ONE DIMENSIONAL RANGE SEARCH

1d range search

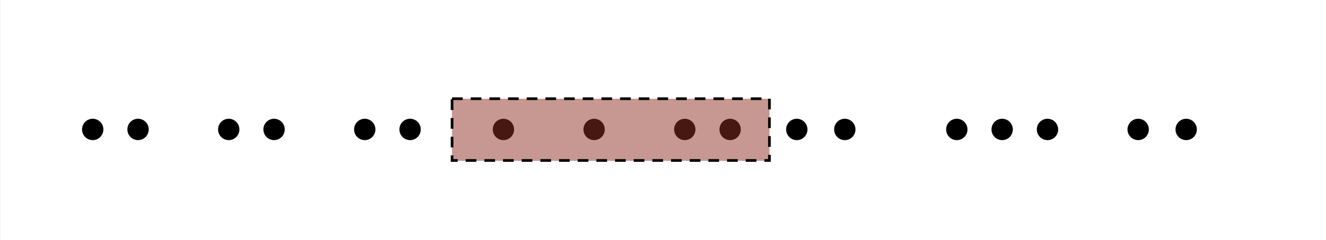
Extension of ordered Binary Search Tree.

- 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.

Geometric interpretation.

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



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

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

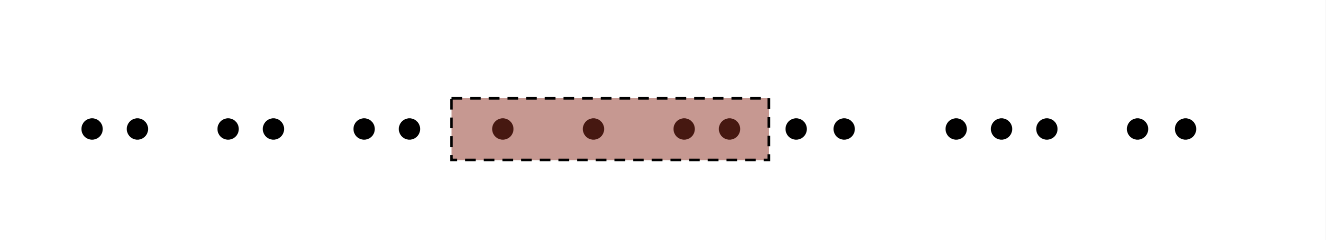
R = number of keys that match

LET'S FOCUS ON RANGE COUNTING

- Range count: number of keys between k_1 and k_2.

1d range count: BST implementation

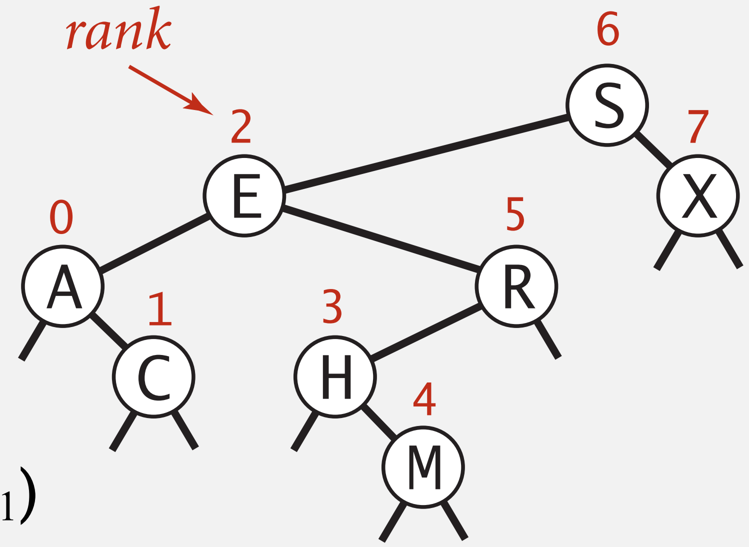
- Question: How do we determine the number of keys between k_1 and k_2



- Step1: Figure how many items less than k_1

Calculate the rank each node in the BST

  - The Rank of a Node: is number of keys that are less than it



- Step2: Subtract the rank(k_2) from rank(k_1)

HOW WOULD WE DO SEARCH

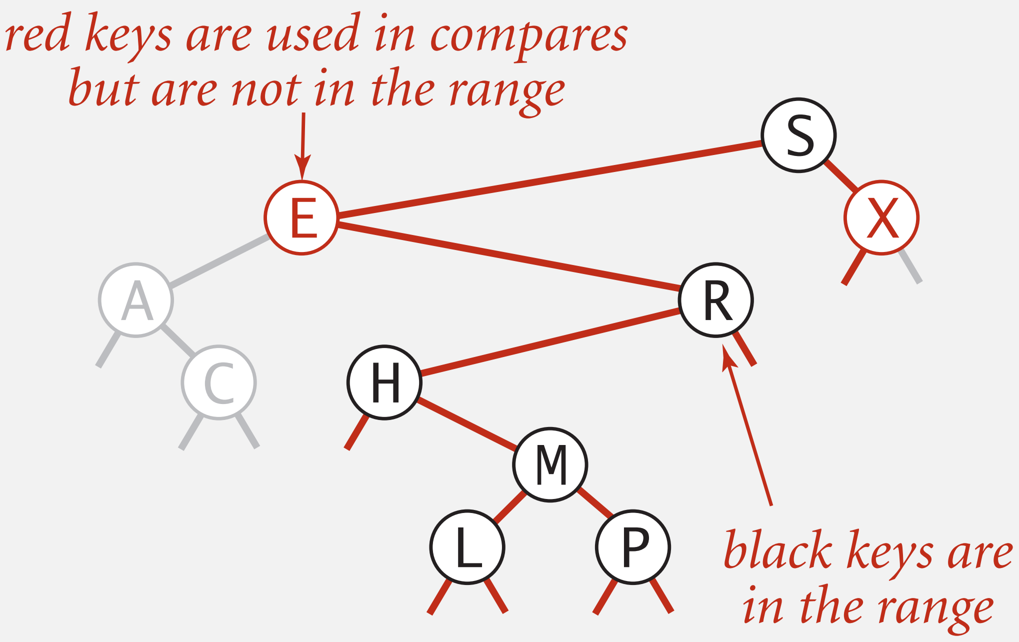
- Range search: find all keys between k_1 and k_2.

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 $R + \log N$.

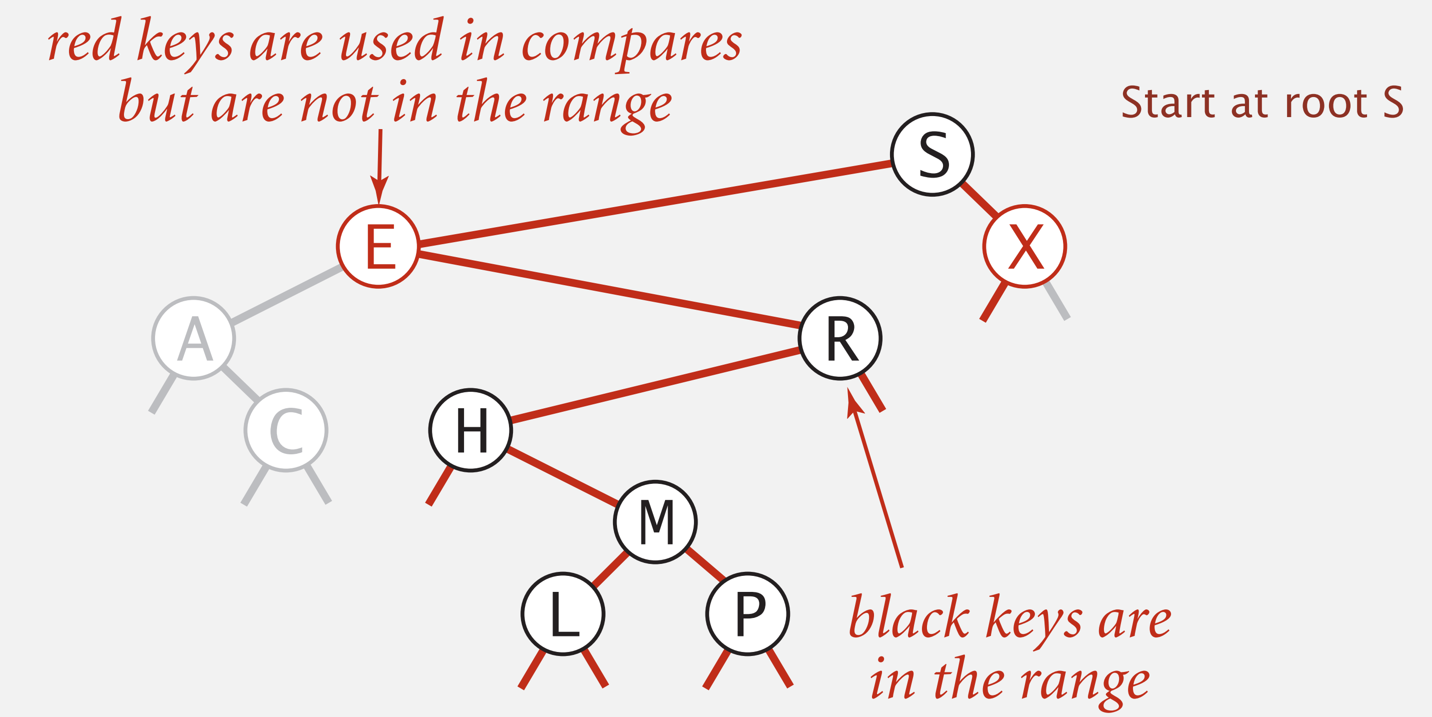
Pf. Nodes examined = search path to `lo` + search path to `hi` + matches.

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 rang

searching in the range [F..T]

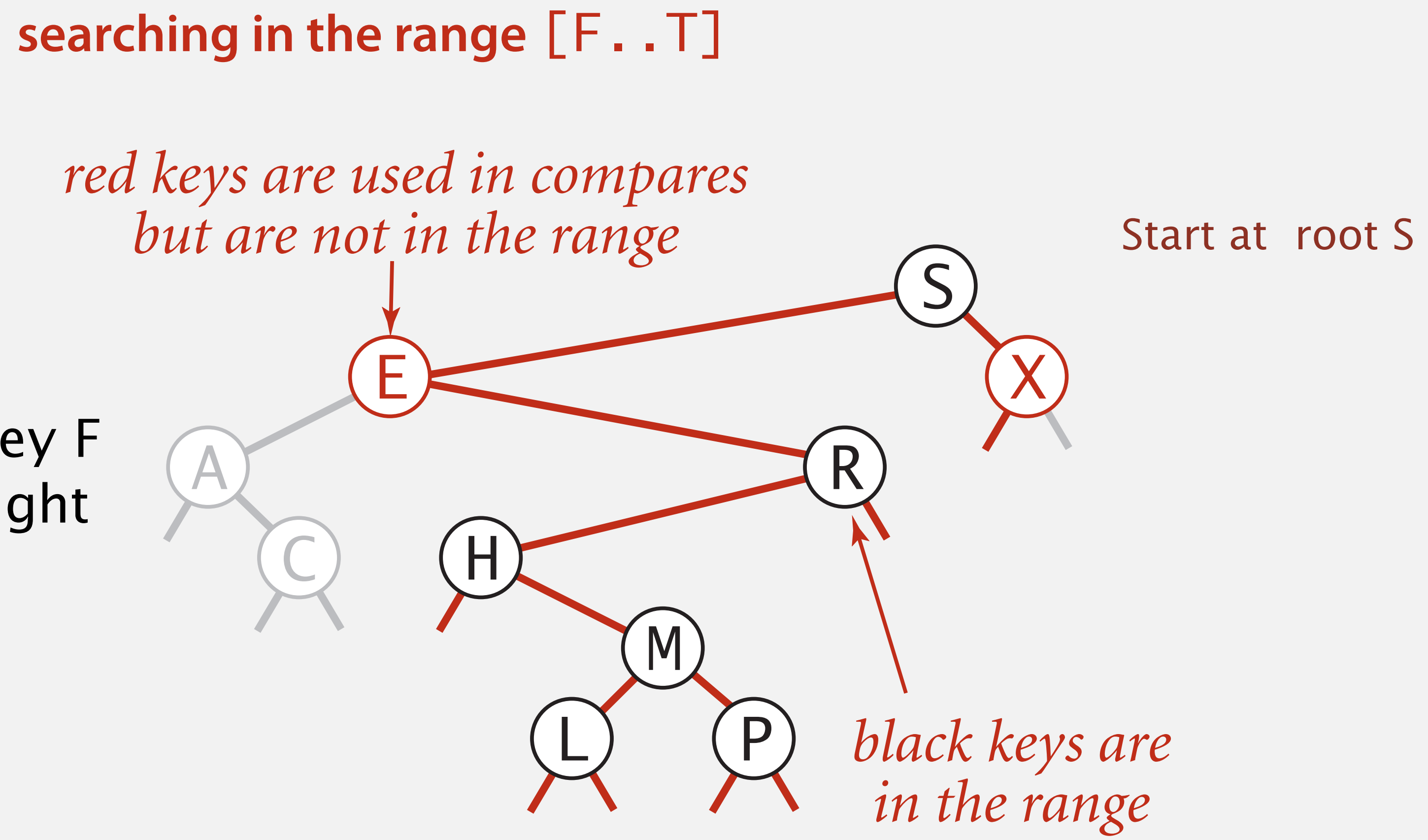


1d range search: BST implementation

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 rang

Check E
1) E is not the range
2) is less than smallest key F
3)Only need to recurse right

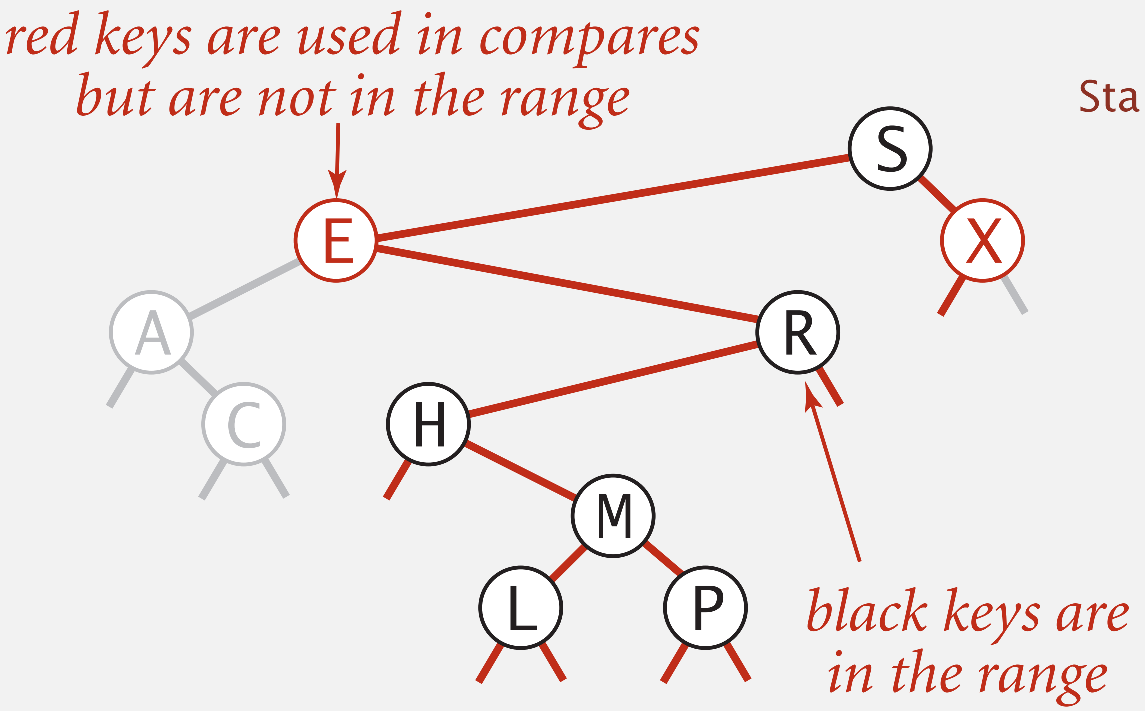

searching in the range [F..T]
red keys are used in compares but are not in the range
Start at root S
S
E
X
A
R
C
H
M
L
P
black keys are in the range

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 rang

searching in the range [F..T]



Start at root S

Check R

1) R is range [F ... T]

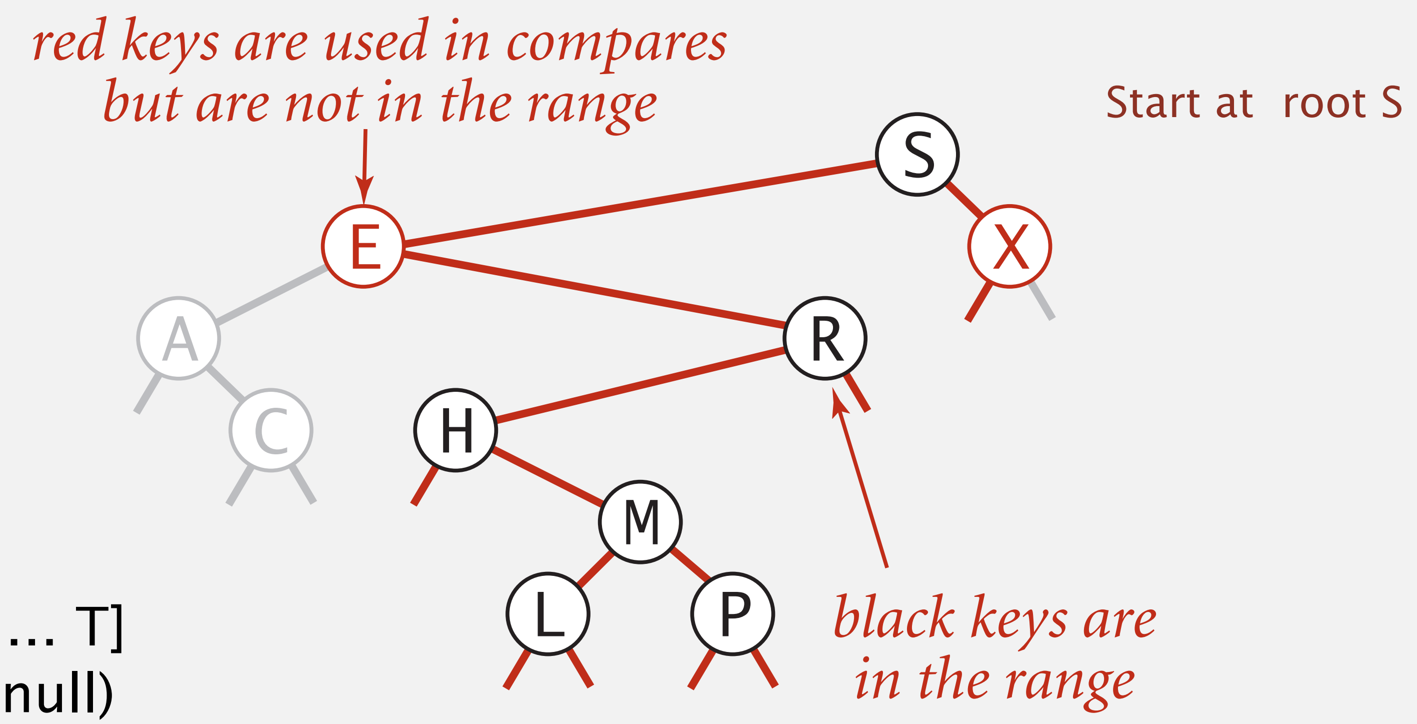
2) Recurse left

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 rang

searching in the range [F..T]



red keys are used in compares but are not in the range
Start at root S
S
E
X
A
R
C
H
M
L
P
black keys are in the range

Check H
1) H is range [F ... T]
2) Recurse left (null)
3) Recurse right

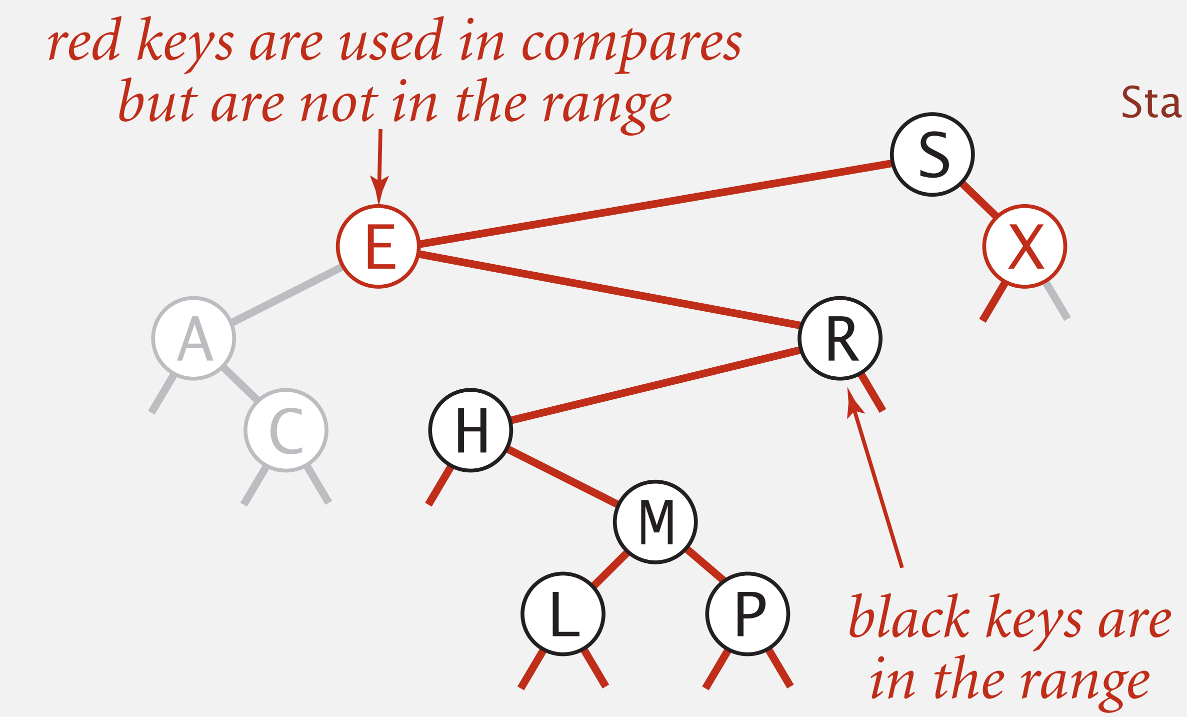
Same process for M, L, P

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 rang

searching in the range [F..T]



Start at root S

Check X

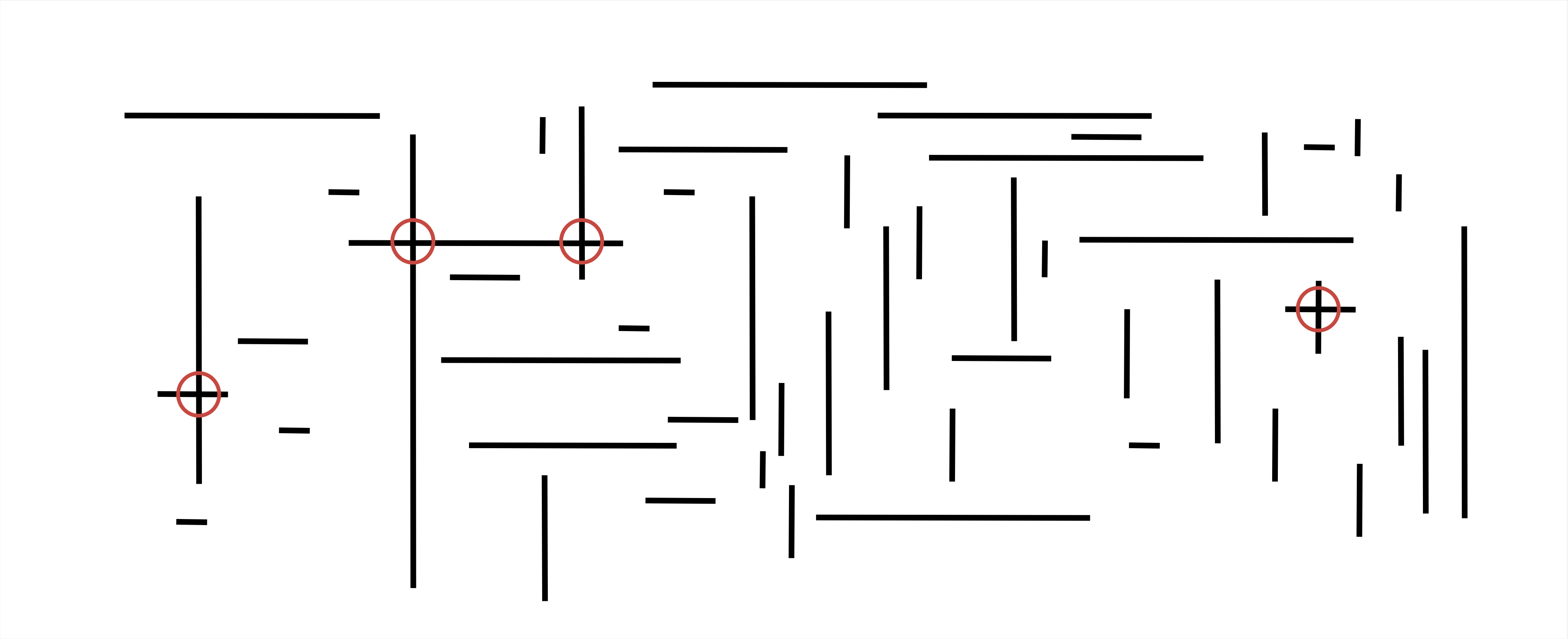
1) X is not range [F ... T]

2) end recursion

LETS EXTEND THIS IDEA TO THE 2D CASE

Orthogonal line segment intersection

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



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

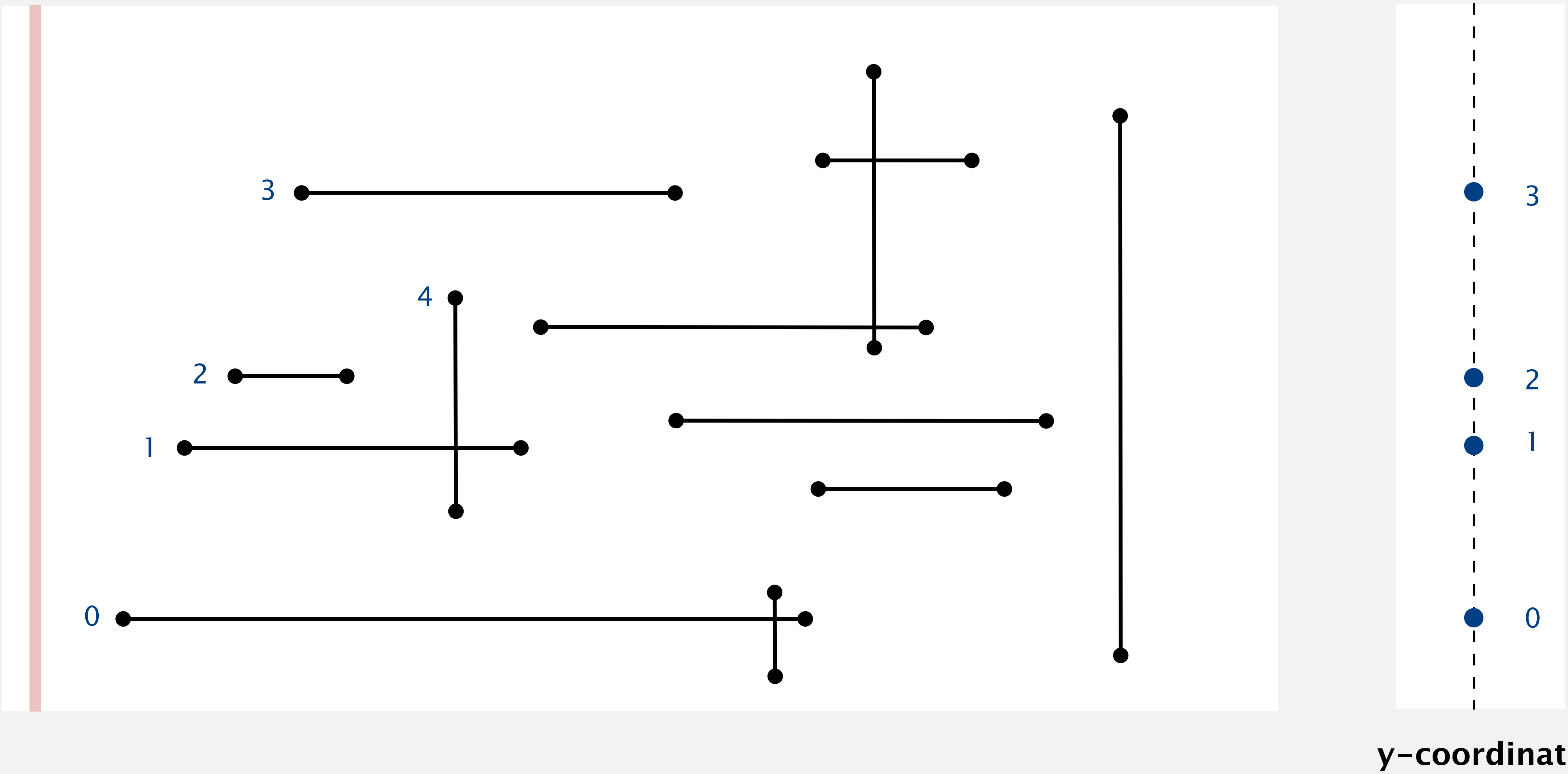
Nondegeneracy assumption. All x- and y-coordinates are distinct.

- Remove all the lines that touch with intersecting.

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

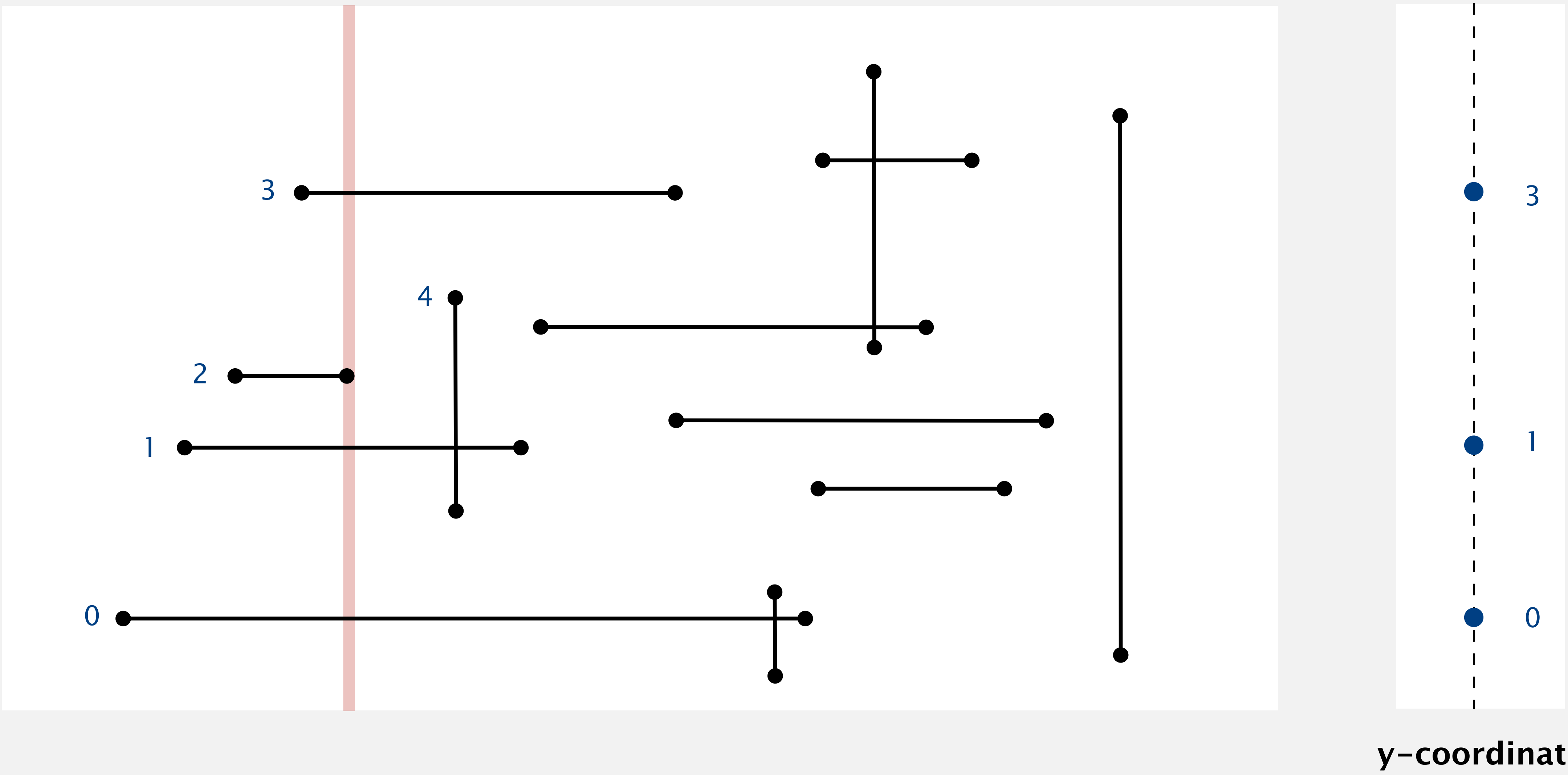
- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.



Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

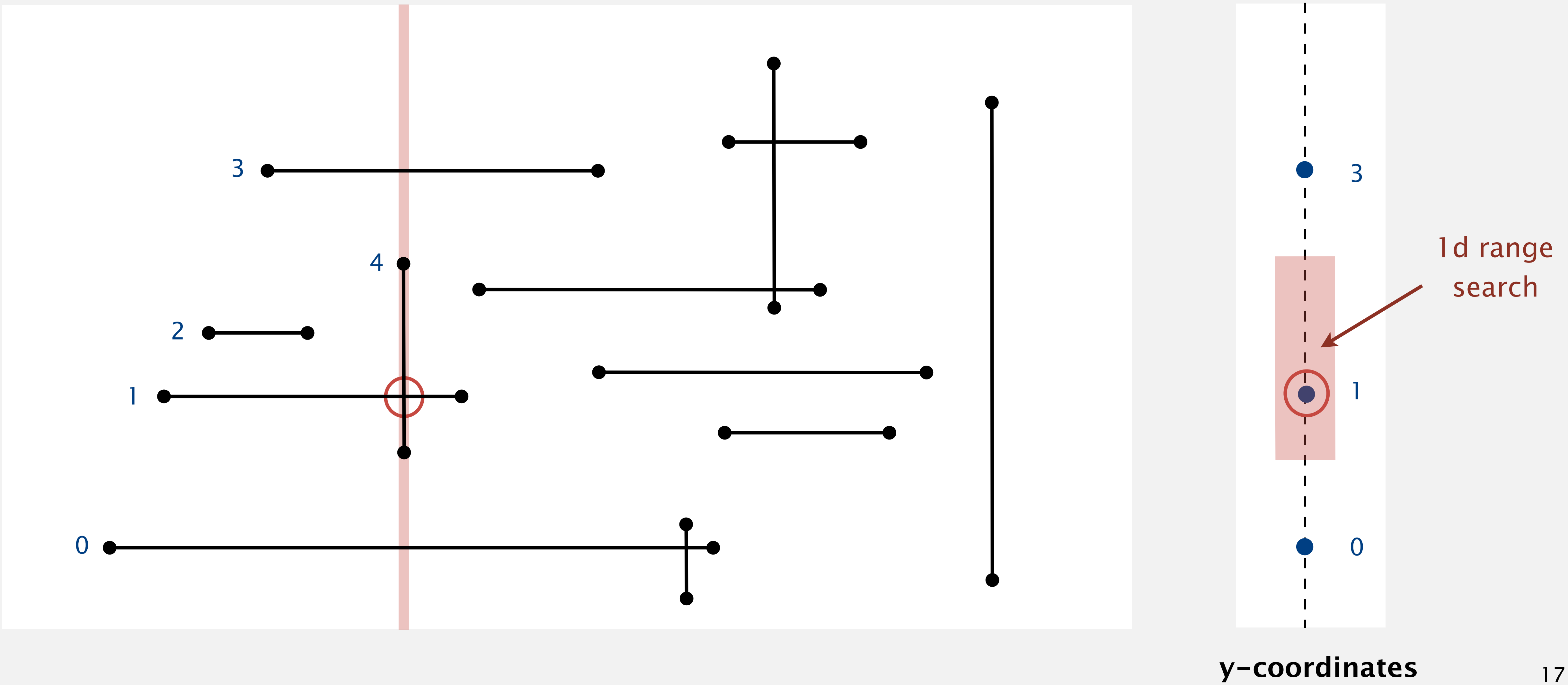
- x-coordinates define events.
- h-segment (left endpoint): insert y-coordinate into BST.
- h-segment (right endpoint): remove y-coordinate from BST.



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.

LETS EXTEND THE ALGORITHM TO 2D SPACE

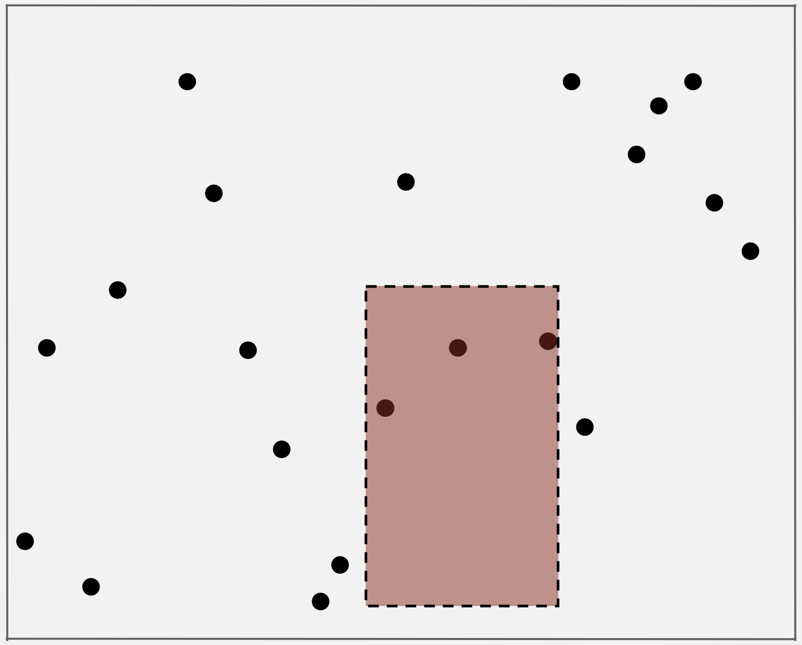
2-d orthogonal range search

Extension to 2d keys.

- Insert a 2d key. (Points)
- 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.

Geometric interpretation.

- Keys are point in the plane.
- Find/count points in a given rectangle



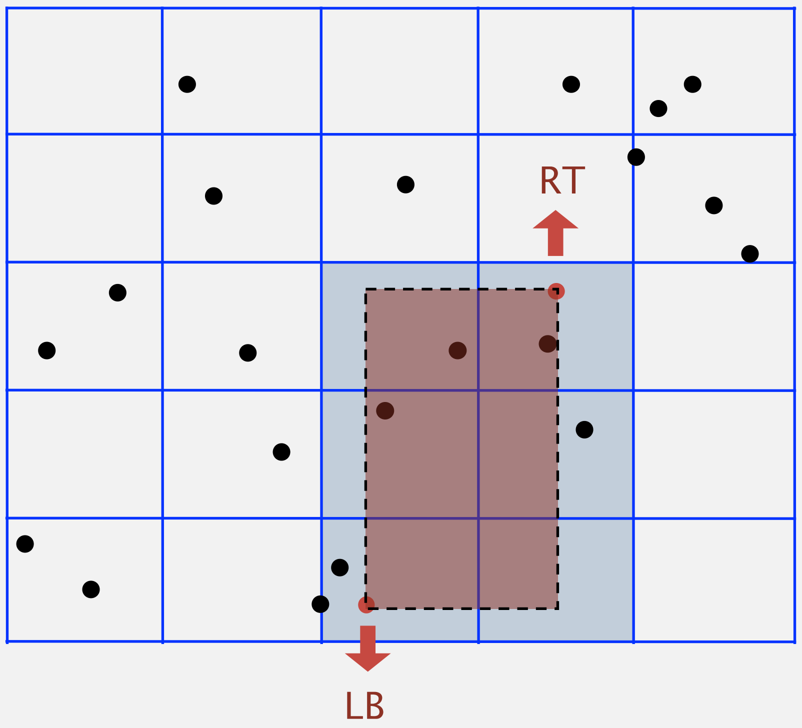
2d orthogonal range search: grid implementation

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.

Cell ID	1	2	MxM
Point ID	5	6	7

- M-by-M grid

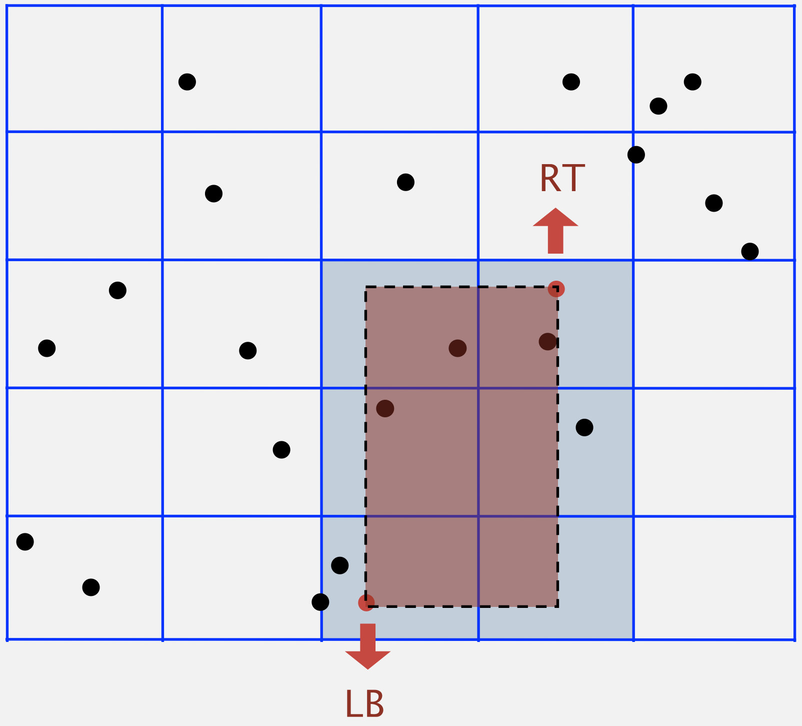


RT = Right Top
LB = Left Bottom

2d orthogonal range search: grid implementation analysis

Choose grid square size to tune performance.

- Too small: wastes space.
  - (Searching square that don't contain anything)
- Too large: too many points per square.
  - (Can't distinguish between points)



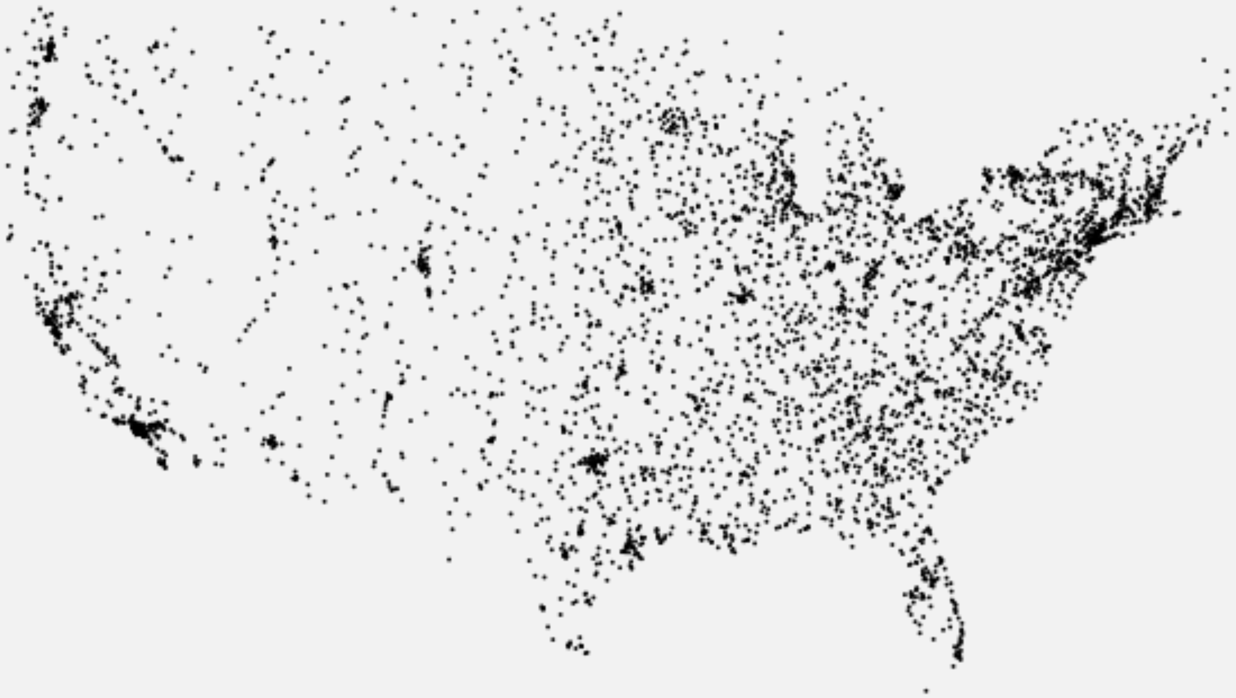
HOWEVER, POINTS ARE ALWAYS EVENING DISTRIBUTED

Clustering

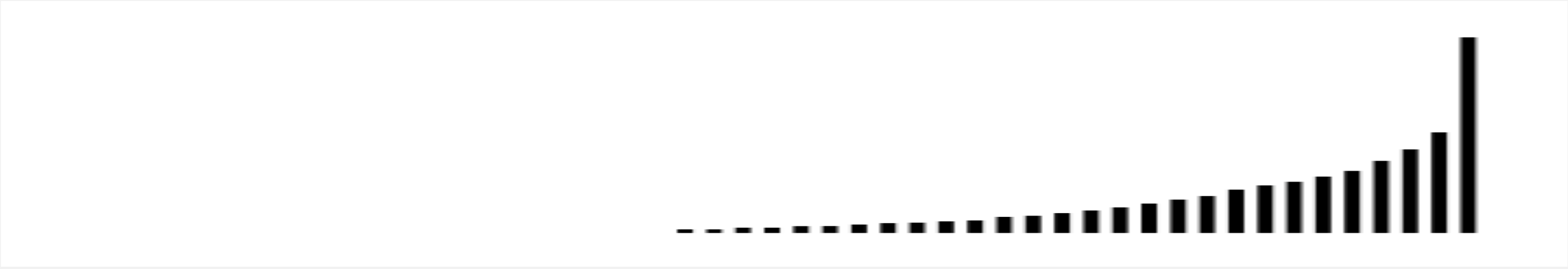
Grid implementation. Fast, simple solution for evenly-distributed points.

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

Ex. USA map data.



13,000 points, 1000 grid squares



half the squares are empty

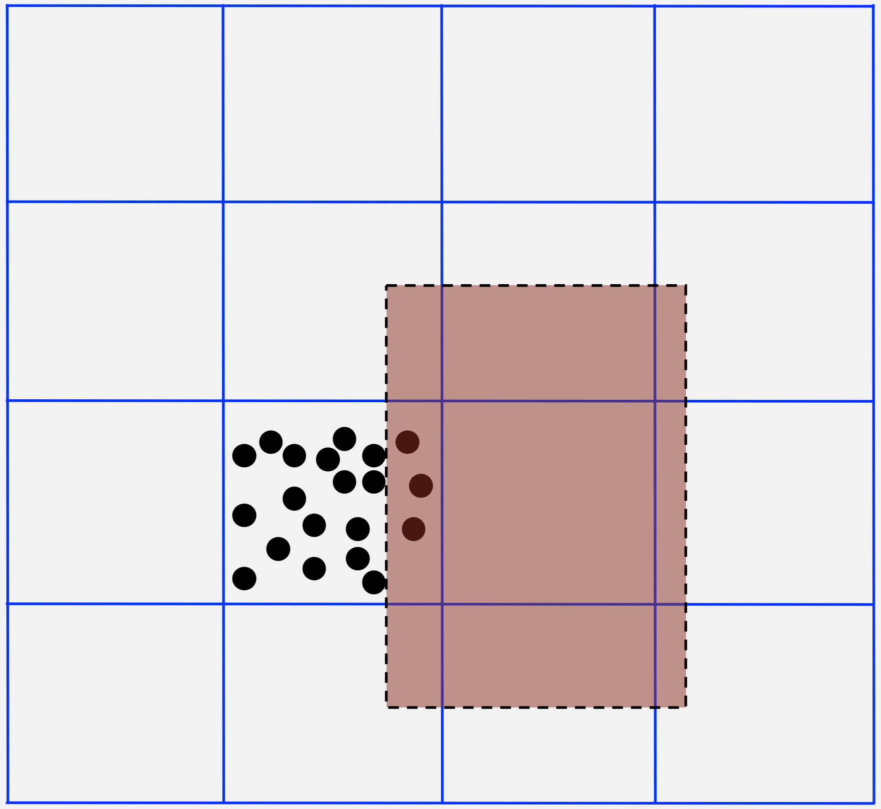
half the points are in 10% of the squares

Clustering

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

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

- Need data structure that adapts gracefully to data.



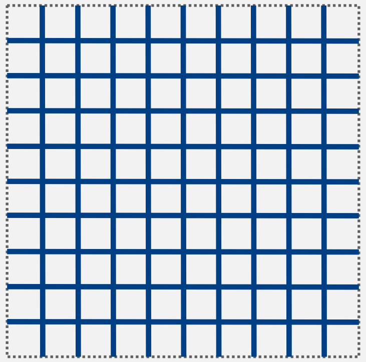
NEED PARTITIONING APPROACH THAT FITS THE DISTRIBUTION OF THE DATA

Space-partitioning trees

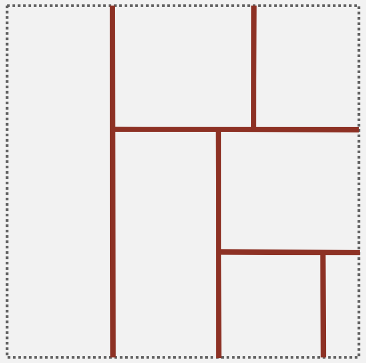
Use a tree to represent a recursive subdivision of 2d space.

Grid. Divide space uniformly into squares.

2d tree. Recursively divide space into two halfplanes.



Grid



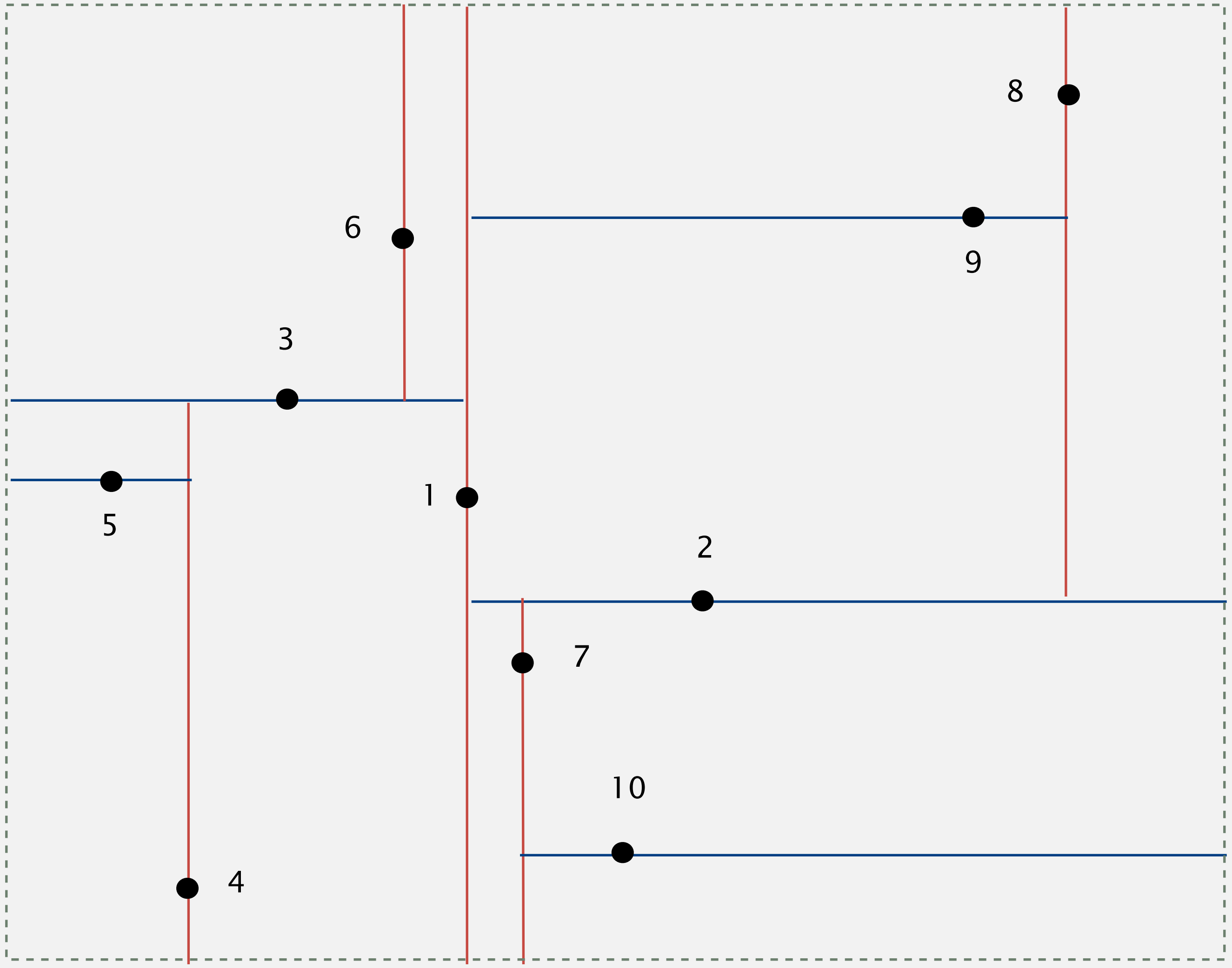
2d tree

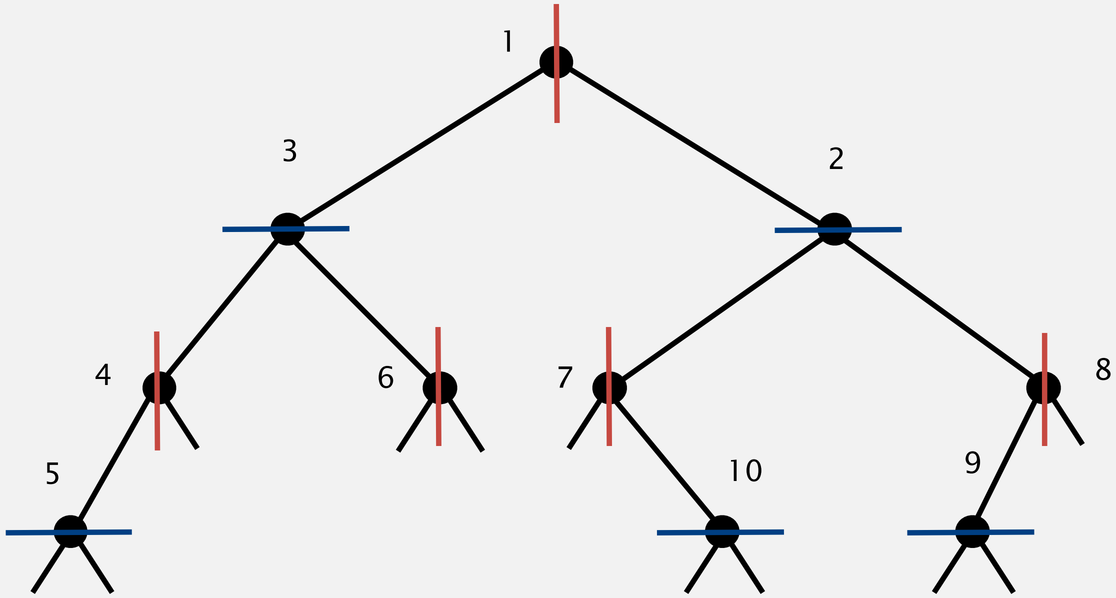
CONSTRUCTING A 2D TREE

2d tree construction

Recursively partition plane into two halfplanes.

