BFST - First Year Project, Spring 2015

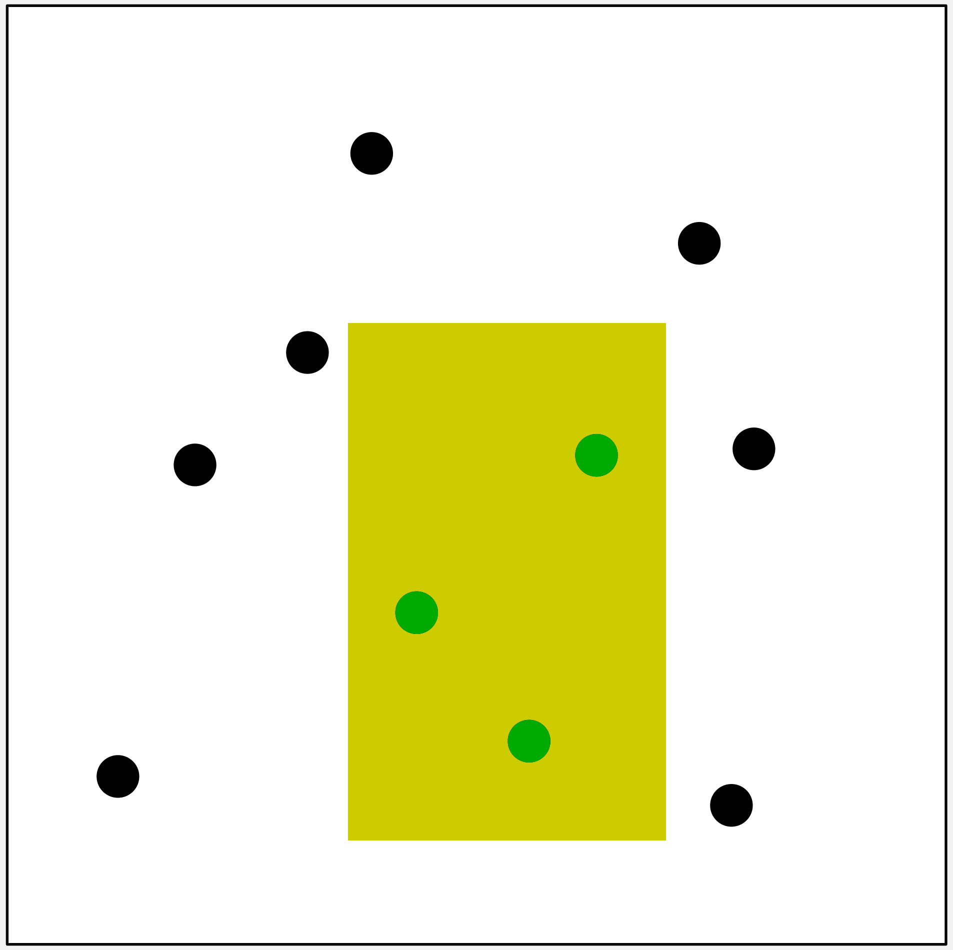
Danmarkskort: Visualisering, Navigation, Søgning og Ruteplanlægning

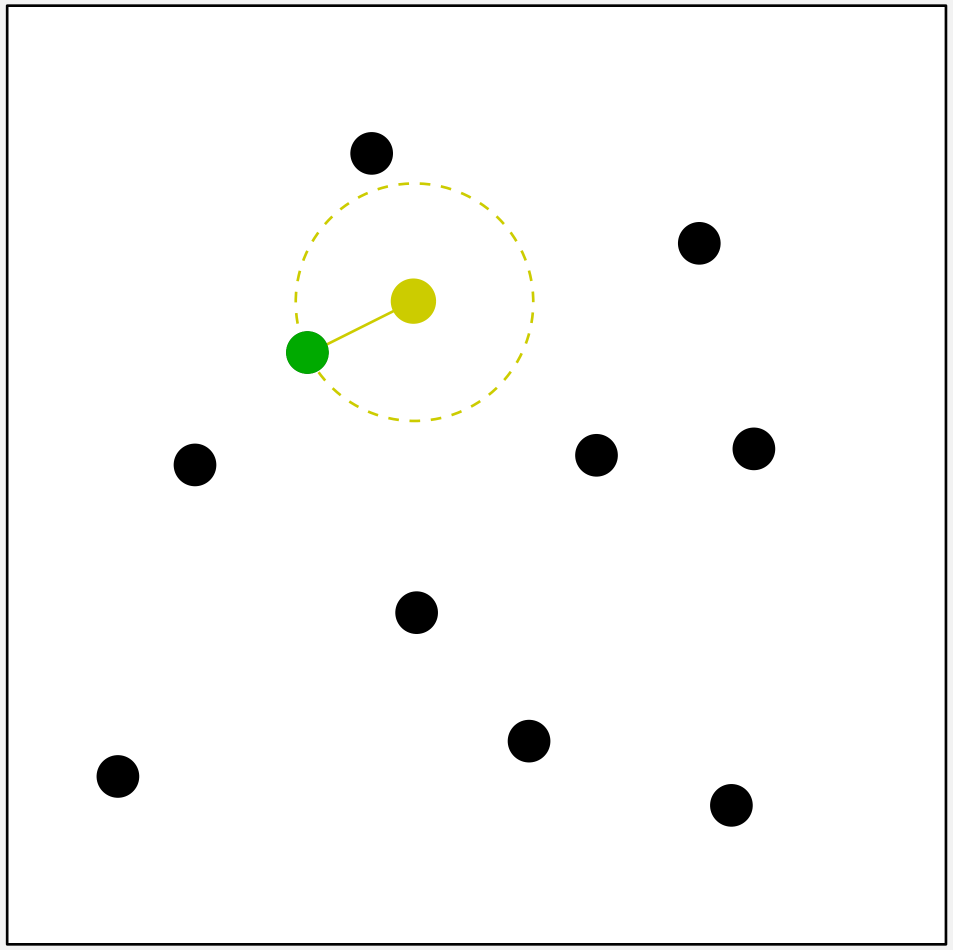
Lecture 10: Spatial Data Structures

Troels Bjerre Sørensen

Overview

- 2D range queries
- Nearest neighbor search





Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

Geometric Applications of

- *1d range search*
- *line segment intersection*
- *kd trees*
- *interval search trees*
- *rectangle intersection*

1d range search

Extension of ordered symbol table.

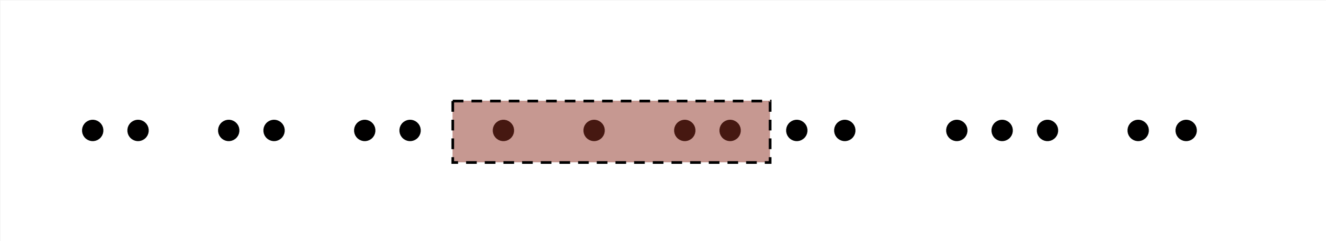
- Insert key-value pair.
- Search for key k.
- Delete key k.
- Range search: find all keys between k_1 and k_2.
- Range count: number of keys between k_1 and k_2.

Application. Database queries.

insert B	B
insert D	B D
insert A	A B D
insert I	A B D I
insert H	A B D H I
insert F	A B D F H I
insert P	A B D F H I P
search G to K	H I
count G to K	2

Geometric interpretation.

- Keys are point on a line.
- Find/count points in a given 1d interval.



1d range search: elementary implementations

Unordered list. Fast insert, slow range search.

Ordered array. Slow insert, binary search for k_1 and k_2 to do range search.

order of growth of running time for 1d range search

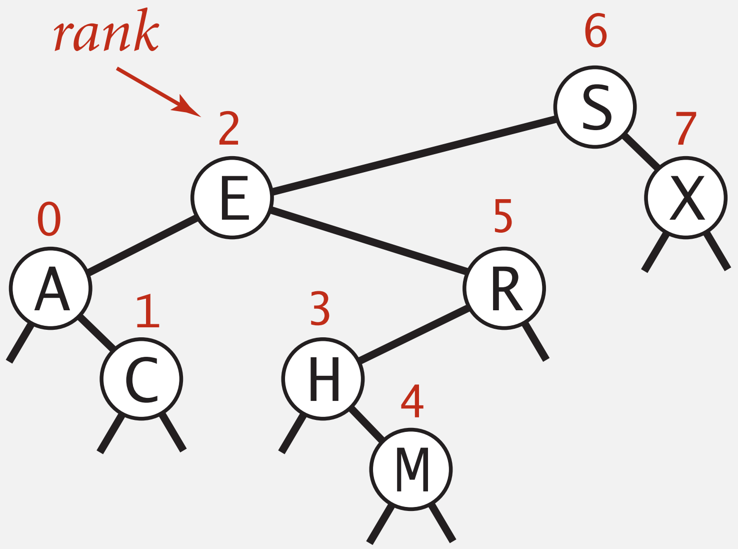
data structure	insert	range count	range search
unordered list	1	N	N
ordered array	N	$\log N$	$R + \log N$
goal	$\log N$	$\log N$	$R + \log N$

N = number of keys

R = number of keys that match

1d range count: BST implementation

1d range count. How many keys between `lo` and `hi` ?



```
public int size(Key lo, Key hi)
{
   if (contains(hi)) return rank(hi) - rank(lo) + 1;
   else              return rank(hi) - rank(lo);
}
```

number of keys < hi

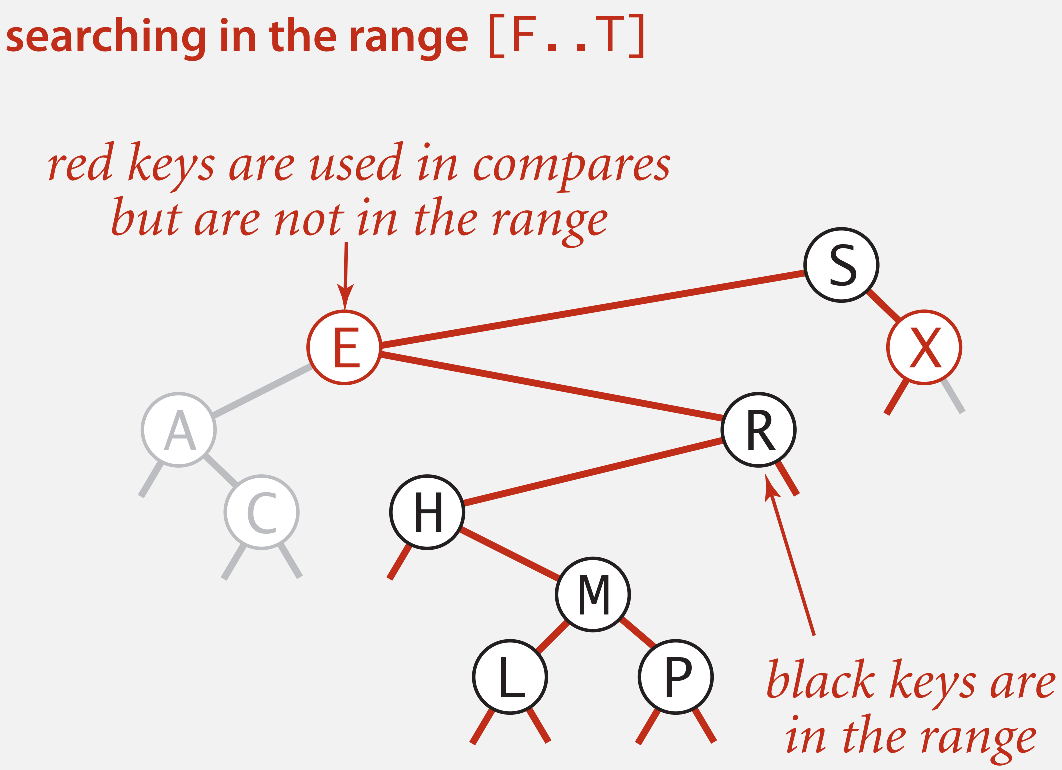
Proposition. Running time proportional to $\log N$.

Pf. Nodes examined = search path to `lo` + search path to `hi`.

1d range search: BST implementation

1d range search. Find all keys between `lo` and `hi`.

- Recursively find all keys in left subtree (if any could fall in range).
- Check key in current node.
- Recursively find all keys in right subtree (if any could fall in range).


searching in the range [F..T]
red keys are used in compares but are not in the range
S
E
X
A
R
C
H
M
L
P
black keys are in the range

Proposition. Running time proportional to $R + \log N$.

Pf. Nodes examined = search path to `lo` + search path to `hi` + matches.

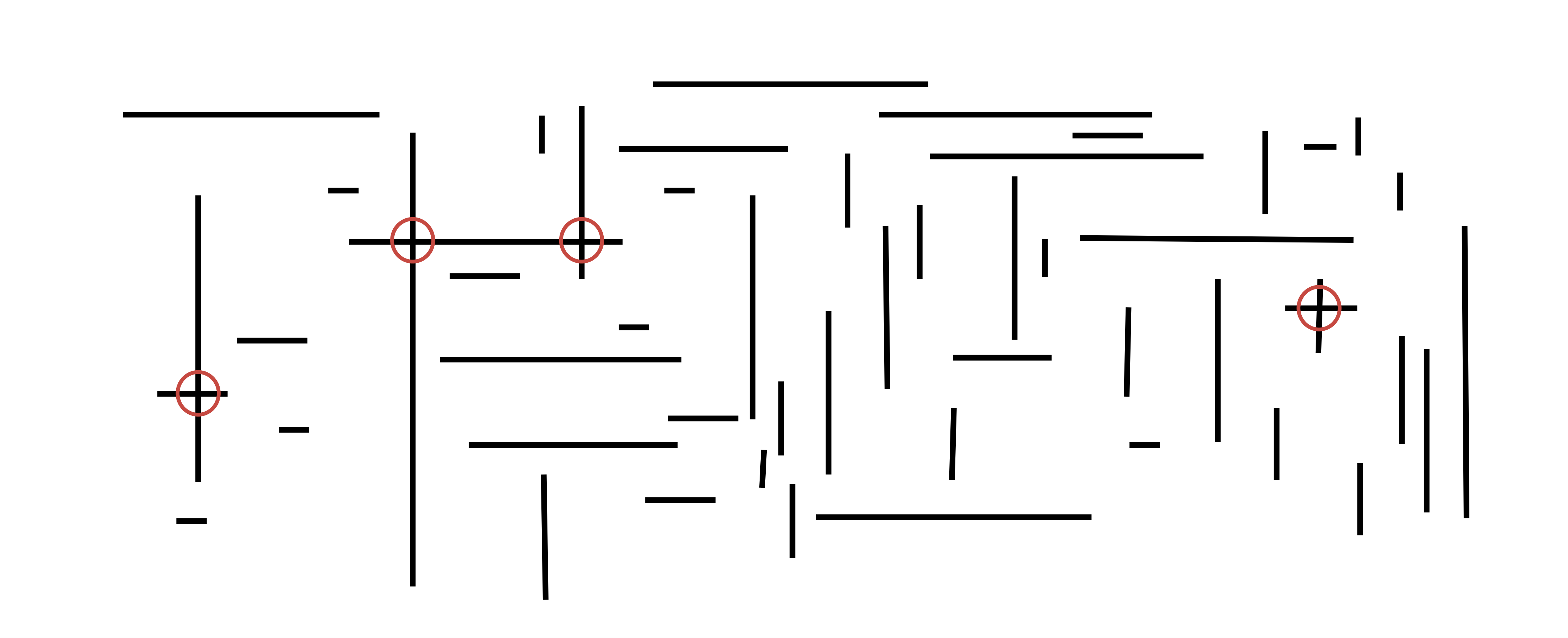
http://algs4.cs.princeton.edu

Geometric Applications of

- *1d range search*
- *line segment intersection*
- *kd trees*
- *interval search trees*
- *rectangle intersection*

Orthogonal line segment intersection

Given N horizontal and vertical line segments, find all intersections.



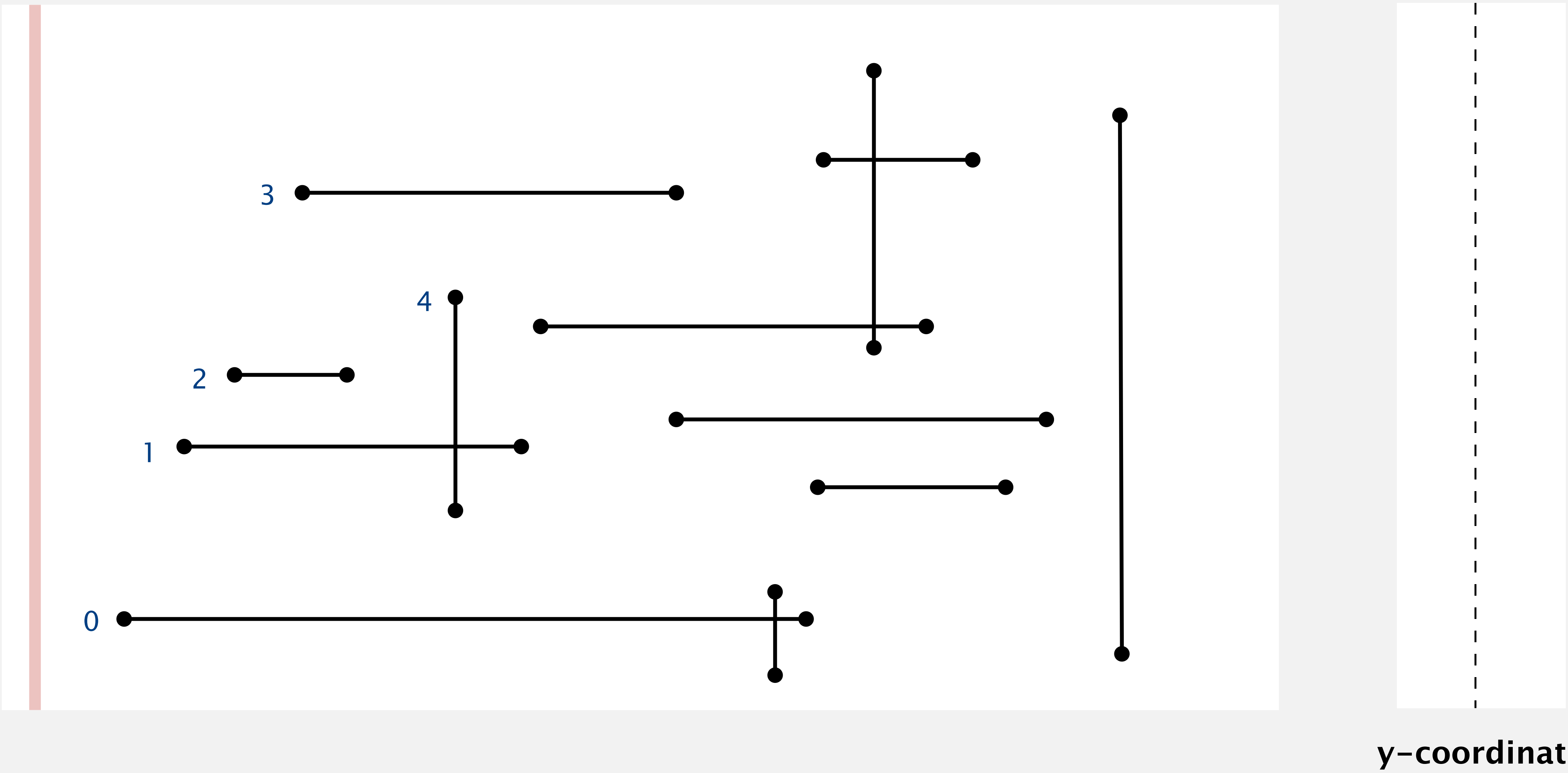
Quadratic algorithm. Check all pairs of line segments for intersection.

Nondegeneracy assumption. All x- and y-coordinates are distinct.

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

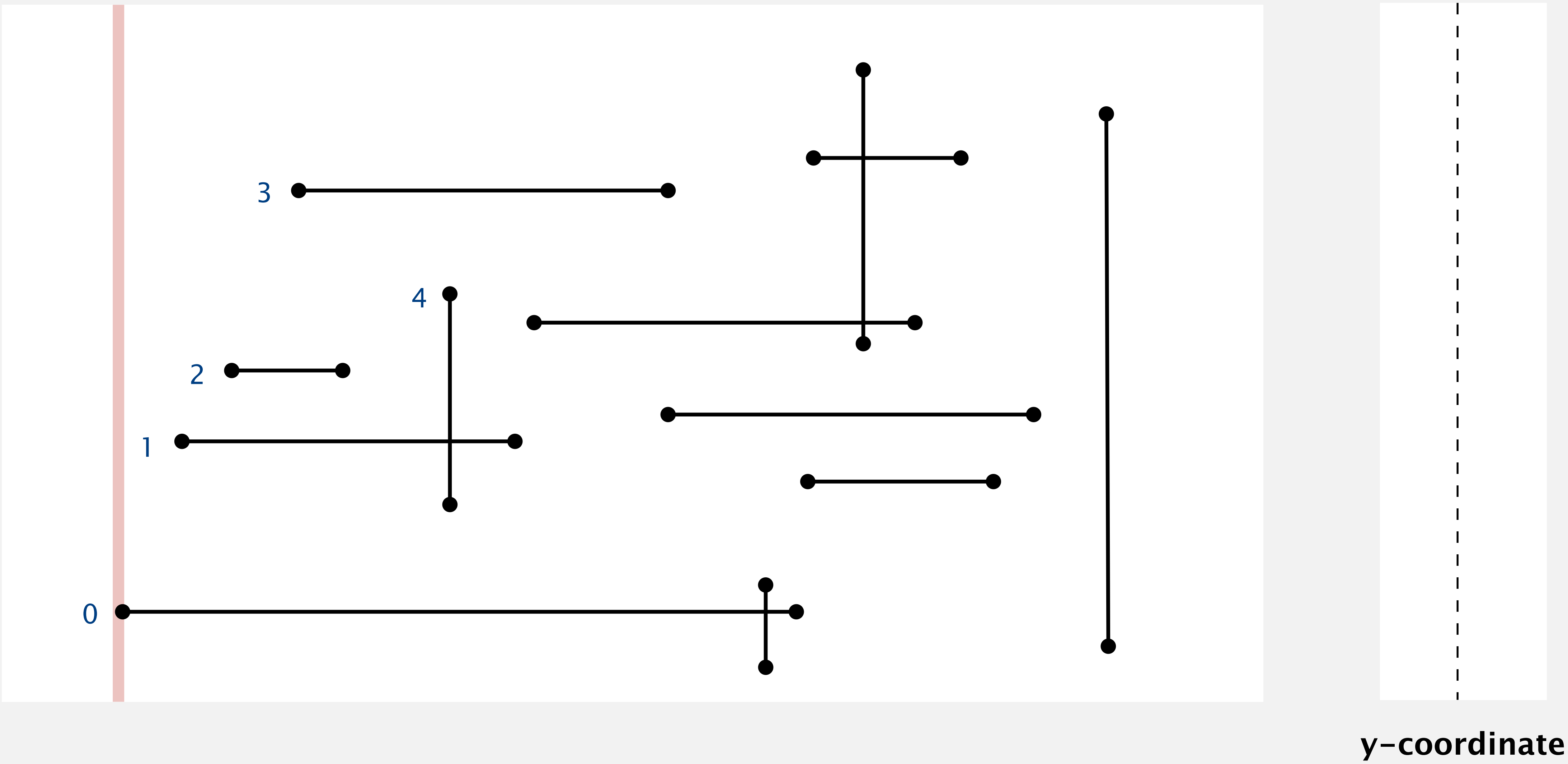
- *x*-coordinates define events.
- *h*-segment (left endpoint): insert *y*-coordinate into BST.



Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

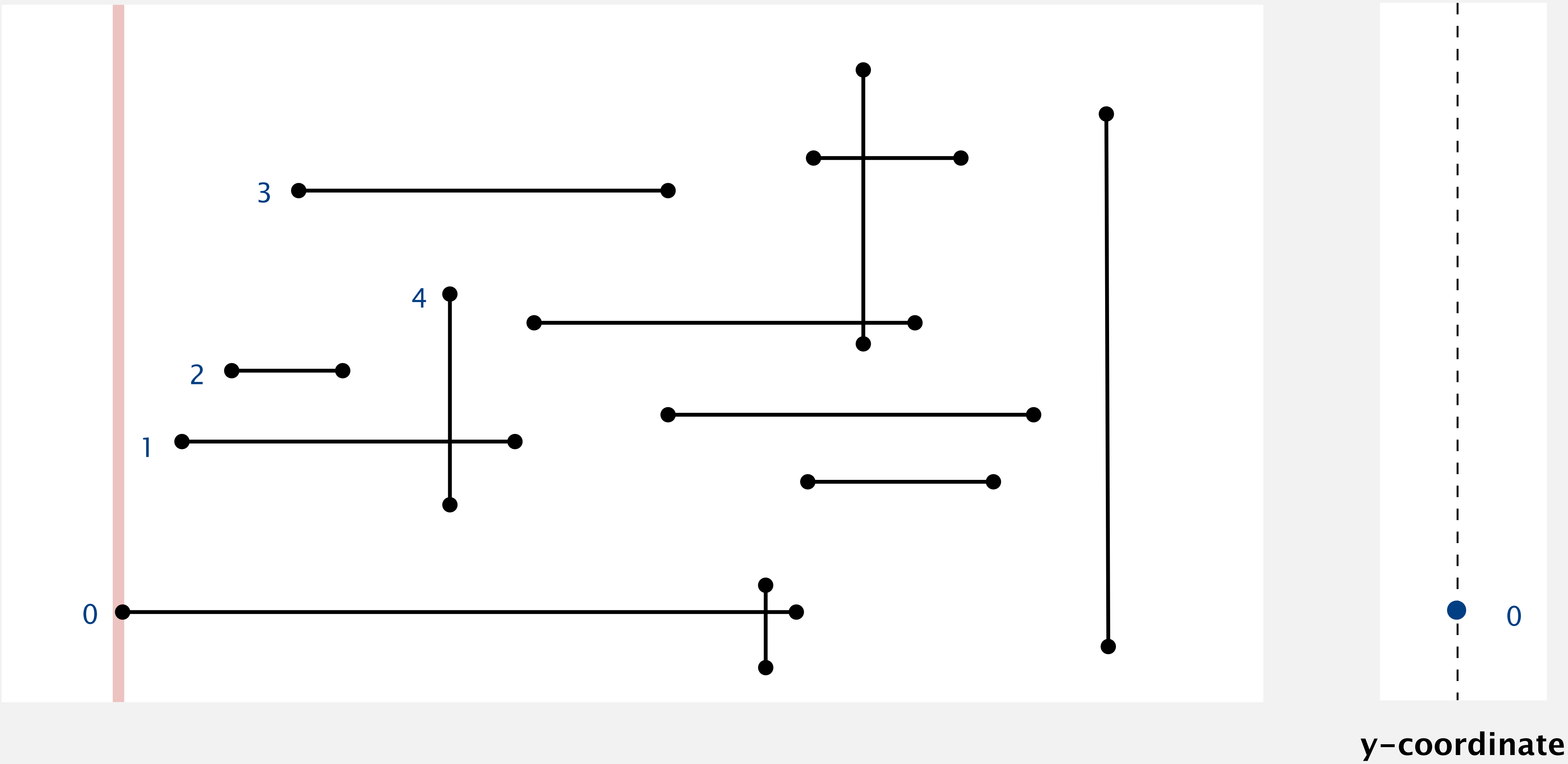
- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.



Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

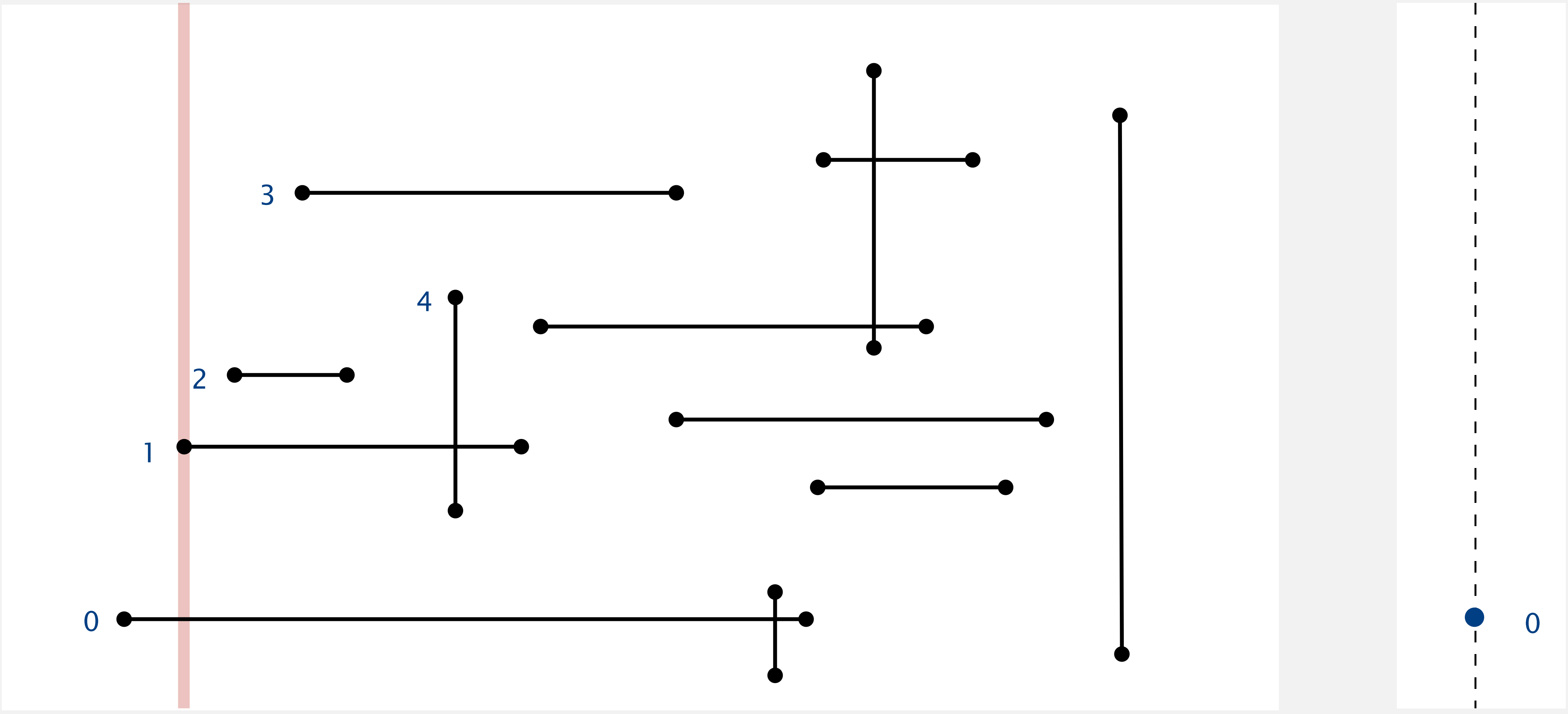
- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.



Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.

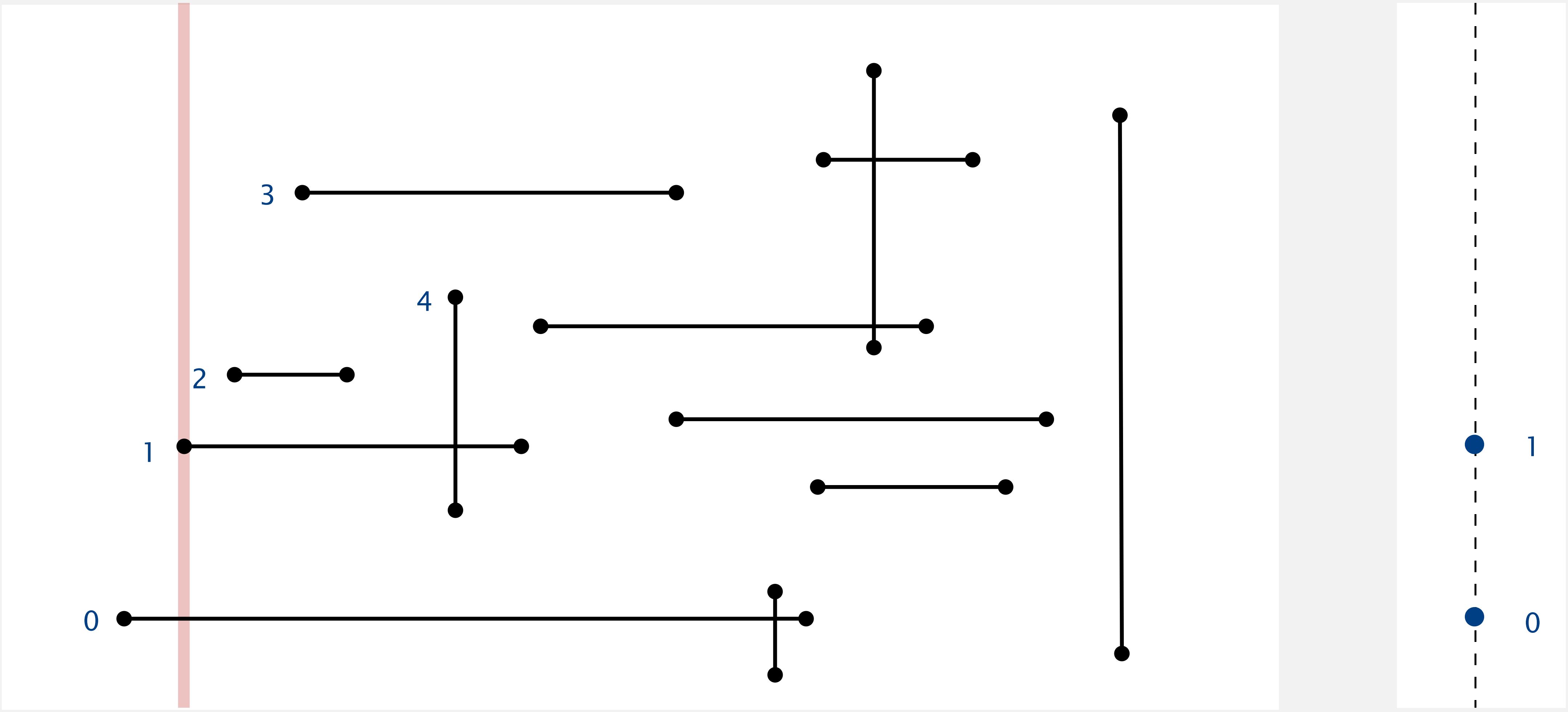


y−coordinates

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.

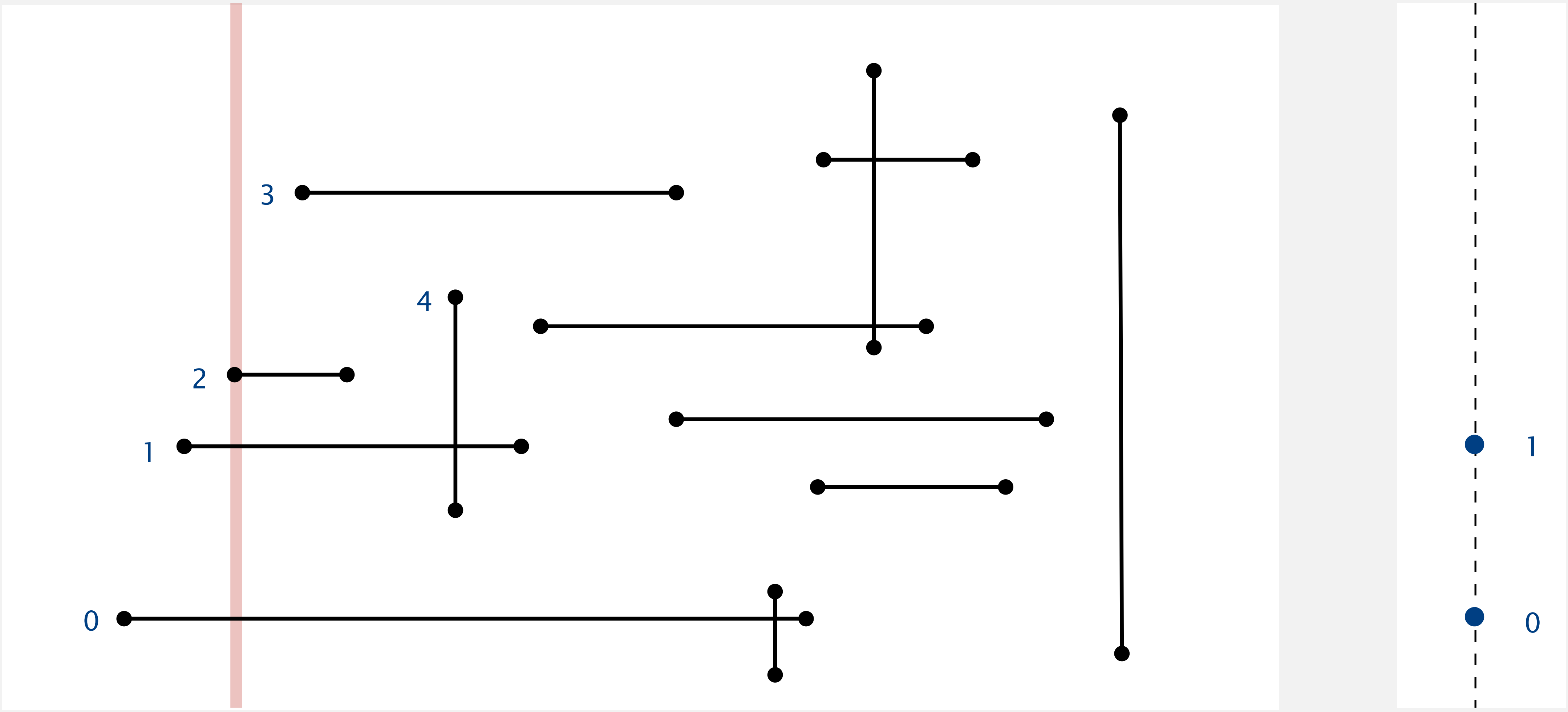


y-coordinates

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.

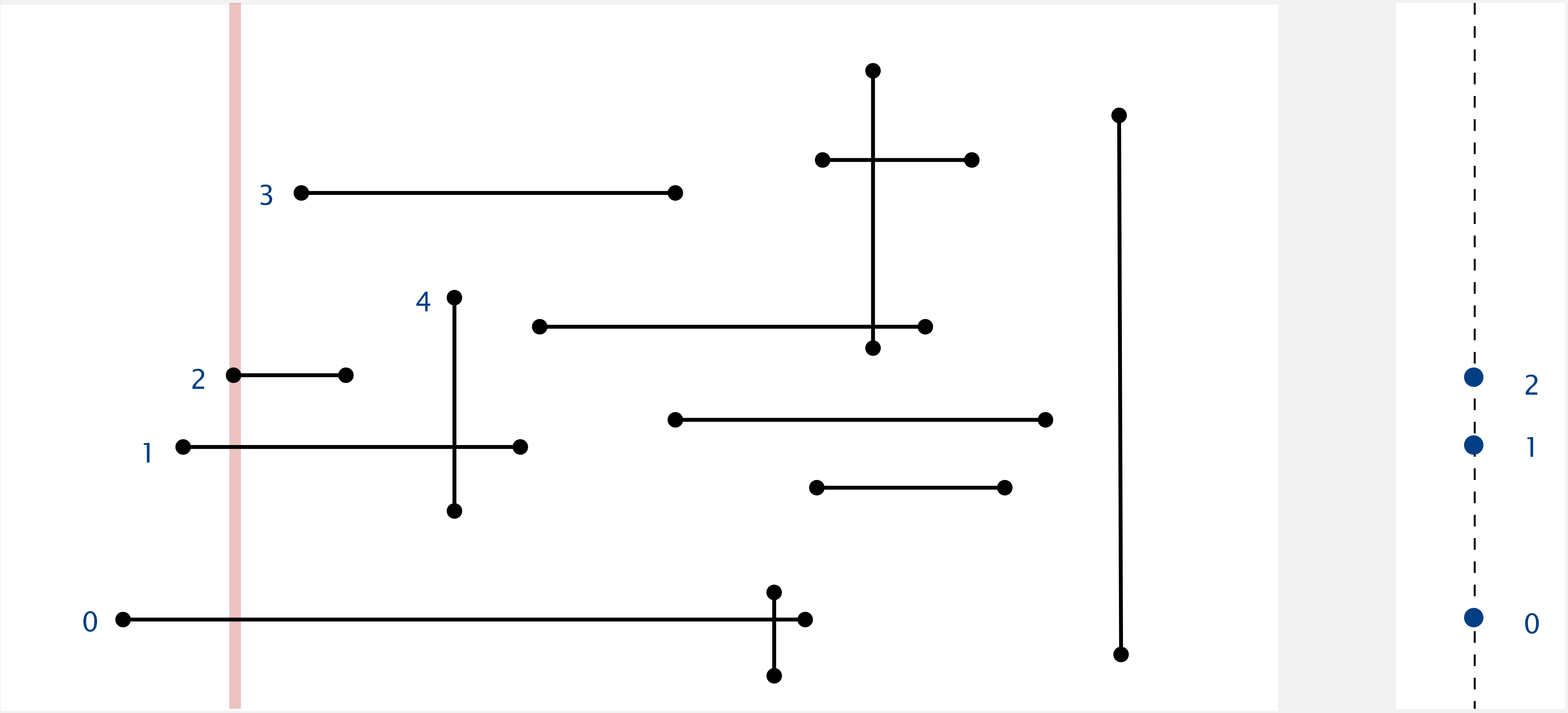


y-coordinates

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.

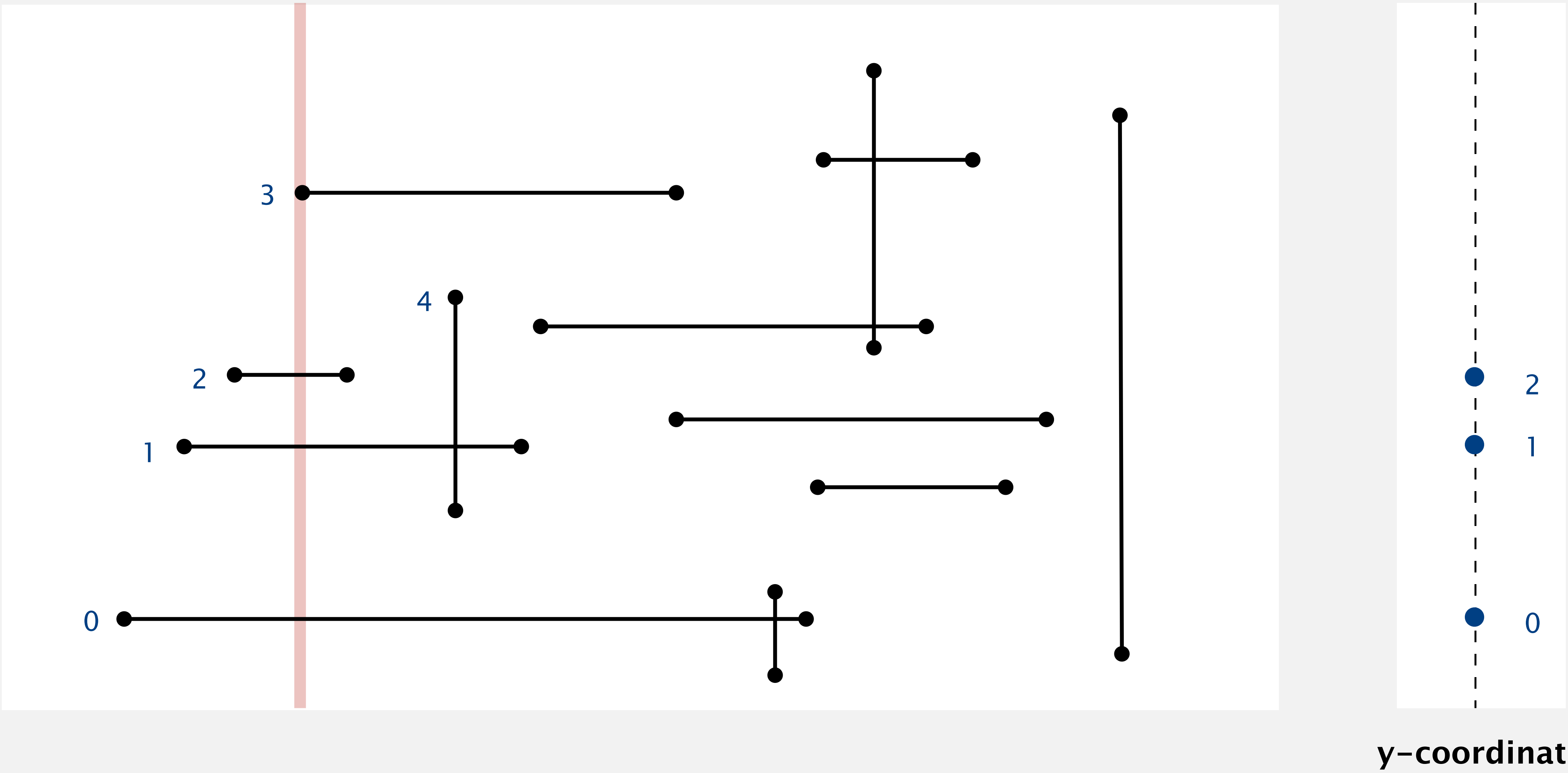


y−coordinates

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

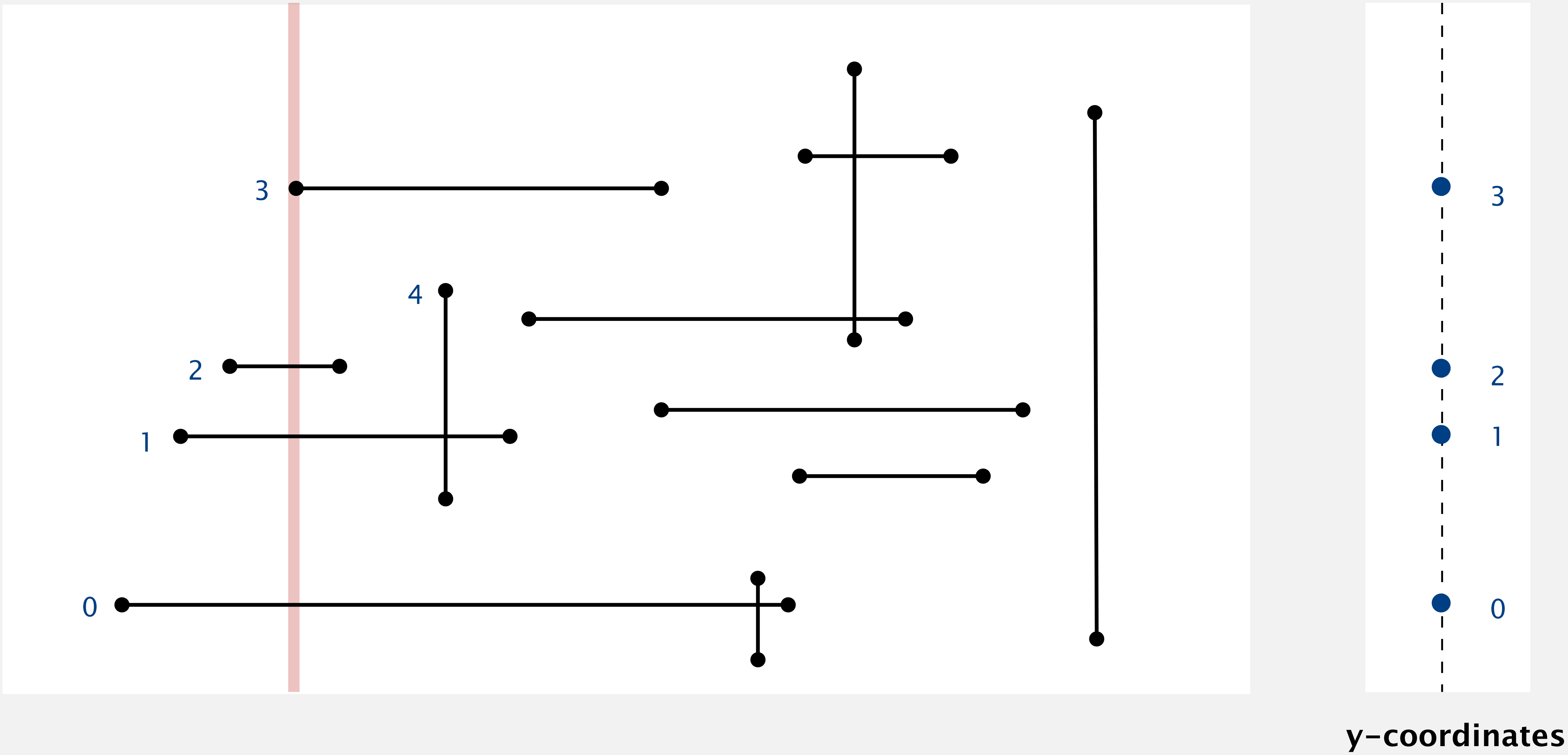
- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.



Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

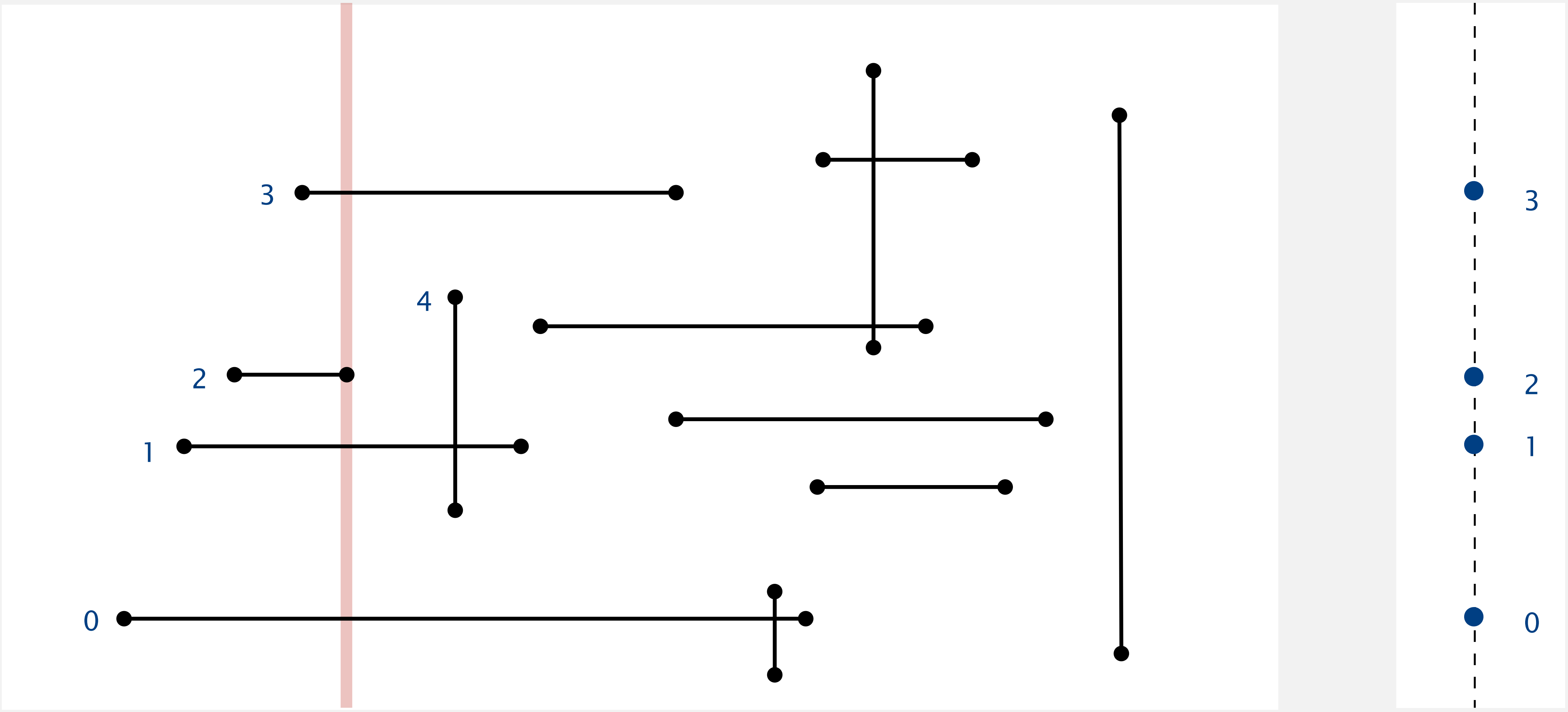
- *x*-coordinates define events.
- *h*-segment (left endpoint): insert *y*-coordinate into BST.



Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- *x*-coordinates define events.
- *h*-segment (left endpoint): insert *y*-coordinate into BST.

