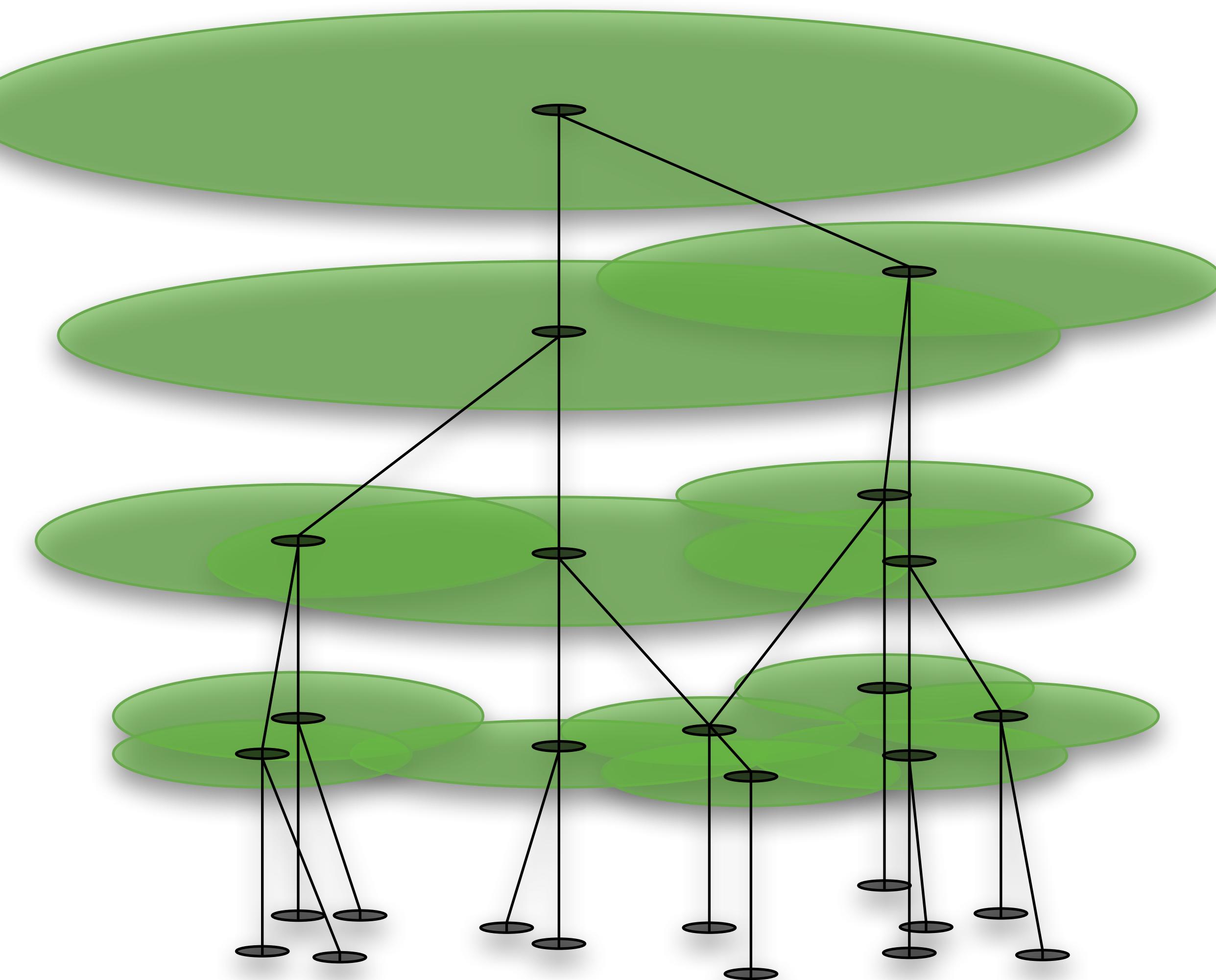


Simple Construction of Greedy Trees and Greedy Permutations



Oliver, Sid, Don, FWCG'24

Greedy Permutations

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

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

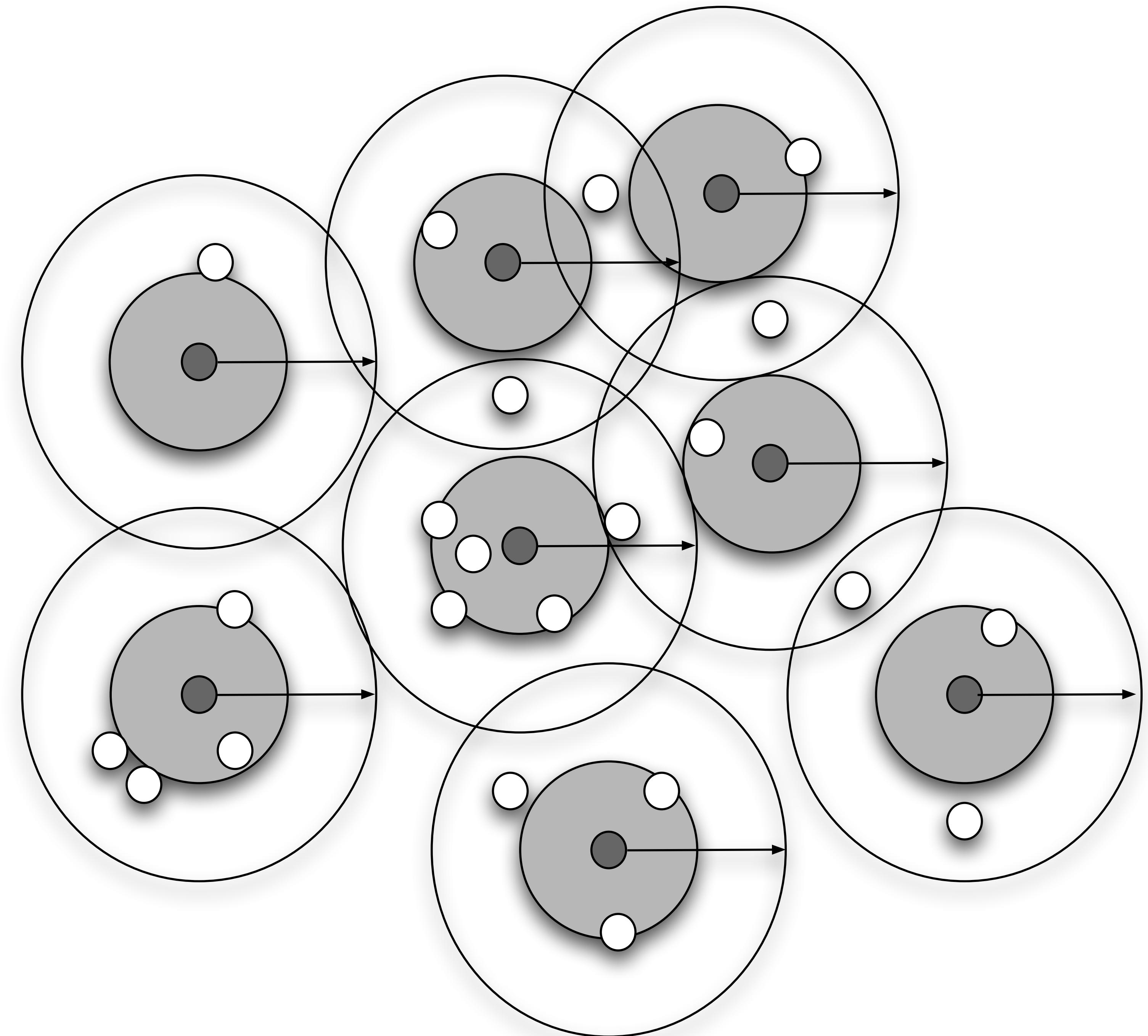
$$X_i = (x_0, x_1, \dots, x_{i-1}).$$

X is a **greedy permutation** if

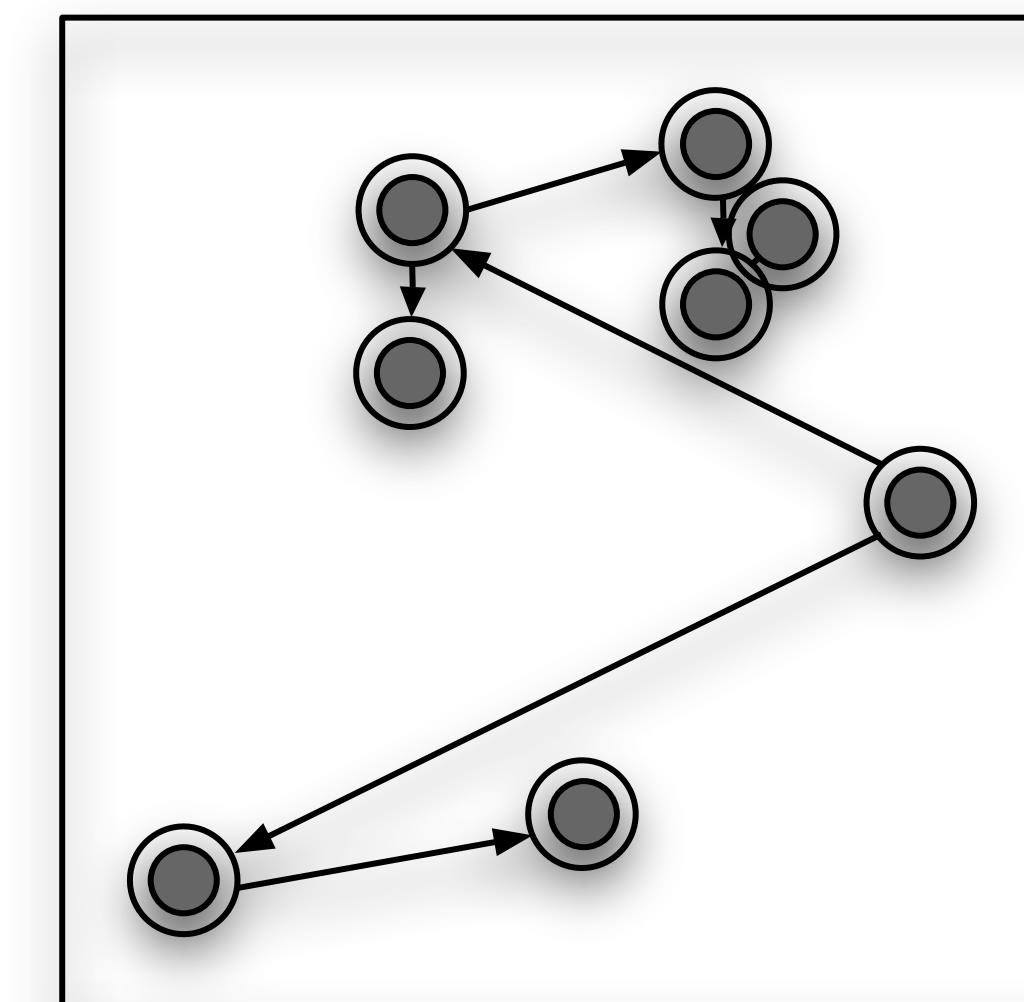
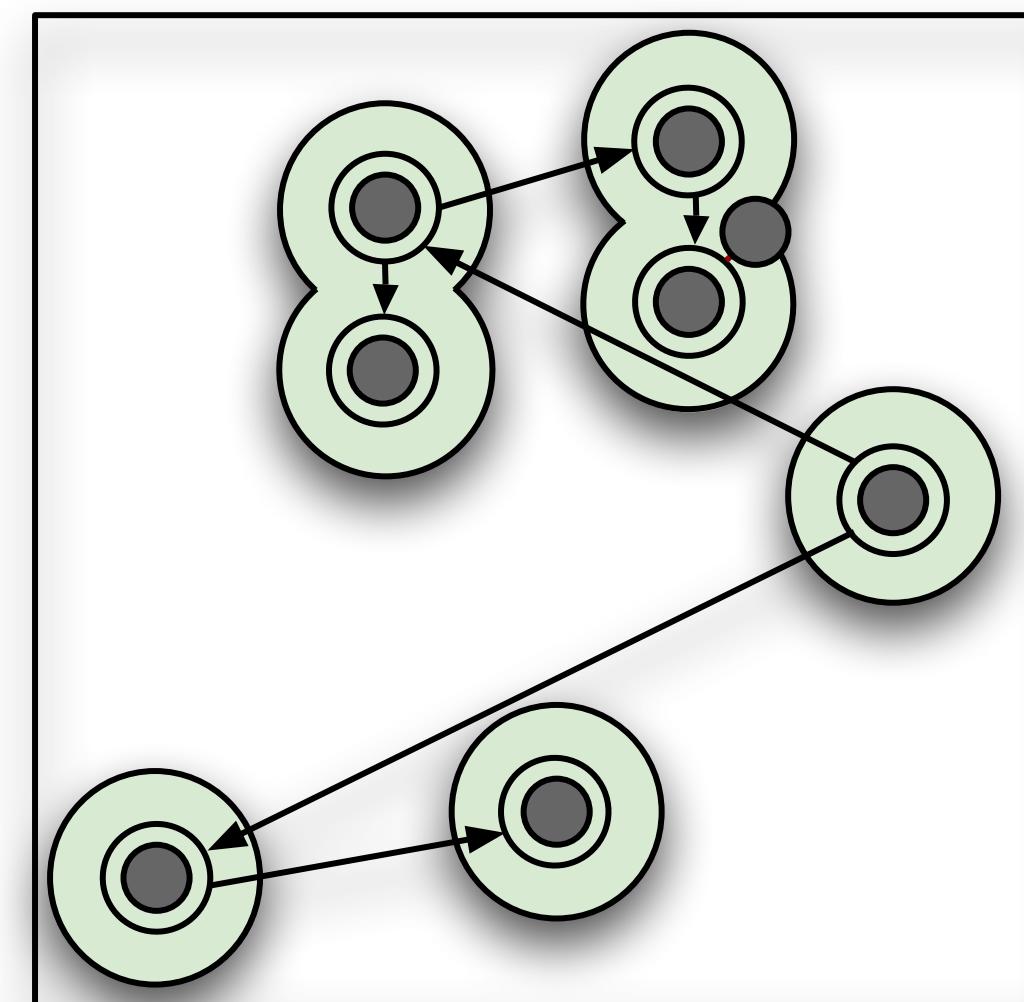
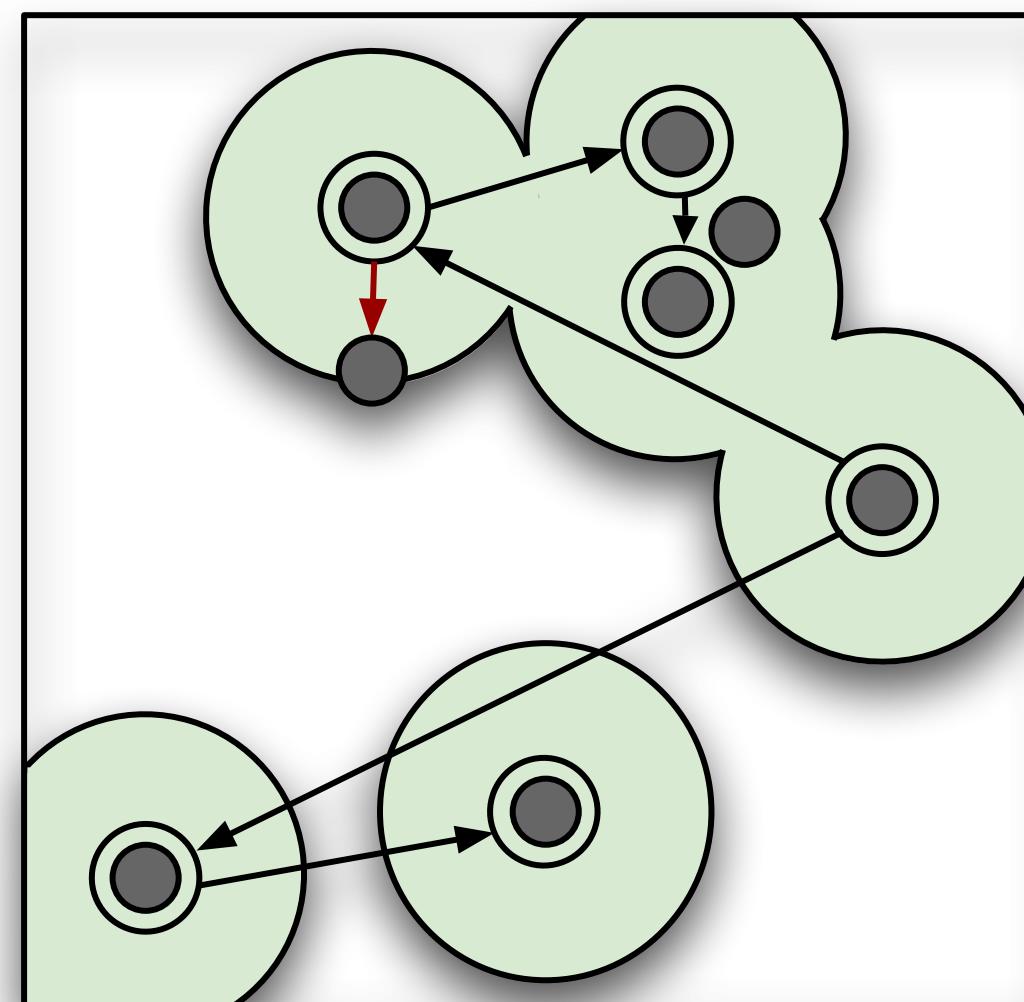
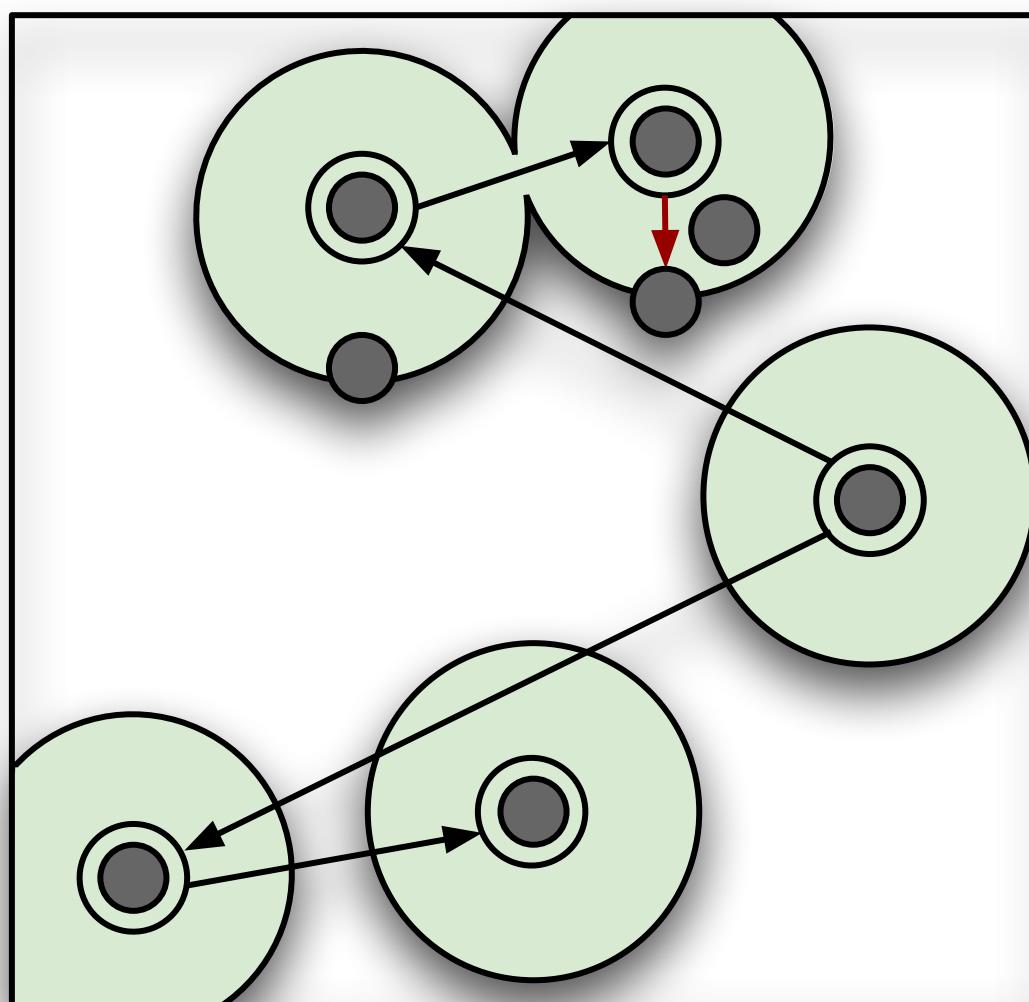
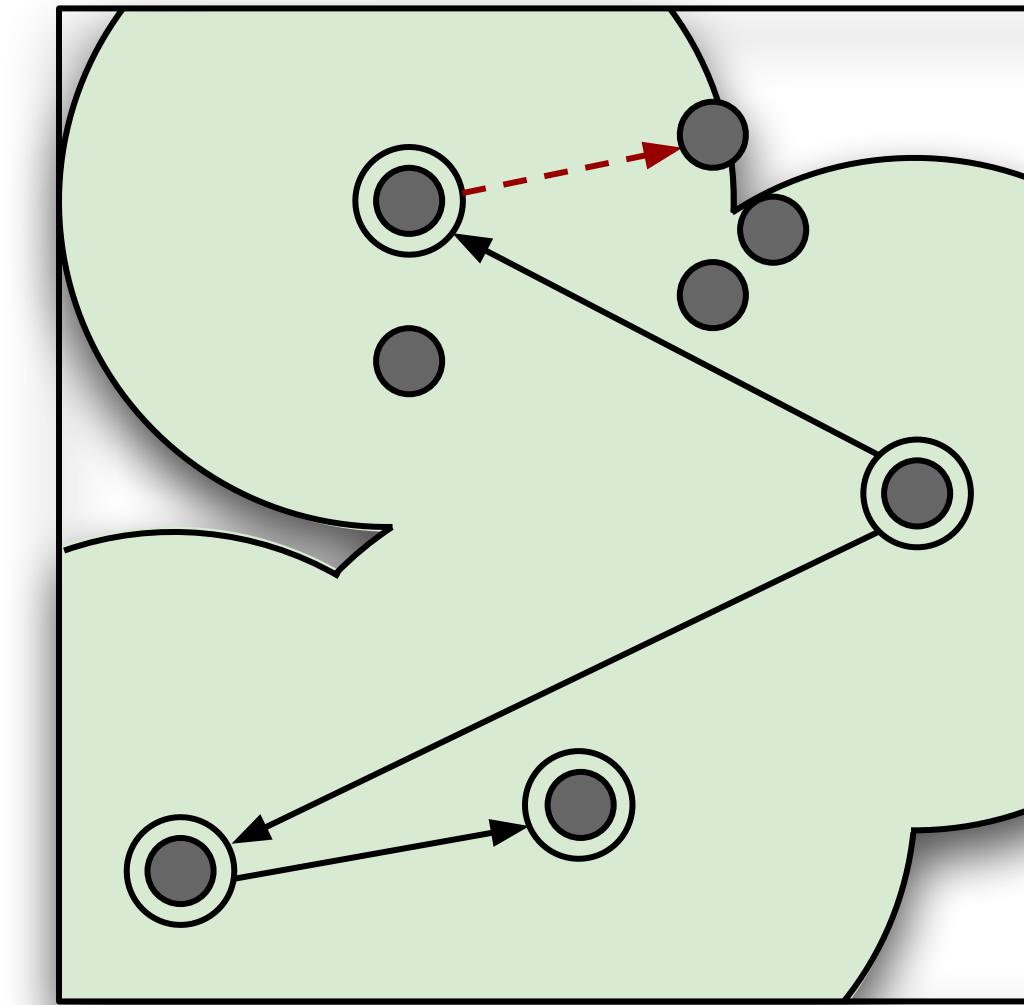
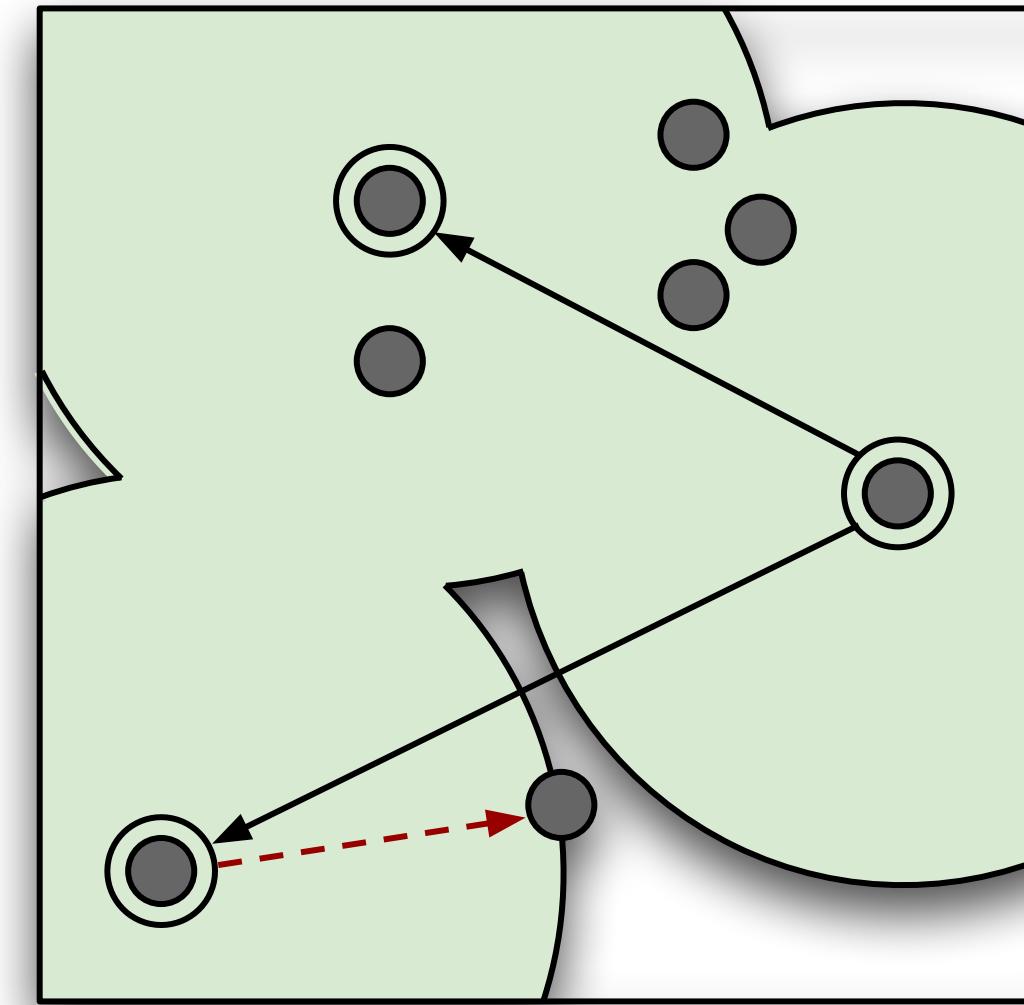
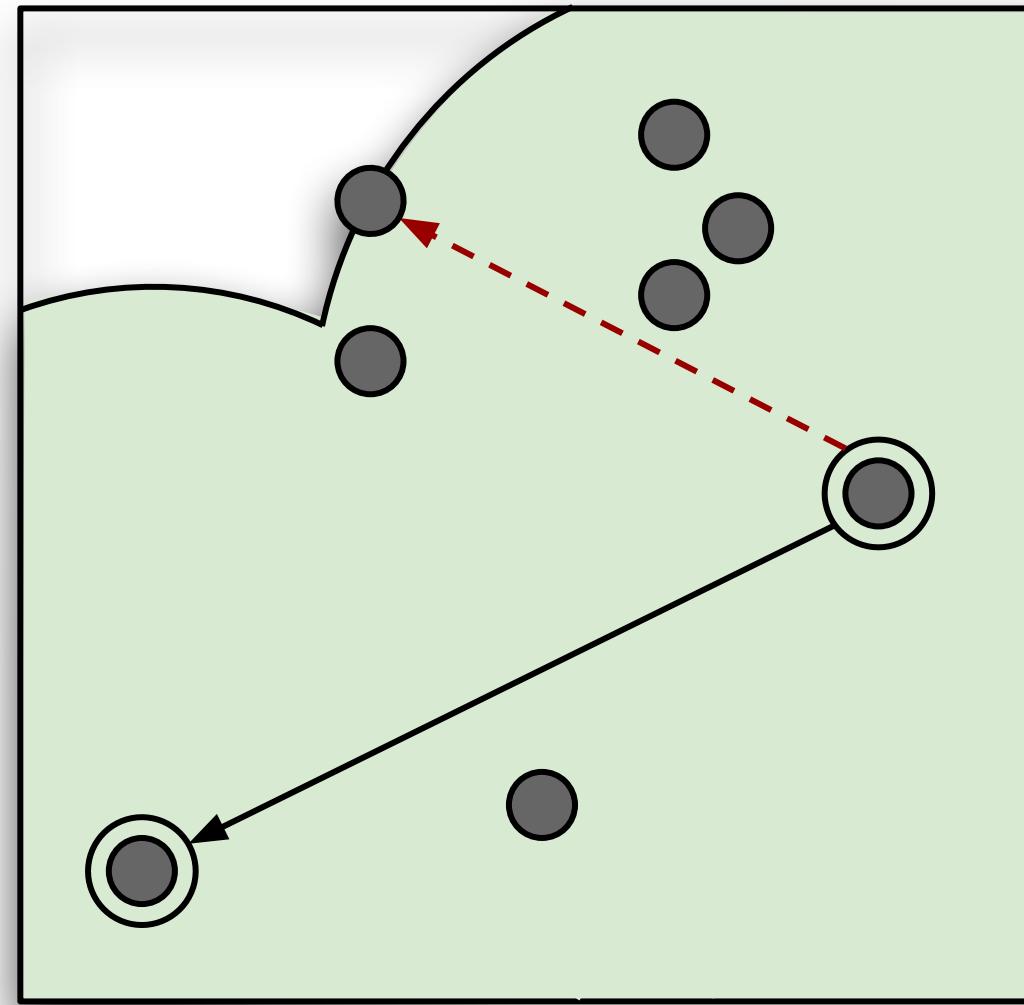