Alternative between | And — at each level in the tree

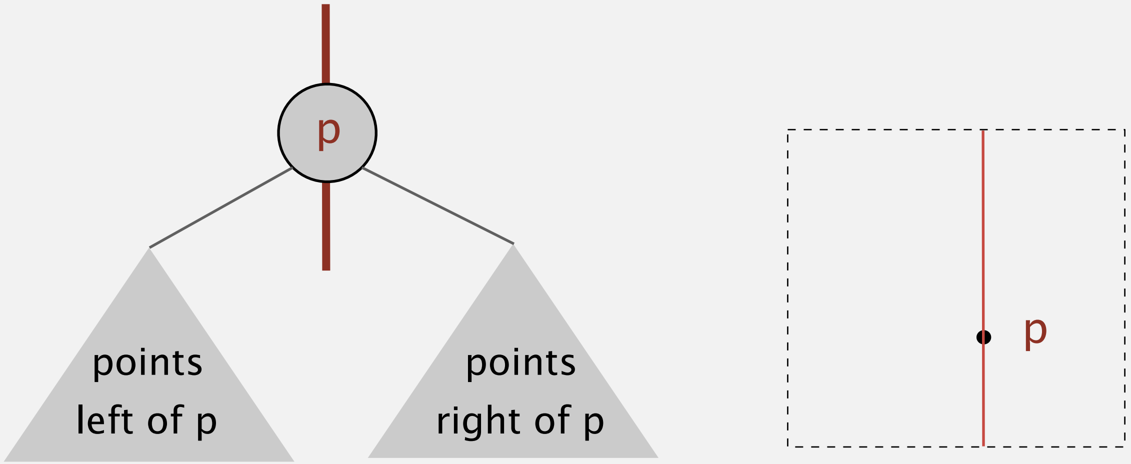




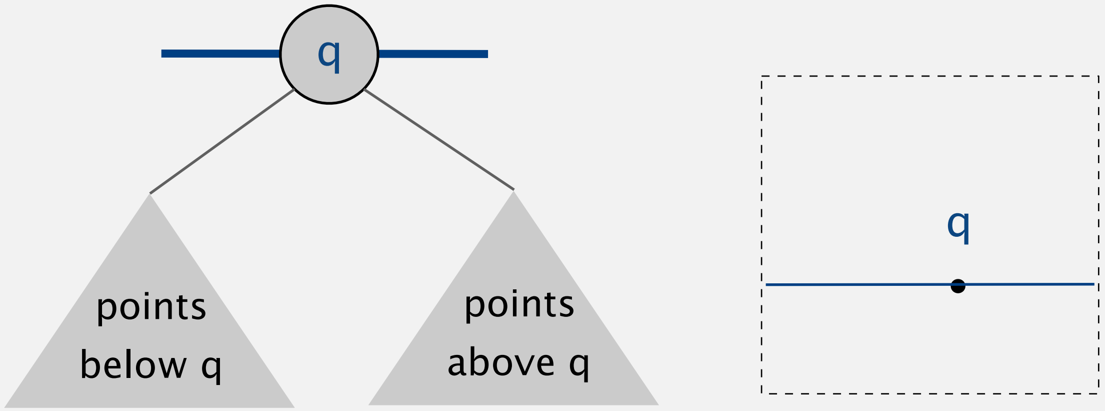
2d tree implementation

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

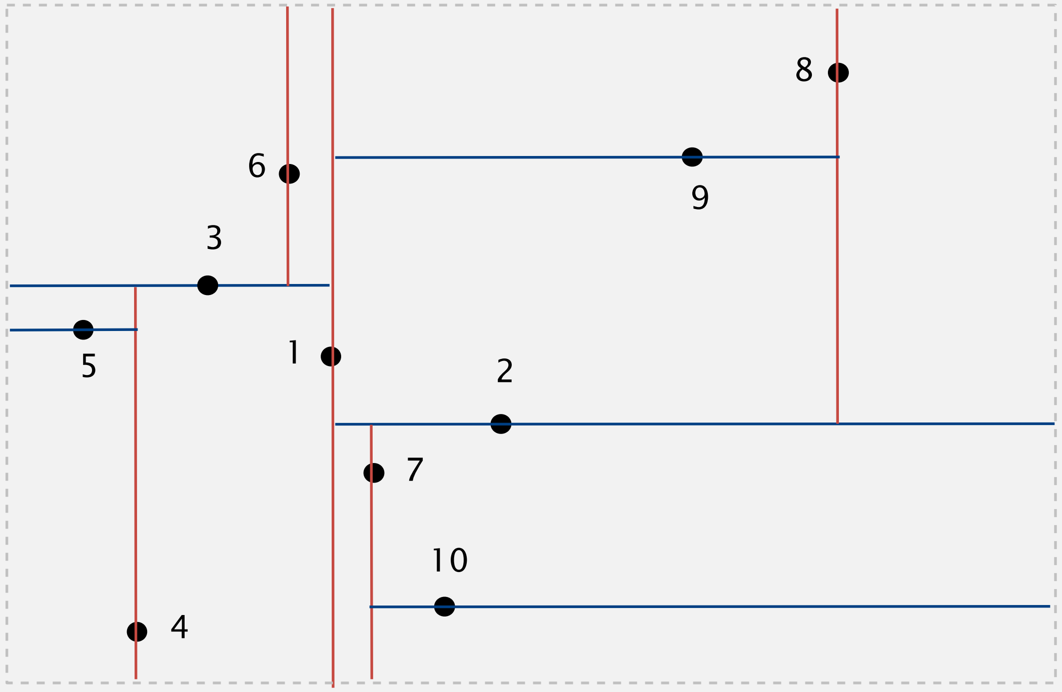
- Search gives rectangle containing point.
- Insert further subdivides the plane.

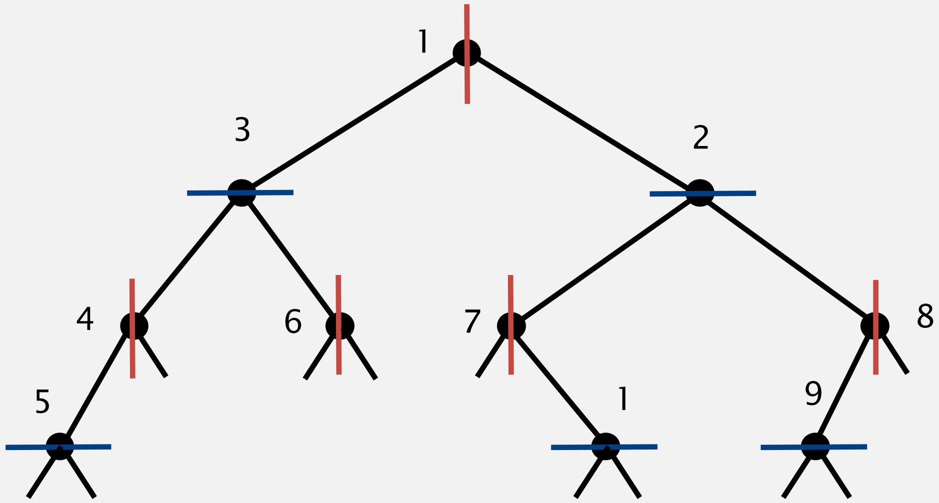


even levels of the tree



odd levels of the tree



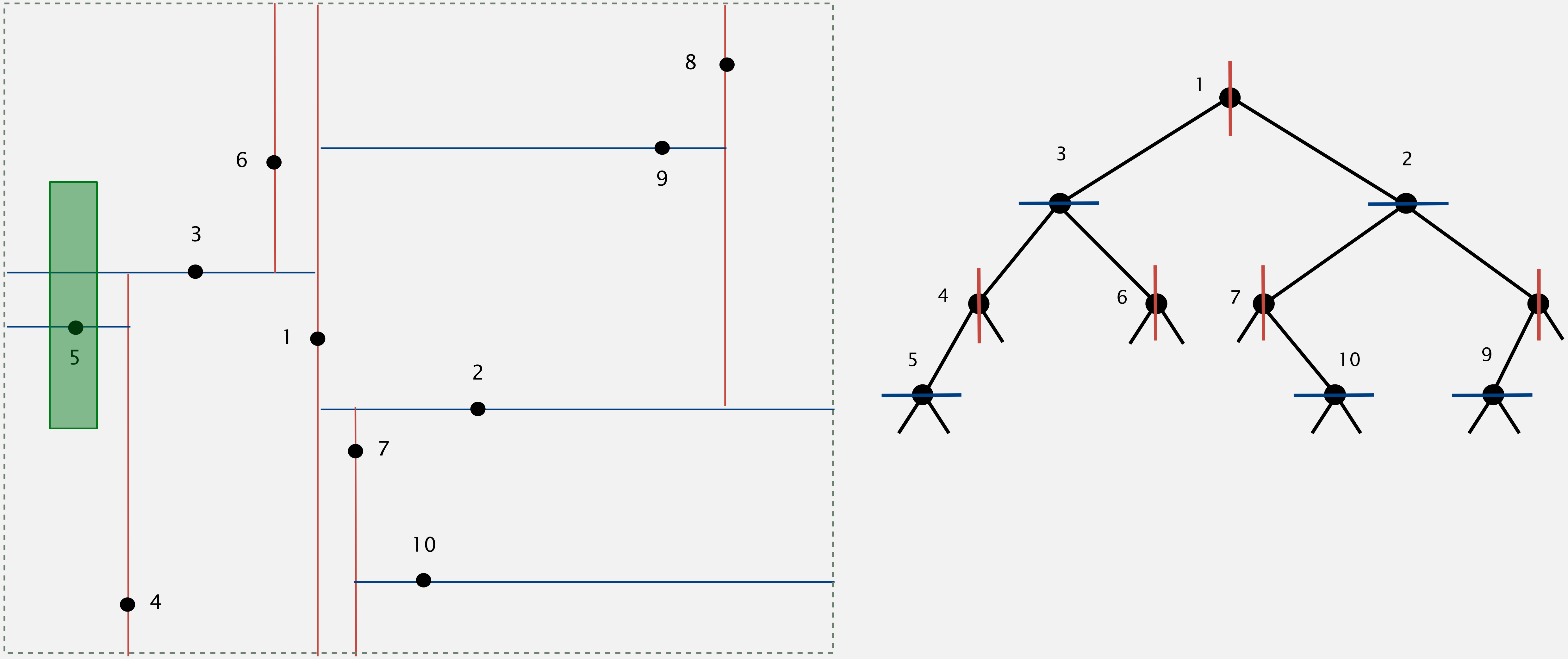


SO WHAT ABOUT RANGE SEARCH

2d tree demo: range search

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

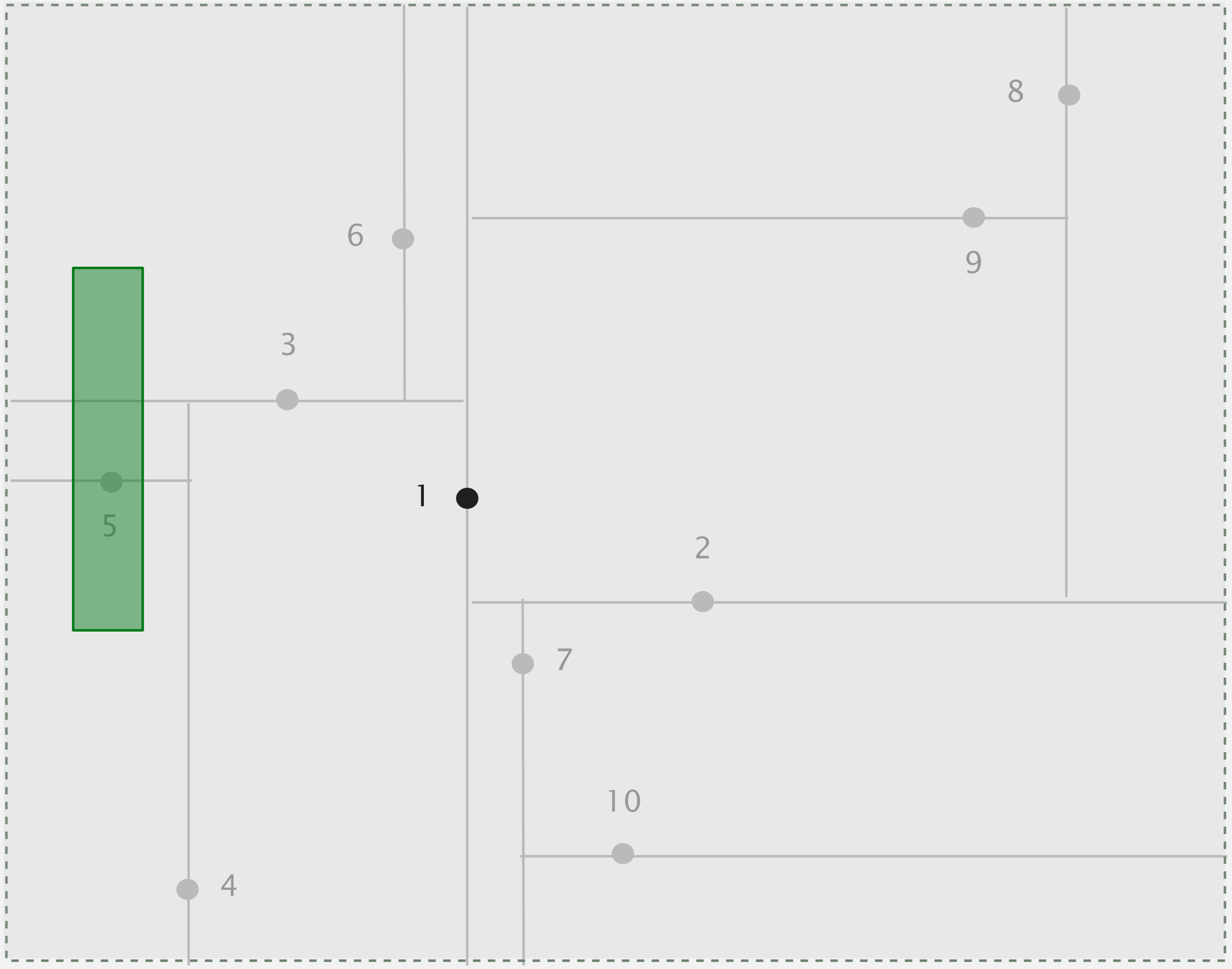
- Check if point in node lies in given rectangle.
- Recursively search left (if any points could fall in rectangle).
- Recursively search right (if any points could fall in rectangle).

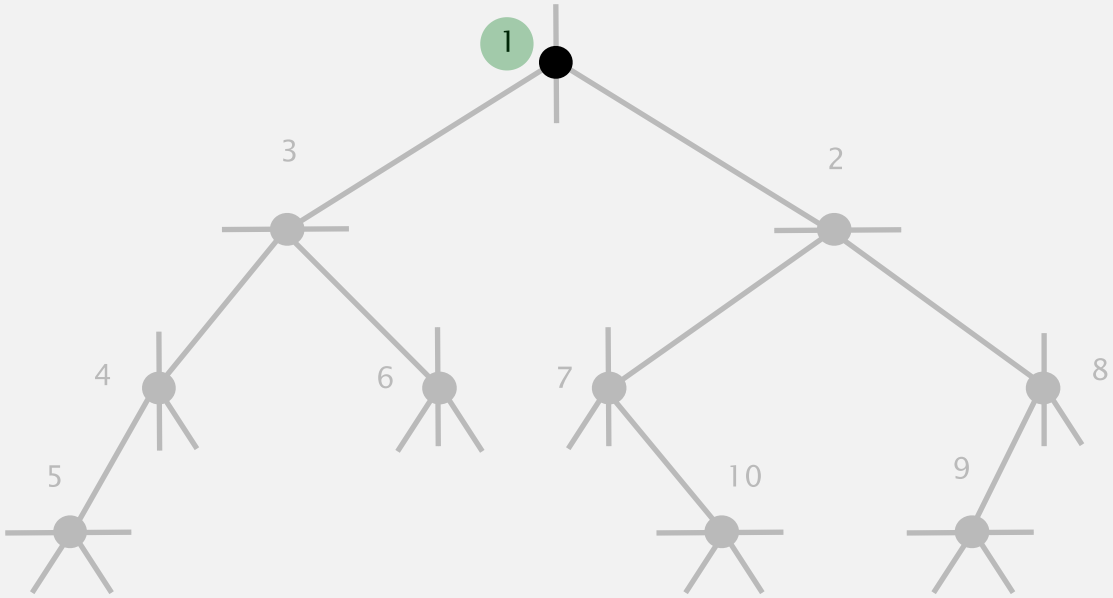


2d tree demo: range search

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).





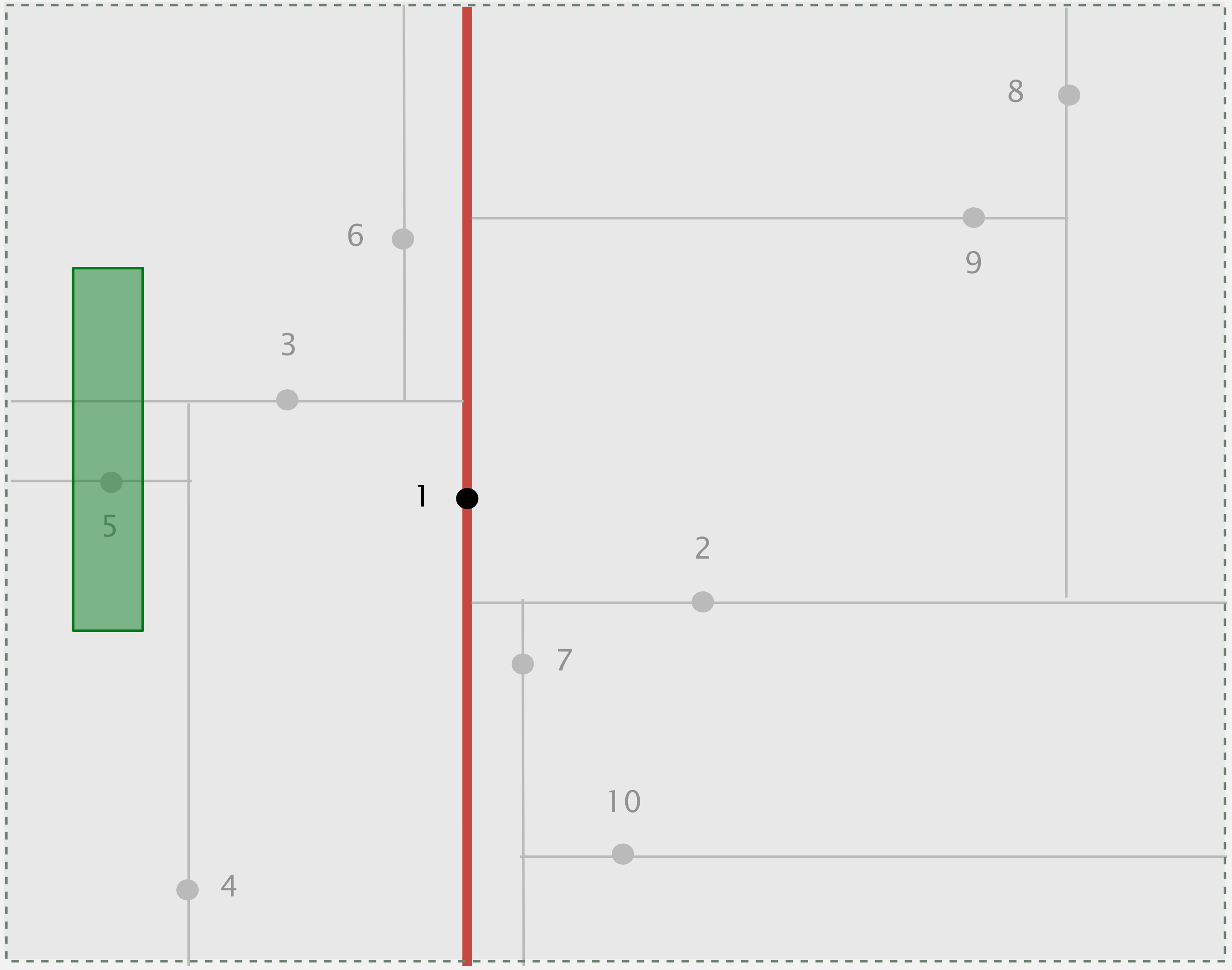
search root node

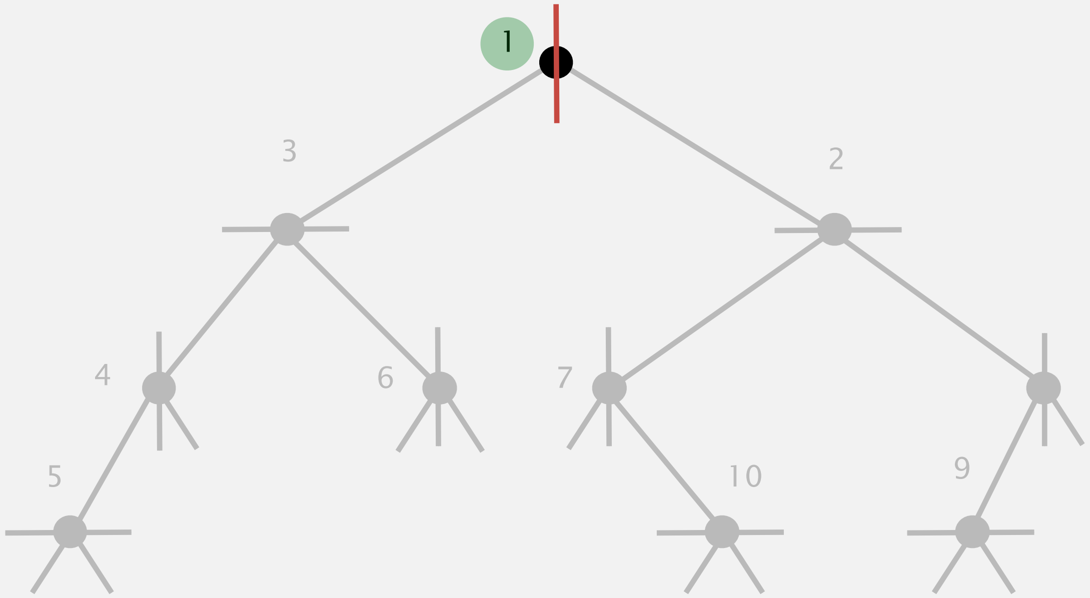
check if query rectangle contains point 1

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).



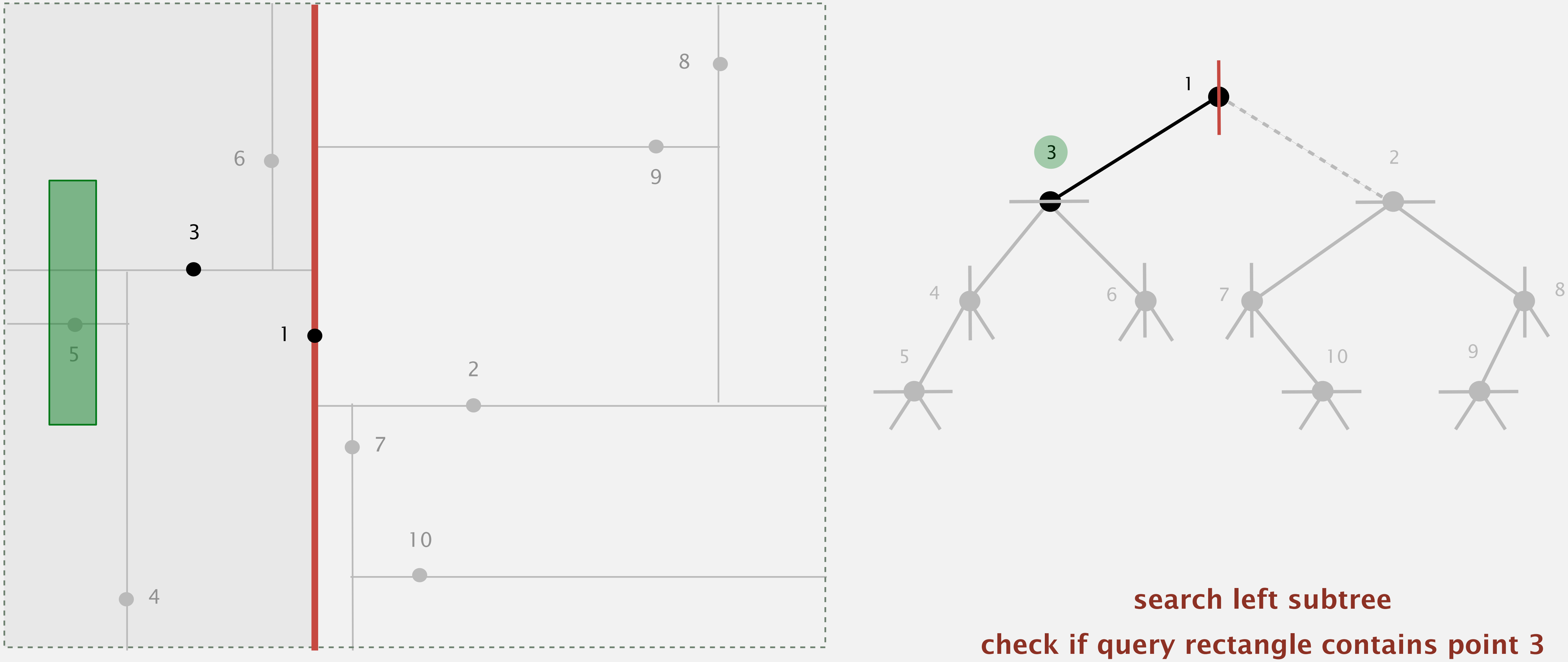


query rectangle to left of splitting line
search only in left subtree

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).



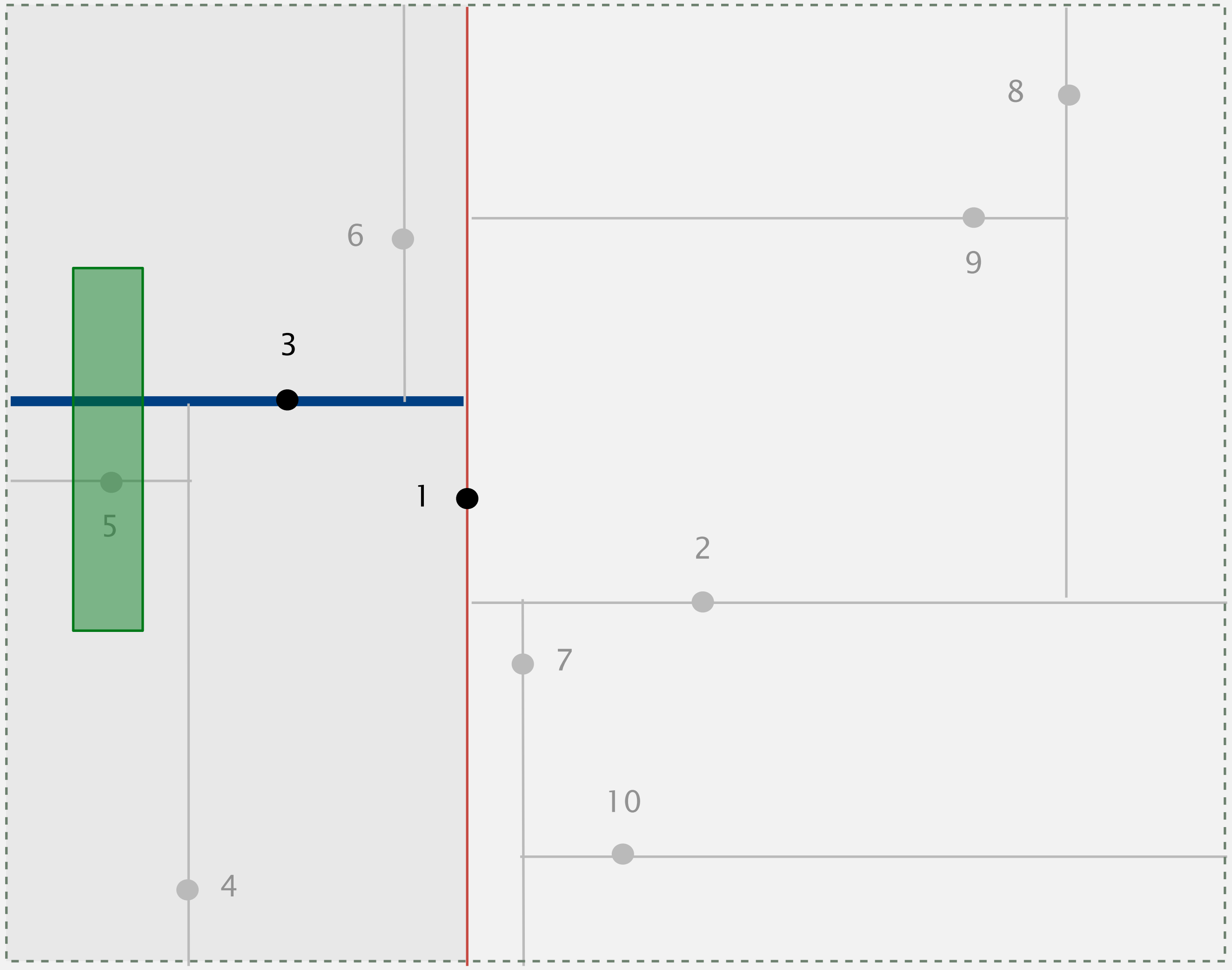
search left subtree

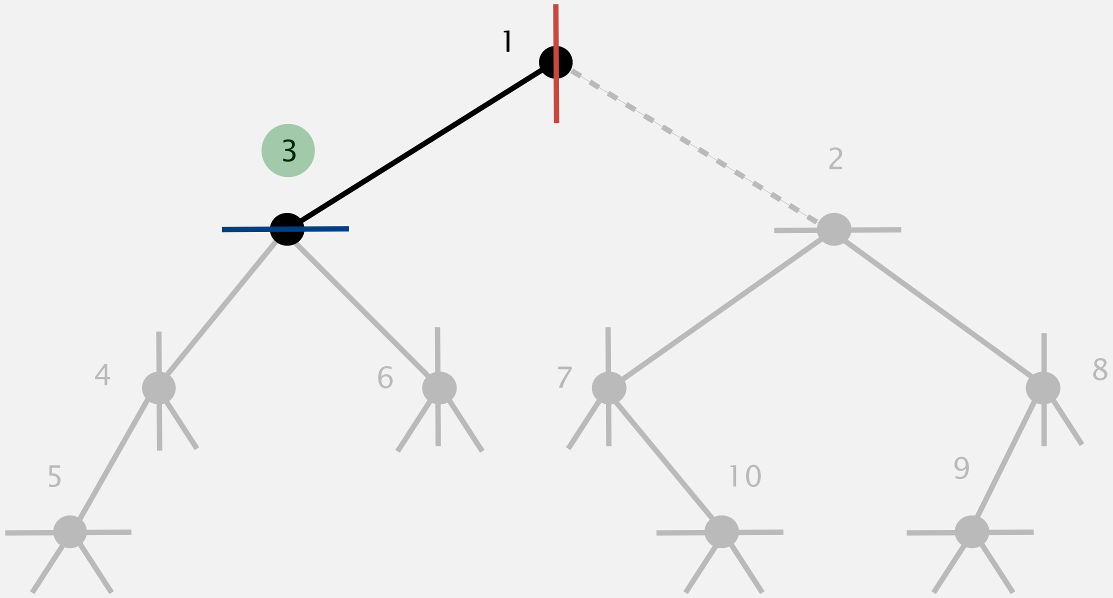
check if query rectangle contains point 3

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).



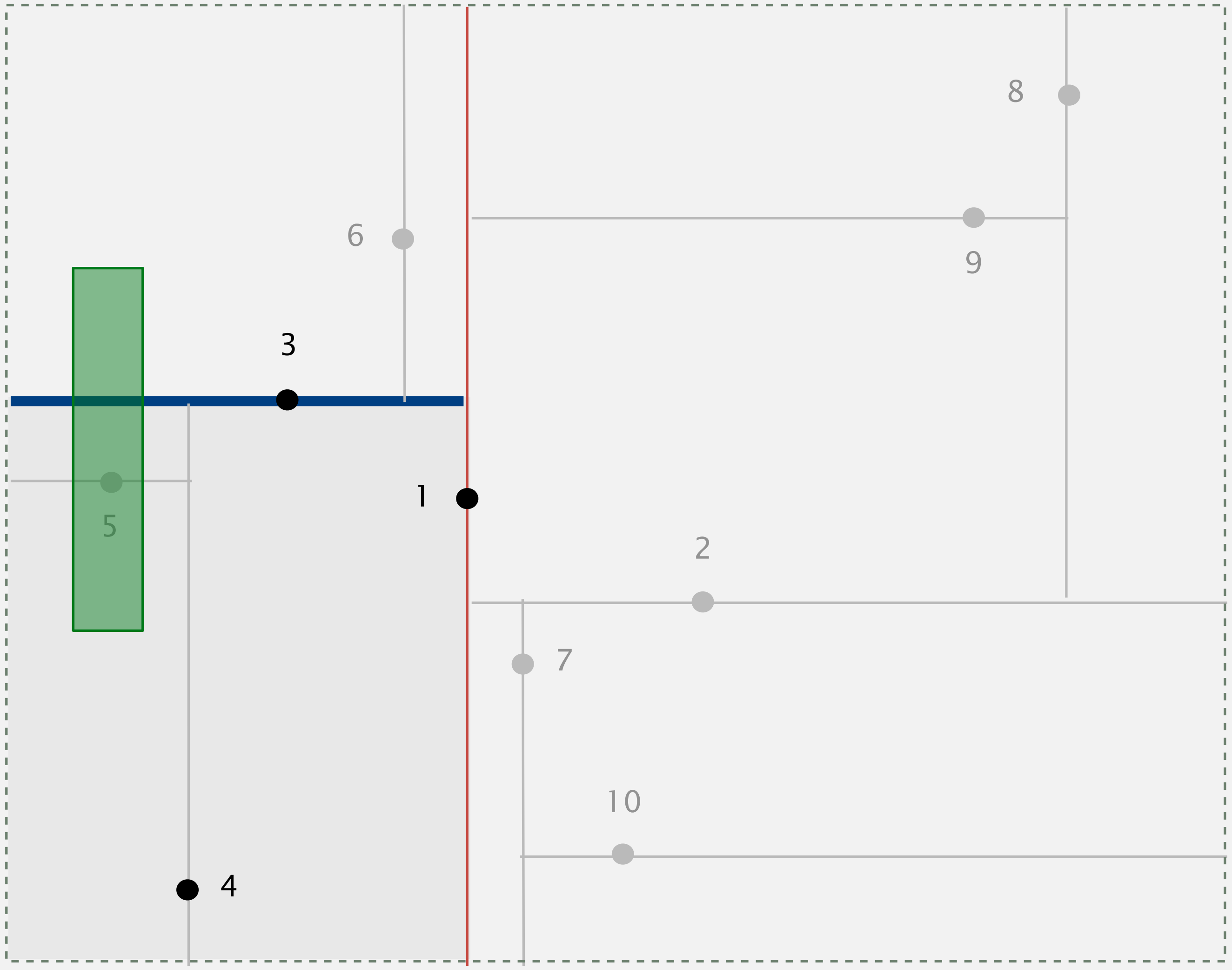


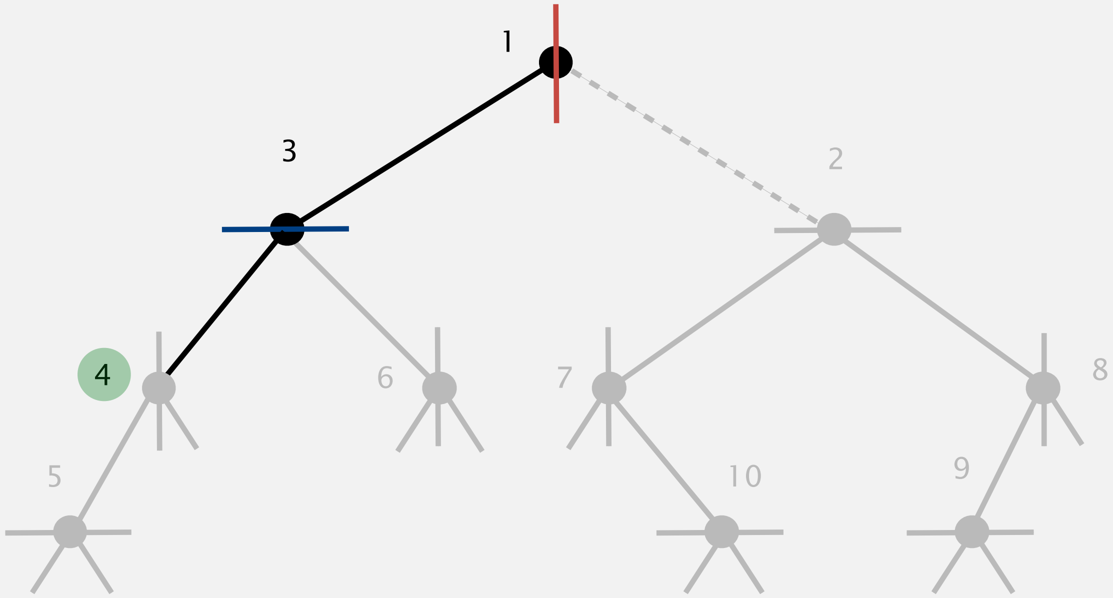
query rectangle intersects splitting line
search bottom and top subtrees

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).





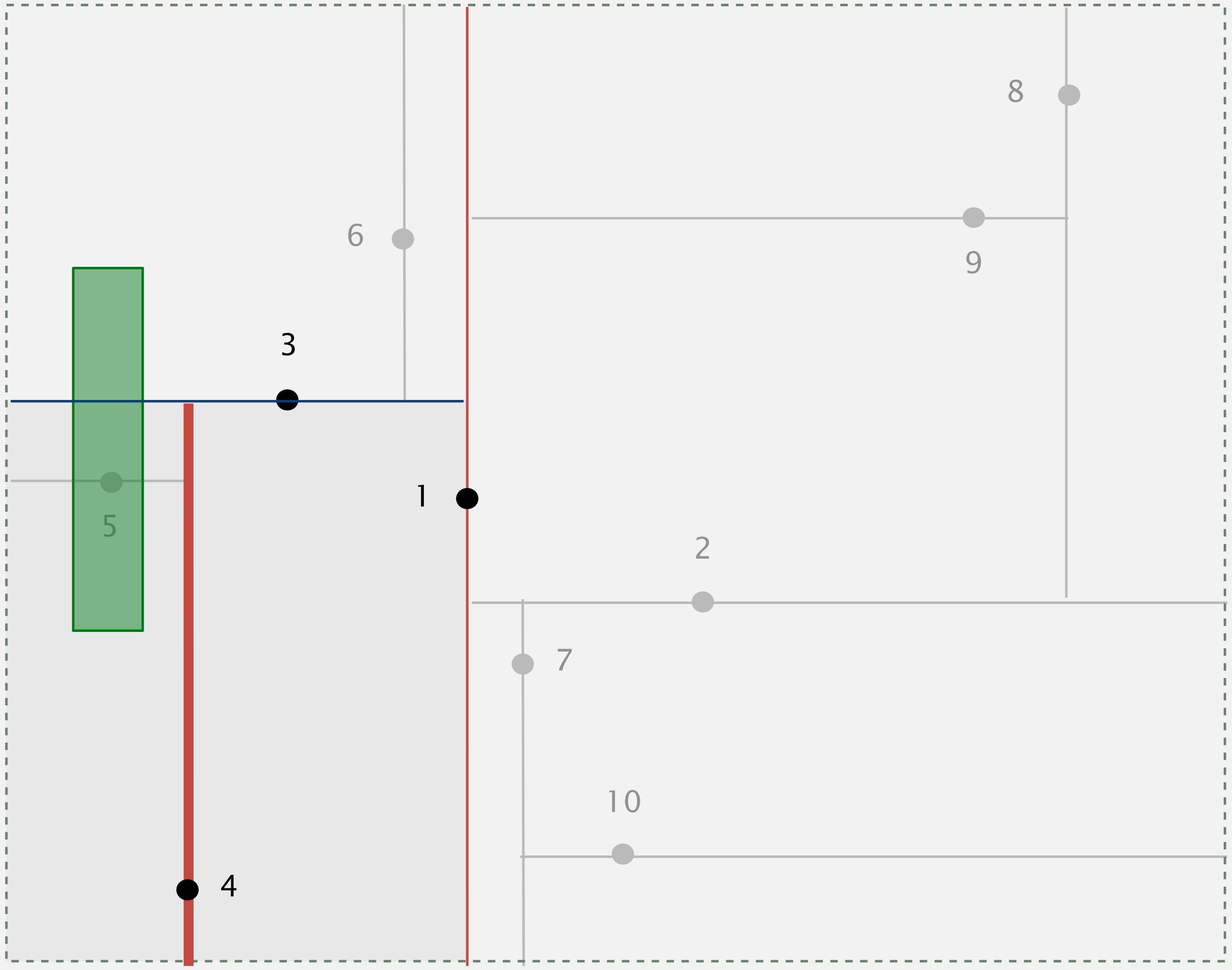
search left subtree

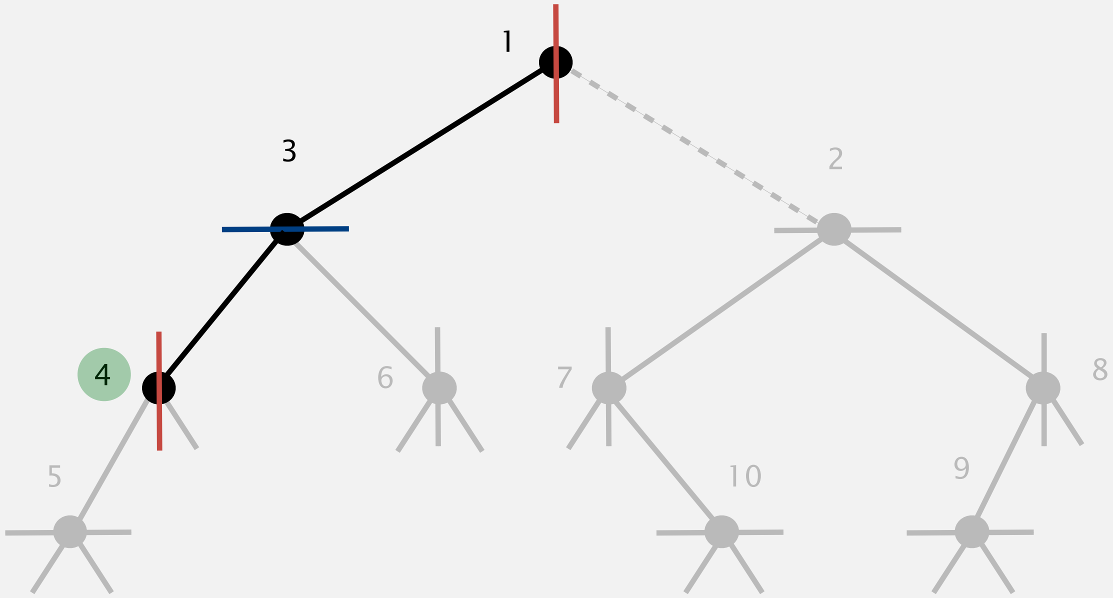
check if query rectangle contains point 4

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).



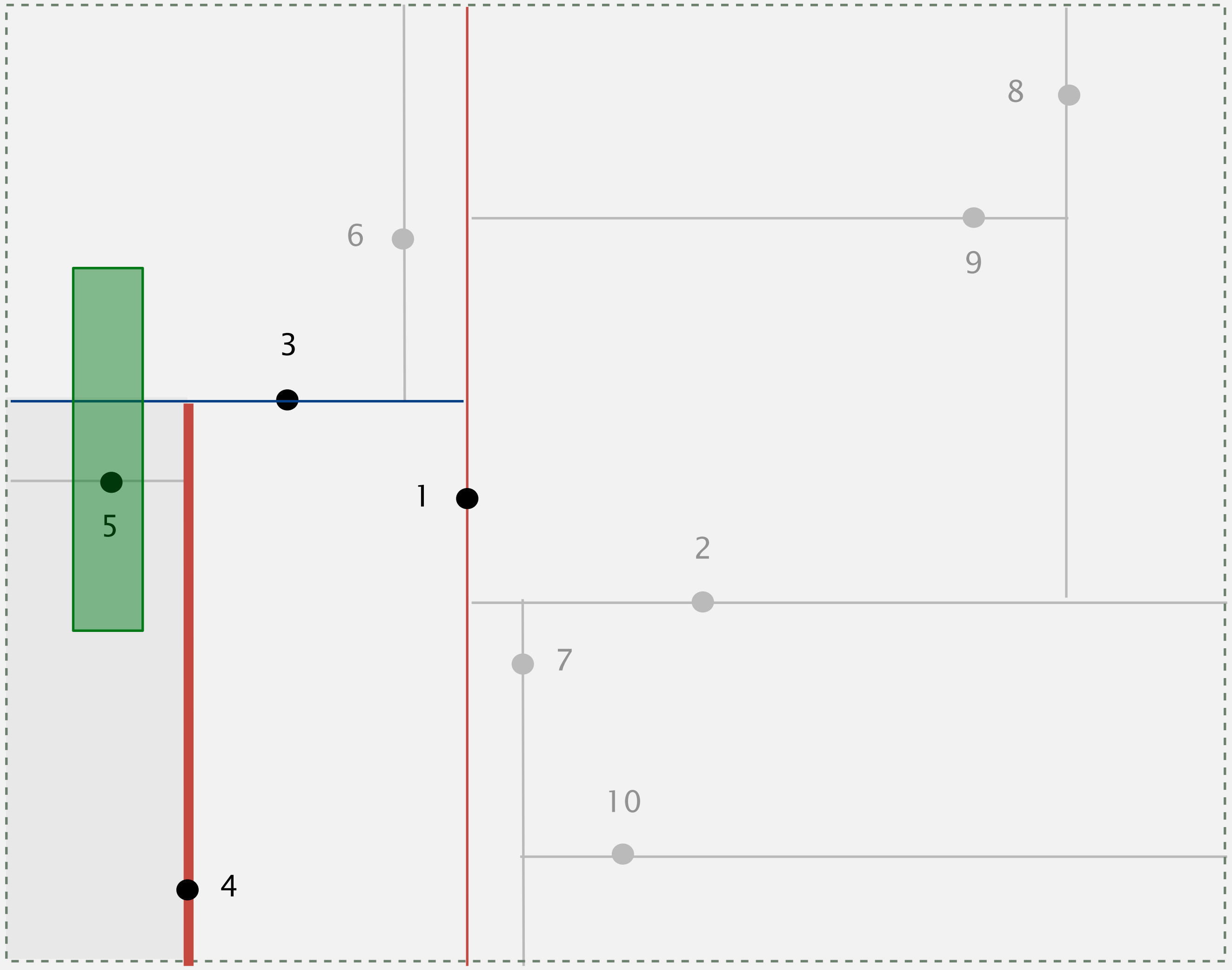


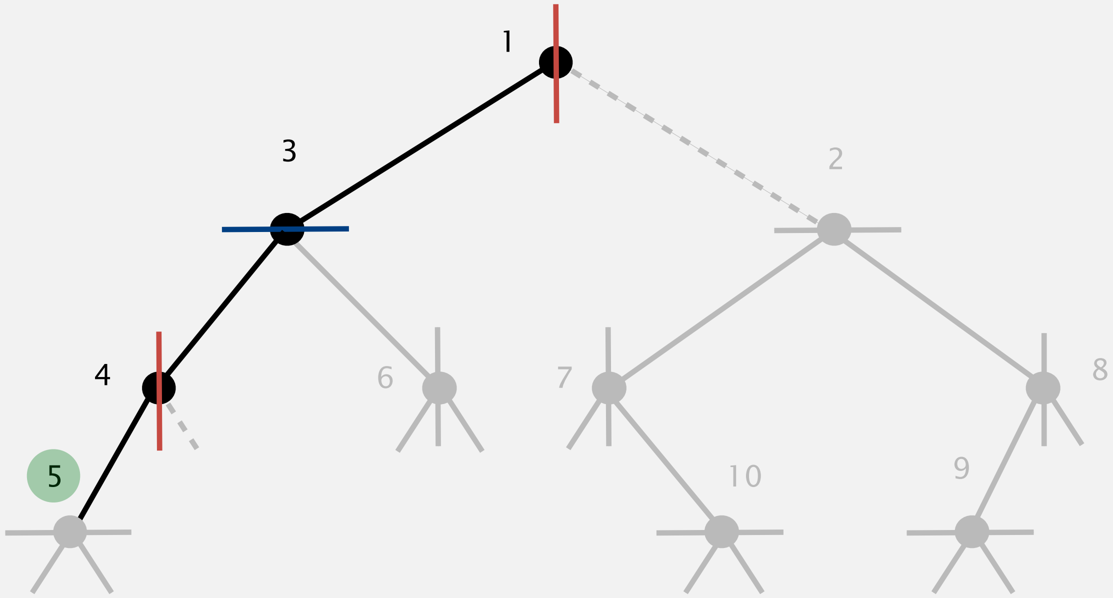
query rectangle to left of splitting line
search only in left subtree

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

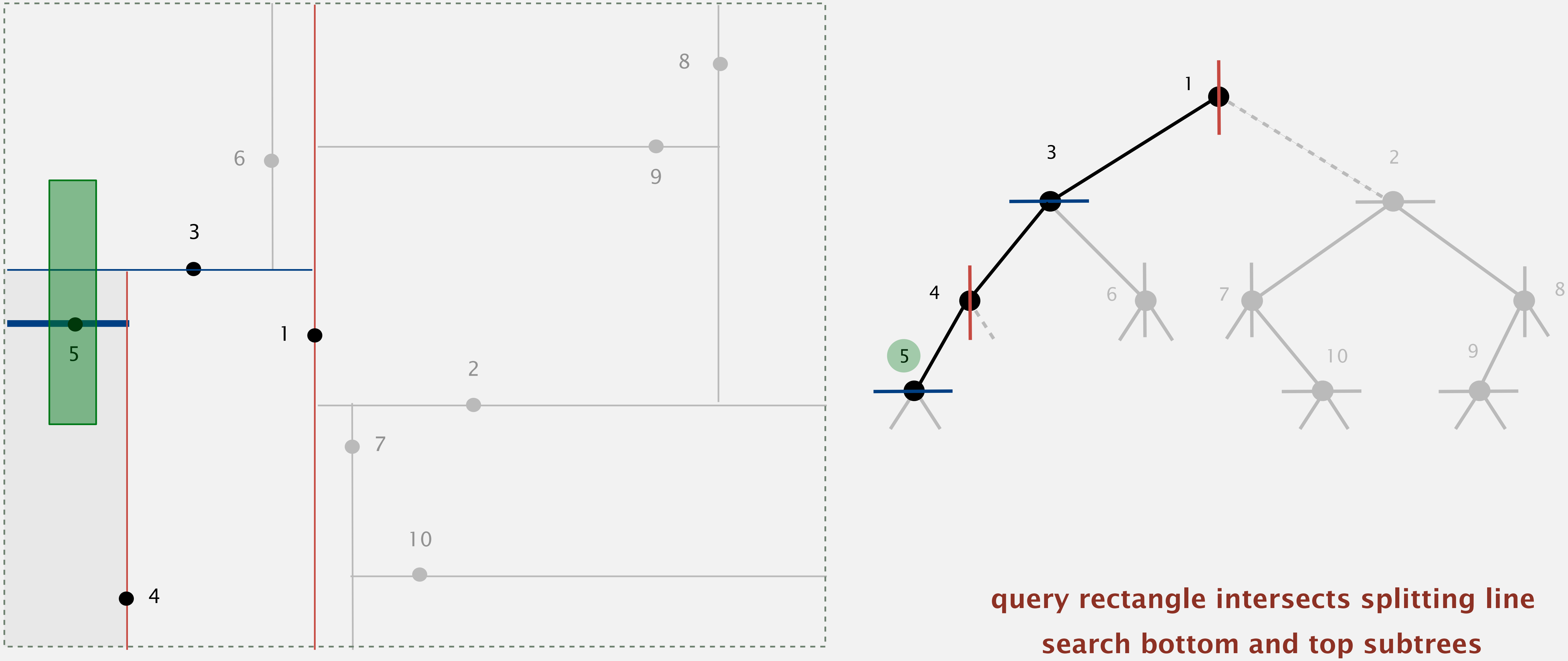
check if query rectangle contains point 5

(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).

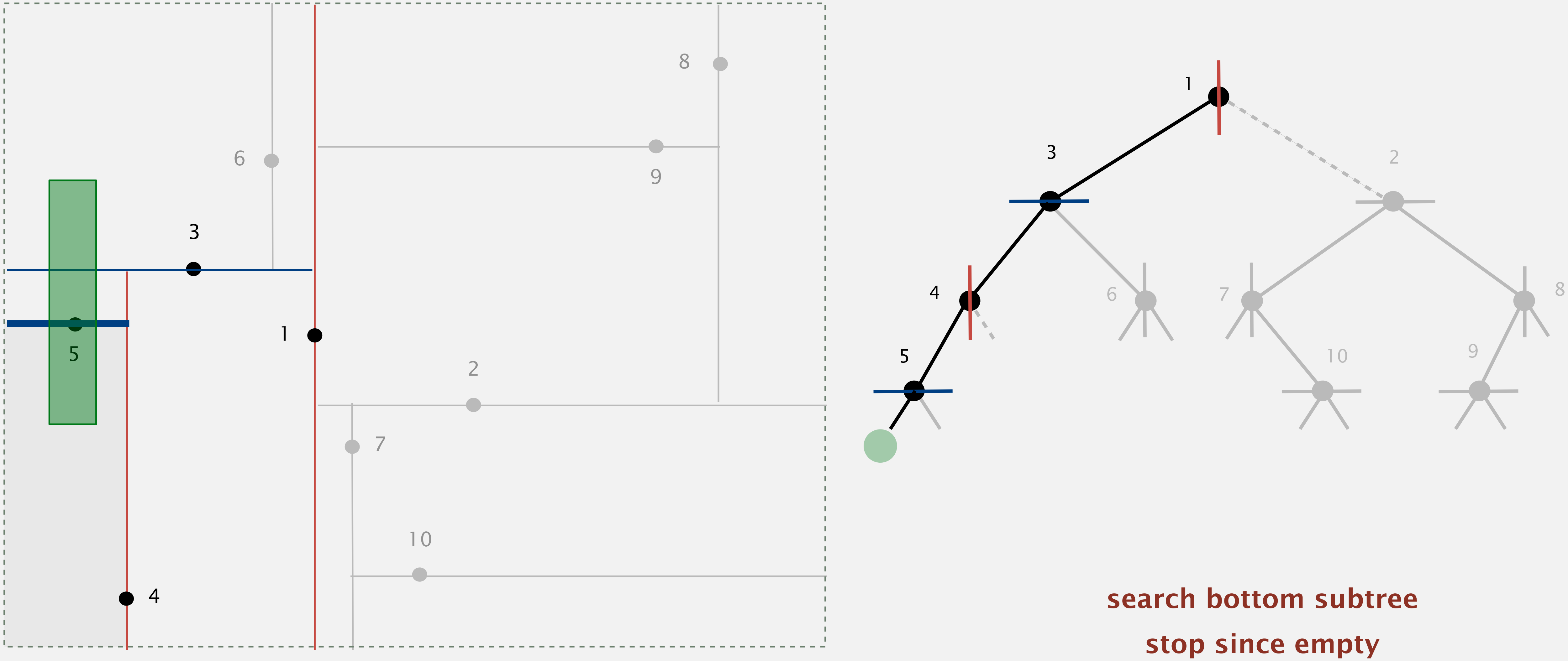


query rectangle intersects splitting line
search bottom and top subtrees

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).

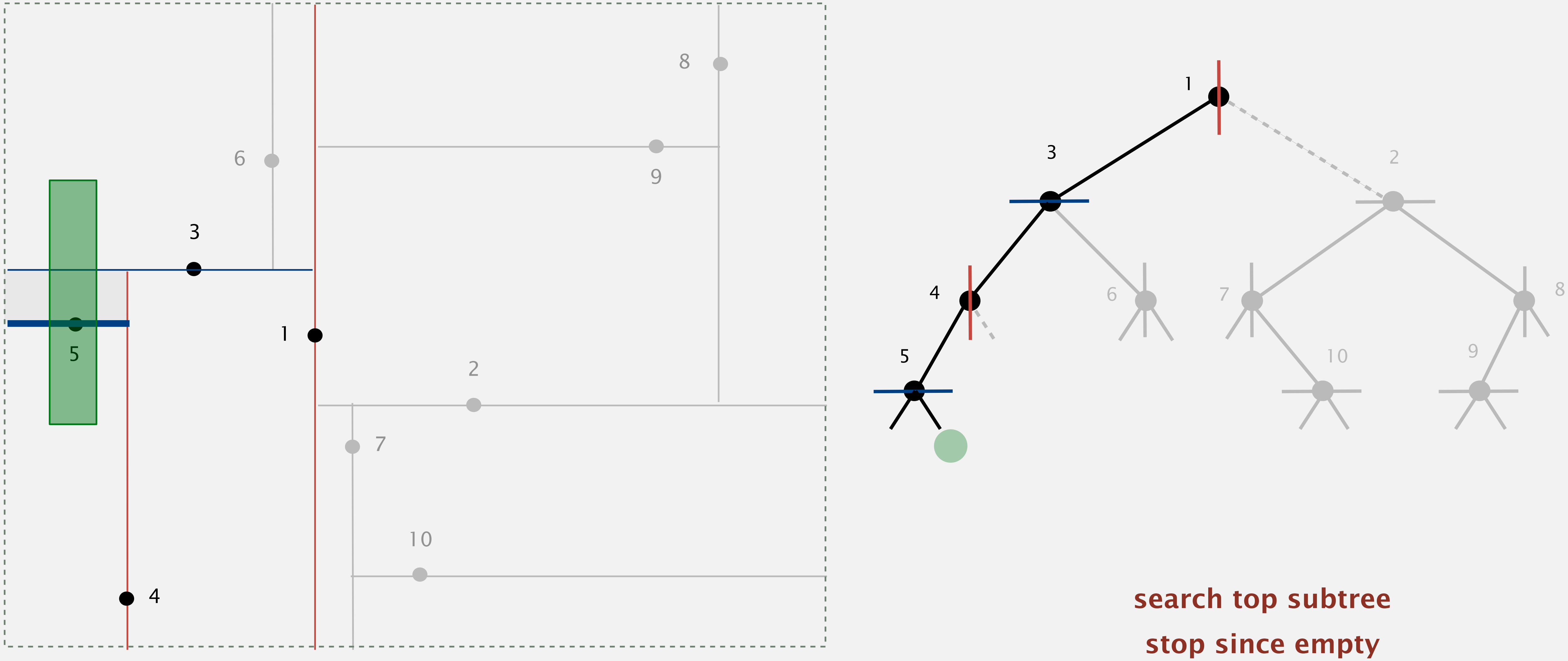


search bottom subtree
stop since empty

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).

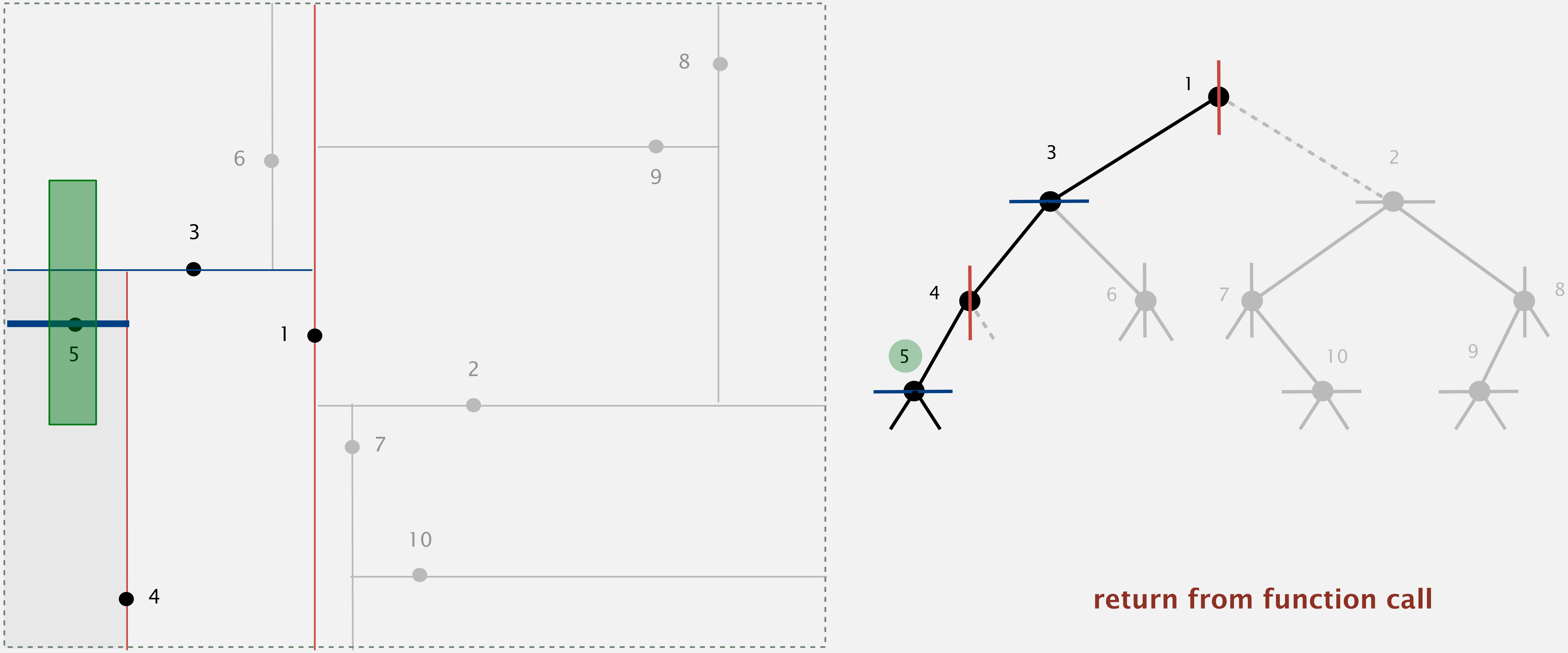


search top subtree
stop since empty

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).

