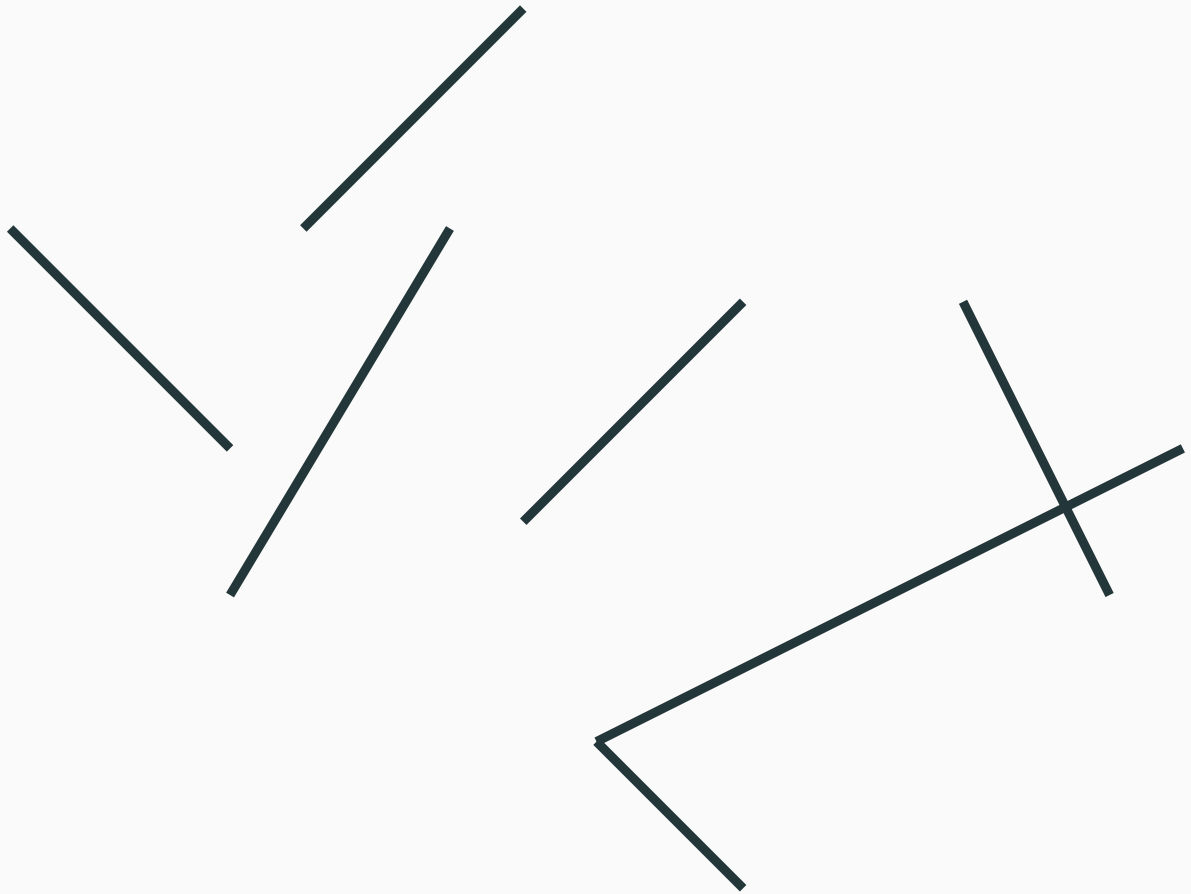
Covering a set of segments with a few squares

Joachim Gudmundsson, Mees van de Kerkhof, André van Renssen, Frank Staals, Lionov Wiratma, Sampson Wong*

Sydney University and Utrecht University

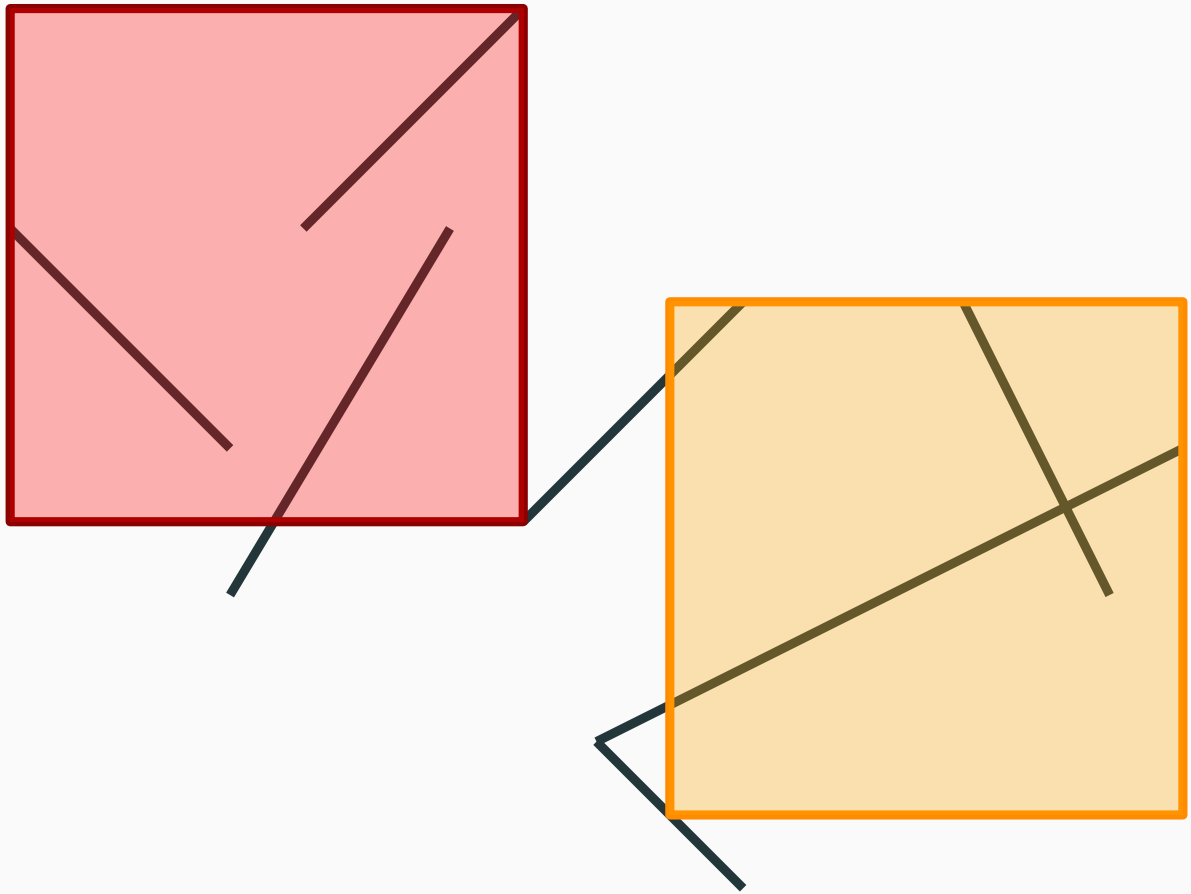
Our Problem

Our Problem



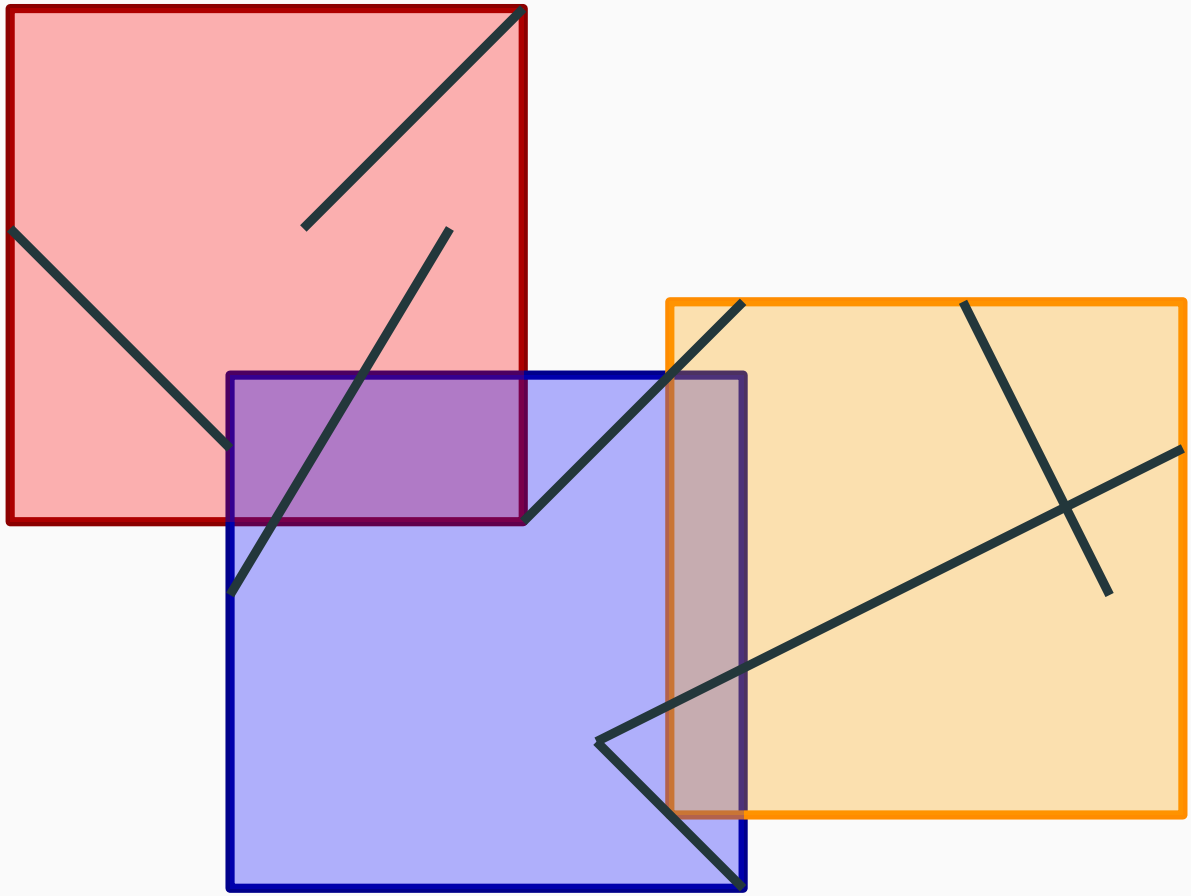
Segments in the plane

Our Problem



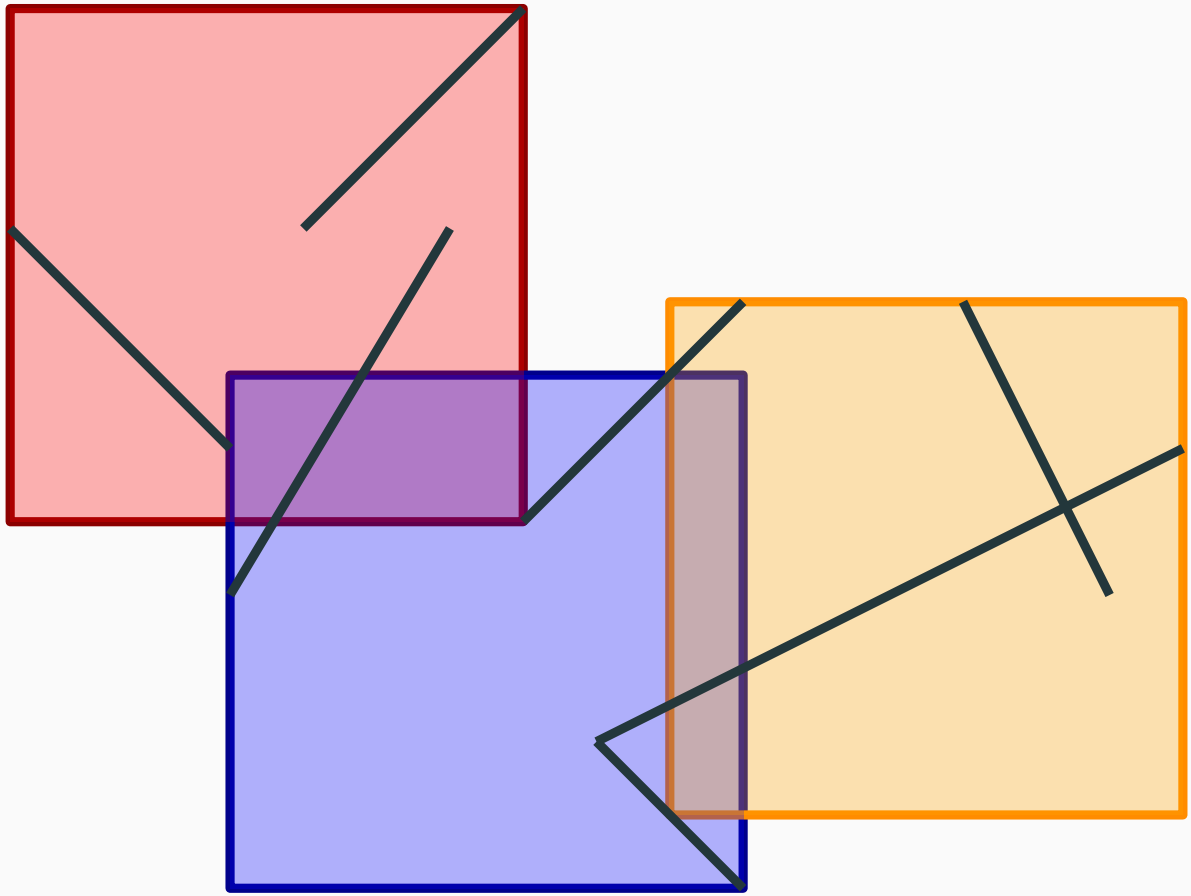
Cover with k unit-sized axis-parallel squares?

Our Problem



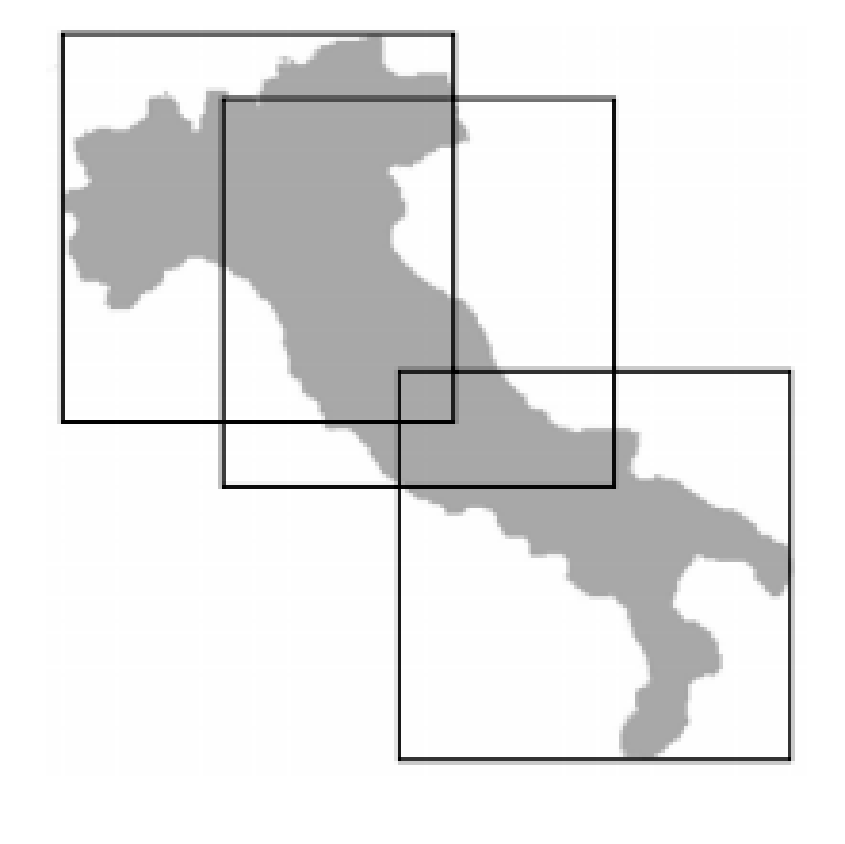
Cover with k unit-sized axis-parallel squares?

Our Problem



Are segments k-coverable?

