Nearest Neighbor / Similarity Search

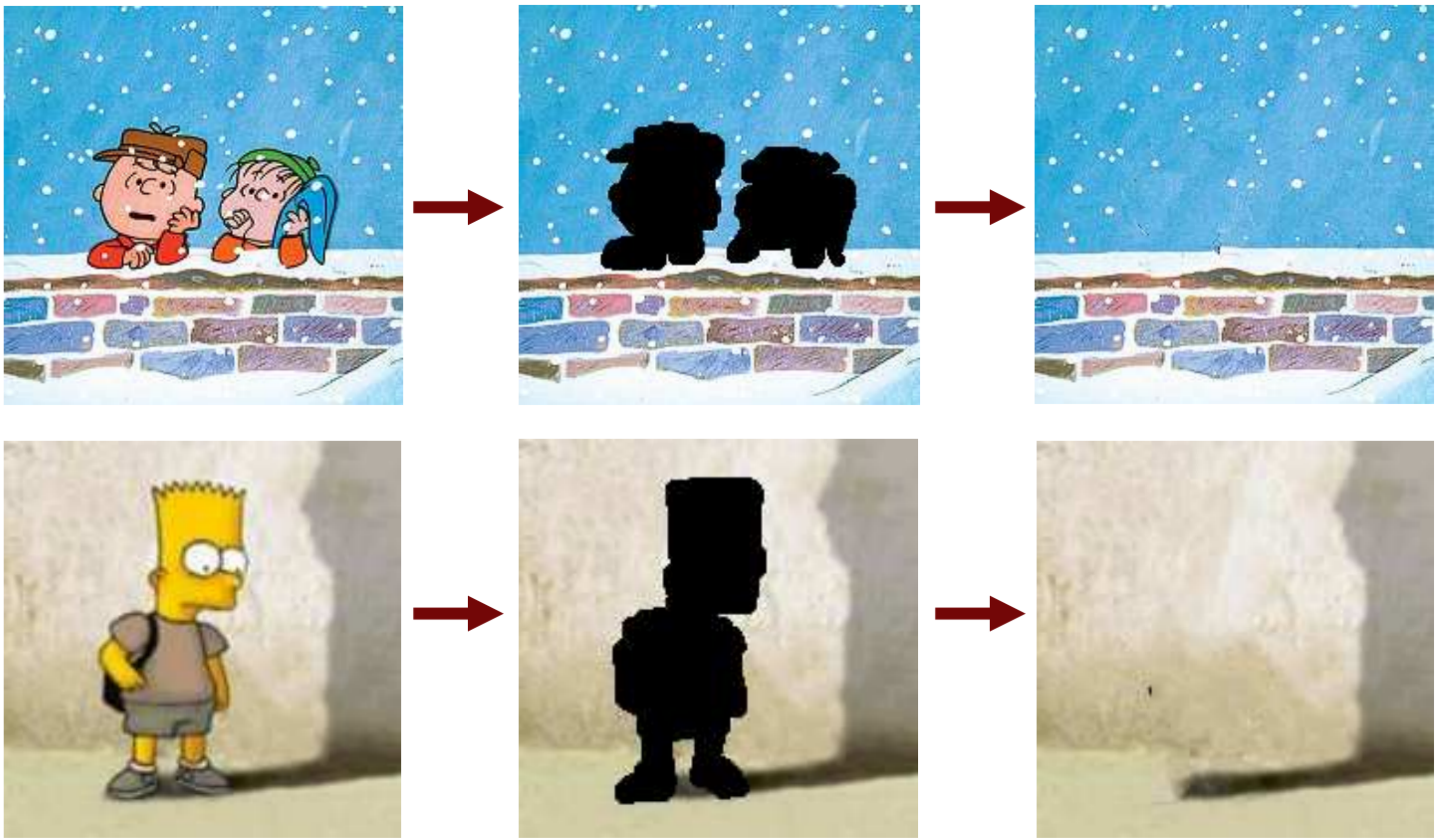
CS 584: Big Data Analytics

Material adapted from
Piotr Indyk
(https://people.csail.mit.edu/indyk/helsinki-1.pdf) & Andrew Moore

Nearest Neighbor (NN)

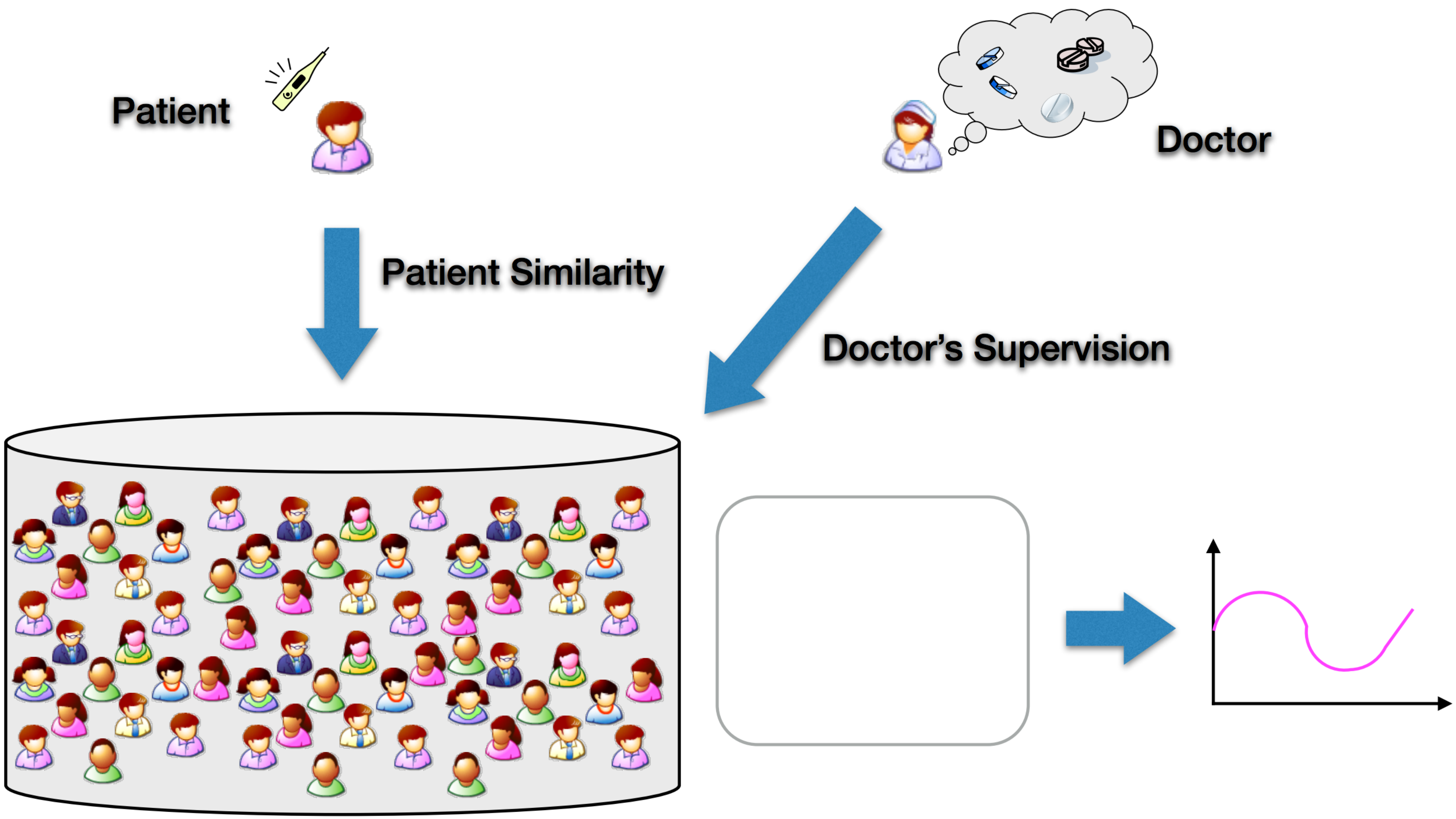
- Problem Statement: Given a set of points or samples {p1, …, pN}, and a new point q, find the data point nearest to q
- (AKA) Closest-point problem or post office problem
- Many problems can be expressed as finding “similar” sets
  - Data compression
  - Information retrieval
  - Pattern recognition
  - …