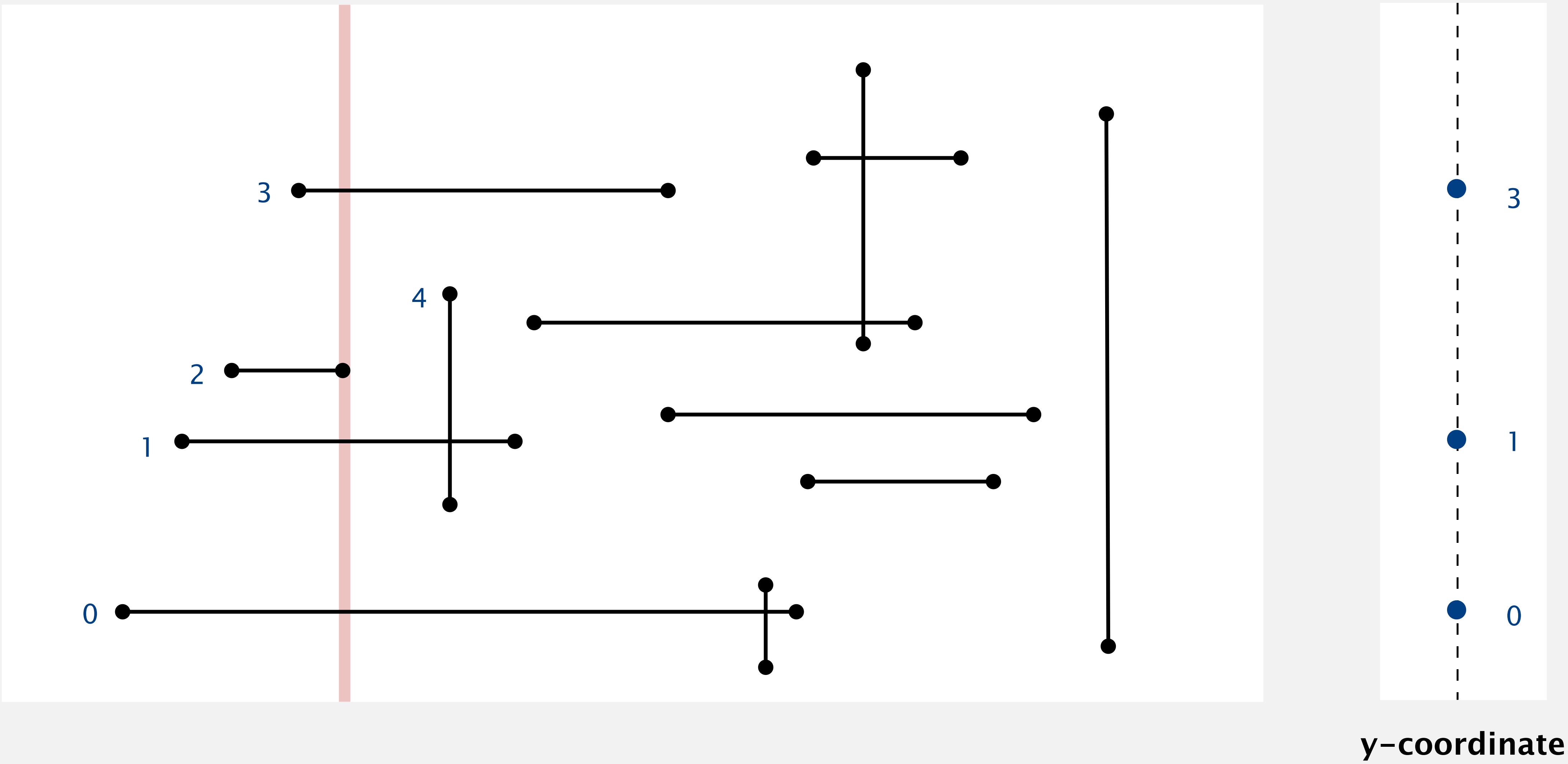


y−coordinates

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

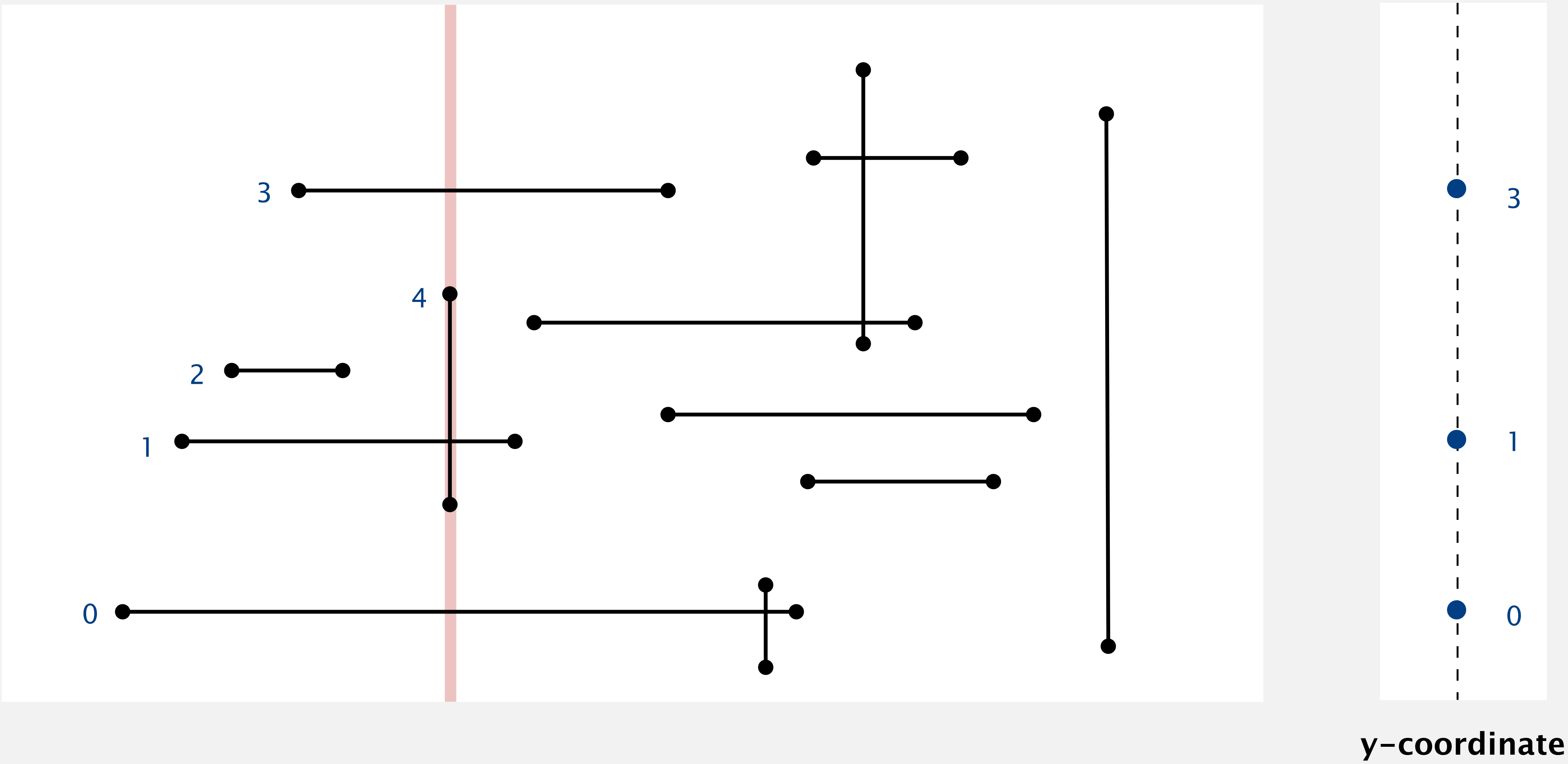
- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.
- h-segment (right endpoint): remove y-coordinate from BST.



Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

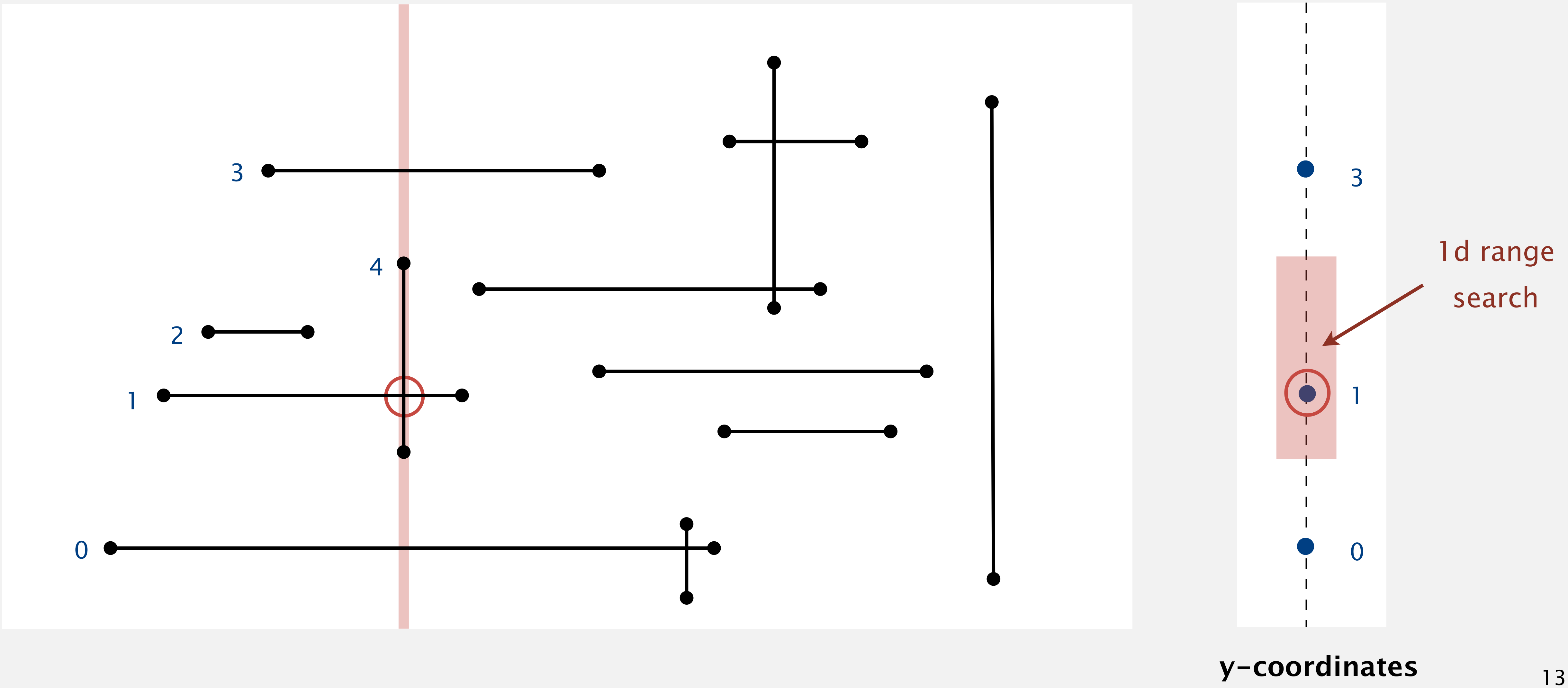
- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.
- h-segment (right endpoint): remove y-coordinate from BST.



Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- *x*-coordinates define events.
- *h*-segment (left endpoint): insert *y*-coordinate into BST.
- *h*-segment (right endpoint): remove *y*-coordinate from BST.
- *v*-segment: range search for interval of *y*-endpoints.



Orthogonal line segment intersection: sweep-line analysis

Proposition. The sweep-line algorithm takes time proportional to $N \log N + R$ to find all R intersections among N orthogonal line segments.

Pf.

- Put x-coordinates on a PQ (or sort). ← N log N
- Insert y-coordinates into BST. ← N log N
- Delete y-coordinates from BST. ← N log N
- Range searches in BST. ← N log N + R

Bottom line. Sweep line reduces 2d orthogonal line segment intersection search to 1d range search.

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

Geometric Applications of

- *1d range search*
- *line segment intersection*
- *kd trees*
- *interval search trees*
- *rectangle intersection*

2-d orthogonal range search

Extension of ordered symbol-table to 2d keys.

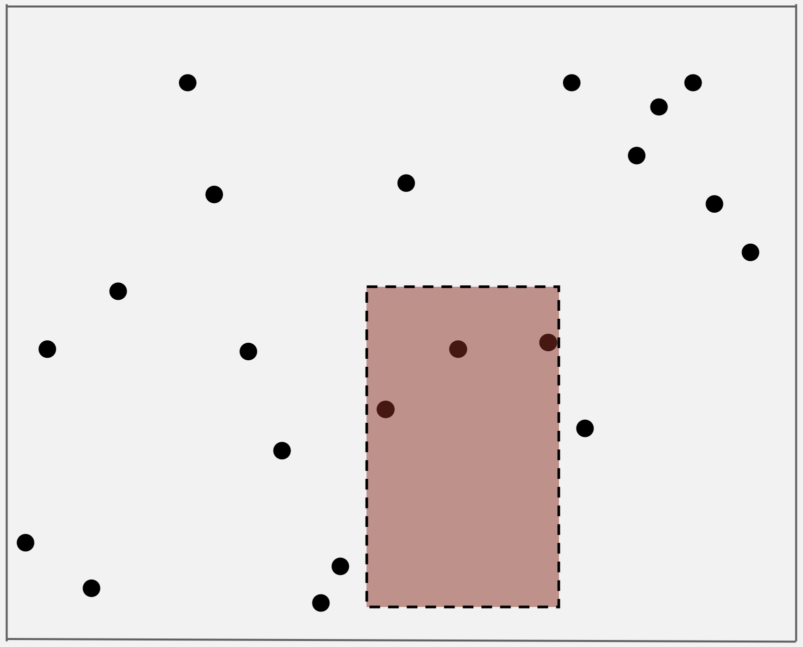
- Insert a 2d key.
- Delete a 2d key.
- Search for a 2d key.
- Range search: find all keys that lie in a 2d range.
- Range count: number of keys that lie in a 2d range.

Applications. Networking, circuit design, databases, ...

Geometric interpretation.

- Keys are point in the plane.
- Find/count points in a given *h*–*v* rectangle

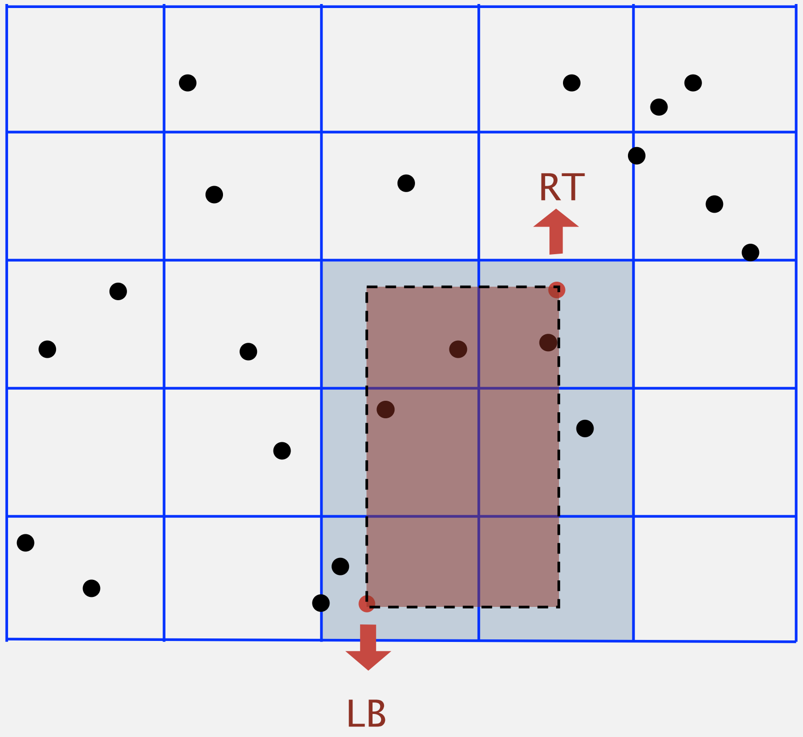
rectangle is axis-aligned



2d orthogonal range search: grid implementation

Grid implementation.

- Divide space into M-by-M grid of squares.
- Create list of points contained in each square.
- Use 2d array to directly index relevant square.
- Insert: add (x, y) to list for corresponding square.
- Range search: examine only squares that intersect 2d range query.



2d orthogonal range search: grid implementation analysis

Space-time tradeoff.

- Space: $M^2 + N$.
- Time: $1 + N / M^2$ per square examined, on average.

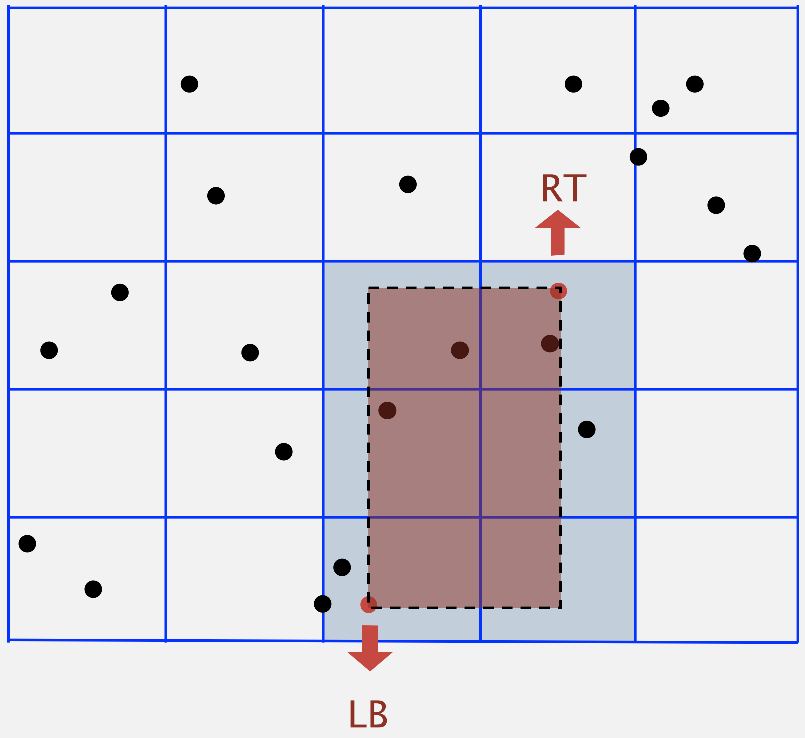
Choose grid square size to tune performance.

- Too small: wastes space.
- Too large: too many points per square.
- Rule of thumb: $\sqrt{N}$-by-$\sqrt{N}$ grid.

Running time. [if points are evenly distributed]

- Initialize data structure: N.
- Insert point: 1.
- Range search: 1 per point in range.

choose M ~ √N

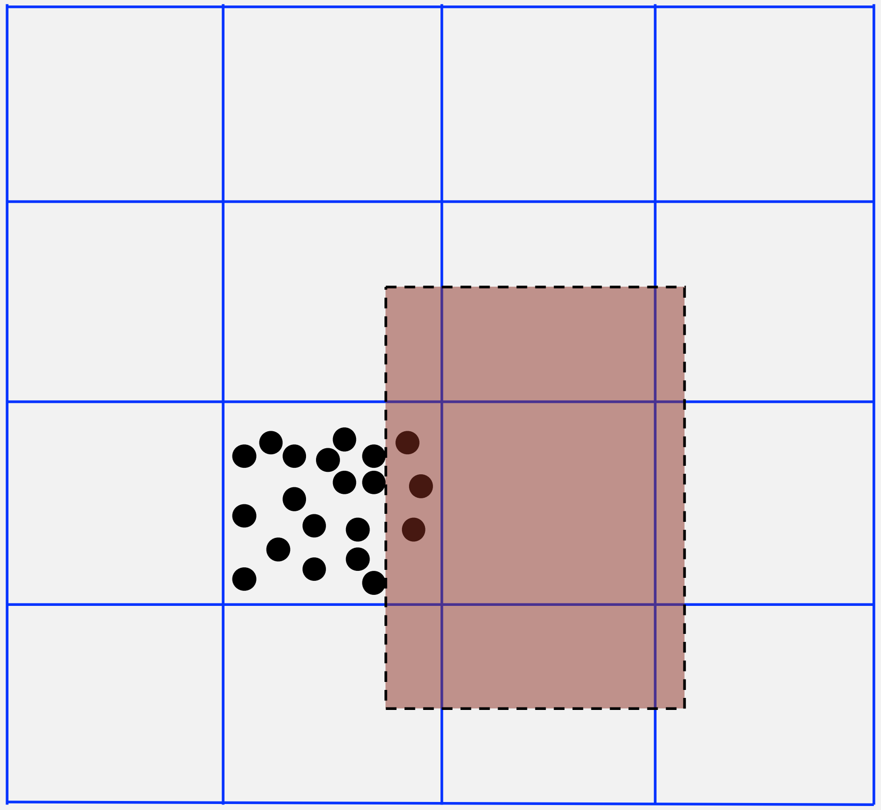


Clustering

Grid implementation. Fast, simple solution for evenly-distributed points.

Problem. Clustering a well-known phenomenon in geometric data.

- Lists are too long, even though average length is short.
- Need data structure that adapts gracefully to data.

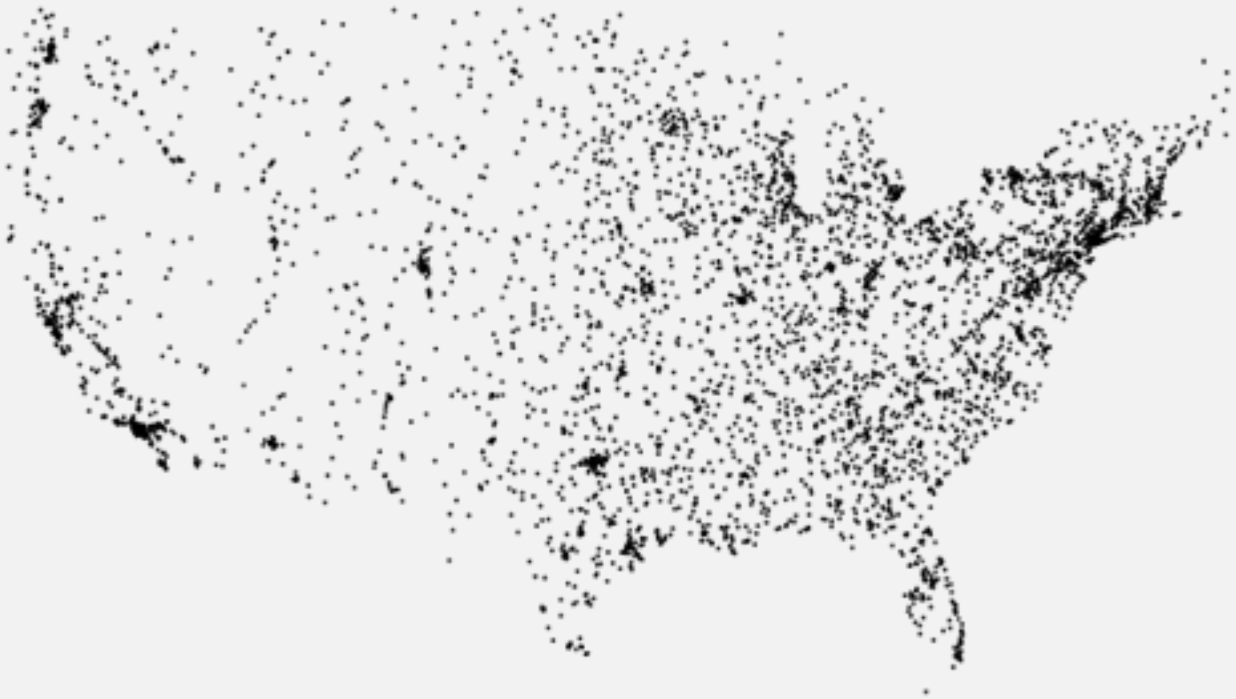


Clustering

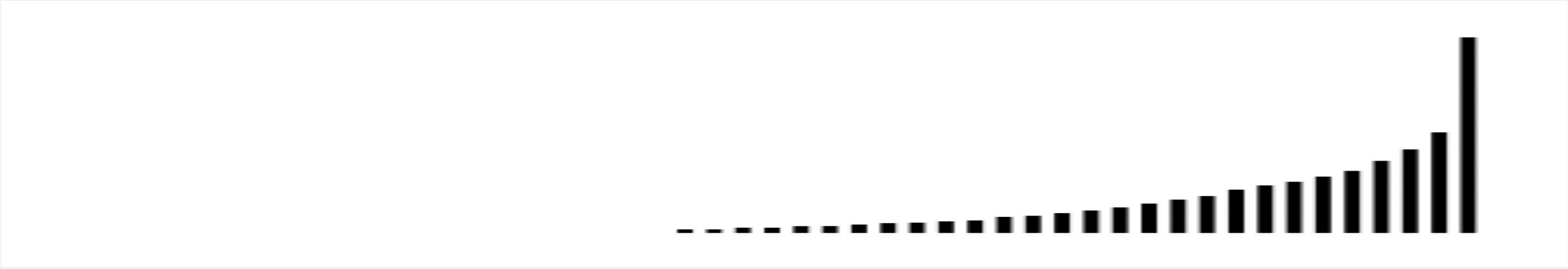
Grid implementation. Fast, simple solution for evenly-distributed points.

Problem. Clustering a well-known phenomenon in geometric data.

Ex. USA map data.



13,000 points, 1000 grid squares



half the squares are empty

half the points are in 10% of the squares

Space-partitioning trees

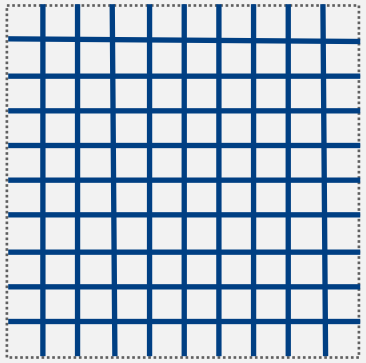
Use a tree to represent a recursive subdivision of 2d space.

Grid. Divide space uniformly into squares.

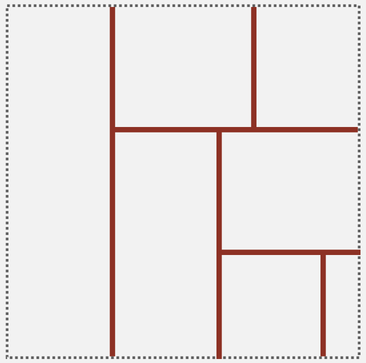
2d tree. Recursively divide space into two halfplanes.

Quadtree. Recursively divide space into four quadrants.

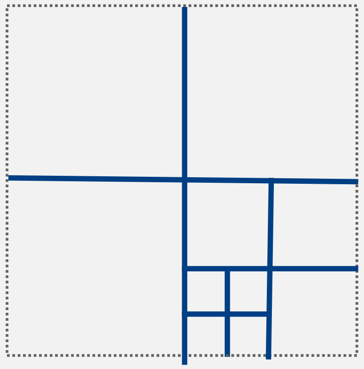
BSP tree. Recursively divide space into two regions.



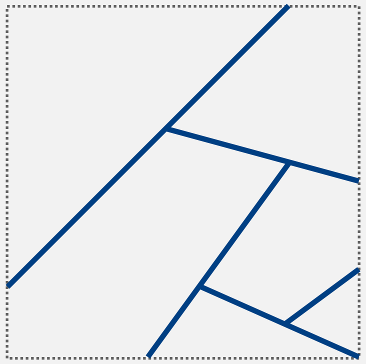
Grid



2d tree



Quadtree

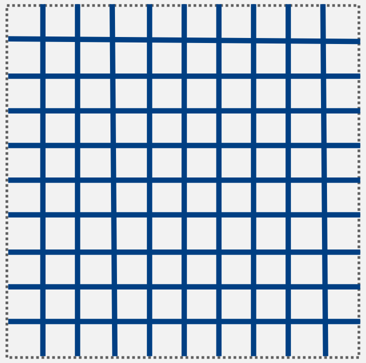


BSP tree

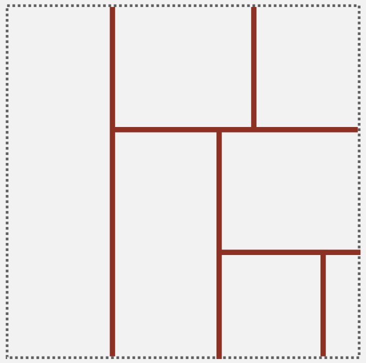
Space-partitioning trees: applications

Applications.

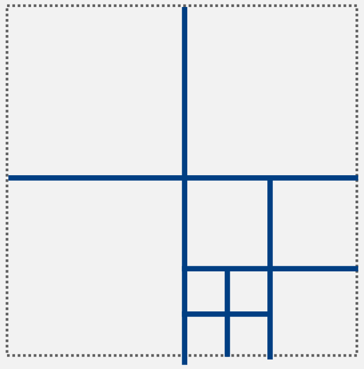
- Ray tracing.
- 2d range search.
- Flight simulators.
- N-body simulation.
- Collision detection.
- Astronomical databases.
- Nearest neighbor search.
- Adaptive mesh generation.
- Accelerate rendering in Doom.
- Hidden surface removal and shadow casting.



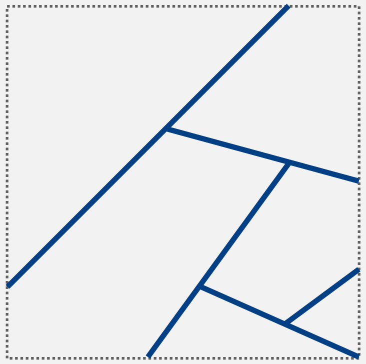
Grid



2d tree



Quadtree

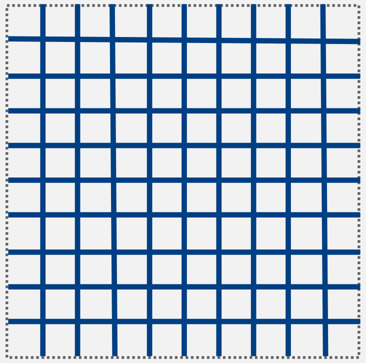


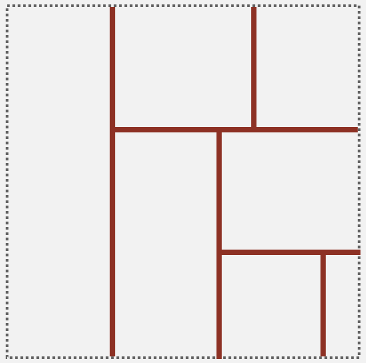
BSP tree

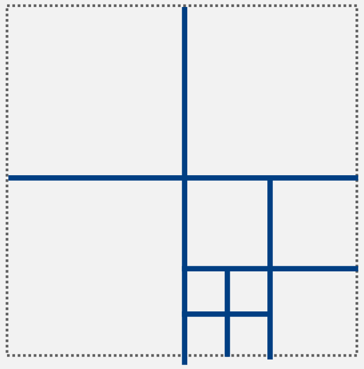
Space-partitioning trees: applications

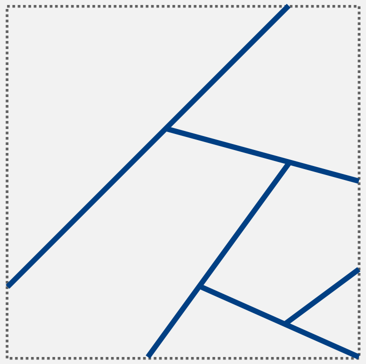
Applications.

- Ray tracing.
- 2d range search.
- Flight simulators.
- N-body simulation.
- Collision detection.
- Astronomical databases.
- Nearest neighbor search.
- Adaptive mesh generation.
- Accelerate rendering in Doom.
- Hidden surface removal and shadow casting.


Grid


2d tree


Quadtree

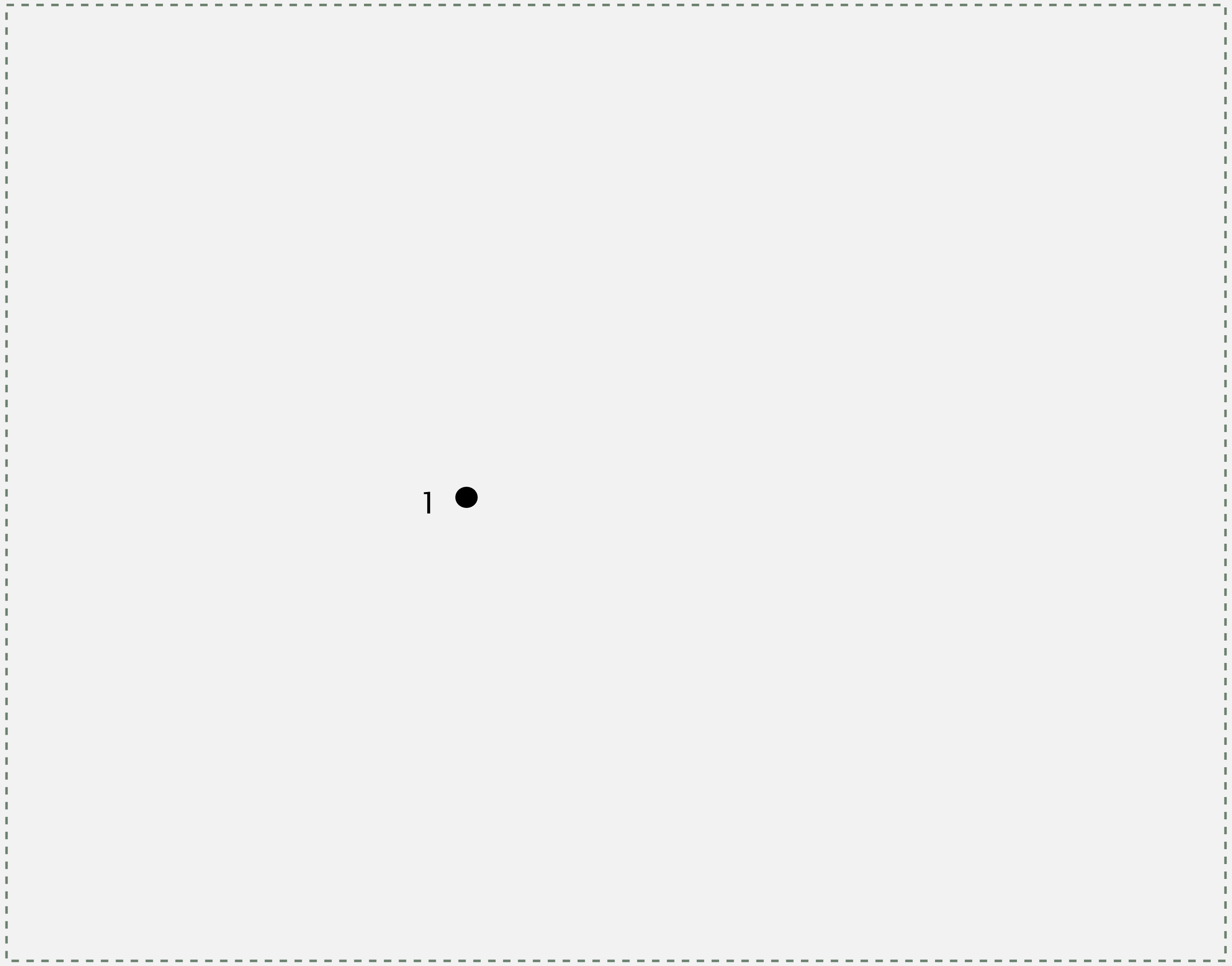

BSP tree

2d tree construction

Recursively partition plane into two halfplanes.

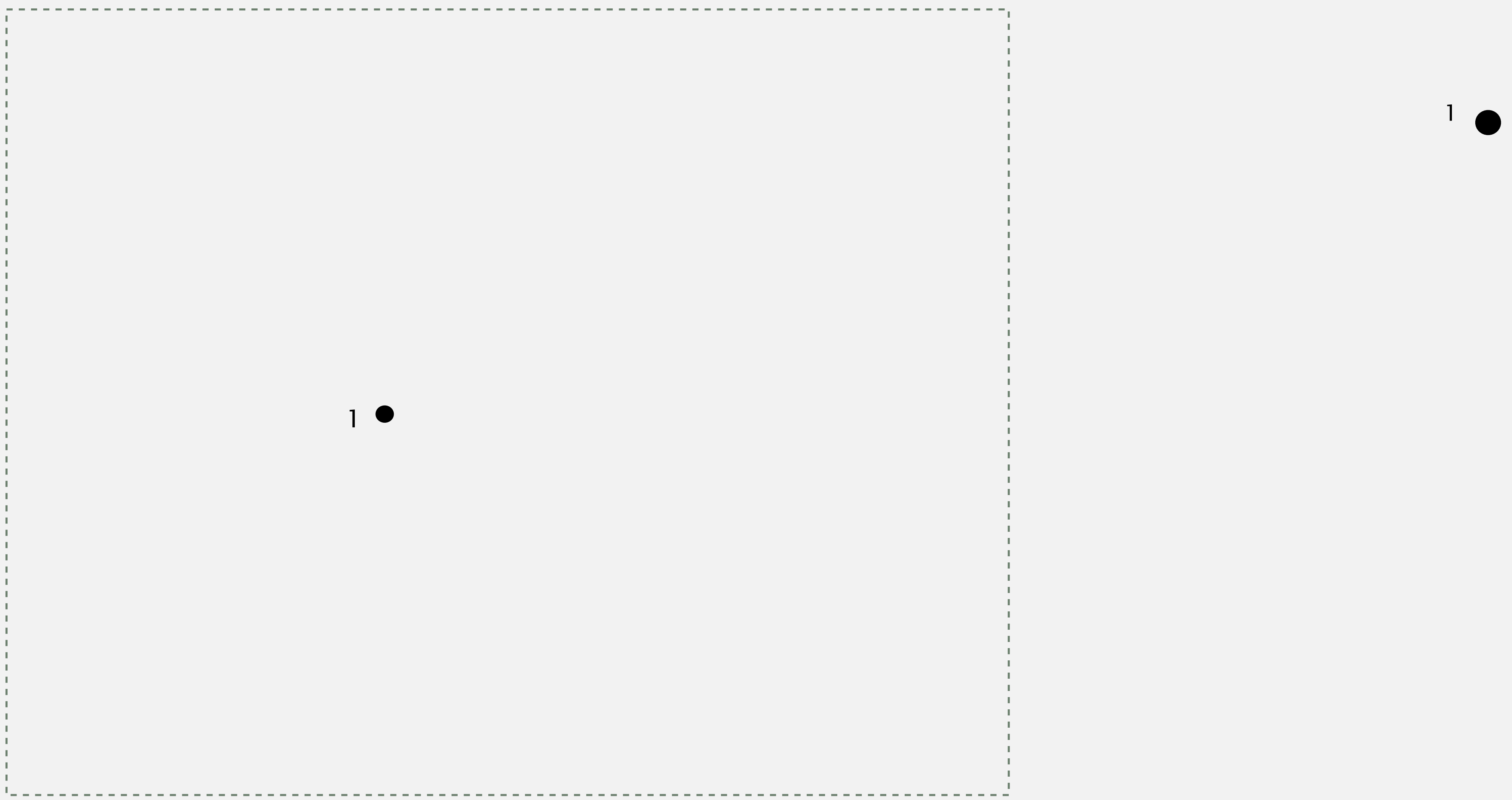
2d tree construction

Recursively partition plane into two halfplanes.



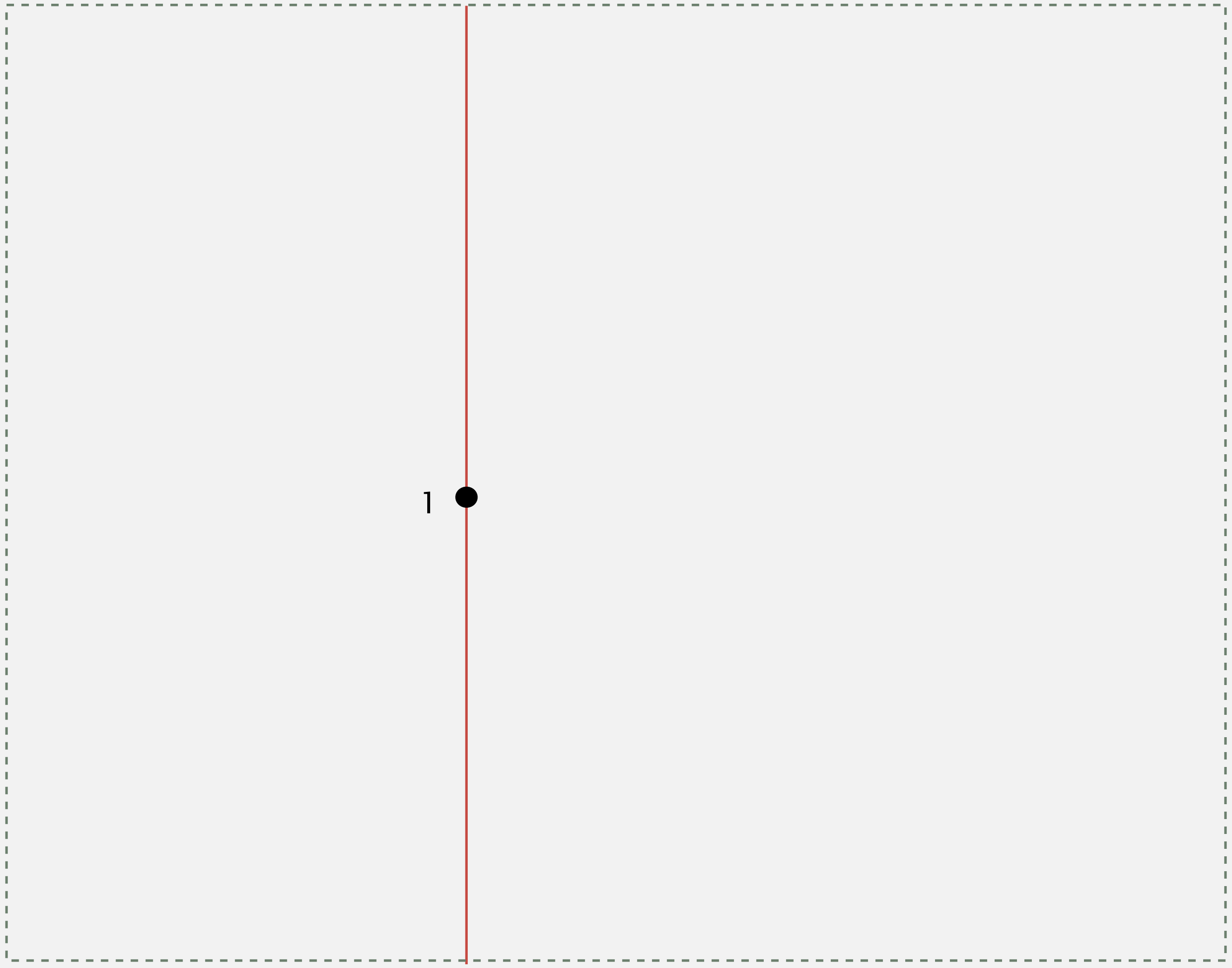
2d tree construction

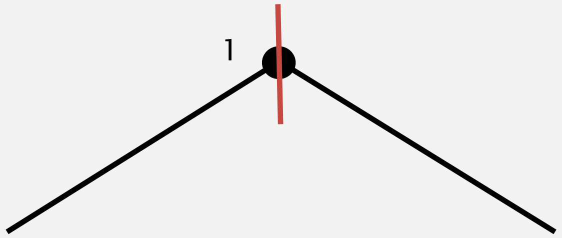
Recursively partition plane into two halfplanes.



2d tree construction

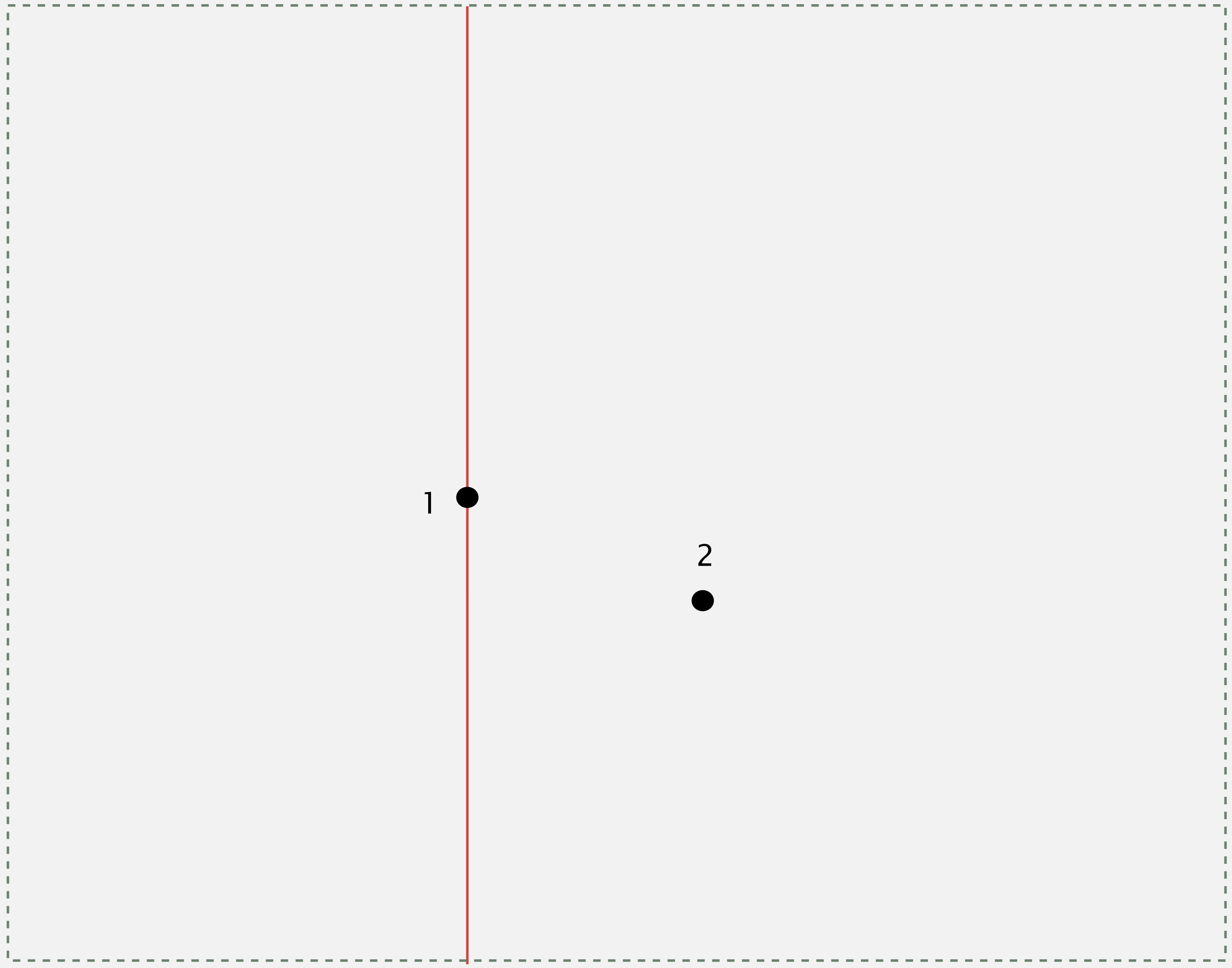
Recursively partition plane into two halfplanes.

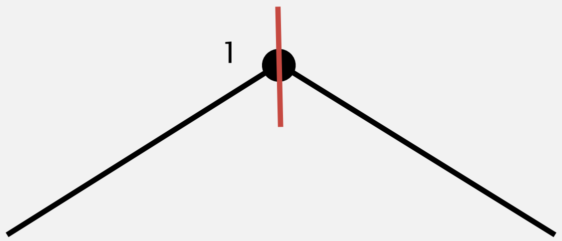




2d tree construction

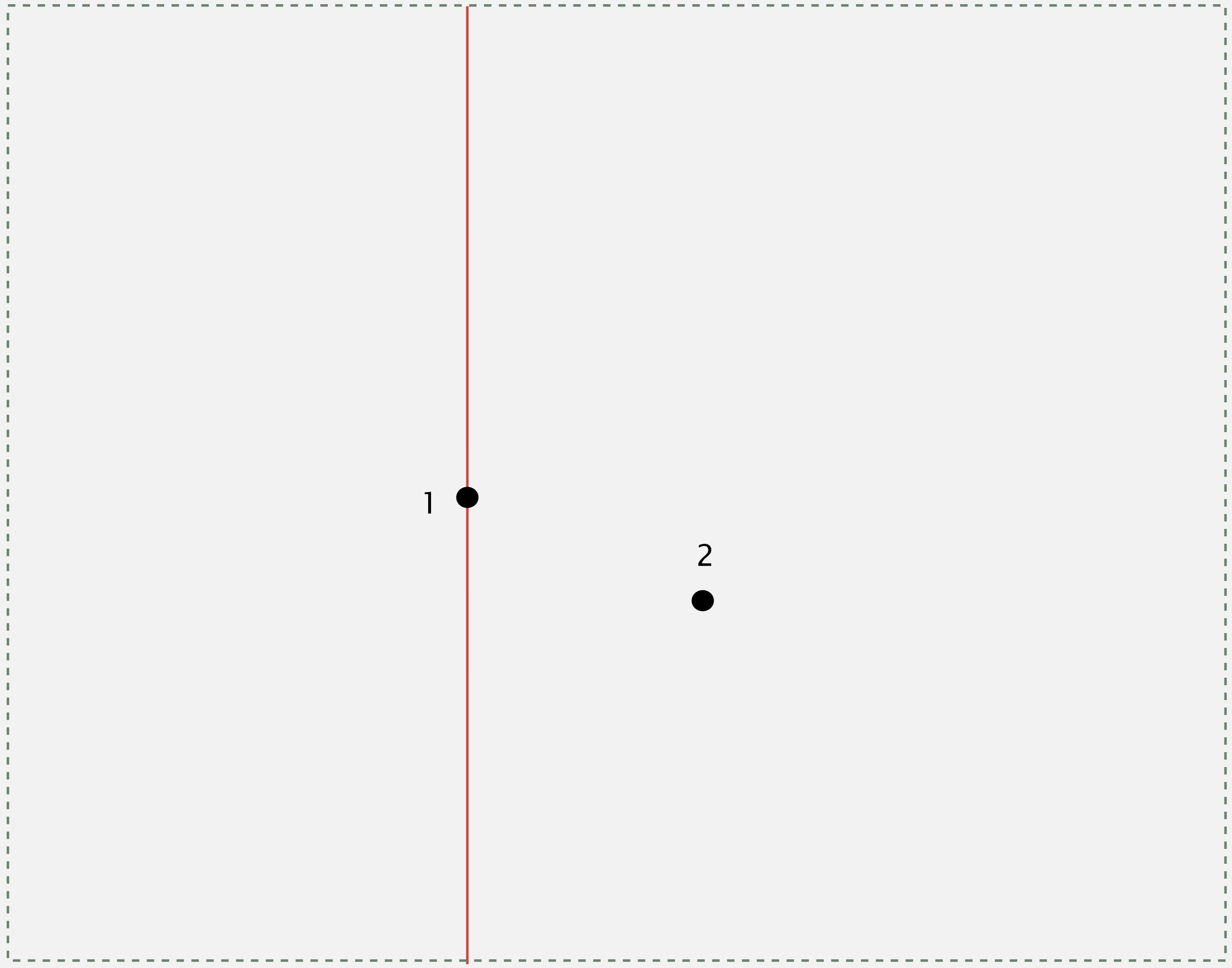
Recursively partition plane into two halfplanes.

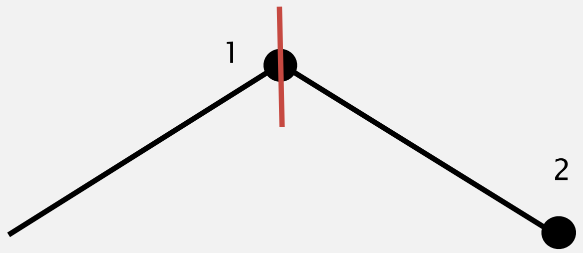




2d tree construction

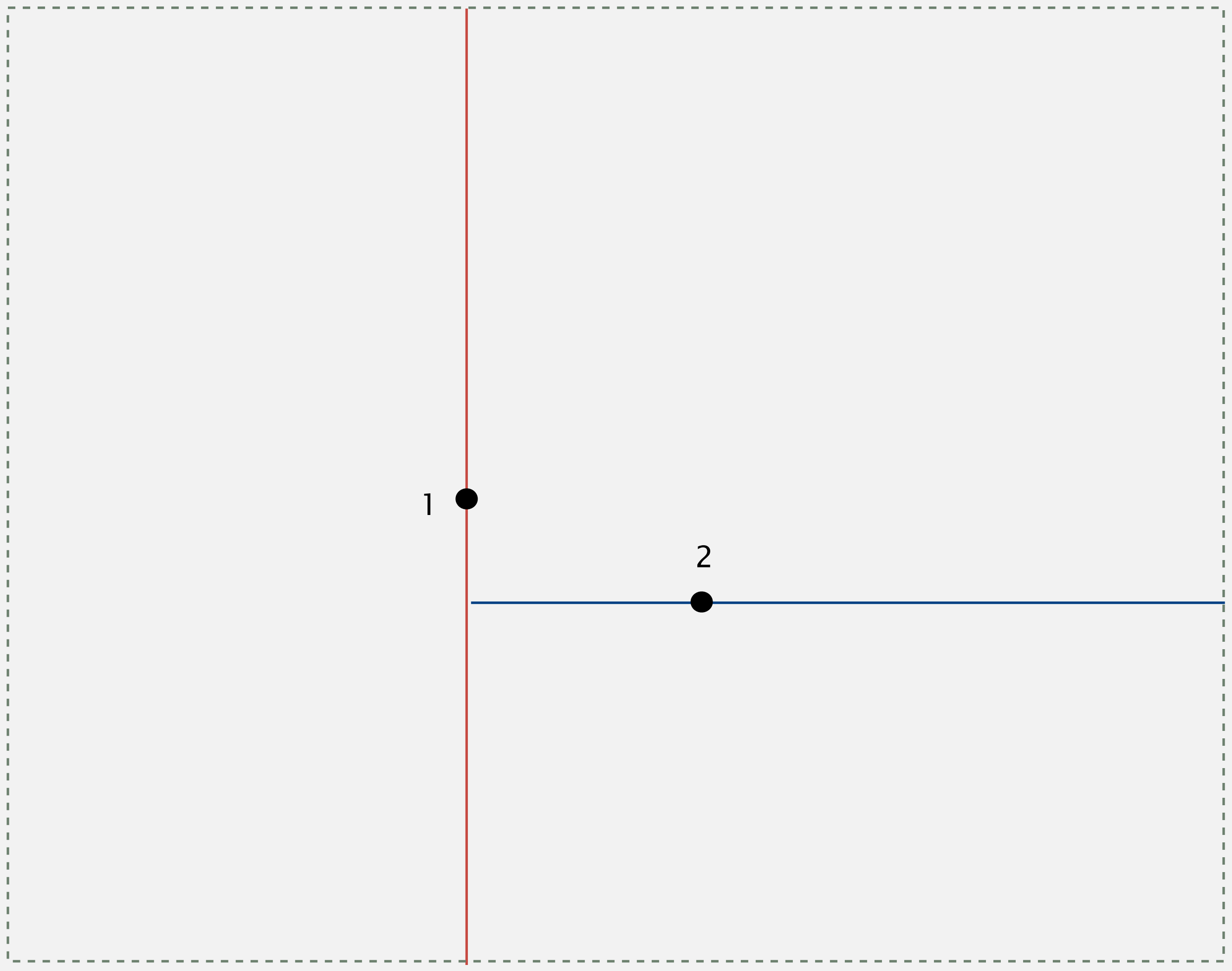
Recursively partition plane into two halfplanes.

