Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems

Institute for Informatics

Advanced Algorithms

Winter term 2019/20

Lecture 3. 2D Linear Programming via sweep-lines and randomization

Source: **CG: A&A** §4

Steven Chaplick & Alexander Wolff *Chair for Computer Science I*

Maximizing Profit

You are the boss of a small company that produces two products, P_1 and P_2. If you produce x_1 units of P_1 and x_2 units of P_2, your profit in € is

$$G(x_1, x_2) = 300x_1 + 500x_2$$

Maximizing Profit

You are the boss of a small company that produces two products, P_1 and P_2. If you produce x_1 units of P_1 and x_2 units of P_2, your profit in € is

$$G(x_1, x_2) = 300x_1 + 500x_2$$

Your production runs on three machines M_A, M_B, and M_C with the following capacities:

$$\begin{aligned} M_A: &\quad 4x_1 + 11x_2 \leq 880 \\ M_B: &\quad x_1 + x_2 \leq 150 \\ M_C: &\quad x_2 \leq 60 \end{aligned}$$

Maximizing Profit

You are the boss of a small company that produces two products, P_1 and P_2. If you produce x_1 units of P_1 and x_2 units of P_2, your profit in € is

$$G(x_1, x_2) = 300x_1 + 500x_2$$

Your production runs on three machines M_A, M_B, and M_C with the following capacities:

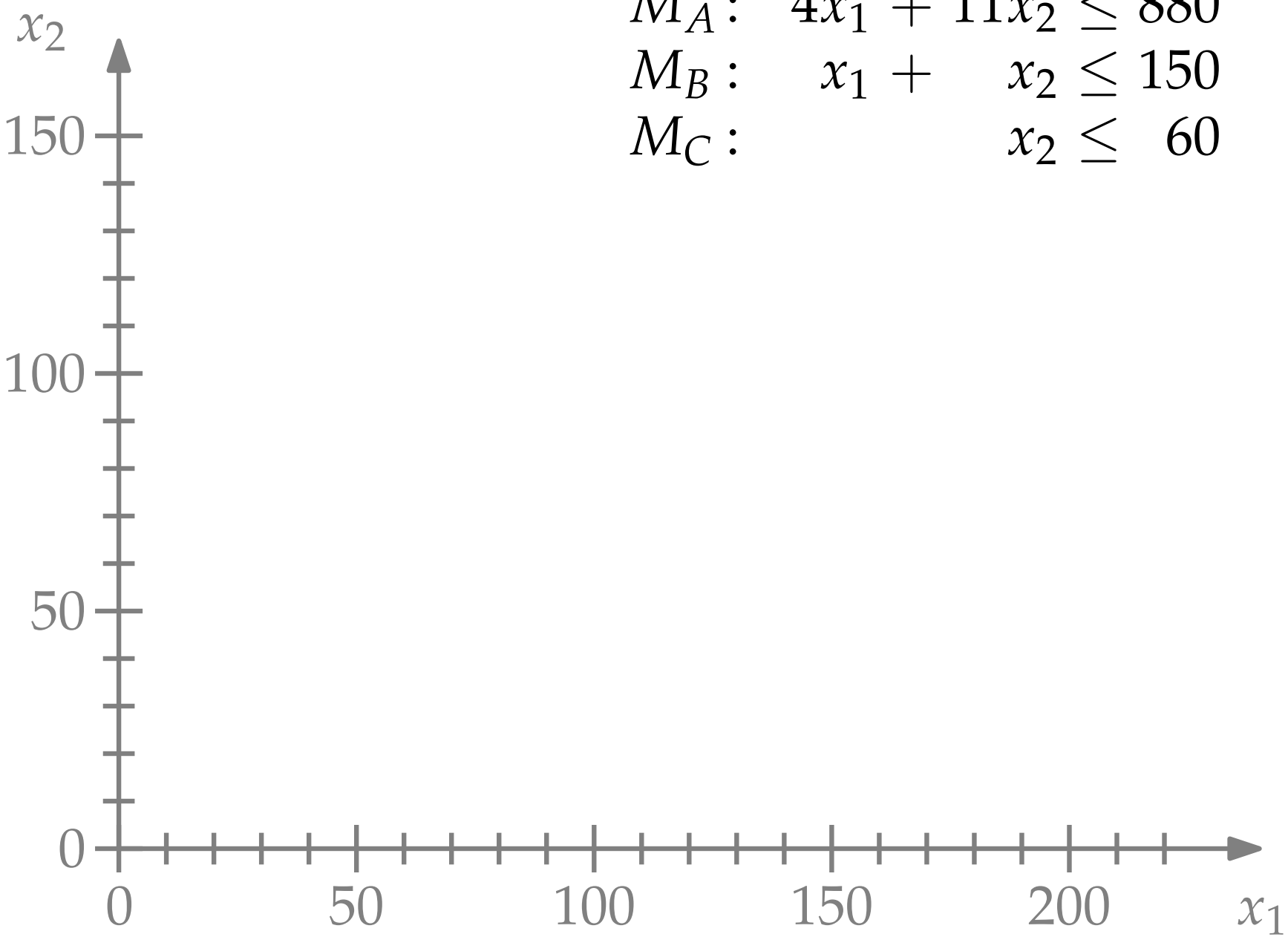
$$\begin{aligned} M_A: &\quad 4x_1 + 11x_2 \leq 880 \\ M_B: &\quad x_1 + x_2 \leq 150 \\ M_C: &\quad x_2 \leq 60 \end{aligned}$$

Which choice of (x_1, x_2) maximizes your profit?

The Answer

linear constraints:

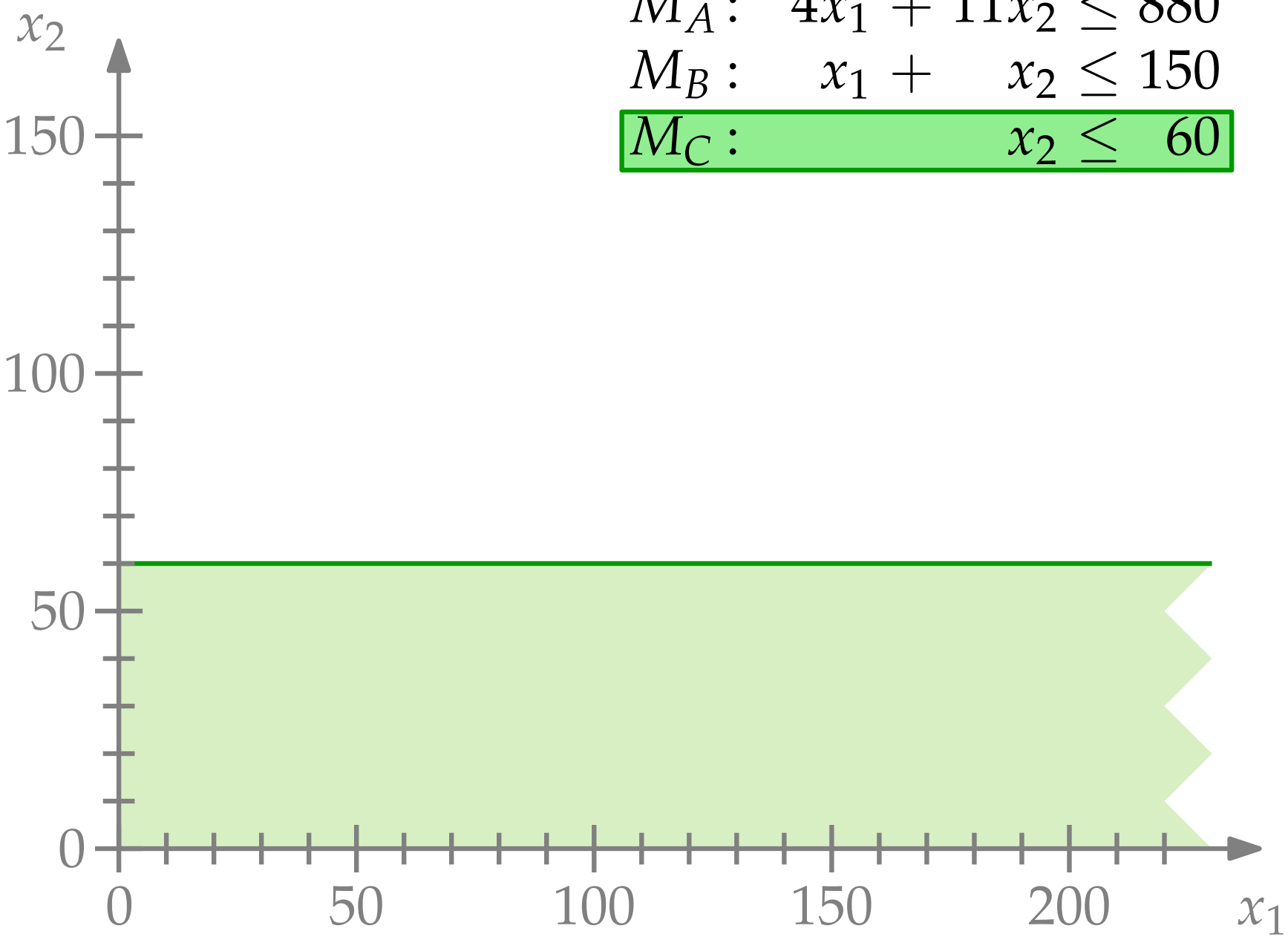
$$
\begin{aligned}
M_A: &\quad 4x_1 + 11x_2 \leq 880 \\
M_B: &\quad x_1 + x_2 \leq 150 \\
M_C: &\quad x_2 \leq 60
\end{aligned}
$$



The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$



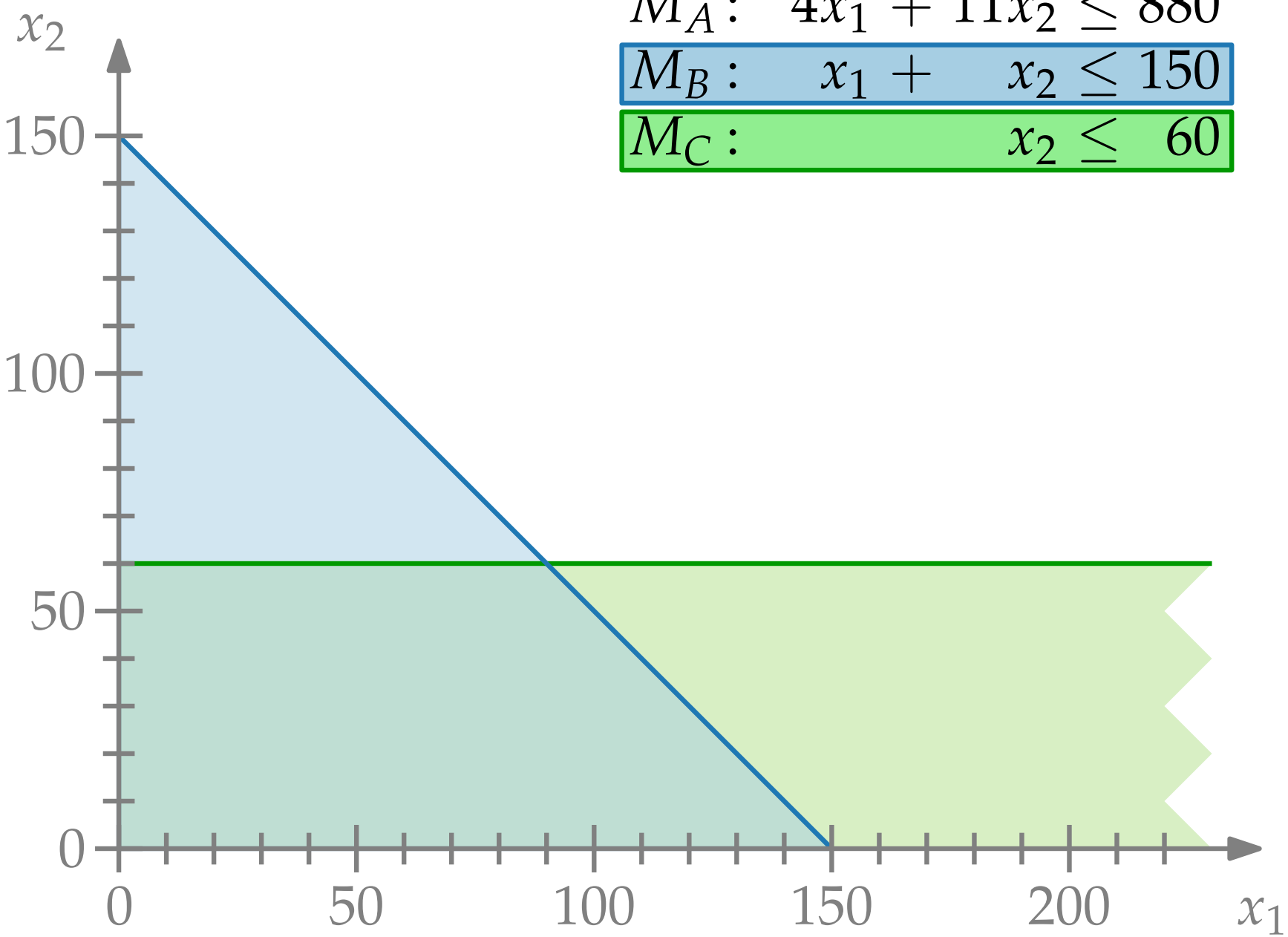
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$

$$M_B: \quad x_1 + x_2 \leq 150$$

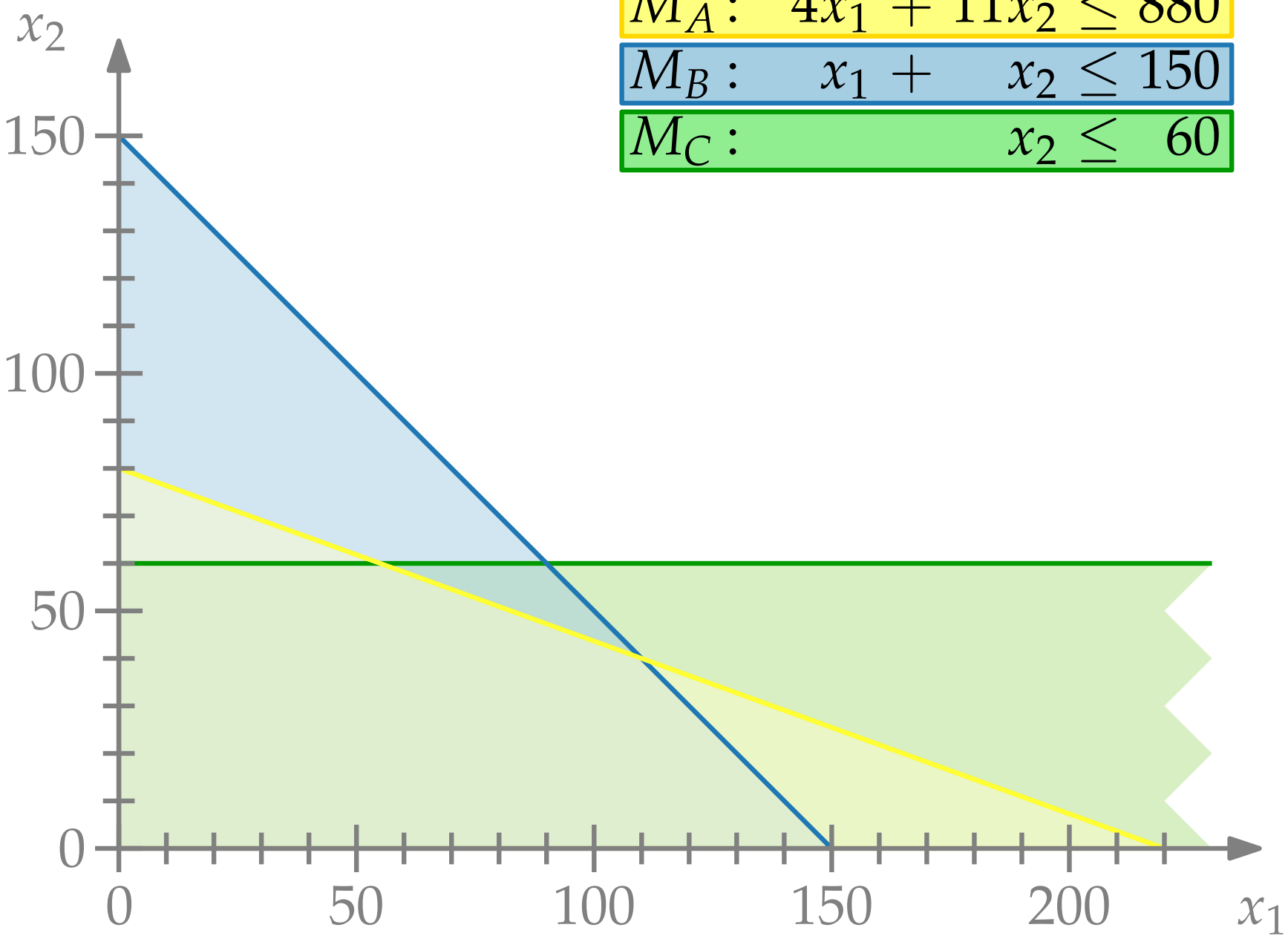
$$M_C: \quad x_2 \leq 60$$



The Answer

linear constraints:

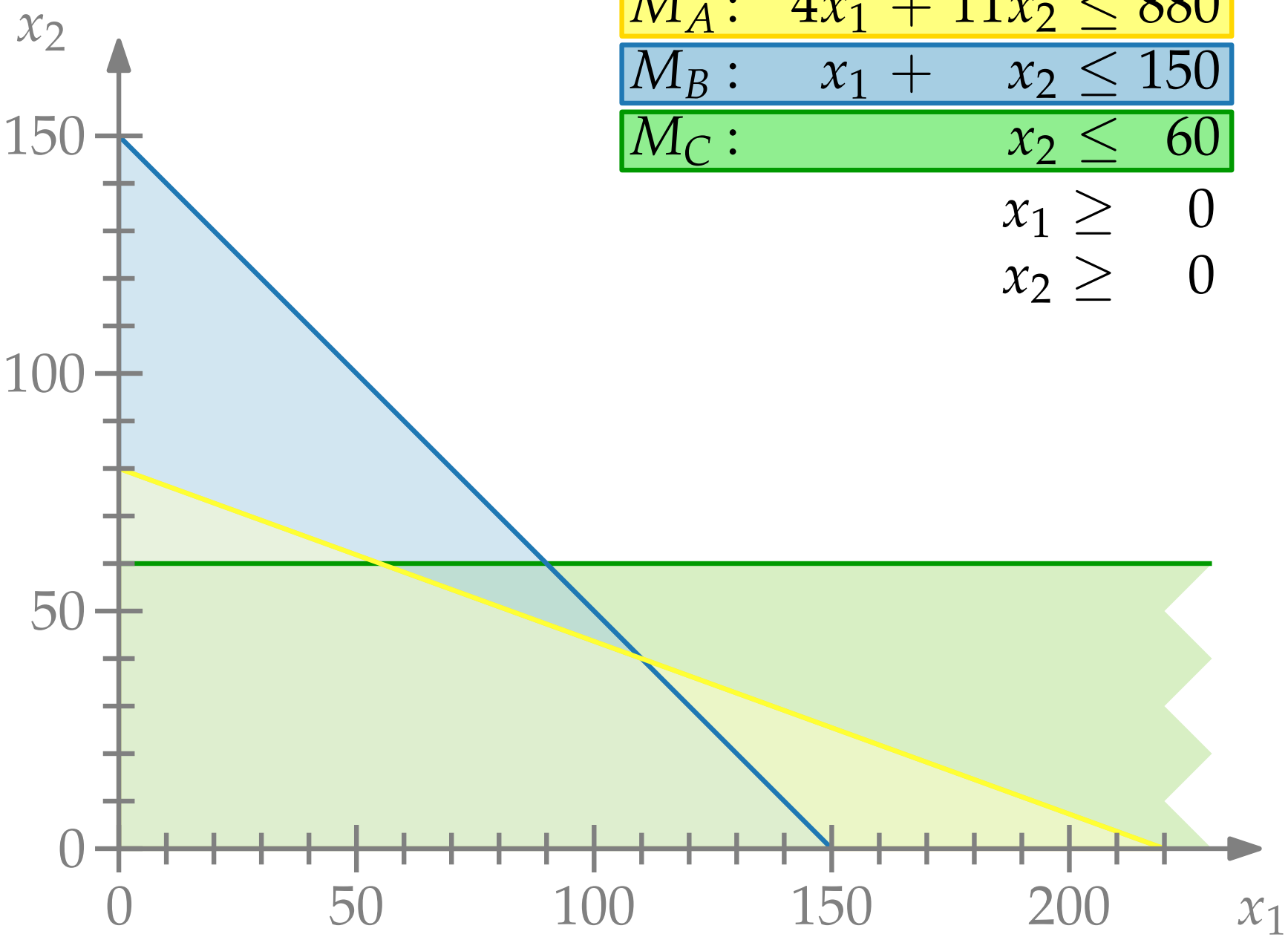
$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$



The Answer

linear constraints:

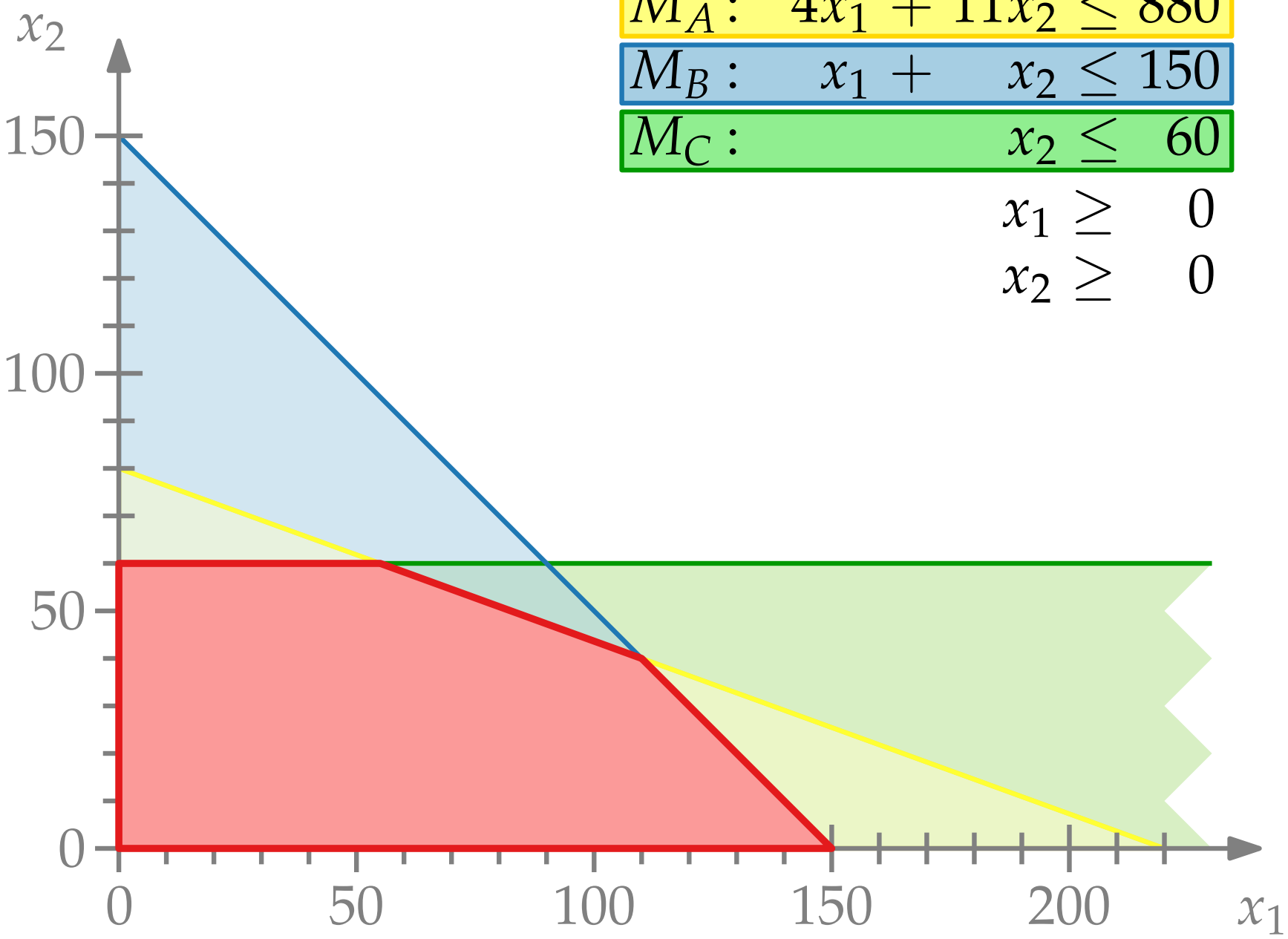
$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$



The Answer

linear constraints:

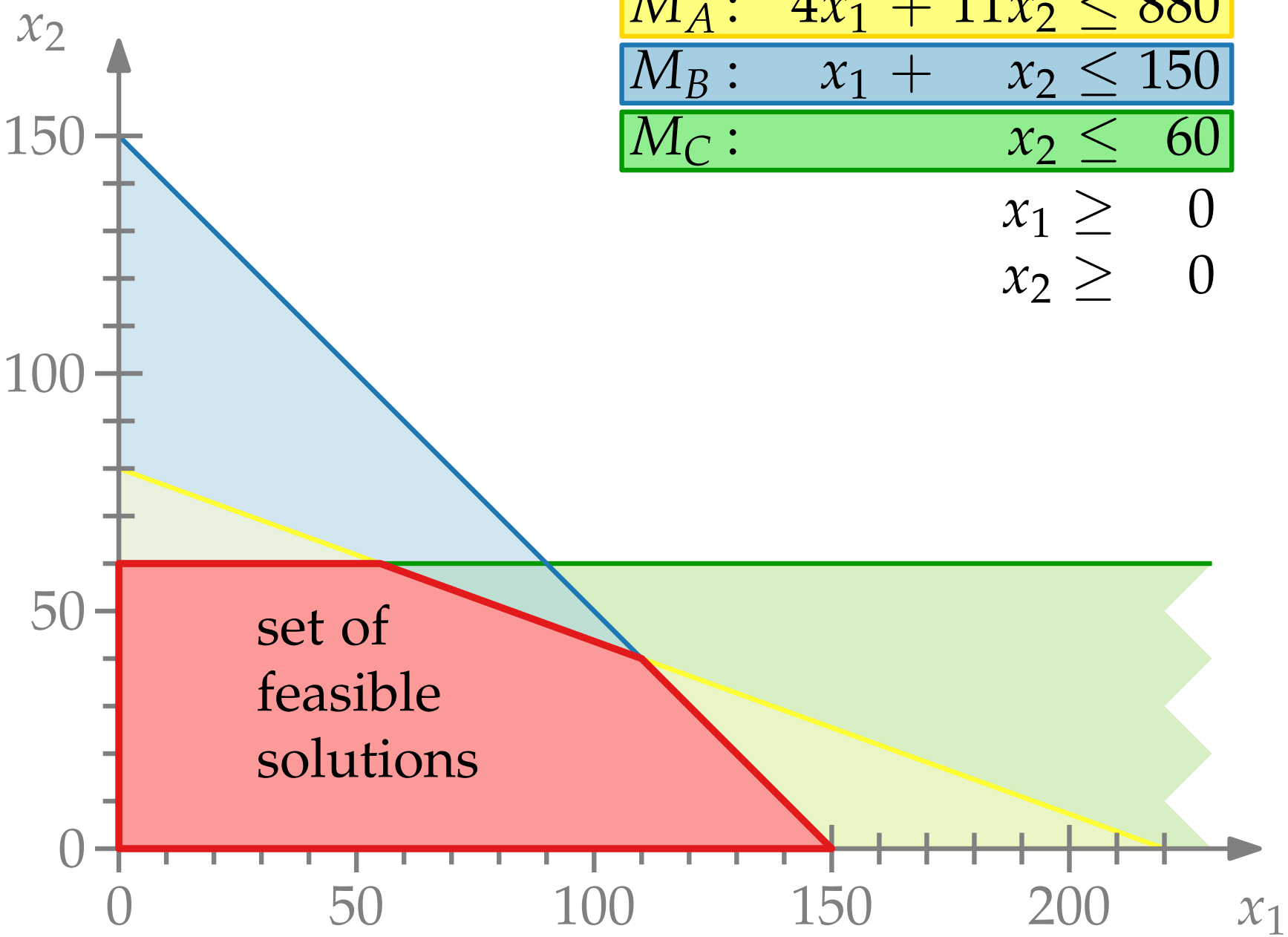
$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$



The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$



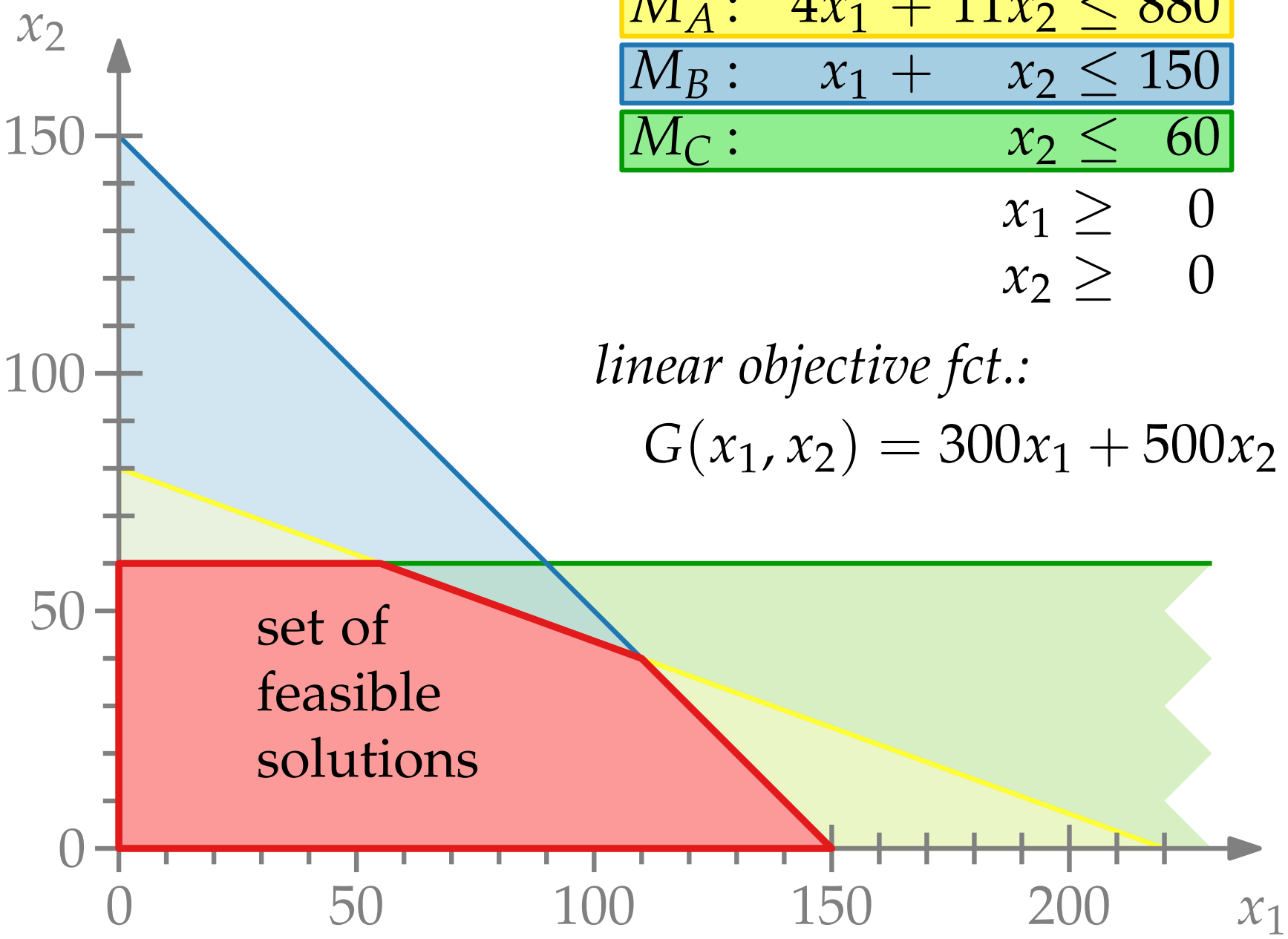
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2$$