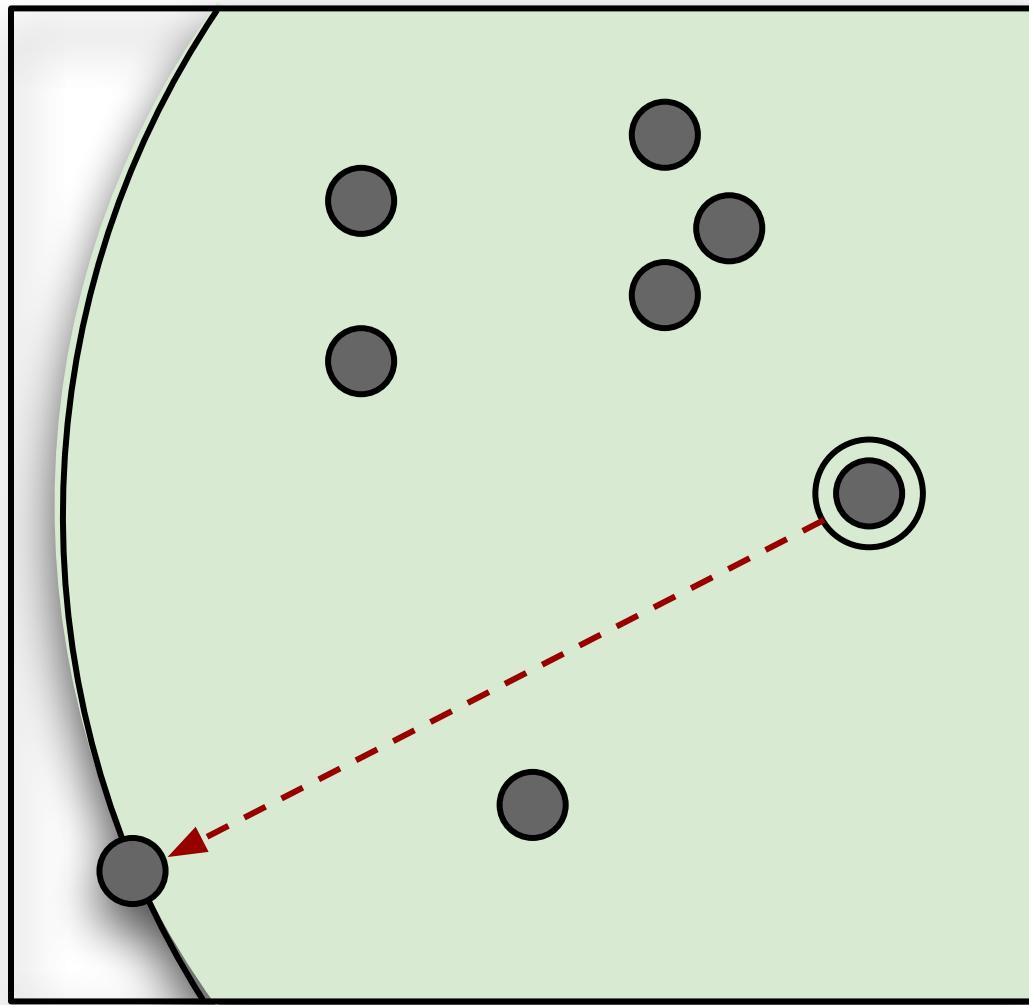
$$d(x_i, X_i) \geq \max_{j \geq i} d(x_j, X_i) \text{ for all } i > 0.$$

Nets

A Packing and Covering



Greedy Permutations



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 .**

(Later we can reframe this in terms of Voronoi diagrams...)

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 pre-processing**
- $2^{O(d)} \log n + (1/\varepsilon)^{O(d)}$ **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]

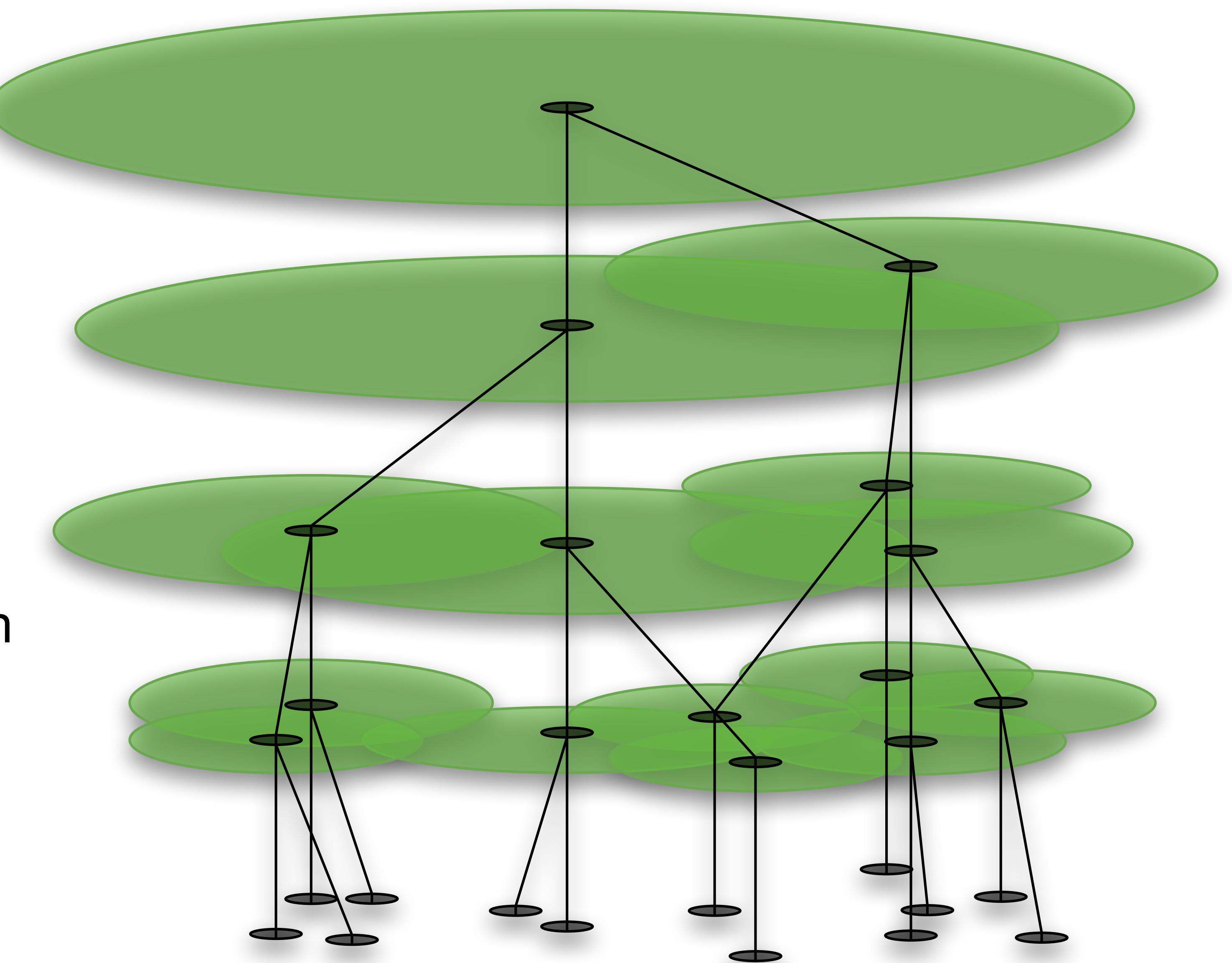
Summary of Results

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

Goal: match or improve on prior work using a simpler algorithm.

