# Adaptive Data Structures for Permutations and Binary Relations

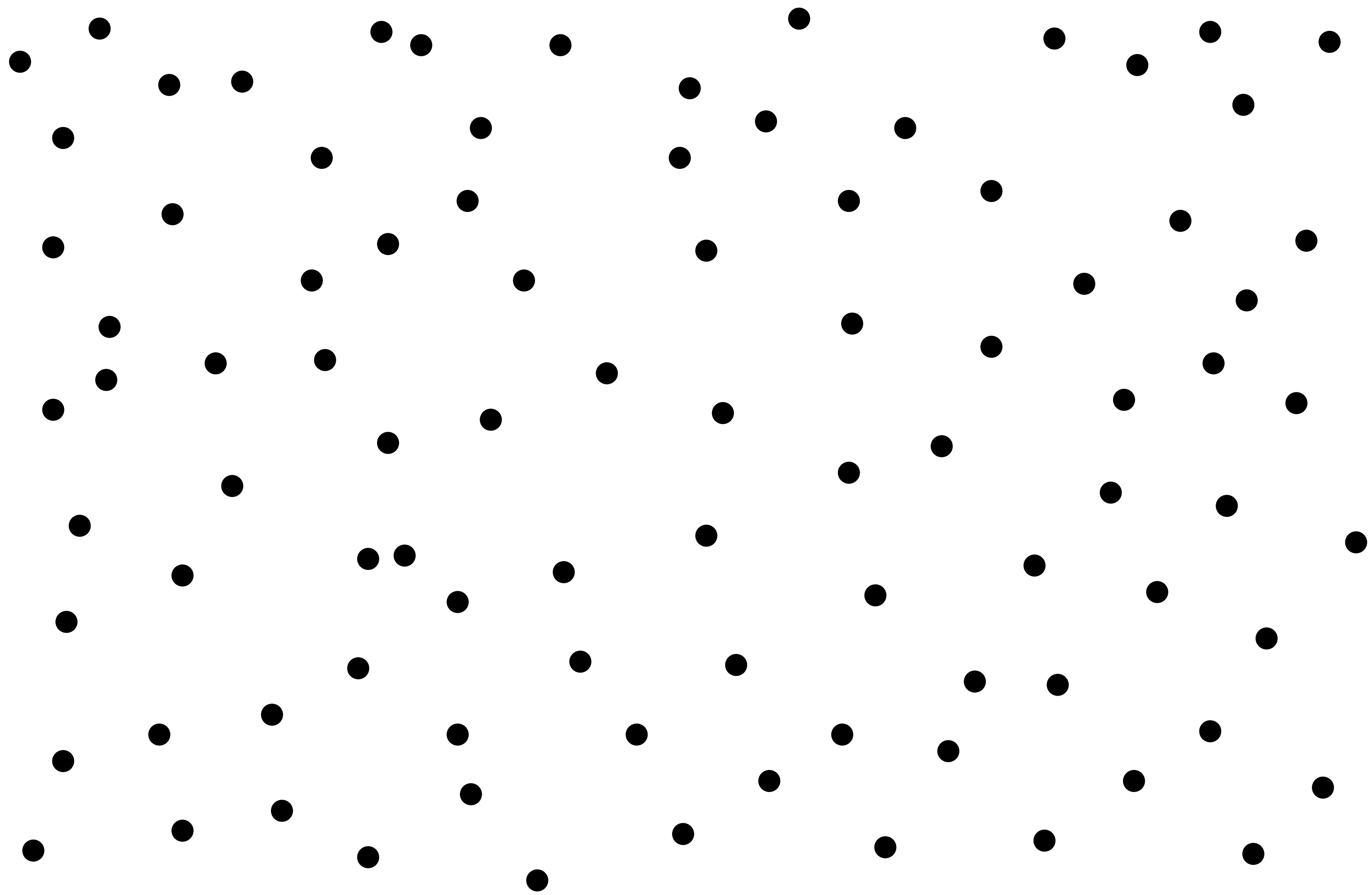
Francisco Claude and J. Ian Munro

NII Shonan Seminar #29, 28 September 2013

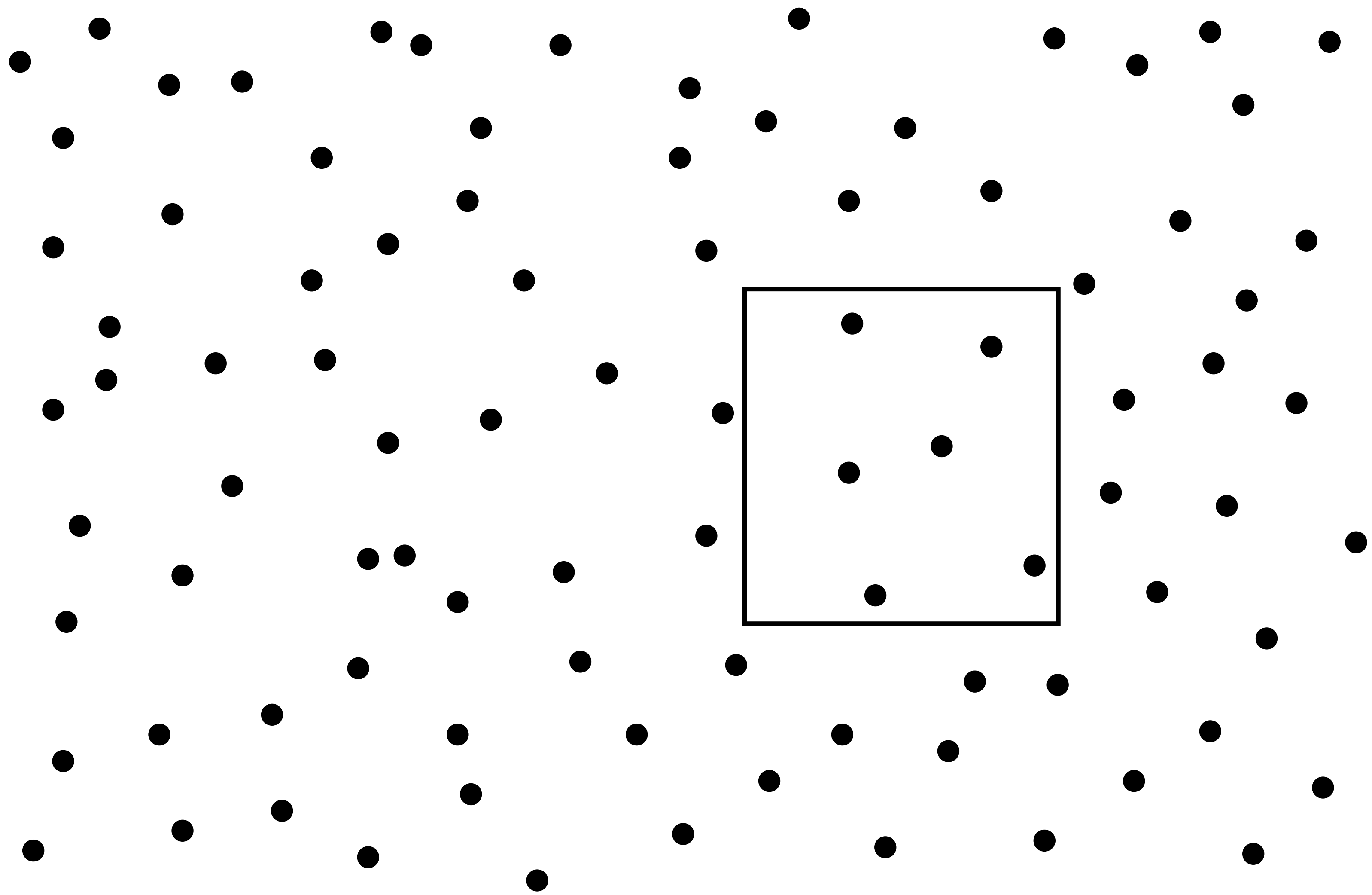
# Related Work

- Range Searching - joint work with Diego Arroyuelo, Reza Dorrigiv, Stephane Durocher, Meng He, Alejandro López-Ortiz, J. Ian Munro, Patrick Nicholson, Alejandro Salinger, and Matthew Skala.

