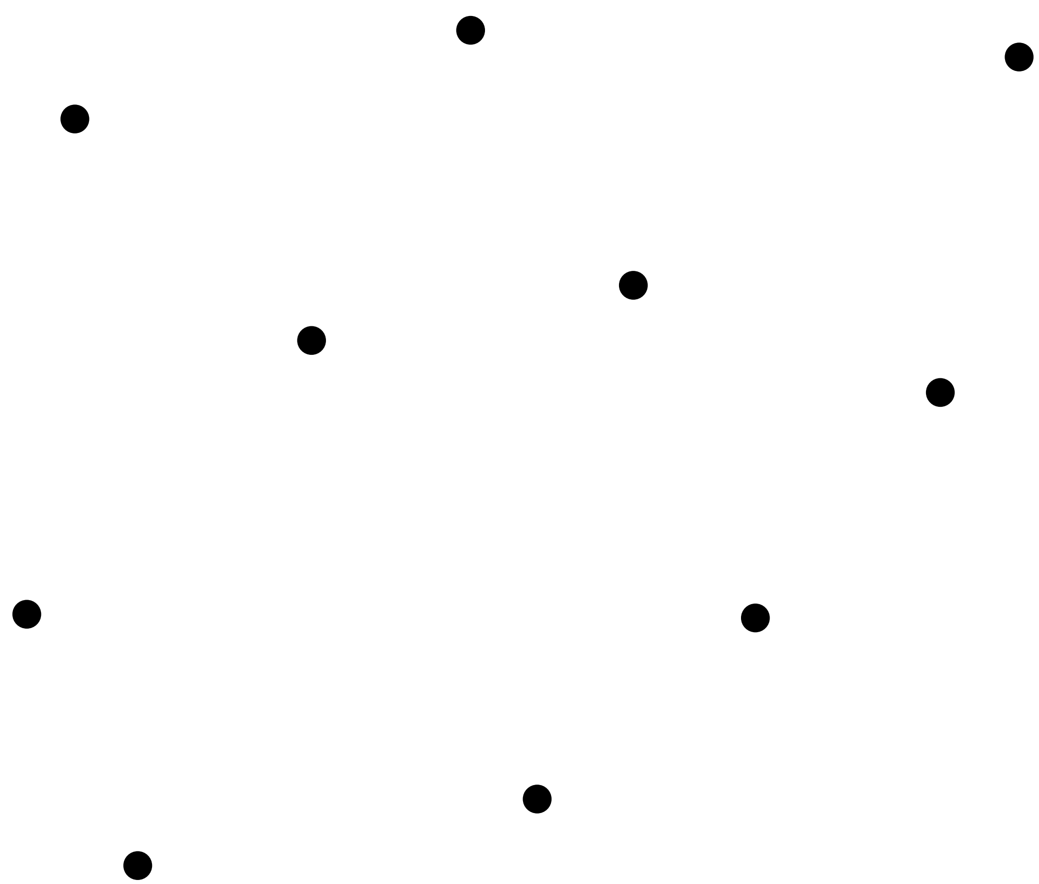
Nearest Neighbor Search

Steve Oudot

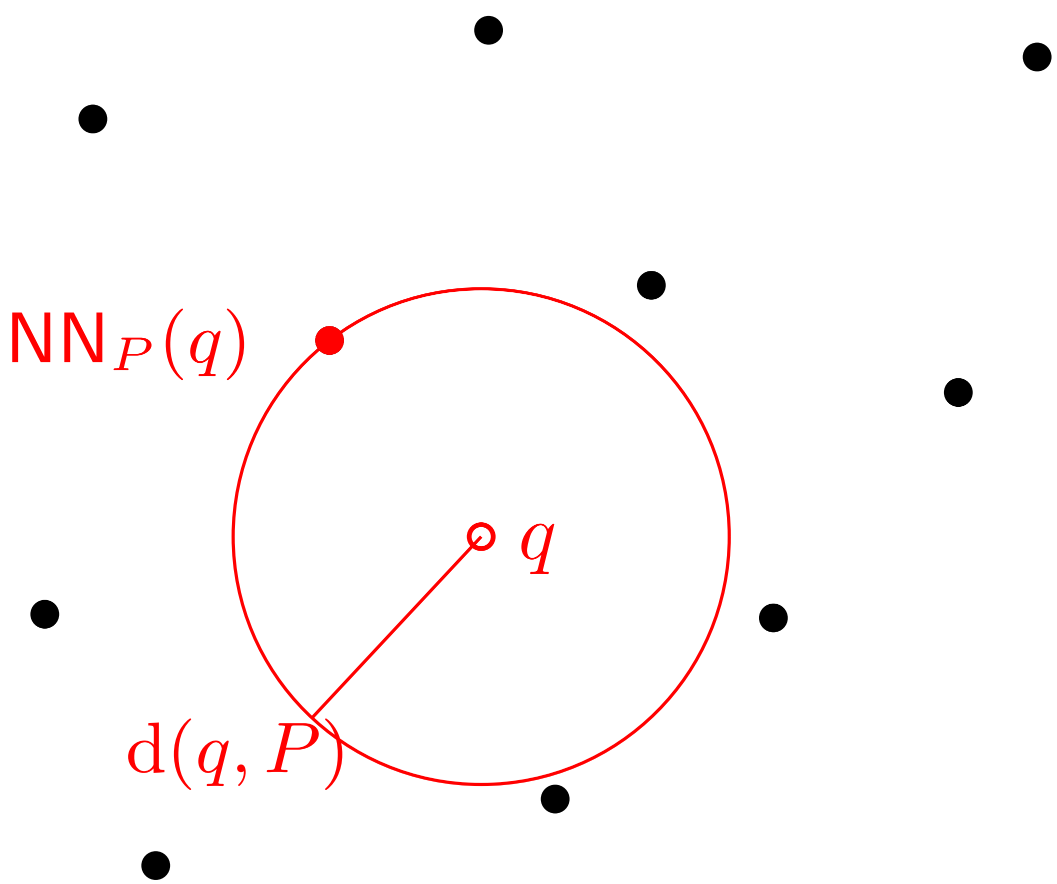
ÉCOLE POLYTECHNIQUE

Nearest-Neighbor problem

pre-processing input: P



Nearest-Neighbor problem



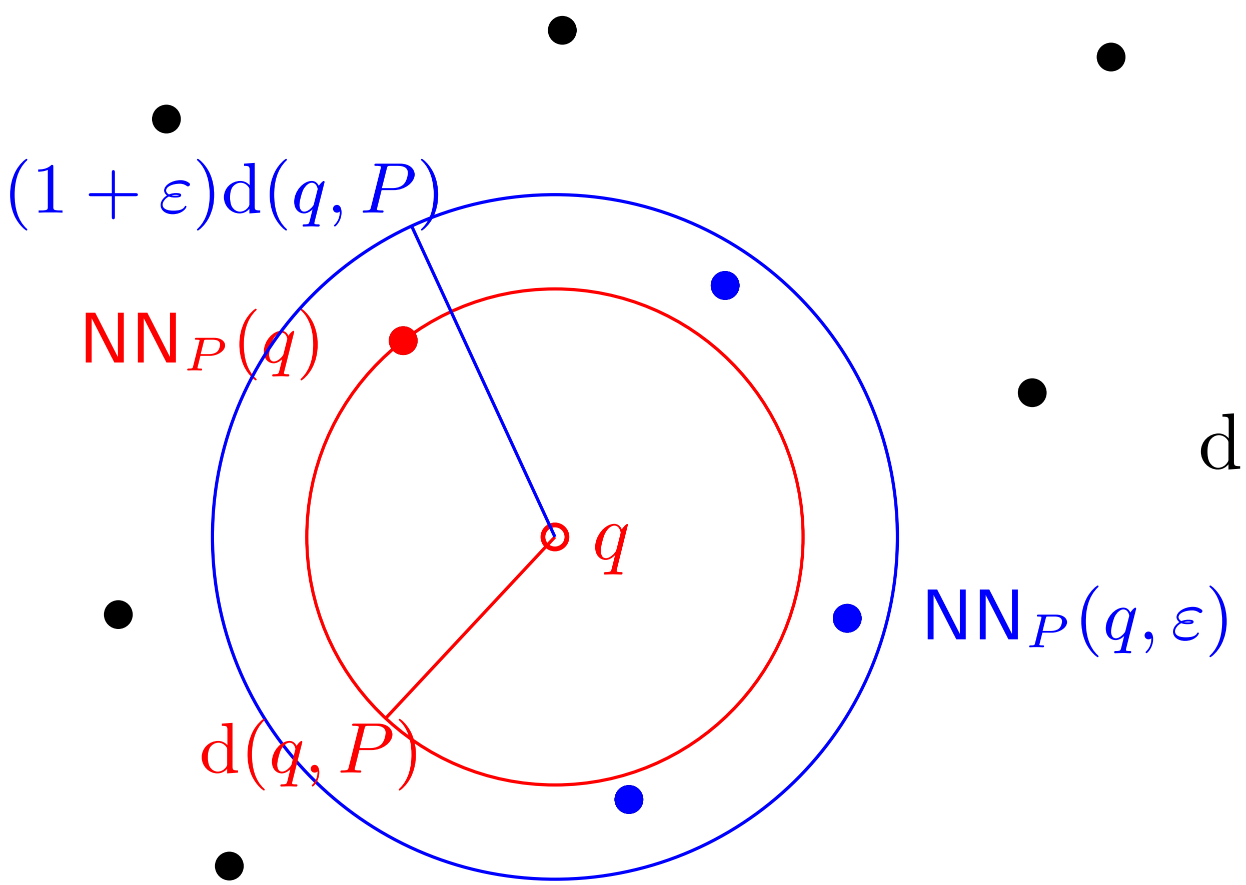
pre-processing input: P

query input: q

goal: find $p \in \mathrm{NN}_P(q)$

$\mathrm{d}(q, p) = \min_{p' \in P} \mathrm{d}(q, p')$

ε-Nearest-Neighbor problem



pre-processing input: P, ε

query input: q

goal: find $p \in \mathrm{NN}_P(q,\varepsilon)$

$$\mathrm{d}(q,p) \le (1+\varepsilon)\ \min_{p' \in P} \mathrm{d}(q,p')$$

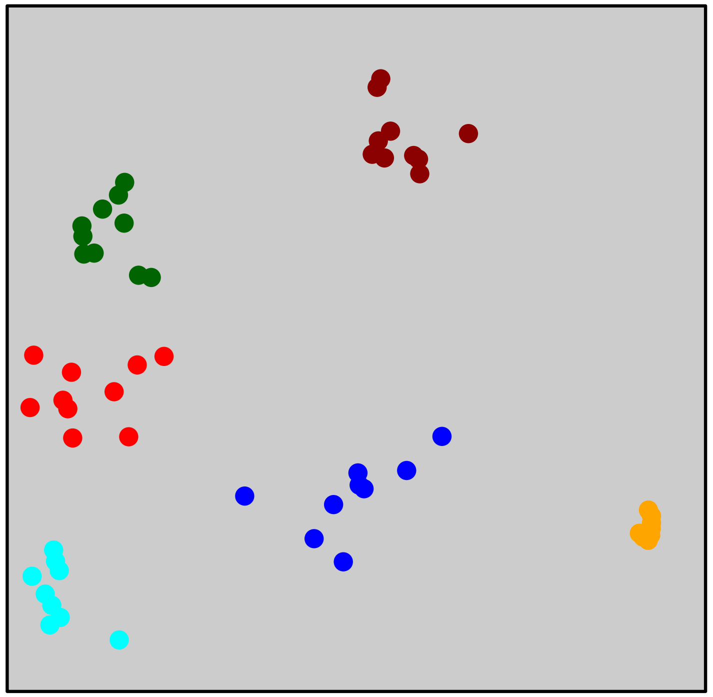
Nearest-Neighbor problem

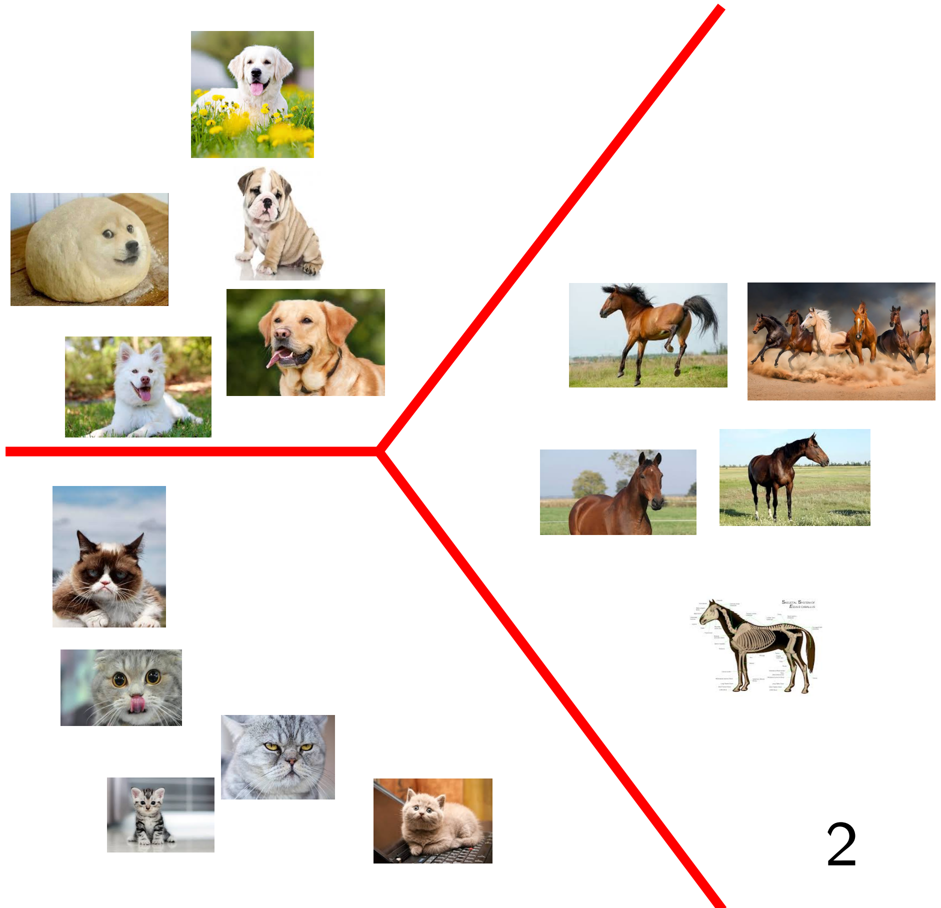
Variants:

- k-nearest neighbors: find the k points closest to q in P
- r-nearest neighbor: find a point $p \in P$ such that $\mathrm{d}(q, p) \leq r$
- metrics:
  - ℓ_2, ℓ_p, ℓ_∞
  - strings: Hamming distance
  - images: optimal transport distances
  - point clouds: (Gromov-)Hausdorff distances
  - proteins: RMSD distances
  - $\cdots$

Applications

- clustering, e.g. k-means, mean-shift
- information retrieval in databases
- information theory, e.g. vector quantization
- supervised learning, e.g. NN-classifiers
- ...





Strategy and Challenges

Strategy:

- preprocess the n point of P in $\mathbb{R}^d$ into some data structure DS for fast nearest-neighbor queries answers

Ideal wish list:

- DS should have linear size in n and polynomial size in d
- a query should take sublinear time in n and polynomial time in d

  e.g. binary search trees in $d = 1$: linear size, $O(\log n)$ time

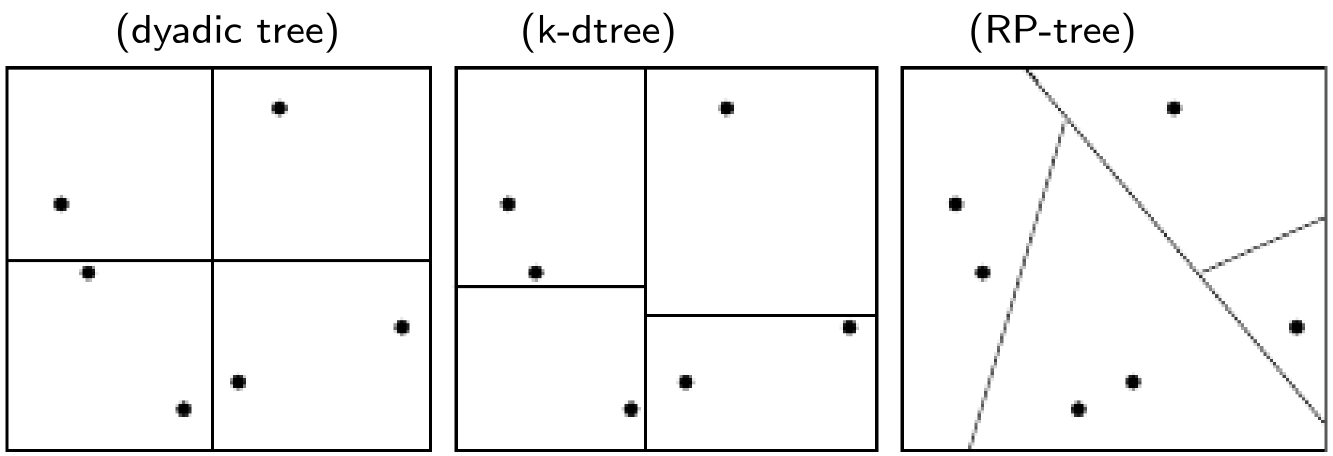
Core difficulties:

- *Curse of dimensionality*: hard to outperform linear scan in high d
- Interpretation: meaningfulness of distances in high d (concentration)

Approaches

$O(dn)$ space and time

- Linear scan
- Voronoi diagrams
- Tree-like data structures



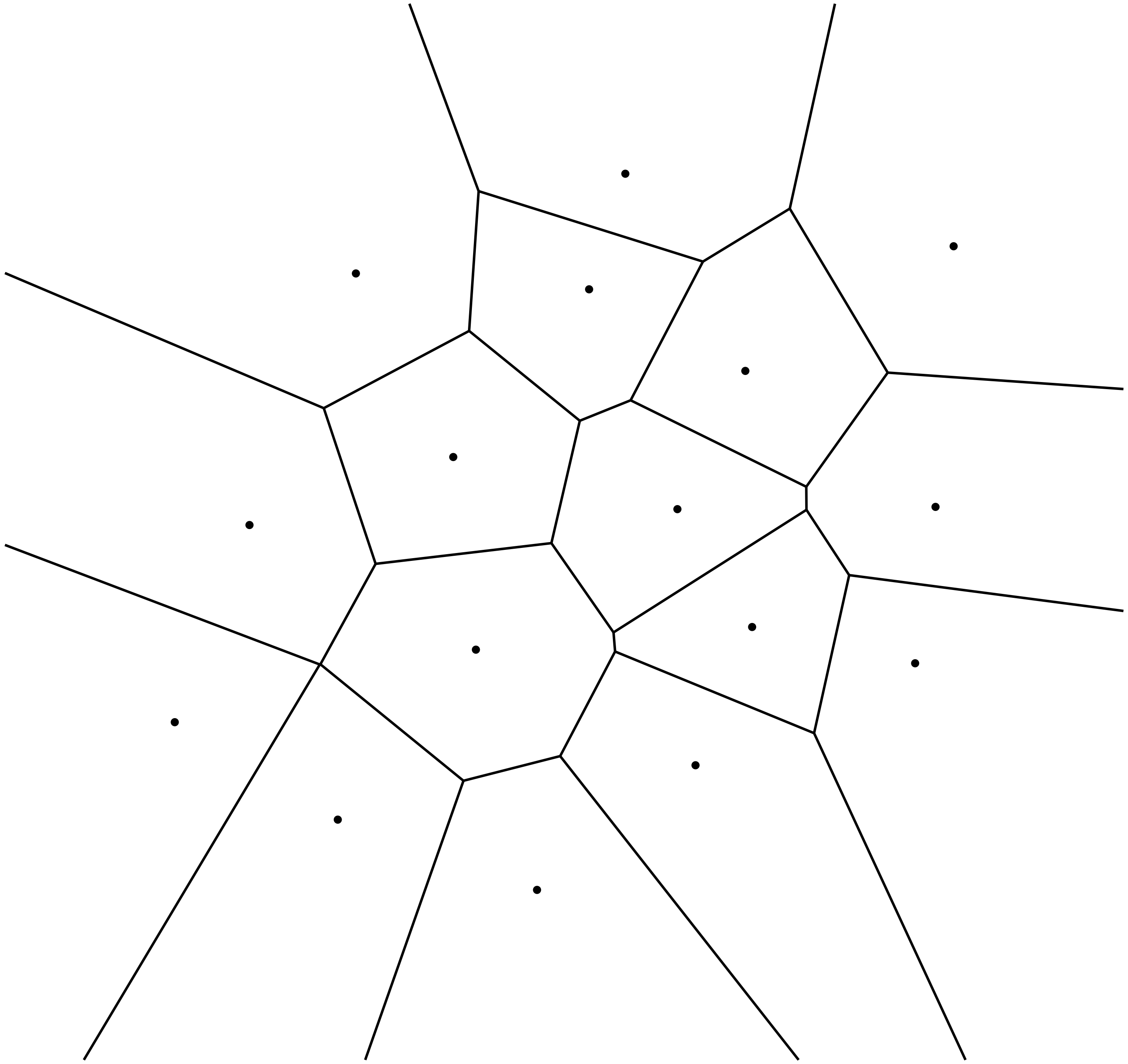
  - quadtrees (split at midpoint in all coordinates)
  - tries / dyadic trees (split at mean, cycle around coordinates)
  - kd-trees (split at median, cycle around coordinates)
  - Random Projection trees (split at median along random coordinates)
  - PCA trees (split at median along 1st eigenvector of covariance matrix)
  - $\cdots$

Binary Space Partitions

- Locality Sensitive Hashing

Voronoi diagrams

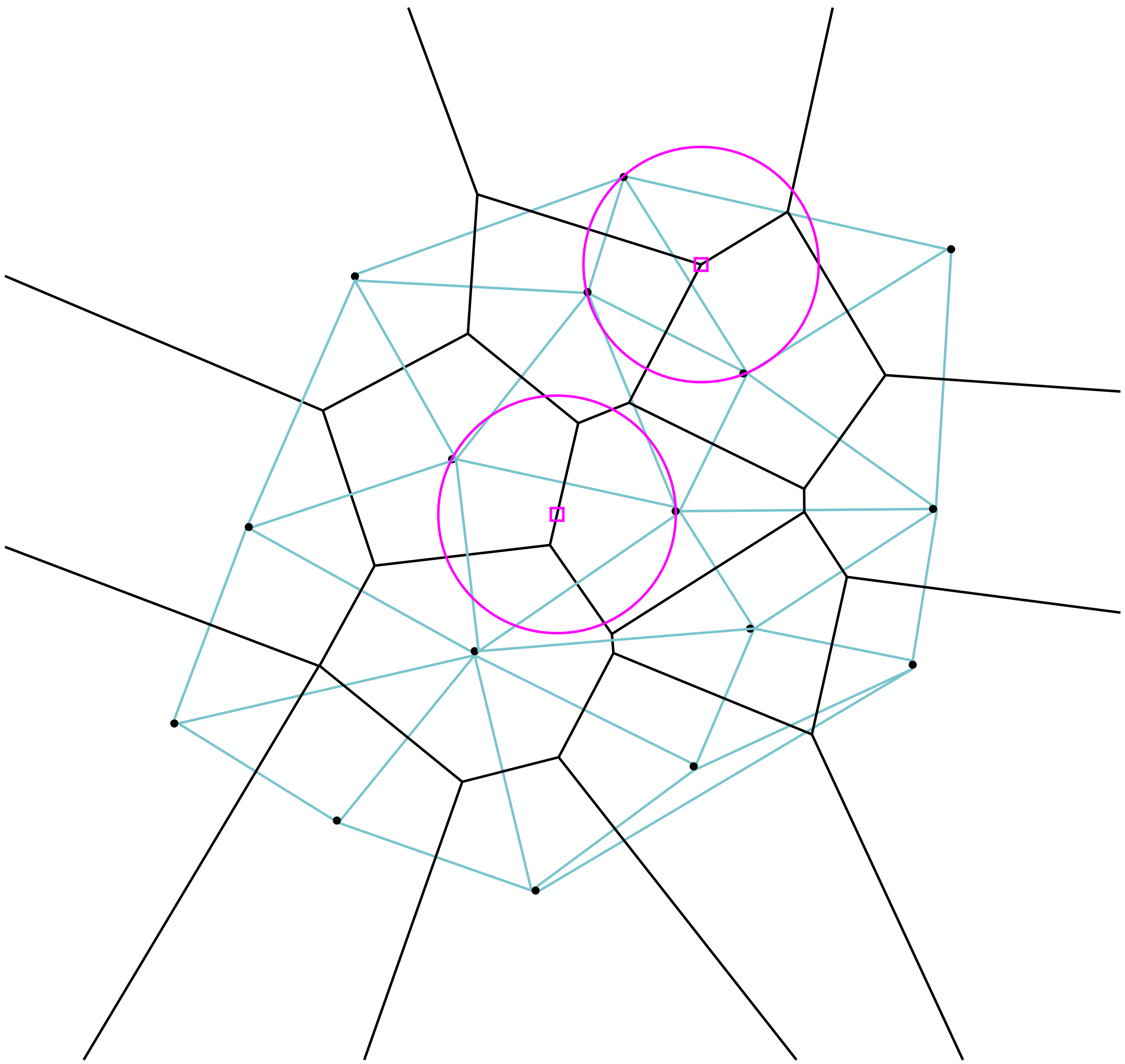
Definition



$$V(p) := \{q \in \mathbb{R}^d \mid p \in \mathrm{NN}_P(q)\}$$

affine diagram

Definition



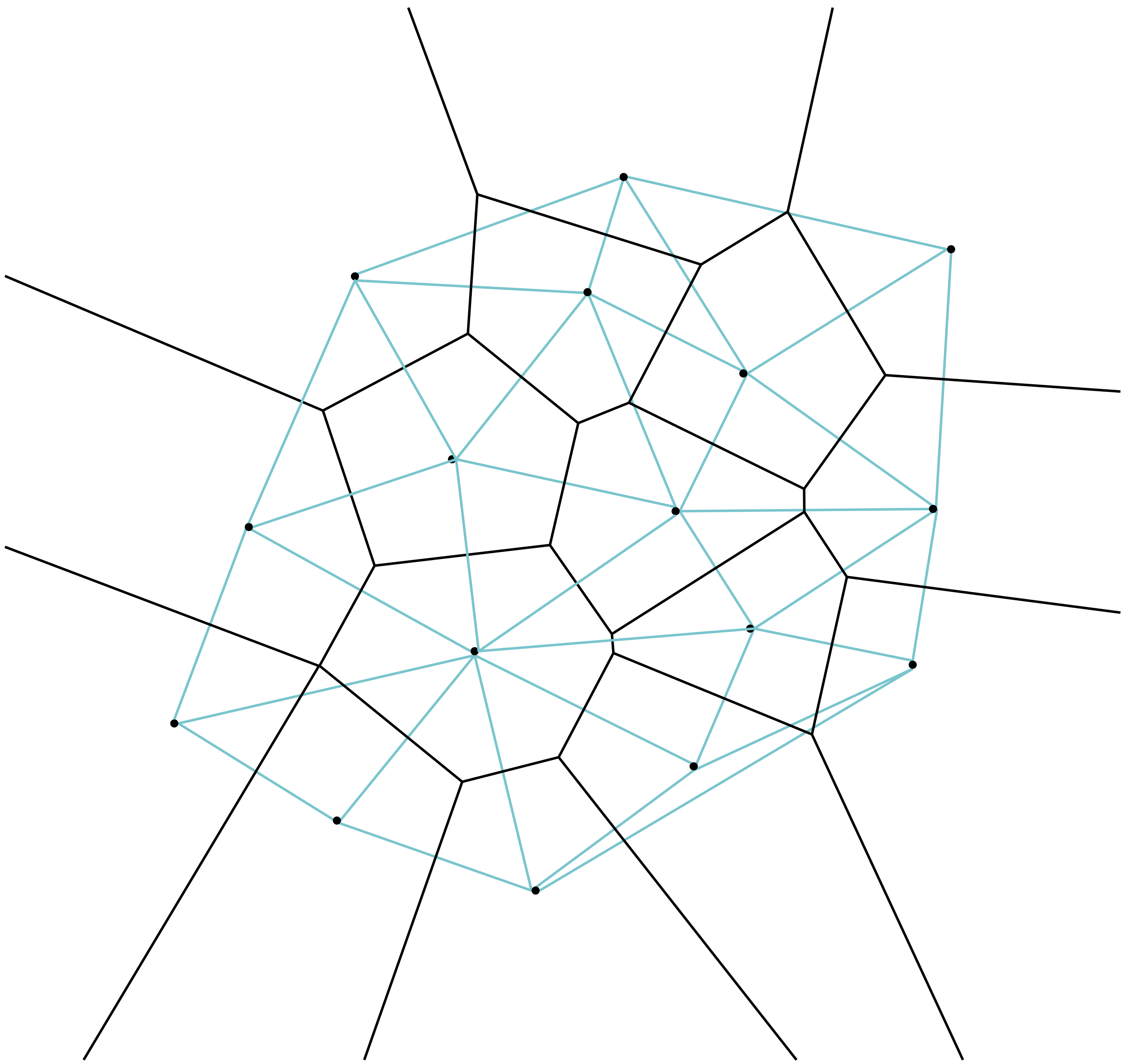
$V(p) := \{q \in \mathbb{R}^d \mid p \in \mathrm{NN}_P(q)\}$

affine diagram

computed/stored via dual

(Delaunay triangulation)

Definition



$V(p) := \{q \in \mathbb{R}^d \mid p \in \mathrm{NN}_P(q)\}$

affine diagram

computed/stored via dual

(Delaunay triangulation)

size:

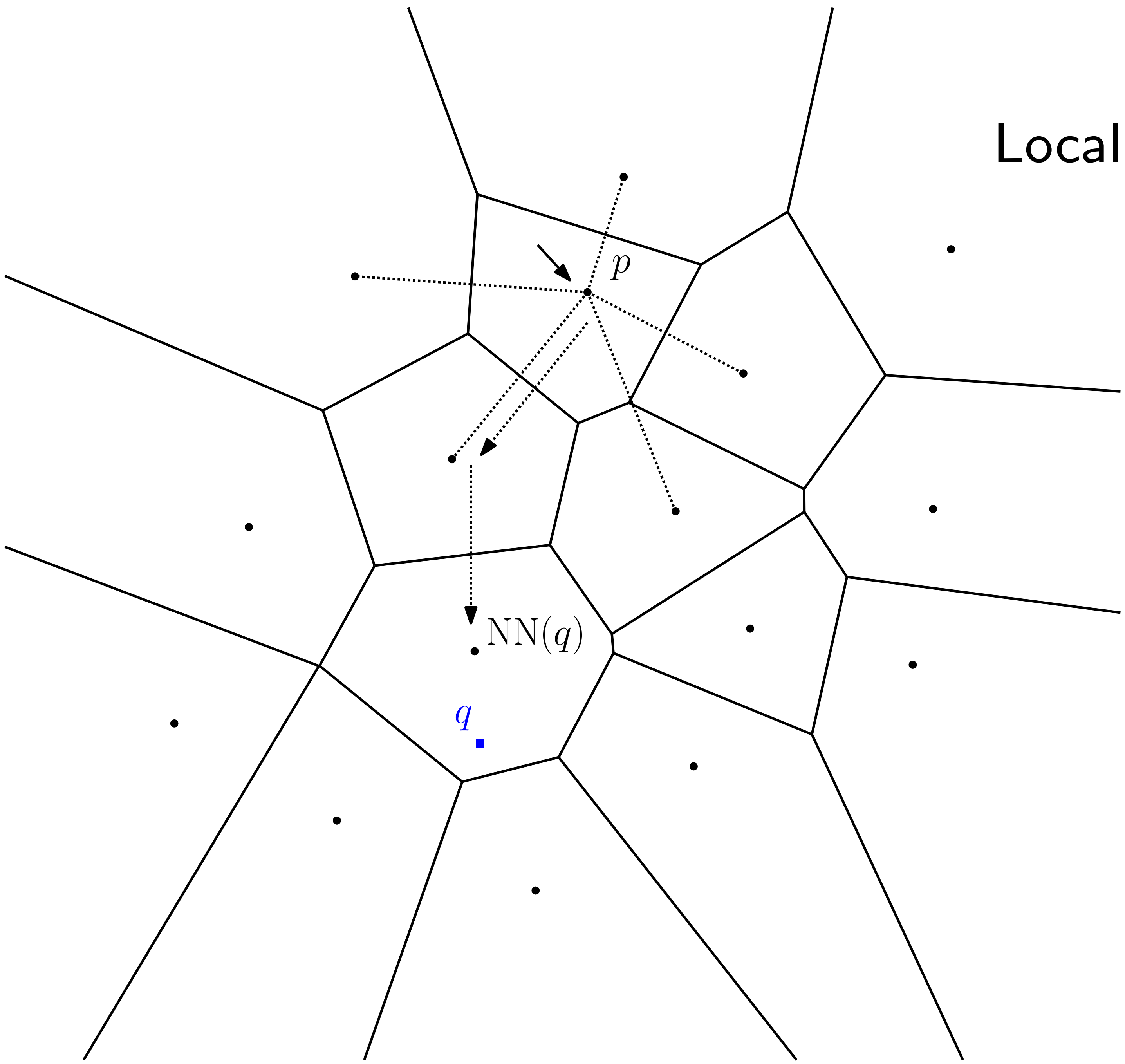
- worst case: $\Theta\left(n^{\lceil d/2 \rceil}\right)$

  Upper Bound Thm [McMullen'70]

- average case (unif. distrib.):

  $2^{O(d \log d)} n$

Usage for NN-search



Localizing by **walk**:

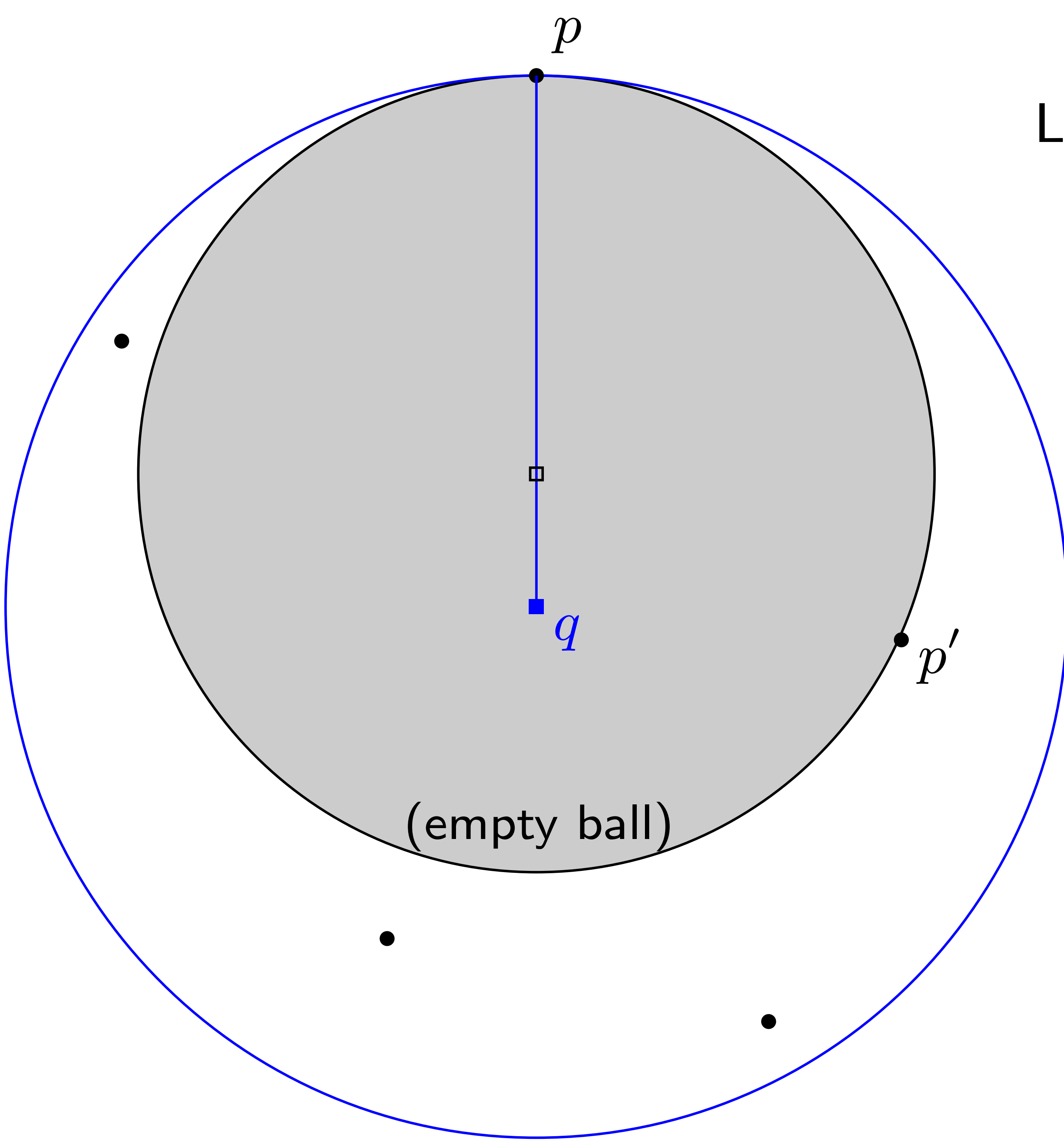
start from $p \in P$ random

while $\exists\, p'$ neighb. of p in Del.

s.t. $\mathrm{d}(q, p') < \mathrm{d}(q, p)$:

update $p := p'$

Usage for NN-search



Localizing by **walk**:

start from $p \in P$ random

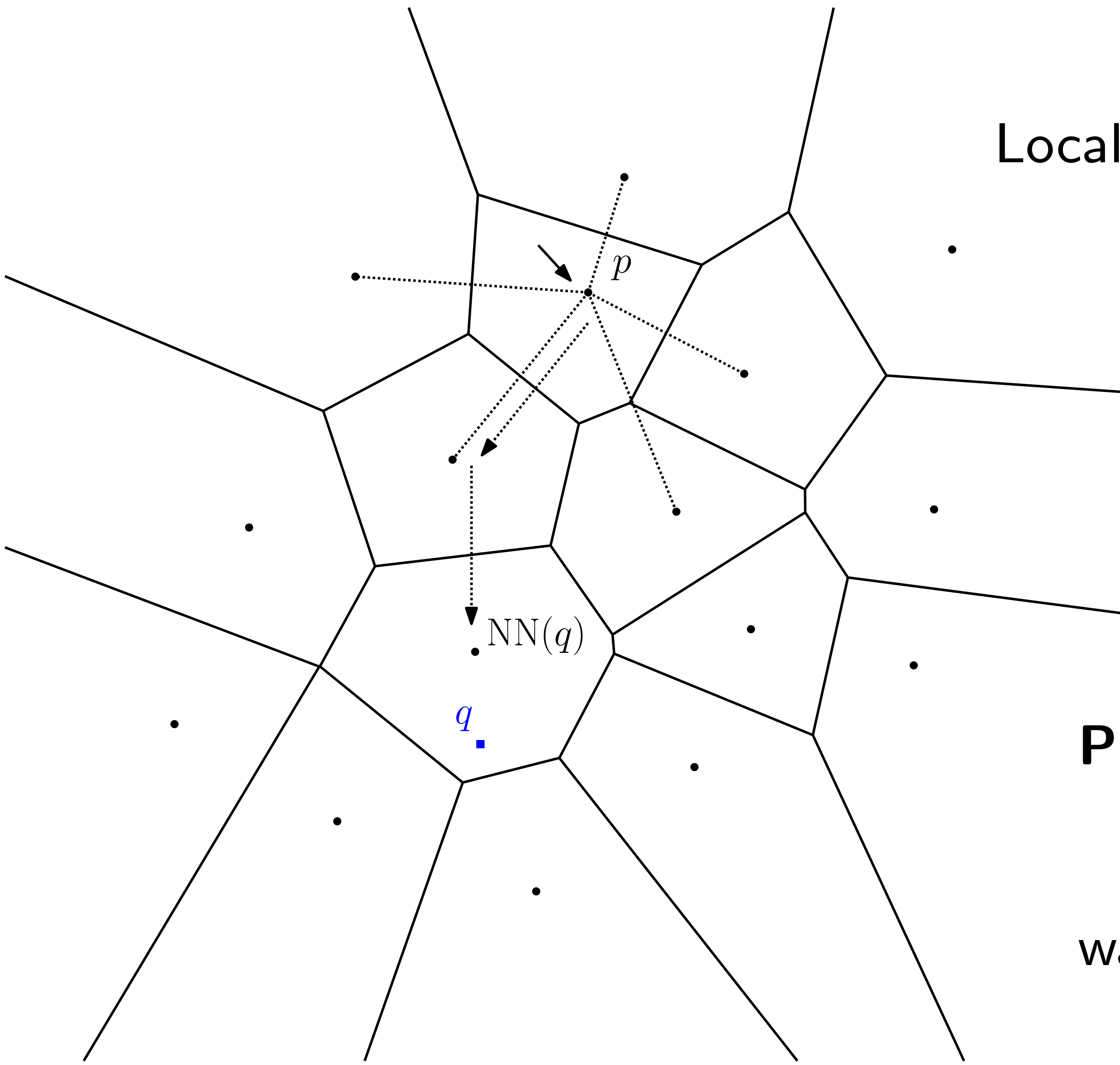
while $\exists\ p'$ neighb. of p in Del.