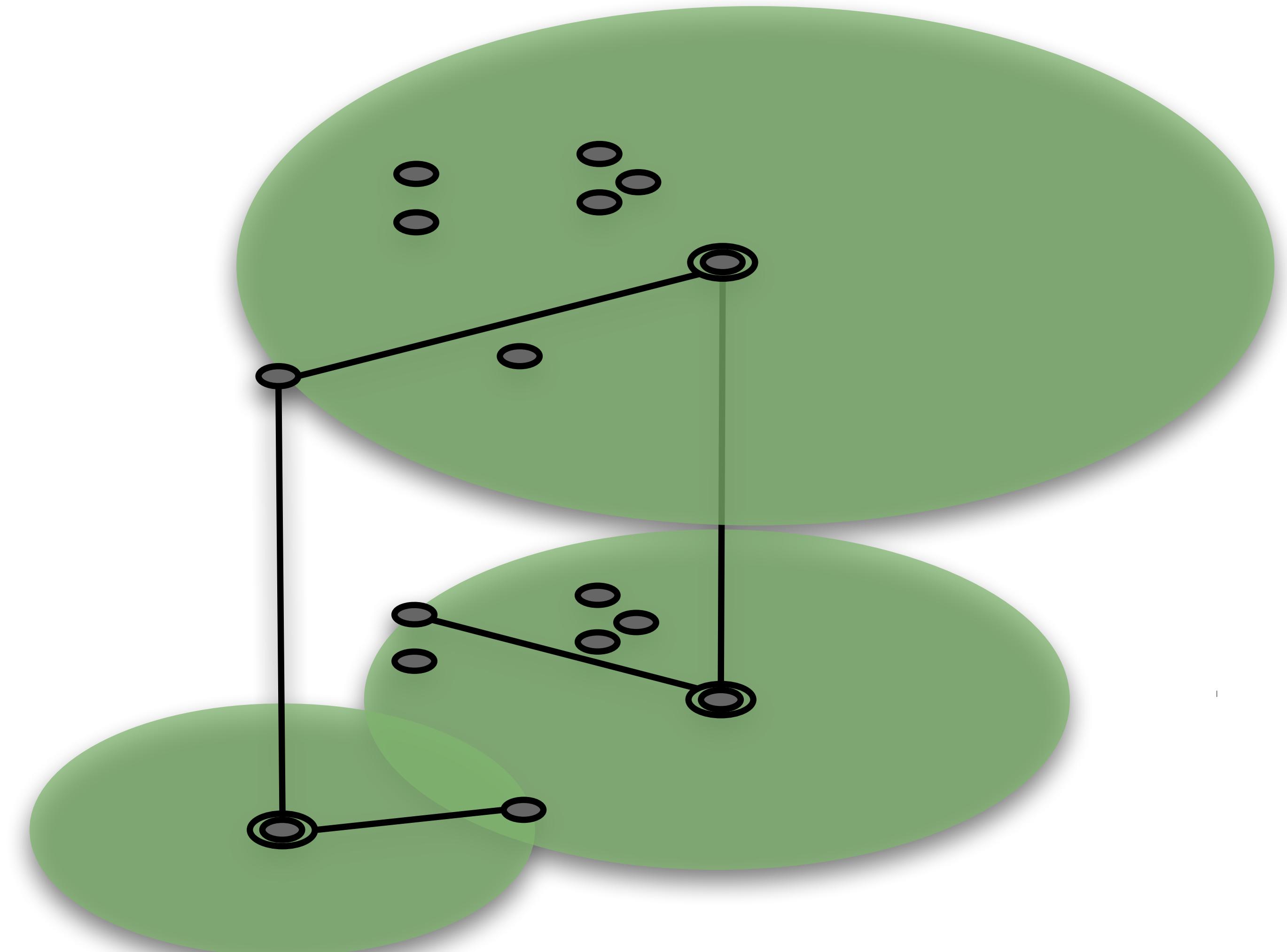
Ball Trees

- 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



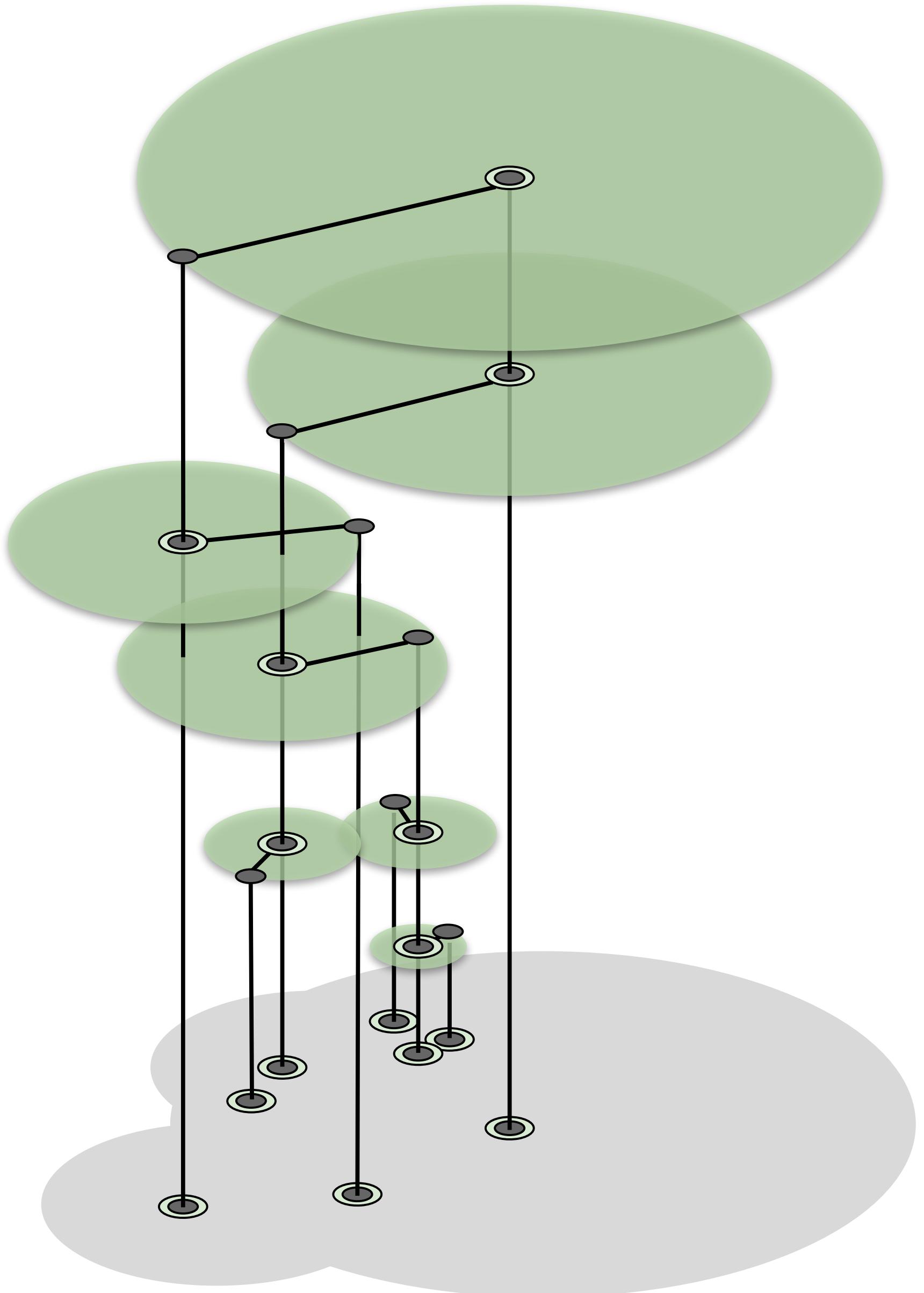
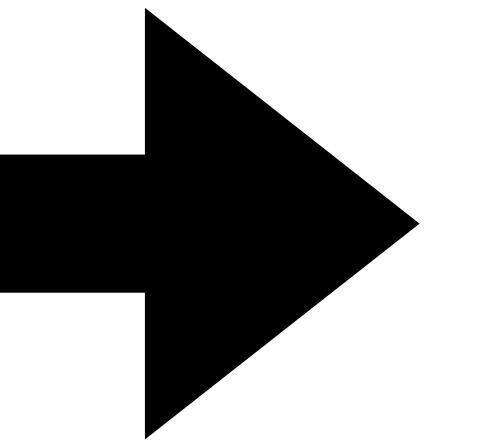
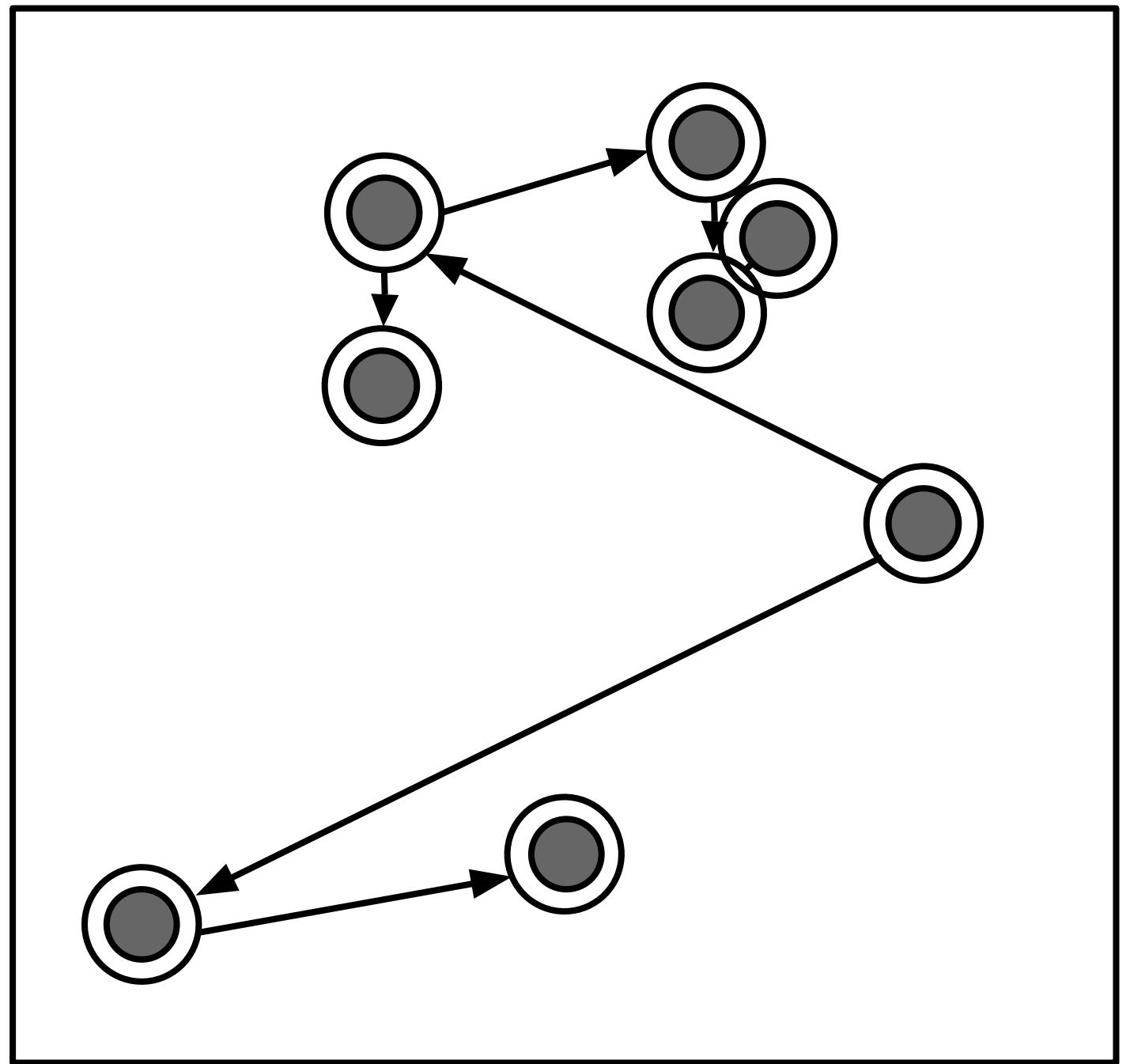
Split a node

- 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.



Greedy Trees

A ball tree constructed using
a greedy permutation and
predecessor mapping



Constructing a Greedy Tree

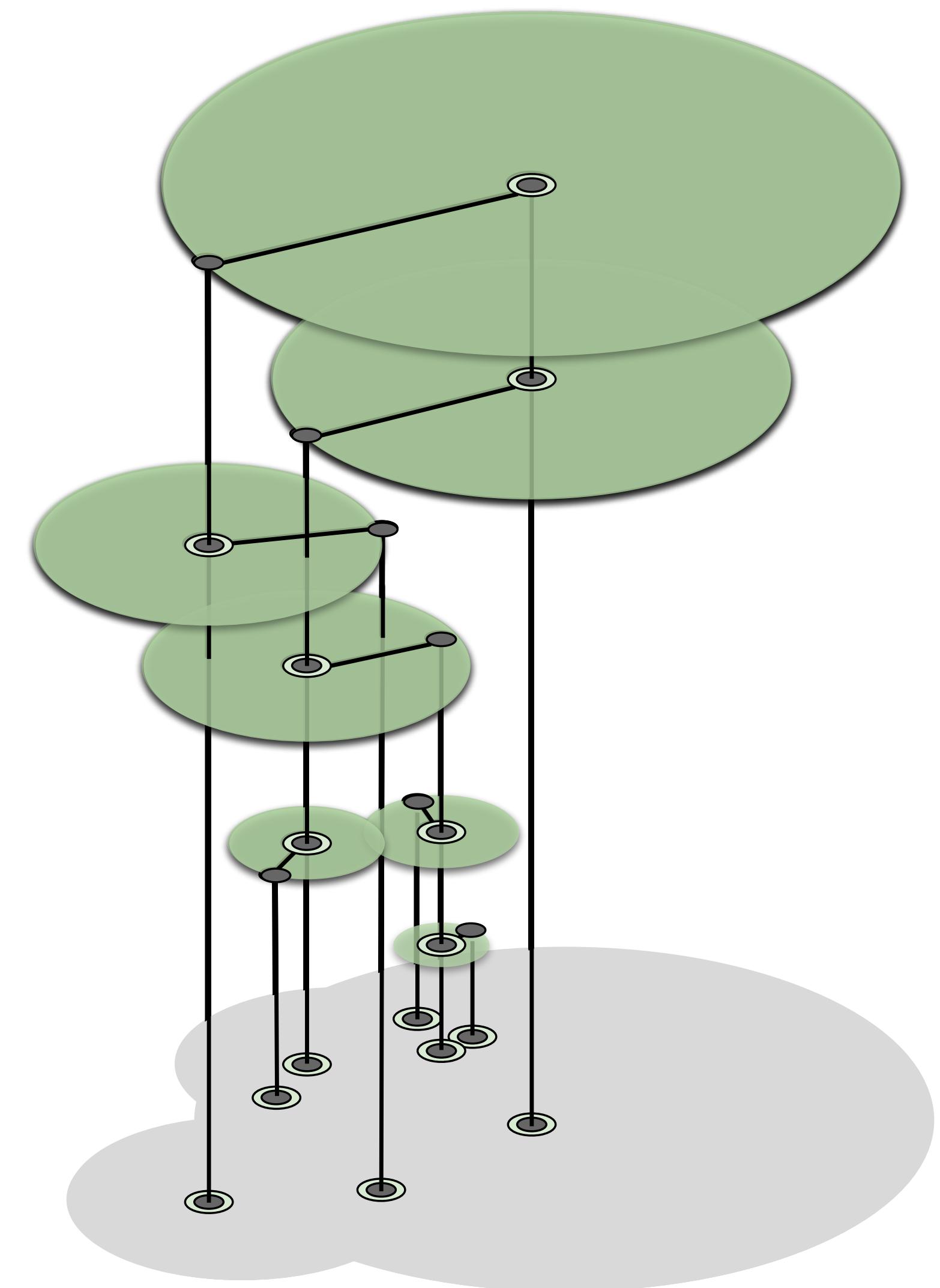
Input: A greedy permutation $X = (x_0, x_1, \dots, x_{n-1})$

