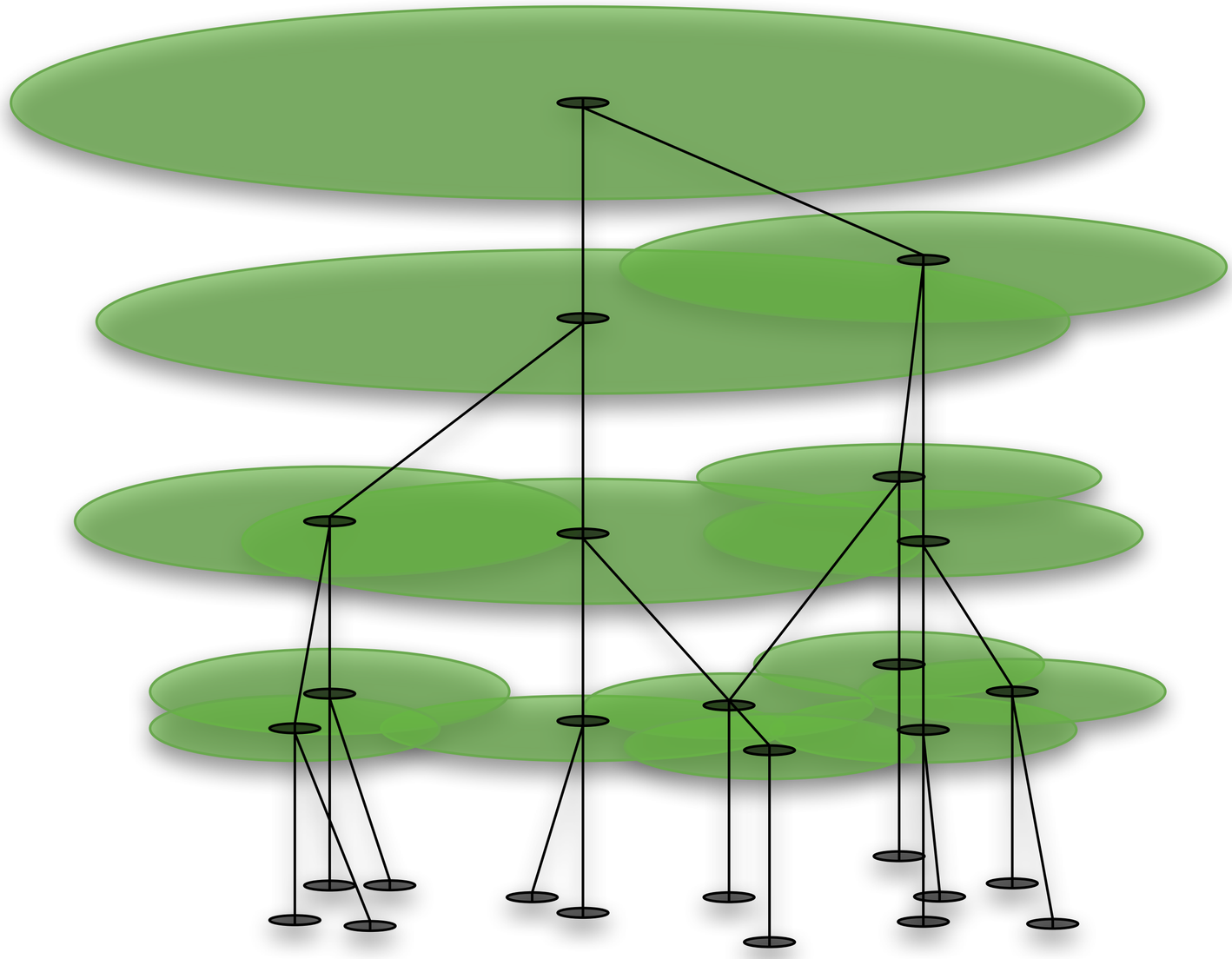
Proximity Search
In Doubling Metrics
Using the Greedy Tree



Oliver Chubet, May 28, 2024

Initial Questions

- **How do we do proximity search in Euclidean space?**
- **What goes wrong in general metric spaces?**
- **Can we build data structures for proximity search in any metric?**

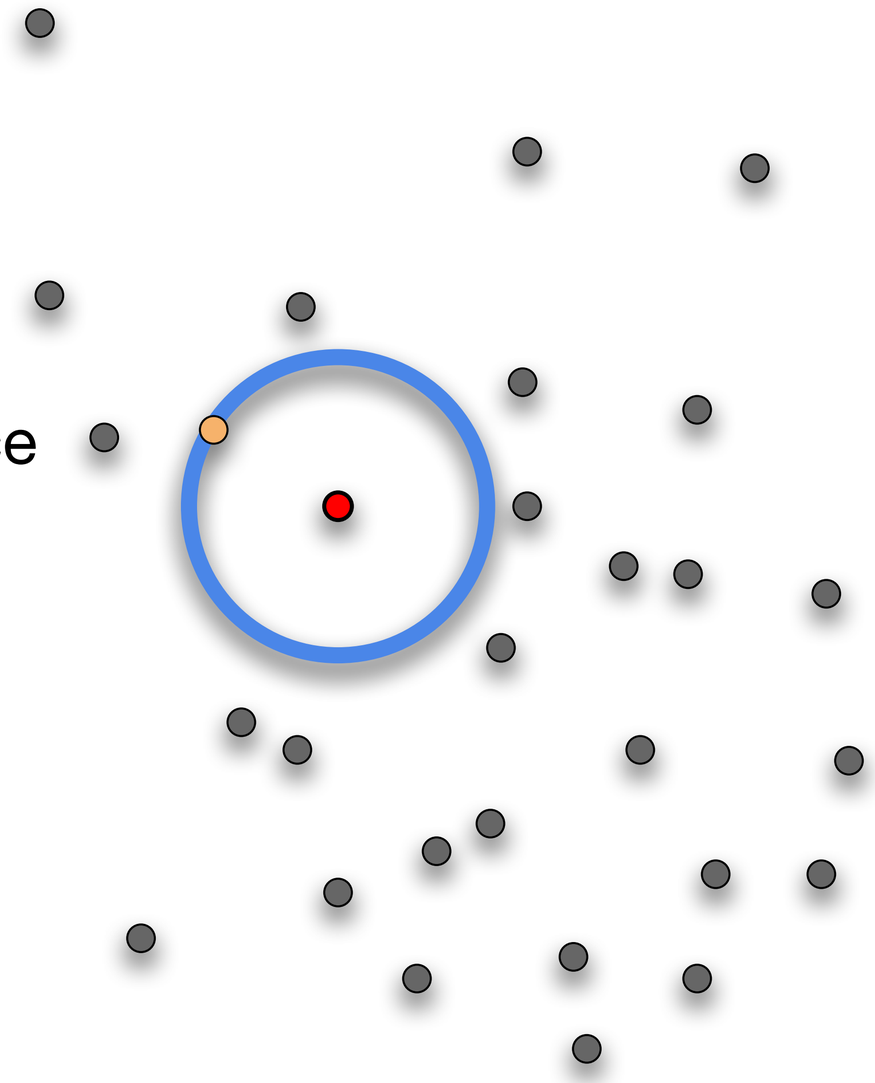
Proximity Search

Applications

- Pattern recognition
- Statistical classification
- Database information retrieval
- Maximum likelihood decoding in coding theory
- Spell checking
- Data compression
- Recommendation systems

Proximity Search

Problems



1. Nearest neighbor

Input: $P \subseteq X$ a set, where (X, d) is a metric space

Query: $q \in X$ a query point

Output: a point $p \in P$ such that

$$d(p, q) \leq d(p', q) \text{ for any } p' \in P$$

Proximity Search

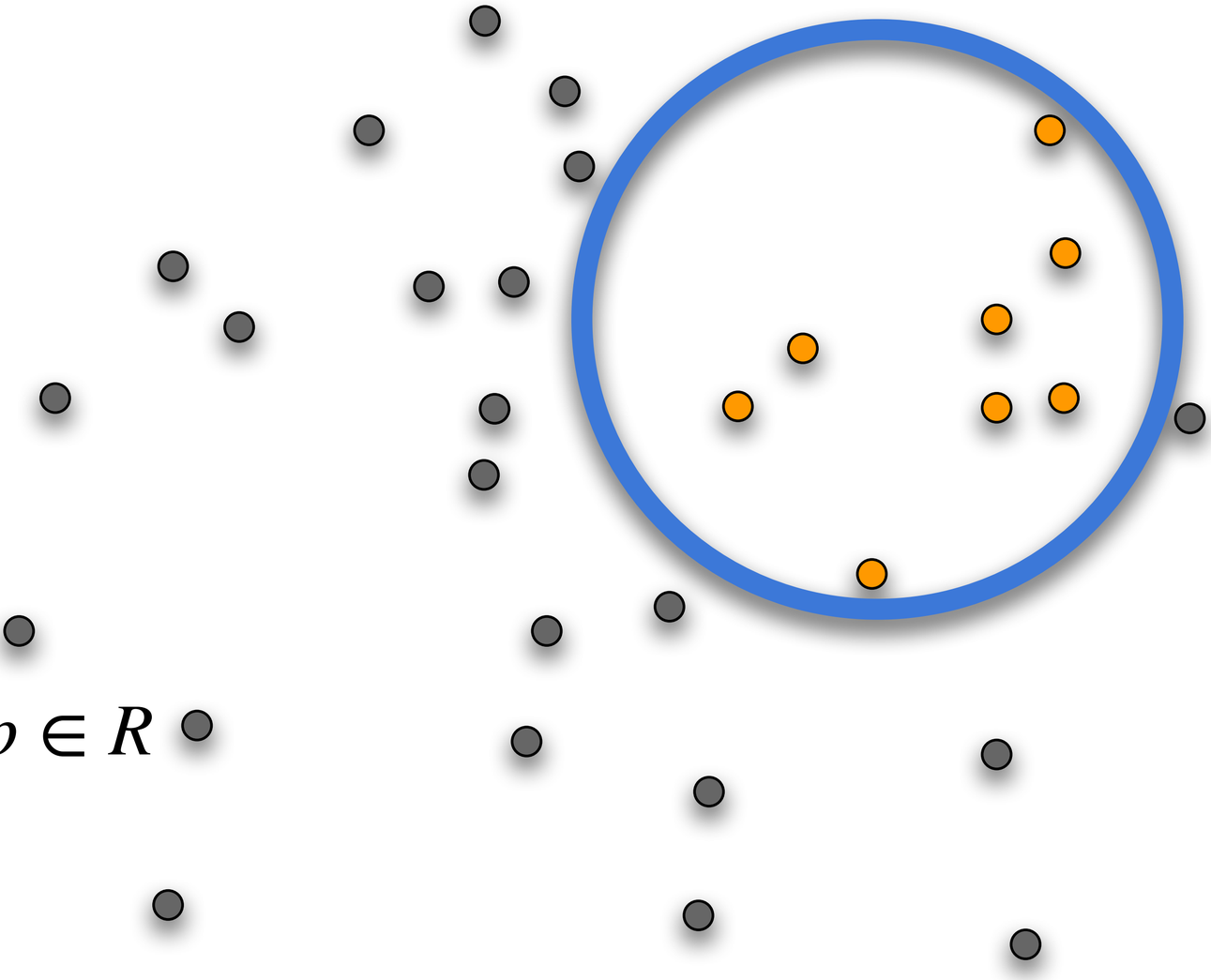
Problems

2. Range Search

Input: $P \subseteq \mathbb{R}^d$ a finite set

Query: $R \subseteq \mathbb{R}^d$ a query region

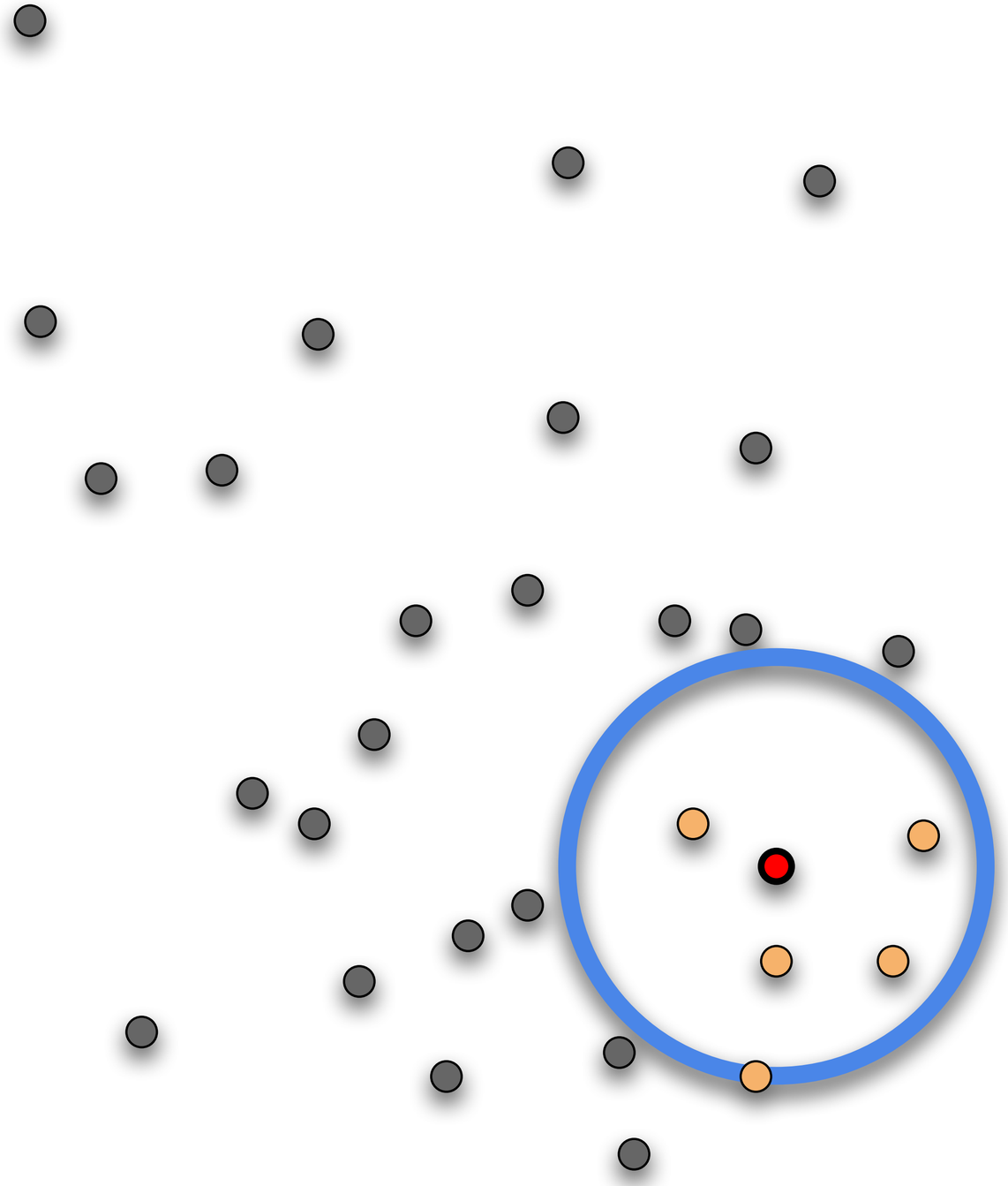
Output: all points $p \in P$ such that $p \in R$