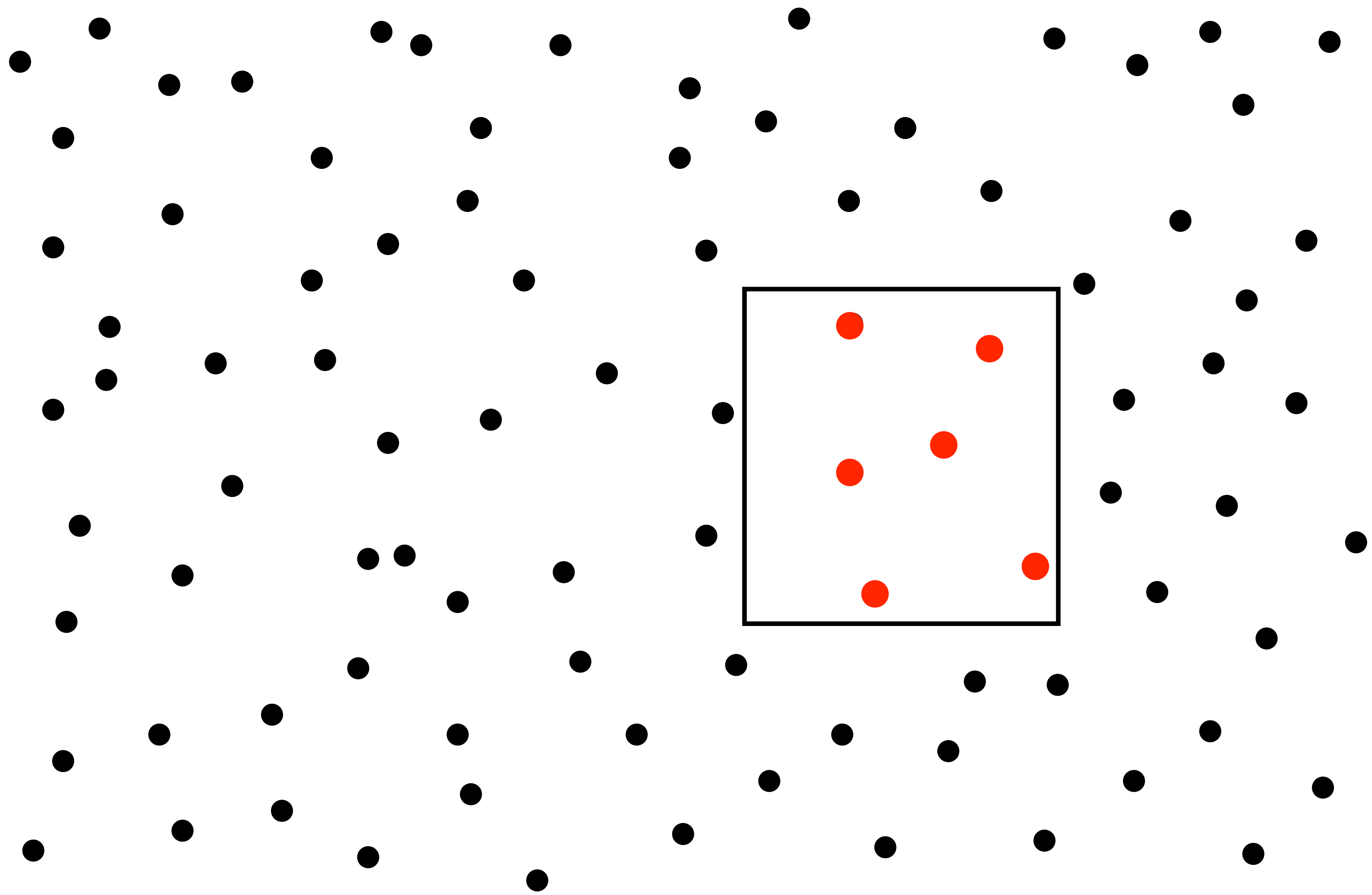
# Adaptive Range Searching



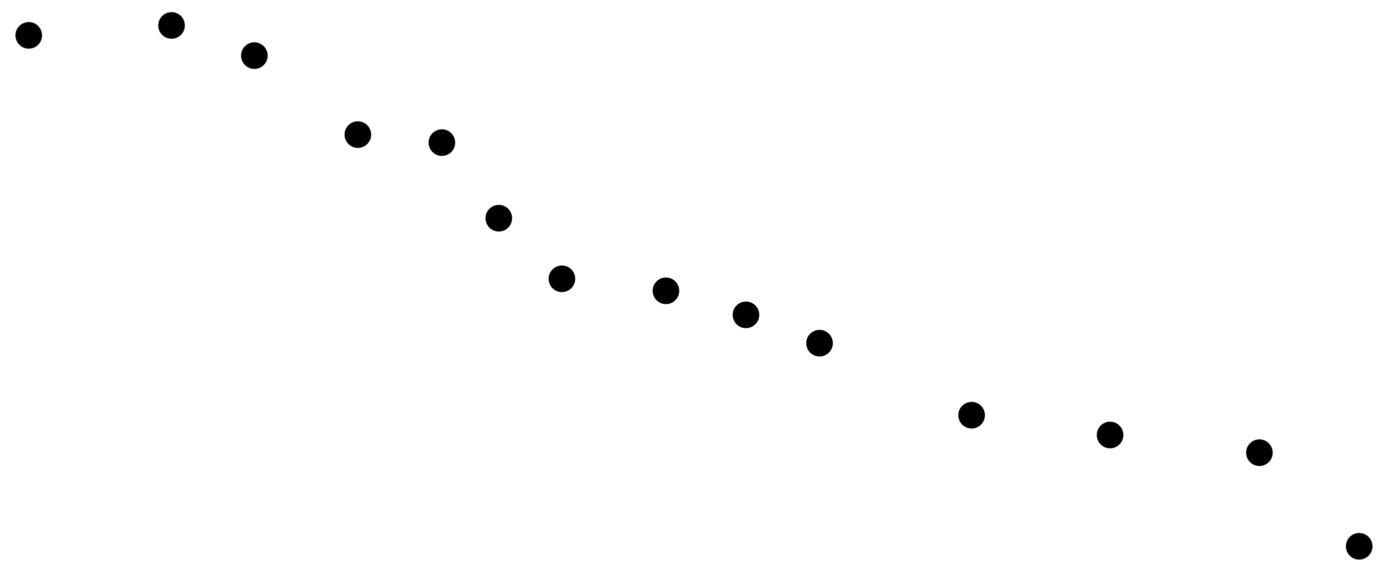
# Adaptive Range Searching



# Adaptive Range Searching

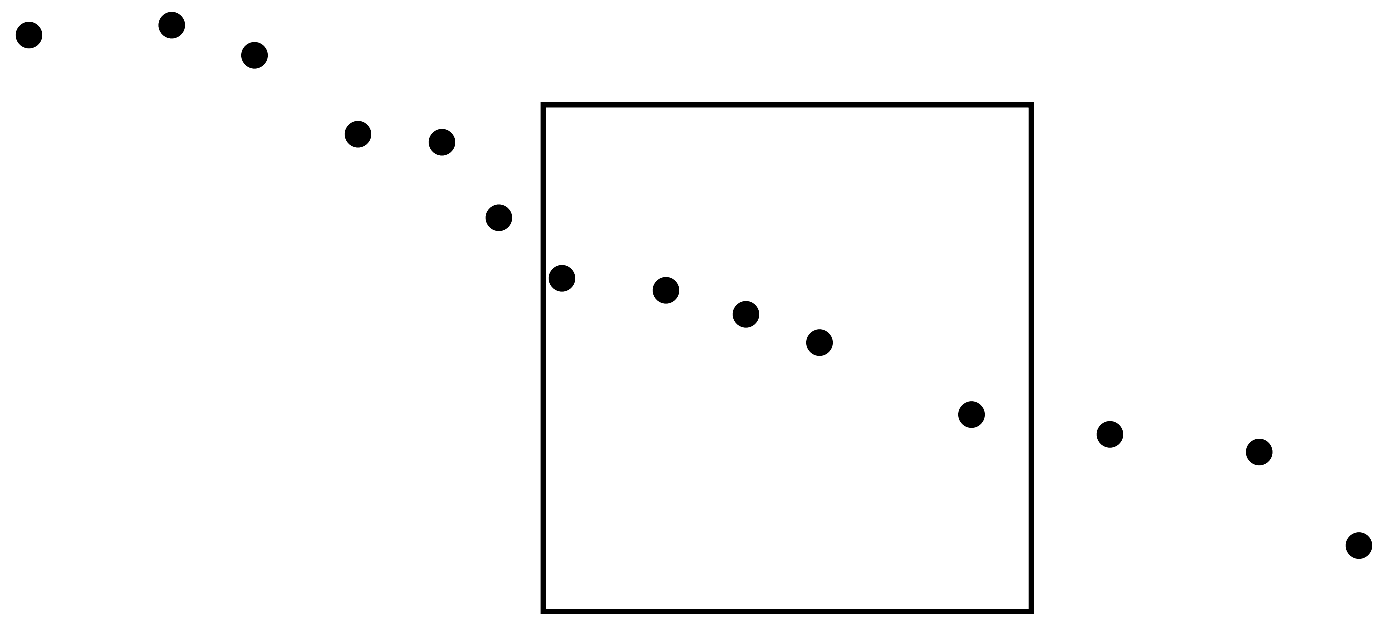


# Adaptive Idea



$O(\lg n)$ search time

# Adaptive Idea

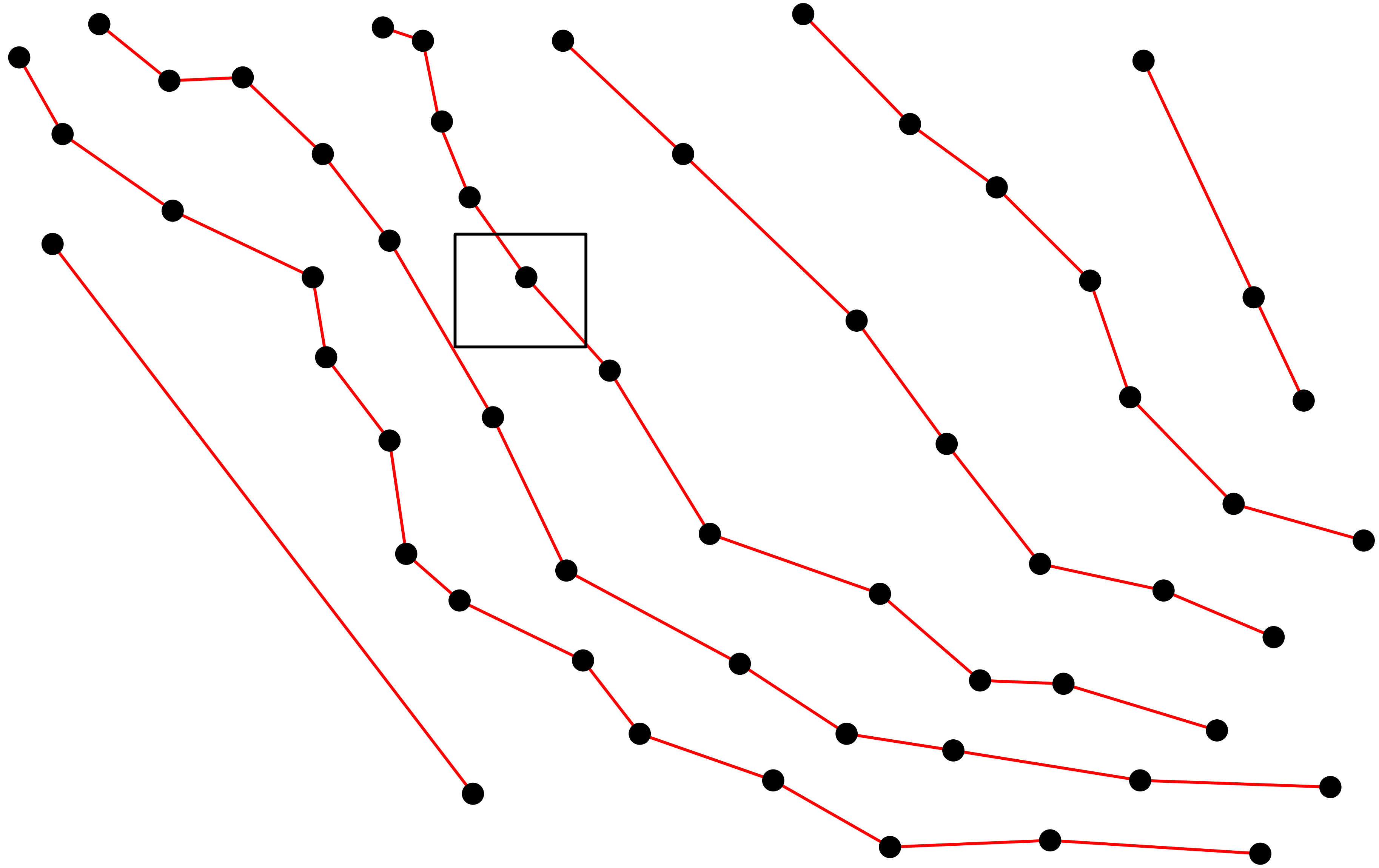


$O(\lg n)$ search time

# Idea

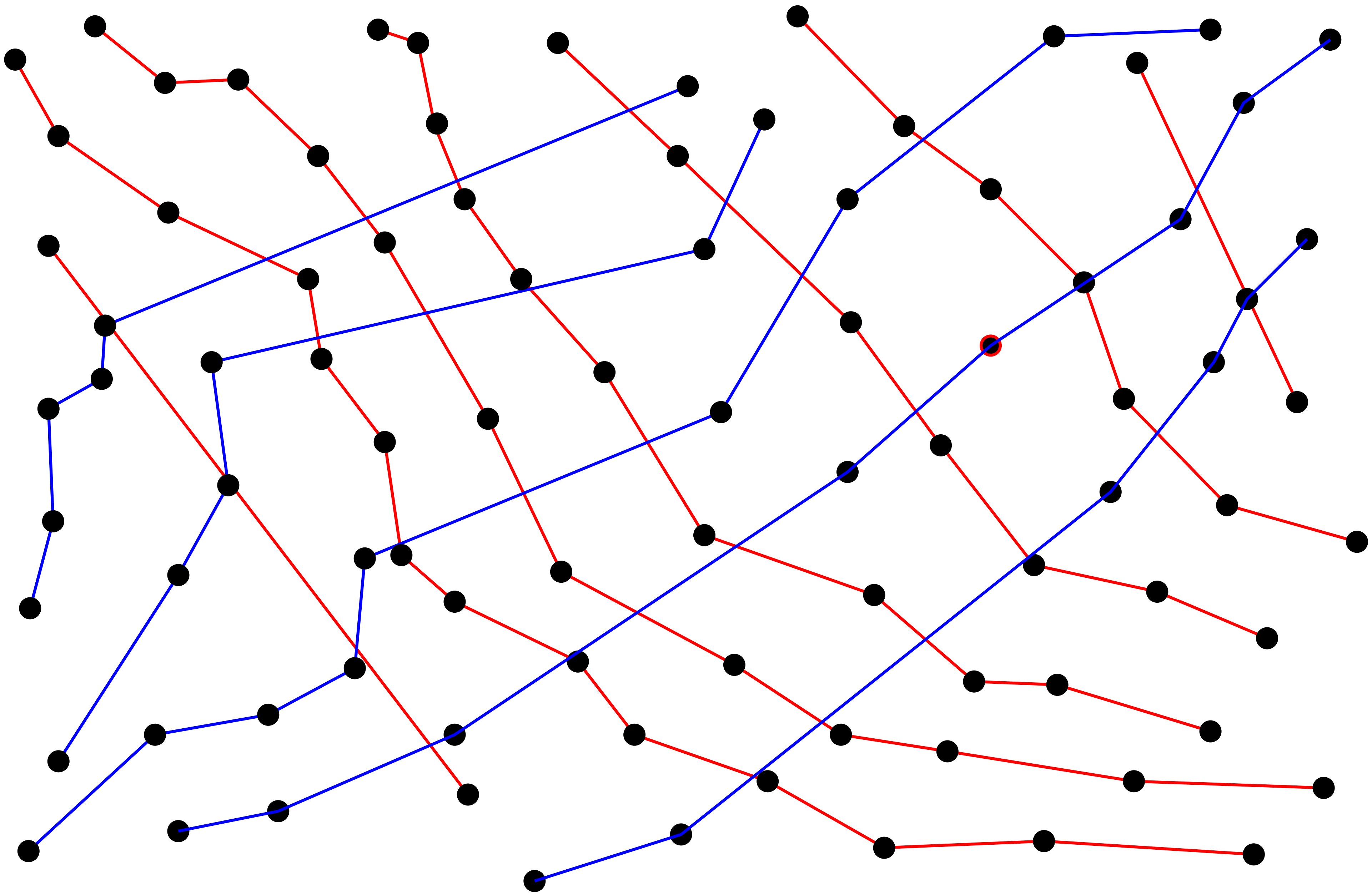
- Split the points into k descending chains
- Search inside each chain
- $O(k \lg n)$ time

