Computational Geometry
[csci 3250]

Laura Toma

Bowdoin College

Line segment intersection

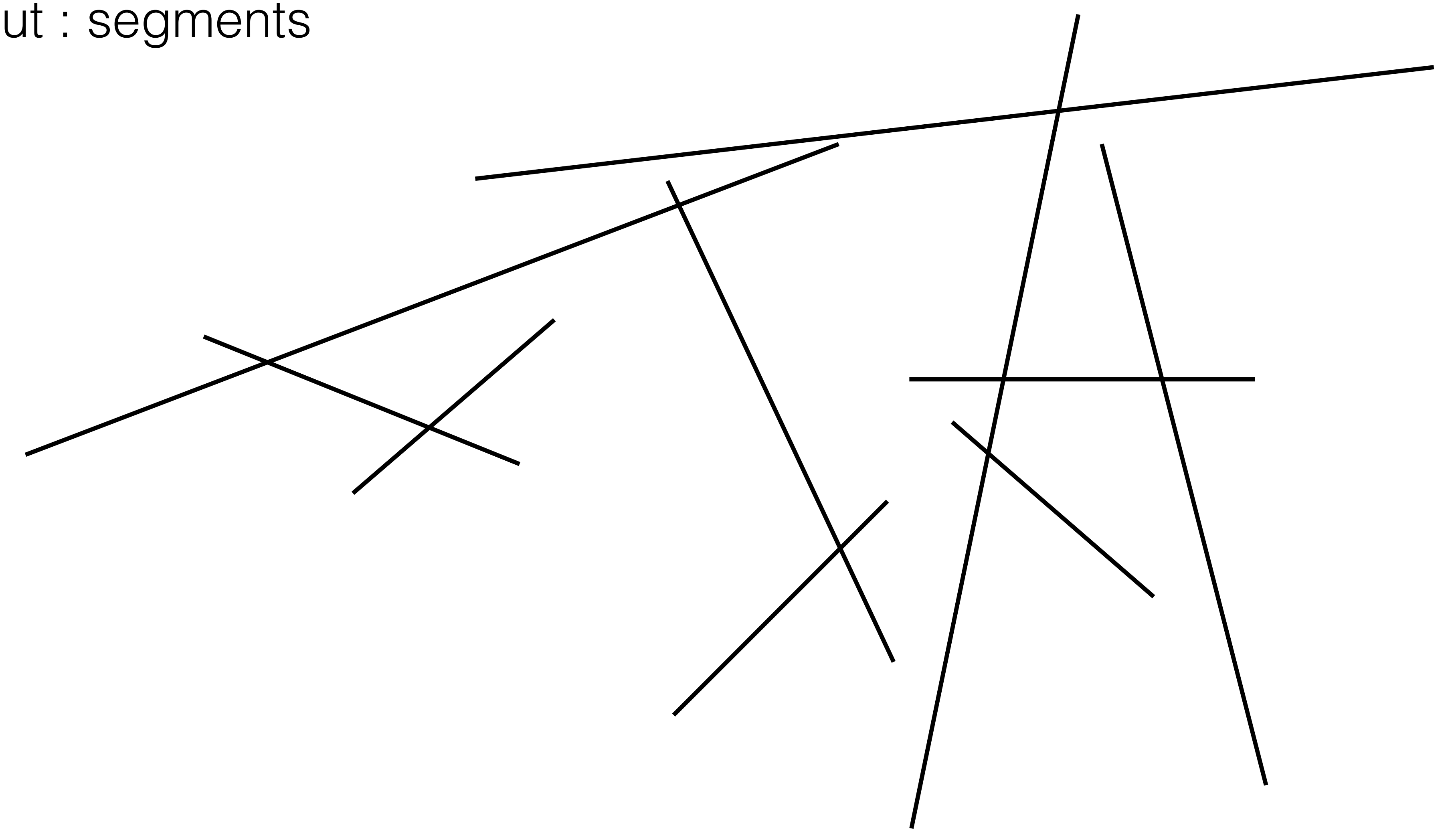
Line segment intersection

Given a set of line segments in 2D, find (report) all their pairwise intersections.

Line segment intersection

Given a set of line segments in 2D, find (report) all their pairwise intersections.

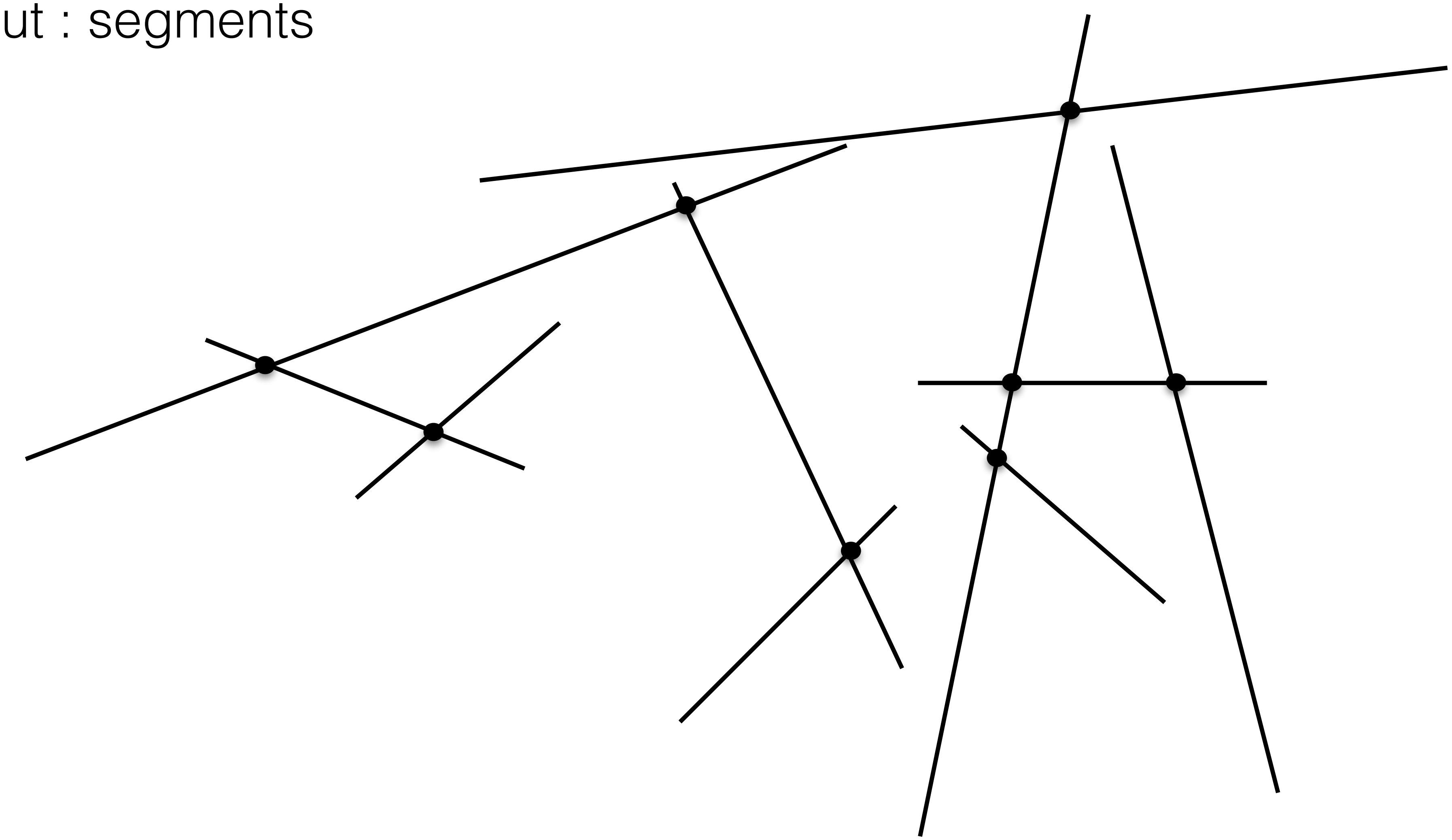
- Input : segments



Line segment intersection

Given a set of line segments in 2D, find (report) all their pairwise intersections.

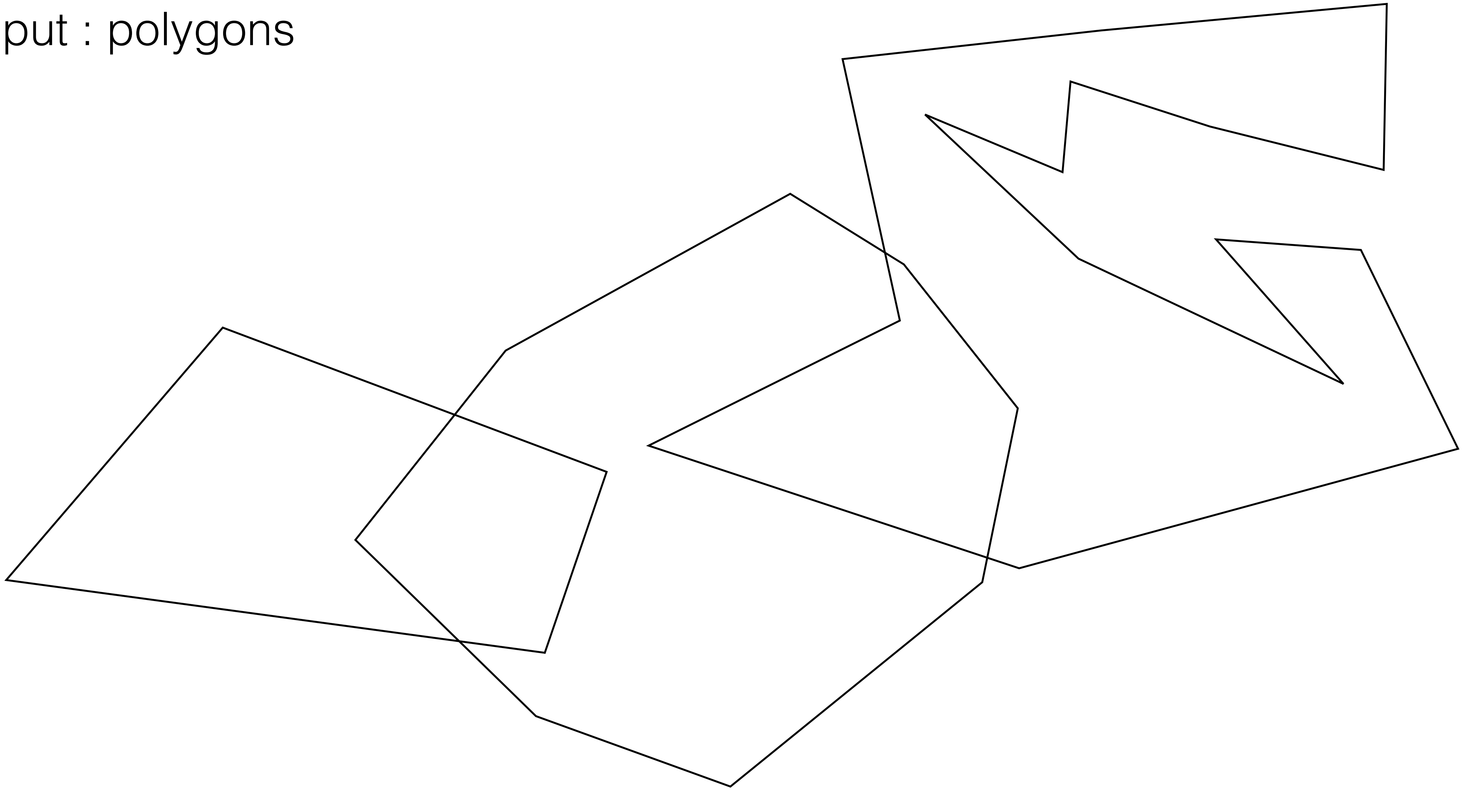
- Input : segments



Line segment intersection

Given a set of line segments in 2D, find (report) all their pairwise intersections.

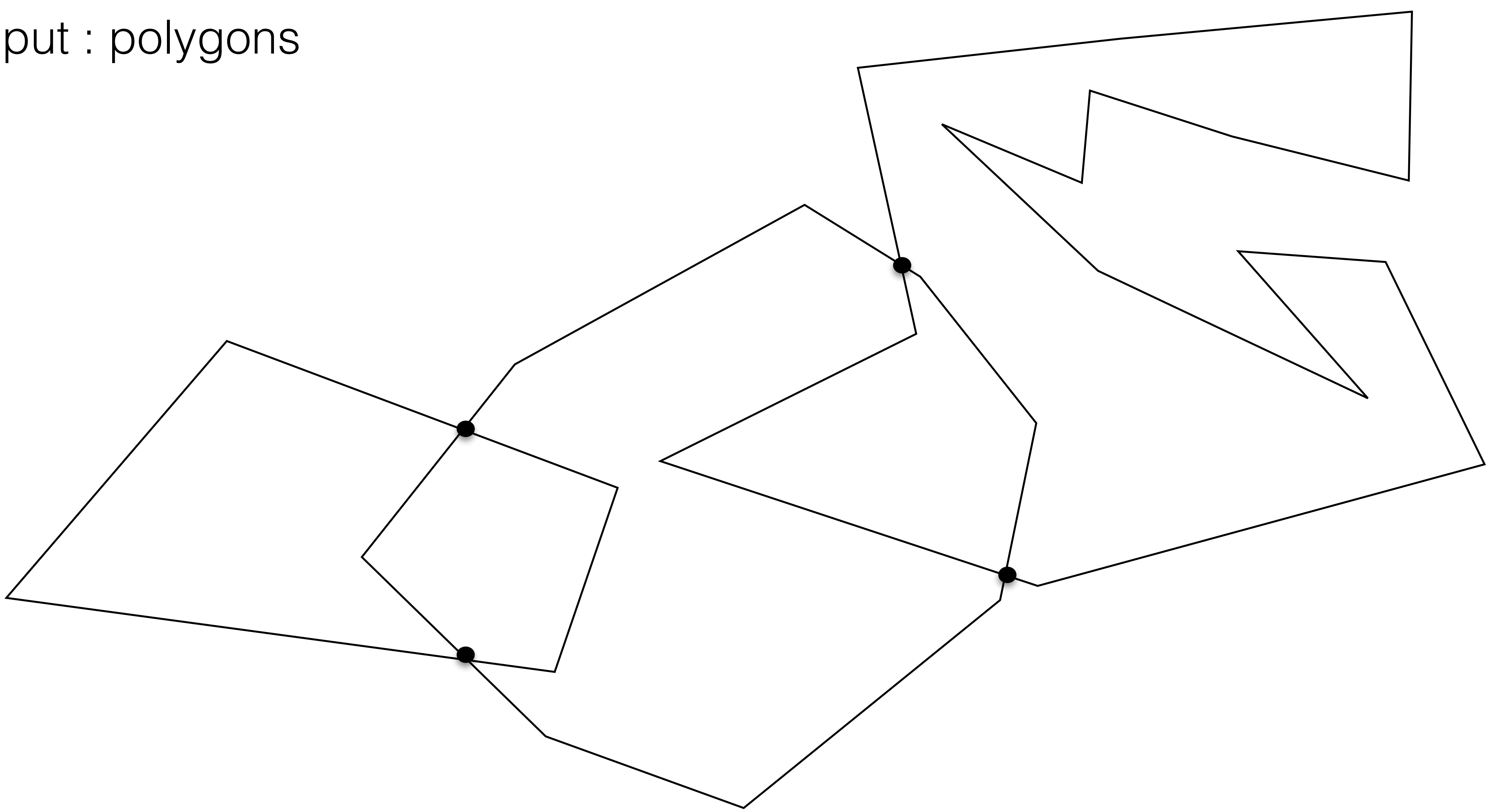
- Input : polygons



Line segment intersection

Given a set of line segments in 2D, find (report) all their pairwise intersections.

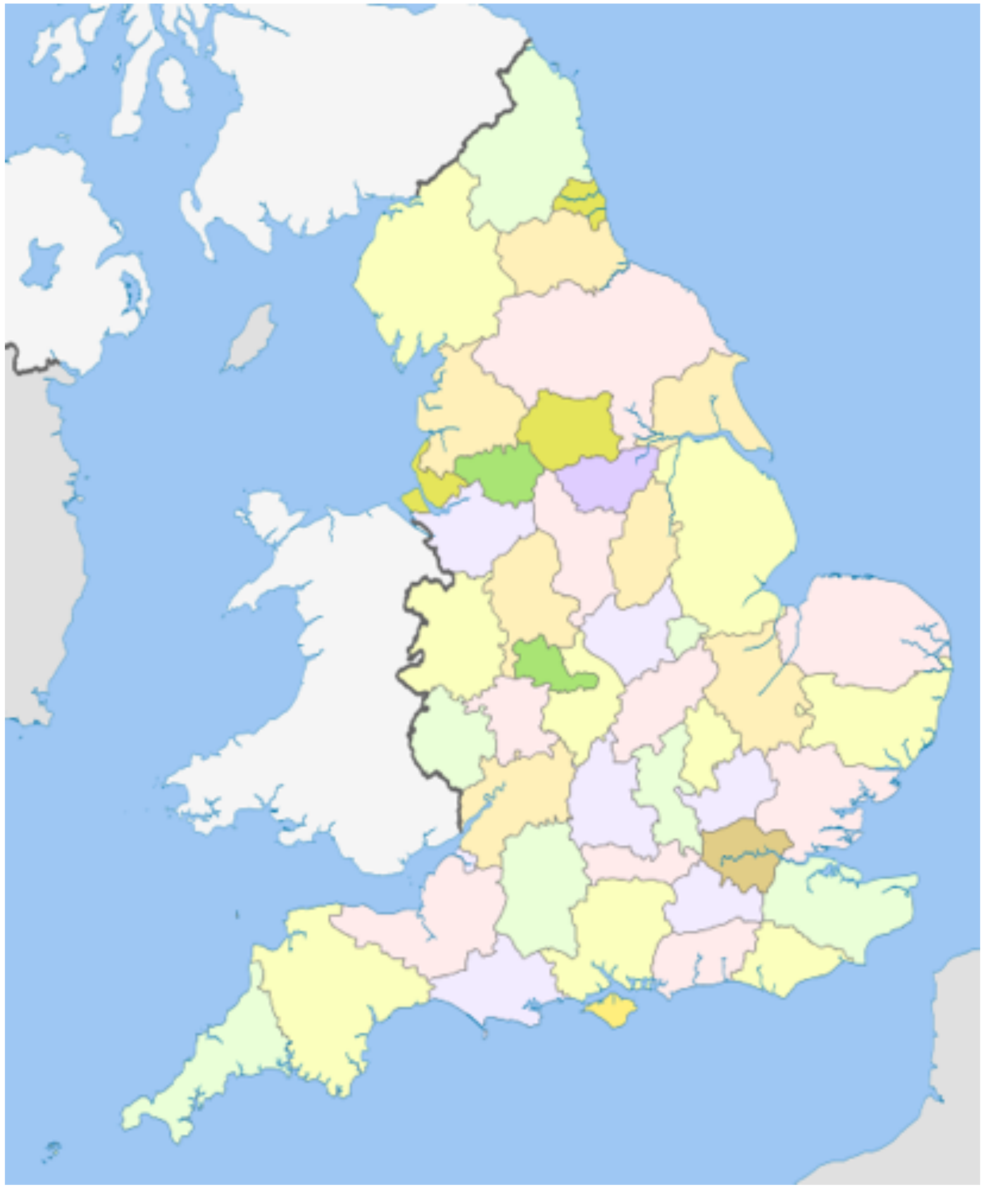
- Input : polygons



Line segment intersection

Given a set of line segments in 2D, find (report) all their pairwise intersections.

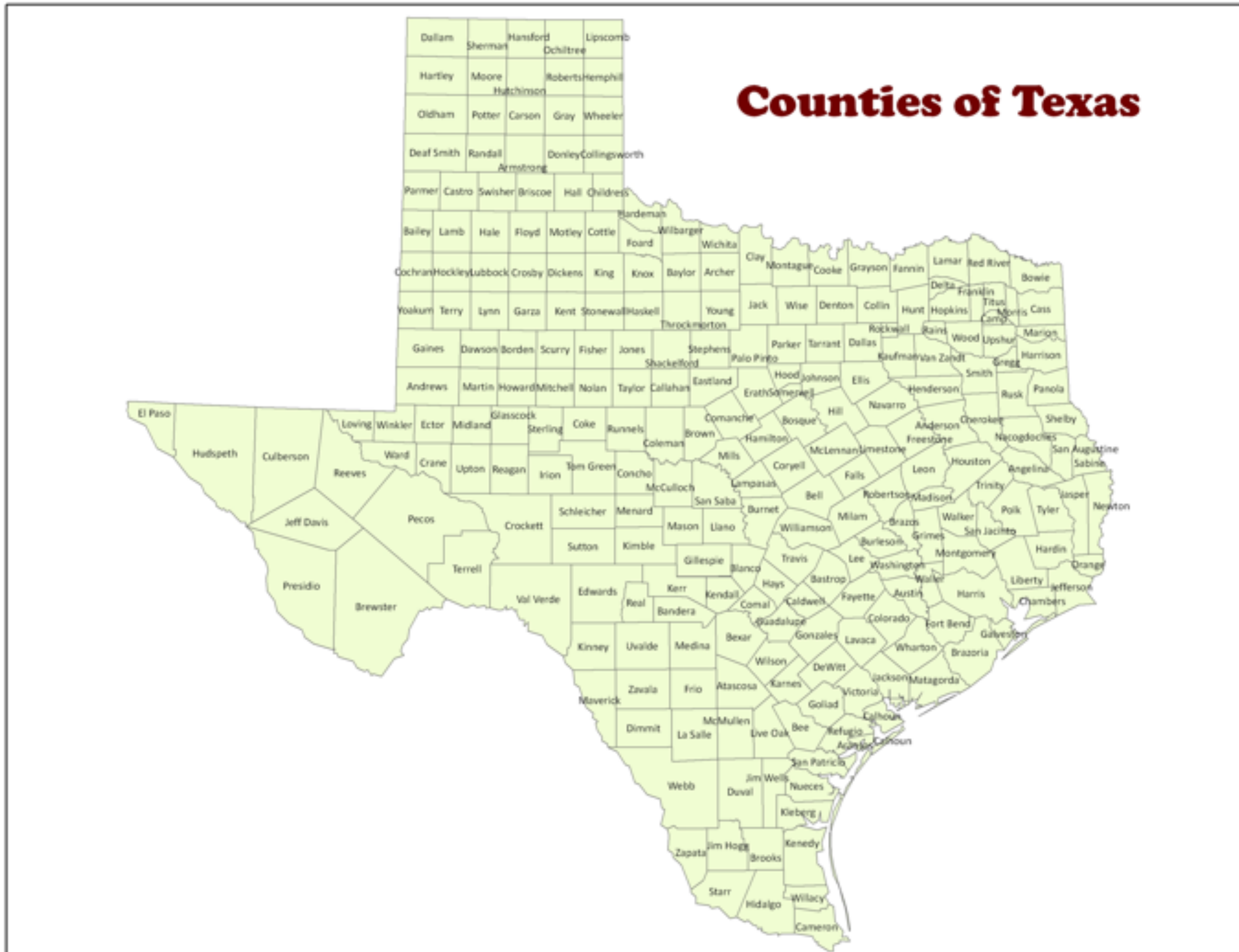
- Input : planar maps



Line segment intersection

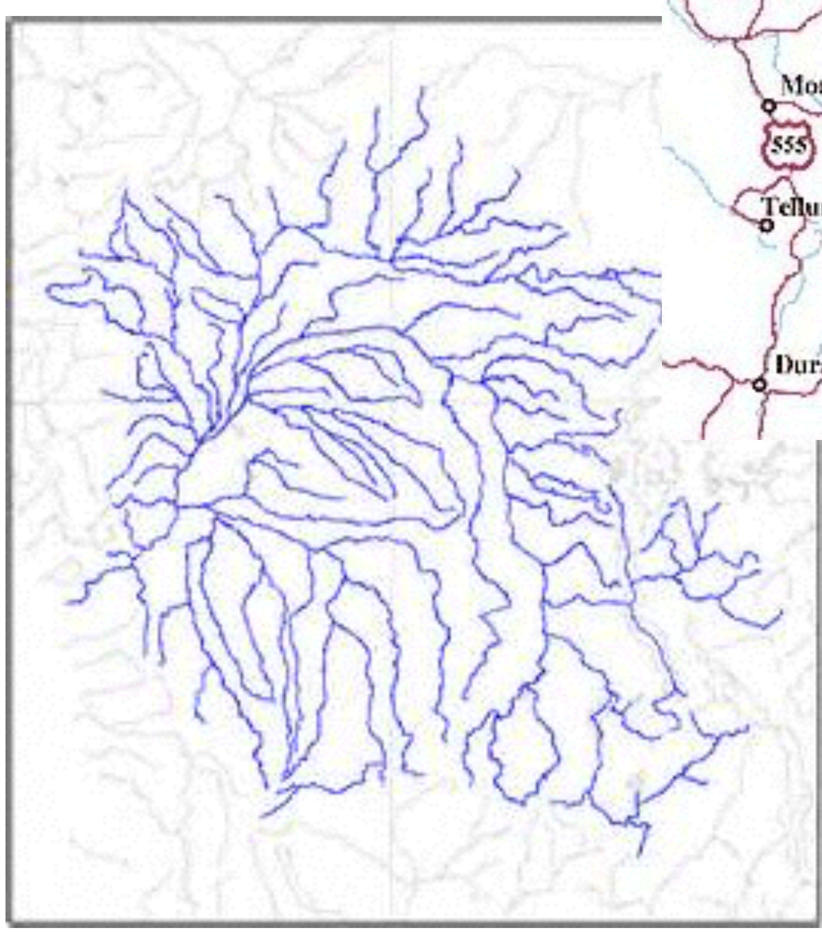
Given a set of line segments in 2D, find (report) all their pairwise intersections.

