K-D Trees

CSE 373

Data Structures

Lecture 22

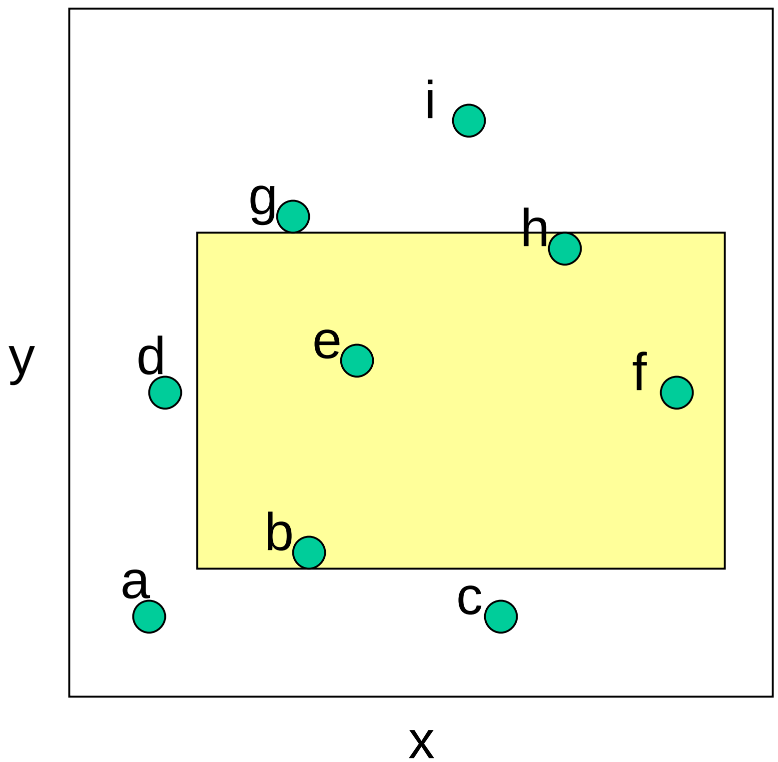
Geometric Data Structures

- Organization of points, lines, planes, … to support faster processing
- Applications
  - Astrophysical simulation – evolution of galaxies
  - Graphics – computing object intersections
  - Data compression
    - Points are representatives of 2x2 blocks in an image
    - Nearest neighbor search

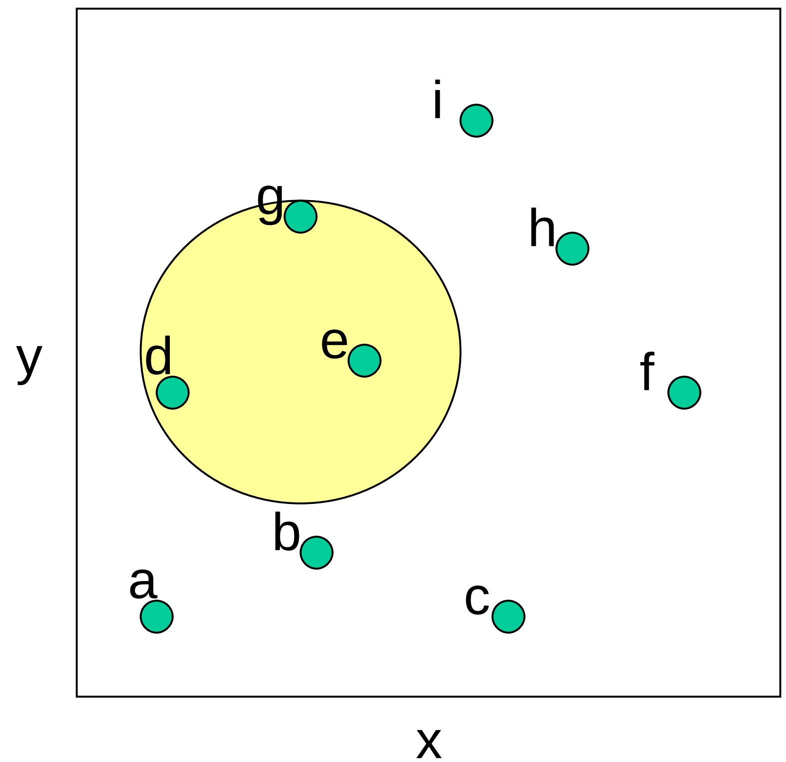
k-d Trees

- 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.

Range Queries

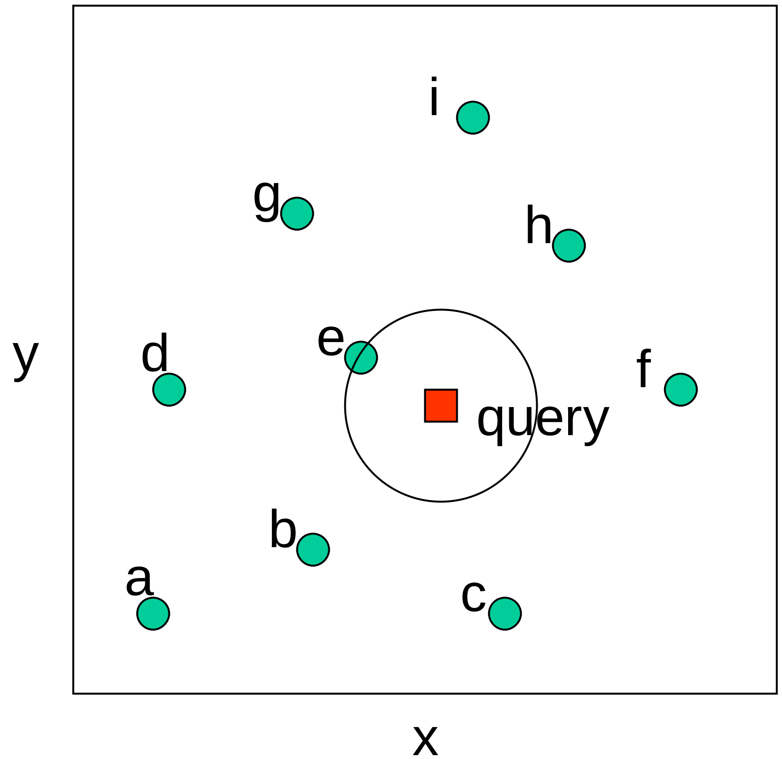


Rectangular query



Circular query

Nearest Neighbor Search

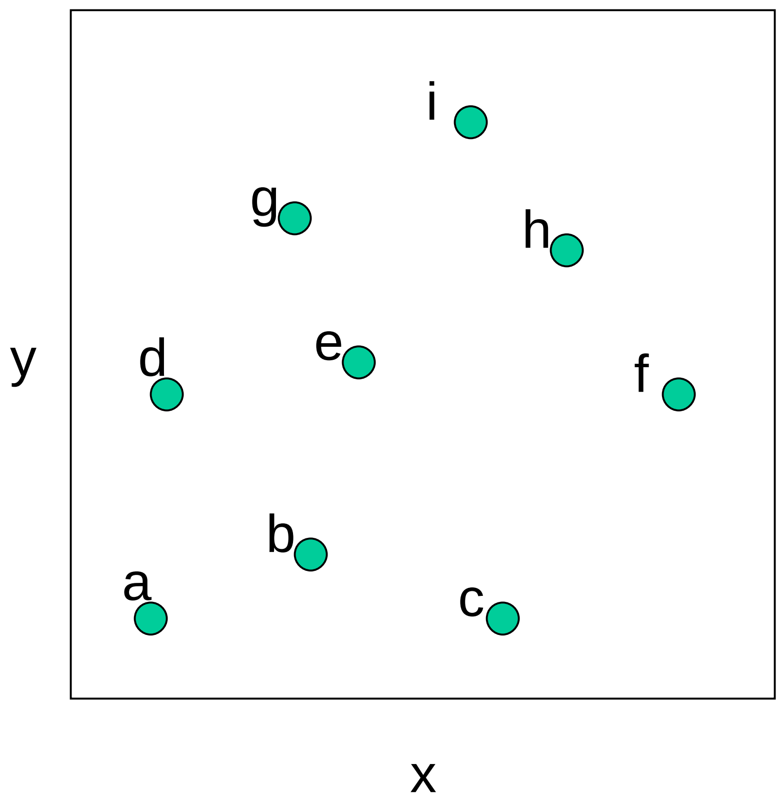


Nearest neighbor is e.

k-d Tree Construction

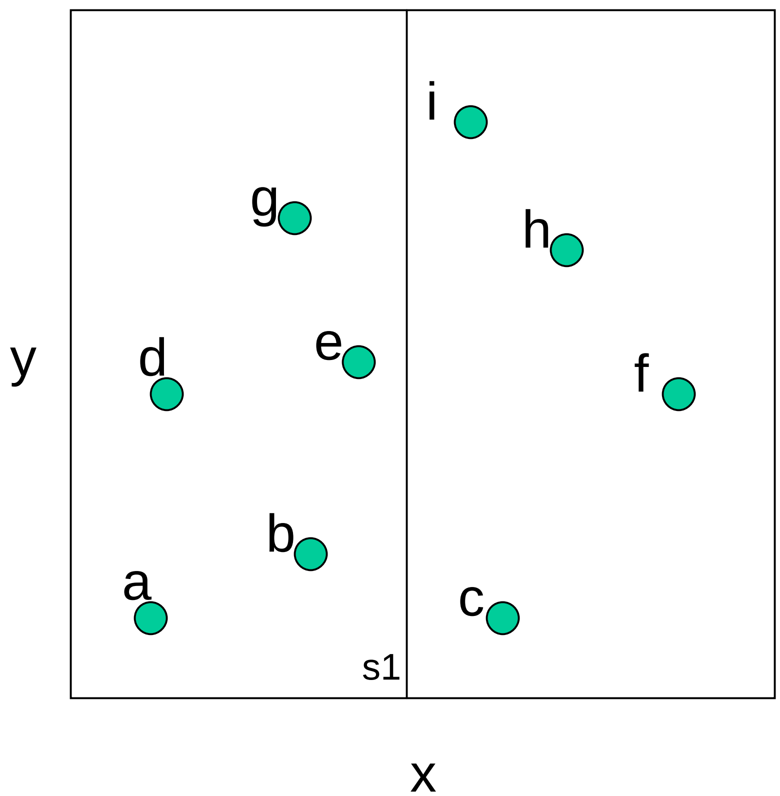
- If there is just one point, form a leaf with that point.
- Otherwise, divide the points in half by a line perpendicular to one of the axes.
- Recursively construct k-d trees for the two sets of points.
- Division strategies
  - divide points perpendicular to the axis with widest spread.
  - divide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

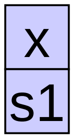
k-d Tree Construction (1)



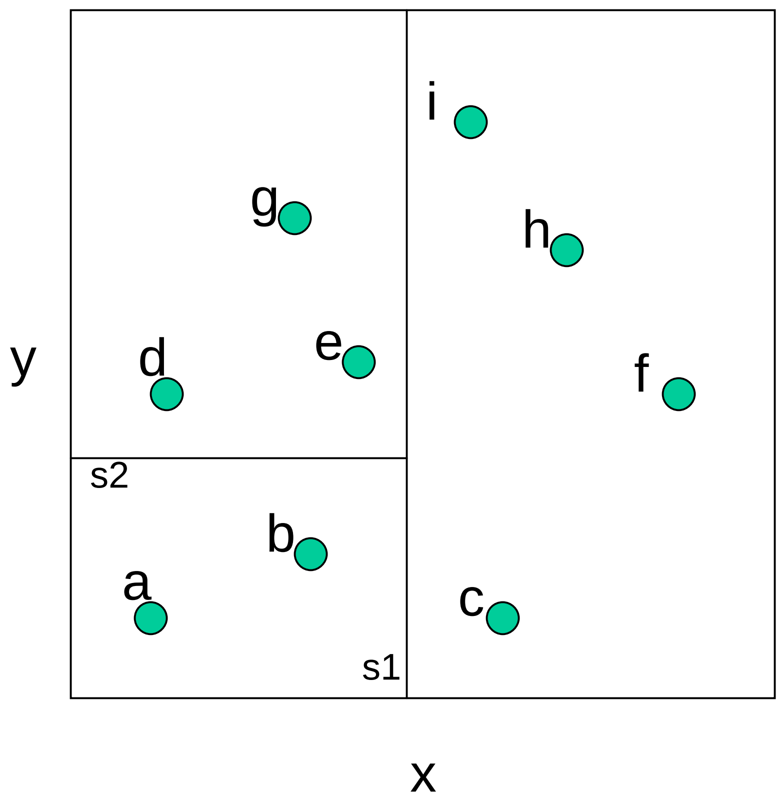
divide perpendicular to the widest spread.

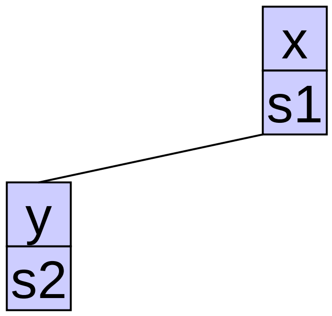
k-d Tree Construction (2)



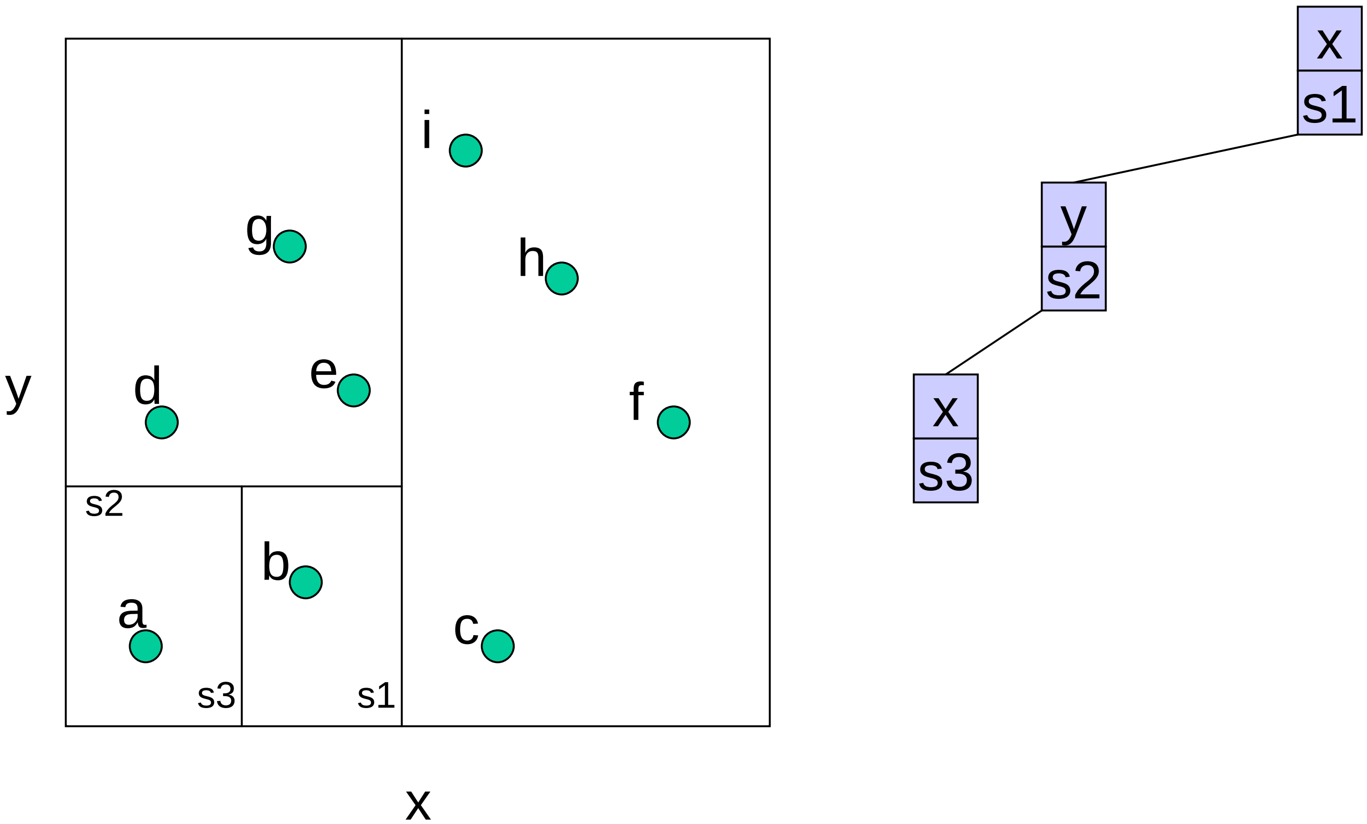


k-d Tree Construction (3)

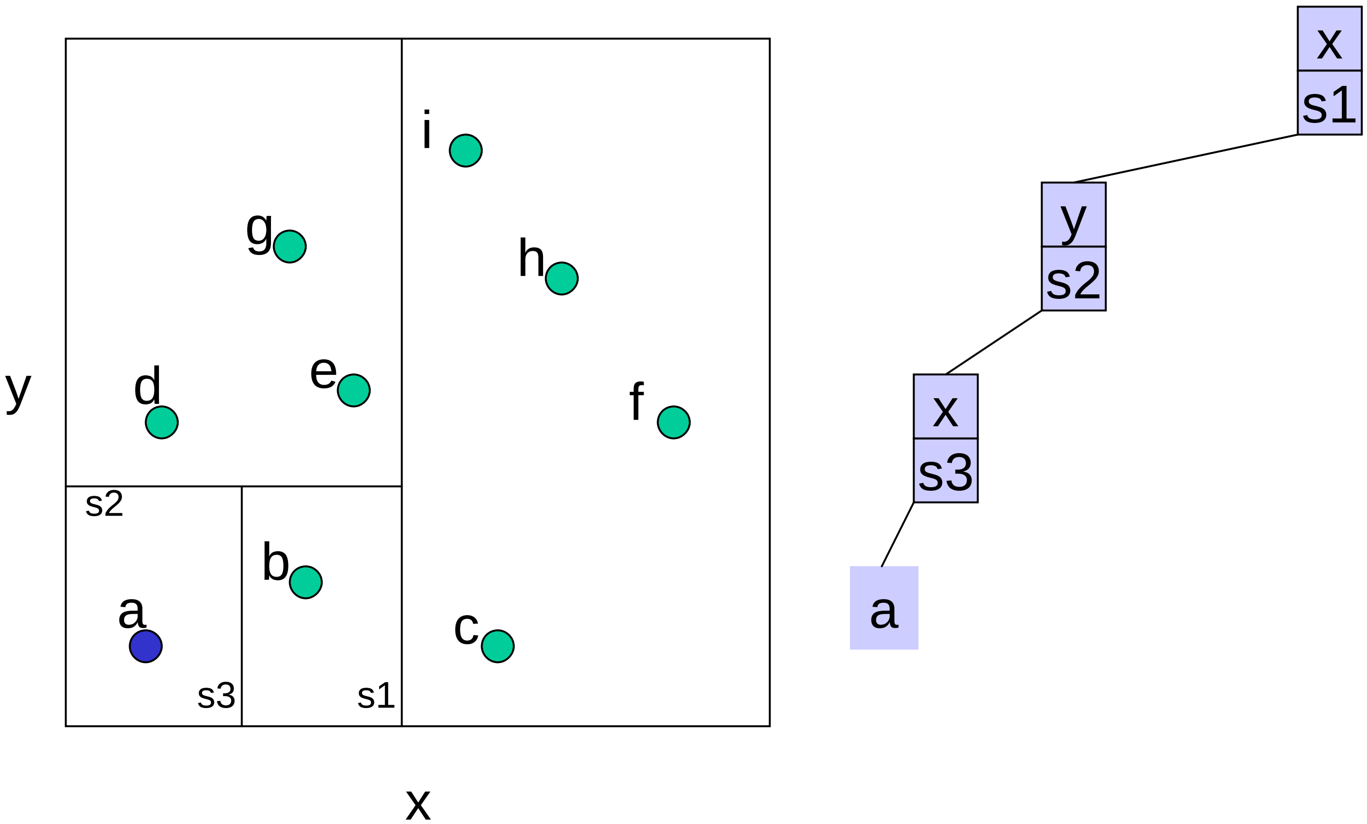




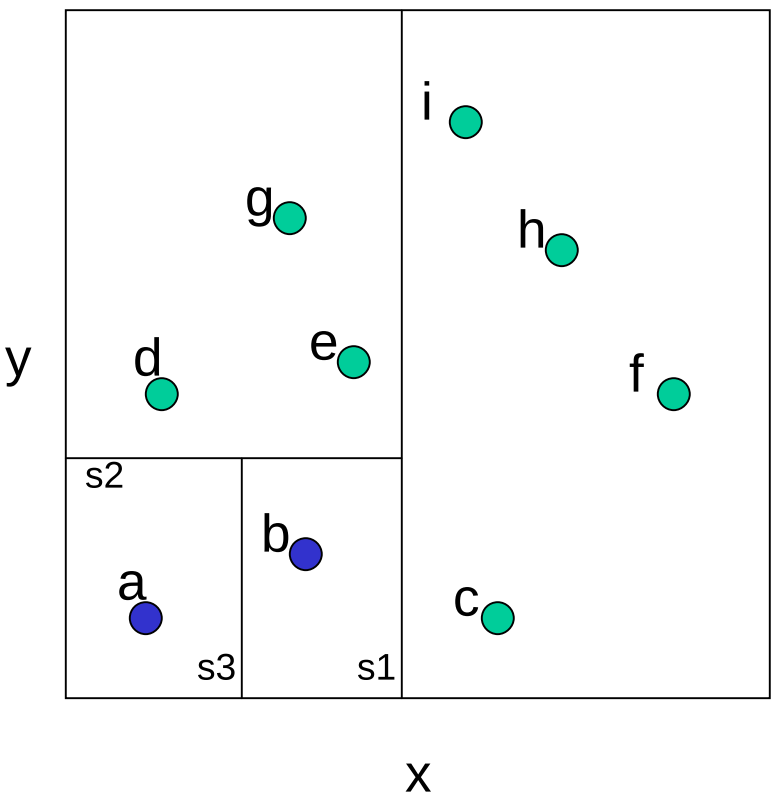
k-d Tree Construction (4)