# We can improve



$$O(\lg k \lg n + k' \lg n)$$

# And...



at most $c\sqrt{n}$ chains

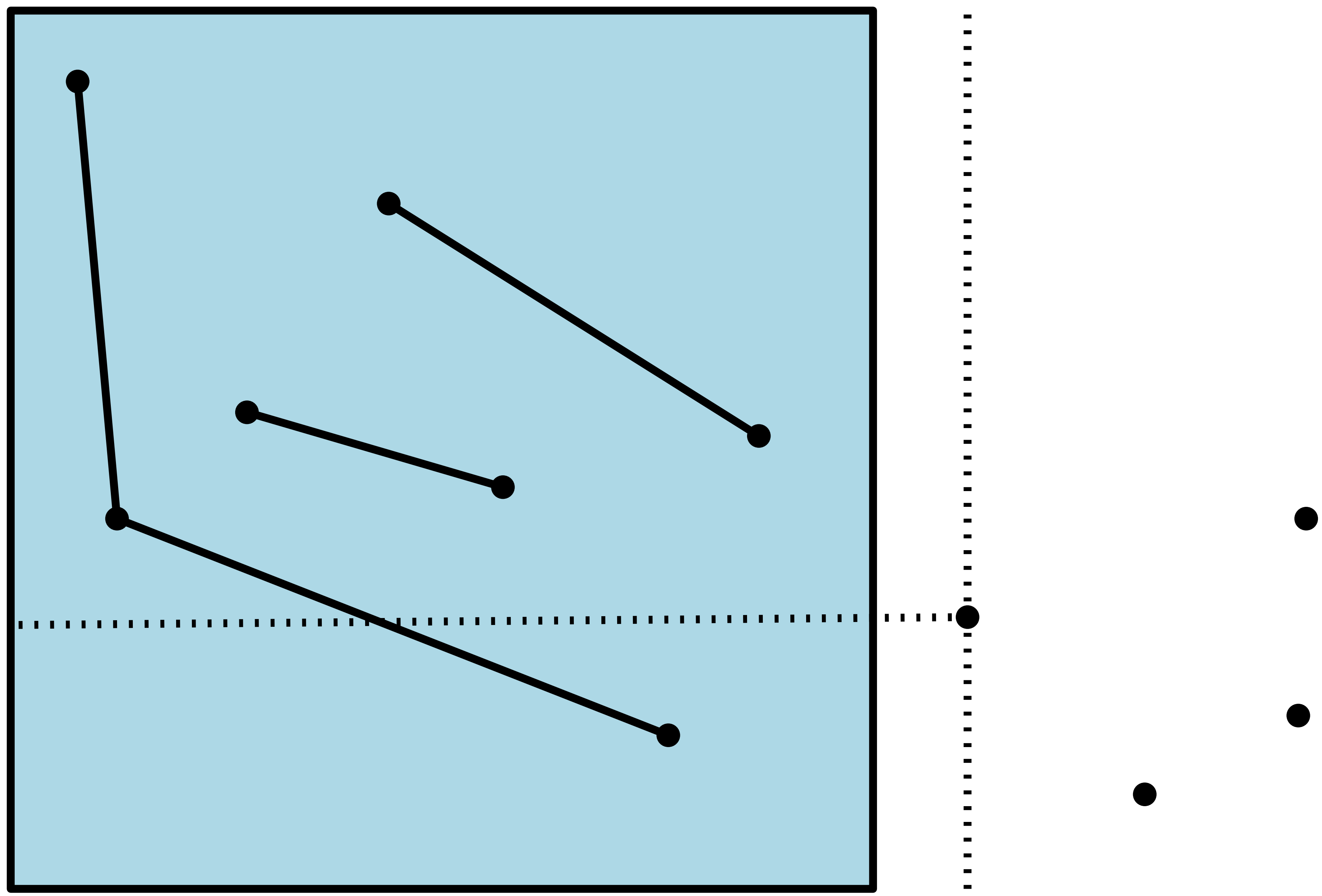
# How to build this?

- Separating chains is NP-Hard
- We can approximate
- Generate the sets (ascending and descending) and work separately on each one of them

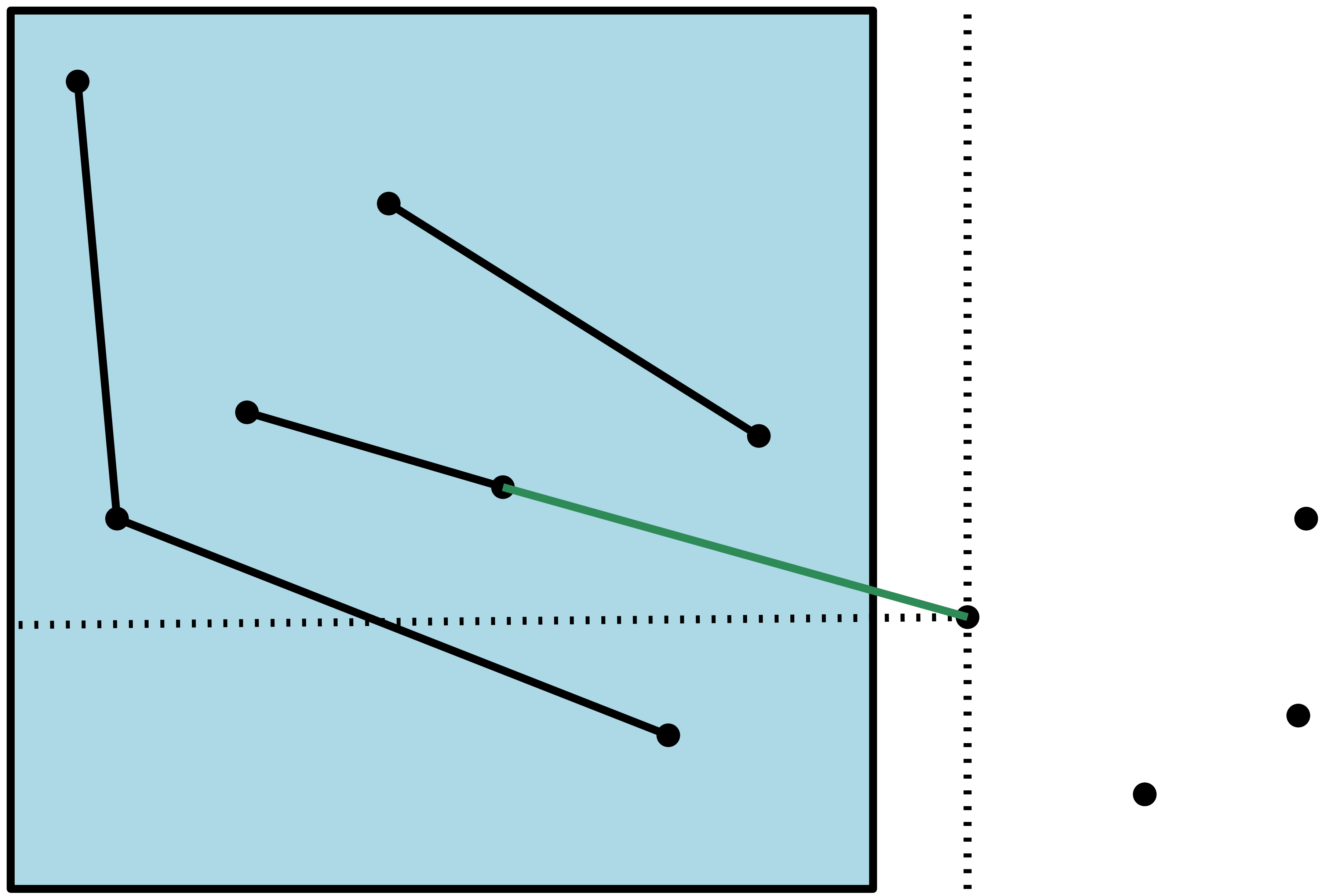
# Descending chains

- Supowit's algorithm can generate the minimum number of chains in one direction
- Takes $O(n \lg n)$ time
- Idea: add each point to the lowest compatible chain