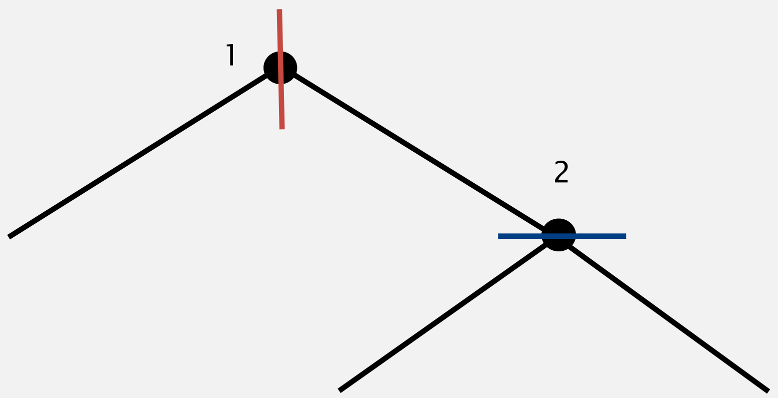




2d tree construction

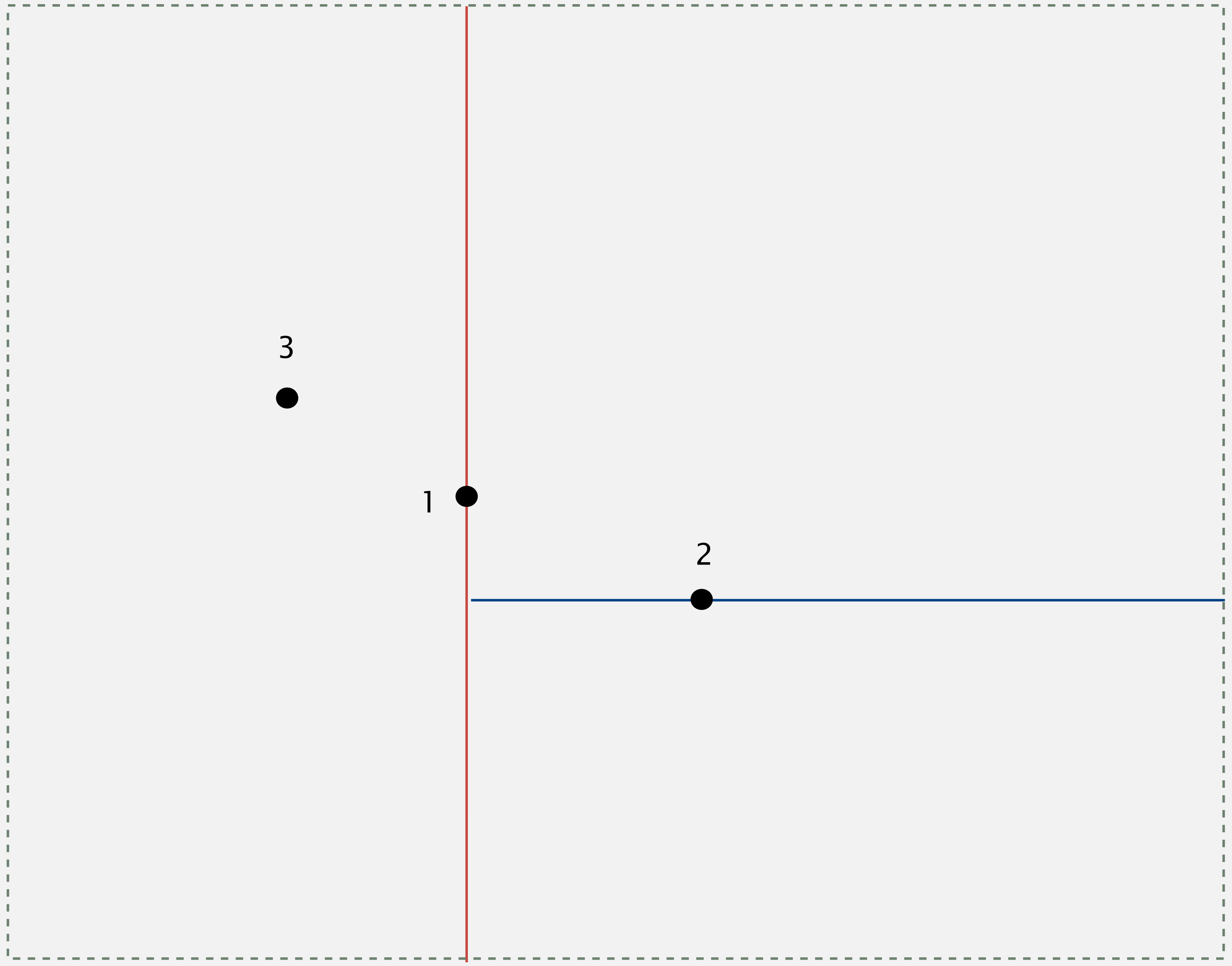
Recursively partition plane into two halfplanes.

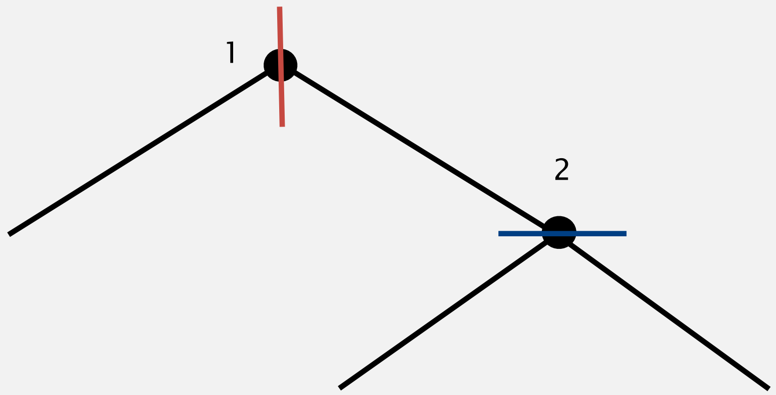




2d tree construction

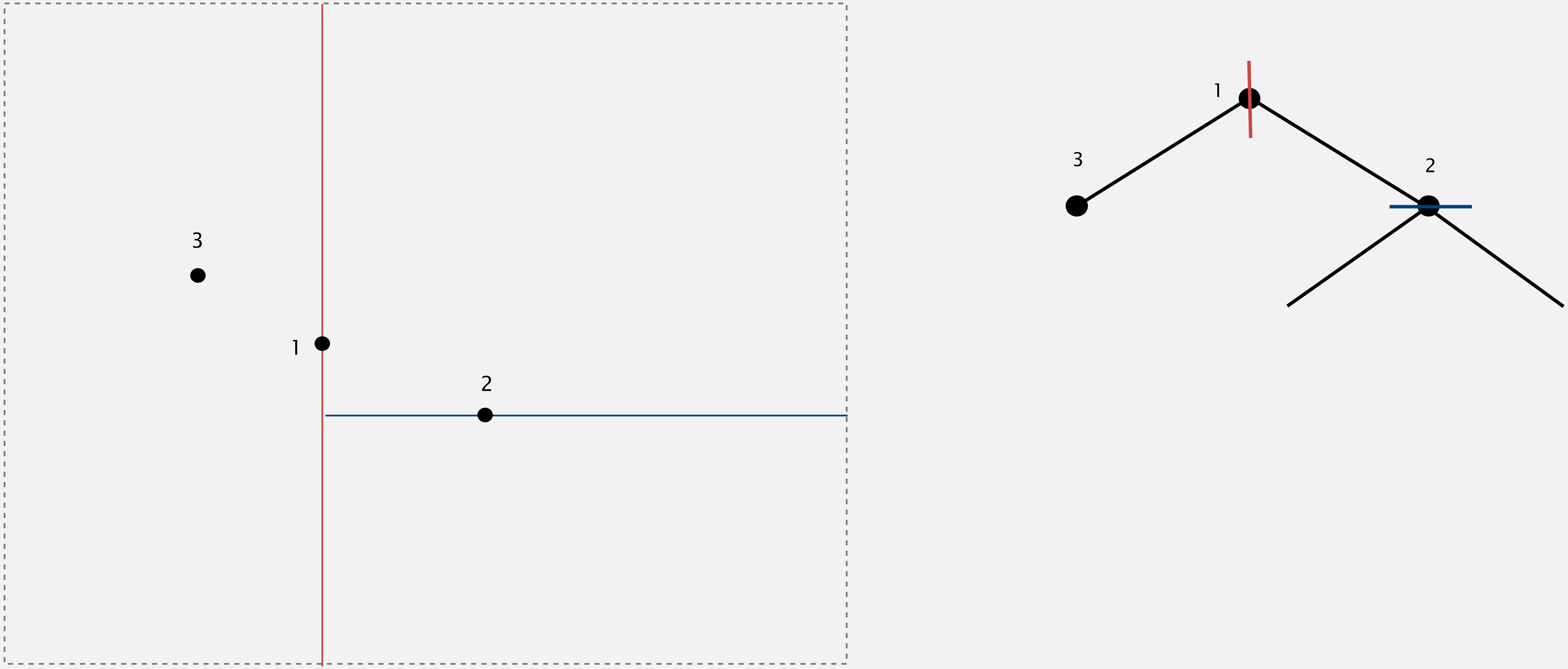
Recursively partition plane into two halfplanes.





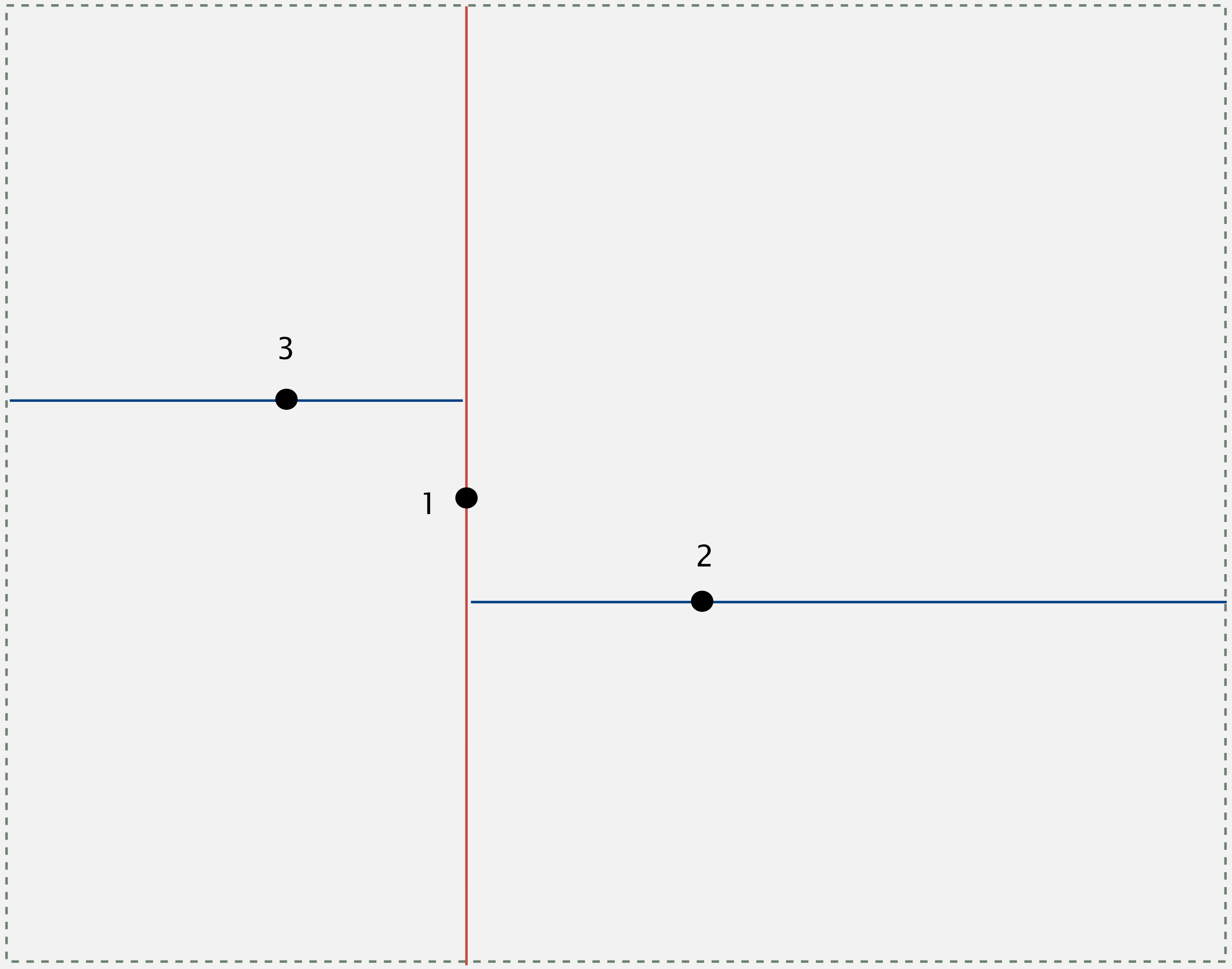
2d tree construction

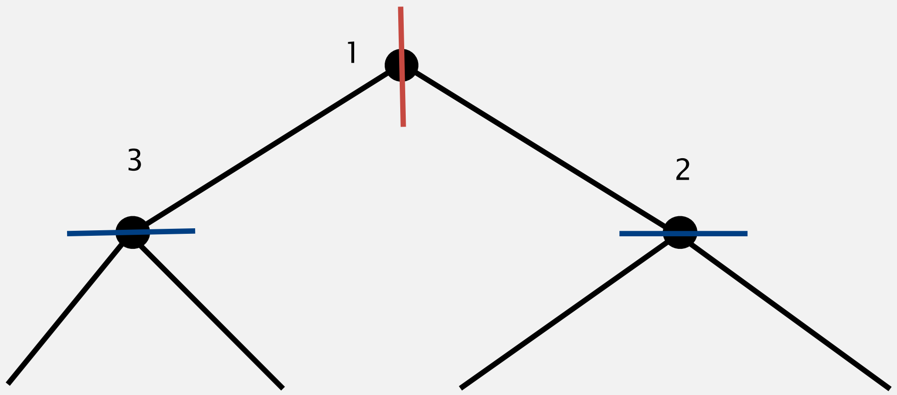
Recursively partition plane into two halfplanes.



2d tree construction

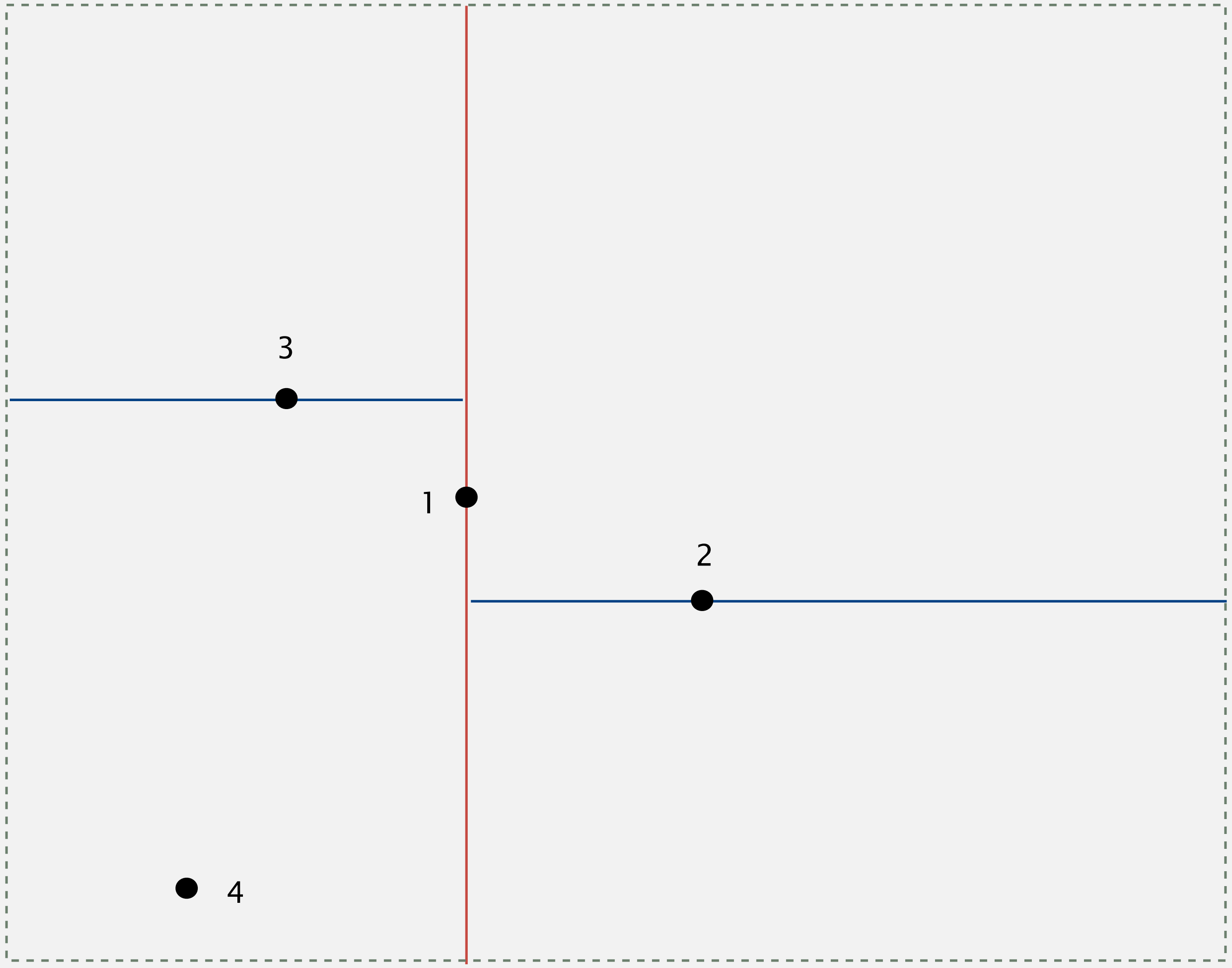
Recursively partition plane into two halfplanes.

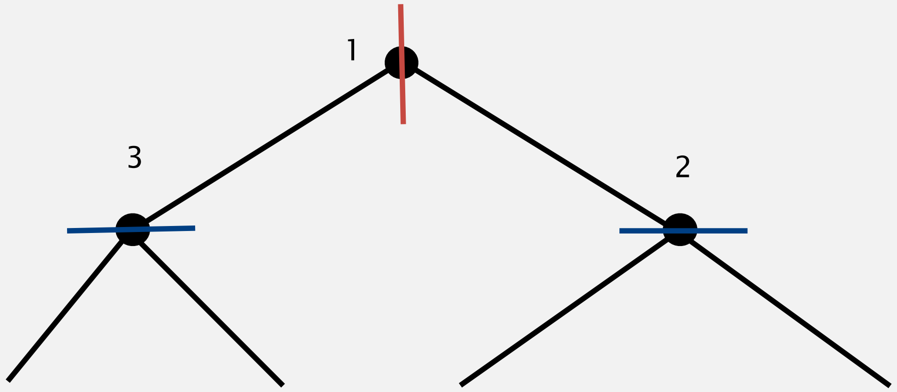




2d tree construction

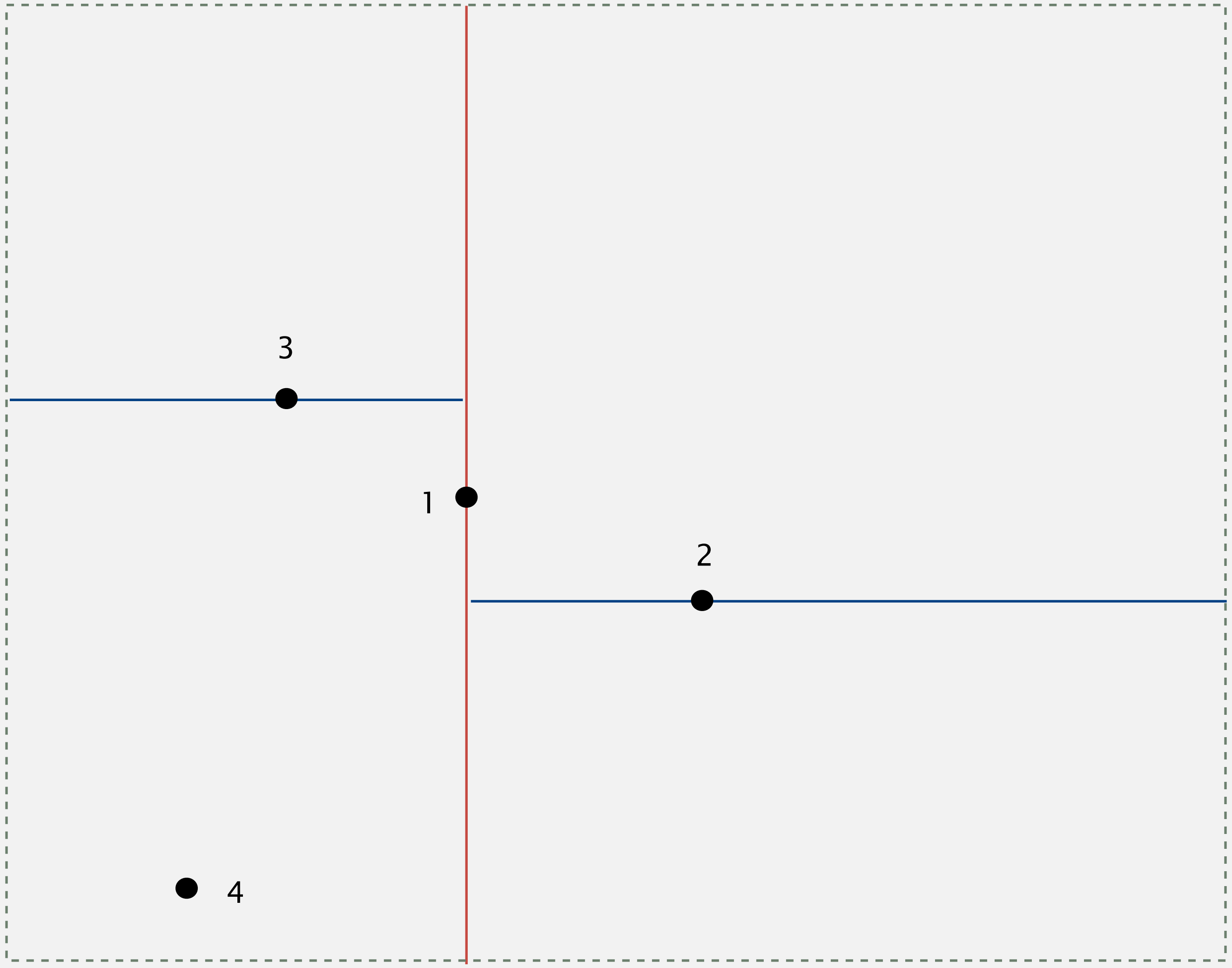
Recursively partition plane into two halfplanes.

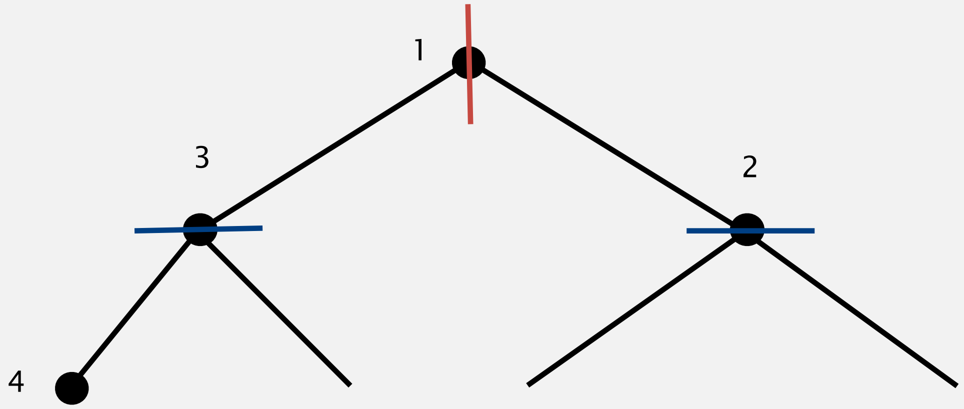




2d tree construction

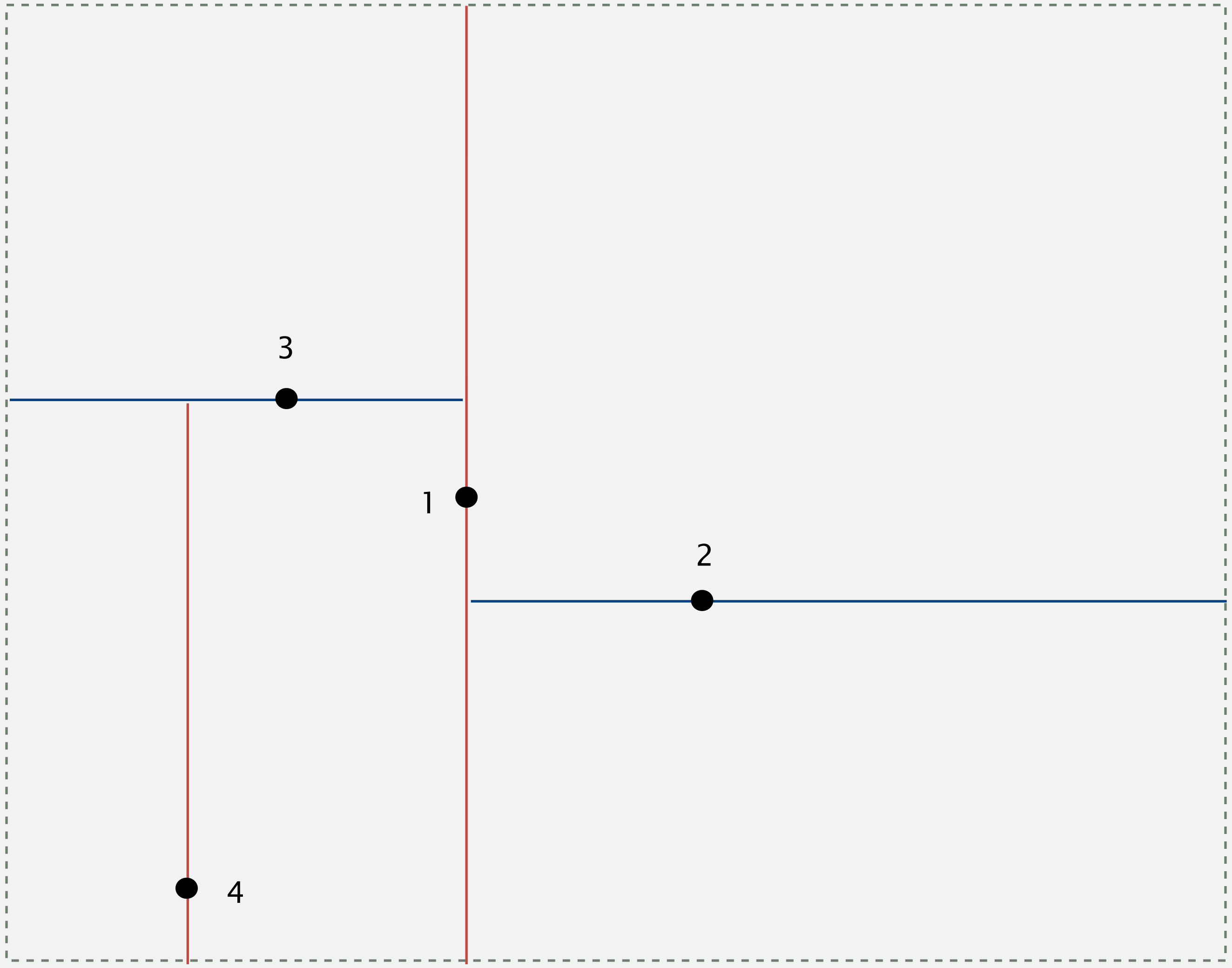
Recursively partition plane into two halfplanes.

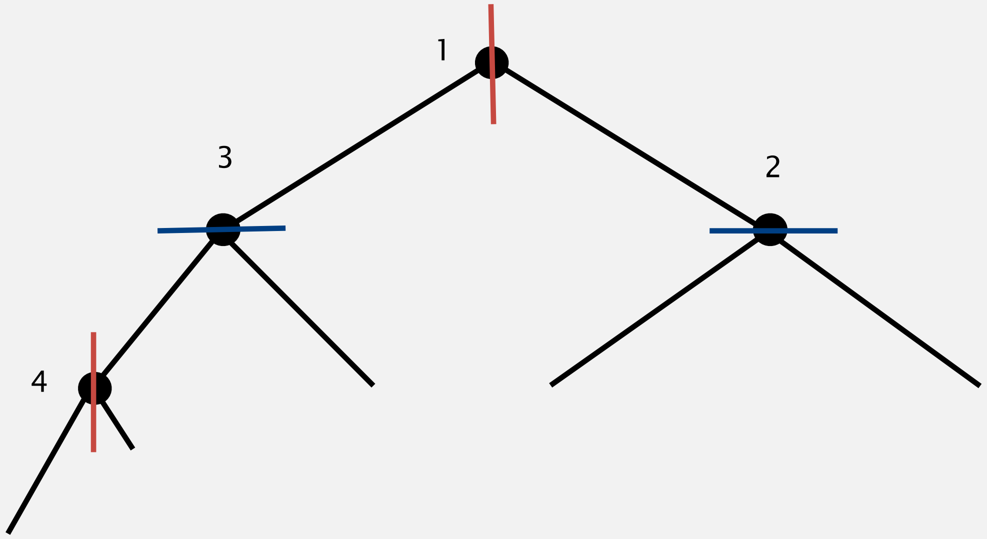




2d tree construction

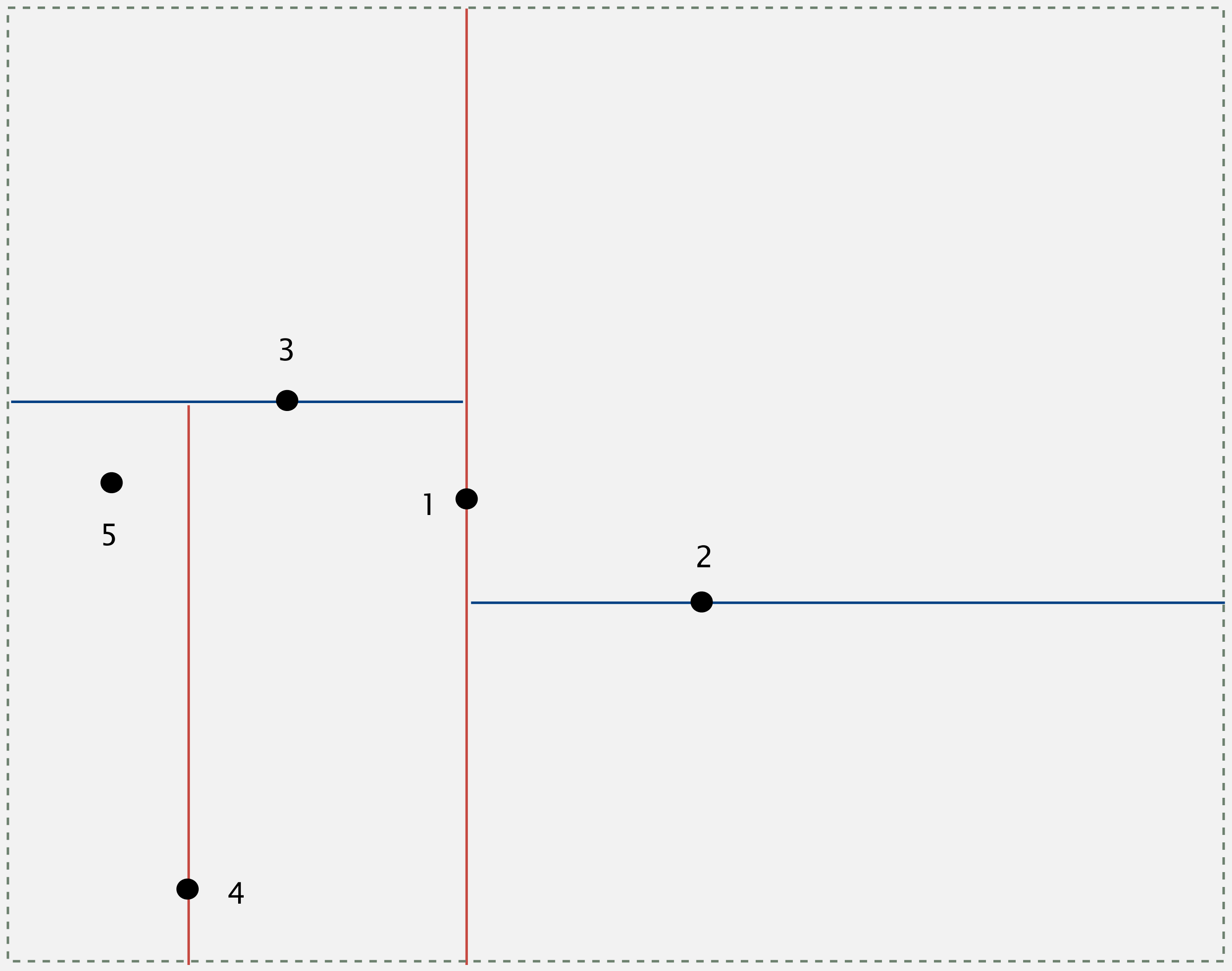
Recursively partition plane into two halfplanes.

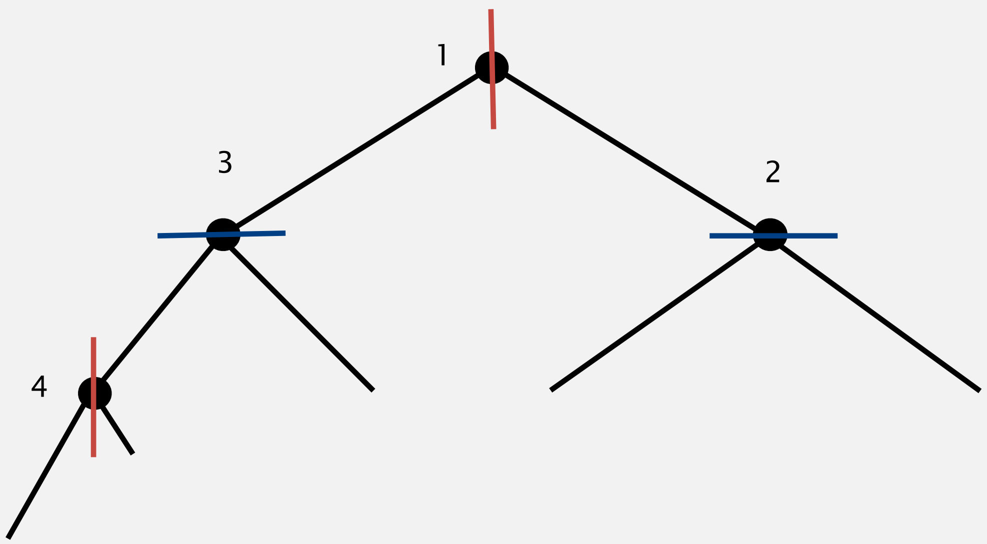




2d tree construction

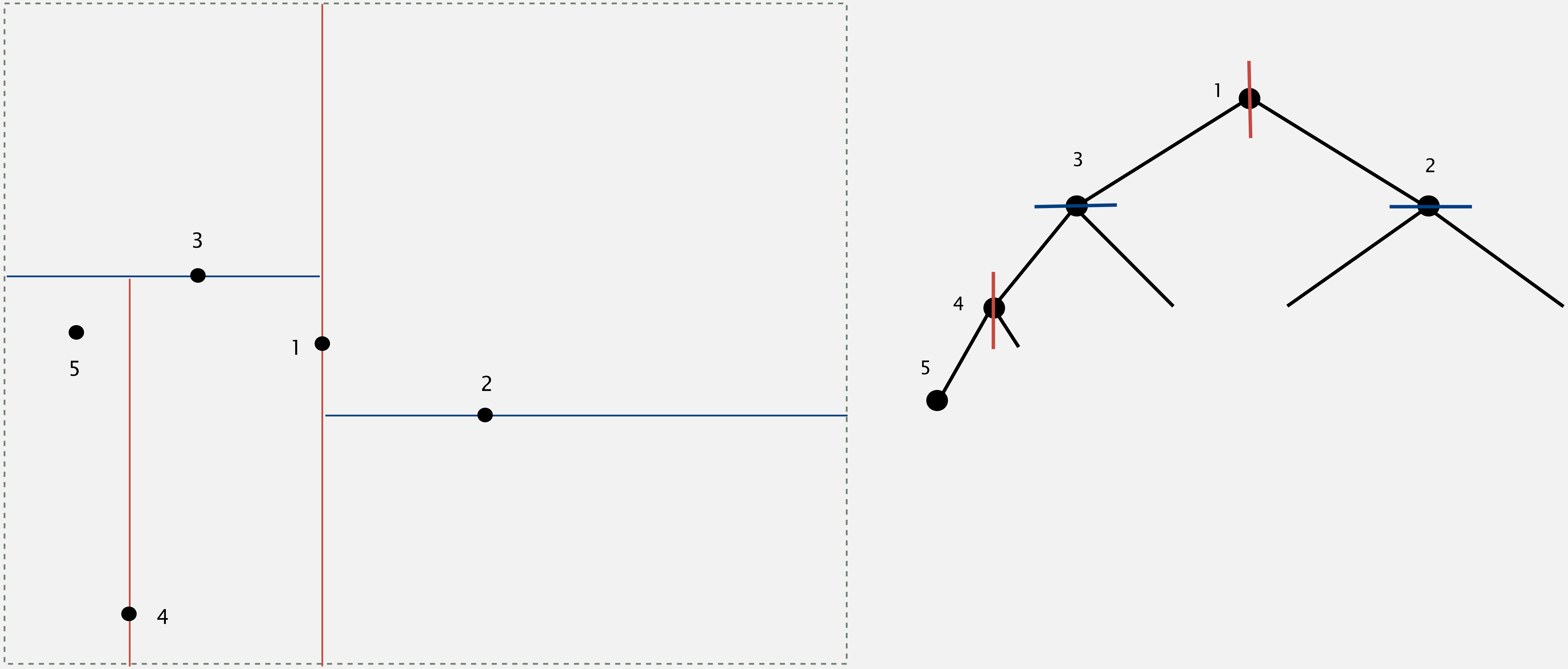
Recursively partition plane into two halfplanes.





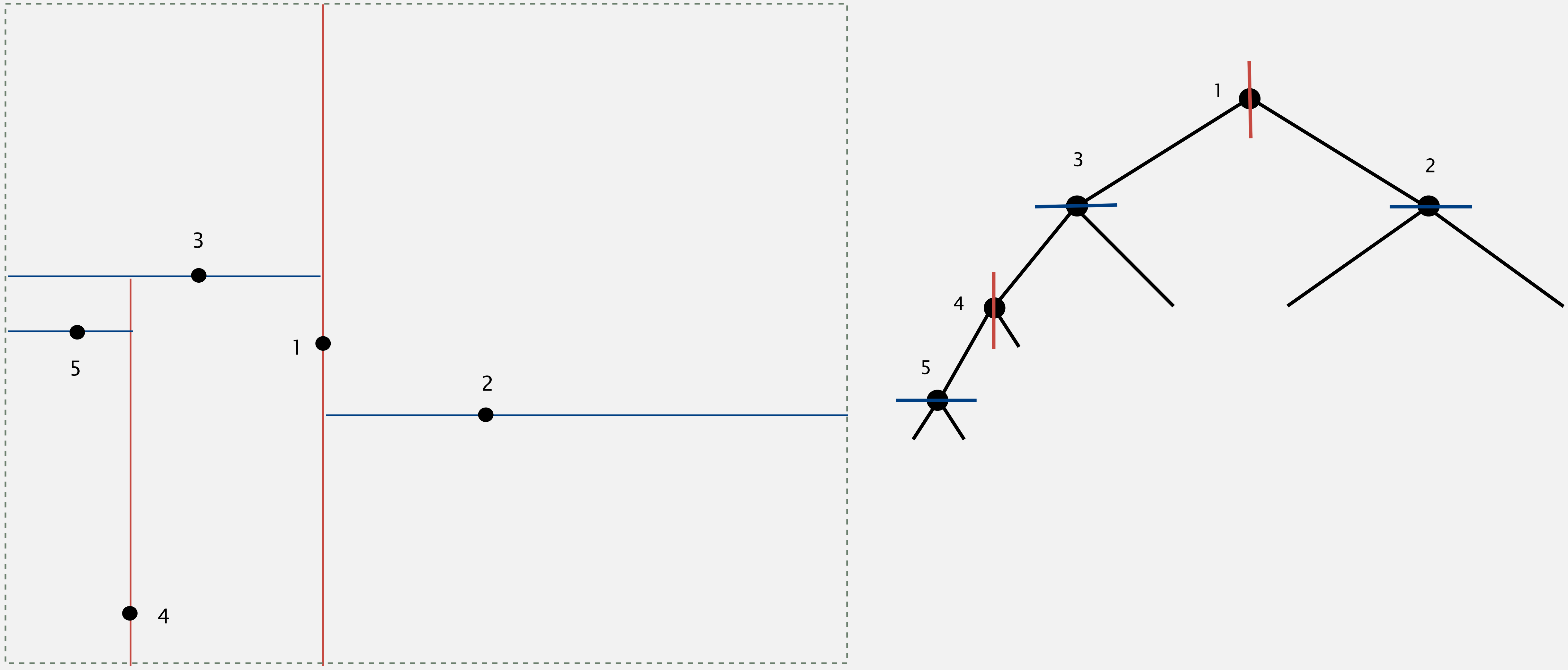
2d tree construction

Recursively partition plane into two halfplanes.



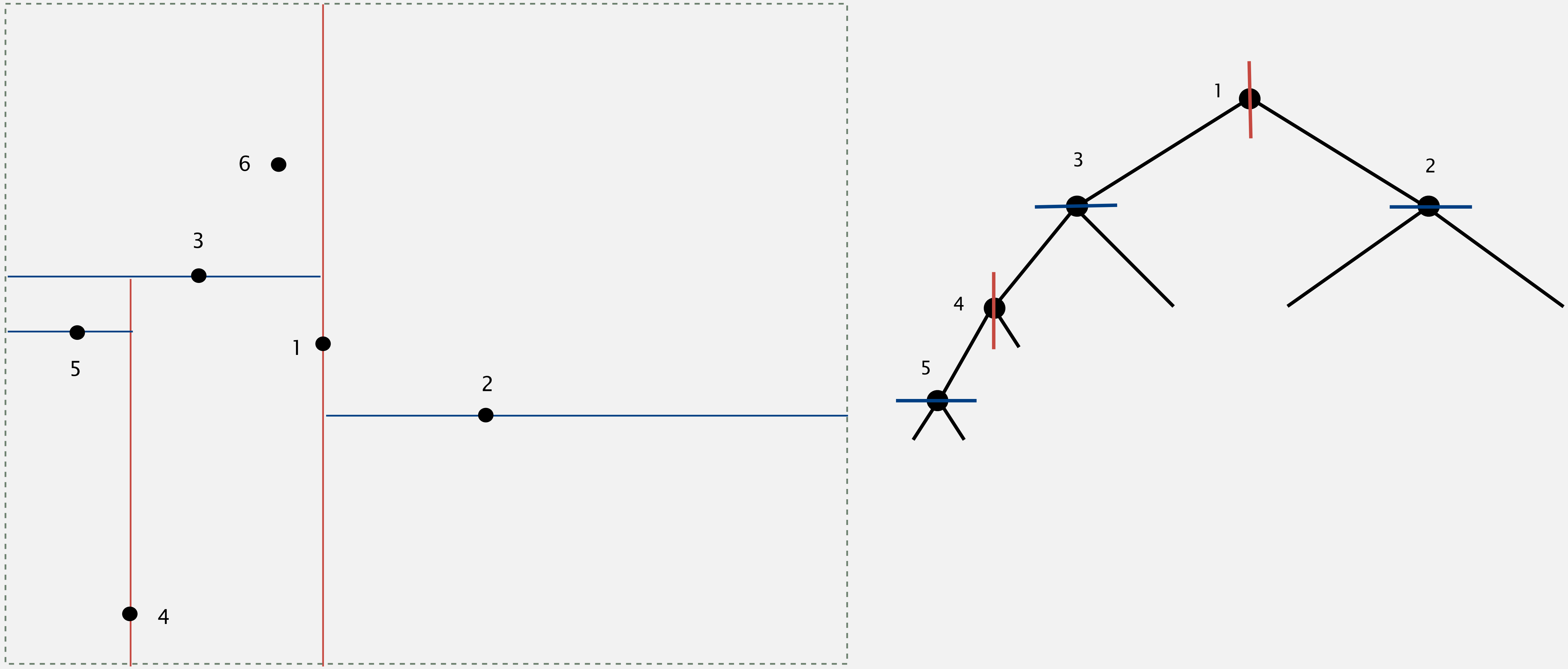
2d tree construction

Recursively partition plane into two halfplanes.



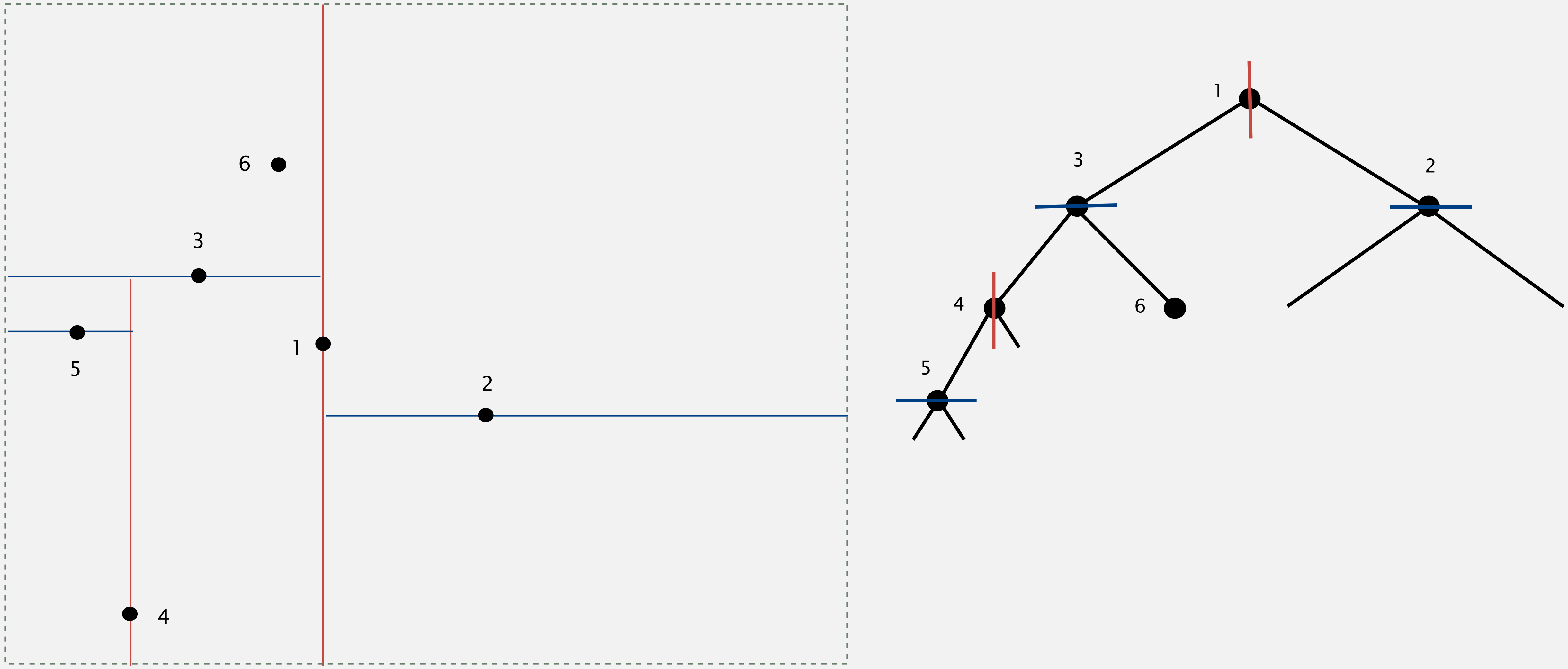
2d tree construction

Recursively partition plane into two halfplanes.



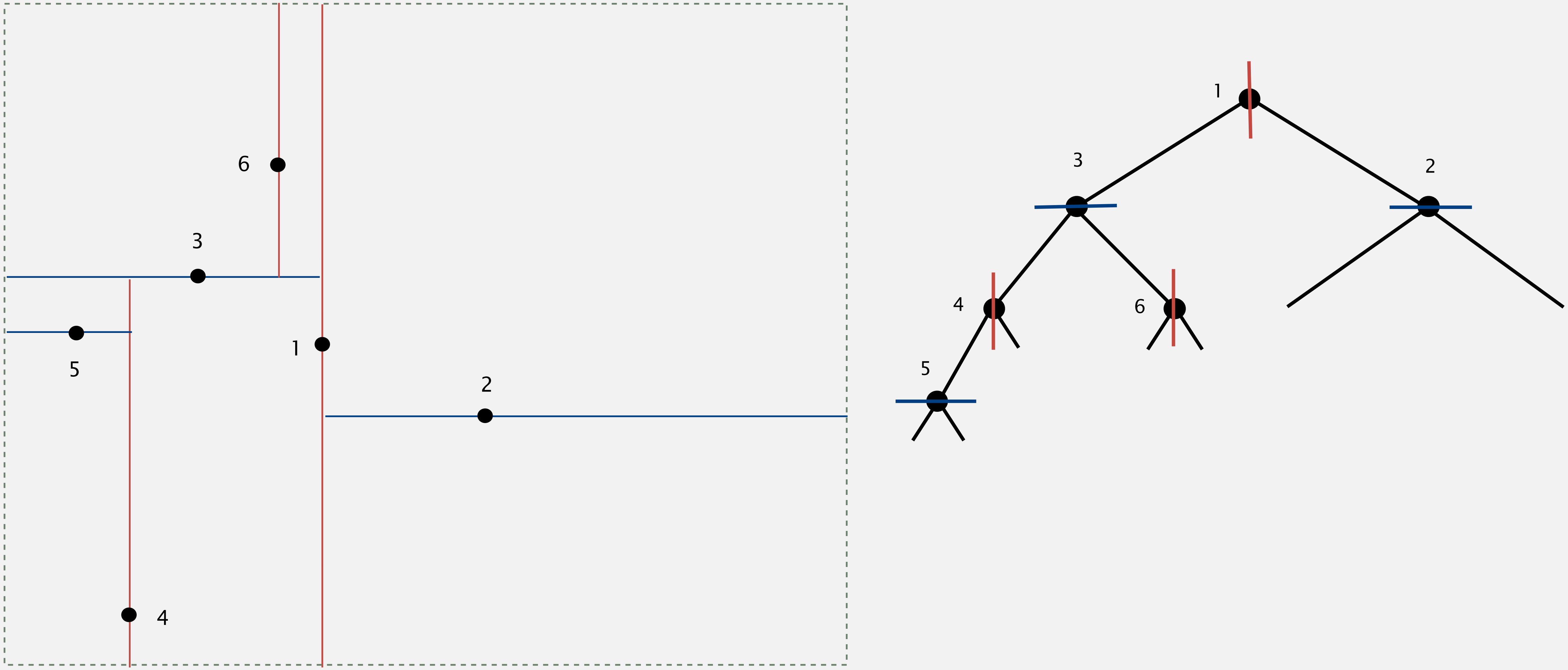
2d tree construction

Recursively partition plane into two halfplanes.



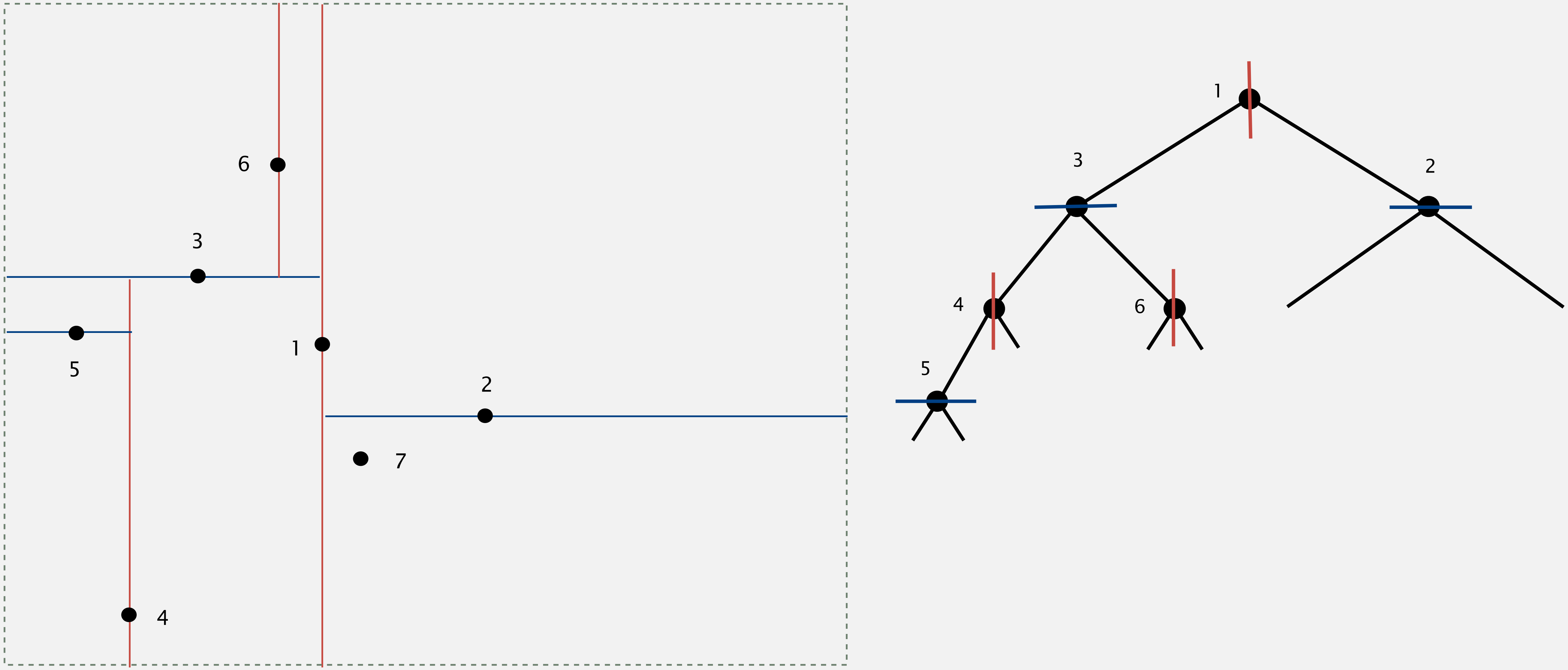
2d tree construction

Recursively partition plane into two halfplanes.