- Input : planar maps



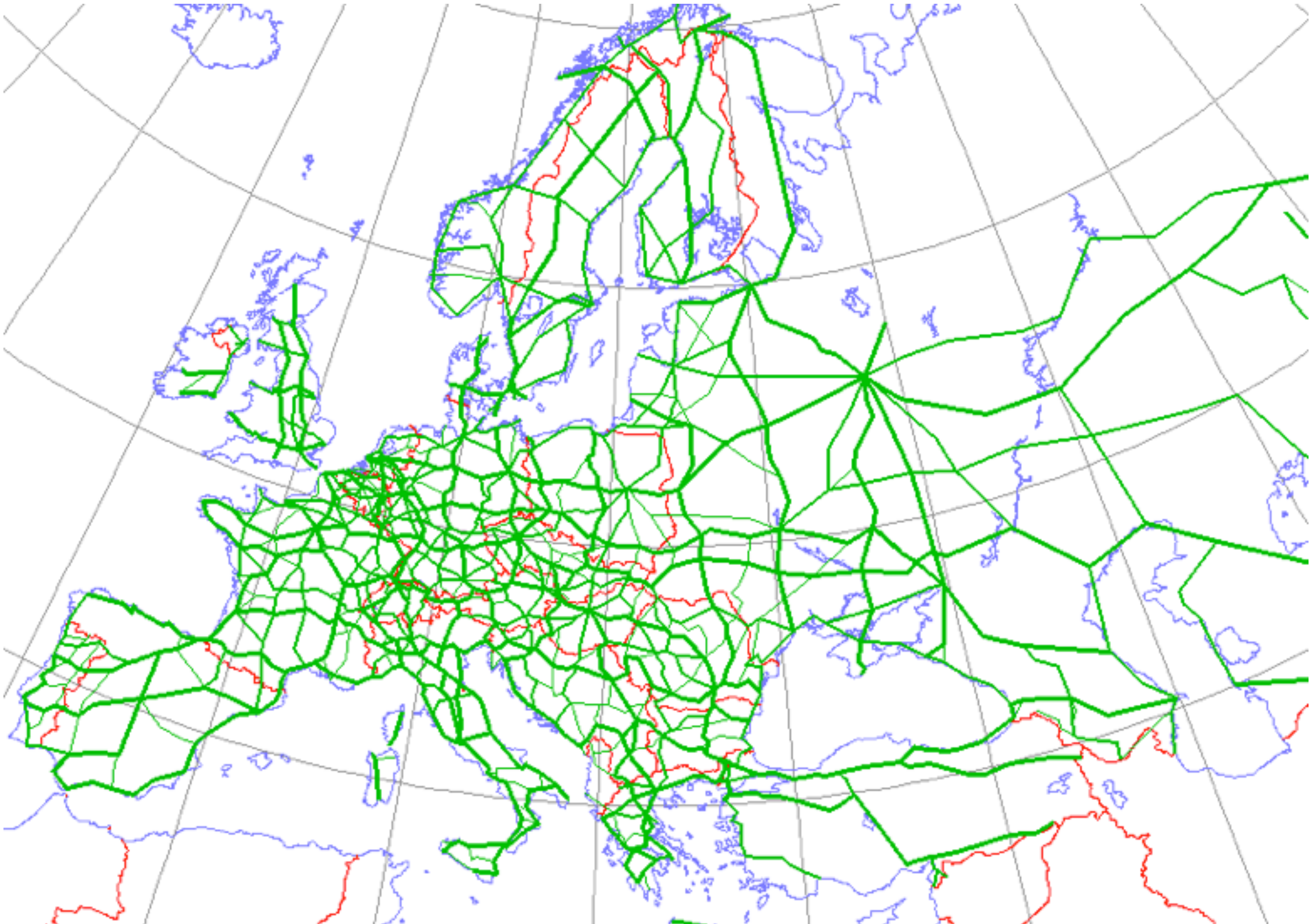
Applications

Segment data in GIS: river networks, road networks, railways, counties, etc



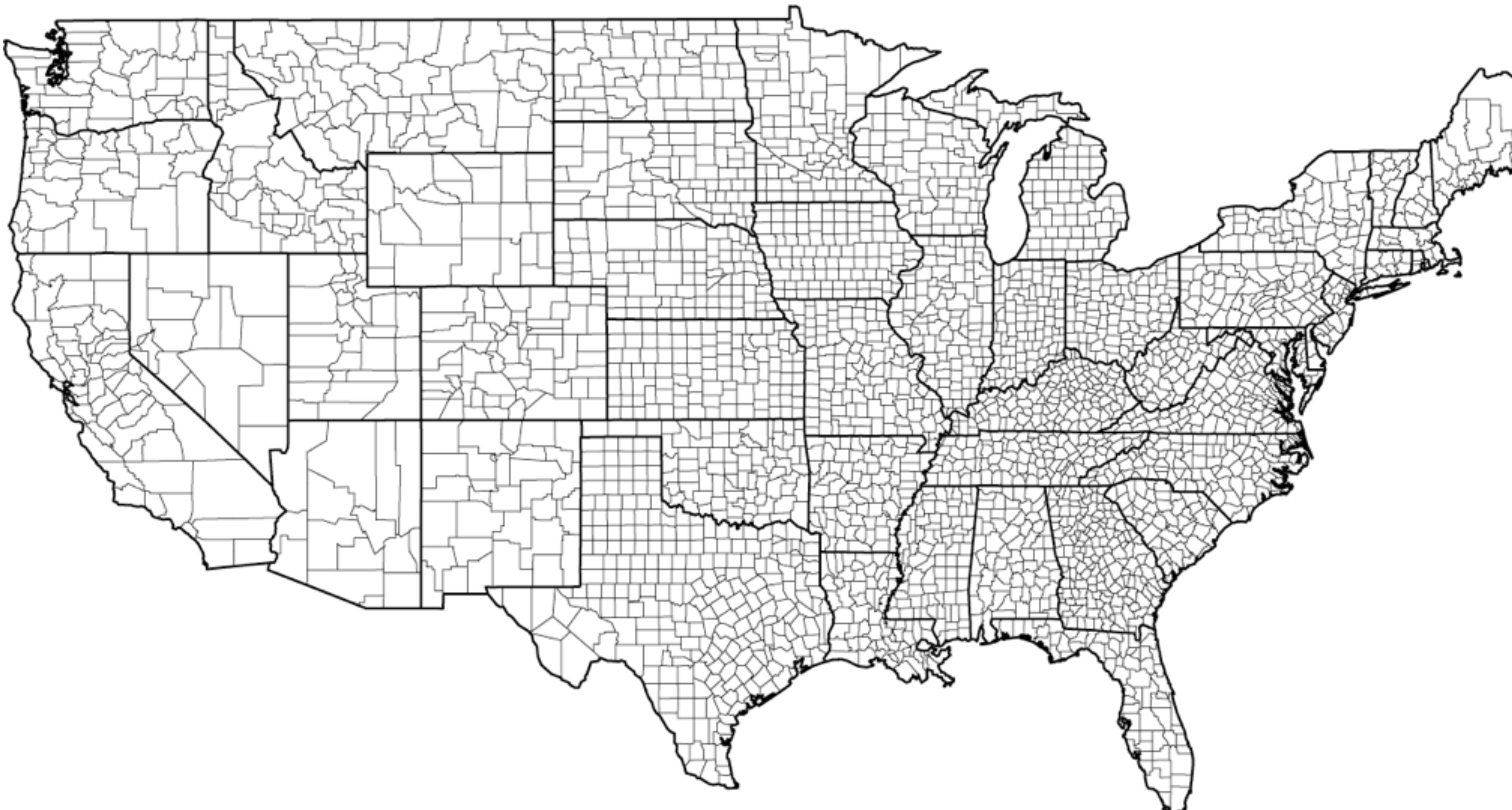
Applications

Segment data in GIS: river network, road networks, counties, etc



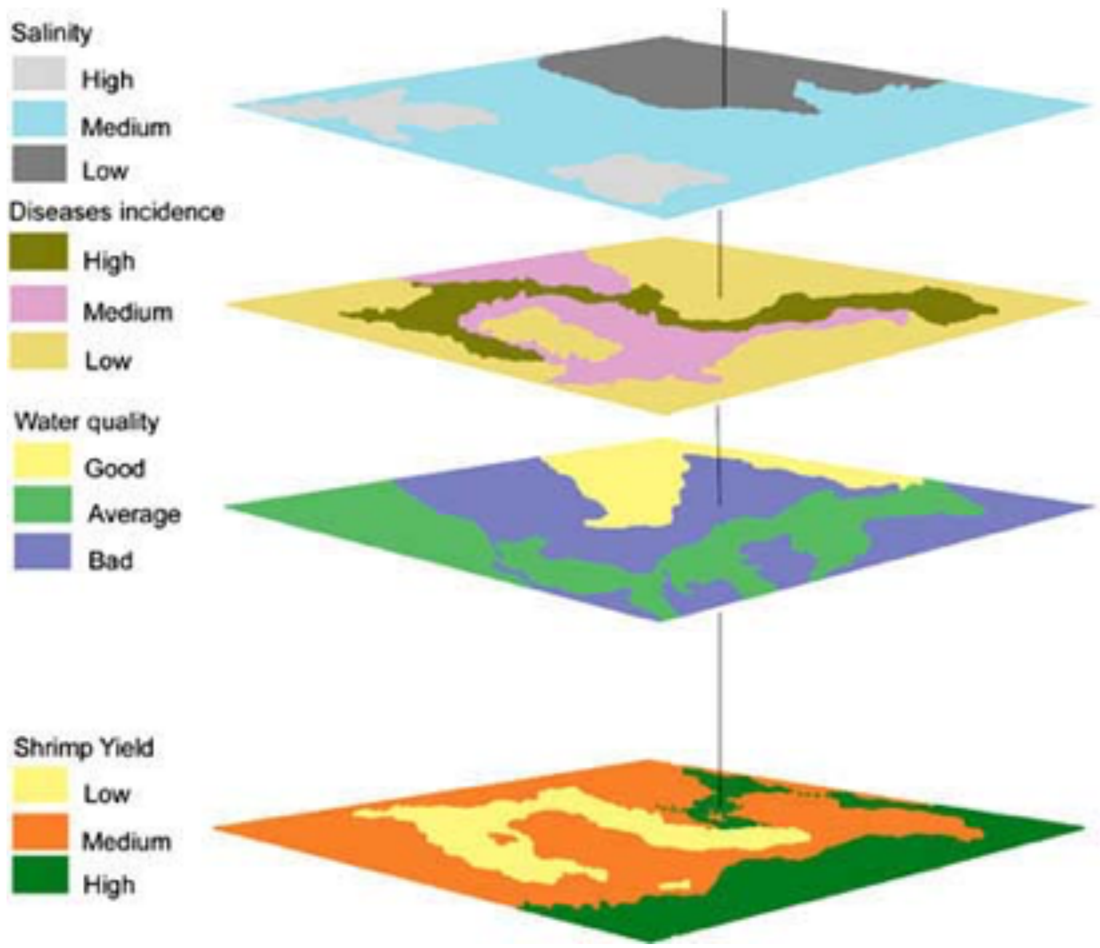
Applications

Segment data in GIS: river network, road networks, counties, etc



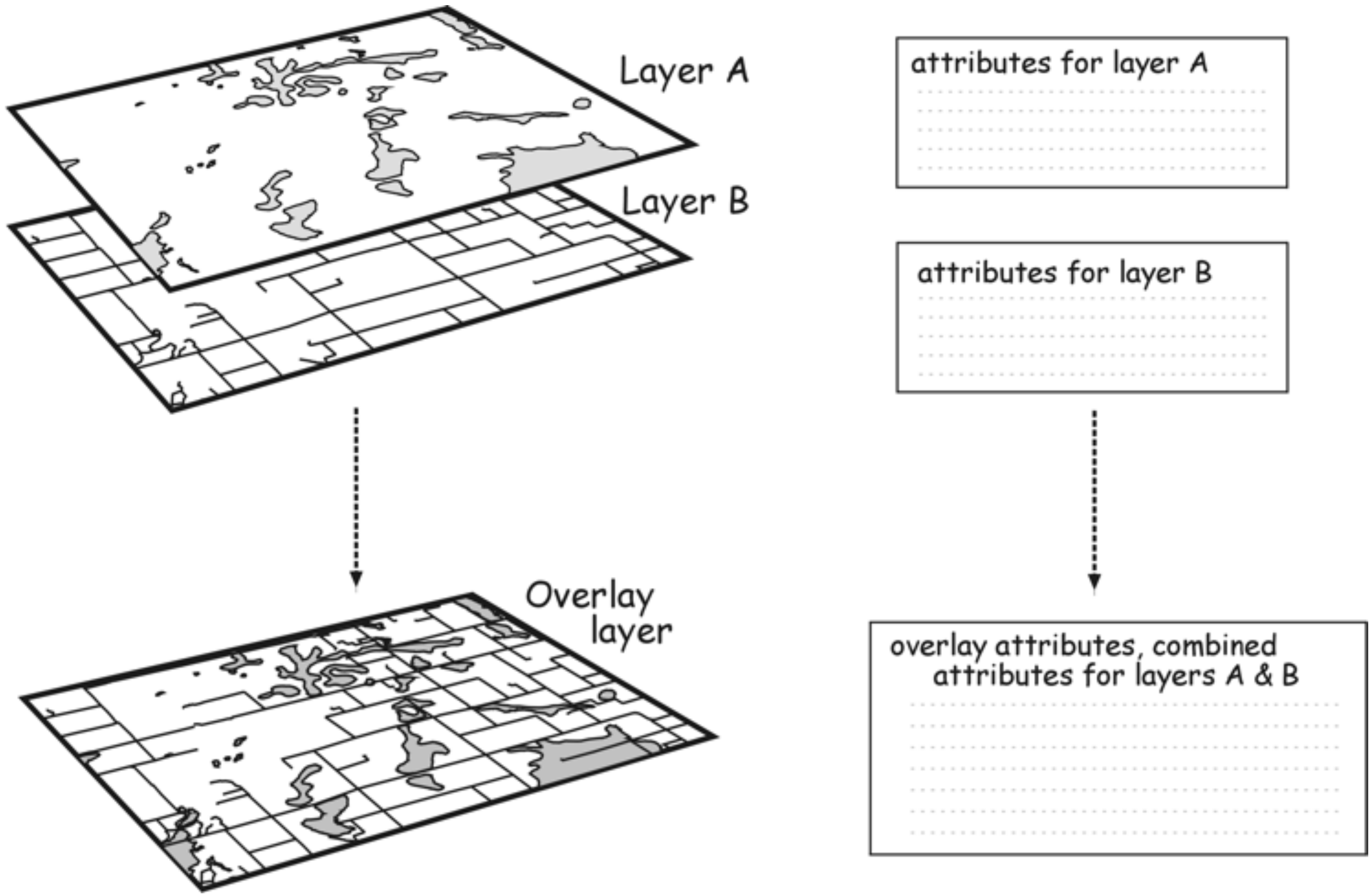
Applications

Map overlay in GIS



Applications

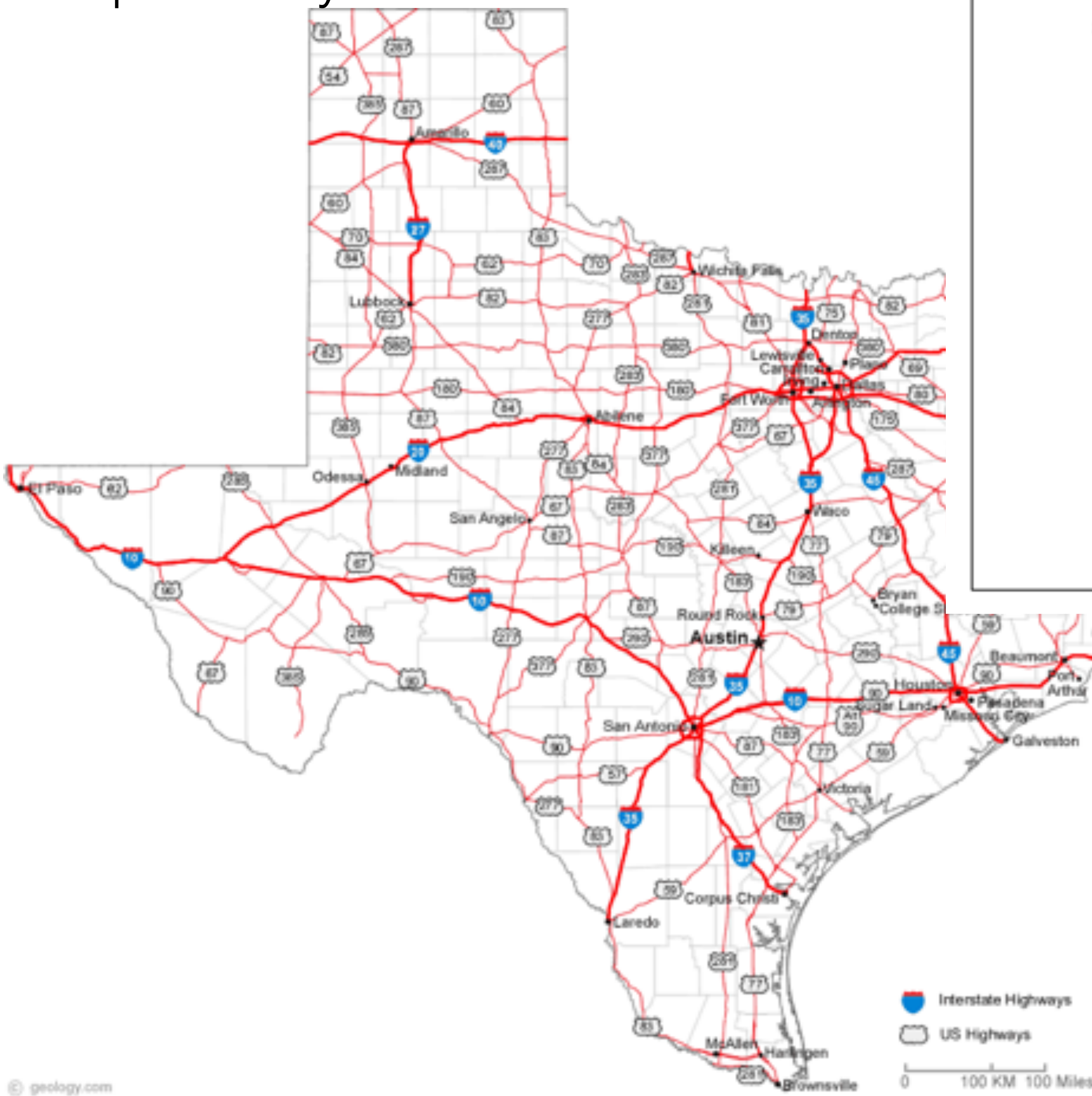
Map overlay in GIS

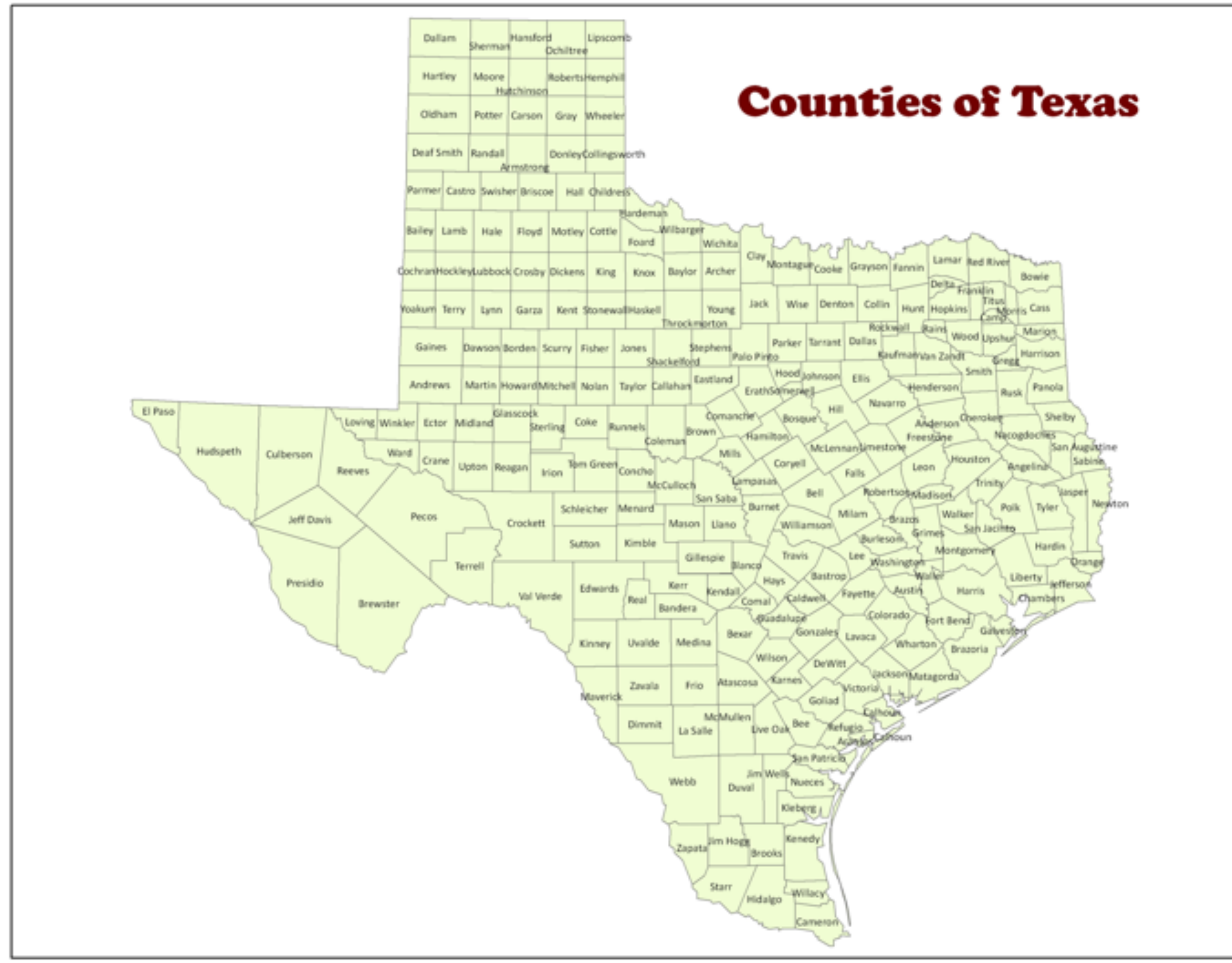


from: www.geo.hunter.cuny.edu/aierulli/gis2/lectures/Lecture2/fig9-30_raster_overlay.gif