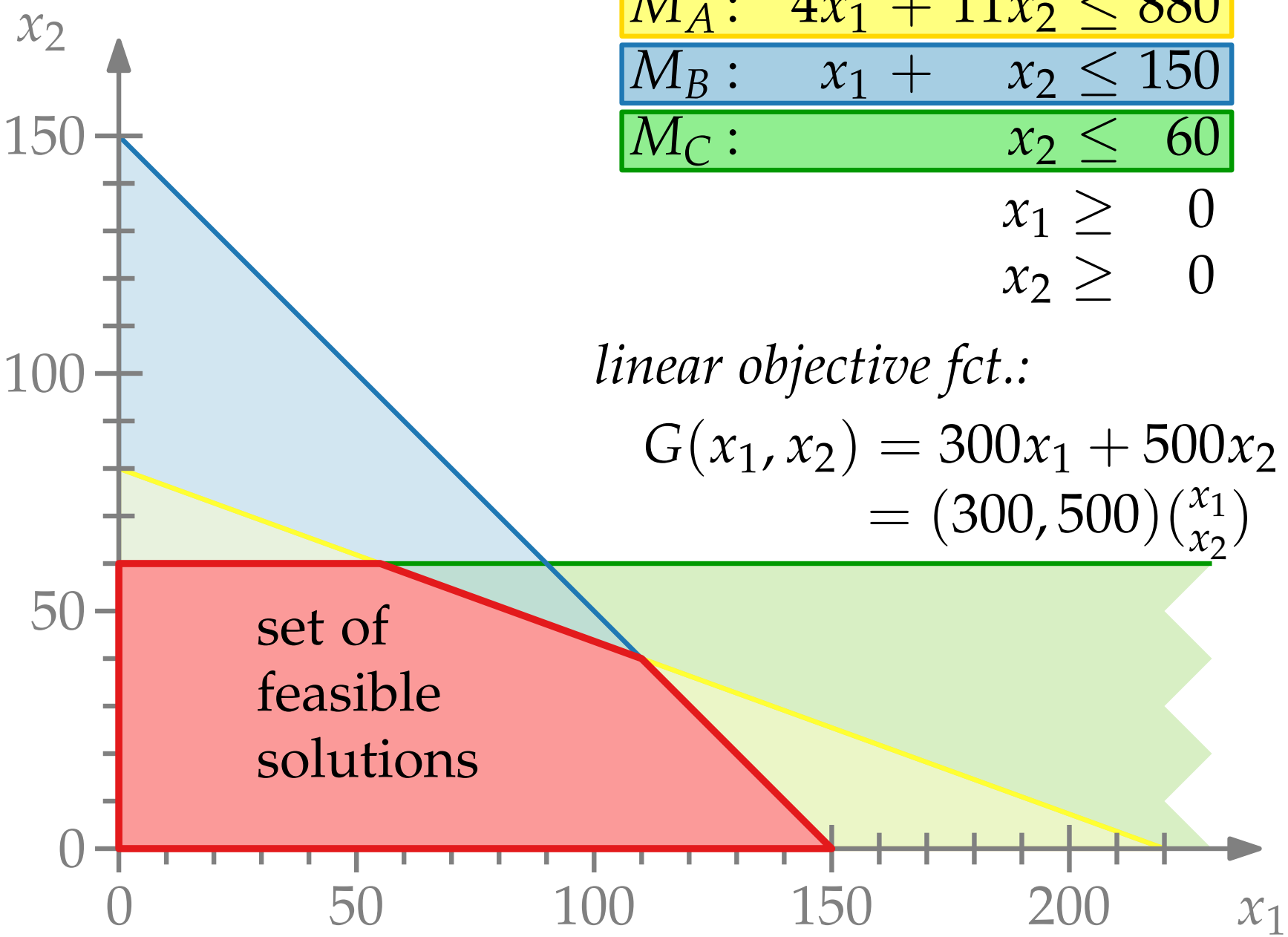
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2$$
$$= (300, 500)\binom{x_1}{x_2}$$



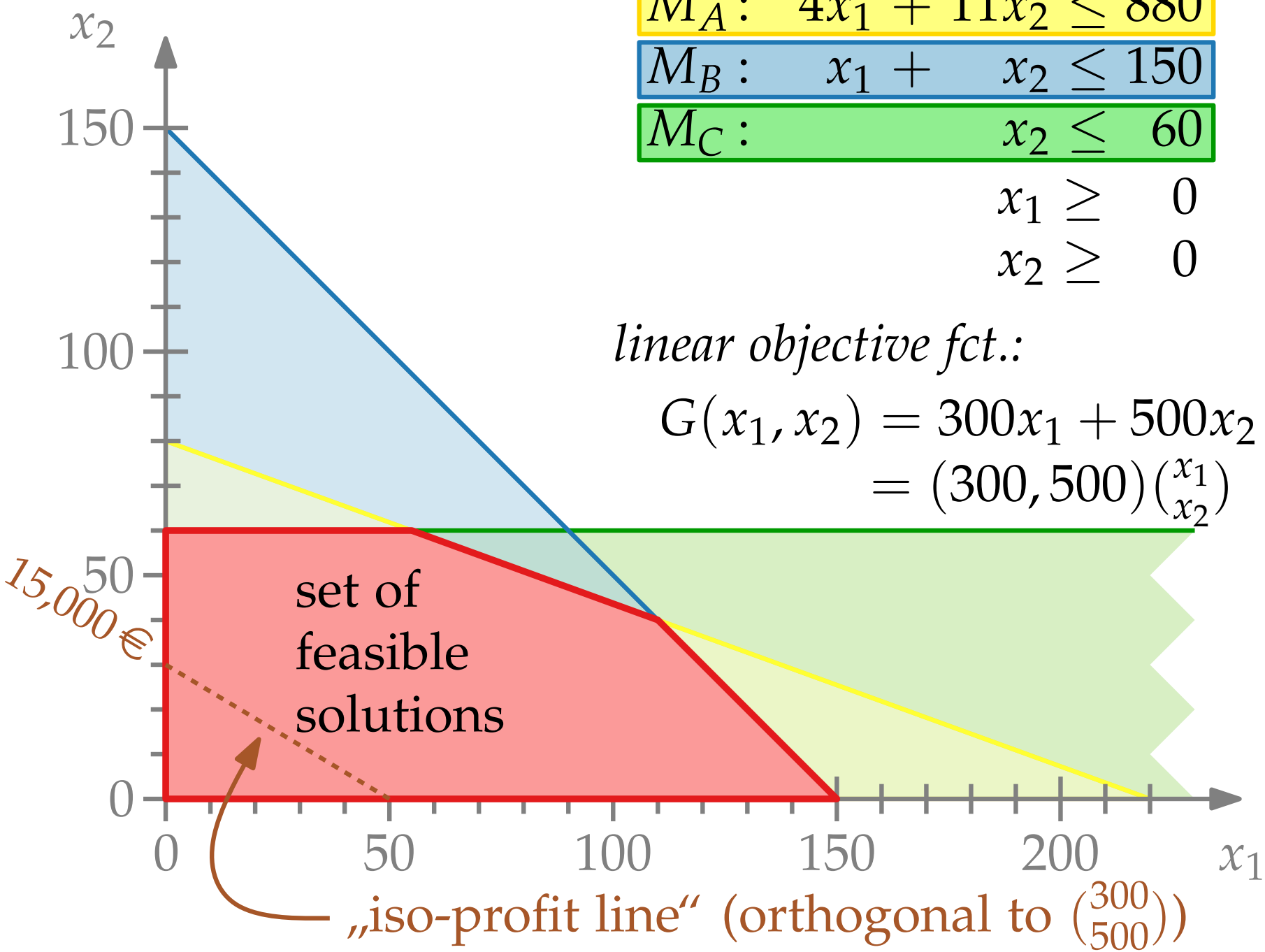
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



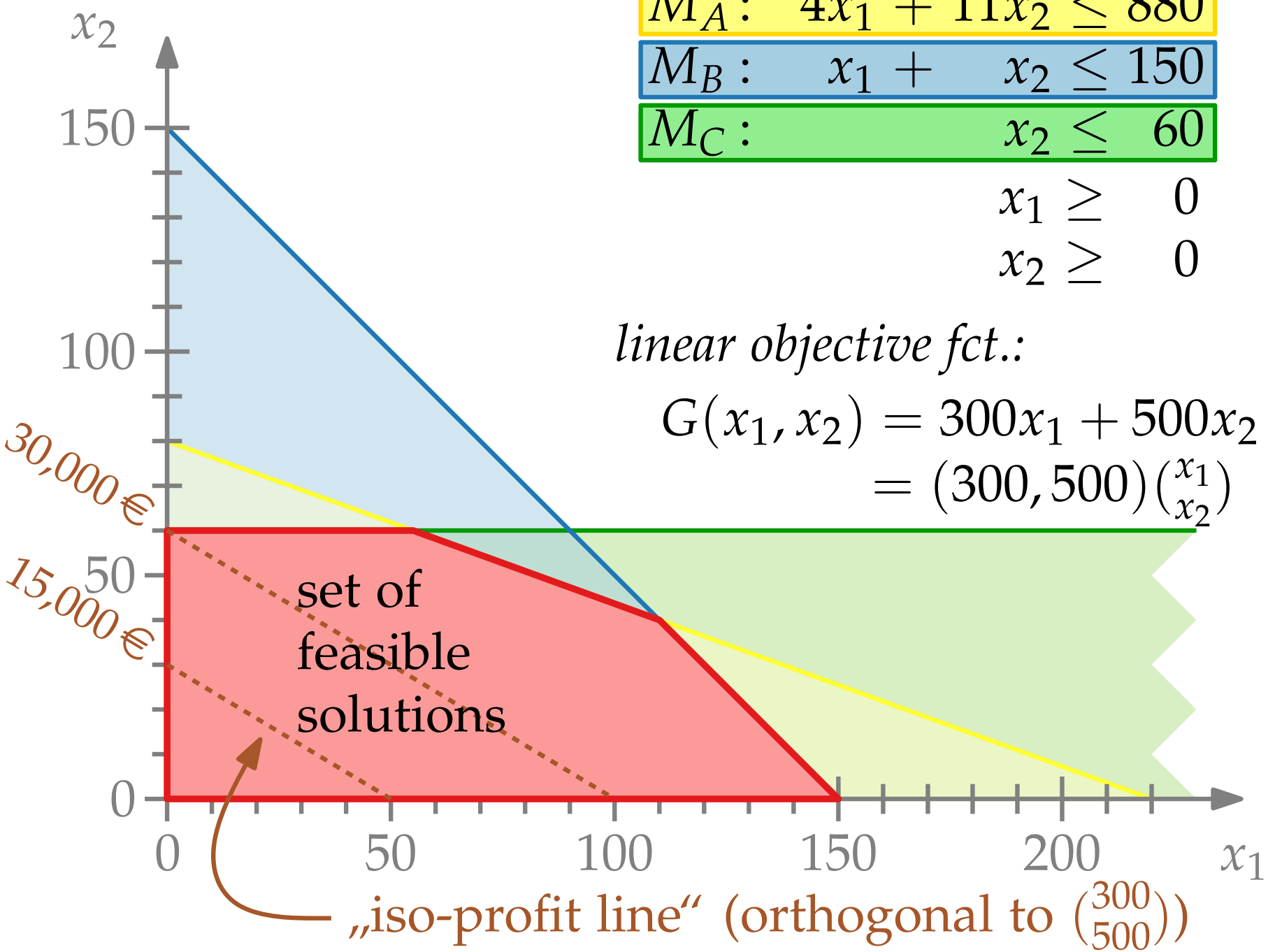
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



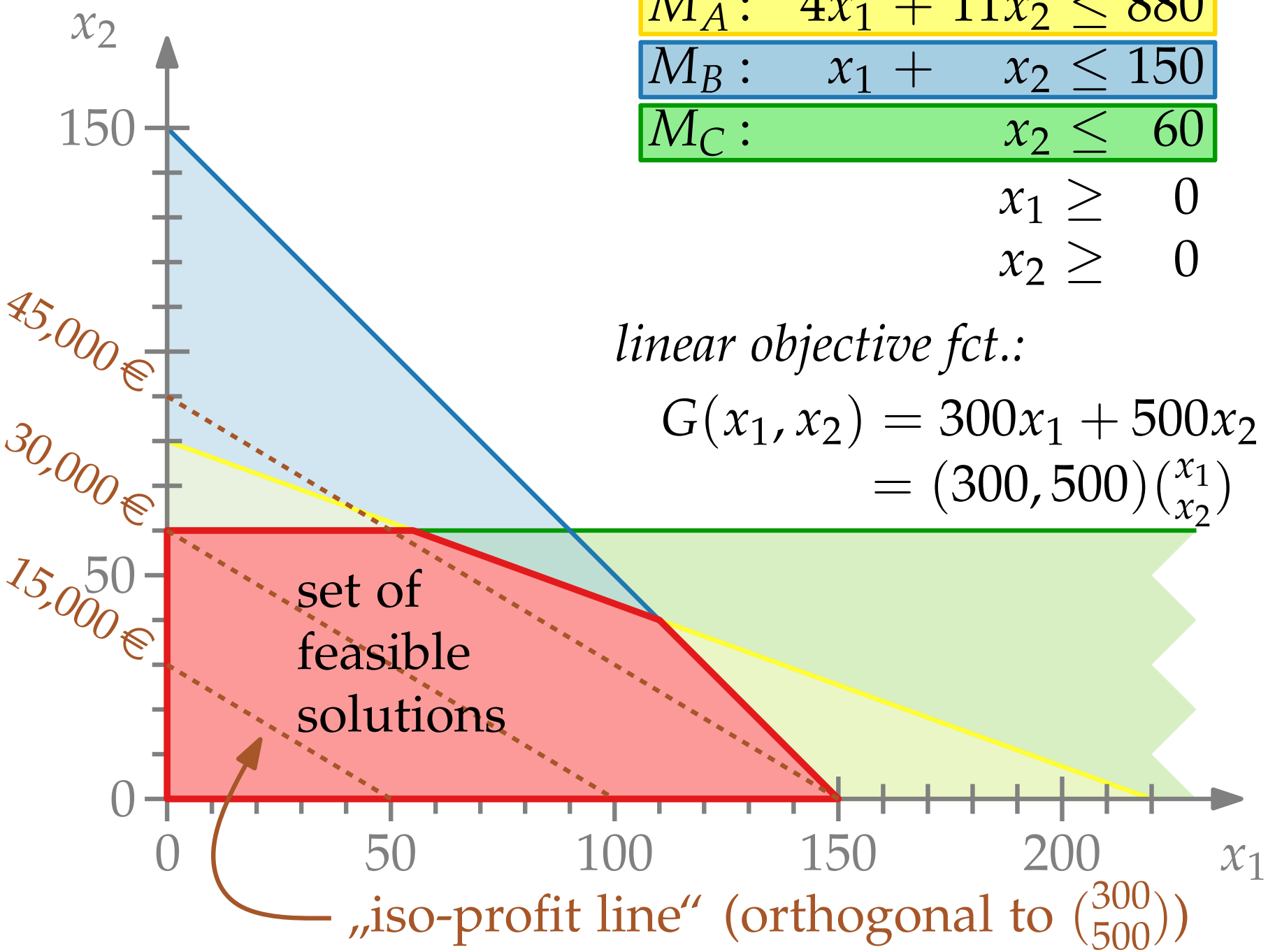
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



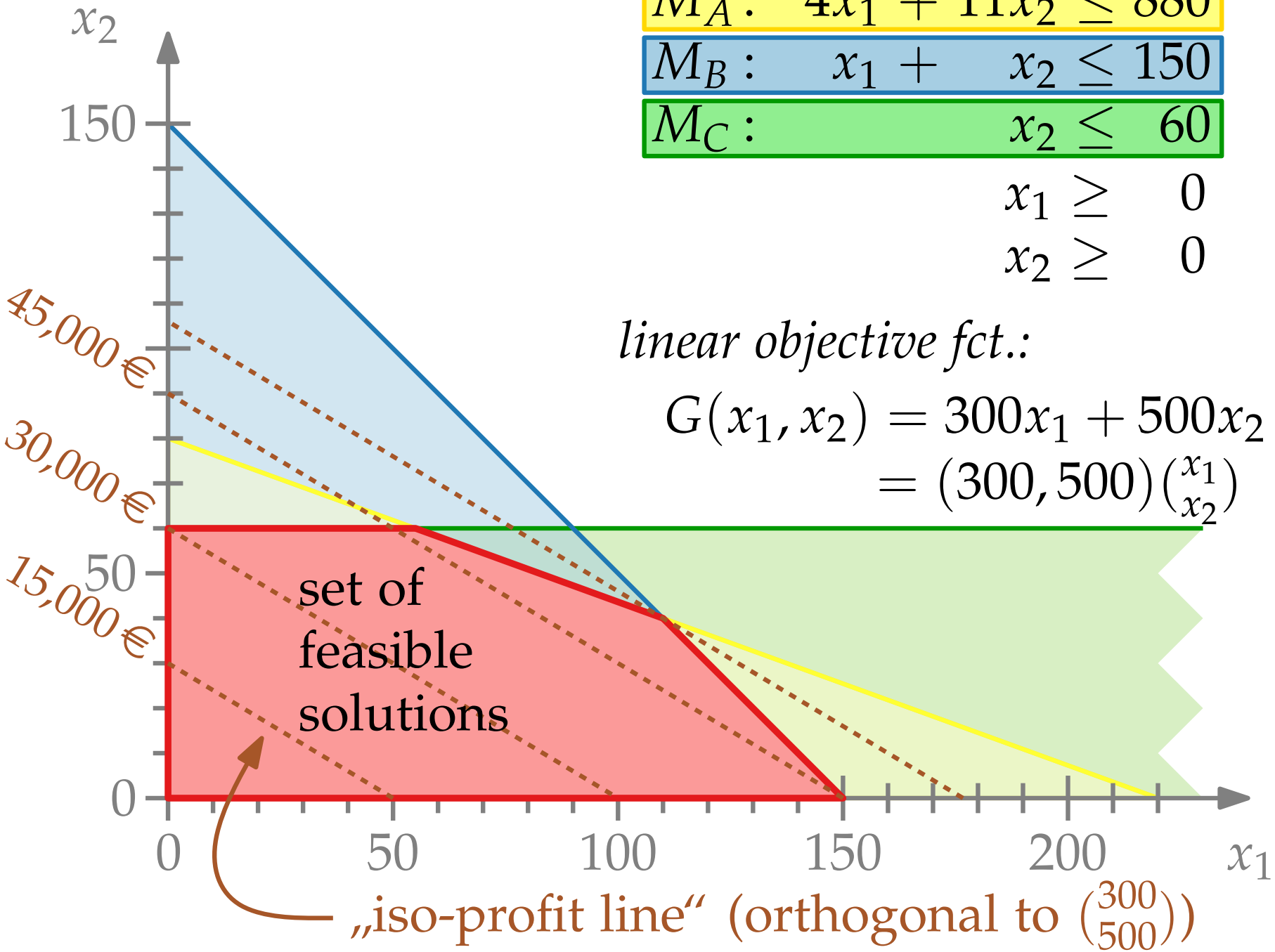
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



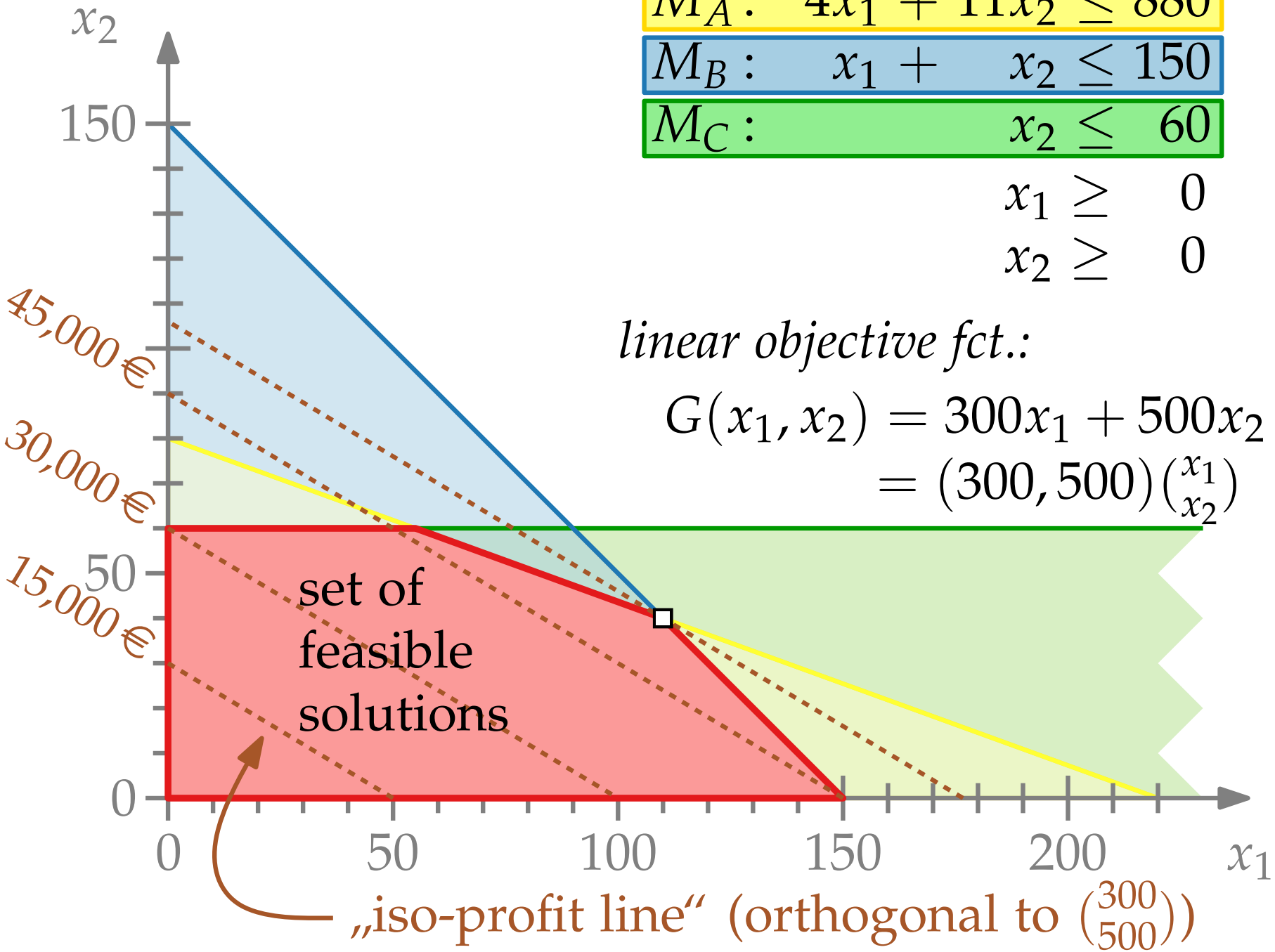
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



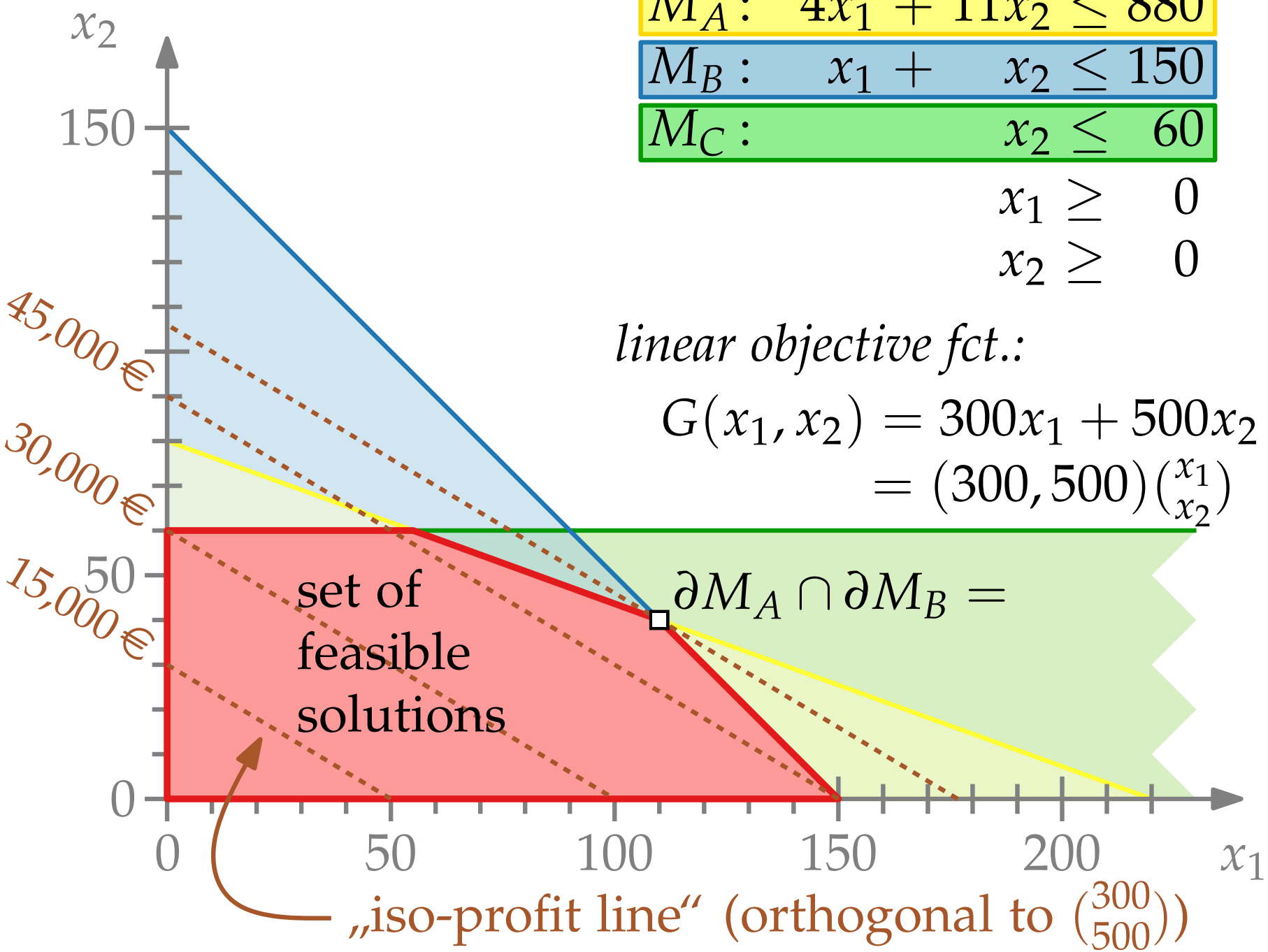
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



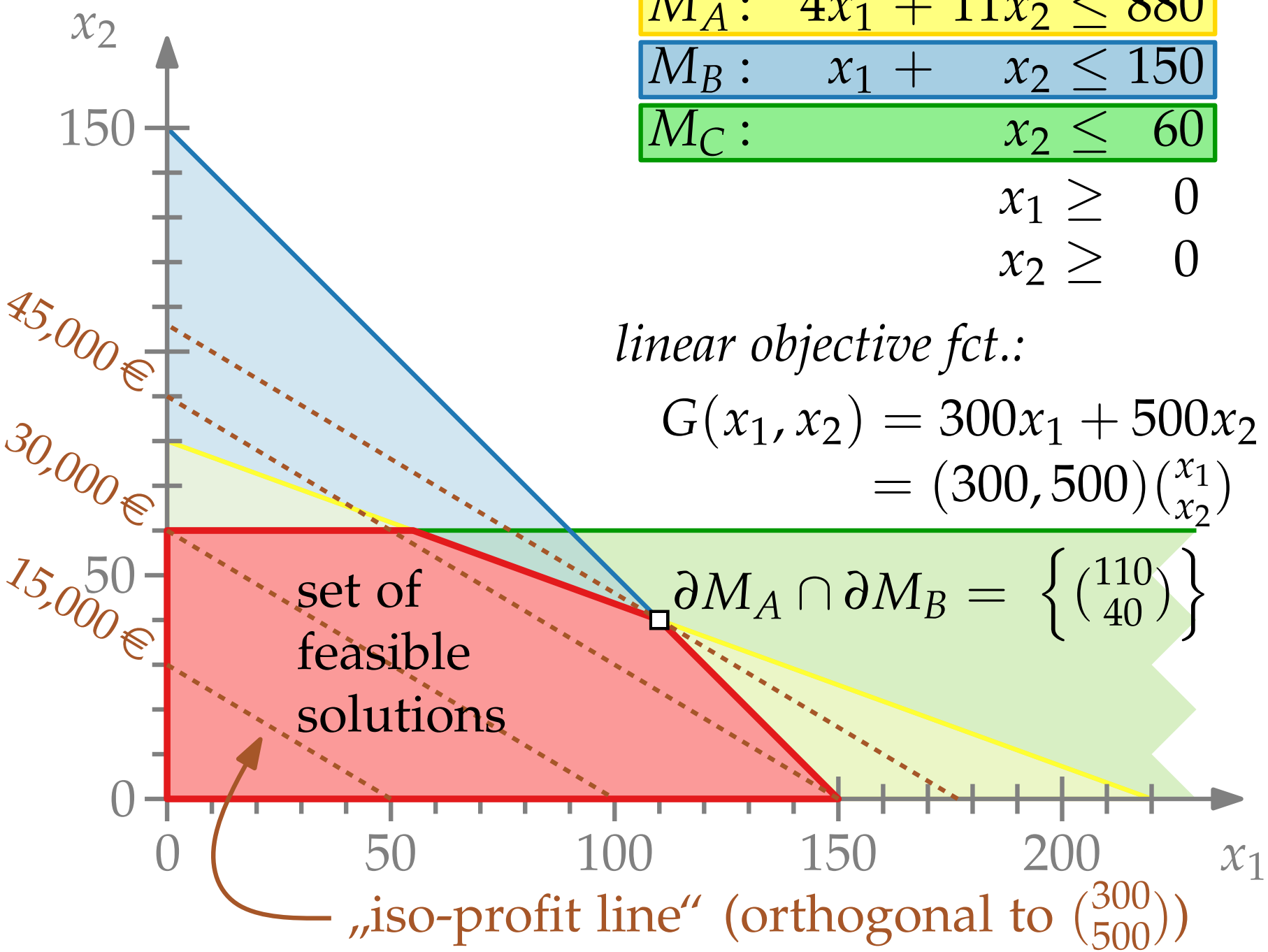
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



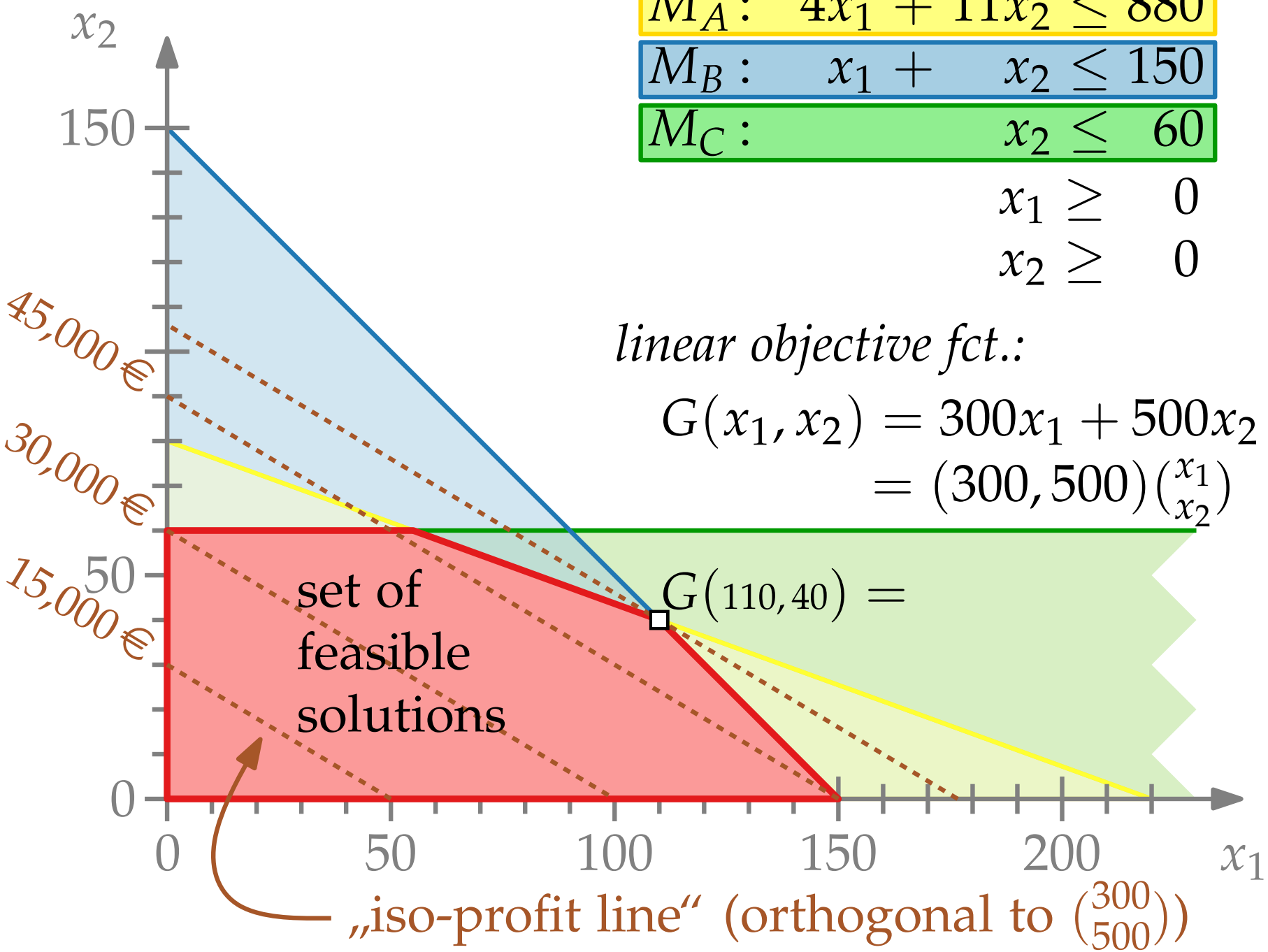
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



