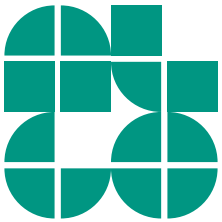
Vorlesung Algorithmische Kartografie

Beschriftung in Dynamischen Karten

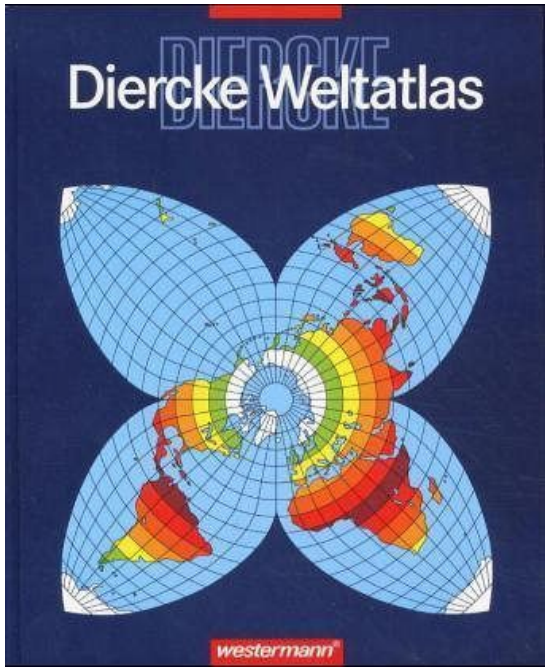
LEHRSTUHL FÜR ALGORITHMIK I · INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

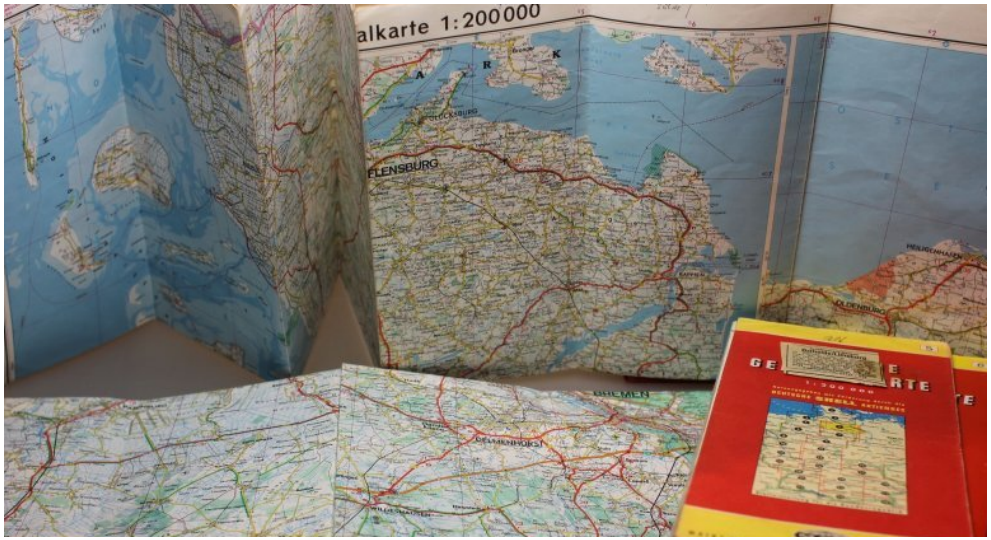
Martin Nöllenburg

04.06.2013

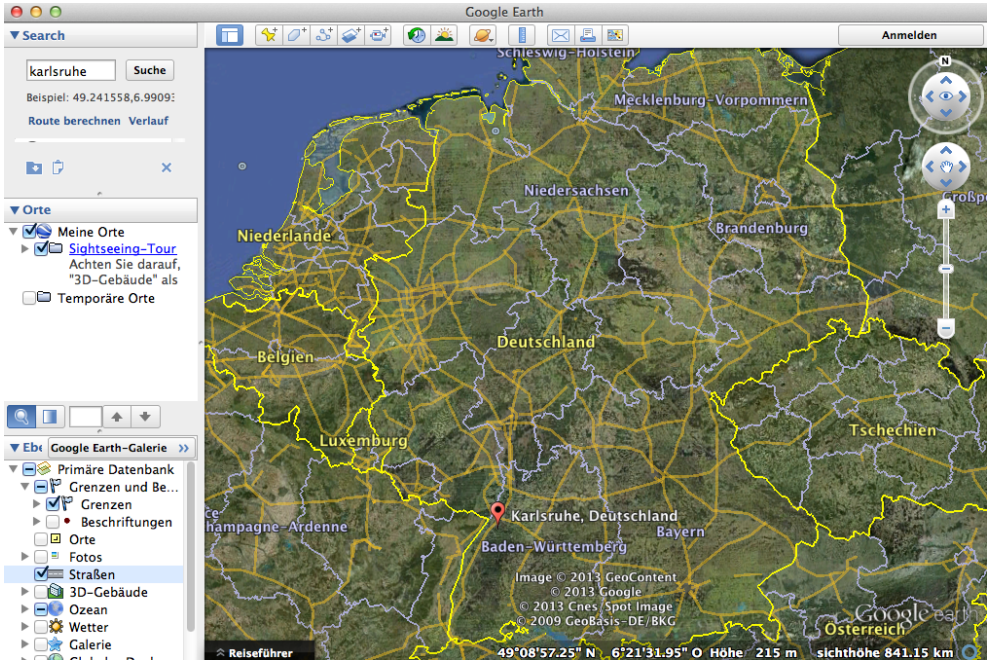


Was ist eine Landkarte?





oder



Die Ära der dynamischen Karten

Die meisten Karten sind heute nicht mehr statisch und allgemein, sondern dynamisch und individuell.

Die Ära der dynamischen Karten

Die meisten Karten sind heute nicht mehr statisch und allgemein, sondern dynamisch und individuell.

Welche Eigenschaften haben dynamische Karten, wie wirkt sich das auf die Beschriftung aus?

Die Ära der dynamischen Karten

Die meisten Karten sind heute nicht mehr statisch und allgemein, sondern dynamisch und individuell.

Kartenansicht bewegt sich kontinuierlich wenn der Nutzer

- zoomt
- verschiebt
- rotiert
- neigt

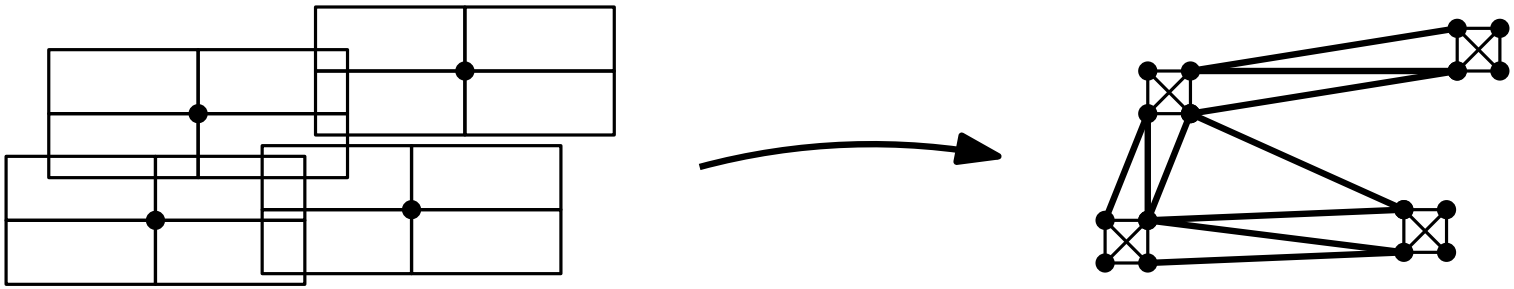
Layout und Beschriftung müssen sich an die dynamische Kartenbewegung anpassen

→ kontinuierliche Generalisierung

→ **kontinuierliche Kartenbeschriftung**

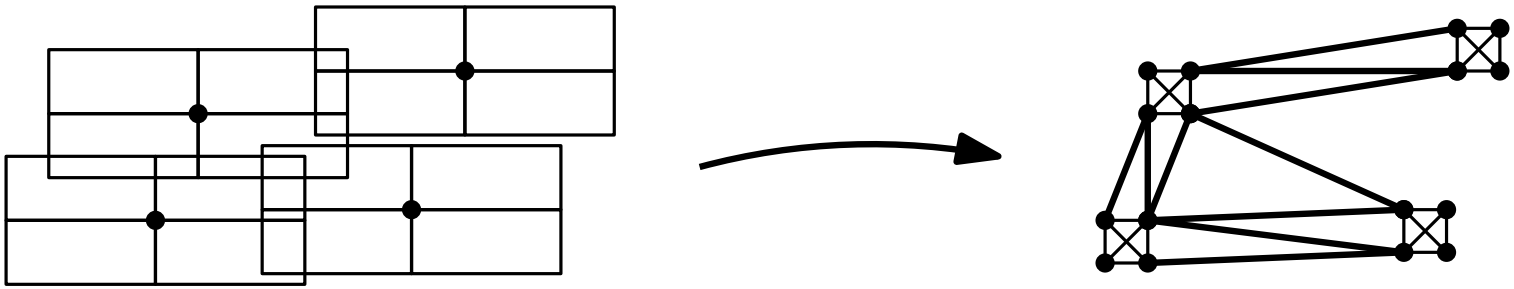
Lassen sich statische Verfahren nutzen?

Die meisten heuristischen **statischen** Ansätze nutzen Konfliktgraph als Modell für Überlappungen und suchen große unabhängige Labelmenge.



Lassen sich statische Verfahren nutzen?

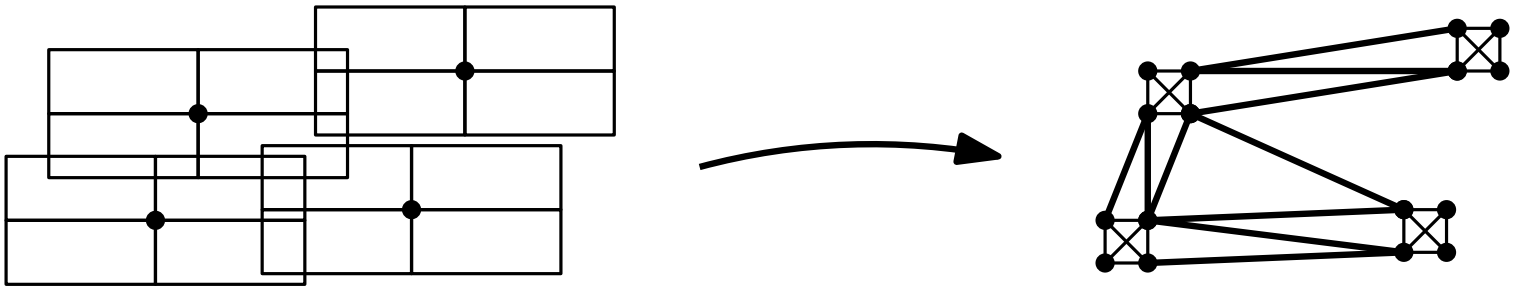
Die meisten heuristischen **statischen** Ansätze nutzen Konfliktgraph als Modell für Überlappungen und suchen große unabhängige Labelmenge.



Übertragung auf **dynamische** Karten: schnelle Algorithmen um jeden Frame einer Animation in Echtzeit zu beschriften.

Lassen sich statische Verfahren nutzen?

Die meisten heuristischen **statischen** Ansätze nutzen Konfliktgraph als Modell für Überlappungen und suchen große unabhängige Labelmenge.



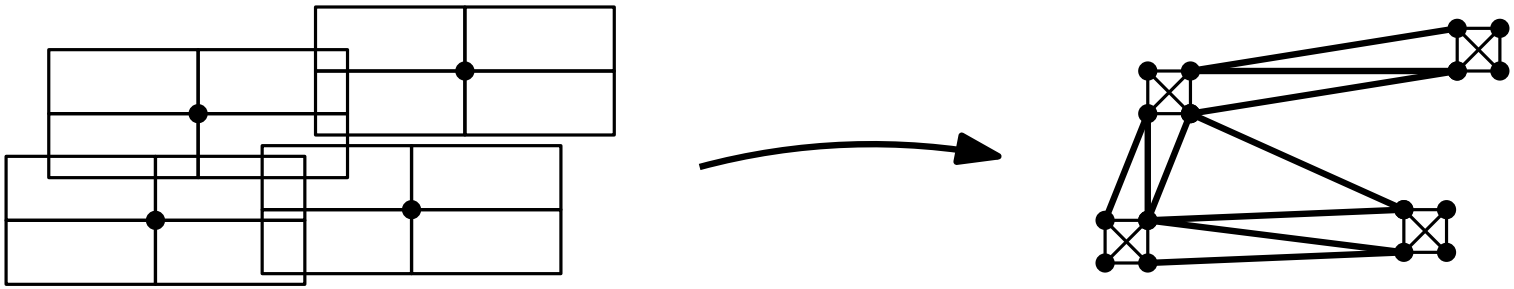
Übertragung auf **dynamische** Karten: schnelle Algorithmen um jeden Frame einer Animation in Echtzeit zu beschriften.

Ansatz 1: Vorberechnung & Abfragen **[Petzold, Gröger, Plümer '03]**

- konstruiere *reaktiven* Konfliktgraph, Abfrage während Interaktion, verwende greedy Verfahren für statische Labelauswahl

Lassen sich statische Verfahren nutzen?

Die meisten heuristischen **statischen** Ansätze nutzen Konfliktgraph als Modell für Überlappungen und suchen große unabhängige Labelmenge.



Übertragung auf **dynamische** Karten: schnelle Algorithmen um jeden Frame einer Animation in Echtzeit zu beschriften.

Ansatz 1: Vorberechnung & Abfragen **[Petzold, Gröger, Plümer '03]**

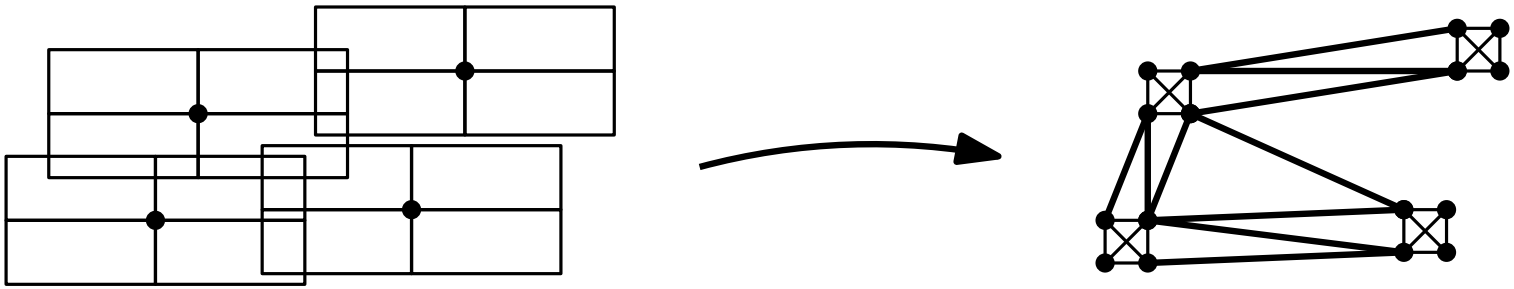
- konstruiere *reaktiven* Konfliktgraph, Abfrage während Interaktion, verwende greedy Verfahren für statische Labelauswahl

Ansatz 2: schnelle on-the-fly Berechnung

- berechne lokalen Konfliktgraph, schätze Positionskosten, suche greedy günstigste Position **[Mote '07]**
- partikelbasierte Konflikterkennung, sequentielle Platzierung nach absteigender Positionspräferenz **[Luboschik, Schumann, Cords '08]**

Lassen sich statische Verfahren nutzen?

Die meisten heuristischen **statischen** Ansätze nutzen Konfliktgraph als Modell für Überlappungen und suchen große unabhängige Labelmenge.



Übertragung auf **dynamische** Karten: schnelle Algorithmen um jeden Frame einer Animation in Echtzeit zu beschriften.

Ansatz 1: Vorberechnung & Abfragen **[Petzold, Gröger, Plümer '03]**

- konstruiere *reaktiven* Konfliktgraph, Abfrage während Interaktion, verwende greedy Verfahren für statische Labelauswahl

Ist das Problem damit gelöst?

Ansatz 2: schnelle on-the-fly Berechnung

- berechne lokalen Konfliktgraph, schätze Positionskosten, suche greedy günstigste Position **[Mote '07]**
- partikelbasierte Konflikterkennung, sequentielle Platzierung nach absteigender Positionspräferenz **[Luboschik, Schumann, Cords '08]**

Neue Verfahren sind notwendig

Karlsruhe Institute of Technology

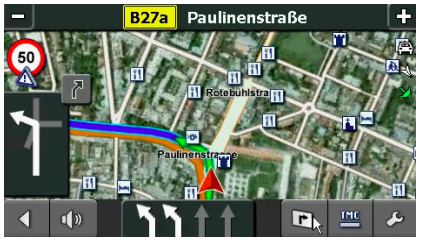
Die bisherigen Ansätze sind zwar schnell genug, aber die resultierenden Kartenanimationen sind oft unbefriedigend.

- jeder Frame wird unabhängig von Nachbarframes beschriftet
- Label neigen zu Flackern und Sprüngen
- **zeitliche Kohärenz** oder **Konsistenz** wird nicht betrachtet

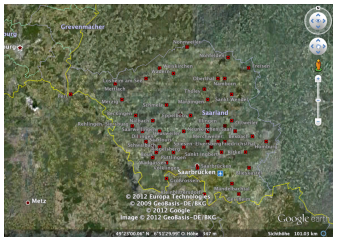
Neue Verfahren sind notwendig

Die bisherigen Ansätze sind zwar schnell genug, aber die resultierenden Kartenanimationen sind oft unbefriedigend.

- jeder Frame wird unabhängig von Nachbarframes beschriftet
- Label neigen zu Flackern und Sprüngen
- **zeitliche Kohärenz** oder **Konsistenz** wird nicht betrachtet



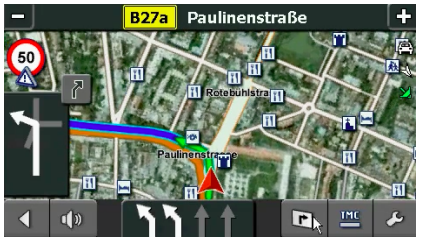
Aktuelle reale Beispiele zeigen noch immer negative Effekte!



