Karlsruher Institut für Technologie

Algorithmen II
Übung am 04.02.2014

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

Organisatorisches

Fragestunde und Wiederholung am 13.02.14:

Fragen können auch bereits im Voraus an uns geschickt werden:
thomas.blaesius@kit.edu, benjamin.niedermann@kit.edu

Für die Klausur können Wörterbücher (Deutsch $\leftrightarrow$ Englisch) ausgeliehen werden.

↳ Bei Interesse bis zum 13.02.14 eine E-Mail schreiben.

Karlsruher Institut für Technologie

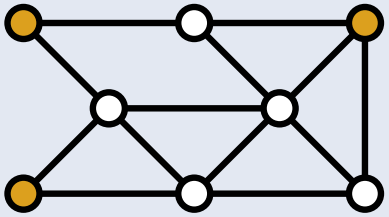
Parametrisierte Algorithmen

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

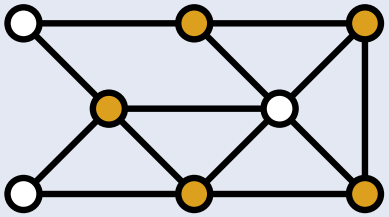
Drei Probleme

Gegeben: Ein Graph $G = (V, E)$, sowie ein Parameter $k \in \mathbb{N}$.

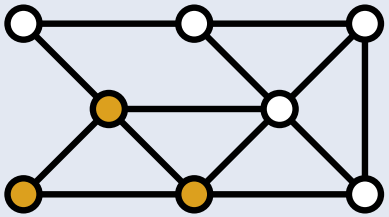
Problem: INDEPENDENT SET
Finde *unabhängige Menge* $V' \subseteq V$ mit $|V'| \geq k$. V' heißt unabhängig genau dann, wenn für alle $v, w \in V'$ gilt $\{v, w\} \notin E$.



Problem: VERTEX COVER
Finde *Vertex Cover* $V' \subseteq V$ mit $|V'| \leq k$. V' heißt Vertex Cover genau dann, wenn für jede Kante $\{v, w\} \in E$ gilt $v \in V'$ oder $w \in V'$.



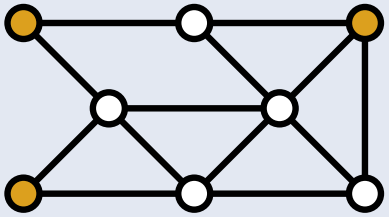
Problem: CLIQUE
Finde *Clique* $V' \subseteq V$ mit $|V'| \geq k$. V' heißt *Clique* genau dann, wenn für jedes Paar $u, v \in V'$ gilt, dass $\{u, v\} \in E$.



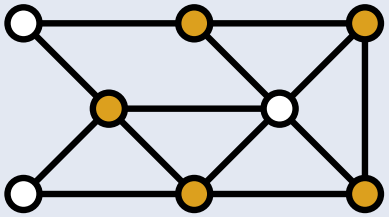
Drei Probleme

Gegeben: Ein Graph $G = (V, E)$, sowie ein Parameter $k \in \mathbb{N}$.

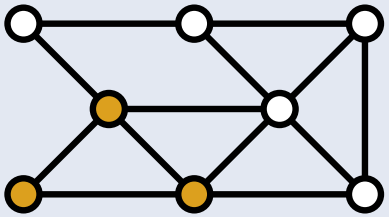
Problem: INDEPENDENT SET
Finde *unabhängige Menge* $V' \subseteq V$ mit $|V'| \geq k$. V' heißt unabhängig genau dann, wenn für alle $v, w \in V'$ gilt $\{v, w\} \notin E$.



Problem: VERTEX COVER
Finde *Vertex Cover* $V' \subseteq V$ mit $|V'| \leq k$. V' heißt Vertex Cover genau dann, wenn für jede Kante $\{v, w\} \in E$ gilt $v \in V'$ oder $w \in V'$.



Problem: CLIQUE
Finde *Clique* $V' \subseteq V$ mit $|V'| \geq k$. V' heißt *Clique* genau dann, wenn für jedes Paar $u, v \in V'$ gilt, dass $\{u, v\} \in E$.



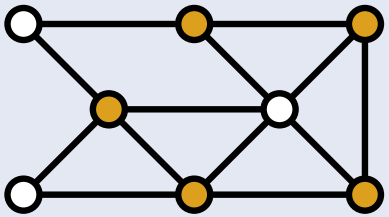
INDEPENDENT SET, VERTEX COVER und CLIQUE sind $\mathcal{NP}$-schwer.

$\rightarrow$ Aufzählung aller $\binom{n}{k}$ Teilmengen (Brute-Force) V' mit $|V'| = k$ liefert Algorithmus mit Laufzeit $O(n^k \cdot (n + m))$. Für konstantes k polynomiell! Geht es noch besser?

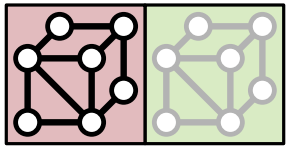
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Algorithmus für VERTEX COVER

Problem: VERTEX COVER
Finde *Vertex Cover* $V' \subseteq V$ mit $|V'| \leq k$. V' heißt Vertex Cover genau dann, wenn für jede Kante $\{v, w\} \in E$ gilt $v \in V'$ oder $w \in V'$.



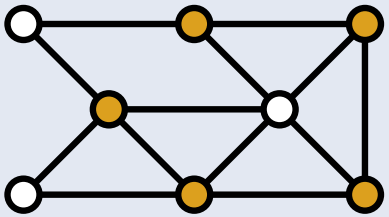
noch zu überdeckender Teilgraph ausgewählte Knoten & überdeckte Kanten



Gibt es ein VERTEX COVER mit maximal 3 Knoten?

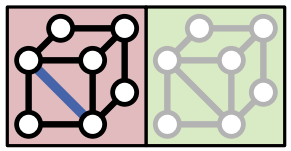
Algorithmus für VERTEX COVER

Problem: VERTEX COVER
Finde *Vertex Cover* $V' \subseteq V$ mit $|V'| \leq k$. V' heißt Vertex Cover genau dann, wenn für jede Kante $\{v, w\} \in E$ gilt $v \in V'$ oder $w \in V'$.



noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

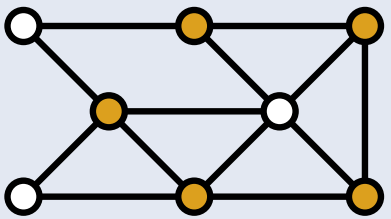
Jede Kante muss noch überdeckt werden. $\rightarrow$ wähle eine beliebige.



Gibt es ein VERTEX COVER mit maximal 3 Knoten?

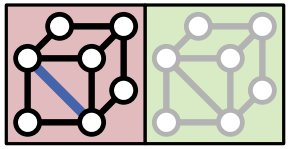
Algorithmus für VERTEX COVER

Problem: VERTEX COVER
Finde *Vertex Cover* $V' \subseteq V$ mit $|V'| \leq k$. V' heißt Vertex Cover genau dann, wenn für jede Kante $\{v, w\} \in E$ gilt $v \in V'$ oder $w \in V'$.

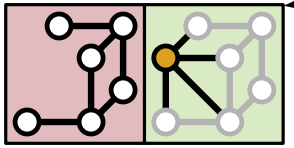


noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

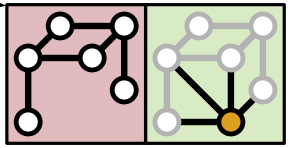
Jede Kante muss noch überdeckt werden. $\rightarrow$ wähle eine beliebige.



Gibt es ein VERTEX COVER mit maximal 3 Knoten?



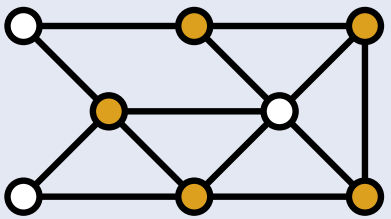
Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
$\rightarrow$ binärer Entscheidungsbaum



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

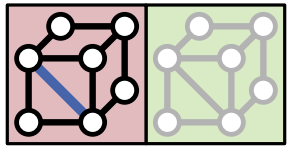
Algorithmus für VERTEX COVER

Problem: VERTEX COVER
Finde *Vertex Cover* $V' \subseteq V$ mit $|V'| \leq k$. V' heißt Vertex Cover genau dann, wenn für jede Kante $\{v, w\} \in E$ gilt $v \in V'$ oder $w \in V'$.

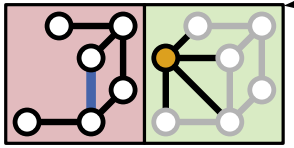


noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

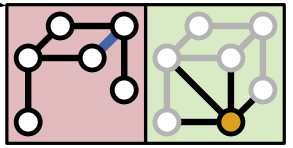
Jede Kante muss noch überdeckt werden. $\rightarrow$ wähle eine beliebige.