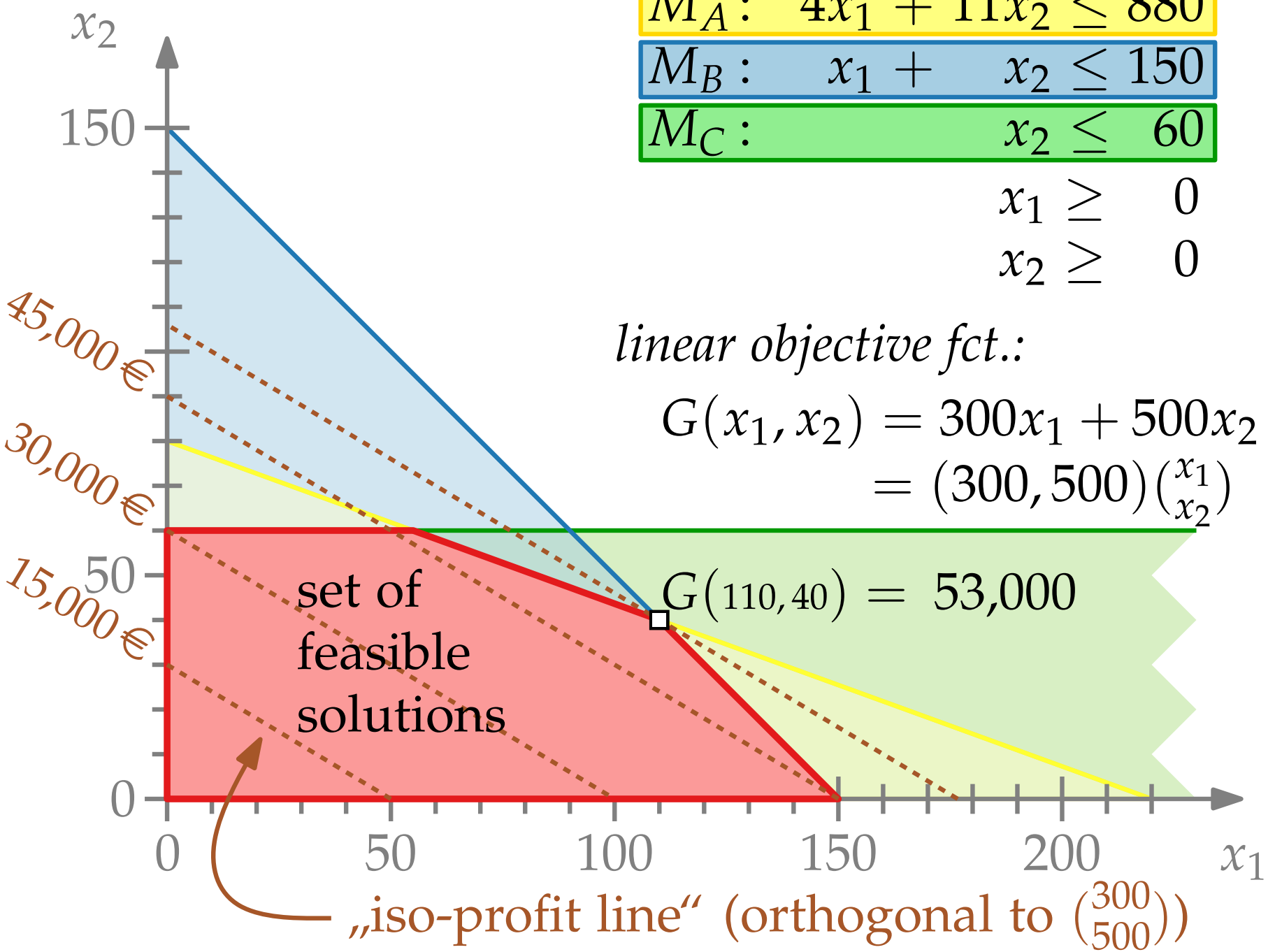
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



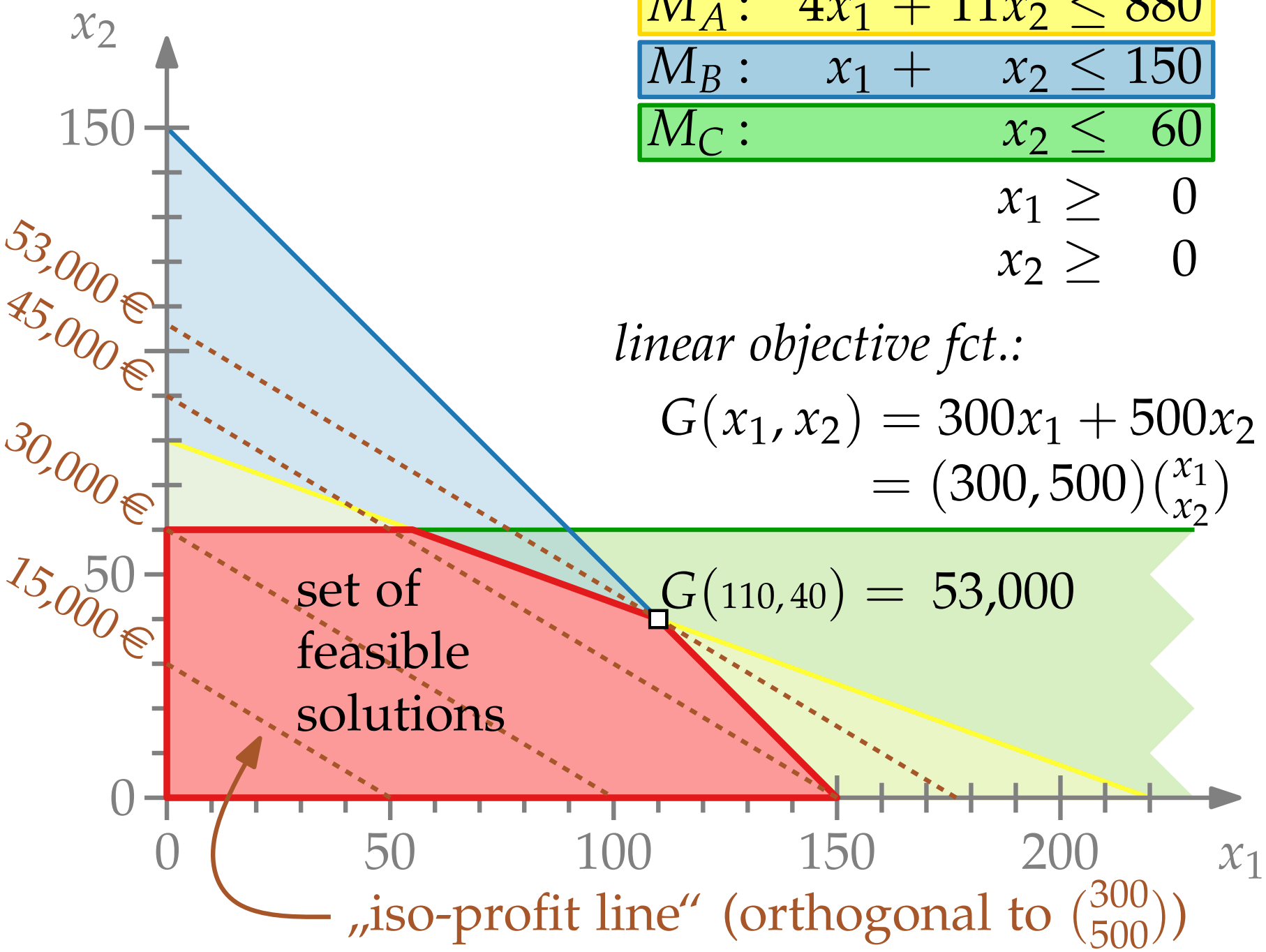
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



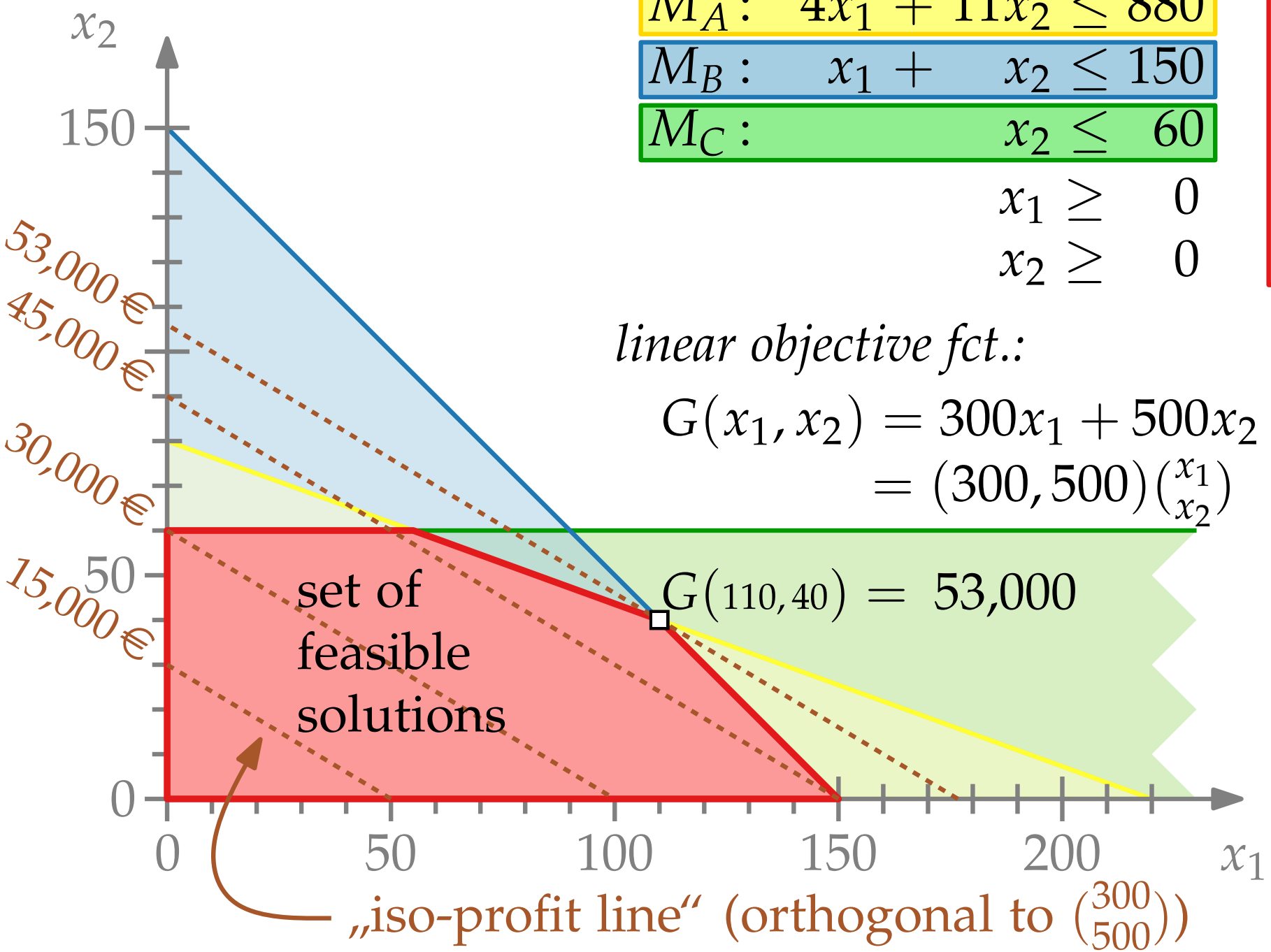
The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



The Answer

linear constraints:

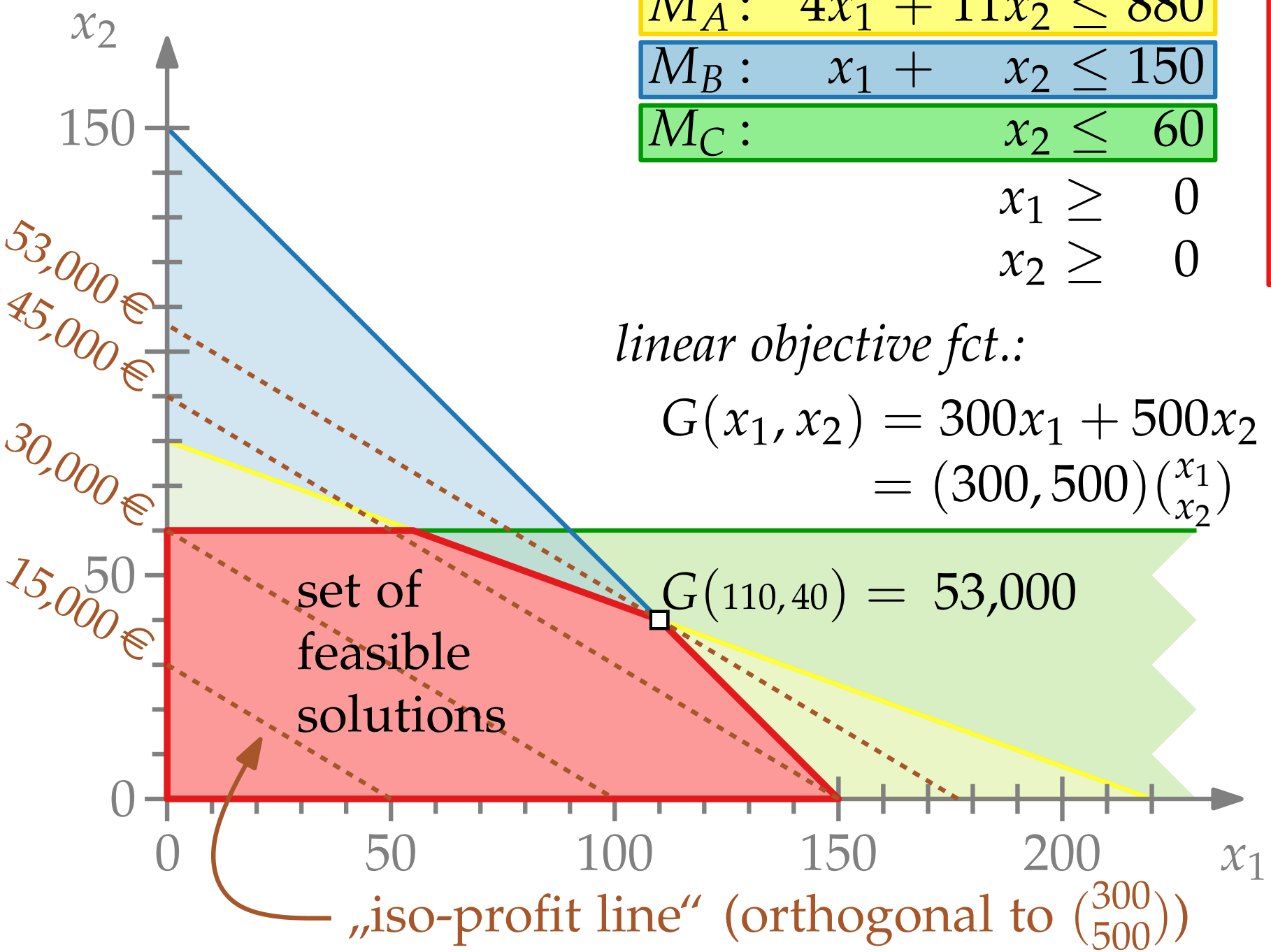
$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

$Ax \leq b$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$

$G(110, 40) = 53{,}000$



The Answer

linear constraints:

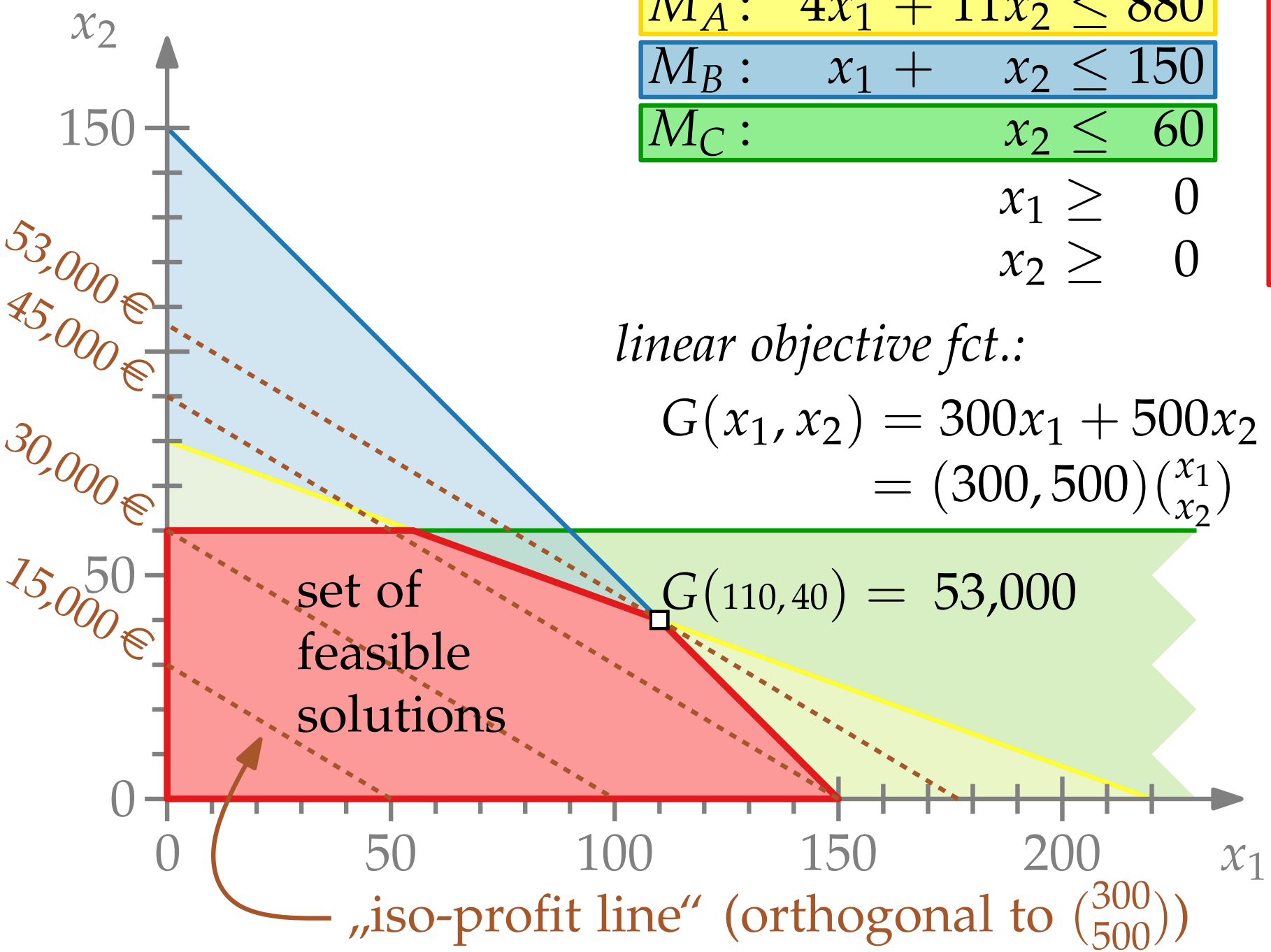
$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

$$Ax \leq b$$
$$x \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$

$$G(110, 40) = 53{,}000$$



The Answer

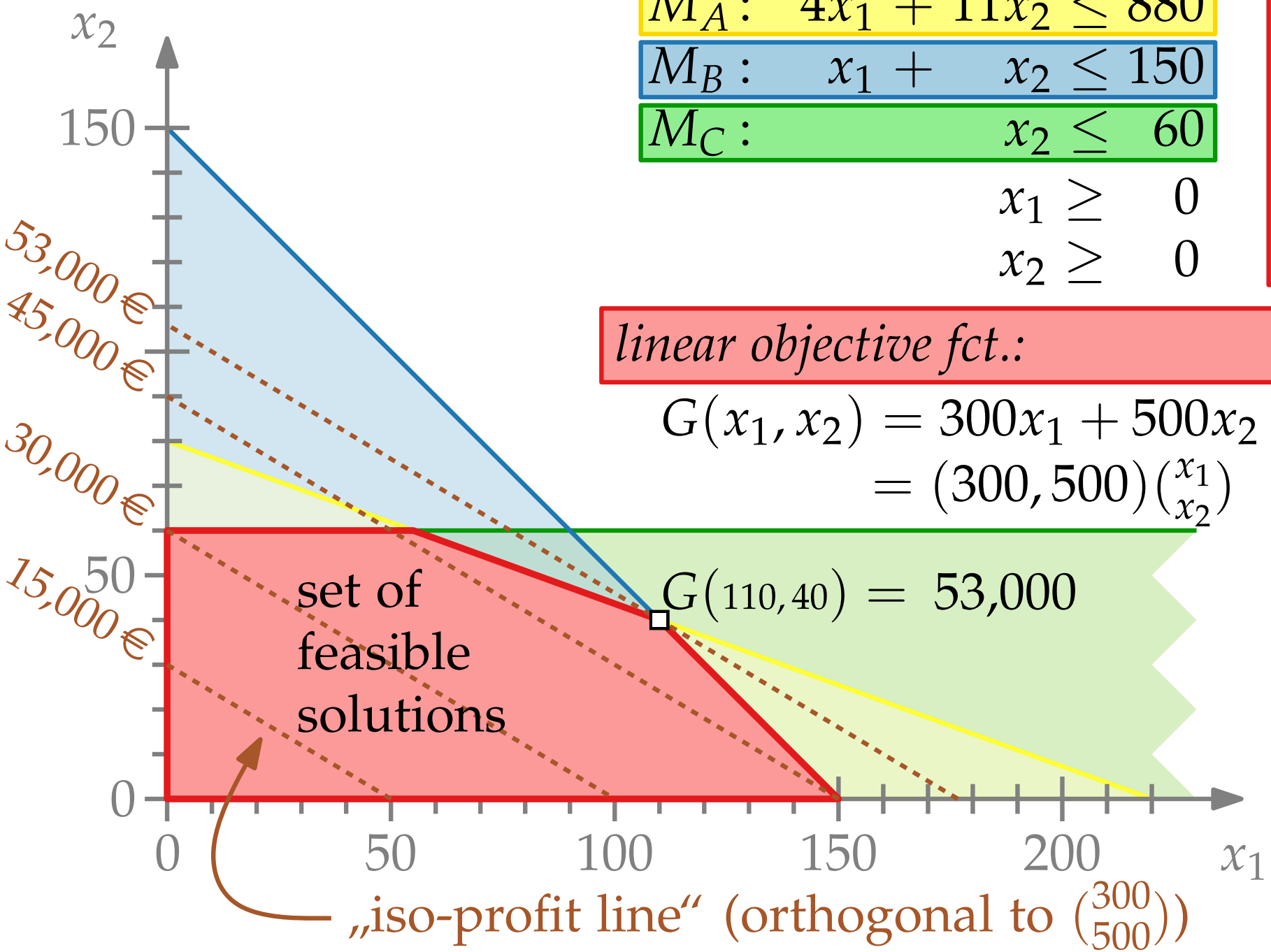
linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

$$Ax \leq b$$
$$x \geq 0$$

linear objective fct.:

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



The Answer

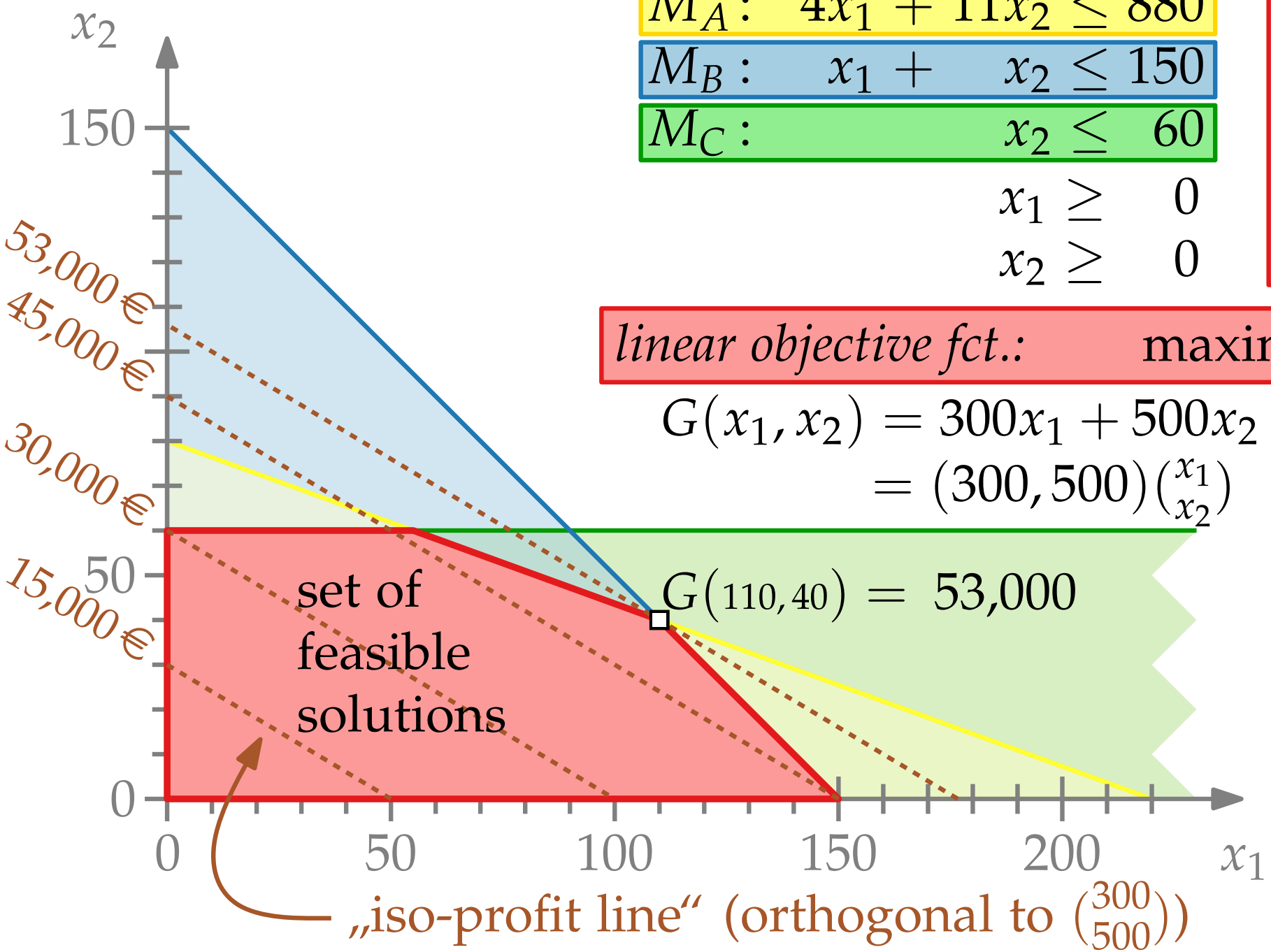
linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

$$Ax \leq b$$
$$x \geq 0$$

linear objective fct.: maximize $c^T x$

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



The Answer

linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

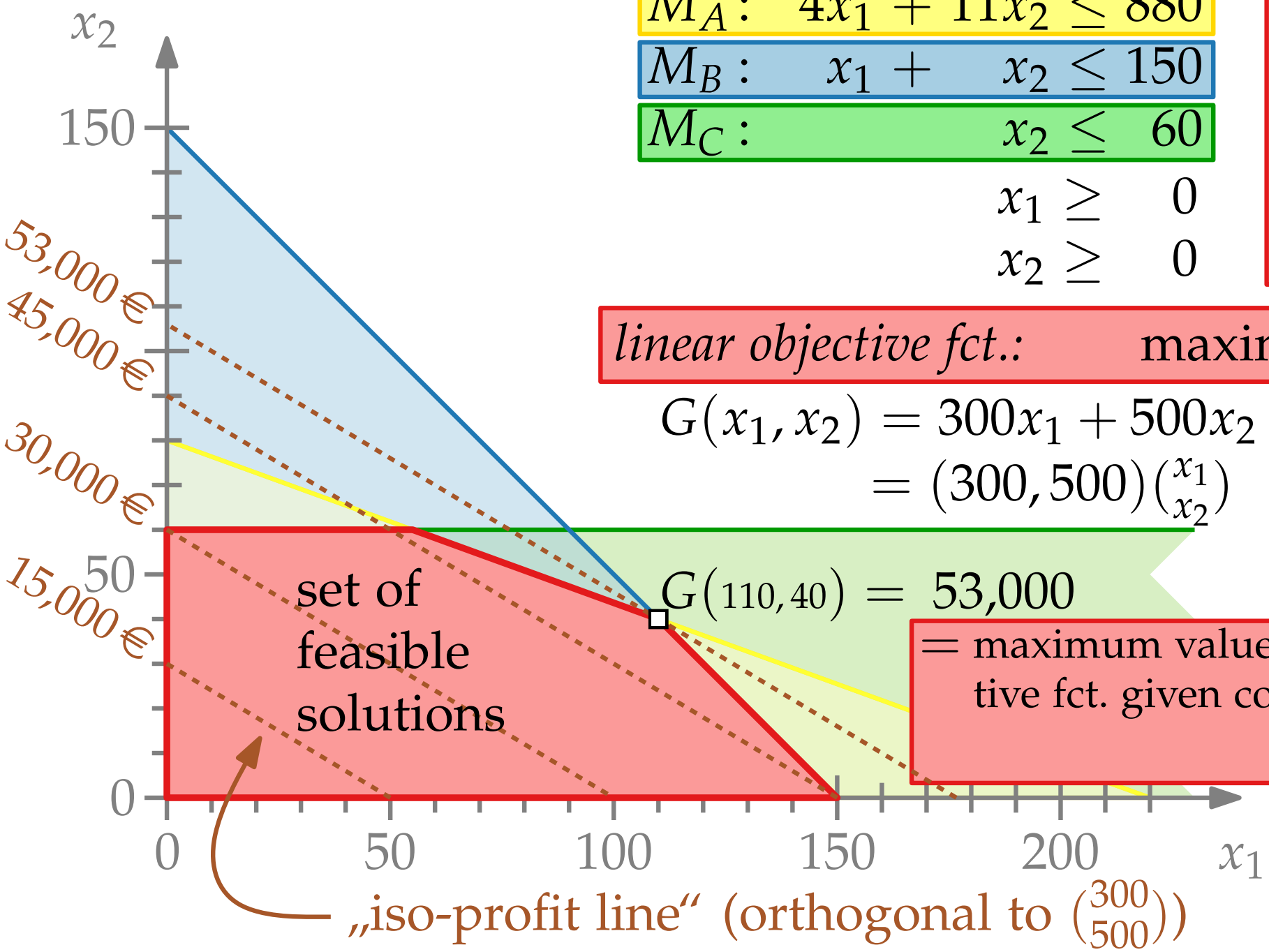
$$Ax \leq b$$
$$x \geq 0$$

linear objective fct.: maximize $c^T x$

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$

$$G(110, 40) = 53{,}000$$

= maximum value of objective fct. given constraints



The Answer

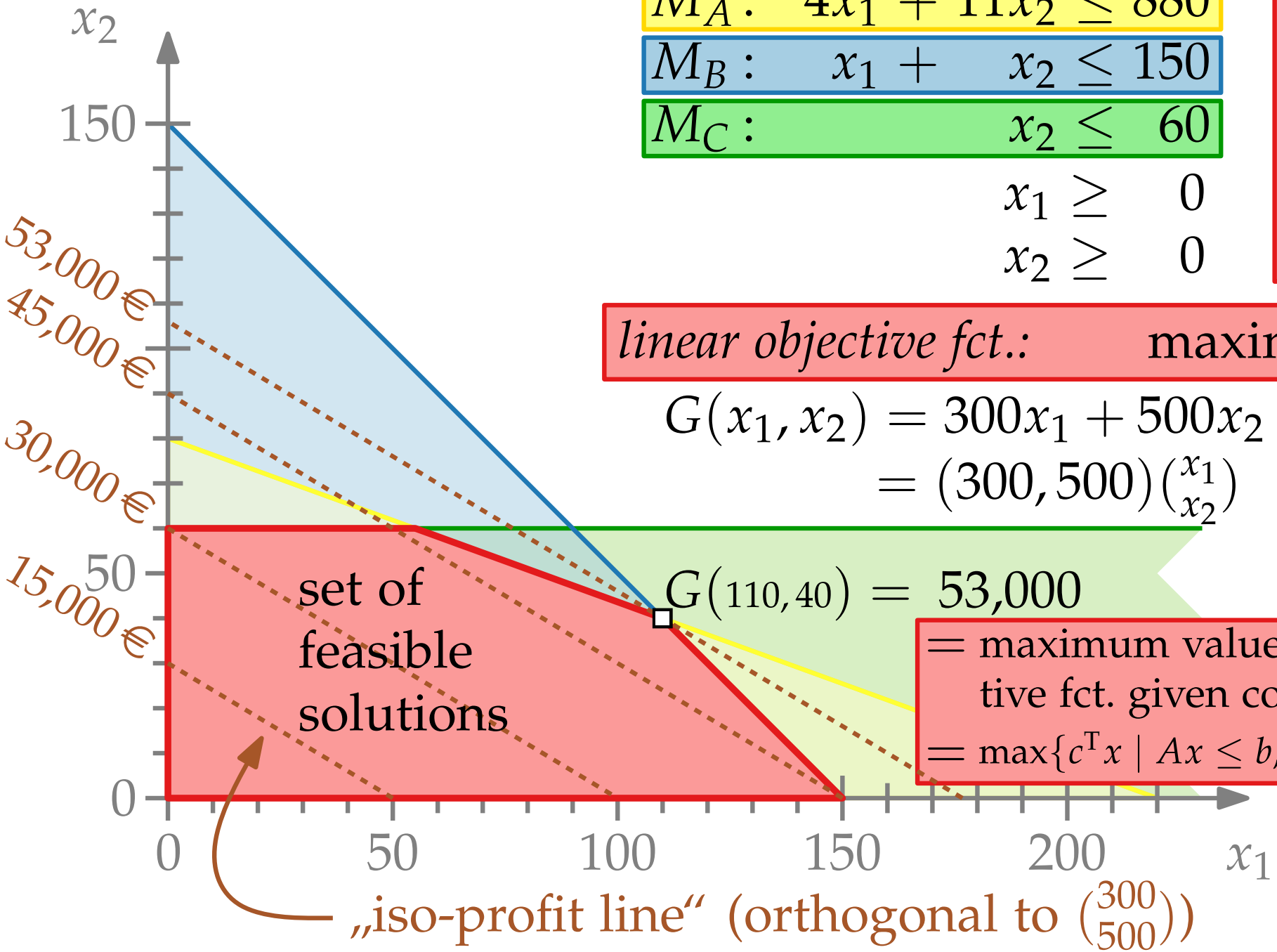
linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

$$Ax \leq b$$
$$x \geq 0$$

linear objective fct.: maximize $c^{\mathrm{T}}x$

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)\binom{x_1}{x_2}$$



= maximum value of objective fct. given constraints

$= \max\{c^{\mathrm{T}}x \mid Ax \leq b, x \geq 0\}$

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

Good for instances where *n and d* are large.

