A Data Structure for Dynamic Segment Intersection

Kalina Petrova

Department of Computer Science, Princeton University

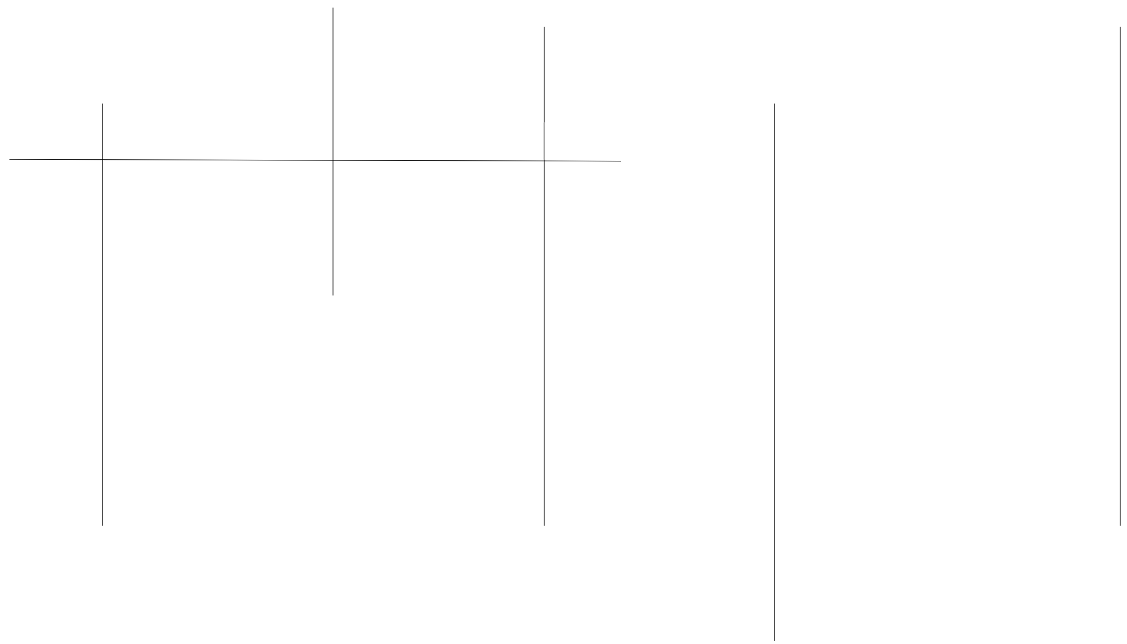
Robert Tarjan

Department of Computer Science, Princeton University,

and Intertrust Technologies

Problem Statement
The Segment Intersection Problem

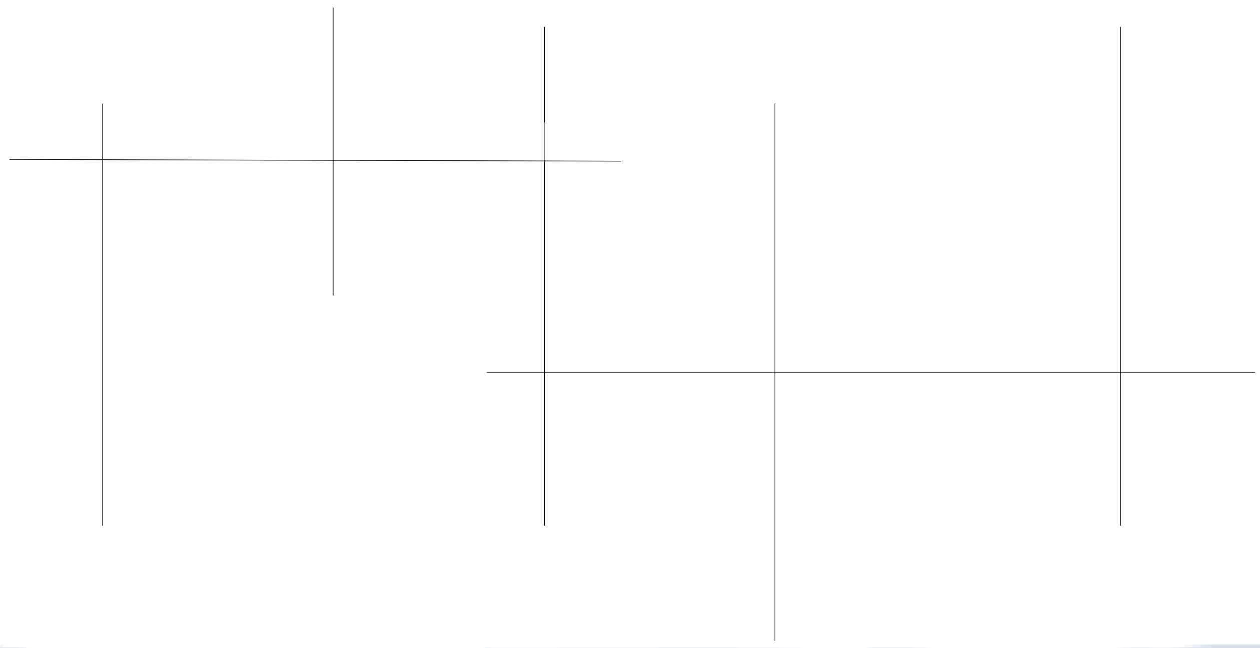
Given a set of axis-aligned segments in the plane, support the following operations



Problem Statement
The Segment Intersection Problem

Given a set of axis-aligned segments in the plane, support the following operations

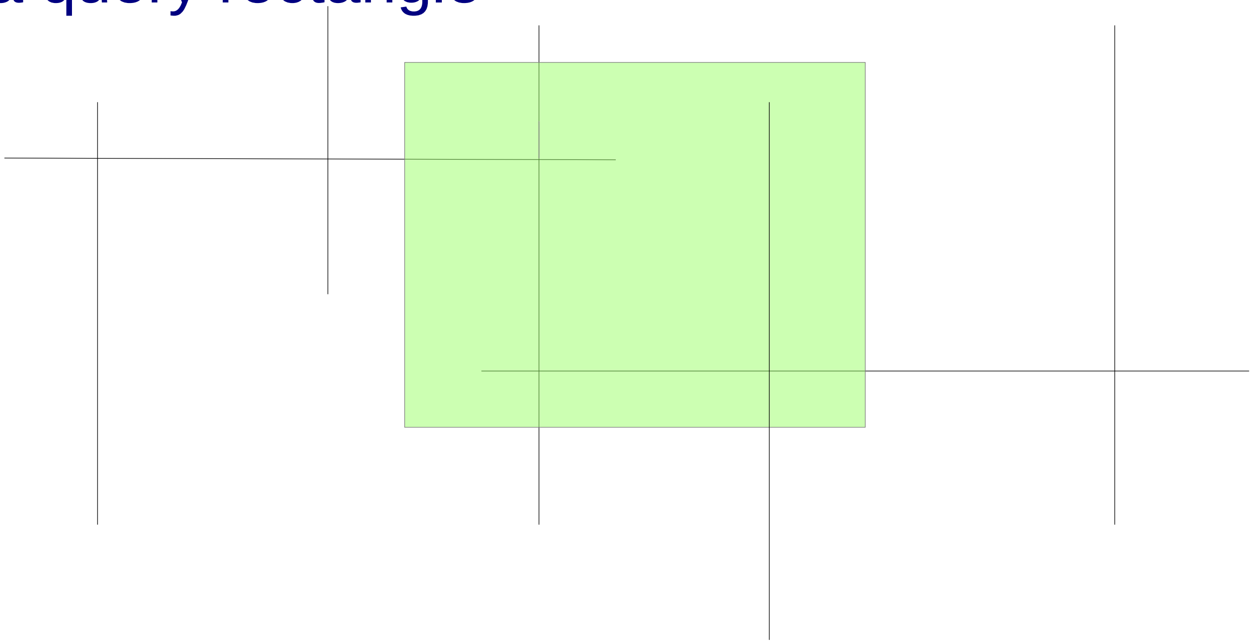
- **Insertion**: insert a new segment



Problem Statement
The Segment Intersection Problem

Given a set of axis-aligned segments in the plane, support the following operations

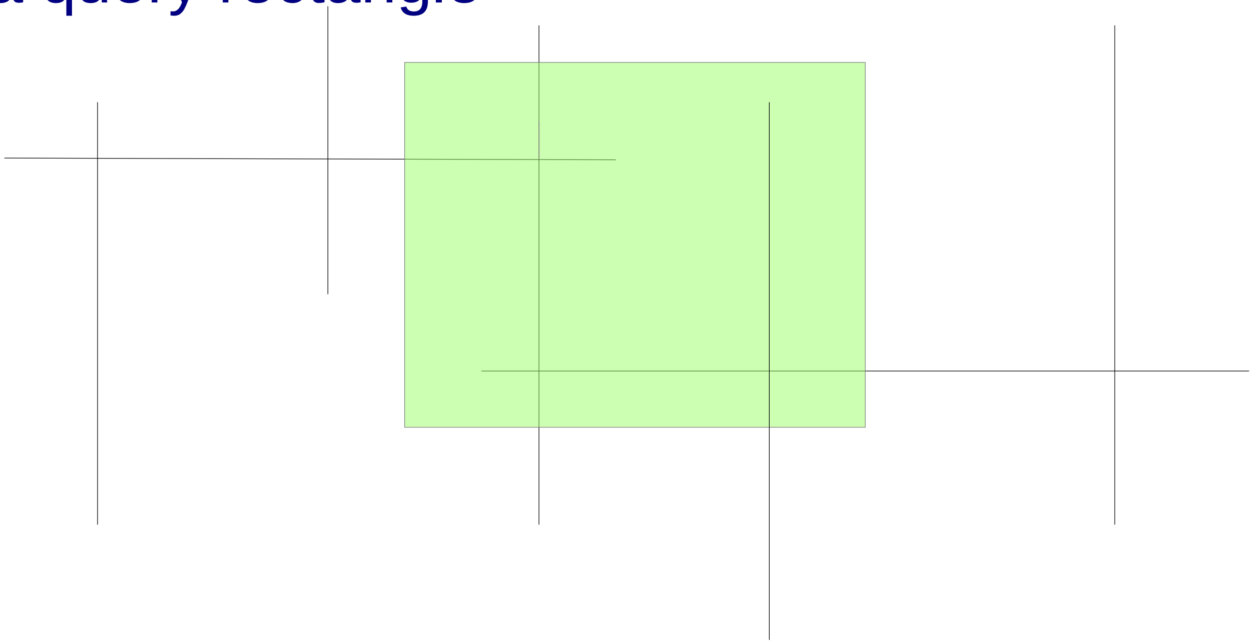
- **Insertion**: insert a new segment
- **Query**: report all pairwise intersection points of segments inside a query rectangle



Problem Statement
The Segment Intersection Problem

Given a set of axis-aligned segments in the plane, support the following operations

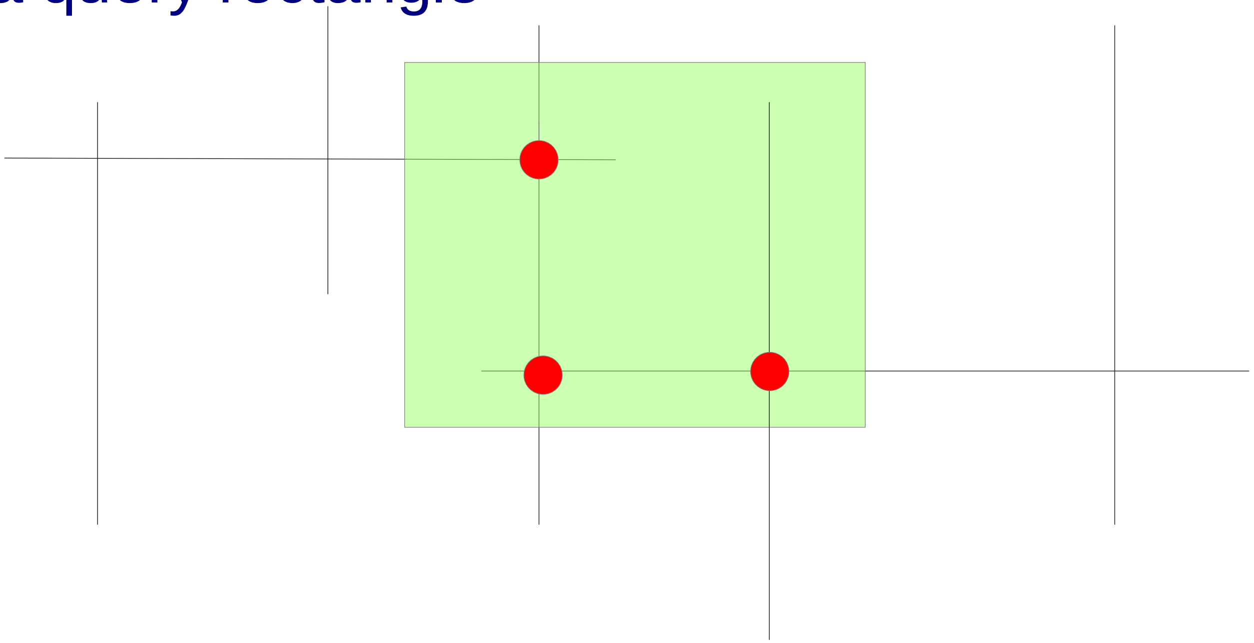
- **Insertion**: insert a new segment
- **Query**: report all pairwise intersection points of segments inside a query rectangle



Problem Statement
The Segment Intersection Problem

Given a set of axis-aligned segments in the plane, support the following operations

- **Insertion**: insert a new segment
- **Query**: report all pairwise intersection points of segments inside a query rectangle



Desired properties

- Small time complexity of insertion
- Small time complexity of query
- Small preprocessing time
- Small space complexity

Related work
Static version of the problem

Finding Pairwise Intersections Inside a Query Range

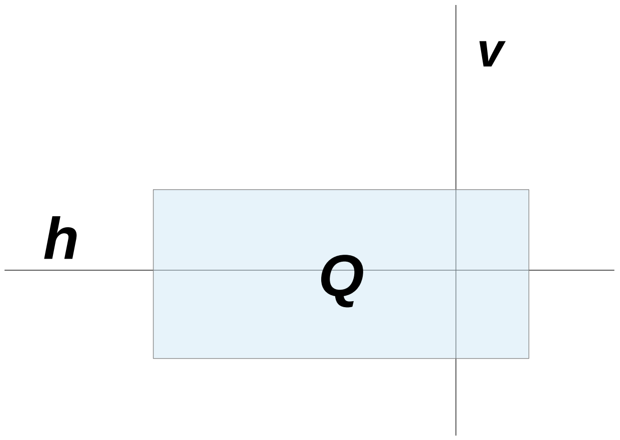
- by Mark de Berg, Joachim Gudmundsson, and Ali D. Mehrabi
- Supports only queries, not insertions

Outline

- Problem statement
- Static version of the problem
- Reduction of the dynamic version to a different problem
- Solution
- Results
- Conclusion

Approach for the static case

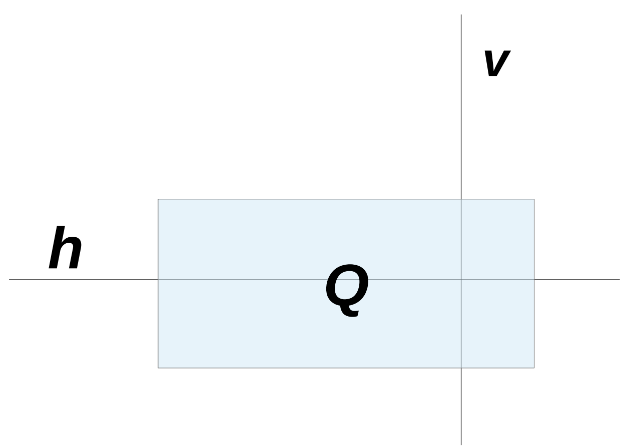
- Construct a set $O^*(Q)$:
  - If $h \cap v \in Q$, $h \in O^*(Q)$ or $v \in O^*(Q)$.
- For each $s \in O^*(Q)$:
  - Find all segments t: $s \cap Q \cap t \neq \emptyset$ using a multidimensional range tree.



