Fast *k*-Nearest Neighbour Search via Prioritized DCI

Ke Li

Jitendra Malik

Berkeley
UNIVERSITY OF CALIFORNIA

Introduction

- The method of *k*-nearest neighbours is a fundamental building block of many machine learning methods.
- Problem definition: Given a database of *n* points and the query, find the *k* points that are closest to the query.

Notions of Dimensionality

- The hardness of a dataset can be characterized using two notions of dimensionality.
  - Ambient dimensionality: the dimensionality of the space that contains the data points.
  - Intrinsic dimensionality: can be roughly thought of as the dimensionality of the data manifold.

Intrinsic Dimensionality

- *Definition:*

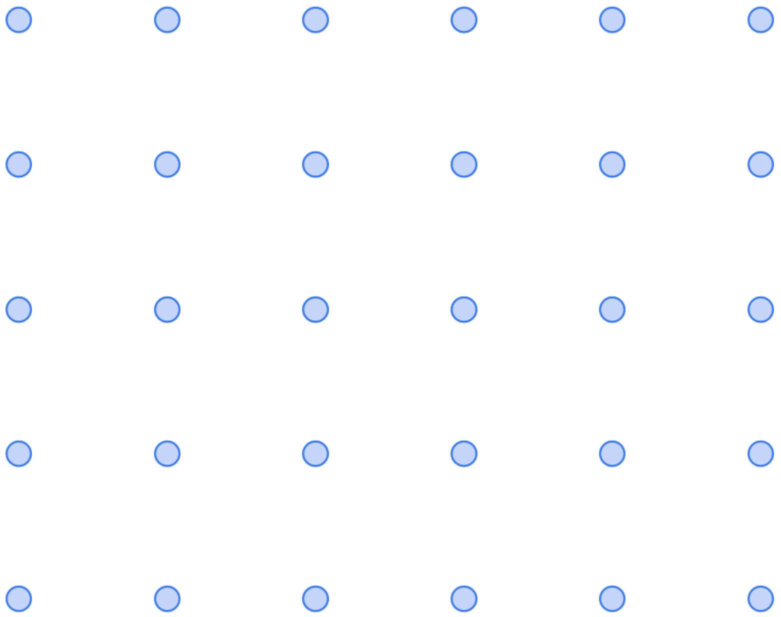
  A dataset D has intrinsic dimensionality[1] d' if for all $r > 0$, $\alpha > 1$ and p such that $|B_p(r)| \geq k$,

$$|B_p(\alpha r)| \leq \alpha^{d'} \, |B_p(r)|$$

[1] This is also known as the expansion dimension or the KR-dimension.

Intrinsic Dimensionality

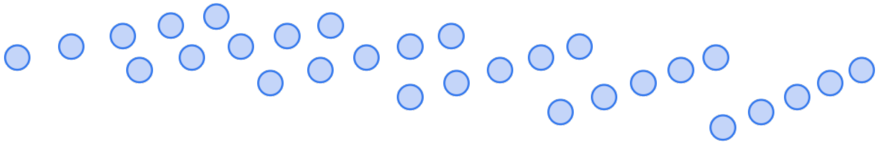
- A d'-dimensional uniform grid $\mathbb{Z}^{d'}$ has intrinsic dimensionality d'.



Berkeley UNIVERSITY OF CALIFORNIA

Intrinsic Dimensionality

- If it were embedded in a higher-dimensional space, it would retain its intrinsic dimensionality.



Intrinsic Dimensionality

- Equivalently:

$$\log_2 |B_p(\alpha r)| \leq d' \log_2 (\alpha r) + (\log_2 |B_p(r)| - d' \log_2 r)$$

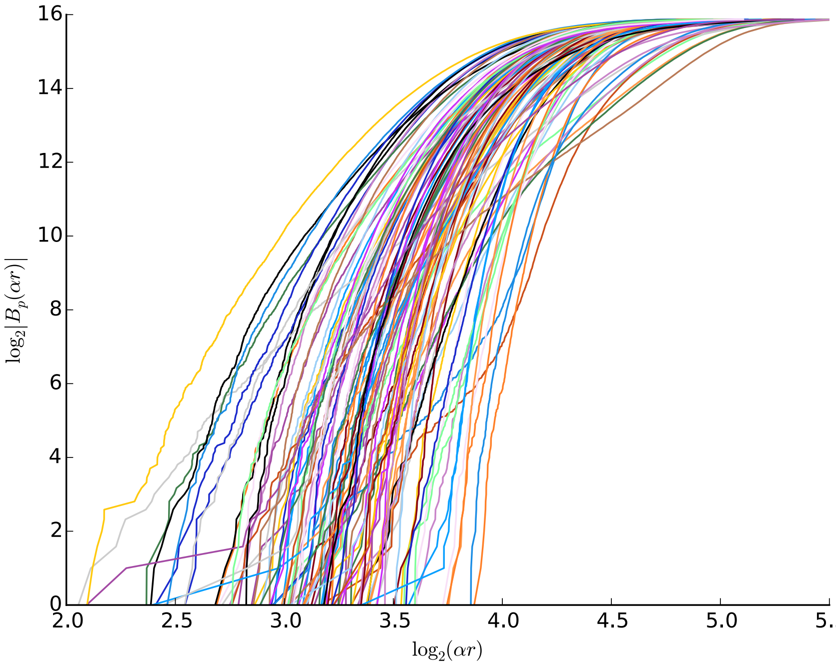
  - Plot $\log_2 |B_p(\alpha r)|$ against $\log_2 (\alpha r)$
  - Maximum slope upper bounds the intrinsic dimensionality.

Intrinsic Dimensionality

- Equivalently:

$$\log_2 |B_p(\alpha r)| \leq d' \log_2 (\alpha r) + (\log_2 |B_p(r)| - d' \log_2 r)$$

  - Plot $\log_2 |B_p(\alpha r)|$ against $\log_2 (\alpha r)$
  - Maximum slope upper bounds the intrinsic dimensionality.

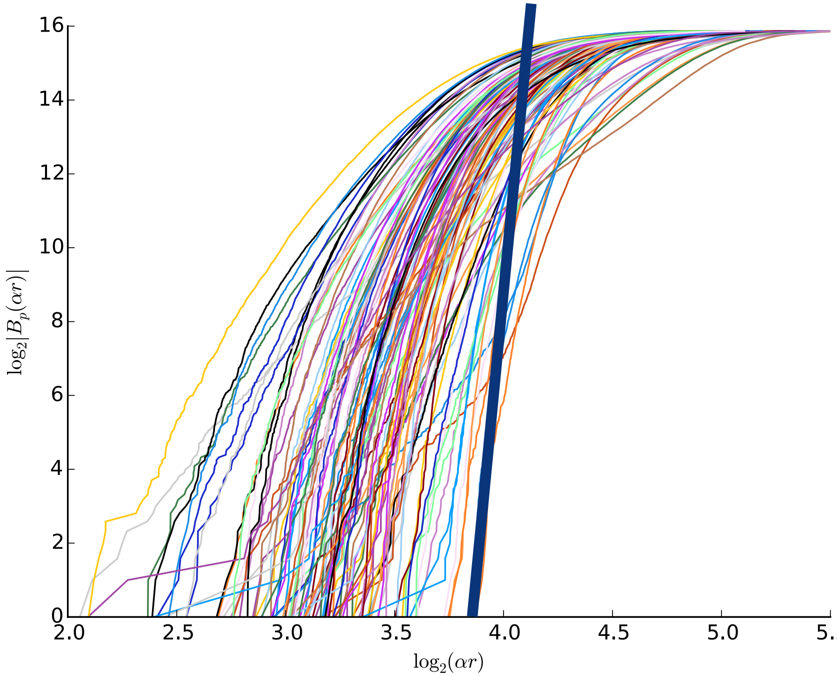


Intrinsic Dimensionality

- Equivalently:

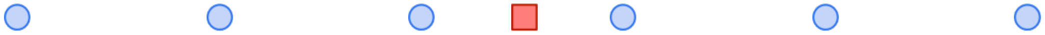
$$\log_2 |B_p(\alpha r)| \leq d' \log_2 (\alpha r) + (\log_2 |B_p(r)| - d' \log_2 r)$$

  - Plot $\log_2 |B_p(\alpha r)|$ against $\log_2 (\alpha r)$
  - Maximum slope upper bounds the intrinsic dimensionality.



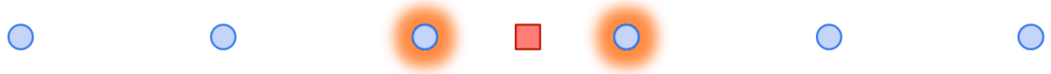
Why is High Dimensionality Hard?

$$d' = 1$$



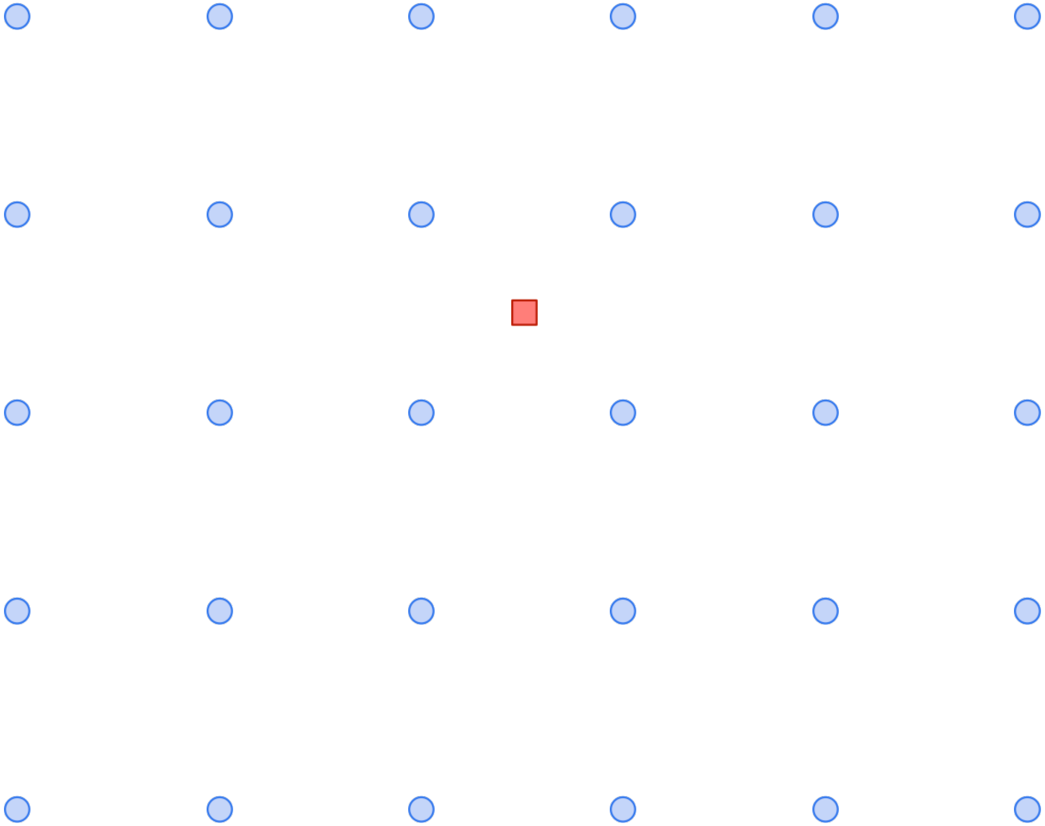
Why is High Dimensionality Hard?

$$d' = 1$$



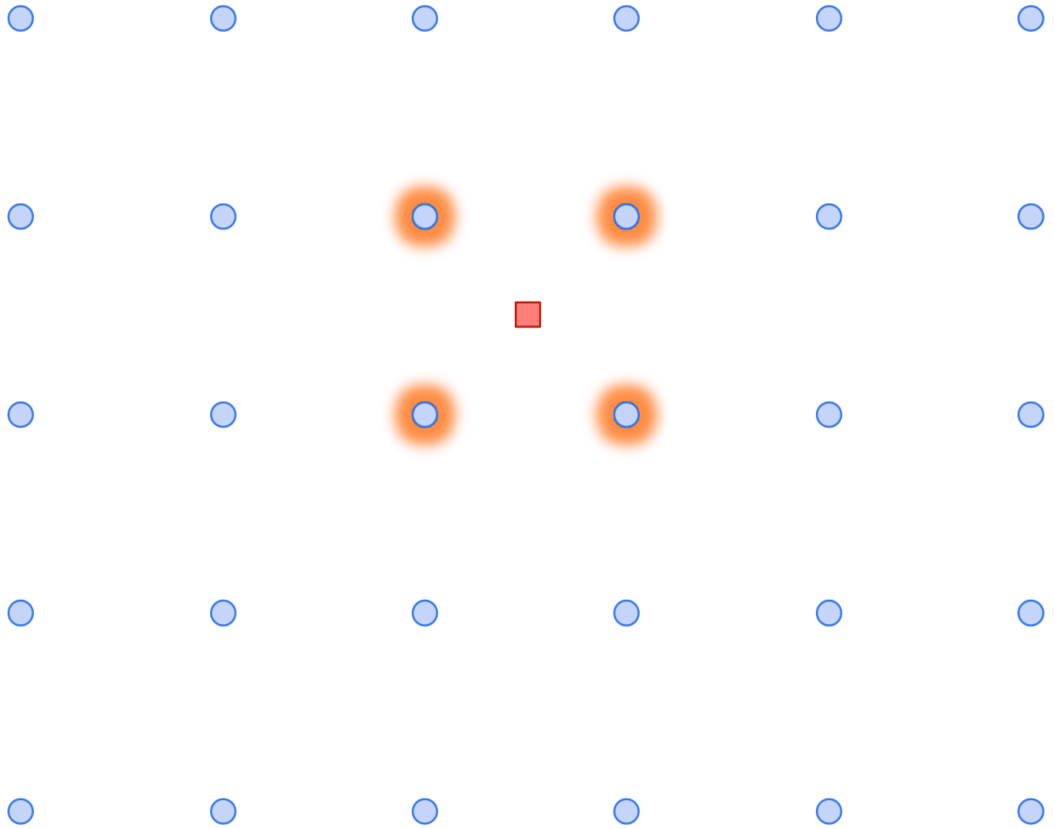
Why is High Dimensionality Hard?

$$d' = 2$$



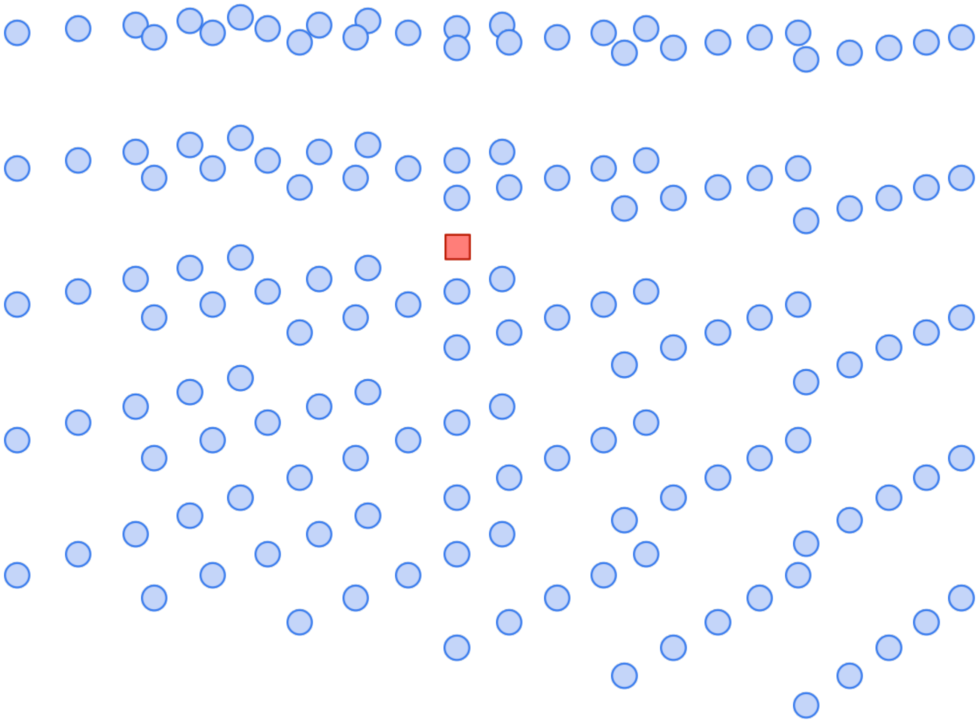
Why is High Dimensionality Hard?

$$d' = 2$$



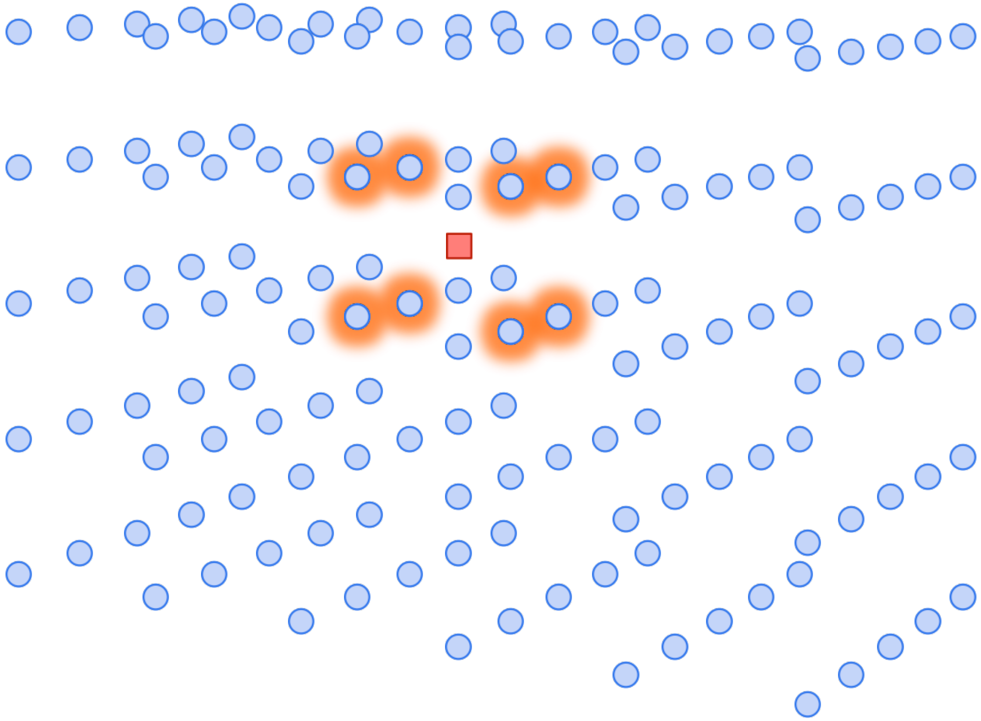
Why is High Dimensionality Hard?

$$d' = 3$$



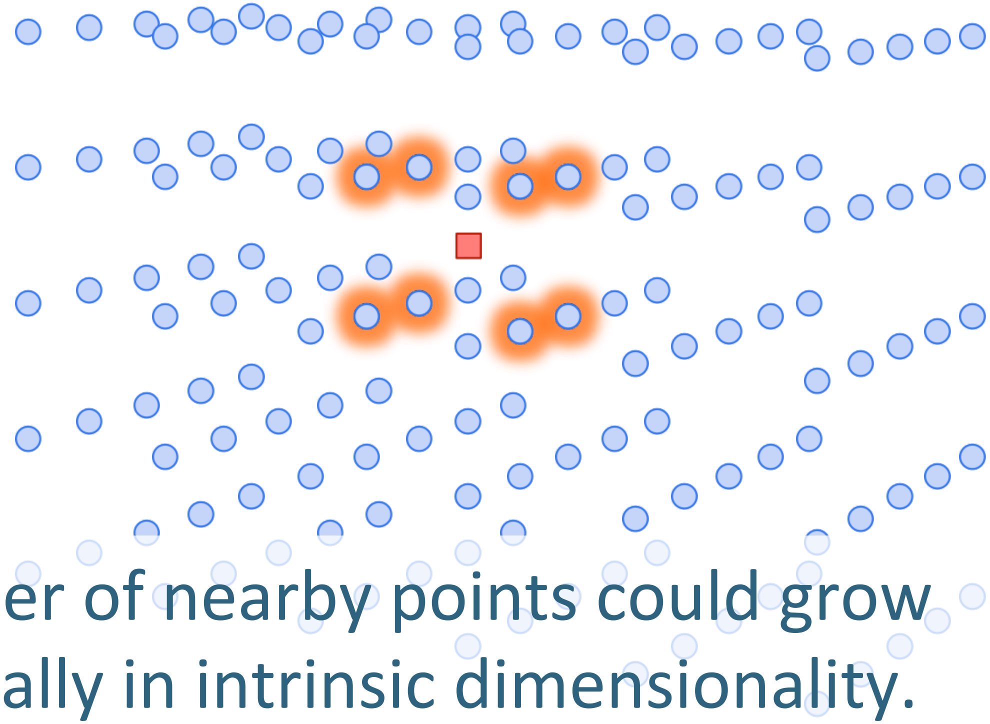
Why is High Dimensionality Hard?

$$d' = 3$$



Why is High Dimensionality Hard?

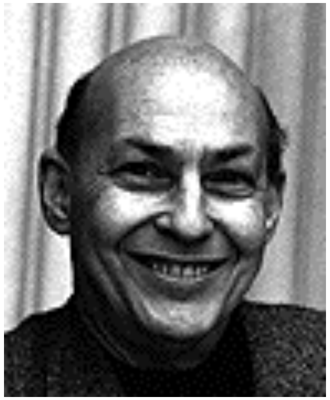
$$d' = 3$$

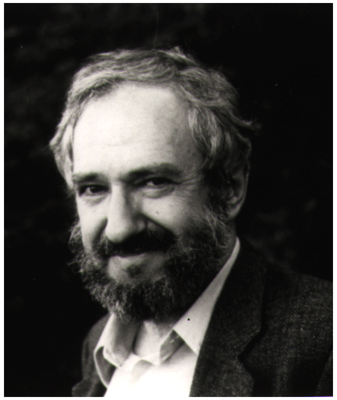


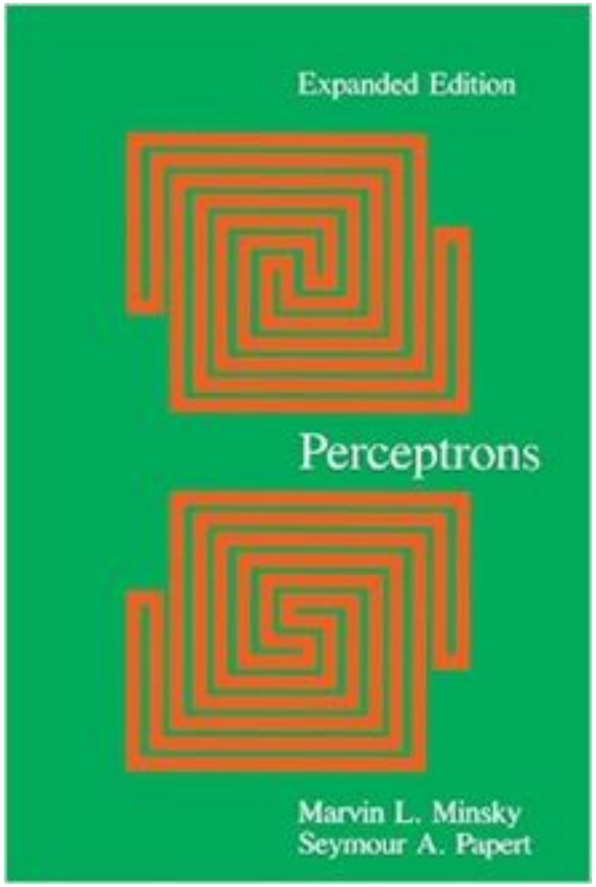
- The number of nearby points could grow exponentially in intrinsic dimensionality.

History

- The problem of nearest neighbour search was formalized by Cover & Hart (1967) and Minsky & Papert (1969) in their seminal book, *Perceptrons*.

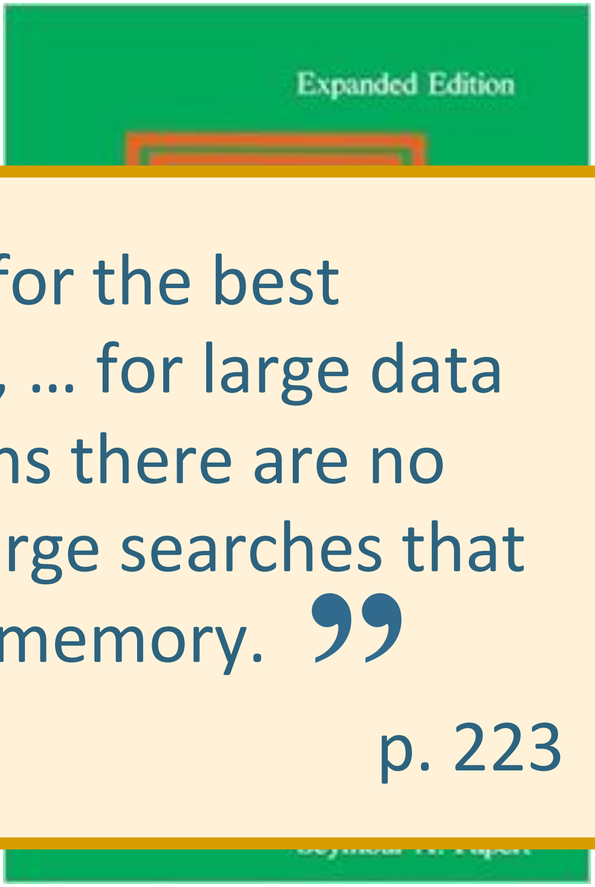






History

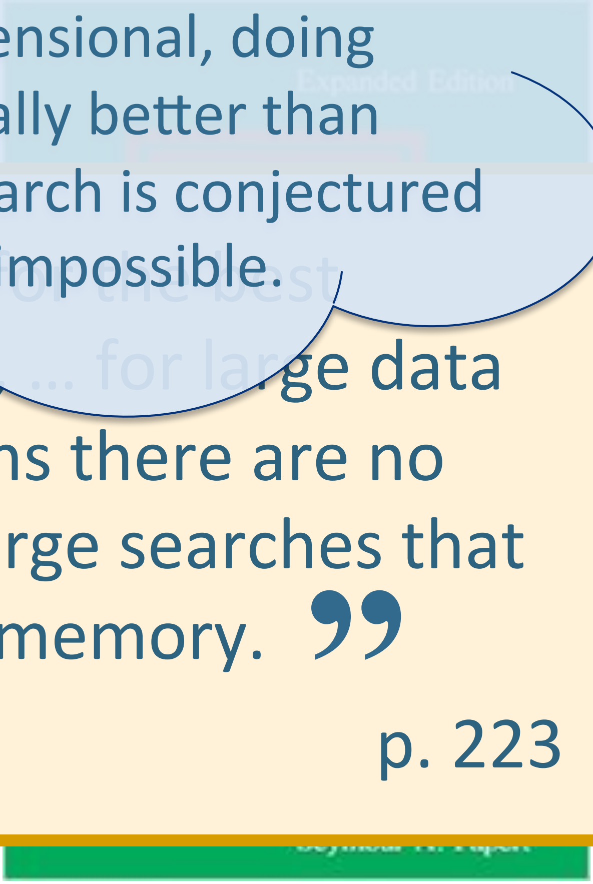
- The problem of nearest neighbour search was



> “We conjecture that even for the best possible $\mathbf{A}_{\text{file}}$ - $\mathbf{A}_{\text{find}}$ pairs, … for large data sets with long word lengths there are no practical alternatives to large searches that inspect large parts of the memory.”
>
> p. 223

History

- The problem of nearest neighbour search was

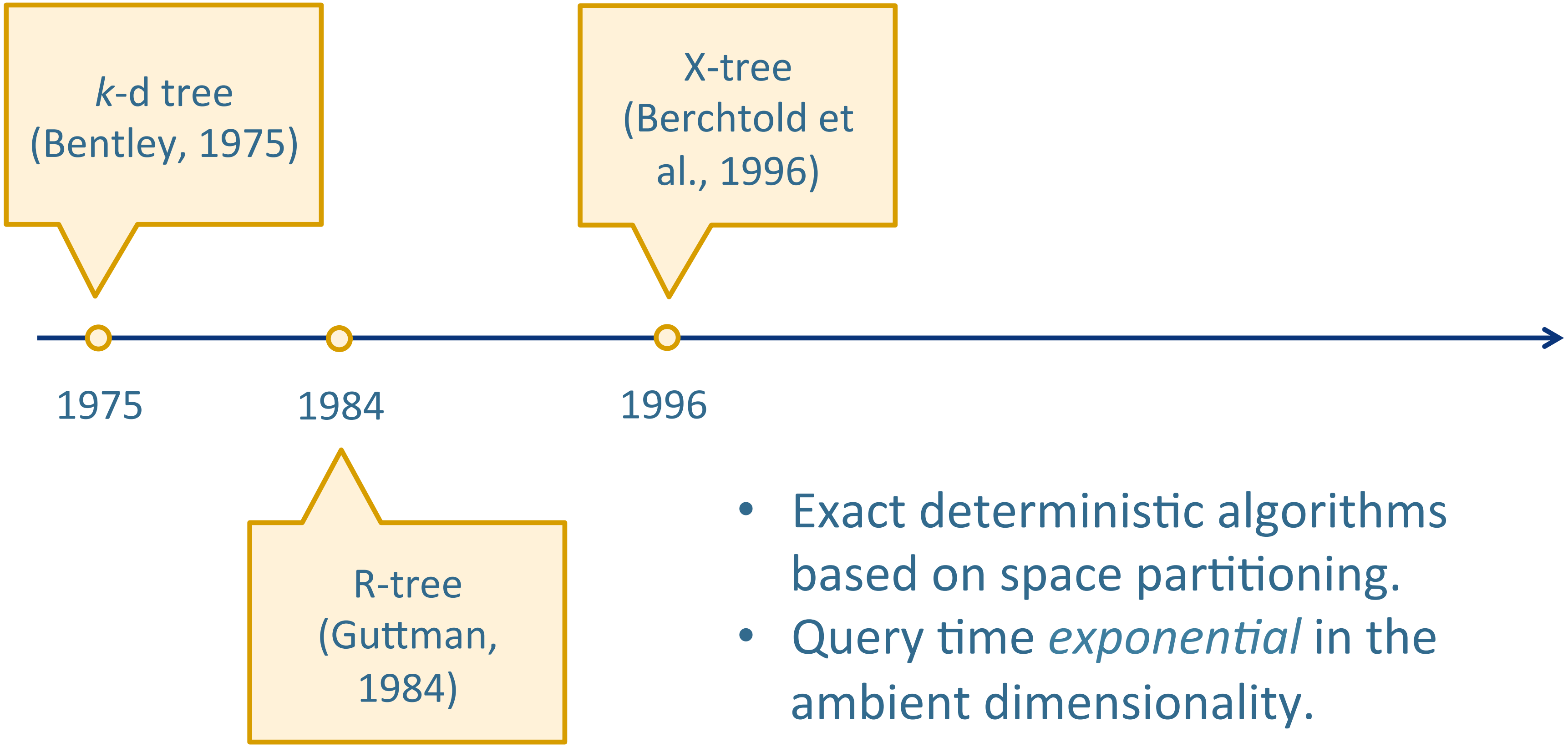


> We conjecture that even for the best possible $\mathbf{A}_{\text{file}}$ - $\mathbf{A}_{\text{find}}$ pairs, … for large data sets with long word lengths there are no practical alternatives to large searches that inspect large parts of the memory.

p. 223

In other words, even for the best choice of dataset and queries, when the dataset is large and high-dimensional, doing substantially better than exhaustive search is conjectured to be impossible.

The Curse of Dimensionality



k-d tree (Bentley, 1975)

R-tree (Guttman, 1984)

X-tree (Berchtold et al., 1996)

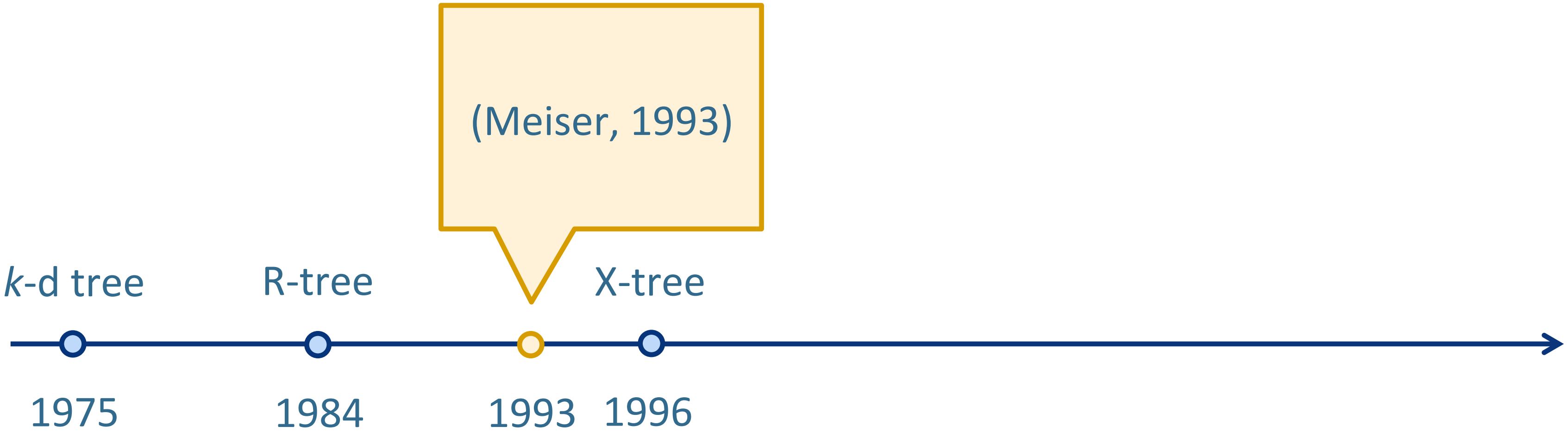
1975

1984

1996

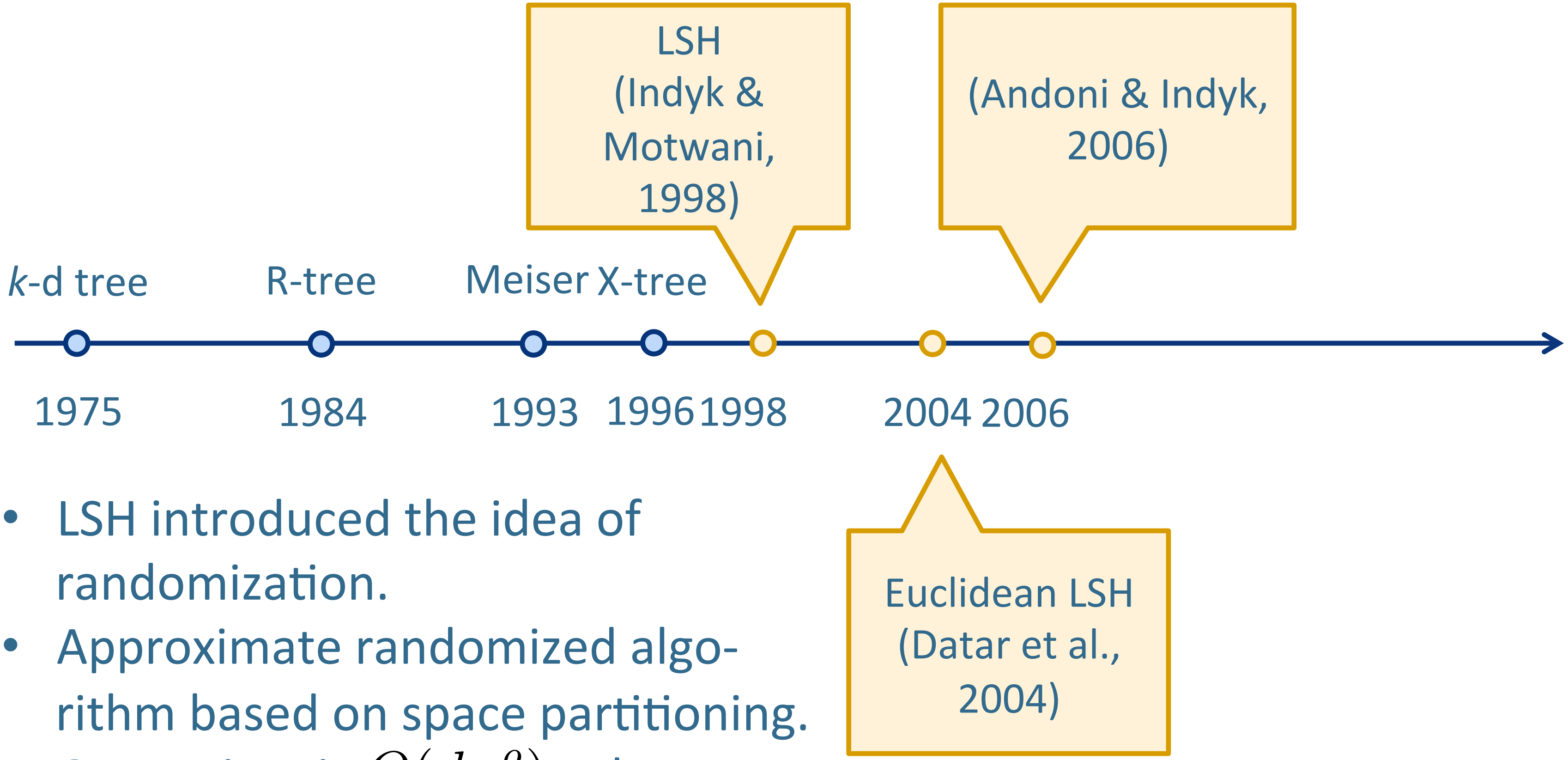
- Exact deterministic algorithms based on space partitioning.
- Query time *exponential* in the ambient dimensionality.