Motivation



Map printing

[1] Hoffmann, EuroCG, 2001

Previous and Our Results

Results

	Previous	This paper	Running time
$k = 2$	[1,2]		$O(n)$
$k = 3$	[1]	✓	$O(n)$
$k = 4$		✓	$O(n \log n)$

[1] Hoffmann, EuroCG, 2001

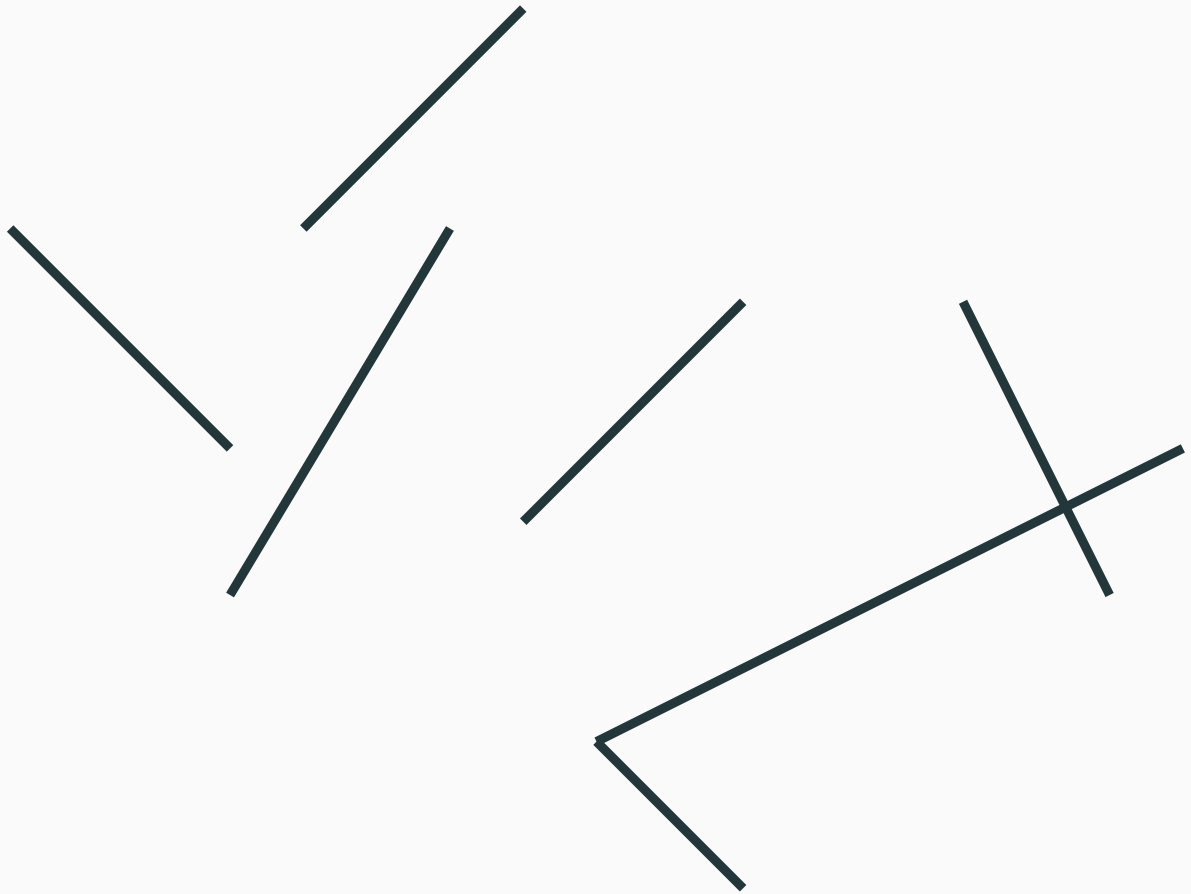
[2] Sadhu, Roy, Nandy, and Roy, Theor. Comp. Sci. 769:63-74, 2019

This talk

- $k = 2$
- $k = 3$
- $k = 4$, main ideas

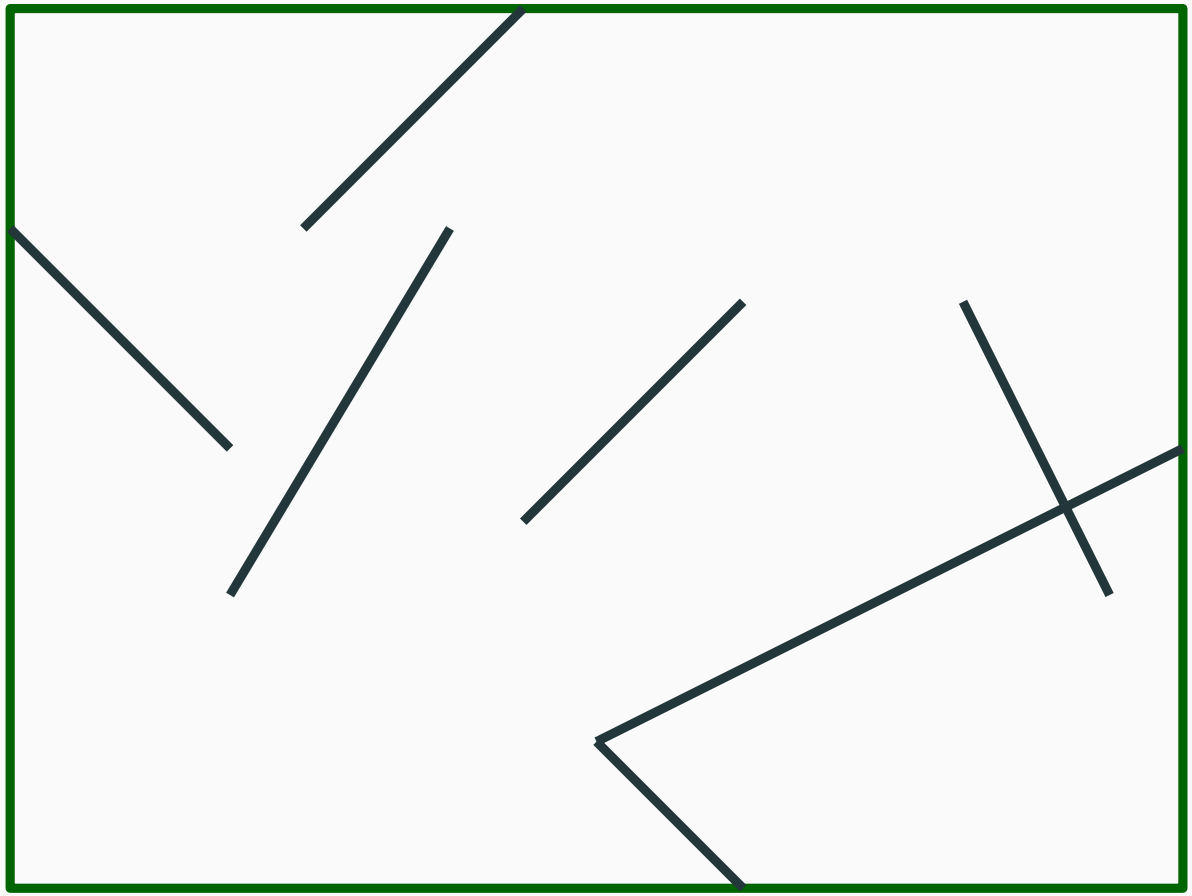
$k = 2$

$k = 2$



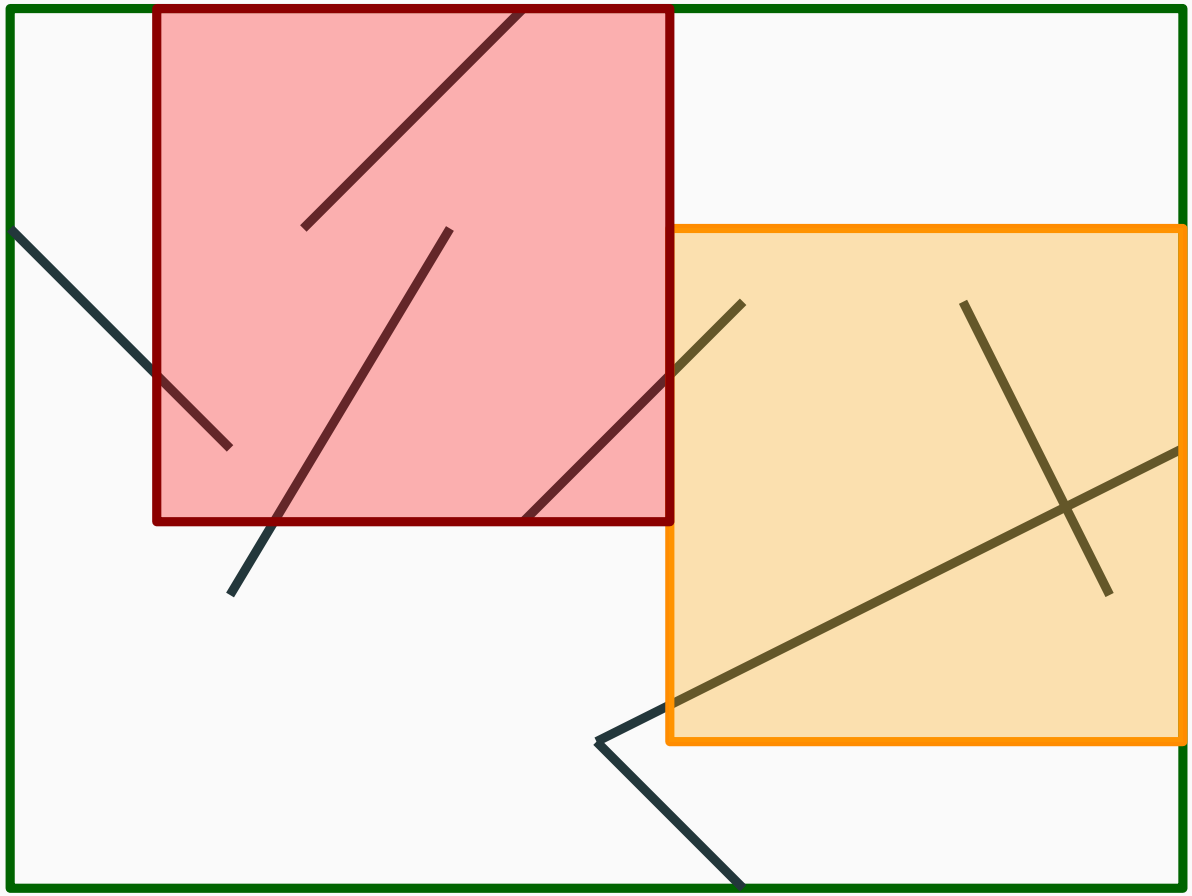
Draw bounding box

$k = 2$



Observation: a 2-covering must touch every side of the bounding box

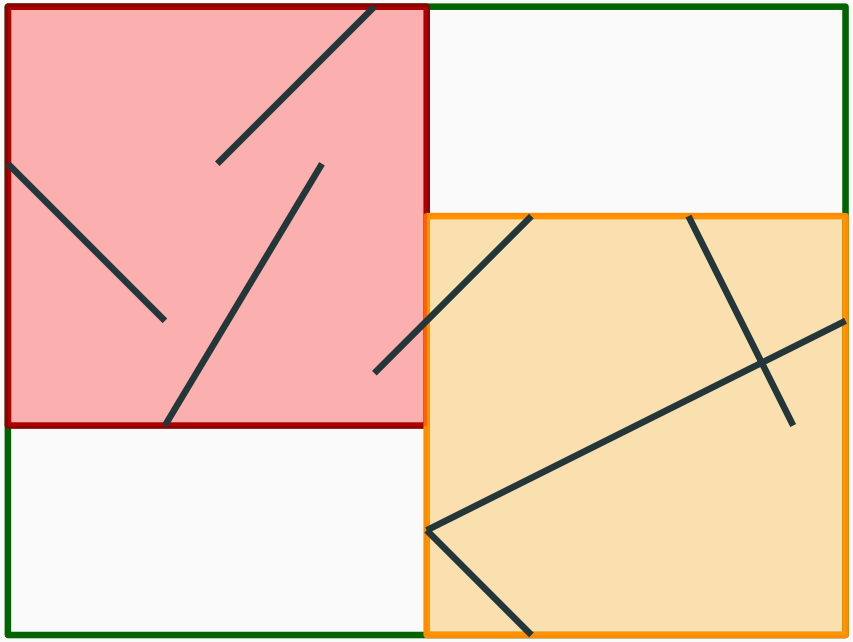
$k = 2$



Reason: if left side not touched, leftmost vertex not covered

$k = 2$

Lemma If there is a 2-covering, there is an opposite-corner covering.

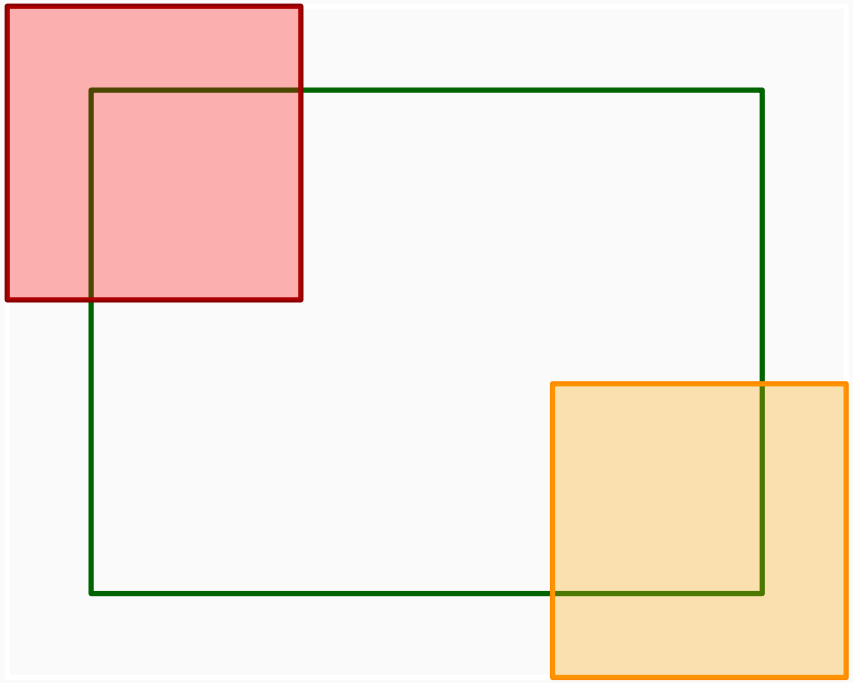


Opposite-corner covering

Proof: Fat Case

Suppose height and width > 1.

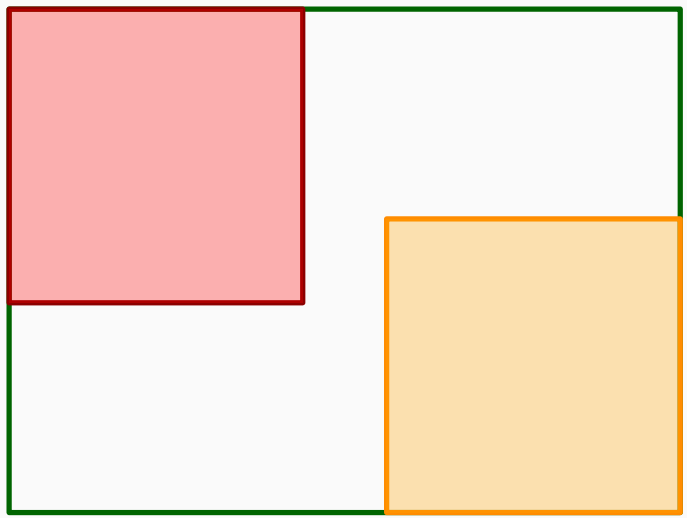
Squares touch two adjacent sides.



Proof: Fat Case

Suppose height and width > 1.

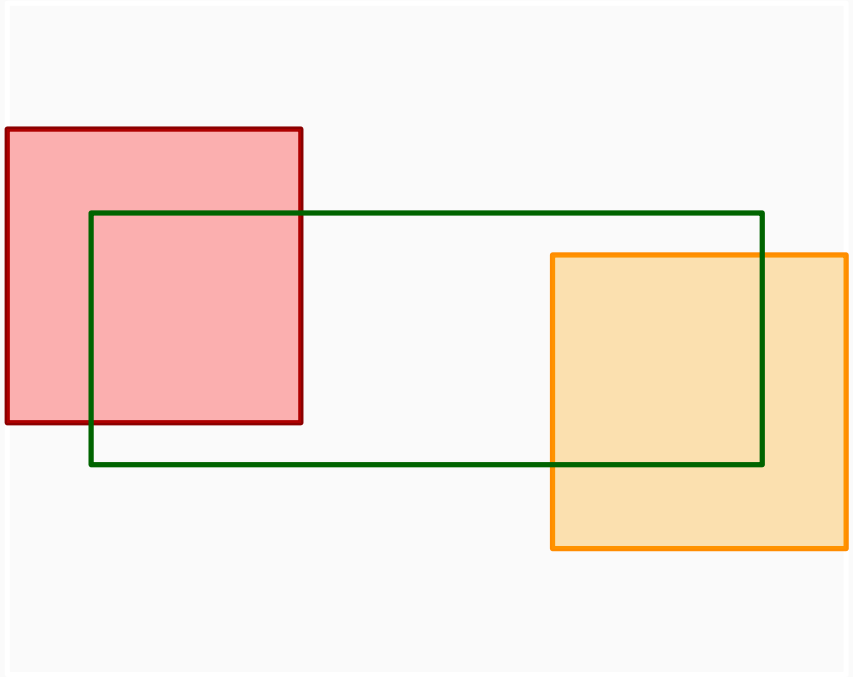
Squares touch two adjacent sides.



Proof: Thin Case

Suppose height (or width) ≤ 1.