Proximity Search

Other Problem Variations

- k-nearest neighbors
- Approximate nearest neighbor
- Range counting

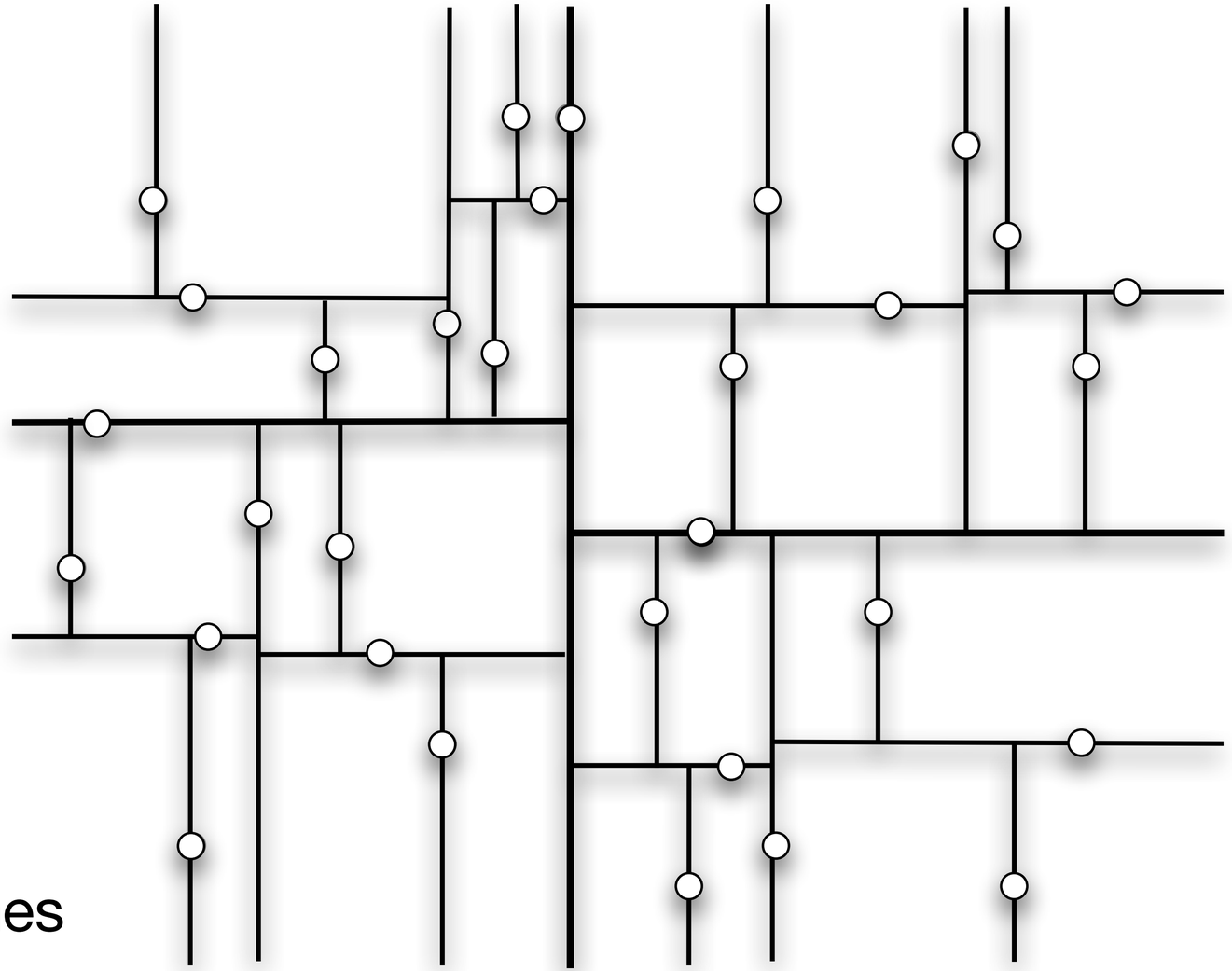


Proximity Search

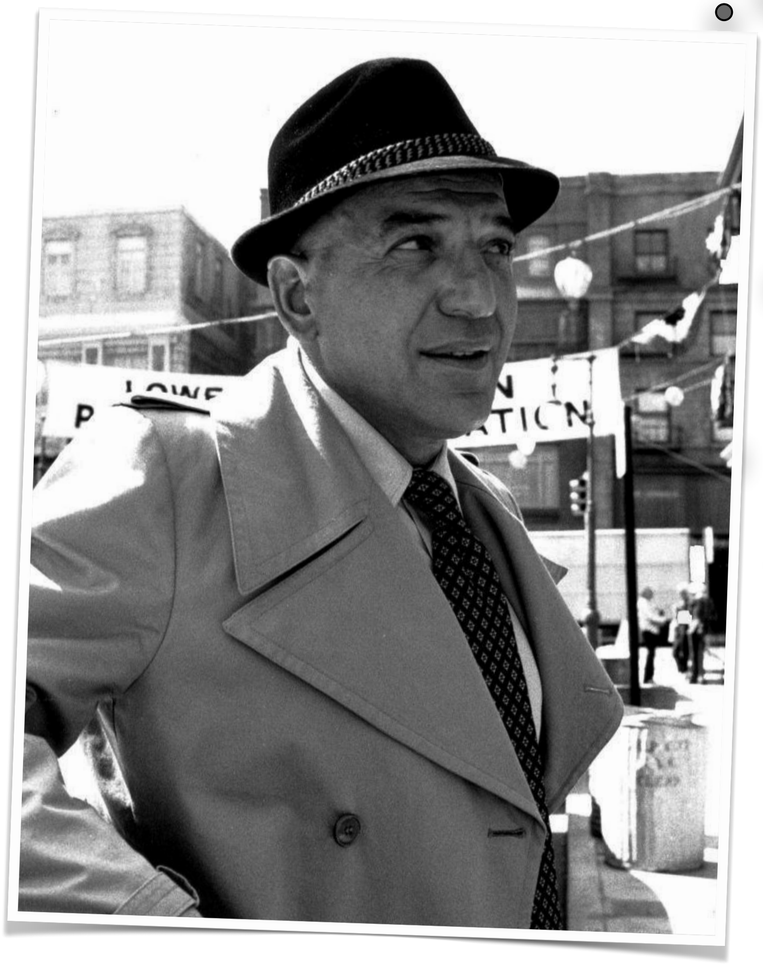
Data Structures for $\mathbb{R}^d$

- Hierarchical space partitioning
    - Quadtrees
    - kd-trees

Splitting a node into its children refines the scale of the search.



There are two ways they can search [for the murder weapon]: from the body outward in a spiral, or divide the room up into squares--that's the grid method.[1]

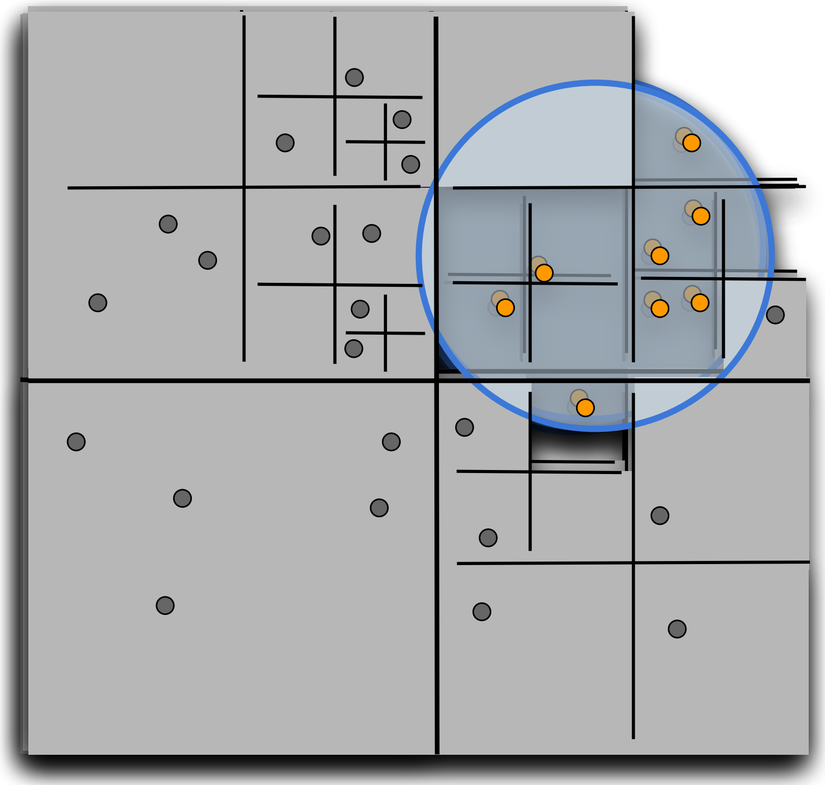


[1] From the CBS series *Kojak*, "Death Is Not a Passing Grade".

[Bentley, Friedman, Maurer 1978]

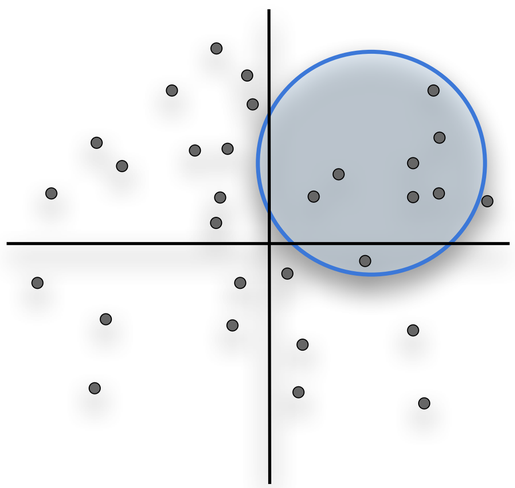
Range Search

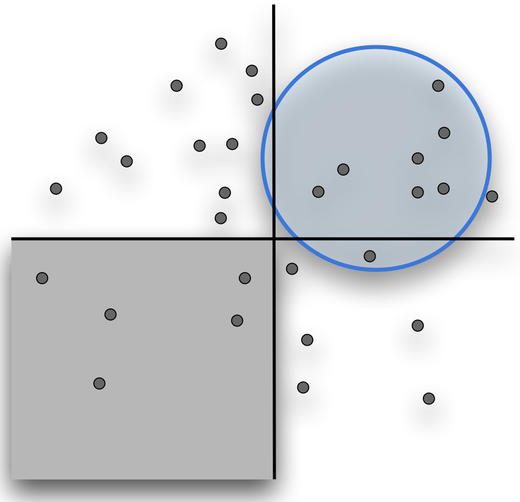
Using a quad tree

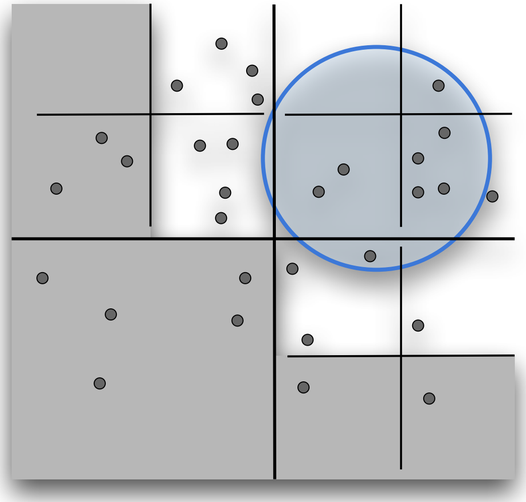


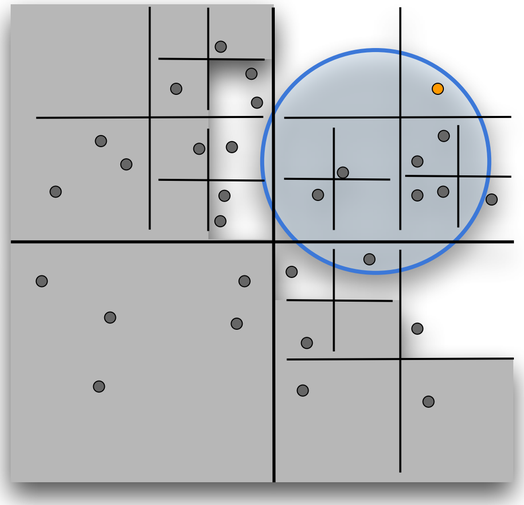
Range Search

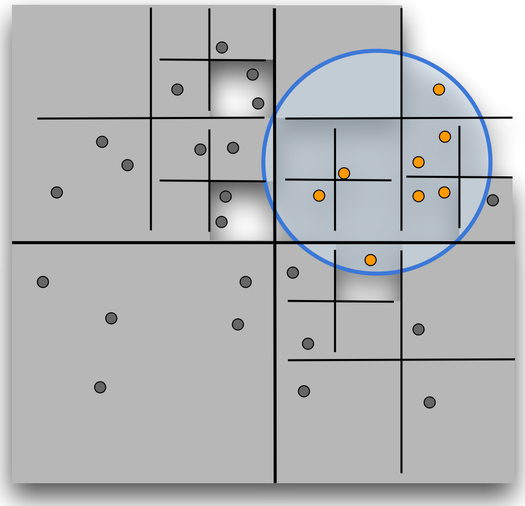
Using a quad tree

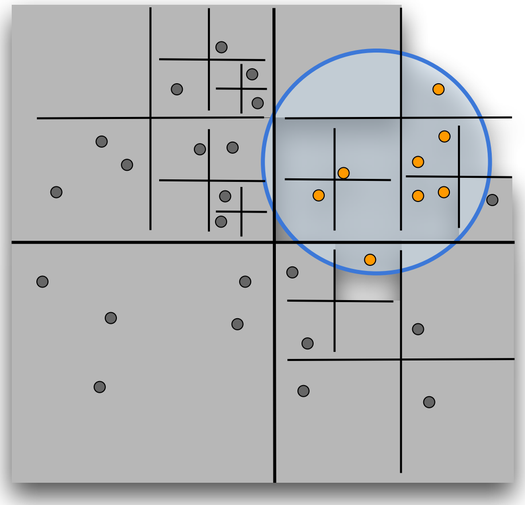












What can go wrong?