The Curse of Dimensionality



- Exact deterministic algorithm.
- Query time *polynomial* in ambient dimensionality.
- Space complexity *exponential* in ambient dimensionality.

The Curse of Dimensionality



LSH (Indyk & Motwani, 1998)

(Andoni & Indyk, 2006)

k-d tree

R-tree

Meiser

X-tree

1975

1984

1993

1996

1998

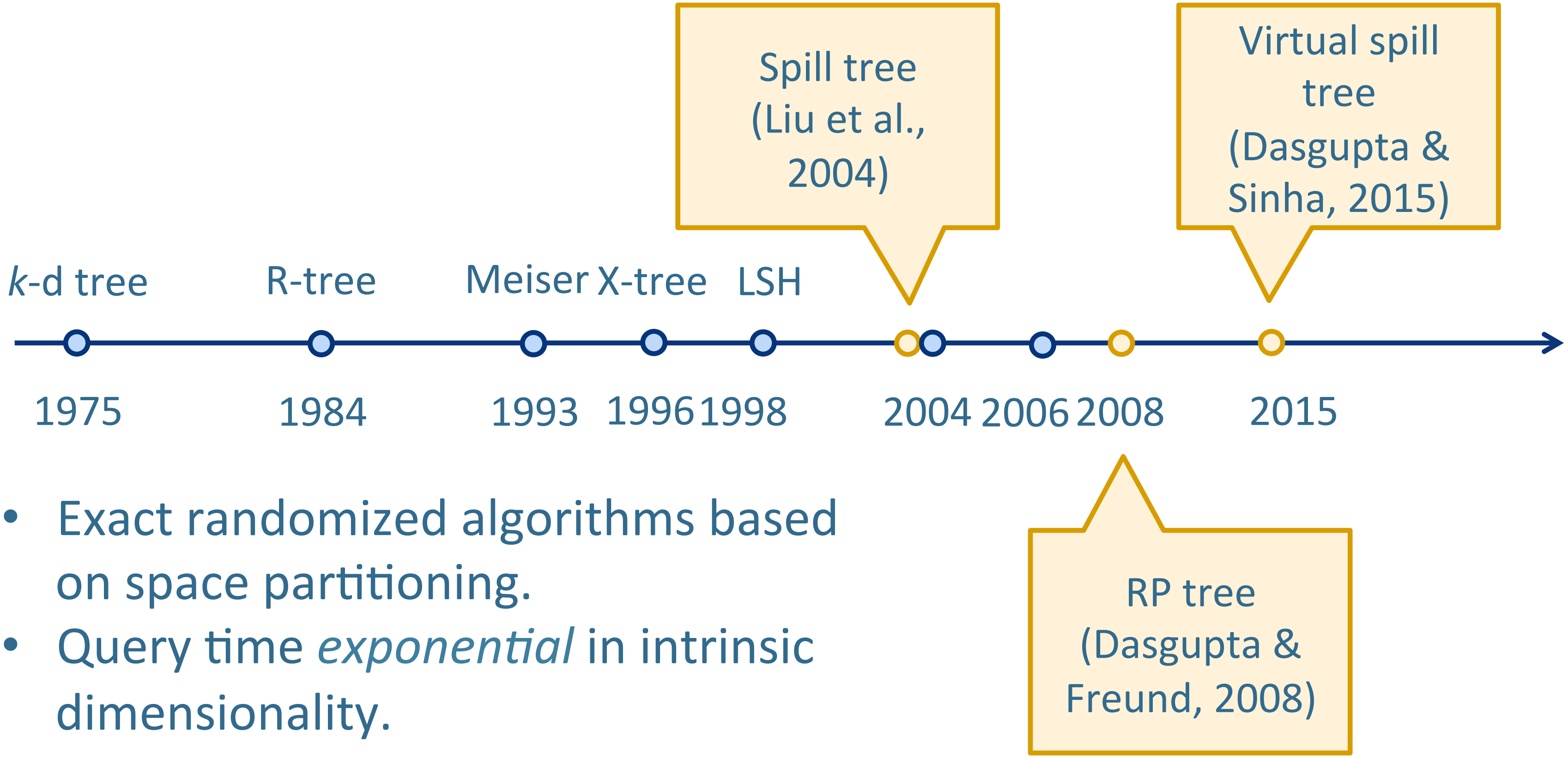
2004

2006

Euclidean LSH (Datar et al., 2004)

- LSH introduced the idea of randomization.
- Approximate randomized algorithm based on space partitioning.
- Query time is $O(dn^\rho)$, where $\rho \approx 1/(1+\epsilon)^2$.

The Curse of Dimensionality



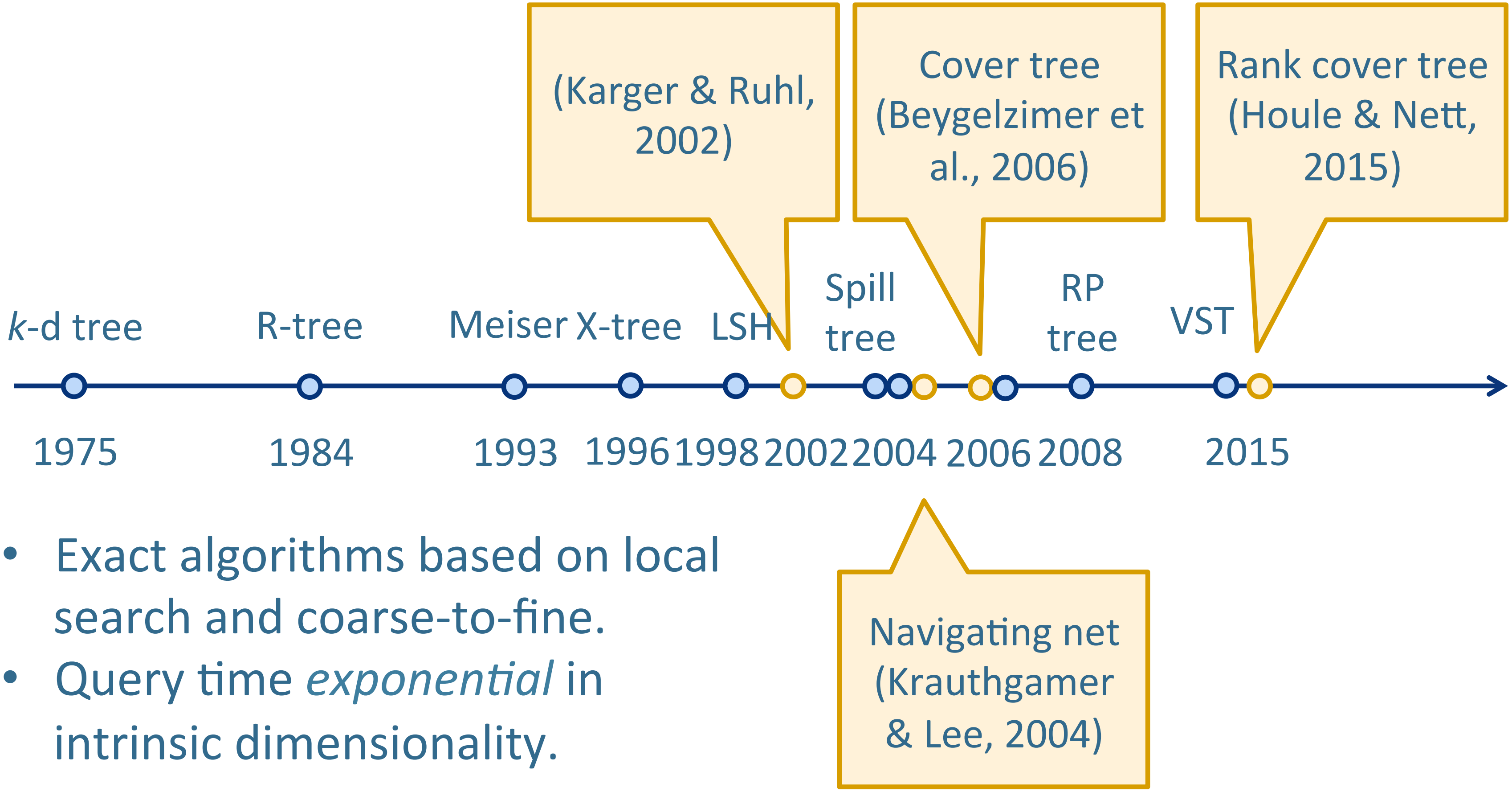
Spill tree (Liu et al., 2004)

Virtual spill tree (Dasgupta & Sinha, 2015)

RP tree (Dasgupta & Freund, 2008)

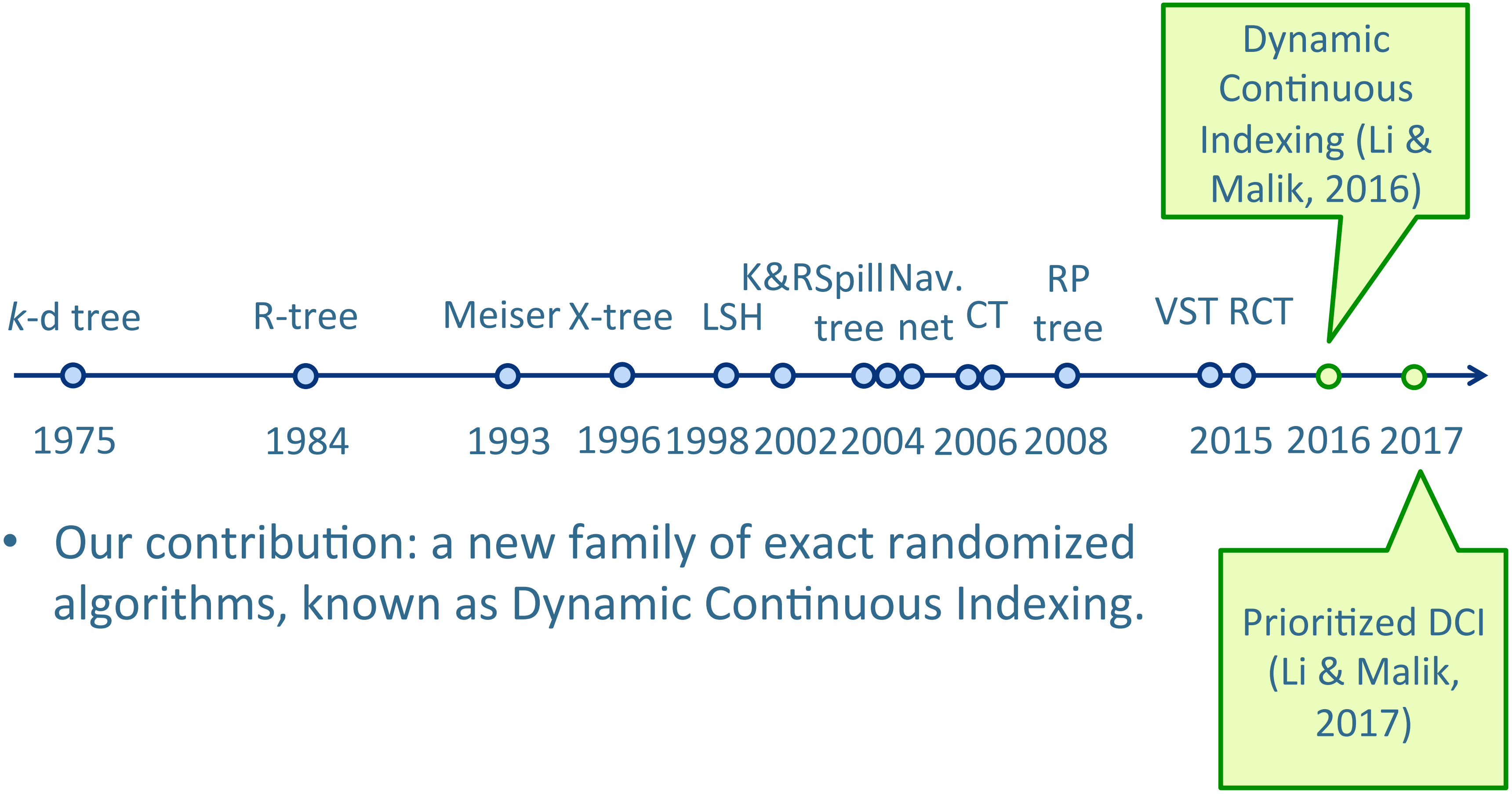
- Exact randomized algorithms based on space partitioning.
- Query time *exponential* in intrinsic dimensionality.

The Curse of Dimensionality


(Karger & Ruhl, 2002)
Cover tree (Beygelzimer et al., 2006)
Rank cover tree (Houle & Nett, 2015)
k-d tree
R-tree
Meiser
X-tree
LSH
Spill tree
RP tree
VST
1975
1984
1993
1996
1998
2002
2004
2006
2008
2015
Navigating net (Krauthgamer & Lee, 2004)

- Exact algorithms based on local search and coarse-to-fine.
- Query time *exponential* in intrinsic dimensionality.

The Curse of Dimensionality

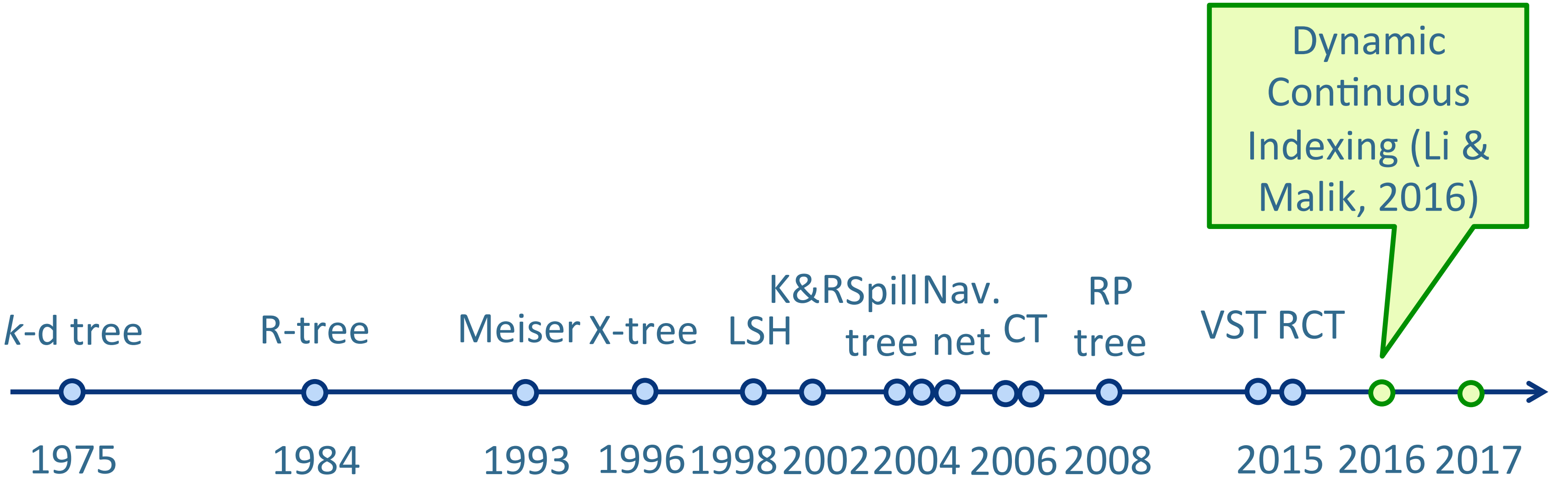


Dynamic Continuous Indexing (Li & Malik, 2016)

Prioritized DCI (Li & Malik, 2017)

- Our contribution: a new family of exact randomized algorithms, known as Dynamic Continuous Indexing.

The Curse of Dimensionality



k-d tree

R-tree

Meiser

X-tree

LSH

K&R

Spill tree

Nav. net

CT

RP tree

VST

RCT

1975

1984

1993

1996

1998

2002

2004

2006

2008

2015

2016

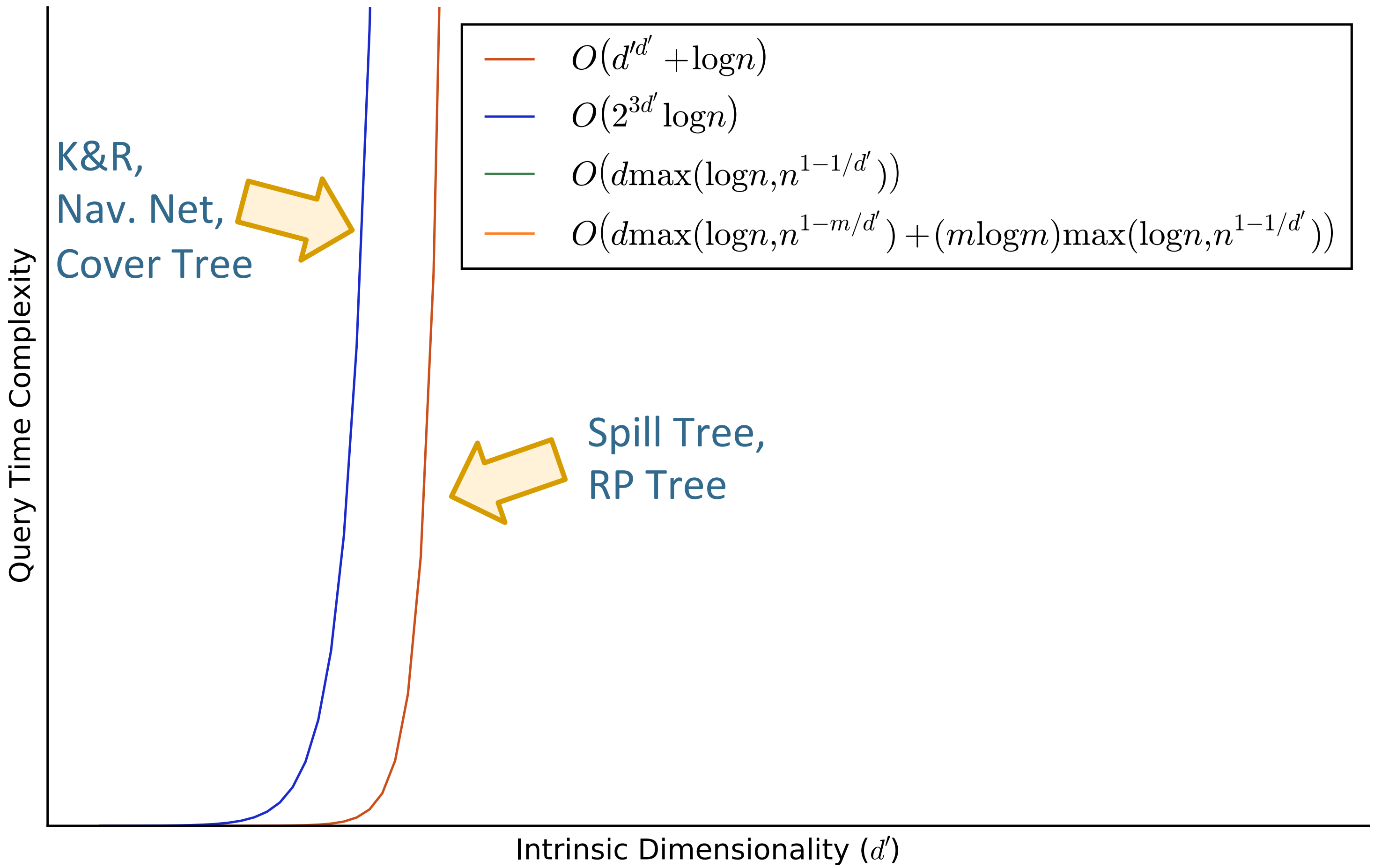
2017

Dynamic Continuous Indexing (Li & Malik, 2016)

- Query time *linear* in ambient dimensionality and *sublinear* in intrinsic dimensionality.
- Space complexity independent of ambient or intrinsic dimensionality.

Prioritized DCI (Li & Malik, 2017)

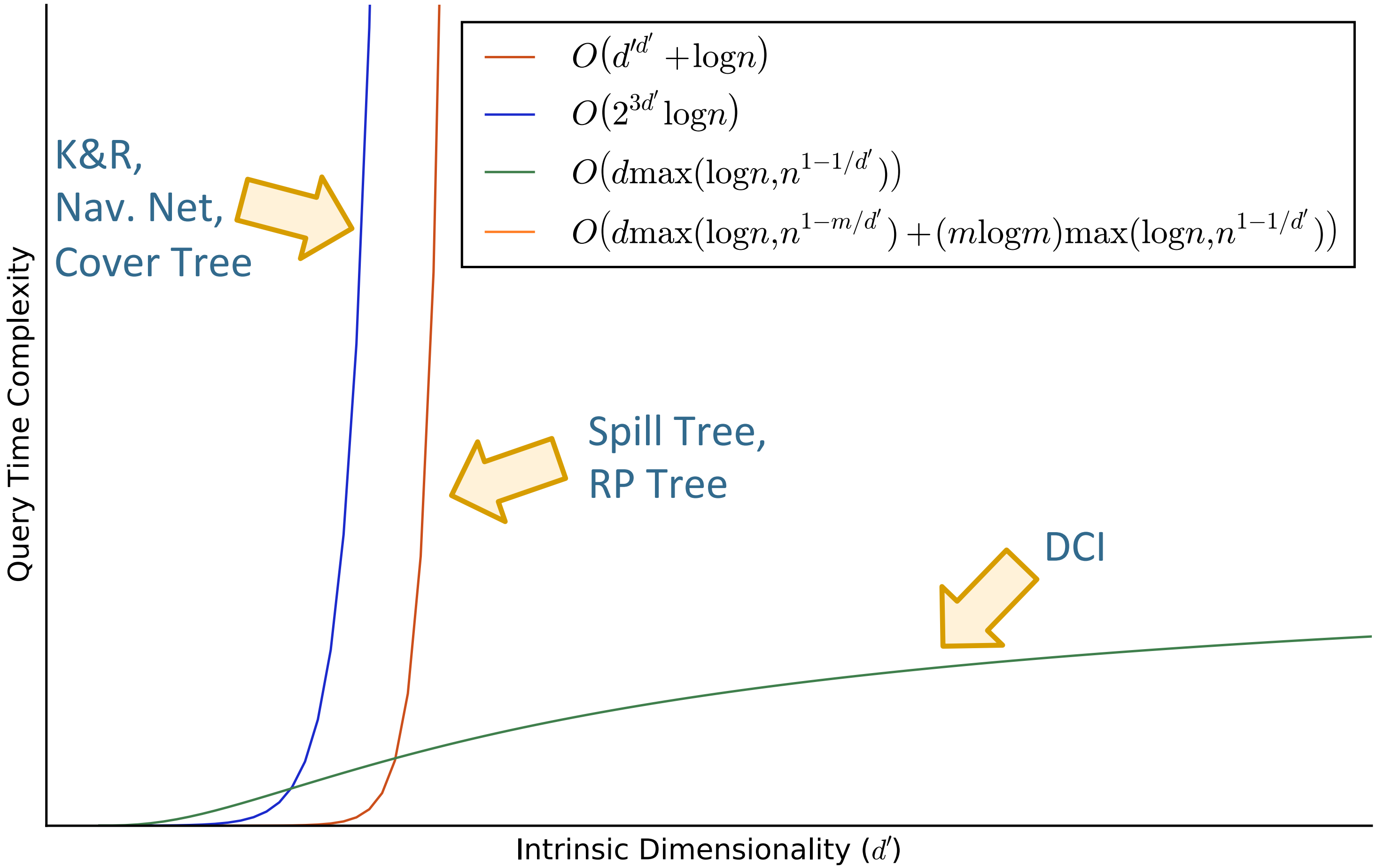
The Curse of Dimensionality



K&R,
Nav. Net,
Cover Tree

Spill Tree,
RP Tree

The Curse of Dimensionality

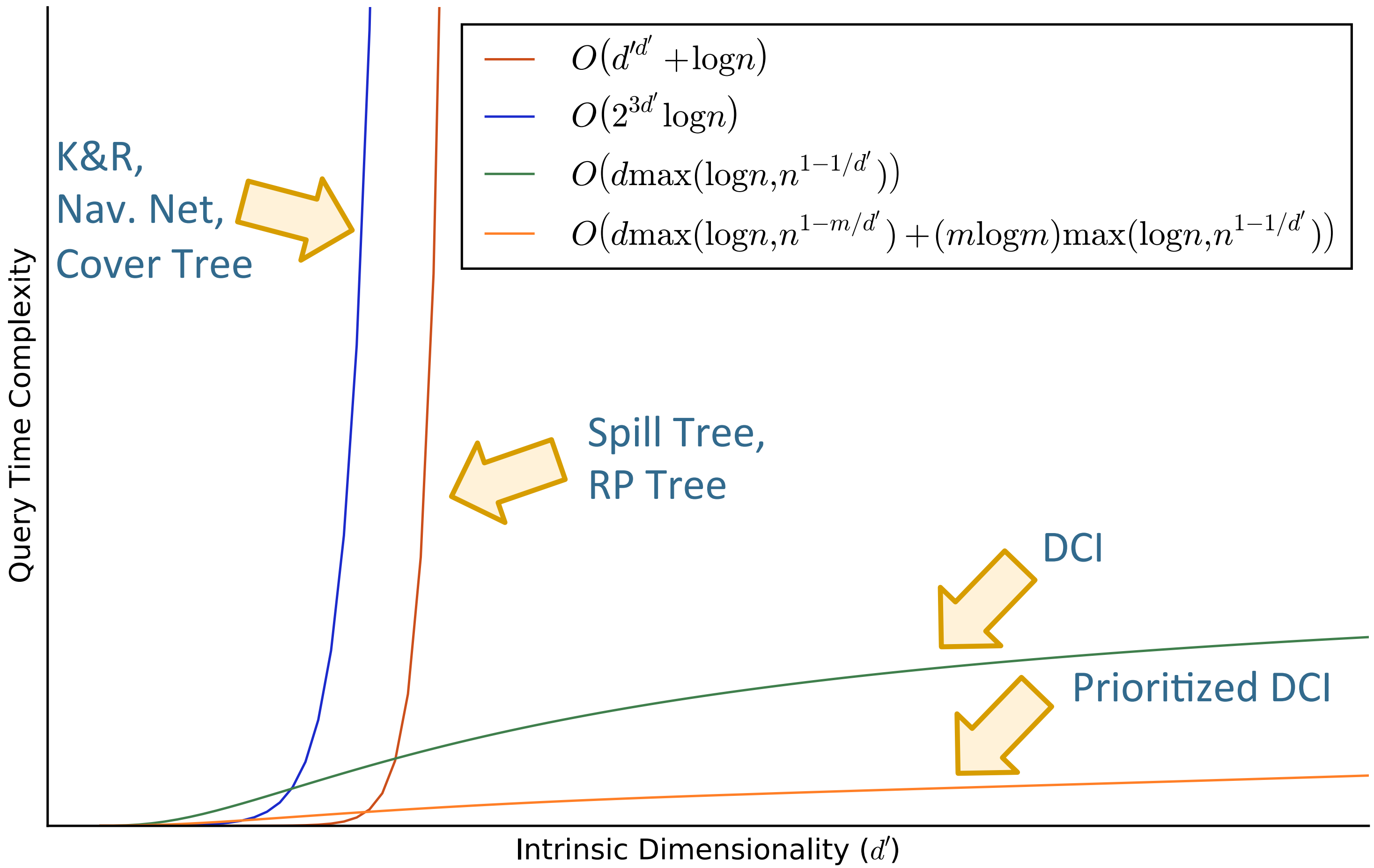


K&R,
Nav. Net,
Cover Tree

Spill Tree,
RP Tree

DCI

The Curse of Dimensionality



K&R,
Nav. Net,
Cover Tree

Spill Tree,
RP Tree

DCI

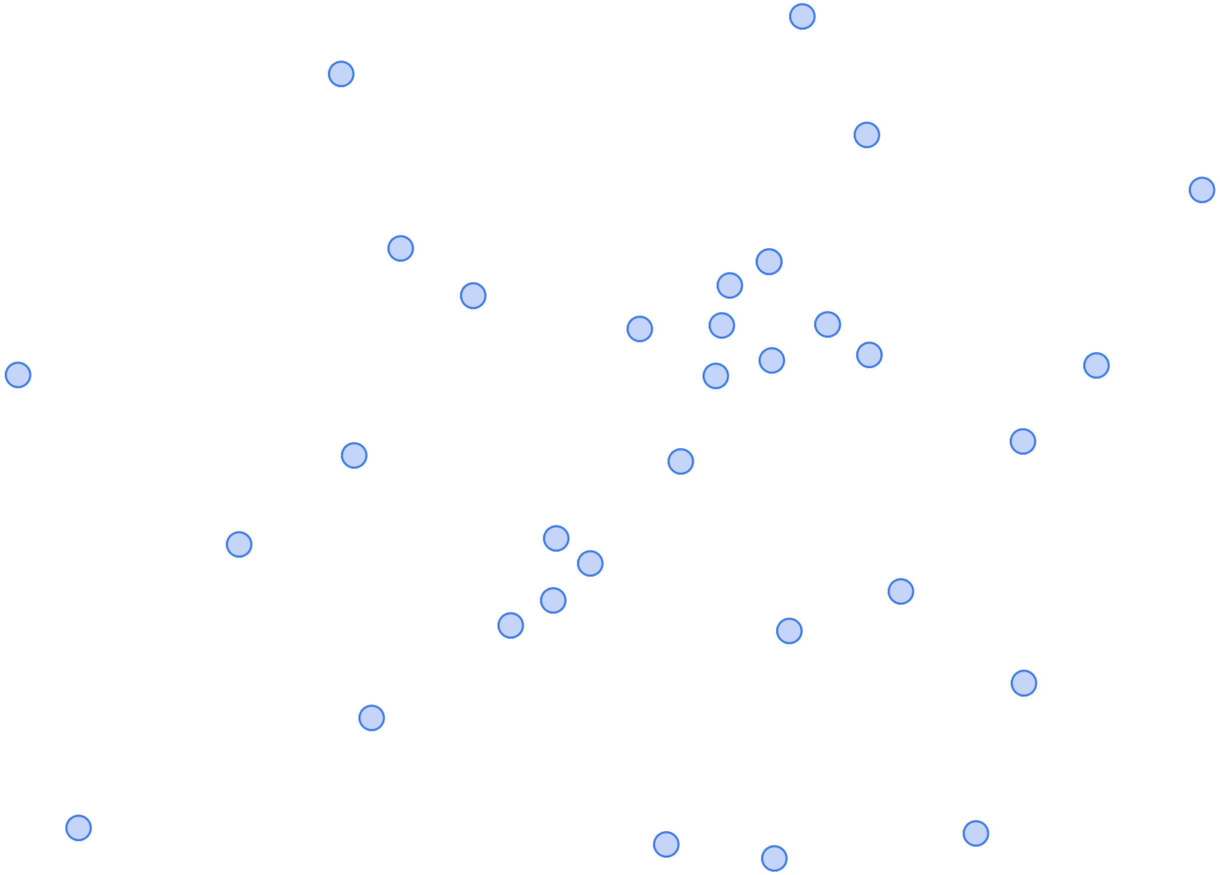
Prioritized DCI

Our Approach

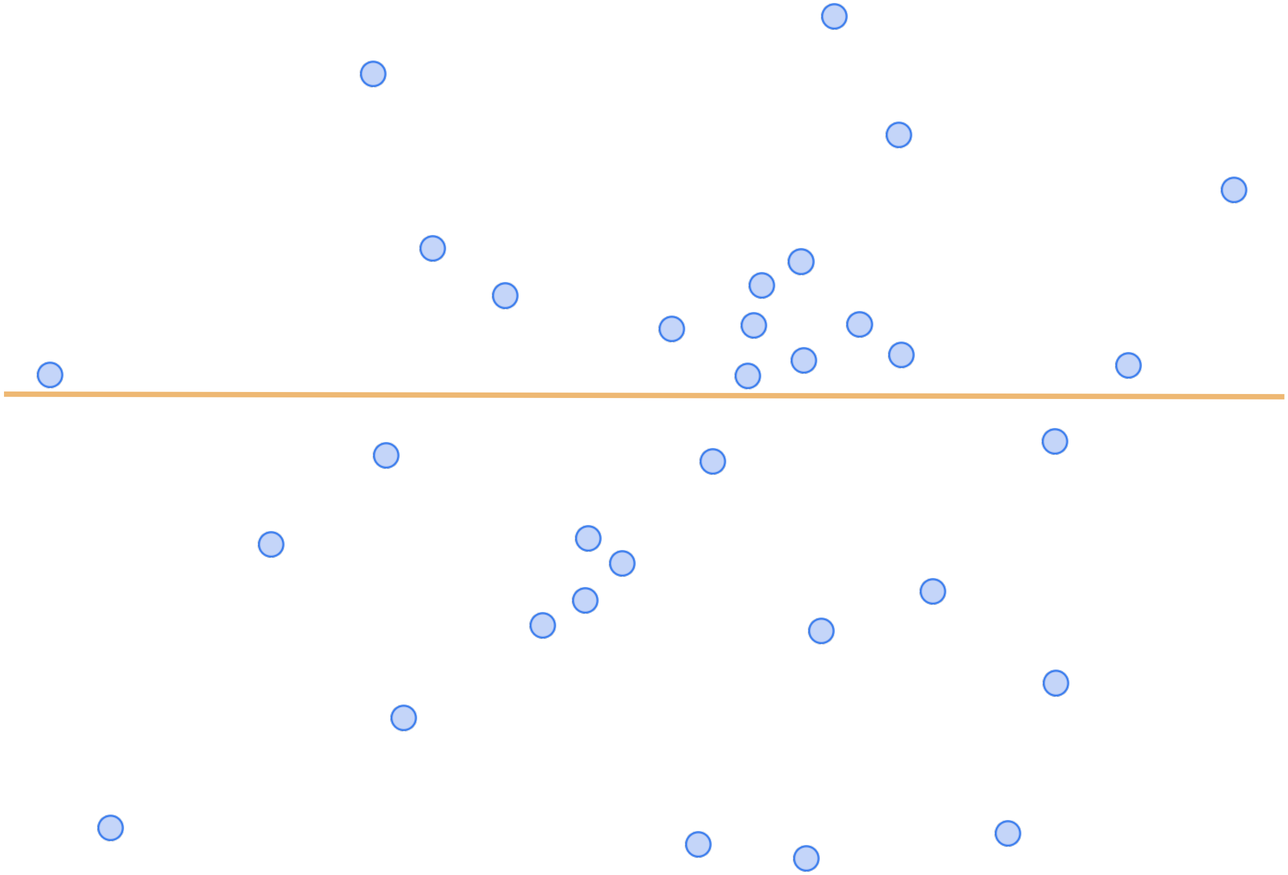
- Key difference from prior methods: Dynamic Continuous Indexing (DCI) avoids *space partitioning*.
- Space partitioning is a divide-and-conquer strategy that underlies most existing methods, including *k*-d trees and locality-sensitive hashing (LSH).
  - It works by partitioning the space into discrete cells and keeping track of points contained in each.
- We conjecture that the curse of dimensionality stems from the inherent deficiencies of space partitioning.