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

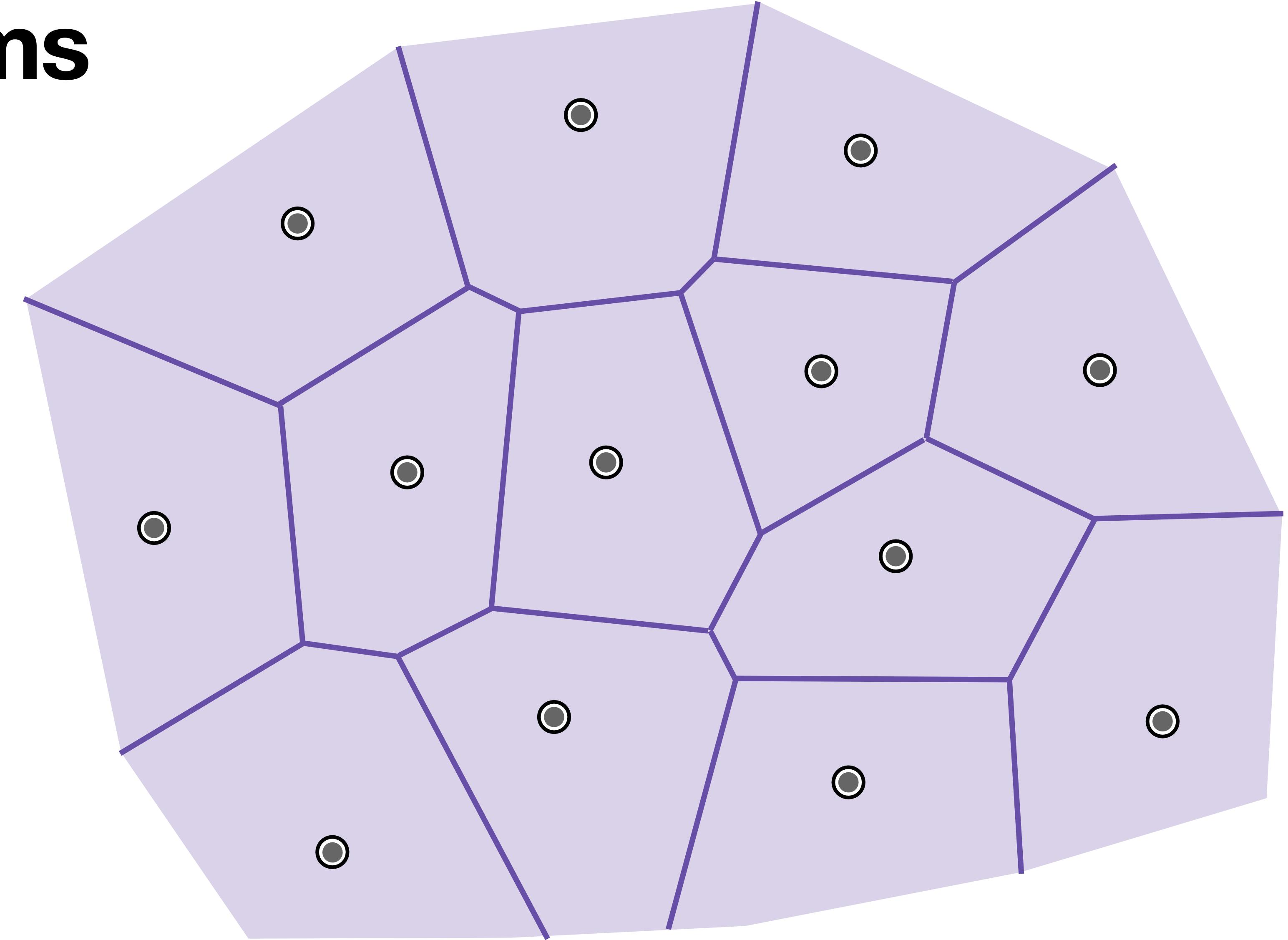


Voronoi Diagrams

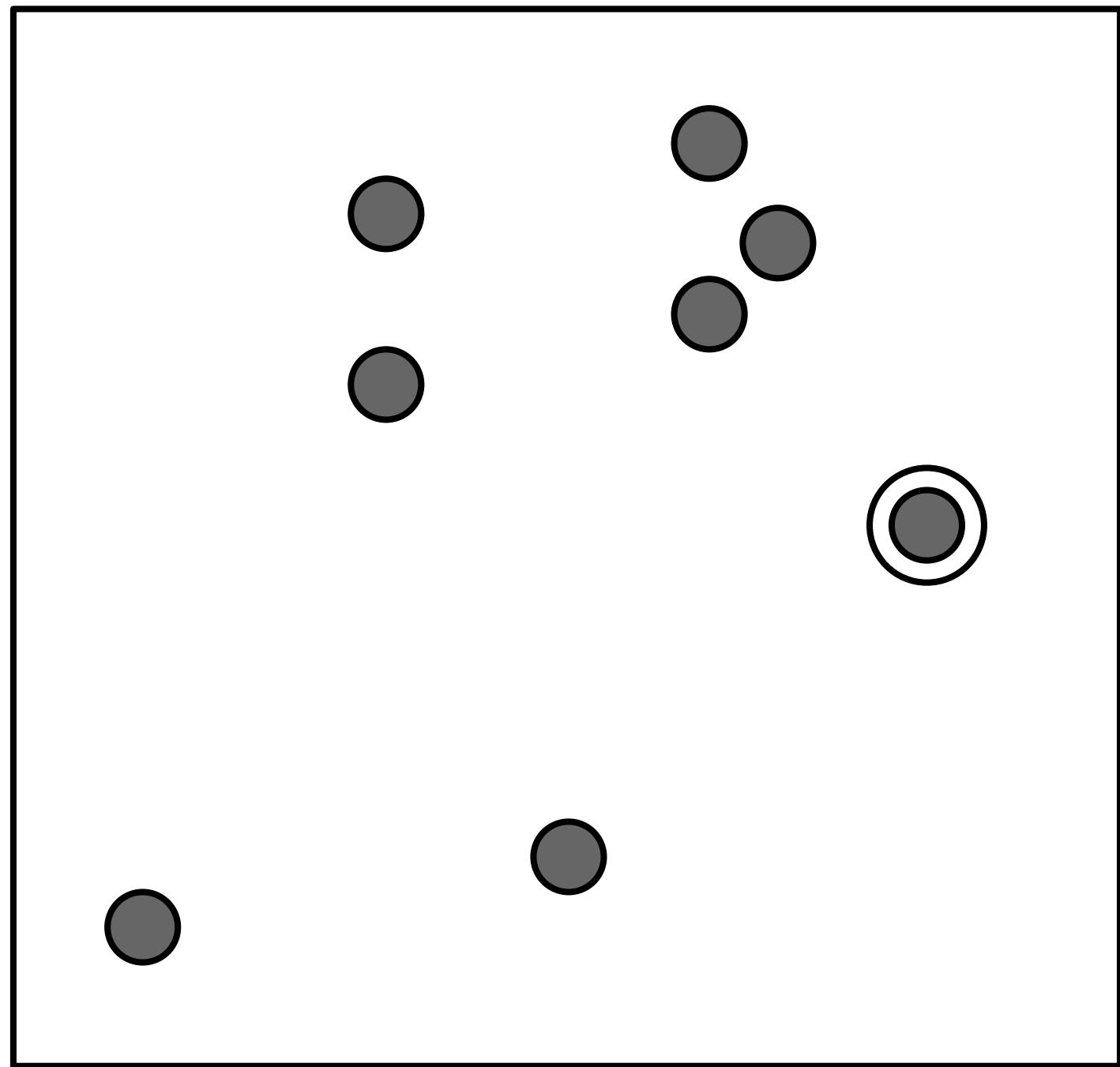
There is a set of sites.

**Each site is the center
of a cell.**

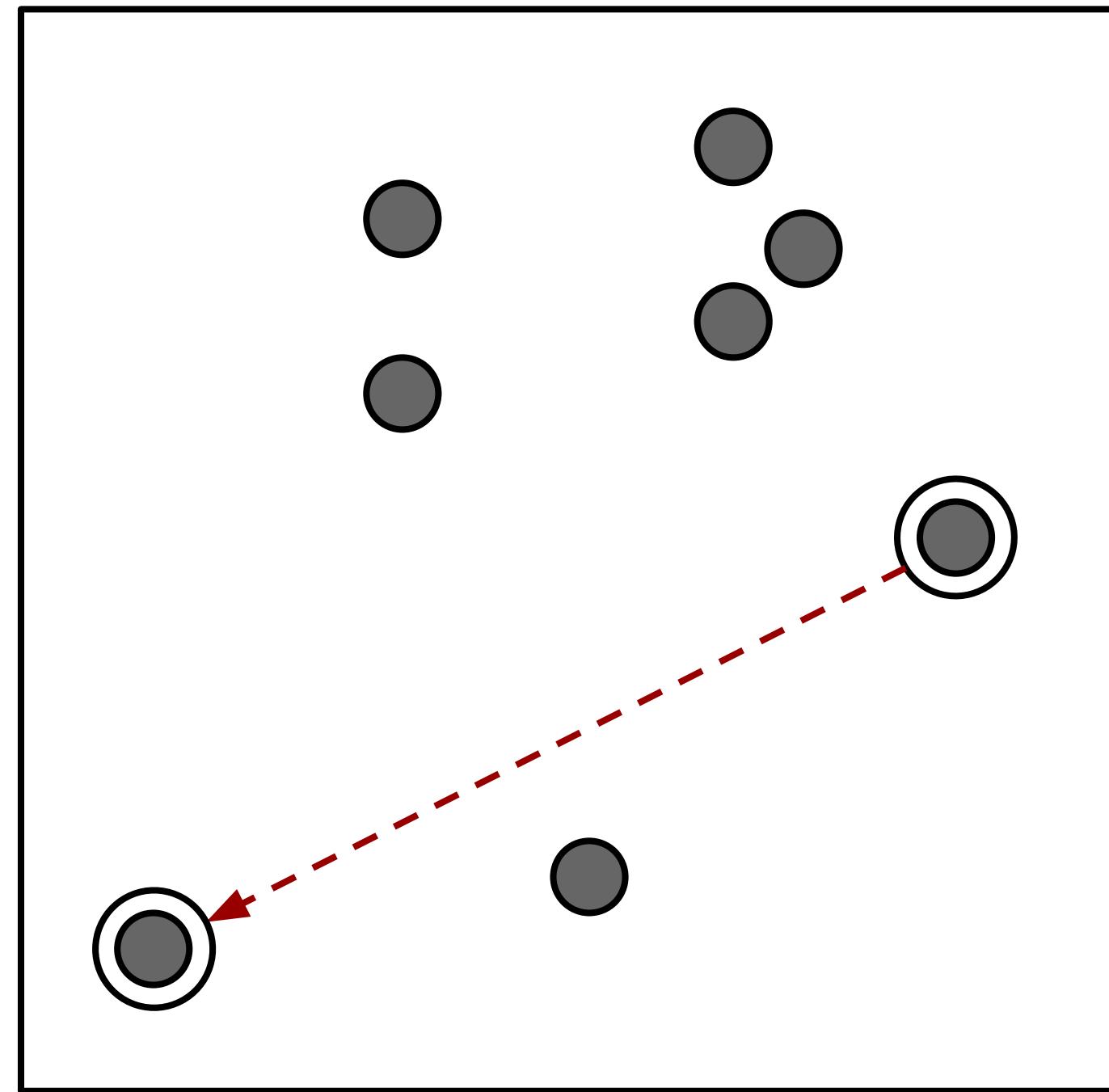
**Points belong in the cell
of the closest site.**