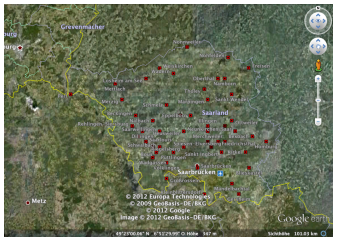
Neue Verfahren sind notwendig

Die bisherigen Ansätze sind zwar schnell genug, aber die resultierenden Kartenanimationen sind oft unbefriedigend.

- jeder Frame wird unabhängig von Nachbarframes beschriftet
- Label neigen zu Flackern und Sprüngen
- **zeitliche Kohärenz** oder **Konsistenz** wird nicht betrachtet



Aktuelle reale Beispiele zeigen noch immer negative Effekte!



Konsistenzkriterien

[Been, Daiches, Yap '06]

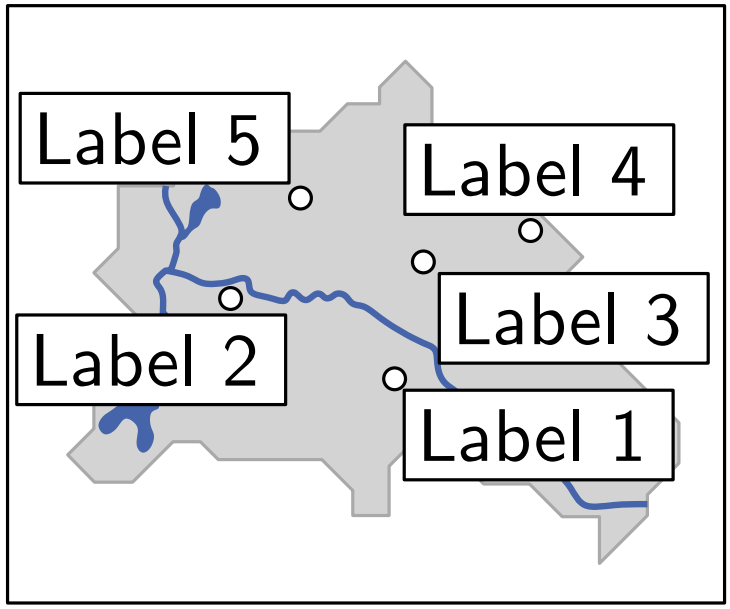
- Label sollen beim Vergrößern nicht verschwinden und beim Verkleinern nicht auftauchen (Monotonität)
- sichtbare Label sollen Position und Größe kontinuierlich ändern
- Platzierung und Auswahl der Label soll eine Funktion des Kartenzustands sein und nicht von der Historie abhängen
- feste Labelgröße auf dem Bildschirm

1) Beschriftung für dynamisches Zoomen

[Been et al. 2010]

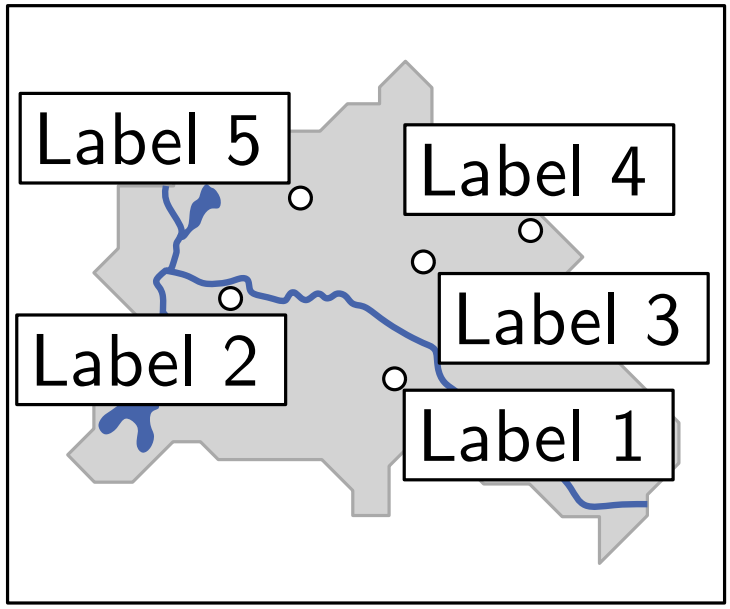
Label in Karten mit Zoomfunktion

großer Maßstab

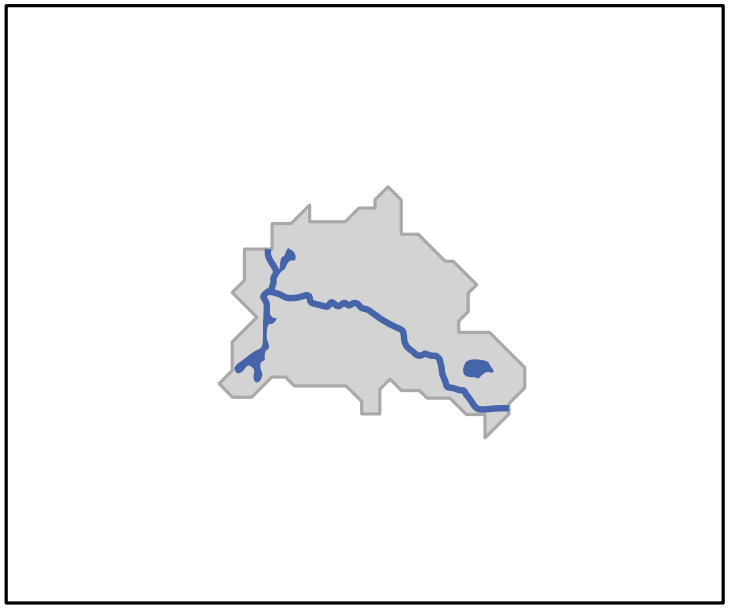


Label in Karten mit Zoomfunktion

großer Maßstab

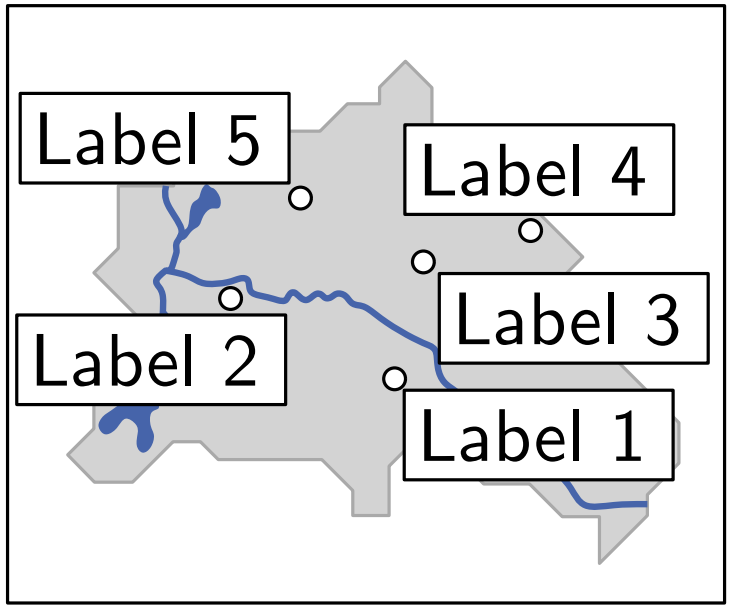


kleiner Maßstab

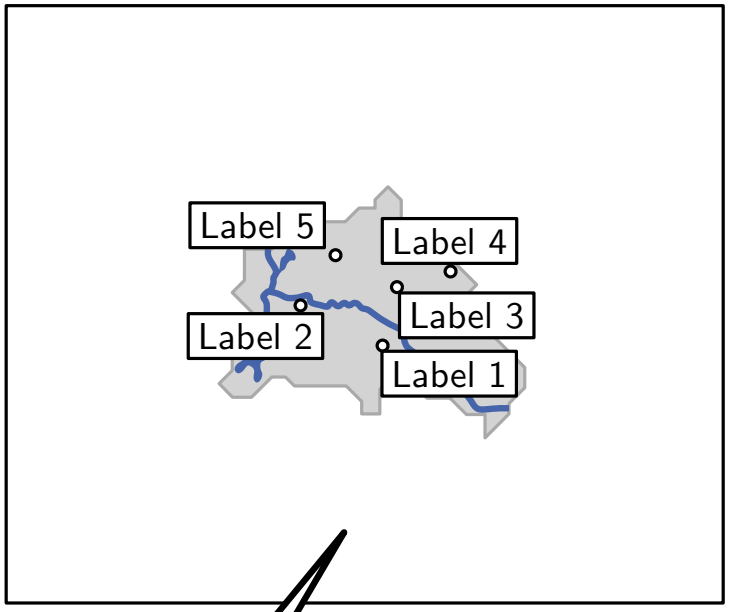


Label in Karten mit Zoomfunktion

großer Maßstab



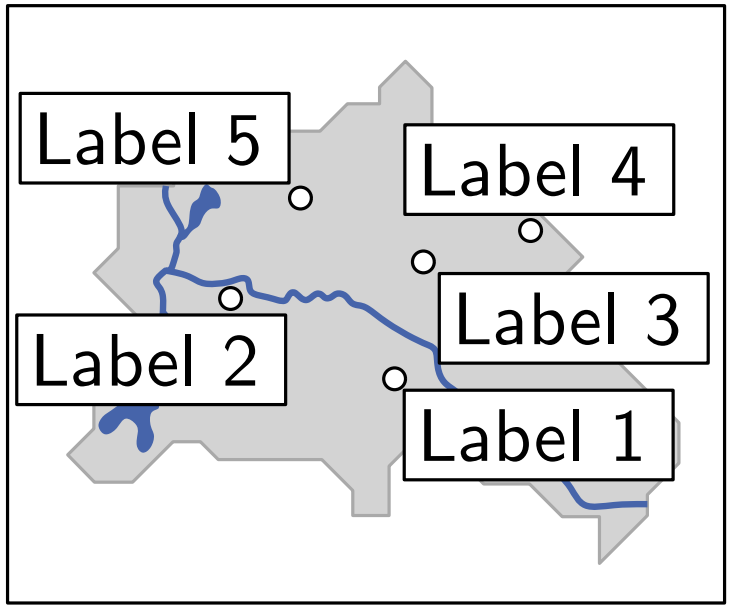
kleiner Maßstab



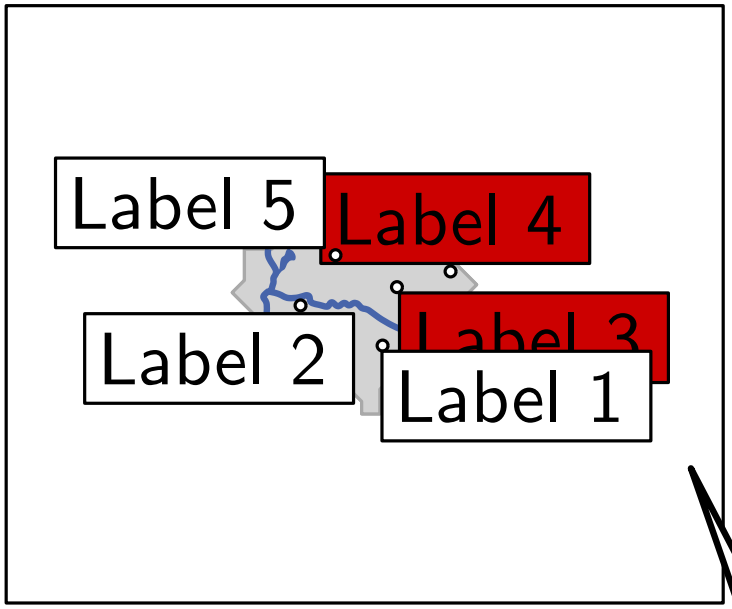
grafisches Zoomen erzeugt kleine Label

Label in Karten mit Zoomfunktion

großer Maßstab



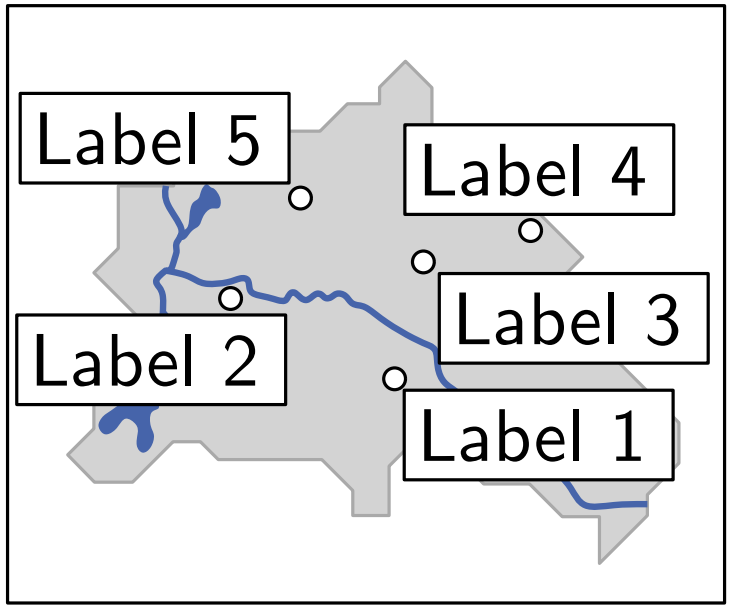
kleiner Maßstab



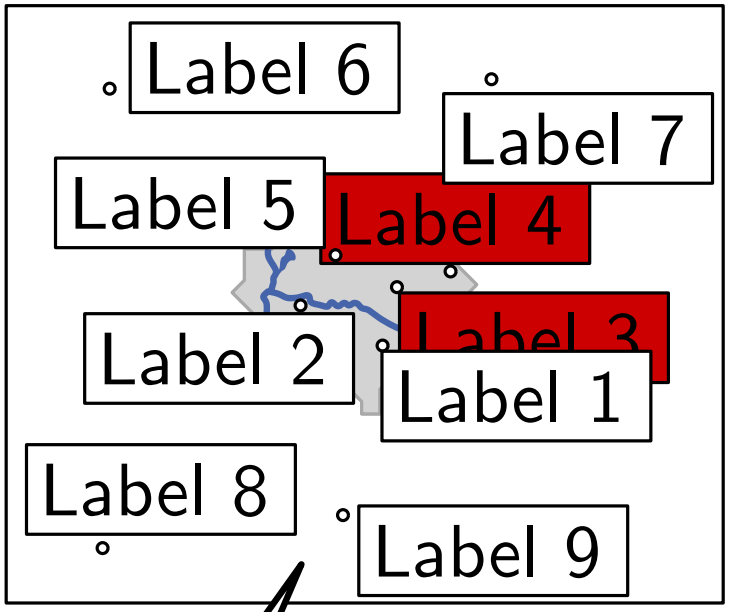
identische Labelgröße erzeugt neue Konflikte

Label in Karten mit Zoomfunktion

großer Maßstab



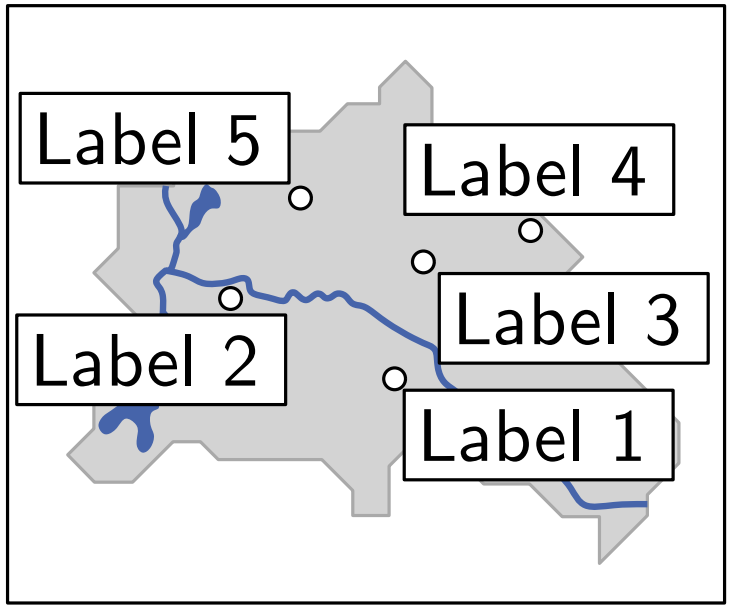
kleiner Maßstab



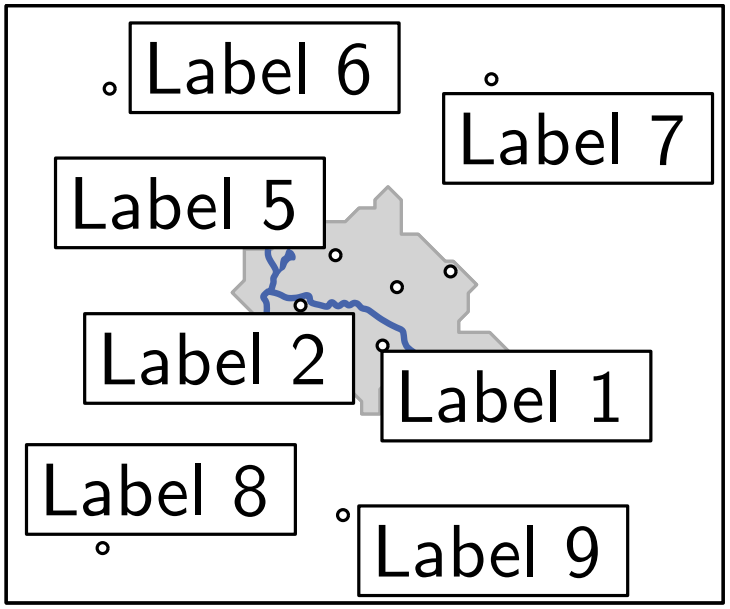
neue Punkte kommen hinzu

Label in Karten mit Zoomfunktion

großer Maßstab



kleiner Maßstab

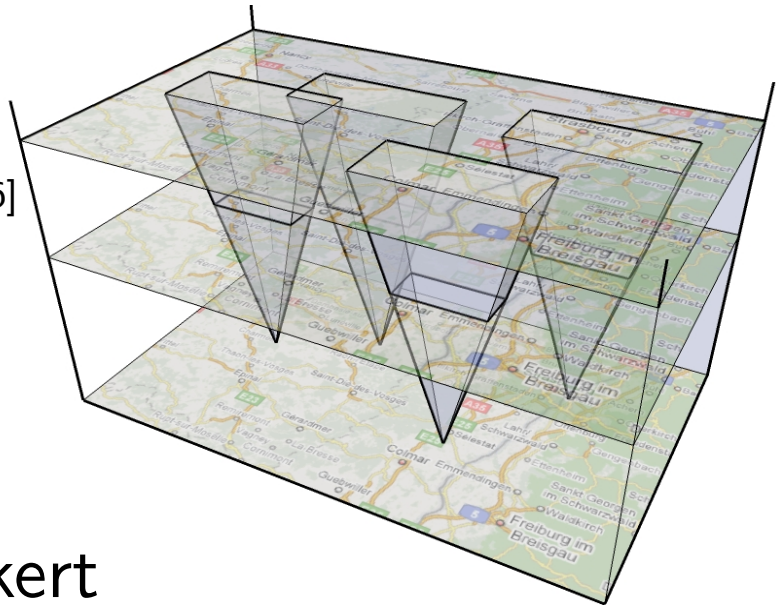


⇒ beim Herauszoomen wachsen Label relativ zur Karte und neue Konflikte müssen aufgelöst werden, ohne dass Label springen oder flackern