The Case Against Space Partitioning

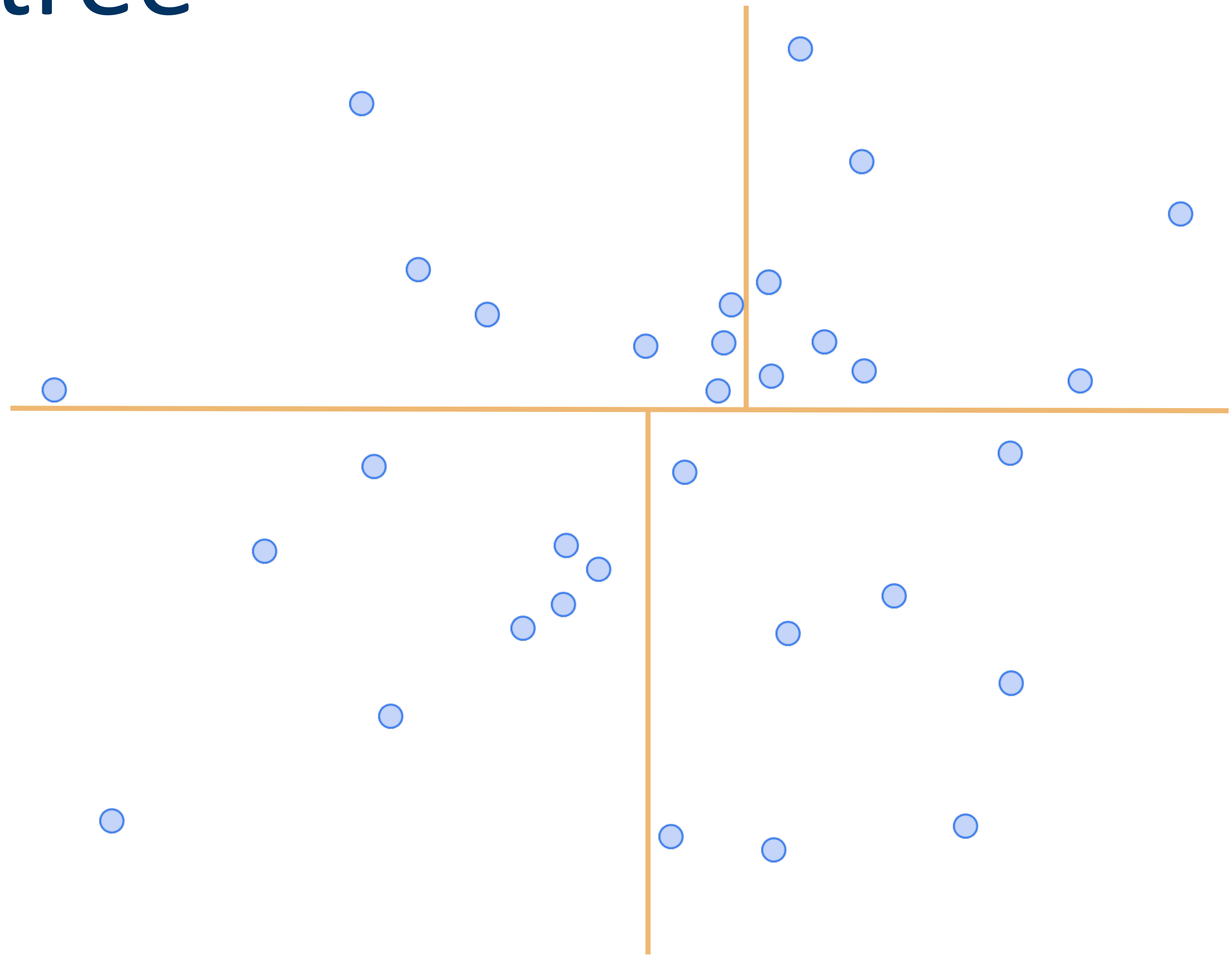
k-d tree



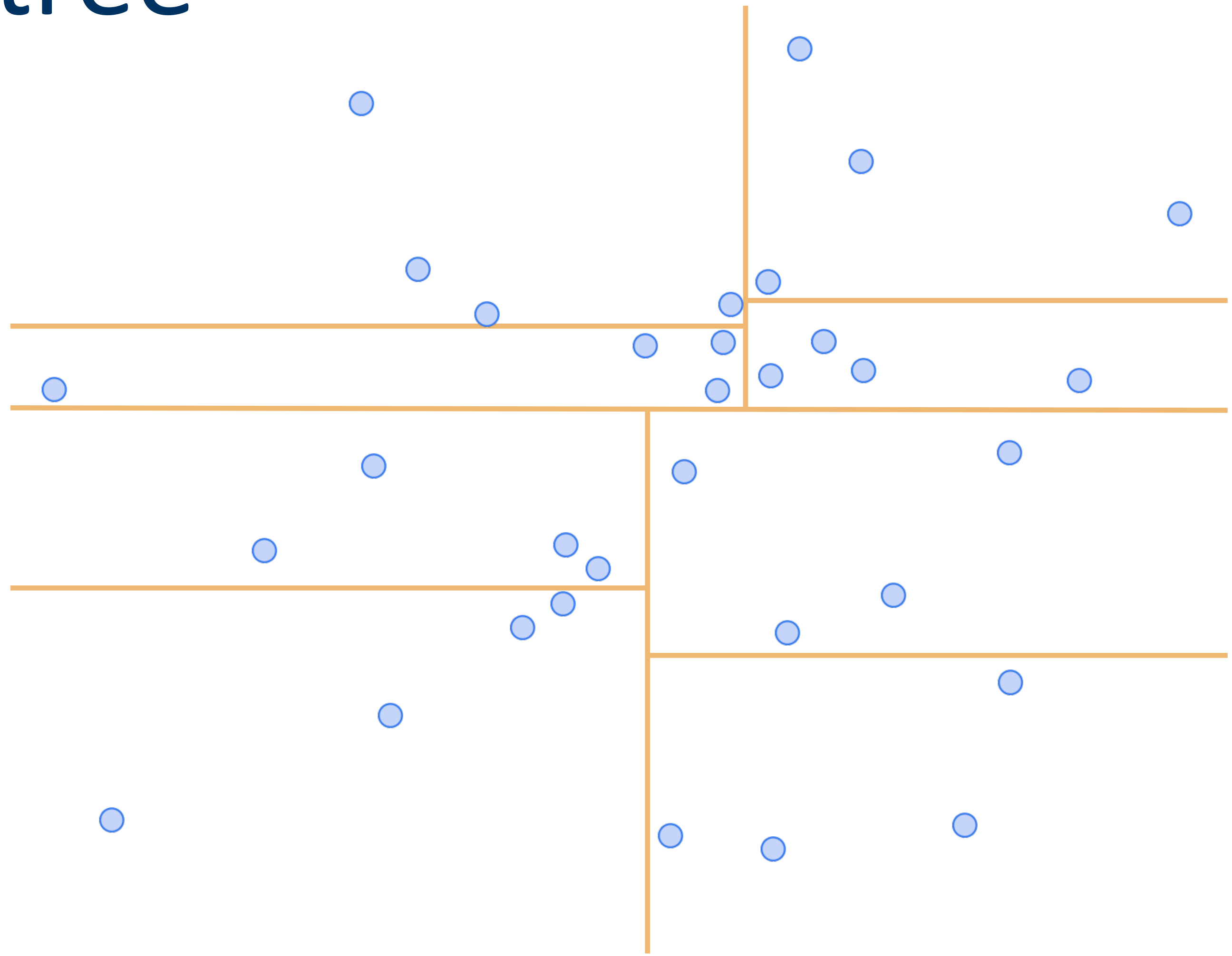
k-d tree



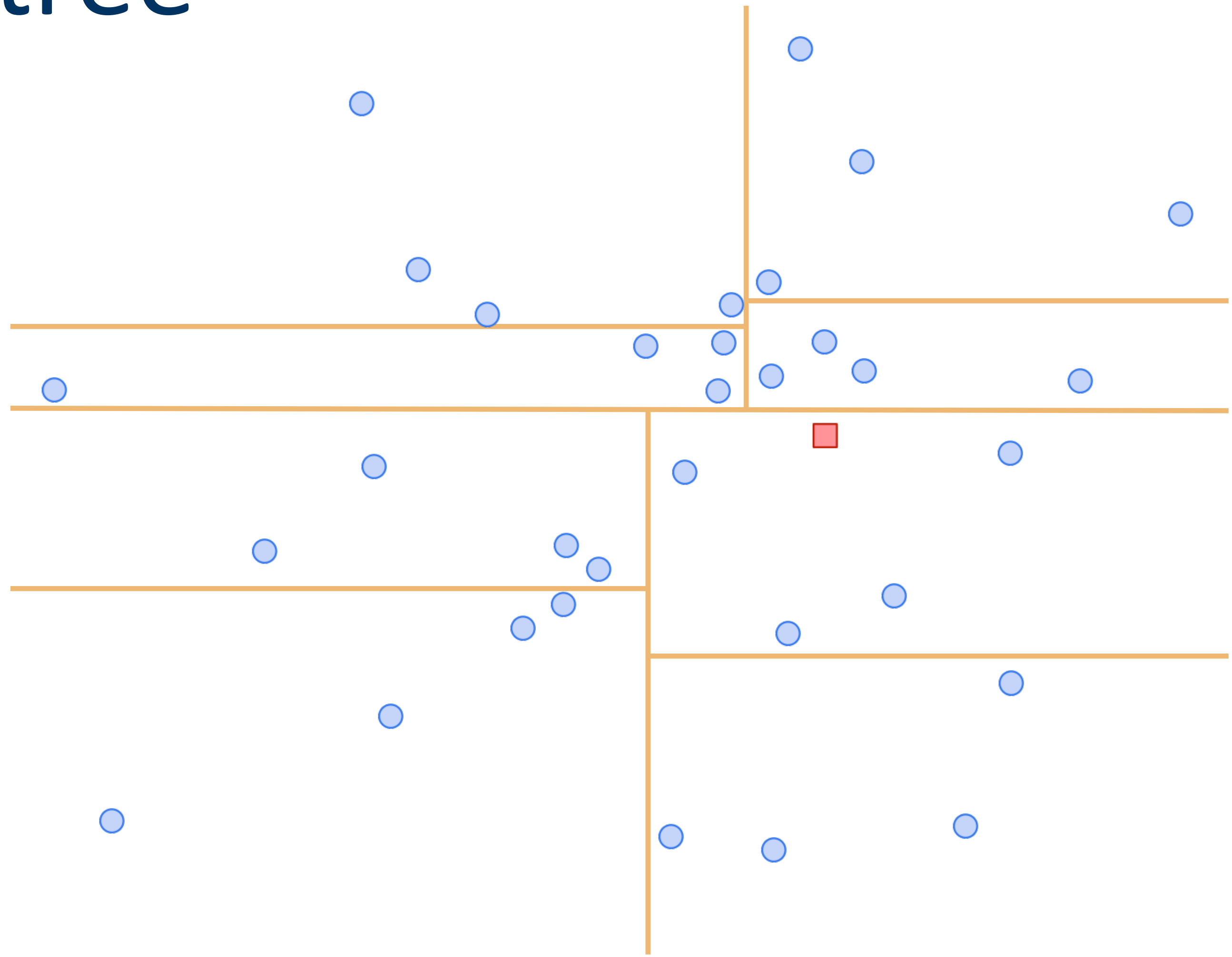
k-d tree



k-d tree

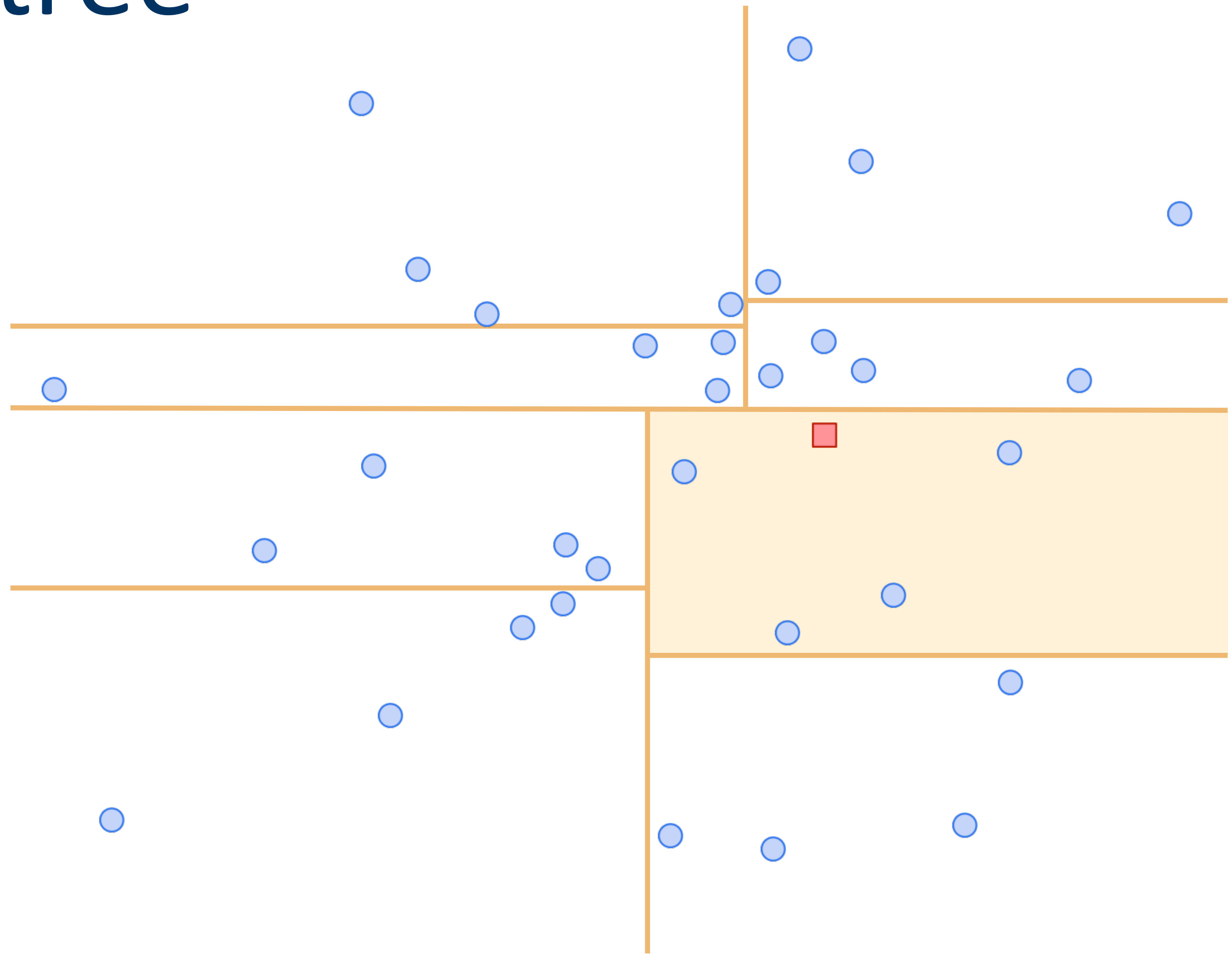


k-d tree

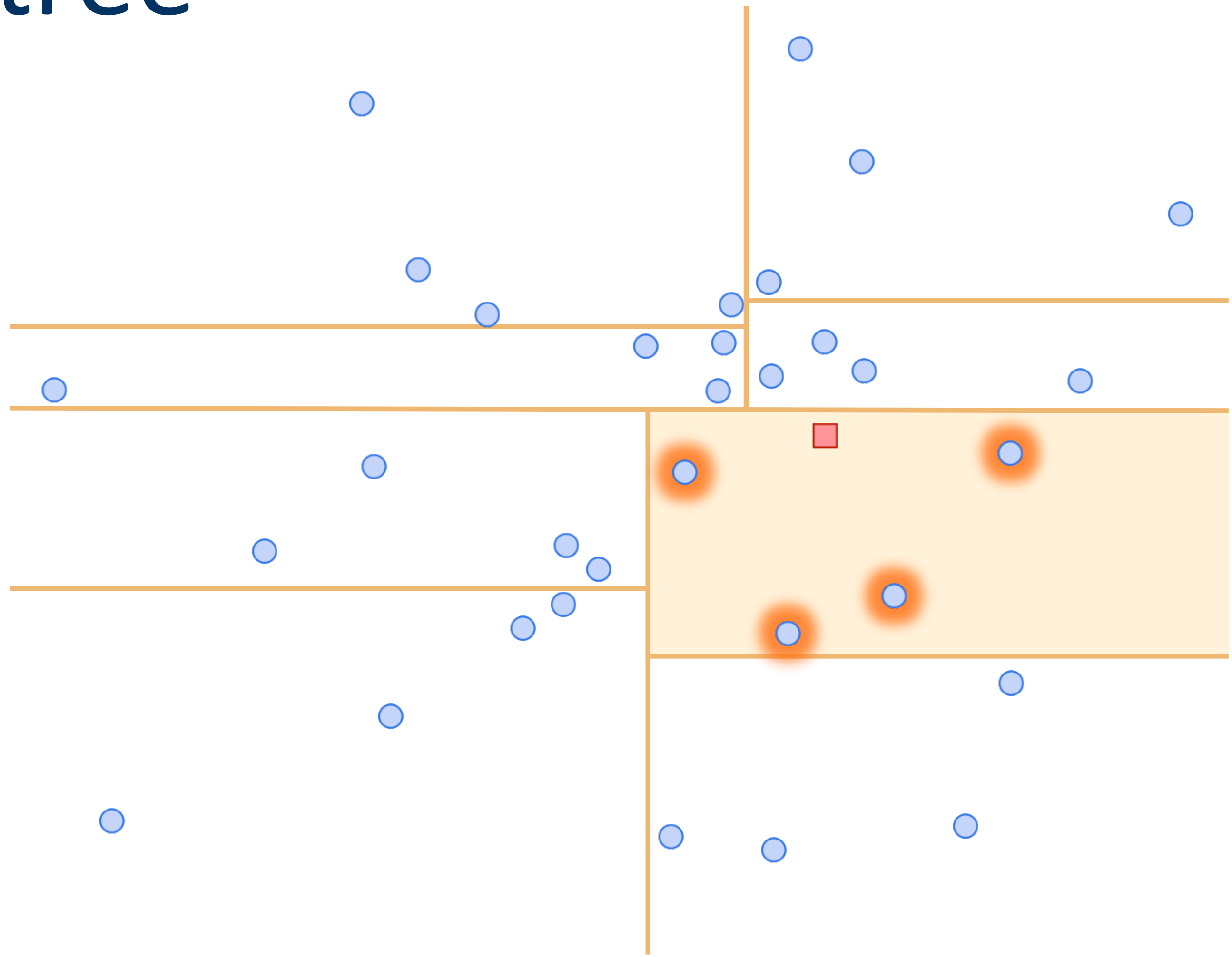


Berkeley
UNIVERSITY OF CALIFORNIA

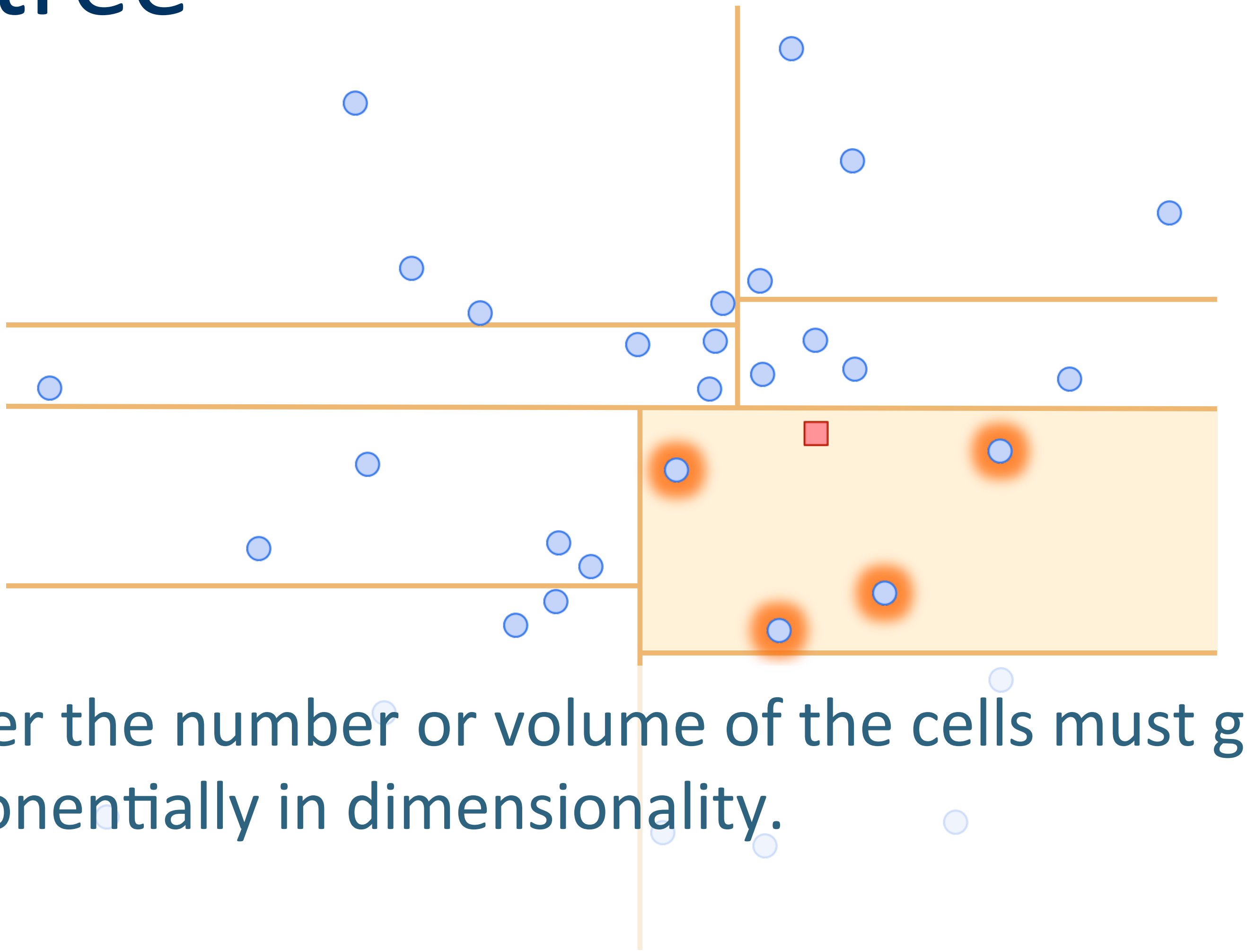
k-d tree



k-d tree

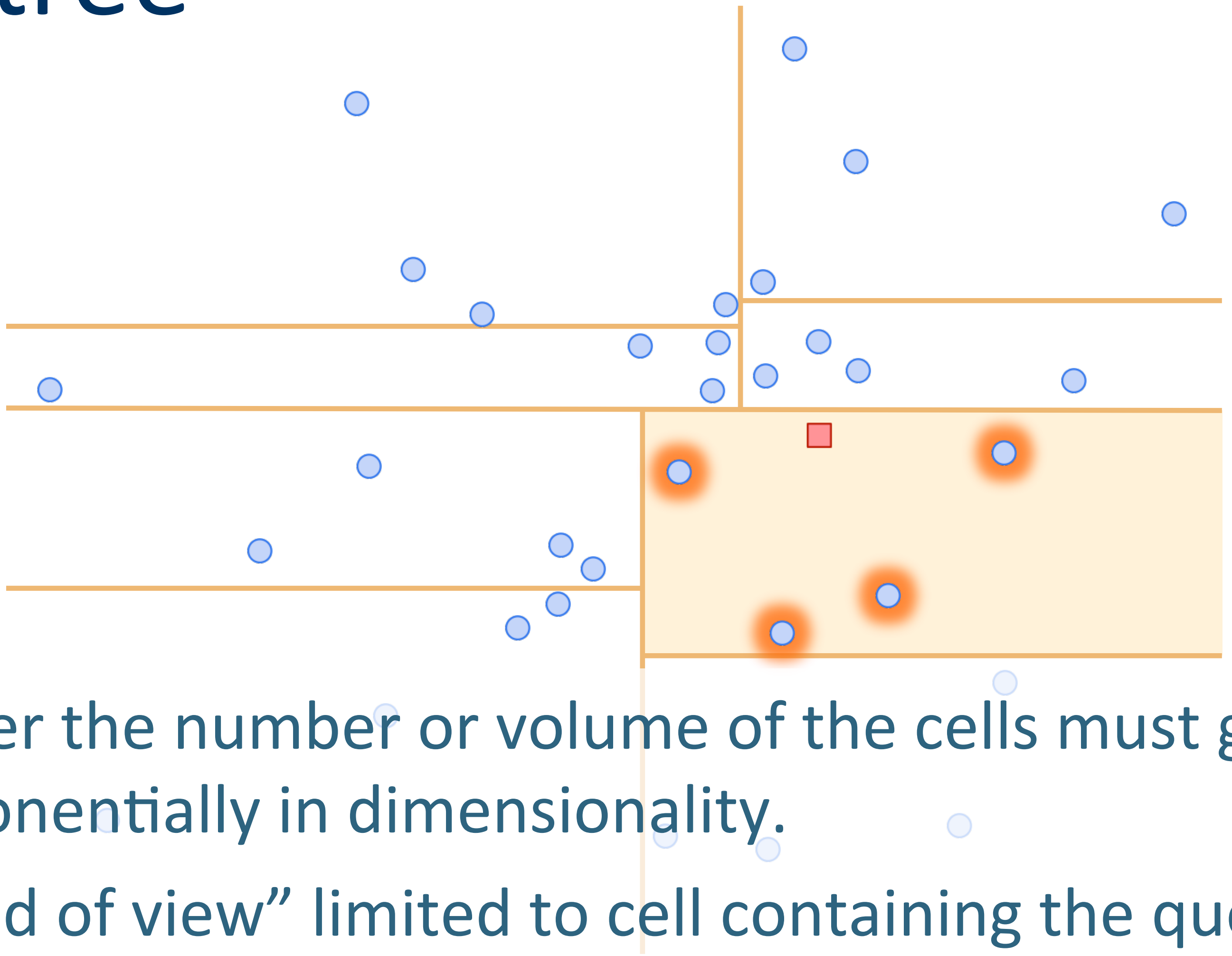


k-d tree



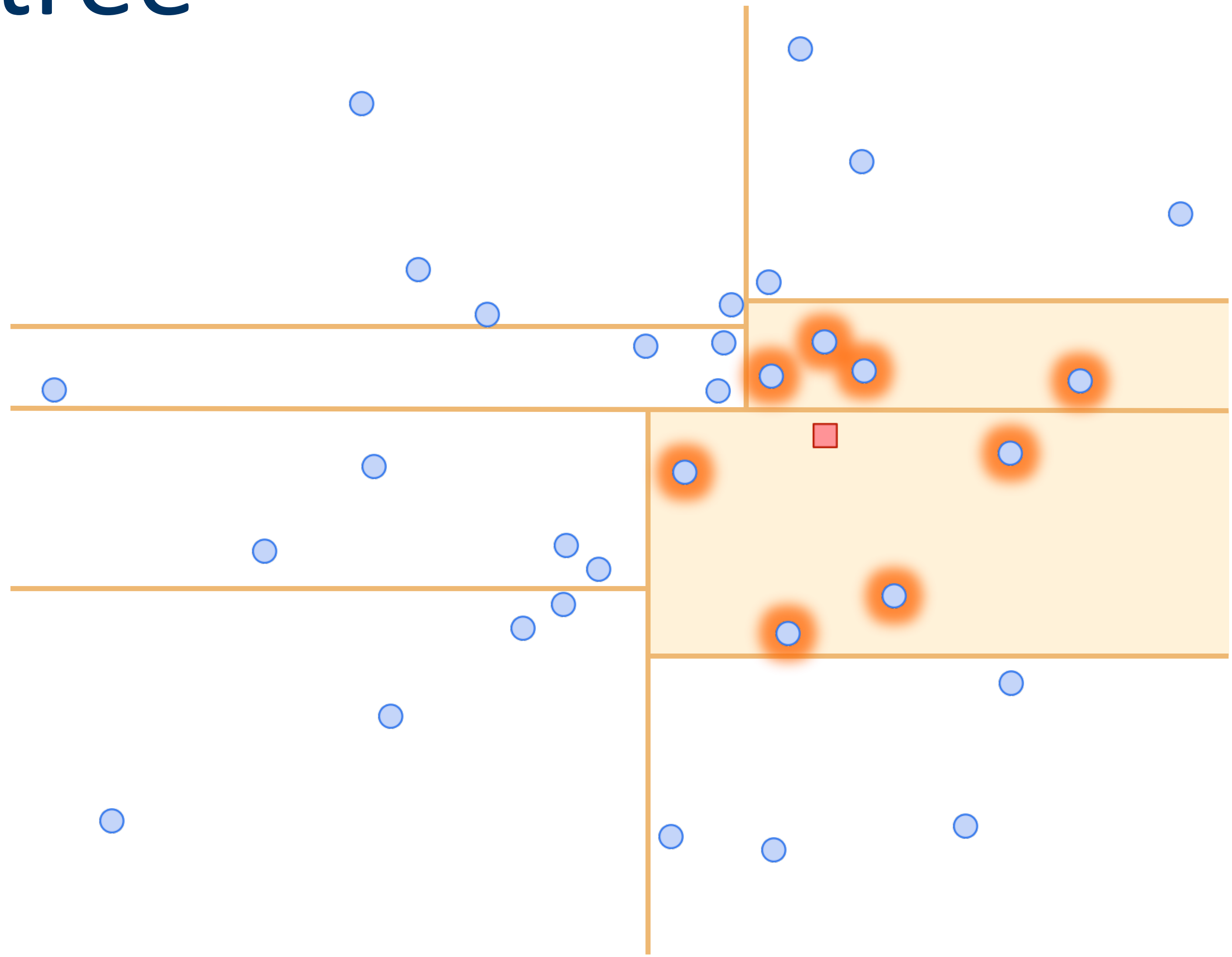
- Either the number or volume of the cells must grow exponentially in dimensionality.

k-d tree

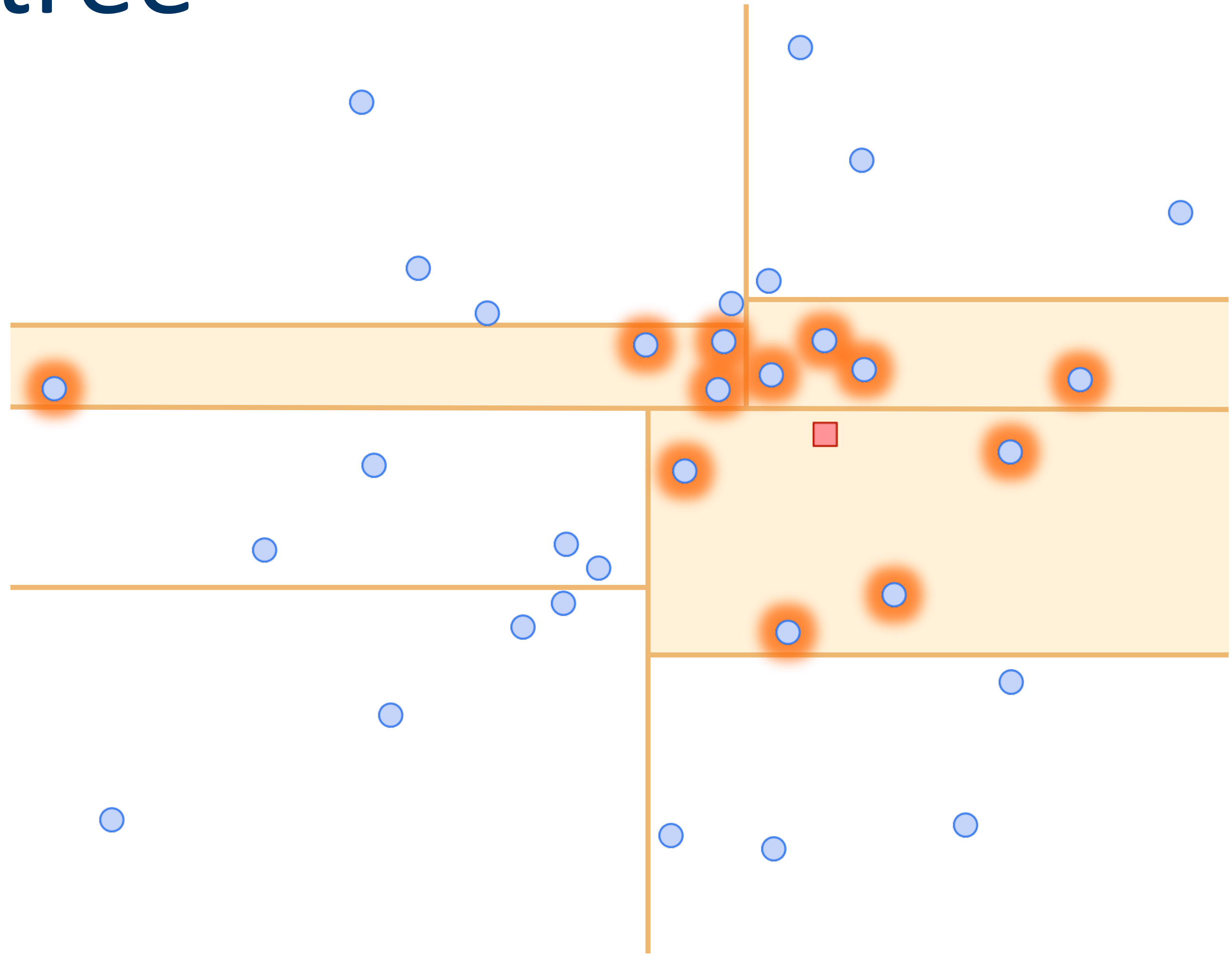


- Either the number or volume of the cells must grow exponentially in dimensionality.
- “Field of view” limited to cell containing the query.

