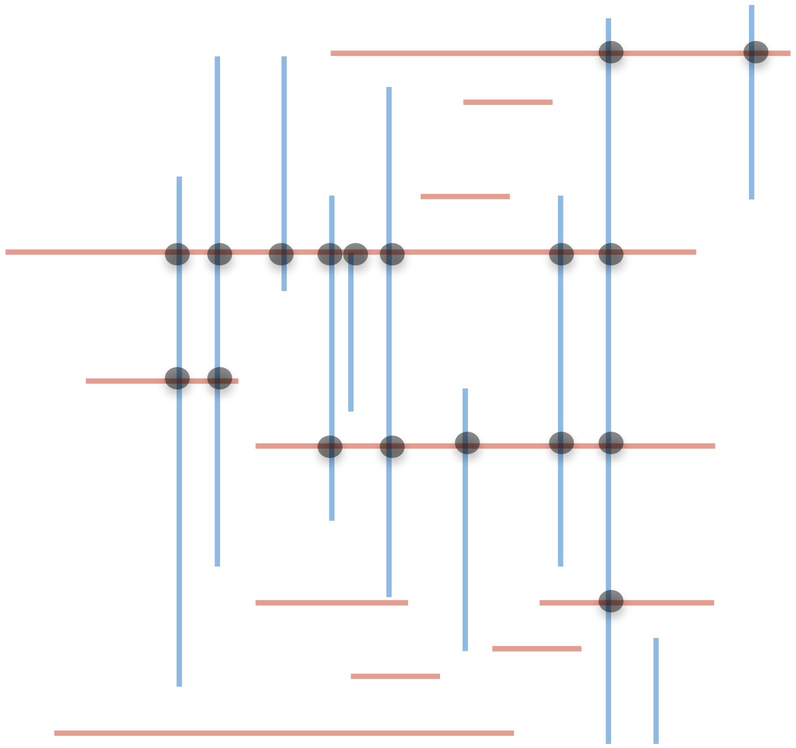
Line segment intersection:

(I) **Orthogonal** line segment intersection



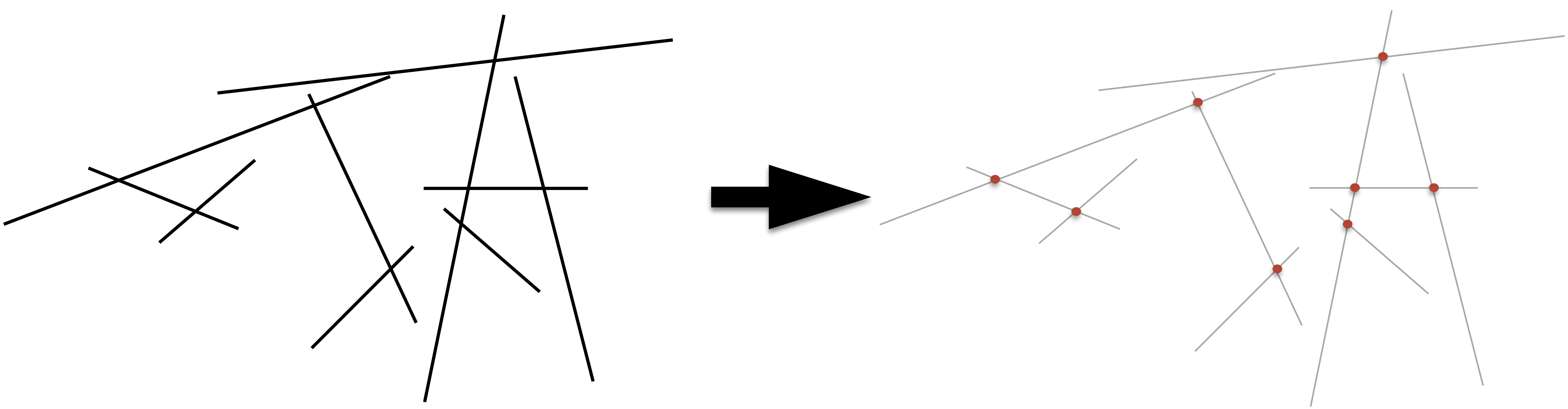
Computational Geometry [csci 3250]
Laura Toma
Bowdoin College

Line segment intersection

- The problem (what)
- Applications (why)
- Algorithms (how)
    - A special case: Orthogonal line segments
    - Next time: General case: Bentley-Otman line sweep algorithm

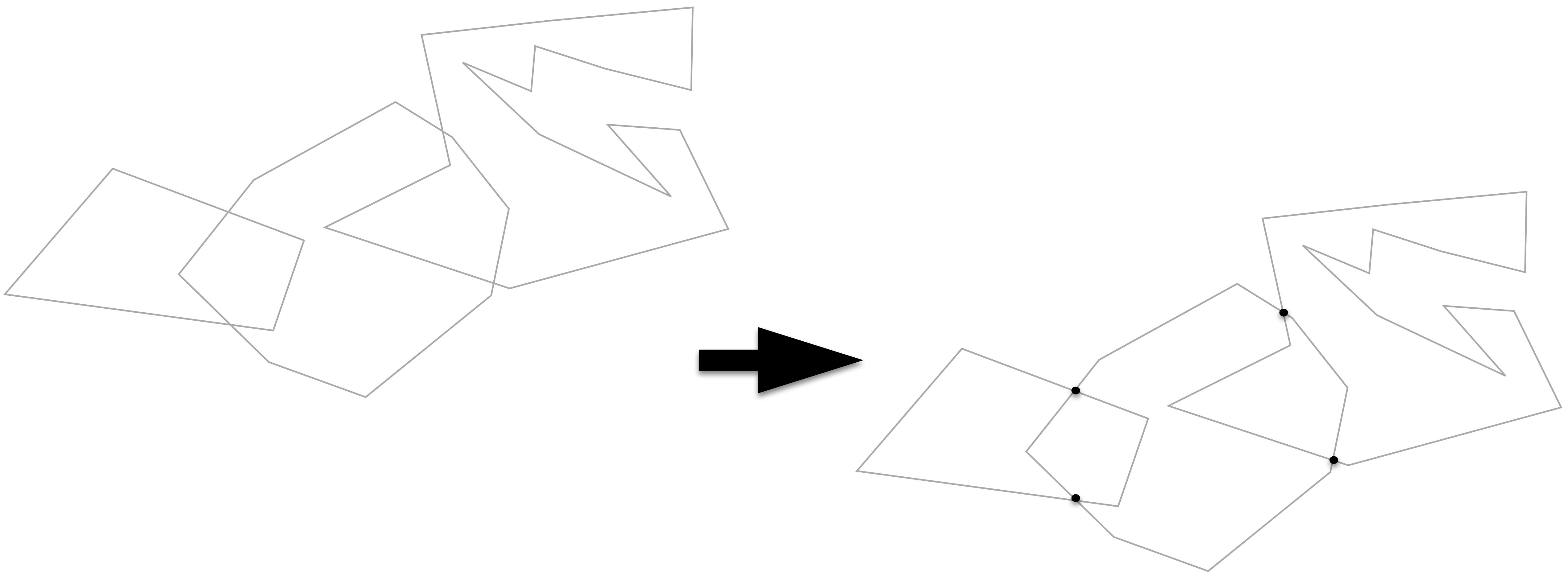
Line segment intersection

Problem: Given a set of line segments in 2D, find all their pairwise intersections.



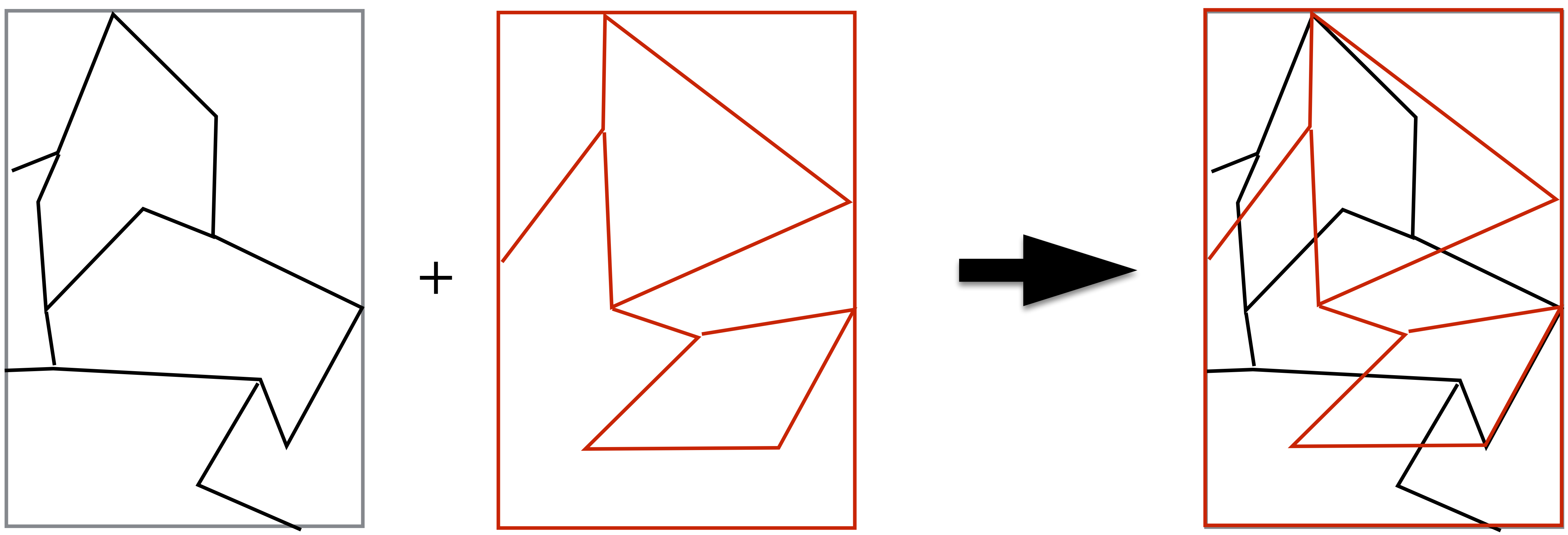
Line segment intersection

Problem: Given a set of line segments in 2D, find all their pairwise intersections.



Line segment intersection

Problem: Given a set of line segments in 2D, find all their pairwise intersections.

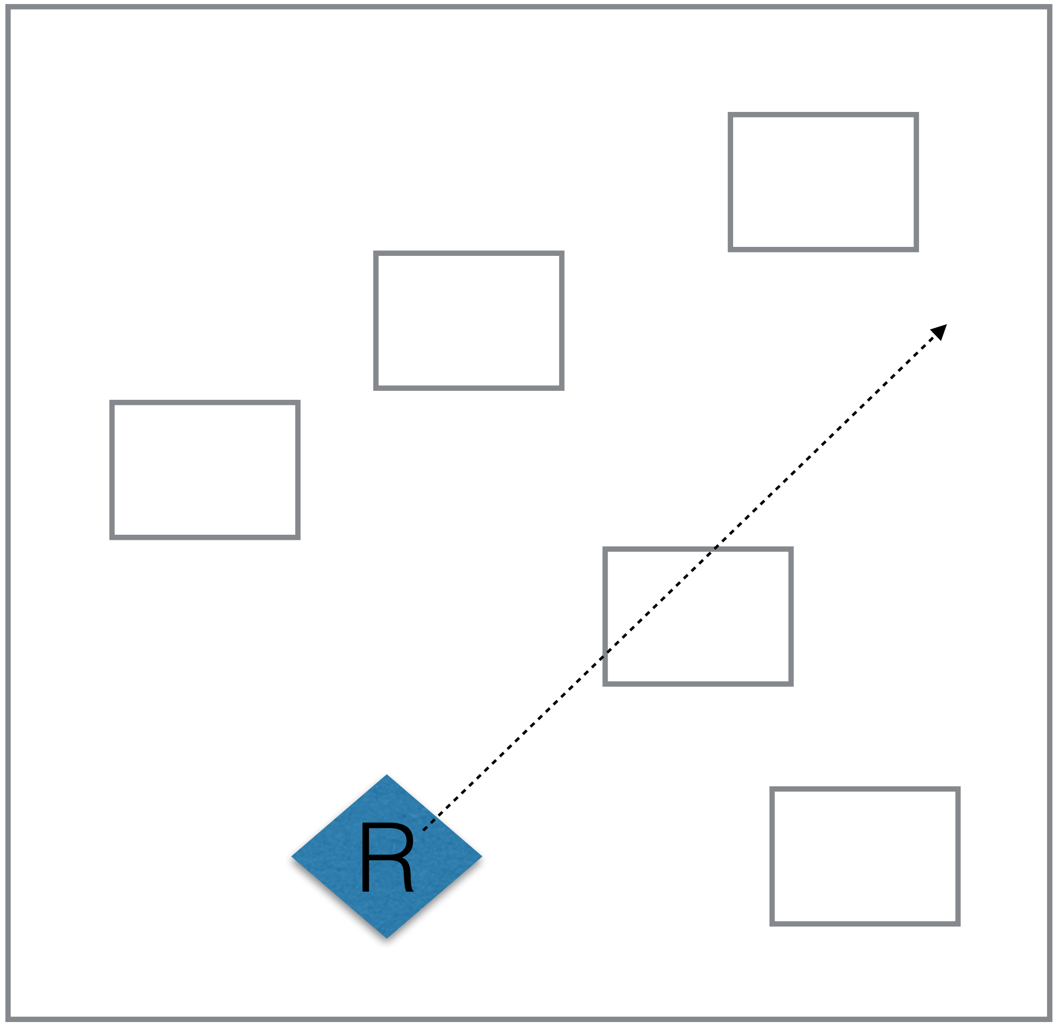


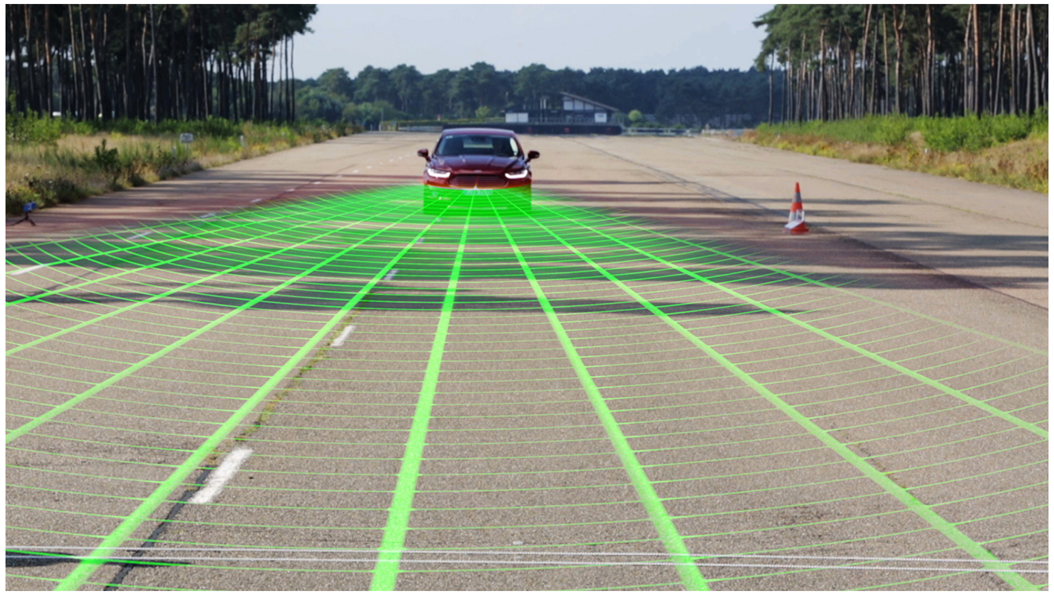
Line segment intersection:

Applications

Applications

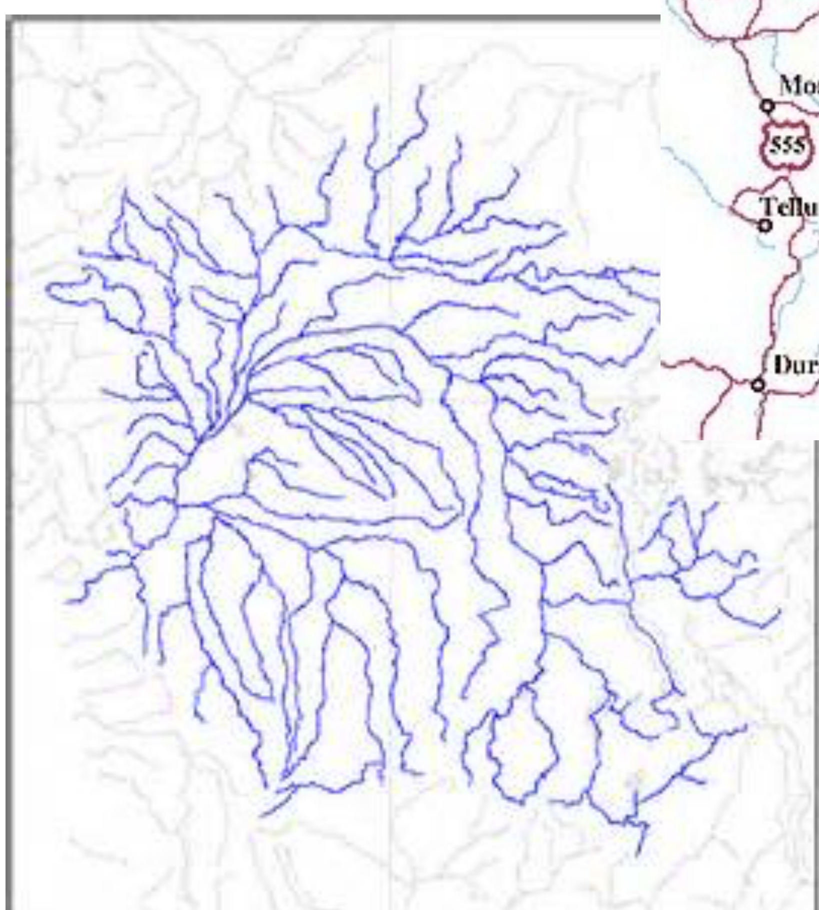
Motion planning and collision detection in autonomous systems/robotics





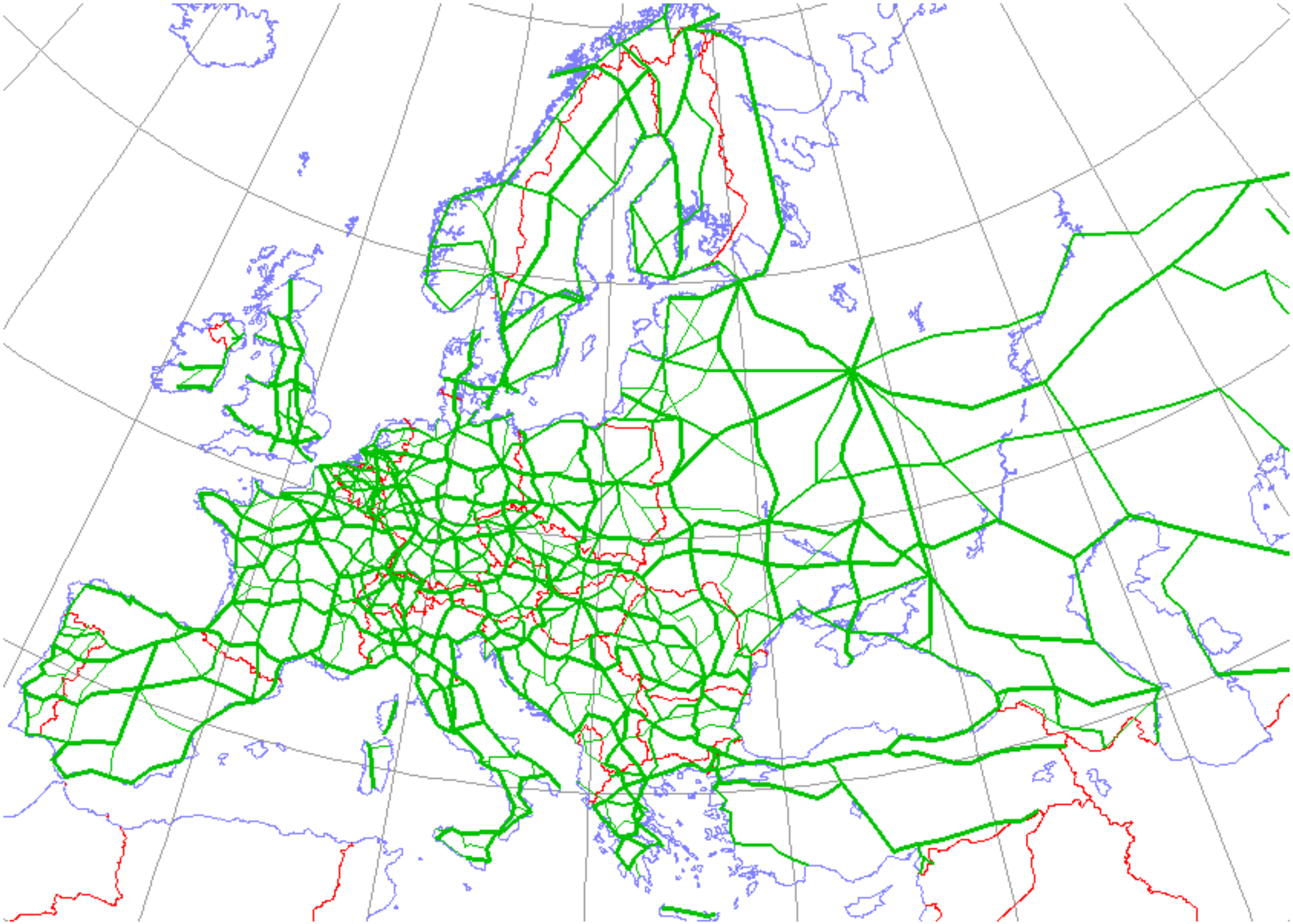
Applications

Geographic data: River networks, road networks, railways, ..



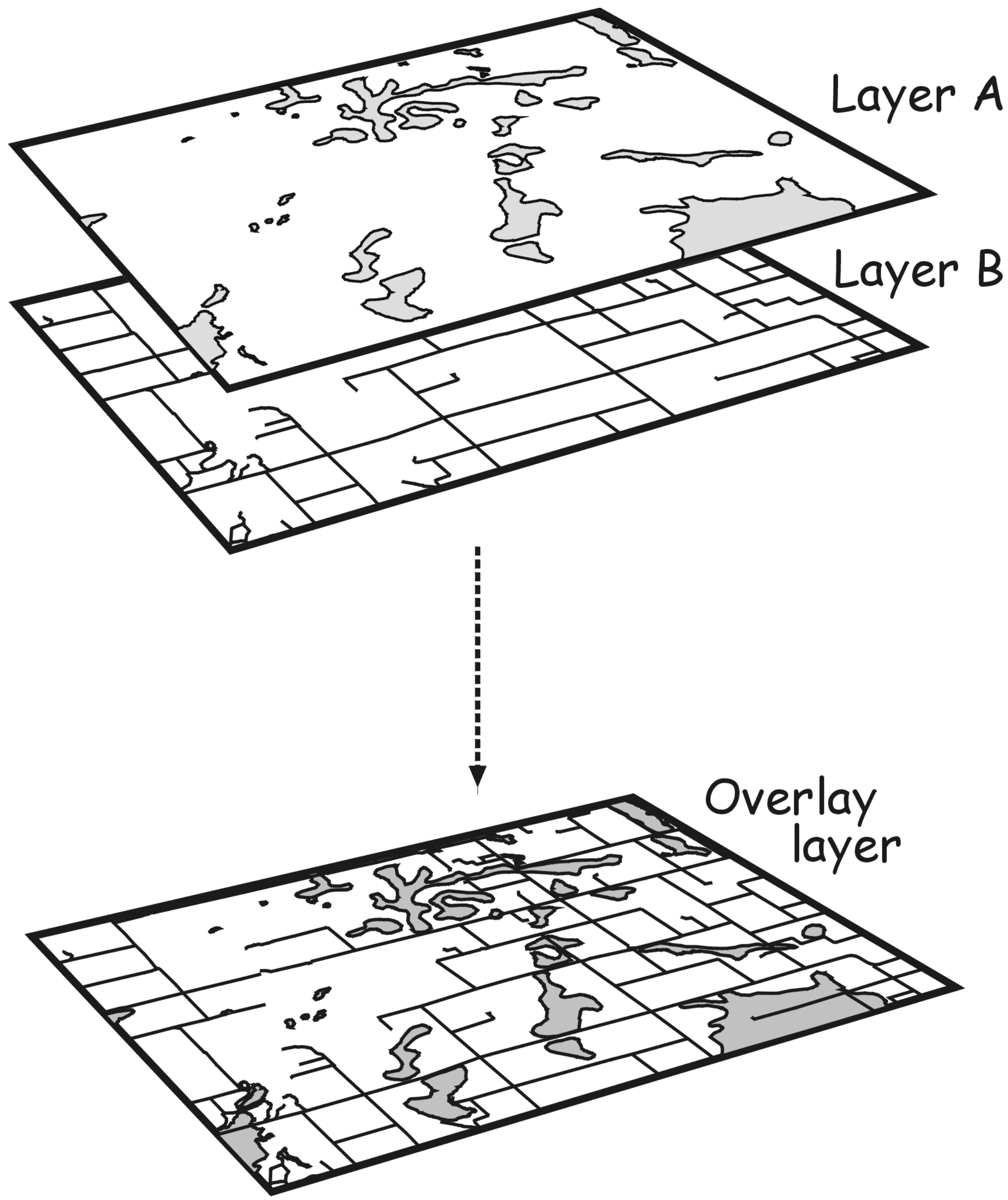
Applications

Geographic data: River networks, road networks, railways, ..



