Course : Data mining
Topic : Similarity search

Aristides Gionis
Aalto University
Department of Computer Science

visiting in Sapienza University of Rome
fall 2016

reading assignment

Leskovec, Rajaraman, and Ullman

Mining of massive datasets

Cambridge University Press and online

http://www.mmds.org/

LRU book : chapter 3

An introductory tutorial on k-d trees

by Andrew Moore

finding similar objects

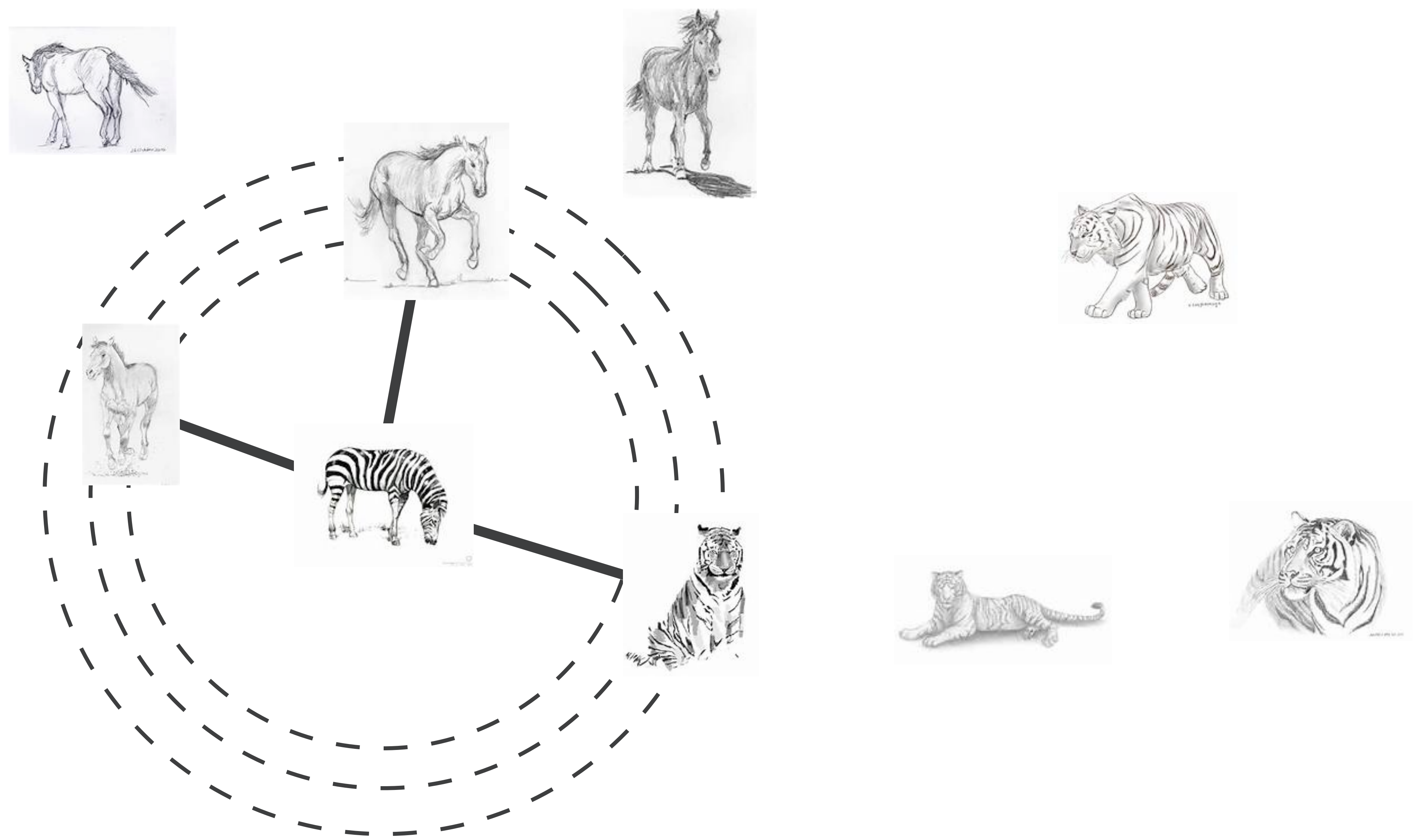
nearest-neighbor search

objects can be

- documents
- records of users
- images
- videos
- strings
- time series

similarity search: applications

in machine learning : nearest-neighbor rule



similarity search: applications

in information retrieval

a user wants to find similar documents or similar images to a given one

for clustering algorithms

the k-means algorithm assigns points to their nearest centers

finding similar objects

informal definition

two problems

1. similarity search problem

   given a set X of objects (off-line)

   given a query object q (query time)

   find the object in X that is most similar to q

2. all-pairs similarity problem

   given a set X of objects (off-line)

   find all pairs of objects in X that are similar

naive solutions

(assume a distance function $d : X \times X \to \mathbb{R}$)

1. similarity search problem

   given a set X of objects (off-line)

   given a query object q (query time)

   find the object in X that is most similar to q

naive solution:

compute $d(q, x)$ for all $x \in X$

return $x^* = \arg\min_{x \in X} d(q, x)$

naive solutions

(assume a distance function $d : X \times X \to \mathbb{R}$)

2. all-pairs similarity problem

   given a set X of objects (off-line)

   find all pairs of objects in X that are similar

   (say distance less than t)

naive solution:

   compute $d(x, y)$ for all $x, y \in X$

   return all pairs such that $d(x, y) \leq t$

naive solutions too inefficient

1. similarity search problem

   given a set X of objects (off-line)

   given a query object q (query time)

   find the object in X that is most similar to q

complexity $O(nd)$

   applications often require fast answers (milliseconds)

   we cannot afford scanning through all objects

goal to beat linear-time algorithm

   what does it mean?

   $O(\log n)$ $O(\text{poly}(\log n))$ $O(n^{1/2})$ $O(n^{1-e})$ $O(n+d)$?

naive solutions too inefficient

2. all-pairs similarity problem

   given a set X of objects (off-line)

   find all pairs of objects in X that are similar

complexity $O(n^2 d)$

   quadratic time is prohibitive for almost anything

warm up

let's focus on problem 1

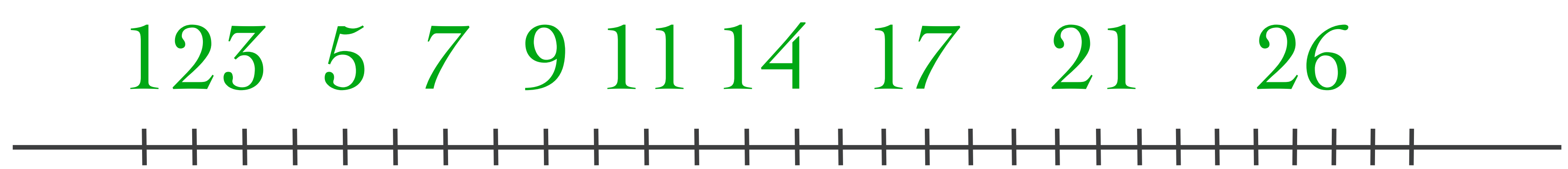
how to solve a problem for 1-d points?

example:

given $X = \{ 5, 9, 1, 11, 14, 3, 21, 7, 2, 17, 26 \}$

given $q=6$, what is the nearest point of q in X?

answer: sorting and binary search!



any lessons to learn?

1. trade-off preprocessing for query time
2. with one comparison prune away many points

generalization of the idea

space-partition algorithms

many algorithms that follow these principles

k-d trees is a popular variant

k-d trees in 2-d

a data structure to support range queries in R^2

- not the most efficient solution in theory
- everyone uses it in practice

preprocessing time : $O(n\log n)$

space complexity : $O(n)$

query time : $O(n^{1/2}+m)$

k-d trees in 2-d

algorithm :

choose x or y coordinate (alternate)

choose the median of the coordinate;

(this defines a horizontal or vertical line)