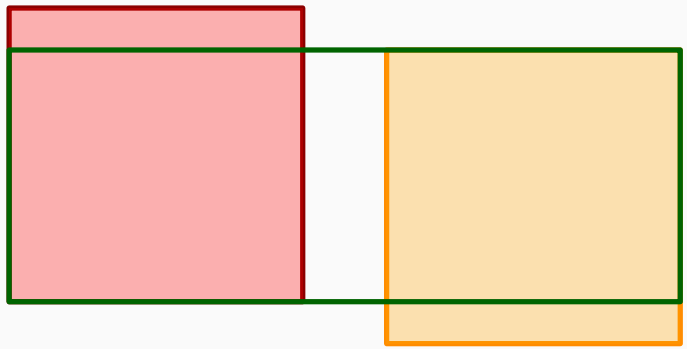
Squares touching left, and right.



Proof: Thin Case

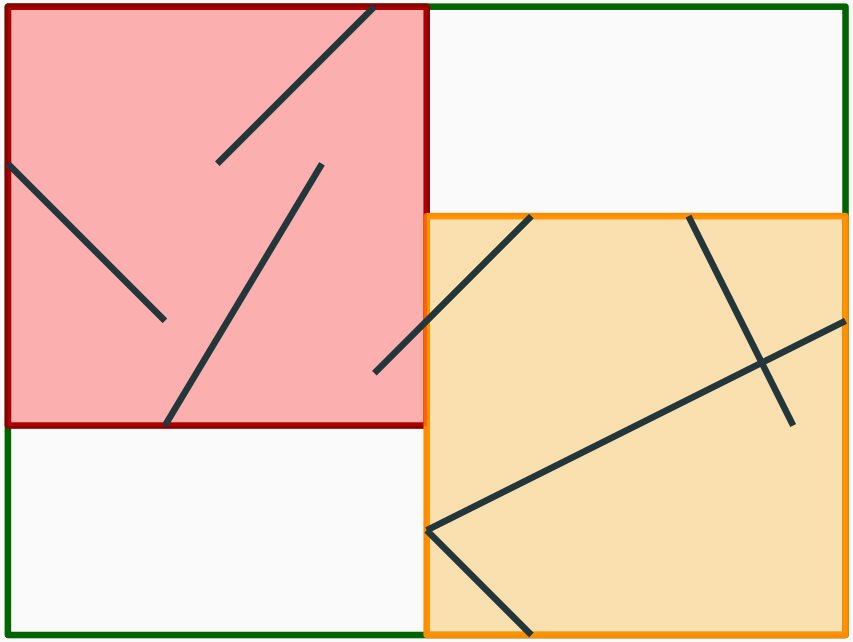
Suppose height (or width) ≤ 1.

Squares touching left, and right.



$k = 2$

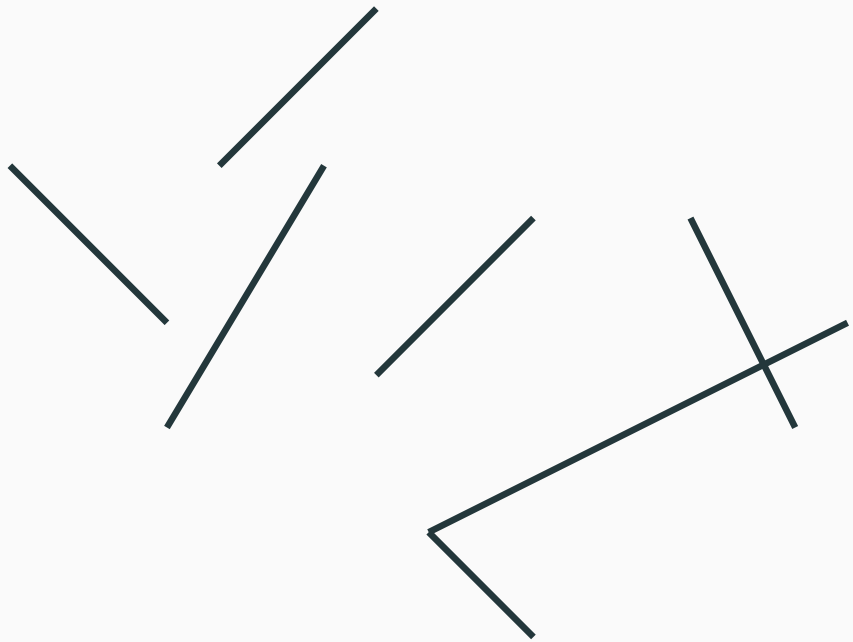
Lemma If there is a 2-covering, there is an opposite-corner covering.



Opposite-corner covering

$k = 2$

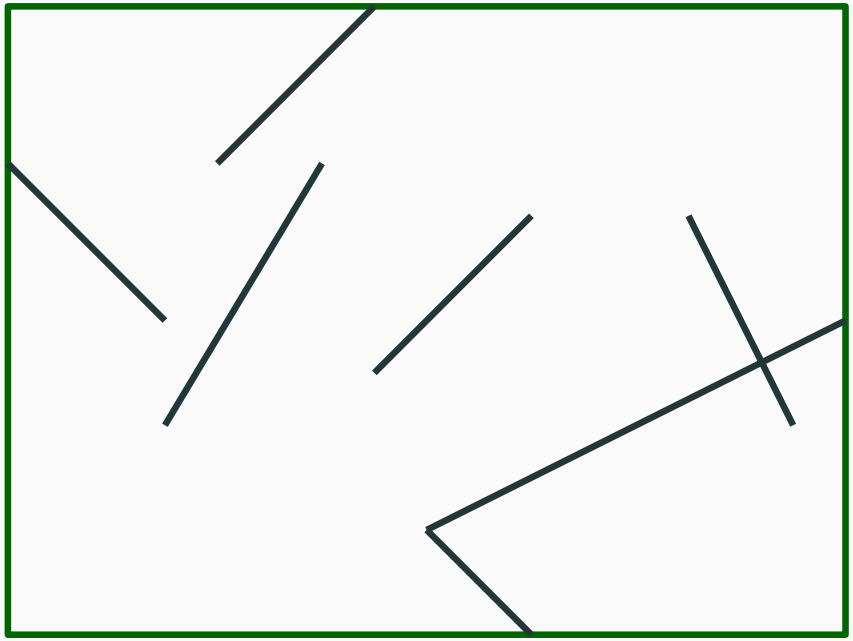
Algorithm



$k = 2$

Algorithm

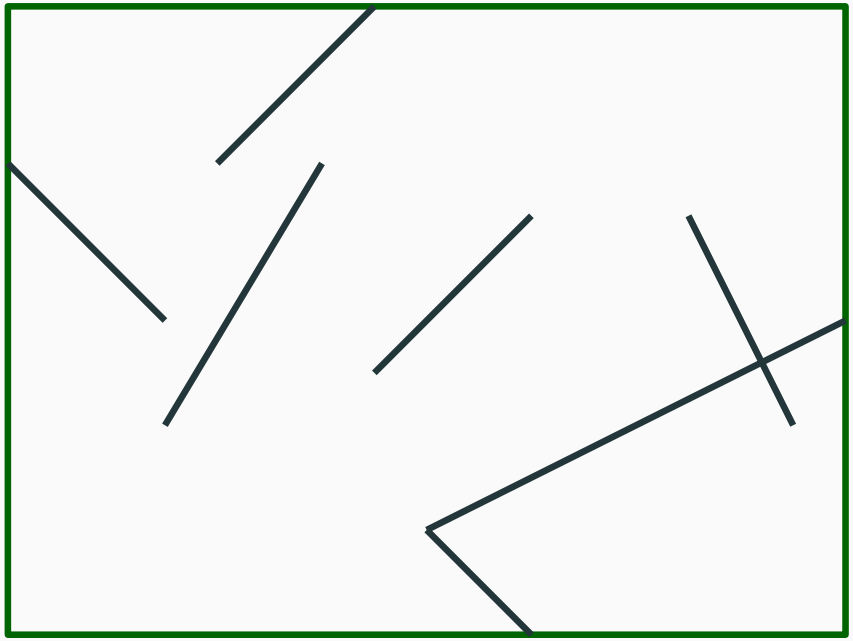
- Compute bounding box



$k = 2$

Algorithm

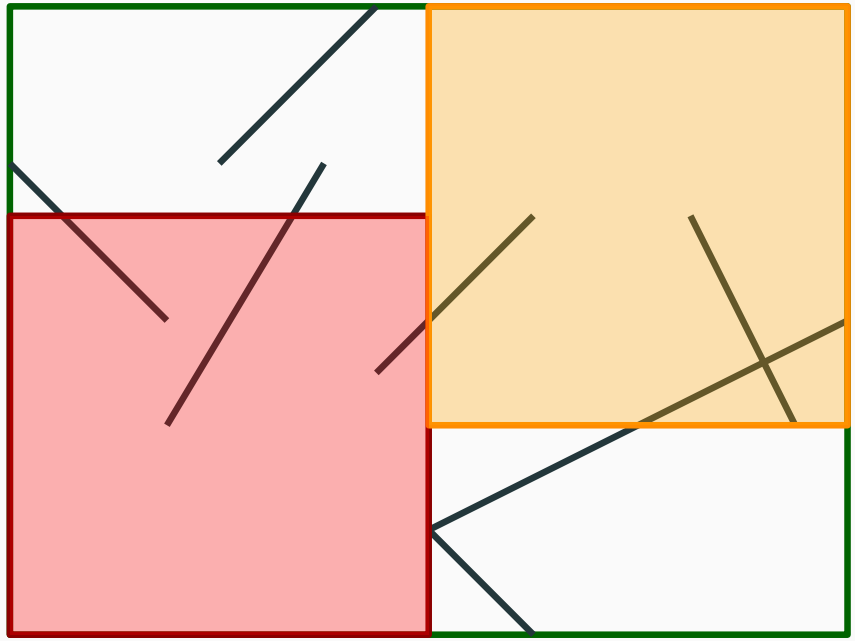
- Compute bounding box
- For each opposite corner do:



$k = 2$

Algorithm

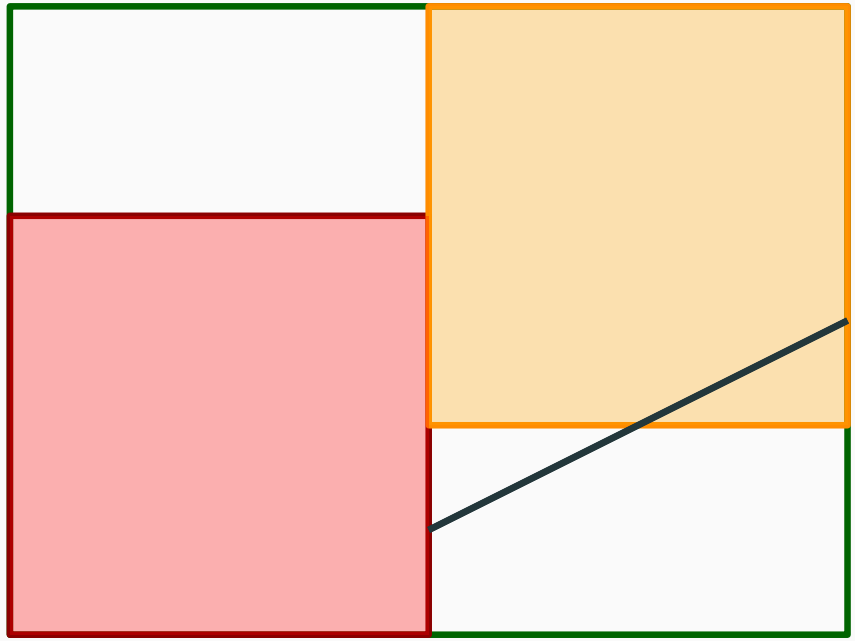
- Compute bounding box
- For each opposite corner do:
  - Place squares in corners



$k = 2$

Algorithm

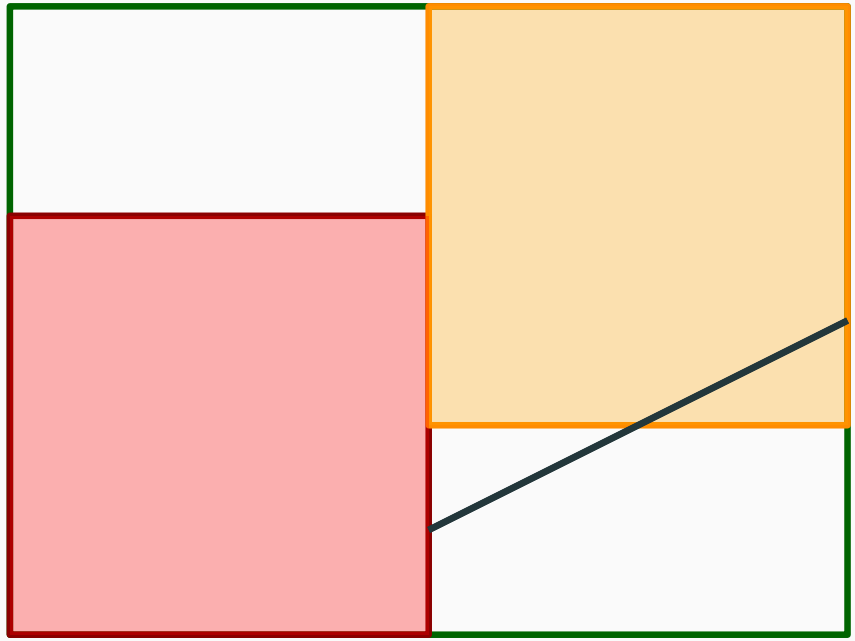
- Compute bounding box
- For each opposite corner do:
  - Place squares in corners
  - For each edge do:



$k = 2$

Algorithm

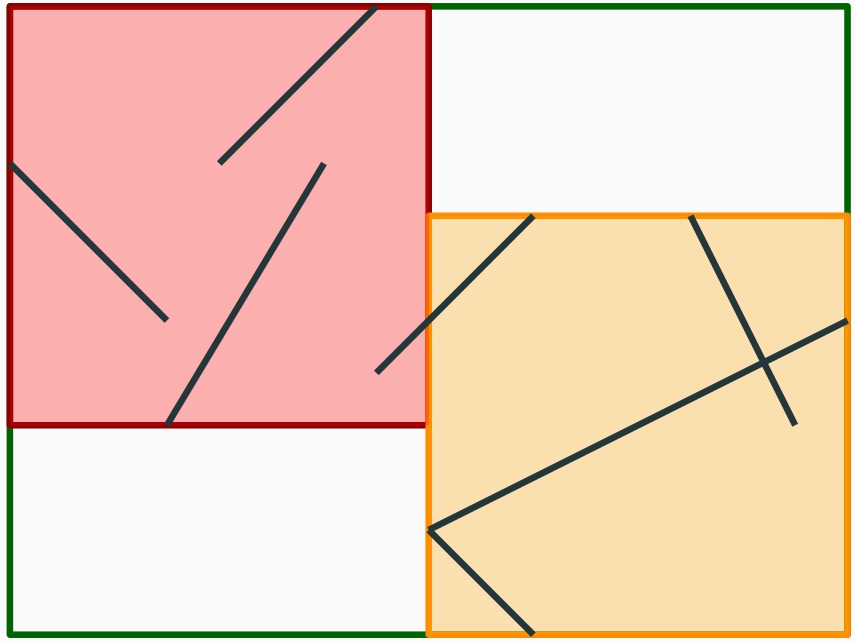
- Compute bounding box
- For each opposite corner do:
  - Place squares in corners
  - For each edge do:
    - Check if edge covered



$k = 2$

Algorithm

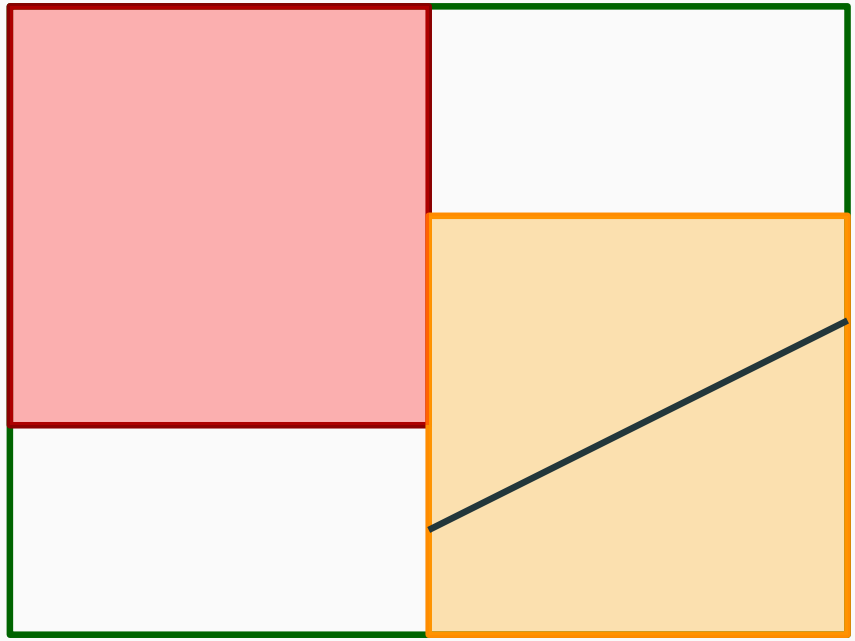
- Compute bounding box
- For each opposite corner do:
  - Place squares in corners