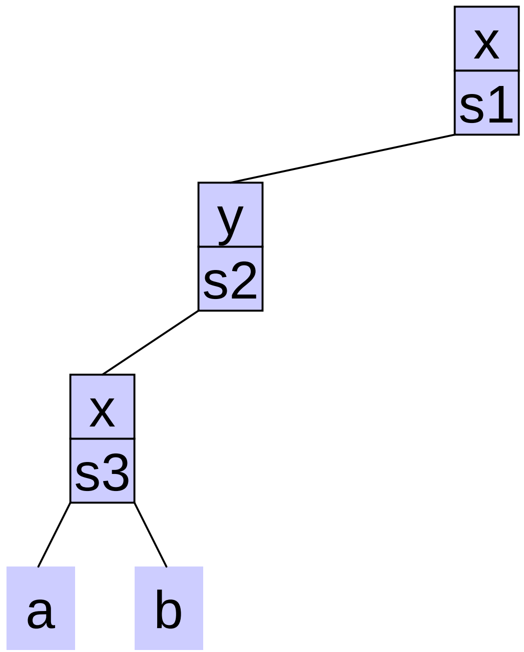


k-d Tree Construction (5)

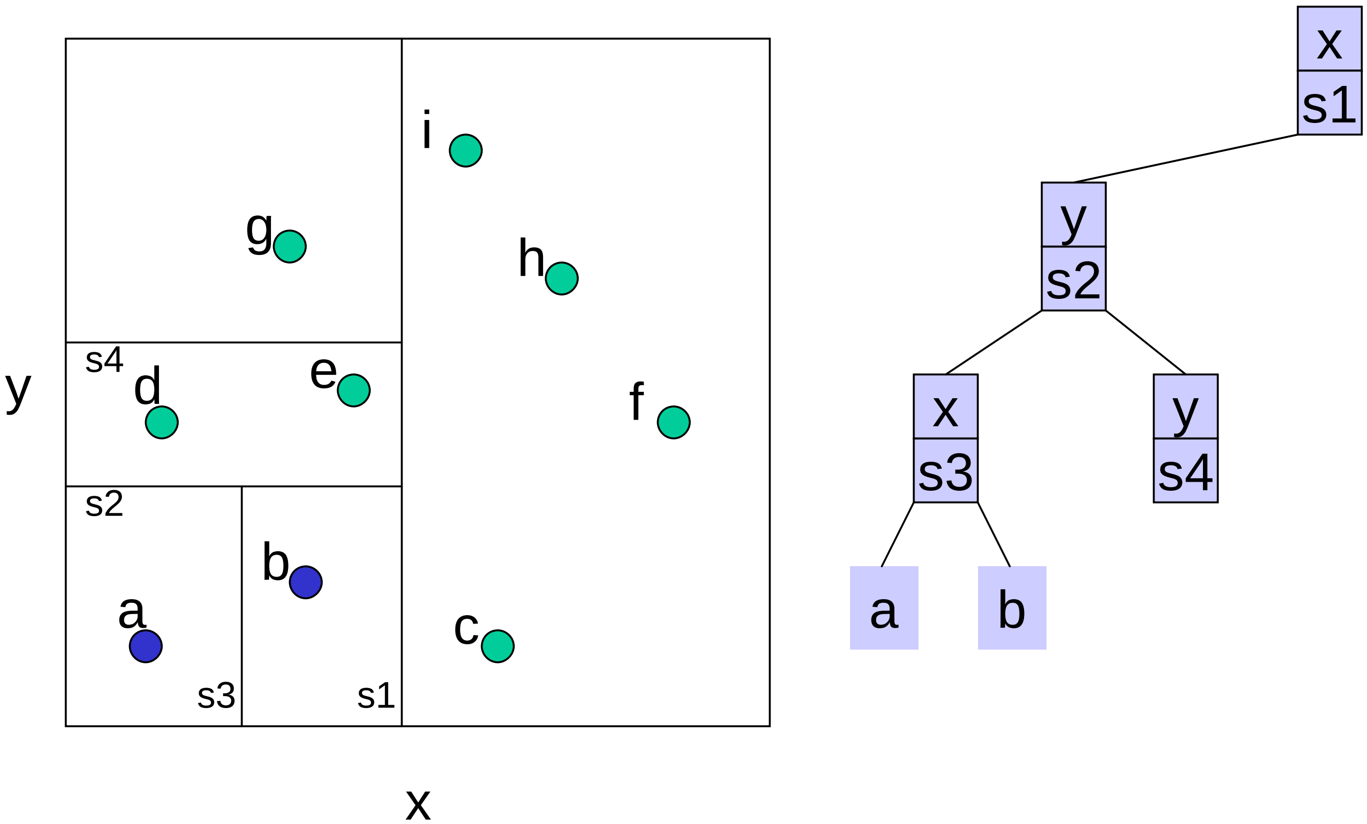


k-d Tree Construction (6)

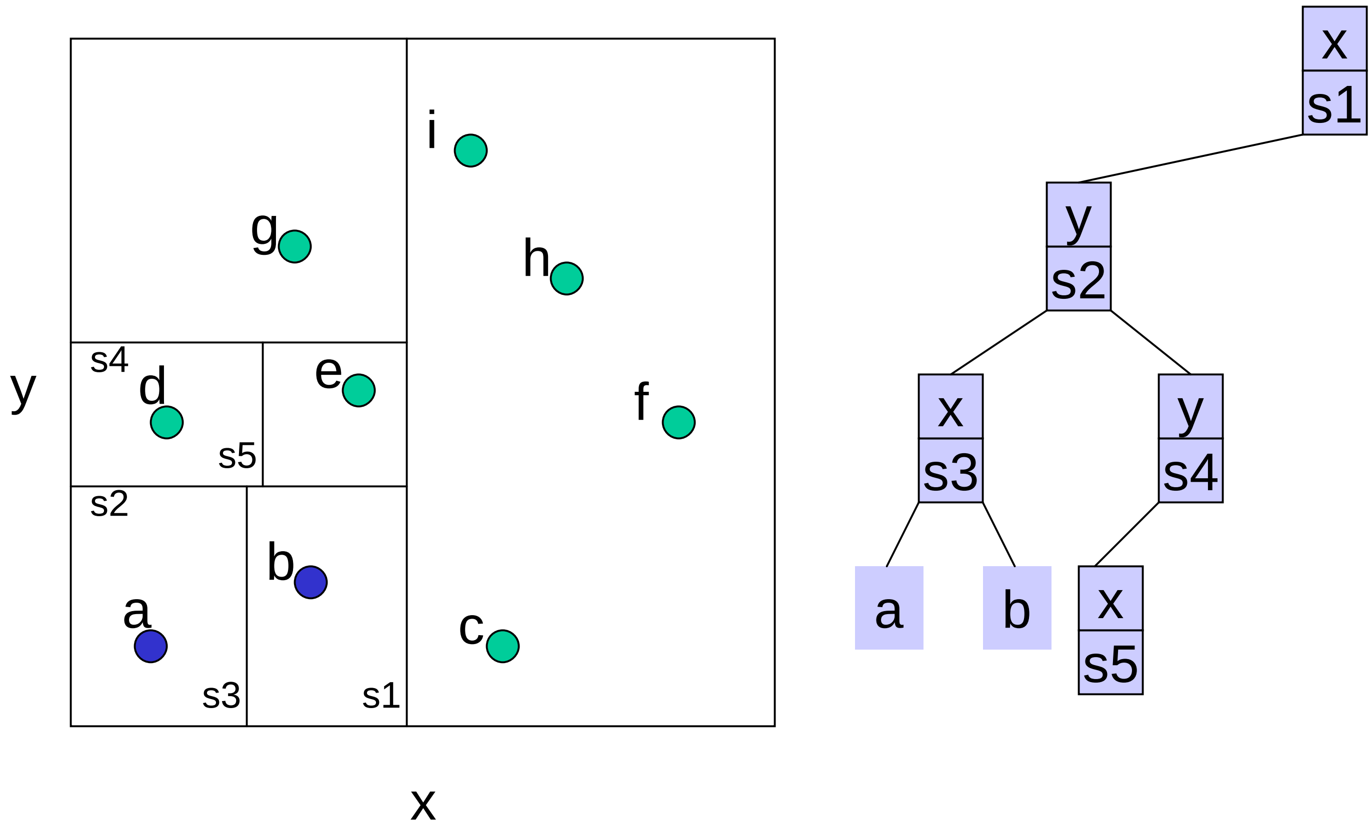




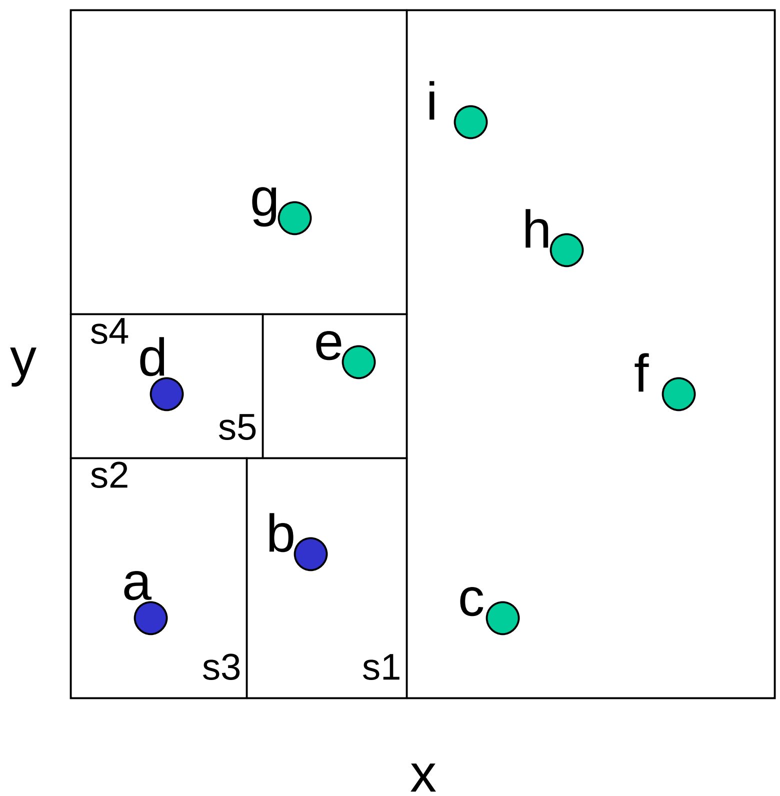
k-d Tree Construction (7)

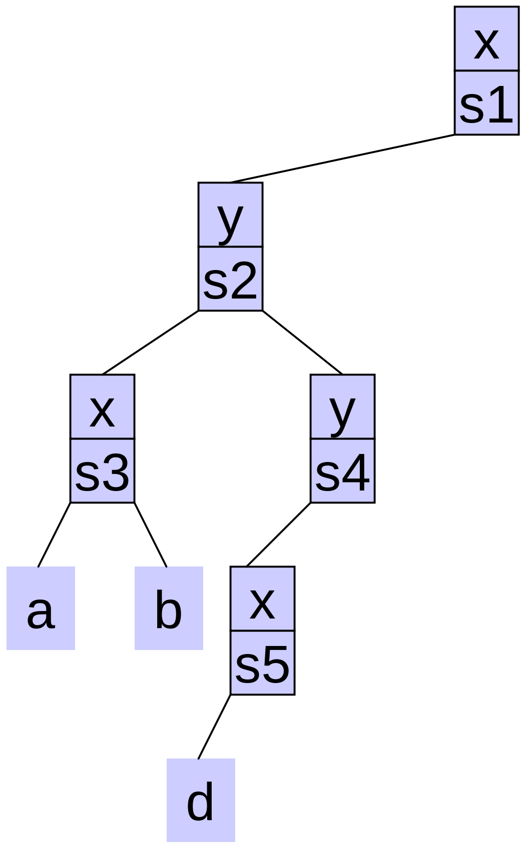


k-d Tree Construction (8)

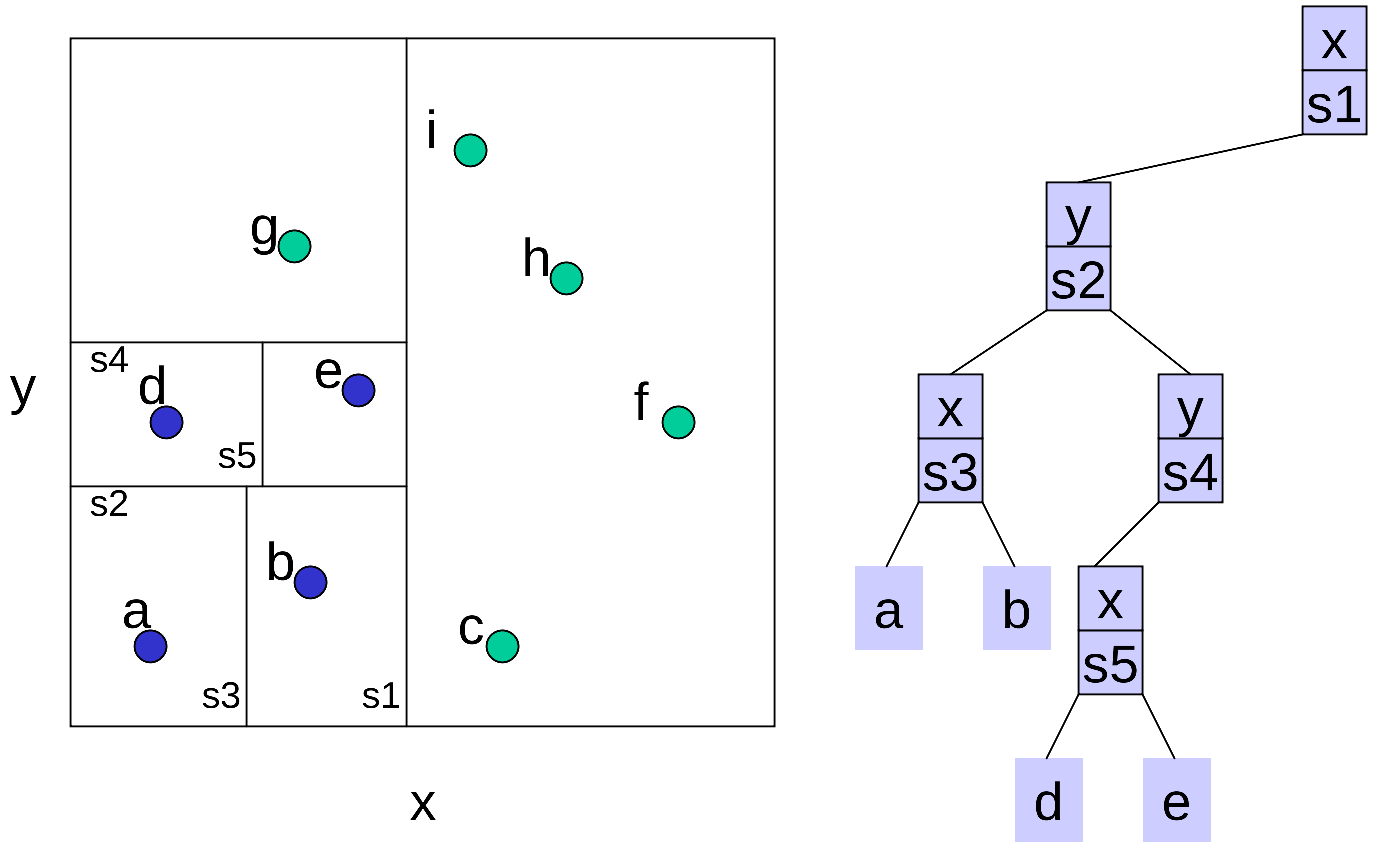


k-d Tree Construction (9)

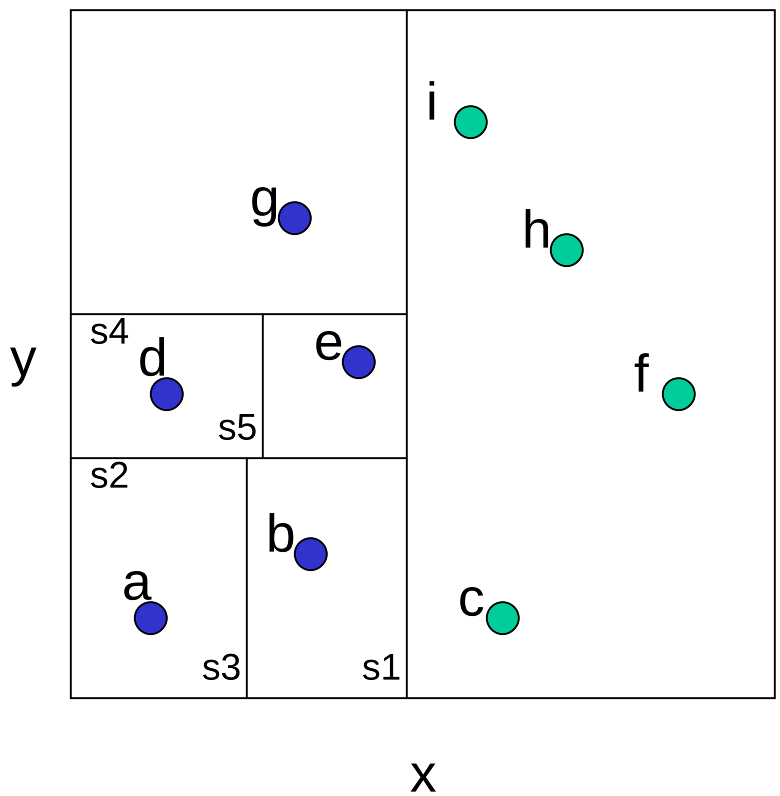


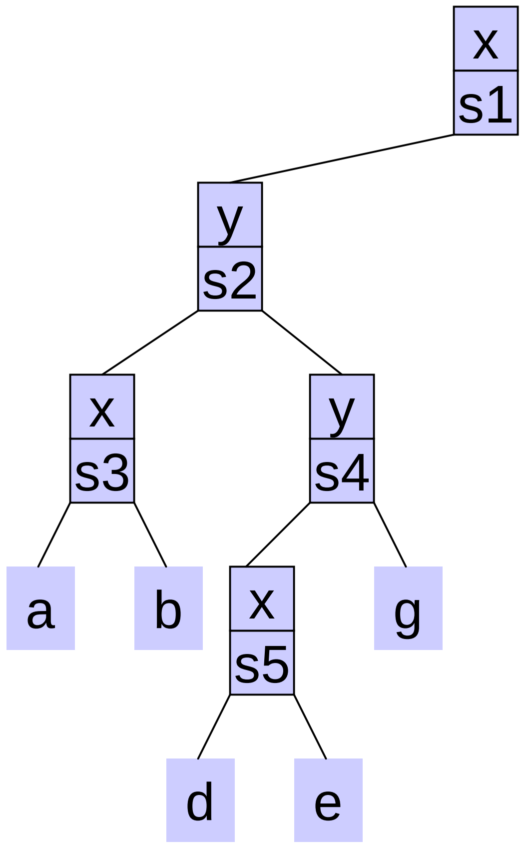


k-d Tree Construction (10)