Applications: Image Completion



https://graphics.stanford.edu/courses/cs468-06-fall/Slide/aneesh-michael.pdf

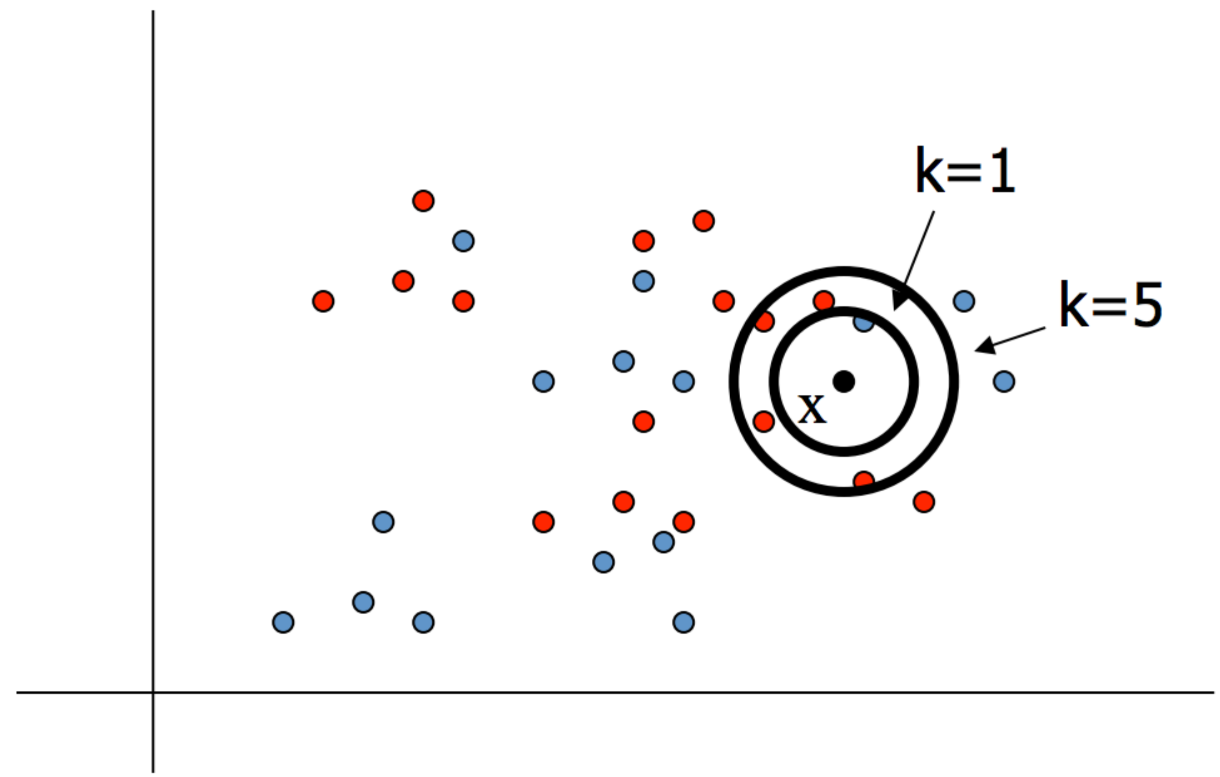
Applications: Patient Prognosis



http://engr.uconn.edu/~fwang/tutorials/CIKM14_Tutorial.pdf

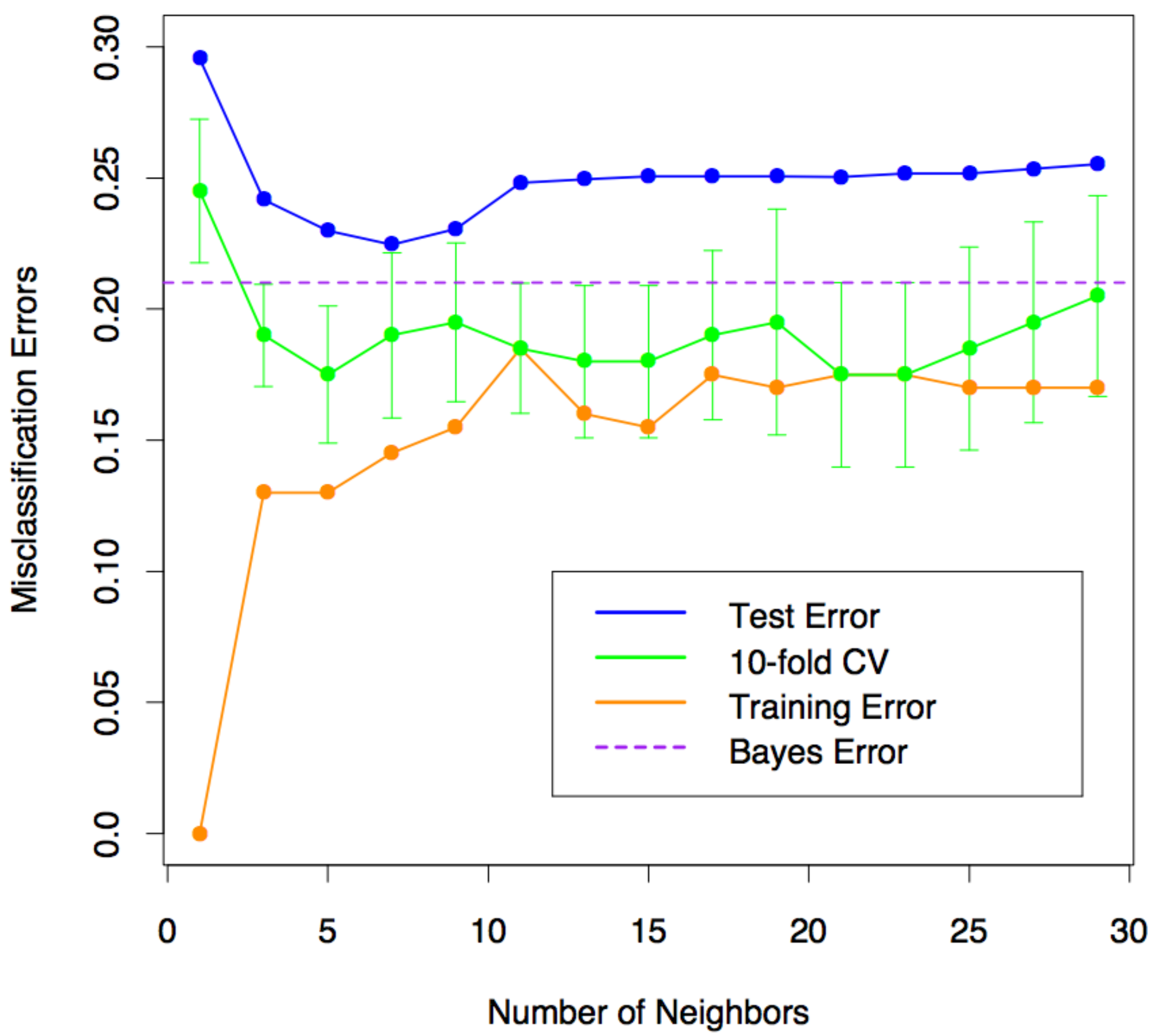
k-NN Algorithm

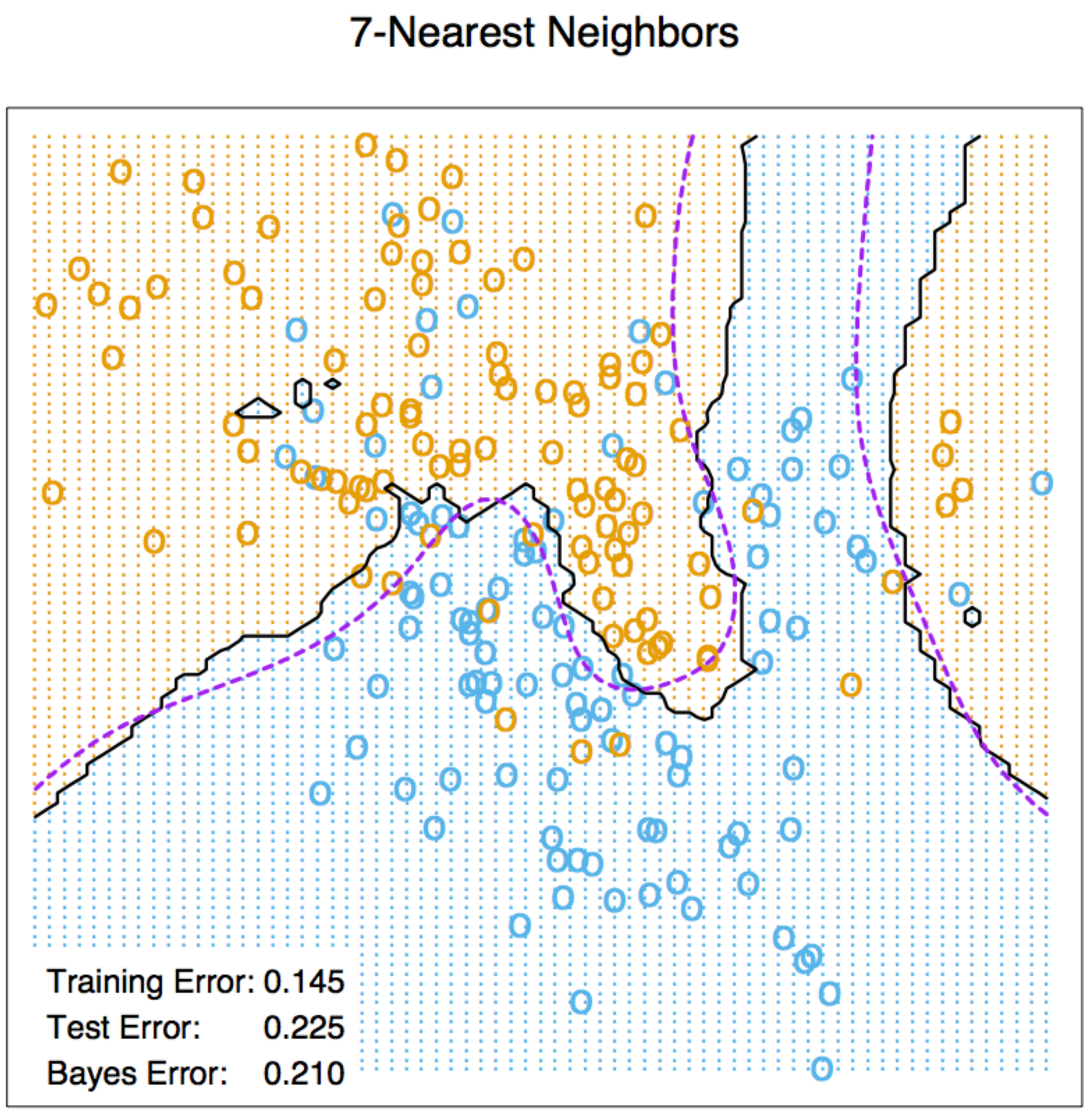
- Examine the k-“closest” training data points to new point x
    - Closest depends on the distance metric used
- Assign the object the most frequently occurring class
- Example http://www.theparticle.com/applets/ml/nearest_neighbor