Applications

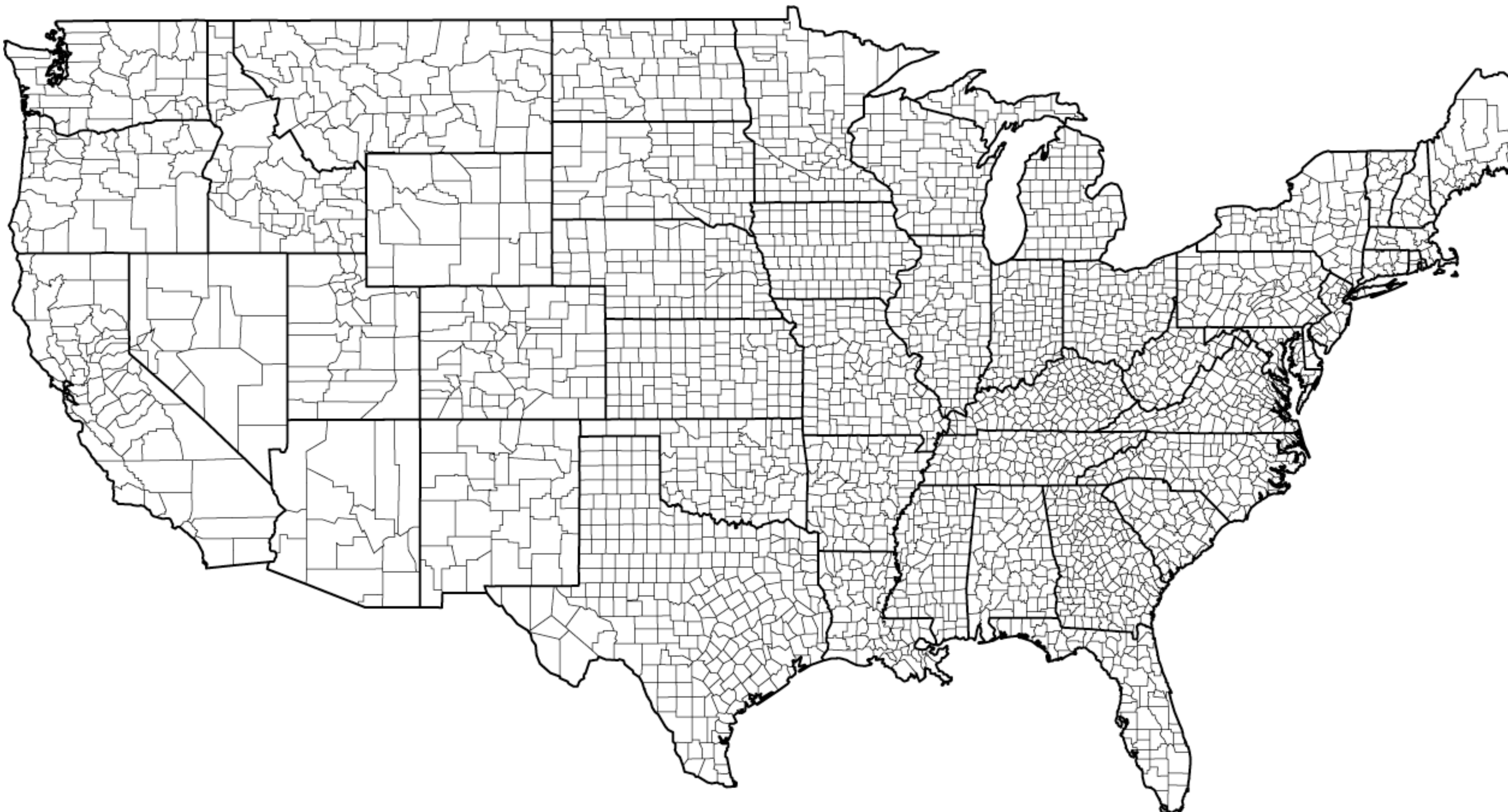
Map overlay in GIS



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

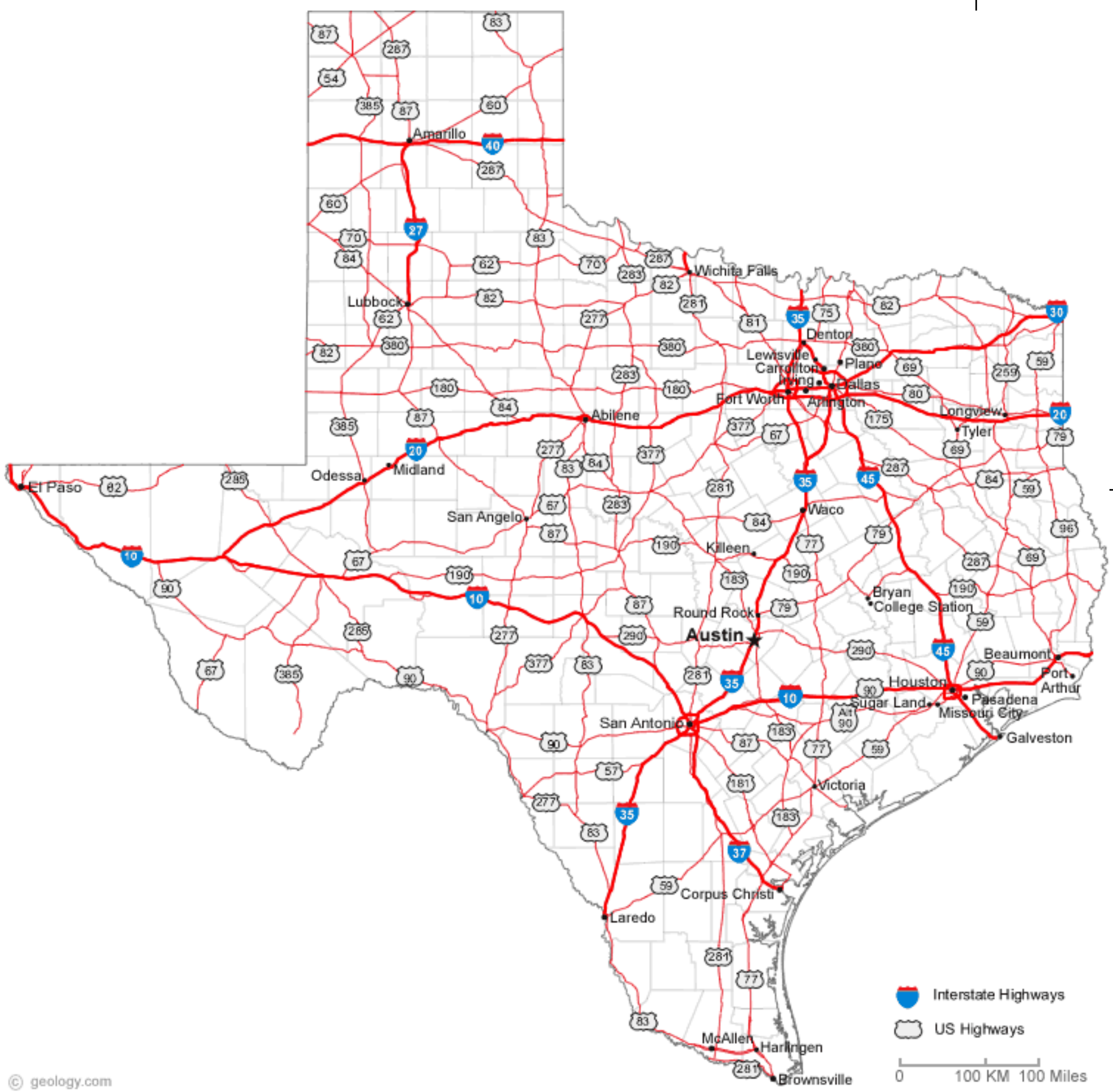
Applications

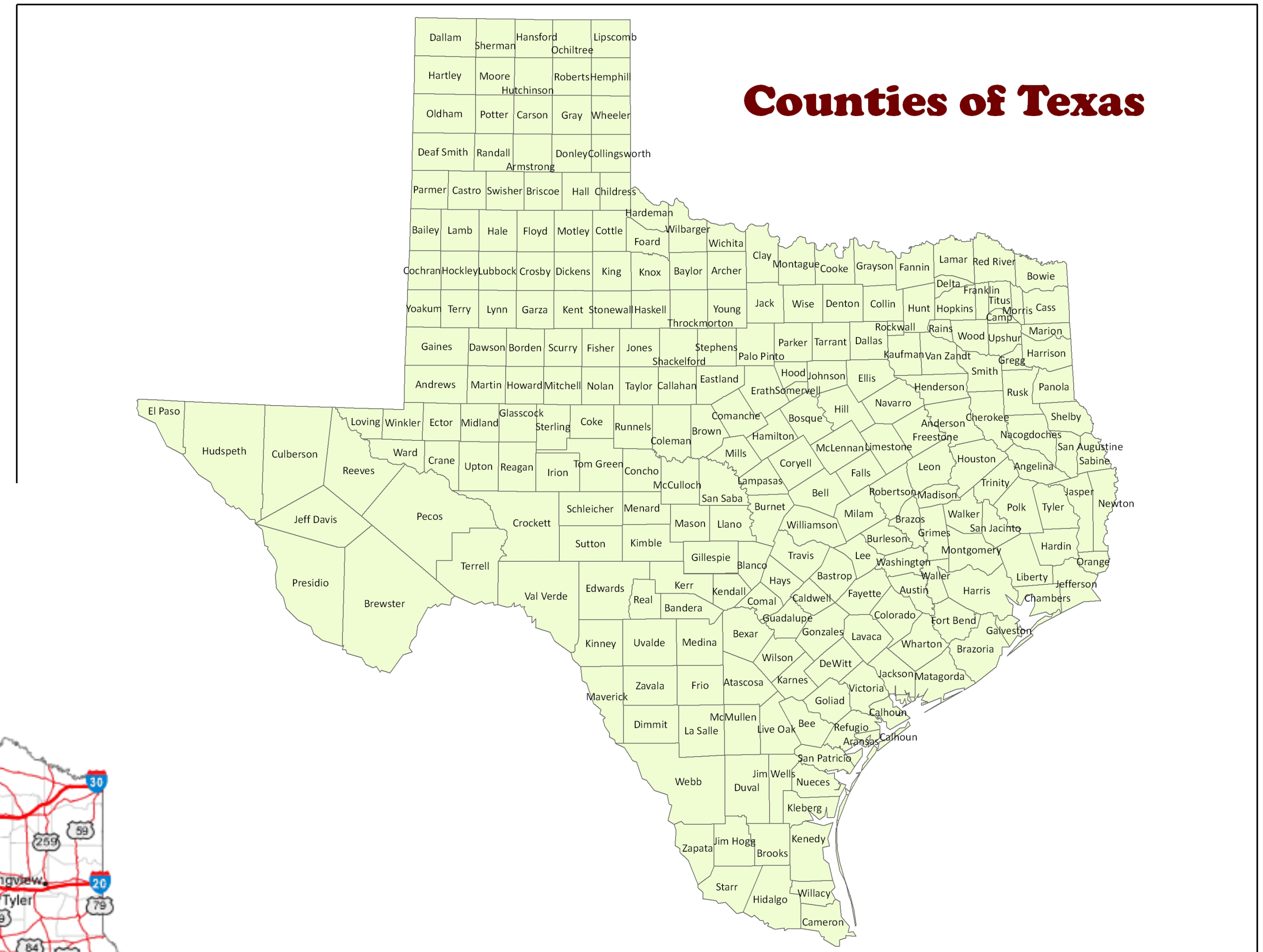
Segment data in GIS: river network, road networks, counties, etc



Applications

Map overlay in GIS





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

Computing line segment intersection:

Algorithms

Notation

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

Naive

Problem: Given a set of n line segments in 2D, find all their pairwise intersections.

Naive

Notation

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

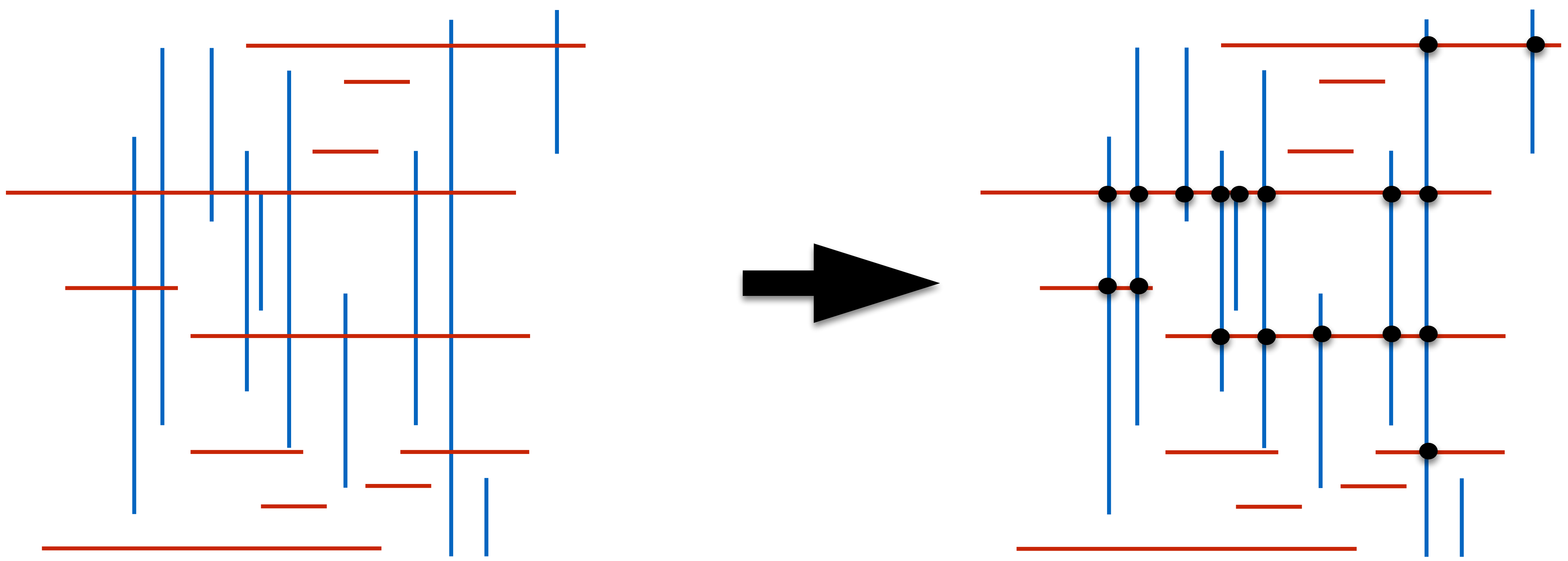
Problem: Given a set of n line segments in 2D, find all their pairwise intersections.

To think:

- Give upper and lower bounds for k, draw examples that achieve these bounds.
- 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? (how fast would you hope to be able to solve it?)

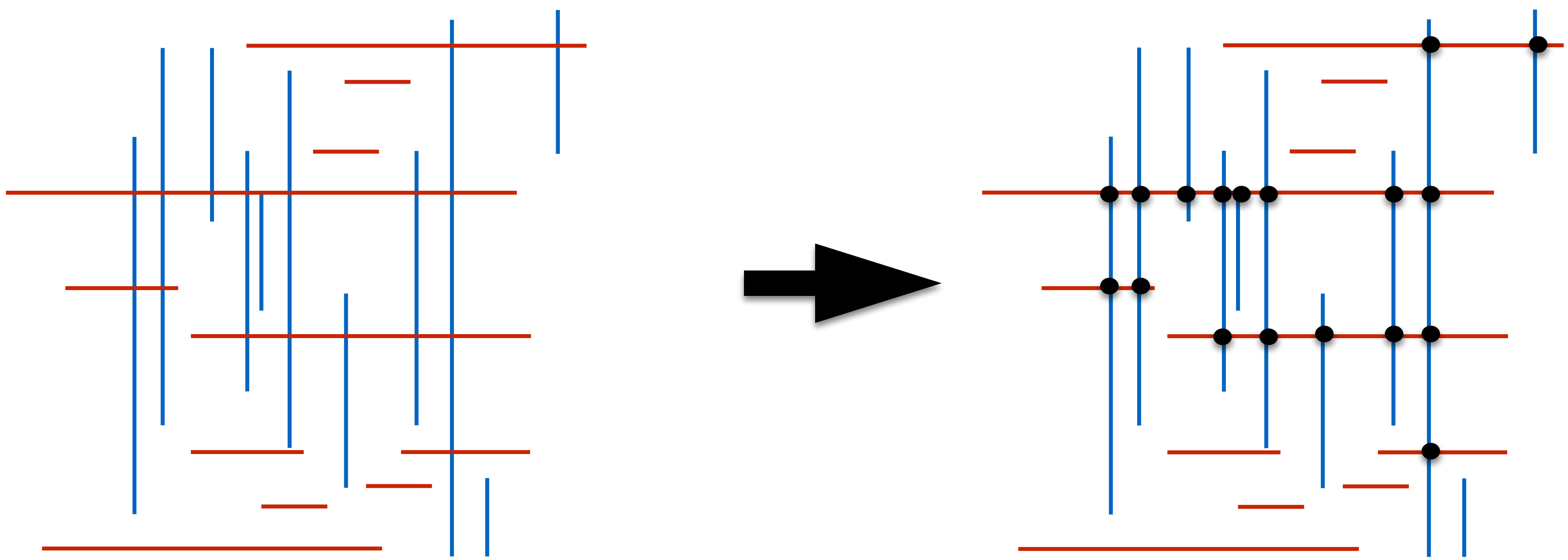
A special case: Orthogonal line segment intersection

Problem: Given a set of **orthogonal** line segments in 2D, find all their pairwise intersections.



A special case: Orthogonal line segment intersection

Problem: Given a set of **orthogonal** line segments in 2D, find all their pairwise intersections.



To think:

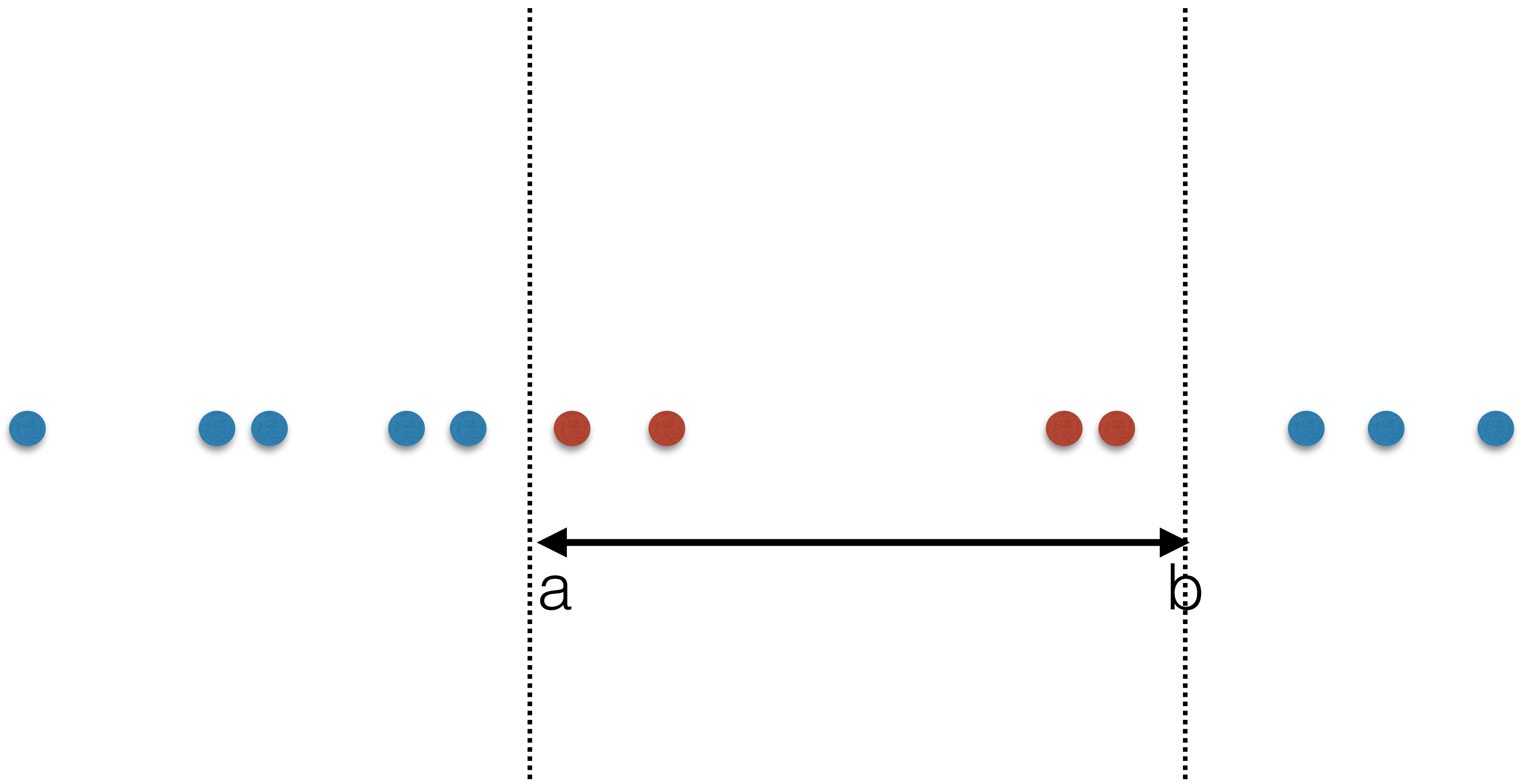
- Come up with a straightforward algorithm and analyze its time
- Can you do better?

detour: range searching

1D Range Searching

- Given a set of values $P = \{x_1, x_2, x3, \ldots x_n\}$
- Pre-process it in order to answer

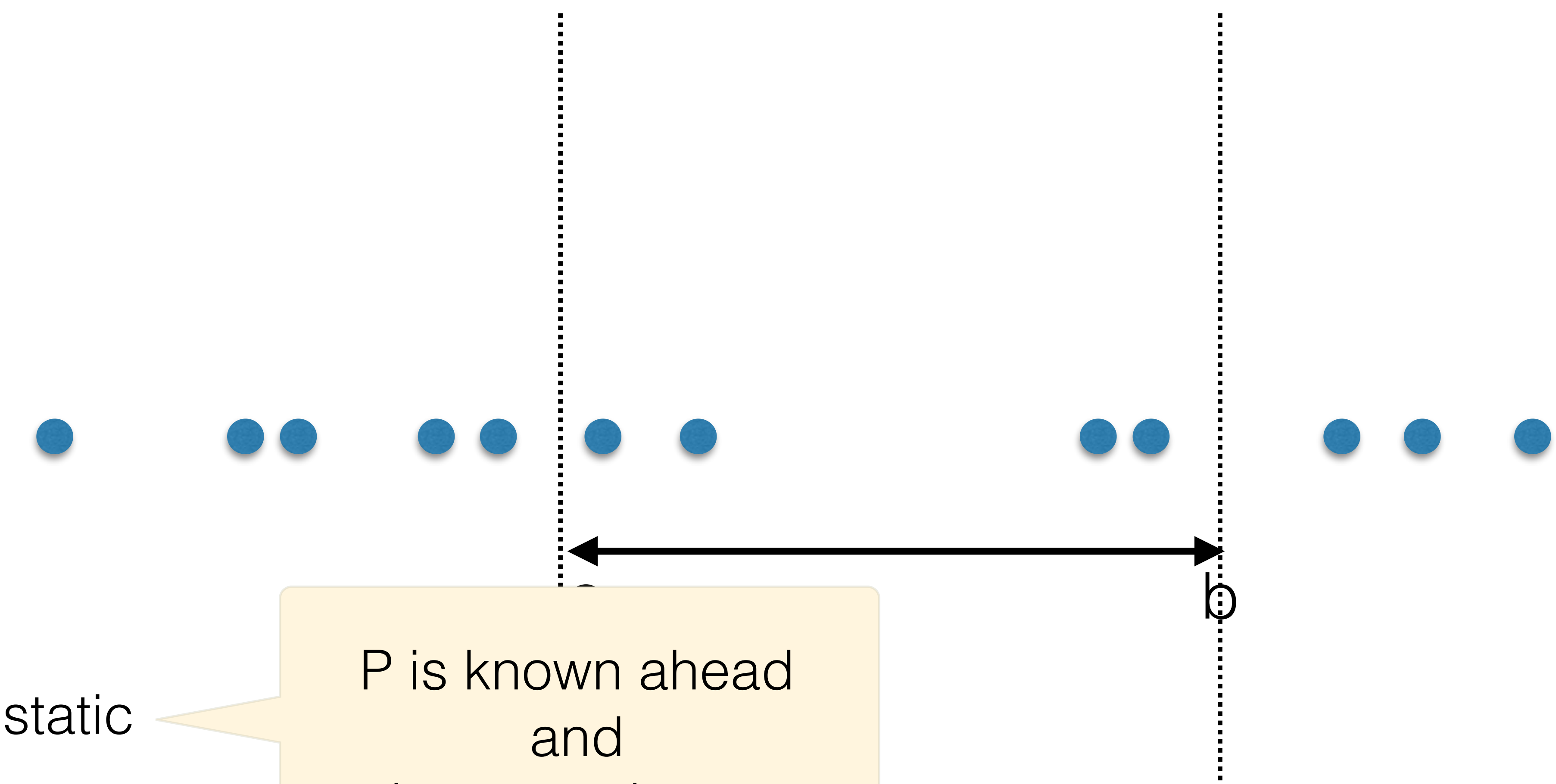
    rangeSearch(a,b): return all elements in P in interval (a,b)