- **High dimensional spaces**
- **Metrics without coordinates**

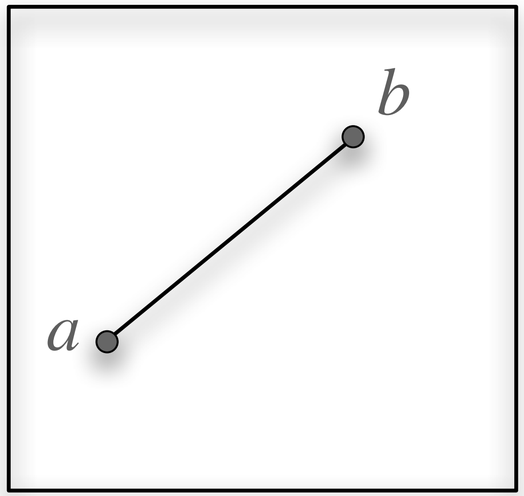
General Metric Spaces

Definition

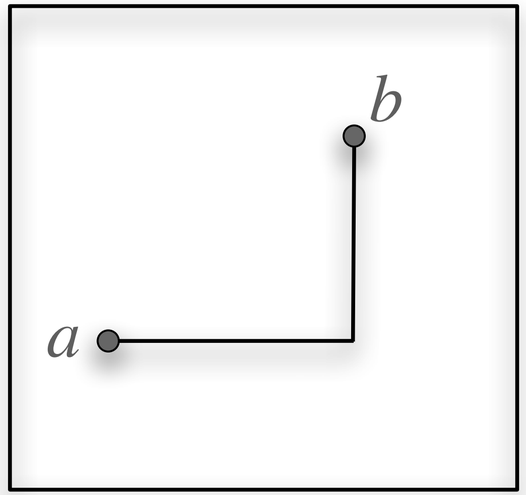
A **metric space** is a set X with a function $d : X \times X \to \mathbb{R}$ that has the following properties:
For any $x, y, z \in X$,

1. **Non-negative:** $d(x, y) \geq 0$
2. $d(x, y) = 0$ **iff** $x = y$
3. **Symmetric:** $d(x, y) = d(y, x)$
4. **Triangle Inequality:** $d(x, z) \leq d(x, y) + d(y, z)$

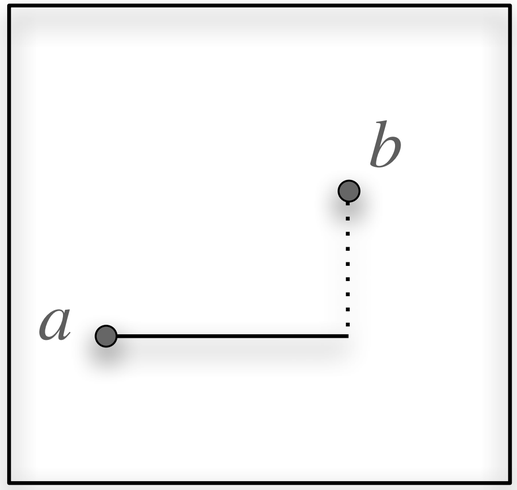
General Metric Examples



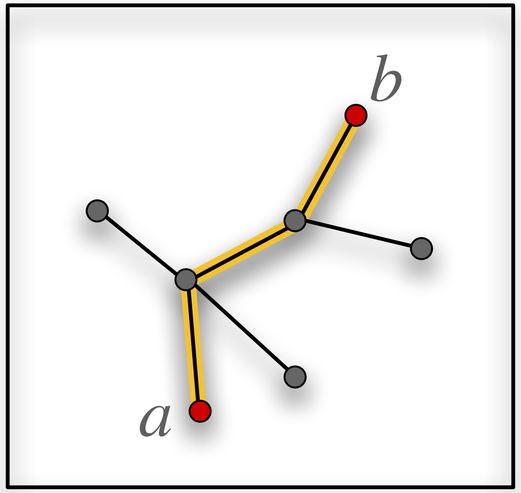
$$d(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$$



$$d(a,b) = |a_x - b_x| + |a_y - b_y|$$



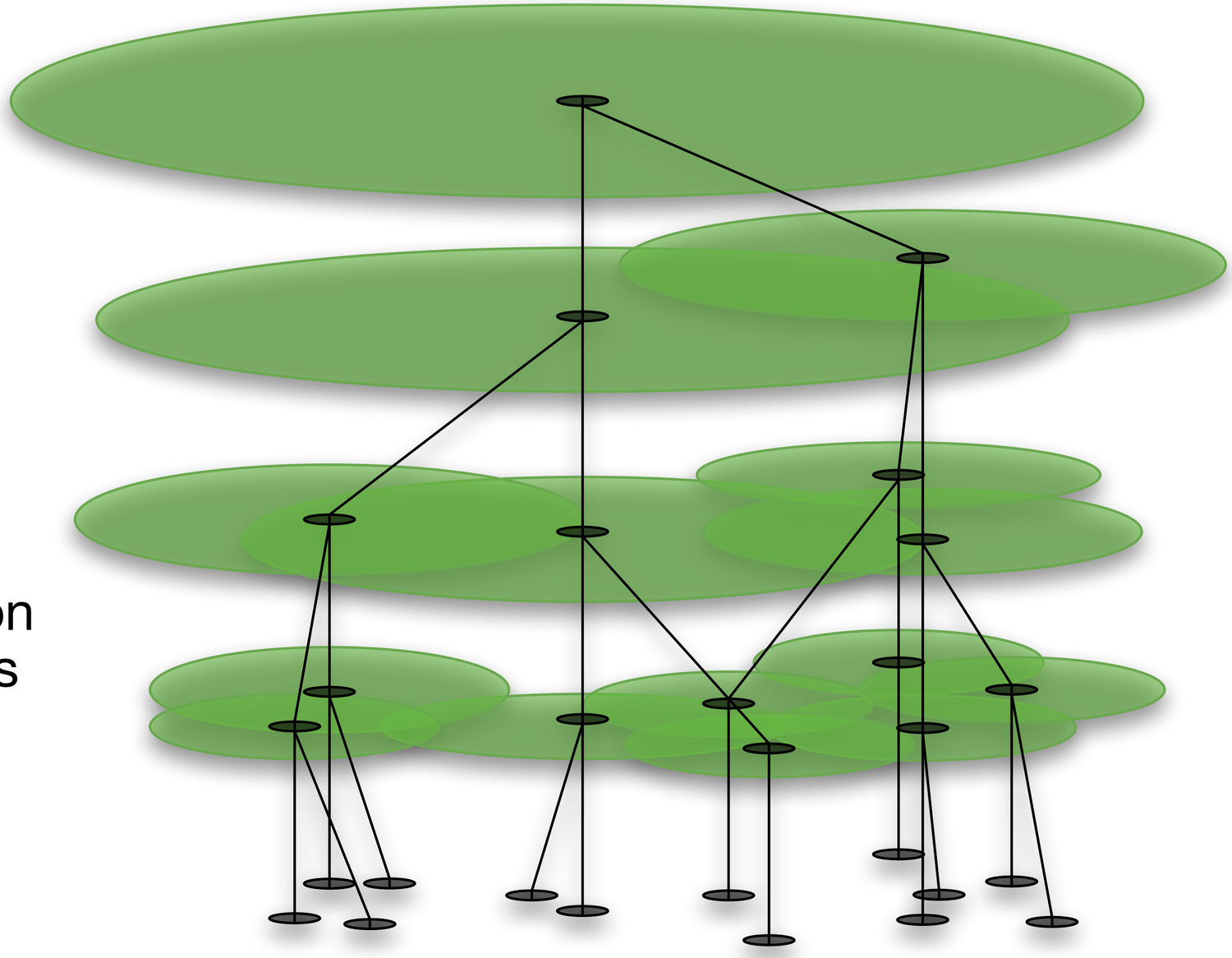
$$d(a,b) = \max\{|a_x - b_x|, |a_y - b_y|\}$$



Length of the shortest path

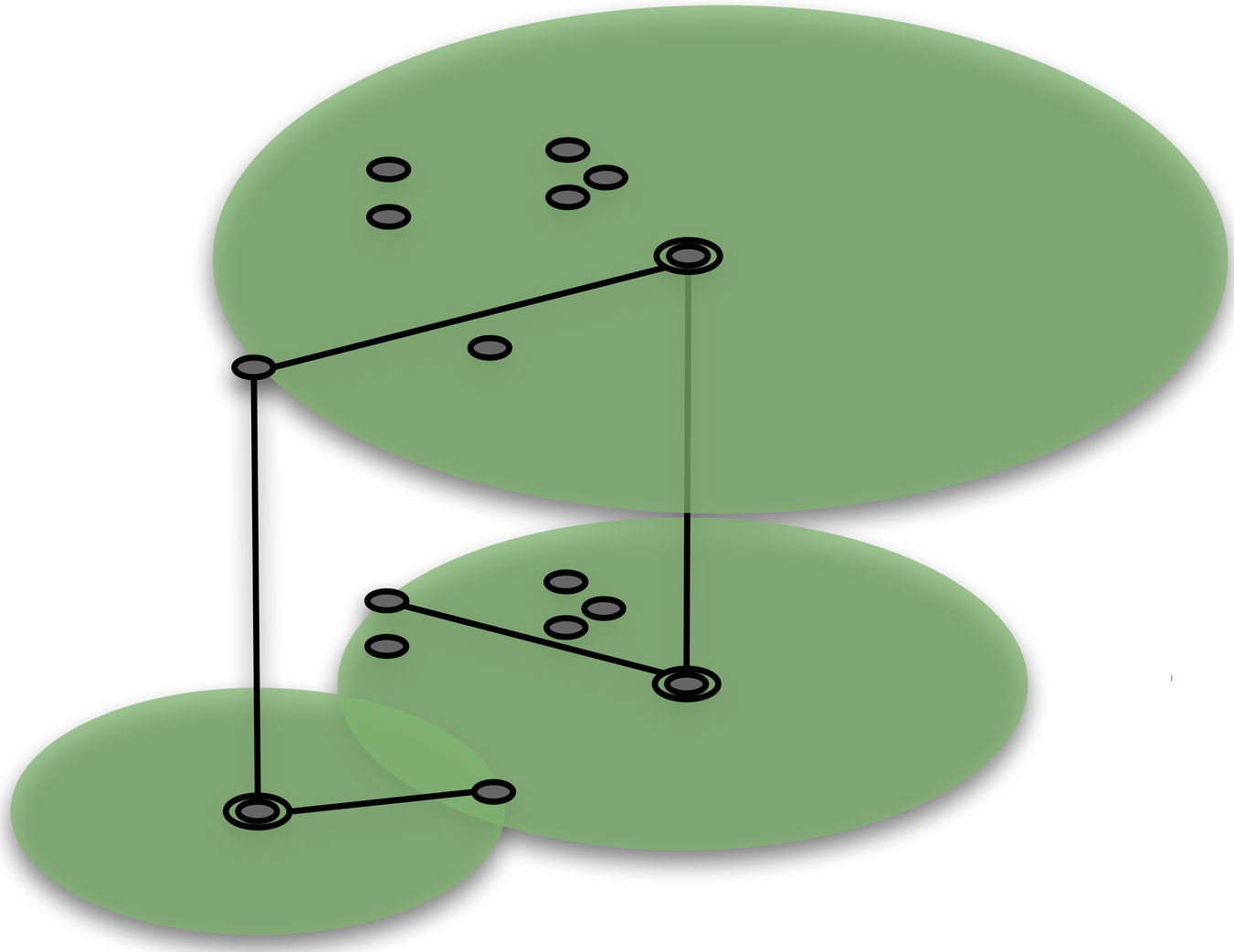
Ball Trees

- A binary tree
- Constructed by recursive partitioning of the data set
- Each node covers its partition (the points at the leaves in its subtree) with a ball



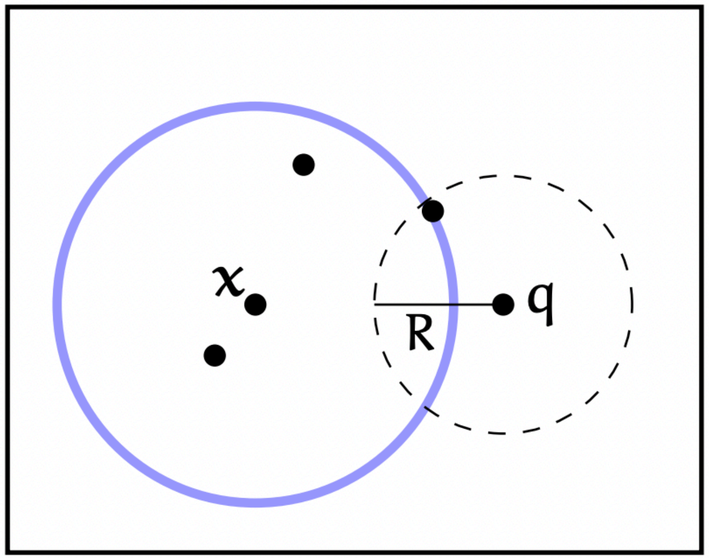
Split a node

- A node covers all the points in the leaves of its subtree.
- The root covers everything.
- The children partition the points of their parent.
- Children have smaller radii than their parent.