3D Modell für dynamisches Beschriften

Dynamisches Beschriftungsmodell

[Been, Daiches, Yap '06]



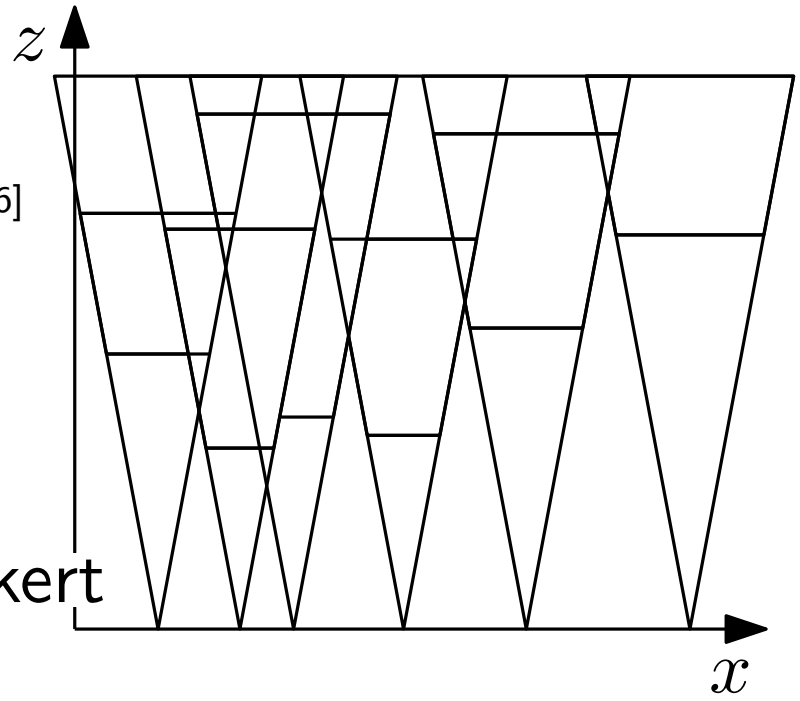
- (inverser) Maßstab auf z-Achse
- horizontale Scheibe bei $z = z_0$ liefert Karte in Maßstab $1/z_0$
- jedes Label an festem Punkt verankert

  $\Rightarrow$ **kein Springen**

3D Modell für dynamisches Beschriften

Dynamisches Beschriftungsmodell

[Been, Daiches, Yap '06]

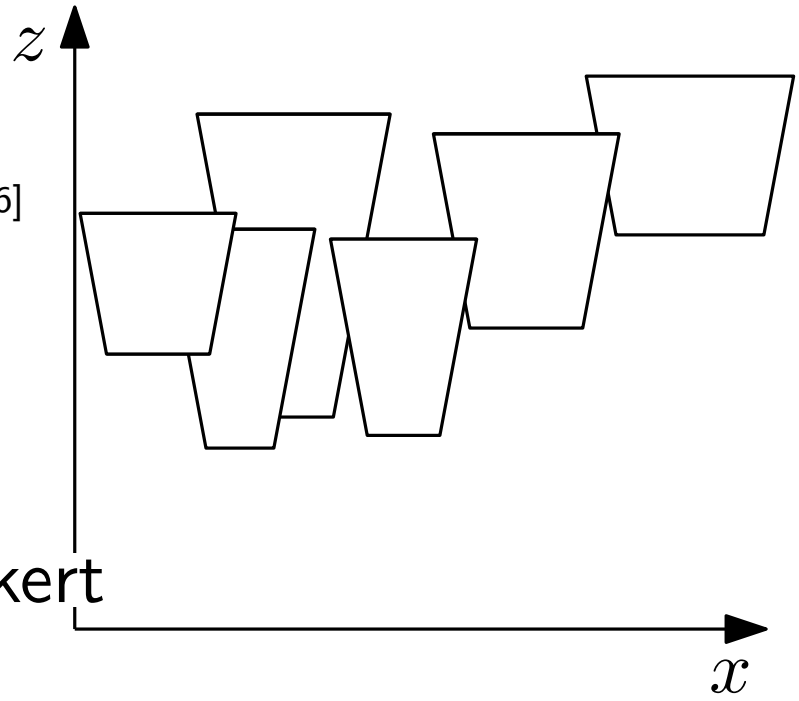


- (inverser) Maßstab auf z-Achse
- horizontale Scheibe bei $z = z_0$ liefert Karte in Maßstab $1/z_0$
- jedes Label an festem Punkt verankert
  $\Rightarrow$ **kein Springen**
- verfügbares Maßstabsintervall S_L für jedes L

3D Modell für dynamisches Beschriften

Dynamisches Beschriftungsmodell

[Been, Daiches, Yap '06]

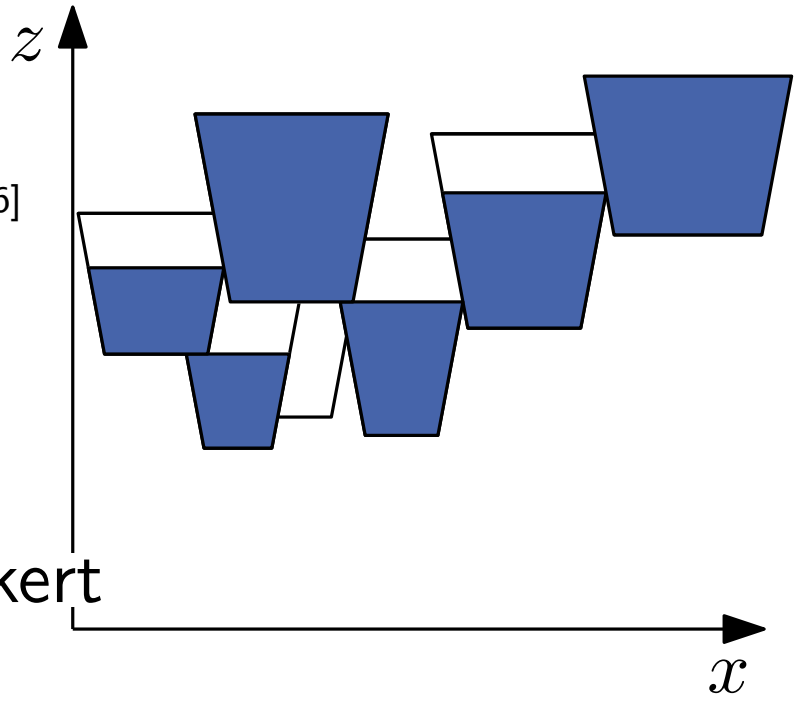


- (inverser) Maßstab auf z-Achse
- horizontale Scheibe bei $z = z_0$ liefert Karte in Maßstab $1/z_0$
- jedes Label an festem Punkt verankert
  $\Rightarrow$ **kein Springen**
- verfügbares Maßstabsintervall S_L für jedes L

3D Modell für dynamisches Beschriften

Dynamisches Beschriftungsmodell

[Been, Daiches, Yap '06]



- (inverser) Maßstab auf z-Achse
- horizontale Scheibe bei $z = z_0$ liefert Karte in Maßstab $1/z_0$
- jedes Label an festem Punkt verankert
  $\Rightarrow$ **kein Springen**
- verfügbares Maßstabsintervall S_L für jedes L
- berechne ein **aktives Intervall** $A_L \subseteq S_L$
  $\Rightarrow$ **kein Flackern**
- aktive Intervalle müssen disjunkte Pyramidenstümpfe (bzw. allg. Labelkörper) induzieren $\Rightarrow$ **gültige Beschriftung**
- **Ziel:** maximiere **aktive Gesamthöhe** $\sum_L |A_L|$

Komplexität

Satz 1:

Es ist NP-schwer $\sum_L |A_L|$ zu maximieren, bzw. für geg. $K > 0$ ist es NP-vollständig zu entscheiden ob es eine Beschriftung mit $\sum_L |A_L| \geq K$.* gibt.

* selbst wenn $S_L = [0, z_{\max}]$ für alle Label L!

Komplexität

Satz 1:

Es ist NP-schwer $\sum_L |A_L|$ zu maximieren, bzw. für geg. $K > 0$ ist es NP-vollständig zu entscheiden ob es eine Beschriftung mit $\sum_L |A_L| \geq K$.* gibt.

* selbst wenn $S_L = [0, z_{\max}]$ für alle Label L!

Beweisskizze:

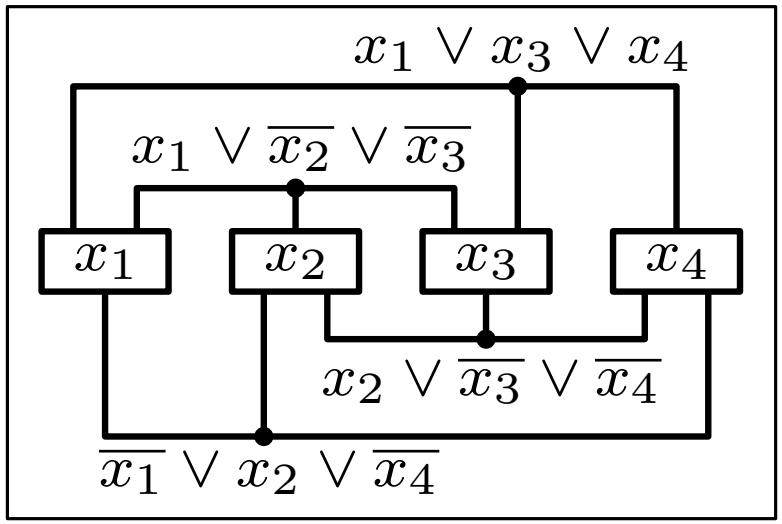
Reduktion von PLANAREM 3-SAT:

(NP-schwer: [Lichtenstein '82])

planare 3-Sat Formel φ

↓

Menge von Punkten & Labeln, $K > 0$:
φ erfüllbar $\Leftrightarrow \max \sum_L |A_L| \geq K$



Komplexität

Satz 1:

Es ist NP-schwer $\sum_L |A_L|$ zu maximieren, bzw. für geg. $K > 0$ ist es NP-vollständig zu entscheiden ob es eine Beschriftung mit $\sum_L |A_L| \geq K$.* gibt.

* selbst wenn $S_L = [0, z_{\max}]$ für alle Label L!

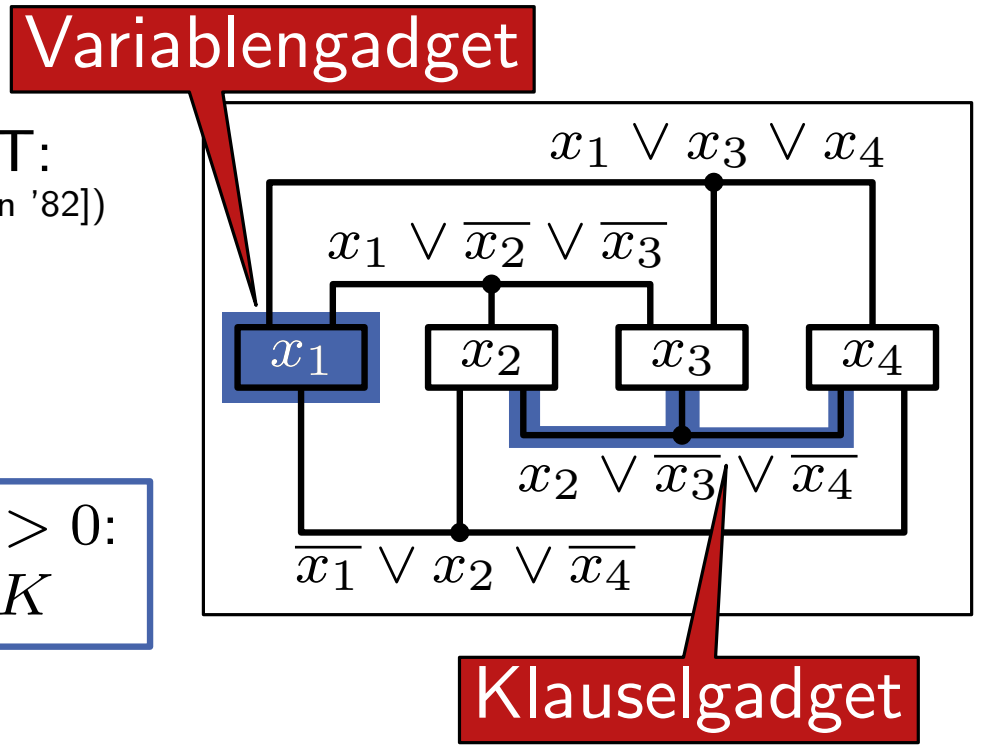
Beweisskizze:

Reduktion von PLANAREM 3-SAT:
(NP-schwer: [Lichtenstein '82])

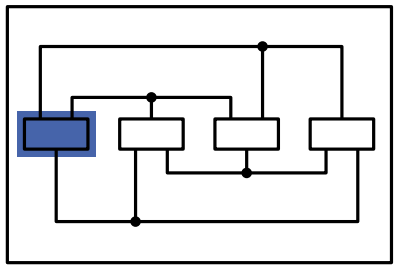
planare 3-Sat Formel φ

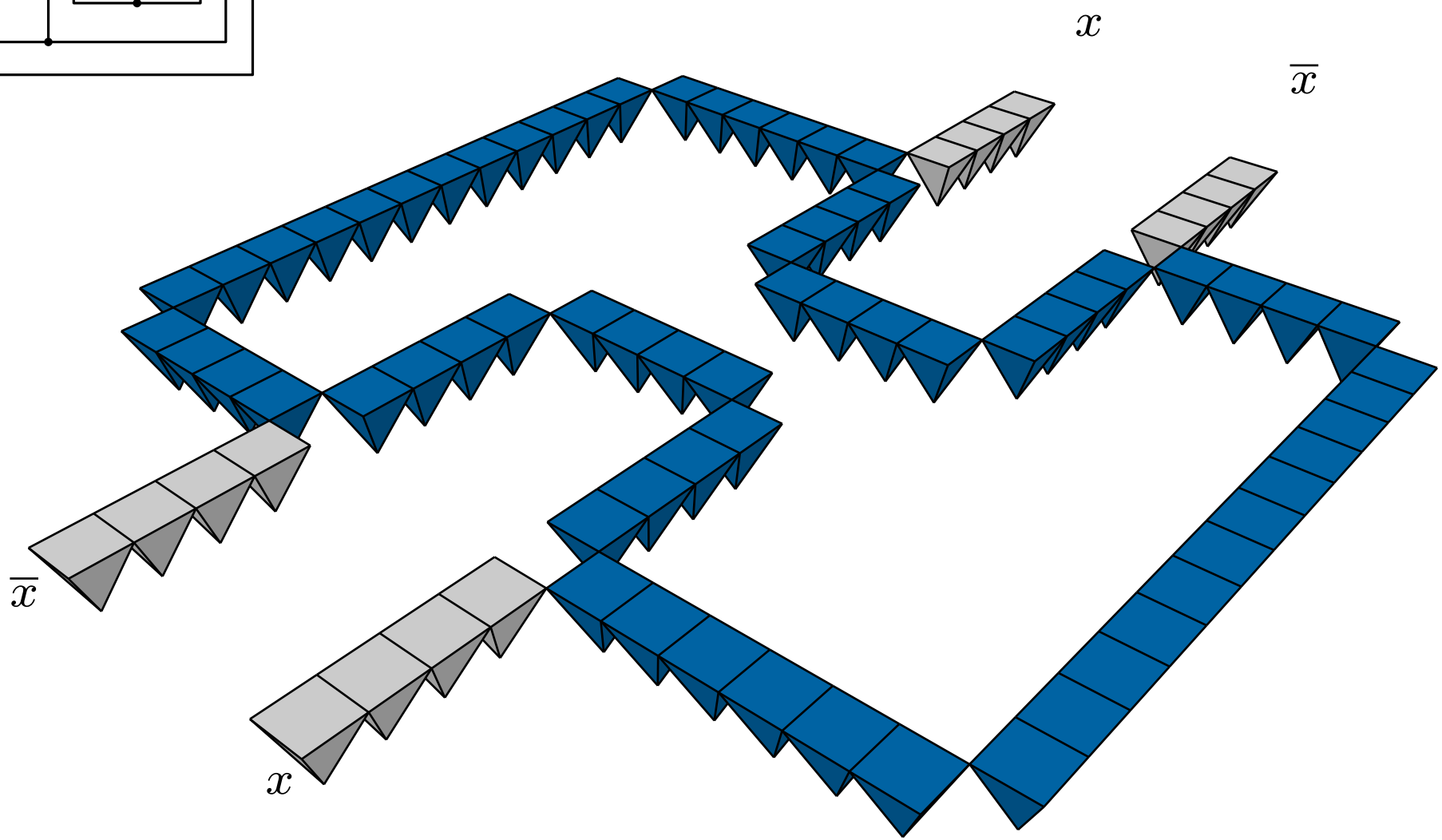
↓

Menge von Punkten & Labeln, $K > 0$:
φ erfüllbar $\Leftrightarrow \max \sum_L |A_L| \geq K$