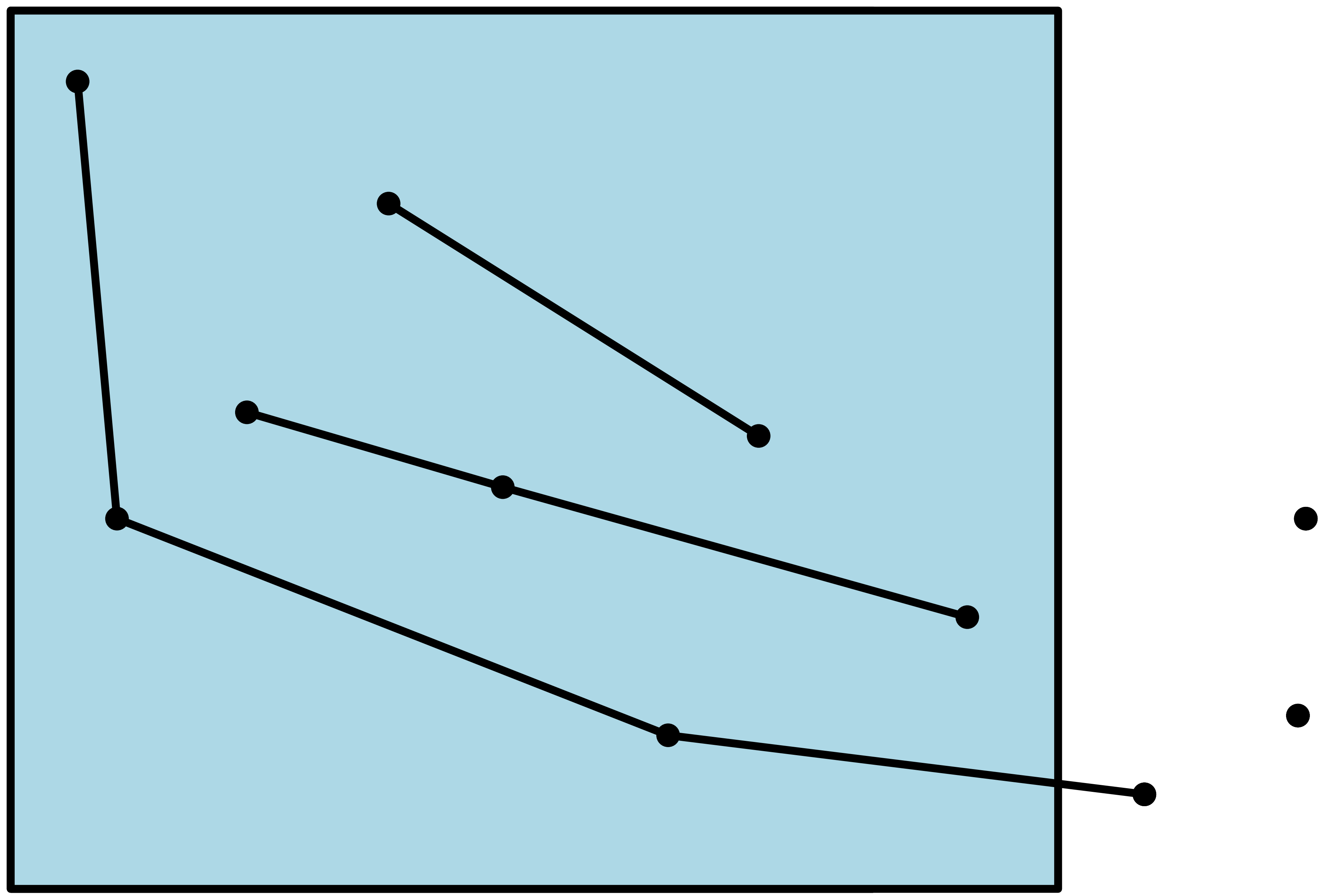
# Supowit's algorithm



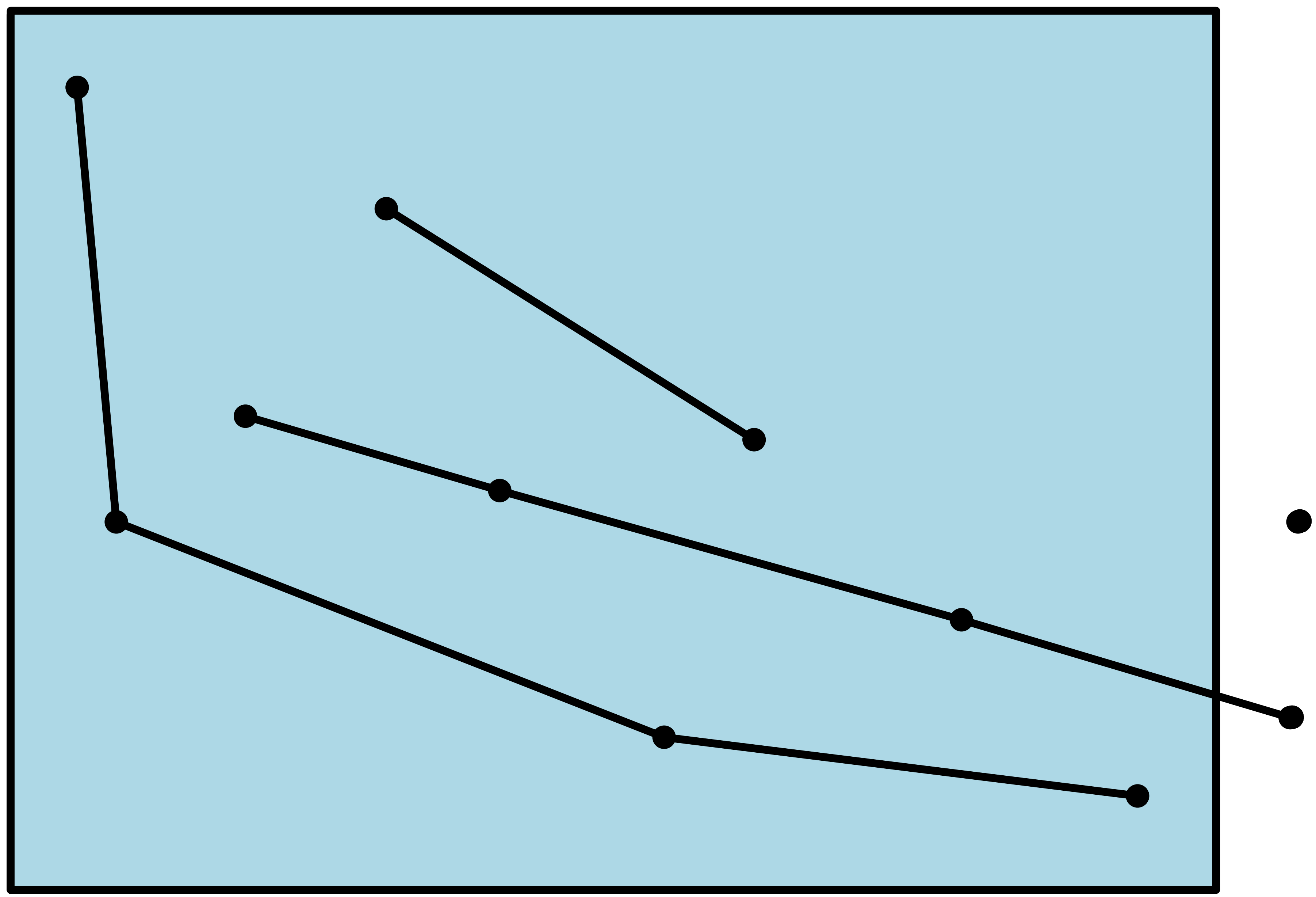
# Supowit's algorithm



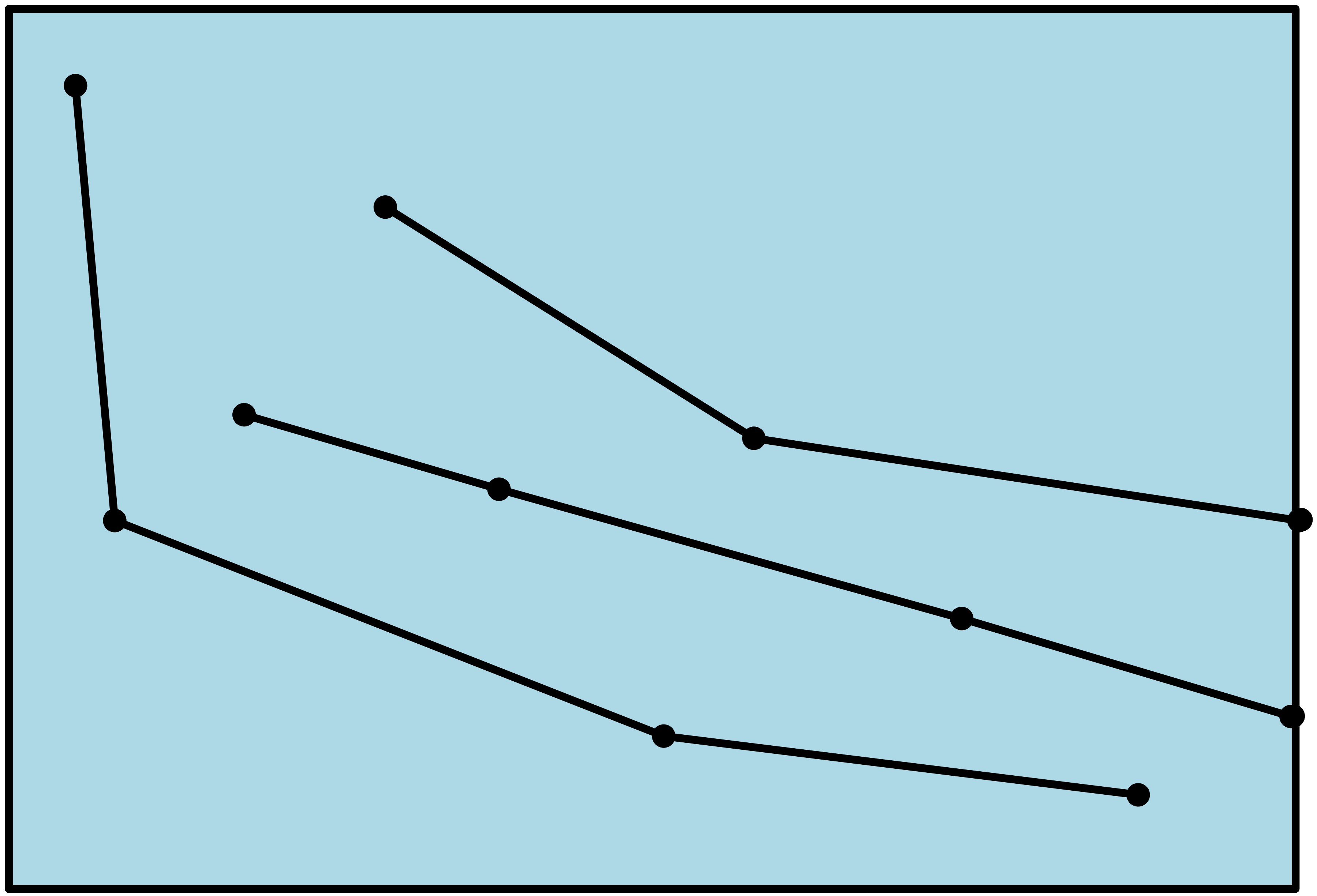
# Supowit's algorithm



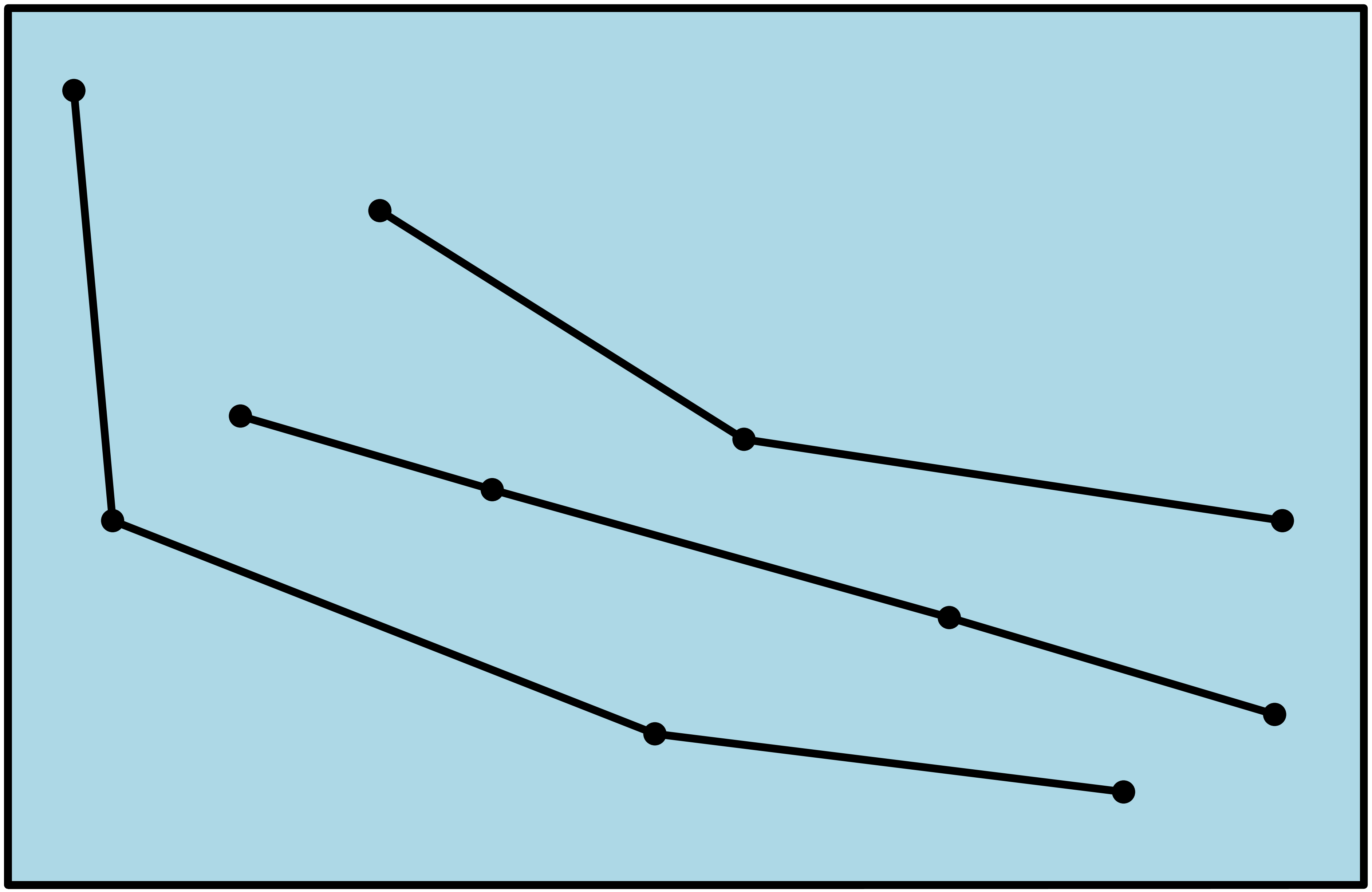
# Supowit's algorithm



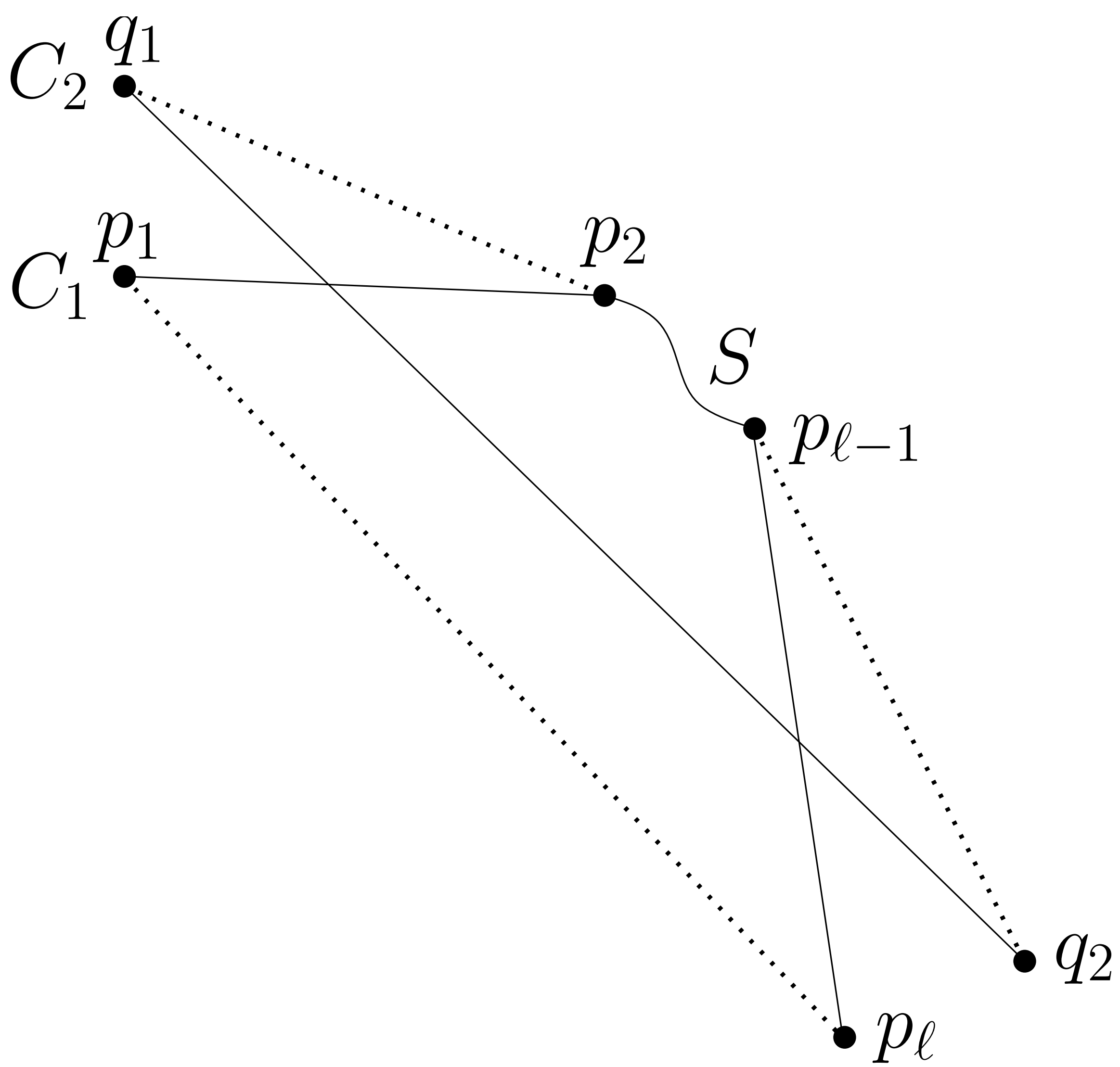
# Supowit's algorithm

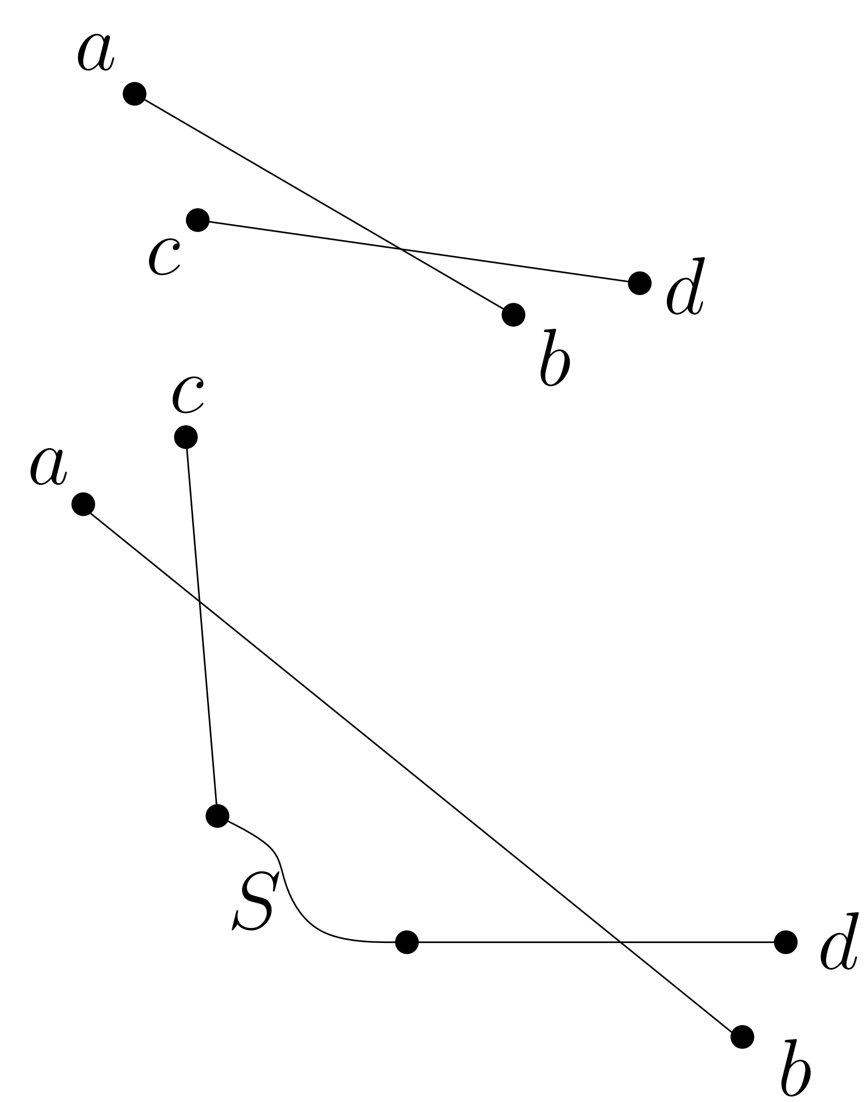


# Supowit's algorithm

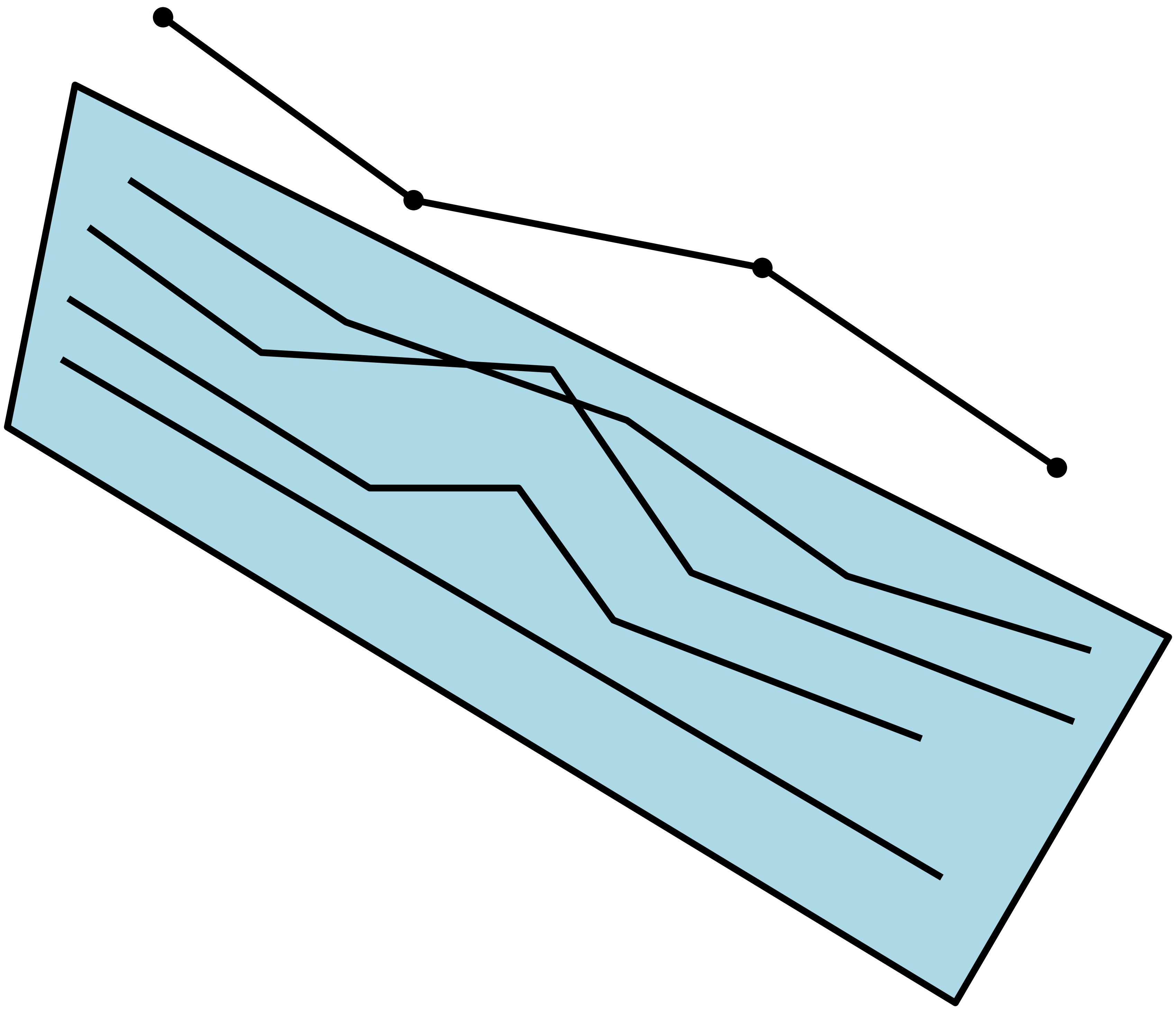


# Crossing lines

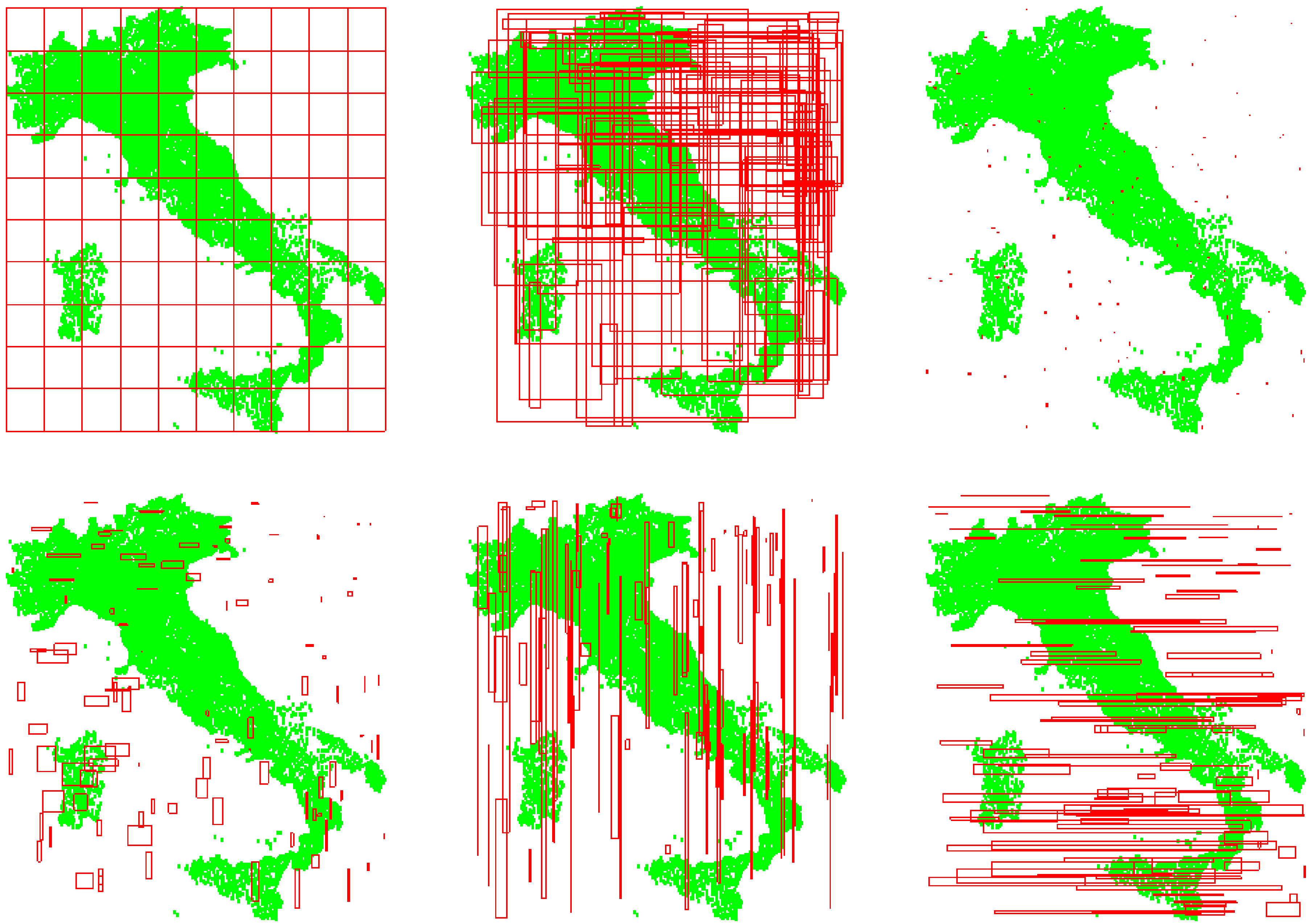


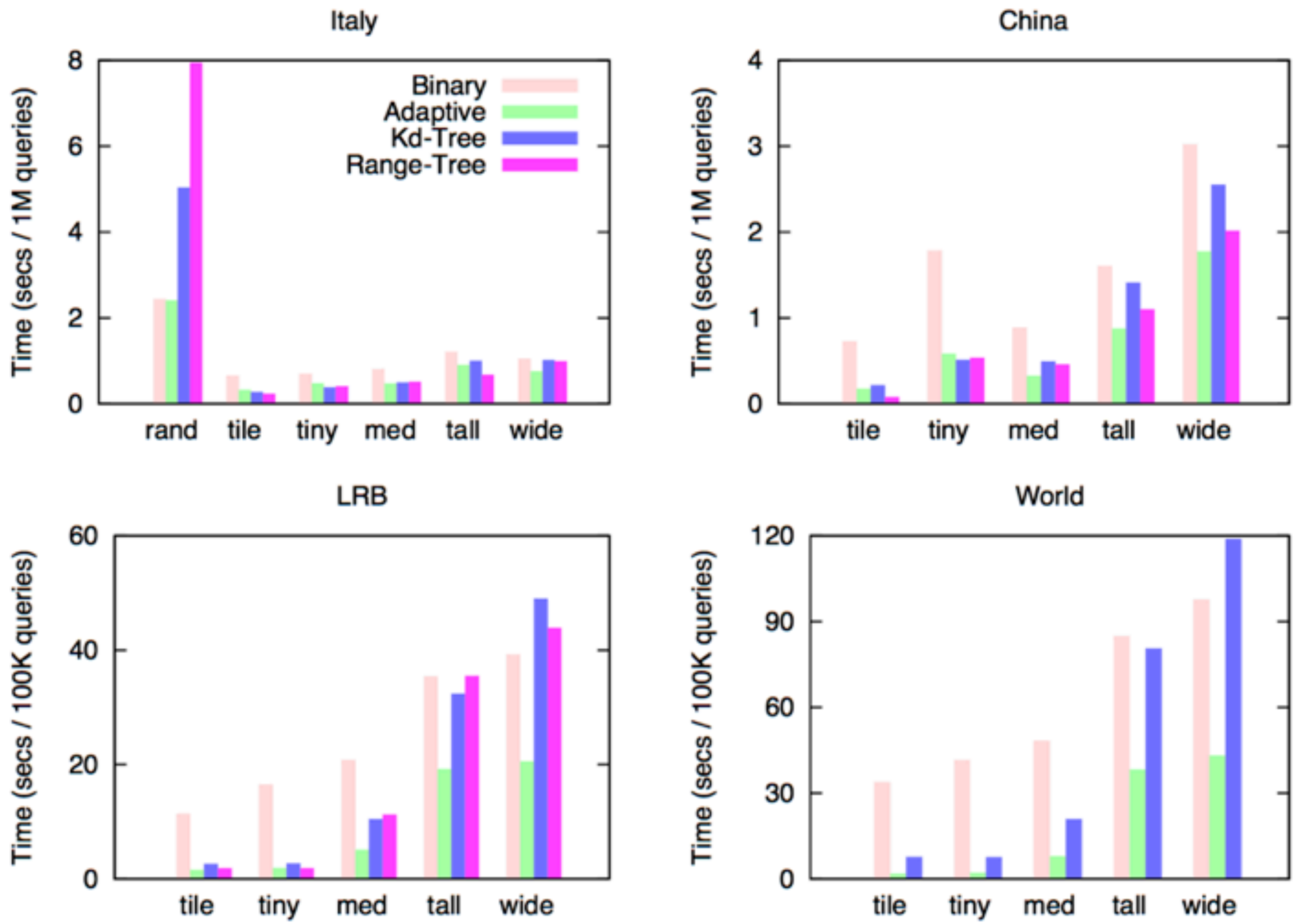
# Untangling idea



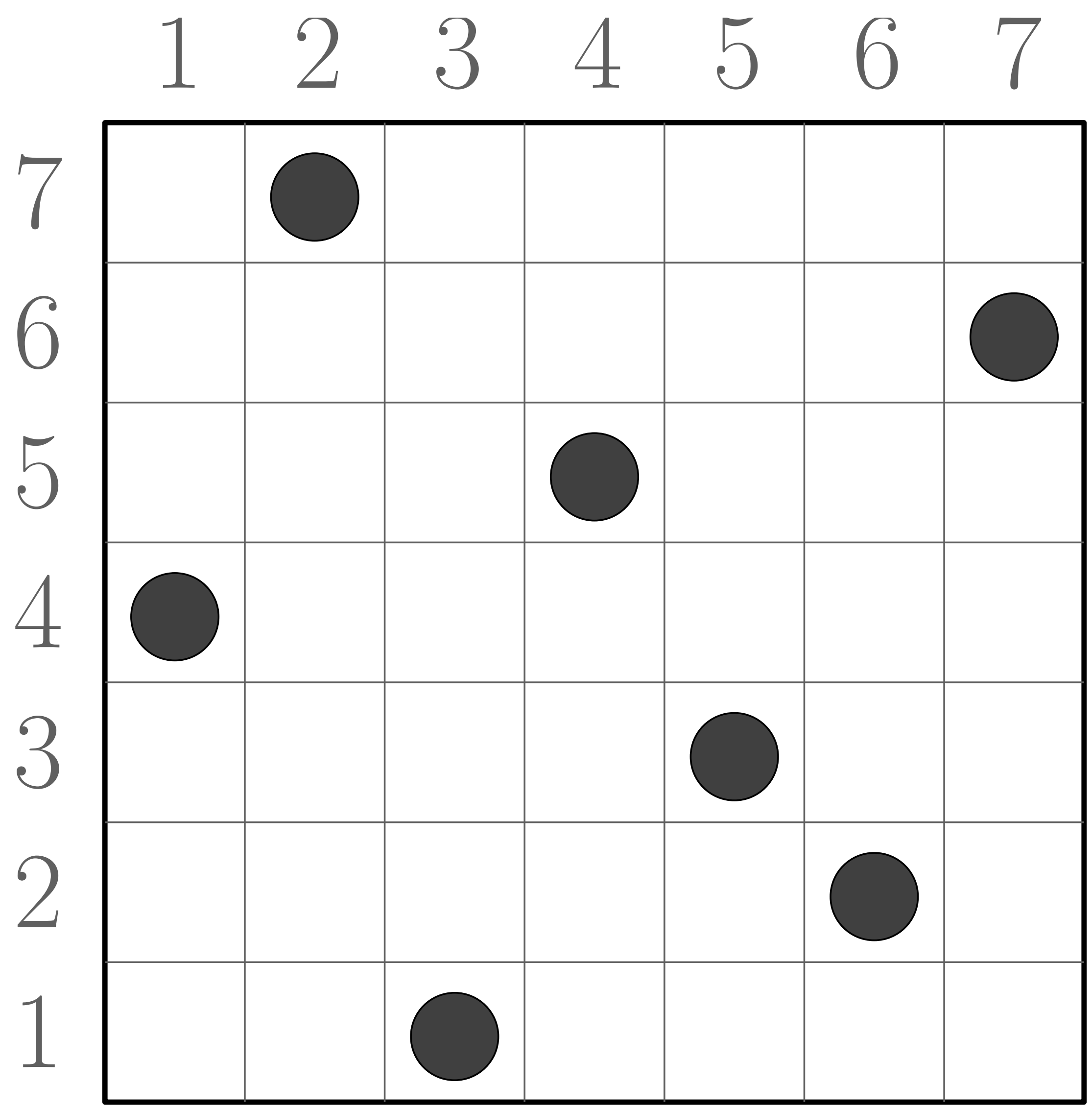
# Example



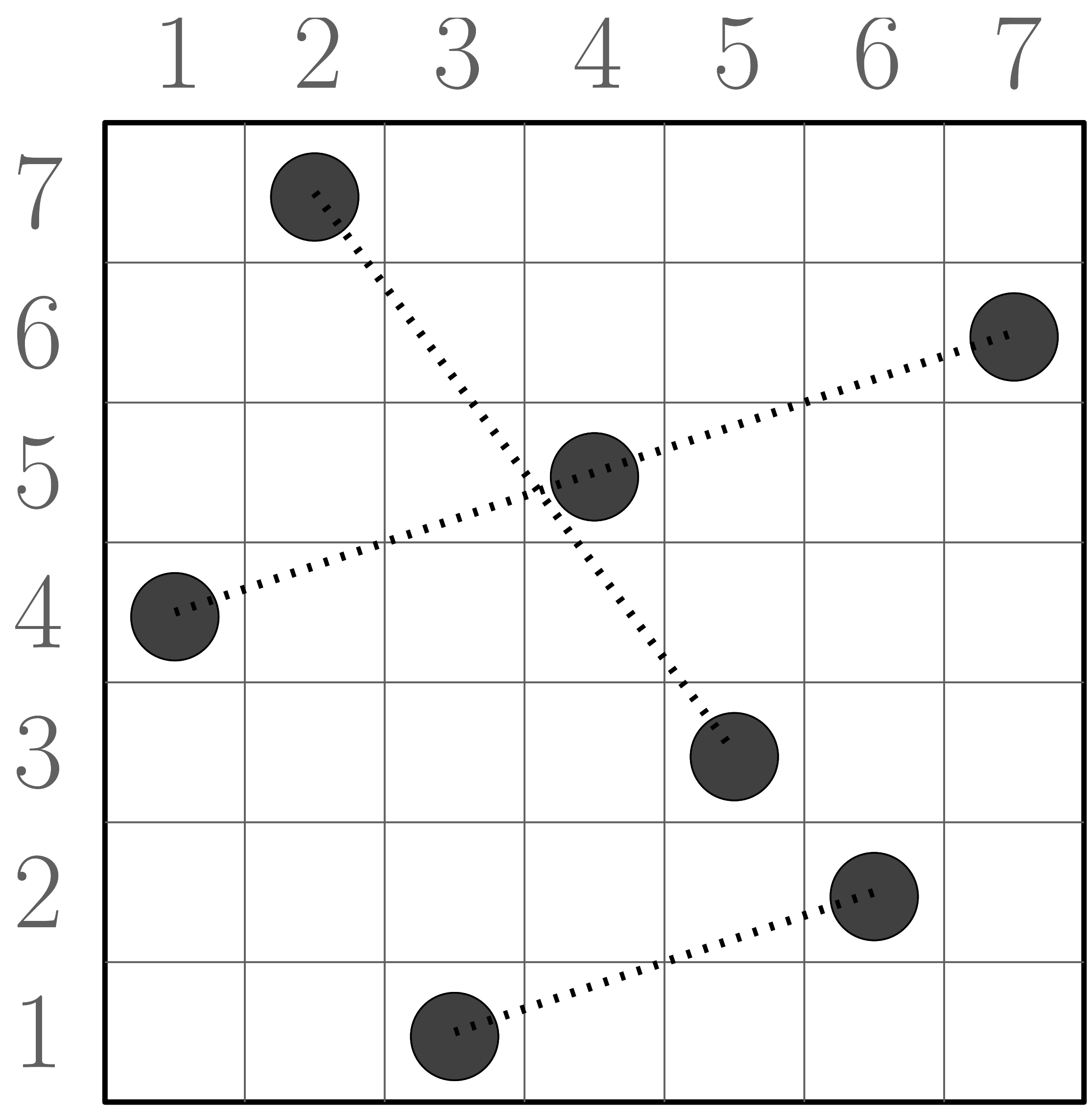
# Time


Italy
Binary