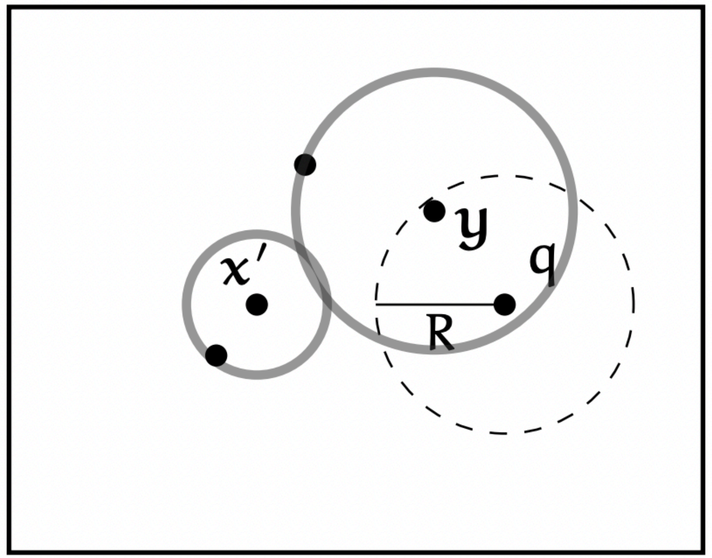


Range Search

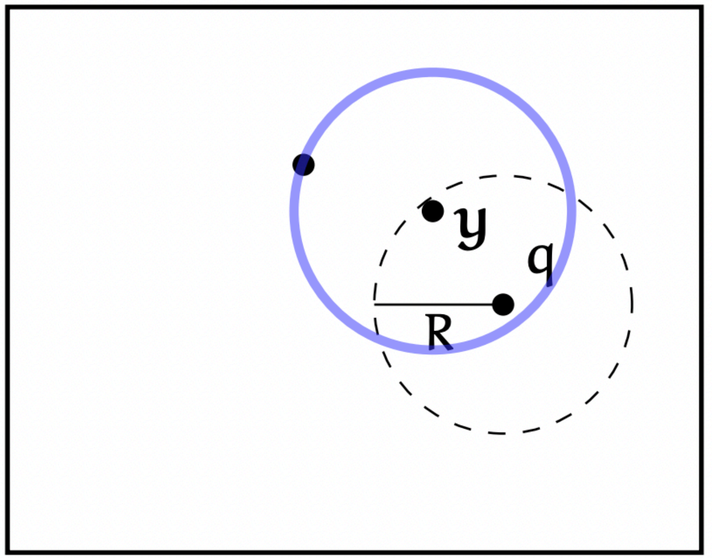
In a ball tree



Viable



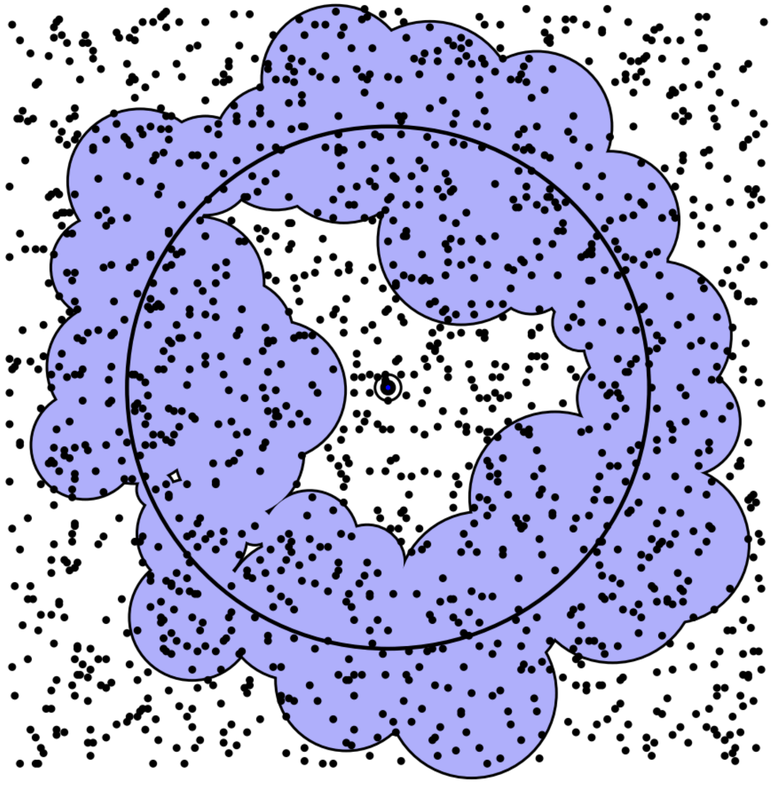
Split

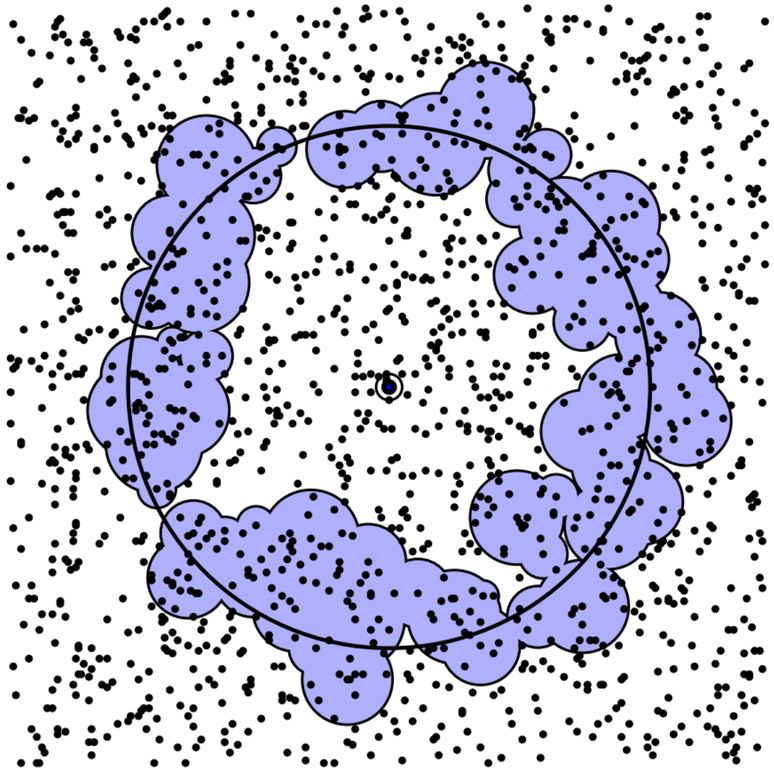


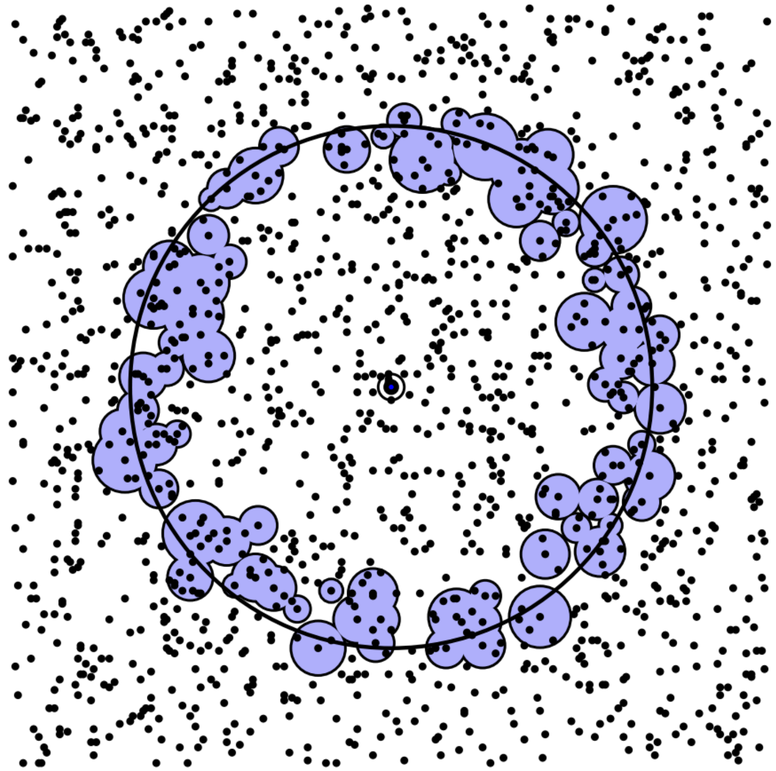
Viable

Range Search

In a ball tree

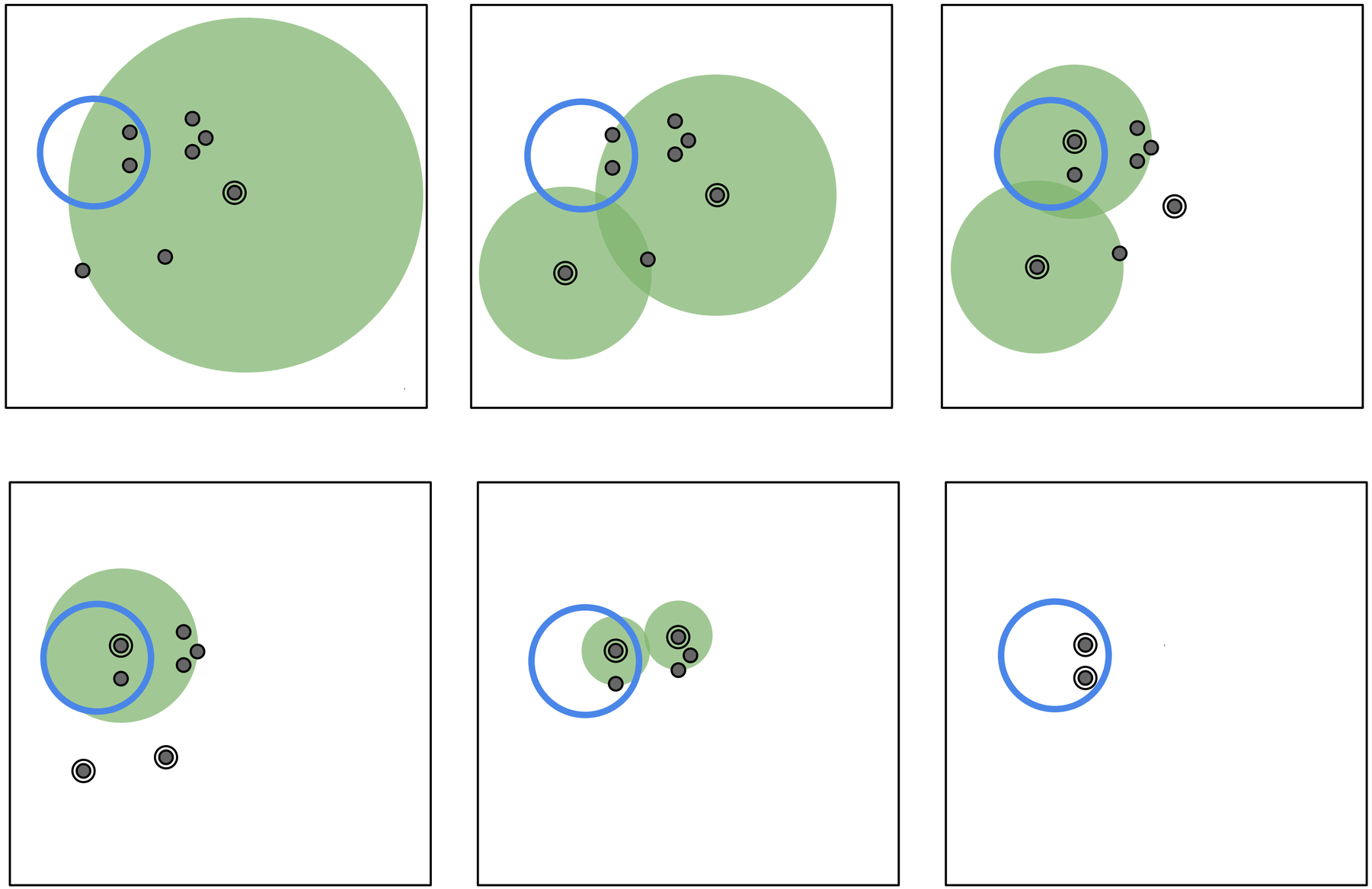






Nodes completely contained within in the range can be added to the output.

Range Search



Generic Search in a Ball Tree

Store the root in a max-heap H, keyed by radius

While H is not empty:

1. Remove and split the max node
2. If possible, add them to the output
3. Otherwise, check the viability of each (discard or put on the heap)

Analyzing Proximity Search in a Ball Tree

- Running time is $O((\#\ iterations) \times (time\ for\ a\ heap\ operation))$
- $w = search\ width$, (the max cardinality of the heap)
- $h = height\ of\ the\ tree.$
- $(\#\ iterations)$ is $O(wh)$
- $(time\ for\ a\ heap\ operation)$ is $O(\log w)$

Bound running time by bounding the $search\ width$ **and** $height$

Guarantees?

Ball trees *in general* do not give strong guarantees for search times.