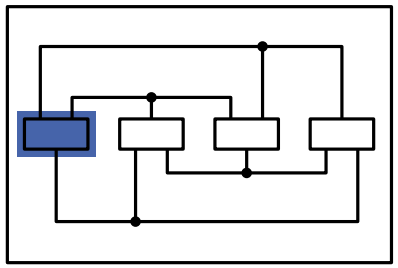
Variablengadget

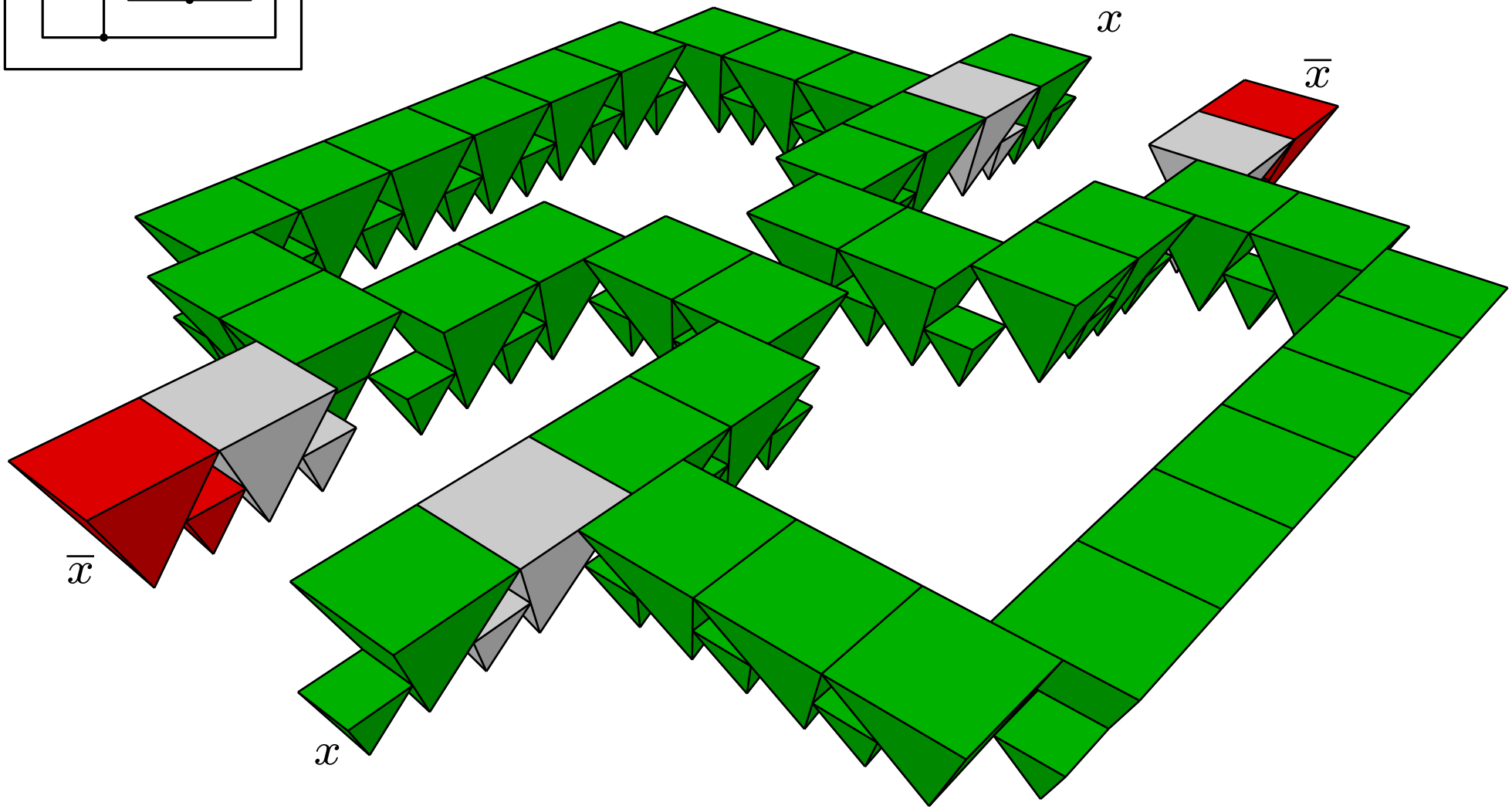




Pyramiden berühren sich bei Höhe $z_{\max}/2$

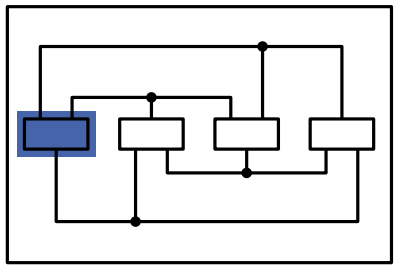
Variablengadget

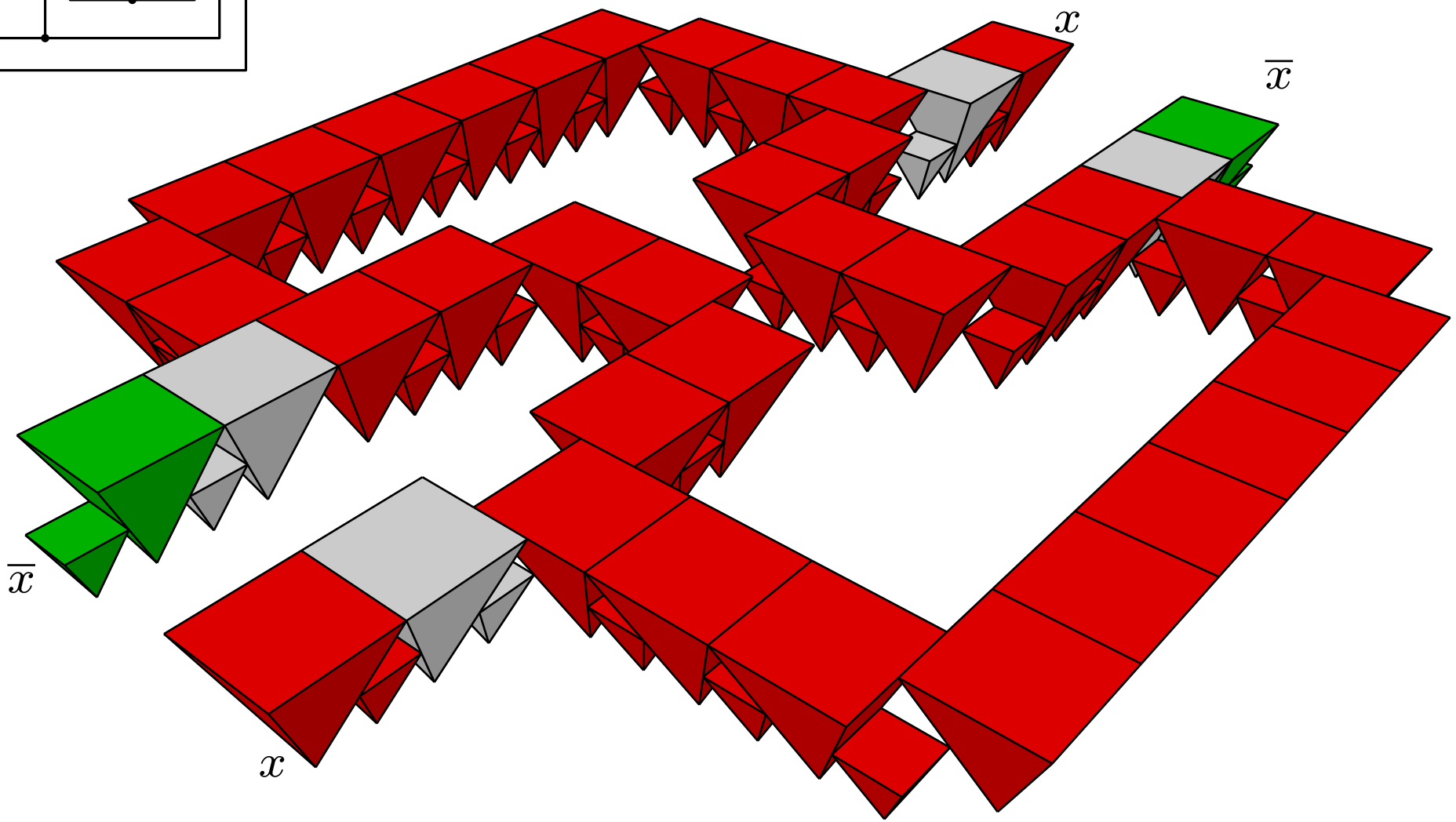




Variable x ist wahr

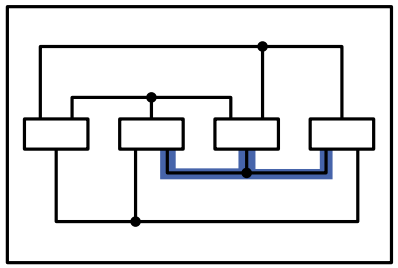
Variablengadget

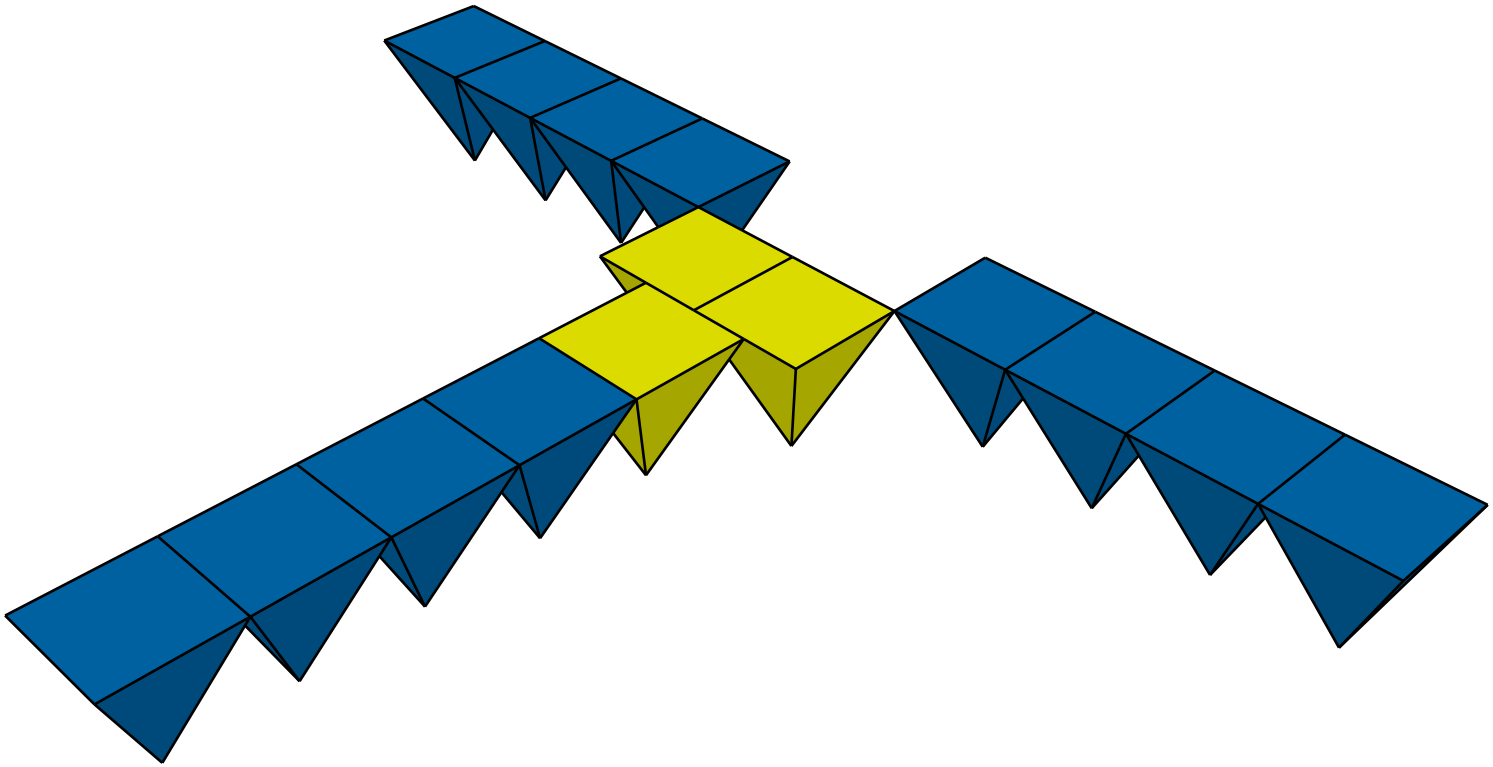




Variable x ist falsch

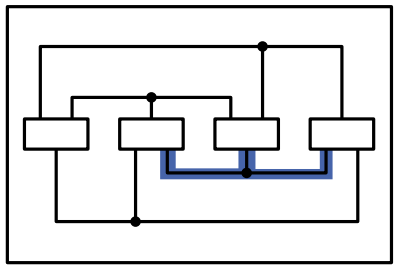
Klauselgadget

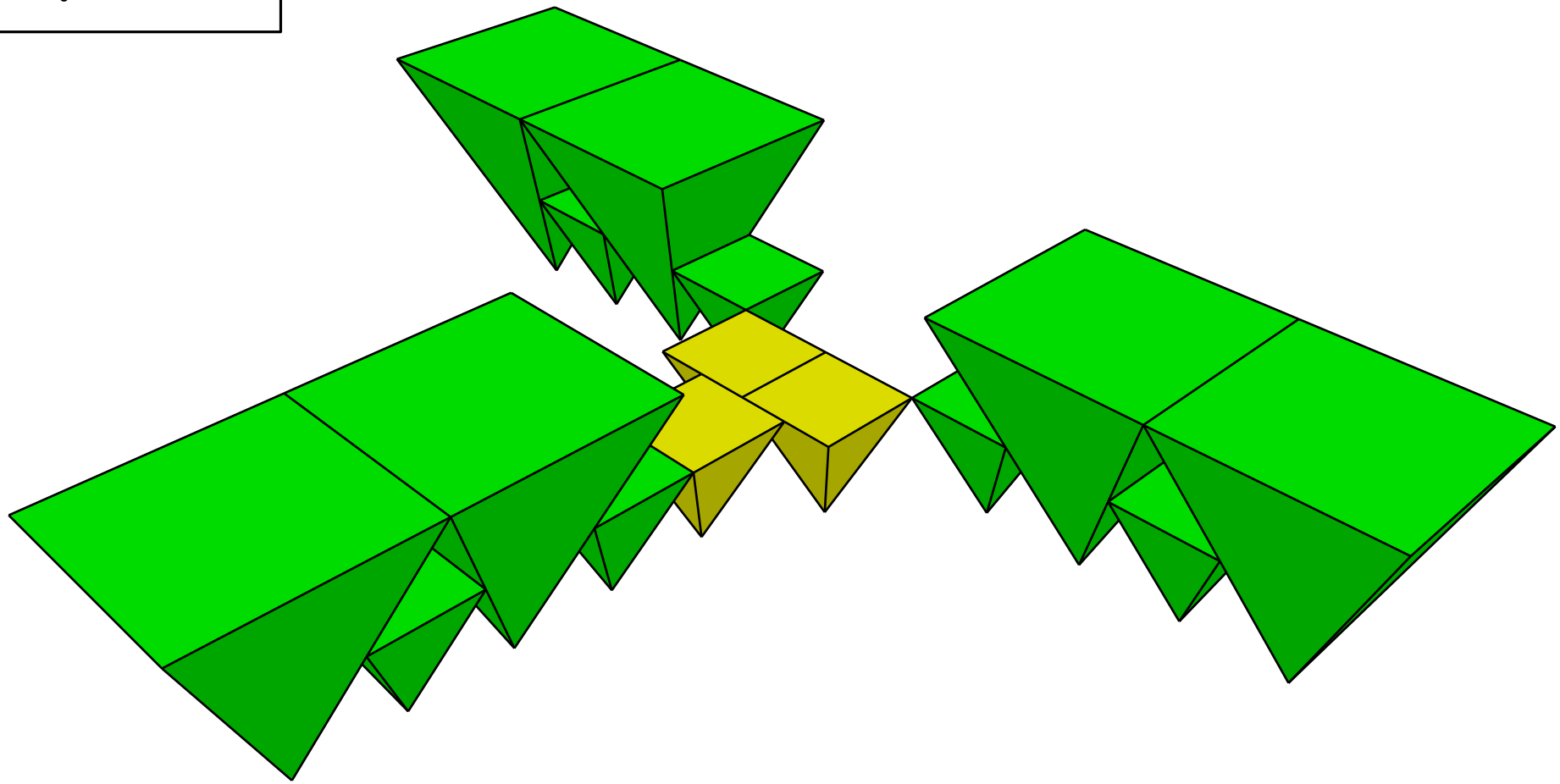




Pyramiden berühren sich bei Höhe $z_{\max}/2$

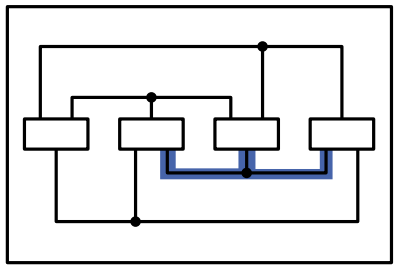
Klauselgadget

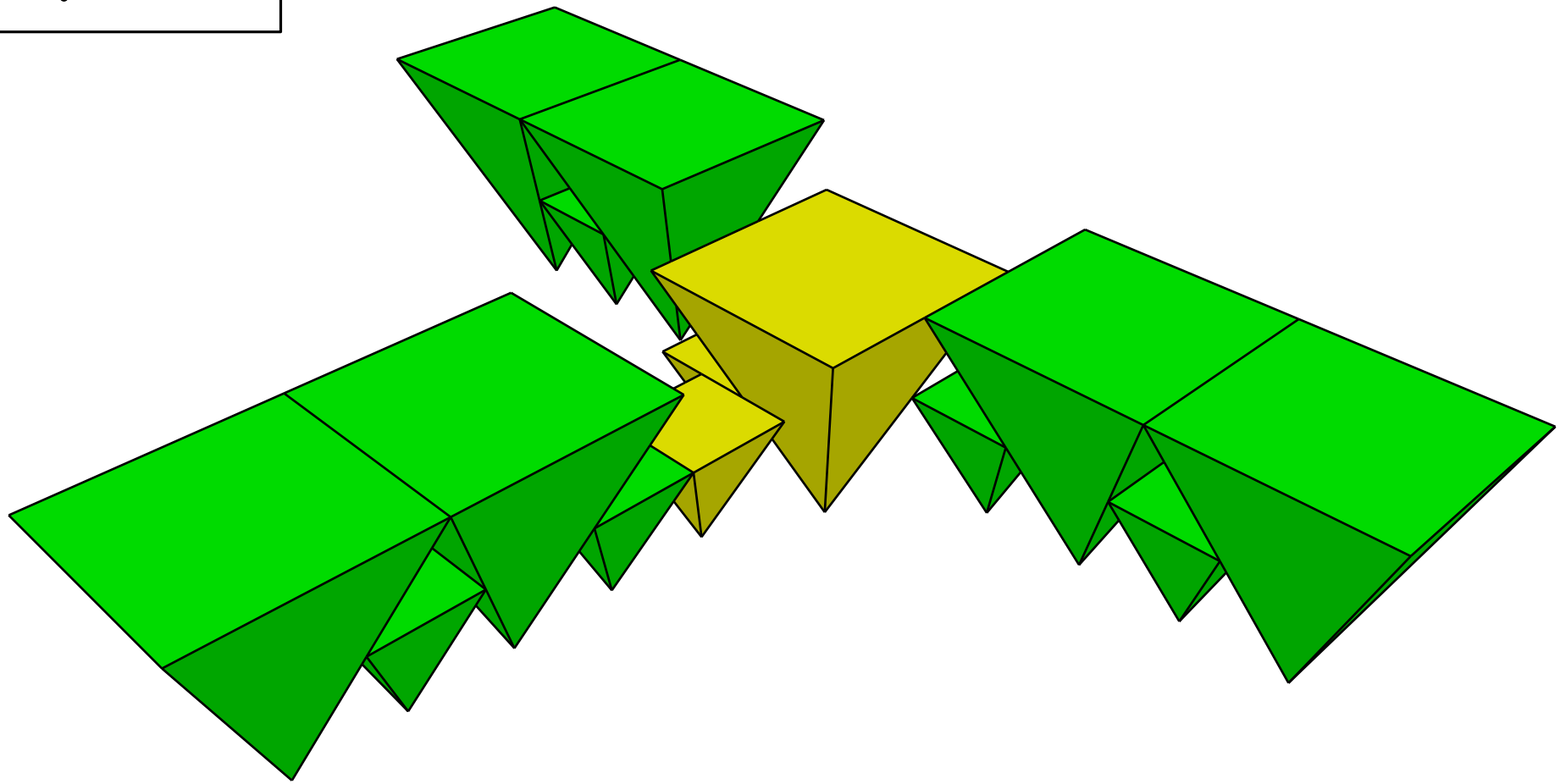




3 Literale sind wahr $\rightarrow$ Klausel-Beitrag: ?

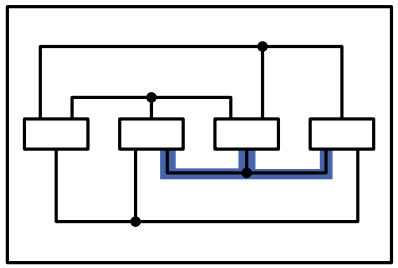
Klauselgadget

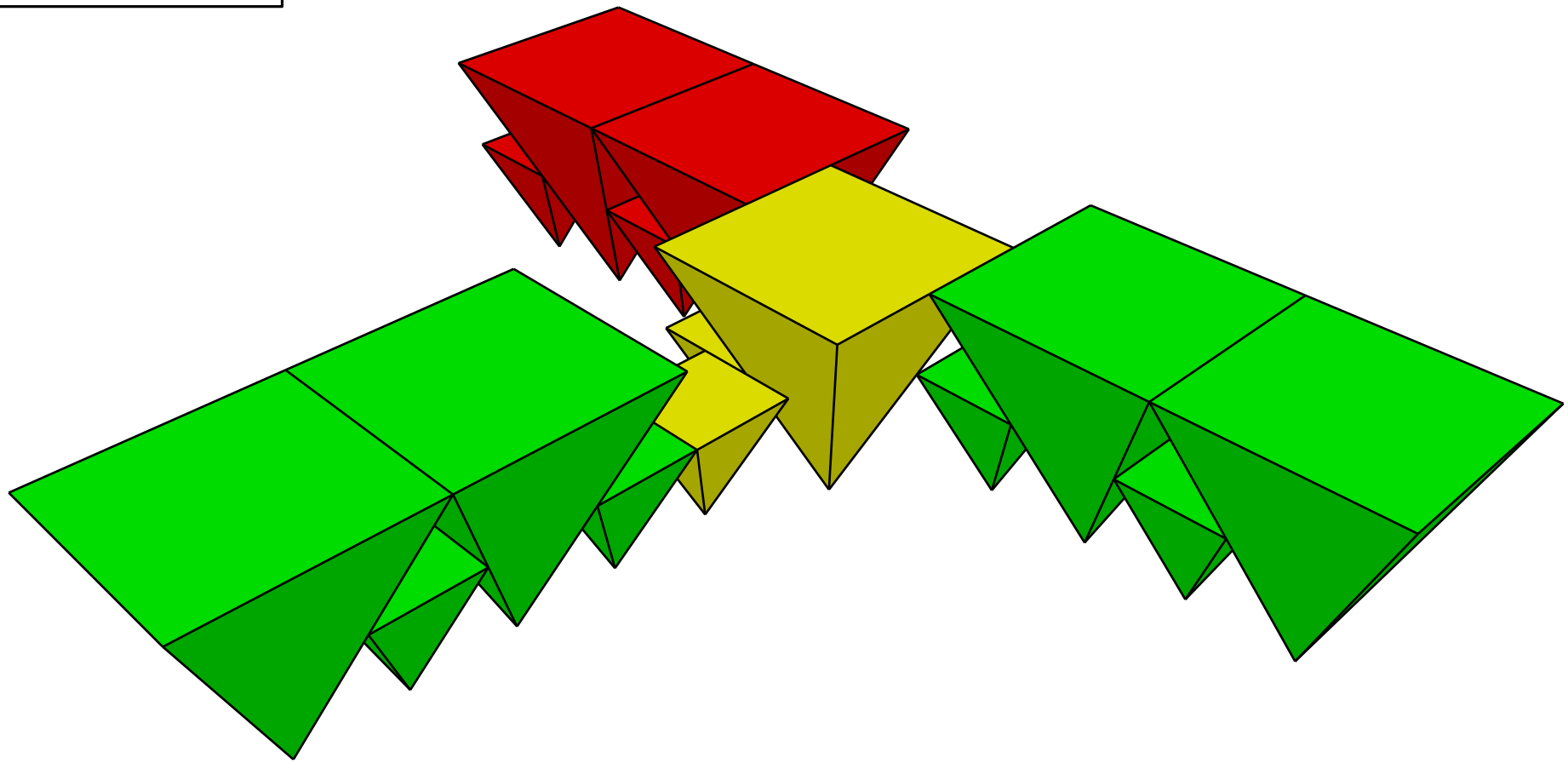




3 Literale sind wahr $\rightarrow$ Klausel-Beitrag: $2 \cdot z_{\max}$

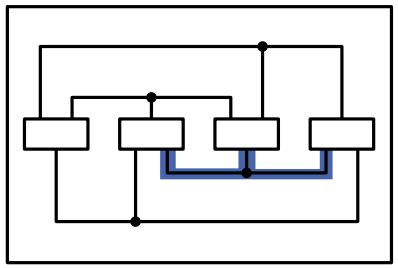
Klauselgadget

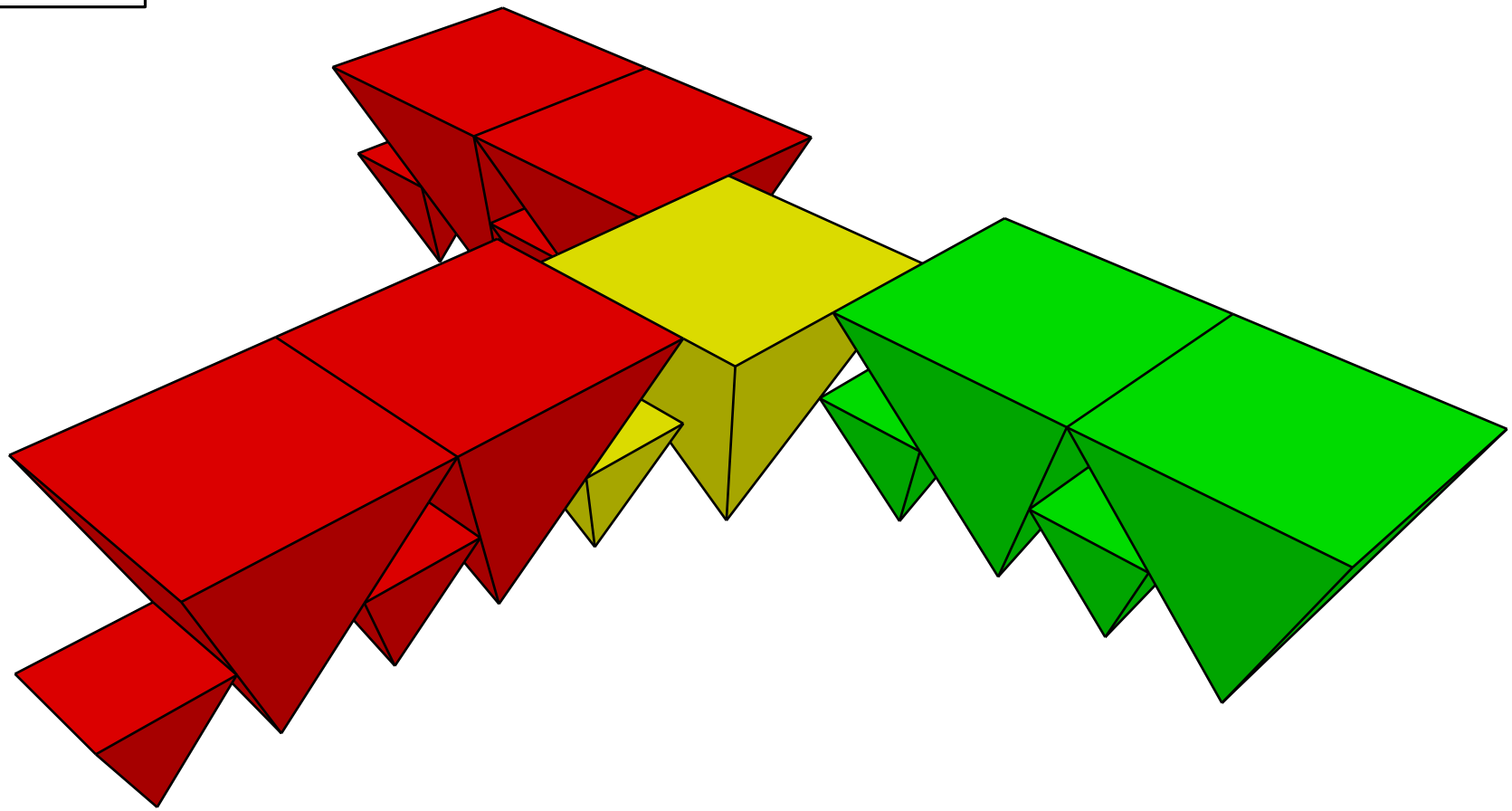




2 Literale sind wahr $\rightarrow$ Klausel-Beitrag: $2 \cdot z_{\max}$

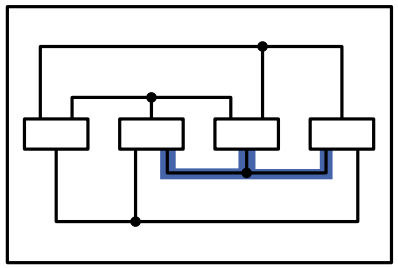
Klauselgadget

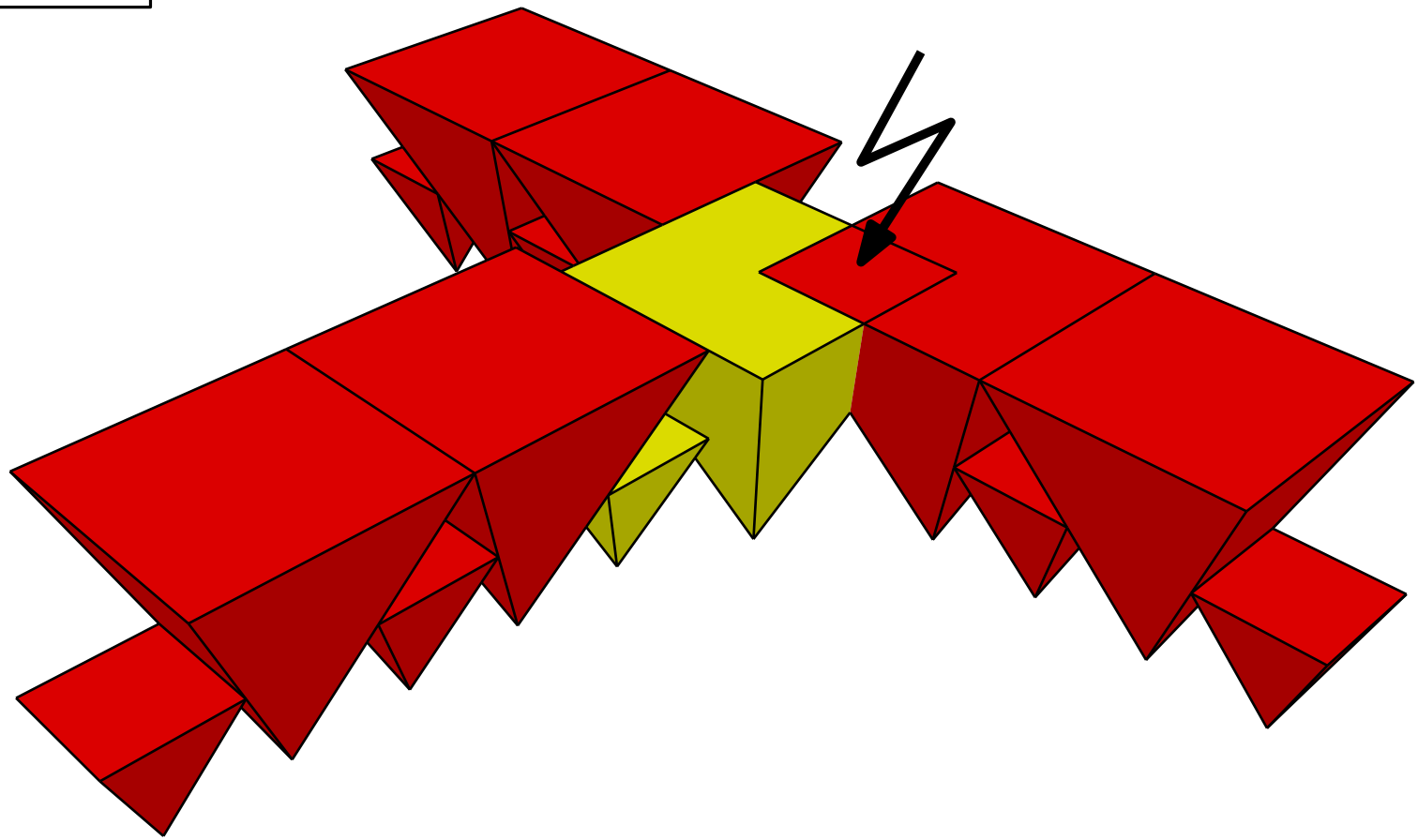




1 Literal ist wahr $\rightarrow$ Klausel-Beitrag: $2 \cdot z_{\max}$

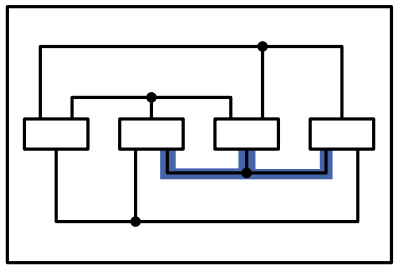
Klauselgadget

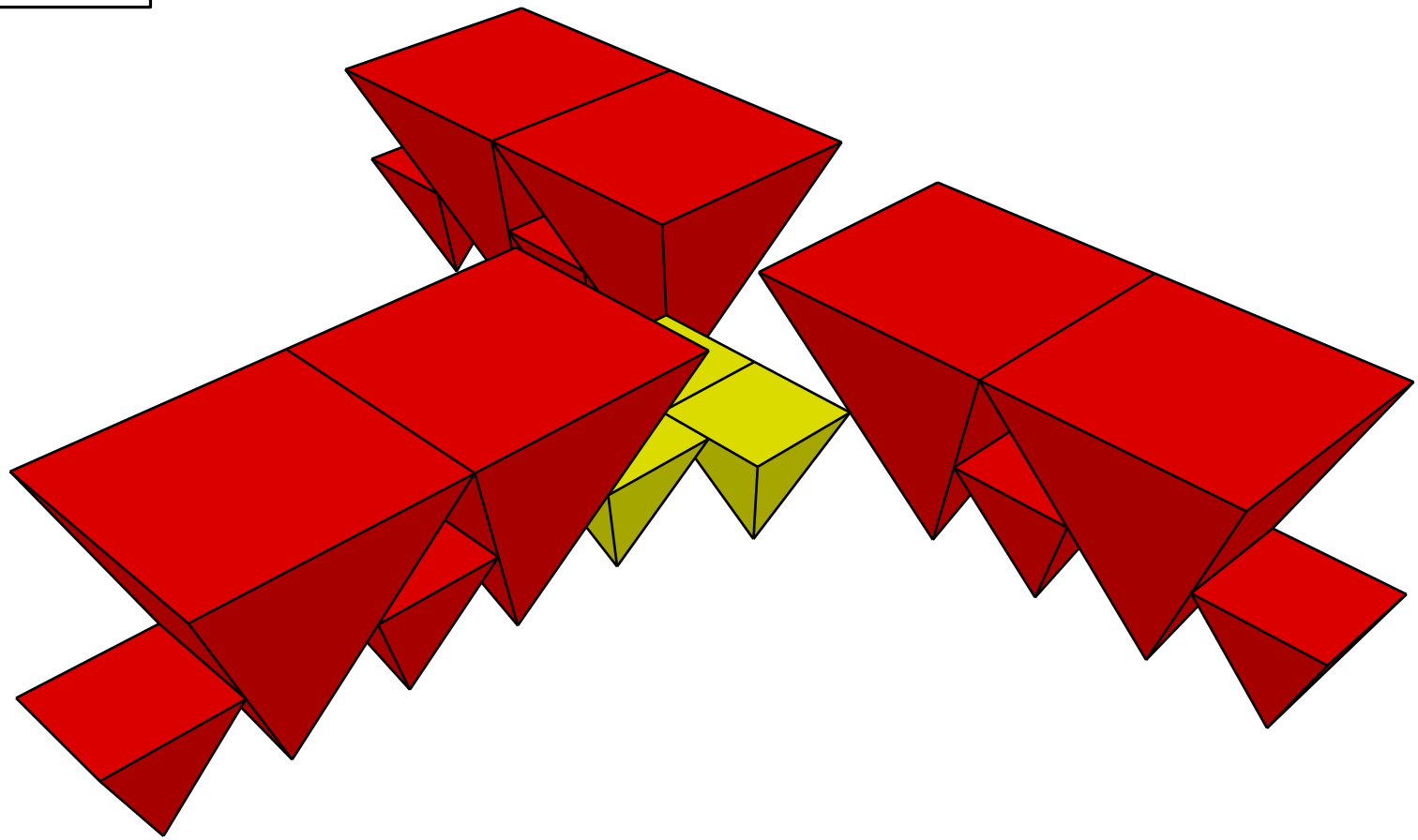




0 Literale sind wahr $\rightarrow$ Klausel-Beitrag: ?

Klauselgadget





0 Literale sind wahr $\rightarrow$ Klausel-Beitrag: $1.5 \cdot z_{\max}$ □