Approach for the static case

- **Construct a set $O^*(Q)$:**
  - **If $h \cap v \in Q$, $h \in O^*(Q)$ or $v \in O^*(Q)$.**
- **For each $s \in O^*(Q)$:**
  - **Find all segments t: $s \cap Q \cap t \neq \emptyset$ using a multidimensional range tree.**

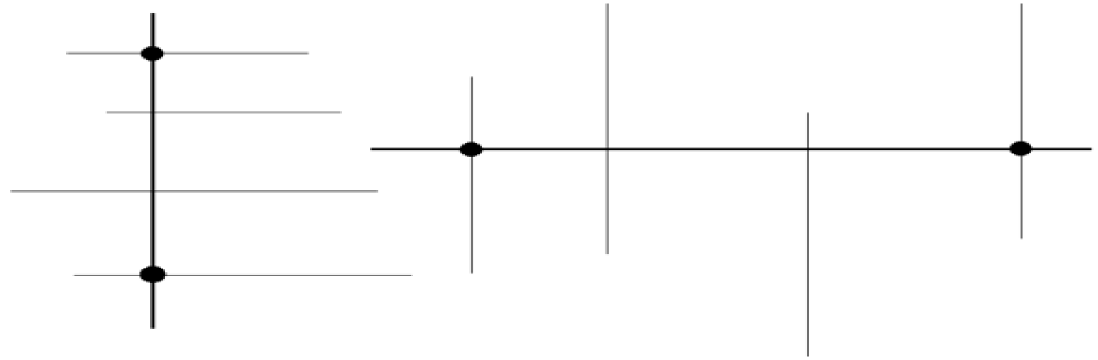
Easy: See [de Berg, Gudmundsson, and Mehrabi, 2015]



Approach for the static case

Witness points W:

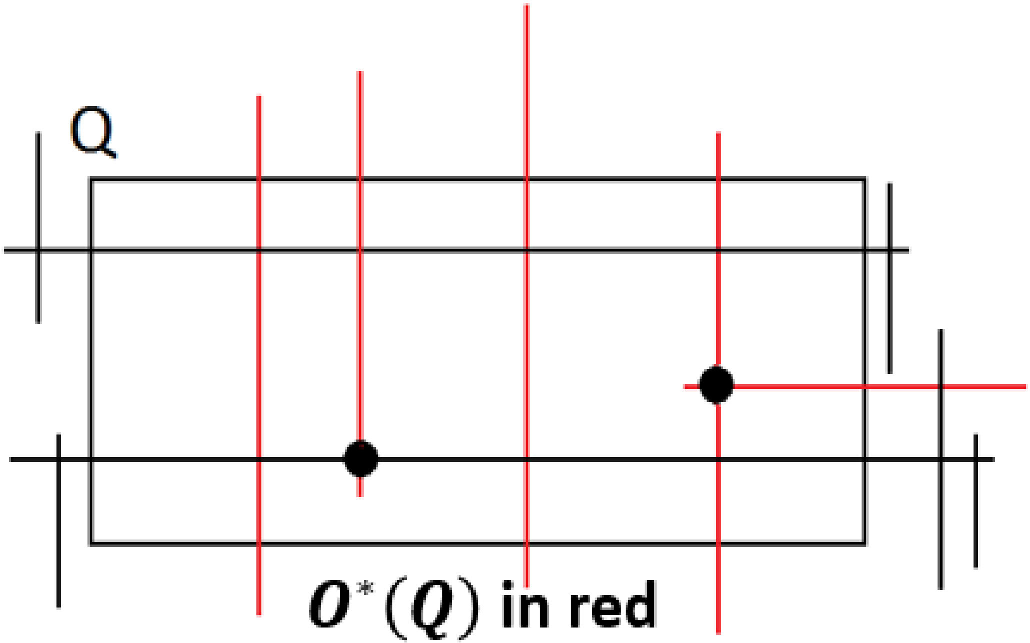
- $\forall$ v**, the topmost and bottommost intersection points of** v**,** $w_1, w_2 \in W$ $(s_{w_1} = v, s_{w_2} = v)$
- $\forall$ h**, the leftmost and rightmost intersection points of** h**,** $w_1, w_2 \in W$ $(s_{w_1} = h, s_{w_2} = h)$



Approach for the static case

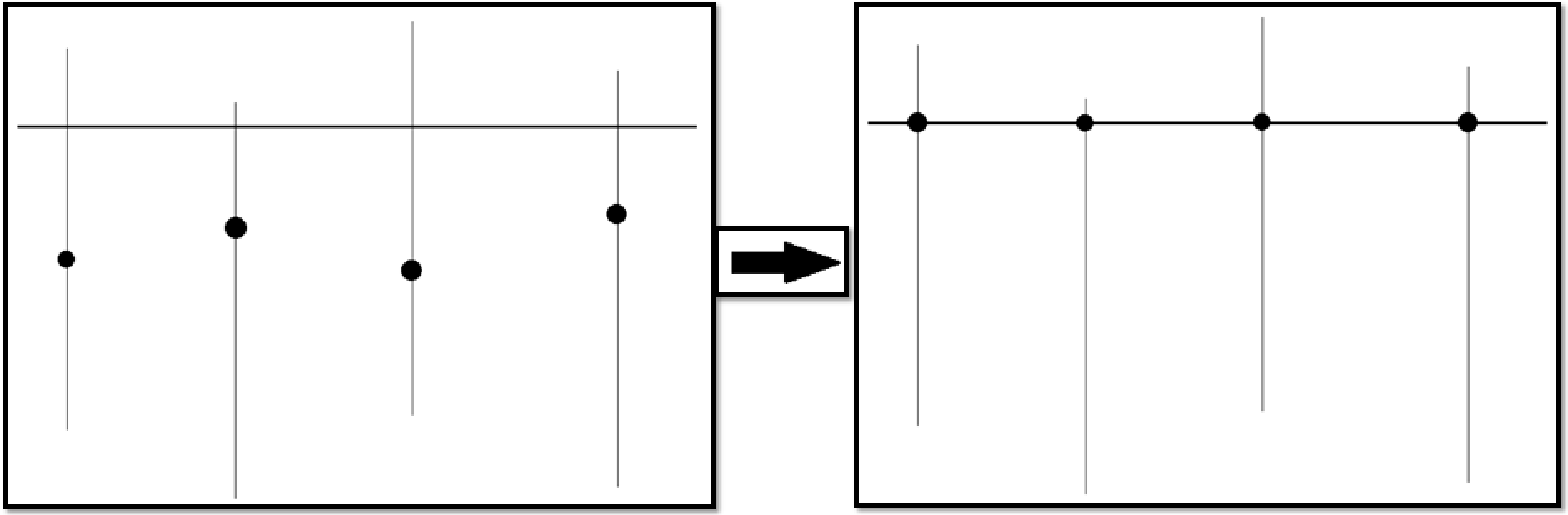
$O^*(Q)$ contains:

- $\{s_w | w \in Q, w \in W\}$ (each segment that has an associated witness point in Q)
- All segments that intersect Q entirely from top to bottom
  (only if there are segments that intersect Q entirely from left to right)



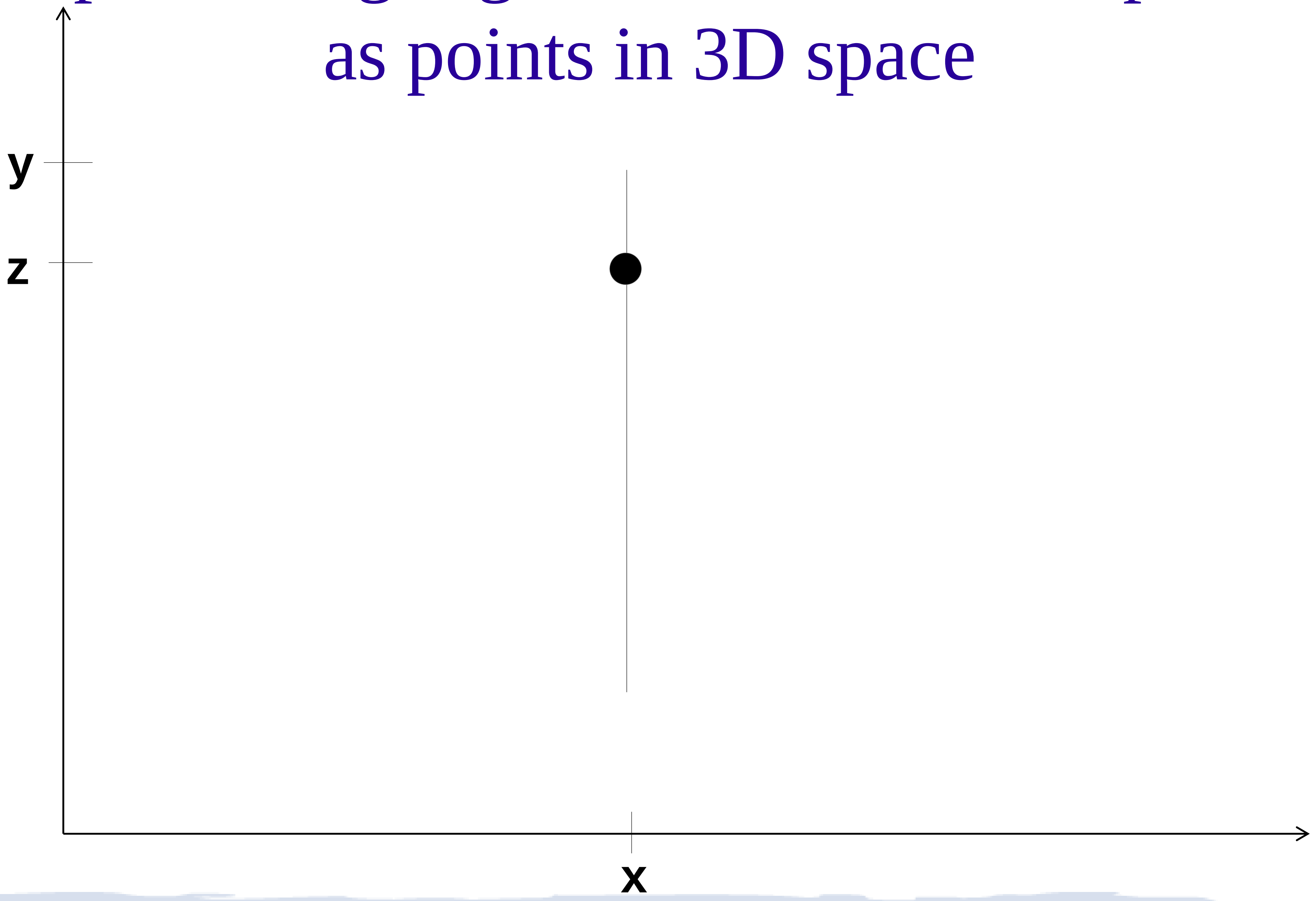
$O^*(Q)$ in red

Approach for the dynamic case
Insertion of a new horizontal segment

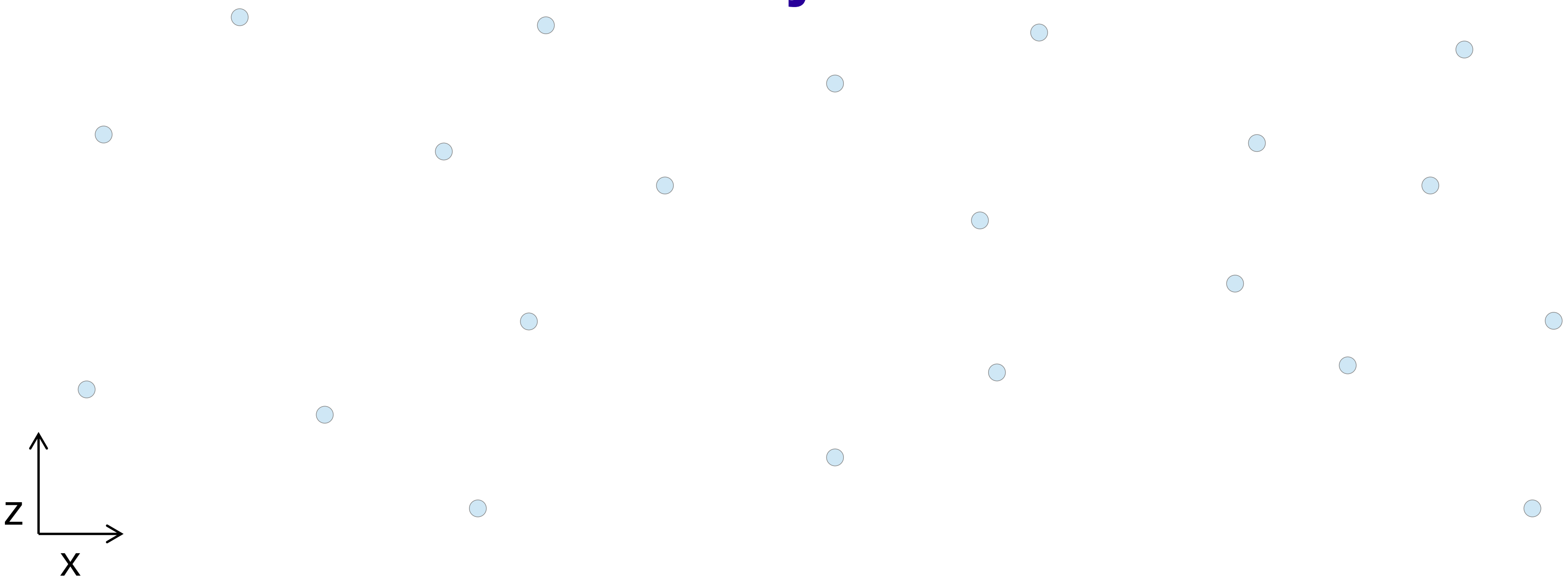


How the witness points of vertical segments change as a result of the insertion of a new horizontal segment