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

We consider $d = 2$.

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
["Book" application: casting]

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

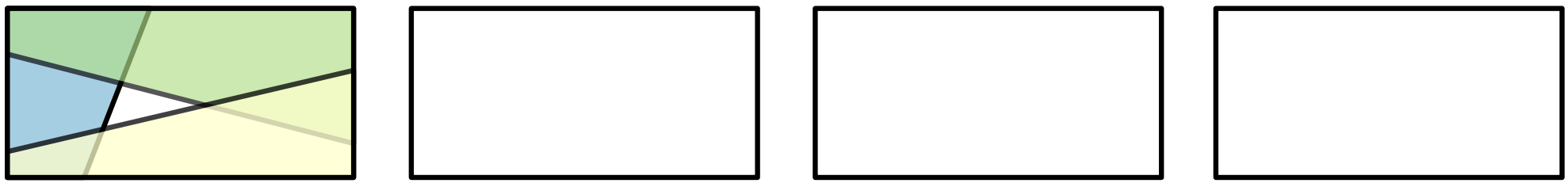
– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
["Book" application: casting]



Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

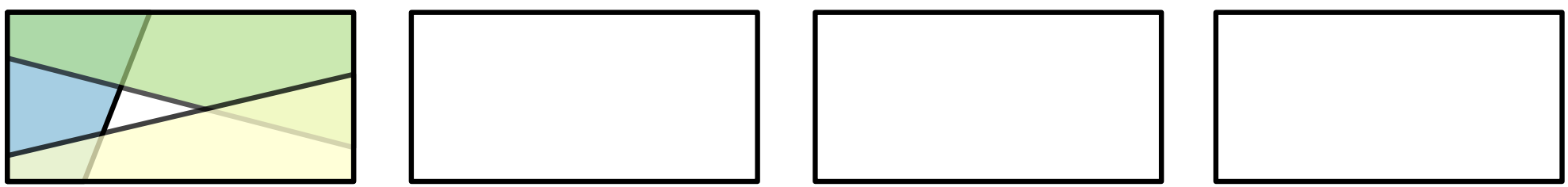
– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
["Book" application: casting]



$\bigcap H = \emptyset$

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

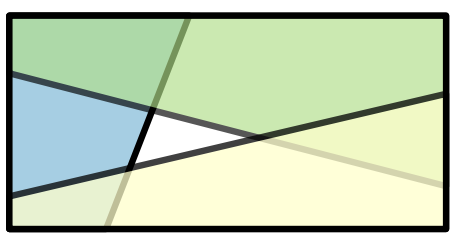
– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

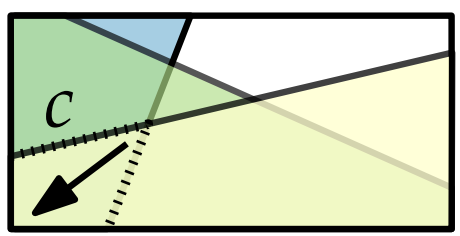
Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
["Book" application: casting]



$\bigcap H = \emptyset$



Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

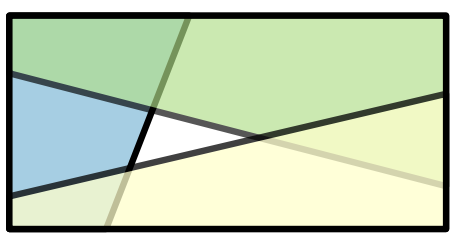
– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

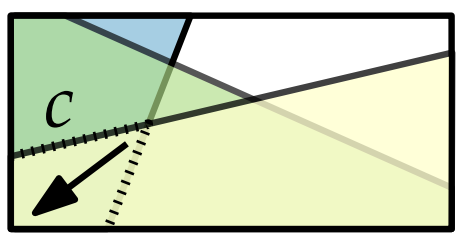
Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
["Book" application: casting]



$\bigcap H = \emptyset$



$\bigcap H$ unbnd. in dir. c

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

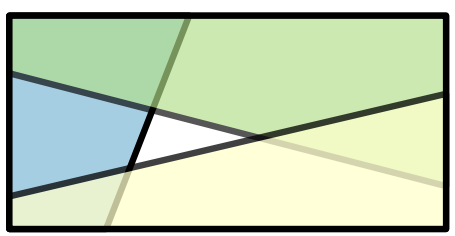
– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

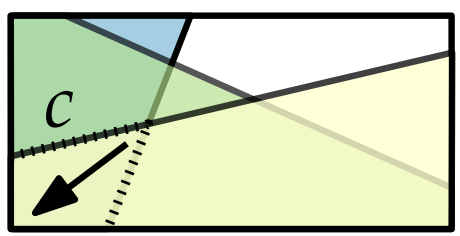
Good for instances where n *and* d are large.

We consider $d = 2$.

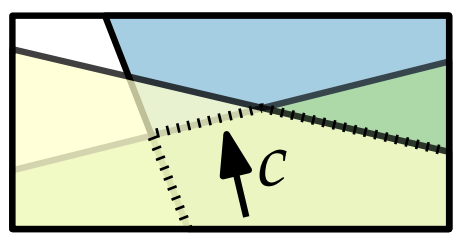
VERY important problem, e.g., in Operations Research.
["Book" application: casting]



$\bigcap H = \emptyset$



$\bigcap H$ unbnd. in dir. c



Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

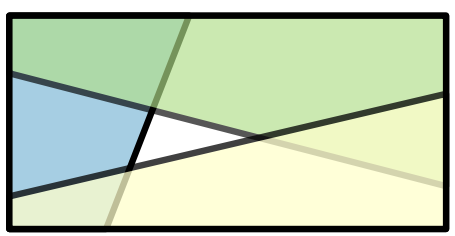
– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

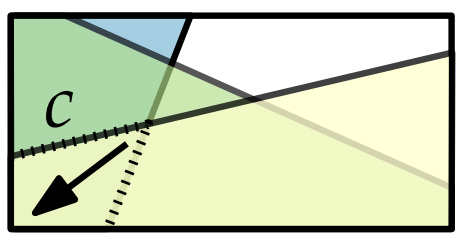
Good for instances where n *and* d are large.

We consider $d = 2$.

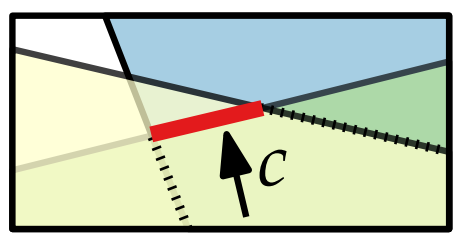
VERY important problem, e.g., in Operations Research.
["Book" application: casting]



$\bigcap H = \emptyset$



$\bigcap H$ unbnd. in dir. c



Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

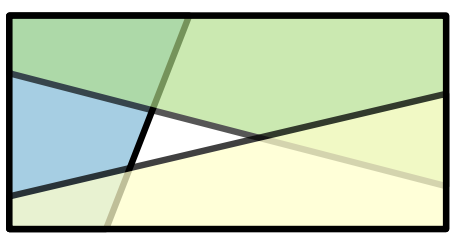
– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

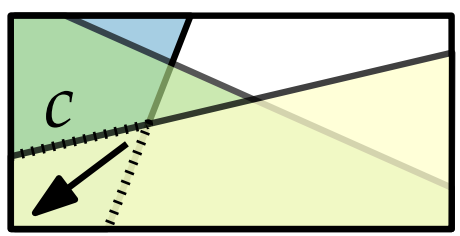
Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
["Book" application: casting]

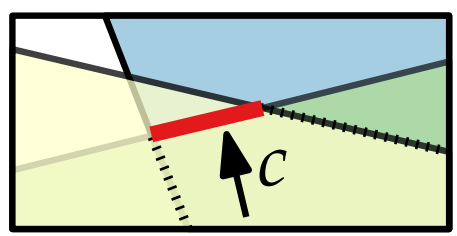


$\bigcap H = \emptyset$



$\bigcap H$ unbnd. in dir. c

$\bigcap H$ bounded.



Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

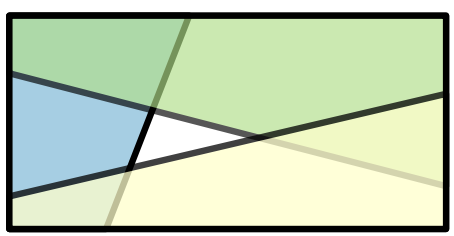
– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

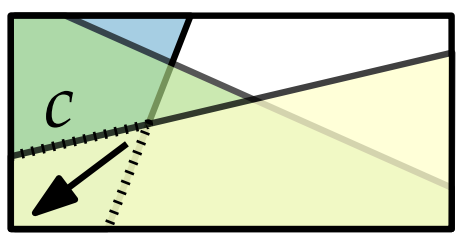
Good for instances where n *and* d are large.

We consider $d = 2$.

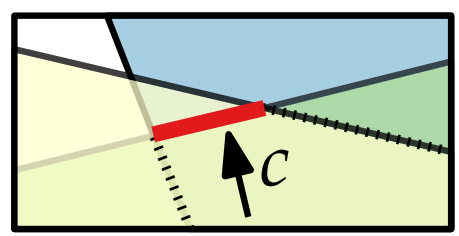
VERY important problem, e.g., in Operations Research.
["Book" application: casting]



$\bigcap H = \emptyset$



$\bigcap H$ unbnd. in dir. c



set of optima: segment

$\bigcap H$ bounded.

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

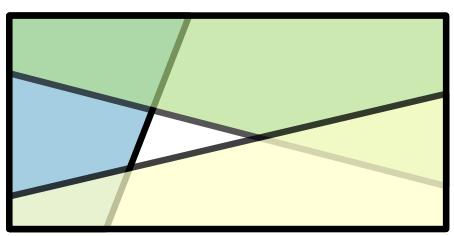
Many algorithms known, e.g.:

- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

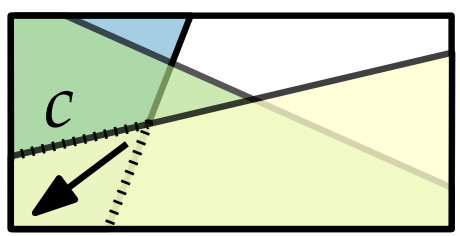
Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
["Book" application: casting]

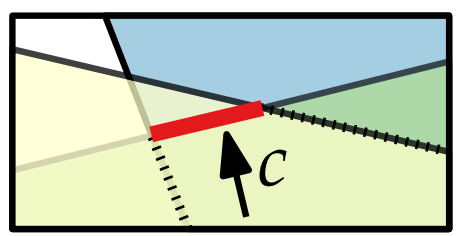


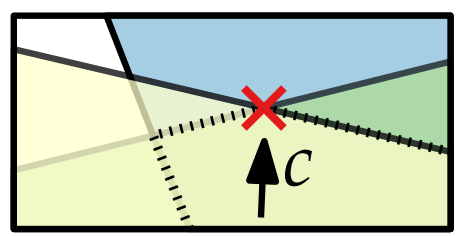
$\bigcap H = \emptyset$



$\bigcap H$ unbnd. in dir. c

$\bigcap H$ bounded.





set of optima: segment vs. point

First Approach

- compute $\bigcap H$ iteratively

First Approach

- compute $\bigcap H$ iteratively
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

First Approach

- compute $\bigcap H$ iteratively
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
foreach i from 2 to n **do**
  $C \leftarrow C \cap h_i$
return C

First Approach

- compute $\bigcap H$ iteratively
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
foreach i from 2 to n **do**
  $C \leftarrow C \cap h_i$
return C

Running time:

First Approach

- compute $\bigcap H$ iteratively
- walk $\partial(\bigcap H)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
foreach i from 2 to n **do**
  $C \leftarrow C \cap h_i$
return C

How??

Running time: $T_{\text{IH}}(n) = n \cdot$

First Approach

- compute $\bigcap H$ iteratively
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
foreach i from 2 to n **do**
  $C \leftarrow C \cap h_i$
return C

How??

$C :=$ chain of line segments $(s_1, \ldots, s_t)$

Running time: $T_{\text{IH}}(n) = n \cdot$

First Approach

- compute $\bigcap H$ iteratively
- walk $\partial(\bigcap H)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
foreach i from 2 to n **do**
  $C \leftarrow C \cap h_i$
return C

How??

$C :=$ chain of line segments $(s_1, \ldots, s_t)$

Walk around C to find $s_j, s_{j'} \in C$ intersecting h_i

Running time: $T_{\text{IH}}(n) = n \cdot$

First Approach

- compute $\bigcap H$ iteratively
- walk $\partial(\bigcap H)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
foreach i from 2 to n **do**
  $C \leftarrow C \cap h_i$
return C

How??

$C :=$ chain of line segments $(s_1, \ldots, s_t)$

Walk around C to find $s_j, s_{j'} \in C$ intersecting h_i

Update C

Running time: $T_{\text{IH}}(n) = n \cdot$

First Approach

- compute $\bigcap H$ iteratively
- walk $\partial(\bigcap H)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
foreach i from 2 to n **do**
  $C \leftarrow C \cap h_i$
return C

How??

$C :=$ chain of line segments $(s_1, \ldots, s_t)$

Walk around C to find $s_j, s_{j'} \in C$ intersecting h_i

Update C

Running time: $T_{\text{IH}}(n) = n \cdot O(n)$

First Approach

- compute $\bigcap H$ iteratively
- walk $\partial(\bigcap H)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
foreach i from 2 to n **do**
  $C \leftarrow C \cap h_i$
return C

How??

$C :=$ chain of line segments $(s_1, \ldots, s_t)$

Walk around C to find $s_j, s_{j'} \in C$ intersecting h_i

Update C

Running time: $T_{\text{IH}}(n) = n \cdot O(n)$

Total Time: $O(n^2)$:(

First Approach

- compute $\bigcap H$ iteratively
- walk $\partial(\bigcap H)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
foreach i from 2 to n **do**
  $C \leftarrow C \cap h_i$
return C

How??

$C :=$ chain of line segments $(s_1, \ldots, s_t)$

Walk around C to find $s_j, s_{j'} \in C$ intersecting h_i

Update C

Running time: $T_{\text{IH}}(n) = n \cdot O(n)$

Total Time: $O(n^2)$:(

Exercise: Compute $C \cap h_i$ faster.

Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

if $|H| = 1$ **then**

  $C \leftarrow h$, where $\{h\} = H$

else

return C

Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

if $|H| = 1$ **then**
  $C \leftarrow h$, where $\{h\} = H$
else
  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)
  $C_2 \leftarrow$ IntersectHalfplanes(H_2)
  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)
return C

Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

if $|H| = 1$ **then**
  $C \leftarrow h$, where $\{h\} = H$
else
  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)
  $C_2 \leftarrow$ IntersectHalfplanes(H_2)
  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)
return C

Running time:

Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

if $|H| = 1$ **then**
  $C \leftarrow h$, where $\{h\} = H$
else
  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)
  $C_2 \leftarrow$ IntersectHalfplanes(H_2)
  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)
return C

Running time: $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

```
IntersectHalfplanes(H)
  if |H| = 1 then
    C ← h, where {h} = H
  else
    split H into sets H1 and H2 with |H1|, |H2| ≈ |H|/2
    C1 ← IntersectHalfplanes(H1)
    C2 ← IntersectHalfplanes(H2)
    C ← IntersectConvexRegions(C1, C2)
  return C
```

How??

Running time: $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

```
IntersectHalfplanes(H)
  if |H| = 1 then
    C ← h, where {h} = H
  else
    split H into sets H1 and H2 with |H1|, |H2| ≈ |H|/2
    C1 ← IntersectHalfplanes(H1)
    C2 ← IntersectHalfplanes(H2)
    C ← IntersectConvexRegions(C1, C2)
  return C
```

How complex can the new region be?

How??

Running time: $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial\left(\bigcap H\right)$, find vertex x w/ cx maximum, $O(n)$ time

IntersectHalfplanes(H)

if $|H| = 1$ **then**

  $C \leftarrow h$, where $\{h\} = H$

else

  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$

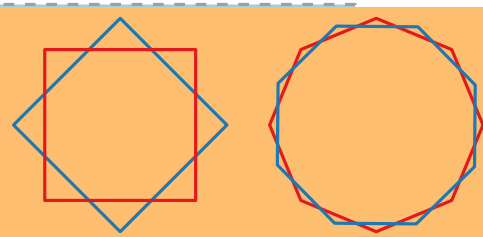
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)

  $C_2 \leftarrow$ IntersectHalfplanes(H_2)

  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)

return C

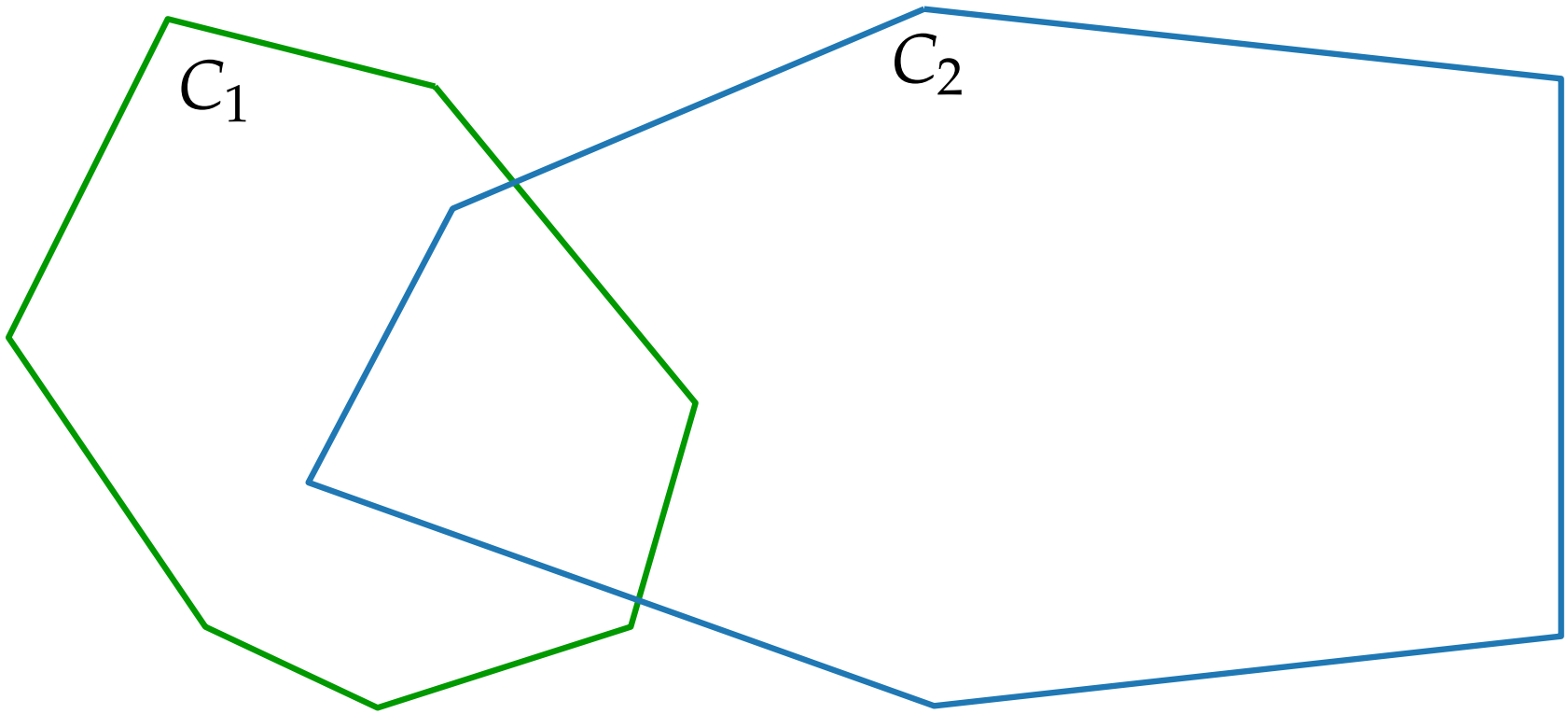
How complex can the new region be?



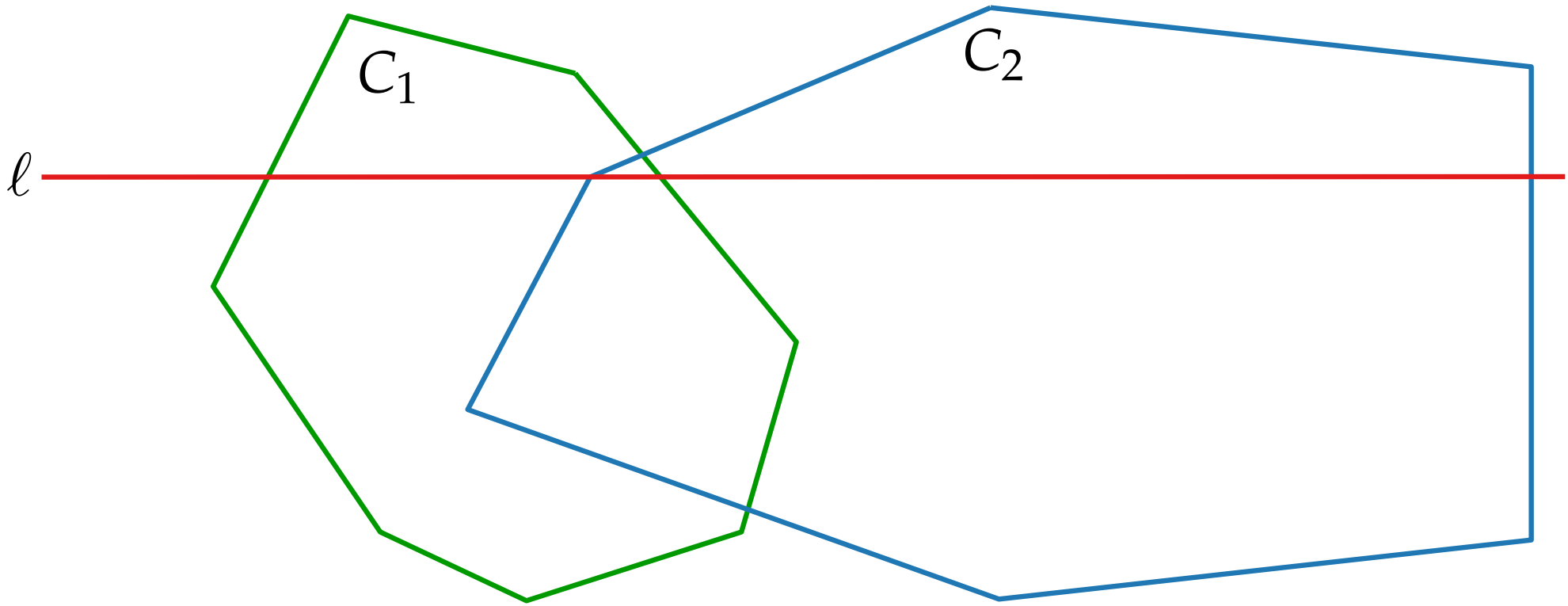
How??