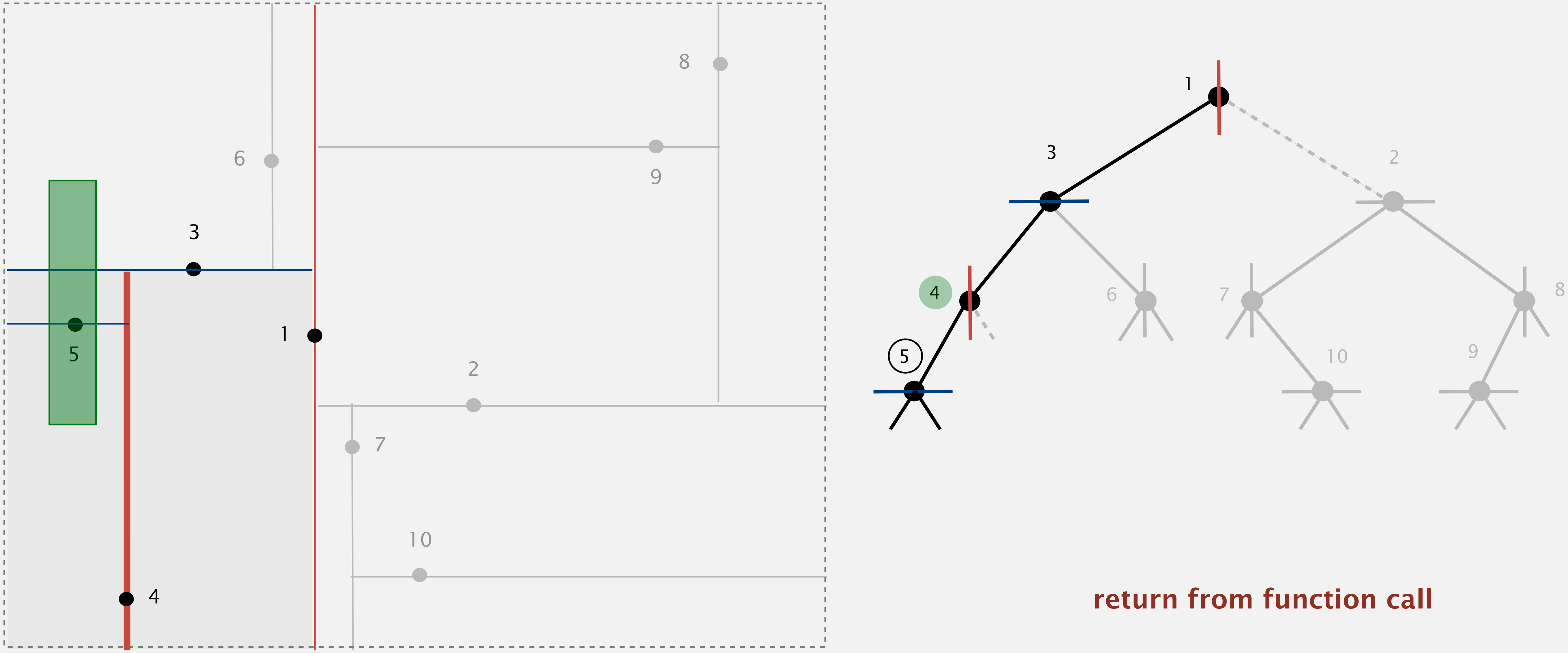


return from function call

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).

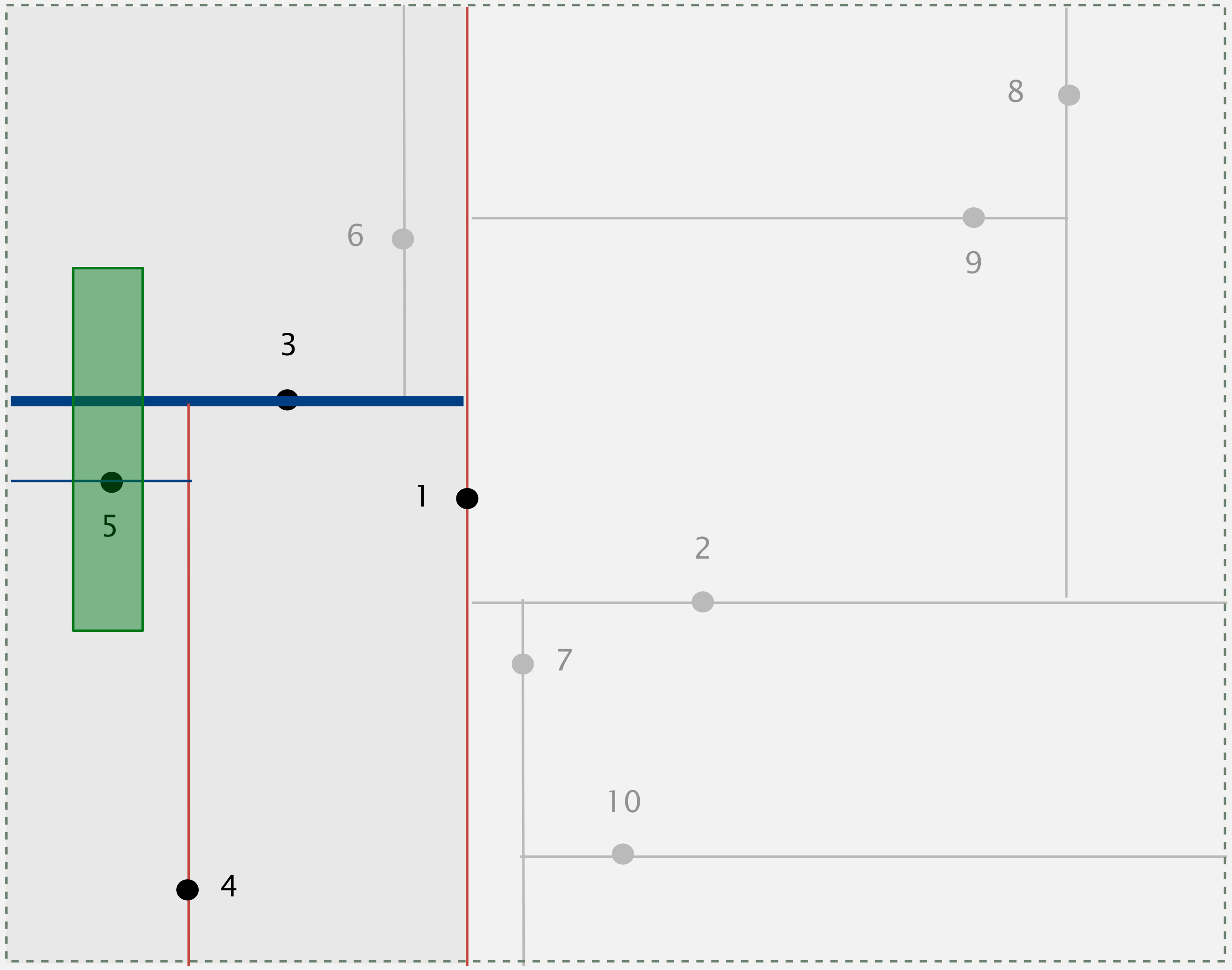


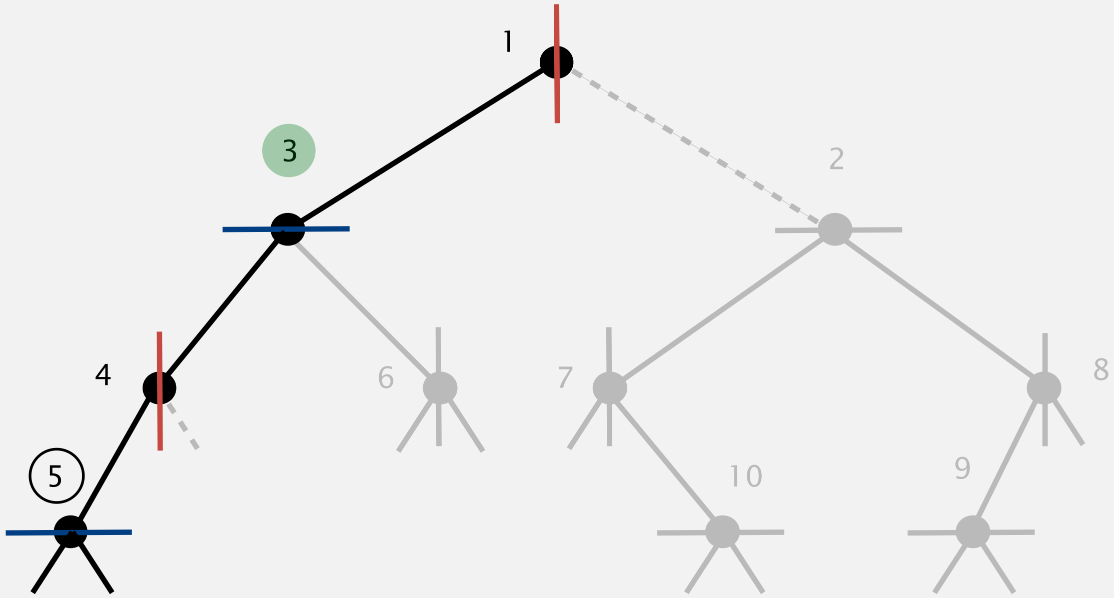
return from function call

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).



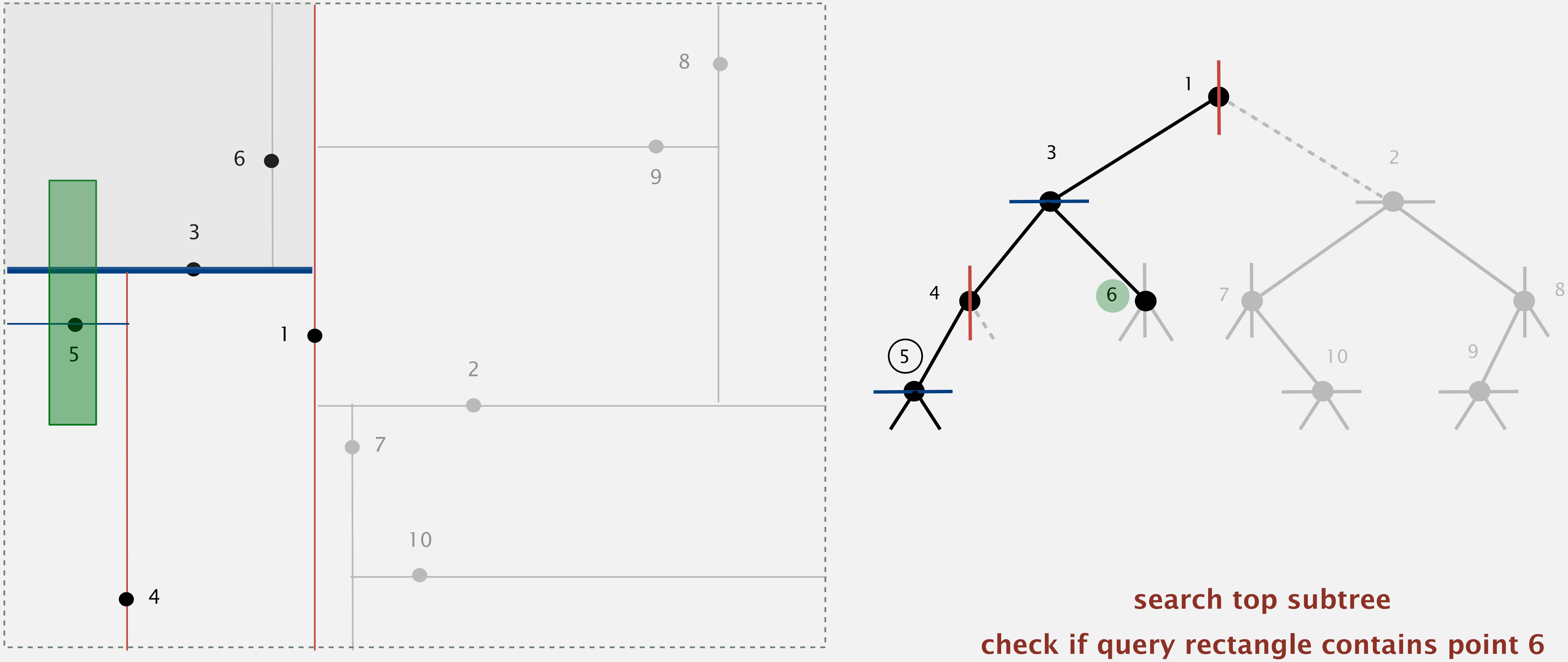


return from function call

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).

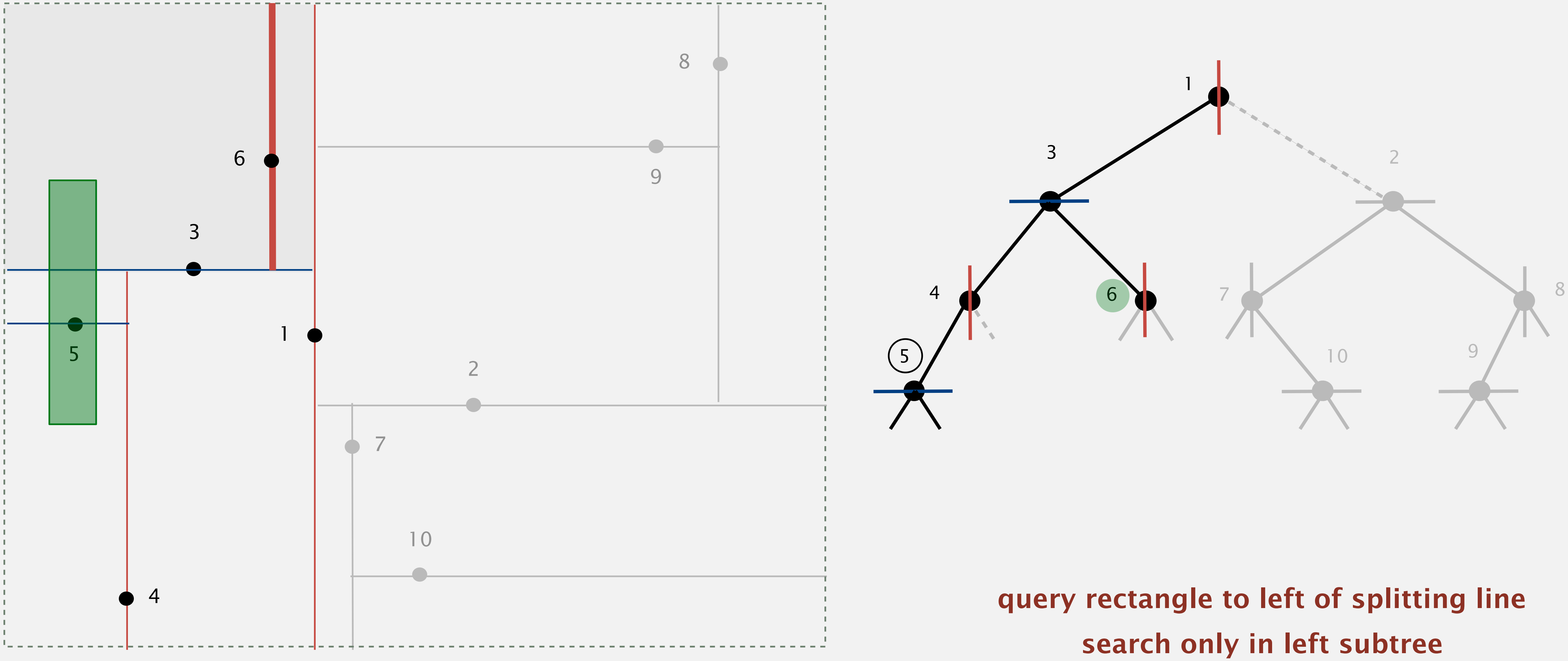


search top subtree
check if query rectangle contains point 6

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).

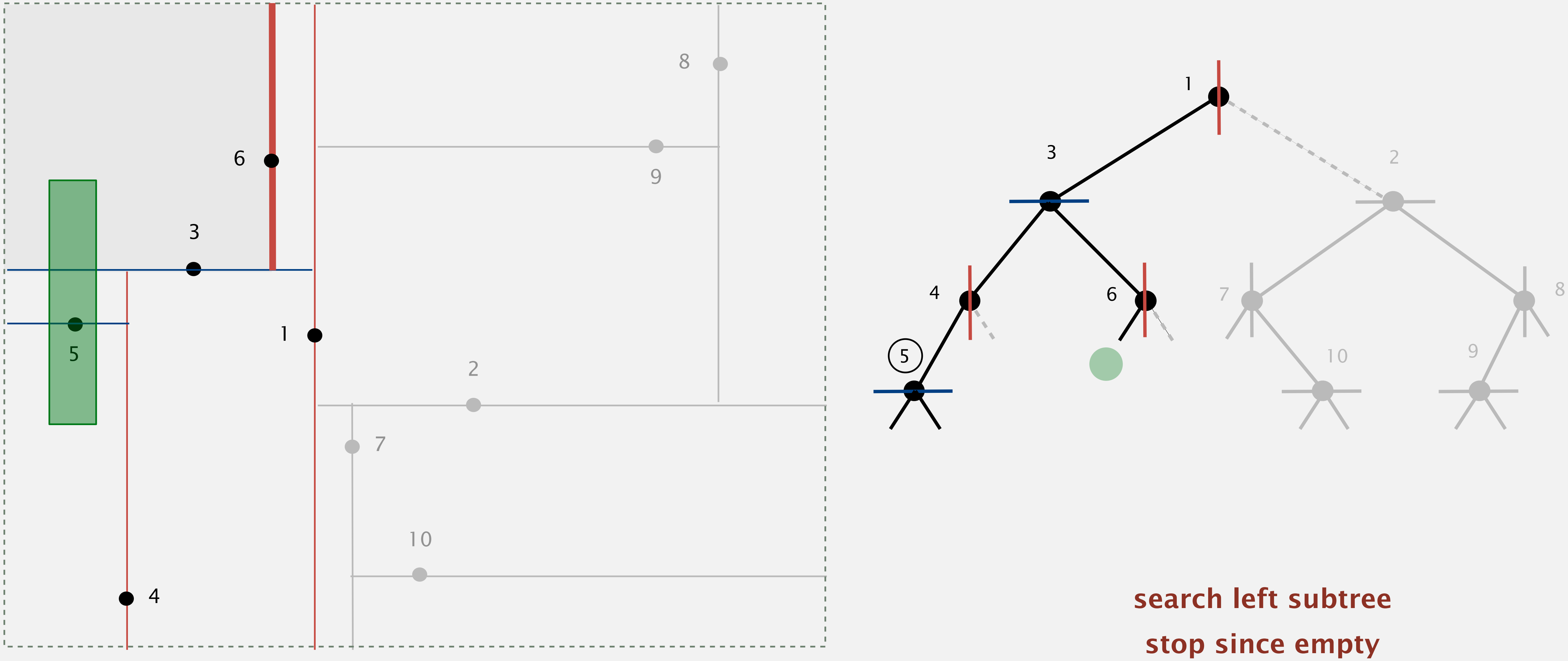


query rectangle to left of splitting line
search only in left subtree

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).

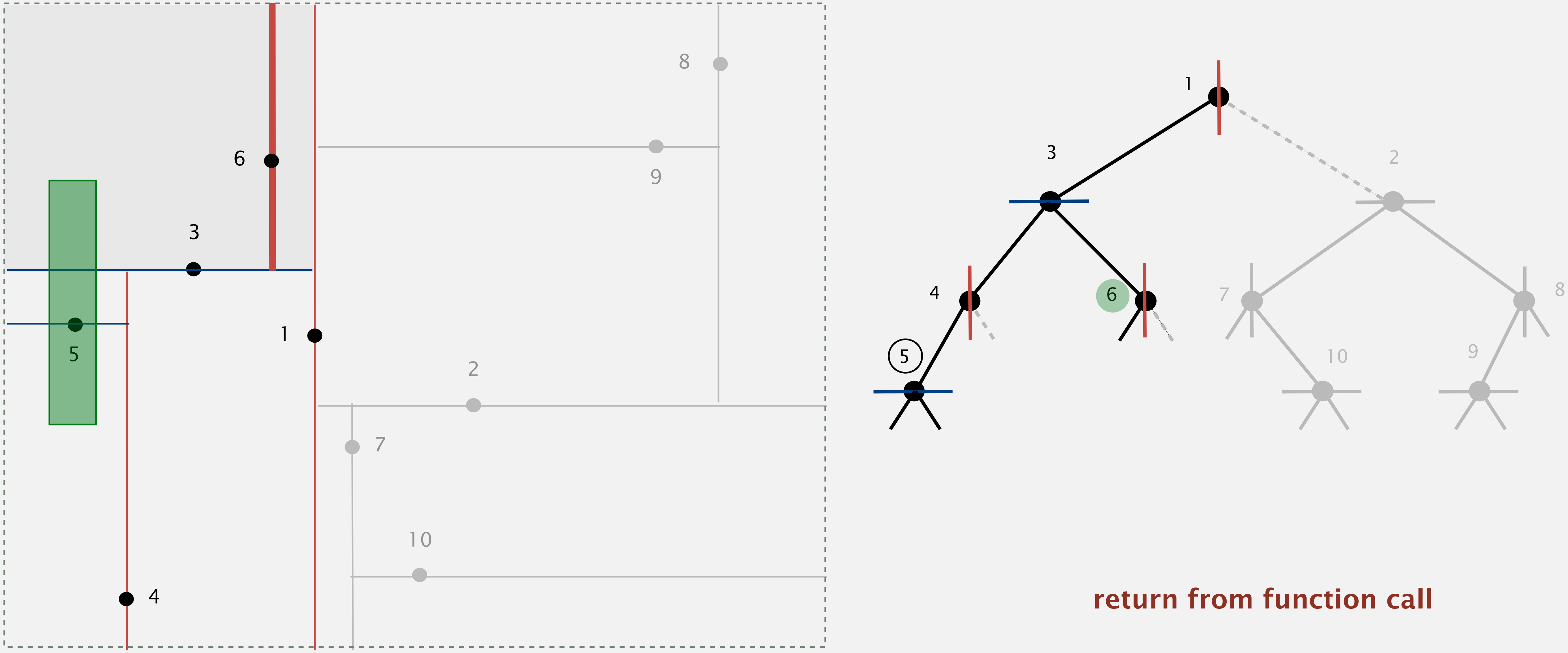


search left subtree
stop since empty

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).

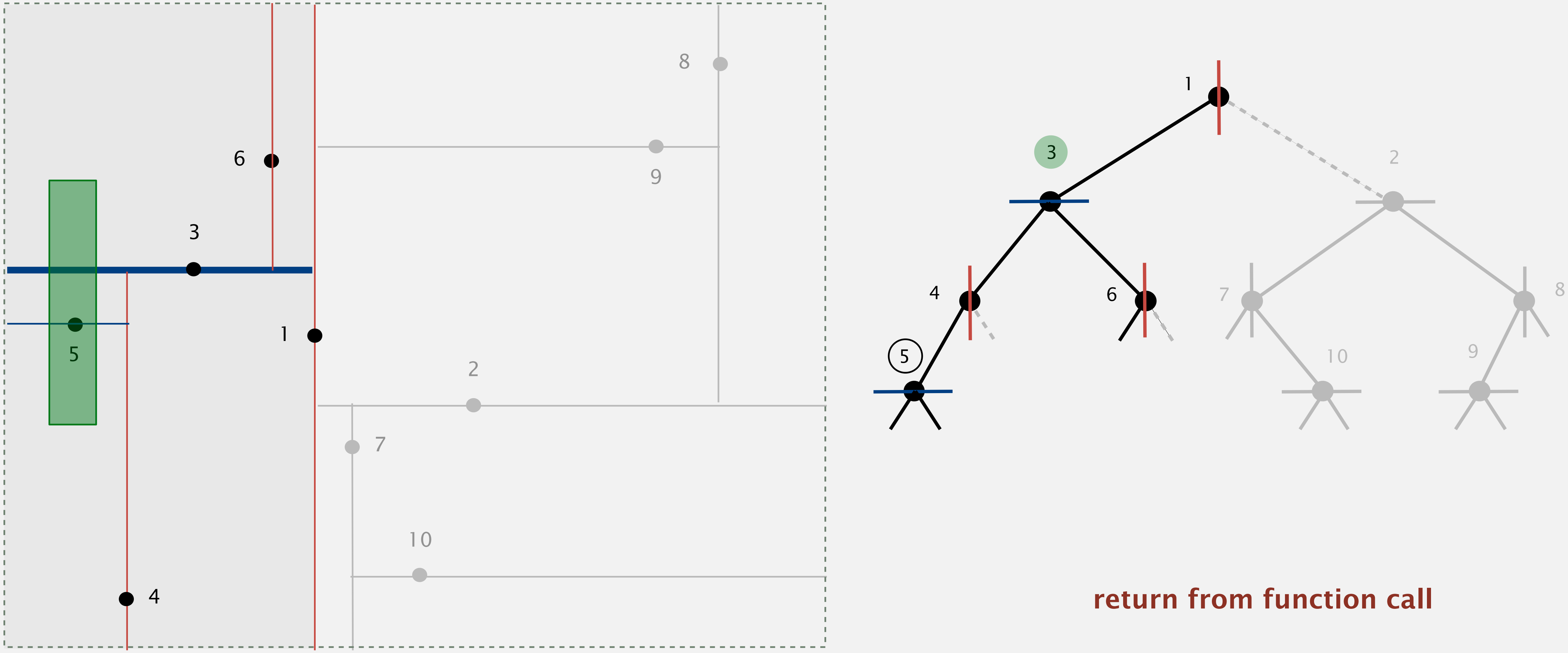


return from function call

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).

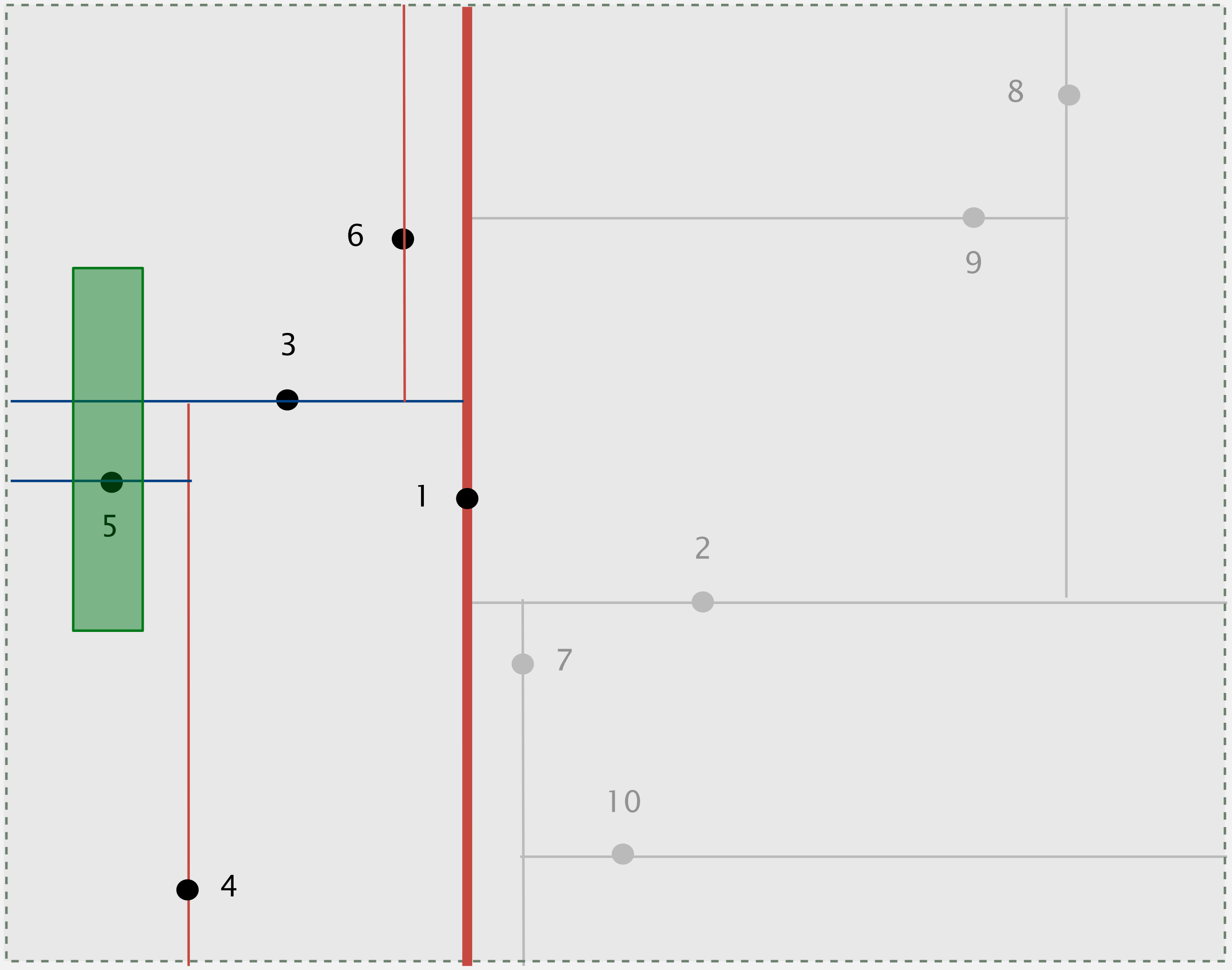


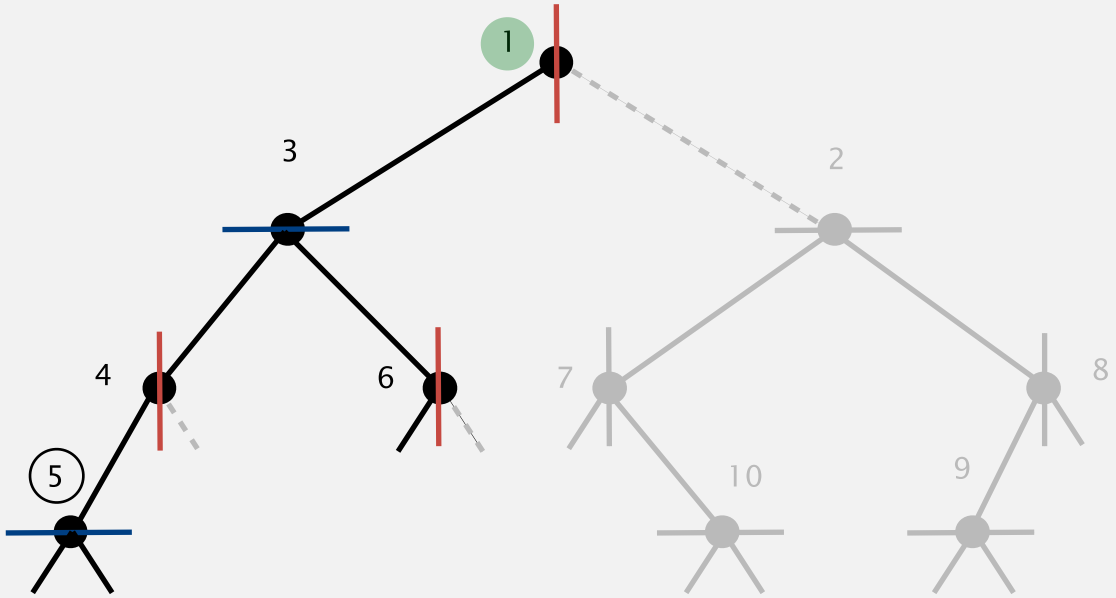
return from function call

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).



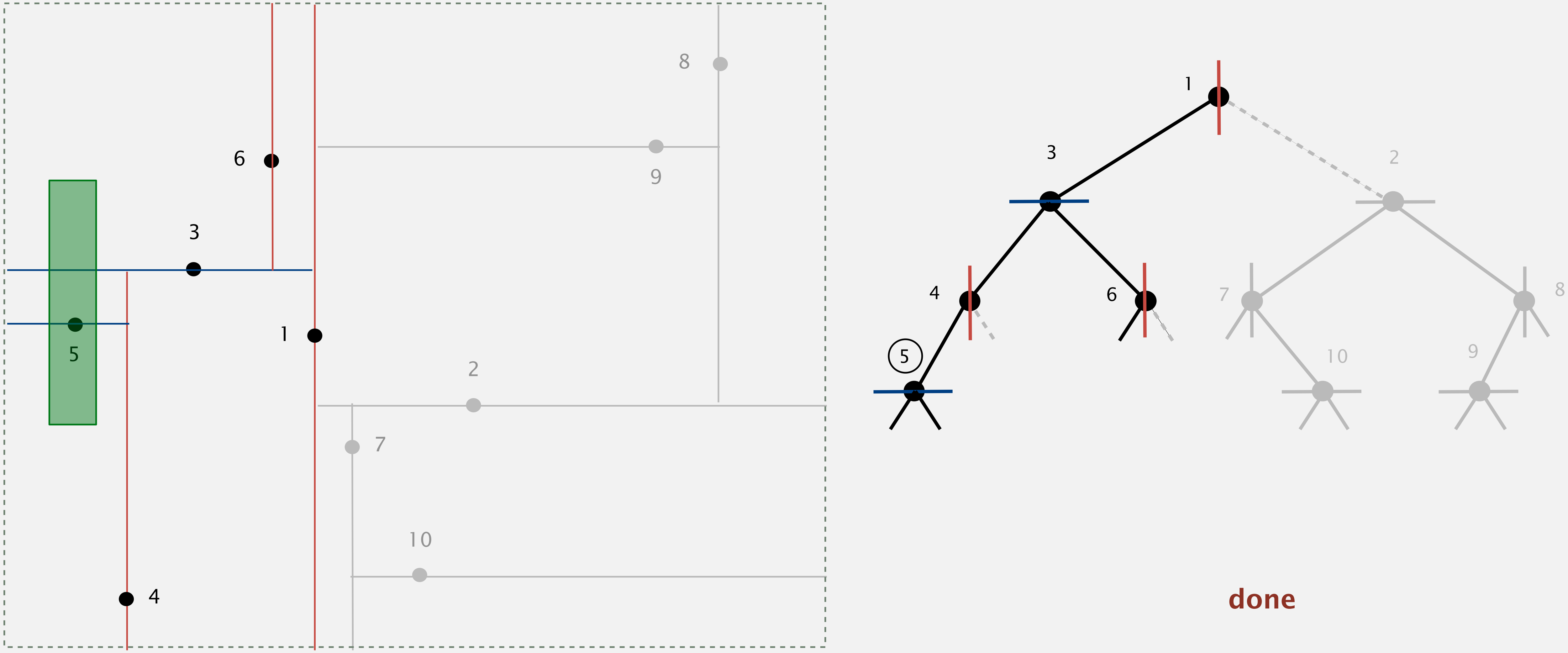


return from function call

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).

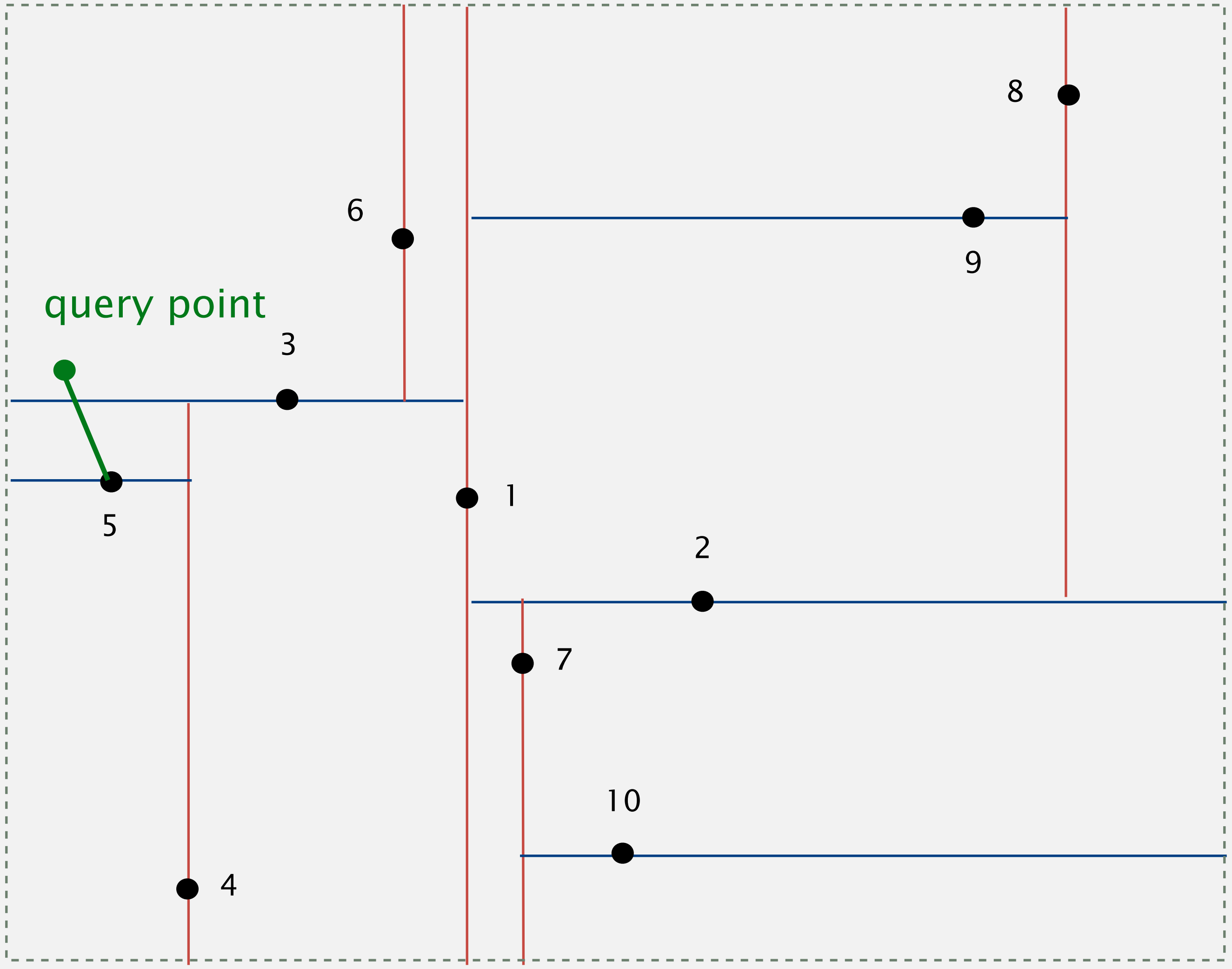


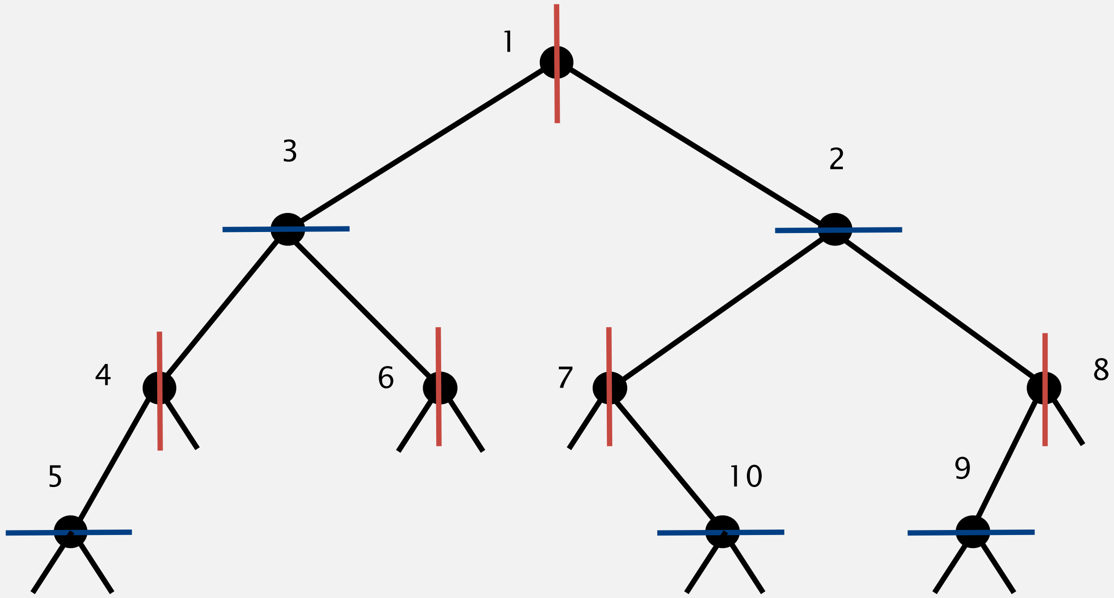
done

NEAREST NEIGHBOR

2d tree demo: nearest neighbor

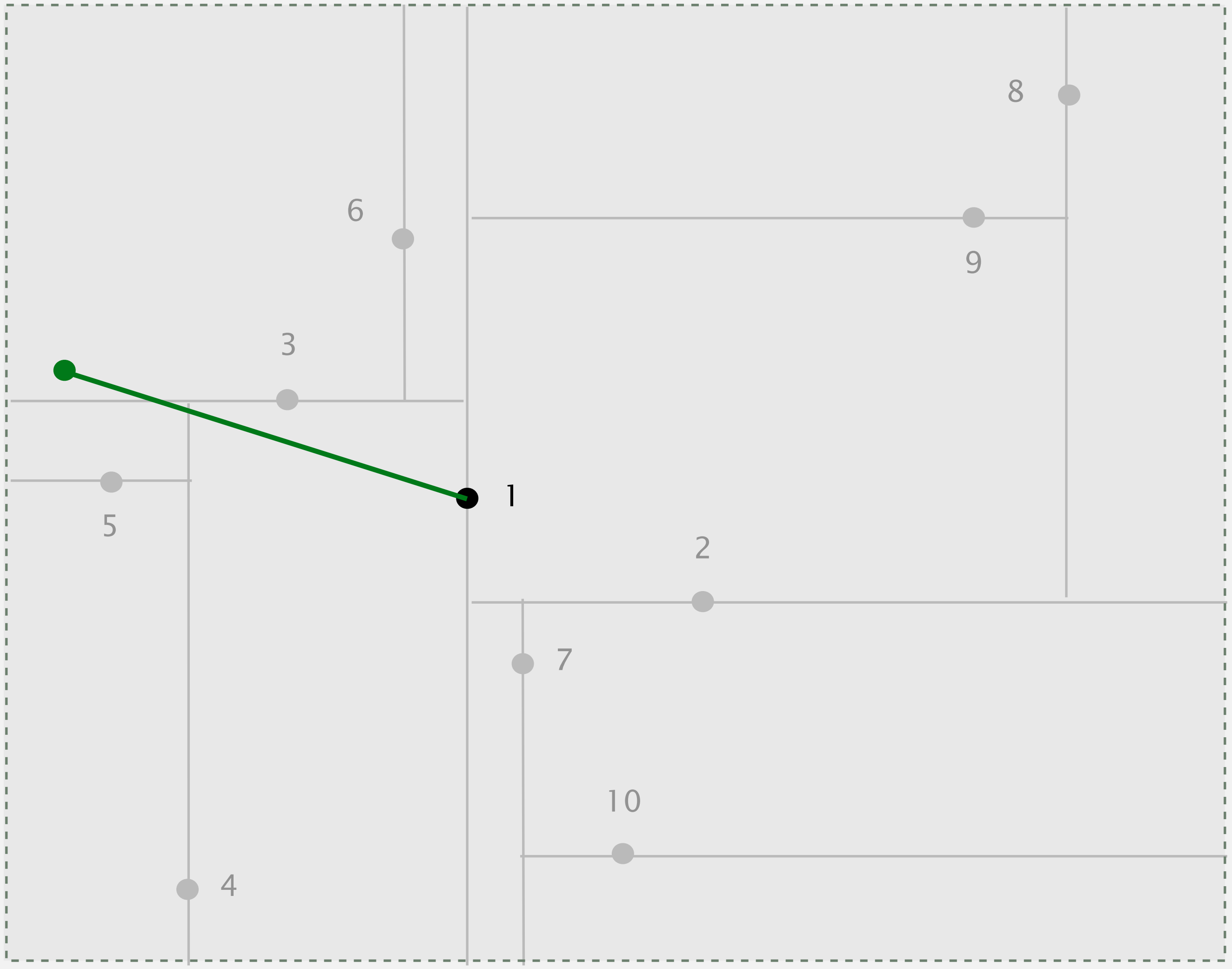
Goal. Find closest point to query point.

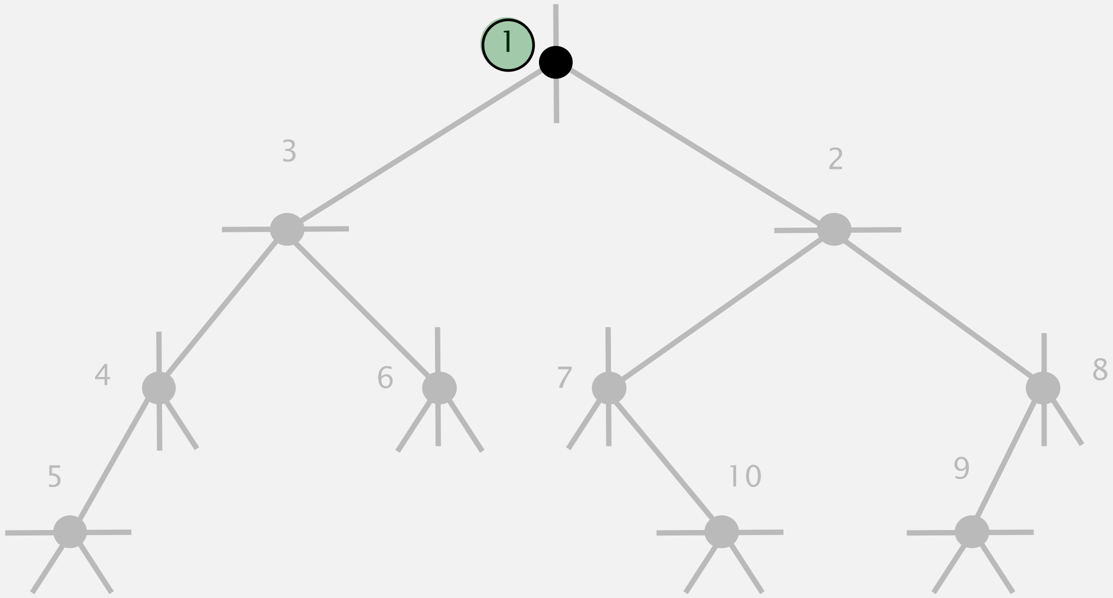




2d tree demo: nearest neighbor

- 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).

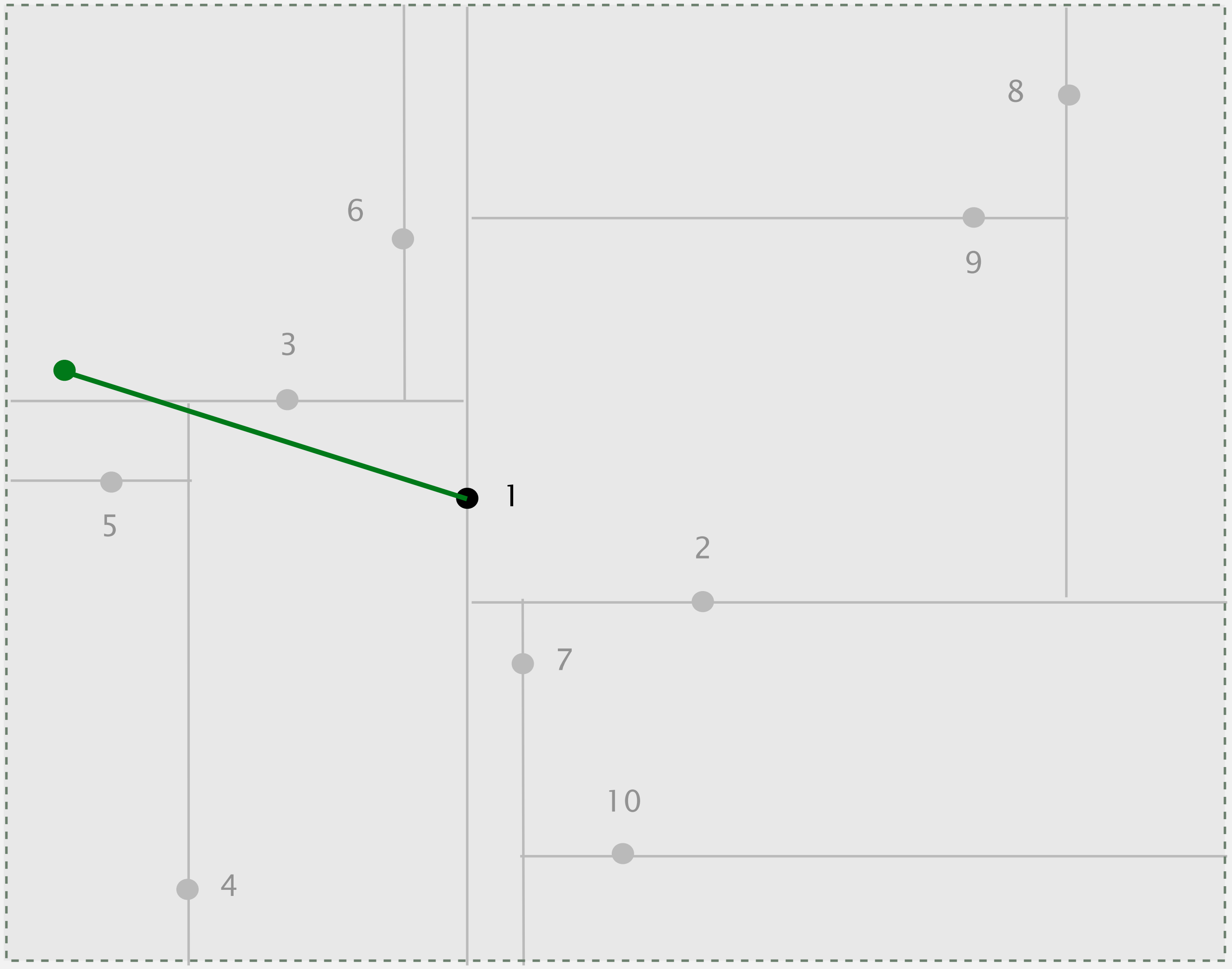


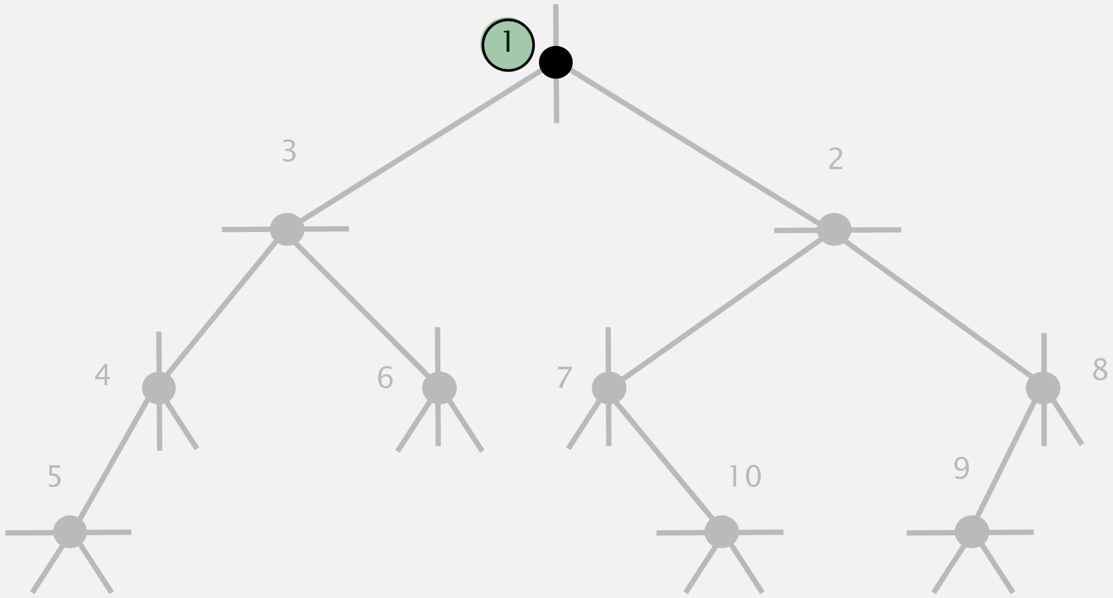


search root node
compute distance from query point to 1
(update champion nearest neighbor)

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).

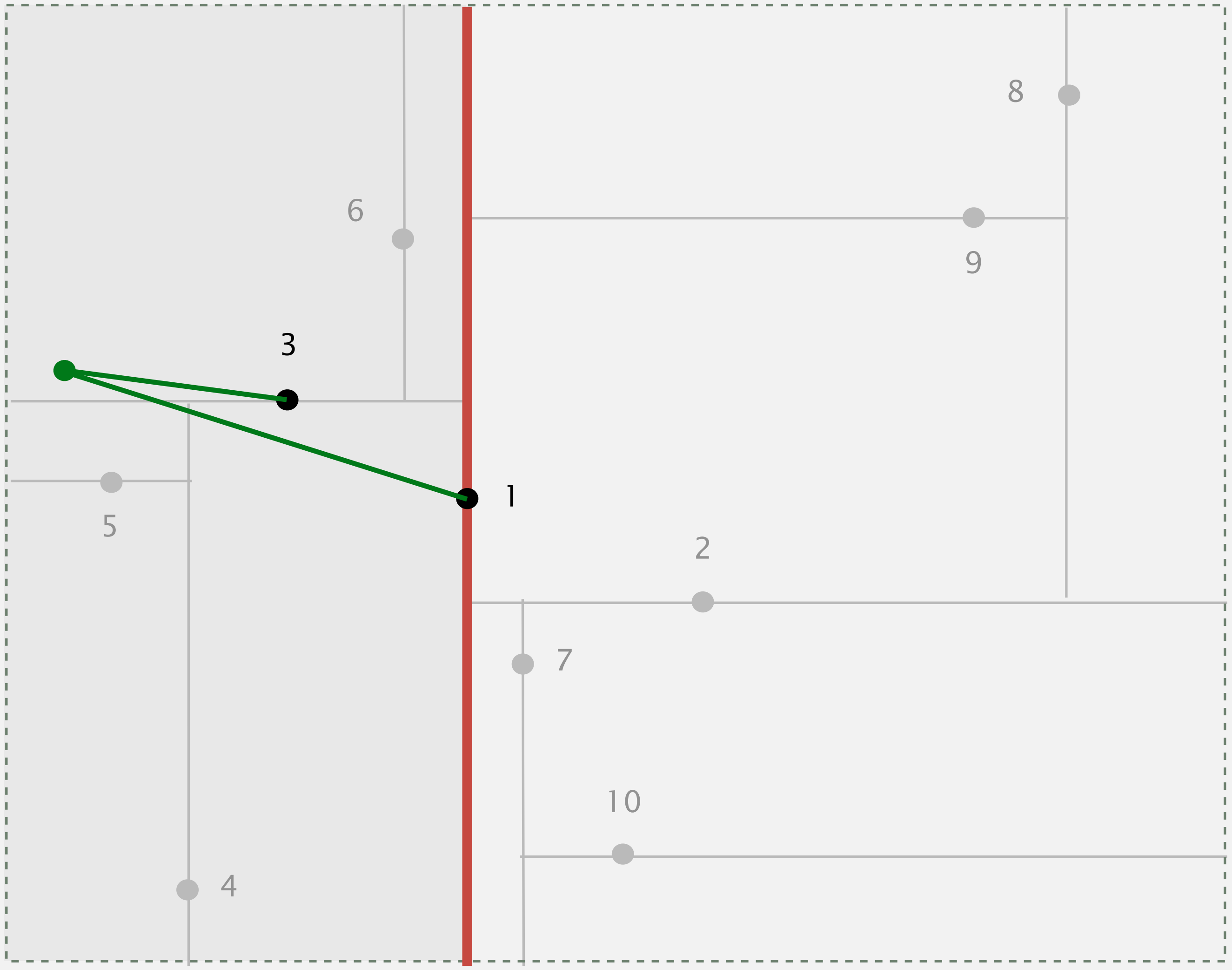


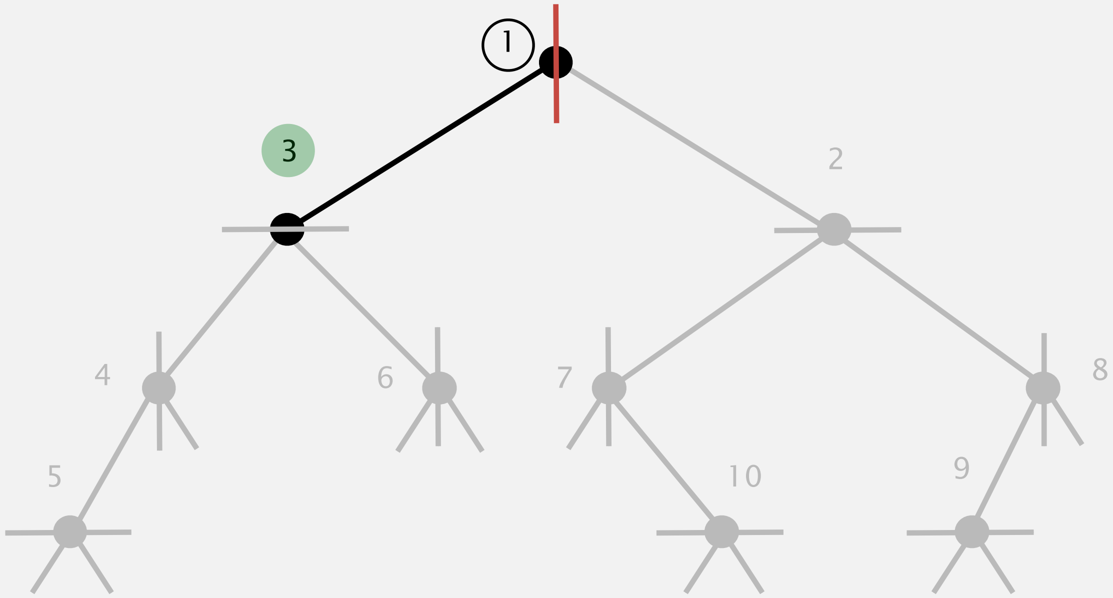


query point is to the left of splitting line
search left subtree first

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).