Applications

Map overlay in GIS

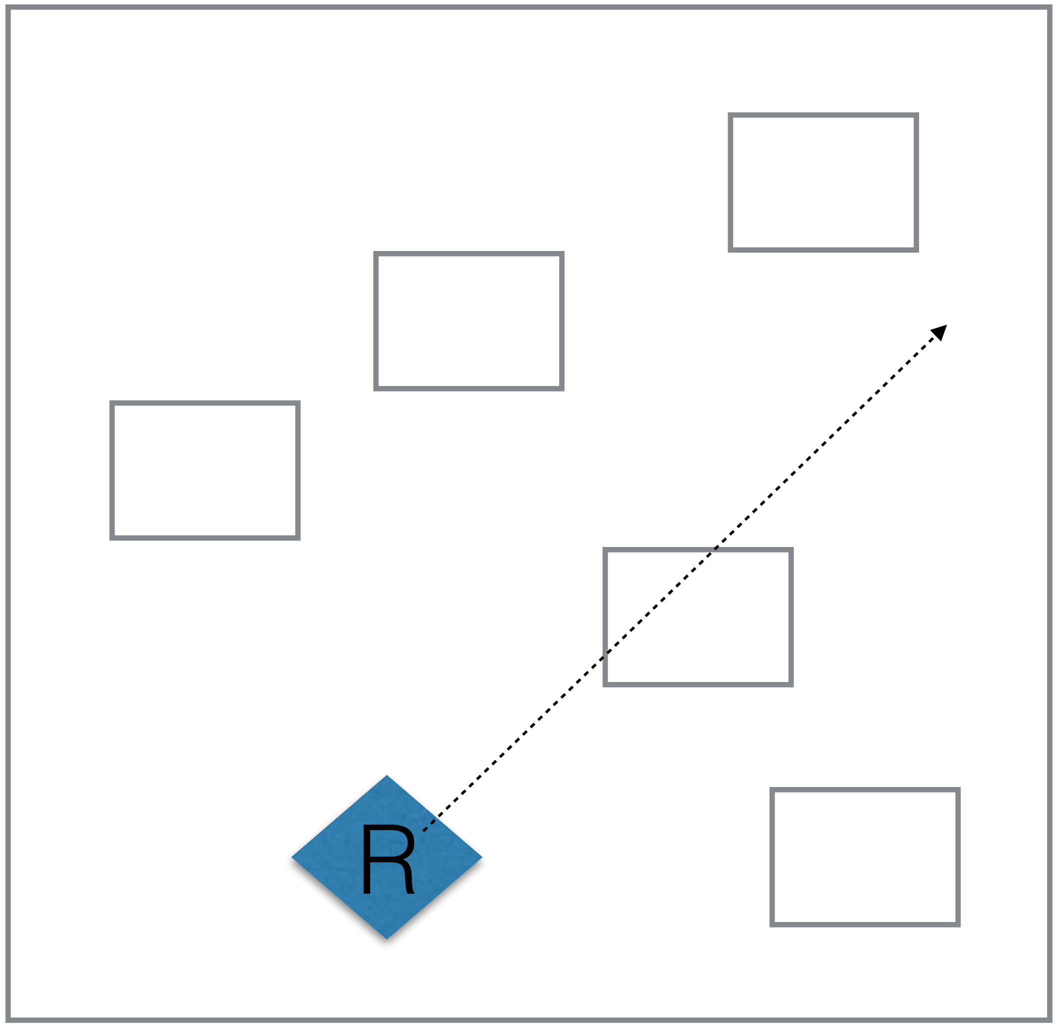




from: www.geo.hunter.cuny.edu/aierulli/gis2/lectures/Lecture2/fig9-30_raster_overlay.gif

Applications

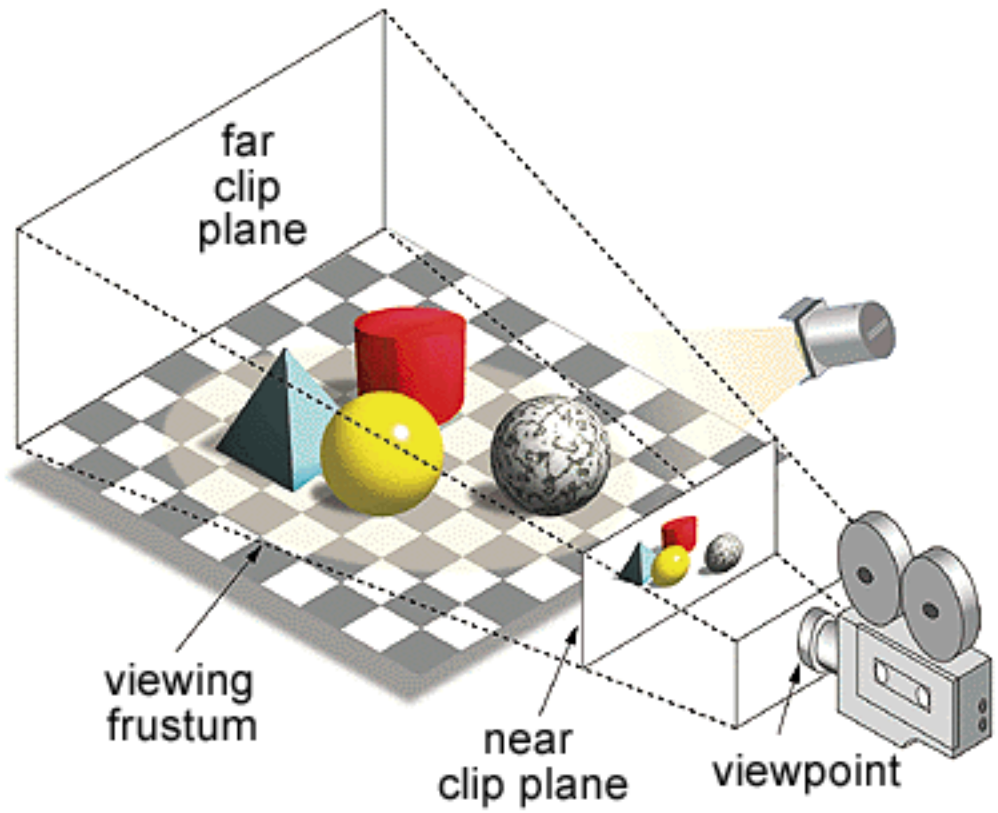
Motion planning and collision detection in robotics

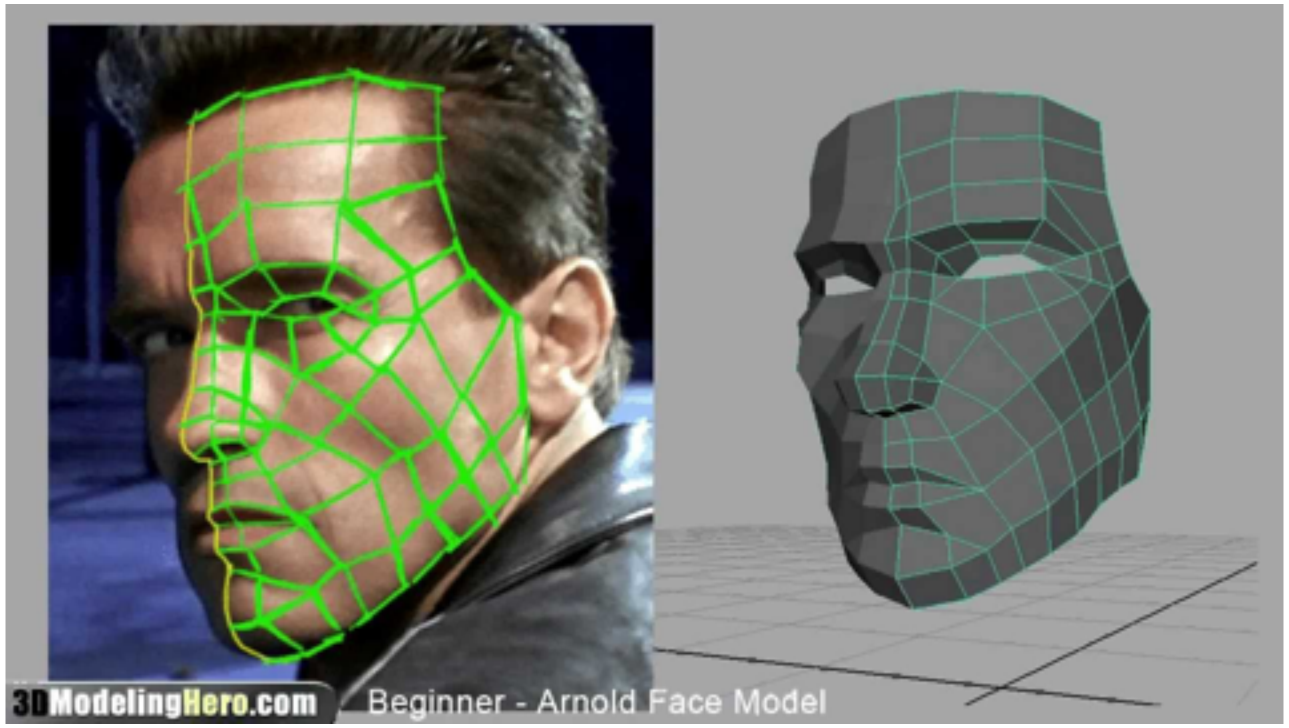


Applications

Rendering in graphics

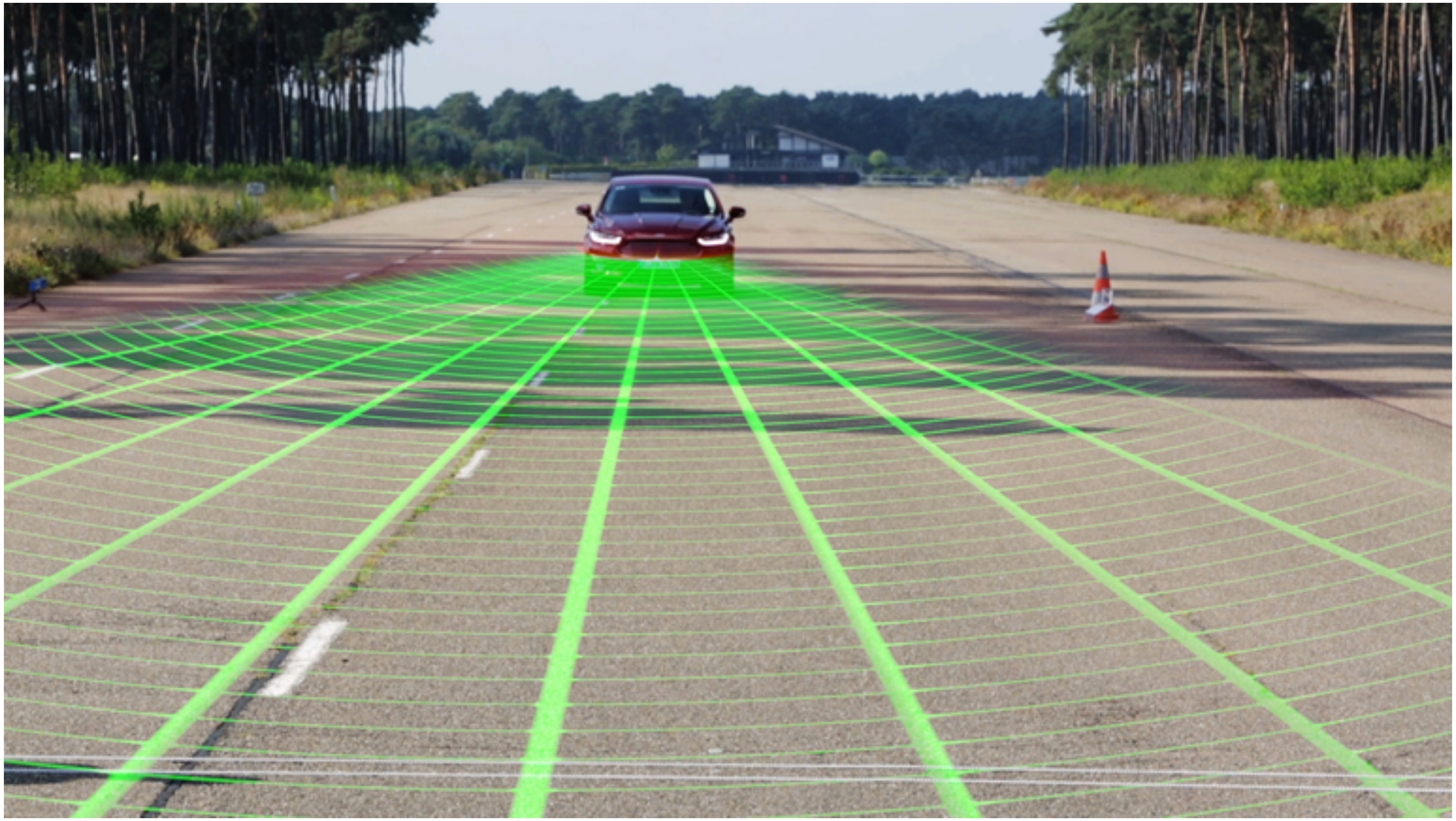
- involves intersections with objects





Applications

Collision detection



Line segment intersection

Given a set of line segments in 2D, find (report) all their pairwise intersections.

- Notation
  - n: size of the input (number of segments)
  - k: size of output (number of intersections)

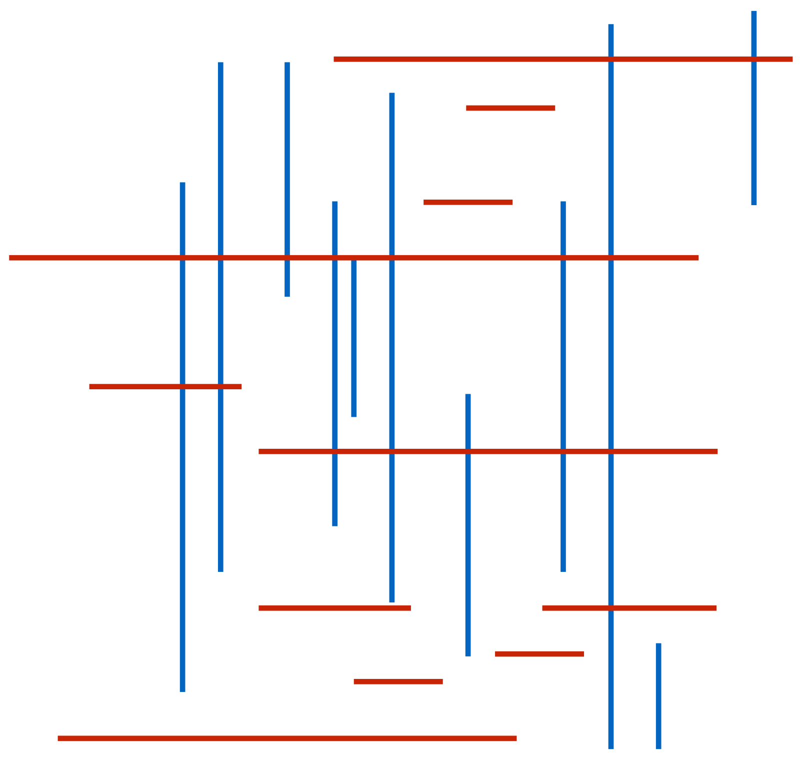
- Exercise 1:
  - Give upper and lower bounds for k.
  - Draw examples that achieve these bounds.
- Exercise 2:
  - Give a straightforward algorithm that computes all intersections and analyze its running time. Give scenarios when this algorithm is efficient/inefficient.
  - What is your intuition of an upper bound for this problem? (that is, how fast would you hope to be able to solve it?)

Line segment intersection

First we are going to look at a special case…

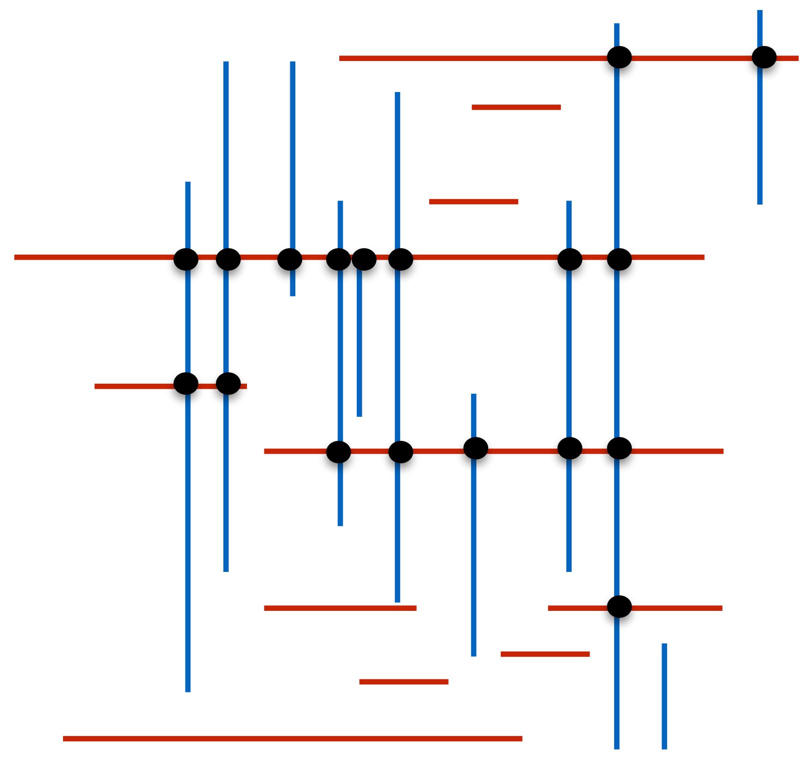
Orthogonal line segment intersection:

- Given a set of n orthogonal line segments, find all their pairwise intersections



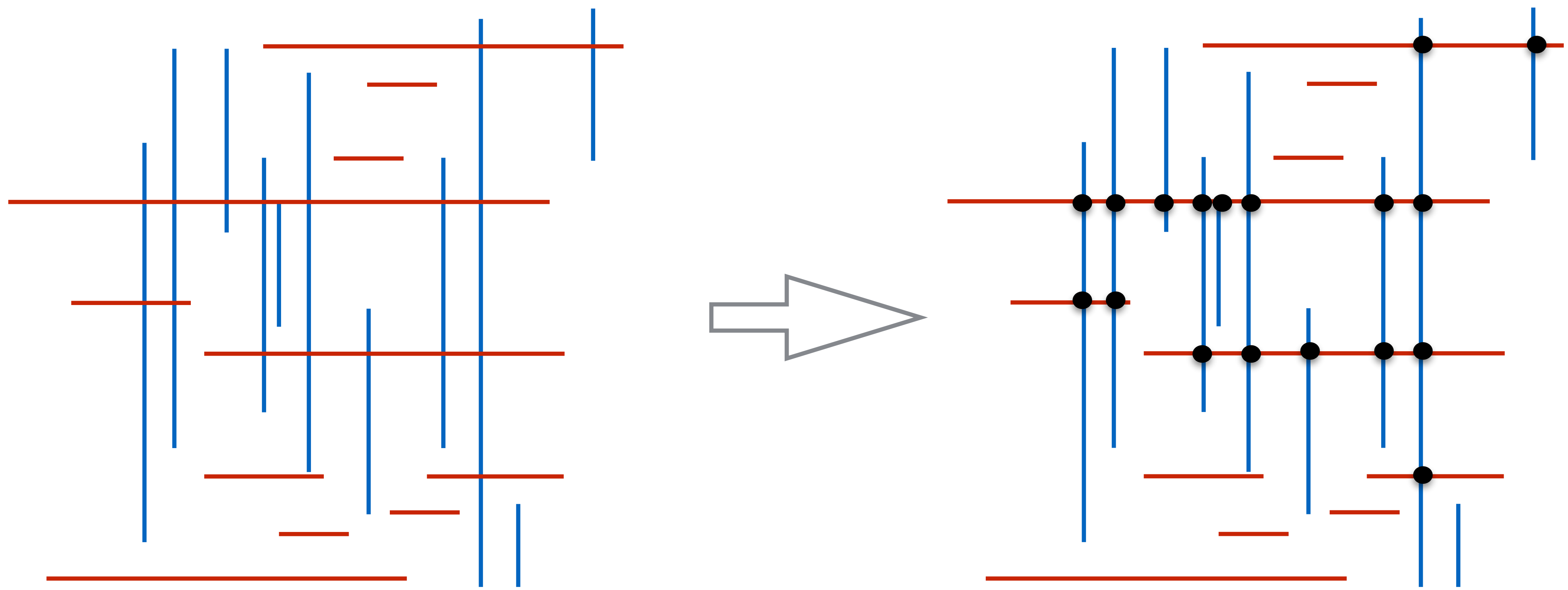
Orthogonal line segment intersection:

- Given a set of n orthogonal line segments, find all their pairwise intersections



Orthogonal line segment intersection:

- Given a set of n orthogonal line segments, find all their pairwise intersections

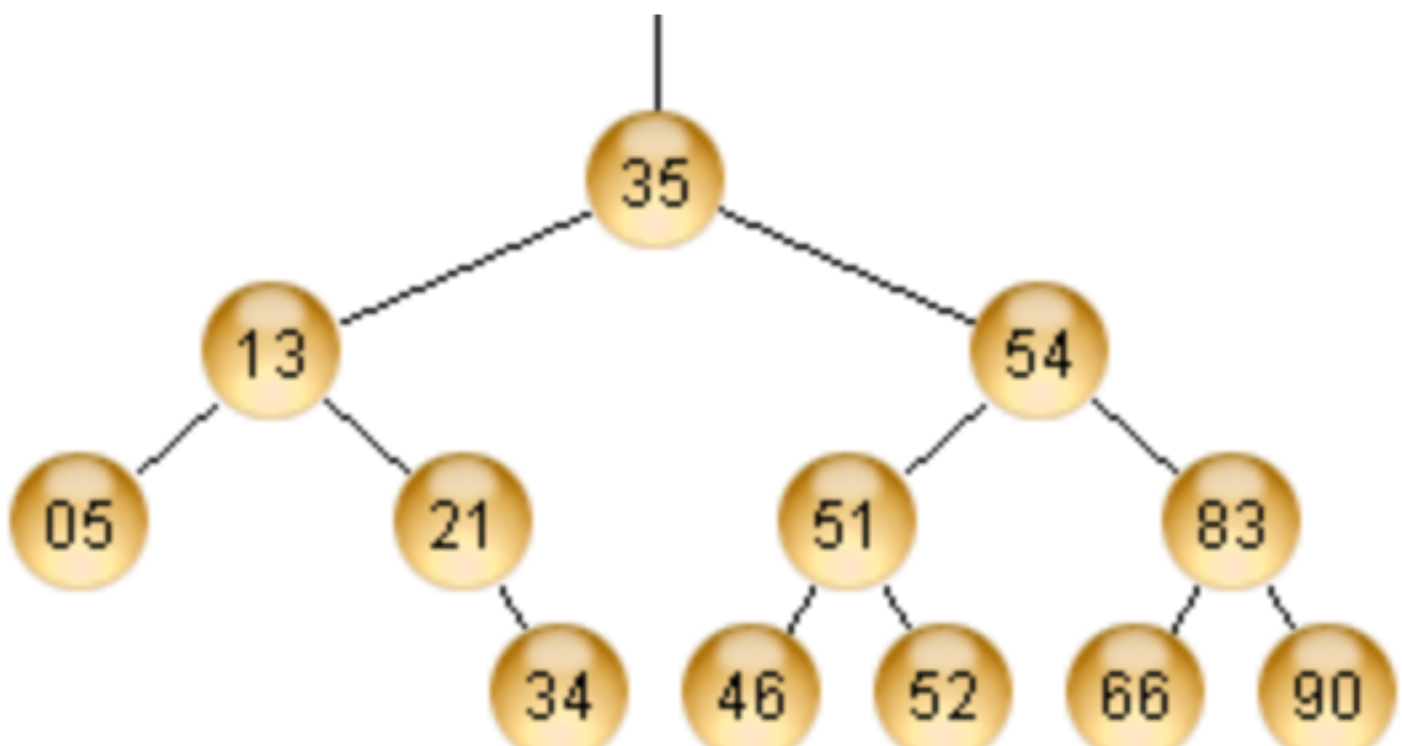


- Exercises
  - Come up with a straightforward algorithm and analyze its time
  - Can you come up with an improved algorithm?
    - Hint: use a BBST

Binary Search Trees review

Binary Search Trees (BST)

- Operations
  - insert
  - delete
  - search
  - successor, predecessor
  - traversals (in order, ..)
  - min, max