http://cs.nyu.edu/~dsontag/courses/ml13/slides/lecture11.pdf

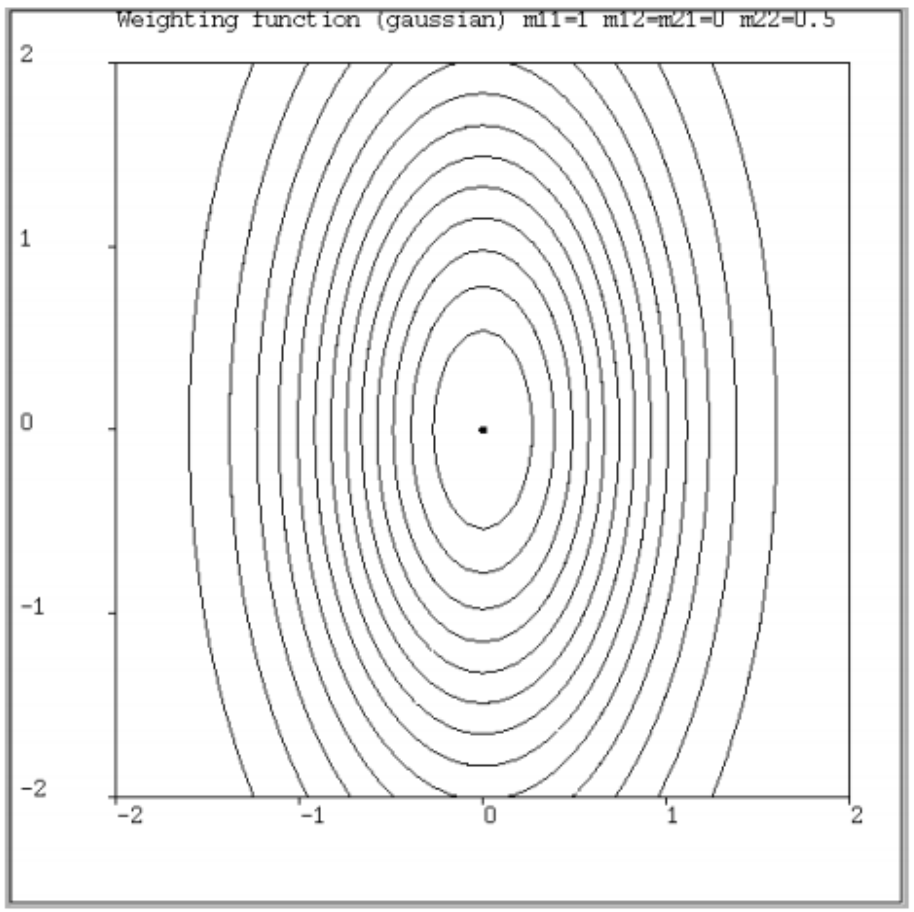
Example: k-NN Results



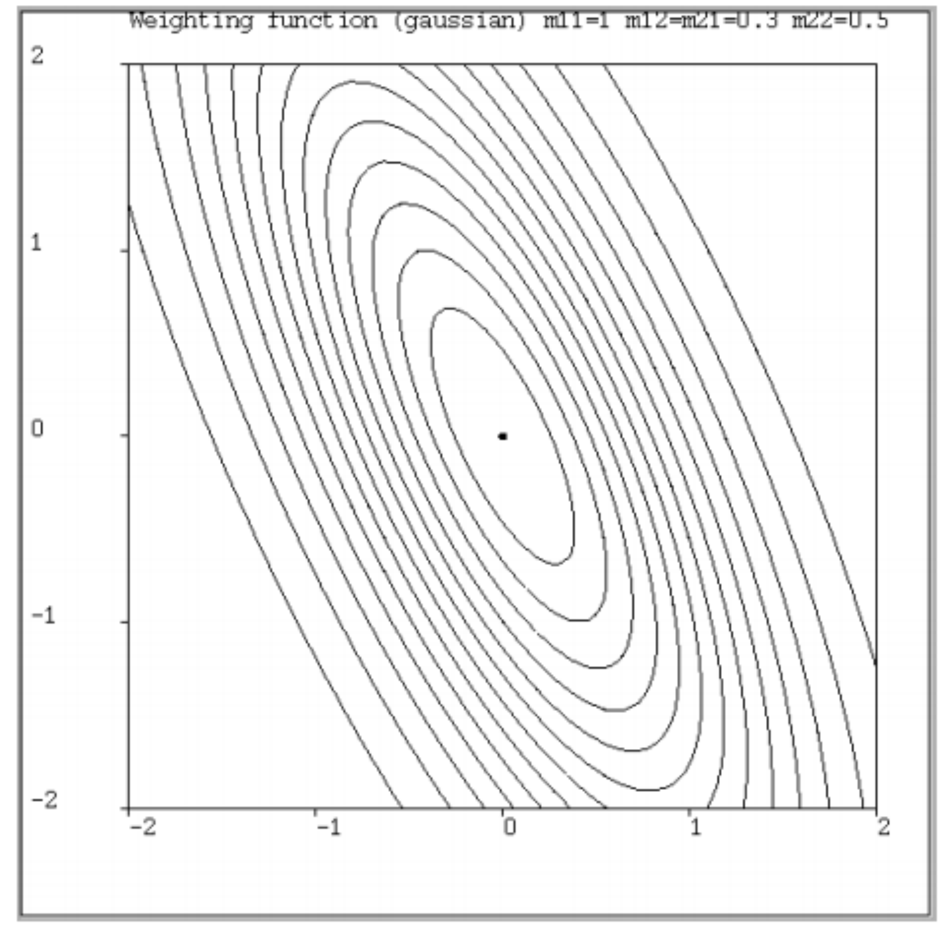


Figures from Chapter 13 of ESL by Hastie & Tibshirani

Notable Distance Metrics



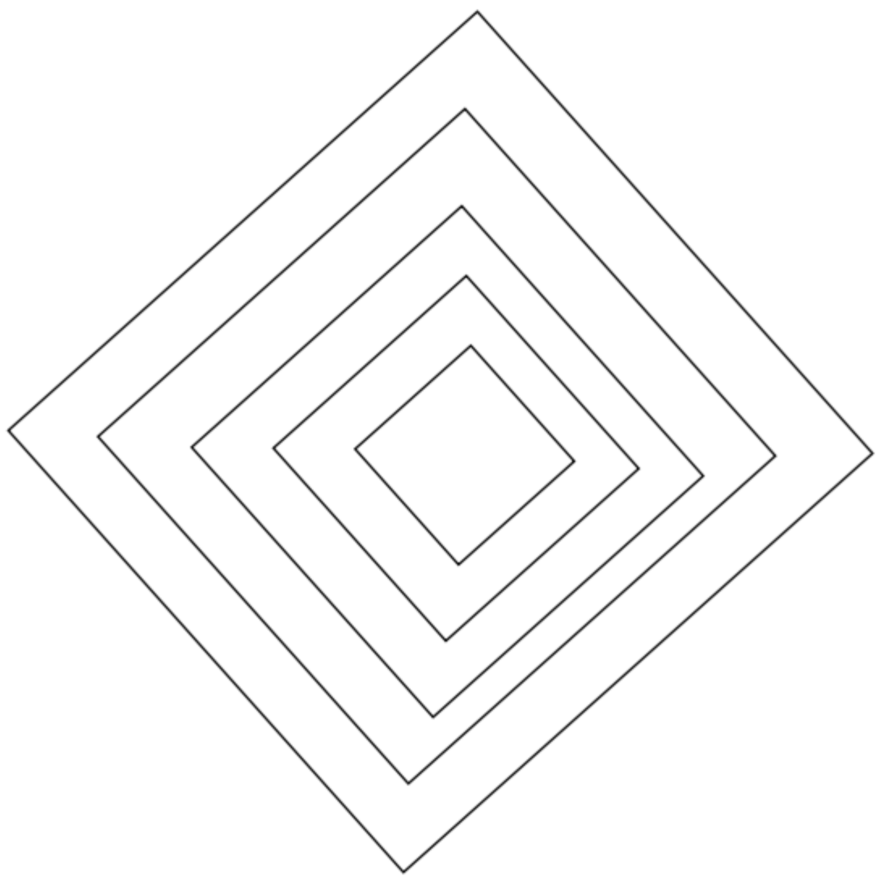
Scaled Euclidean
(diagonal covariance)



Mahalanobis
(full covariance)

$$D(x,y) = \sqrt{(x-y)^\top S^{-1}(x-y)}$$

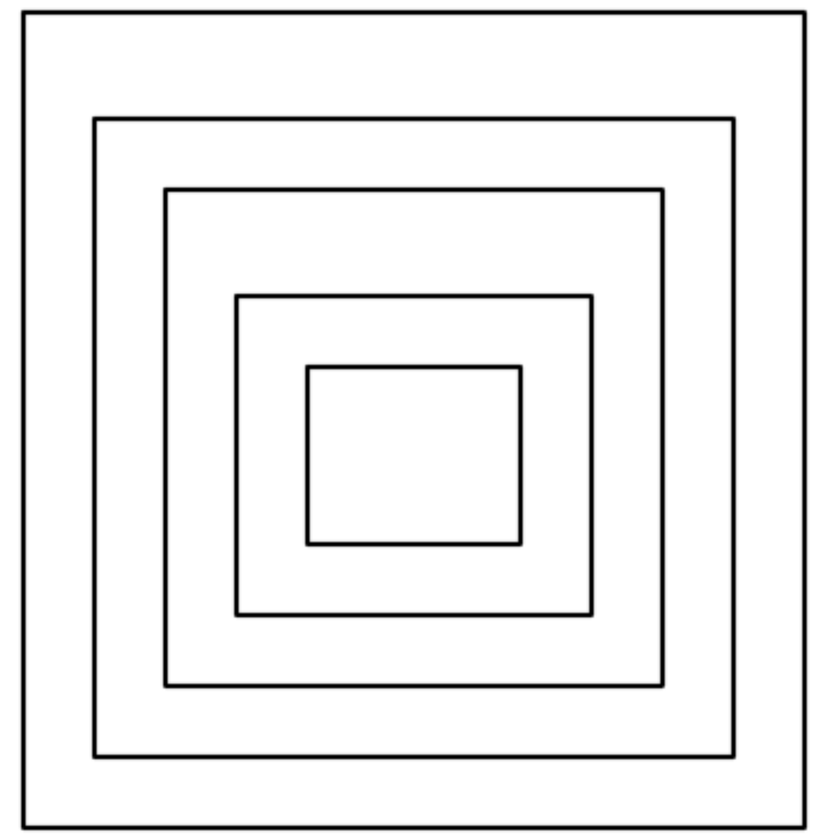
Notable Distance Metrics (2)



Manhattan or taxicab

L_1 norm

$$D(x, y) = \sum_{i=1}^{d} |x_i - y_i|$$



Maximum norm

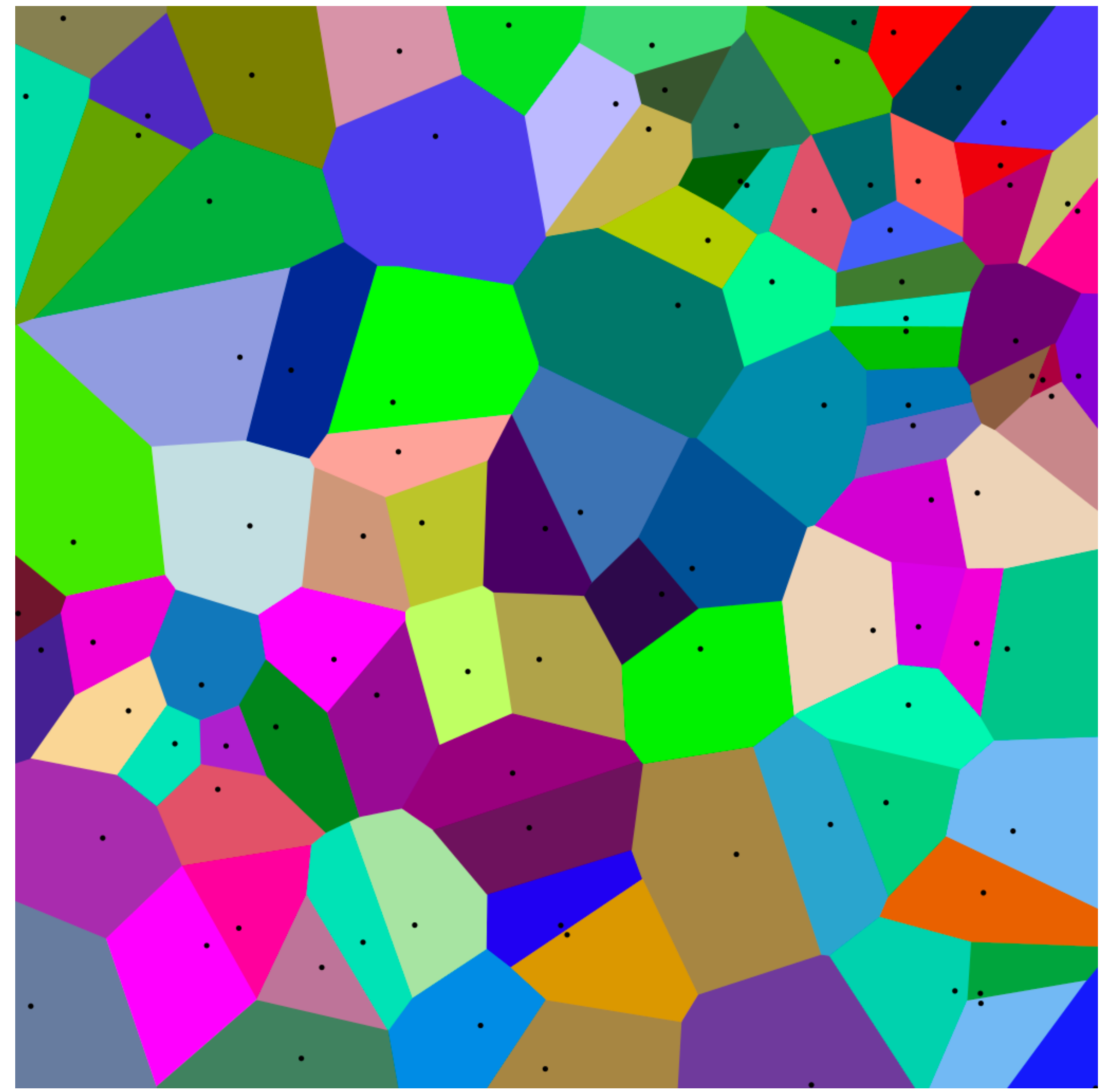
$$D(x, y) = \max_{i=1}^{d} |x_i - y_i|$$

NN (and kNN) Advantages

- Instance-based learning or lazy learning so building the model is very cheap
- Easy to understand and easy to implement
- Can be quite accurate (dependent on distance metric)
- Well-suited of multi-modal classes as well as a variety of applications
- One of the most popular algorithms
(ranked 8th by KD Nuggets)

NN: When d = 2 (Euclidean Distance)

- Compute Voronoi diagram from the set of points
  - Each line segment is equidistant between two points
- Given q, perform point location
- Performance:
  - Space: O(n)
  - Query time: O(log n)



https://en.wikipedia.org/wiki/Voronoi_diagram

NN: When d > 2 (Euclidean Distance)

- Generalization of Voronoi to higher dimension achieves ~$O(n^{d/2})$ space
    - Impractical on a dataset of even just a million points for d greater than or equal to 3
- Query can be performed via linear scan: $O(dn)$ time
- Tree-based data structures with pre-processing: kd-trees

kd-Trees

kd-Trees [Bentley '75]

- Not the most efficient solution in theory but used in practice
- Name originally meant 3d-trees, 4d-trees, …, where k was the number of dimensions
- Idea: Each level of the tree compares against 1 dimension