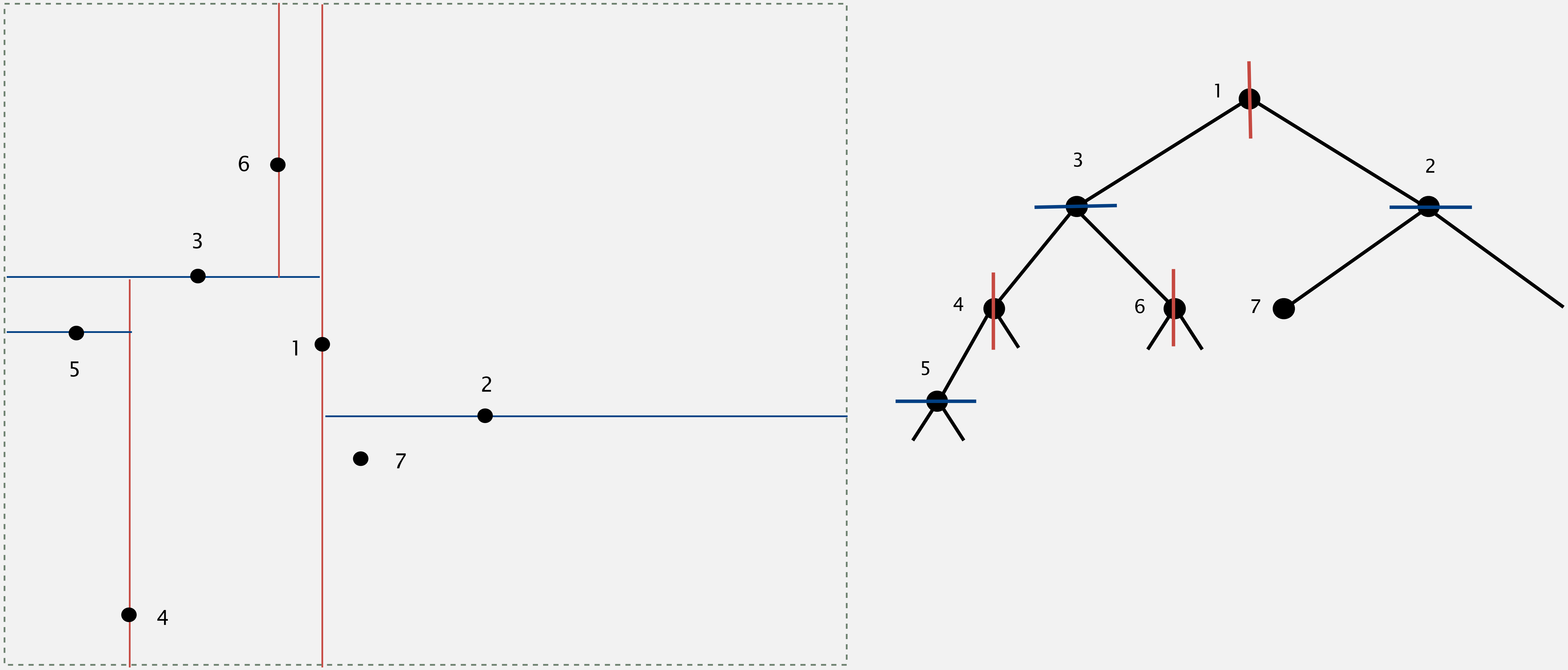
2d tree construction

Recursively partition plane into two halfplanes.



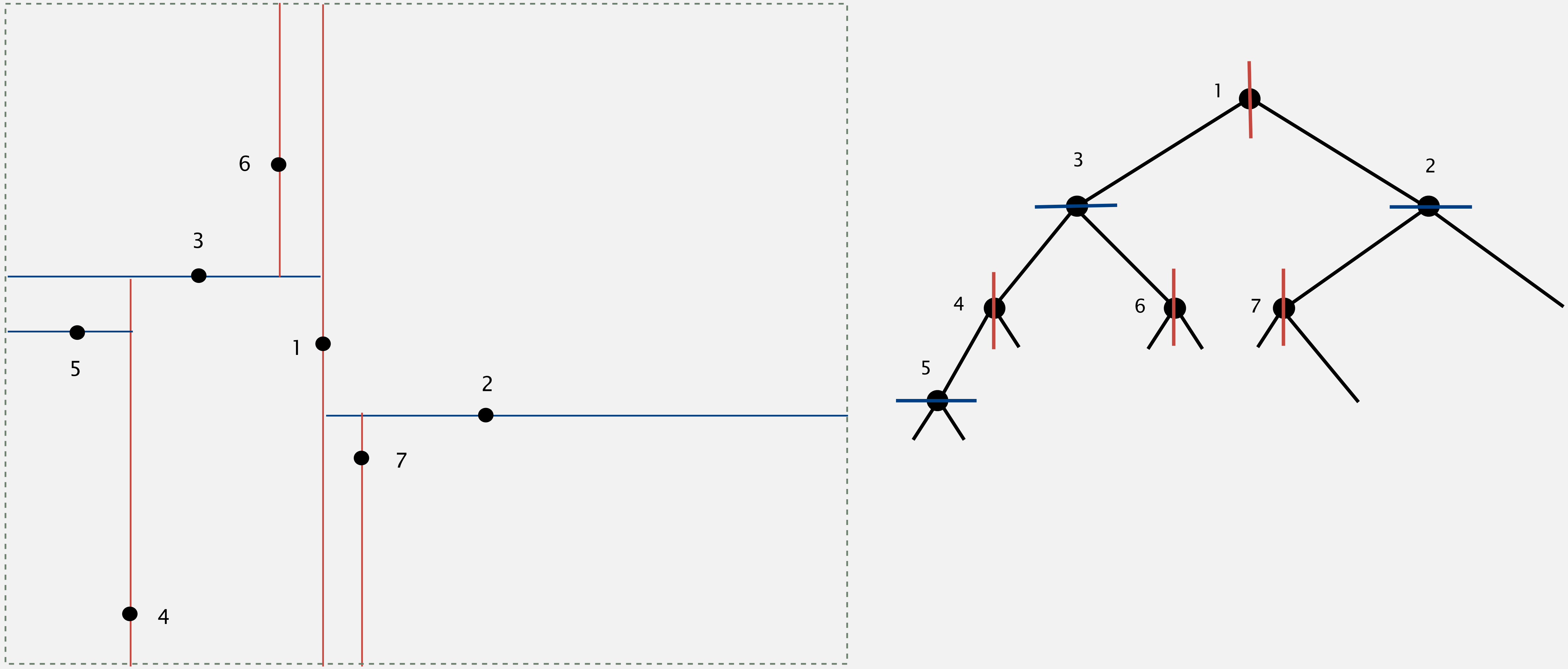
2d tree construction

Recursively partition plane into two halfplanes.



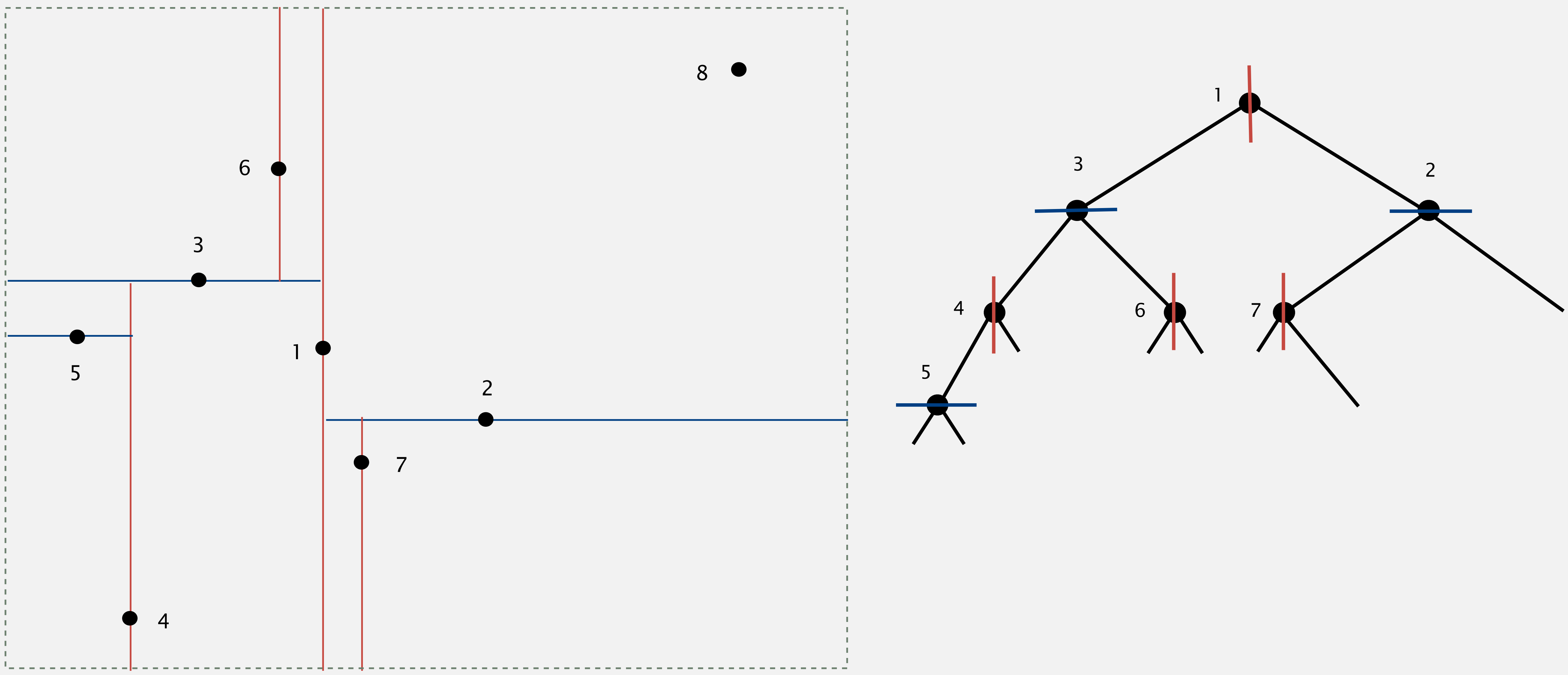
2d tree construction

Recursively partition plane into two halfplanes.



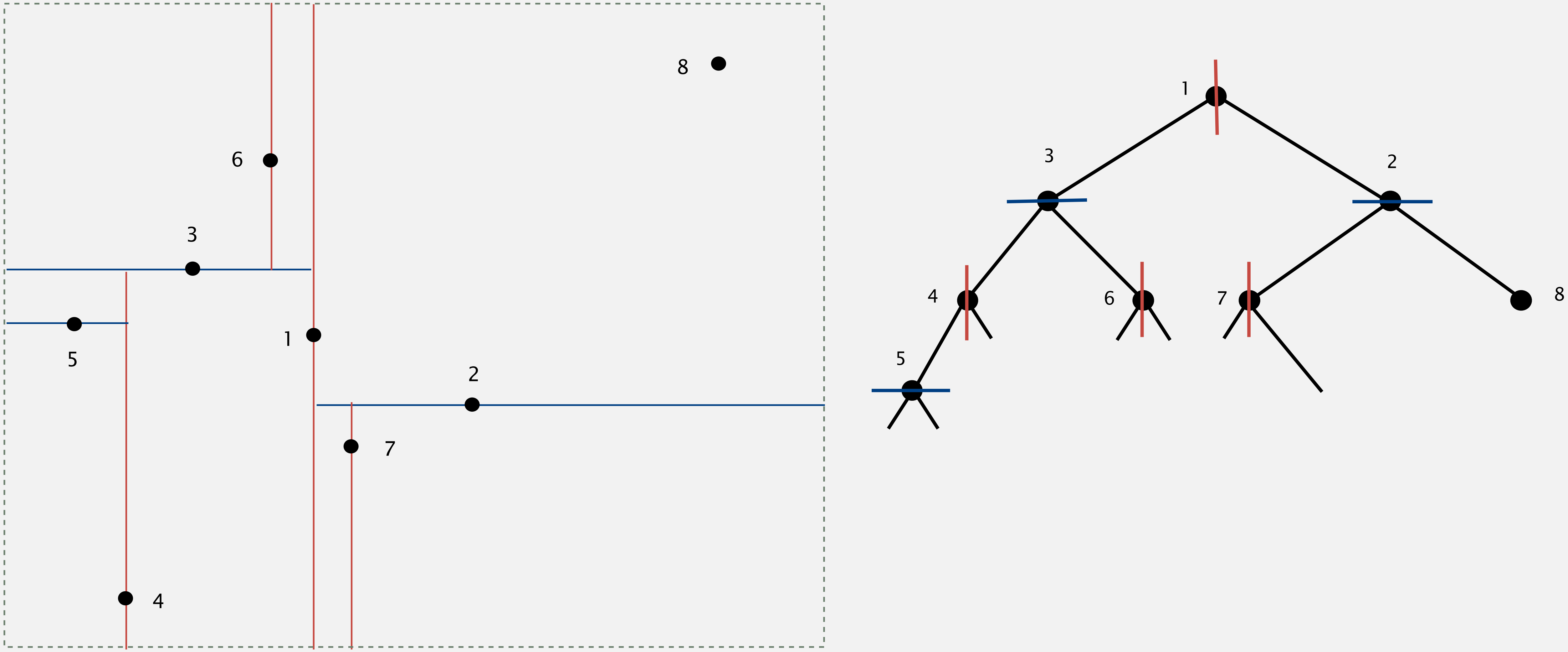
2d tree construction

Recursively partition plane into two halfplanes.



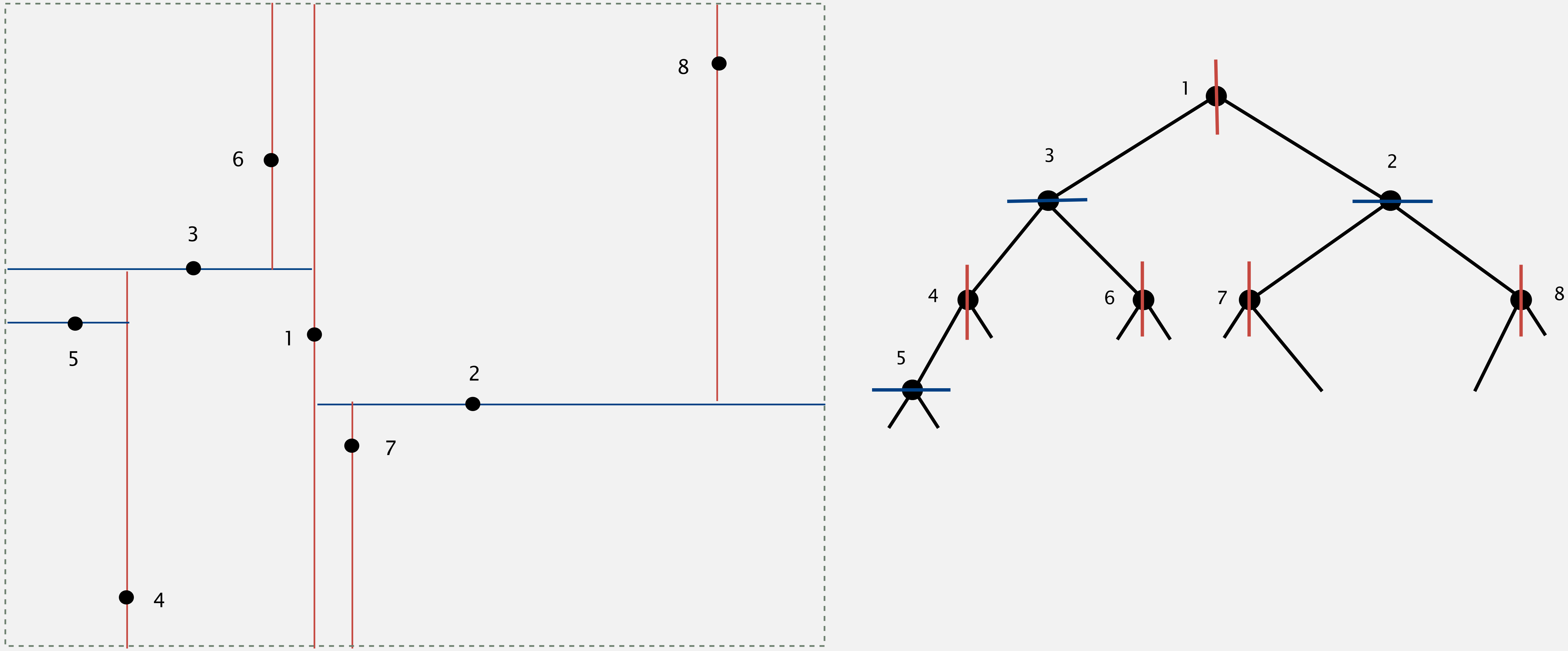
2d tree construction

Recursively partition plane into two halfplanes.



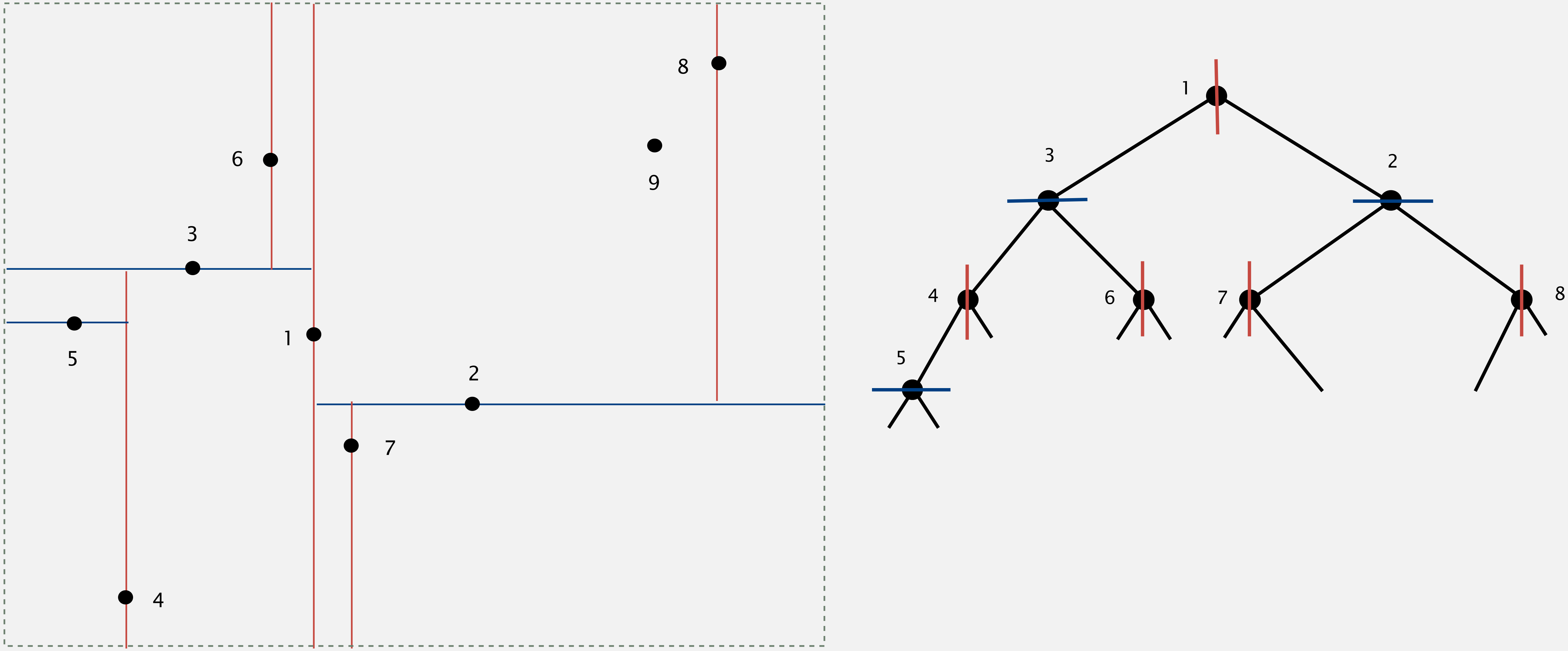
2d tree construction

Recursively partition plane into two halfplanes.



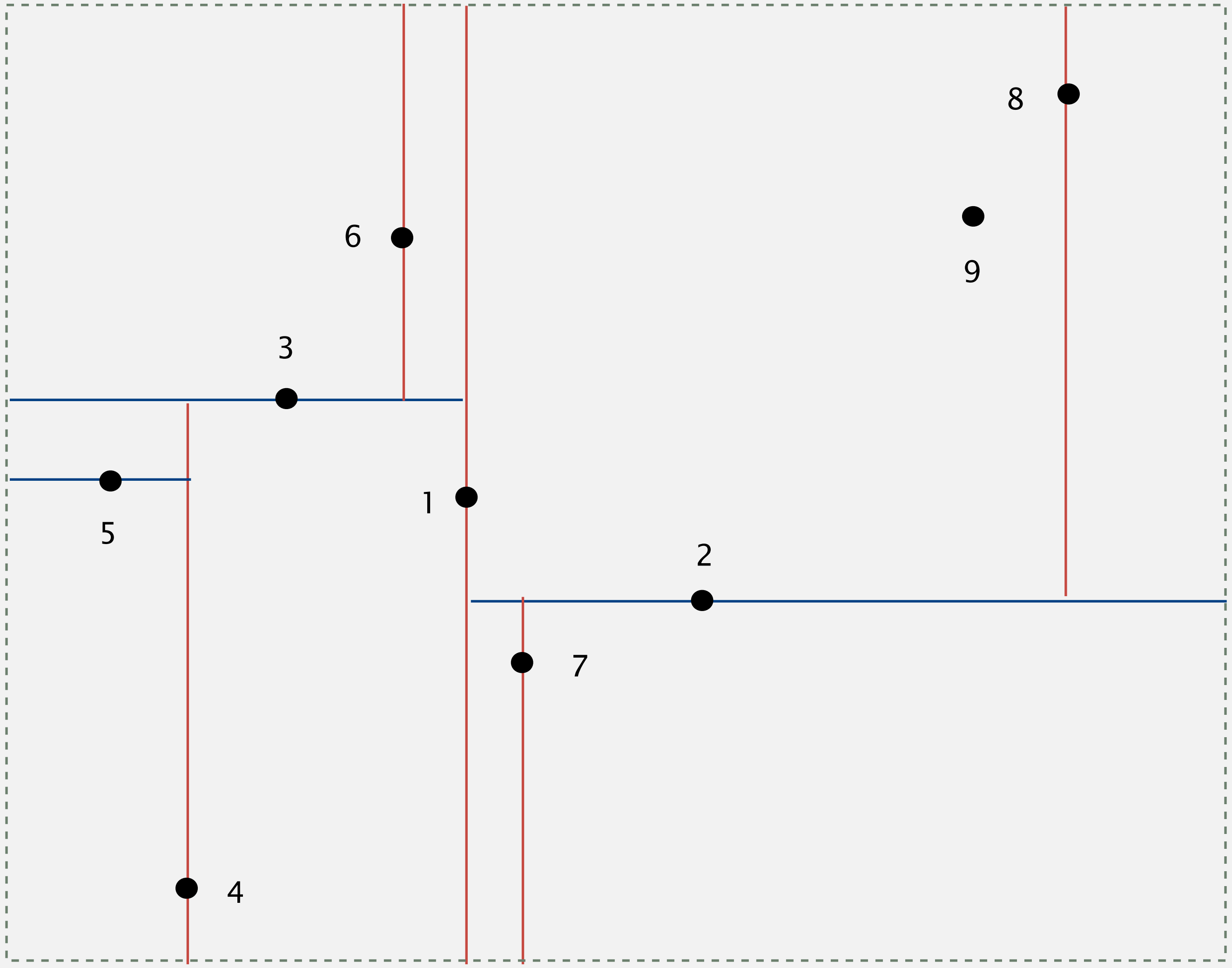
2d tree construction

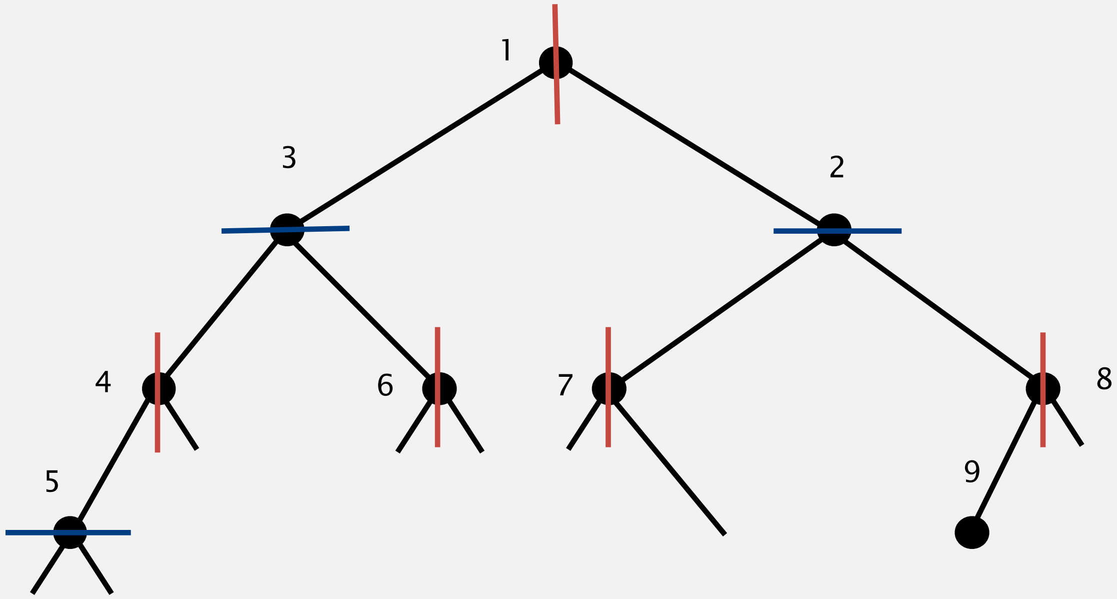
Recursively partition plane into two halfplanes.



2d tree construction

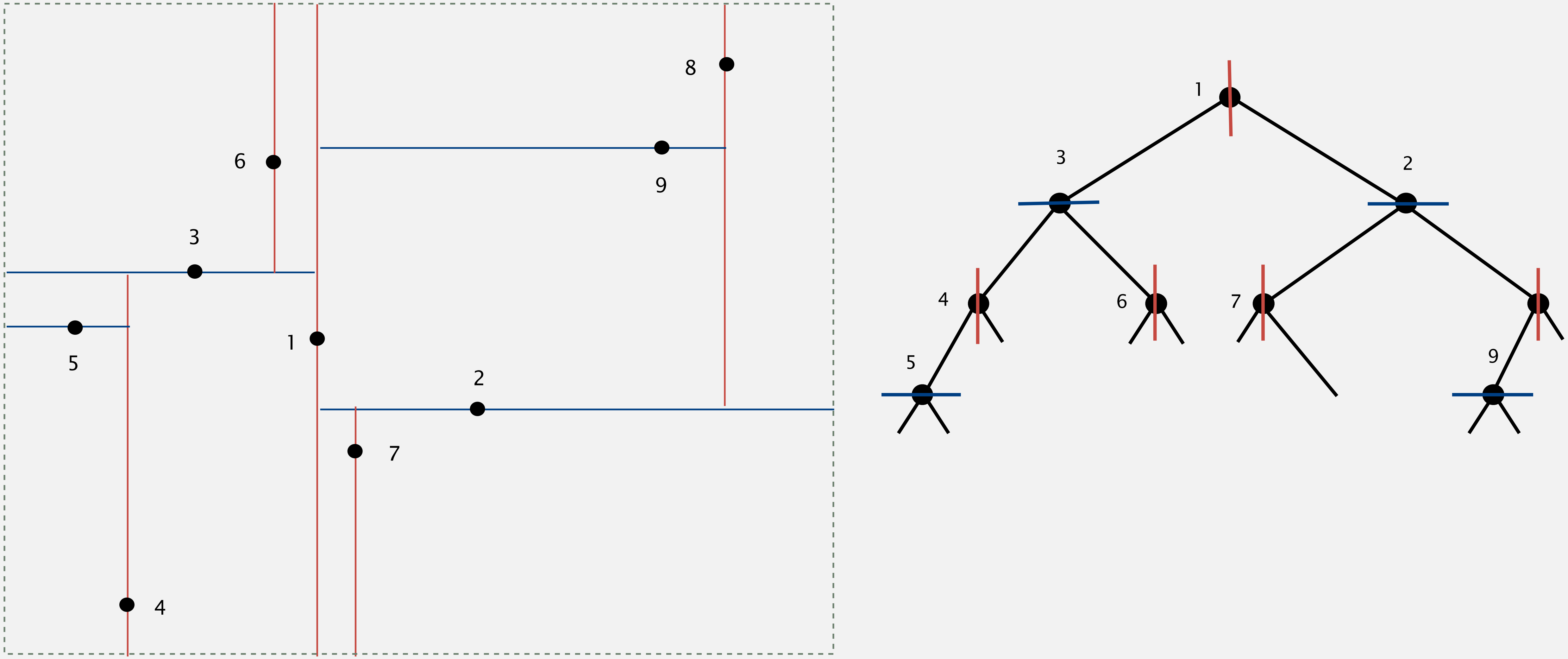
Recursively partition plane into two halfplanes.





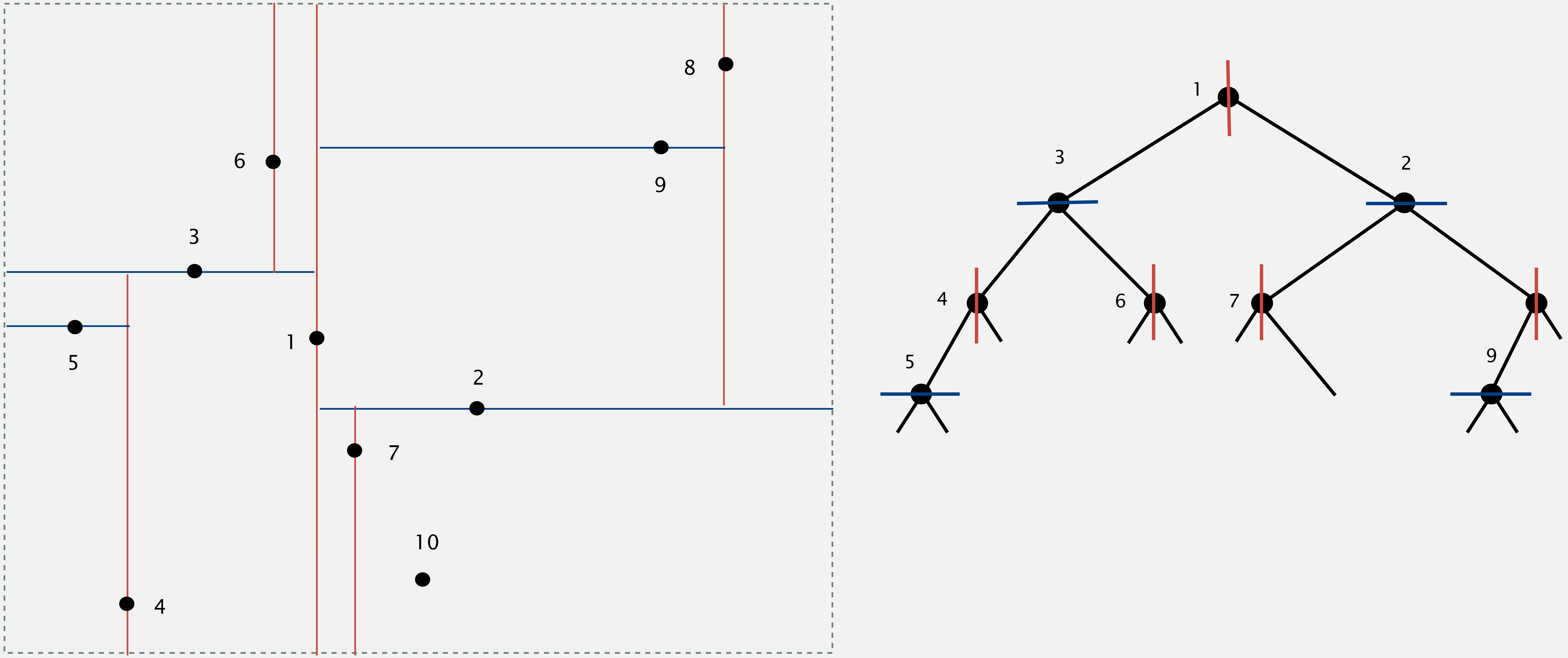
2d tree construction

Recursively partition plane into two halfplanes.



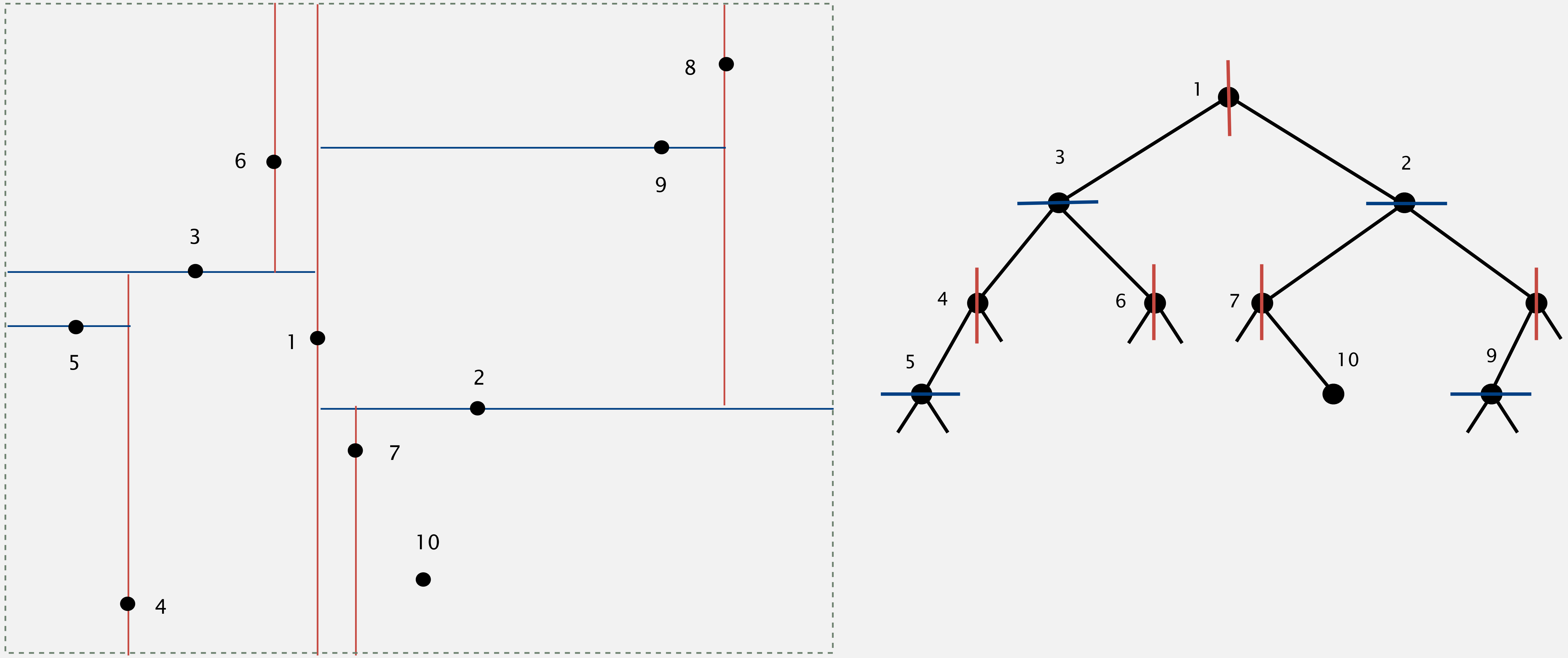
2d tree construction

Recursively partition plane into two halfplanes.



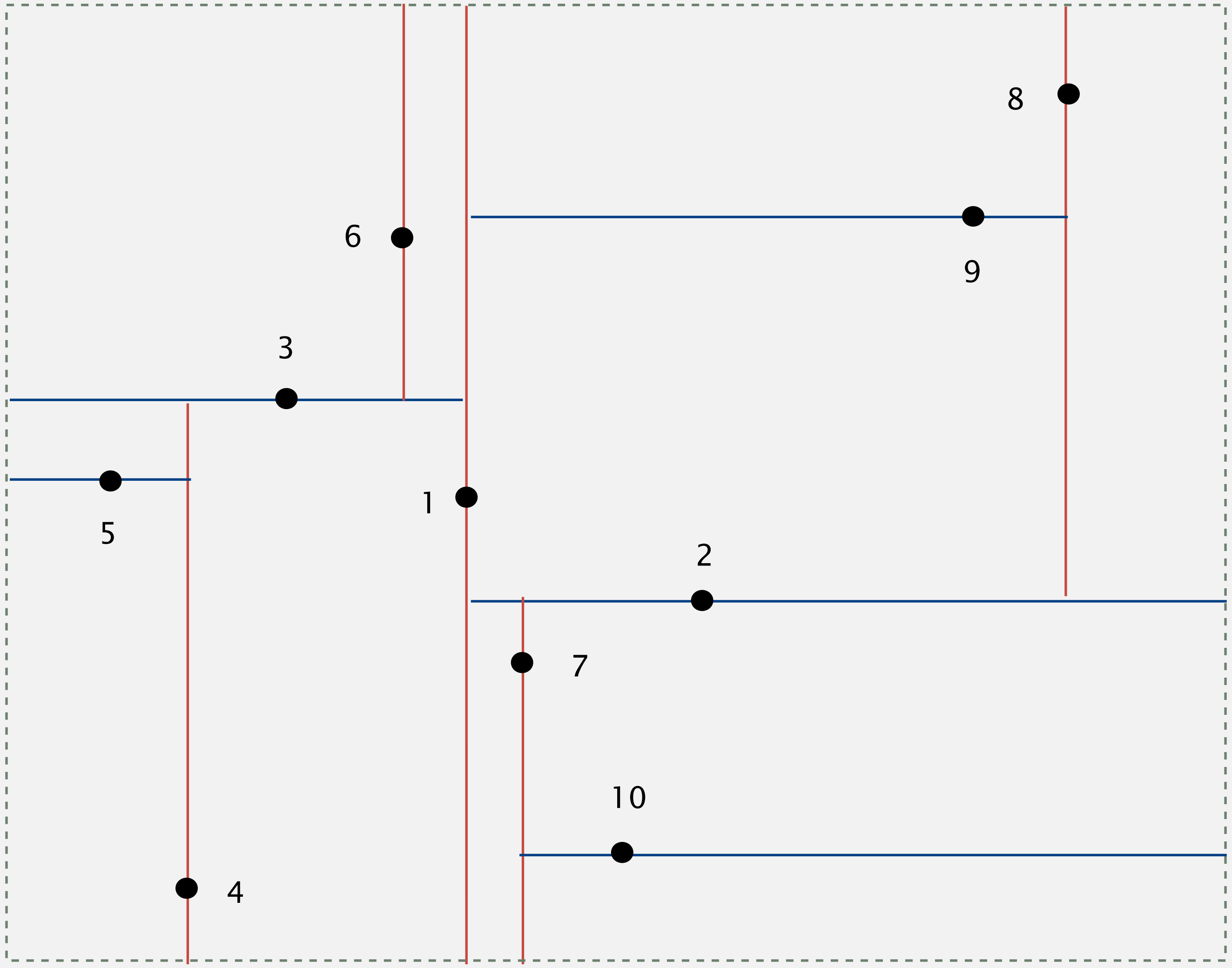
2d tree construction

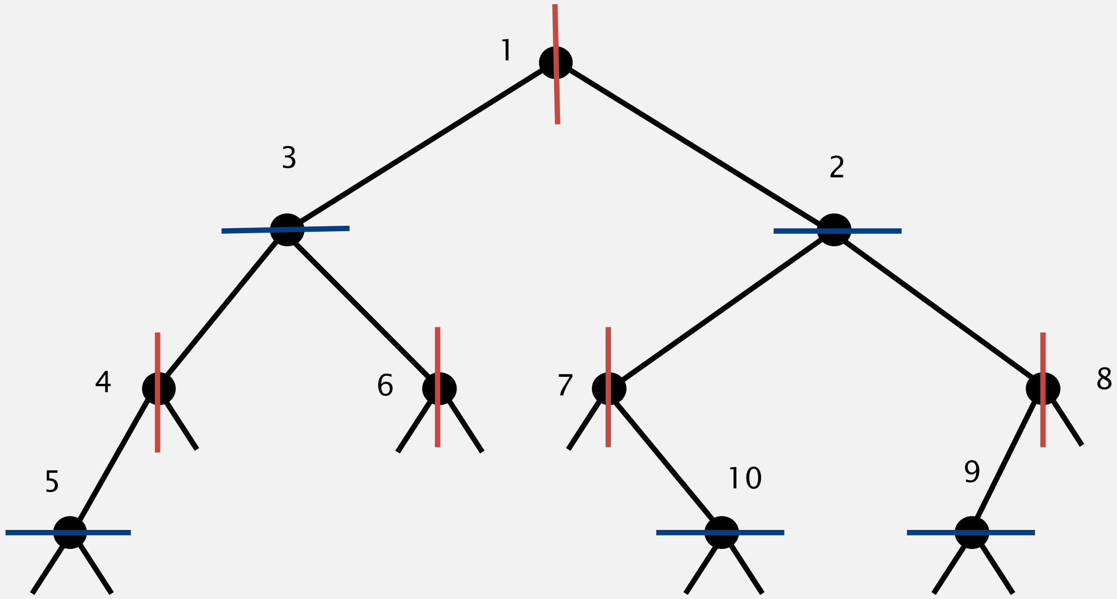
Recursively partition plane into two halfplanes.



2d tree construction

Recursively partition plane into two halfplanes.

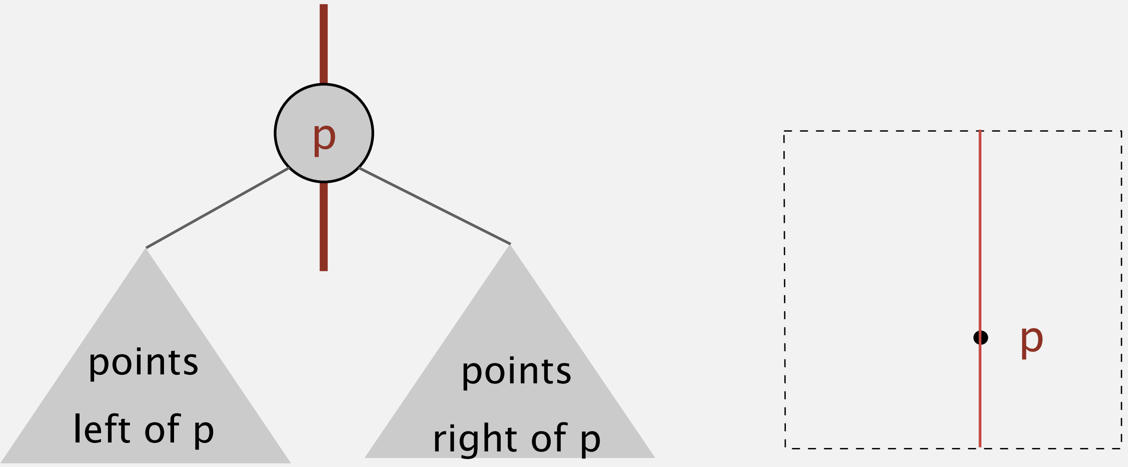




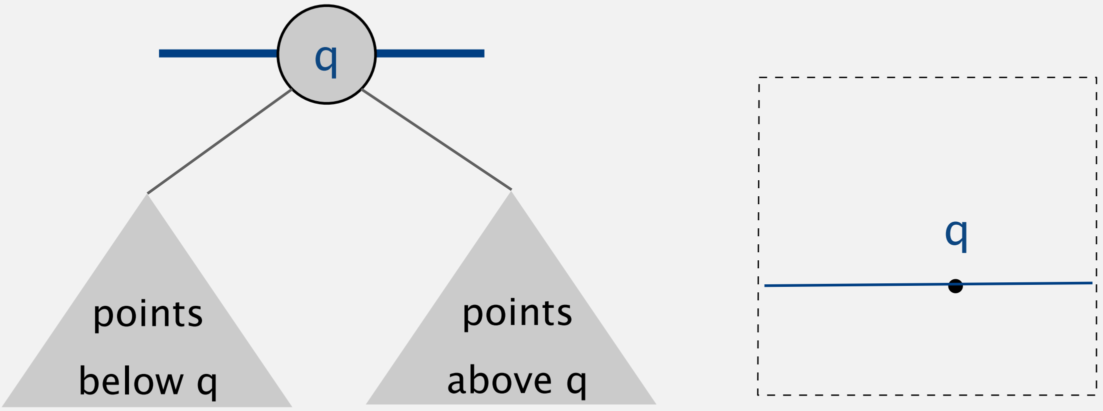
2d tree implementation

Data structure. BST, but alternate using *x*- and *y*-coordinates as key.

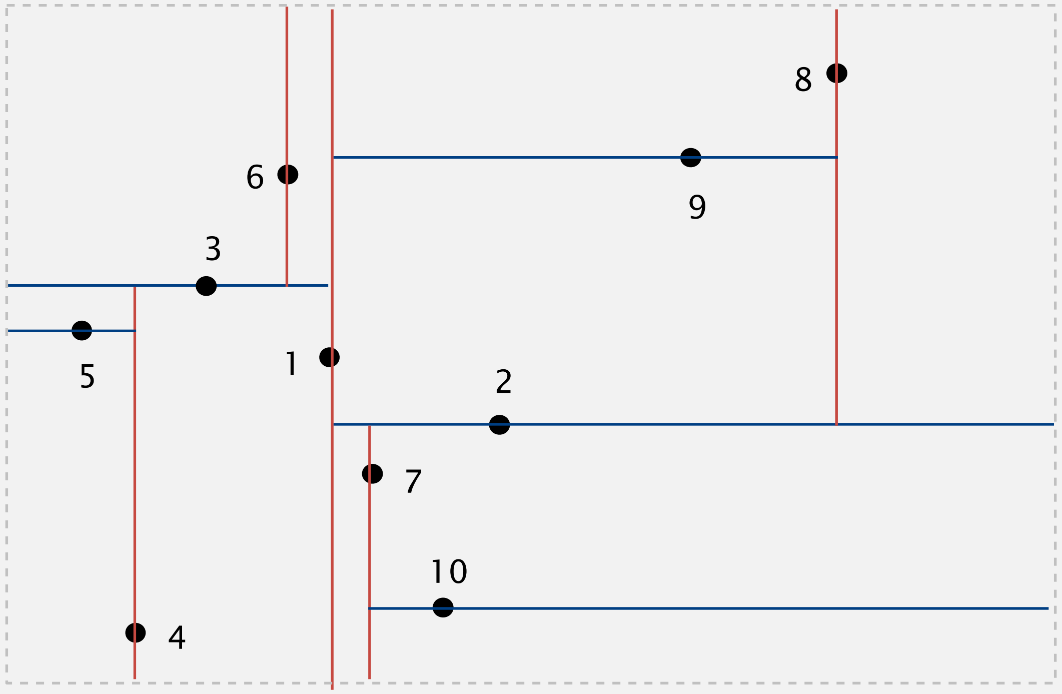
- Search gives rectangle containing point.
- Insert further subdivides the plane.

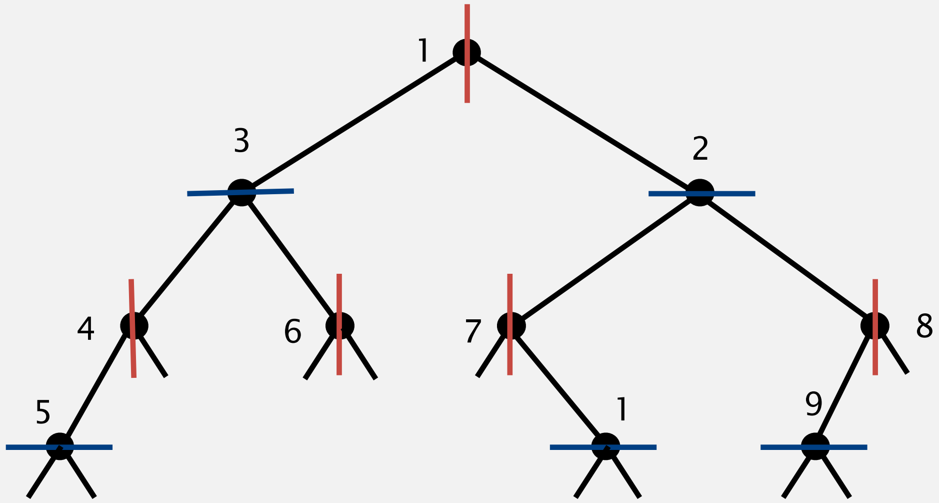


even levels



odd levels

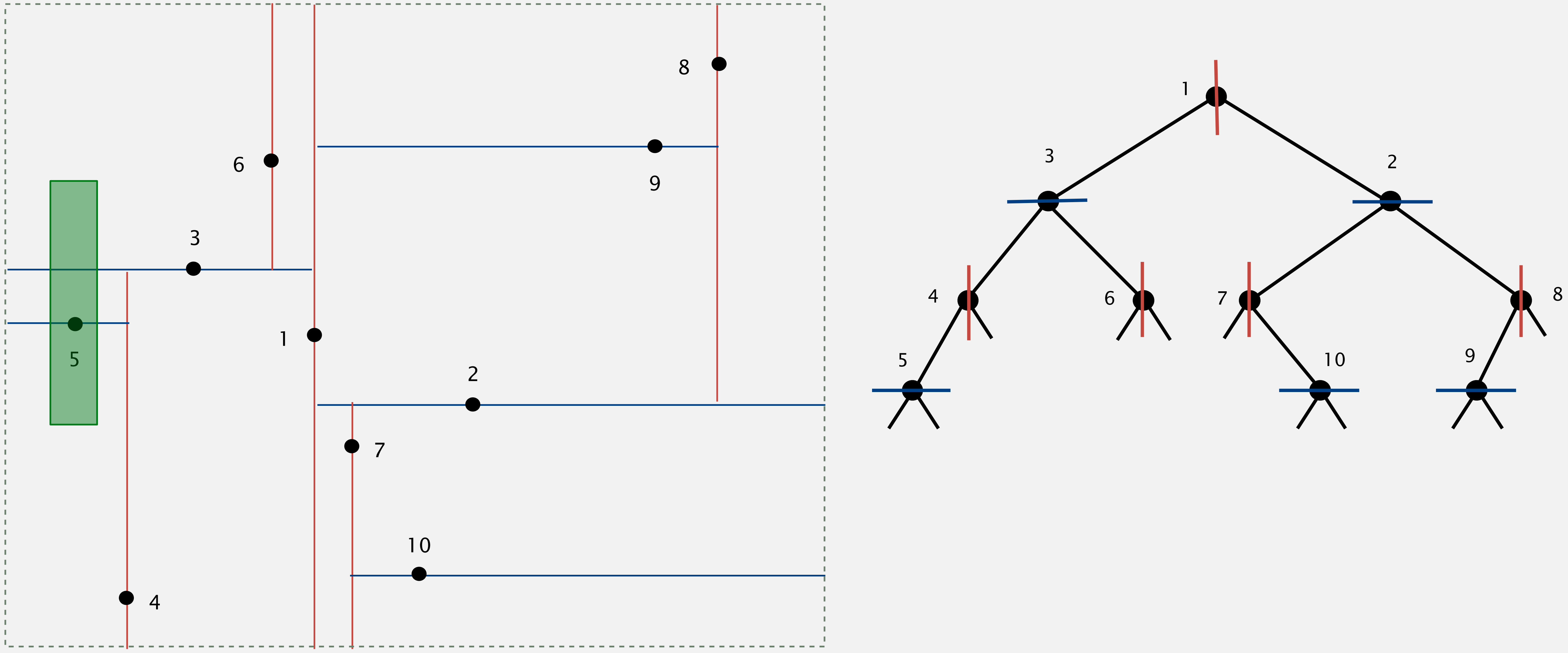




2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

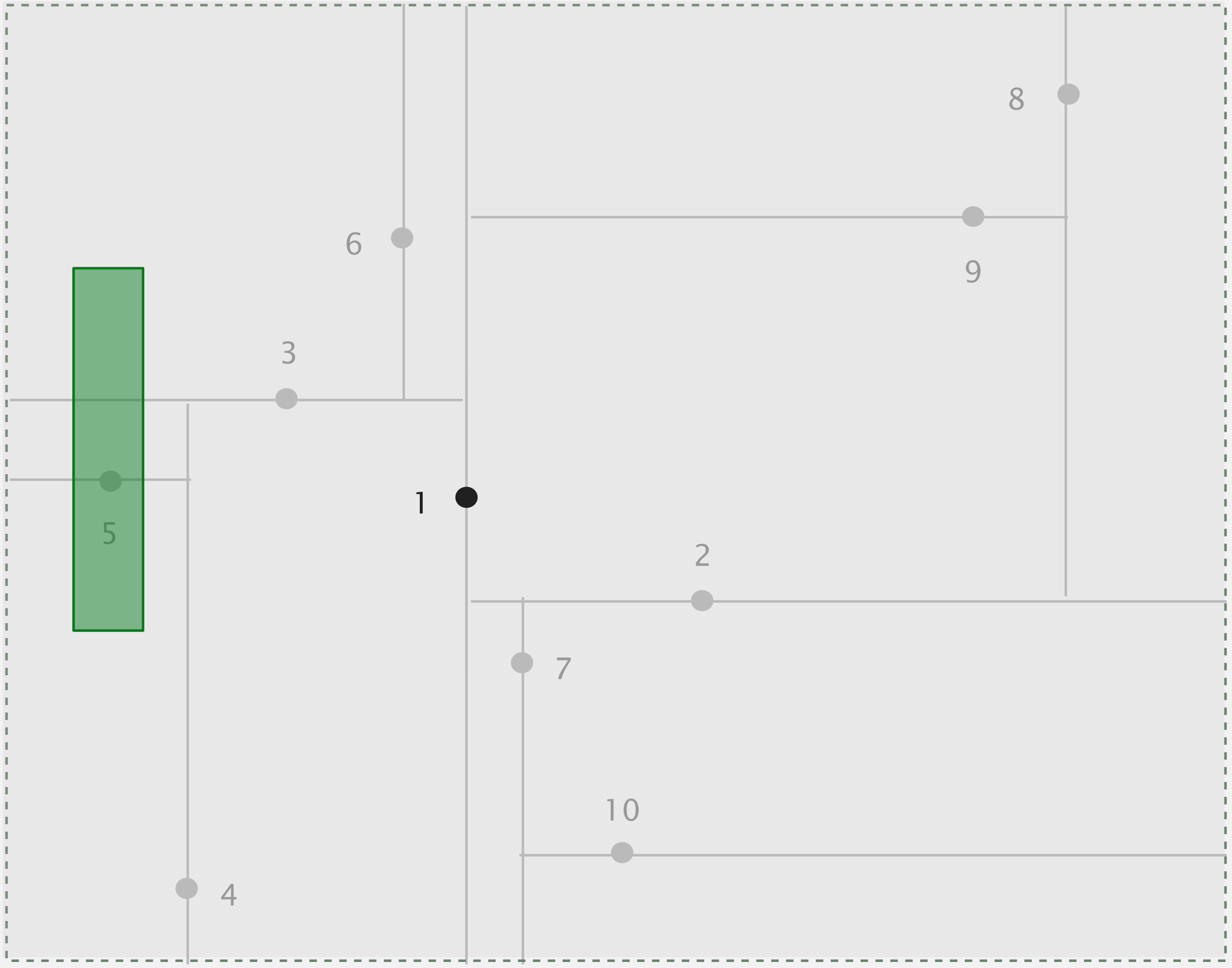
- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

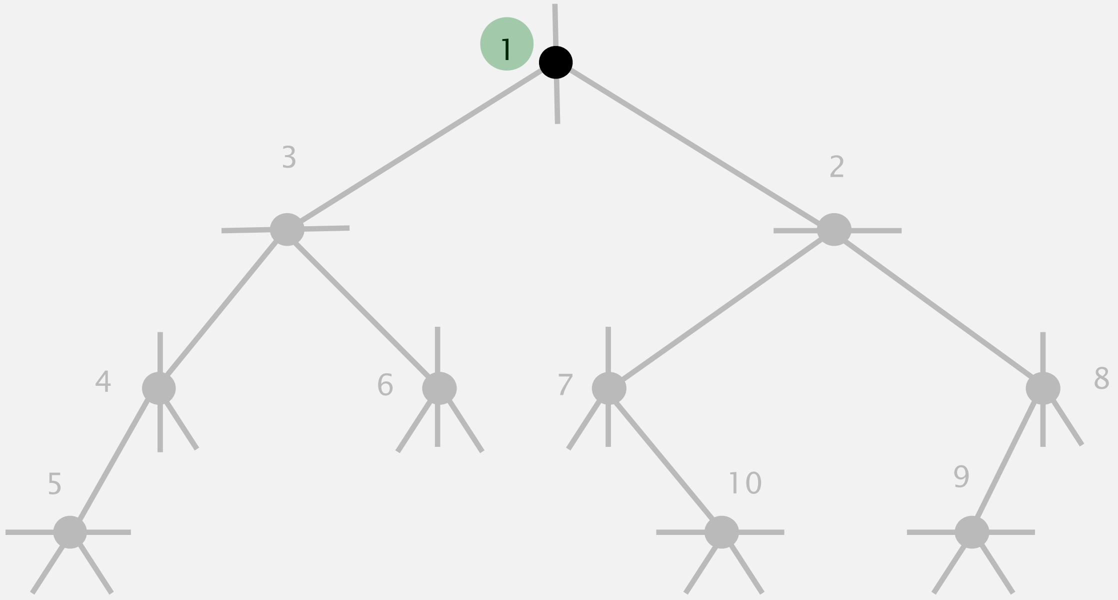


2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).





