CSEP 590
Data Compression
Autumn 2007

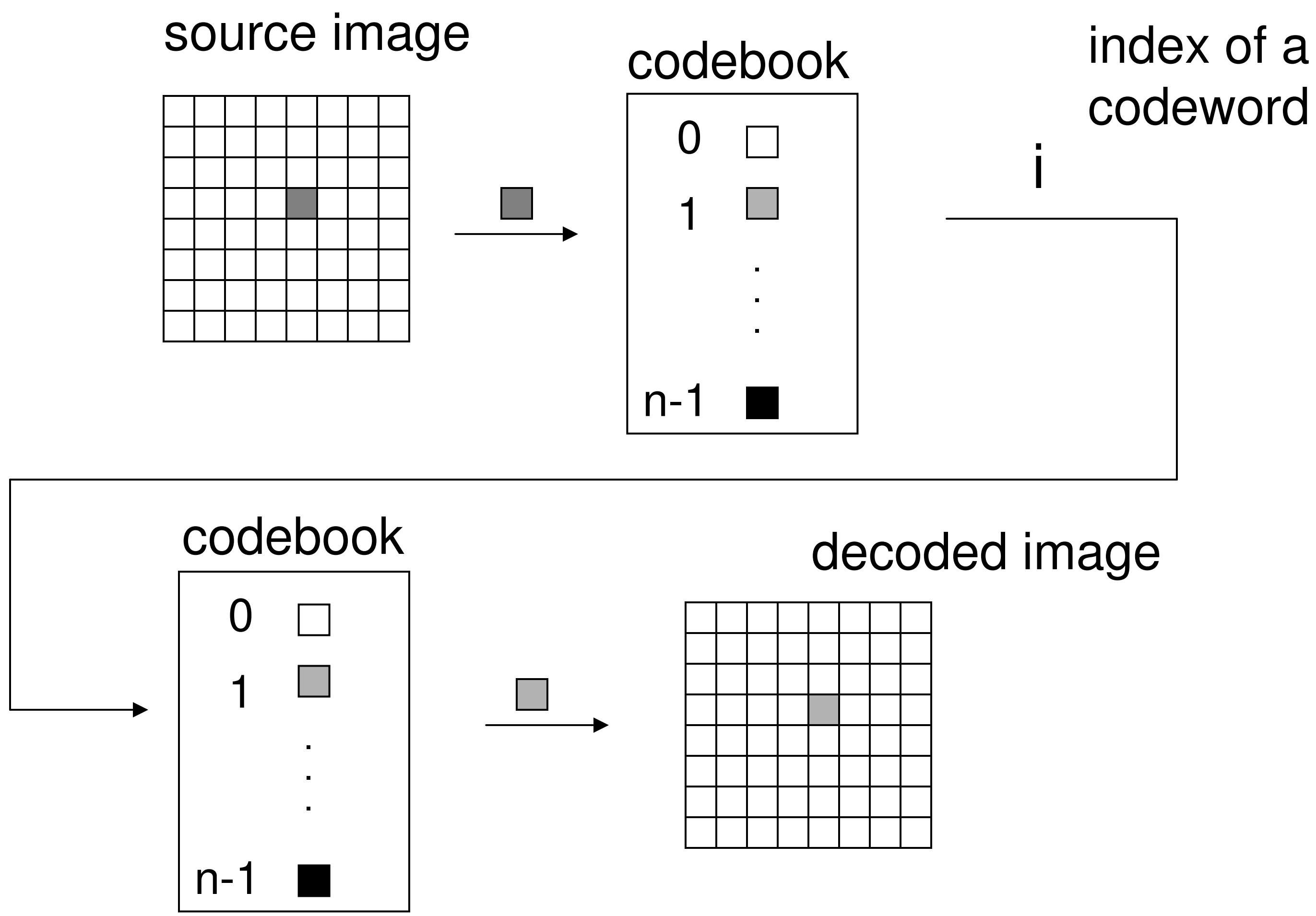
Scalar Quantization

Vector Quantization

Lossy Image Compression Methods

- DCT Compression
  - JPEG
- Scalar quantization (SQ).
- Vector quantization (VQ).
- Wavelet Compression
  - SPIHT
  - GTW
  - EBCOT
  - JPEG 2000

Scalar Quantization



Scalar Quantization Strategies

- Build a codebook with a training set. Encode and decode with fixed codebook.
  - Most common use of quantization
- Build a codebook for each image. Transmit the codebook with the image.
- Training can be slow.

Distortion

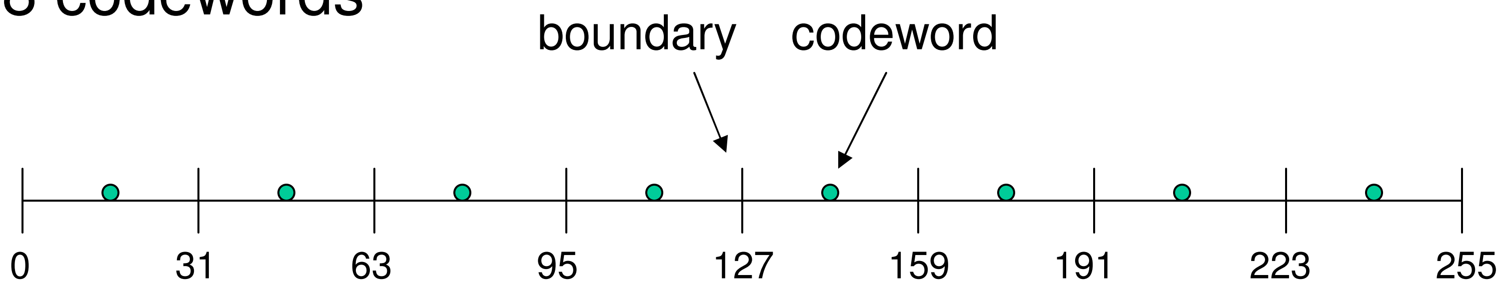
- Let the image be pixels x_1, x_2, … x_T.
- Define index(x) to be the index transmitted on input x.
- Define c(j) to be the codeword indexed by j.

$$D = \sum_{i=1}^{T} (x_i - c(\text{index}(x_i)))^2 \qquad \text{(Distortion)}$$

$$\text{MSE} = \frac{D}{T}$$

Uniform Quantization Example

- 512 x 512 image with 8 bits per pixel.
- 8 codewords



Codebook

Index	0	1	2	3	4	5	6	7
Codeword	16	47	79	111	143	175	207	239

Uniform Quantization Example

Encoder

input	0-31	32-63	64-95	96-127	128-159	160-191	192-223	224-255
code	000	001	010	011	100	101	110	111

Decoder

code	000	001	010	011	100	101	110	111
output	16	47	79	111	143	175	207	239

Bit rate = 3 bits per pixel
Compression ratio = 8/3 = 2.67

Example

- [0,100) with 5 symbols
- Q = 20

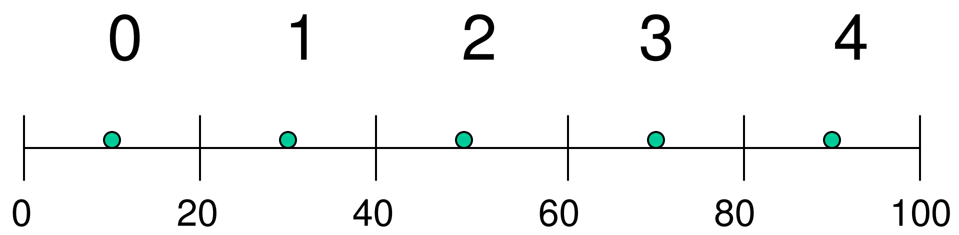
Encode Decode

$$0 = \left\lfloor \frac{10}{20} \right\rfloor \quad (0 + 1/2) \cdot 20 = 10$$

$$1 = \left\lfloor \frac{30}{20} \right\rfloor \quad (1 + 1/2) \cdot 20 = 30$$

$$2 = \left\lfloor \frac{50}{20} \right\rfloor \quad (2 + 1/2) \cdot 20 = 50$$

$$\vdots$$



Alternative Uniform Quantization Calculation with Push to Zero

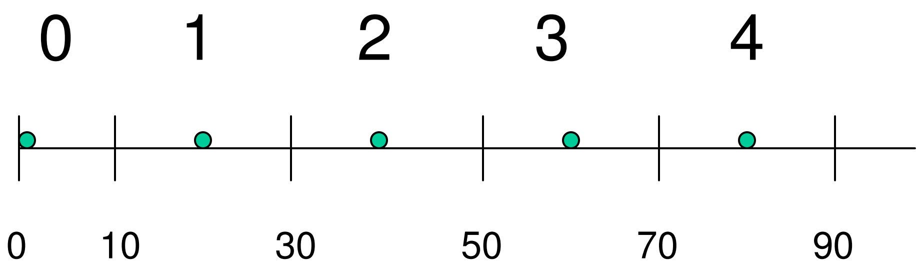
- Range = [min, max)
- Target is S symbols
- Choose Q = (max – min)/S
- Encode x $$s = \left\lfloor \frac{x}{Q} + 1/2 \right\rfloor$$
- Decode s $$x' = s\,Q$$

Example

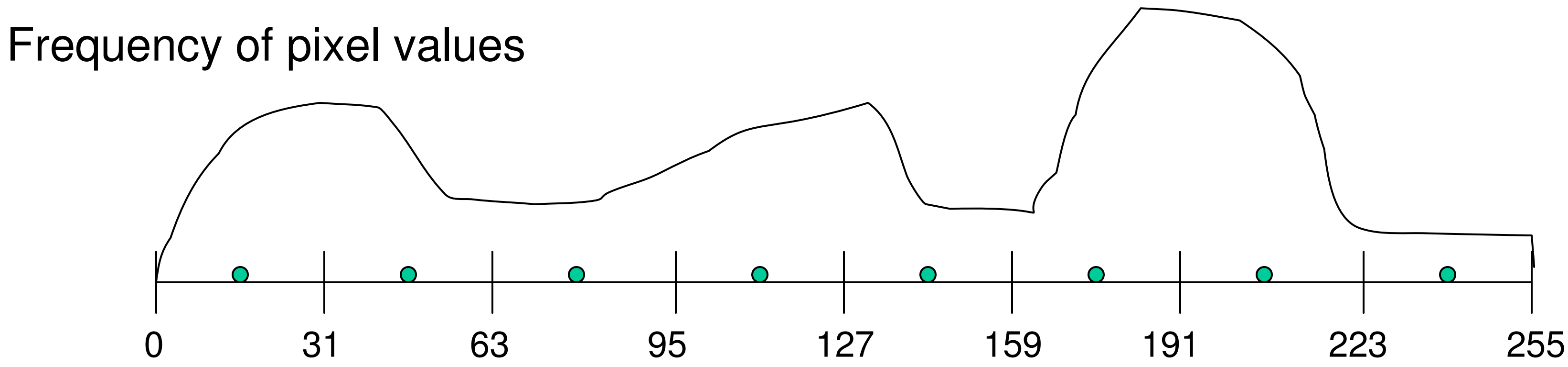
- [0,90) with 5 symbols
- Q = 20

Encode	Decode
$0 = \left\lfloor \frac{9.99}{20} + 1/2 \right\rfloor$	$0 \cdot 20 = 0$
$1 = \left\lfloor \frac{29.99}{20} + 1/2 \right\rfloor$	$1 \cdot 20 = 20$
$2 = \left\lfloor \frac{49.99}{20} + 1/2 \right\rfloor$	$2 \cdot 20 = 40$

$\vdots$



Improving Bit Rate



q_j = the probability that a pixel is coded to index j
Potential average bit rate is entropy.

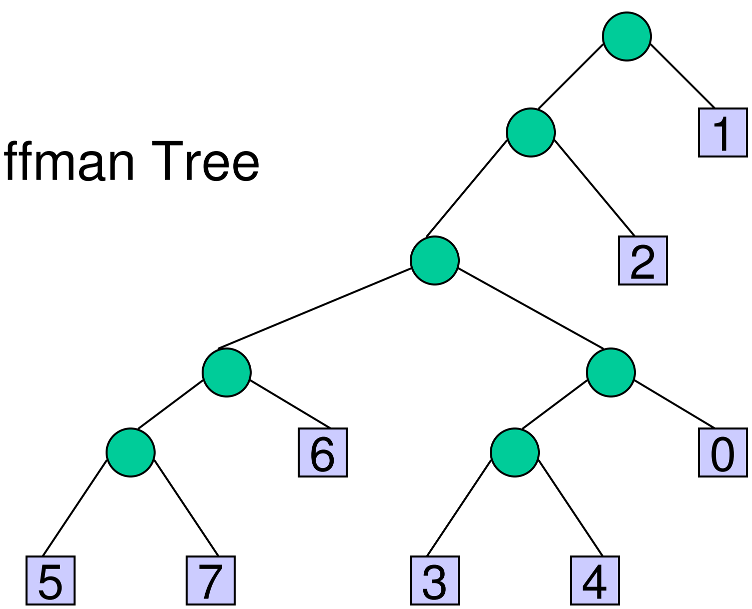
$$H = \sum_{j=0}^{7} q_j \log_2\left(\frac{1}{q_j}\right)$$

Example

- 512 x 512 image = 216,144 pixels

index	0	1	2	3	4	5	6	7
input	0-31	32-63	64-95	96-127	128-159	160-191	192-223	224-255
frequency	25,000	100,000	90,000	10,000	10,000	10,000	18,000	9,144

Huffman Tree

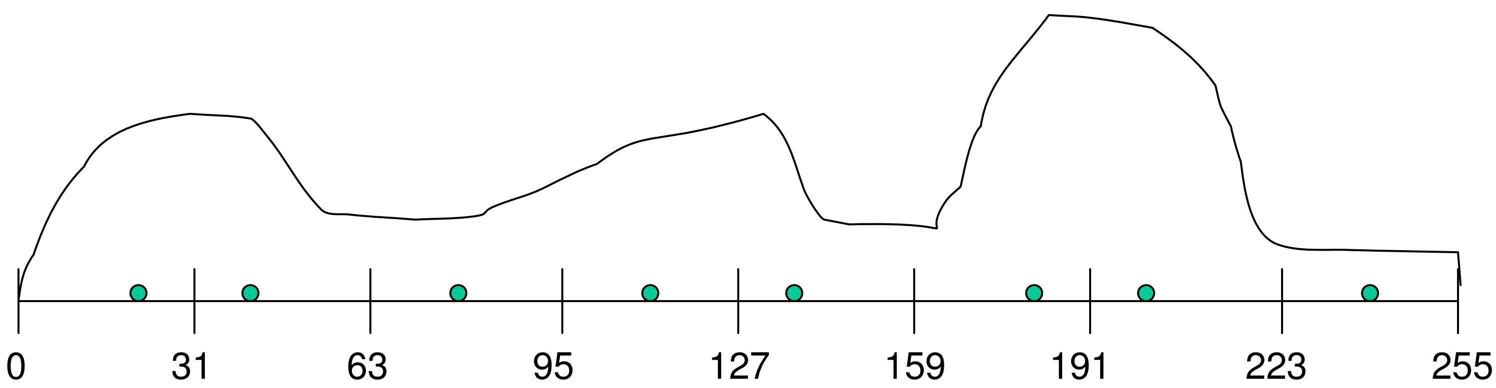


ABR= (100000 x 1+
90000 x 2 +
43000 x 4 +
39144 x 5)/216144
=2.997

Arithmetic coding should work better.

Improving Distortion

- Choose the codeword as a weighted average



Let p_x be the probability that a pixel has value x.
Let $[L_j, R_j)$ be the input interval for index j.
c(j) is the codeword indexed j

$$c(j) = \text{round}\left(\sum_{L_j \le x < R_j} x \cdot p_x \right)$$

Example

All pixels have the same index.

pixel value	8	9	10	11	12	13	14	15
frequency	100	100	100	40	30	20	10	0

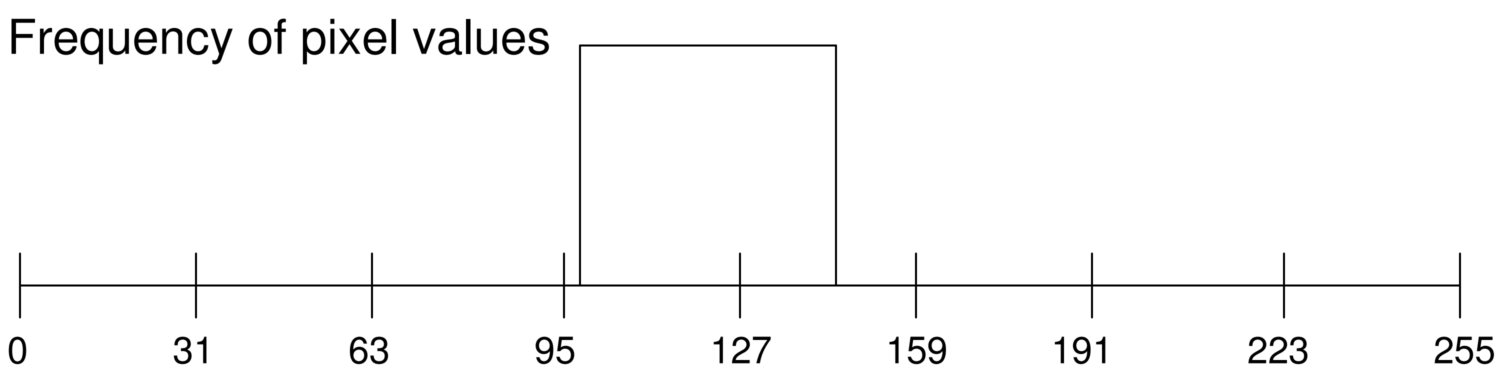
$$\text{New Codeword} = \text{round}\left(\frac{8\cdot 100 + 9\cdot 100 + 10\cdot 100 + 11\cdot 40 + 12\cdot 30 + 13\cdot 20 + 14\cdot 10 + 15\cdot 0}{400}\right) = 10$$

$$\text{Old Codeword} = 11$$

$$\text{New Distortion} = 140\cdot 1^2 + 130\cdot 2^2 + 20\cdot 3^2 + 10\cdot 4^2 = 10000$$

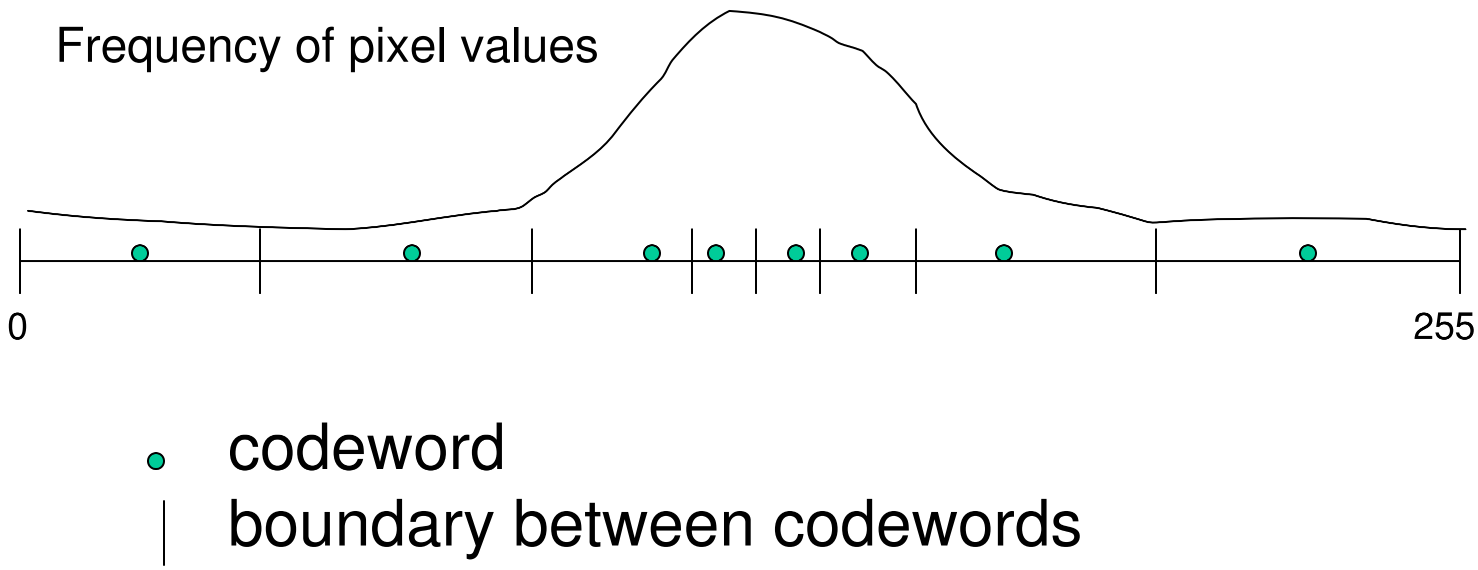
$$\text{Old Distortion} = 130\cdot 1^2 + 120\cdot 2^2 + 110\cdot 3^2 = 16000$$

An Extreme Case



Only two codewords are ever used!!

Non-uniform Scalar Quantization



Lloyd Algorithm

- Lloyd (1957)
- Creates an optimized codebook of size n.
- Let p_x be the probability of pixel value x.
  - Probabilities might come from a training set
- Given codewords c(0),c(1),...,c(n-1) and pixel x let index(x) be the index of the closest code word to x.
- Expected distortion is

$$D = \sum_x p_x (x - c(\mathrm{index}(x)))^2$$

- Goal of the Lloyd algorithm is to find the codewords that minimize distortion.
- Lloyd finds a local minimum by an iteration process.

Lloyd Algorithm

Choose a small error tolerance $\varepsilon > 0$.
Choose start codewords c(0),c(1),...,c(n-1)
Compute X(j) := {x : x is a pixel value closest to c(j)}
Compute distortion D for c(0),c(1),...,c(n-1)
Repeat
  Compute new codewords

$$c'(j) := \text{round}\left(\sum_{x \in X(j)} x \cdot p_x / p_{X(j)}\right)$$

  Compute X'(j) = {x : x is a pixel value closest to c'(j)}
  Compute distortion D' for c'(0),c'(1),...,c'(n-1)
  if $|(D - D')/D| < \varepsilon$ then quit
  else c := c'; X := X', D := D'
End{repeat}

Example

Initially c(0) = 2 and c(1) = 5

pixel value	0	1	2	3	4	5	6	7
frequency	100	100	100	40	30	20	10	0

$$X(0) = [0,3],\ X(1) = [4,7]$$

$$D(0) = 140 \cdot 1^2 + 100 \cdot 2^2 = 540;\ D(1) = 40 \cdot 1^2 = 40$$

$$D = D(0) + D(1) = 580$$

$$c'(0) = \text{round}((100 \cdot 0 + 100 \cdot 1 + 100 \cdot 2 + 40 \cdot 3)/340) = 1$$

$$c'(1) = \text{round}((30 \cdot 4 + 20 \cdot 5 + 10 \cdot 6 + 0 \cdot 7)/60) = 5$$

Example

pixel value	0	1	2	3	4	5	6	7
frequency	100	100	100	40	30	20	10	0

$$c'(0) = 1;\, c'(1) = 5$$

$$X'(0) = [0,2];\; X'(1) = [3,7]$$

$$D'(0) = 200 \cdot 1^2 = 200$$

$$D'(1) = 40 \cdot 1^2 + 40 \cdot 2^2 = 200$$

$$D' = D'(0) + D'(1) = 400$$

$$\left|(D - D')/D\right| = (580 - 400)/580 = .31$$

$$c := c';\, X := X';\, D := D'$$

Example

pixel value	0	1	2	3	4	5	6	7
frequency	100	100	100	40	30	20	10	0

$$c(0) = 1; c(1) = 5$$

$$X(0) = [0,2]; X(1) = [3,7]$$

$$D = 400$$

$$c'(0) = \text{round}((100 \cdot 0 + 100 \cdot 1 + 100 \cdot 2)/300) = 1$$

$$c'(1) = \text{round}((40 \cdot 3 + 30 \cdot 4 + 20 \cdot 5 + 10 \cdot 6 + 0 \cdot 7)/100) = 4$$

Example

pixel value	0	1	2	3	4	5	6	7
frequency	100	100	100	40	30	20	10	0

$$c'(0) = 1; c'(1) = 4$$

$$X'(0) = [0,2]; X'(1) = [3,7]$$

$$D'(0) = 200 \cdot 1^2 = 200$$

$$D'(1) = 60 \cdot 1^2 + 10 \cdot 2^2 = 100$$

$$D' = D'(0) + D'(1) = 300$$

$$\left|(D - D')/D\right| = (400 - 300)/580 = .17$$

$$c := c'; X := X'; D := D'$$

Example

pixel value	0	1	2	3	4	5	6	7
frequency	100	100	100	40	30	20	10	0

$$c(0) = 1; c(1) = 4$$

$$X(0) = [0,2]; X(1) = [3,7]$$

$$D = 400$$

$$c'(0) = \text{round}((100 \cdot 0 + 100 \cdot 1 + 100 \cdot 2)/300) = 1$$

$$c'(1) = \text{round}((40 \cdot 3 + 30 \cdot 4 + 20 \cdot 5 + 10 \cdot 6 + 0 \cdot 7)/100) = 4$$

Example

pixel value	0	1	2	3	4	5	6	7
frequency	100	100	100	40	30	20	10	0

$$c'(0) = 1; c'(1) = 4$$

$$X'(0) = [0,2]; X'(1) = [3,7]$$

$$D'(0) = 200 \cdot 1^2 = 200$$

$$D'(1) = 60 \cdot 1^2 + 10 \cdot 2^2 = 100$$

$$D' = D'(0) + D'(1) = 300$$

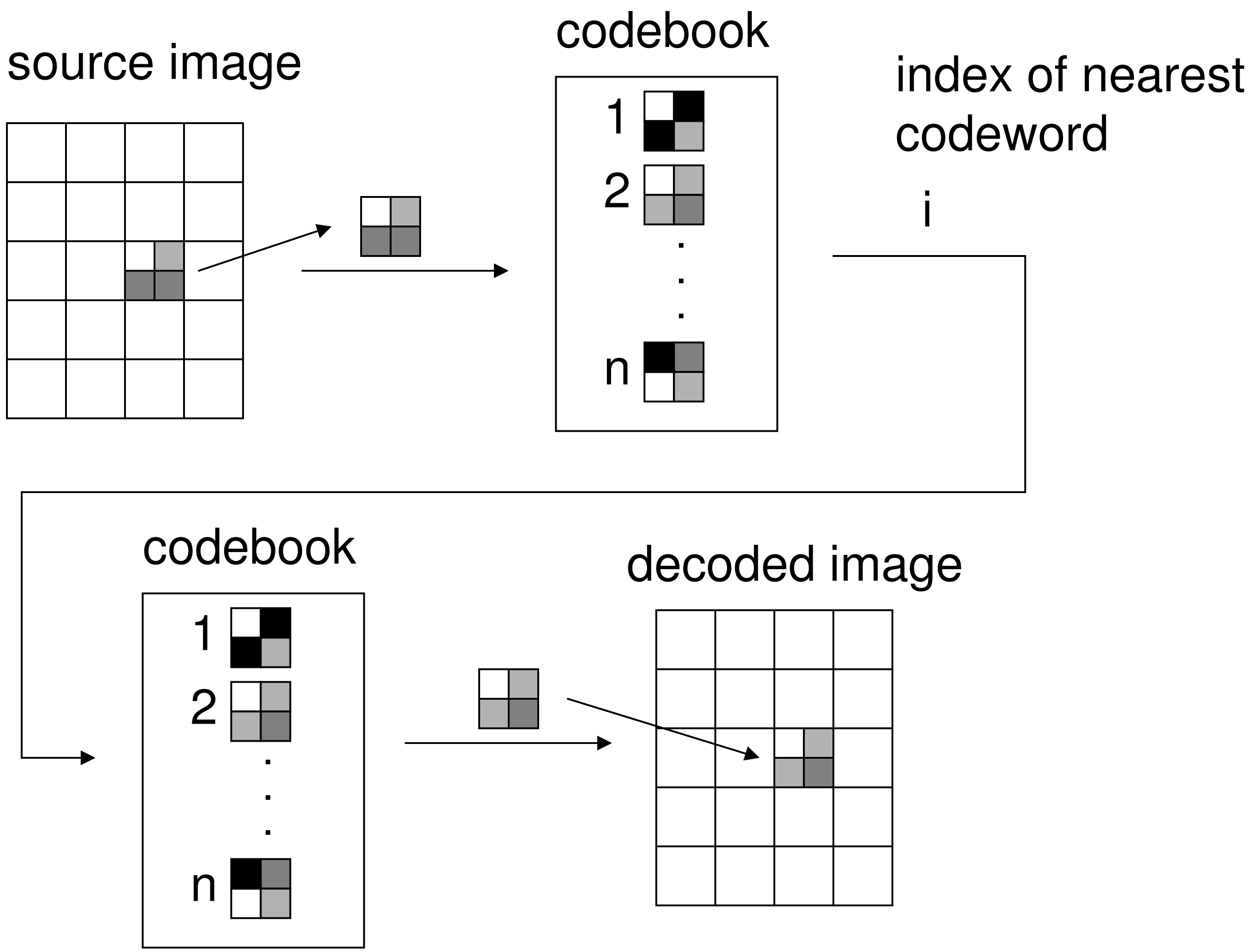
$$\left|(D - D')/D\right| = (300 - 300)/580 = 0$$

Exit with codeword $c(0) = 1$ and $c(1) = 4$.

Scalar Quantization Notes

- Useful for analog to digital conversion.
- Useful for estimating a large set of values with a small set of values.
- With entropy coding yields good lossy compression.
- Lloyd algorithm works very well in practice, but can take many iterations.
  - For n codewords should use about 20n size representative training set.
  - imagine 1024 codewords.

Vector Quantization



Vectors

- An a x b block can be considered to be a vector of dimension ab.

w	x
y	z

block (above) = (w,x,y,z) vector

- Nearest means in terms of Euclidian distance or Euclidian squared distance. Both equivalent.

$$\text{Distance} = \sqrt{(w_1 - w_2)^2 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

$$\text{Squared Distance} = (w_1 - w_2)^2 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$$

- Squared distance is easier to calculate.

Vector Quantization Facts

- The image is partitioned into a x b blocks.
- The codebook has n representative a x b blocks called codewords, each with an index.
- Compression with fixed length codes is

$$\frac{\log_2 n}{ab} \text{ bpp}$$

- Example: a = b = 4 and n = 1,024
  - compression is 10/16 = .63 bpp
  - compression ratio is 8 : .63 = 12.8 : 1
- Better compression with entropy coding of indices

Examples

4 x 4 blocks
.63 bpp

4 x 8 blocks
.31 bpp

8 x 8 blocks
.16 bpp

Codebook size = 1,024

Scalar vs. Vector

- Pixels within a block are correlated.
  - This tends to minimize the number of codewords needed to represent the vectors well.
- More flexibility.
  - Different size blocks
  - Different size codebooks

Encoding and Decoding

- Encoding:
  - Scan the a x b blocks of the image. For each block find the nearest codeword in the codebook and output its index.
  - Nearest neighbor search.
- Decoding:
  - For each index output the codeword with that index into the destination image.
  - Table lookup.

The Codebook

- Both encoder and decoder must have the same codebook.
- The codebook must be useful for many images and be stored someplace.
- The codebook must be designed properly to be effective.
- Design requires a representative training set.
- These are major drawbacks to VQ.

Codebook Design Problem

- Input: A training set X of vectors of dimension d and a number n. (d = a x b and n is number of codewords)
- Ouput: n codewords c(0), c(1),...,c(n-1) that minimize the distortion.

$$D = \sum_{x \in X} \left\| x - c(\text{index}(x)) \right\|^2$$ sum of squared distances

where index(x) is the index of the nearest codeword to x.

$$\left\| (x_0, x_1 \cdots, x_{d-1}) \right\|^2 = {x_0}^2 + {x_1}^2 + \cdots + {x_{d-1}}^2$$ squared norm

GLA

- The Generalized Lloyd Algorithm (GLA) extends the Lloyd algorithm for scalars.
  - Also called LBG after inventors Linde, Buzo, Gray (1980)
- It can be very slow for large training sets.