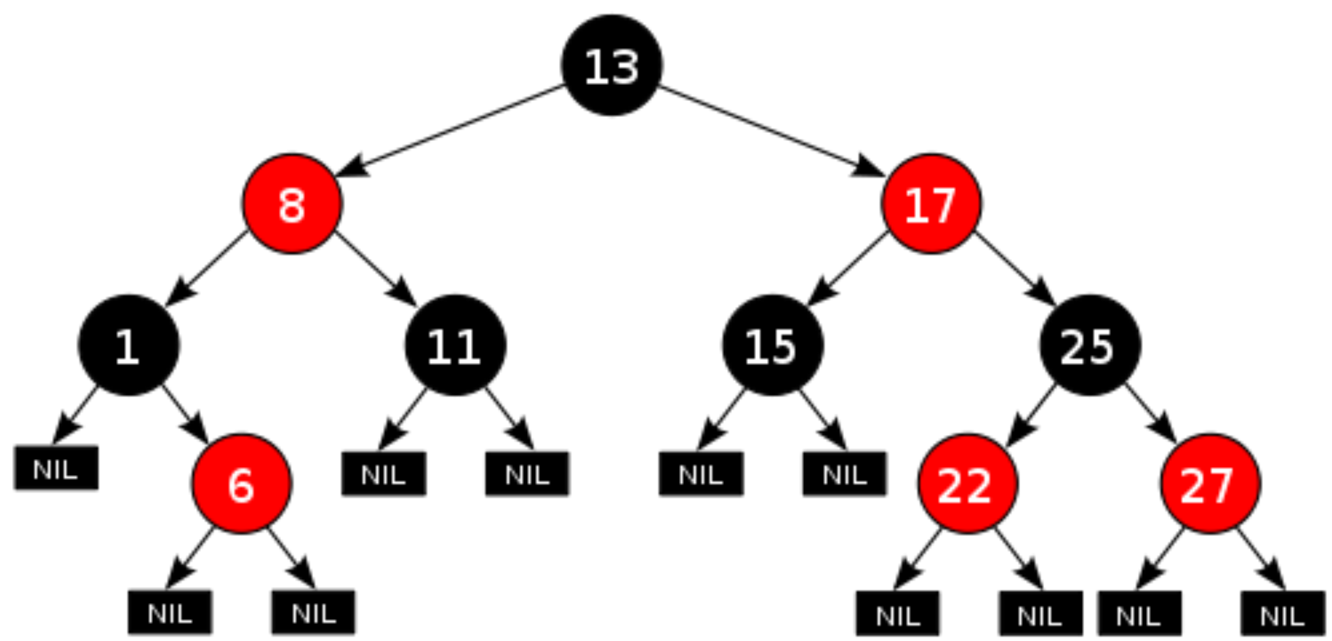
Balanced Binary Search Trees (BBST)

- Binary search trees + invariants that constrain the tree to be balanced (and thus have logarithmic height)
- These invariants have to be maintained when inserting and deleting (so we can think of the tree as self-balancing)

- BBST variants
  - red-black trees
  - AVL trees
  - B-trees
  - (a,b) trees
  - …

Example: Red-Black trees

- Binary search trees such that
  - Each node is Red or Black
  - The children of a Red node must be Black
  - The number of Black nodes on any path from the root to a node that does not have two children must be the same

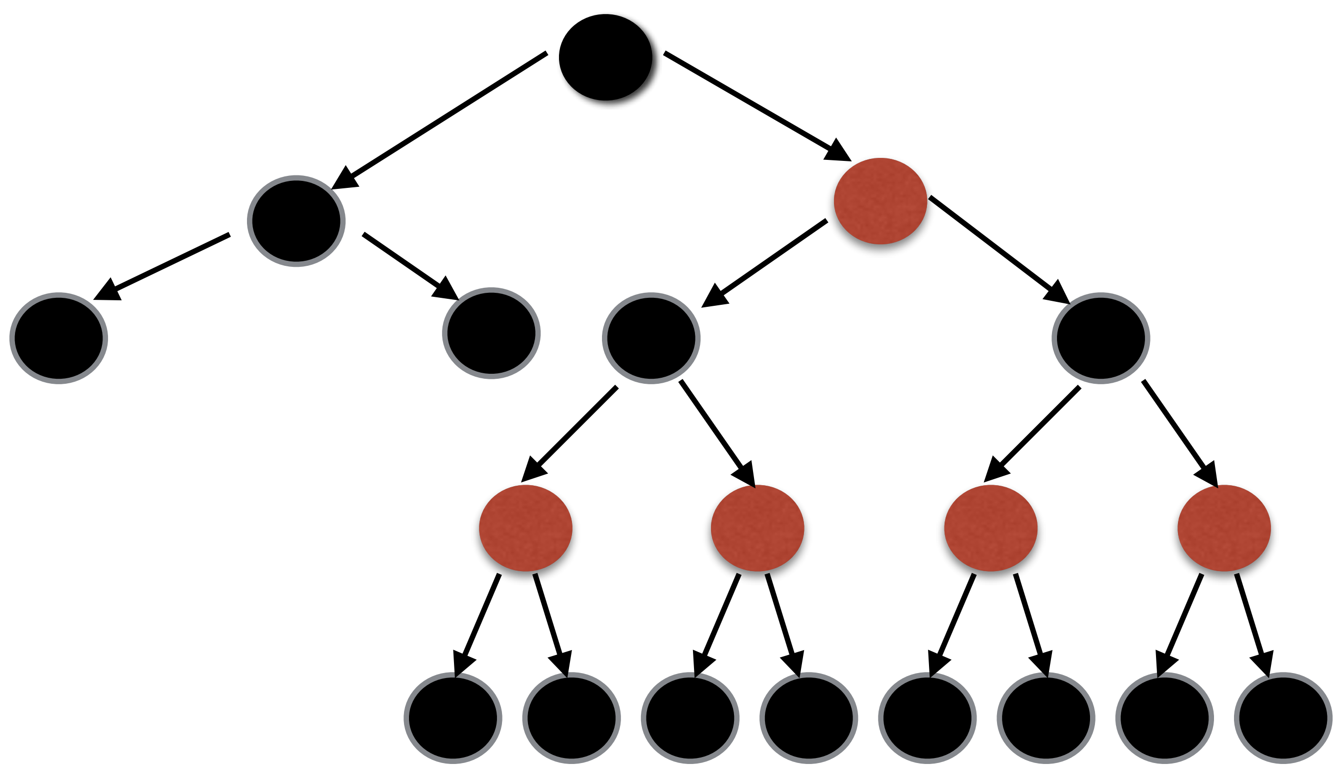


Note:

- easier to conceptualize the tree as containing explicit NULL leaves, all Black
- the number of Black nodes on any root-to-leaf path must be the same

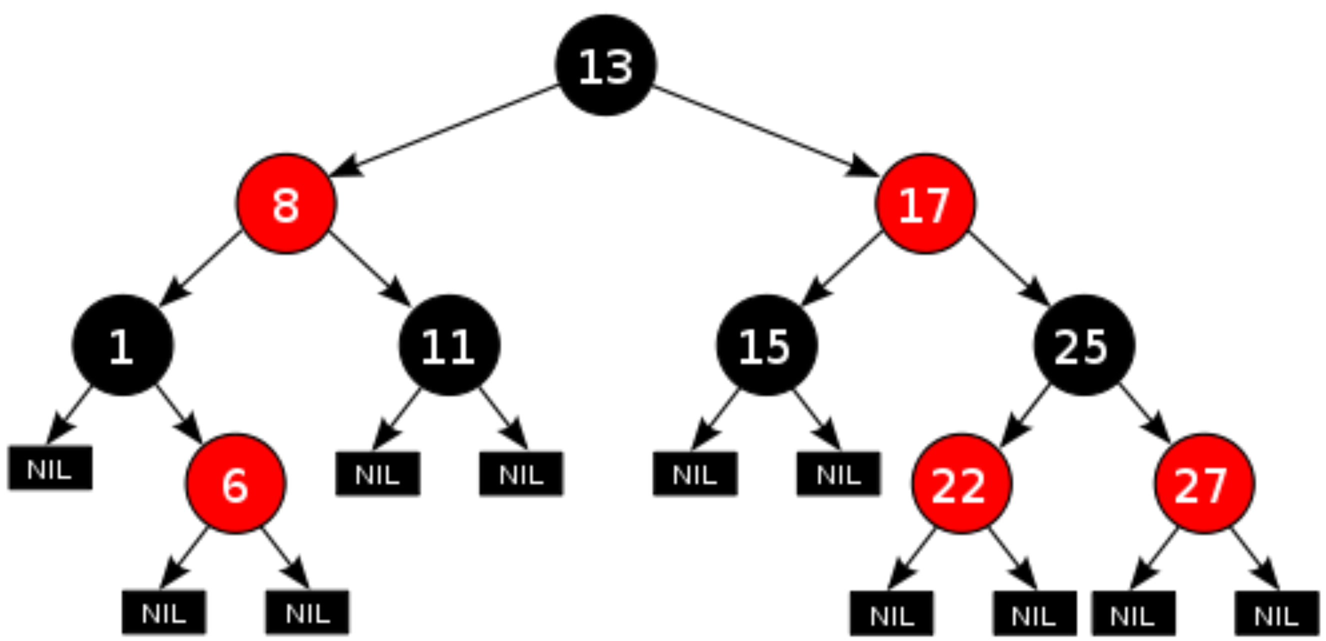
Example: Red-Black trees

- Theorem:
  - A Red-Black tree of n nodes has height Theta(lg n).



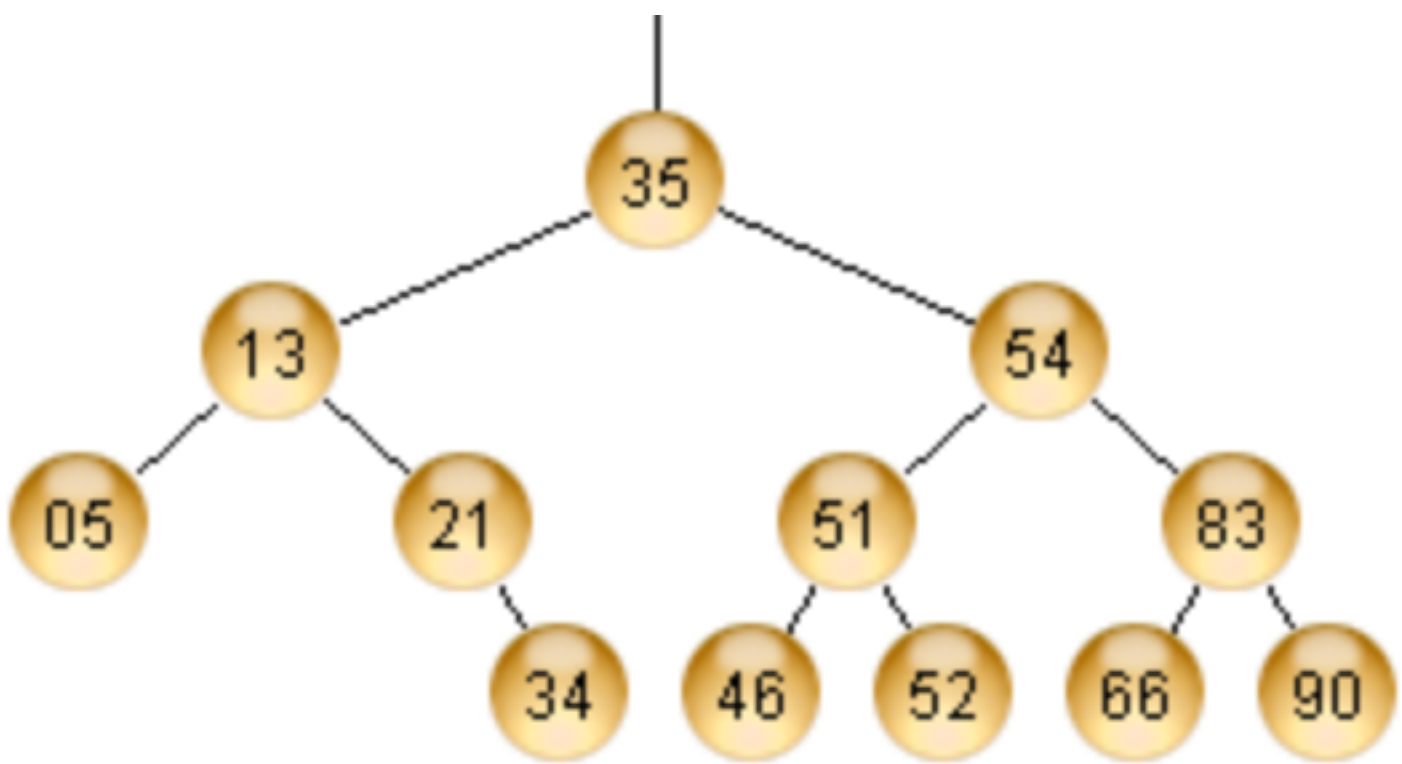
Example: Red-Black trees

- Theorem:
  - After an insertion or a deletion, the RB tree invariants can be maintained in additional O(lg n) time. This is done by performing rotations and recoloring on the path from the inserted/deleted node up to the root.



Binary Search Trees

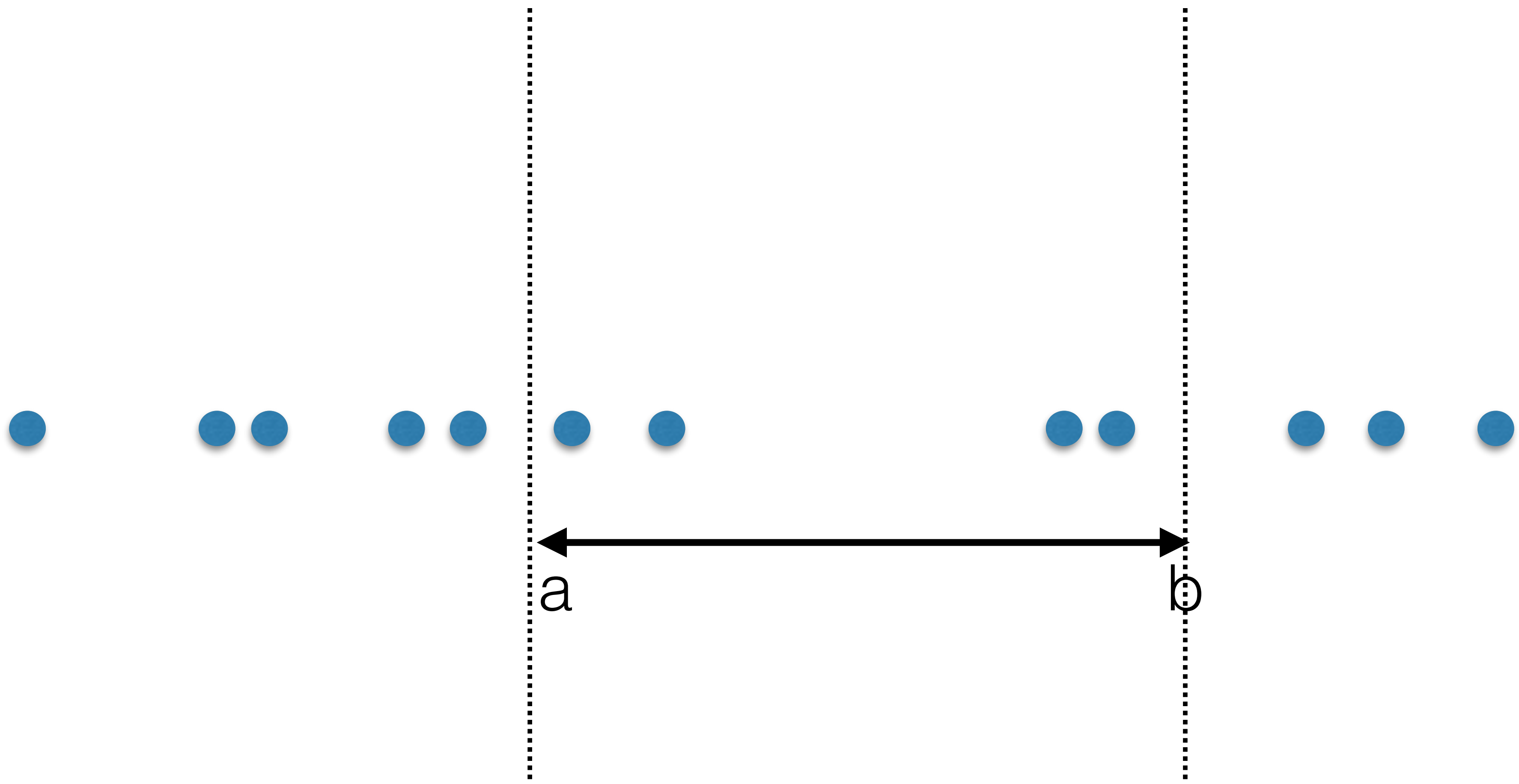
- Operations
  - insert
  - delete
  - search
  - successor, predecessor
  - traversals (in order, ..)
  - min, max
  - range search (1D)



1D Range Searching

- Given a set of values P = {x_1, x_2, x3, …x_n }
- Want to answer Range Search queries:

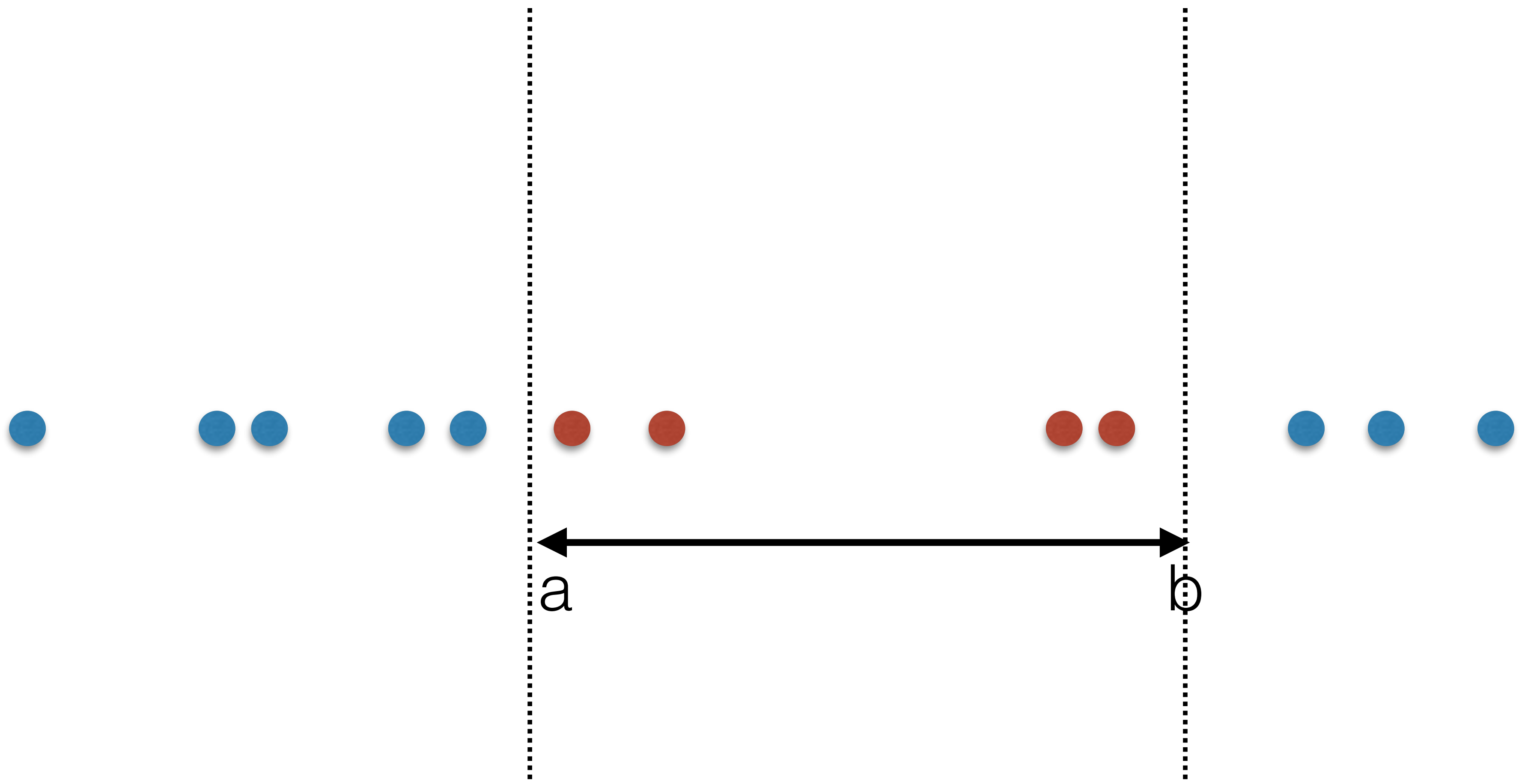
rangeSearch(a,b): return all elements in P in interval (a,b)



1D Range Searching

- Given a set of values $P = \{x_1, x_2, x3, \ldots x_n\}$
- Want to answer Range Search queries:

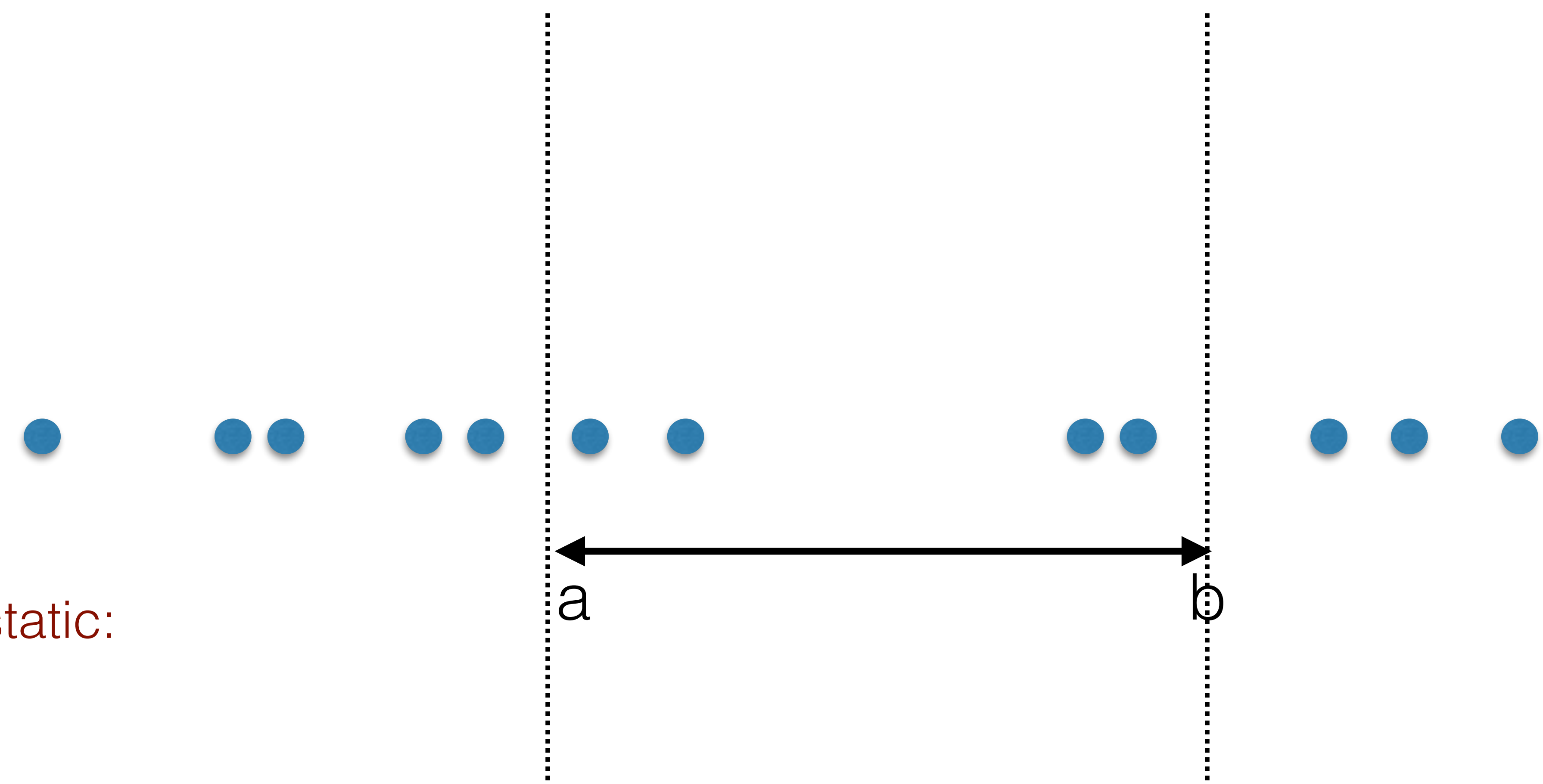
  rangeSearch(a,b): return all elements in P in interval (a,b)



1D Range Searching

- Given a set of values P = {x_1, x_2, x3, …x_n }
- Want to answer Range Search queries:

  rangeSearch(a,b): return all elements in P in interval (a,b)