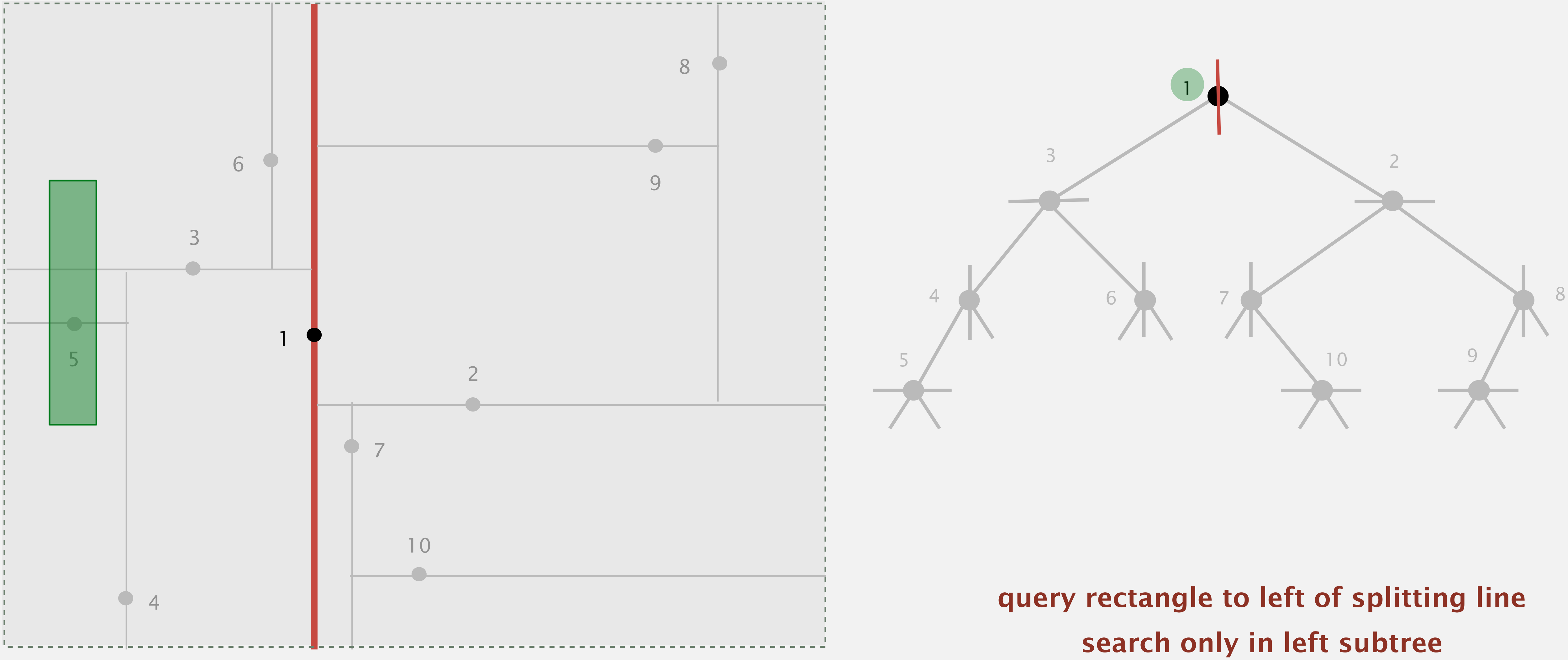
search root node

check if query rectangle contains point 1

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

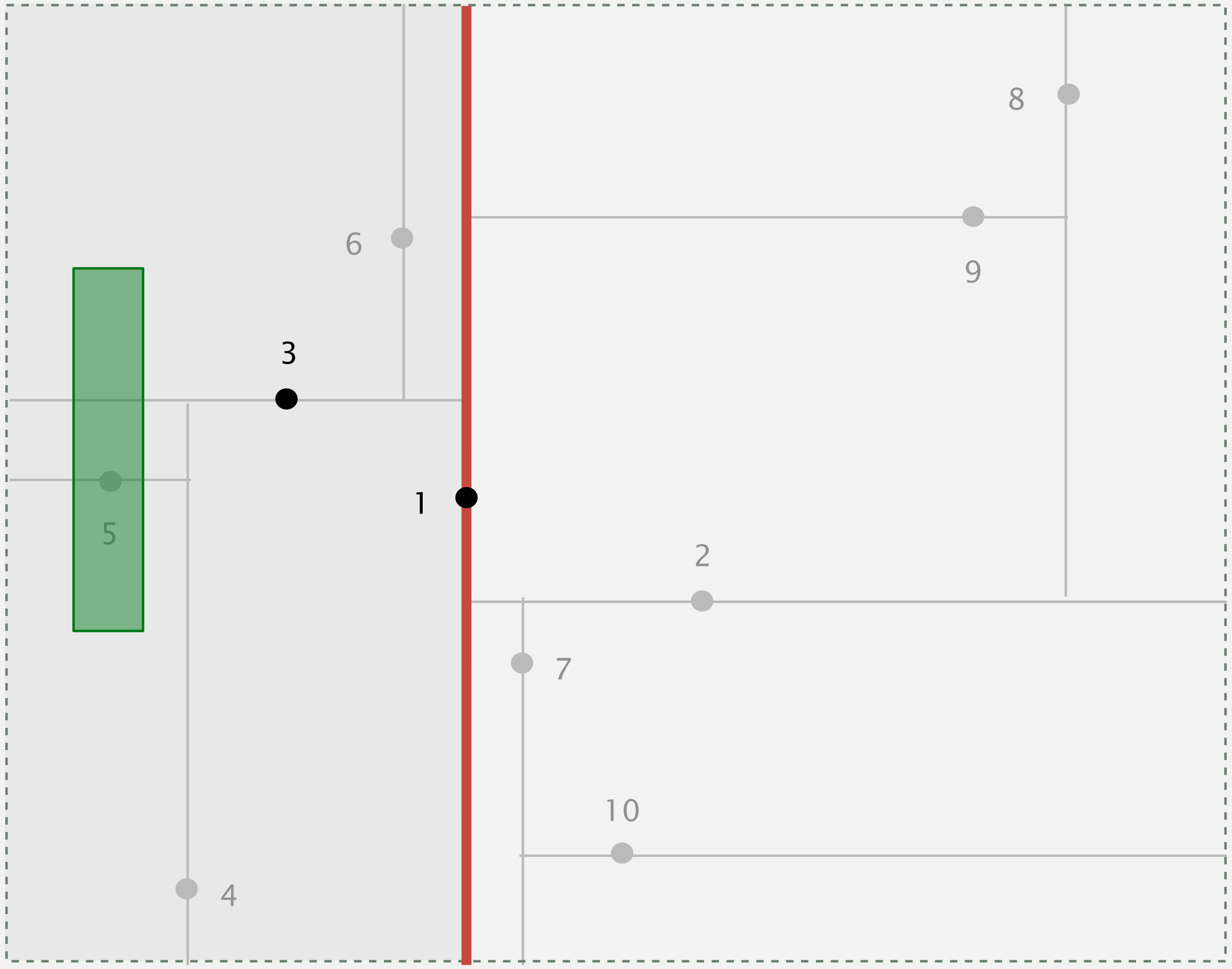


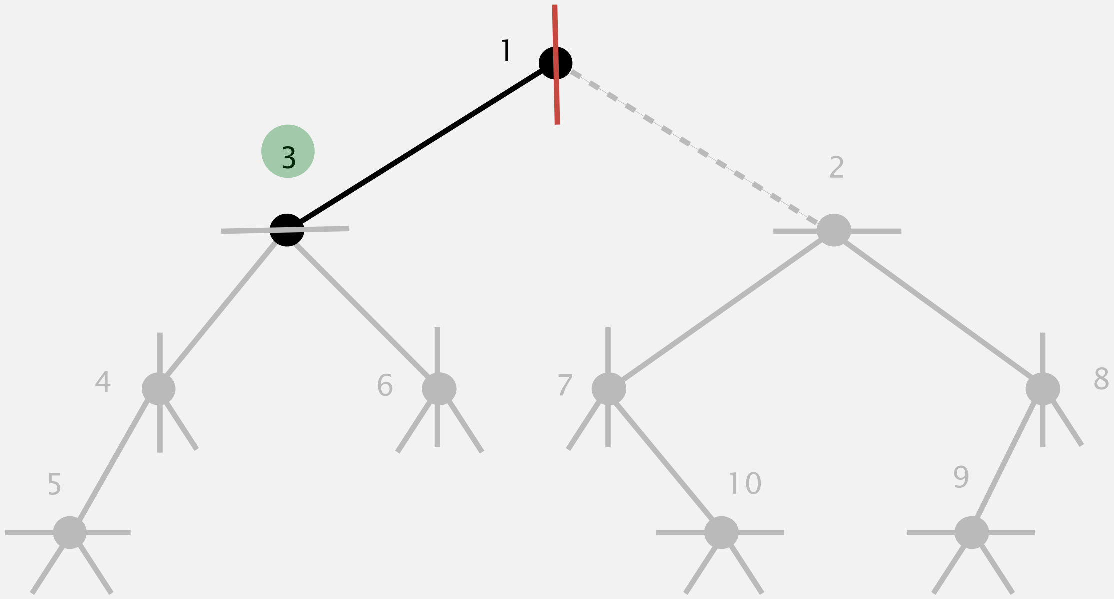
query rectangle to left of splitting line
search only in left subtree

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).





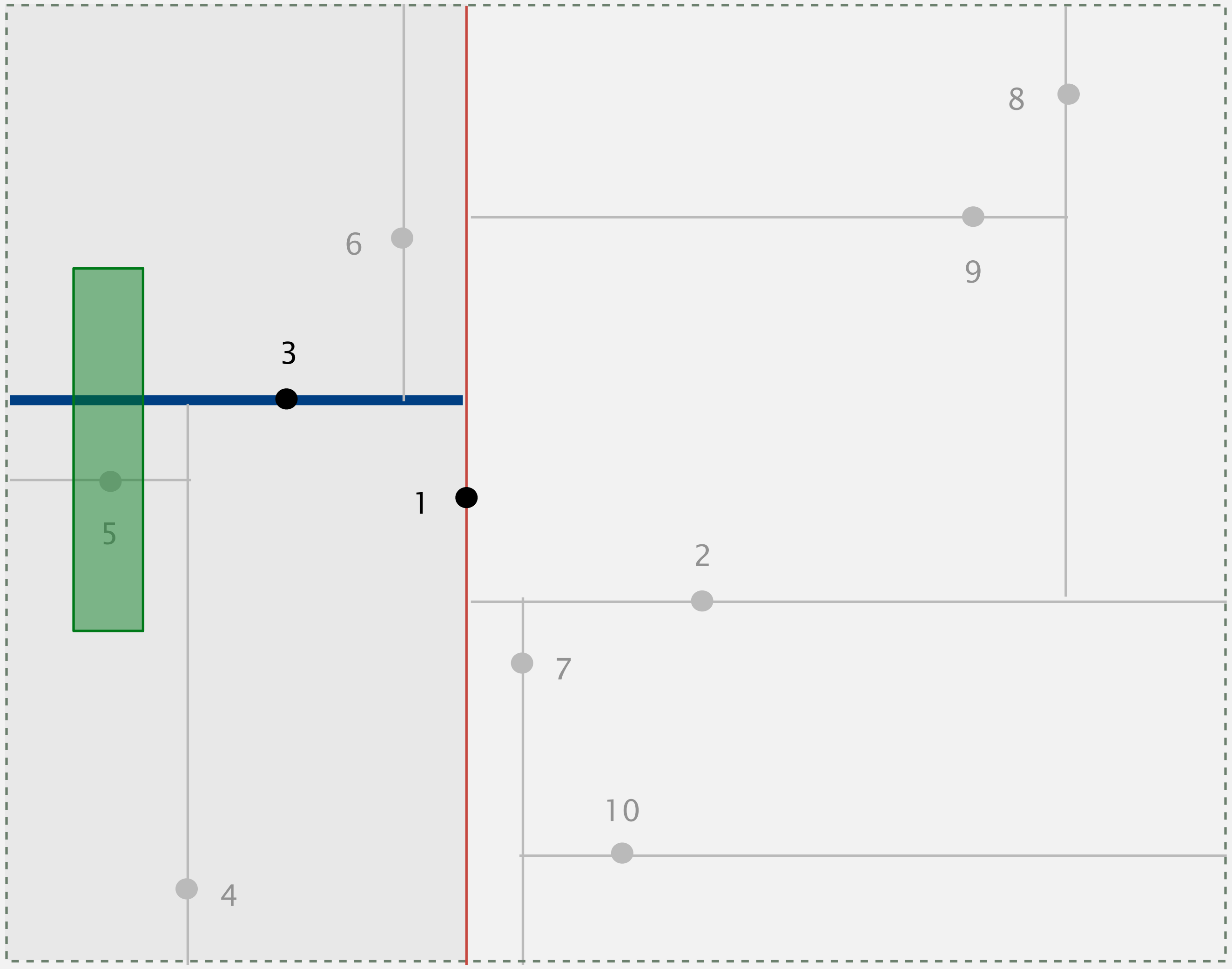
search left subtree

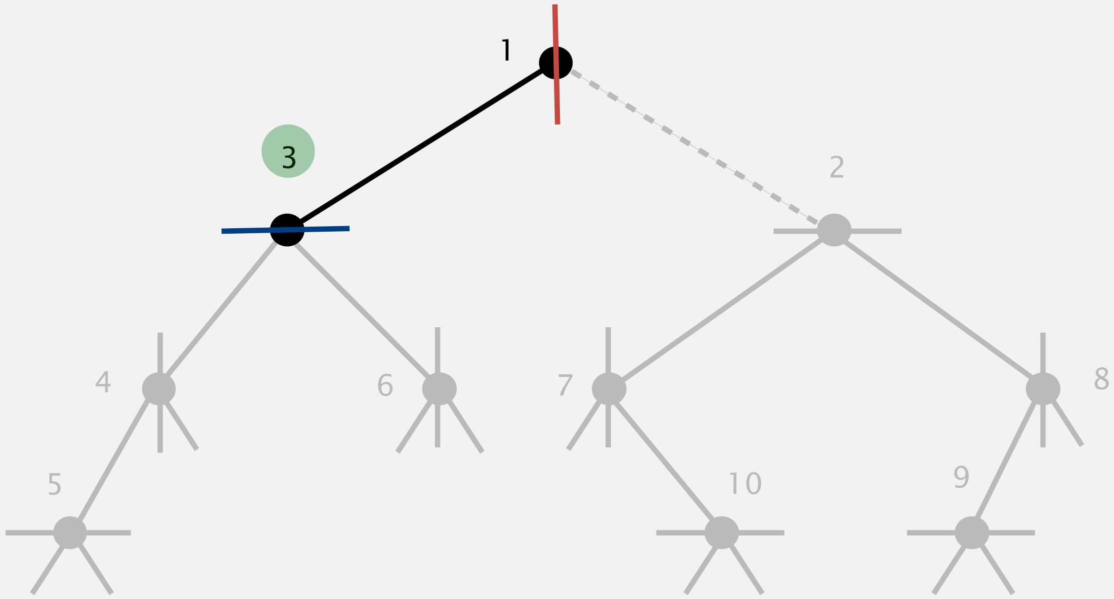
check if query rectangle contains point 3

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



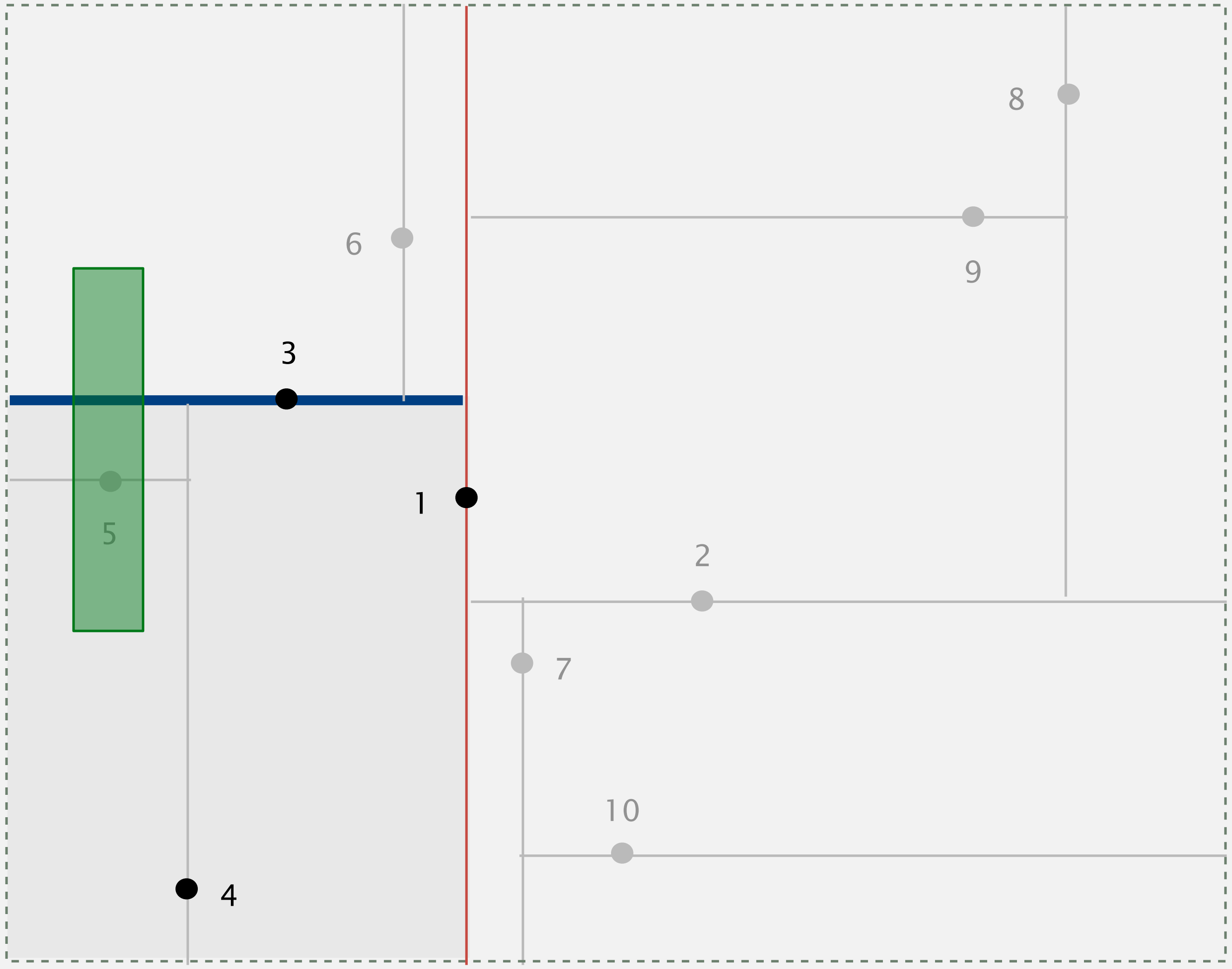


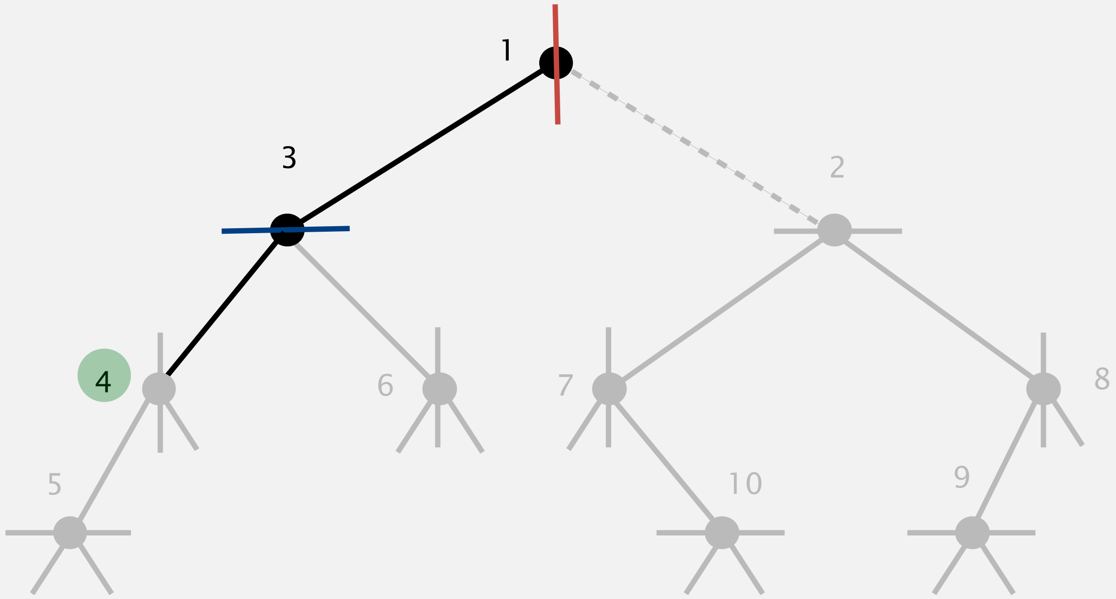
query rectangle intersects splitting line
search bottom and top subtrees

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).





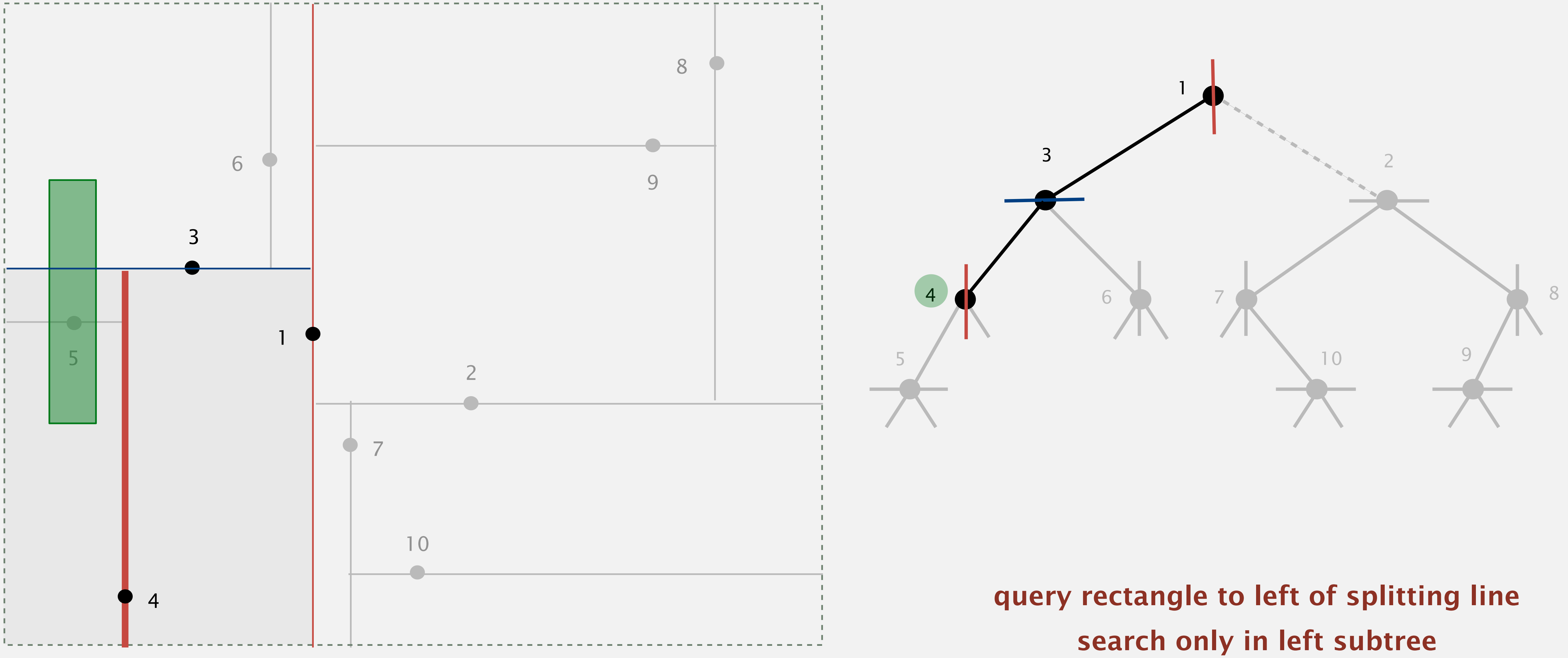
search left subtree

check if query rectangle contains point 4

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

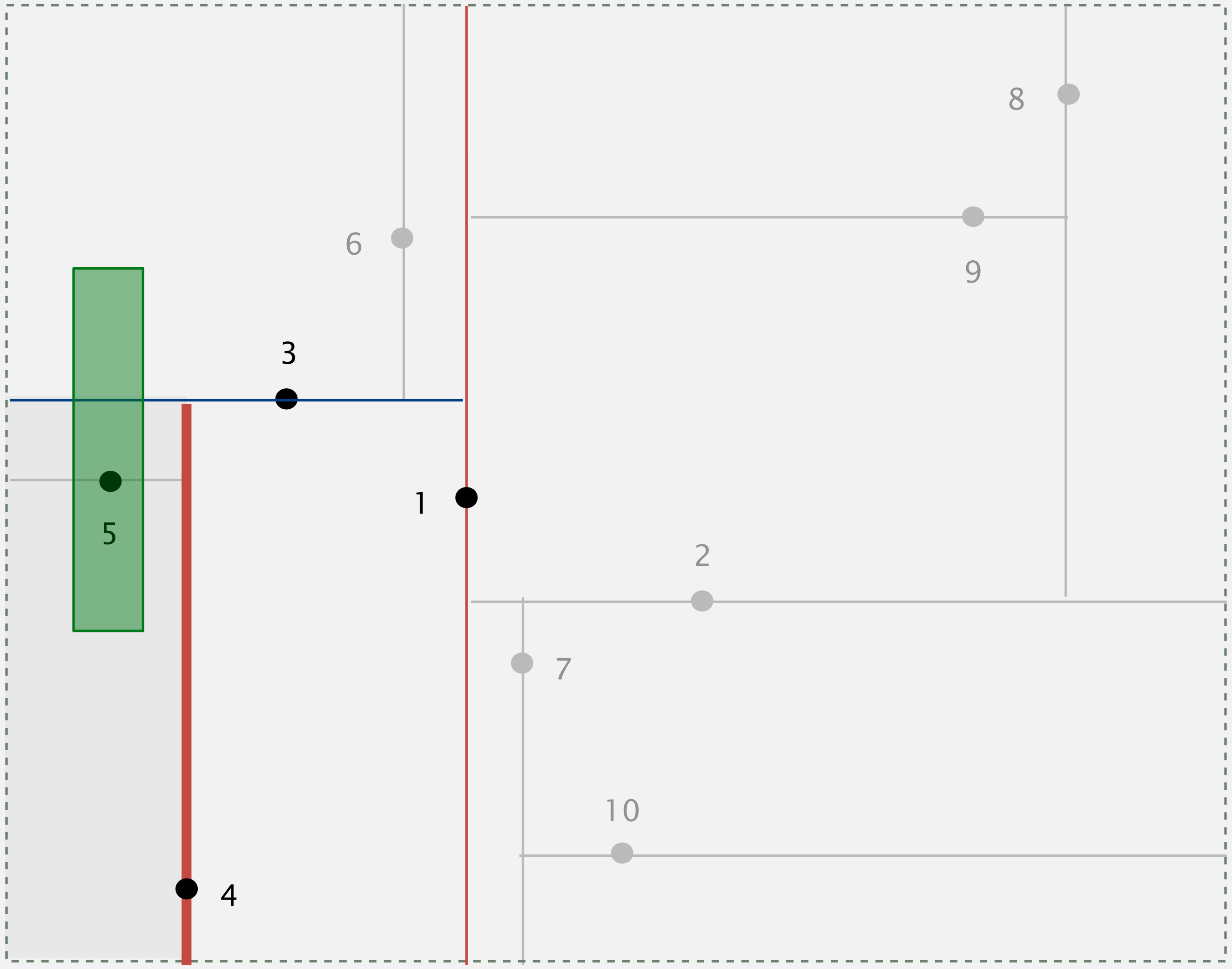


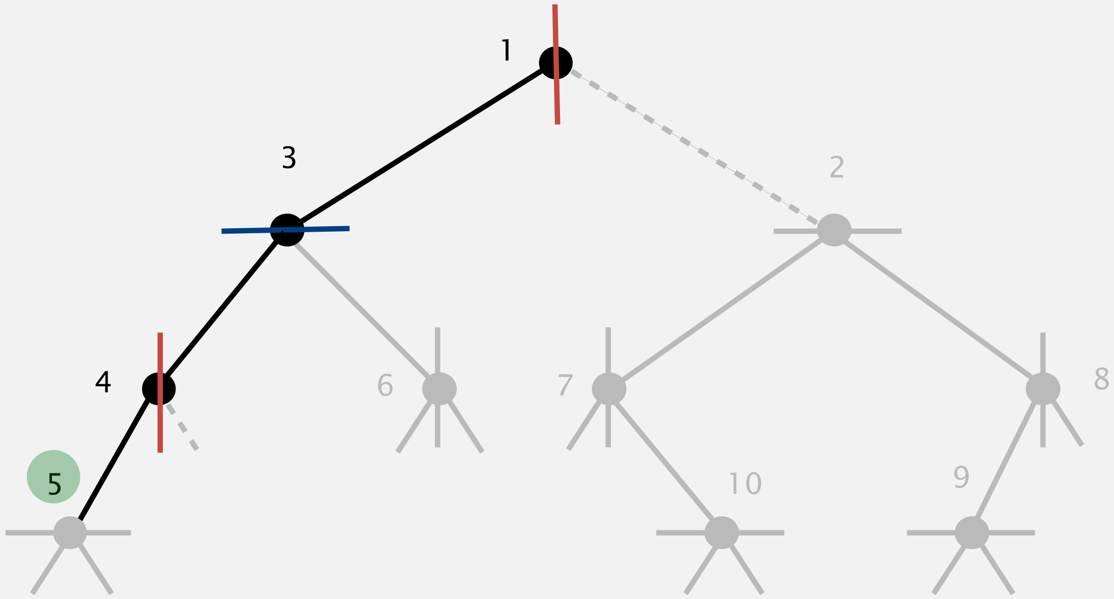
query rectangle to left of splitting line
search only in left subtree

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).





search left subtree

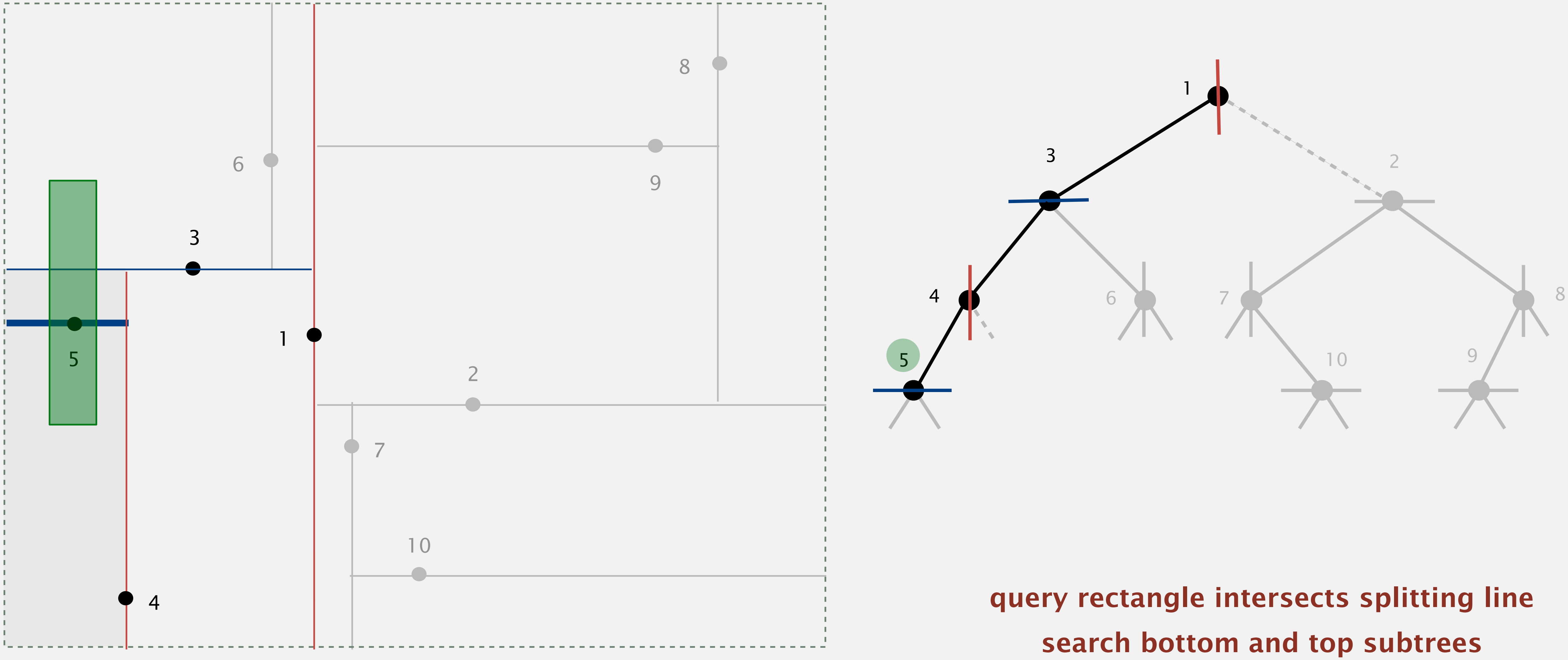
check if query rectangle contains point 5

(search hit)

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

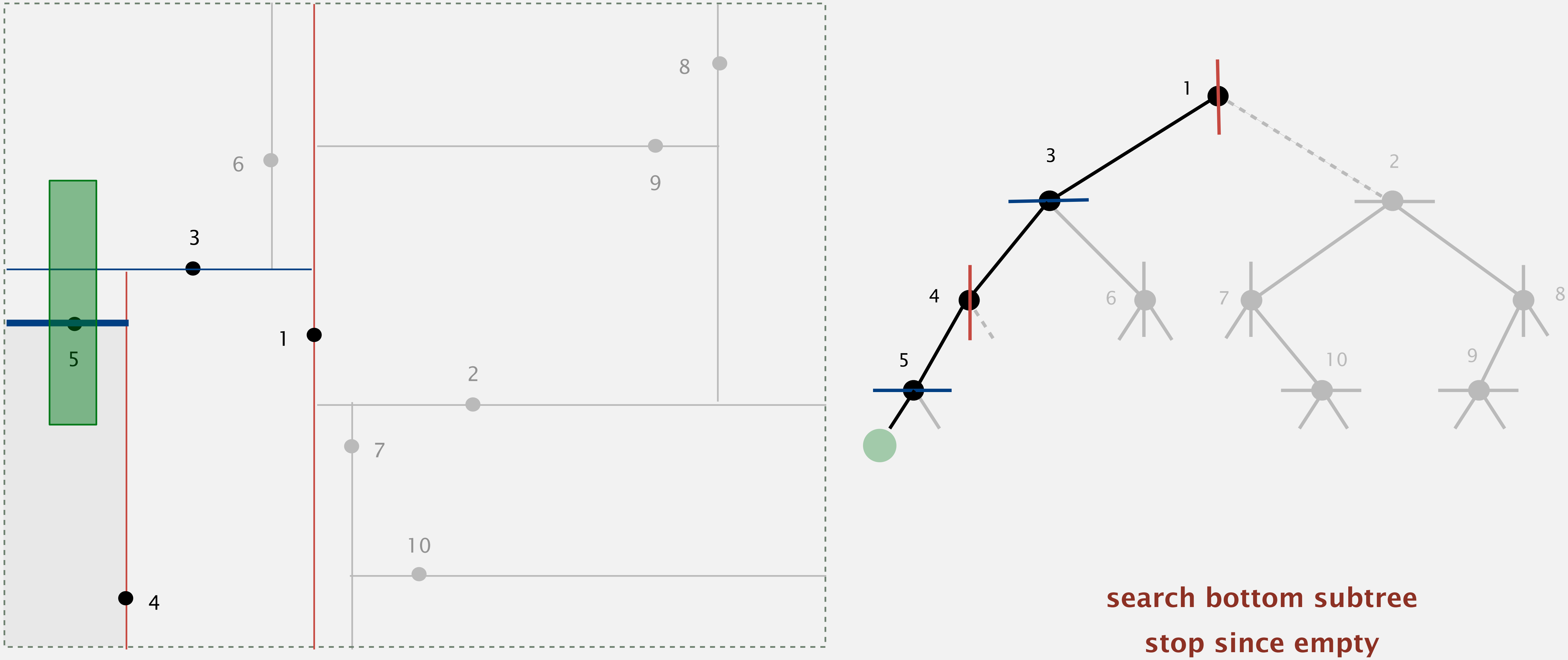


query rectangle intersects splitting line
search bottom and top subtrees

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

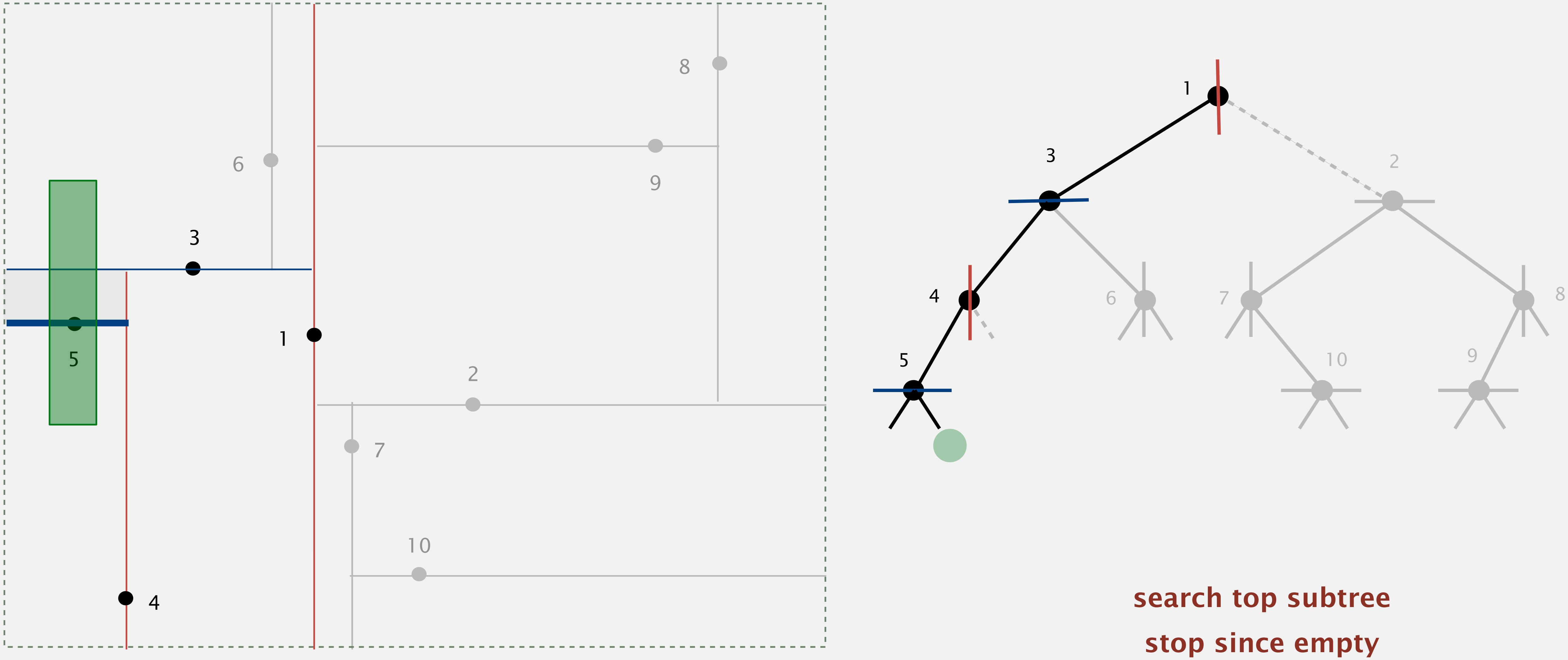


search bottom subtree
stop since empty

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

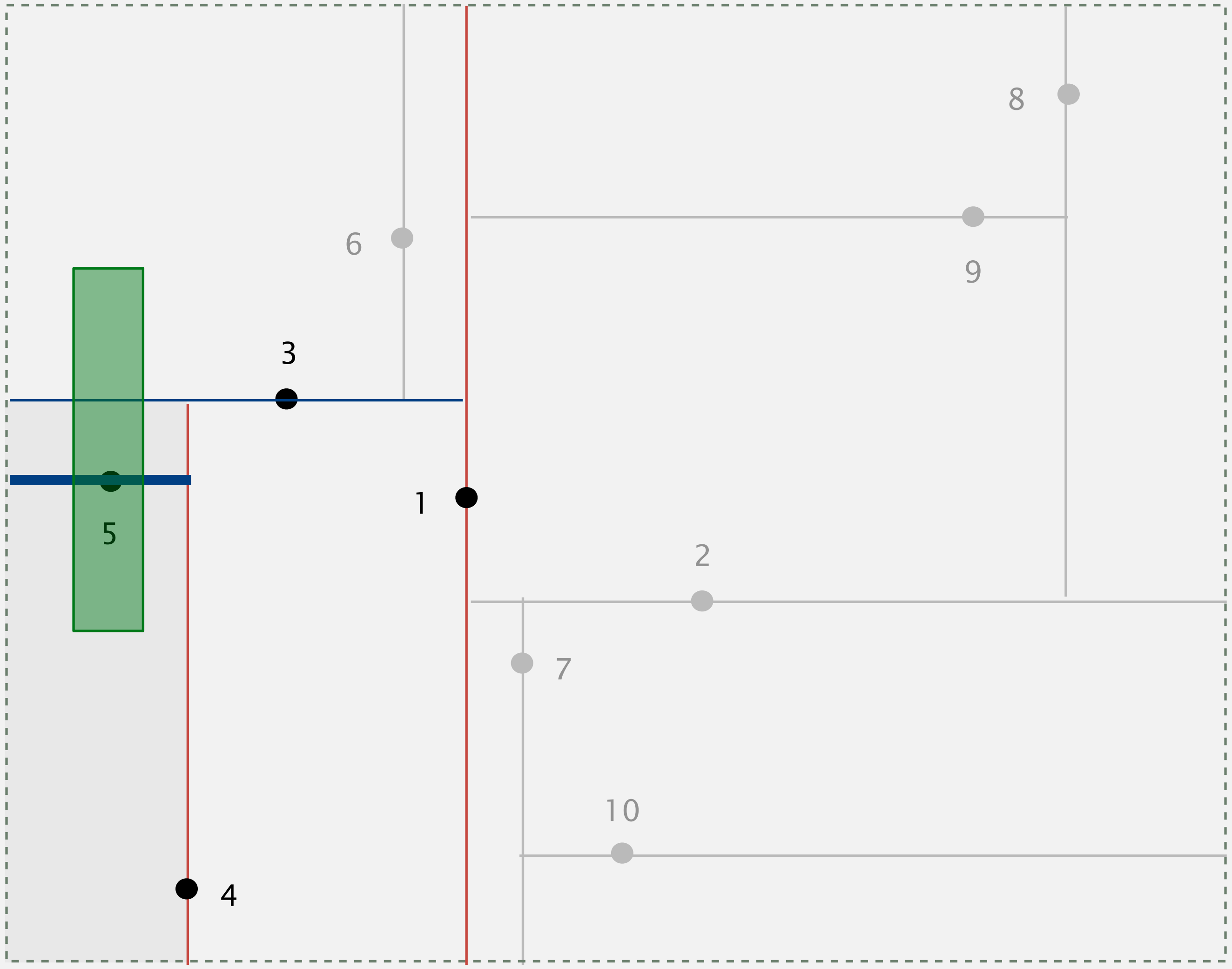


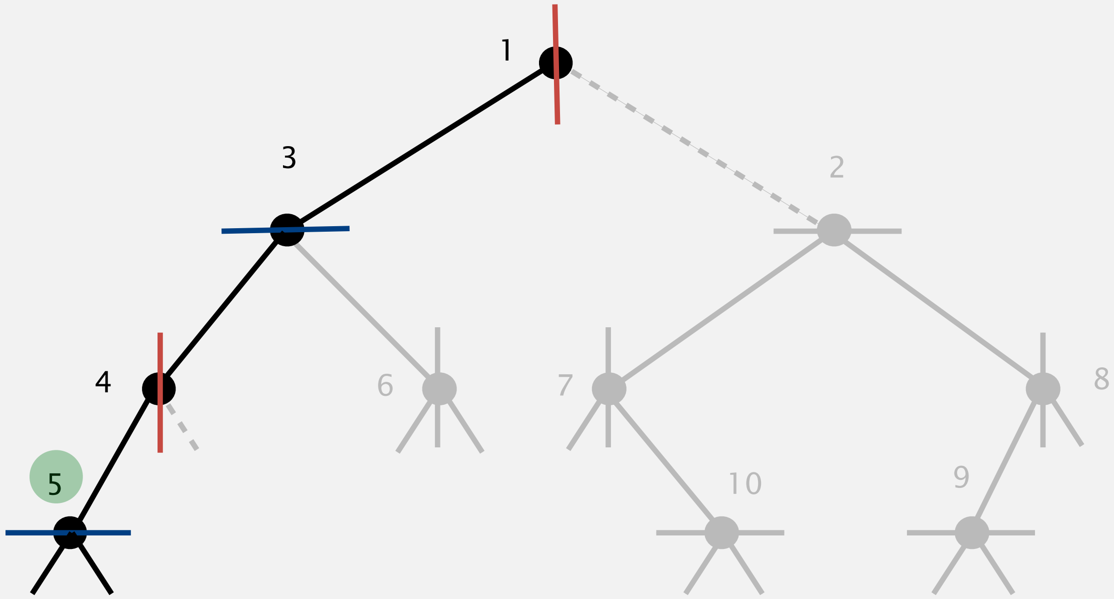
search top subtree
stop since empty

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



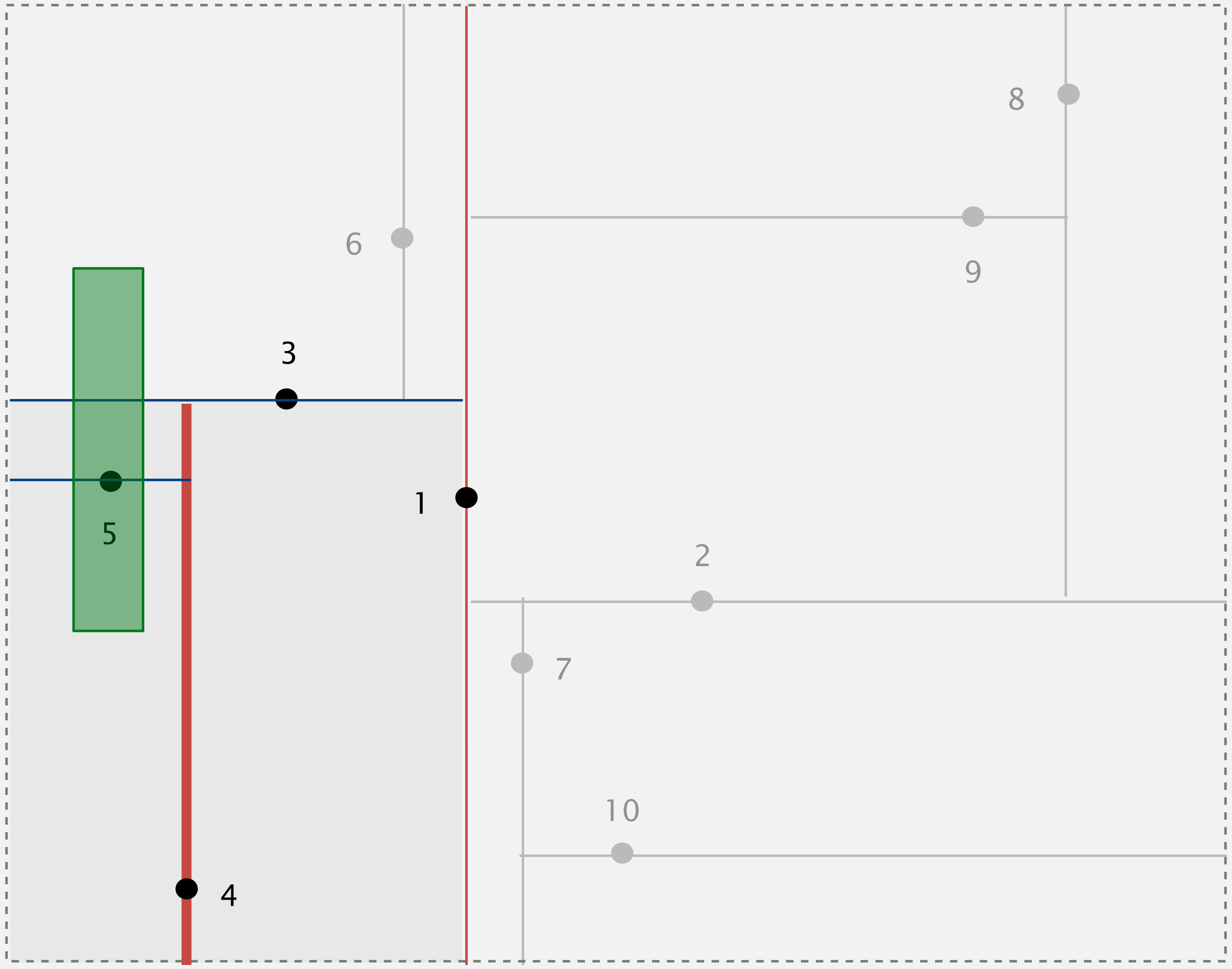


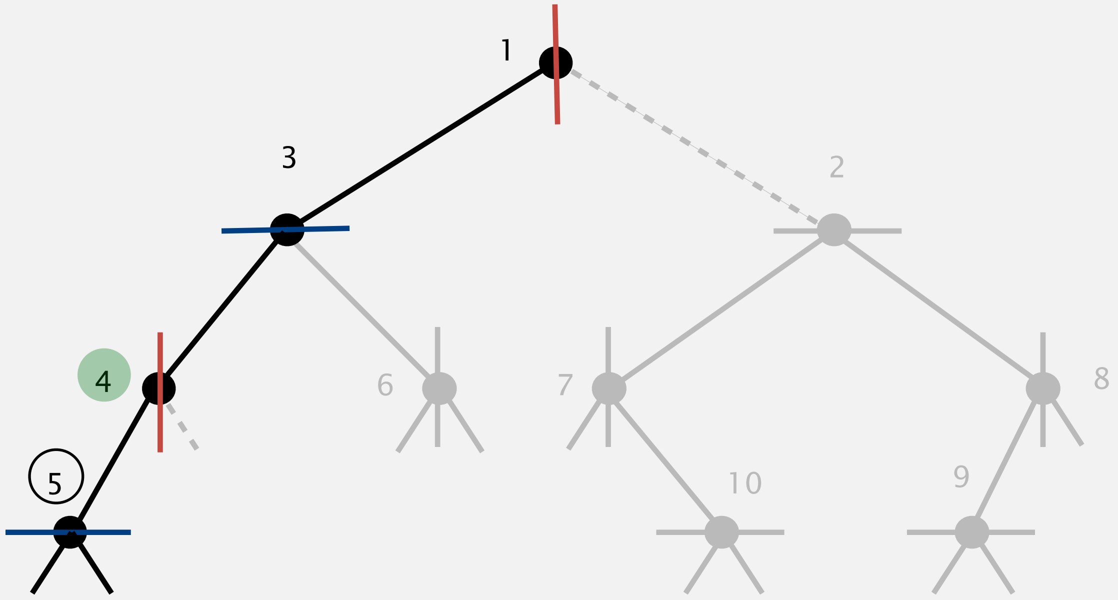
return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



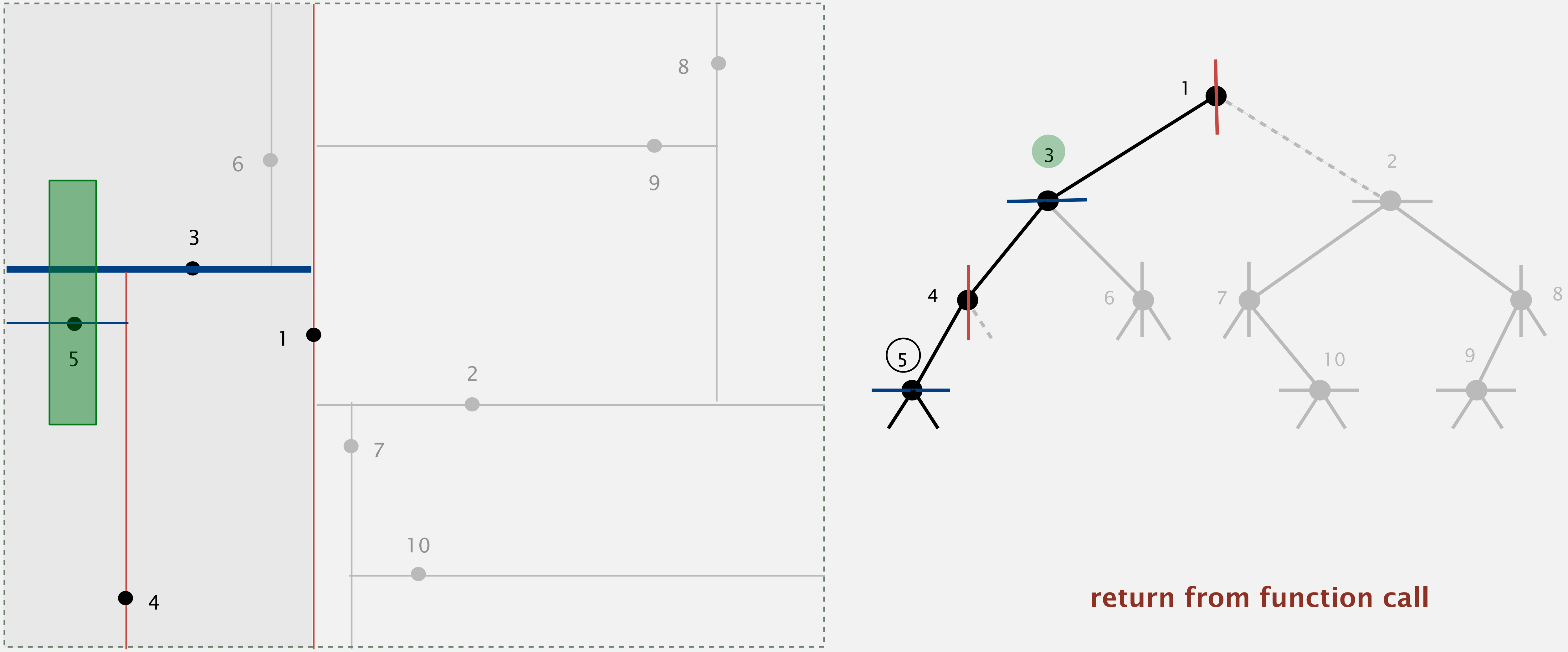


return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

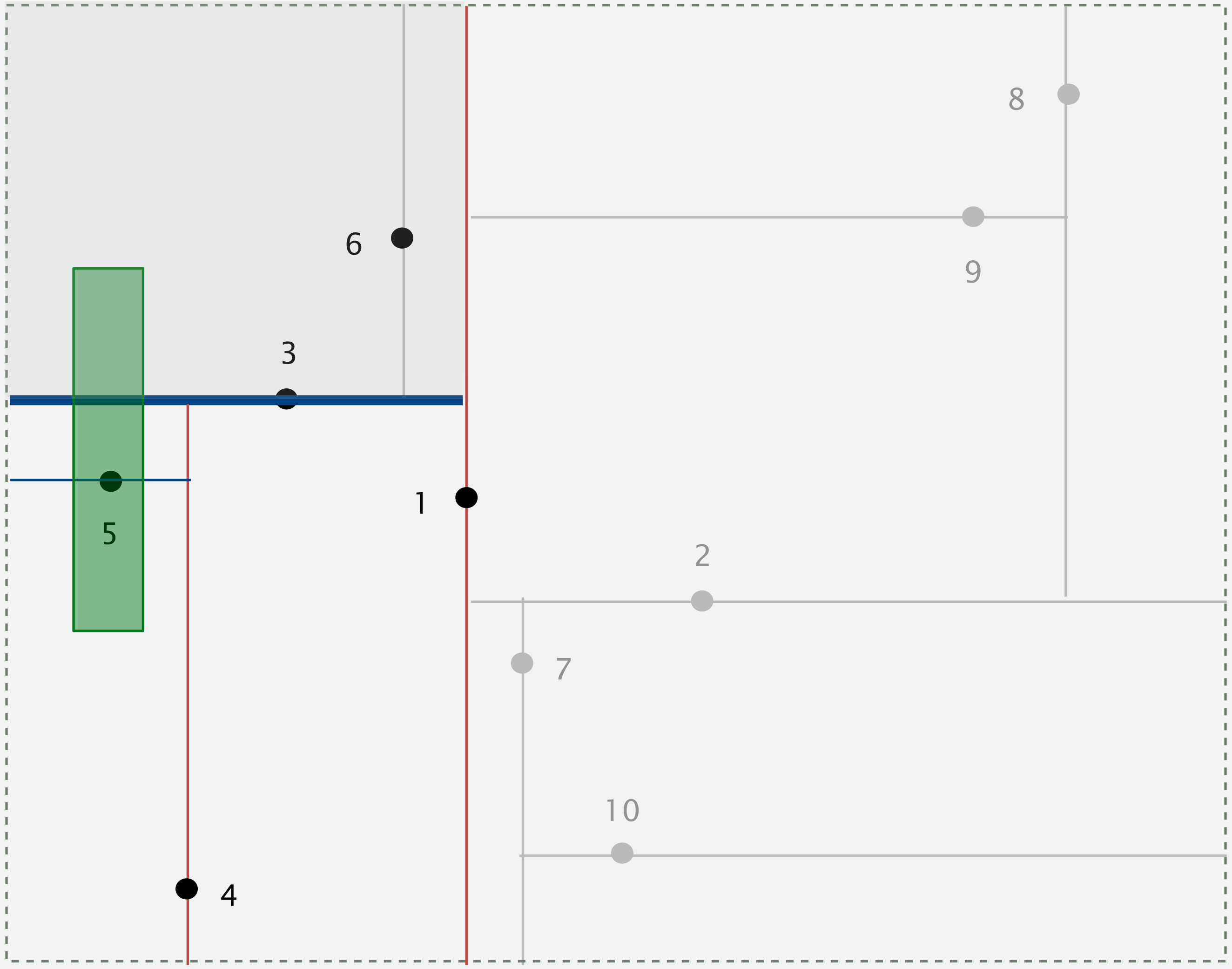


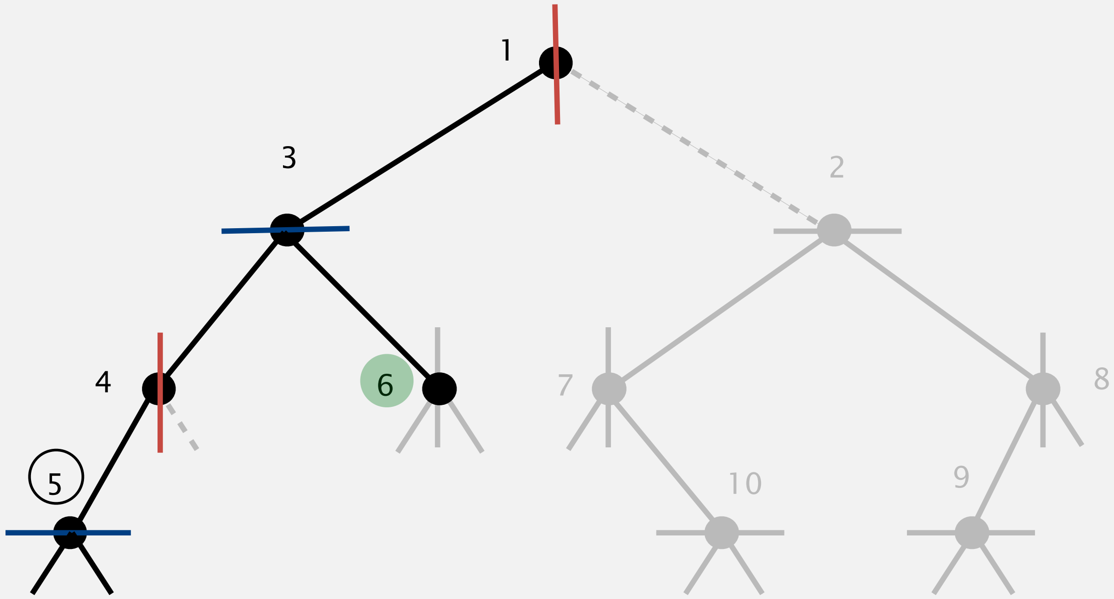
return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).