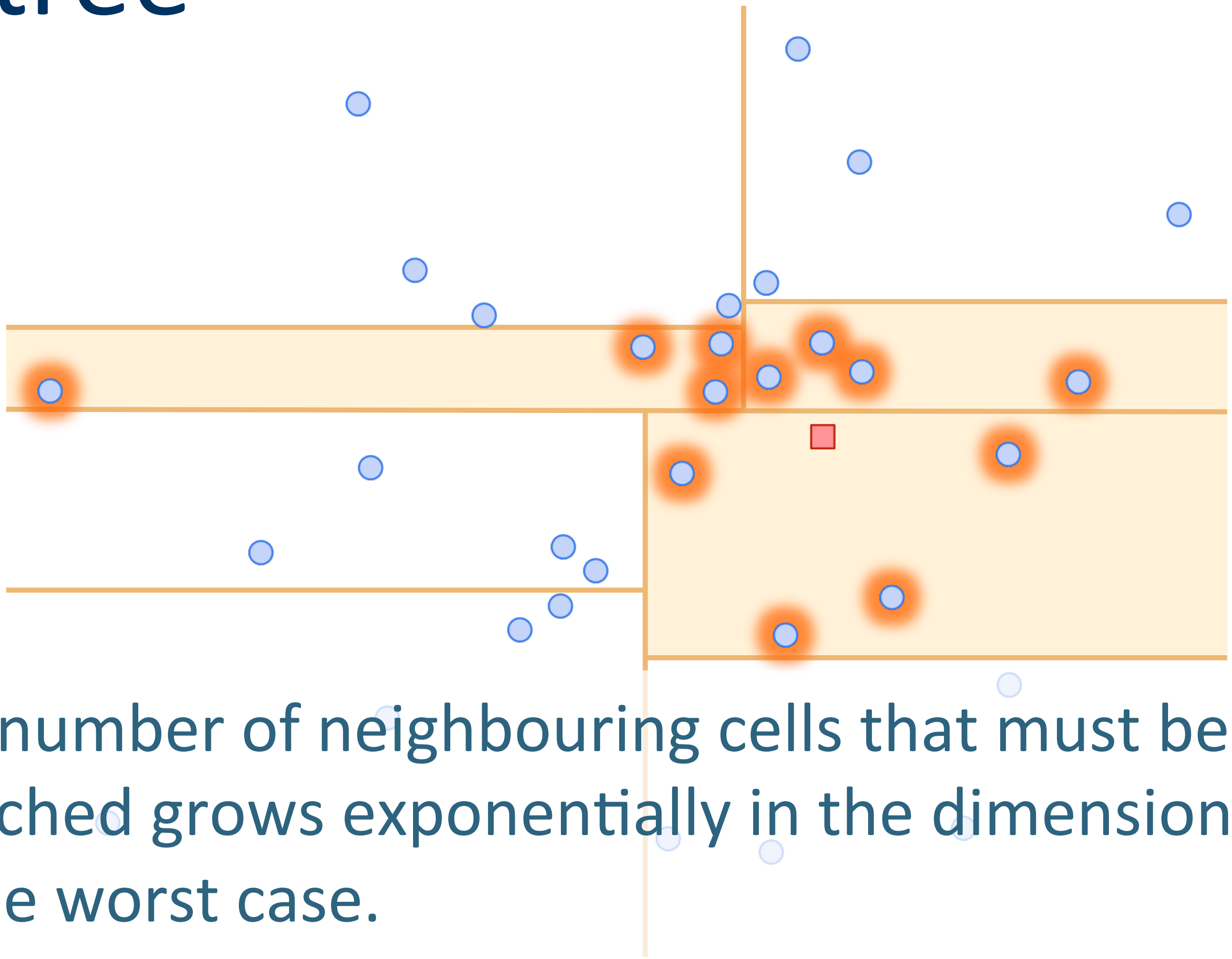
k-d tree



k-d tree

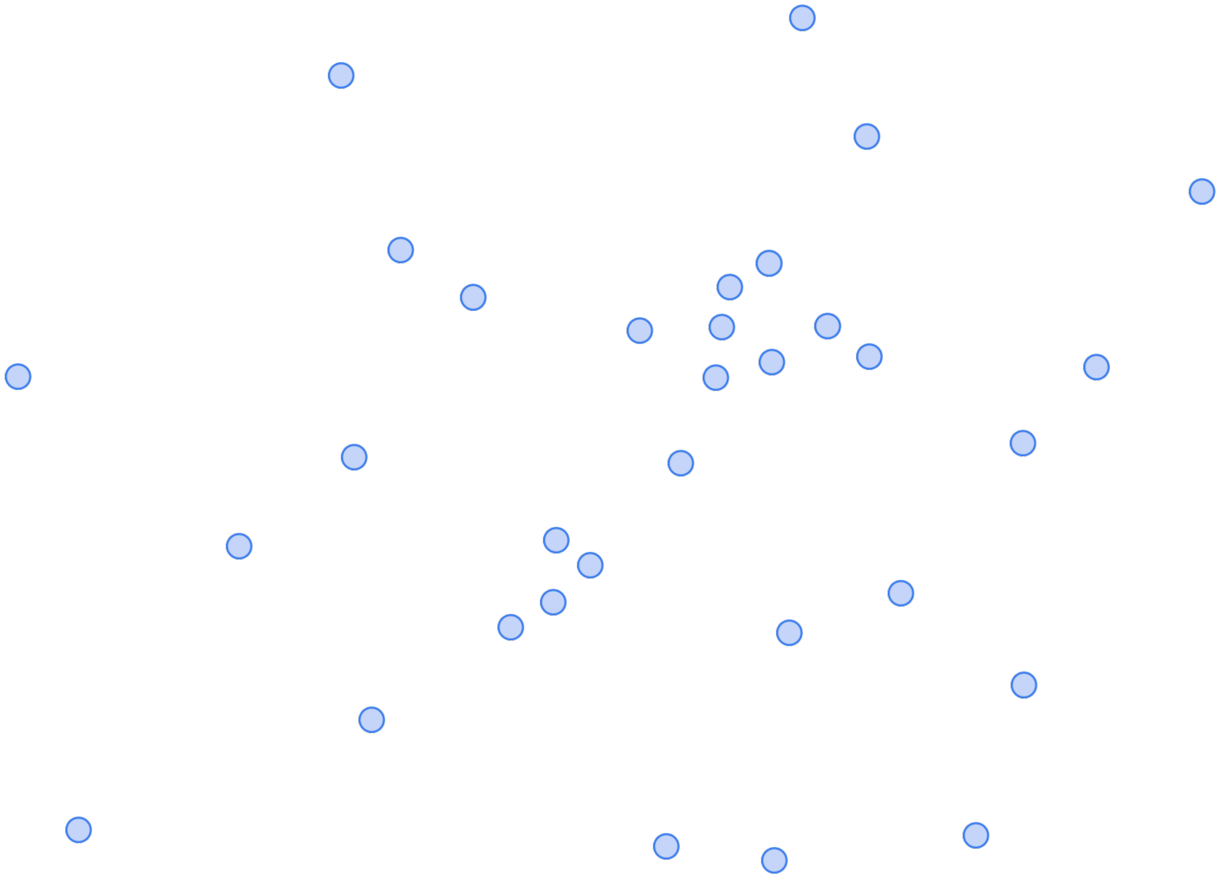


k-d tree

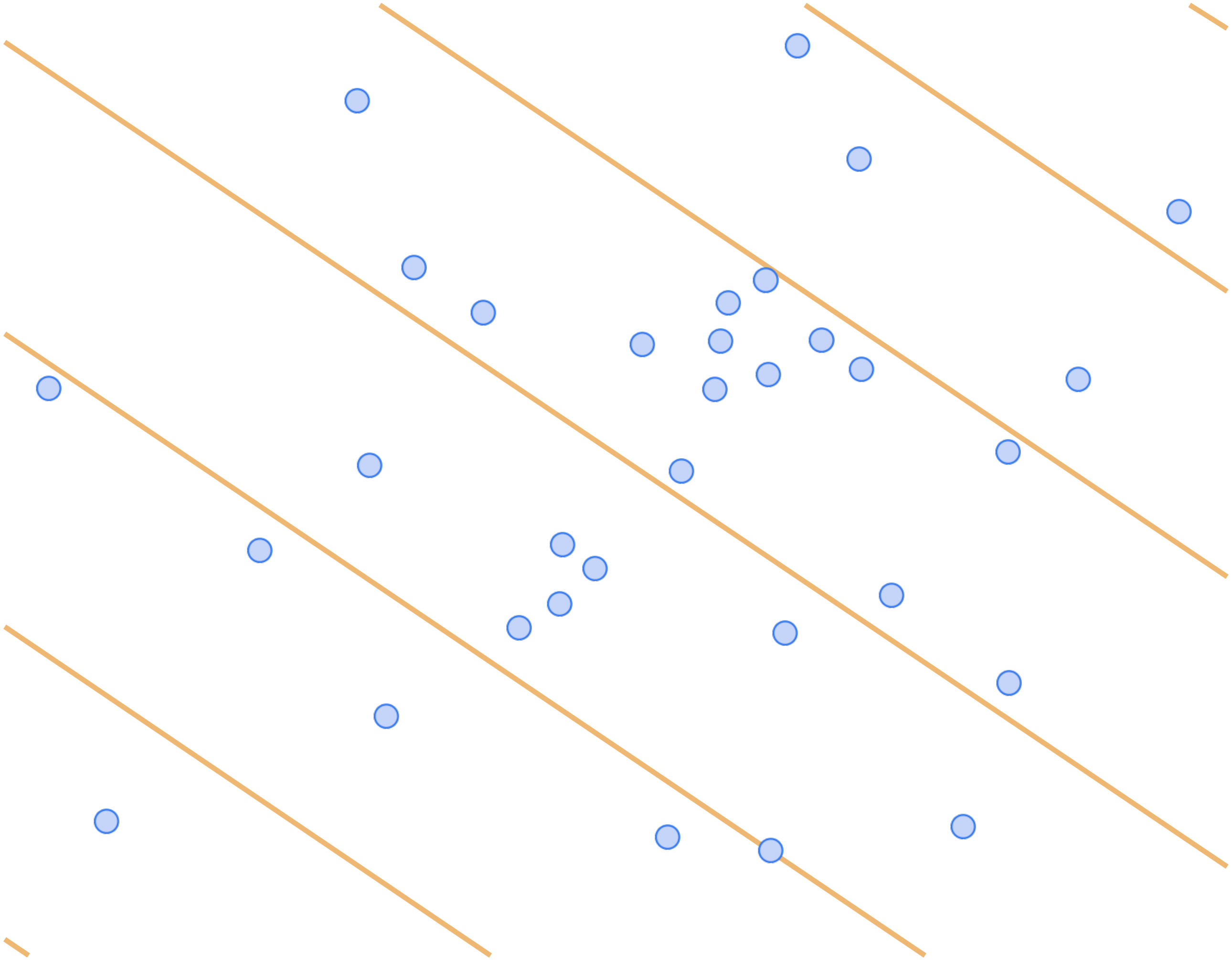


- The number of neighbouring cells that must be searched grows exponentially in the dimensionality in the worst case.

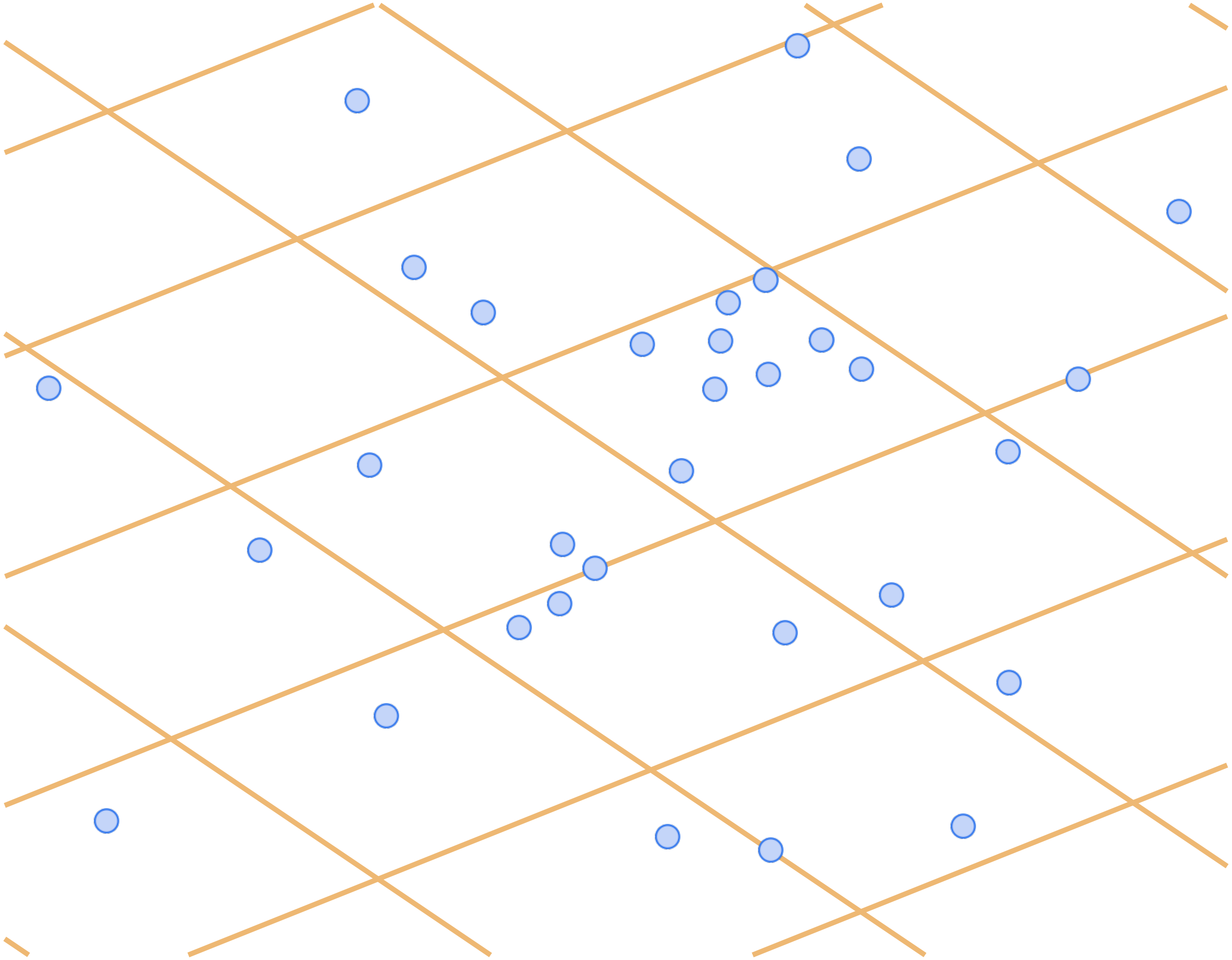
LSH



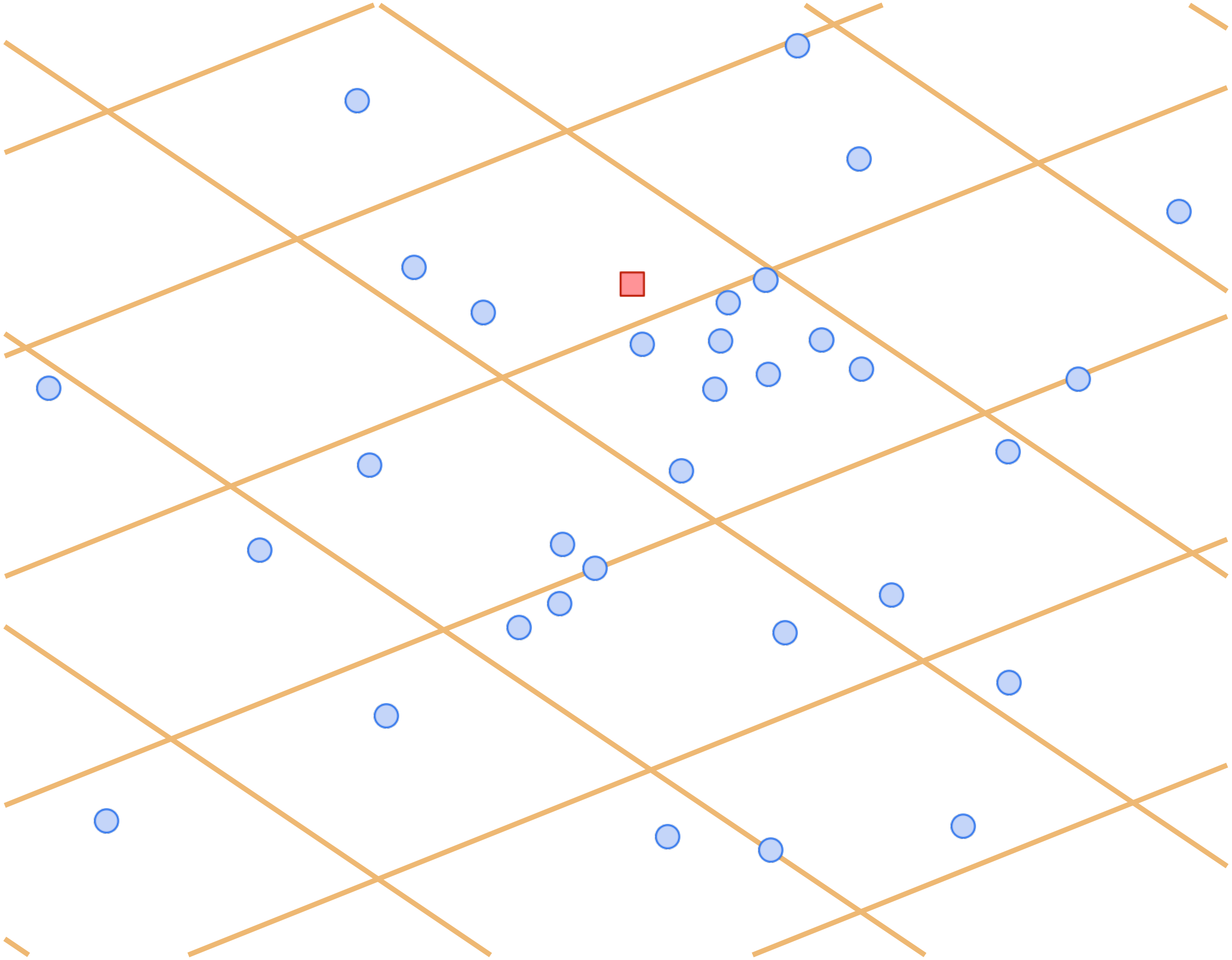
LSH



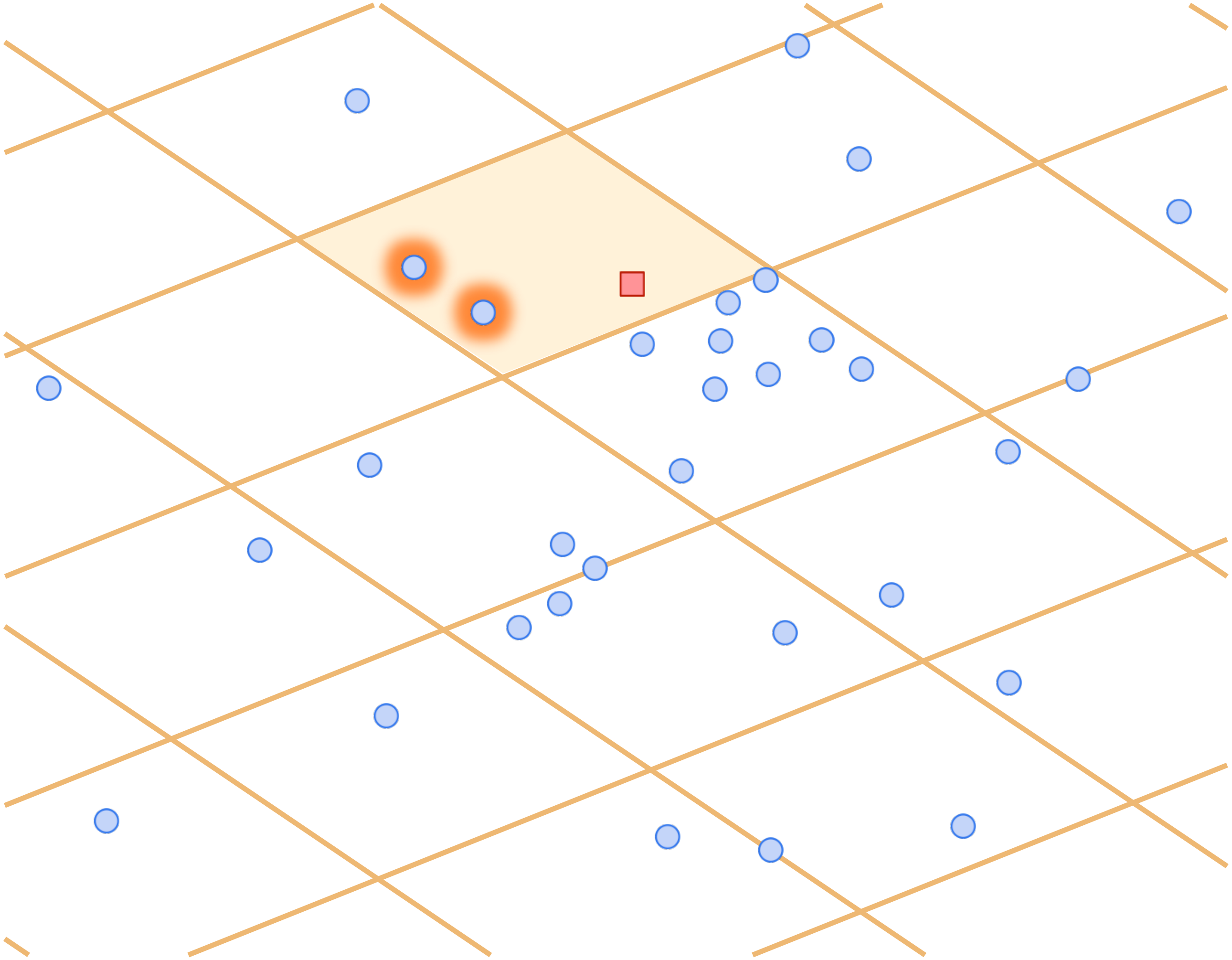
LSH



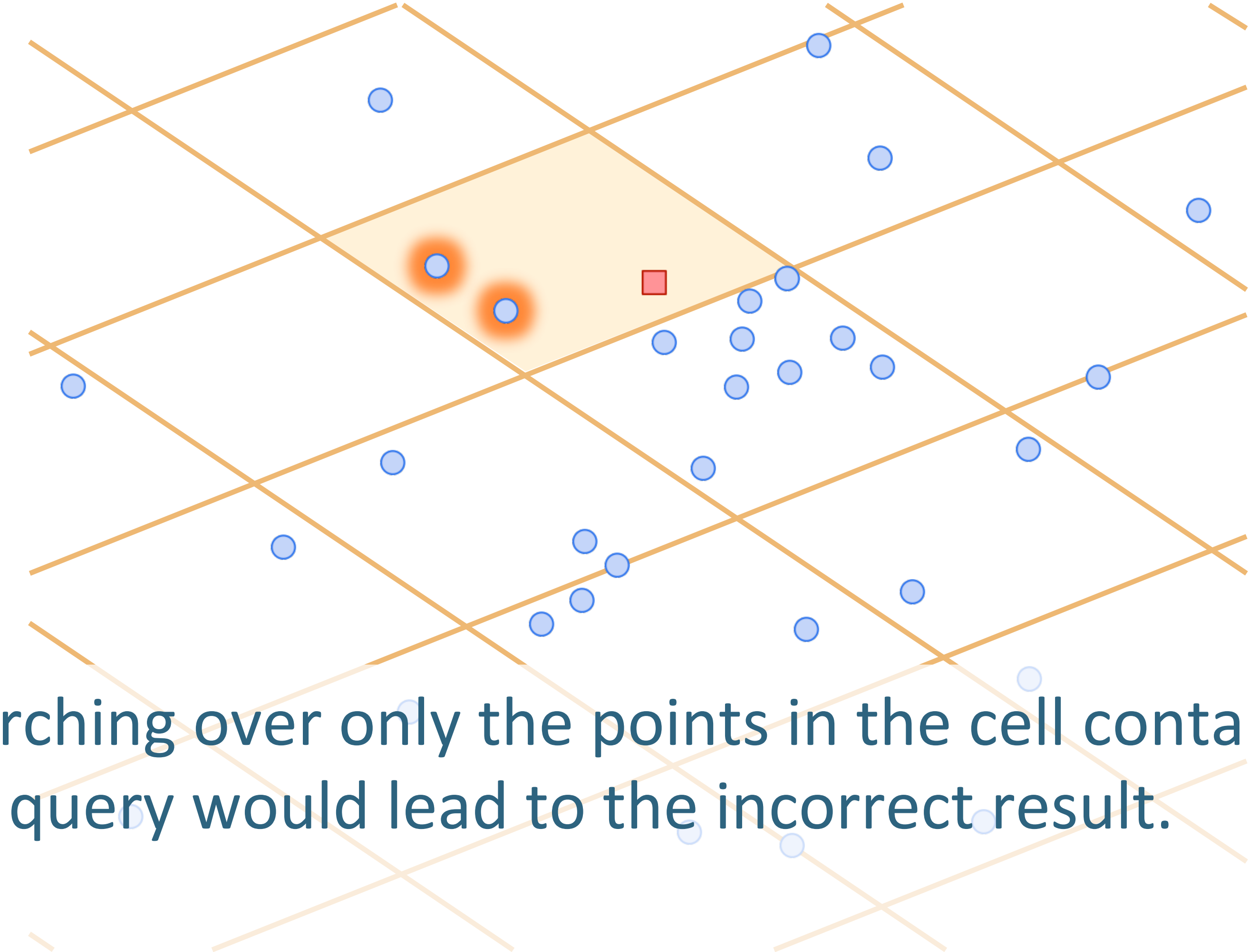
LSH



LSH

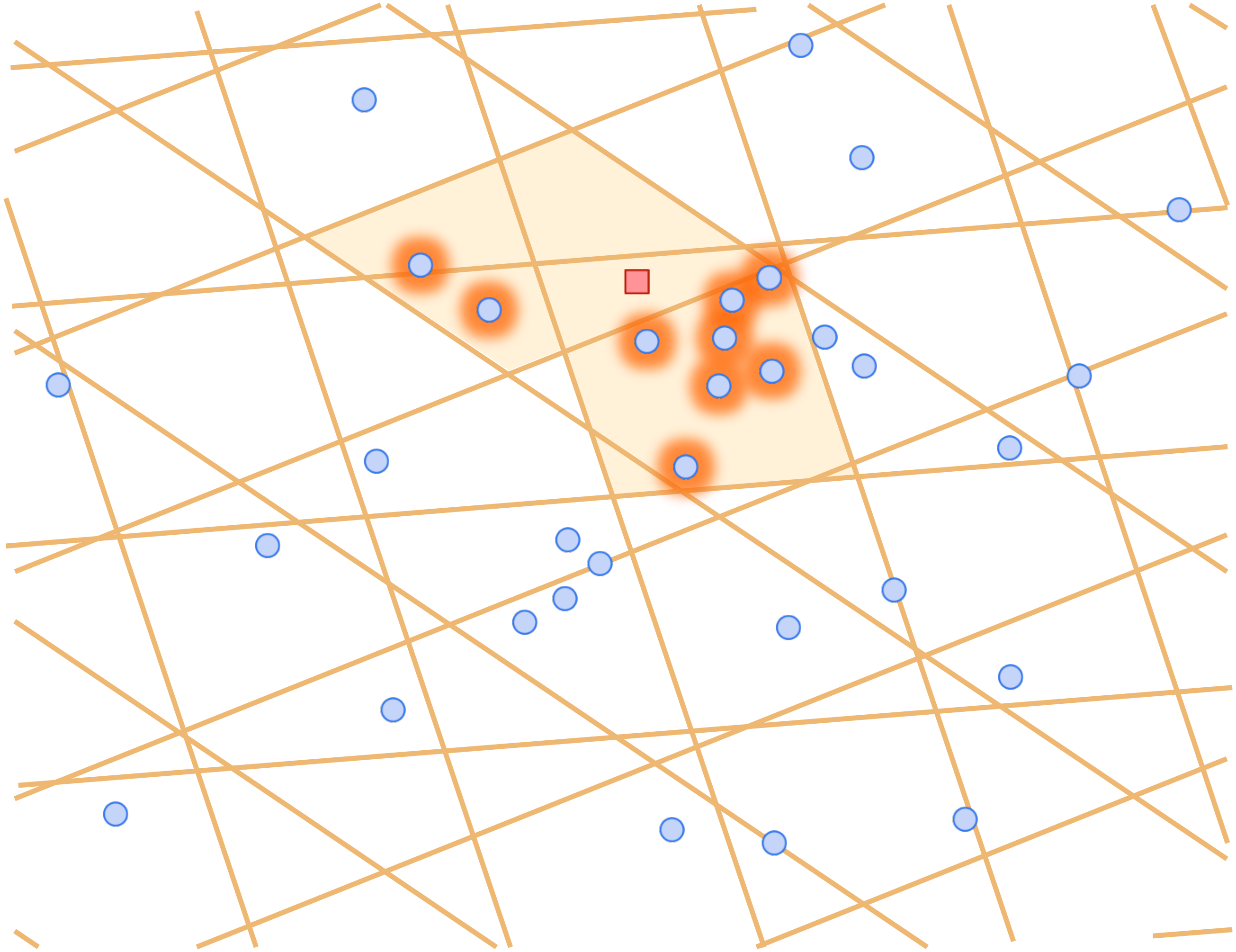


LSH

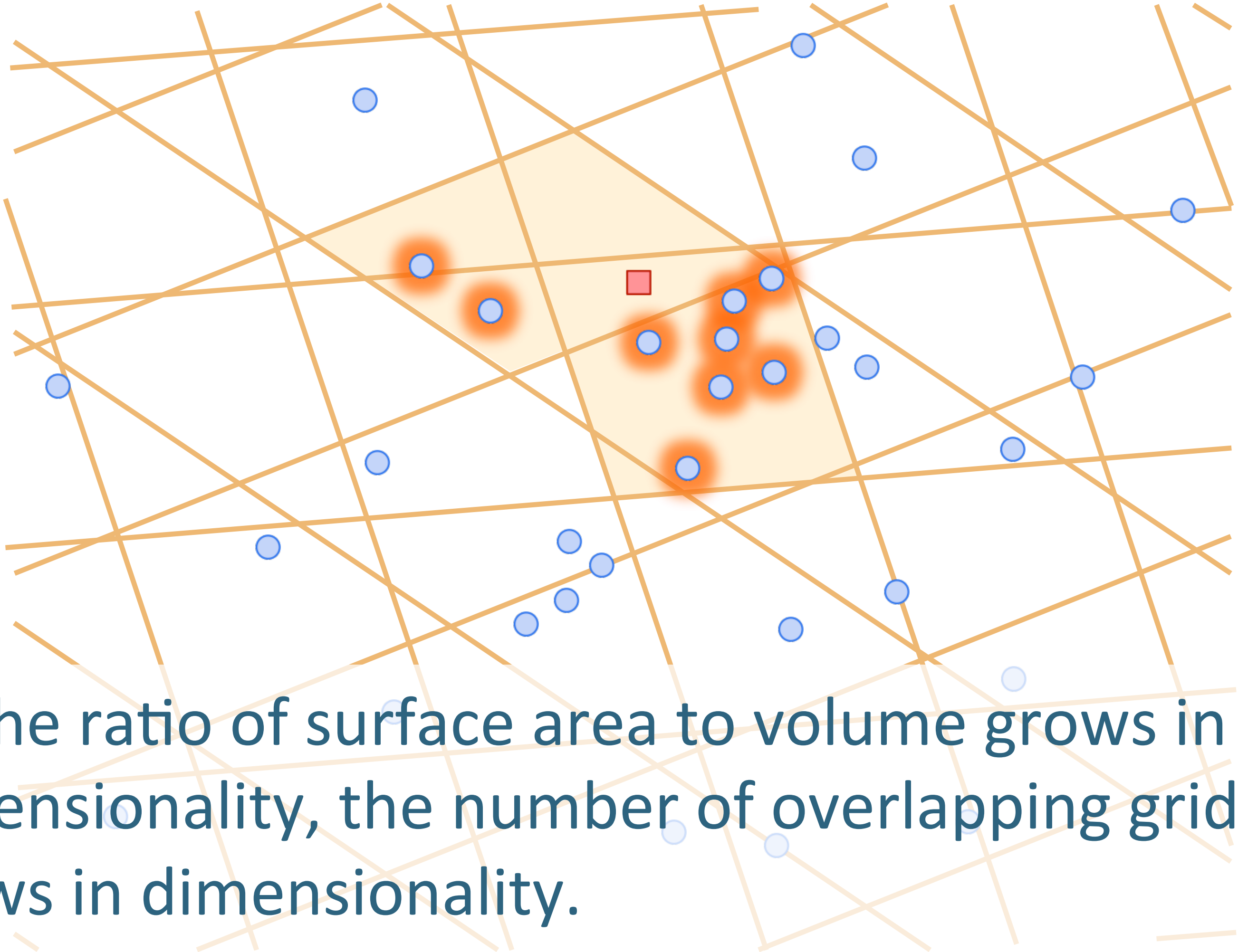


- Searching over only the points in the cell containing the query would lead to the incorrect result.

LSH

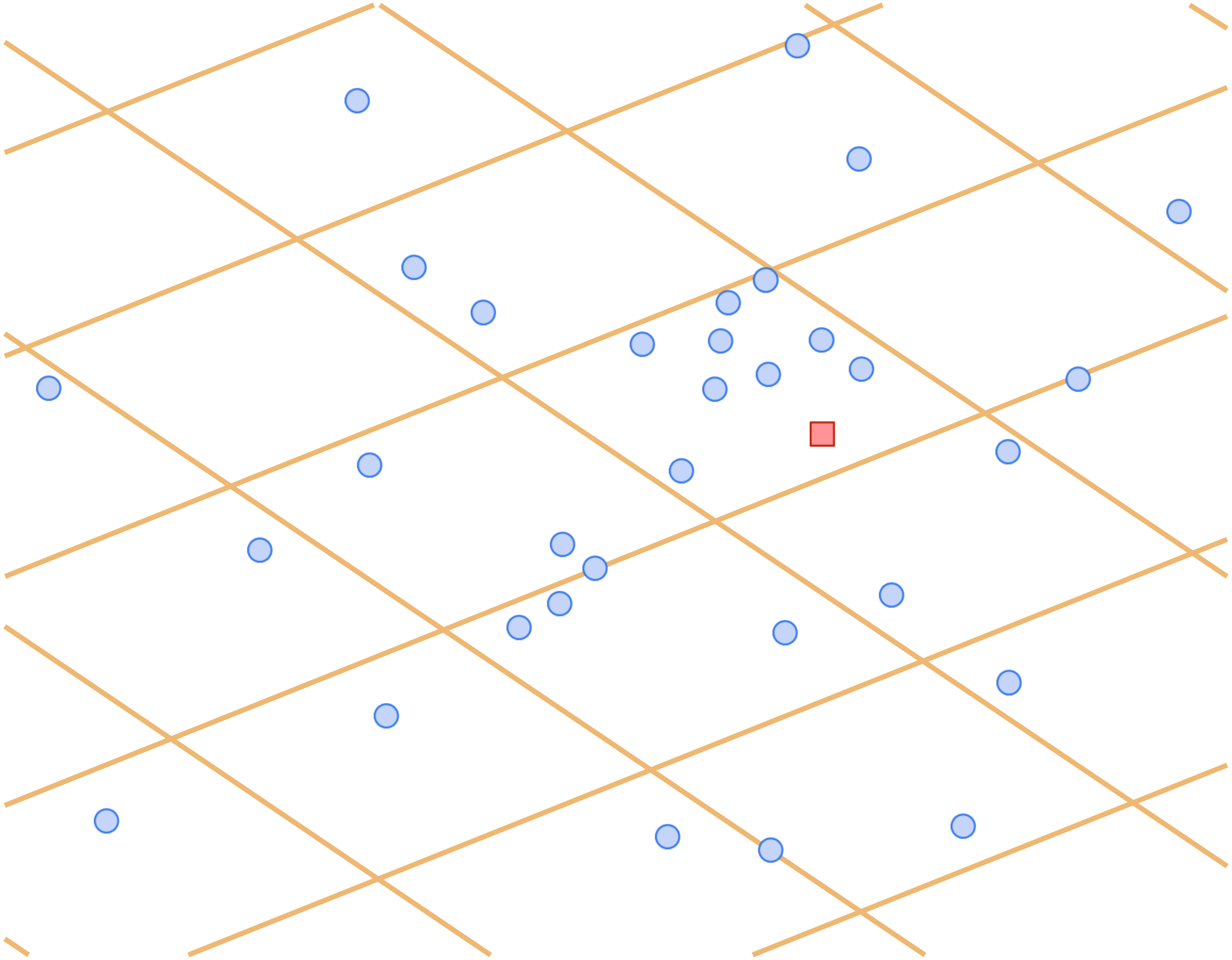


LSH

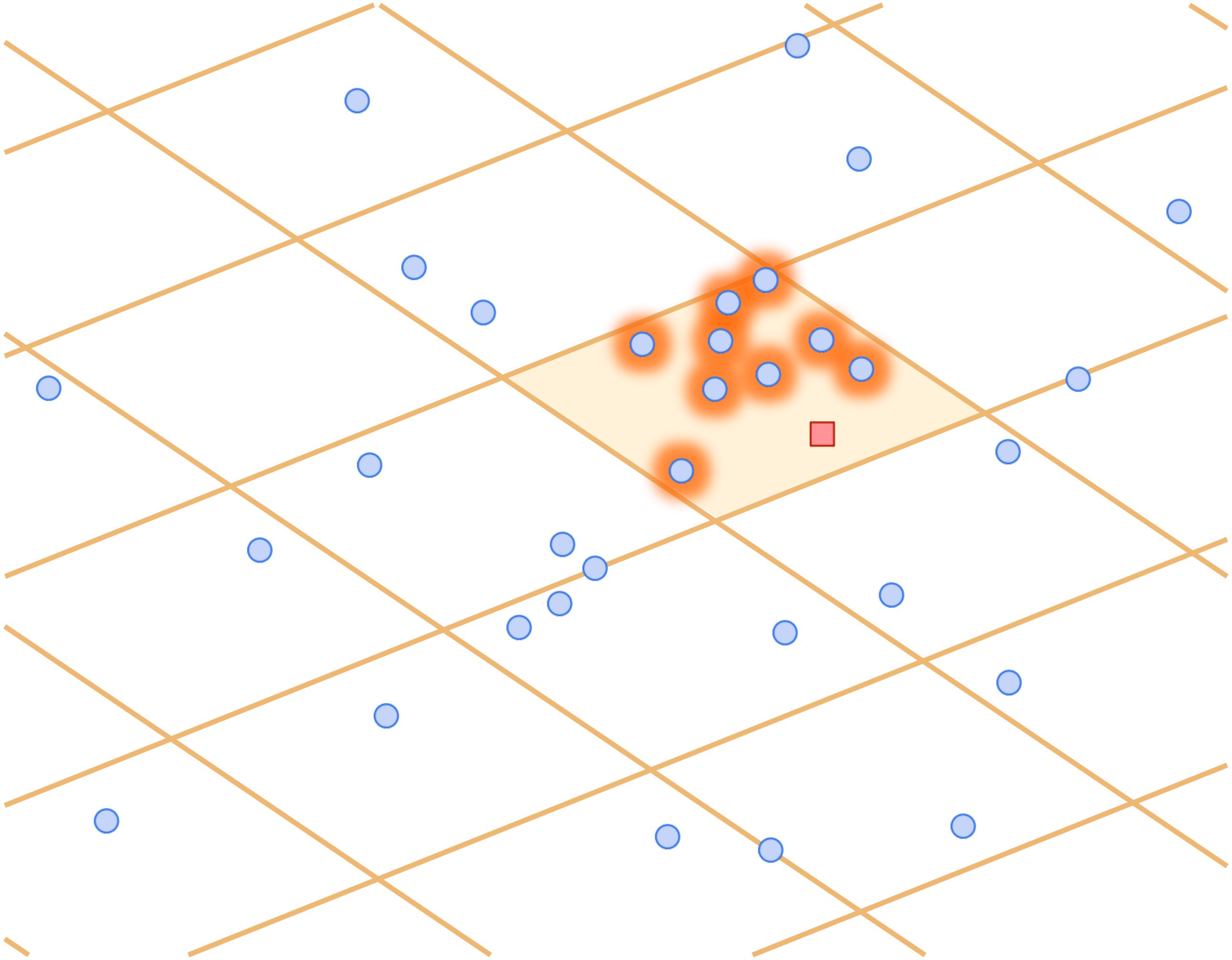


- As the ratio of surface area to volume grows in dimensionality, the number of overlapping grids grows in dimensionality.