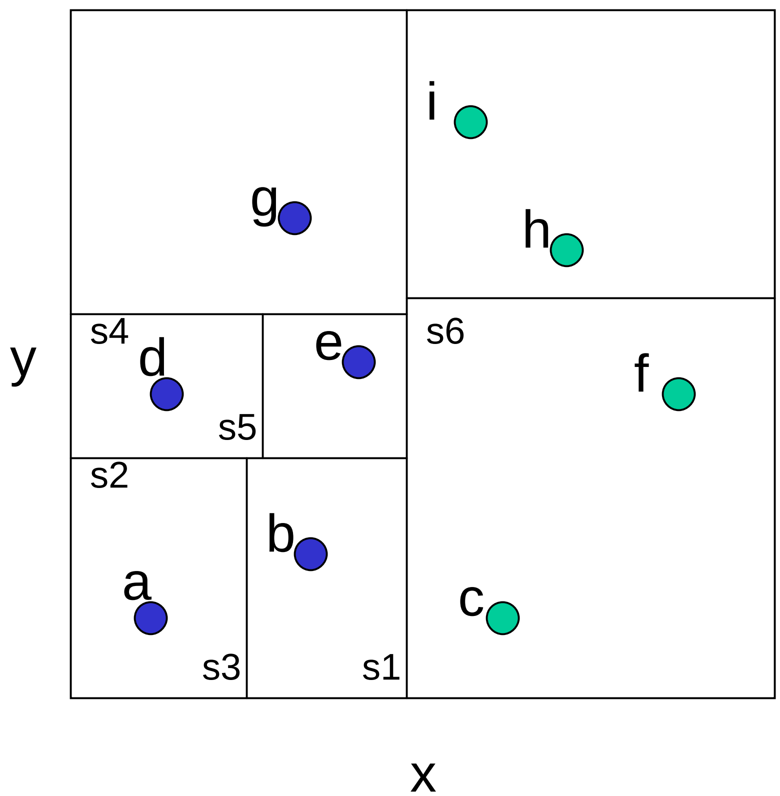


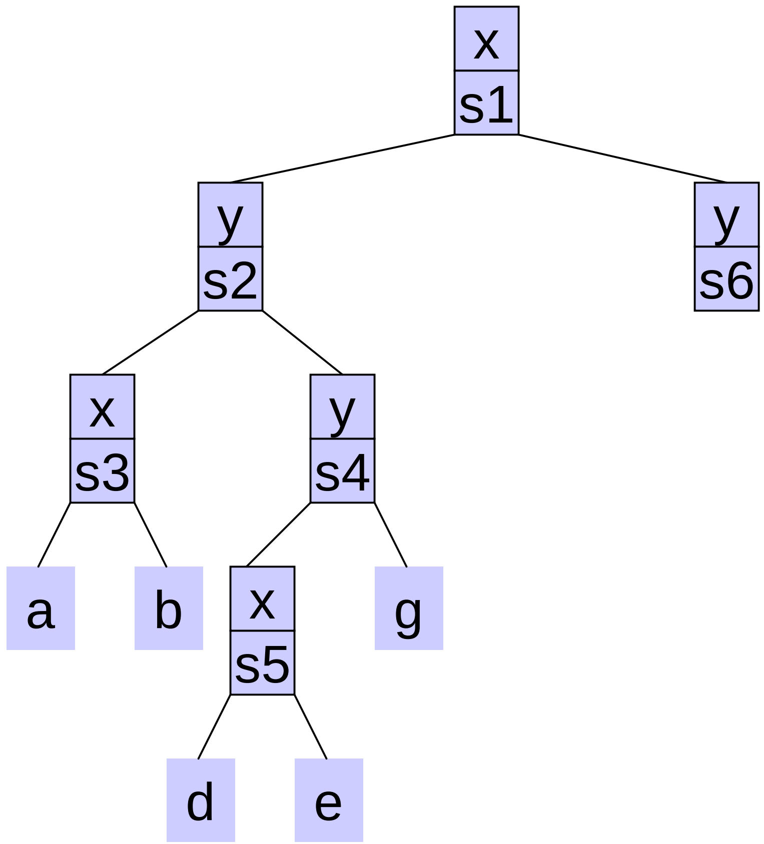
k-d Tree Construction (11)



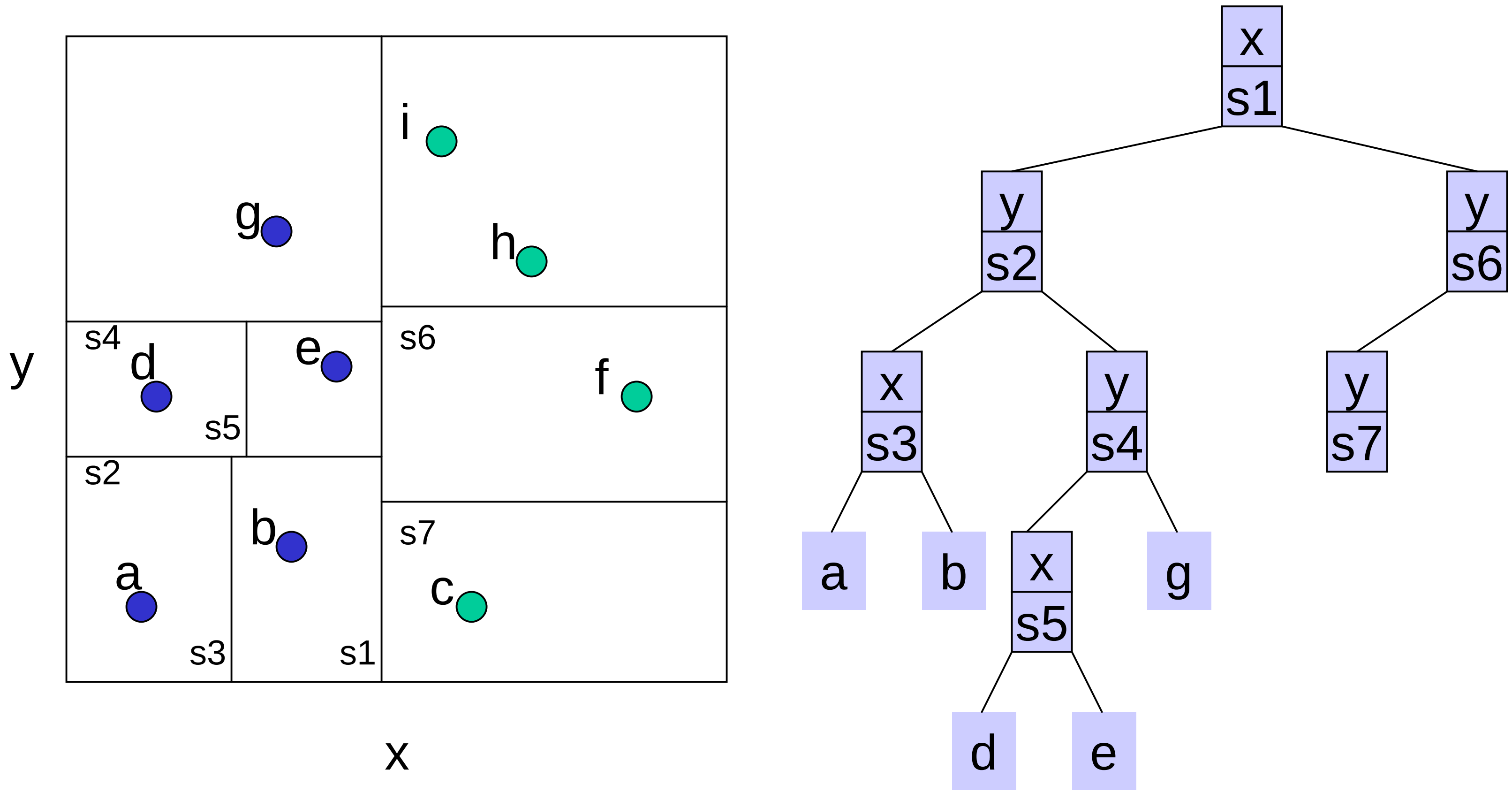


k-d Tree Construction (12)

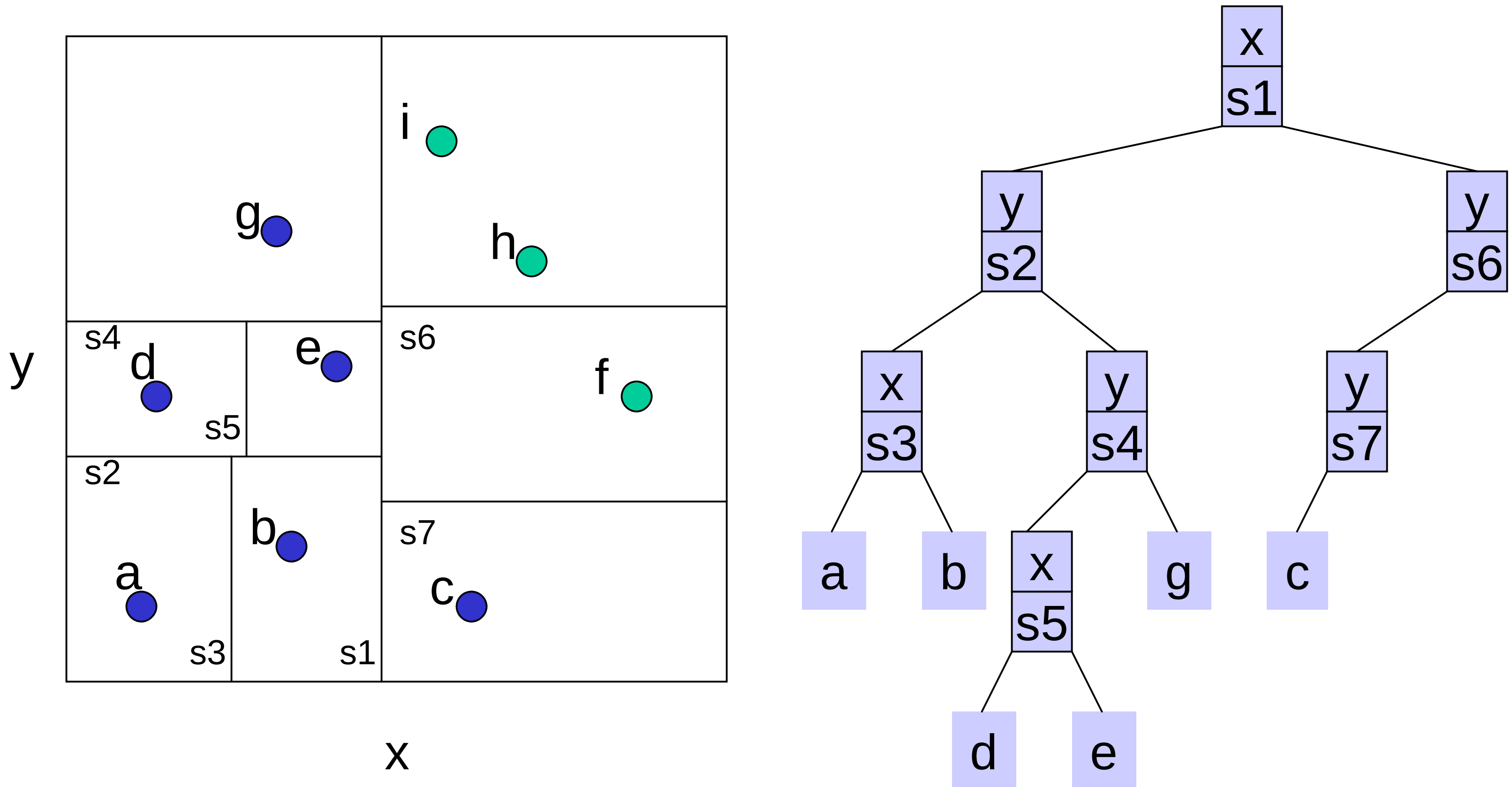




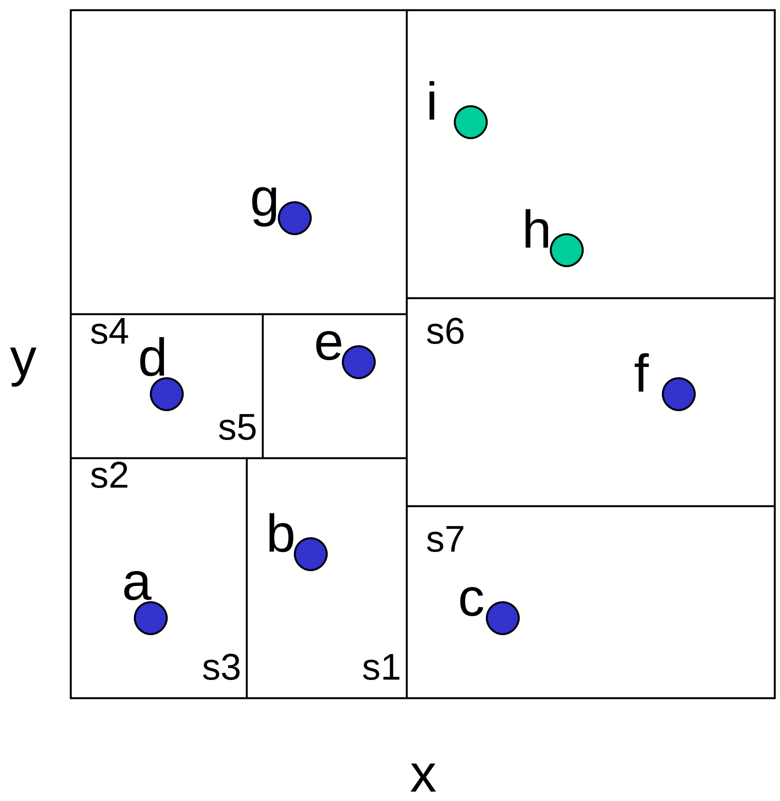
k-d Tree Construction (13)

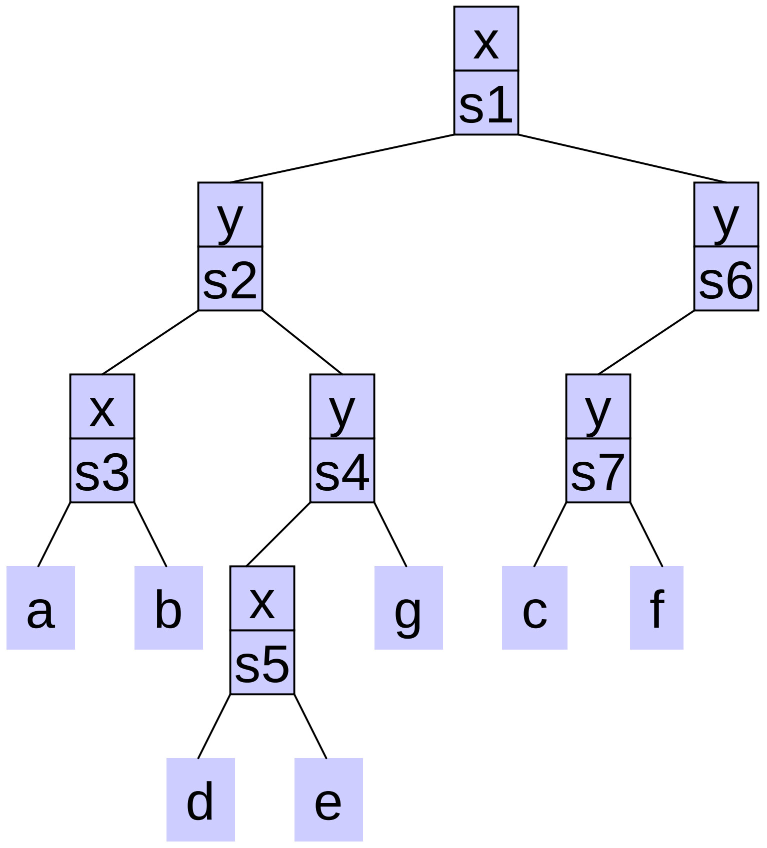


k-d Tree Construction (14)

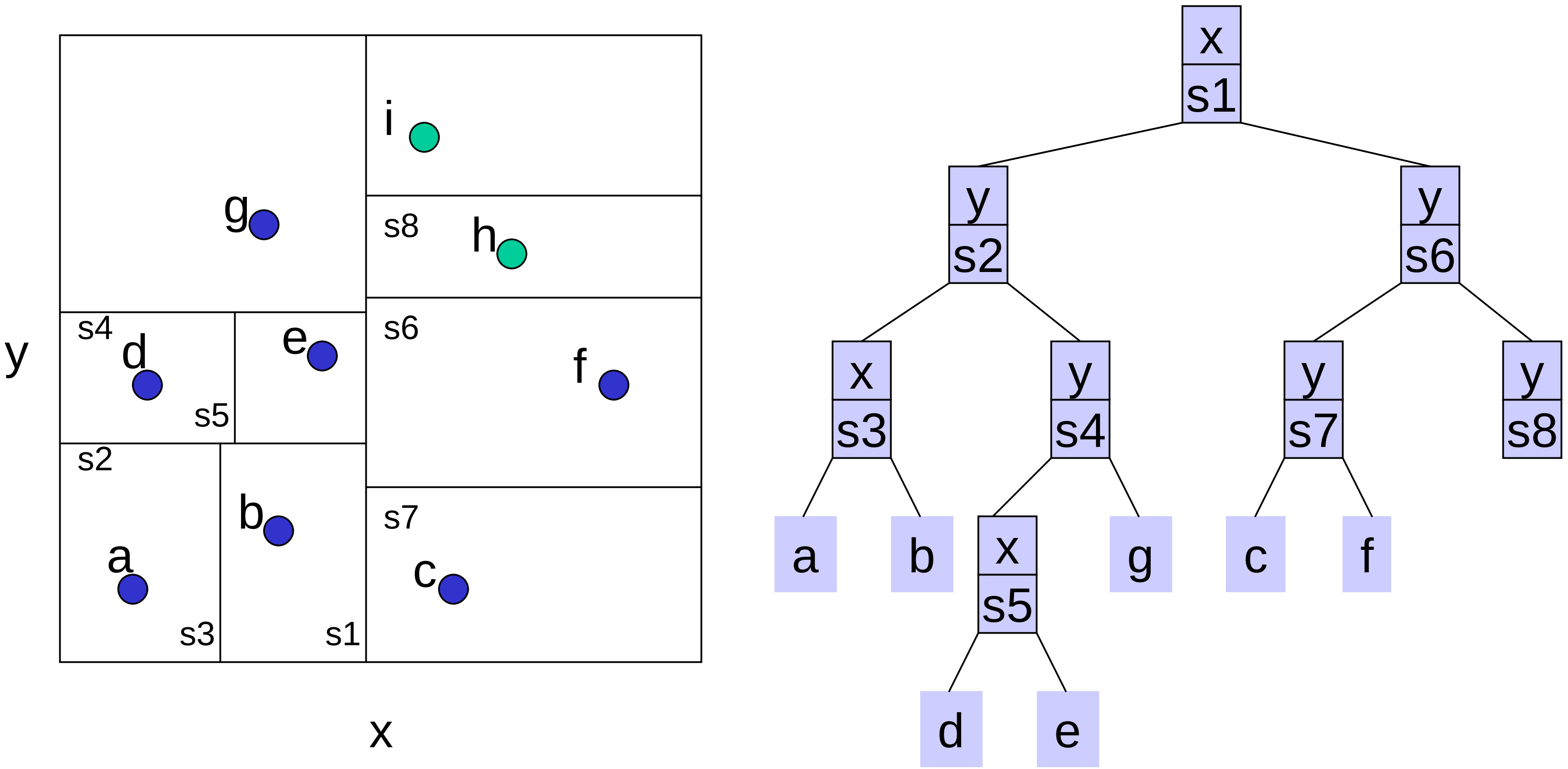


k-d Tree Construction (15)