Representing segments with witness points as points in 3D space

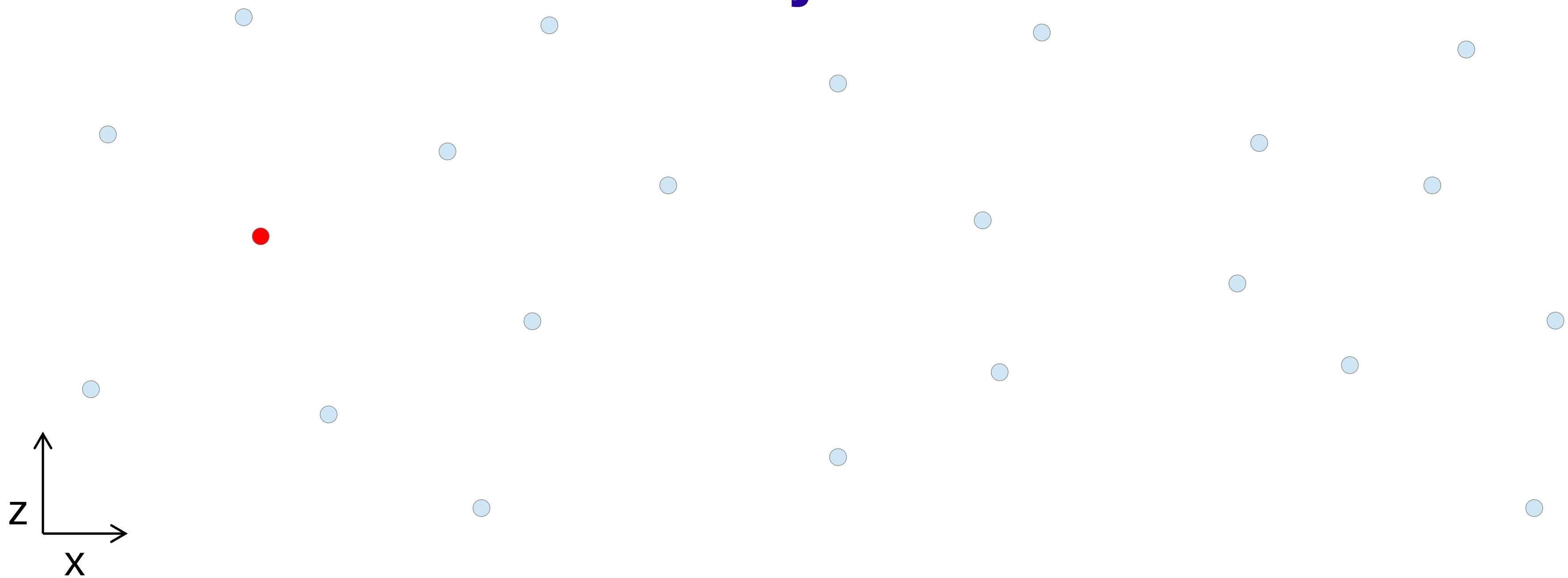


The Core of the Problem
The Point Projection Problem



Support the following operations on a set of points in 3D space:

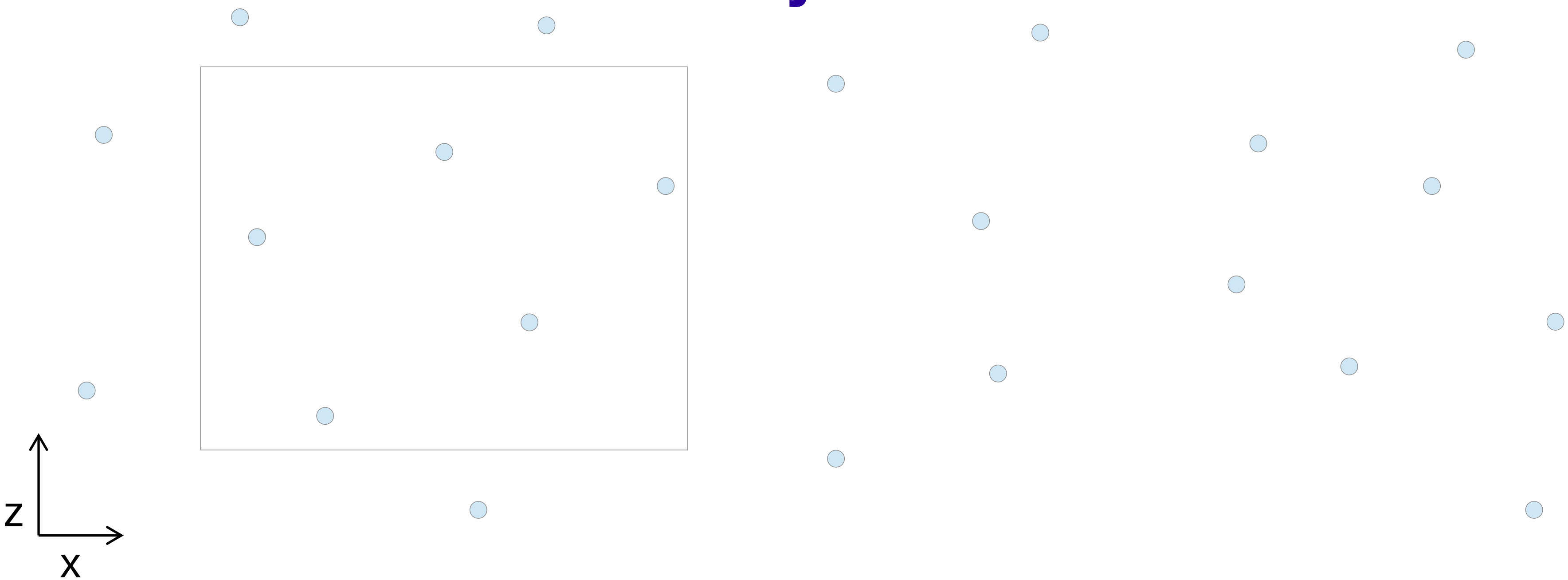
The Core of the Problem
The Point Projection Problem



Support the following operations on a set of points in 3D space:

- Insert a point (x, y, z). **(insertion)**

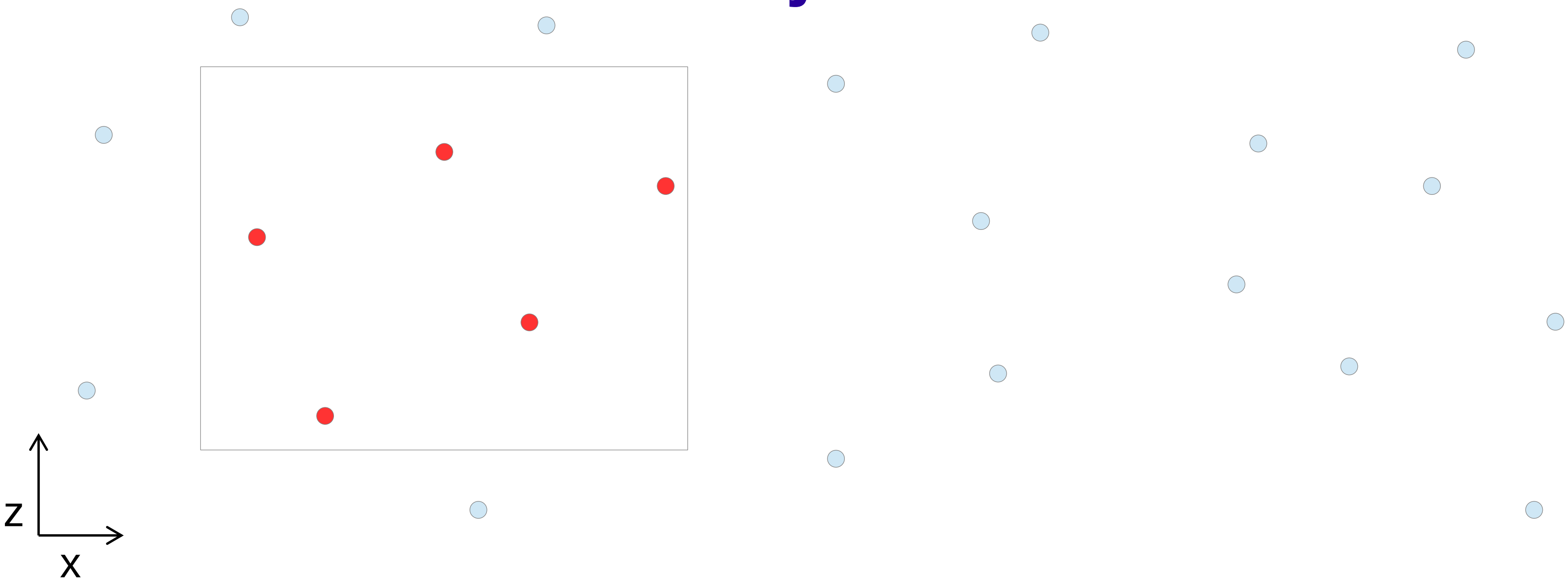
The Core of the Problem
The Point Projection Problem



Support the following operations on a set of points in 3D space:

- Insert a point (x, y, z).
- Report all points in the box $[x_1, x_2] \times (-\infty, \infty) \times [z_1, z_2]$. **(query)**

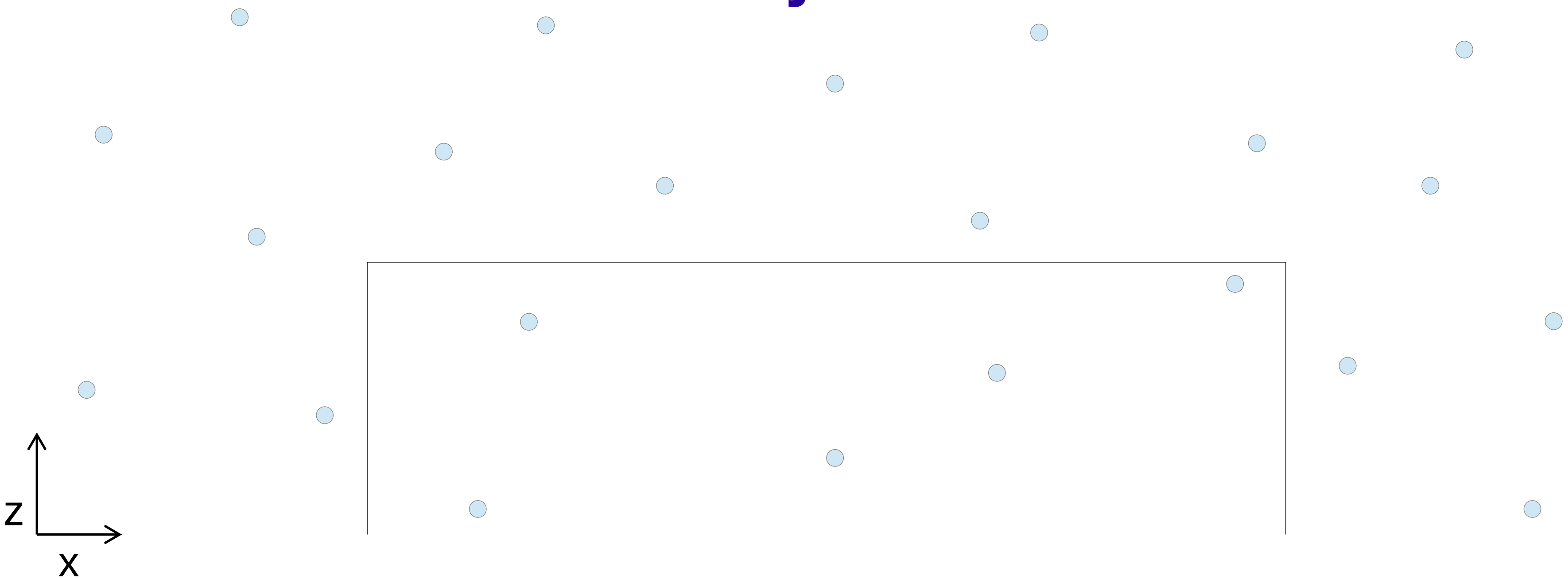
The Core of the Problem
The Point Projection Problem



Support the following operations on a set of points in 3D space:

- Insert a point (x, y, z).
- Report all points in the box $[x_1, x_2] \times (-\infty, \infty) \times [z_1, z_2]$.

The Core of the Problem
The Point Projection Problem

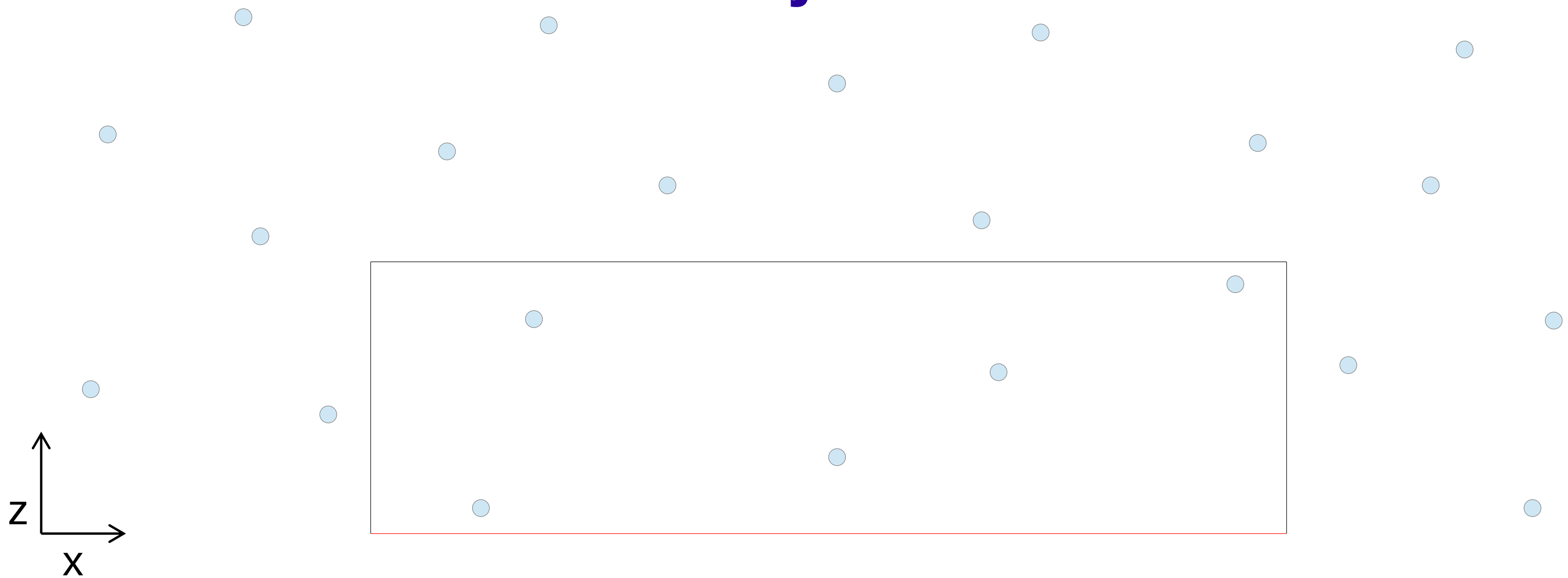


Support the following operations on a set of points in 3D space:

- Insert a point (x, y, z).
- Report all points in the box $[x_1, x_2] \times (-\infty, \infty) \times [z_1, z_2]$.
- For all points (x, y, z) in the box $[x_1, x_2] \times (y_1, \infty) \times [-\infty, z_2]$, change z to z_2.