Running time: $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

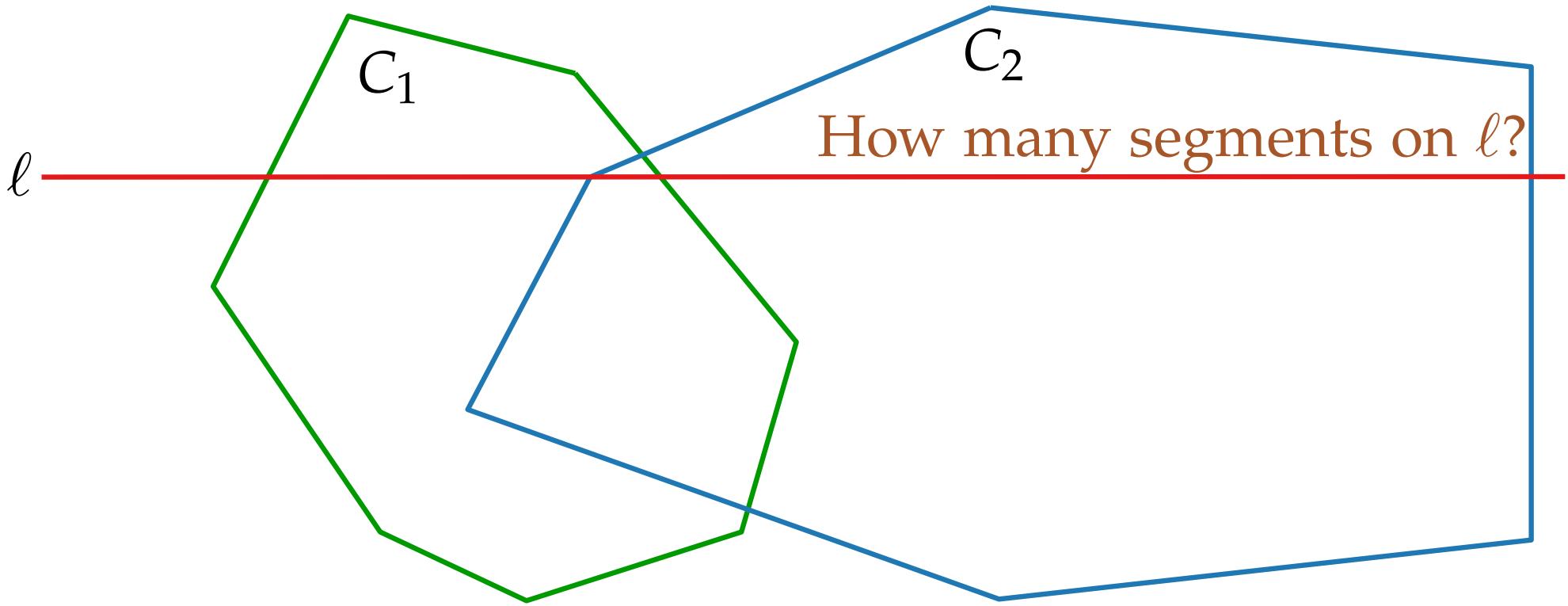
Intersecting Convex Regions



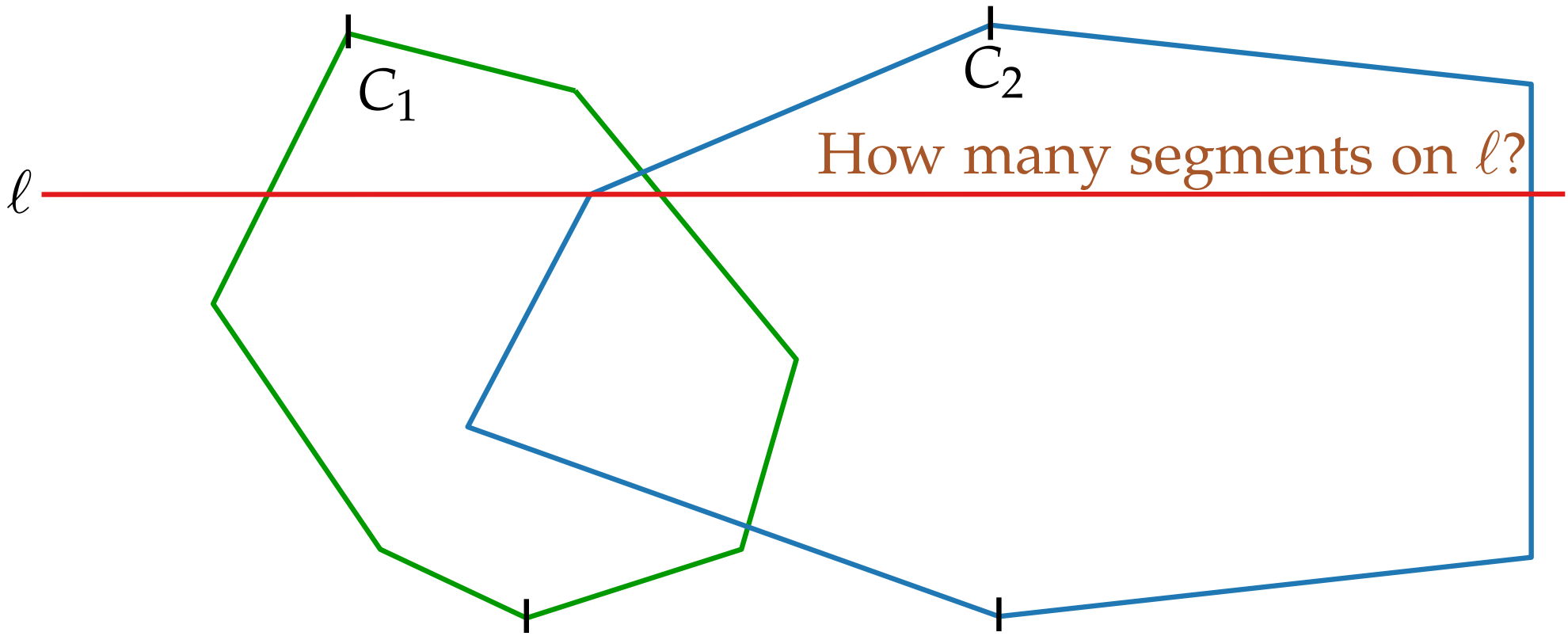
Intersecting Convex Regions



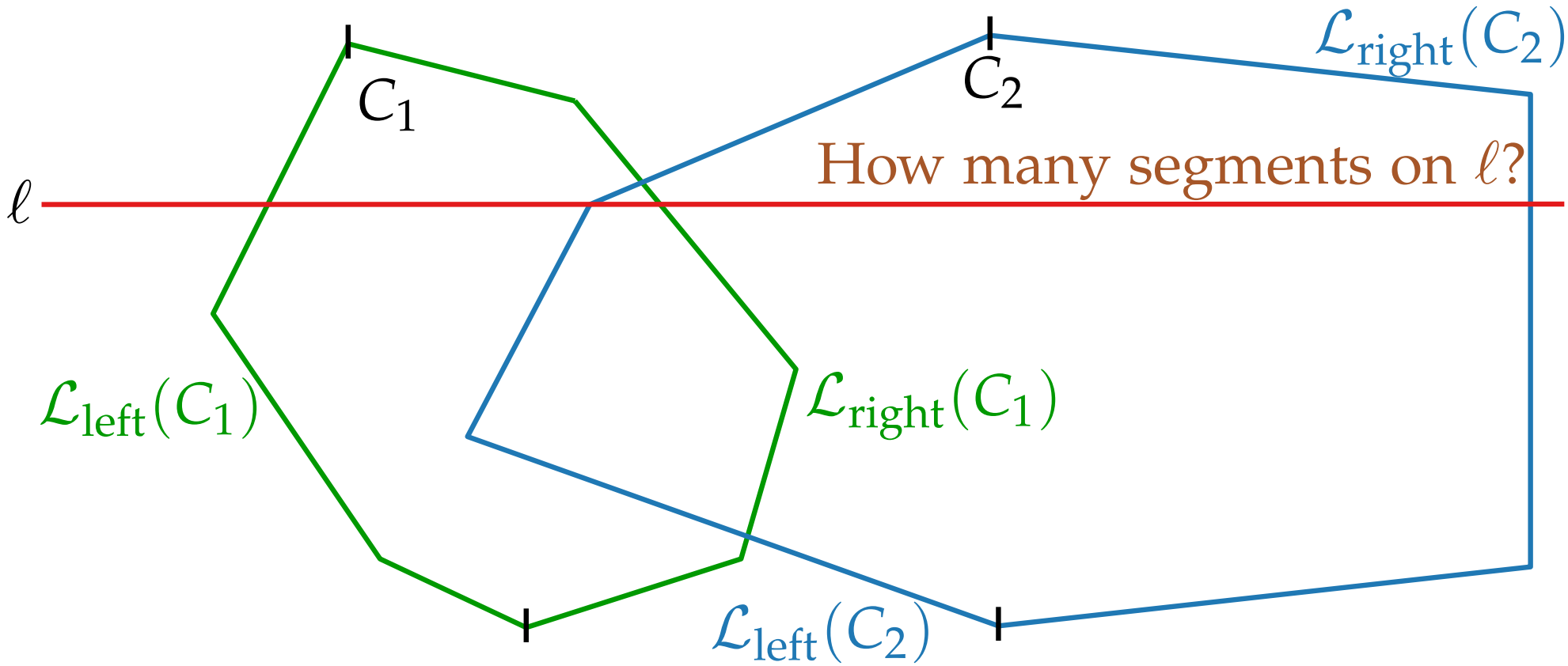
Intersecting Convex Regions



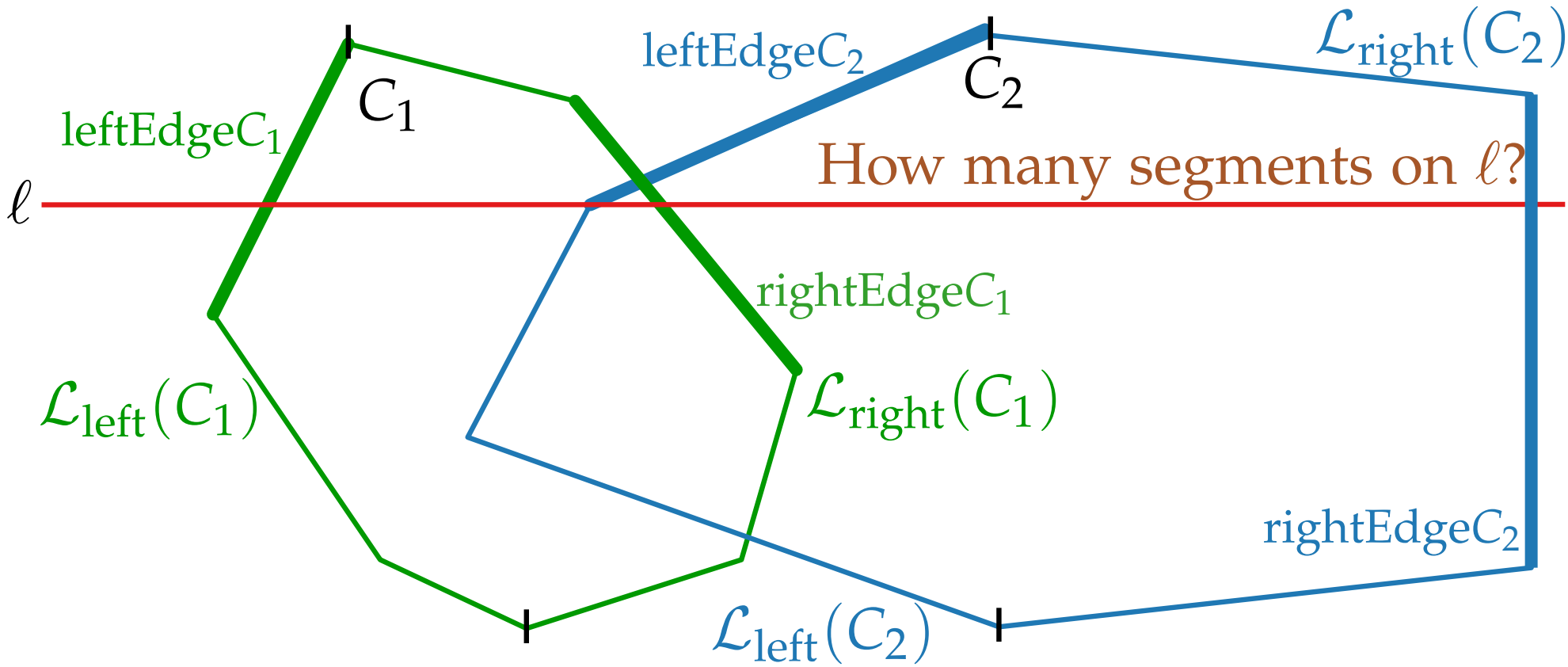
Intersecting Convex Regions



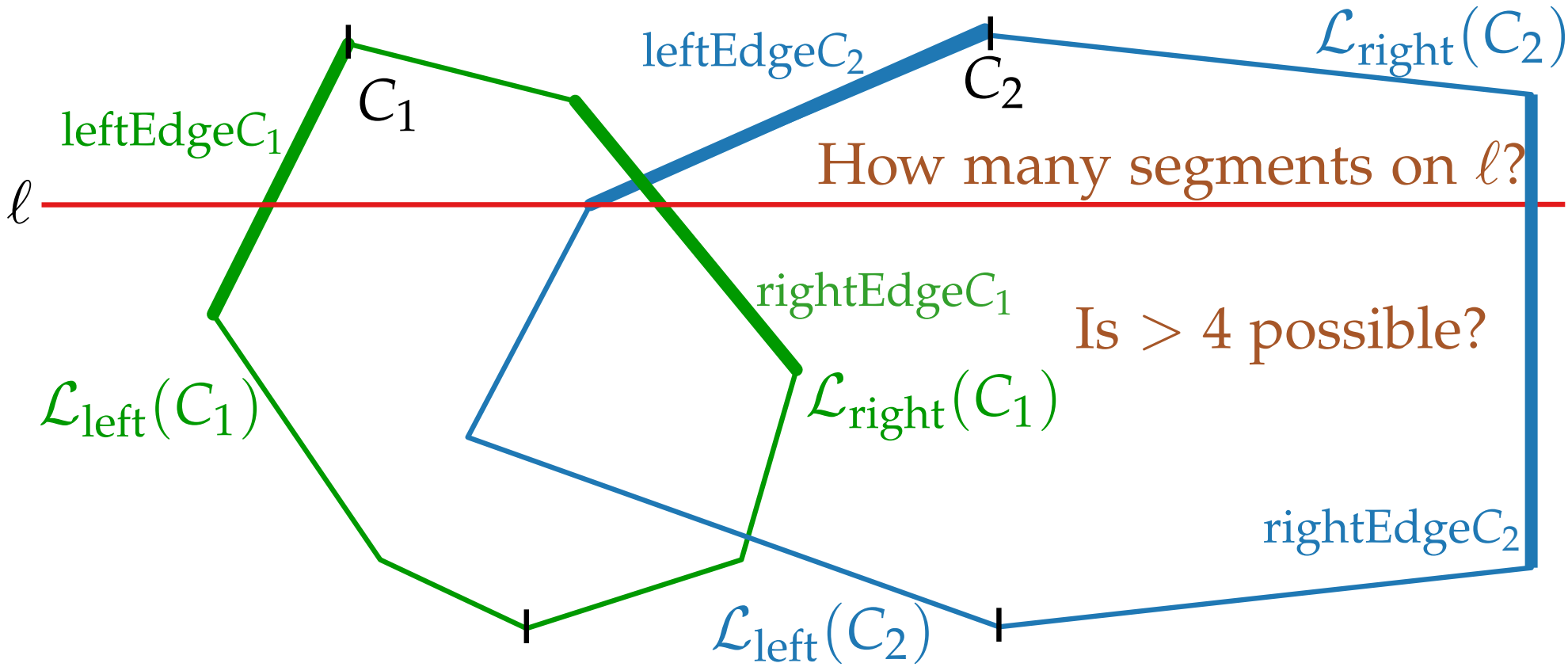
Intersecting Convex Regions



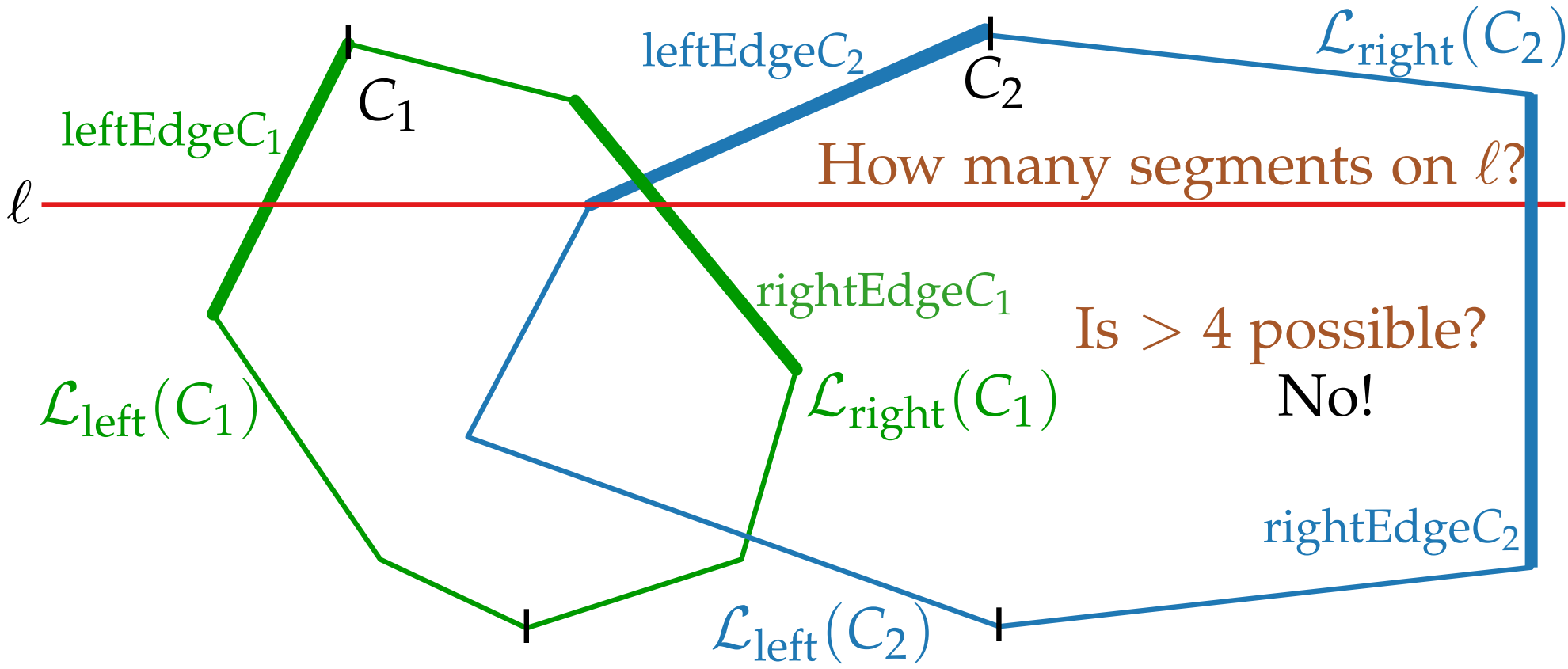
Intersecting Convex Regions



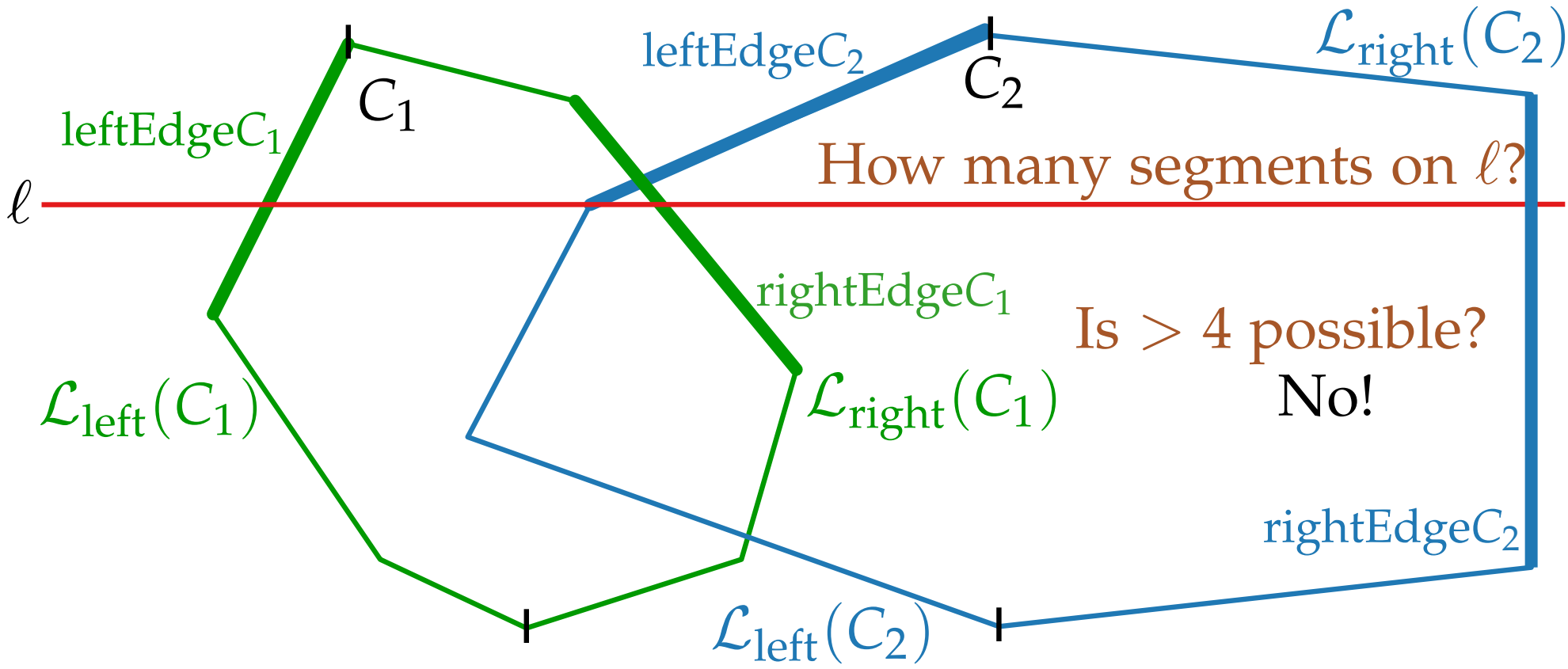
Intersecting Convex Regions



Intersecting Convex Regions

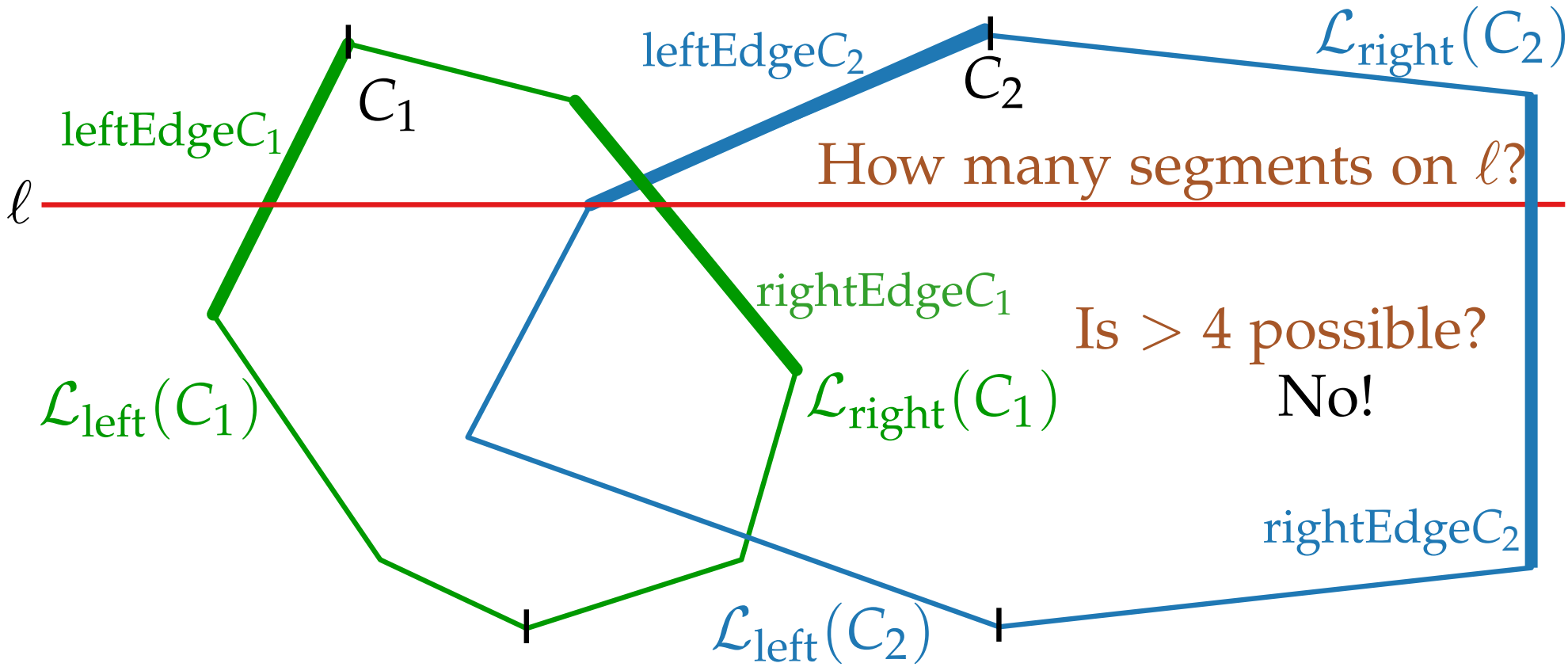


Intersecting Convex Regions



How does this help us?

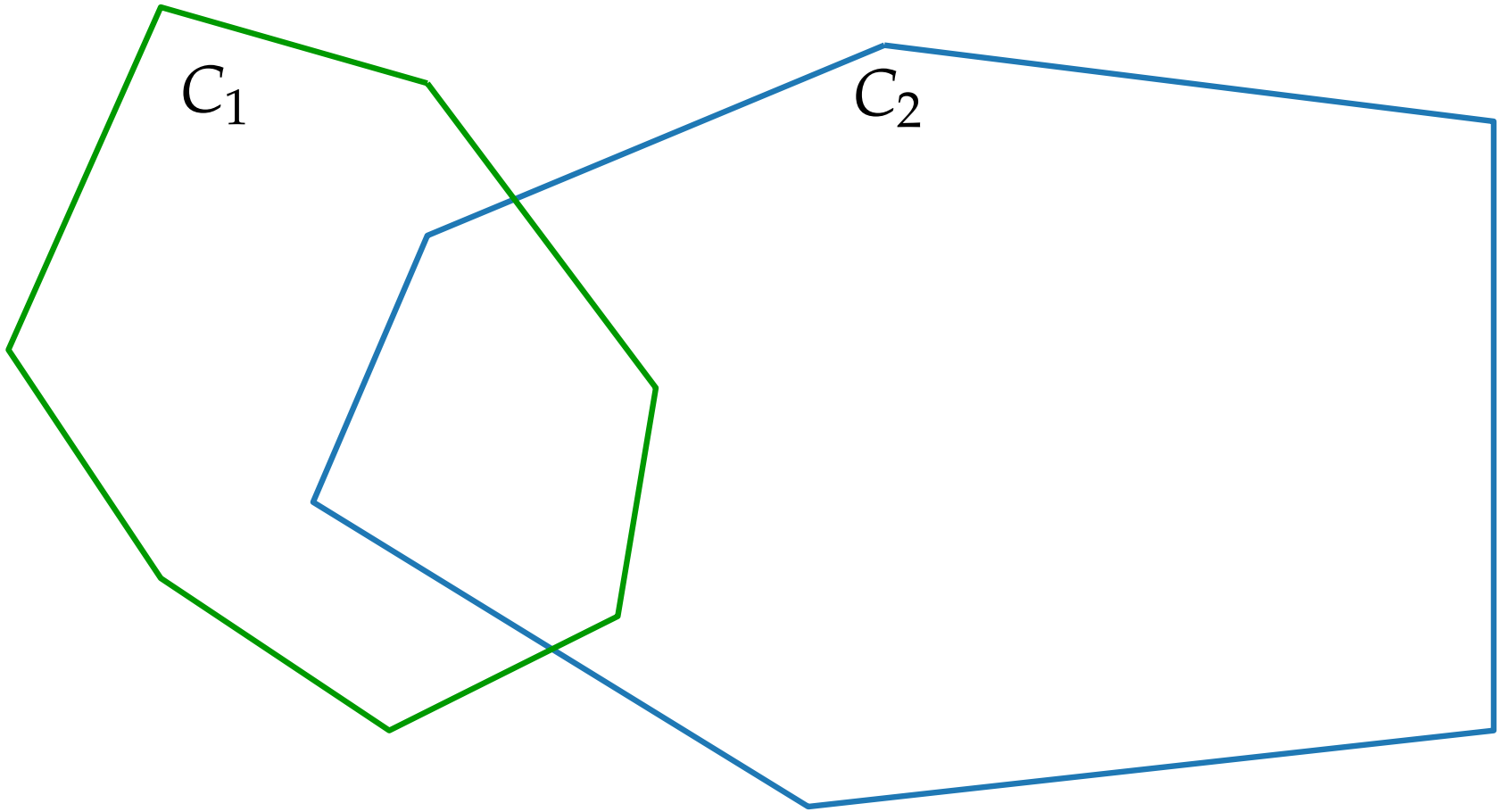
Intersecting Convex Regions



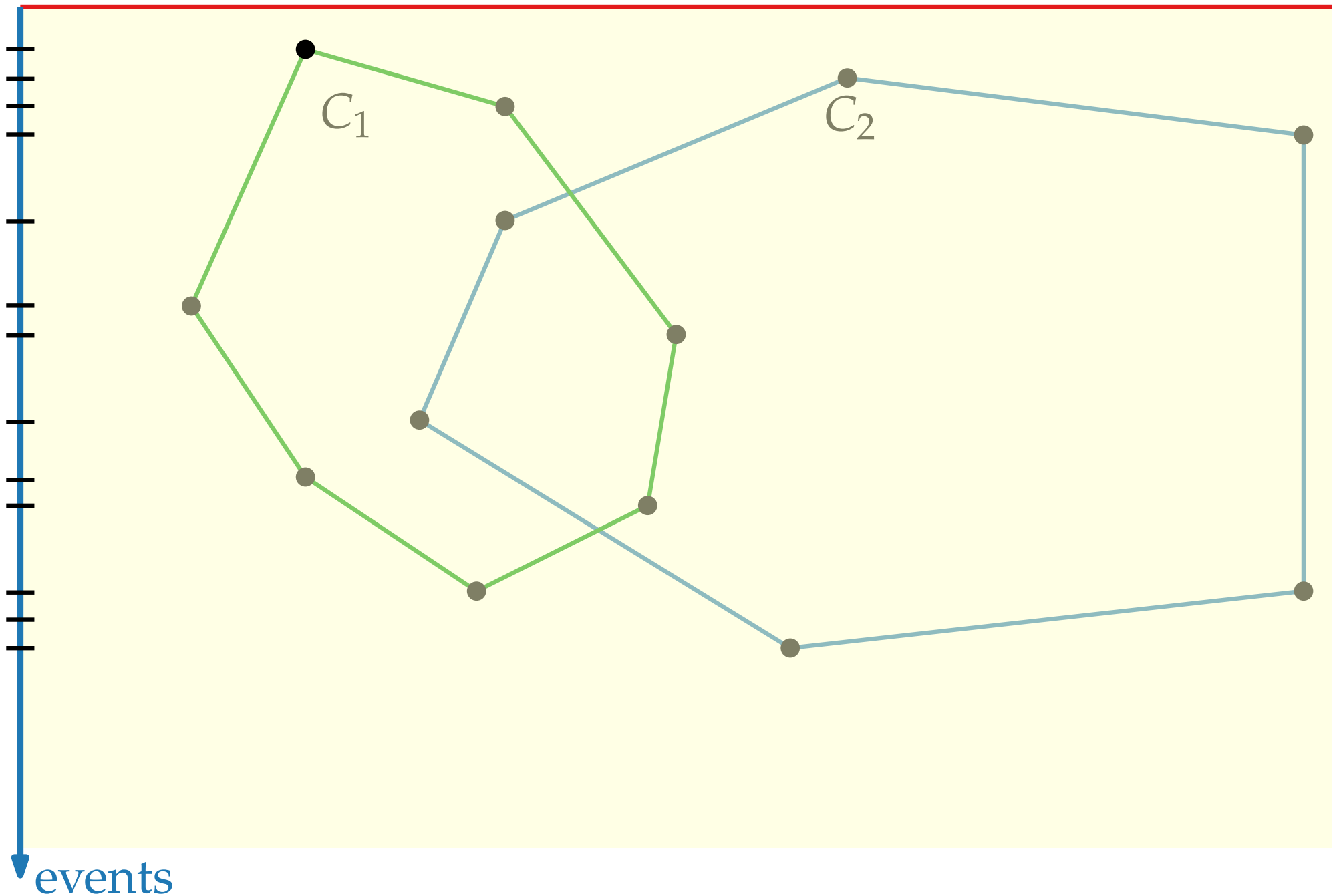
How does this help us?
$\rightsquigarrow$ sweep-line algorithm!

Theorem. The intersection of two convex polygonal regions can be computed in linear time.