s.t. $\mathrm{d}(q, p') < \mathrm{d}(q, p)$:

update $p := p'$

Prop: Del. neighborhood is complete

Usage for NN-search



Localizing by **walk**:

start from $p \in P$ random

while $\exists\ p'$ neighb. of p in Del.

s.t. $\mathrm{d}(q, p') < \mathrm{d}(q, p)$:

update $p := p'$

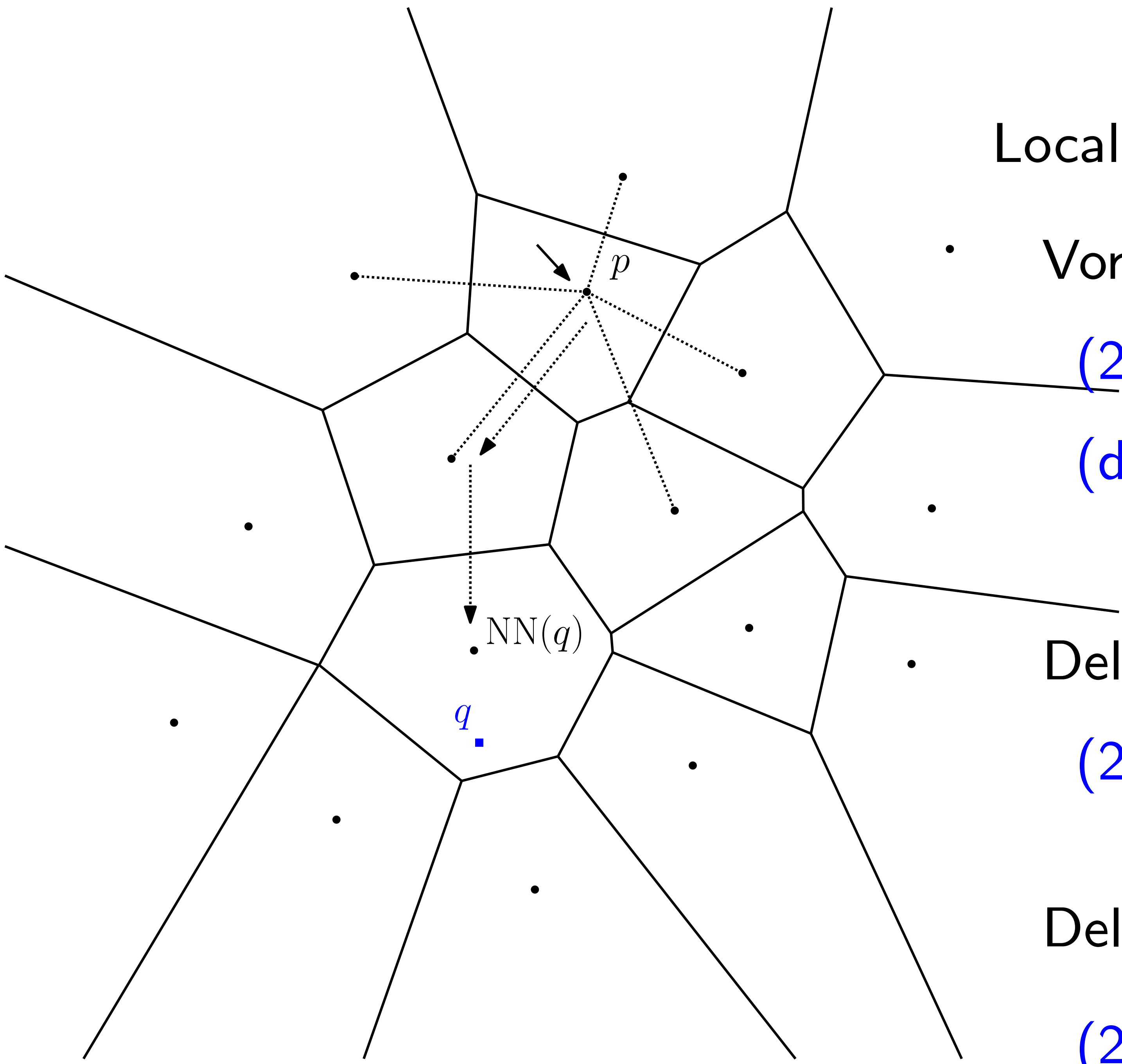
Prop: Del. neighborhood is complete

walk time:

worst case: $O(|\mathsf{Del}(P)|)$

avg. case (2d): $O(\sqrt{n})$

Usage for NN-search



Localizing by **hierarchy**:

Voronoi subdivision [Kirk.'83, Meiser'93]:

(2D) $O(n)$ space, $O(\log n)$ time

(dD) $\Theta(n^d)$ space, $O(d^5 \log n)$ time

Delaunay tree [Mulmuley'91]:

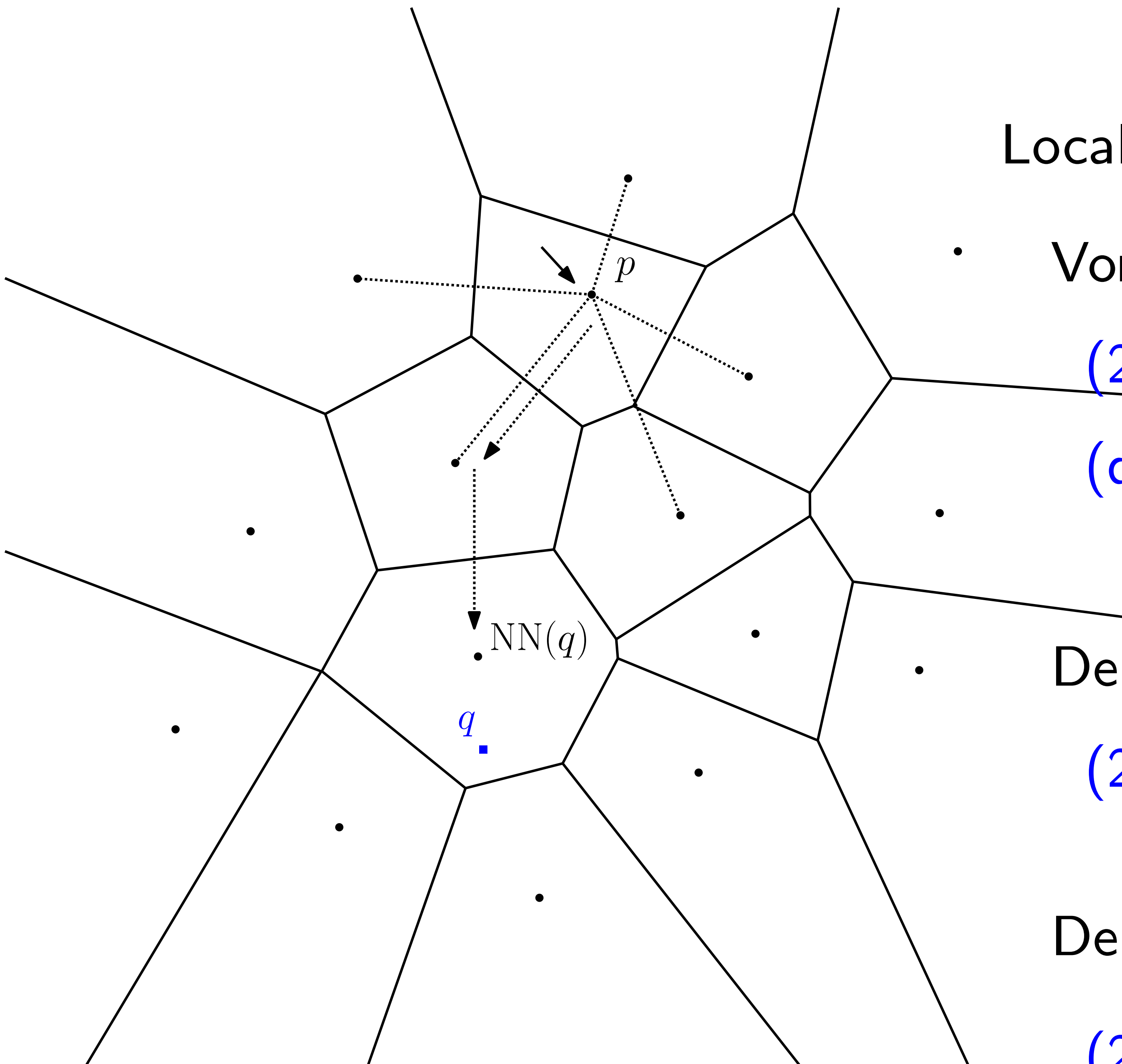
(2D) $O(n \log n)$ space, $O(\log n)$ time

Delaunay tree + walk [Devillers'02]:

(2D) $O(n \log n)$ space, $O(\log n)$ time

(dD) $O(n^{\lceil \frac{d}{2} \rceil})$ space, $O(n^{\lceil \frac{d-2}{2} \rceil})$ time

Usage for NN-search



For small dimensions (2 or 3) only!

Localizing by **hierarchy**:

- Voronoi subdivision [Kirk.'83, Meiser'93]:

  (2D) $O(n)$ space, $O(\log n)$ time

  (dD) $\Theta(n^d)$ space, $O(d^5 \log n)$ time

Delaunay tree [Mulmuley'91]:

(2D) $O(n \log n)$ space, $O(\log n)$ time

Delaunay tree + walk [Devillers'02]:

(2D) $O(n \log n)$ space, $O(\log n)$ time

(dD) $O(n^{\lceil \frac{d}{2} \rceil})$ space, $O(n^{\lceil \frac{d-2}{2} \rceil})$ time

k-d trees

Definition

- a binary tree

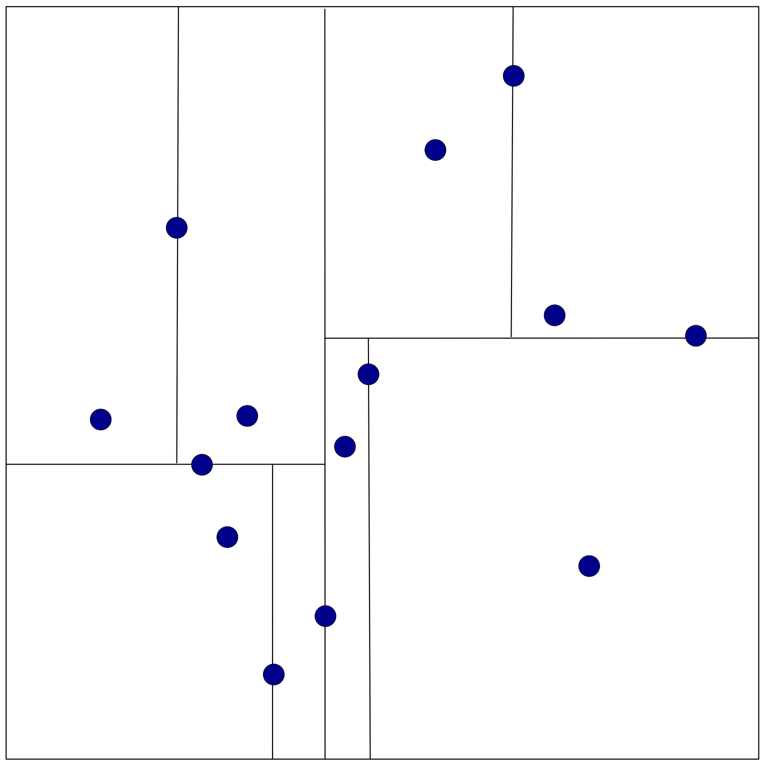
- each internal node implements a spatial partition induced by a hyperplane H, splitting the point cloud into two equal subsets

  - ▶ right subtree: all points lying on one side of H

  - ▶ left subtree: remaining points

- subdivision stops whenever fewer than n_0 remain

  $\rightsquigarrow$ size: $O(dn)$



$(n_0 = 1)$

Definition

- a binary tree

- each internal node implements a spatial partition induced by a hyperplane H, splitting the point cloud into two equal subsets

  - ▶ right subtree: all points lying on one side of H

  - ▶ left subtree: remaining points

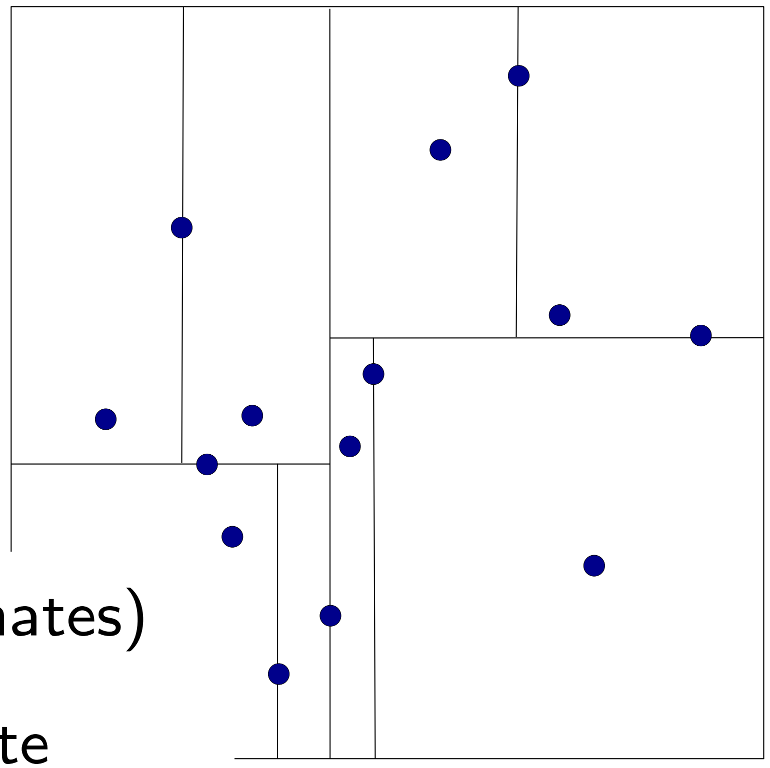
- subdivision stops whenever fewer than n_0 remain

$\rightsquigarrow$ size: $O(dn)$

kd-tree specifics:

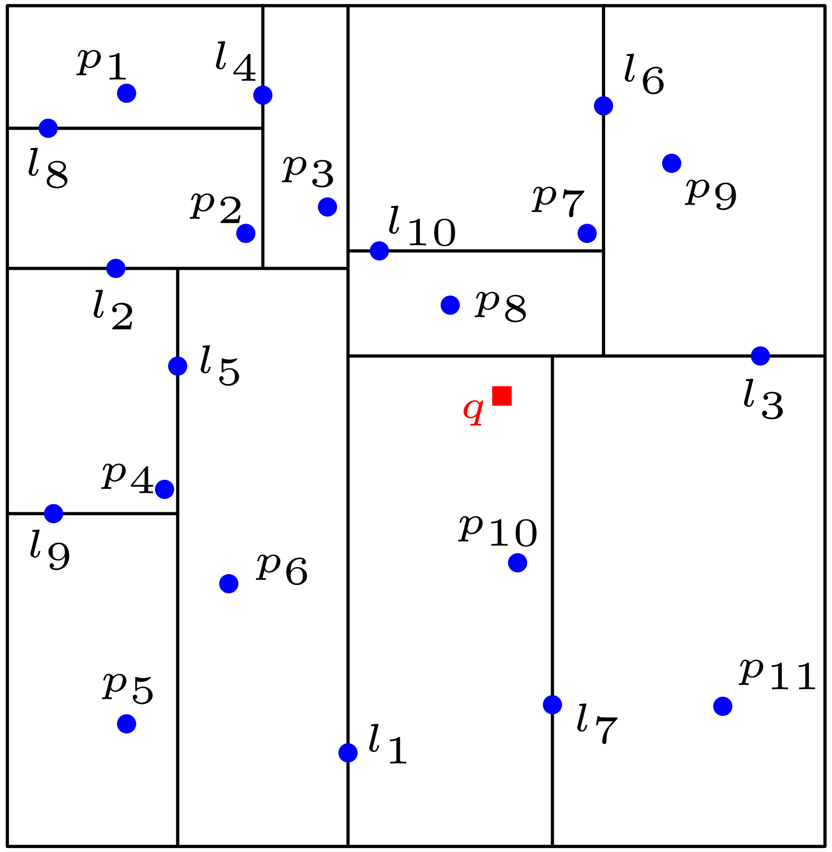
H orthogonal to coordinate axis (cycle through coordinates)

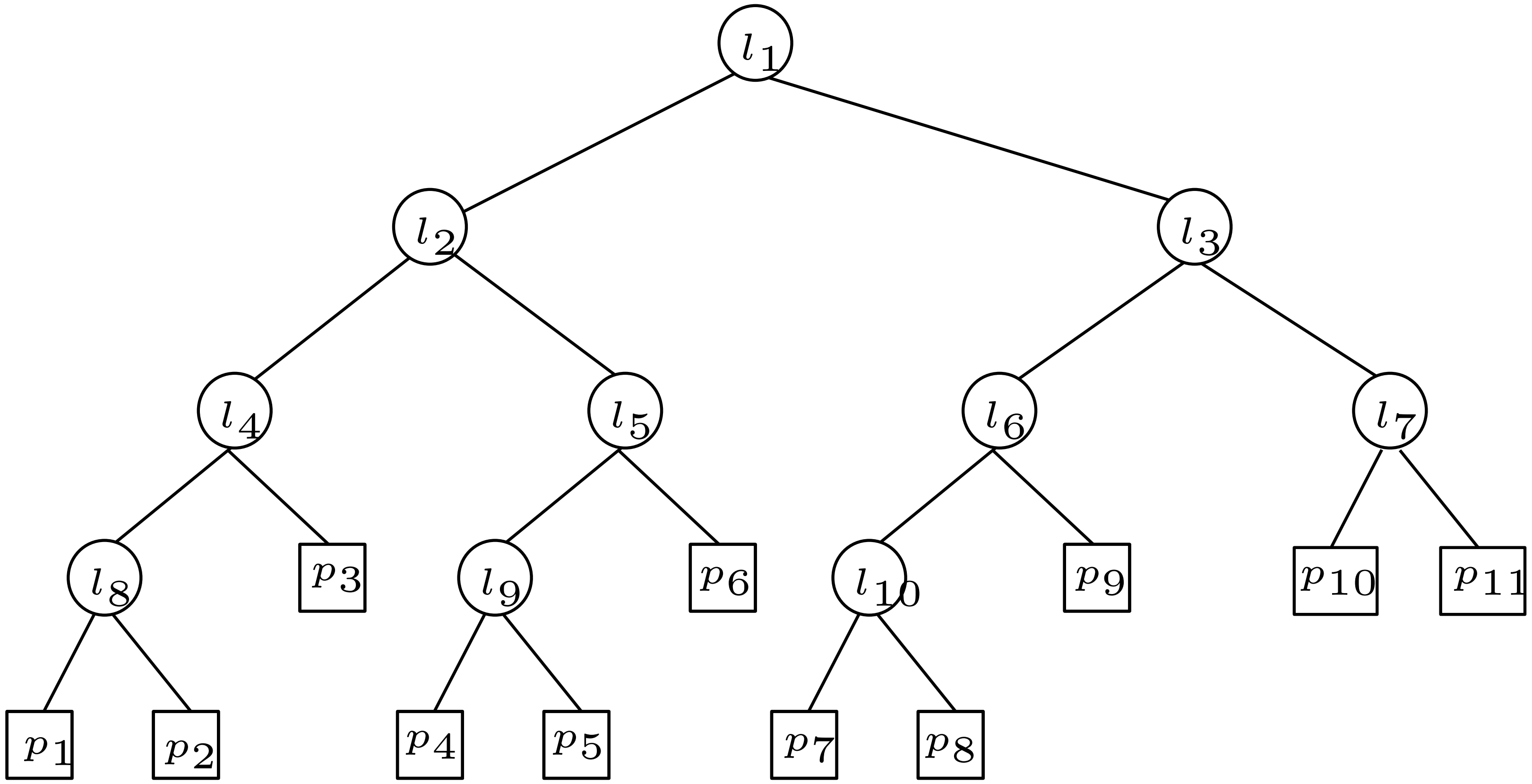
H goes through the median in the considered coordinate



$(n_0 = 1)$

Example





l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Usage for NN search

Strategy 1: defeatist search

$\mathrm{d_{min}} := \infty$ (dist. to pts viewed so far)

search ($node$): ($node = root$ initially)

if $node = leaf$:

$\mathrm{d_{min}} := \min\{\mathrm{d}_{min},\ \min_{p \in node.batch} \mathrm{d}(q,p)\}$

else:

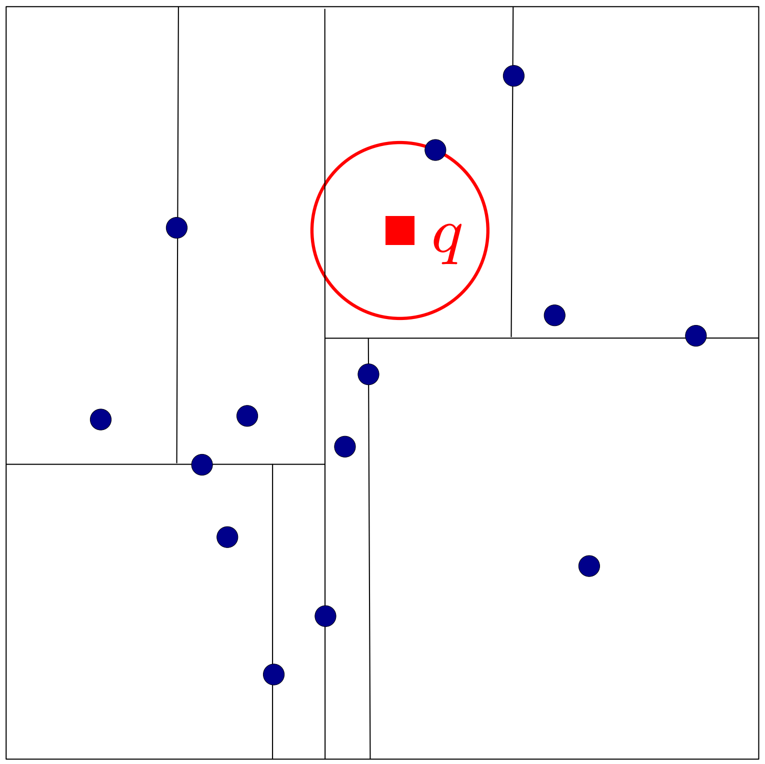
$\mathrm{d_{min}} := \min\{\mathrm{d_{min}}, \mathrm{d}(q, node.point)\}$

if q on "left" side of $node.H$

recurse on $node.left$

else (q on "right" side of $node.H$)

recurse on $node.right$



$(n_0 = 1)$

Usage for NN search

Strategy 1: defeatist search

$\mathrm{d}_{\min} := \infty$ (dist. to pts viewed so far)

Query time: $O(d\,(n_0 + \log \frac{n}{n_0}))$

search ($node$): ($node = root$ initially)

if $node = leaf$:

$\mathrm{d}_{\min} := \min\{\mathrm{d}_{min},\ \min_{p \in node.batch} \mathrm{d}(q, p)\}$

else:

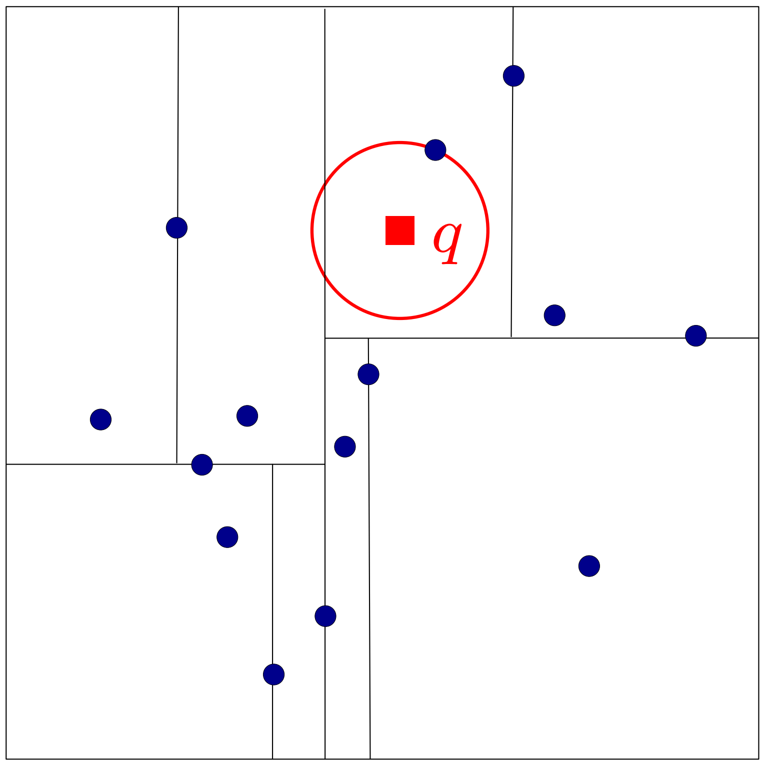
$\mathrm{d}_{\min} := \min\{\mathrm{d}_{\min}, \mathrm{d}(q, node.point)\}$

if q on "left" side of $node.H$

recurse on $node.left$

else (q on "right" side of $node.H$)

recurse on $node.right$



$(n_0 = 1)$

Usage for NN search

Strategy 1: defeatist search

$\mathrm{d}_{\min} := \infty$ (dist. to pts viewed so far)

search $(node)$: $(node = root$ initially)

if $node = leaf$:

$\mathrm{d}_{\min} := \min\{\mathrm{d}_{min},\ \min_{p\in node.batch} \mathrm{d}(q,p)\}$

else:

$\mathrm{d}_{\min} := \min\{\mathrm{d}_{\min}, \mathrm{d}(q, node.point)\}$

if q on "left" side of $node.H$

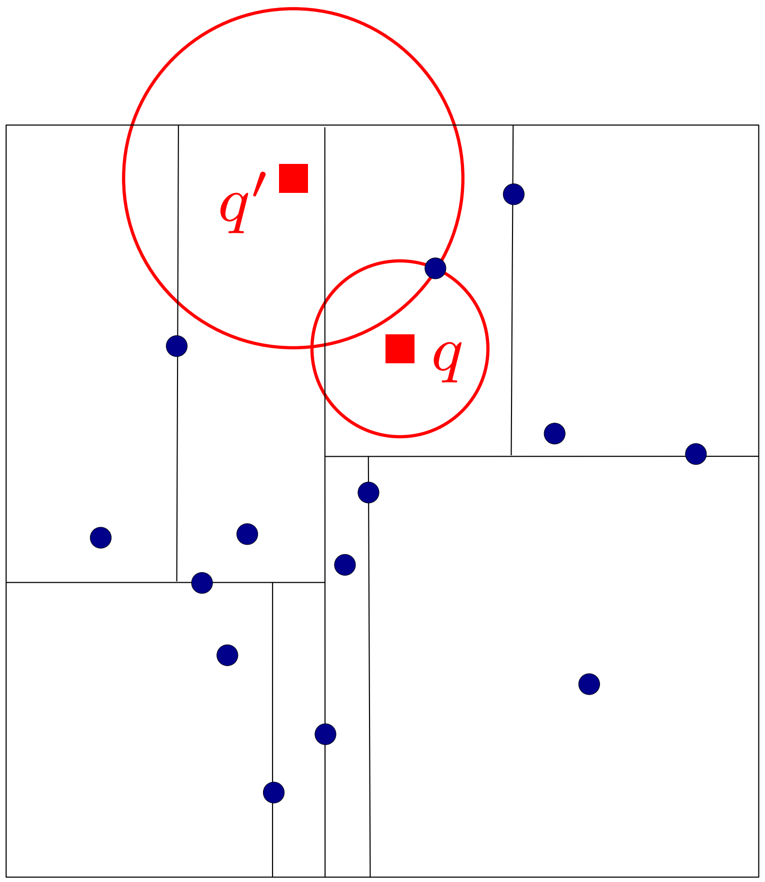
recurse on $node.left$

else (q on "right" side of $node.H$)

recurse on $node.right$

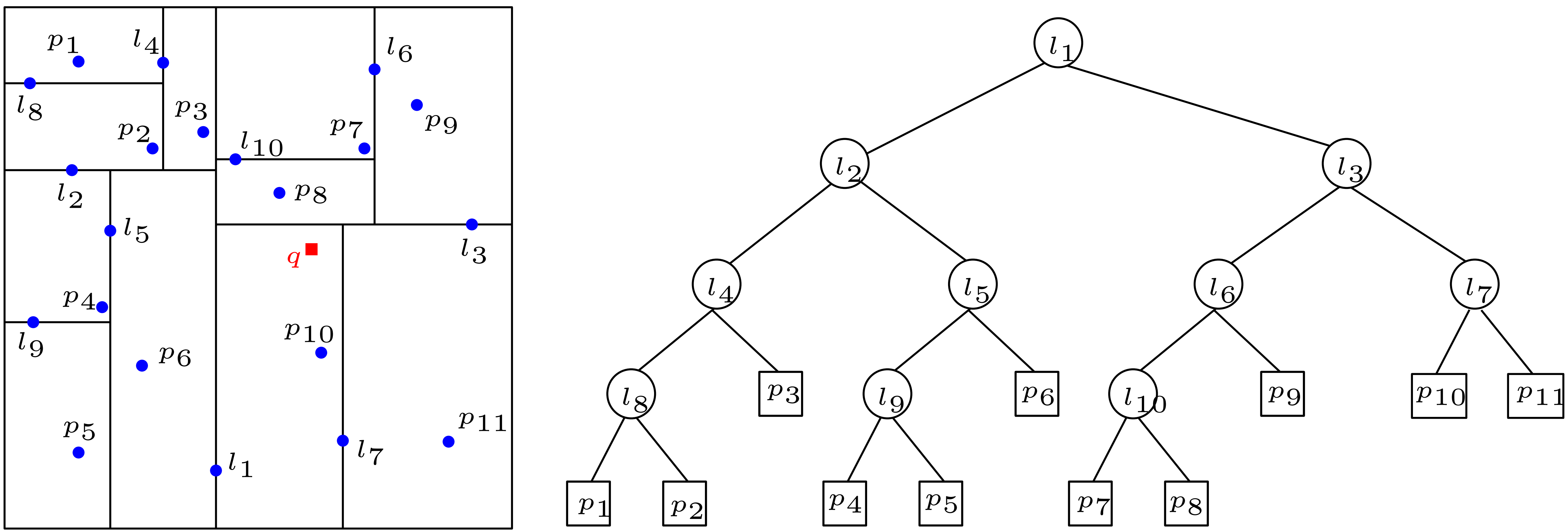
Query time: $O(d\,(n_0 + \log \frac{n}{n_0}))$

May fail!



$(n_0 = 1)$

Example

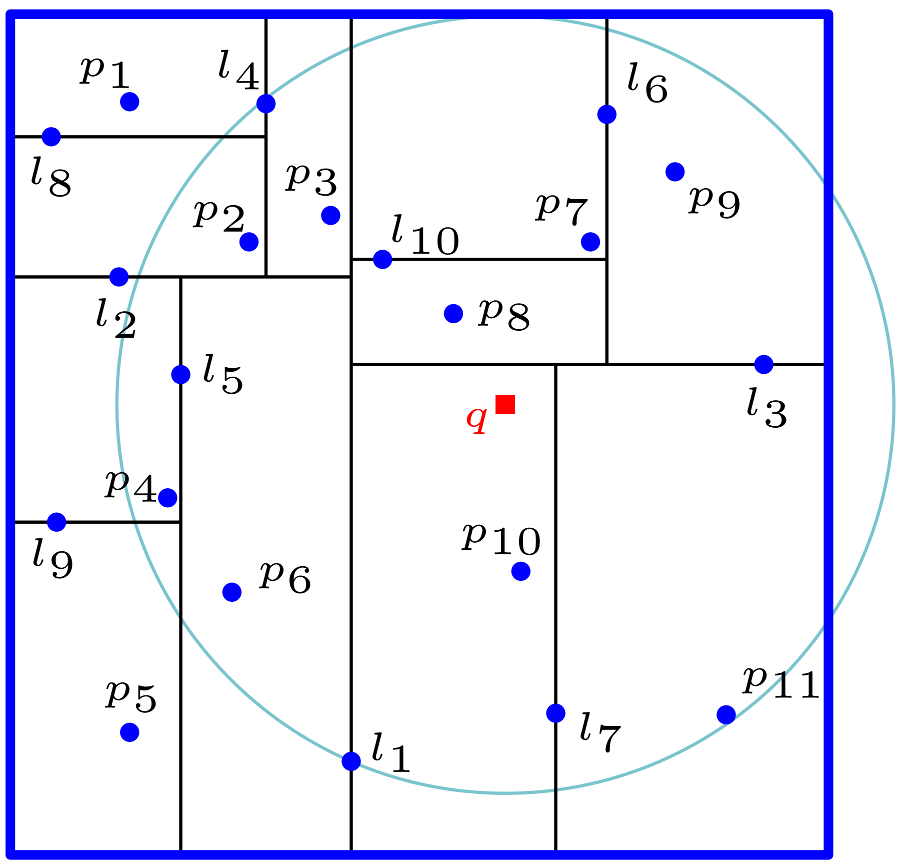


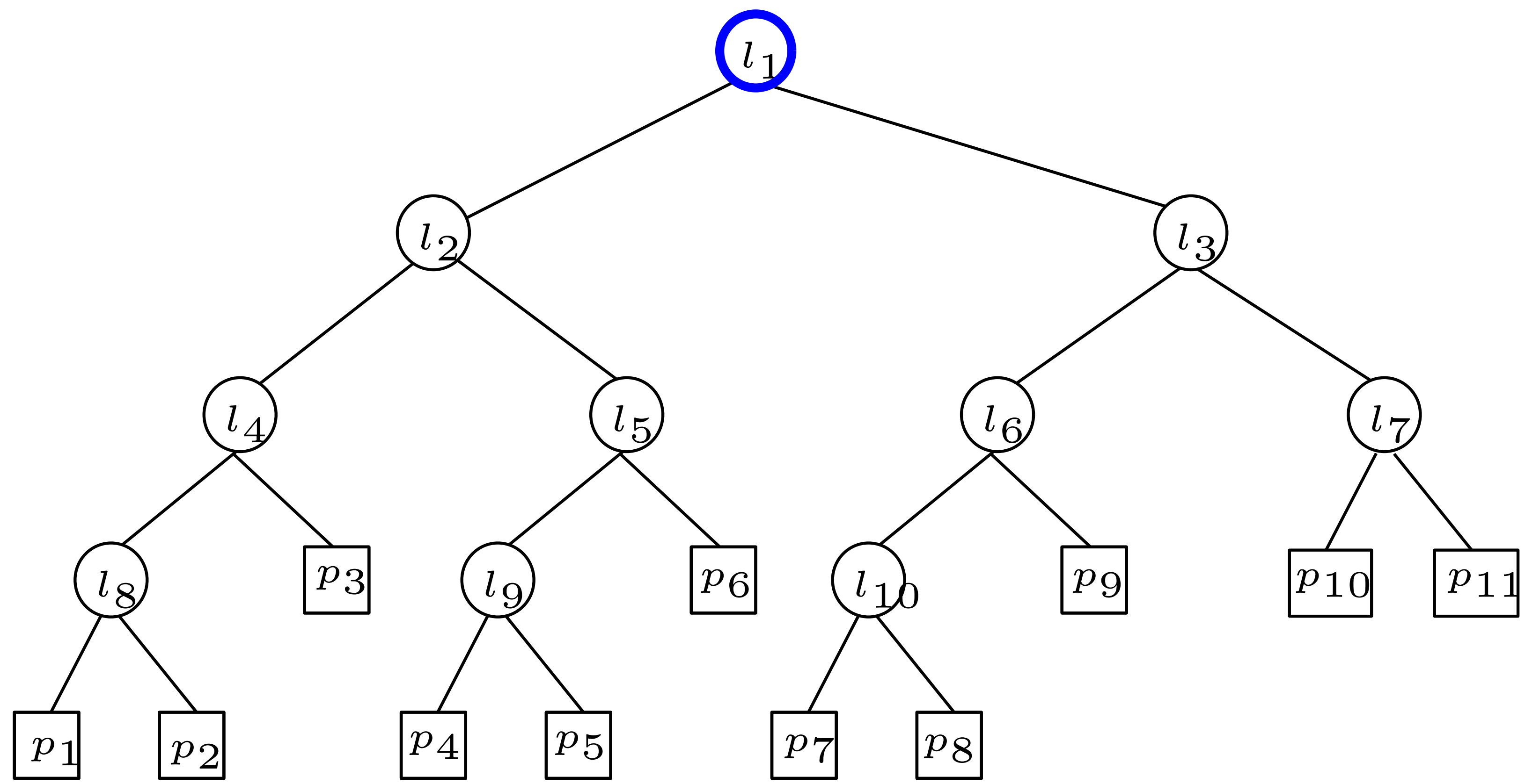
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



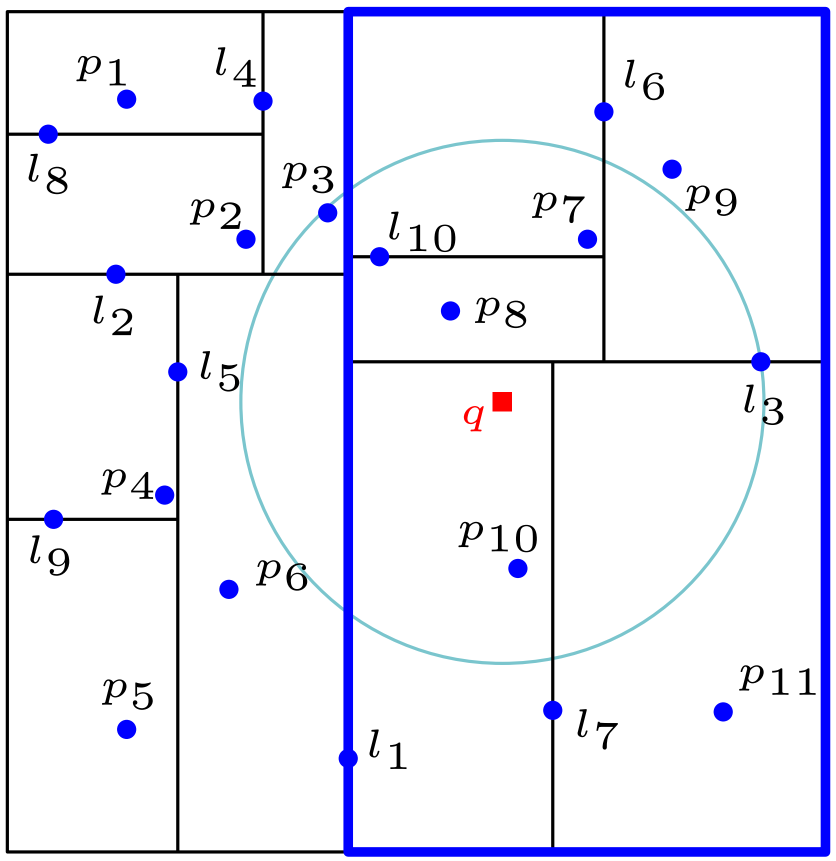


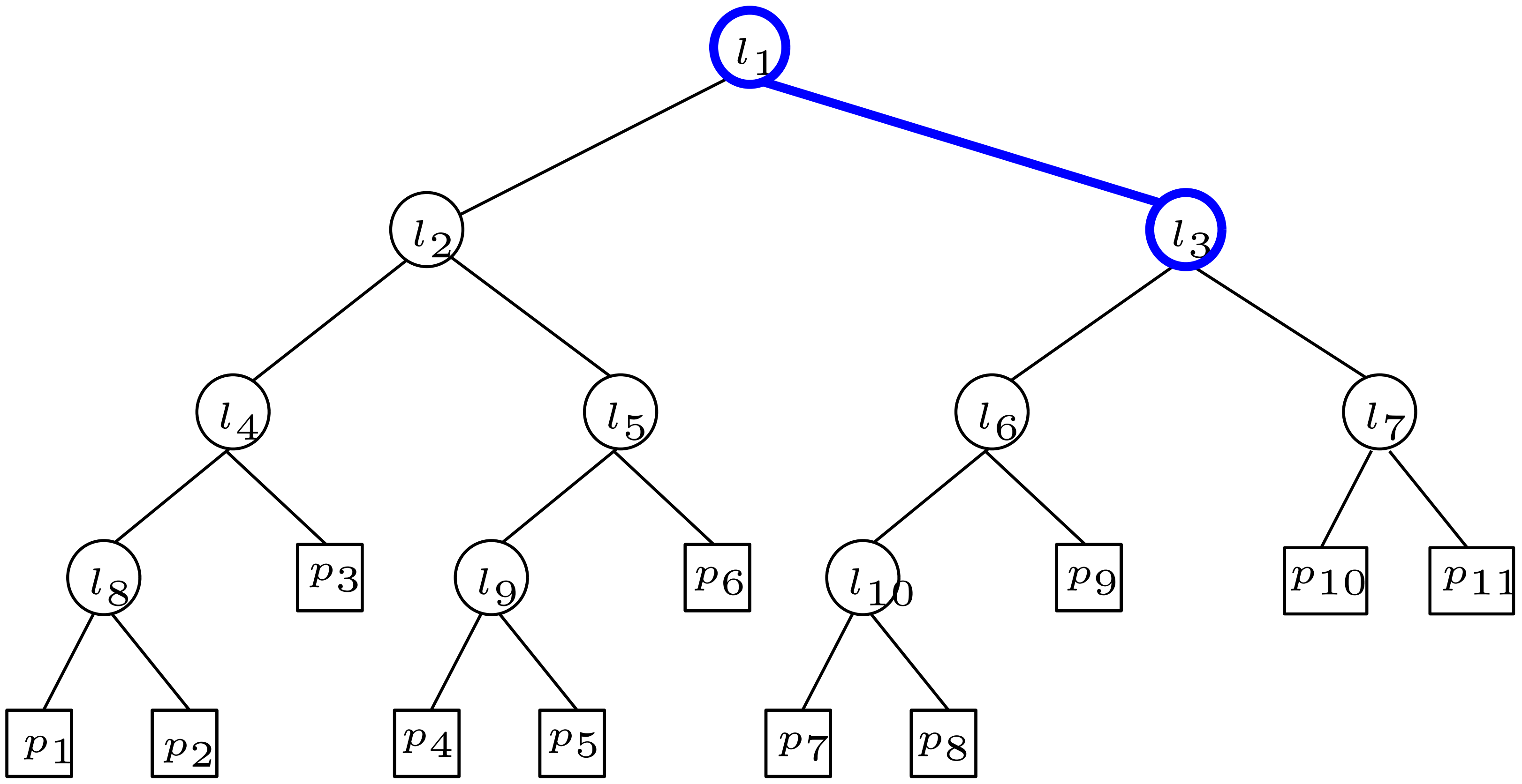
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