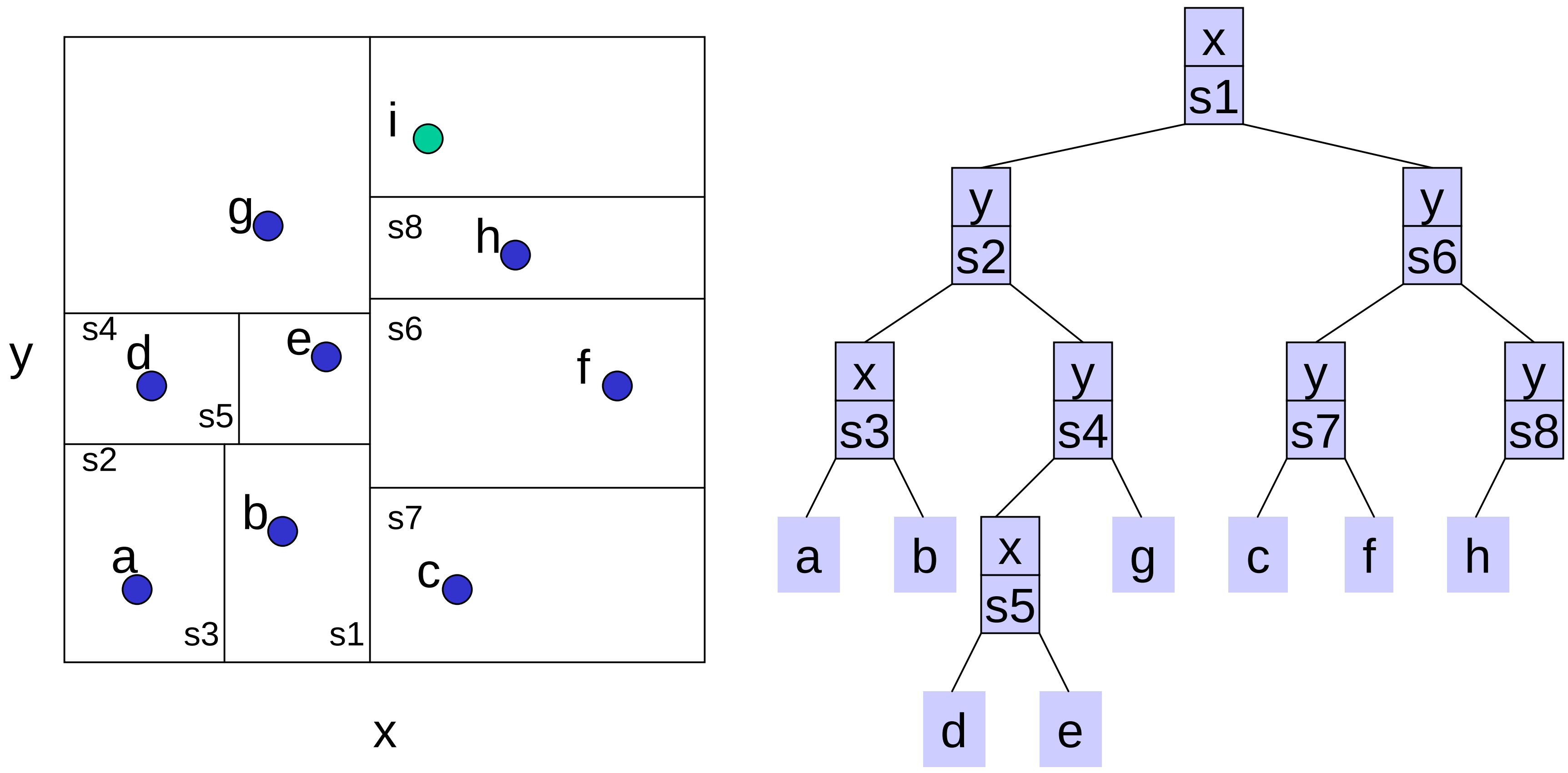




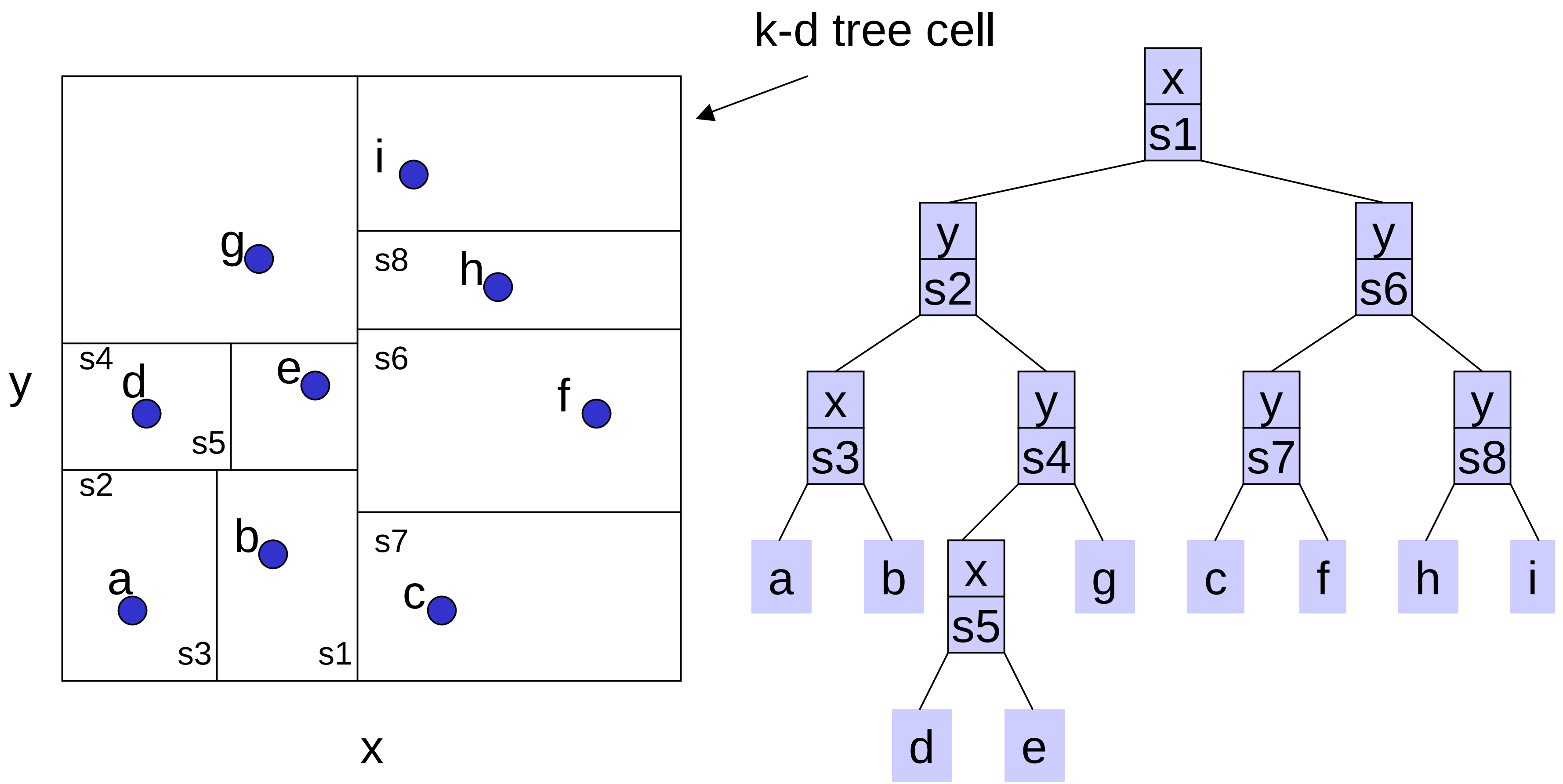
k-d Tree Construction (16)



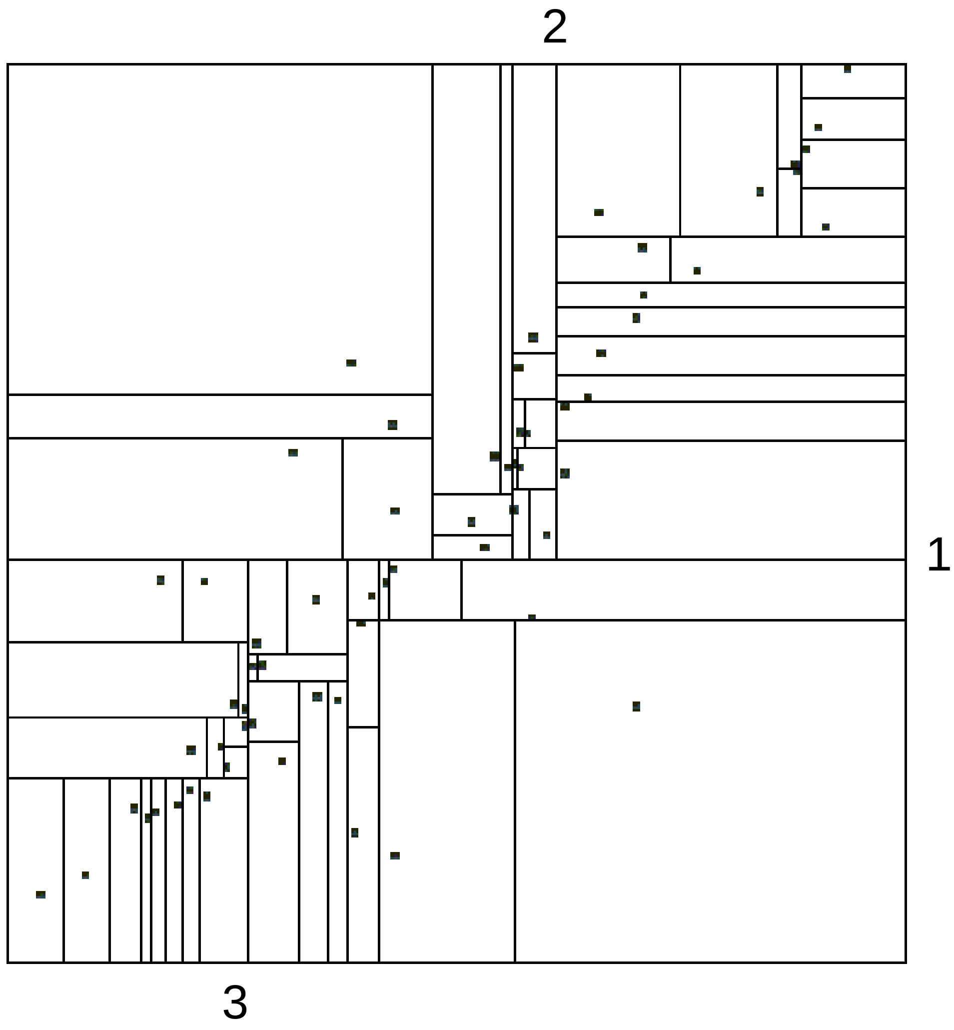
k-d Tree Construction (17)



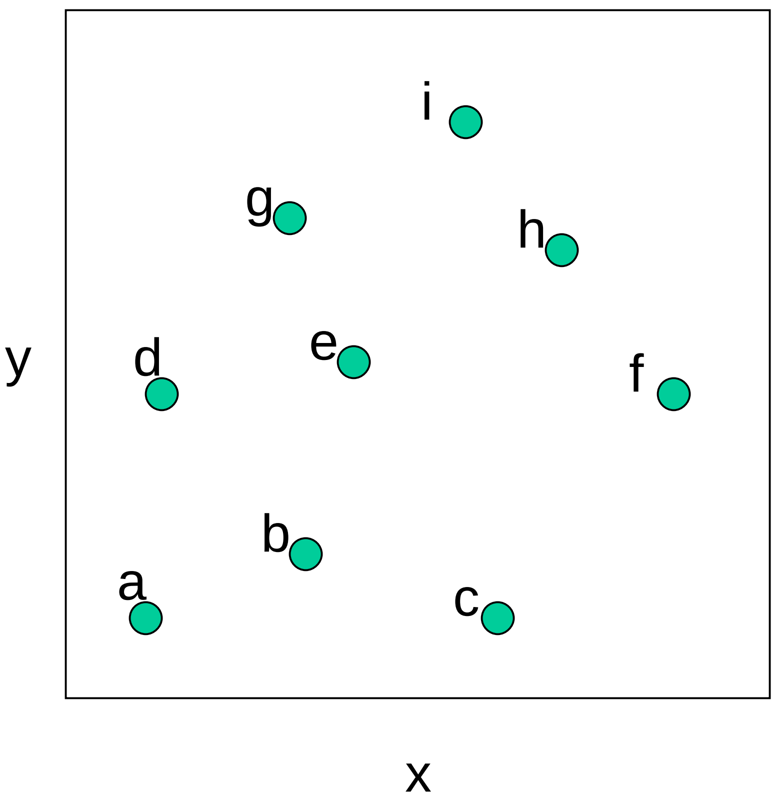
k-d Tree Construction (18)



2-d Tree Decomposition



k-d Tree Splitting

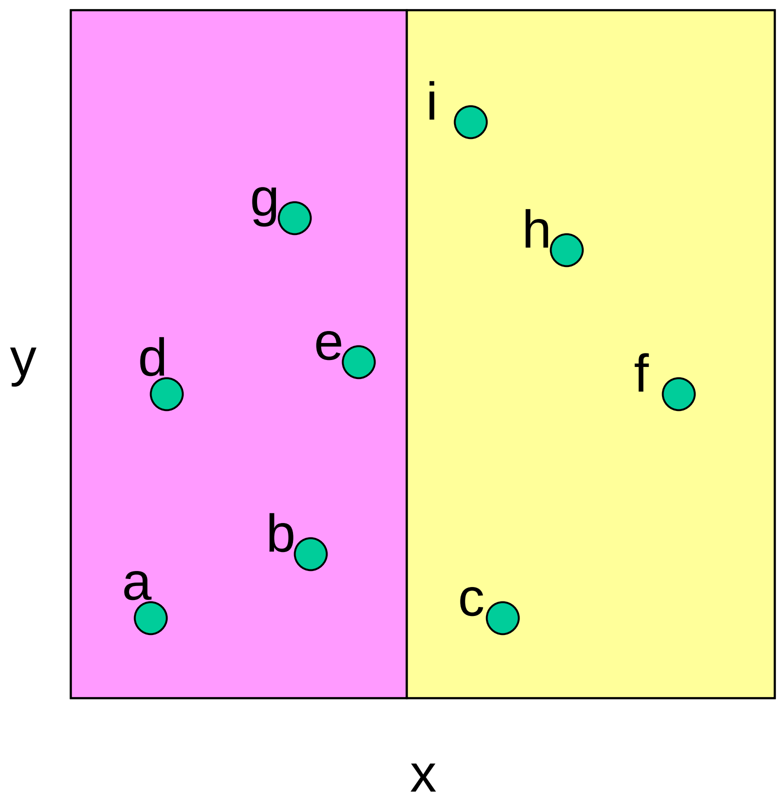


sorted points in each dimension

	1	2	3	4	5	6	7	8	9
x	a	d	g	b	e	i	c	h	f
y	a	c	b	d	f	e	h	g	i

- max spread is the max of $f_x - a_x$ and $i_y - a_y$.
- In the selected dimension the middle point in the list splits the data.
- To build the sorted lists for the other dimensions scan the sorted list adding each point to one of two sorted lists.

k-d Tree Splitting



sorted points in each dimension

	1	2	3	4	5	6	7	8	9
x	a	d	g	b	e	i	c	h	f
y	a	c	b	d	f	e	h	g	i

indicator for each set

a	b	c	d	e	f	g	h	i
0	0	1	0	0	1	0	1	1

scan sorted points in y dimension
and add to correct set

y	a	b	d	e	g	c	f	h	i

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 divide into two subsets can be done in O(dn) time.
- We have the recurrence
  - $T(n,d) \leq 2T(n/2,d) + O(dn)$
- Constructing the k-d tree can be done in O(dn log n) and dn storage

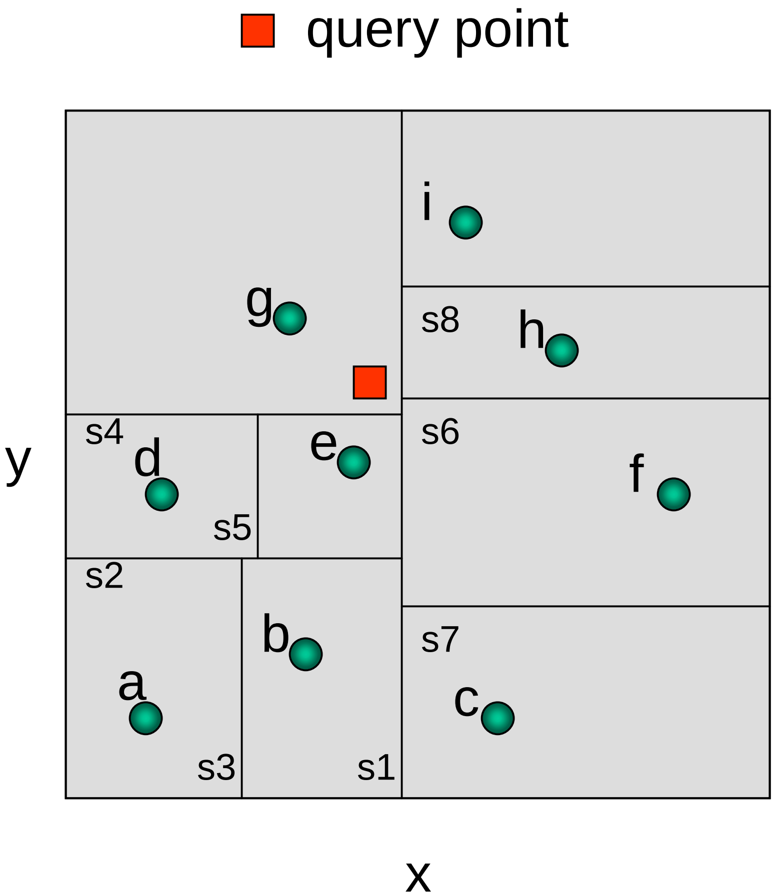
Node Structure for k-d Trees

- 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)

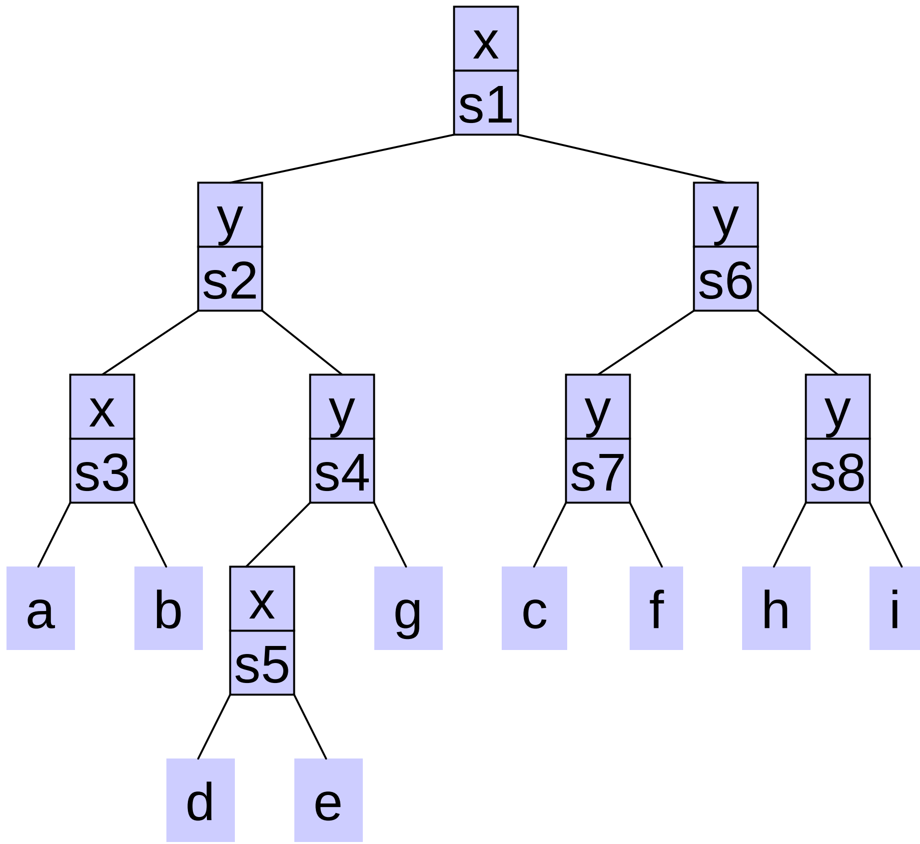
k-d Tree Nearest Neighbor Search

- Search recursively to find the point in the same cell as the query.
- On the return search each subtree where a closer point than the one you already know about might be found.

k-d Tree NNS (1)

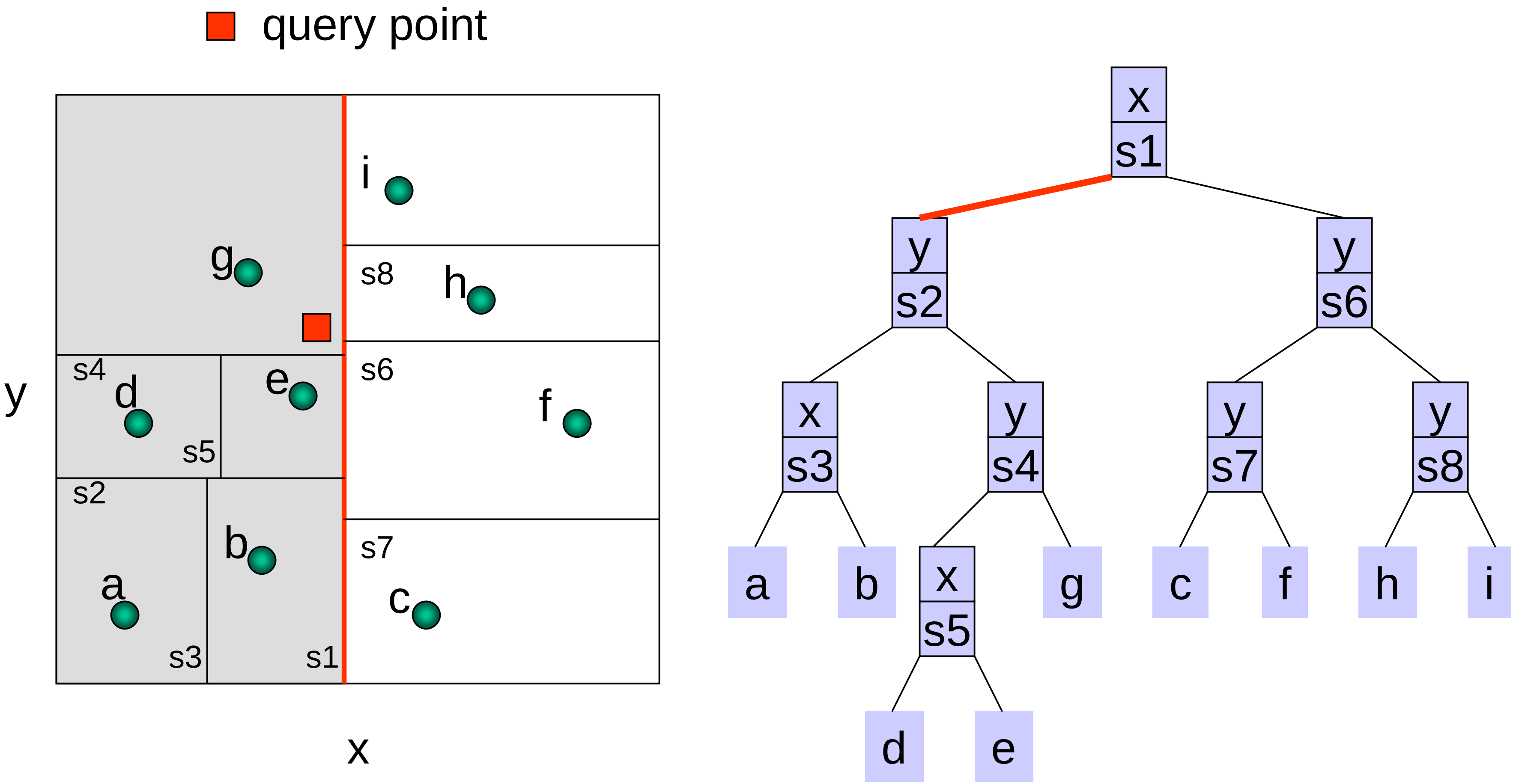


query point
i
g
s8
h
s4
d
e
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x