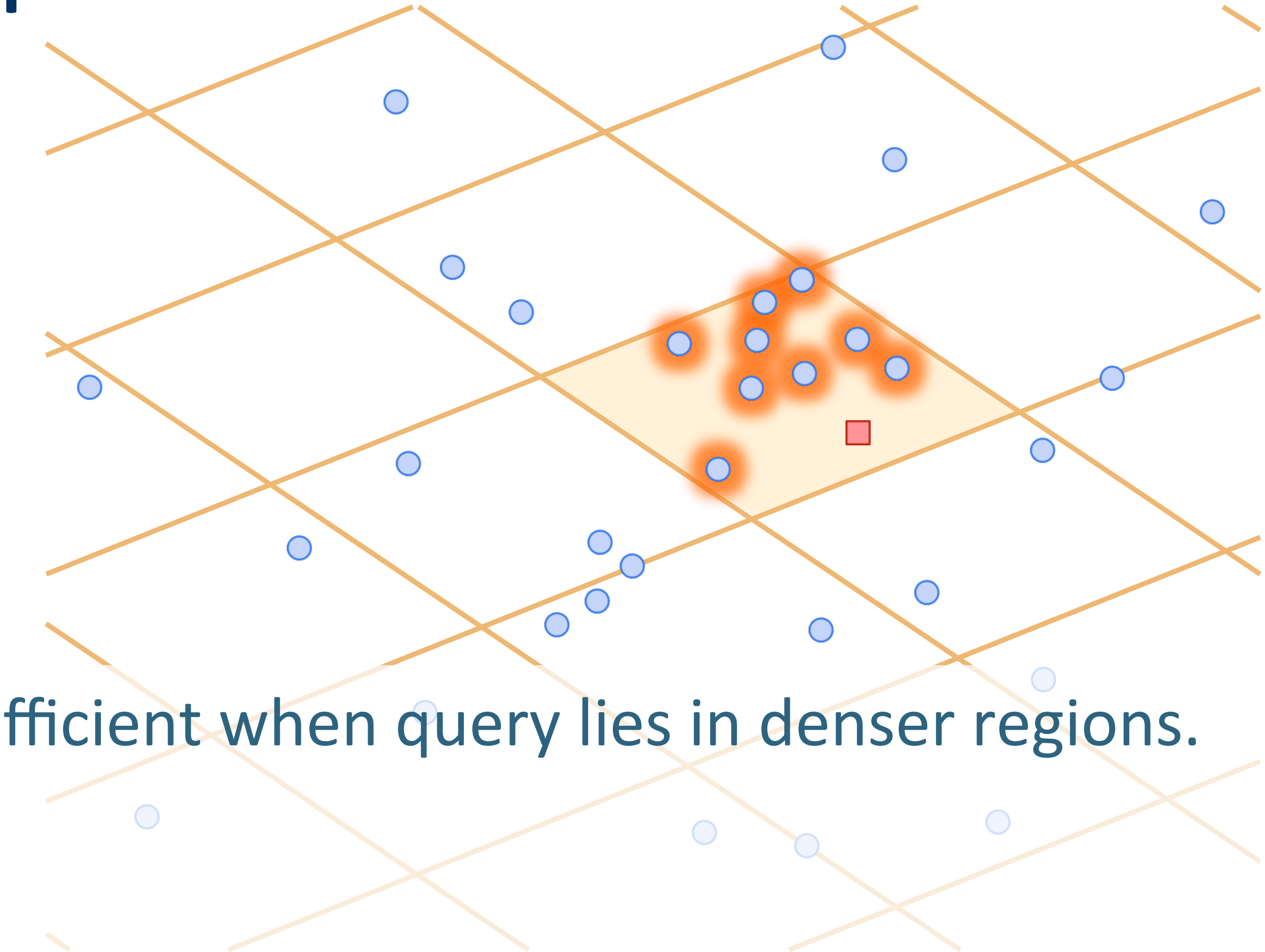
LSH



LSH

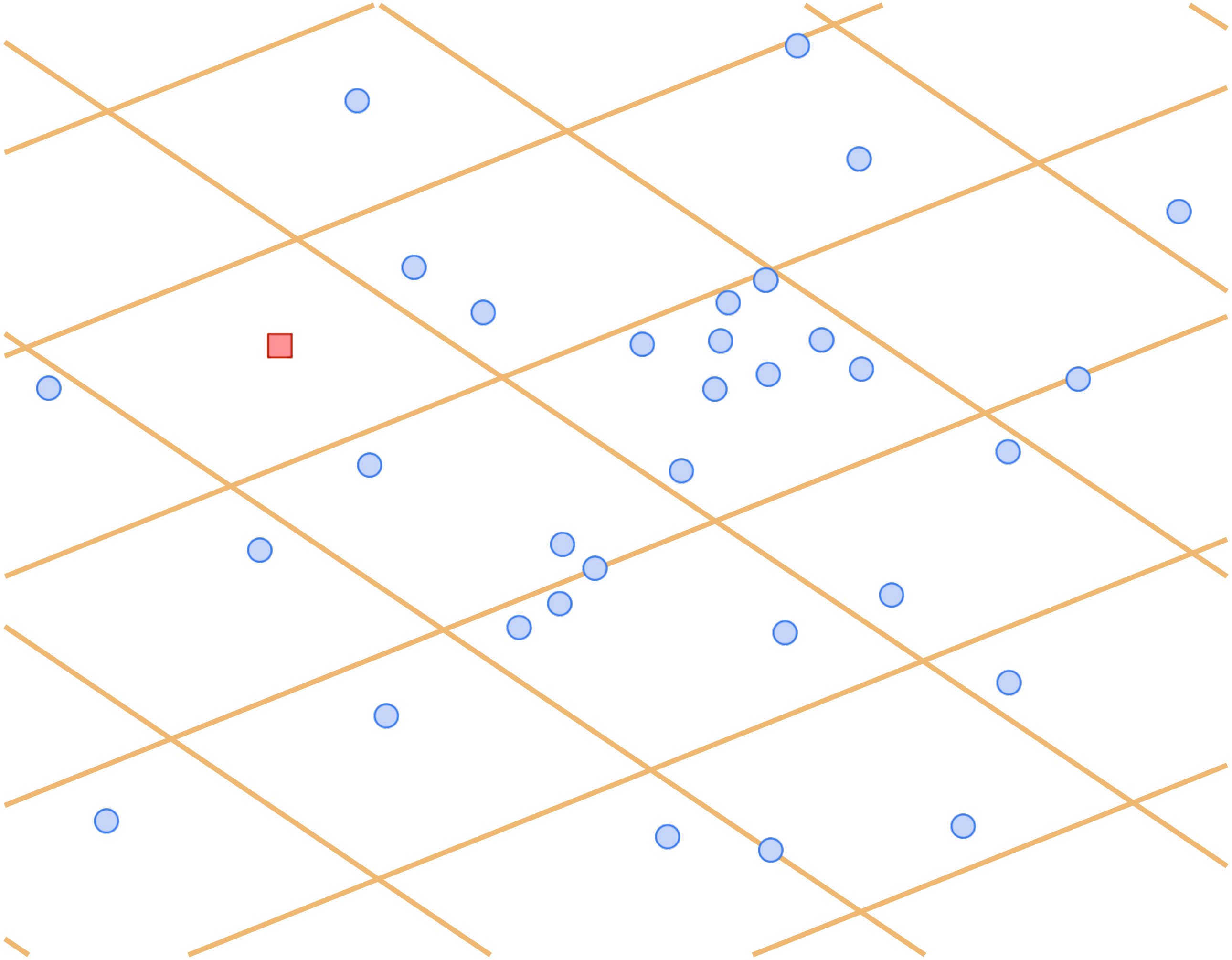


LSH

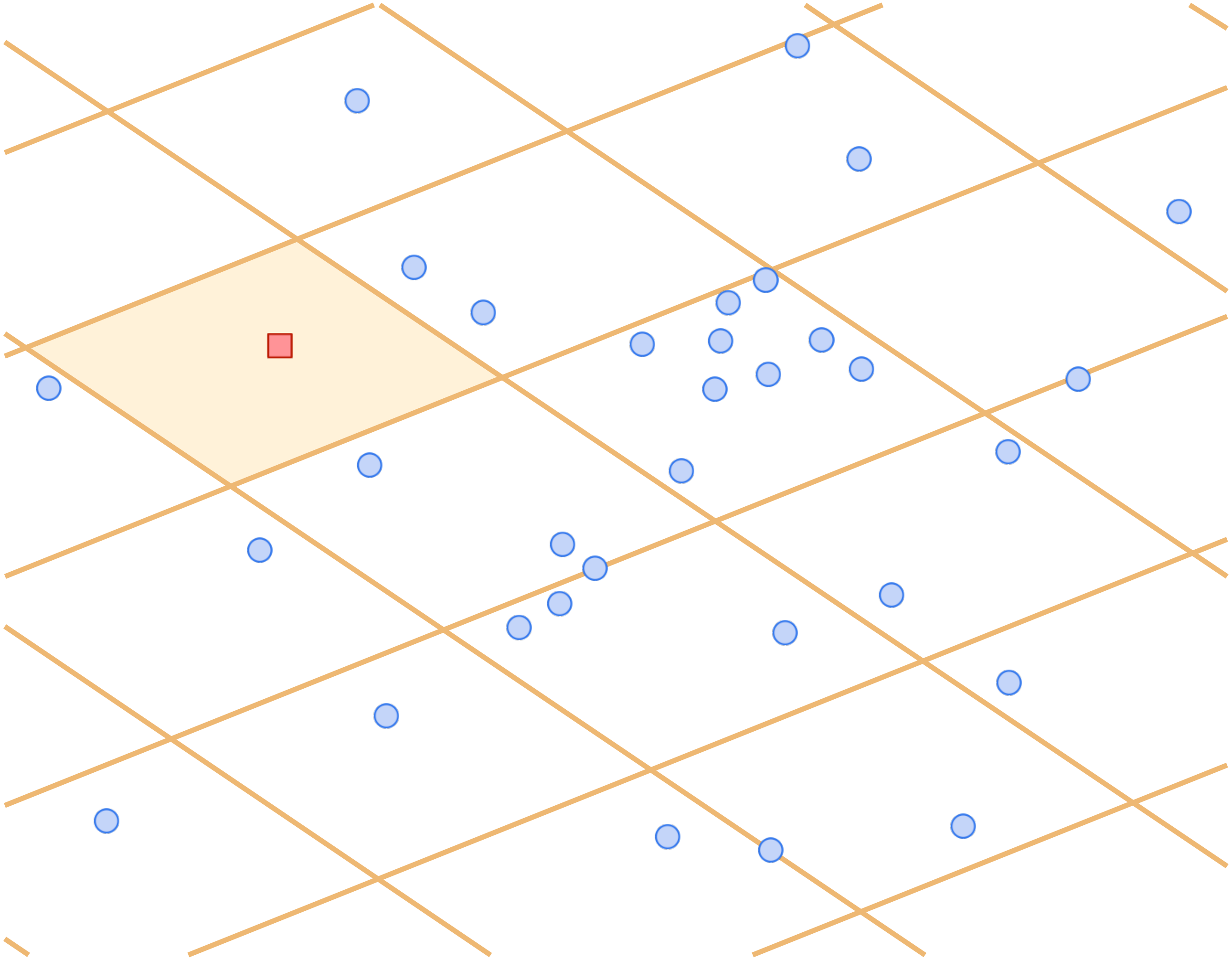


- Inefficient when query lies in denser regions.

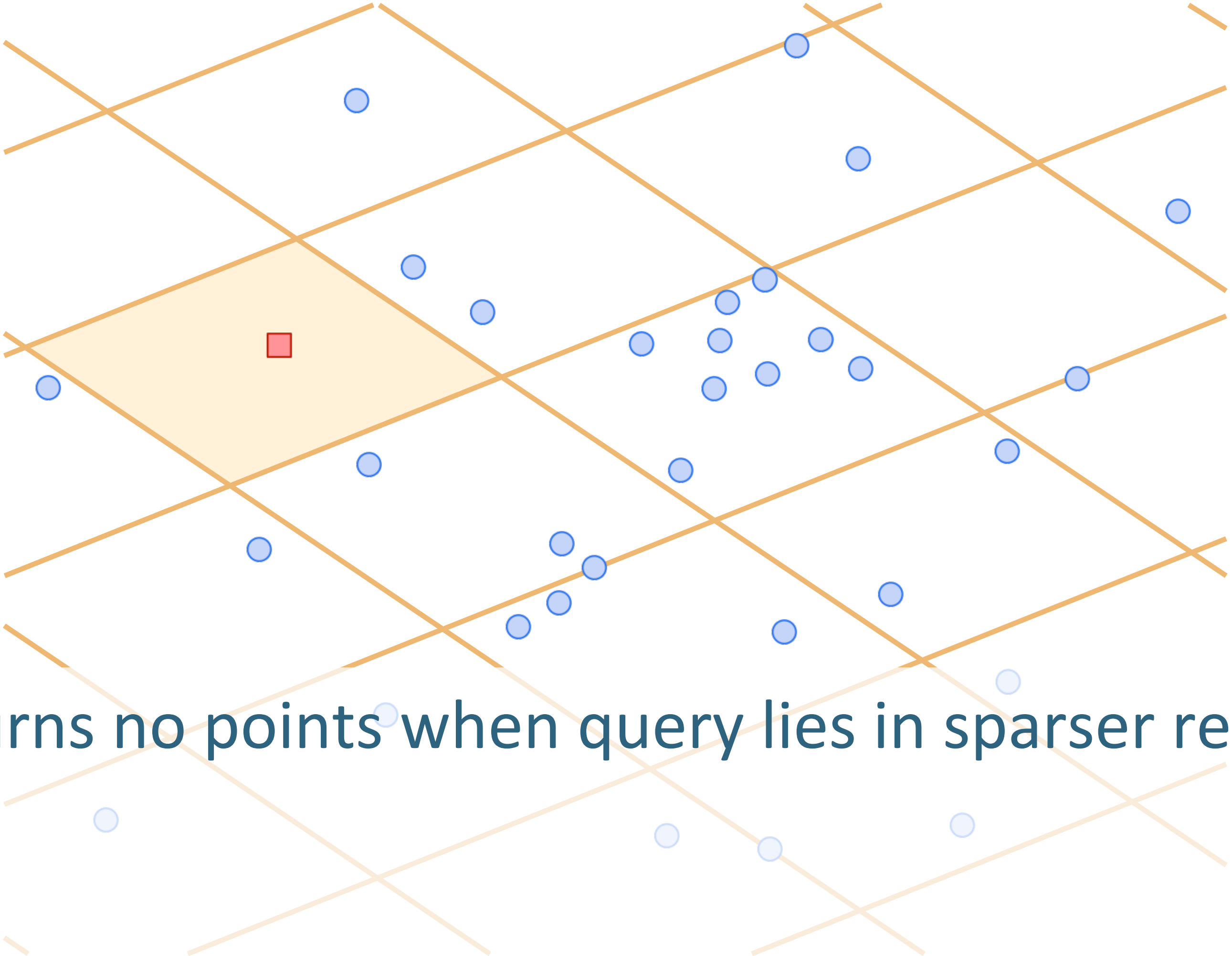
LSH



LSH

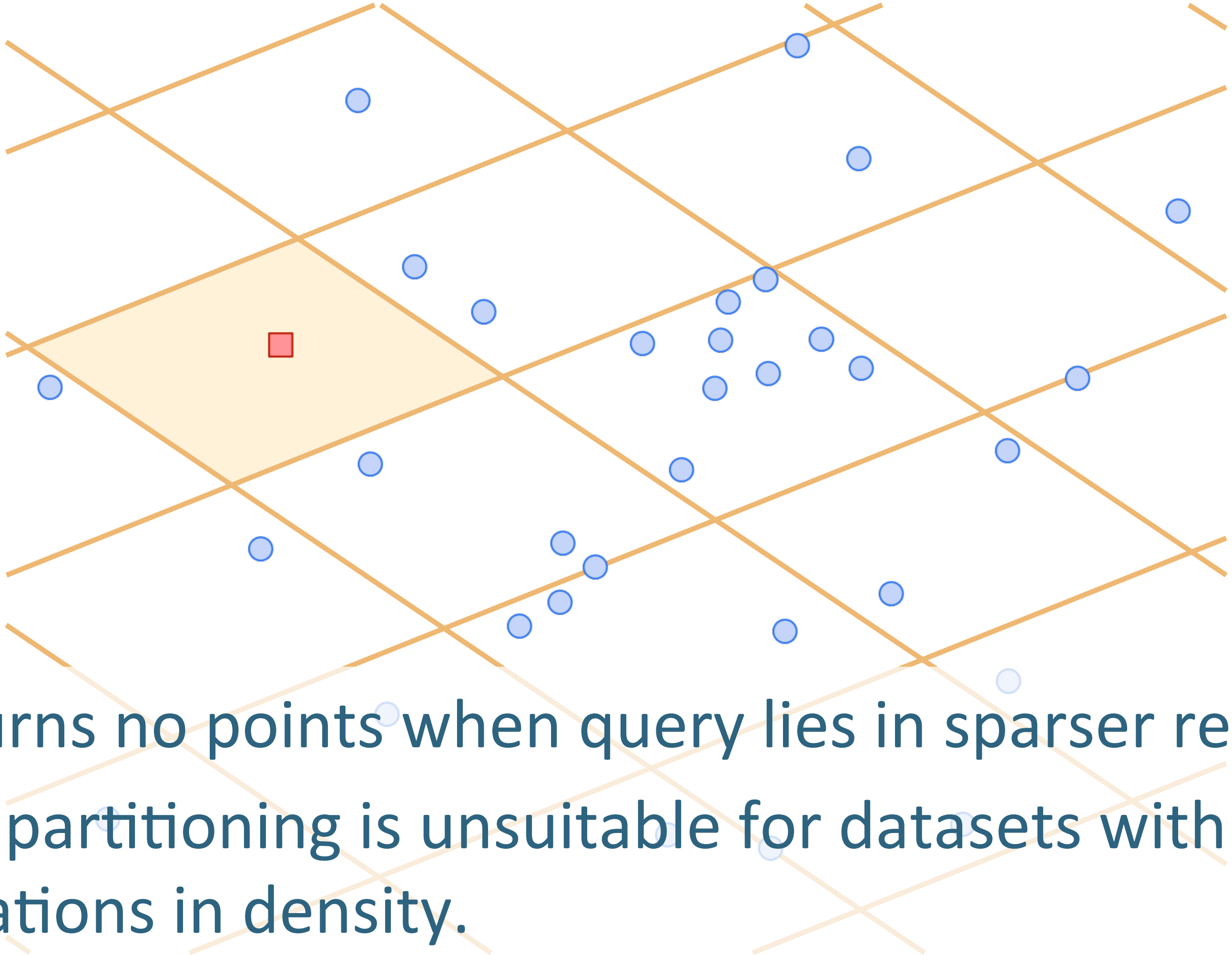


LSH



- Returns no points when query lies in sparser regions.

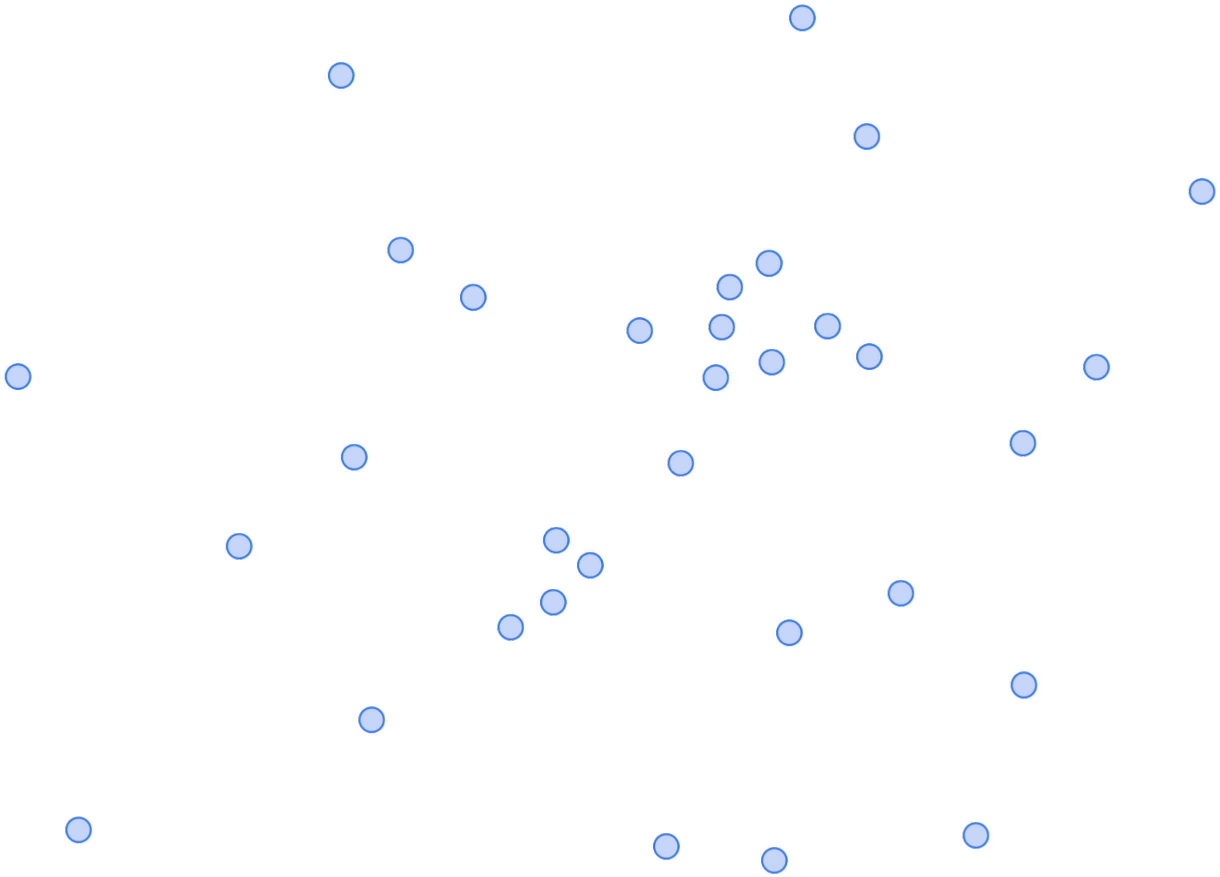
LSH



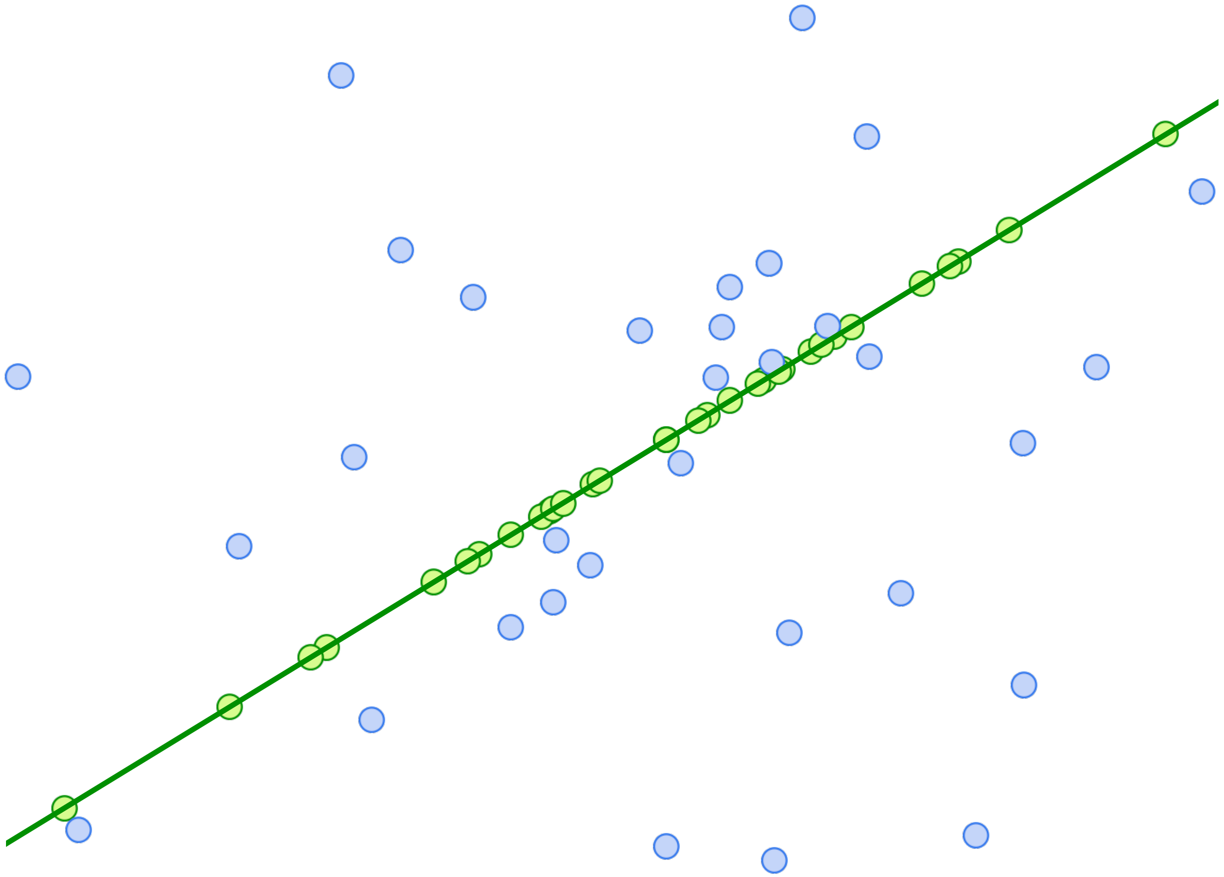
- Returns no points when query lies in sparser regions.
- This partitioning is unsuitable for datasets with large variations in density.

Prioritized DCI

Prioritized DCI

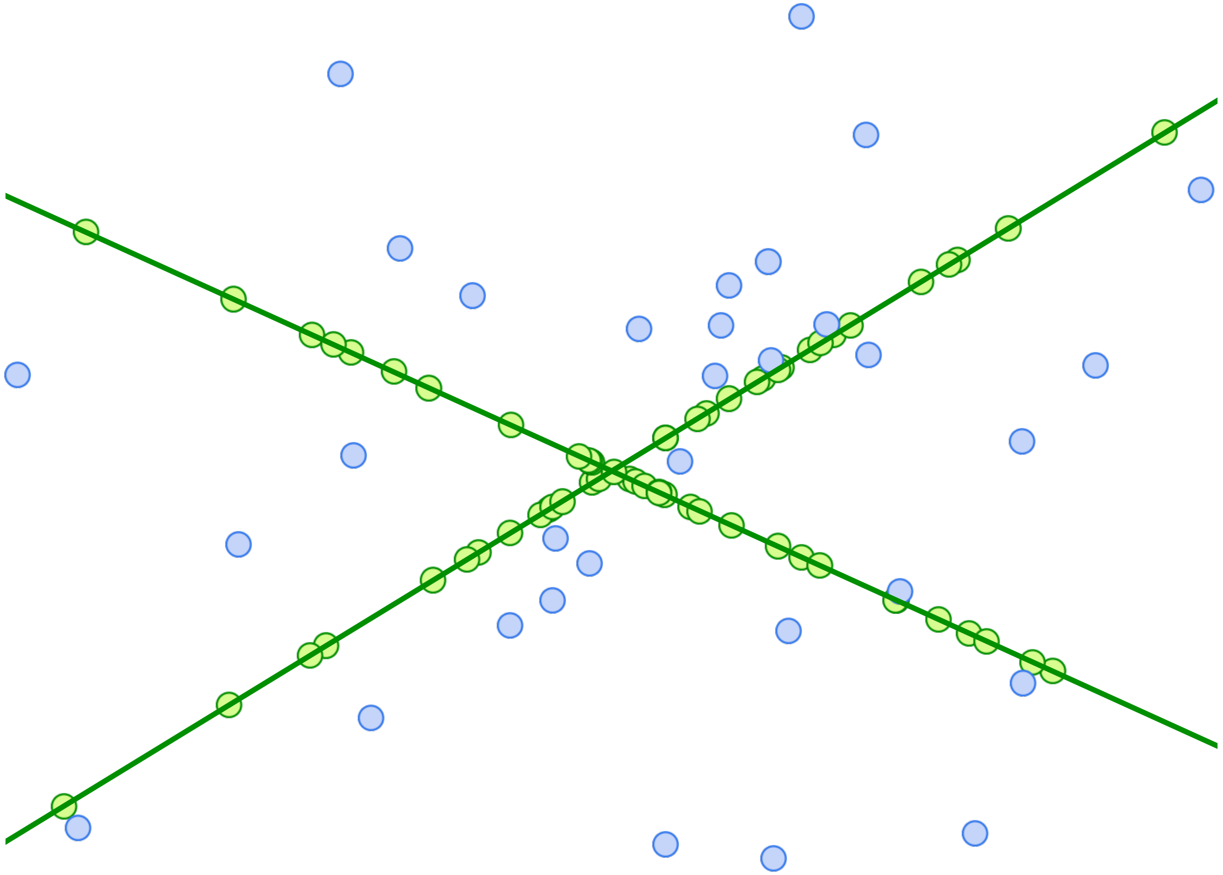


Prioritized DCI



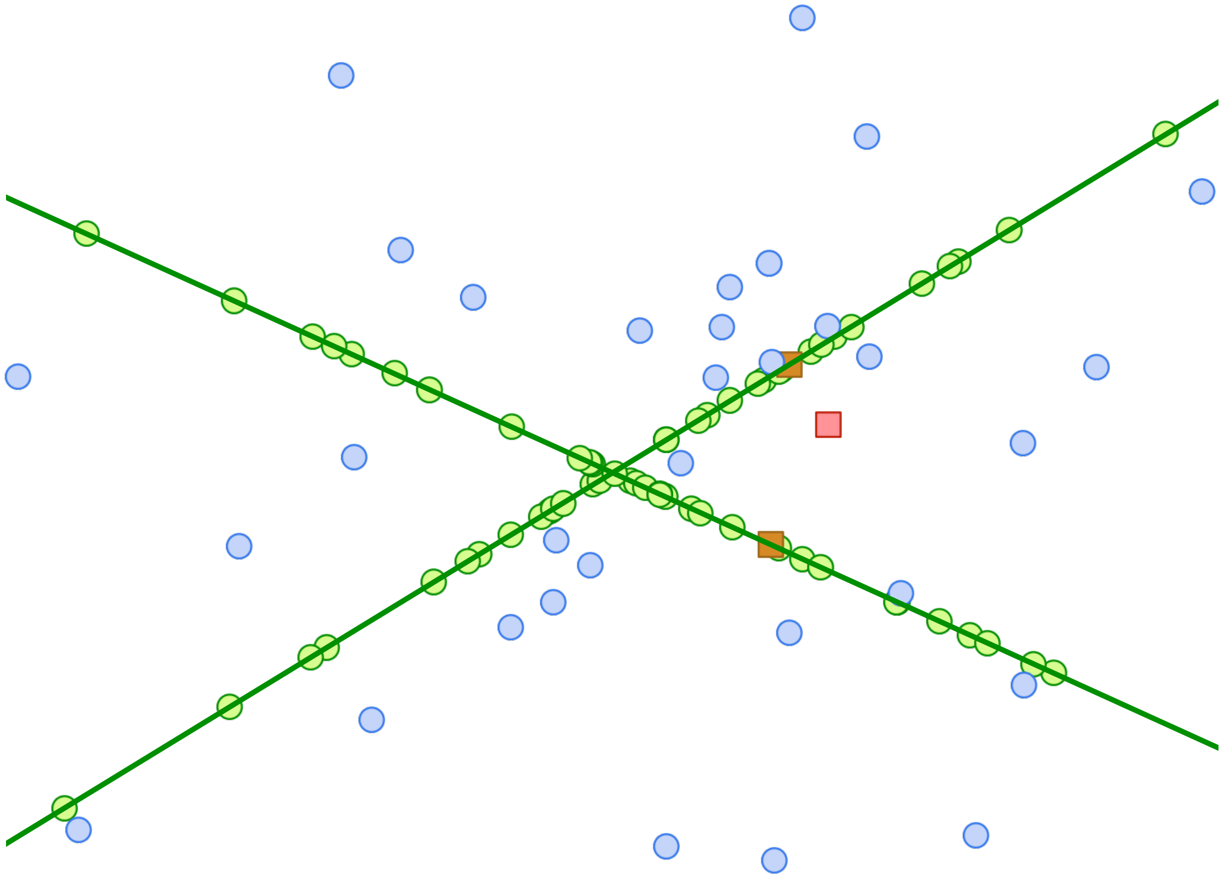
Project all data points along a random direction.

Prioritized DCI



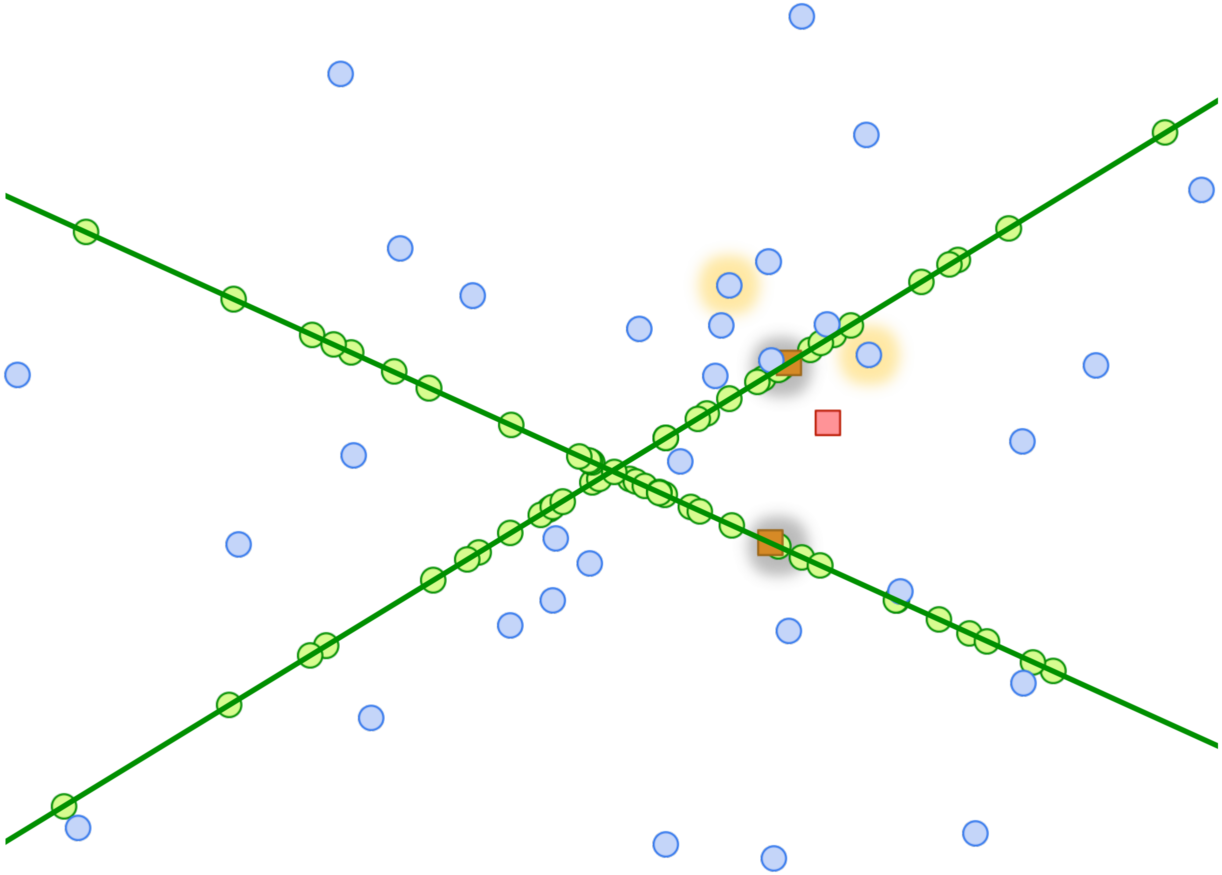
Project all data points along multiple random directions.

Prioritized DCI



Project the query along each projection direction.

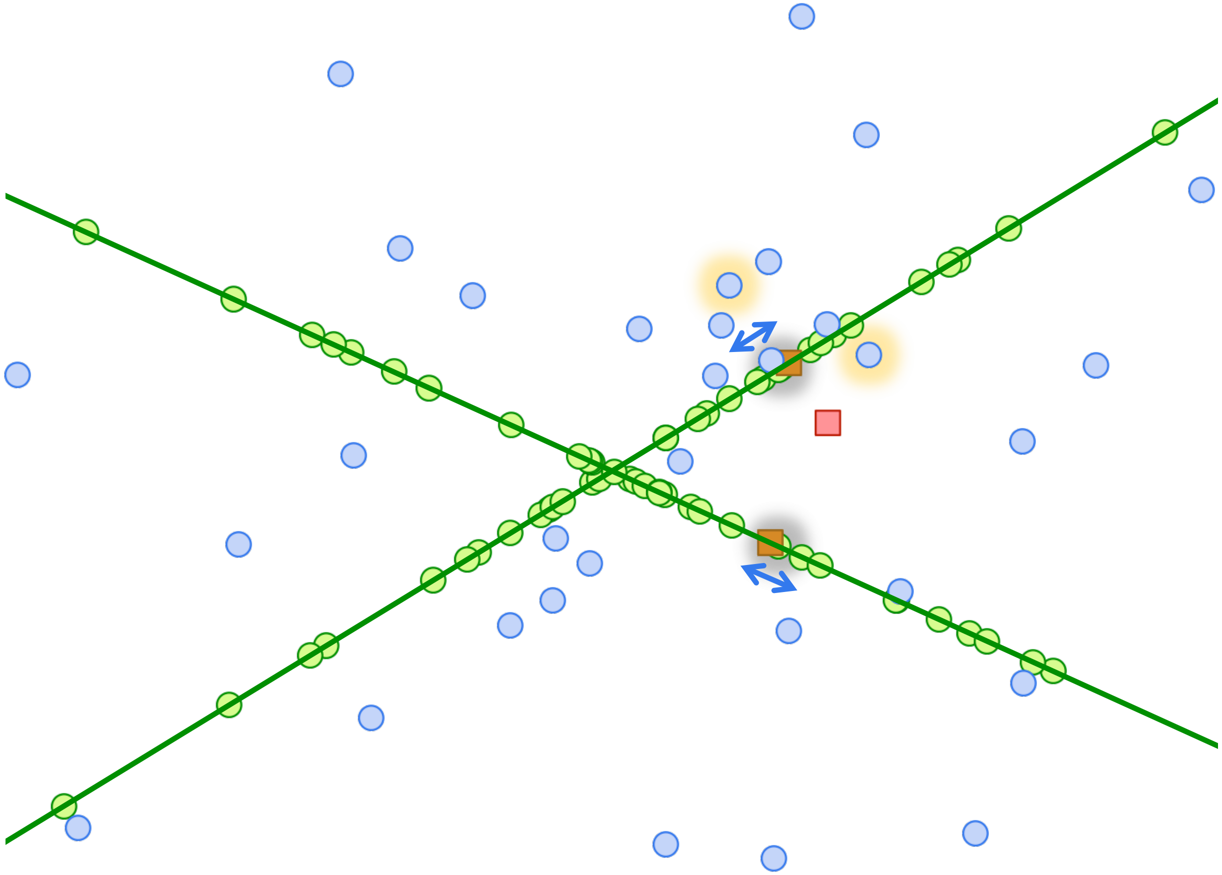
Prioritized DCI



Find the closest point to the query along each projection direction and add them to the frontier.