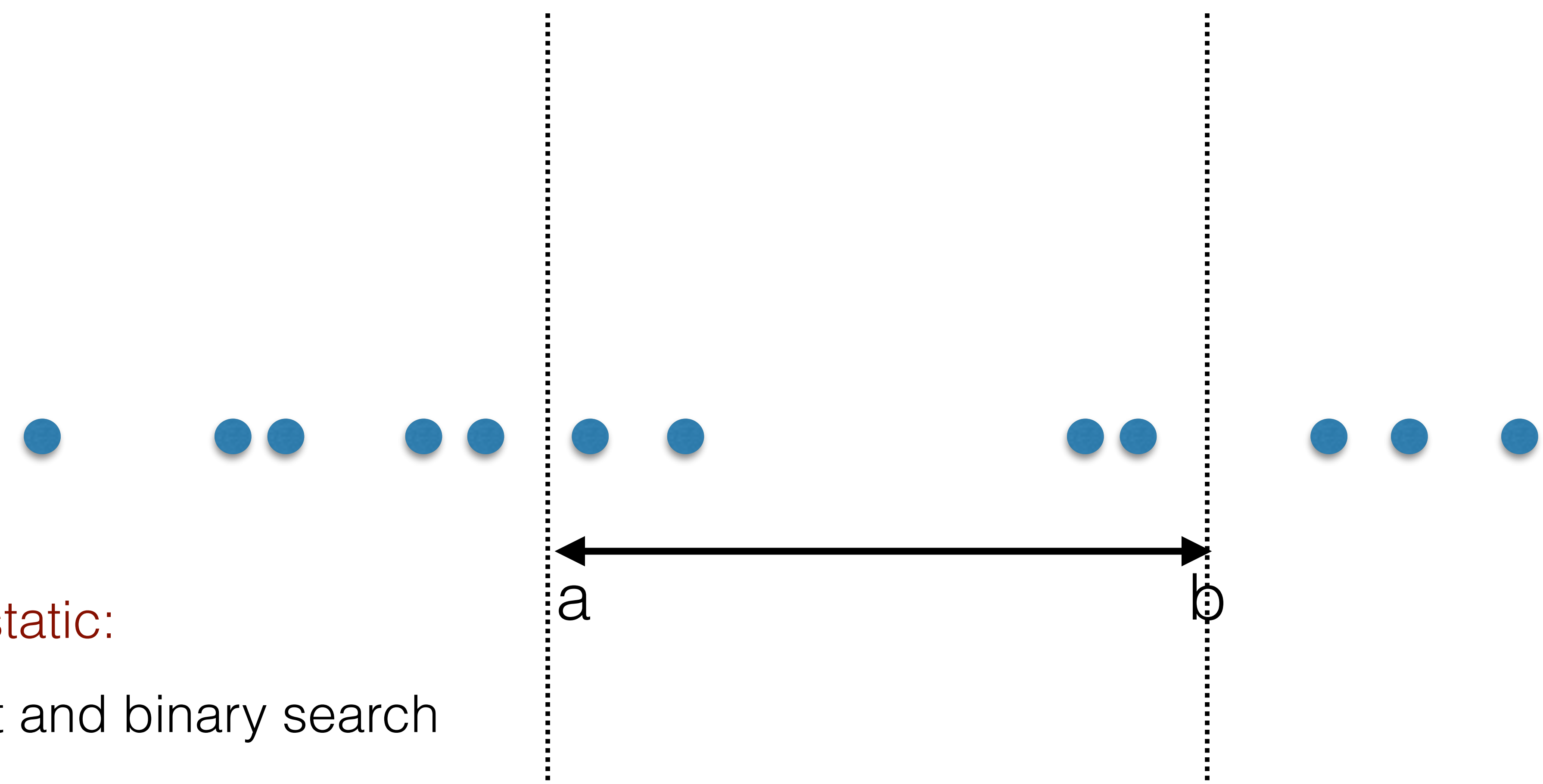


- If P is static:

1D Range Searching

- Given a set of values P = {x_1, x_2, x3, …x_n }
- Want to answer Range Search queries:

    rangeSearch(a,b): return all elements in P in interval (a,b)

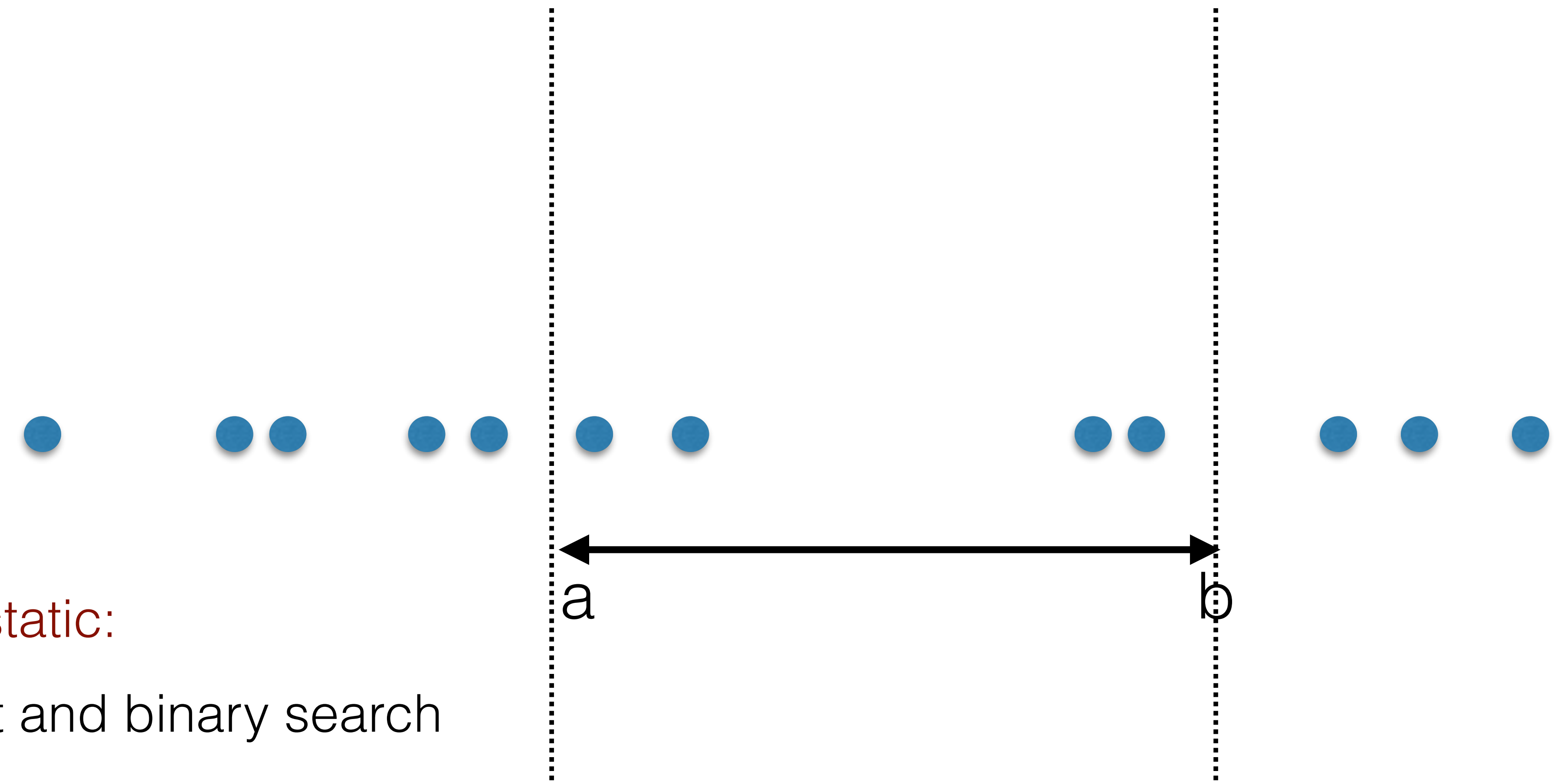


- If P is static:
    - sort and binary search

1D Range Searching

- Given a set of values $P = \{x_1, x_2, x3, \ldots x_n\}$
- Want to answer Range Search queries:

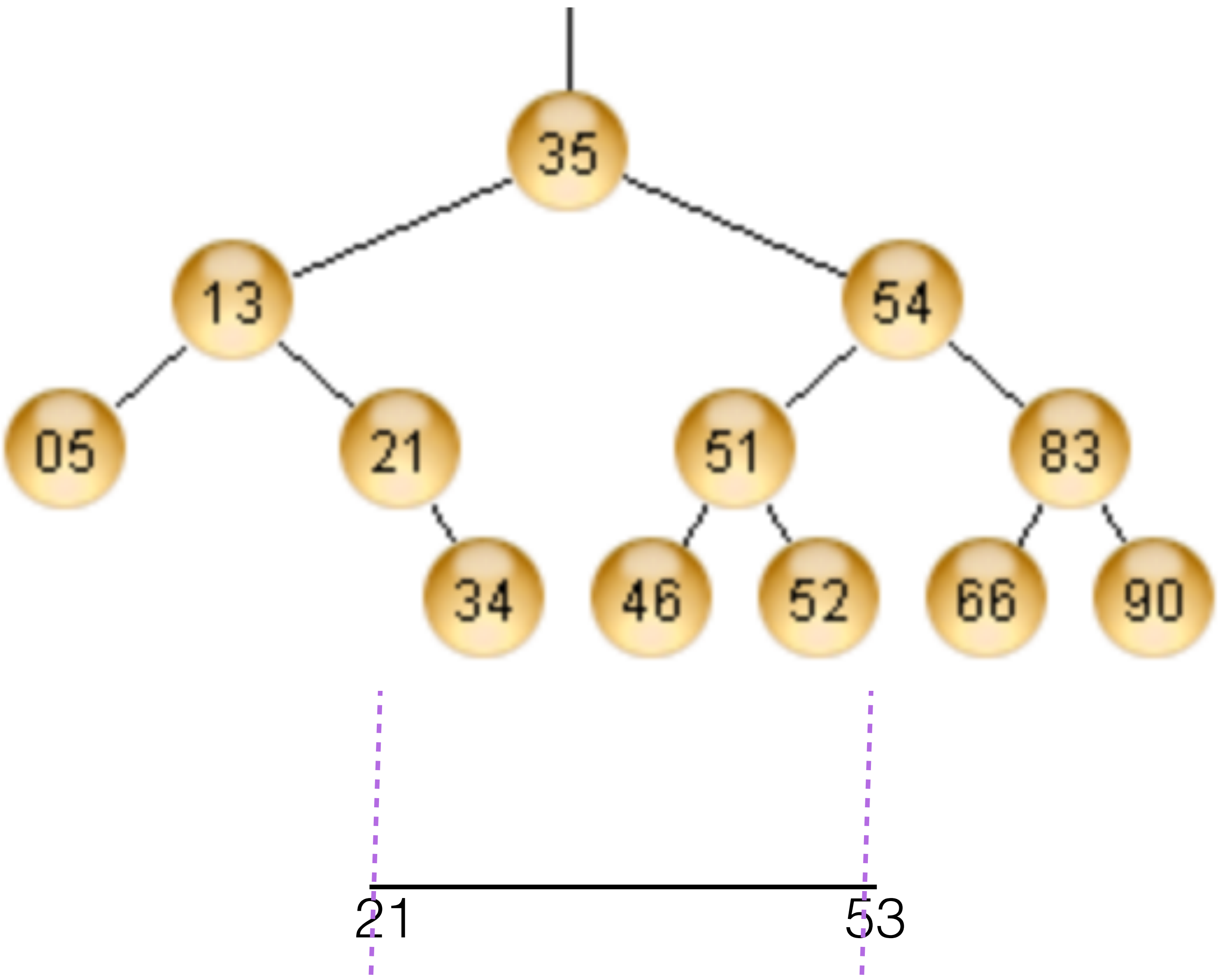
    rangeSearch(a,b): return all elements in P in interval (a,b)



- If P is static:
    - sort and binary search
- If P is dynamic:
    - use BBST

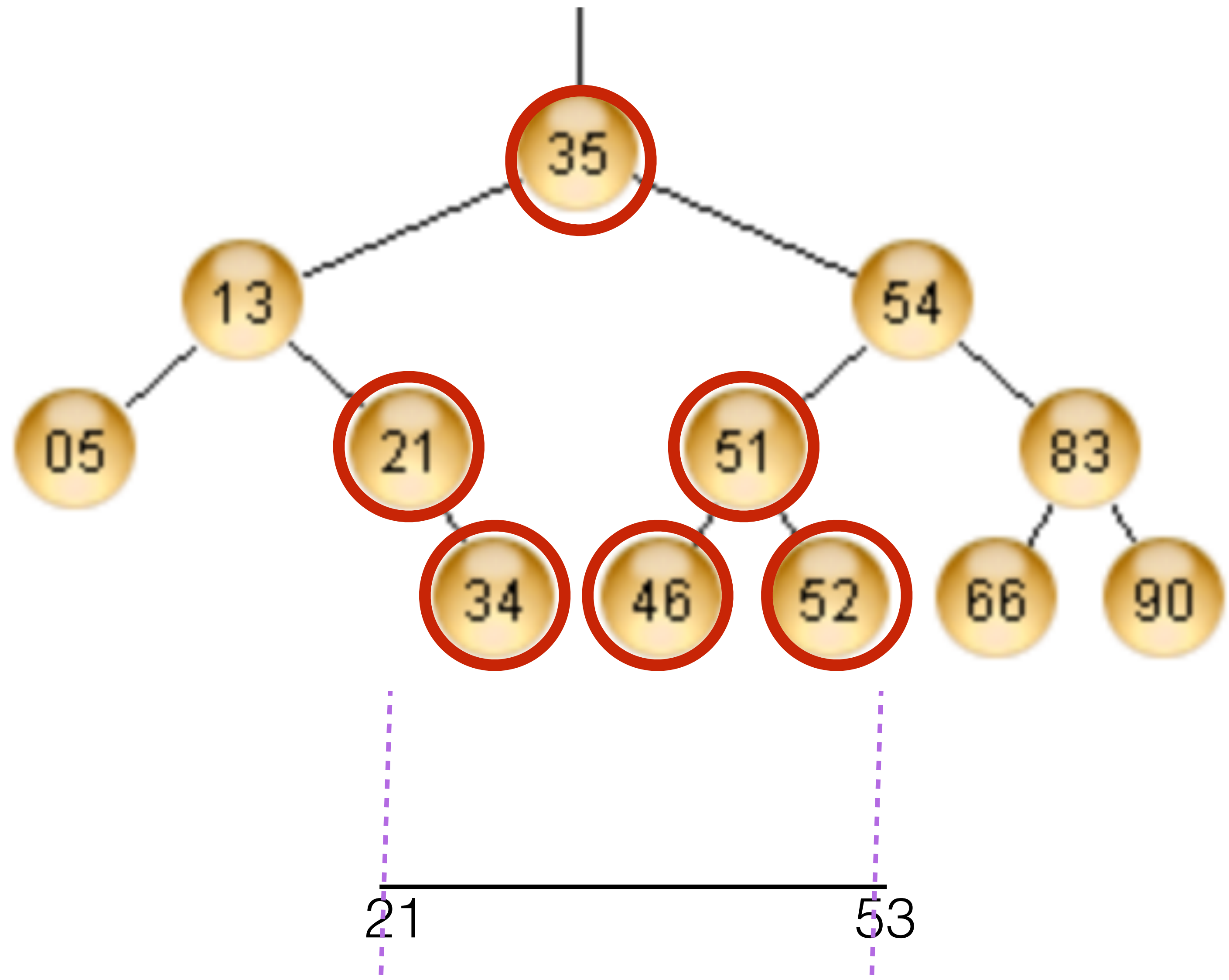
1D range searching with Binary Search Trees

Example: range_search(21, 53): return 21, 34, 35, 46, 51, 52



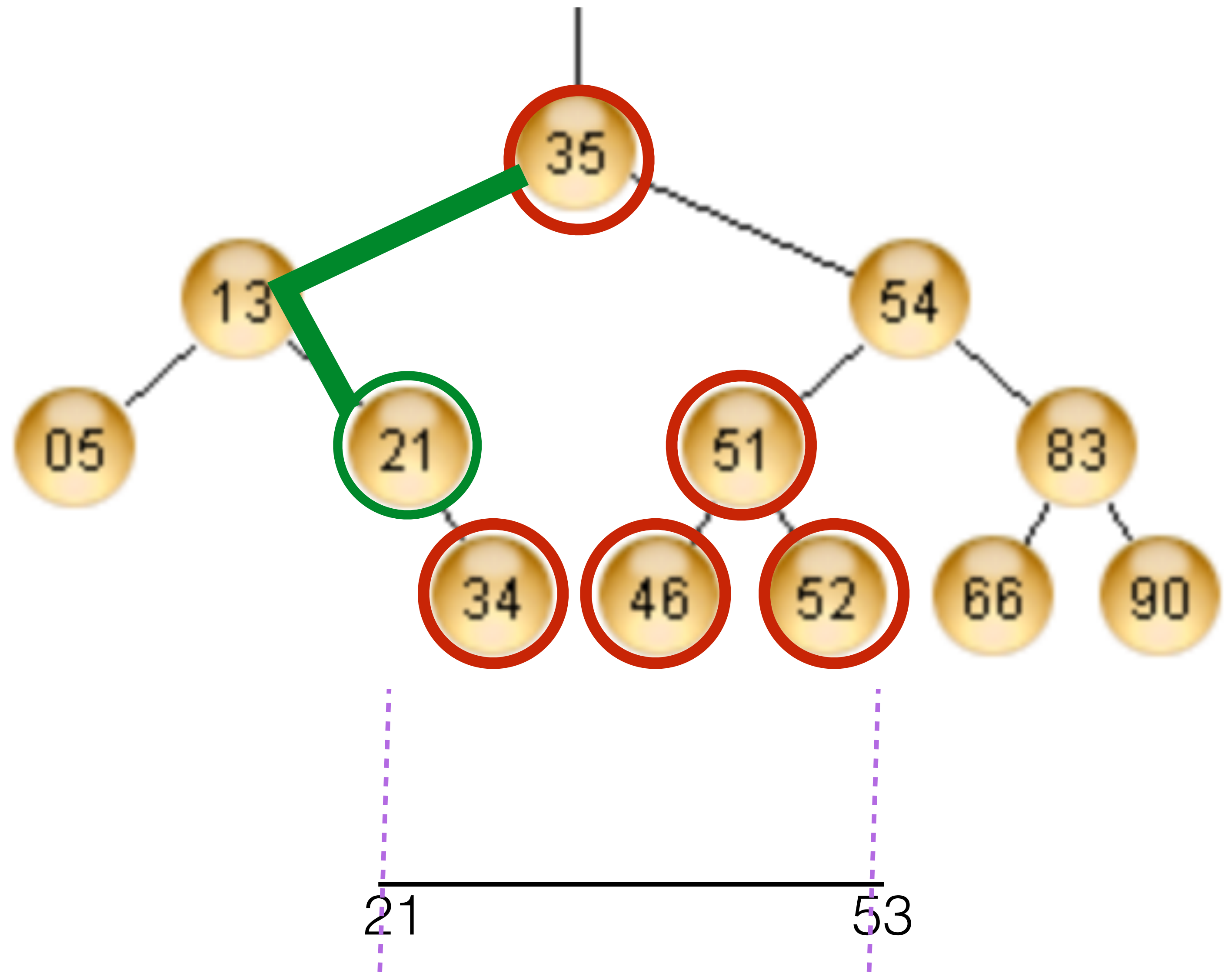
1D range searching with Binary Search Trees

Example: range_search(21, 53): return 21, 34, 35, 46, 51, 52



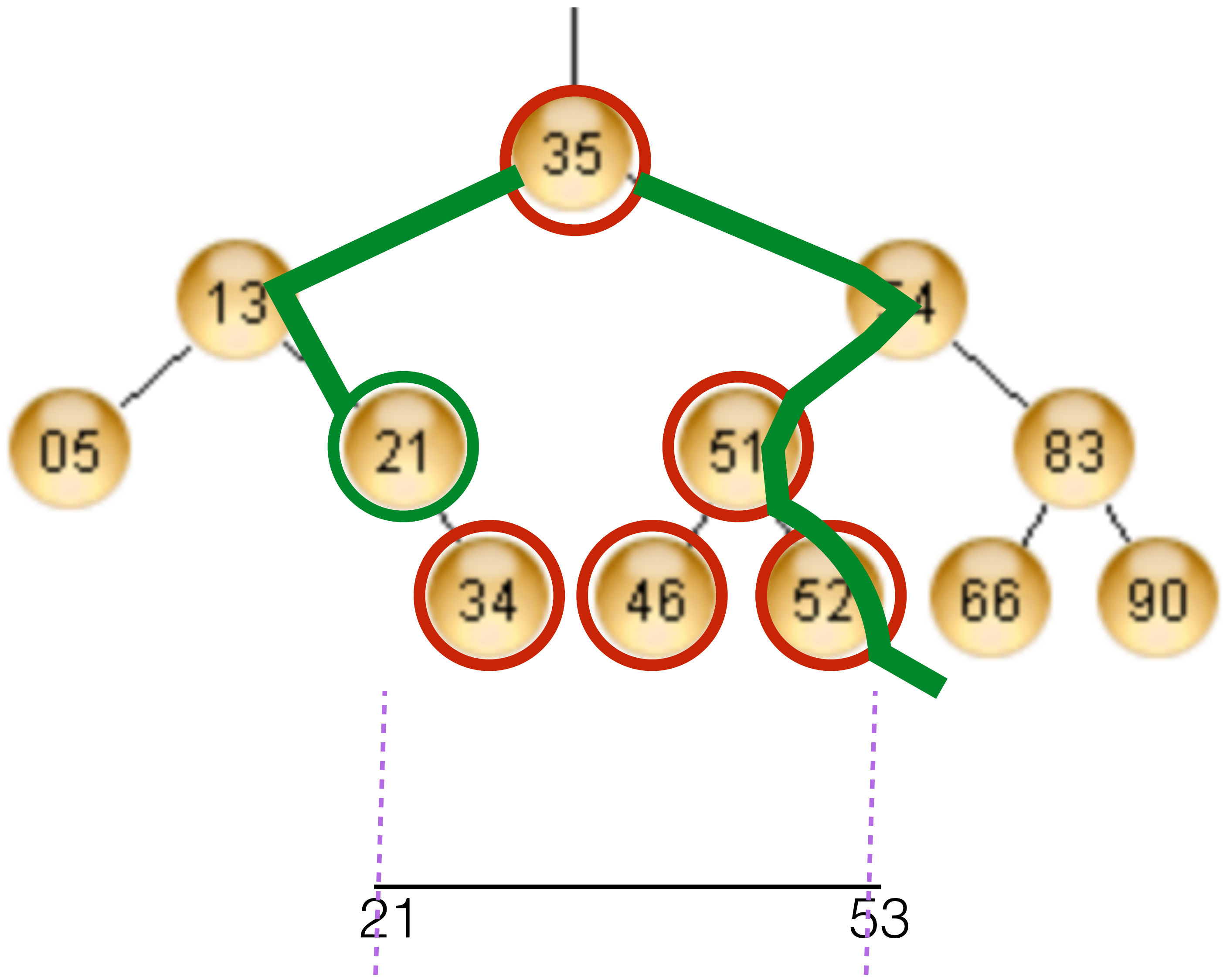
1D range searching with Binary Search Trees

Example: range_search(21, 53): return 21, 34, 35, 46, 51, 52



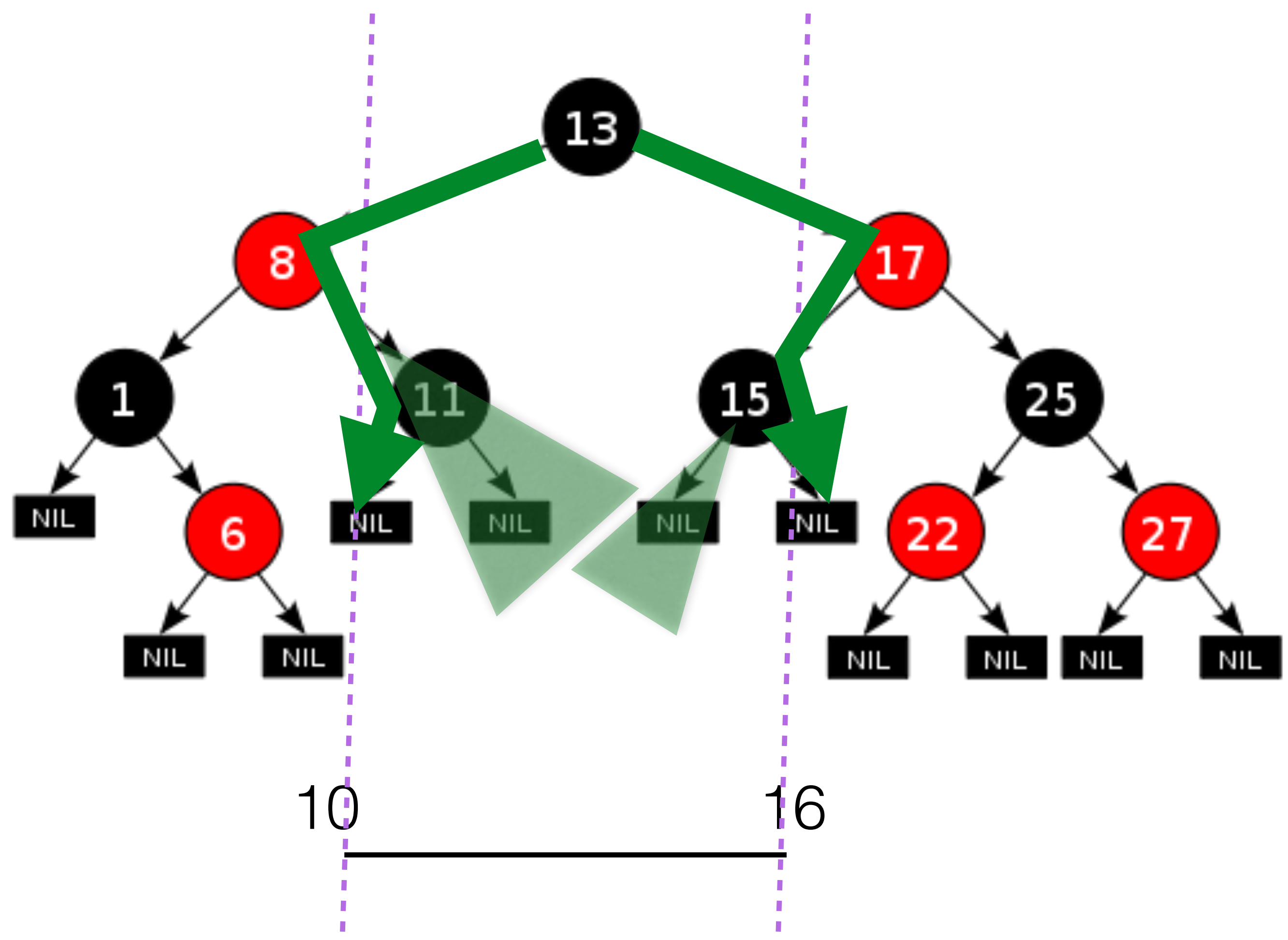
1D range searching with Binary Search Trees