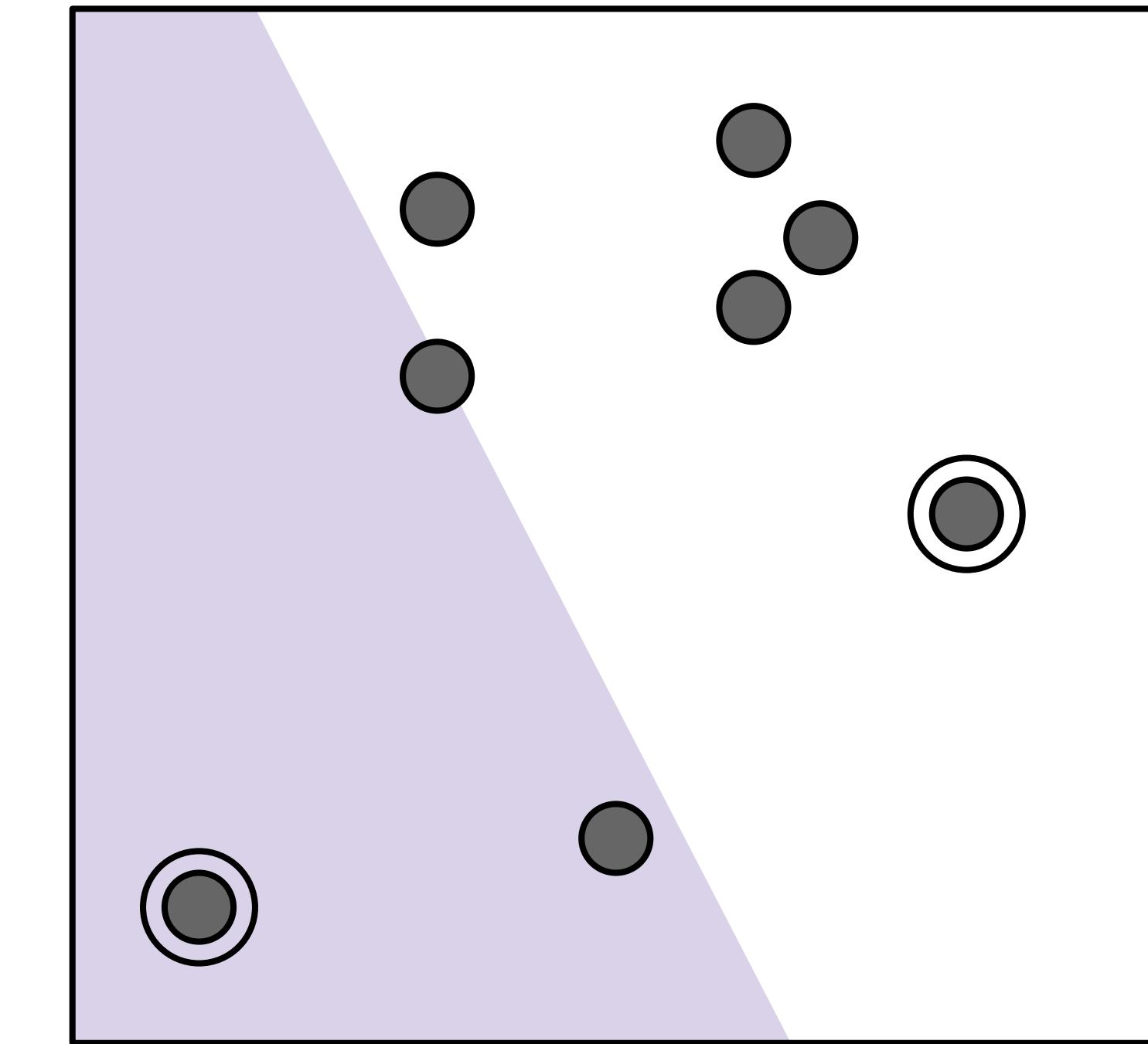
Constructing a Voronoi Diagram



All points start
in the same cell

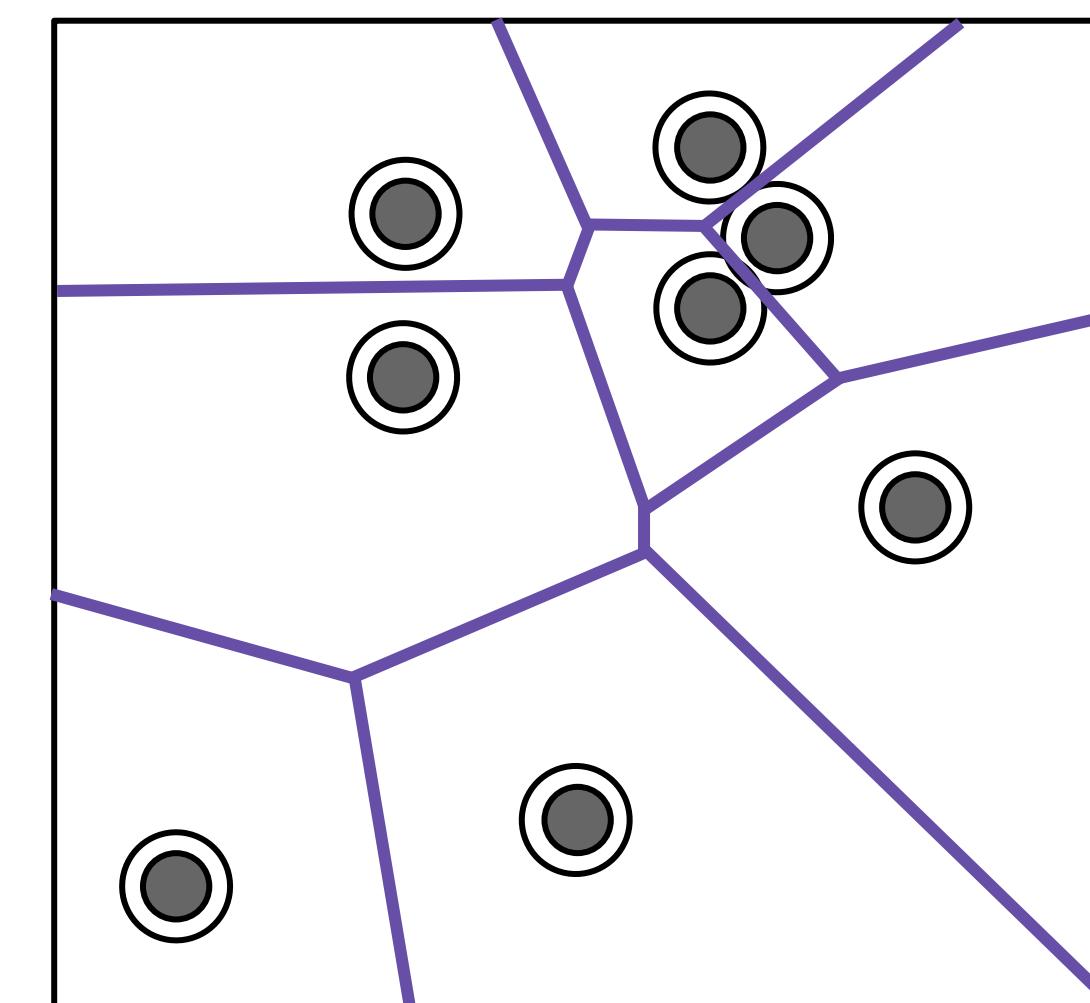
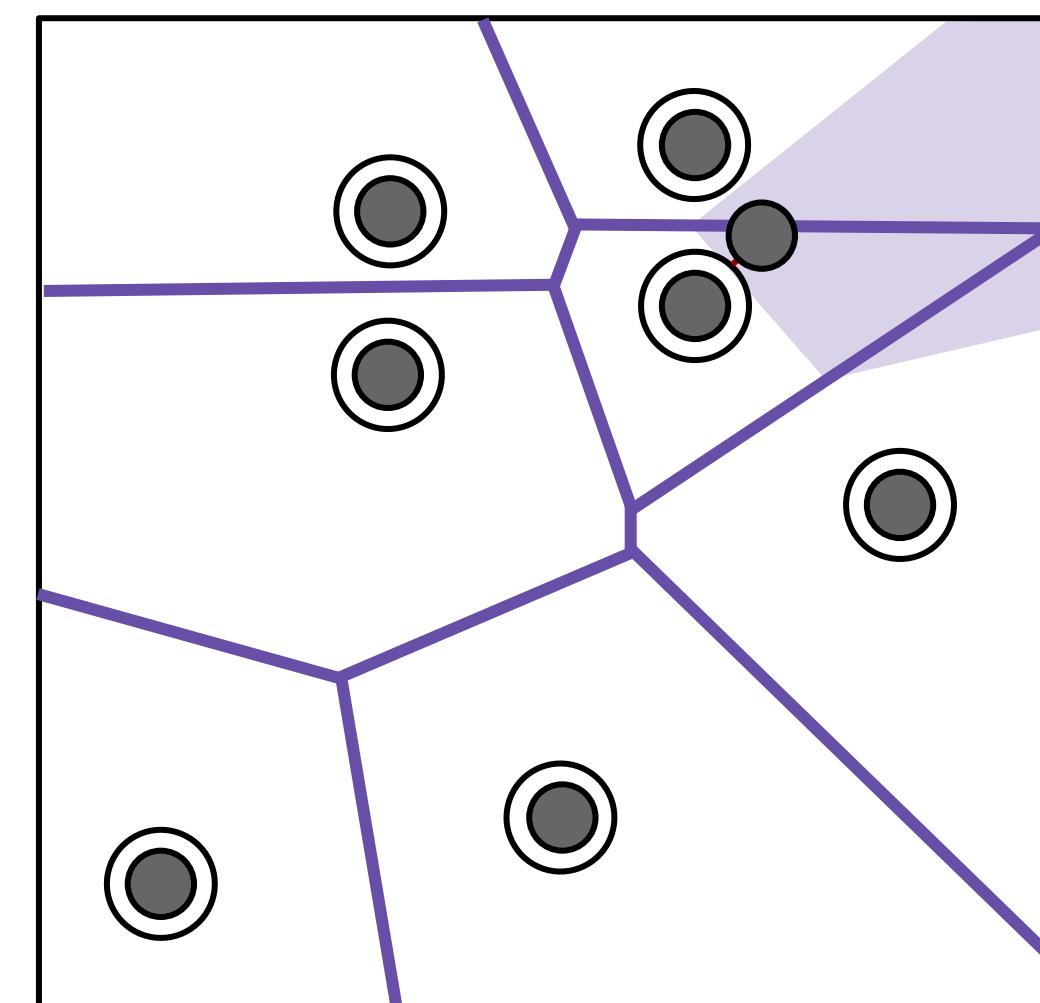
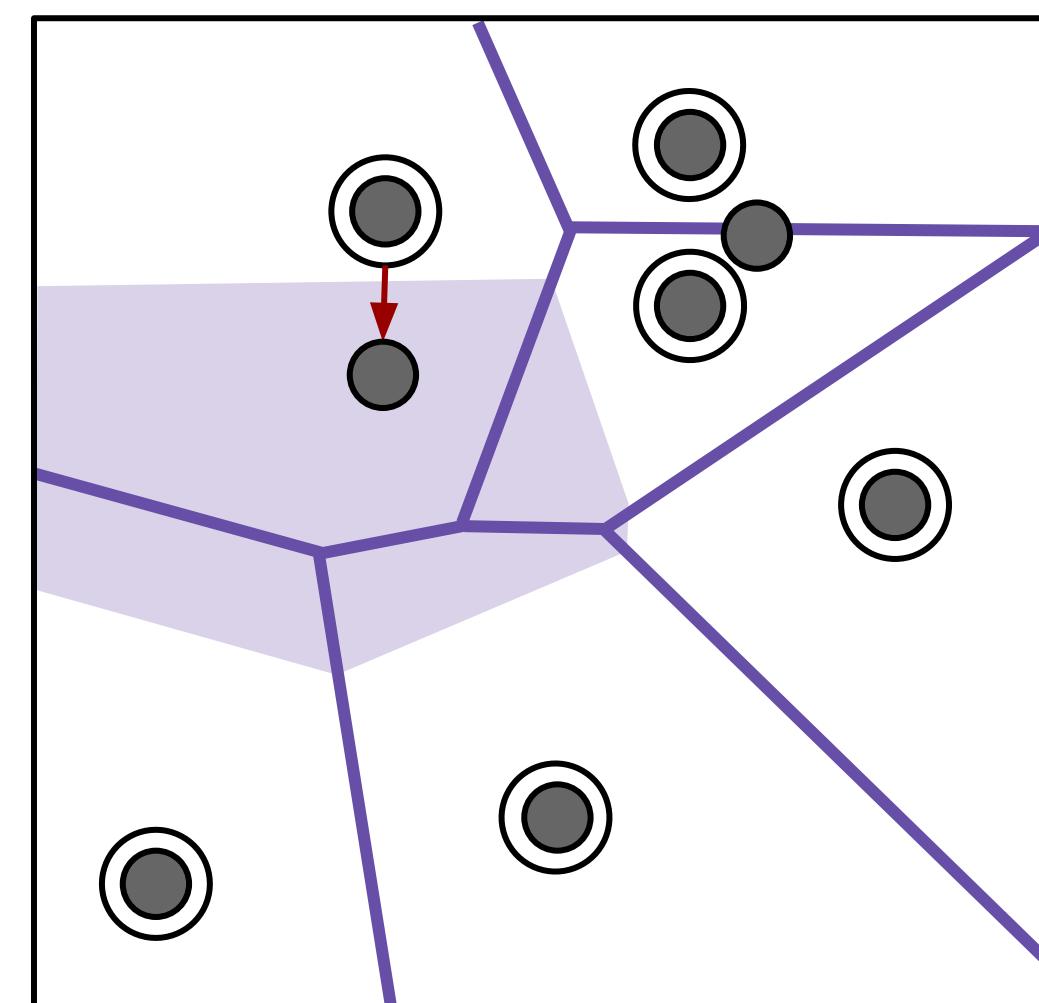
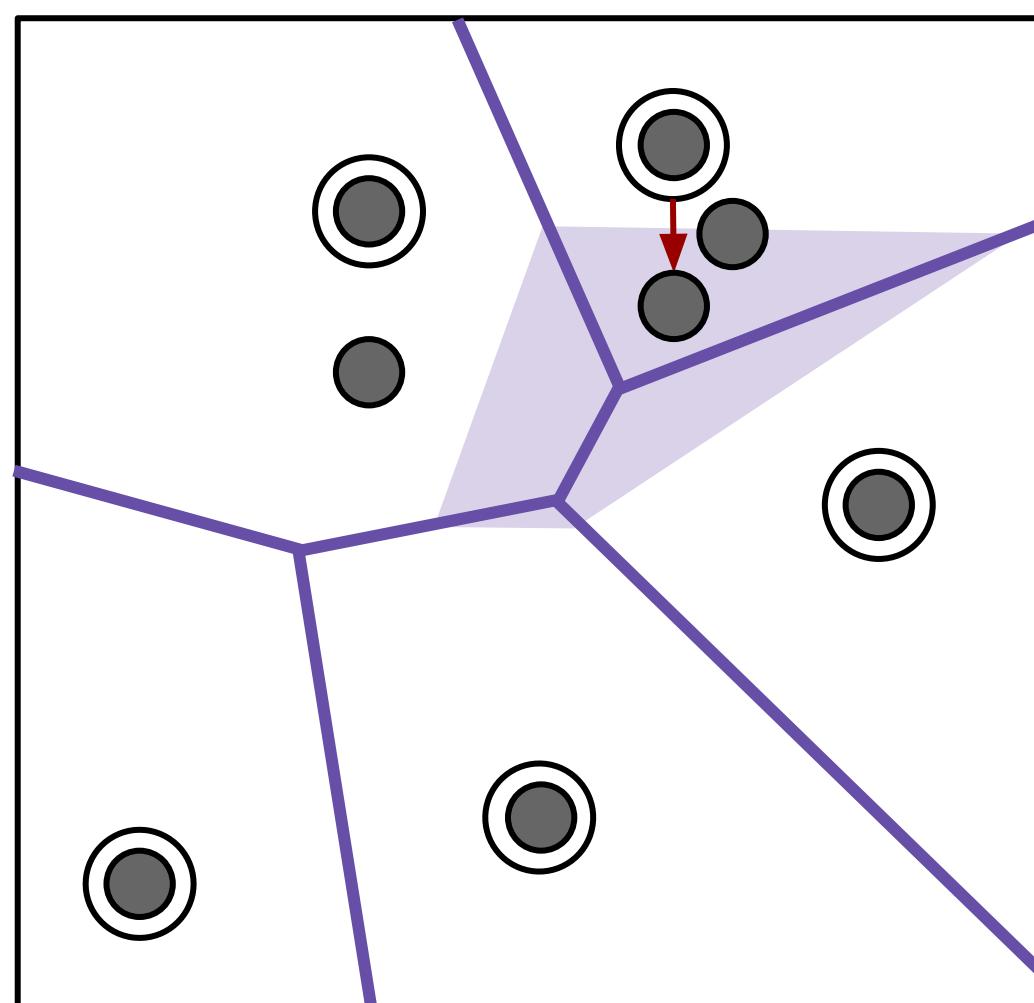
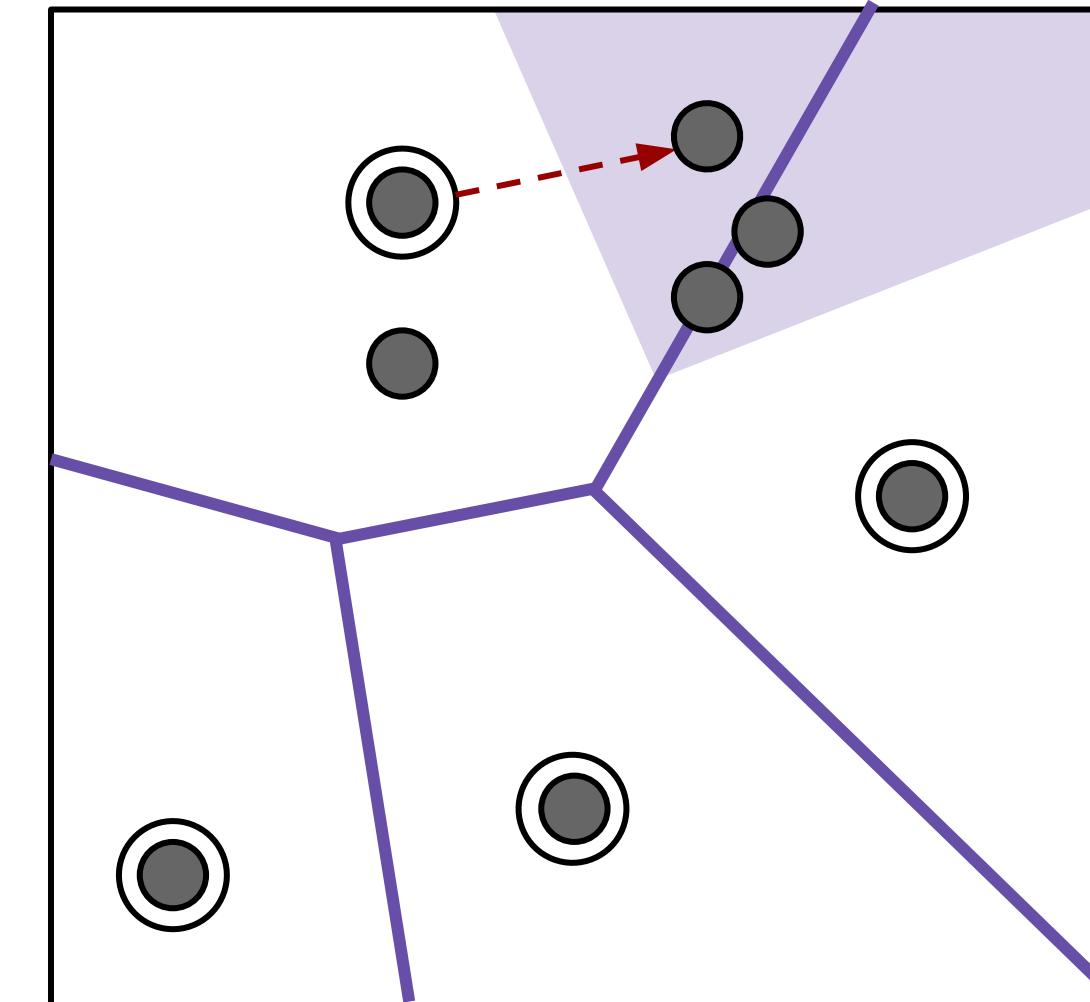
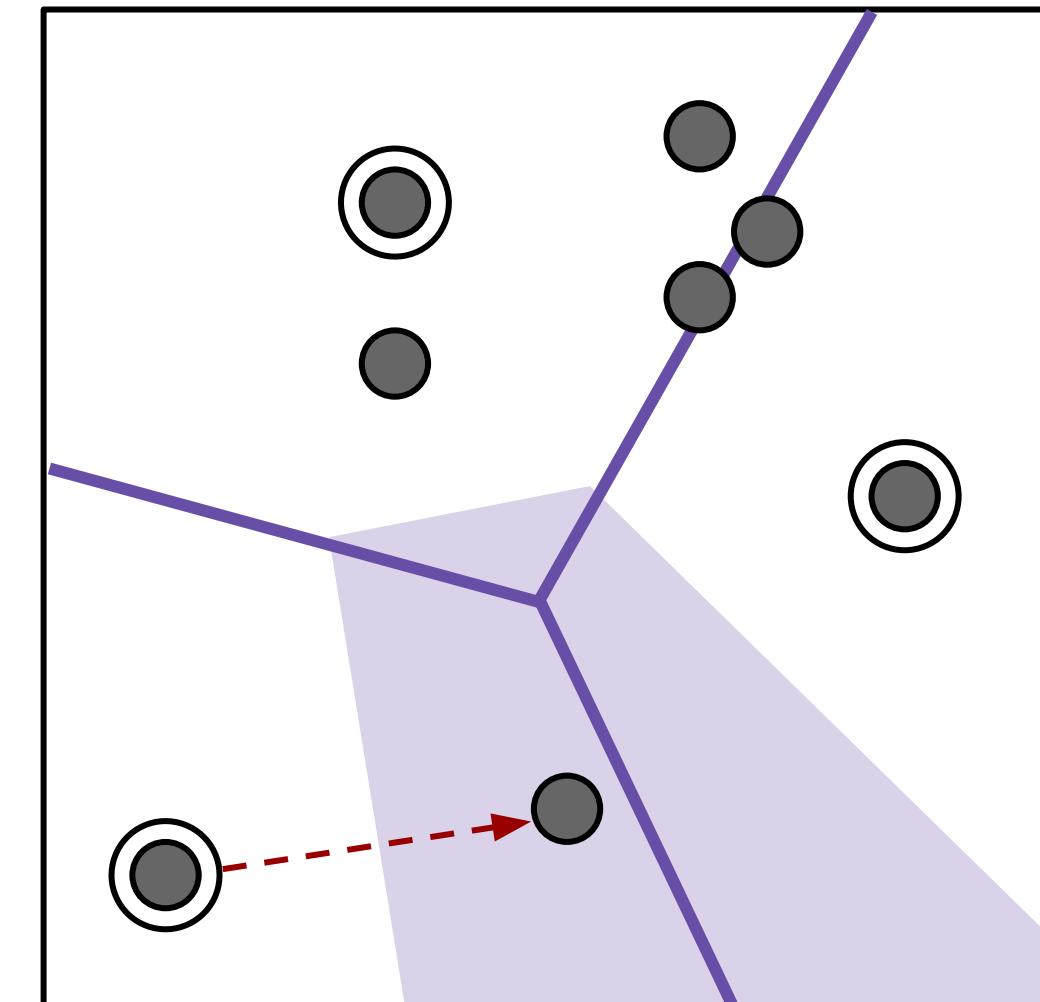
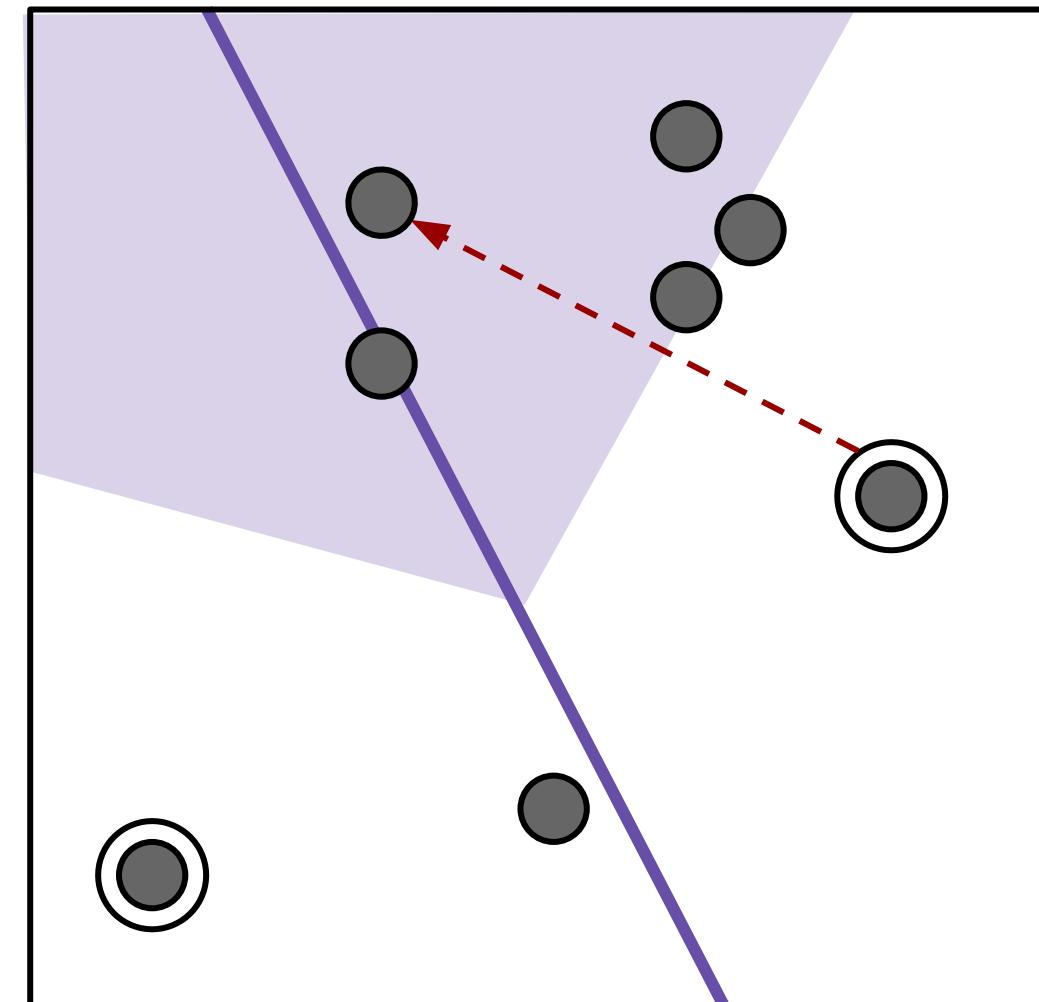
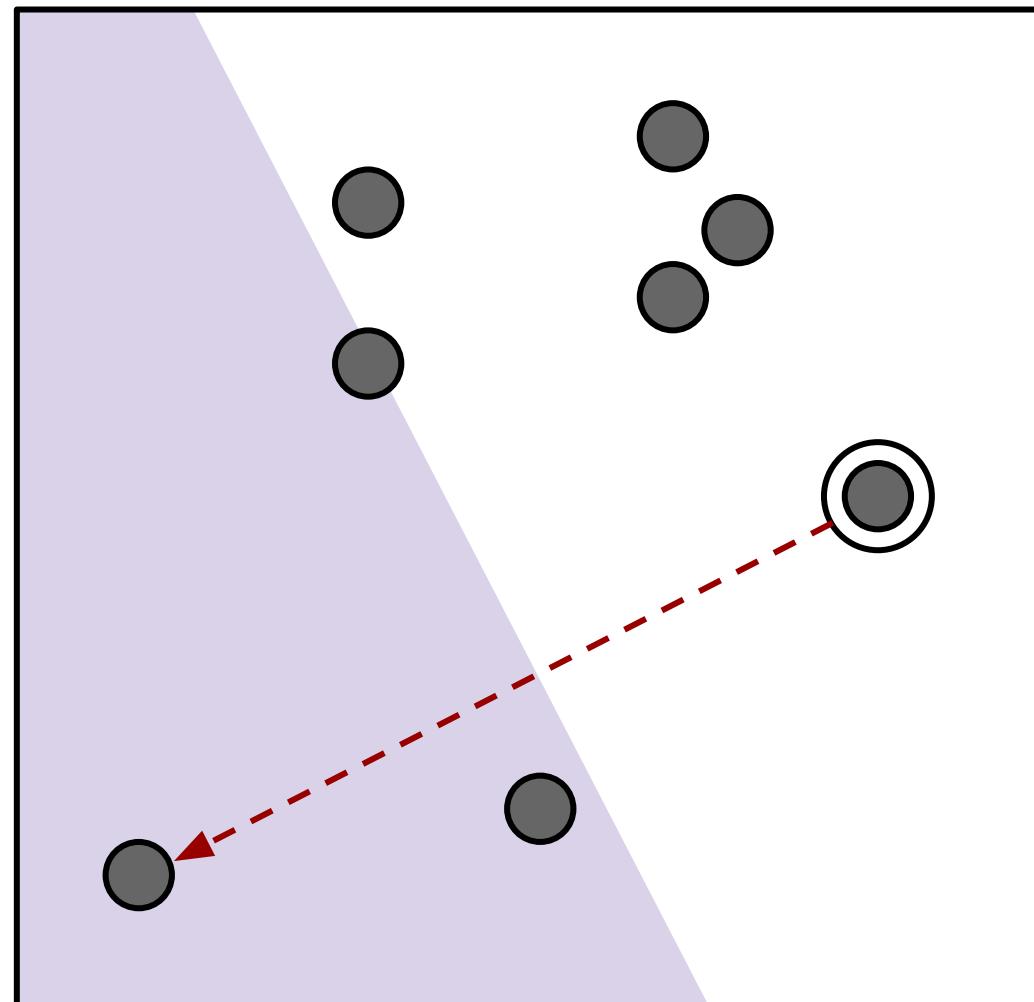