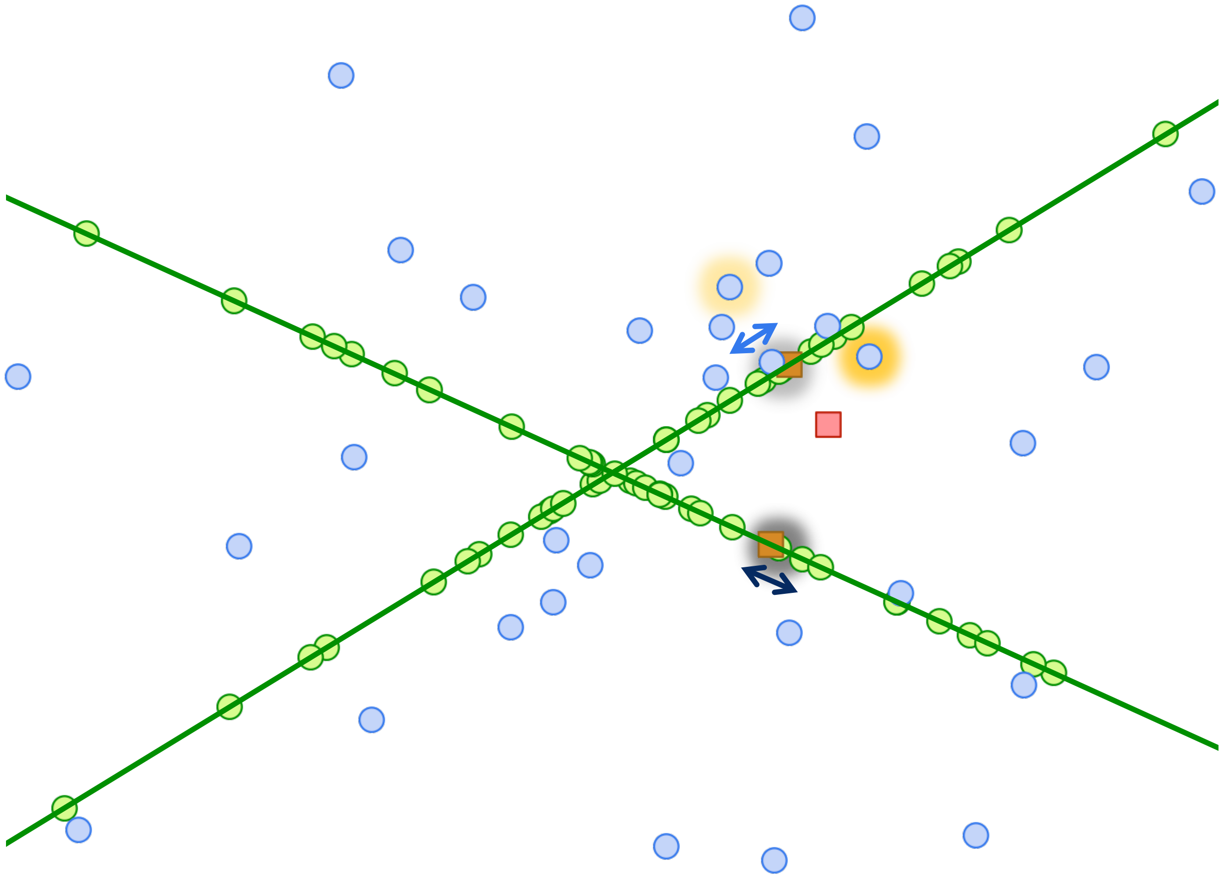
Berkeley
UNIVERSITY OF CALIFORNIA

Prioritized DCI



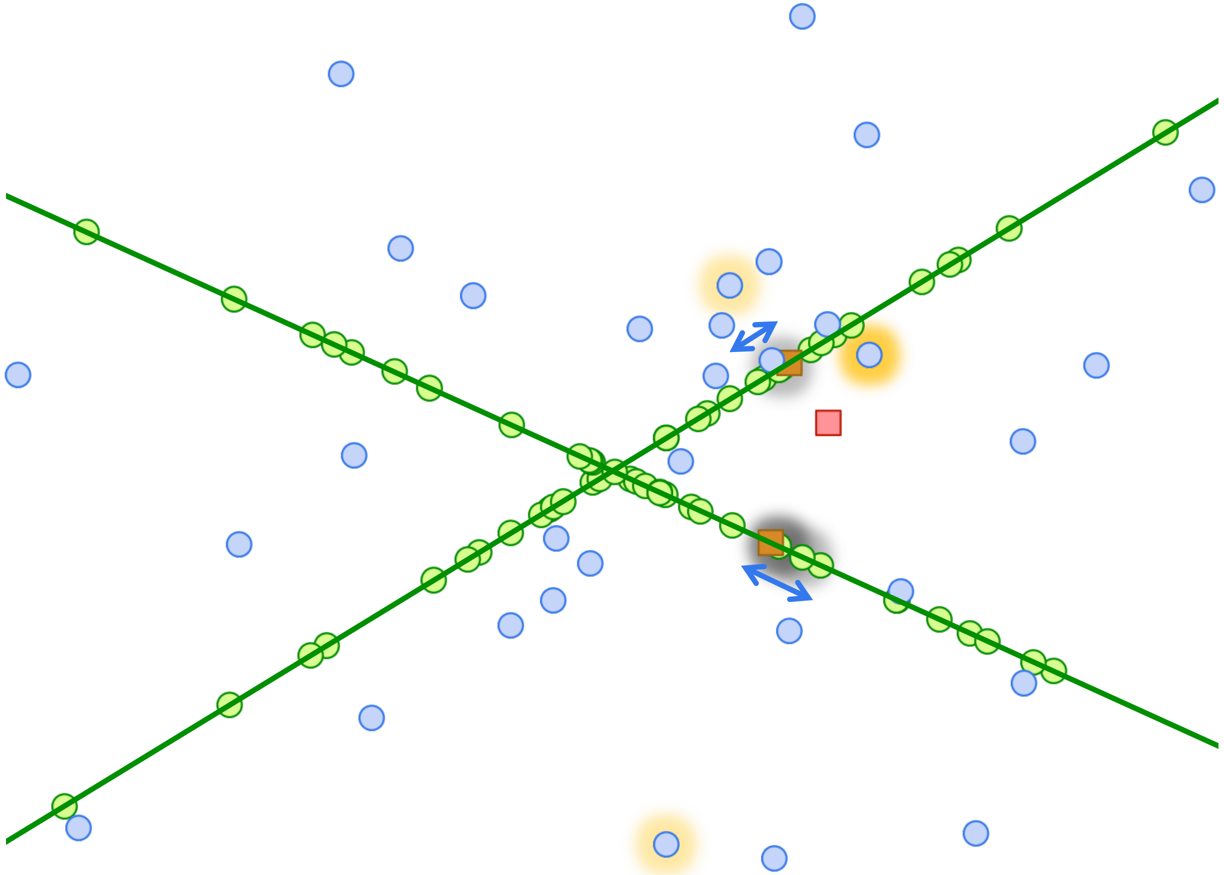
Compare their projected distances to the query.

Prioritized DCI



Visit the point with the shortest projected distance.

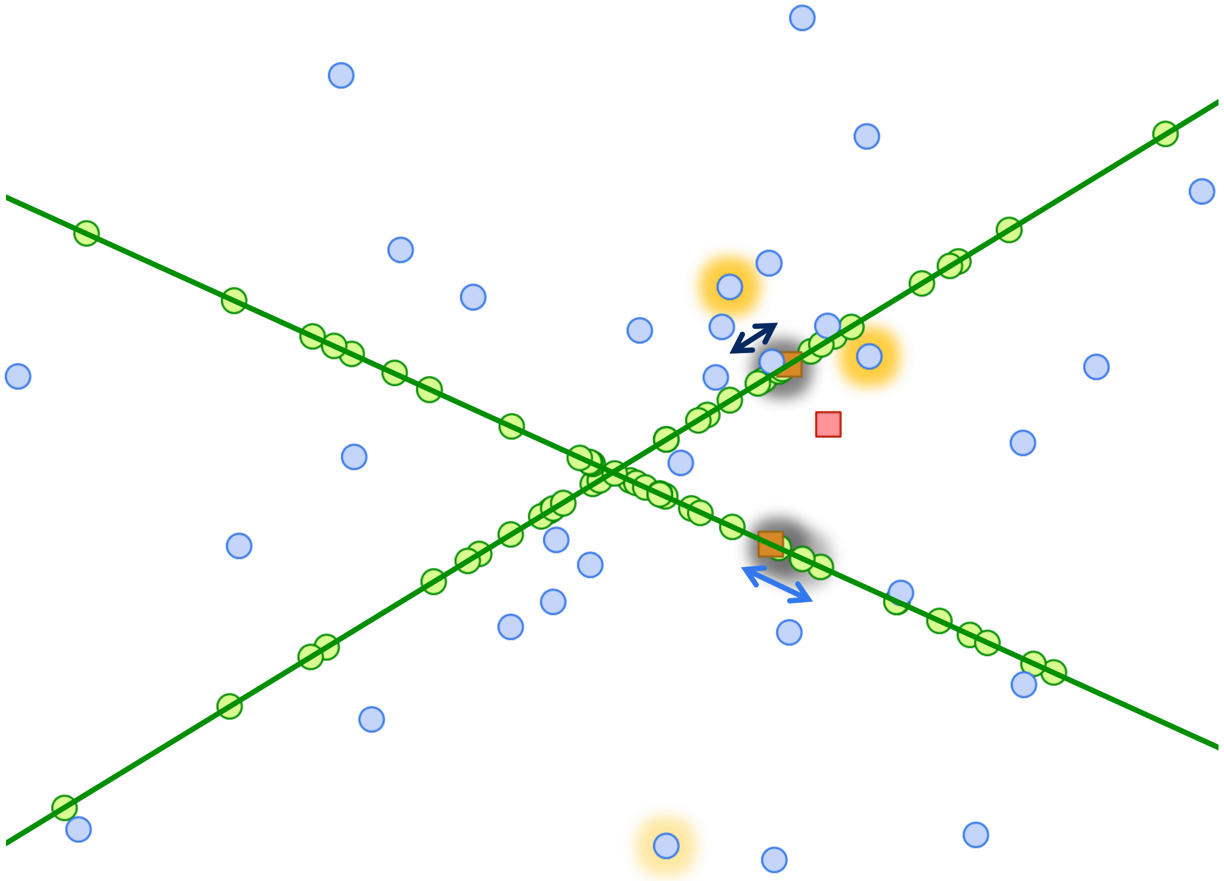
Prioritized DCI



Find the next closest point along the projection direction that has just been processed and add it to the frontier.

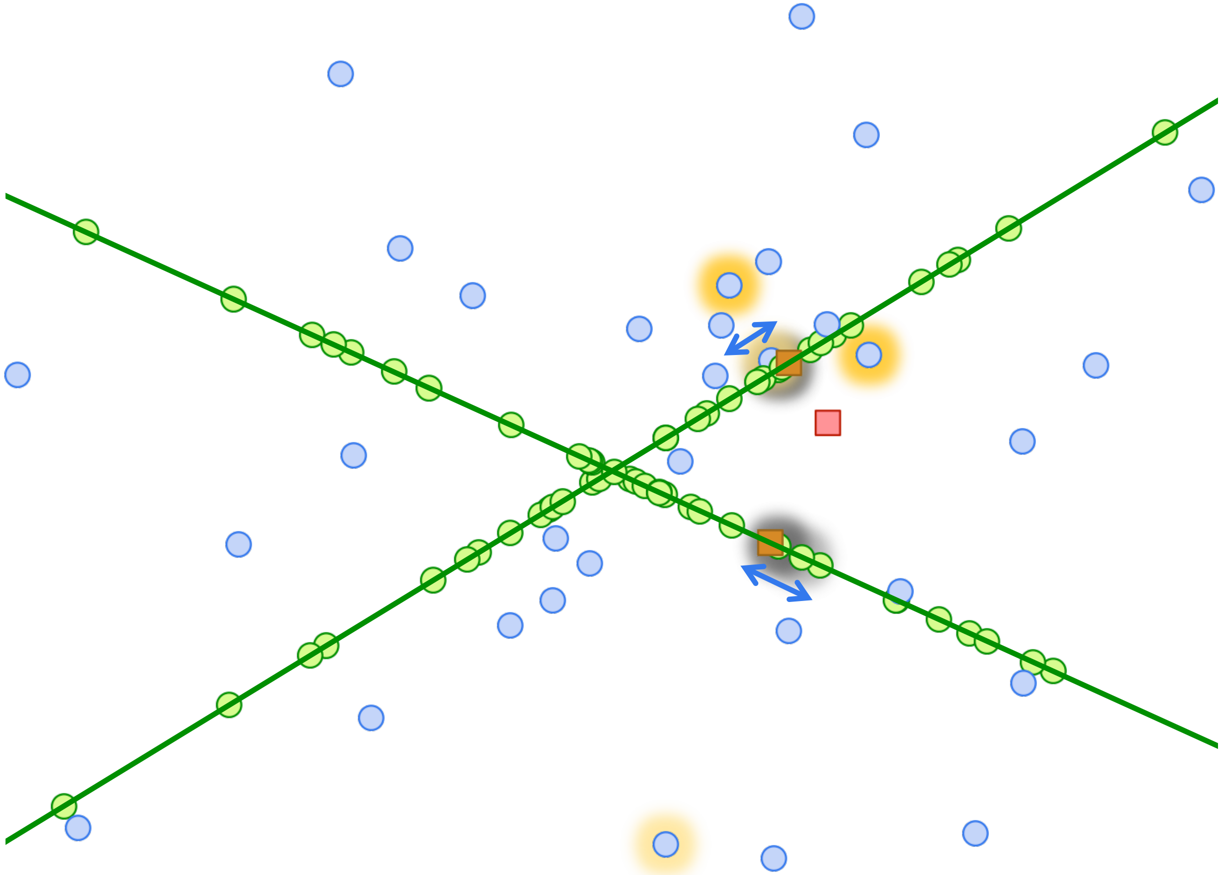
Berkeley
UNIVERSITY OF CALIFORNIA

Prioritized DCI



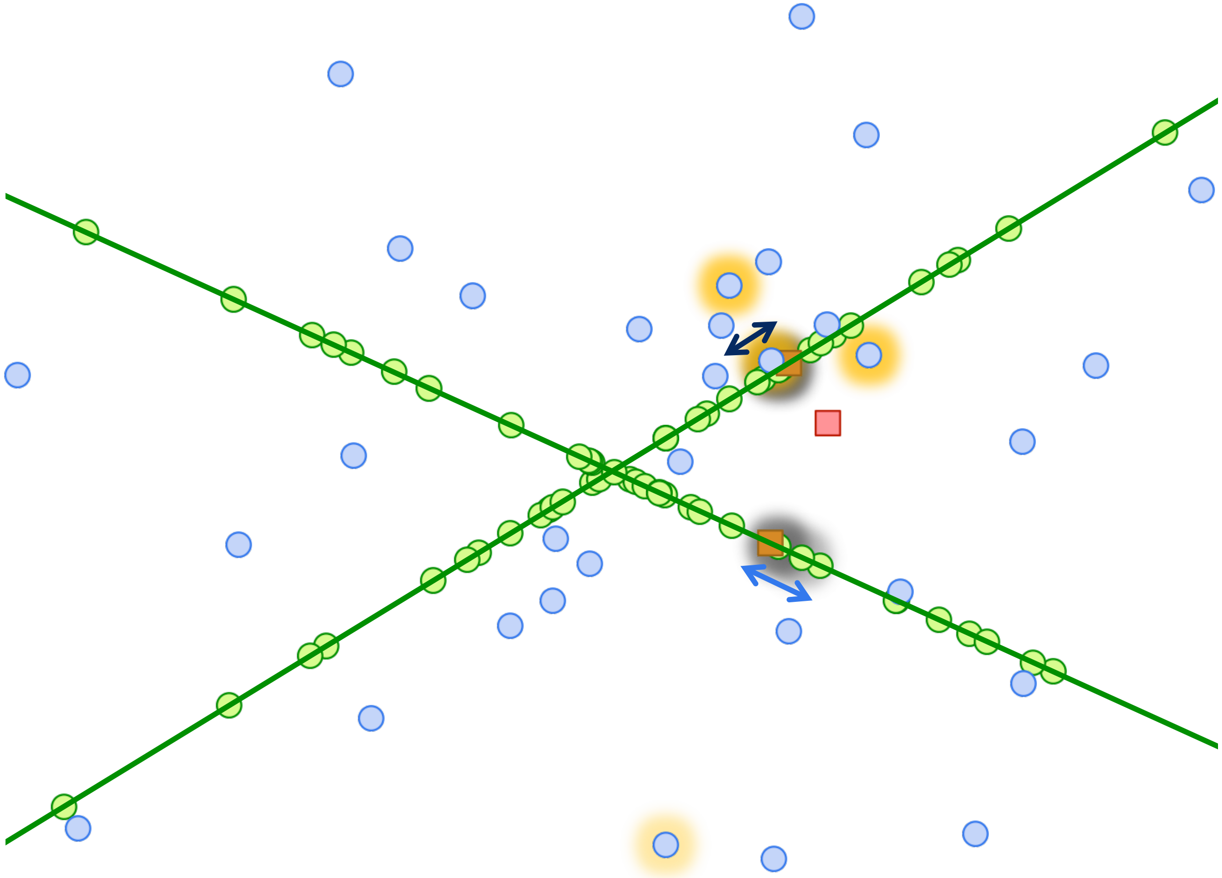
Compare projected distances of points on the frontier and visit the one with the shortest projected distance.

Prioritized DCI



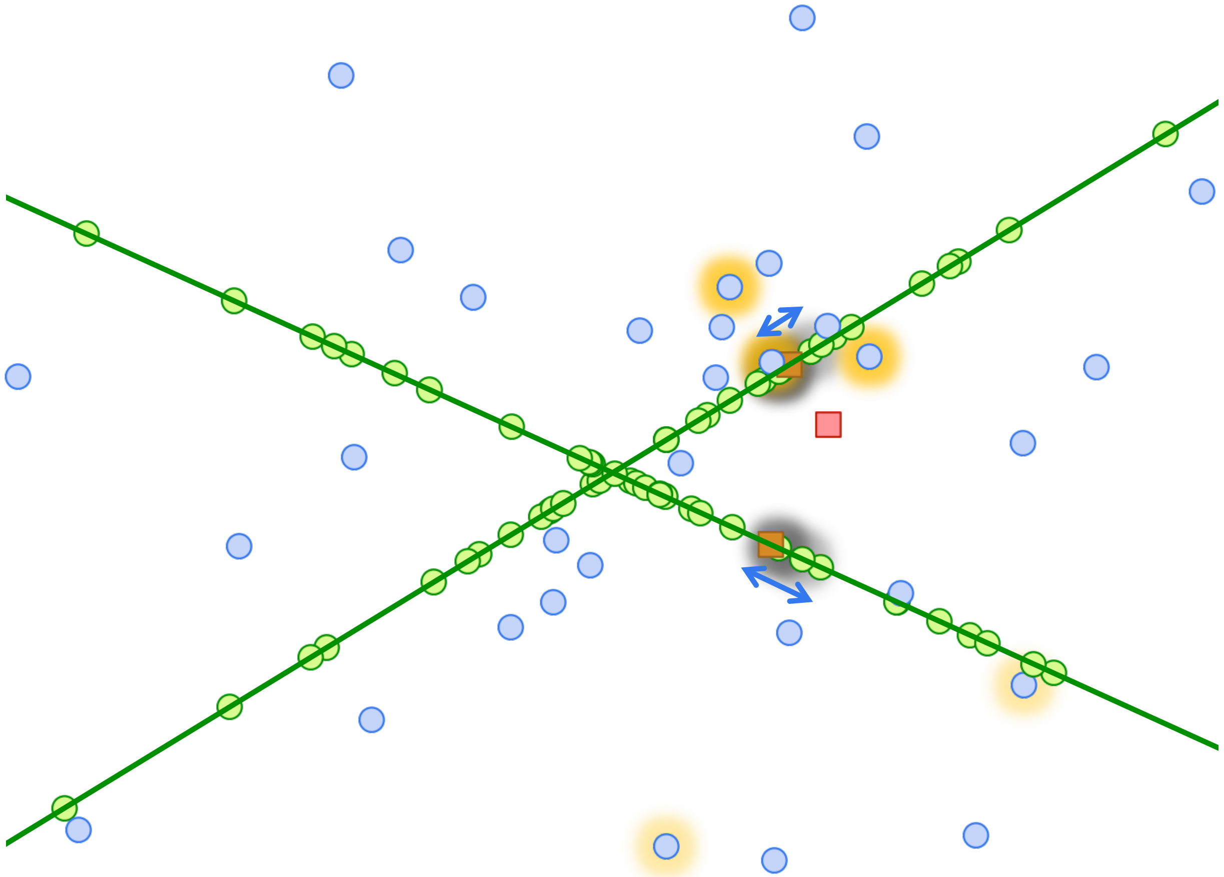
Find the next closest point along the projection direction that has just been processed and add it to the frontier.

Prioritized DCI



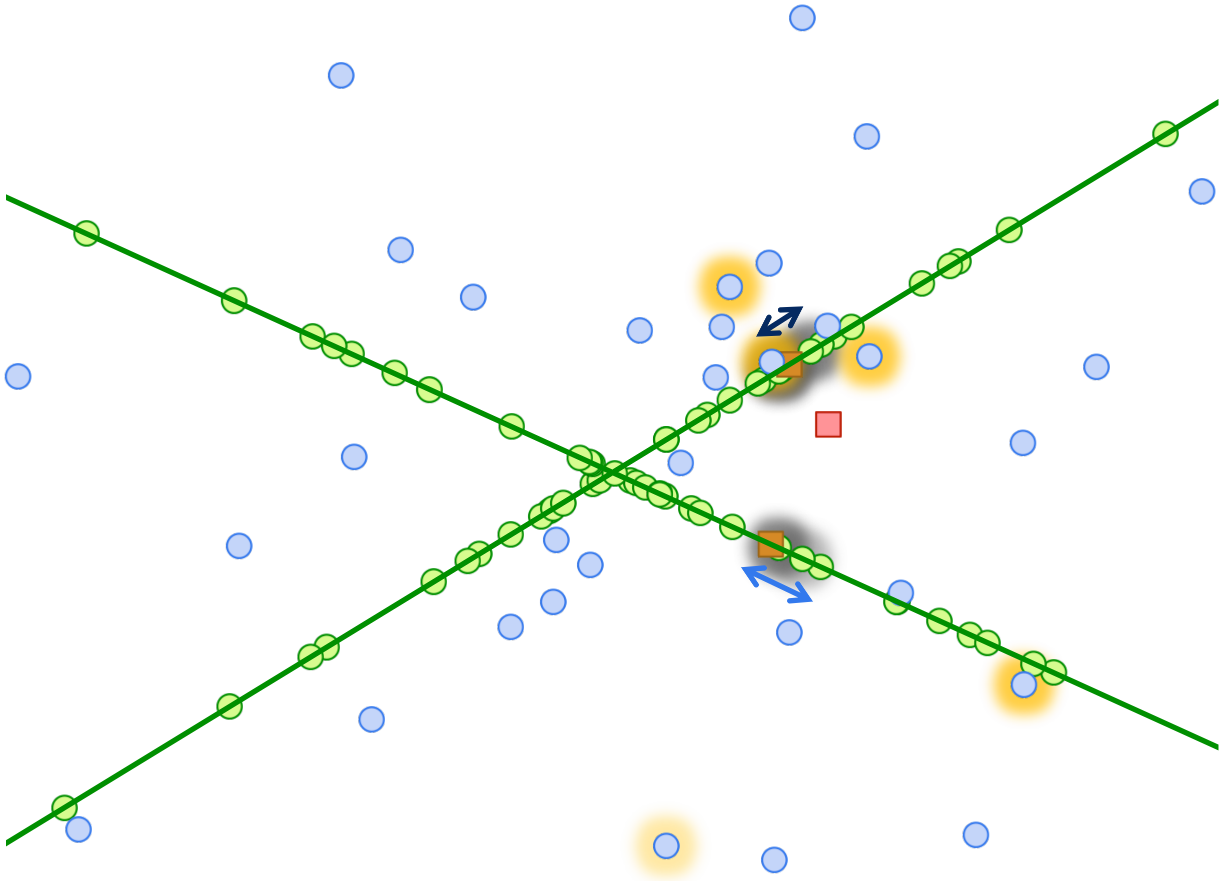
Compare projected distances of points on the frontier and visit the one with the shortest projected distance.

Prioritized DCI



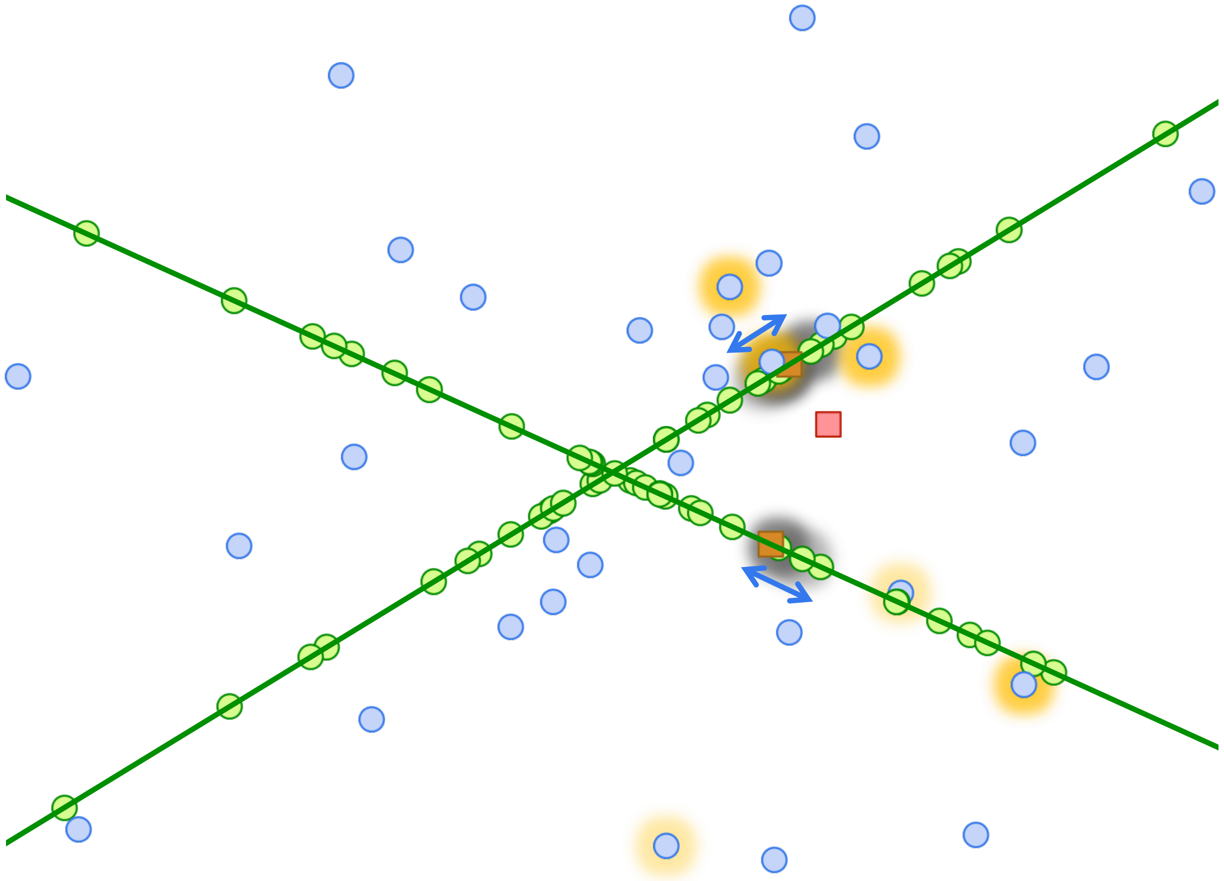
Find the next closest point along the projection direction that has just been processed and add it to the frontier.

Prioritized DCI



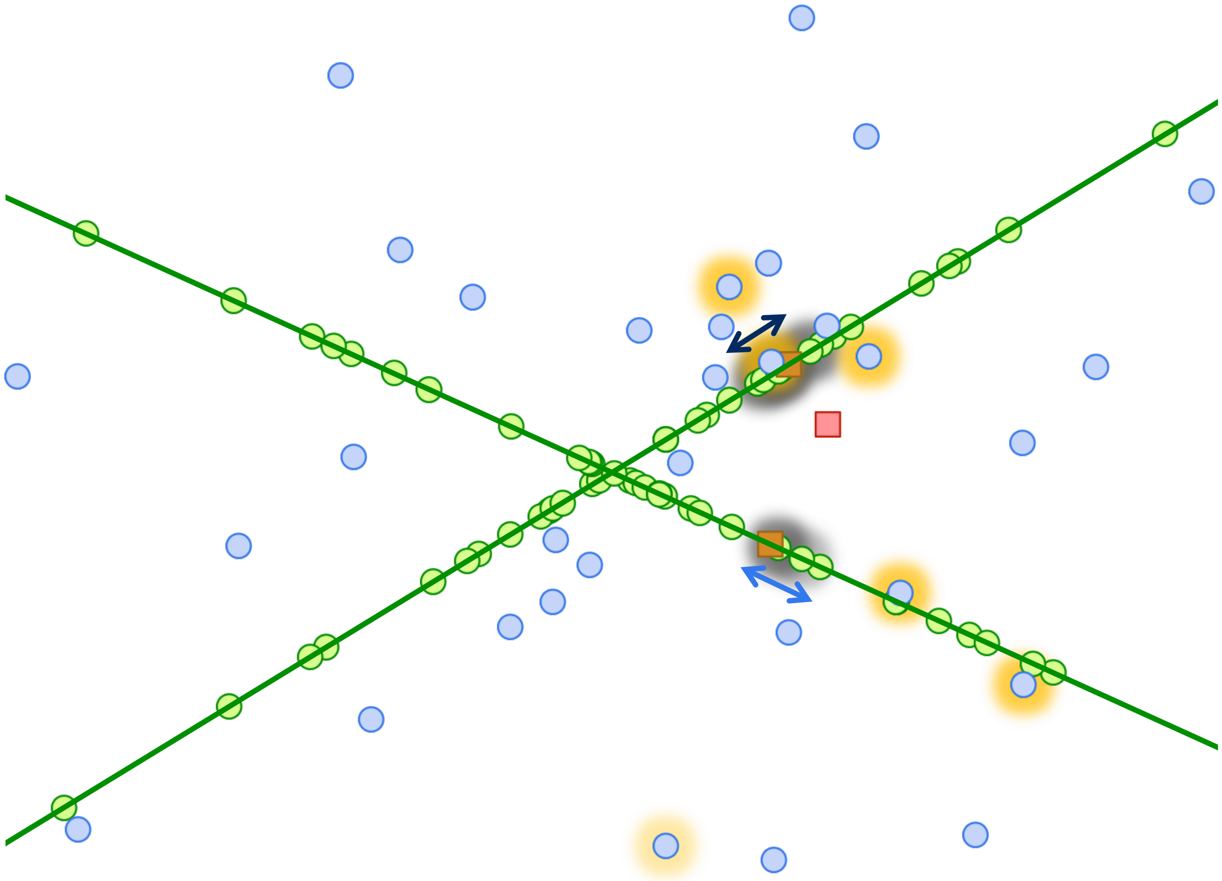
Compare projected distances of points on the frontier and visit the one with the shortest projected distance.

Prioritized DCI



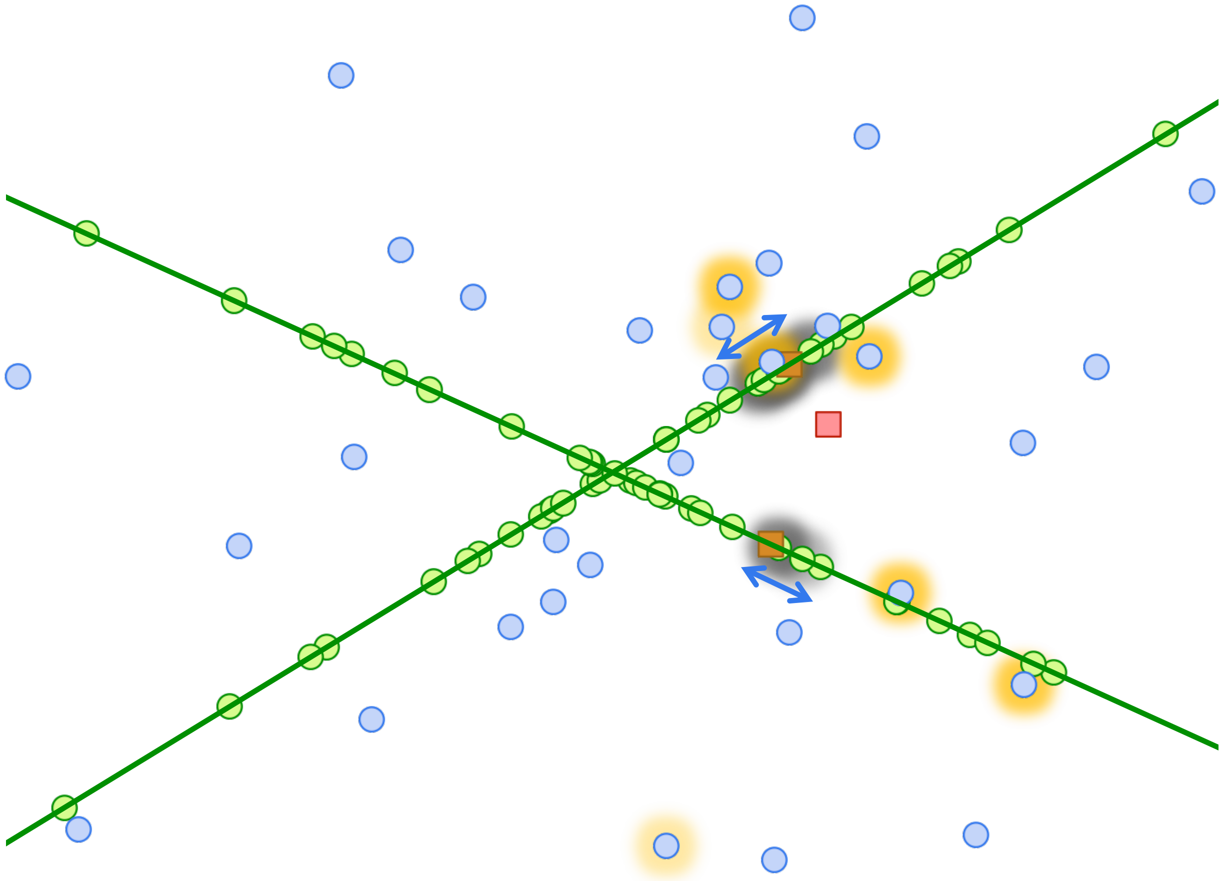
Find the next closest point along the projection direction that has just been processed and add it to the frontier.

Prioritized DCI



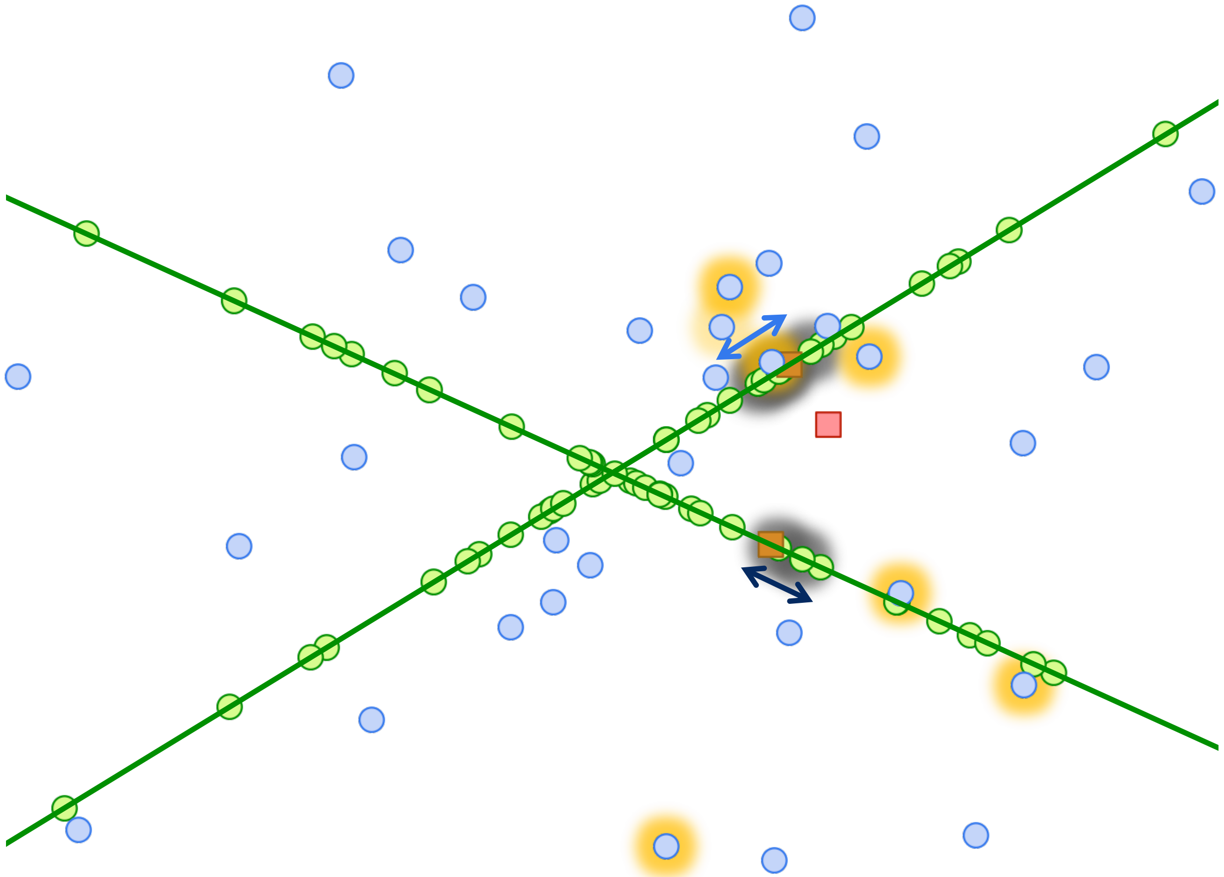
Compare projected distances of points on the frontier and visit the one with the shortest projected distance.