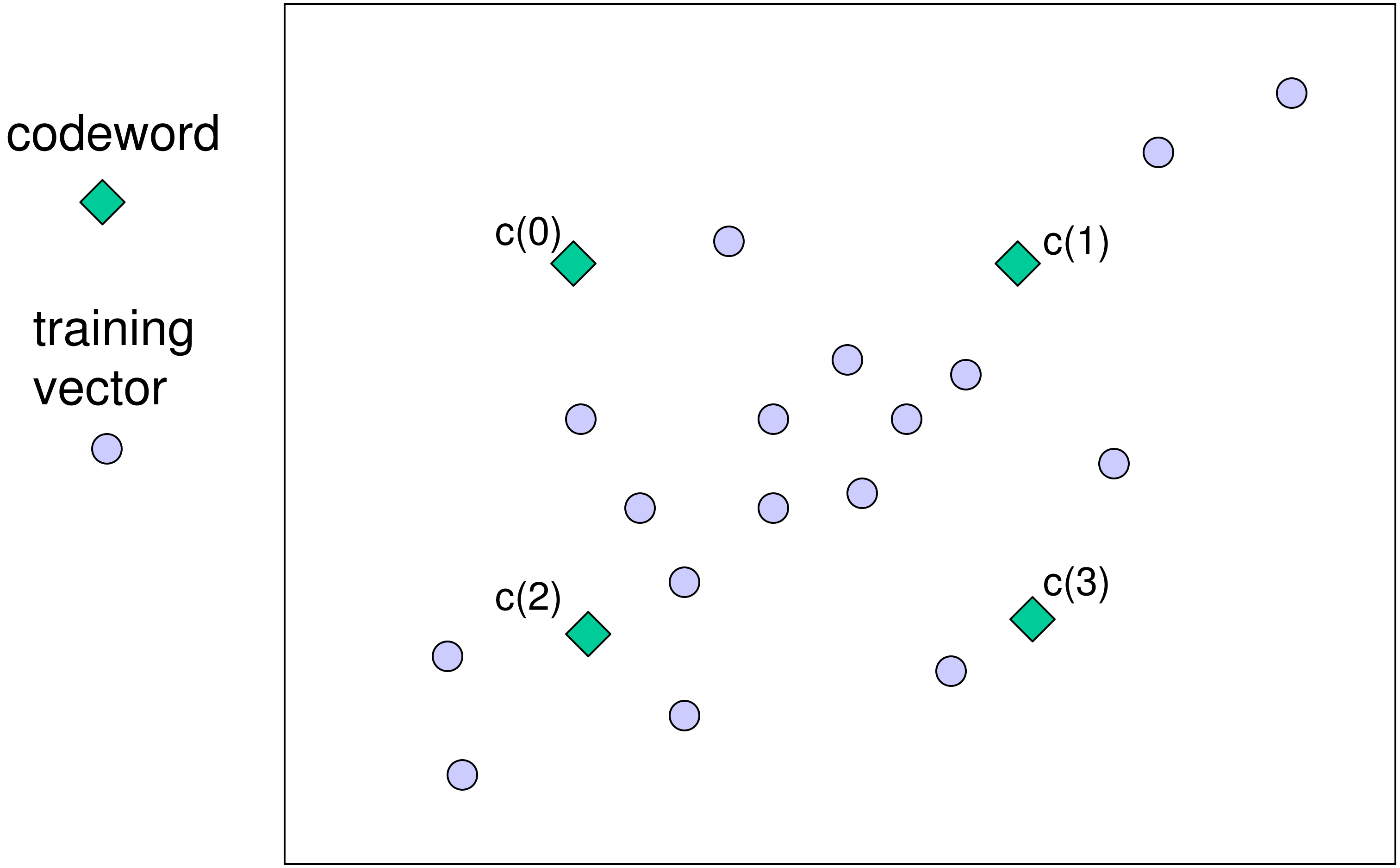
GLA

Choose a training set X and small error tolerance $\varepsilon > 0$.
Choose start codewords c(0),c(1),...,c(n-1)
Compute X(j) := {x : x is a vector in X closest to c(j)}
Compute distortion D for c(0),c(1),...,c(n-1)
Repeat
  Compute new codewords

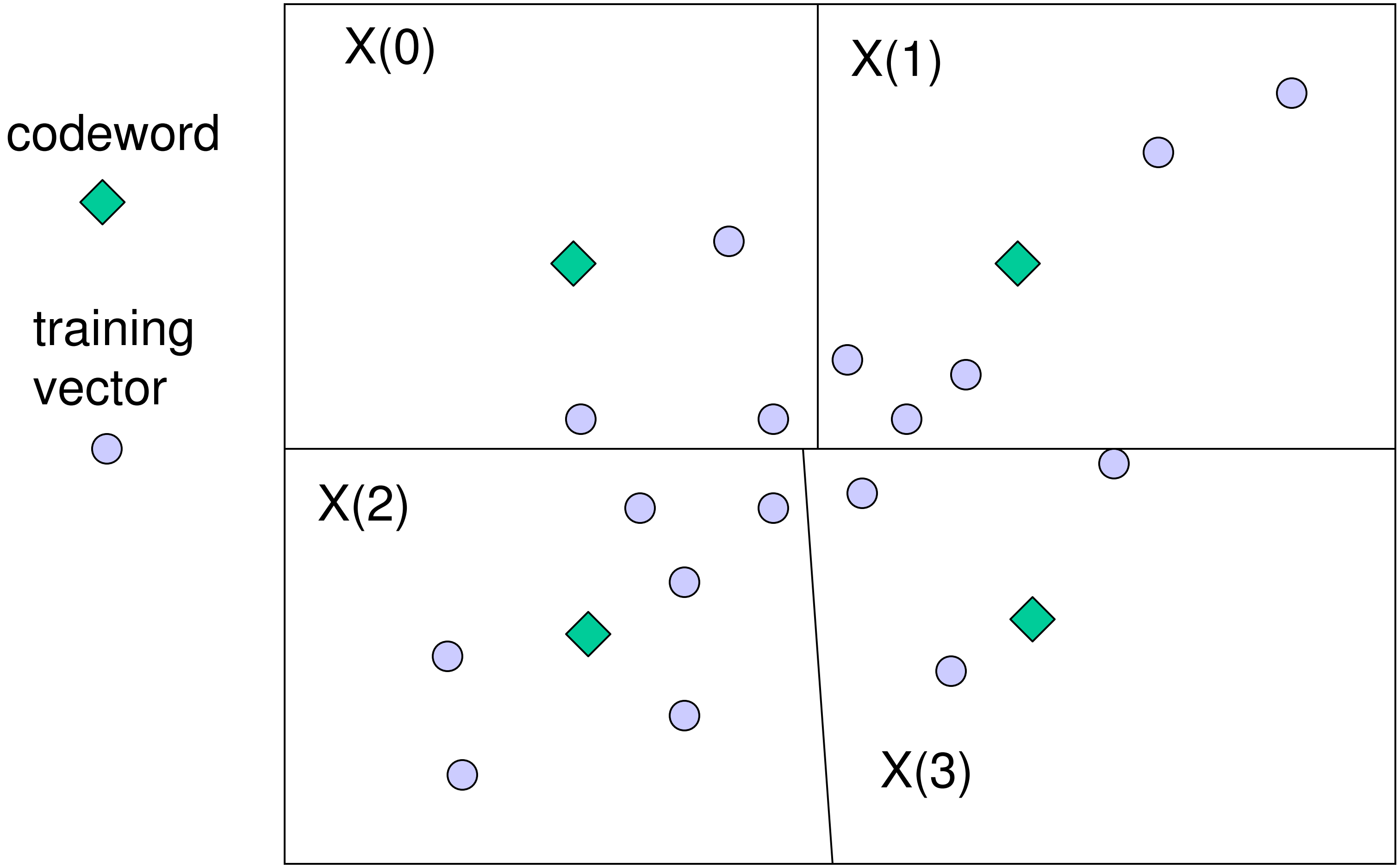
$$c'(j) := \text{round}\left(\frac{1}{|X(j)|}\sum_{x \in X(j)} x\right) \quad \text{(centroid)}$$

  Compute X'(j) = {x : x is a vector in X closest to c'(j)}
  Compute distortion D' for c'(0),c'(1),...,c'(n-1)
  if $|(D - D')/D| < \varepsilon$ then quit
  else c := c'; X := X', D := D'
End{repeat}

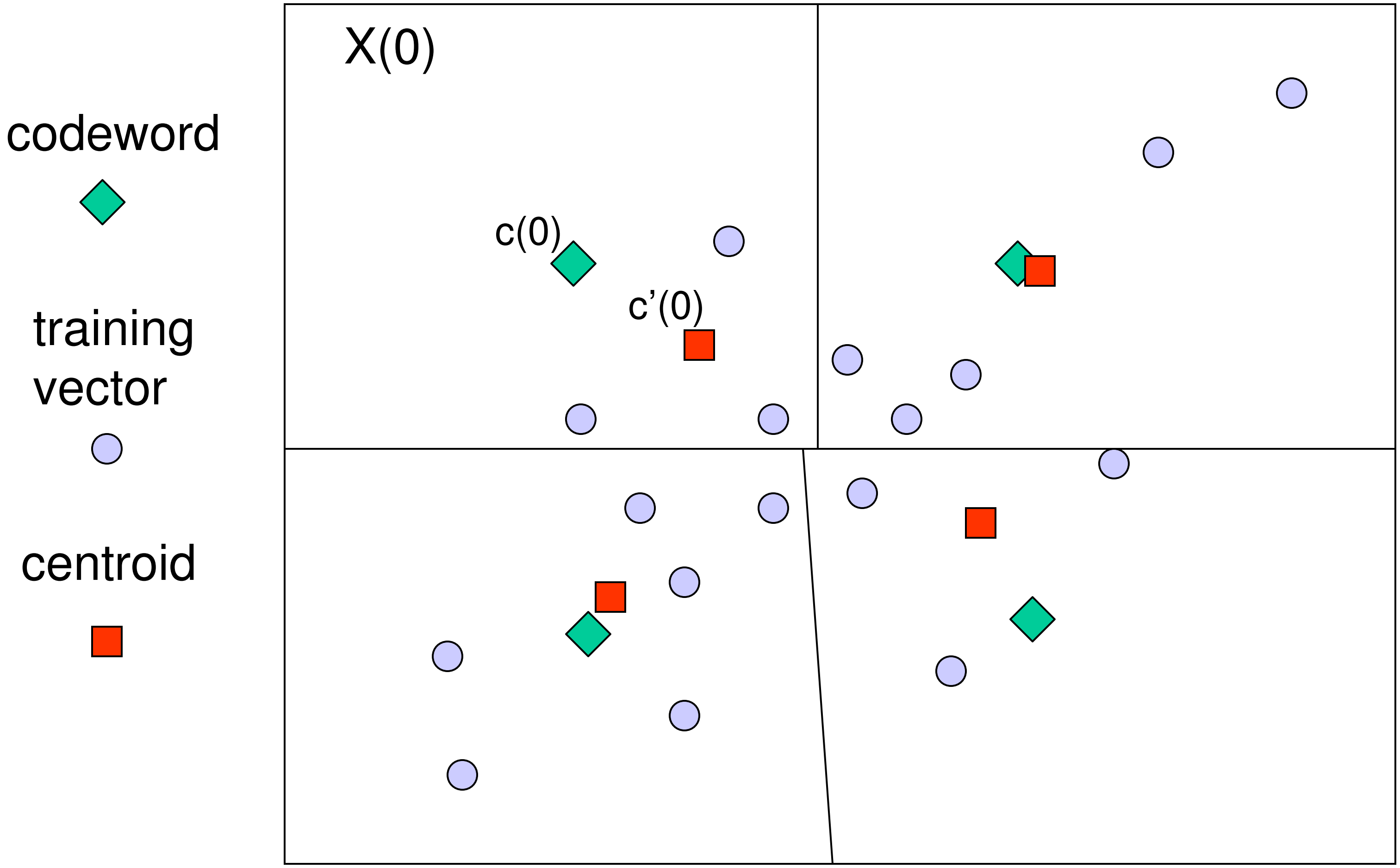
GLA Example (1)



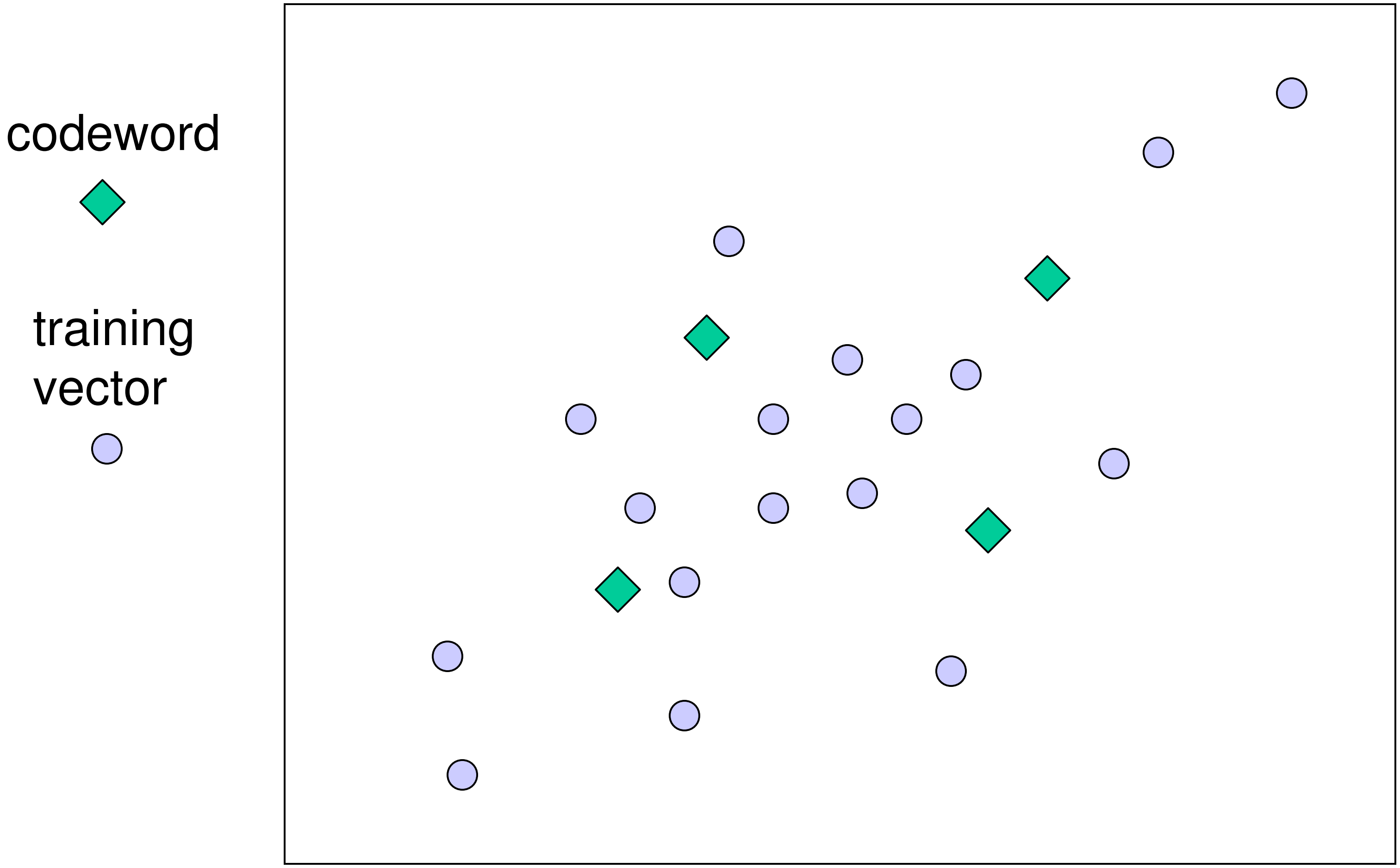
GLA Example (2)



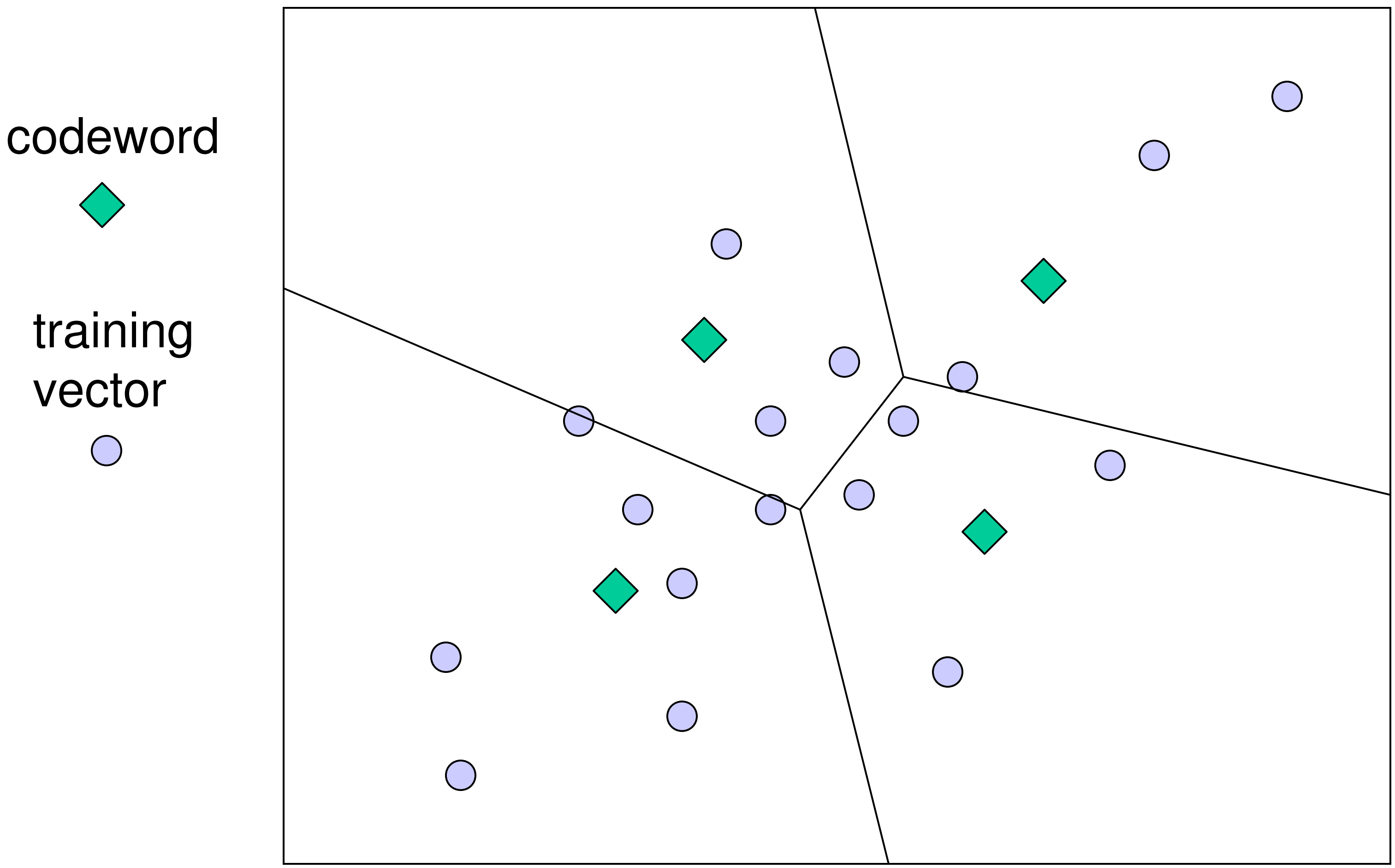
GLA Example (3)



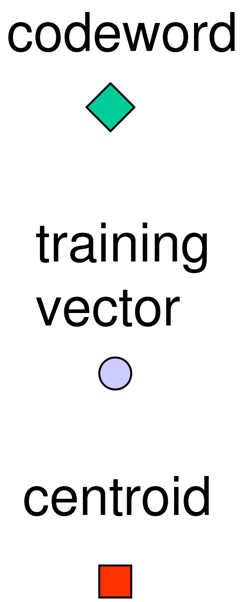
GLA Example (4)

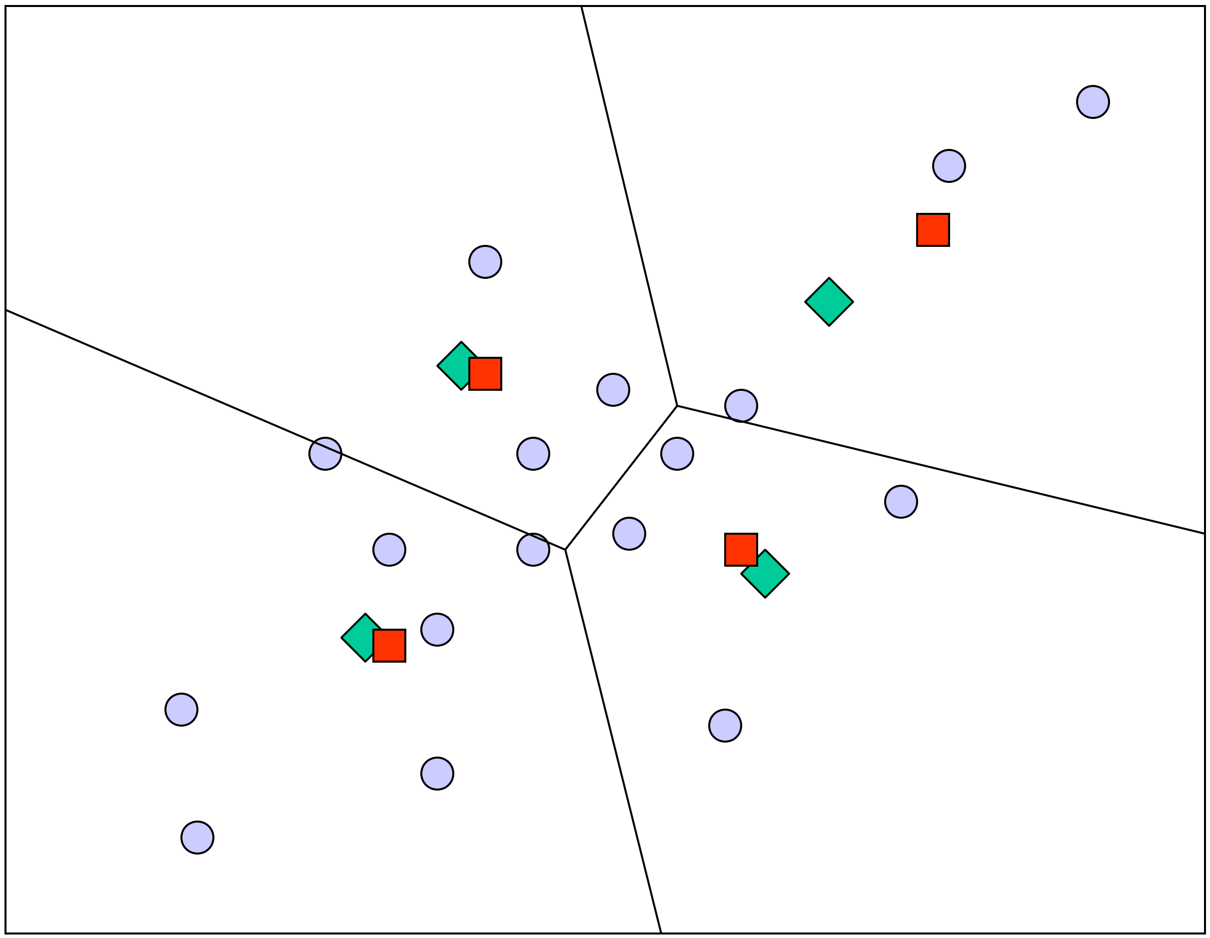


GLA Example (5)

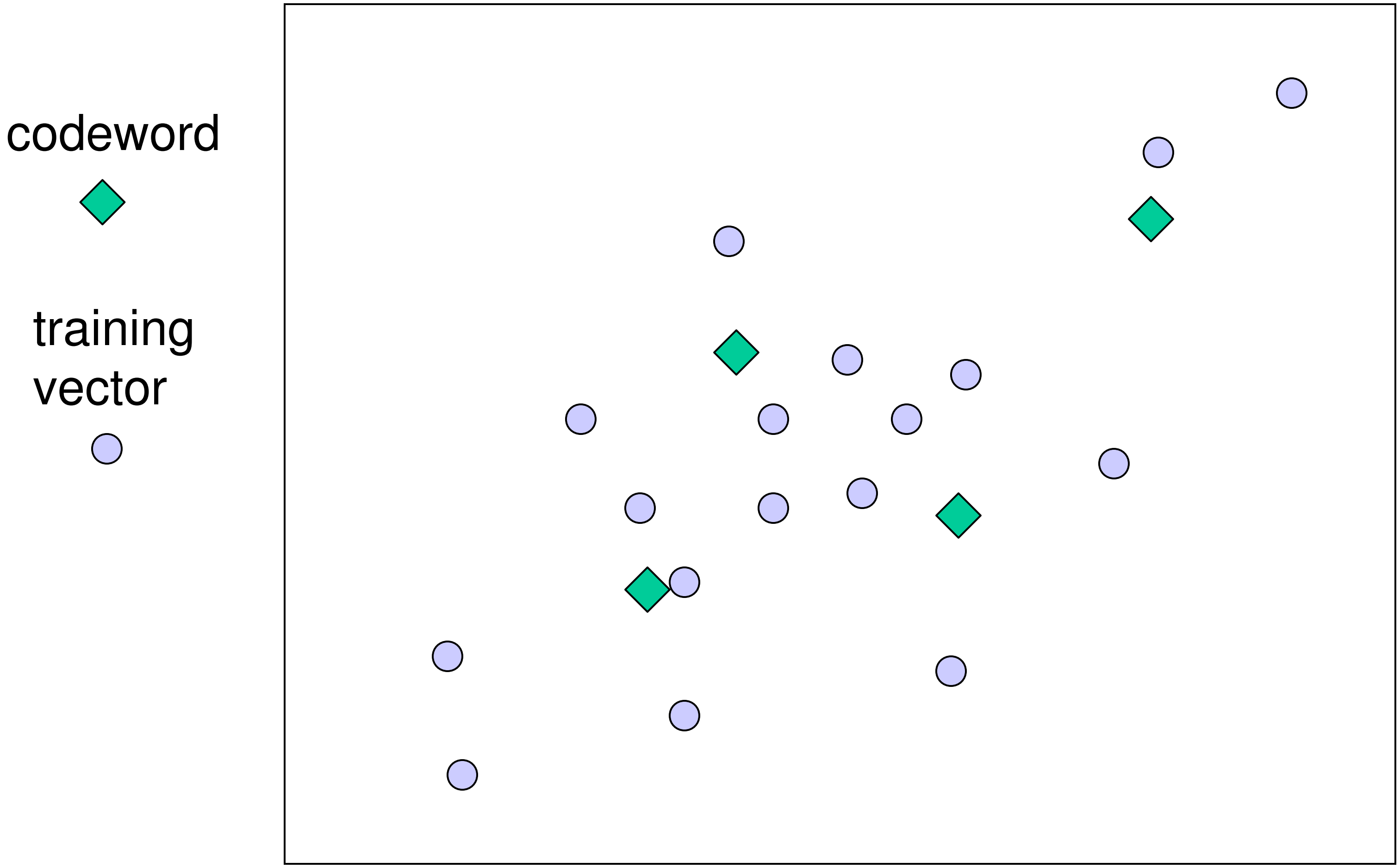


GLA Example (6)

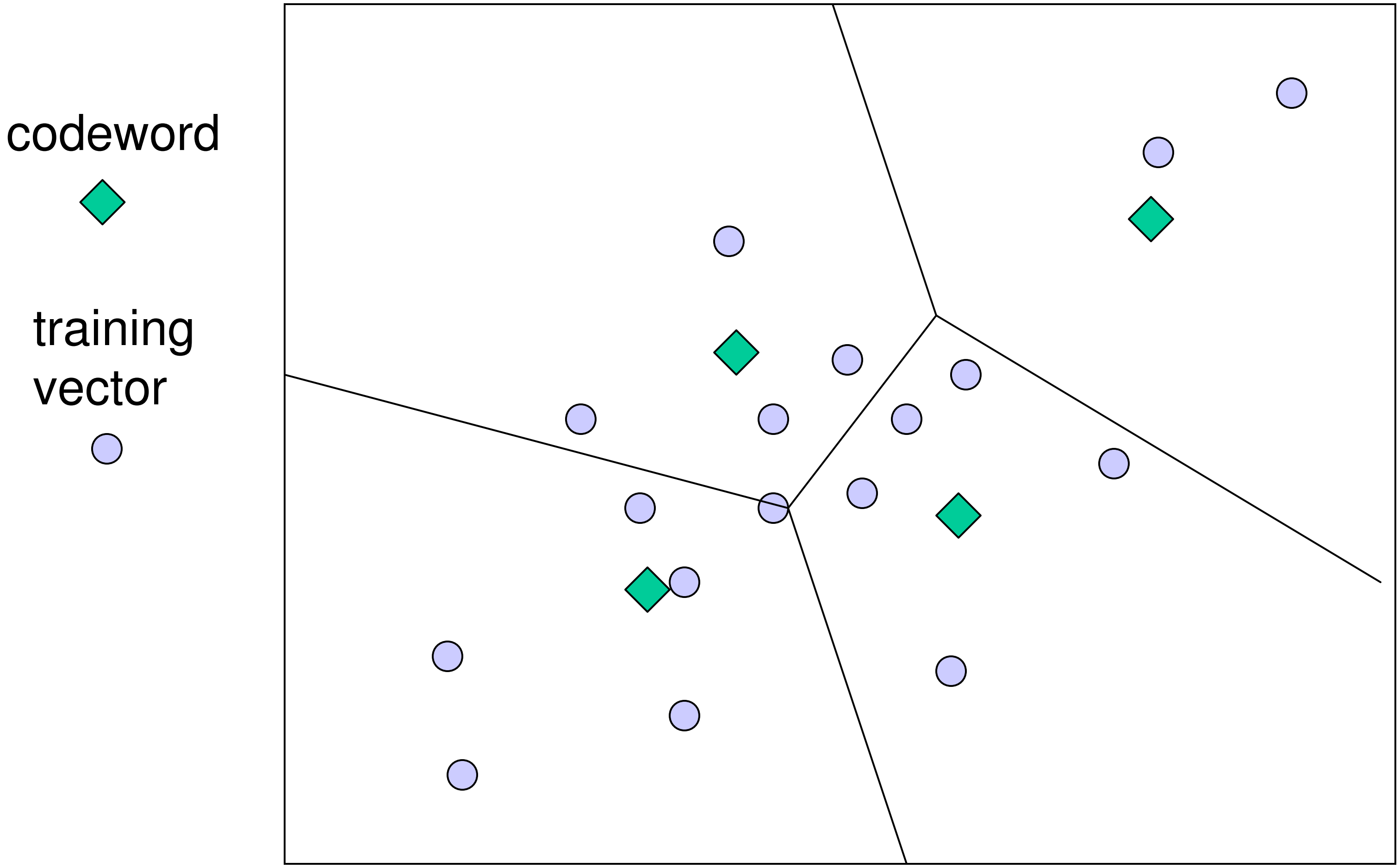




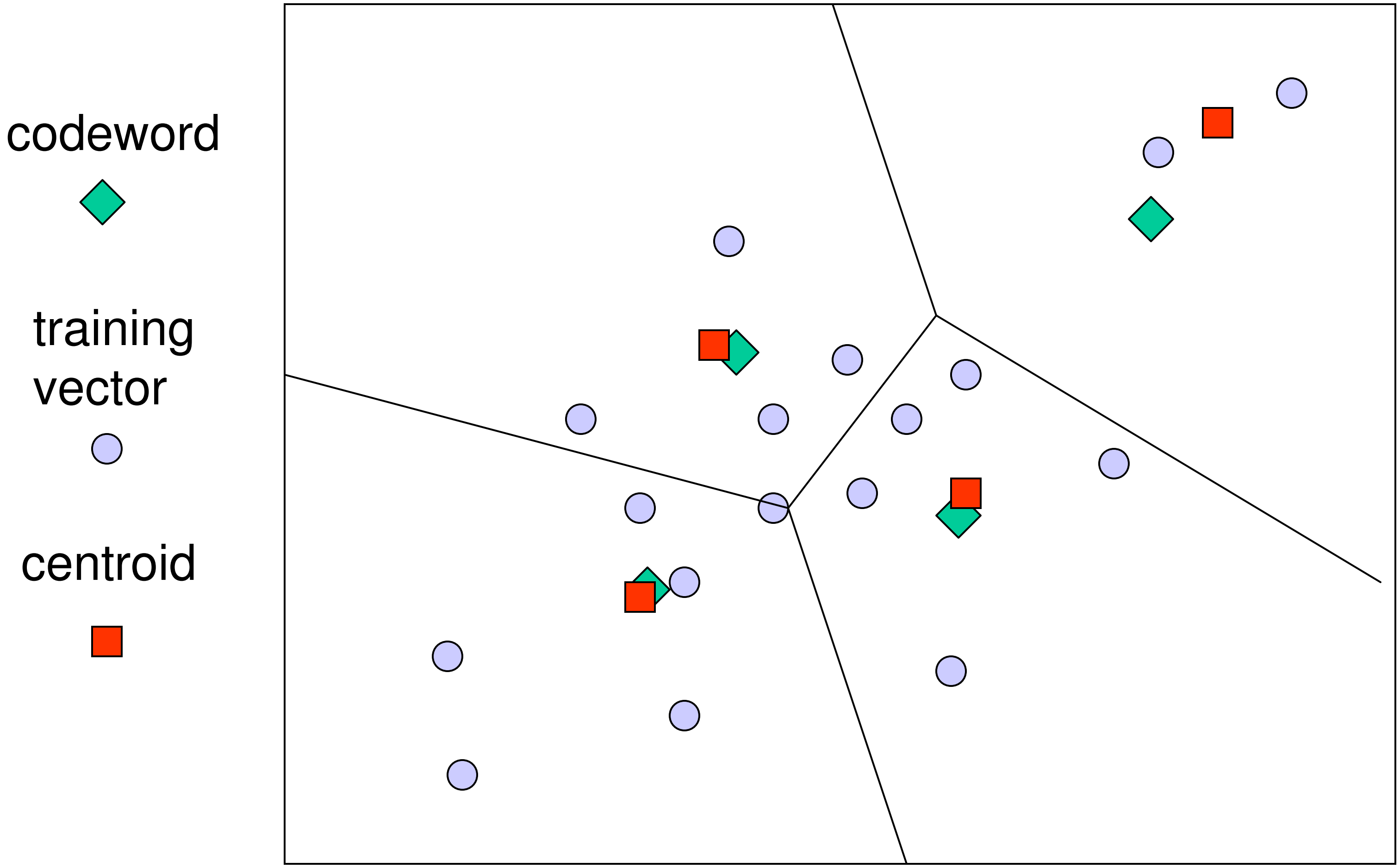
GLA Example (7)