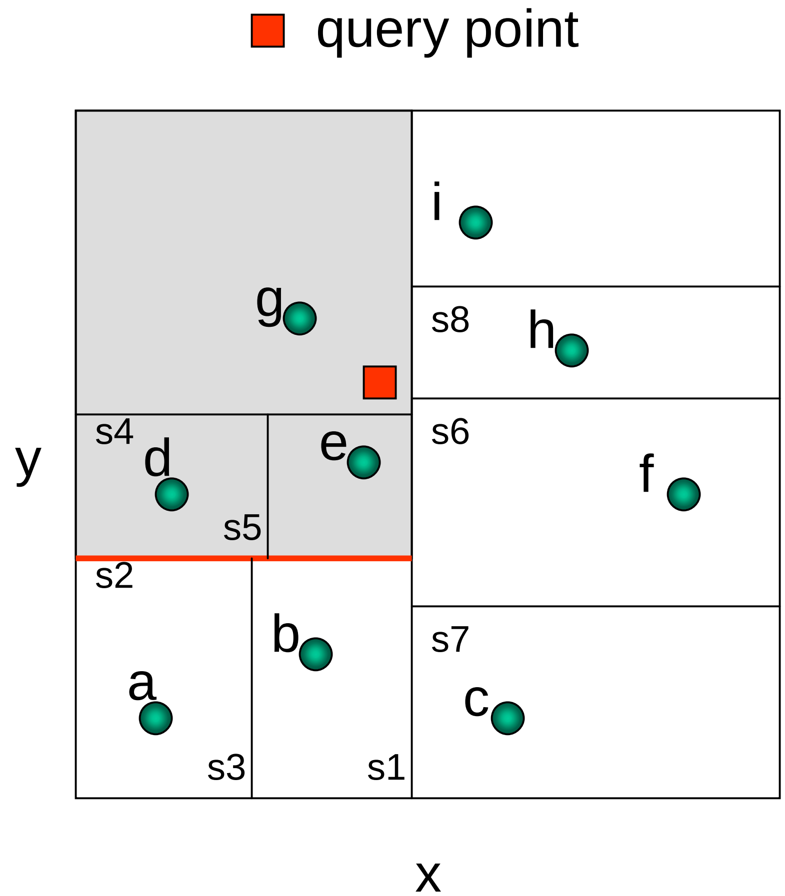
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

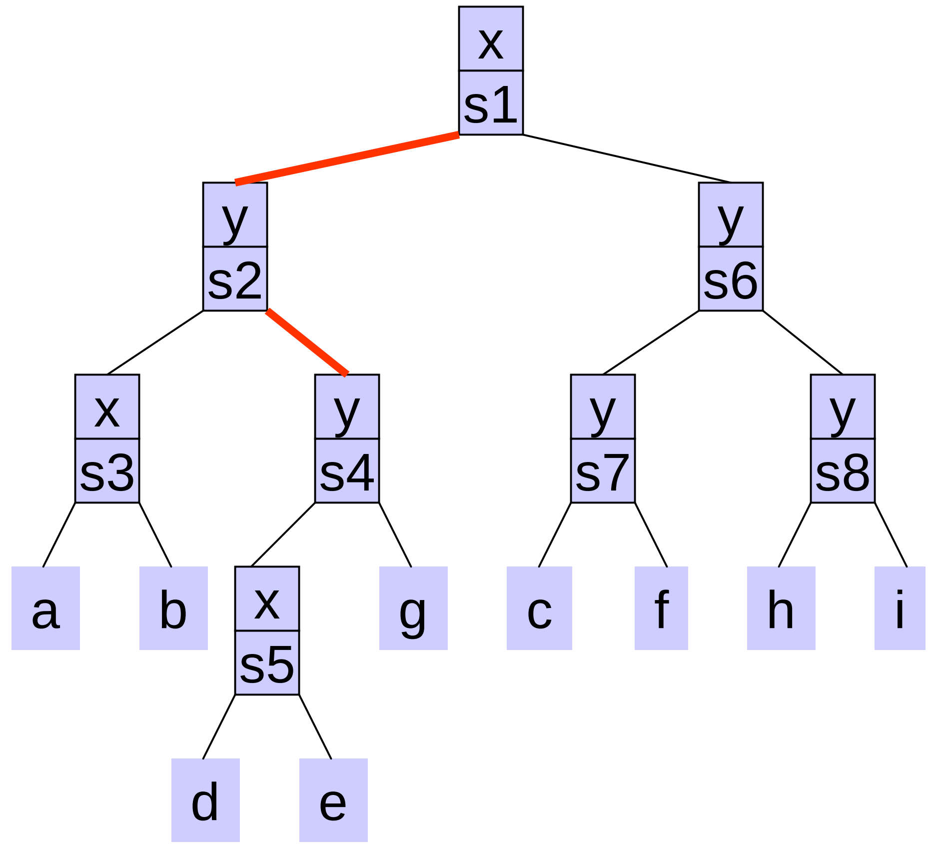
k-d Tree NNS (2)



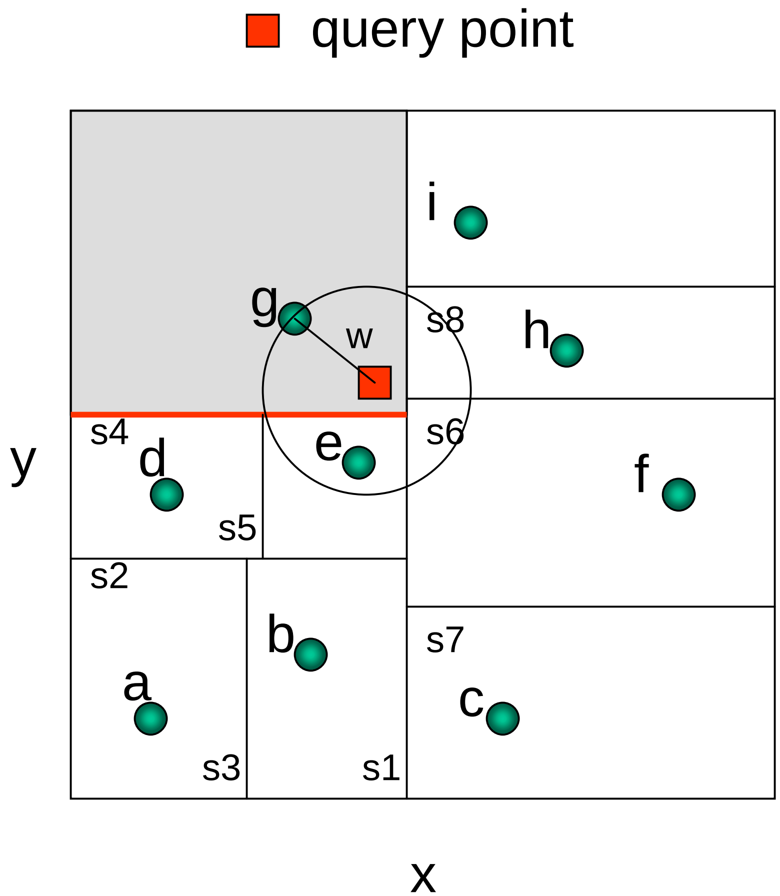
query point
i
g
s8
h
s4
d
e
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (3)

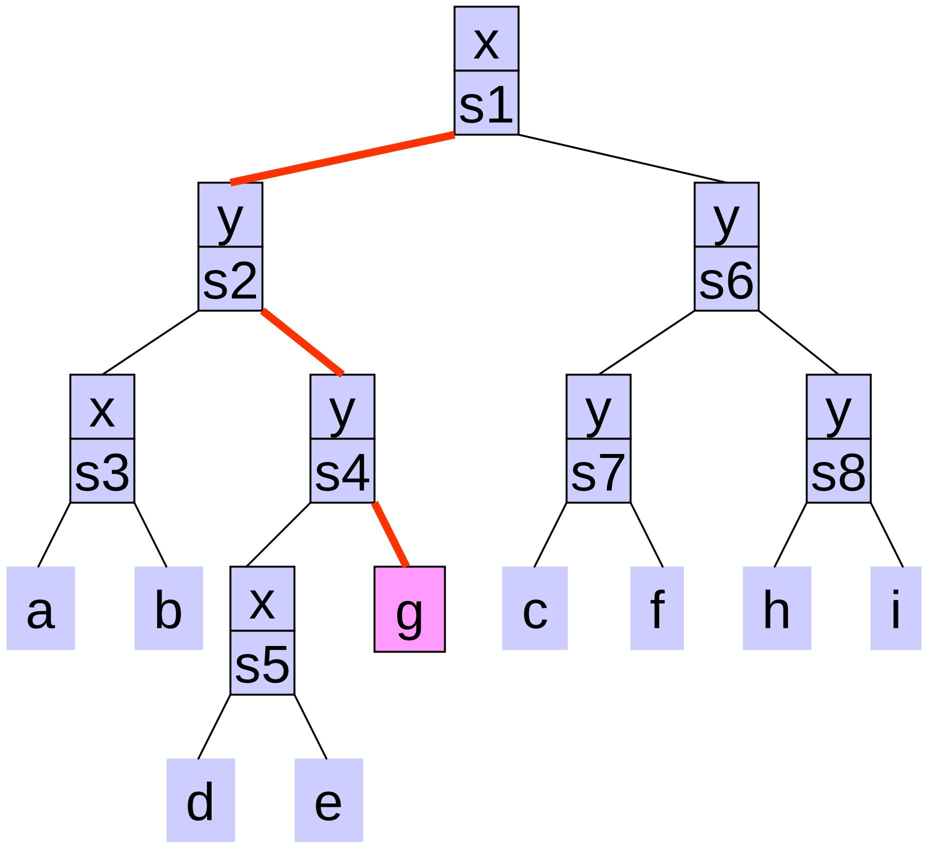

query point
i
g
s8
h
s4
d
e
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x


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 (4)

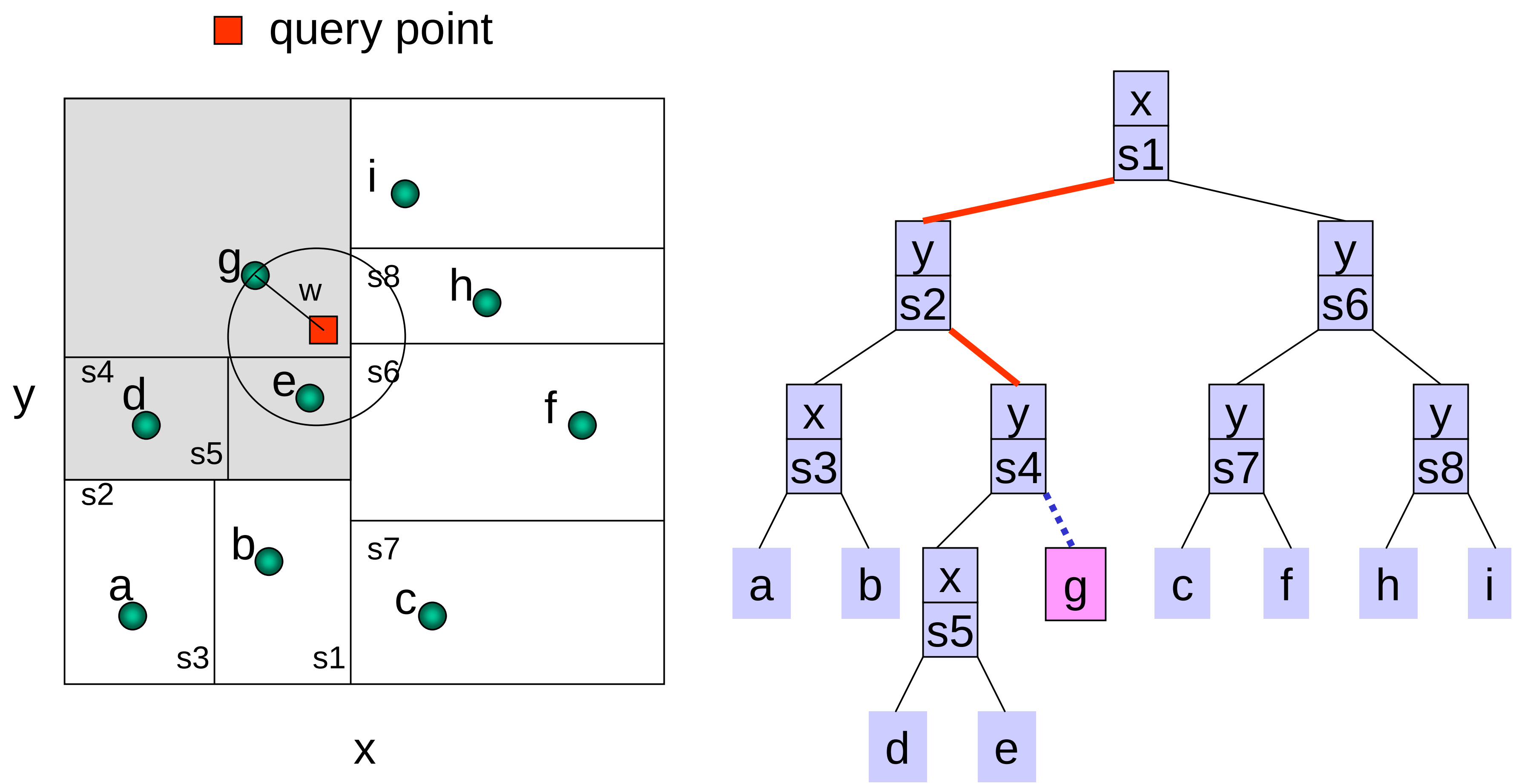


query point
i
g
w
s8
h
s4
d
e
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x



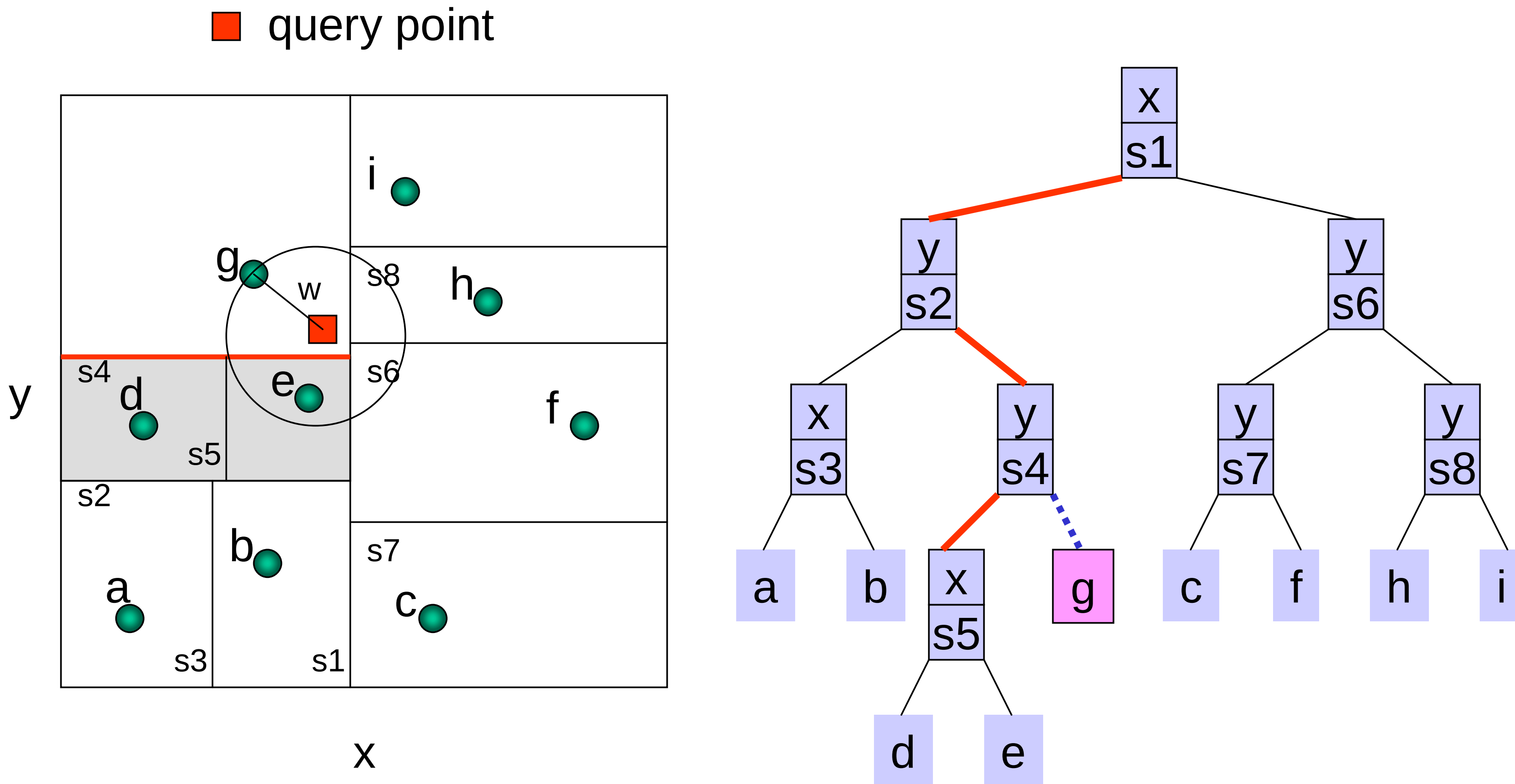
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (5)