(update)

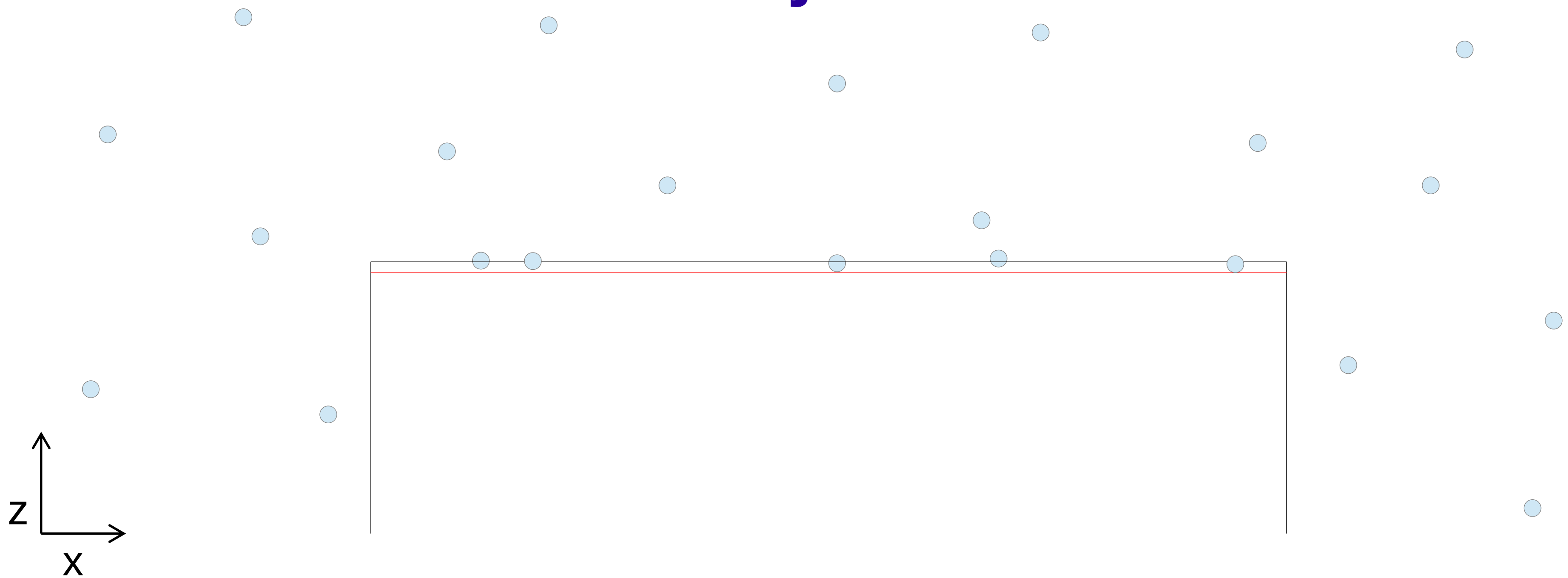
The Core of the Problem
The Point Projection Problem



Support the following operations on a set of points in 3D space:

- Insert a point (x, y, z).
- Report all points in the box $[x_1, x_2] \times (-\infty, \infty) \times [z_1, z_2]$.
- For all points (x, y, z) in the box $[x_1, x_2] \times (y_1, \infty) \times [-\infty, z_2]$, change z to z_2.

The Core of the Problem
The Point Projection Problem



Support the following operations on a set of points in 3D space:

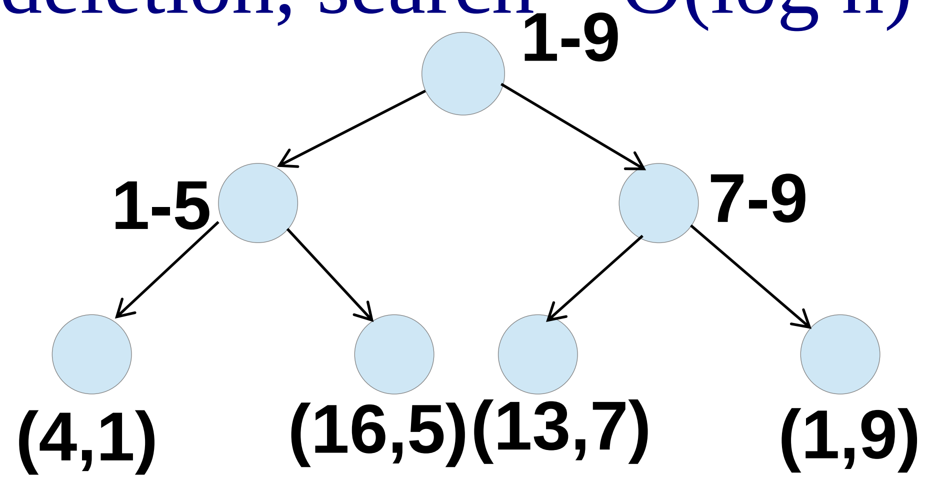
- Insert a point (x, y, z).
- Report all points in the box $[x_1, x_2] \times (-\infty, \infty) \times [z_1, z_2]$.
- For all points (x, y, z) in the box $[x_1, x_2] \times (y_1, \infty) \times [-\infty, z_2]$, change z to z_2.

Data Structure

- K-d tree
- AVL range trees [Lamoureux, 1995]
- The binary static to dynamic transformation [Saxe and Bentley, 1979]
- Potential function analysis

AVL range tree

- Space-partitioning data structure for organizing points by one of their dimensions
- Dynamic, balanced
- Insertion, deletion, search – O(log n)

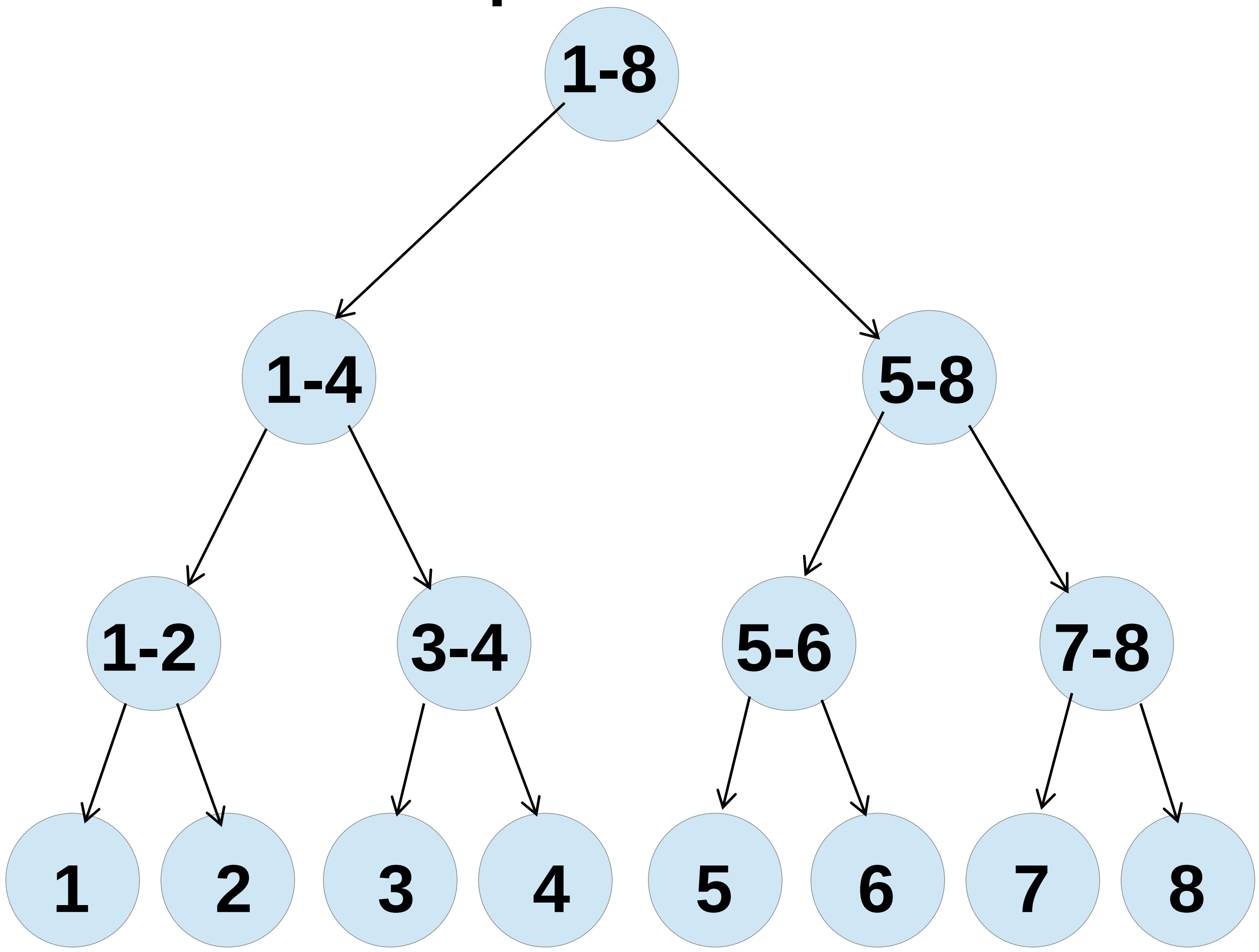


A range tree organized by the y coordinate

AVL range tree - an important fact

We can represent any interval using at most 2 log n nodes of the tree.

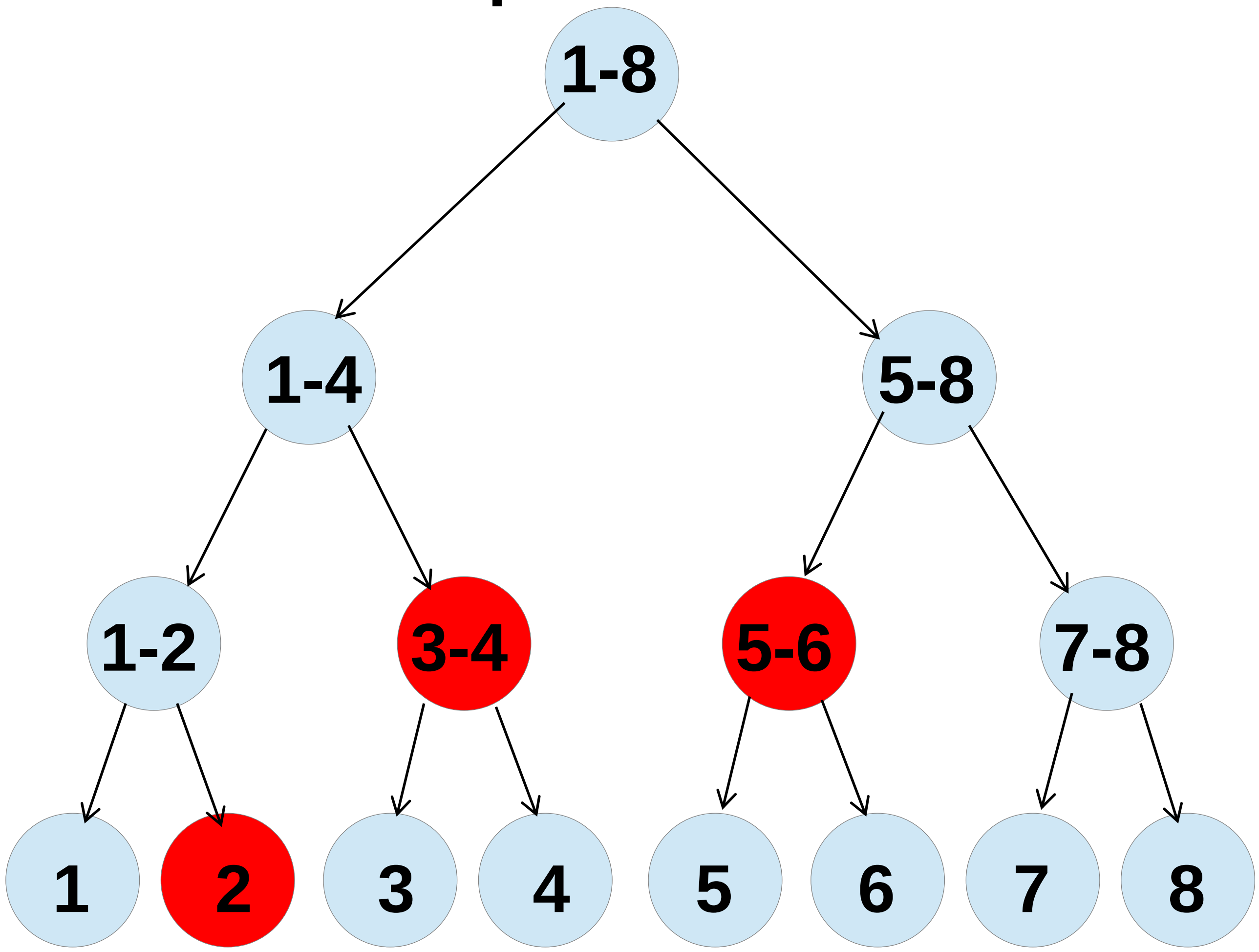
Example: interval 2-6



AVL range tree - an important fact

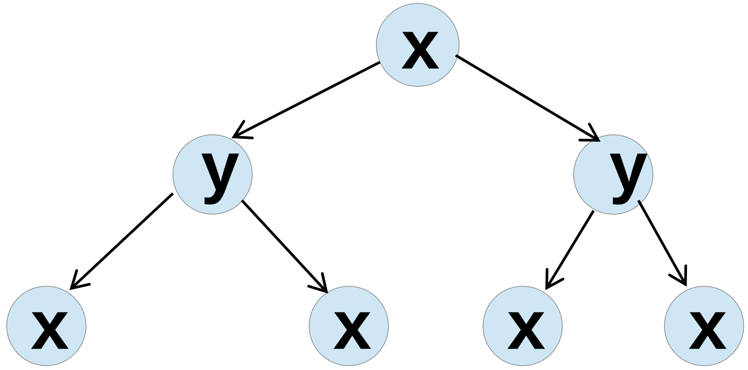
We can represent any interval using at most 2 log n nodes of the tree.