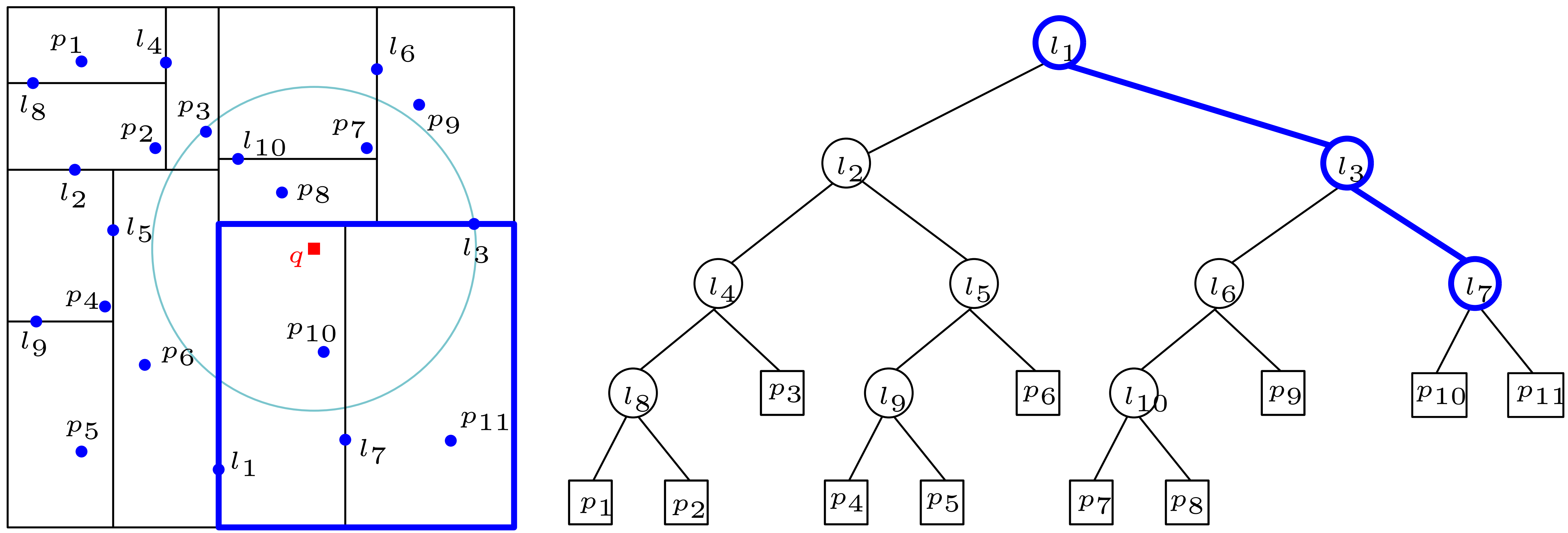


l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example

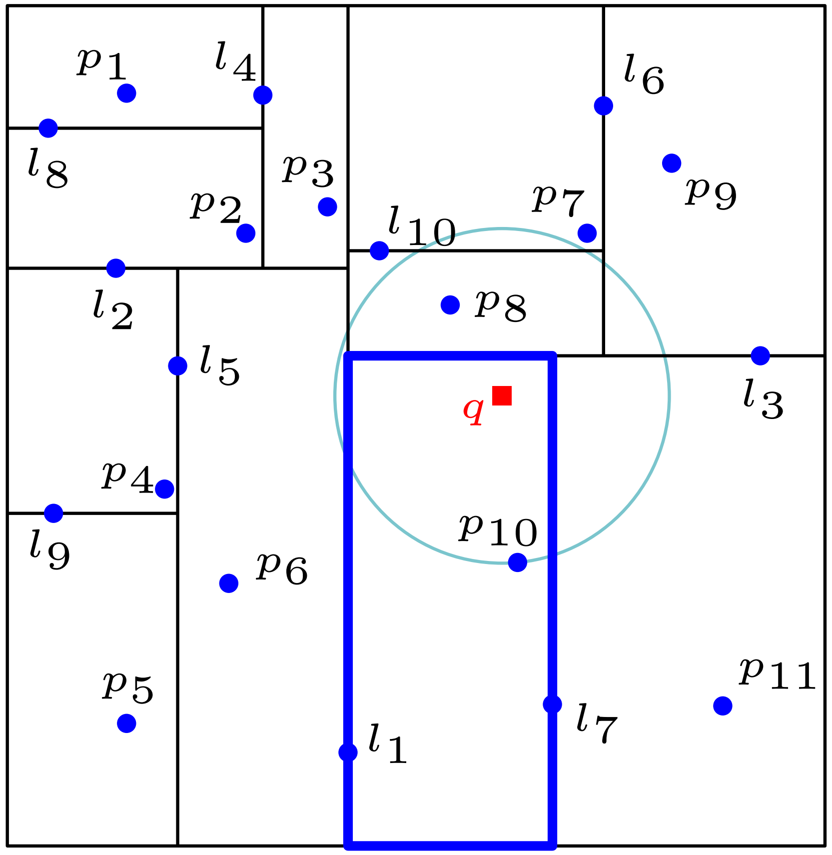


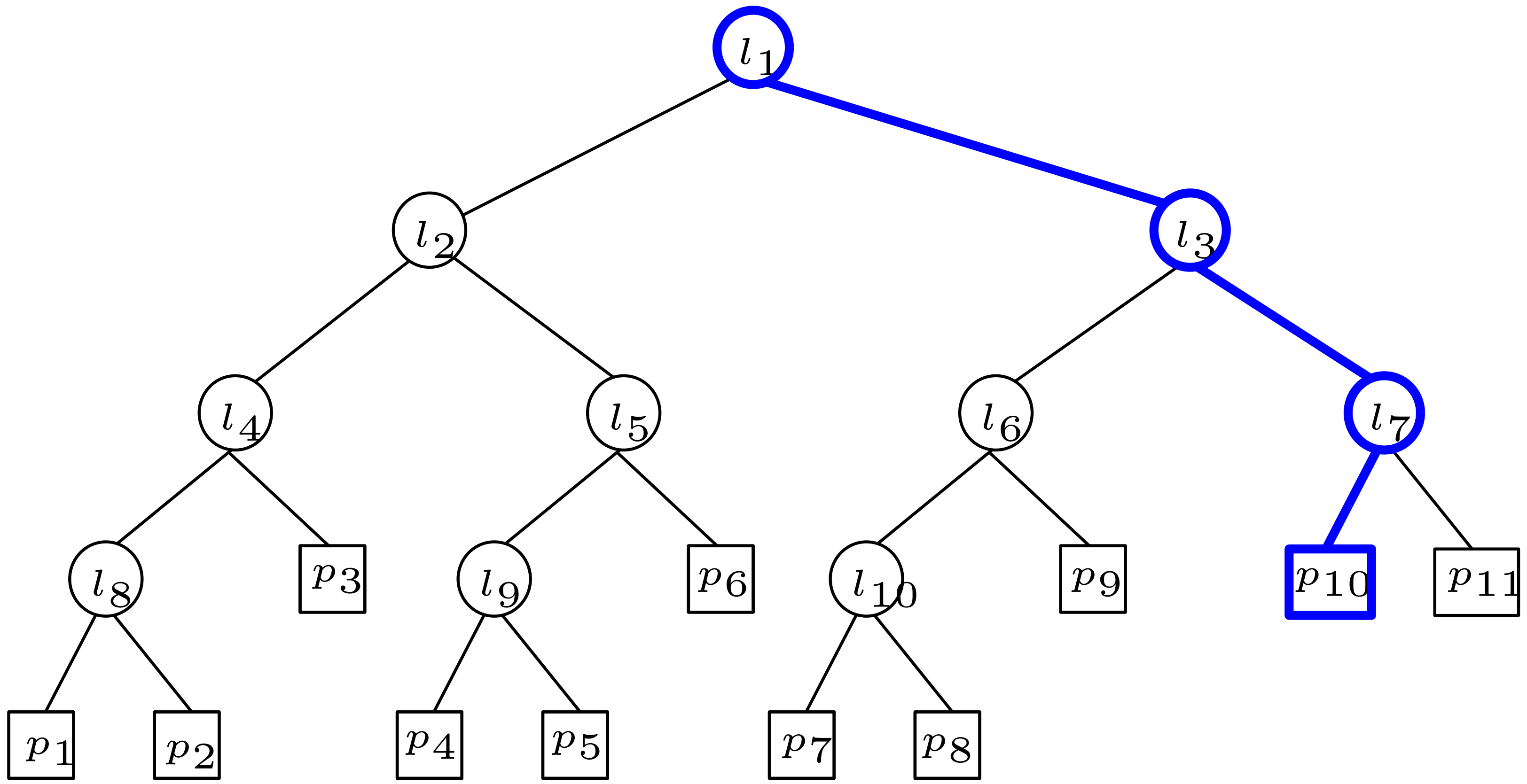
l_i: data at internal node

(note: left-right labels are arbitrary)

p_i: data at leaf node

Example





l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Usage for NN search

Strategy 2: backtracking search

$\mathrm{d}_{\min} := \infty$ (dist. to pts viewed so far)

search $(node)$: $(node = root$ initially)

if $node = leaf$:

$\mathrm{d}_{\min} := \min\{\mathrm{d}_{\min},\ \min_{p \in node.batch} \mathrm{d}(q,p)\}$

else:

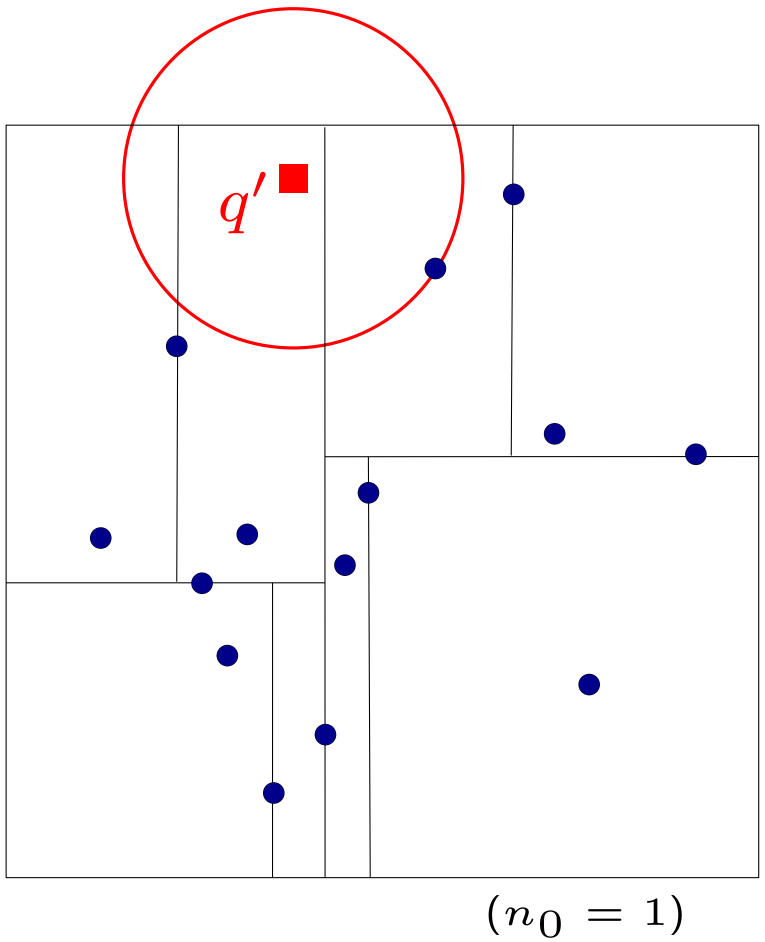
$\mathrm{d}_{\min} := \min\{\mathrm{d}_{\min}, \mathrm{d}(q, node.point)\}$

if $B(q, \mathrm{d}_{min})$ intersects "left" side of $node.H$

recurse on $node.left$

if $B(q, \mathrm{d}_{\min})$ intersects "right" side of $node.H$

recurse on $node.right$



Usage for NN search

Strategy 2: backtracking search

$\mathrm{d}_{\min} := \infty$ (dist. to pts viewed so far)

search $(node)$: $(node = root$ initially)

if $node = leaf$:

$\mathrm{d}_{\min} := \min\{\mathrm{d}_{\min},\ \min_{p \in node.batch} \mathrm{d}(q,p)\}$

else:

$\mathrm{d}_{\min} := \min\{\mathrm{d}_{\min}, \mathrm{d}(q, node.point)\}$

if $B(q, \mathrm{d}_{min})$ intersects "left" side of $node.H$

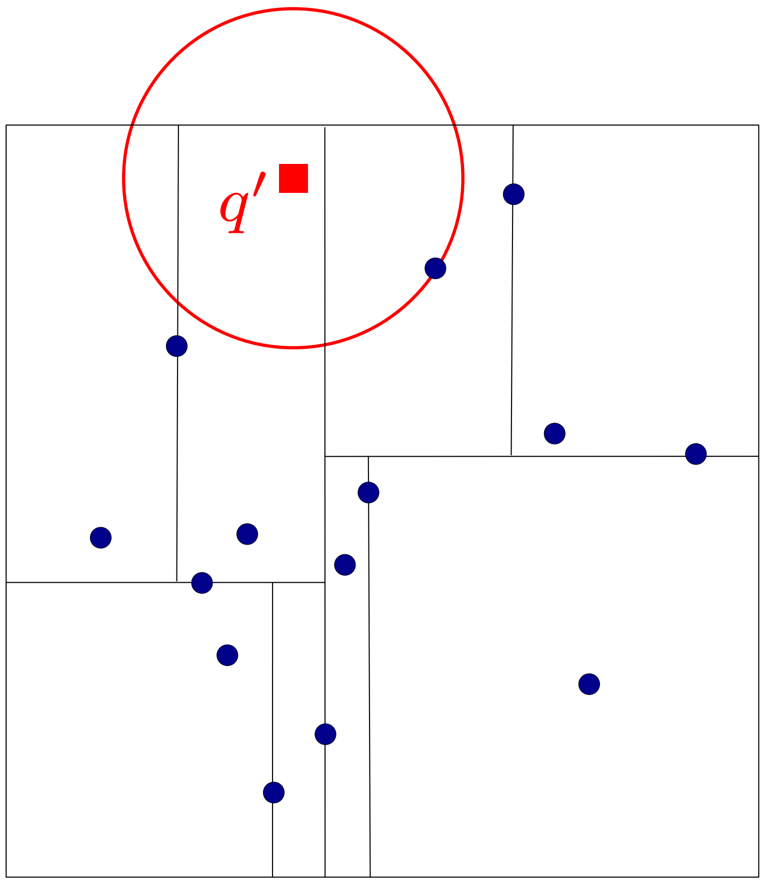
recurse on $node.left$

if $B(q, \mathrm{d}_{\min})$ intersects "right" side of $node.H$

recurse on $node.right$

Always succeeds

$\mathrm{d}_{\min} \geq \mathrm{d}(q, \mathrm{NN}(q)) \Rightarrow B(q, \mathrm{d}_{\min})$ intersects all cells containing $\mathrm{NN}(q)$ in subdivision throughout search



$(n_0 = 1)$

Usage for NN search

Strategy 2: backtracking search

$d_{\min} := \infty$ (dist. to pts viewed so far)

search ($node$): ($node = root$ initially)

if $node = leaf$:

$d_{\min} := \min\{d_{\min},\ \min_{p \in node.batch} d(q,p)\}$

else:

$d_{\min} := \min\{d_{\min}, d(q, node.point)\}$

if $B(q, d_{min})$ intersects "left" side of $node.H$

recurse on $node.left$

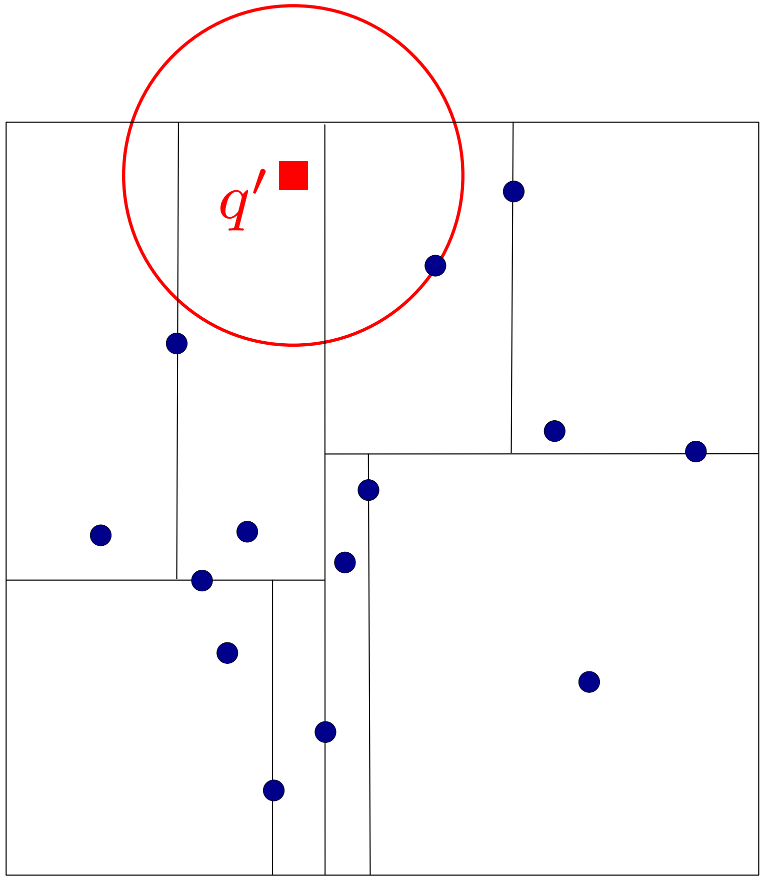
if $B(q, d_{\min})$ intersects "right" side of $node.H$

recurse on $node.right$

Always succeeds

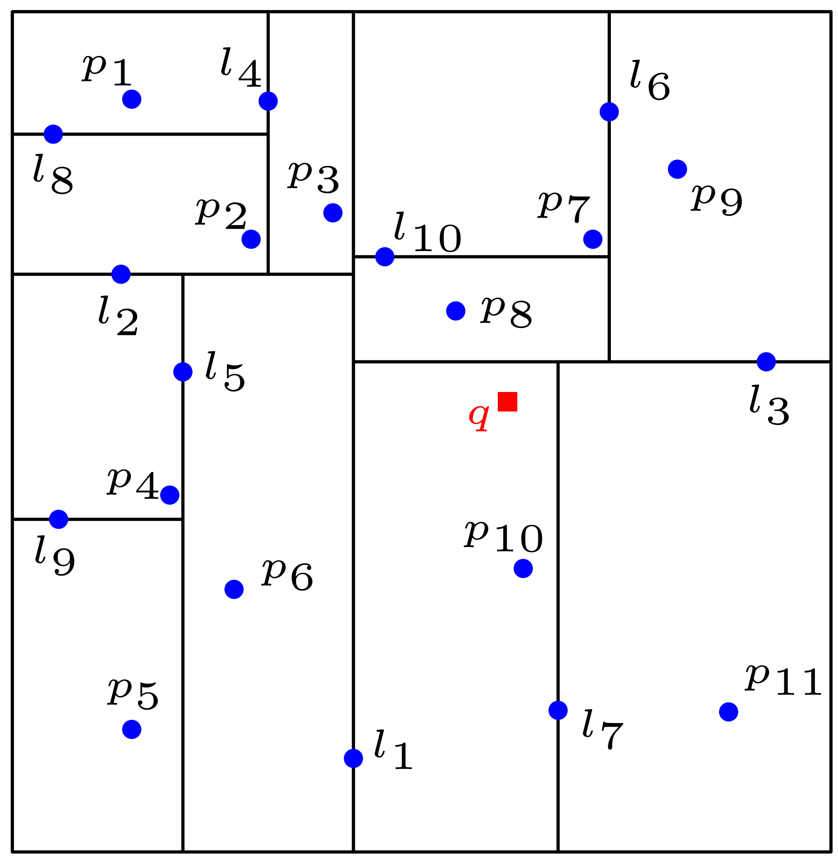
Query time may be up to linear

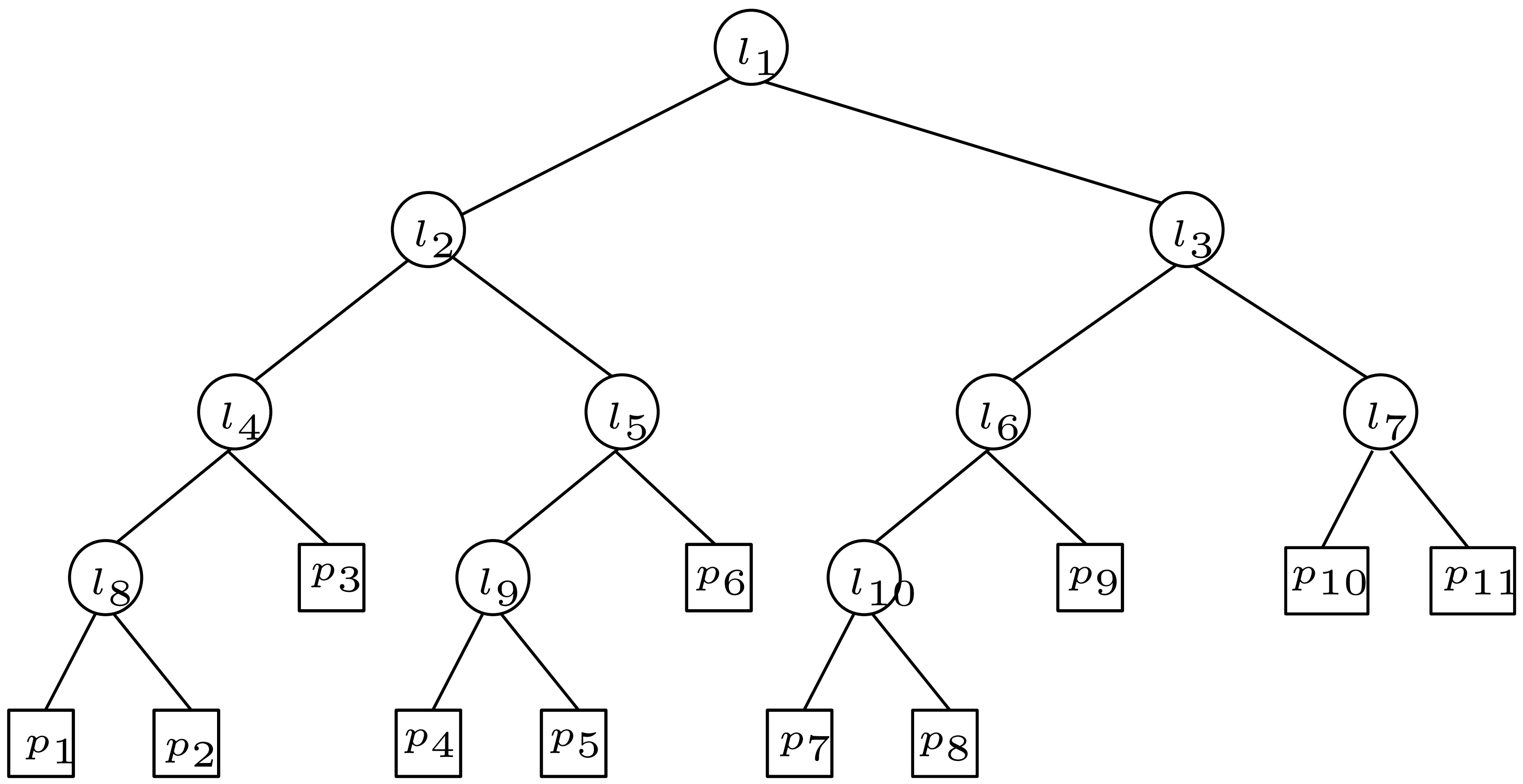
(all cells visited)



$(n_0 = 1)$

Example



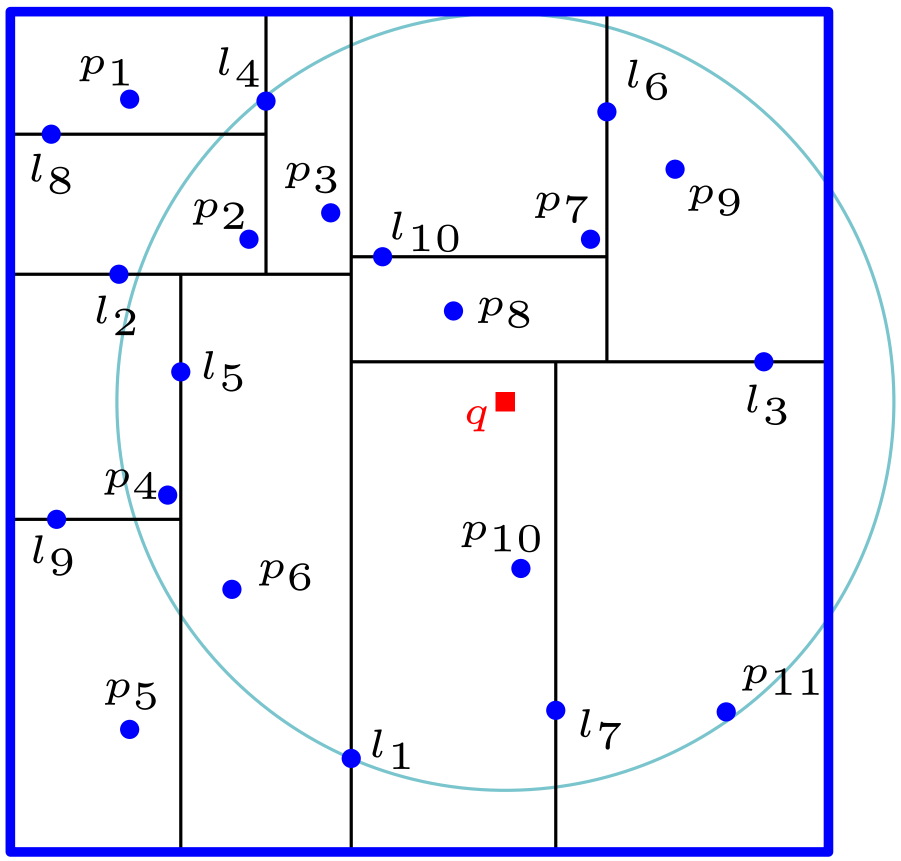


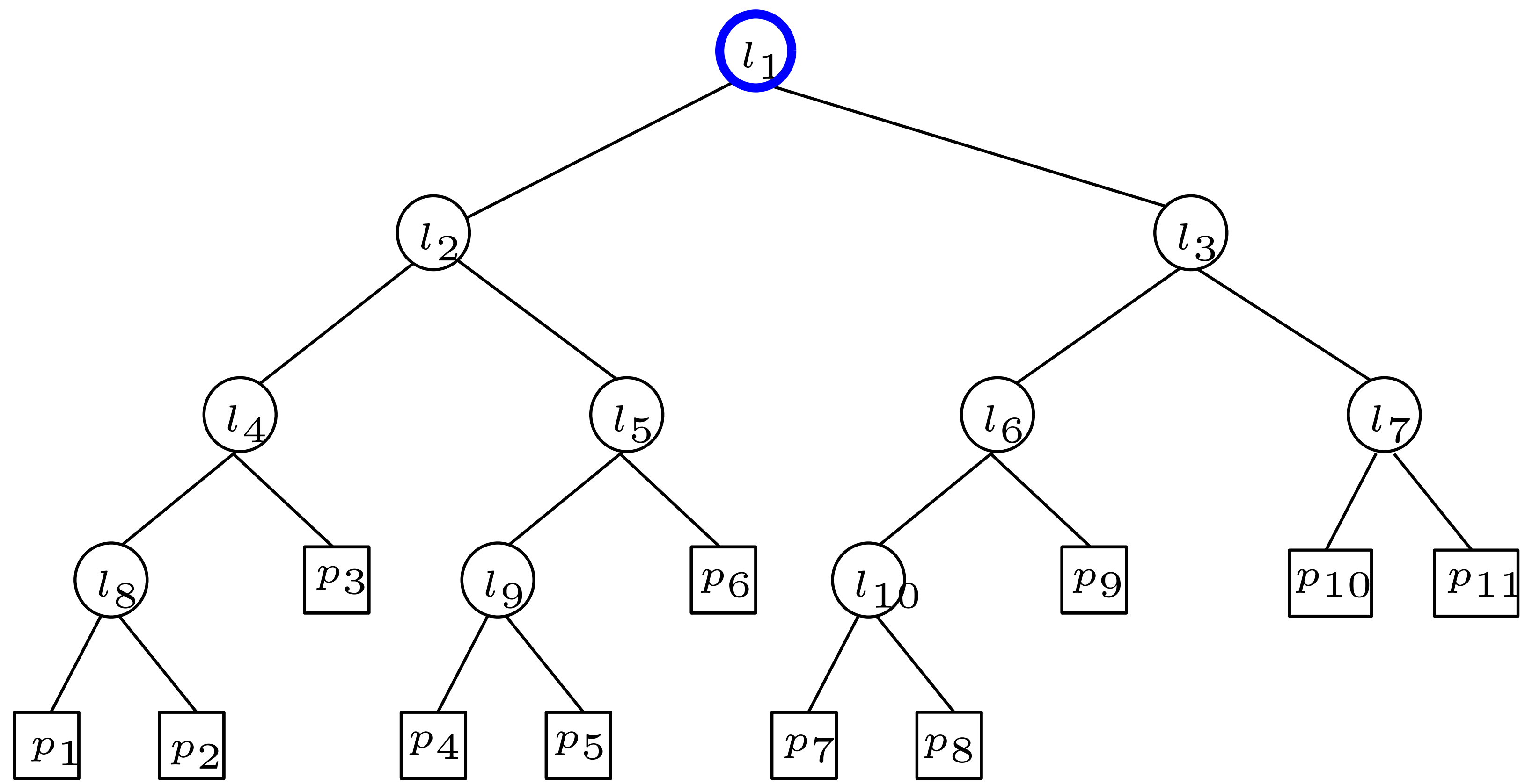
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



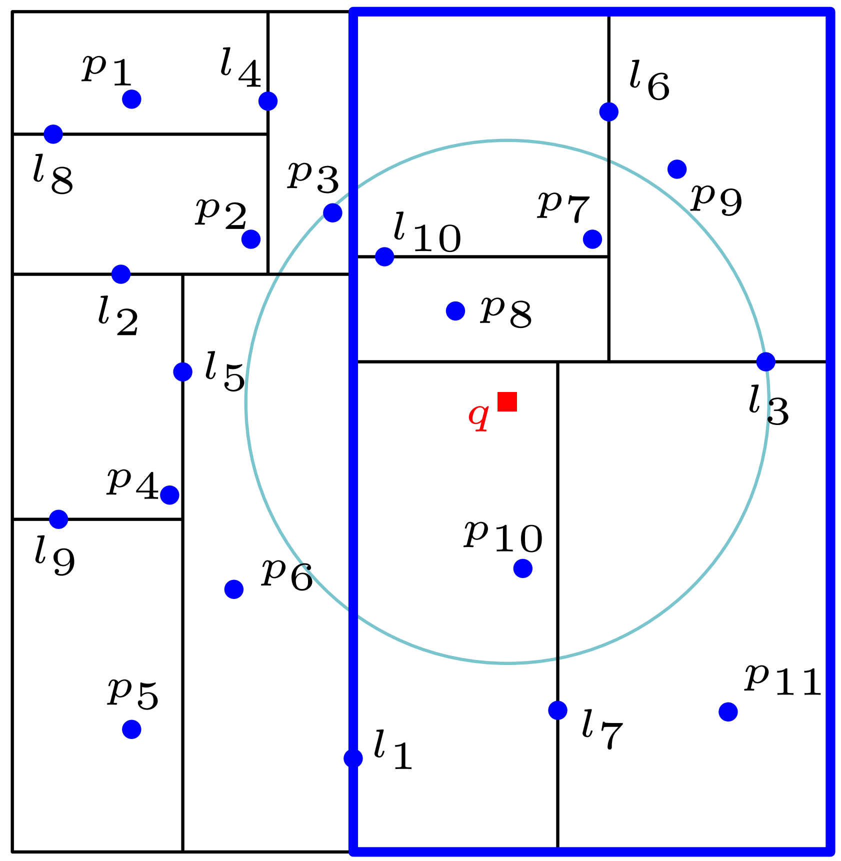


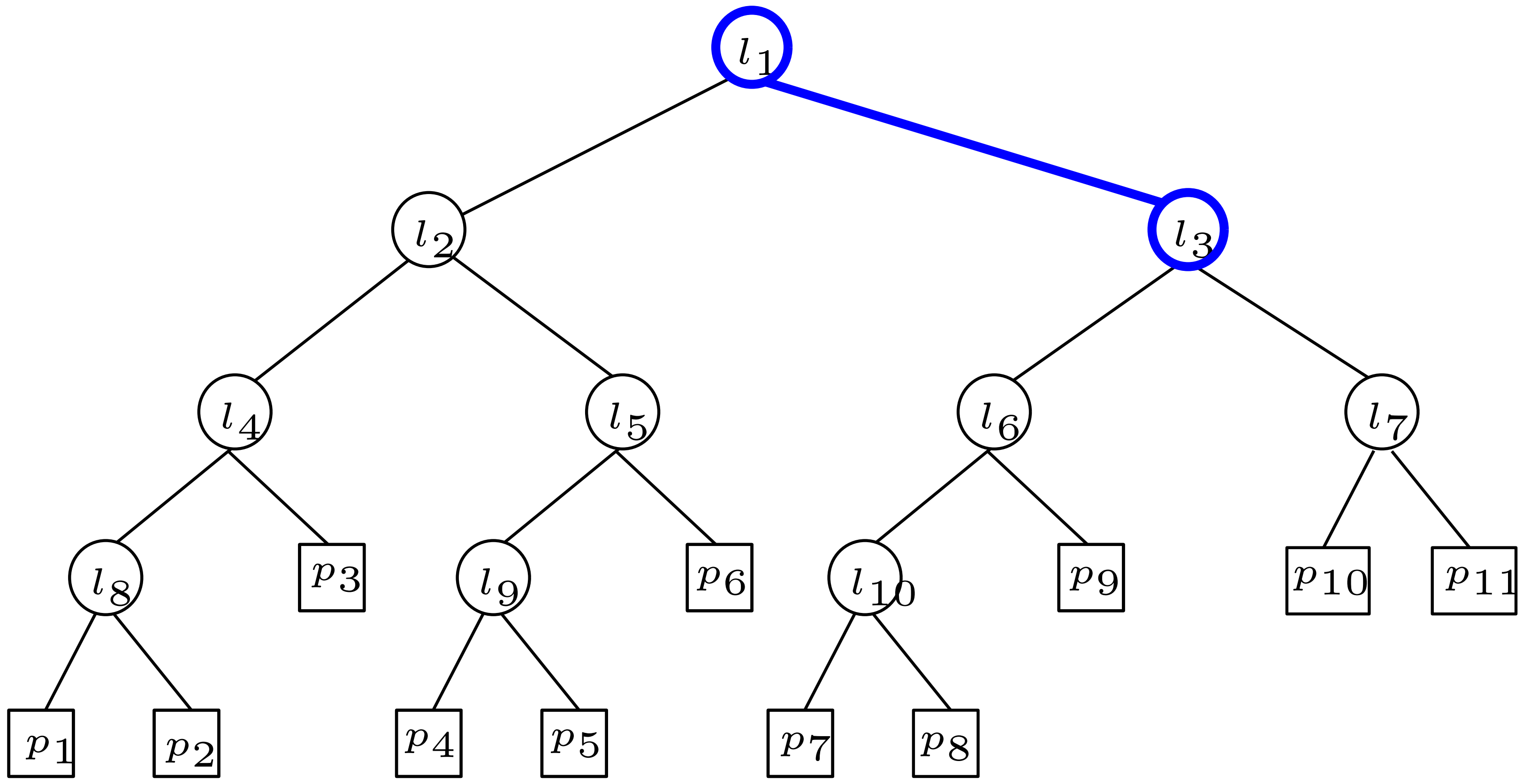
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