$k = 2$

Algorithm

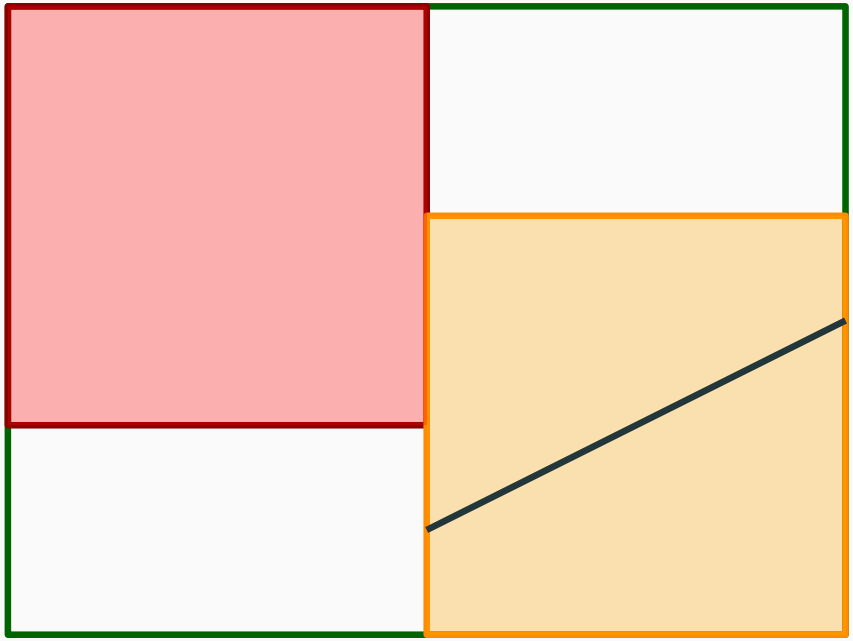
- Compute bounding box
- For each opposite corner do:
  - Place squares in corners
  - For each edge do:



$k = 2$

Algorithm

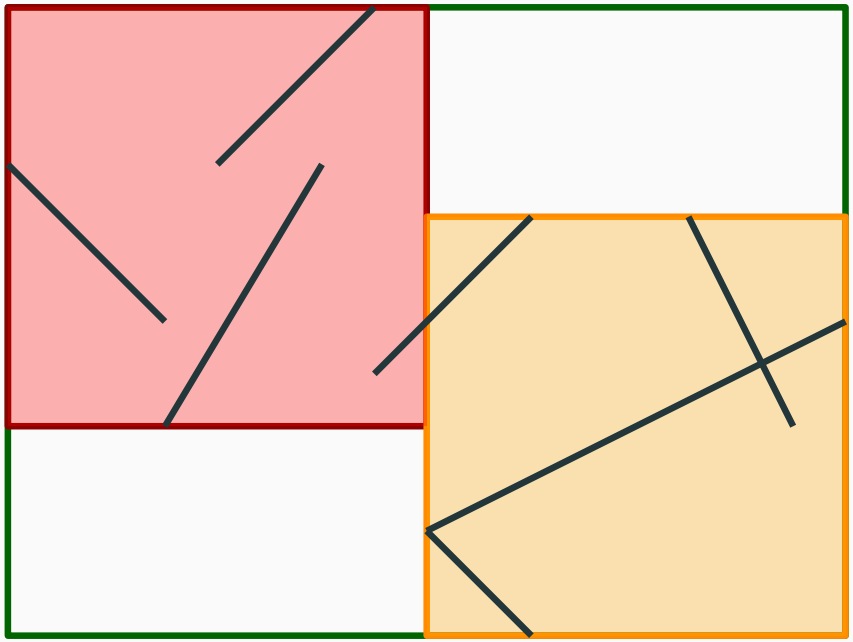
- Compute bounding box
- For each opposite corner do:
  - Place squares in corners
  - For each edge do:
    - Check if edge covered



$k = 2$

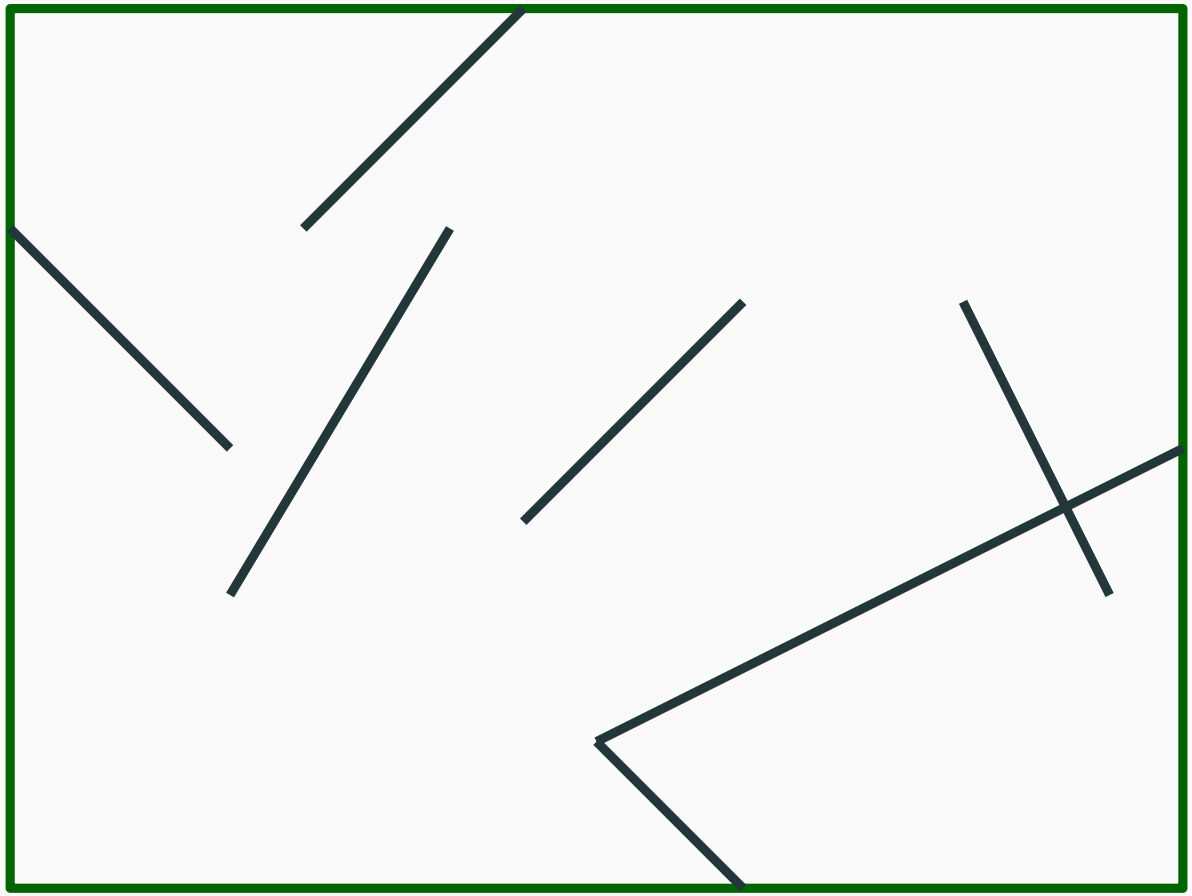
Algorithm

- Compute bounding box
- For each opposite corner do:
  - Place squares in corners
  - For each edge do:
    - Check if edge covered
- Overall takes $O(n)$ time



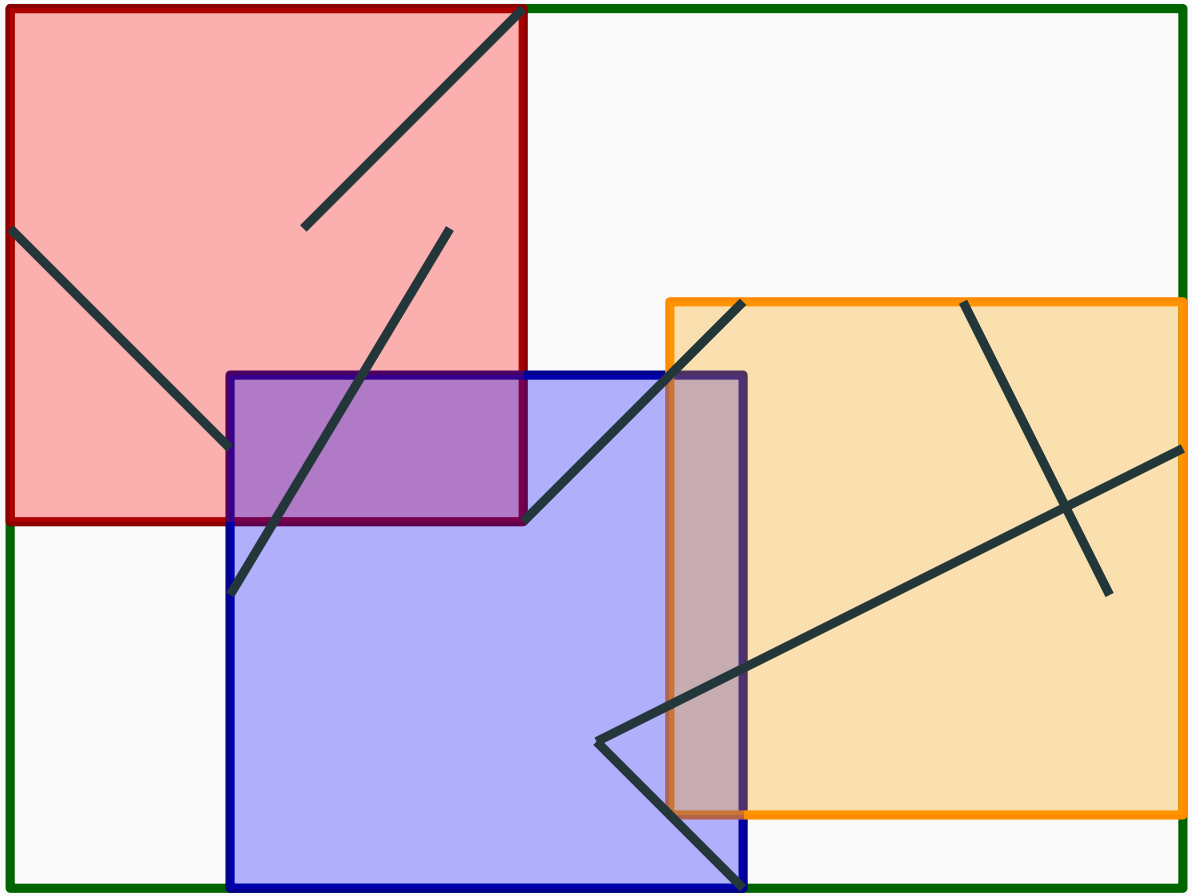
$k = 3$

$k = 3$



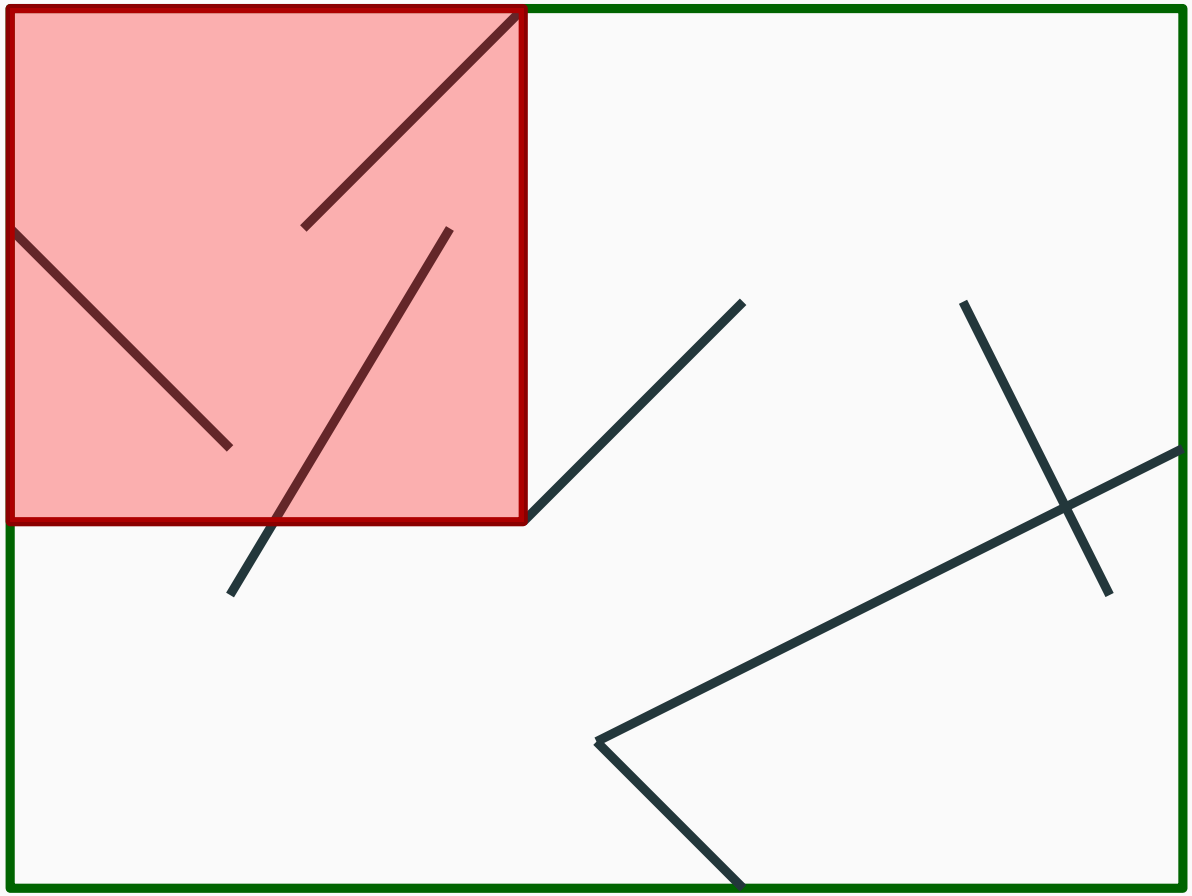
Can we make a similar observation?

$k = 3$



Observation: a 3-covering must touch every side of the bounding box

$k = 3$

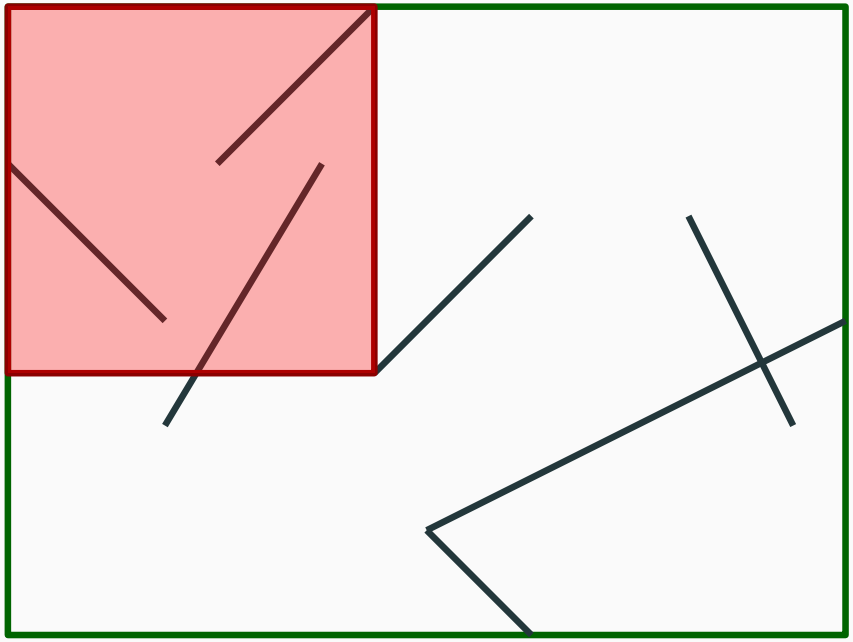


One square touches two sides

$k = 3$

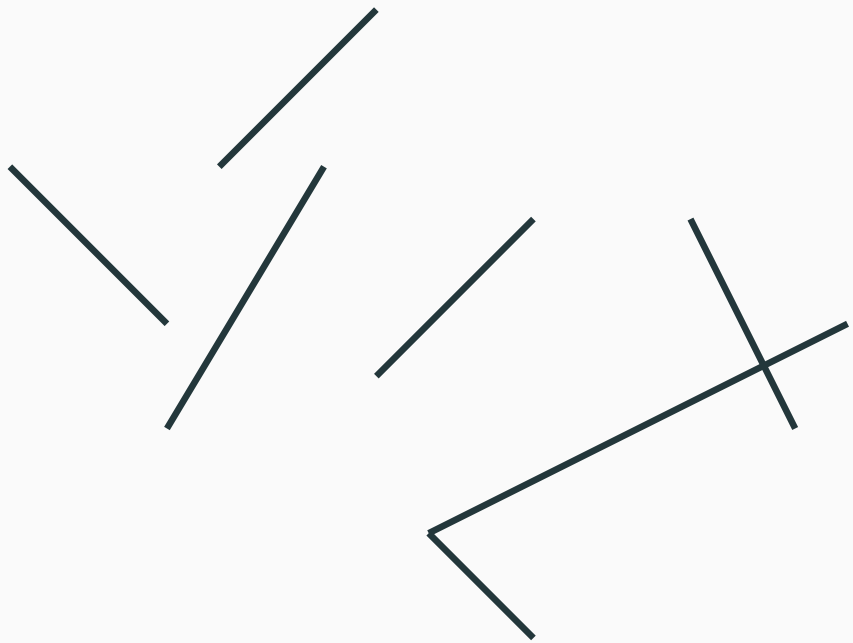
Lemma

If there is a 3-covering, there is a covering with a square in a corner of the bounding box.