Example: interval 2-6



K-d tree

- Space-partitioning data structure for organizing points in k-dimensional space
- Used to store a set of points
- Partitioning alternating between the dimensions



A 2-dimensional k-d tree

K-d tree
Operations

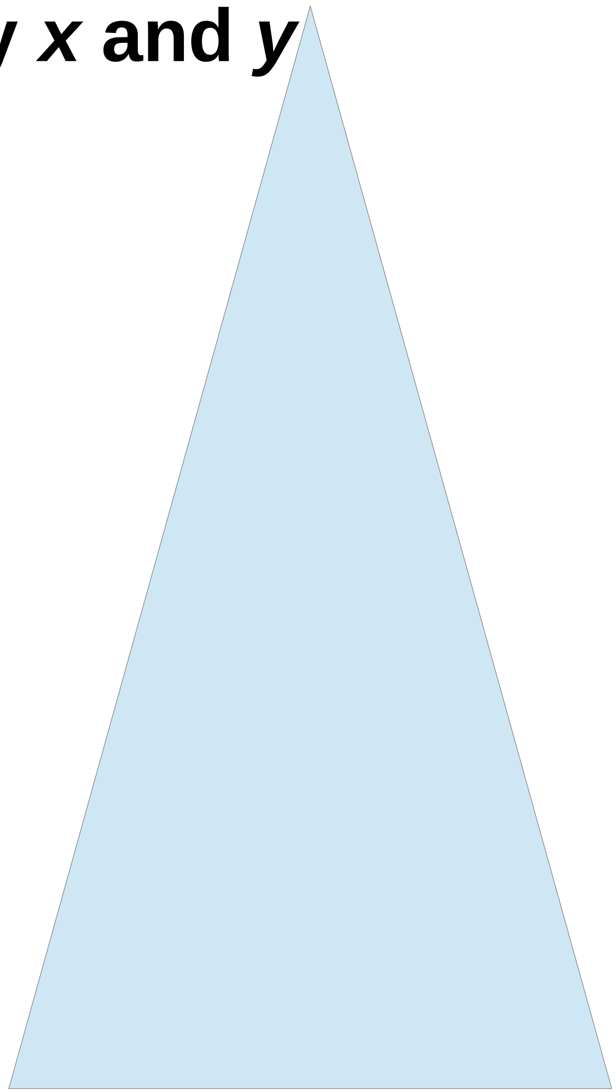
- Search – O($n^{1/d}$), where d is the number of
- dimensions
- Preprocessing time – O(n log n)
- Static!

K-d tree: an important fact

- We can represent any interval using square root of *n* nodes of the tree.