1D Range Searching

- Given a set of values P = {x_1, x_2, x3, …x_n }
- Pre-process it in order to answer

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



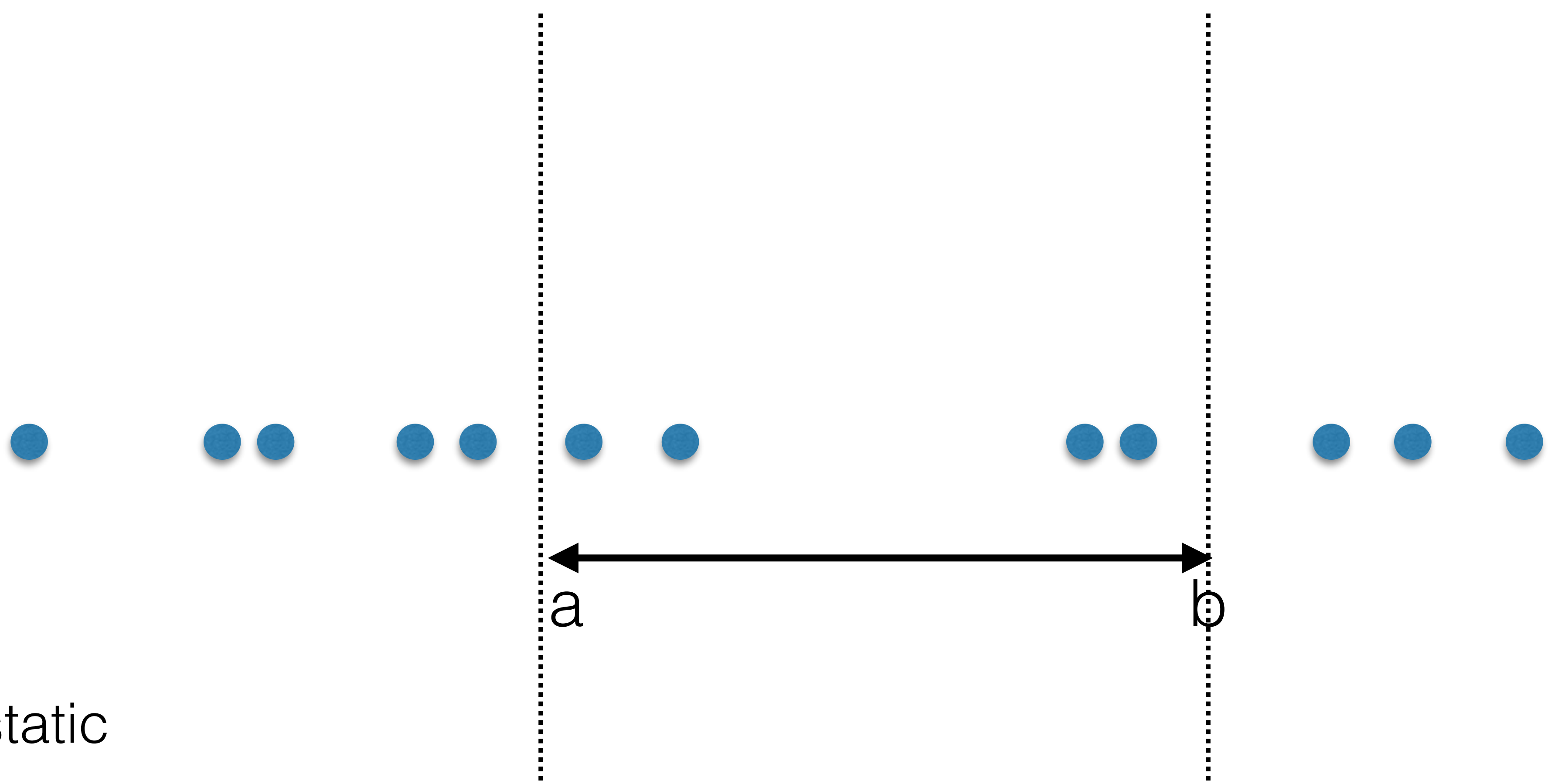
- If P is static

P is known ahead
and
does not change

1D Range Searching

- Given a set of values P = $\{x_1, x_2, x3, \ldots x_n\}$
- Pre-process it in order to answer

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

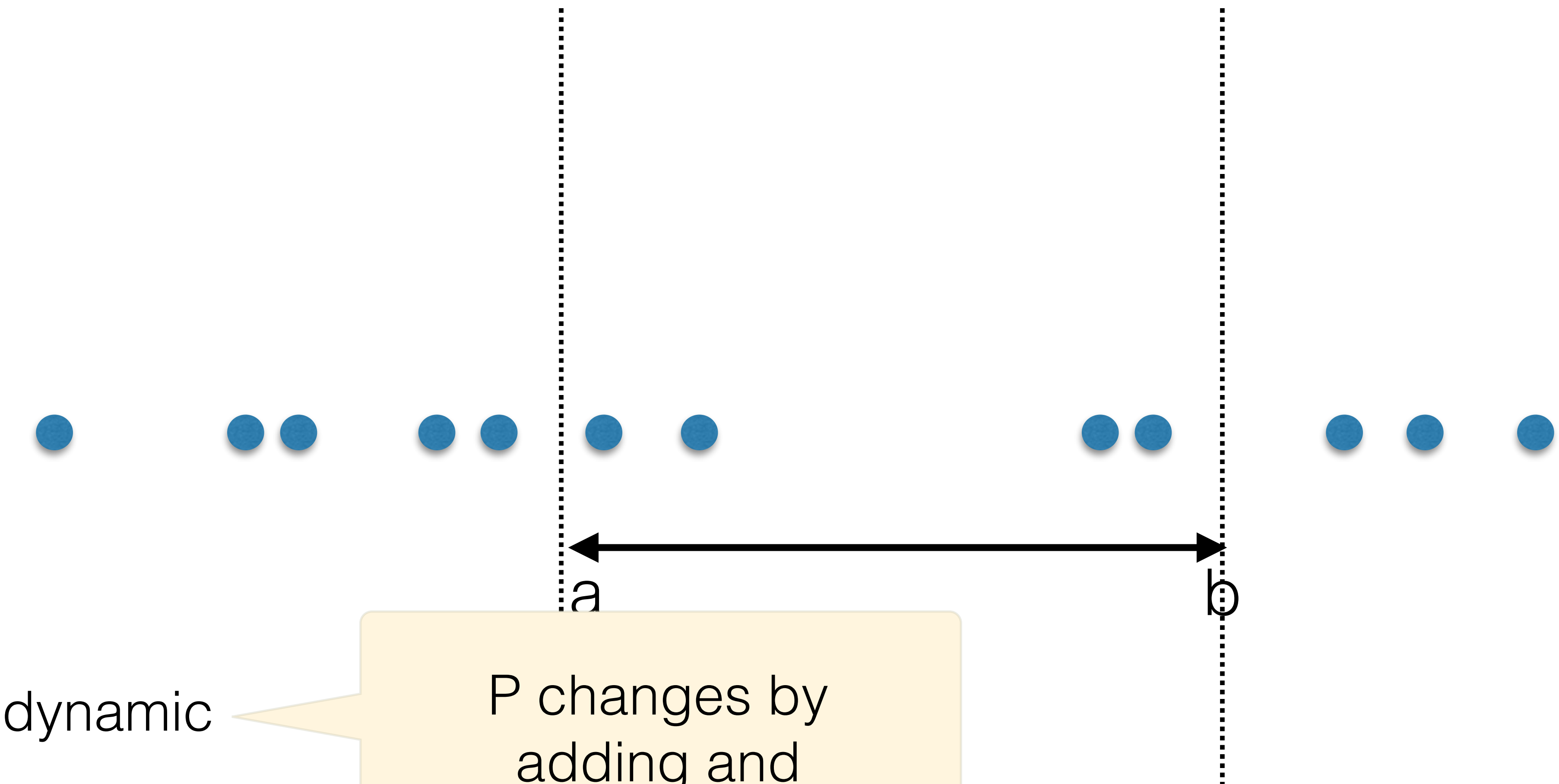


- If P is static
    - sort, then binary search for a and walk. O(lg n + k) per query

1D Range Searching

- Given a set of values $P = \{x_1, x_2, x3, \ldots x_n\}$
- Pre-process it in order to answer

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

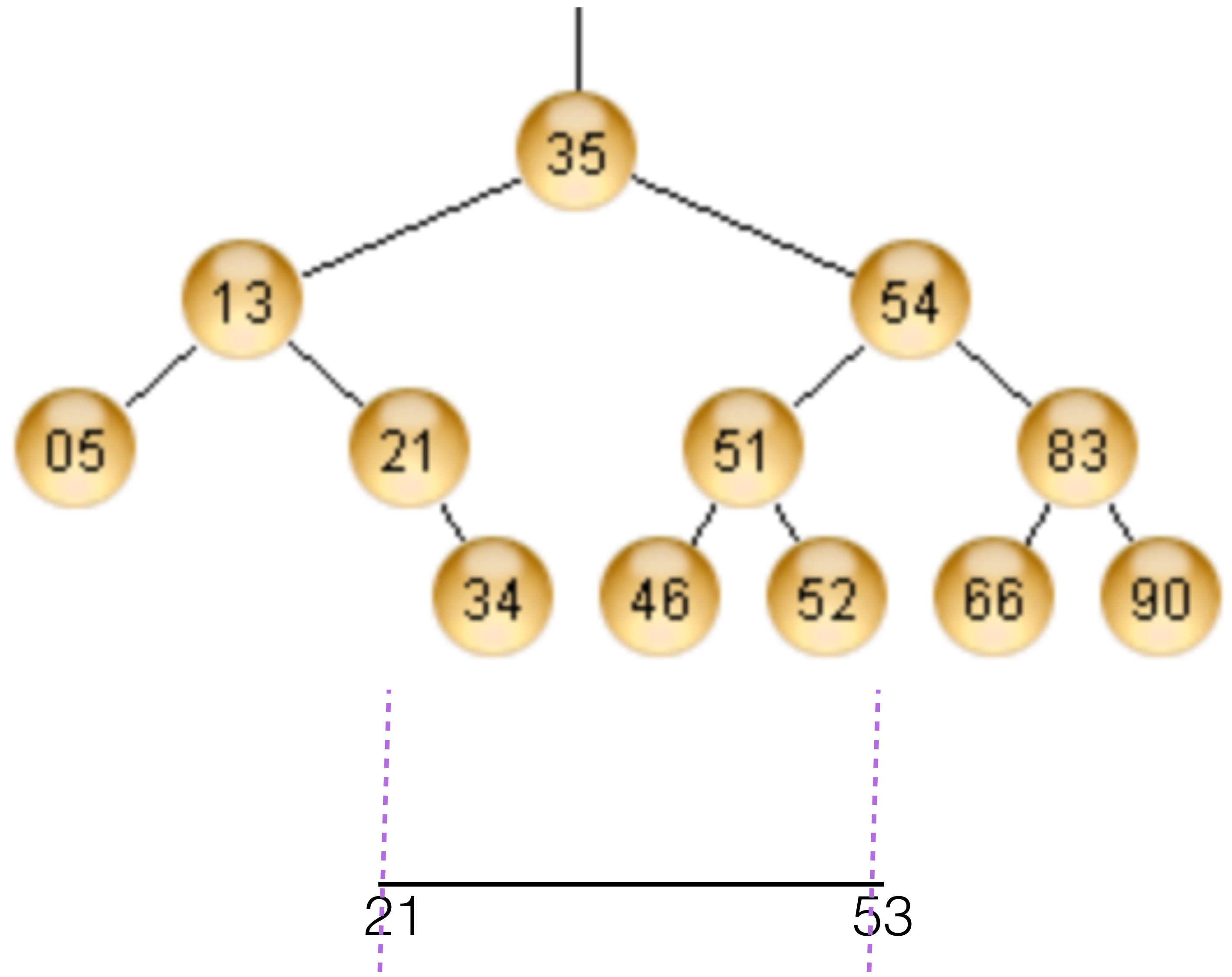


- If P is dynamic
    - use a BBST

P changes by adding and deleting values

1D range searching with Binary Search Trees

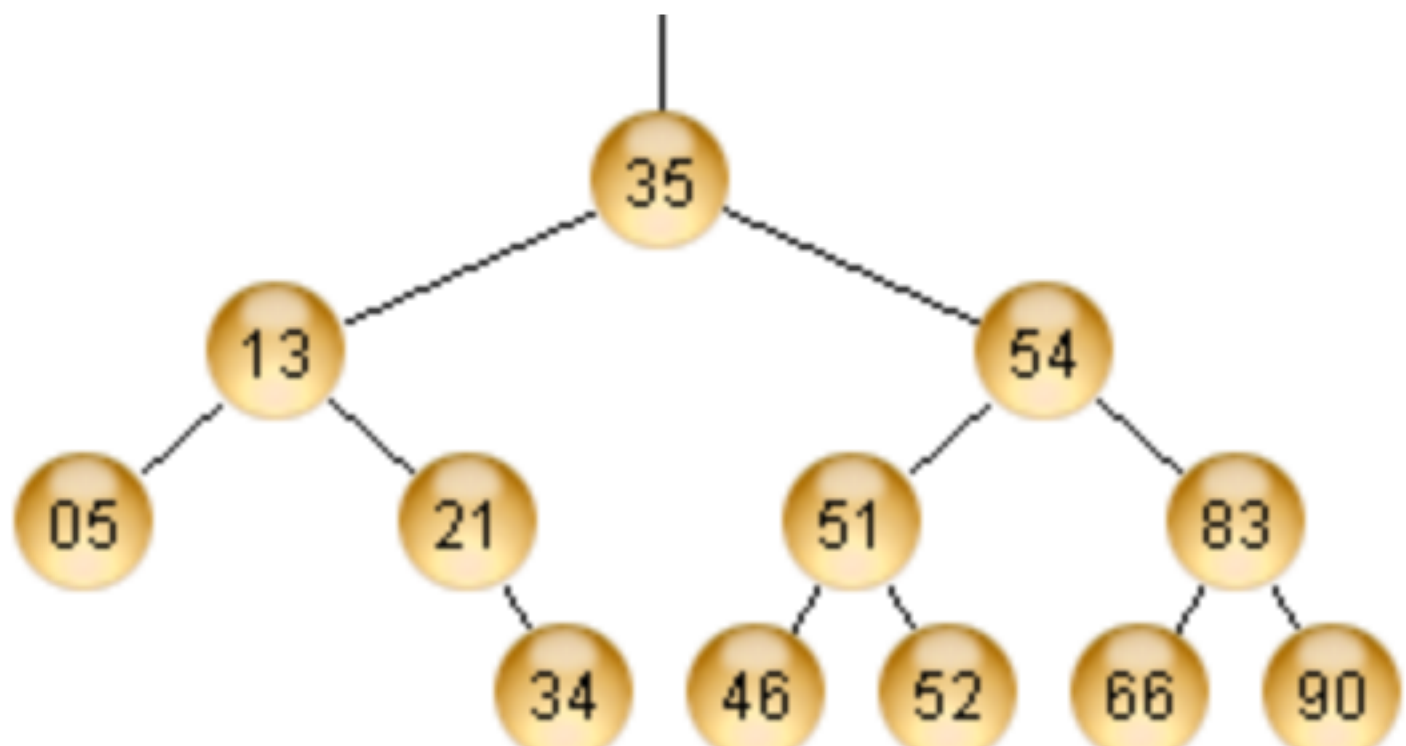
Example: range_search(21, 53): return 21, 34, 35, 46, 51, 52



Balanced Binary Search Trees
- crash course -

Binary Search Trees (BST)

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

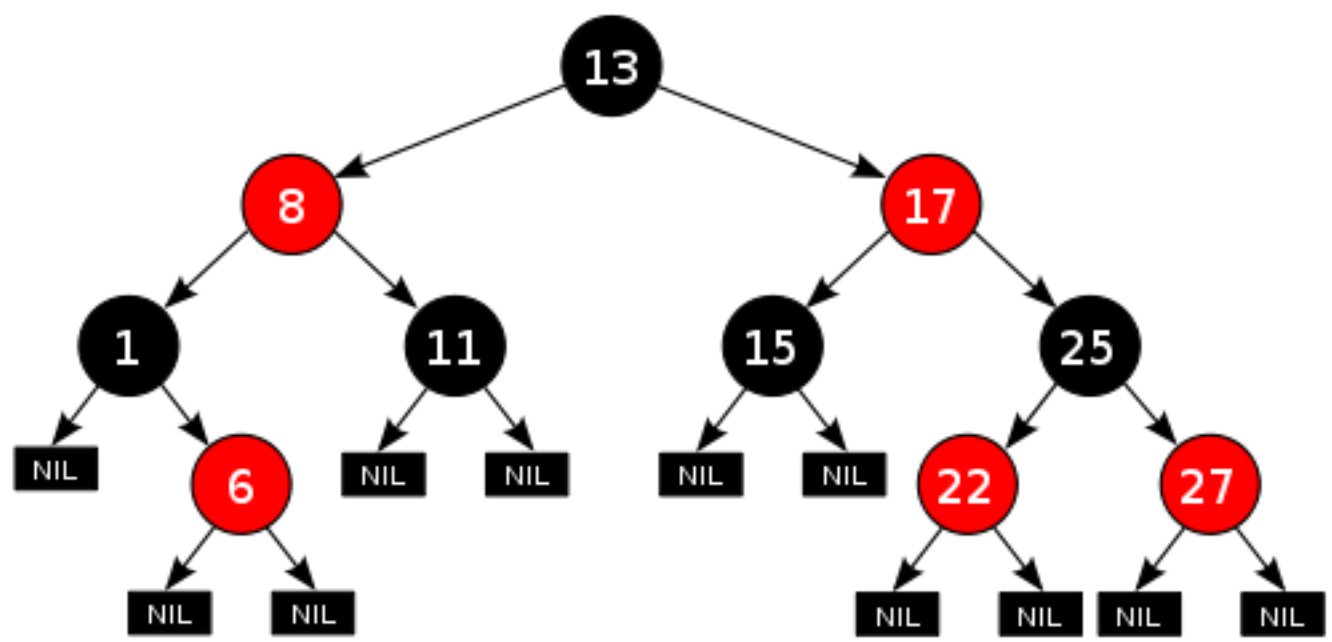


Balanced Binary Search Trees (BBST)

- Binary search trees + invariants that constrain the tree to be balanced
    - $h = O(\lg n)$
- These invariants have to be maintained when inserting and deleting
    - 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

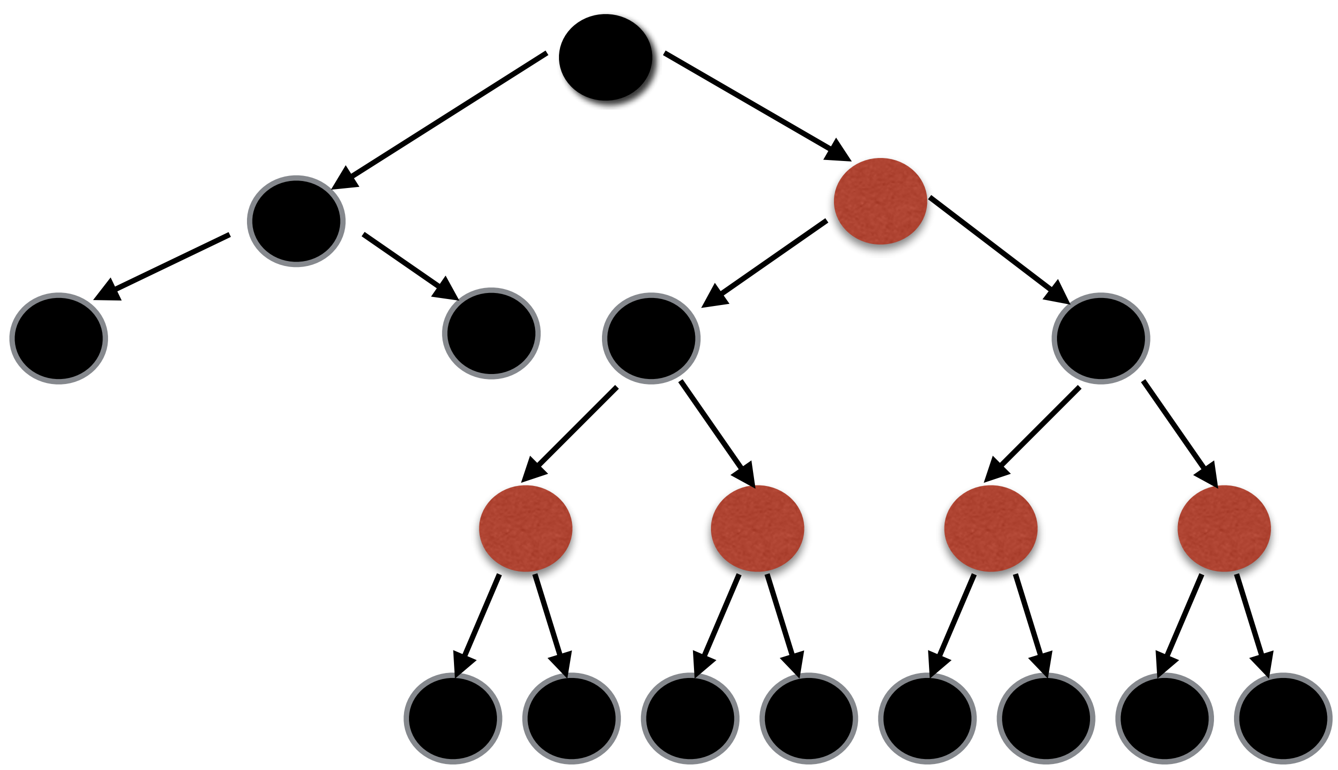
- Binary search tree, and
    - Each node is Red or Black
    - The children of a Red node must be Black
    - The number of Black nodes on any path from the root to any node that does not have two children must be the same