Approximationsalgorithmen

Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

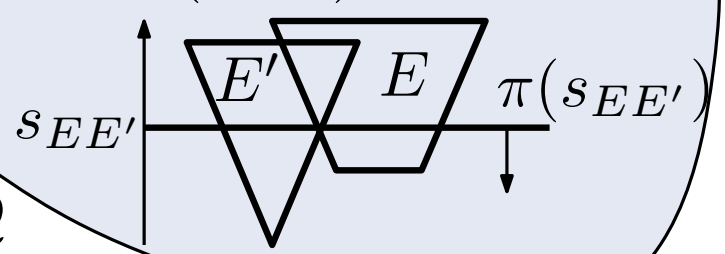
foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$

$s_{EE'}$: größter z-Wert, so dass $E \cap E' \cap \pi(s_{EE'}) = \emptyset$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

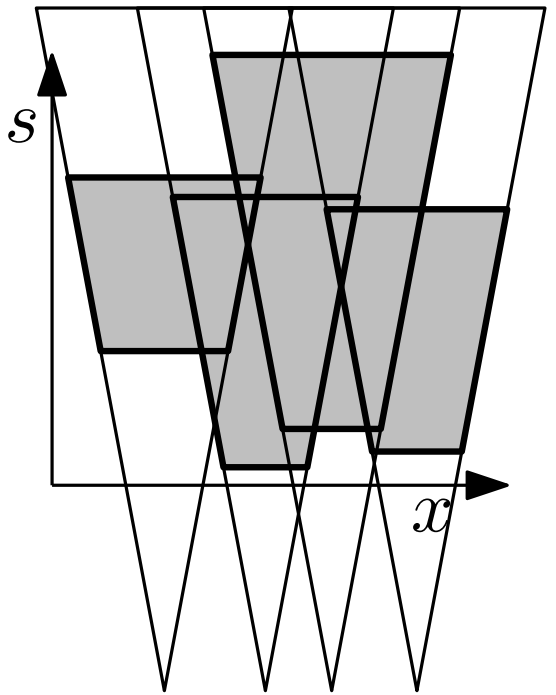
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

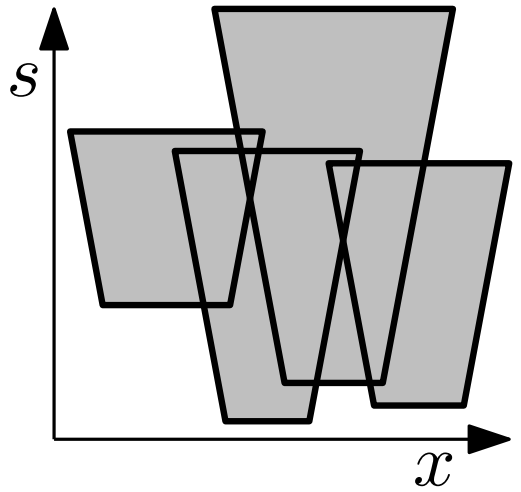
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

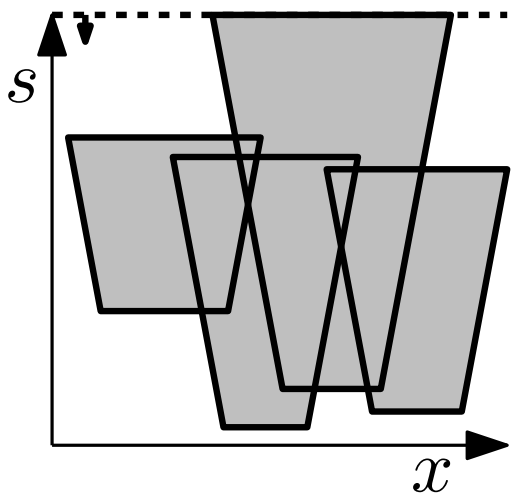
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

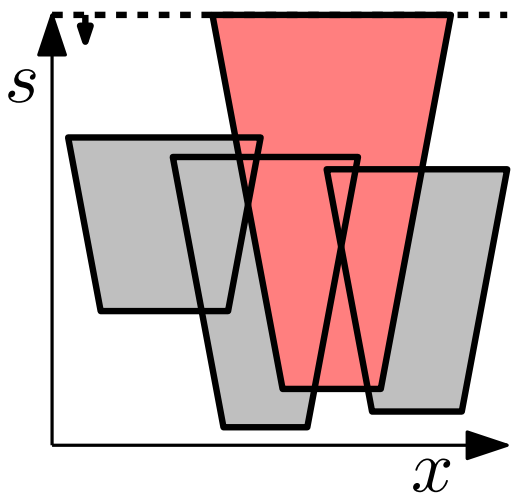
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

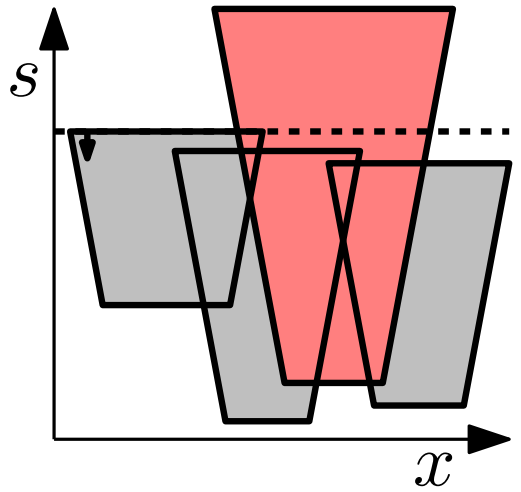
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

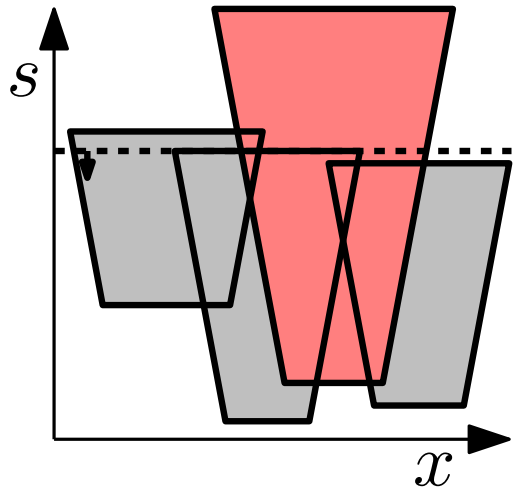
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

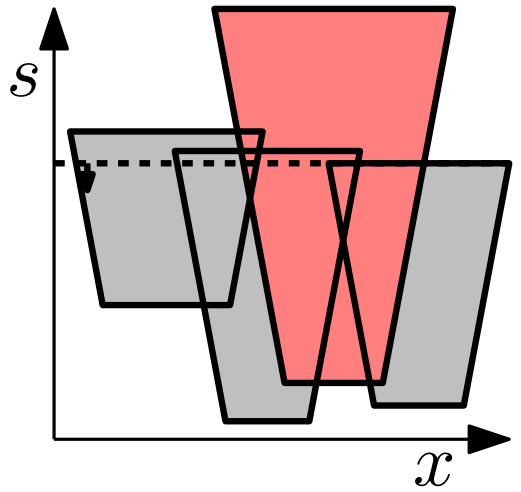
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

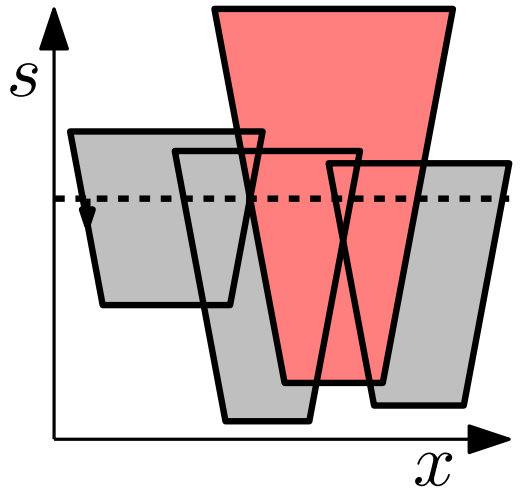
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

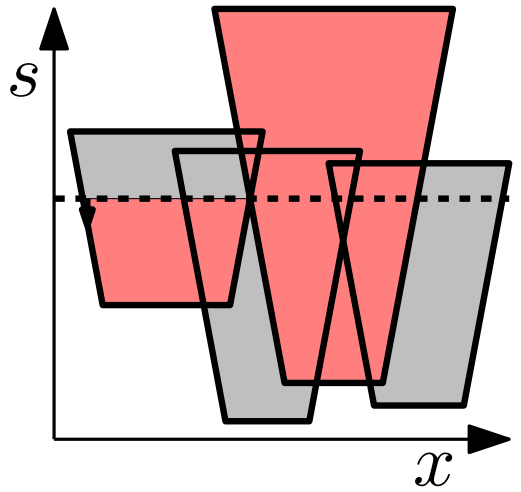
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

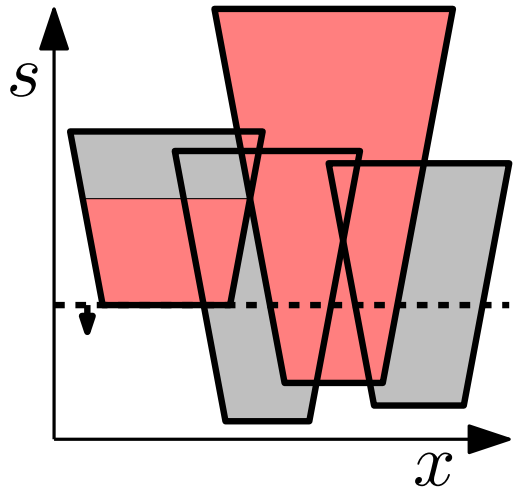
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

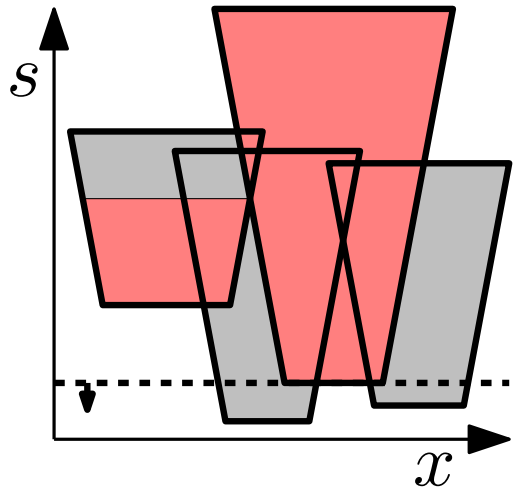
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

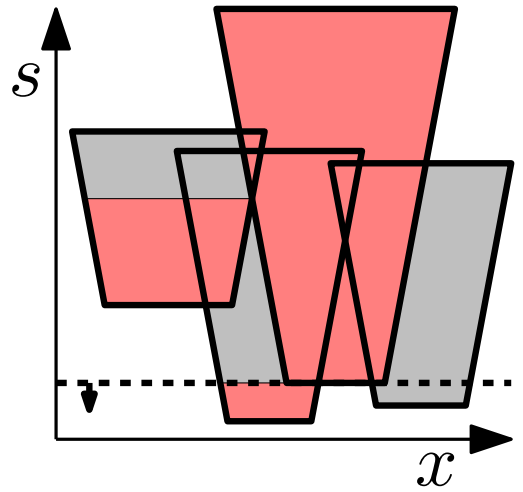