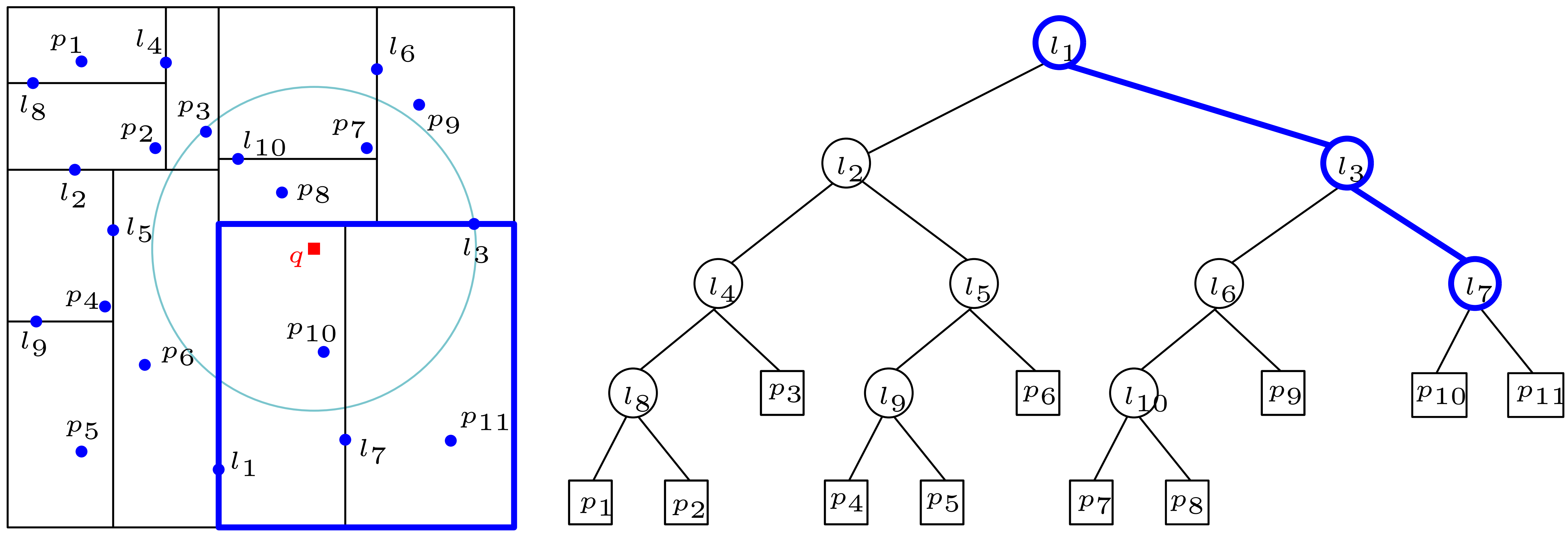


l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example

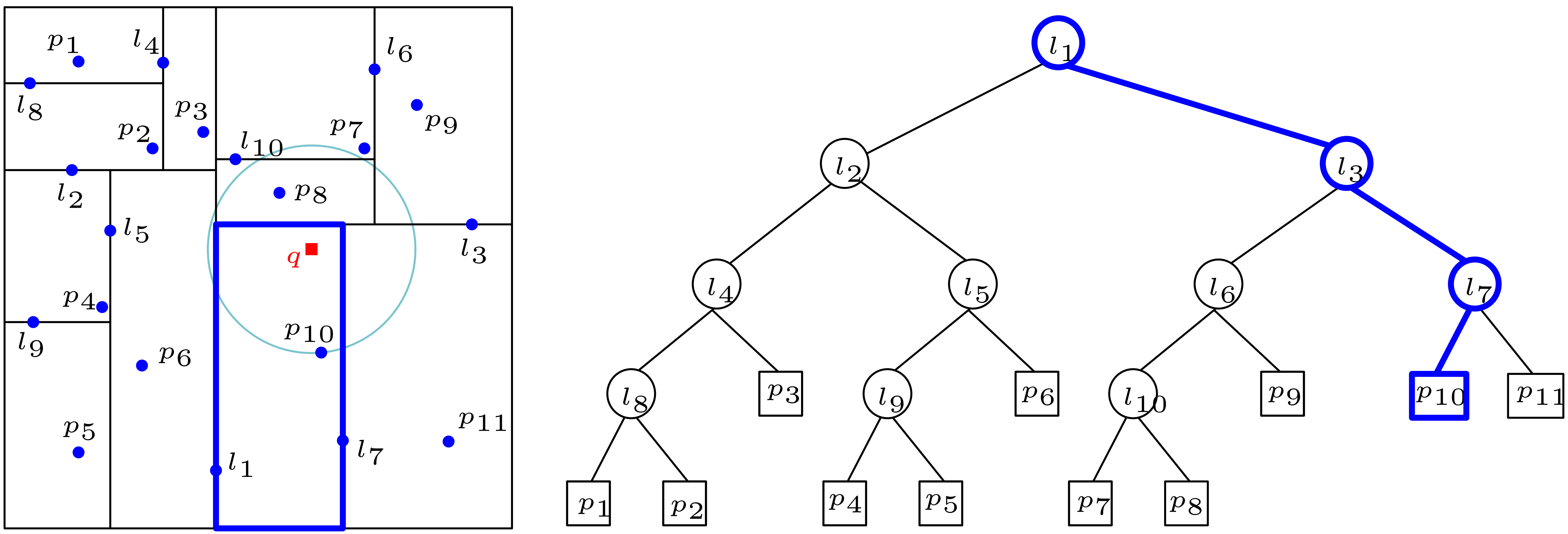


l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example

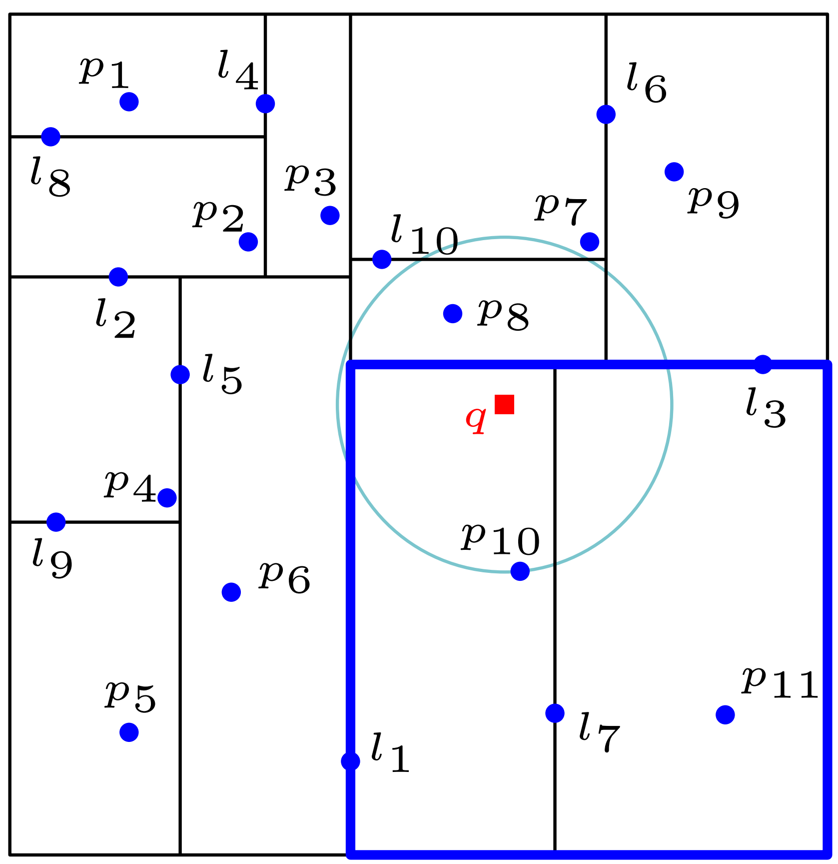


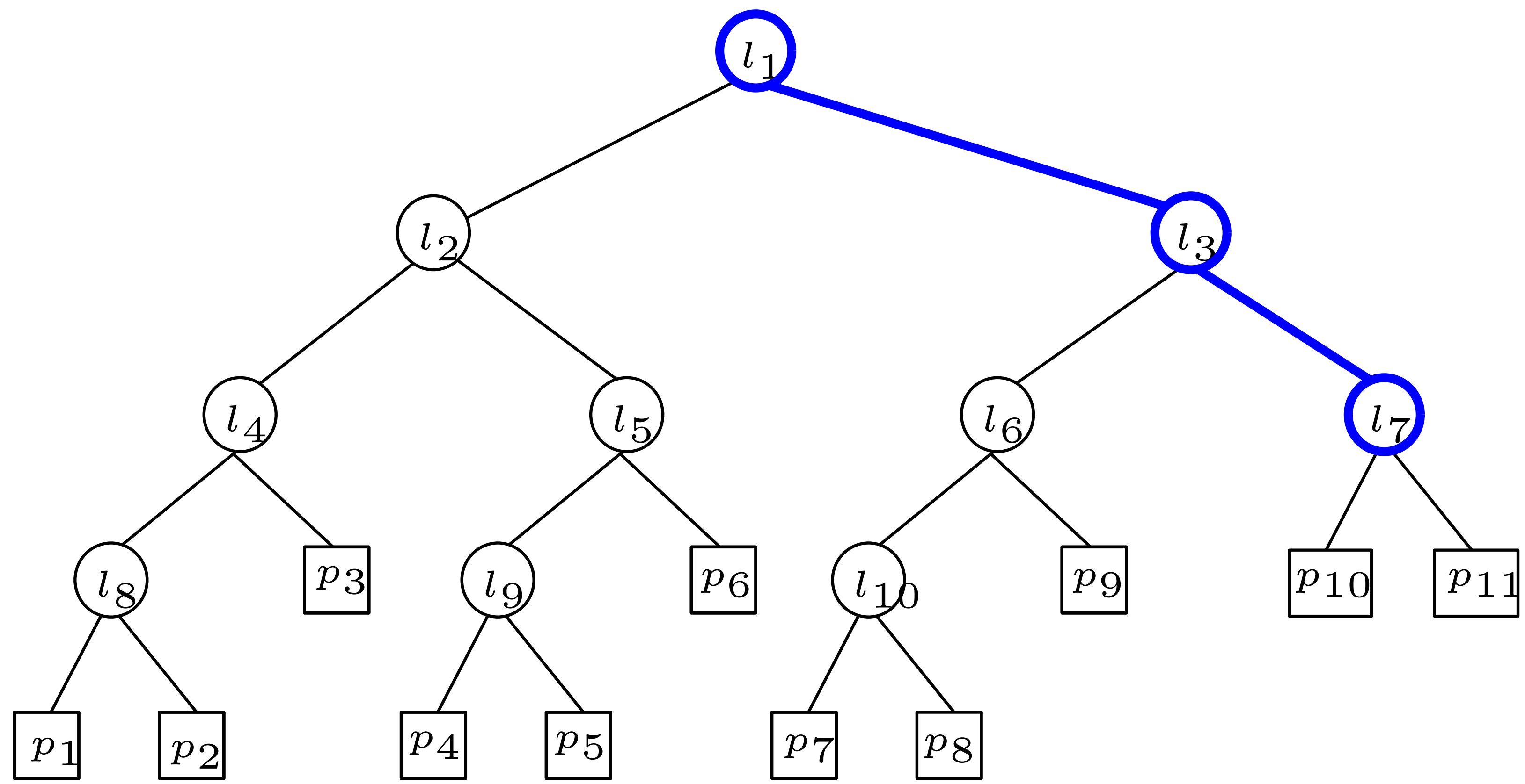
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



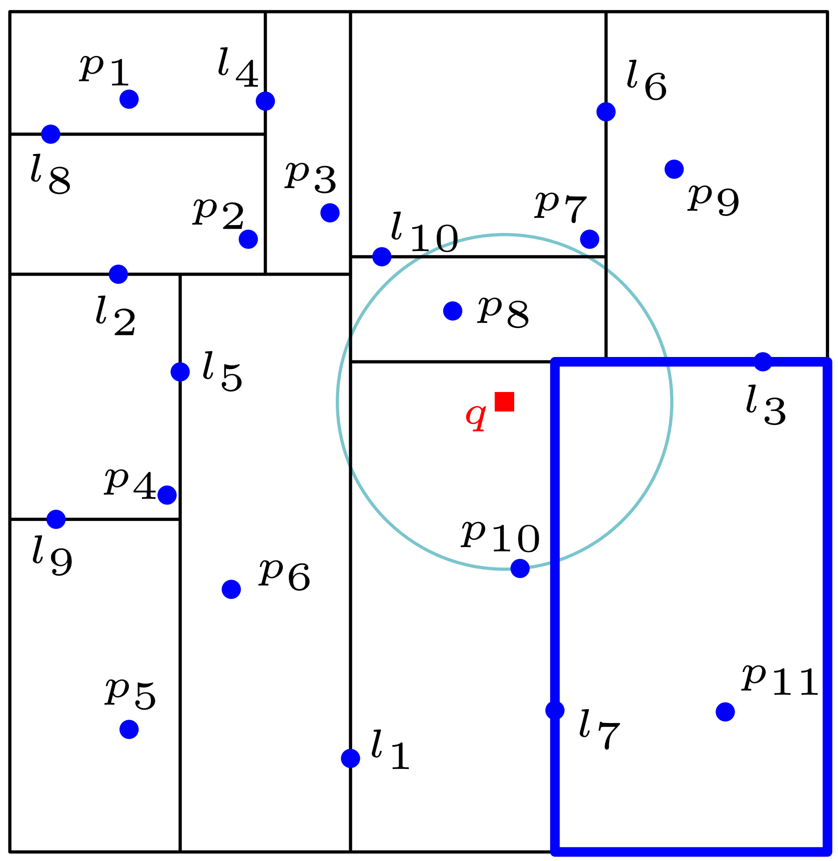


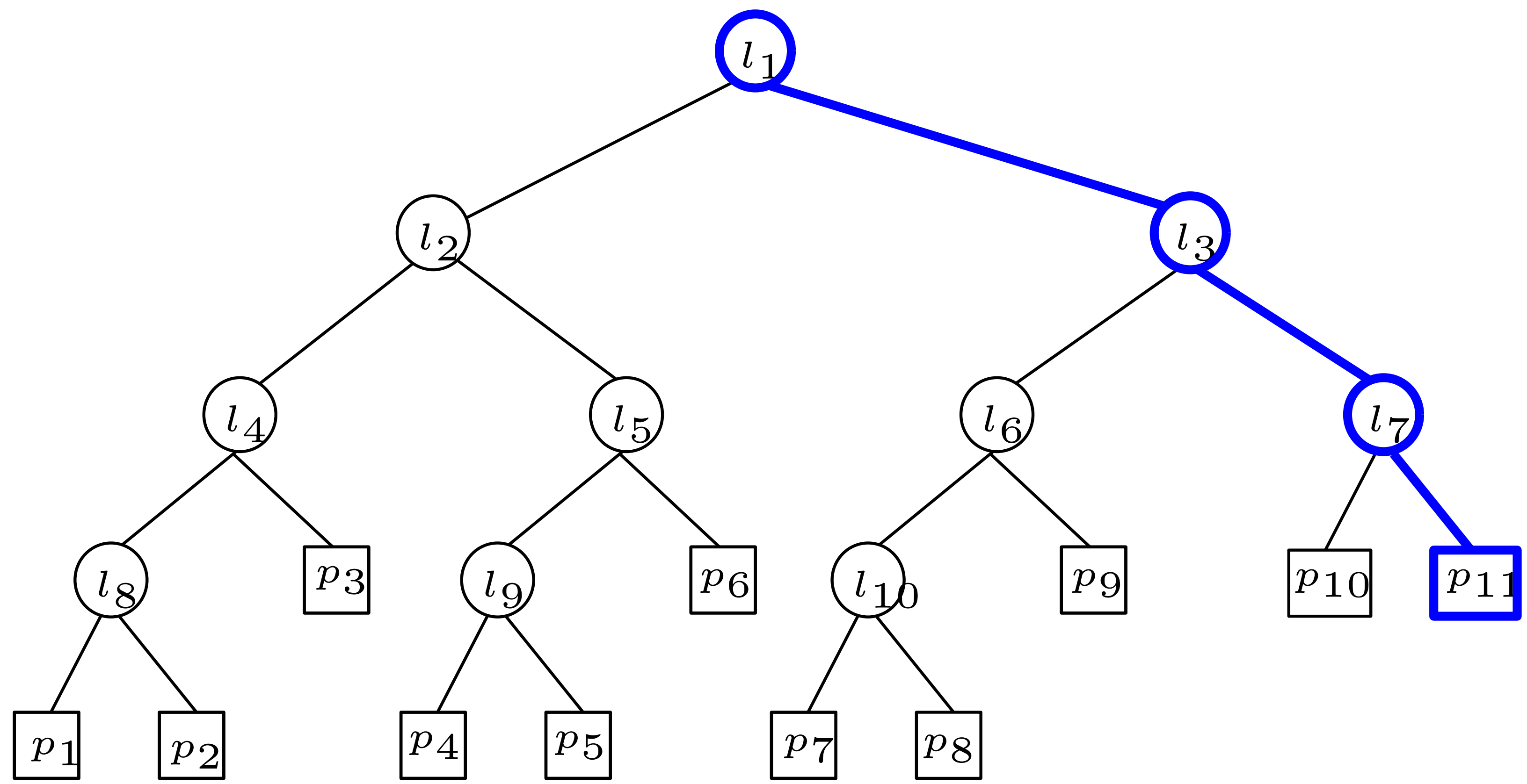
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



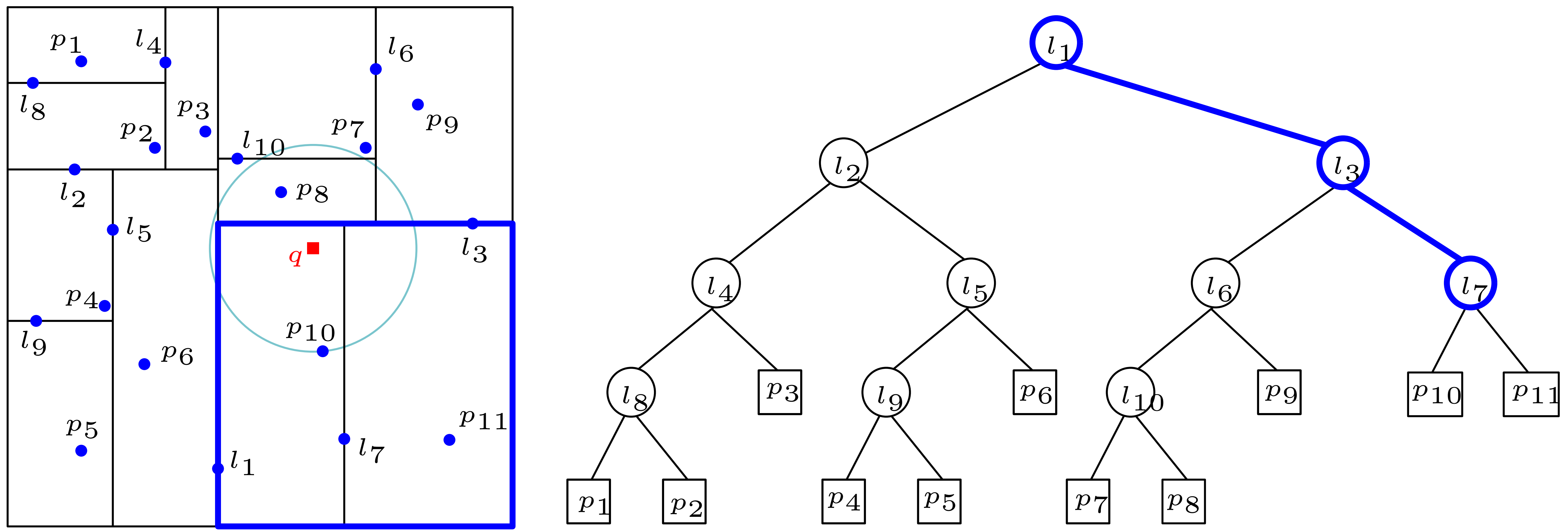


l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example

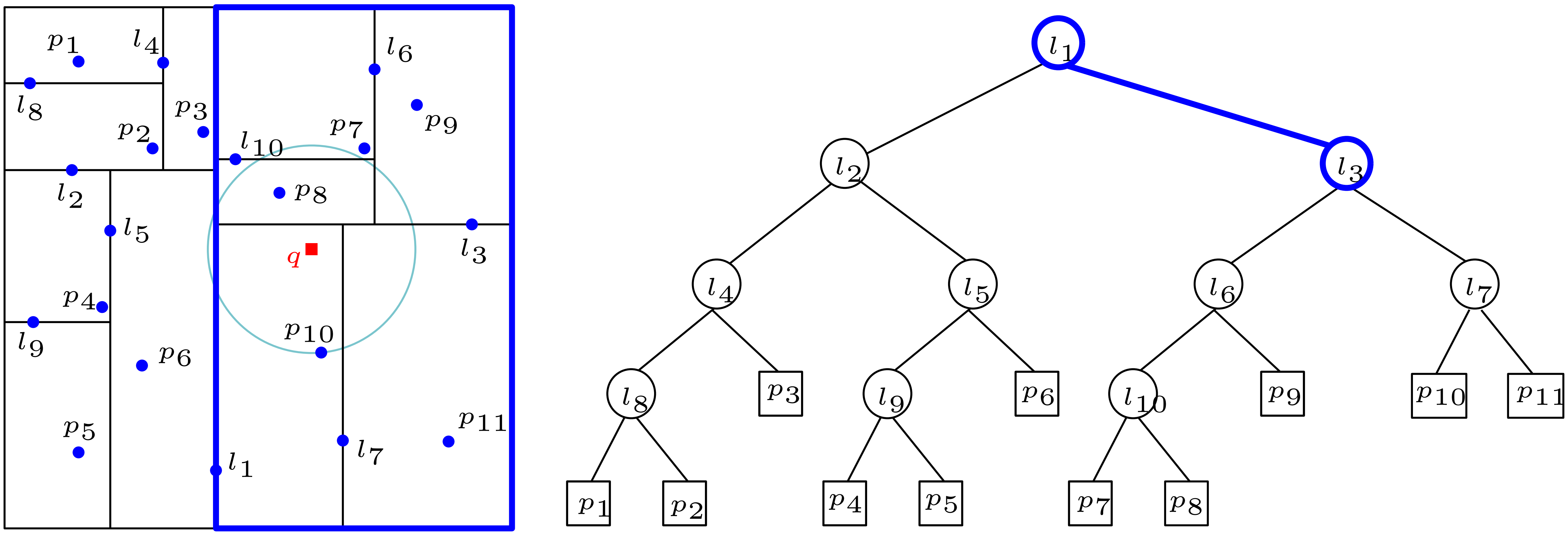


l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example

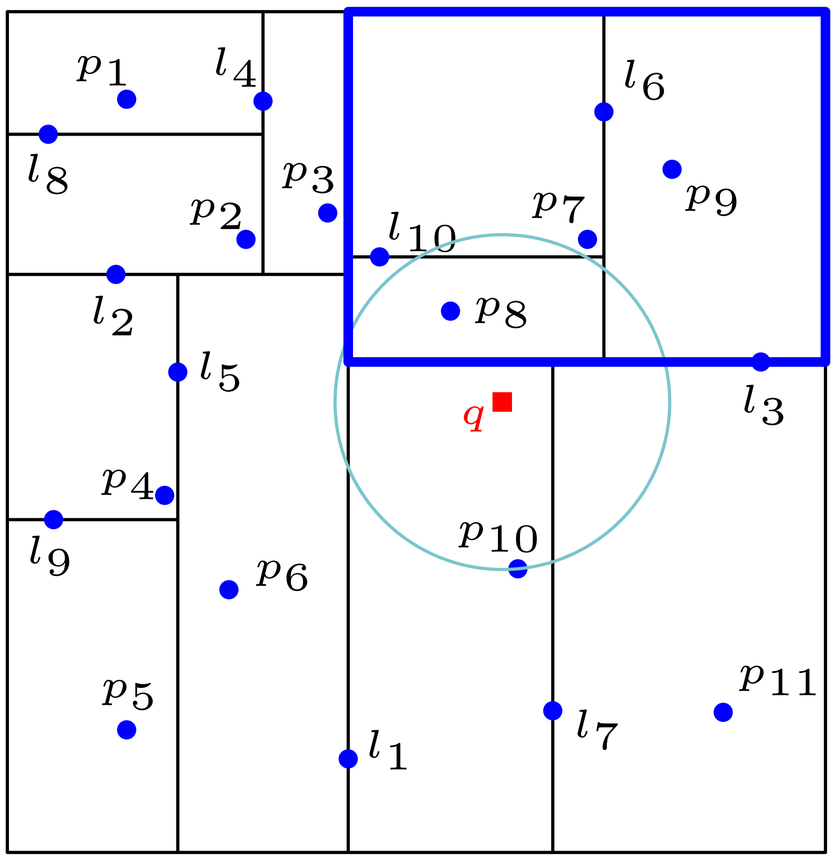


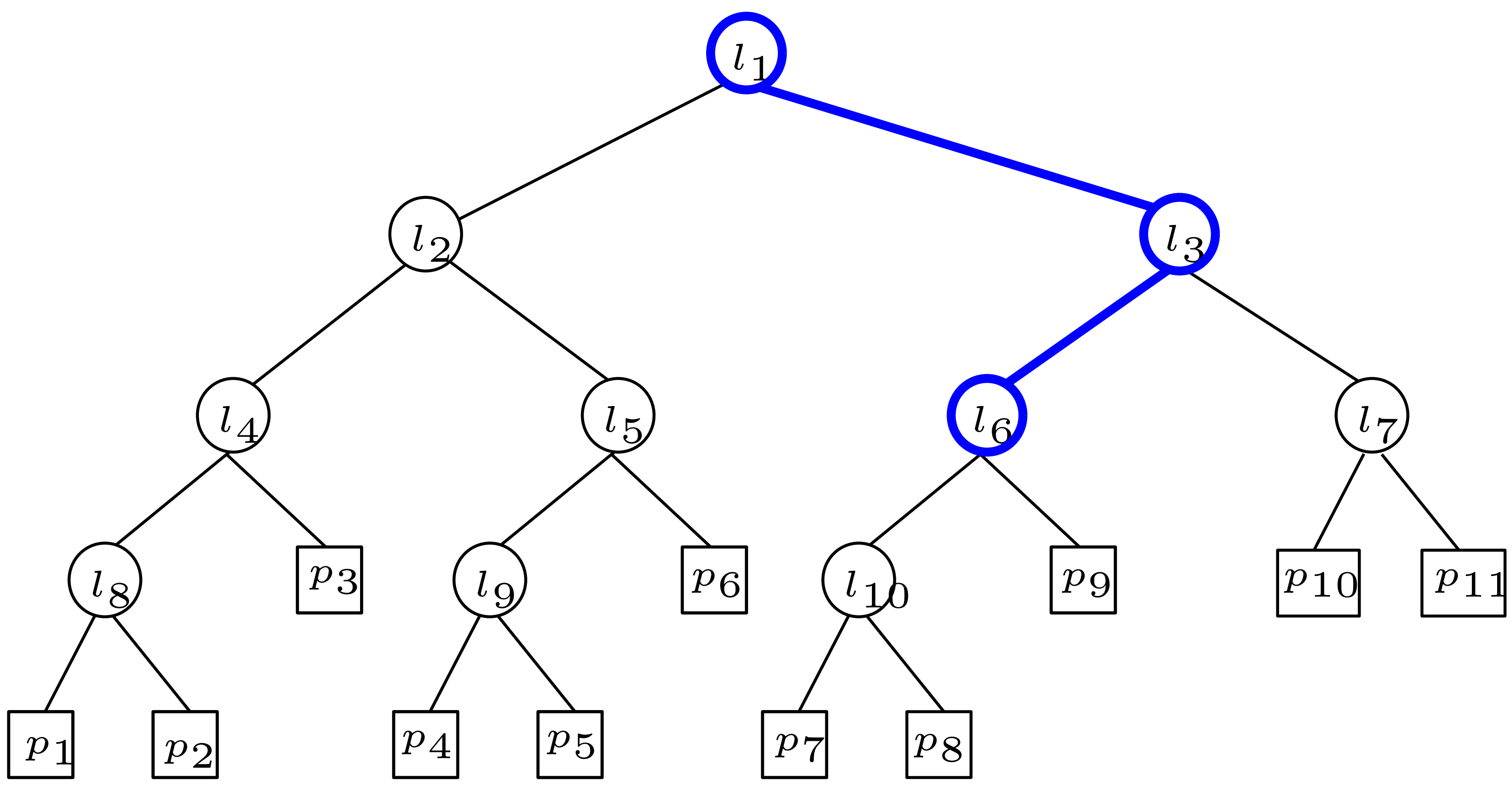
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



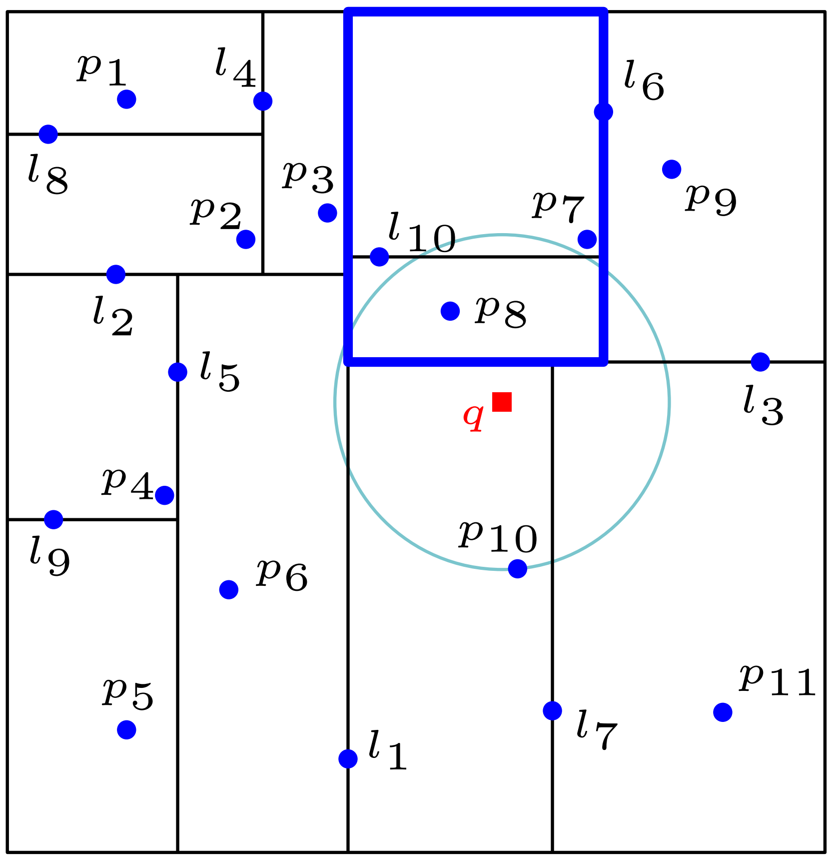


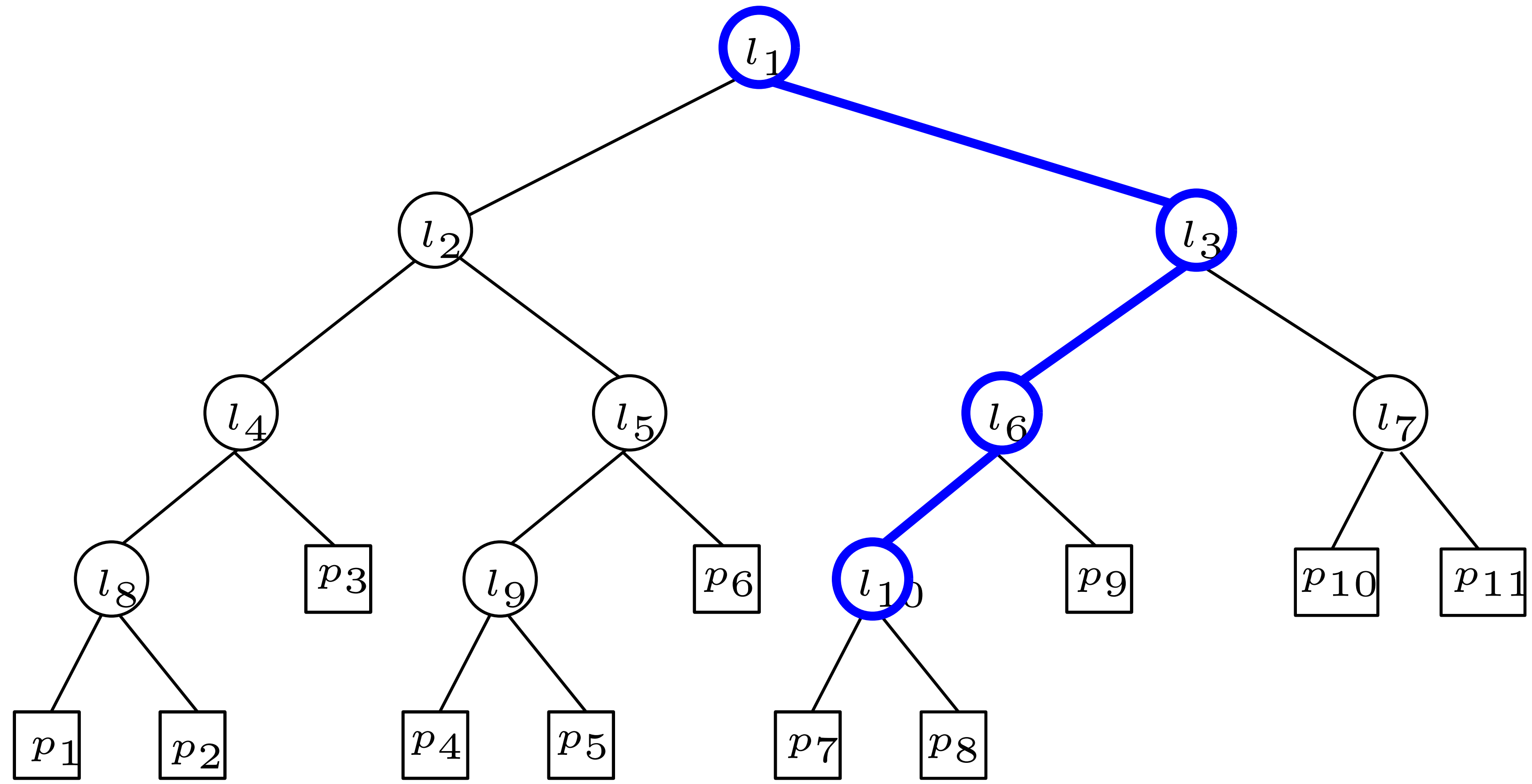
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



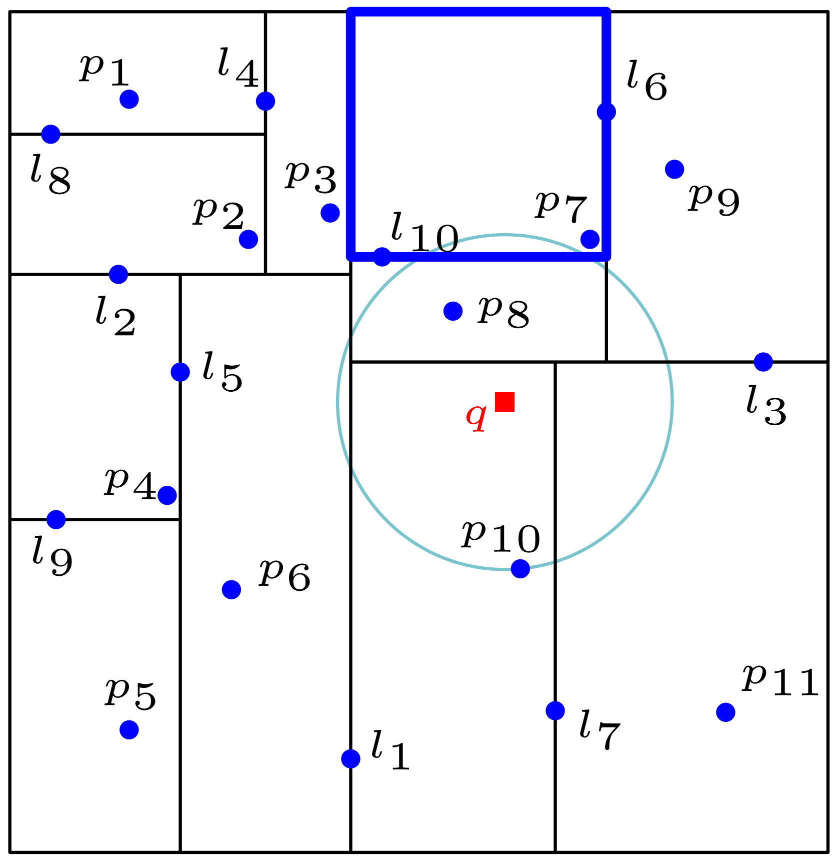


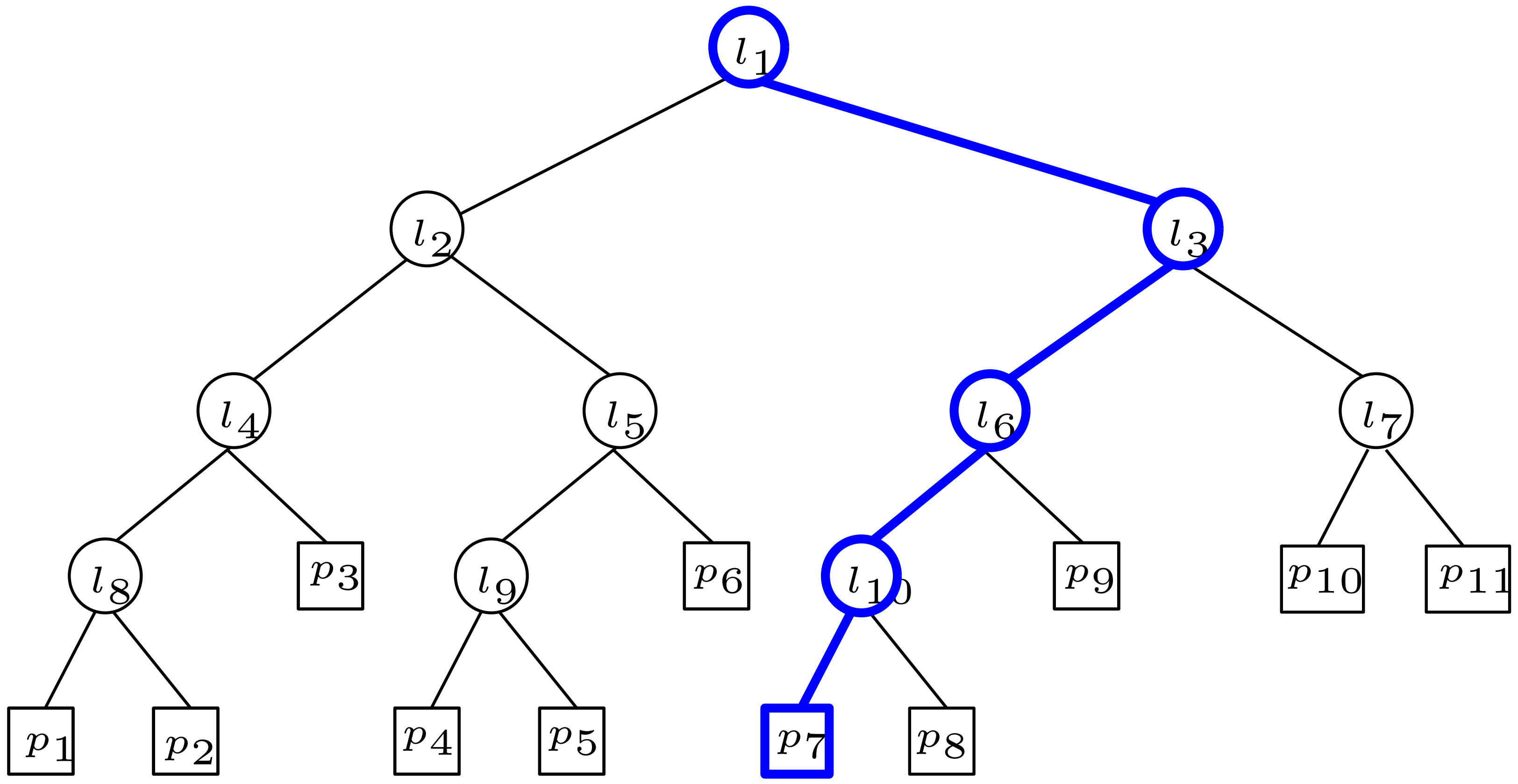
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