Adaptive
Kd-Tree
Range-Tree
Time (secs / 1M queries)
0
2
4
6
8
rand
tile
tiny
med
tall
wide
China
Time (secs / 1M queries)
0
1
2
3
4
tile
tiny
med
tall
wide
LRB
Time (secs / 100K queries)
0
20
40
60
tile
tiny
med
tall
wide
World
Time (secs / 100K queries)
0
30
60
90
120
tile
tiny
med
tall
wide

# Now to our topic...



# Now to our topic...



# Representing One Chain

- Consider a chain $[(x_1, y_1), (x_2, y_2) \ldots, (x_m, y_m)]$
- We can represent it with 2 bitmaps, each with m ones and length n

$$I = [x_1, x_2, \ldots, x_m]$$
$$J = [y_1, y_2, \ldots, y_m]$$

# Operations Supported

- Get any index of the pairs
- Obtain points in a given orthogonal range

# Getting pairs

$$I = [x_1, x_2, \ldots, x_m]$$

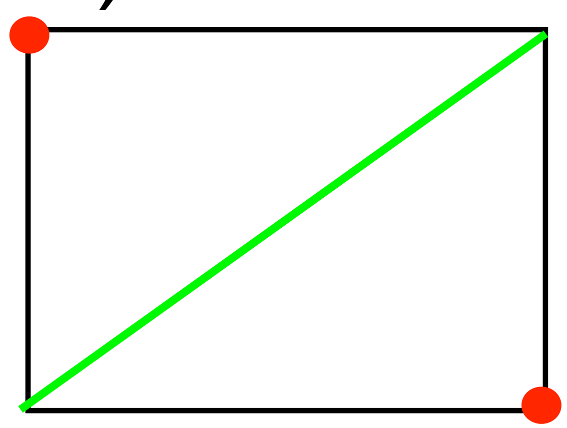
$$J = [y_1, y_2, \ldots, y_m]$$

$$\mathrm{getj}_{\mathcal{C}}(i) = \begin{cases} \perp & \text{if } I[i] = 0 \\ \mathrm{select}_{B_J}(\mathrm{rank}_{B_I}(i)) & \text{otherwise} \end{cases}$$

00100100001100000100101 [3,6,11,12,18,21,23]

01000100010100010100010 [2,6,10,12,16,18,22]

# Range




Use a function similar to getj

0010010000110000000100101 [3,6,11,12,18,21,23]

0100010001010001010001 0 [2,6,10,12,16,18,22]

# Results for all chains

| | |
|---|---|
| $2n \lg k + O\left(k \lg n + \dfrac{kn}{\lg^c n}\right)$ | $O(c)$<br>Patrascu |
| $2n \lg k + O(k \lg n + n)$ | $O\left(\lg \dfrac{n}{m} + \dfrac{\lg^4 m}{\lg n}\right)$<br>Okanohara & Sadakane |

# $\pi$ and $\pi^{-1}$

- We need to double the space to know which chain contains the query coordinate

$$4n \lg k + O(k \lg n + n)$$

$$\leq 2n \lg n + O(k \lg n + n)$$

# Observation

- For each position, one and only one chain has a one
- We can represent this with a sequence over an alphabet k
- This converges to something we knew ...

# Observation

- For each position, one and only one chain has a one
- We can represent this with a sequence over an alphabet k
- This converges to something we knew ...

  Barbay & Navarro, STACS 2009

# Did we gain anything?

# Did we gain anything?

YES!