recurse on both sides

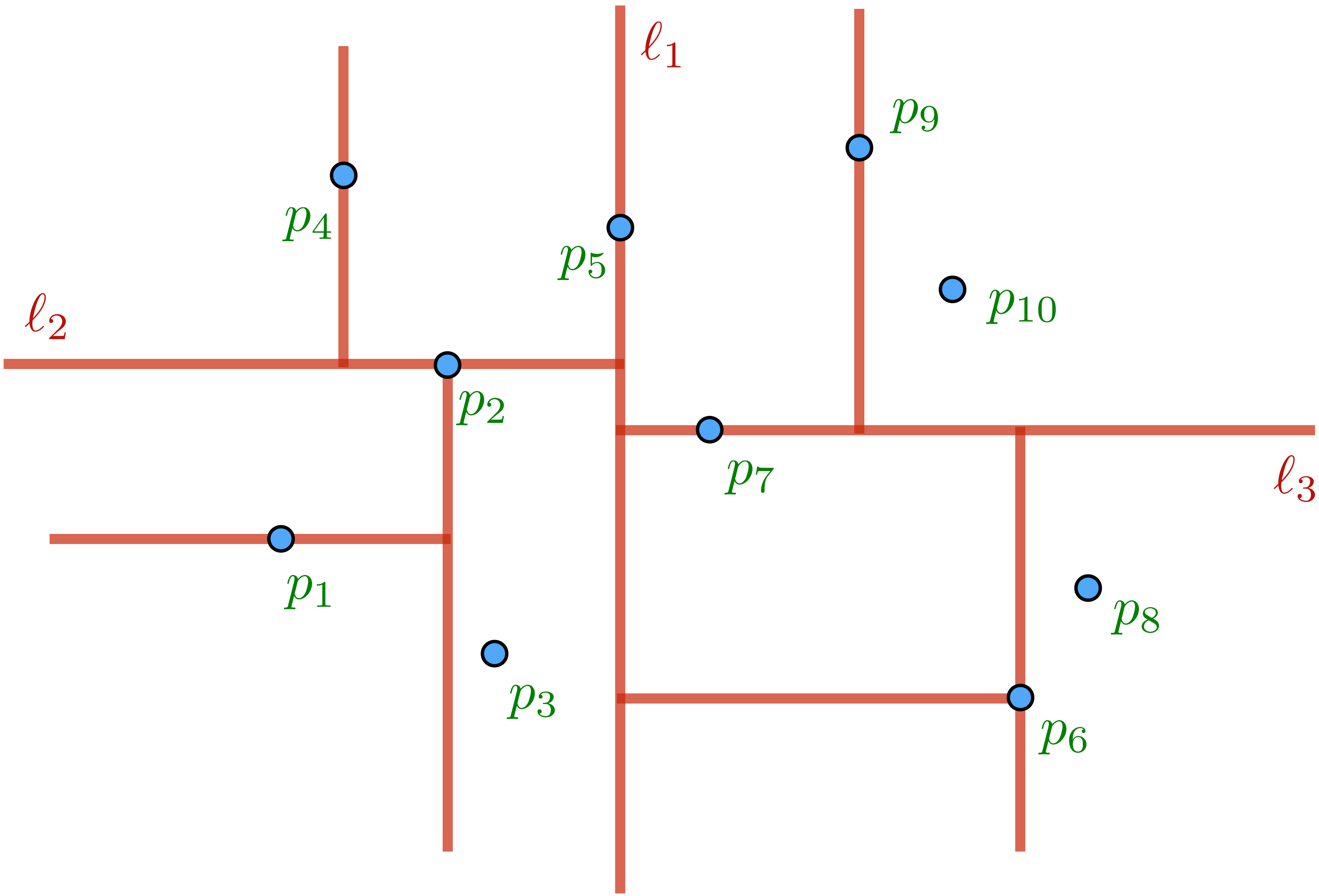
we get a binary tree

size : $O(n)$

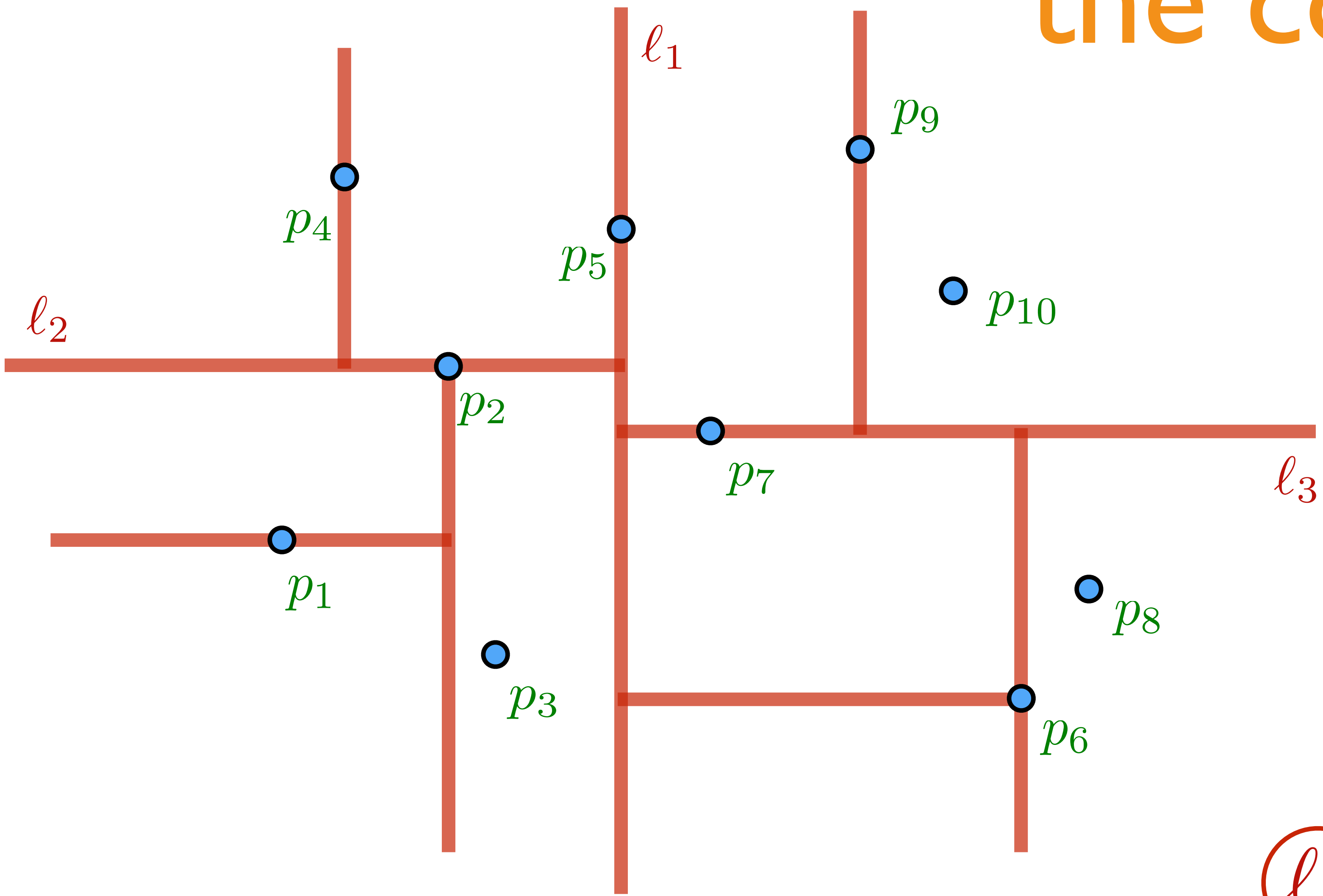
depth : $O(\log n)$

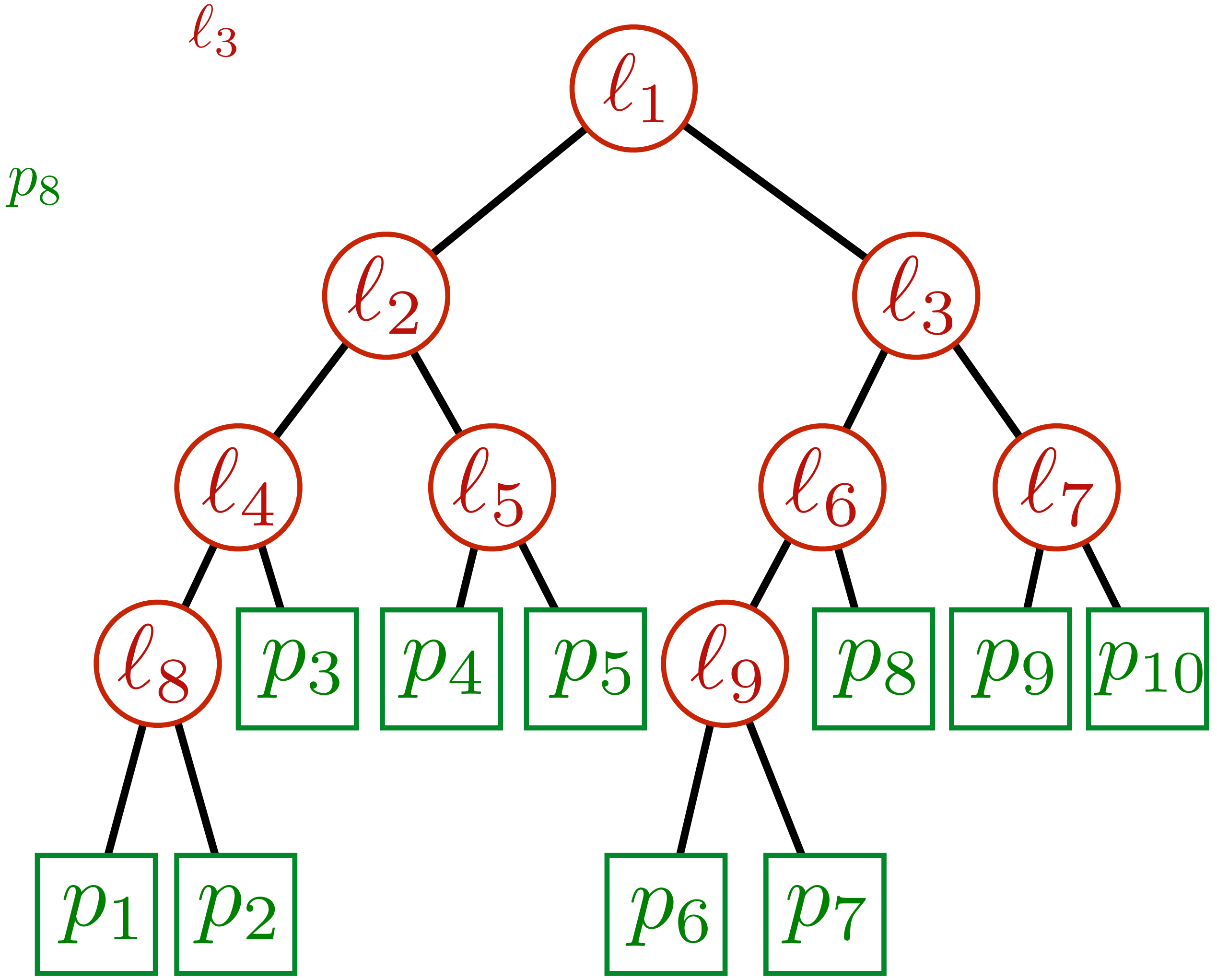
construction time : $O(n \log n)$

construction of k-d trees

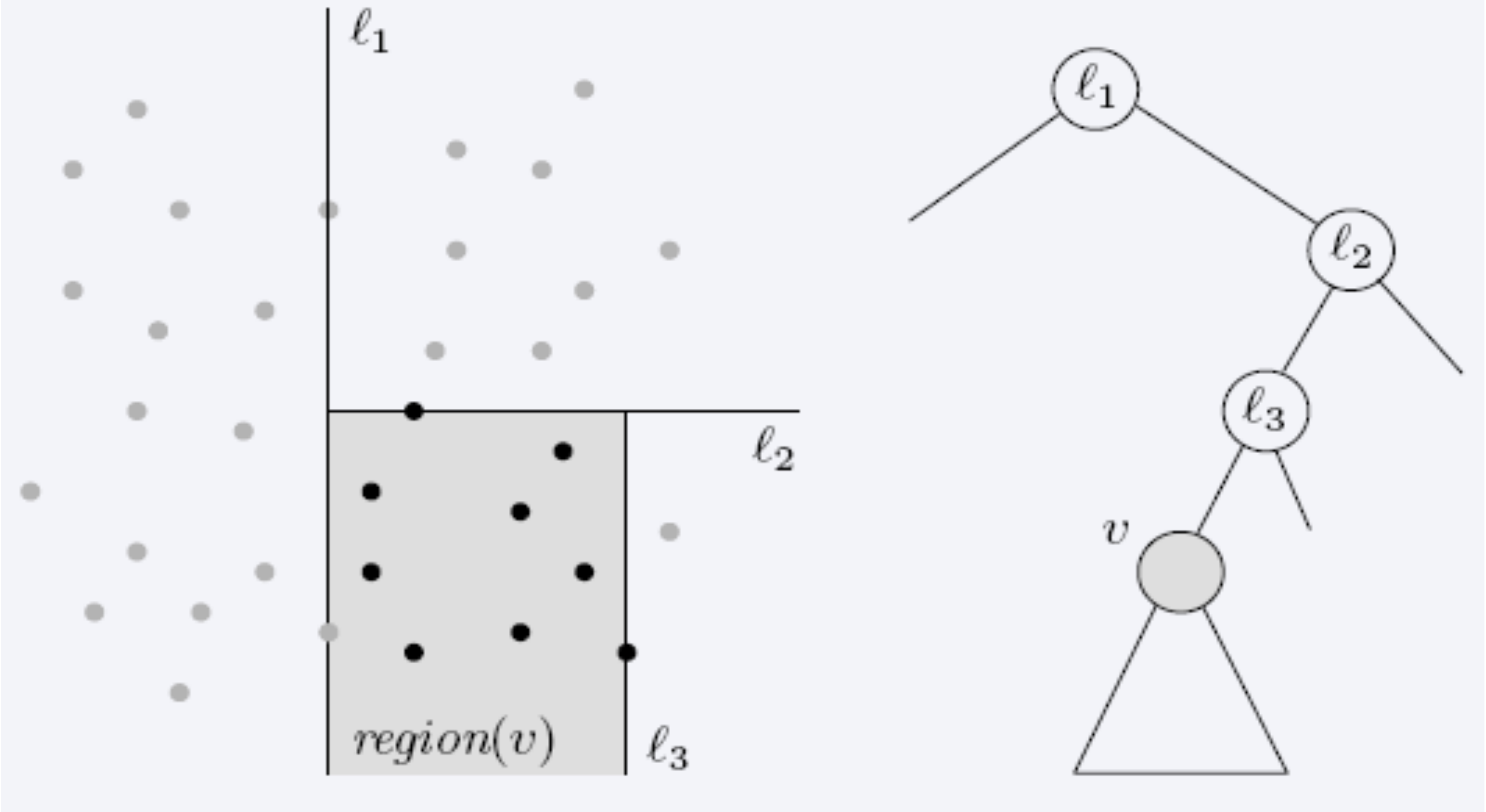


the complete k-d tree





region of a node