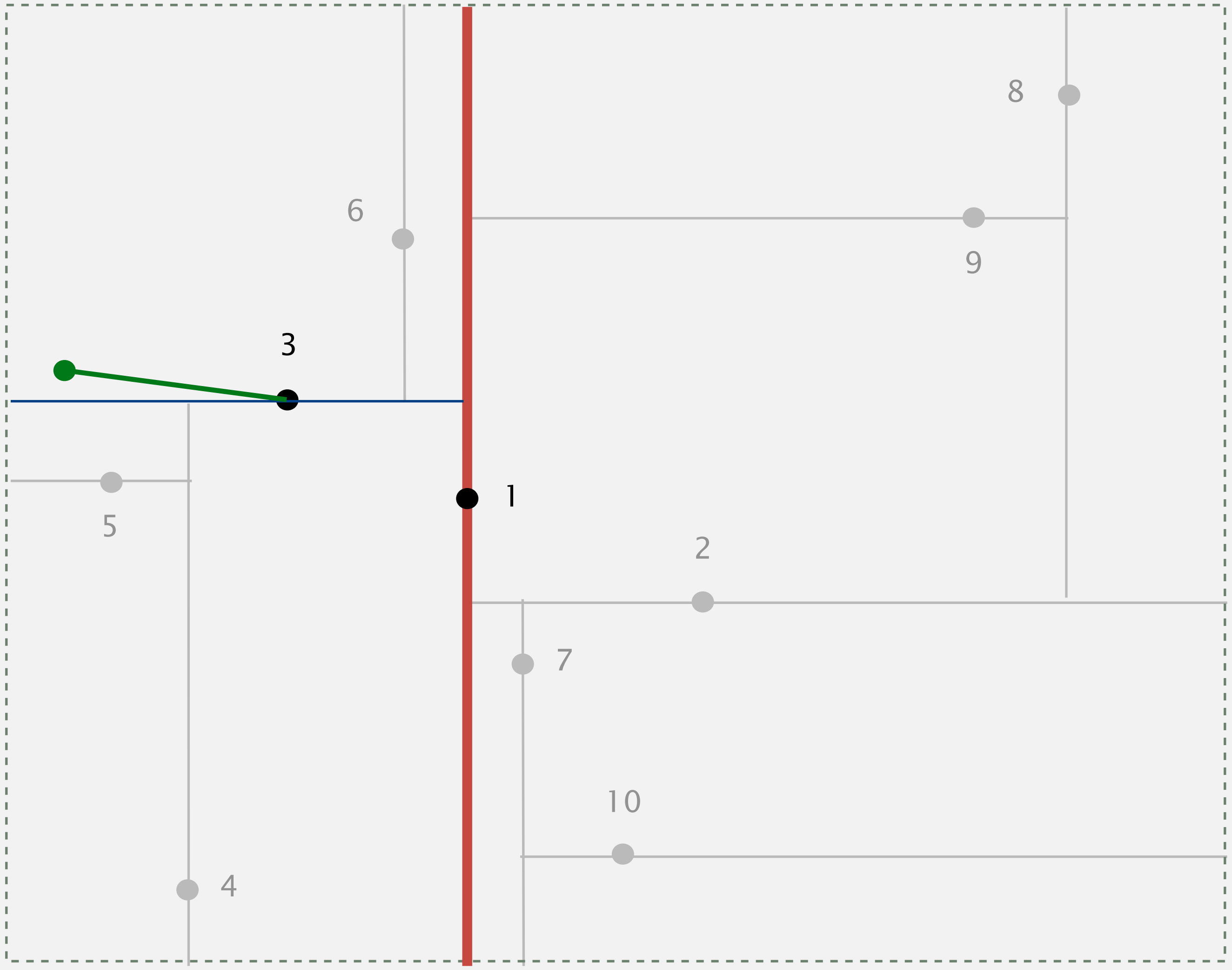
search left subtree

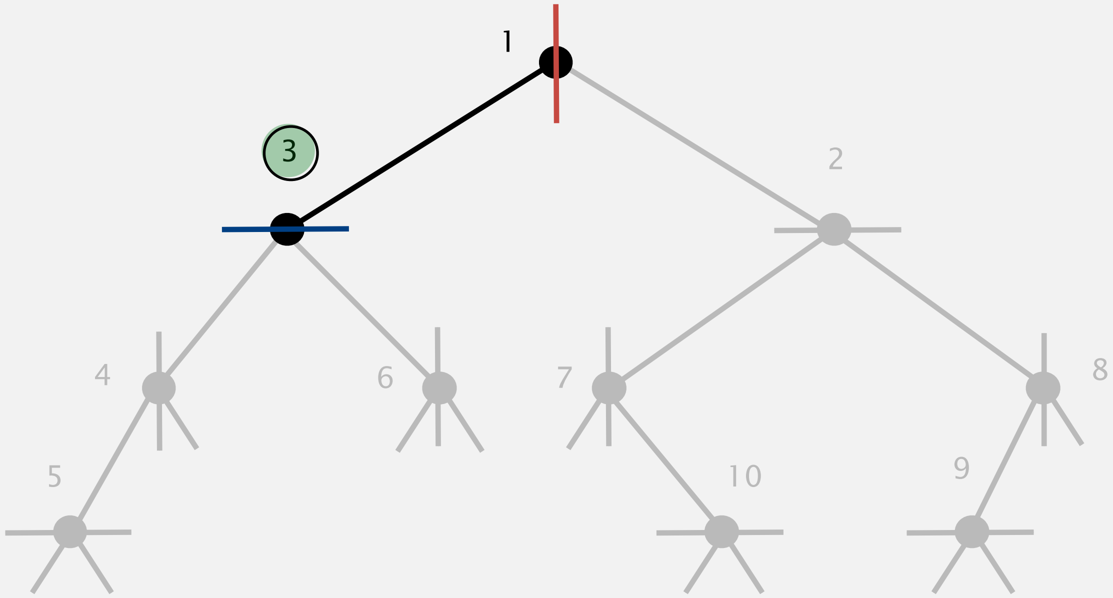
compute distance from query point to 3

(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).

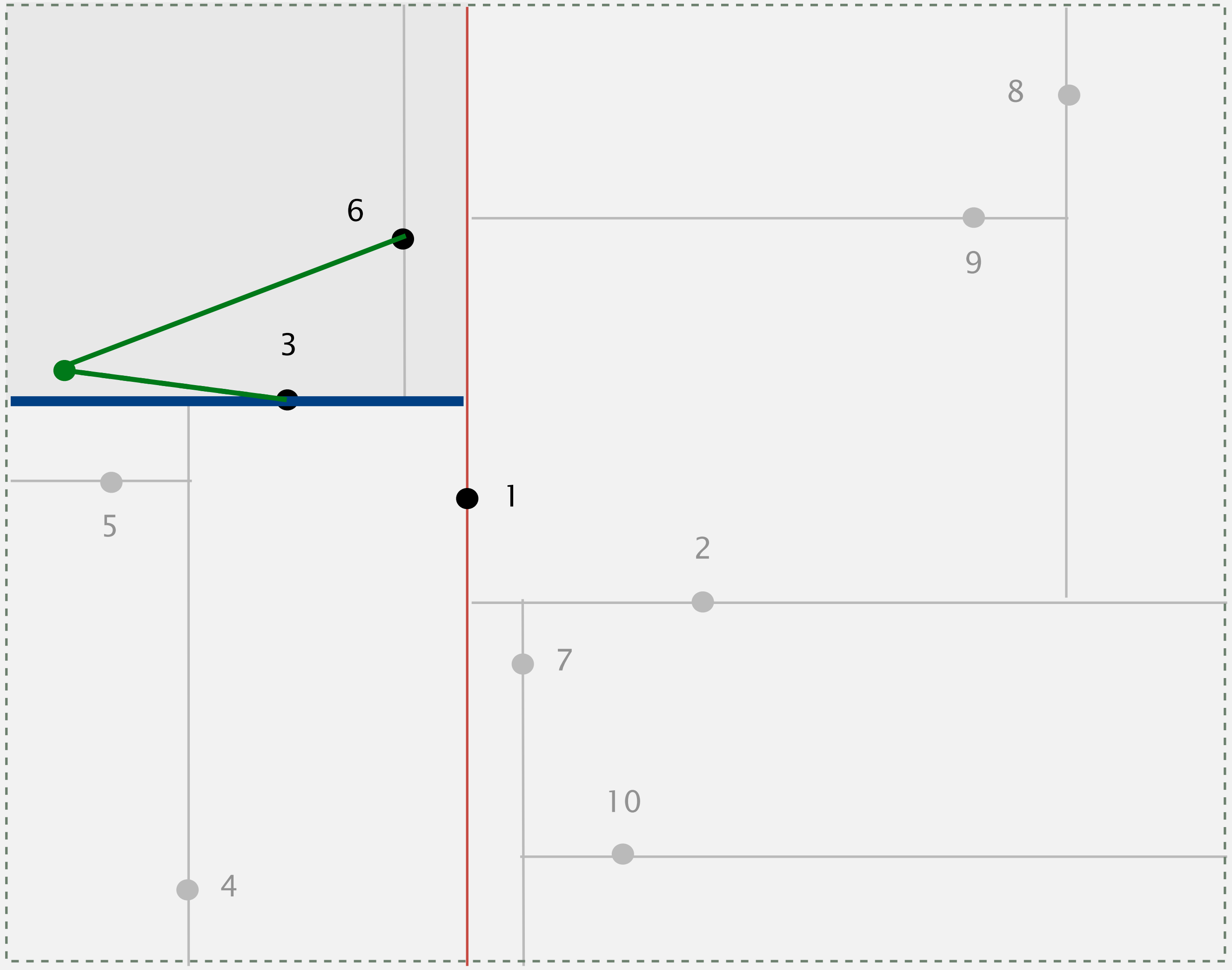


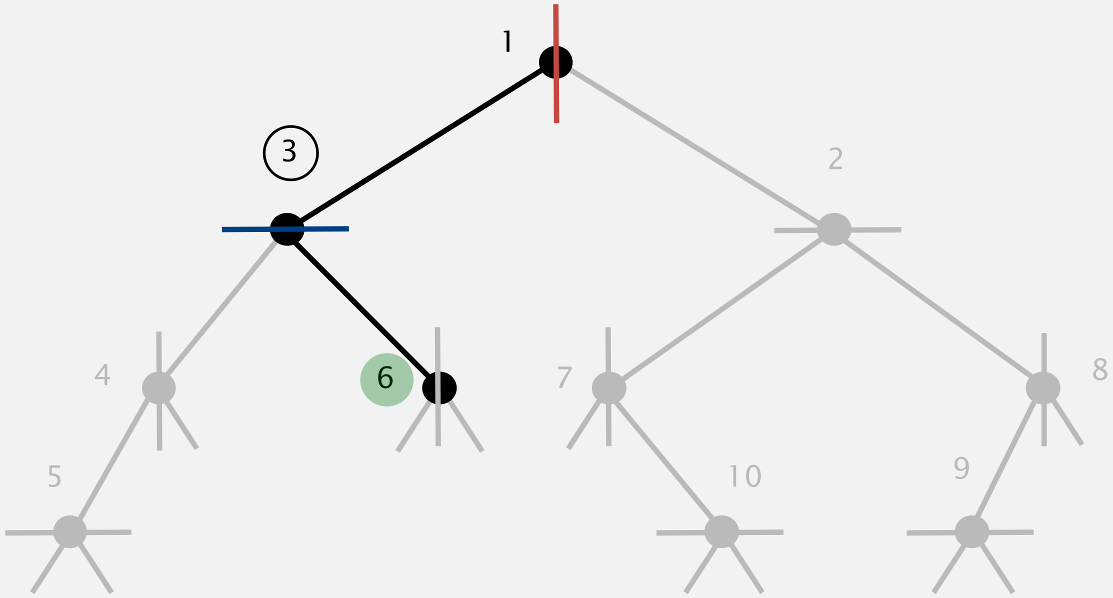


query point is above splitting line
search top subtree first

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).



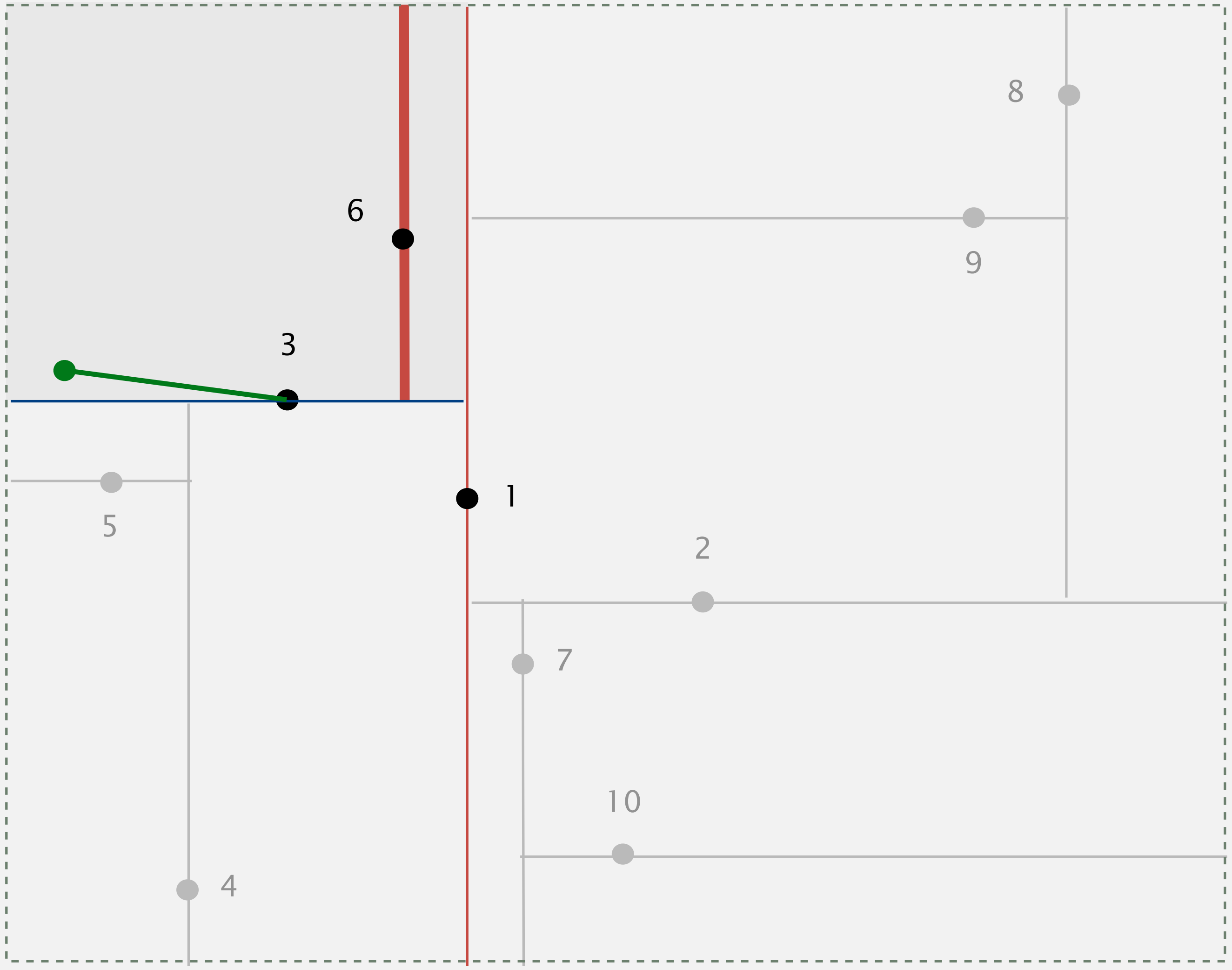


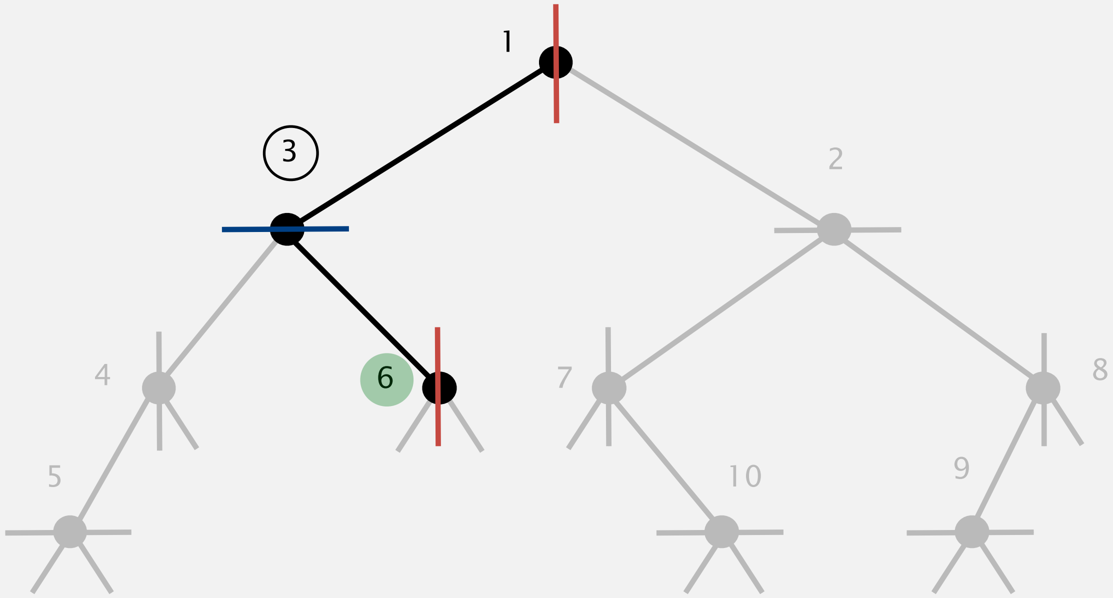
search top subtree

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).

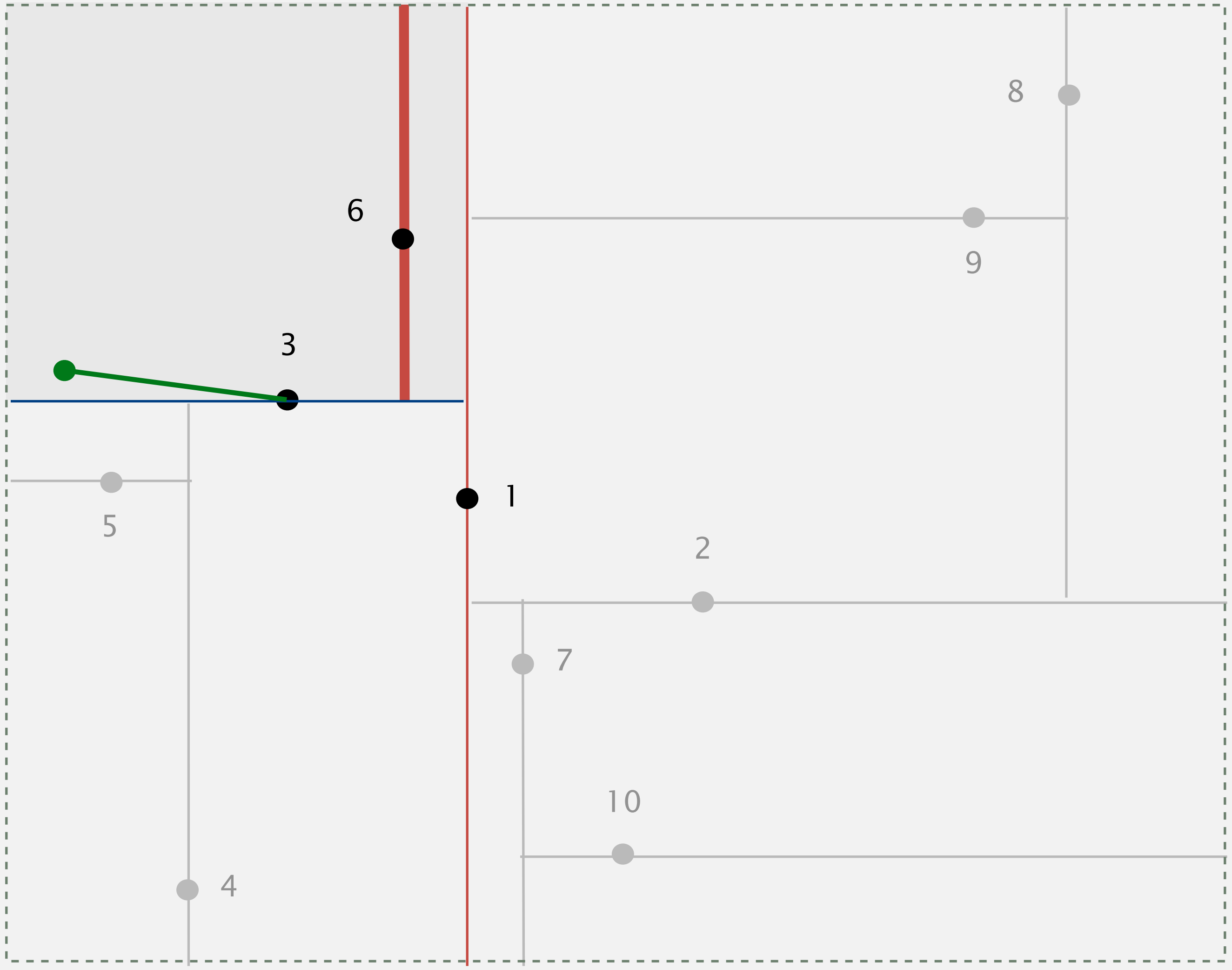


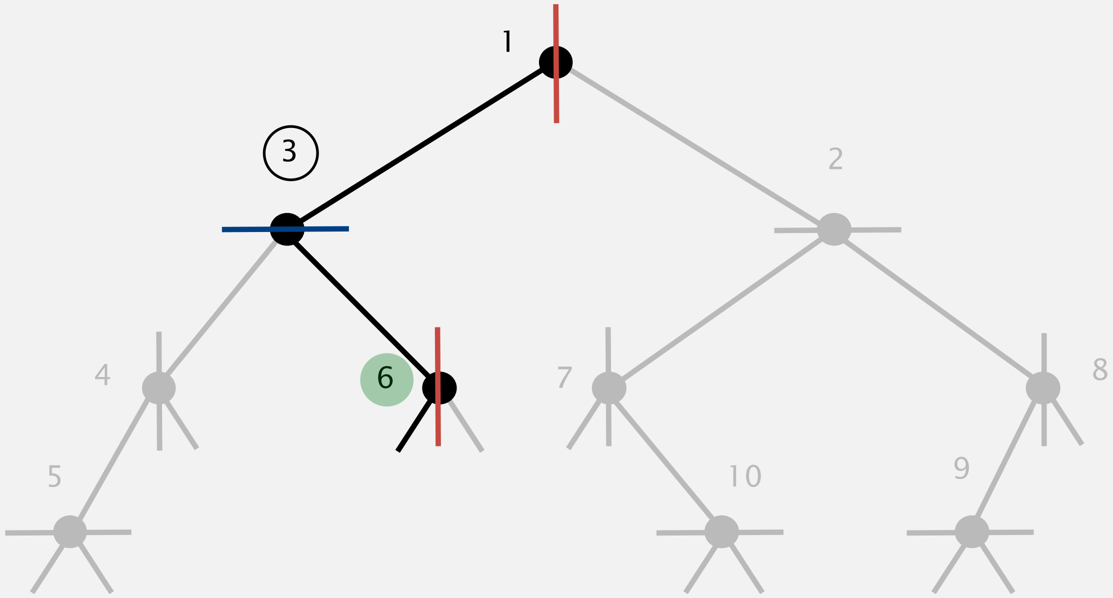


query point is to left of splitting line
search left subtree first

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).

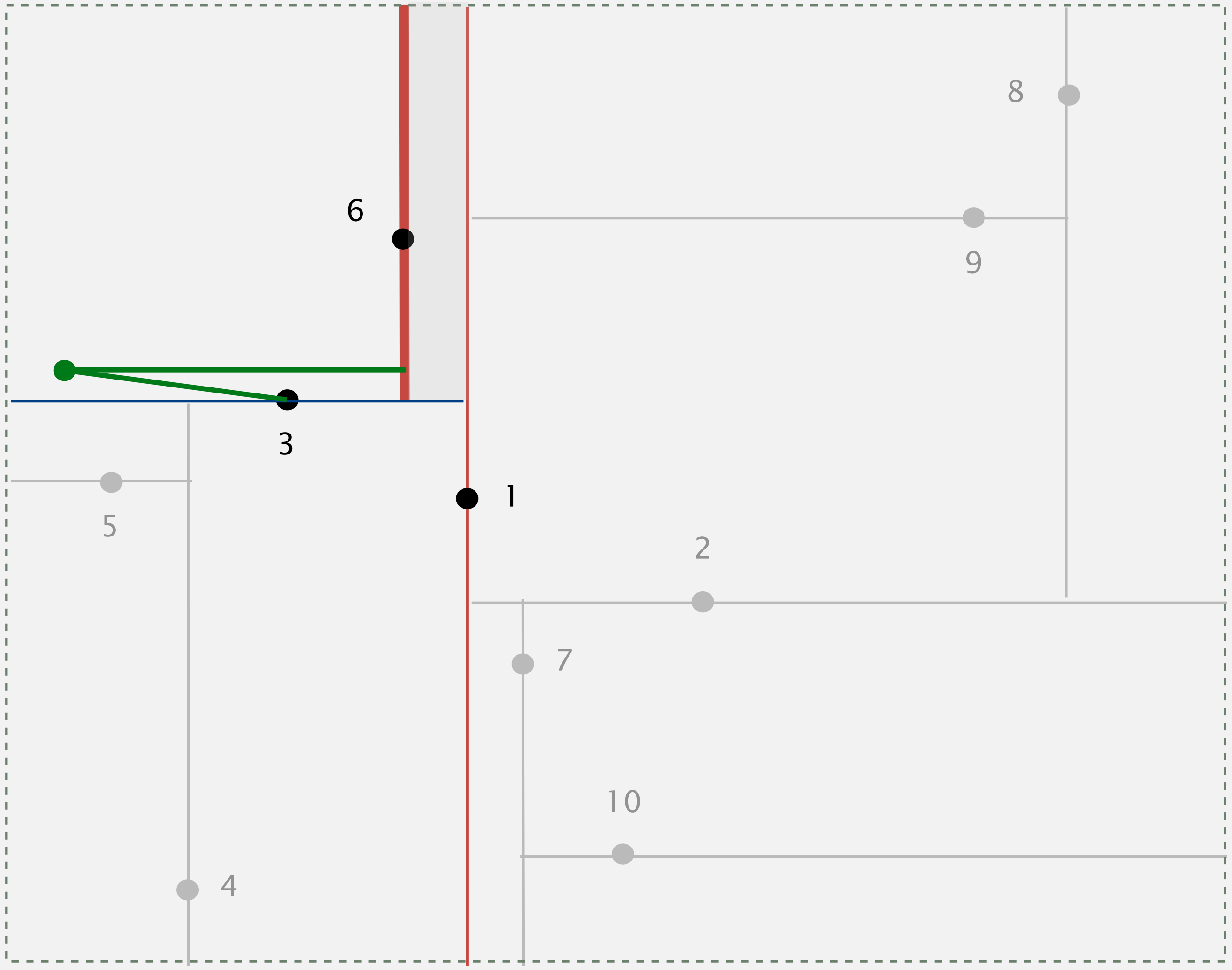


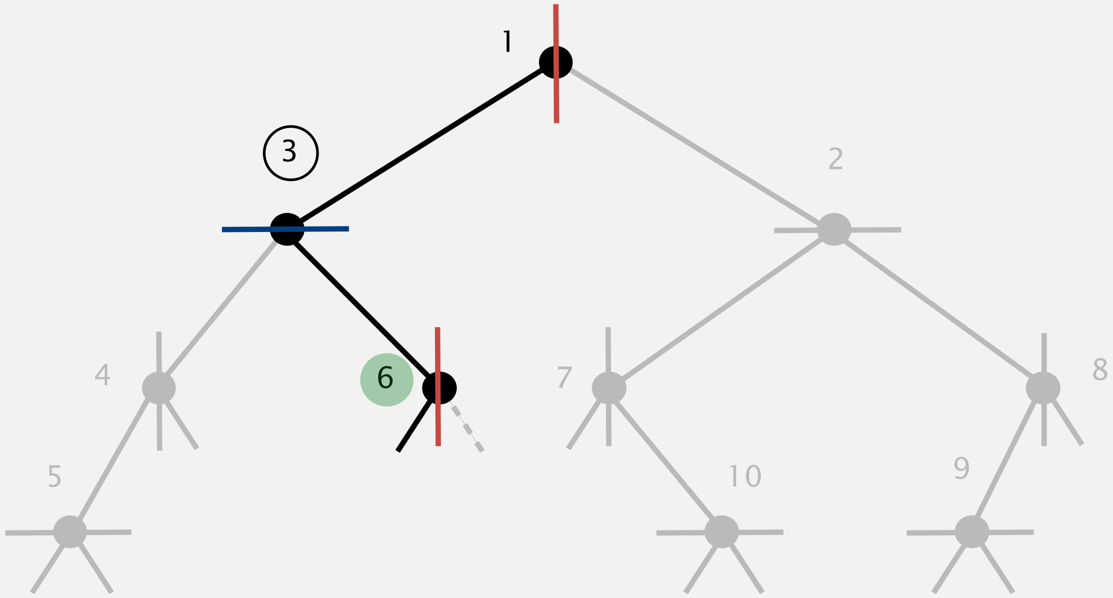


search left subtree
return since empty

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).

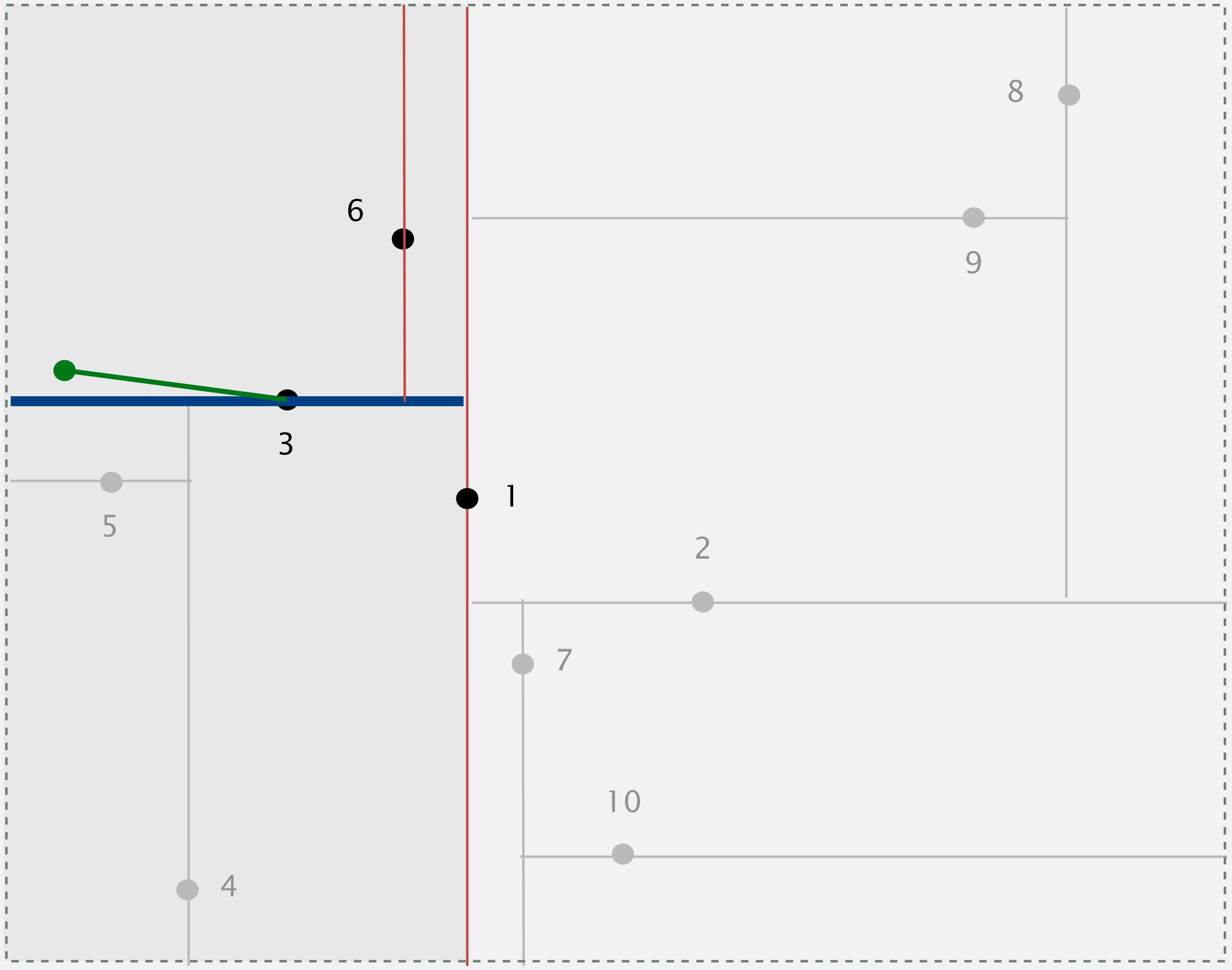


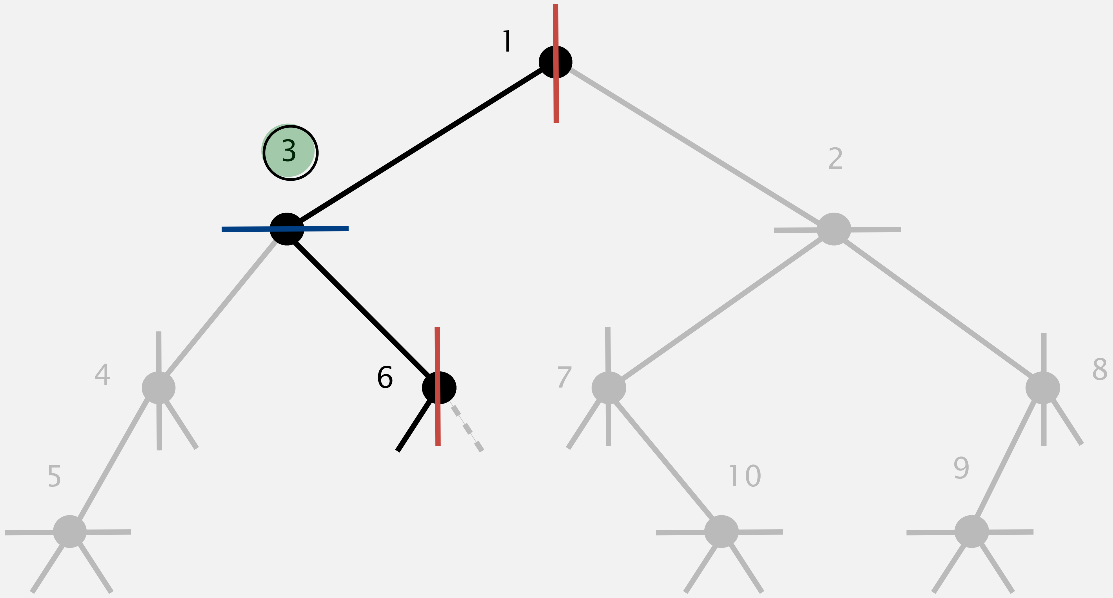


search right subtree
prune since nearest neighbor
can't be here

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).

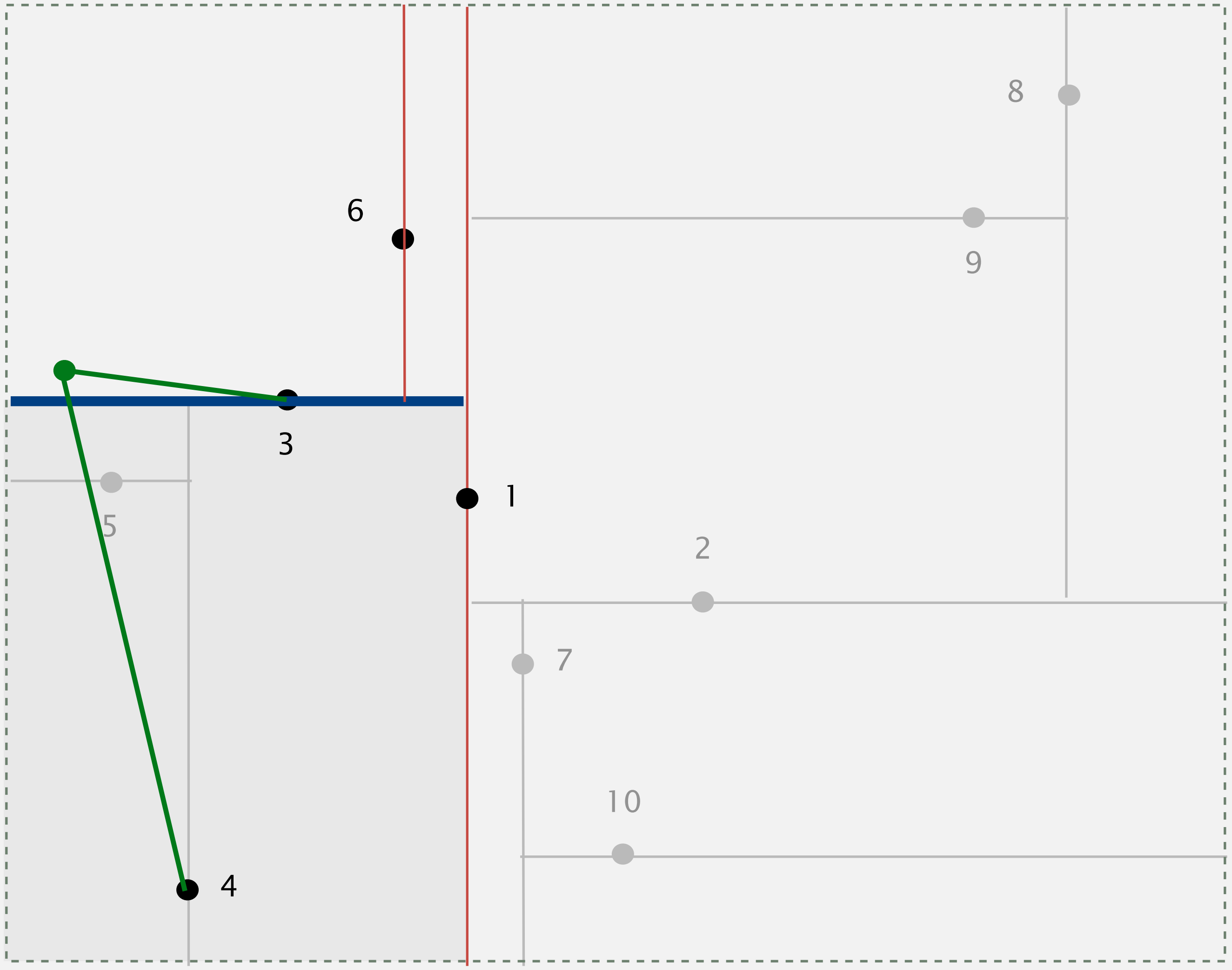


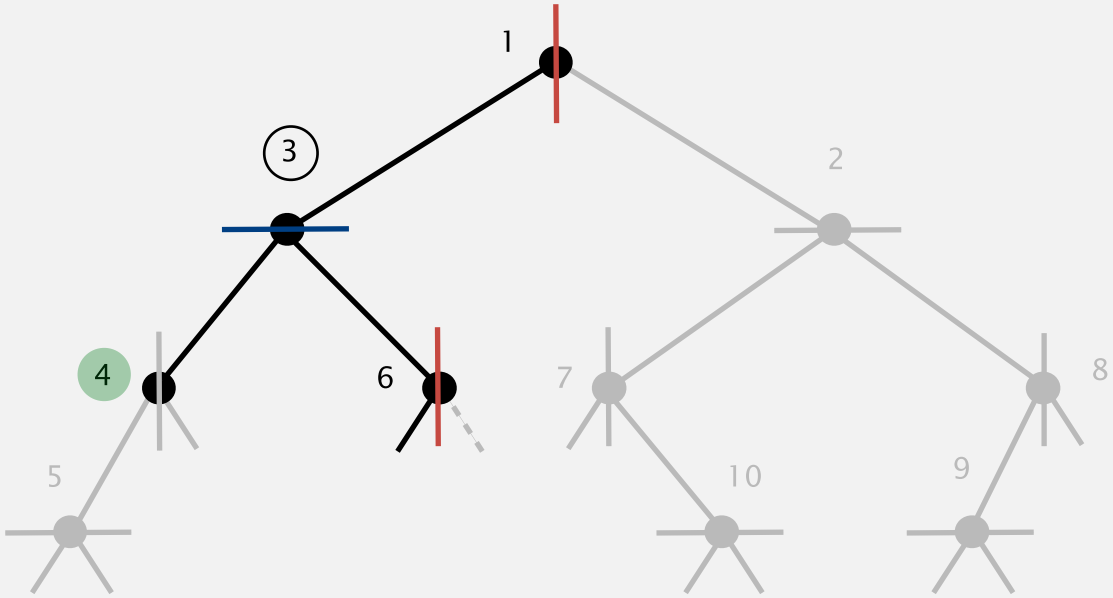


return from function call
search bottom subtree next

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).

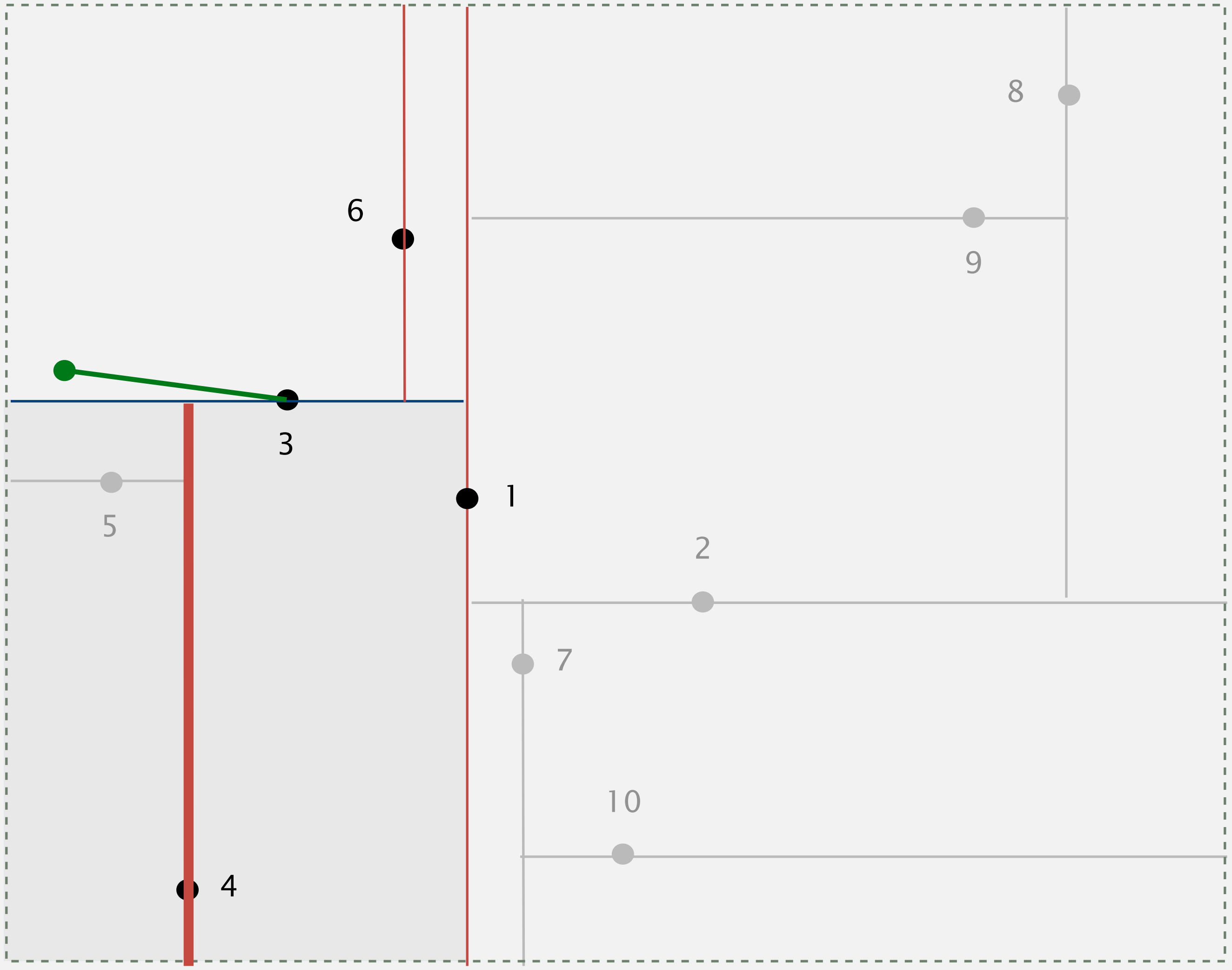


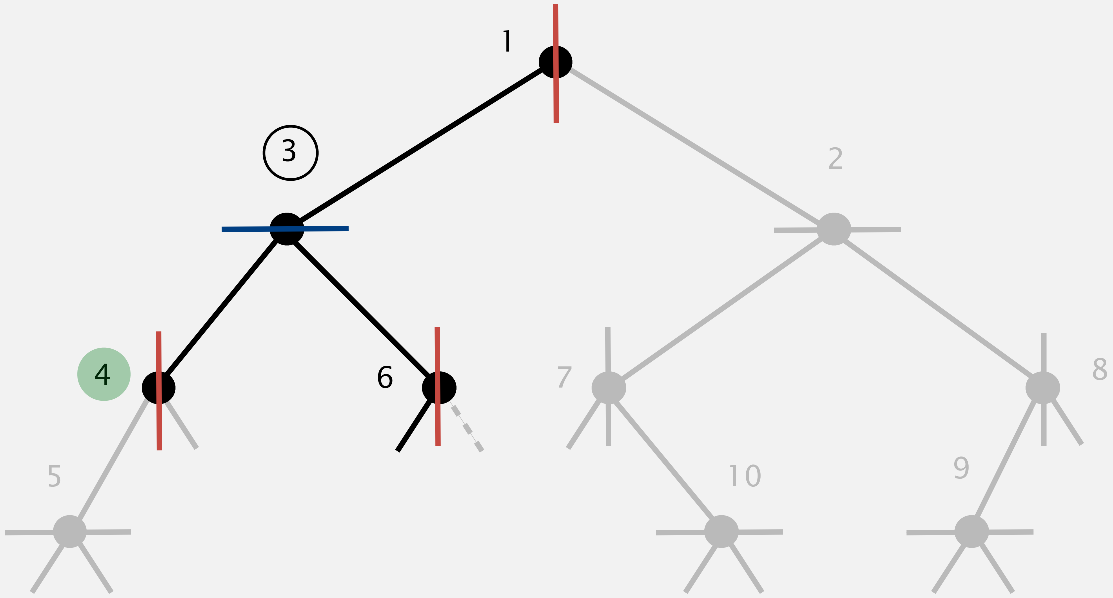


search bottom subtree
compute distance from query point to 4

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).

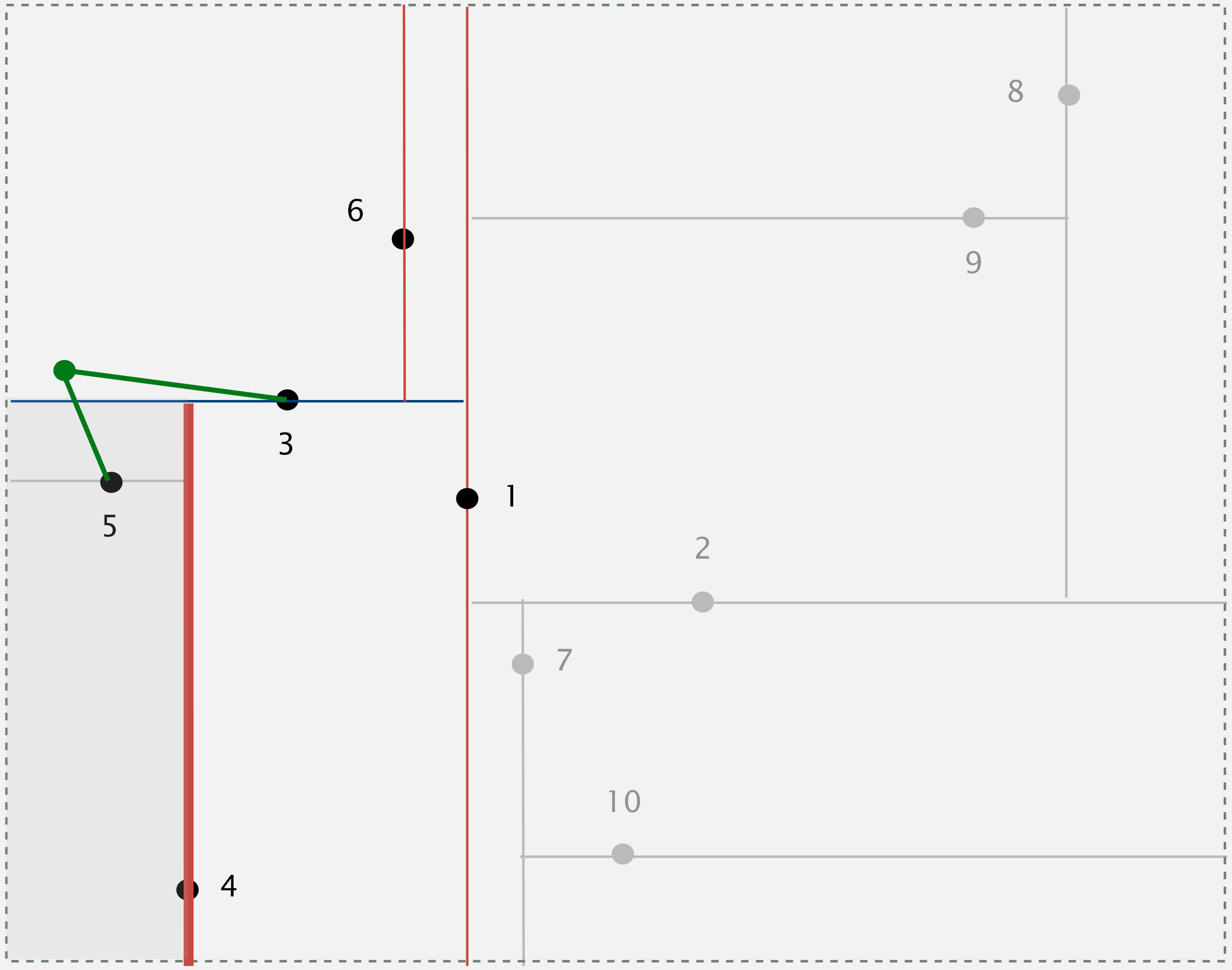


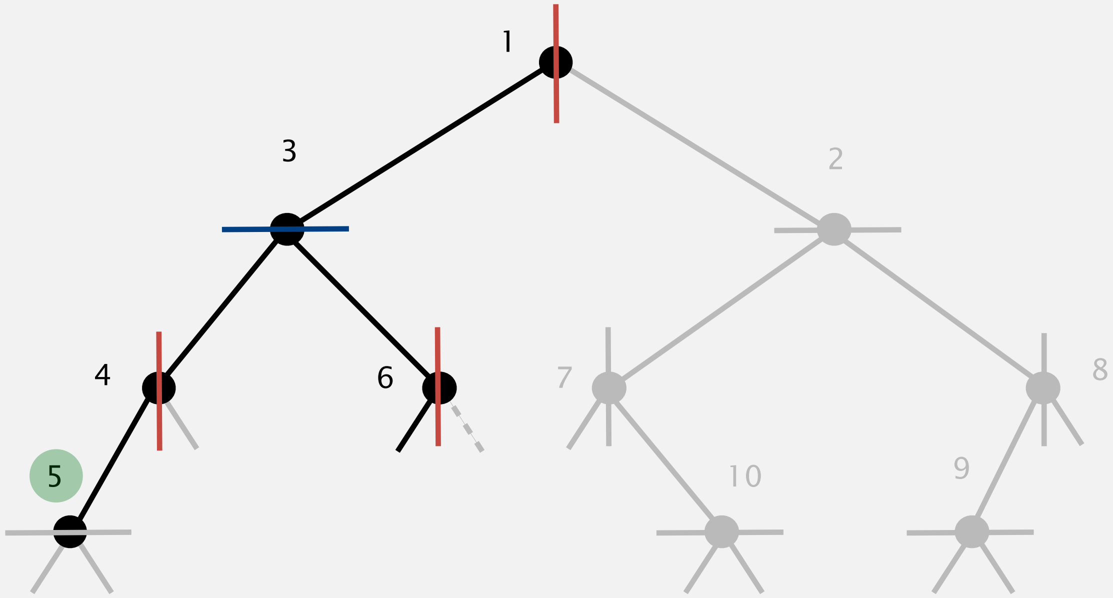


query point is to left of splitting line
search left subtree first

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).





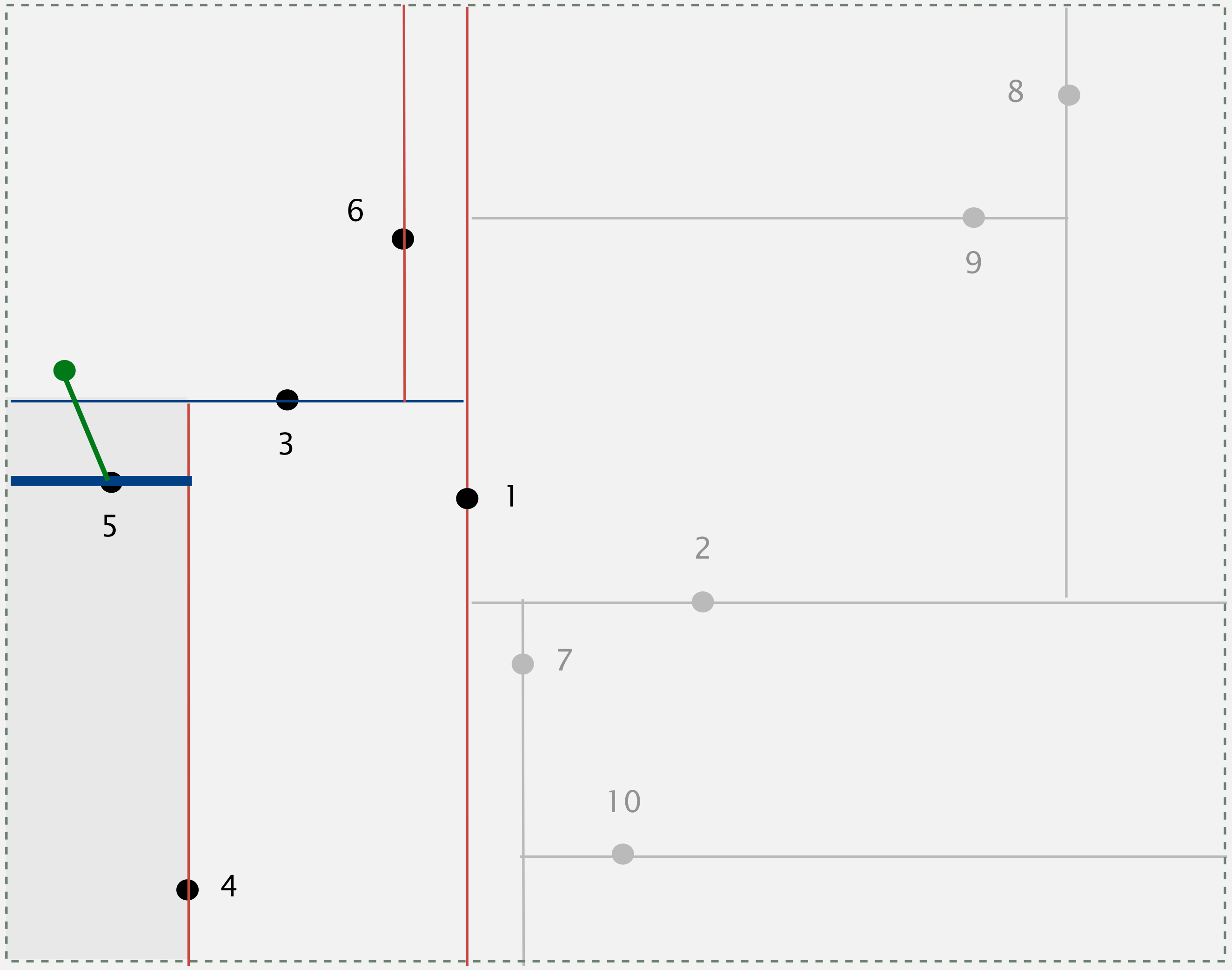
search left subtree

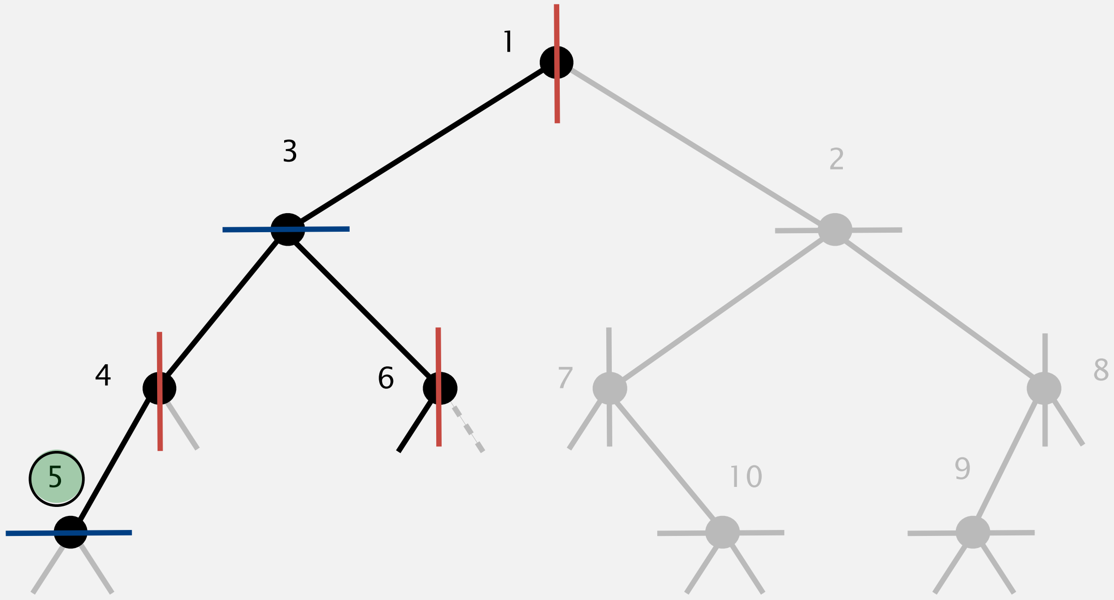
compute distance from query point to 5

(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).