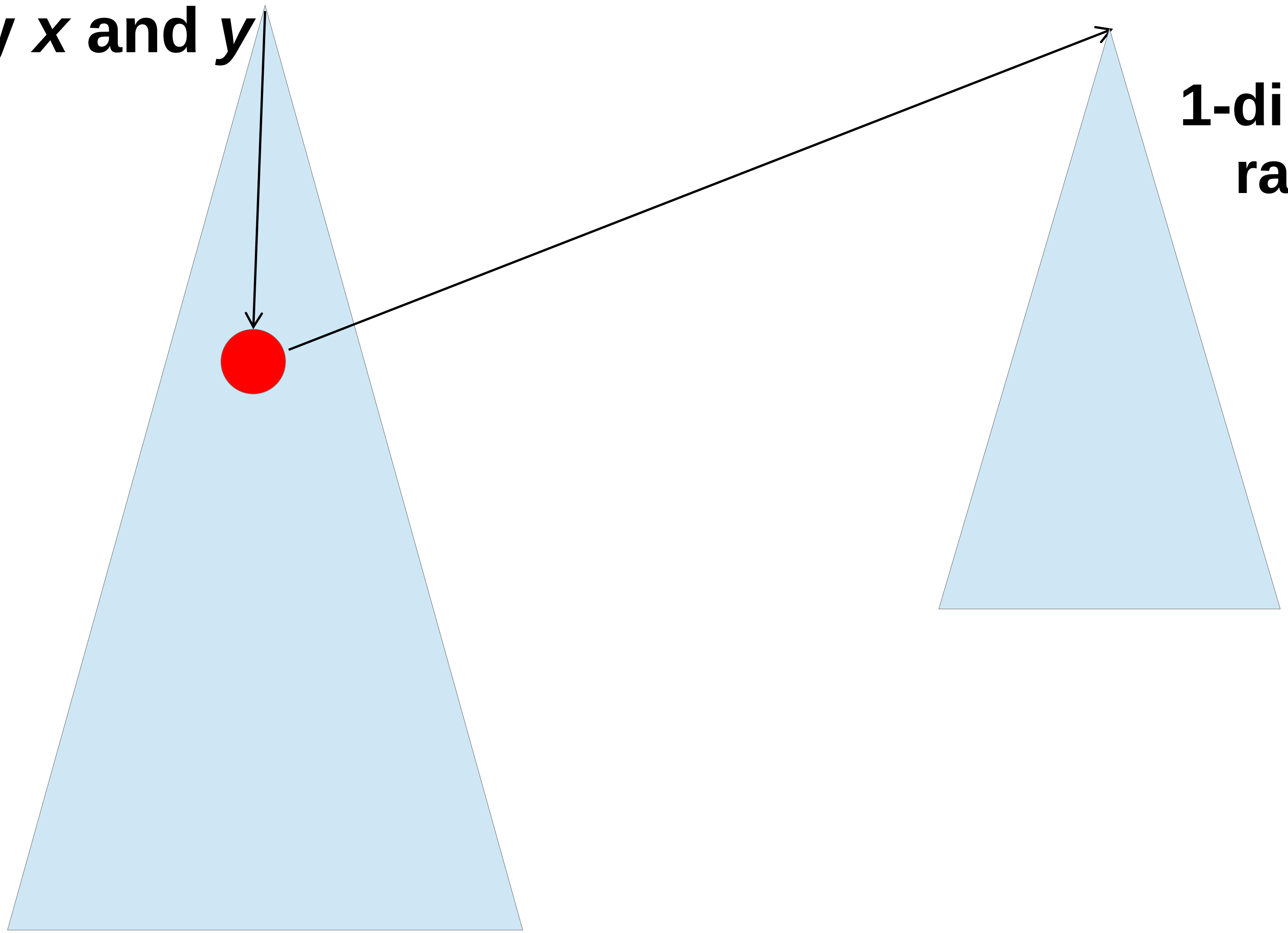
Data Structure

2-dimensional
k-d tree
by *x* and *y*



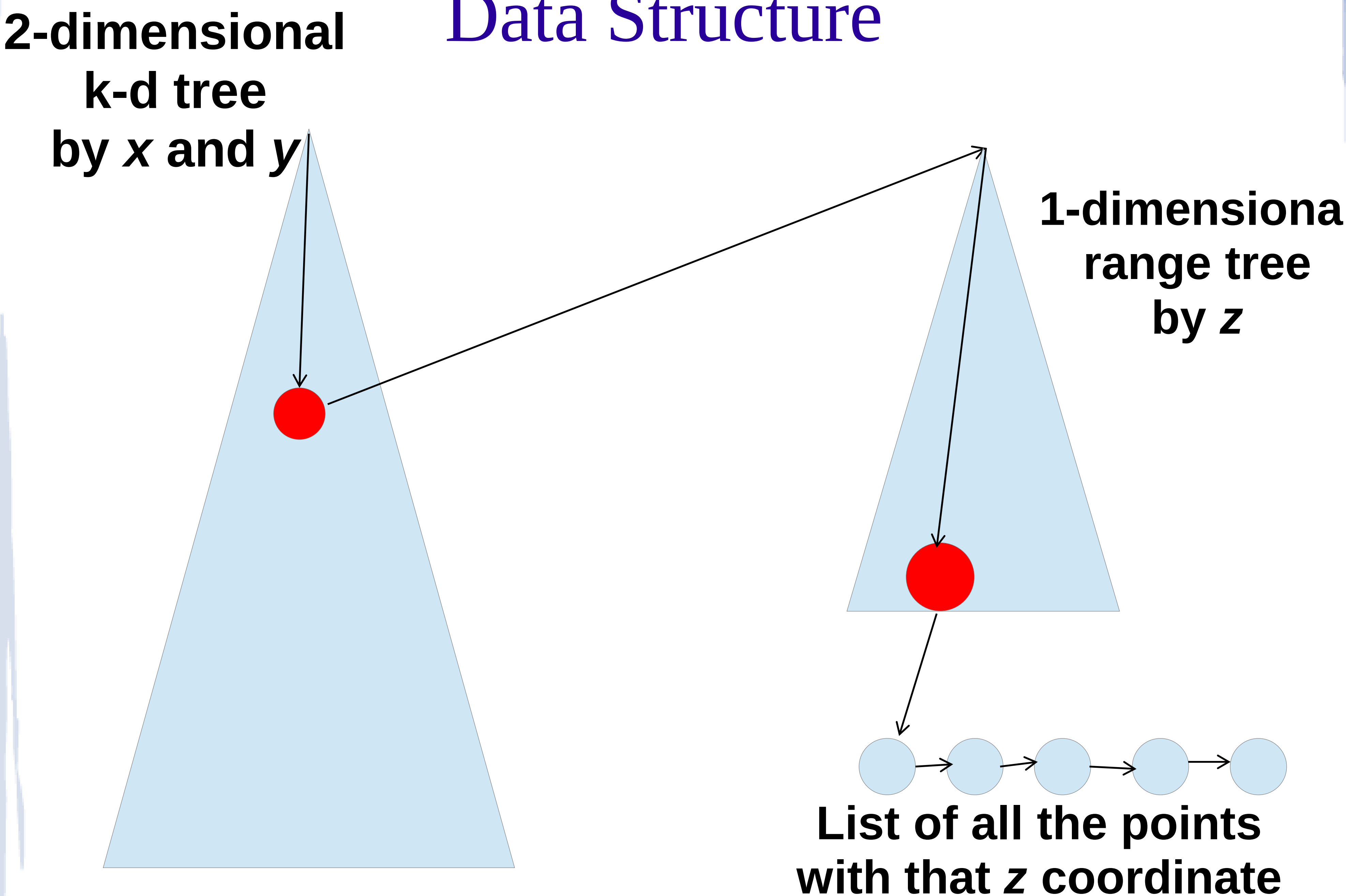
Data Structure

2-dimensional k-d tree by *x* and *y*



1-dimensional range tree by *z*

Data Structure

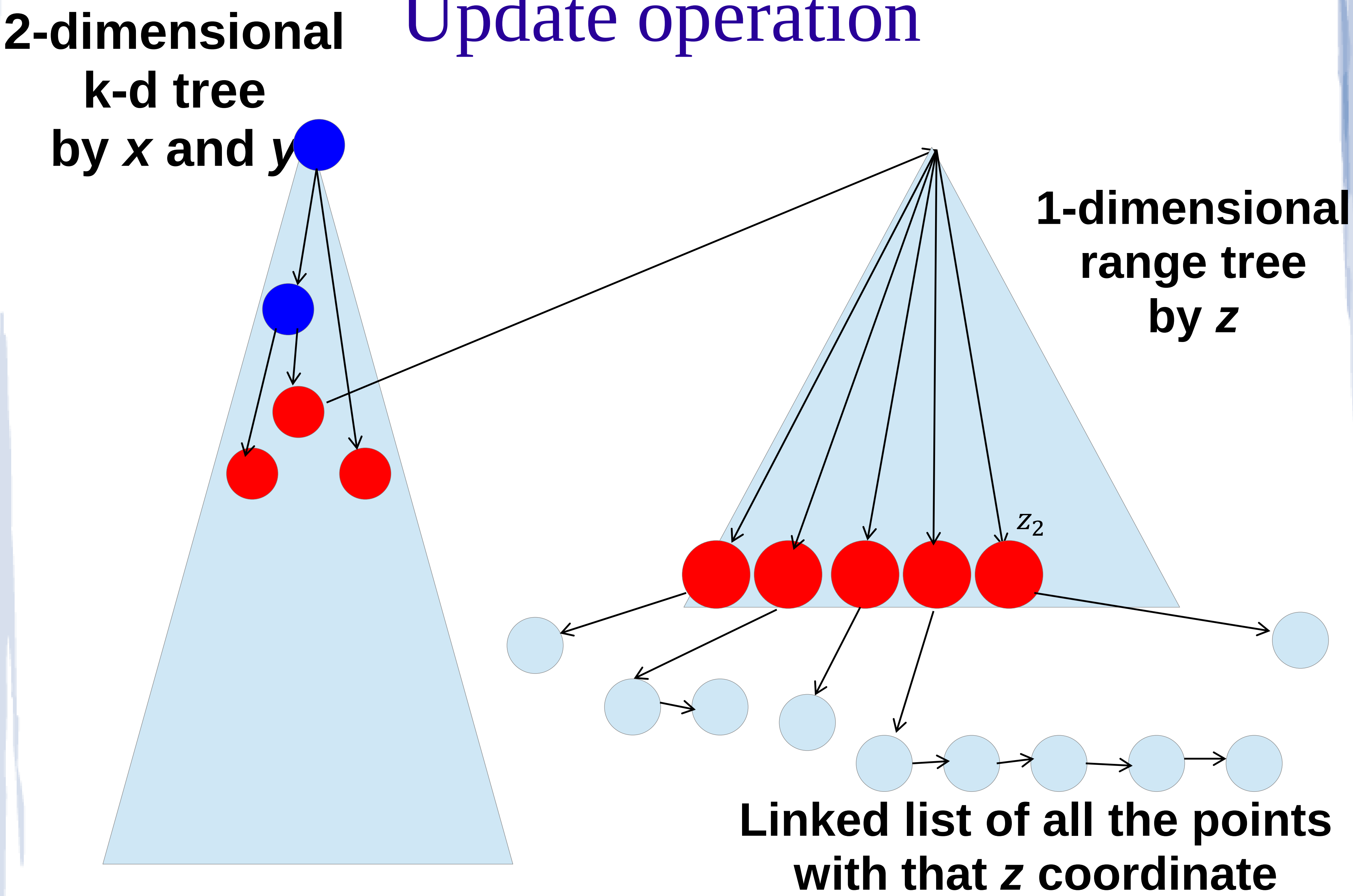


Update operation

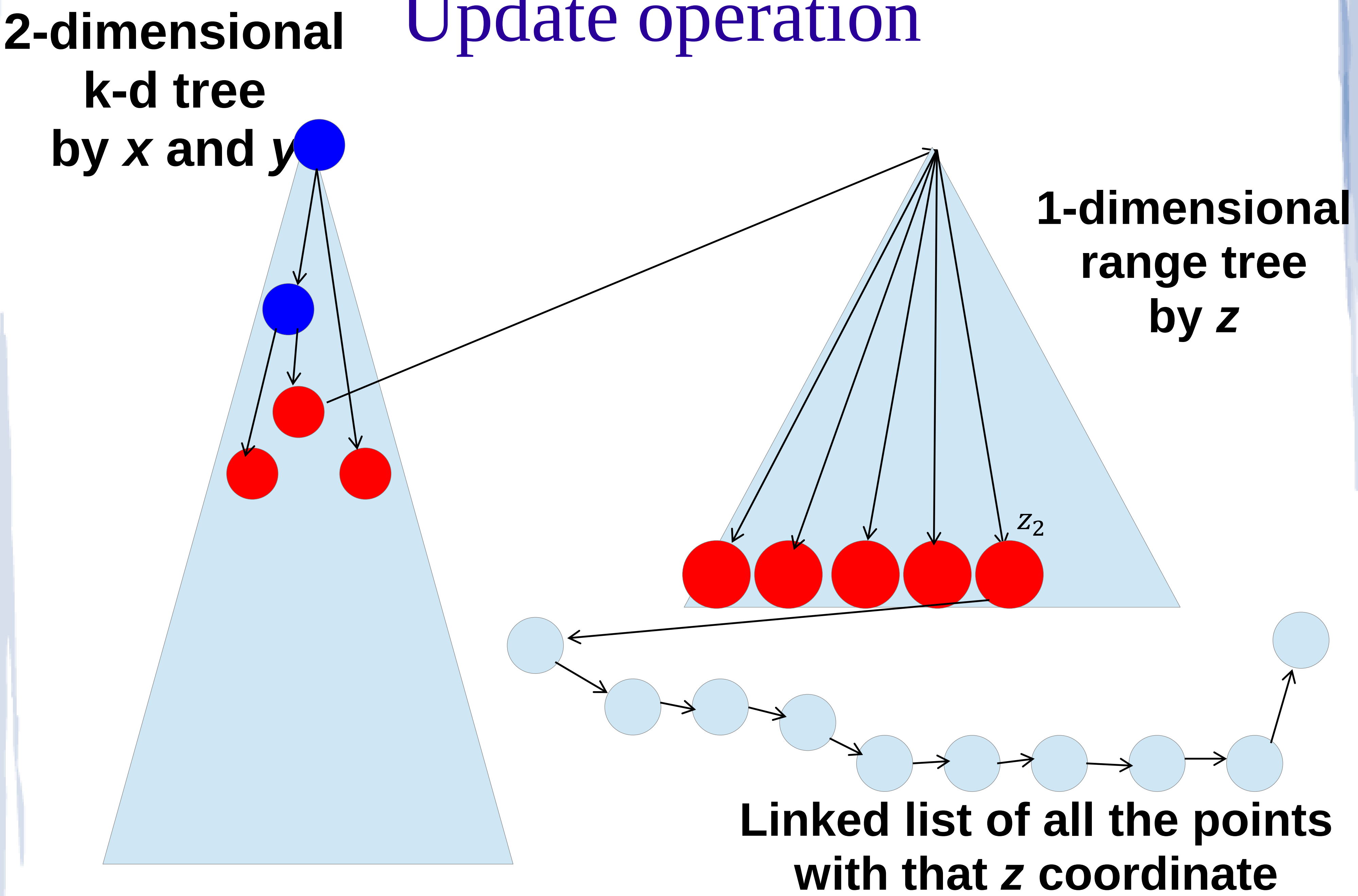
Update: For all points in the interval $[x_{1,}x_2];[y_{1,}\infty];[-\infty,z_2]$ increase their *z* coordinate to z_2.

Step 1: Divide the interval $[x_{1,}x_2];[y_{1,}\infty]$ in $n^{1/2}$ nodes of the k-d tree. For each such node and each of their ancestors, handle the range tree that corresponds to them by moving the contents of each leaf one by one.

Update operation



Update operation



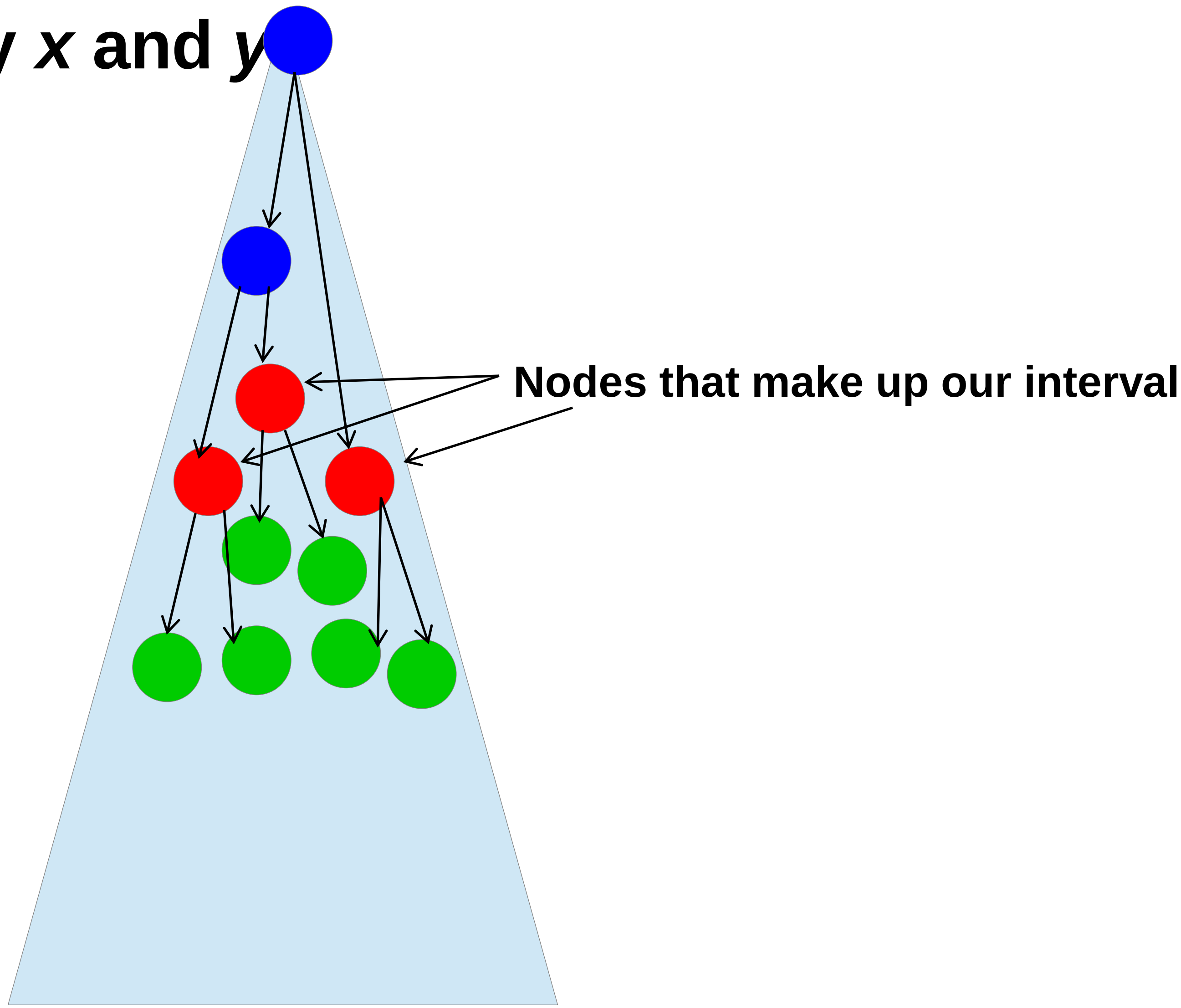
Update operation

Update: For all points in the interval $[x_{1,} x_2]; [y_{1,} \infty]; [-\infty, z_2]$ increase their *z* coordinate to z_2.

Step 2: For each of the children of these nodes, update the To Do label.

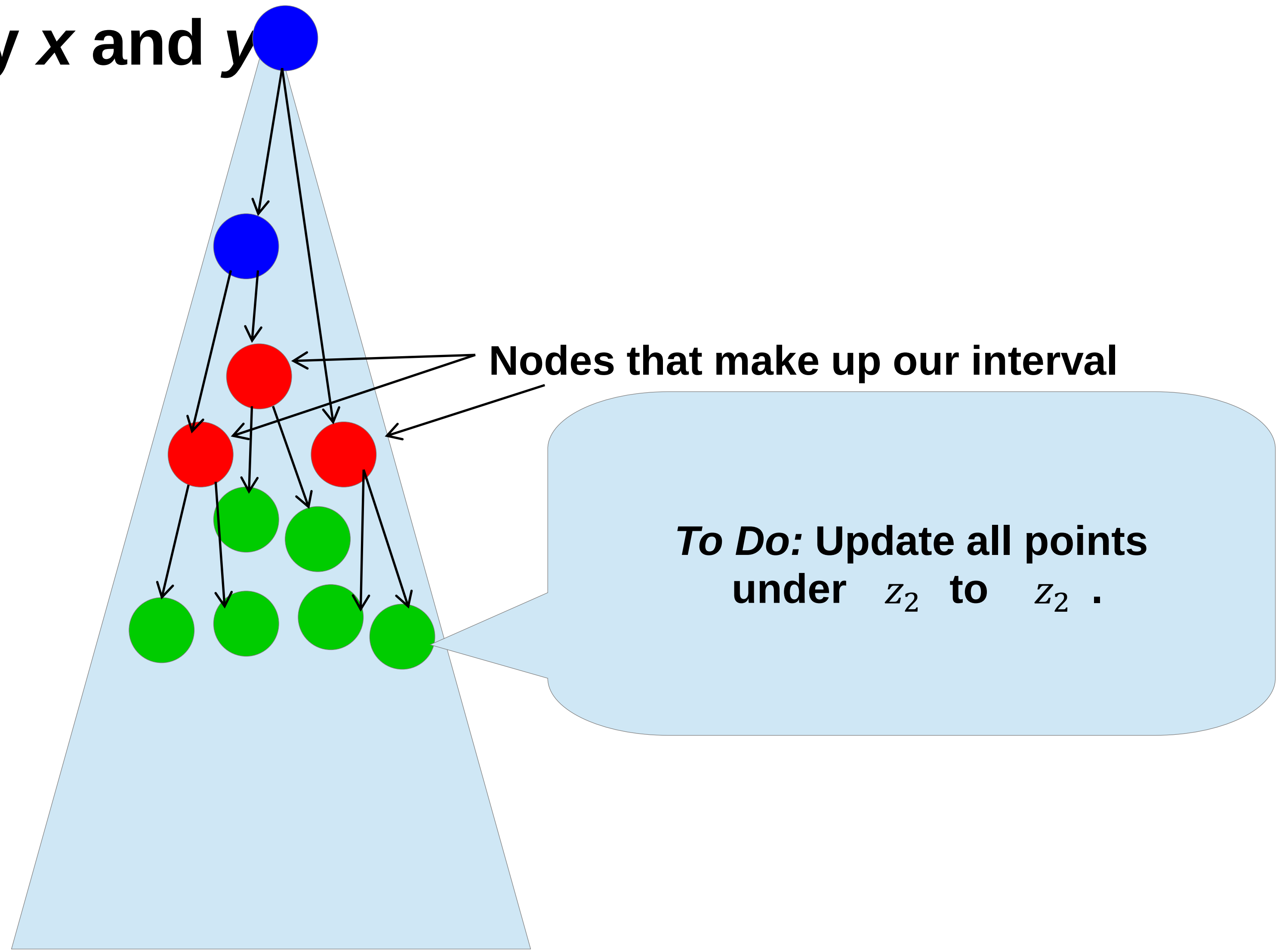
Update operation

2-dimensional
k-d tree
by *x* and *y*



Update operation

2-dimensional k-d tree by *x* and *y*



Problem

K-d trees are static, and we might need to insert new points!

Solution – the binary static to dynamic transformation

- Due to Saxe and Benteley, 1979
- O(log n) overhead for each operation

The binary static to dynamic transformation – how it works

- Suppose the static data structure is of type A.
- Keep at most *log n* data structures of type A, dividing the points in them in powers of 2.
- 19 points →
- 16 points in A1, 2 points in A2, 1 point in A3.

The binary static to dynamic transformation – how it works

- Suppose the static data structure is of type A.
- Keep at most *log n* data structures of type A, dividing the points in them in powers of 2.
- 19 points →
- 16 points in A1, 2 points in A2, 1 point in A3.

The binary static to dynamic transformation
Insertion of a new point

- Suppose we insert a new point now:
- Destroy some of the data structures as necessary to create a new one with all of their points plus the new one.

The binary static to dynamic transformation
Insertion of a new point

- Suppose we insert a new point now:
- Destroy some of the data structures as necessary to create a new one with all of their points plus the new one.

The binary static to dynamic transformation
Insertion of a new point

- Suppose we insert a new point now:
- Destroy some of the data structures as necessary to create a new one with all of their points plus the new one.

The binary static to dynamic transformation
Insertion of a new point

- Suppose we insert a new point now:
- Destroy some of the data structures as necessary to create a new one with all of their points plus the new one.

Notation
The Point Projection Problem

Total number of points	N
Number of update operations	M
Number of points reported in a query	K

Results
The Point Projection Problem

	2D case	3D case
Query	$O(\log N \log(N+M)+K)$	$O(\sqrt{N}\log_{(} N+M)+K)$
Insertion	$O(\log^2 N \log(N+M))$	$O(\sqrt{N}\log^2 N \log(N+M))$
Update	$O(\log^2 N \log(N+M))$	$O(\sqrt{N}\log^2 N \log(N+M))$
Storage	$O(N \log N \log(N+M))$	$O(N \log N \log(N+M))$
Preprocessing time	$O(N \log^2 N \log(N+M))$	$O(N\sqrt{N}\log^2 N \log(N+M))$

Notation
The Segment Intersection Problem

Number of segments	n
Number of points reported in a query	k

Results
The Segment Intersection Problem

	Static	Dynamic Trivial solution 1	Dynamic Trivial solution 2	Dynamic Our solution
Query	$O((k+1)\log n \log^* n)$	$O(\log^2 n + k)$	$\Omega(n \log n)$	$O(\sqrt{n}\log n + k \log^3 n)$
Insertion	Not supported	$\Omega(n \log^3 n)$	$O(1)$	$O(\sqrt{n}\log^3 n)$
Storage	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log n)$	$O(n \log^2 n)$
Pre-processing time	$O(n \log n)$	$O(n \log^2 n)$	$O(n)$	$O(n\sqrt{n}\log^3 n)$

Conclusions

- Goal: designing a data structure that supports a set of axis-aligned segments and the following operations on them:
  - Insert a new segment
  - Report all intersection points of pairs of segments inside a query rectangle

Conclusions

- Goal: designing a data structure that supports a set of axis-aligned segments and the following operations on them:
  - Insert a new segment
  - Report all intersection points of pairs of segments inside a query rectangle
- Results:
  - sublinear time complexity of both operations, subquadratic storage space and subquadratic preprocessing time