- 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

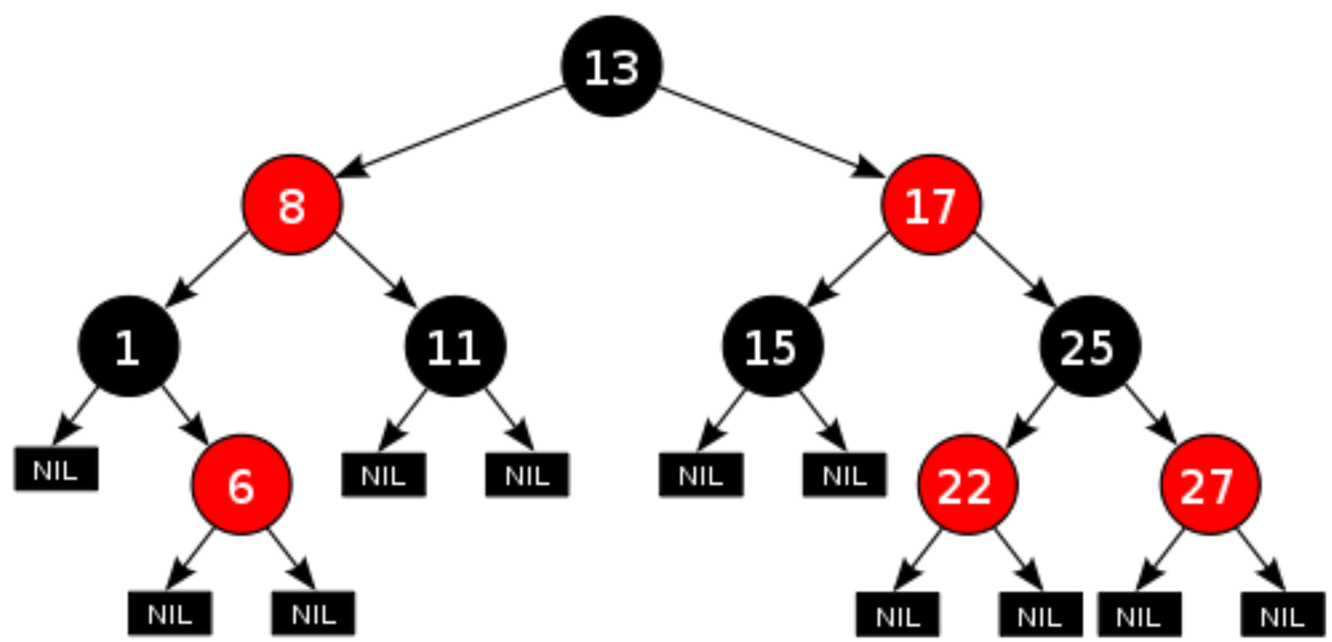
Example: Red-Black trees

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



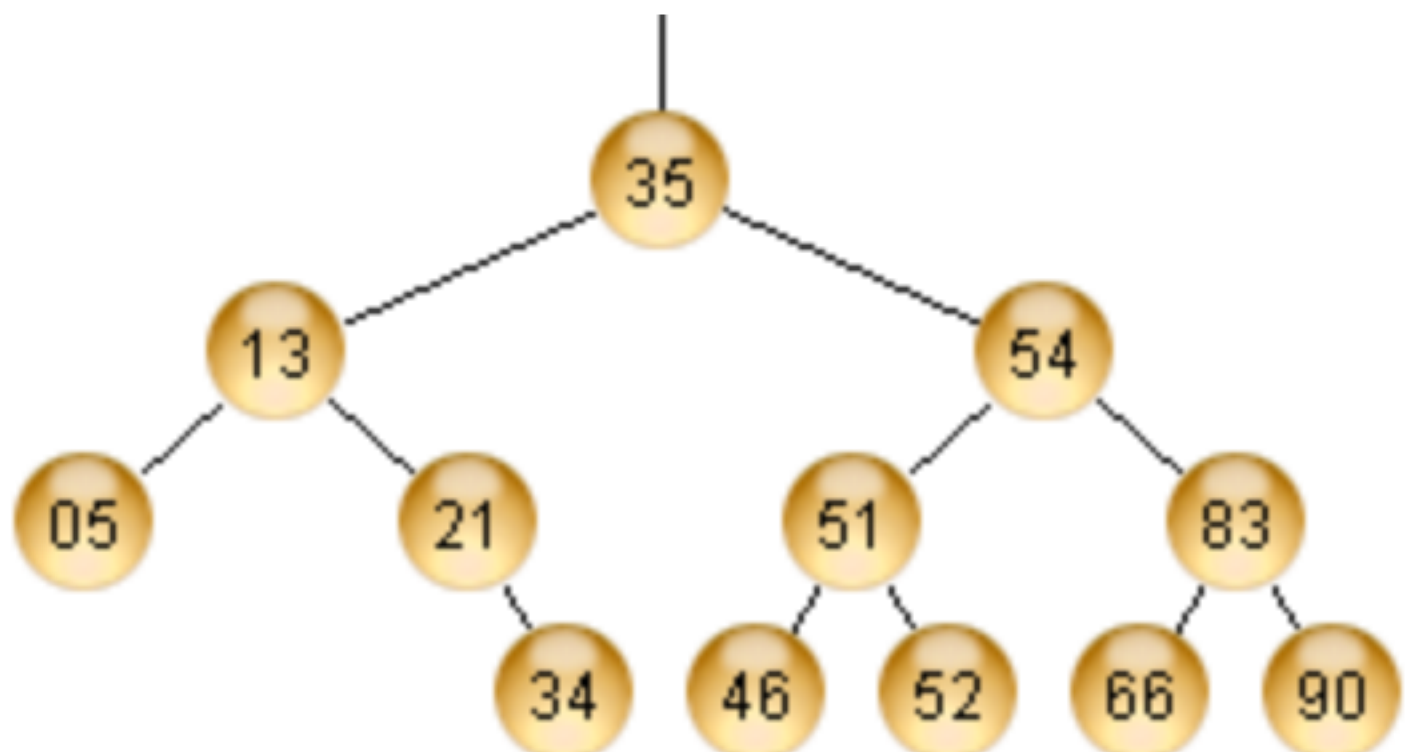
Example: Red-Black trees

- 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 nodes on the path from the inserted/deleted node to the root.



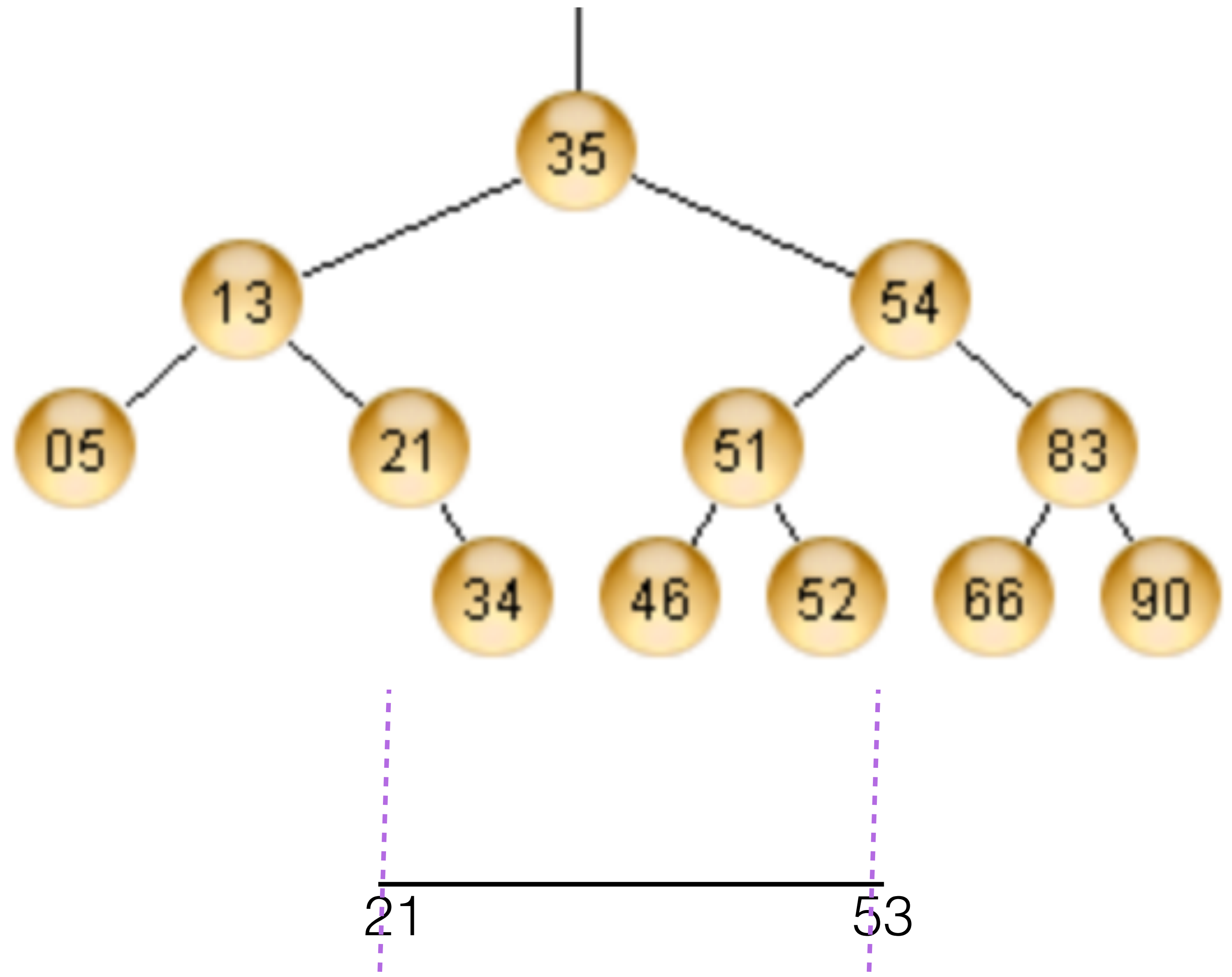
Binary Search Trees

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



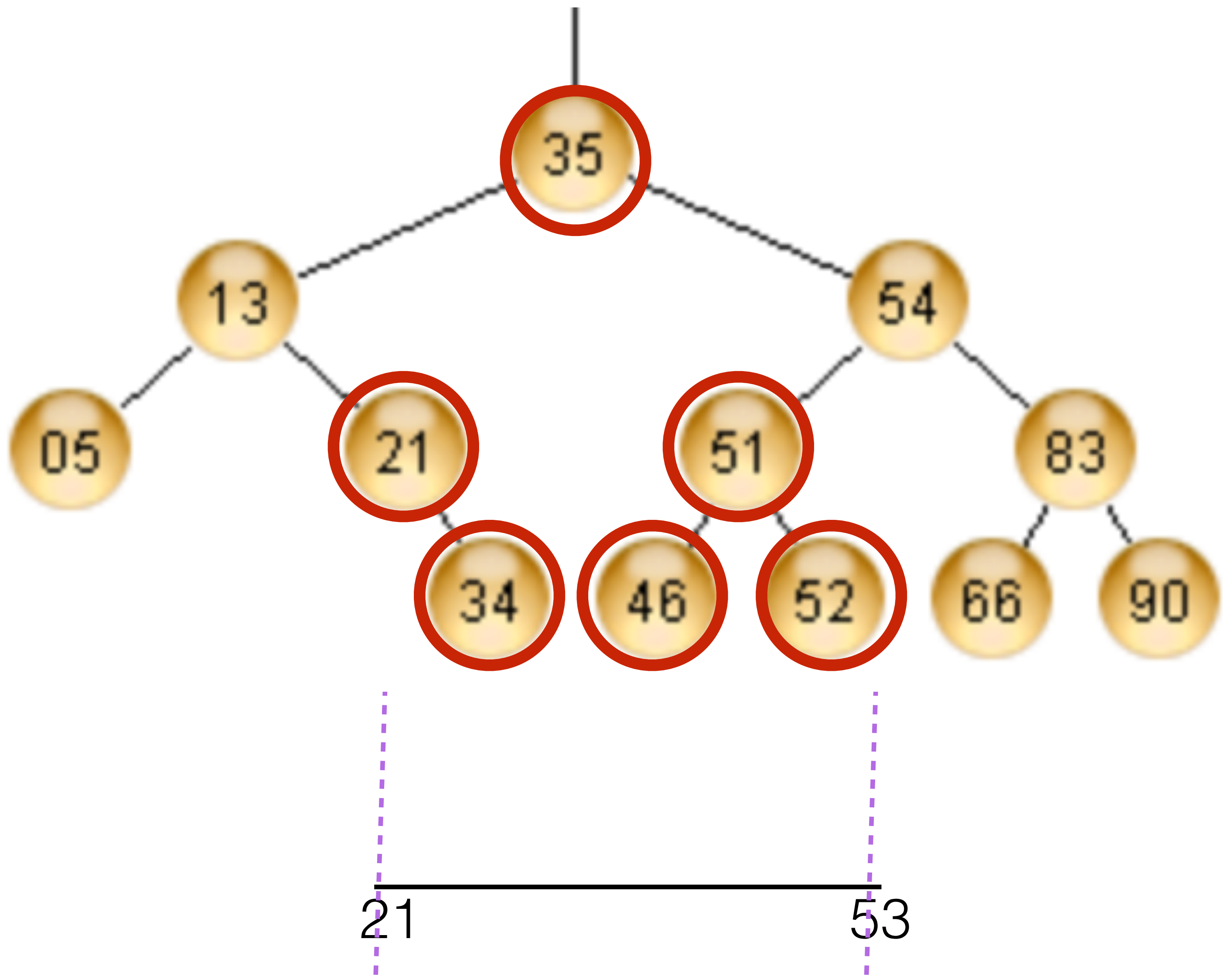
1D range searching with Binary Search Trees

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



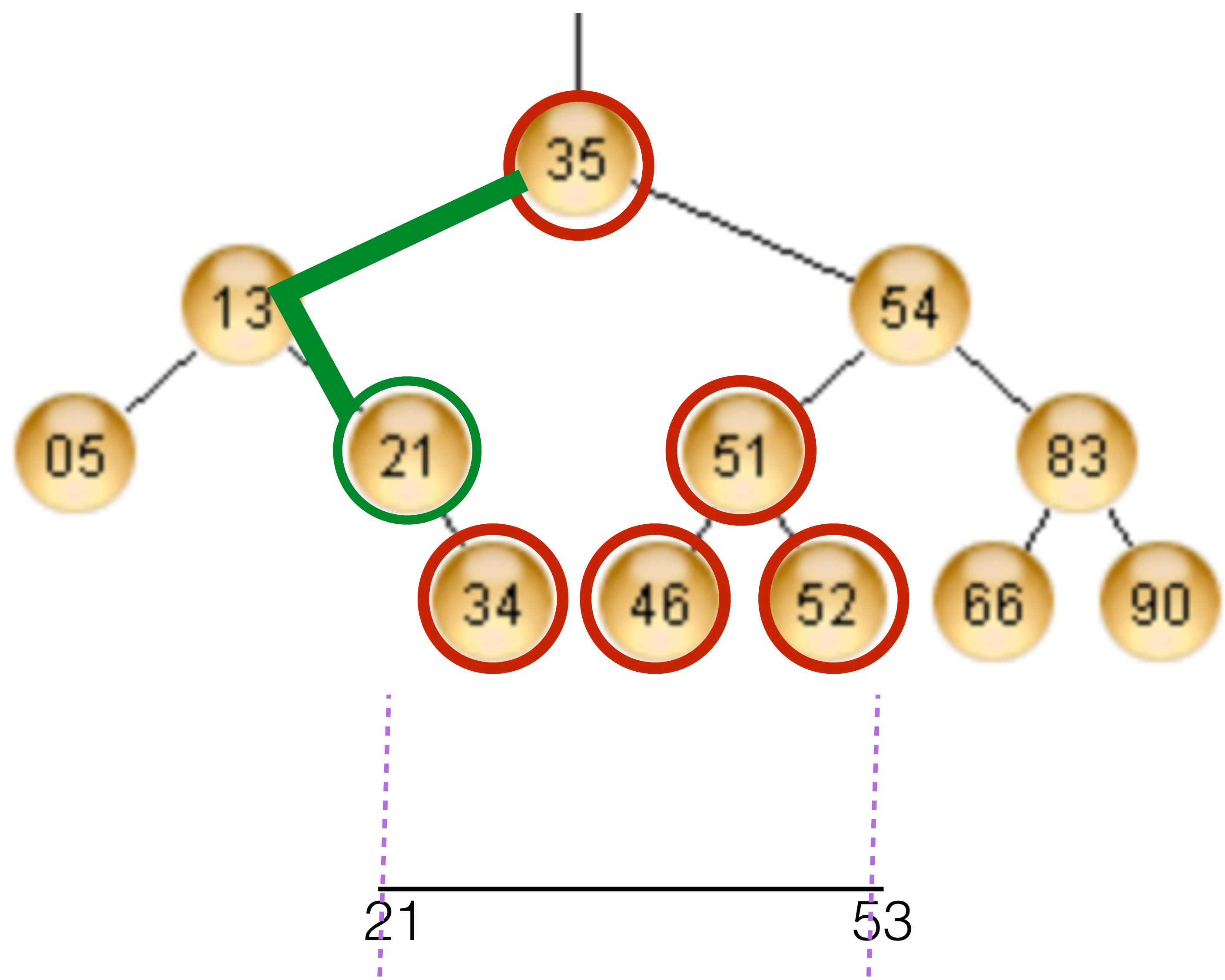
1D range searching with Binary Search Trees

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



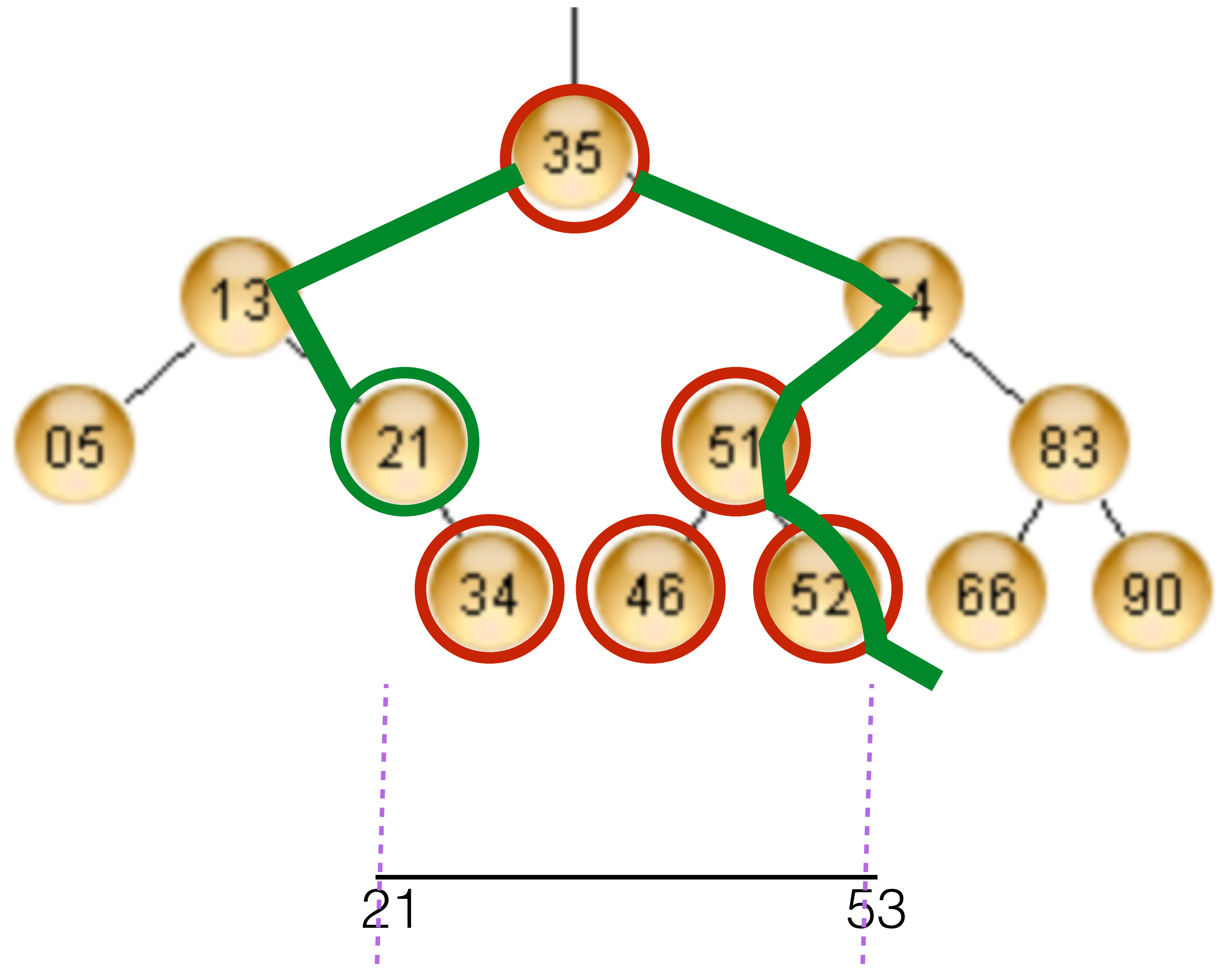
1D range searching with Binary Search Trees

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



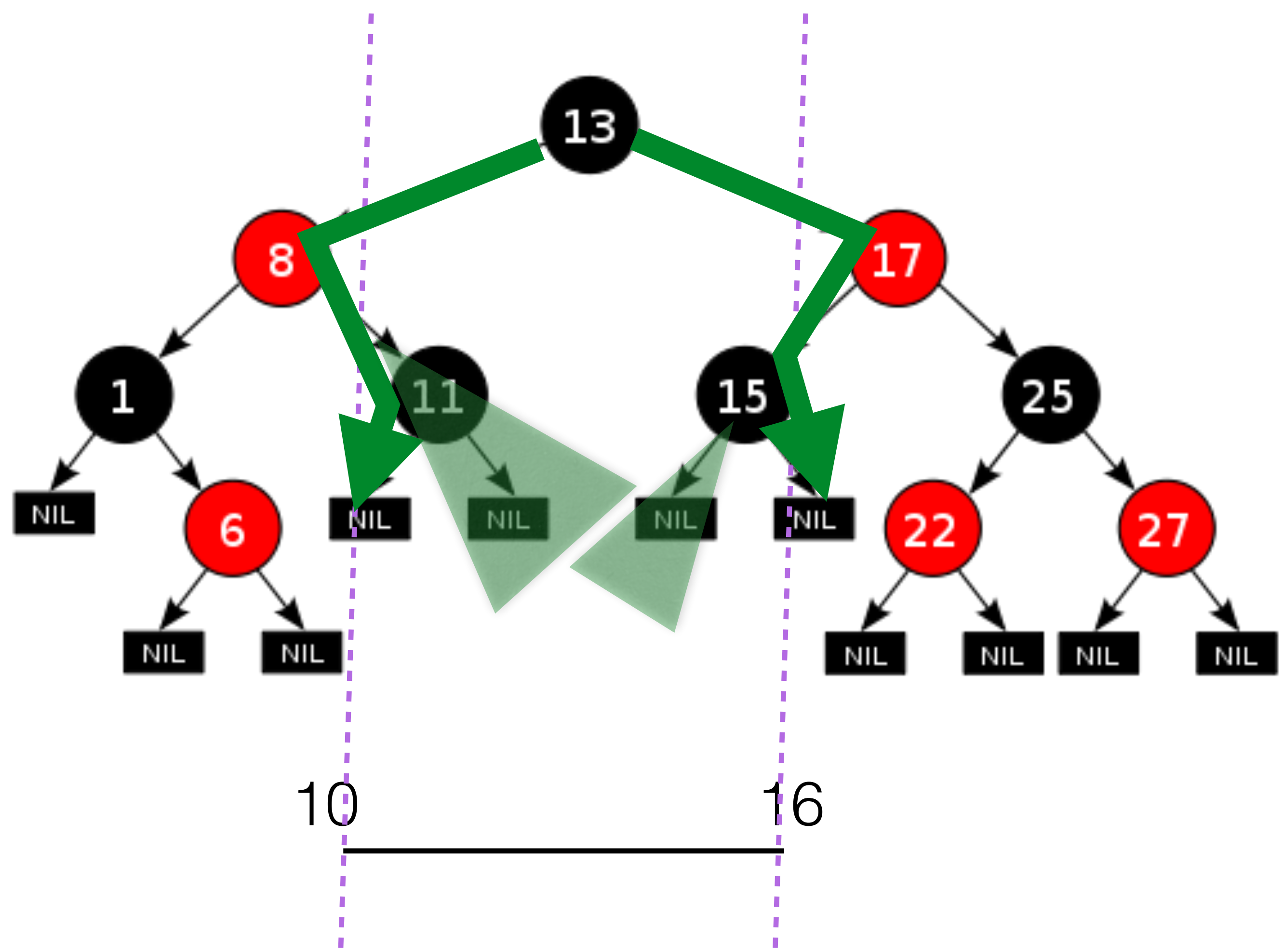
1D range searching with Binary Search Trees