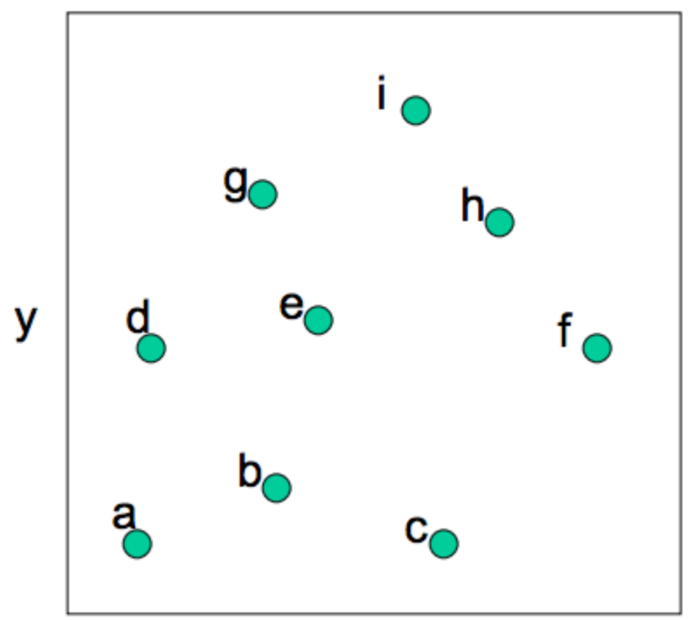
kd-Trees

- Binary tree (data structure) for storing finite set of points from a k-dimensional space
- Applications
  - Nearest neighbor search
  - Range queries
  - Fast look-up
- Guaranteed $\log_2 n$ depth where n is the number of points in the set

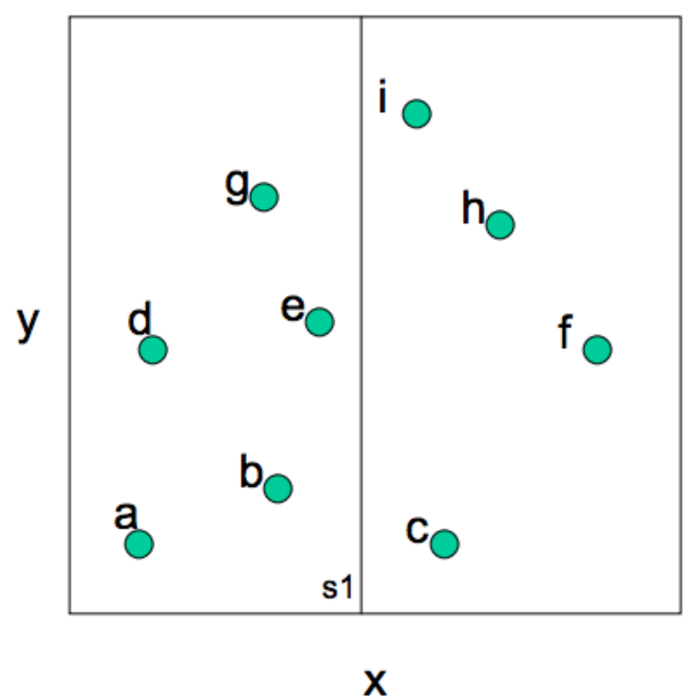
kd-Tree Construction

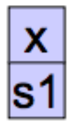
- If just one point, form a leaf with that point
- Otherwise, choose an axis and divide the points in half via the median of the axis
- Recursively construct kd-trees for the two sets of points
- Binary tree with:
  - Size: O(n)
  - Depth: O(log n)
  - Construction: O(n log n)

Example: kd-Tree Construction



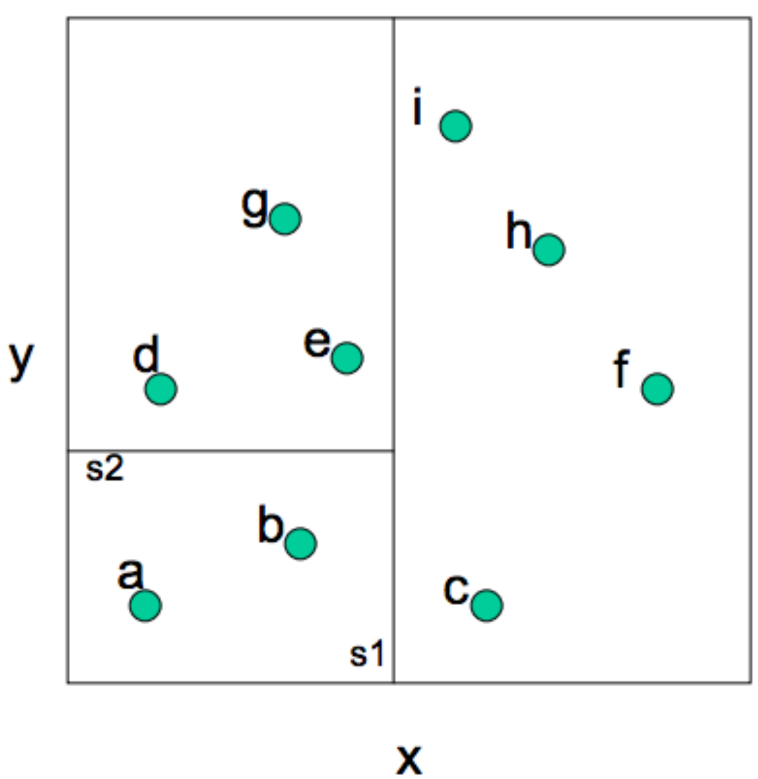
Select an axis (x) and choose the median line

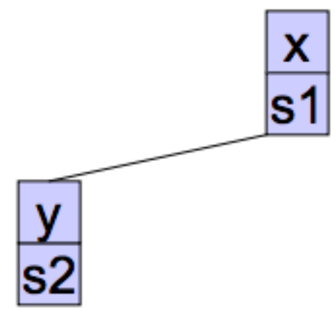


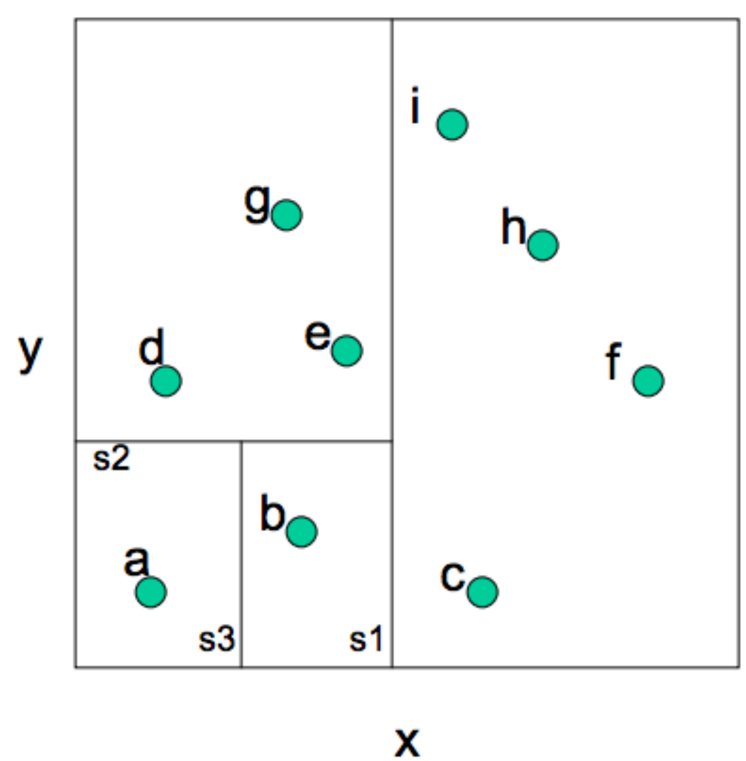


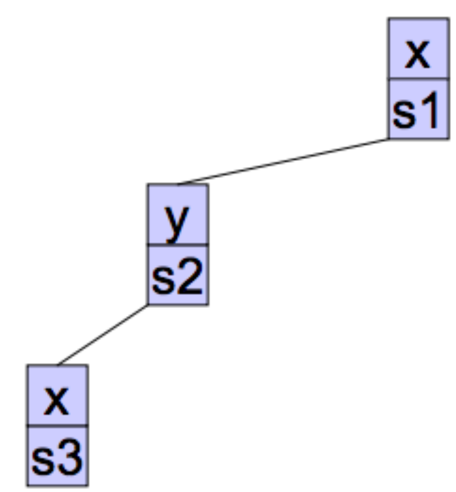
https://courses.cs.washington.edu/courses/cse373/02au/lectures/lecture22l.pdf

Example: kd-Tree Construction (1)