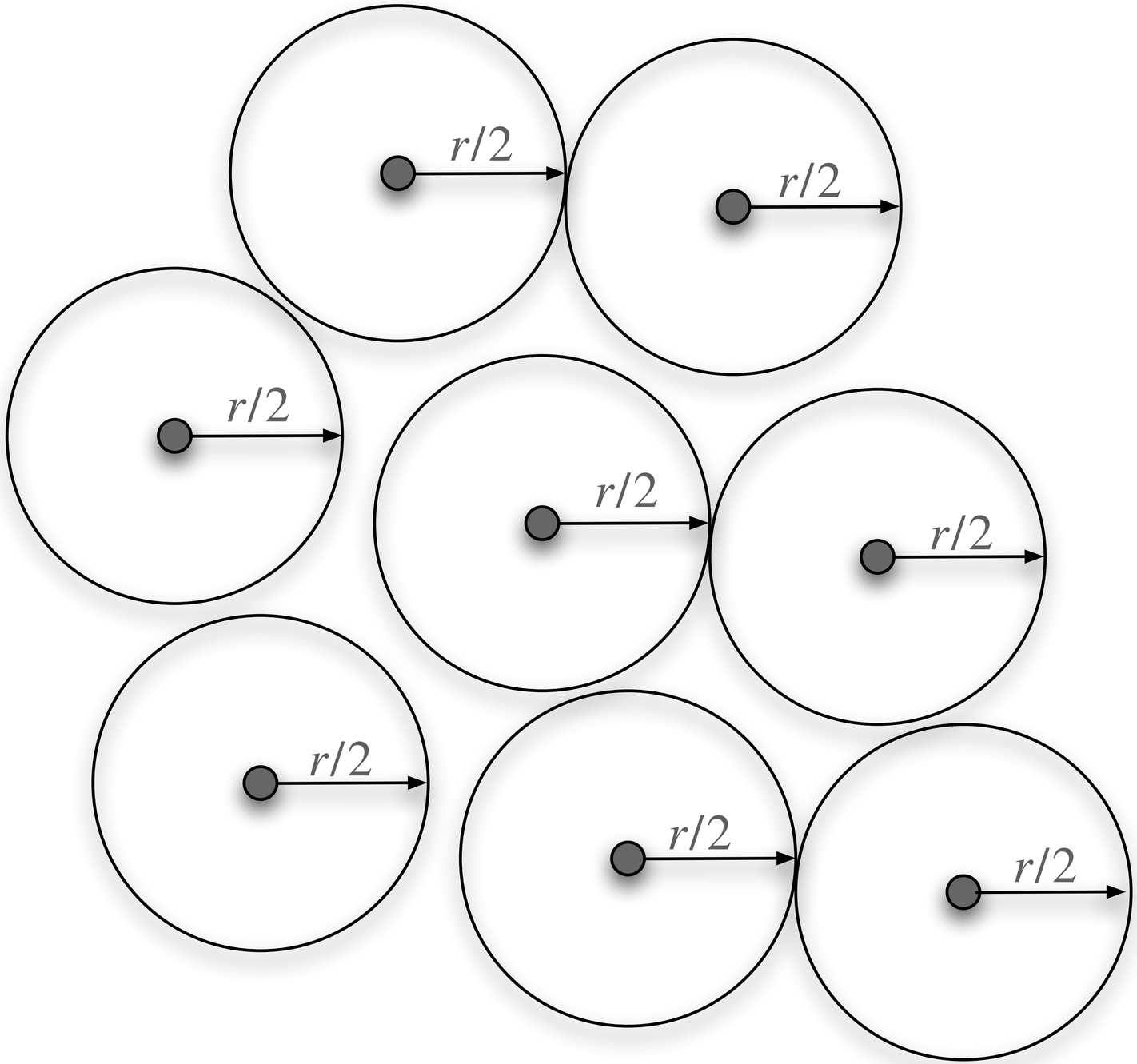
Structural properties to bound search times:

1. The height of the tree can be bounded by scaling node radii by a constant factor per level.

2. Search width can be bounded by upper bounding the number of nodes of similar radii.

Packing

A set is r-packed if
$$d(a, b) \geq r$$
for any distinct a and b.

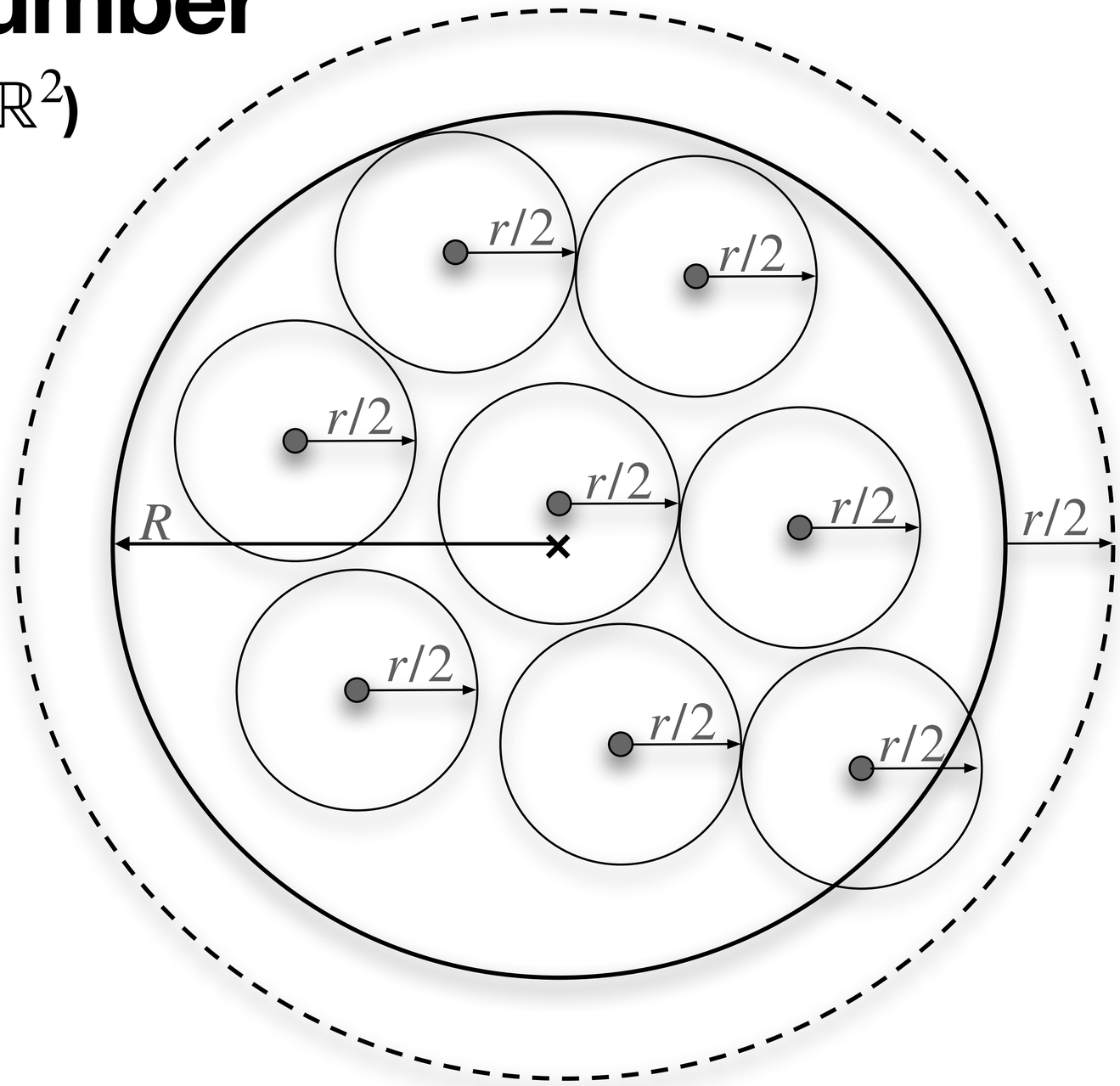


Upper bound the number of packed nodes (in $\mathbb{R}^2$)

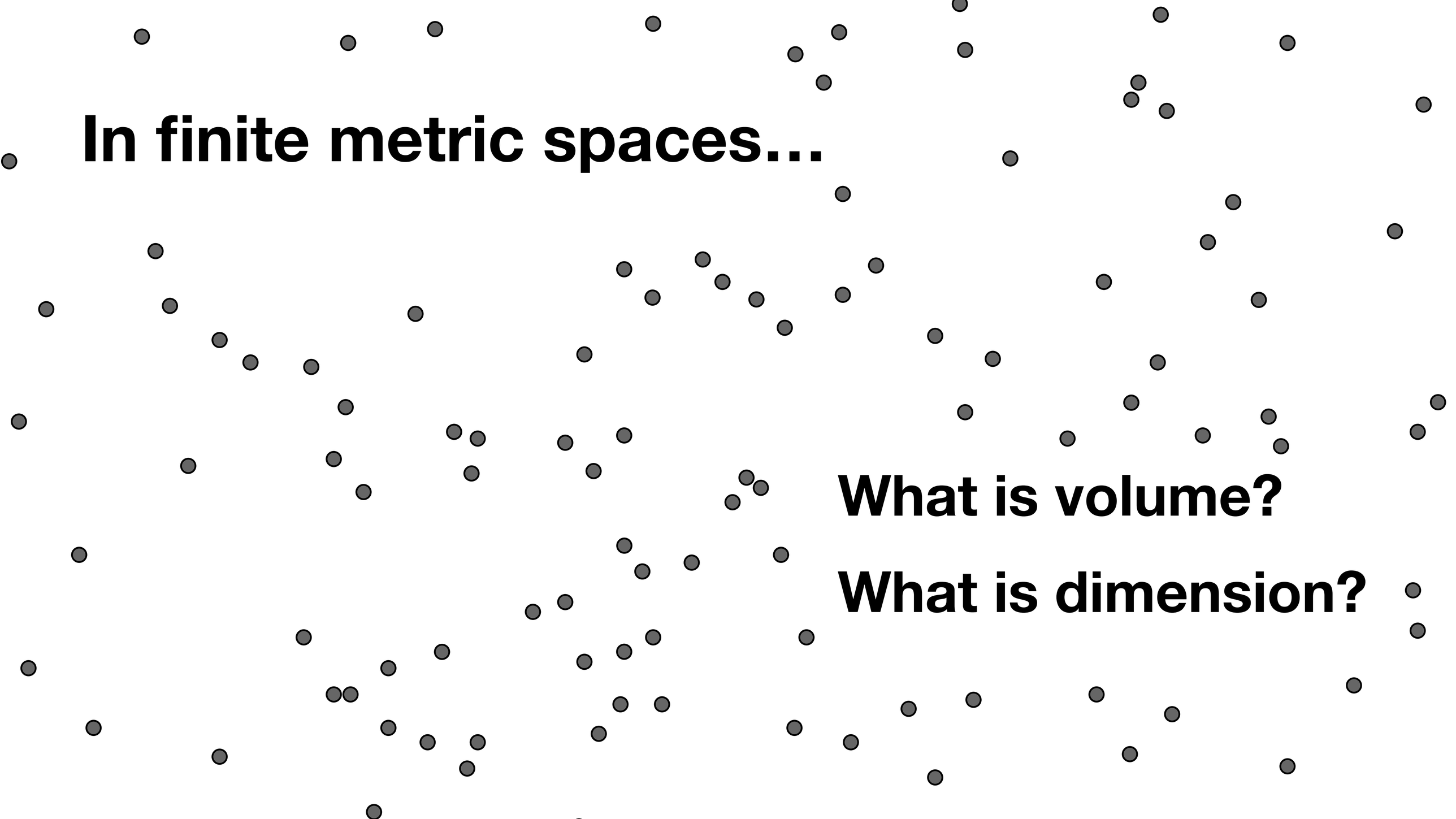
The set H is

1. **r-packed and**
2. **covered by a ball with radius R**

$$|H| \leq \frac{\pi\left(R+\frac{r}{2}\right)^2}{\pi\left(\frac{r}{2}\right)^2} = \left(\frac{2R+r}{r}\right)^2$$



In finite metric spaces…



What is volume?

What is dimension?

Doubling Dimension

Let X be a metric space.

The **doubling constant** of X is the minimum integer ρ such that any ball $B \subseteq X$ of radius r, can be covered with ρ balls of radius $r/2$.

The **doubling dimension** is $\log \rho$.