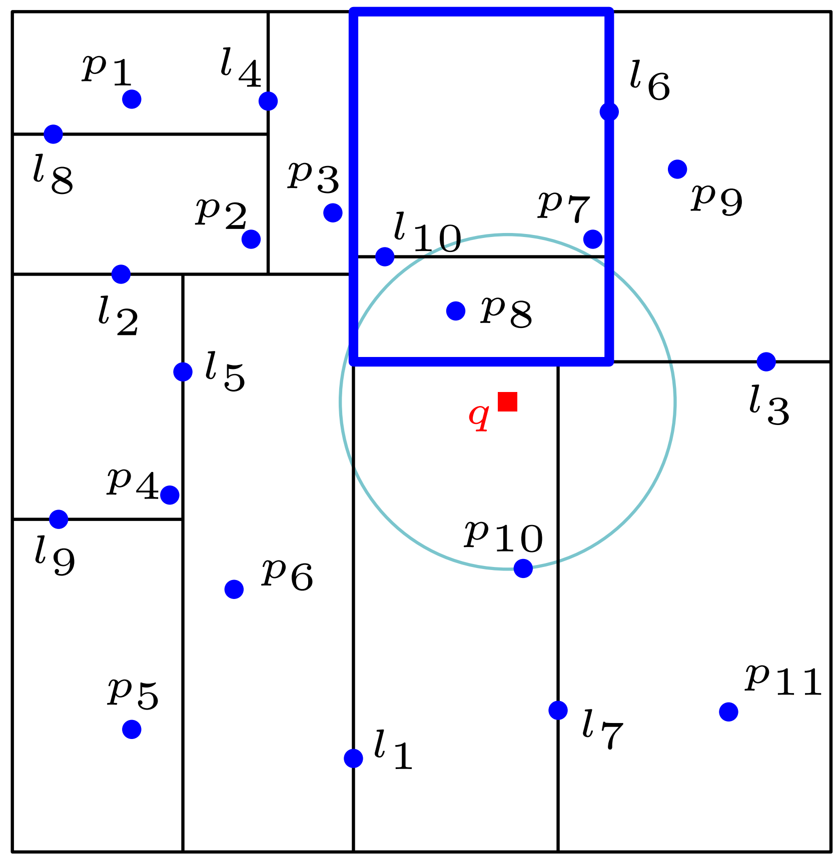


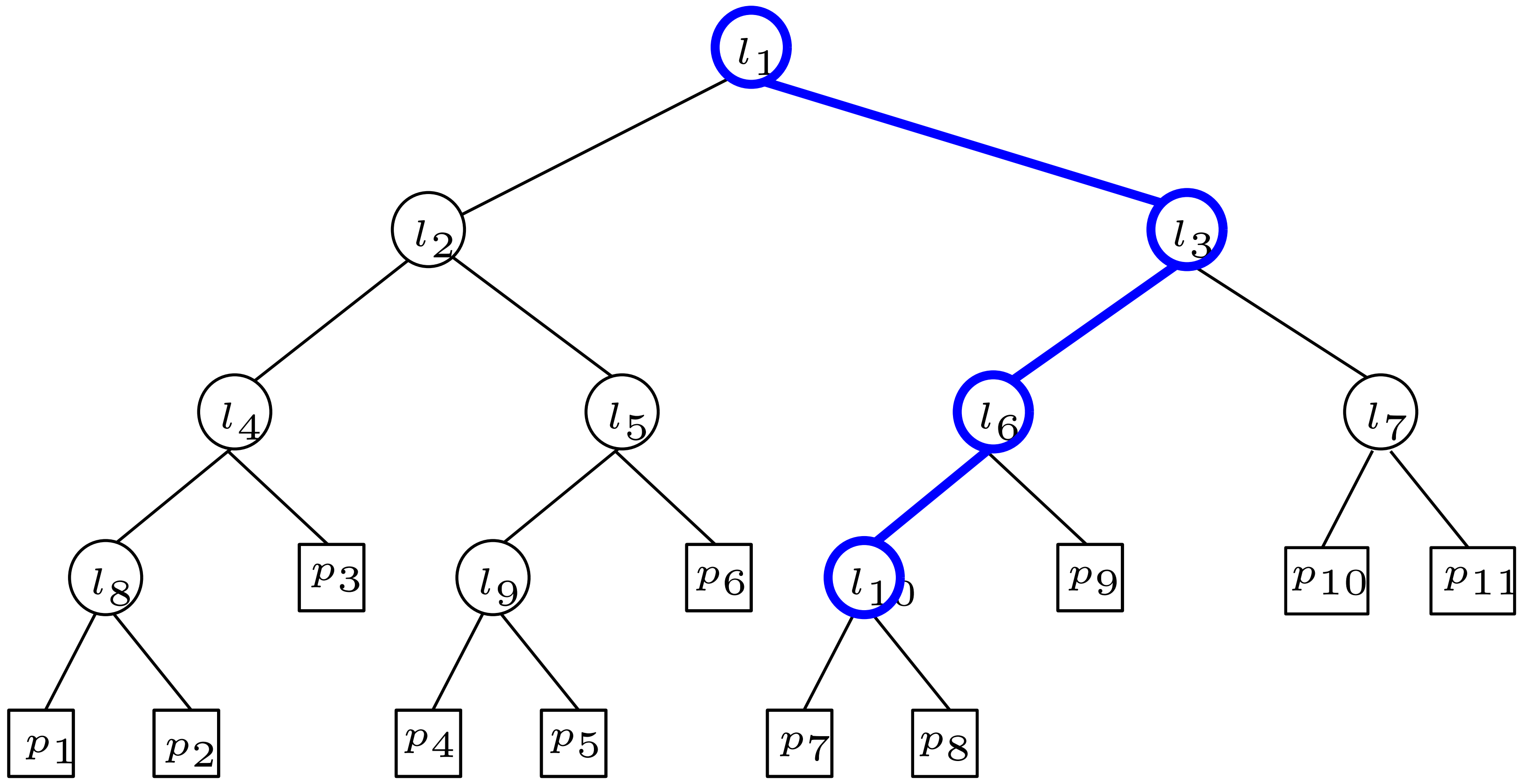
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



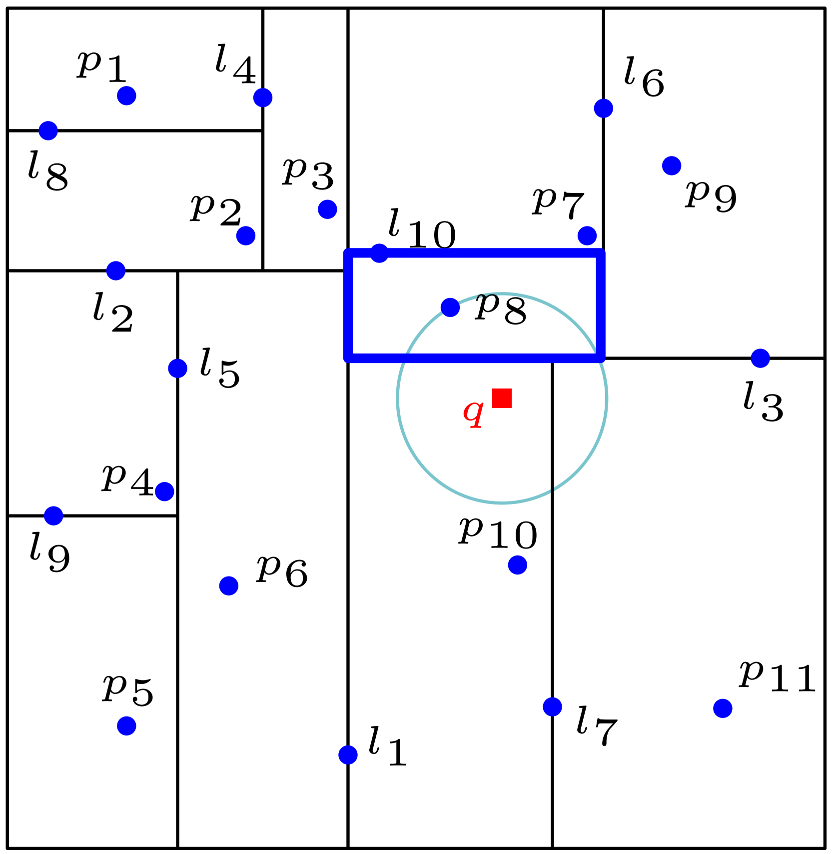


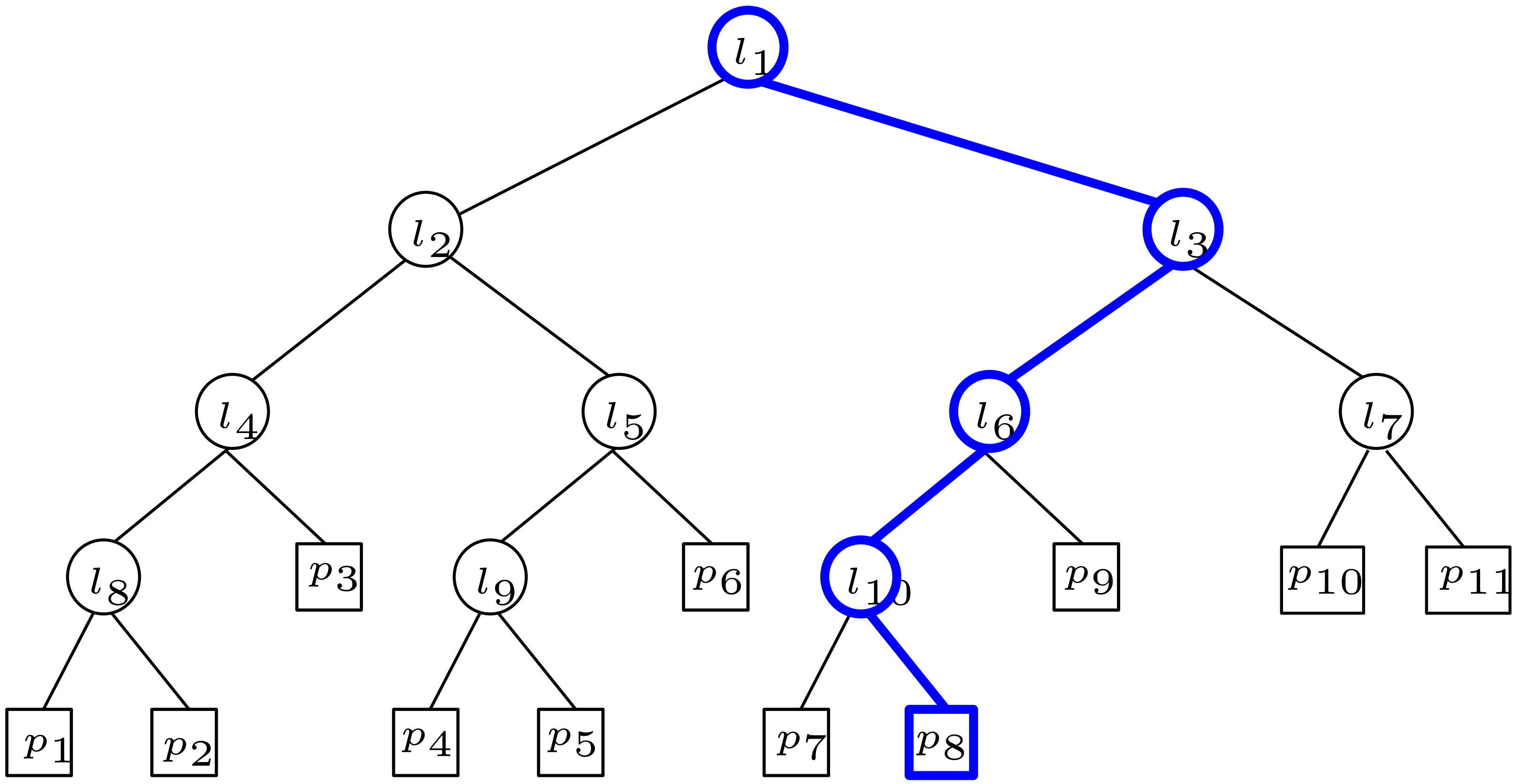
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



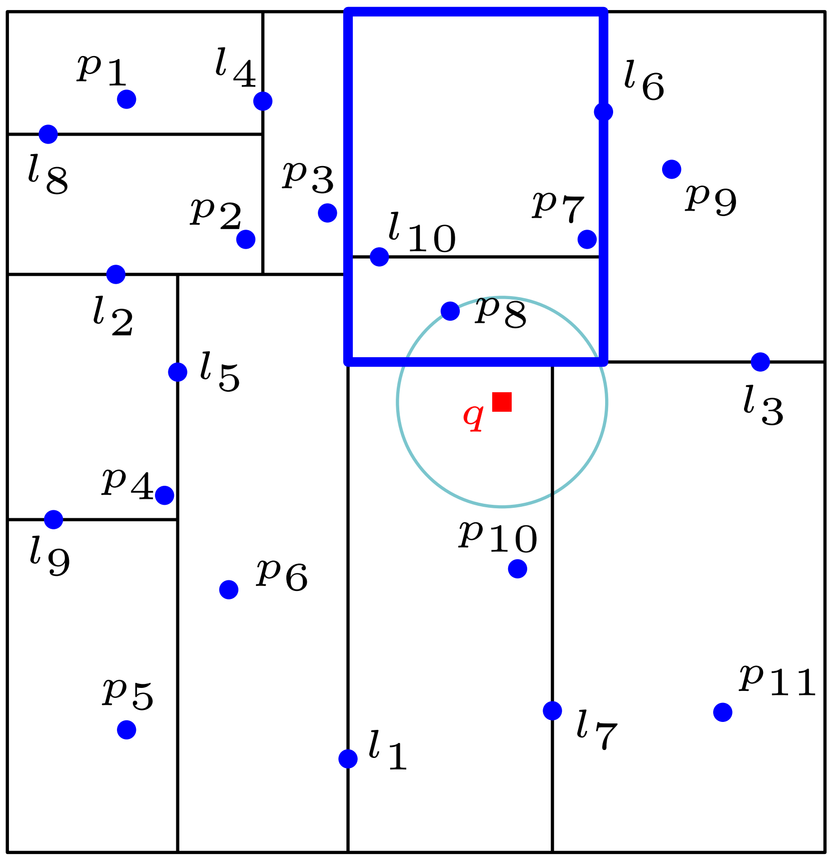


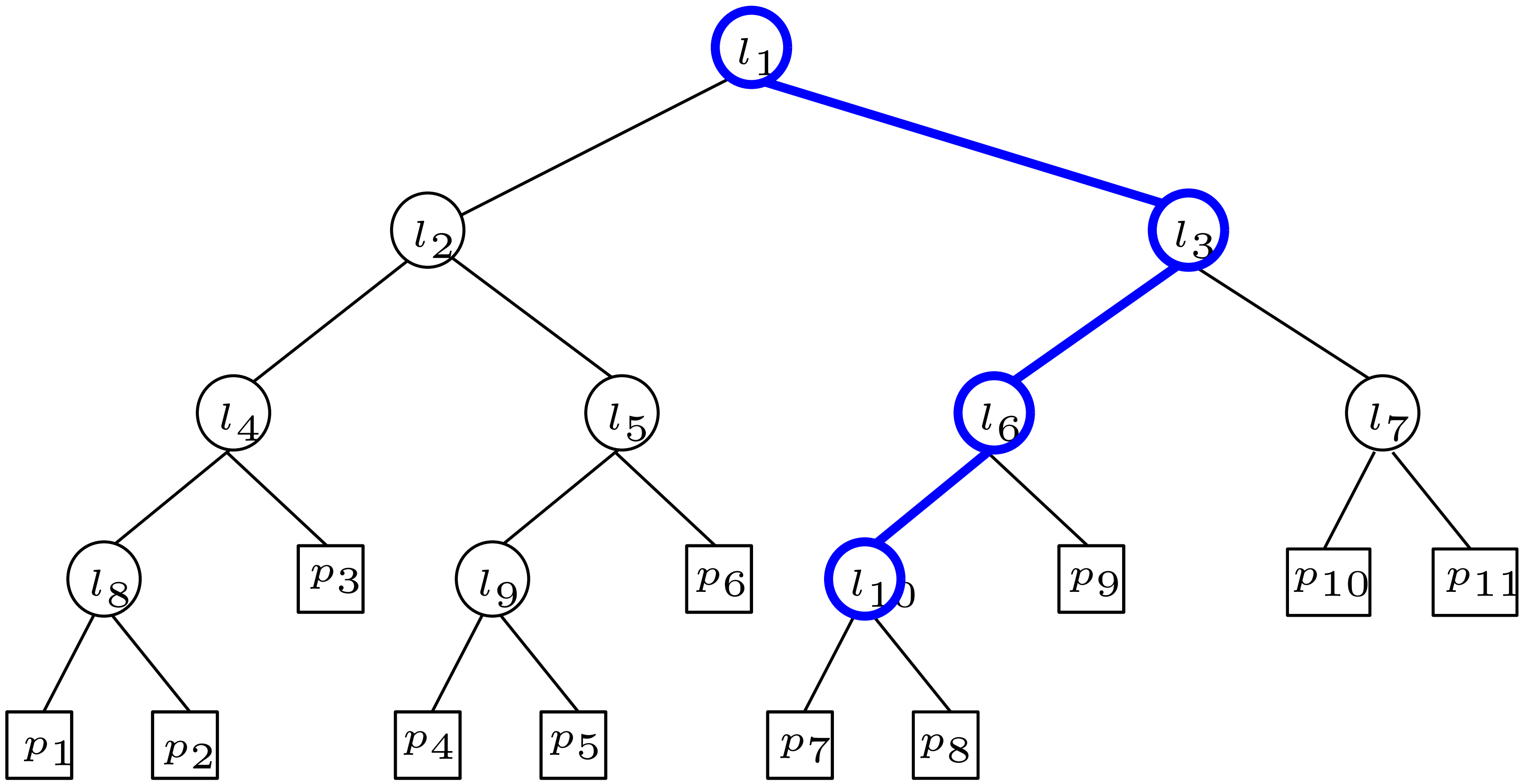
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



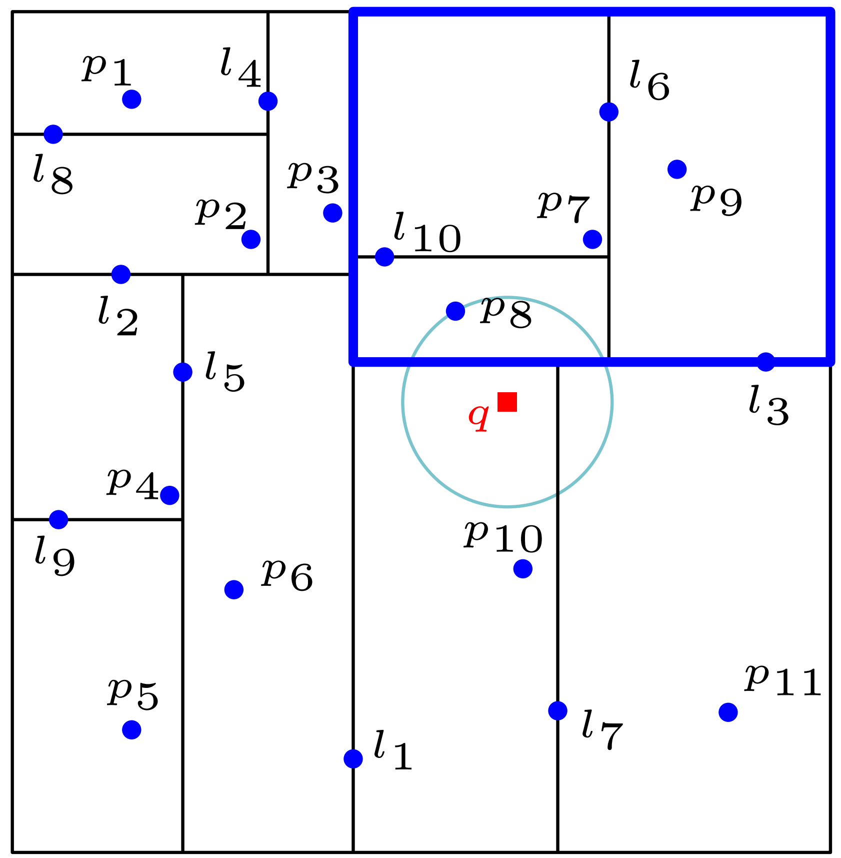


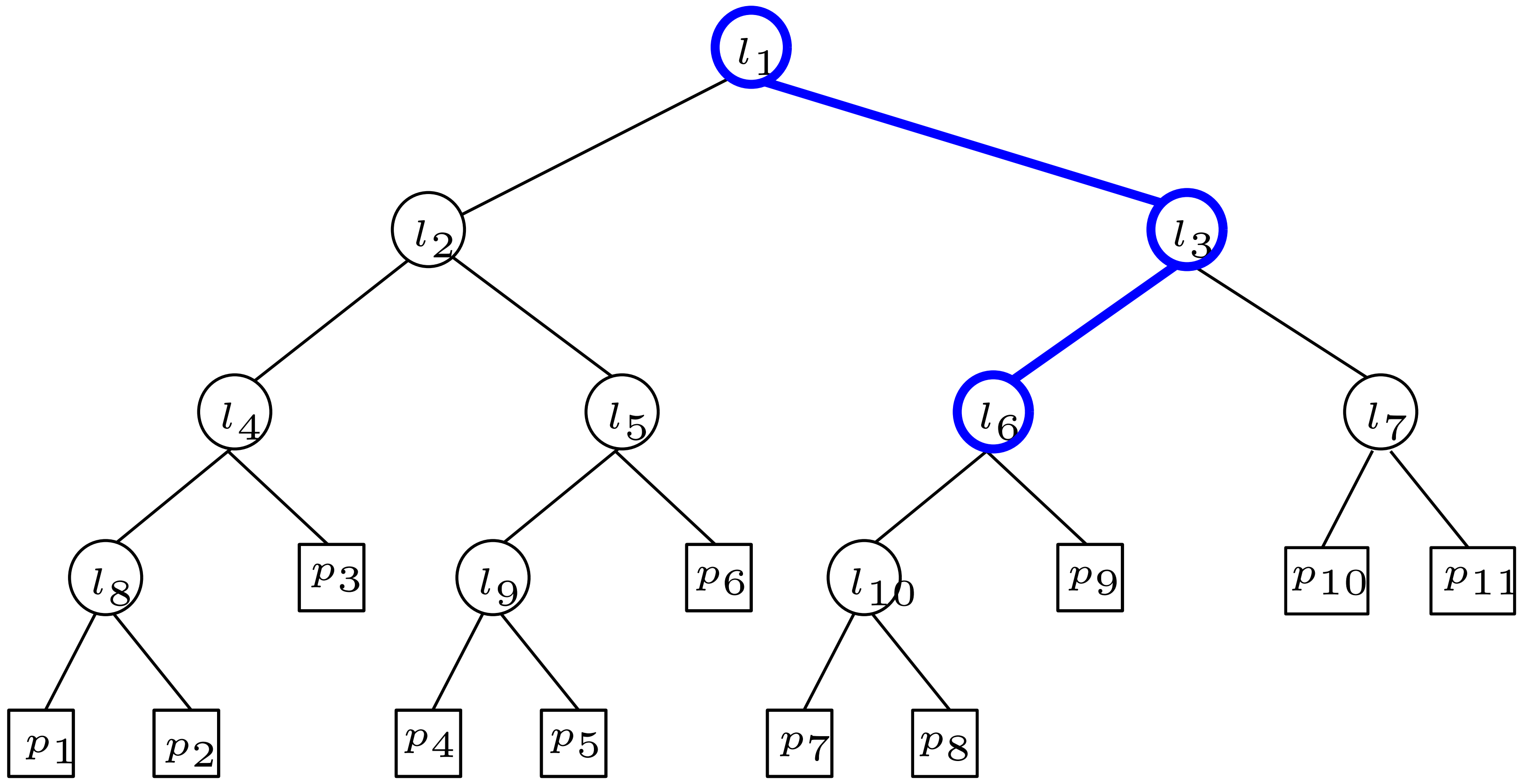
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



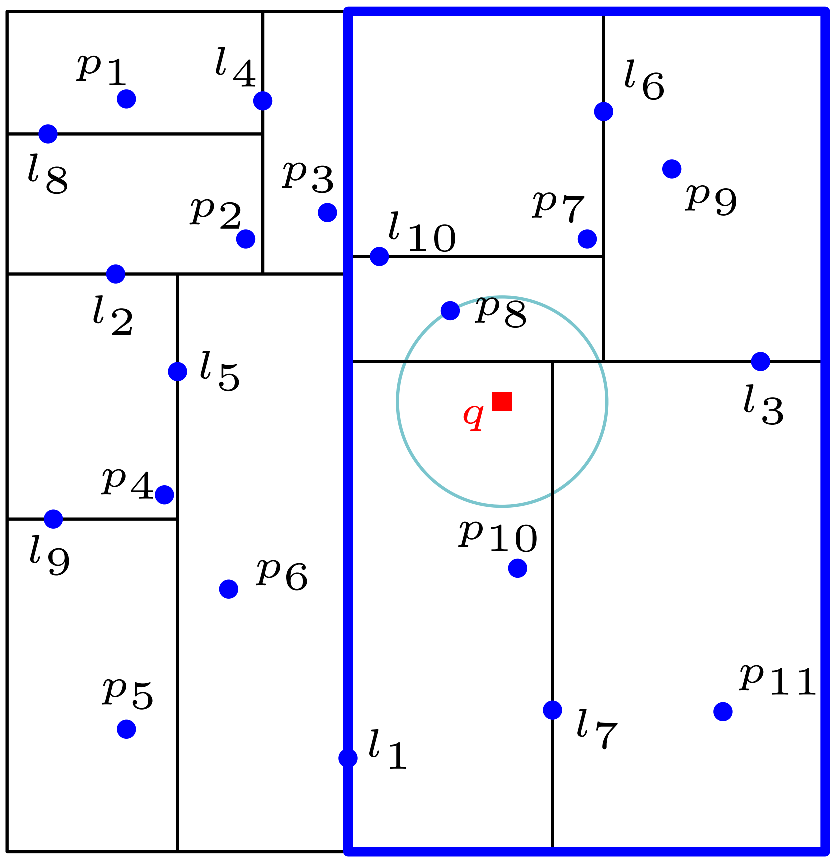


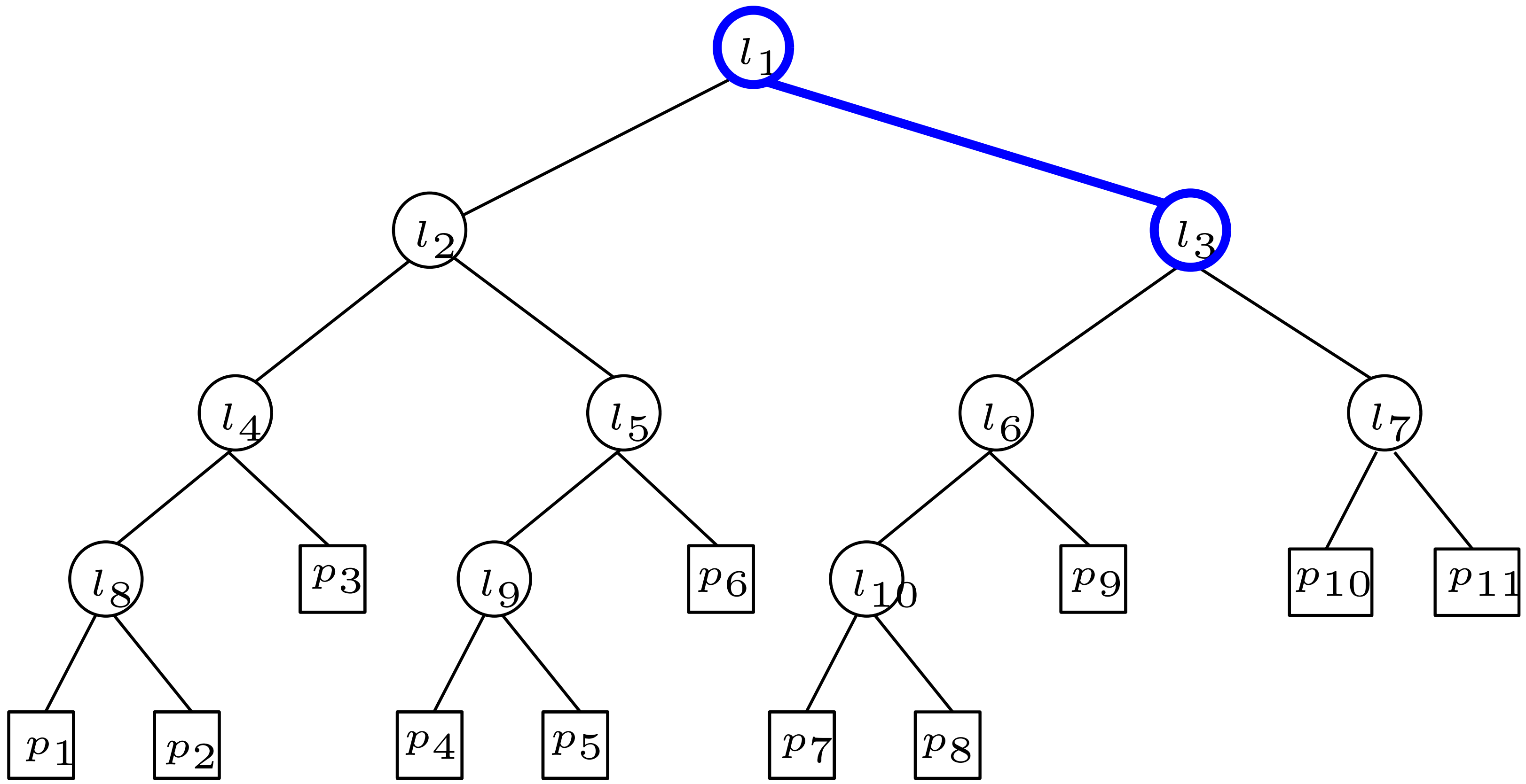
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



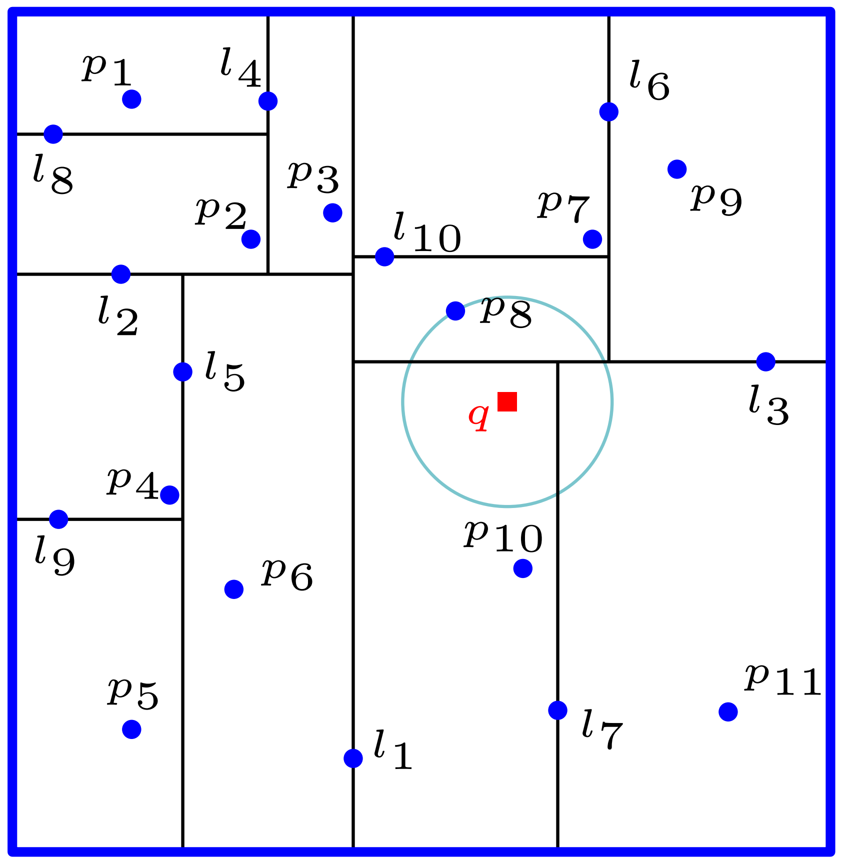


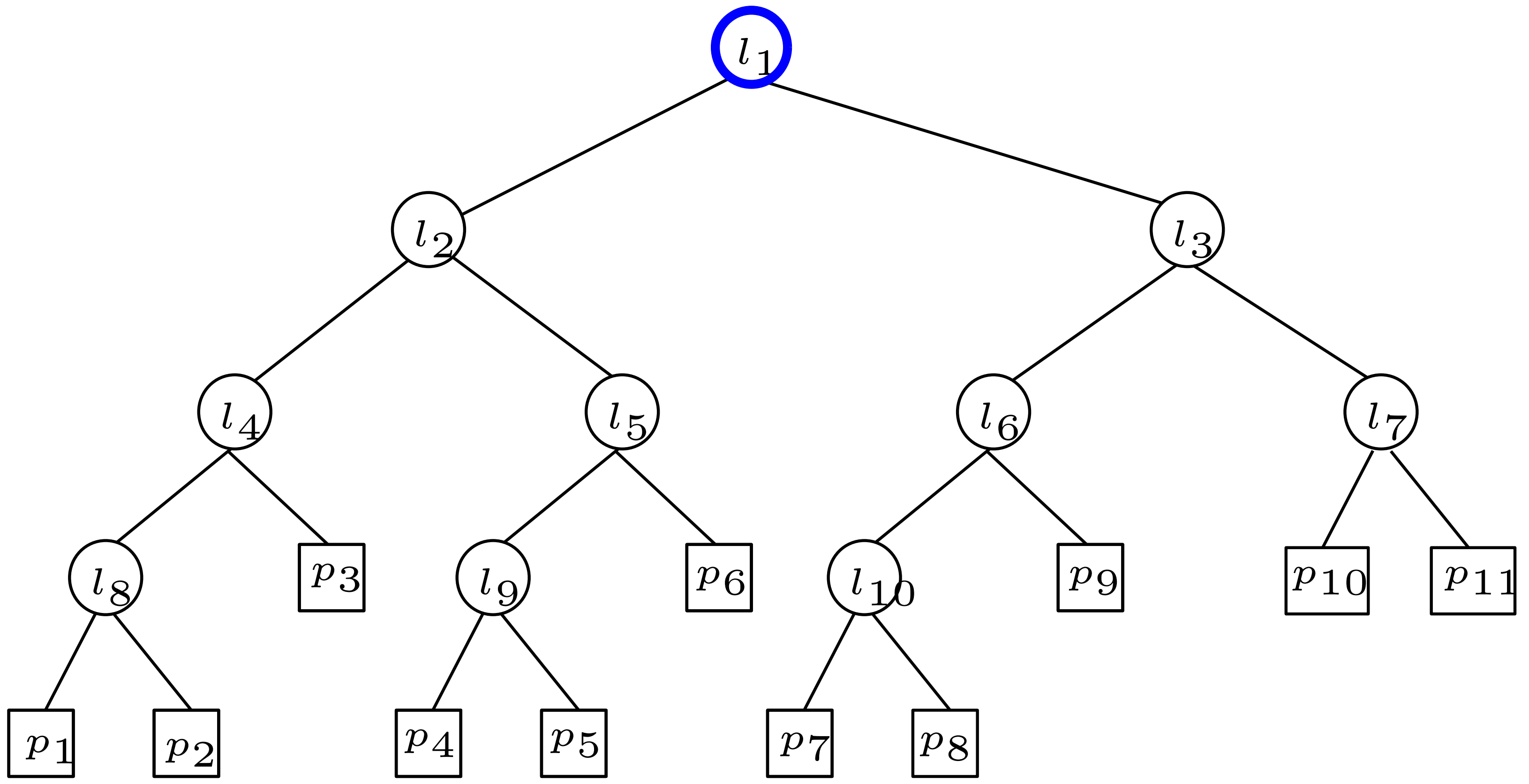
l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



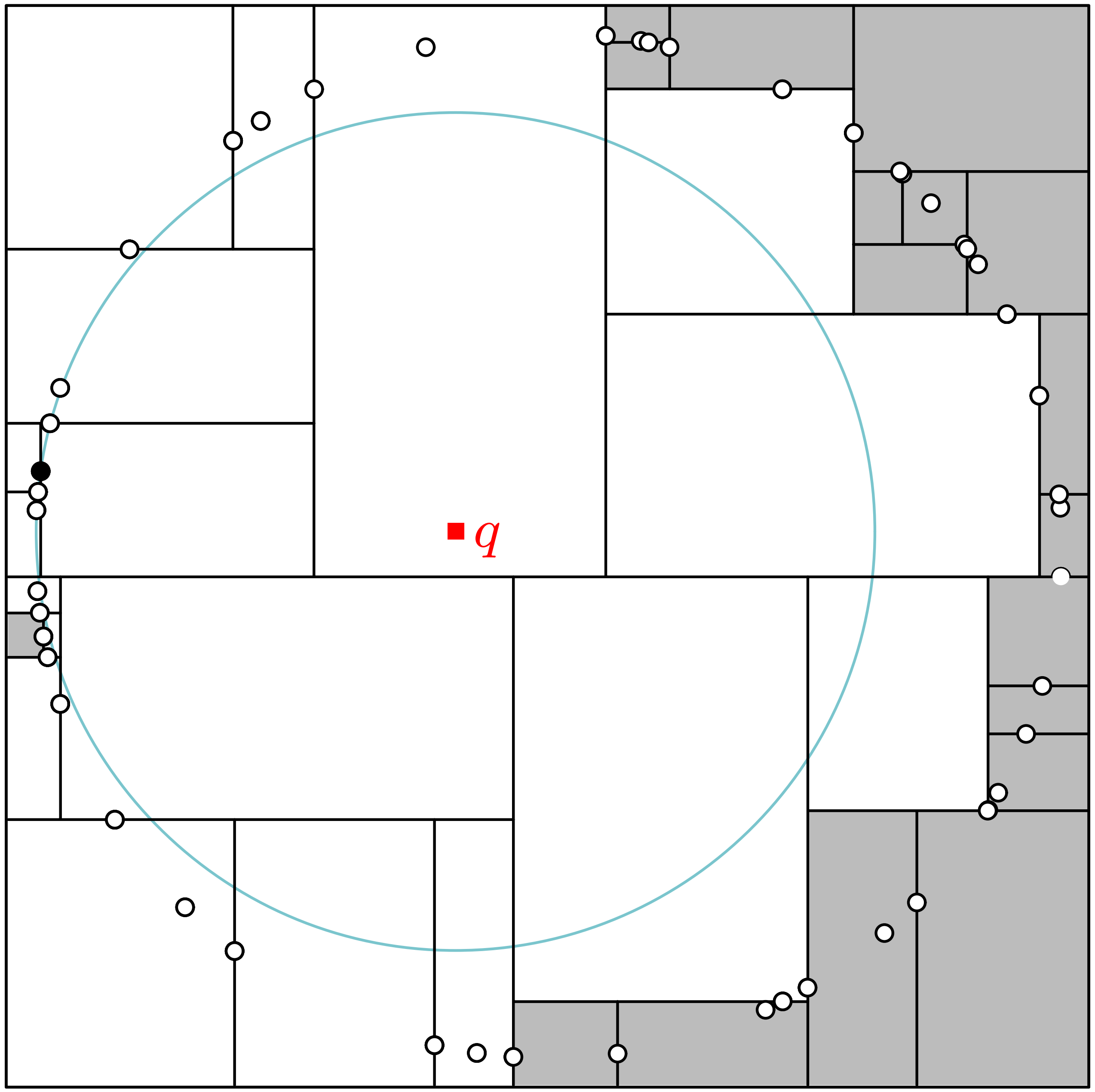


l_i: data at internal node

p_i: data at leaf node

(note: left-right labels are arbitrary)

Example



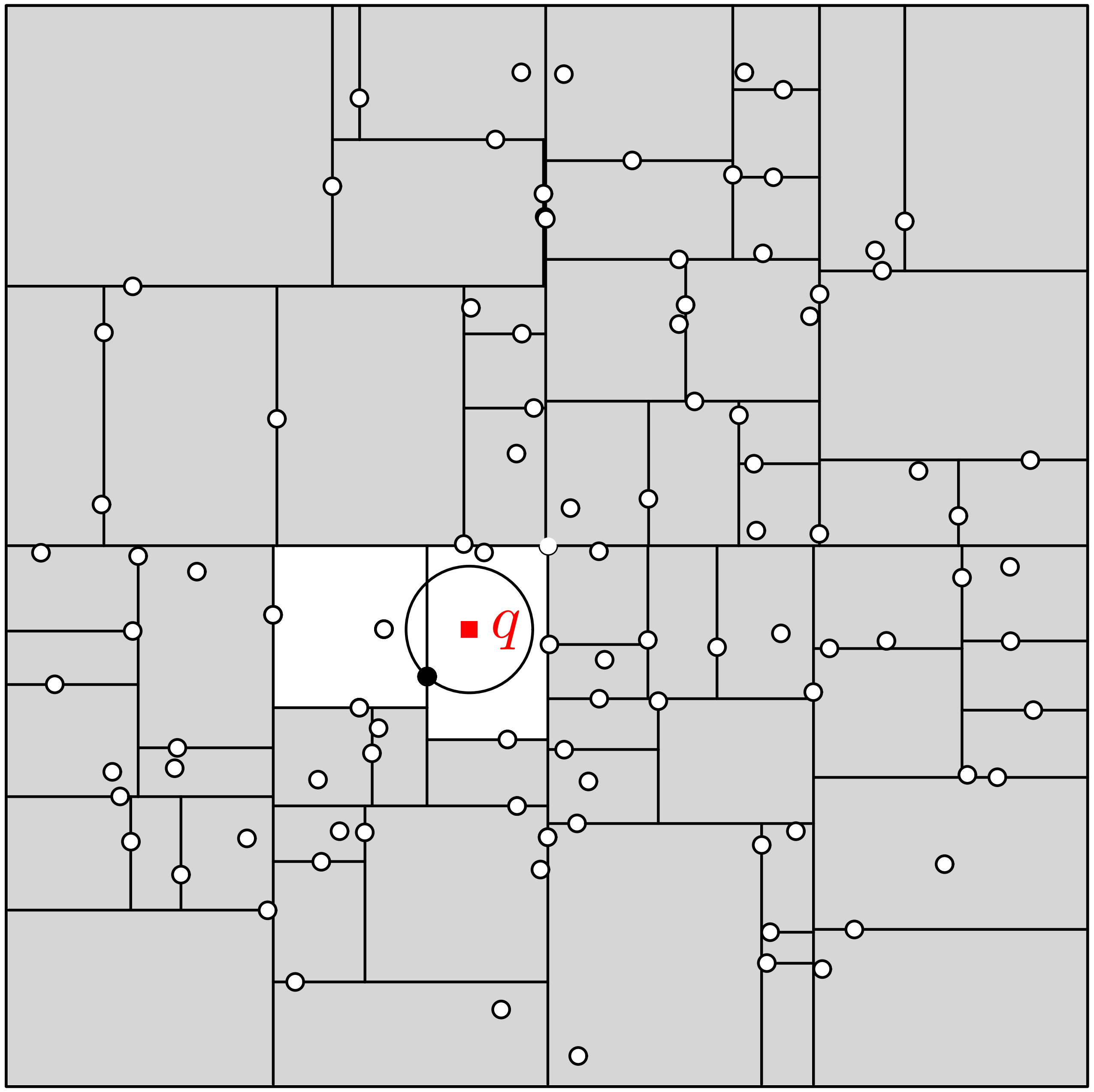
worst-case input (non-unif. distrib.):

long skinny cells

$$\Downarrow$$

query time $= \Omega(dn)$

Example



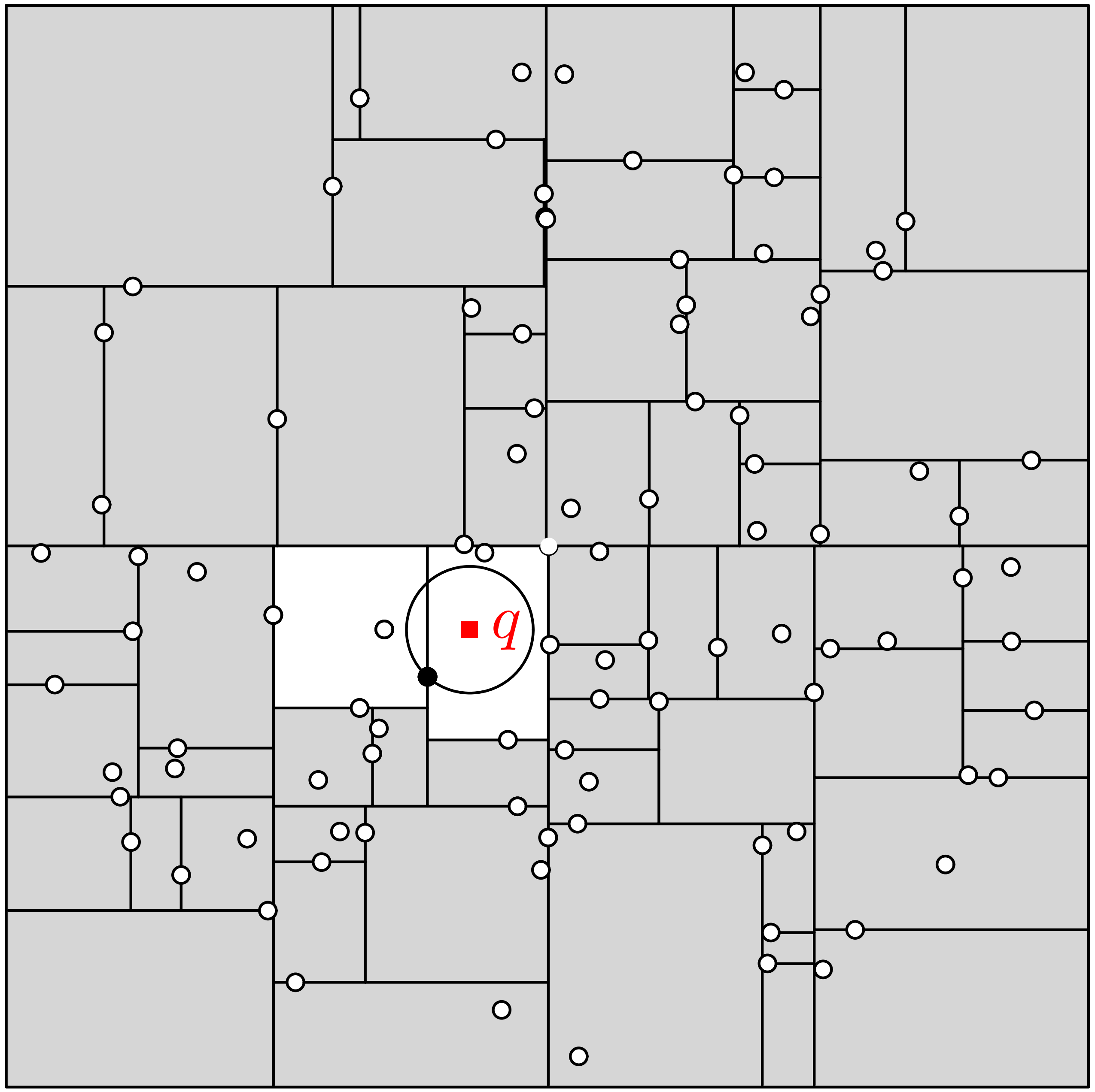
best-case input (unif. distrib.):

small fat cells

$$\Downarrow$$

query time $= O(c_d \log n)$

Example



best-case input (unif. distrib.):

small fat cells

$$\Downarrow$$

query time $= O(c_d \log n)$

Randomness should help!