Example: range_search(21, 53): return 21, 34, 35, 46, 51, 52



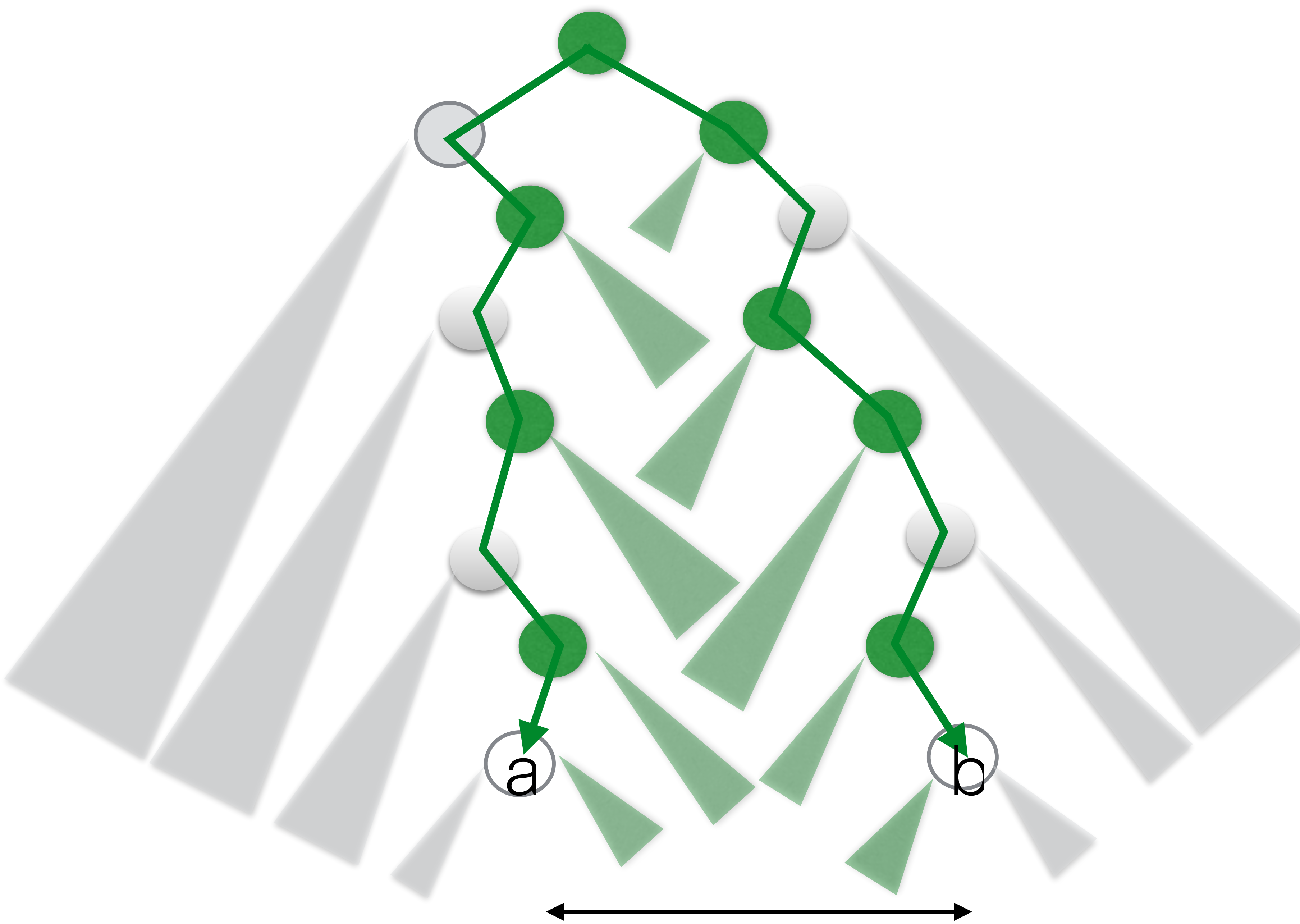
1D Range Searching with Red-Black Trees

Example: range_search(10, 16): return 11, 13, 15



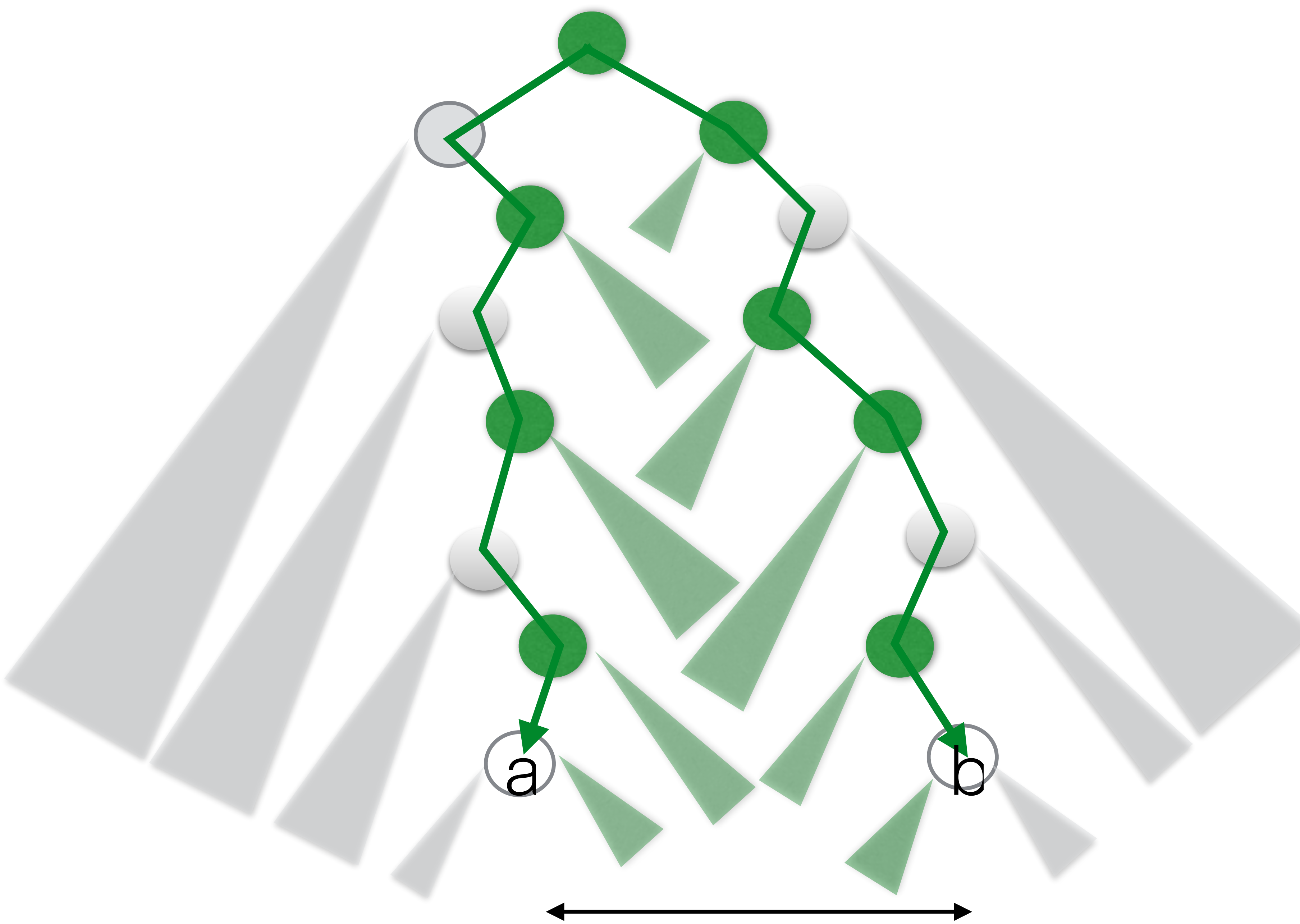
1D range searching with Binary Search Trees

- Range search (a,b): return all elements in this interval

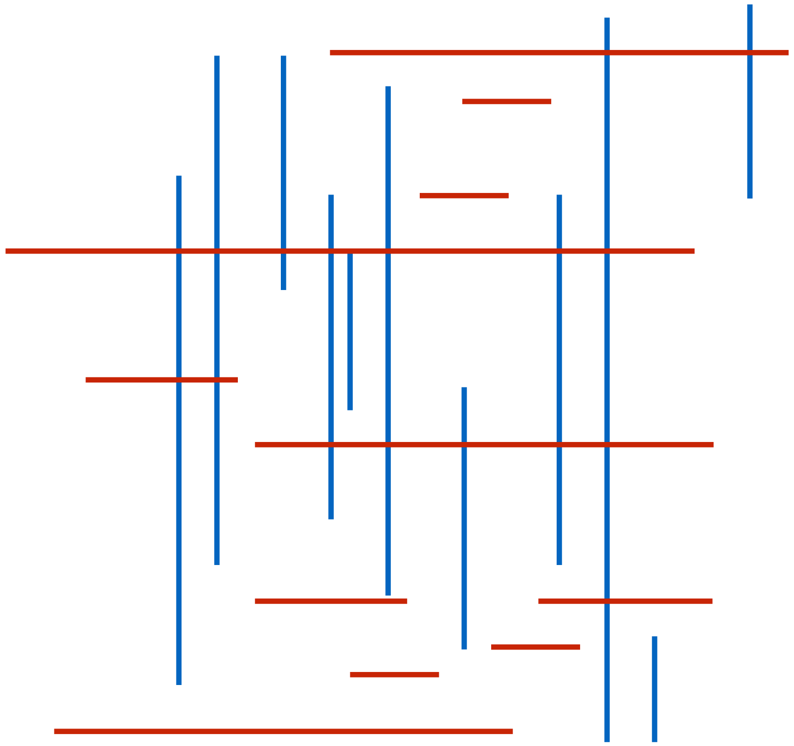


1D range searching with Binary Search Trees

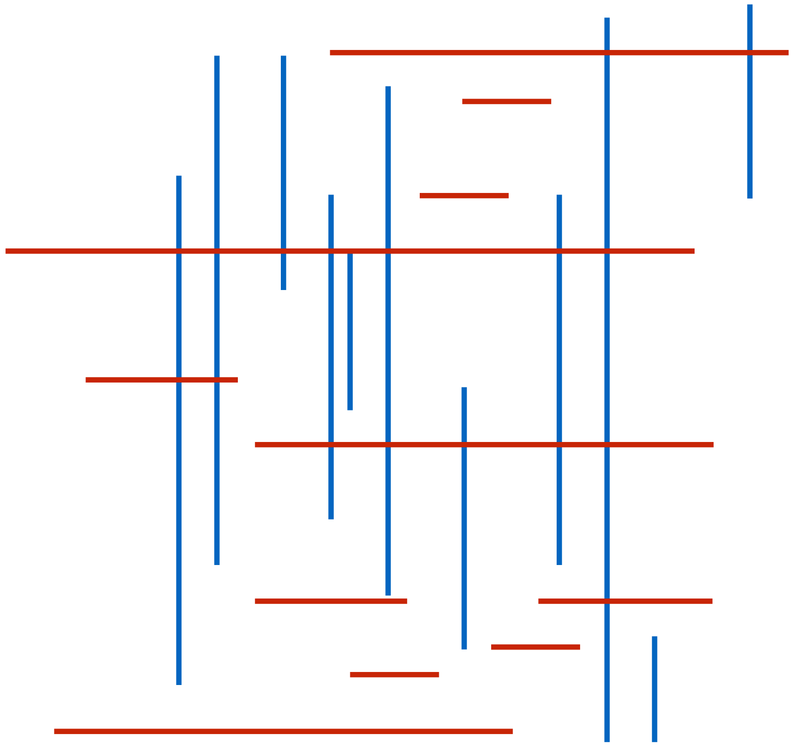
- Range search (a,b): return all elements in this interval
- Can be answered in O(lg n+k), where k = O(n) is the size of output



Orthogonal line segment intersection using BST

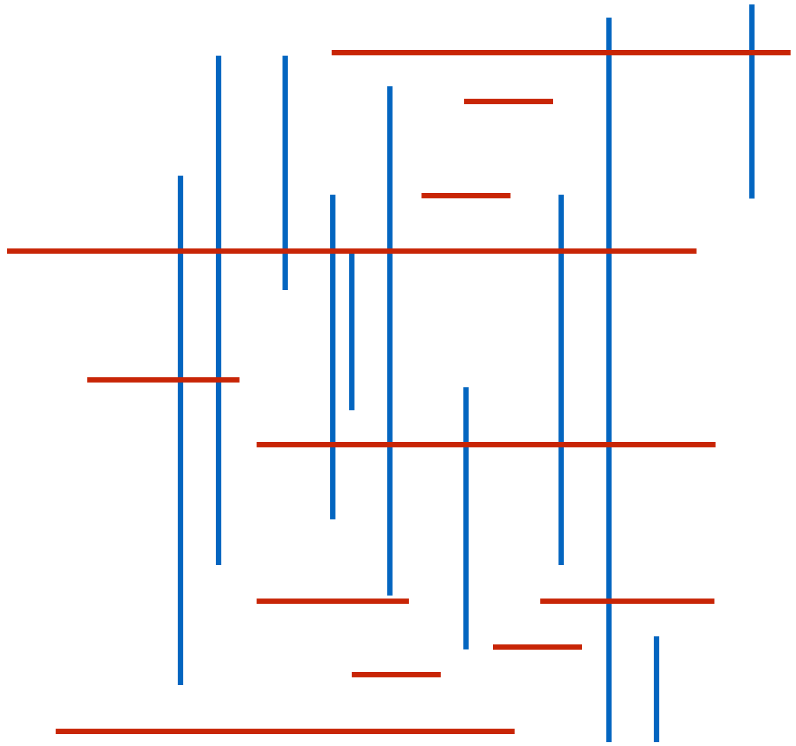


Orthogonal line segment intersection using BST



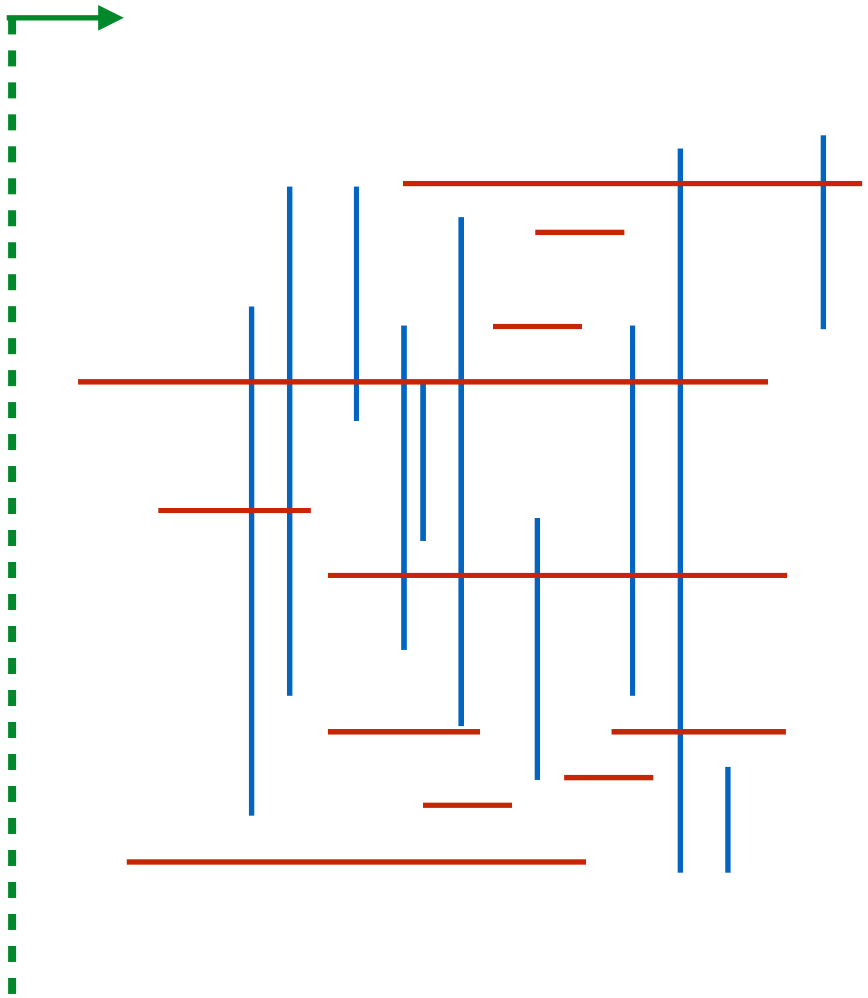
- Let X be the set of x-coordinates of all segments //our "events"
  - horizontal segment: x_start, x_end
  - vertical segment: x

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our “events”
- Sort X and traverse the events in order

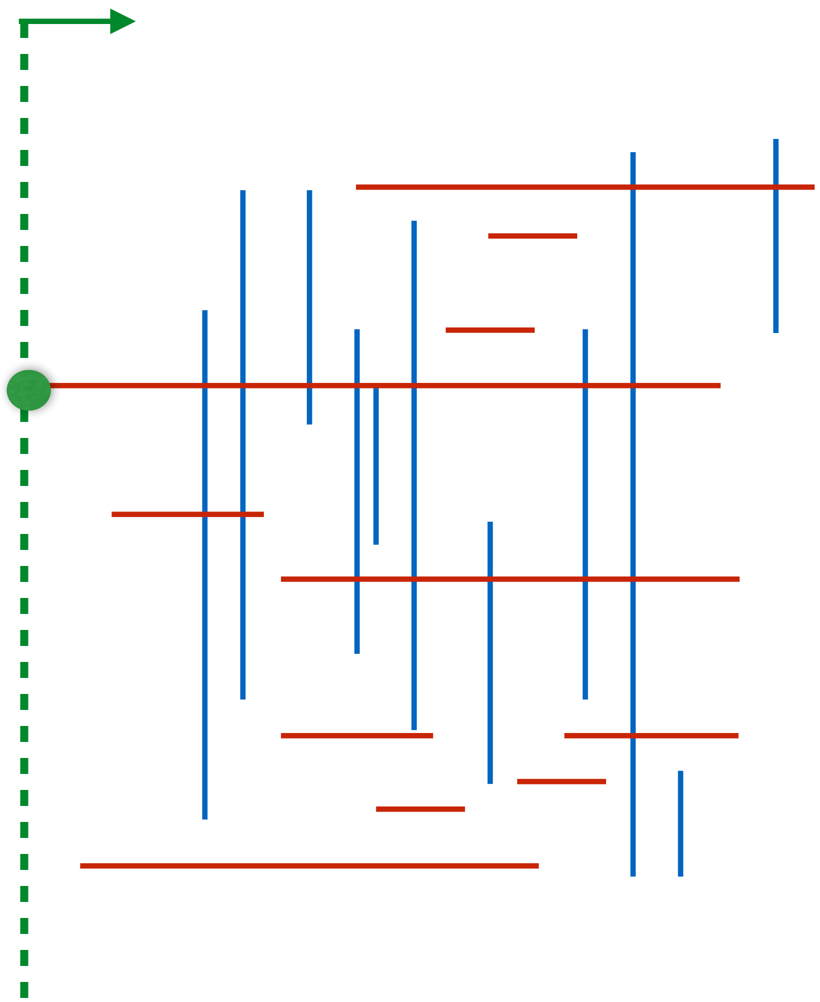
Orthogonal line segment intersection using BST



line sweep

- Let X be the set of x-coordinates of all segments   //our “events”
- Sort X and traverse the events in order

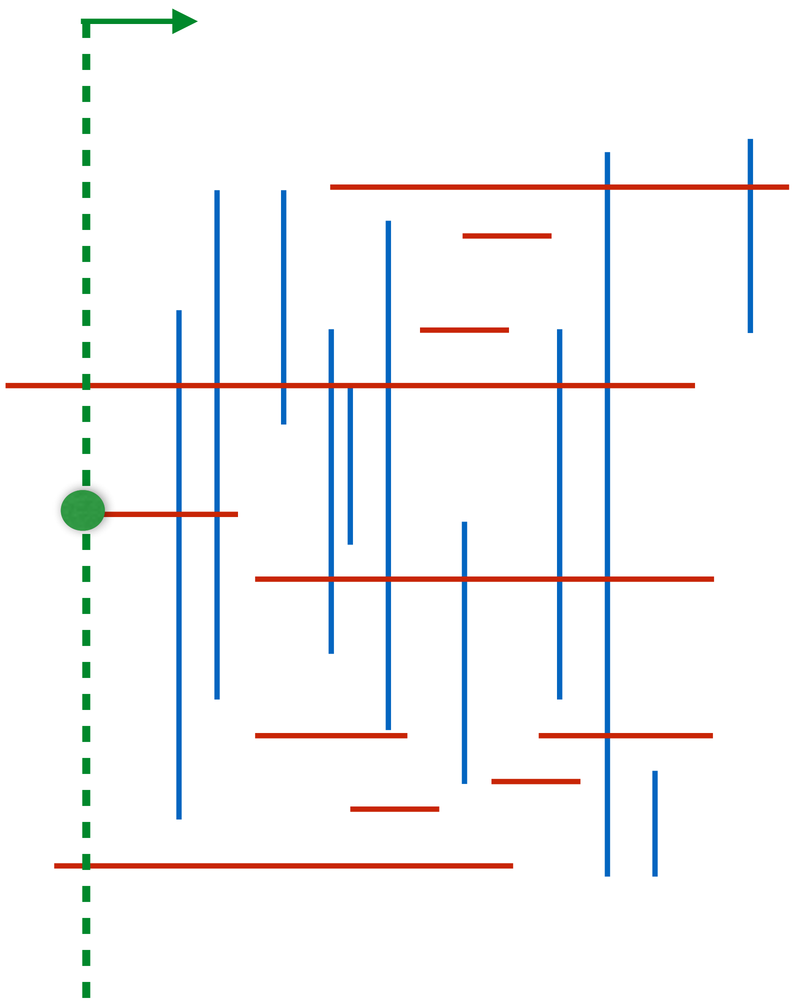
Orthogonal line segment intersection using BST



line sweep

- Let X be the set of x-coordinates of all segments //our "events"
- Sort X and traverse the events in order