Gibt es ein VERTEX COVER mit maximal 3 Knoten?



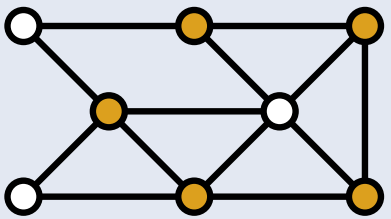
Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
$\rightarrow$ binärer Entscheidungsbaum



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

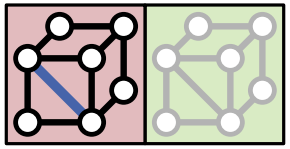
Algorithmus für VERTEX COVER

Problem: VERTEX COVER
Finde *Vertex Cover* $V' \subseteq V$ mit $|V'| \leq k$. V' heißt Vertex Cover genau dann, wenn für jede Kante $\{v, w\} \in E$ gilt $v \in V'$ oder $w \in V'$.



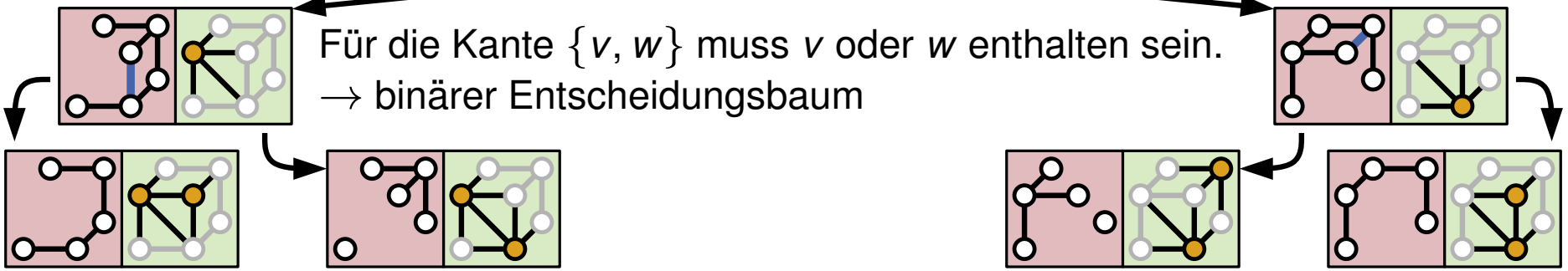
noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

Jede Kante muss noch überdeckt werden. → wähle eine beliebige.



Gibt es ein VERTEX COVER mit maximal 3 Knoten?

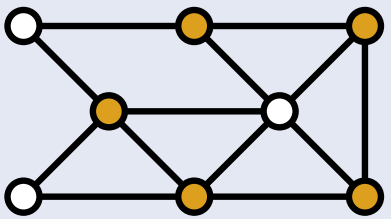
Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
→ binärer Entscheidungsbaum



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

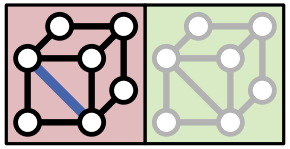
Algorithmus für VERTEX COVER

Problem: VERTEX COVER
Finde *Vertex Cover* $V' \subseteq V$ mit $|V'| \leq k$. V' heißt Vertex Cover genau dann, wenn für jede Kante $\{v, w\} \in E$ gilt $v \in V'$ oder $w \in V'$.



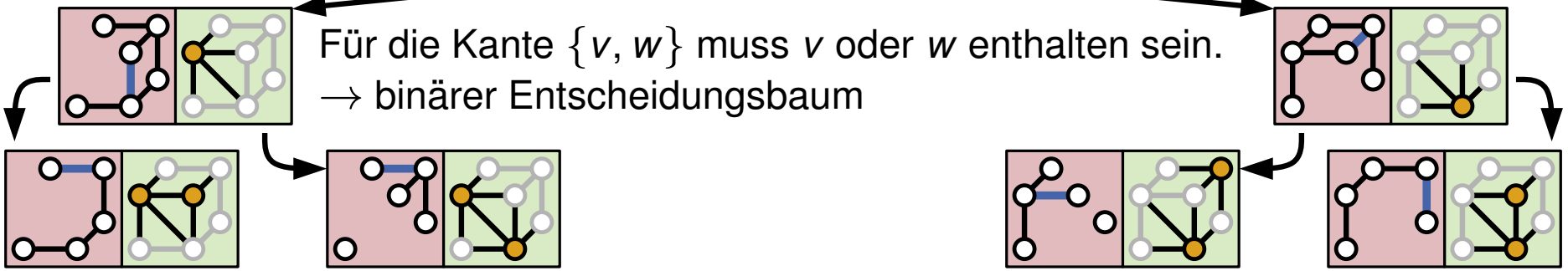
noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

Jede Kante muss noch überdeckt werden. → wähle eine beliebige.



Gibt es ein VERTEX COVER mit maximal 3 Knoten?

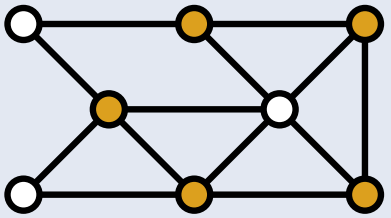
Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
→ binärer Entscheidungsbaum



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

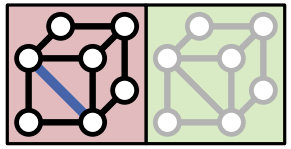
Algorithmus für VERTEX COVER

Problem: VERTEX COVER
Finde *Vertex Cover* $V' \subseteq V$ mit $|V'| \leq k$. V' heißt Vertex Cover genau dann, wenn für jede Kante $\{v, w\} \in E$ gilt $v \in V'$ oder $w \in V'$.



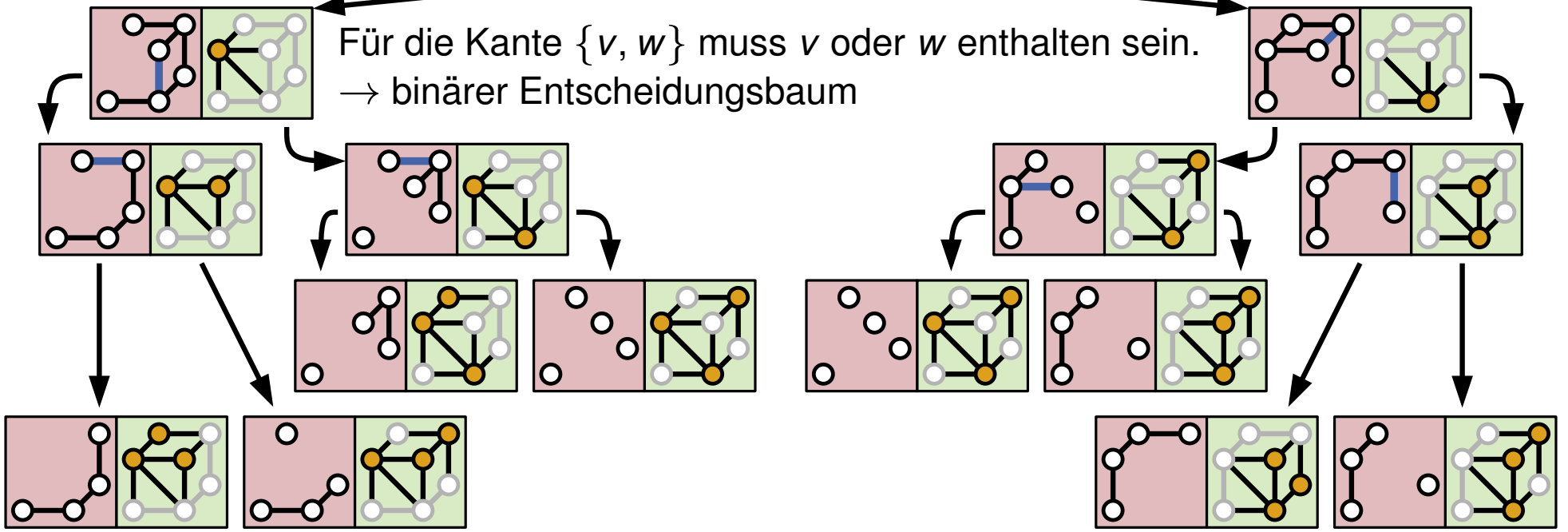
noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

Jede Kante muss noch überdeckt werden. → wähle eine beliebige.



Gibt es ein VERTEX COVER mit maximal 3 Knoten?