Insert a point



Point location
(Decide who moves)

Construct a Greedy Permutation



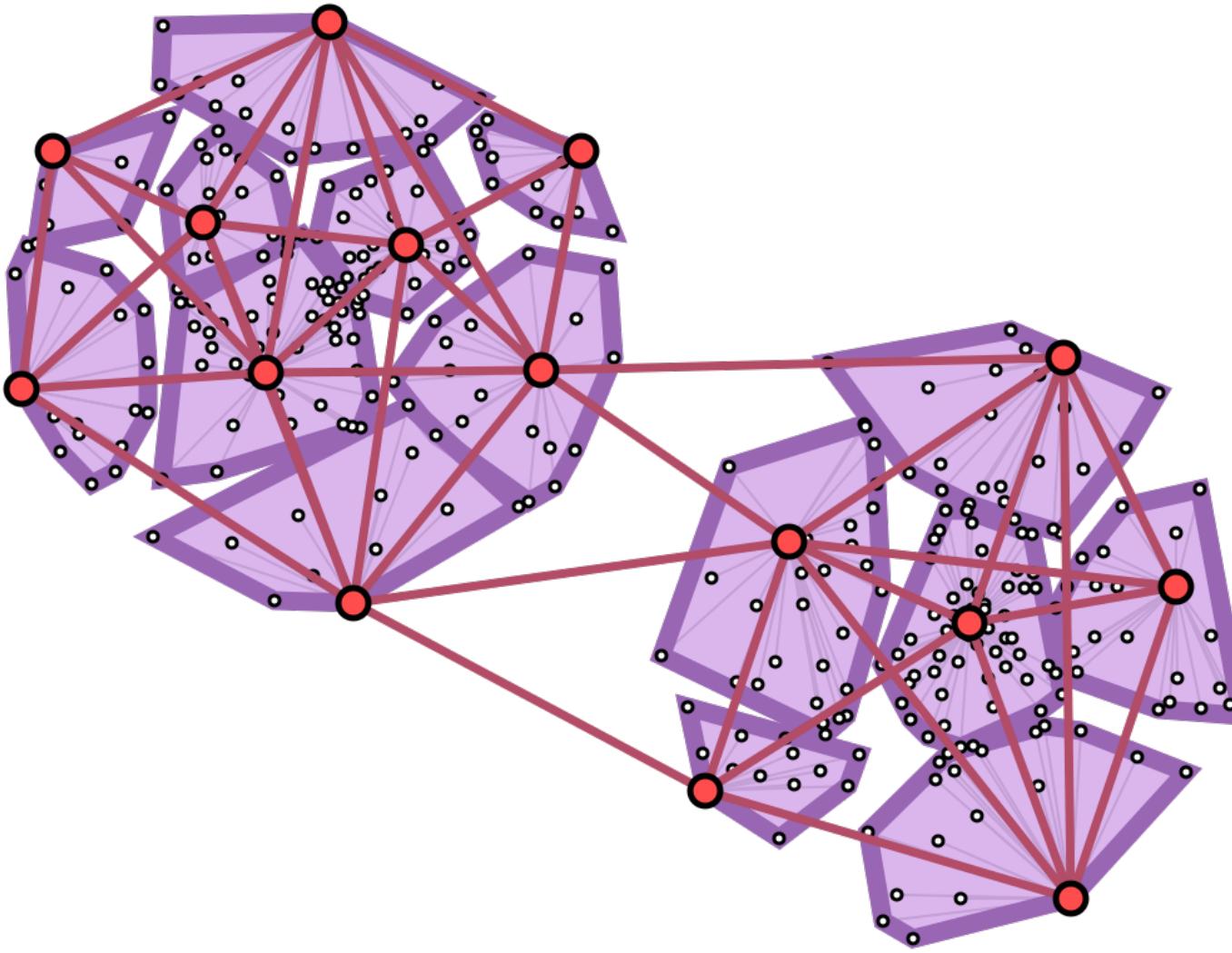
Improving The Naive Algorithm

- 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.

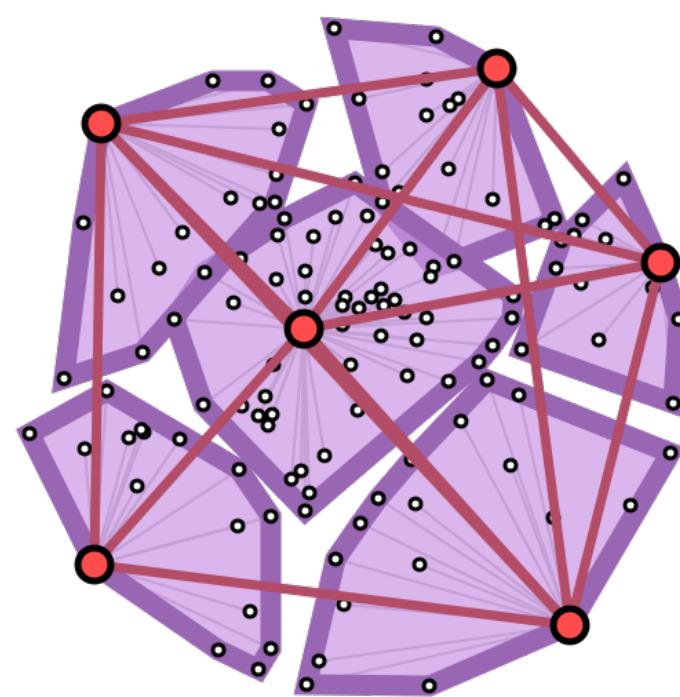
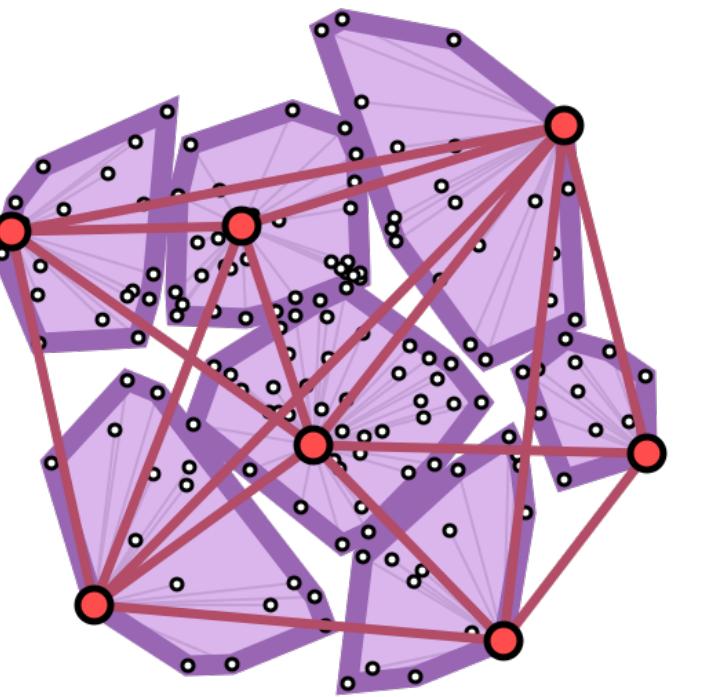
Neighbor Graph

- In Clarkson's sb data structure, “close enough” information is stored as edges in a graph.
- Har-Peled and Mendel show that Clarkson's algorithm runs in $O(n \log \Delta)$.
 - Δ is the ratio of the diameter to distance of the closest pair

Neighbor Graph

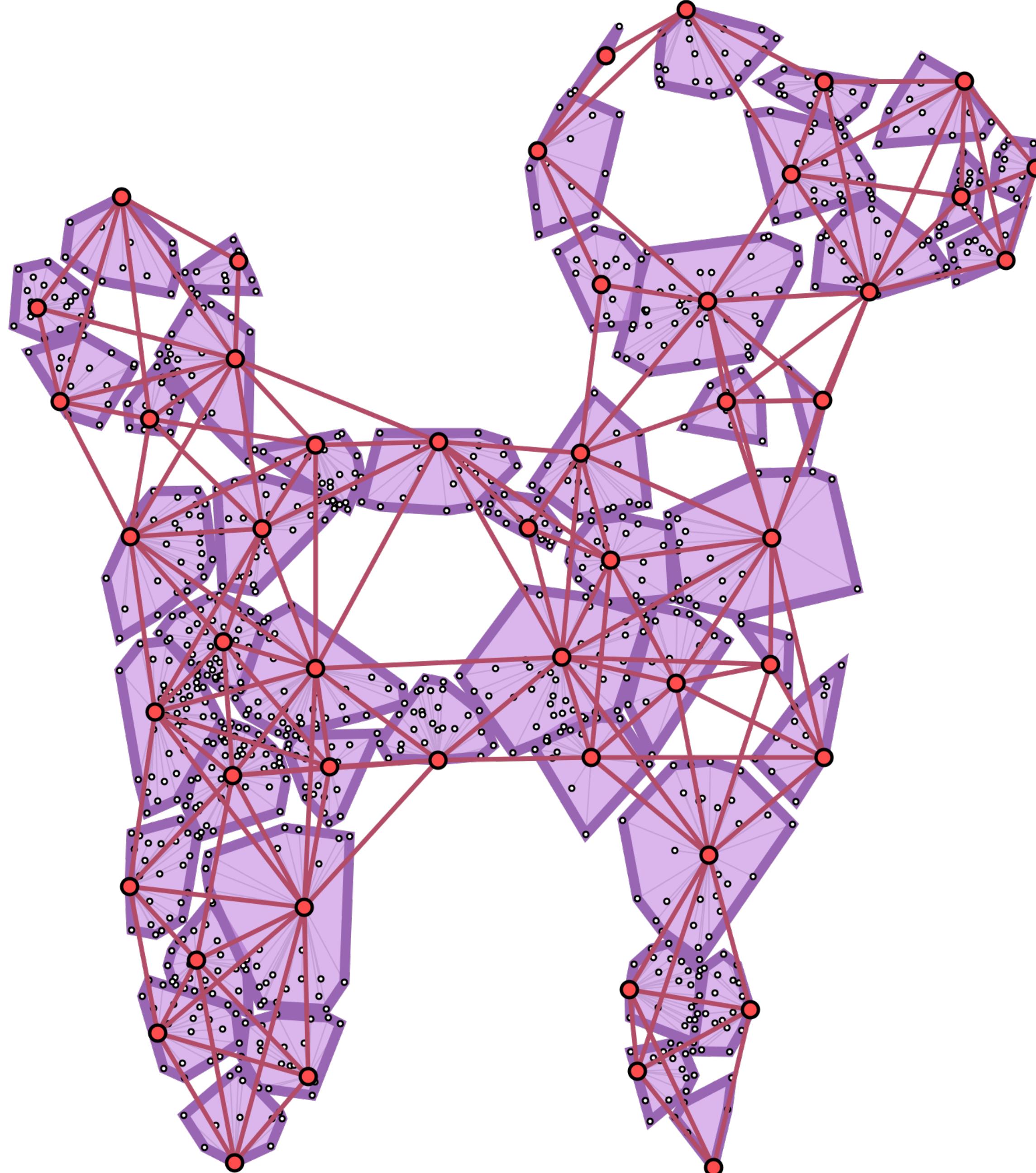


Avoids spending too much time checking far away points during the point location step



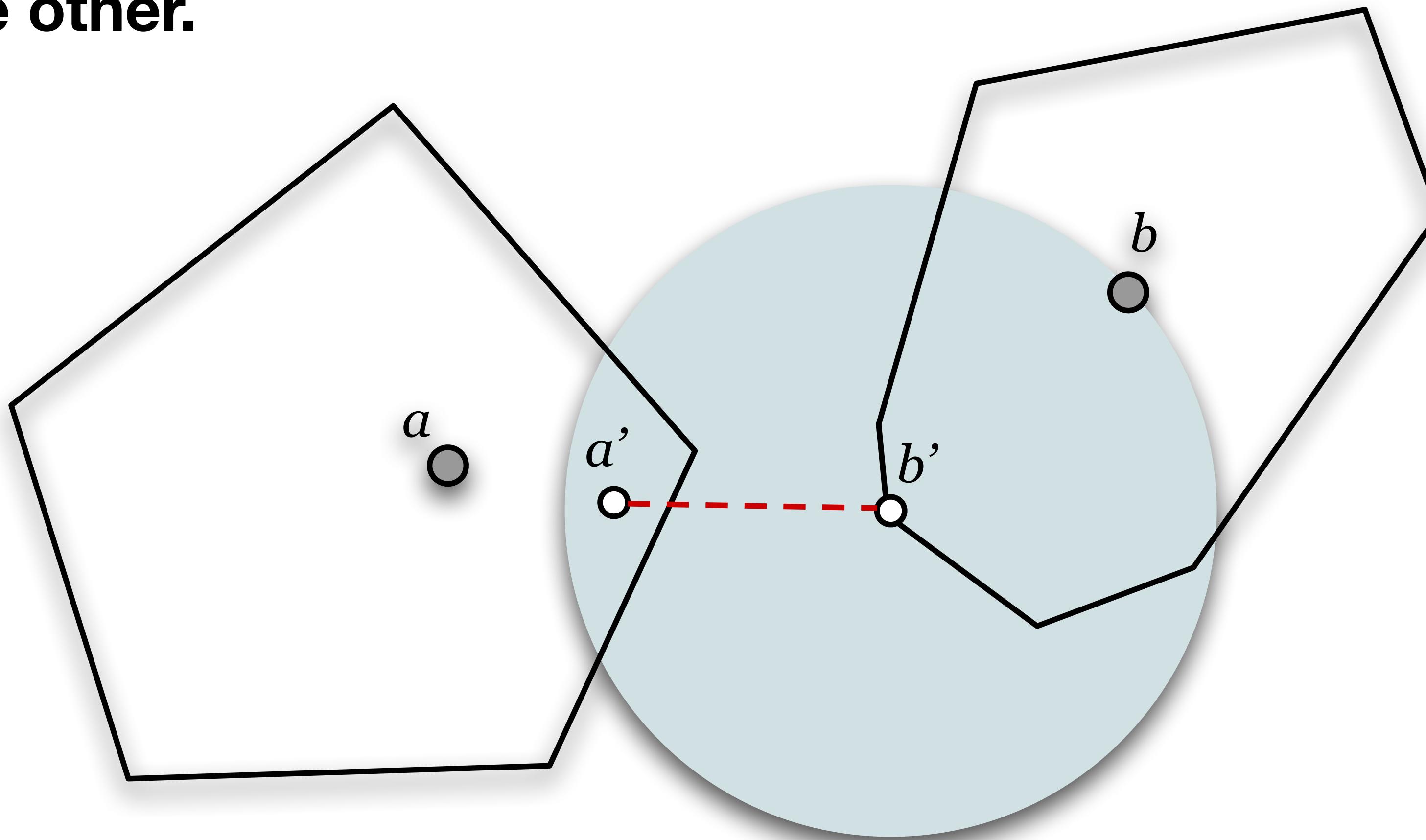
Neighbor Graph

Avoids spending too much time checking far away points during the point location step