A square in a corner

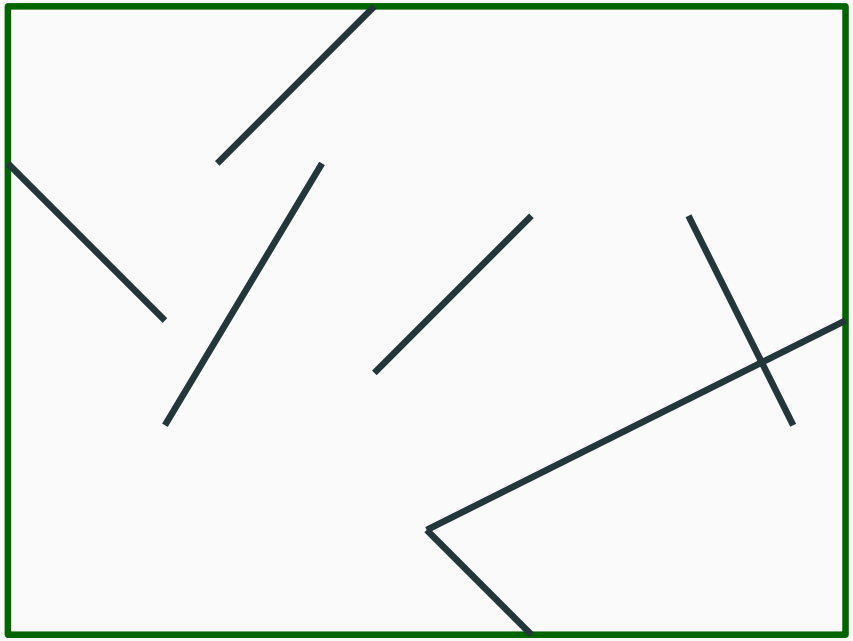
Algorithm



$k = 3$

Algorithm

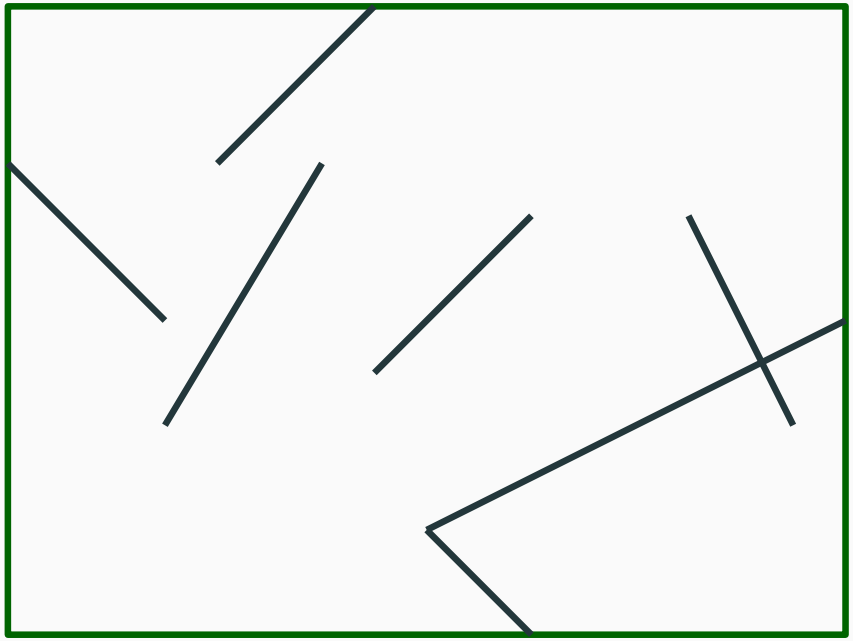
- Compute bounding box



$k = 3$

Algorithm

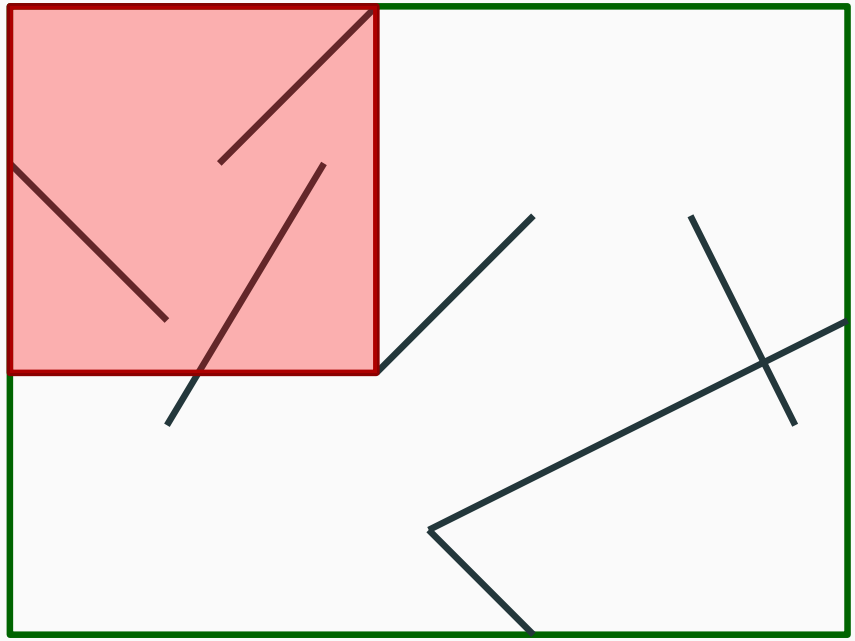
- Compute bounding box
- For each corner do:



$k = 3$

Algorithm

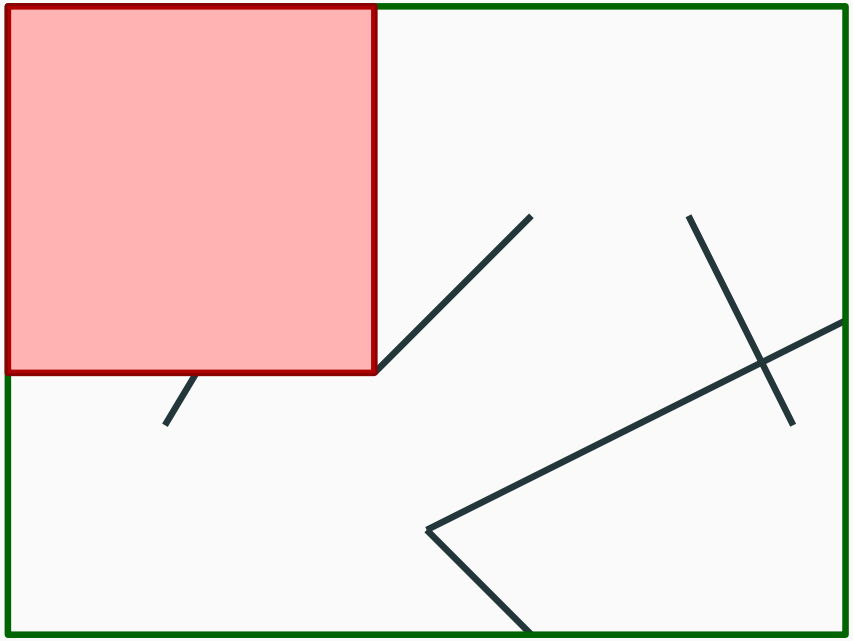
- Compute bounding box
- For each corner do:
  - Place square in corner



$k = 3$

Algorithm

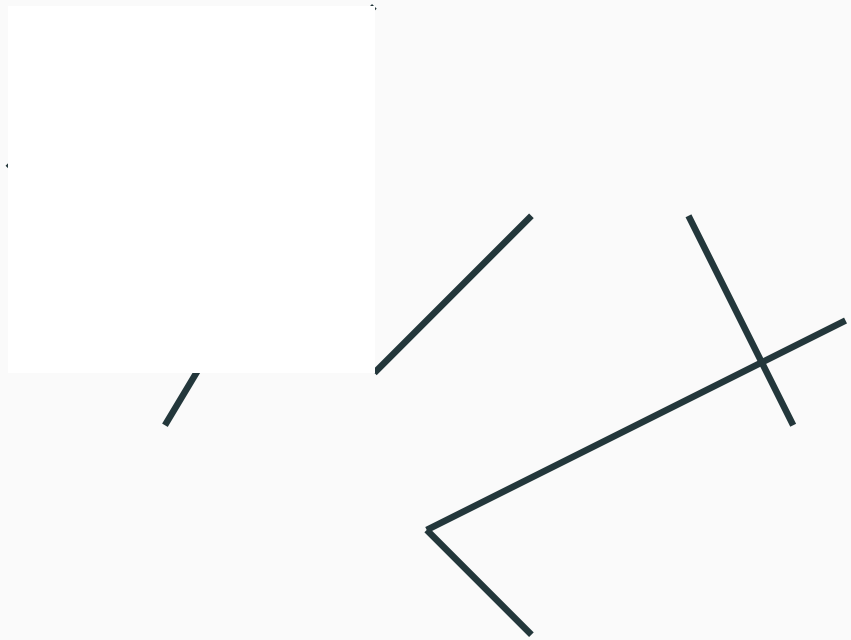
- Compute bounding box
- For each corner do:
  - Place square in corner
  - Ignore covered subsegmts



$k = 3$

Algorithm

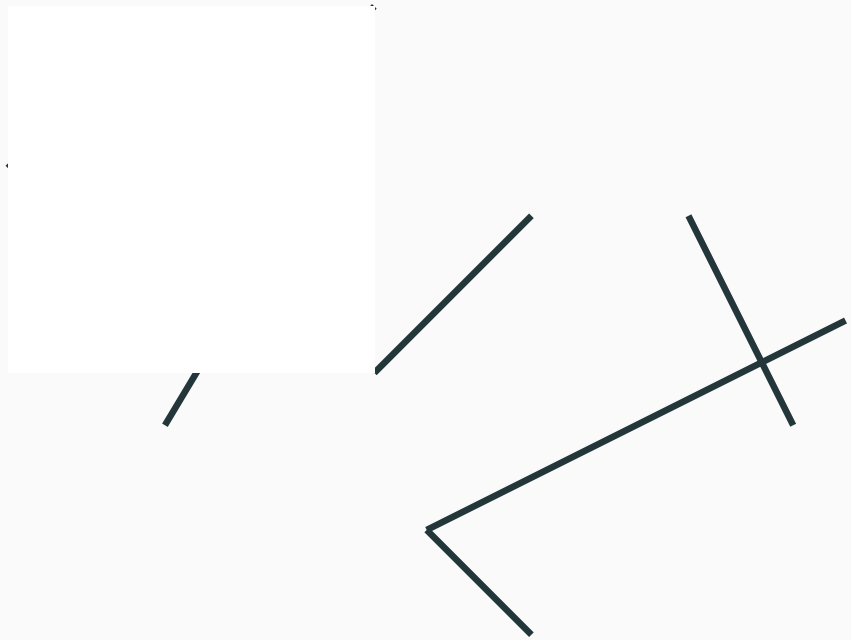
- Compute bounding box
- For each corner do:
  - Place square in corner
  - Ignore covered subsegmts
  - Remaining subsegments



$k = 3$

Algorithm

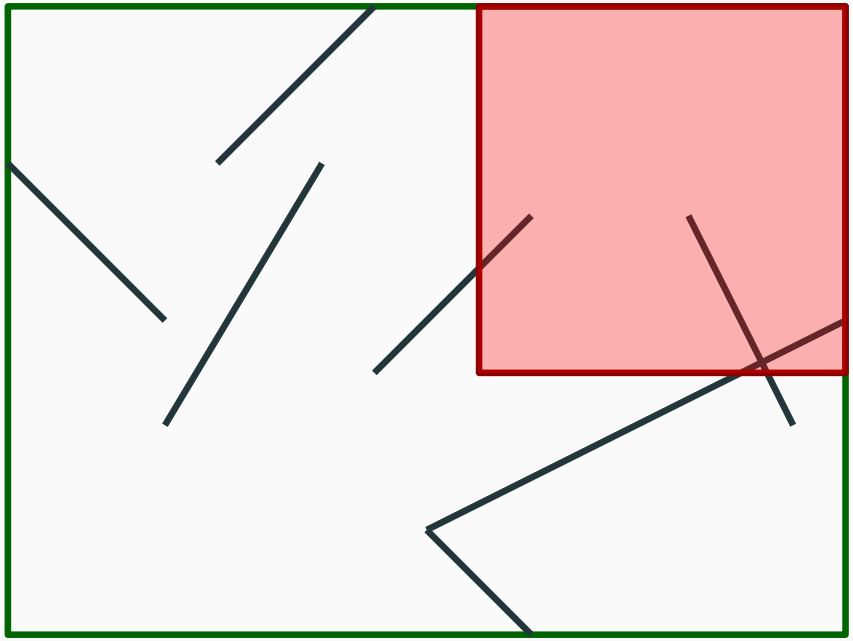
- Compute bounding box
- For each corner do:
  - Place square in corner
  - Ignore covered subsegmts
  - Remaining subsegments
  - Solve $k = 2$ subproblem



$k = 3$

Algorithm

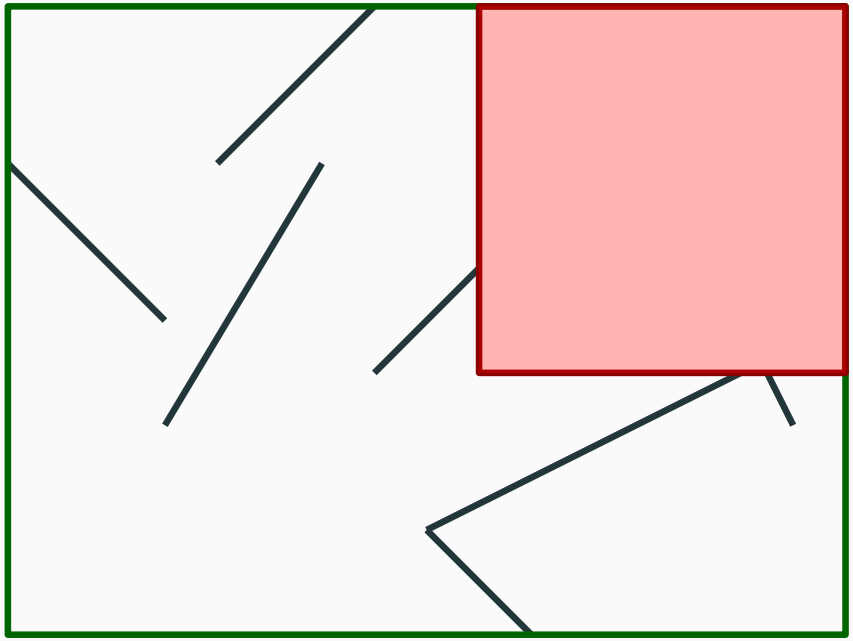
- Compute bounding box
- For each corner do:
  - Place square in corner



$k = 3$

Algorithm

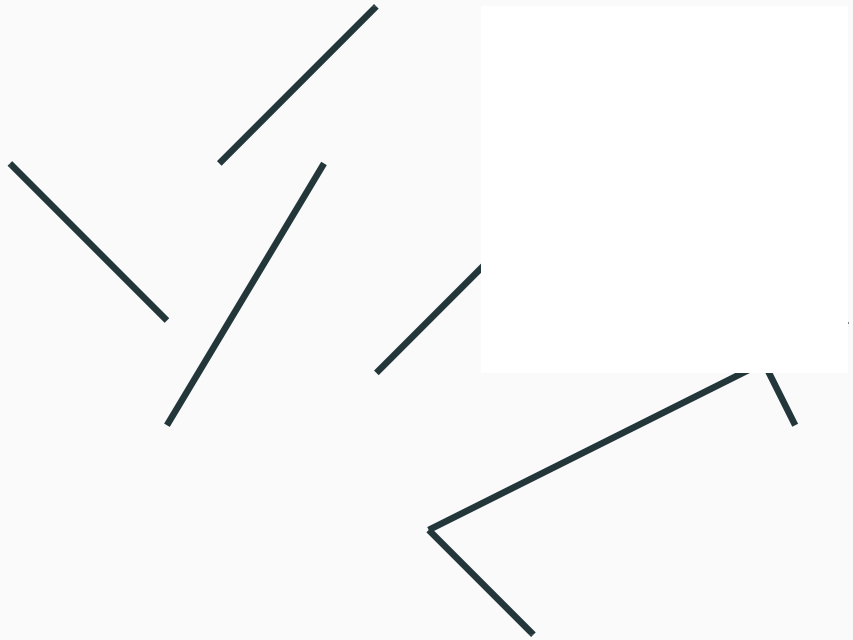
- Compute bounding box
- For each corner do:
  - Place square in corner
  - Ignore covered subsegmts