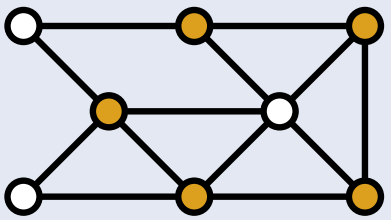
Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
→ binärer Entscheidungsbaum



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

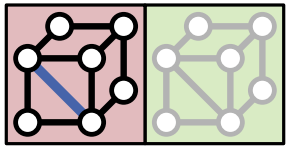
Algorithmus für VERTEX COVER

Problem: VERTEX COVER
Finde *Vertex Cover* $V' \subseteq V$ mit $|V'| \leq k$. V' heißt Vertex Cover genau dann, wenn für jede Kante $\{v, w\} \in E$ gilt $v \in V'$ oder $w \in V'$.



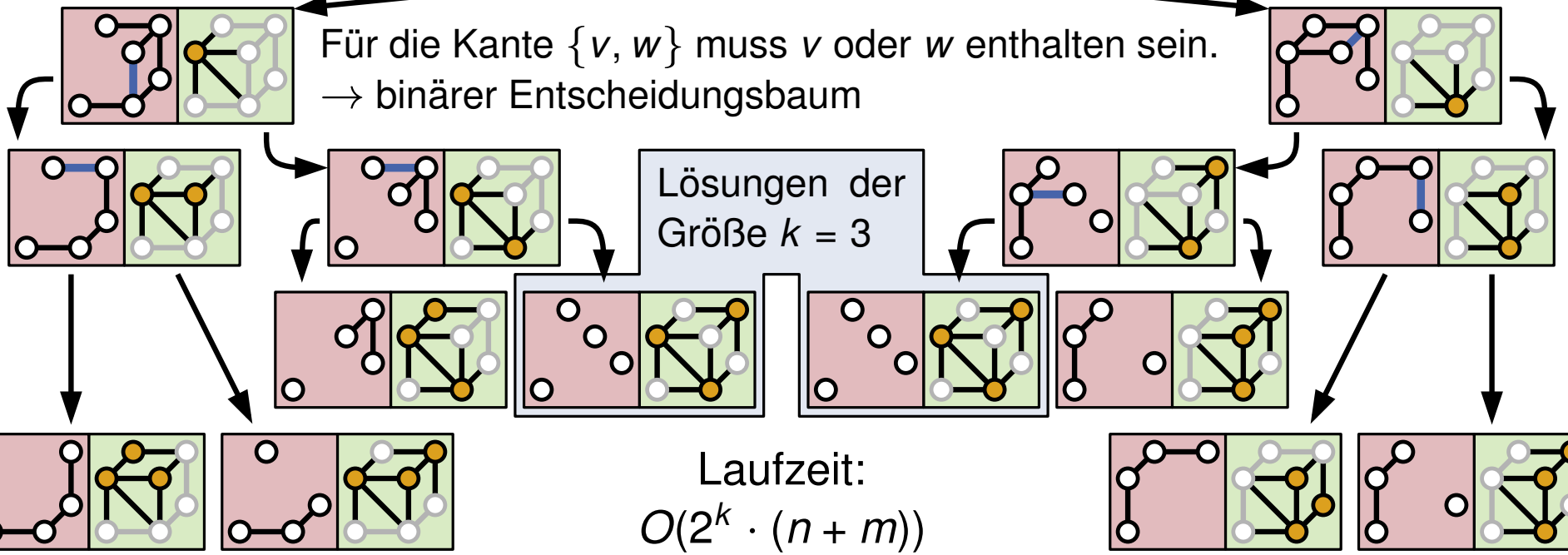
noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

Jede Kante muss noch überdeckt werden. → wähle eine beliebige.



Gibt es ein VERTEX COVER mit maximal 3 Knoten?

Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
→ binärer Entscheidungsbaum



Laufzeit:
$O(2^k \cdot (n + m))$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Fixed Parameter Tractable (FPT)

Definition: Fixed Parameter Tractable **(Definition 10.1)**

Ein parametrisiertes Problem Π heißt *fixed parameter tractable*, wenn es in $O(\mathcal{C}(k) \cdot p(n))$ gelöst werden kann. Dabei ist n die Eingabegröße, p ein Polynom, k der Parameter und $\mathcal{C}$ eine berechenbare Funktion, die nur von k abhängt.

Definition: Komplexitätsklasse FPT **(Definition 10.2)**

FPT ist die Klasse aller Problem, die fixed parameter tractable sind.

Bemerkung: **(Bemerkung 10.3)**

- VERTEX COVER ist in FPT.
- Es ist unbekannt ob INDEPENDENT SET oder DOMINATING SET in FPT sind. Man vermutet, dass sie nicht in FPT sind.

PUNKTÜBERDECKUNG – Alte Klausuraufgabe

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

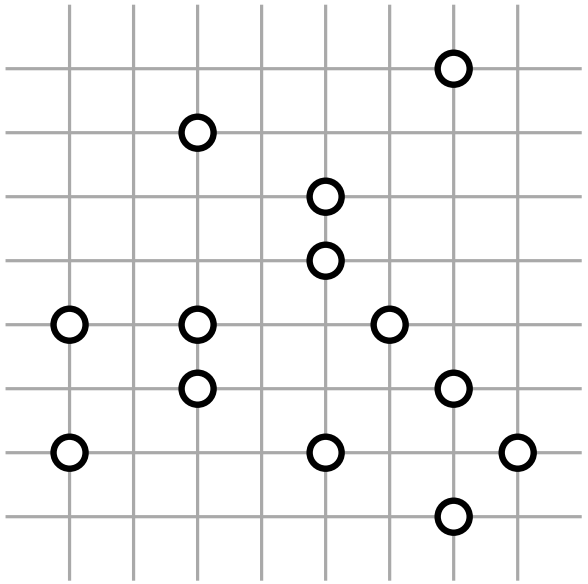
Problemdefinition

PUNKTÜBERDECKUNG:

Gegeben: Eine Menge von Punkten $P = \{p_1, \ldots, p_n\}$ in der Ebene, sowie ein Parameter k.

Gesucht: Menge von maximal k Geraden $G = \{g_1, \ldots, g_k\}$, sodass es fur jeden Punkt $p_i \in P$ eine Gerade $g_j \in G$ mit $p_i \in g_j$ gibt.

a) Zeichnen Sie 3 Geraden ein, die das Problem PUNKTÜBERDECKUNG für $k = 3$ löst.



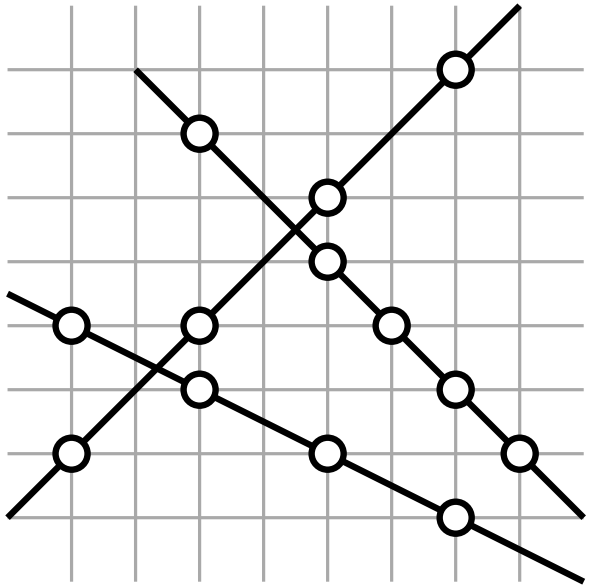
Problemdefinition

PUNKTÜBERDECKUNG:

Gegeben: Eine Menge von Punkten $P = \{p_1, \ldots, p_n\}$ in der Ebene, sowie ein Parameter k.

Gesucht: Menge von maximal k Geraden $G = \{g_1, \ldots, g_k\}$, sodass es fur jeden Punkt $p_i \in P$ eine Gerade $g_j \in G$ mit $p_i \in g_j$ gibt.

a) Zeichnen Sie 3 Geraden ein, die das Problem PUNKTÜBERDECKUNG für $k = 3$ löst.



Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Behauptung: Eine Gerade, die mehr als k Punkte aus P enthält, ist in jeder Lösung enthalten.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

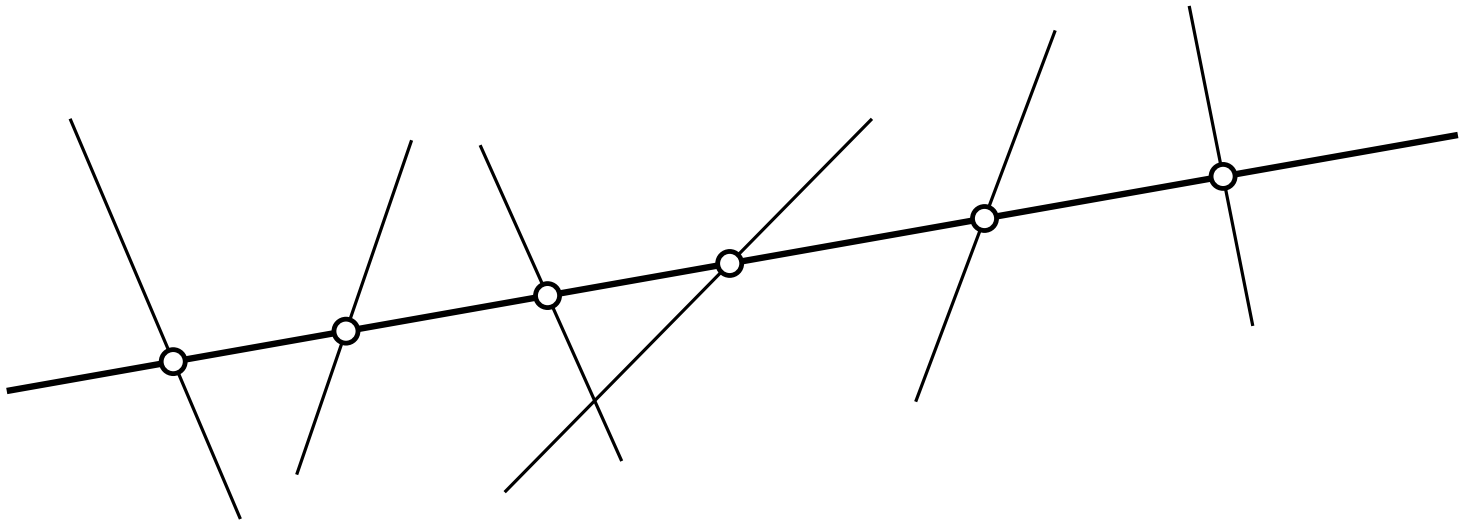
Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Behauptung: Eine Gerade, die mehr als k Punkte aus P enthält, ist in jeder Lösung enthalten.

Sei g eine Gerade, die ℓ Punkte aus P überdeckt.

→ g ist in der Lösung G enthalten oder durch jeden der ℓ Punkte geht eine andere aus G.

→ falls $\ell > k$ dann muss g in G enthalten sein, da sonst $|G| > k$.



Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Eine Gerade, die mehr als k Punkte aus P enthält, ist in jeder Lösung enthalten.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Eine Gerade, die mehr als k Punkte aus P enthält, ist in jeder Lösung enthalten.

Algorithmus

1. Schritt: Solange es Gerade gibt, die mehr als k Punkte überdeckt:

- Füge Gerade zur Lösung G.
- Entferne die entsprechenden Punkte.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Eine Gerade, die mehr als k Punkte aus P enthält, ist in jeder Lösung enthalten.

Algorithmus

1. Schritt: Solange es Gerade gibt, die mehr als k Punkte überdeckt:

- Füge Gerade zur Lösung G.
- Entferne die entsprechenden Punkte.

→ Ergibt neue Instanz mit verbleibenden Punkten P' und $k' = k - |G|$

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Eine Gerade, die mehr als k Punkte aus P enthält, ist in jeder Lösung enthalten.

Algorithmus

1. Schritt: Solange es Gerade gibt, die mehr als k Punkte überdeckt:

- Füge Gerade zur Lösung G.
- Entferne die entsprechenden Punkte.

→ Ergibt neue Instanz mit verbleibenden Punkten P' und $k' = k - |G|$

Es gibt Lösung für Instanz (P, k) ⟷ Es gibt Lösung für Instanz (P', k')

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Eine Gerade, die mehr als k Punkte aus P enthält, ist in jeder Lösung enthalten.

Algorithmus

1. Schritt: Solange es Gerade gibt, die mehr als k Punkte überdeckt:

Laufzeit: $O(n^3)$

- Füge Gerade zur Lösung G.
- Entferne die entsprechenden Punkte.

→ Ergibt neue Instanz mit verbleibenden Punkten P' und $k' = k - |G|$

Es gibt Lösung für Instanz (P, k) ⟷ Es gibt Lösung für Instanz (P', k')

Laufzeit: Es gibt $O(n^2)$ Geraden, die durch mindestens zwei Punkte in P gehen.

Für jede dieser Geraden überprüft man, ob sie k Punkte überdeckt.

→ Insgesamt wird $O(n^3)$ Zeit benötigt.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Eine Gerade, die mehr als k Punkte aus P enthält, ist in jeder Lösung enthalten.

Algorithmus

1. Schritt: Solange es Gerade gibt, die mehr als k Punkte überdeckt: **Laufzeit:** $O(n^3)$

- Füge Gerade zur Lösung G.
- Entferne die entsprechenden Punkte.

2. Schritt: Falls $|P'| > k^2$ dann gebe FALSCH zurück. **Laufzeit:** $O(1)$

Begründung: Jede Gerade kann nur k der übrigen Punkte enthalten. Daher kann man mit $k' \leq k$ Geraden nicht mehr als k^2 Punkte uberdecken.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Eine Gerade, die mehr als k Punkte aus P enthält, ist in jeder Lösung enthalten.

Algorithmus

1. Schritt: Solange es Gerade gibt, die mehr als k Punkte überdeckt: **Laufzeit:** $O(n^3)$

- Füge Gerade zur Lösung G.
- Entferne die entsprechenden Punkte.

2. Schritt: Falls $|P'| > k^2$ dann gebe FALSCH zurück. **Laufzeit:** $O(1)$

3. Schritt:Betrachte die Menge aller Geraden $G^\star$, die mindestens zwei Punkte aus P enthalten. Uberprüfe für jede Teilmenge von $G^\star$ der Größe k', ob sie alle Punkte in P überdeckt.

- Es gibt solche Teilmenge: gib sie zusammen mit den in Schritt 1. gefundenen Geraden aus.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Wegen $|P'| < k^2$ gilt $|G^\star| \in O(k^4)$.

3. Schritt:Betrachte die Menge aller Geraden $G^\star$, die mindestens zwei Punkte aus P enthalten. Uberprüfe für jede Teilmenge von $G^\star$ der Größe k', ob sie alle Punkte in P überdeckt.

↳ Es gibt solche Teilmenge: gib sie zusammen mit den in Schritt 1. gefundenen Geraden aus.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Wegen $|P'| < k^2$ gilt $|G^\star| \in O(k^4)$.

⟶ Anzahl Teilmengen von $G^\star$ der Größe k liegt also in $O((k^4)^k) = O(k^{4k})$

3. Schritt:Betrachte die Menge aller Geraden $G^\star$, die mindestens zwei Punkte aus P enthalten. Uberprüfe für jede Teilmenge von $G^\star$ der Größe k', ob sie alle Punkte in P überdeckt.

↳ Es gibt solche Teilmenge: gib sie zusammen mit den in Schritt 1. gefundenen Geraden aus.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Wegen $|P'| < k^2$ gilt $|G^\star| \in O(k^4)$.

- ⟶ Anzahl Teilmengen von $G^\star$ der Größe k liegt also in $O((k^4)^k) = O(k^{4k})$
- ⟶ Pro Teilmenge benötigt man $O(|P'| \cdot k) = O(k^3)$ Zeit, für Überprüfung ob alle Punkte in P' überdeckt sind.

3. Schritt:Betrachte die Menge aller Geraden $G^\star$, die mindestens zwei Punkte aus P enthalten. Uberprüfe für jede Teilmenge von $G^\star$ der Größe k', ob sie alle Punkte in P überdeckt.

↳ Es gibt solche Teilmenge: gib sie zusammen mit den in Schritt 1. gefundenen Geraden aus.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Wegen $|P'| < k^2$ gilt $|G^\star| \in O(k^4)$.

→ Anzahl Teilmengen von $G^\star$ der Größe k liegt also in $O((k^4)^k) = O(k^{4k})$

→ Pro Teilmenge benötigt man $O(|P'| \cdot k) = O(k^3)$ Zeit, für Überprüfung ob alle Punkte in P' überdeckt sind.

→ Schritt 3 benötigt $O(k^{4k} \cdot k^3)$ Zeit.

3. Schritt:Betrachte die Menge aller Geraden $G^\star$, die mindestens zwei Punkte aus P enthalten. Uberprüfe für jede Teilmenge von $G^\star$ der Größe k', ob sie alle Punkte in P überdeckt.

↳ Es gibt solche Teilmenge: gib sie zusammen mit den in Schritt 1. gefundenen Geraden aus.

Punktüberdeckung

b) Geben Sie einen FPT-Algorithmus für PUNKTÜBERDECKUNG an. Begründen Sie, warum ihr Algorithmus korrekt ist.