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



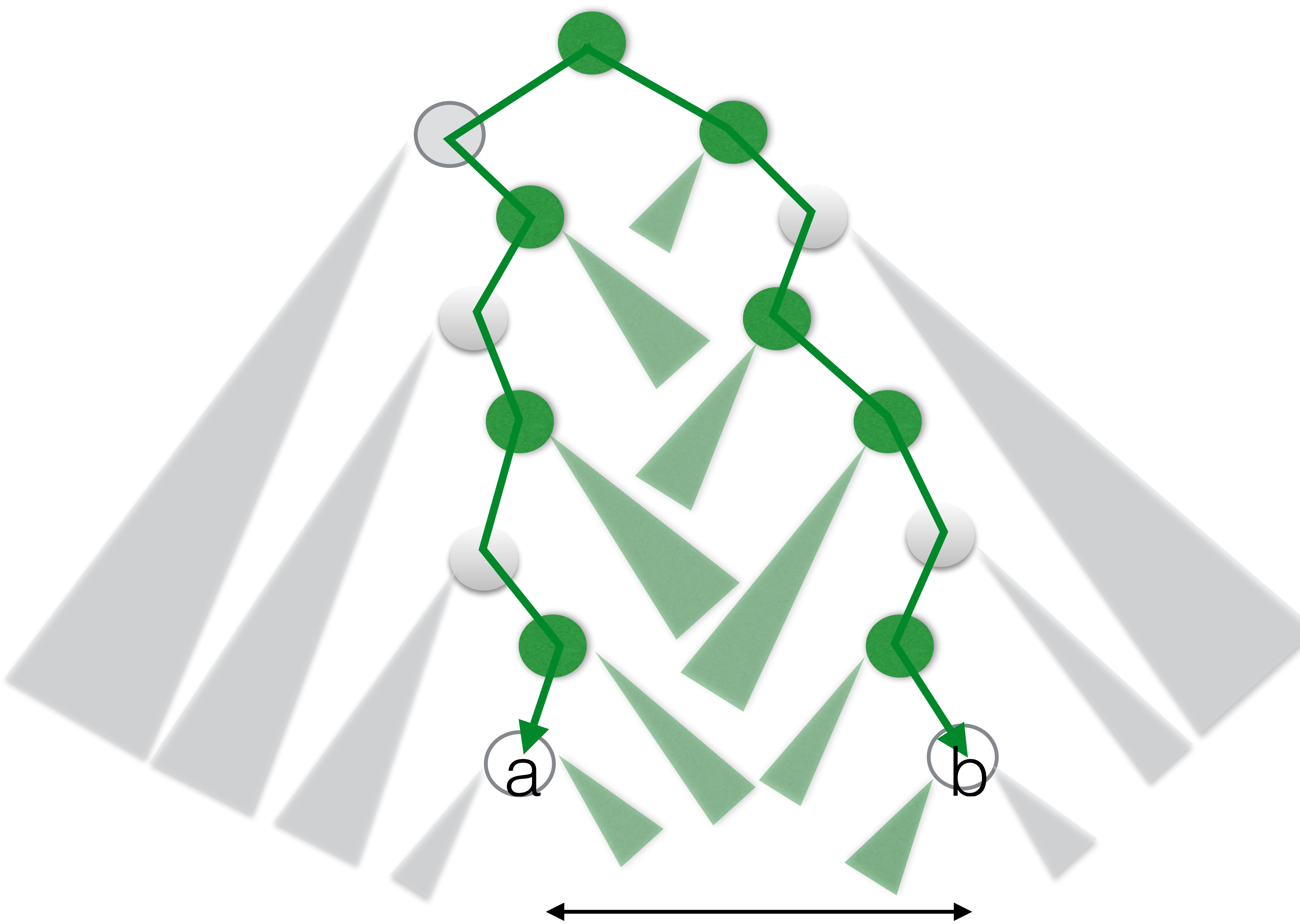
1D Range Searching with Red-Black Trees

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



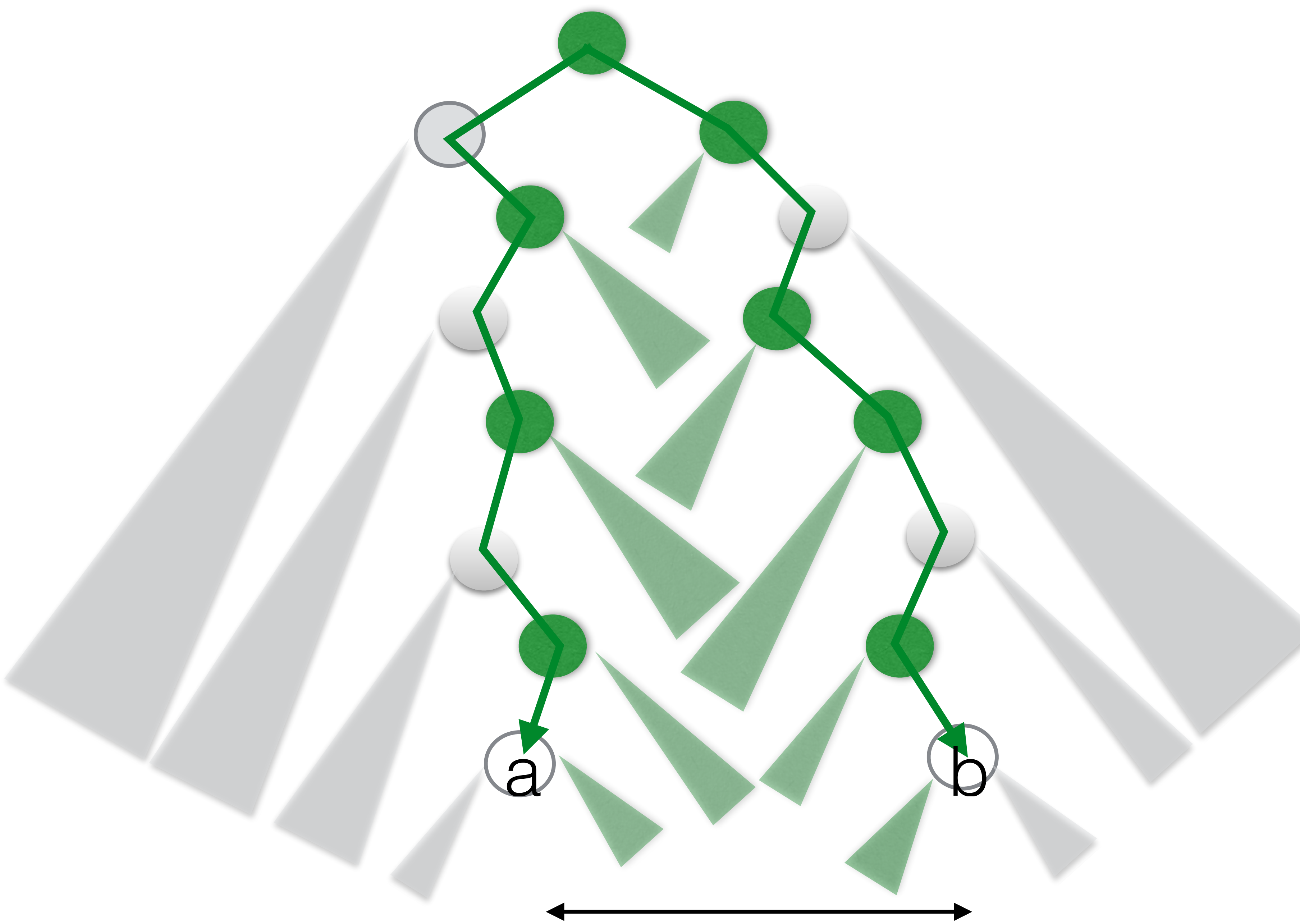
1D range searching with Binary Search Trees

- Range search (a,b):



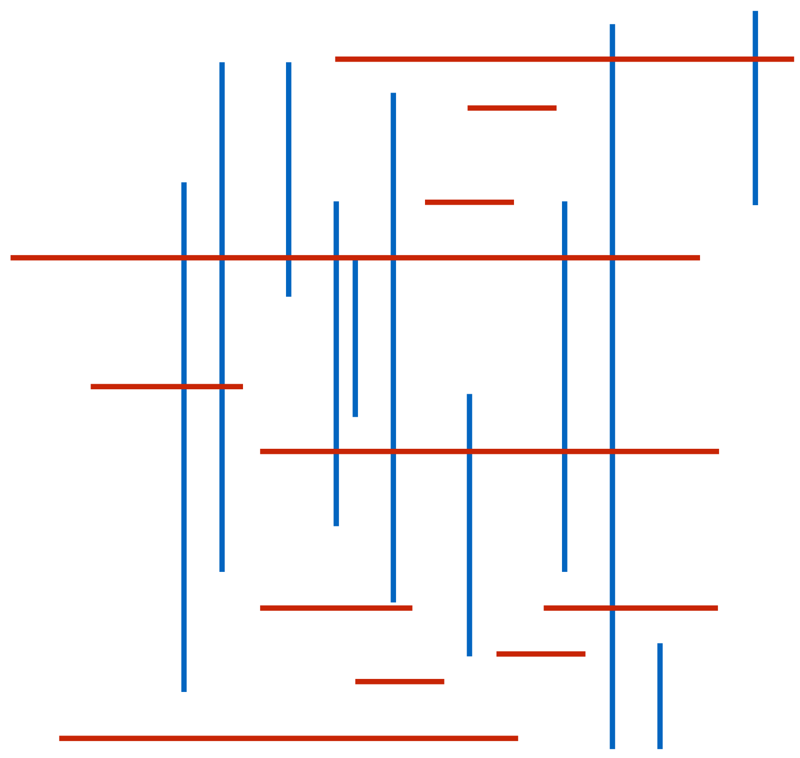
1D range searching with Binary Search Trees

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

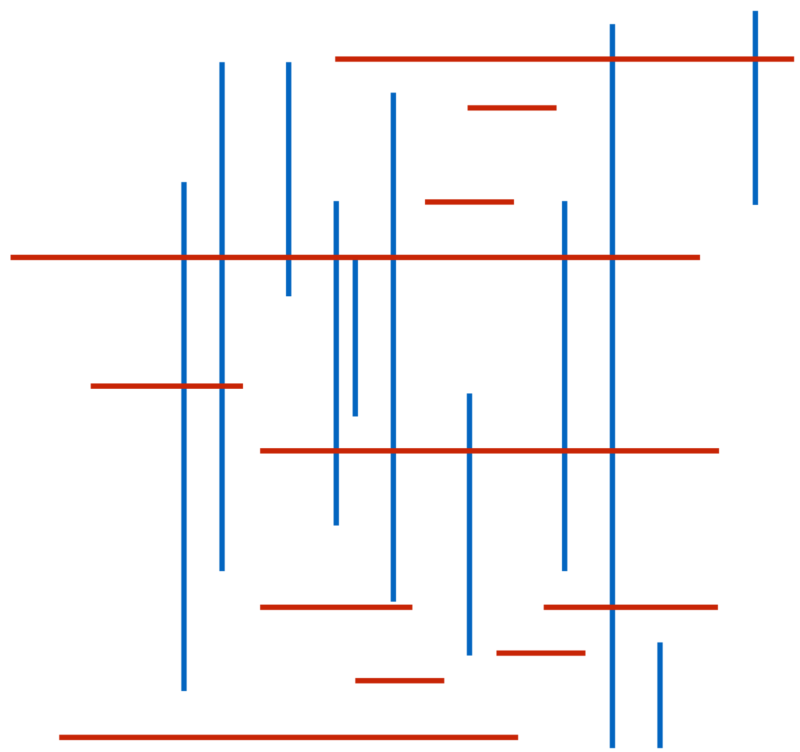


Balanced Binary Search Trees
- end -

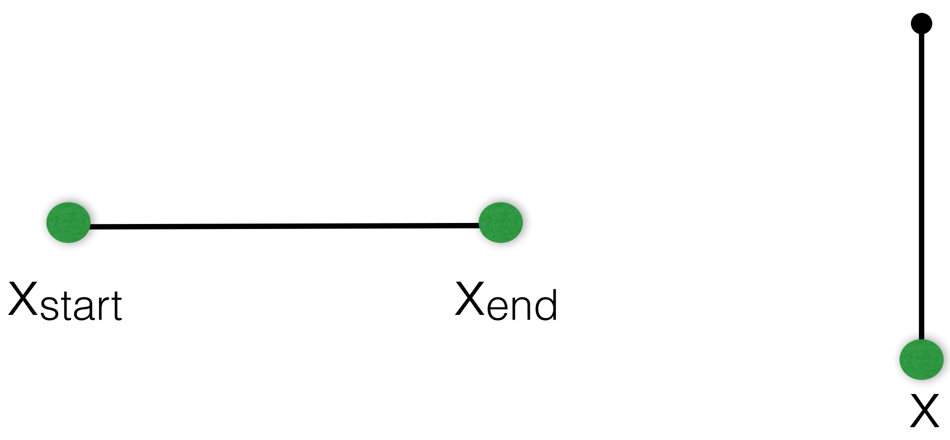
The line sweep technique



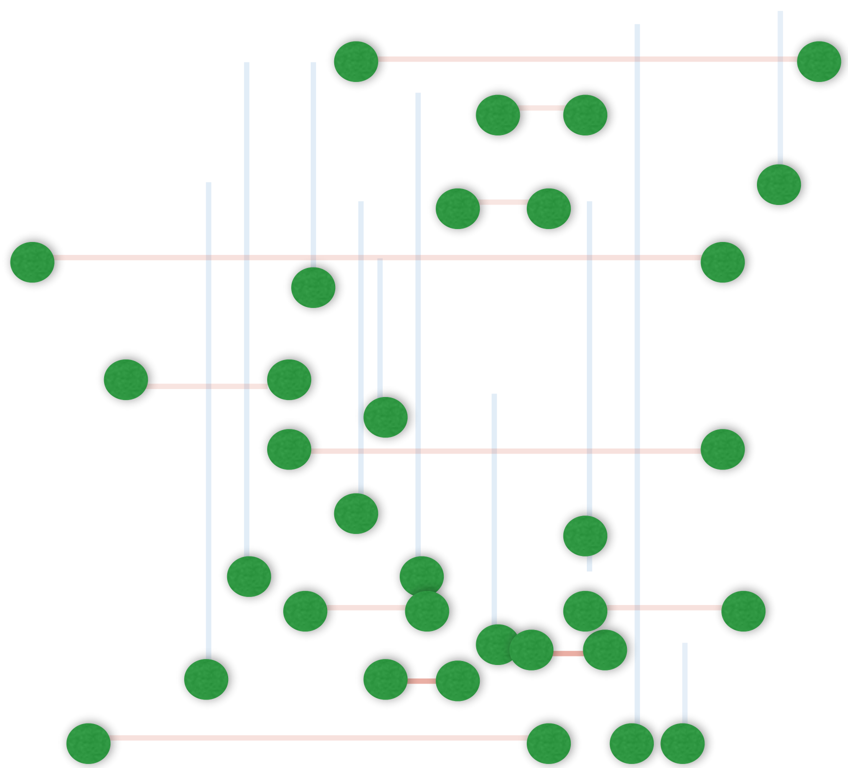
The line sweep technique



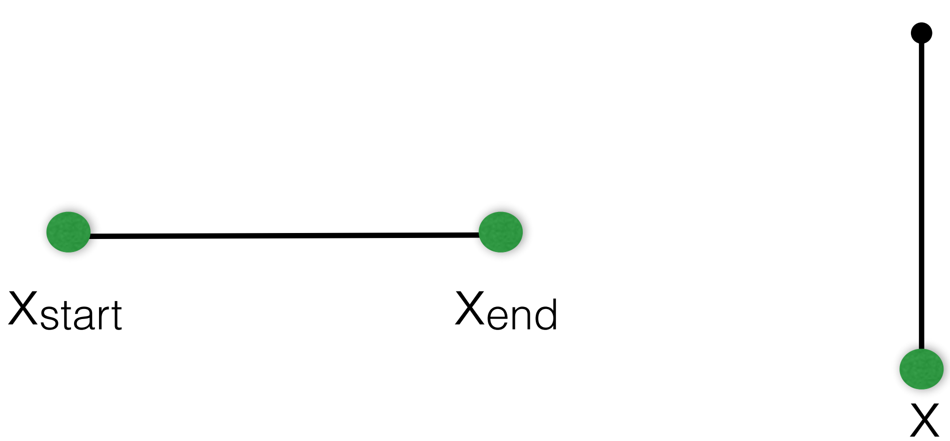
- Let X be the set of x-coordinates of all segments: these are the "events"



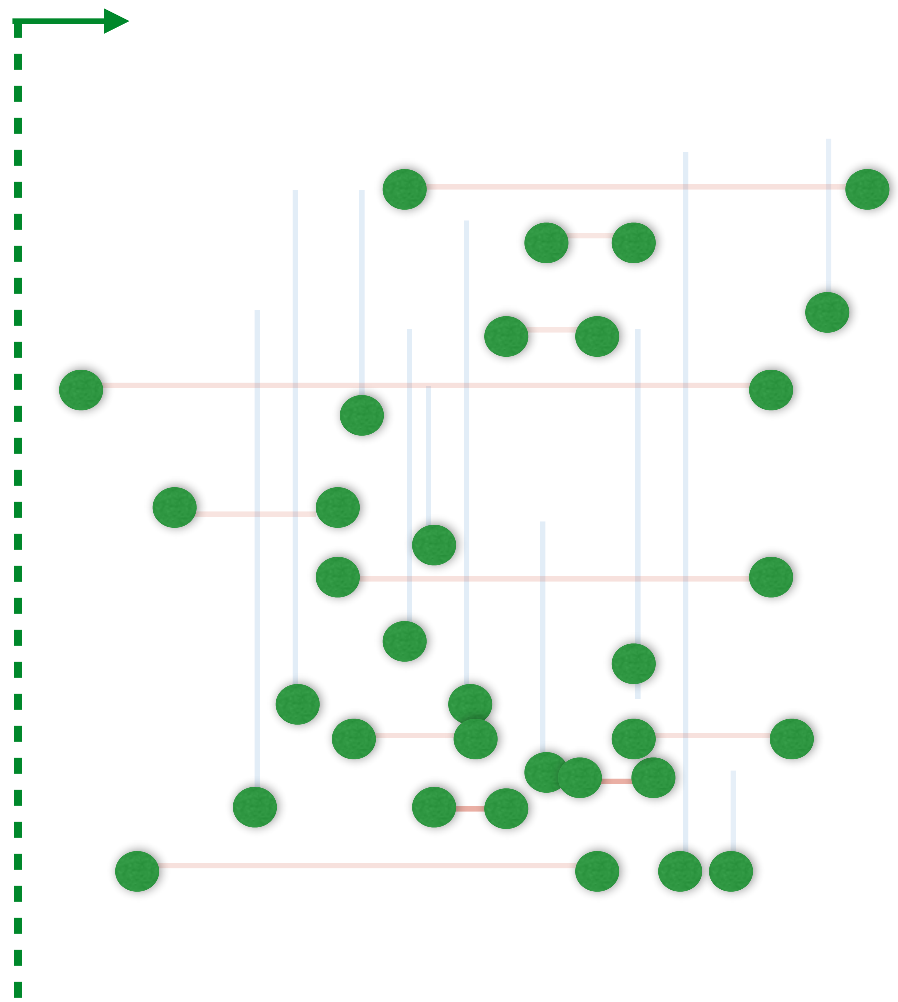
The line sweep technique



- Let X be the set of x-coordinates of all segments: these are the “events”