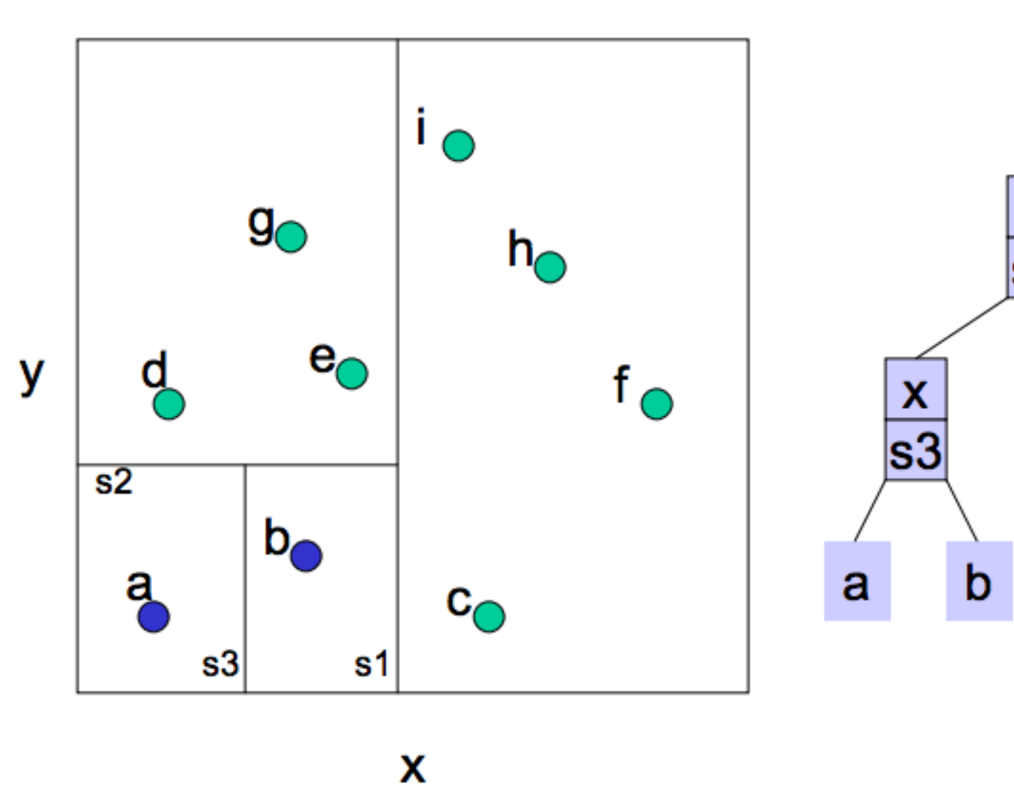


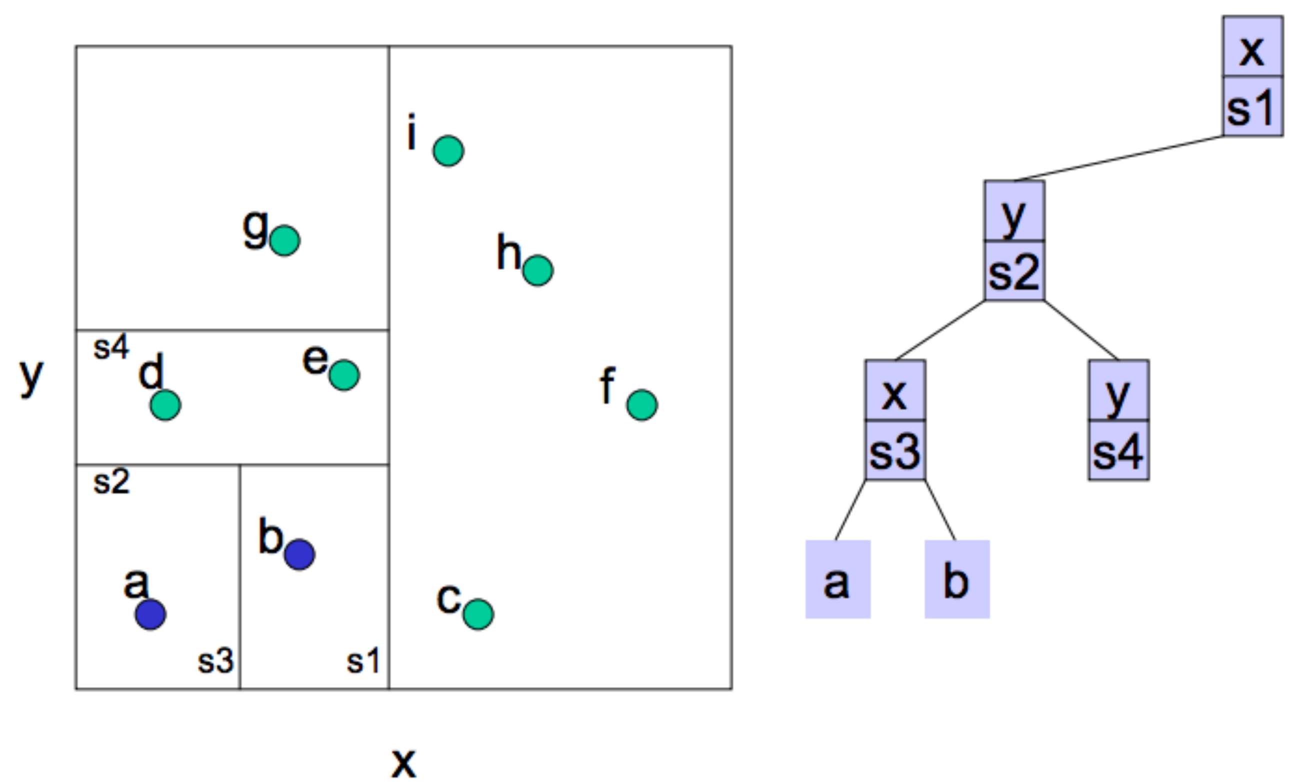




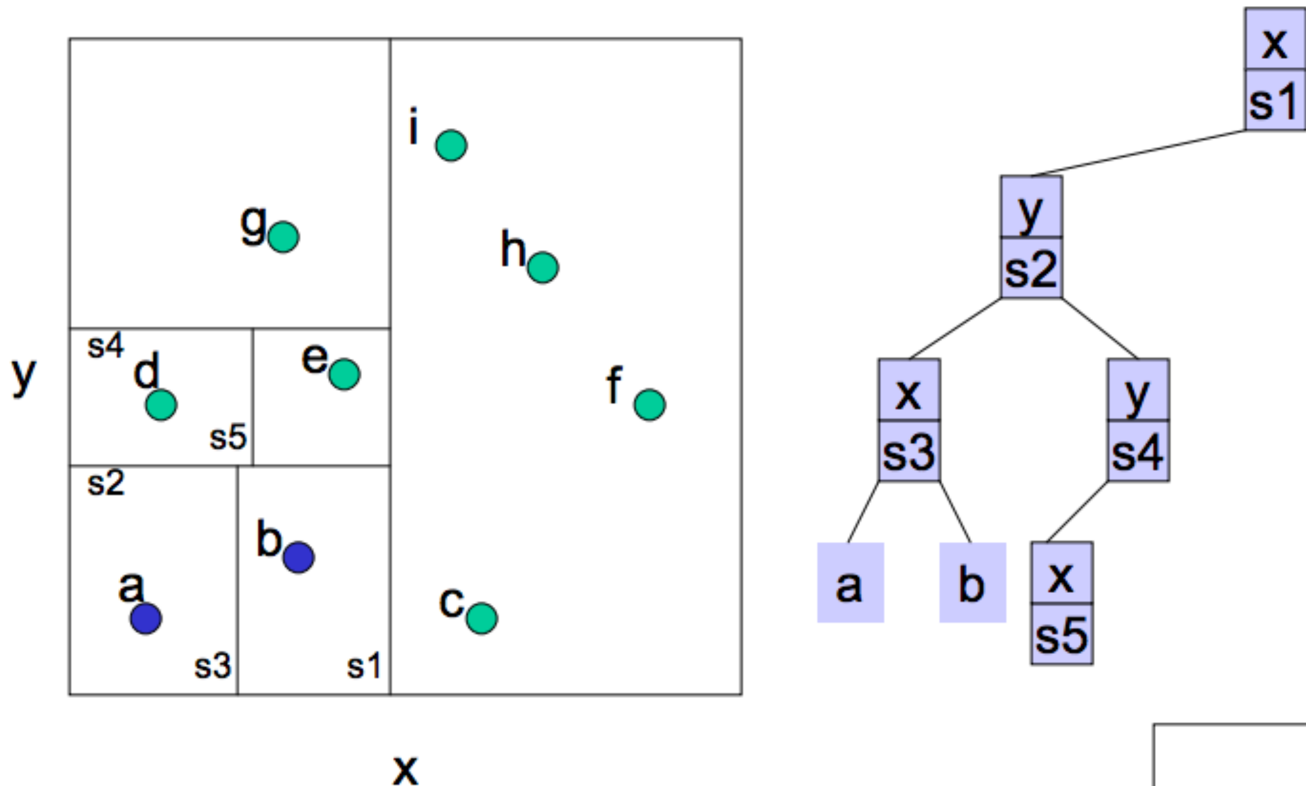


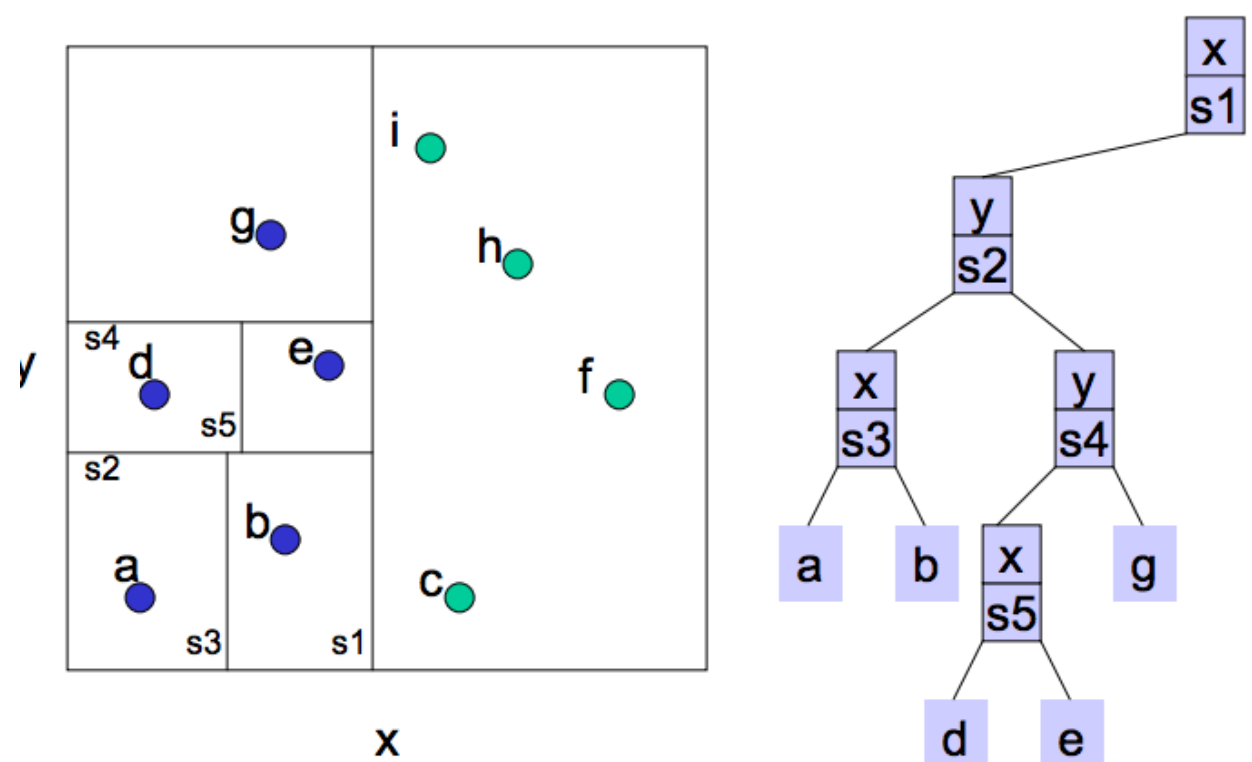
Example: kd-Tree Construction (2)