Hinweis: Benutzen Sie die Technik der Kernbildung. Zeigen Sie dazu zunächst, dass eine Gerade, die mehr als k Punkte aus P enthält, in jeder Lösung enthalten sein muss.

Eine Gerade, die mehr als k Punkte aus P enthält, ist in jeder Lösung enthalten.

Algorithmus

1. Schritt: Solange es Gerade gibt, die mehr als k Punkte überdeckt: **Laufzeit:** $O(n^3)$

- Füge Gerade zur Lösung G.
- Entferne die entsprechenden Punkte.

2. Schritt: Falls $|P'| > k^2$ dann gebe FALSCH zurück. **Laufzeit:** $O(1)$

3. Schritt:Betrachte die Menge aller Geraden $G^\star$, die mindestens zwei Punkte aus P enthalten. Uberprüfe für jede Teilmenge von $G^\star$ der Größe k', ob sie alle Punkte in P überdeckt.

- Es gibt solche Teilmenge: gib sie zusammen mit den in Schritt 1. gefundenen Geraden aus.

Laufzeit: $O(k^{4k} \cdot k^3)$

Insgesamt: $O(k^{4k} \cdot k^3 + n^3)$

FPT-Algorithmus für MULTICUT

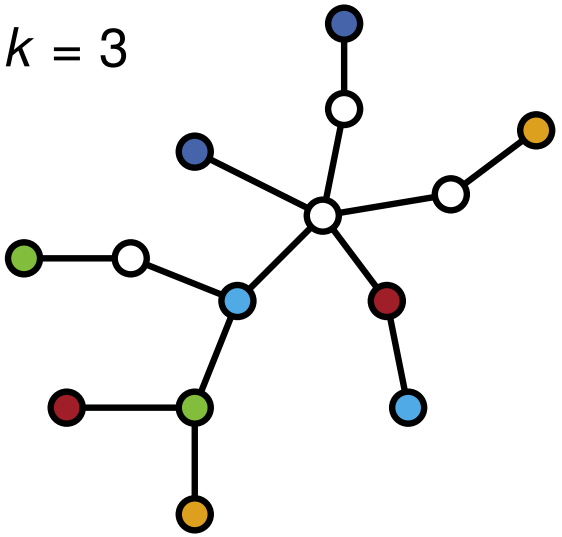
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

MULTICUT in Bäumen

Eingabe:

Baum $T = (V, E)$, eine Menge von Knotenpaaren $H \subseteq \binom{V}{2}$, sowie Parameter k.

$k = 3$

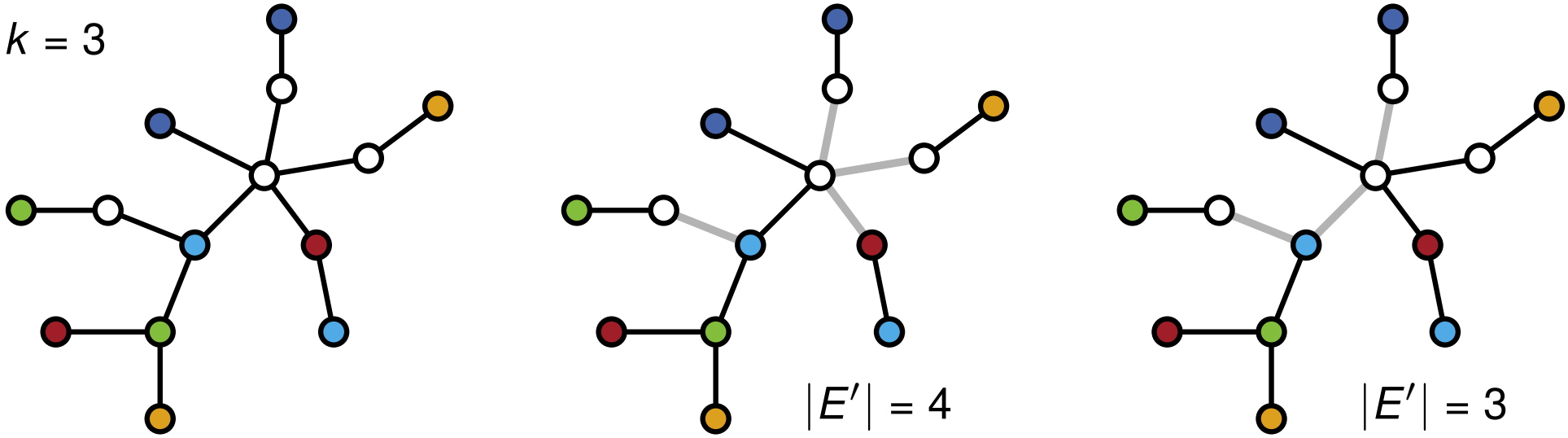


Gibt es eine Kantenmenge $E' \subseteq E$ mit $|E'| \leq k$, die alle Paare trennt?

MULTICUT in Bäumen

Eingabe:

Baum $T = (V, E)$, eine Menge von Knotenpaaren $H \subseteq \binom{V}{2}$, sowie Parameter k.

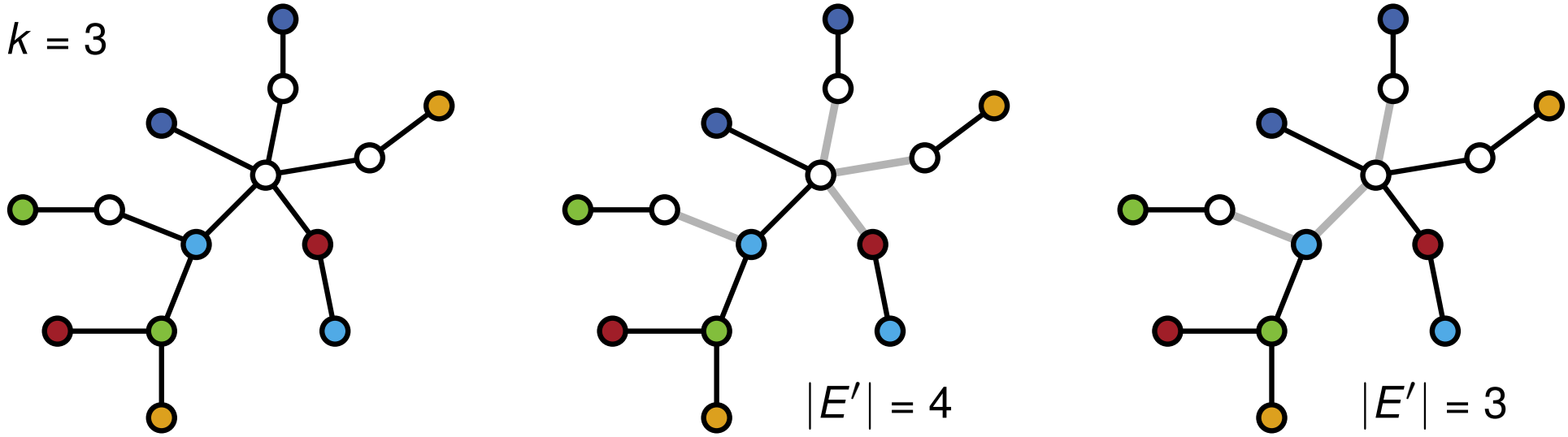


Gibt es eine Kantenmenge $E' \subseteq E$ mit $|E'| \leq k$, die alle Paare trennt?

MULTICUT in Bäumen

Eingabe:

Baum $T = (V, E)$, eine Menge von Knotenpaaren $H \subseteq \binom{V}{2}$, sowie Parameter k.



Gibt es eine Kantenmenge $E' \subseteq E$ mit $|E'| \leq k$, die alle Paare trennt?

Problem 2:

Zeigen Sie, dass MULTICUT auf Bäumen Fixed Parameter Tractable bezüglich k ist.

Erinnerung:

Ein Problem ist FPT, wenn es in $O(\mathcal{C}(k) \cdot p(n))$ Zeit gelöst werden kann.