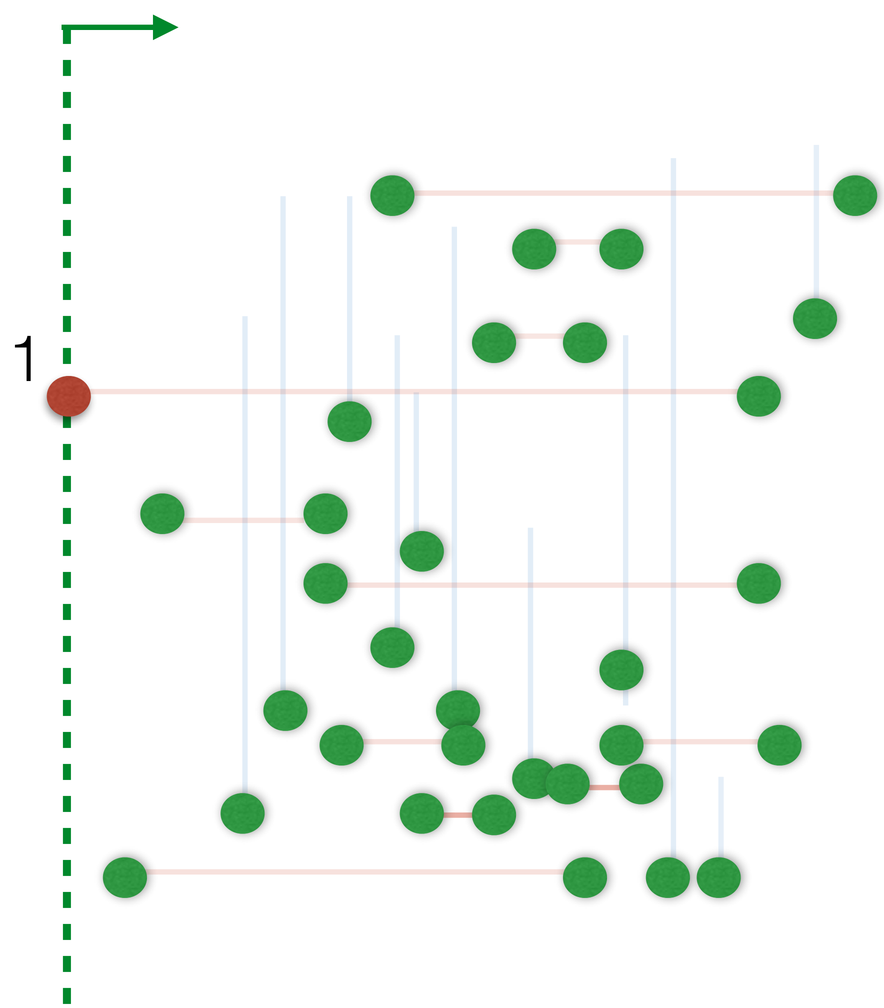


The line sweep technique



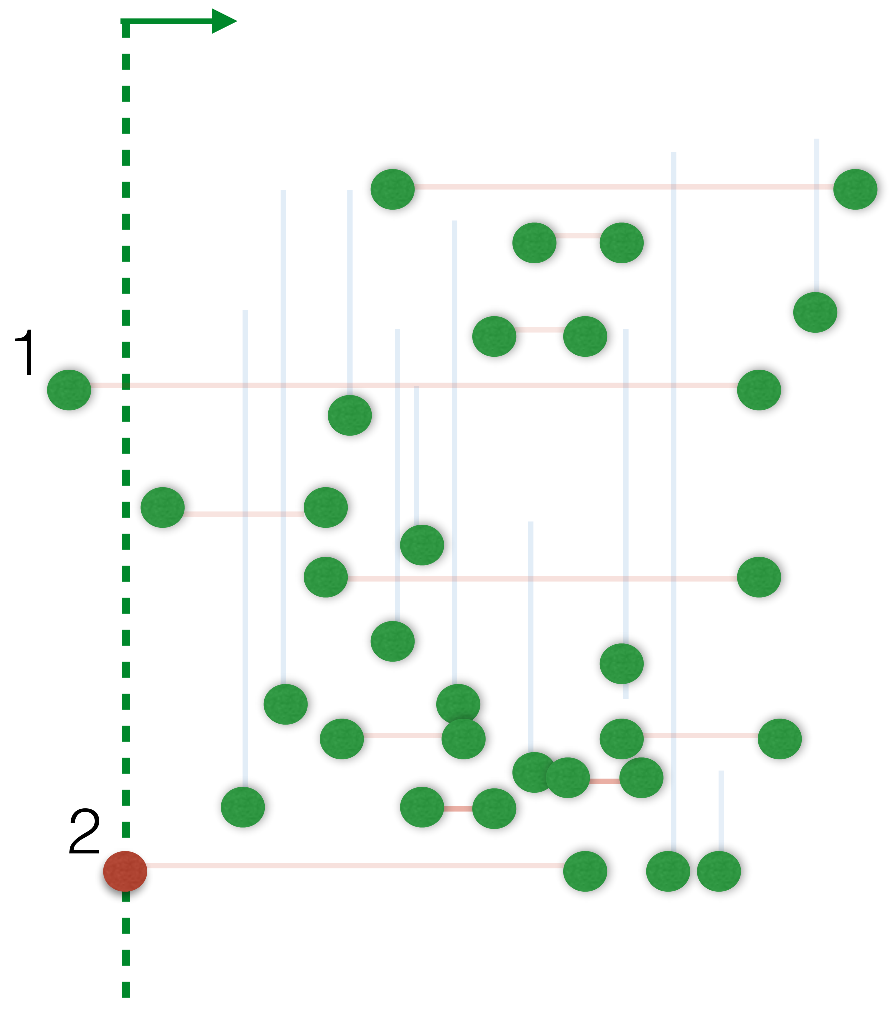
- Traverse the **events** in order

The line sweep technique



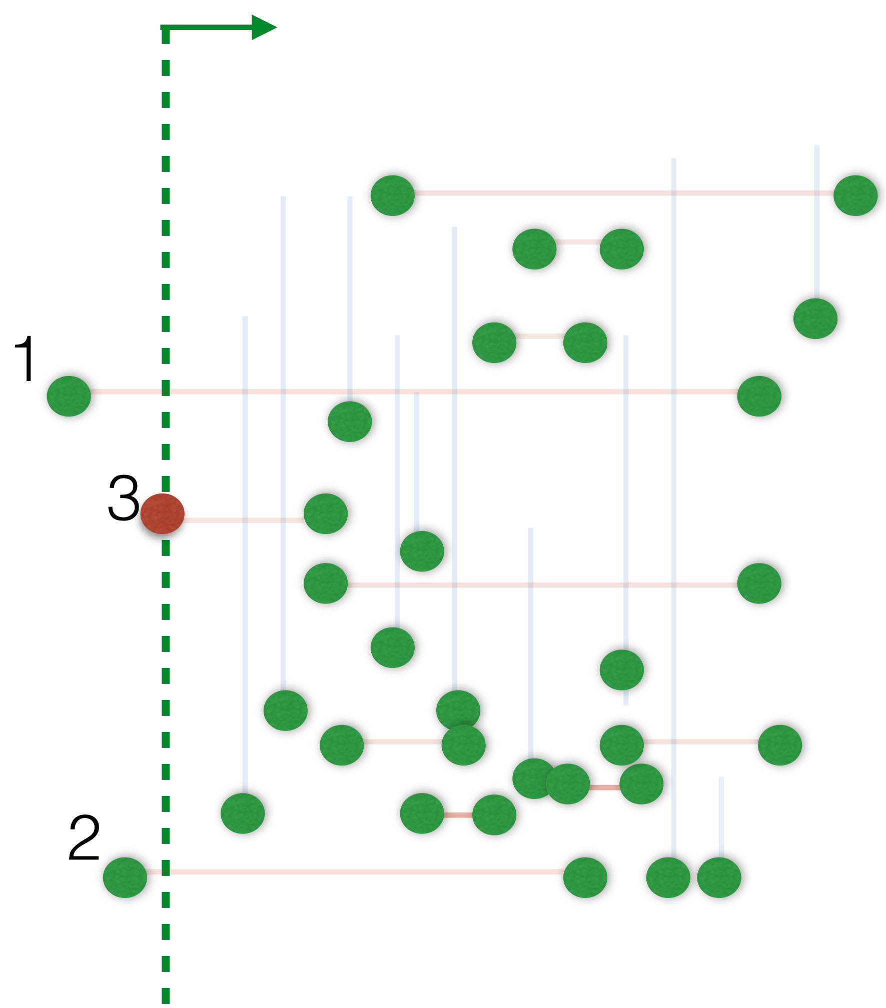
- Traverse the **events** in order

The line sweep technique



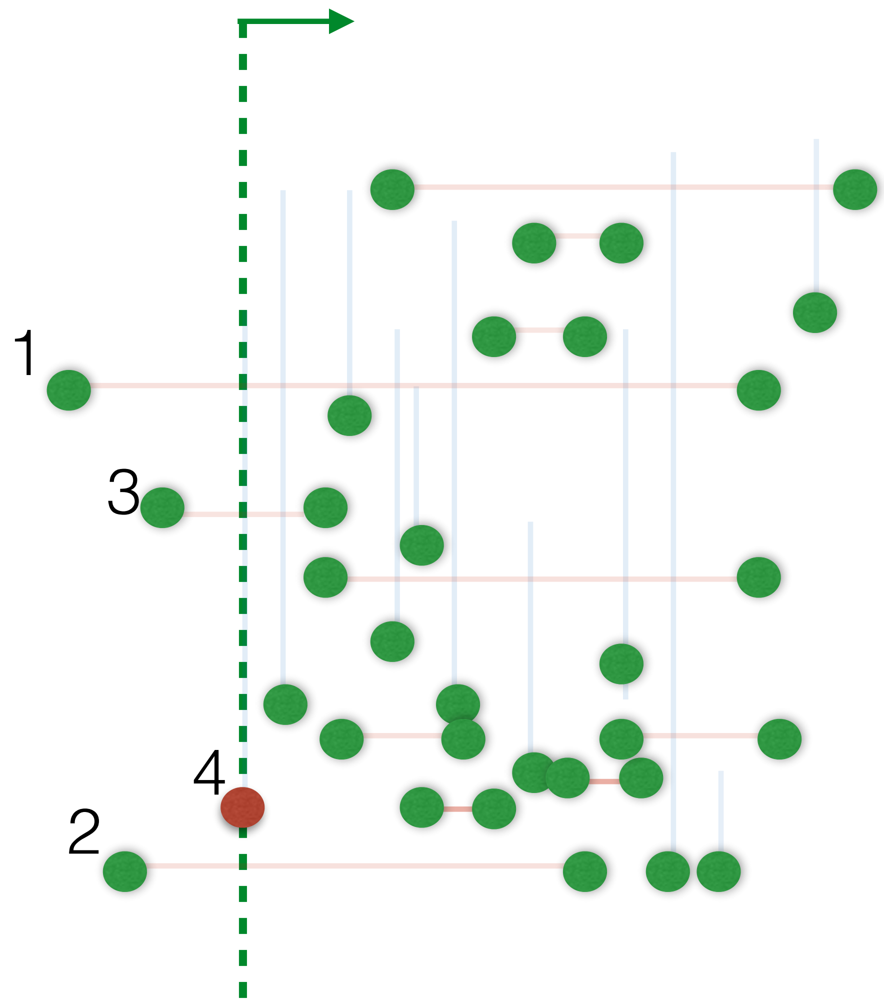
- Traverse the **events** in order

The line sweep technique



- Traverse the **events** in order

The line sweep technique

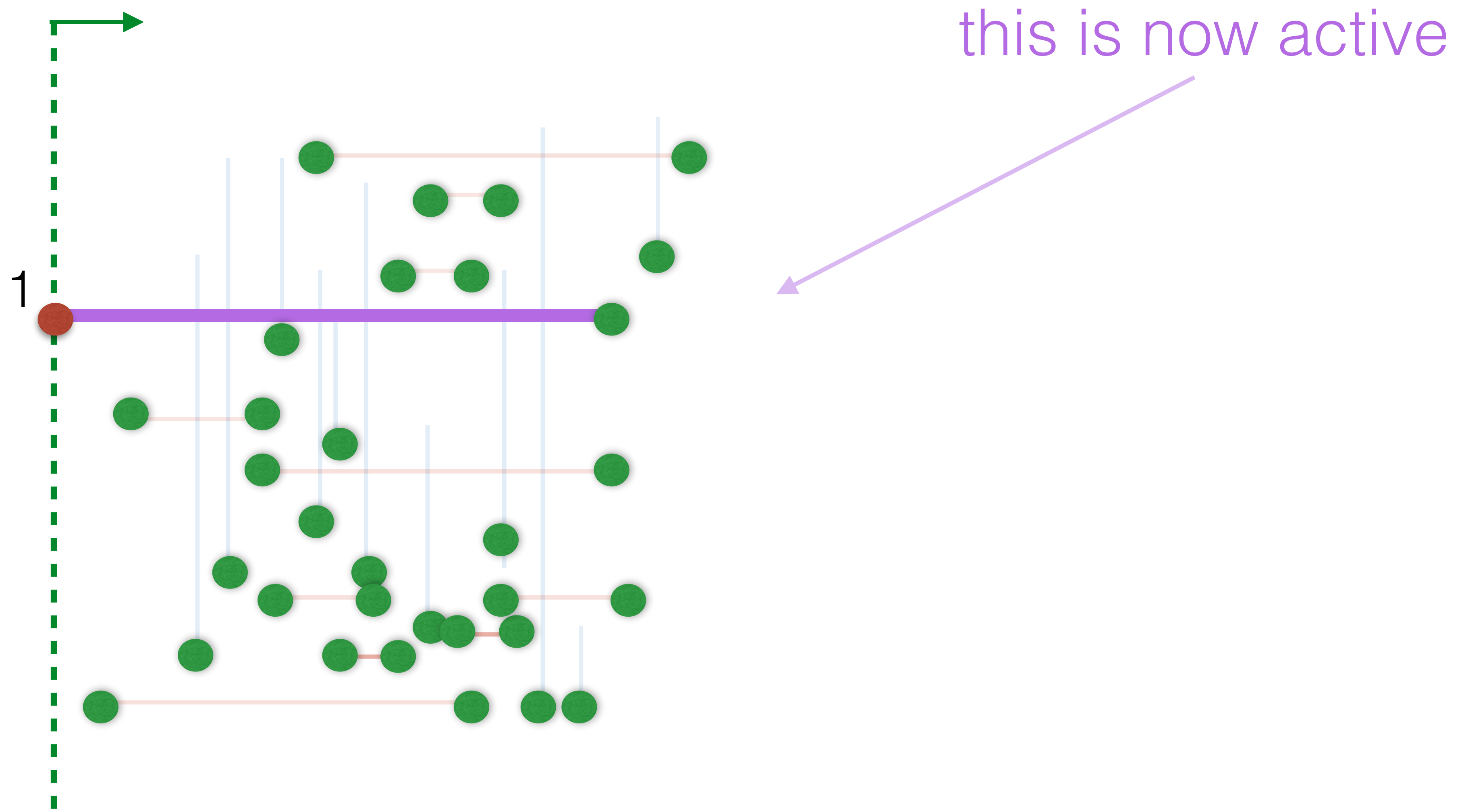


And so on.

How to process events?

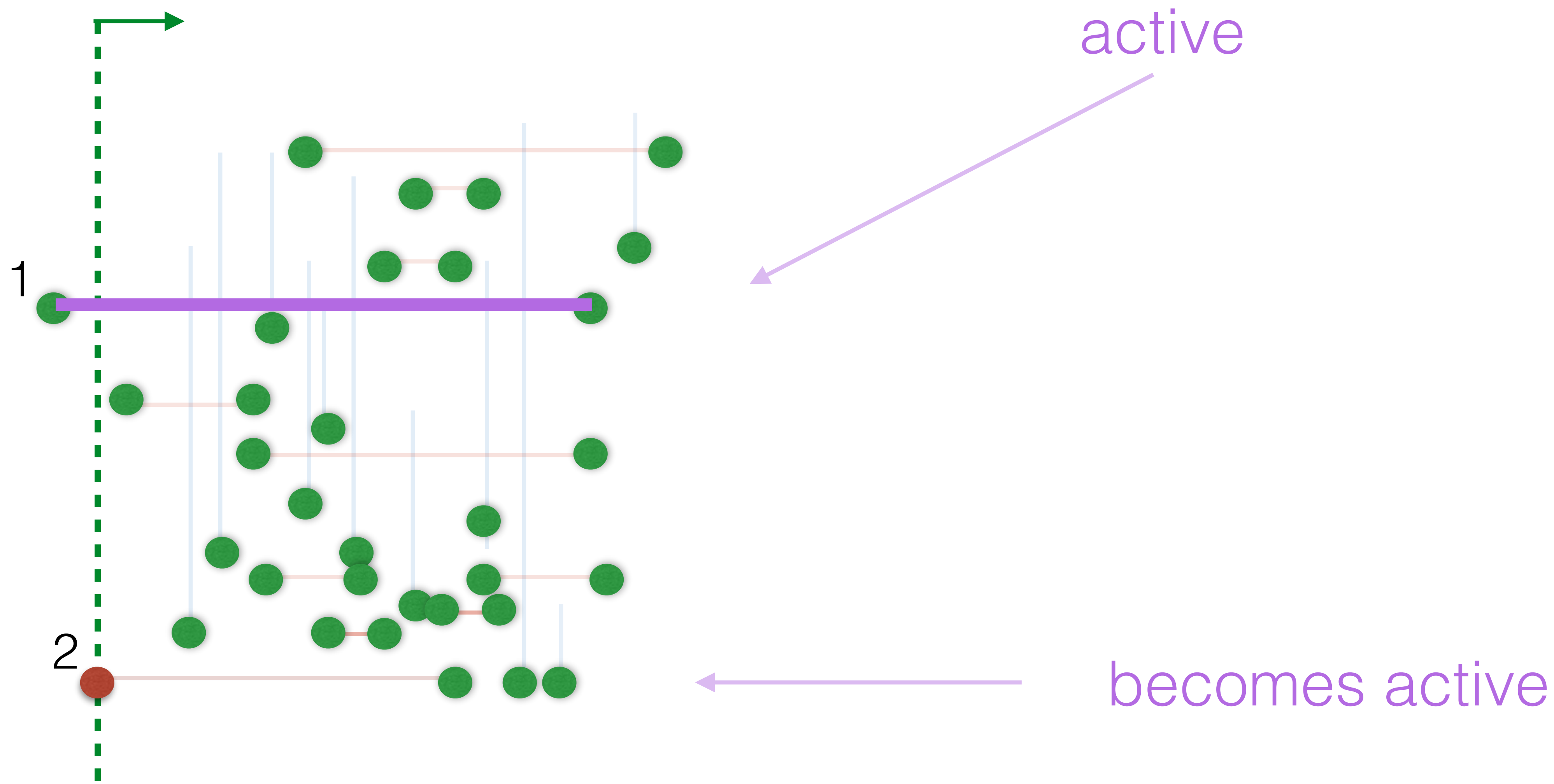
When reach a start event: segment becomes active

The line sweep technique



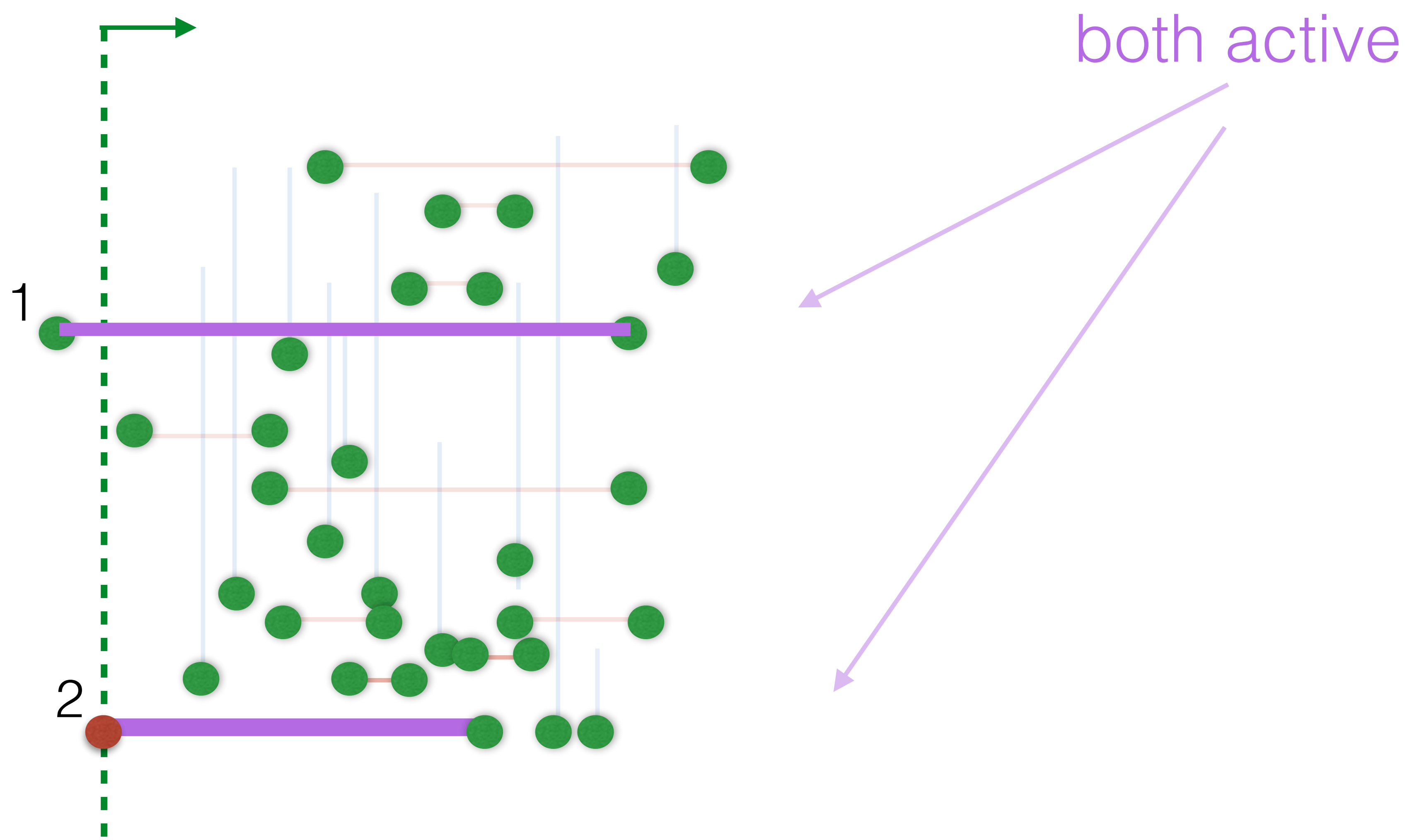
When reach a start event: segment becomes active

The line sweep technique



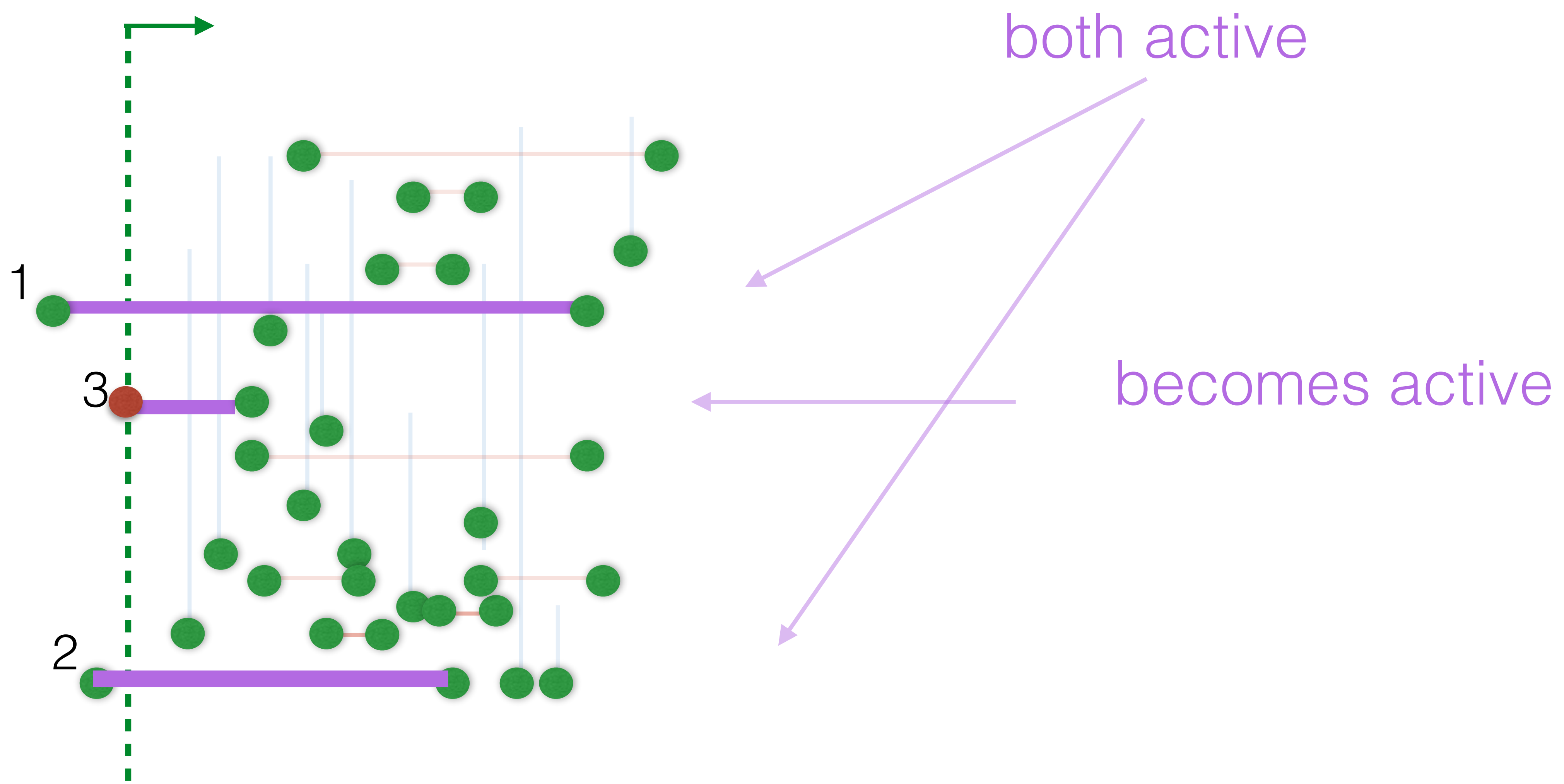
When reach a start event: segment becomes active

The line sweep technique



When reach a start event: segment becomes active

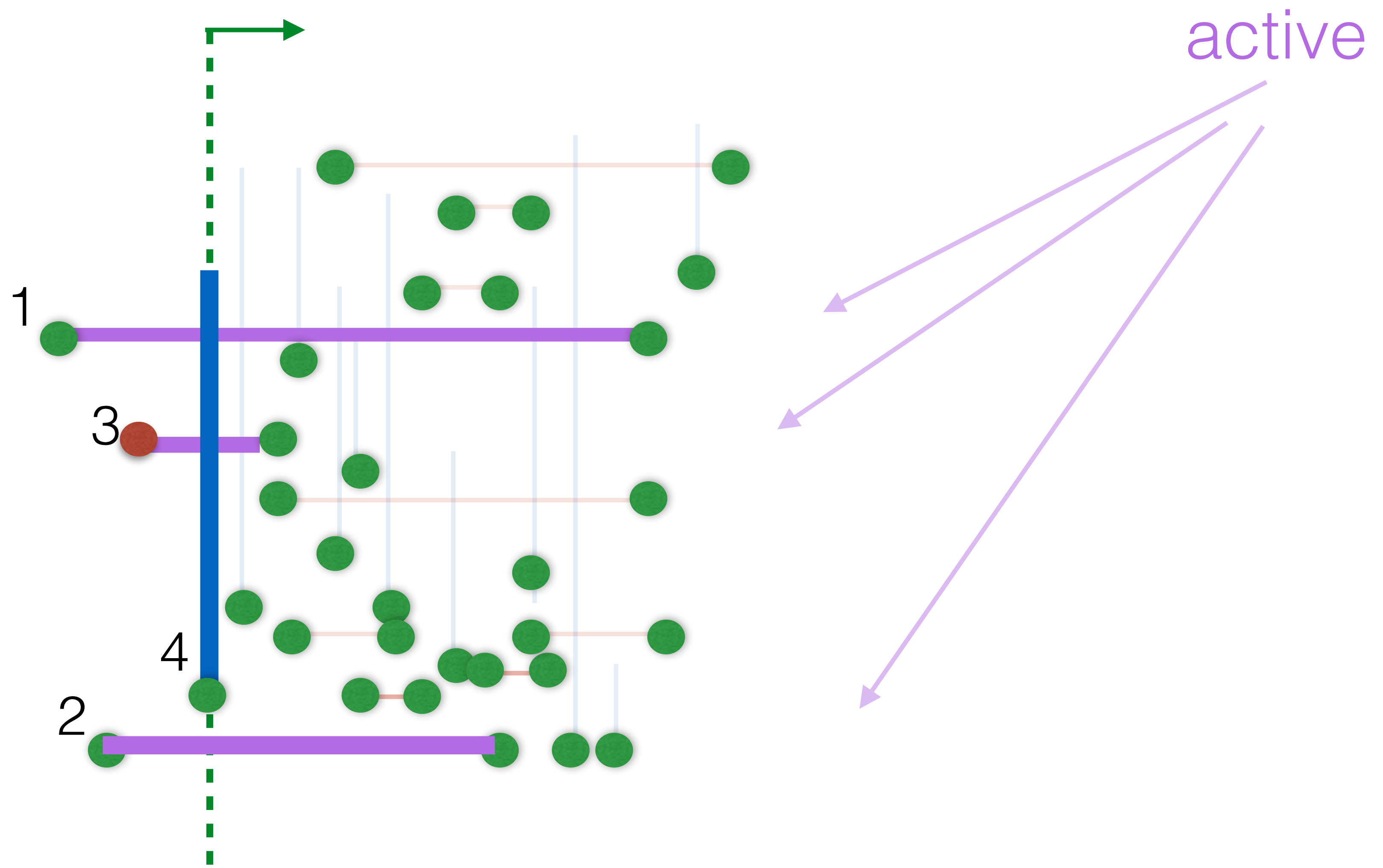
The line sweep technique



When reach a start event: segment becomes active

Let’s see what happens when we reach an event corresponding to a vertical segment