Future work

- Getting rid of the square root of n factor

Future work

- Getting rid of the square root of n factor
- Handling axis-aligned rectangles

Future work

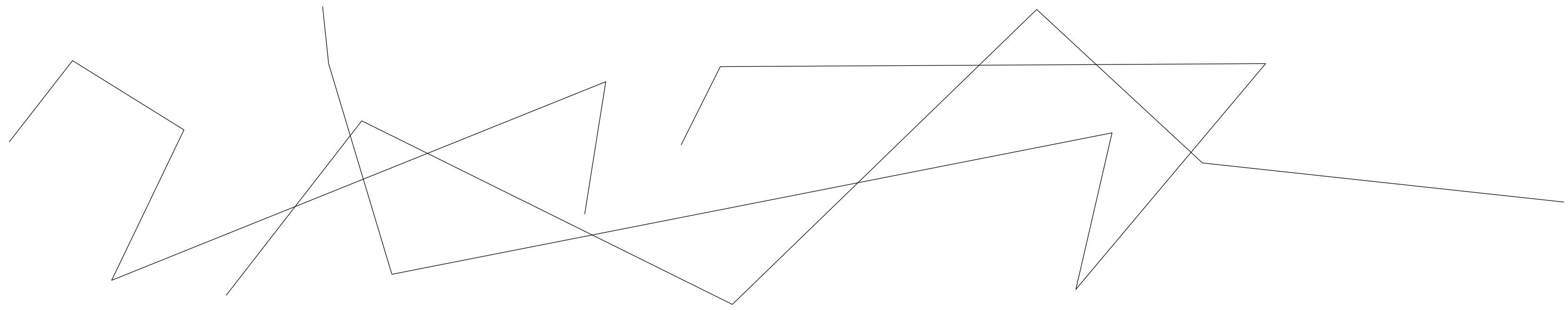
- Getting rid of the square root of n factor
- Handling axis-aligned rectangles
- Handling deletions of segments

Future work
Long-term goals

- Handling segments with arbitrary orientation

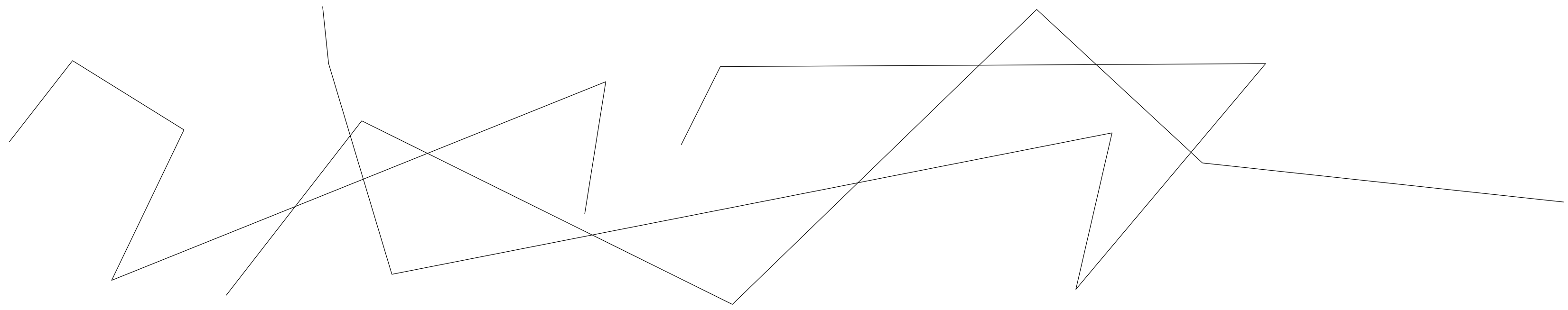
Future work
Long-term goals

- Handling segments with arbitrary orientation
- Handling sequences of segments



Future work
Long-term goals

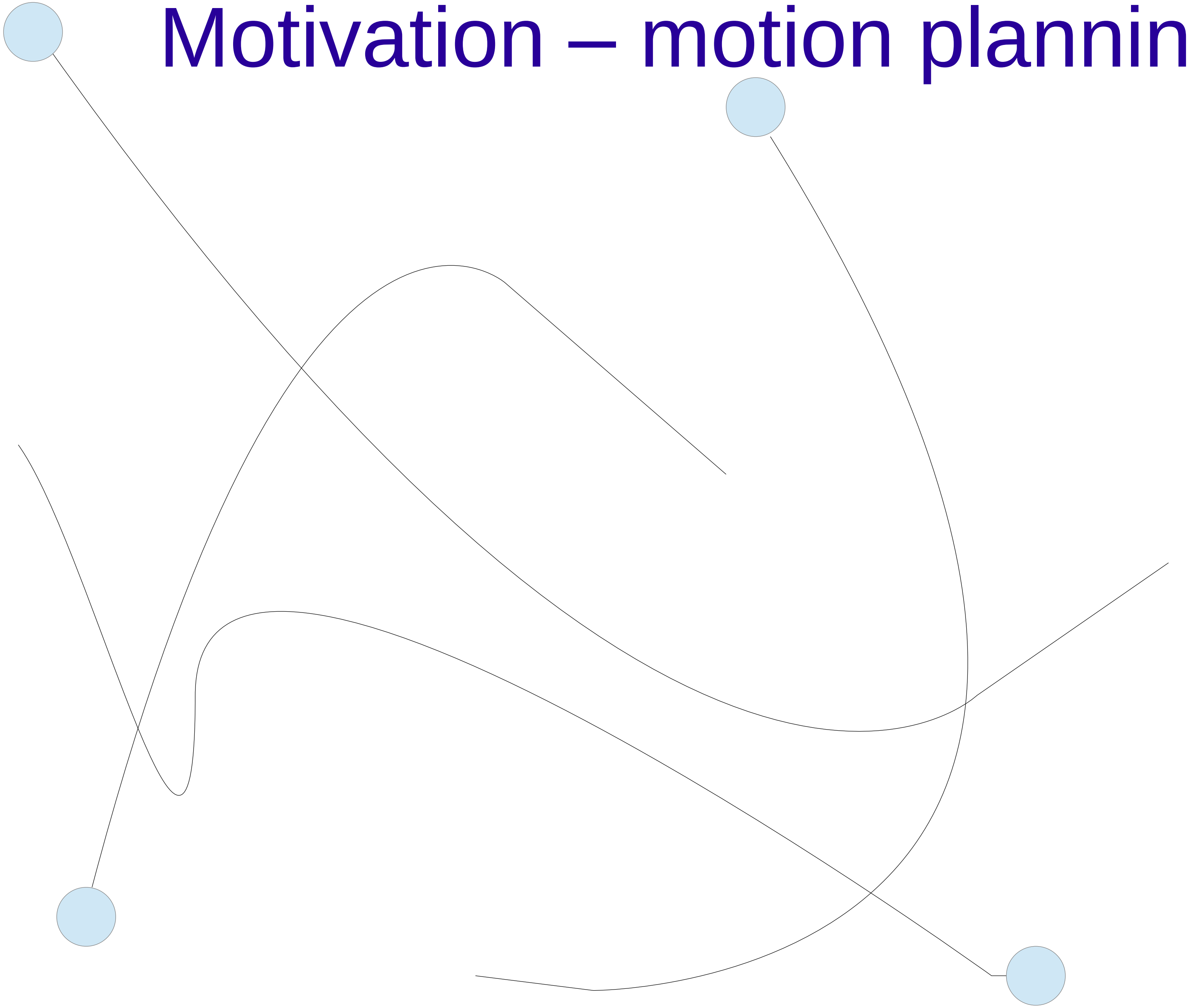
- Handling segments with arbitrary orientation
- Handling sequences of segments



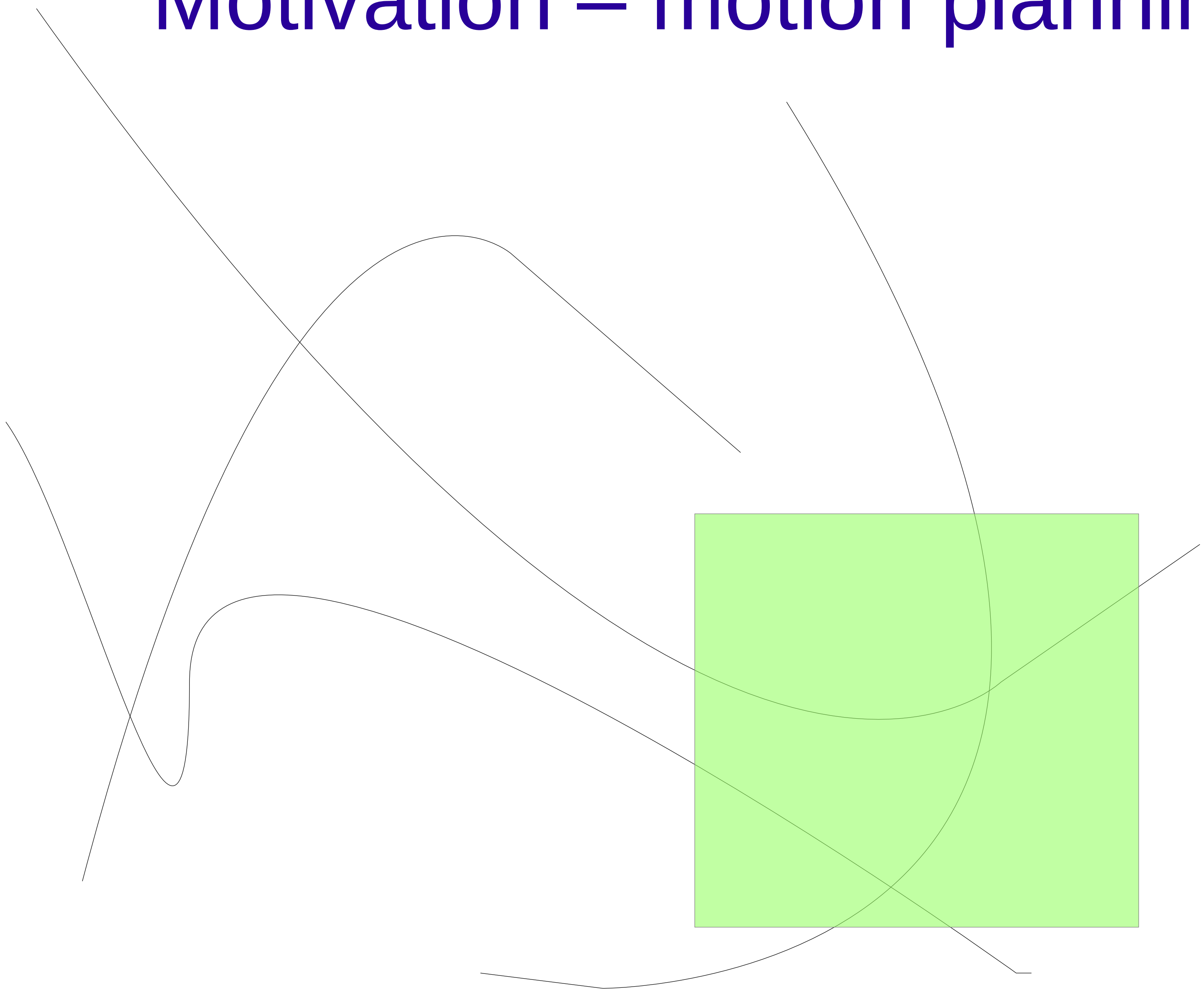
- Handling intersections of curves

Thank you for your attention!