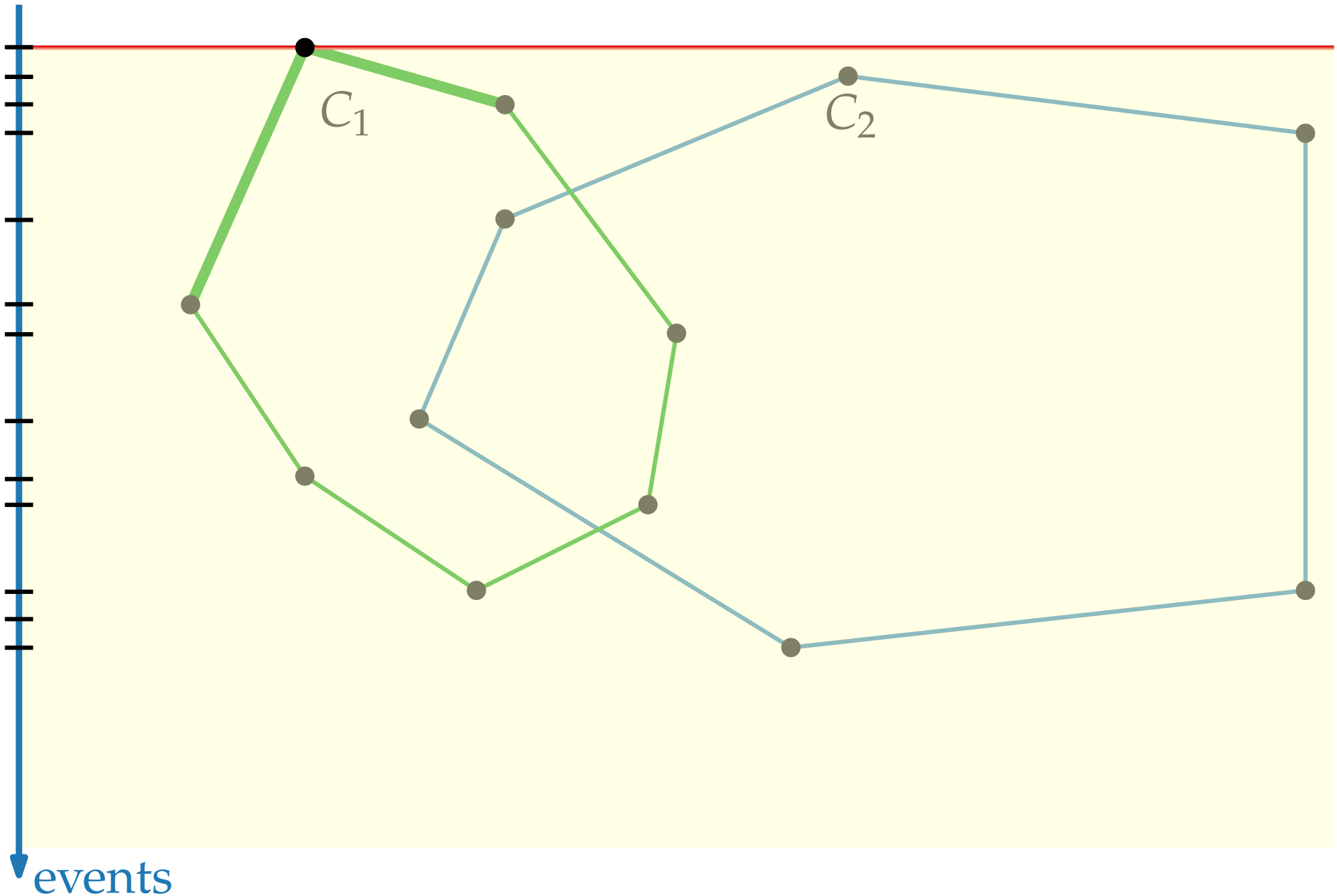
Sweep-Line Algorithm



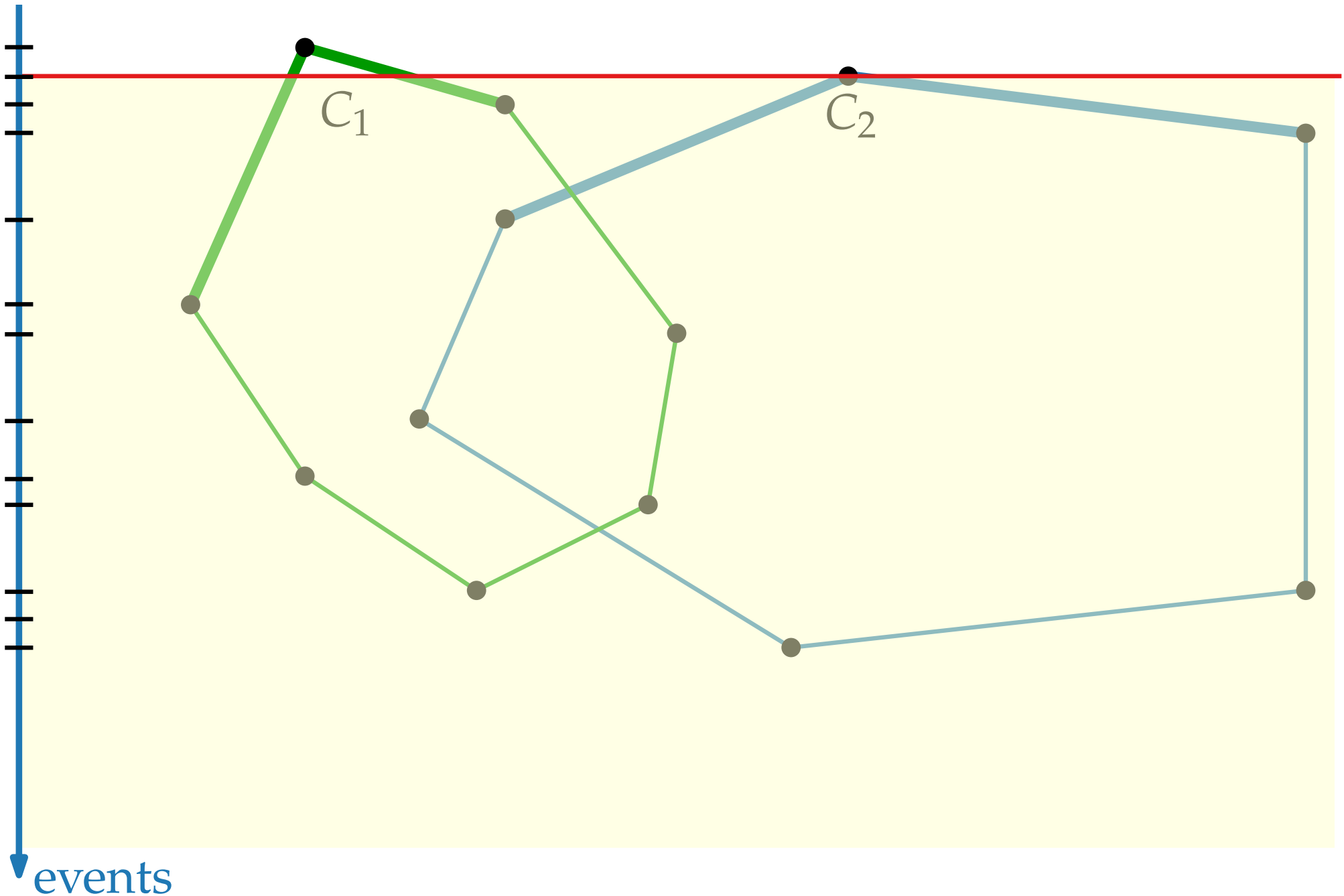
Sweep-Line Algorithm



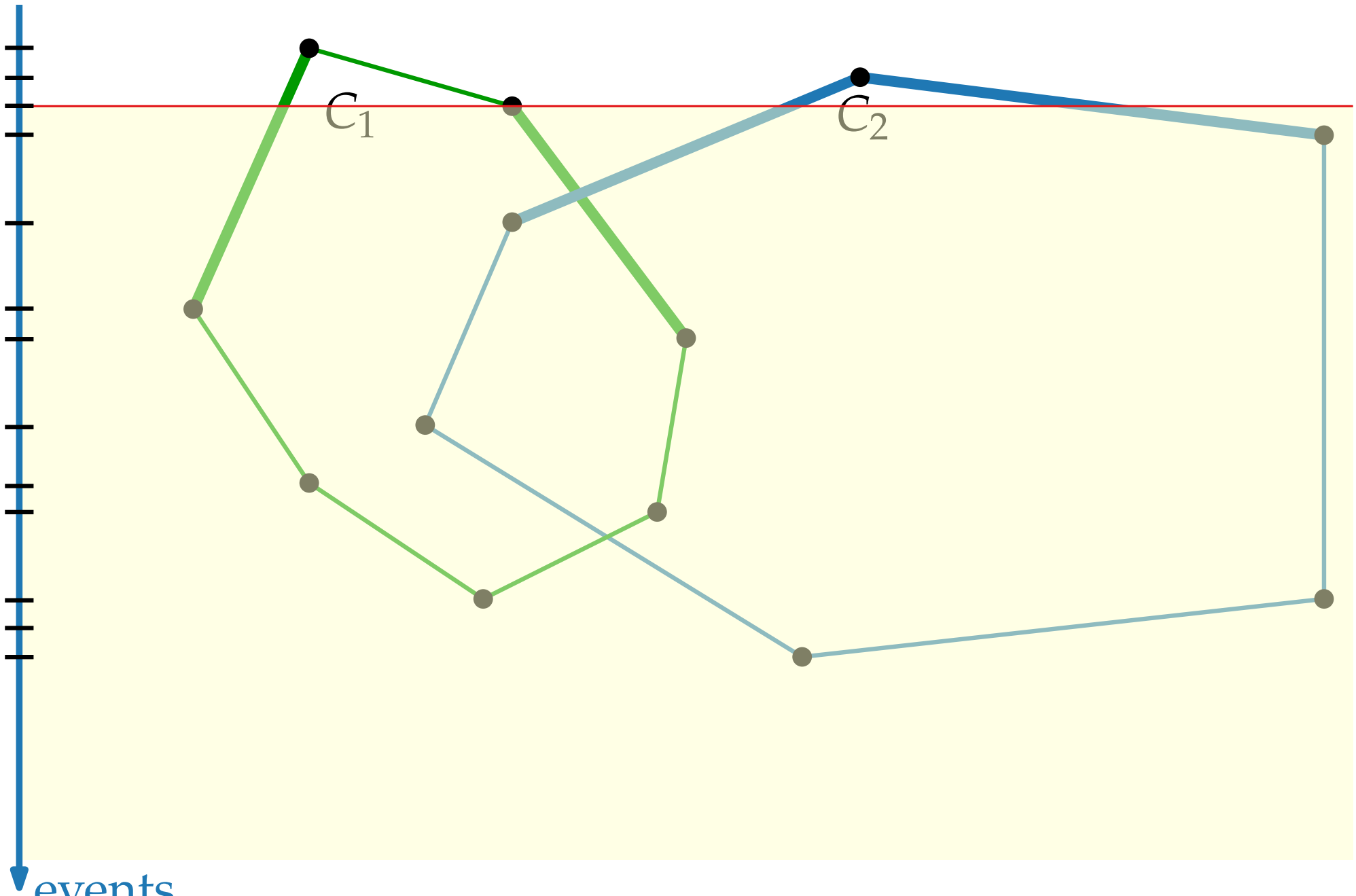
Sweep-Line Algorithm



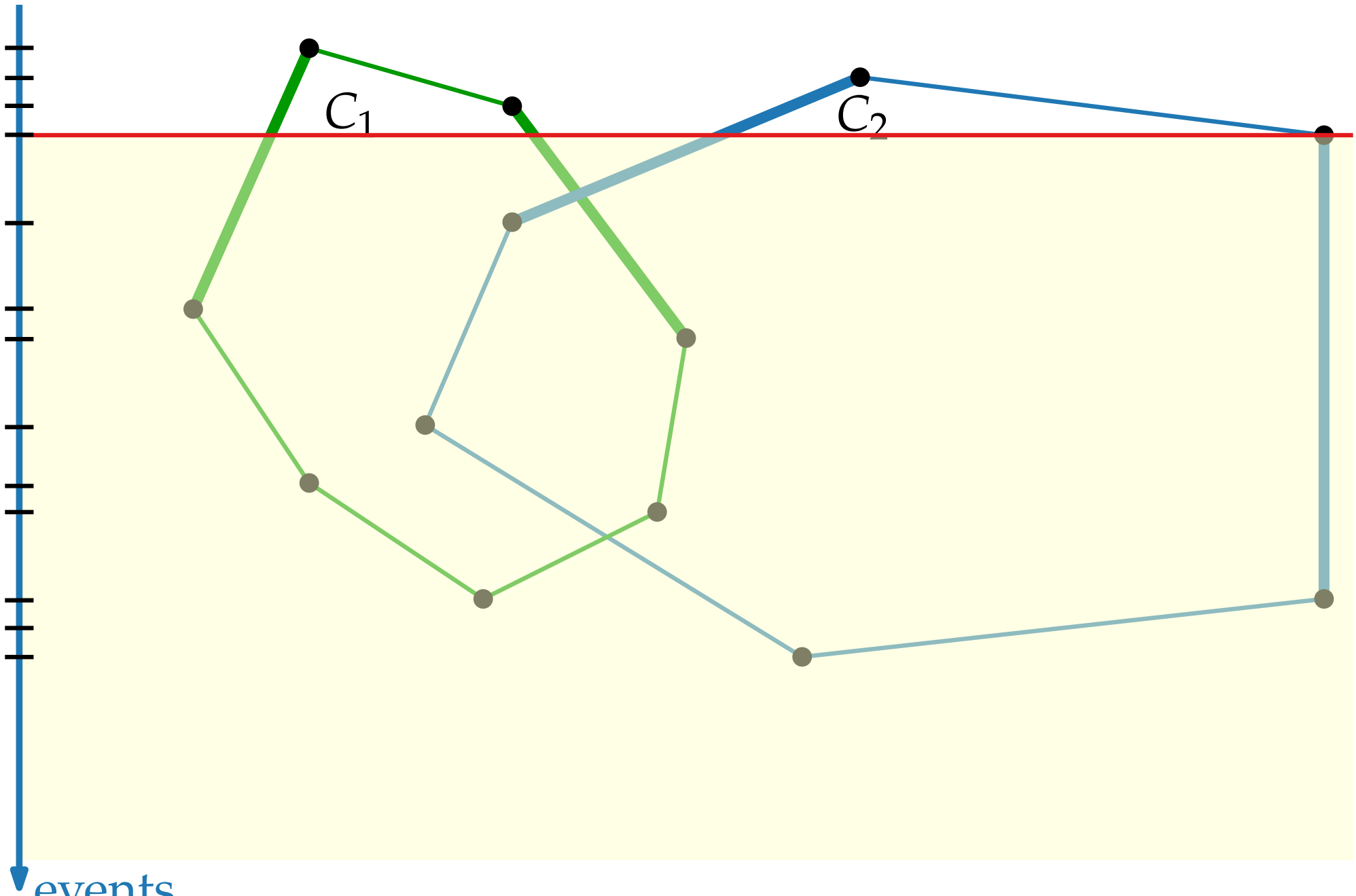
Sweep-Line Algorithm



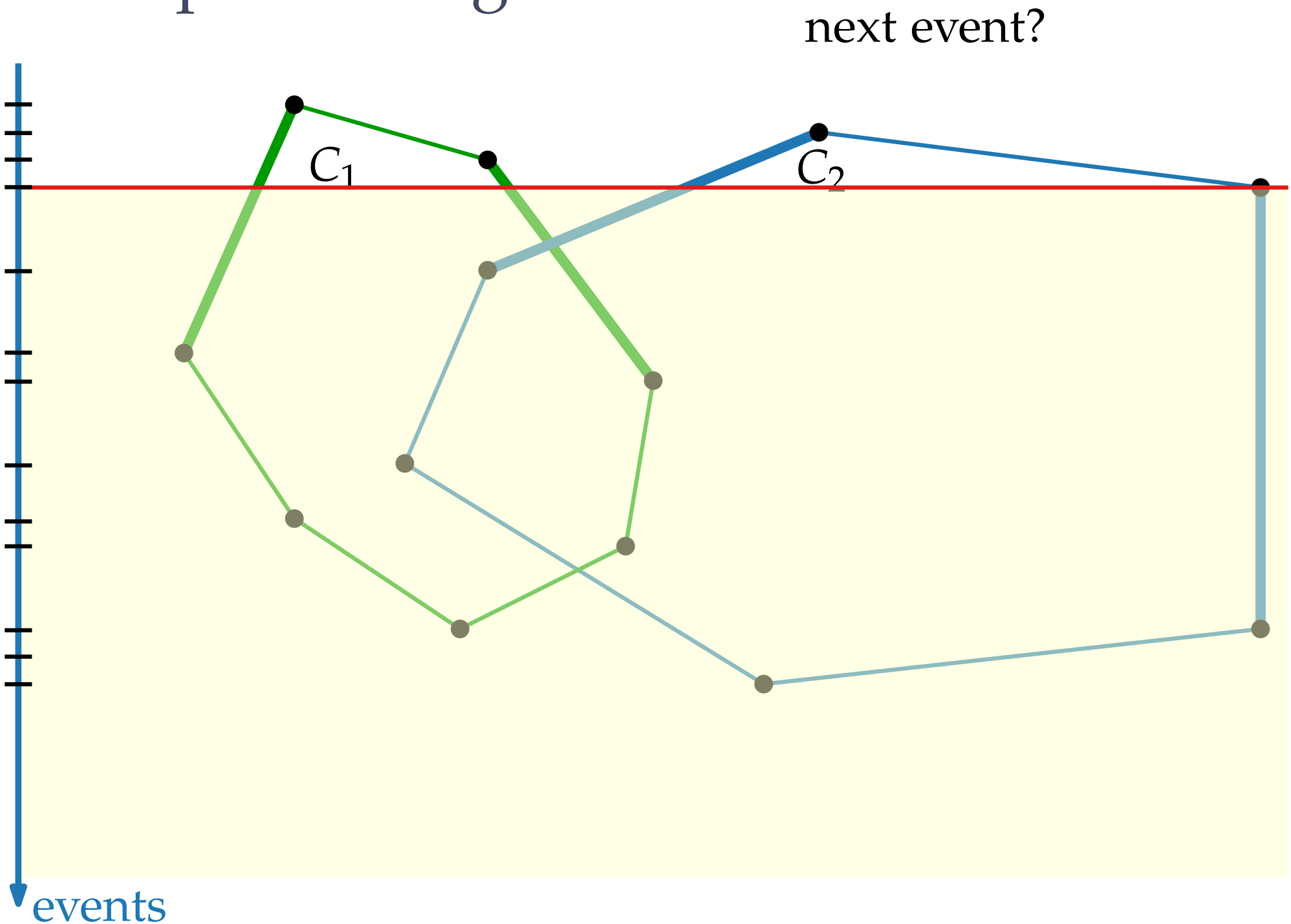
Sweep-Line Algorithm



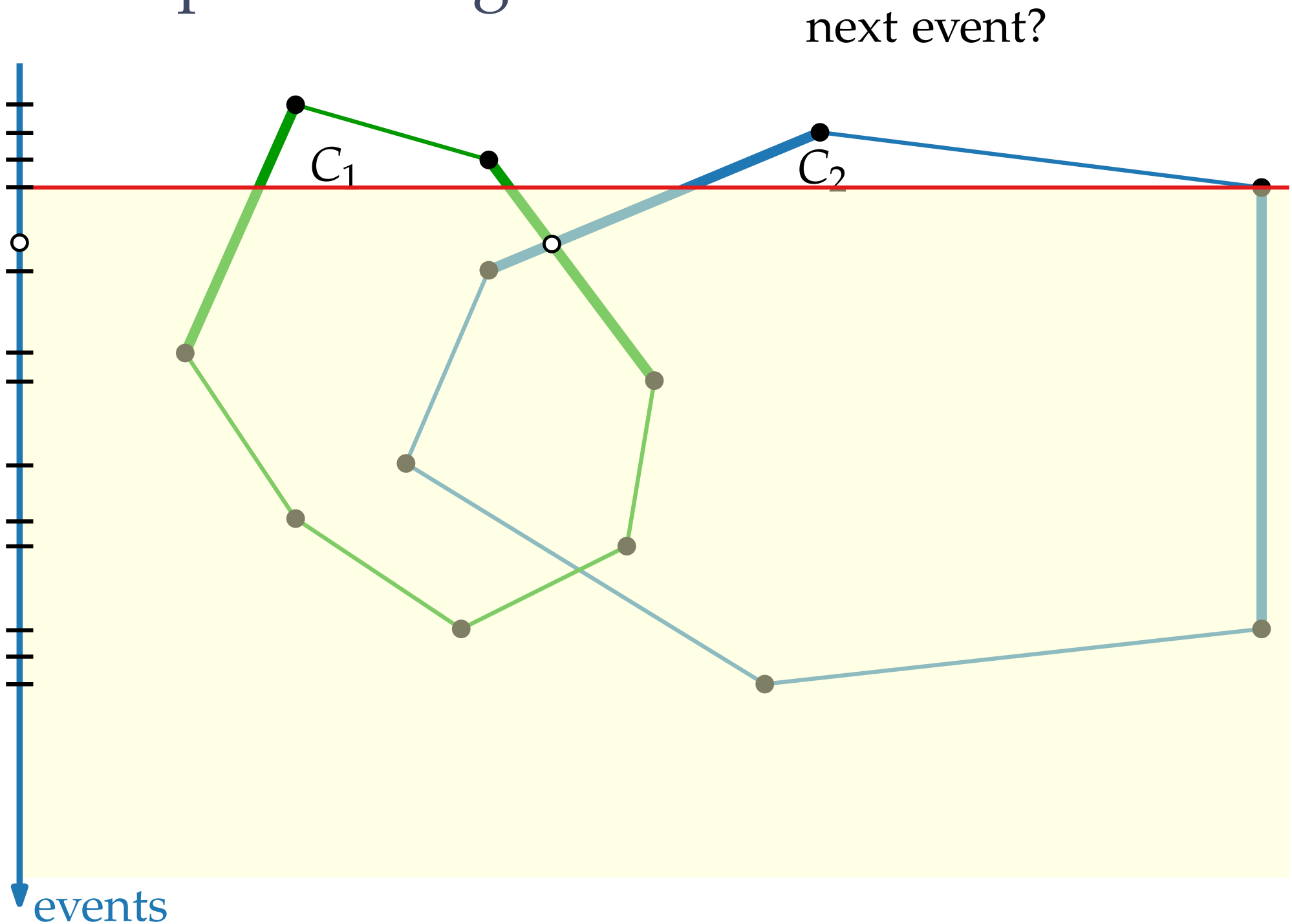
Sweep-Line Algorithm



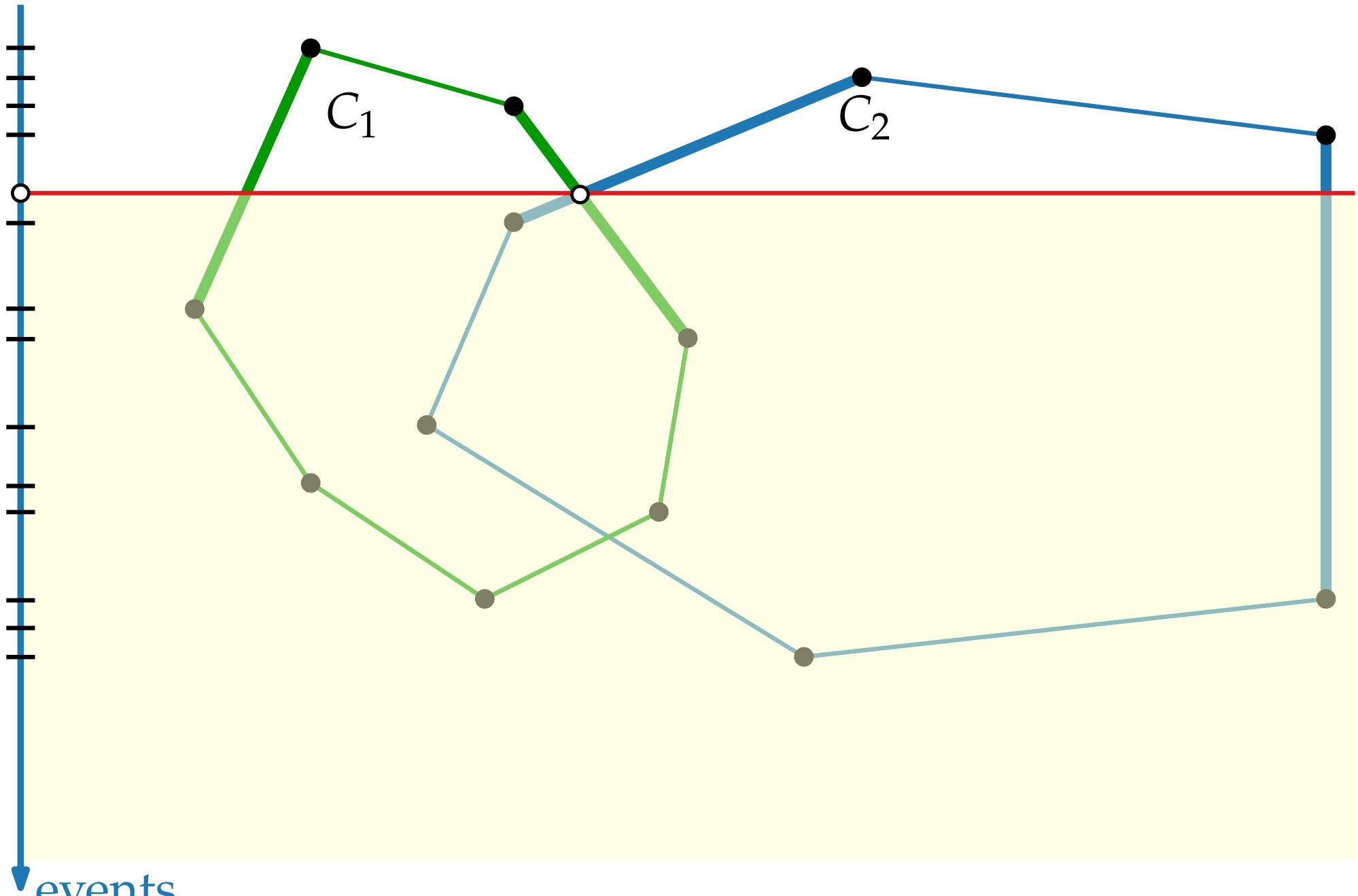
Sweep-Line Algorithm



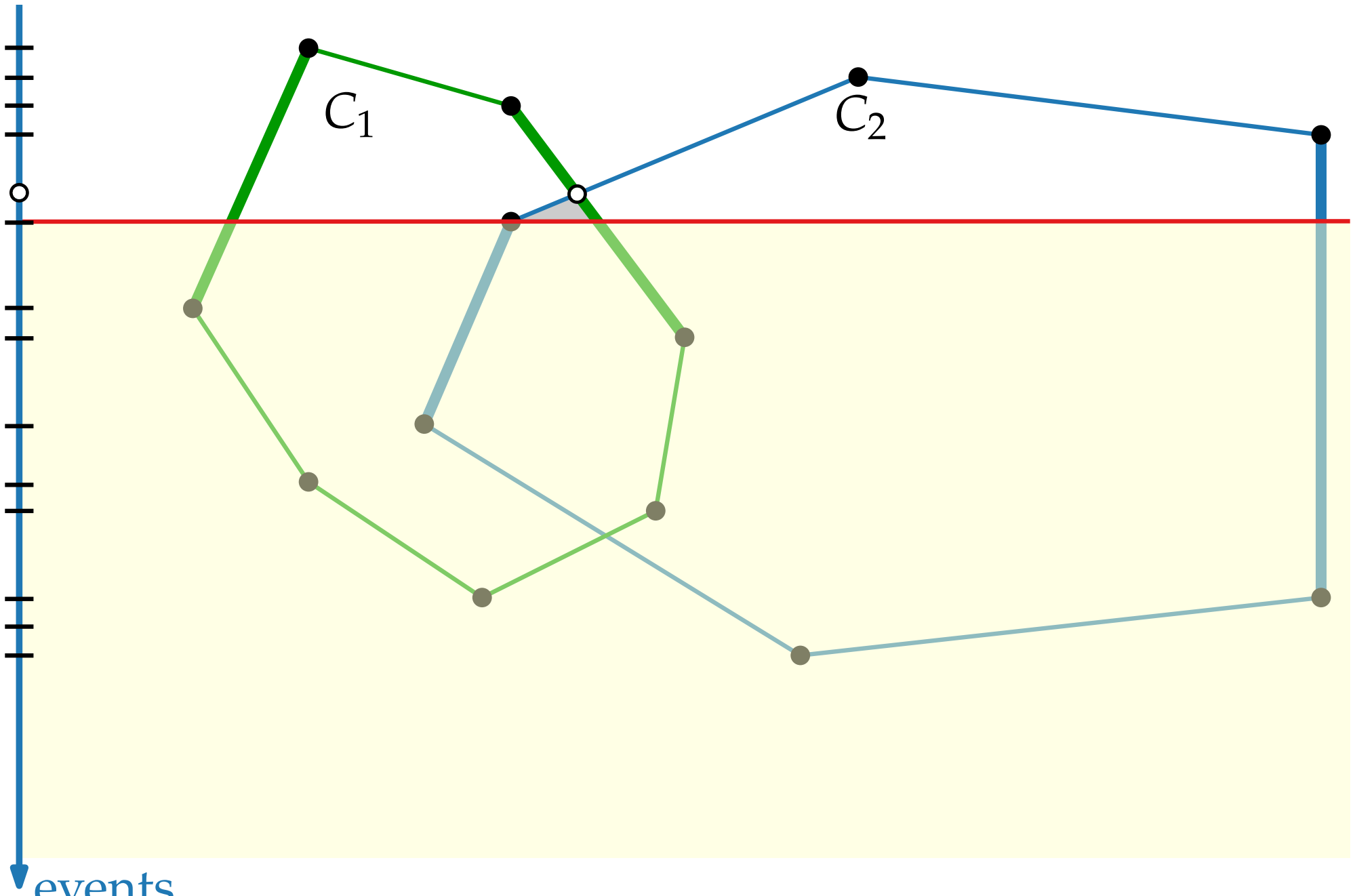
Sweep-Line Algorithm



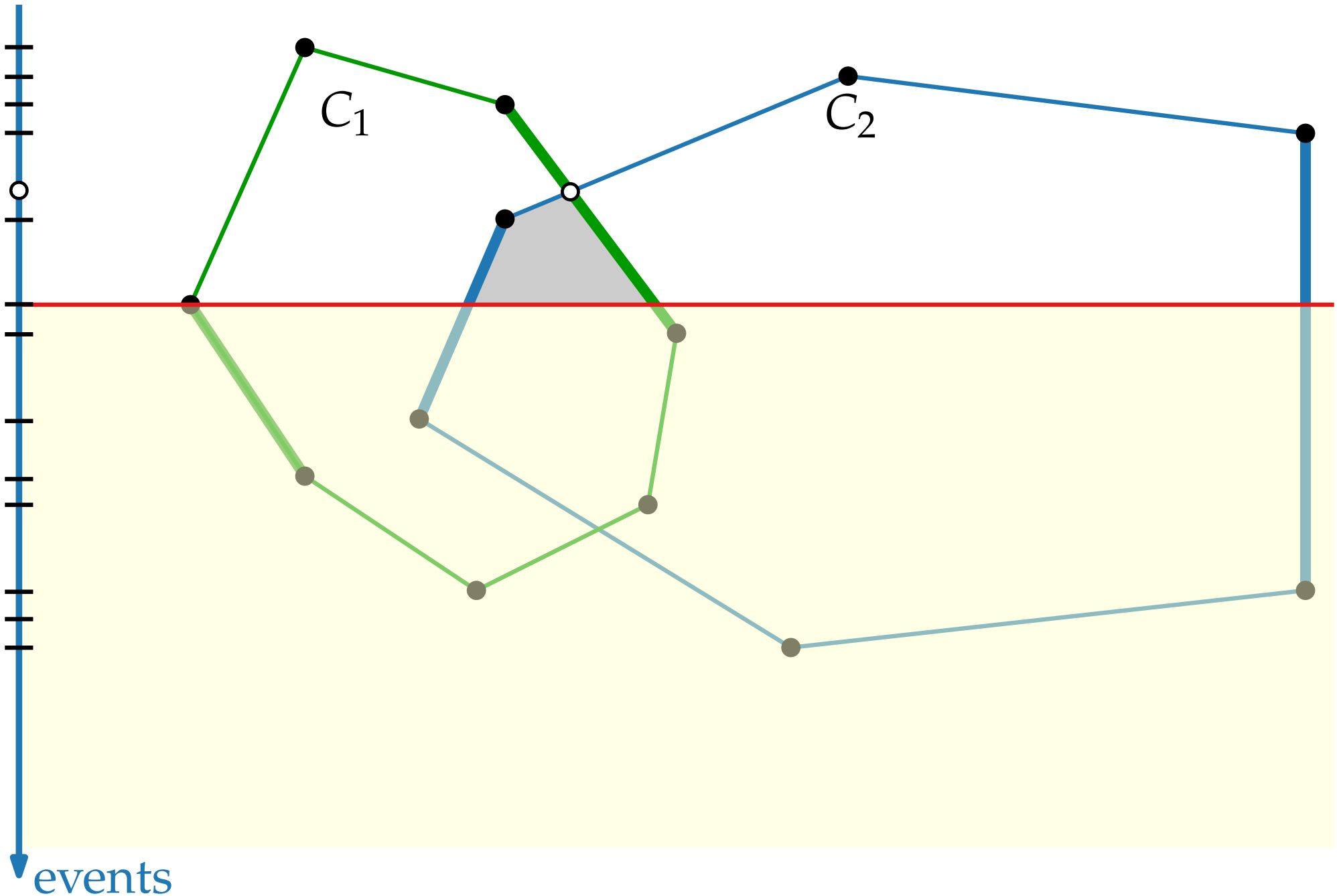
Sweep-Line Algorithm



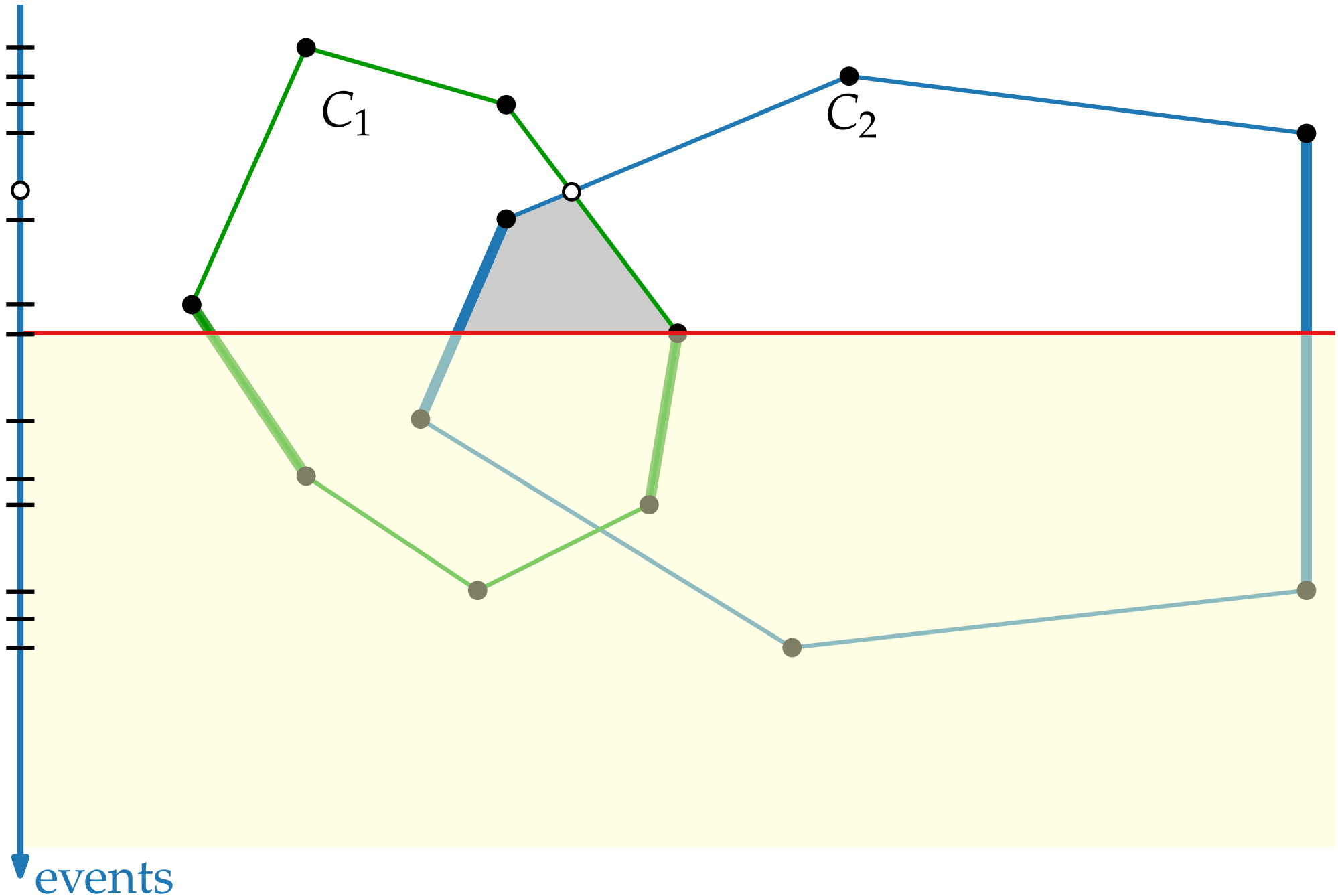
Sweep-Line Algorithm



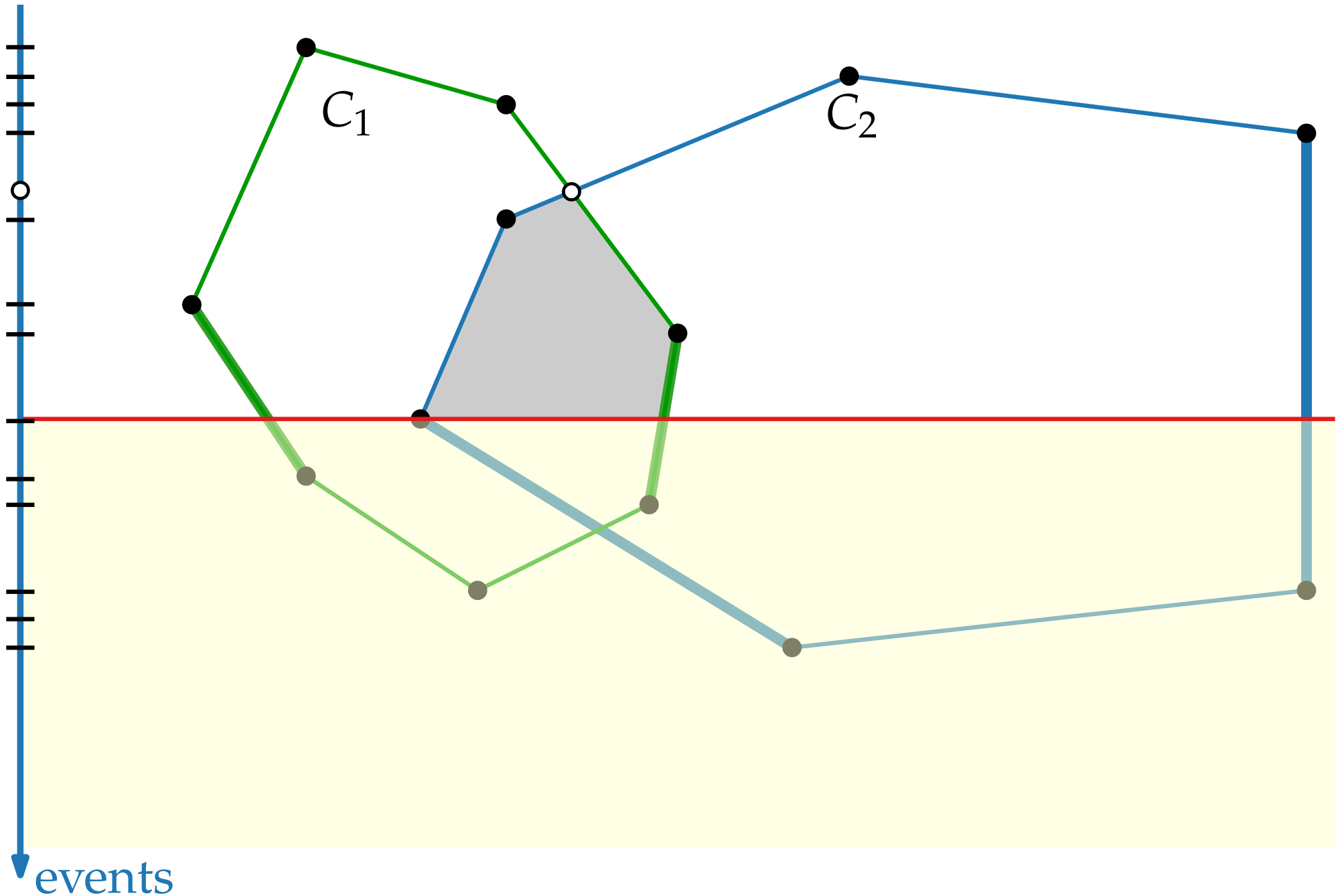
Sweep-Line Algorithm



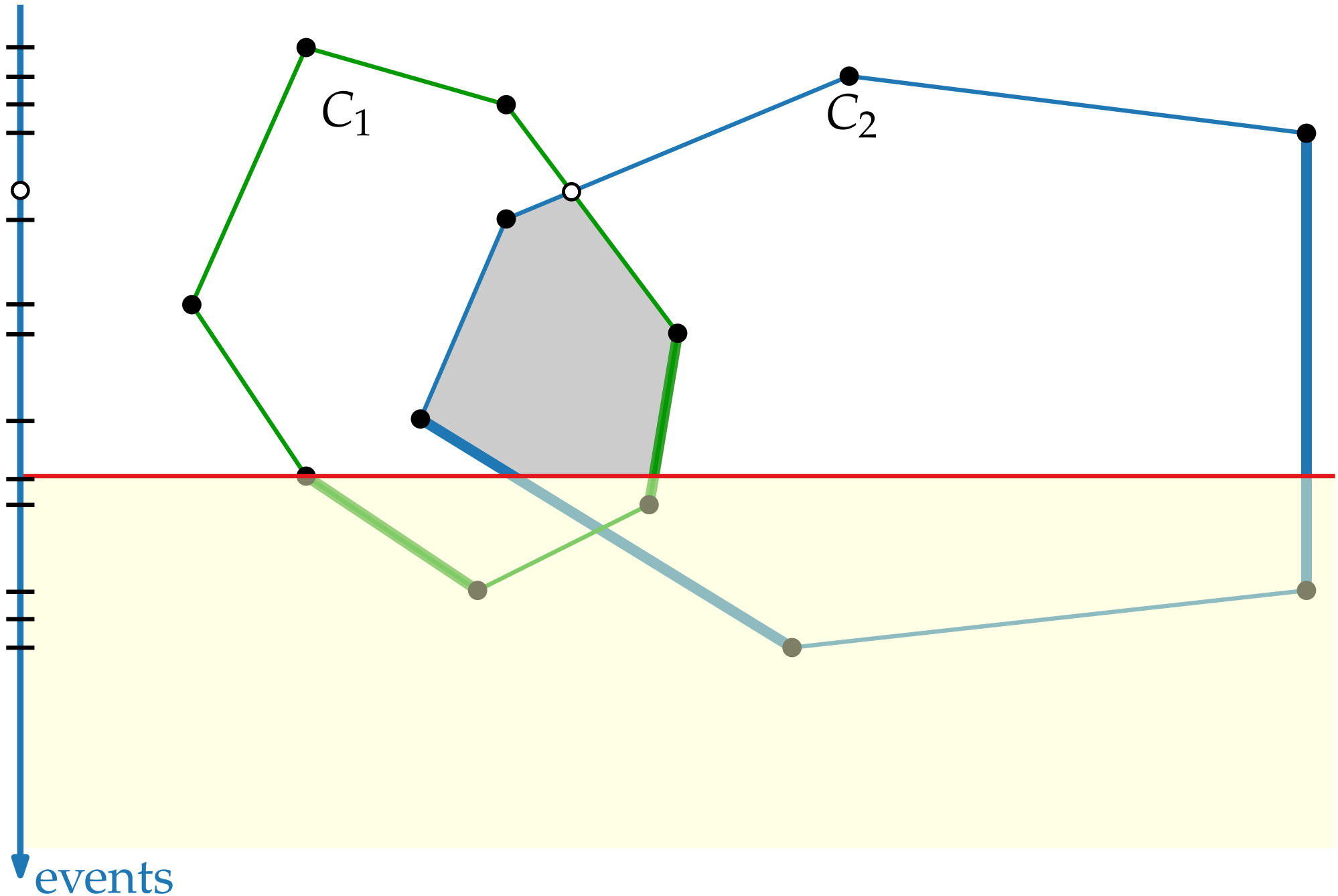
Sweep-Line Algorithm



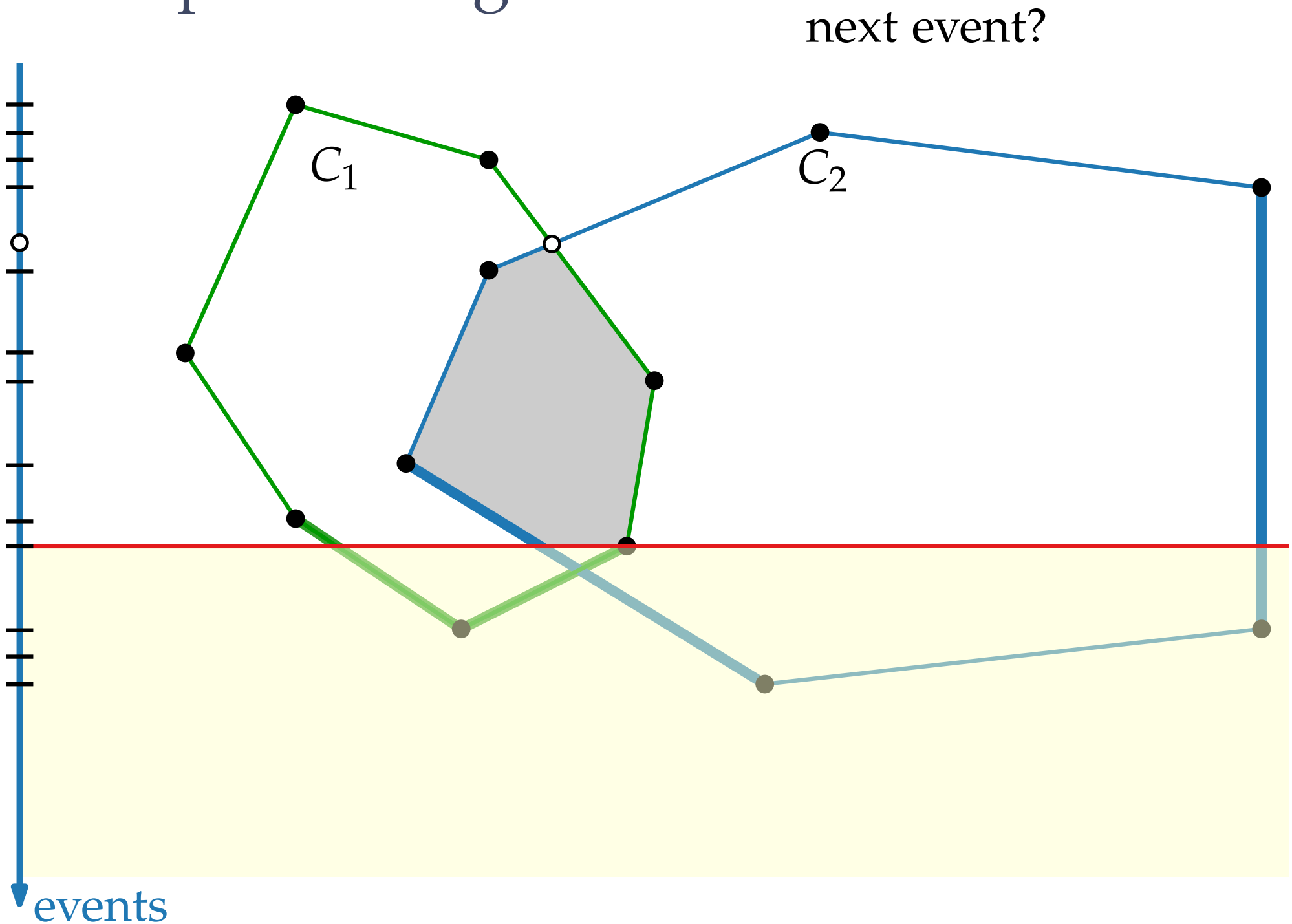
Sweep-Line Algorithm



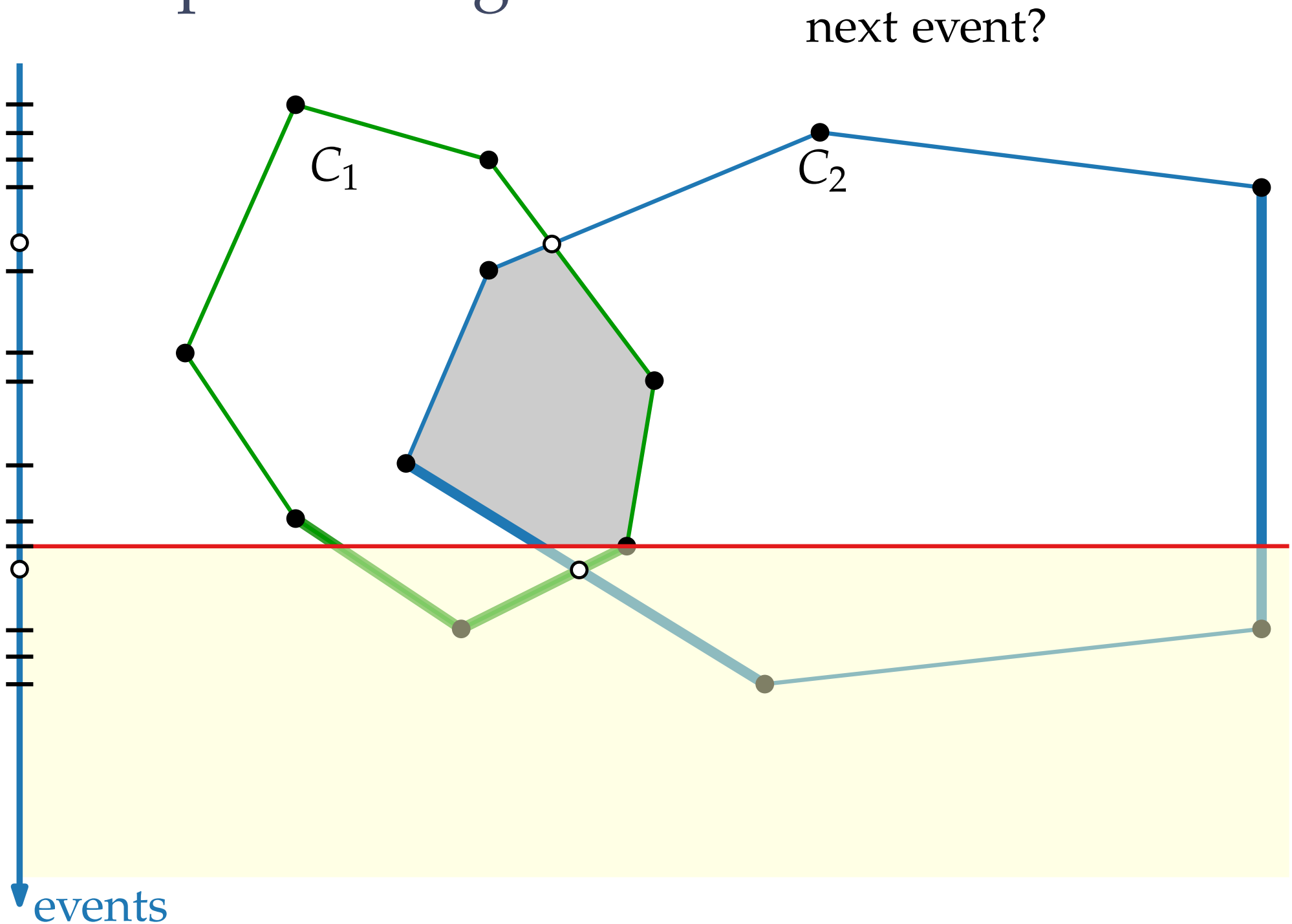
Sweep-Line Algorithm



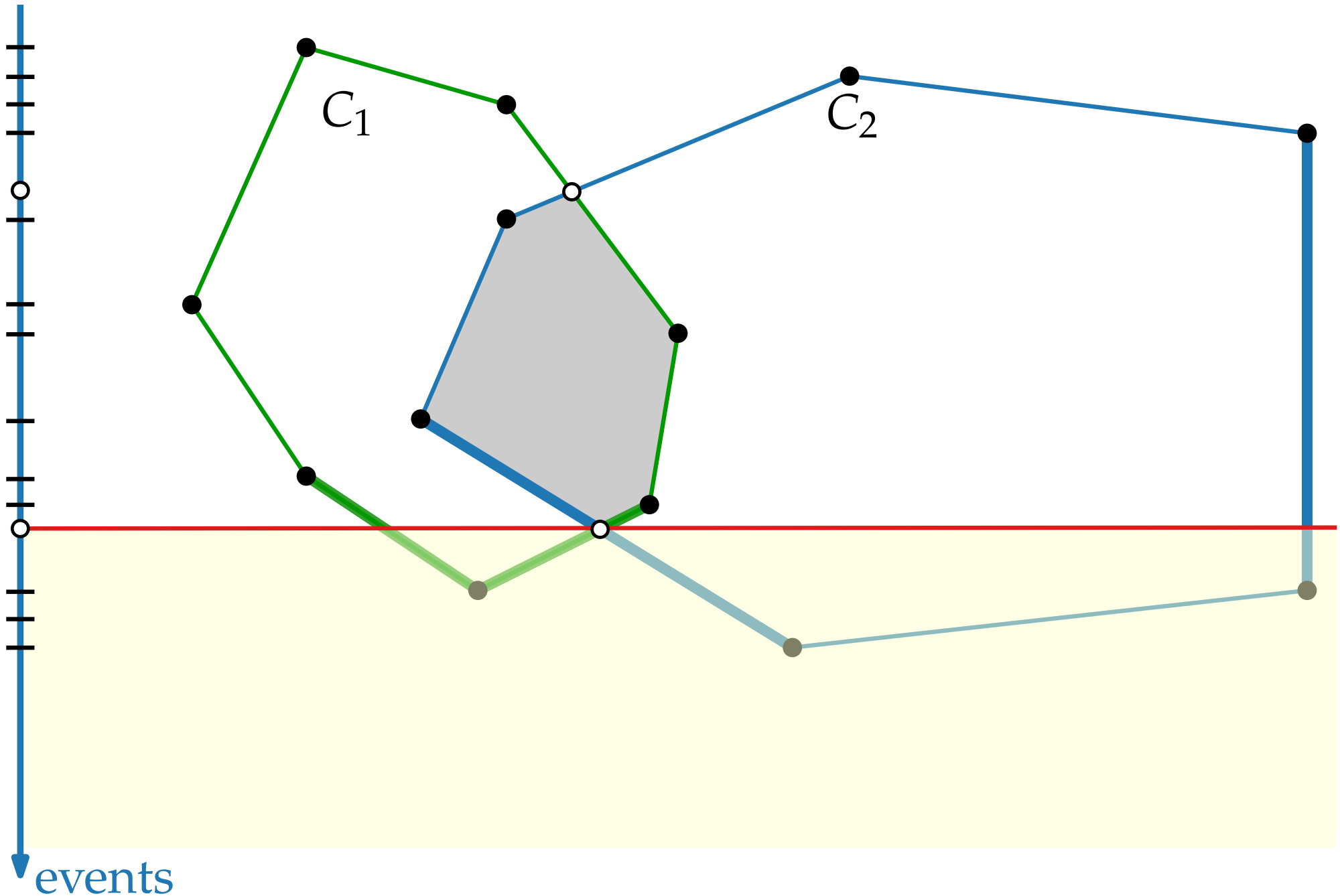
Sweep-Line Algorithm



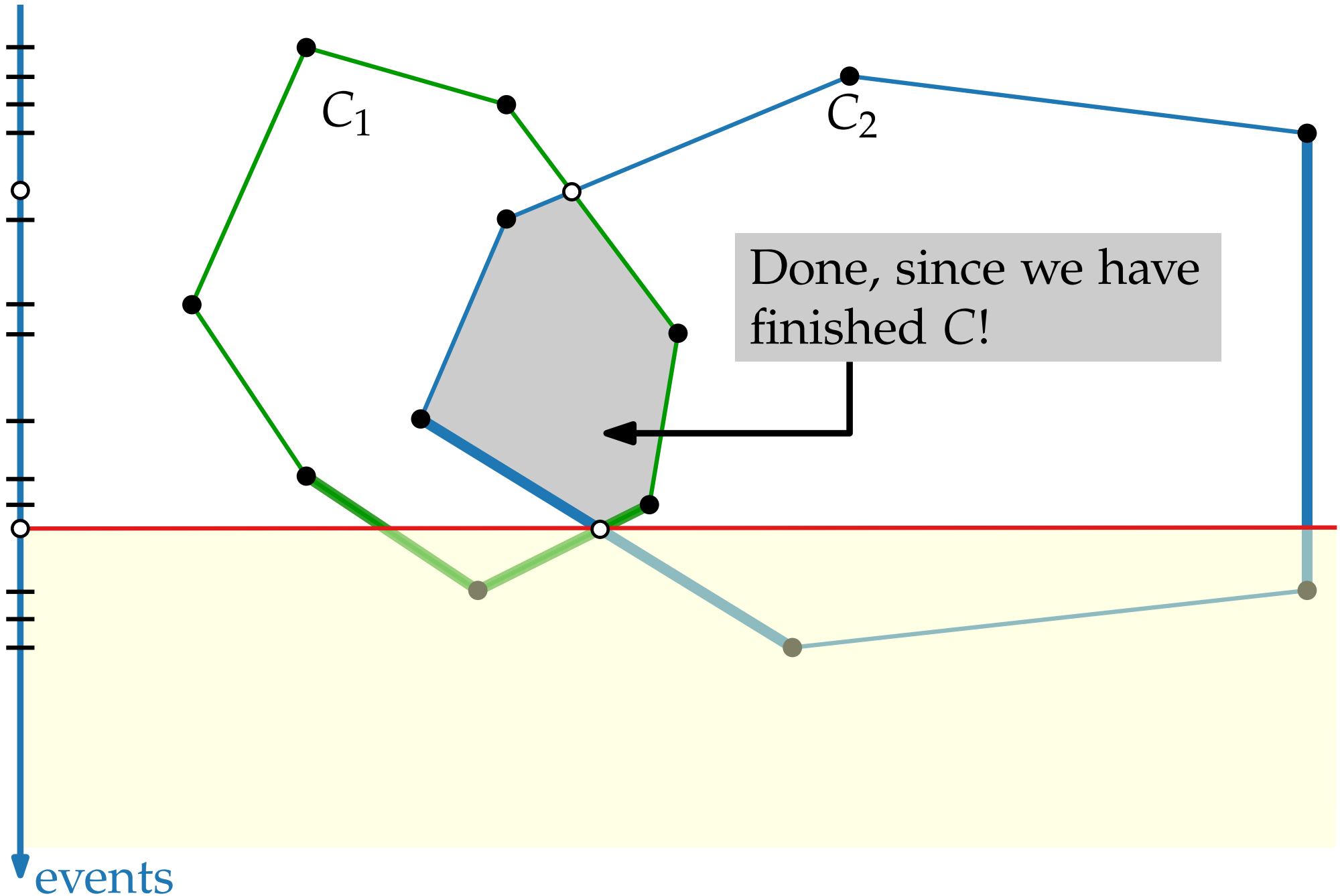
Sweep-Line Algorithm



Sweep-Line Algorithm



Sweep-Line Algorithm



Data Structures

1) event (-point) queue $\mathcal{Q}$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}}$$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q$$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q$$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

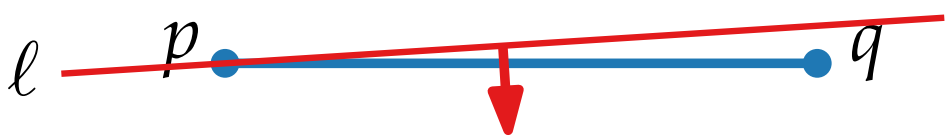
$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$

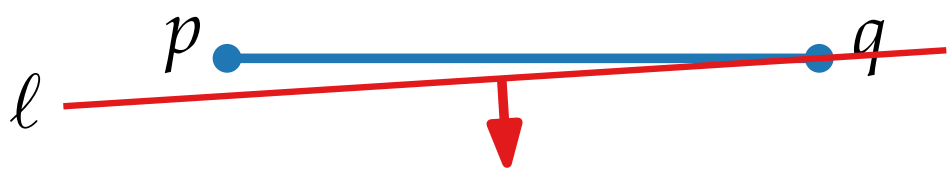


2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$

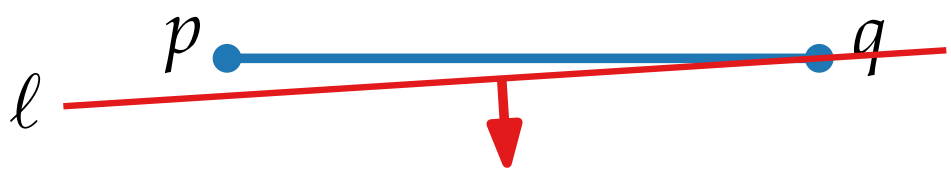


2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$



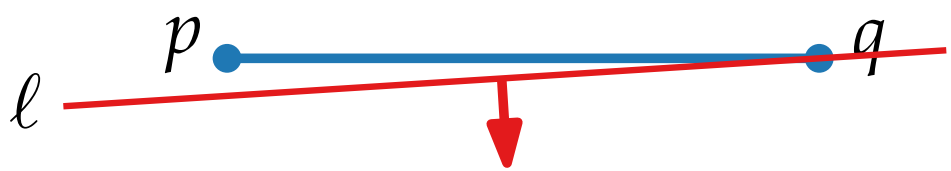
Store event pts in *sorted order* acc. to $\prec$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$



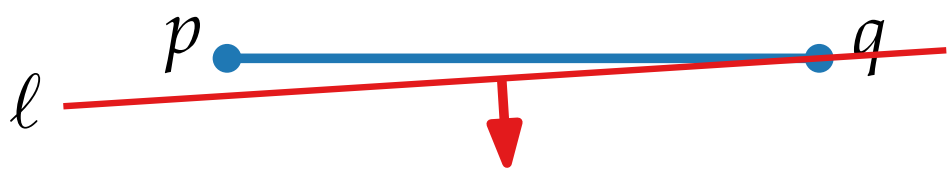
Store event pts in *sorted order* acc. to $\prec$ ... linear time?

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$



Store event pts in *sorted order* acc. to $\prec$

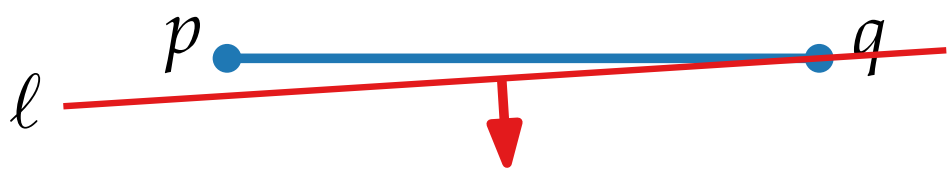
nextEvent() : either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$



Store event pts in *sorted order* acc. to $\prec$

nextEvent() : either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

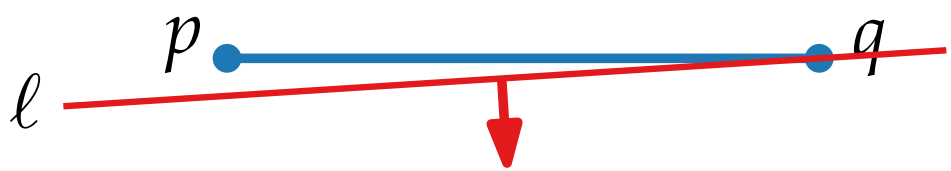
... runtime?

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$



Store event pts in *sorted order* acc. to $\prec$

nextEvent() : either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

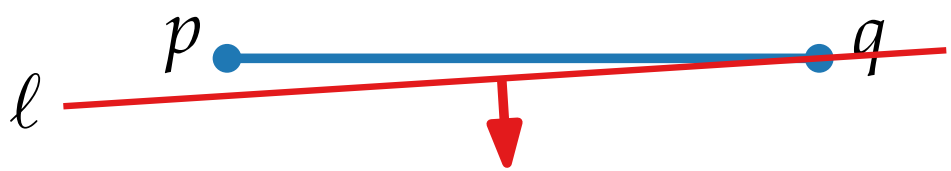
... runtime? $O(1)$, since num. active segments ≤ 4 :)

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$

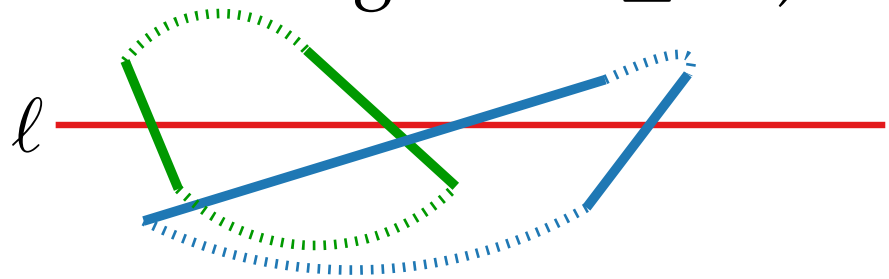


Store event pts in *sorted order* acc. to $\prec$