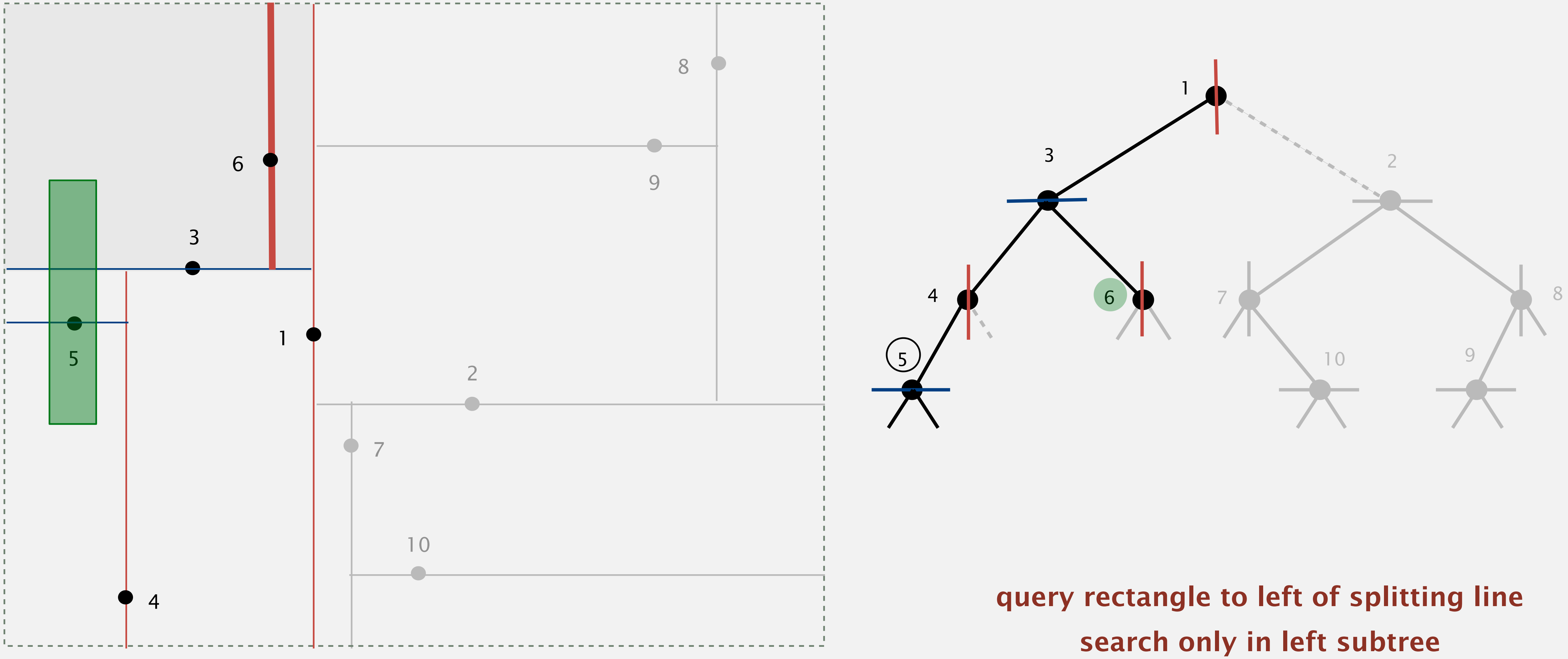
search top subtree

check if query rectangle contains point 6

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

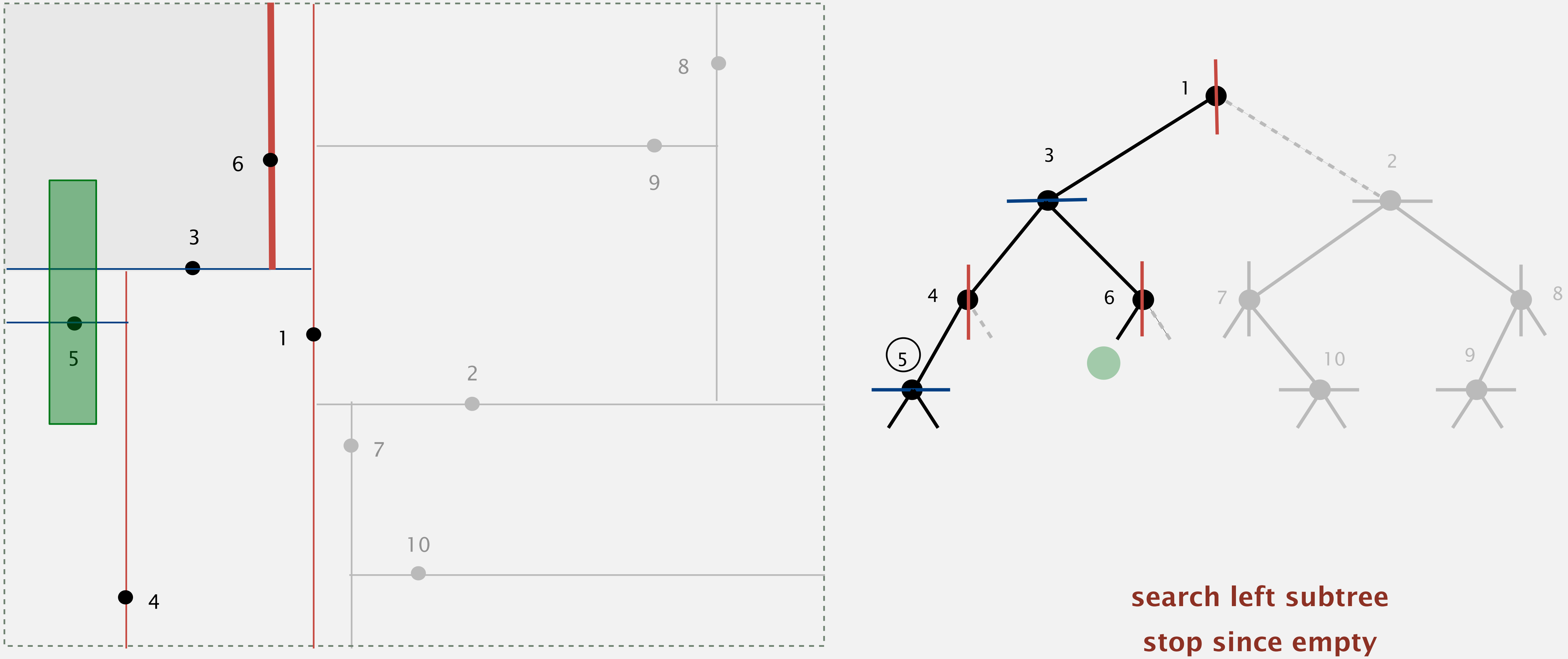


query rectangle to left of splitting line
search only in left subtree

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

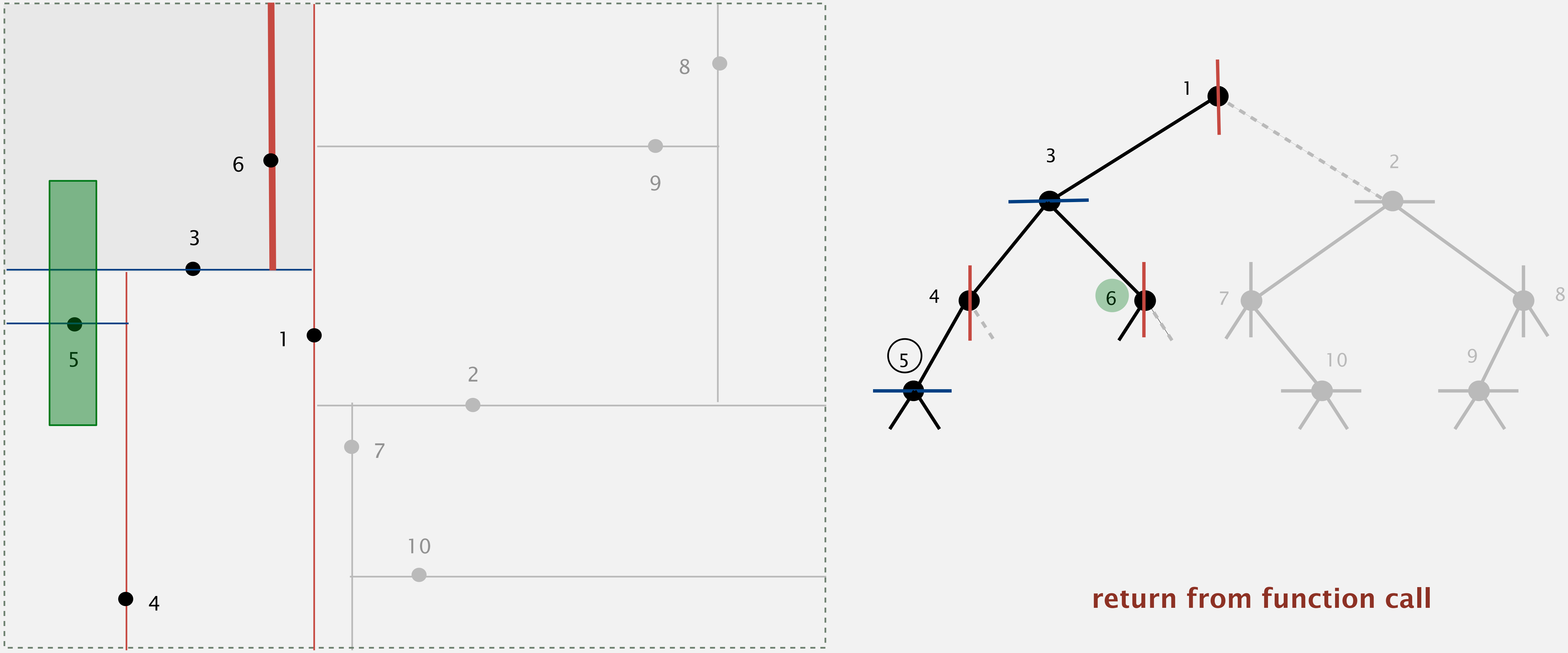


search left subtree
stop since empty

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

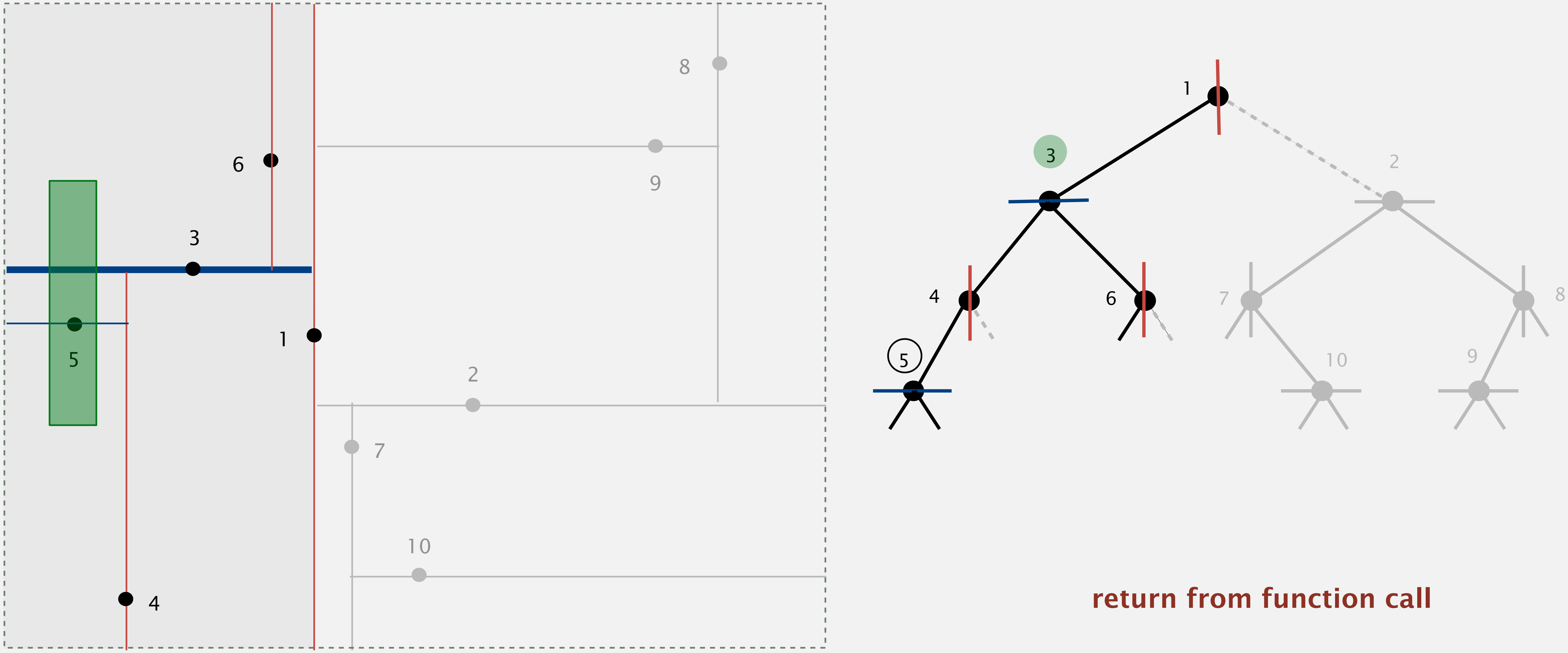


return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

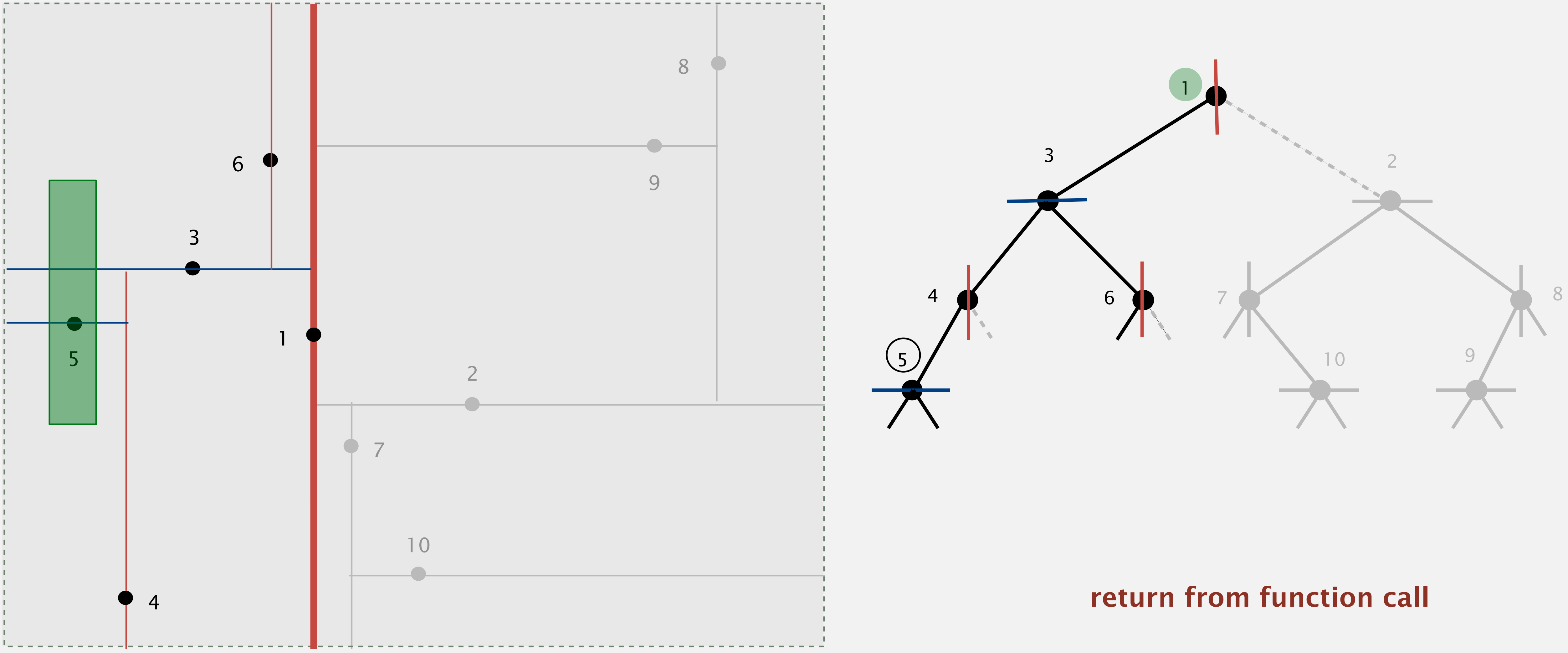


return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

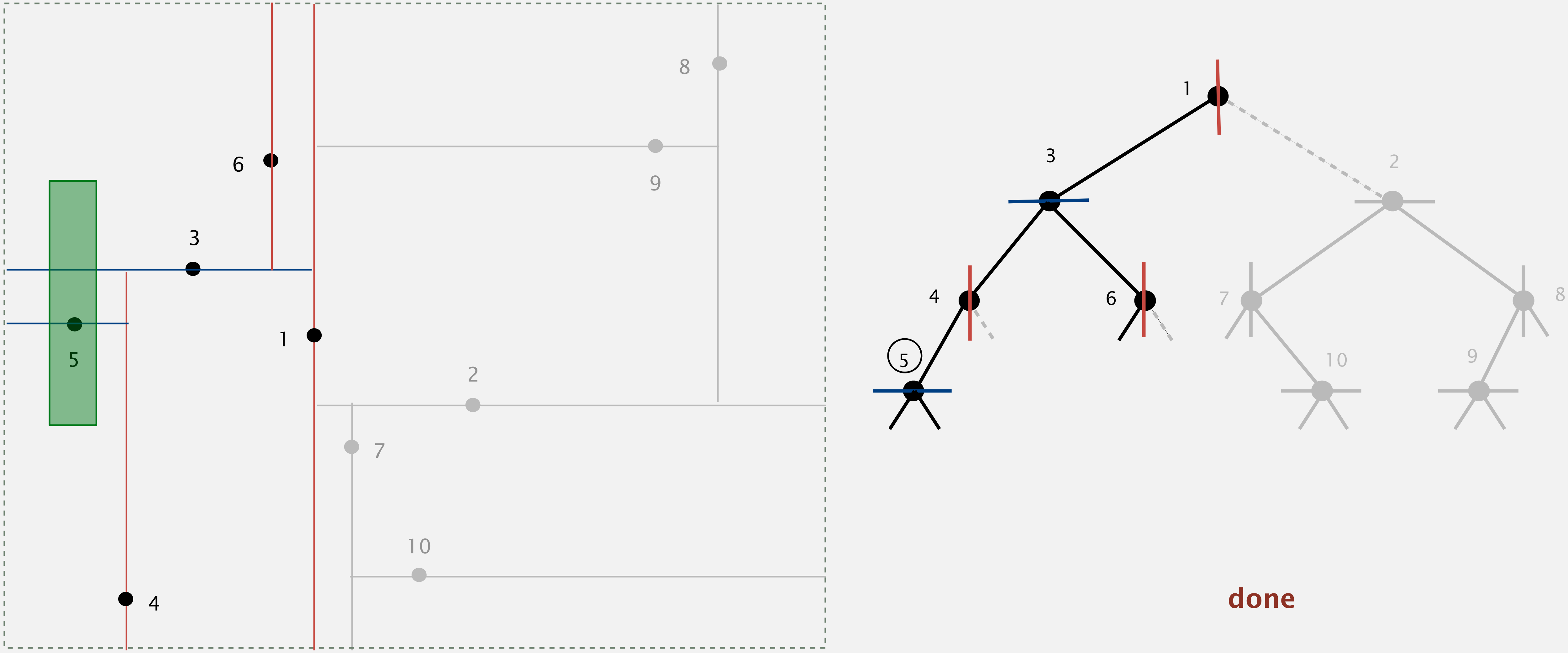


return from function call

2d tree demo: range search

Goal. Find all points in a query axis-aligned rectangle.

- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).

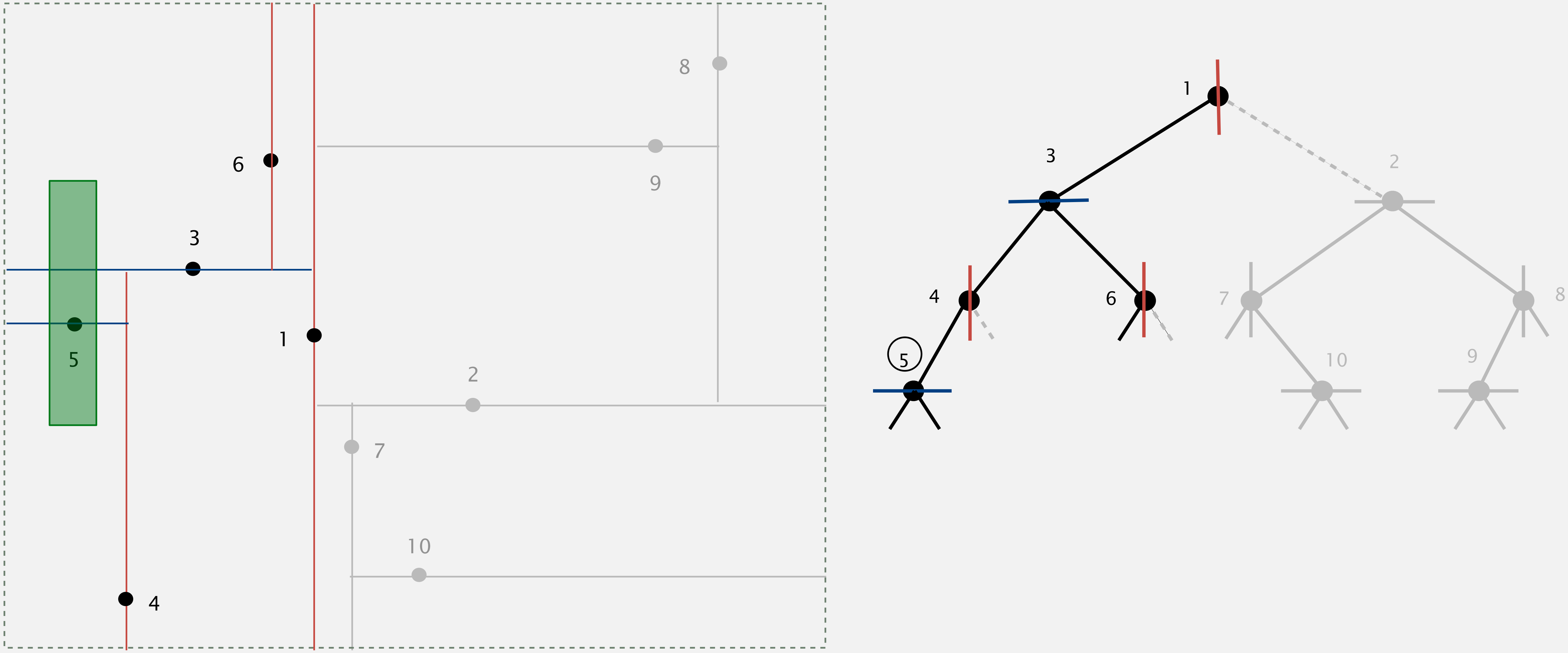


done

Range search in a 2d tree analysis

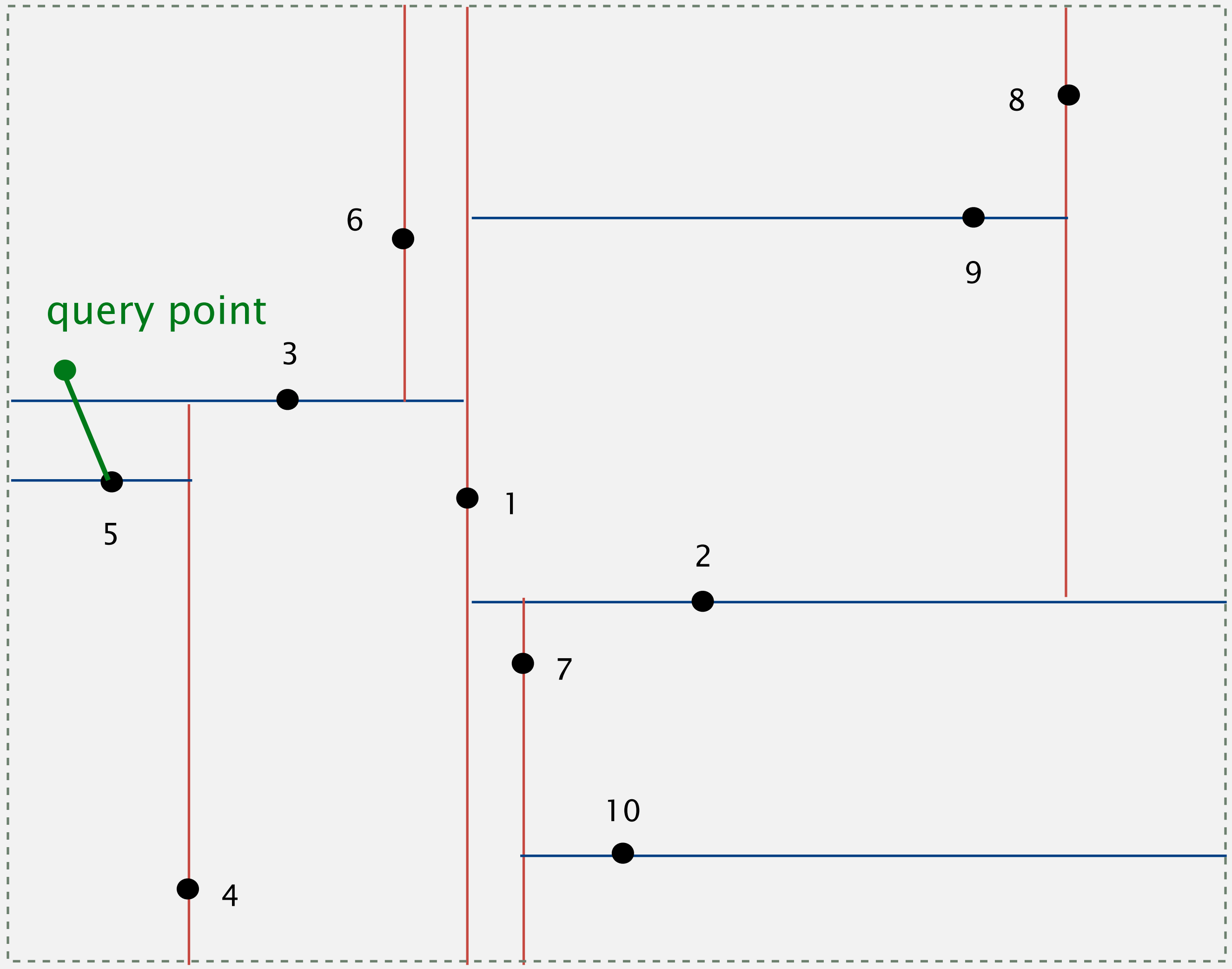
Typical case. $R + \log N$.

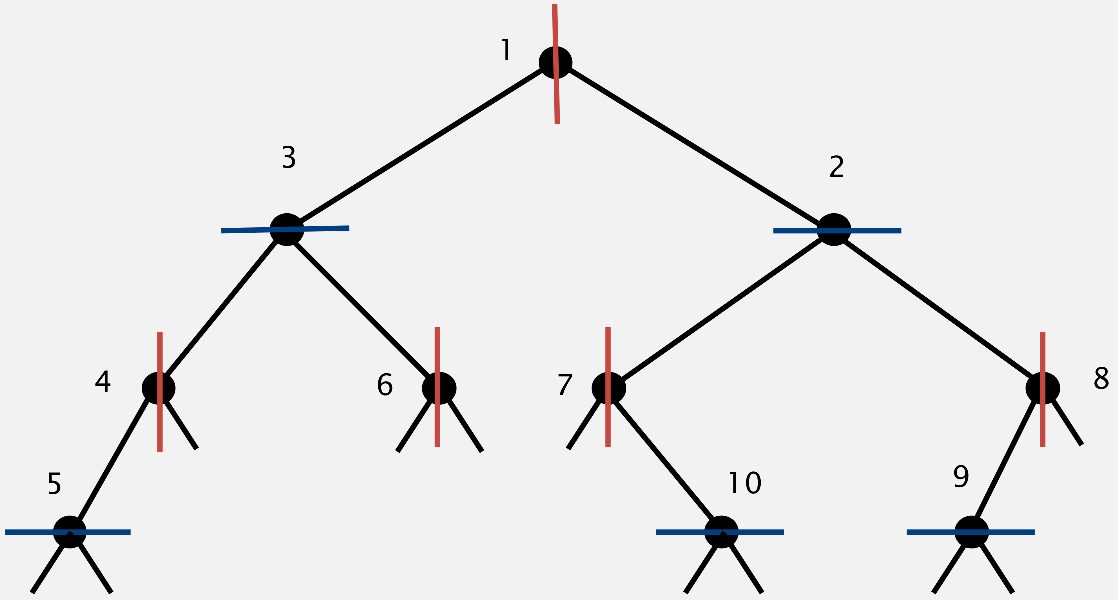
Worst case (assuming tree is balanced). $R + \sqrt{N}$.



2d tree demo: nearest neighbor

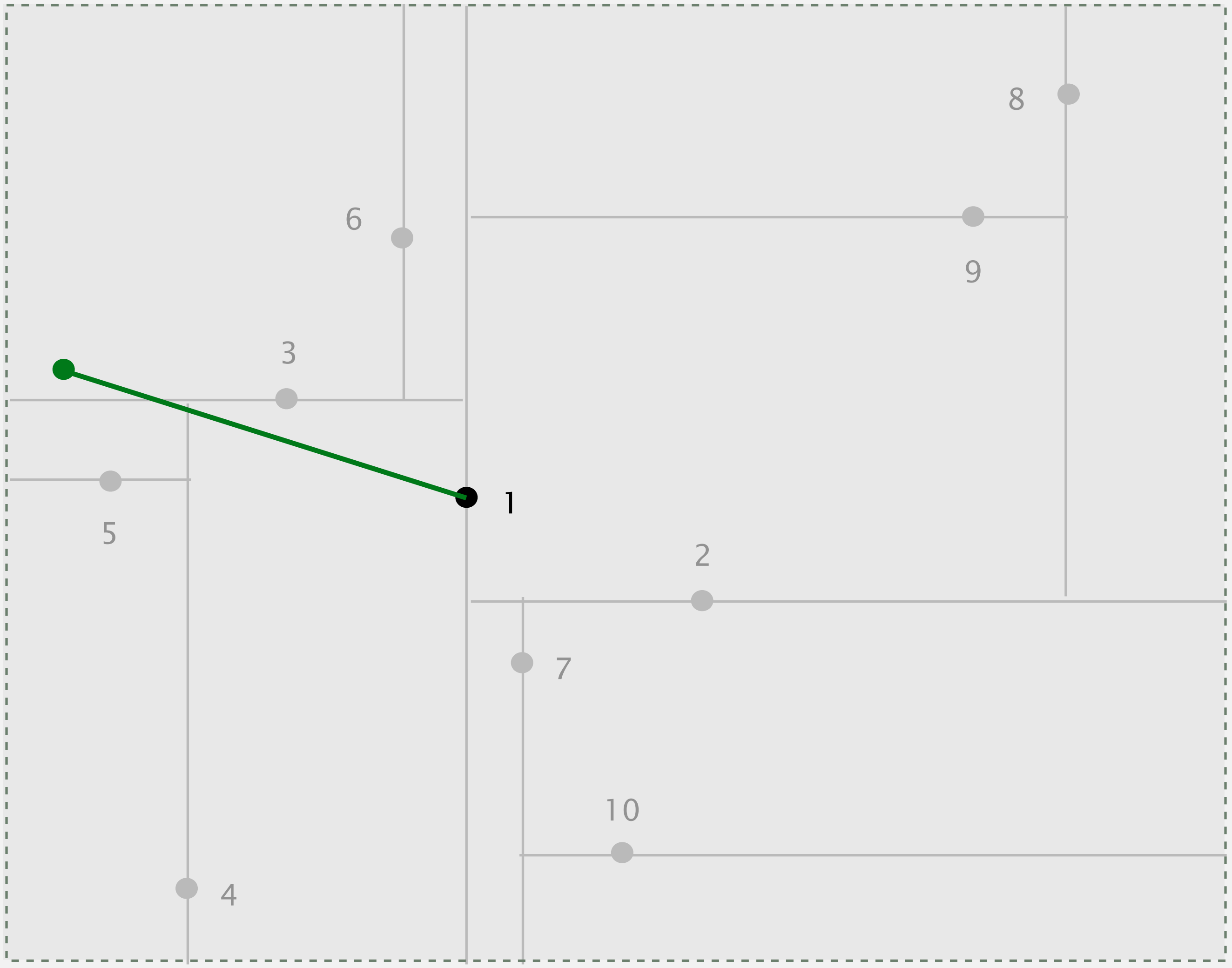
Goal. Find closest point to query point.

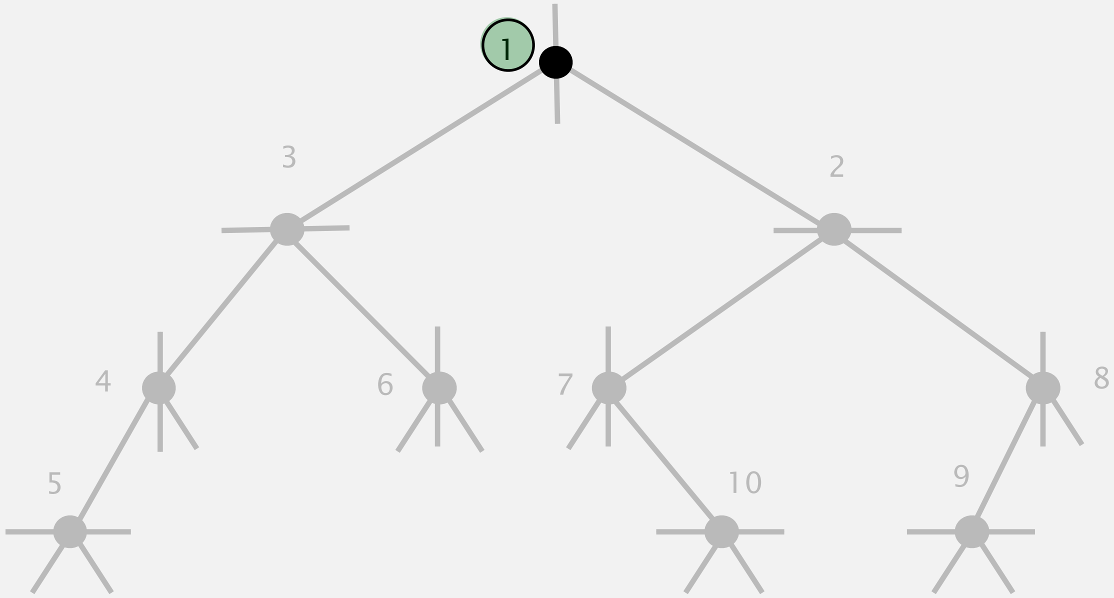




2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.





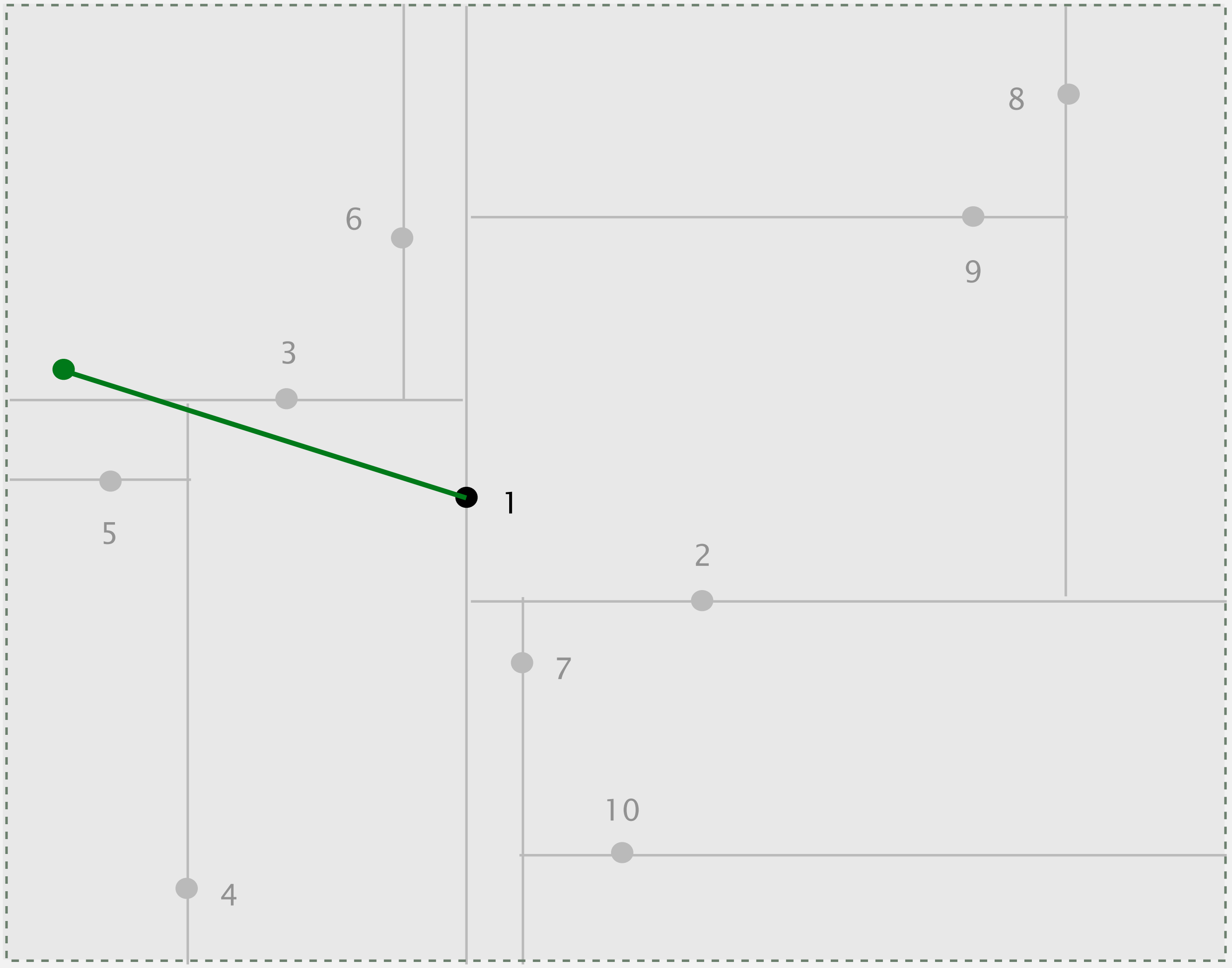
search root node

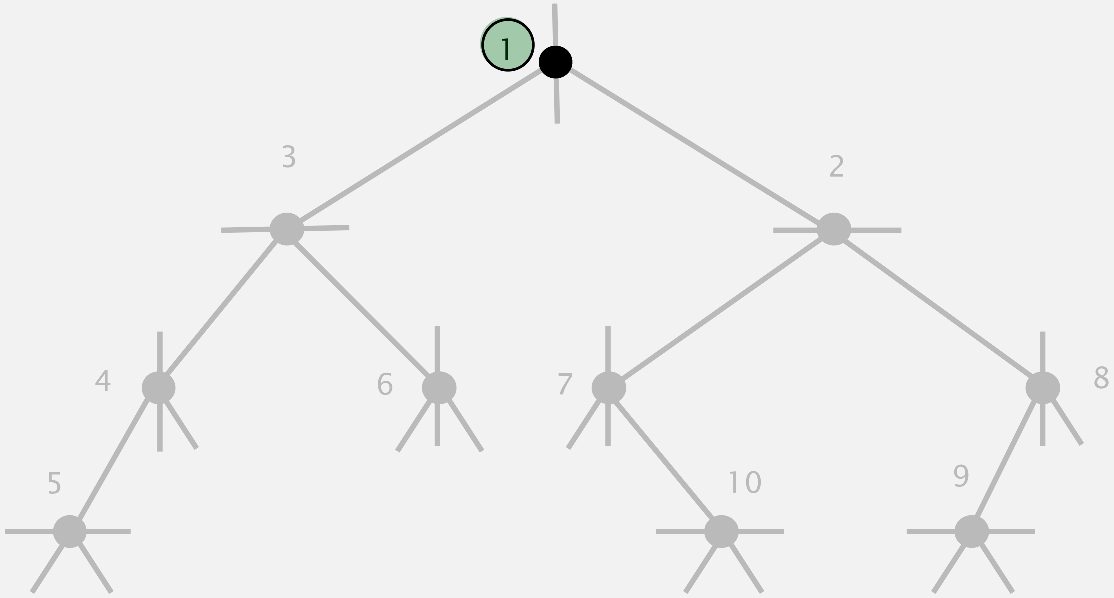
compute distance from query point to 1

(update champion nearest neighbor)

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

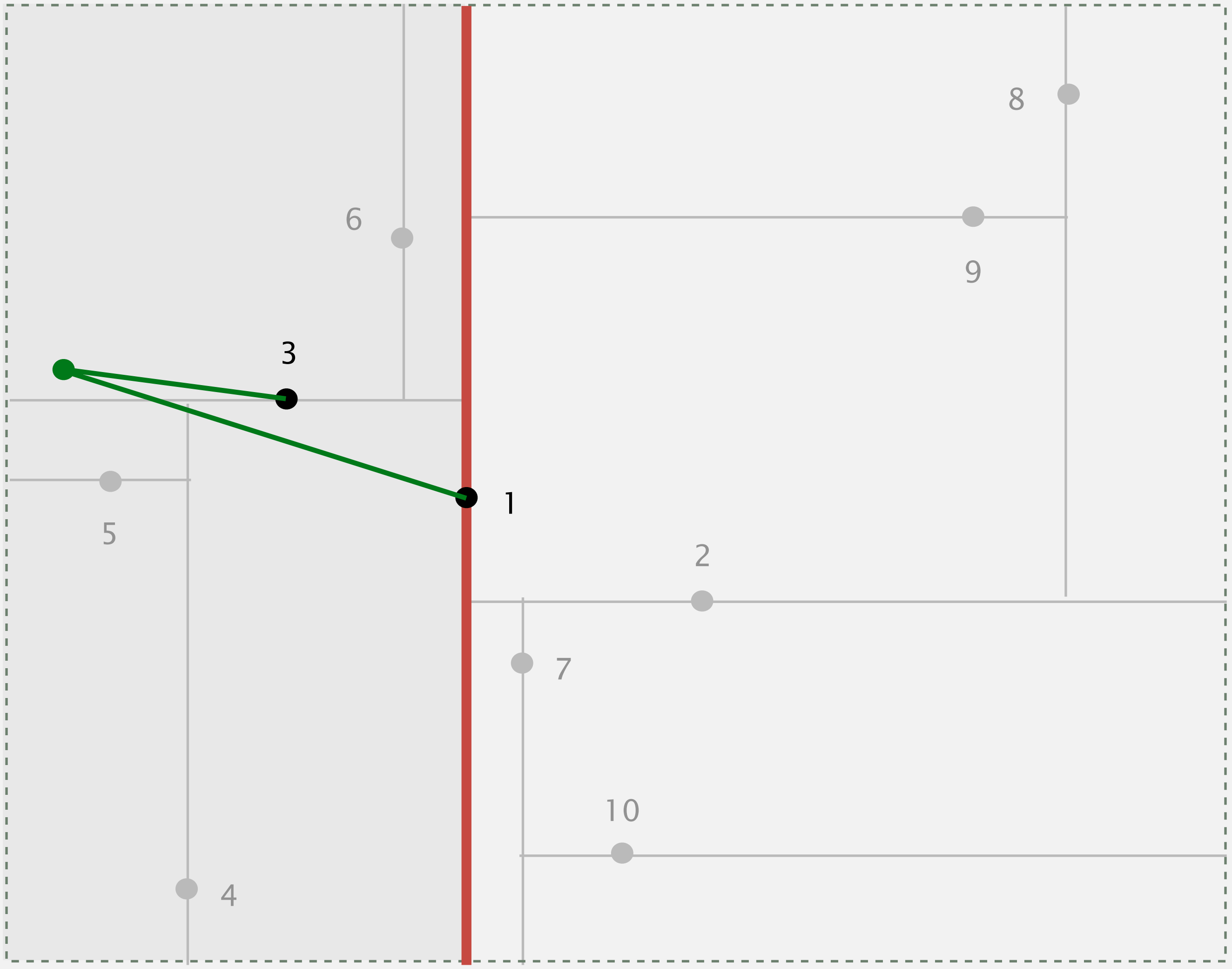


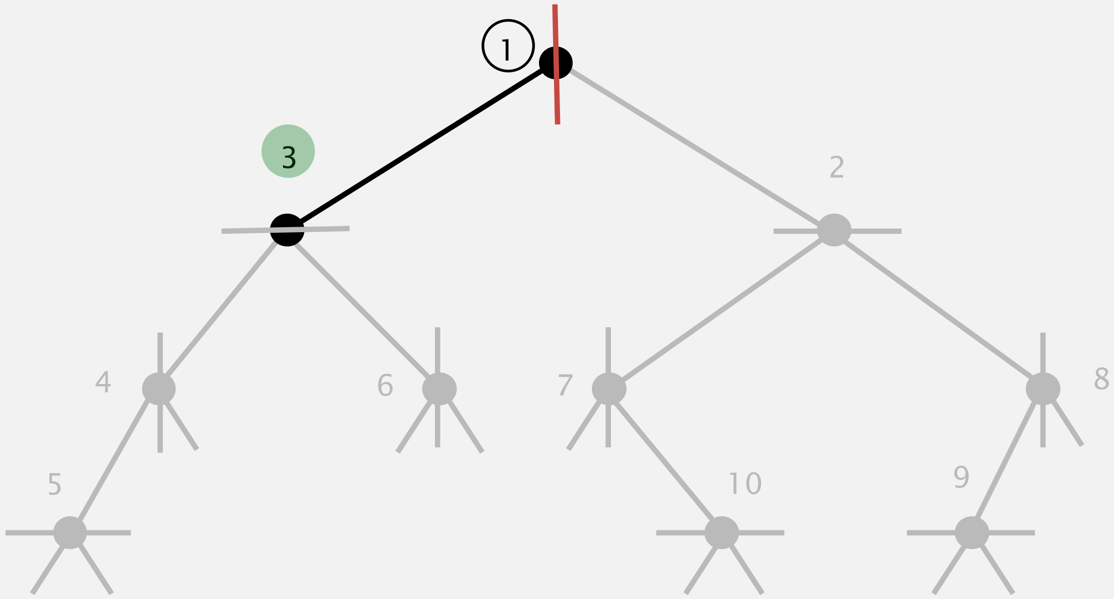


query point is to the left of splitting line
search left subtree first

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.