region(v) : the subtree rooted at v stores the points in **black** dots

searching in k-d trees

searching for nearest neighbor of a query q

start from the root and visit down the tree

at each point keep the NN found so far

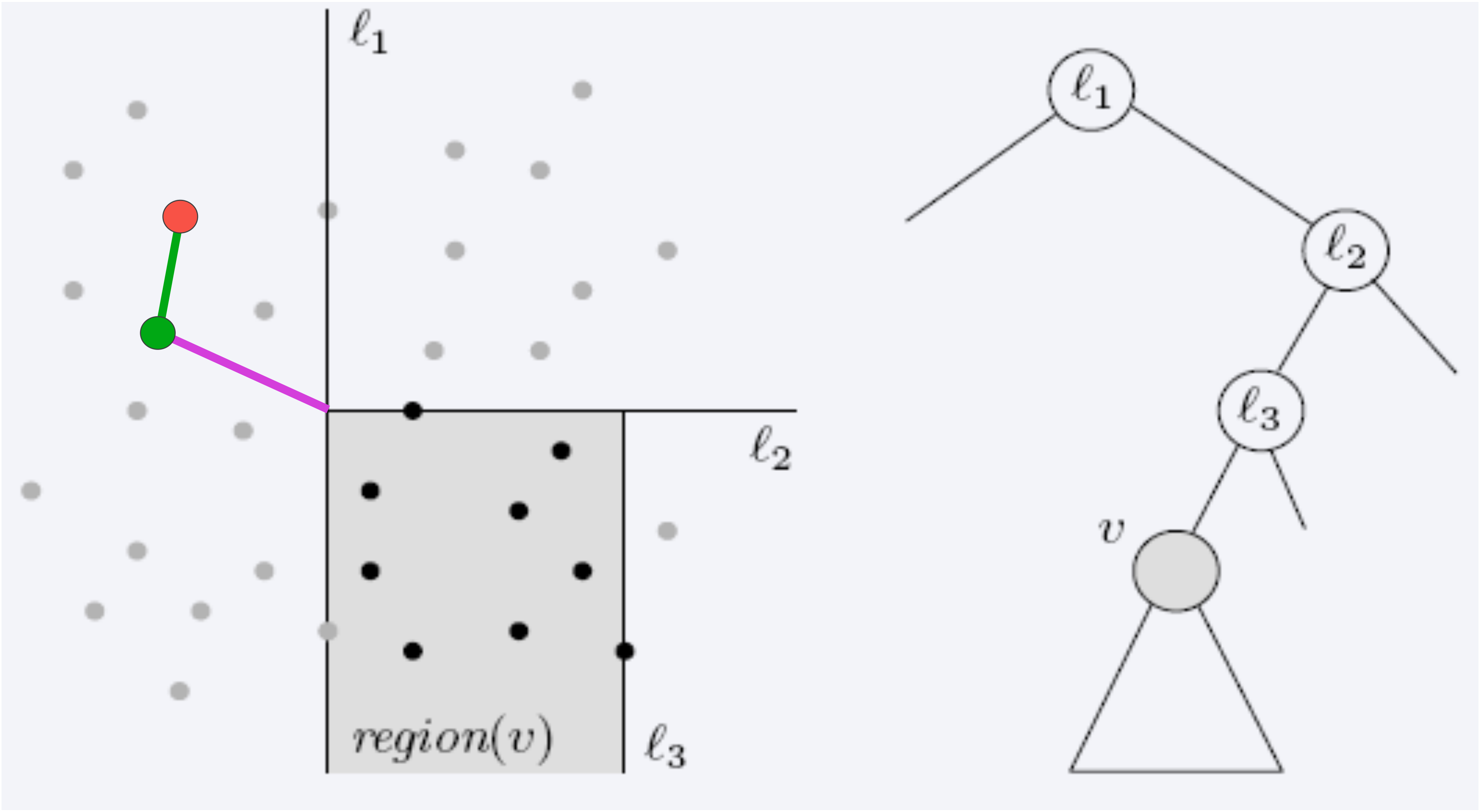
before visiting a tree node

estimate a lower bound distance

if lower bound larger than the current distance to NN, do not visit (prune)