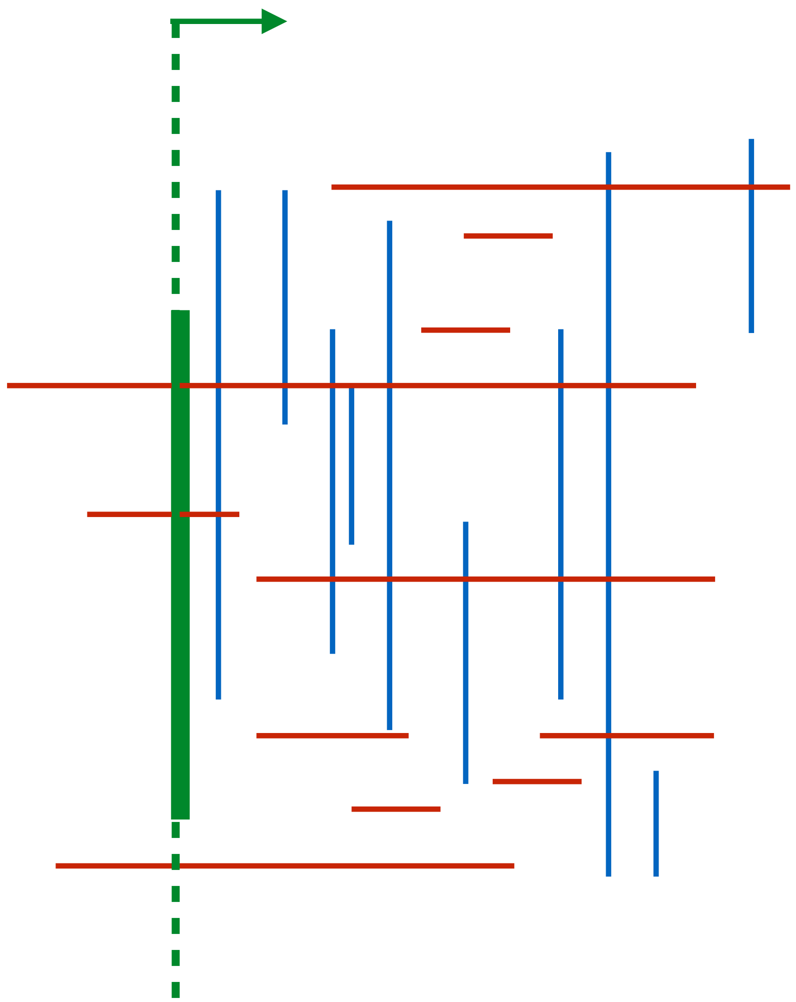
Orthogonal line segment intersection using BST



line sweep

- Let X be the set of x-coordinates of all segments   //our "events"
- Sort X and traverse the events in order

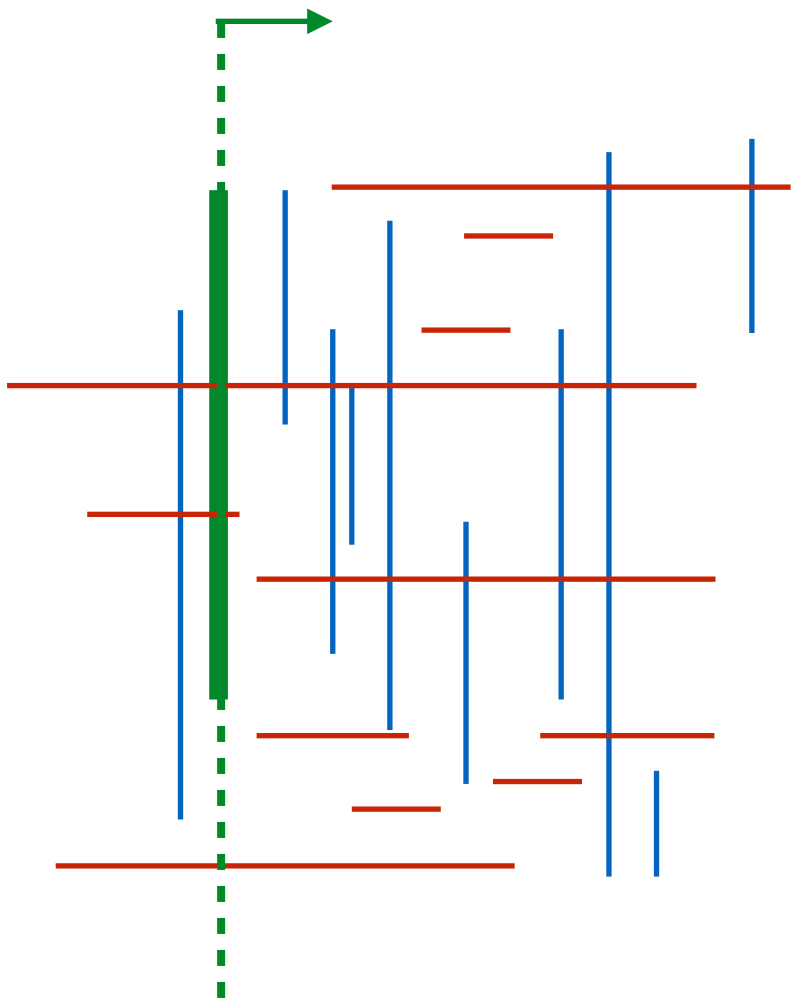
Orthogonal line segment intersection using BST



line sweep

- Let X be the set of x-coordinates of all segments   //our "events"
- Sort X and traverse the events in order

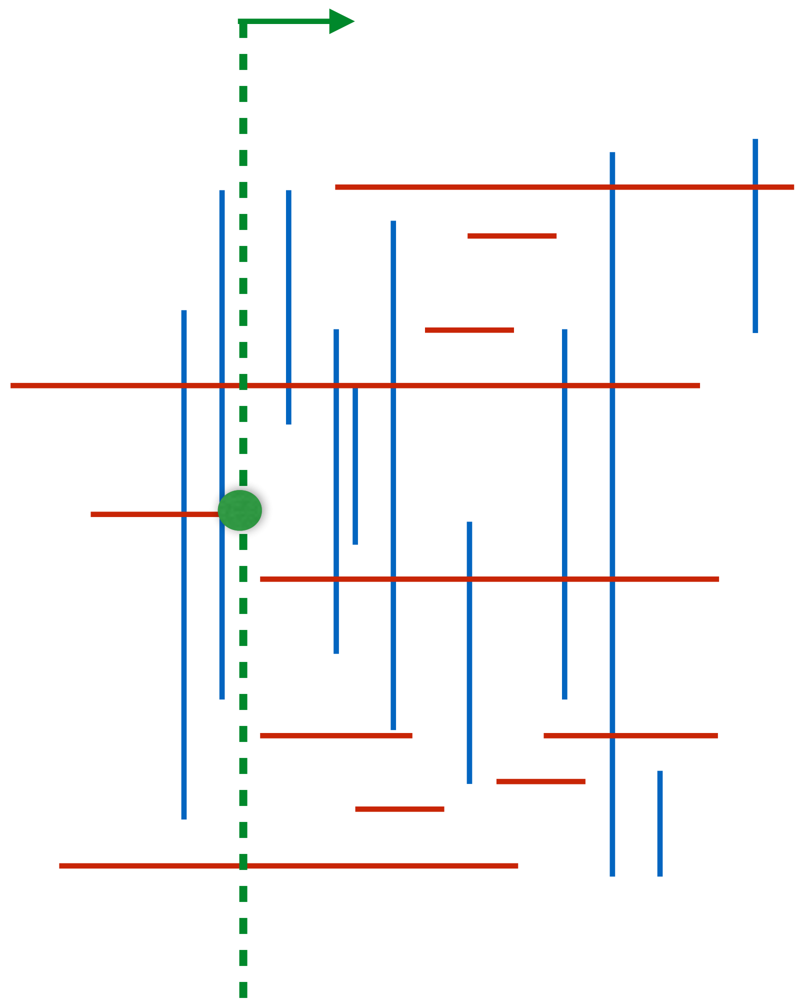
Orthogonal line segment intersection using BST



line sweep

- Let X be the set of x-coordinates of all segments   //our “events”
- Sort X and traverse the events in order

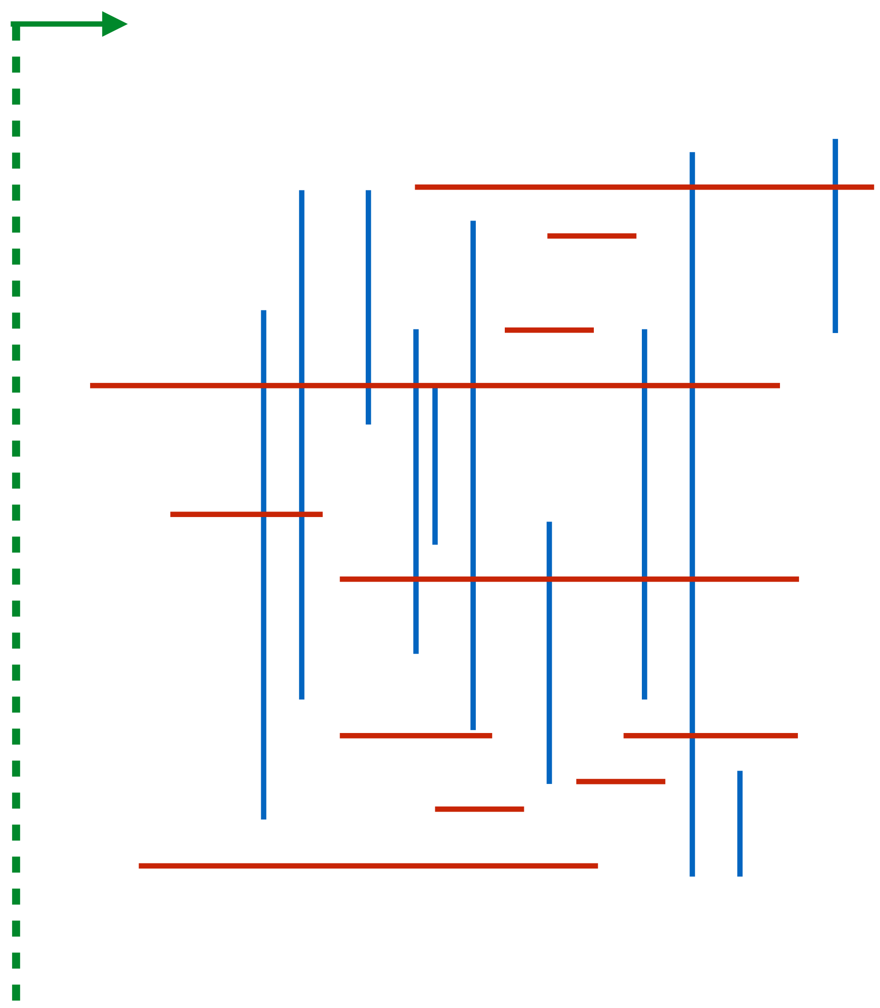
Orthogonal line segment intersection using BST



line sweep

- Let X be the set of x-coordinates of all segments   //our “events”
- Sort X and traverse the events in order

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x', y):

    //segment becomes active

    - insert segment (x,x',y) with key=y in AS
  - if x is end of horizontal segment (x, x', y):

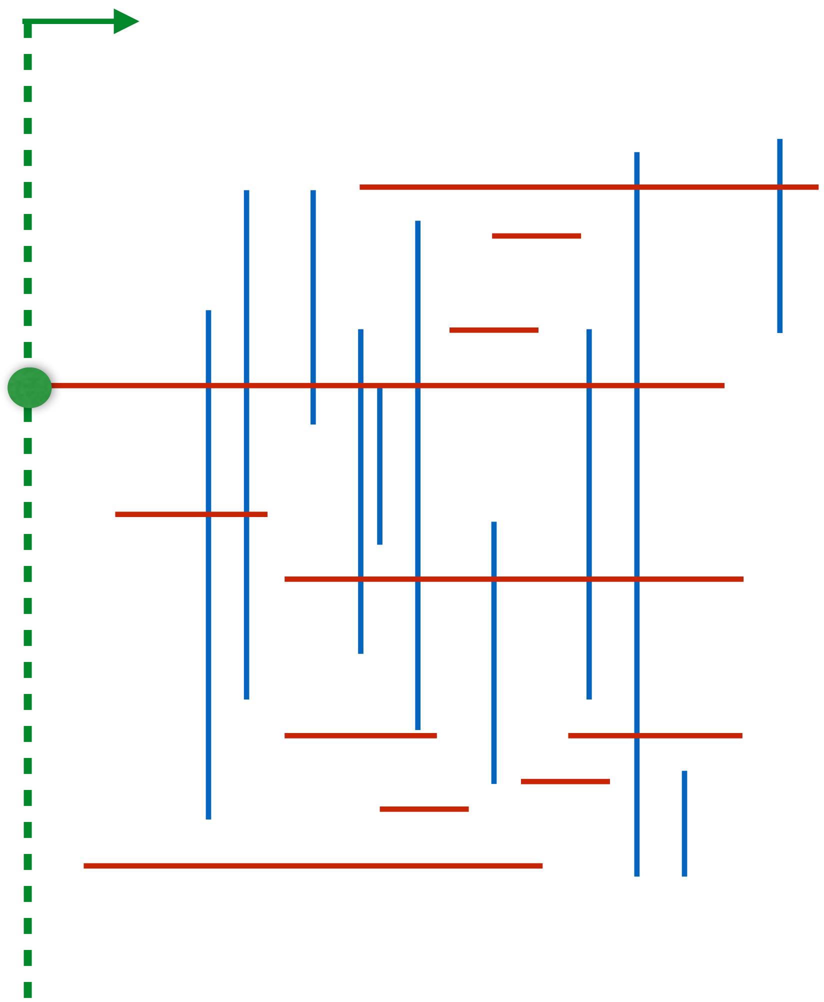
    //segment stops being active

    - delete segment (x,x',y) with key=y from AS
  - if x corresponds to a vertical segment (y, y',x):

    //all active segments start before x and end after x. We need those whose y is in [y,y']

    - range_search (y,y') and report

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x’, y):

    //segment becomes active

    - insert segment (x,x’,y) with key=y in AS
  - if x is end of horizontal segment (x, x’, y):

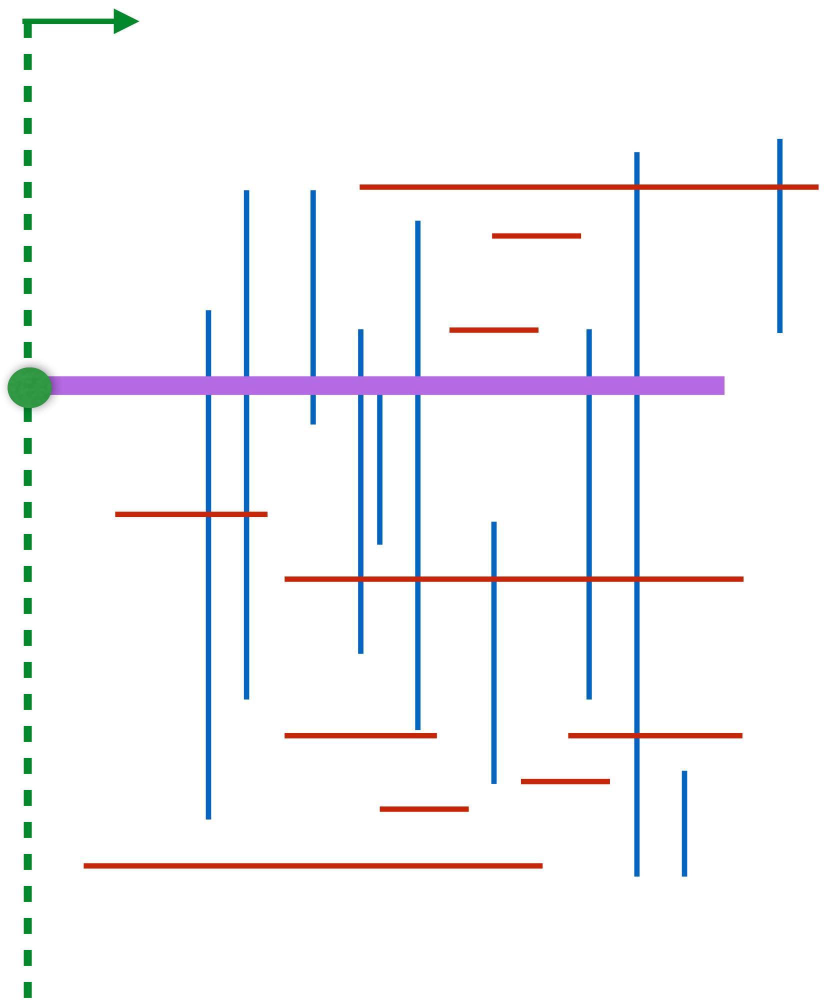
    //segment stops being active

    - delete segment (x,x’,y) with key=y from AS
  - if x corresponds to a vertical segment (y, y’,x):

    //all active segments start before x and end after x. We need those whose y is in [y,y’]

    - range_search (y,y’) and report

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x’, y):

    //segment becomes active

    - insert segment (x,x’,y) with key=y in AS
  - if x is end of horizontal segment (x, x’, y):

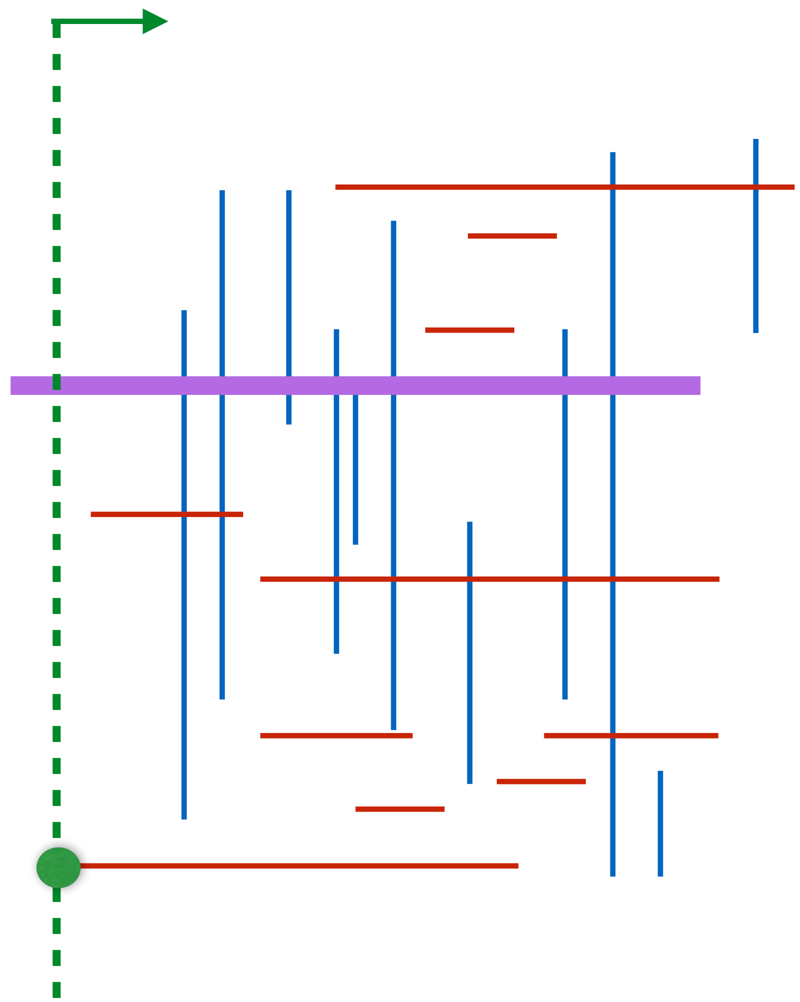
    //segment stops being active

    - delete segment (x,x’,y) with key=y from AS
  - if x corresponds to a vertical segment (y, y’,x):

    //all active segments start before x and end after x. We need those whose y is in [y,y’]

    - range_search (y,y’) and report

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x', y):

    //segment becomes active

    - insert segment (x,x',y) with key=y in AS
  - if x is end of horizontal segment (x, x', y):

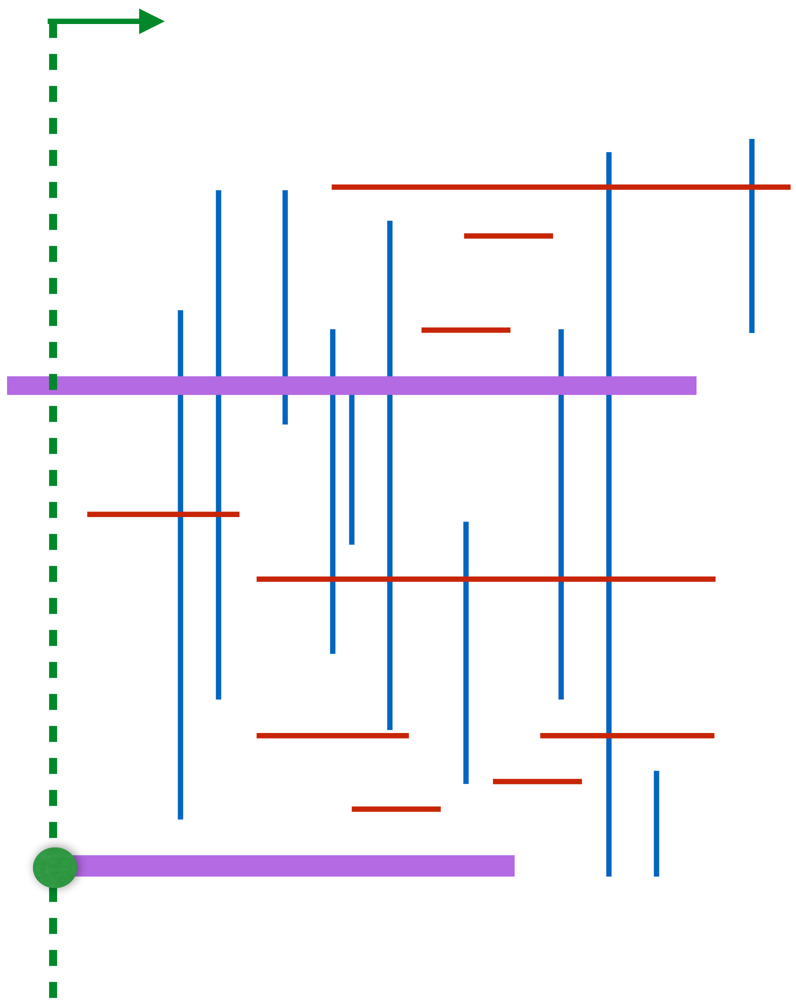
    //segment stops being active

    - delete segment (x,x',y) with key=y from AS
  - if x corresponds to a vertical segment (y, y',x):

    //all active segments start before x and end after x. We need those whose y is in [y,y']

    - range_search (y,y') and report

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x', y):

    //segment becomes active

    - insert segment (x,x',y) with key=y in AS
  - if x is end of horizontal segment (x, x', y):

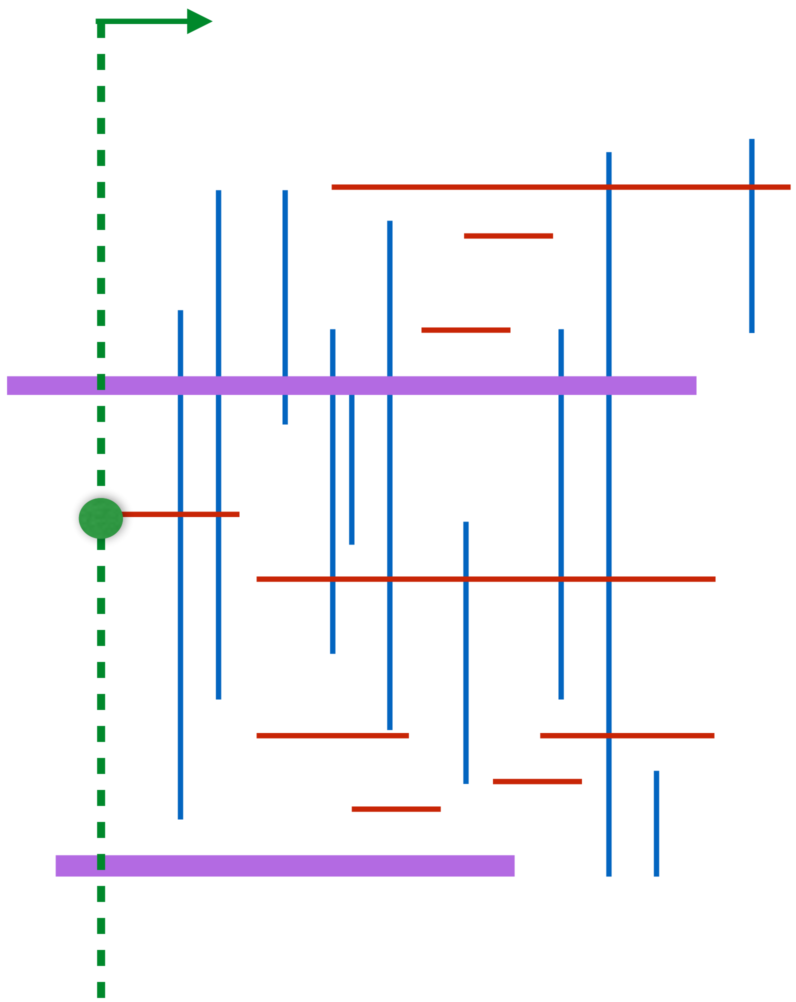
    //segment stops being active

    - delete segment (x,x',y) with key=y from AS
  - if x corresponds to a vertical segment (y, y',x):

    //all active segments start before x and end after x. We need those whose y is in [y,y']