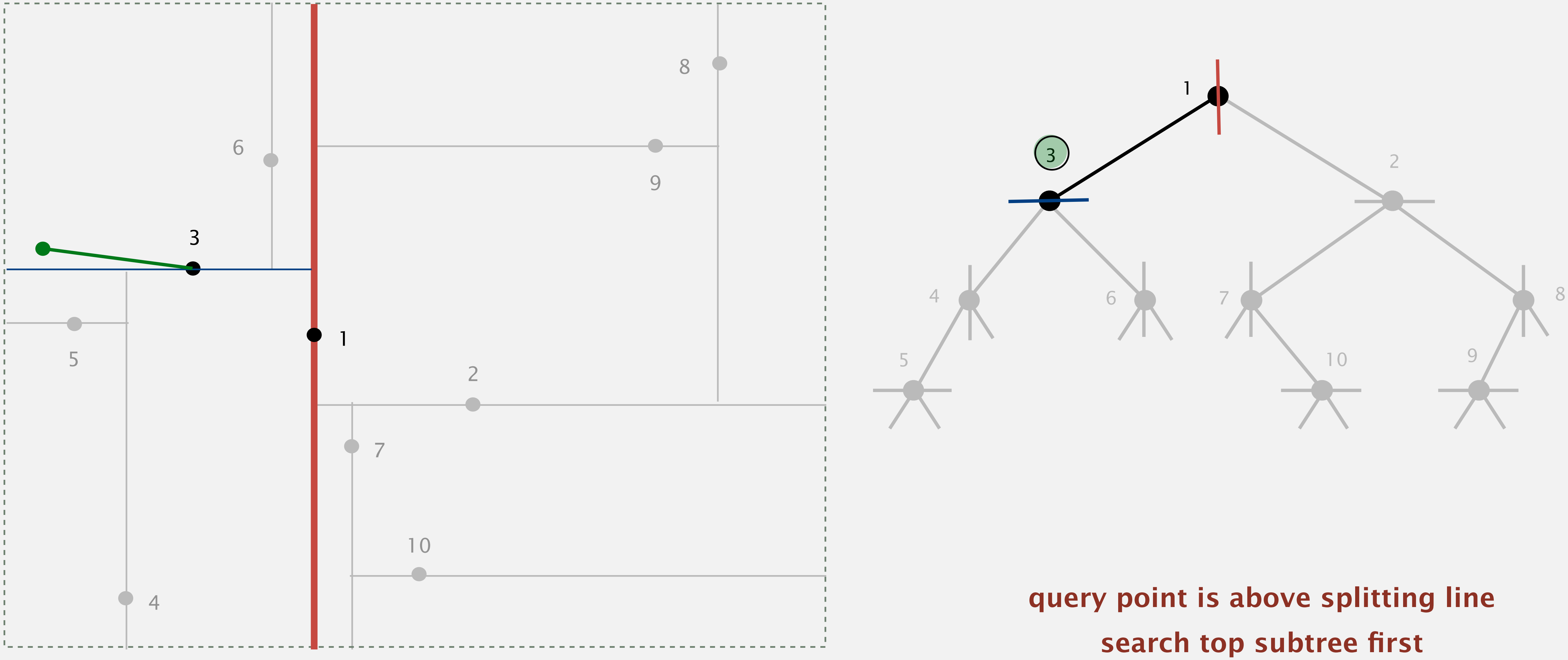
search left subtree

compute distance from query point to 3

(update champion)

2d tree demo: nearest neighbor

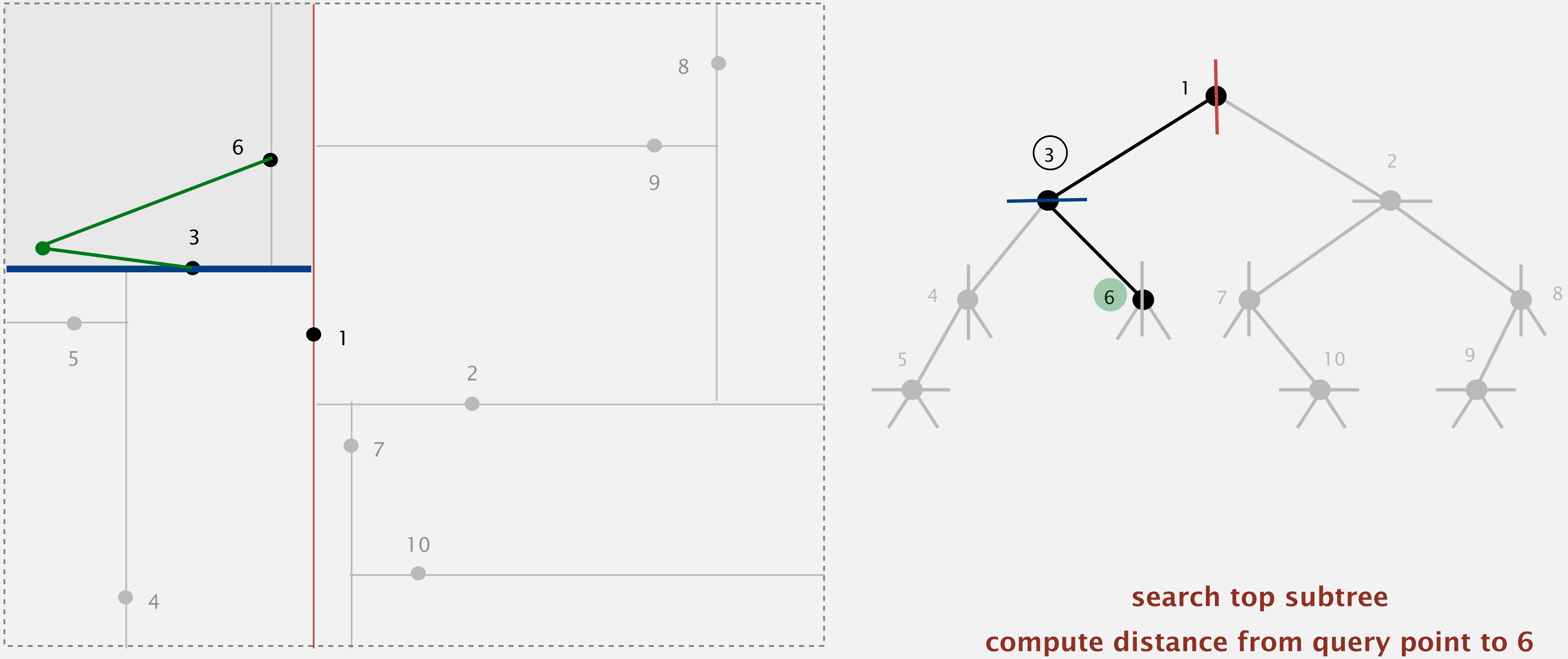
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



query point is above splitting line
search top subtree first

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

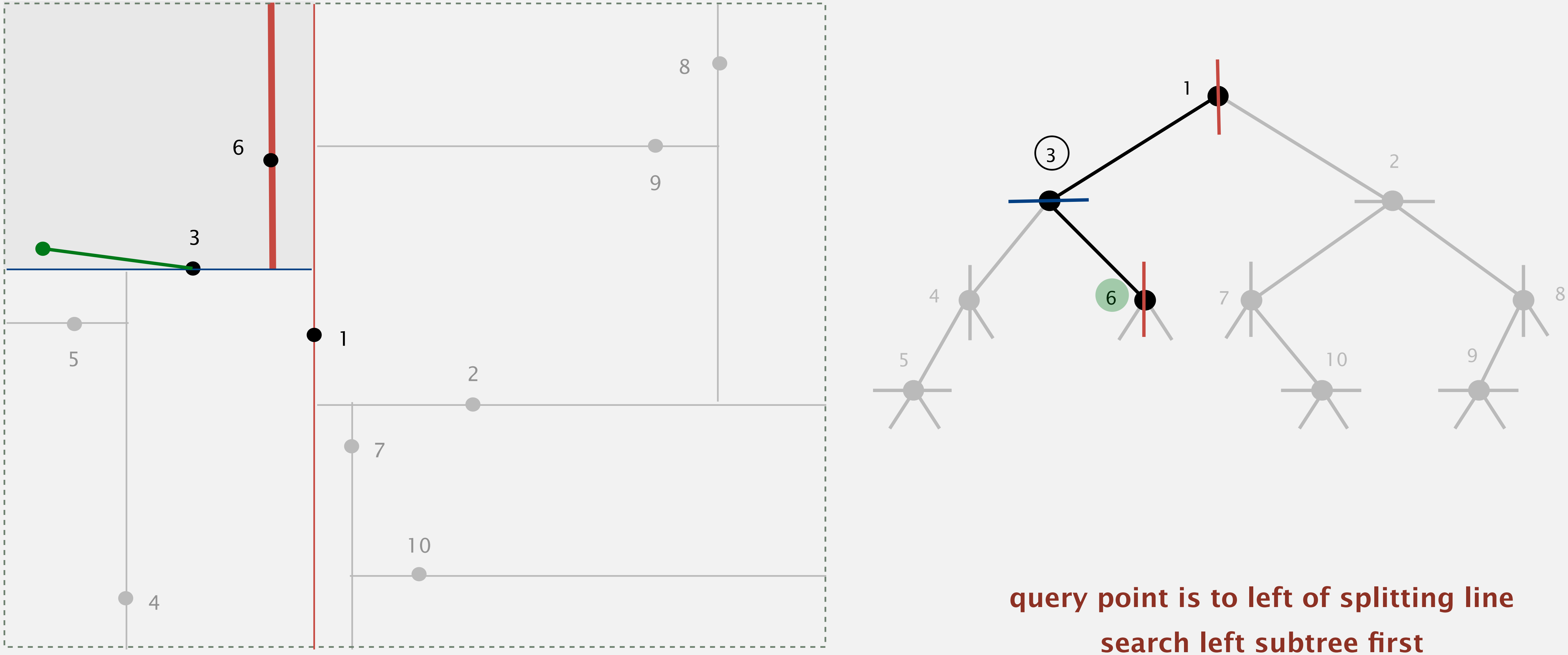


search top subtree

compute distance from query point to 6

2d tree demo: nearest neighbor

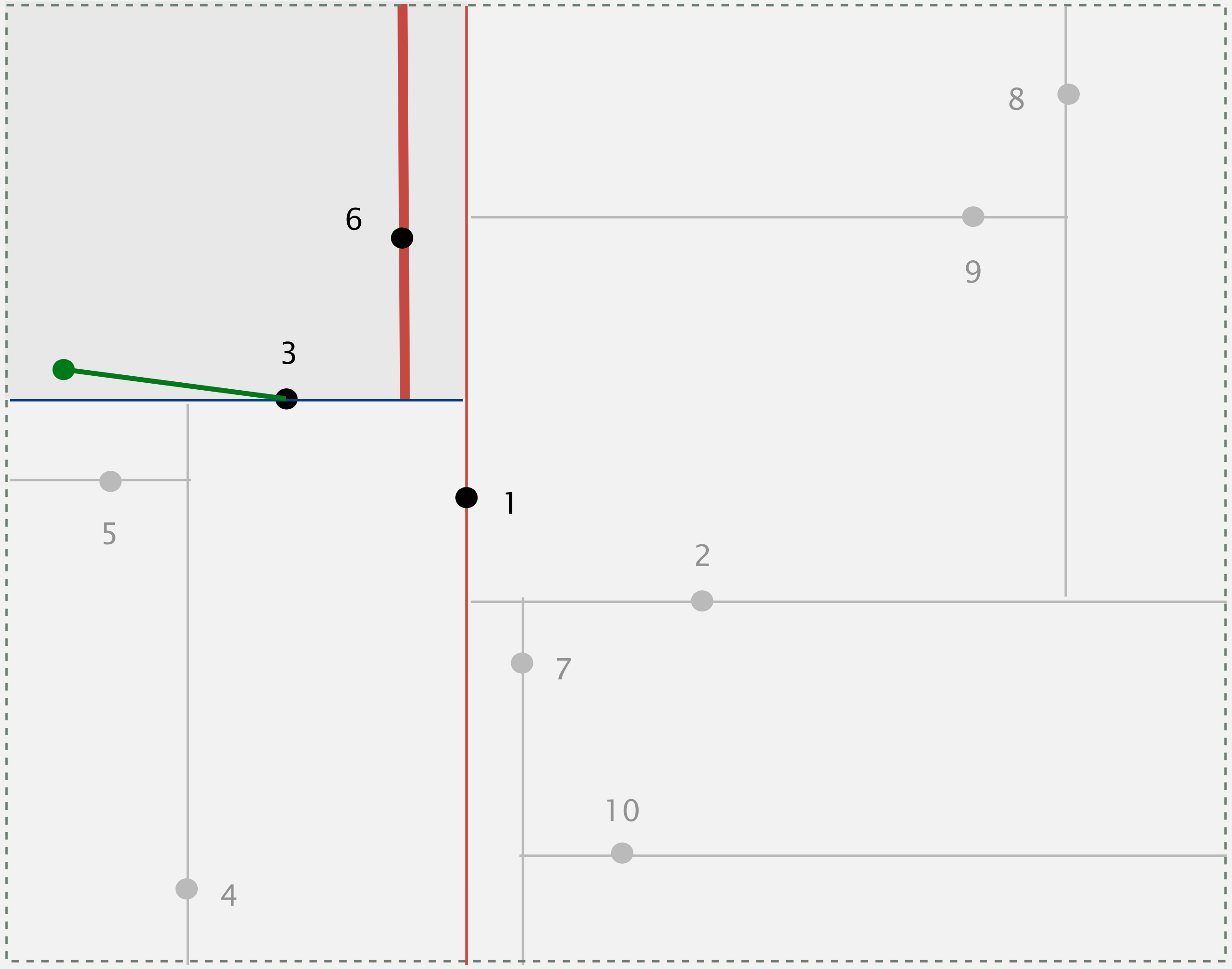
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

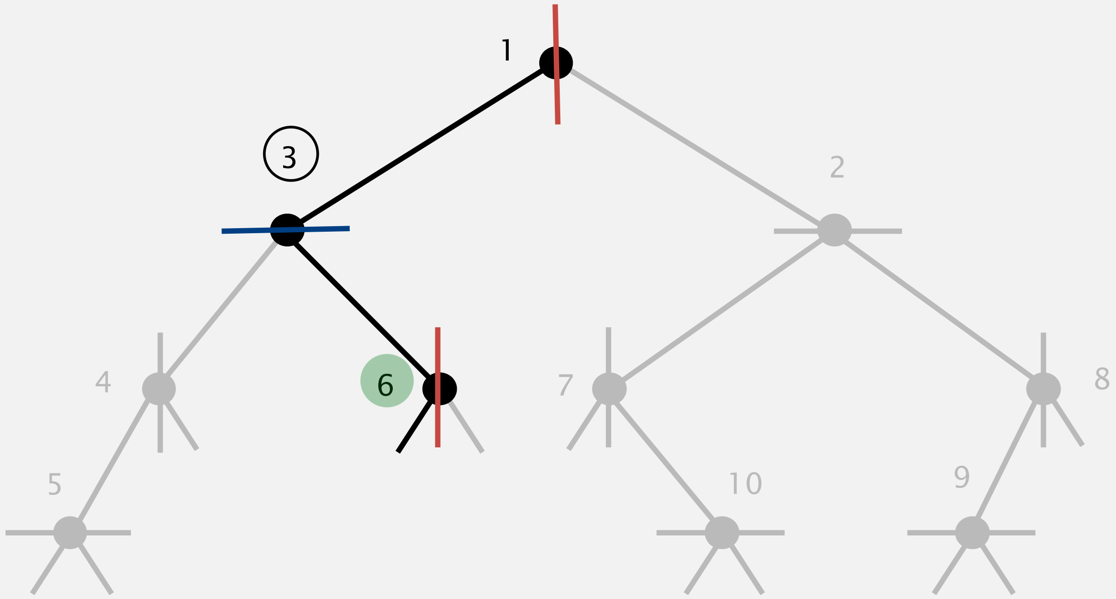


query point is to left of splitting line
search left subtree first

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

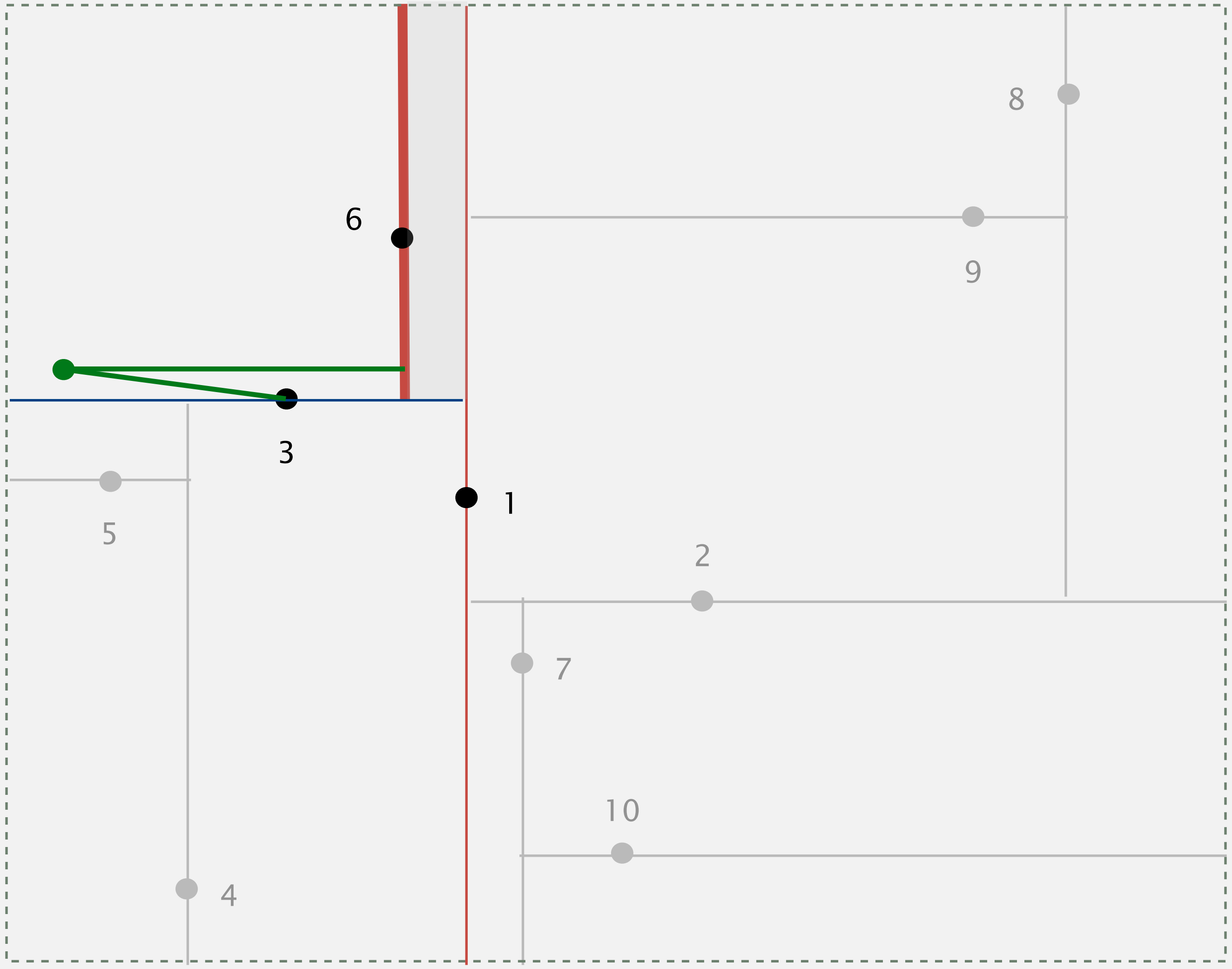


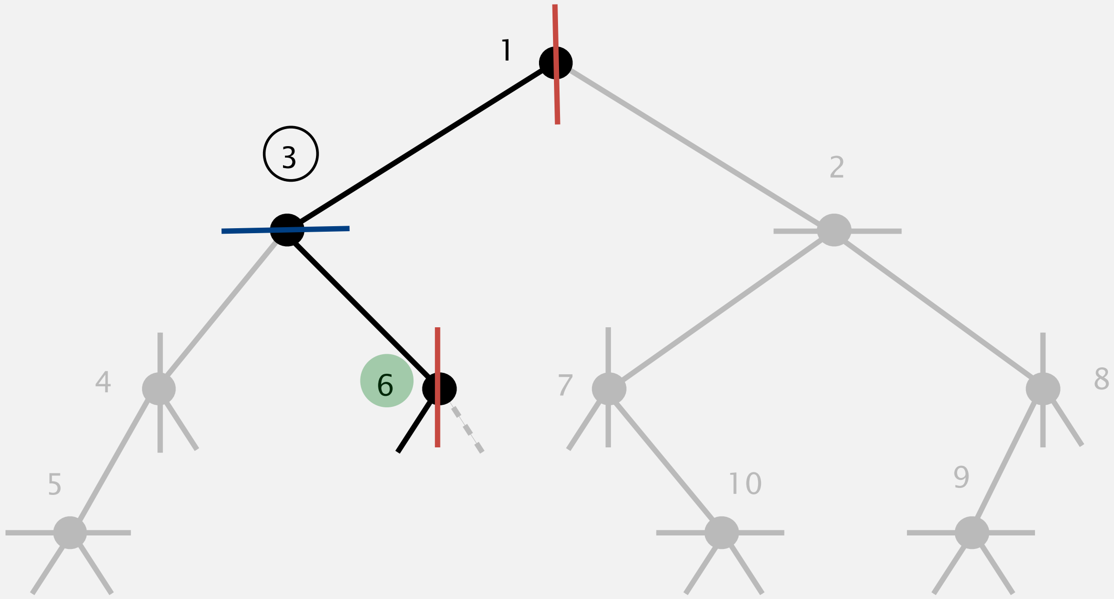


search left subtree
return since empty

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

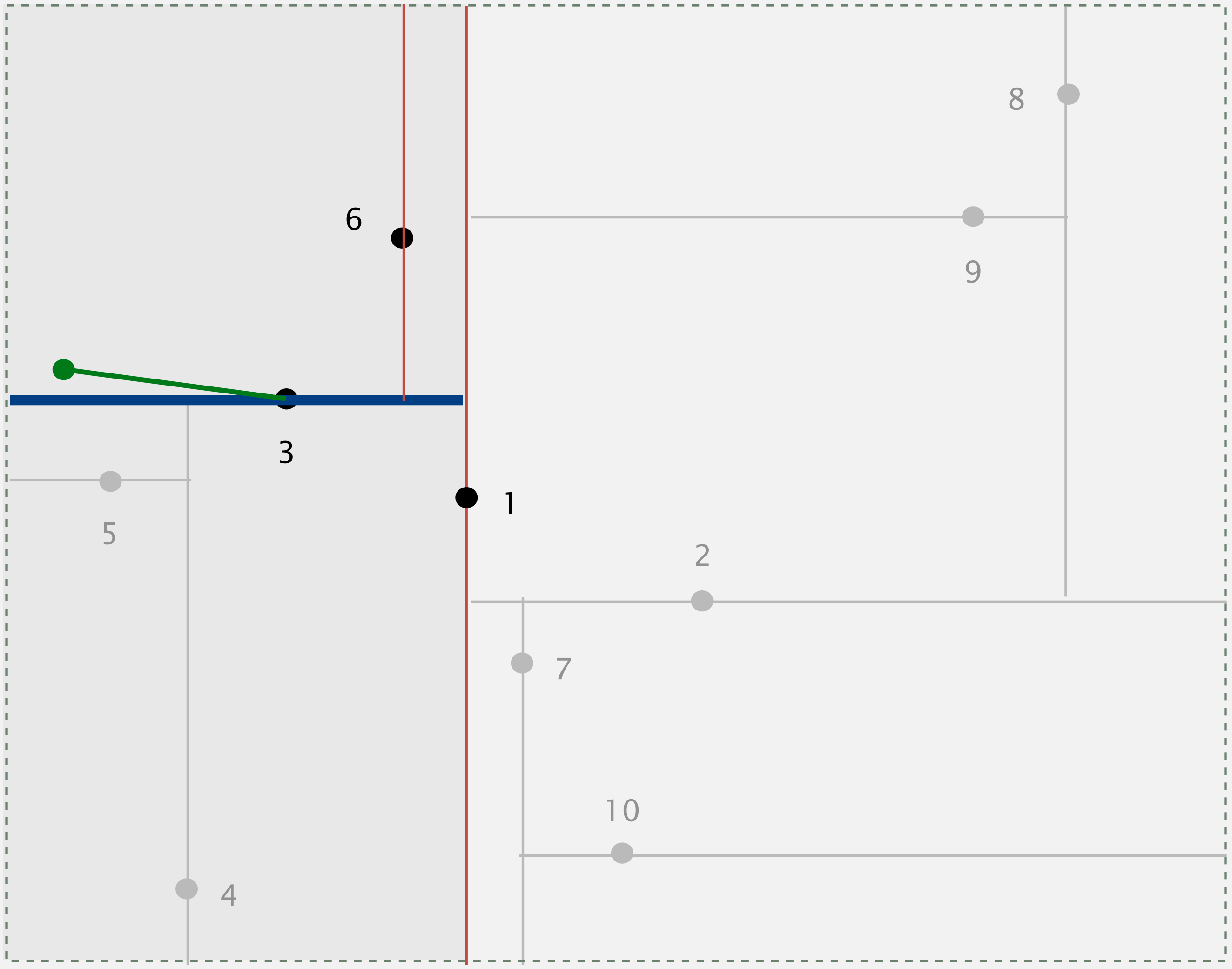


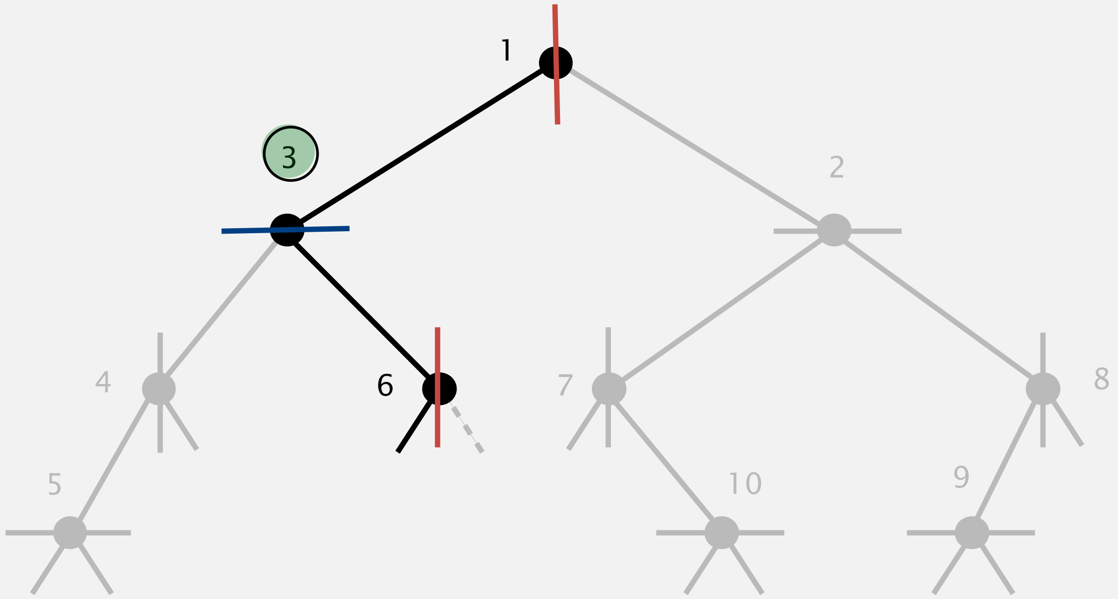


search right subtree
prune since nearest neighbor
can't be here

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

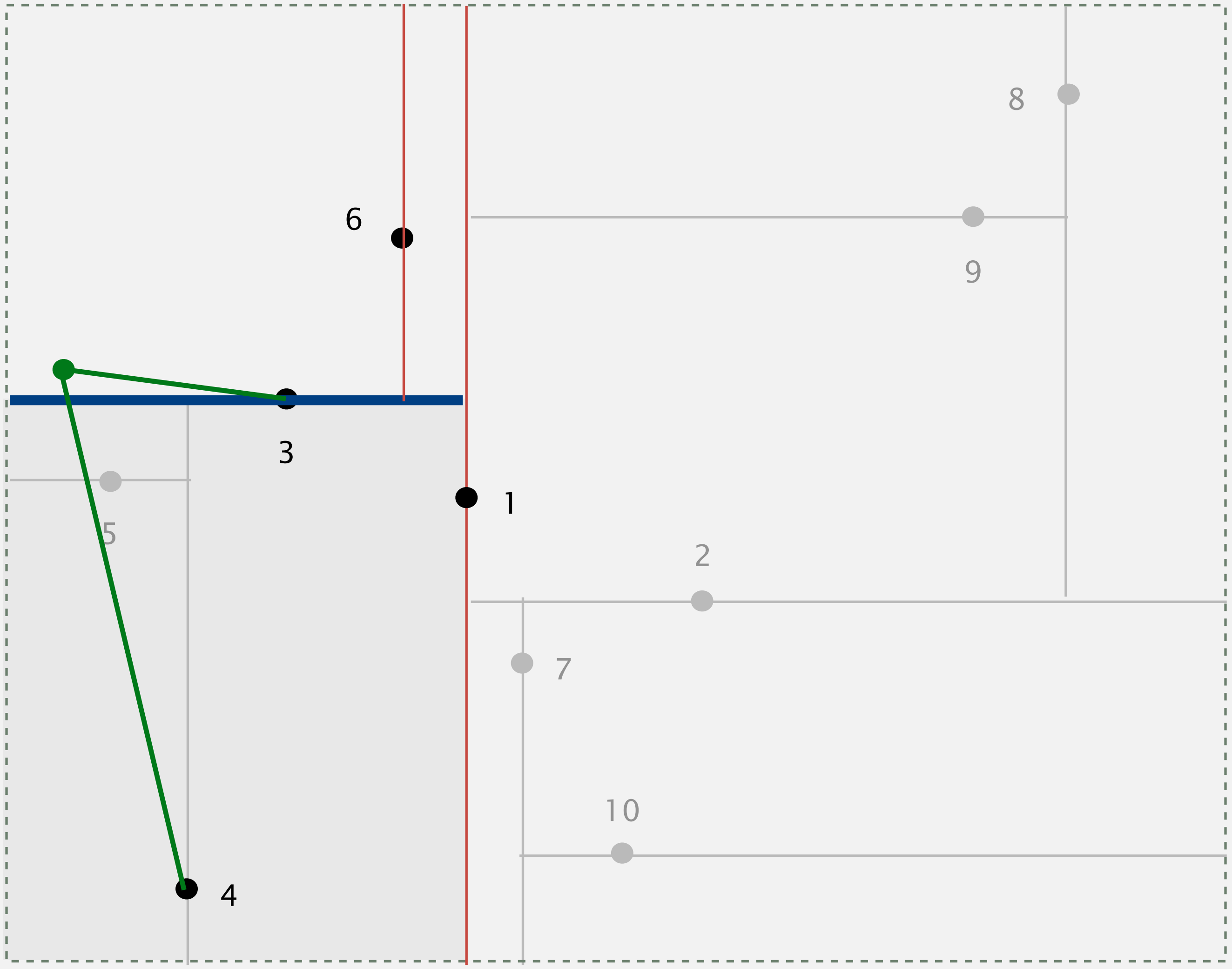


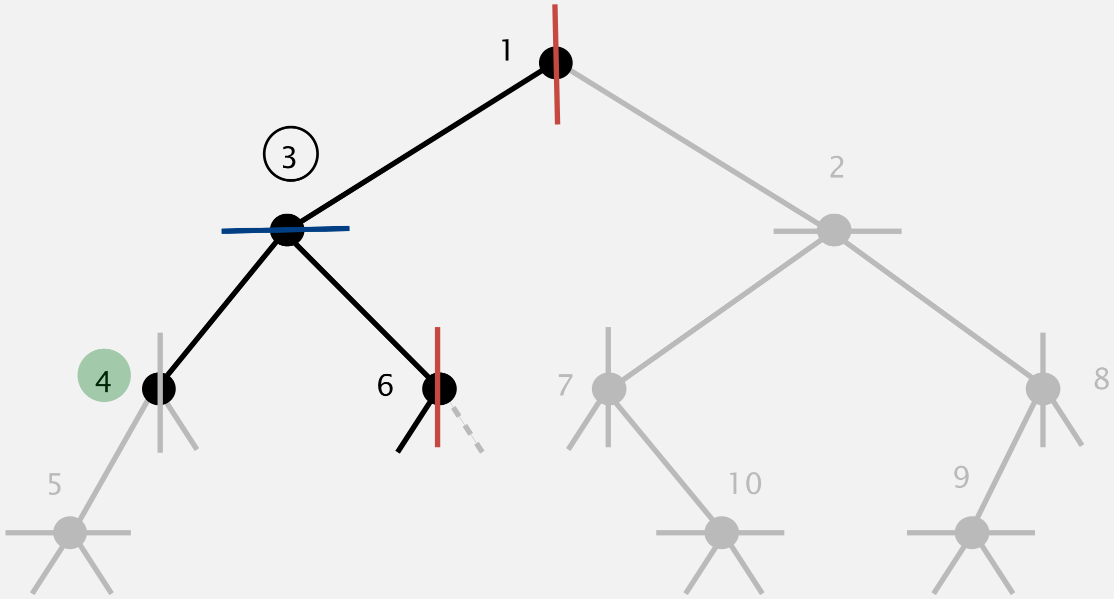


return from function call
search bottom subtree next

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



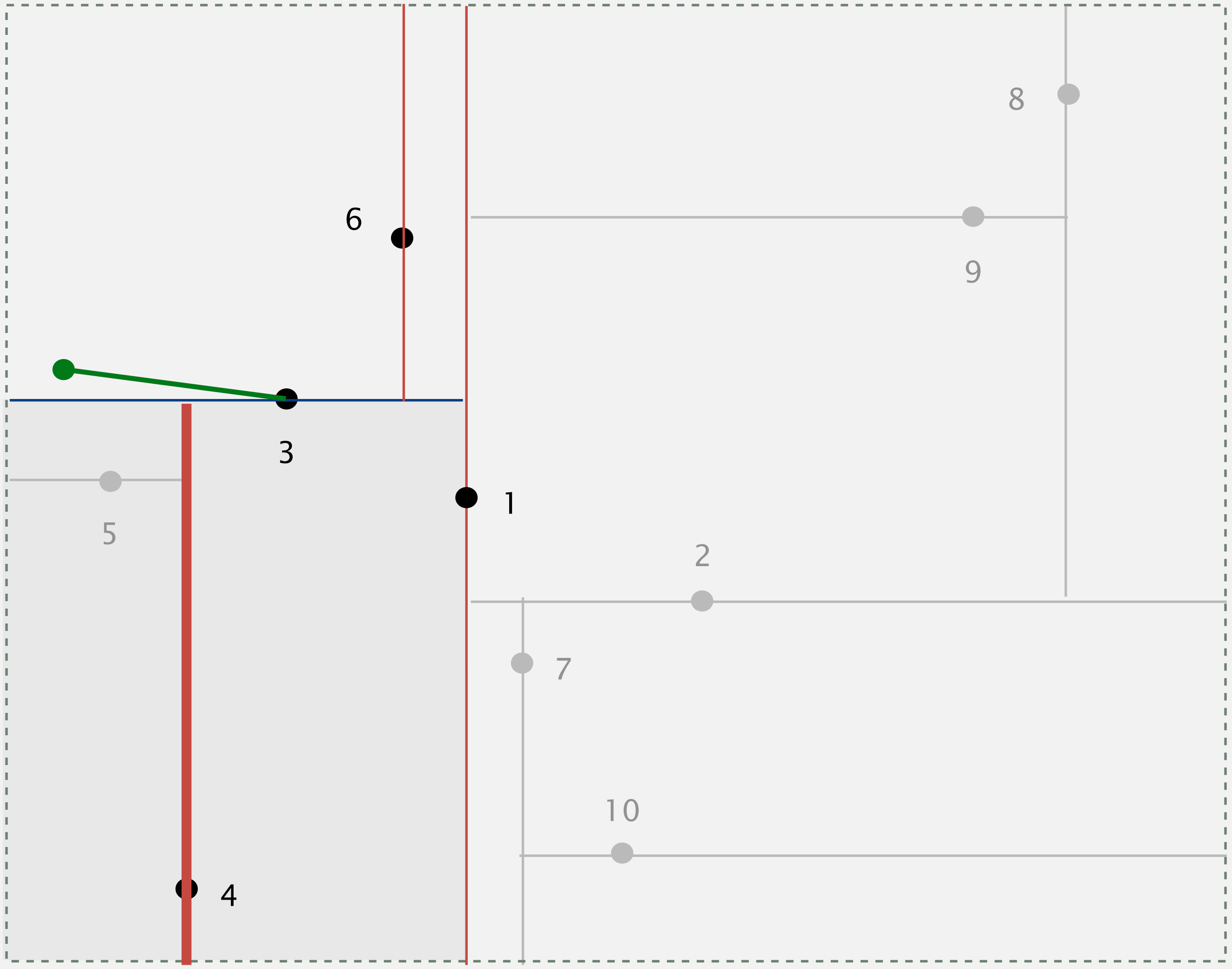


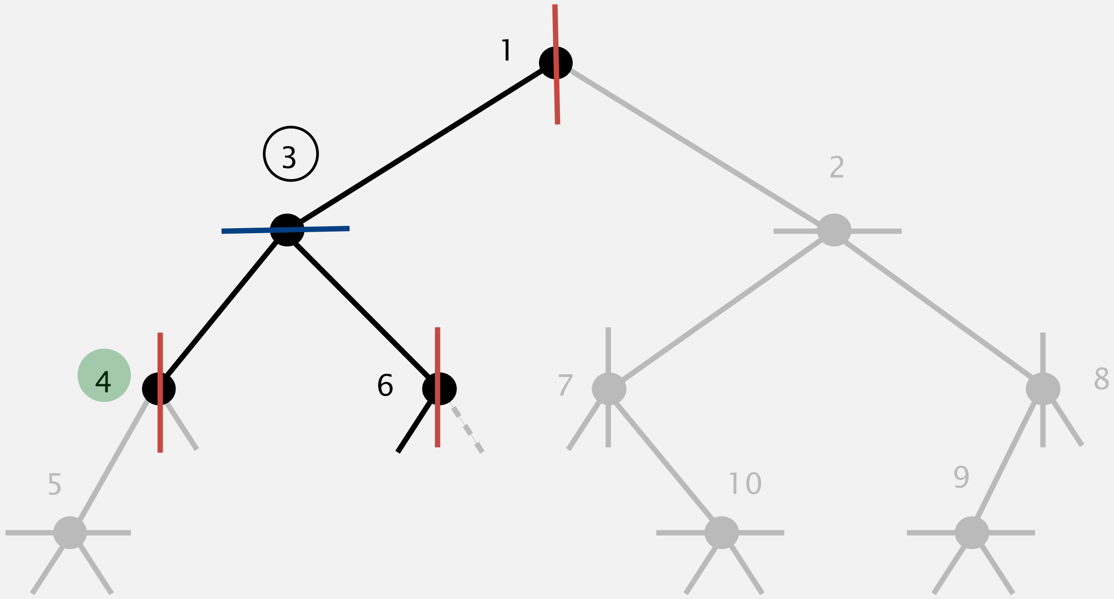
search bottom subtree

compute distance from query point to 4

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

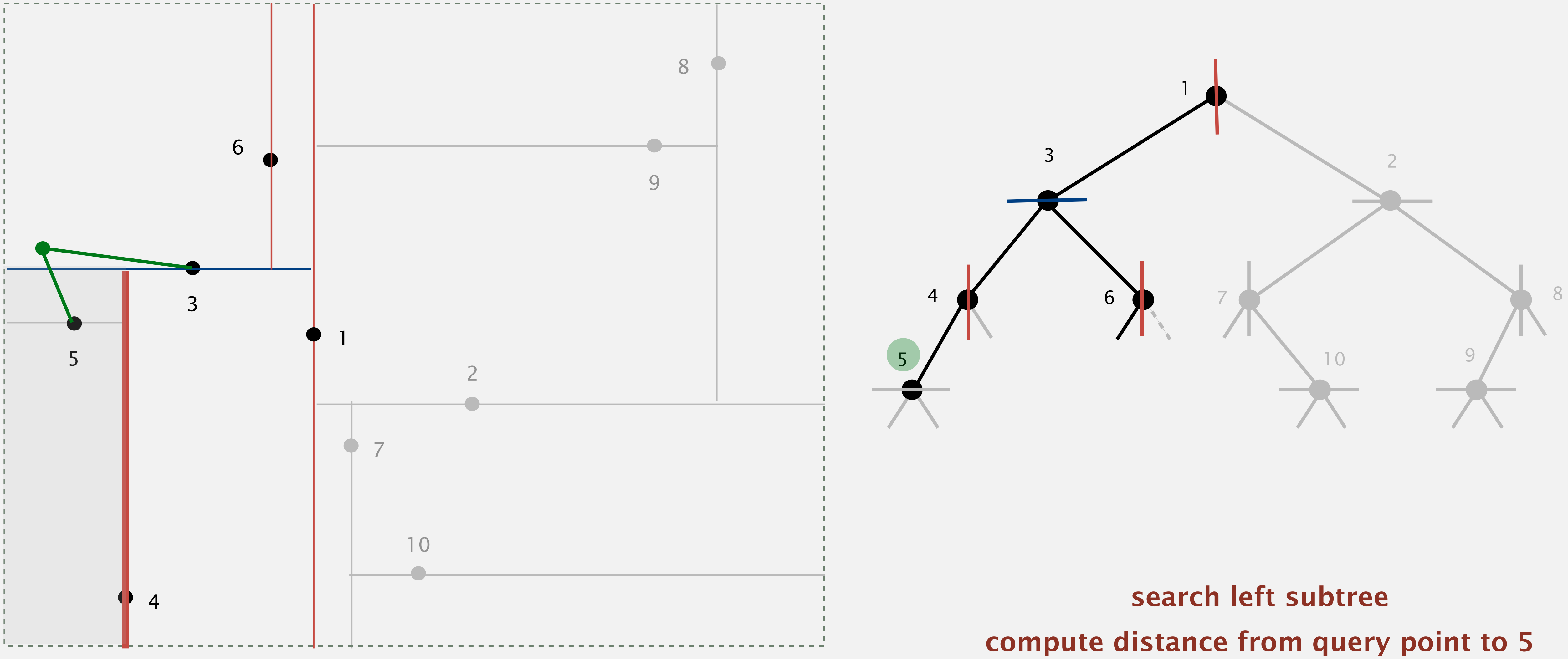




query point is to left of splitting line
search left subtree first

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



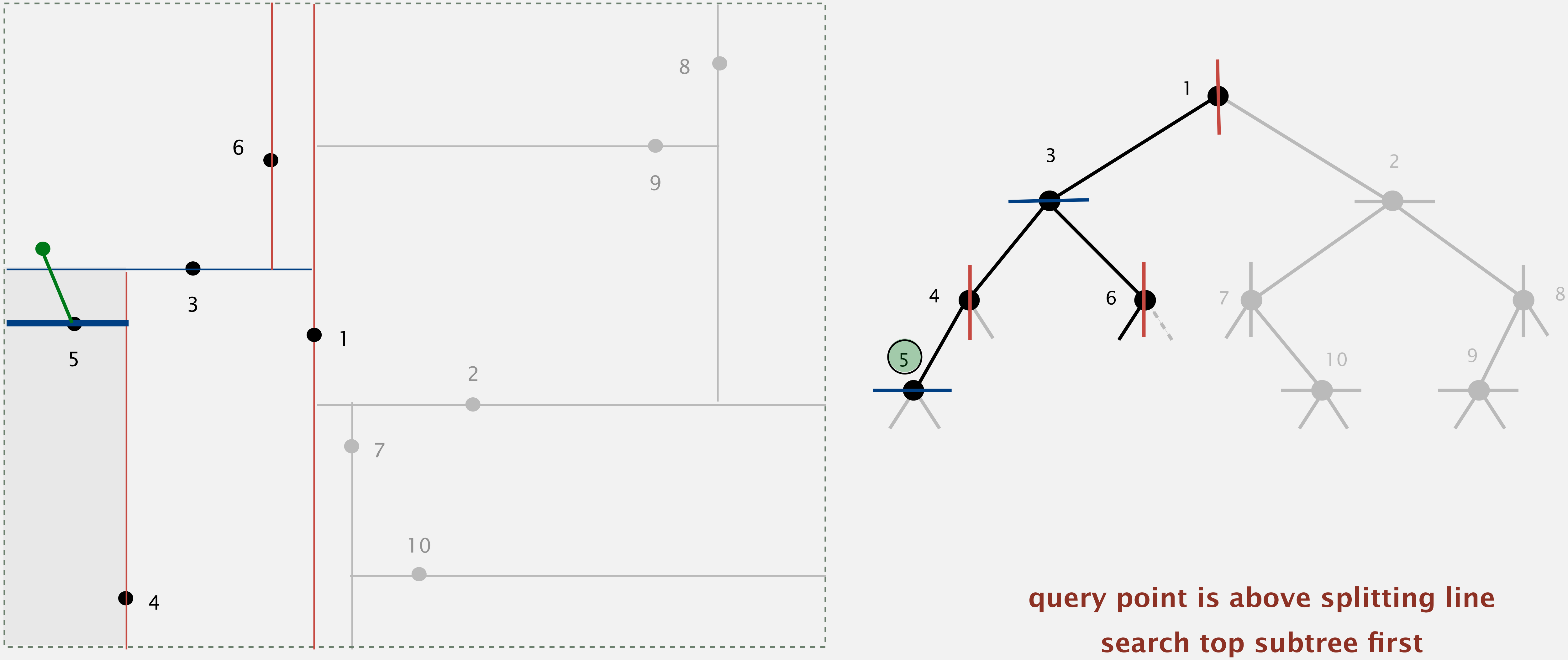
search left subtree

compute distance from query point to 5

(update champion)

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

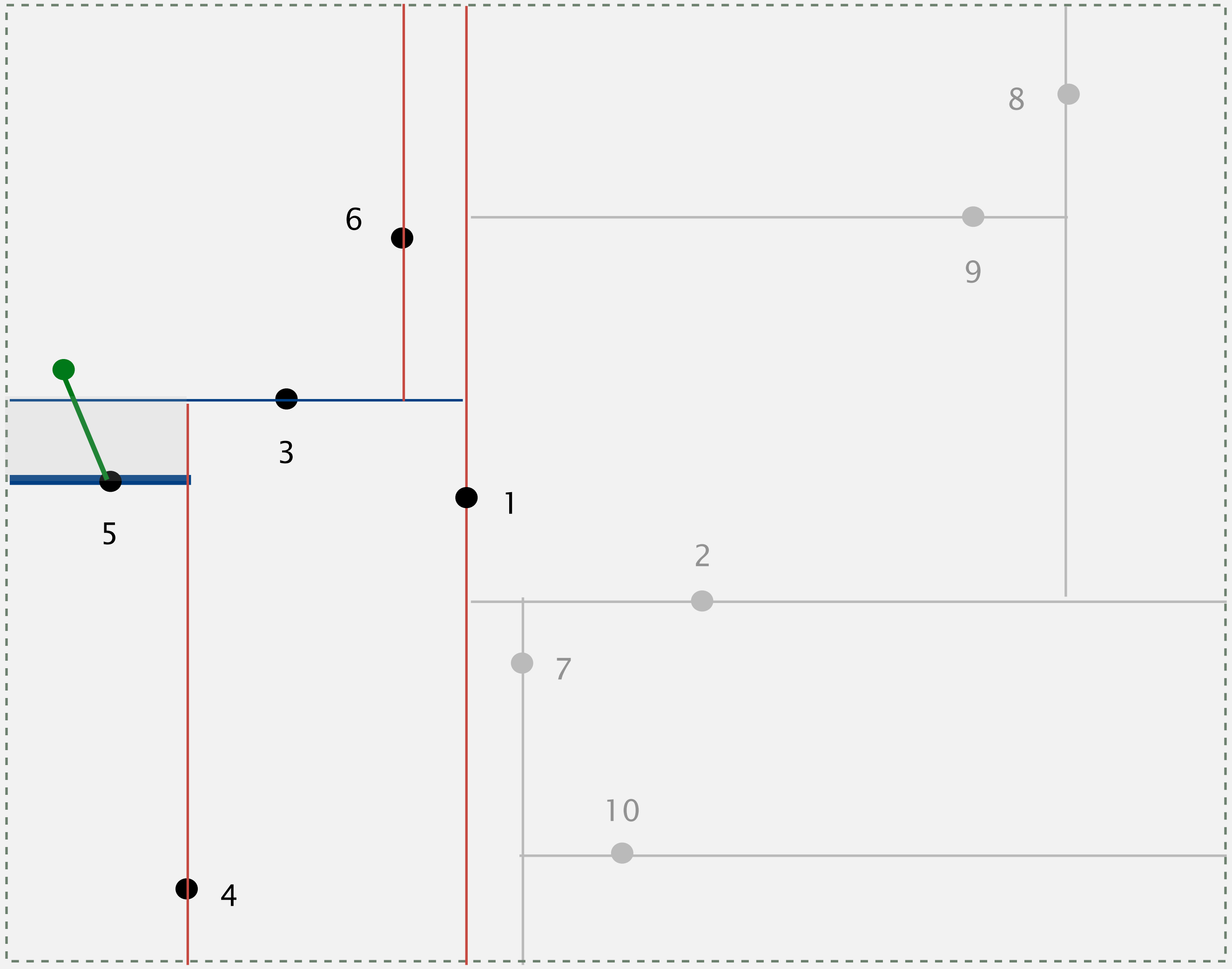


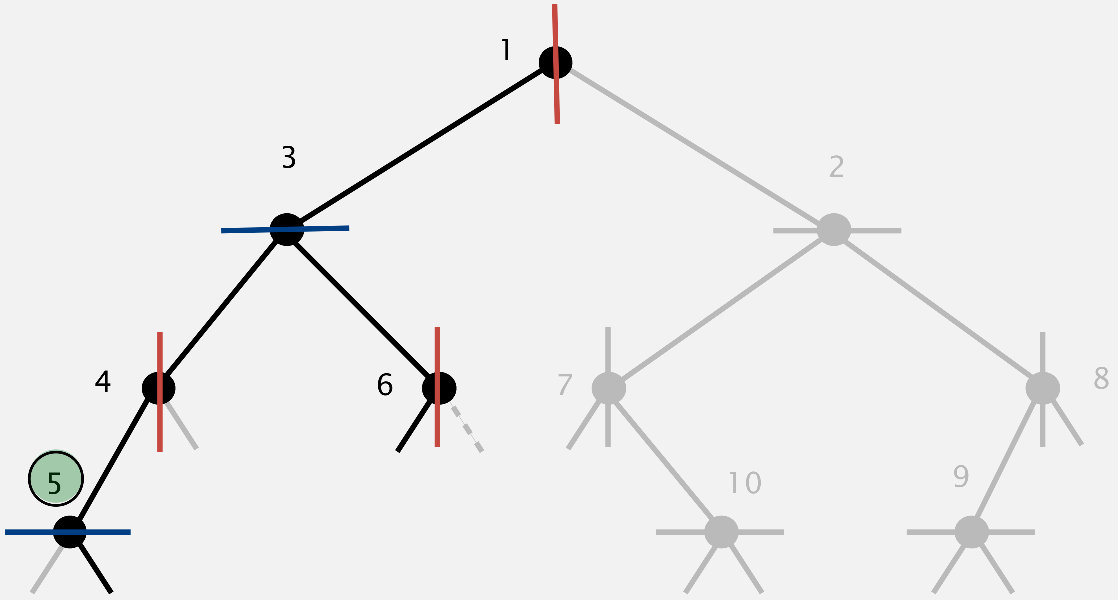
query point is above splitting line

search top subtree first

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

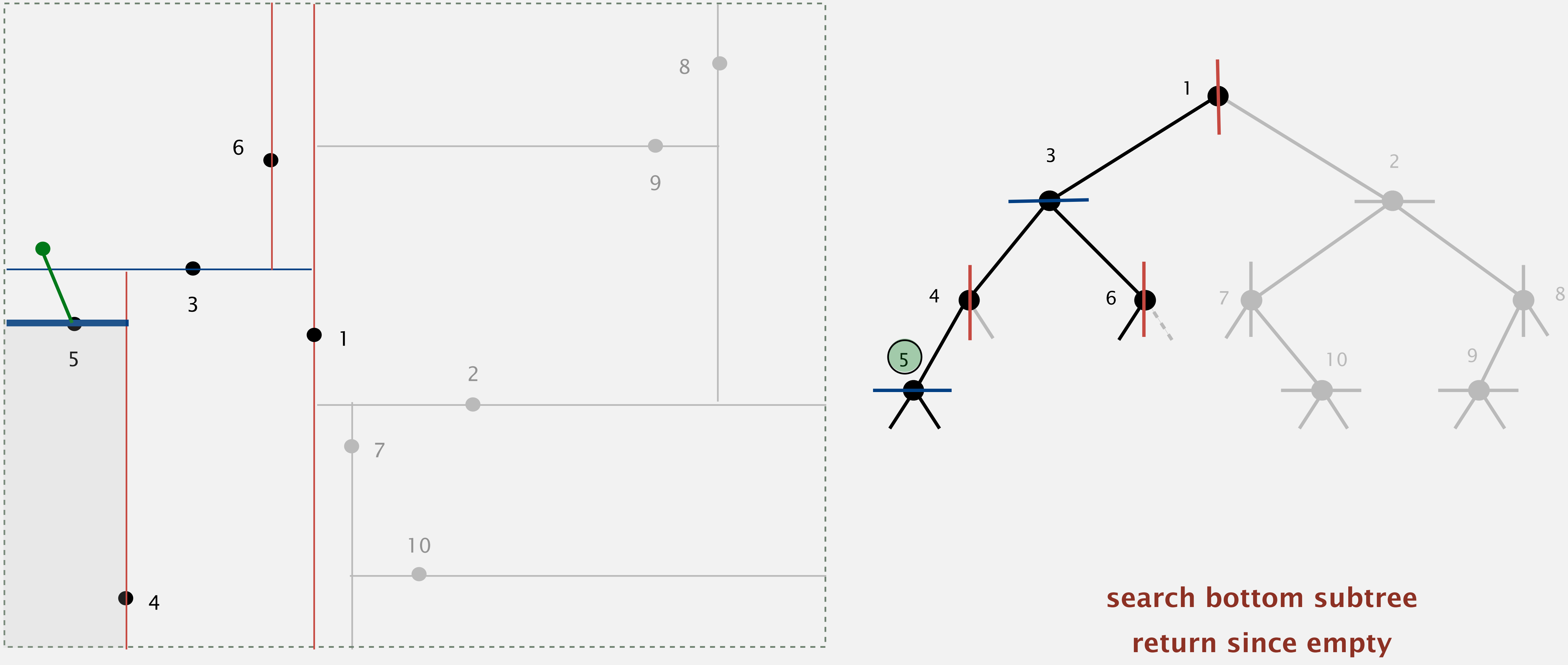




search top subtree
return since empty

2d tree demo: nearest neighbor

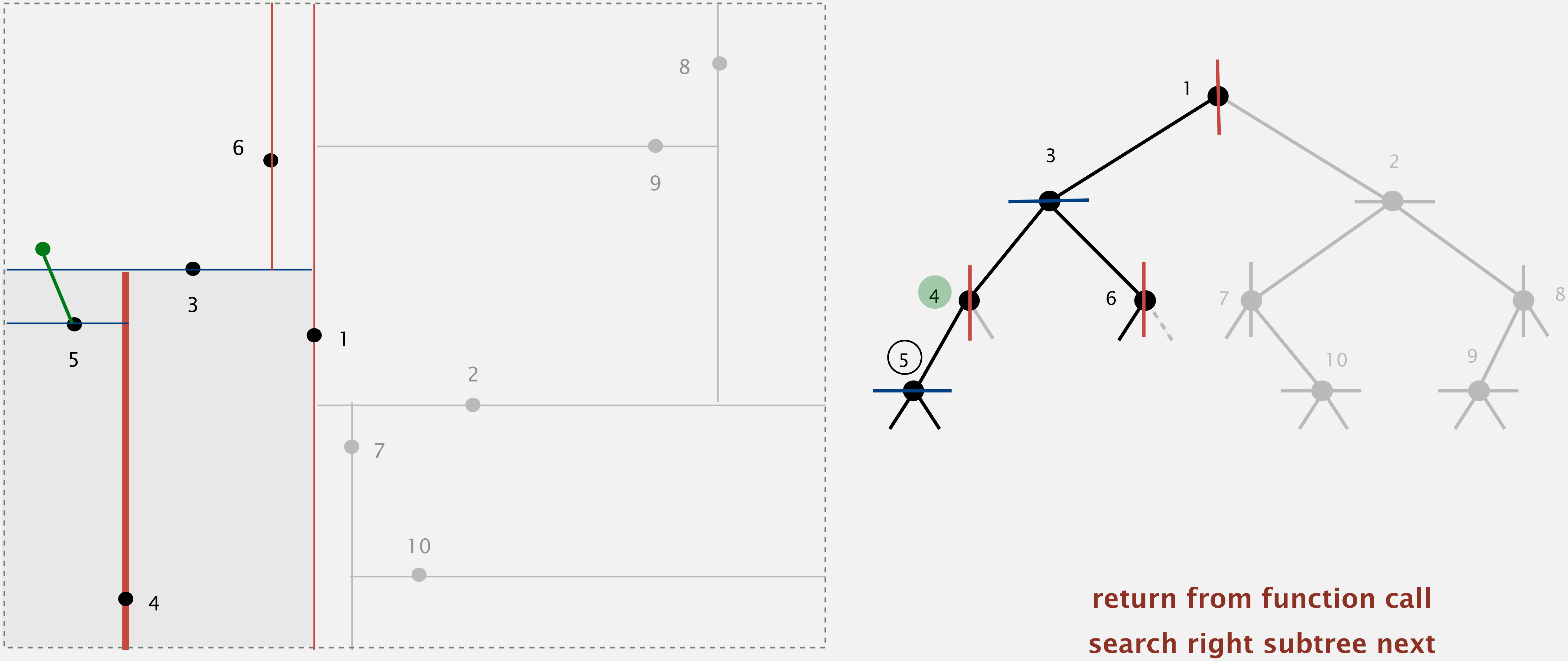
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