The line sweep technique

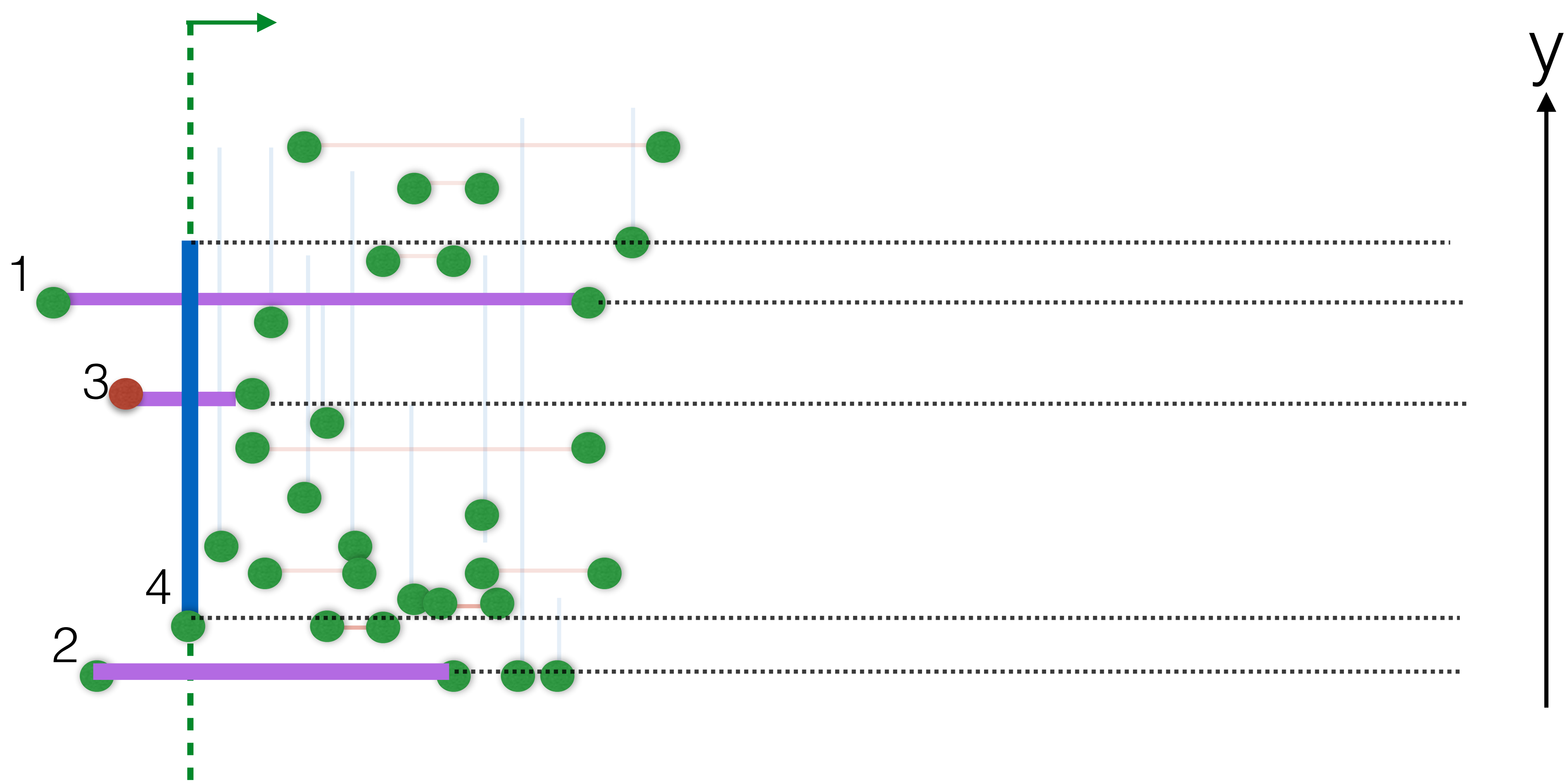


When reach an event corresponding to a vertical segment:

Claim: All horizontal segments that it intersects must be active

But, not all active segments intersect the vertical segment

The line sweep technique



When reach an event corresponding to a vertical segment:

Claim: All horizontal segments that it intersects must be active

But, not all active segments intersect the vertical segment

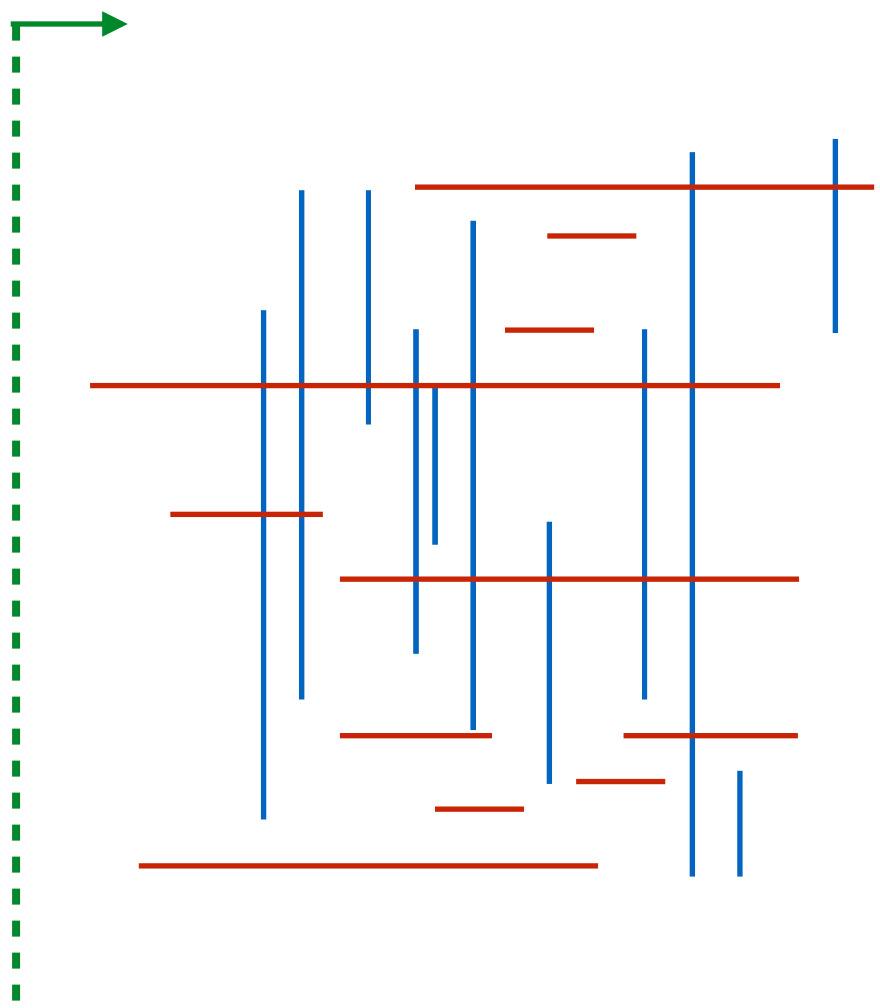
The line sweep technique

When reach an event corresponding to a vertical segment:

Claim: All horizontal segments that it intersects must be active

But, not all active segments intersect the vertical segment

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
  - if x is end of horizontal segment (x, x', y):

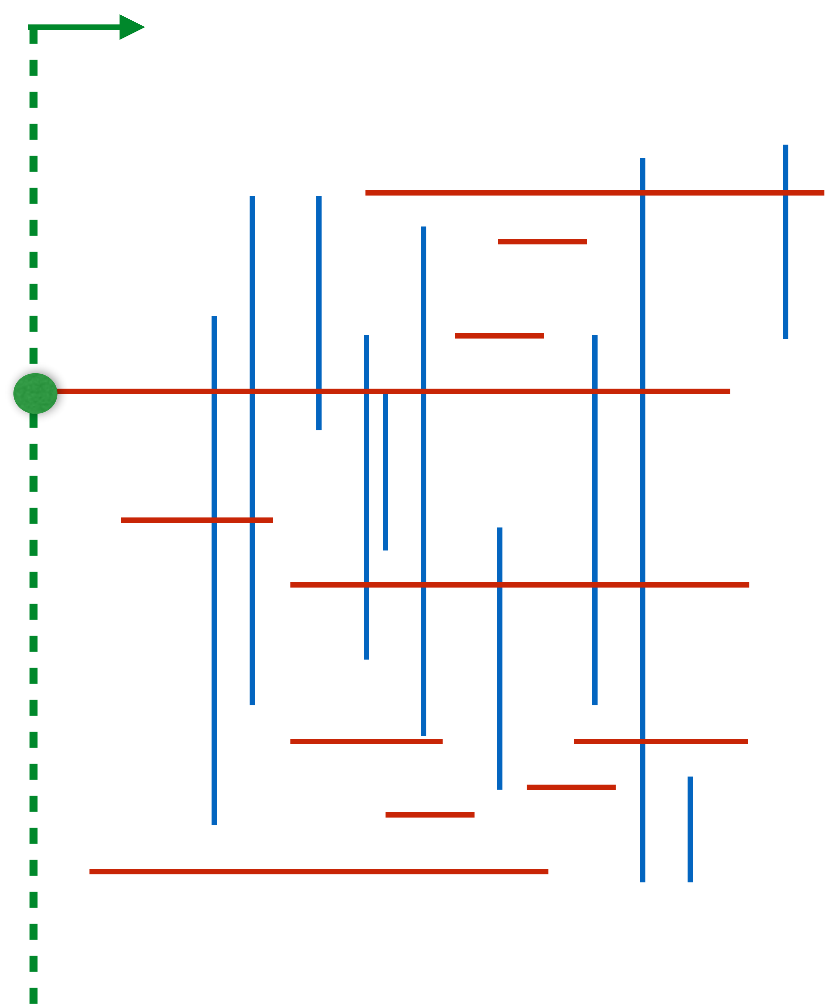
    //segment stops being active

    delete segment (x,x',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']

    search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
  - if x is end of horizontal segment (x, x', y):

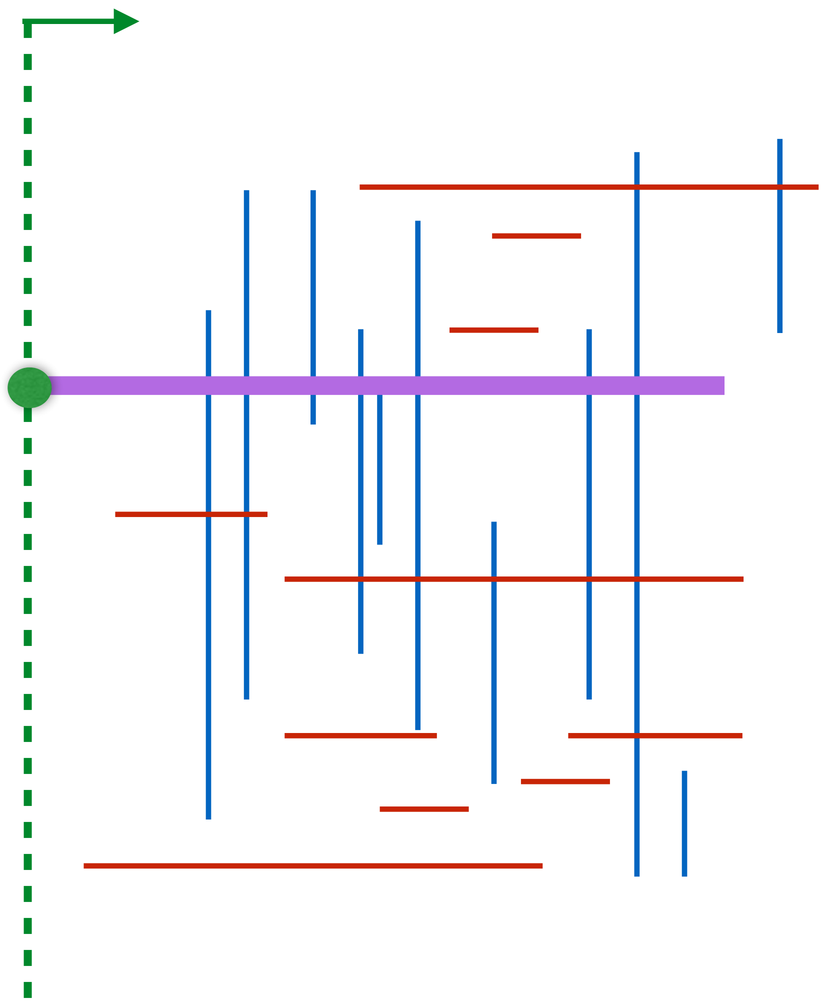
    //segment stops being active

    delete segment (x,x',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']

    search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
  - if x is end of horizontal segment (x, x', y):

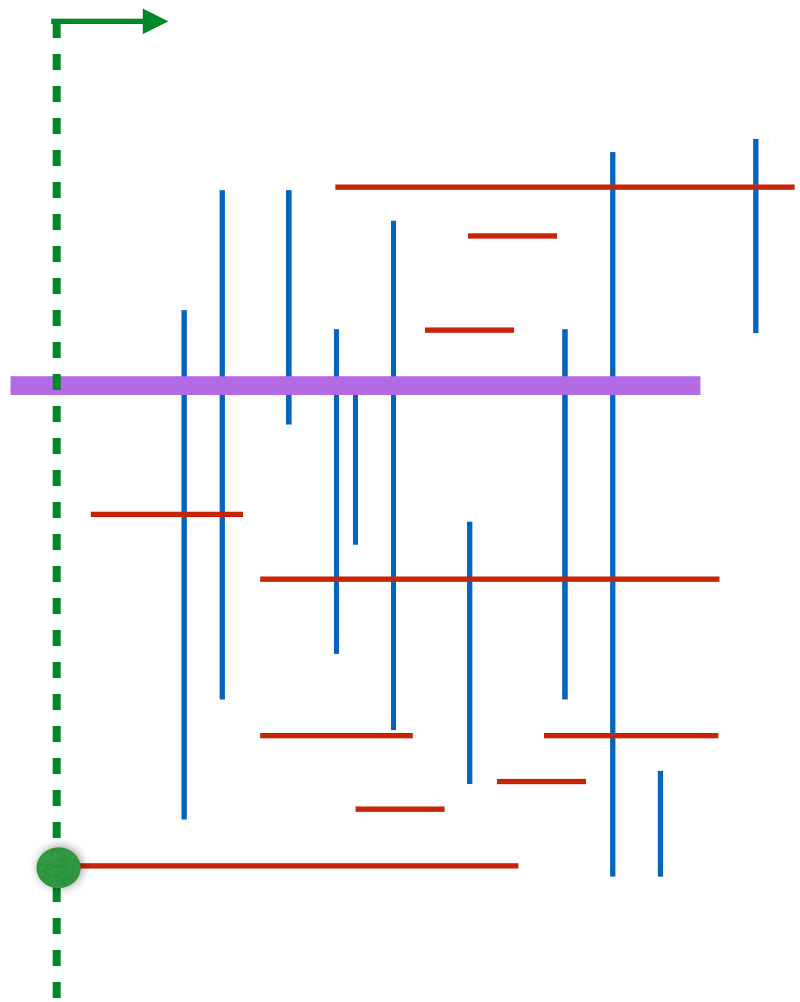
    //segment stops being active

    delete segment (x,x',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']

    search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
  - if x is end of horizontal segment (x, x', y):

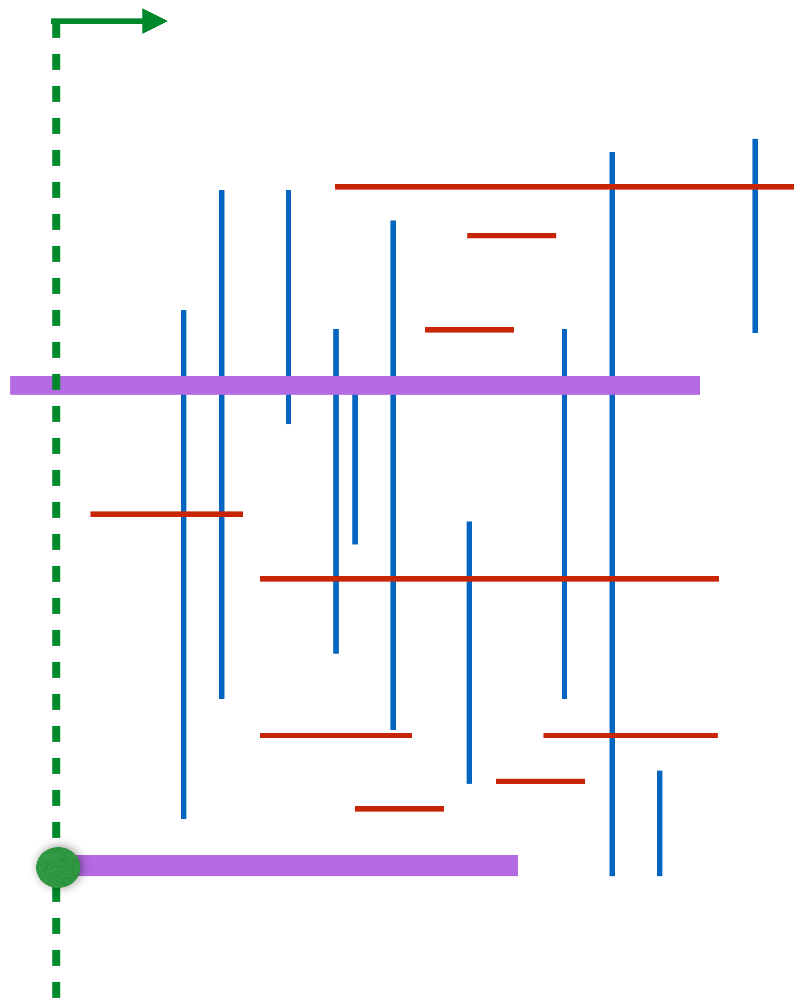
    //segment stops being active

    delete segment (x,x',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']

    search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
    - if x is end of horizontal segment (x, x', y):

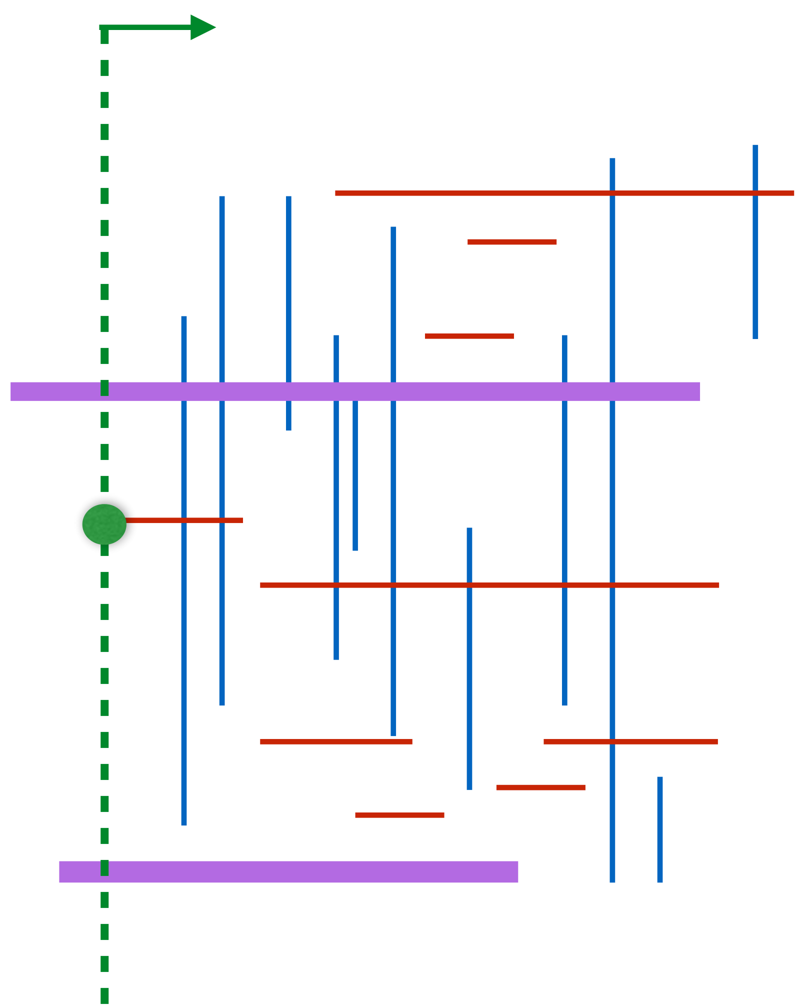
      //segment stops being active

      delete segment (x,x',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']

      search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
  - if x is end of horizontal segment (x, x', y):

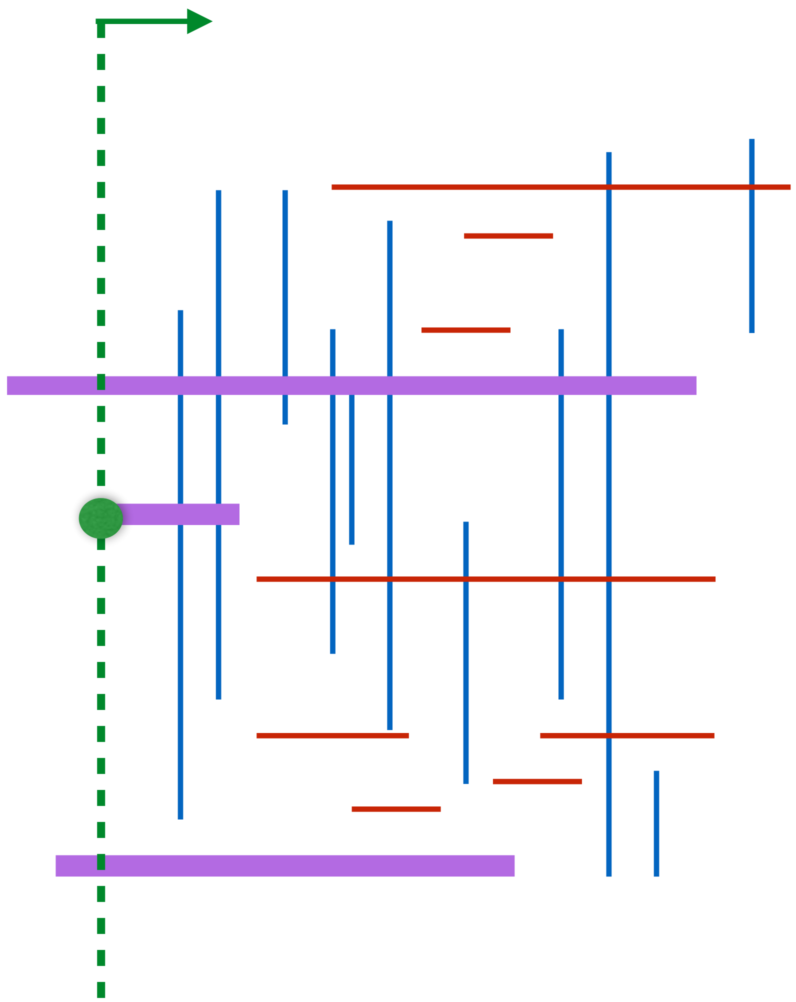
    //segment stops being active

    delete segment (x,x',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']

    search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
    - if x is end of horizontal segment (x, x', y):

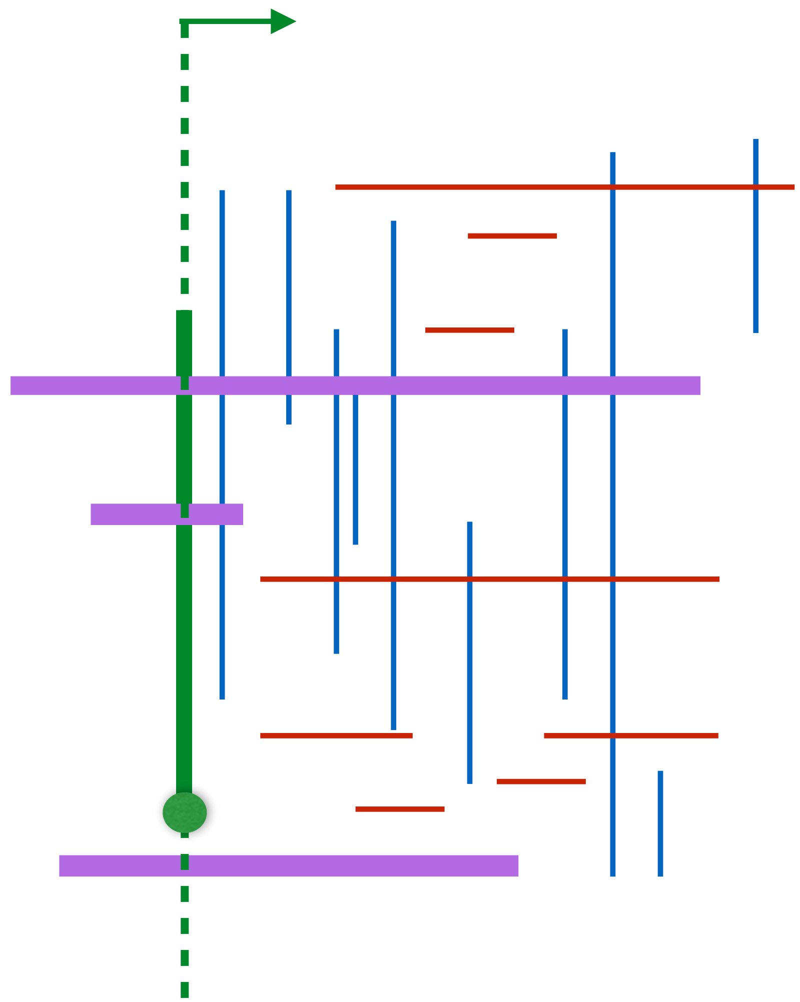
      //segment stops being active

      delete segment (x,x',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']

      search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
  - if x is end of horizontal segment (x, x', y):