$k = 3$

Algorithm

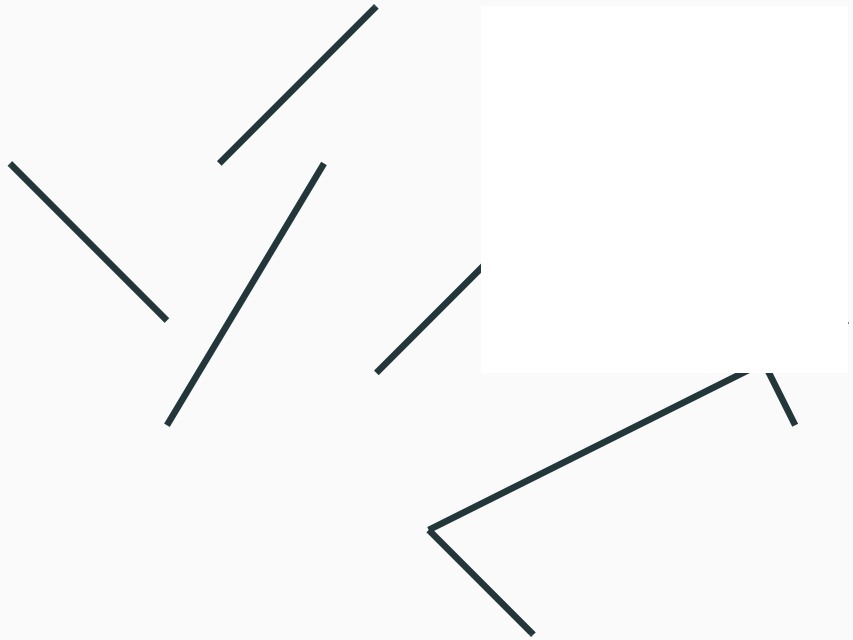
- Compute bounding box
- For each corner do:
  - Place square in corner
  - Ignore covered subsegmts
  - Remaining subsegments



$k = 3$

Algorithm

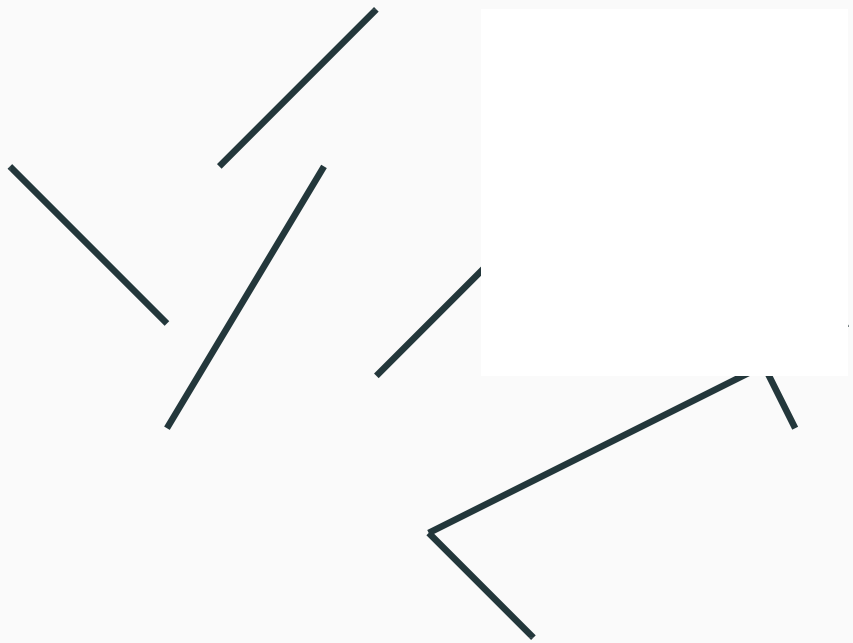
- Compute bounding box
- For each corner do:
  - Place square in corner
  - Ignore covered subsegmts
  - Remaining subsegments
  - Solve $k = 2$ subproblem



$k = 3$

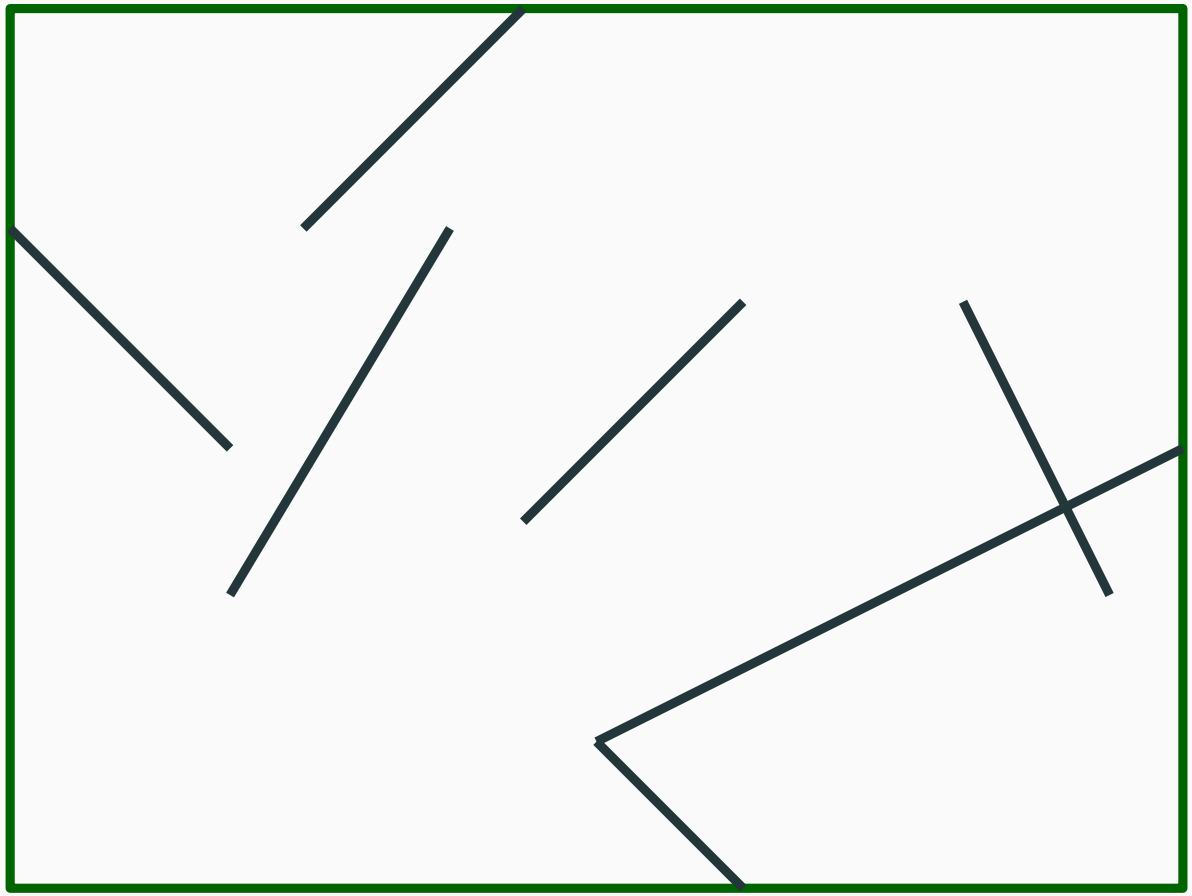
Algorithm

- Compute bounding box
- For each corner do:
  - Place square in corner
  - Ignore covered subsegmts
  - Remaining subsegments
  - Solve $k = 2$ subproblem
- Overall takes $O(n)$ time



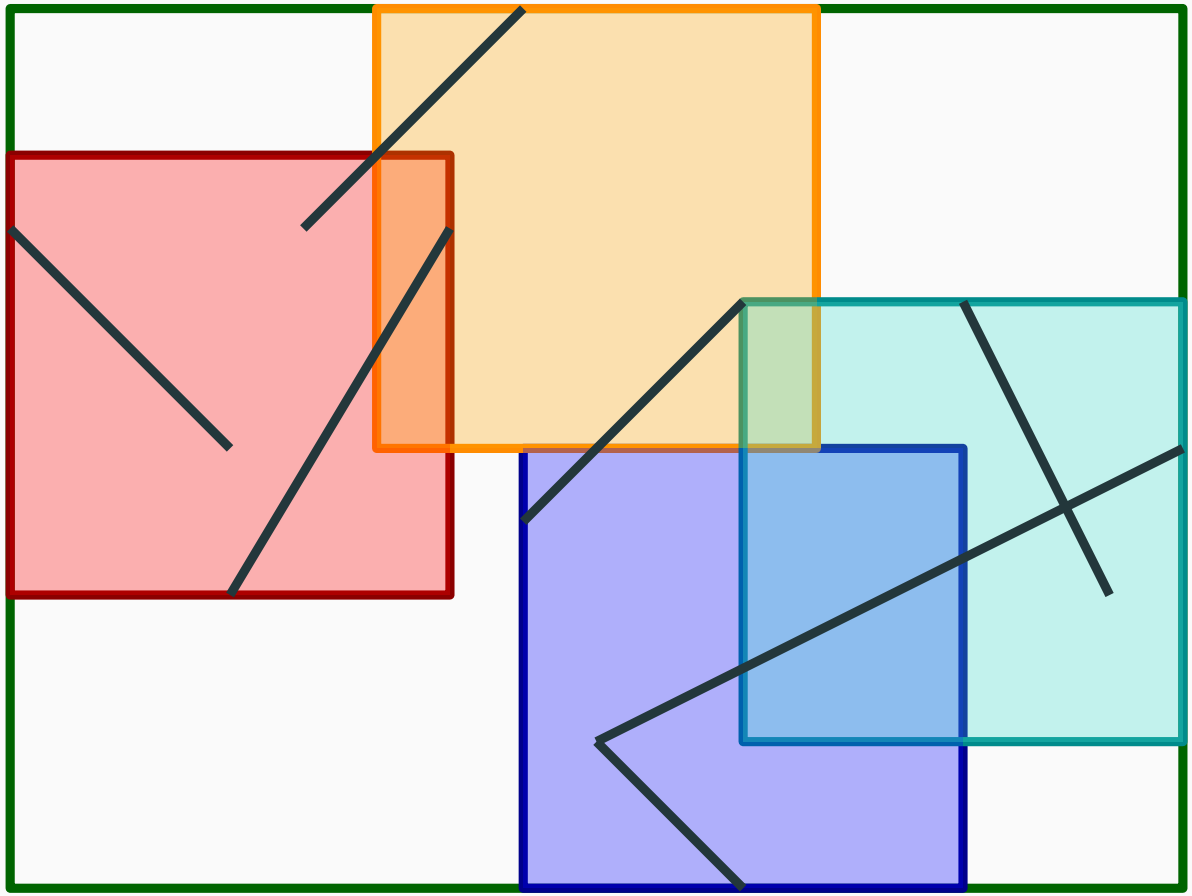
$k = 4$

$k = 4$



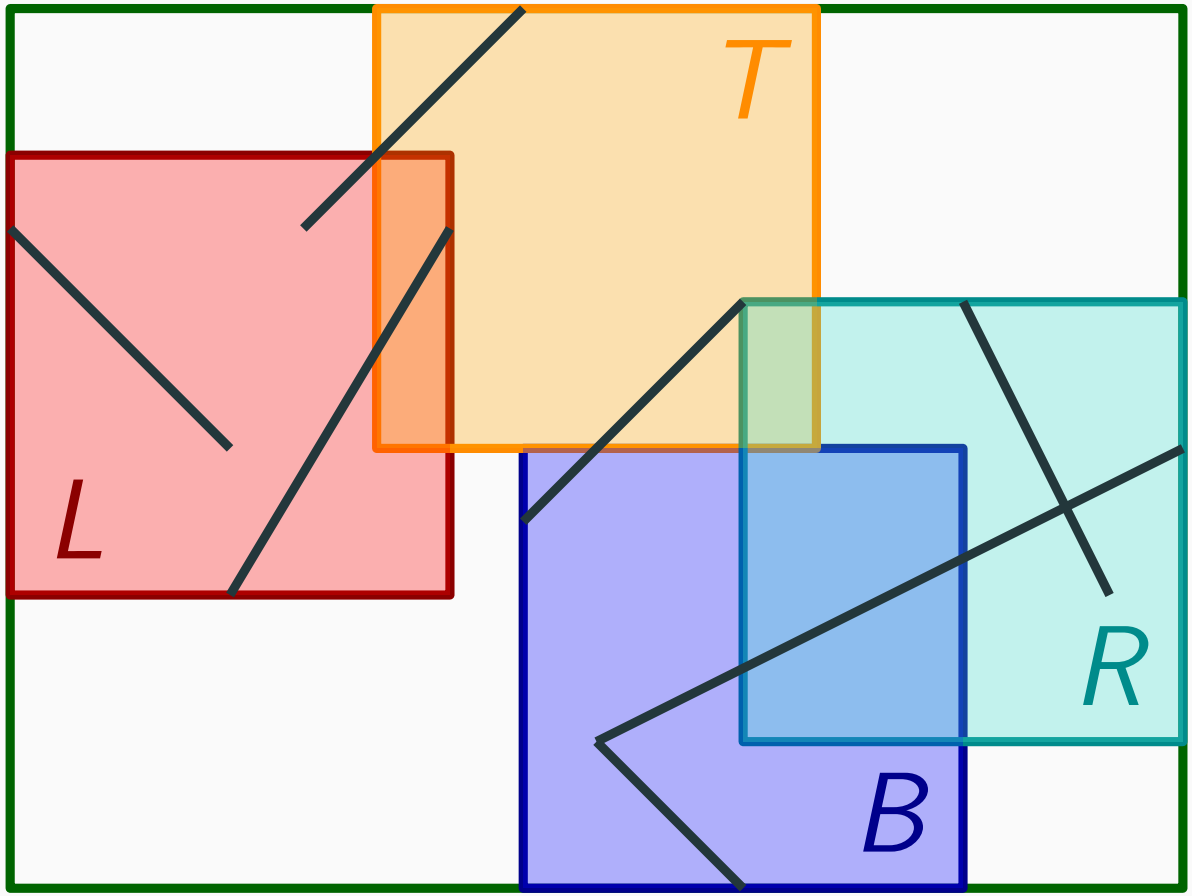
Can we make a similar observation?

$k = 4$



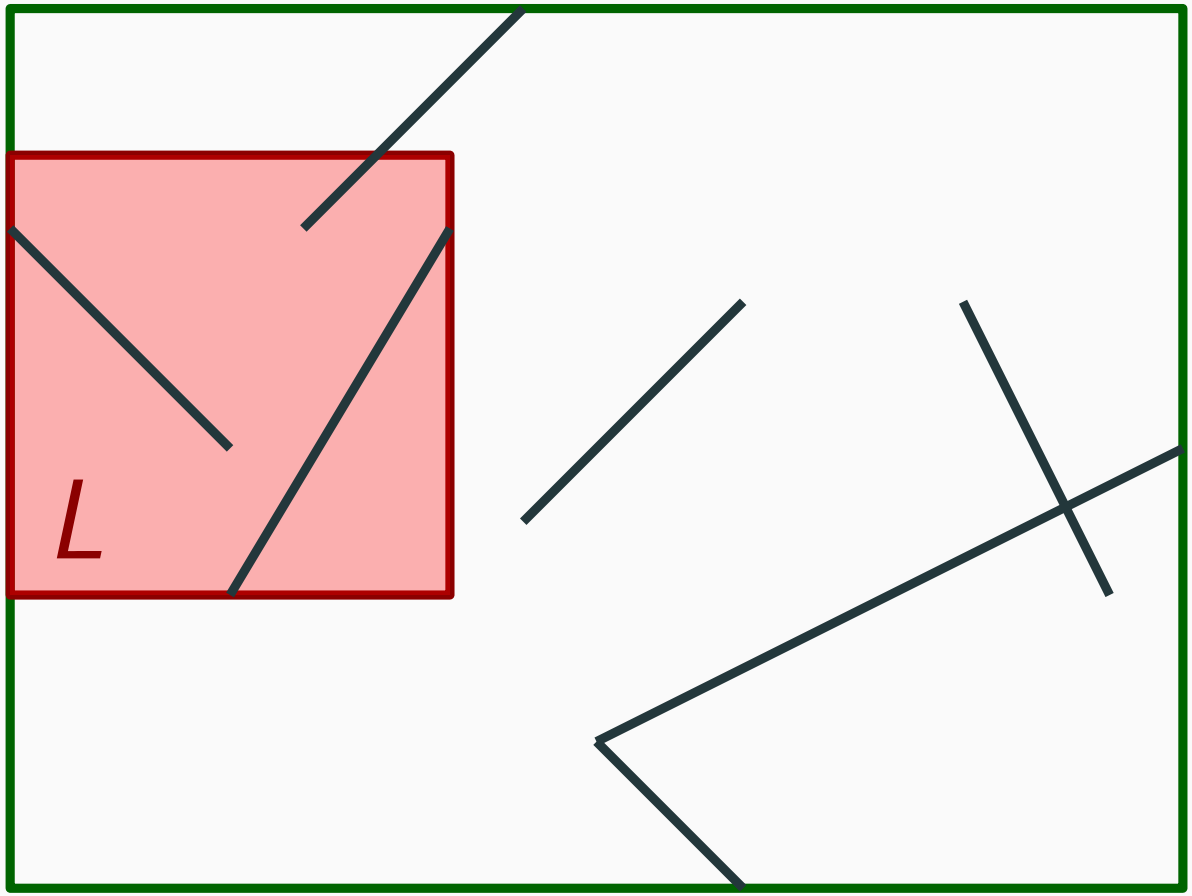
No corner box!