(many variants: priority search, early backtracking, random cutting hyperplanes, etc.)

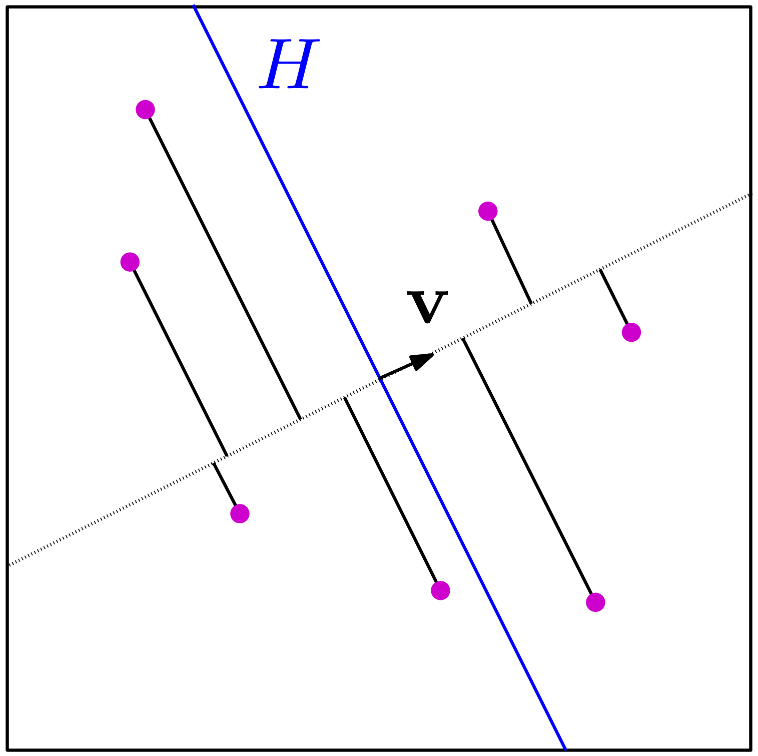
Random Projection/Partition Trees

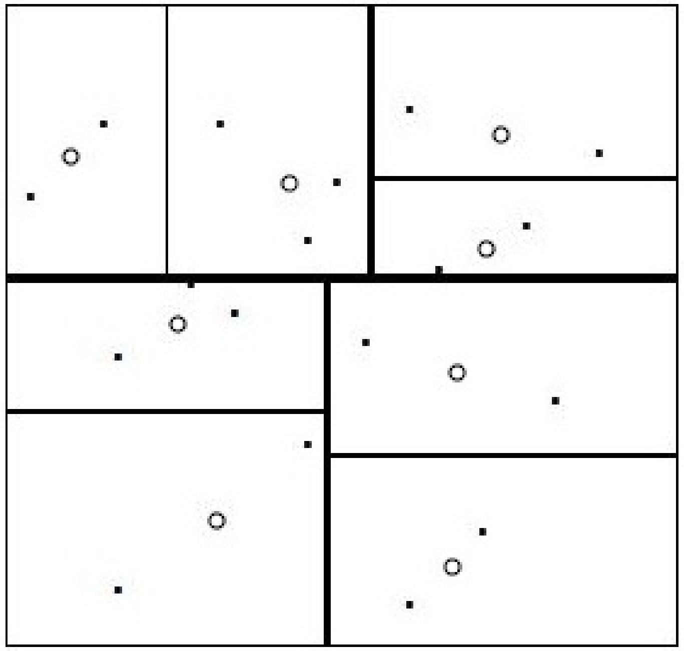
Exploiting randomness: RP-trees

Random Projection/Partition tree:

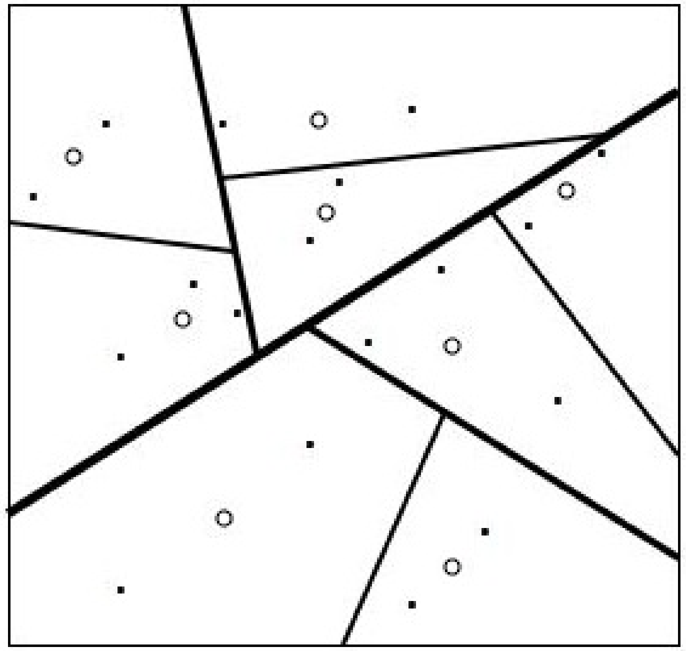
at each node (corresponding to some cell C):

- choose $\mathbf{v} \sim \mathrm{unif}(\mathbb{S}^{d-1})$ and $\beta \sim \mathrm{unif}([\frac{1}{4}, \frac{3}{4}])$
- let $H = \mathbf{v}^{\perp} + \mathrm{median}_{\beta}\{(P \cap C) \cdot \mathbf{v}\}\, \mathbf{v}$
- partition $P \cap C$ by H (as in kd-tree)





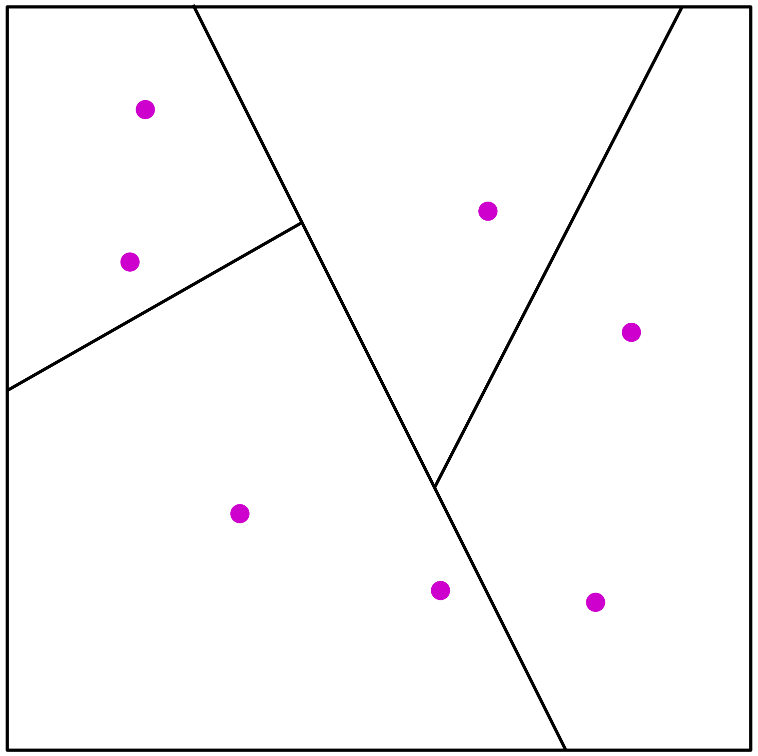
kd-tree



RP-tree

Exploiting randomness: RP-trees

Prop: [Dasgupta, Freund'08]
There is a constant $c > 0$ such that, for any cell C in a RP-tree built on $P \in \mathbb{R}^d$, with probability at least $1/2$ (over the choice of $\mathbf{v}, \beta$) all the cells lying at least $c\,k\log k$ levels below C in the tree have at most half the radius of C, where $k = \dim_2(P \cap C)$.



doubling dimension of $S \subseteq \mathbb{R}^d$: smallest $k \in \mathbb{N}$ such that, for every Euclidean ball B, $B \cap S$ can be covered by 2^k Euclidean balls of half radius.

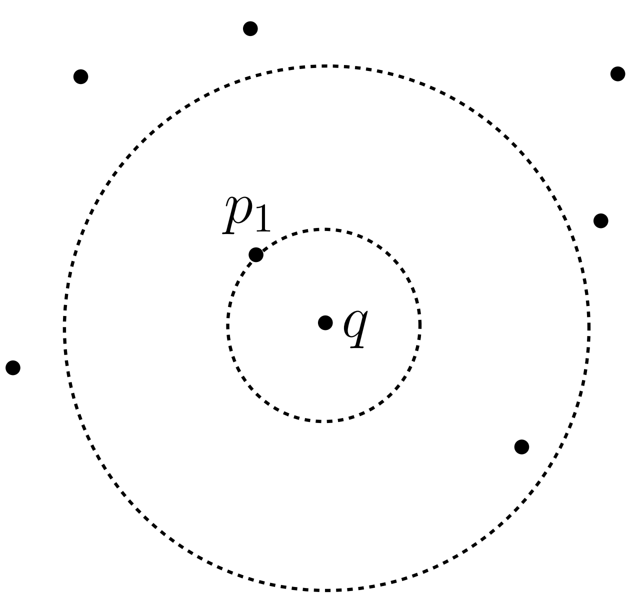
radius of $S \subseteq \mathbb{R}^d$: smallest $r > 0$ such that $\exists x \in C$ with $B(x, r) \supseteq S$.

Exploiting randomness: RP-trees

Thm: [Dasgupta, Sinha'13]
Let $\ell = \log(n/n_0)$ and $\bar{\beta} = \frac{3}{4}$.

$$\mathbb{P}_{\mathbf{v},\beta}\left[\text{defeatist search does not return } \mathrm{NN}_P(q)\right] \le \sum_{i=0}^{\ell} \Phi_{\bar{\beta}^i n} \log \frac{2e}{\Phi_{\bar{\beta}^i n}}$$

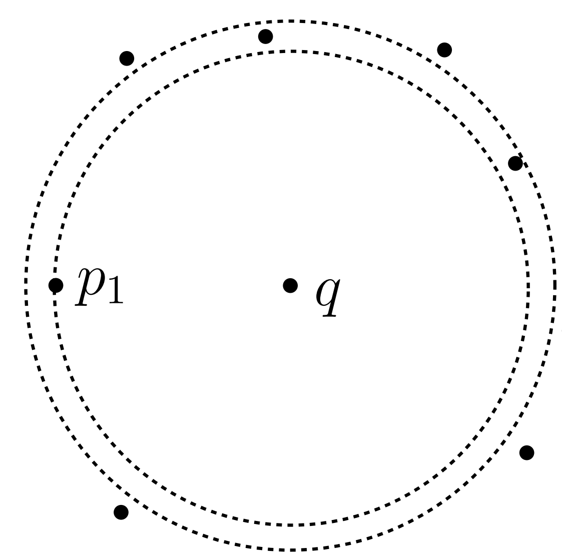
$$\Phi_m := \frac{1}{m} \sum_{i=2}^{m} \frac{\|q - p_1\|}{\|q - p_i\|},$$ where the p_i are ordered by increasing distance to q



Extreme cases:

$\Phi \approx 0$: p_1 isolated, easy to find

$\Phi \approx 1$: p_1 equidistant, hard to find



Exploiting randomness: RP-trees

Thm: [Dasgupta, Sinha'13]
Suppose $p_1, \cdots, p_n \overset{\text{iid}}{\sim} \mu$ continuous probability measure in $\mathbb{R}^d$ with doubling dimension $k \geq 2$. Then $\exists c_0 > 0$ s.t. for any $q \in \mathbb{R}^d$ and $\delta < 1/e$, with proba. $\geq 1 - 3\delta$ over the choice of the p_i's:

$$\mathbb{P}_{\mathbf{v},\beta}\left[\text{defeatist search does not return } \mathrm{NN}_P(q)\right] \leq c_0(k + \ln n_0)\left(\frac{8\,\ln 1/\delta}{n_0}\right)^{1/k}$$

doubling dimension of μ: smallest $k \in \mathbb{N}$ such that, for every $x \in \mathbb{R}^d$ and every $r > 0$: $\mu(B(x, 2r)) \leq 2^k\, \mu(B(x, r))$.

Exploiting randomness: RP-trees

Thm: [Dasgupta, Sinha'13]
Suppose $p_1, \cdots, p_n \overset{\text{iid}}{\sim} \mu$ continuous probability measure in $\mathbb{R}^d$ with doubling dimension $k \geq 2$. Then $\exists c_0 > 0$ s.t. for any $q \in \mathbb{R}^d$ and $\delta < 1/e$, with proba. $\geq 1 - 3\delta$ over the choice of the p_i's:

$$\mathbb{P}_{\mathbf{v},\beta}\left[\text{defeatist search does not return } \mathrm{NN}_P(q)\right] \leq c_0(k+\ln n_0)\left(\frac{8\,\ln 1/\delta}{n_0}\right)^{1/k}$$

doubling dimension of μ: smallest $k \in \mathbb{N}$ such that, for every $x \in \mathbb{R}^d$ and every $r > 0$: $\mu(B(x, 2r)) \leq 2^k\, \mu(B(x, r))$.

$\rightarrow$ take $n_0 \propto (k \ln k)^k \ln 1/\delta$ to make $\mathbb{P}_{\mathbf{v},\beta}\left[\cdots\right]$ an arbitrarily small constant

$\rightarrow$ query time: $O(d((k \ln k)^k + \log n)) \longleftarrow$ sensitive to intrinsic dim.

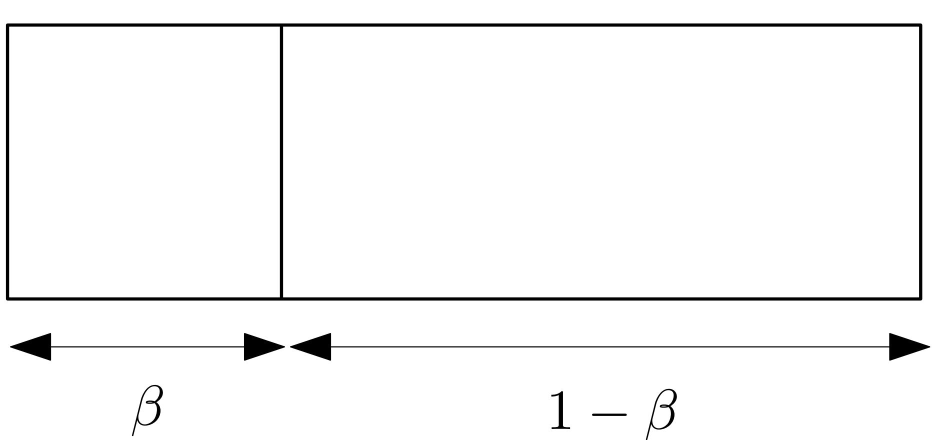
requires to know k

Exploiting randomness: RP-trees

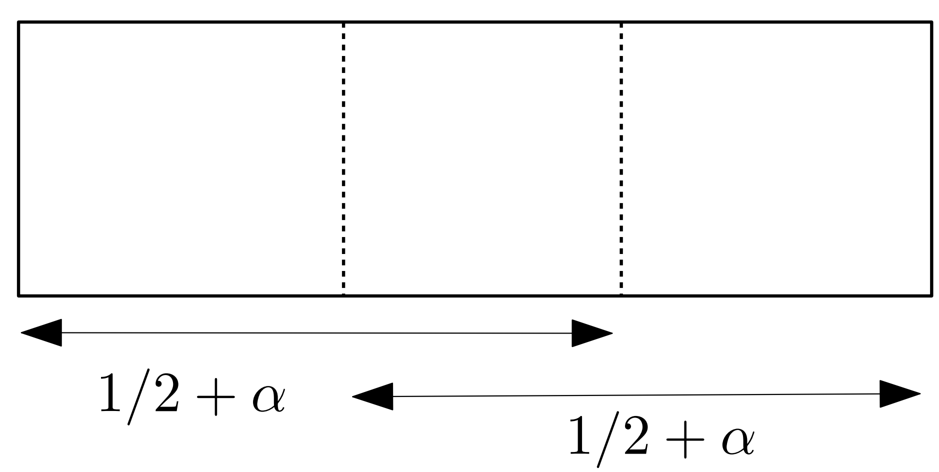
Thm: [Dasgupta, Sinha'13]
Suppose $p_1, \cdots, p_n \overset{\text{iid}}{\sim} \mu$ continuous probability measure in $\mathbb{R}^d$ with doubling dimension $k \geq 2$. Then $\exists c_0 > 0$ s.t. for any $q \in \mathbb{R}^d$ and $\delta < 1/e$, with proba. $\geq 1 - 3\delta$ over the choice of the p_i's:

$$\mathbb{P}_{\mathbf{v},\beta}\left[\text{defeatist search does not return } \mathrm{NN}_P(q)\right] \leq c_0(k + \ln n_0)\left(\frac{8 \ln 1/\delta}{n_0}\right)^{1/k}$$

Variant: **spill-trees** (overlapping splits)



(RP-tree)



(spill-tree)

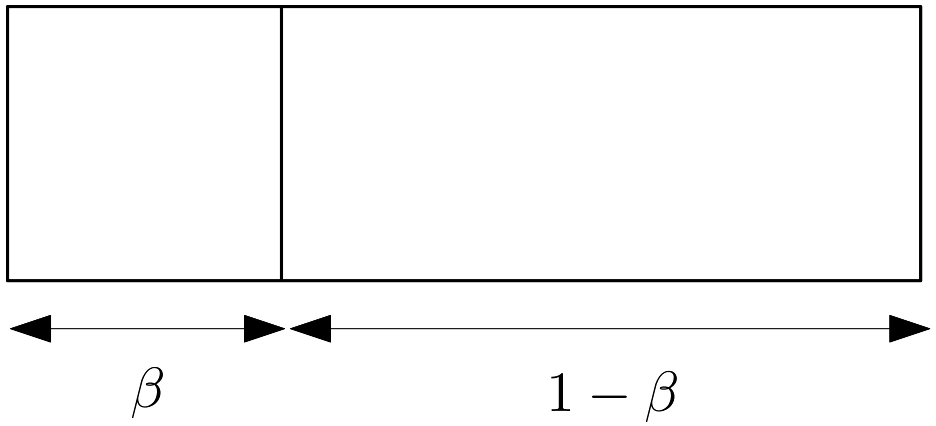
Exploiting randomness: RP-trees

Thm: [Dasgupta, Sinha'13]
Suppose $p_1, \cdots, p_n \overset{\text{iid}}{\sim} \mu$ continuous probability measure in $\mathbb{R}^d$ with doubling dimension $k \geq 2$. Then $\exists c_0 > 0$ s.t. for any $q \in \mathbb{R}^d$ and $\delta < 1/e$, with proba. $\geq 1 - 3\delta$ over the choice of the p_i's:

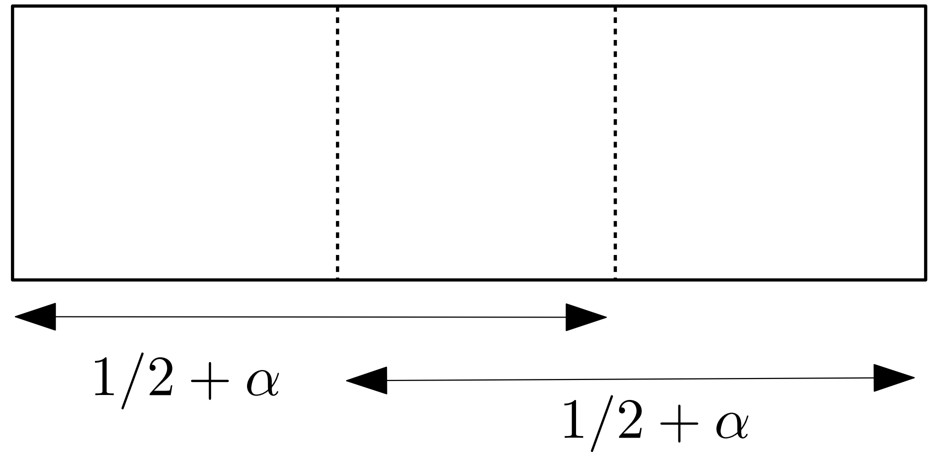
$$\mathbb{P}_{\mathbf{v},\beta}\left[\text{defeatist search does not return } \mathrm{NN}_P(q)\right] \leq c_0(k+\ln n_0)\left(\frac{8\,\ln 1/\delta}{n_0}\right)^{1/k}$$

Variant: **spill-trees** (overlapping splits)

similar behavior



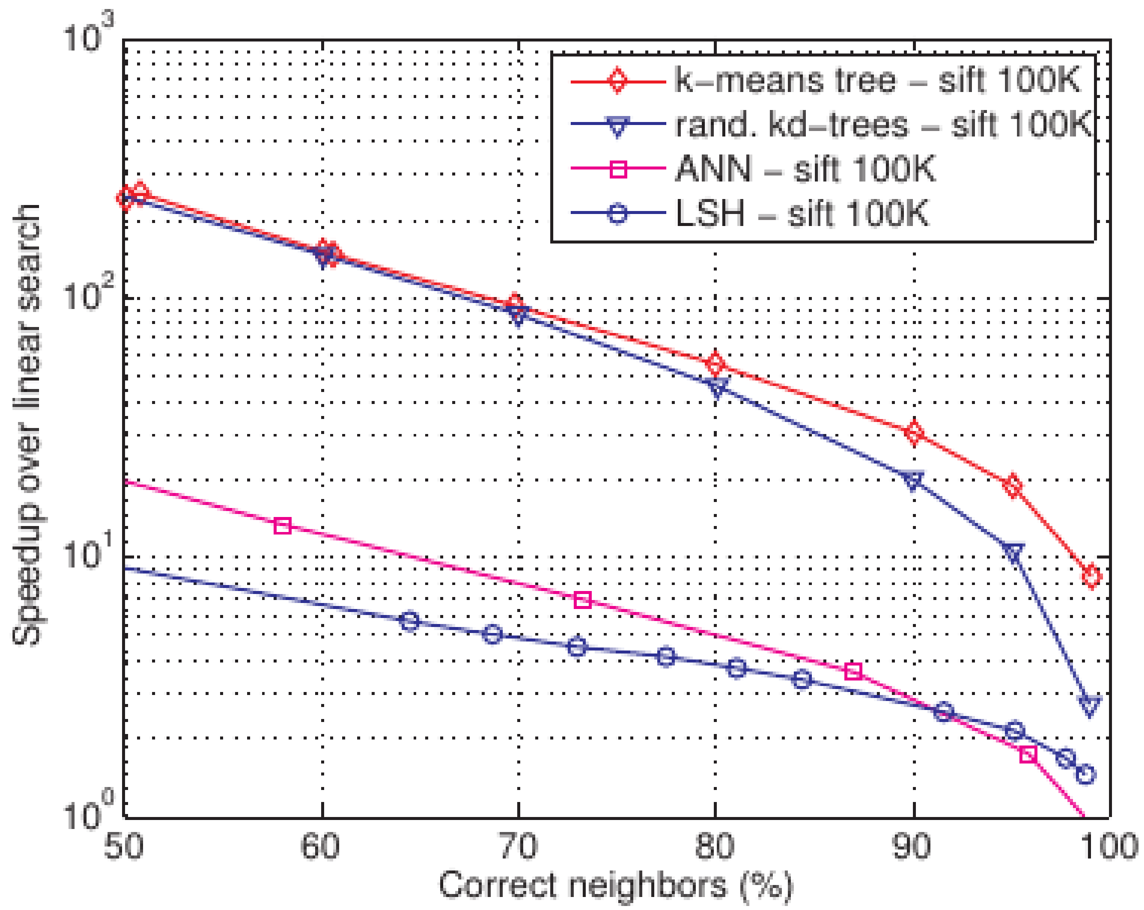
(RP-tree)



(spill-tree)

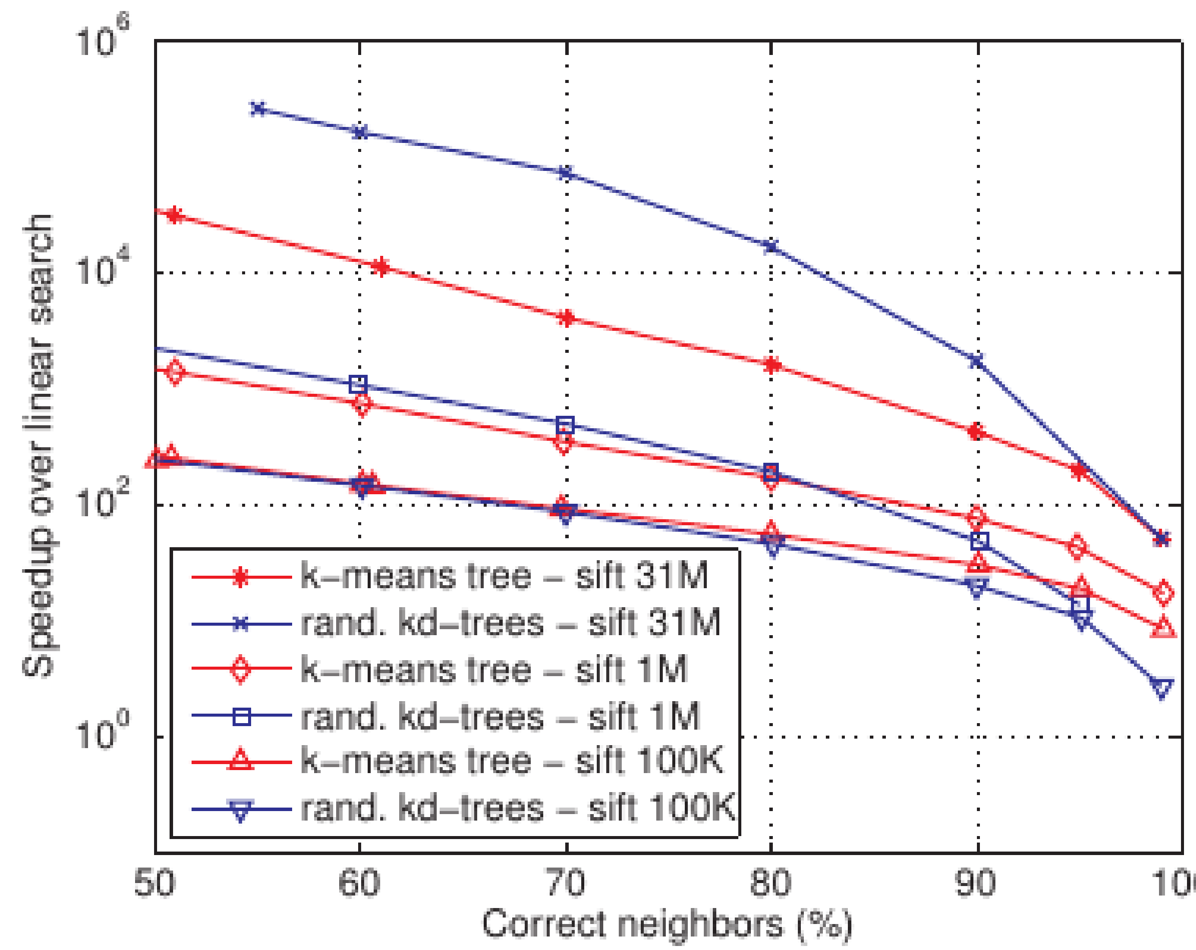
Benchmarking

contenders



(a)

effect of size on winners

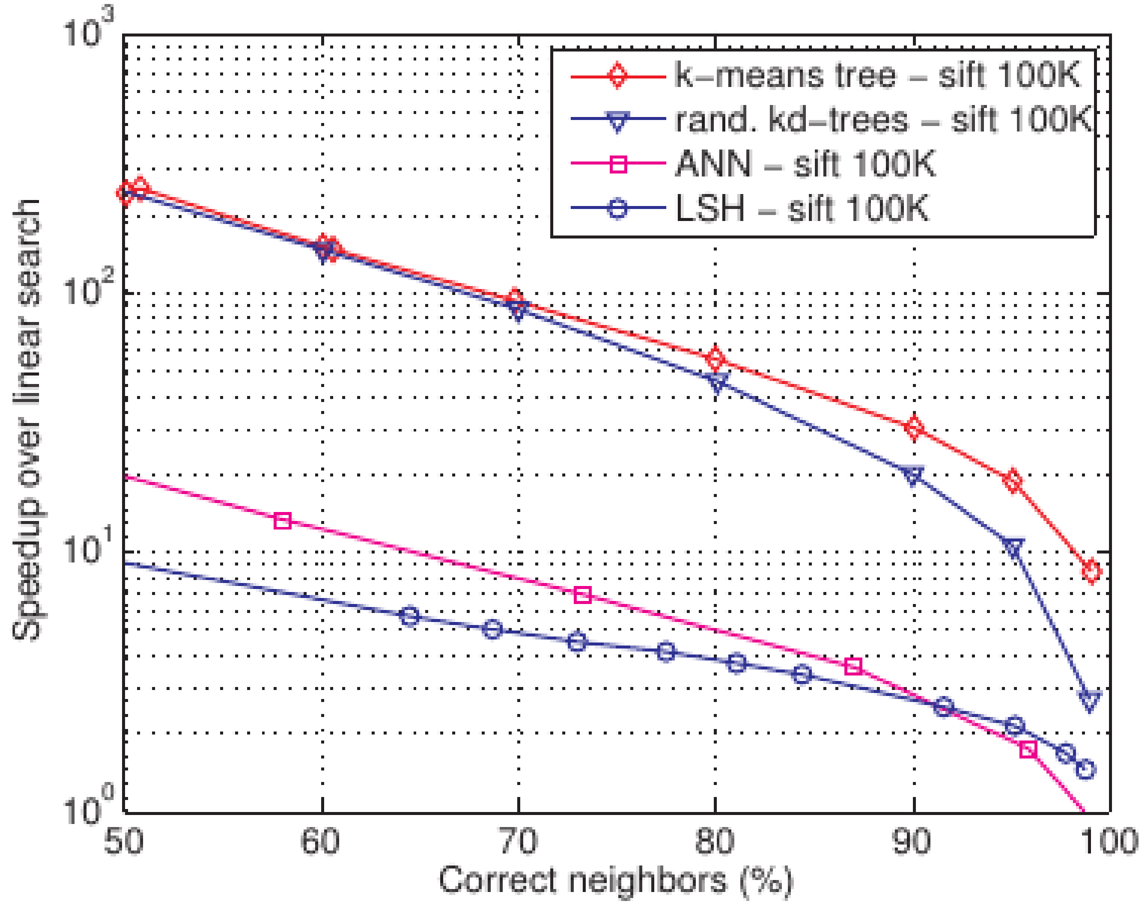


(b)

RP-trees vs. other methods on data sets of 100k, 1M and 31M features

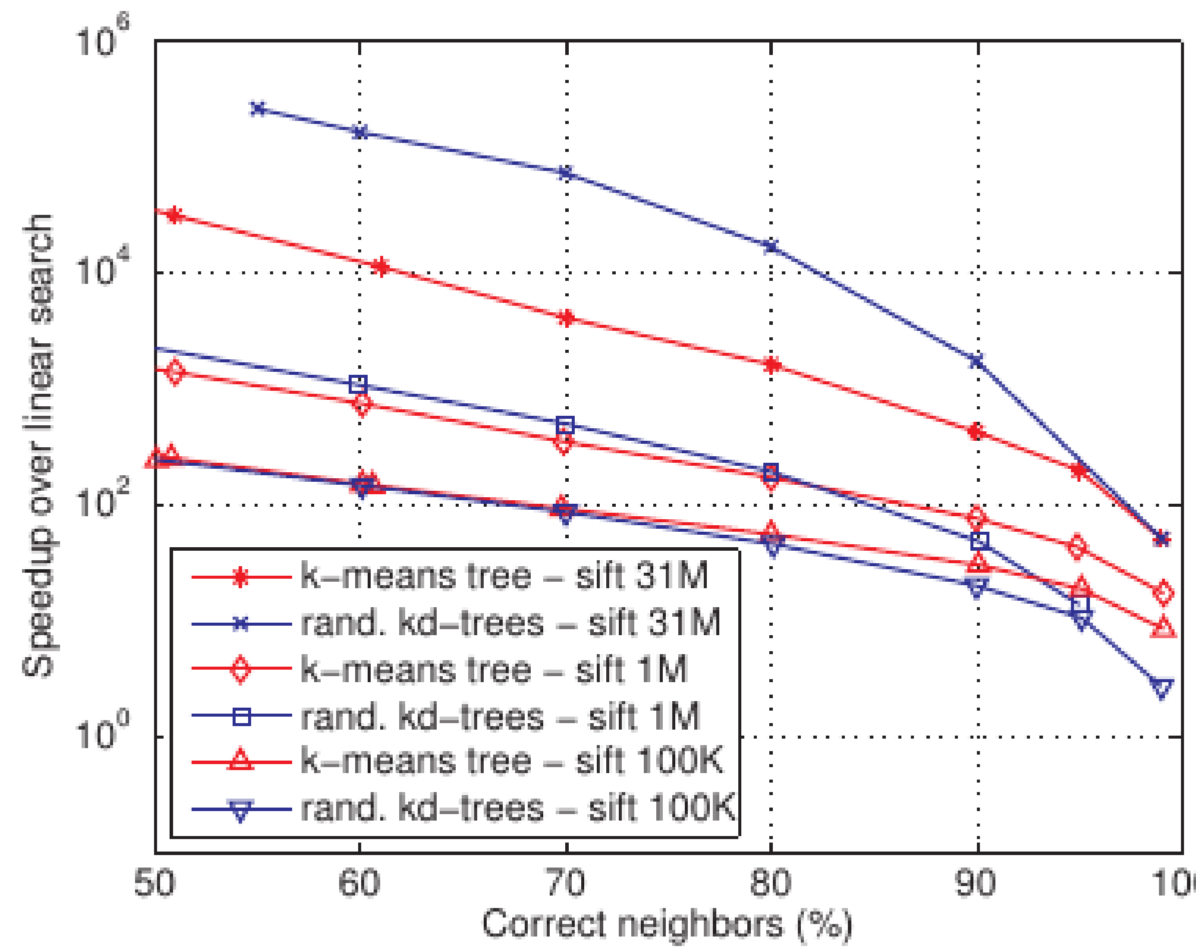
Benchmarking

contenders



(a)

effect of size on winners

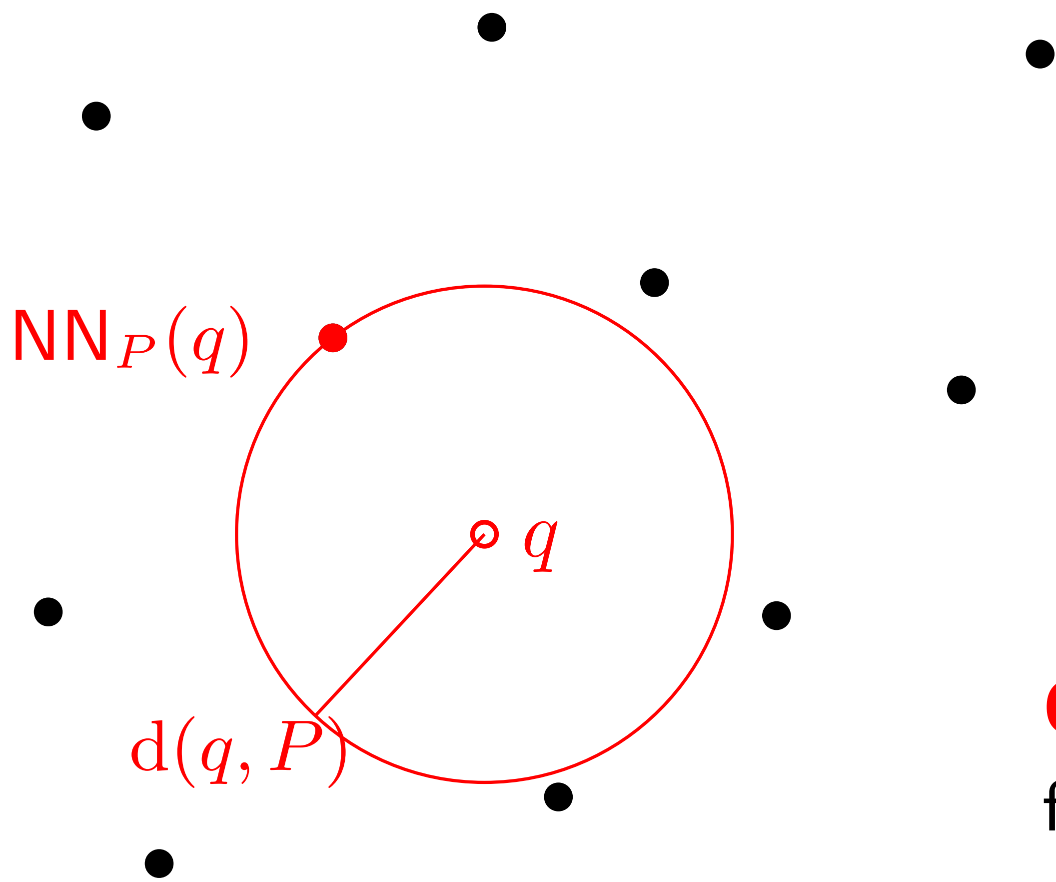


(b)

Random kd-trees (RP-trees, spill-trees) are fast, scalable and reliable on **data with (low-dimensional) intrinsic structure**

Locality-Sensitive Hashing

Back to the NN problem



pre-processing input: P

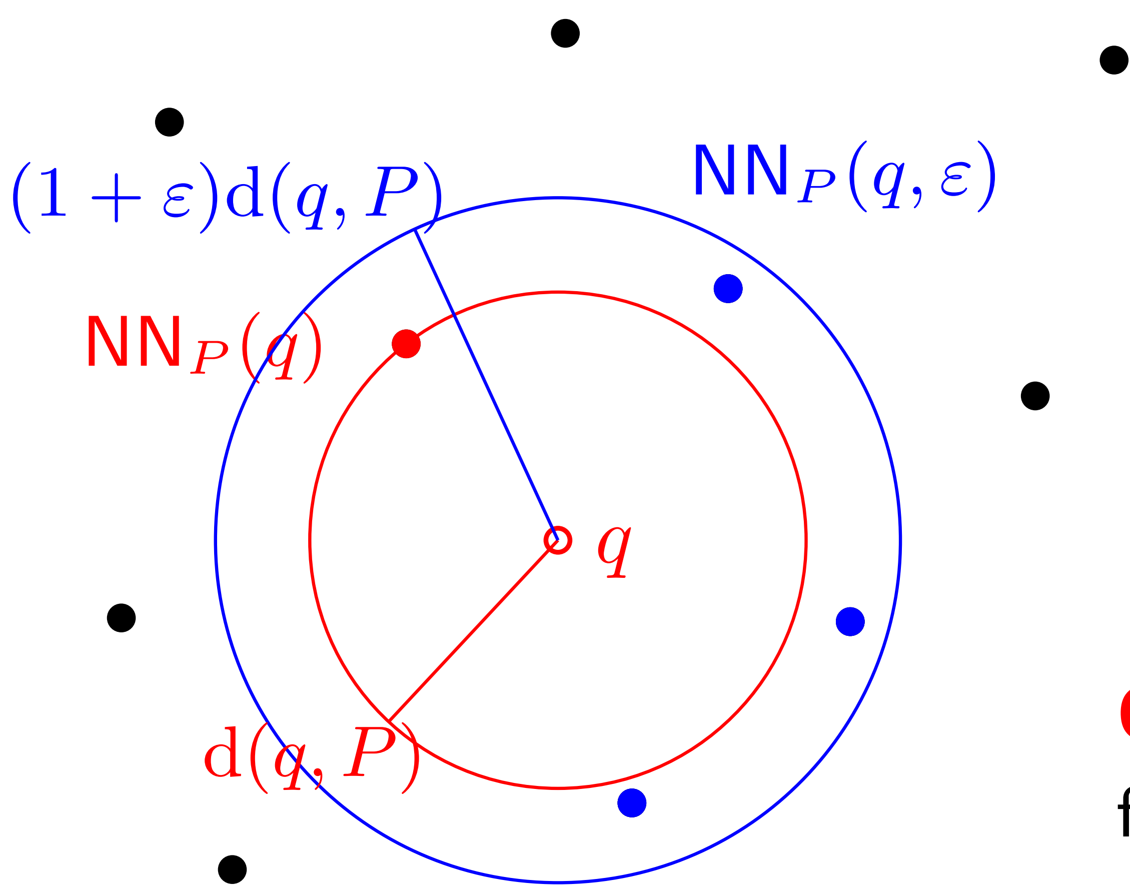
query input: q

goal: find $p \in \mathsf{NN}_P(q)$

Curse of Dimensionality: every DS for NN-search has either exponential size or exponential query time (in d) in the worst case.