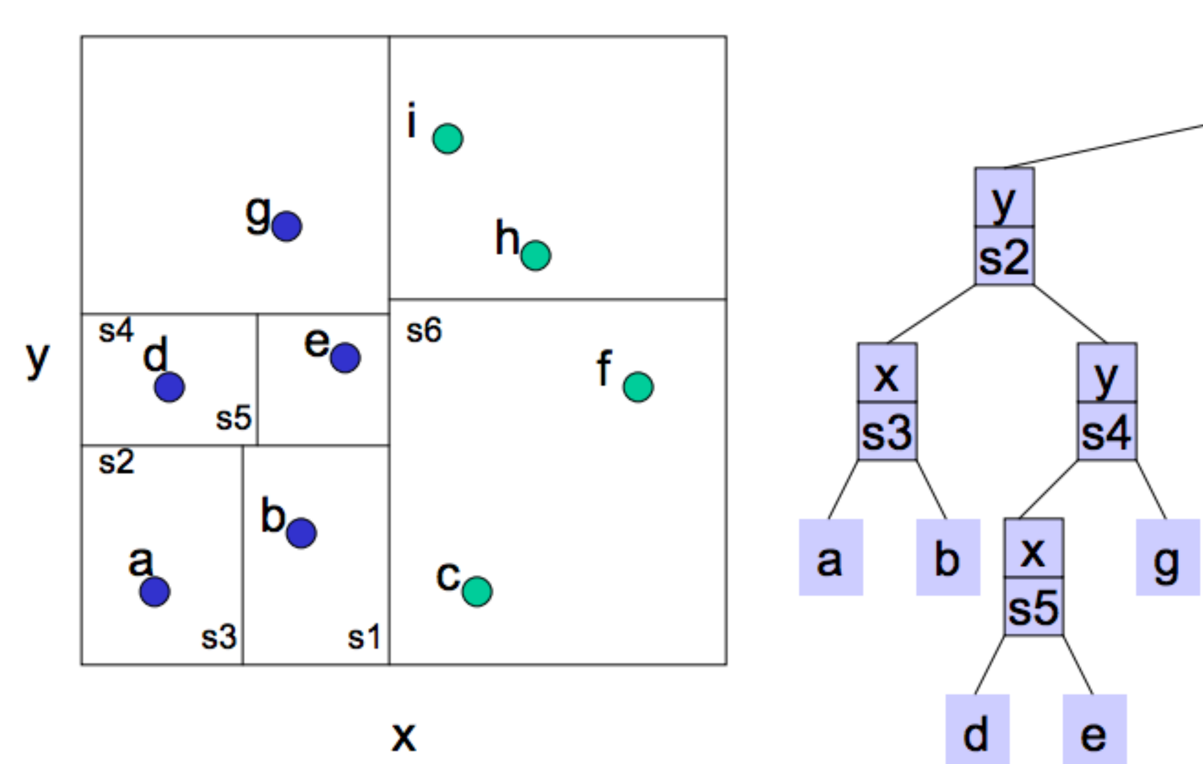


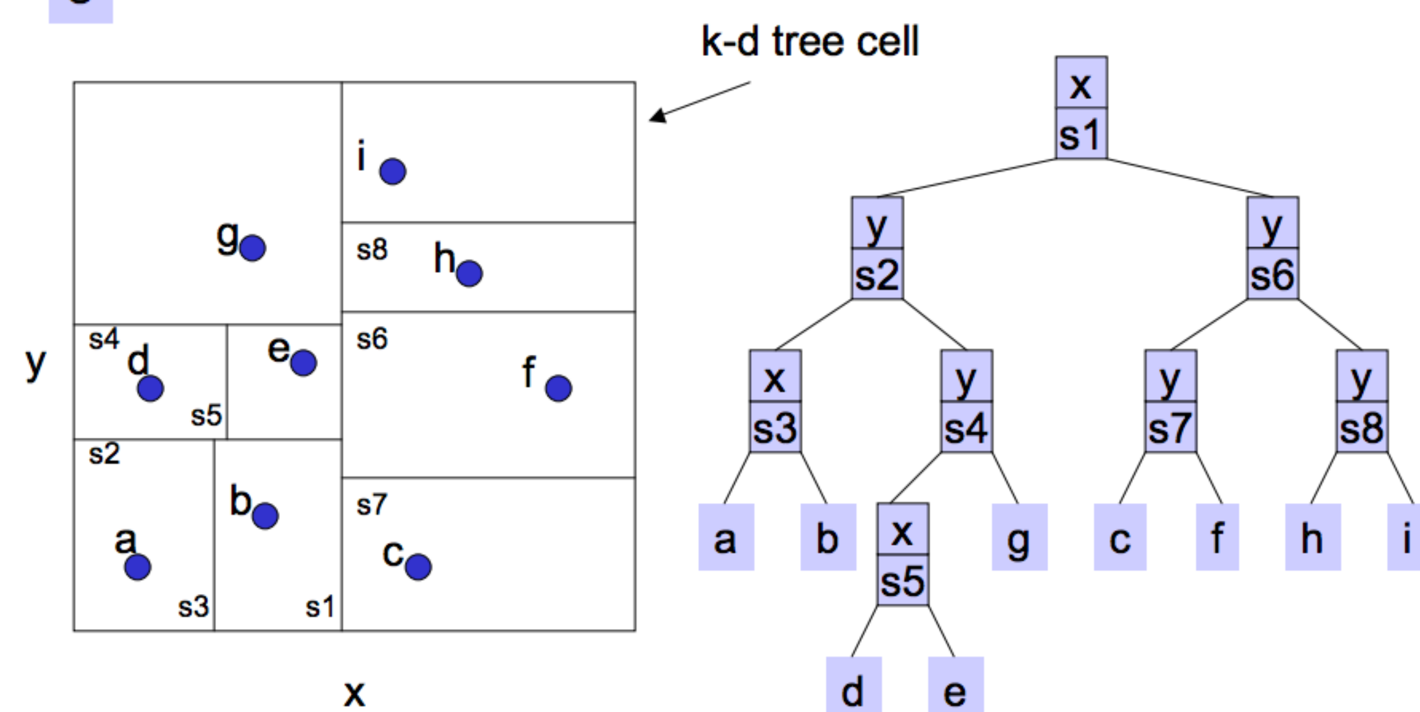
Example: kd-Tree Construction (3)





Example: kd-Tree Construction (4)

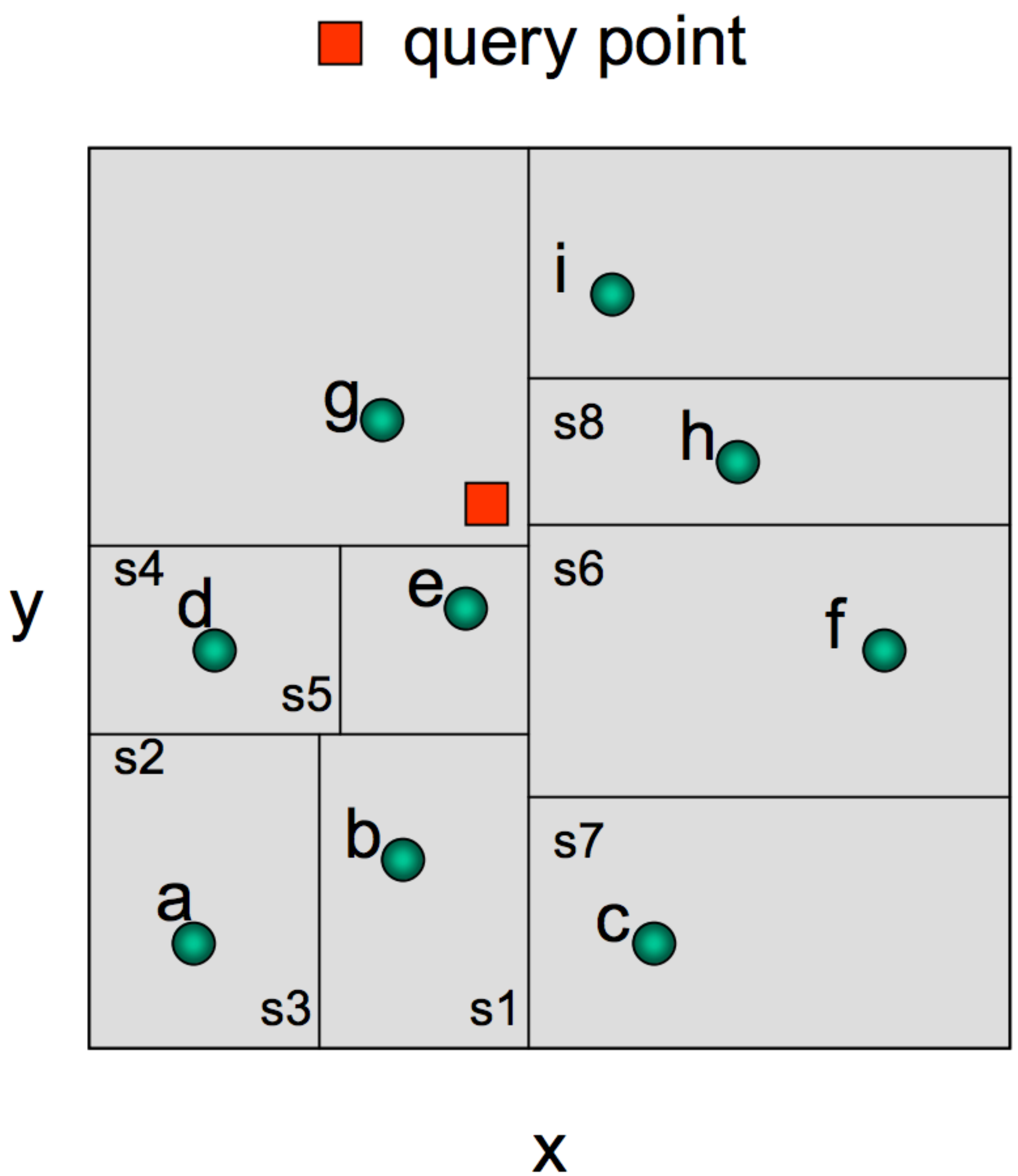


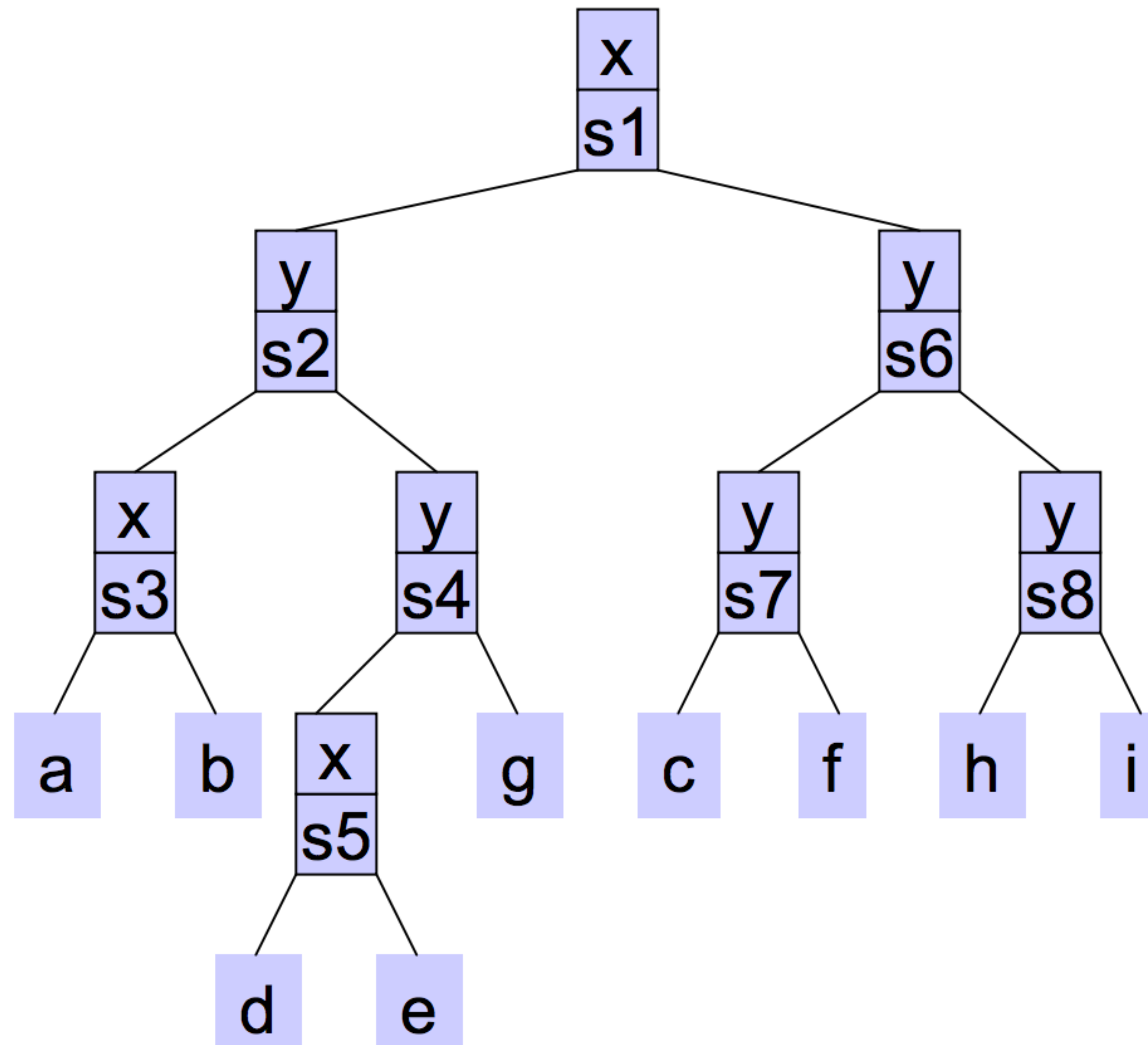


kd-Tree NN Search

- Search recursively to find the point in the same cell as the query
- On return search the each subtree where a closer point other than the one you know about might be found
- Has been shown to run in O(log n) average time per search in a reasonable model (assuming d is constant)