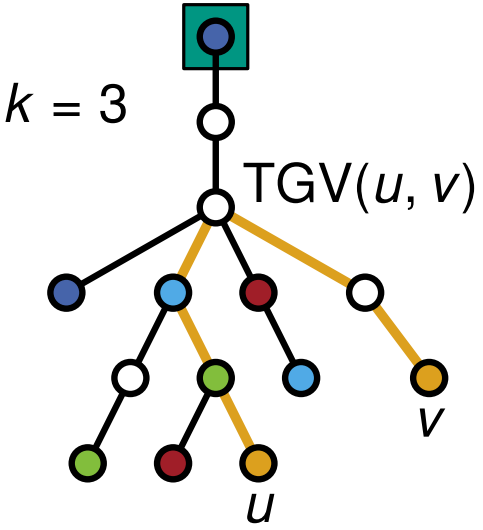
($\mathcal{C}(k)$ ist berechenbar und hängt nur von k ab, $p(n)$ ist ein Polynom in der Eingabegröße n)

Entscheidungsbaum

Ziel: Finde ein Kantenpaar, sodass mindestens eine der beiden Kanten im Schnitt enthalten sein muss. $\rightarrow$ Binärer Entscheidungsbaum

Entscheidungsbaum

Ziel: Finde ein Kantenpaar, sodass mindestens eine der beiden Kanten im Schnitt enthalten sein muss. $\rightarrow$ Binärer Entscheidungsbaum

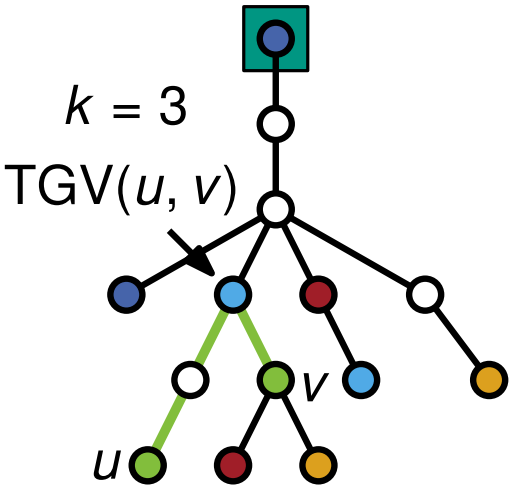


Tiefster gemeinsamer Vorgänger

- Wähle beliebigen Knoten als Wurzel.
- Der Tiefste gemeinsame Vorgänger TGV(u, v) zweier Knoten u und v ist der höchste Knoten auf dem Pfad $\pi(u, v)$ zwischen u und v.

Entscheidungsbaum

Ziel: Finde ein Kantenpaar, sodass mindestens eine der beiden Kanten im Schnitt enthalten sein muss. $\rightarrow$ Binärer Entscheidungsbaum



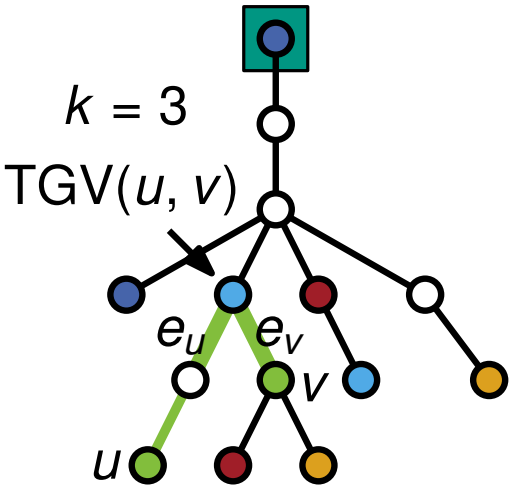
Tiefster gemeinsamer Vorgänger

- Wähle beliebigen Knoten als Wurzel.
- Der Tiefste gemeinsame Vorgänger TGV(u, v) zweier Knoten u und v ist der höchste Knoten auf dem Pfad $\pi(u, v)$ zwischen u und v.

Betrachte Knotenpaar $\{u, v\} \in H$, sodass TGV(u, v) möglichst tief ist.

Entscheidungsbaum

Ziel: Finde ein Kantenpaar, sodass mindestens eine der beiden Kanten im Schnitt enthalten sein muss. $\rightarrow$ Binärer Entscheidungsbaum



Tiefster gemeinsamer Vorgänger

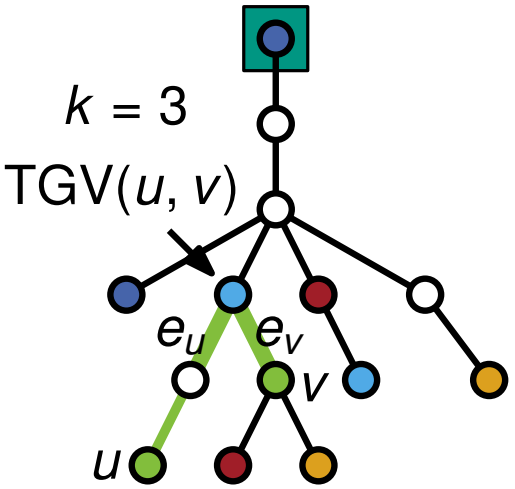
- Wähle beliebigen Knoten als Wurzel.
- Der Tiefste gemeinsame Vorgänger TGV(u, v) zweier Knoten u und v ist der höchste Knoten auf dem Pfad $\pi(u, v)$ zwischen u und v.

Betrachte Knotenpaar $\{u, v\} \in H$, sodass TGV(u, v) möglichst tief ist.

Behauptung: Eine der beiden Kanten e_u und e_v inzident zu TGV(u, v) auf $\pi(u, v)$ ist in jedem minimalen Schnitt enthalten.

Entscheidungsbaum

Ziel: Finde ein Kantenpaar, sodass mindestens eine der beiden Kanten im Schnitt enthalten sein muss. $\rightarrow$ Binärer Entscheidungsbaum



Tiefster gemeinsamer Vorgänger

- Wähle beliebigen Knoten als Wurzel.
- Der Tiefste gemeinsame Vorgänger TGV(u, v) zweier Knoten u und v ist der höchste Knoten auf dem Pfad $\pi(u, v)$ zwischen u und v.

Betrachte Knotenpaar $\{u, v\} \in H$, sodass TGV(u, v) möglichst tief ist.

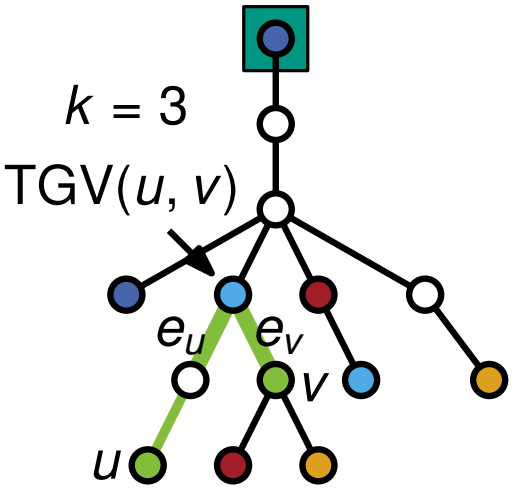
Behauptung: Eine der beiden Kanten e_u und e_v inzident zu TGV(u, v) auf $\pi(u, v)$ ist in jedem minimalen Schnitt enthalten.

Beweis:

- Eine Kante auf $\pi(u, v)$ muss im Schnitt enthalten sein um u von v zu trennen.

Entscheidungsbaum

Ziel: Finde ein Kantenpaar, sodass mindestens eine der beiden Kanten im Schnitt enthalten sein muss. $\rightarrow$ Binärer Entscheidungsbaum



Tiefster gemeinsamer Vorgänger

- Wähle beliebigen Knoten als Wurzel.
- Der Tiefste gemeinsame Vorgänger TGV(u, v) zweier Knoten u und v ist der höchste Knoten auf dem Pfad $\pi(u, v)$ zwischen u und v.

Betrachte Knotenpaar $\{u, v\} \in H$, sodass TGV(u, v) möglichst tief ist.

Behauptung: Eine der beiden Kanten e_u und e_v inzident zu TGV(u, v) auf $\pi(u, v)$ ist in jedem minimalen Schnitt enthalten.

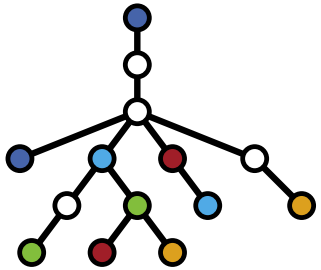
Beweis:

- Eine Kante auf $\pi(u, v)$ muss im Schnitt enthalten sein um u von v zu trennen.
- Keine Kante auf $\pi(u, \text{TGV}(u, v))$ trennt ein Knotenpaar in H, dass nicht von e_u getrennt wird. (analog für e_v)

$\Rightarrow$ Entweder e_u oder e_v müssen gewählt werden.