(possible to visit both children of a node)

lower bound and pruning



green point : query

red point : current NN

purple line : lower bound

searching in k-d trees

range searching in X

given a rectangle R

find all points of X contained in R

range searching in k-d trees

start from v = root

search(v,R)

  if v is a leaf

    then report the point stored in v if it lies in R

  otherwise, if region(v) is contained in R

    report all points in the subtree(v)

  otherwise:

    if region(left(v)) intersects R

      then search(left(v),R)

    if reg(right(v)) intersects R

      then search(right(v),R)

query time analysis

time required by range searching in k-d trees is $O(n^{1/2}+k)$

where k is the number of points reported

total time to report all points is $O(k)$

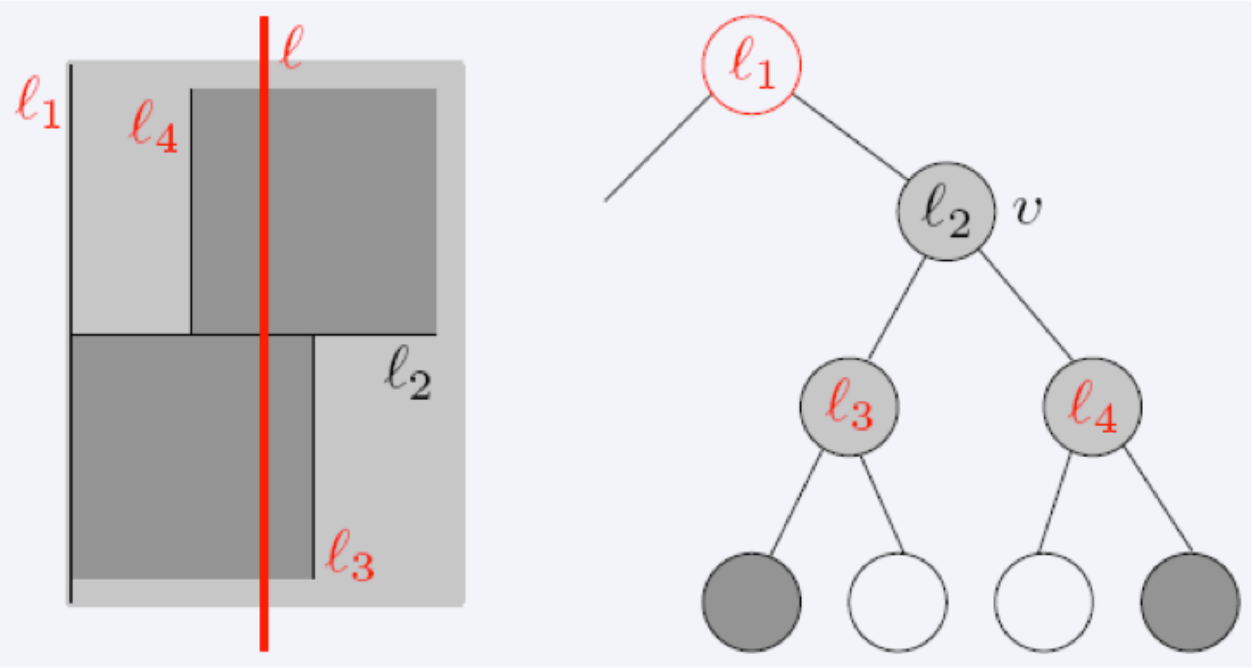
just need to bound the number of nodes v such that region(v)

intersects R

but is not contained in R

query time analysis

let $Q(n)$ be the max number of regions in an n-point k-d tree intersecting a line l, boundary of R



if l intersects region(v)

then after two levels it intersects 2 regions

the number of regions intersecting l is $Q(n)=2+2Q(n/4)$

solving the recurrence gives $Q(n)=(n^{1/2})$

k-d trees in d dimensions

supporting range queries in R^d

preprocessing time : $O(n \log n)$

space complexity : $O(n)$

query time : $O(n^{1-1/d}+k)$

k-d trees in d dimensions

construction is similar as in 2-d

split at the median by alternating coordinates

recursion stops when there is only one point left, which is stored as a leaf

impact of high dimensionality in similarity search

as dimension grows the similarity search problem becomes harder

for the range searching problem this is shown by the $O(n^{1-1/d}+k)$ bound

for the nearest neighbor problem, the pruning rule becomes not effective

as dimension grows the performance of any index degrades to linear search

point of frustration in the research community

a.k.a. the curse of the dimensionality

any catch?

idea relies on having vector-space objects

what happens with points in a metric space?

the space-partition idea generalizes to metric spaces

vantage-point algorithm

consider a metric space (X,d)

partition the objects in X using a binary tree

at each step, when partitioning n objects, choose a point v in X (vantage point)

right subtree $R(v)$:

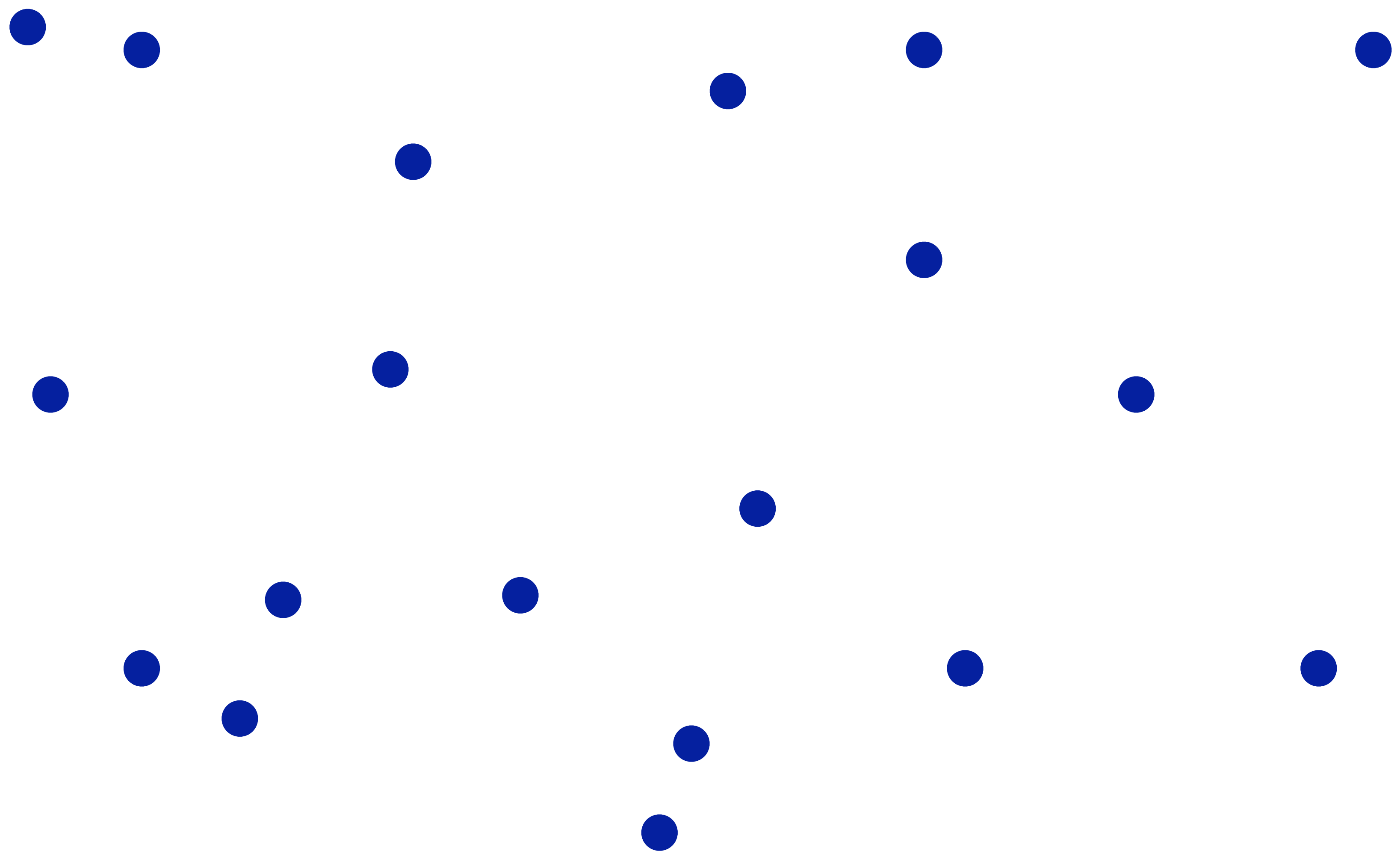
 the set of the $n/2$ points that are closest to v

left subtree $L(v)$:

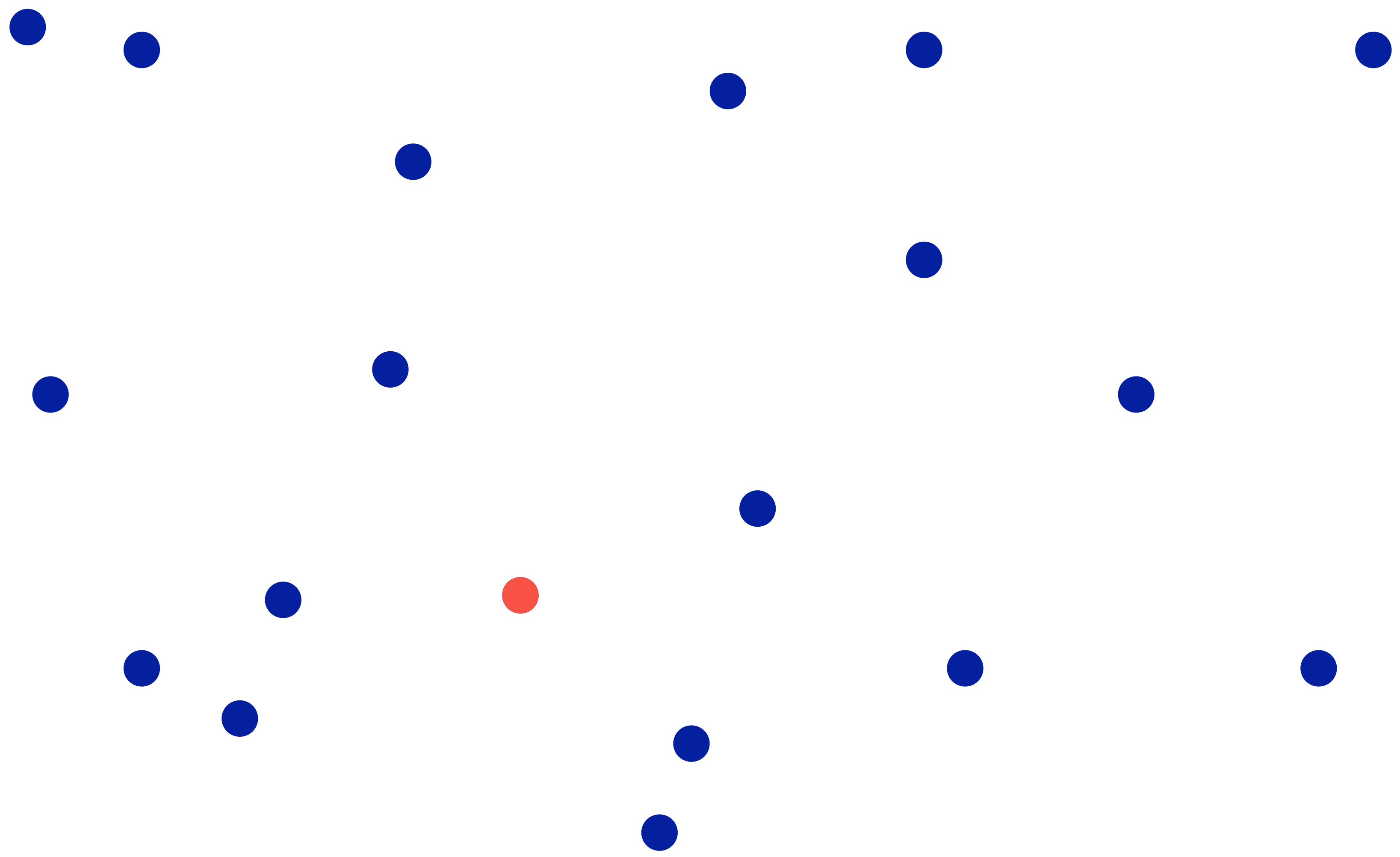
 the rest of the points

recurse on $R(v)$ and $L(v)$

vantage-point algorithm

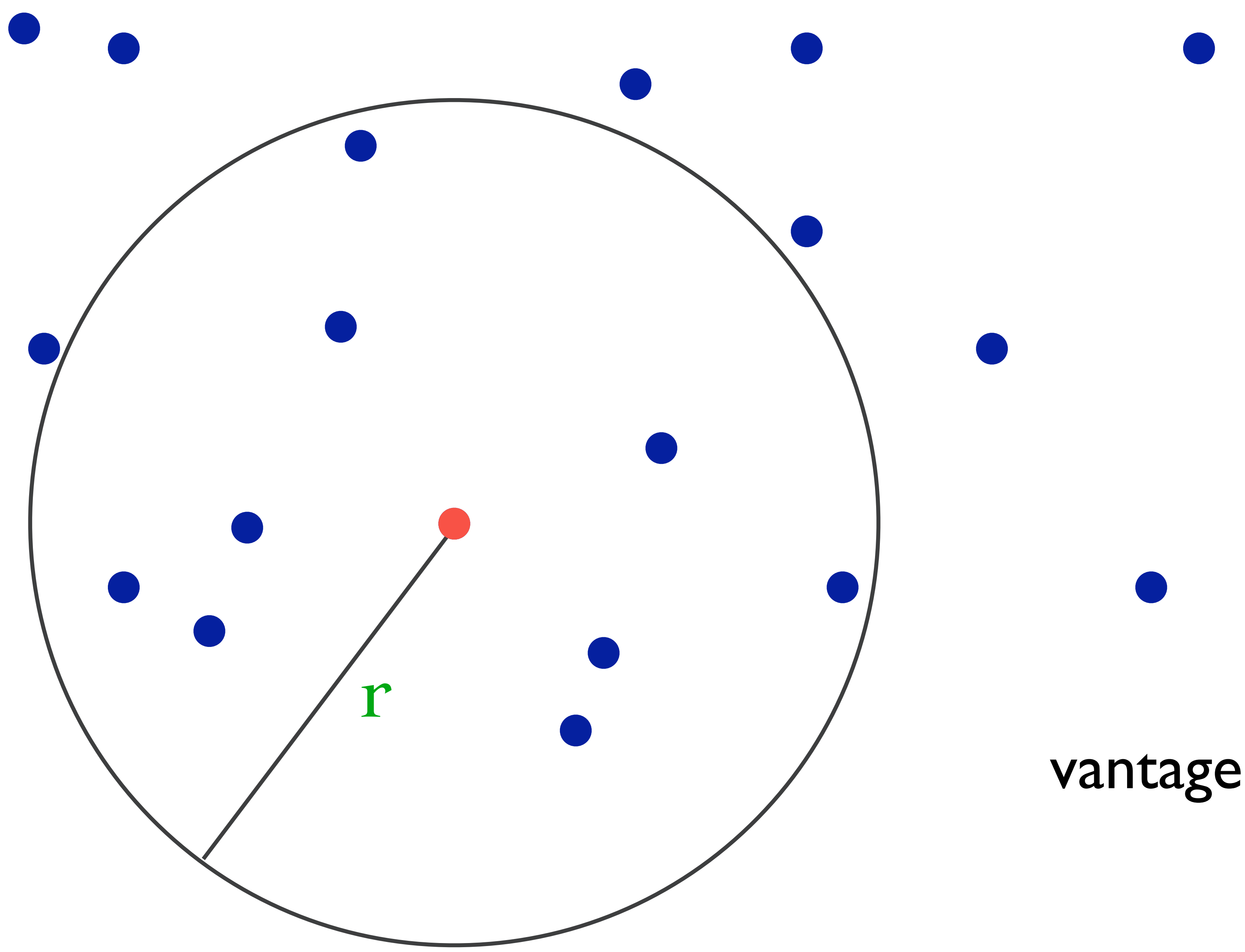


vantage-point algorithm



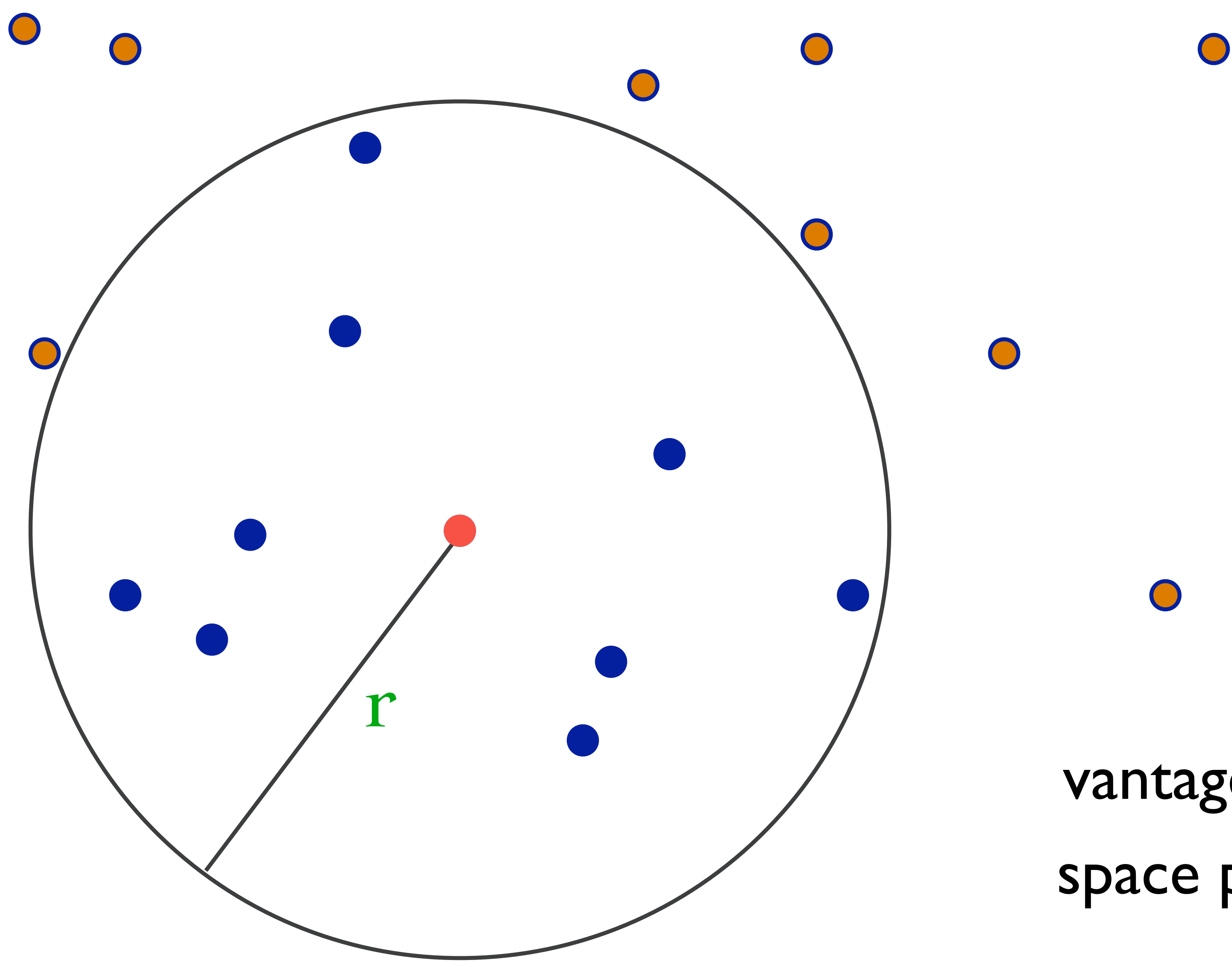
vantage point

vantage-point algorithm



vantage point

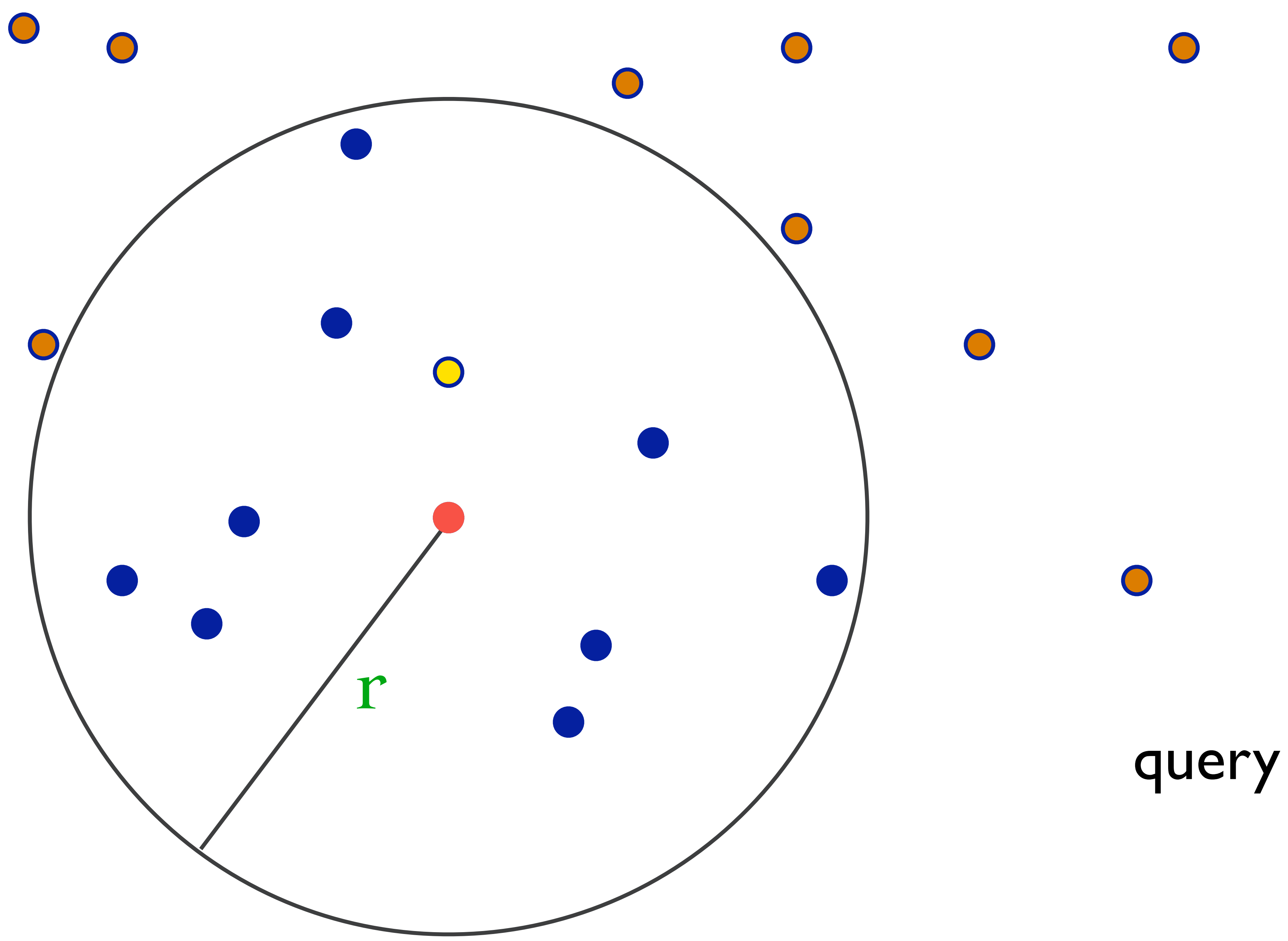
vantage-point algorithm



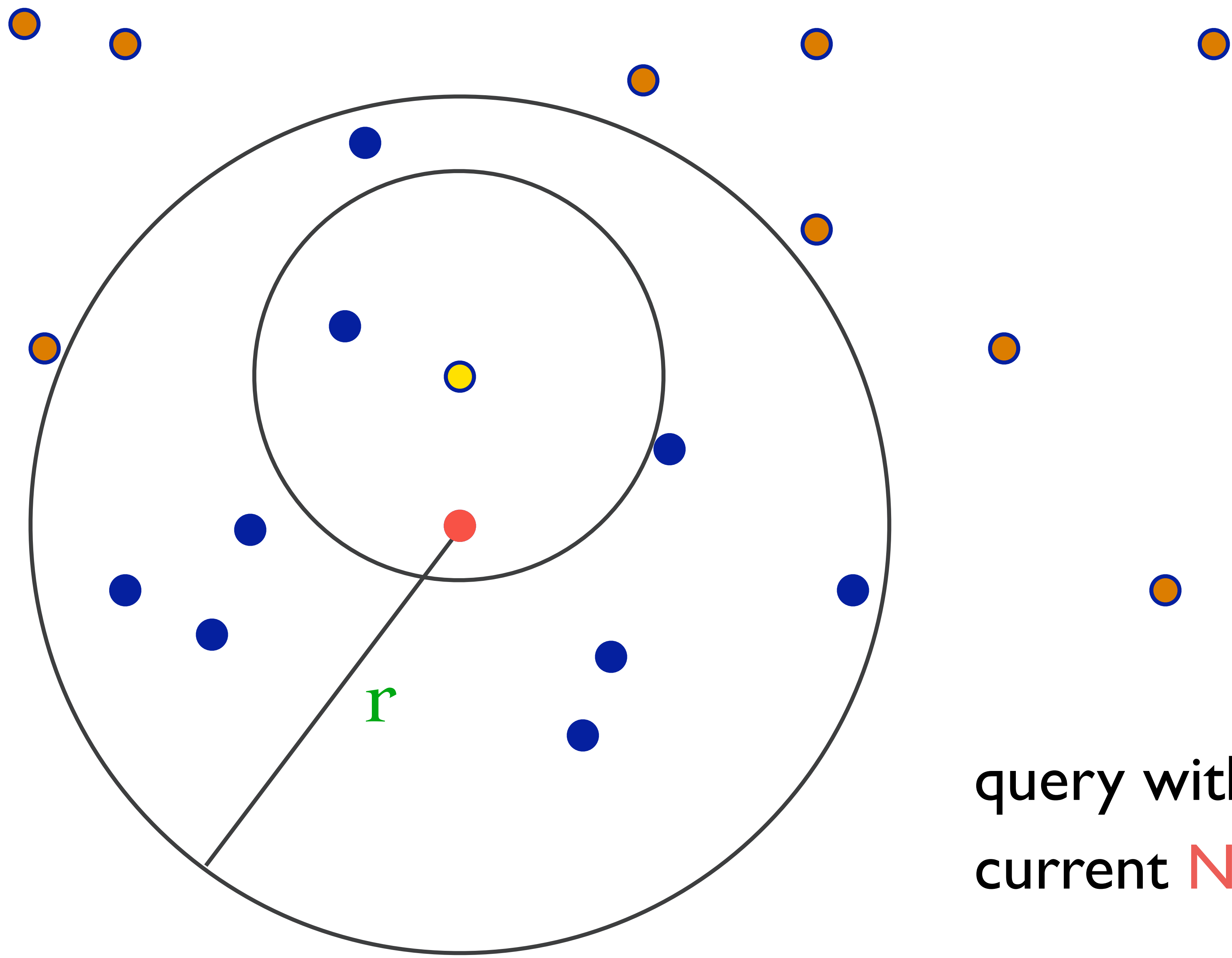
vantage point

space partition

vantage-point algorithm