search bottom subtree
return since empty

2d tree demo: nearest neighbor

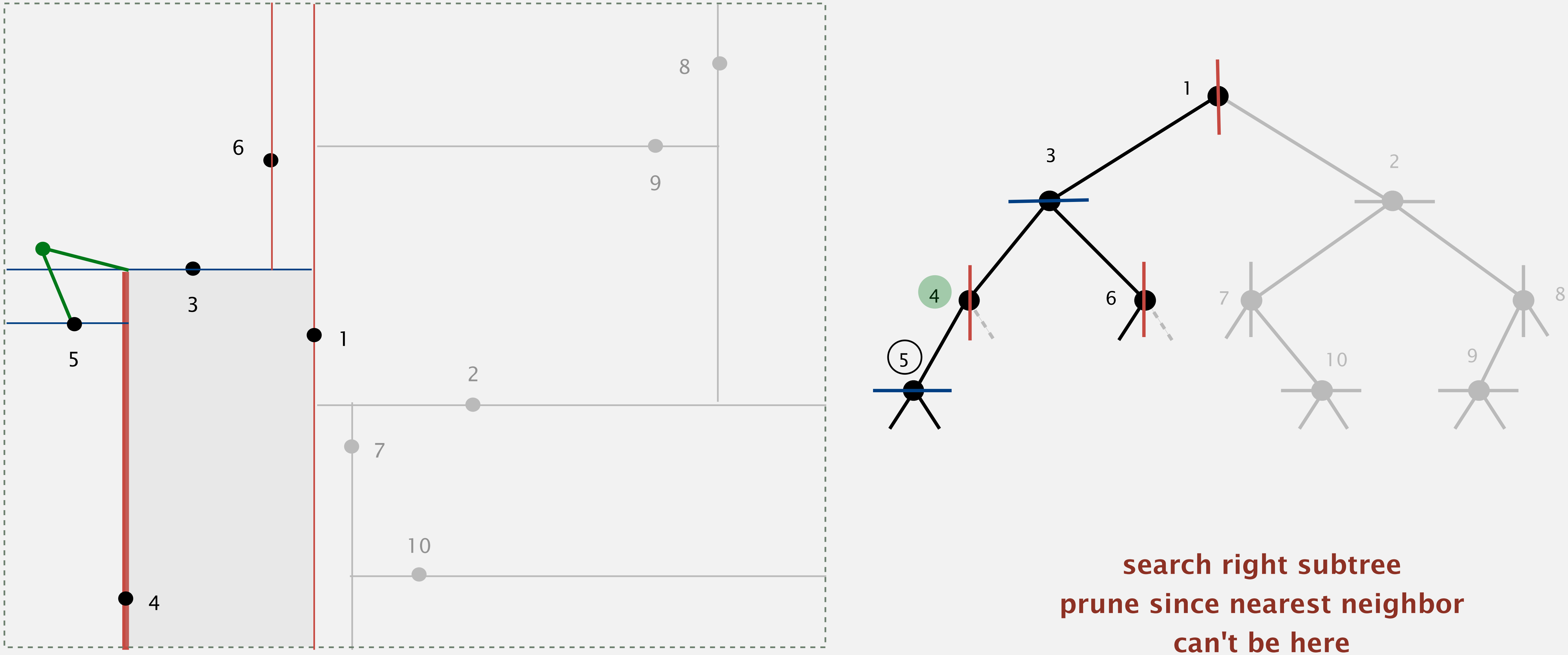
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



return from function call
search right subtree next

2d tree demo: nearest neighbor

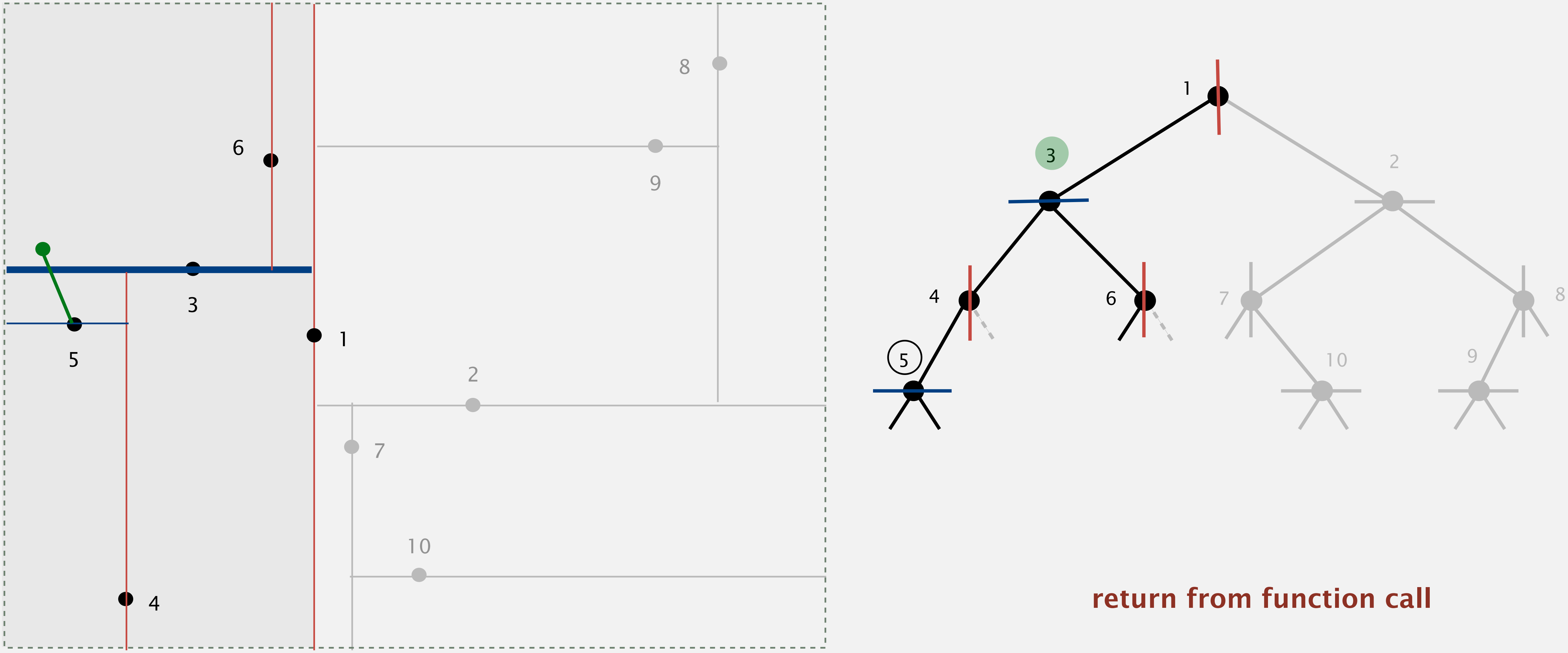
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



search right subtree
prune since nearest neighbor
can't be here
(drawing not quite to scale)

2d tree demo: nearest neighbor

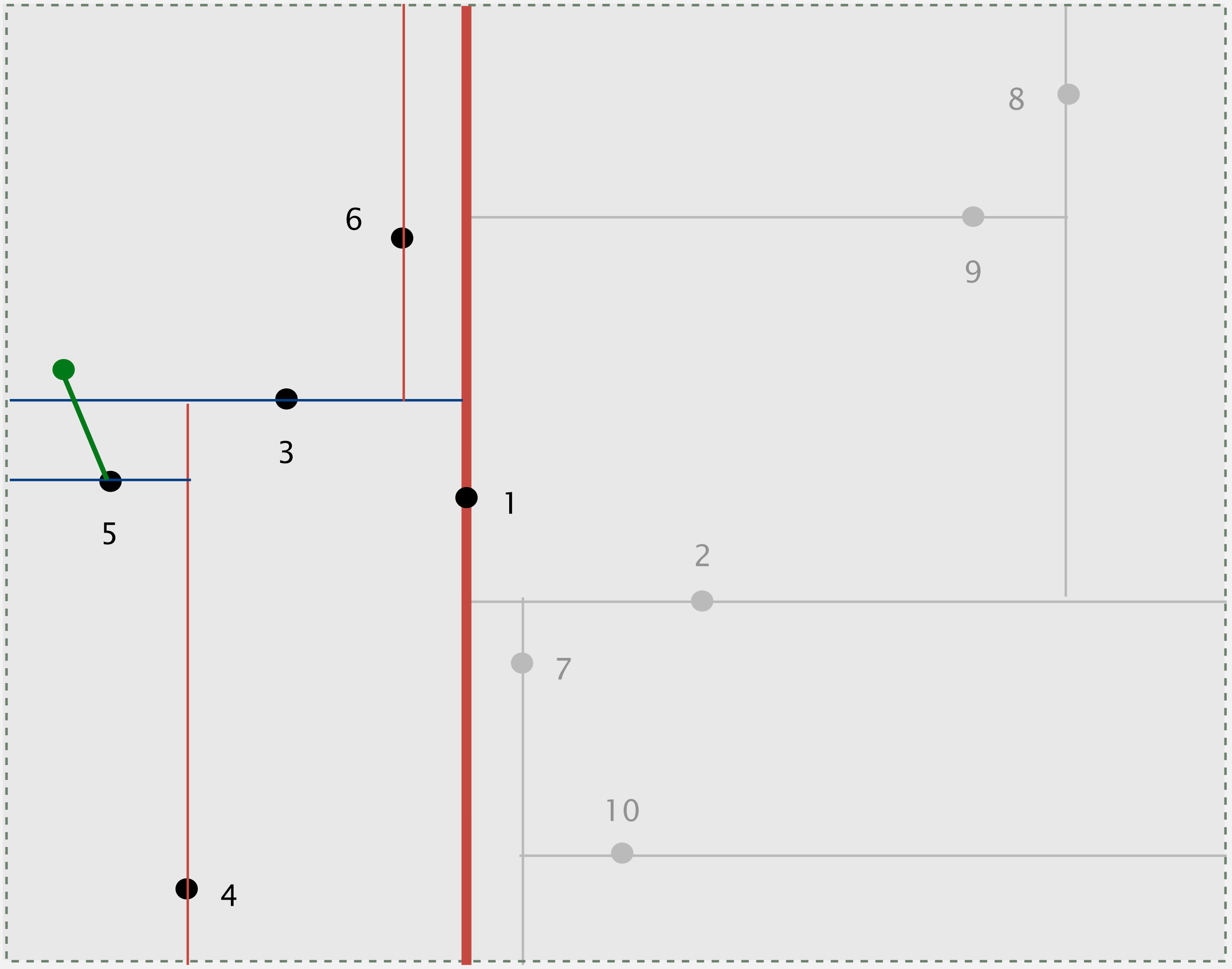
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

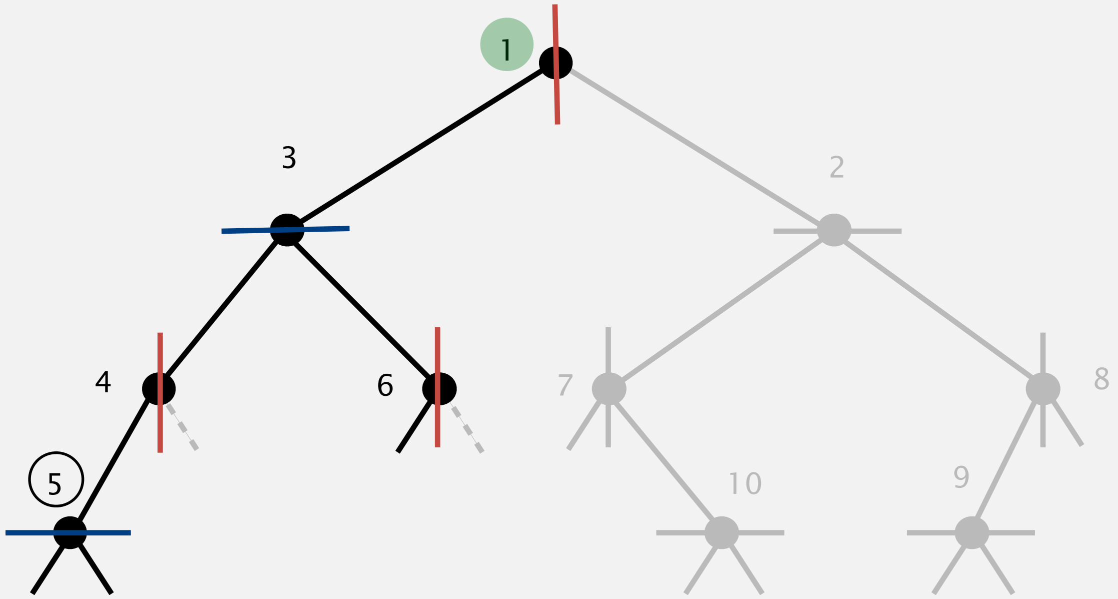


return from function call

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

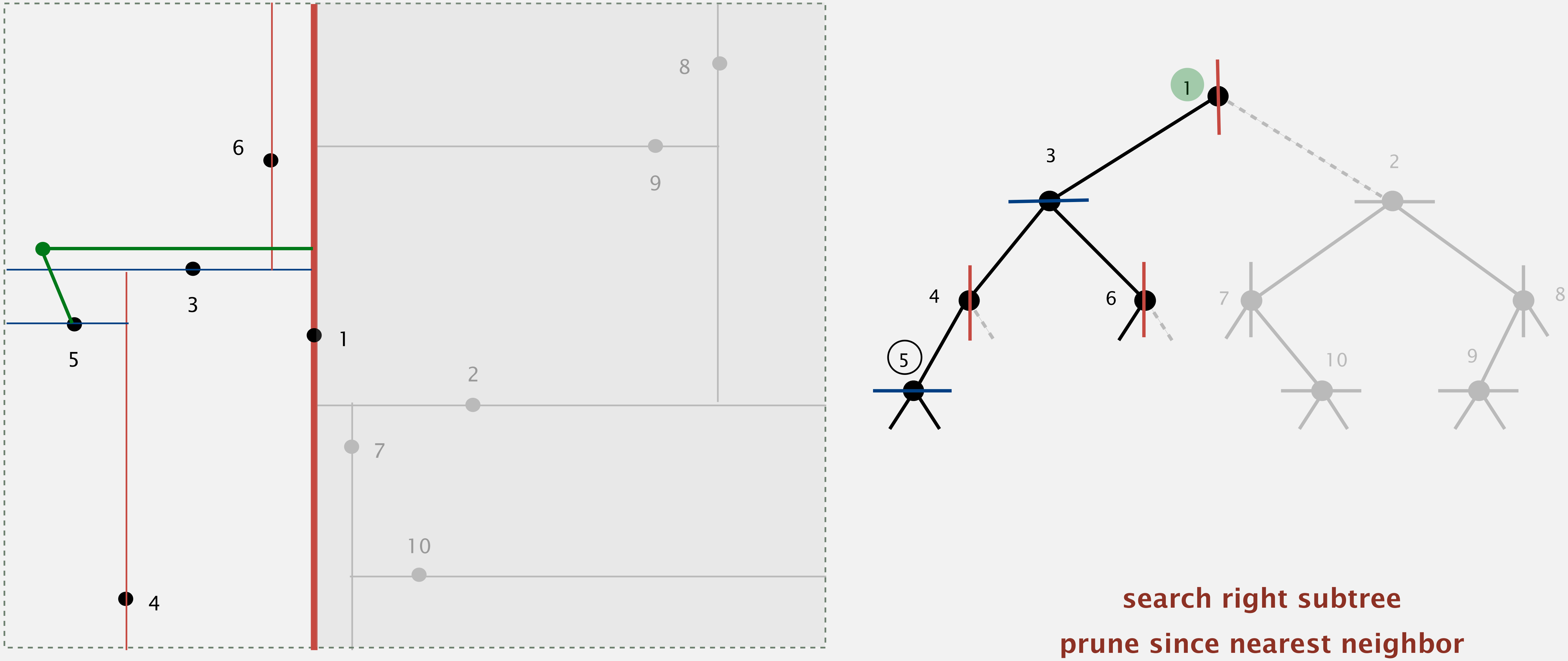




return from function call
search right subtree next

2d tree demo: nearest neighbor

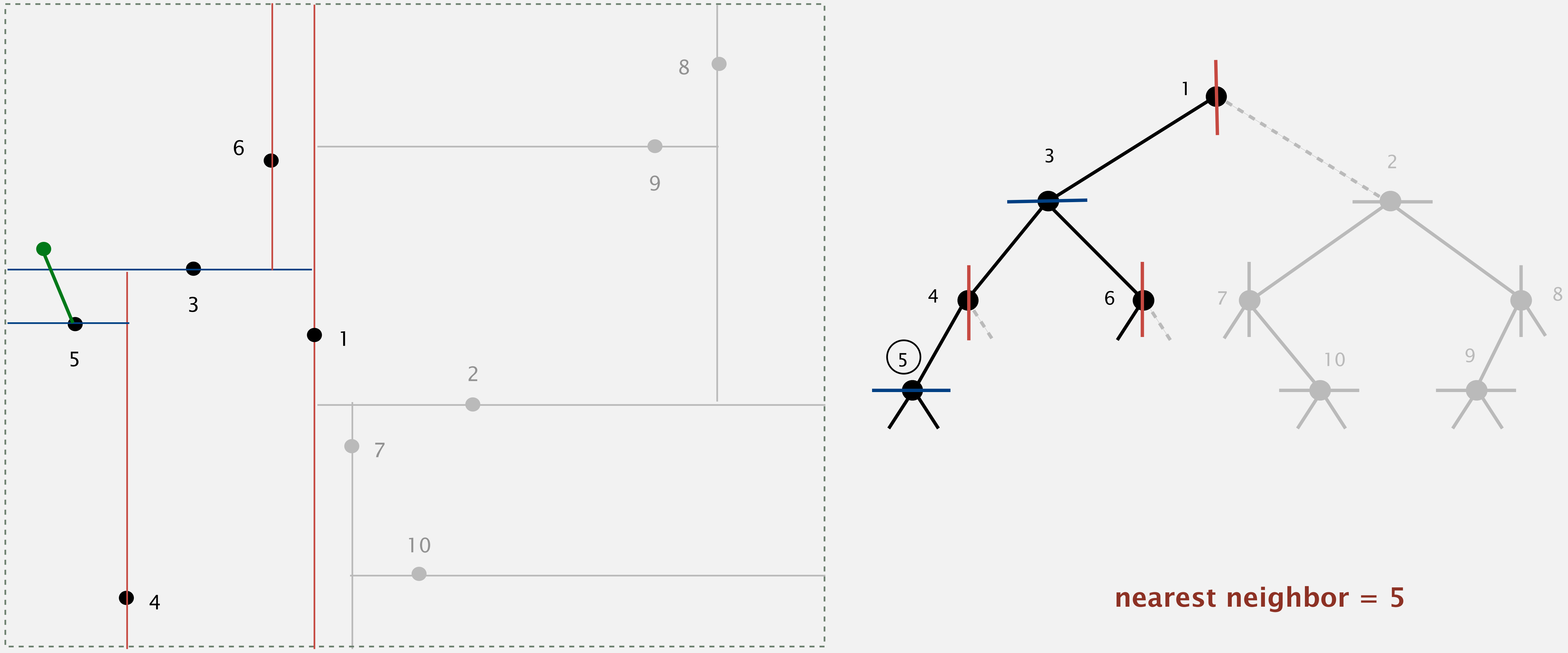
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



search right subtree
prune since nearest neighbor
can't be here

2d tree demo: nearest neighbor

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.

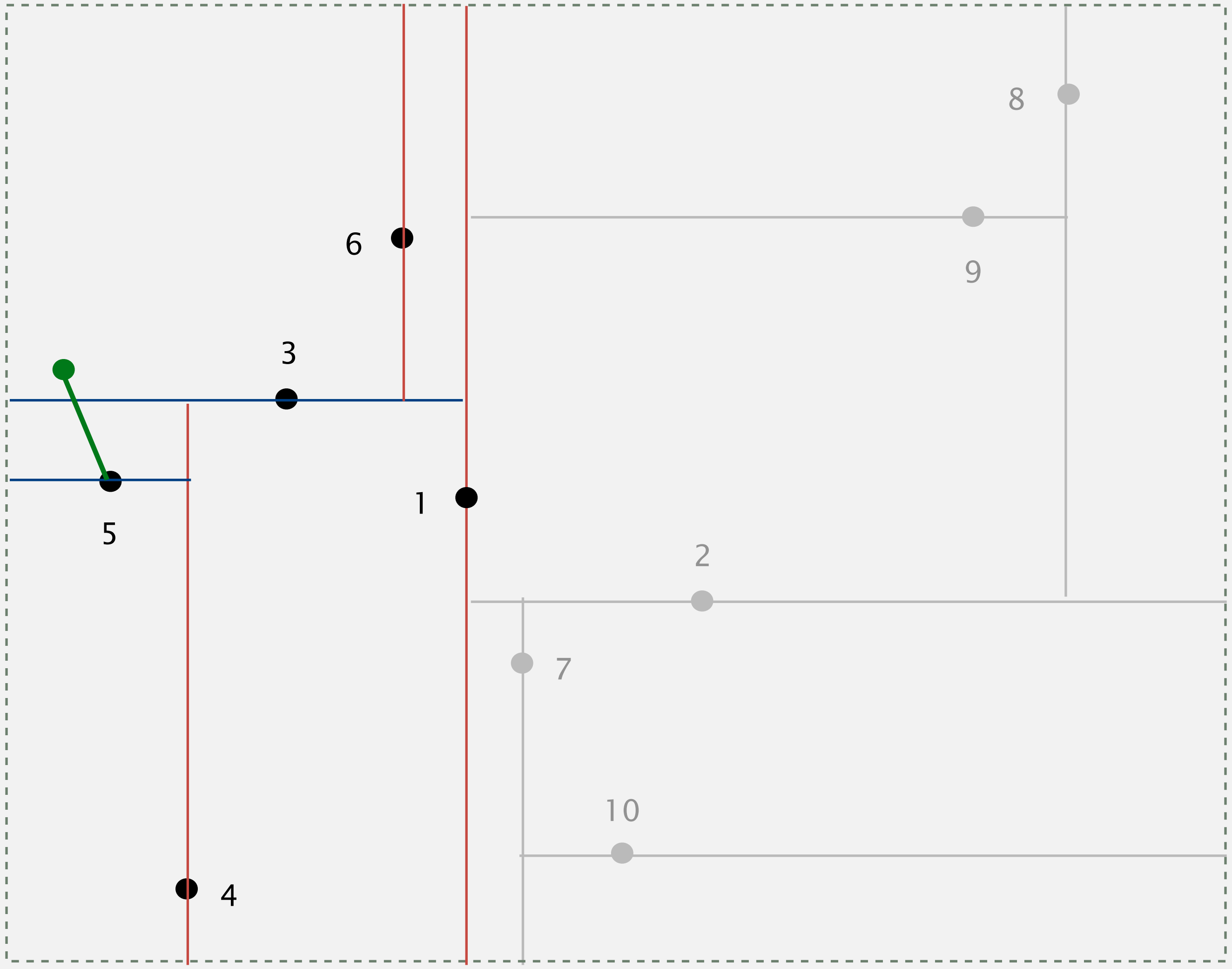


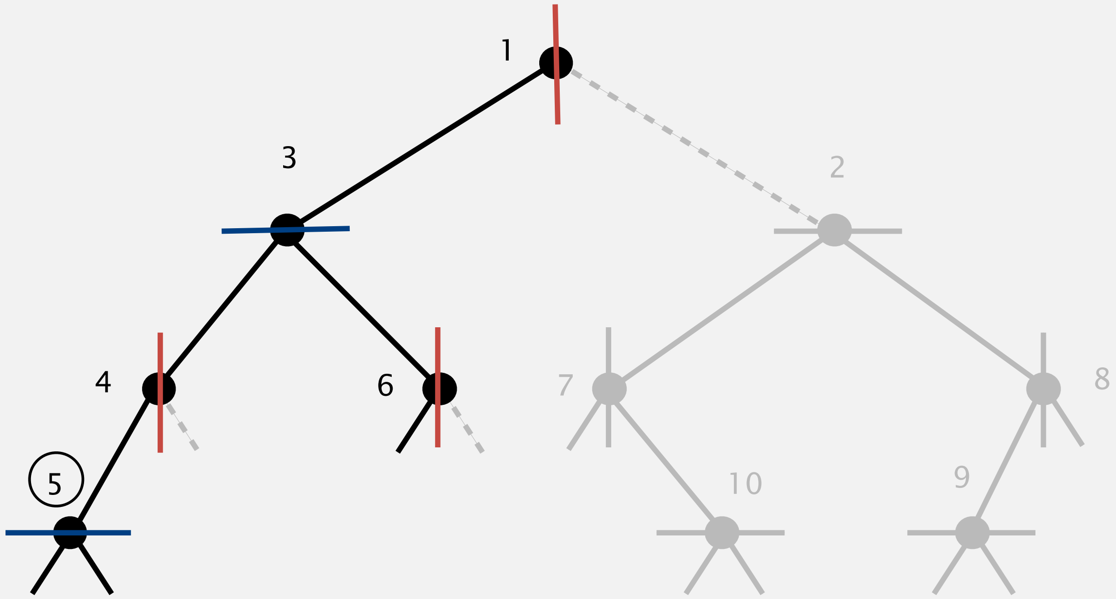
nearest neighbor = 5

Nearest neighbor search in a 2d tree analysis

Typical case. $\log N$.

Worst case (even if tree is balanced). N.





nearest neighbor = 5

Typical memory usage for primitive types and arrays

type	bytes
`boolean`	1
`byte`	1
`char`	2
`int`	4
`float`	4
`long`	8
`double`	8

primitive types

type	bytes
`char[]`	$2N + 16$
`int[]`	$4N + 16$
`double[]`	$8N + 16$

one-dimensional arrays

type	bytes
`char[][]`	$\sim 2\,M N$
`int[][]`	$\sim 4\,M N$
`double[][]`	$\sim 8\,M N$

two-dimensional arrays

Typical memory usage for objects in Java

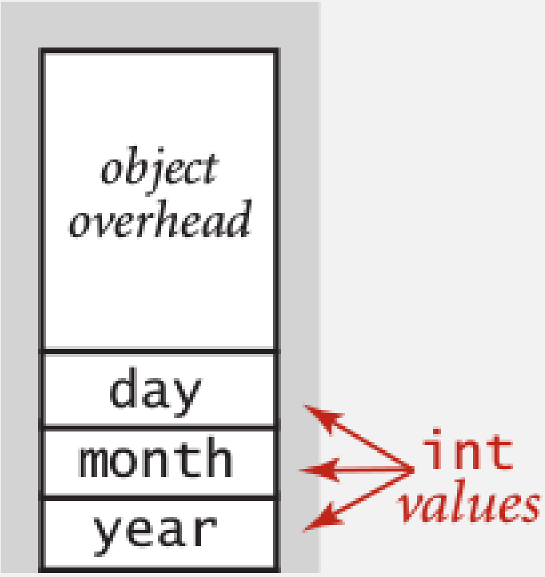
Object overhead. 12 bytes.

Reference. 4 bytes, due to CompressedOops being standard

Padding. Each object uses a multiple of 8 bytes.

Ex 1. A Date object uses 32 bytes of memory.

```
public class Date
{
   private int day;
   private int month;
   private int year;
...
}
```



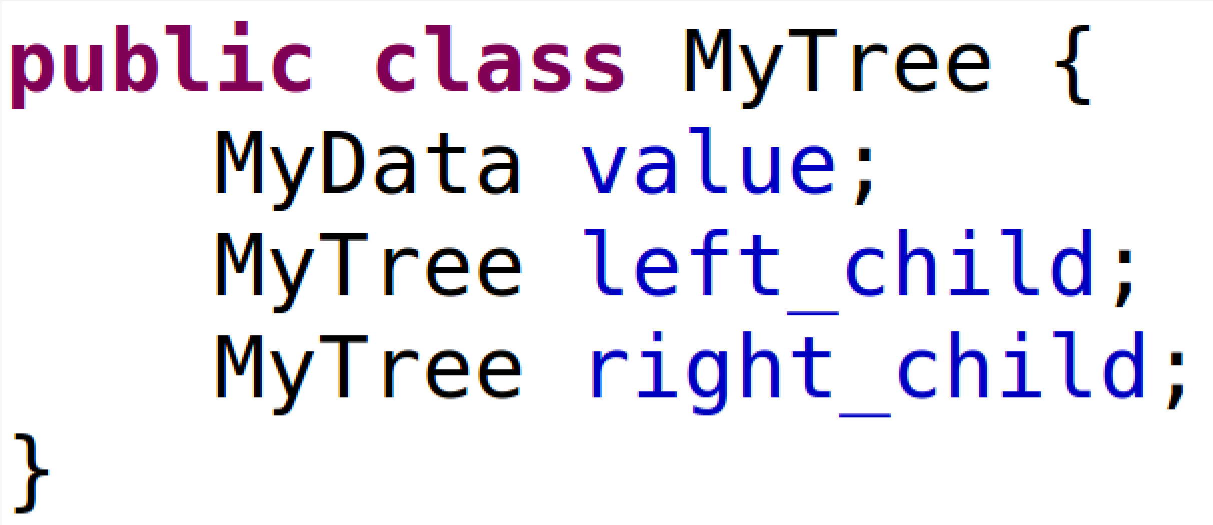
12 bytes (object overhead)

4 bytes (int)
4 bytes (int)
4 bytes (int)

24 bytes

Practical considerations

- Only do recursive subdivision, until some cut-off (e.g. 1000 points)



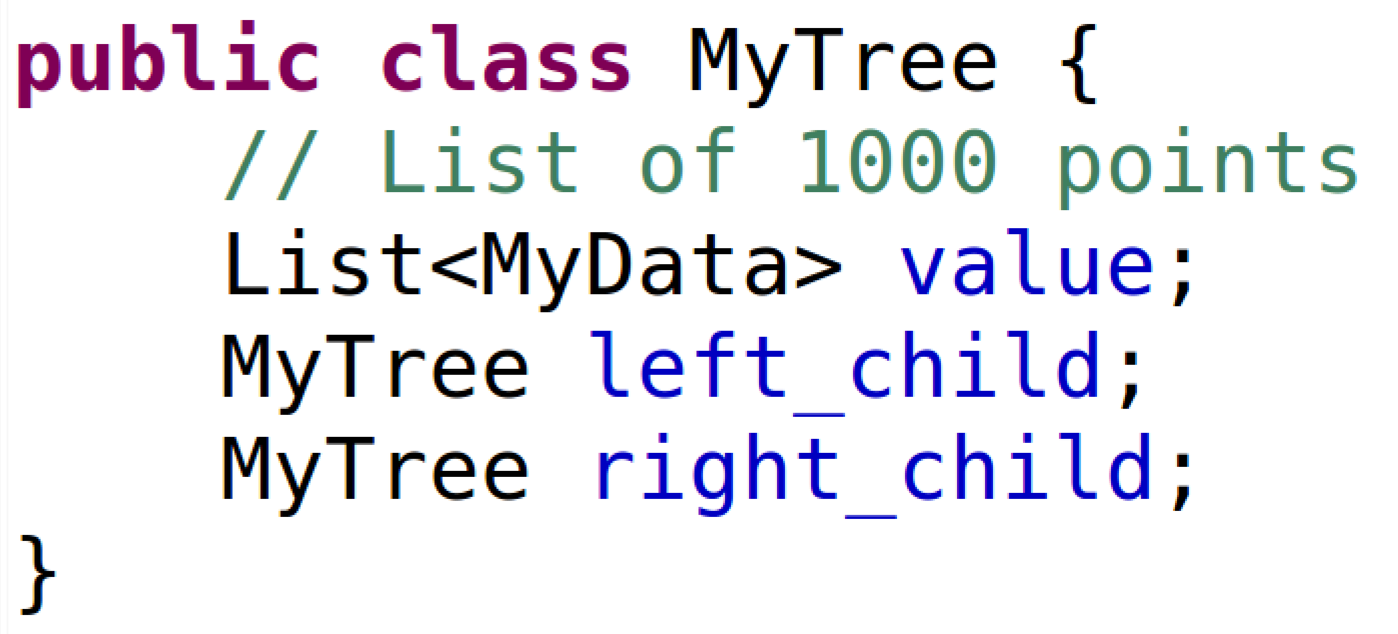
```
public class MyTree {
    MyData value;
    MyTree left_child;
    MyTree right_child;
}
```

For every element:

- Object header: 12 bytes
- 3 references: 12 bytes

Total: $24N$ bytes for data structure

- 1/6 of the space and faster in most cases



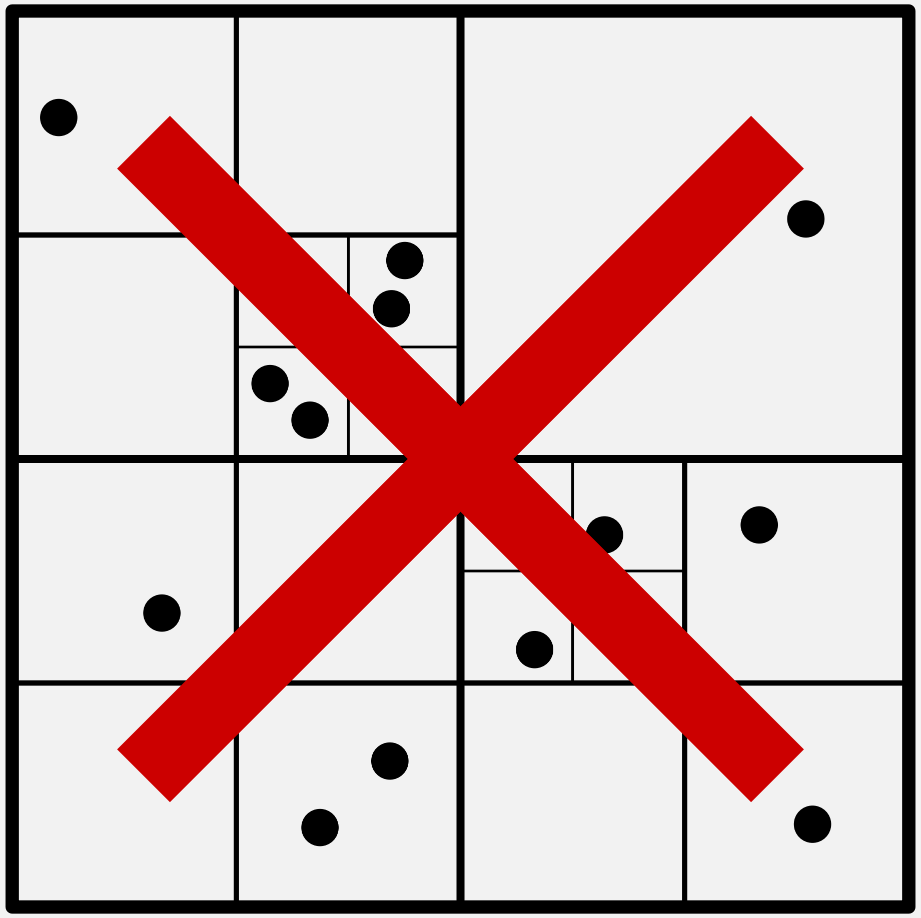
```
public class MyTree {
    // List of 1000 points
    List<MyData> value;
    MyTree left_child;
    MyTree right_child;
}
```

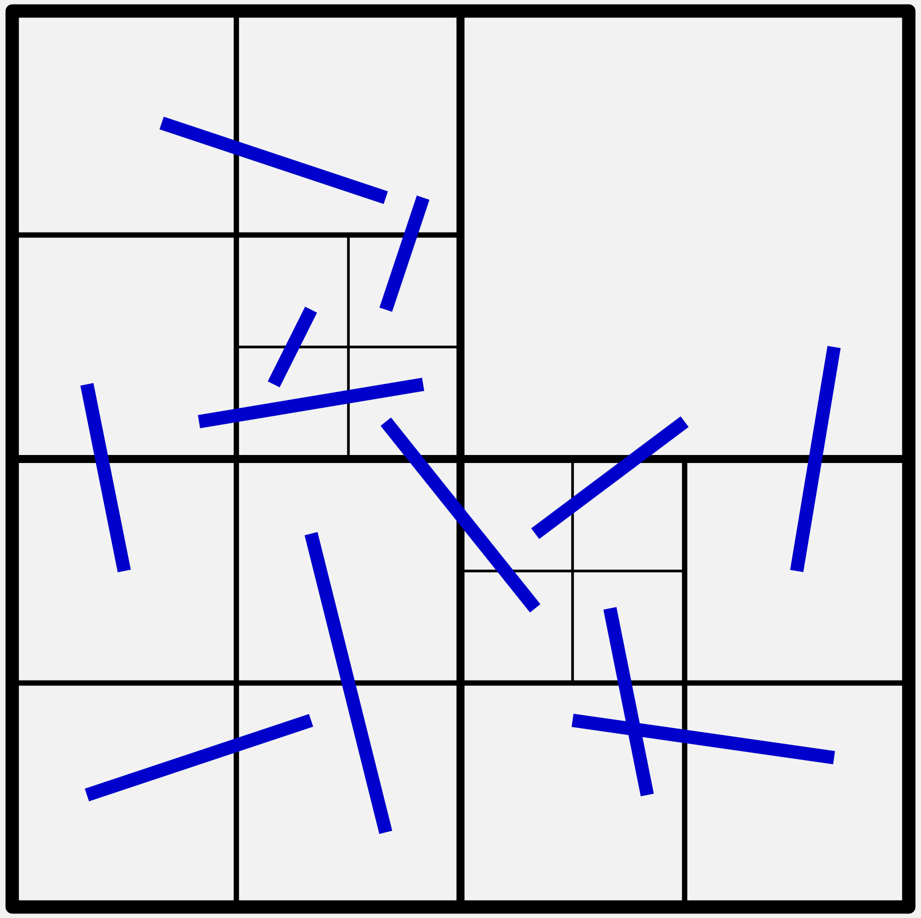
For every 1000 elements:

- Object header: 12 bytes
- 3 references: 12 bytes
- ArrayList: 24 bytes
- MyData array: 4016 bytes

Total: $\sim 4N$ bytes for data structure

Lines are not points





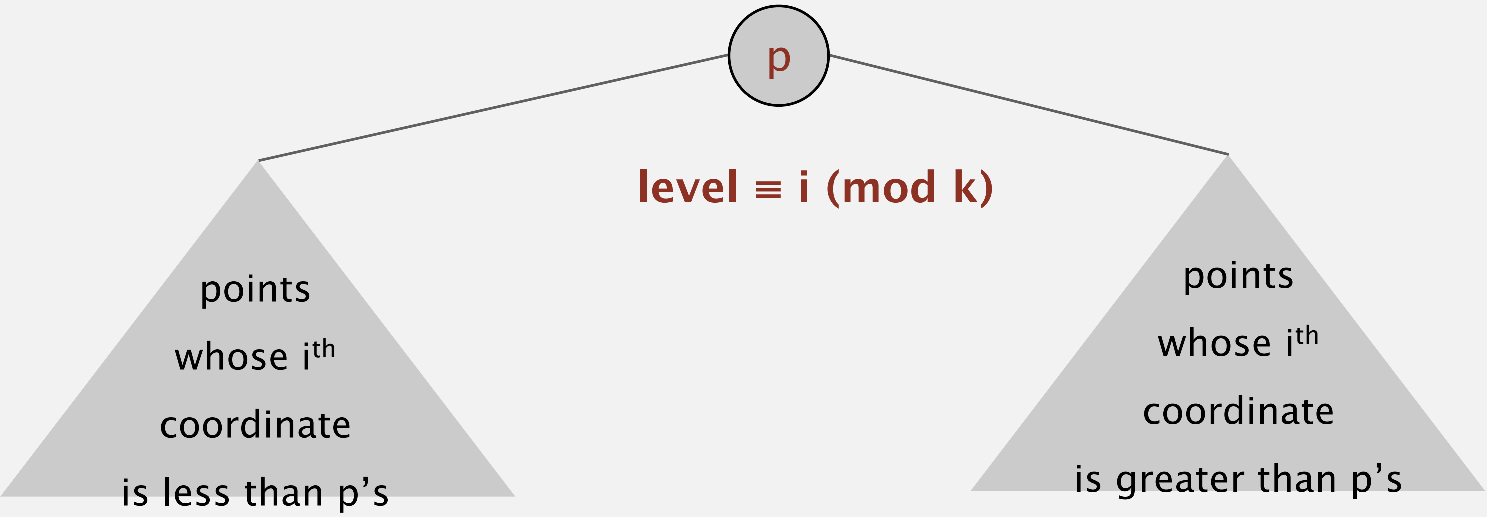
Different solutions to choose from:

- Store elements in all intersected cells
  - Pro: Range search is simpler (check cells intersecting region)
  - Con: Extra space and book keeping for duplicates
- Store elements in a representative cell
  - Pro: Guaranteed linear space)
  - Con: Range search needs to look beyond range

Kd tree

Kd tree. Recursively partition *k*-dimensional space into 2 halfspaces.

Implementation. BST, but cycle through dimensions ala 2d trees.

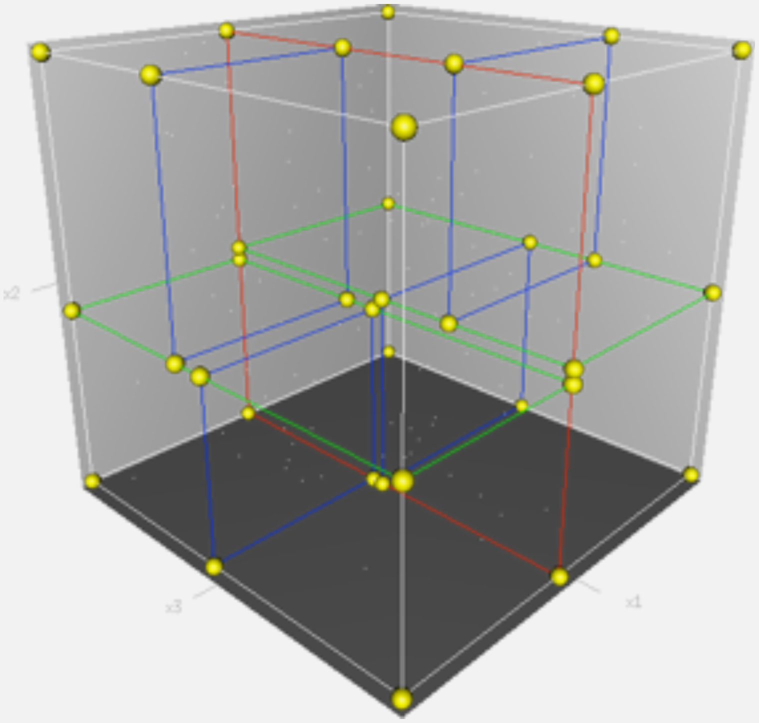


p

level ≡ i (mod k)

points whose i^th coordinate is less than p's

points whose i^th coordinate is greater than p's



Efficient, simple data structure for processing *k*-dimensional data.

- Widely used.
- Adapts well to high-dimensional and clustered data.
- Discovered by an undergrad in an algorithms class!

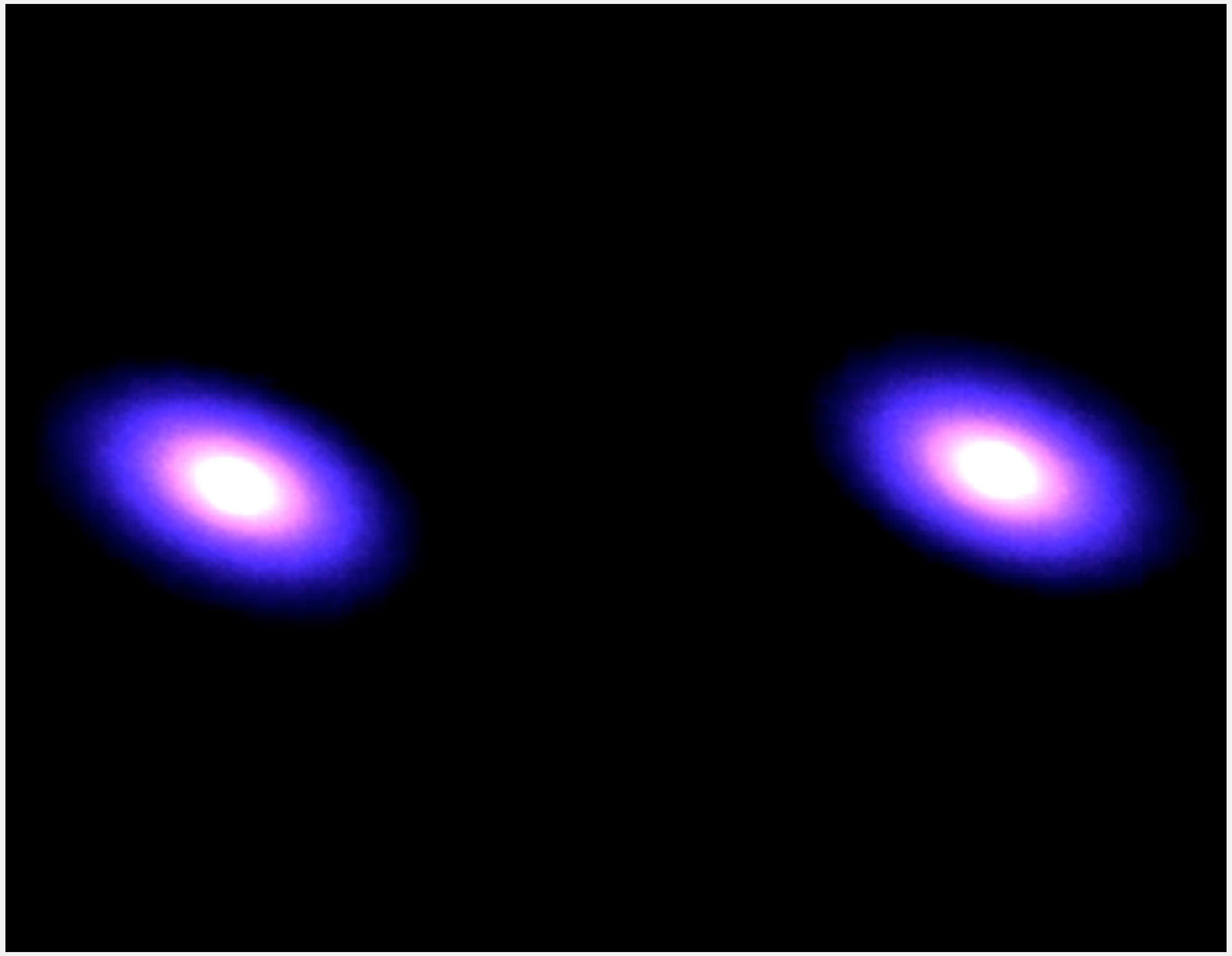
Jon Bentley

N-body simulation

Goal. Simulate the motion of N particles, mutually affected by gravity.

Brute force. For each pair of particles, compute force: $F = \frac{G\, m_1\, m_2}{r^2}$

Running time. Time per step is N^2.



http://www.youtube.com/watch?v=ua7YlN4eL_w

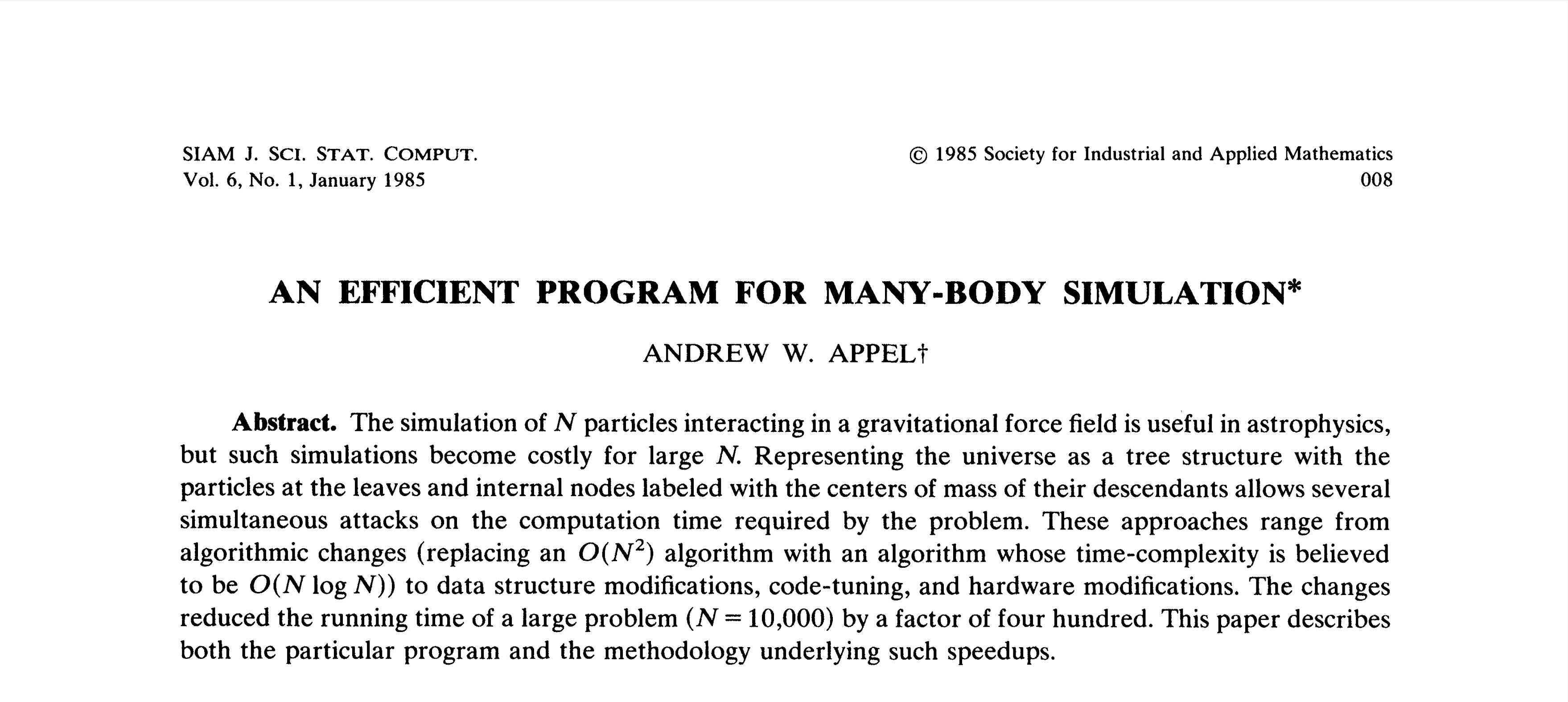
Appel's algorithm for N-body simulation

Key idea. Suppose particle is far, far away from cluster of particles.

- Treat cluster of particles as a single aggregate particle.
- Compute force between particle and center of mass of aggregate.

Appel's algorithm for N-body simulation

- Build 3d-tree with N particles as nodes.
- Store center-of-mass of subtree in each node.
- To compute total force acting on a particle, traverse tree, but stop as soon as distance from particle to subdivision is sufficiently large.



SIAM J. SCI. STAT. COMPUT.
Vol. 6, No. 1, January 1985

AN EFFICIENT PROGRAM FOR MANY-BODY SIMULATION*

ANDREW W. APPEL†

Abstract. The simulation of N particles interacting in a gravitational force field is useful in astrophysics, but such simulations become costly for large N. Representing the universe as a tree structure with the particles at the leaves and internal nodes labeled with the centers of mass of their descendants allows several simultaneous attacks on the computation time required by the problem. These approaches range from algorithmic changes (replacing an $O(N^2)$ algorithm with an algorithm whose time-complexity is believed to be $O(N \log N)$) to data structure modifications, code-tuning, and hardware modifications. The changes reduced the running time of a large problem ($N = 10{,}000$) by a factor of four hundred. This paper describes both the particular program and the methodology underlying such speedups.

Impact. Running time per step is $N \log N$ ⇒ enables new research.

Summary

Many different data structures to choose from:

- Kd-tree (good practical performance)
- QuadTree (easy to implement)
- RTree (designed for non-point data)
- Range-tree (best theoretical guarantee)