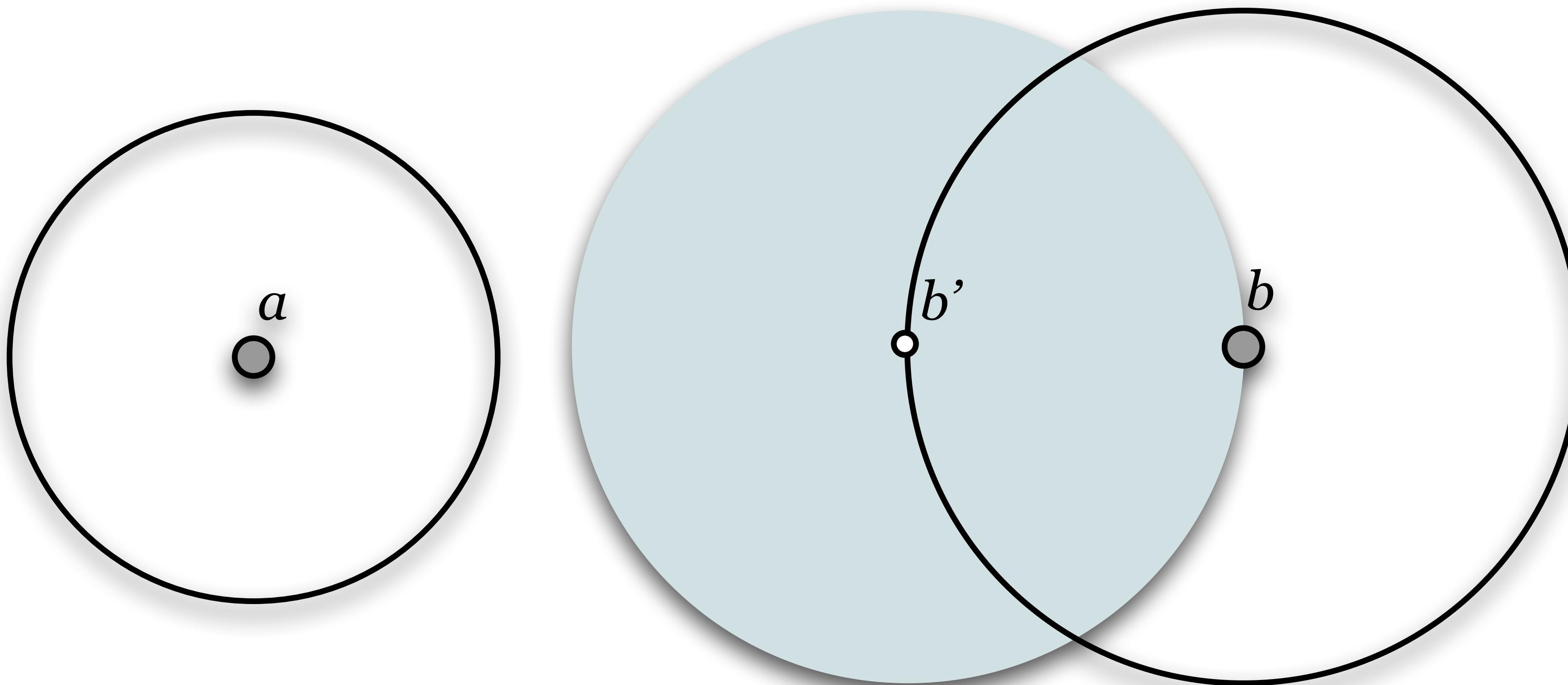
Neighbor Graph

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



Pruning Condition

- Approximate cells by their out-radii to prune unnecessary edges



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.

Some Technicalities

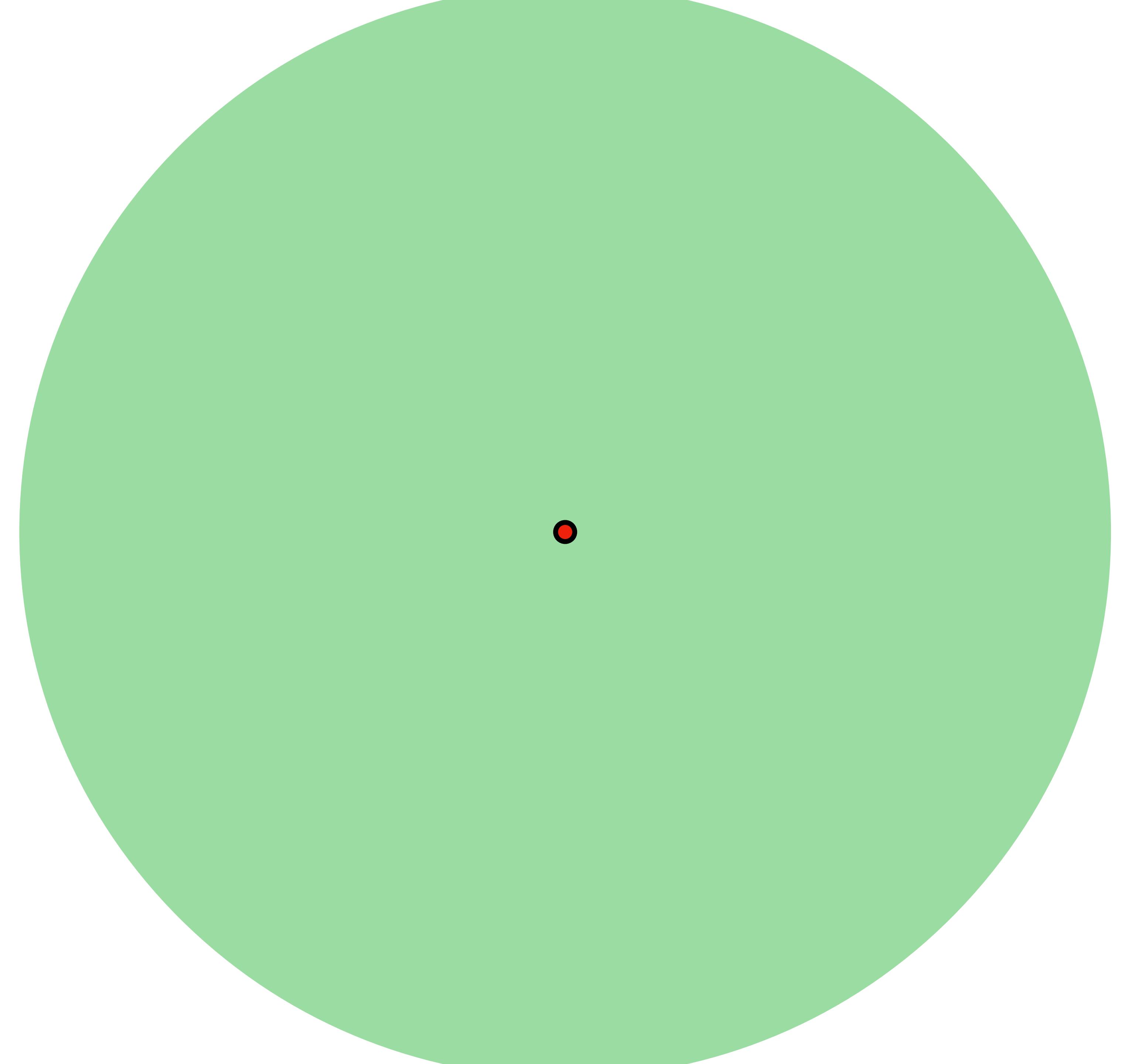
- 1. The heap now needs to store nodes instead of points.**
- 2. We use a bucket queue to achieve constant-time heap operations.**
- 3. Storing clusters in cells introduces some sources of approximation during construction.**

Storing Clusters

- Use a greedy tree as the input
- Start by storing the root node in a cell
- Split nodes to find the approximate farthest point
- Split nodes that can't be unambiguously located

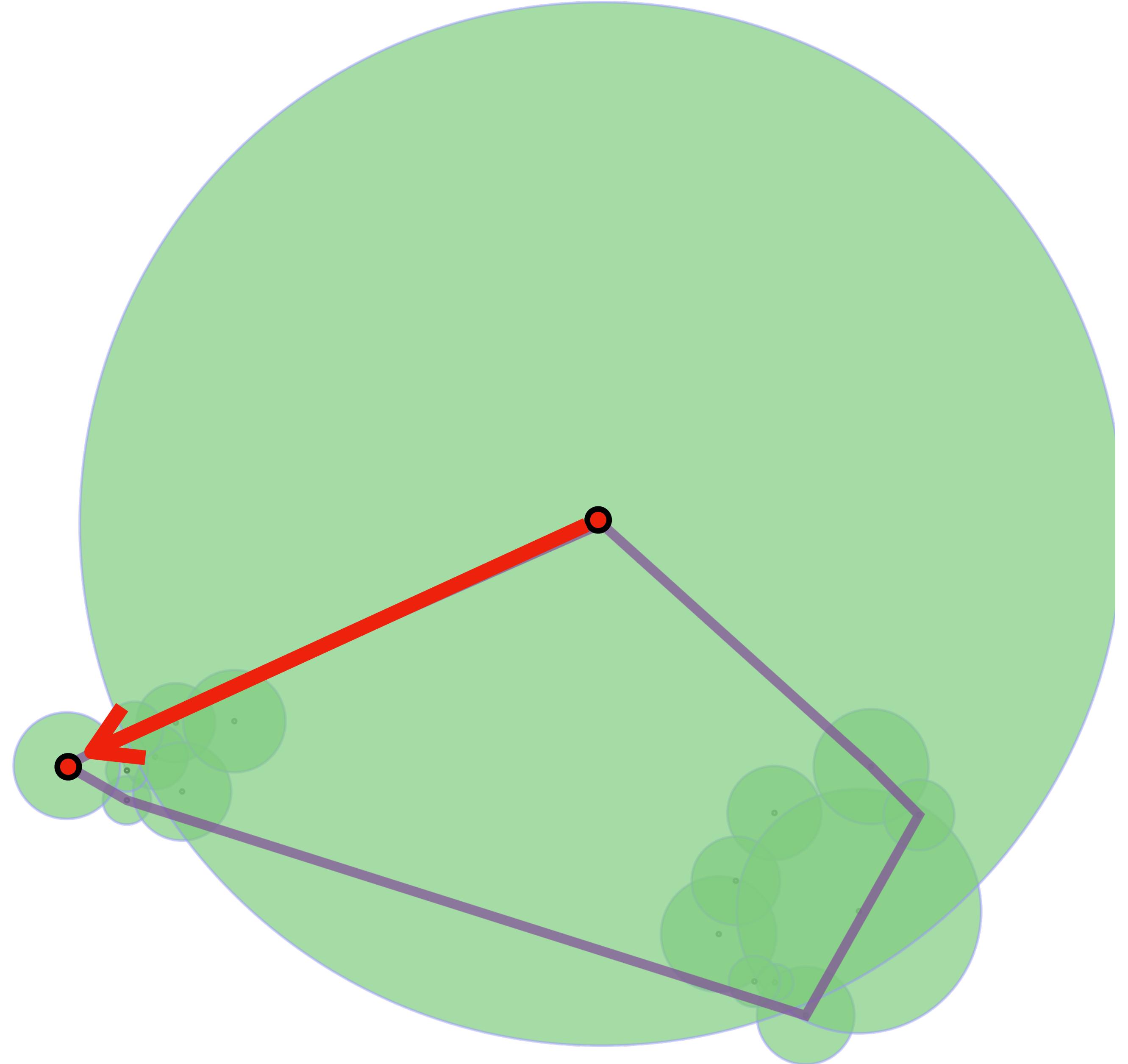
Storing Clusters

- Use a greedy as the input
- Start by storing the root node in a cell
- Split nodes to find the approximate farthest point
- Split nodes that can't be unambiguously located



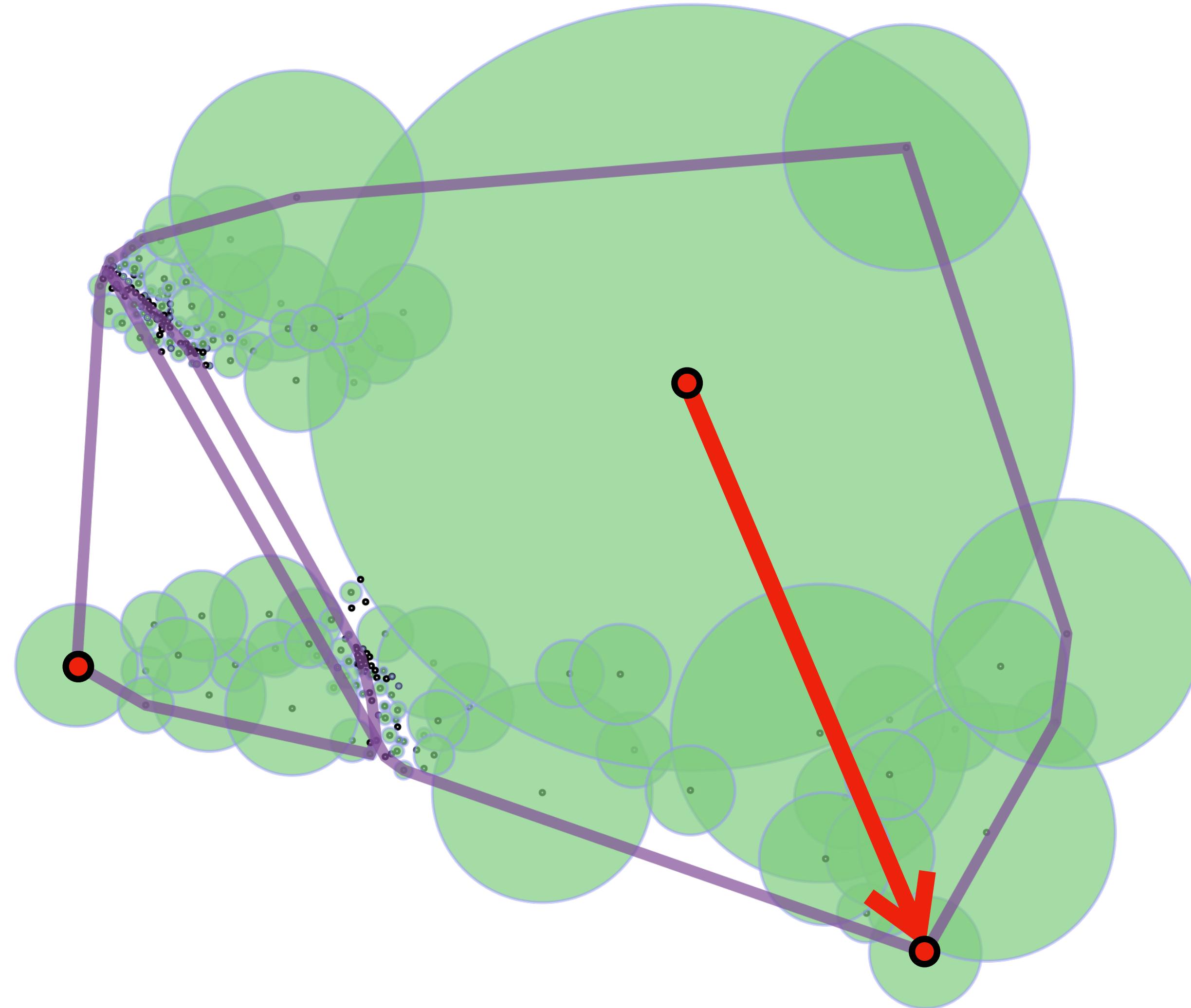
Storing Clusters

- Use a greedy as the input
- Start by storing the root node in a cell
- Split nodes to find the approximate farthest point
- Split nodes that can't be unambiguously located