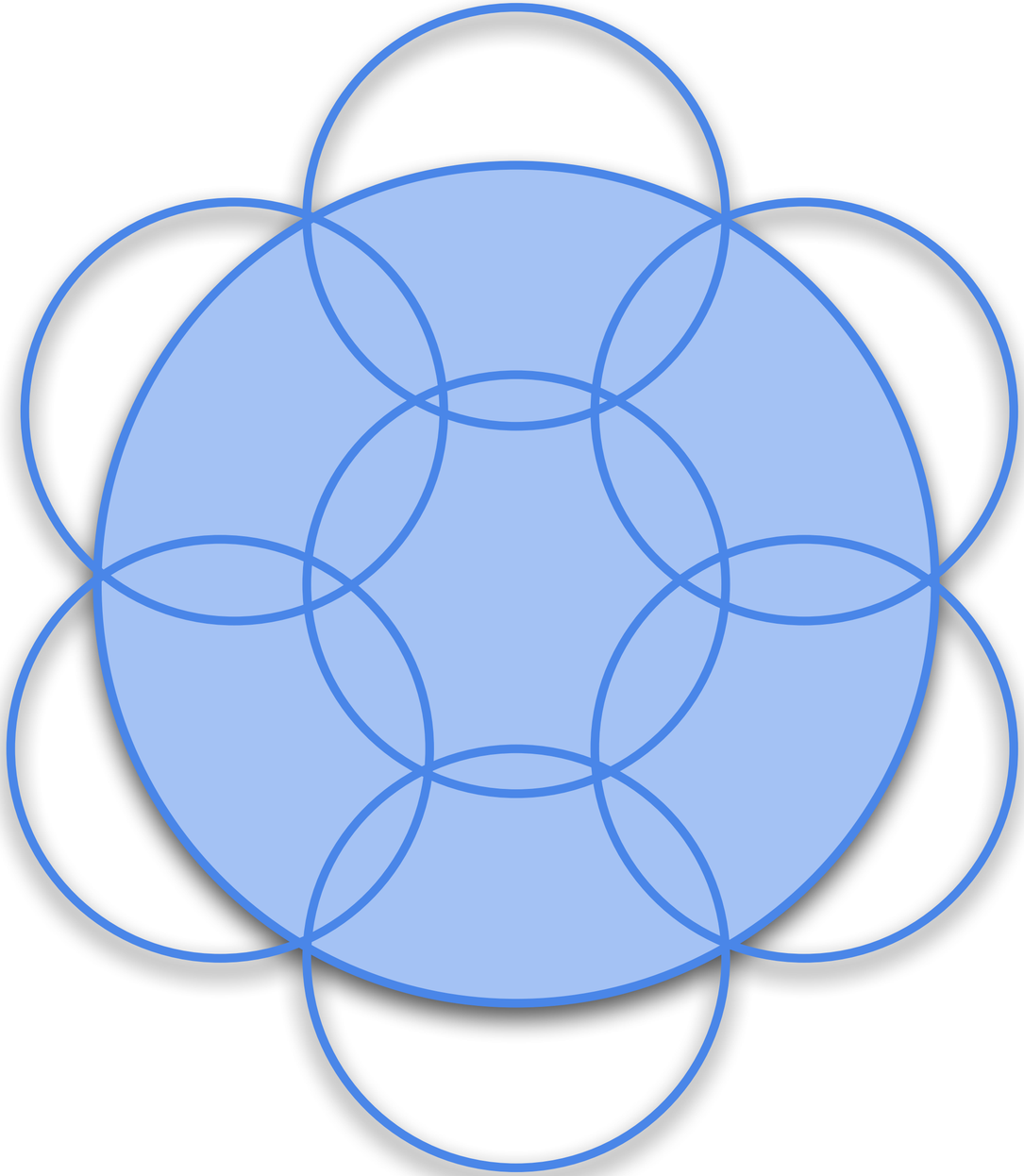
Doubling Dimension

Example: Euclidean

$\mathbb{R}^2$ **with metric**

$$d(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$$

- Any ball of radius r can be covered with 7 balls of radius $r/2$



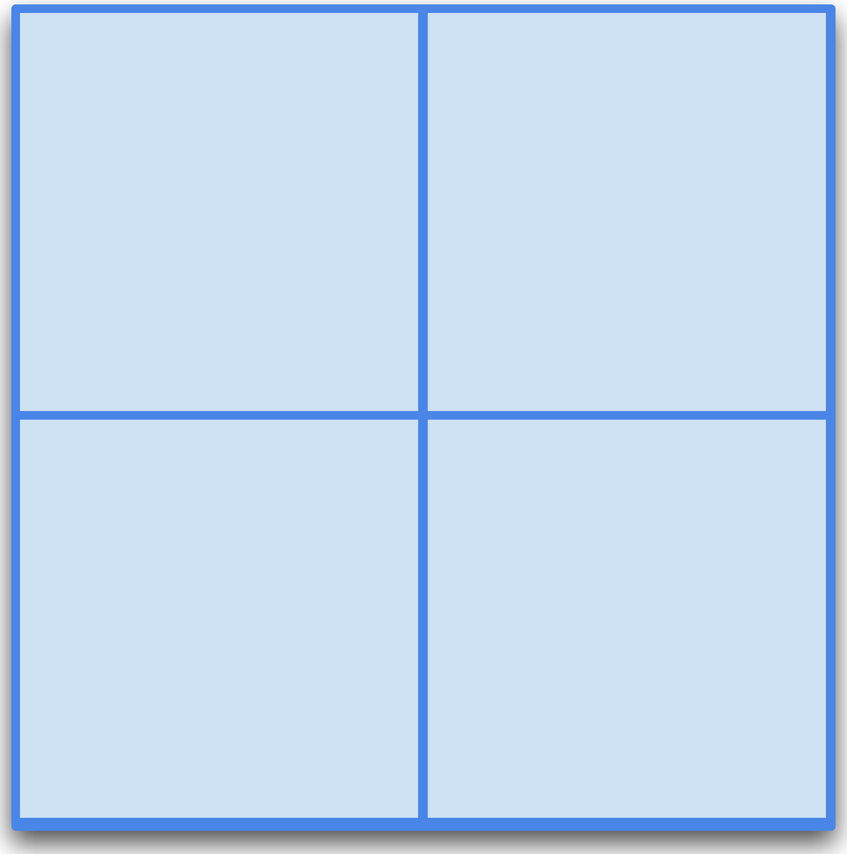
Doubling Dimension

Example: L-infinity

$\mathbb{R}^2$ **with metric**

$$d(a,b) = \max\{\,|a_x - b_x|\,,\,|a_y - b_y|\,\}$$

- Any ball of radius r can be covered with 4 balls of radius $r/2$



Doubling Dimension
Intuition check

Let X be a finite set and $d : X \times X \to \mathbb{R}$ the uniform metric, where $d(x, y) = 1$ for any $x \neq y$ and let $n = |X|$

Q: What is the doubling dimension of X?

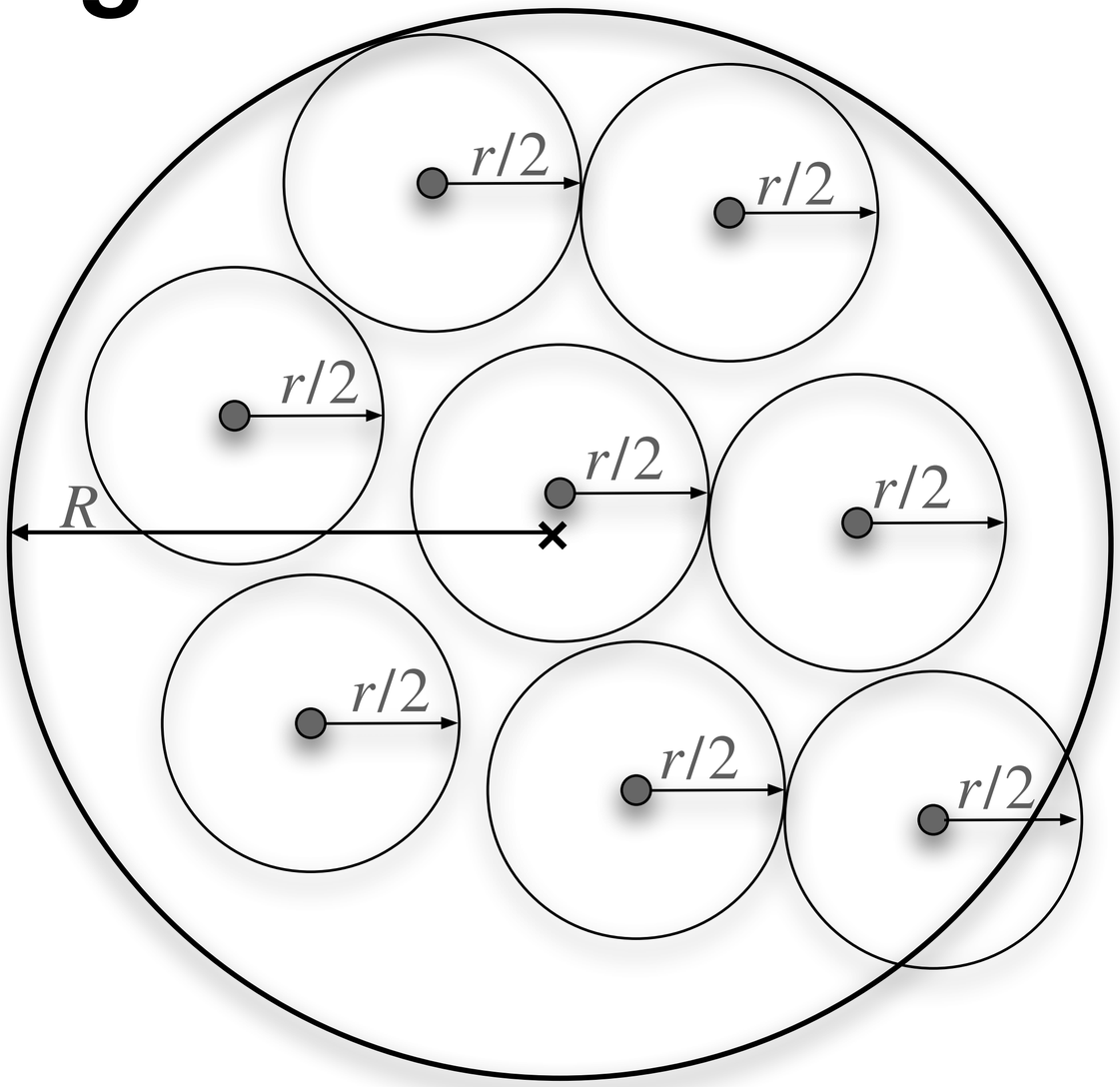
Is it large or small?

Packing Arguments using Doubling Dimension

The set X is

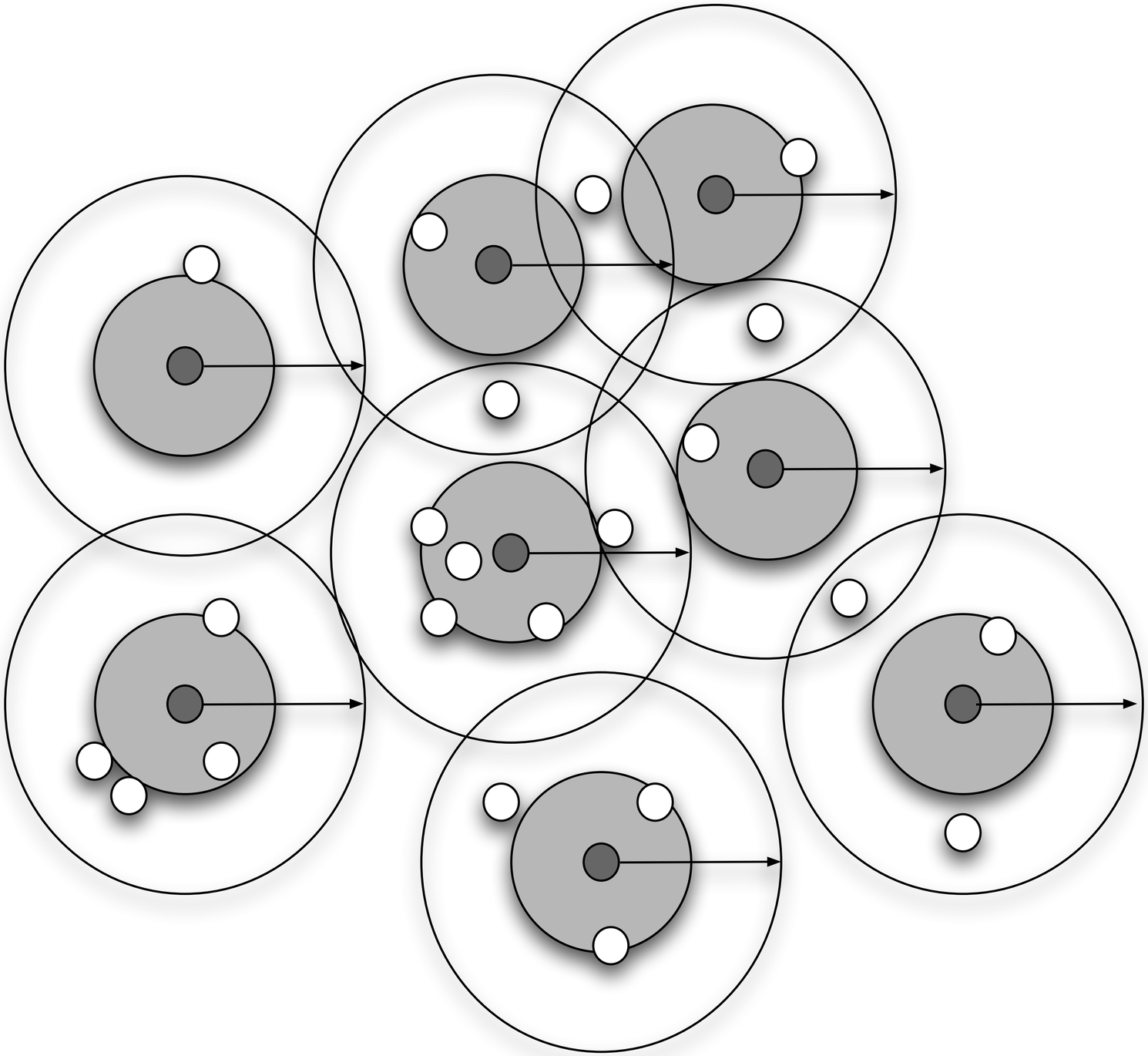
1. **r-packed and**
2. **covered by a ball with radius R**

$$|X| \leq \left(\frac{4R}{r}\right)^d$$



Nets

A Packing and Covering



Net-trees

"The net-tree can be thought of as a representation of nets from all scales"

- $2^{O(d)} n \log n$ **expected randomized pre-processing**
- $O(n \log n)$ **space data structure**
- $2^{O(d)} \log n + (1/\varepsilon)^{O(d)}$ **query time**
- **Built using a farthest-point traversal of the data set**
  (a.k.a. a greedy permutation or Gonzalez ordering)

[Har-Peled and Mendel 2005]

Goals

1. **Construct a ball tree where nodes of similar radii are packed**

2. **Match net-tree construction times using a deterministic algorithm and linear space**

We start by using a similar strategy of using the greedy permutation.

Greedy Permutations

Let $X = (x_0, x_1, x_2, \ldots, x_{n-1})$.

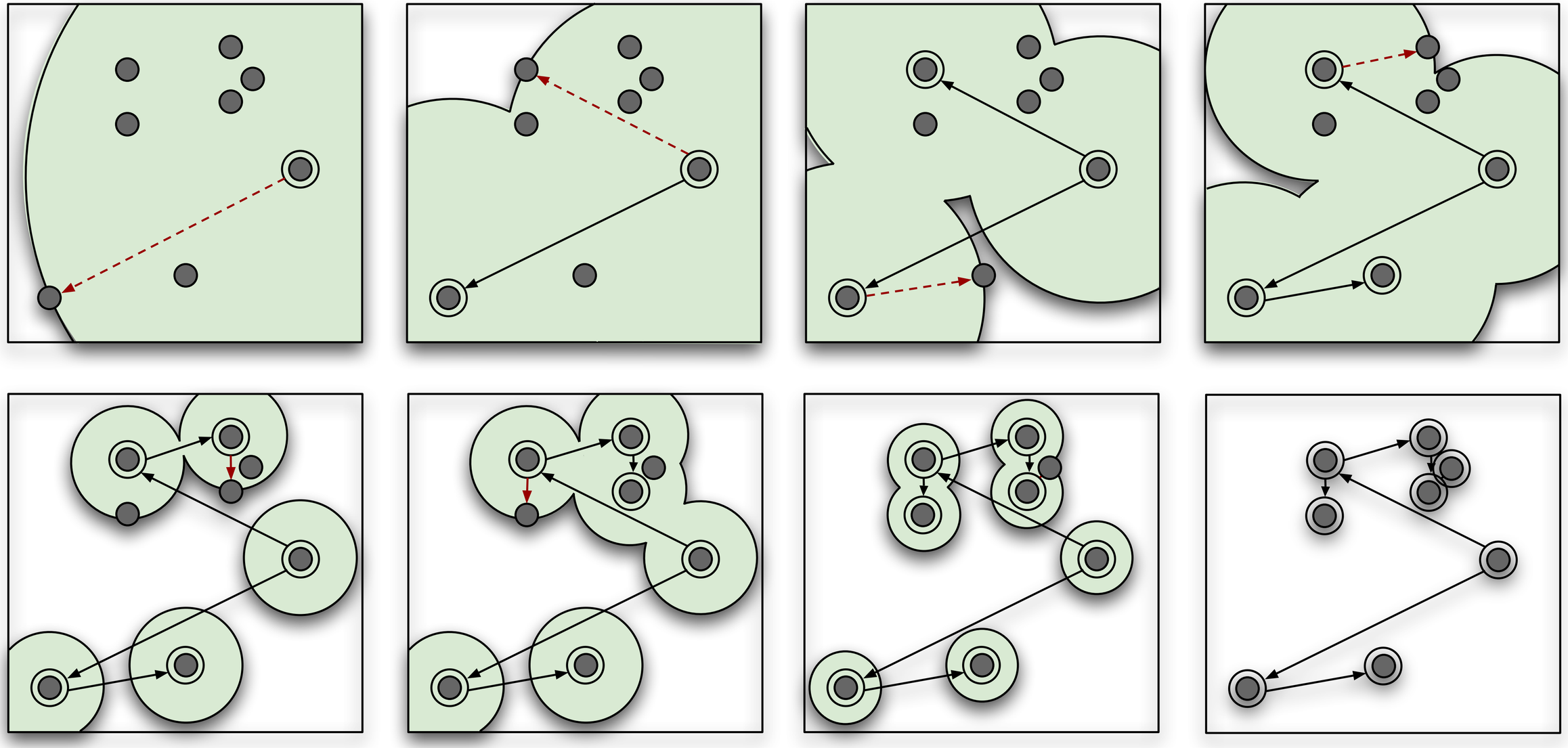
The i**-th prefix** of X is the first i points of X,