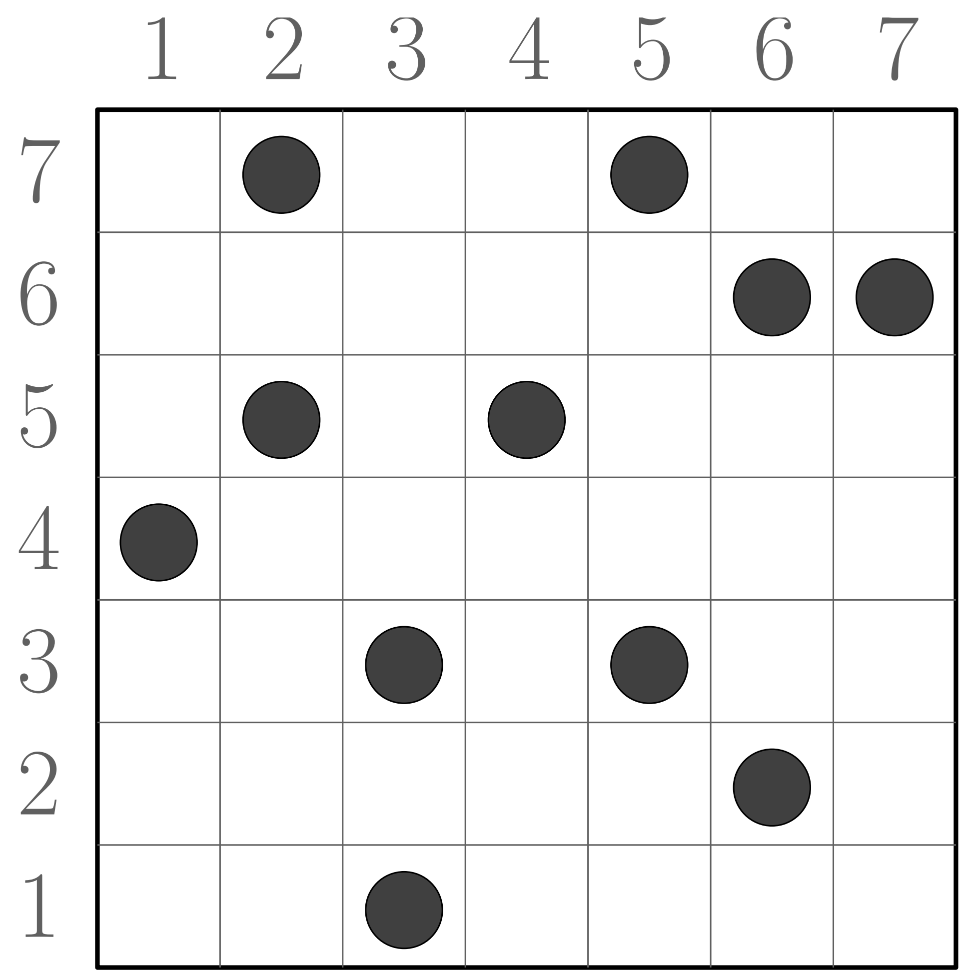
# Did we gain anything?

YES!

Range searching can be performed in

$$O(\lg^{1+\epsilon} k + k' \lg \lg k)$$

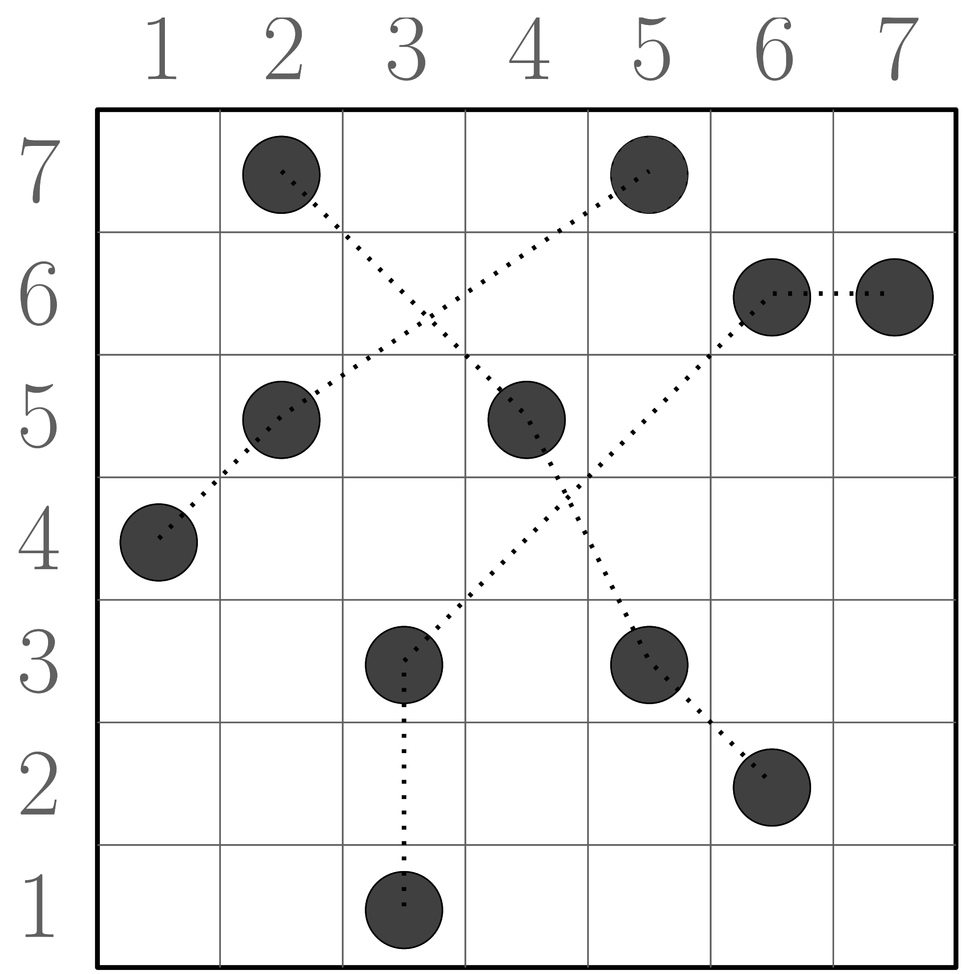
# Binary Relations




n x n

t elements

# Binary Relations



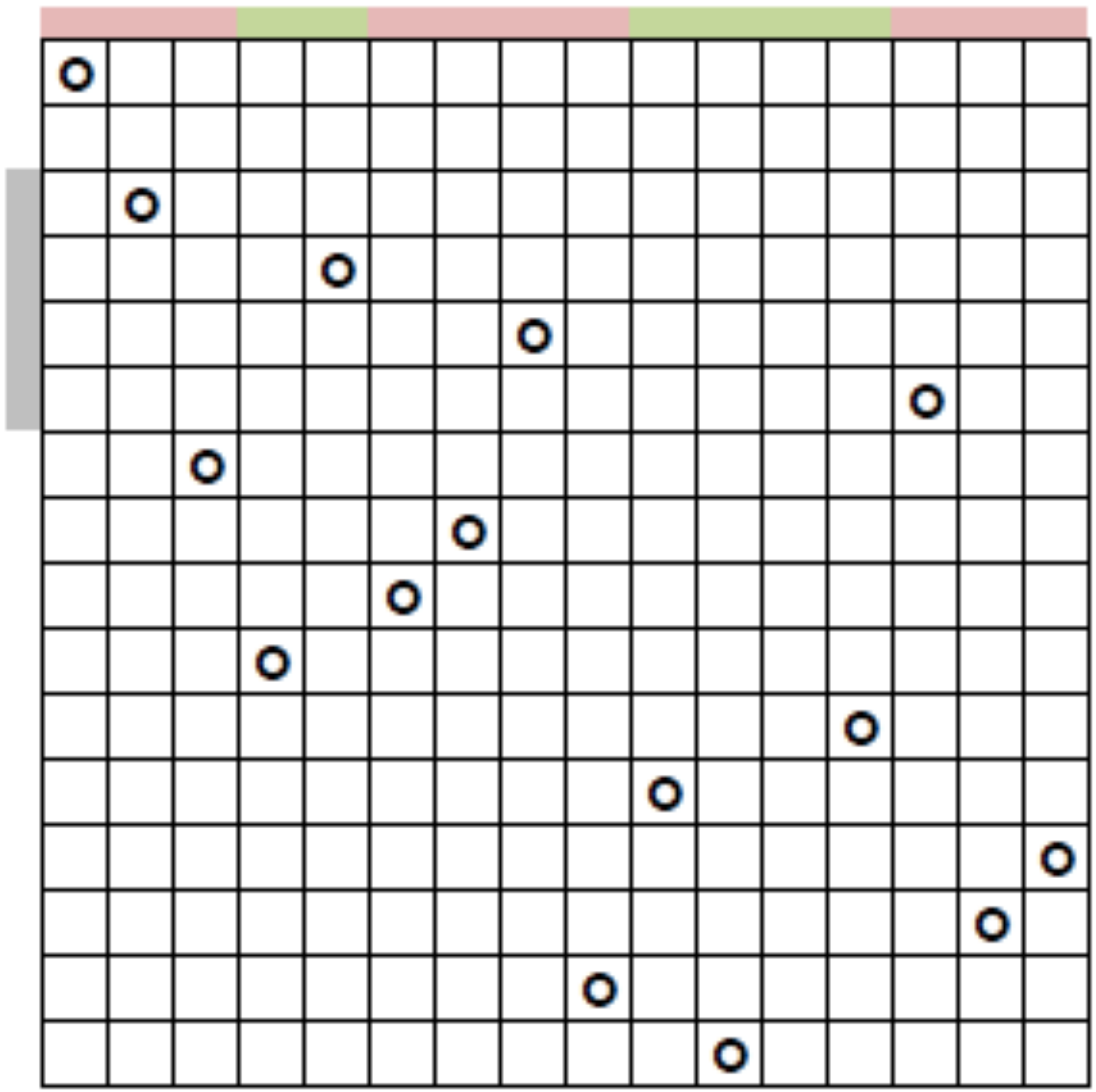

n x n

t elements

# Option 1

**Data:** $\mathcal{R}$
**Result:** $\Pi, B_1, B_2$
$\bar{\mathcal{R}} \leftarrow$ empty binary relation of size $t \times n_2$
$row \leftarrow 1$
$B_1 \leftarrow 0^t$
**for** $i \leftarrow 1$ *to* $\sigma$ **do**
  $B_1[row] \leftarrow 1$
  **for** $j$ *such that* $(i, j) \in \mathcal{R}$ **do**
    add $(row, j)$ to $\bar{\mathcal{R}}$
    $row \leftarrow row + 1$
$col \leftarrow 1$
$B_2 \leftarrow 0^t$
**for** $j \leftarrow 1$ *to* $n$ **do**
  $B_2[col] \leftarrow 1$
  **for** $i$ *such that* $(i, j) \in \bar{\mathcal{R}}$ **do**
    $\Pi[i] = col$
    $col \leftarrow col + 1$
**return** $(\Pi, B_1, B_2)$

# Option 1



# Option 2

- Use chains directly
- We can do a bit better than before
- Dealing with bitmaps with multiple ones in a given position - trick: set bit A[k]+k

# Results

space: $2t \lg \frac{nk}{t} + 2t \lg k + O\left(\frac{kn}{\lg^c n} + k \lg t\right)$

access: $O(r)$

range: $O(r(\lg k + k'))$

where $r = max(c \lg k, c \lg \lg n)$

# Applications

- Inverted lists
  - We can add RMQs
- Text indexes

# The End