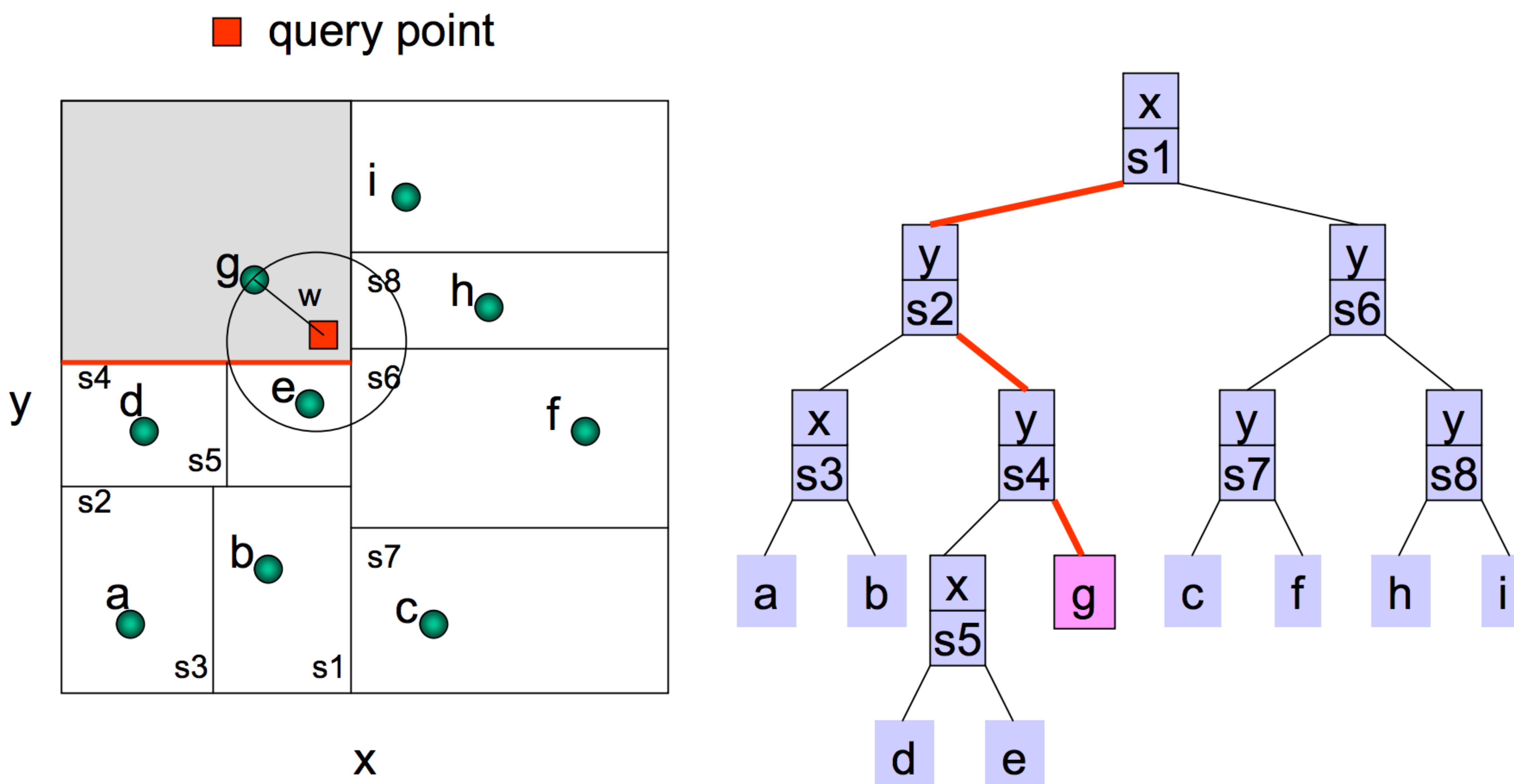
Example: kd-Tree Query



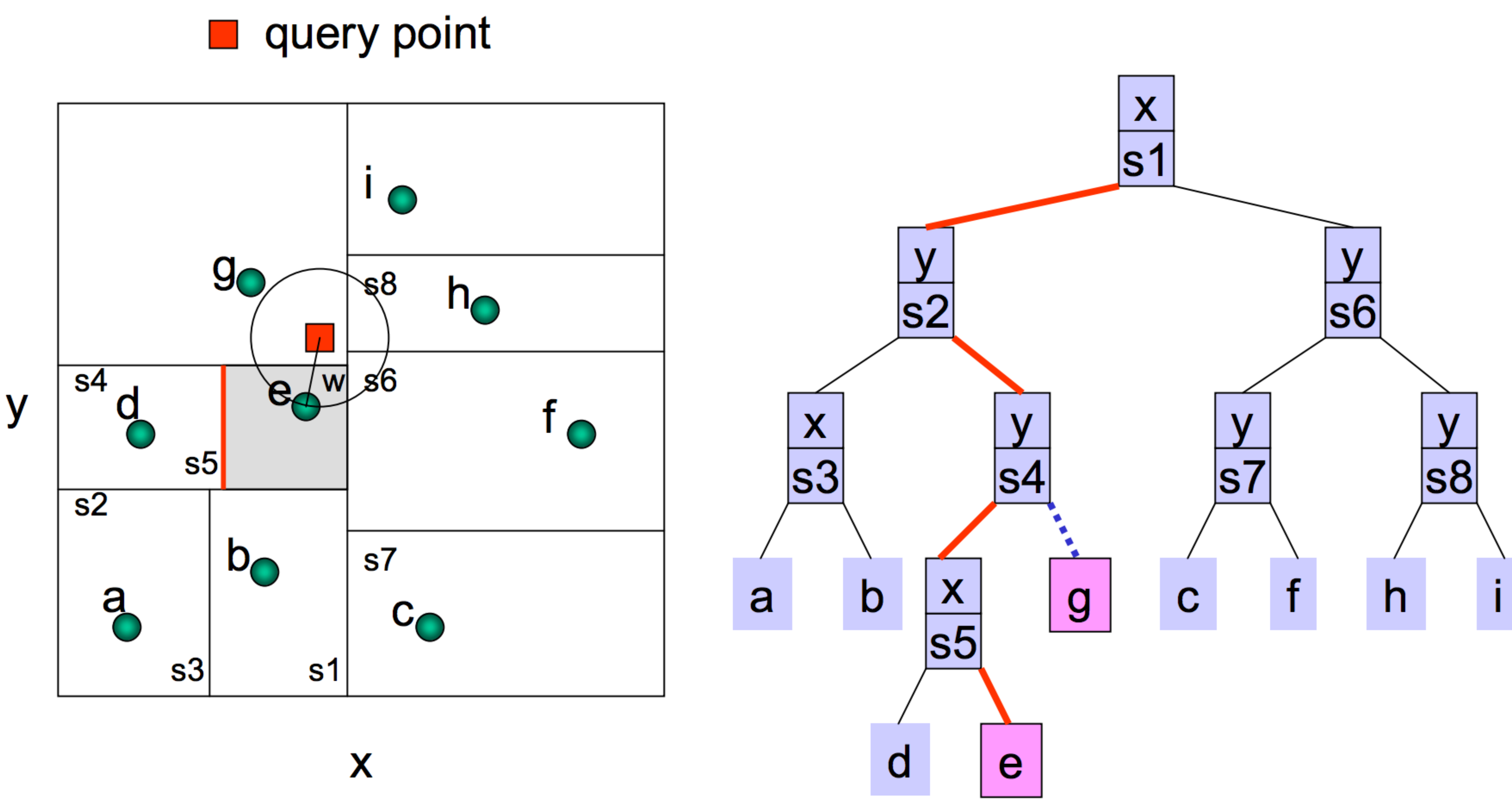


https://courses.cs.washington.edu/courses/cse373/02au/lectures/lecture22l.pdf

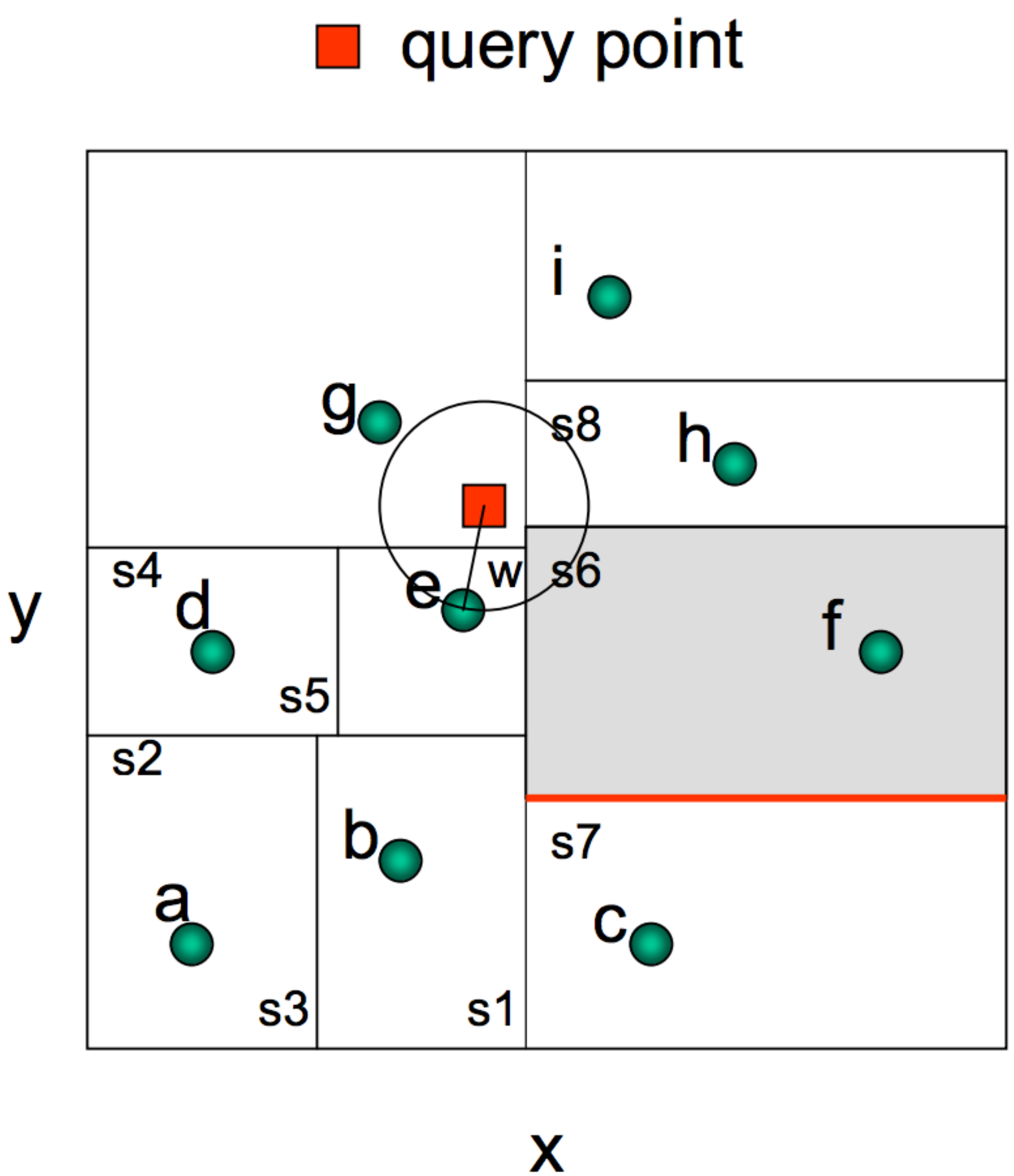
Example: kd-Tree Query (1)