$$X_i = (x_0, x_1, \ldots, x_{i-1}).$$

X is a **greedy permutation** if

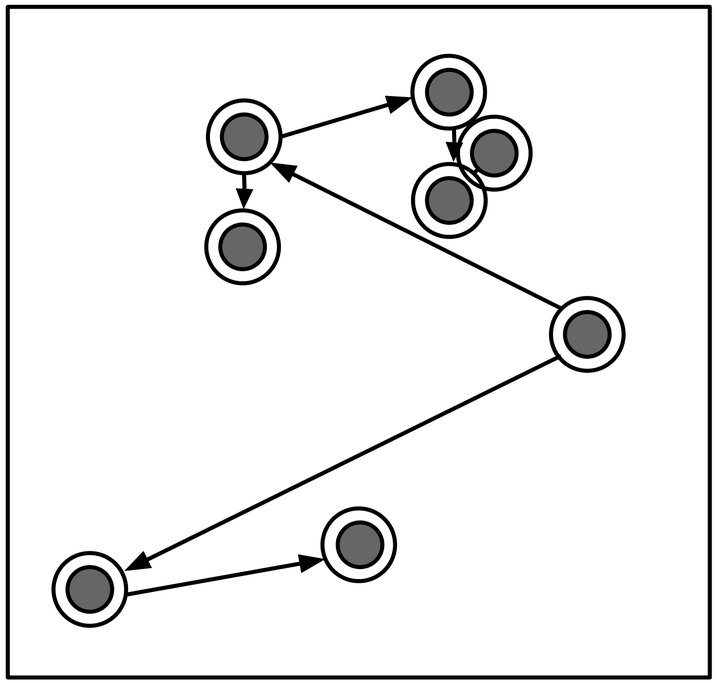
$$d(x_i, X_i) \geq \max_{j \geq i} d(x_j, X_i) \text{ for all } i > 0.$$

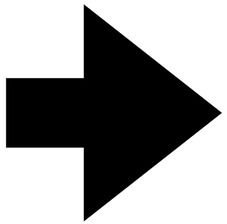
Greedy Permutations

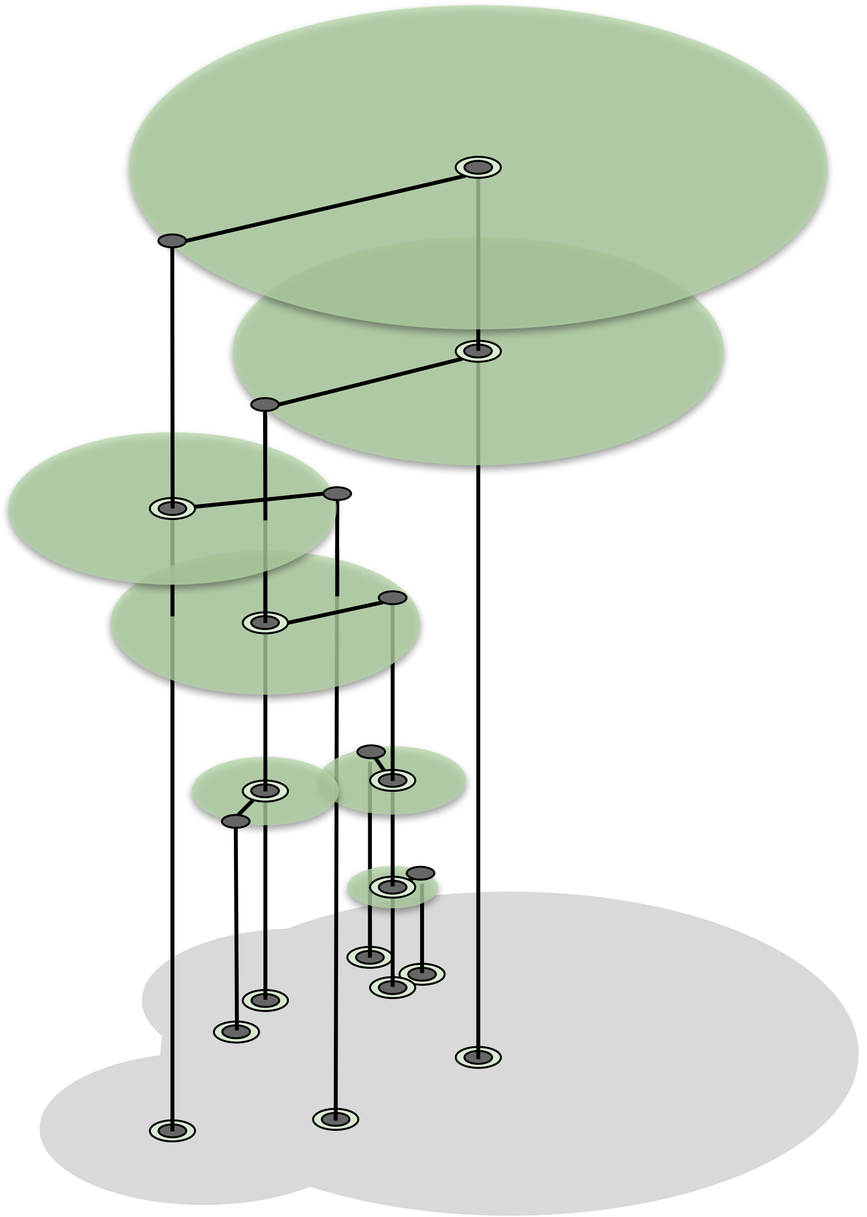


Greedy Trees

A ball tree constructed using a greedy permutation and predecessor mapping







Constructing a Greedy Tree

Input: A greedy permutation $X = (x_0, x_1, \ldots, x_{n-1})$

Insert a root node centered at x_0.

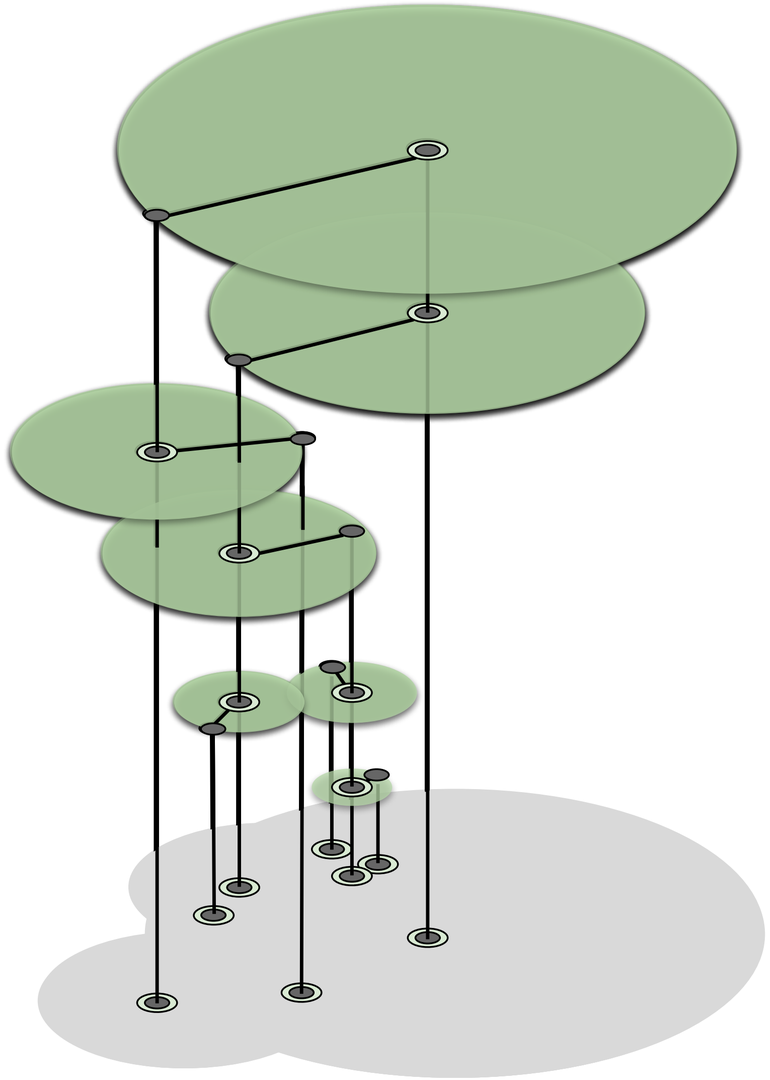
For i from 1 to $n - 1$:

Let y be the predecessor of x_i.

Attach two nodes to the leaf centered at y,

one centered at y, and one centered at x_i.

Constructing a Greedy Tree



Constructing Greedy Permutations

- Gonzalez introduced a simple quadratic time algorithm:
  - Insert one point and assign it as the nearest neighbor of all uninserted points.
  - Each iteration insert the point p with max distance to its nearest neighbor.
  - For each uninserted point, if p is closer than its nearest neighbor, update the nearest neighbor to p.

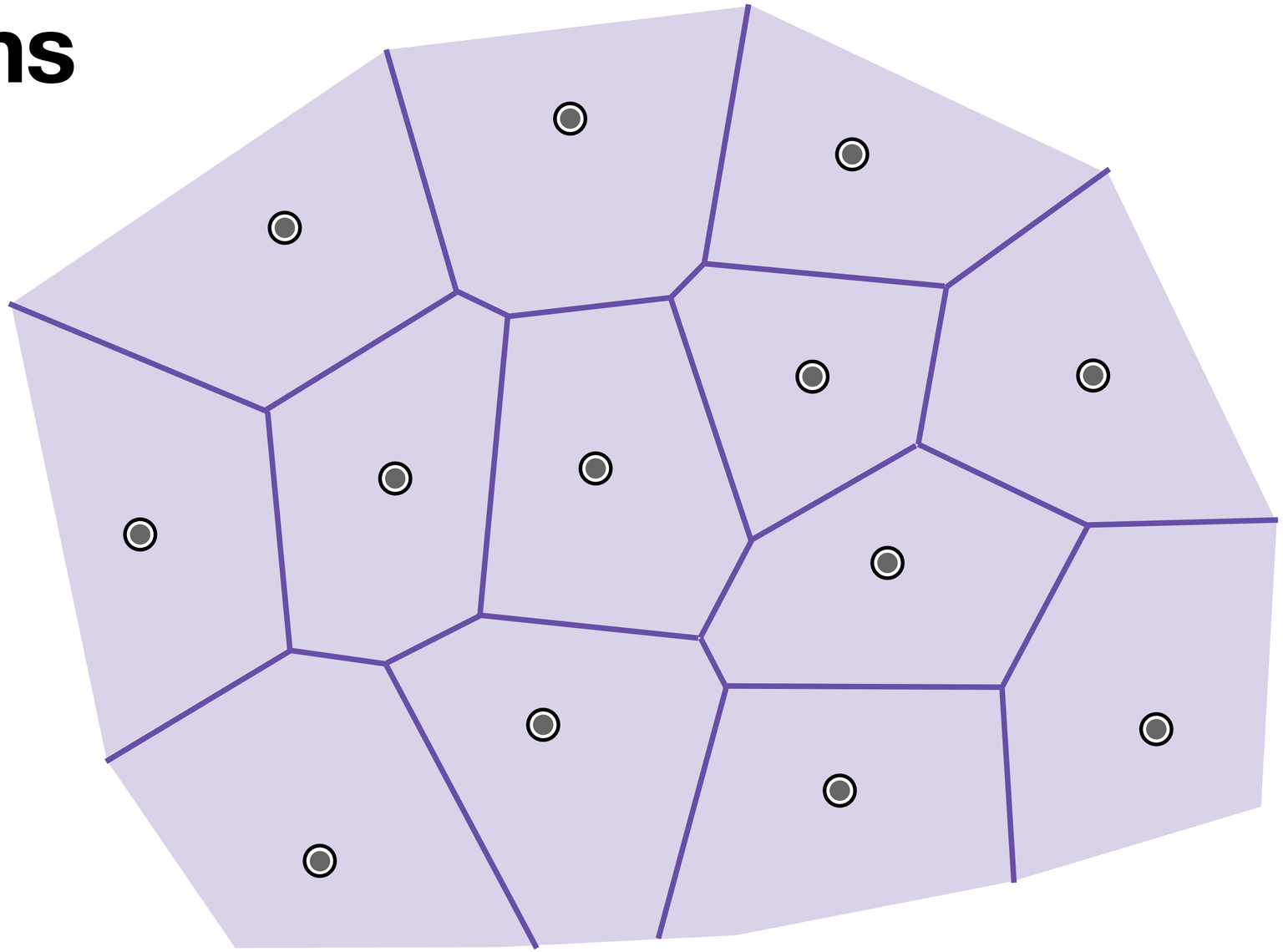
We can reframe this algorithm in terms of Voronoi diagrams…

Voronoi Diagrams

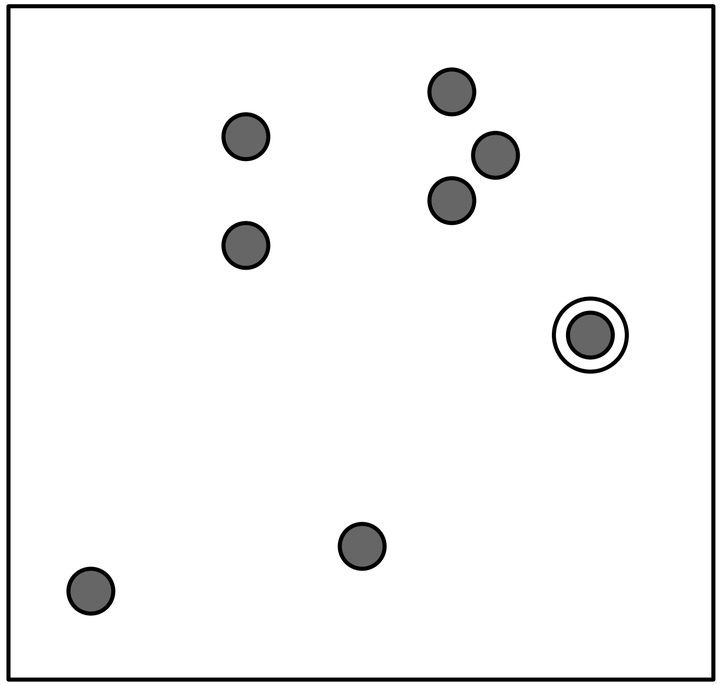
There is a set of sites.