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

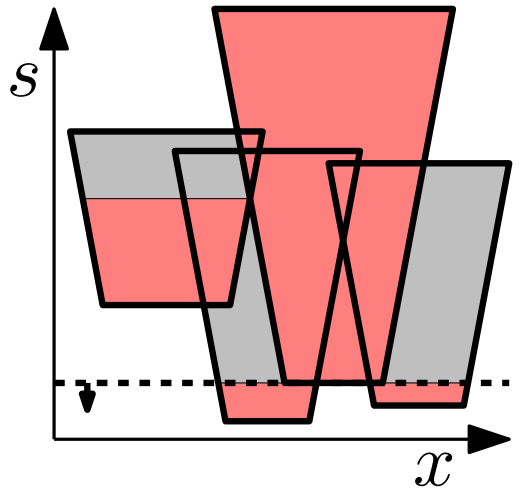
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

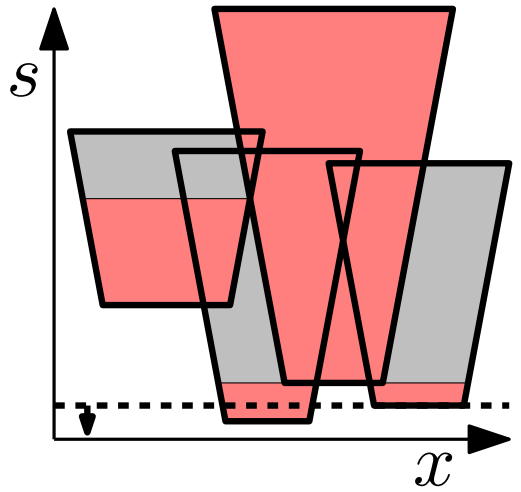
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

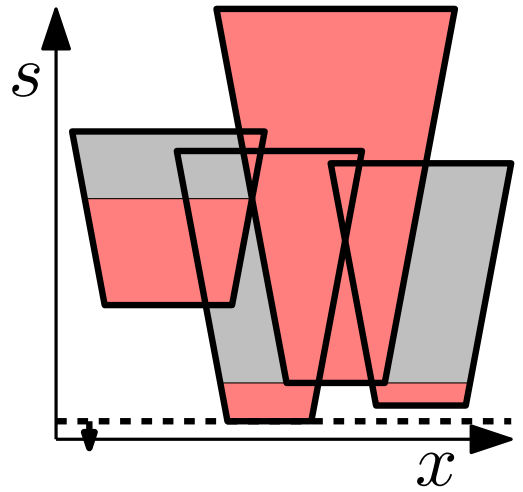
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

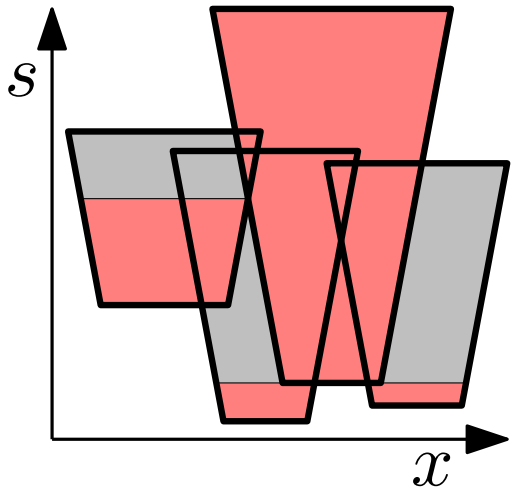
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



Lösung Algorithmus

Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

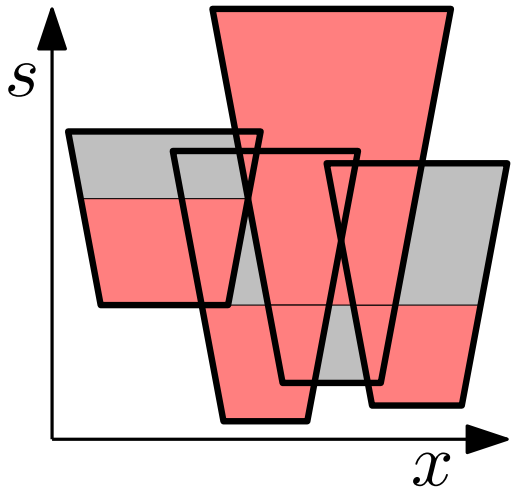
Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$



optimale Lösung

Greedy Top-Down Sweep Algorithmus

Algorithmus 1

Input: Menge $\mathcal{E}$ von (offenen) Labelpyramiden, verfügbare Intervalle (s_E, S_E) für alle $E \in \mathcal{E}$

Output: aktive Intervalle (a_E, A_E) für alle $E \in \mathcal{E}$

foreach $E \in \mathcal{E}$ **do** $(a_E, A_E) \leftarrow (s_E, S_E)$

$\mathcal{Q} \leftarrow$ Priority Queue für $\mathcal{E}$ lexikogr. absteigend sortiert nach (A_E, S_E)

while $\mathcal{Q} \neq \emptyset$ **do**

- $E \leftarrow \mathcal{Q}$.extractMax
- **foreach** $E' \in \mathcal{Q}$ **do**
  - **if** $E' \cap E \neq \emptyset$ **then**
    - $A_{E'} \leftarrow \min\{A_{E'}, s_{EE'}\}$
    - **if** $A_{E'} = a_{E'}$ **then** lösche E' aus $\mathcal{Q}$

Lemma 1: Algorithmus 1 berechnet eine gültige Beschriftung und lässt sich naiv in $O(n^2)$ Zeit und $O(n)$ Platz implementieren.