Beispiel

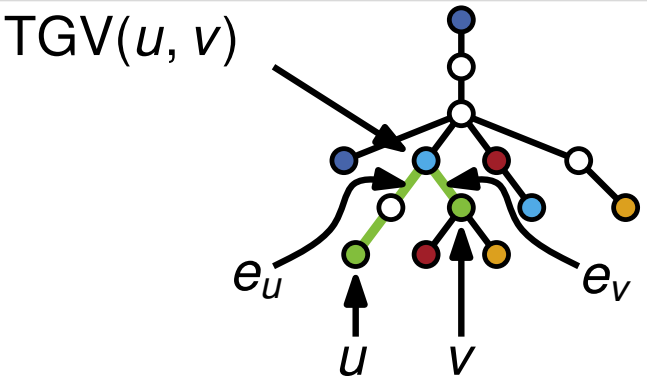
$k = 3$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel

$k = 3$

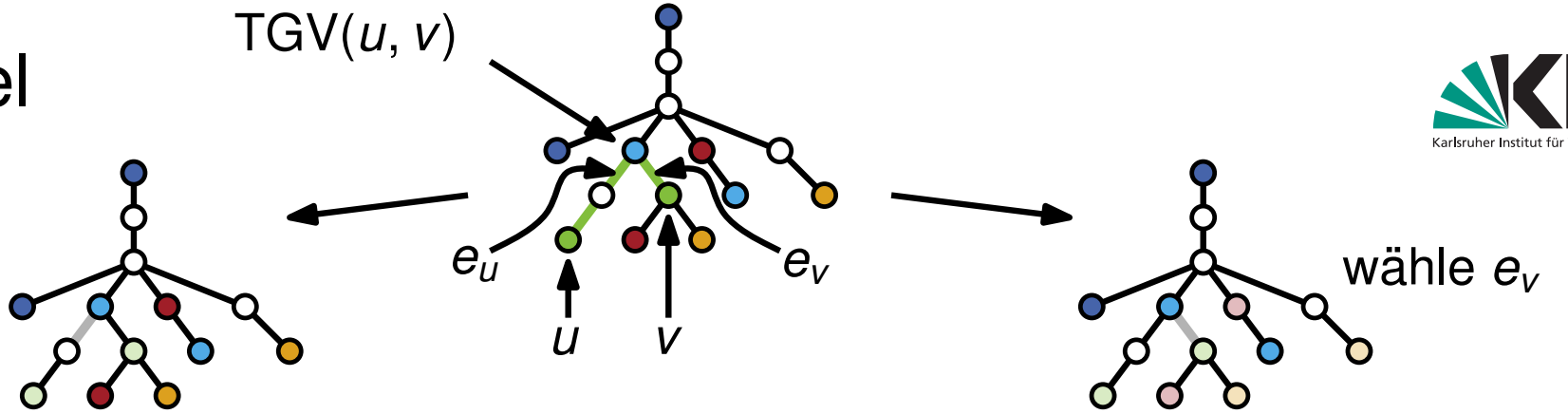


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel

$k = 3$

wähle e_u

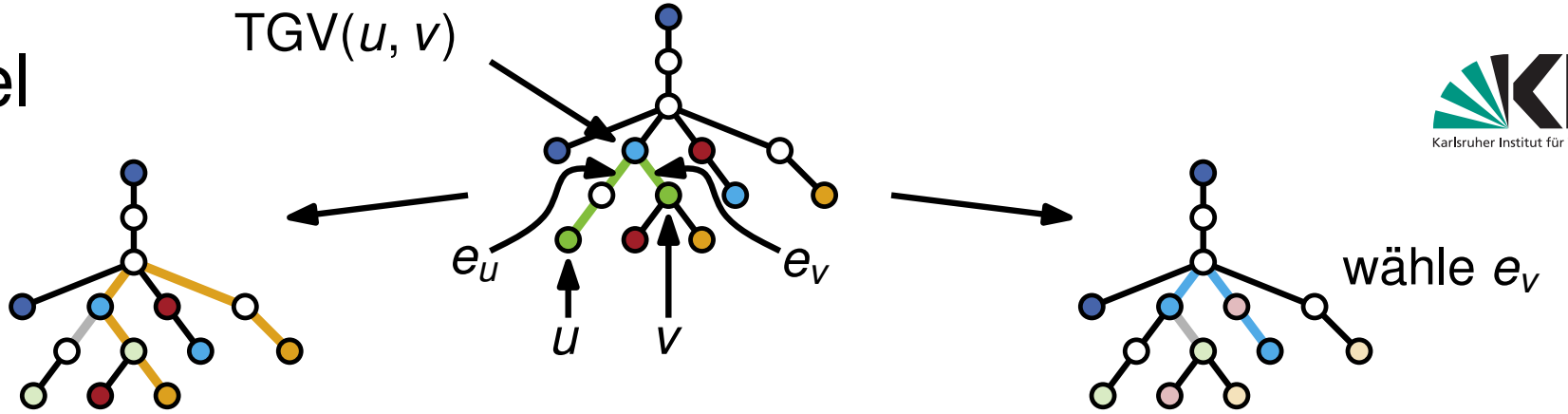


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