nextEvent() : either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

... runtime? $O(1)$, since num. active segments ≤ 4 :)

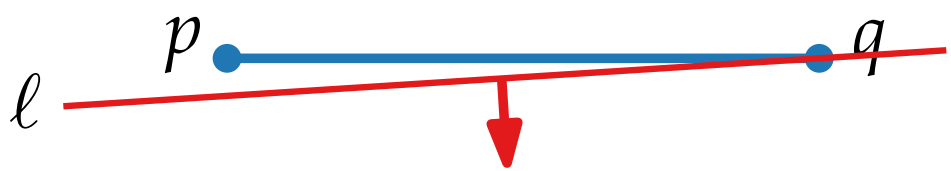
2) (sweep-line) status $\mathcal{T}$



Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$

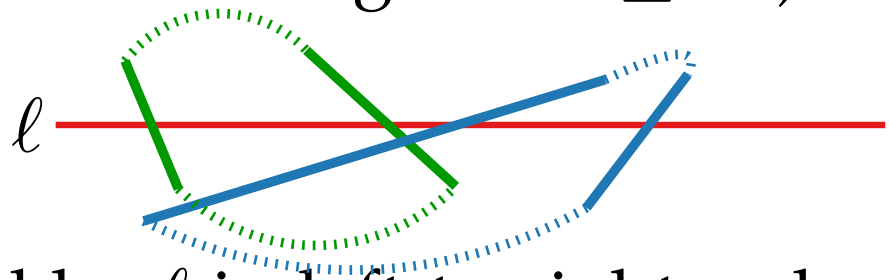


Store event pts in *sorted order* acc. to $\prec$

nextEvent() : either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

... runtime? $O(1)$, since num. active segments ≤ 4 :)

2) (sweep-line) status $\mathcal{T}$

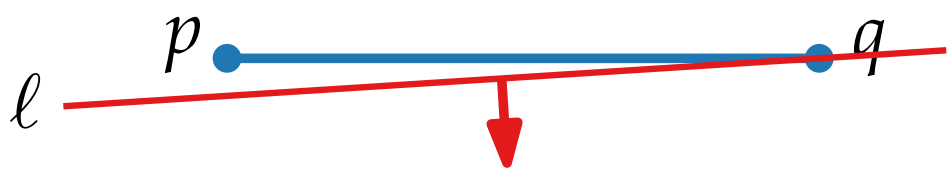


Store the segments intersected by ℓ in left-to-right order.

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$

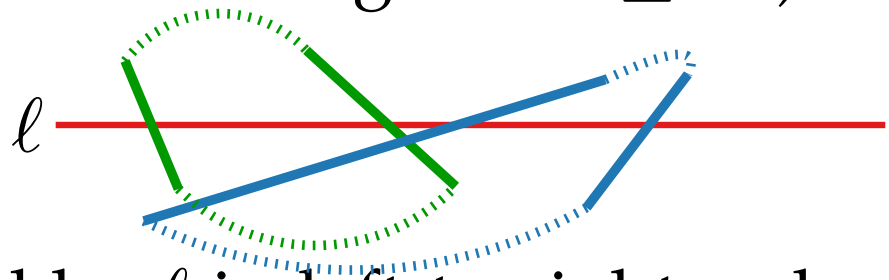


Store event pts in *sorted order* acc. to $\prec$

nextEvent() : either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

... runtime? $O(1)$, since num. active segments ≤ 4 :)

2) (sweep-line) status $\mathcal{T}$



Store the segments intersected by ℓ in left-to-right order.

Also, maintain the new convex hull.

Second Approach: Halfplane Intersection

Theorem. The intersection of two convex polygonal regions can be computed in linear time.

Second Approach: Halfplane Intersection

Theorem. The intersection of two convex polygonal regions can be computed in linear time.

IntersectHalfplanes(H)

if $|H| = 1$ **then** $C \leftarrow h$, where $\{h\} = H$
else
  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)
  $C_2 \leftarrow$ IntersectHalfplanes(H_2)
  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)
return C

Running time: $T_{\text{IH}}(n) = 2T_{\text{IH}}(n/2) + T_{\text{ICR}}(n)$

Second Approach: Halfplane Intersection

Theorem. The intersection of two convex polygonal regions can be computed in linear time.

IntersectHalfplanes(H)

if $|H| = 1$ **then** $C \leftarrow h$, where $\{h\} = H$
else
  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)
  $C_2 \leftarrow$ IntersectHalfplanes(H_2)
  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)
return C

Running time: $T_{\text{IH}}(n) = 2T_{\text{IH}}(n/2) + T_{\text{ICR}}(n)$

Corollary. The intersection of n half planes can be computed in $O(n \log n)$ time.

Second Approach: Halfplane Intersection

Theorem. The intersection of two convex polygonal regions can be computed in linear time.

IntersectHalfplanes(H)

if $|H| = 1$ **then** $C \leftarrow h$, where $\{h\} = H$
else
  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)
  $C_2 \leftarrow$ IntersectHalfplanes(H_2)
  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)
return C

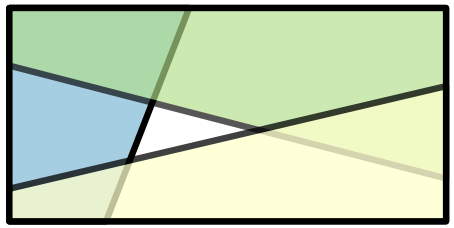
Running time: $T_{\text{IH}}(n) = 2T_{\text{IH}}(n/2) + T_{\text{ICR}}(n)$

Corollary. The intersection of n half planes can be computed in $O(n \log n)$ time.