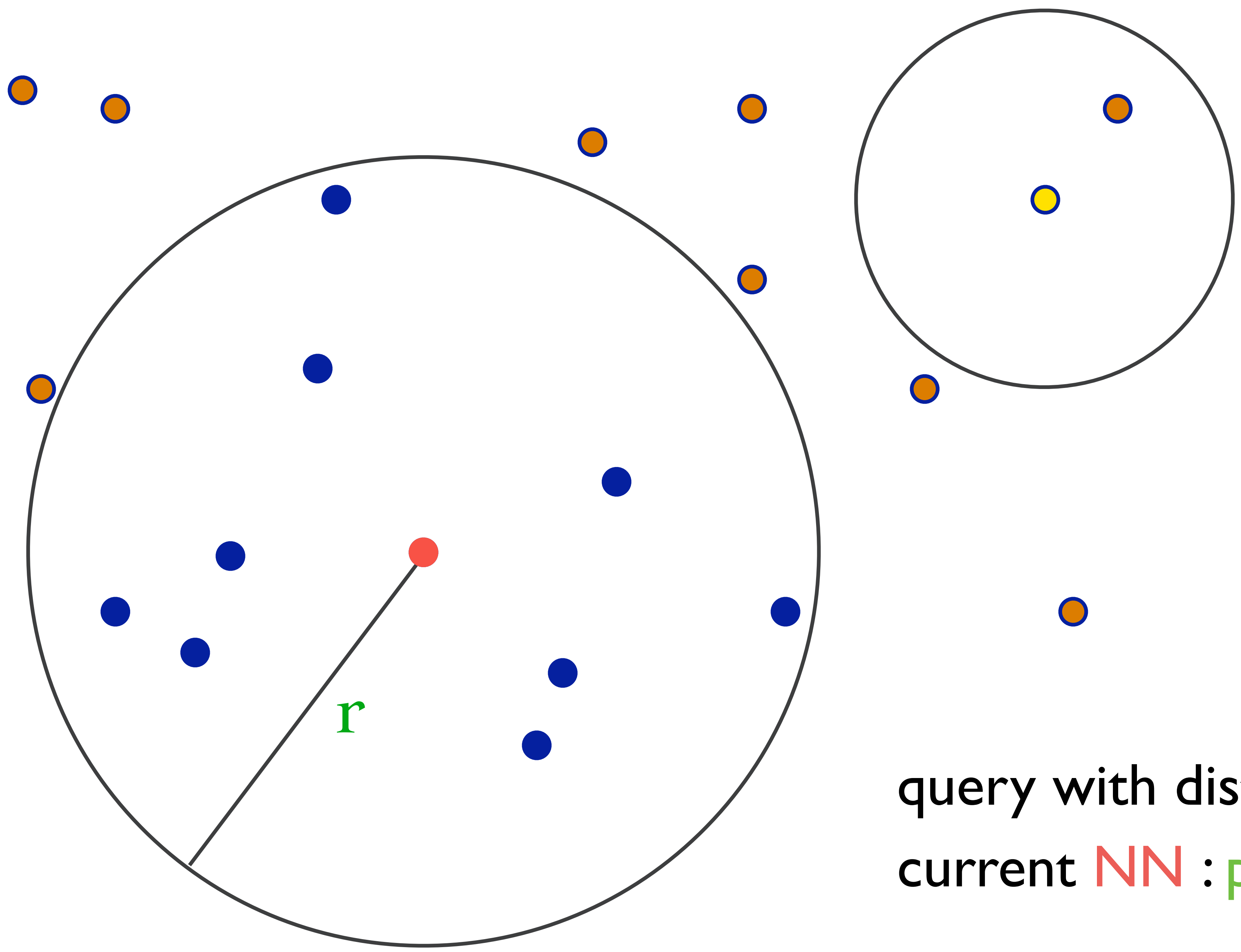


vantage-point algorithm



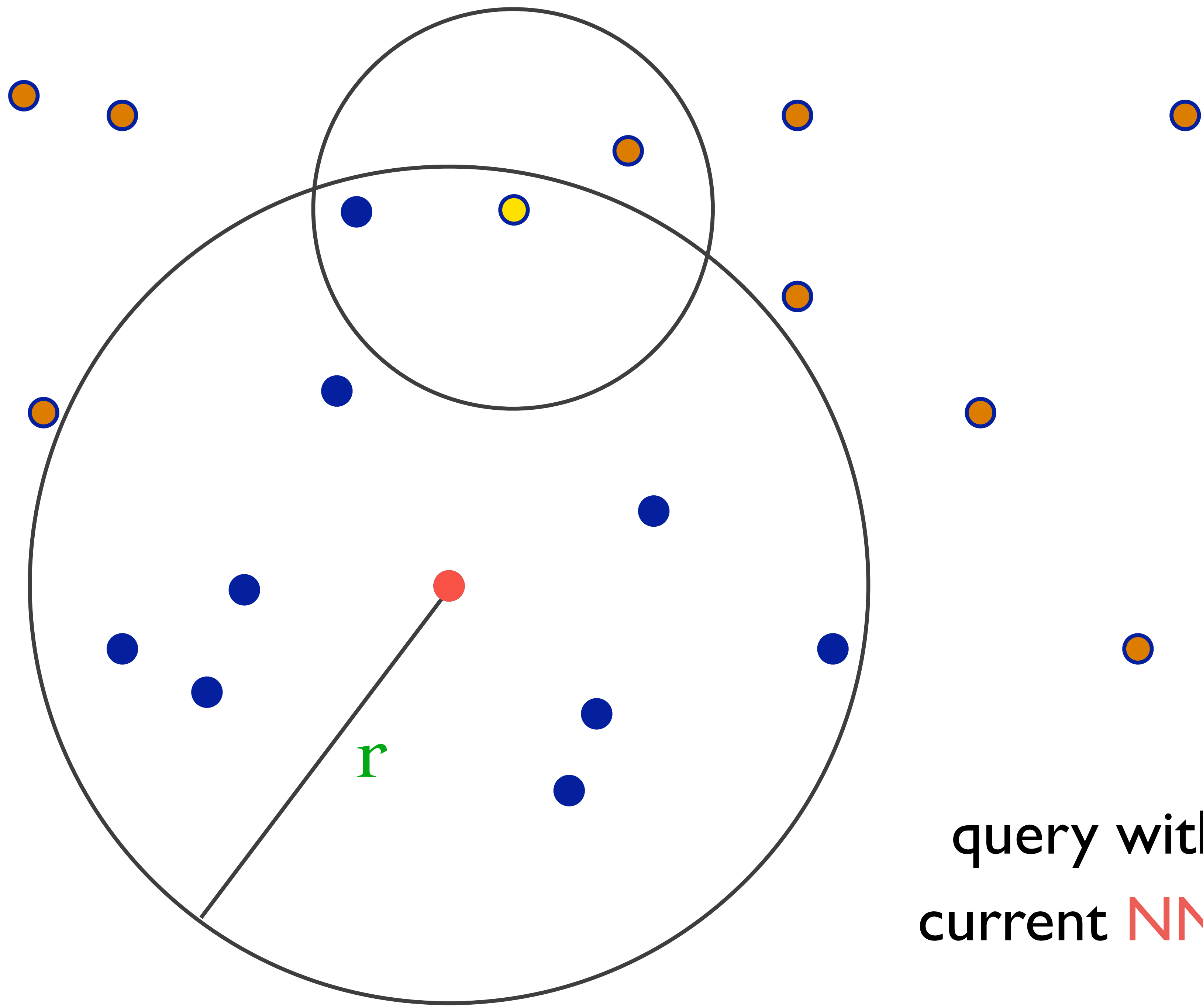
query with distance to current NN : pruning

vantage-point algorithm



query with distance to current NN : pruning

vantage-point algorithm



query with distance to current NN : NO pruning

similarity search in metric spaces

what are the pruning rules ?

can you see how the triangle inequality is used for the vantage-point pruning rules ?