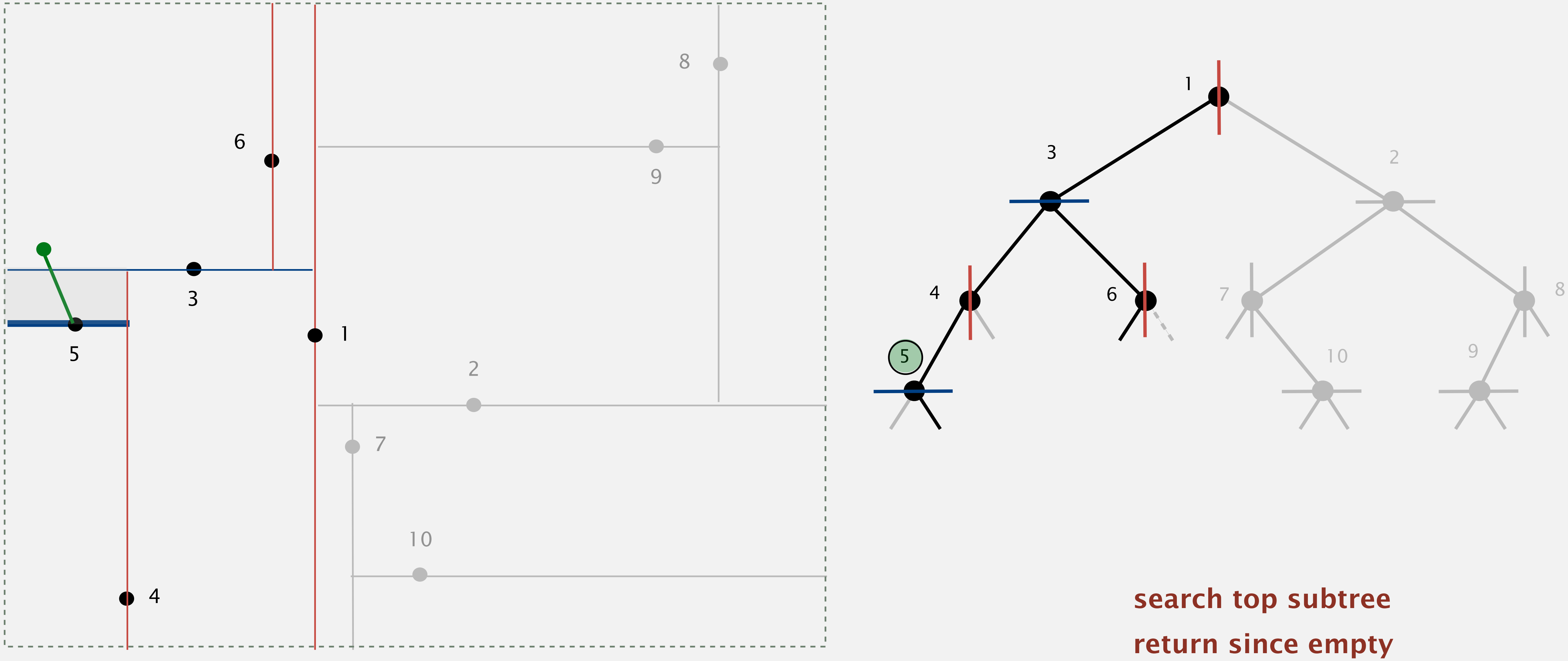




query point is above splitting line
search top subtree first

2d tree demo: nearest neighbor

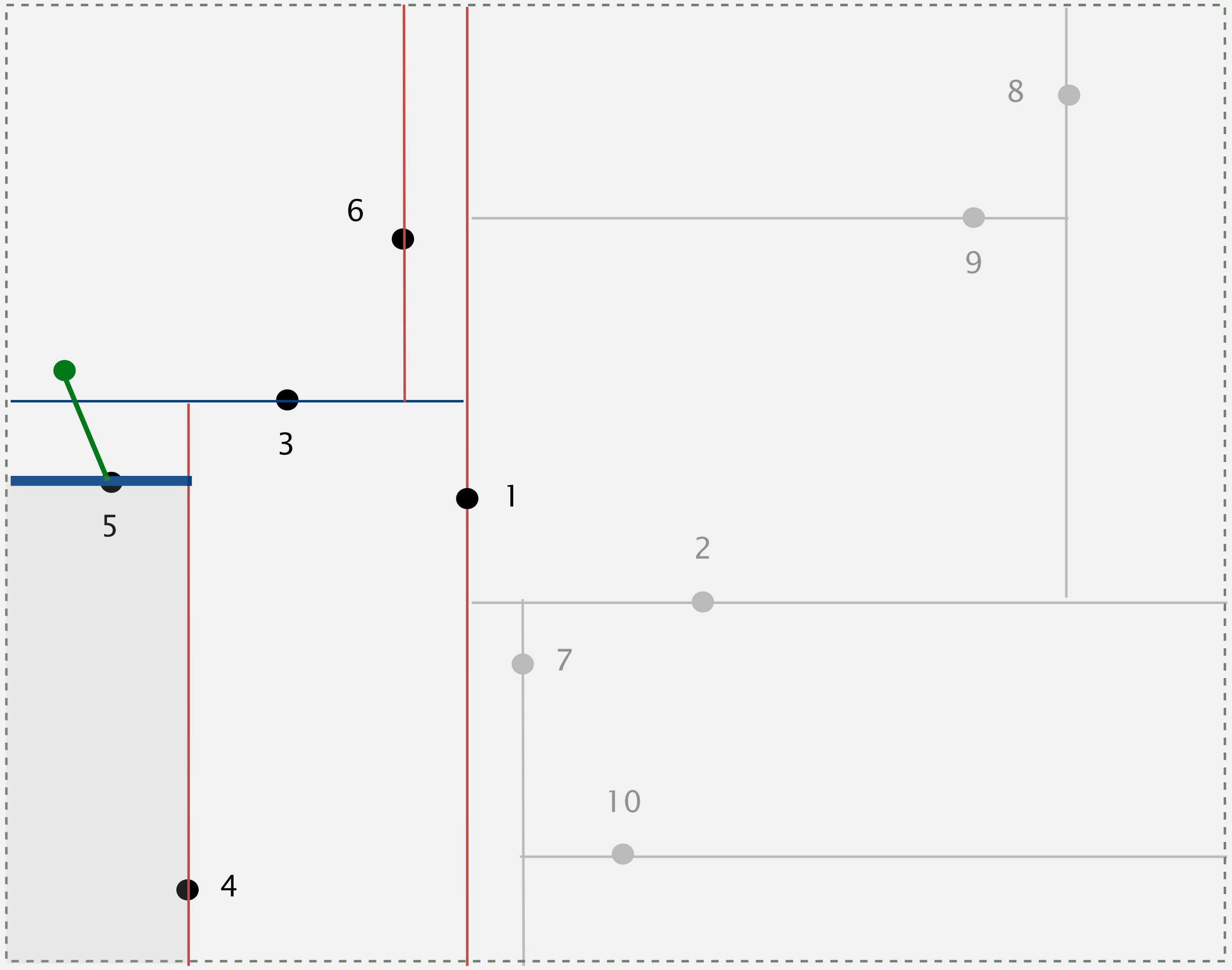
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).

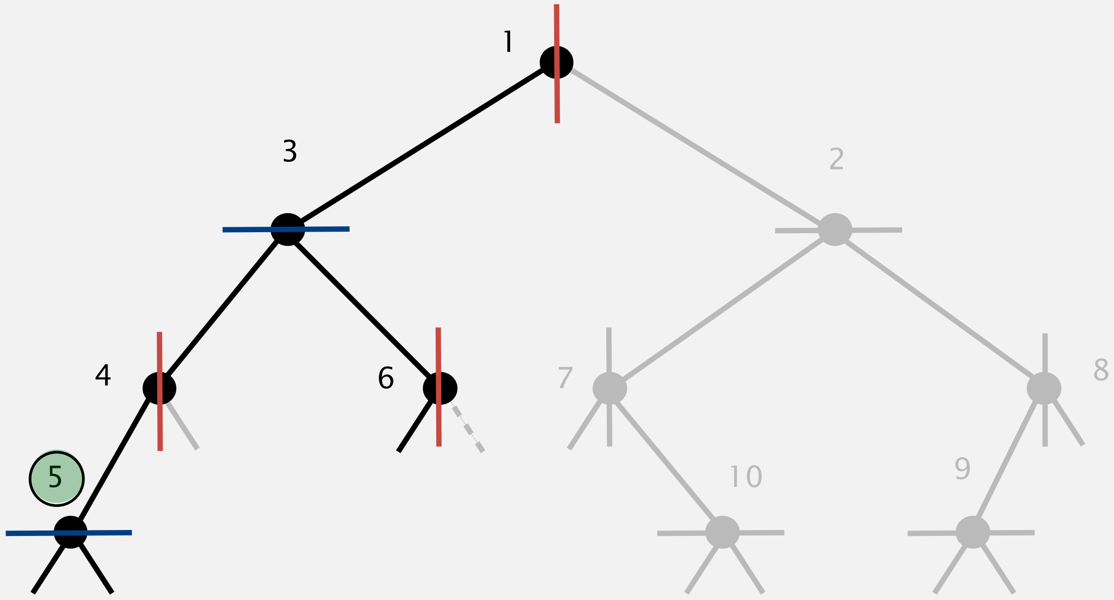


search top subtree
return since empty

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).



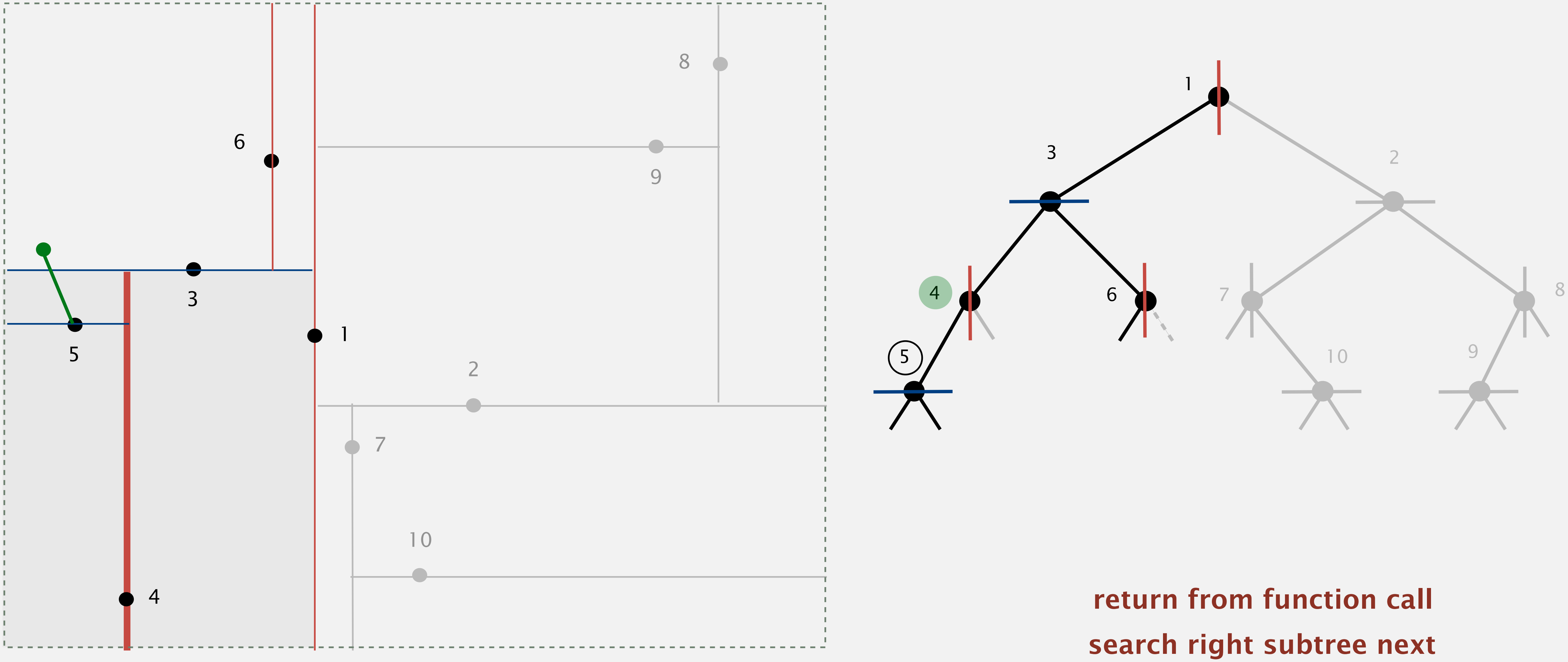


search bottom subtree

return since empty

2d tree demo: nearest neighbor

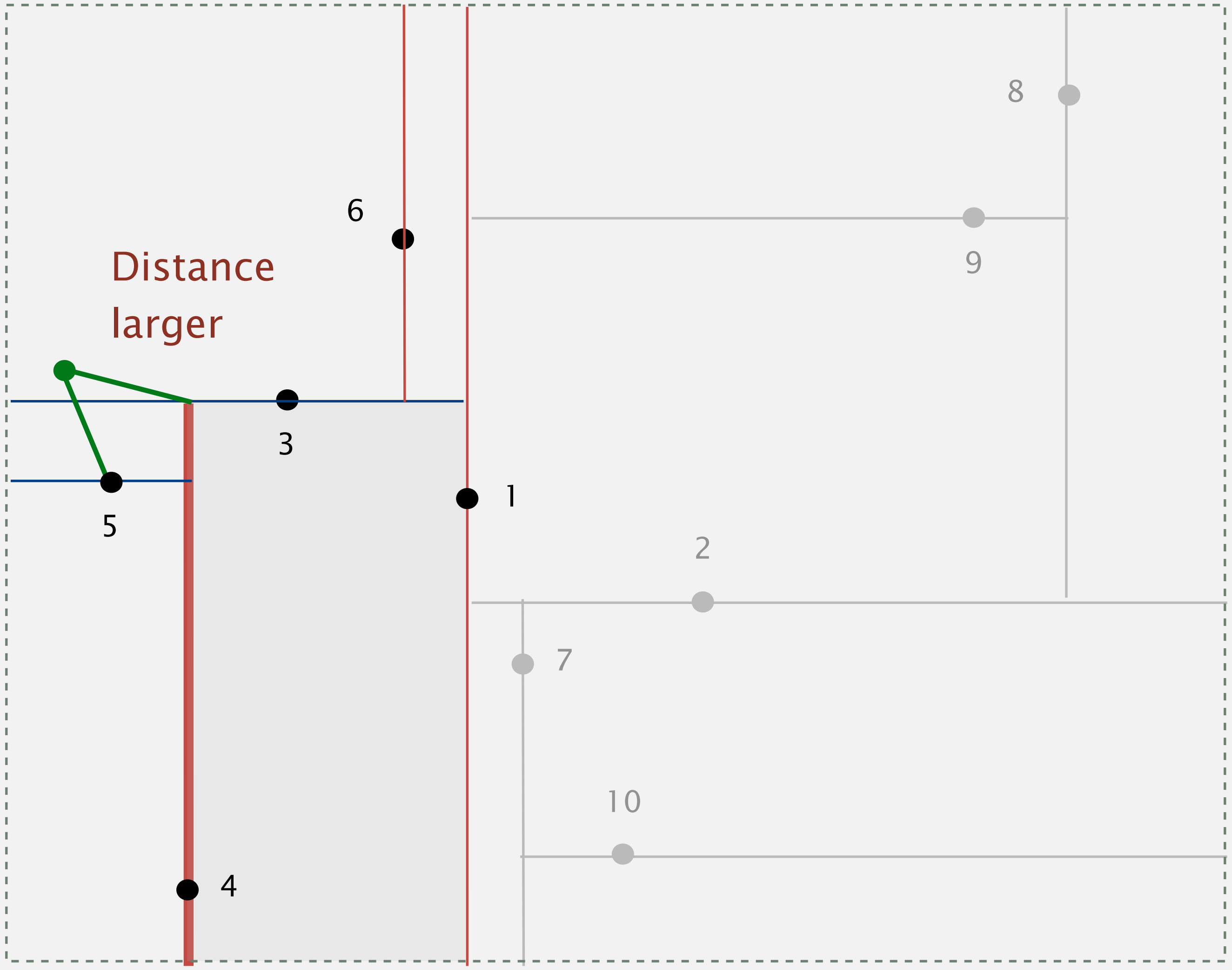
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).

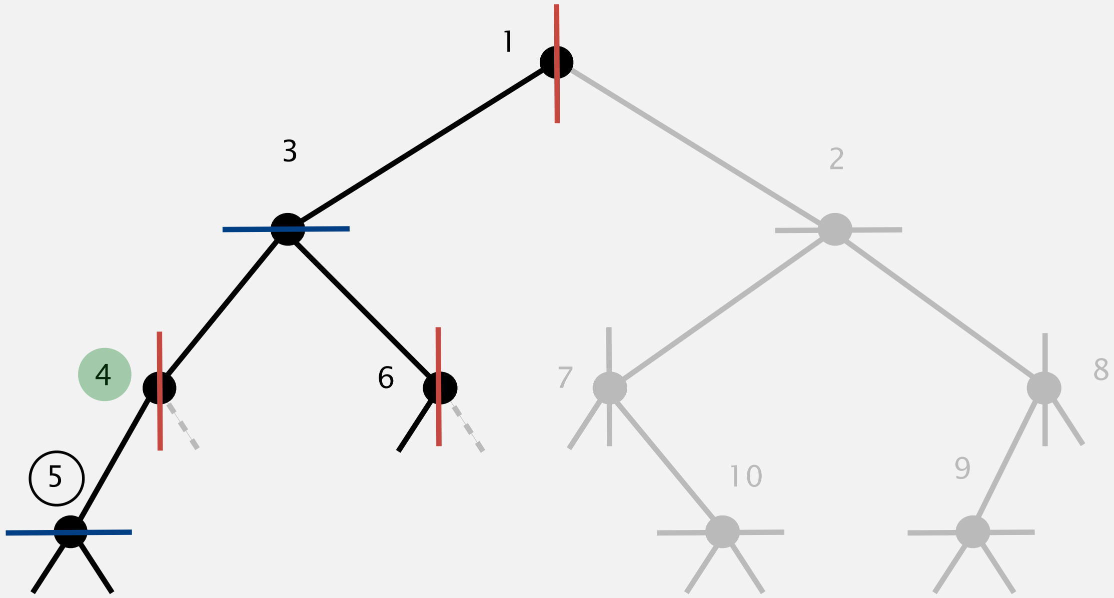


return from function call
search right subtree next

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).

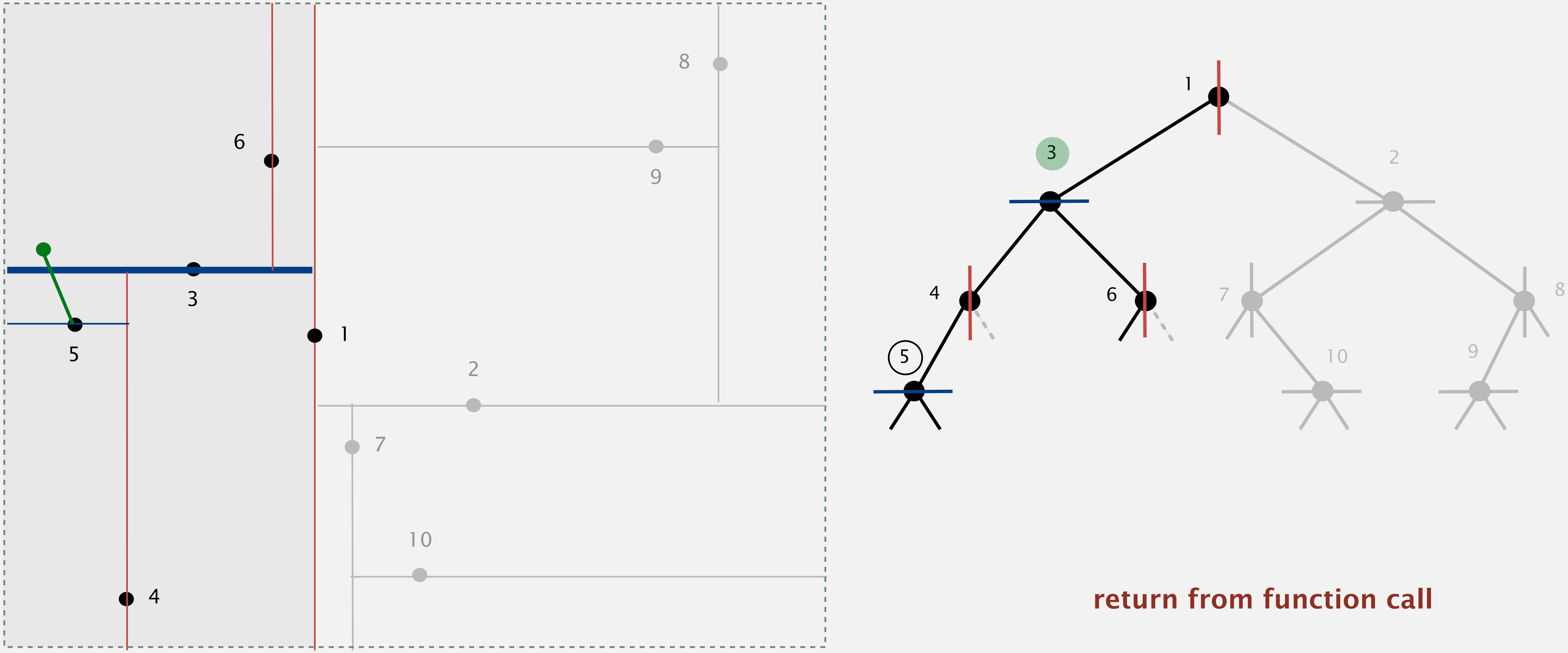




search right subtree
prune since nearest neighbor
can't be here
(drawing not quite to scale)

2d tree demo: nearest neighbor

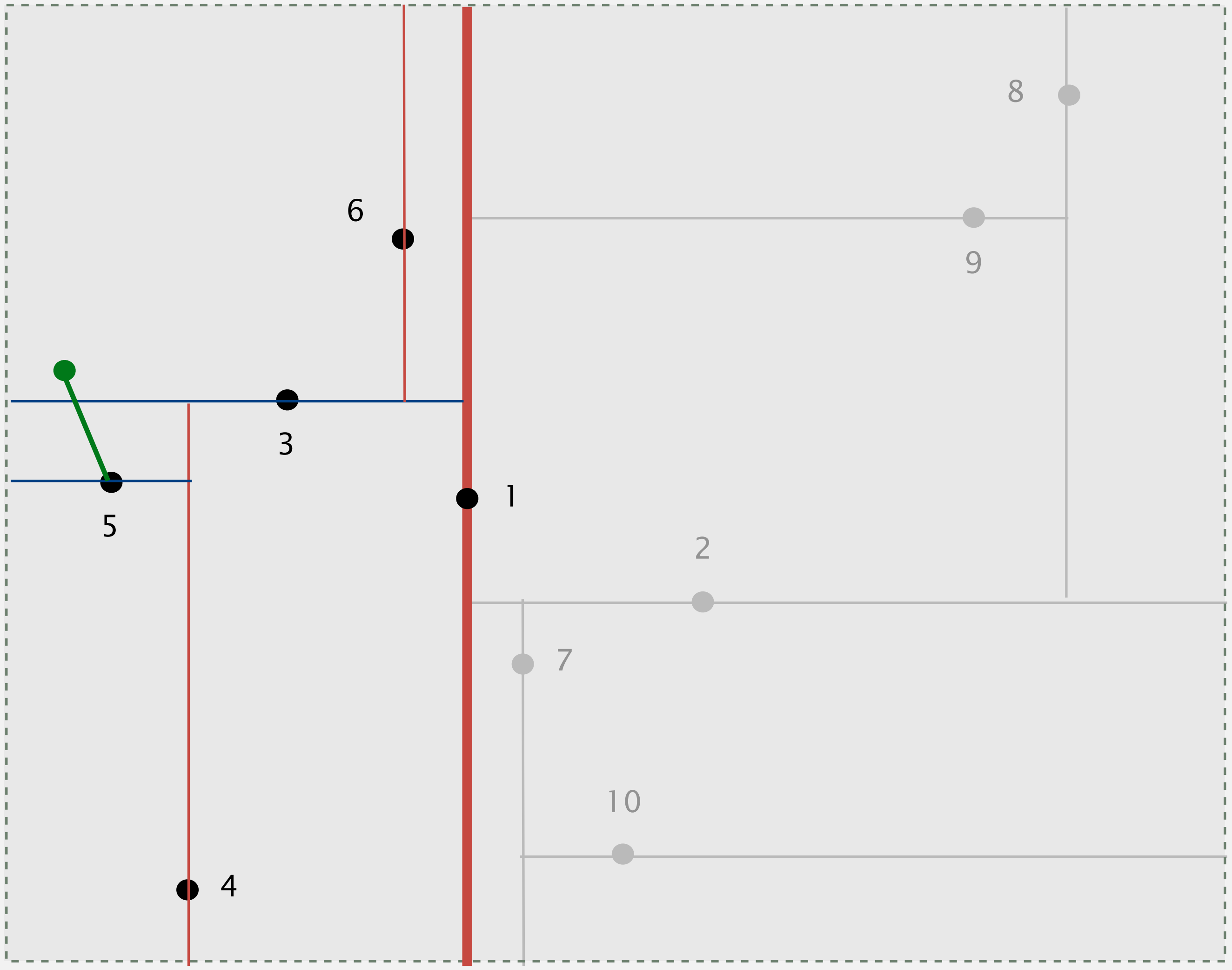
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).

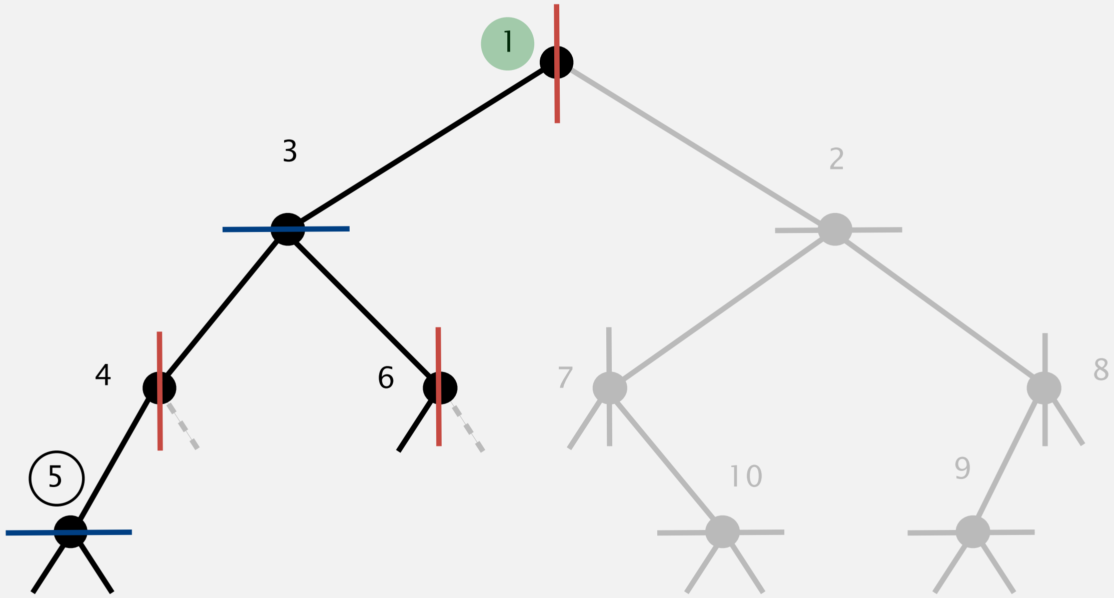


return from function call

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).

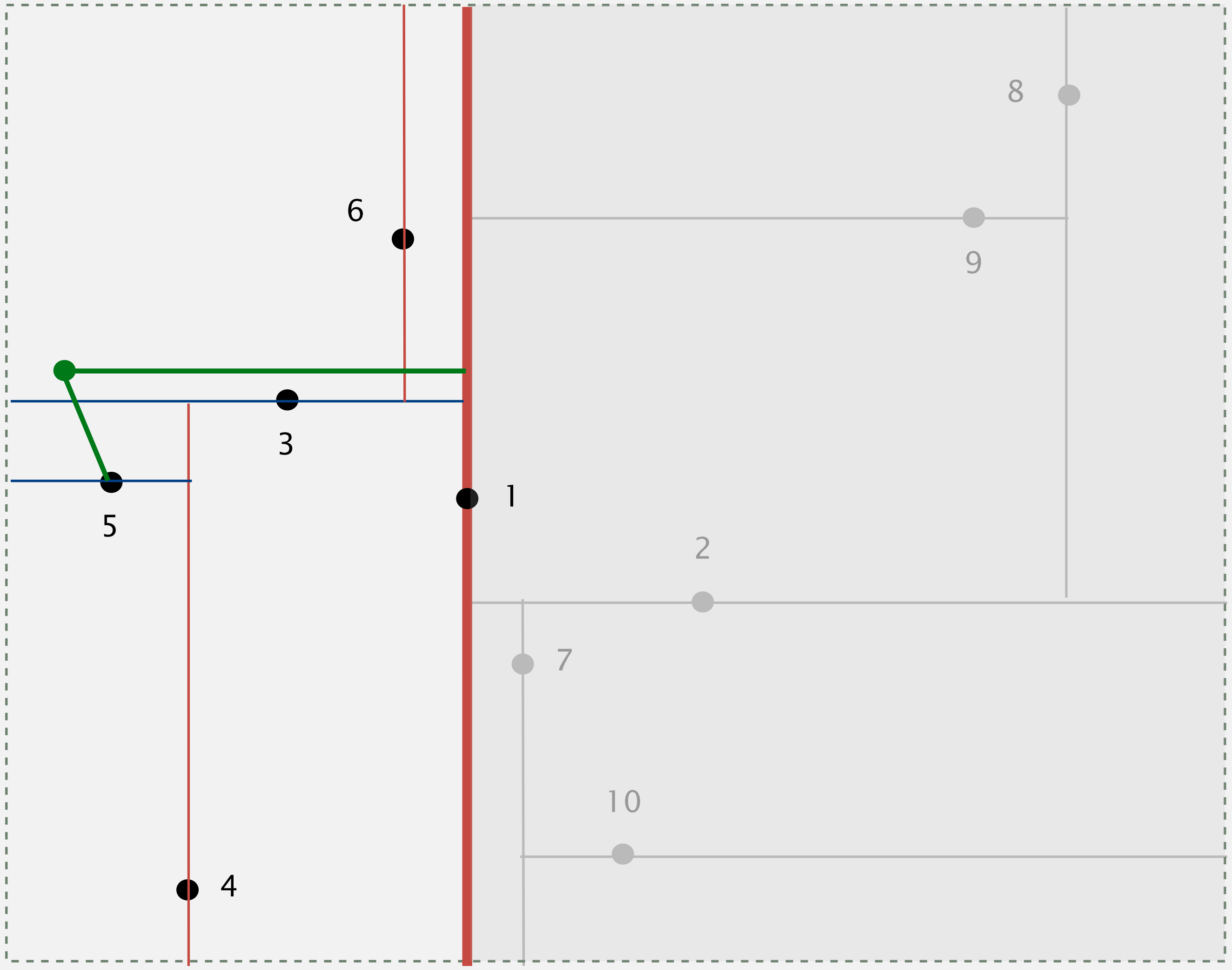


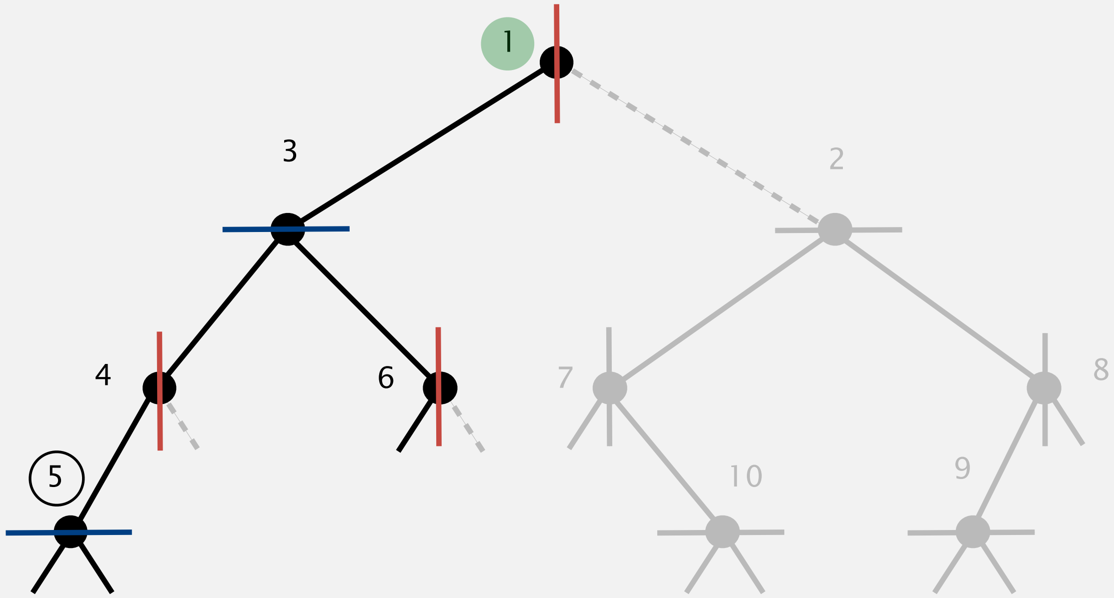


return from function call
search right subtree next

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).

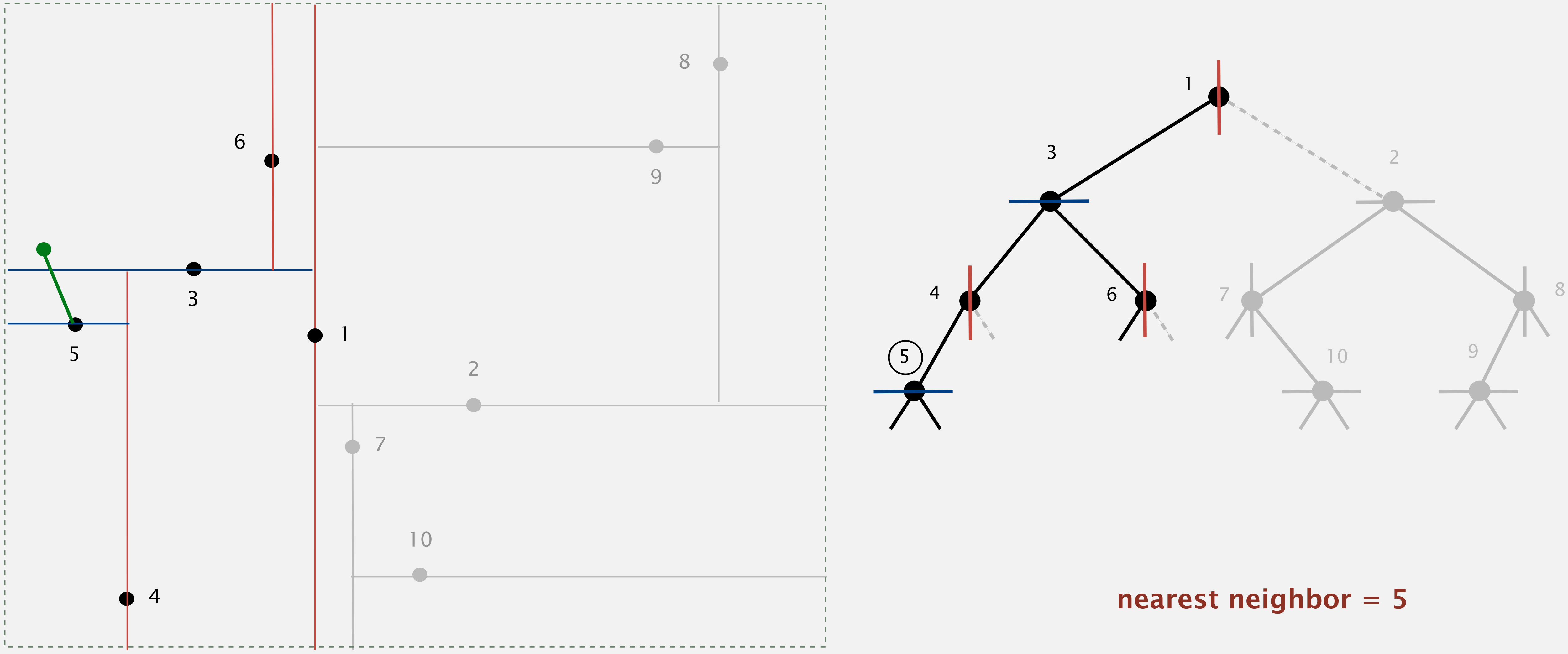




search right subtree
prune since nearest neighbor
can't be here

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).



nearest neighbor = 5

Flocking birds

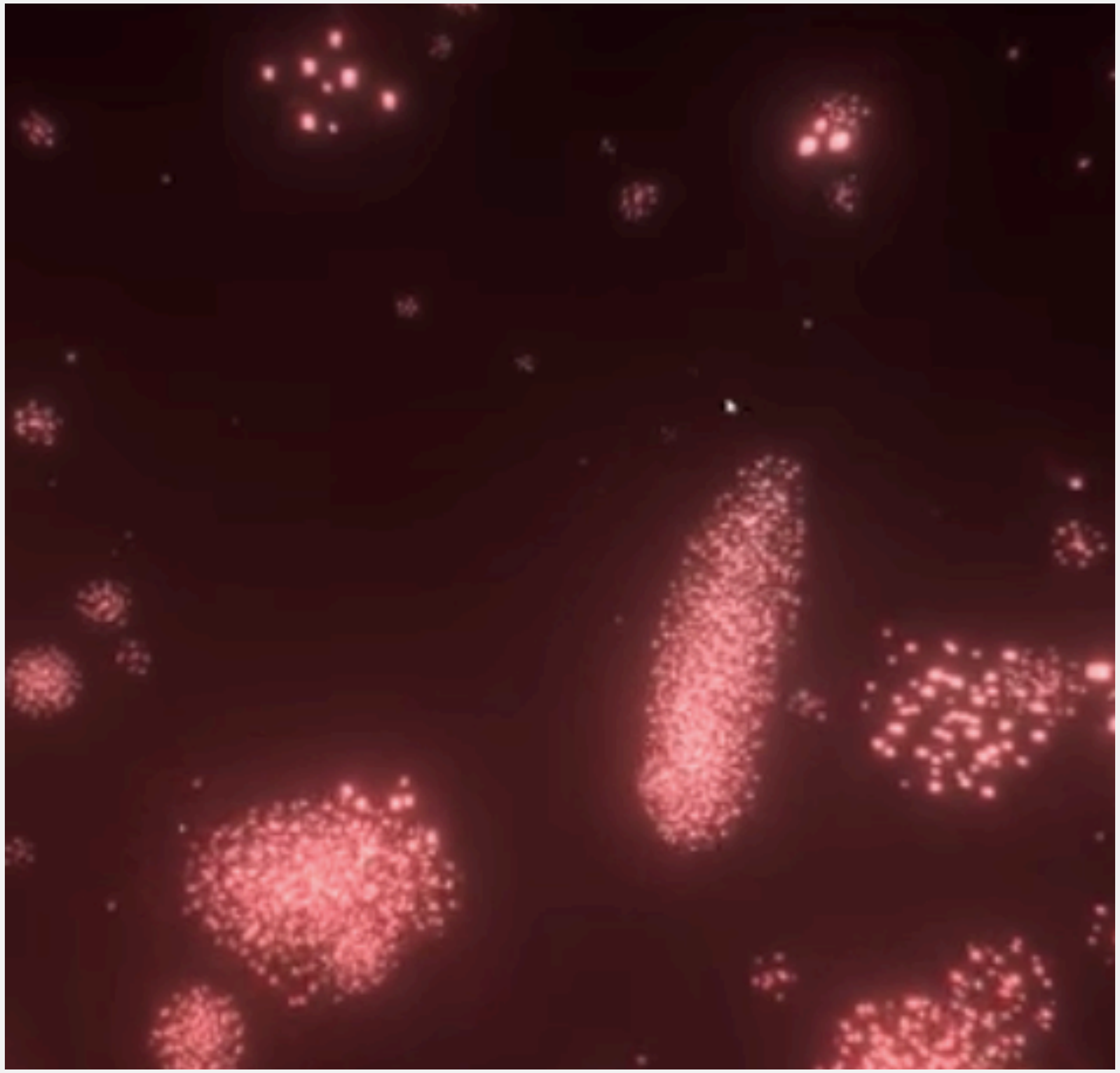
Q. What "natural algorithm" do starlings, migrating geese, starlings, cranes, bait balls of fish, and flashing fireflies use to flock?

http://www.youtube.com/watch?v=XH-groCeKbE

Flocking boids [Craig Reynolds, 1986]

Boids. Three simple rules lead to complex emergent flocking behavior:

- Collision avoidance: point away from k nearest boids.
- Flock centering: point towards the center of mass of k nearest boids.
- Velocity matching: update velocity to the average of k nearest boids.



https://www.youtube.com/watch?v=SJyRkeq4Mgw

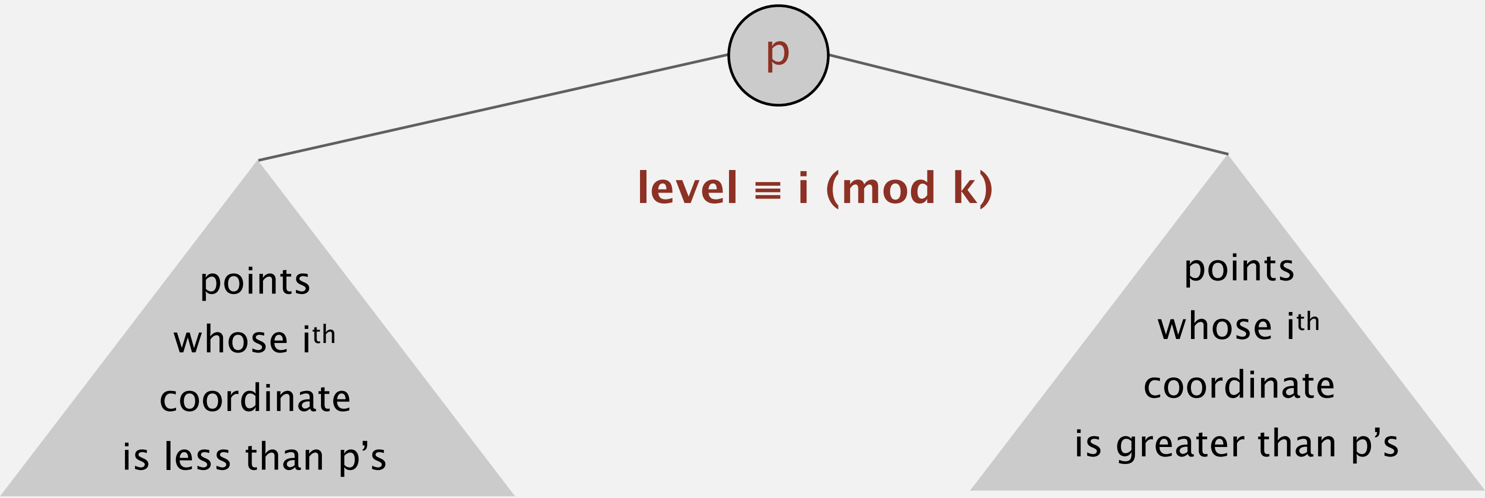
K-D TREE K DIMENSIONS

Kd tree

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

- Just cycle through each dimension at each level of there tree

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

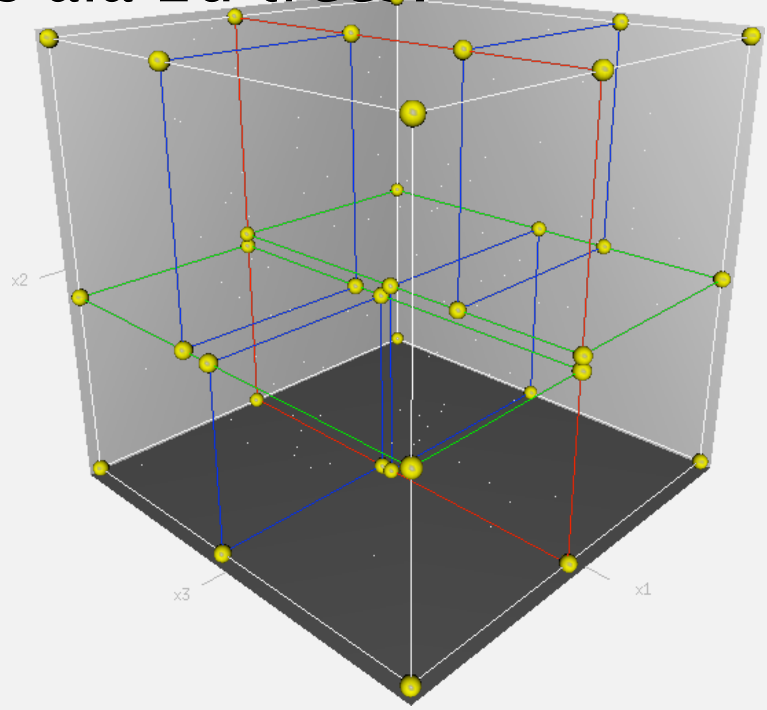


p

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.

Jon Bentley

INTERVAL SEARCH TREES

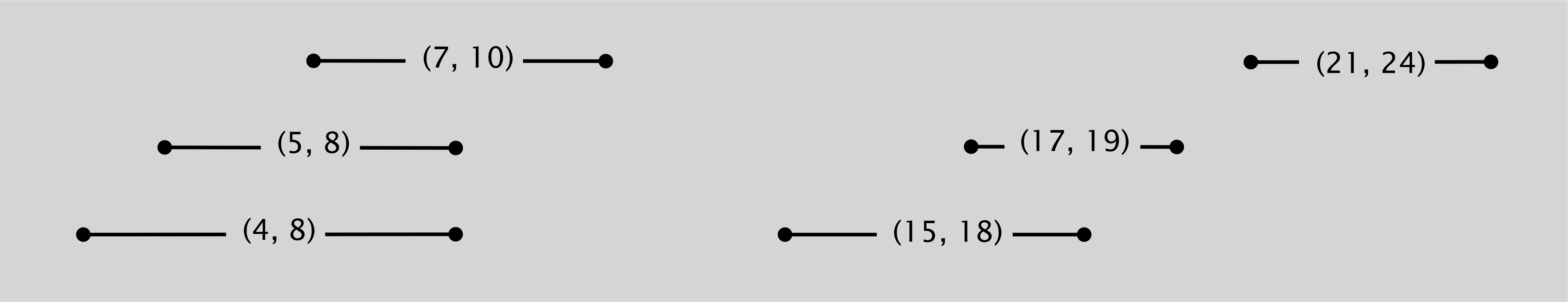
1d interval search

1d interval search. Data structure to hold set of (overlapping) intervals.

- Insert an interval (lo, hi).
- Search for an interval (lo, hi).
- Delete an interval (lo, hi).
- Interval intersection query: given an interval (lo, hi), find all intervals (or one interval) in data structure that intersects (lo, hi).

Q. Which intervals intersect (9, 16) ?

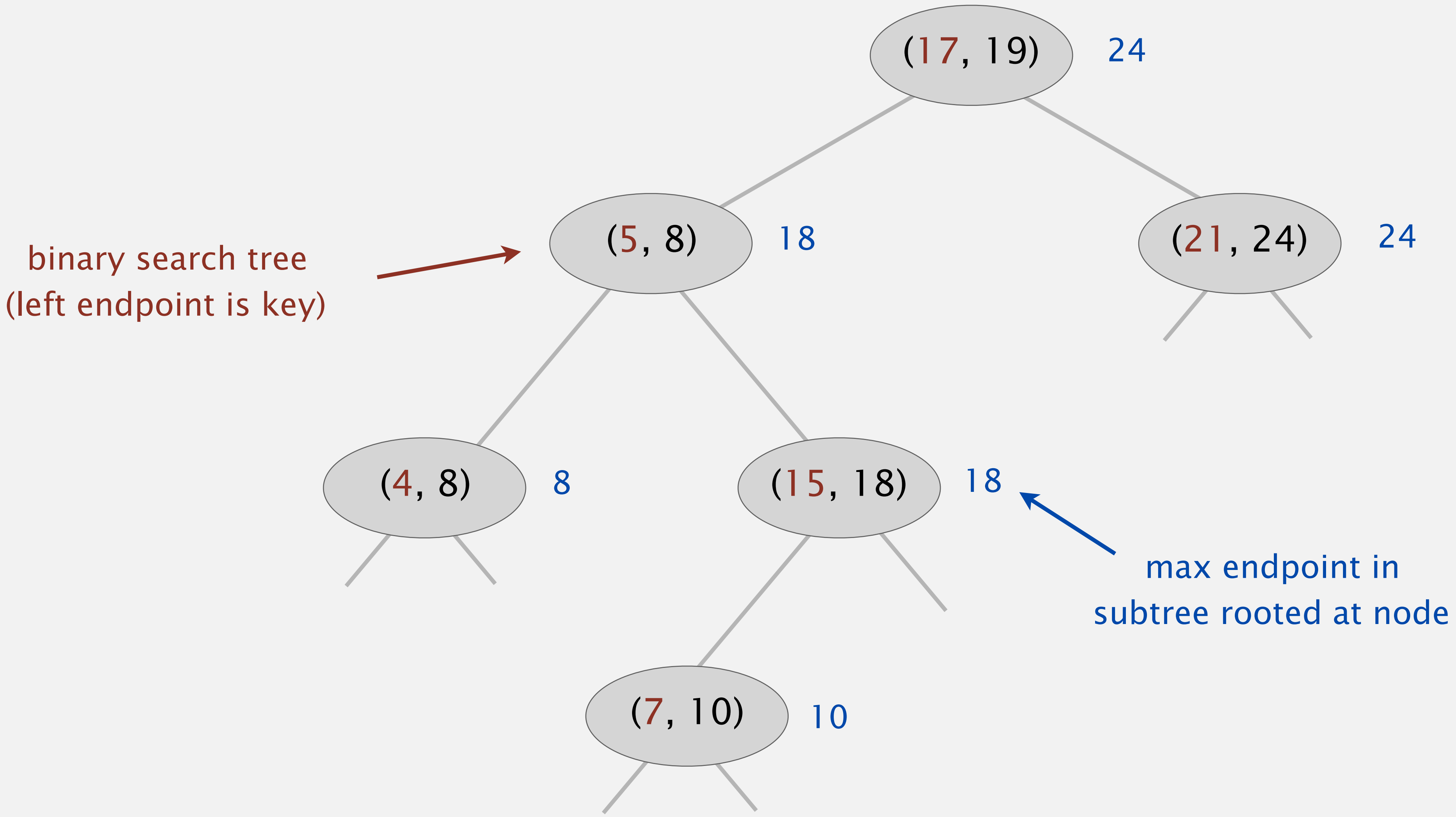
A. (7, 10) and (15, 18).



Interval search trees

Create BST, where each node stores an interval (*lo*, *hi*).