Motivation – motion planning



Motivation – motion planning

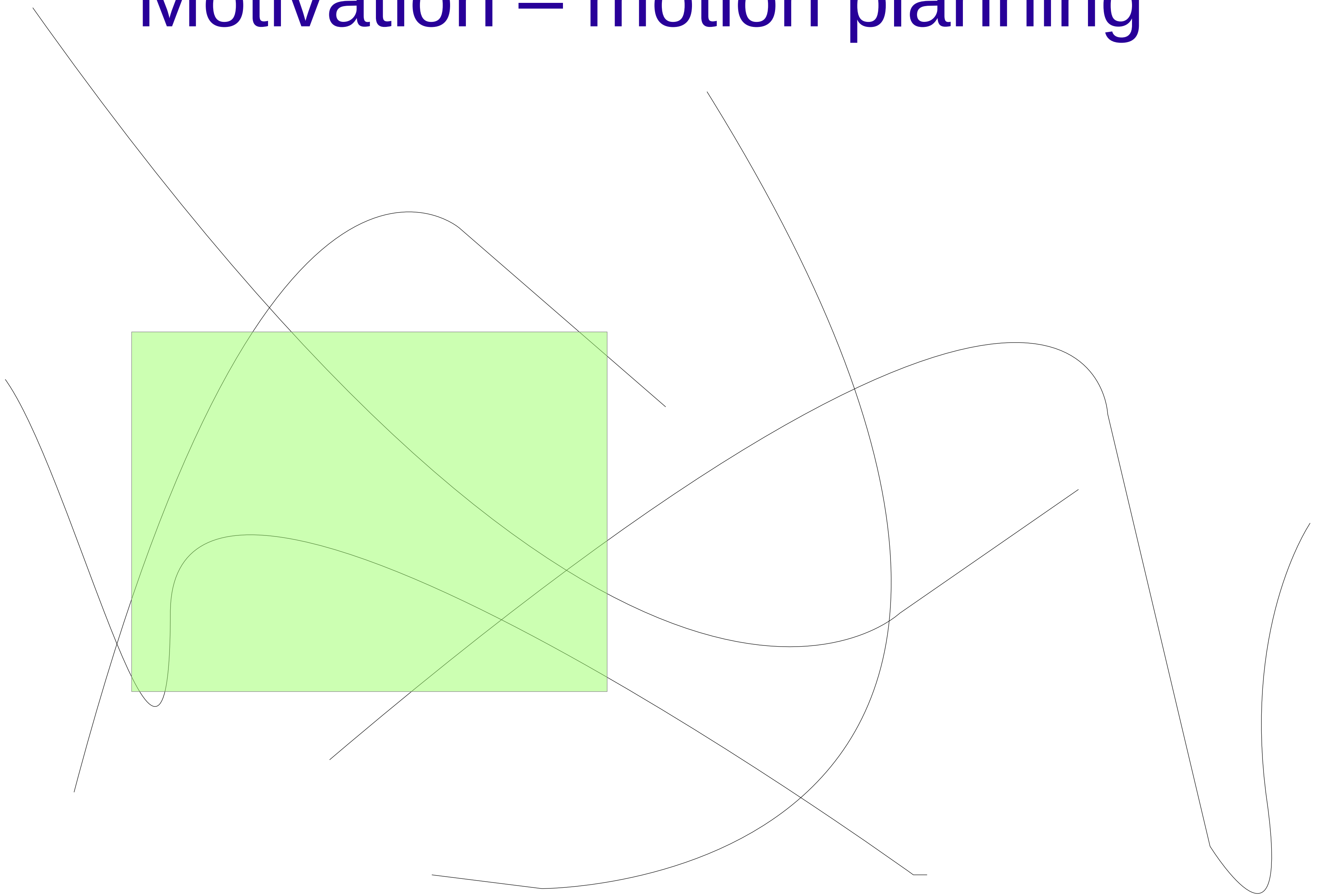


Motivation – motion planning

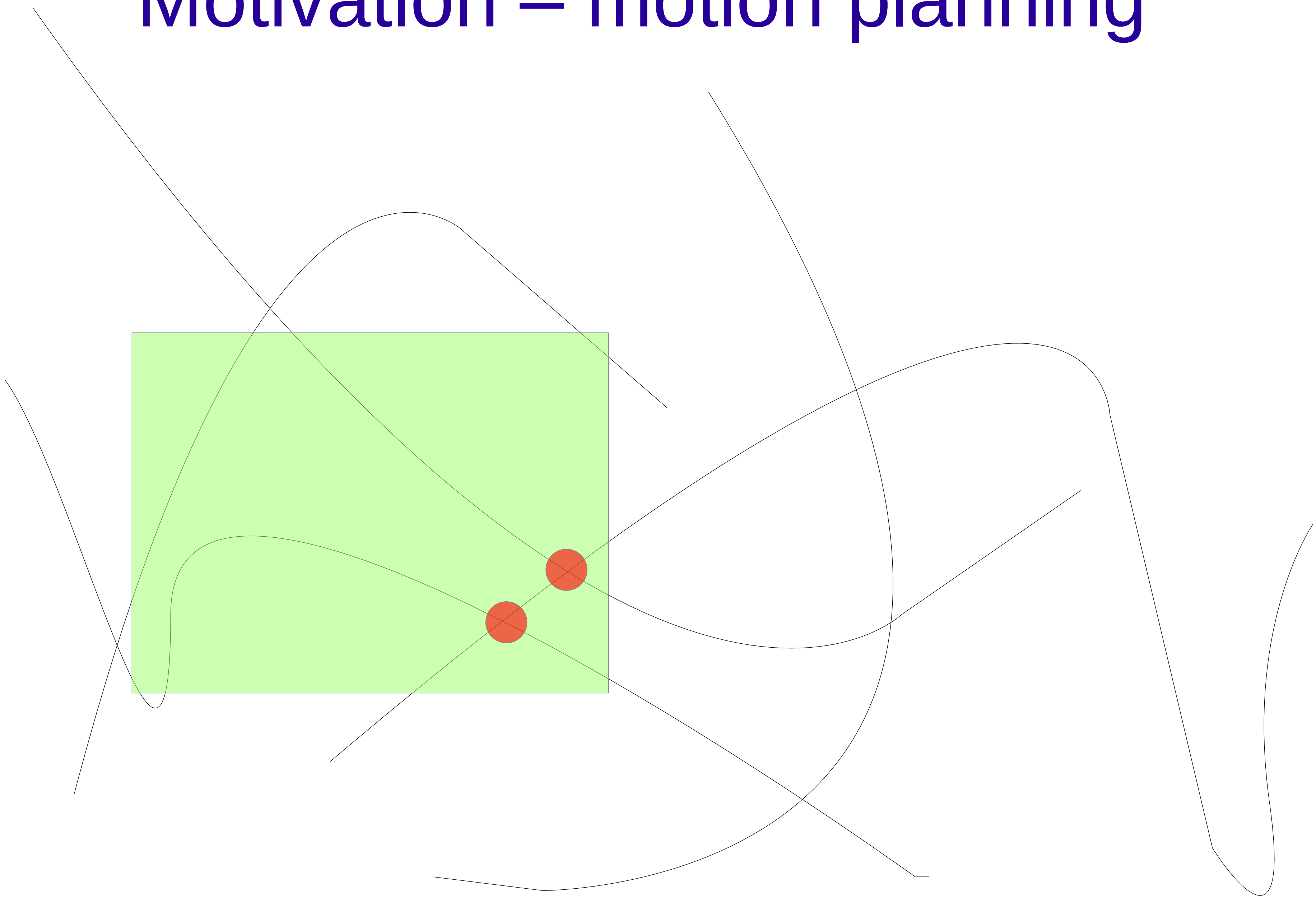
Motivation – motion planning

Motivation – motion planning

Motivation – motion planning

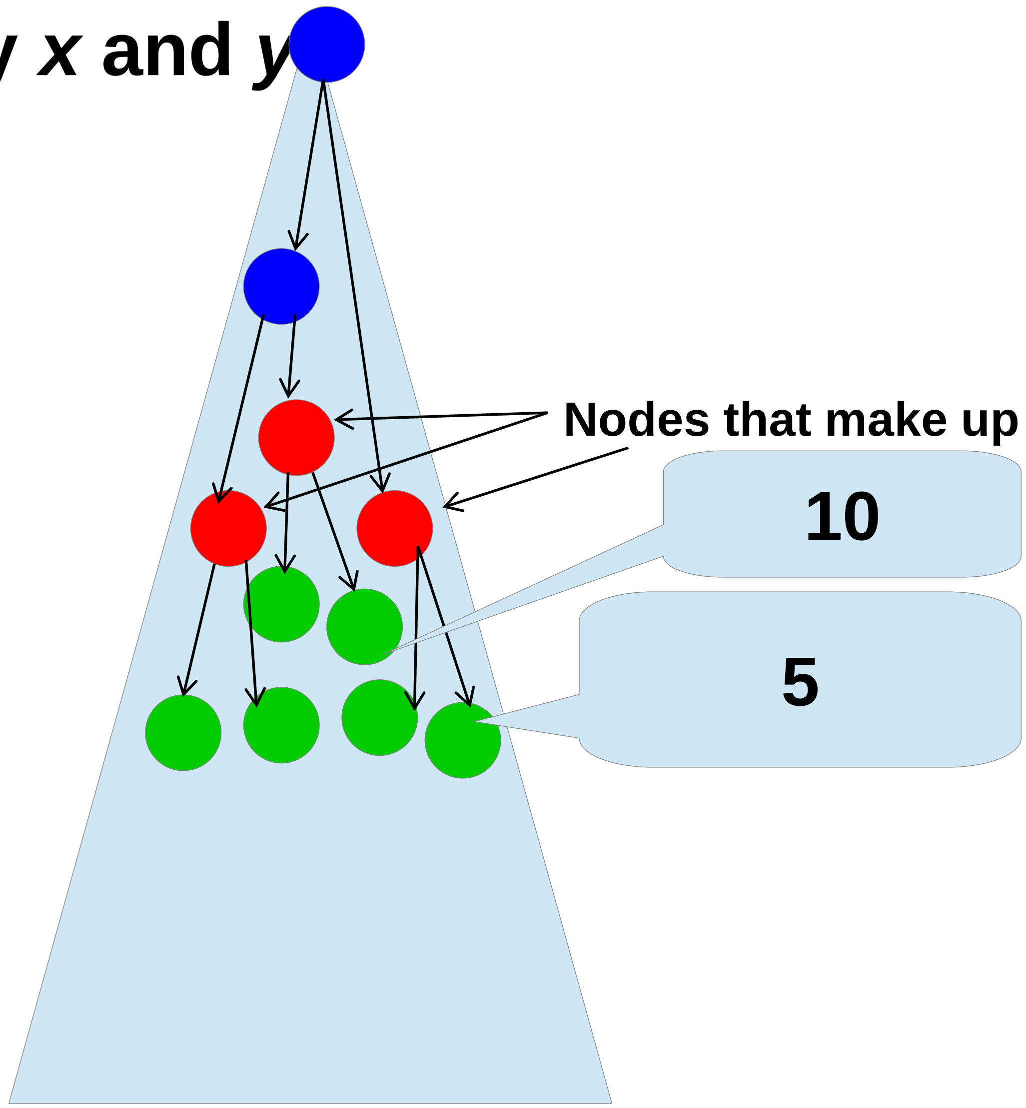


Motivation – motion planning



Update operation

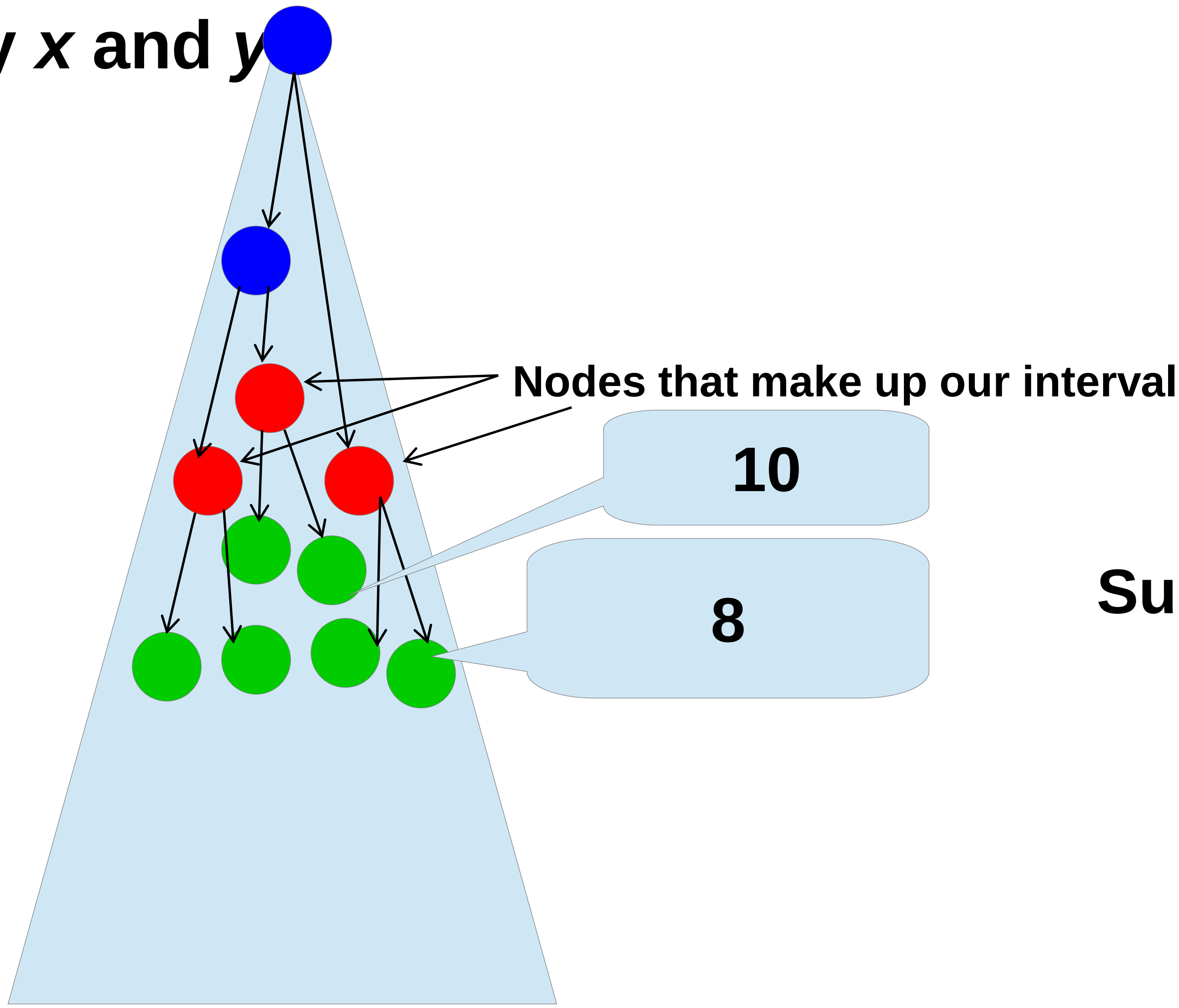
2-dimensional k-d tree by *x* and *y*



Suppose z_2 = 8

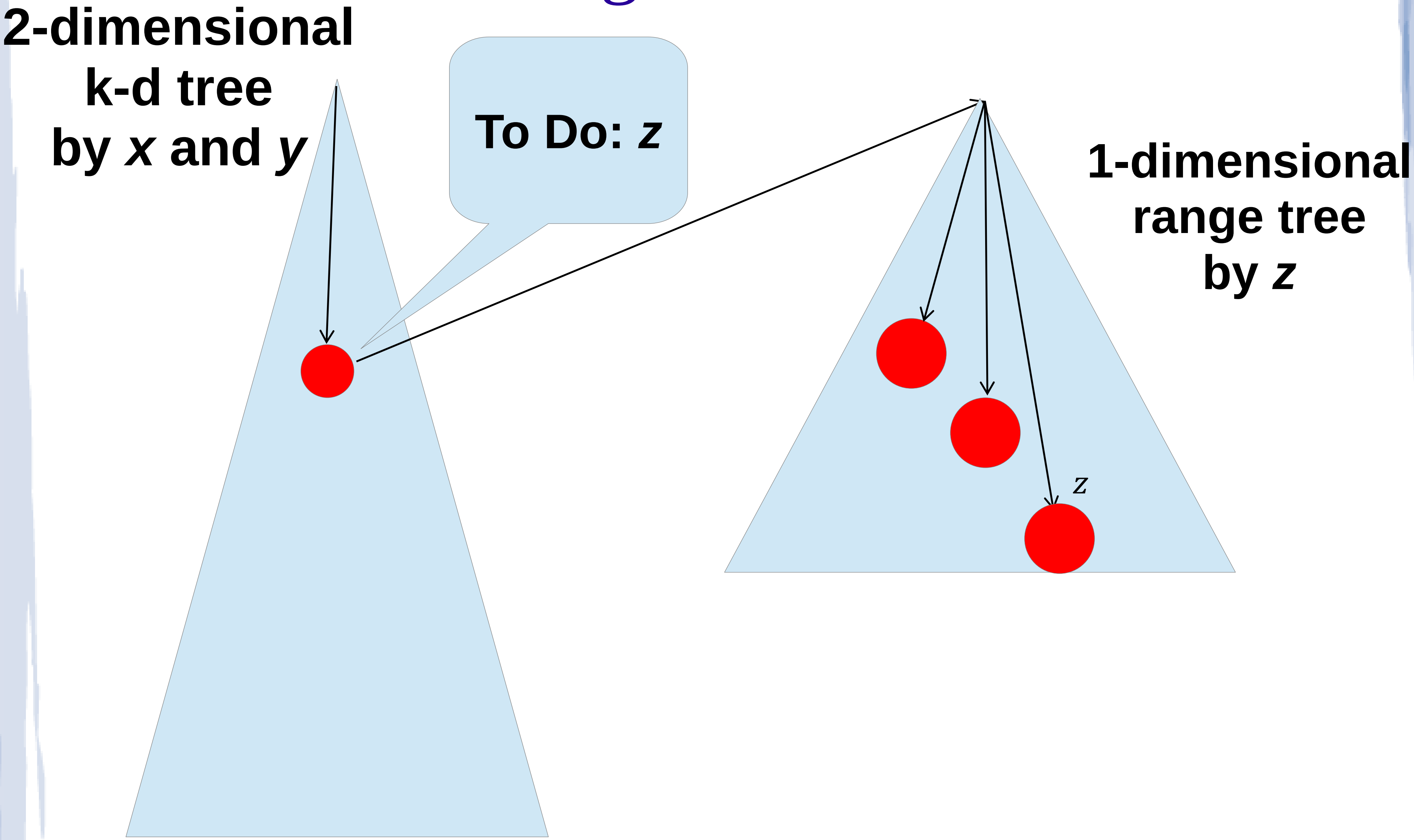
Update operation

2-dimensional k-d tree by *x* and *y*



Suppose z_2 **= 8**

Handling *To Do* labels



Handling *To Do* labels