problem in metric spaces becomes more difficult than in vector spaces

how to fight against the curse of dimensionality?

idea : approximations!

find approximate nearest neighbors

find approximately similar pairs

why does it make sense?

distance functions are proxies to human notion of similarity

approximate nearest neighbor

given a set X of objects (off-line)

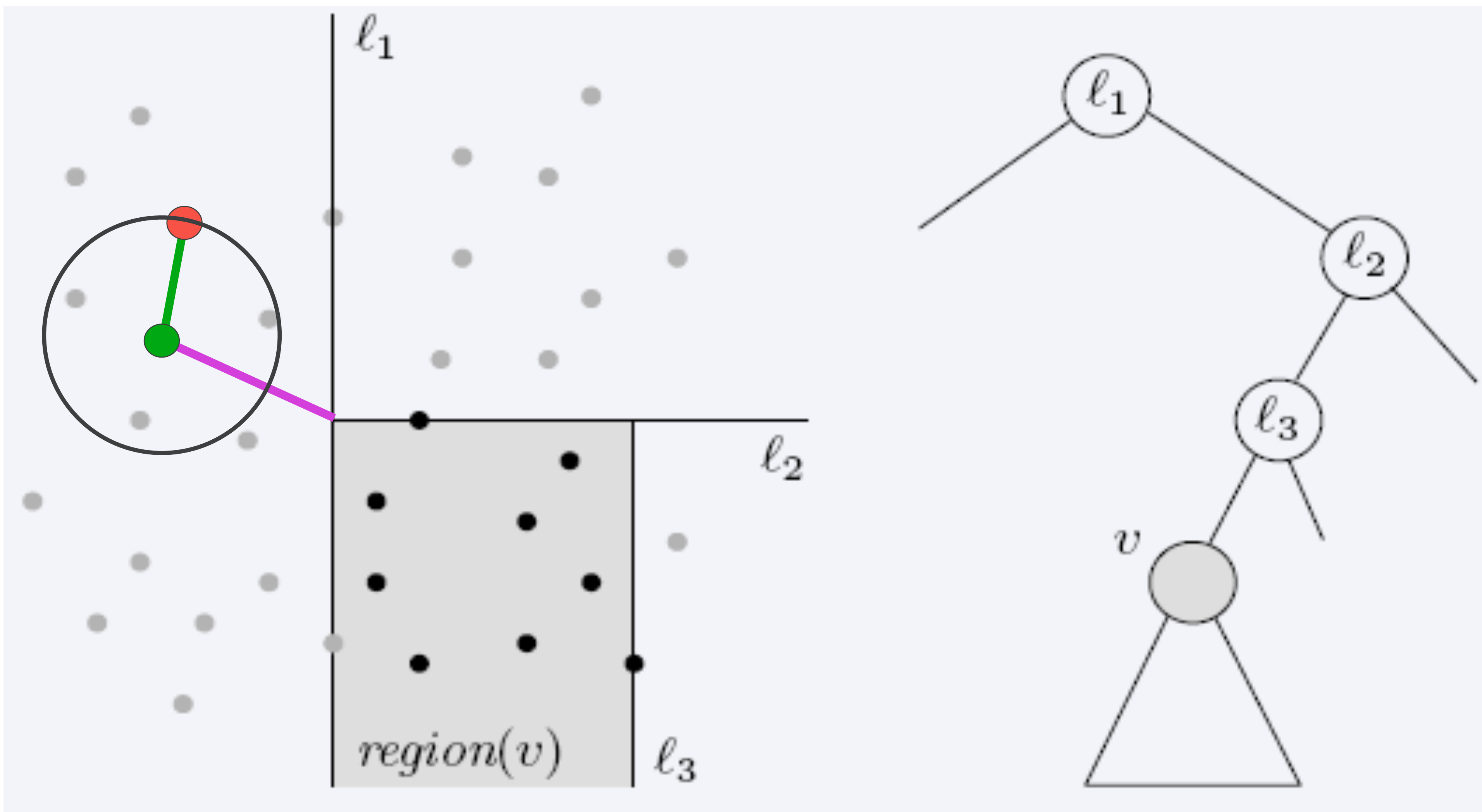
given accuracy parameter e (off-line or query time)

given a query object q (query time)