Prioritized DCI



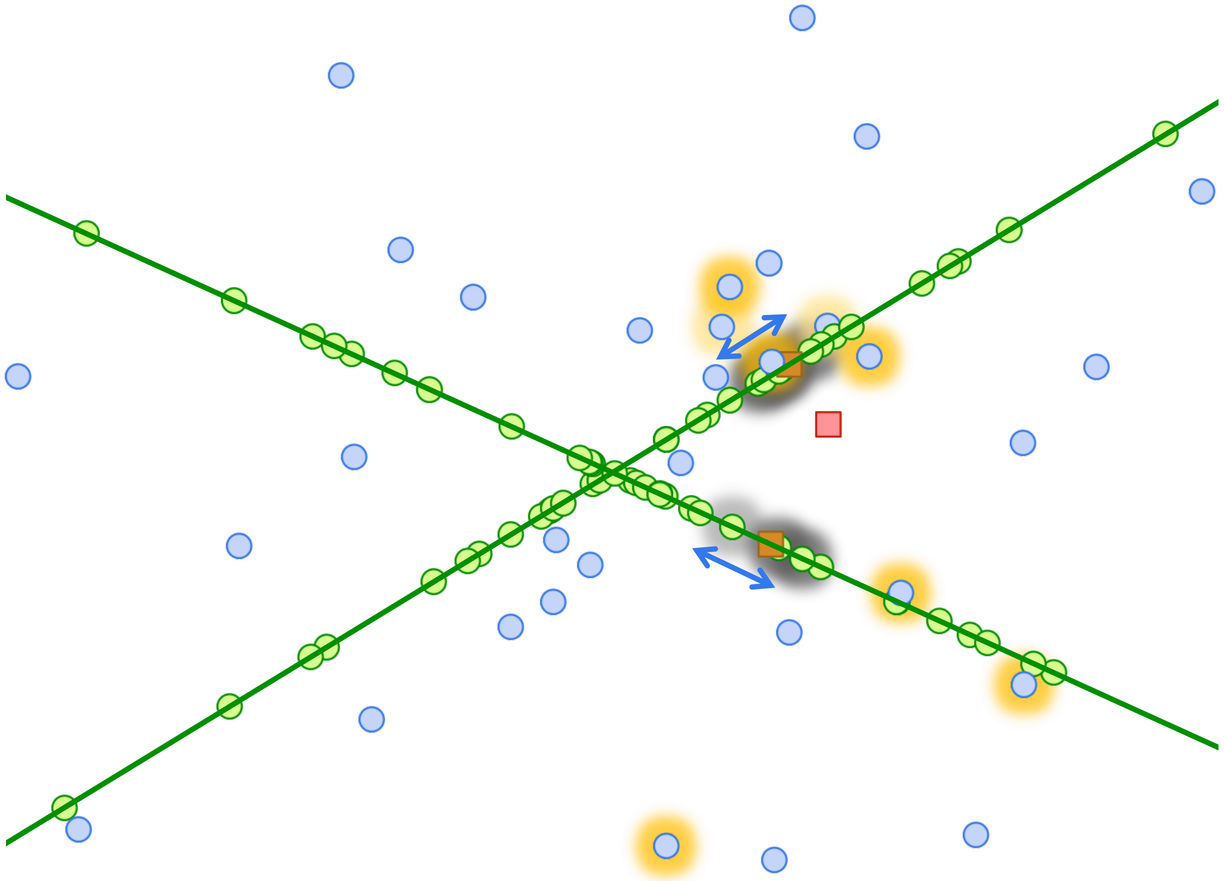
Find the next closest point along the projection direction that has just been processed and add it to the frontier.

Prioritized DCI



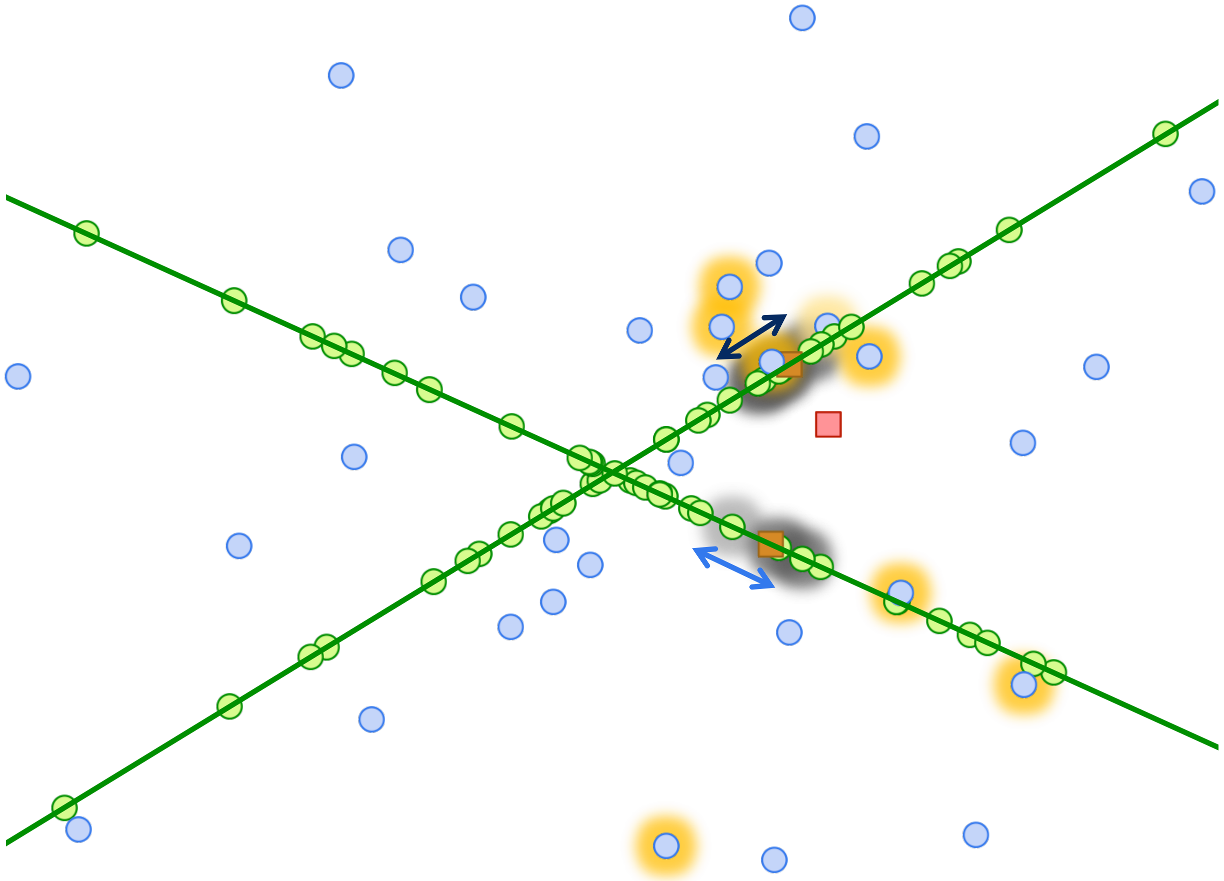
Compare projected distances of points on the frontier and visit the one with the shortest projected distance.

Prioritized DCI



Find the next closest point along the projection direction that has just been processed and add it to the frontier.

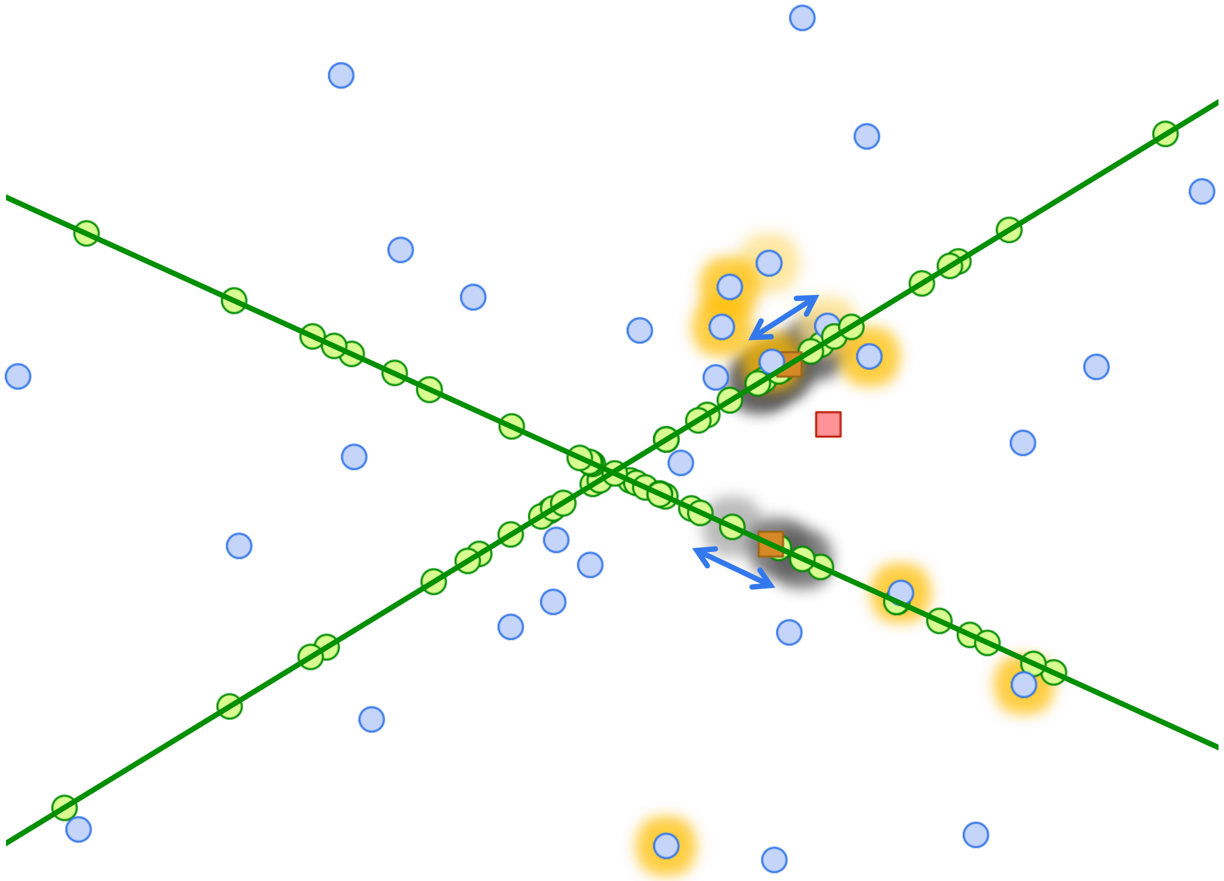
Prioritized DCI



Compare projected distances of points on the frontier and visit the one with the shortest projected distance.

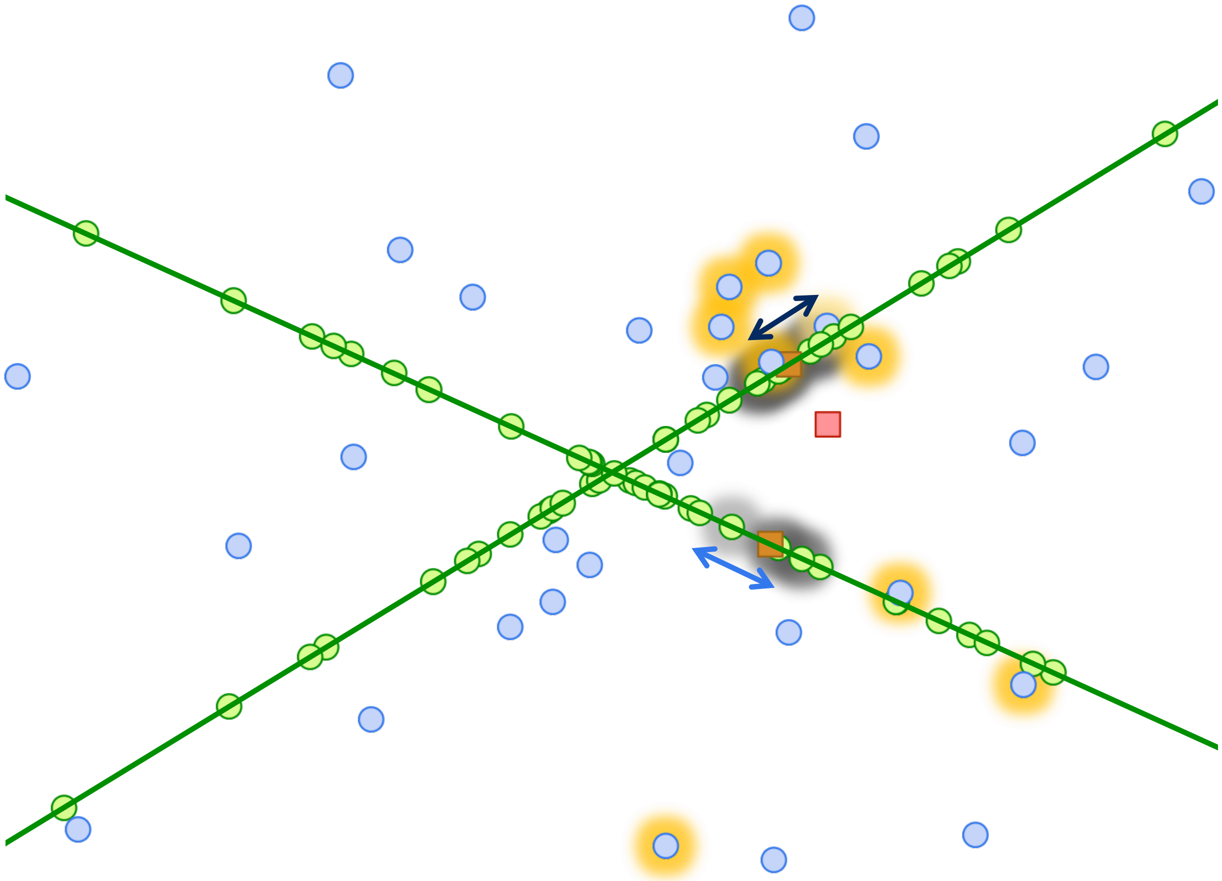
Berkeley
UNIVERSITY OF CALIFORNIA

Prioritized DCI



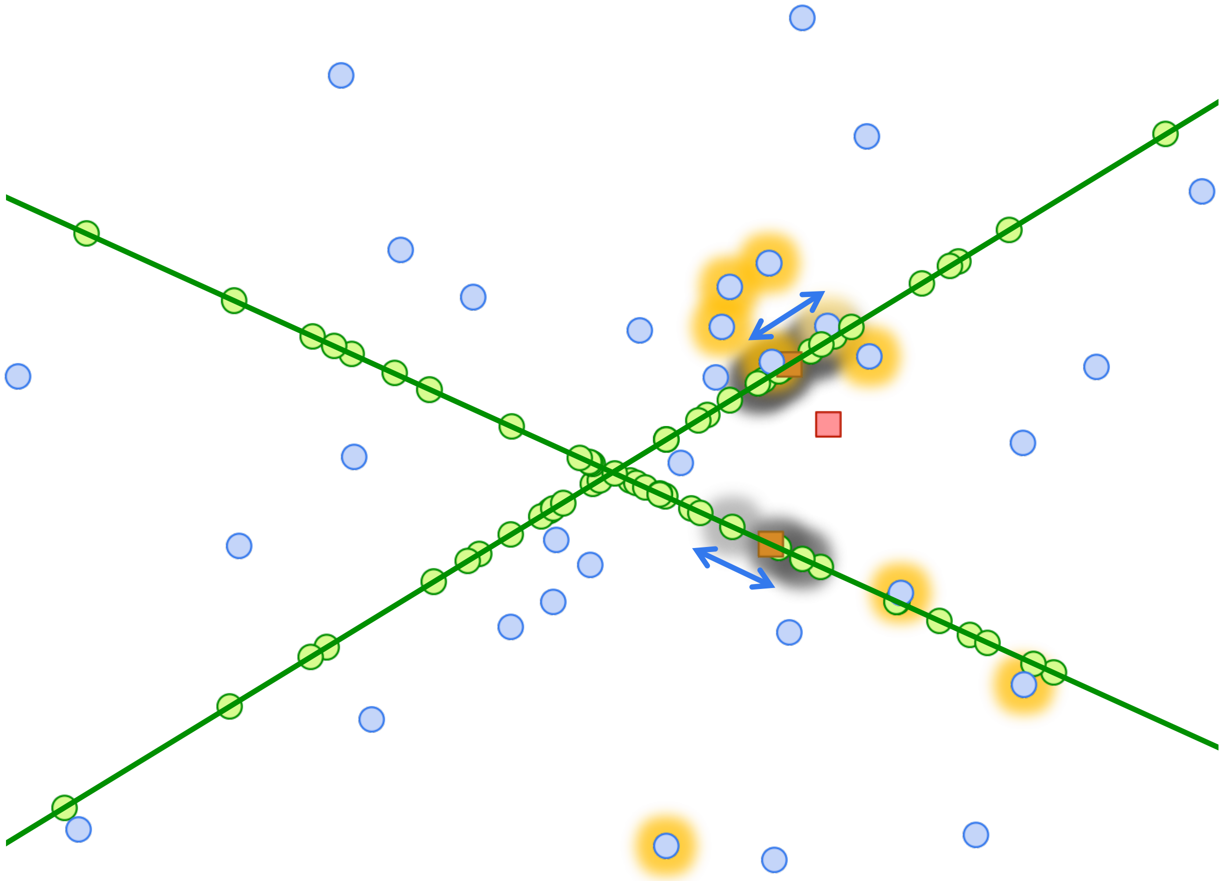
Find the next closest point along the projection direction that has just been processed and add it to the frontier.

Prioritized DCI



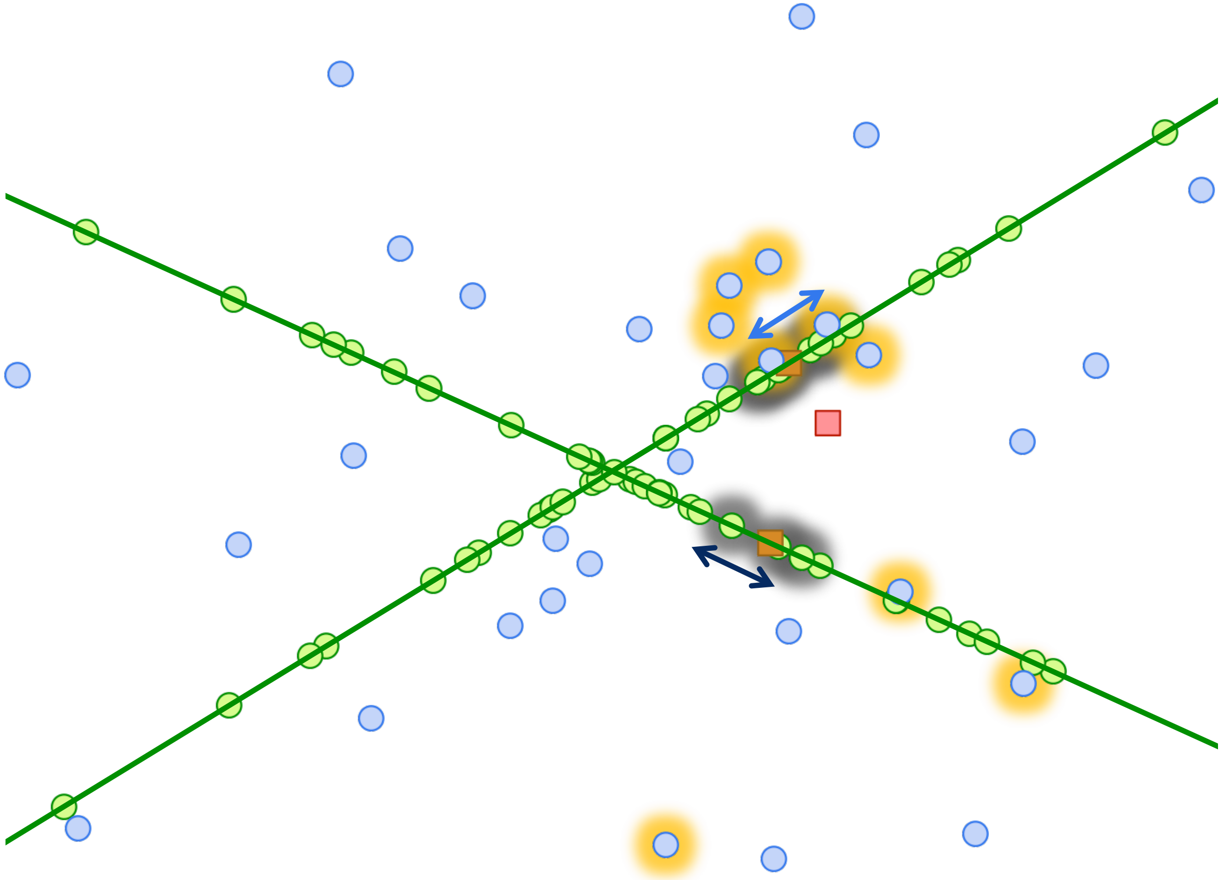
Compare projected distances of points on the frontier and visit the one with the shortest projected distance.

Prioritized DCI



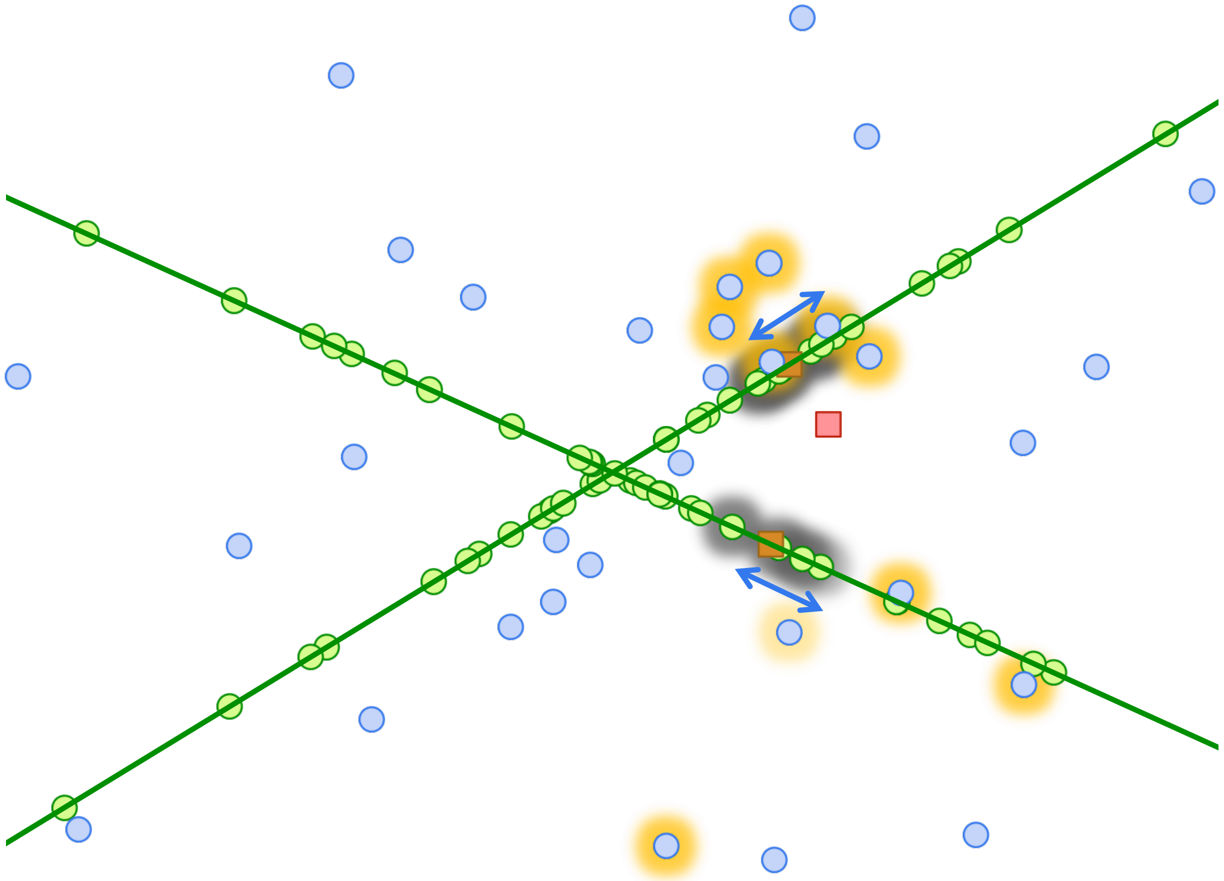
Find the next closest point along the projection direction that has just been processed and add it to the frontier.

Prioritized DCI



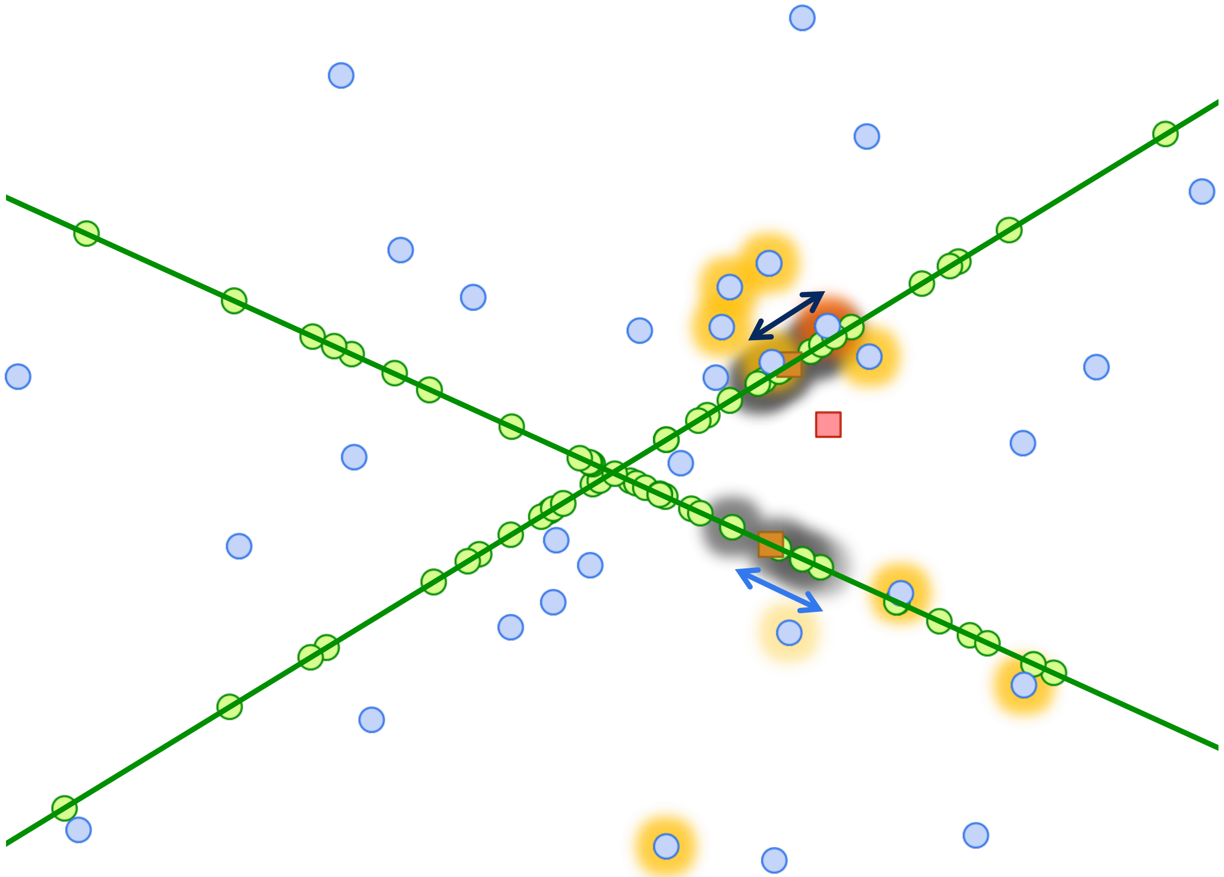
Compare projected distances of points on the frontier and visit the one with the shortest projected distance.

Prioritized DCI



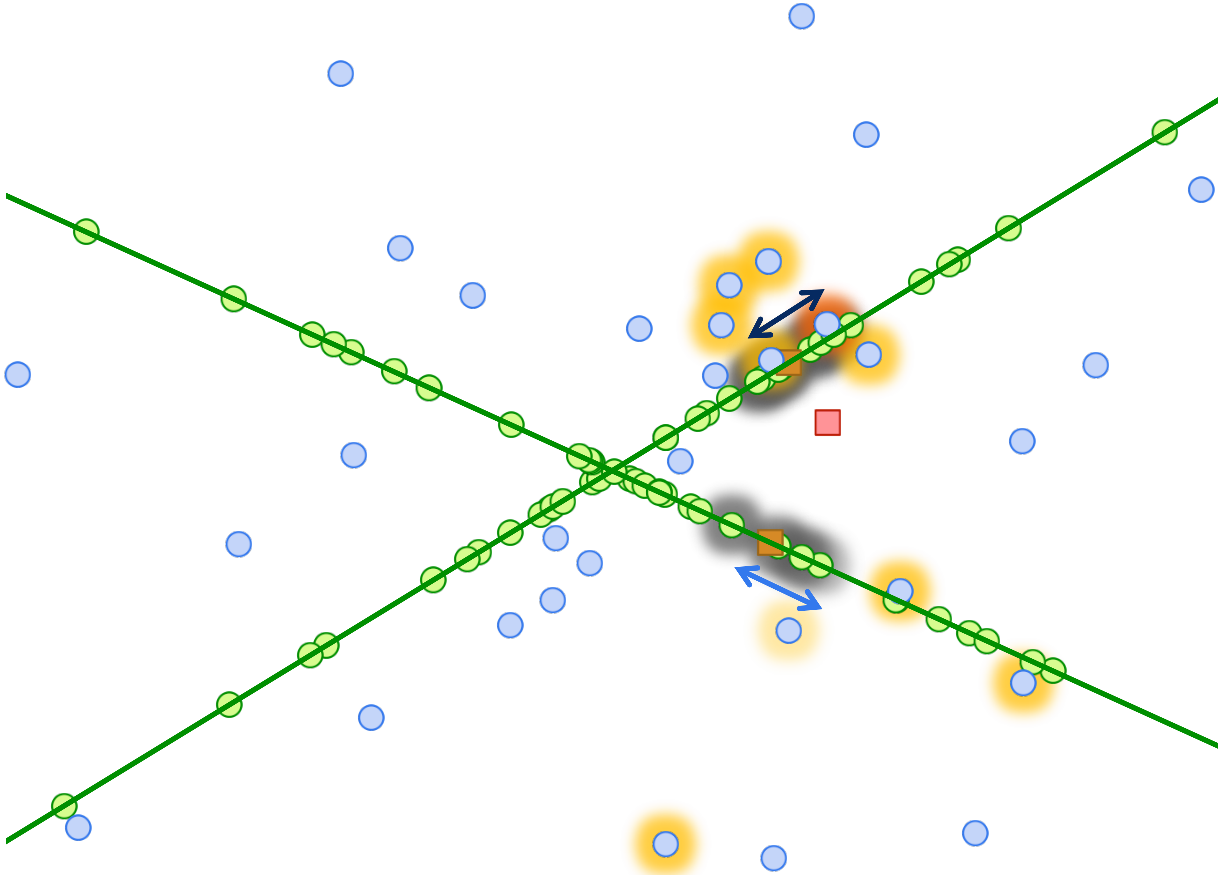
Find the next closest point along the projection direction that has just been processed and add it to the frontier.

Prioritized DCI



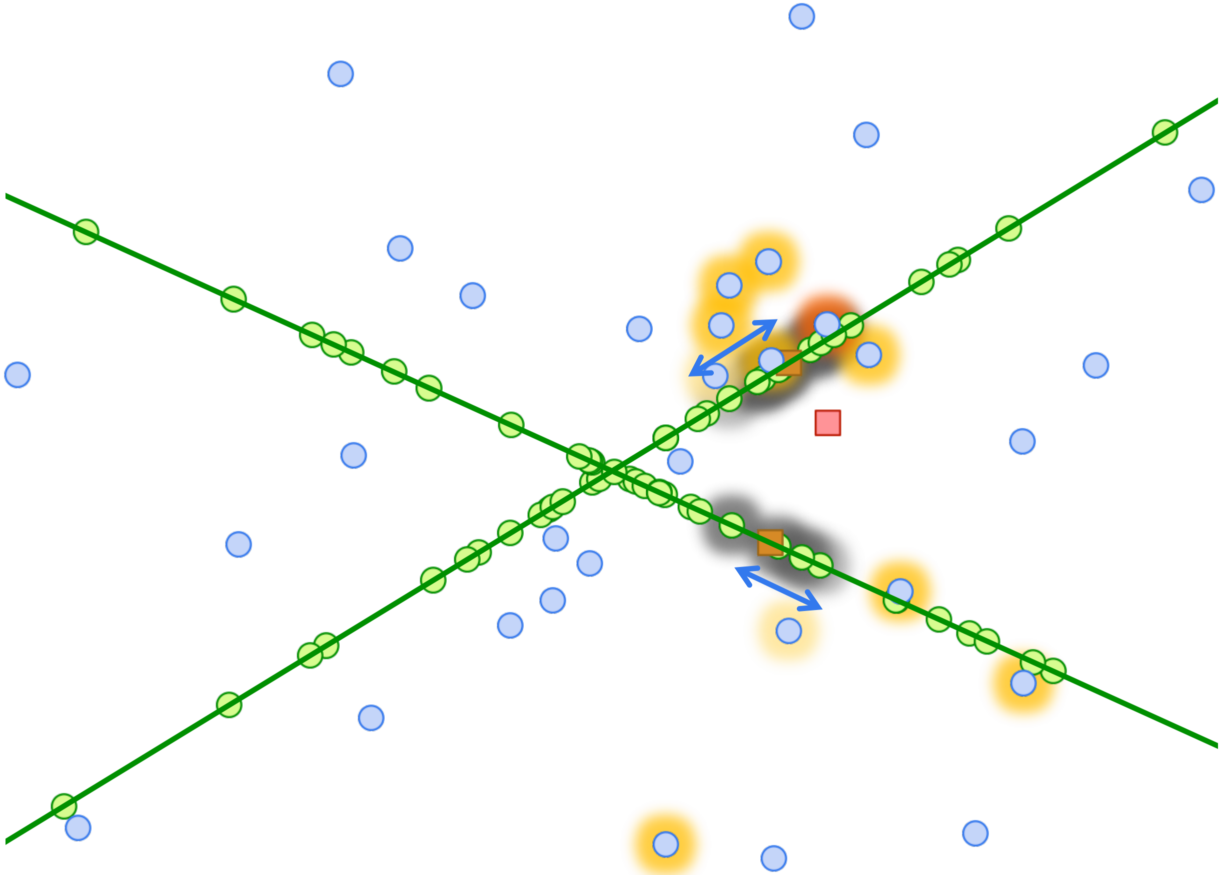
Compare projected distances of points on the frontier and visit the one with the shortest projected distance.