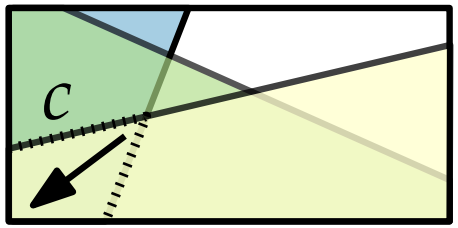
Can we do better?

A Small Trick: Make Solution Unique

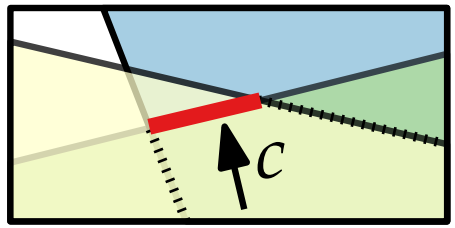
$\bigcap H = \emptyset$

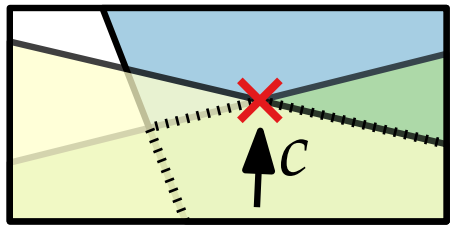


$\bigcap H$ unbnd. in dir. c



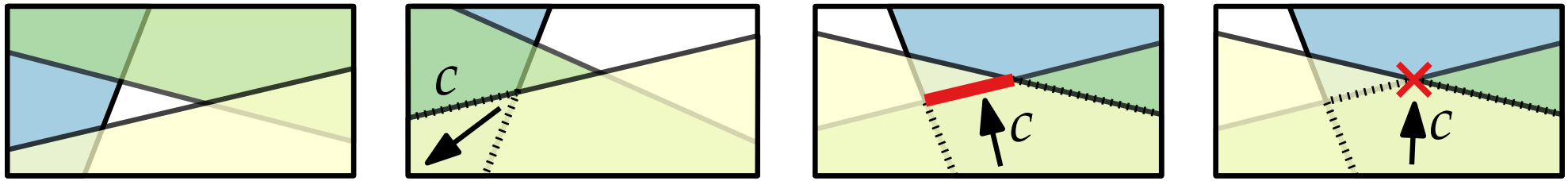
$\bigcap H$ bounded.





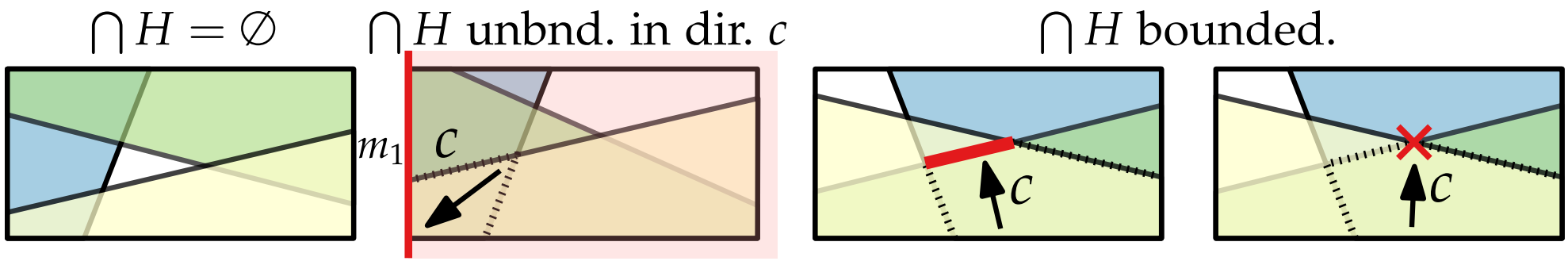
A Small Trick: Make Solution Unique

$\bigcap H = \emptyset$ $\bigcap H$ unbnd. in dir. c $\bigcap H$ bounded.



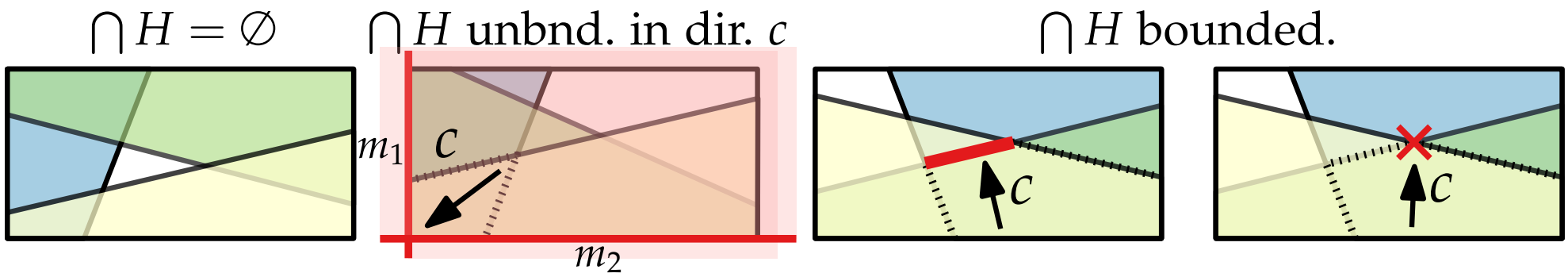
- Add two bounding halfplanes m_1 and m_2

A Small Trick: Make Solution Unique



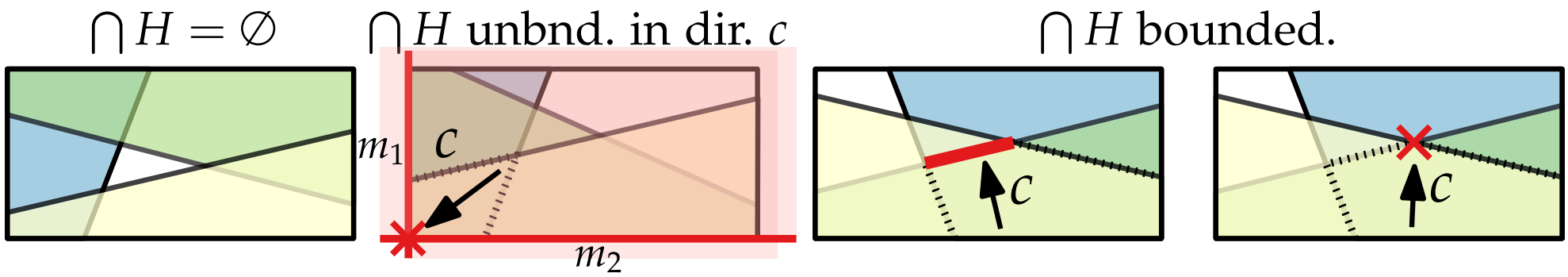
- Add two bounding halfplanes m_1 and m_2

A Small Trick: Make Solution Unique



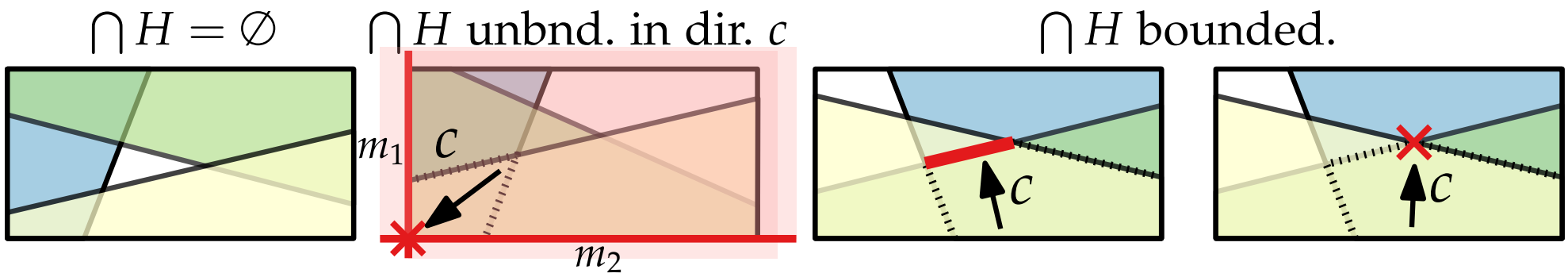
- Add two bounding halfplanes m_1 and m_2

A Small Trick: Make Solution Unique



- Add two bounding halfplanes m_1 and m_2

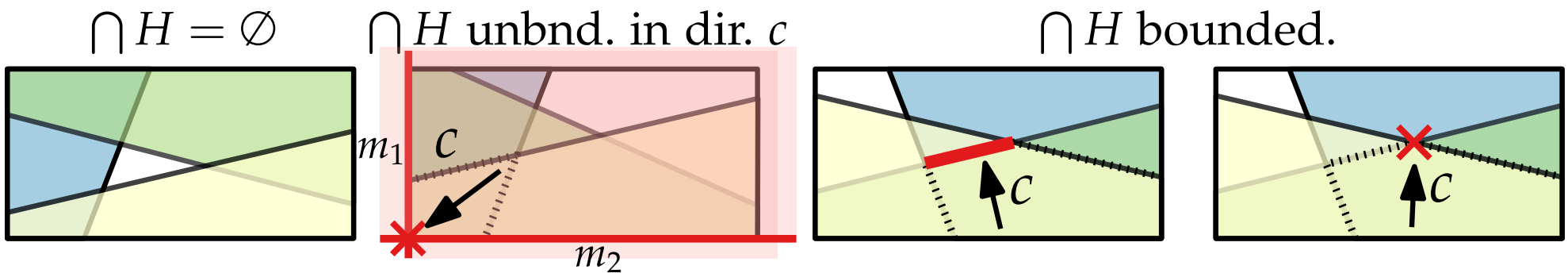
A Small Trick: Make Solution Unique



- Add two bounding halfplanes m_1 and m_2

$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases}$$ for some sufficiently large M

A Small Trick: Make Solution Unique

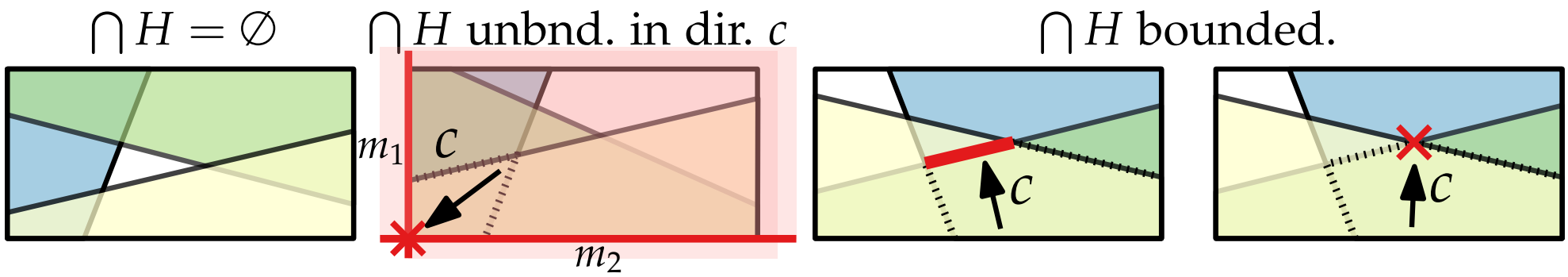


- Add two bounding halfplanes m_1 and m_2

$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases}$$ for some sufficiently large M

$$m_2 = \begin{cases} y \leq M & \text{if } c_y > 0, \\ y \geq M & \text{otherwise.} \end{cases}$$

A Small Trick: Make Solution Unique



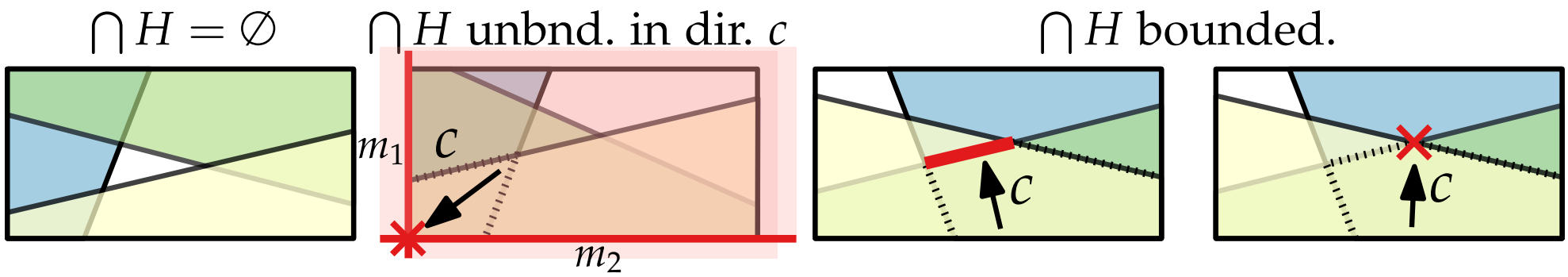
- Add two bounding halfplanes m_1 and m_2

$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases}$$ for some sufficiently large M

$$m_2 = \begin{cases} y \leq M & \text{if } c_y > 0, \\ y \geq M & \text{otherwise.} \end{cases}$$

Idea: M based on obj.fct. c.
see §4.5 of CG: A&A for more on unbounded LPs.

A Small Trick: Make Solution Unique



- Add two bounding halfplanes m_1 and m_2

$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases}$$

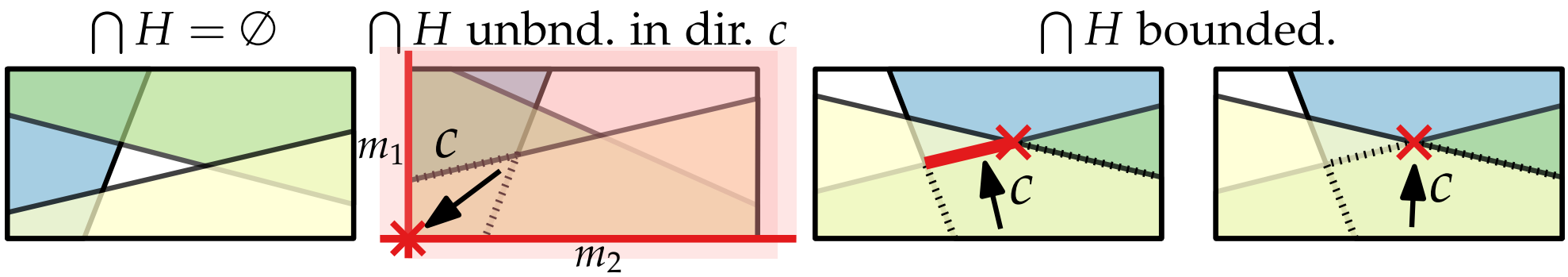
for some sufficiently large M

$$m_2 = \begin{cases} y \leq M & \text{if } c_y > 0, \\ y \geq M & \text{otherwise.} \end{cases}$$

Idea: M based on obj.fct. c.
see §4.5 of CG: A&A for more on unbounded LPs.

- Take the lexicographically largest solution.

A Small Trick: Make Solution Unique



- Add two bounding halfplanes m_1 and m_2

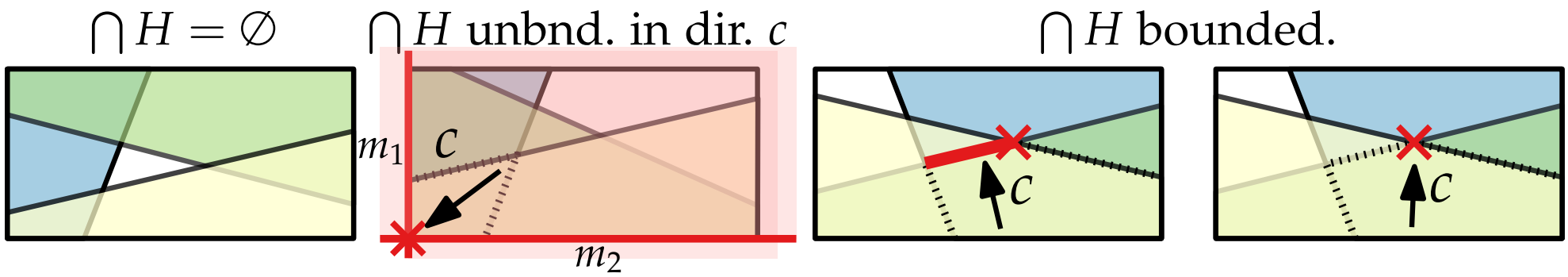
$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases}$$ for some sufficiently large M

$$m_2 = \begin{cases} y \leq M & \text{if } c_y > 0, \\ y \geq M & \text{otherwise.} \end{cases}$$

Idea: M based on obj.fct. c.
see §4.5 of CG: A&A for more on unbounded LPs.

- Take the lexicographically largest solution.

A Small Trick: Make Solution Unique



- Add two bounding halfplanes m_1 and m_2

$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases}$$ for some sufficiently large M

$$m_2 = \begin{cases} y \leq M & \text{if } c_y > 0, \\ y \geq M & \text{otherwise.} \end{cases}$$

Idea: M based on obj.fct. c.
see §4.5 of CG: A&A for more on unbounded LPs.

- Take the lexicographically largest solution.

$\Rightarrow$ Set of solutions is either empty or a uniquely defined pt.

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$

return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
    if $v_{i-1} \in h_i$ **then**

return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow$
  else
    $v_i \leftarrow$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

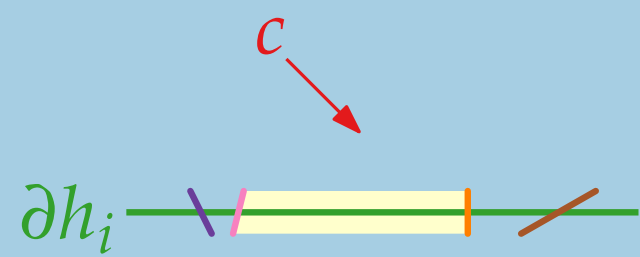
$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

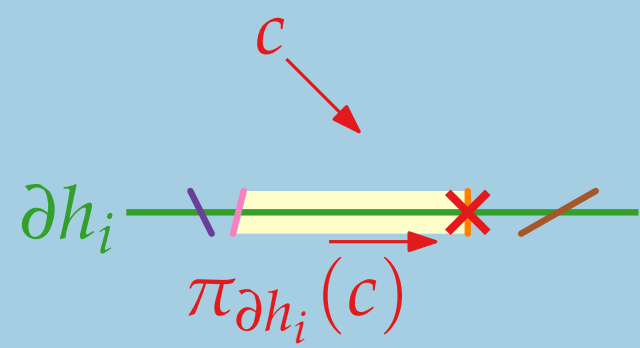


Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

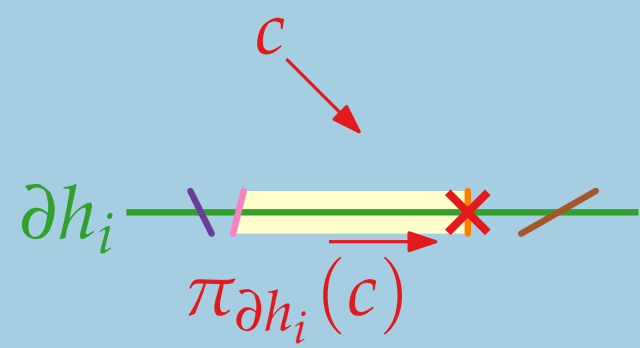


Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n



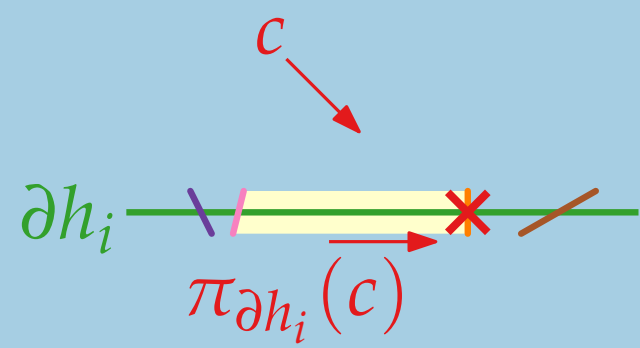
w-c running time:

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n



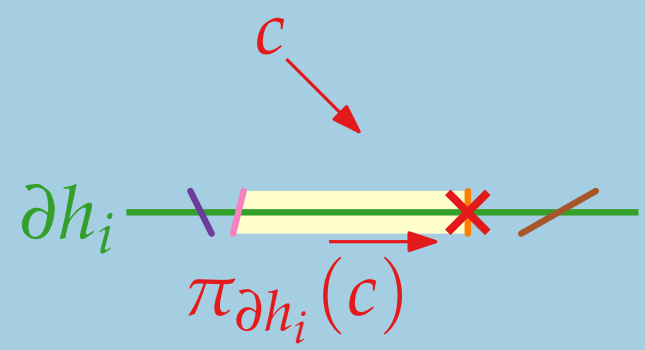
w-c running time:

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$ $O(1)$
return v_n



$O(1)$

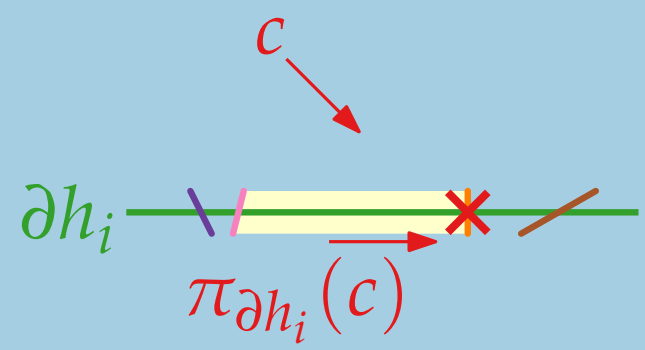
w-c running time:

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$ $O(i)$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$ $O(1)$
return v_n



$O(1)$

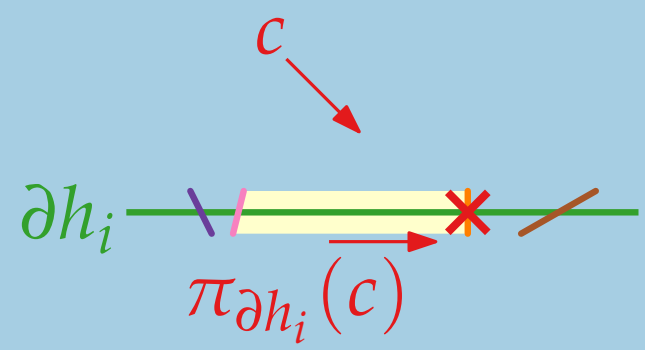
w-c running time:

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$ $O(i)$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$ $O(1)$
return v_n



$O(1)$

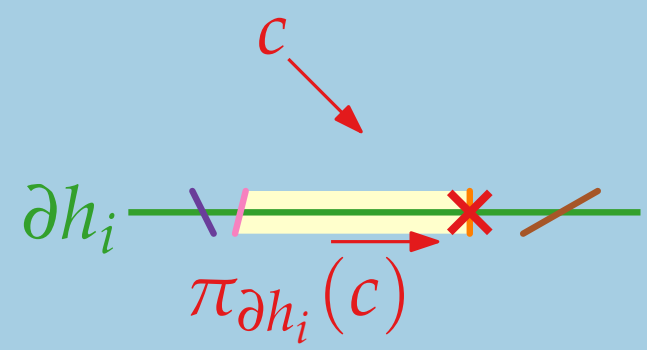
w-c running time:
$T(n) = \sum_{i=1}^{n} O(i) =$

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$ $O(i)$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$ $O(1)$
return v_n



$O(1)$

w-c running time:
$T(n) = \sum_{i=1}^{n} O(i) =$
$= O(n^2)$:-(

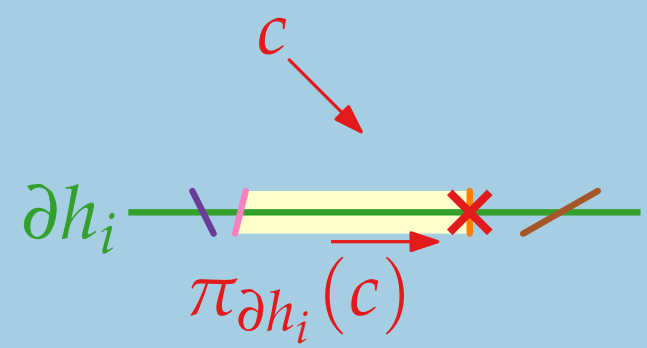
Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

Randomized

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$ $O(i)$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$ $O(1)$
return v_n



$O(1)$

w-c running time:

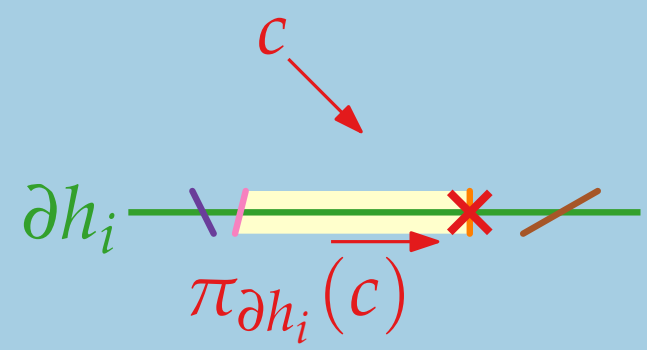
$T(n) = \sum_{i=1}^{n} O(i) =$
$= O(n^2)$:-(

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

Randomized

2DBoundedLP(H, c, m_1, m_2)
  compute random permutation of H
  $H_0 = \{m_1, m_2\}$
  $v_0 \leftarrow$ corner of $m_1 \cap m_2$
  for $i \leftarrow 1$ **to** n **do**
    if $v_{i-1} \in h_i$ **then**
      $v_i \leftarrow v_{i-1}$
    else
      $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$ $O(i)$
      if $v_i =$ nil **then**
        return nil
    $H_i = H_{i-1} \cup \{h_i\}$ $O(1)$
  return v_n



$O(1)$

w-c running time:

$T(n) = \sum_{i=1}^{n} O(i) =$

$= O(n^2)$:-(

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$\mathbf{E}[T_{2d}(n)] =$

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\mathbf{E}[T_{2d}(n)] = \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \end{aligned}$$

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}\left[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)\right] \\ &= O(n) + \sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.
= probability that the optimal solution changes when h_i is removed from H_i.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.
= probability that the optimal solution changes when h_i is removed from H_i.

i.e., when $v_i \in \partial h_i$ and $v_i \in \partial h_j$ for exactly one $j < i$.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.
= probability that the optimal solution changes when h_i is removed from H_i.
$\leq 2/i$.

i.e., when $v_i \in \partial h_i$ and $v_i \in \partial h_j$ for exactly one $j < i$.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.
= probability that the optimal solution changes when h_i is removed from H_i.
$\leq 2/i$. This is independent of the choice of H_i.

i.e., when $v_i \in \partial h_i$ and $v_i \in \partial h_j$ for exactly one $j < i$.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.
= probability that the optimal solution changes when h_i is removed from H_i.
$\leq 2/i$. This is independent of the choice of H_i.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \quad = O(n). \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.
= probability that the optimal solution changes when h_i is removed from H_i.
$\leq 2/i$. This is independent of the choice of H_i. □

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= O(n) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &= O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \quad = O(n). \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.

= probability that the optimal solution changes when h_i is removed from H_i.

$\leq 2/i$. This is independent of the choice of H_i. □

Proof technique: *Backward analysis!*

Alt. for Intersecting Convex Regions

CG: A & A §2

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{MO}}(n) =$

Alt. for Intersecting Convex Regions

CG: A & A §2

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{MO}}(n) = O((n + I)\log n)$,

where I = # intersection points.

Alt. for Intersecting Convex Regions

CG: A & A §2

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{MO}}(n) = O((n + I)\log n)$,

where I = # intersection points.

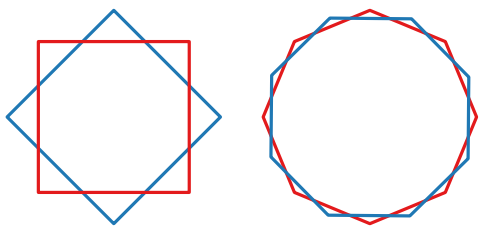
here: $I \leq$

Alt. for Intersecting Convex Regions

CG: A & A §2

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{MO}}(n) = O((n + I)\log n)$,



where I = # intersection points.

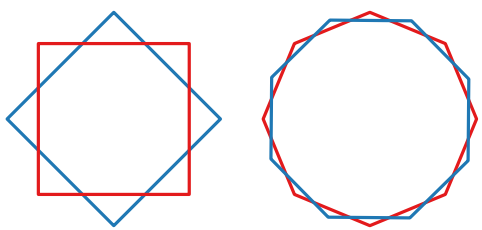
here: $I \leq n \rightarrow O(n \log n)$ for ICR

Alt. for Intersecting Convex Regions

CG: A & A §2

Use sweep-line alg. for map overlay (line-segment intersections) !

Running time $T_{\mathrm{MO}}(n) = O((n + I)\log n)$,



where I = # intersection points.

here: $I \leq n \rightarrow O(n \log n)$ for ICR

Running time $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

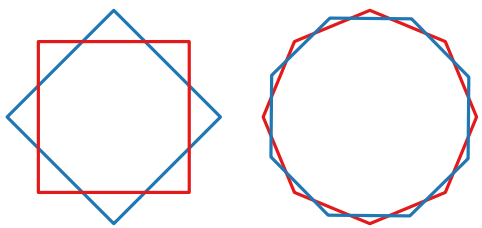
$\leq$

Alt. for Intersecting Convex Regions

CG: A & A §2

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{MO}}(n) = O((n + I)\log n)$,



where I = # intersection points.

here: $I \leq n \rightarrow O(n \log n)$ for ICR

Running time $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

$\leq 2T_{\mathrm{IH}}(n/2) + O(n \log n)$

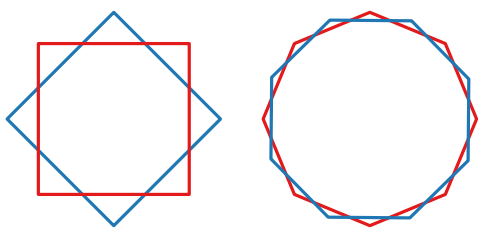
$\in$

Alt. for Intersecting Convex Regions

CG: A & A §2

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{MO}}(n) = O((n + I)\log n)$,



where I = # intersection points.

here: $I \leq n \rightarrow O(n \log n)$ for ICR

Running time $T_{\text{IH}}(n) = 2T_{\text{IH}}(n/2) + T_{\text{ICR}}(n)$

$\leq 2T_{\text{IH}}(n/2) + O(n \log n)$

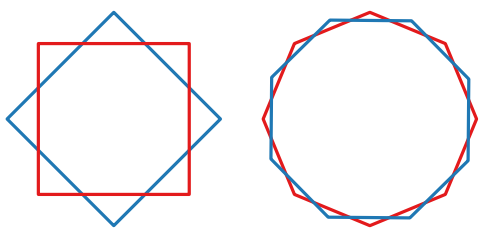
$\in O(n \log^2 n)$

Alt. for Intersecting Convex Regions

CG: A & A §2

Use sweep-line alg. for map overlay (line-segment intersections) !

Running time $T_{\mathrm{MO}}(n) = O((n+I)\log n)$,



where I = # intersection points.

here: $I \leq n \rightarrow O(n \log n)$ for ICR

Running time $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

$$\leq 2T_{\mathrm{IH}}(n/2) + O(n \log n)$$

$$\in O(n \log^2 n)$$

As this is more general, it is unsurprisingly worse ... *

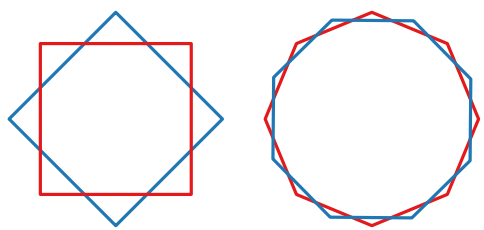
* it can happen sometimes that general algorithms give optimal runtimes for special cases

Alt. for Intersecting Convex Regions

CG: A & A §2

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{MO}}(n) = O((n + I)\log n)$,



where I = # intersection points.

here: $I \leq n \rightarrow O(n \log n)$ for ICR

Running time $T_{\text{IH}}(n) = 2T_{\text{IH}}(n/2) + T_{\text{ICR}}(n)$

$\leq 2T_{\text{IH}}(n/2) + O(n \log n)$

$\in O(n \log^2 n)$

As this is more general, it is unsurprisingly worse ... *

$\rightsquigarrow$ Better to use specialized algorithm for intersecting *convex* regions/polygons

* it can happen sometimes that general algorithms give optimal runtimes for special cases