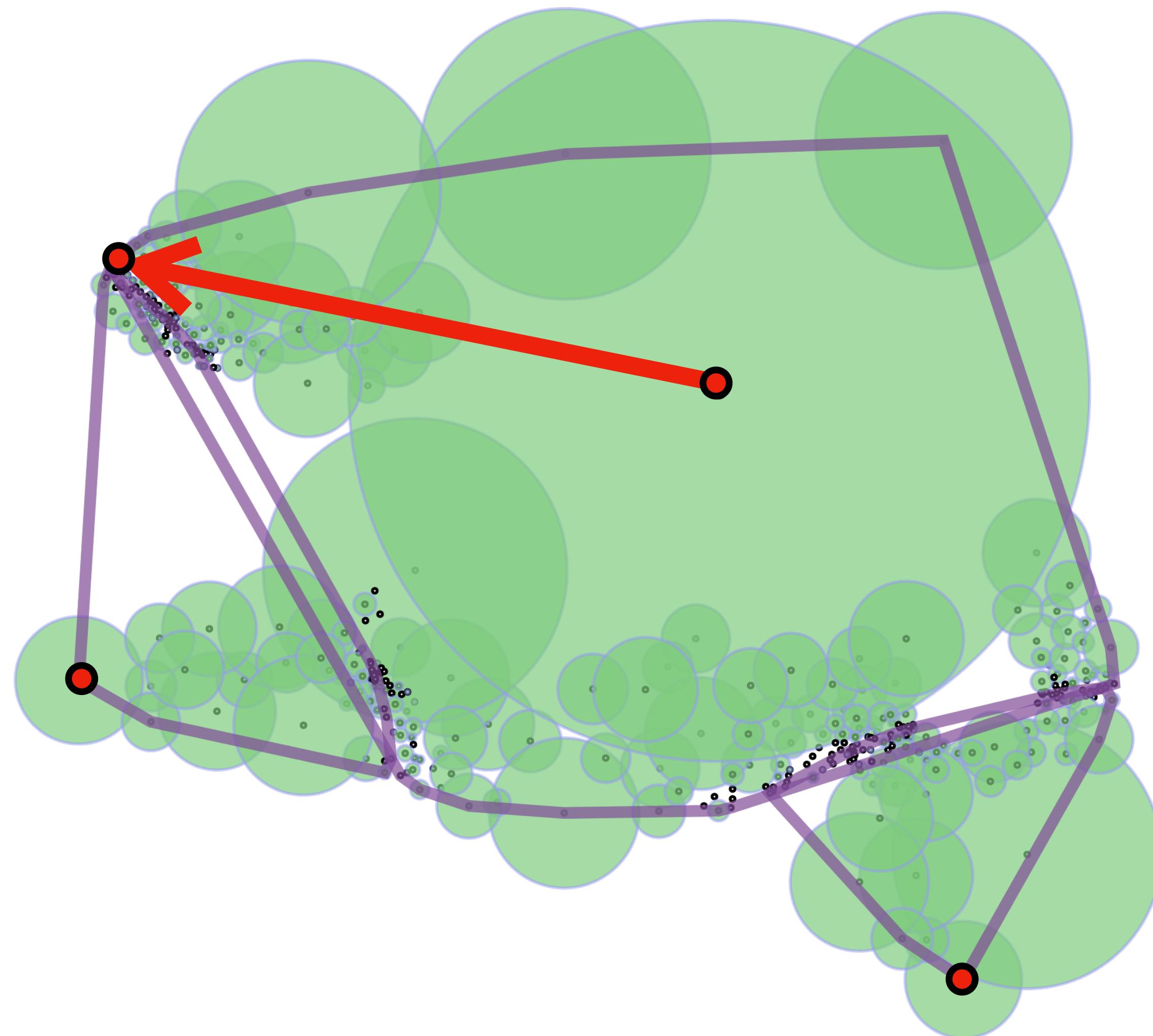
Storing Clusters

- Use a greedy as the input
- Start by storing the root node in a cell
- Split nodes to find the approximate farthest point
- Split nodes that can't be unambiguously located



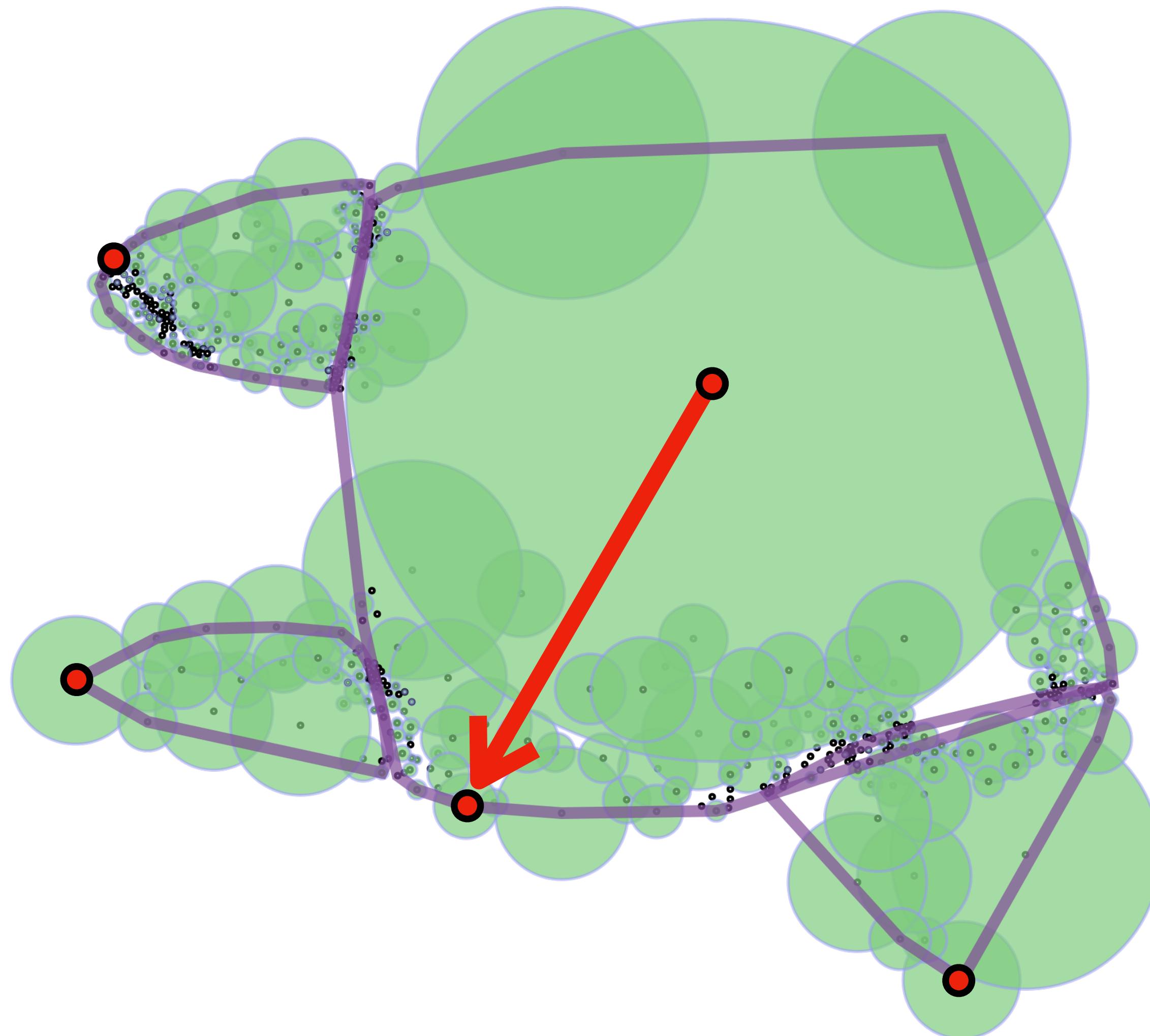
Storing Clusters

- Use a greedy as the input
- Start by storing the root node in a cell
- Split nodes to find the approximate farthest point
- Split nodes that can't be unambiguously located



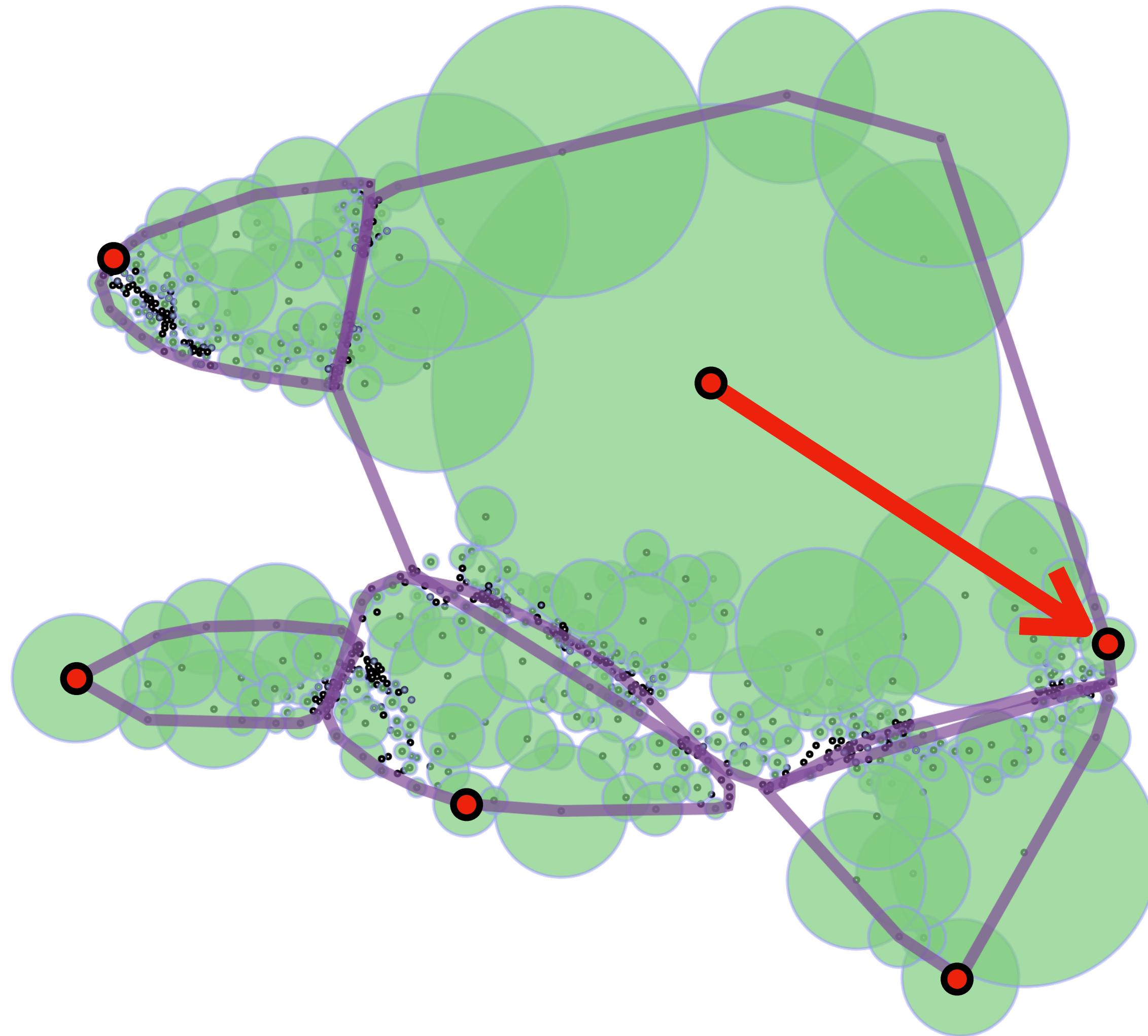
Storing Clusters

- Use a greedy as the input
- Start by storing the root node in a cell
- Split nodes to find the approximate farthest point
- Split nodes that can't be unambiguously located



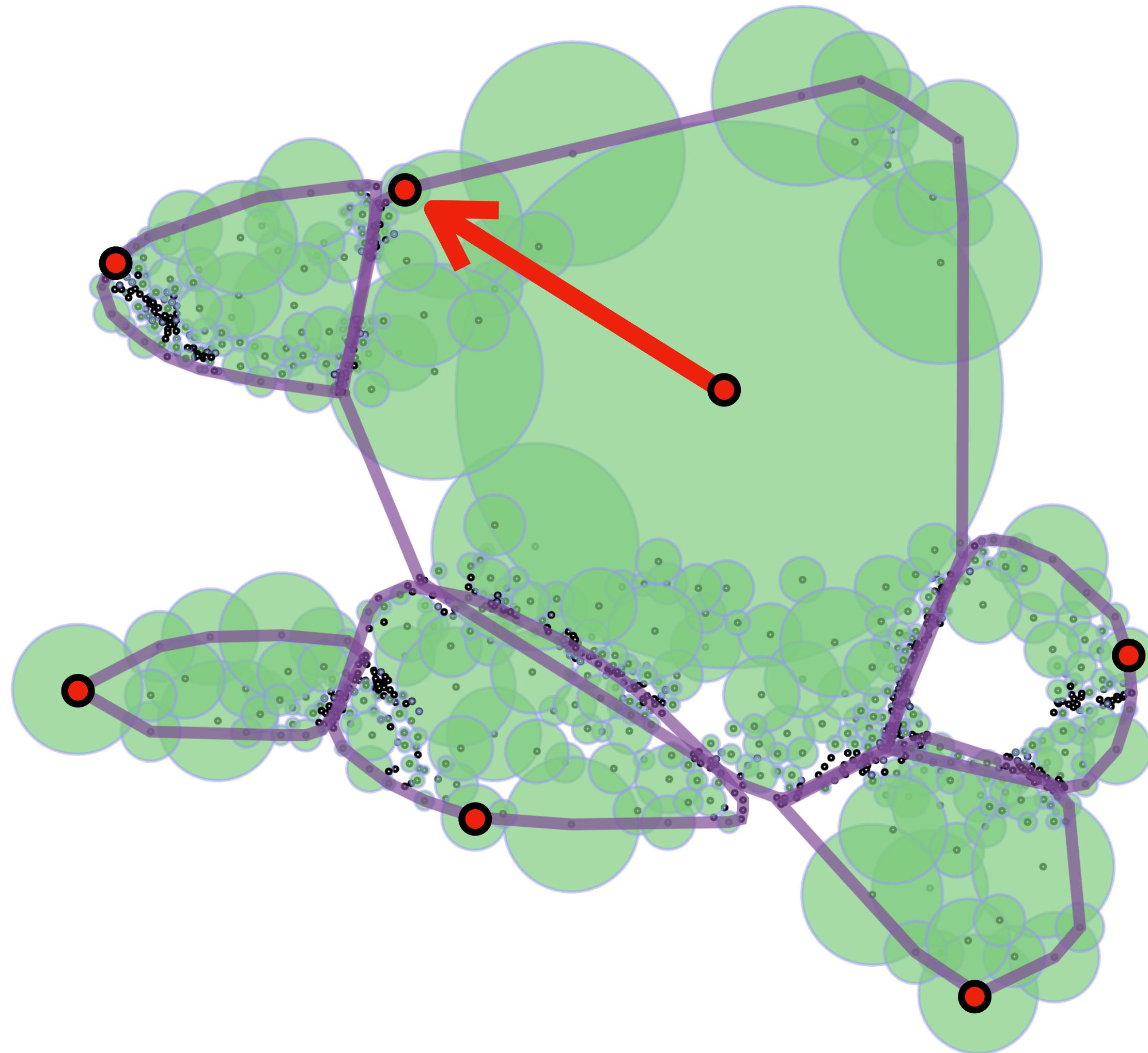
Storing Clusters

- Use a greedy as the input
- Start by storing the root node in a cell
- Split nodes to find the approximate farthest point
- Split nodes that can't be unambiguously located



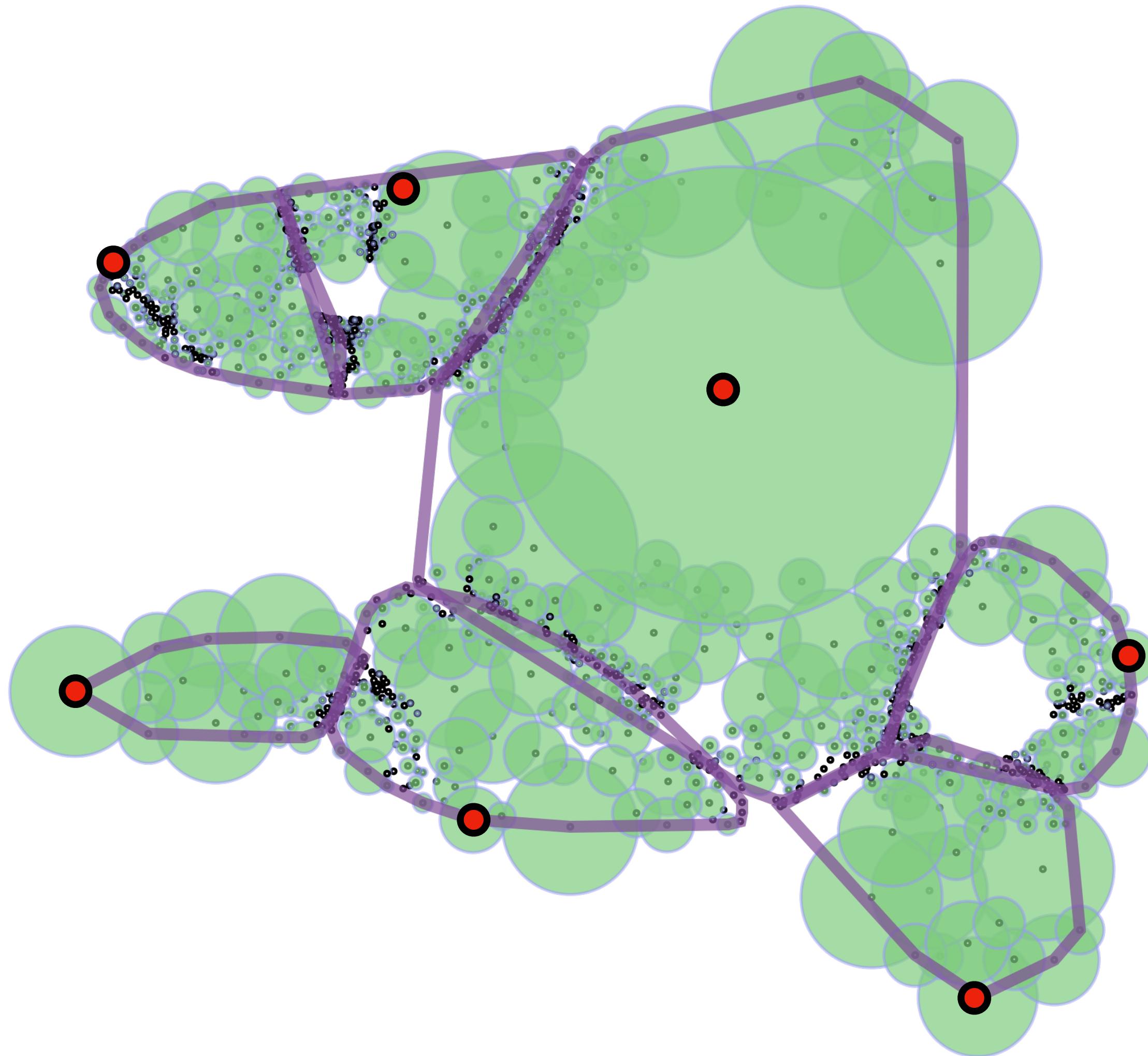
Storing Clusters

- Use a greedy as the input
- Start by storing the root node in a cell
- Split nodes to find the approximate farthest point
- Split nodes that can't be unambiguously located



Storing Clusters

- Use a greedy as the input
- Start by storing the root node in a cell
- Split nodes to find the approximate farthest point
- Split nodes that can't be unambiguously located



Linear Time

- Constant max degrees in the neighbor graph
- Constant cell complexity (clusters per cell)

Using Clusters to Cluster...?

Recursive Merging:

- The input is two greedy trees.
- Initialize the algorithm with one cell containing both root nodes.

This gives us a divide and conquer algorithm.

Conclusion

Theorem:

Using a recursive algorithm, a $(1 + \varepsilon)$ -greedy permutation can be constructed deterministically in $O(n \log n)$ time, and in $O(n)$ space.

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.