GLA Example (8)



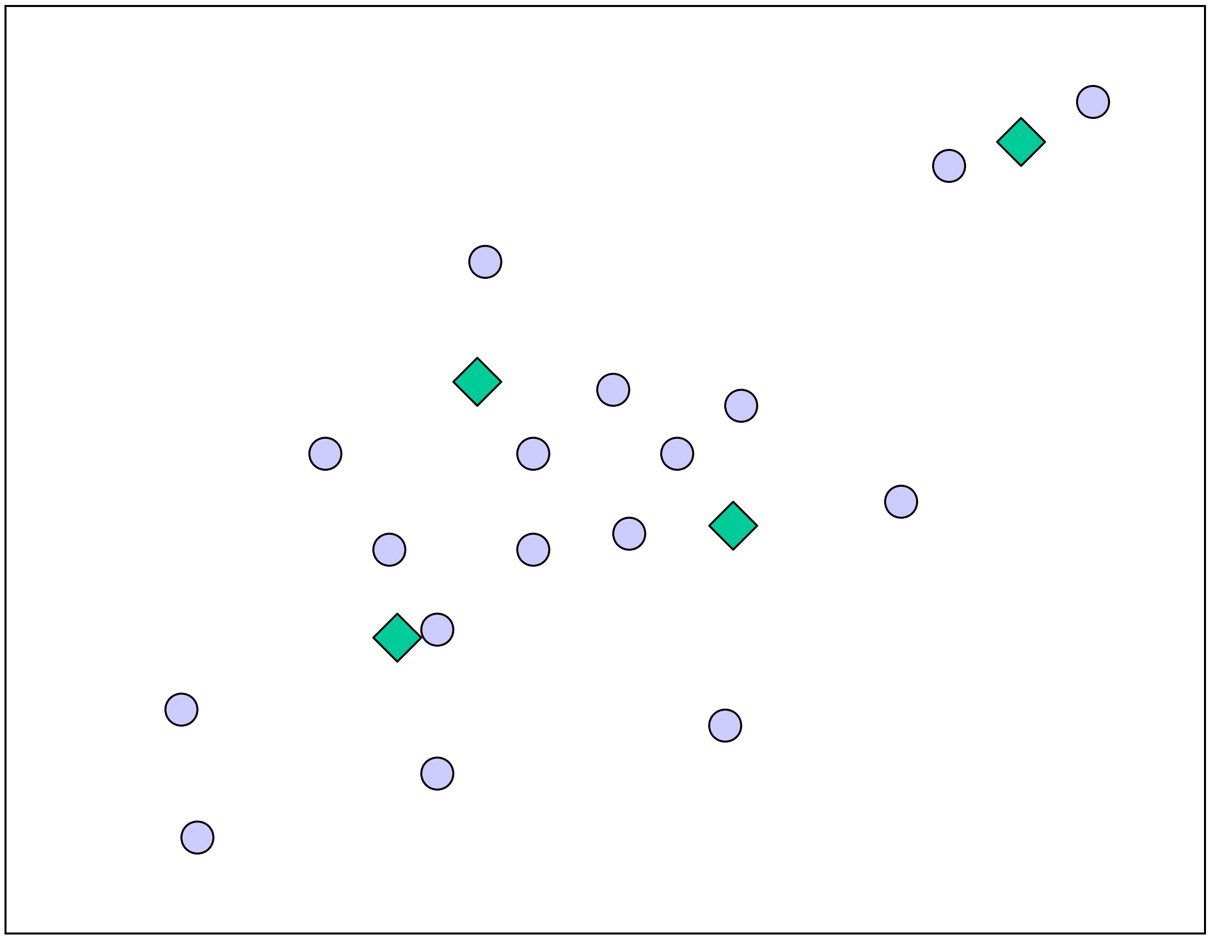
GLA Example (9)



GLA Example (10)

codeword

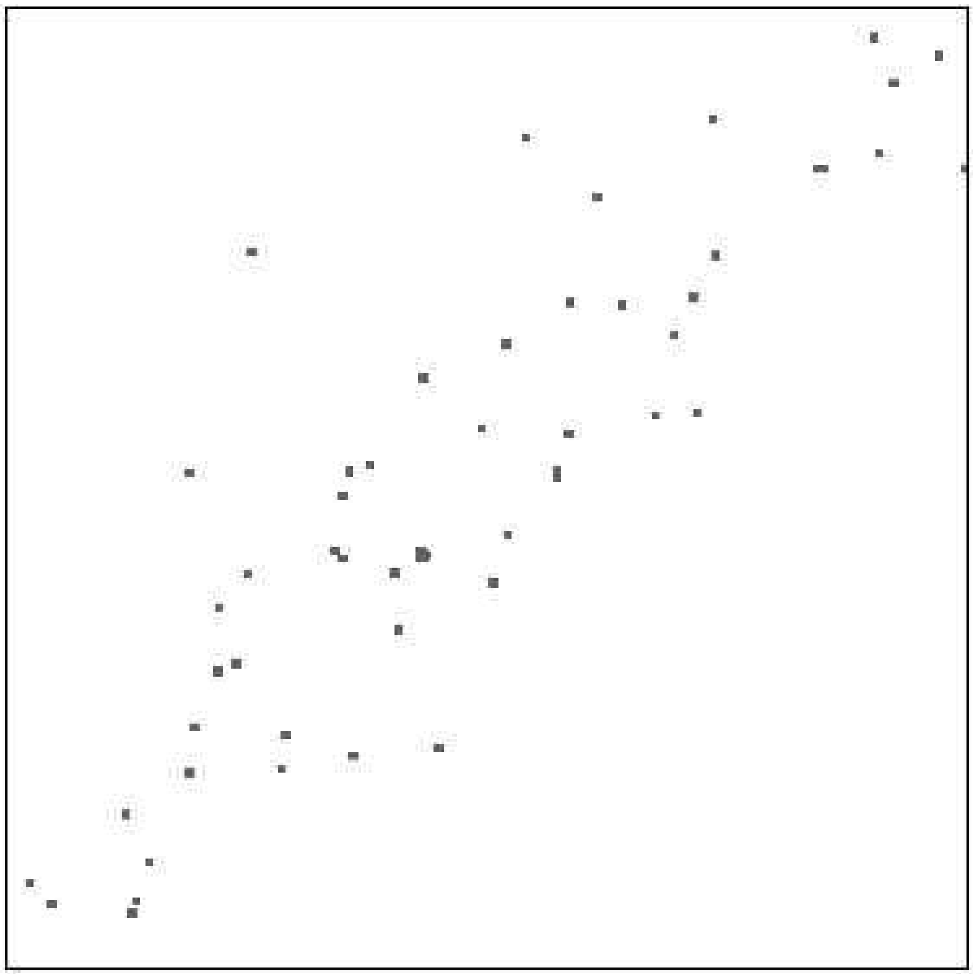
training vector



Codebook

1 x 2 codewords

Note: codewords diagonally spread



Codeword Splitting

- It is possible that a chosen codeword represents no training vectors, that is, X(j) is empty.
  - Splitting is an alternative codebook design algorithm that avoids this problem.
- Basic Idea
  - Select codeword c(j) with the greatest distortion.

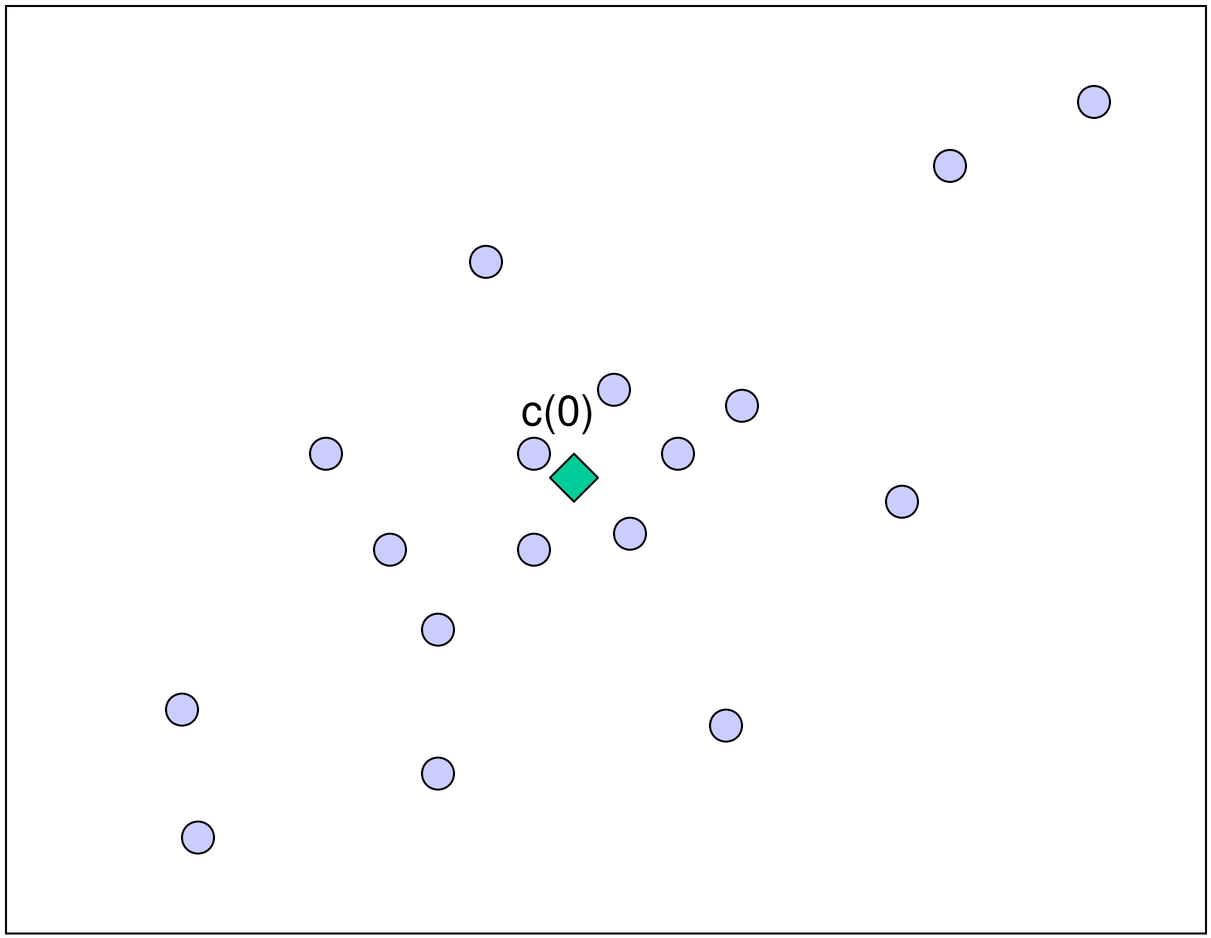
$$D(j) = \sum_{x \in X(j)} \|x - c(j)\|^2$$

  - Split it into two codewords then do the GLA.

Example of Splitting

codeword

training
vector



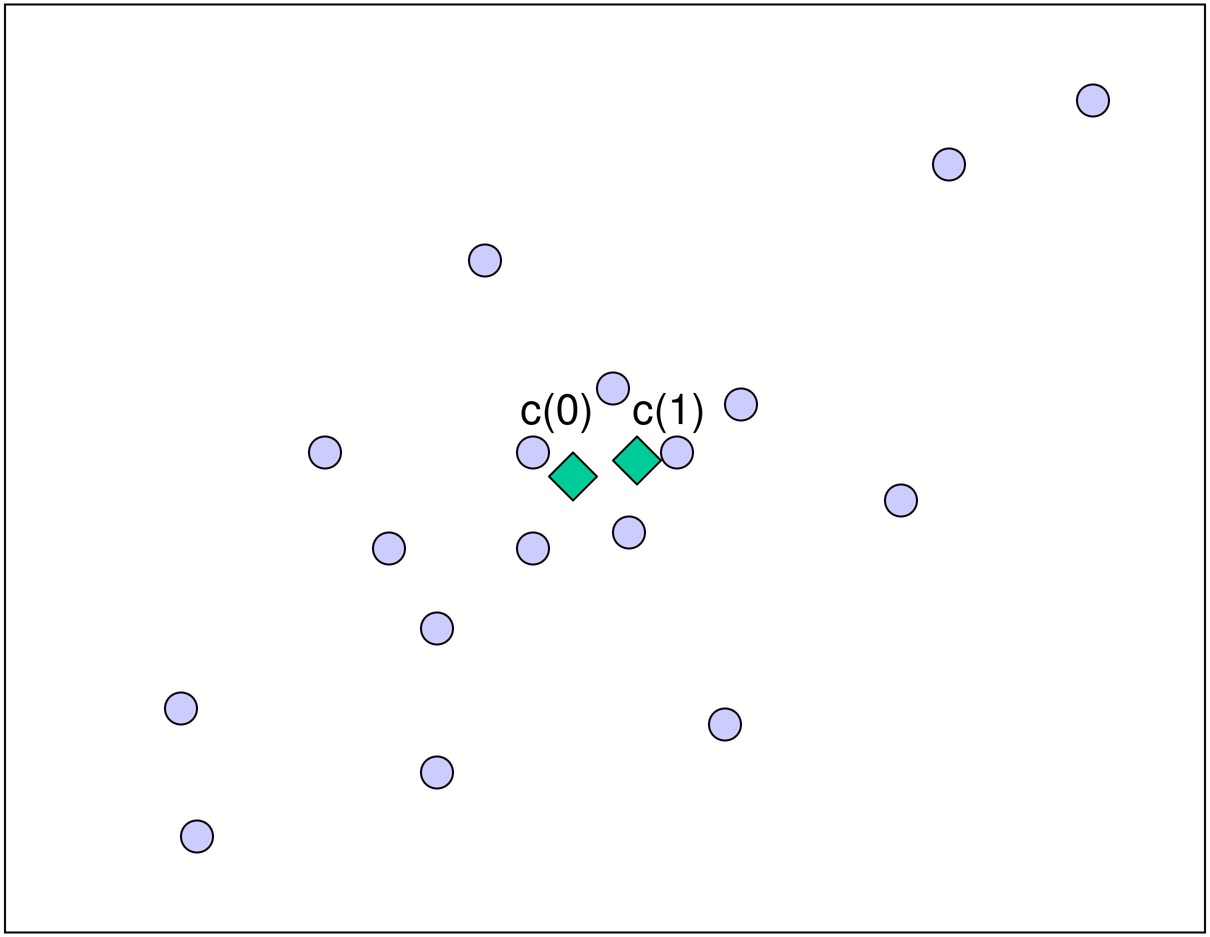
Initially
c(0) is
centroid
of training set

Example of Splitting

codeword

training
vector

Split
$c(1) = c(0) + \varepsilon$

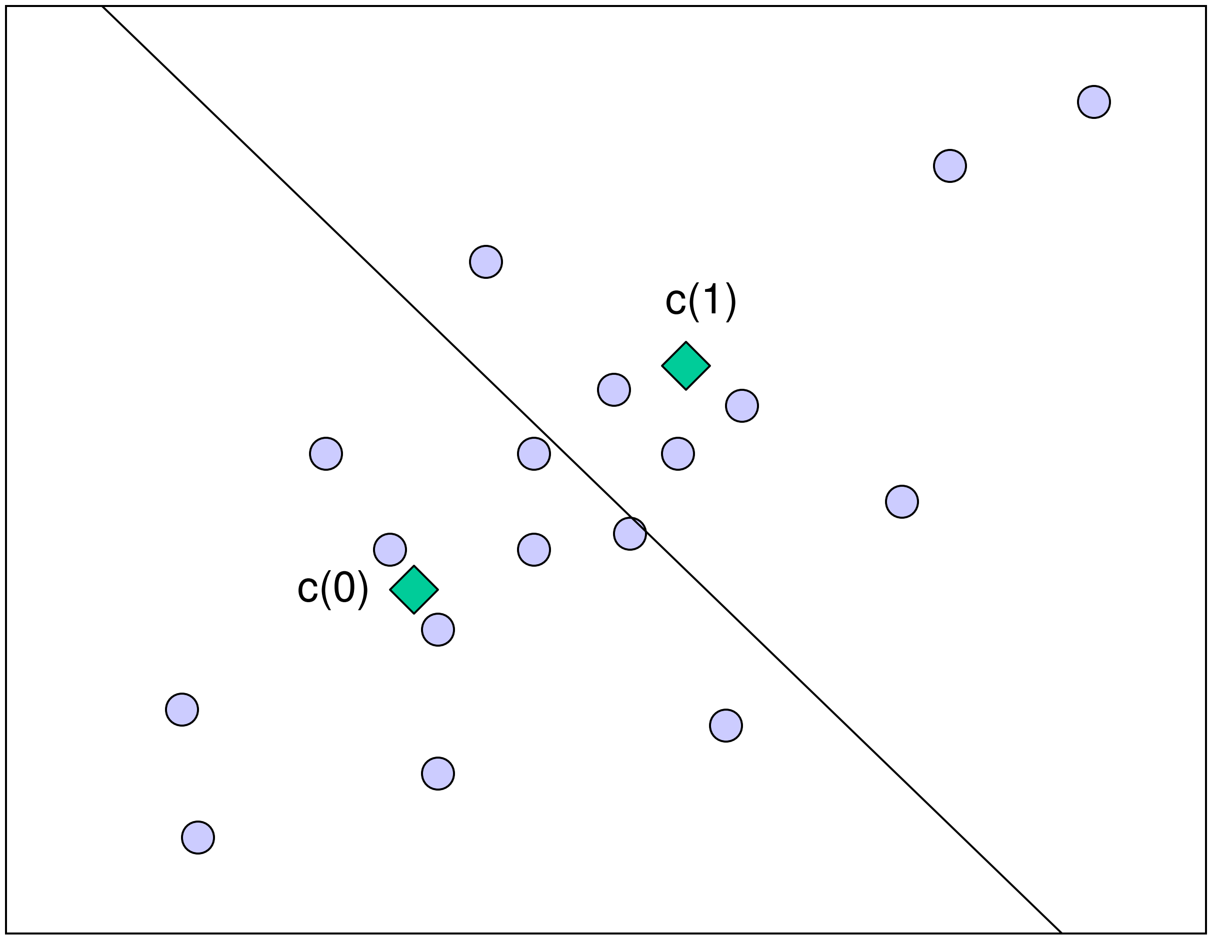


Example of Splitting

codeword

training vector

Apply GLA

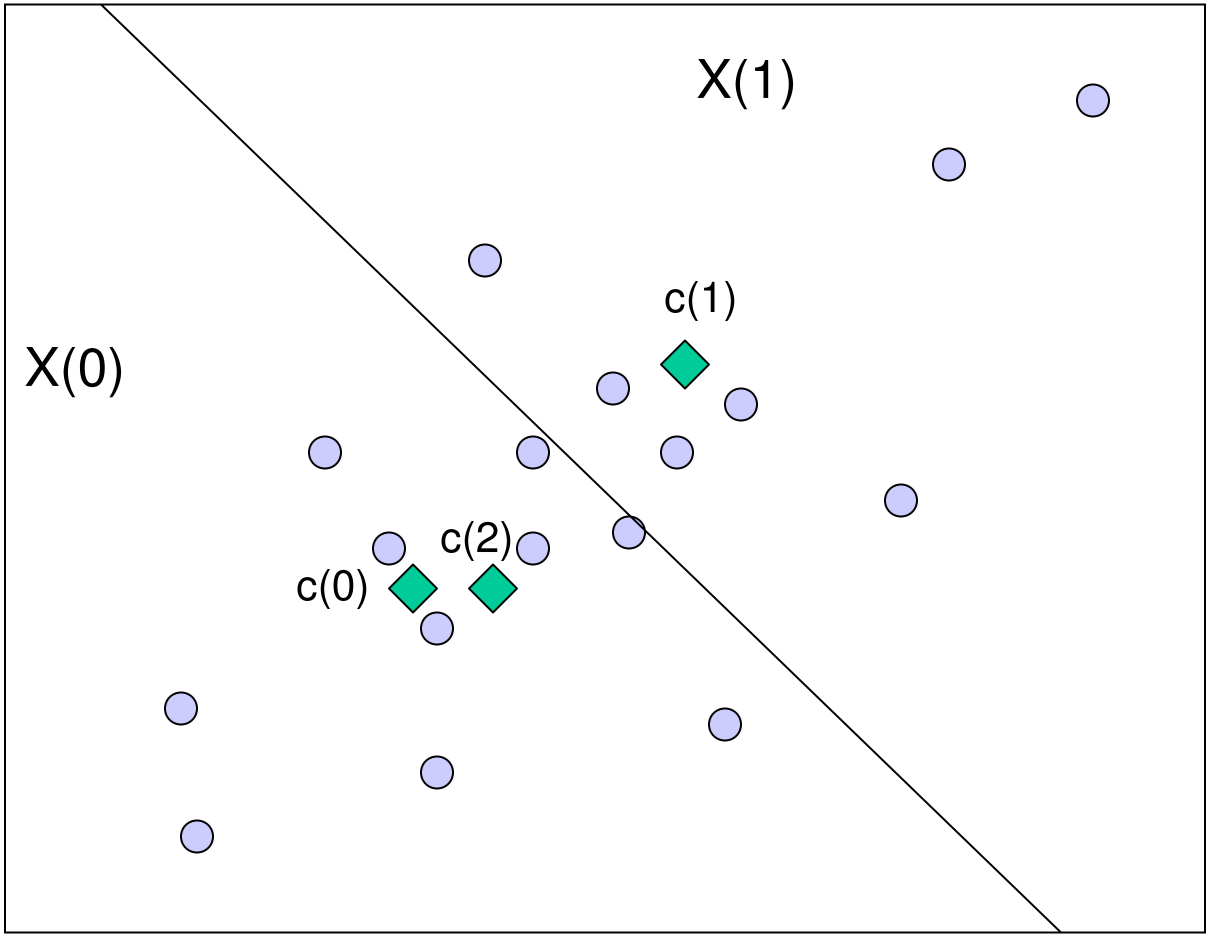


Example of Splitting

codeword

training vector

c(0) has max distortion so split it.

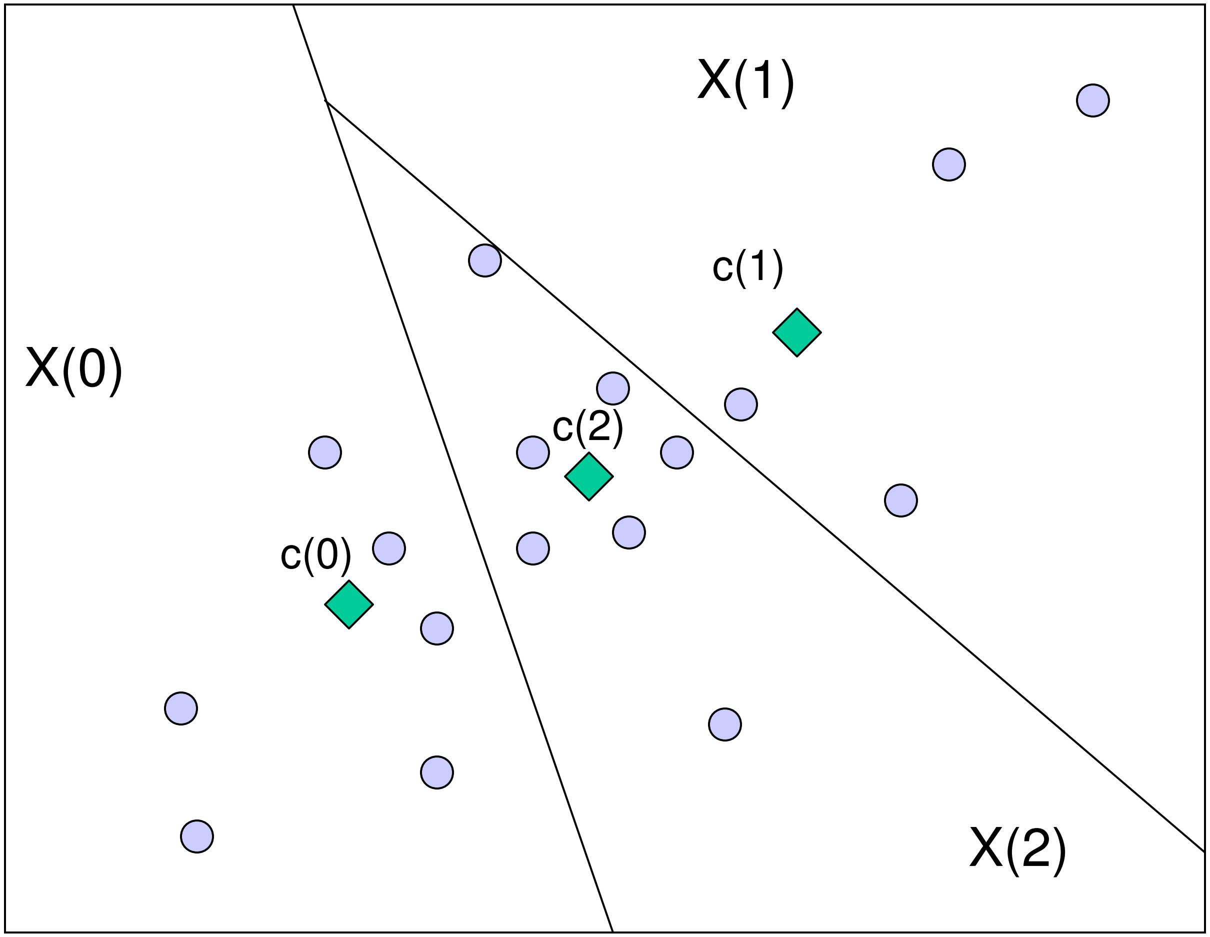


Example of Splitting

codeword

training vector

Apply GLA

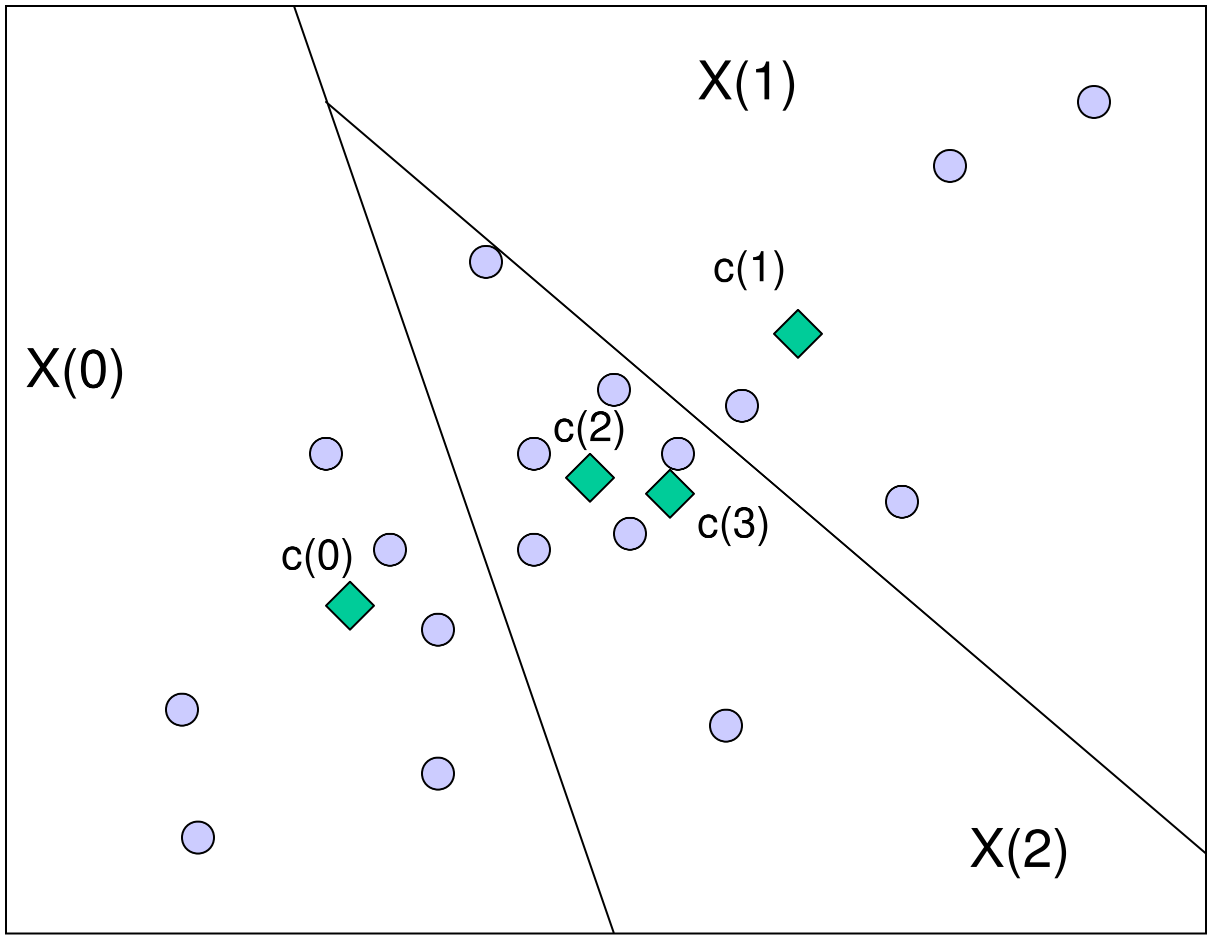


Example of Splitting

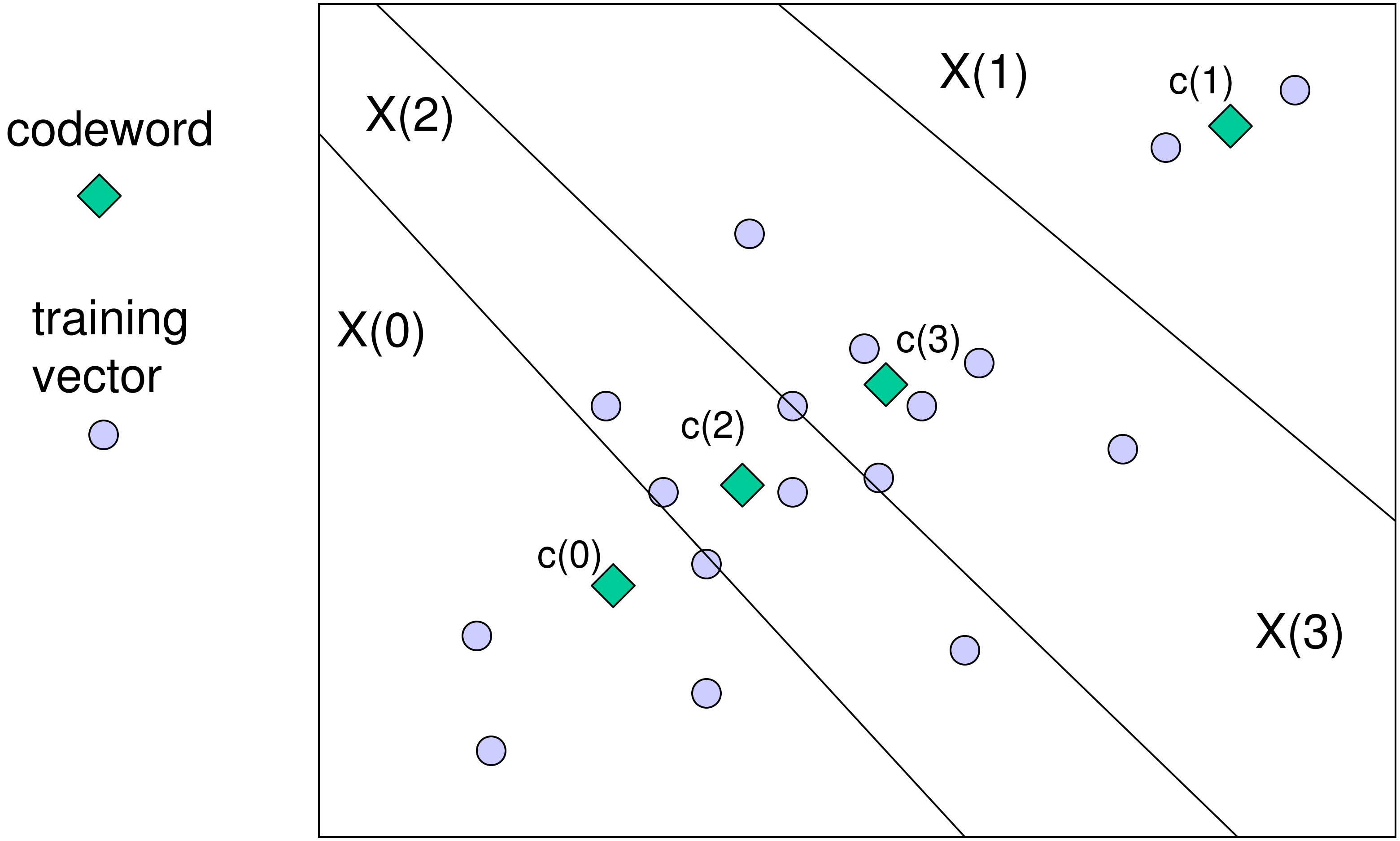
codeword

training vector

c(2) has max distortion so split it



Example of Splitting



GLA Advice

- Time per iteration is dominated by the partitioning step, which is m nearest neighbor searches where m is the training set size.
  - Average time per iteration O(m log n) assuming d is small.
- Training set size.
  - Training set should be at least 20 training vectors per code word to get reasonable performance.
  - Too small a training set results in “over training”.
- Number of iterations can be large.