- Use left endpoint as BST key.
- Store max endpoint in subtree rooted at node.



binary search tree
(left endpoint is key)

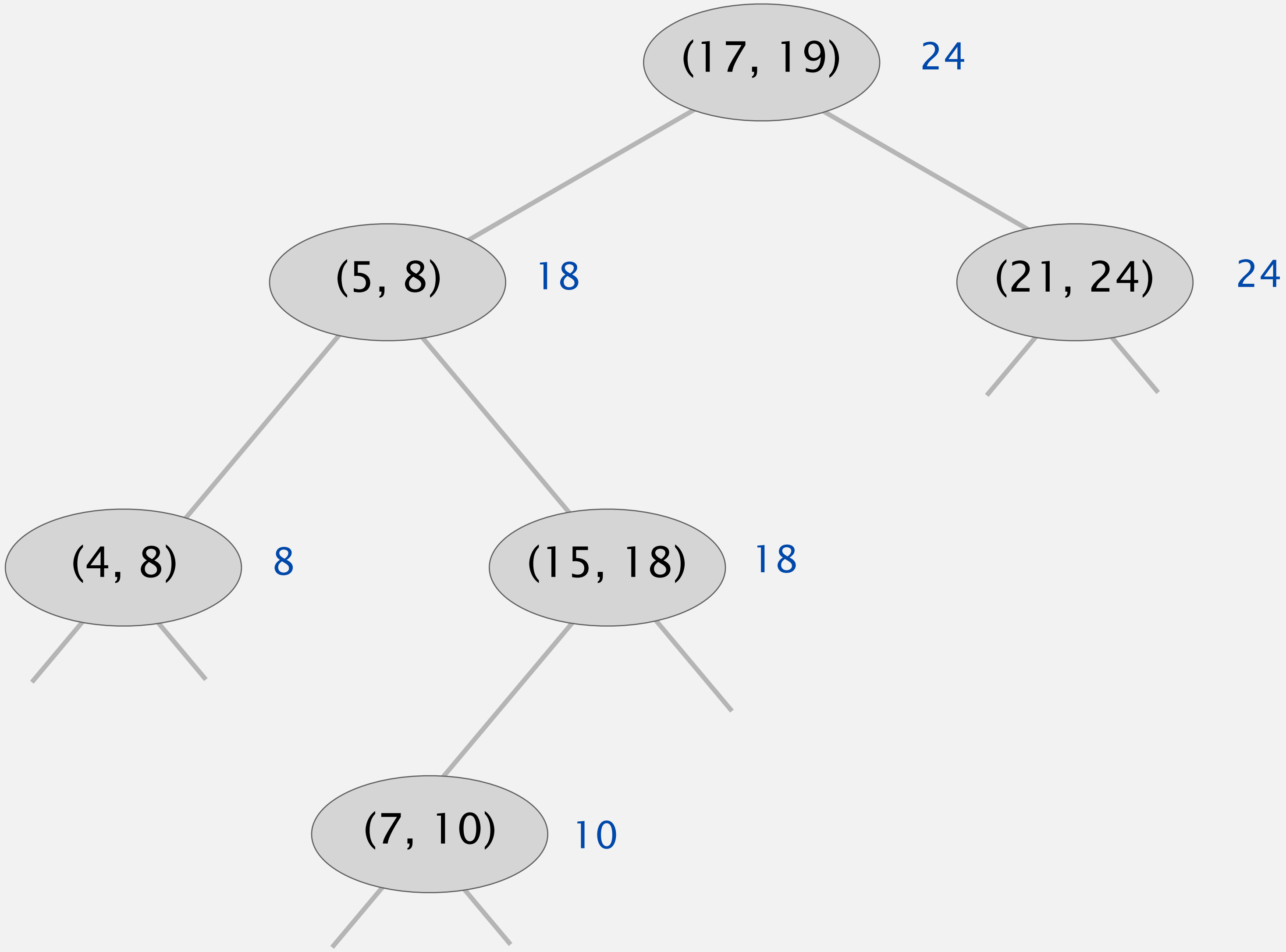
max endpoint in
subtree rooted at node

Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)

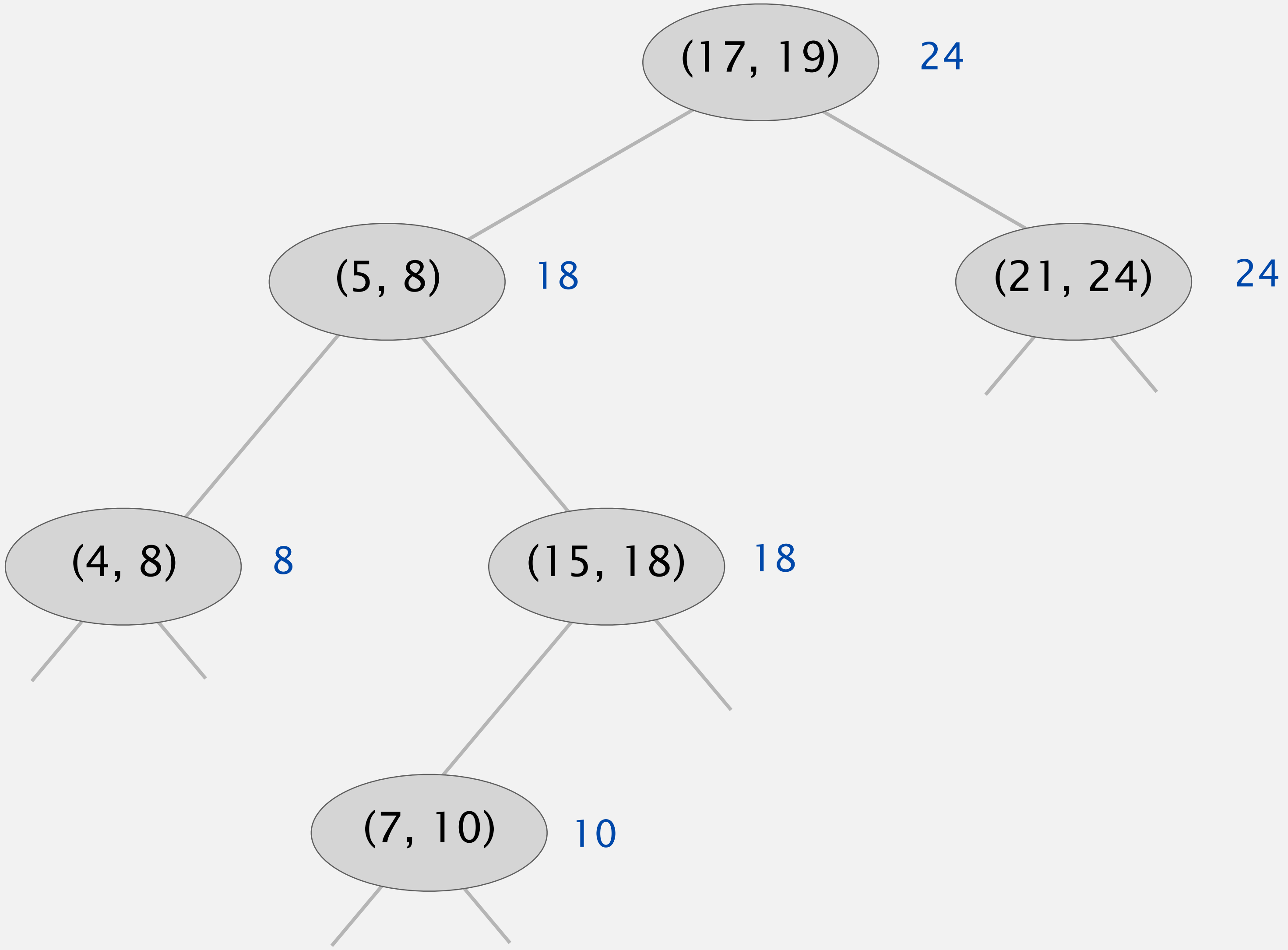


Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)

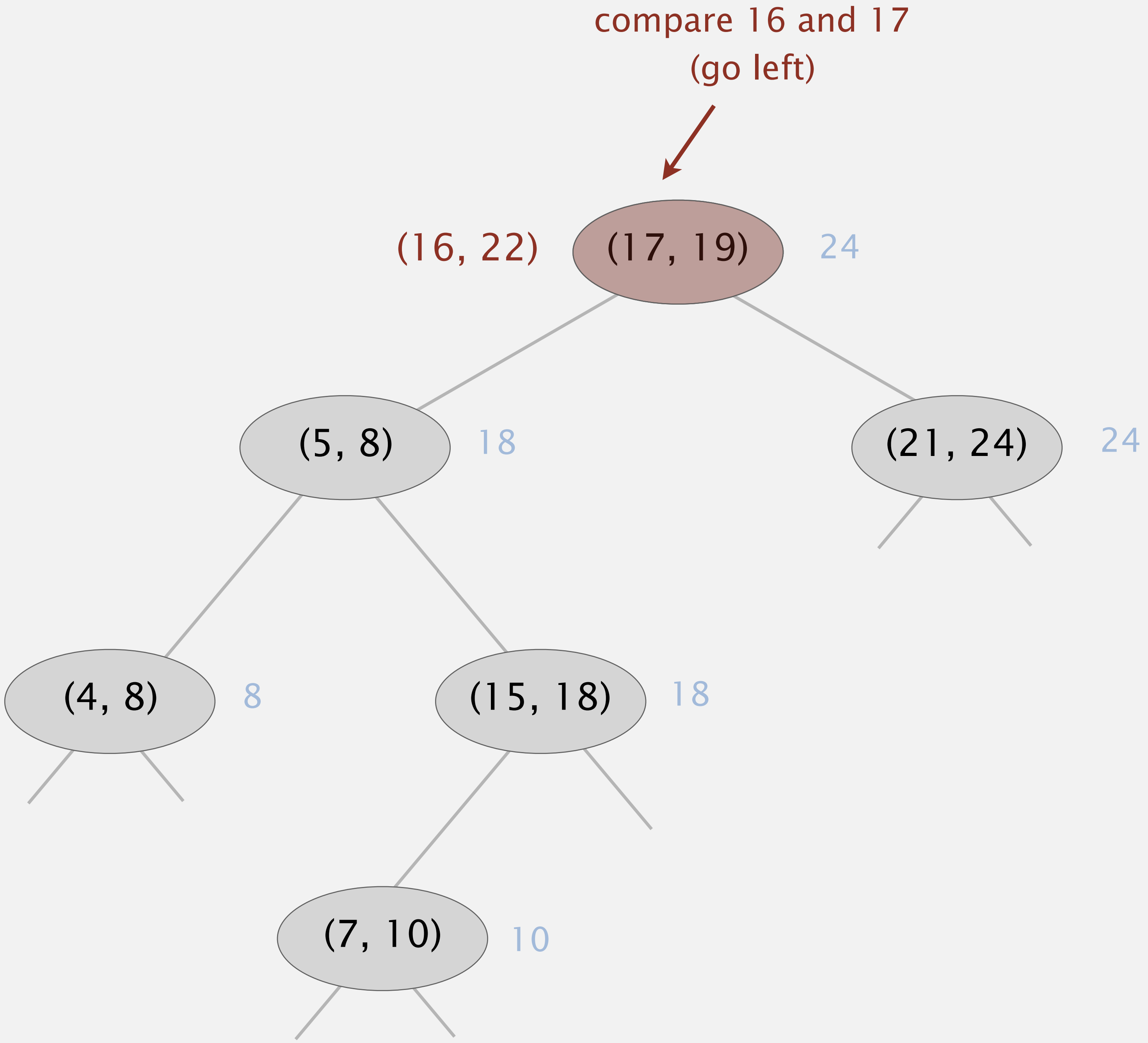


Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)

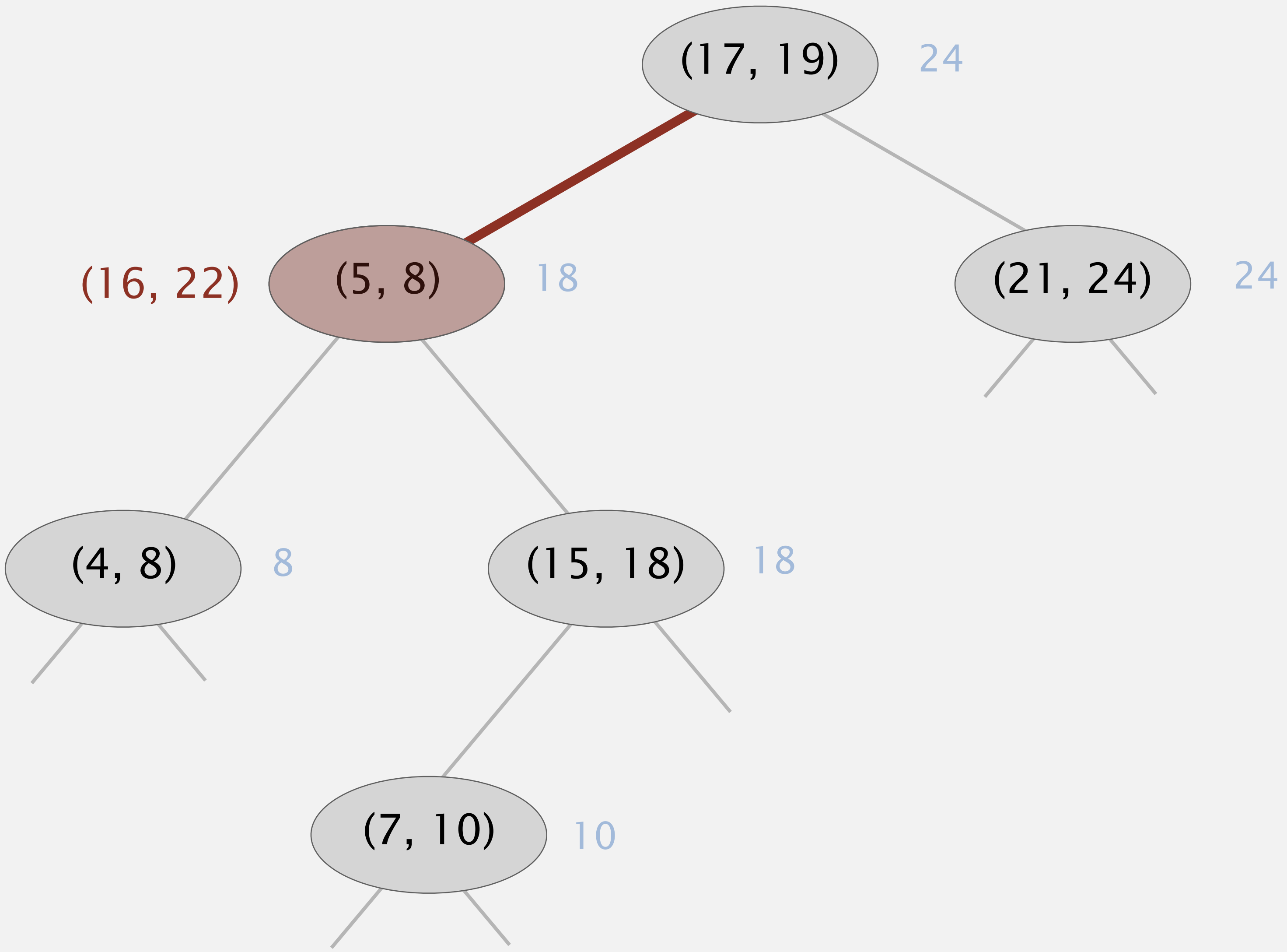


Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)

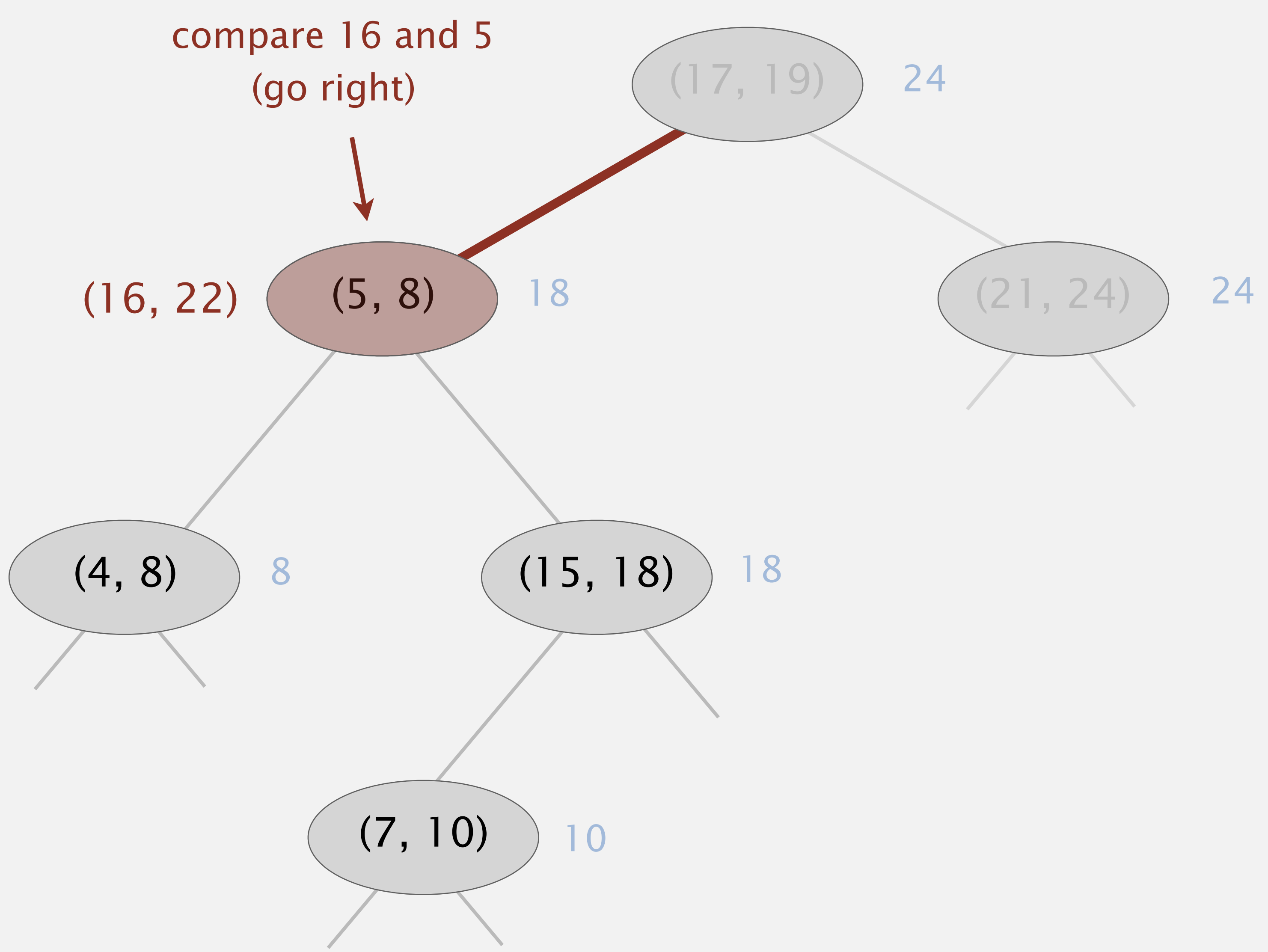


Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)

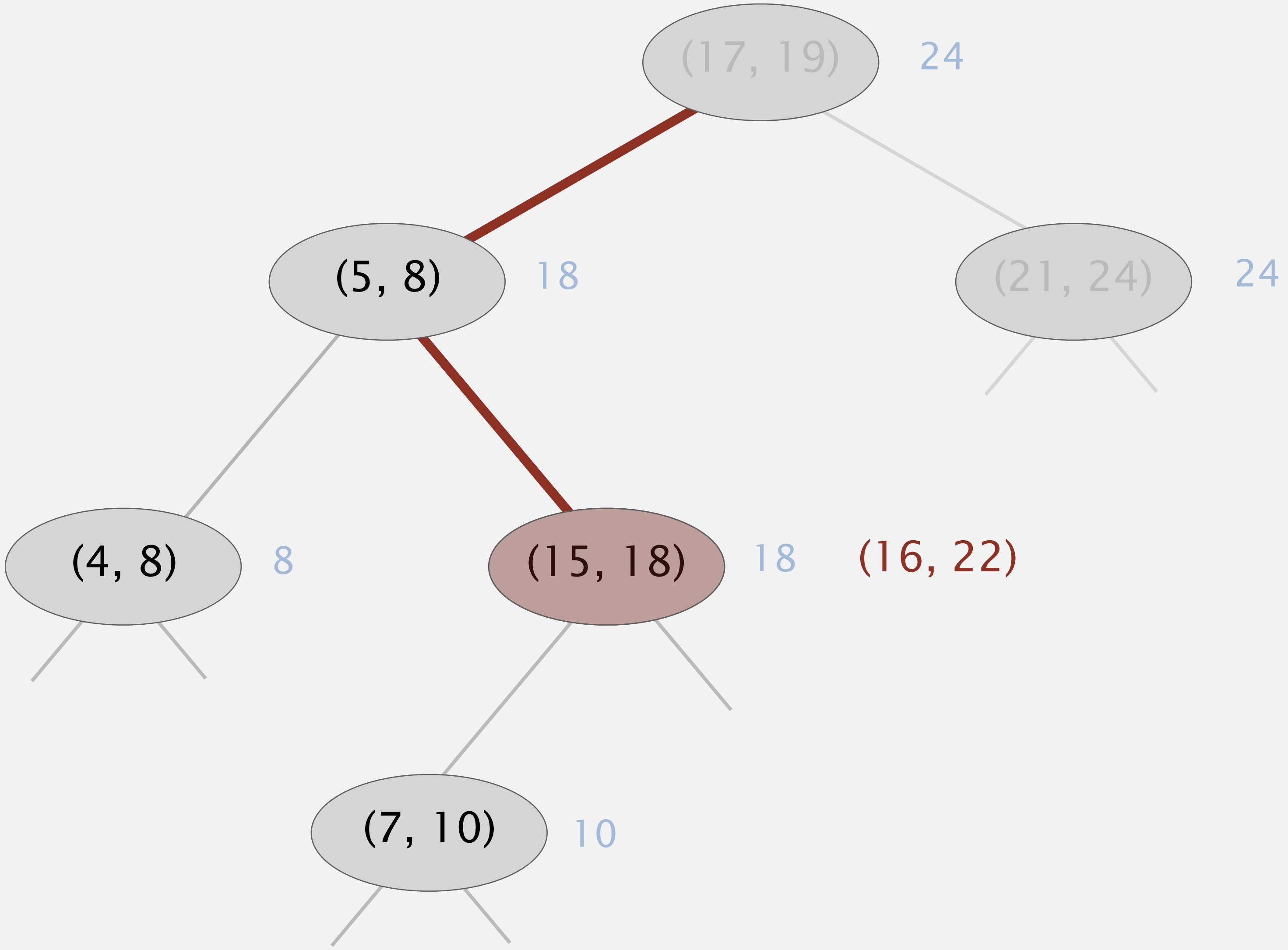


Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)

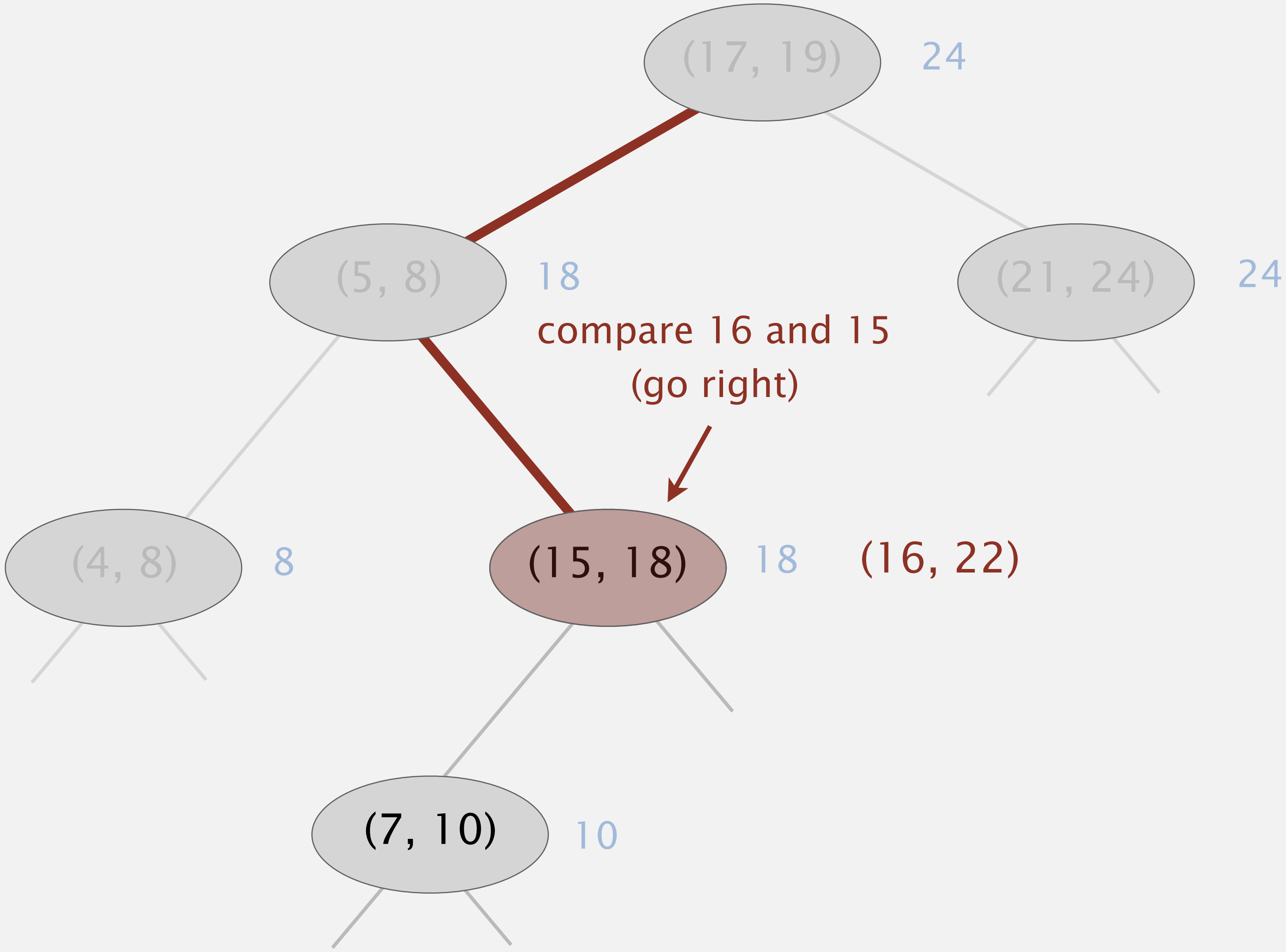


Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)

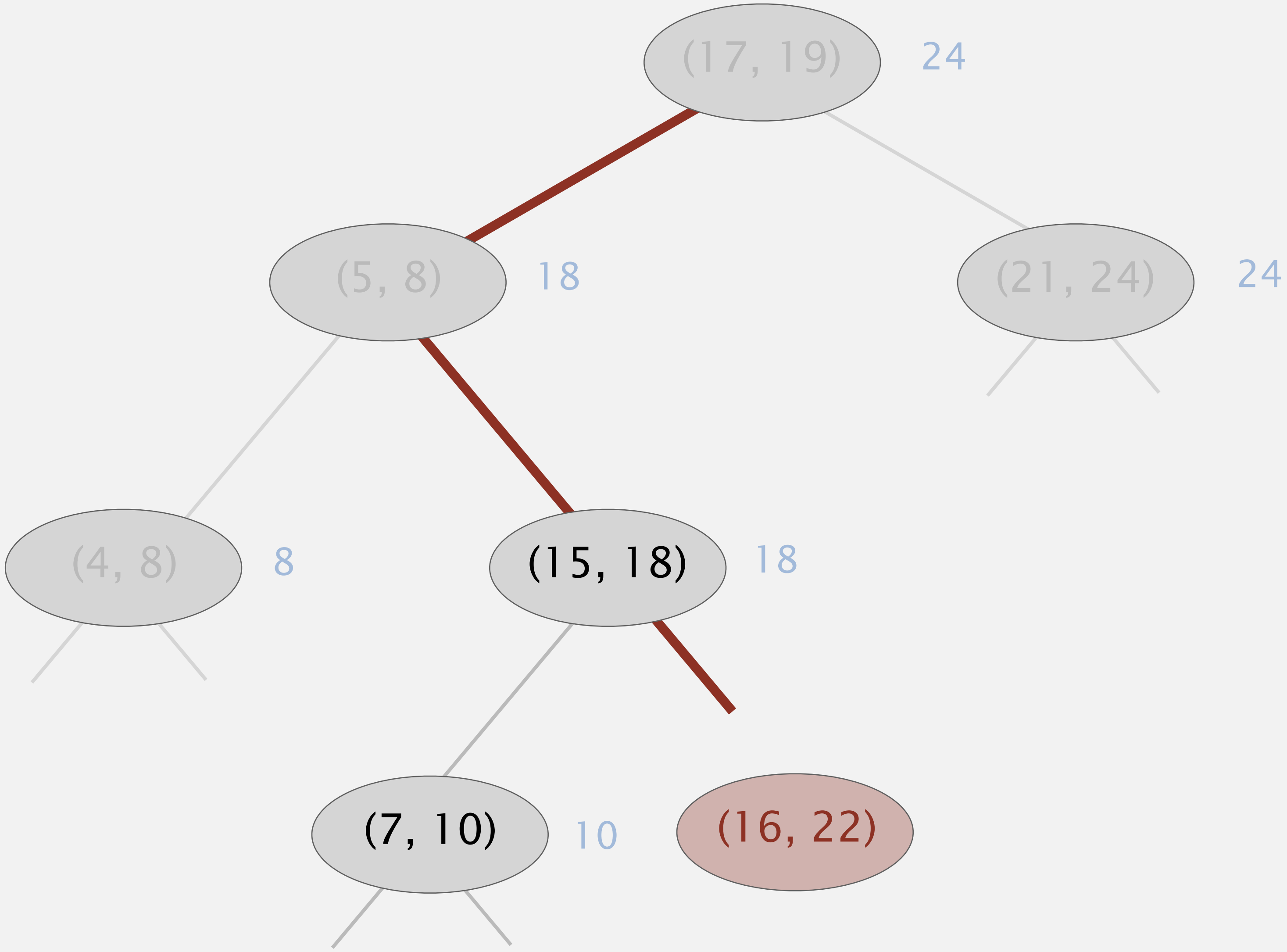


Interval search tree demo: insertion

To insert an interval (lo, hi) :

- Insert into BST, using lo as the key.
- Update max in each node on search path.

insert interval (16, 22)

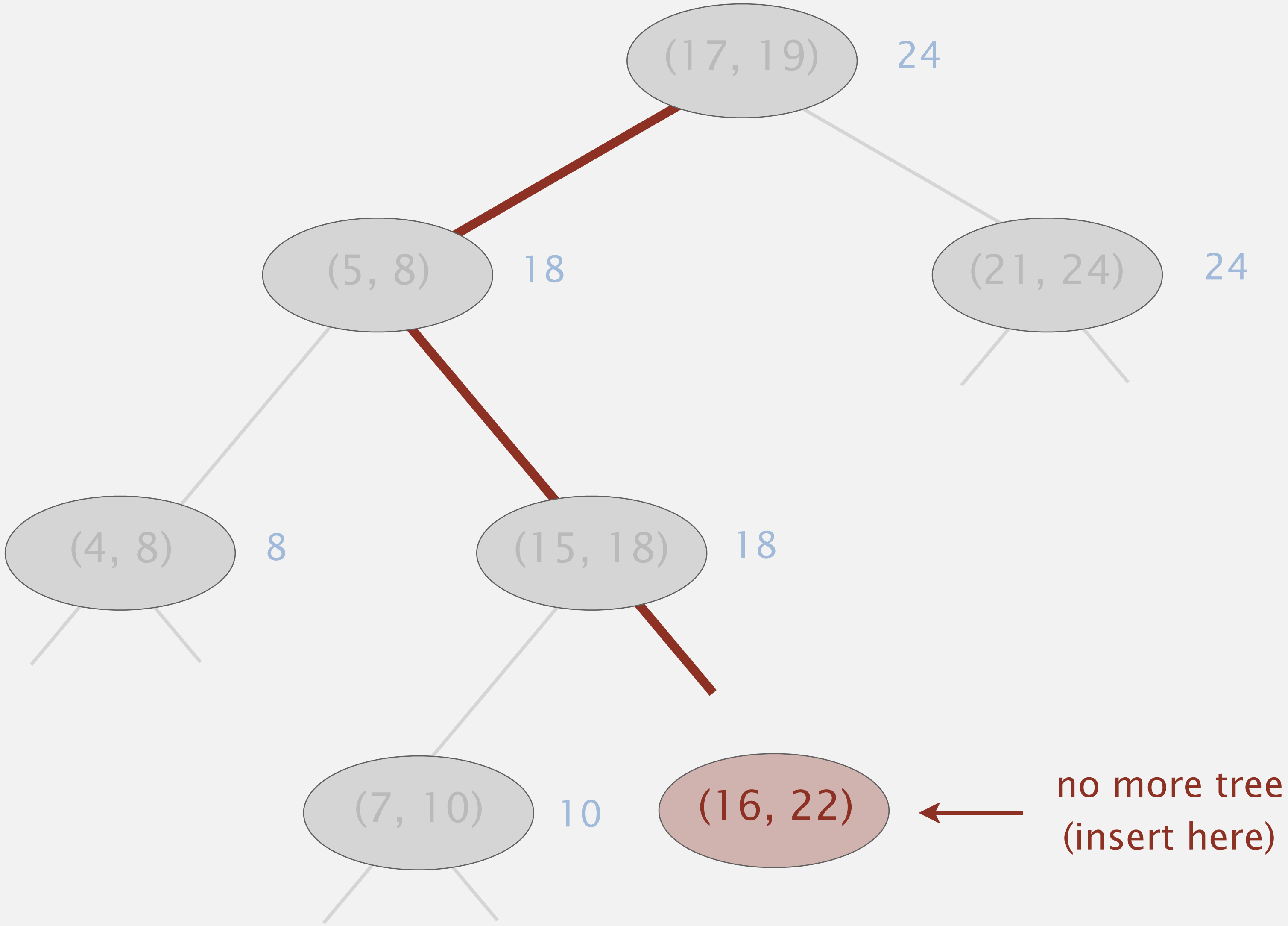


Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)



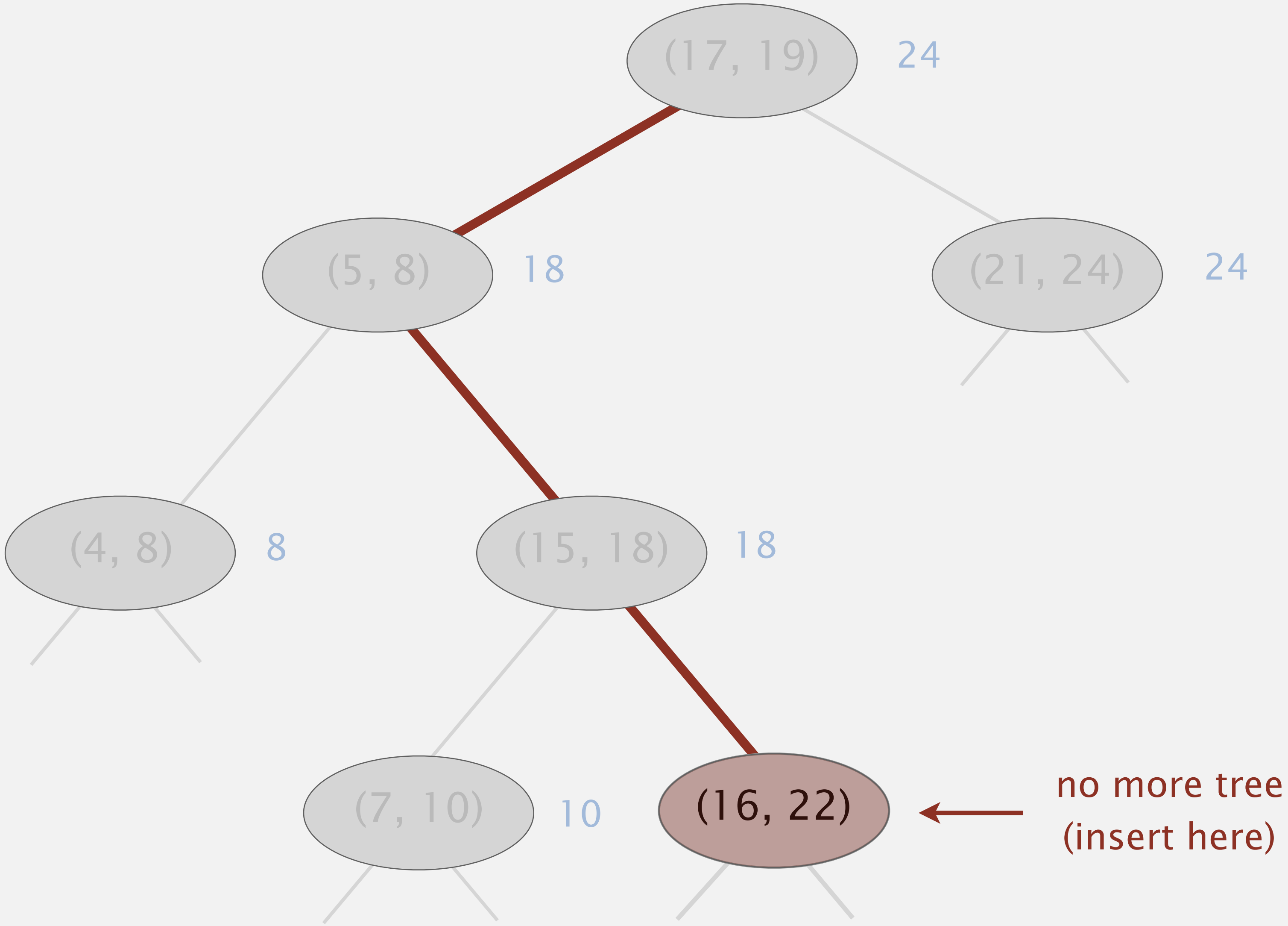
no more tree
(insert here)

Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)

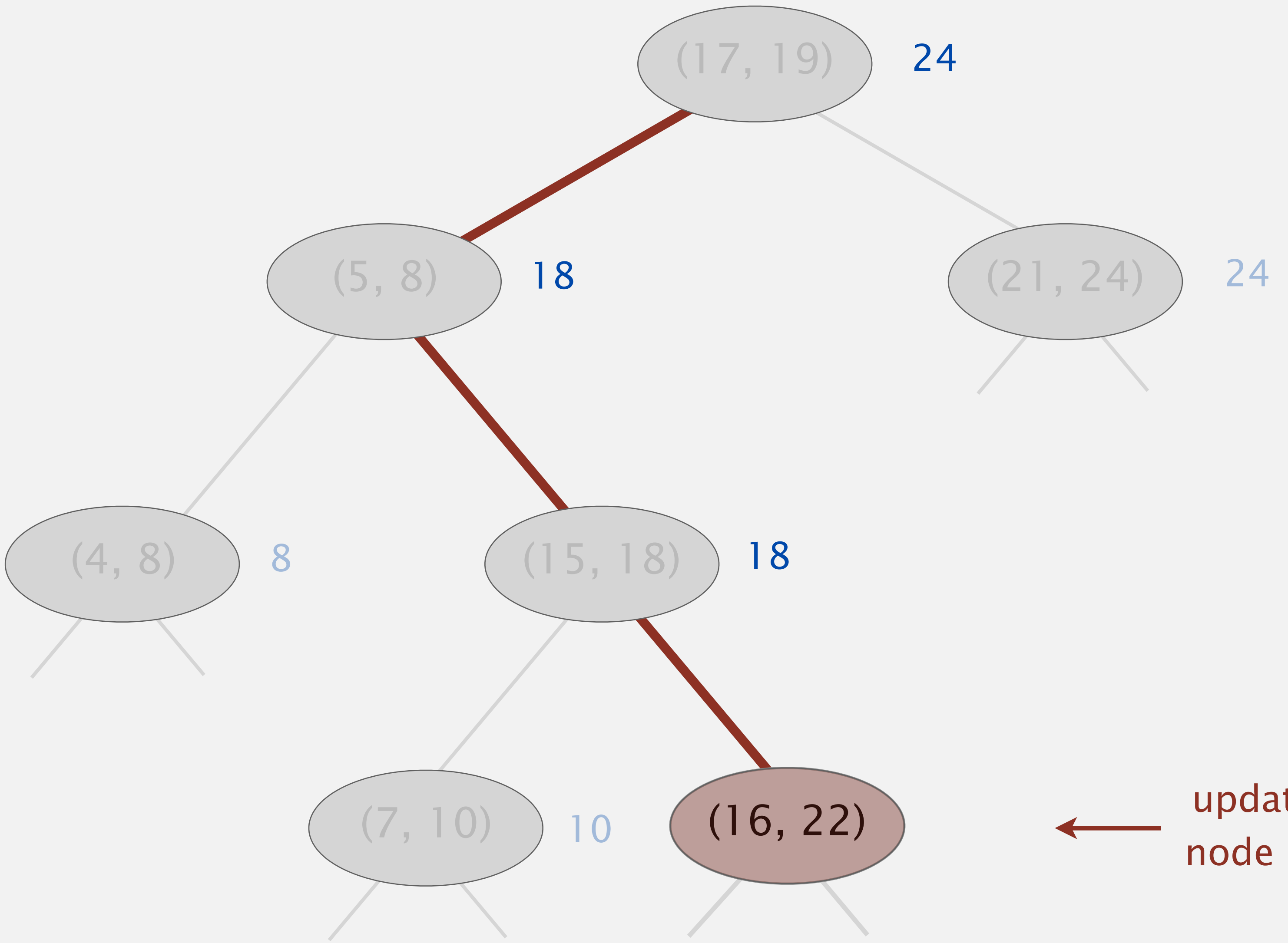


Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)



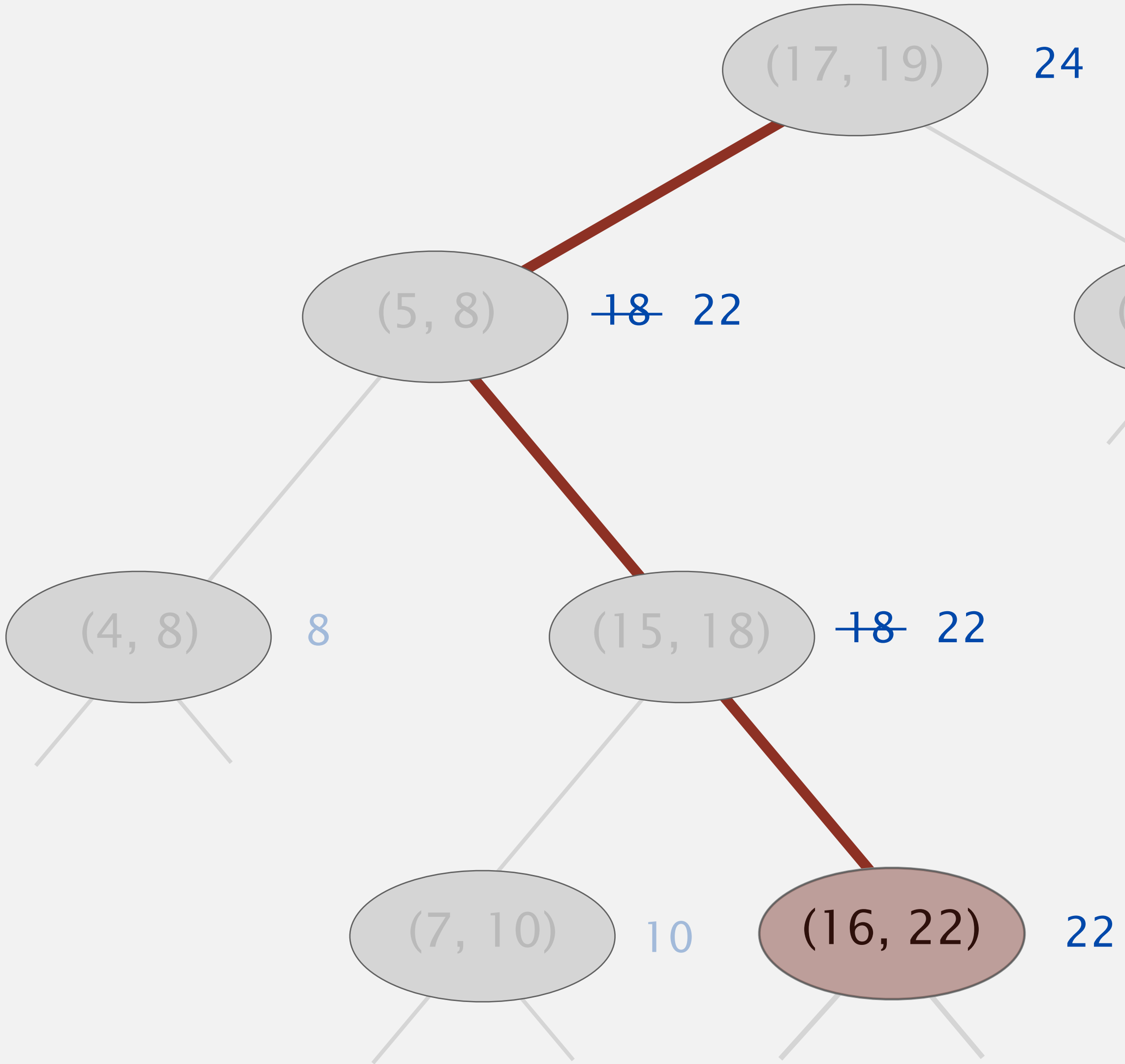
update max in each node on search path

Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)



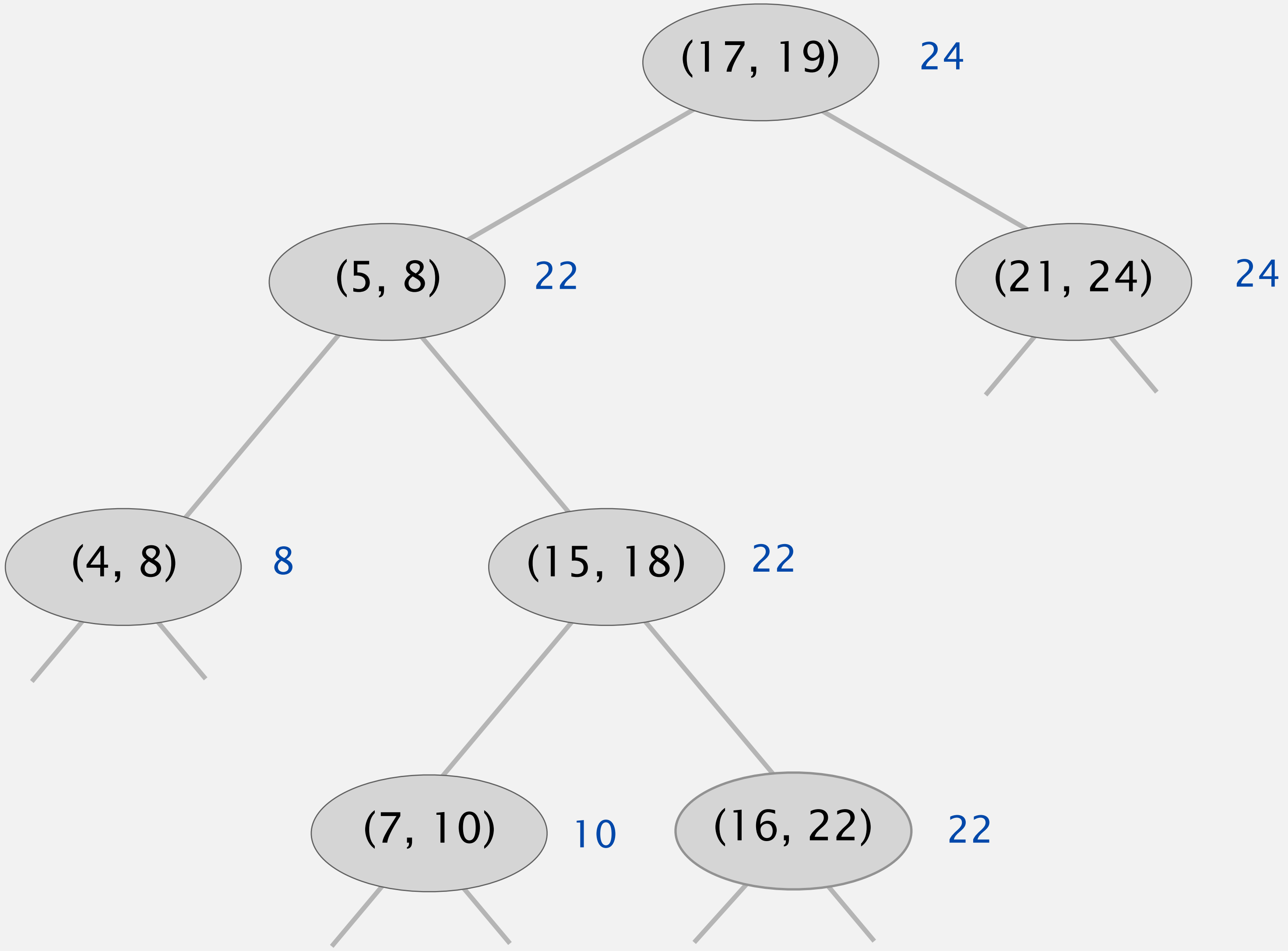
update max in each node on search path

Interval search tree demo: insertion

To insert an interval (*lo*, *hi*) :

- Insert into BST, using *lo* as the key.
- Update max in each node on search path.

insert interval (16, 22)



INTERVAL SEARCH

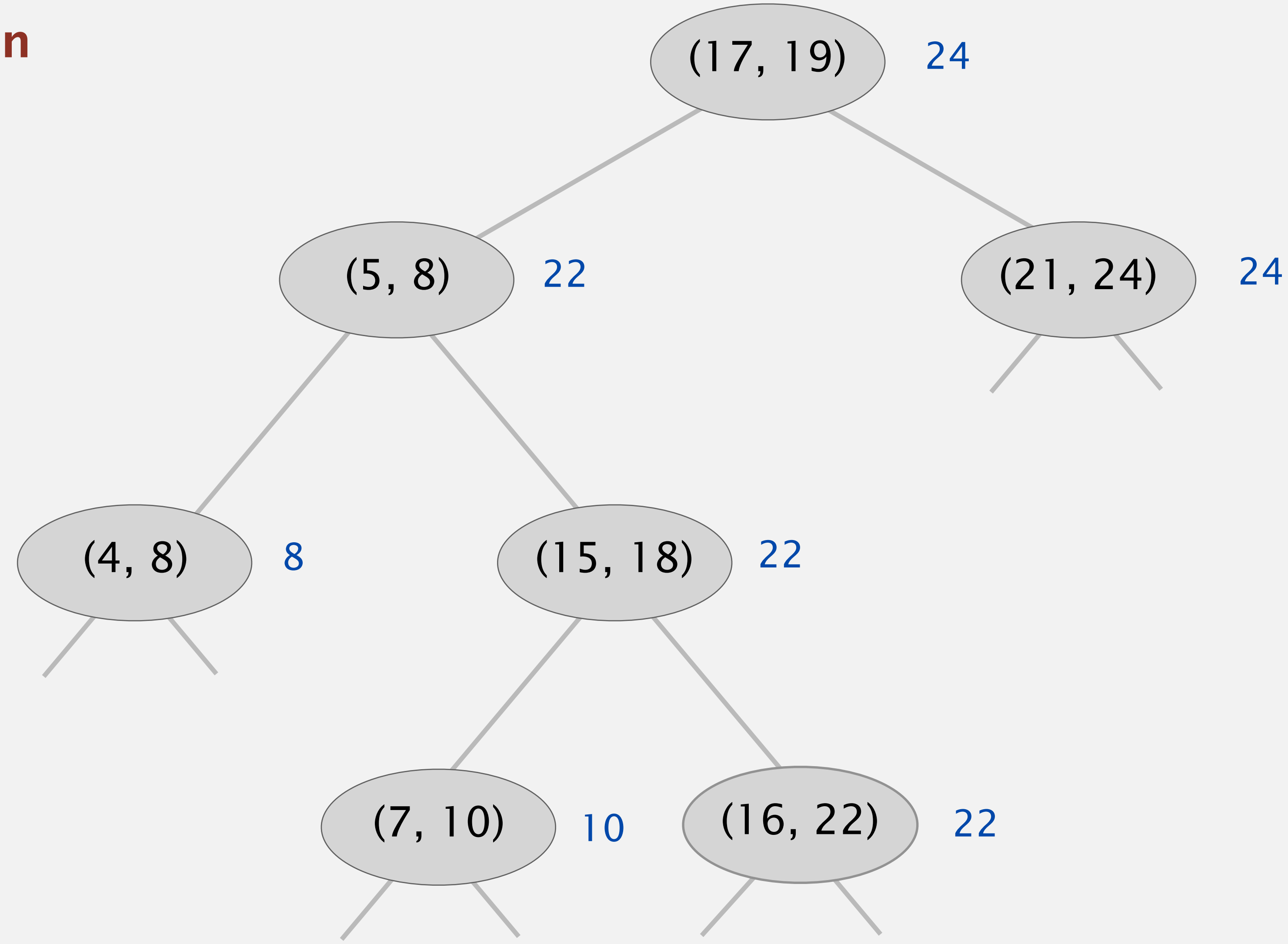
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (23, 25)



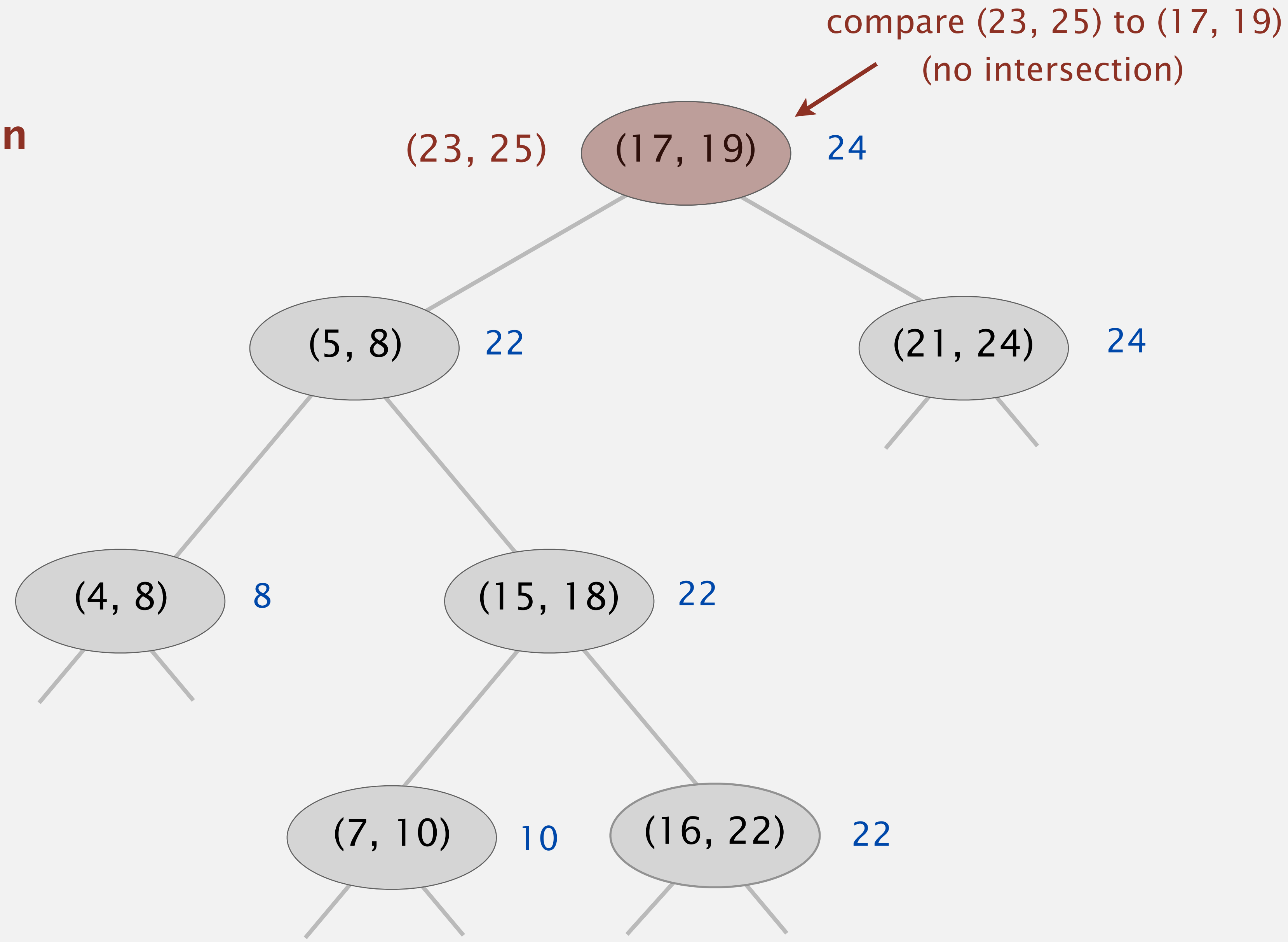
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (23, 25)



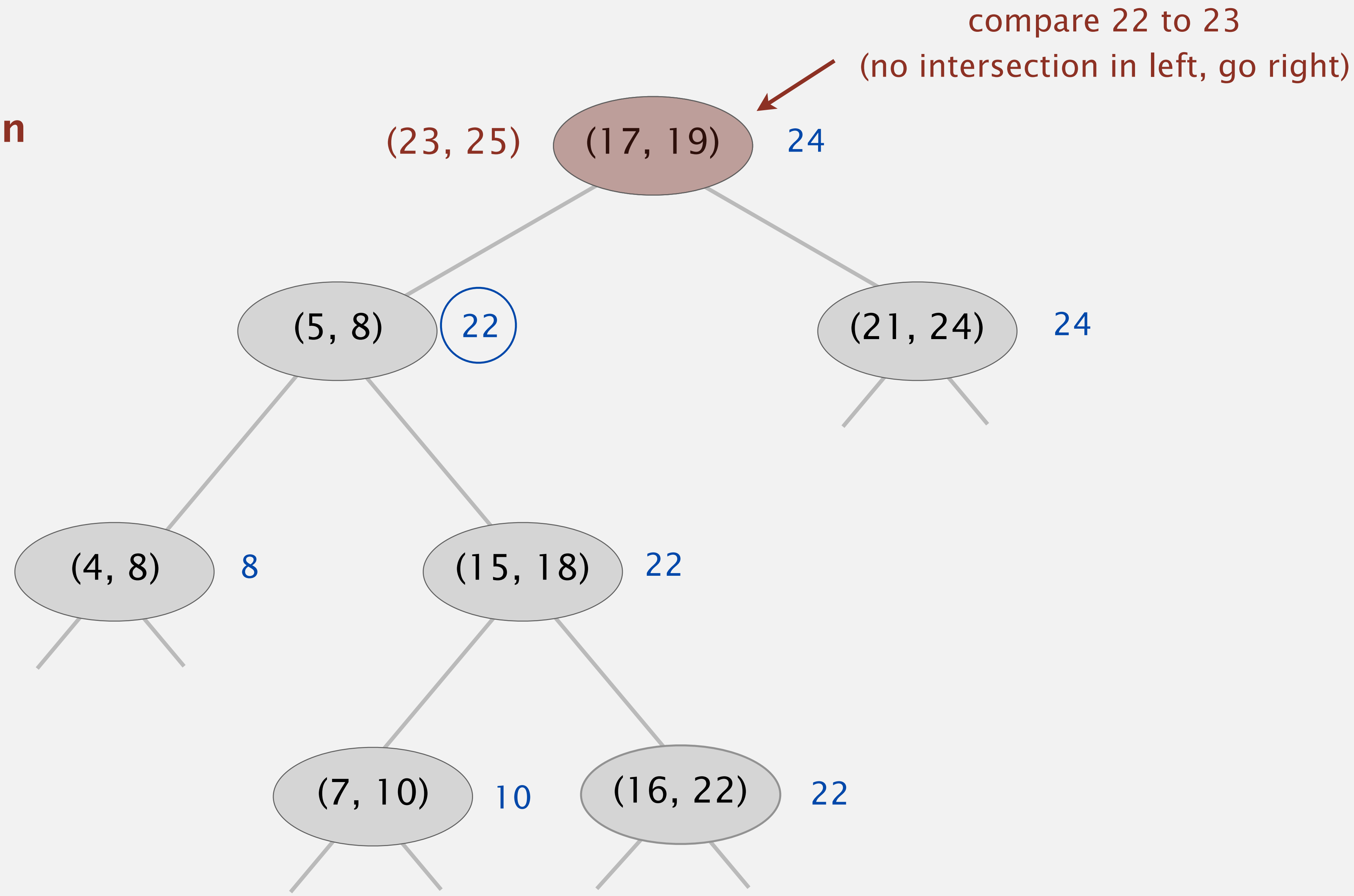
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (23, 25)



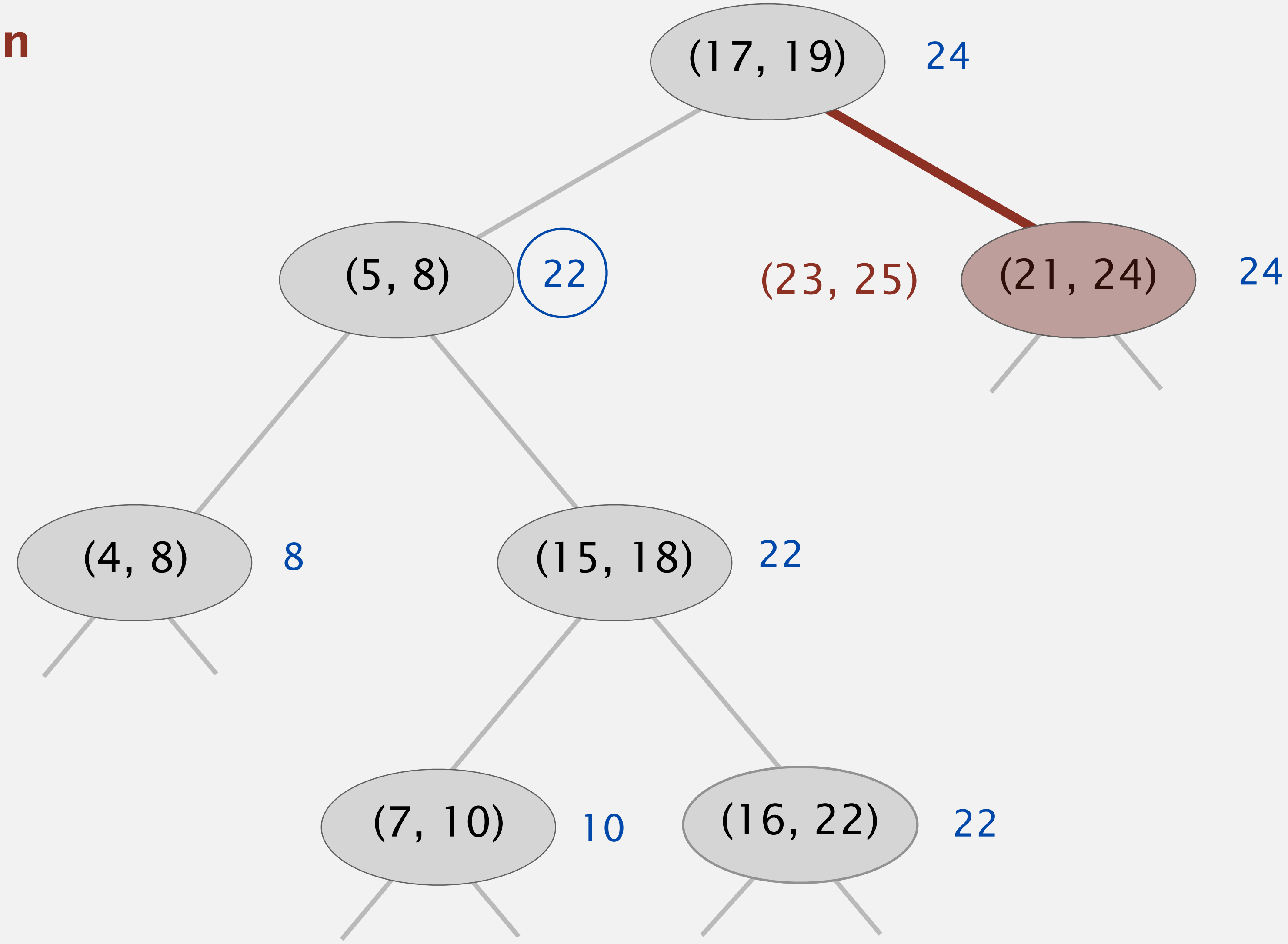
Interval search tree demo: intersection

To search for any one interval that intersects query interval (lo, hi) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo, go right.
- Else go left.

interval intersection

search for (23, 25)



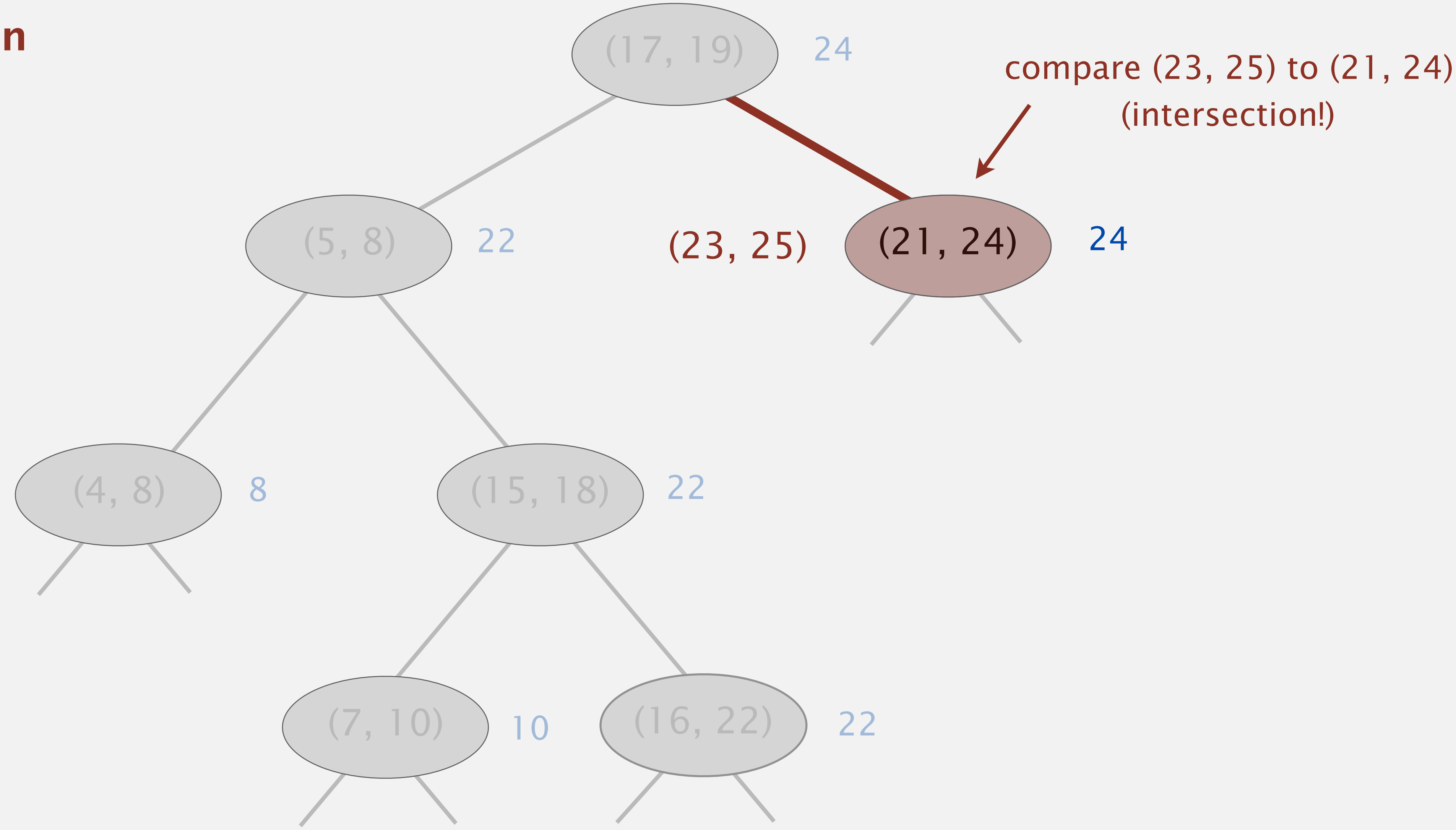
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (23, 25)