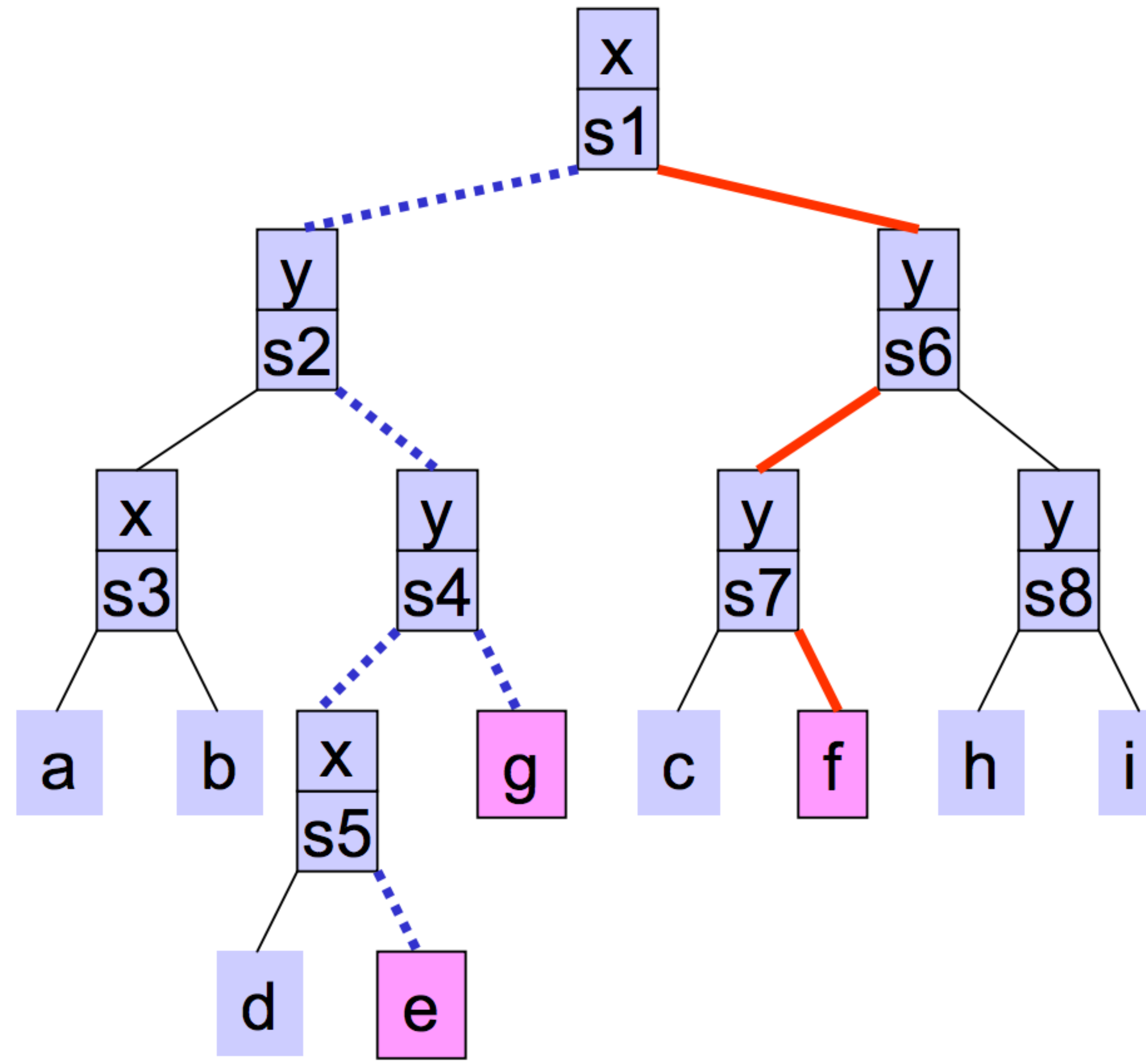


Example: kd-Tree Query (2)

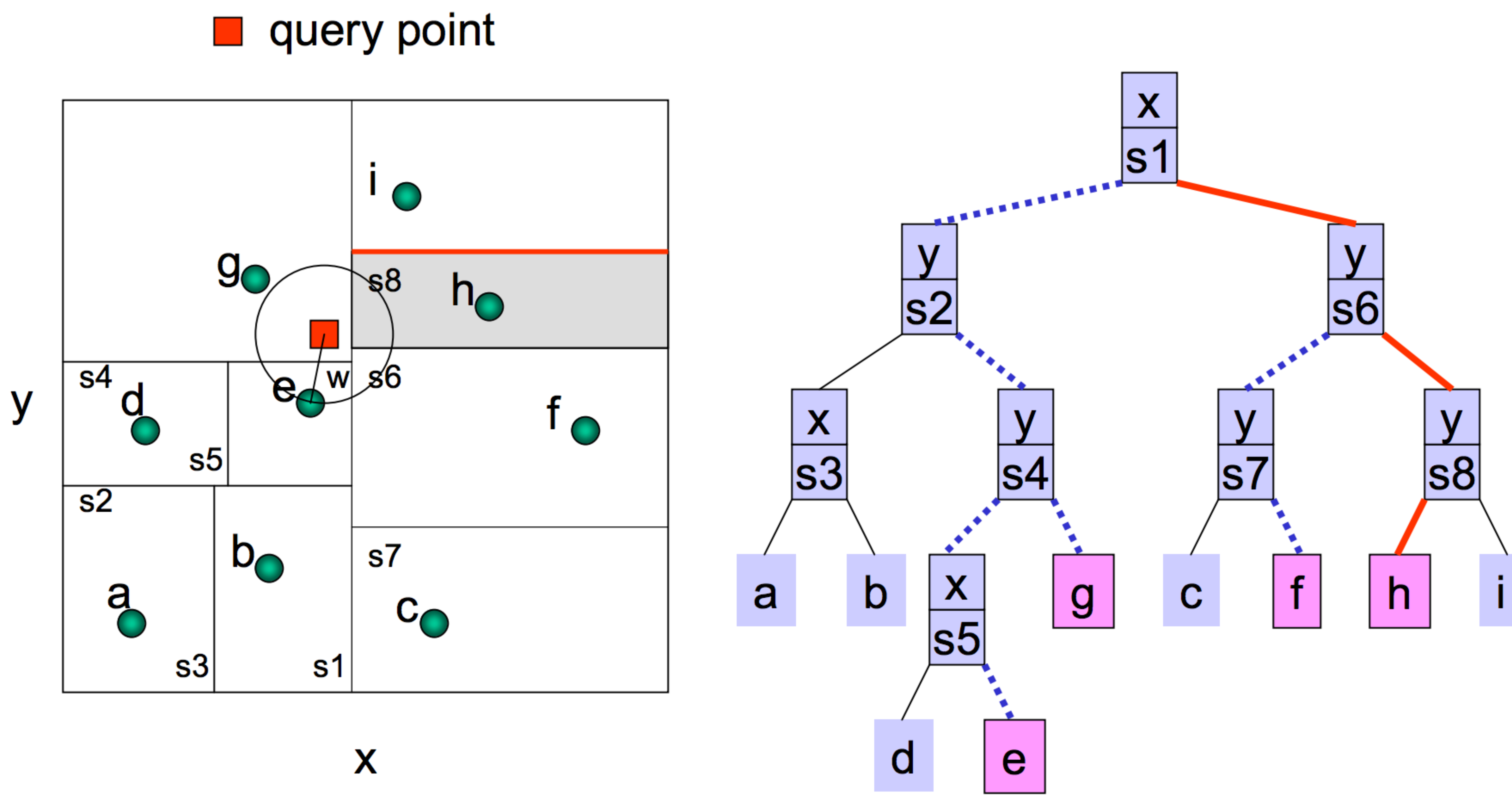


Example: kd-Tree Query (3)





Example: kd-Tree Query (4)



NN and the Curse of Dimensionality

NN breaks down in high-dimensional spaces because the "neighborhood" becomes very large

- d dimensions means d independent neighboring directions to the point

- Volume-distance ratio explodes $O(r^d)$

- Points become "equidistant" from a new point

kNN-Trees Summary

- Tons of variants
  - Construction of trees (e.g., heuristics for splitting, stopping, representing branches)
  - Other representational data structures for fast NN search (e.g., ball trees, …)
- High-dimensional spaces are hard

Python: Scikit-learn

- Brute force: $O(dn^2)$
- kd-tree: $O(d \log n)$ for $d < 20$
- Ball tree: $O(d \log n)$ but tree construction is more costly than kd-tree
- Benchmarking using NN: https://jakevdp.github.io/blog/2013/04/29/benchmarking-nearest-neighbor-searches-in-python/

Approximate Nearest Neighbor (ANN)

- Idea: rather than retrieve the exact closest neighbor, make a "good guess" of the nearest neighbor
- c-ANN: for any query q and points p:
  - r is the distance to the exact nearest neighbor q
  - Returns p in P, $||p - q|| \leq cr$, with probability at least $1 - \delta,\ \delta > 0$

ANN: Altering kd-Tree Search

(Augmented) kd-Trees are used but interrupt search earlier [Arya et al., 1994]

- Prune when distance to bounding box is greater than some distance r over some value alpha
- Saves lots of search time by removing some nodes of the tree
- In practice can get O(d log n) but worst case still has exponential running time

Beyond kd-Trees

- Works for low and medium dimensional data, but has problems with high-dimensional data
- Non-trivial to implement efficiently and still requires some computation of object similarities
- Can we represent similarities between objects in a more succinct manner?
    - Sacrifice exactness for efficiency by using randomization
    - Obtain a “sketch” of the object instead

Johnson-Lindenstrauss Lemma

Main Idea: small set of points in high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved

- One proof of the lemma uses projection onto random subspace
- Used in compressed sensing, manifold learning, dimensionality reduction, and graph embedding

Hash Functions

- A hash function, h, is a function which transforms a key from a set K, into an index in a table of size n
  h: K —> {0, 1, …, n-2, n-1}
- A good hash function should:
  - Minimize collisions
  - Be easy and quick to compute
  - Distribute key values evenly amongst the buckets
  - Use all the information provided in the key

Locality Sensitive Hashing (LSH)

- General idea: Use a hash function that tells whether x and y is a candidate pair (a pair of elements whose similarity must be evaluated)

- A hash function, h, is LSH if it satisfies for some similarity function d:

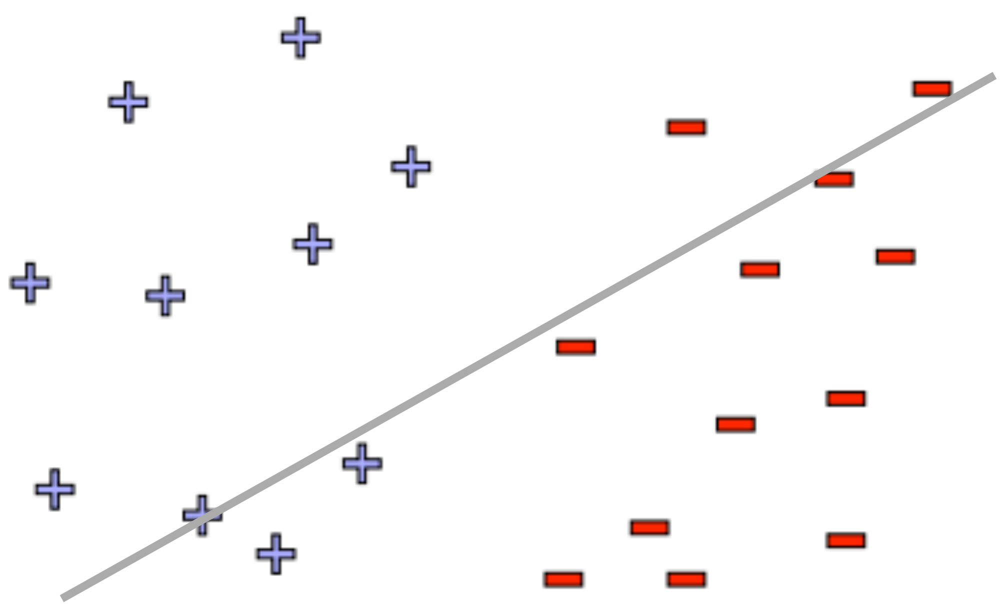
  - P(h(x) = h(y)) is high if $d(x, y) \leq r, \; r > 0$

  - P(h(x) = h(y) is low if $d(x, y) > \alpha r, \; r > 0, \; \alpha > 1$

  - (in between, not sure about probability)

http://courses.cs.washington.edu/courses/cse599c1/13wi/slides/lsh-hashkernels-annotated.pdf

Random Projection Illustrated



- Pick a random vector v using independent Gaussian coordinates
- Project the points onto this random vector
- For most vectors, it will preserve some notion of separability