$k = 4$



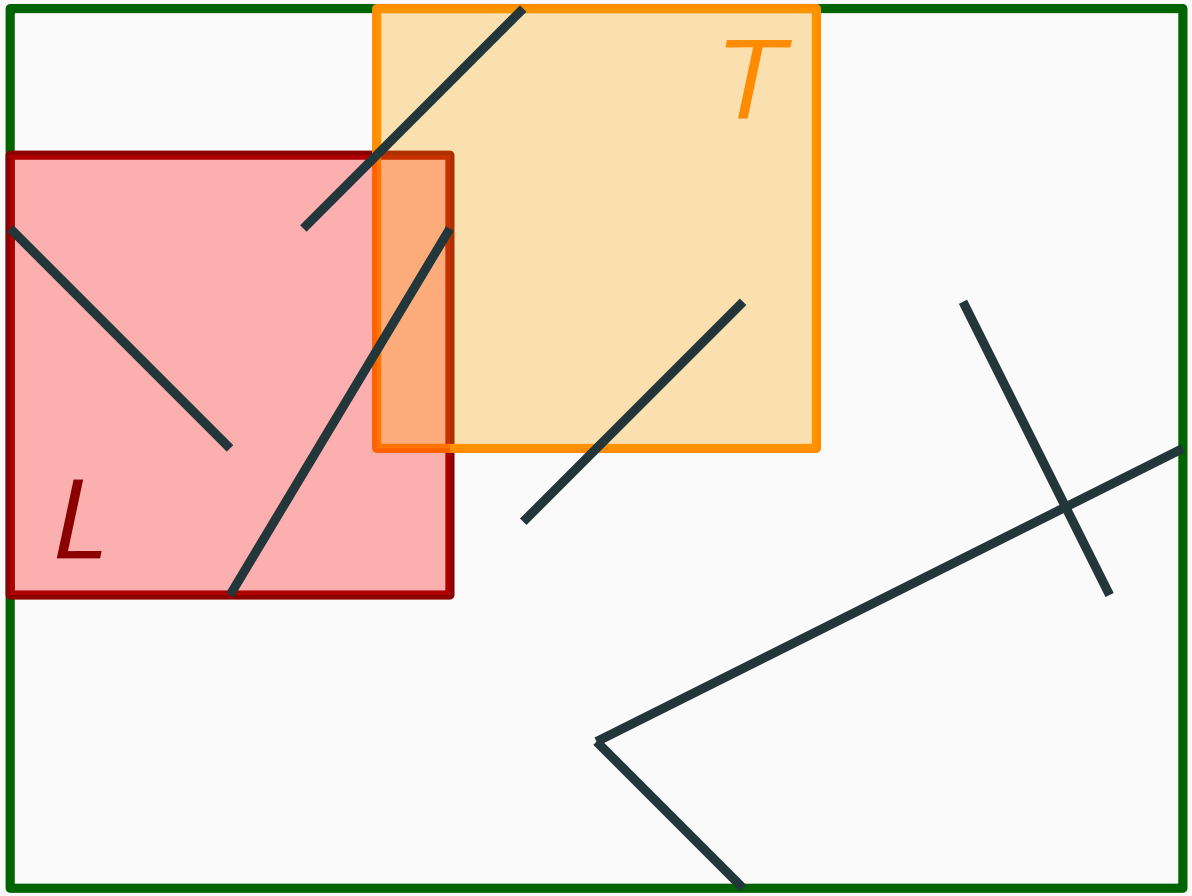
WLOG the left to right order is L, T, B, R.

$k = 4$



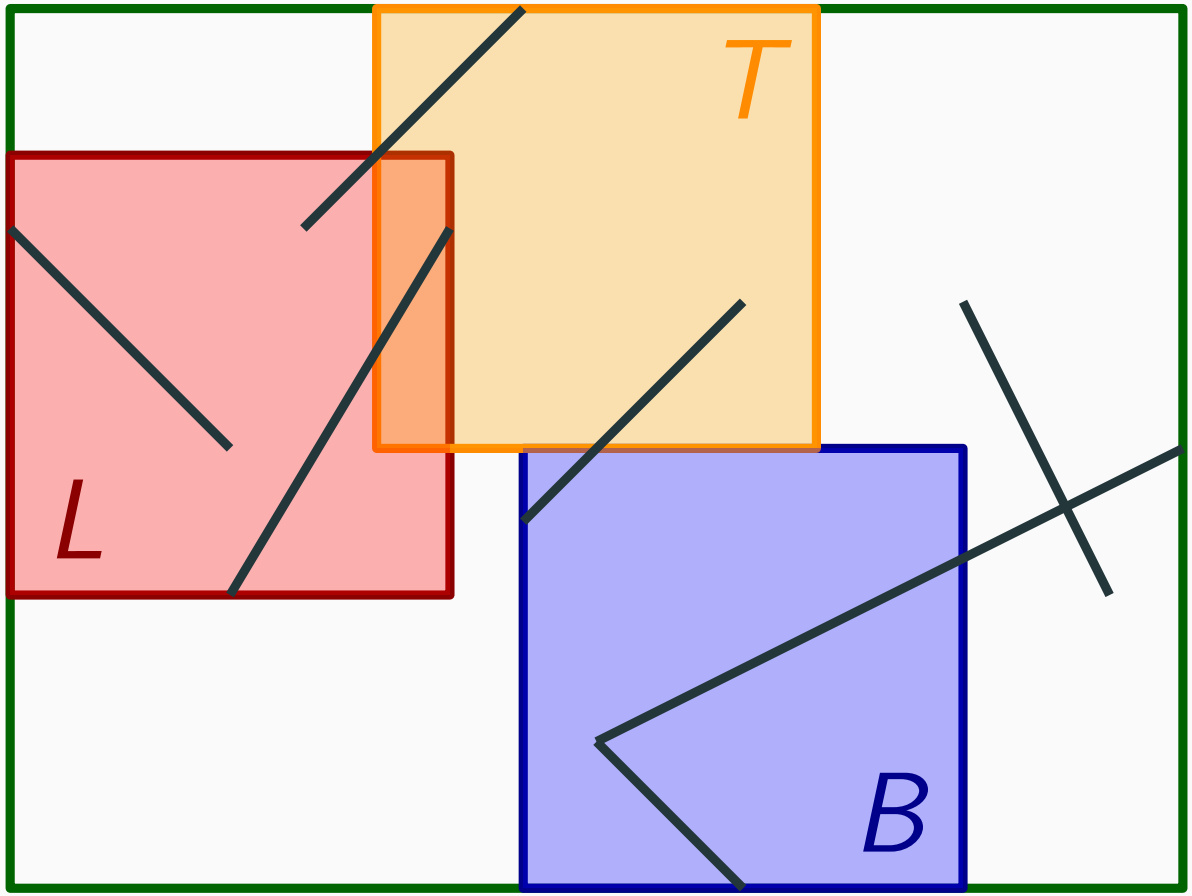
Consider when L is known. Do we know where to put the next box?

$k = 4$



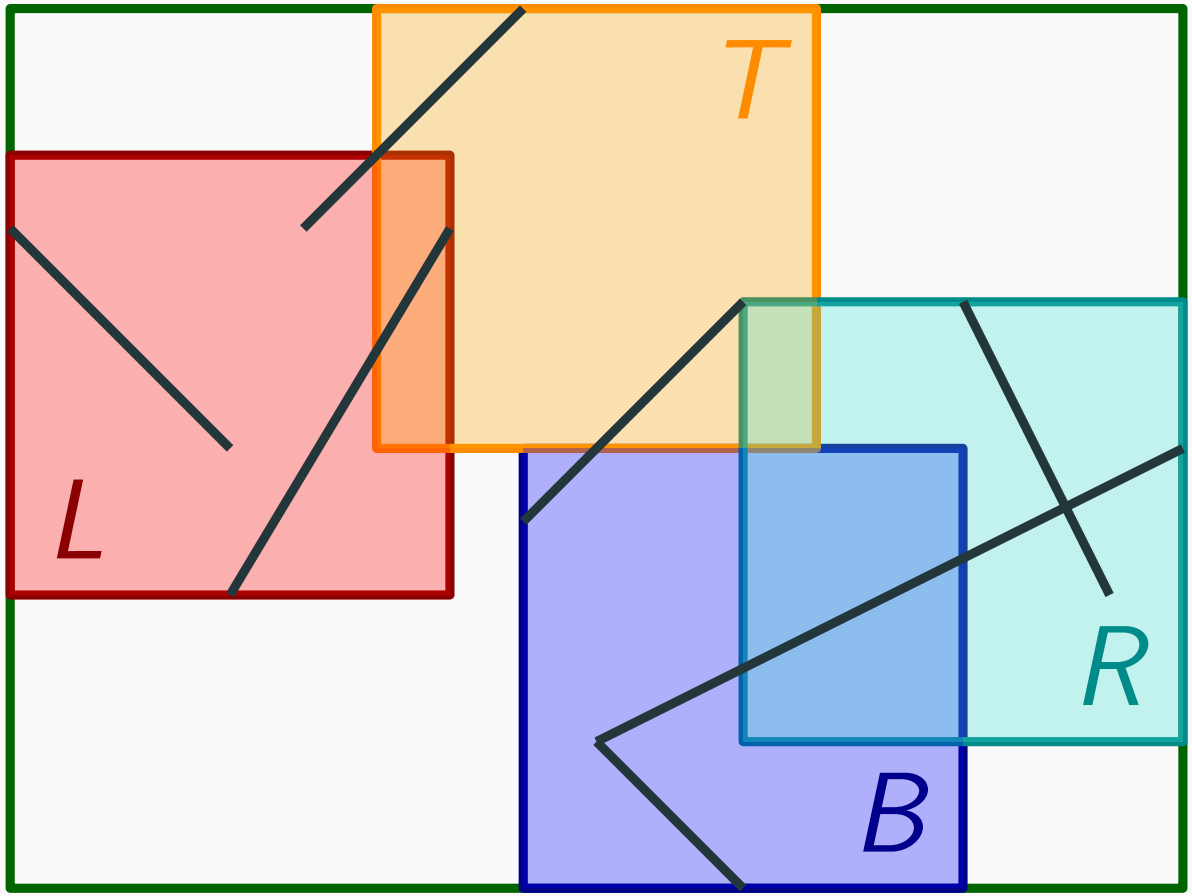
Place T at leftmost point of leftovers.

$k = 4$



Place B at leftmost point of leftovers.

$k = 4$



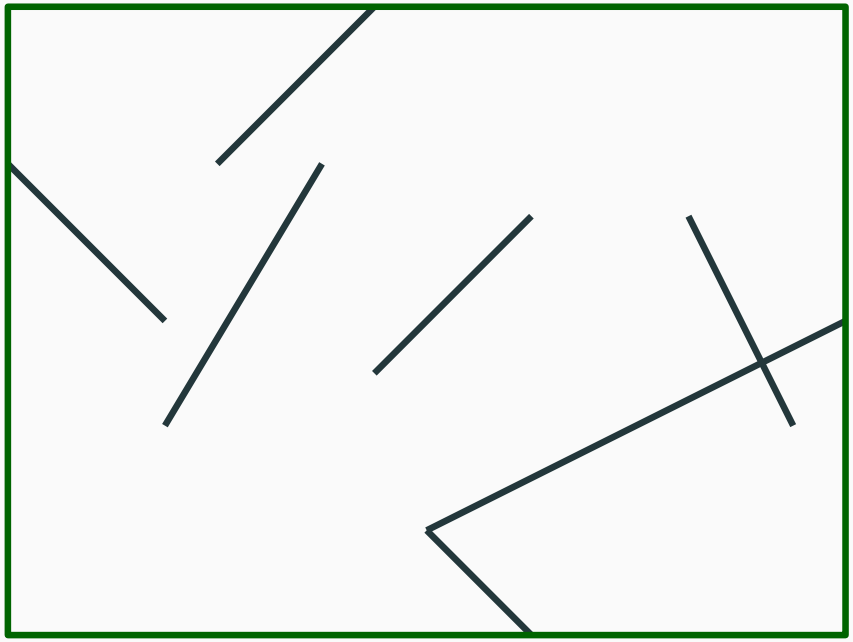
Place R to cover the rest (if big enough).

Lemma

As we vary the position of L, positions of T, B are R are piecewise linear functions which can be computed in $O(n \log n)$.

Algorithm

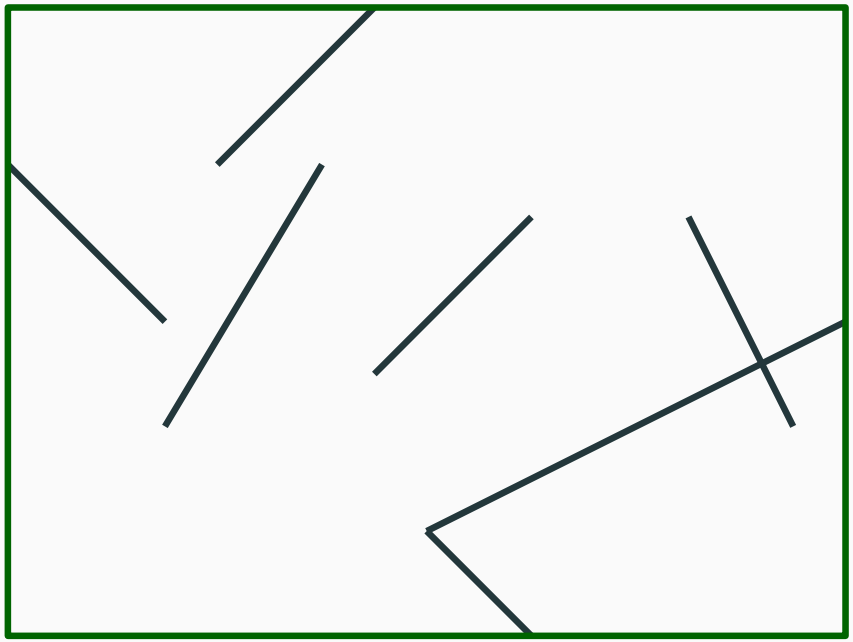
- Compute bounding box



$k = 4$

Algorithm

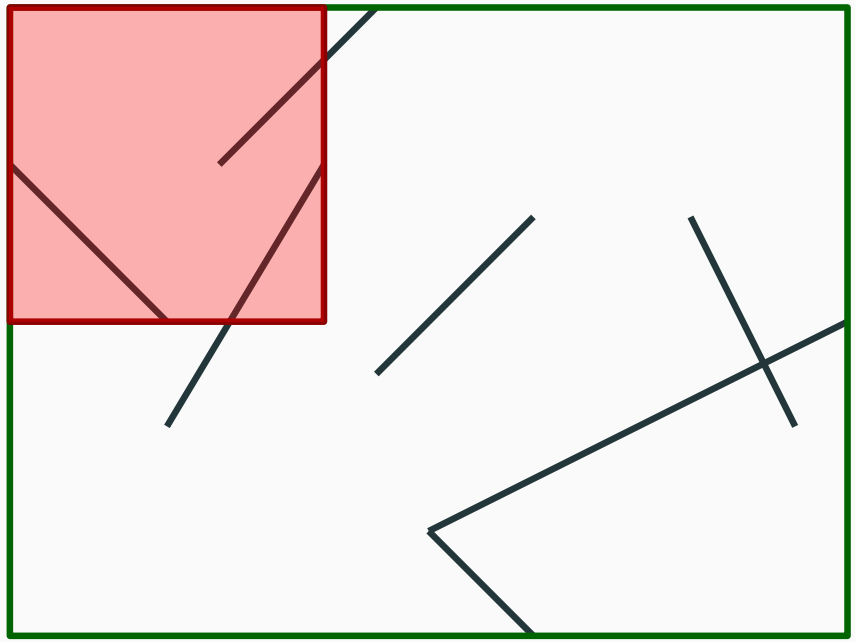
- Compute bounding box
- For each corner do:



$k = 4$

Algorithm

- Compute bounding box
- For each corner do:
  - Place square in corner



$k = 4$

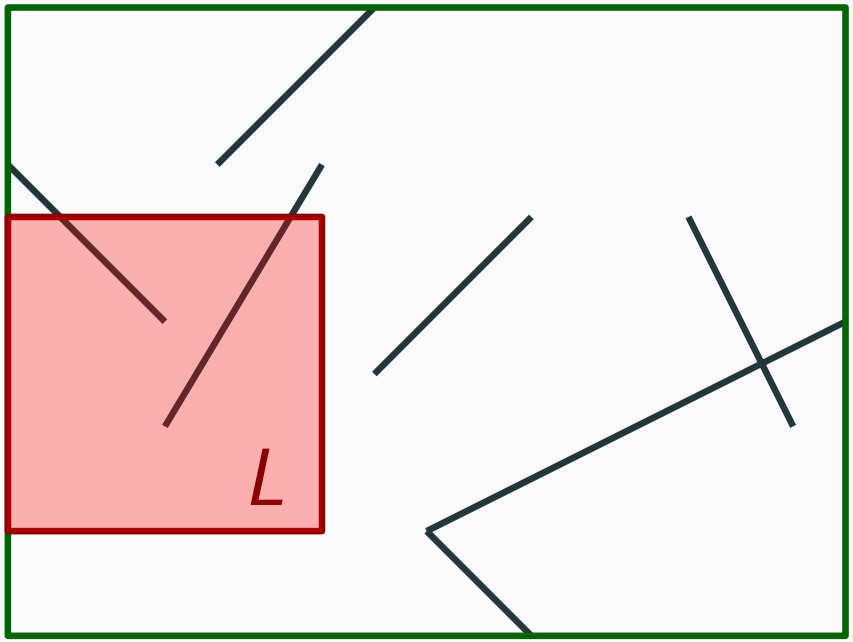
Algorithm

- Compute bounding box
- For each corner do:
  - Place square in corner
  - Solve $k = 3$ subproblem