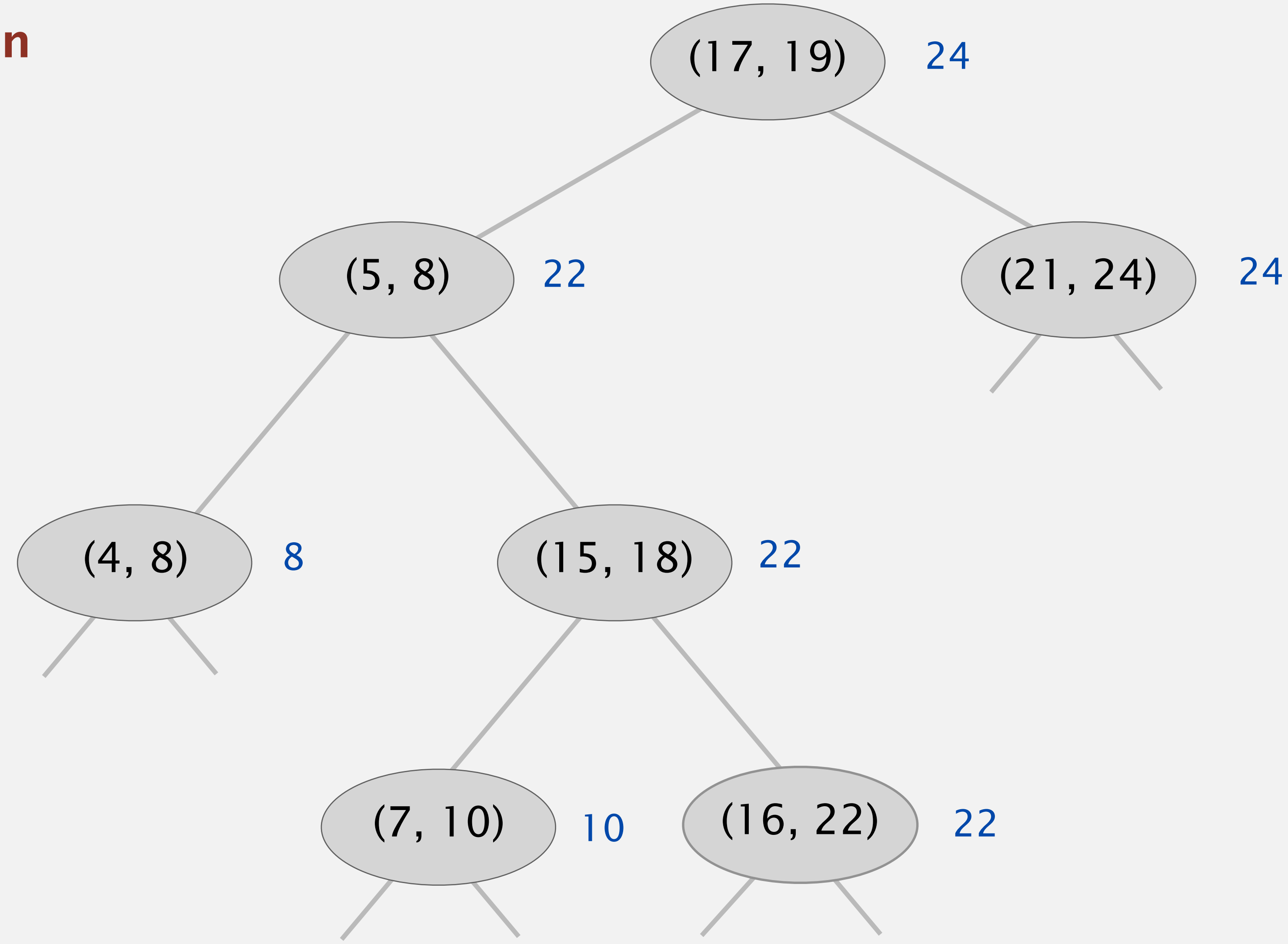
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



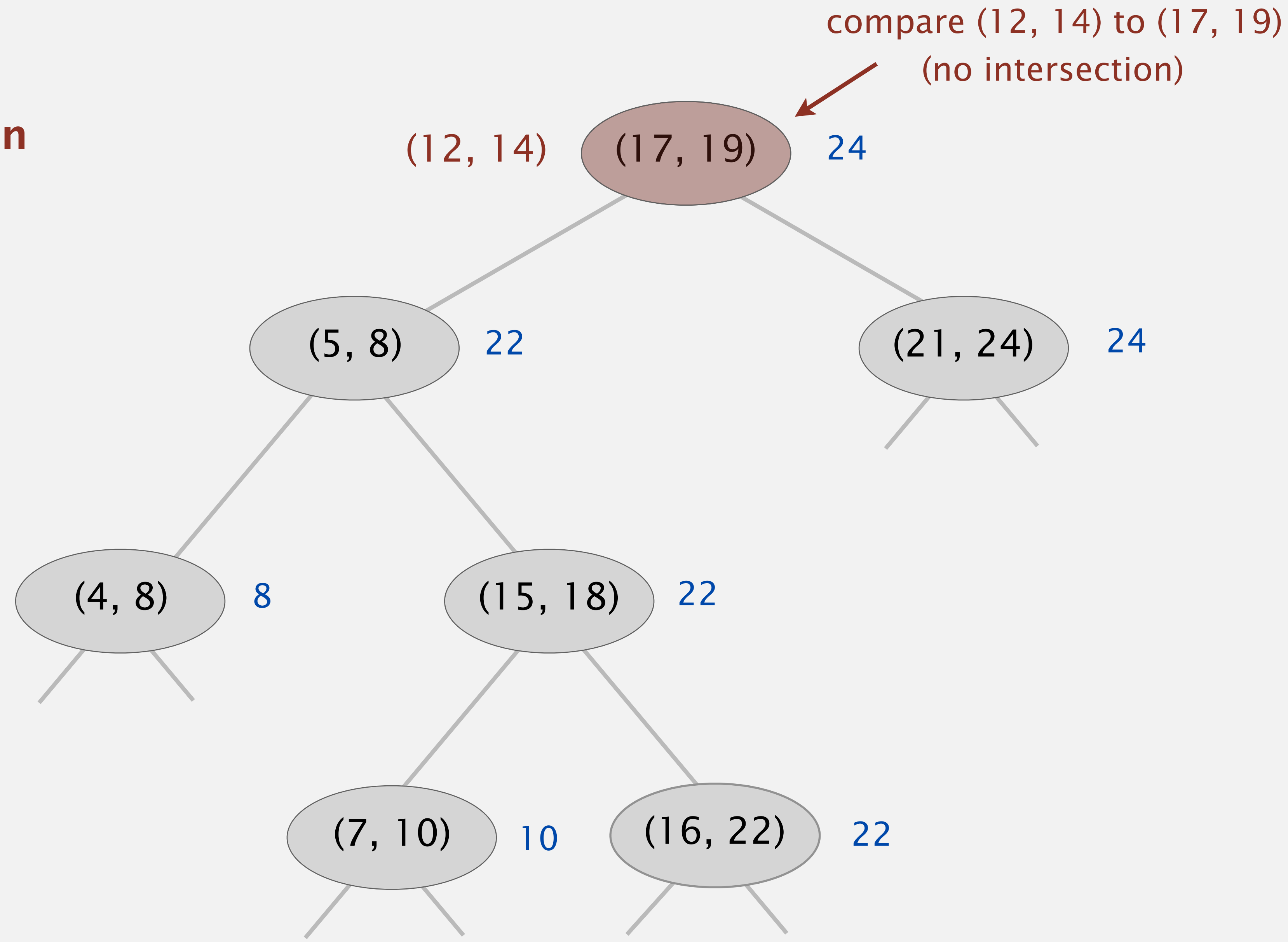
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



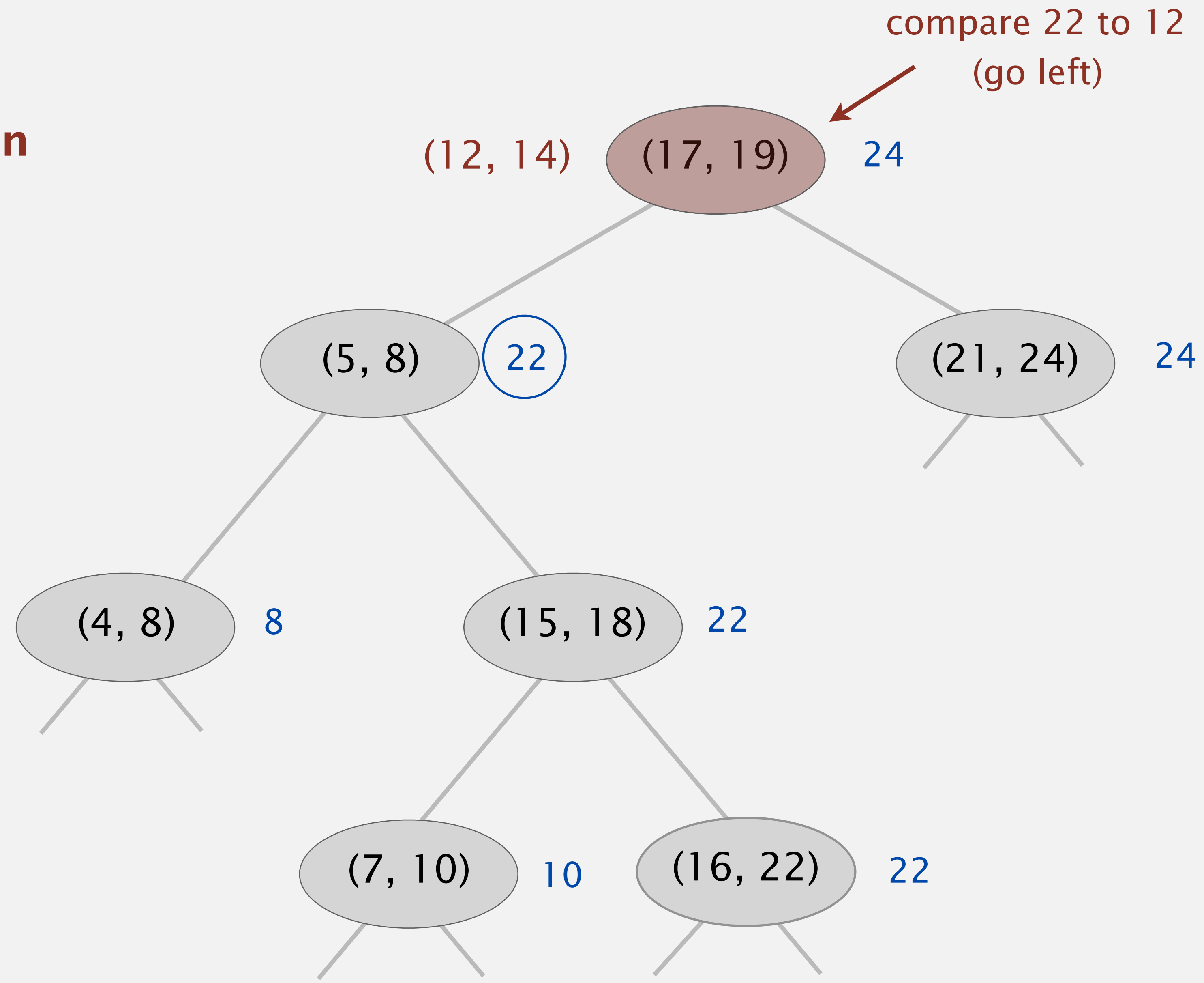
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



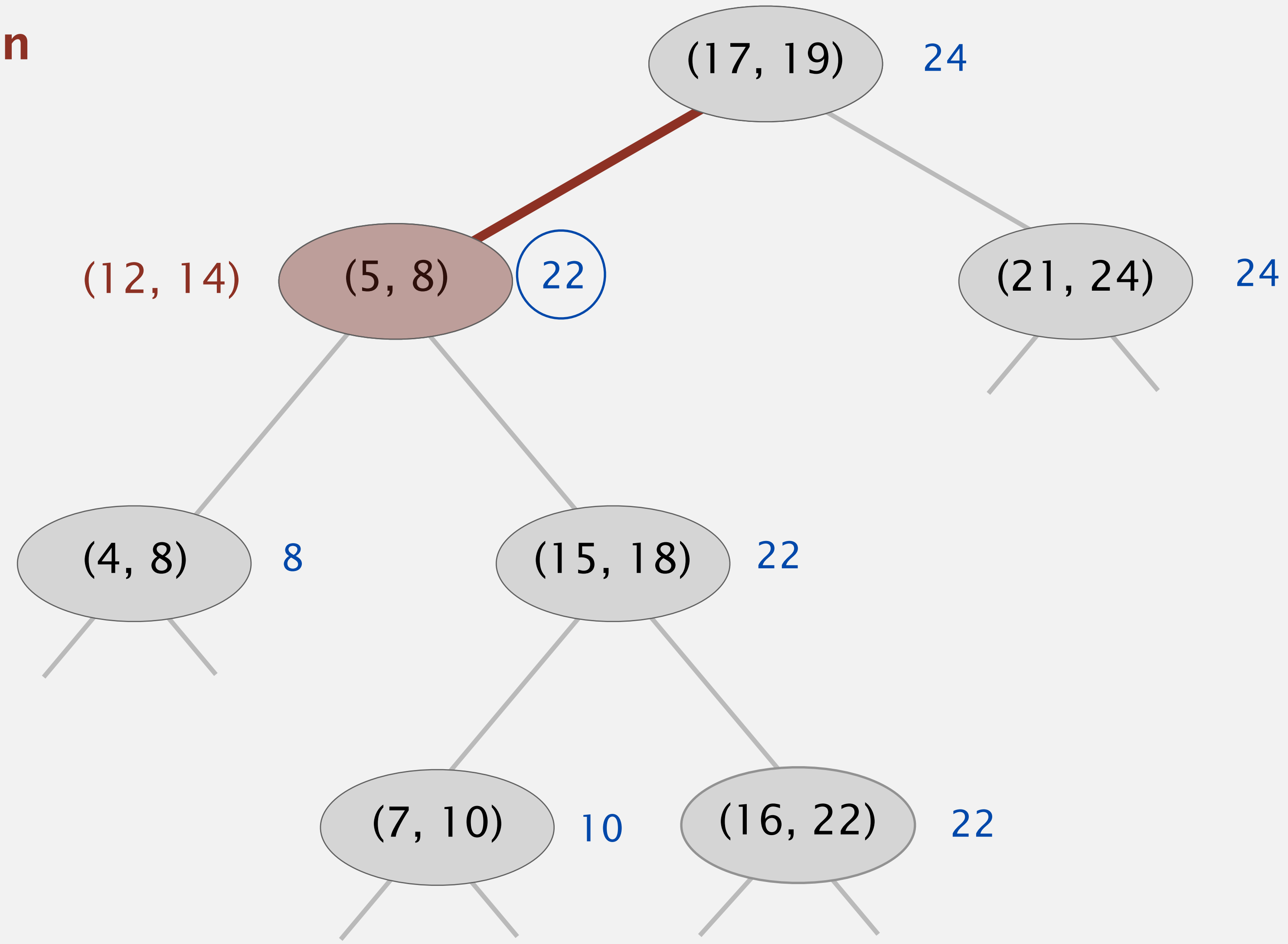
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



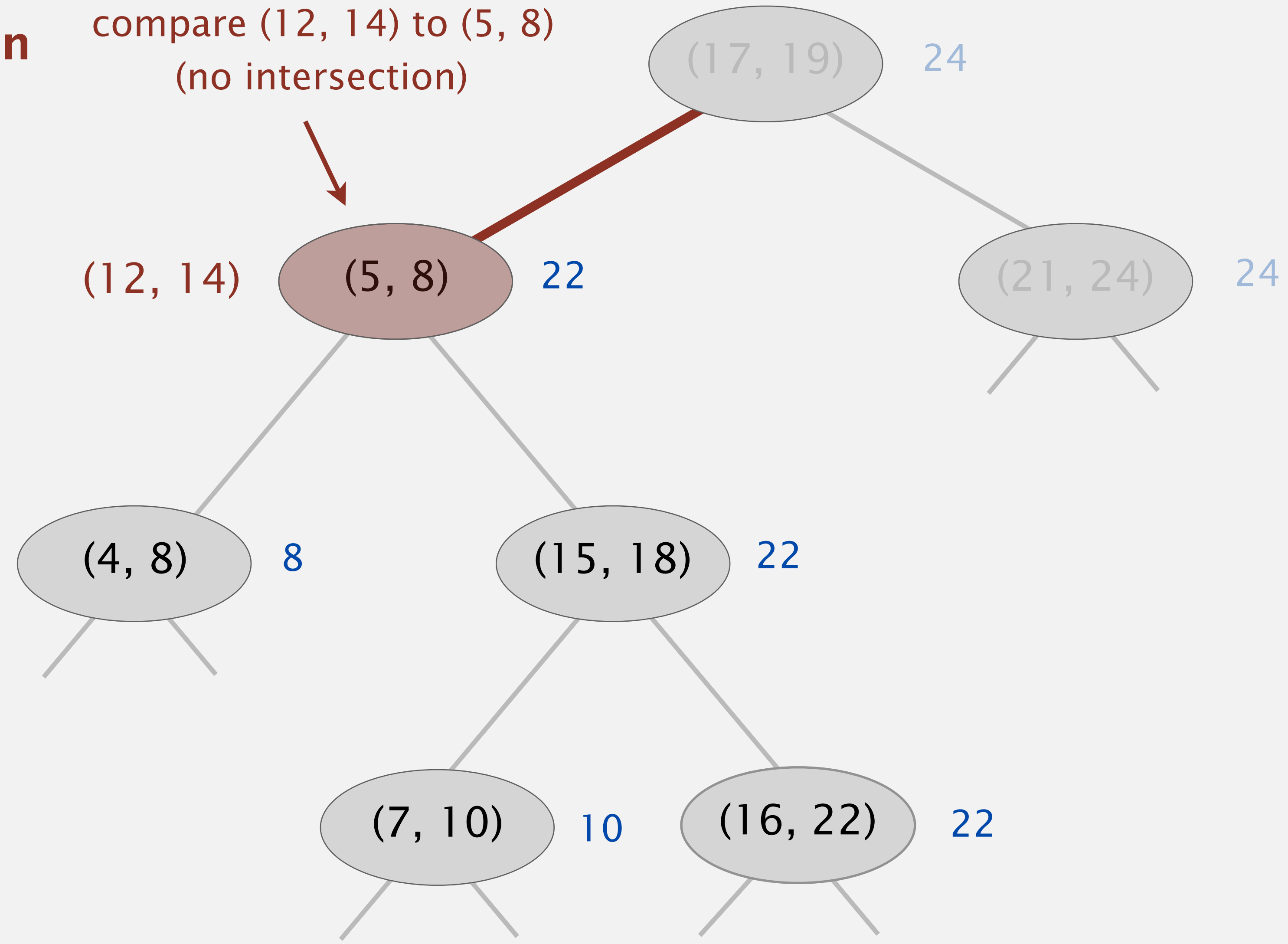
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



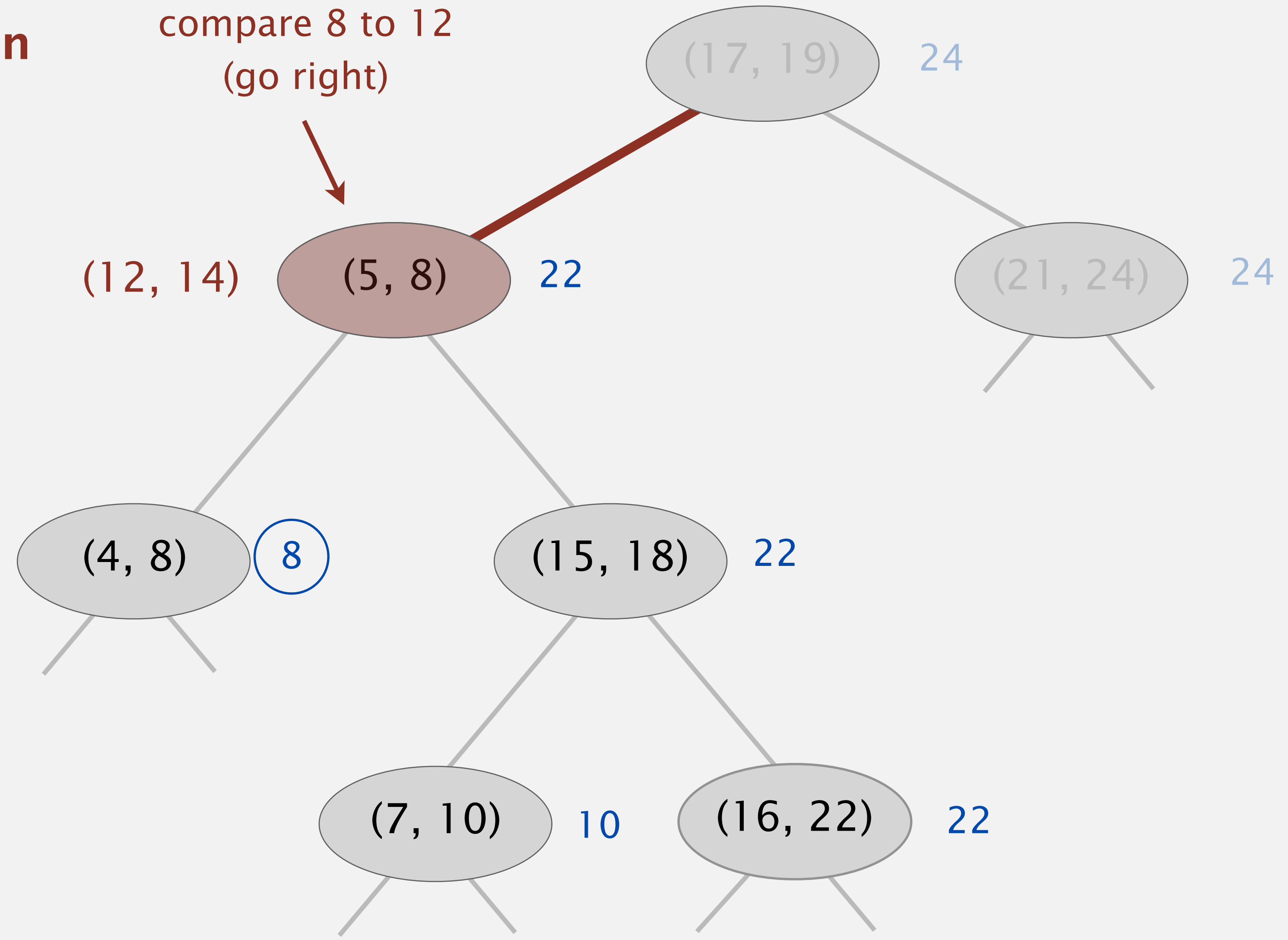
Interval search tree demo: intersection

To search for any one interval that intersects query interval $(lo, \ hi)$:

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo, go right.
- Else go left.

interval intersection

search for (12, 14)



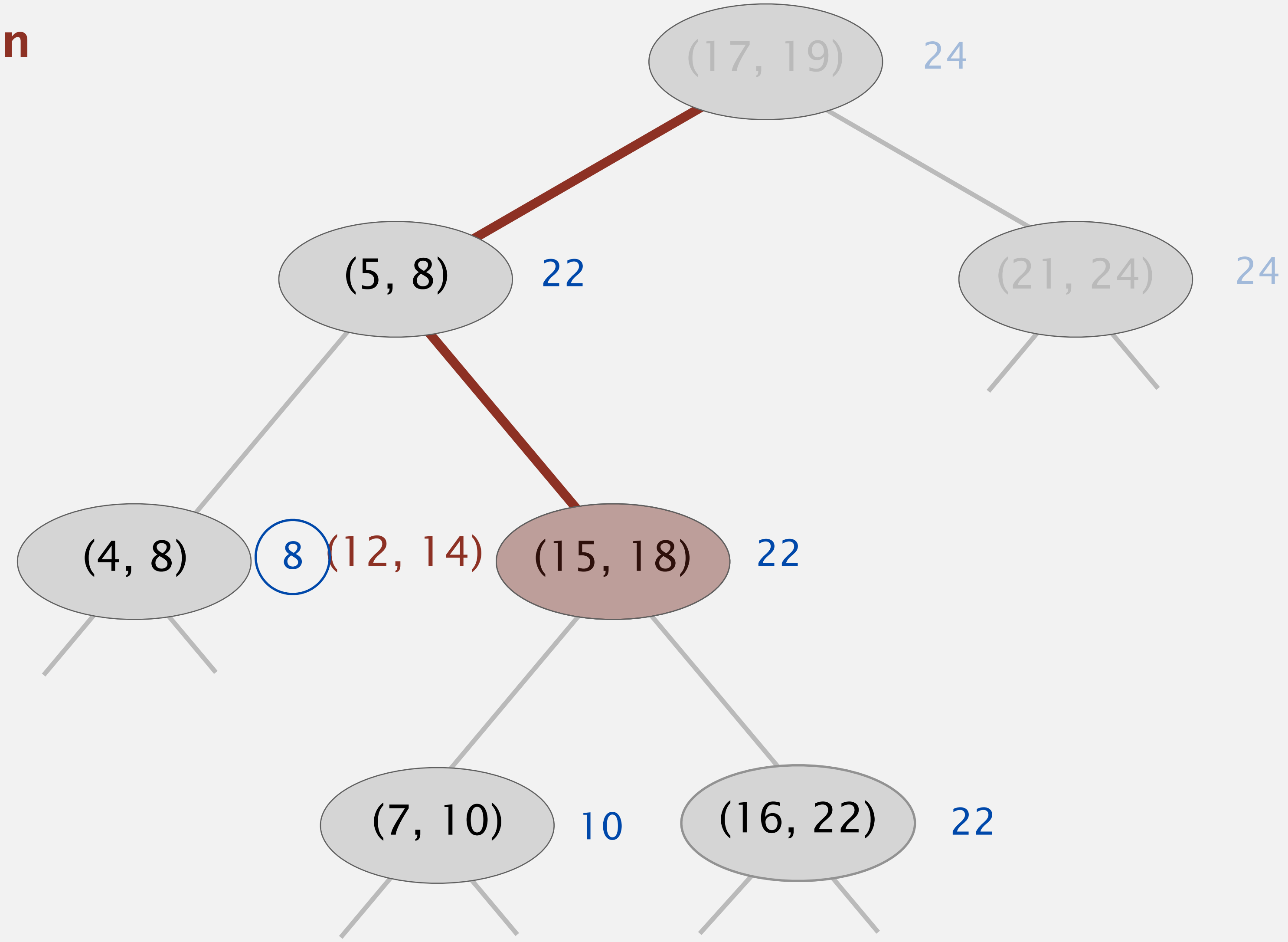
Interval search tree demo: intersection

To search for any one interval that intersects query interval (lo, hi) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo, go right.
- Else go left.

interval intersection

search for (12, 14)



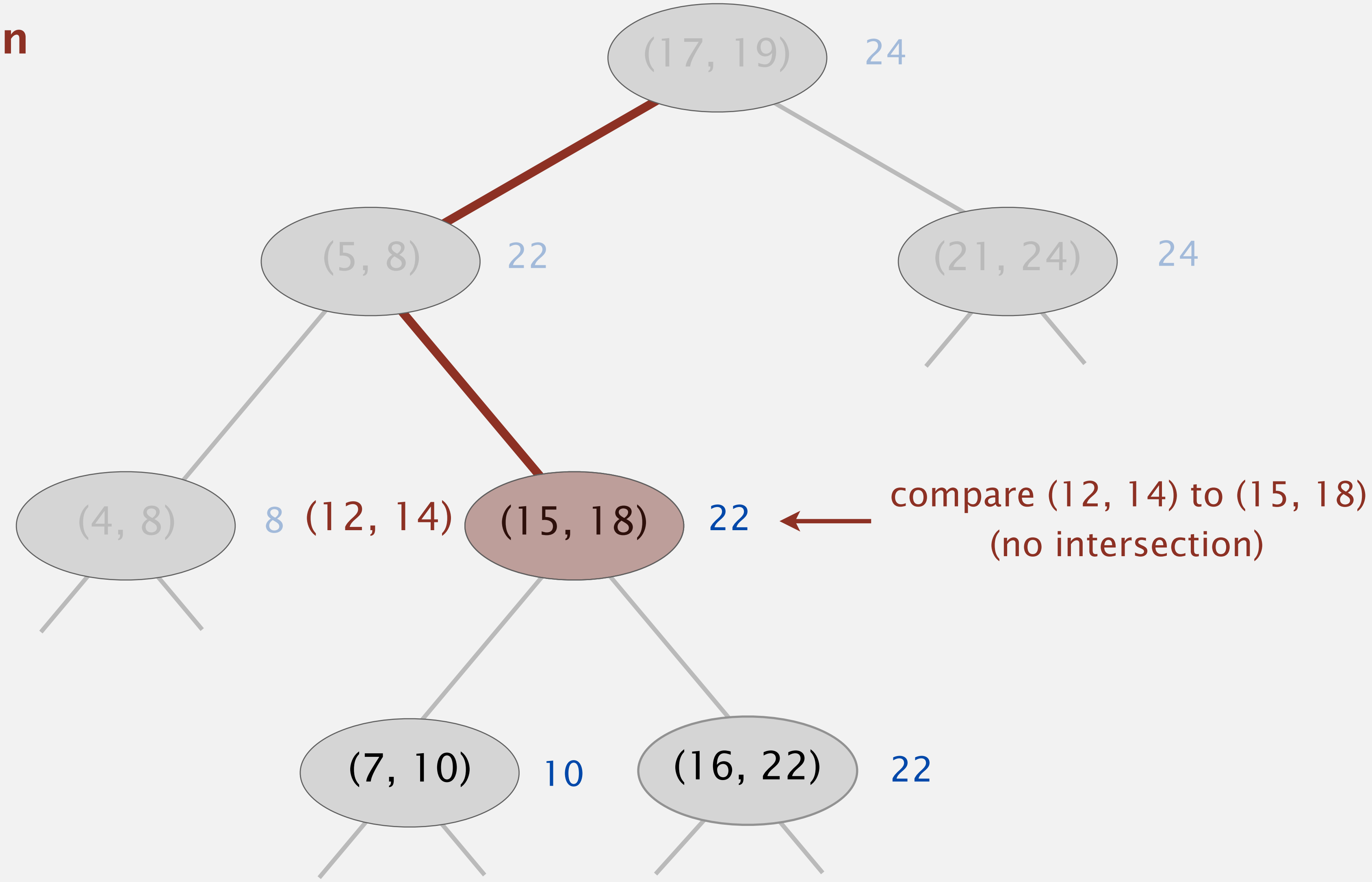
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



compare (12, 14) to (15, 18)
(no intersection)

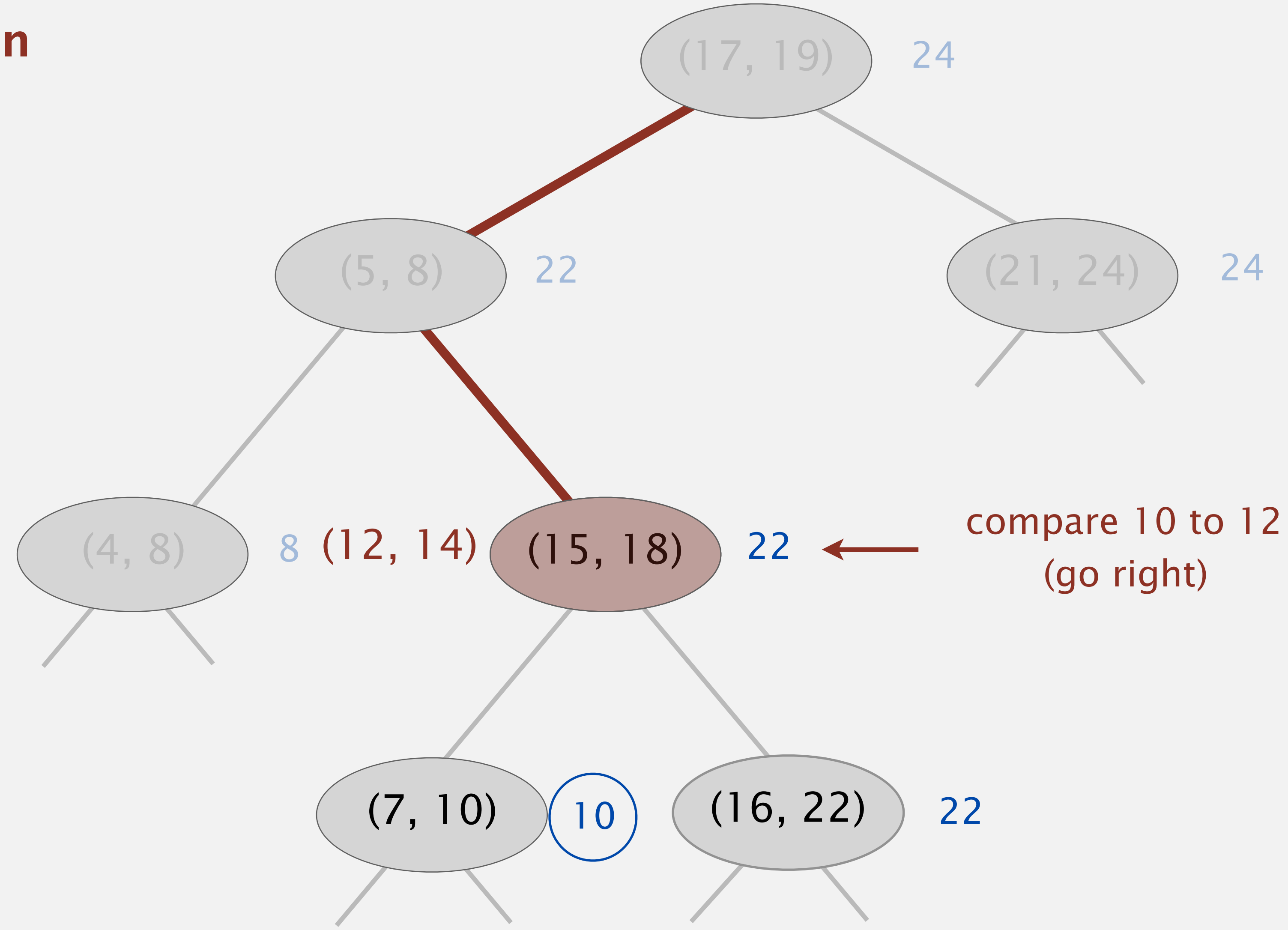
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



compare 10 to 12
(go right)

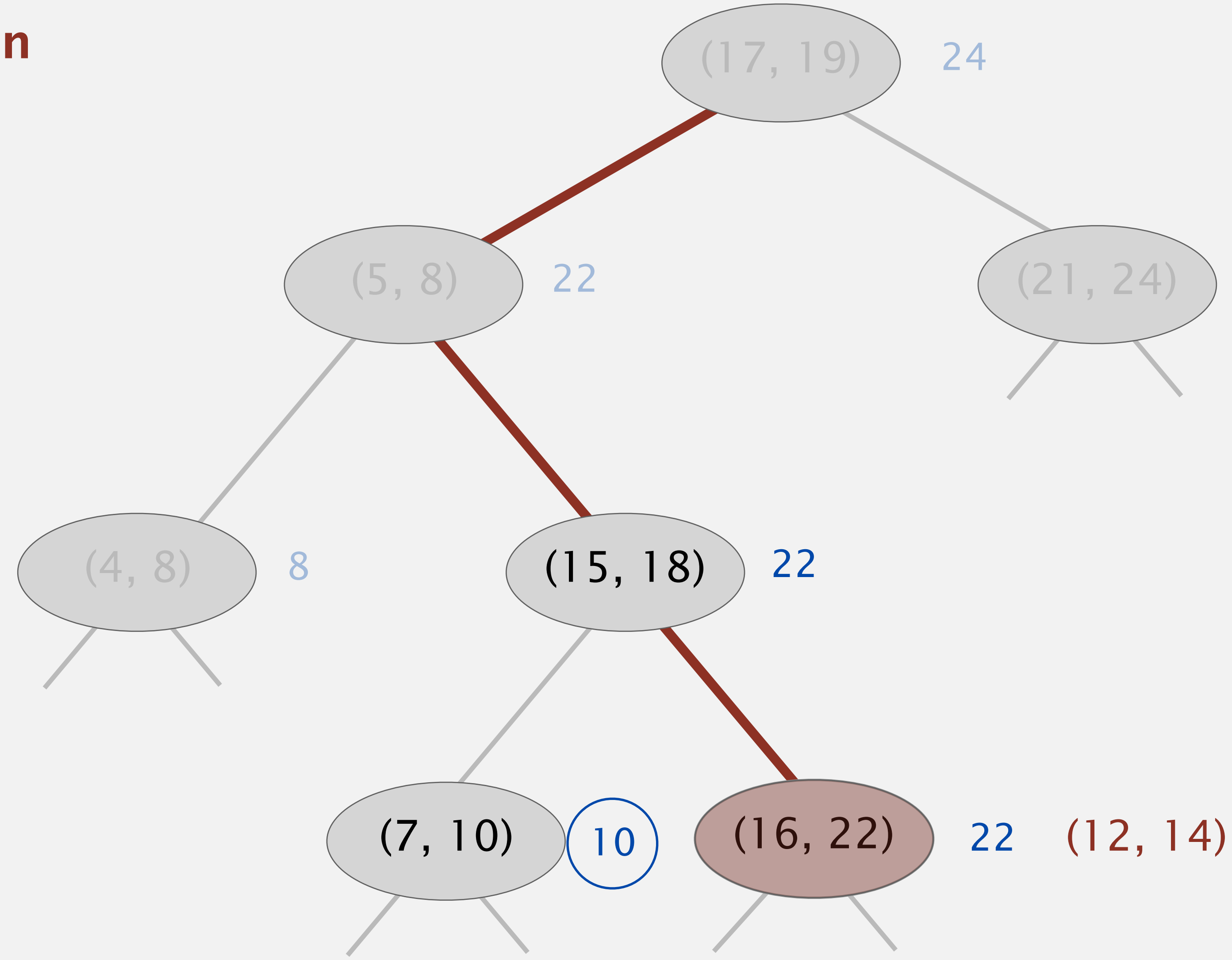
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



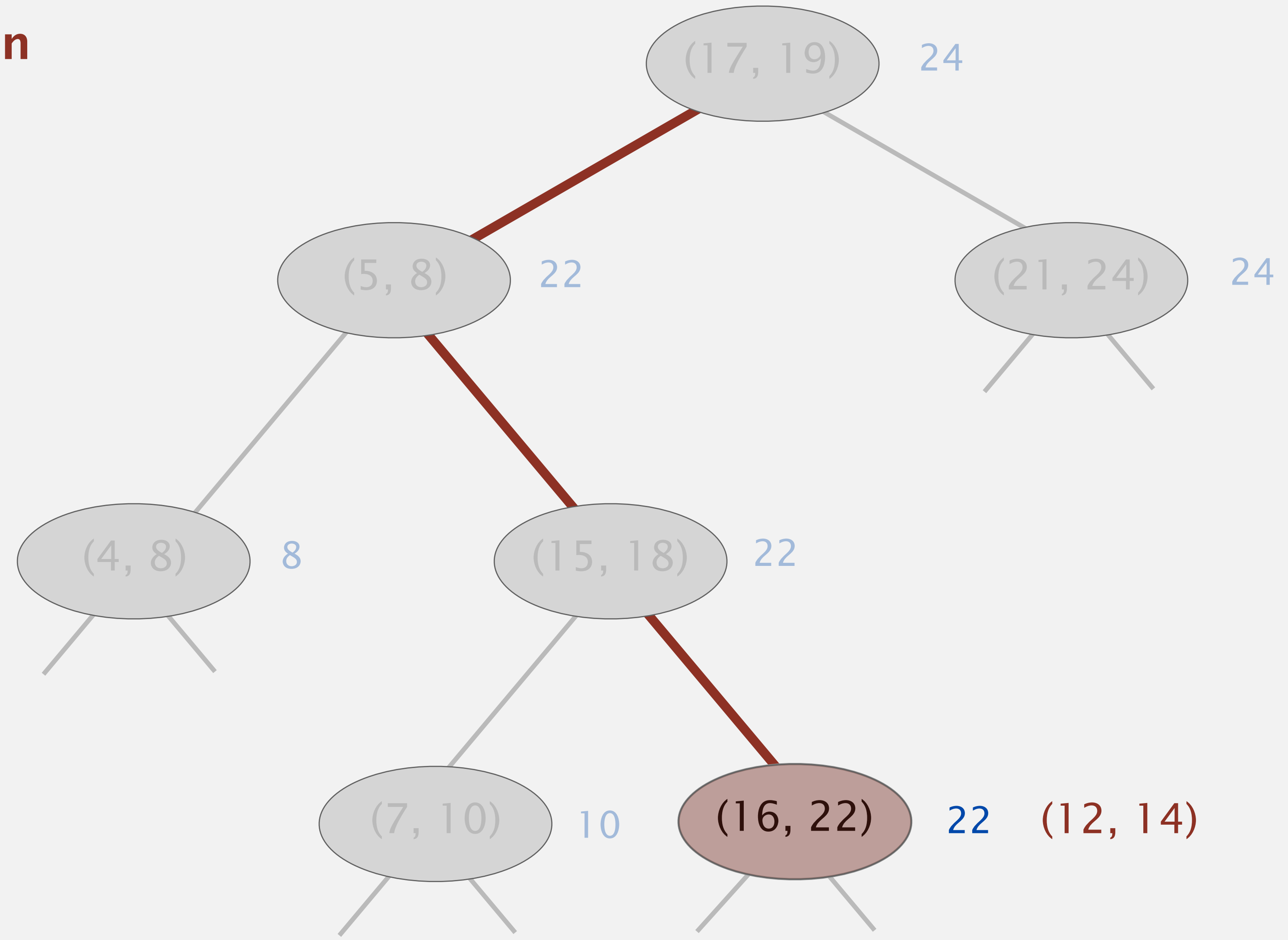
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



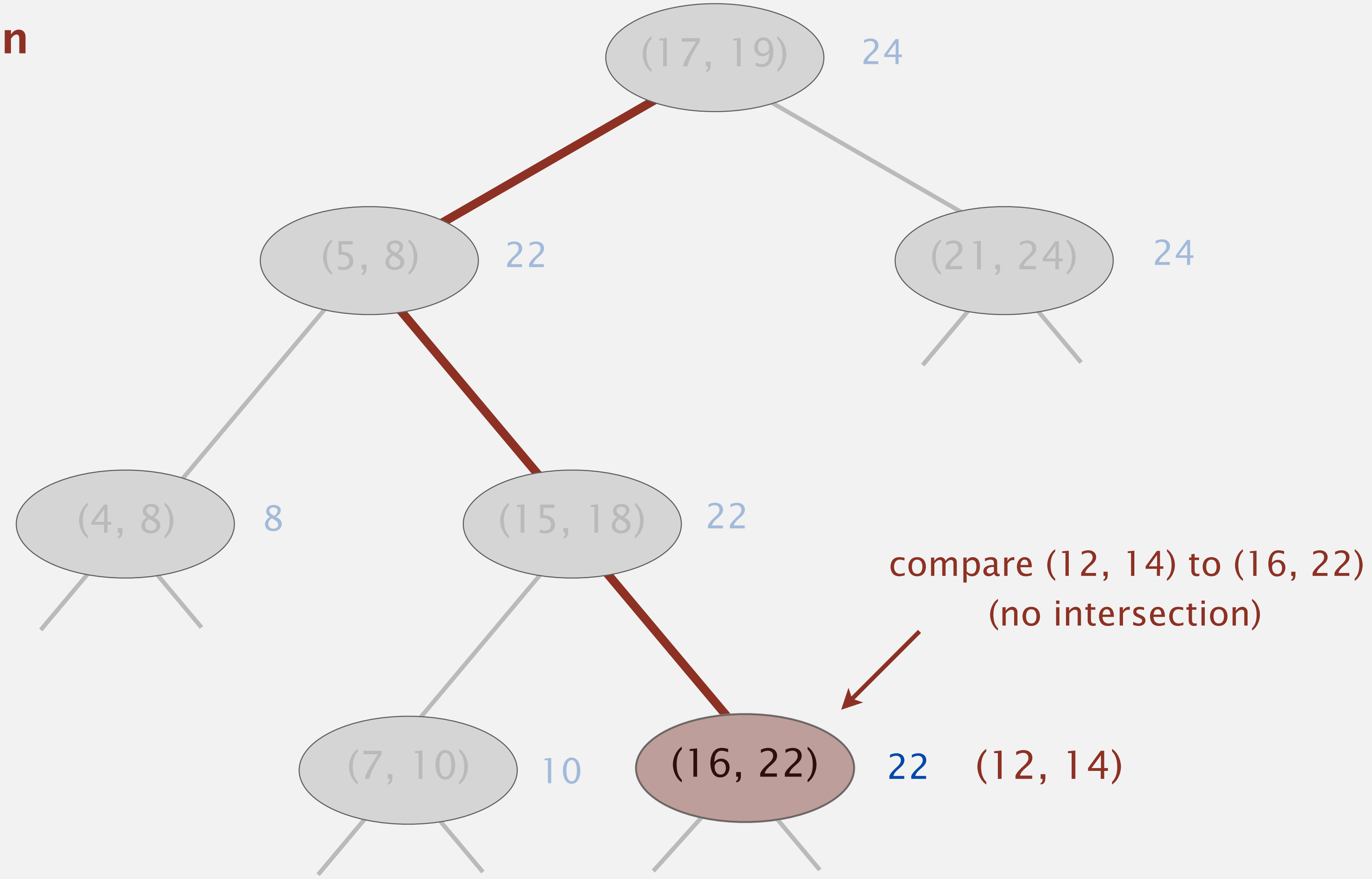
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



compare (12, 14) to (16, 22)
(no intersection)

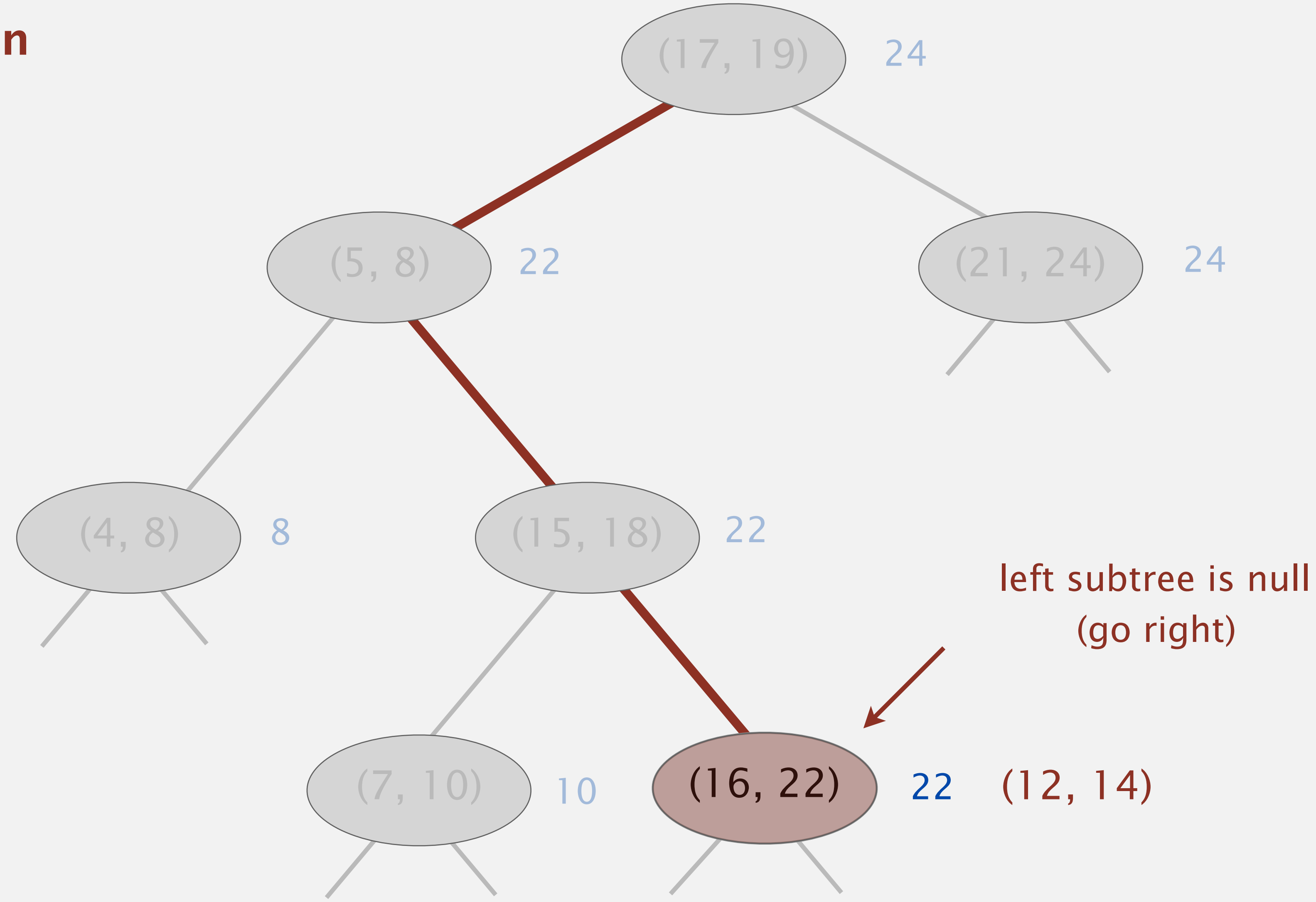
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



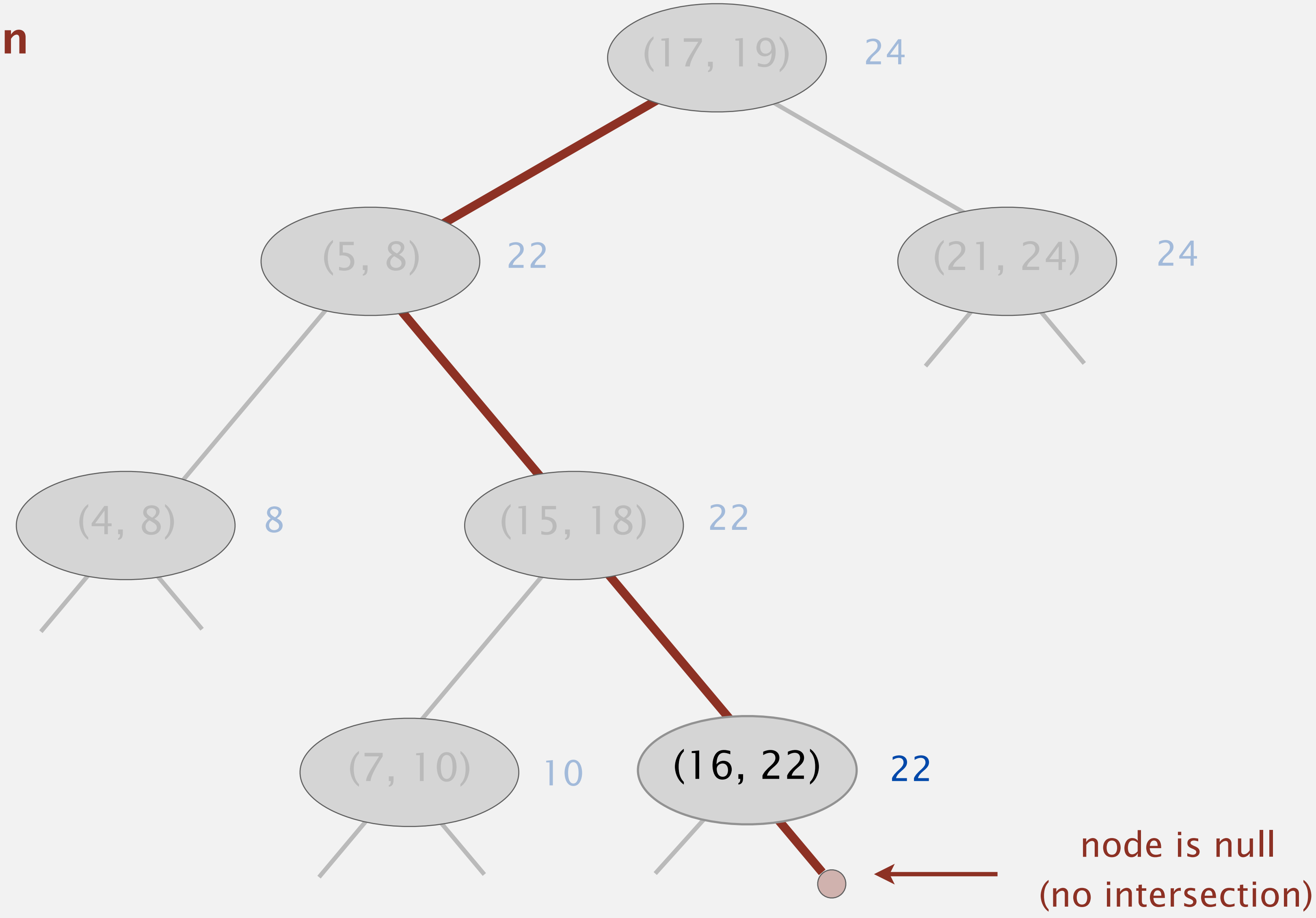
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (12, 14)



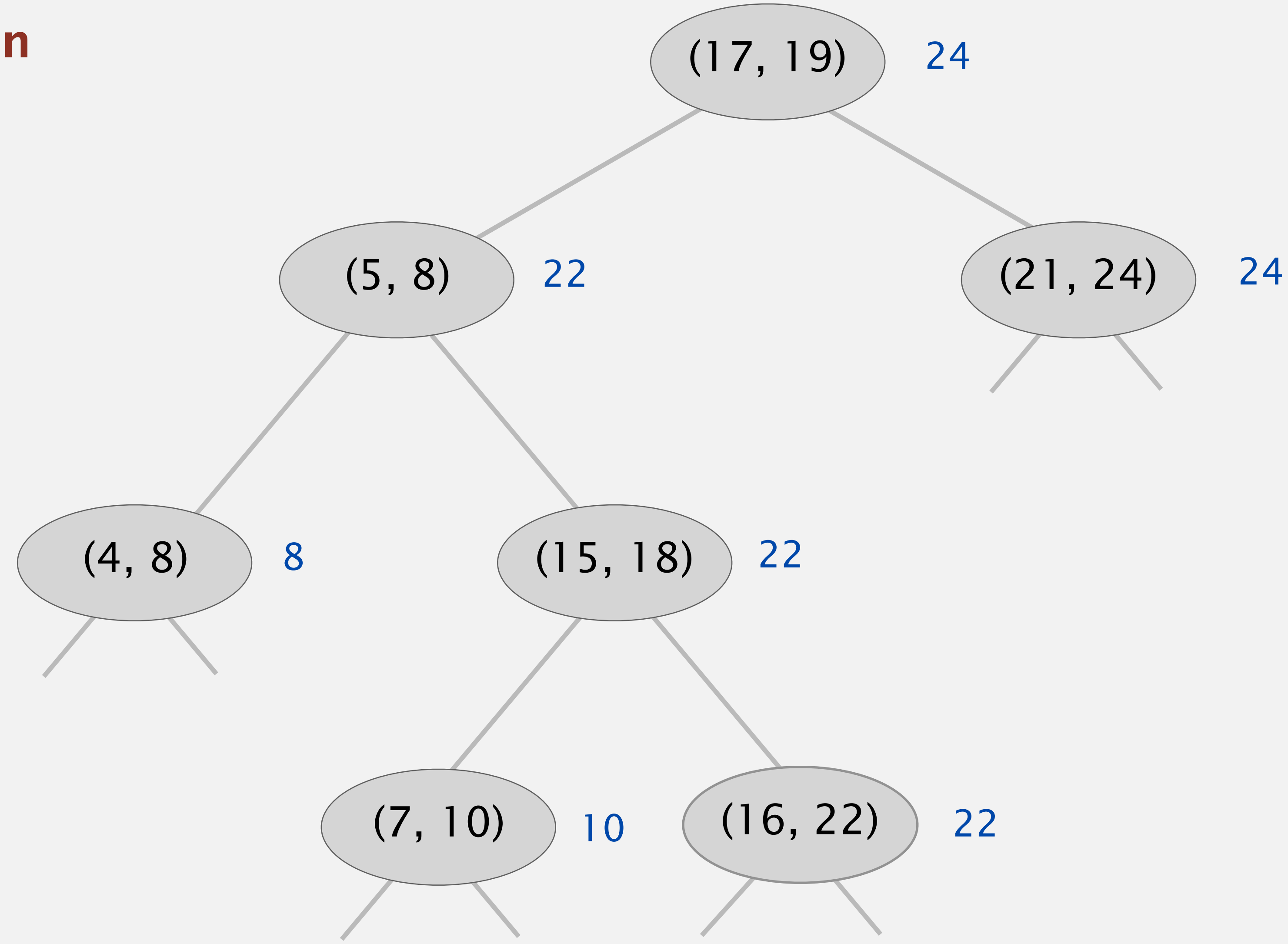
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)