Encoding

- Naive method.
  - For each input block, search the entire codebook to find the closest codeword.
  - Time O(T n) where n is the size of the codebook and T is the number of blocks in the image.
  - Example: n = 1024, T = 256 x 256 = 65,536 (2 x 2 blocks for a 512 x 512 image)
    nT = 1024 x 65536 = $2^{26} \approx$ 67 million distance calculations.
- Faster methods are known for doing “Full Search VQ”. For example, k-d trees.
  - Time O(T log n)

VQ Encoding is Nearest Neighbor Search

- Given an input vector, find the closest codeword in the codebook and output its index.
- Closest is measured in squared Euclidian distance.
- For two vectors (w_1,x_1,y_1,z_1) and (w_2,x_2,y_2,z_2).

$$\text{Squared Distance} = (w_1 - w_2)^2 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$$

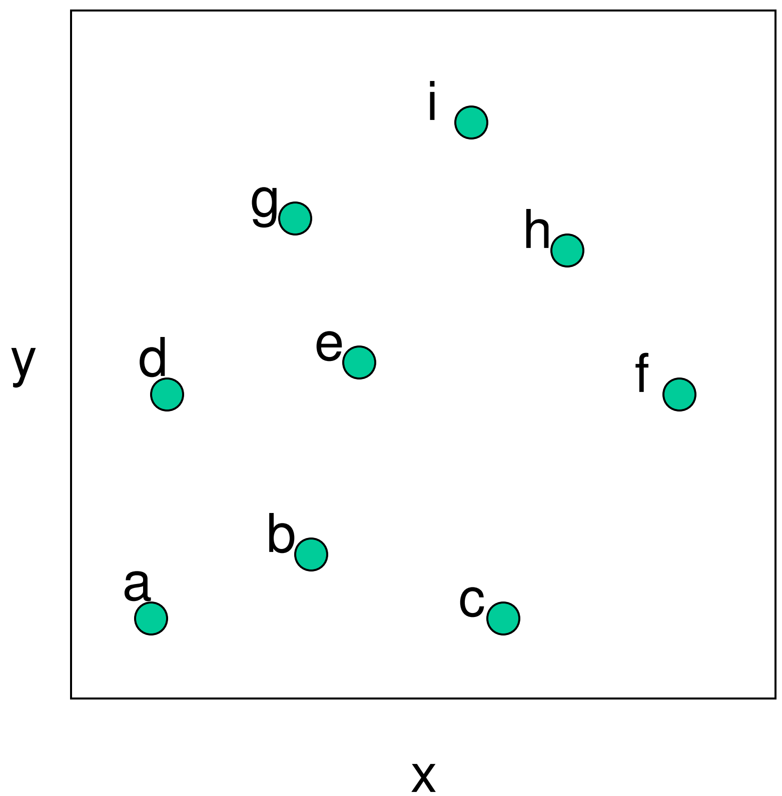
k-d Tree

- Jon Bentley, 1975
- Tree used to store spatial data.
  - Nearest neighbor search.
  - Range queries.
  - Fast look-up
- k-d tree are guaranteed $\log_2 n$ depth where n is the number of points in the set.
  - Traditionally, k-d trees store points in d-dimensional space which are equivalent to vectors in d-dimensional space.

k-d Tree Construction

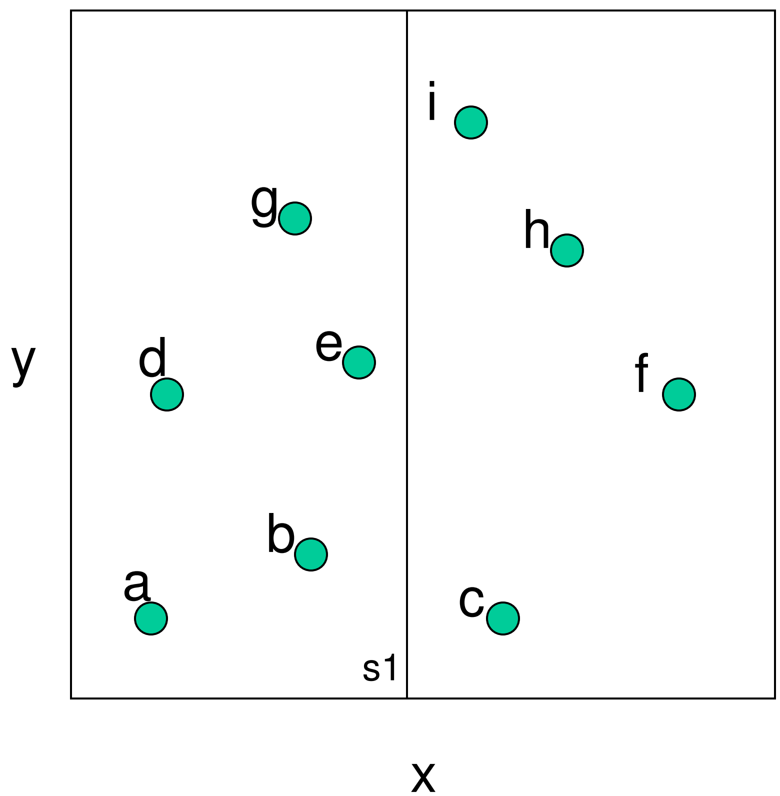
- If there is just one point, form a leaf with that point.
- Otherwise, divide the points in half by a line perpendicular to one of the axes.
- Recursively construct k-d trees for the two sets of points.
- Division strategies
  - divide points perpendicular to the axis with widest spread.
  - divide in a round-robin fashion.

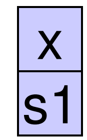
k-d Tree Construction (1)



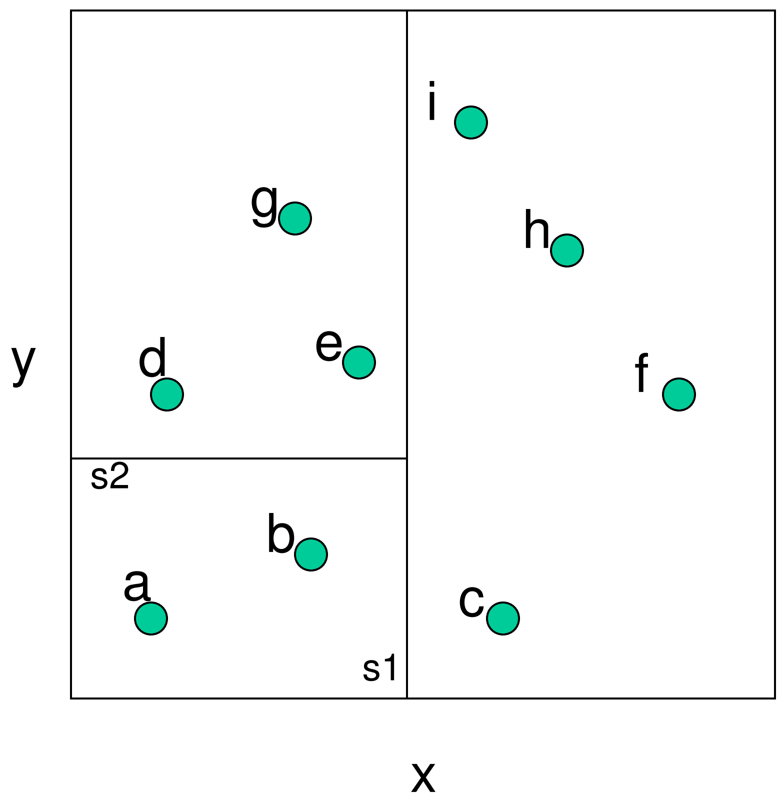
divide perpendicular to the widest spread.

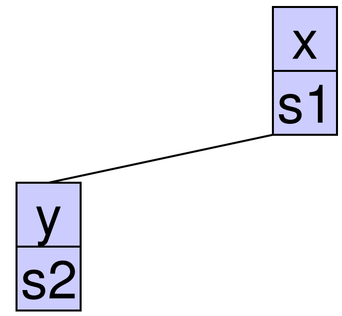
k-d Tree Construction (2)



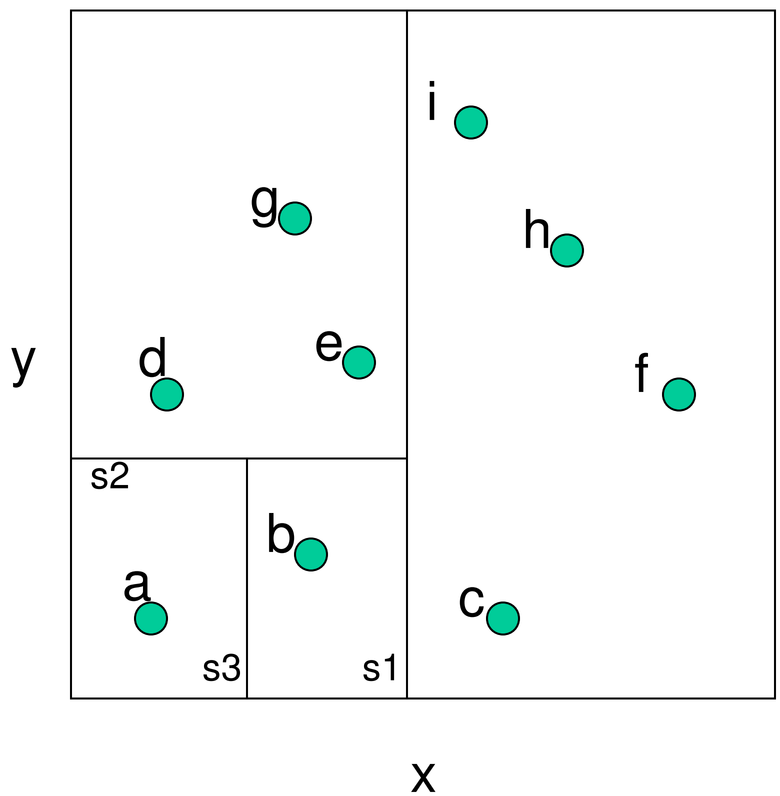


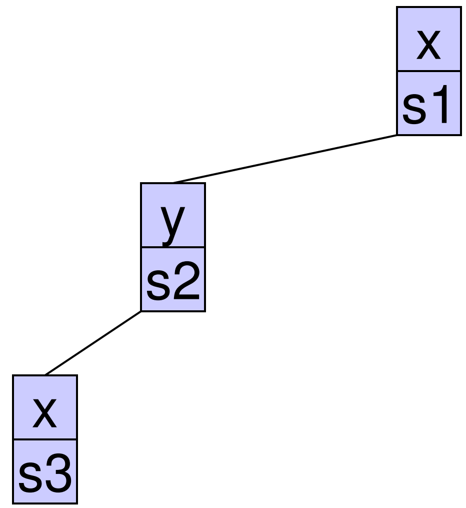
k-d Tree Construction (3)



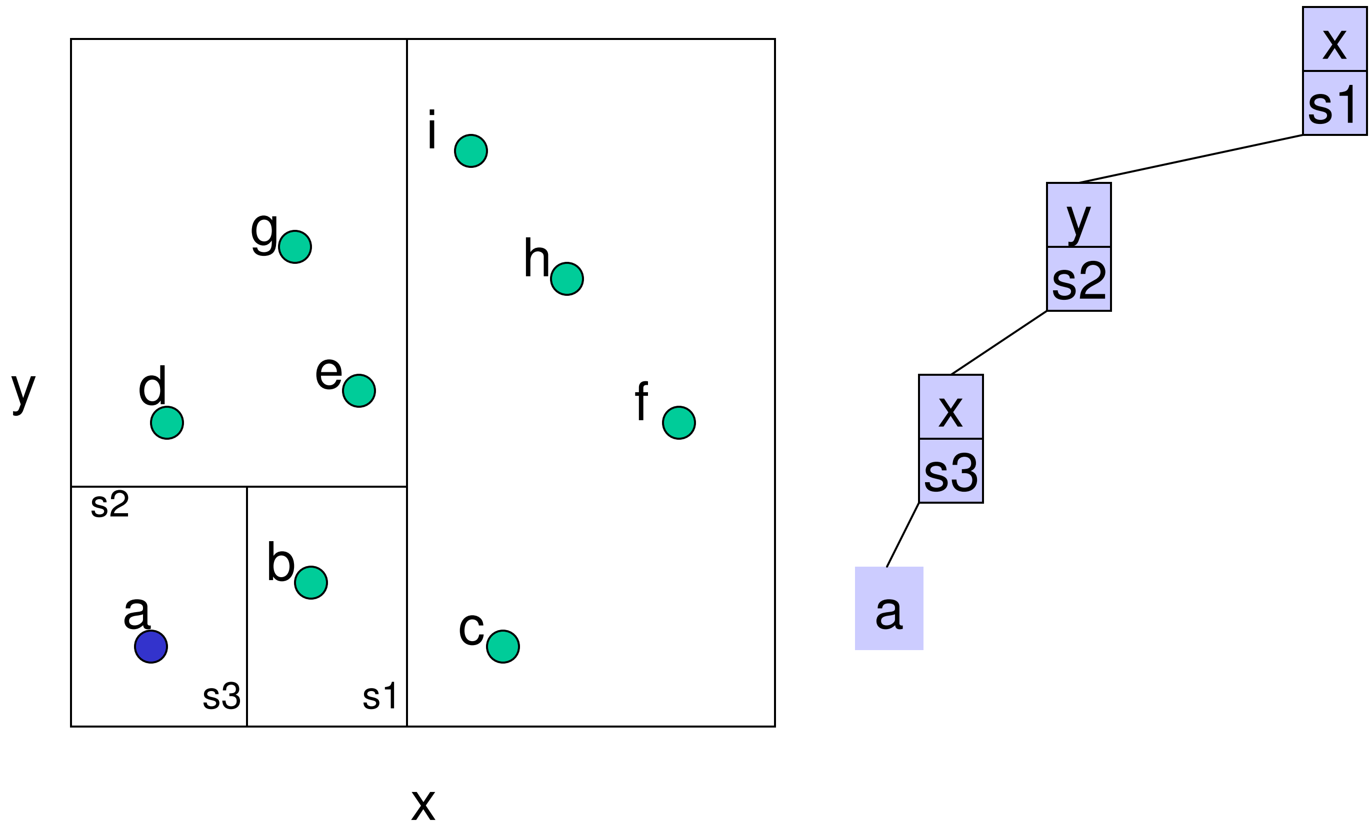


k-d Tree Construction (4)

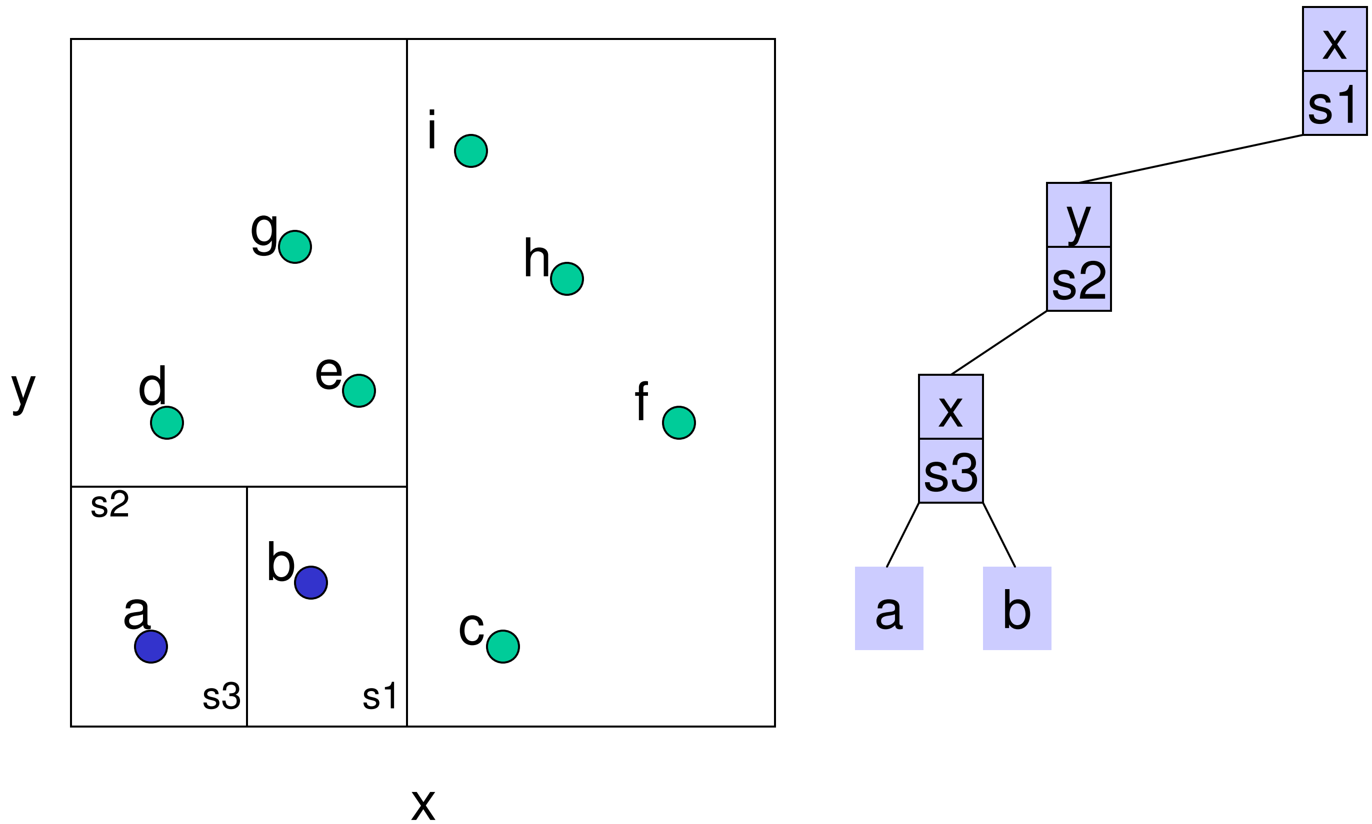




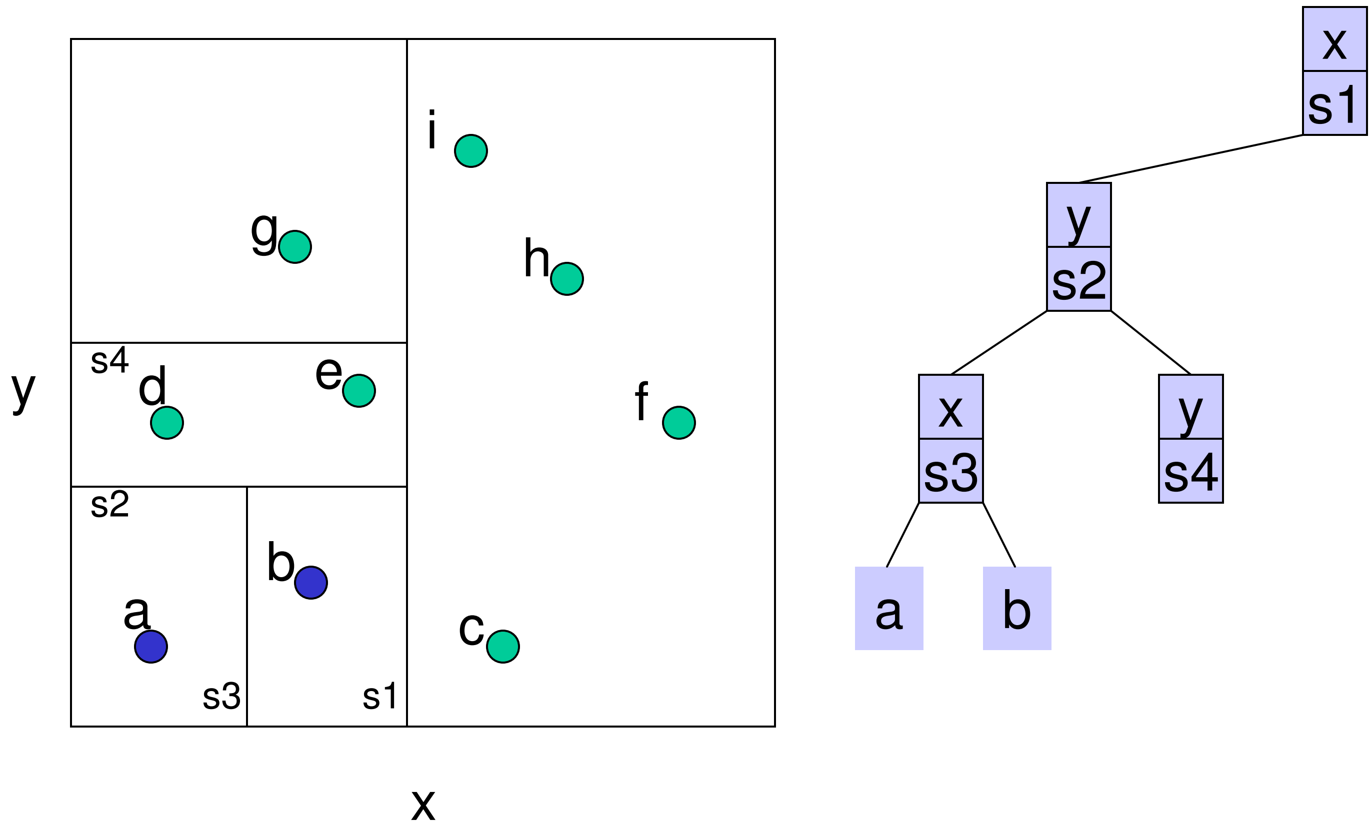
k-d Tree Construction (5)



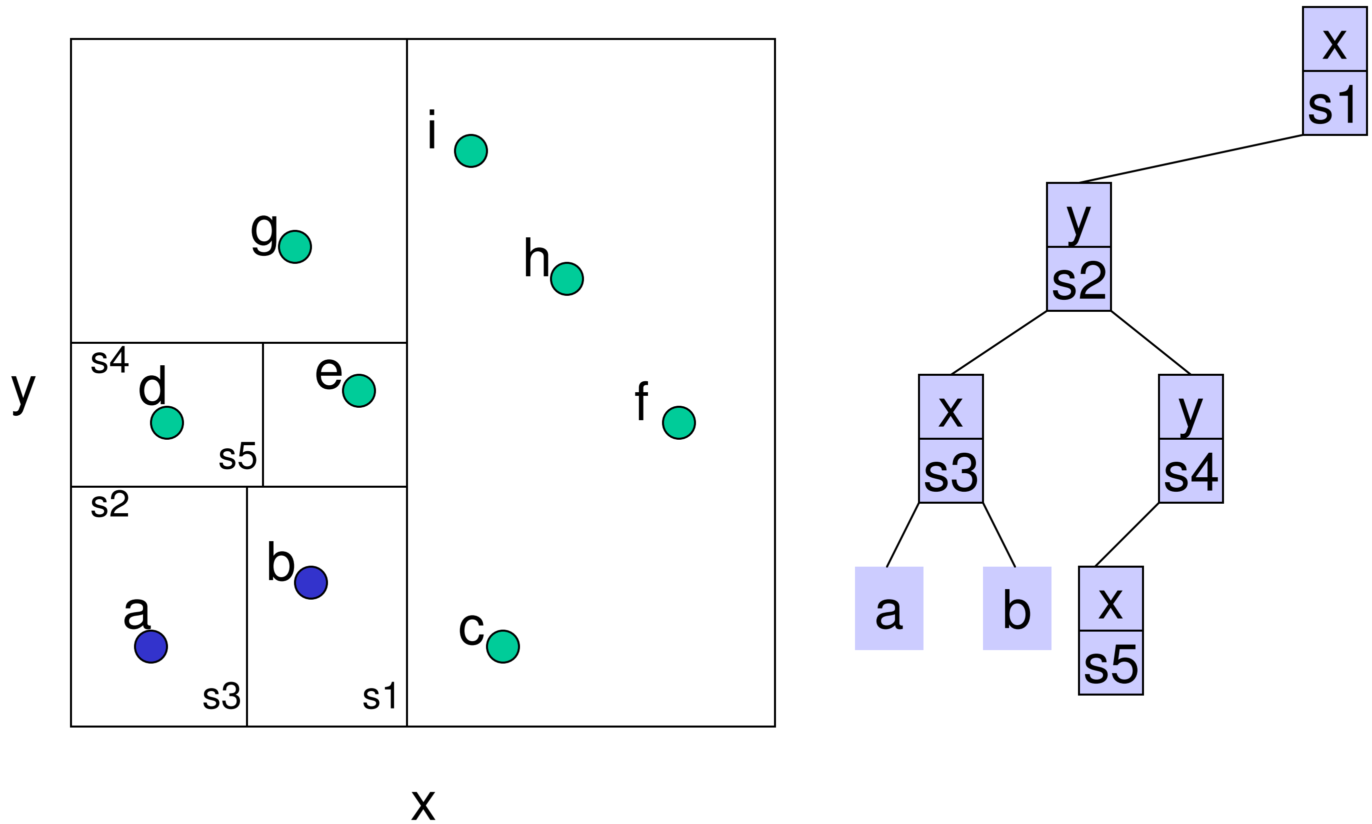
k-d Tree Construction (6)



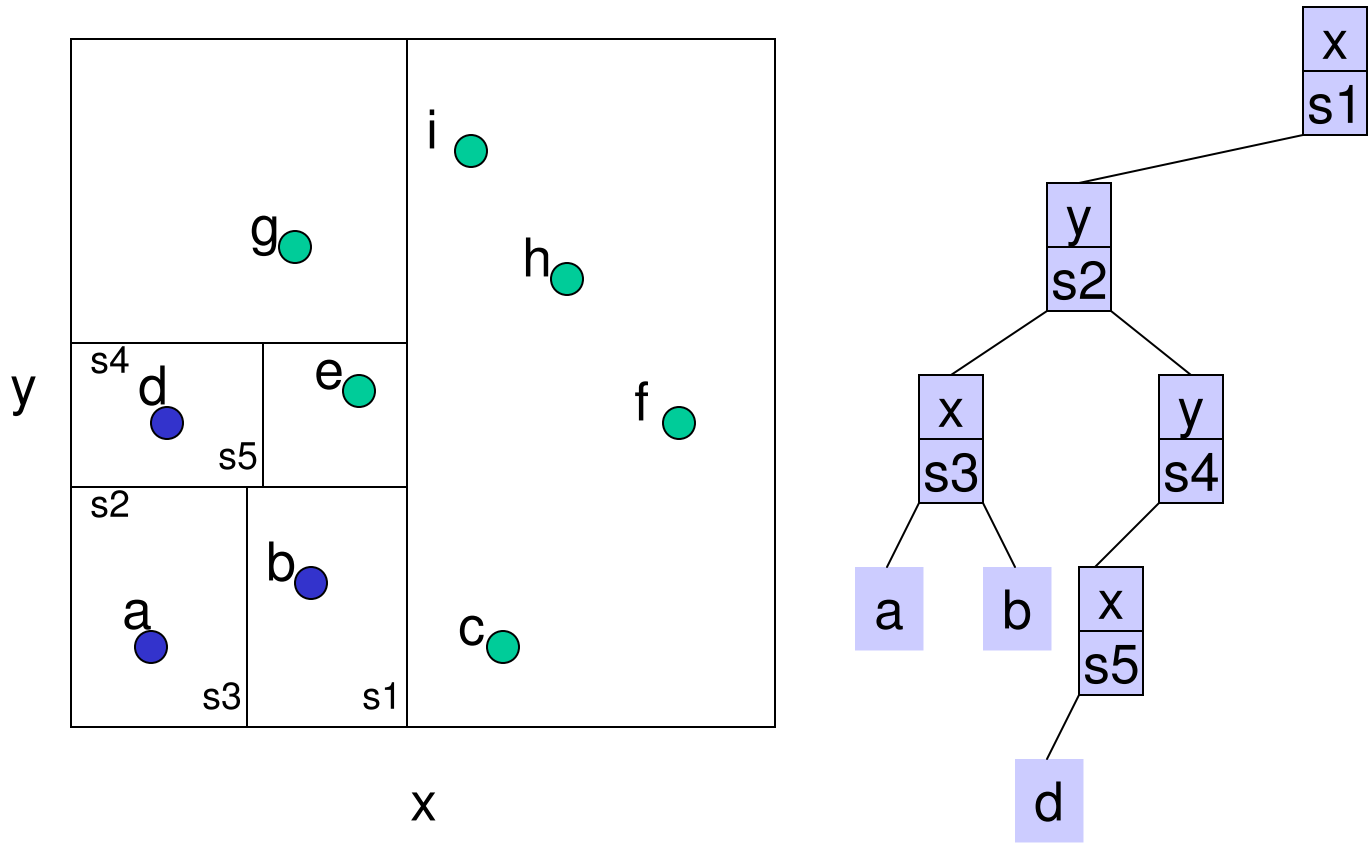
k-d Tree Construction (7)



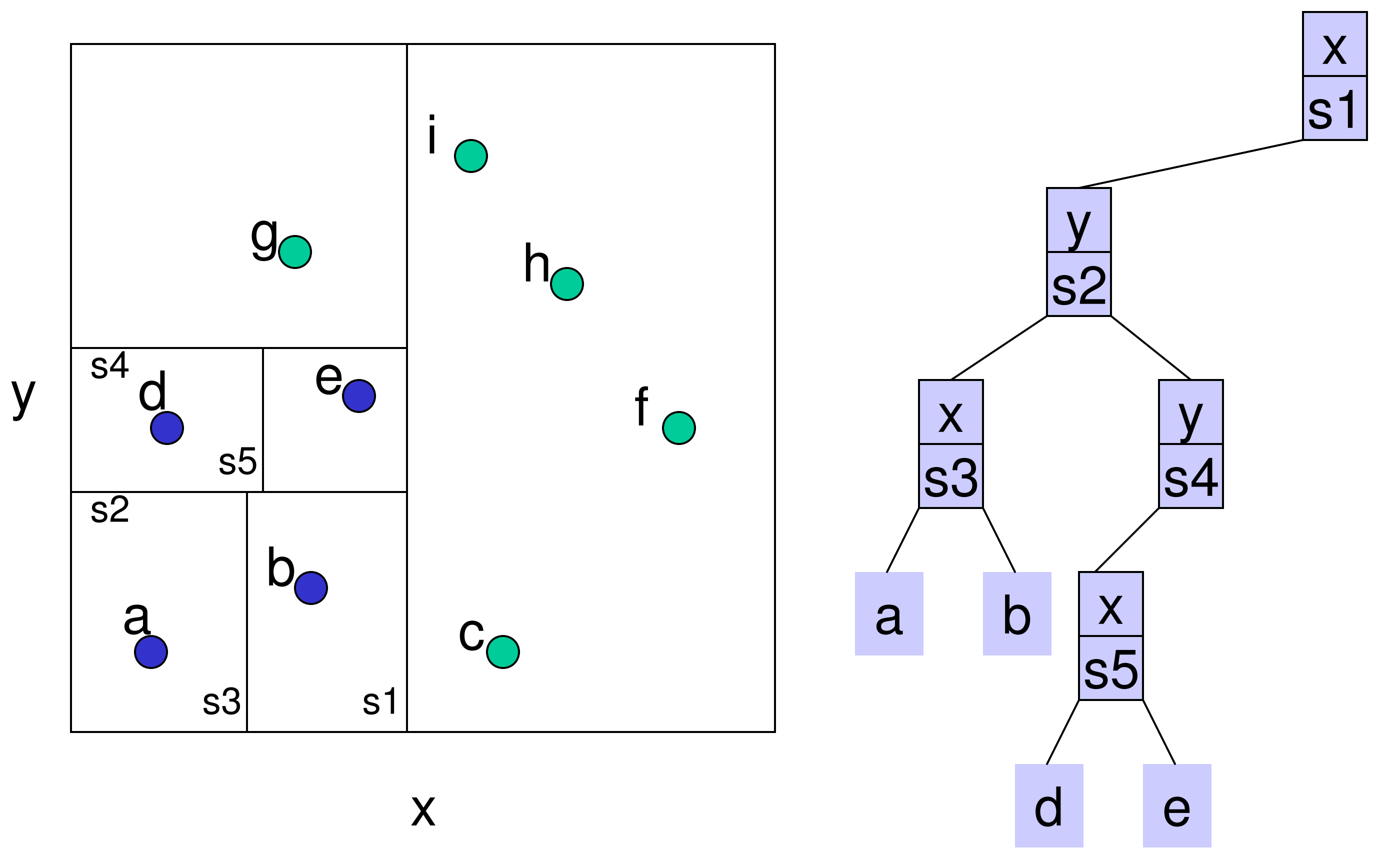
k-d Tree Construction (8)