$\rightarrow$ holds in theory and in practice for exact NN queries [Weber et al. '98]

Back to the ε-NN problem



pre-processing input: P, ε

query input: q

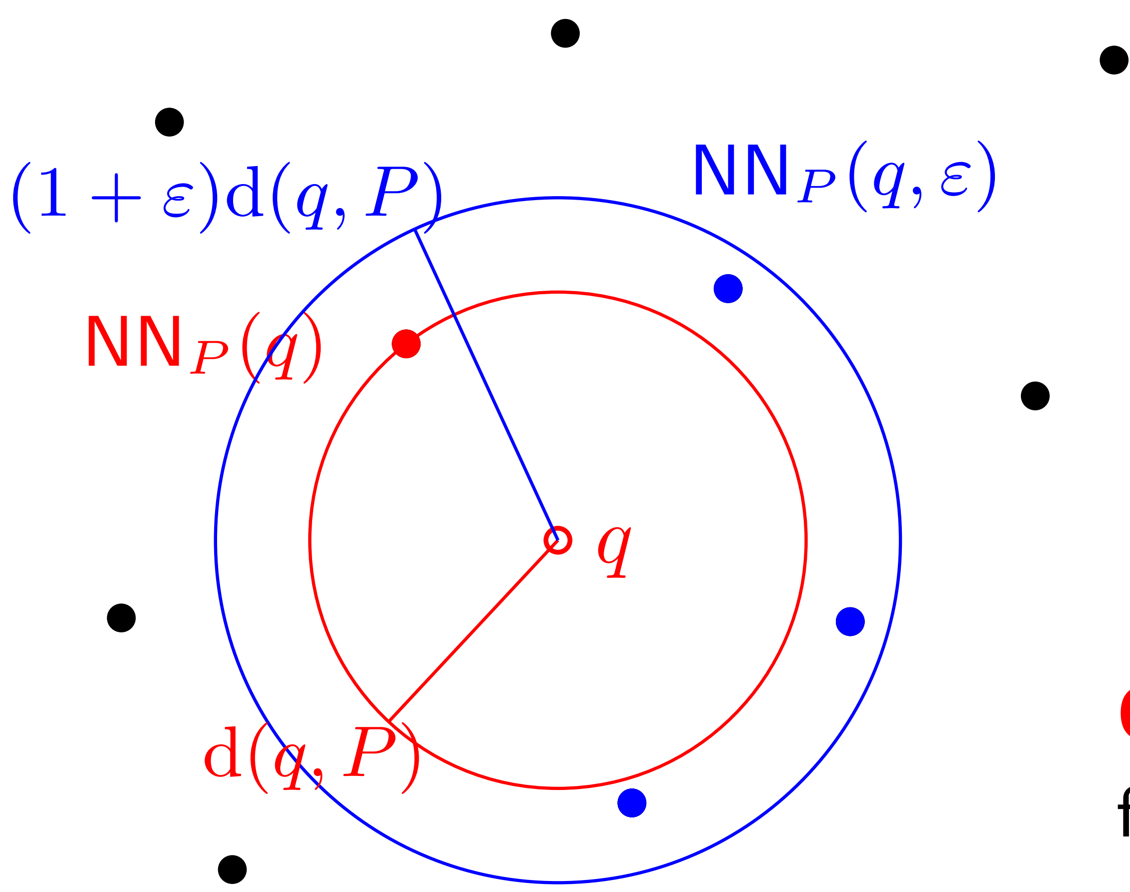
goal: find $p \in \mathsf{NN}_P(q,\varepsilon)$

Curse of Dimensionality: every DS for NN-search has either exponential size or exponential query time (in d) in the worst case.

$\rightarrow$ holds in theory and in practice for exact NN queries [Weber et al. '98]

$\rightarrow$ still holds for approximate queries in decision tree model [Arya et al. '98]

Back to the ε-NN problem



pre-processing input: P, ε

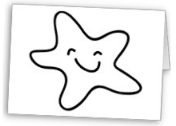
query input: q

goal: find $p \in \mathrm{NN}_P(q,\varepsilon)$

Curse of Dimensionality: every DS for NN-search has either exponential size or exponential query time (in d) in the worst case.

$\rightarrow$ holds in theory and in practice for exact NN queries [Weber et al. '98]

$\rightarrow$ still holds for approximate queries in decision tree model [Arya et al. '98]



$\rightarrow$ no longer true in Real-RAM model thanks to LSH [Indyk, Motwani '98]

Locality-Sensitive Hashing

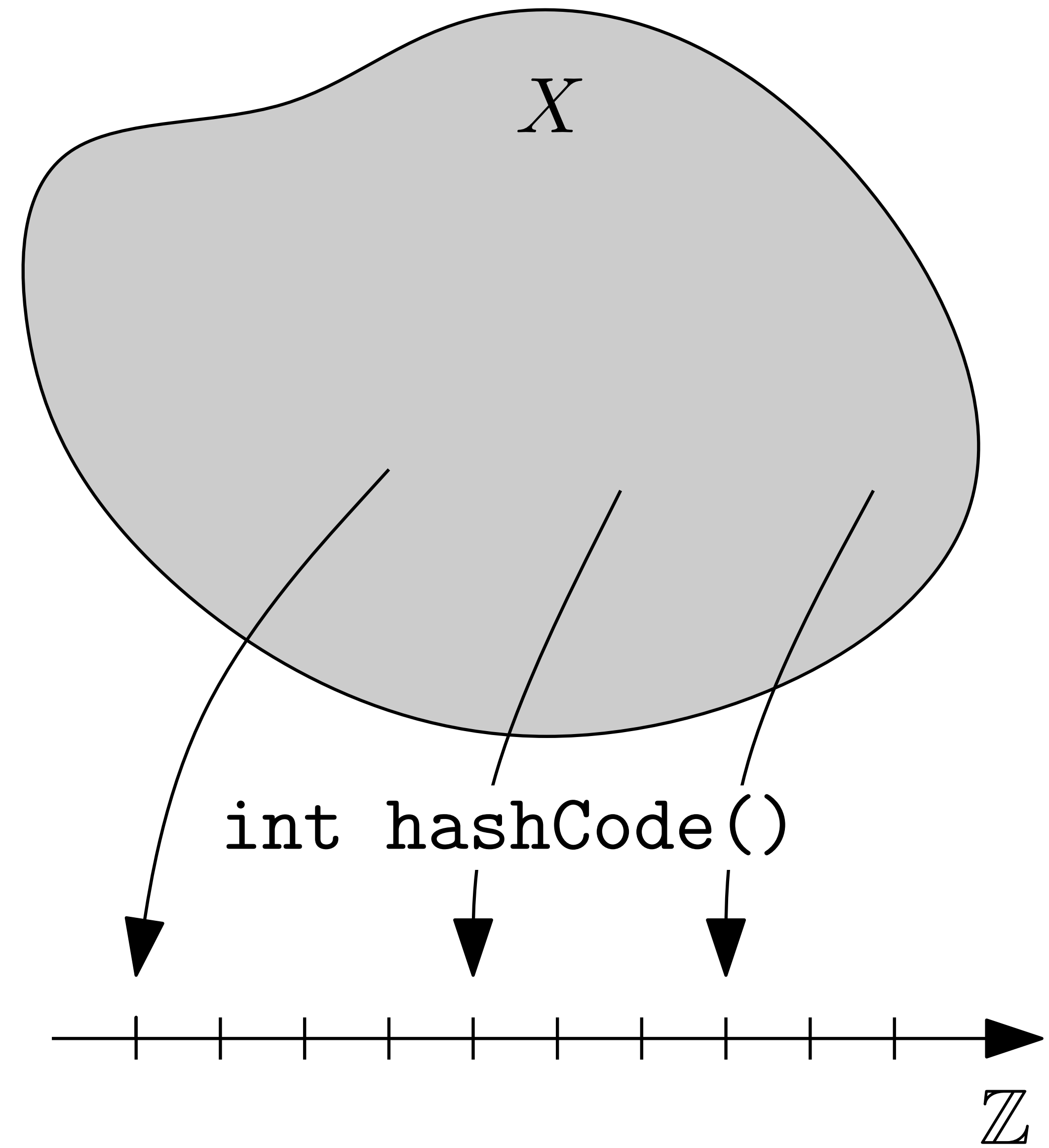
Comparing elements via hashing:

$\texttt{hashCode} : X \to \mathbb{Z}$

$x = y \Rightarrow \texttt{hashCode(}x\texttt{)} = \texttt{hashCode(}y\texttt{)}$

$x \neq y \Rightarrow \texttt{hashCode(}x\texttt{)} \neq \texttt{hashCode(}y\texttt{)}$

(no collisions hypothesis)



Locality-Sensitive Hashing

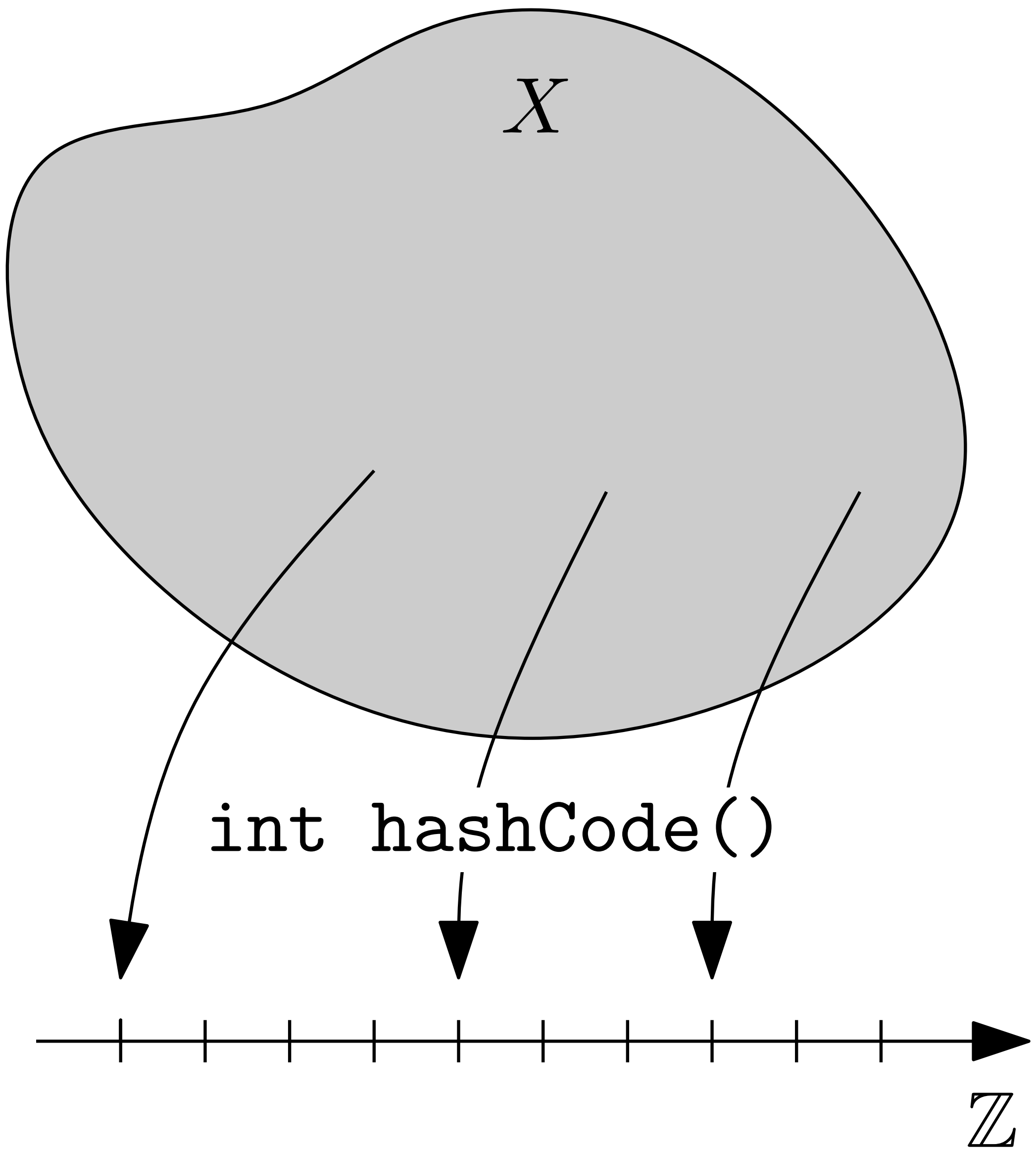
Comparing elements via hashing:

$\texttt{hashCode} : X \to \mathbb{Z}$

$x = y \Rightarrow \texttt{hashCode(}x\texttt{)}=\texttt{hashCode(}y\texttt{)}$

$x \neq y \Rightarrow \texttt{hashCode(}x\texttt{)}\neq\texttt{hashCode(}y\texttt{)}$

(no collisions hypothesis)



Metric case (X, d): given $r > 0$,

$\mathrm{d}(x, y) \leq r \Rightarrow \texttt{hashCode(}x\texttt{)}=\texttt{hashCode(}y\texttt{)}$

$\mathrm{d}(x, y) > r \Rightarrow \texttt{hashCode(}x\texttt{)}\neq\texttt{hashCode(}y\texttt{)}$

Locality-Sensitive Hashing

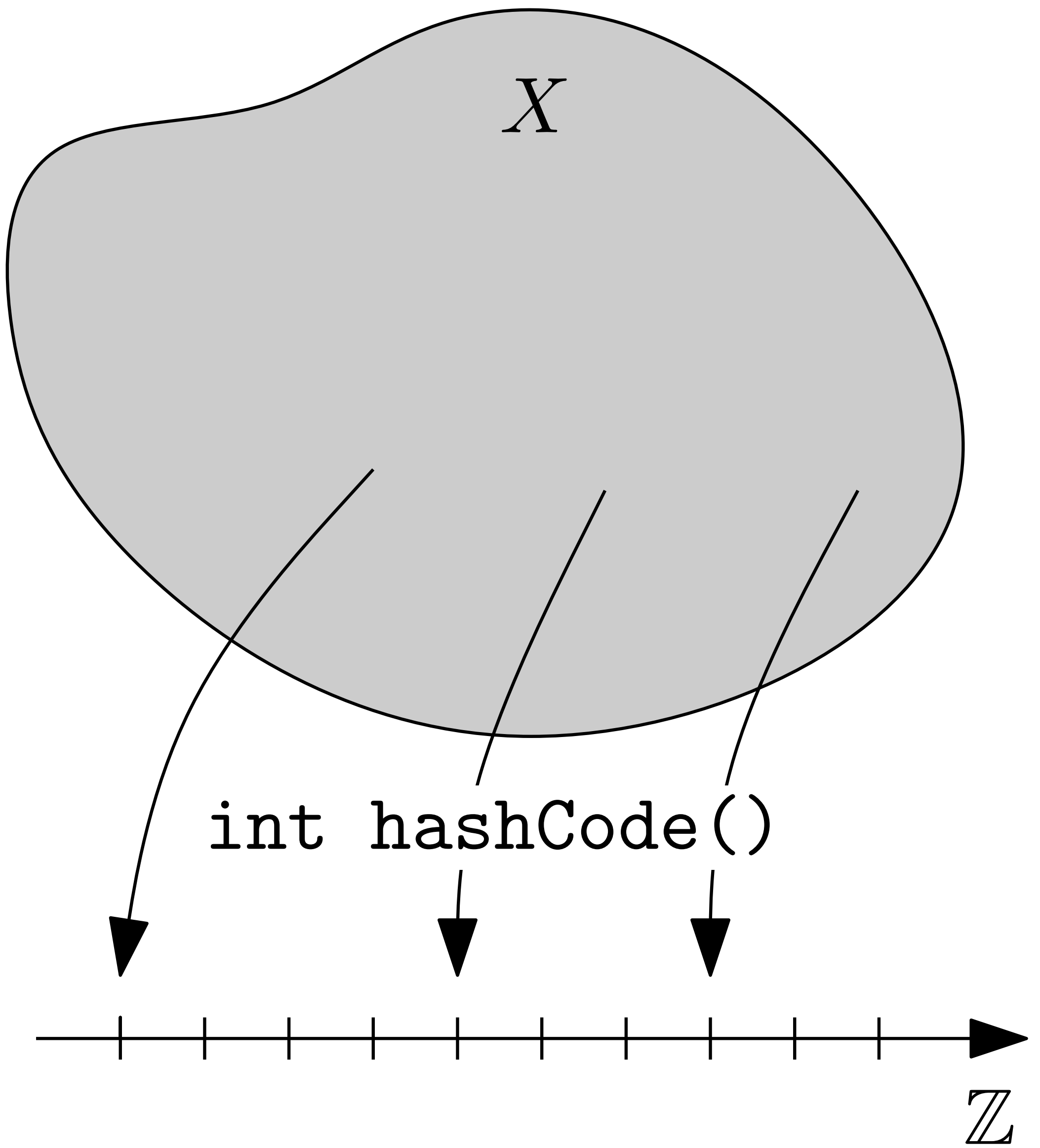
Comparing elements via hashing:

$\texttt{hashCode} : X \to \mathbb{Z}$

$x = y \Rightarrow \texttt{hashCode}(x) {=} \texttt{hashCode}(y)$

$x \neq y \Rightarrow \texttt{hashCode}(x) {\neq} \texttt{hashCode}(y)$

(no collisions hypothesis)



Metric case (X, d): given $r > 0$,

$\mathrm{d}(x, y) \leq r \Rightarrow \texttt{hashCode}(x) {=} \texttt{hashCode}(y)$

$\mathrm{d}(x, y) > r \Rightarrow \texttt{hashCode}(x) {\neq} \texttt{hashCode}(y)$

too good to be true $\to$ allow for some slack

Locality-Sensitive Hashing

Def: Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, \mathrm{d}) \to \mathcal{U}$ is (r_1, r_2, p_1, p_2)-**sensitive** if $\forall x, y \in X$,

- $\mathrm{d}(x, y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1$
- $\mathrm{d}(x, y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2$

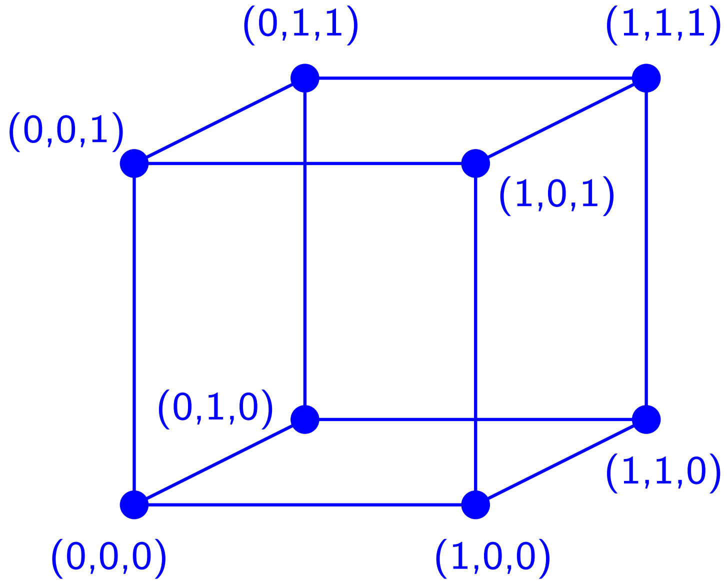
(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)

Locality-Sensitive Hashing

Def: Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, \mathrm{d}) \to \mathcal{U}$ is (r_1, r_2, p_1, p_2)-**sensitive** if $\forall x, y \in X$,

- $\mathrm{d}(x, y) \le r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \ge p_1$
- $\mathrm{d}(x, y) \ge r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \le p_2$

(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)



Example 1: $(X, \mathrm{d}) = (\{0,1\}^d, \mathrm{d}_{\mathcal{H}})$

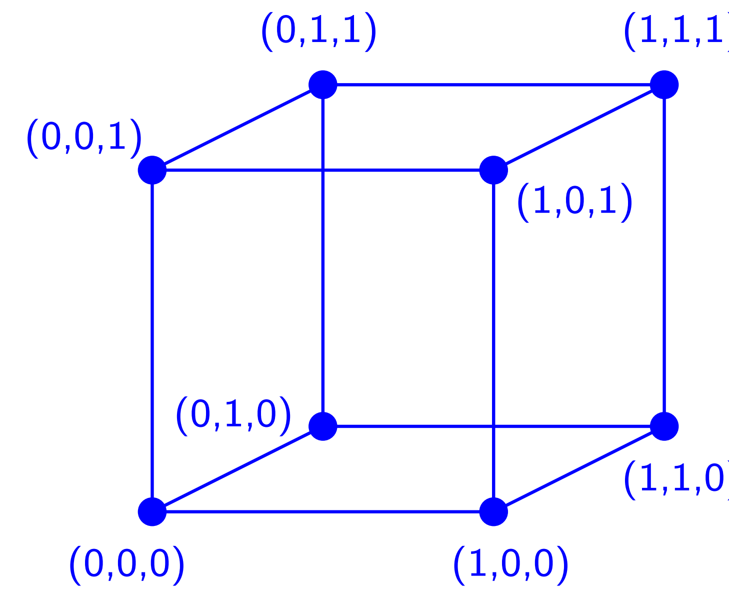
$\rightarrow$ take $\mathcal{F} = \{f_i\}_{i=1}^d$ where $f_i(b_1 \cdots b_d) = b_i$ | unif. proba. on $\mathcal{F}$

$\rightarrow$ $\mathcal{F}$ is $(r, r(1+\varepsilon), 1 - \frac{r}{d}, 1 - \frac{r(1+\varepsilon)}{d})$-sensitive for all $r \ge 1$ and $\varepsilon \ge 0$.

Locality-Sensitive Hashing

Def: Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, \mathrm{d}) \to \mathcal{U}$ is (r_1, r_2, p_1, p_2)-**sensitive** if $\forall x, y \in X$,

- $\mathrm{d}(x,y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1$
- $\mathrm{d}(x,y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2$



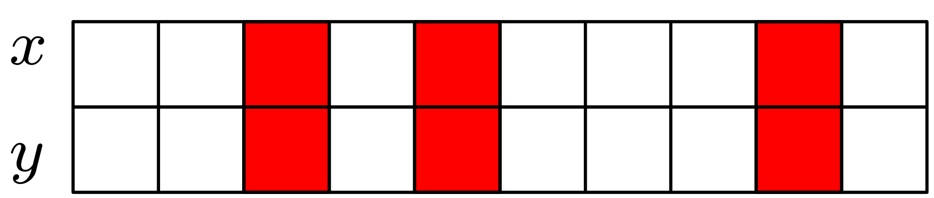
(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)

Example 1: $(X, \mathrm{d}) = (\{0,1\}^d, \mathrm{d}_{\mathcal{H}})$

$\rightarrow$ take $\mathcal{F} = \{f_i\}_{i=1}^d$ where $f_i(b_1 \cdots b_d) = b_i$ | unif. proba. on $\mathcal{F}$

$\rightarrow$ $\mathcal{F}$ is $(r, r(1+\varepsilon), 1 - \frac{r}{d}, 1 - \frac{r(1+\varepsilon)}{d})$-sensitive for all $r \geq 1$ and $\varepsilon \geq 0$.

proof: $\forall x, y$, $\mathbb{P}_f[f(x) = f(y)] = \frac{d - \mathrm{d}_{\mathcal{H}}(x,y)}{d} = 1 - \frac{\mathrm{d}_{\mathcal{H}}(x,y)}{d}$



$\mathrm{d}_{\mathcal{H}}(x,y)$ bits differ $\Rightarrow$ $d - \mathrm{d}_{\mathcal{H}}(x,y)$ functions make x and y collide

Locality-Sensitive Hashing

Def: Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, \mathrm{d}) \to \mathcal{U}$ is (r_1, r_2, p_1, p_2)-**sensitive** if $\forall x, y \in X$,

- $\mathrm{d}(x, y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1$
- $\mathrm{d}(x, y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2$

(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)

Example 2: $(X, \mathrm{d}) = (\mathbb{R}^d, \|\cdot\|_2)$

$\to$ take $\mathcal{F} = \{f_{\mathbf{v},b}\}_{\mathbf{v}\in\mathbb{R}^d}^{b\in[0,r]}$ where $f_{\mathbf{v},b}(x) = \lfloor \frac{x\cdot\mathbf{v}+b}{r} \rfloor$

$\to$ choose $\mathbf{v} = (v_1, \cdots, v_d)$ with $v_i \sim \mathcal{N}(0, 1)$, and b uniformly in $[0, r]$

$\to$ $\mathcal{F}$ is $(r, r(1+\varepsilon), p_1, p_2)$ sensitive for $p_1 = g(1)$ and $p_2 = g(1+\varepsilon)$,

where $g(\kappa) = 1 - 2\mathrm{cdf}(-r/\kappa) - \frac{2}{\sqrt{2\pi}r/\kappa}(1 - e^{-r^2/2\kappa^2})$

cumulative density func. of normal distrib.

Locality-Sensitive Hashing

Def: Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, \mathrm{d}) \to \mathcal{U}$ is (r_1, r_2, p_1, p_2)-**sensitive** if $\forall x, y \in X$,

- $\mathrm{d}(x, y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1$
- $\mathrm{d}(x, y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2$

(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)

Lemma [Johnson, Lindenstrauss 84]:

For any dimensions $0 < k < d$ there is a probability distribution μ over the projections $\mathbb{R}^d \to \mathbb{R}^k$ such that, given any set P of n points in $\mathbb{R}^d$ and any $\varepsilon \in (0, 1)$ with $k > 10 \ln n / \varepsilon^2$, a projection $\pi : \mathbb{R}^d \to \mathbb{R}^k$ sampled at random from μ satisfies w.h.p.

$$\forall p, q \in P,\ (1 - \varepsilon)\|p - q\| \leq \|\pi(p) - \pi(q)\| \leq (1 + \varepsilon)\|p - q\|$$

Locality-Sensitive Hashing

Def: Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, \mathrm{d}) \to \mathcal{U}$ is (r_1, r_2, p_1, p_2)-**sensitive** if $\forall x, y \in X$,

- $\mathrm{d}(x, y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1$
- $\mathrm{d}(x, y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2$

(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)

Lemma [Johnson, Lindenstrauss 84]:

For any dimensions $0 < k < d$ there is a probability distribution μ over the projections $\mathbb{R}^d \to \mathbb{R}^k$ such that, given any set P of n points in $\mathbb{R}^d$ and any $\varepsilon \in (0, 1)$ with $k > 10 \ln n / \varepsilon^2$, a projection $\pi : \mathbb{R}^d \to \mathbb{R}^k$ sampled at random from μ satisfies w.h.p.

$$\forall p, q \in P,\ (1 - \varepsilon)\|p - q\| \leq \|\pi(p) - \pi(q)\| \leq (1 + \varepsilon)\|p - q\|$$

Proof idea: take $\pi(x) = \sum_{i=1}^{k} (x \cdot \mathbf{v}_i)\mathbf{v}_i$, where the $\mathbf{v}_i$ are random vectors in $\mathbb{R}^d$ whose coordinates are chosen i.i.d. according to $\mathcal{N}(0, 1)$. Then exploit the concentration of measure on the sphere to show that π induces a low distortion.

Locality-Sensitive Hashing

Def: Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, \mathrm{d}) \to \mathcal{U}$ is (r_1, r_2, p_1, p_2)-**sensitive** if $\forall x, y \in X$,

- $\mathrm{d}(x, y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1$
- $\mathrm{d}(x, y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2$

(probability is over a random choice of function
according to a given probability distribution over $\mathcal{F}$)

$\to$ **General idea:**

- choose k-dimensional vector of random functions $(f_1, \cdots, f_k) \in \mathcal{F}^k$
- pre-process P by hashing its points into the corresponding hash table
- given $q \in X$, hash q and choose collision with smallest distance

Locality-Sensitive Hashing

Def: Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, \mathrm{d}) \to \mathcal{U}$ is (r_1, r_2, p_1, p_2)-**sensitive** if $\forall x, y \in X$,

- $\mathrm{d}(x, y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1$
- $\mathrm{d}(x, y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2$

(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)

$\rightarrow$ **General idea:**

- choose k-dimensional vector of random functions $(f_1, \cdots, f_k) \in \mathcal{F}^k$
- pre-process P by hashing its points into the corresponding hash table
- given $q \in X$, hash q and choose collision with smallest distance

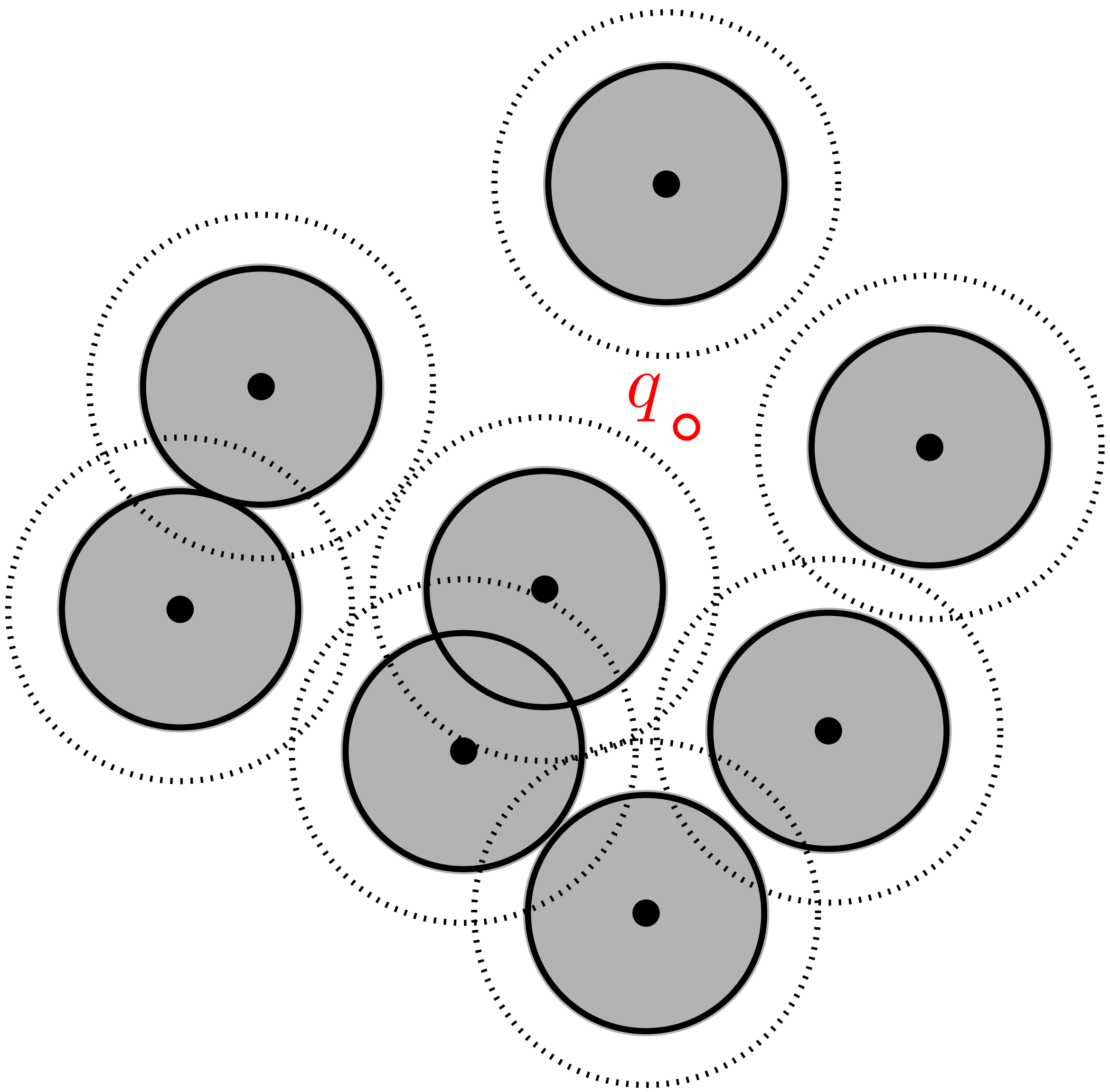
Technical detail: family works only for fixed r_1, r_2

$\rightarrow$ fix $r_1 = r$ and $r_2 = r(1 + \varepsilon)$, and solve (r, ε)-NN query

The (r, ε)-NN problem (PLEB)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in$NN$_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1 + \varepsilon)$ then answer NO,
- else $(r < \mathrm{d}(q, P) \leq r(1 + \varepsilon))$ give any of the above answers.



The (r, ε)-NN problem ($\textsc{pleb}$)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in$NN$_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1 + \varepsilon)$ then answer NO,
- else $(r < \mathrm{d}(q, P) \leq r(1 + \varepsilon))$ give any of the above answers.

Step 1: boost the sensitivity of the hash family

$\mathcal{G} = \{g = (f_1, \cdots, f_k) \in \mathcal{F}^k \mid f_1, \cdots, f_k$ chosen randomly in $\mathcal{F}\}$

The (r, ε)-NN problem ($\textsc{pleb}$)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in$NN$_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1+\varepsilon)$ then answer NO,
- else ($r < \mathrm{d}(q, P) \leq r(1+\varepsilon)$) give any of the above answers.

Step 1: boost the sensitivity of the hash family

$\mathcal{G} = \{g = (f_1, \cdots, f_k) \in \mathcal{F}^k \mid f_1, \cdots, f_k$ chosen randomly in $\mathcal{F}\}$

$\rightarrow \forall x, y$, $\mathrm{d}(x, y) \leq r \Rightarrow \mathbb{P}[g(x) = g(y)] \geq p_1^k$ (coords. are independent)

$\mathrm{d}(x, y) > r(1+\varepsilon) \Rightarrow \mathbb{P}[g(x) = g(y)] \leq p_2^k$

The (r, ε)-NN problem ($\textsc{pleb}$)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in$NN$_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1+\varepsilon)$ then answer NO,
- else $(r < \mathrm{d}(q, P) \leq r(1+\varepsilon))$ give any of the above answers.

Step 2: pre-process the data points

- choose τ random functions $g_1, \cdots, g_\tau$ from boosted hash family $\mathcal{G}$,

- initialise τ hash tables $H_1, \cdots, H_\tau$

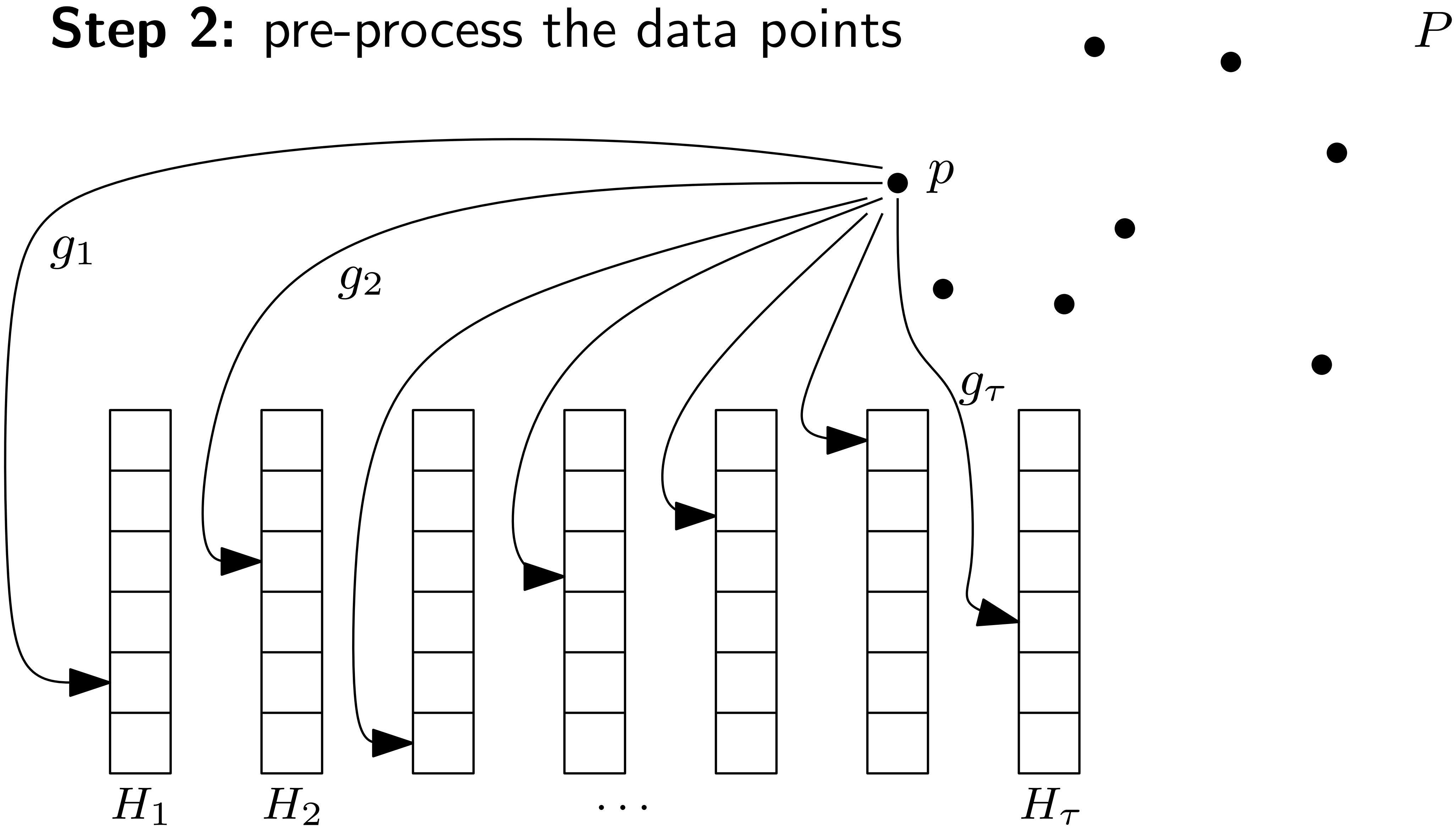
- $\forall i = 1, \cdots, \tau$, hash every point $p \in P$ into H_i using $g_i(p)$ as the key

- keep only one arbitrary point per non-empty entry

The (r, ε)-NN problem (PLEB)

Goal: pre-process P such that, for any query point q,
- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in$NN$_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1+\varepsilon)$ then answer NO,
- else $(r < \mathrm{d}(q, P) \leq r(1+\varepsilon))$ give any of the above answers.