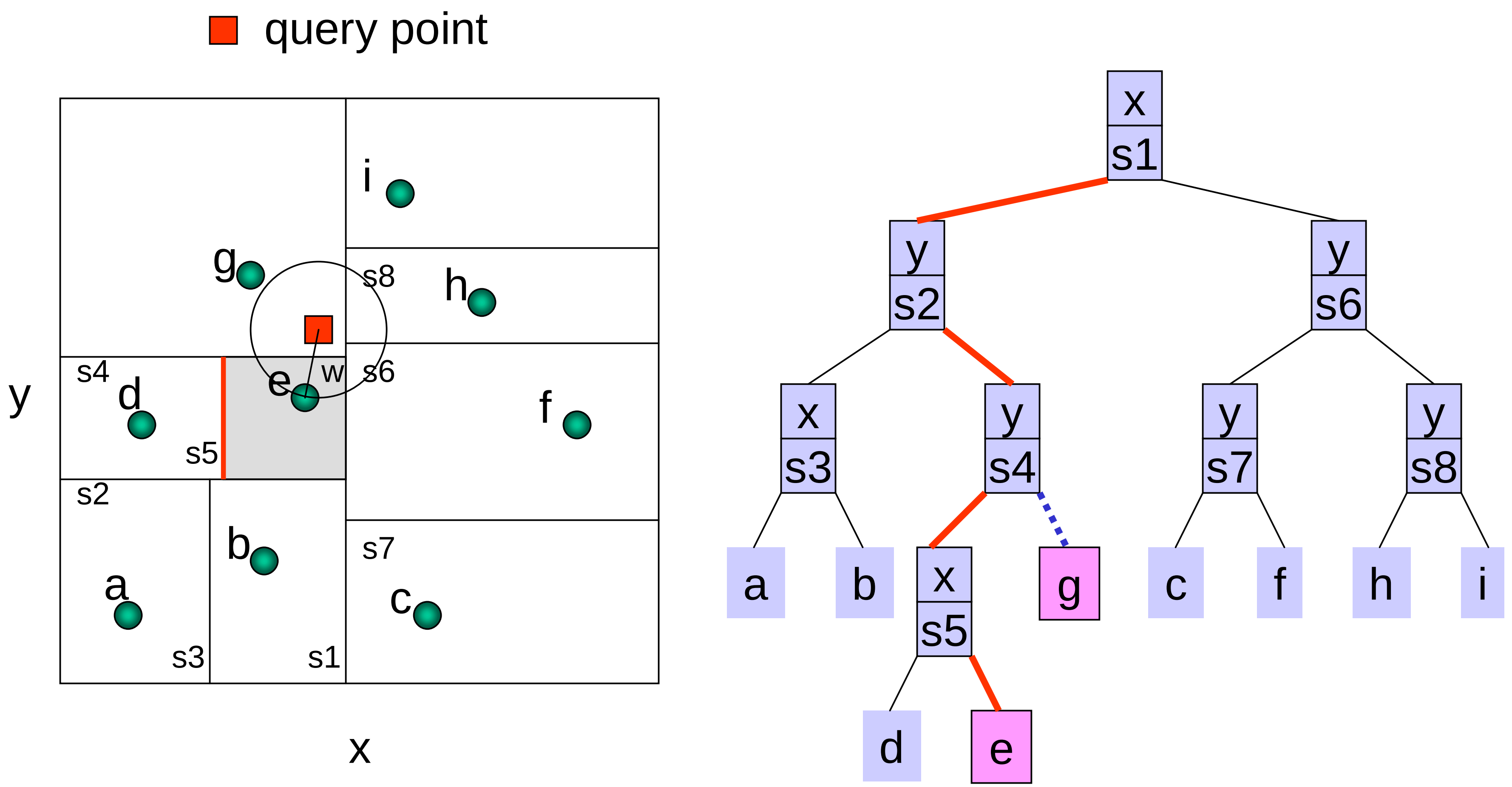
query point
i
g
w
s8
h
s4
d
e
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (6)



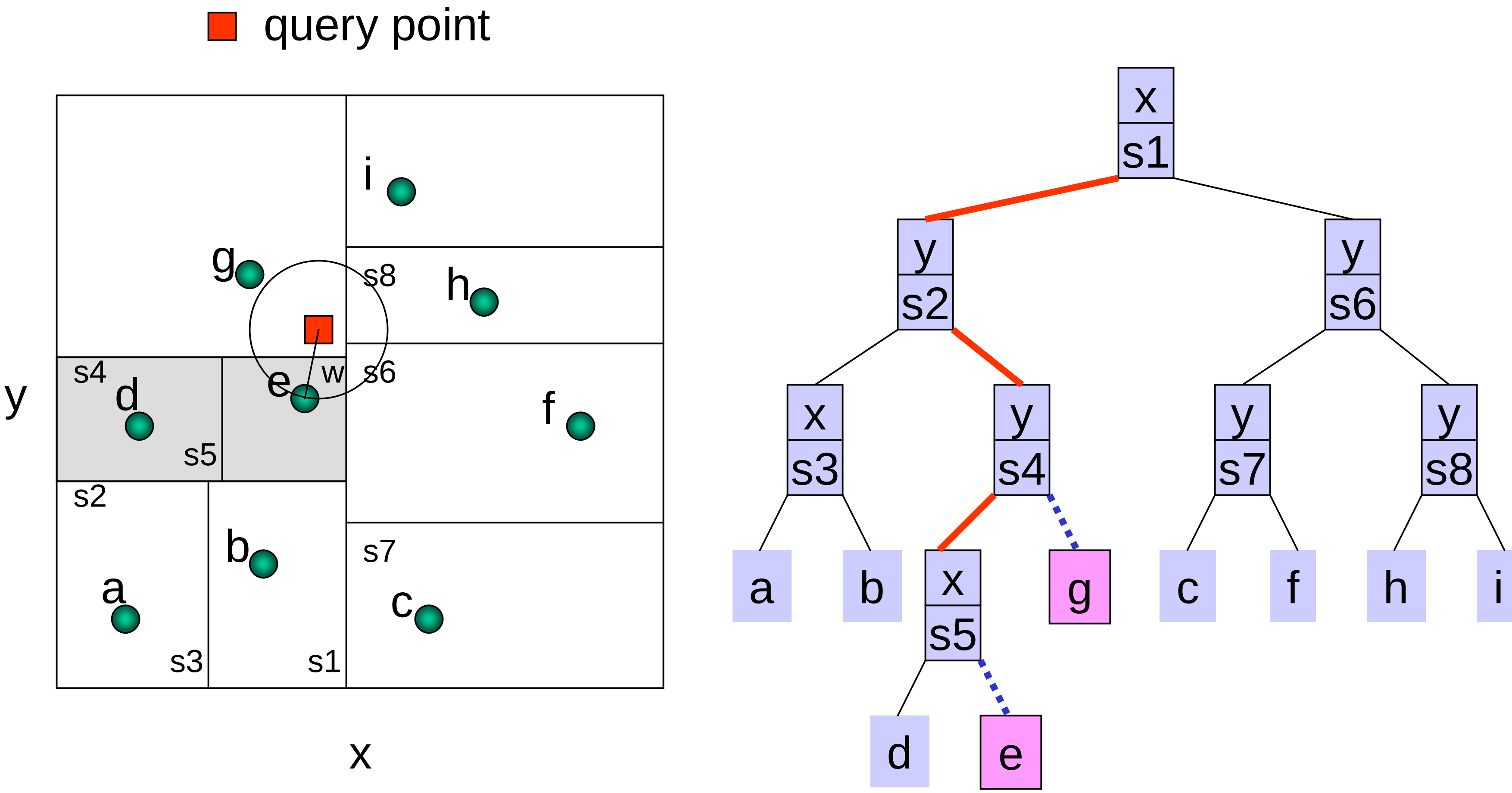
query point
i
g
w
s8
h
s4
d
e
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (7)

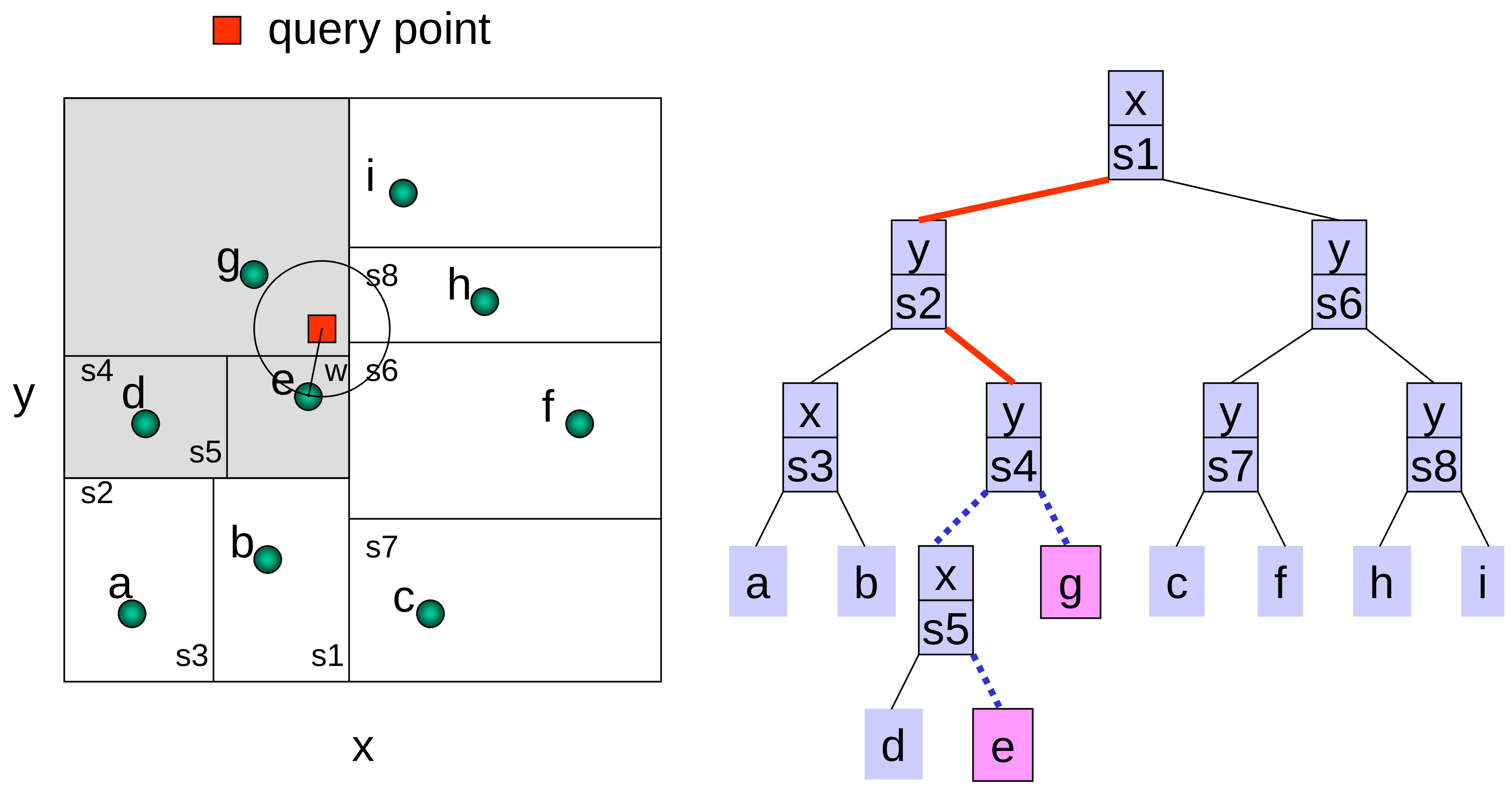


query point
i
g
s8
h
s4
d
e
w
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (8)

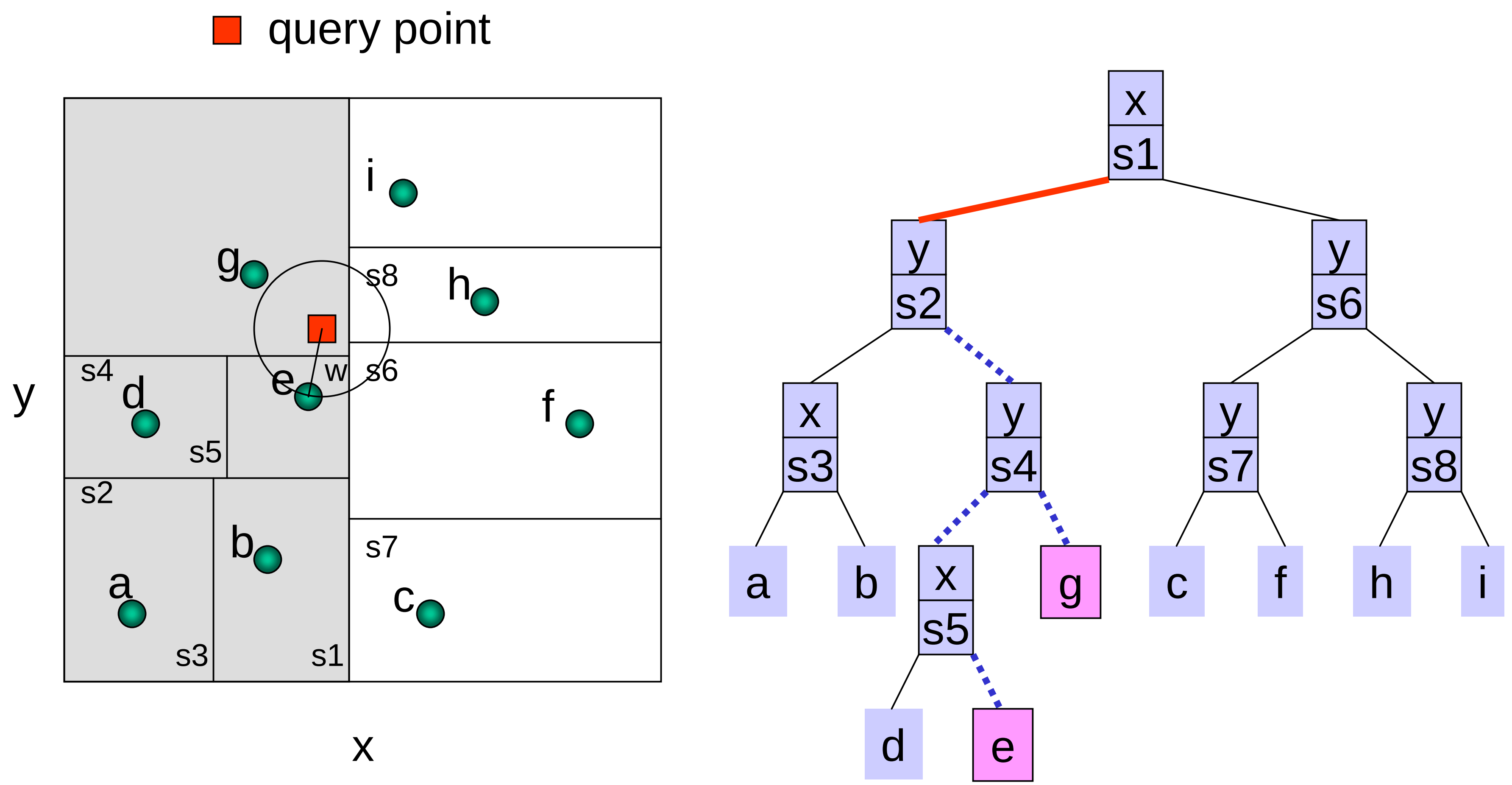

query point
i
g
s8
h
s4
d
e
w
s6
y
f
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (9)

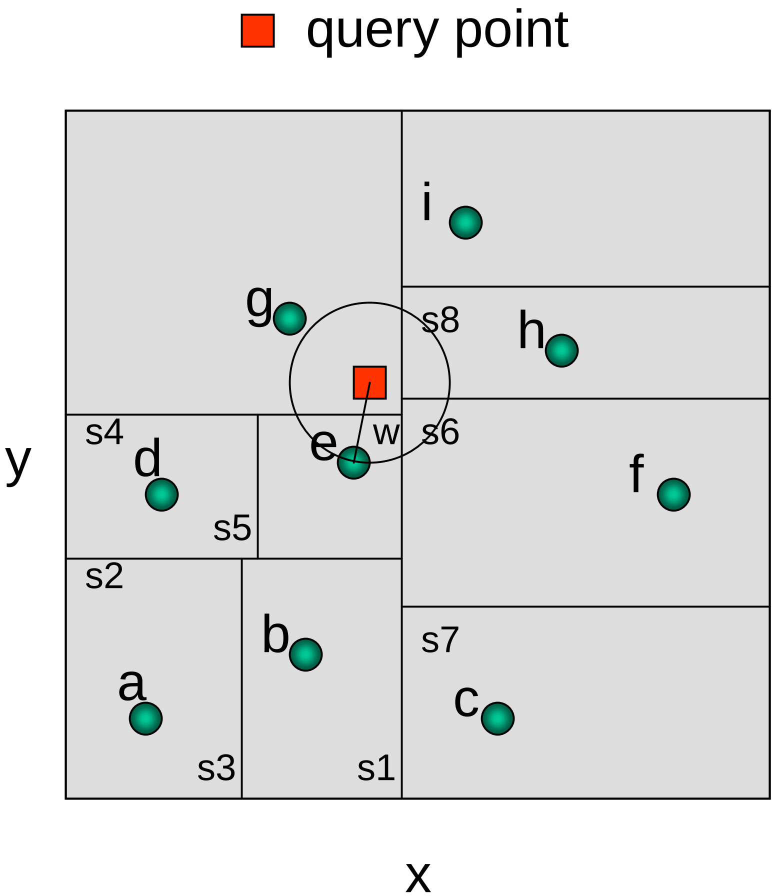


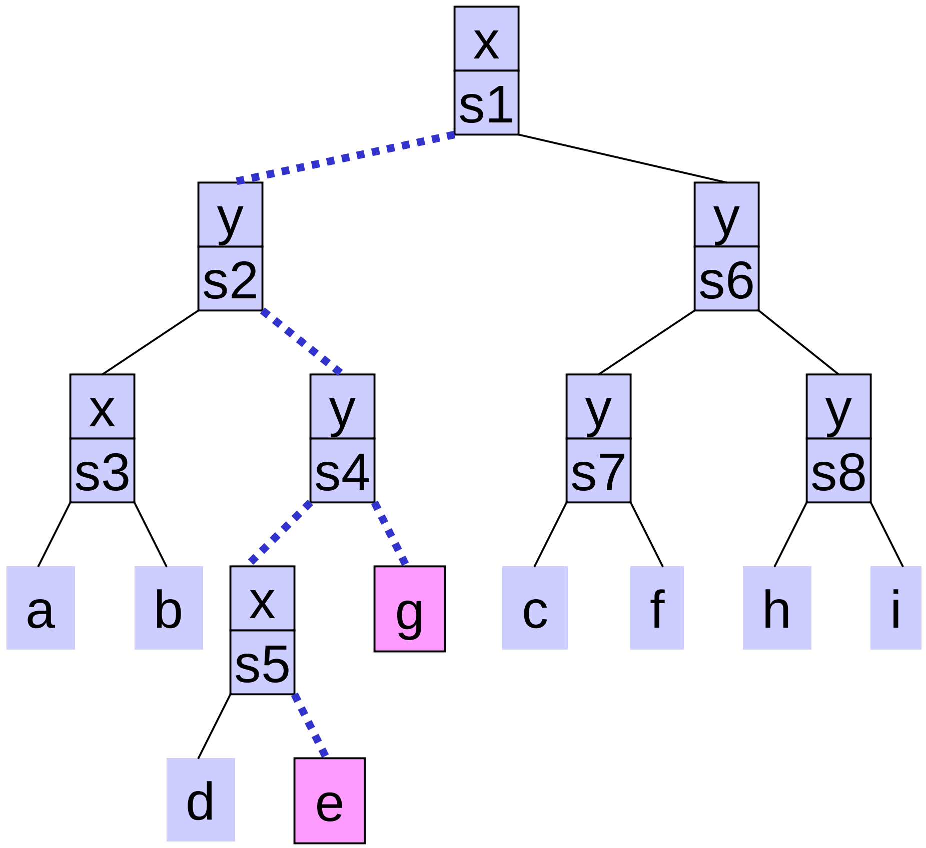
query point
i
g
s8
h
s4
d
e
w
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

k-d Tree NNS (10)

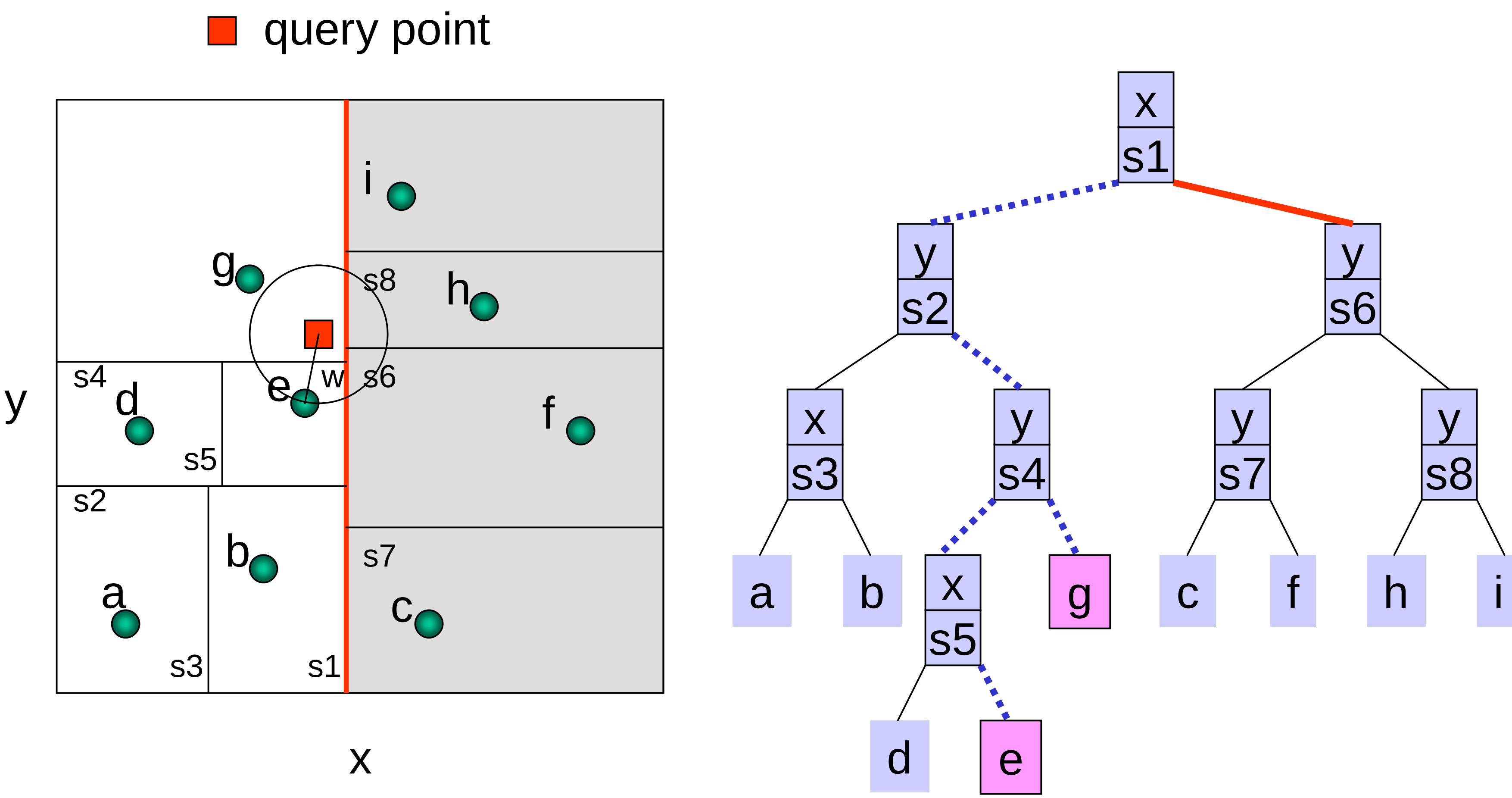


k-d Tree NNS (11)