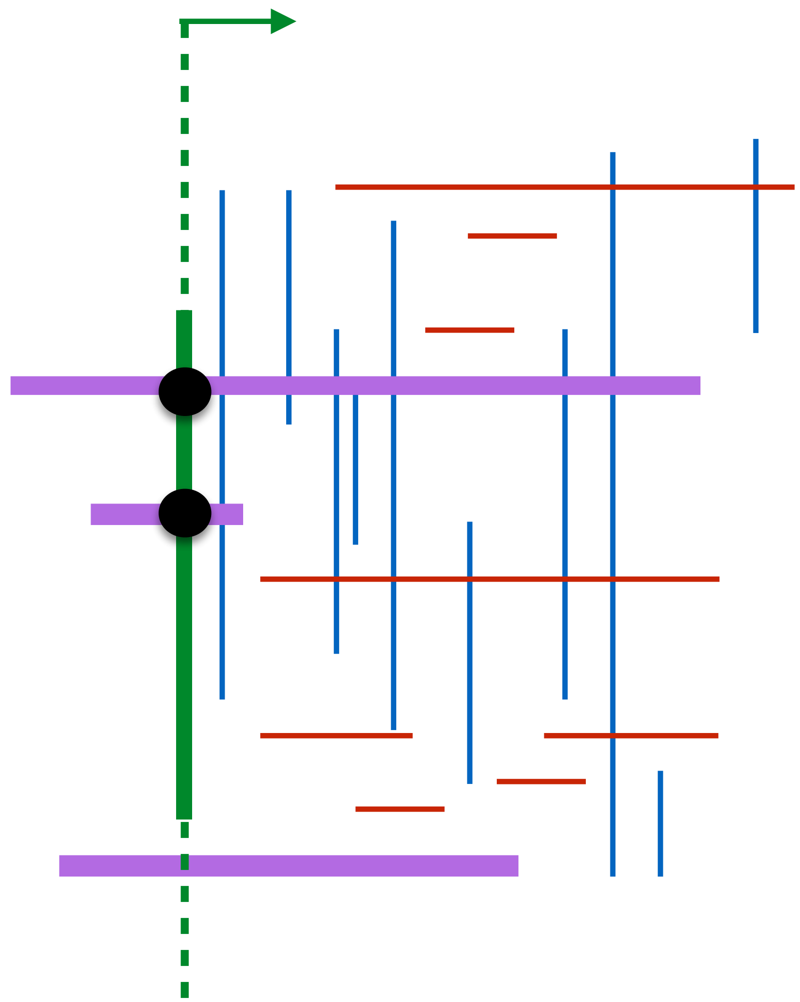
    //segment stops being active

    delete segment (x,x',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']

    search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
    - if x is end of horizontal segment (x, x', y):

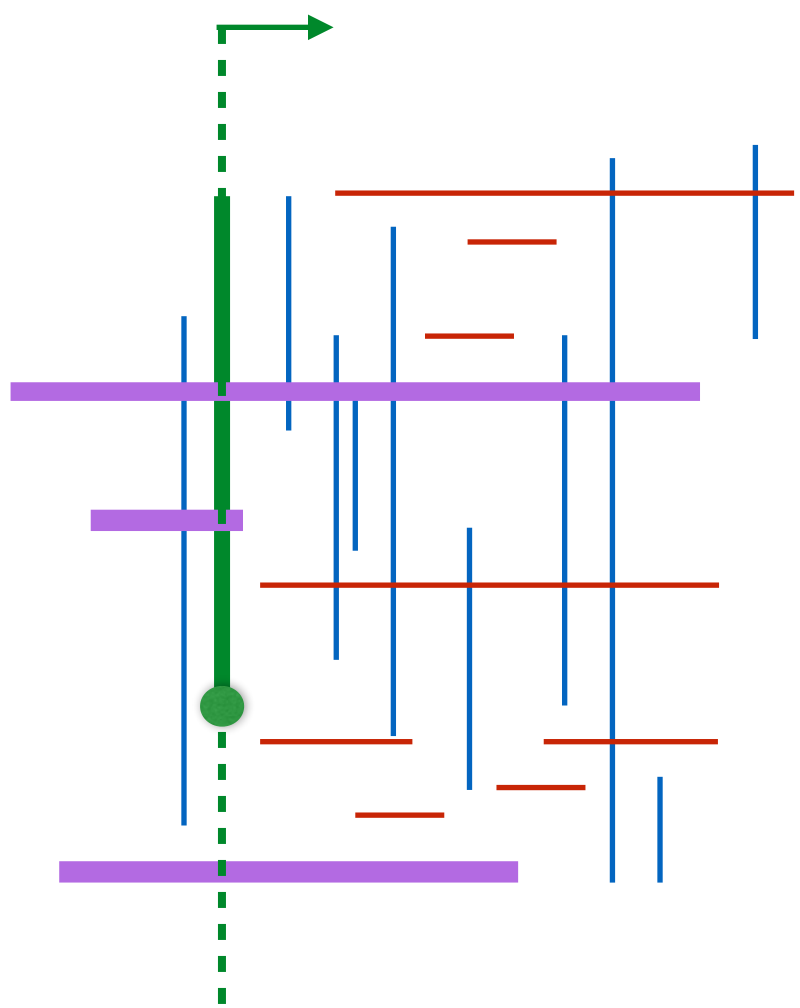
      //segment stops being active

      delete segment (x,x',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']

      search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
  - if x is end of horizontal segment (x, x', y):

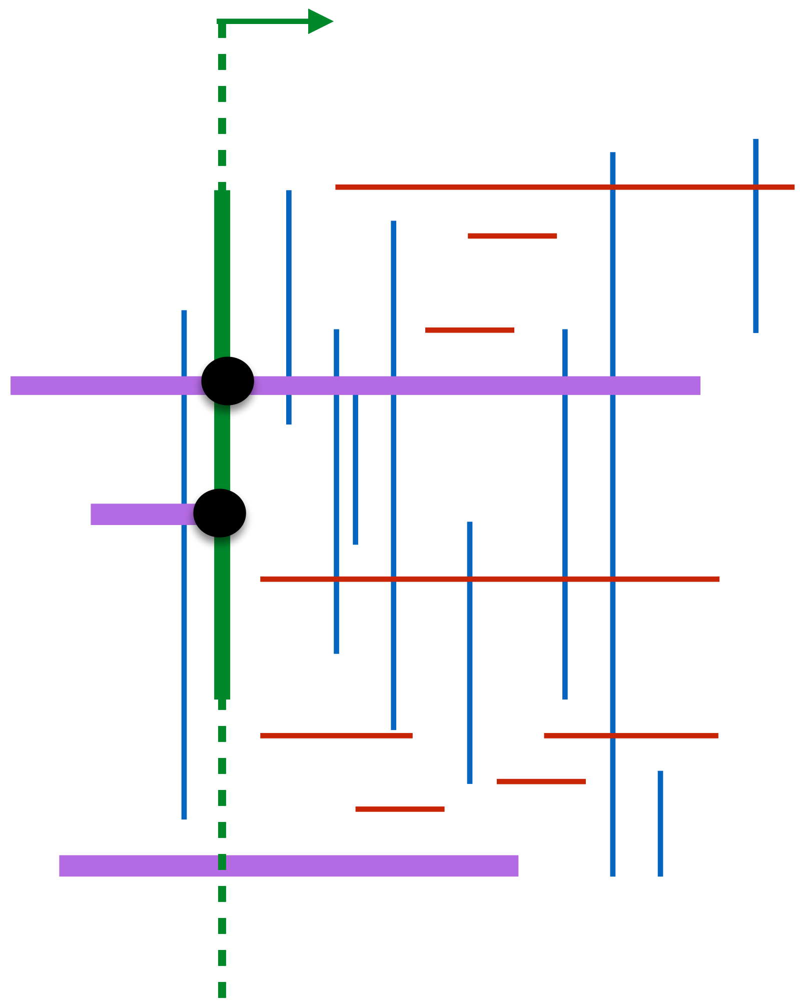
    //segment stops being active

    delete segment (x,x',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']

    search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
  - if x is end of horizontal segment (x, x', y):

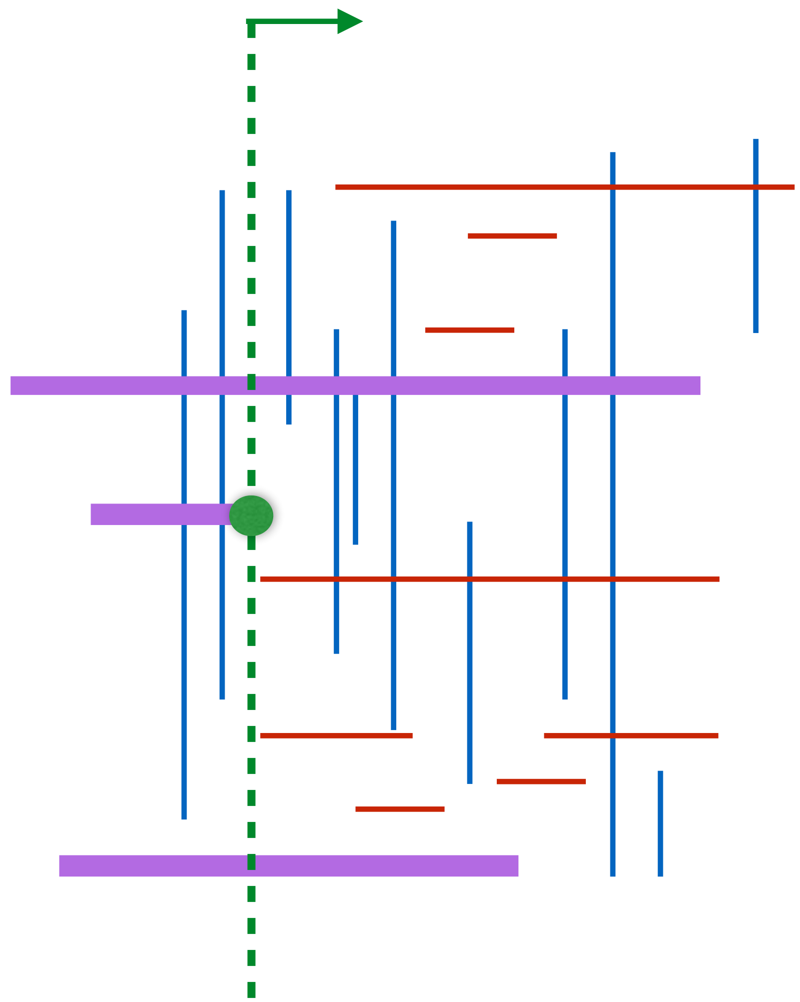
    //segment stops being active

    delete segment (x,x',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']

    search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
  - if x is end of horizontal segment (x, x', y):

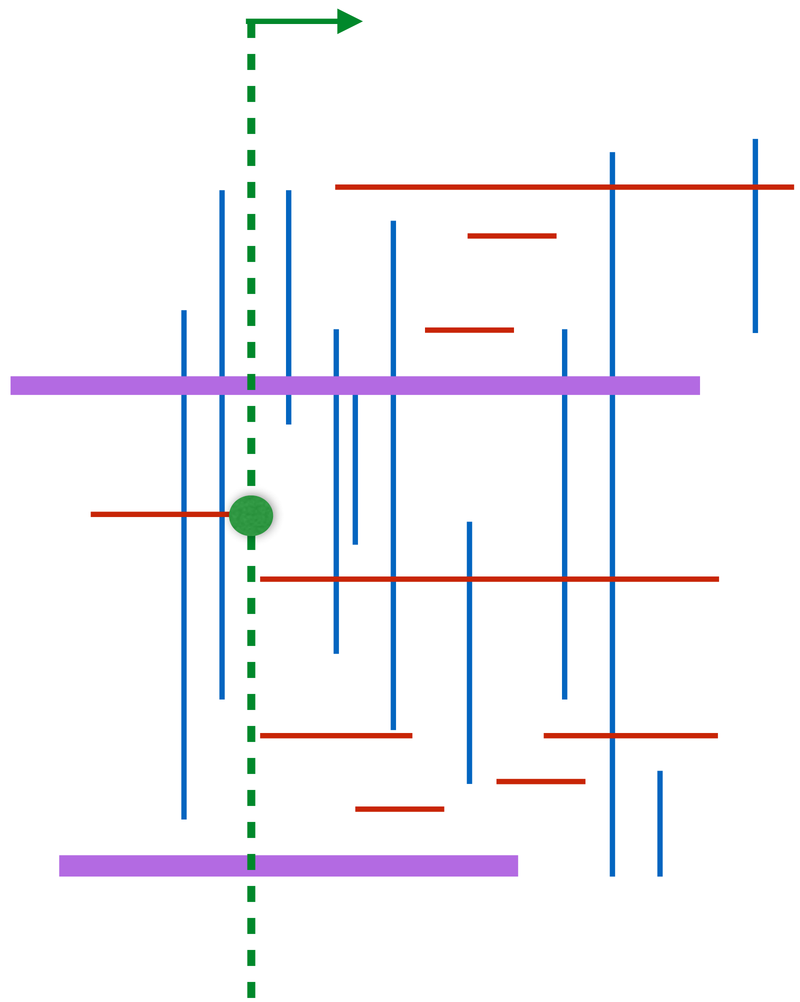
    //segment stops being active

    delete segment (x,x',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']

    search AS for all segments with y-value in given range [y,y'] and report intersections

Orthogonal line segment intersection



- Let X be the set of x-coordinates of all segments //the events
- Initialize AS = {}
- Sort X and traverse the events in sorted 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) in AS
    - if x is end of horizontal segment (x, x’, y):

      //segment stops being active

      delete segment (x,x’,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’]

      search AS for all segments with y-value in given range [y,y’] and report intersections

Orthogonal line segment intersection

- To think
  - How to implement the AS?
  - Analysis

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

    //segment becomes active

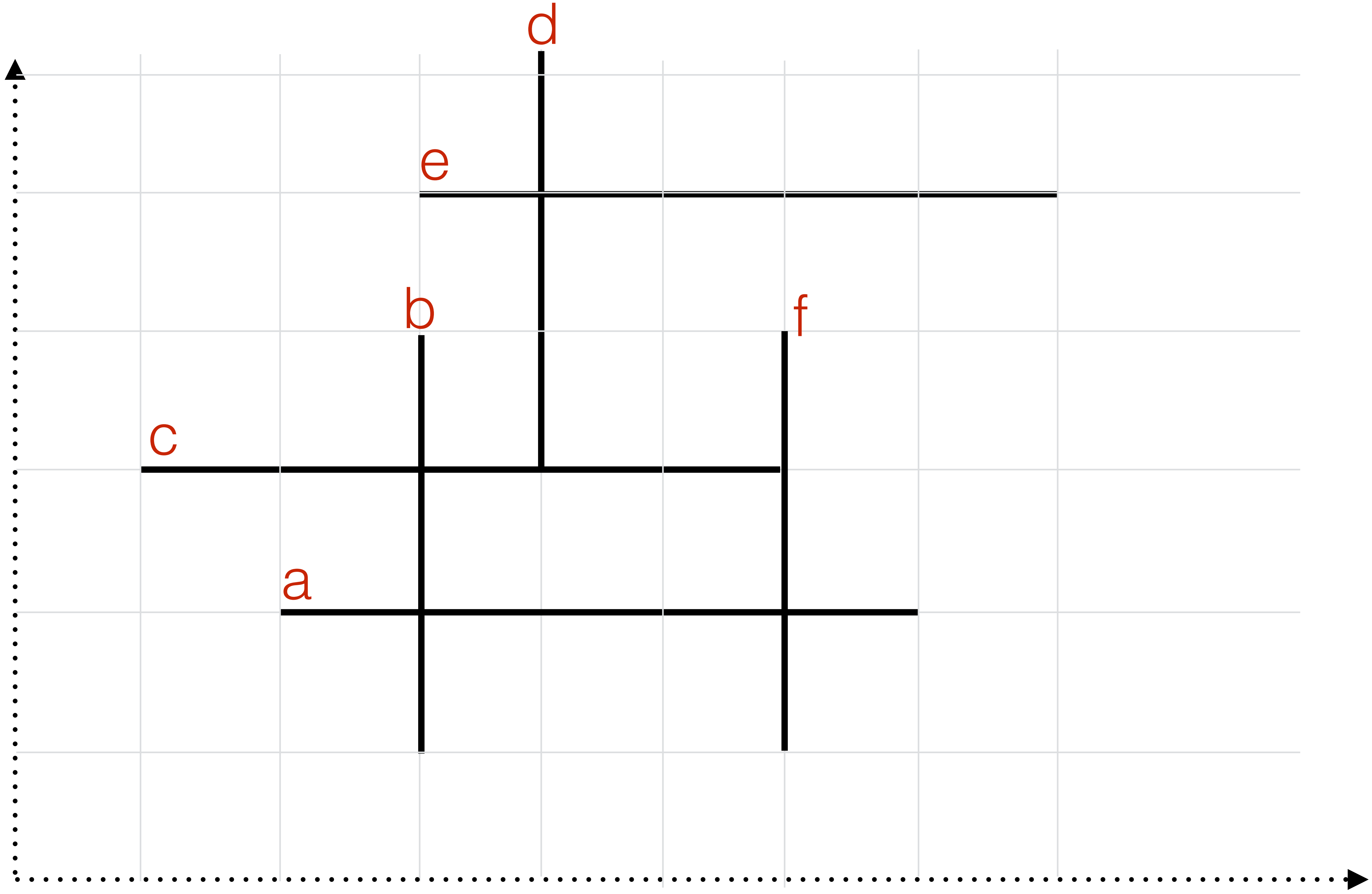
    insert segment (x,x',y) in AS
  - if x is end of horizontal segment (x, x', y):

    //segment stops being active

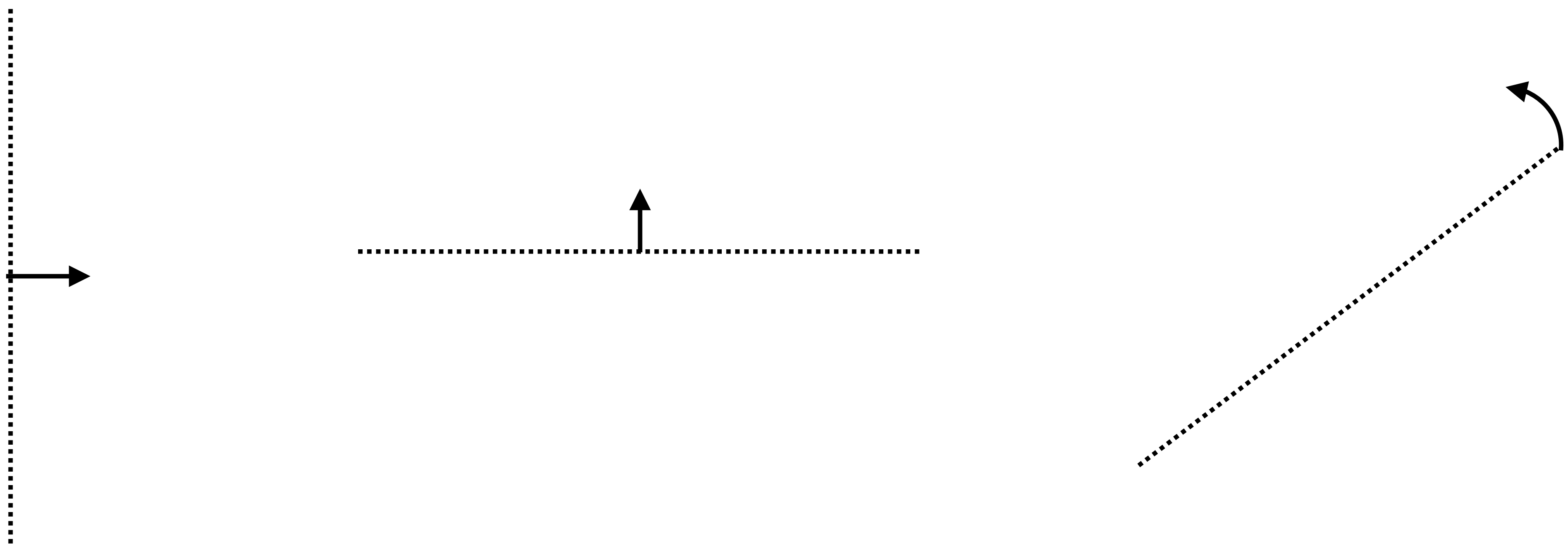
    delete segment (x,x',y) from AS
  - if x corresponds to a vertical segment (y, y',x):

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

    search AS for all segments with y-value in given range [y,y'] and report intersections

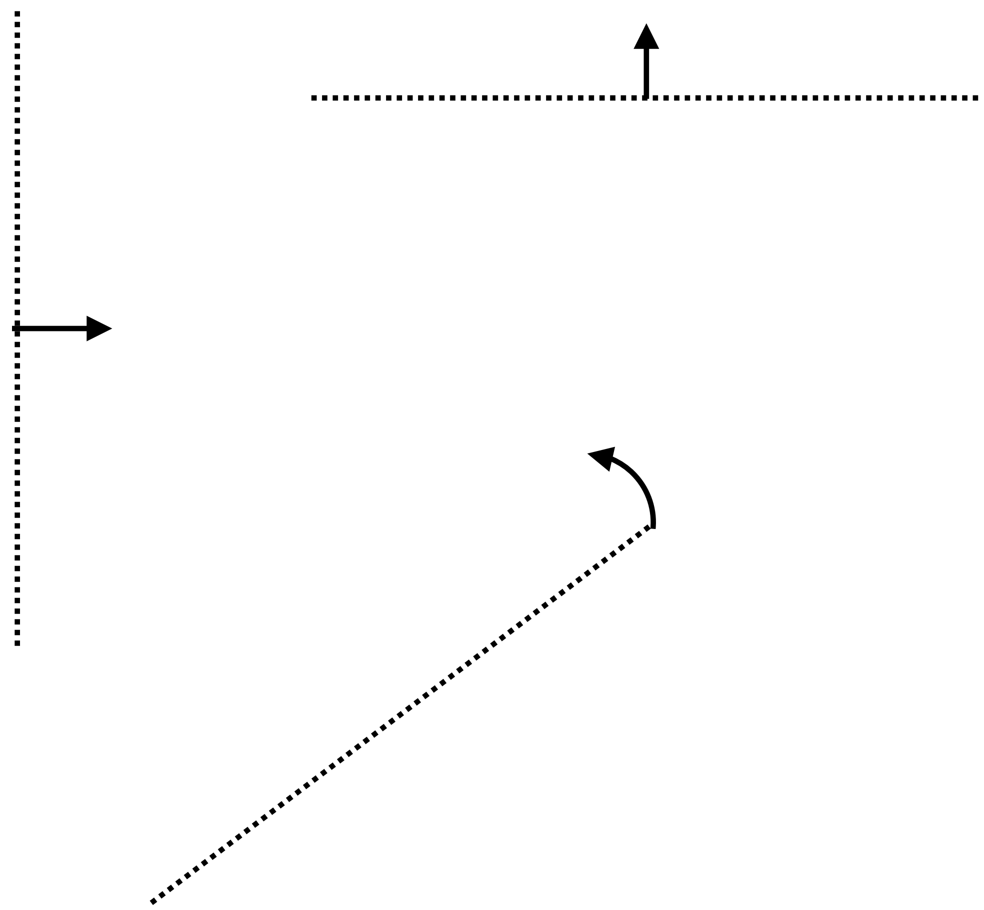

d
e
b
f
c
a

Line sweep



Line sweep algorithms

- Powerful, elegant, frequently used technique
- Line can be horizontal or vertical or radial or ….



- Traverse events in order and maintain an Active Structure (AS)
    - AS contains objects that are “active” (started but not ended) in other words they are intersected by the current sweep line
    - at some events, insert in AS
    - at some events, delete from AS
    - at some events, query AS