Step 2: pre-process the data points

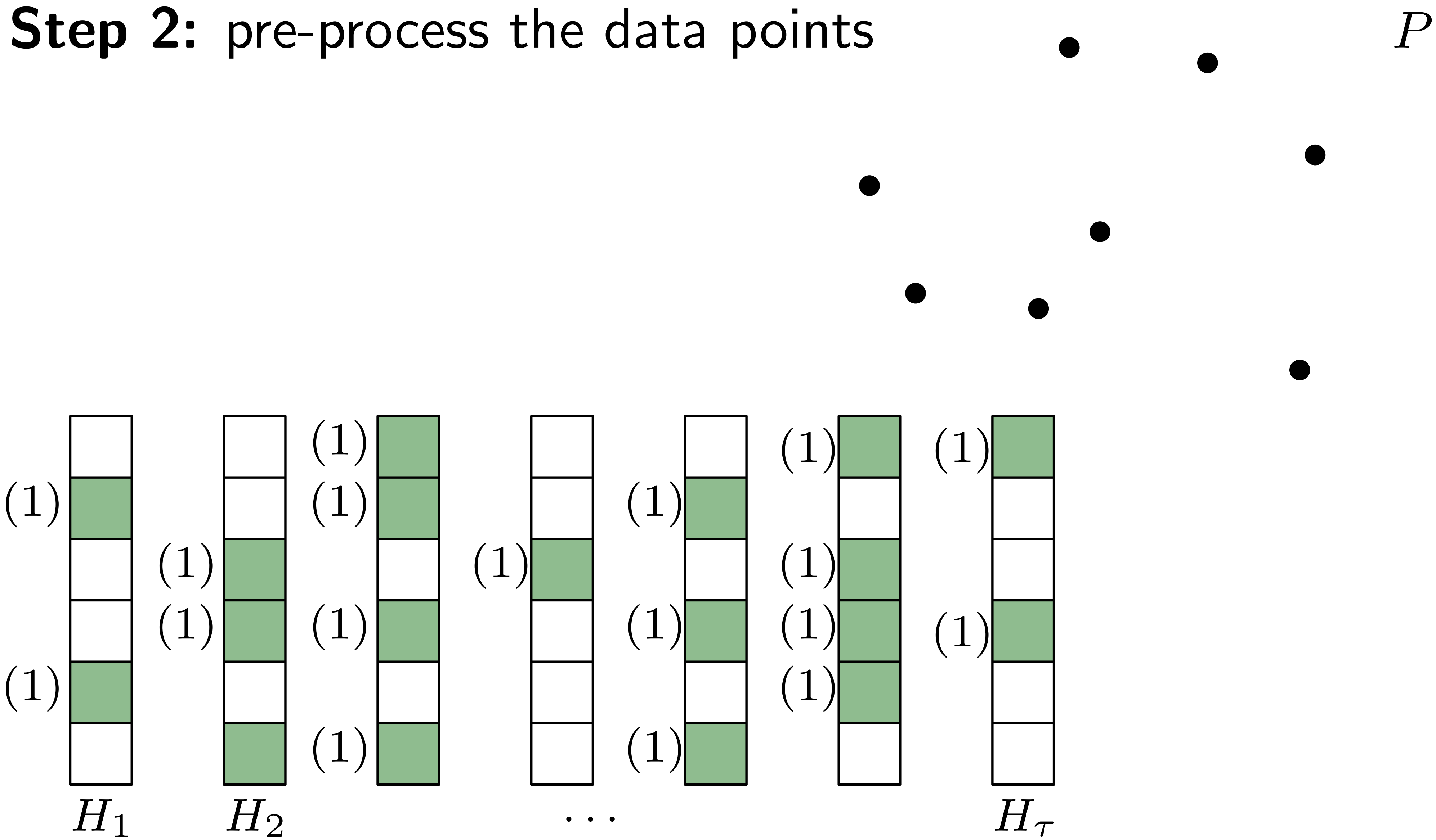


The (r, ε)-NN problem (PLEB)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in \mathrm{NN}_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1+\varepsilon)$ then answer NO,
- else ($r < \mathrm{d}(q, P) \leq r(1+\varepsilon)$) give any of the above answers.

Step 2: pre-process the data points

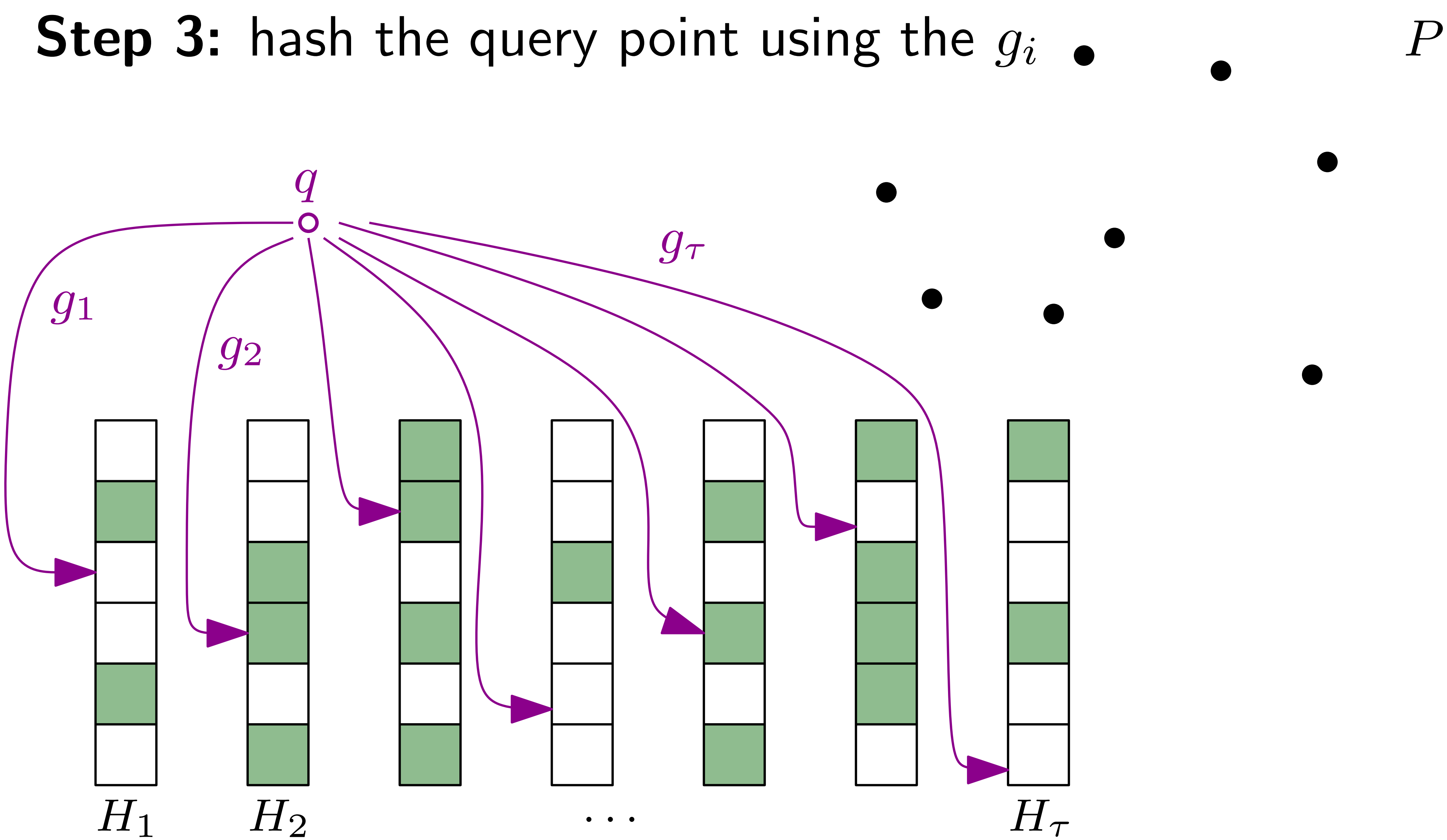


The (r, ε)-NN problem (PLEB)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in \mathrm{NN}_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1+\varepsilon)$ then answer NO,
- else $(r < \mathrm{d}(q, P) \leq r(1+\varepsilon))$ give any of the above answers.

Step 3: hash the query point using the g_i



The (r, ε)-NN problem (PLEB)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in$NN$_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1 + \varepsilon)$ then answer NO,
- else $(r < \mathrm{d}(q, P) \leq r(1 + \varepsilon))$ give any of the above answers.

Step 3: hash the query point using the g_i

- let $p_1, \cdots, p_l$ be the collisions ($l \leq \tau$)

- if some p_j is such that $\mathrm{d}(p_j, q) \leq r(1 + \varepsilon)$ then return YES and p_j

- else return NO

The (r, ε)-NN problem (PLEB)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in$NN$_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1+\varepsilon)$ then answer NO,
- else $(r < \mathrm{d}(q, P) \leq r(1+\varepsilon))$ give any of the above answers.

Analysis in a nutshell:

- test $\Rightarrow$ return NO whenever $\mathrm{d}(q, P) > r(1+\varepsilon)$

The (r, ε)-NN problem ($\textsc{pleb}$)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in$NN$_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1+\varepsilon)$ then answer NO,
- else ($r < \mathrm{d}(q, P) \leq r(1+\varepsilon)$) give any of the above answers.

Analysis in a nutshell:

- test $\Rightarrow$ return NO whenever $\mathrm{d}(q, P) > r(1+\varepsilon)$

- if $\exists p \in P$ s.t. $\mathrm{d}(p, q) \leq r$, then for a fixed $i \in \{1, \cdots, \tau\}$,

$\mathbb{P}[p$ collides with q in $H_i] \geq p_1^k$

$\forall p' \in P \setminus B(q, r(1+\varepsilon))$, $\mathbb{P}[p'$ collides with q in $H_i \mid p$ collides with q in $H_i]$

$$= \frac{\mathbb{P}[p \text{ and } p' \text{ collide with } q \text{ in } H_i]}{\mathbb{P}[p \text{ collides with } q \text{ in } H_i]}$$

$$\leq \frac{\mathbb{P}[p' \text{ collides with } q \text{ in } H_i]}{\mathbb{P}[p \text{ collides with } q \text{ in } H_i]} \leq \left(\frac{p_2}{p_1}\right)^k$$

The (r, ε)-NN problem ($\textsc{pleb}$)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in$NN$_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1+\varepsilon)$ then answer NO,
- else $(r < \mathrm{d}(q, P) \leq r(1+\varepsilon))$ give any of the above answers.

Analysis in a nutshell:

- test $\Rightarrow$ return NO whenever $\mathrm{d}(q, P) > r(1+\varepsilon)$

- if $\exists p \in P$ s.t. $\mathrm{d}(p, q) \leq r$, then for a fixed $i \in \{1, \cdots, \tau\}$,

union bound $\Rightarrow \mathbb{P}[H_i$ *succeeds*$] \geq p_1^k \left(1 - n\left(\frac{p_2}{p_1}\right)^k\right)$

τ independent hash tables $\Rightarrow$

$$\mathbb{P}[\text{some } H_i \textit{ succeeds}] \geq 1 - \left(1 - p_1^k\left(1 - n\left(\frac{p_2}{p_1}\right)\right)\right)^\tau$$

The (r, ε)-NN problem ($\textsc{PLEB}$)

Goal: pre-process P such that, for any query point q,

- if $\mathrm{d}(q, P) \leq r$ then answer YES and return some $p \in$NN$_P(q, r, \varepsilon)$,
- if $\mathrm{d}(q, P) > r(1+\varepsilon)$ then answer NO,
- else $(r < \mathrm{d}(q, P) \leq r(1+\varepsilon))$ give any of the above answers.

Analysis in a nutshell:

- test $\Rightarrow$ return NO whenever $\mathrm{d}(q, P) > r(1+\varepsilon)$

- if $\exists p \in P$ s.t. $\mathrm{d}(p, q) \leq r$, then for a fixed $i \in \{1, \cdots, \tau\}$,

union bound $\Rightarrow \mathbb{P}[H_i \textit{ succeeds}] \geq p_1^k \left(1 - n \left(\frac{p_2}{p_1}\right)^k\right)$

τ independent hash tables $\Rightarrow$

$$\mathbb{P}[\text{some } H_i \textit{ succeeds}] \geq 1 - \left(1 - p_1^k \left(1 - n \left(\frac{p_2}{p_1}\right)\right)\right)^\tau$$

Let $k = c \log n$ and $\tau = n^\varrho$ where $\varrho = \frac{\ln p_1}{\ln p_2} \in (0, 1)$.

$\Rightarrow$ $\boxed{\text{query time} = O(n^\varrho \log n),\ \Pr[\text{success}] \geq 1 - 1/n^{c\varrho}}$

From (r, ε)-NN to ε-NN

- Special case: $(X, \mathrm{d}) = (\{0,1\}^d, \mathrm{d}_{\mathcal{H}})$

  Observation: inter-point distances lie within $\{0, 1, 2, \cdots, d\}$

From (r, ε)-NN to ε-NN

- Special case: $(X, \mathrm{d}) = (\{0,1\}^d, \mathrm{d}_{\mathcal{H}})$

  Observation: inter-point distances lie within $\{0, 1, 2, \cdots, d\}$

$\rightarrow$ solve case $\mathrm{d}_{\mathcal{H}}(q, P) = 0$ independently (use lexicographical sorting)

$\rightarrow$ take geometric sequence $r_0 = 1$, $r_1 = 1 + \varepsilon$, $\cdots$, $r_j = (1+\varepsilon)^j$, $\cdots$

$\rightarrow$ for $j = 0$ to $\lceil \log_{1+\varepsilon} d \rceil = O(\frac{1}{\varepsilon} \log d)$, solve (r_j, r_{j+1})-NN query

$\rightarrow$ let j_l be the lowest j s.t. the answer to (r_j, r_{j+1})-MM query is YES

$\rightarrow$ return the output point of the (r_{j_l}, r_{j_l+1})-NN query

$\rightarrow$ if no YES answer, return output of case $\mathrm{d}_{\mathcal{H}}(q, P) = 0$

From (r, ε)-NN to ε-NN

- Special case: $(X, \mathrm{d}) = (\{0,1\}^d, \mathrm{d}_{\mathcal{H}})$

  Observation: inter-point distances lie within $\{0, 1, 2, \cdots, d\}$

  $\rightarrow$ query time $= O(\frac{1}{\varepsilon} n^{\varrho} \log n \log d)$

  (becomes $O(dn^{\varrho} \log n)$ if arithmetic sequence is used)

From (r, ε)-NN to ε-NN

- Special case: $(X, \mathrm{d}) = (\{0,1\}^d, \mathrm{d}_{\mathcal{H}})$

Observation: inter-point distances lie within $\{0, 1, 2, \cdots, d\}$

$\rightarrow$ query time $= O(\frac{1}{\varepsilon} n^{\varrho} \log n \log d)$

(becomes $O(dn^{\varrho} \log n)$ if arithmetic sequence is used)

Observation: deterministically, $r_{j_l} \geq \mathrm{d}_{\mathcal{H}}(q, P)/(1+\varepsilon)$

$\Rightarrow$ output $\in NN_P(q, \varepsilon(2+\varepsilon))$ iff LSH data structure works for $j = j_l$

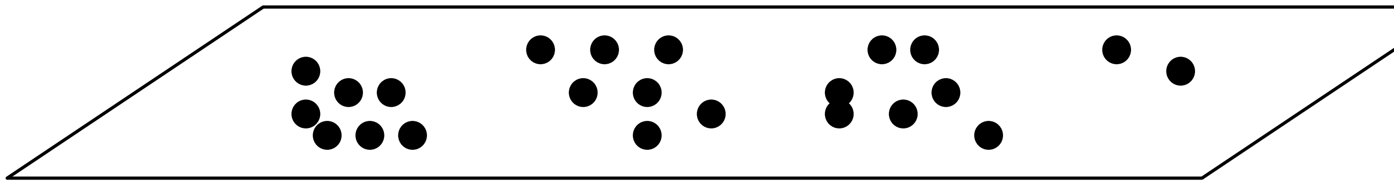
$\Rightarrow \mathbb{P}[\textit{success}] \geq 1 - 1/n^{c\varrho}$

From (r, ε)-NN to ε-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

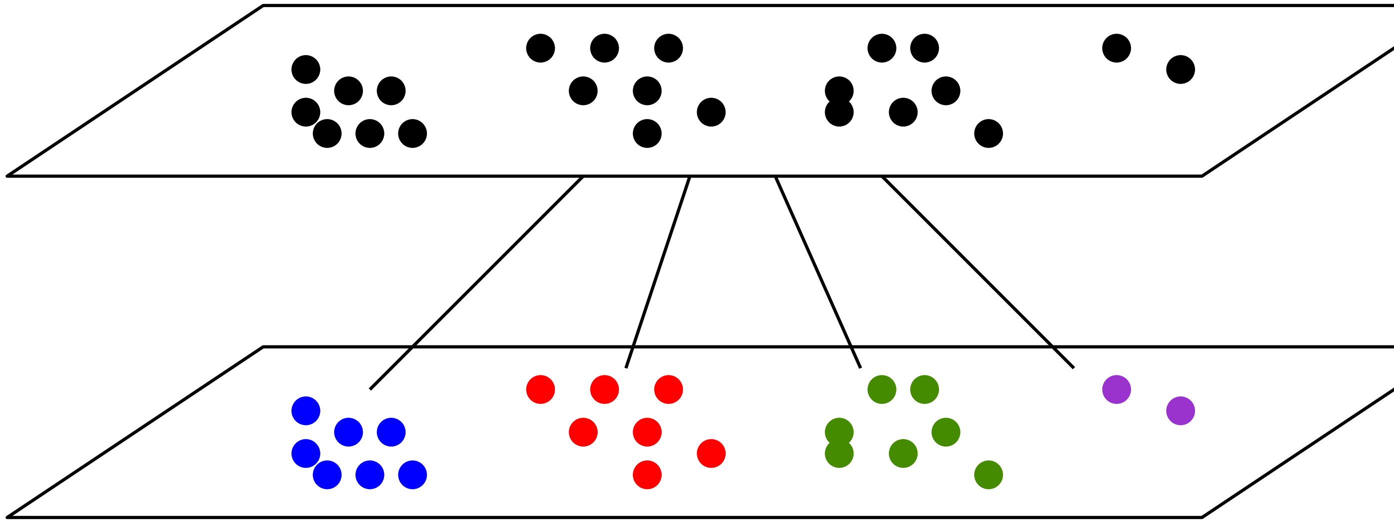
From (r, ε)-NN to ε-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences



From (r, ε)-NN to ε-NN

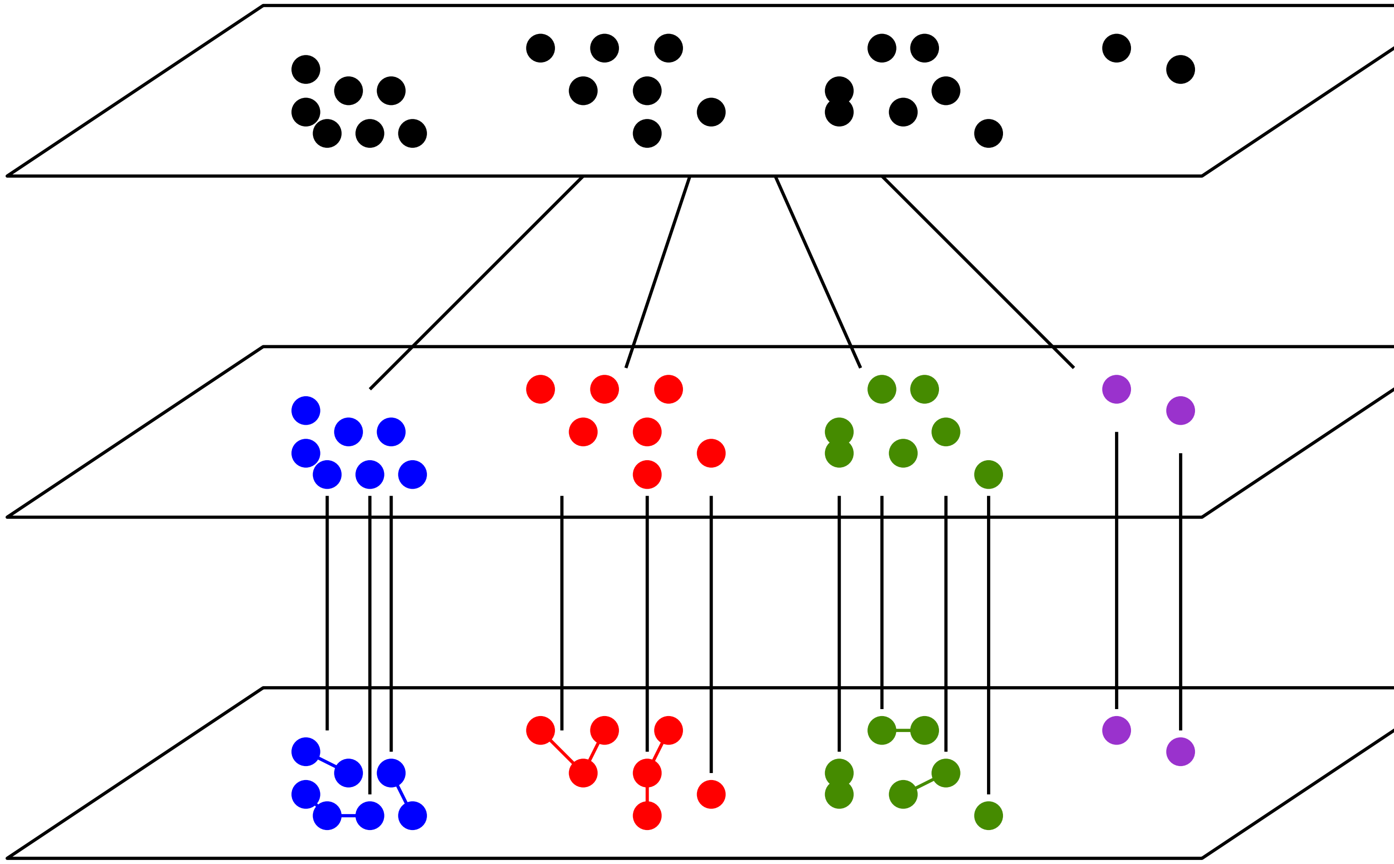
- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences



From (r, ε)-NN to ε-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

Assign $P_v \subseteq P$ and $[r_v, R_v]$ to each node v



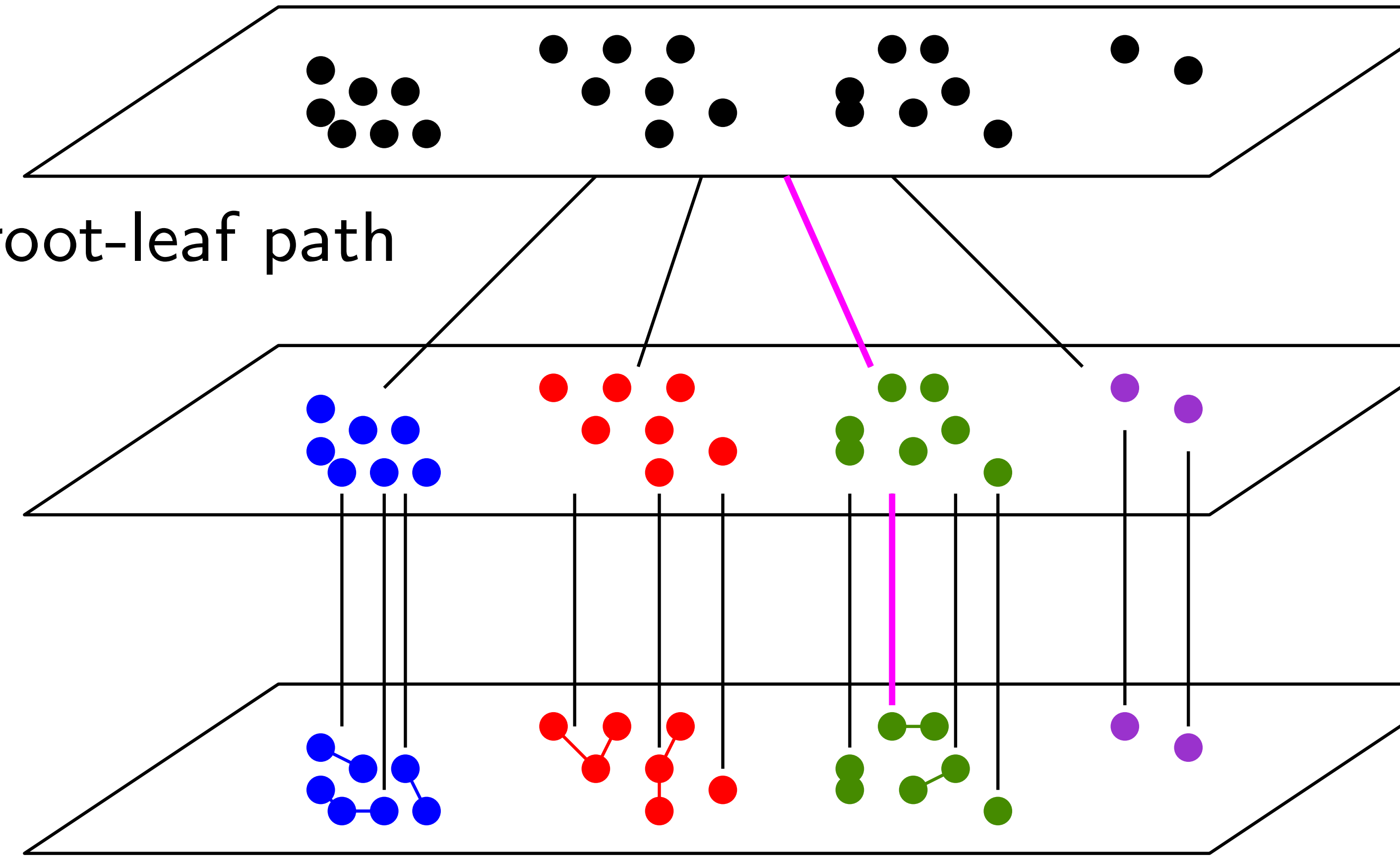
From (r, ε)-NN to ε-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

Assign $P_v \subseteq P$ and $[r_v, R_v]$ to each node v

ε-NN query:

- traverse down the tree along one root-leaf path



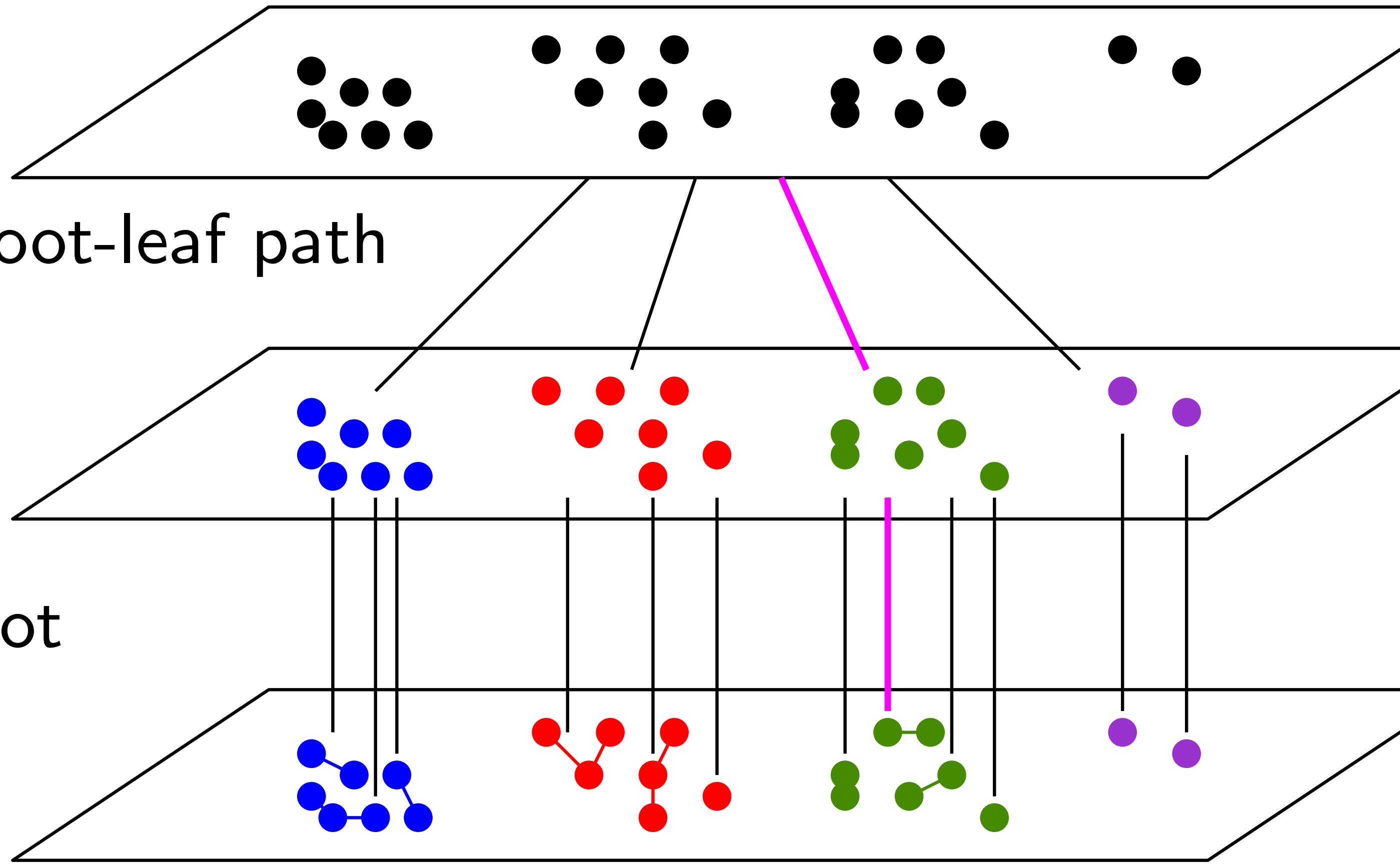
From (r, ε)-NN to ε-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

Assign $P_v \subseteq P$ and $[r_v, R_v]$ to each node v

ε-NN query:

- traverse down the tree along one root-leaf path
- at each visited node v, perform
  (r_v, ε)-NN and (R_v, ε)-NN queries
  $\rightarrow$ decide if $\mathrm{d}(q, P) \in [r_v, R_v]$ or not



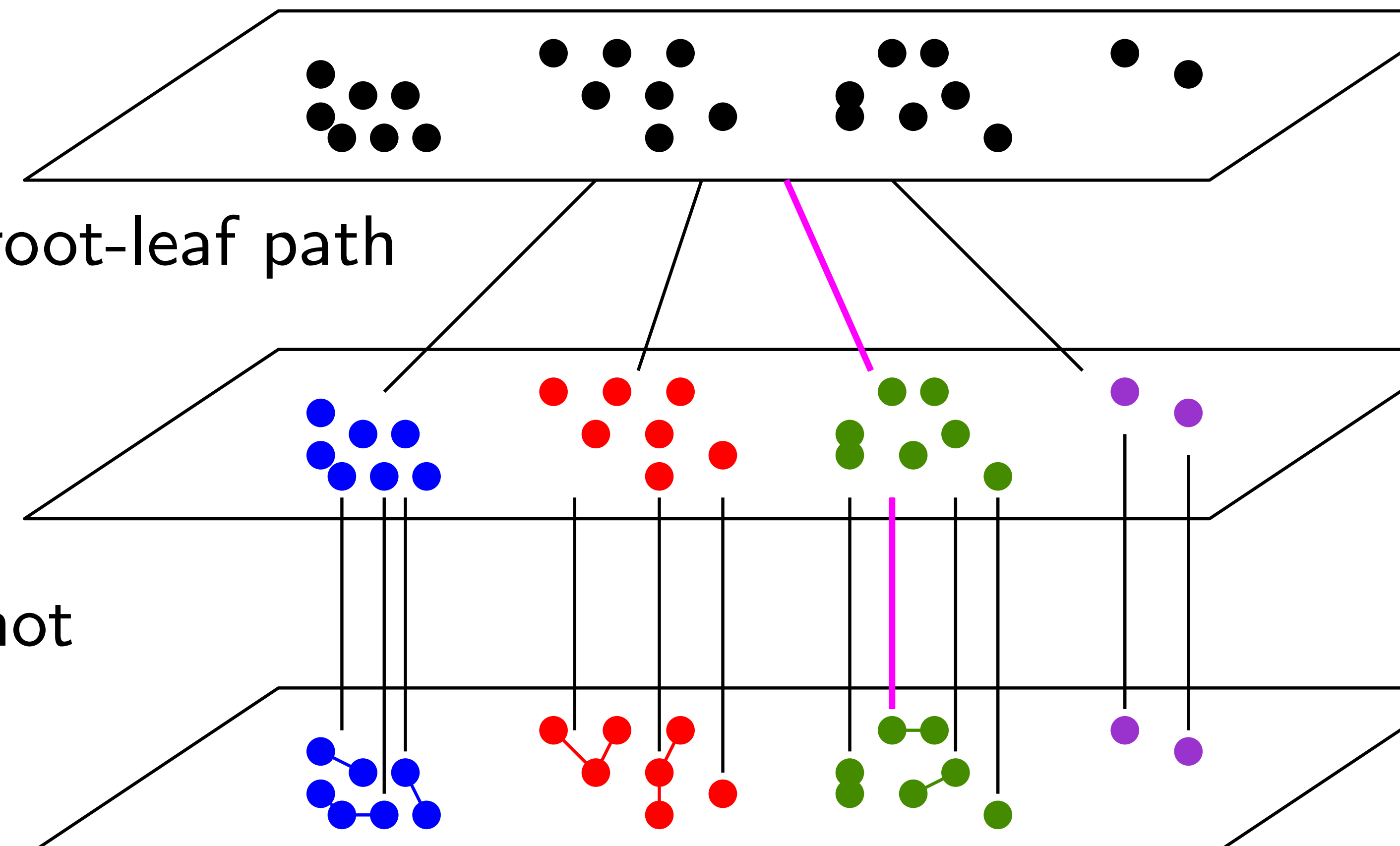
From (r, ε)-NN to ε-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

Assign $P_v \subseteq P$ and $[r_v, R_v]$ to each node v

ε-NN query:

- traverse down the tree along one root-leaf path
- at each visited node v, perform
  (r_v, ε)-NN and (R_v, ε)-NN queries
  $\rightarrow$ decide if $\mathrm{d}(q, P) \in [r_v, R_v]$ or not
  $\rightarrow$ if so, locate $\mathrm{d}(q, P)$ in $[r_v, R_v]$
  $\rightarrow$ if not, recurse into one child only



From (r, ε)-NN to ε-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

Assign $P_v \subseteq P$ and $[r_v, R_v]$ to each node v

ε-NN query:

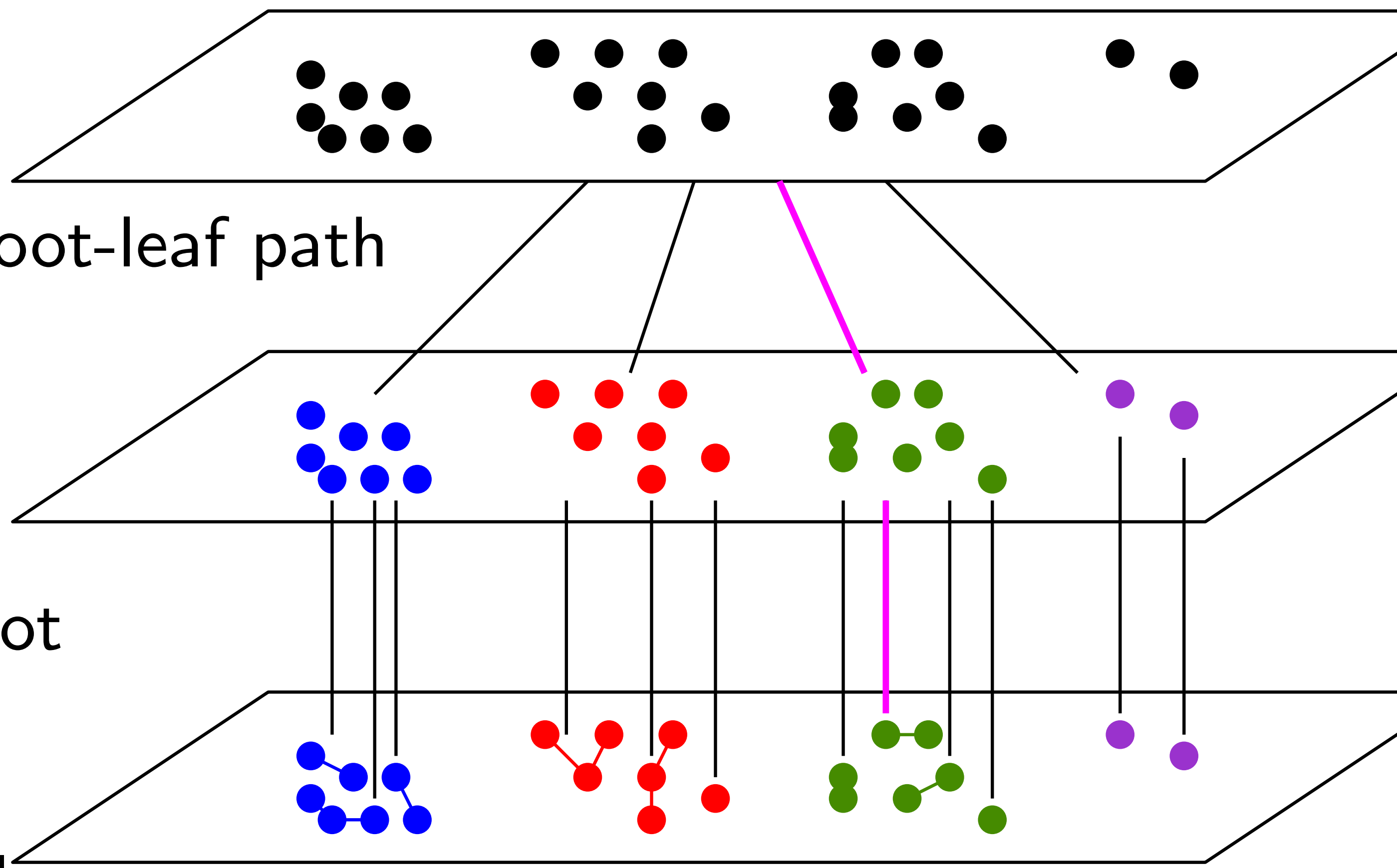
- traverse down the tree along one root-leaf path
- at each visited node v, perform

  (r_v, ε)-NN and (R_v, ε)-NN queries

  $\rightarrow$ decide if $\mathrm{d}(q, P) \in [r_v, R_v]$ or not

  $\rightarrow$ if so, locate $\mathrm{d}(q, P)$ in $[r_v, R_v]$

  $\rightarrow$ if not, recurse into one child only



$O(\frac{1}{\varepsilon} \log \frac{n}{\varepsilon})$ (r, ε)-NN queries per ε-NN query $\Rightarrow O(\frac{1}{\varepsilon} n^{\varrho} \log \frac{n}{\varepsilon})$ query time

Take-Home Messages

• (Approximate) NN search requires an exponential amount of resources (space/time) in the algebraic comparison tree model [Arya et al. 98].

• Using random hashing allows to beat the *curse of dimensionality*.

• The price to pay is that algorithms become almost linear
$\rightarrow$ in practice, a trade-off must be found.

• The complexity of the exact NN search problem is not fully understood.

$\rightarrow$ what about *reverse* NN search? [Cheong et al. 09], [Arthur, O. 10], ...