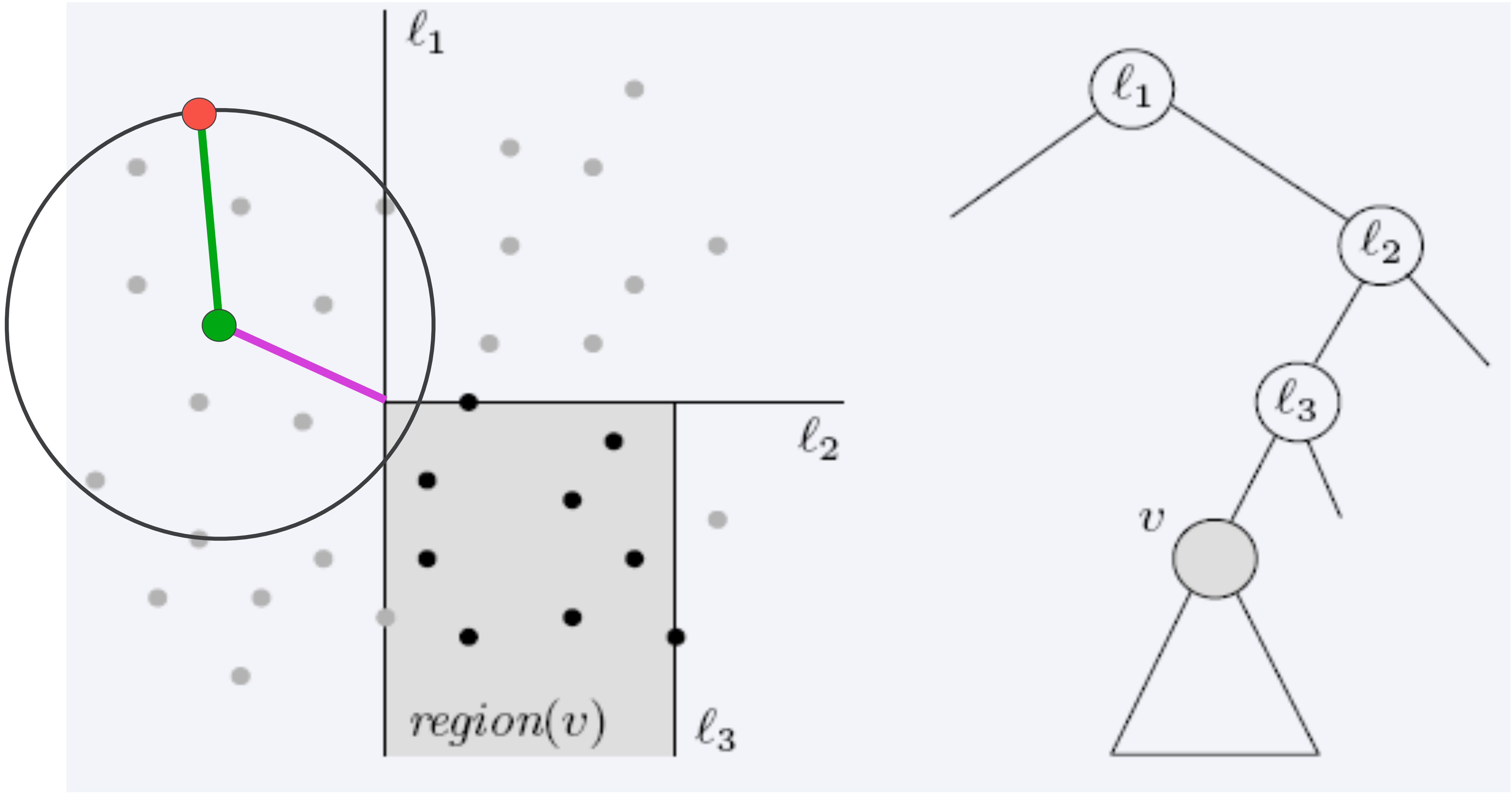
find an object z in X, such that

$$d(q, z) \leq (1 + e) d(q, x)$$ for all x in X

k-d trees for approximate similarity search

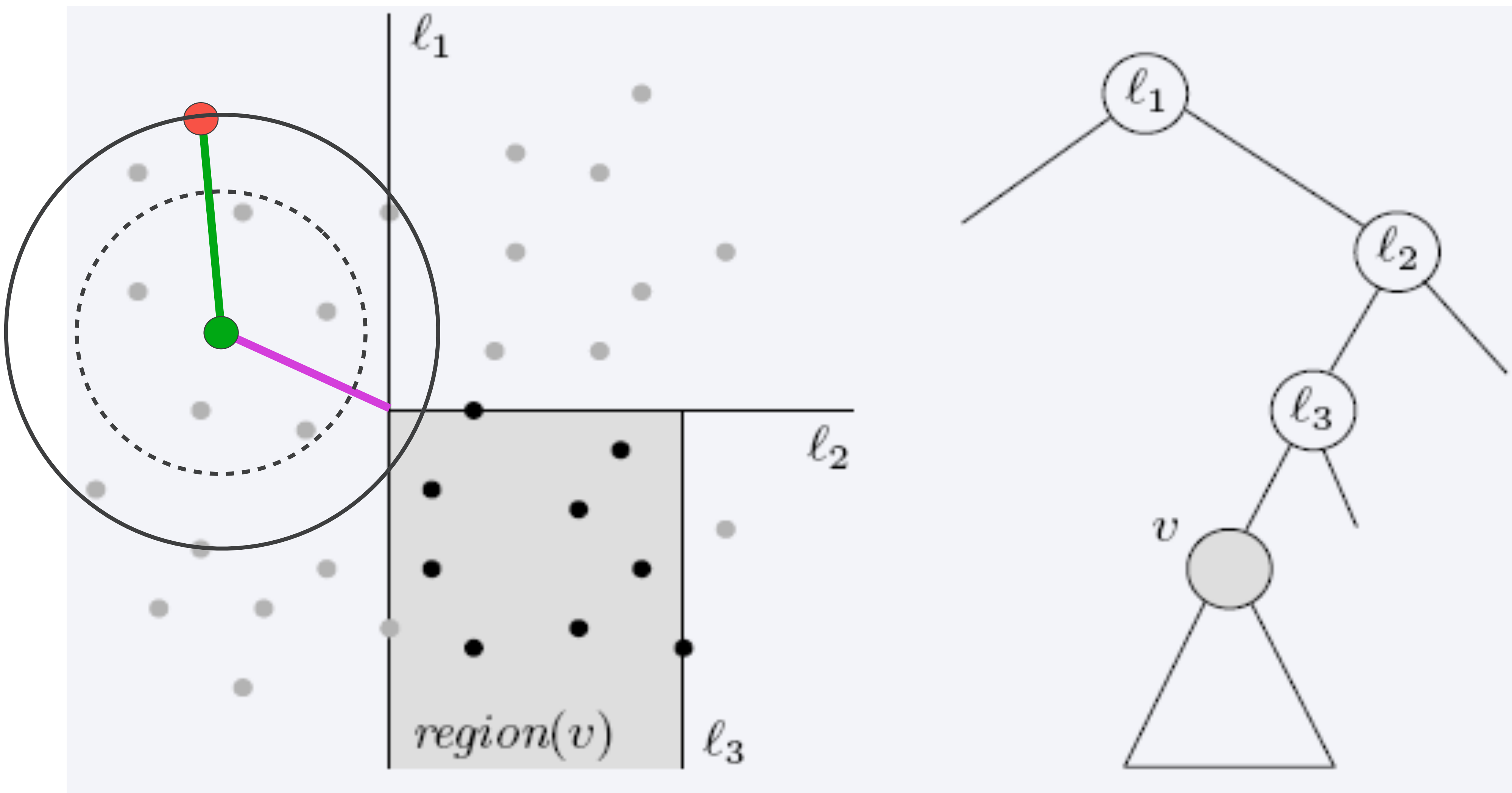


k-d trees for approximate similarity search



solid circle has radius $d(q, x)$

k-d trees for approximate similarity search



dashed circle has radius $d(q,x)/(1+e)$

next lecture

locality sensitive hashing