Blockade-Lemma

Lemma 2: Sei $\mathcal{E}$ eine Menge von *Labelkörpern*. Falls gilt, dass jedes $E \in \mathcal{E}$ für jeden z-Wert s nicht mehr als c paarweise unabhängige Labelkörper blockiert, dann liefert Algorithmus 1 eine $(1/c)$-Approximation.

Blockade-Lemma

Lemma 2: Sei $\mathcal{E}$ eine Menge von *Labelkörpern*. Falls gilt, dass jedes $E \in \mathcal{E}$ für jeden z-Wert s nicht mehr als c paarweise unabhängige Labelkörper blockiert, dann liefert Algorithmus 1 eine $(1/c)$-Approximation.

Begriffe:

- $E \in \mathcal{E}$ **blockiert** $E' \in \mathcal{E}$ am Wert s in einer Lösung $\mathcal{A}$, falls $a_E \leq s \leq A_E$ und $tr_s(E) \cap tr_s(E') \neq \emptyset$
- die **Spur** $tr_s(E)$ eines Labelkörpers $E \in \mathcal{E}$ zum Wert s ist der Schnitt von E mit der Ebene $\pi(s) : z = s$
- $E, E' \in \mathcal{E}$ sind **unabhängig** am Wert s, falls $tr_s(E) \cap tr_s(E') = \emptyset$

Blockade-Lemma

Lemma 2: Sei $\mathcal{E}$ eine Menge von *Labelkörpern*. Falls gilt, dass jedes $E \in \mathcal{E}$ für jeden z-Wert s nicht mehr als c paarweise unabhängige Labelkörper blockiert, dann liefert Algorithmus 1 eine $(1/c)$-Approximation.

Begriffe:

- $E \in \mathcal{E}$ **blockiert** $E' \in \mathcal{E}$ am Wert s in einer Lösung $\mathcal{A}$, falls $a_E \leq s \leq A_E$ und $tr_s(E) \cap tr_s(E') \neq \emptyset$
- die **Spur** $tr_s(E)$ eines Labelkörpers $E \in \mathcal{E}$ zum Wert s ist der Schnitt von E mit der Ebene $\pi(s) : z = s$
- $E, E' \in \mathcal{E}$ sind **unabhängig** am Wert s, falls $tr_s(E) \cap tr_s(E') = \emptyset$

Was ist c für den Fall von quadratischen Einheitslabeln?
Was ist c für den Fall einheitlich großer Rechtecke?

Approximation für Einheitslabel

Satz 2: Für n achsenparallele Label gleicher Größe berechnet Algorithmus 1 eine $1/4$-Approximation der optimalen aktiven Gesamthöhe. Der Algorithmus lässt sich in $O((n+k)\log^2 n)$ Zeit und $O(n \log n)$ Platz implementieren.

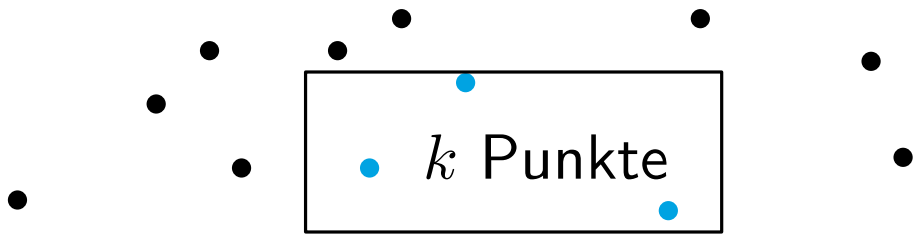
k ist die Anzahl von Seitenflächen-Schnitten der Labelpyramiden

Approximation für Einheitslabel

Satz 2: Für n achsenparallele Label gleicher Größe berechnet Algorithmus 1 eine $1/4$-Approximation der optimalen aktiven Gesamthöhe. Der Algorithmus lässt sich in $O((n+k)\log^2 n)$ Zeit und $O(n \log n)$ Platz implementieren.

k ist die Anzahl von Seitenflächen-Schnitten der Labelpyramiden

Range Tree: geometrische Datenstruktur zur Beantwortung rechteckiger Bereichsanfragen für eine Punktmenge P der Größe n. Aufbau: $O(n \log n)$, Speicher: $O(n \log n)$, Query: $O(k + \log^2 n)$



Ebenenbasierter Greedy-Algorithmus

Algorithmus 2

Input: Menge $\mathcal{E}$ von quadratischen Labelpyramiden, verfügbare Intervalle $(0, S_{\max})$ für alle $E \in \mathcal{E}$

Output: aktive Intervalle $(0, A_E)$ für alle $E \in \mathcal{E}$

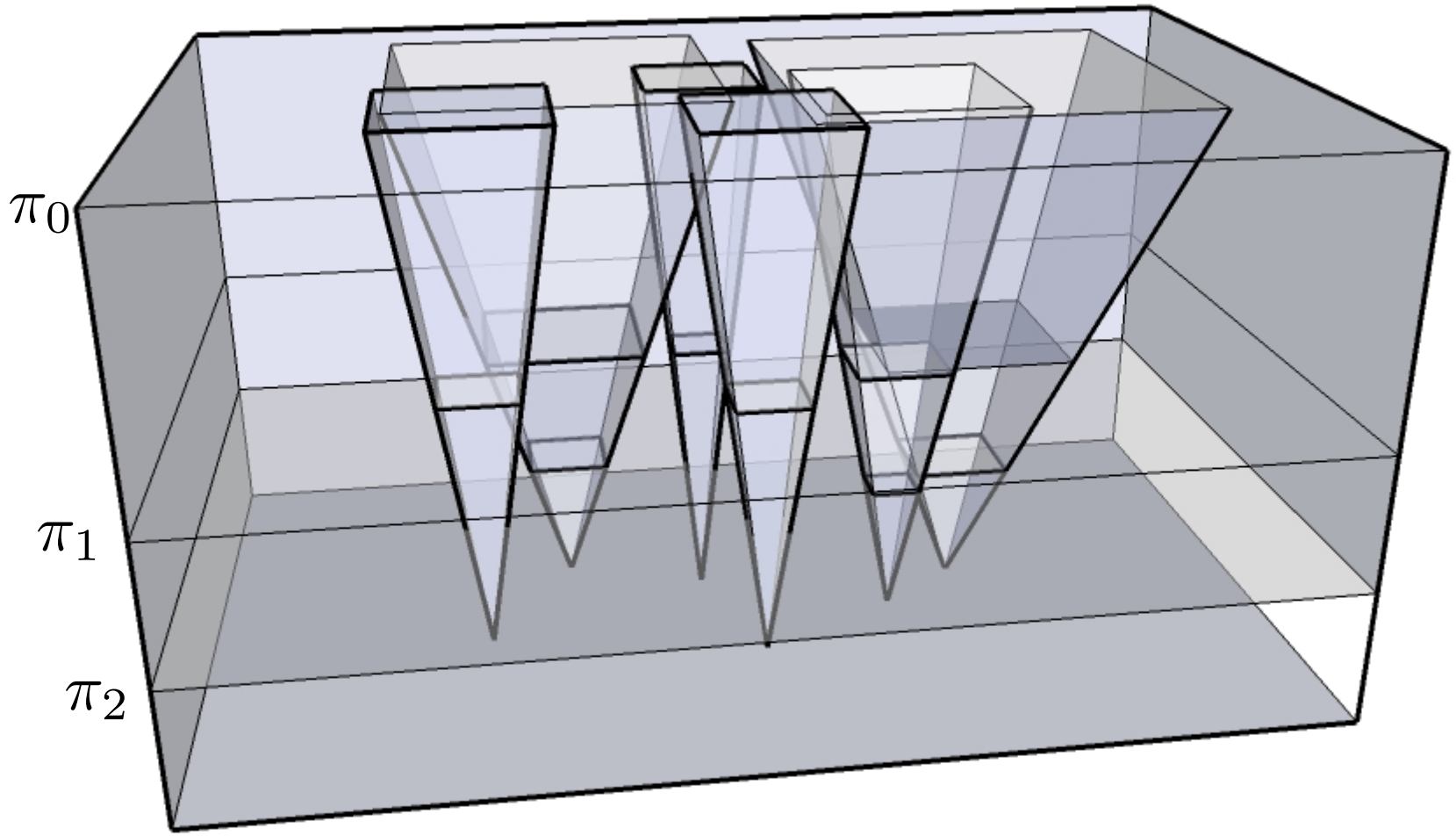
foreach $E \in \mathcal{E}$ **do** initialisiere E als inaktiv; $A_E \leftarrow 0$

for $i = 0$ **to** $\log_2 n$ `// Phase` i

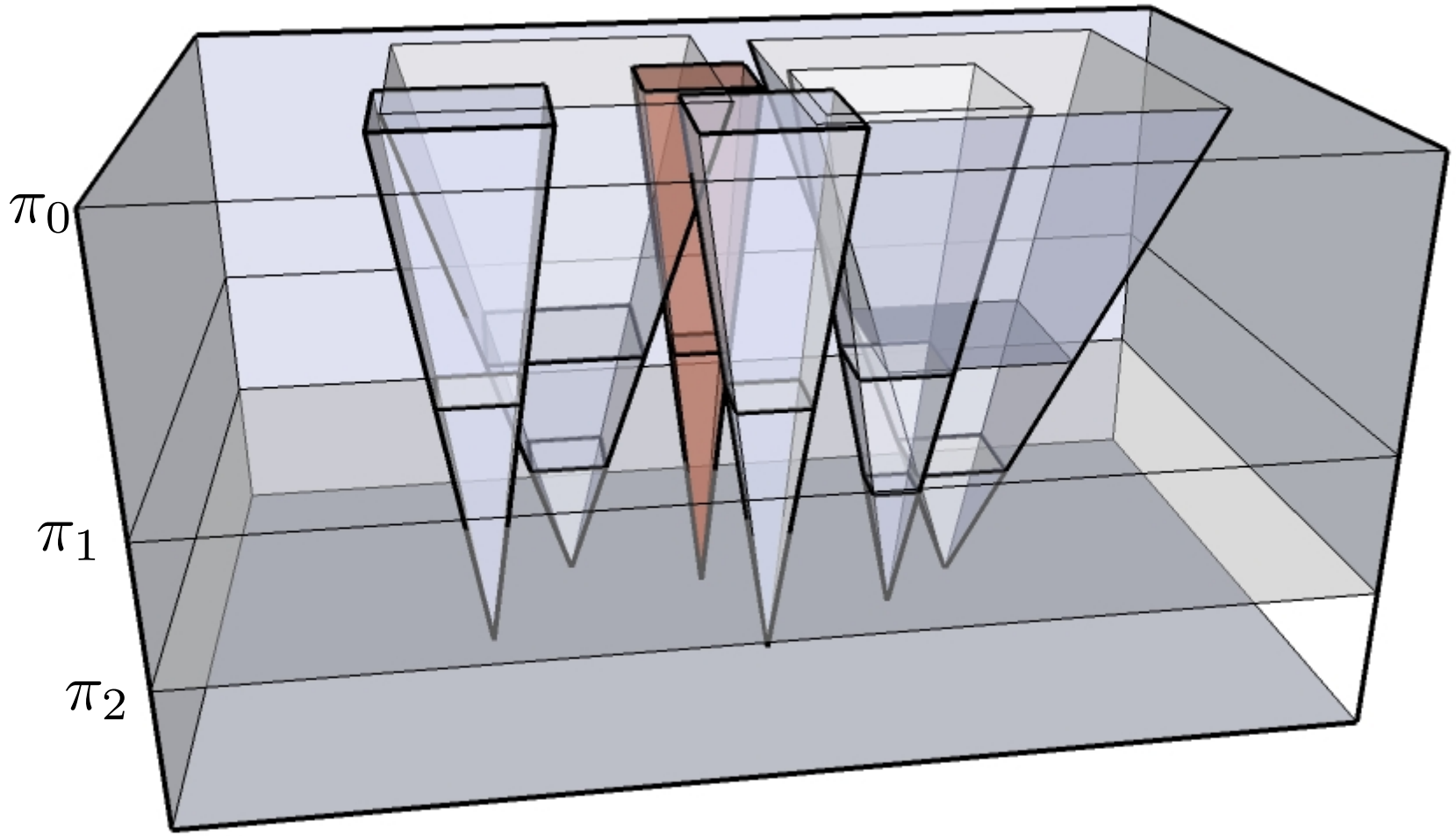
do

- $s_i \leftarrow S_{\max}/2^i$
- $\mathcal{C}_i \leftarrow$ Menge inaktiver Spuren in $\pi_i = \pi(s_i)$, die keine aktiven Spuren in π_i schneiden
- **while** $\mathcal{C}_i \neq \emptyset$ **do**
  - $T \leftarrow$ kleinste Spur in $\mathcal{C}_i$; $E \leftarrow$ Pyramide zu T
  - markiere E und T aktiv und setze $A_E \leftarrow s_i$
  - entferne T und alle geschnittenen Spuren aus $\mathcal{C}_i$