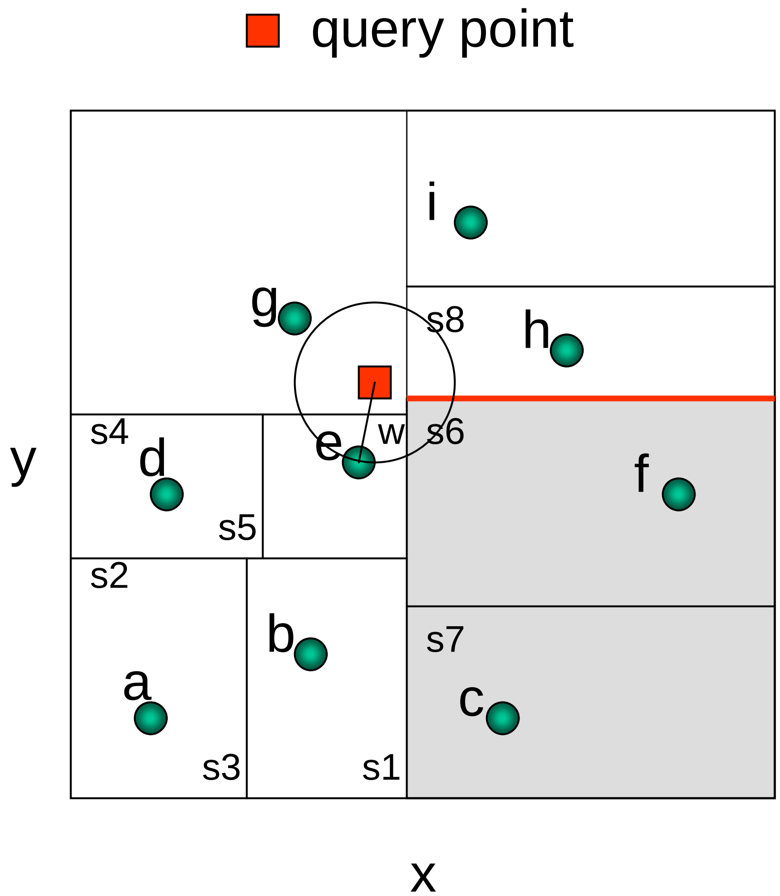


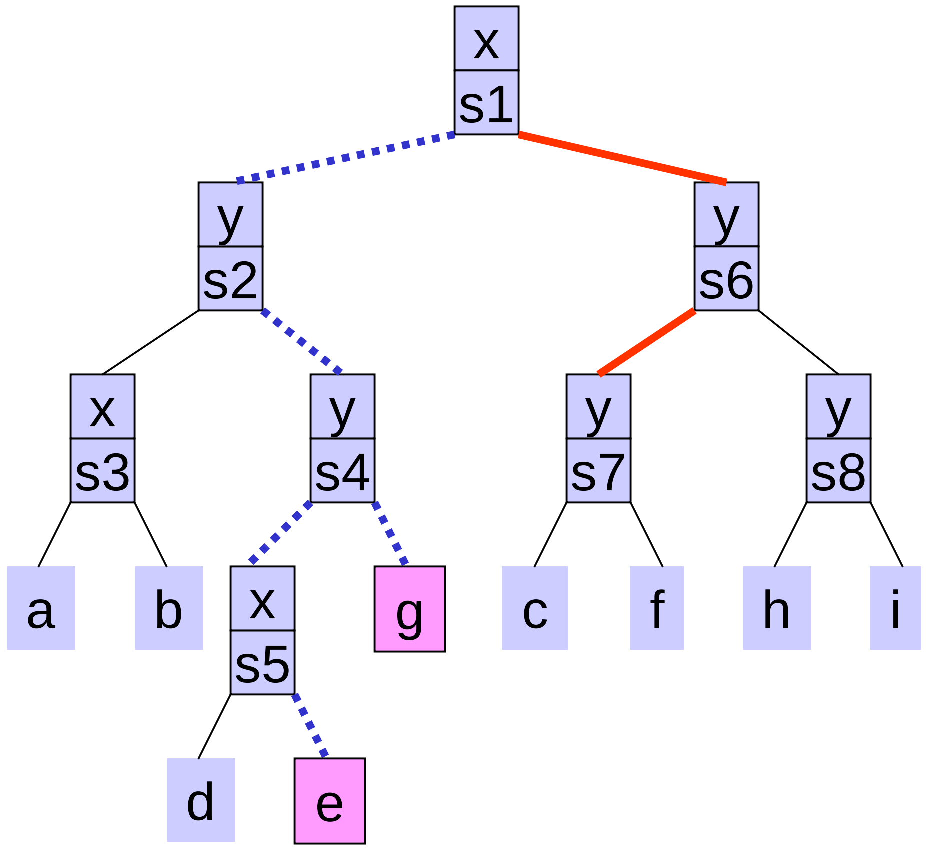


k-d Tree NNS (12)

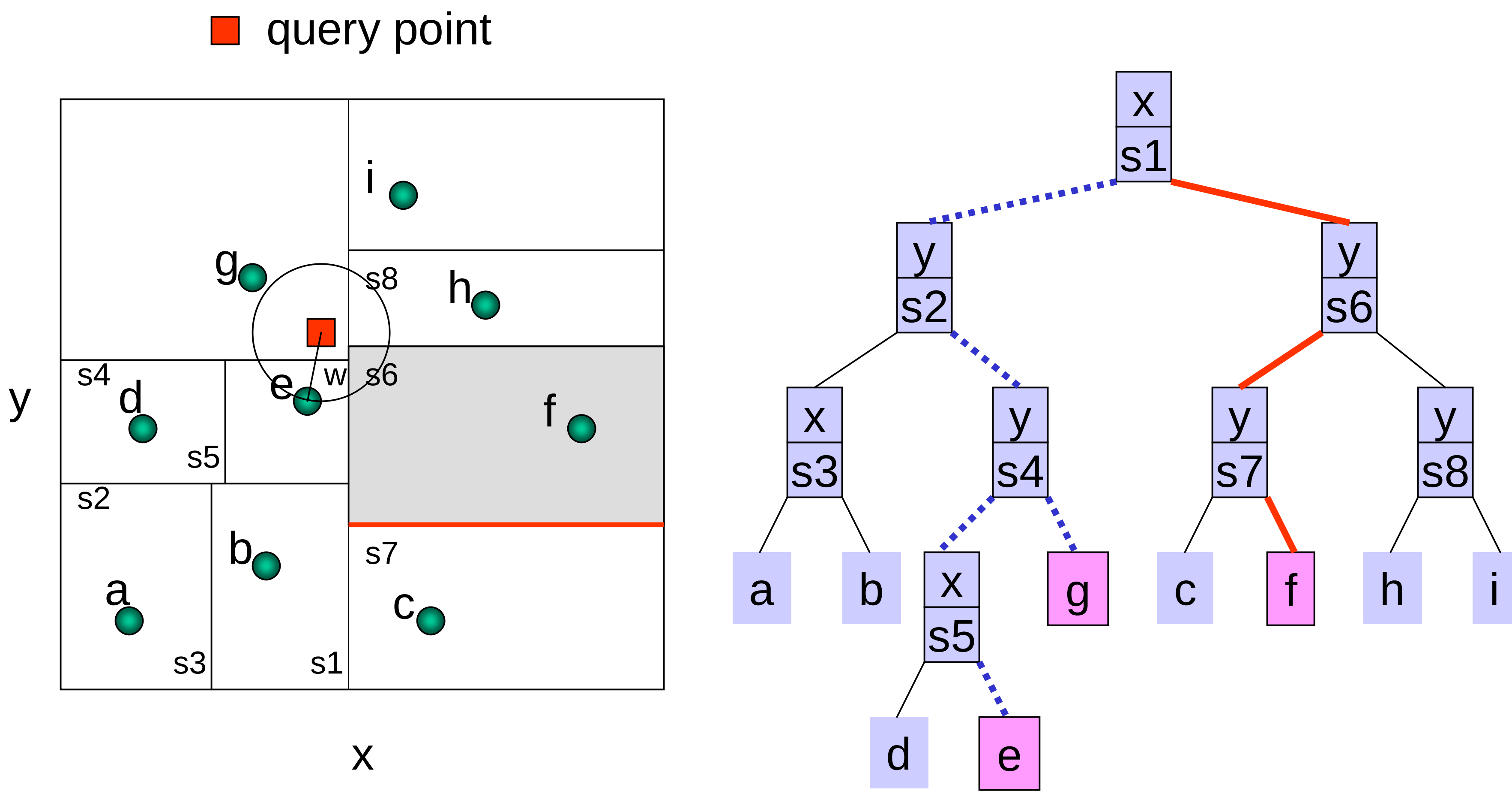


k-d Tree NNS (13)

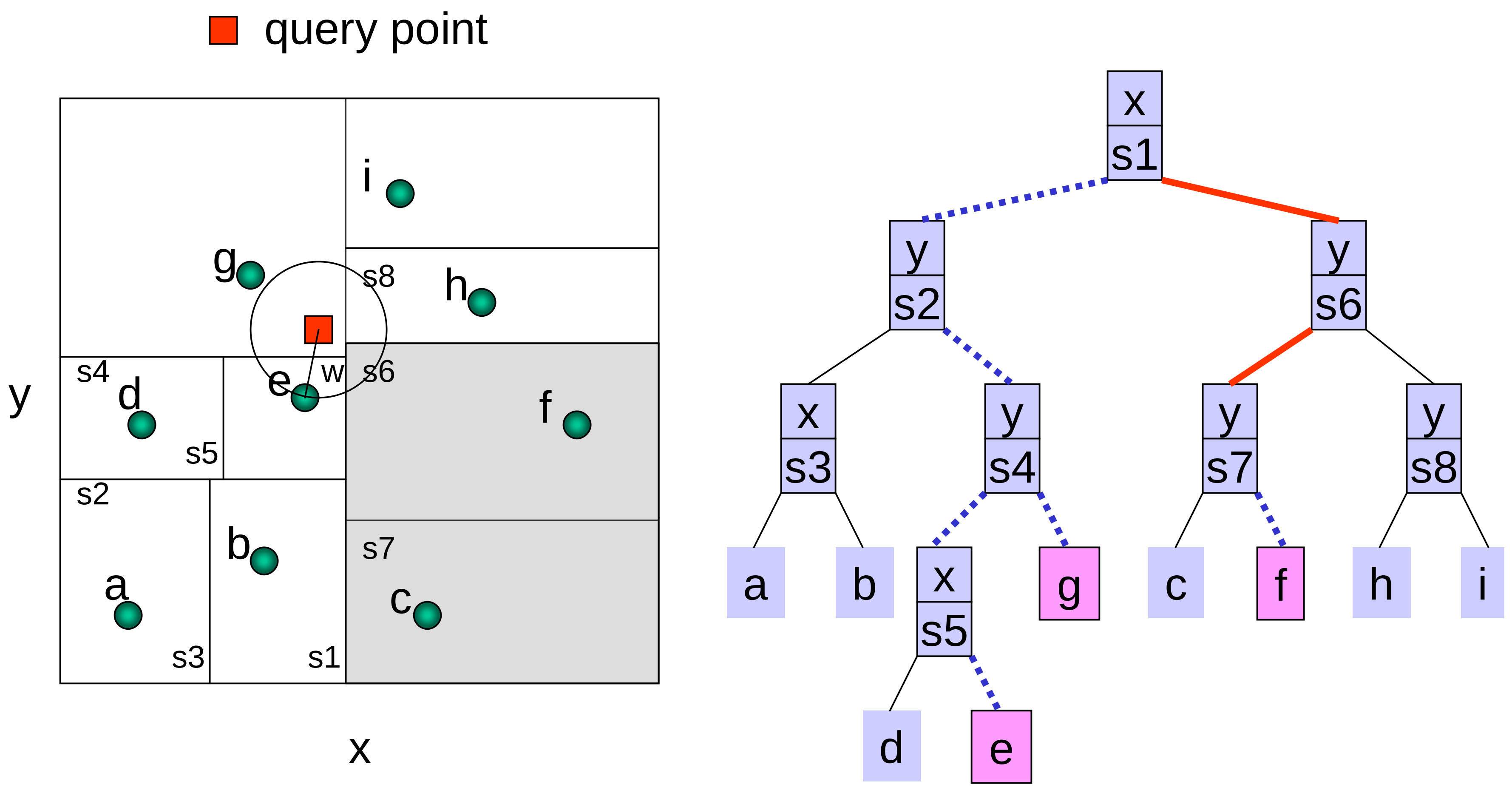




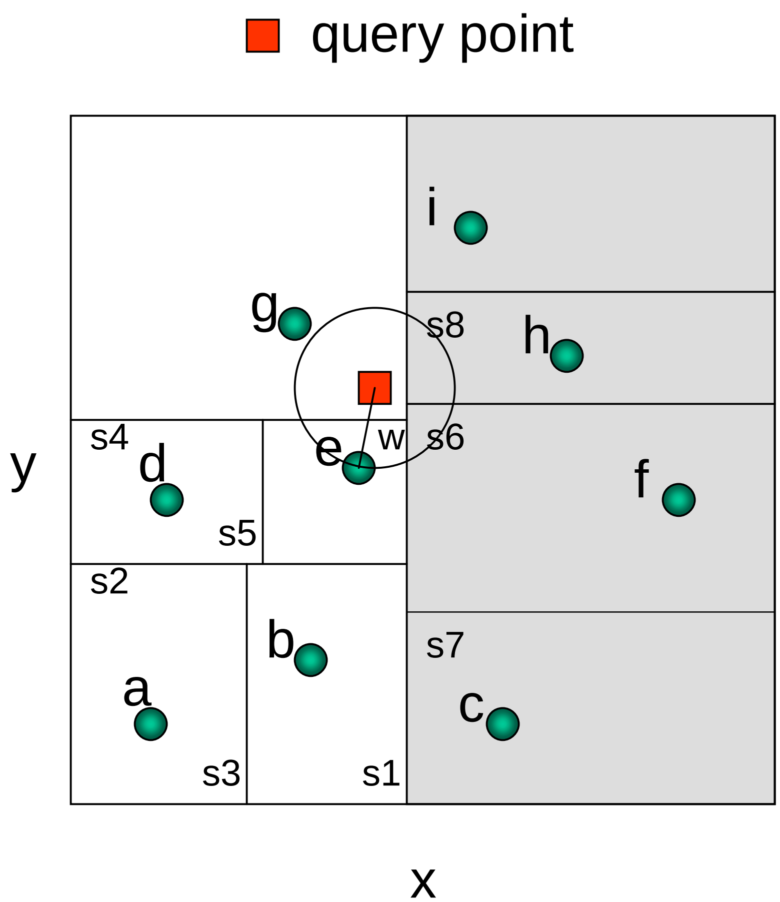
k-d Tree NNS (14)

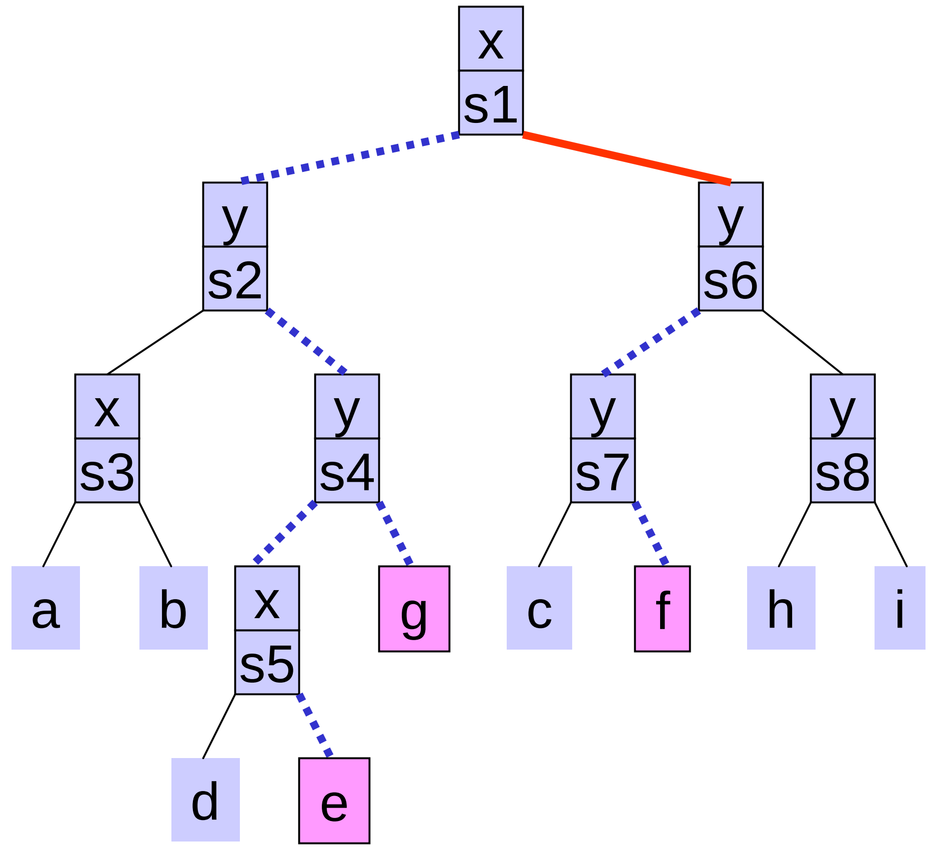


k-d Tree NNS (15)

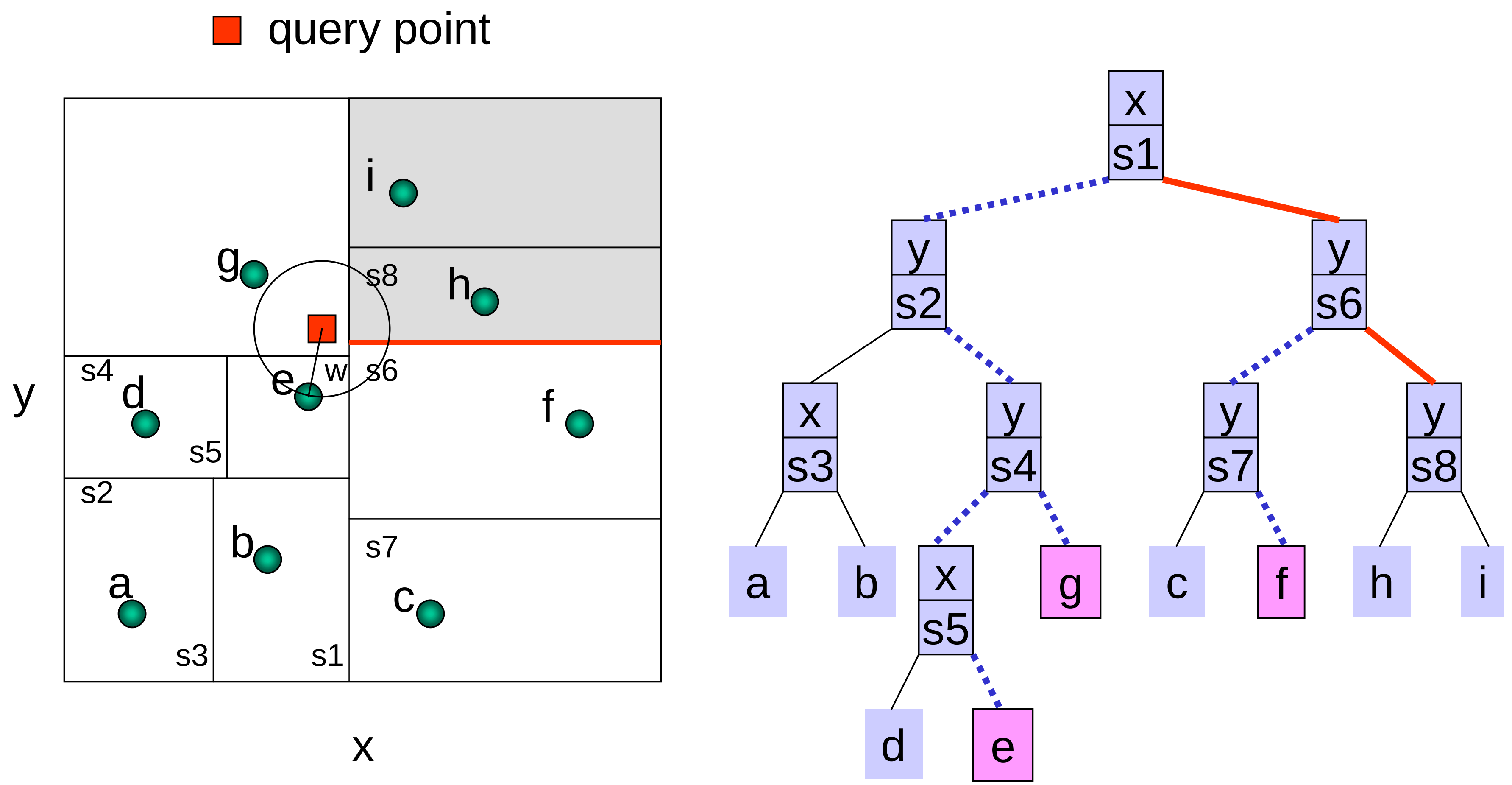


k-d Tree NNS (16)