Each site is the center of a cell.

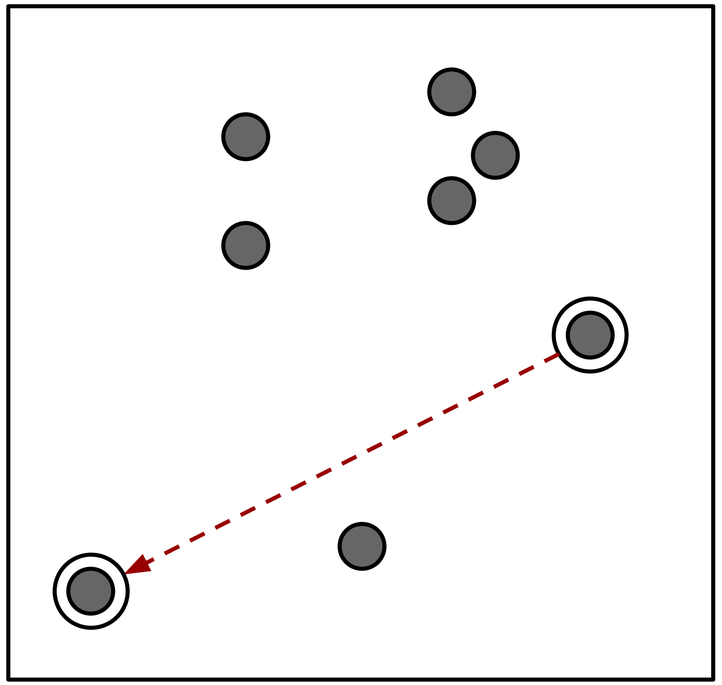
Points belong in the cell of the closest site.



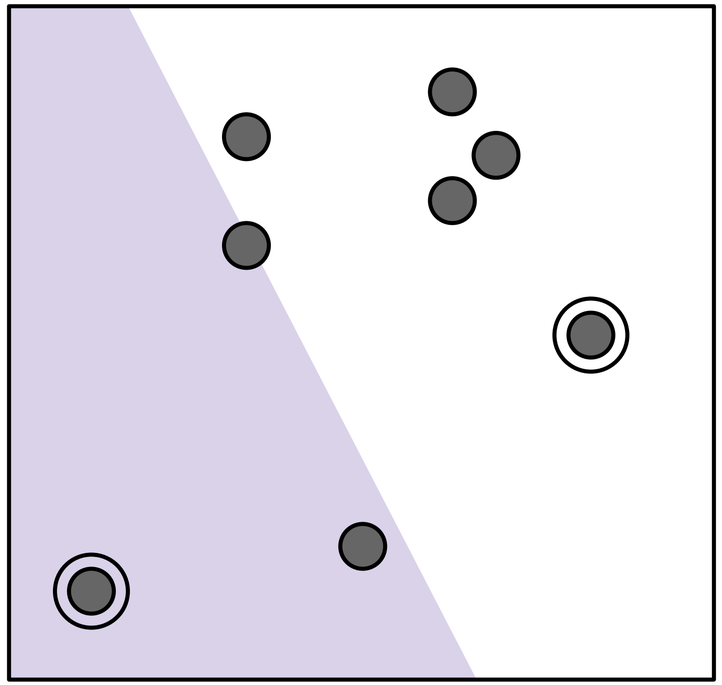
Constructing a Voronoi Diagram



All points start
in the same cell

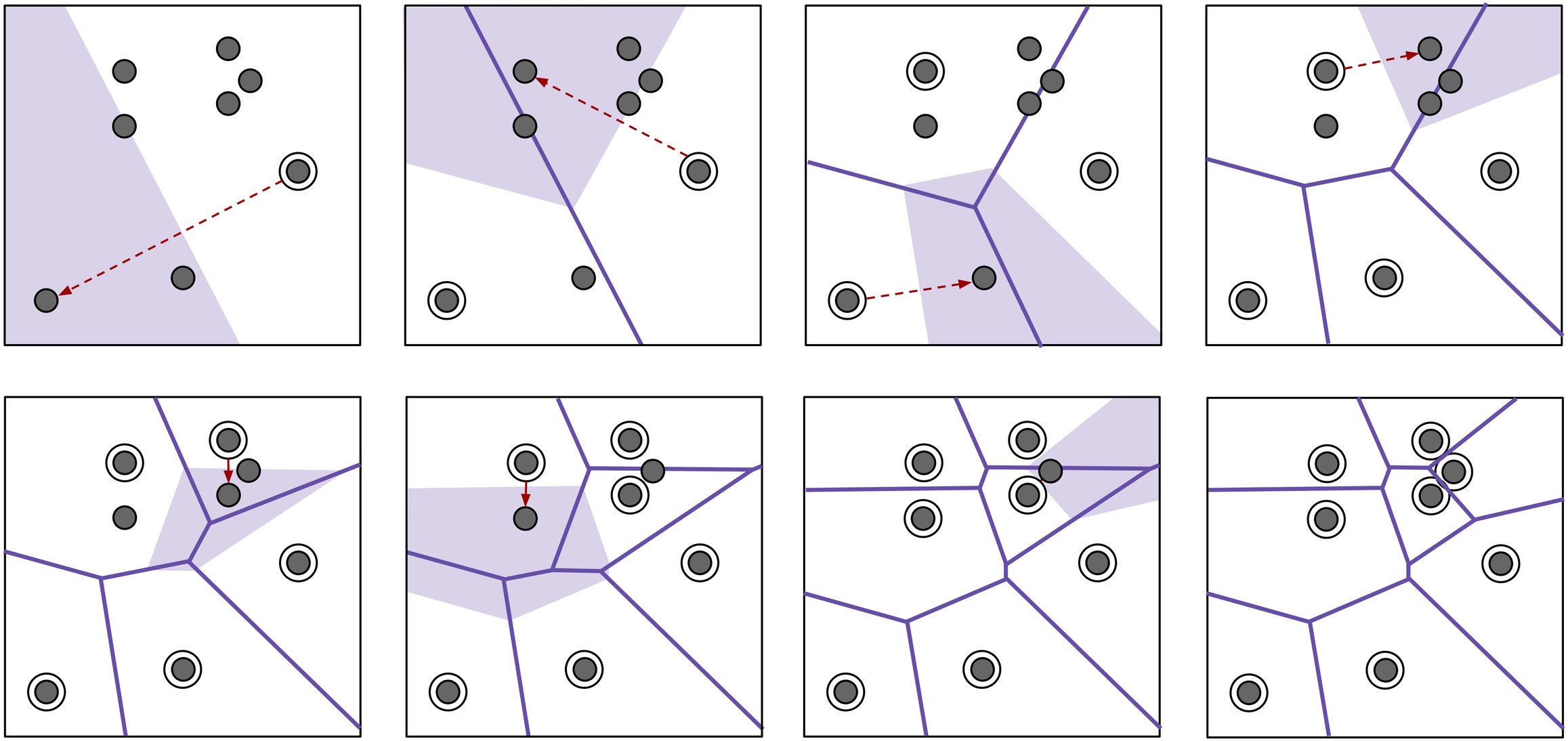


Insert a point



Point location
(Decide who moves)

Construct a Greedy Permutation



Improving The Naive Algorithm

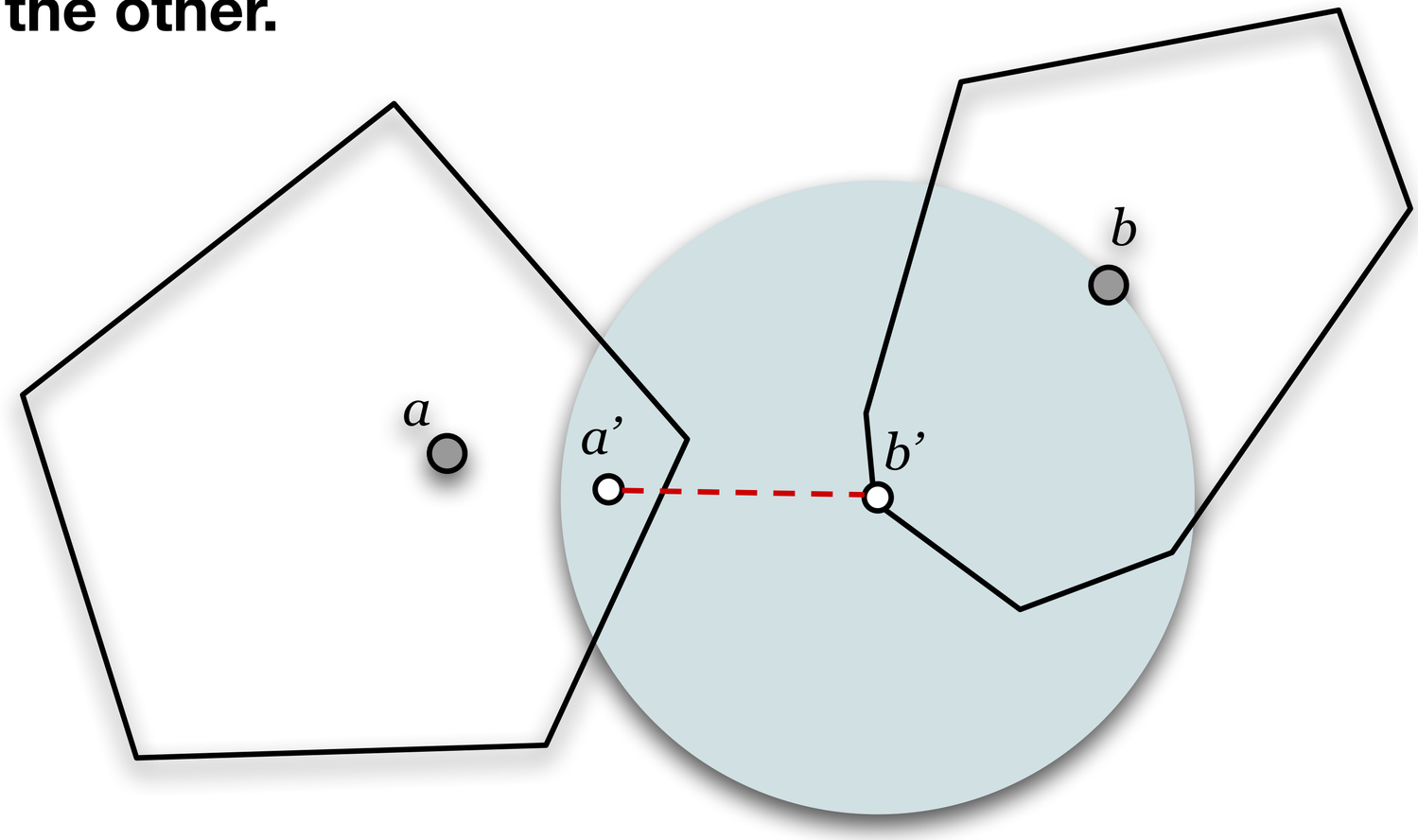
1. **Only check points in cells “close enough” to the new site.**
2. **Put uninserted points into a heap keyed by the distance to their site.**
3. **Move clusters of points at a time.**

Neighbor Graph

- **In Clarkson's algorithm, "close enough" information is stored as edges in a graph.**
- **Har-Peled and Mendel show that Clarkson's algorithm runs in** $O(n \log \Delta)$**.**

Neighbor Graph

- **Only keep an edge if inserting a point from one cell would cause a point to move in the other.**



Improving The Naive Algorithm

1. Only check points in cells “close enough” to the new site.
2. Put uninserted points into a heap keyed by the distance to their site.
3. **Move clusters of points at a time.**

Recursive Merging

- The input is two greedy trees.
- We can merge greedy trees in linear time using a variation of Clarkson's algorithm.
- This gives us a divide and conquer algorithm to construct the greedy tree.

Technicalities

(for linear-time merge)

1. The heap now needs to store nodes instead of points.
2. We use a bucket queue to achieve constant-time heap operations.
3. Storing clusters in cells introduces some sources of approximation during construction.

Conclusion

Theorem:

Using a recursive algorithm, the *greedy tree* can be constructed deterministically in $O(n \log n)$ time, and in $O(n)$ space.

Summary of Results

	Net-trees	Greedy Trees
Construction Time	$2^{O(d)} n \log n$ **expected, randomized**	$2^{O(d)} n \log n$ **deterministic**
Space	$O(n \log n)$	$O(n)$

We improve on prior work while also achieving an added degree of simplicity.

References

Bentley, J. L., Friedman, J. H., & Maurer, H. A. (1978). Two papers on range searching.

Har-Peled, S., & Mendel, M. (2005, June). Fast construction of nets in low dimensional metrics, and their applications. In *Proceedings of the twenty-first annual symposium on Computational geometry* (pp. 150-158).

Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, *38*, 293-306.

Clarkson, K. L. (2003). Nearest neighbor searching in metric spaces: Experimental results for sb (S). *Preliminary version presented at ALENEX99*, 63-93.