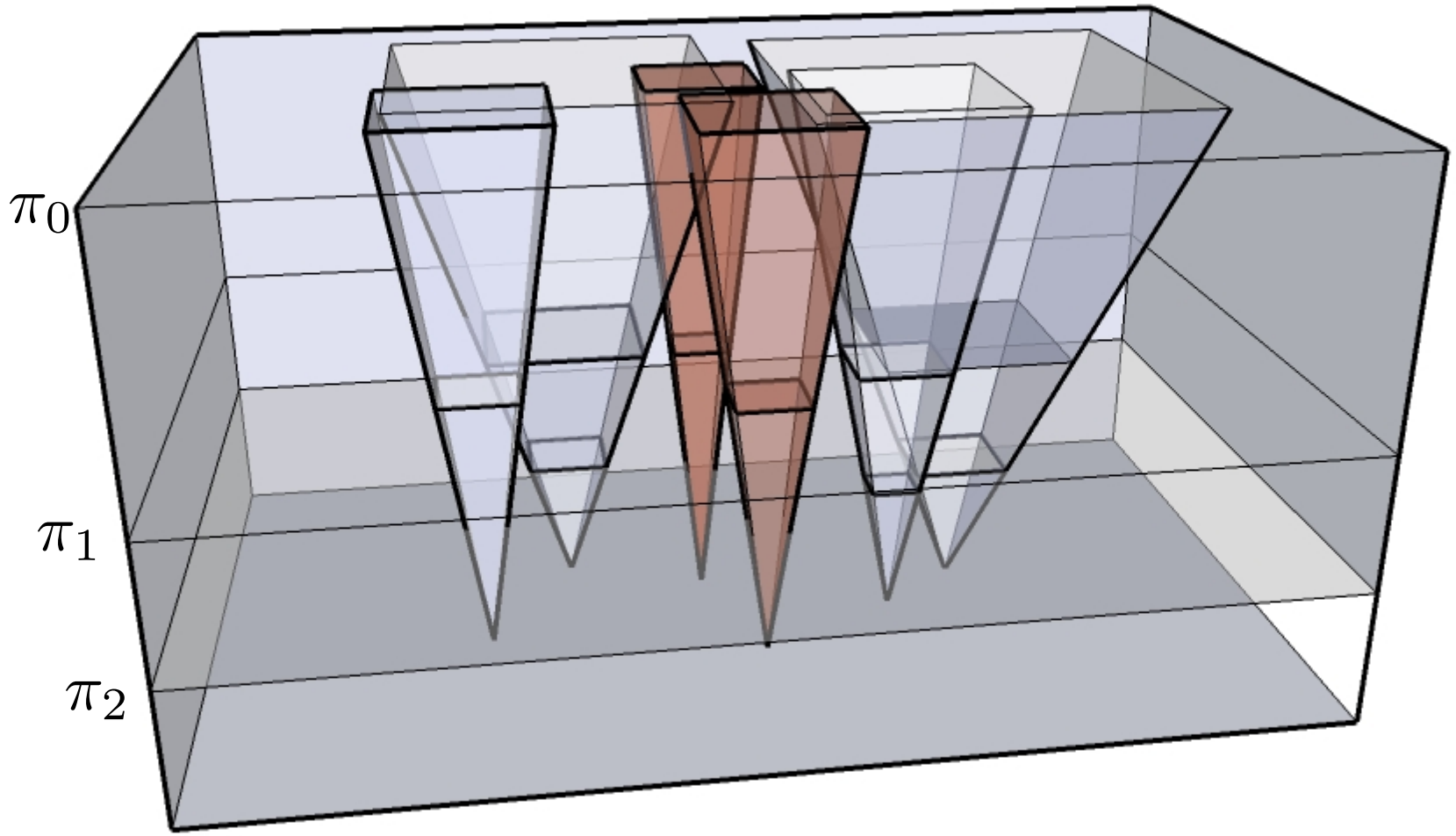
Beispiel



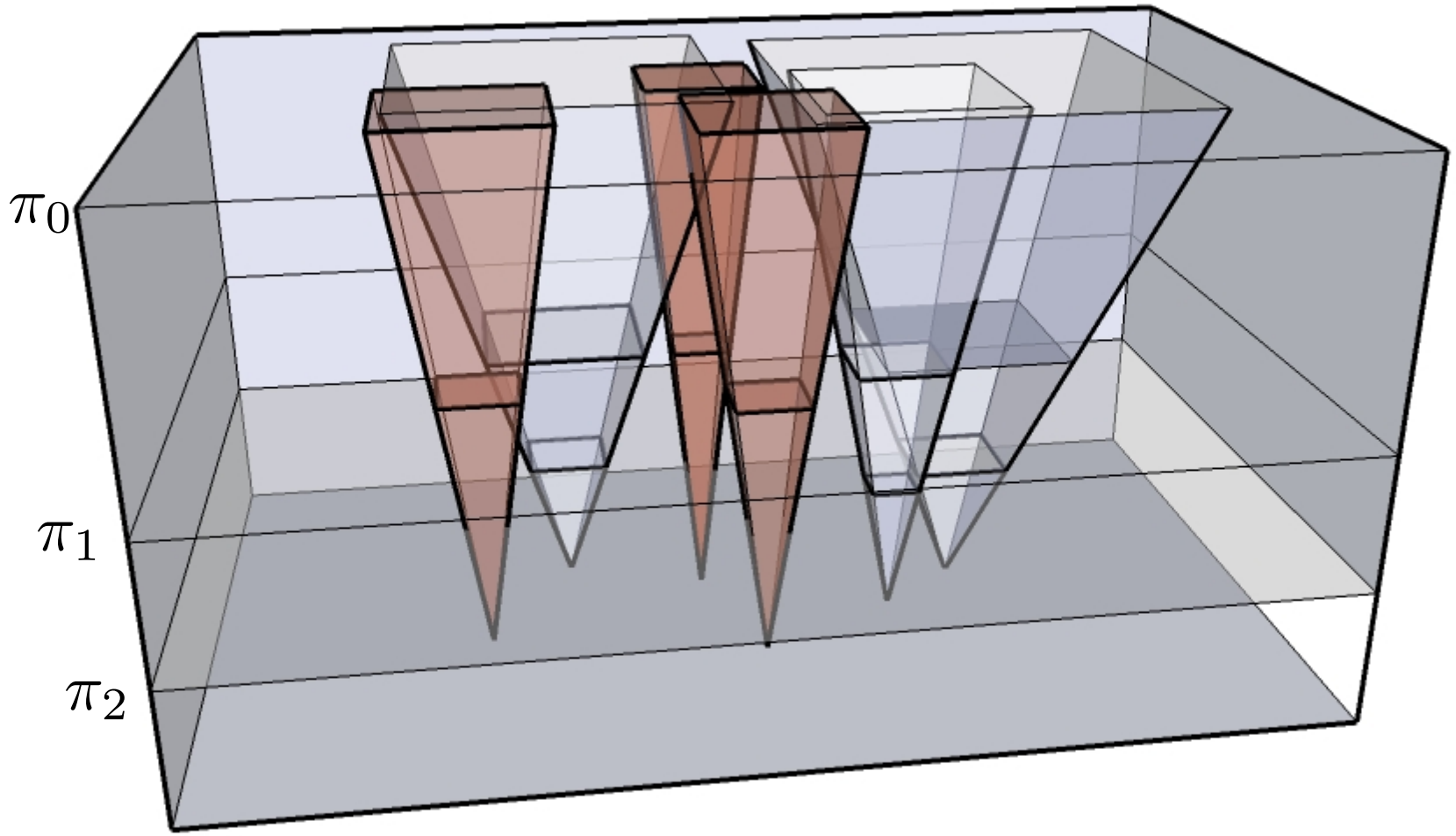
Beispiel



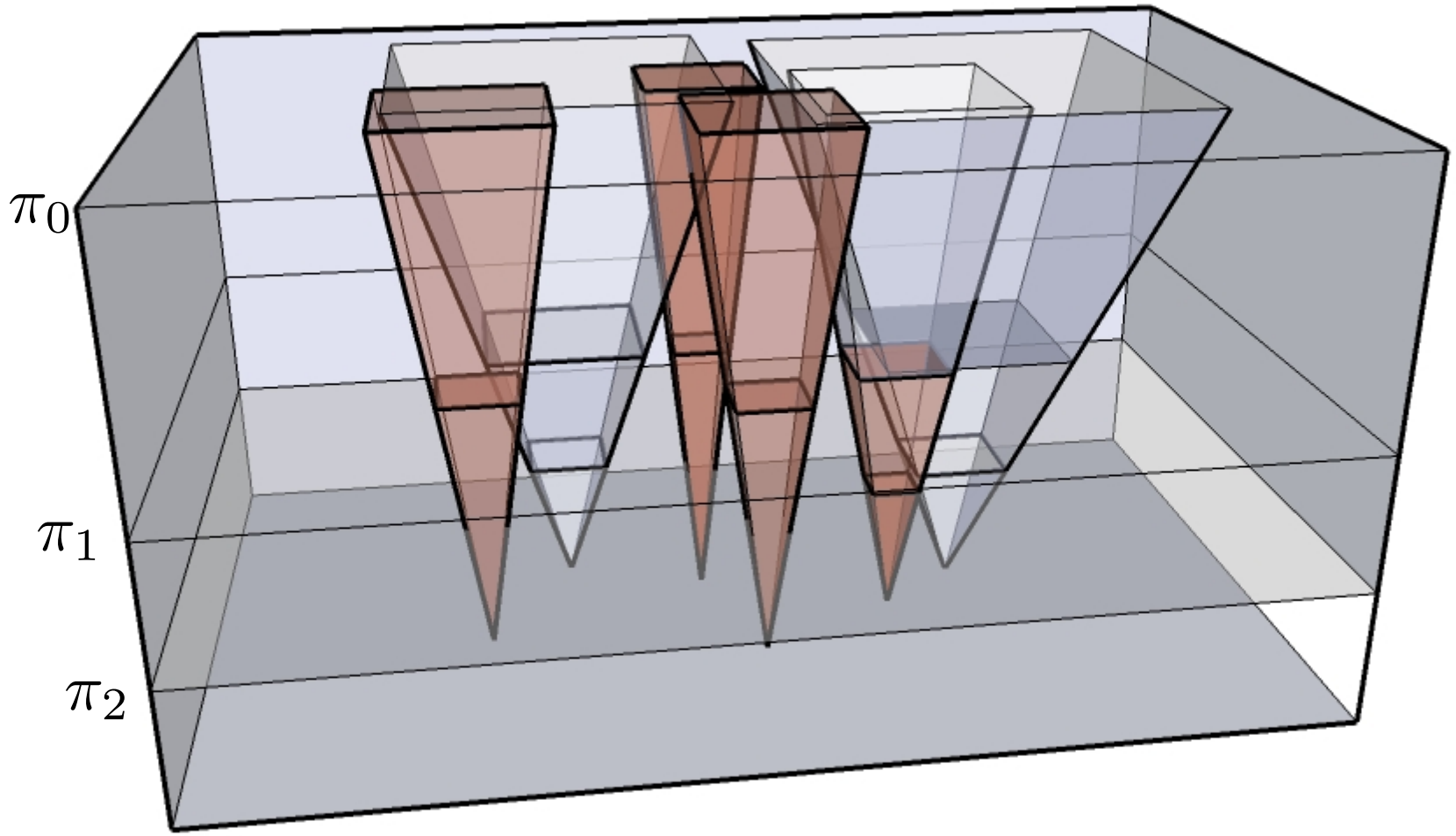
Beispiel



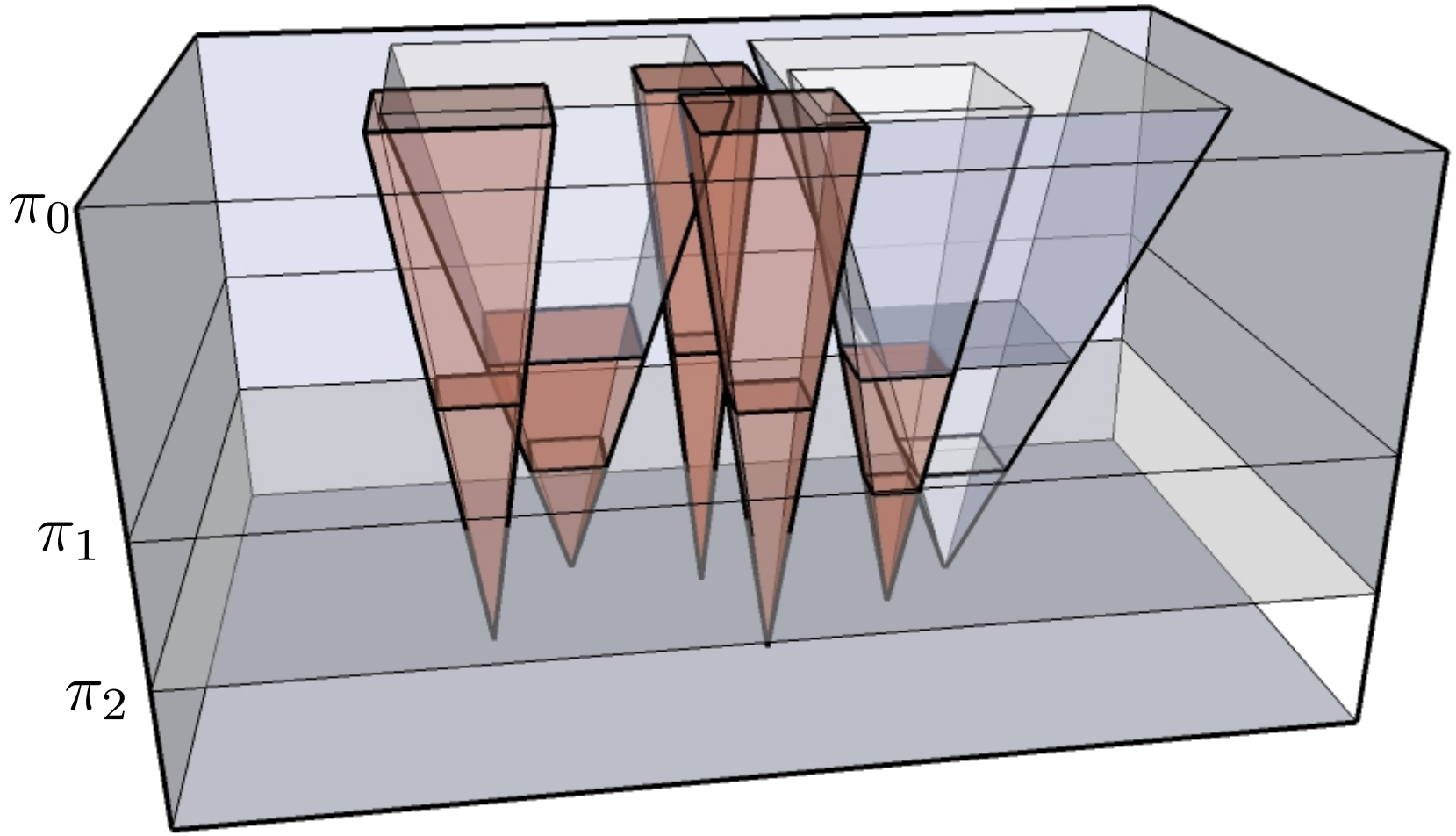
Beispiel



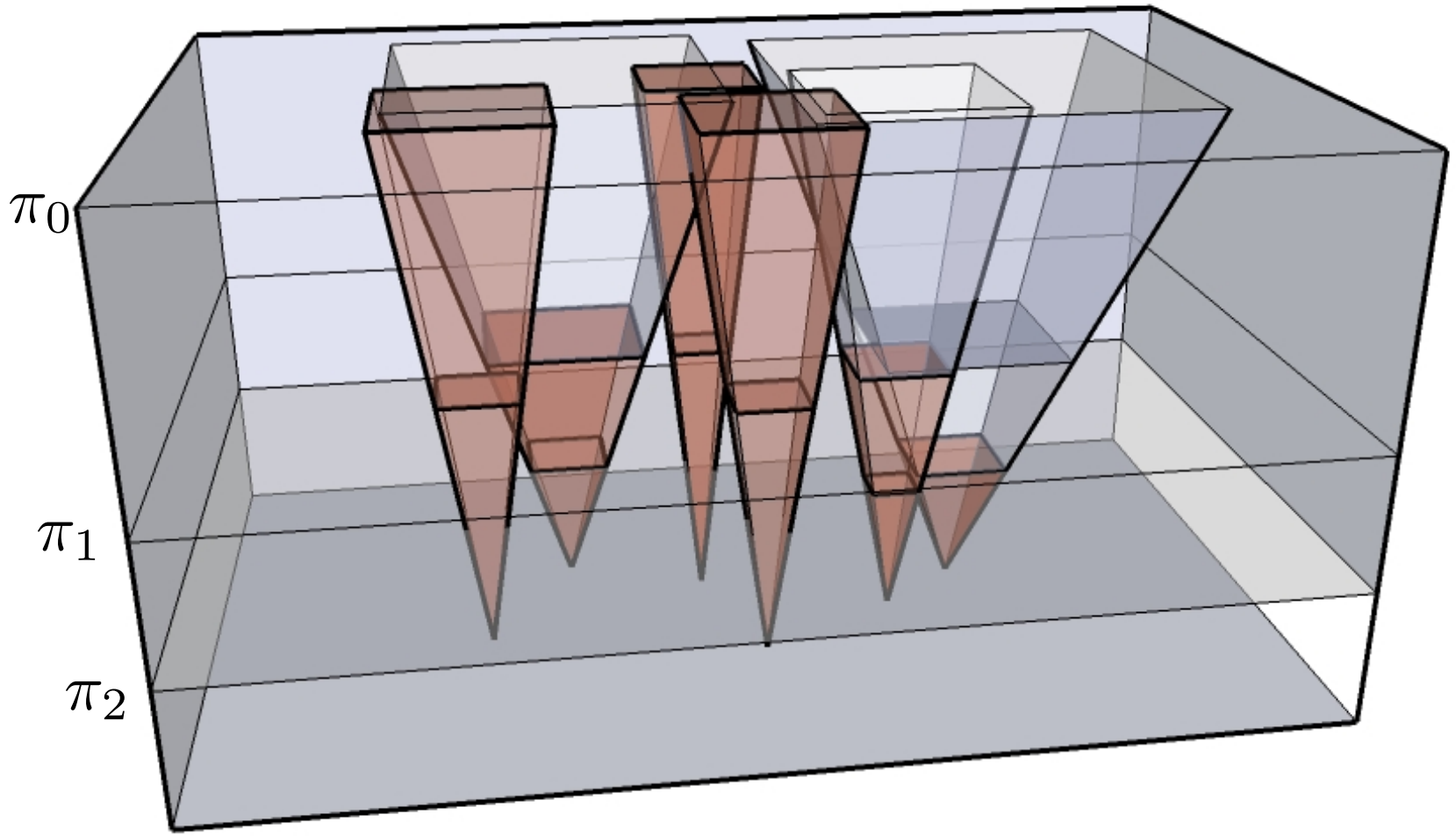
Beispiel



Beispiel



Beispiel



Blocker-Zuordnung

Am Ende von Phase i schneiden alle inaktiven Spuren eine aktive Spur in Ebene π_i, sie werden **blockiert**.

Sei T eine blockierte Spur. Wir weisen T einer aktiven Spur zu:

- falls T anfangs in $\mathcal{C}_i$ war, wurde T durch neu aktivierte Spur T' blockiert; weise T der Spur T' zu
- sonst weise T einer beliebigen blockierenden Spur T' zu, die schon am Anfang von Phase i aktiv war

Blocker-Zuordnung

Am Ende von Phase i schneiden alle inaktiven Spuren eine aktive Spur in Ebene π_i, sie werden **blockiert**.

Sei T eine blockierte Spur. Wir weisen T einer aktiven Spur zu:

- falls T anfangs in $\mathcal{C}_i$ war, wurde T durch neu aktivierte Spur T' blockiert; weise T der Spur T' zu
- sonst weise T einer beliebigen blockierenden Spur T' zu, die schon am Anfang von Phase i aktiv war

Lemma 3: Sei T eine aktive Spur in Ebene π_i mit Seitenlänge ℓ. Dann hat jede inaktive und T zugeordnete Spur Seitenlänge mindestens $\ell/3$ und schneidet den Rand von T.

Abschätzung aktive Gesamthöhe

Sei $\mathcal{S}$ optimale Lösung und $\mathcal{A}$ Lösung des Algorithmus 2.
Sei $\pi_{\log_2 n+1} = \pi(0)$.
Definiere $\mathcal{S}_i$ und $\mathcal{A}_i$ als Menge der aktiven Pyramidenstümpfe von $\mathcal{S}$ bzw. $\mathcal{A}$ zwischen π_{i-1} und π_i.

Abschätzung aktive Gesamthöhe

Sei $\mathcal{S}$ optimale Lösung und $\mathcal{A}$ Lösung des Algorithmus 2.
Sei $\pi_{\log_2 n+1} = \pi(0)$.
Definiere $\mathcal{S}_i$ und $\mathcal{A}_i$ als Menge der aktiven Pyramidenstümpfe von $\mathcal{S}$ bzw. $\mathcal{A}$ zwischen π_{i-1} und π_i.

Weise aktive Gesamthöhe $H(\mathcal{S}_i)$ der Gesamthöhe $H(\mathcal{A}_{i+1})$ zu; weise $H(\mathcal{S}_{\log_2 n})$ und $H(\mathcal{S}_{\log_2 n+1})$ der Gesamthöhe $H(\mathcal{A}_1)$ zu.

Lemma 4: Es gilt $H(\mathcal{A}_1) \geq (H(\mathcal{S}_{\log_2 n}) + H(\mathcal{S}_{\log_2 n+1}))/4$.
Wenn nicht mehr als c Spuren in $\mathcal{S}$ einer beliebigen Spur in $\mathcal{A}$ zugeordnet sind, dann gilt für $1 \leq i < \log_2 n$: $H(\mathcal{A}_{i+1}) \geq H(\mathcal{S}_i)/(2c)$.

Zusammenfassung

Satz 3: Für n achsenparallele quadratische Label beliebiger Größe berechnet Algorithmus 2 eine $1/24$-Approximation der optimalen aktiven Gesamthöhe. Der Algorithmus lässt sich in $O(n \log^3 n)$ Zeit und $O(n \log n)$ Platz implementieren.

Zusammenfassung

Satz 3: Für n achsenparallele quadratische Label beliebiger Größe berechnet Algorithmus 2 eine $1/24$-Approximation der optimalen aktiven Gesamthöhe. Der Algorithmus lässt sich in $O(n \log^3 n)$ Zeit und $O(n \log n)$ Platz implementieren.

Abschlussfrage:
Was macht Algorithmus 2 für Label einheitlicher Größe?