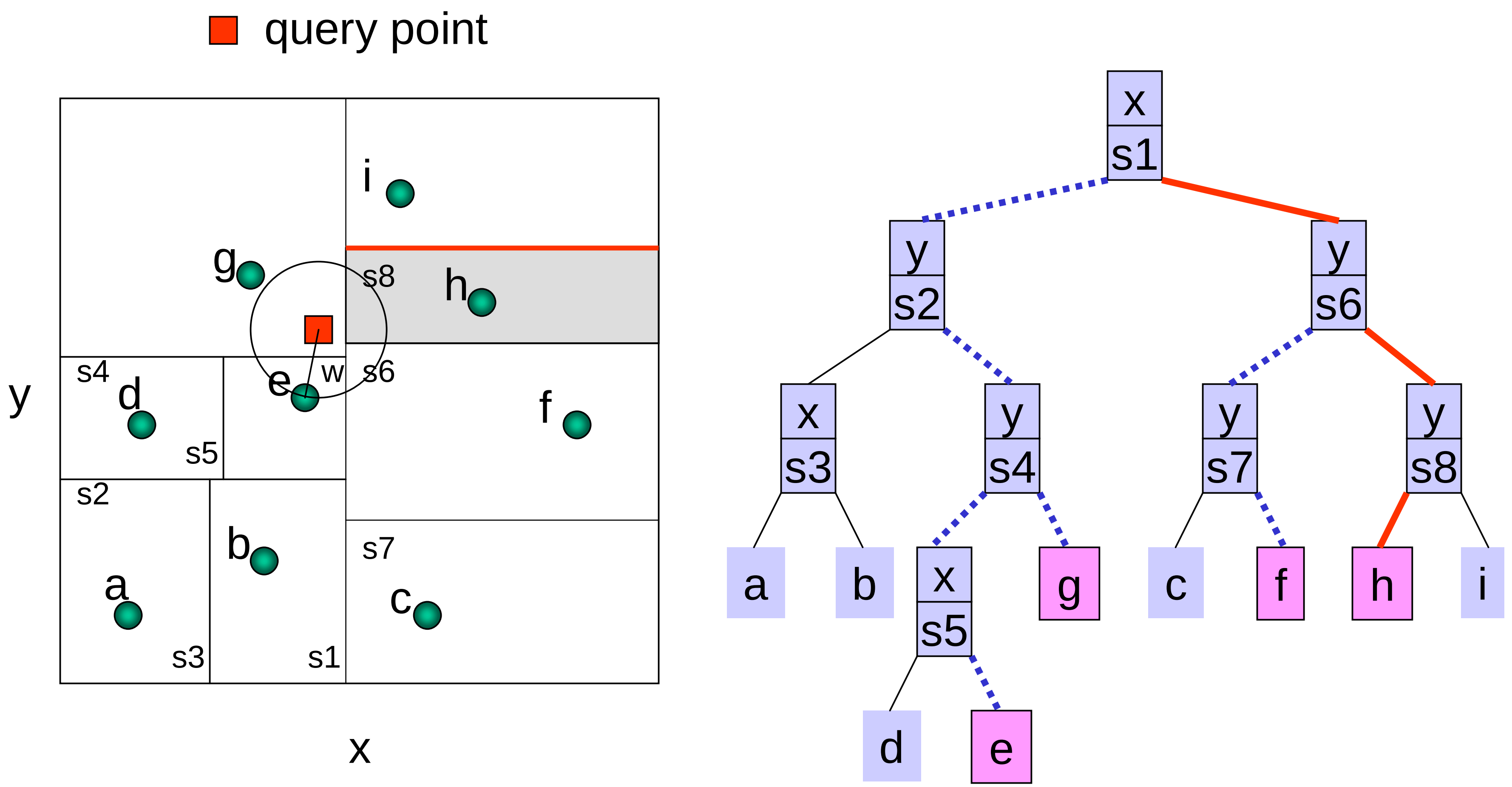




k-d Tree NNS (17)

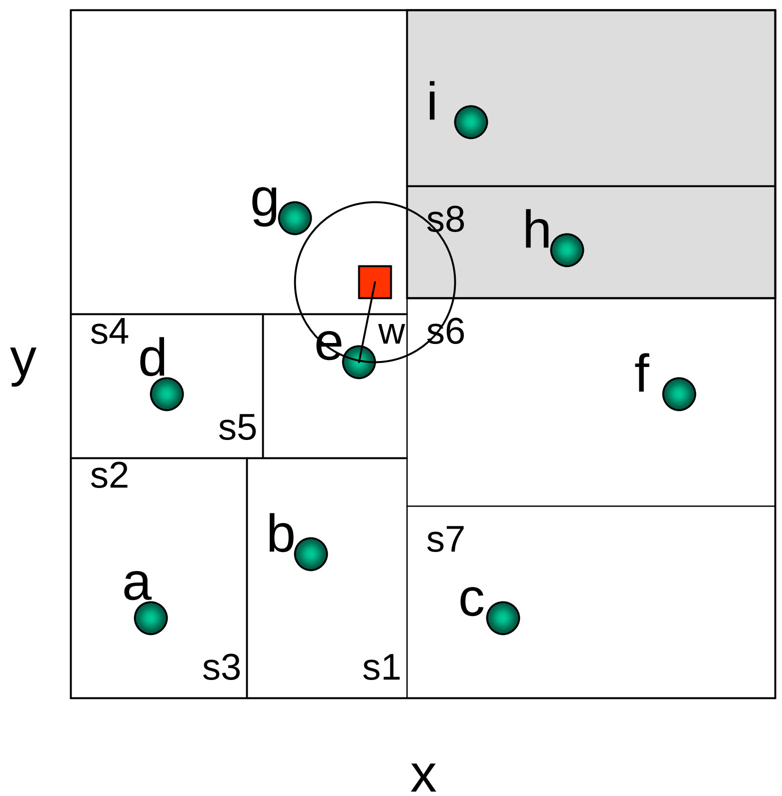


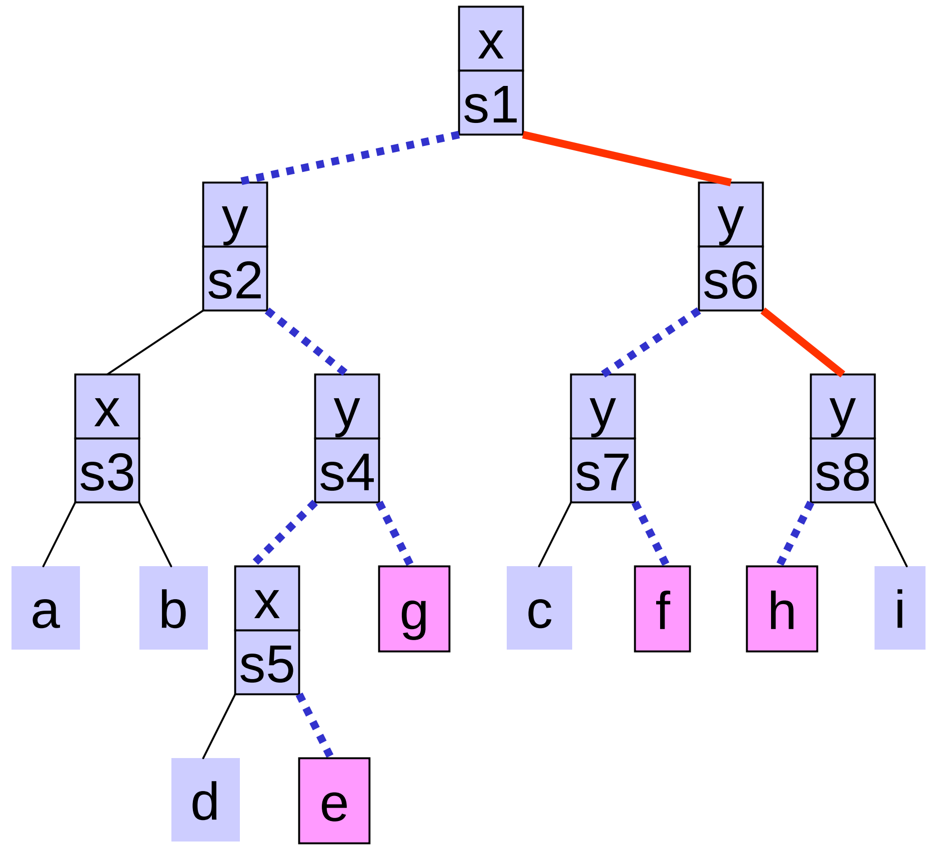
k-d Tree NNS (18)



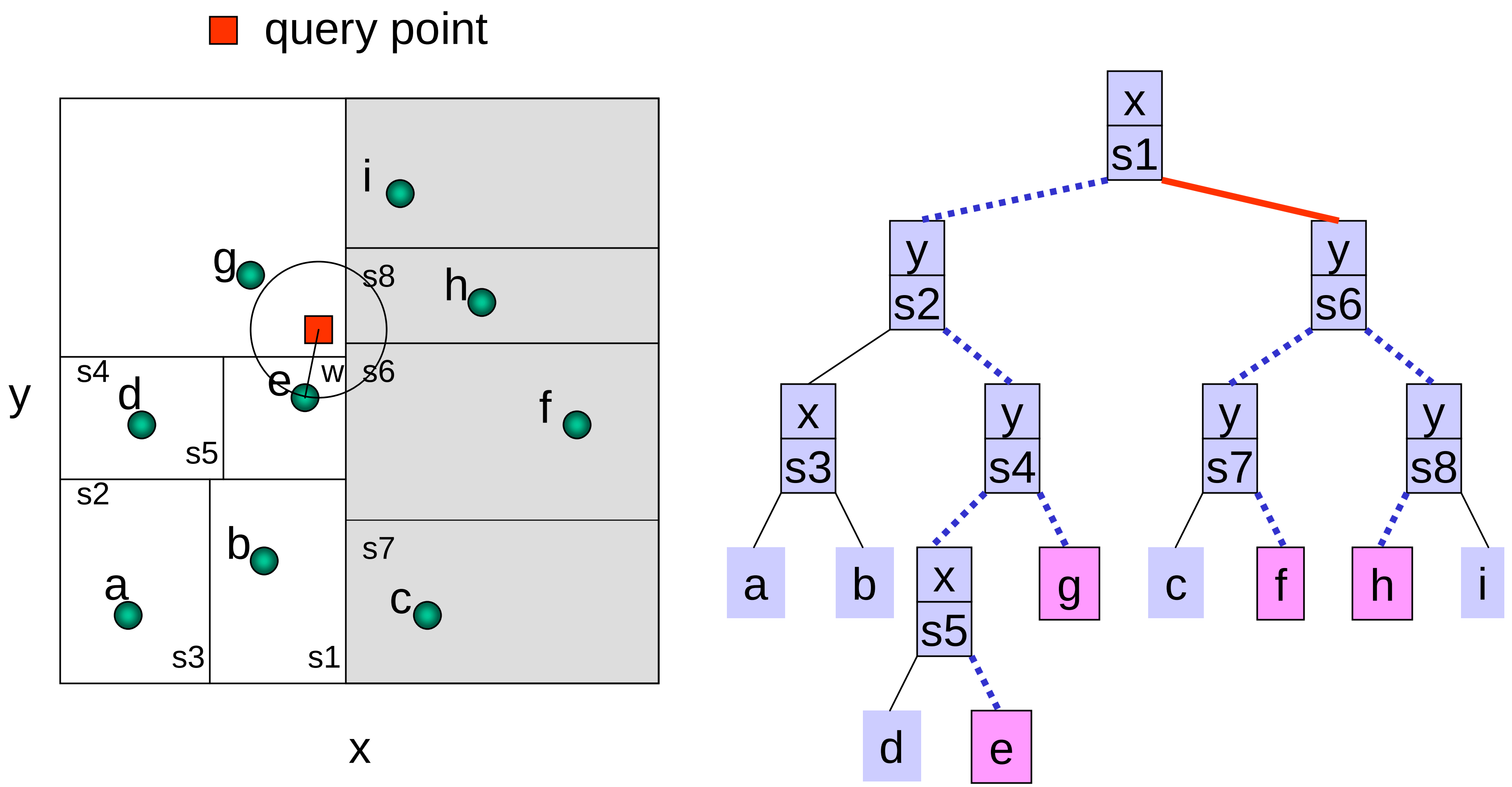
k-d Tree NNS (19)

query point

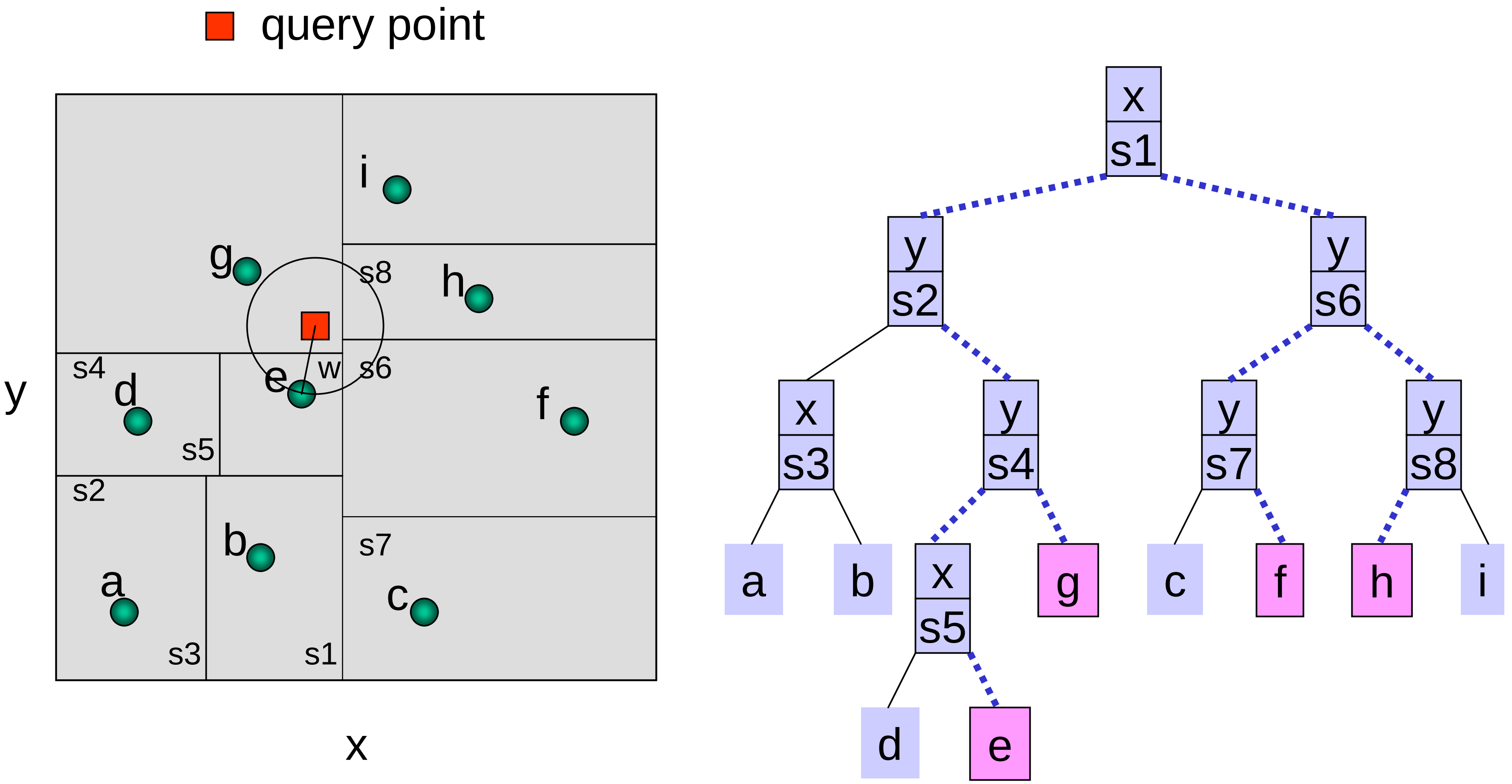




k-d Tree NNS (20)



k-d Tree NNS (21)



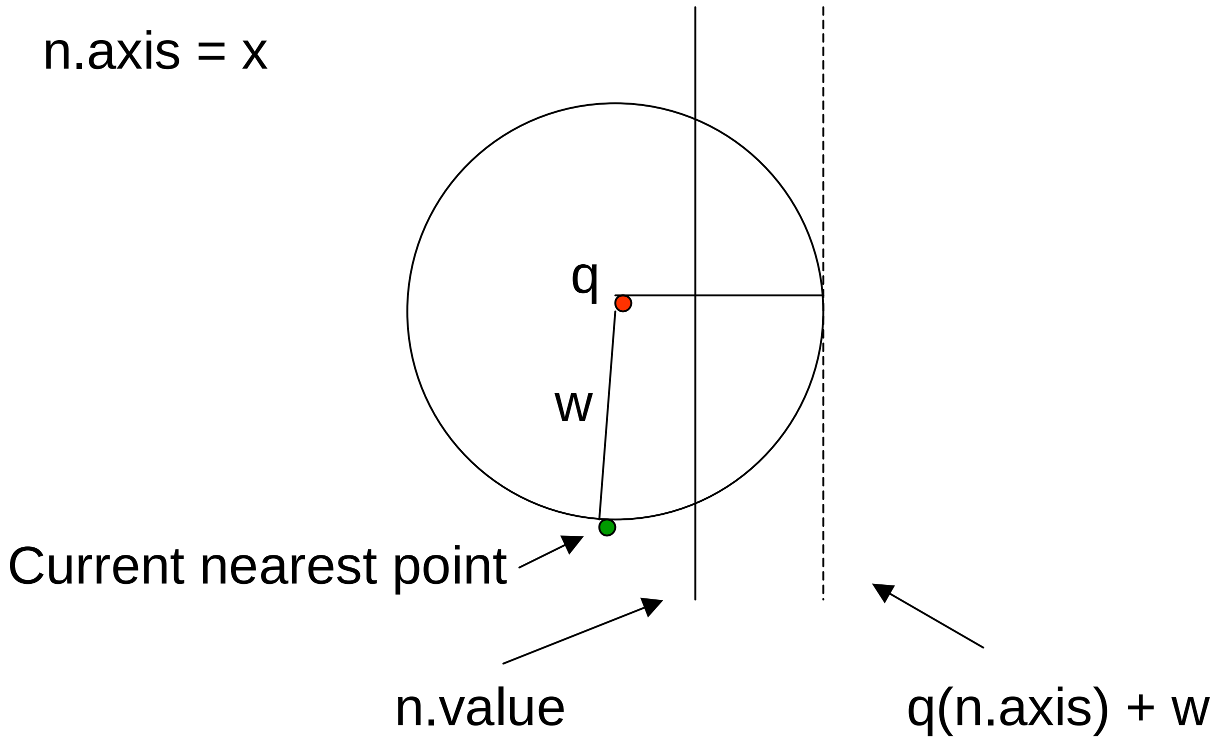
Main is NNS(q,root,null,infinity)

Nearest Neighbor Search

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

The Conditional

```
q(n.axis) + w > n.value
```



Notes on k-d NNS

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

Geometric Data Structures

- Geometric data structures are common.
- The k-d tree is one of the simplest.
  - Nearest neighbor search
  - Range queries
- Other data structures used for
  - 3-d graphics models
  - Physical simulations