$k = 4$

Algorithm

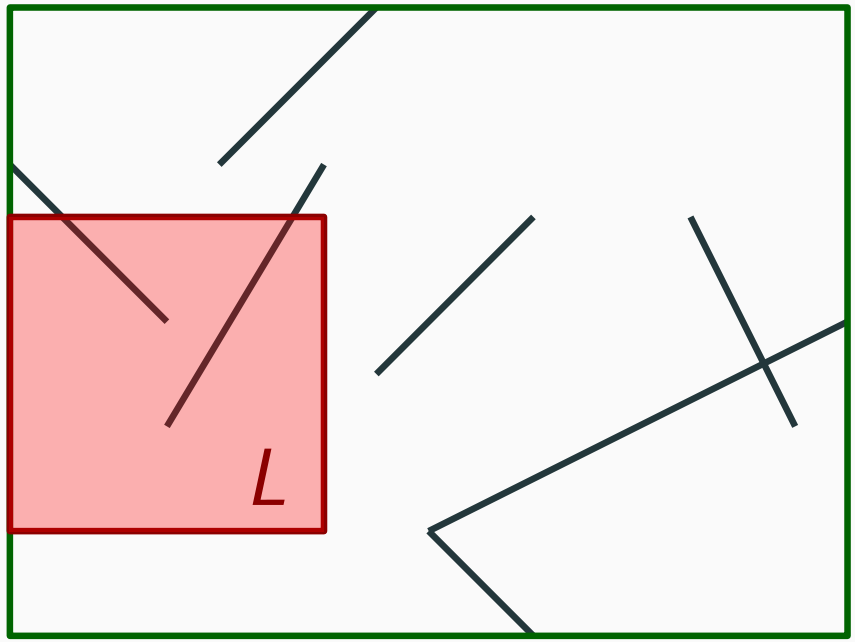
- Compute bounding box
- For each corner do:
  - Place square in corner
  - Solve $k = 3$ subproblem
- Let L be on left side



$k = 4$

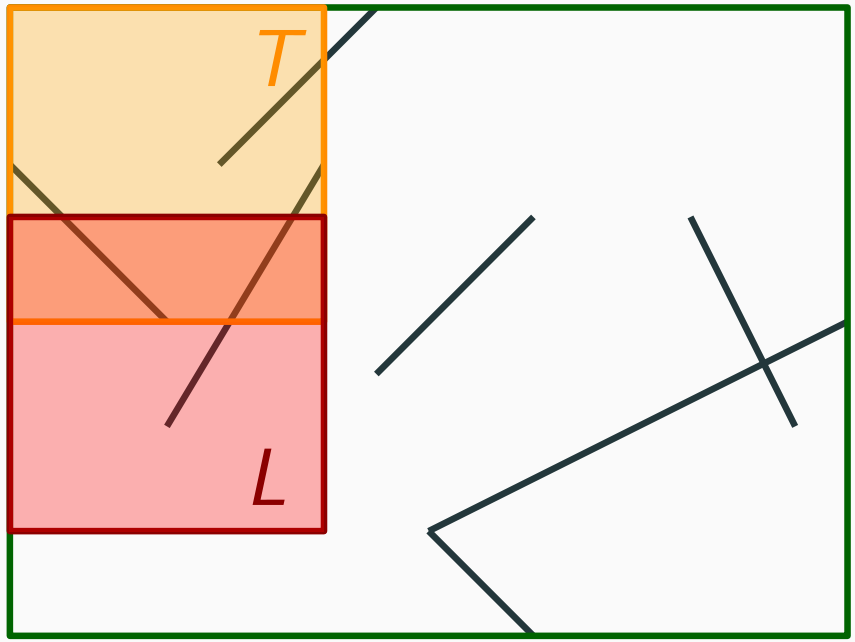
Algorithm

- Compute bounding box
- For each corner do:
  - Place square in corner
  - Solve $k = 3$ subproblem
- Let L be on left side
- As L varies do:



Algorithm

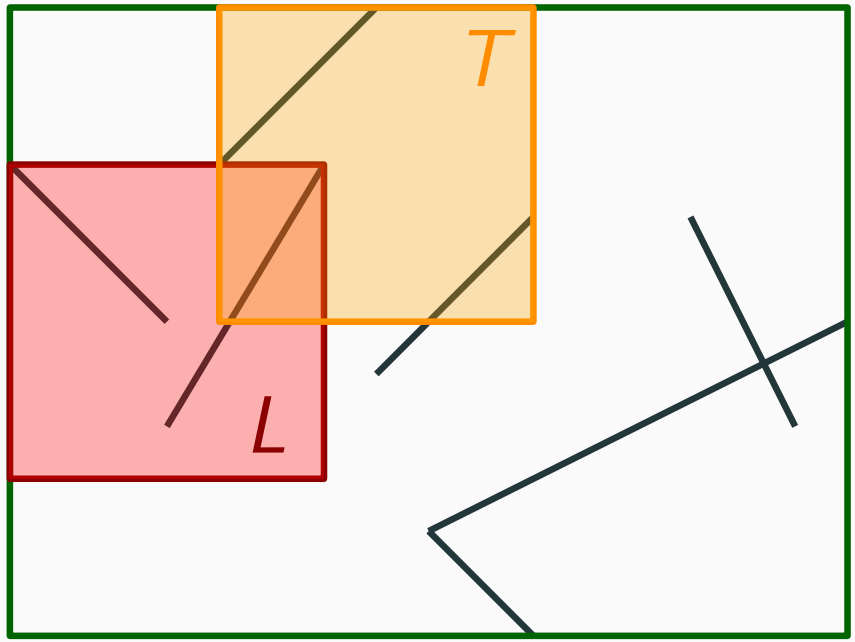
- Compute bounding box
- For each corner do:
  - Place square in corner
  - Solve $k = 3$ subproblem
- Let L be on left side
- As L varies do:
  - Compute function for T



$k = 4$

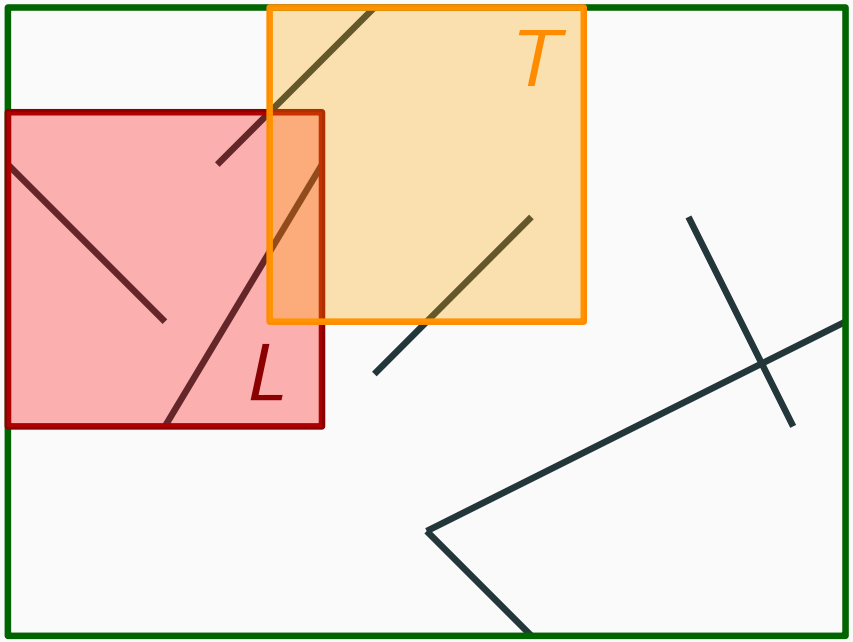
Algorithm

- Compute bounding box
- For each corner do:
  - Place square in corner
  - Solve $k = 3$ subproblem
- Let L be on left side
- As L varies do:
  - Compute function for T



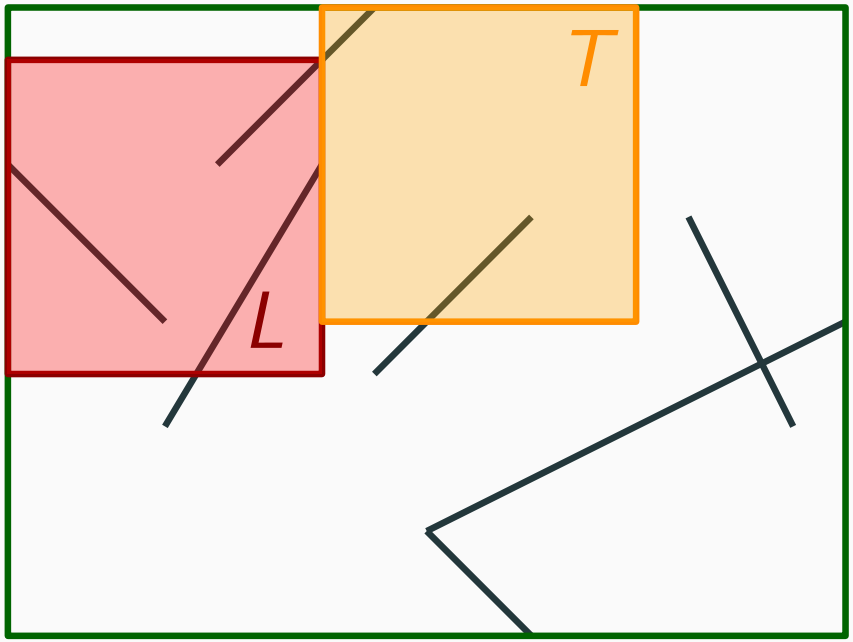
Algorithm

- Compute bounding box
- For each corner do:
  - Place square in corner
  - Solve $k = 3$ subproblem
- Let L be on left side
- As L varies do:
  - Compute function for T



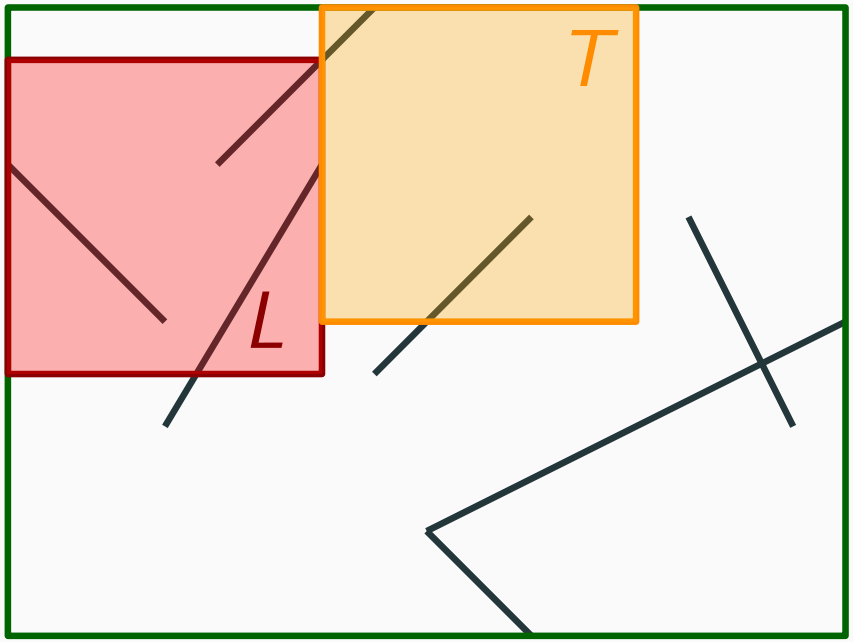
Algorithm

- Compute bounding box
- For each corner do:
  - Place square in corner
  - Solve $k = 3$ subproblem
- Let L be on left side
- As L varies do:
  - Compute function for T



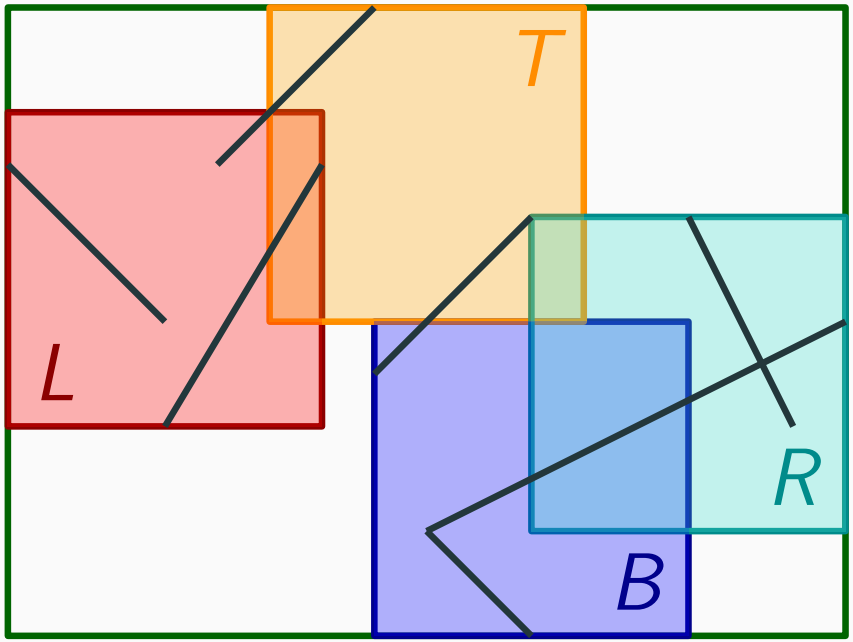
Algorithm

- Compute bounding box
- For each corner do:
  - Place square in corner
  - Solve $k = 3$ subproblem
- Let L be on left side
- As L varies do:
  - Compute function for T
  - Compute function for B
  - Compute function for R



Algorithm

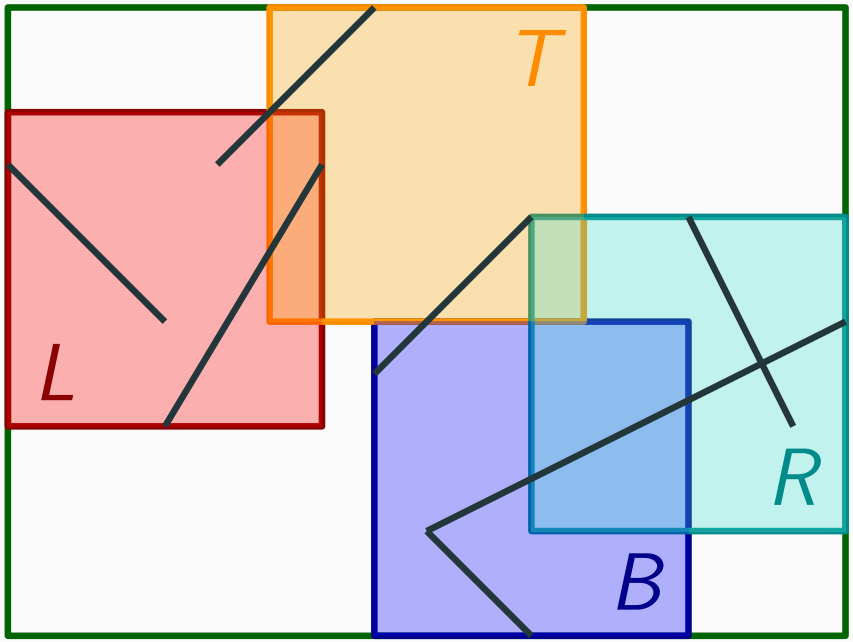
- Compute bounding box
- For each corner do:
  - Place square in corner
  - Solve $k = 3$ subproblem
- Let L be on left side
- As L varies do:
  - Compute function for T
  - Compute function for B
  - Compute function for R
- Answer if L can be placed so that all segments covered



$k = 4$

Algorithm

- Compute bounding box
- For each corner do:
  - Place square in corner
  - Solve $k = 3$ subproblem
- Let L be on left side
- As L varies do:
  - Compute function for T
  - Compute function for B
  - Compute function for R
- Answer if L can be placed so that all segments covered
- Overall $O(n \log n)$ time



Future Work

- Can we extend to $k \geq 5$?

Three Problems

Three Problems

1. Decide if a set of segments is k-coverable.
2. Data structure answers if subtrajectory is k-coverable.
3. Find longest k-coverable subtrajectory.

Three Problems

1. Decide if a set of segments is k-coverable.
2. Data structure answers if subtrajectory is k-coverable.
3. Find longest k-coverable subtrajectory.

	$k = 2$	$k = 3$	$k = 4$
Problem 1	✓	✓	✓
Problem 2	✓	✓	
Problem 3	✓		

Running times are $O(n \cdot \text{polylog}(n))$. Query times are $O(\log n)$.

Thank you