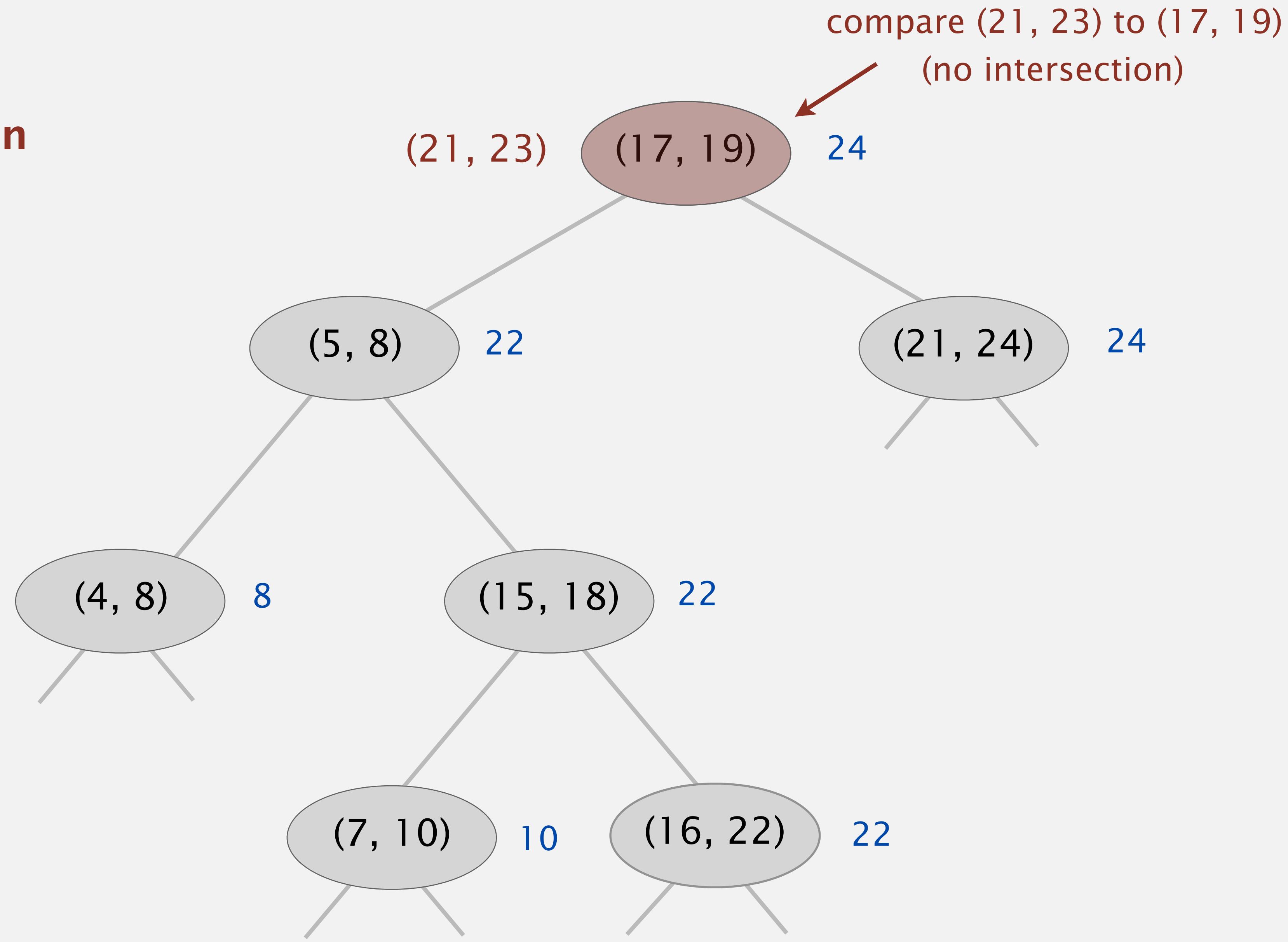
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)



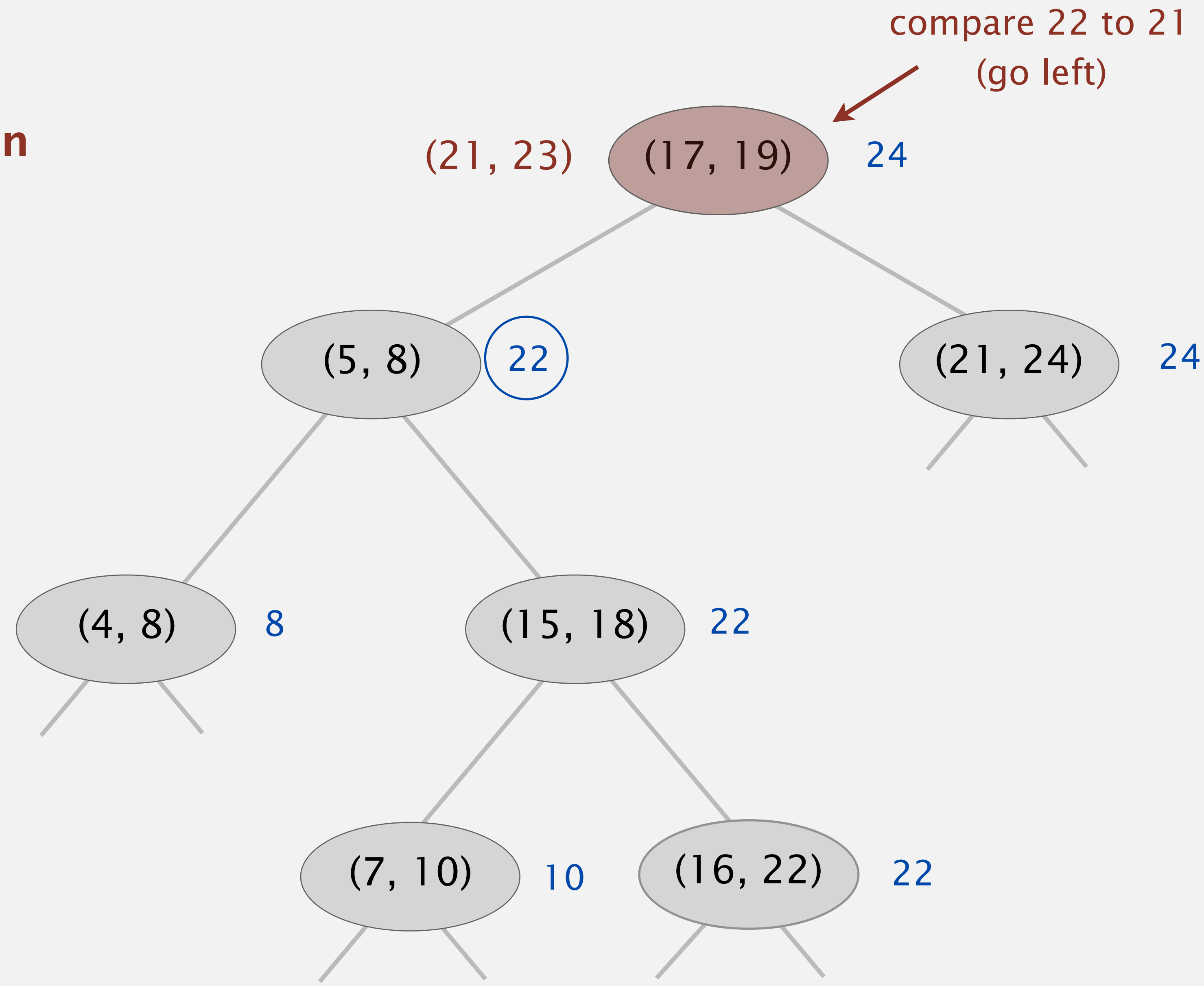
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)



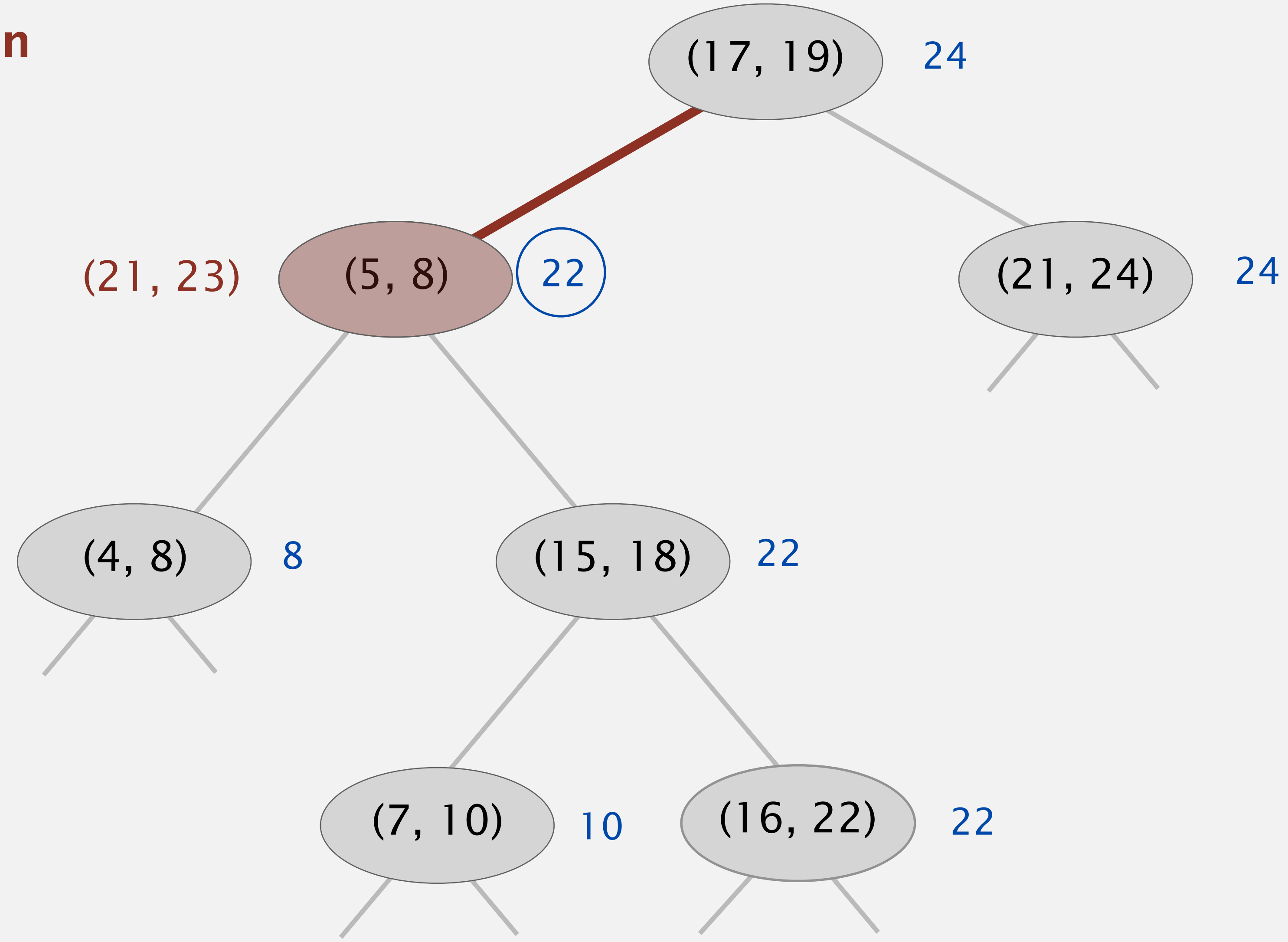
Interval search tree demo: intersection

To search for any one interval that intersects query interval (lo, hi) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo, go right.
- Else go left.

interval intersection

search for (21, 23)



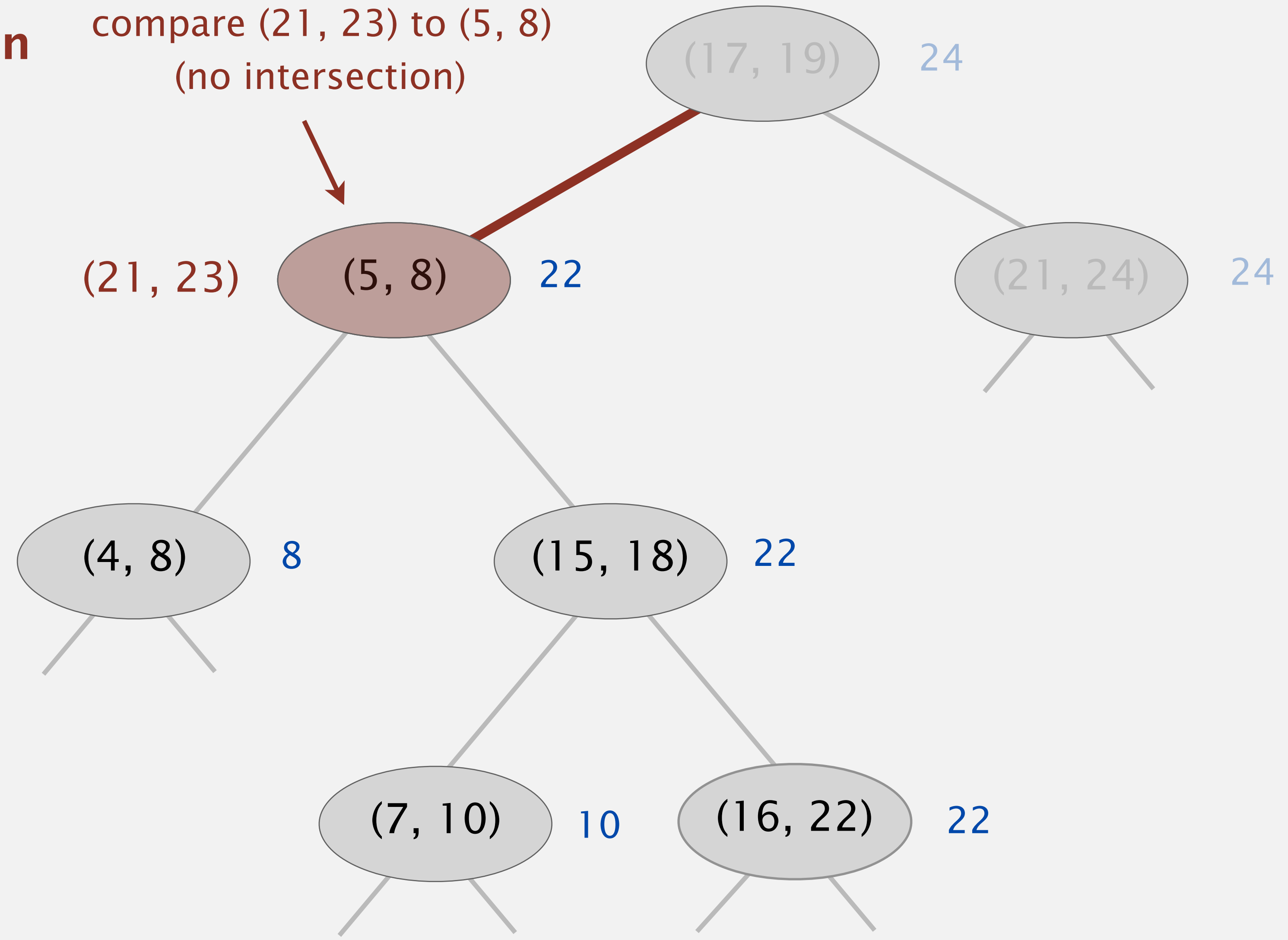
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)



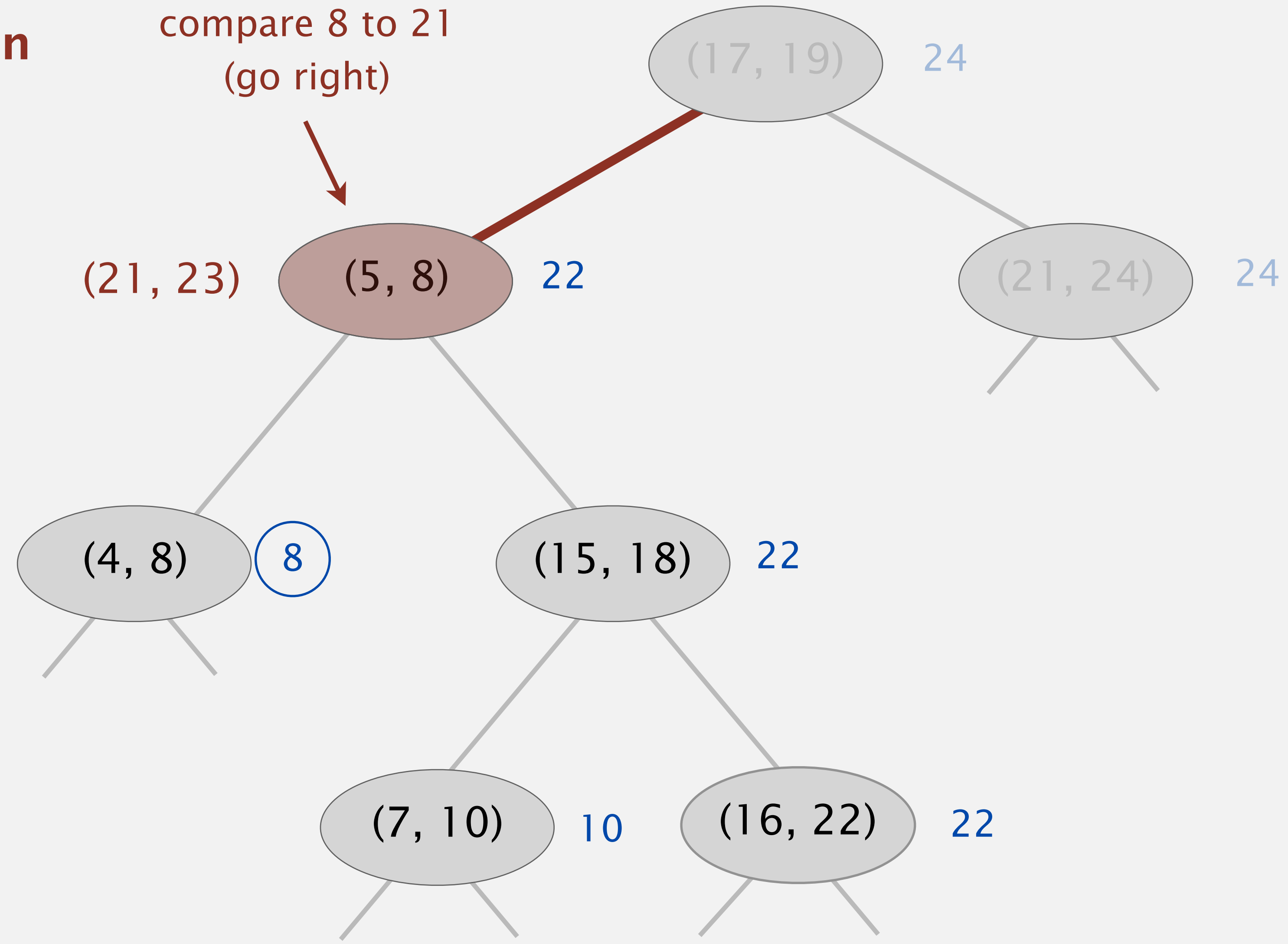
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)



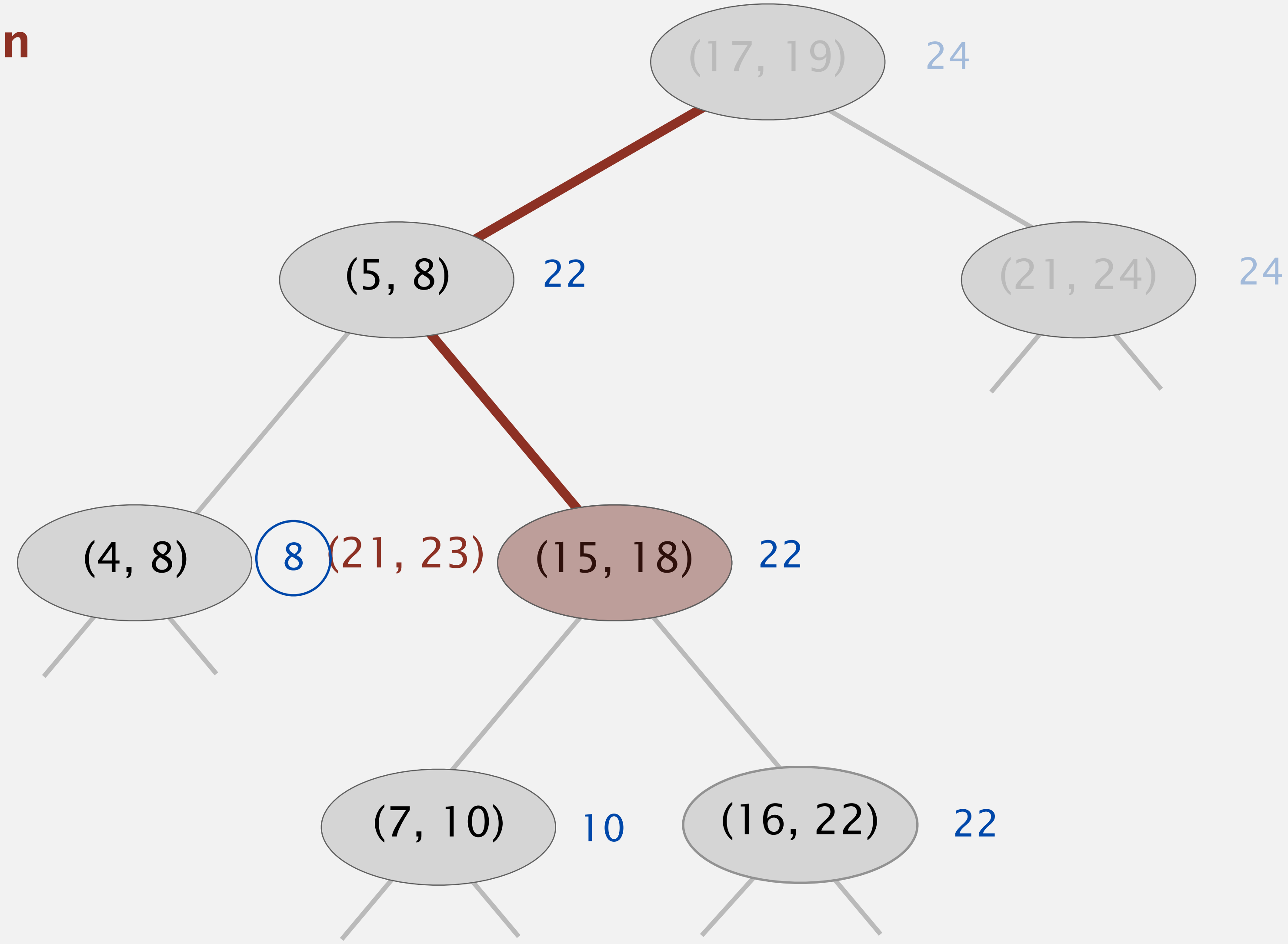
Interval search tree demo: intersection

To search for any one interval that intersects query interval (lo, hi) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo, go right.
- Else go left.

interval intersection

search for (21, 23)



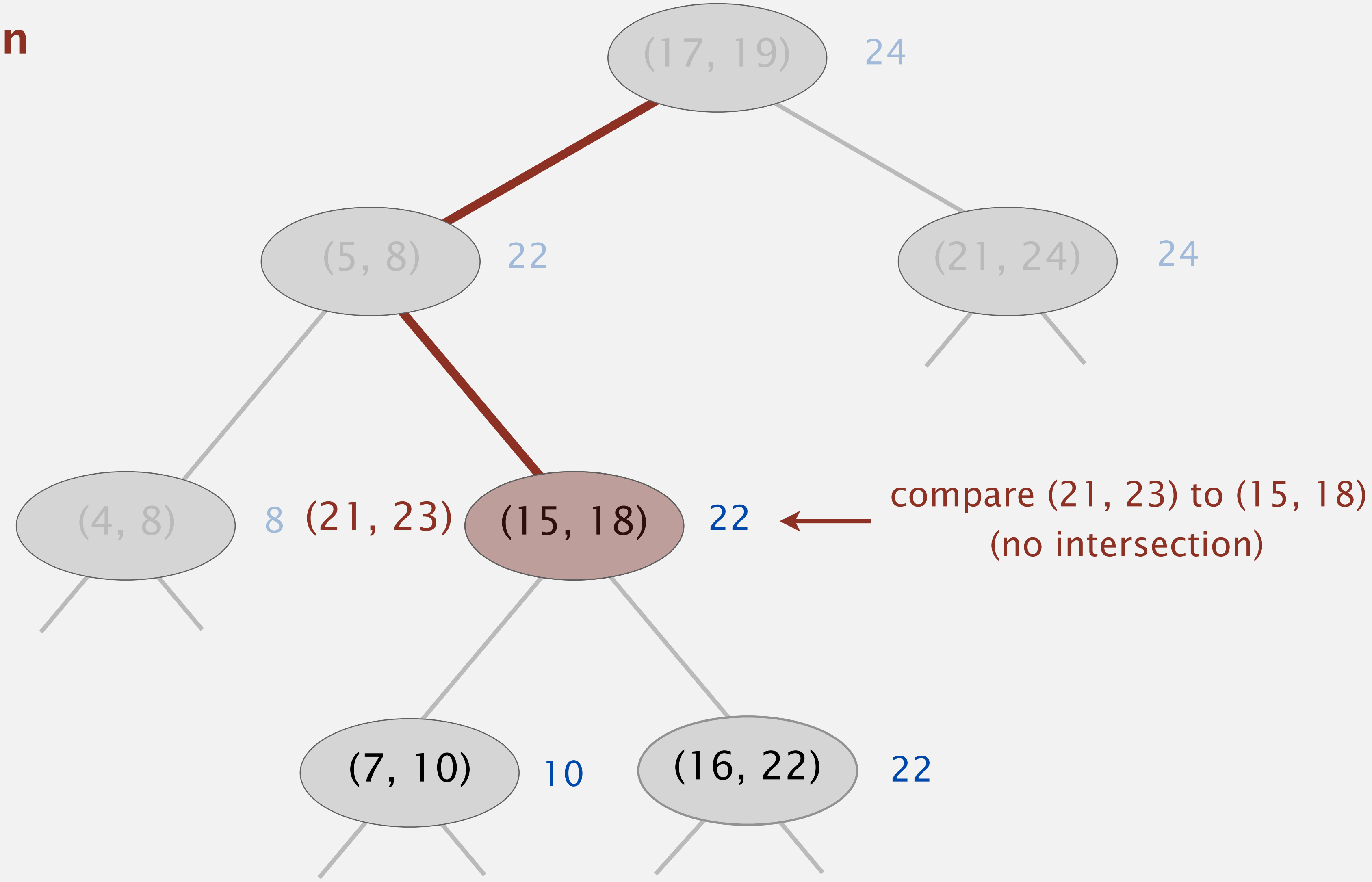
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)



compare (21, 23) to (15, 18)
(no intersection)

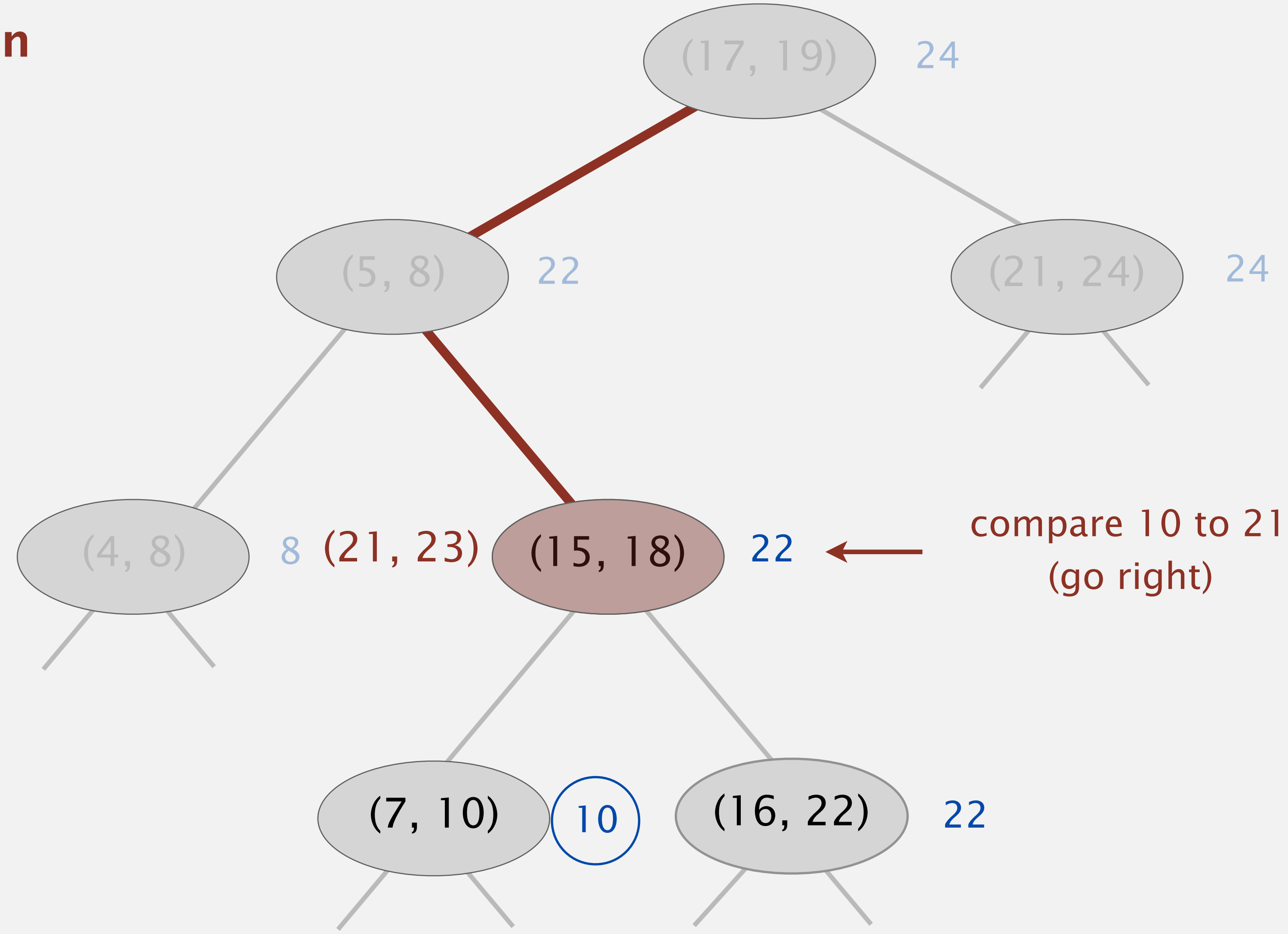
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)



compare 10 to 21
(go right)

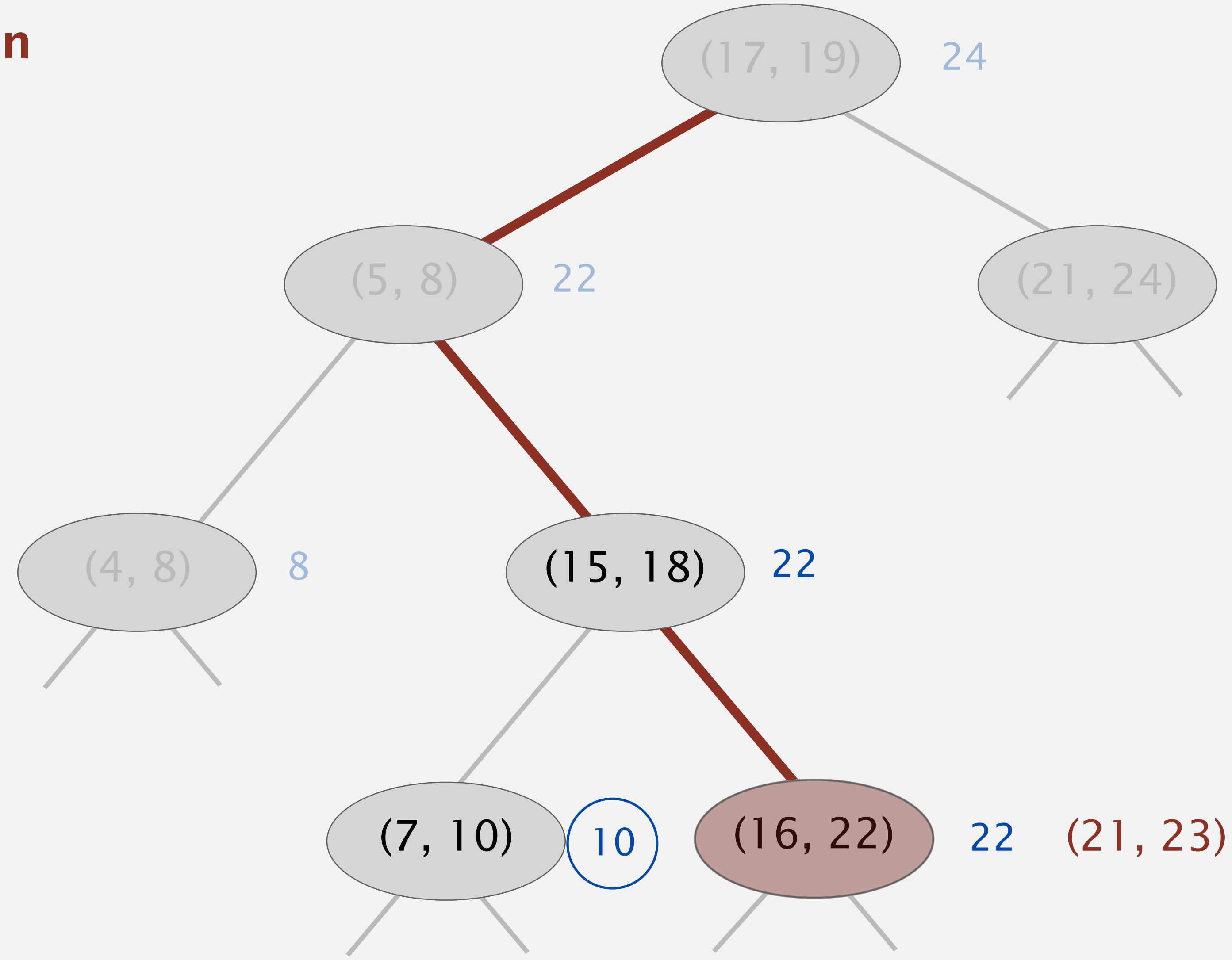
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)



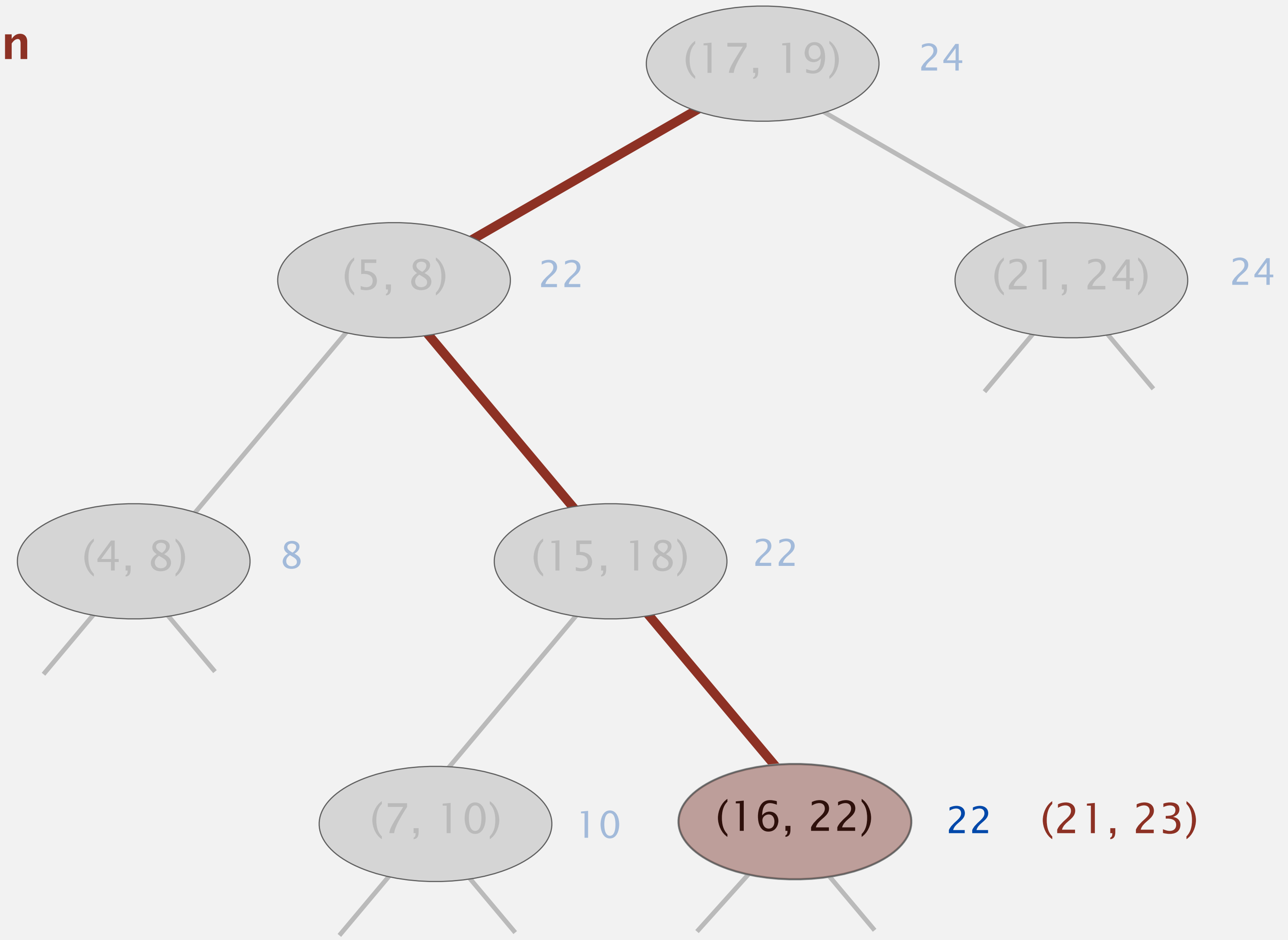
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)



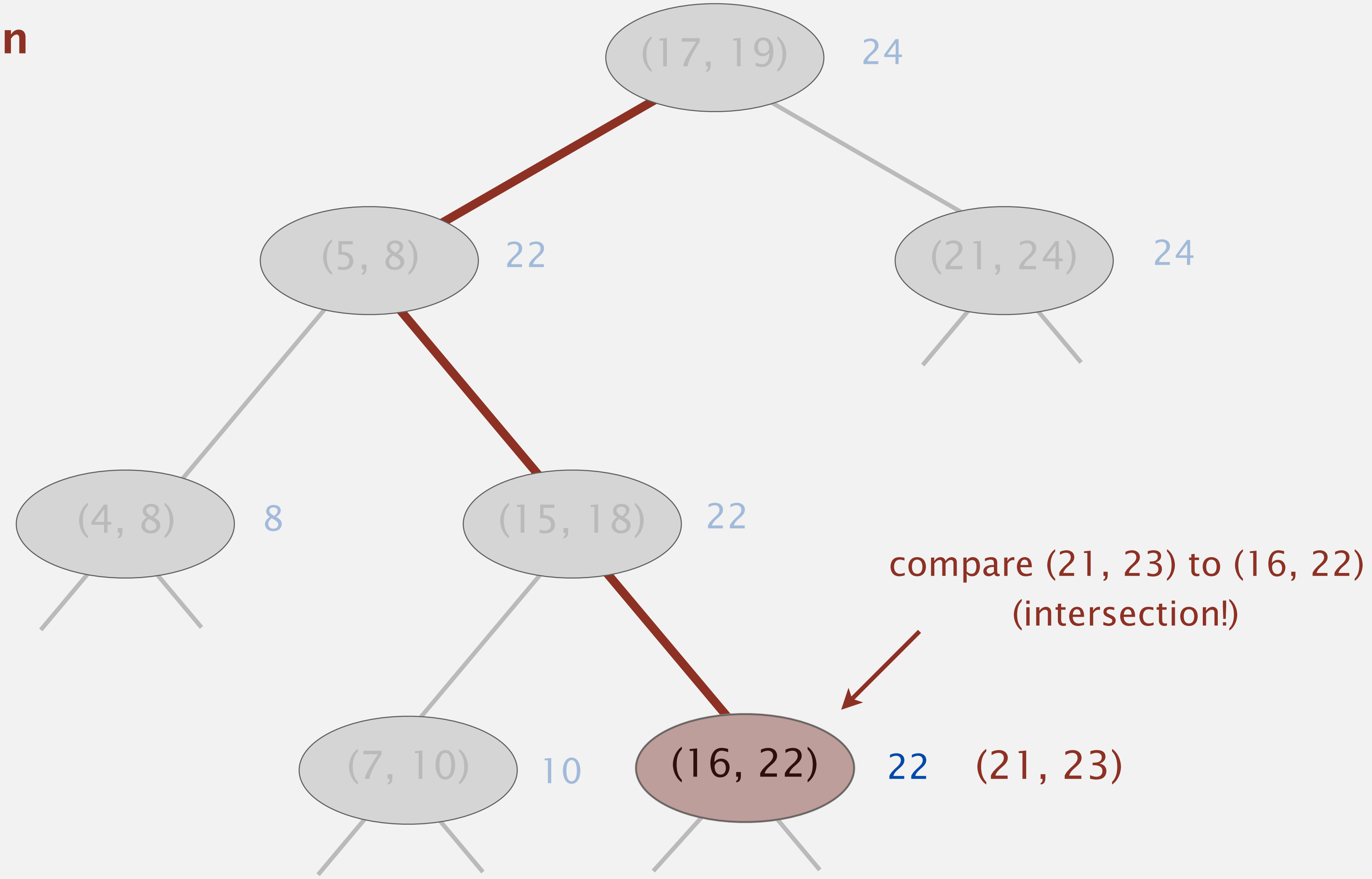
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)



compare (21, 23) to (16, 22)
(intersection!)

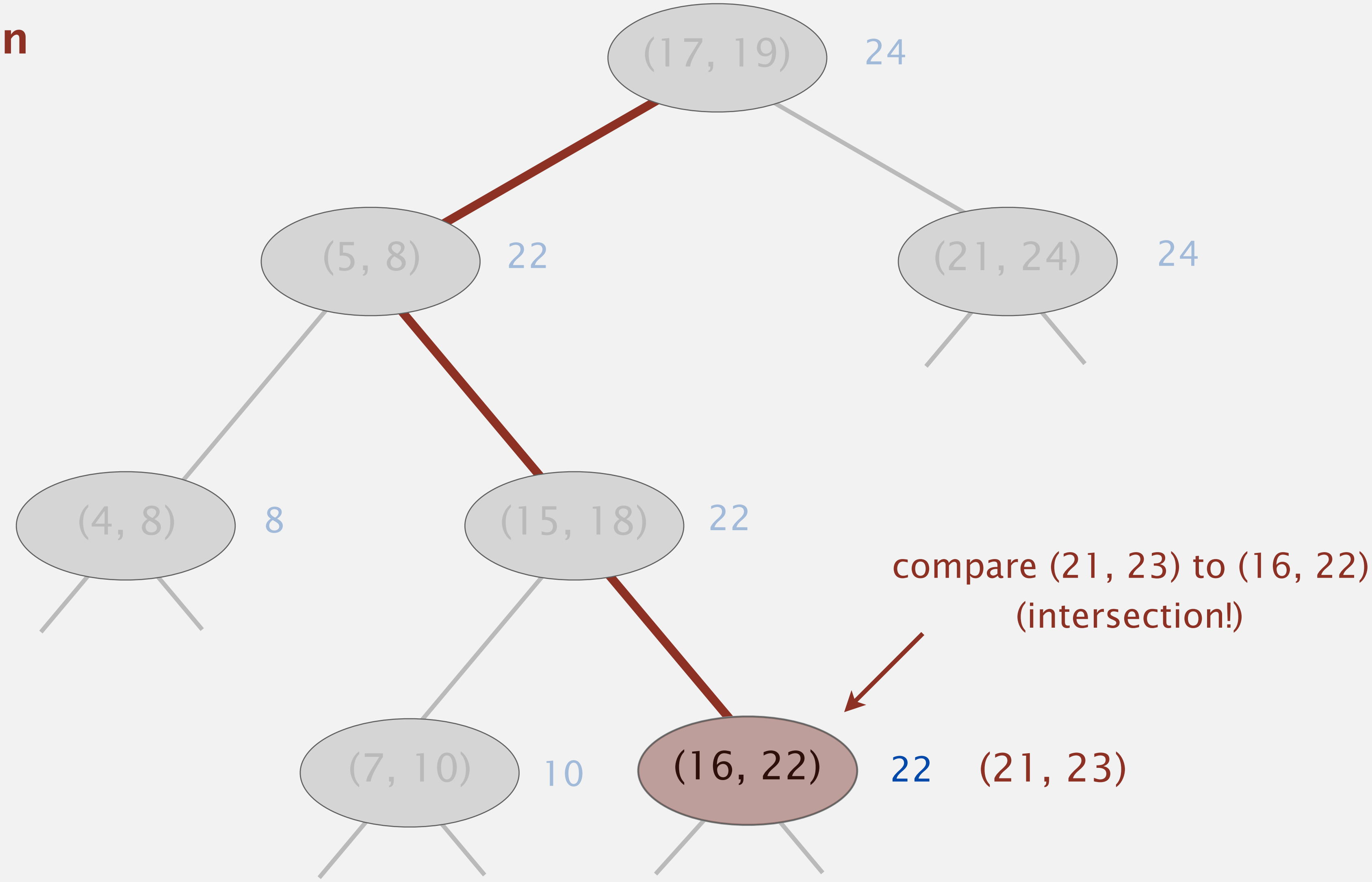
Interval search tree demo: intersection

To search for any one interval that intersects query interval (*lo*, *hi*) :

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than *lo*, go right.
- Else go left.

interval intersection

search for (21, 23)

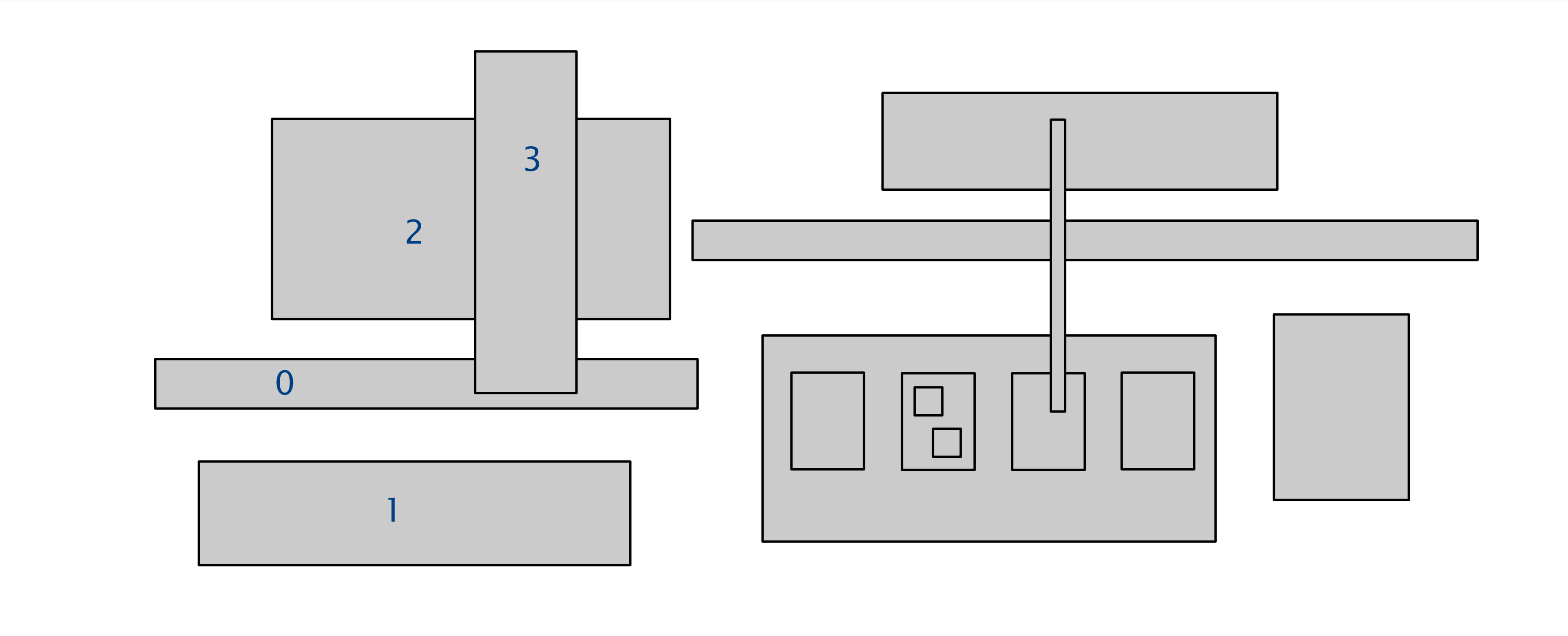


compare (21, 23) to (16, 22)
(intersection!)

Orthogonal rectangle intersection

Goal. Find all intersections among a set of N orthogonal rectangles.

Quadratic algorithm. Check all pairs of rectangles for intersection.



Non-degeneracy assumption. All x- and y-coordinates are distinct.

INTERSECTION RECTANGLES

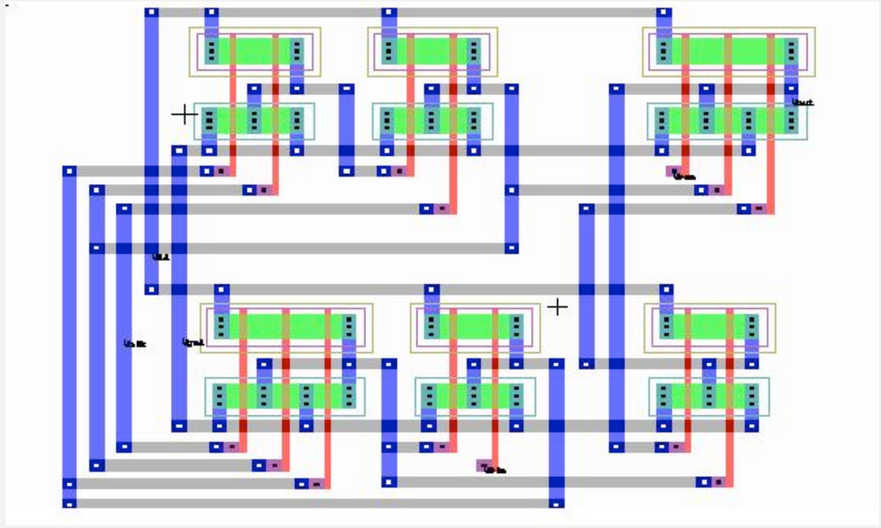
Microprocessors and geometry

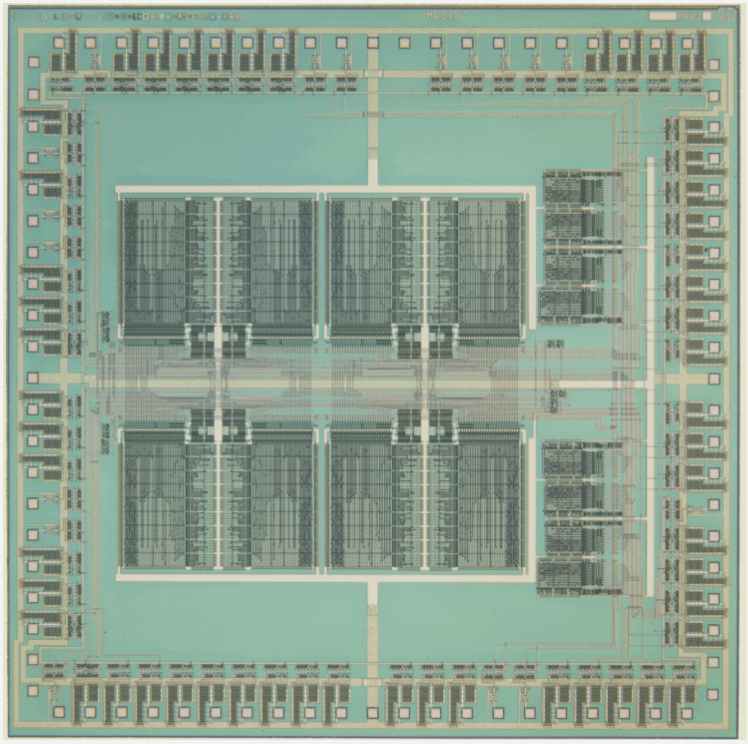
Early 1970s. microprocessor design became a geometric problem.

- Very Large Scale Integration (VLSI).
- Computer-Aided Design (CAD).

Design-rule checking.

- Certain wires cannot intersect.
- Certain spacing needed between different types of wires.
- Debugging = orthogonal rectangle intersection search.

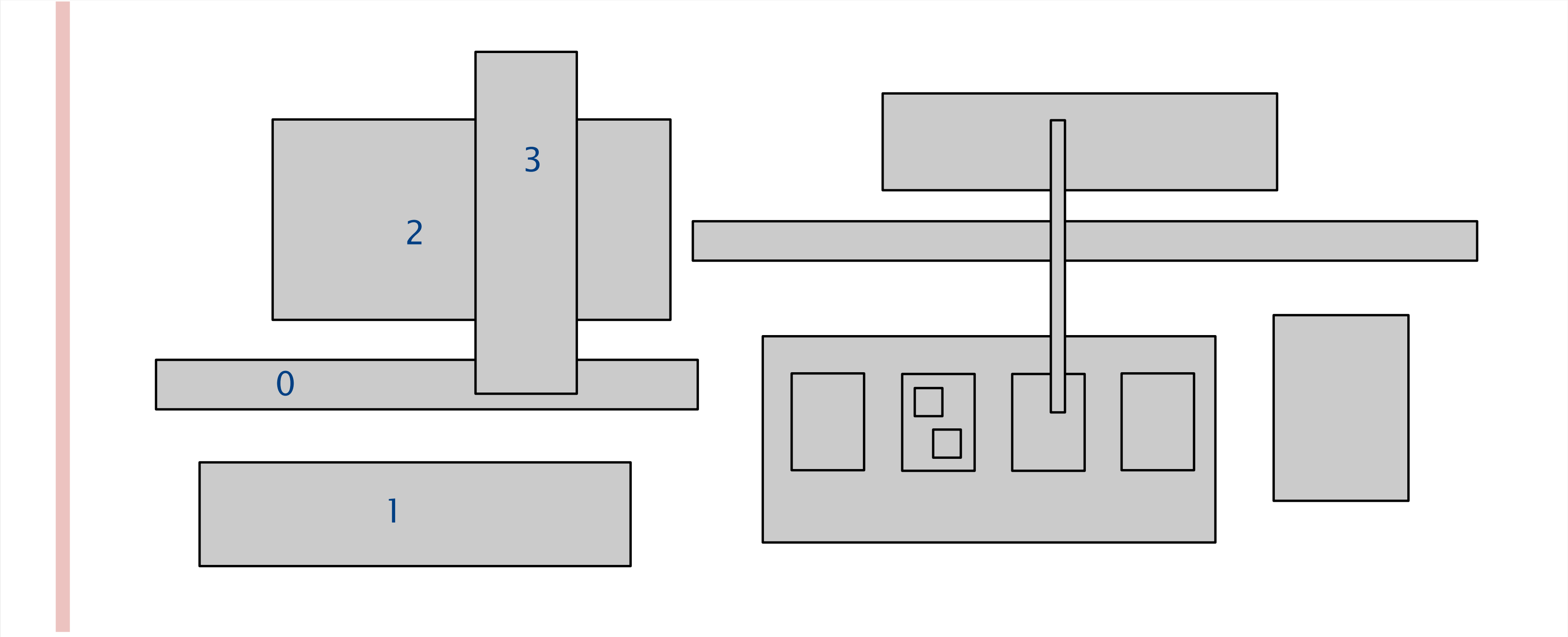


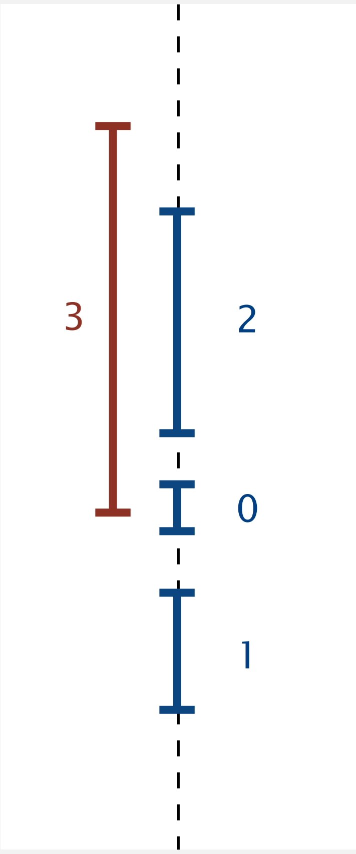


Orthogonal rectangle intersection: sweep-line algorithm

Sweep vertical line from left to right.

- x-coordinates of left and right endpoints define events.
- Maintain set of rectangles that intersect the sweep line in an interval search tree (using y-intervals of rectangle).
- Left endpoint: interval search for y-interval of rectangle; insert y-interval.
- Right endpoint: remove y-interval.





y–coordinates

Orthogonal rectangle intersection: sweep-line analysis

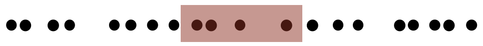
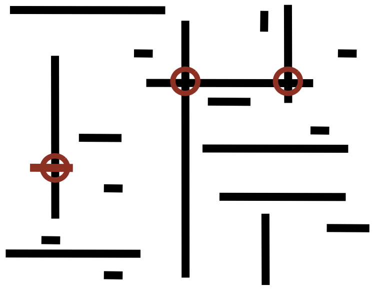
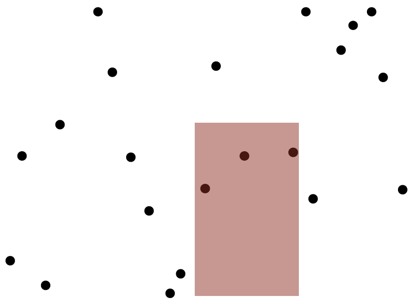
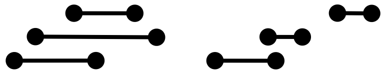
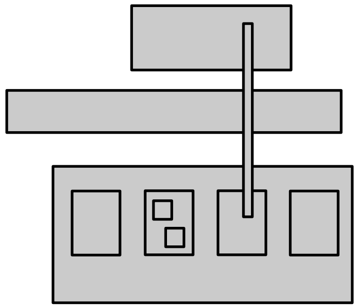
Proposition. Sweep line algorithm takes time proportional to $N \log N + R \log N$ to find R intersections among a set of N rectangles.

Pf.

- Put x-coordinates on a PQ (or sort). ⟵ N log N
- Insert y-intervals into ST. ⟵ N log N
- Delete y-intervals from ST. ⟵ N log N
- Interval searches for y-intervals. ⟵ N log N + R log N

Bottom line. Sweep line reduces 2d orthogonal rectangle intersection search to 1d interval search.

Geometric applications of BSTs

problem	example	solution
1d range search		BST
2d orthogonal line segment intersection		sweep line reduces to 1d range search
kd range search		kd tree
1d interval search		interval search tree
2d orthogonal rectangle intersection		sweep line reduces to 1d interval search

http://algs4.cs.princeton.edu

GEOMETRIC APPLICATIONS OF

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