Prioritized DCI



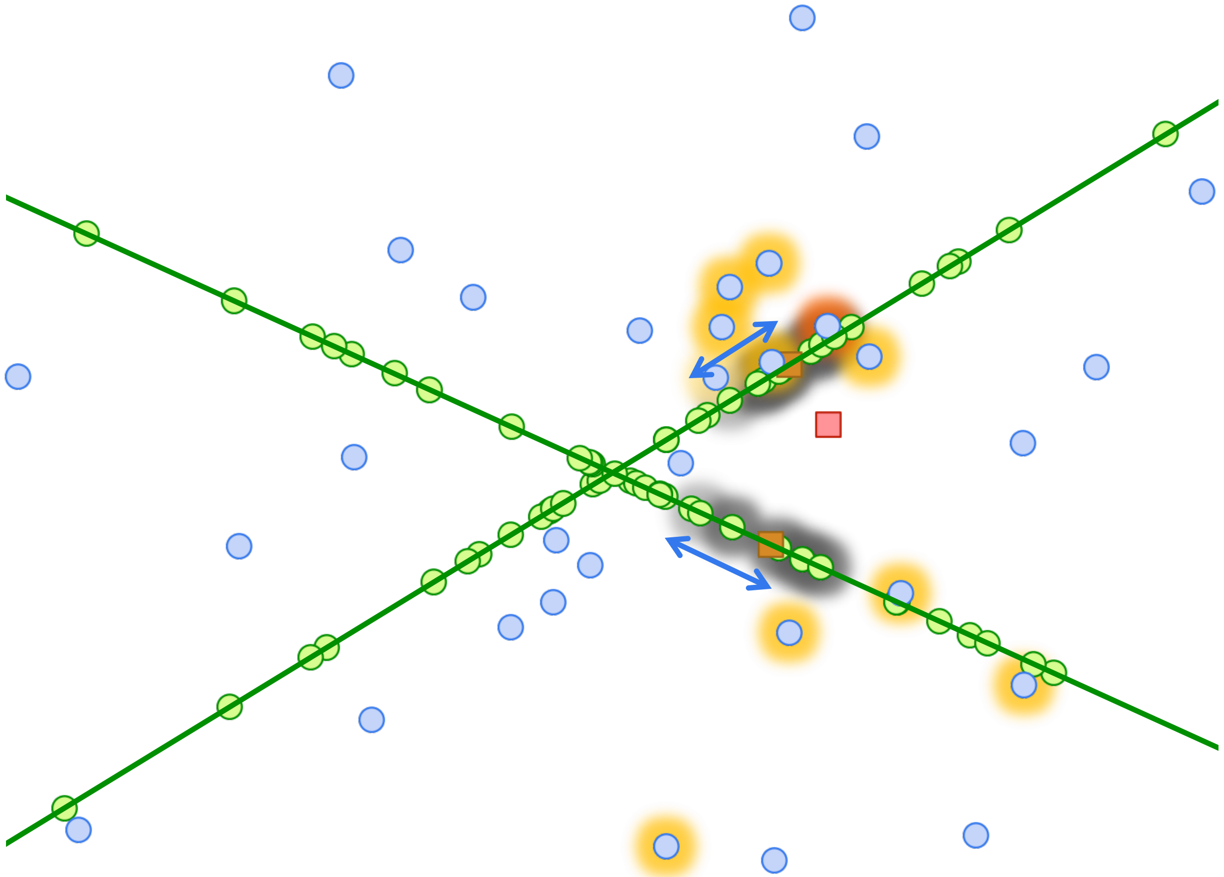
There is now a point that has been visited along all projection directions. We add it to the candidate set.

Prioritized DCI



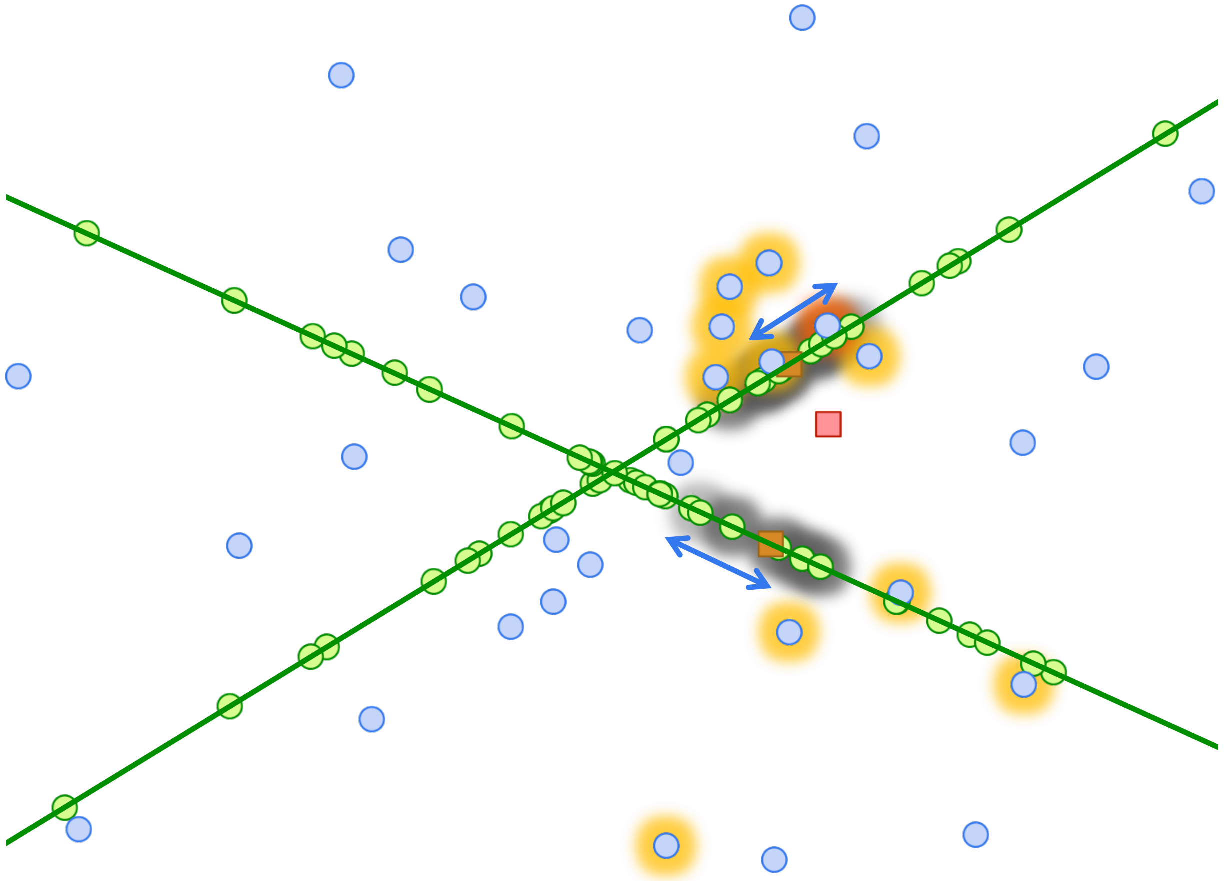
Visit the next point.

Prioritized DCI



Visit the next point.

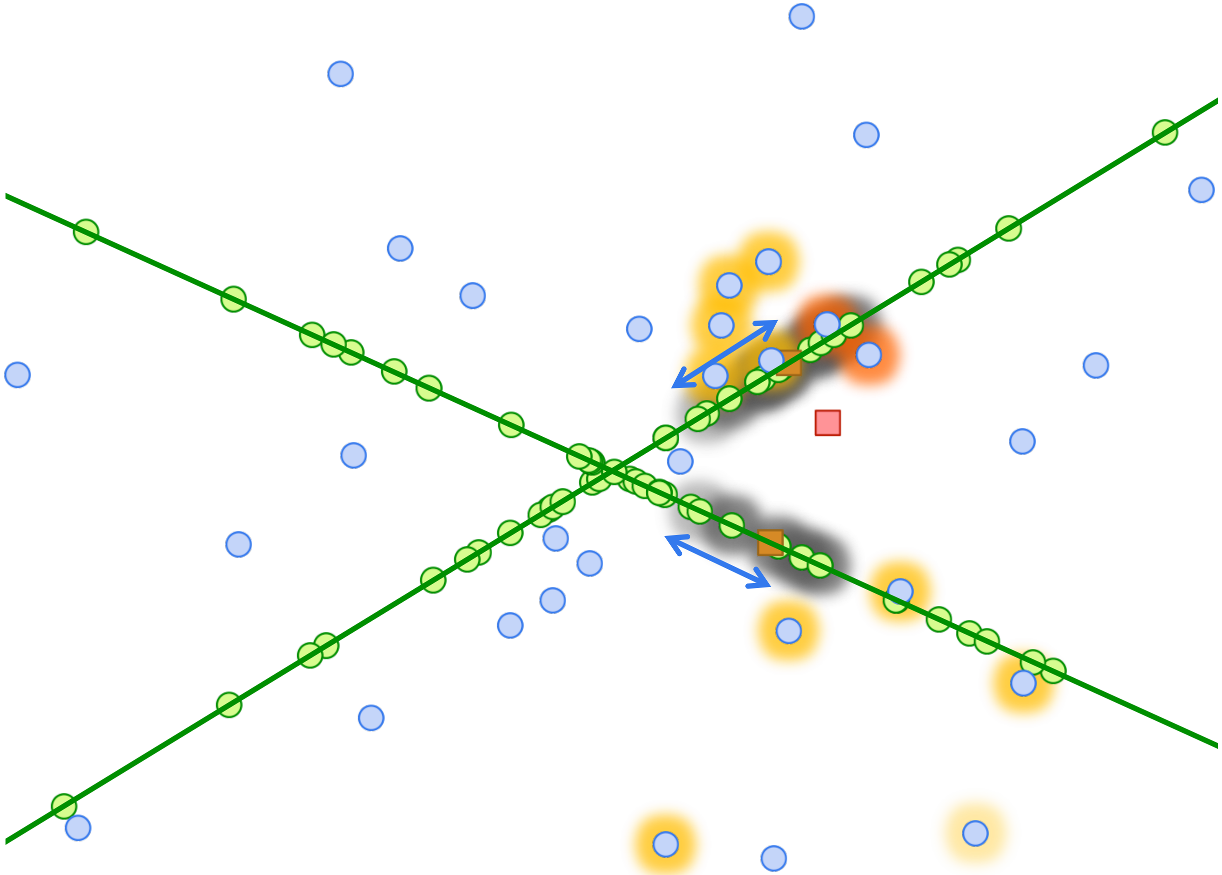
Prioritized DCI



Visit the next point.

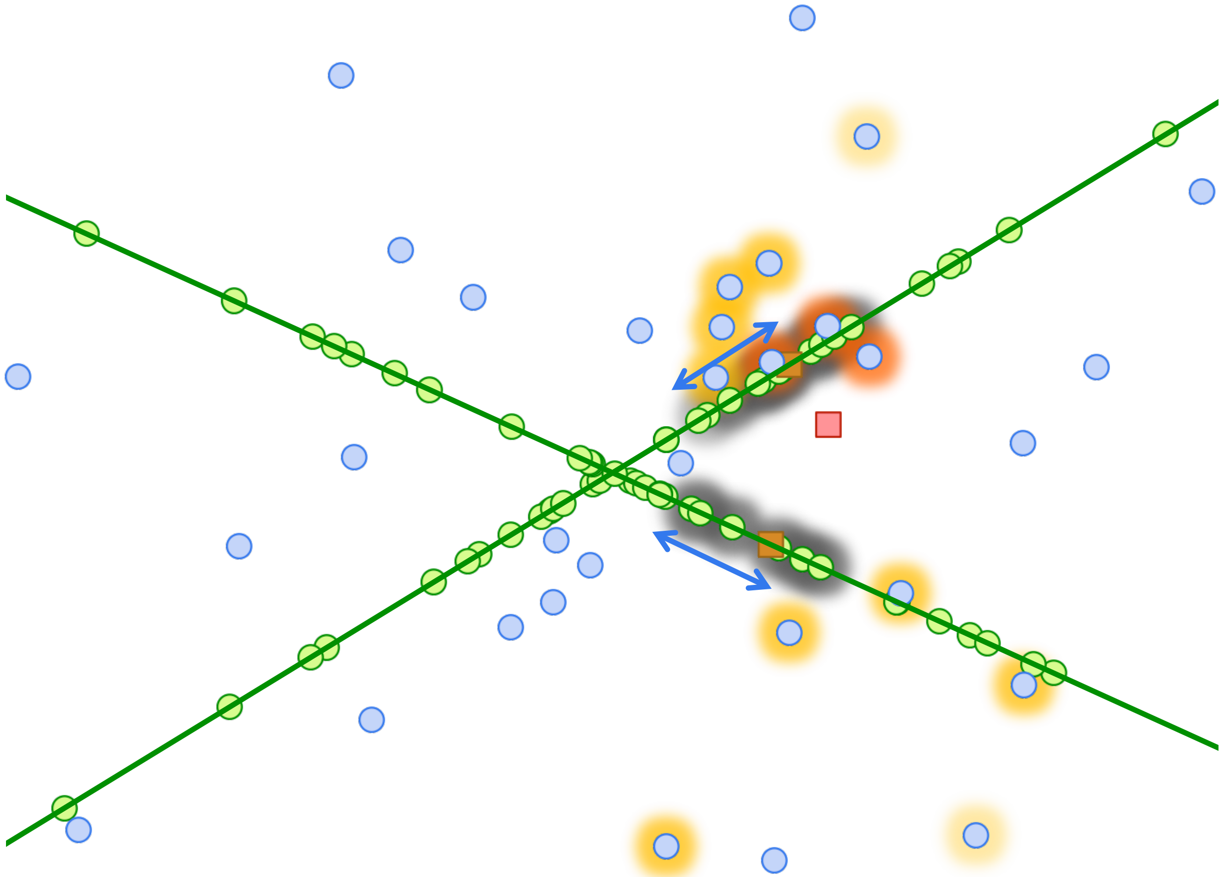
Berkeley UNIVERSITY OF CALIFORNIA

Prioritized DCI



Visit the next point and add it to the candidate set.

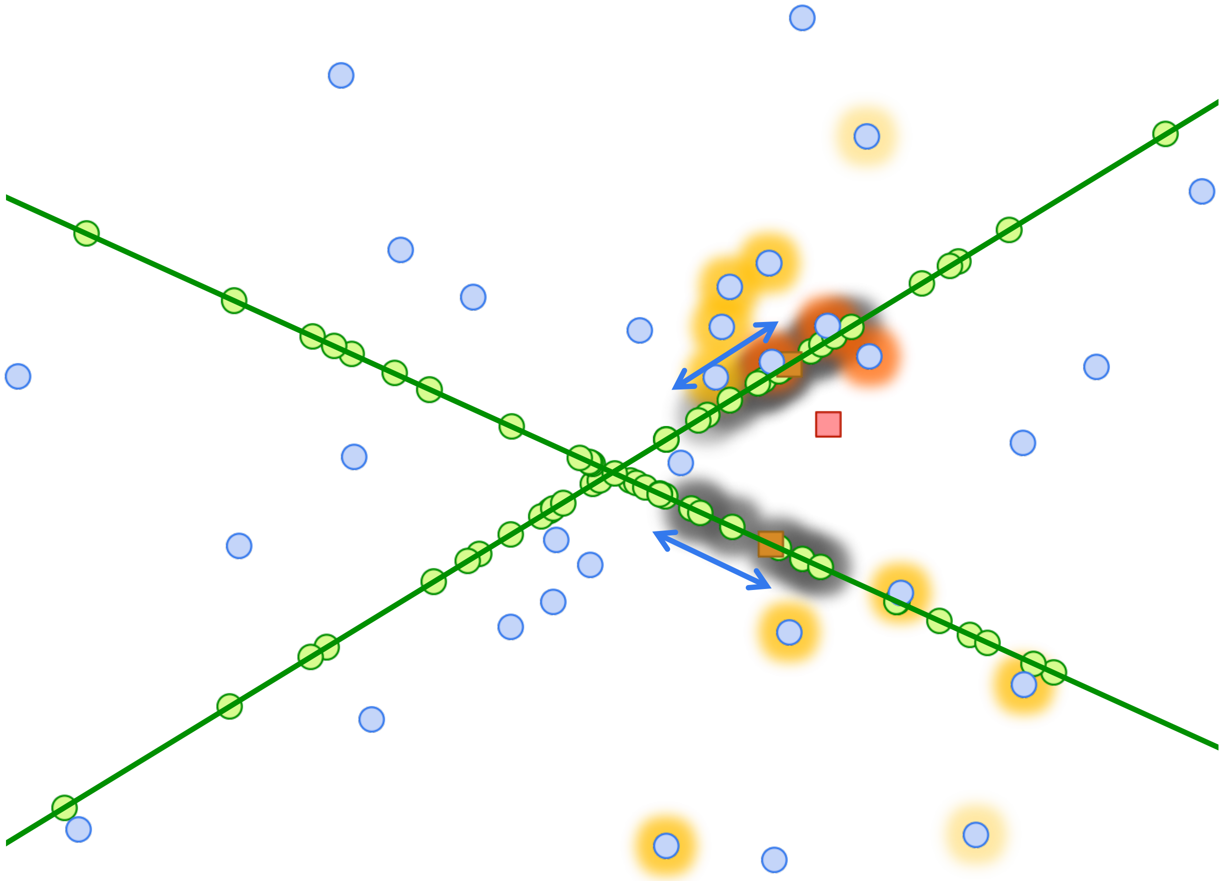
Prioritized DCI



Visit the next point and add it to the candidate set.

Berkeley
UNIVERSITY OF CALIFORNIA

Prioritized DCI



Perform exhaustive search over candidate points and return *k* points that are closest to the query.

Intuition

- Points are added to the candidate set in the order of their maximum projected distance to the query.
- Maximum projected distance is a lower bound on the true distance.
- As the number of projection directions increases, this lower bound approaches the true distance.

$$\max_j \left\{ \left| \langle p^i, u_j \rangle - \langle q, u_j \rangle \right| \right\} = \max_j \left\{ \left| \langle p^i - q, u_j \rangle \right| \right\} \leq \left\| p^i - q \right\|_2$$

where $\left\| u_j \right\|_2 = 1 \quad \forall j$

Lemma

- We derived the following lemma, which may be of independent interest:
- For any set of events $E_1, \ldots, E_n$, the probability that at least m of them occur is at most:

$$\frac{1}{m} \sum_{i=1}^{n} \Pr\left(E_i\right)$$

- When $m = 1$, this reduces to the union bound.

(Proof is in the "Fast k-Nearest Neighbour Search via Prioritized DCI" paper, though two students, Eric Xia and Zipeng Qin, later came up with simpler proofs, one using measure theory, and one using Markov's inequality.)

Complexity

- Construction Time: $O(m(dn + n\log n))$
- Query Time: $O(dk\max(\log(n/k), (n/k)^{1-m/d'})+$ $mk\log m(\max(\log(n/k), (n/k)^{1-1/d'})))$
- Insertion Time: $O(m(d + \log n))$
- Deletion Time: $O(m\log n)$
- Space: $O(mn)$

where $m \geq 1$ is the number of projection directions chosen by the user.

Complexity

- Construction Time: $O(m(dn + n\log n))$
- Query Time: $O(\boxed{d}k\max(\log(n/k), \boxed{(n/k)^{1-m/d'}})+$
  $mk\log m(\max(\log(n/k), \boxed{(n/k)^{1-1/d'}})))$
- Insertion $\log n))$
- Deletion $)$
- Space: $O(mn)$

where $m \geq 1$ is the number of projection directions chosen by the user.

Linear dependence on ambient dimensionality

Sublinear dependence on intrinsic dimensionality

Berkeley UNIVERSITY OF CALIFORNIA

Complexity

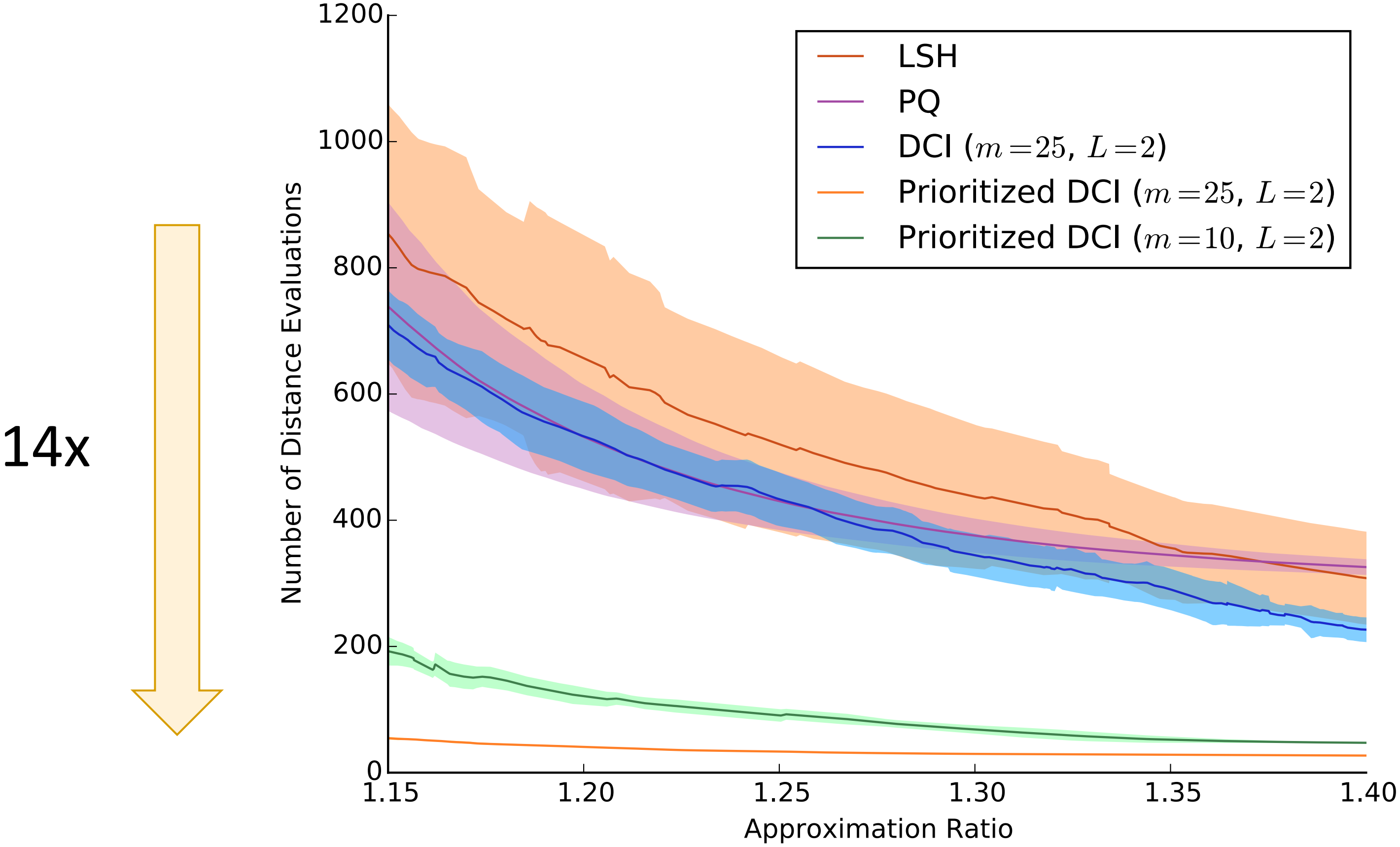
- Construction Time: $O(m(dn + n\log n))$
- Query Time: $O(dk\max(\log(n/k), \boxed{(n/k)^{1-m/d'}}) + mk\log m(\max(\log(n/k), (n/k)^{1-1/d'})))$
- Insertion Time: $O(m(d + \log n))$
- Deletion Time: $O(m\log n)$
- Space: $O(mn)$

A linear increase in intrinsic dimensionality can be mostly counteracted with a linear increase in the number of projection directions.

where $m \geq 1$ is the number of projection directions chosen by the user.

Experiments

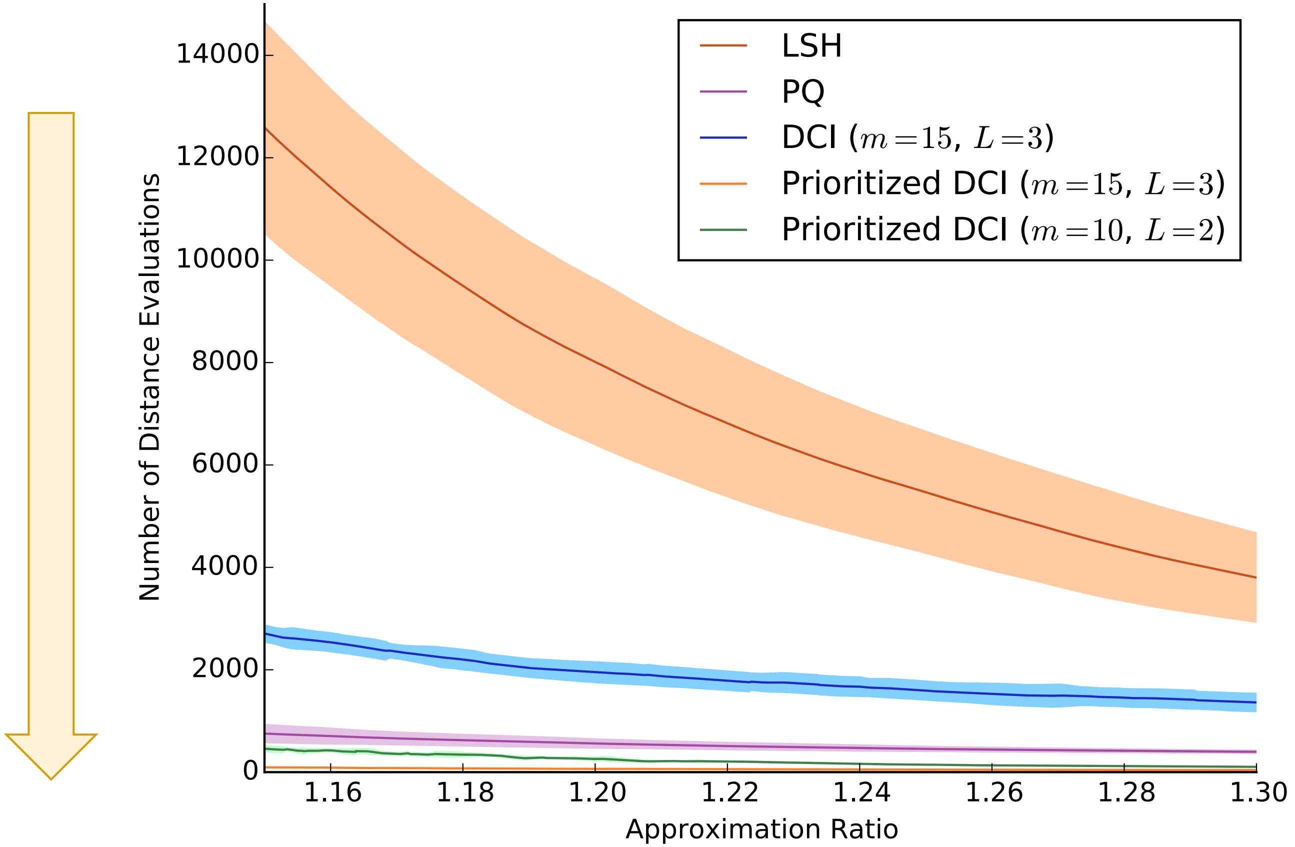
Query Time on CIFAR-100



$$\text{approximation ratio} = \frac{\text{distance to retrieved nearest neighbours}}{\text{distance to true nearest neighbours}}$$

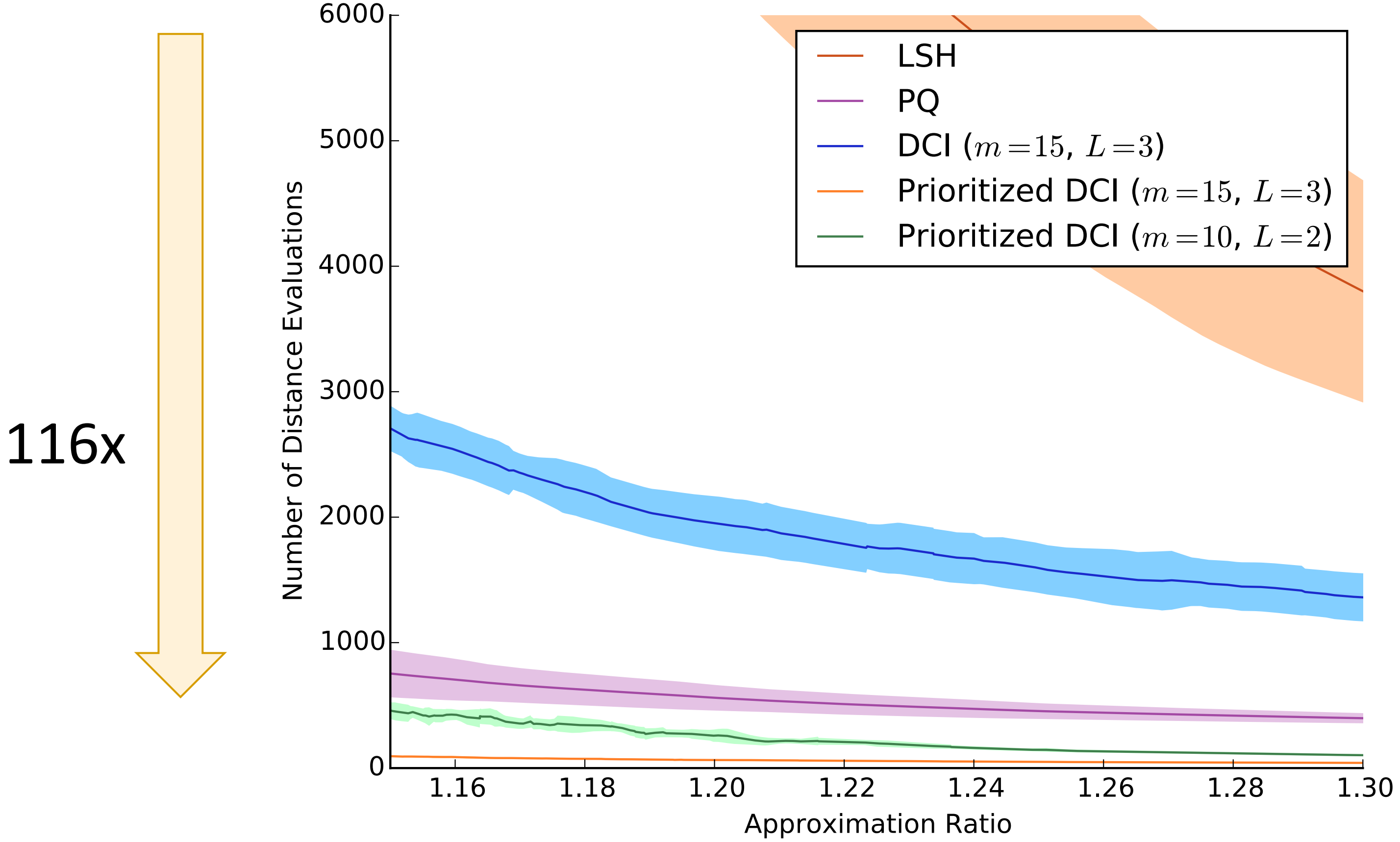
Query Time on MNIST

116x



$$\text{approximation ratio} = \frac{\text{distance to retrieved nearest neighbours}}{\text{distance to true nearest neighbours}}$$

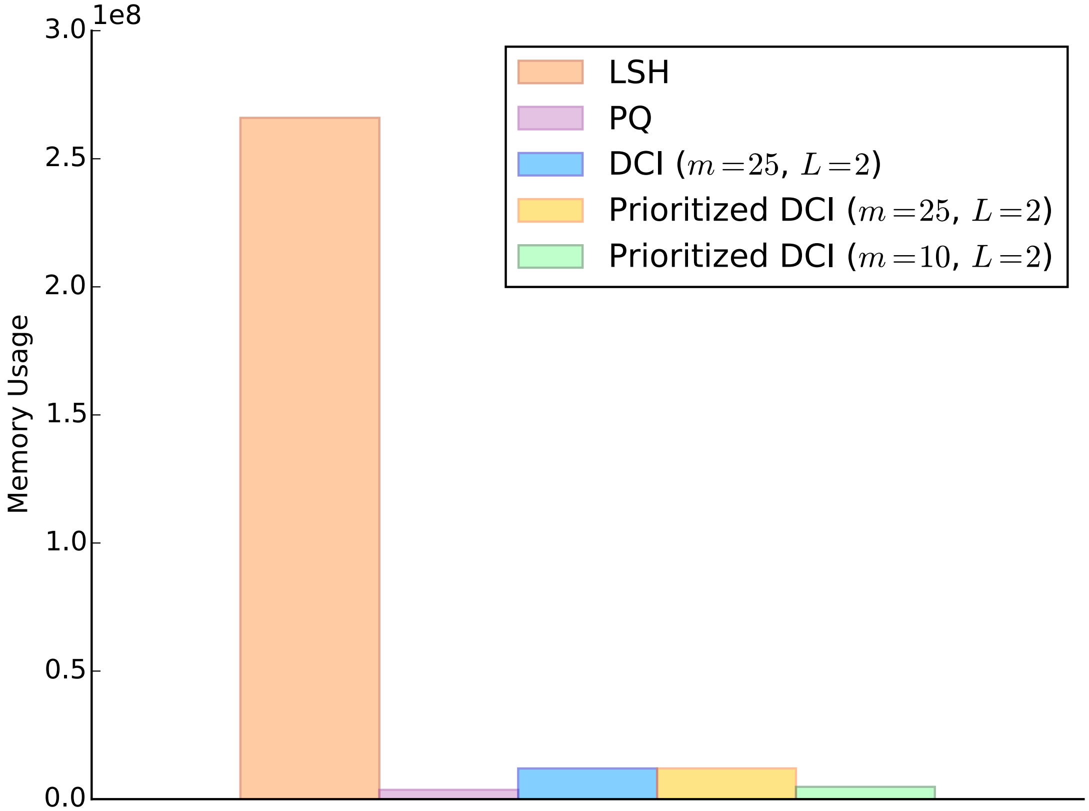
Query Time on MNIST



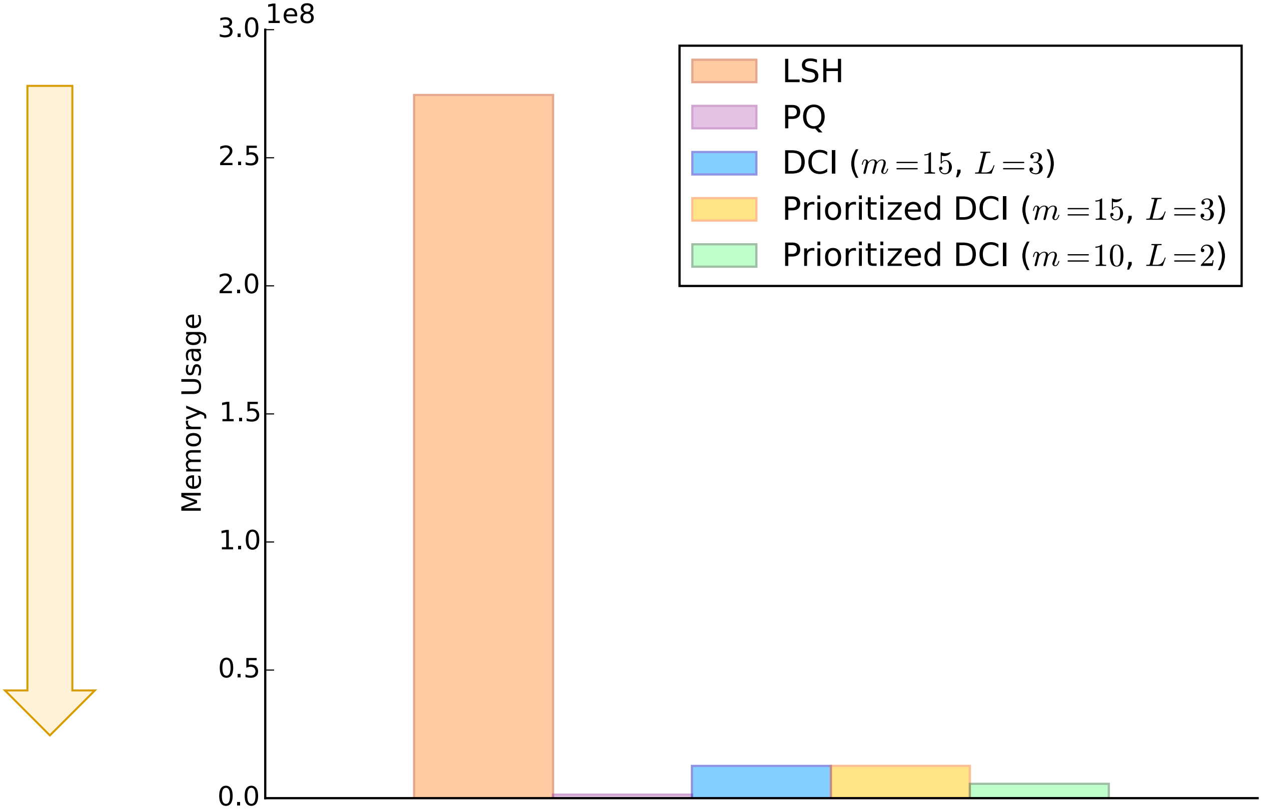
$$\text{approximation ratio} = \frac{\text{distance to retrieved nearest neighbours}}{\text{distance to true nearest neighbours}}$$

Space Efficiency on CIFAR-100

22x



Space Efficiency on MNIST



Berkeley UNIVERSITY OF CALIFORNIA

For More Details…

Fast *k*-Nearest Neighbour Search via Dynamic Continuous Indexing

Ke Li, Jitendra Malik

ICML, 2016

Fast *k*-Nearest Neighbour Search via Prioritized DCI

Ke Li, Jitendra Malik

ICML, 2017