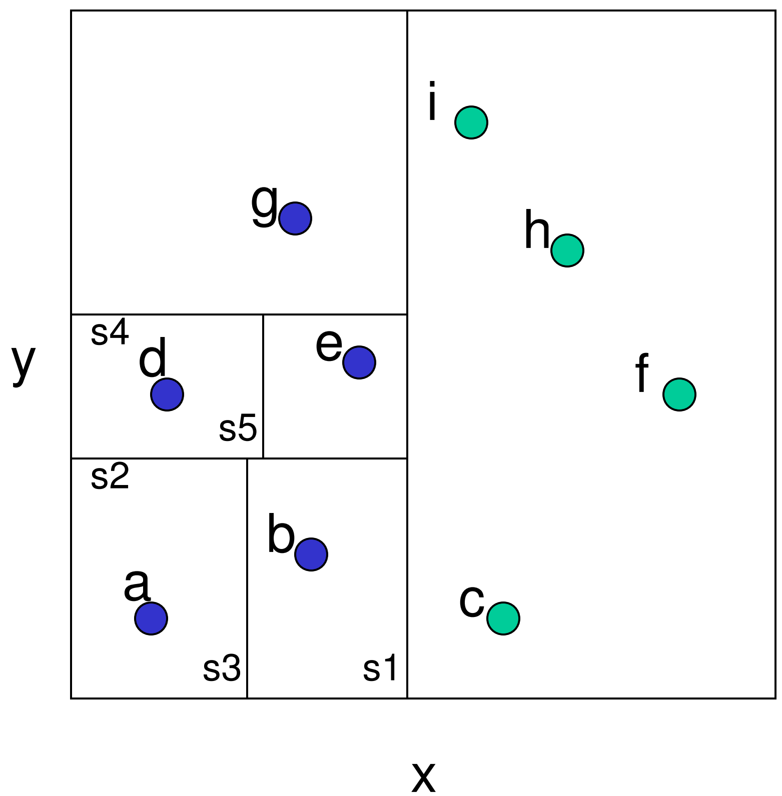
k-d Tree Construction (9)

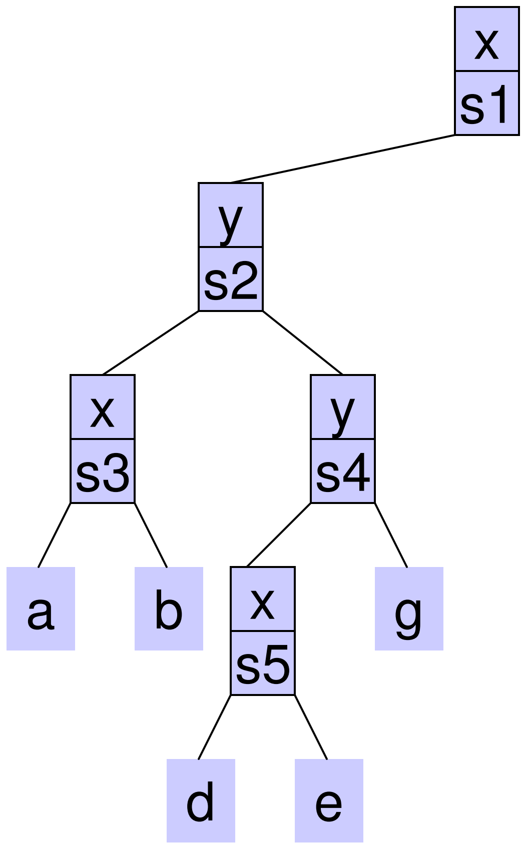


k-d Tree Construction (10)

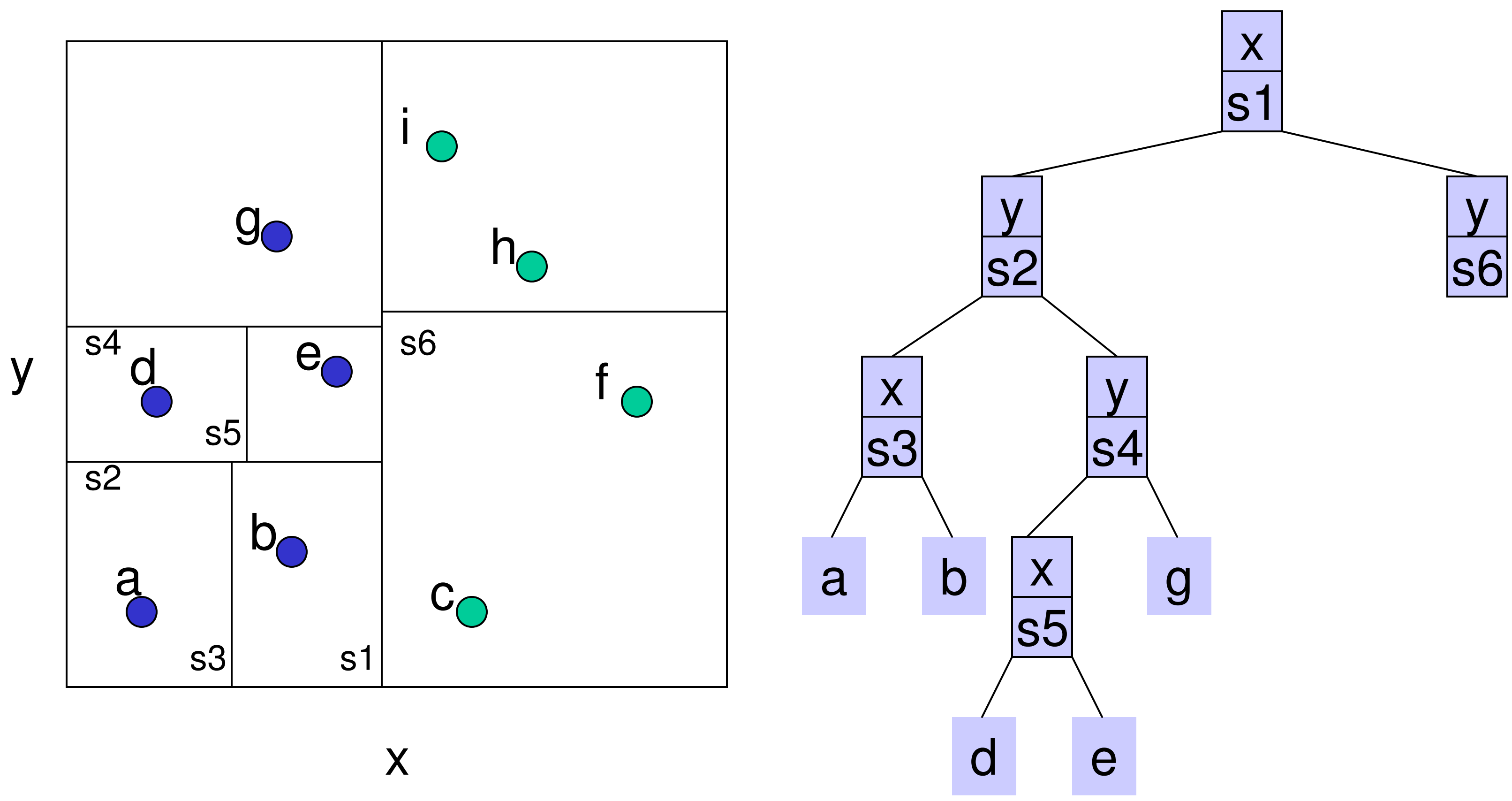


k-d Tree Construction (11)

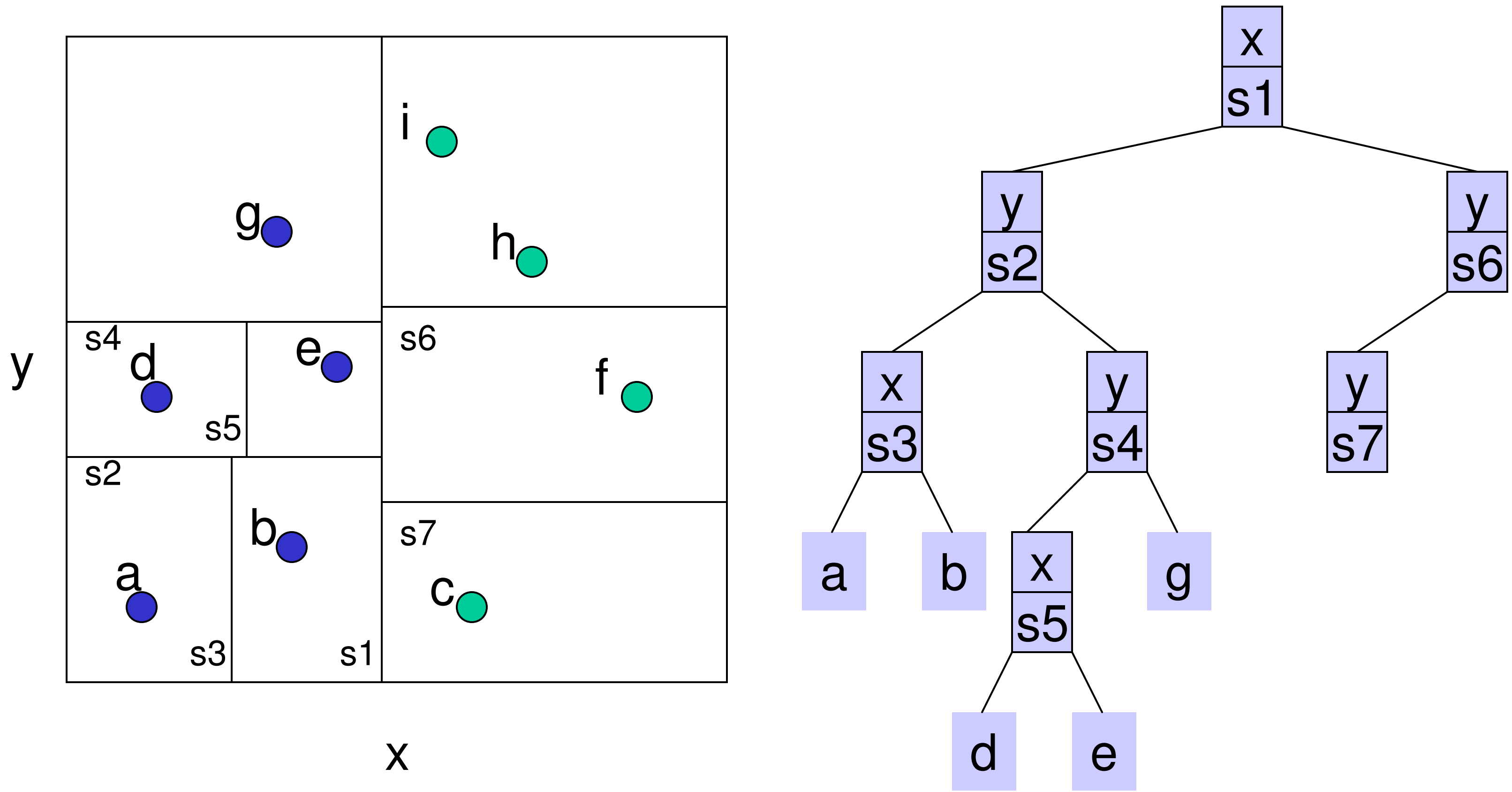




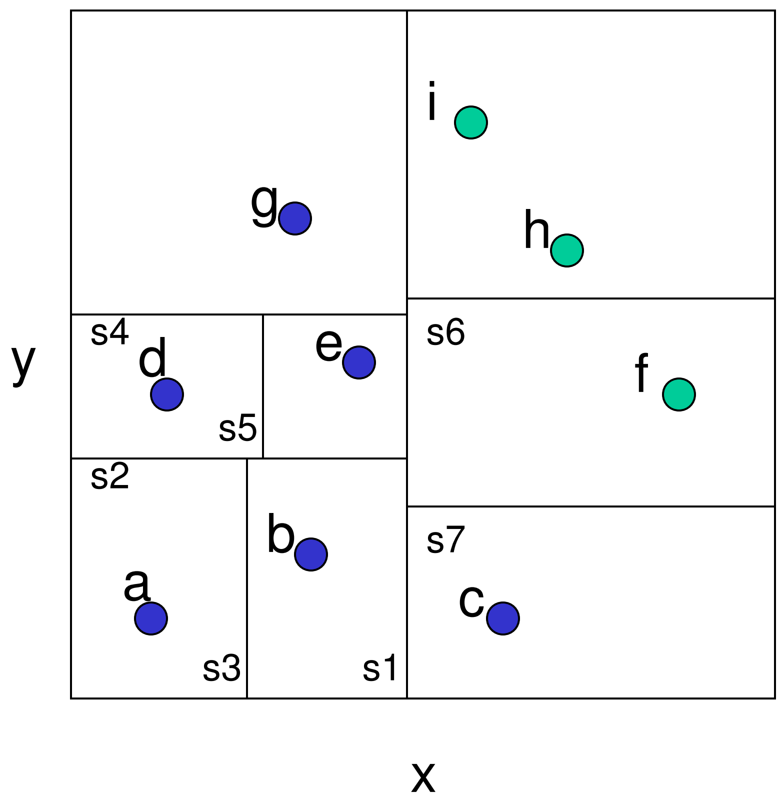
k-d Tree Construction (12)

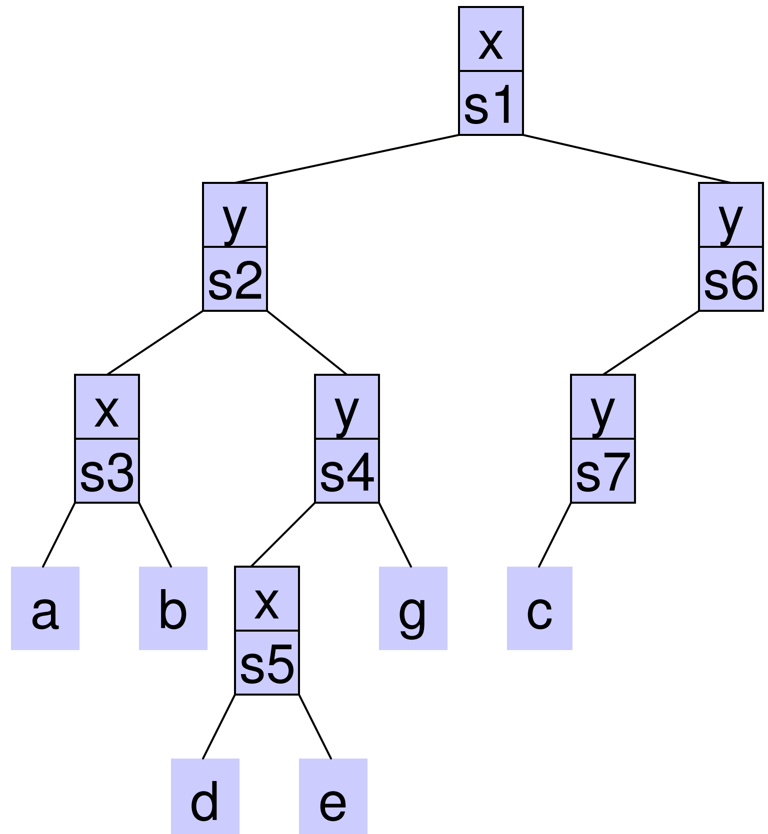


k-d Tree Construction (13)

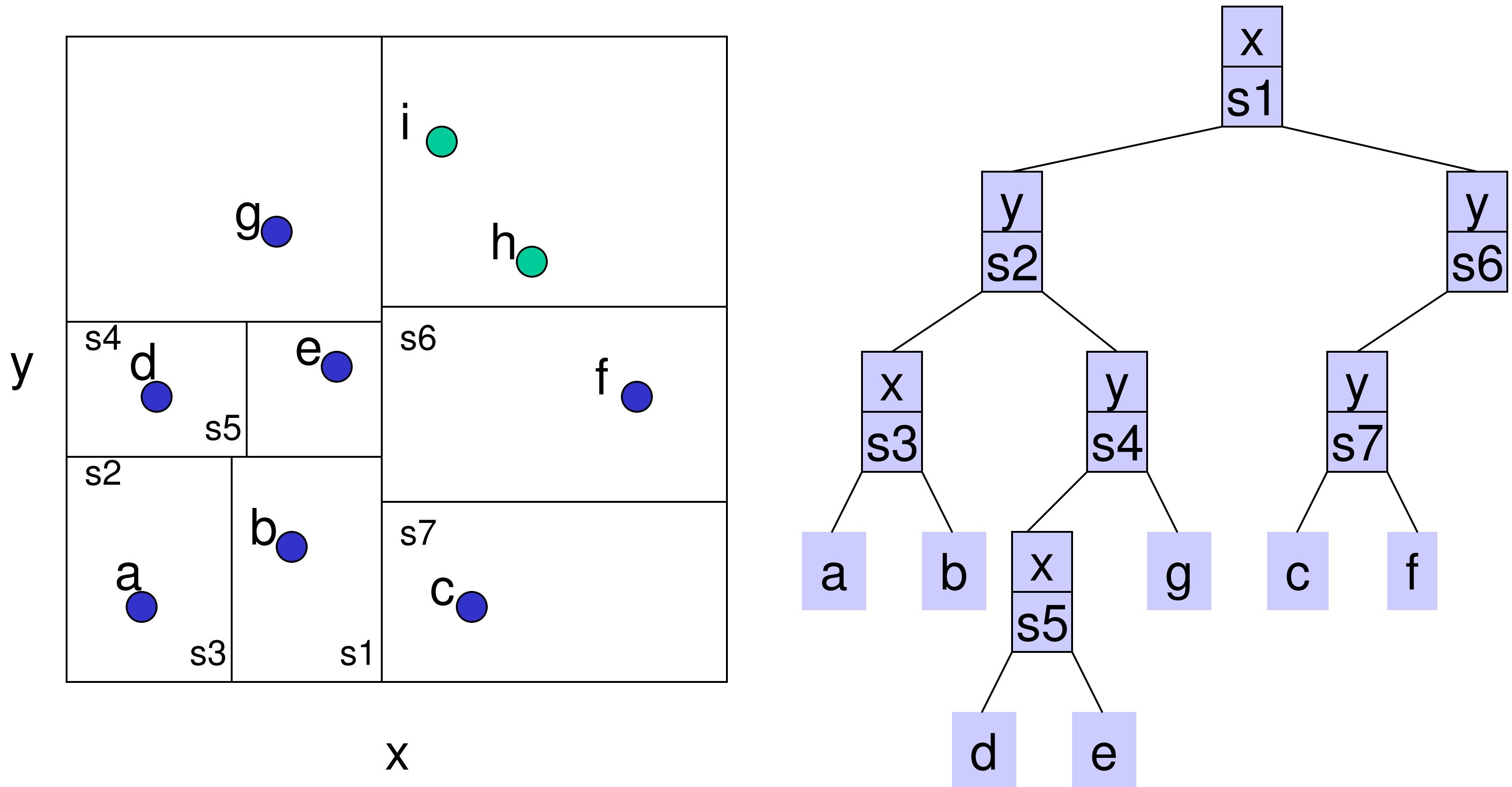


k-d Tree Construction (14)

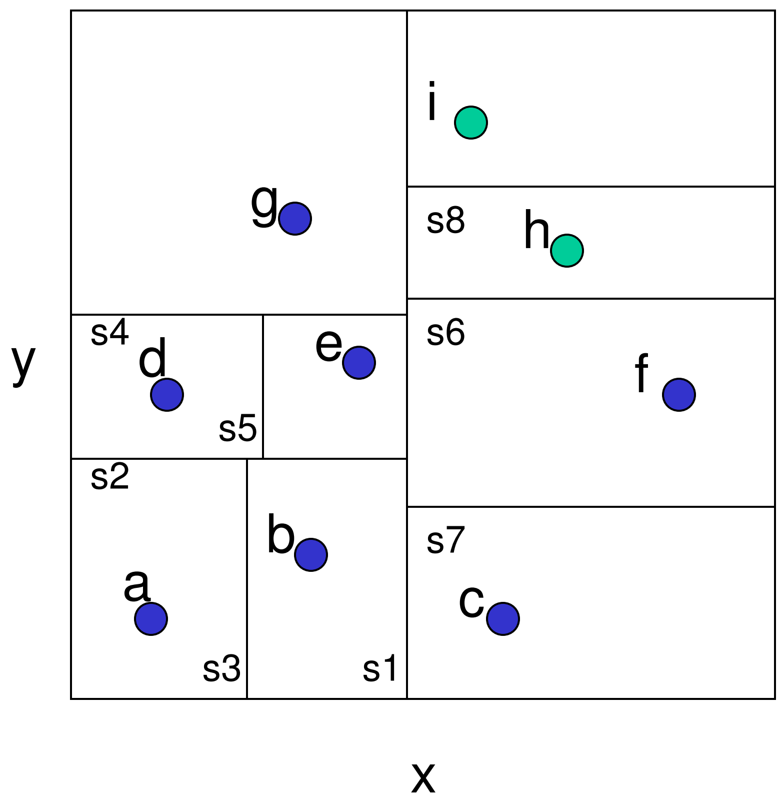


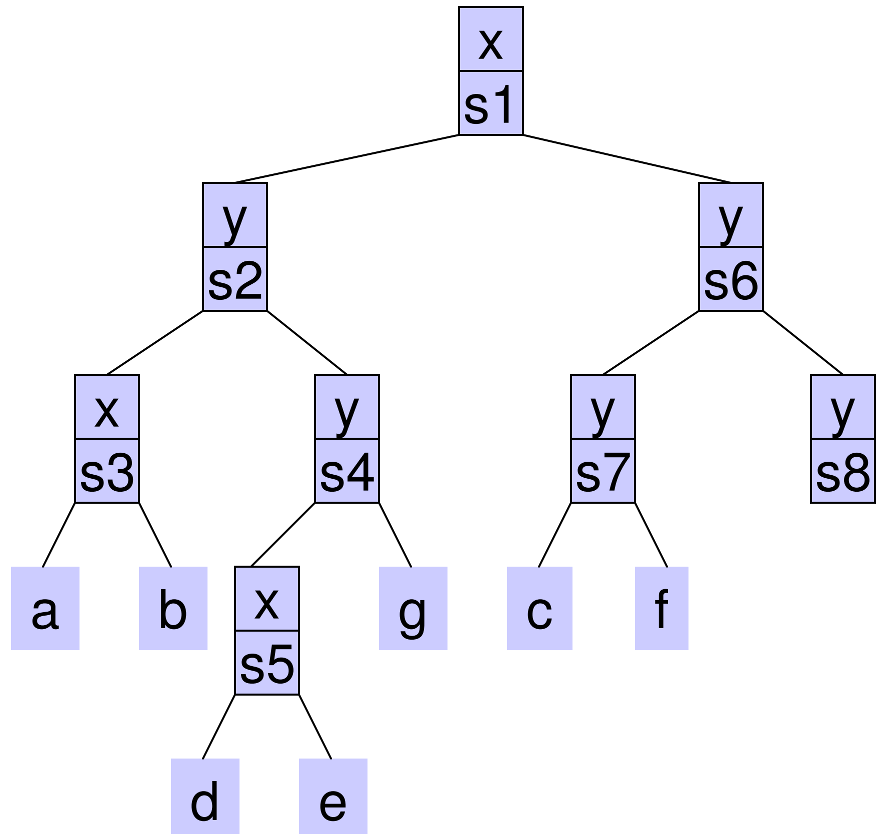


k-d Tree Construction (15)

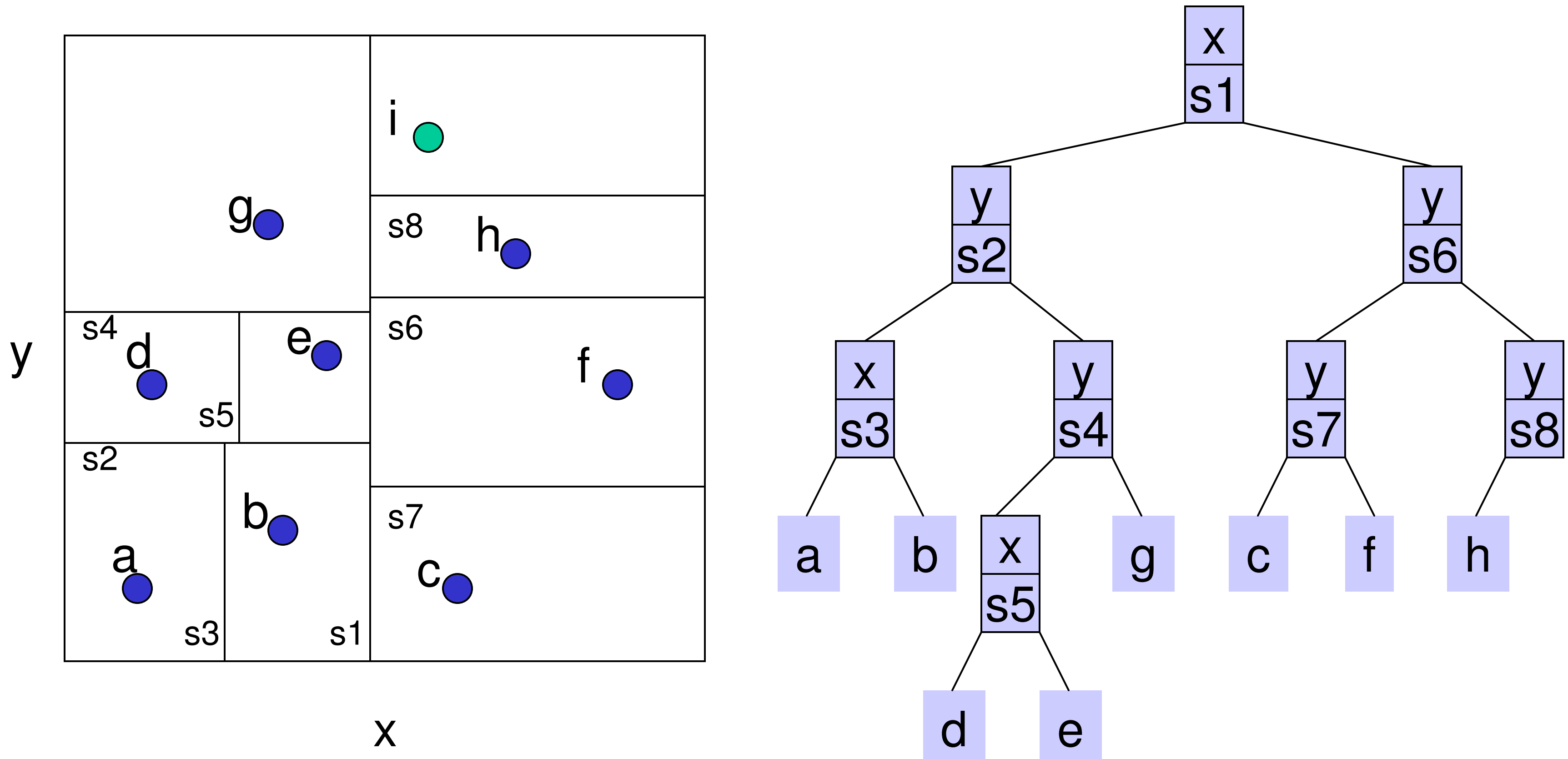


k-d Tree Construction (16)

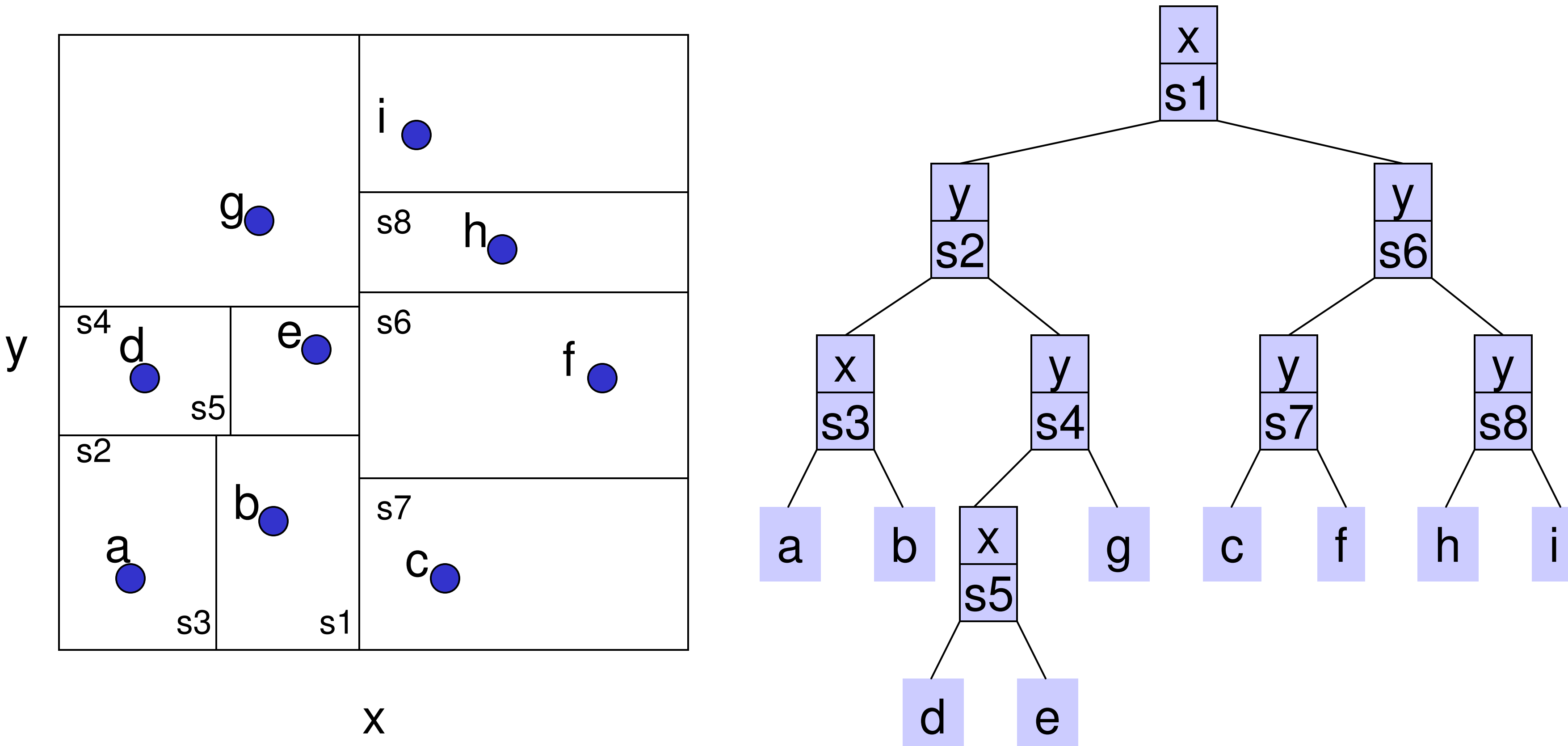




k-d Tree Construction (17)



k-d Tree Construction (18)

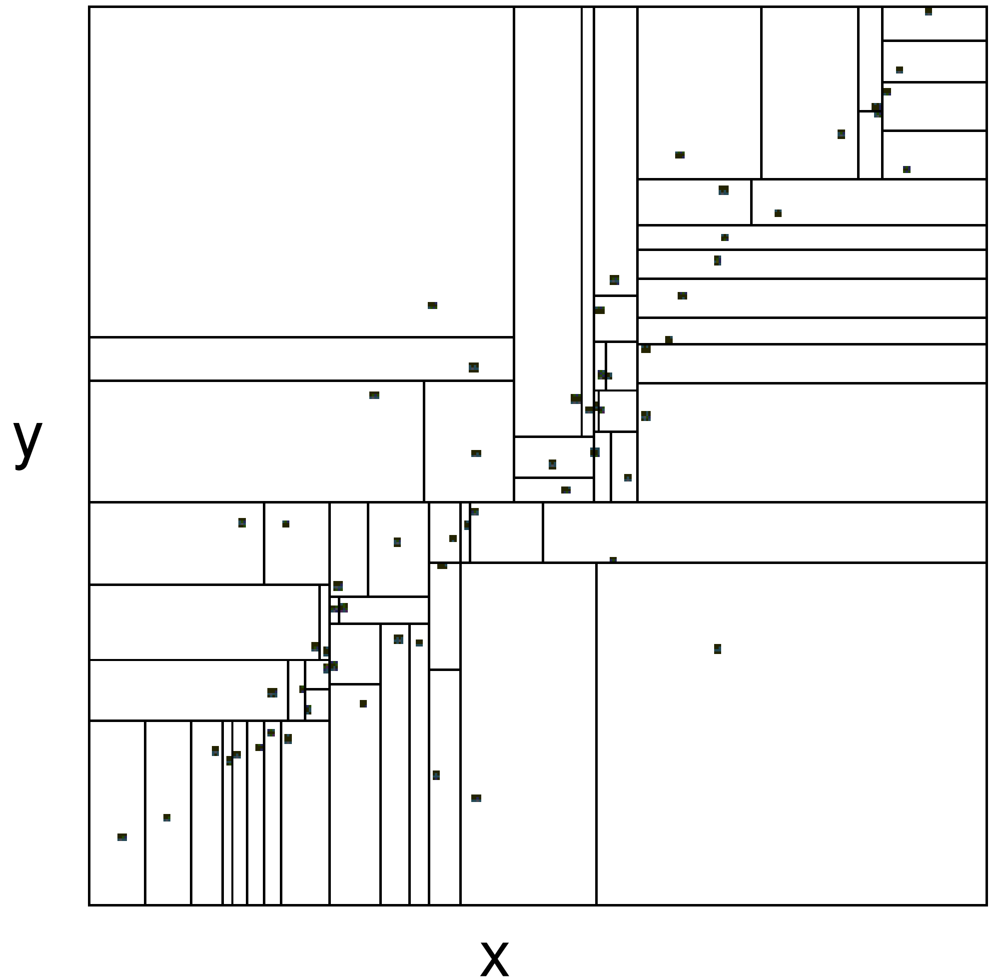


k-d Tree Construction Complexity

- First sort the points in each dimension.
  - O(dn log n) time and dn storage.
  - These are stored in A[1..d,1..n]
- Finding the widest spread and equally dividing into two subsets can be done in O(dn) time.
- Constructing the k-d tree can be done in O(dn log n) and dn storage

k-d Tree Codebook Organization

2-d vectors
(x,y)