    - range_search (y,y') and report

Orthogonal line segment intersection using BST



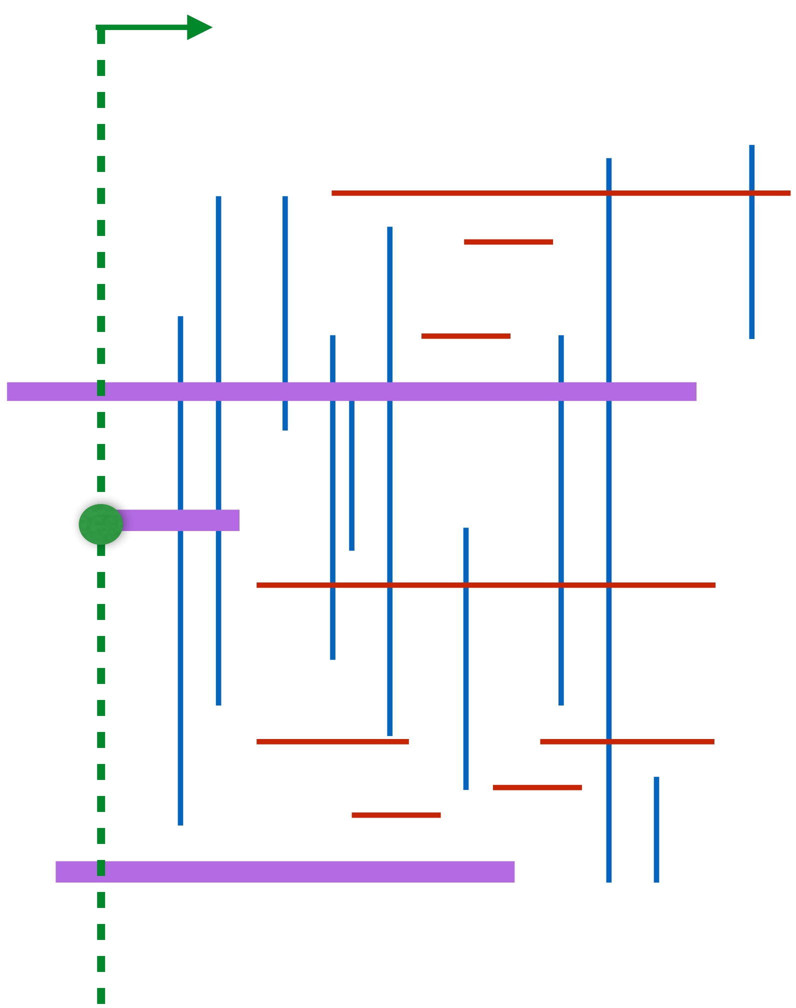
- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x', y):

    //segment becomes active
    - insert segment (x,x',y) with key=y in AS
  - if x is end of horizontal segment (x, x', y):

    //segment stops being active
    - delete segment (x,x',y) with key=y from AS
  - if x corresponds to a vertical segment (y, y',x):

    //all active segments start before x and end after x. We need those whose y is in [y,y']
    - range_search (y,y') and report

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x', y):

    //segment becomes active

    - insert segment (x,x',y) with key=y in AS
  - if x is end of horizontal segment (x, x', y):

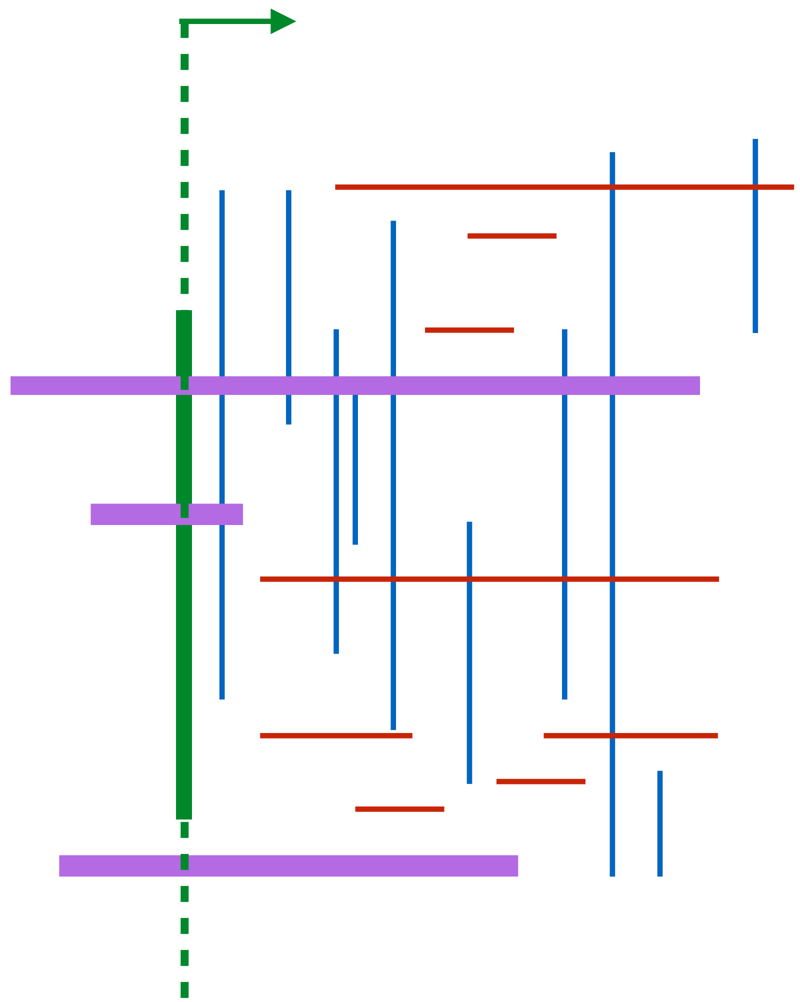
    //segment stops being active

    - delete segment (x,x',y) with key=y from AS
  - if x corresponds to a vertical segment (y, y',x):

    //all active segments start before x and end after x. We need those whose y is in [y,y']

    - range_search (y,y') and report

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x', y):

    //segment becomes active

    - insert segment (x,x',y) with key=y in AS
  - if x is end of horizontal segment (x, x', y):

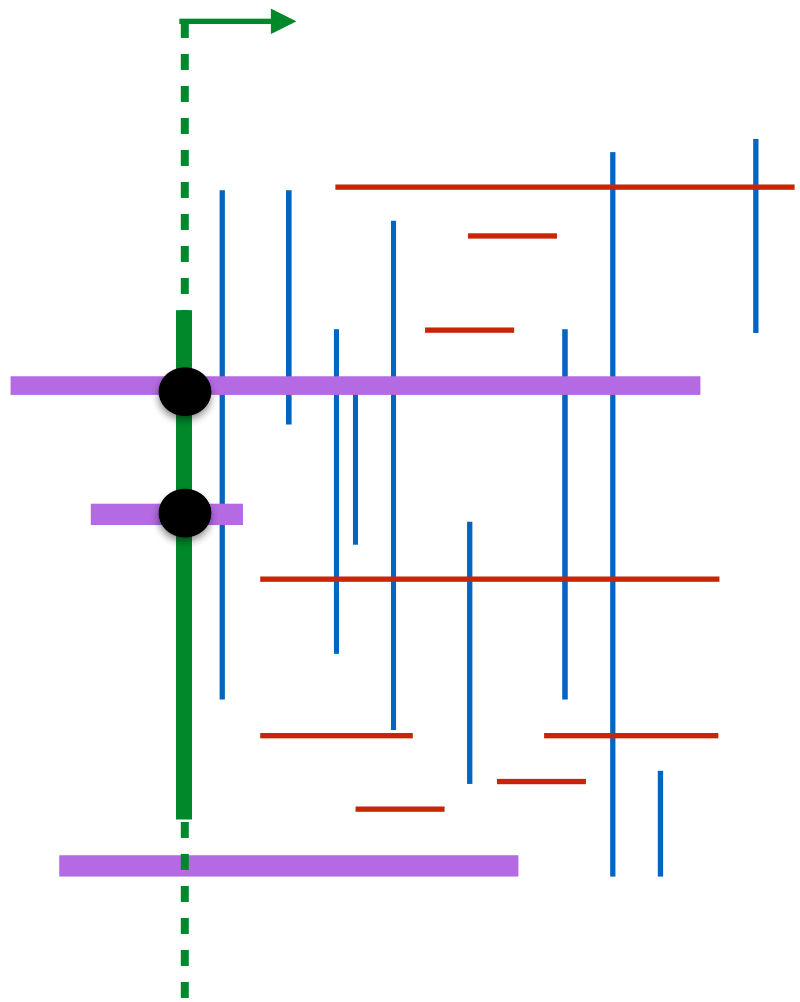
    //segment stops being active

    - delete segment (x,x',y) with key=y from AS
  - if x corresponds to a vertical segment (y, y',x):

    //all active segments start before x and end after x. We need those whose y is in [y,y']

    - range_search (y,y') and report

Orthogonal line segment intersection using BST



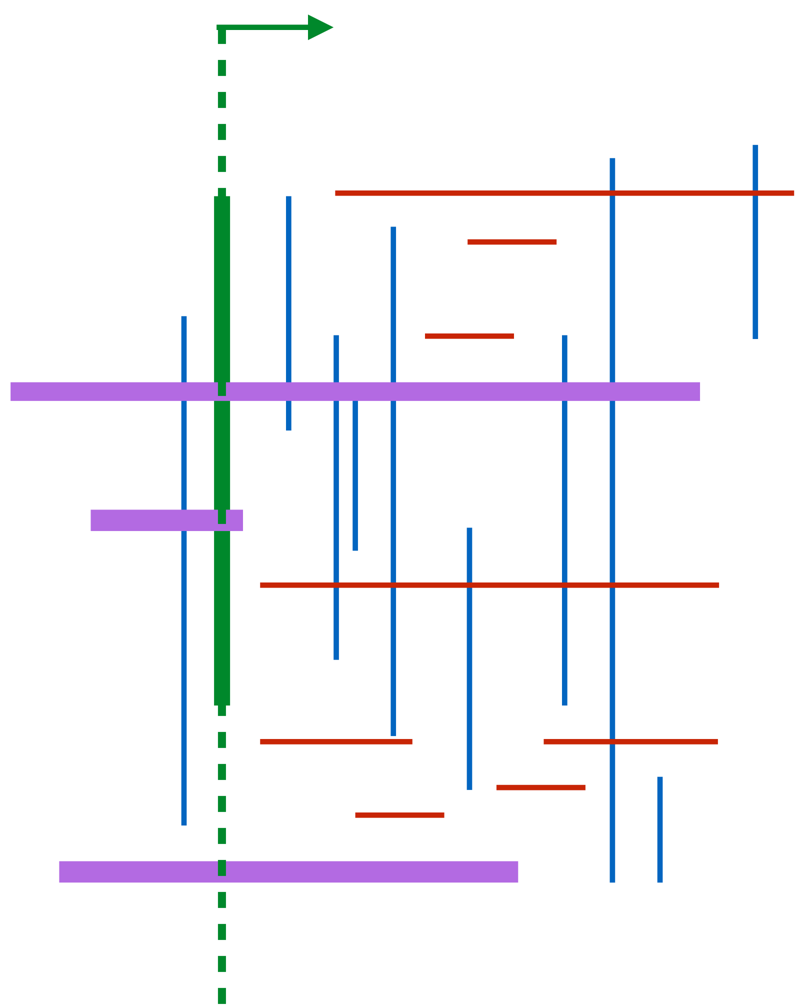
- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x', y):

    //segment becomes active
    - insert segment (x,x',y) with key=y in AS
  - if x is end of horizontal segment (x, x', y):

    //segment stops being active
    - delete segment (x,x',y) with key=y from AS
  - if x corresponds to a vertical segment (y, y',x):

    //all active segments start before x and end after x. We need those whose y is in [y,y']
    - range_search (y,y') and report

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x’, y):

    //segment becomes active

    - insert segment (x,x’,y) with key=y in AS
  - if x is end of horizontal segment (x, x’, y):

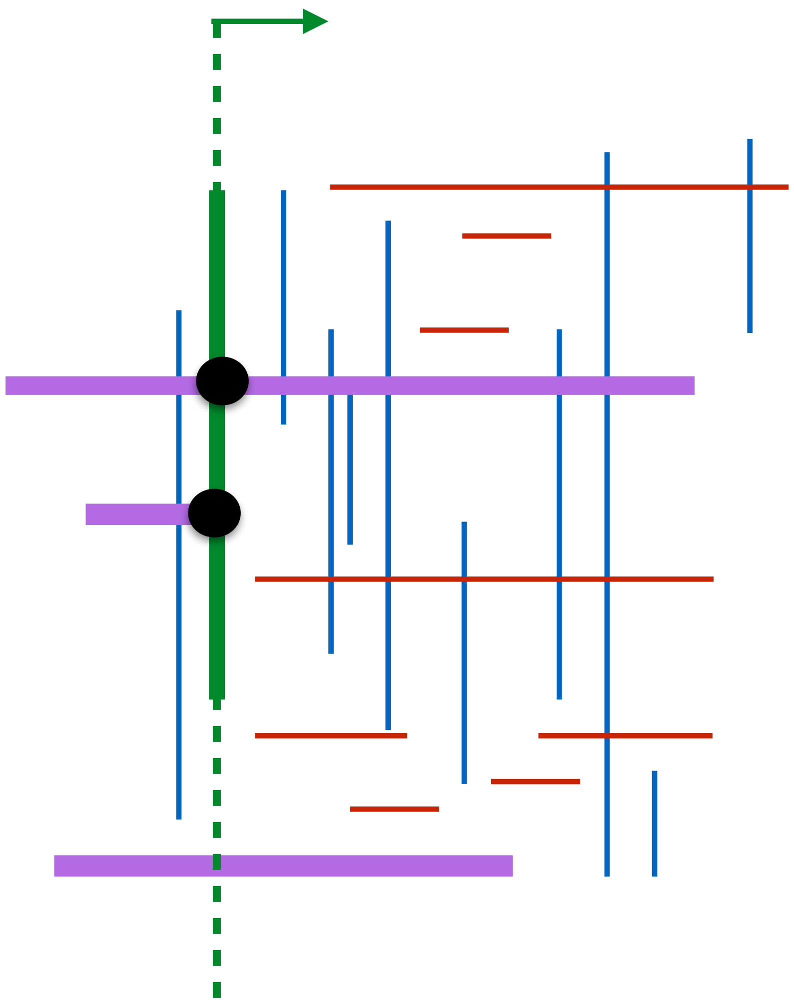
    //segment stops being active

    - delete segment (x,x’,y) with key=y from AS
  - if x corresponds to a vertical segment (y, y’,x):

    //all active segments start before x and end after x. We need those whose y is in [y,y’]

    - range_search (y,y’) and report

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x', y):

    //segment becomes active

    - insert segment (x,x',y) with key=y in AS
  - if x is end of horizontal segment (x, x', y):

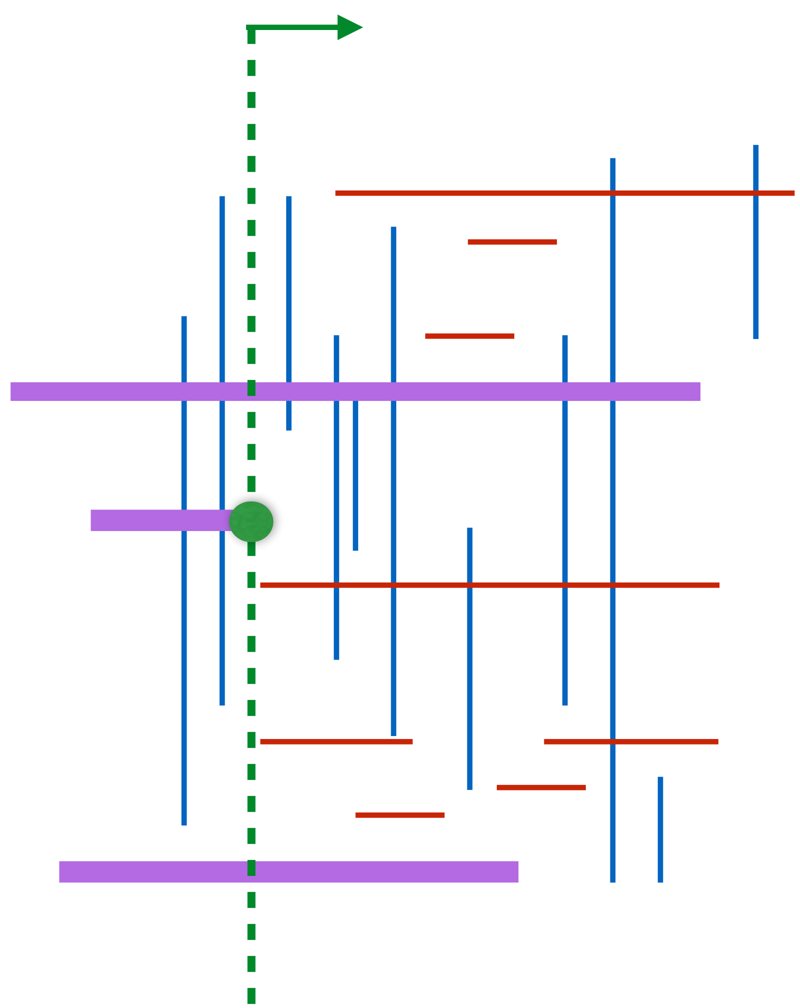
    //segment stops being active

    - delete segment (x,x',y) with key=y from AS
  - if x corresponds to a vertical segment (y, y',x):

    //all active segments start before x and end after x. We need those whose y is in [y,y']

    - range_search (y,y') and report

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
    - if x is start of horizontal segment (x, x', y):

      //segment becomes active

        - insert segment (x,x',y) with key=y in AS
    - if x is end of horizontal segment (x, x', y):

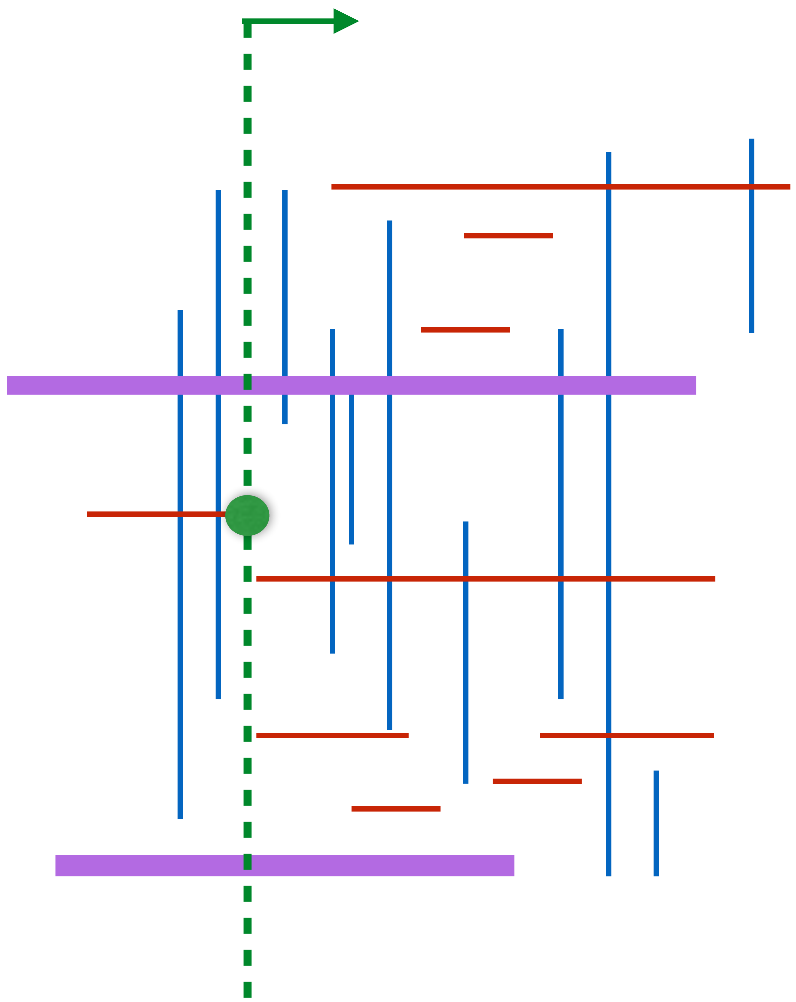
      //segment stops being active

        - delete segment (x,x',y) with key=y from AS
    - if x corresponds to a vertical segment (y, y',x):

      //all active segments start before x and end after x. We need those whose y is in [y,y']

        - range_search (y,y') and report

Orthogonal line segment intersection using BST



- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x', y):

    //segment becomes active

    - insert segment (x,x',y) with key=y in AS
  - if x is end of horizontal segment (x, x', y):

    //segment stops being active

    - delete segment (x,x',y) with key=y from AS
  - if x corresponds to a vertical segment (y, y',x):

    //all active segments start before x and end after x. We need those whose y is in [y,y']

    - range_search (y,y') and report

Orthogonal line segment intersection using BST

Exercises:

- Pick another example and simulate the algorithm
- How do you implement the AS?
- Analysis?

- Let X be the set of x-coordinates of all segments //our events
- Initialize AS = {}
- Sort X and traverse the events in order; let x be the next event in X
  - if x is start of horizontal segment (x, x', y):

    //segment becomes active

    - insert segment (x,x',y) with key=y in AS
  - if x is end of horizontal segment (x, x', y):

    //segment stops being active

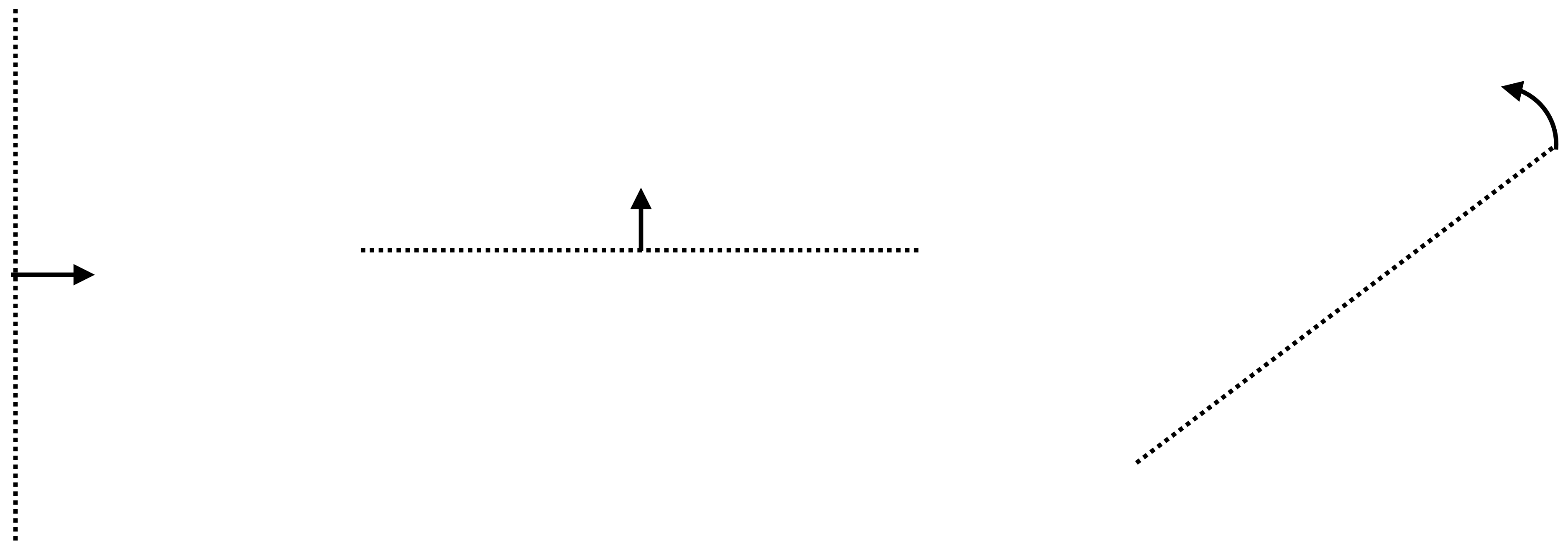
    - delete segment (x,x',y) with key=y from AS
  - if x corresponds to a vertical segment (y, y',x):

    //all active segments start before x and end after x. We need those whose y is in [y,y']

    - range_search (y,y') and report

Line sweep

- Frequently used technique
- Line can be horizontal or vertical or radial or ….



Line sweep

- Frequently used technique
- Line can be horizontal or vertical or radial or ….
- Traverse events in order and maintain an Active Structure (AS)
  - AS maintains objects that are “active” (started but not ended) in other words they are intersected by the present sweep line
  - at certain events, insert in AS
  - at certain events, delete from AS
  - at other events, query AS