Node Structure for k-d Trees

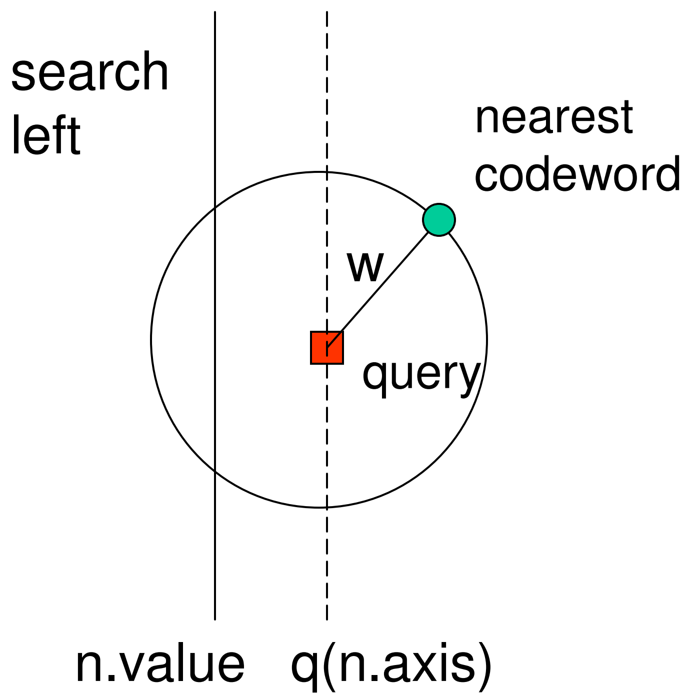
- A node has 5 fields
  - axis (splitting axis)
  - value (splitting value)
  - left (left subtree)
  - right (right subtree)
  - point (holds a point if left and right children are null)

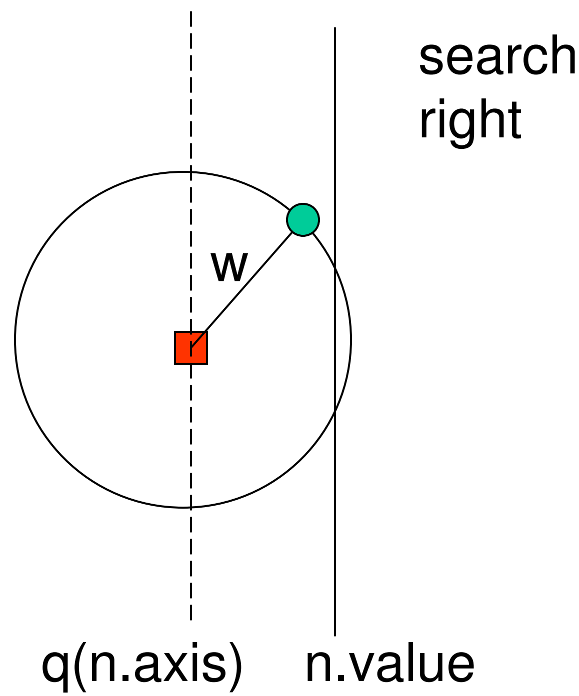
k-d Tree Nearest Neighbor Search

```
NNS(q: point, n: node, p: ref point w: ref distance)
if n.left = n.right = null then {leaf case}
   w' := ||q - n.point||;
   if w' < w then w := w'; p := n.point;
else
   if w = infinity then
        if q(n.axis) ≤ n.value then
            NNS(q, n.left, p, w);
            if q(n.axis) + w > n.value then NNS(q, n.right, p, w);
        else
            NNS(q, n.right, p, w);
            if q(n.axis) - w ≤ n.value then NNS(q, n.left, p, w)
   else {w is finite}
        if q(n.axis) - w ≤ n.value then NNS(q, n.left, p, w)
        if q(n.axis) + w > n.value then NNS(q, n.right, p, w);
```

initial call `NNS(q, root, p, infinity)`

Explanation

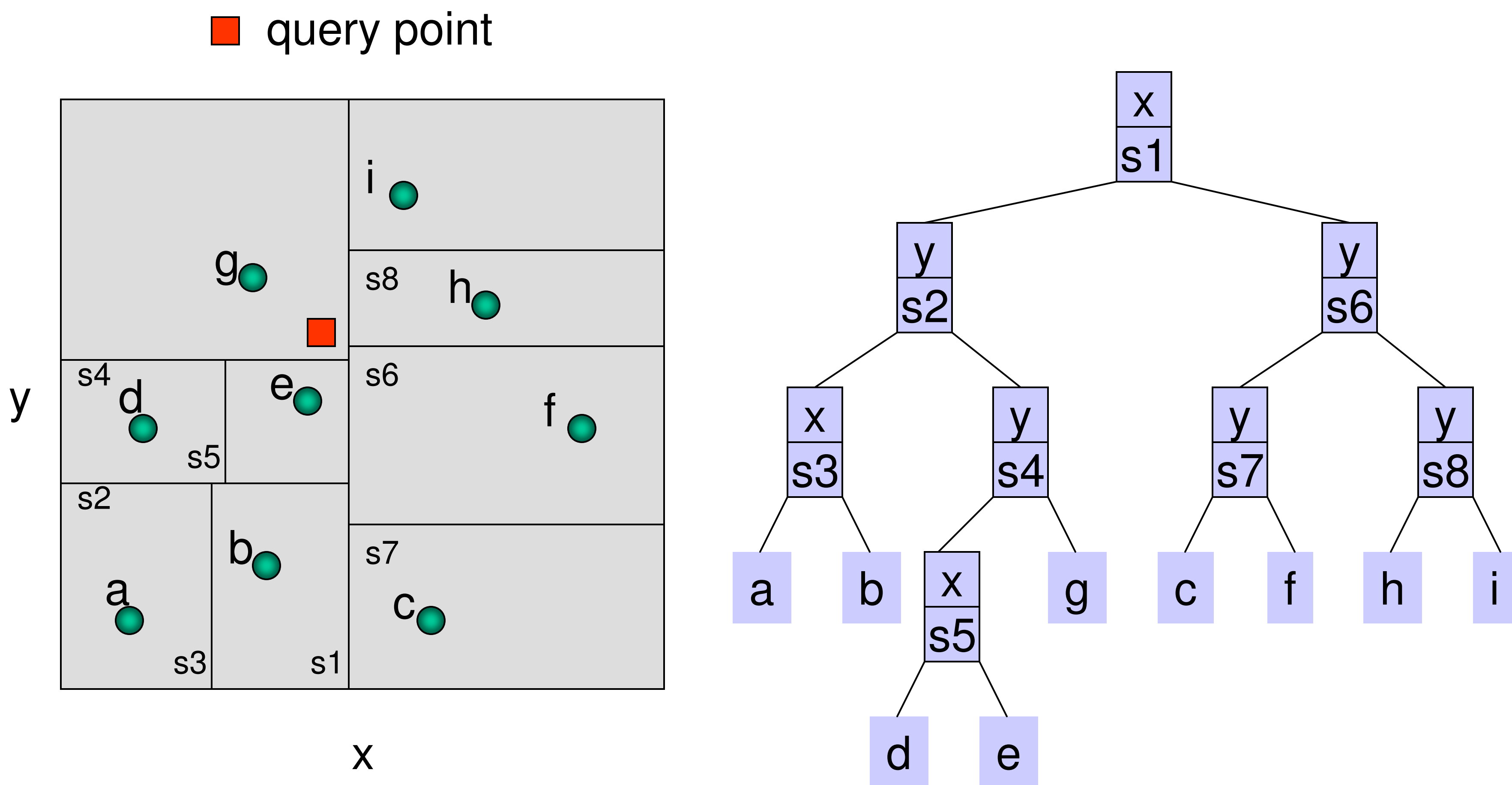




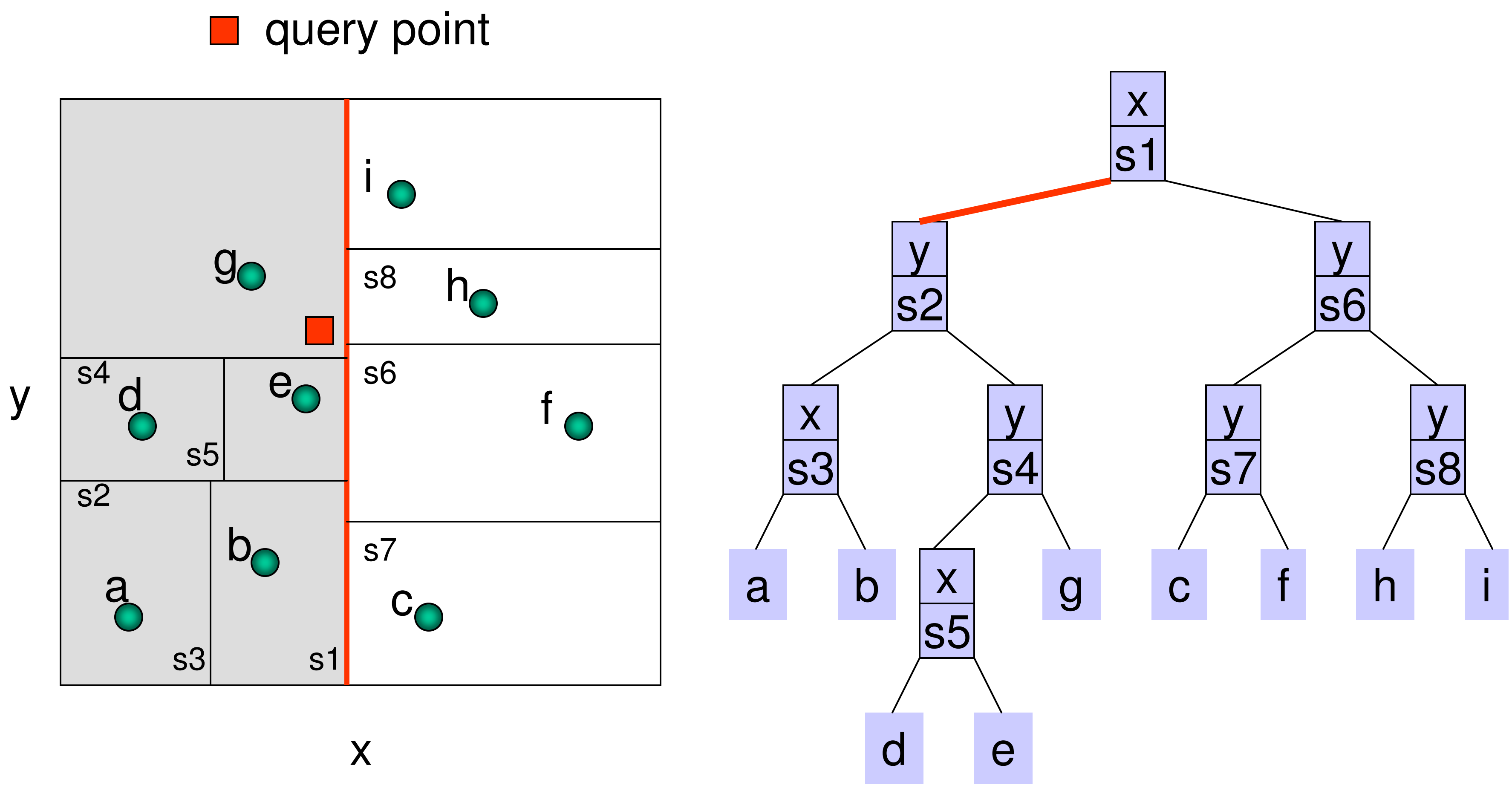
$q(n.axis) - w \leq n.value$
means the circle overlaps
the left subtree.

$q(n.axis) + w > n.value$
means the circle overlaps
the right subtree.

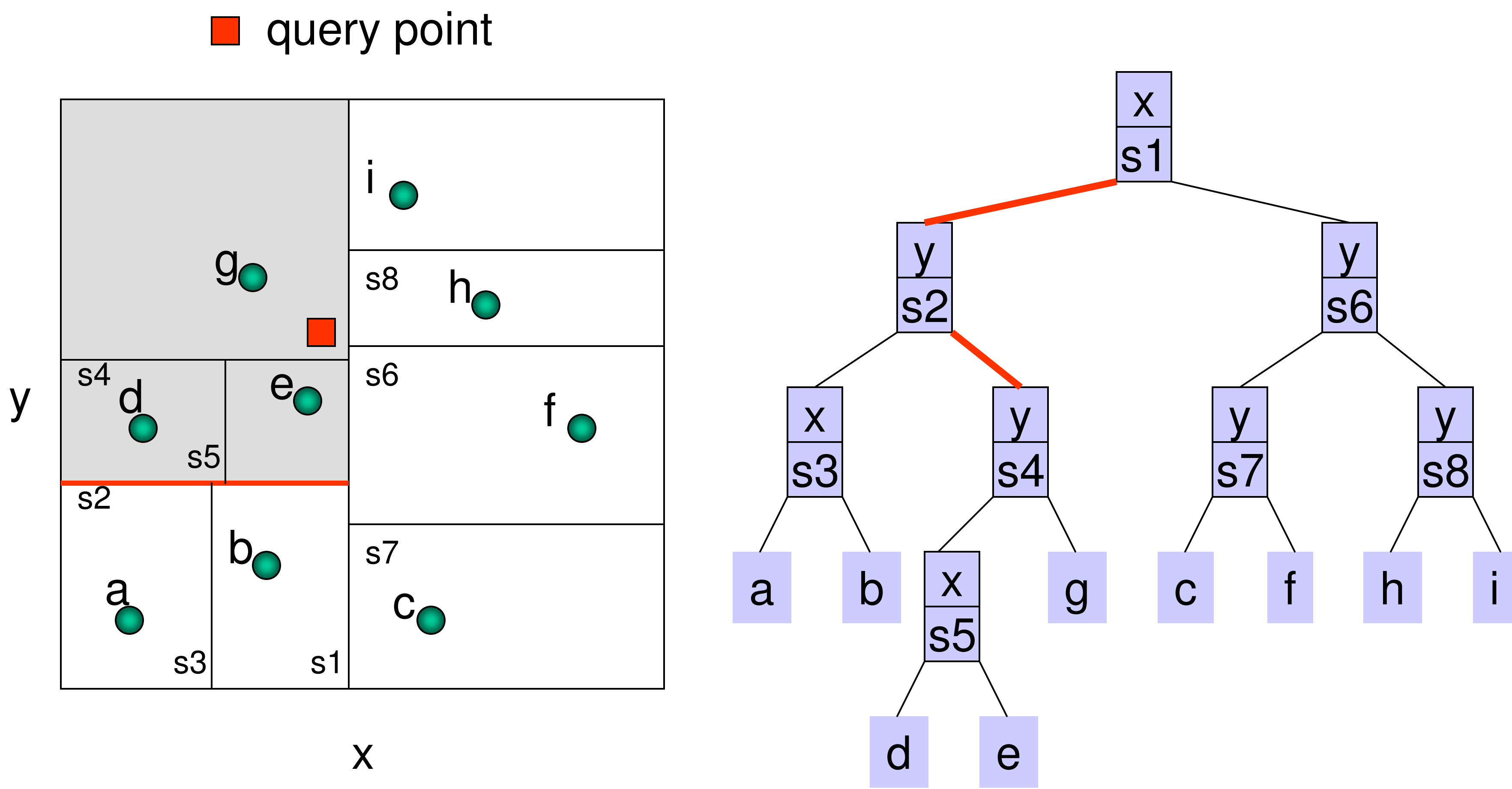
k-d Tree NNS (1)


query point
i
g
s8
h
s4
d
e
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (2)

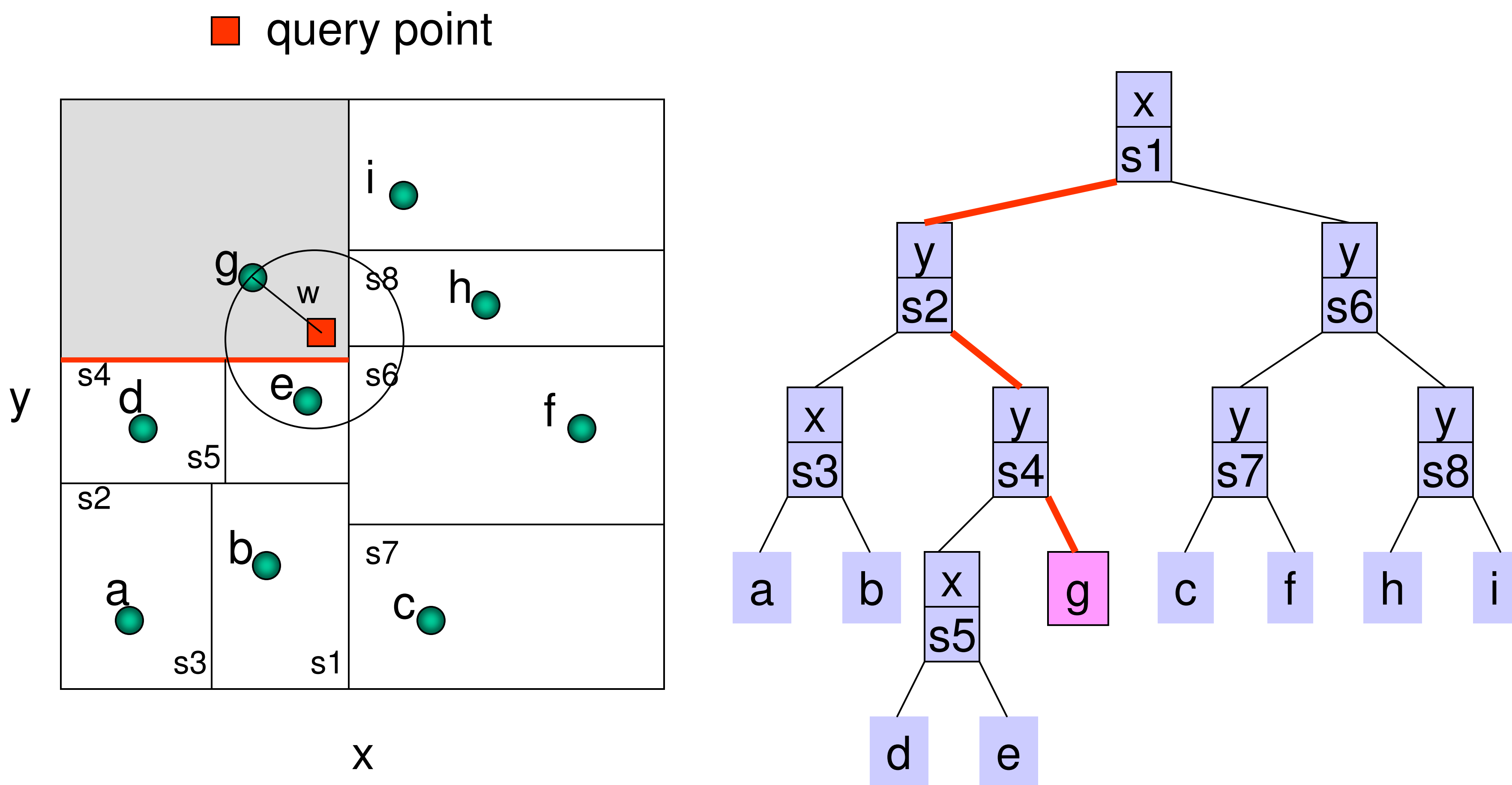

query point
i
g
s8
h
s4
d
e
s6
y
f
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (3)



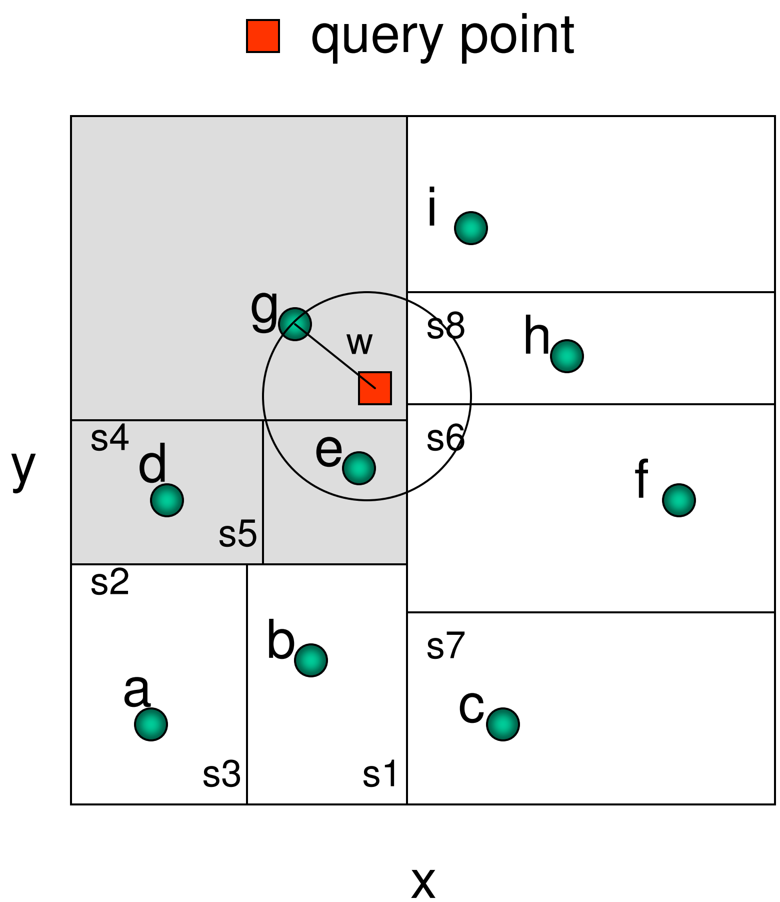
query point
y
x
s4
d
e
s5
s2
a
b
s3
s1
g
i
s8
h
s6
f
s7
c
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (4)

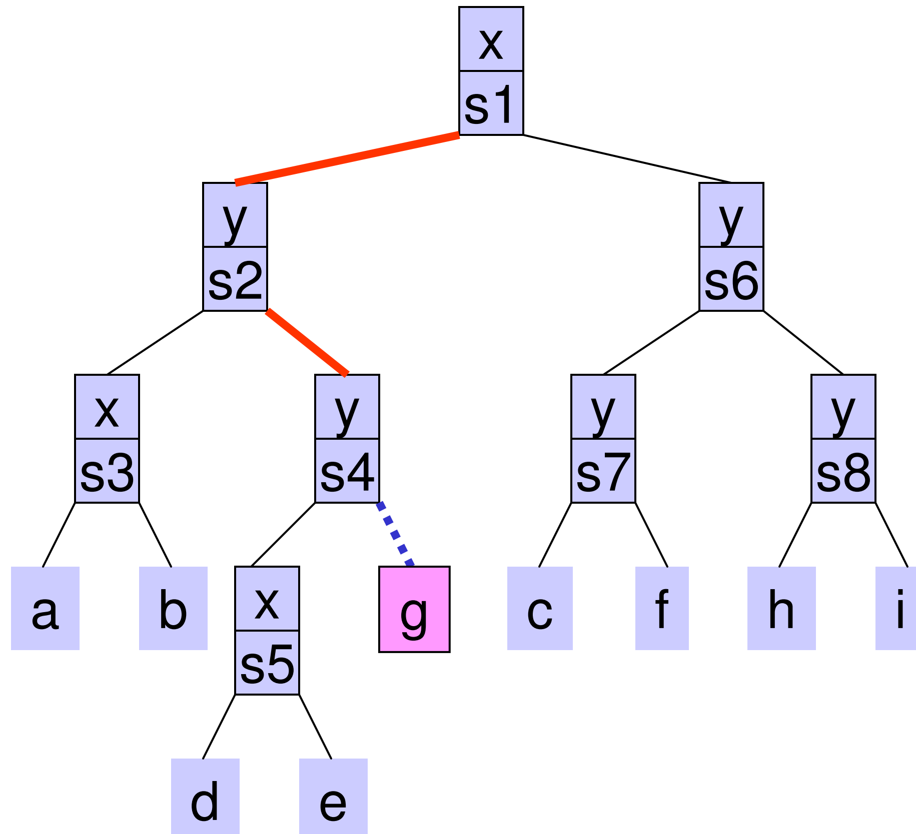


query point
i
g
w
s8
h
s4
d
e
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (5)

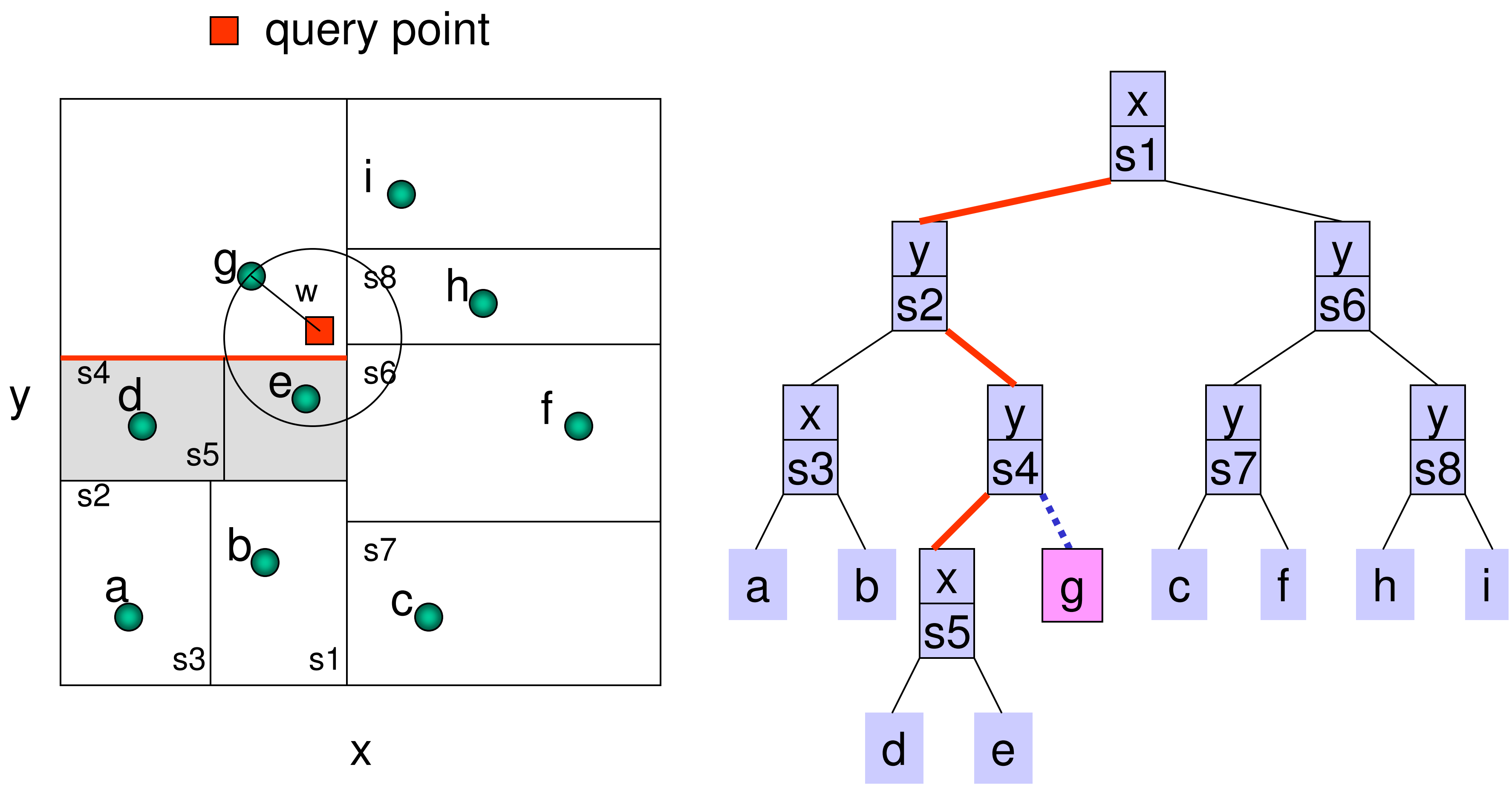


query point
i
g
w
s8
h
s4
d
e
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x

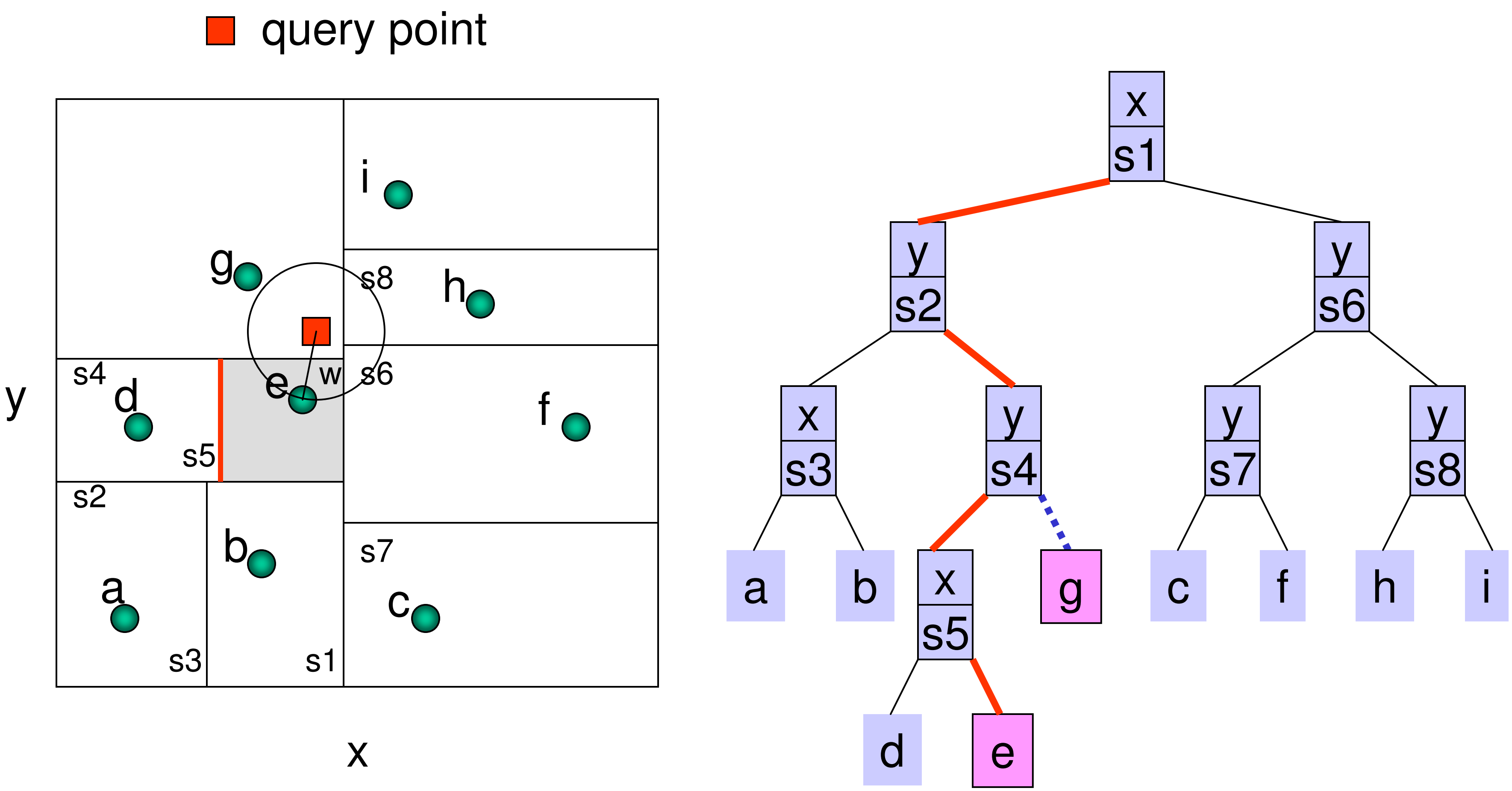


x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

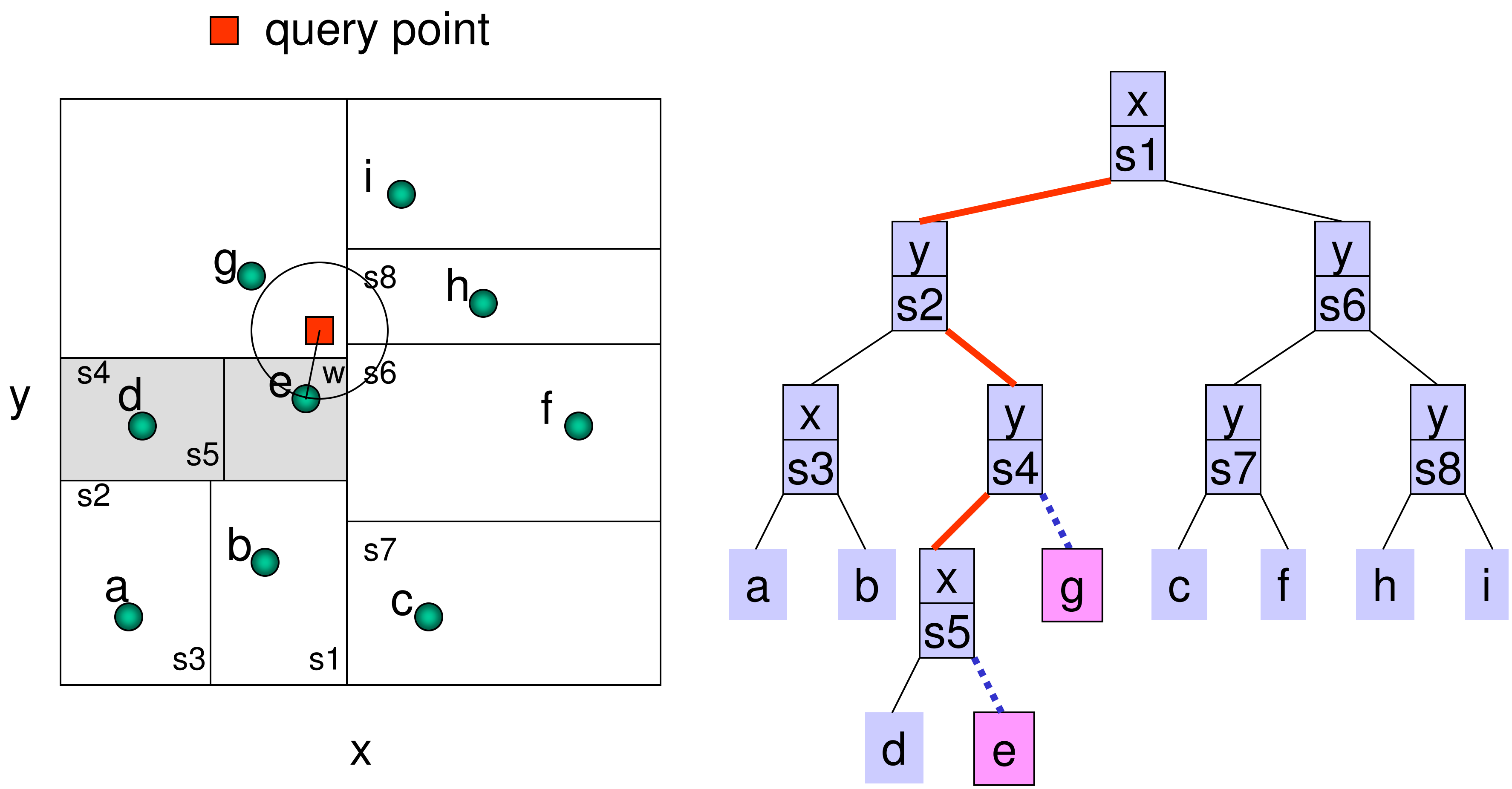
k-d Tree NNS (6)


query point
y
x
i
g
w
s8
h
s4
d
e
s6
f
s5
s2
b
a
s7
c
s3
s1
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

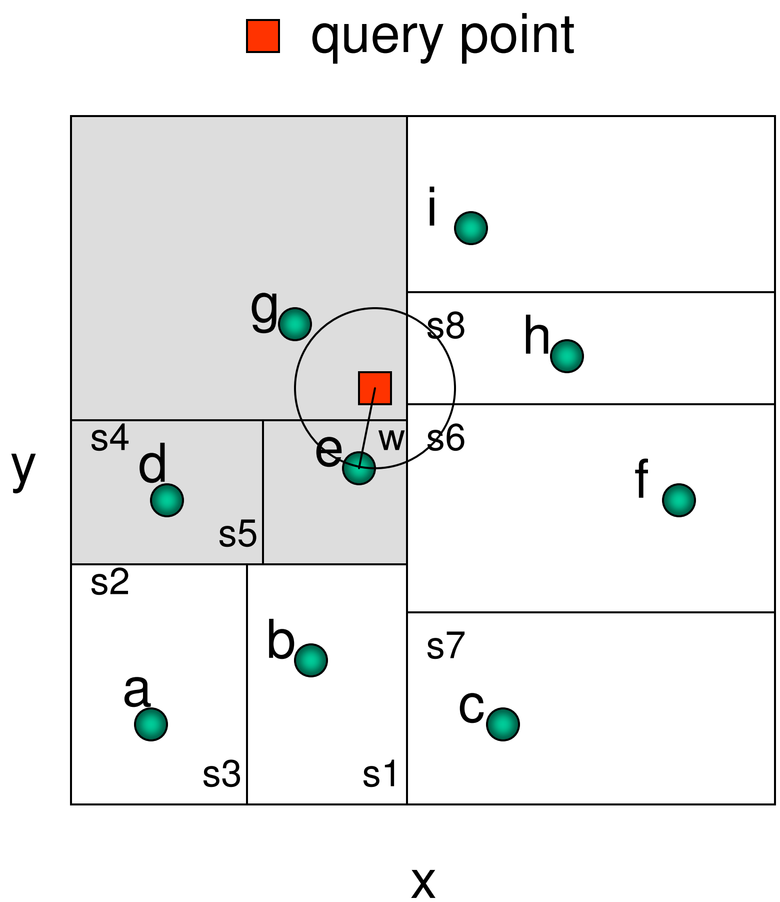
k-d Tree NNS (7)


query point
i
g
s8
h
s4
d
e
w
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

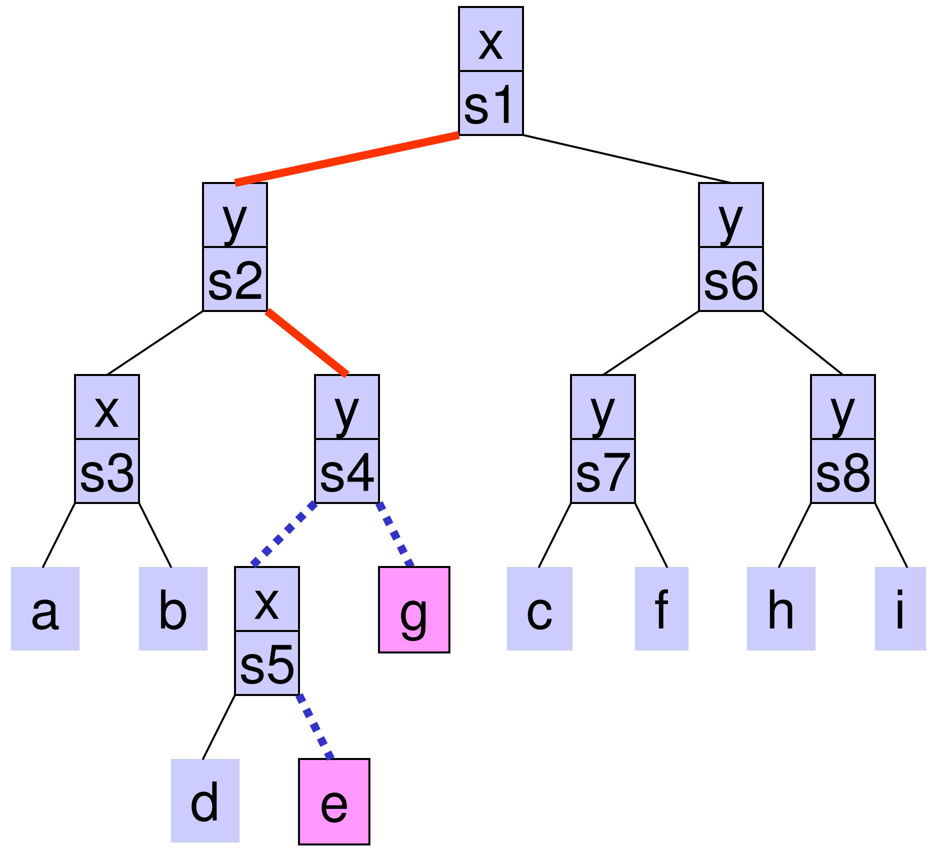
k-d Tree NNS (8)


query point
i
g
s8
h
s4
d
e
w
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (9)

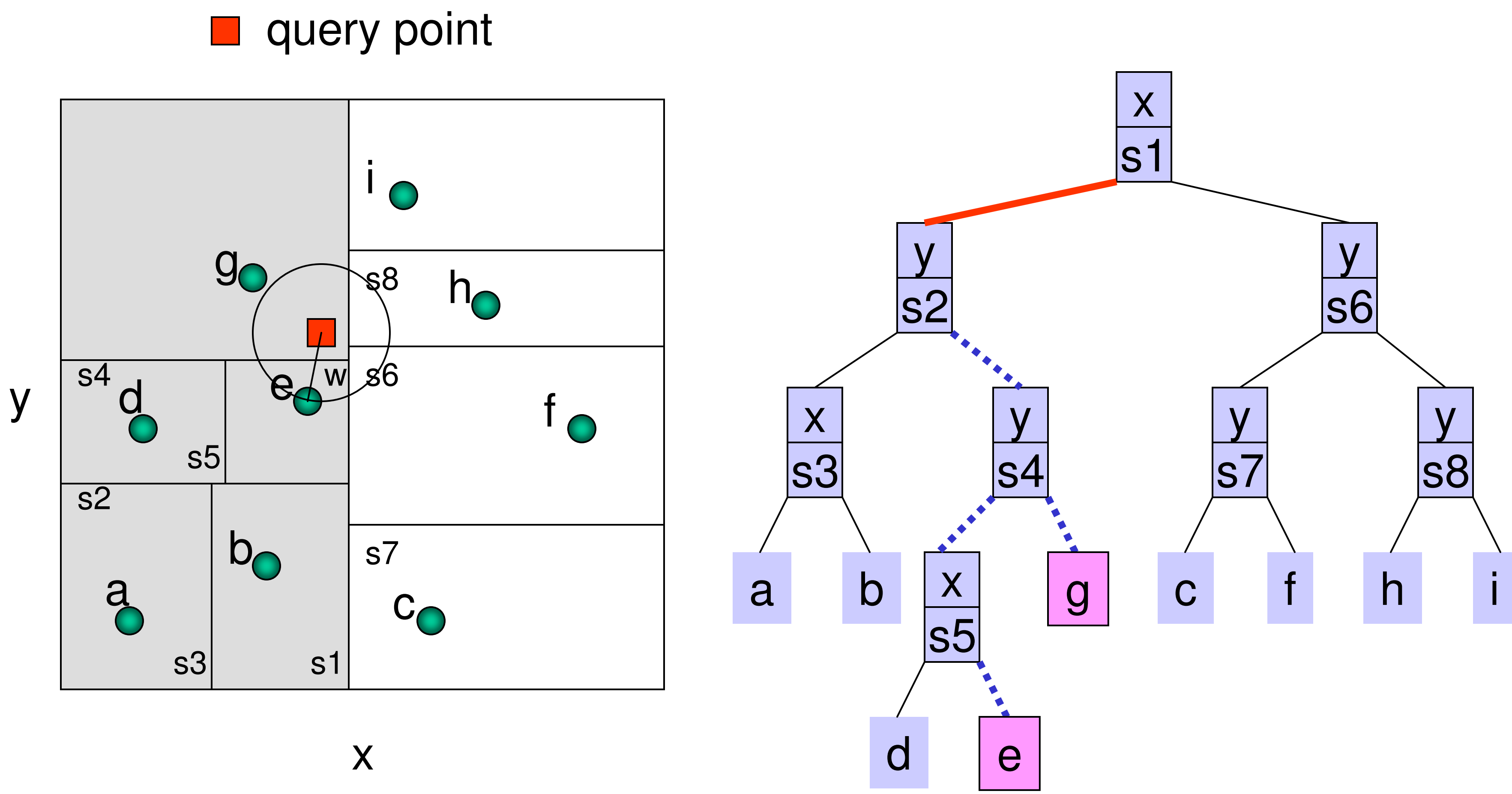


query point
y
x
s1
s2
s3
s4
s5
s6
s7
s8
a
b
c
d
e
f
g
h
i
w

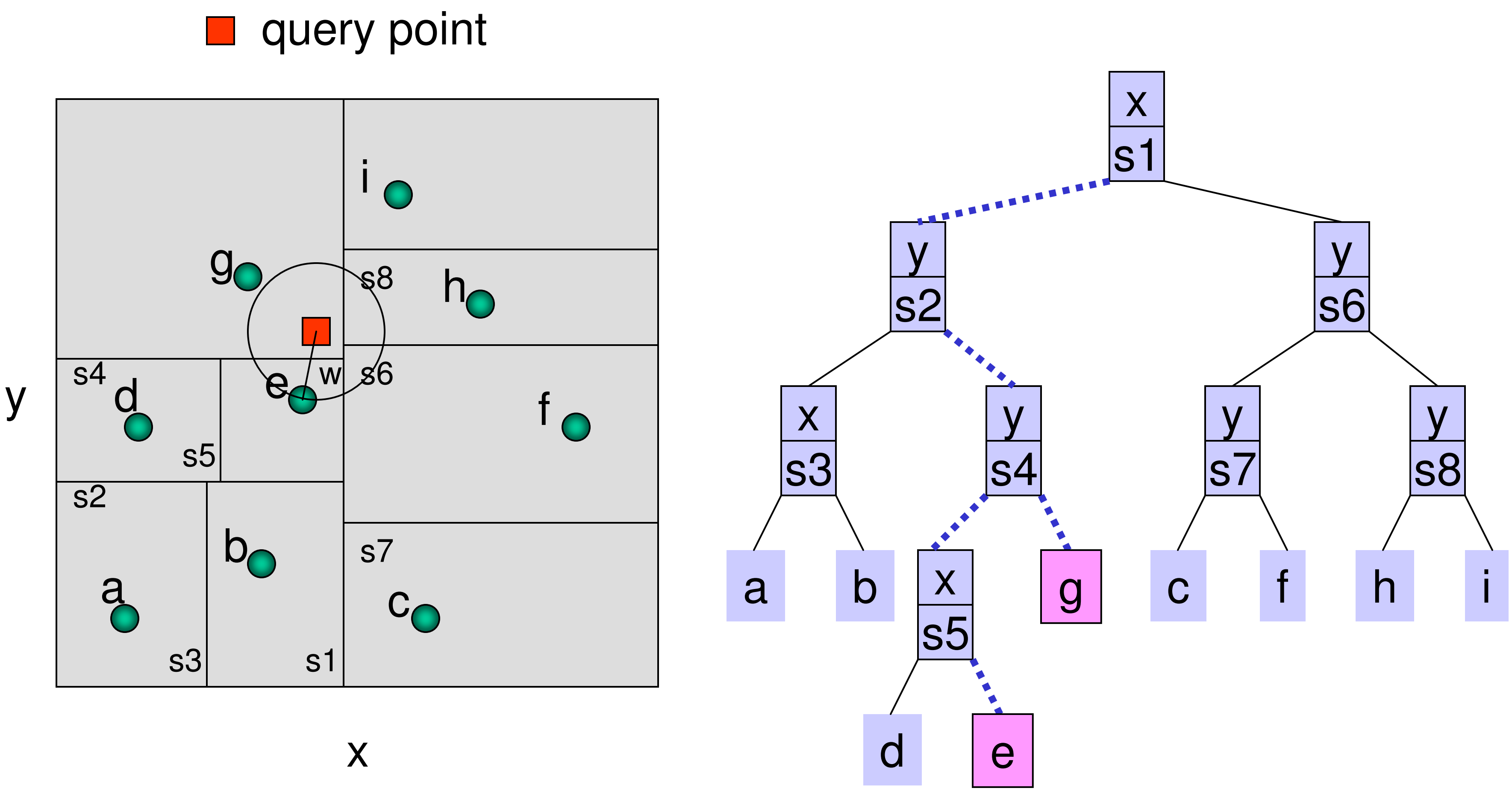


x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

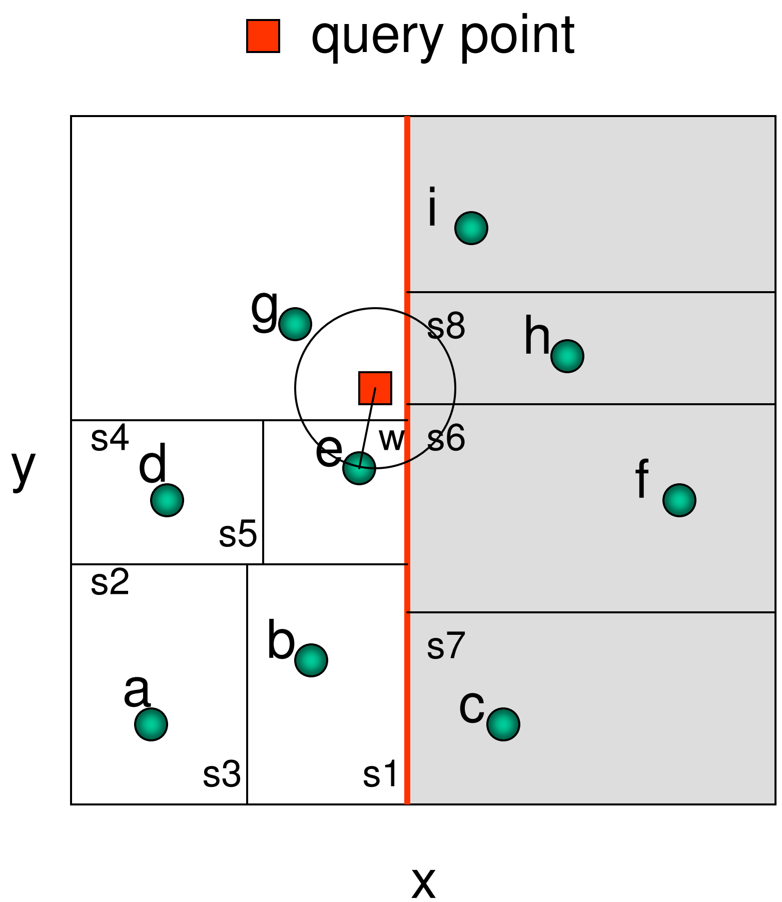
k-d Tree NNS (10)

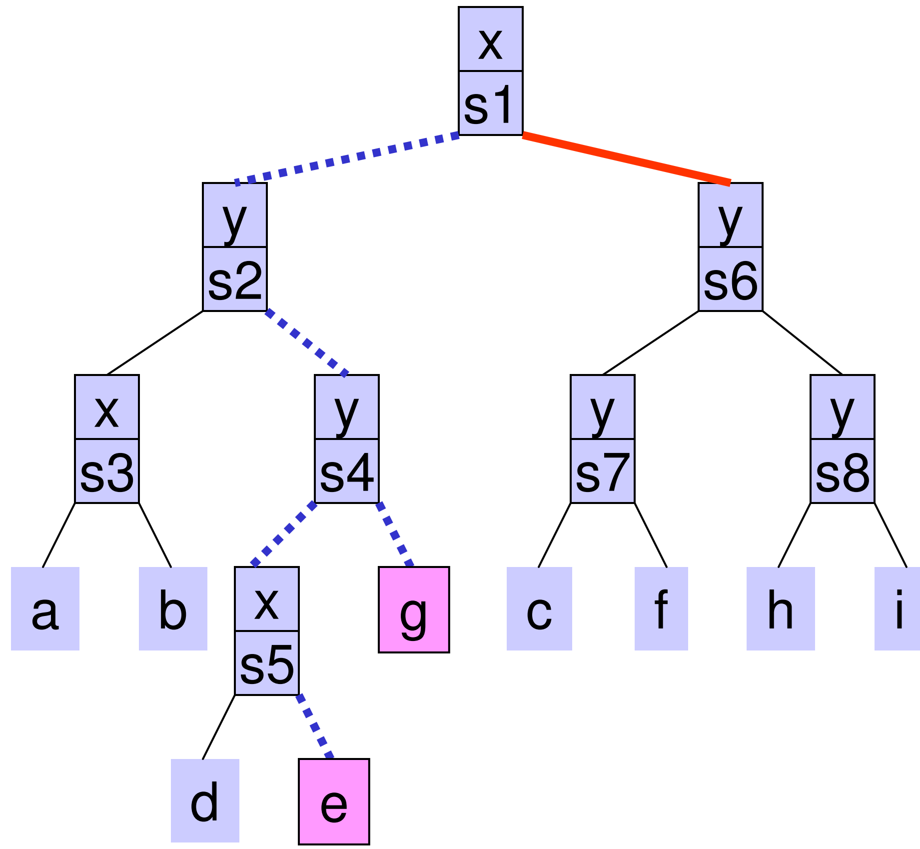


k-d Tree NNS (11)

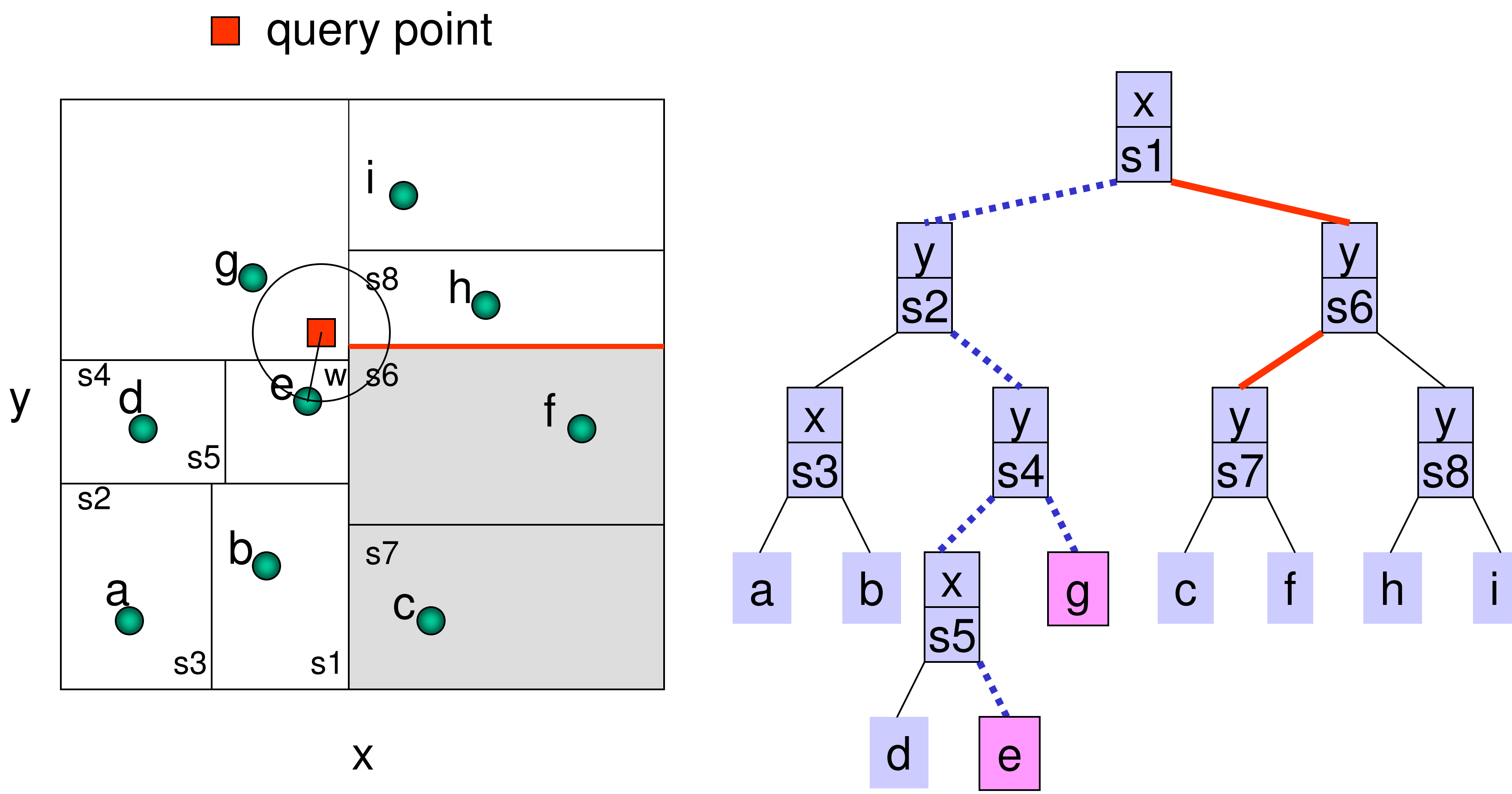


k-d Tree NNS (12)

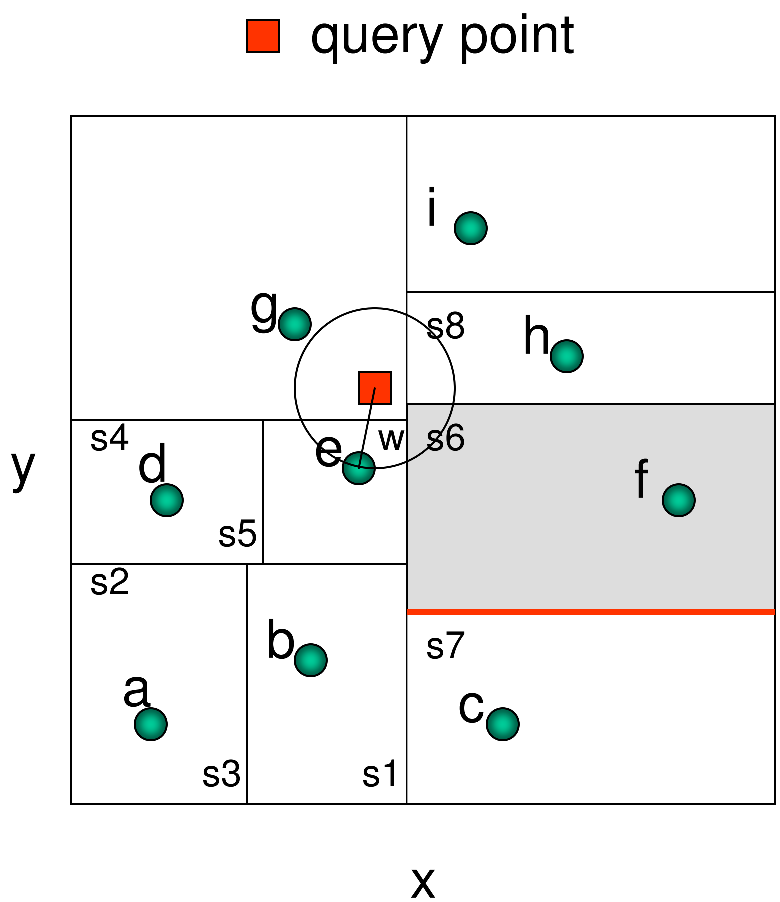


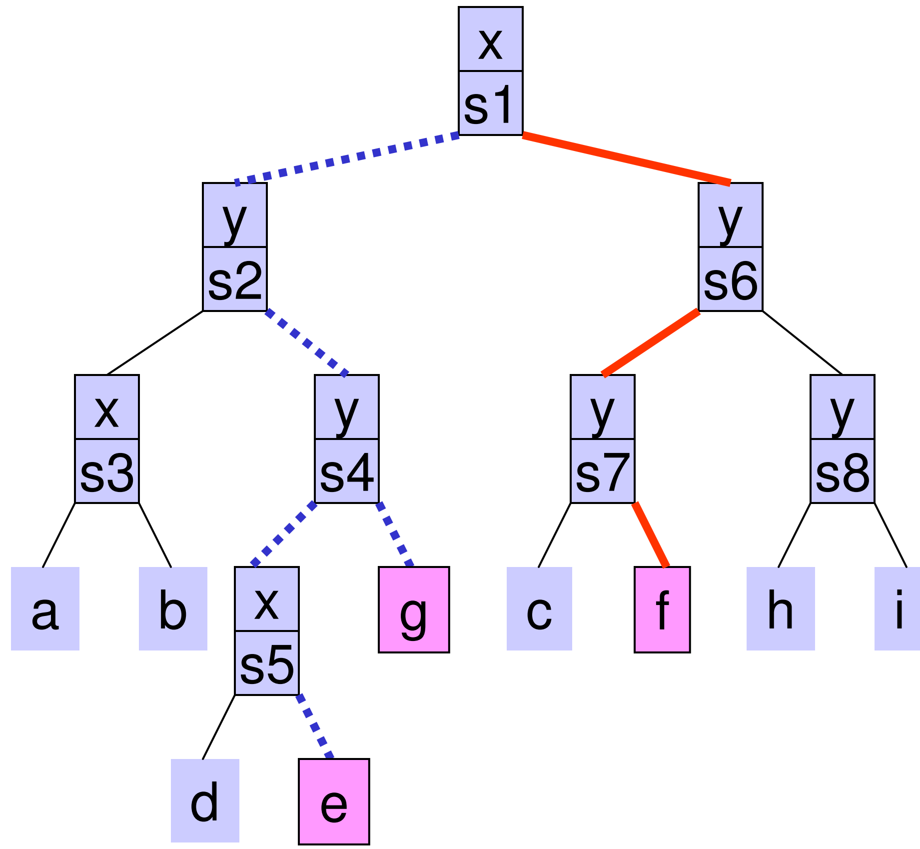


k-d Tree NNS (13)

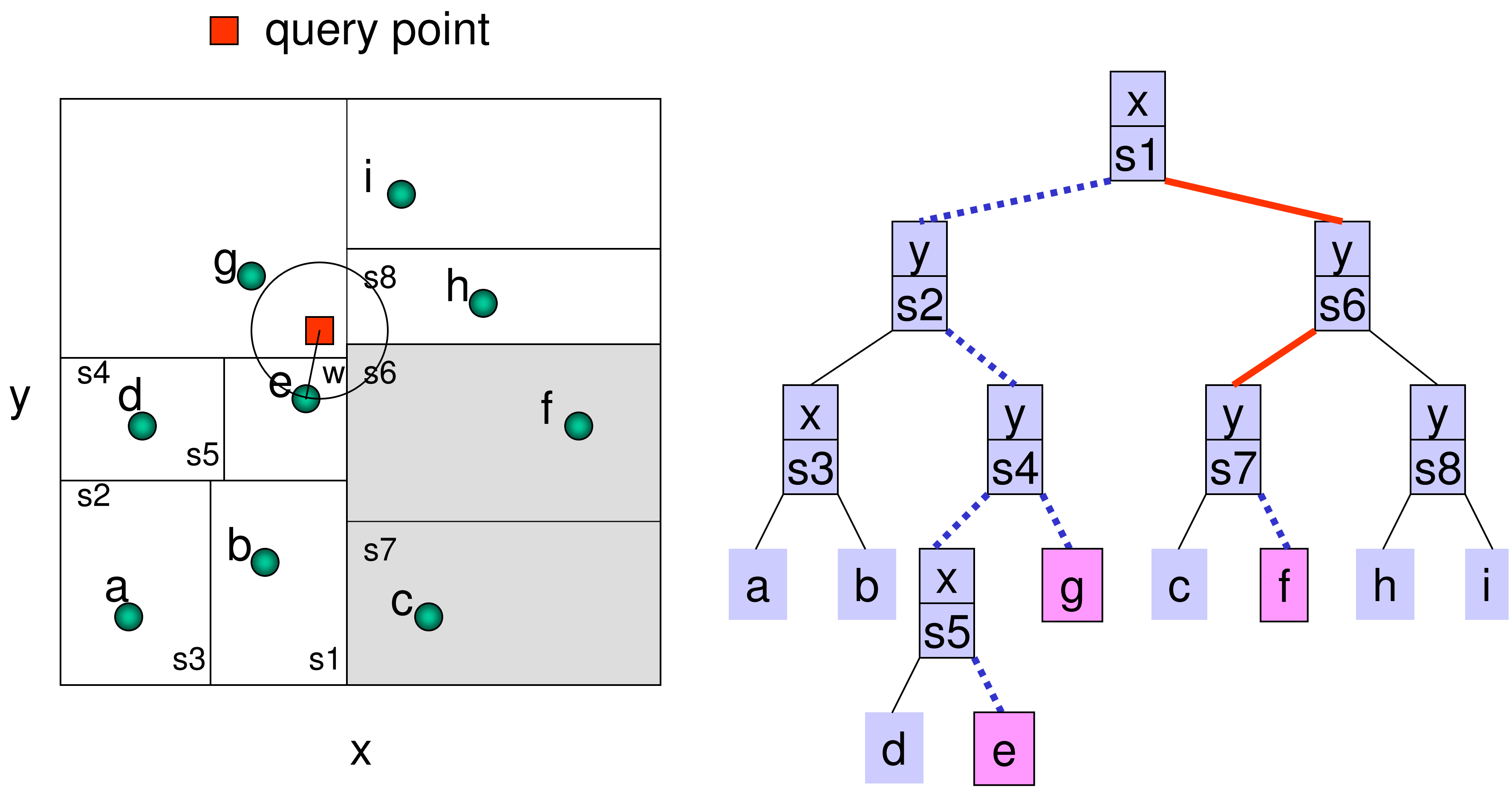


k-d Tree NNS (14)

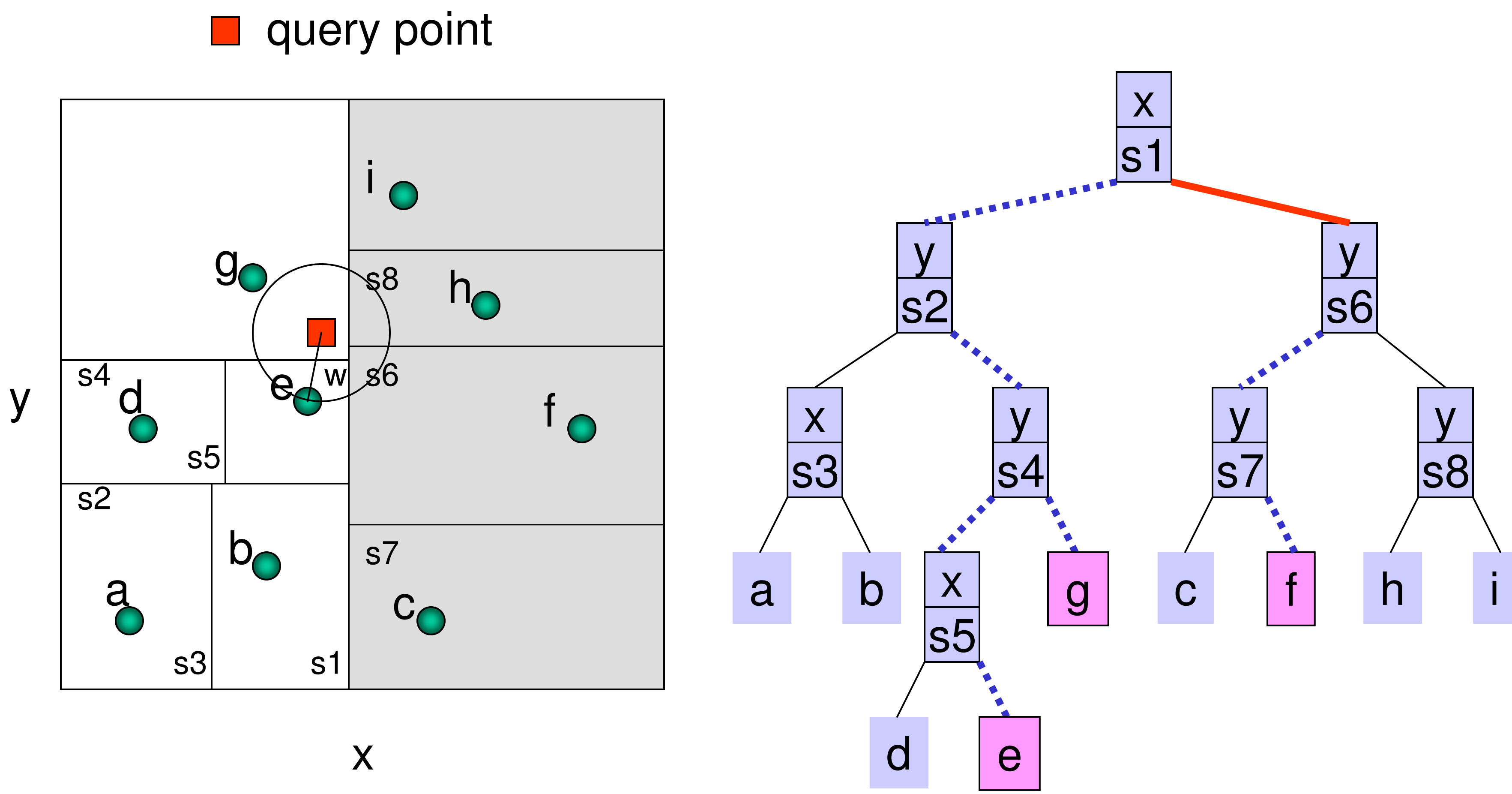




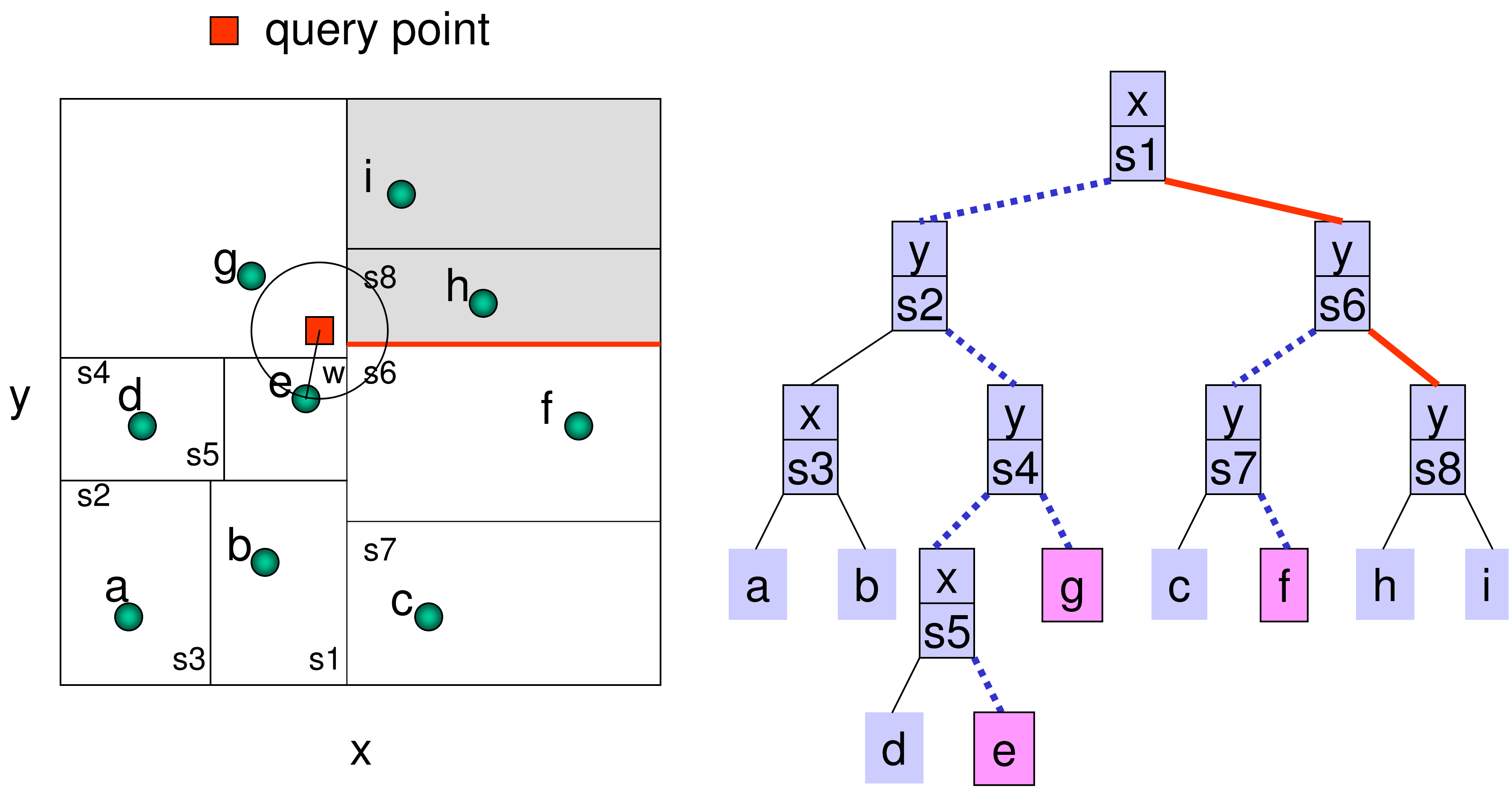
k-d Tree NNS (15)



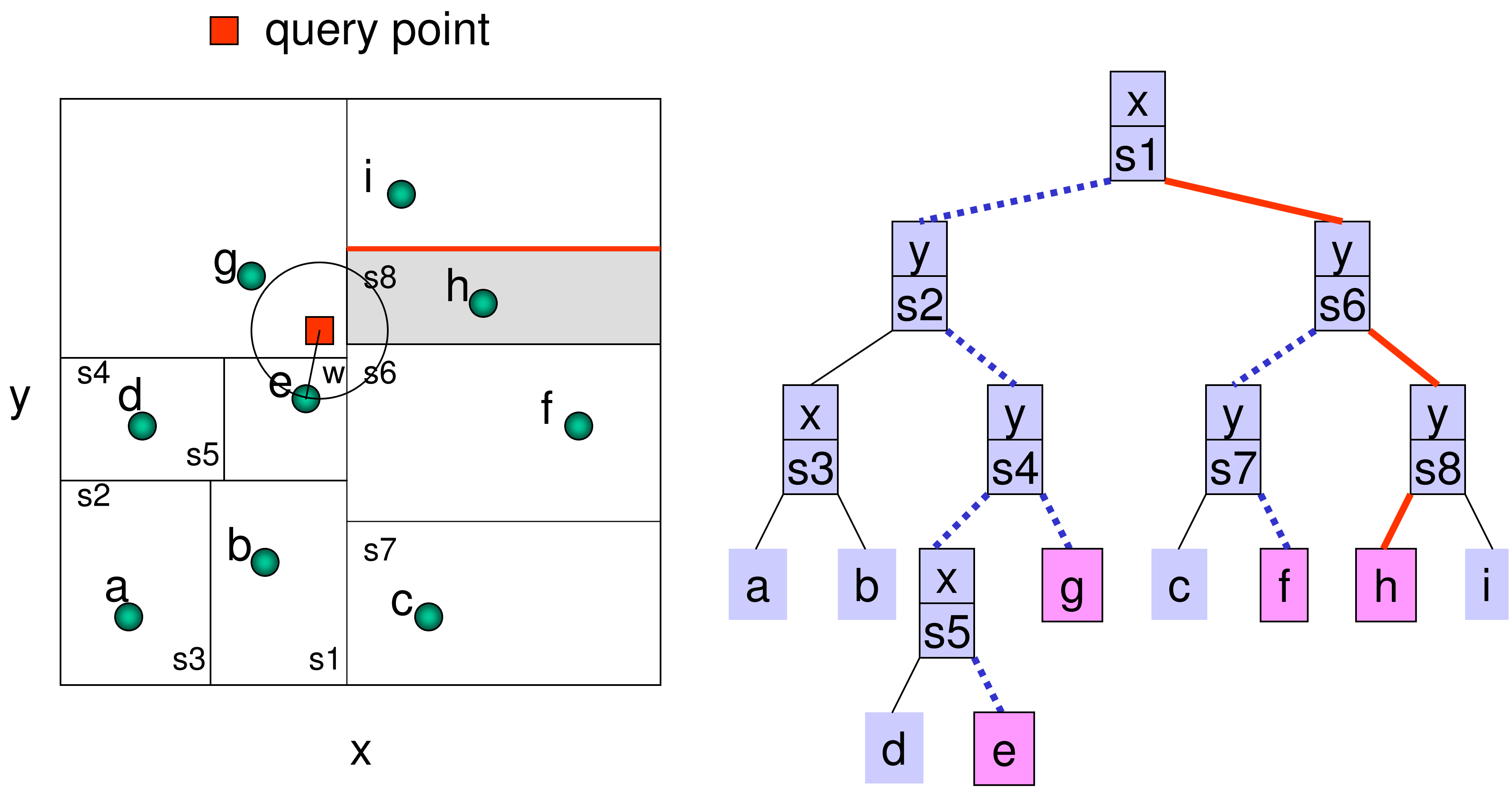
k-d Tree NNS (16)



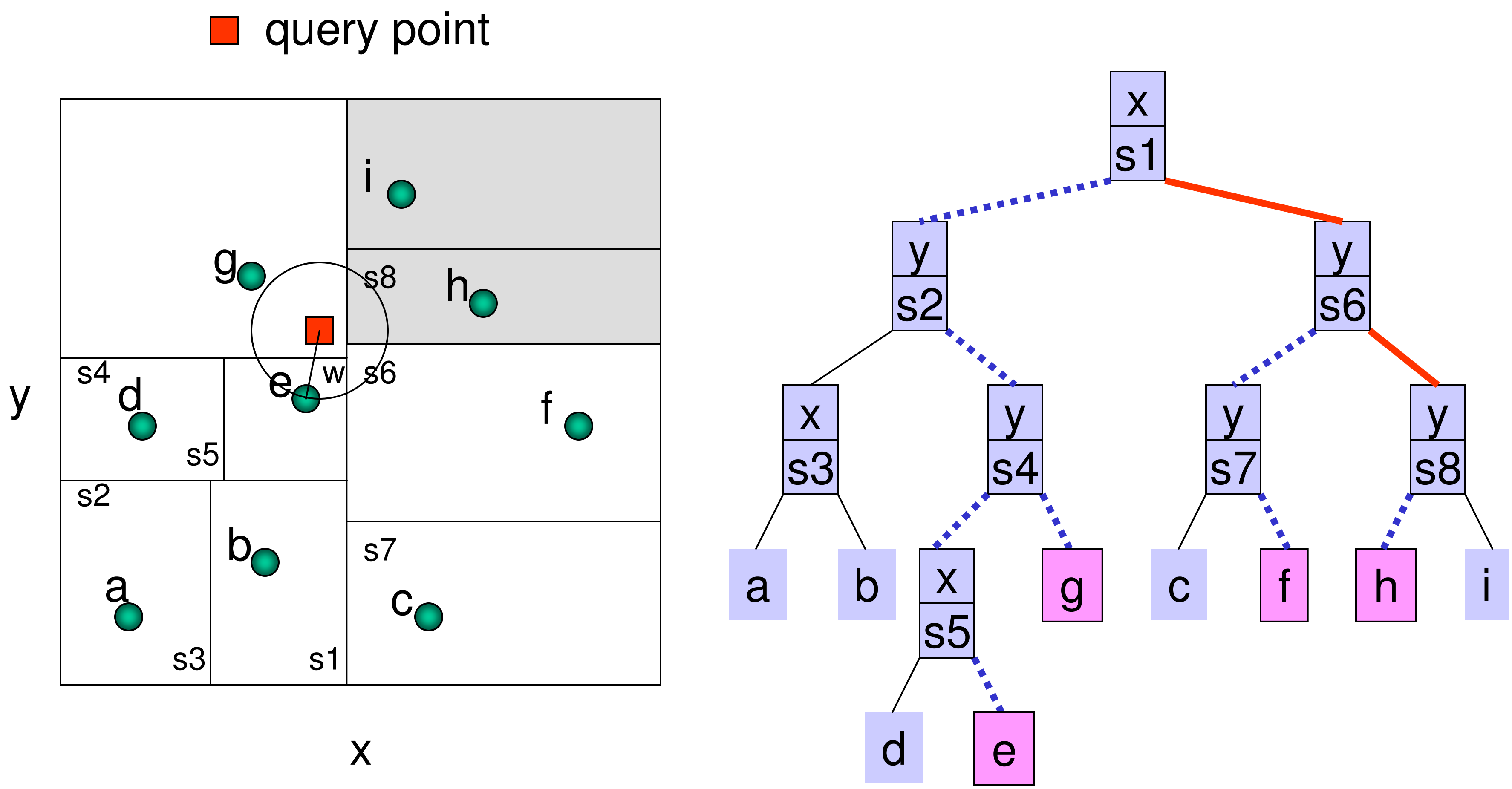
k-d Tree NNS (17)



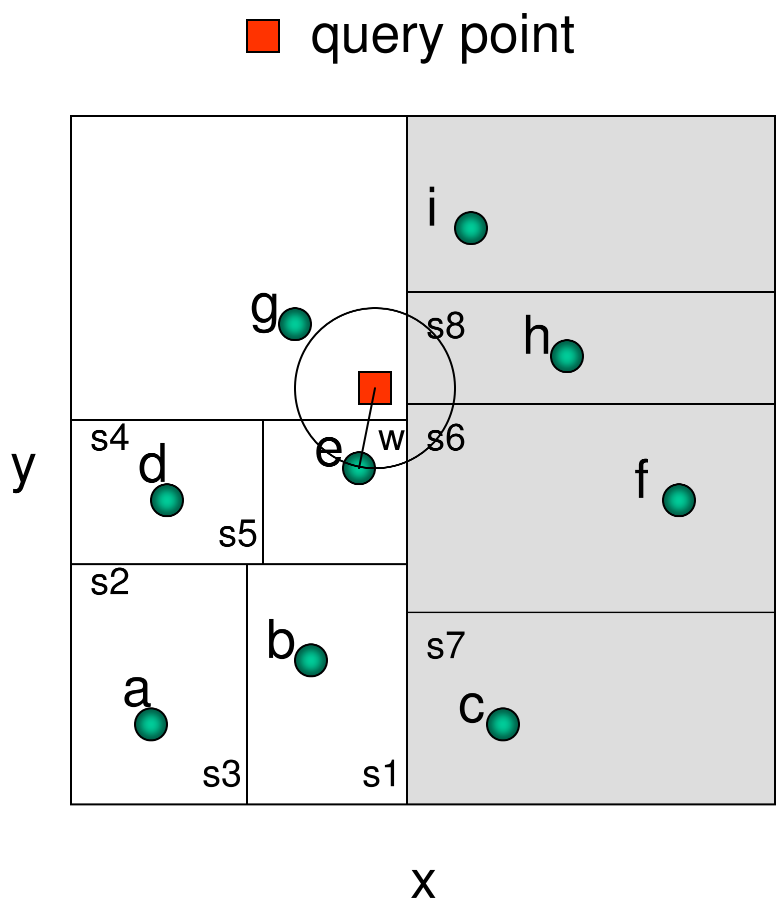
k-d Tree NNS (18)

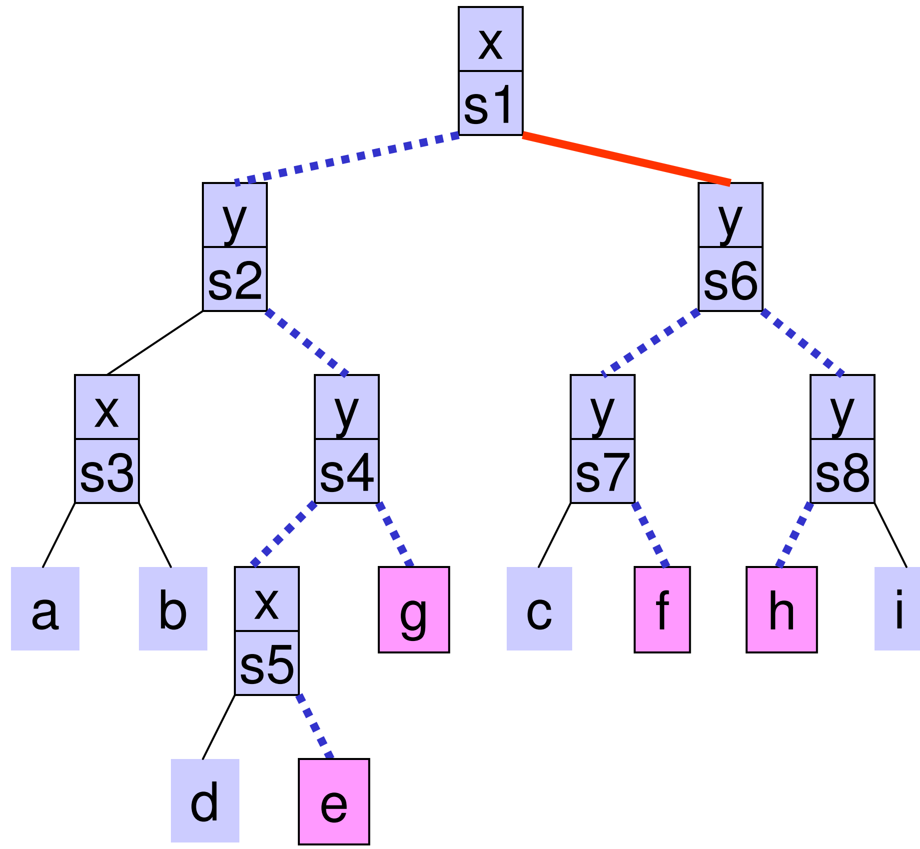


k-d Tree NNS (19)

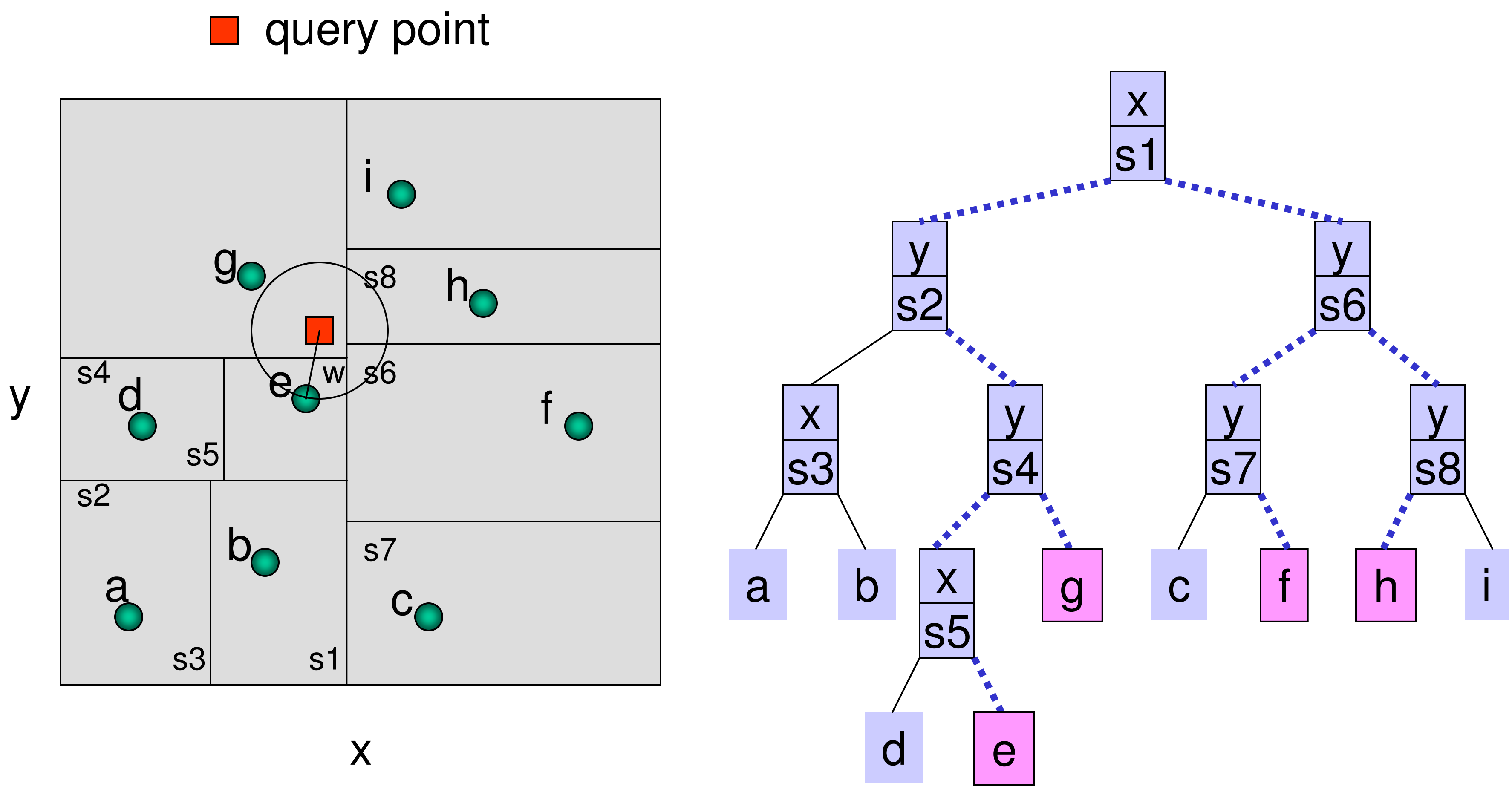


k-d Tree NNS (20)





k-d Tree NNS (21)



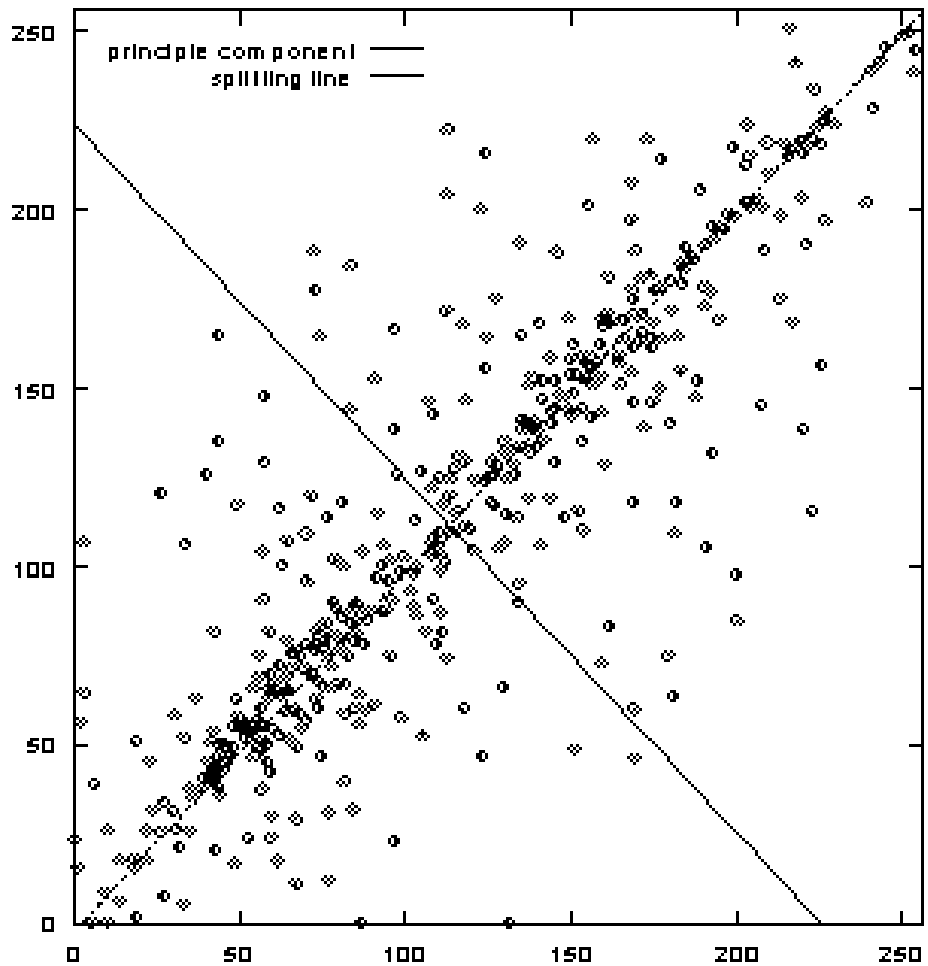
Notes on k-d Tree NNS

- Has been shown to run in O(log n) average time per search in a reasonable model. (Assume d a constant)
- For VQ it appears that O(log n) is correct.
- Storage for the k-d tree is O(n).
- Preprocessing time is O(n log n) assuming d is a constant.

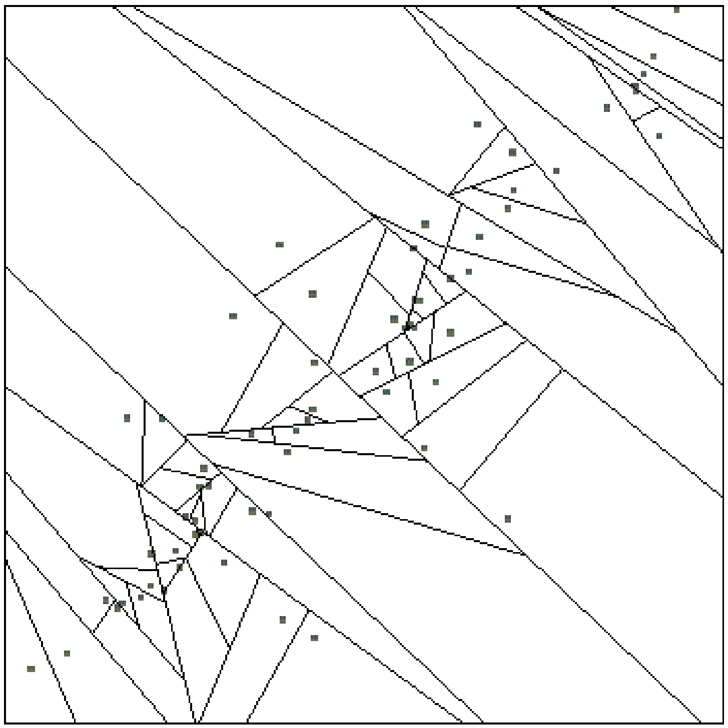
Alternatives

- Orchard's Algorithm (1991)
  - Uses $O(n^2)$ storage but is very fast
- Annulus Algorithm
  - Similar to Orchard but uses O(n) storage. Does many more distance calculations.
- PCP Principal Component Partitioning
  - Zatloukal, Johnson, Ladner (1999)
  - Similar to k-d trees
  - Also very fast

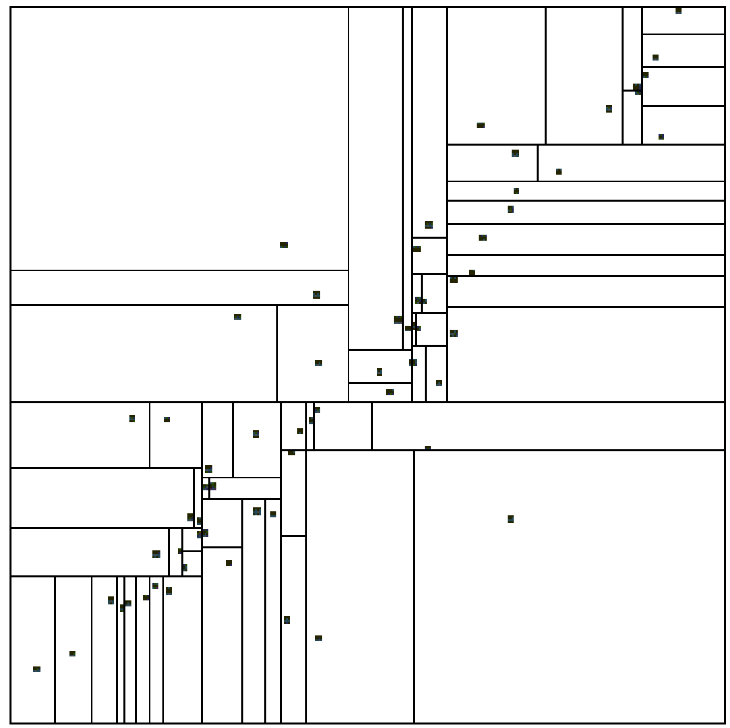
Principal Component Partition



PCP Tree vs. k-d tree

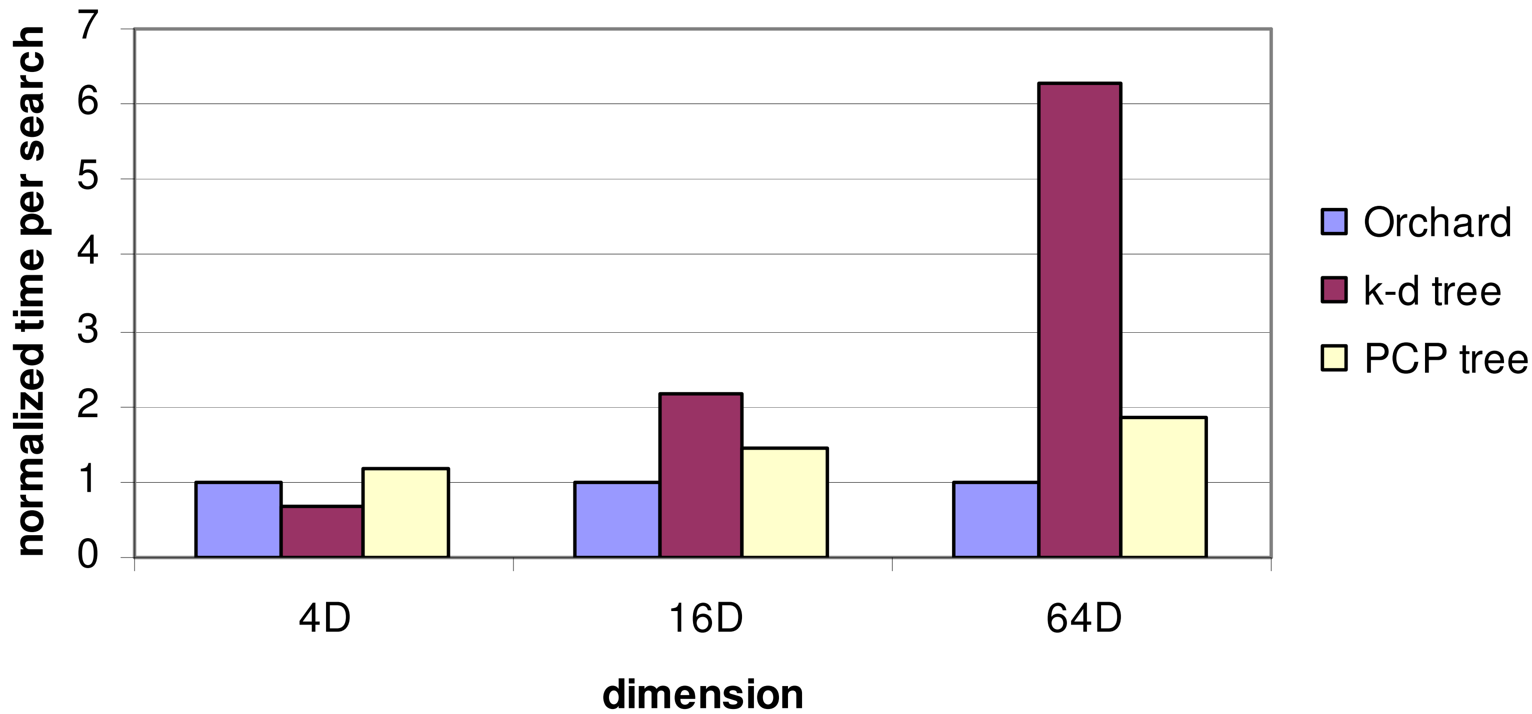


PCP



k-d

Comparison in Time per Search



4,096 codewords

Notes on VQ

- Works well in some applications.
  - Requires training
- Has some interesting algorithms.
  - Codebook design
  - Nearest neighbor search
- Variable length codes for VQ.
  - PTSVQ - pruned tree structured VQ (Chou, Lookabaugh and Gray, 1989)
  - ECVQ - entropy constrained VQ (Chou, Lookabaugh and Gray, 1989)