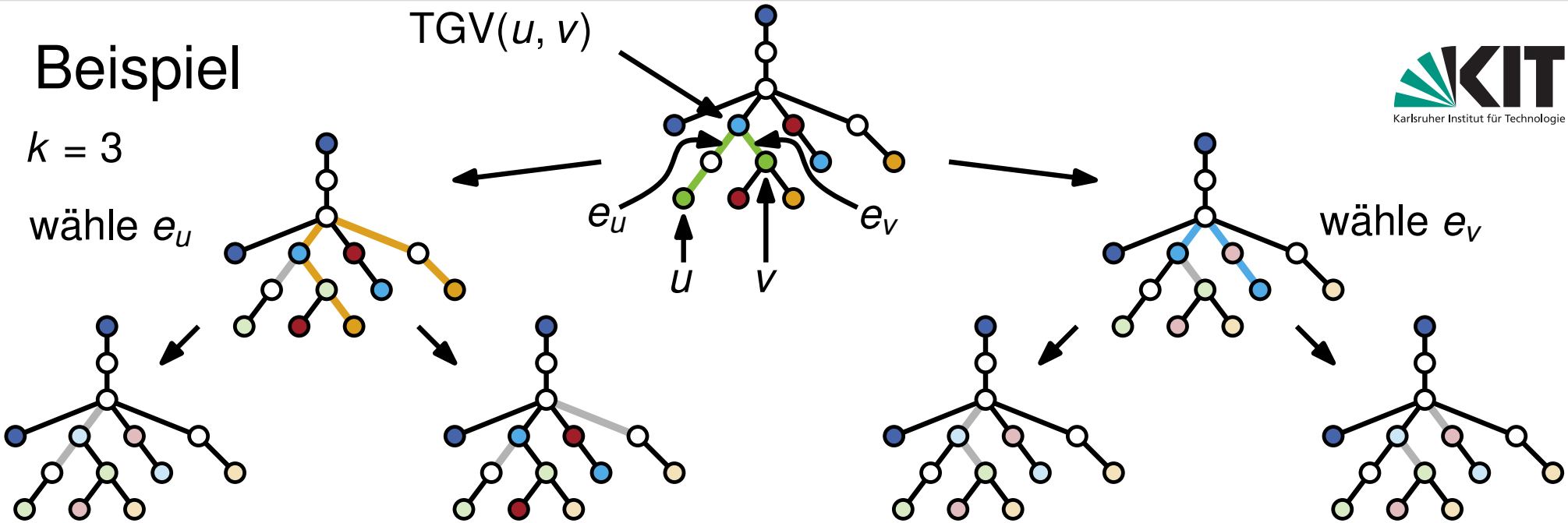
Beispiel

$k = 3$

wähle e_u

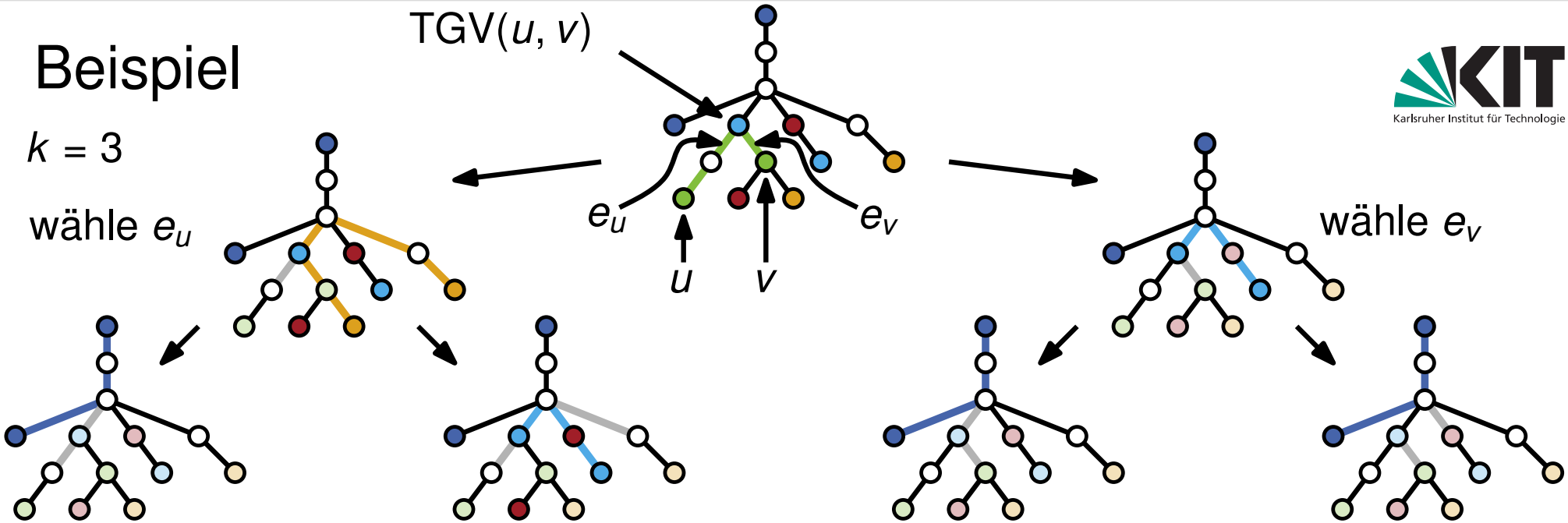


Beispiel



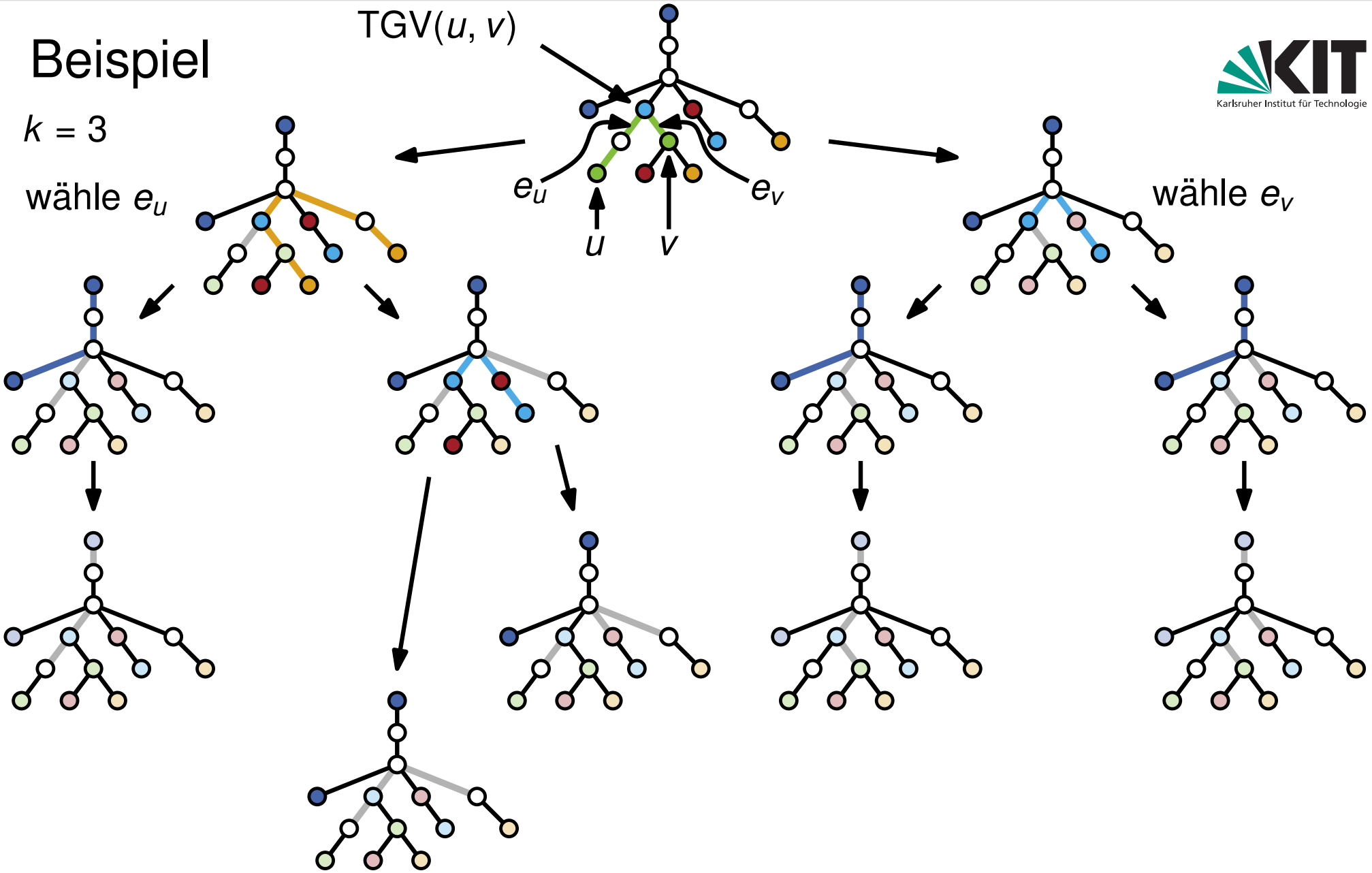
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel



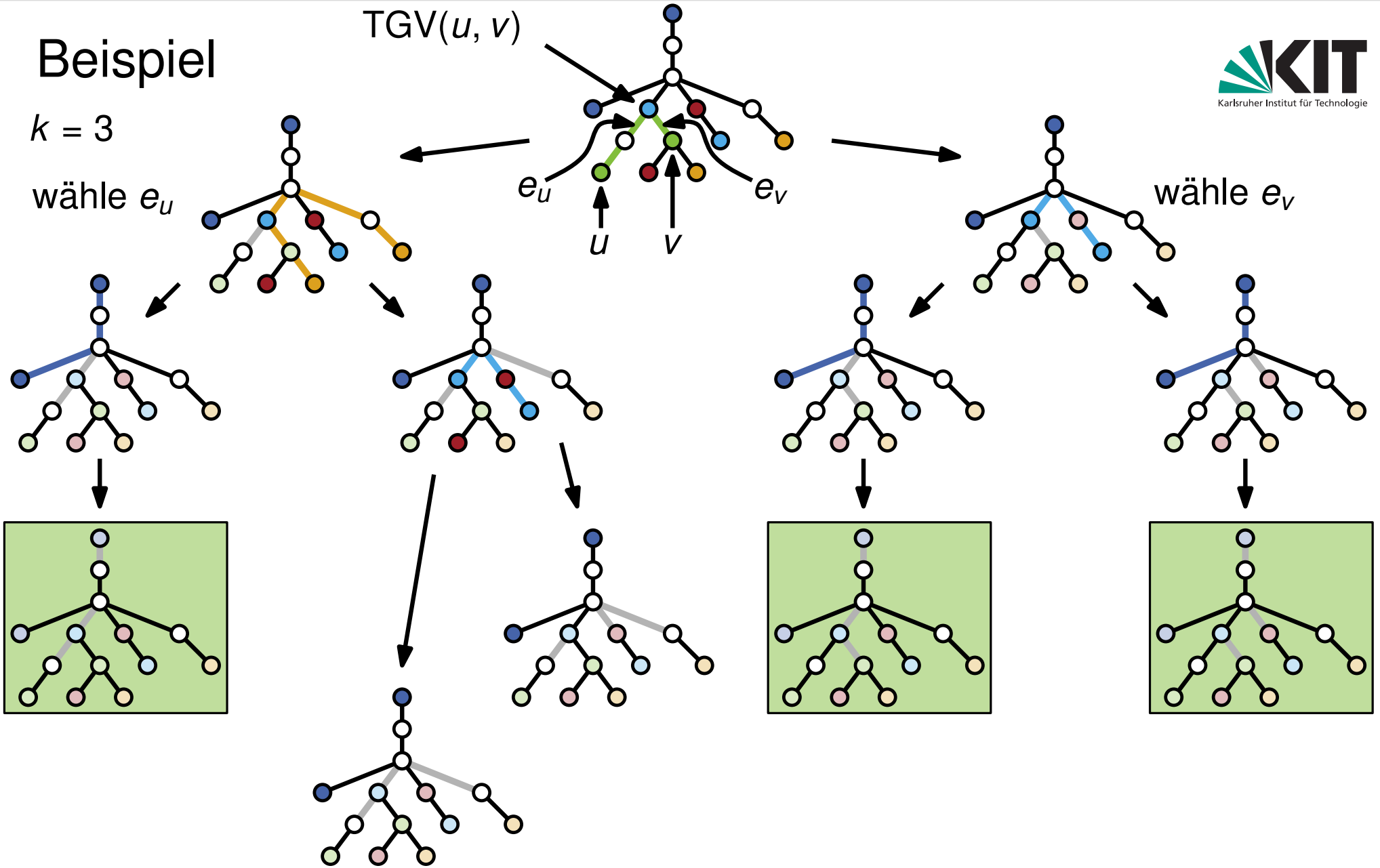
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel



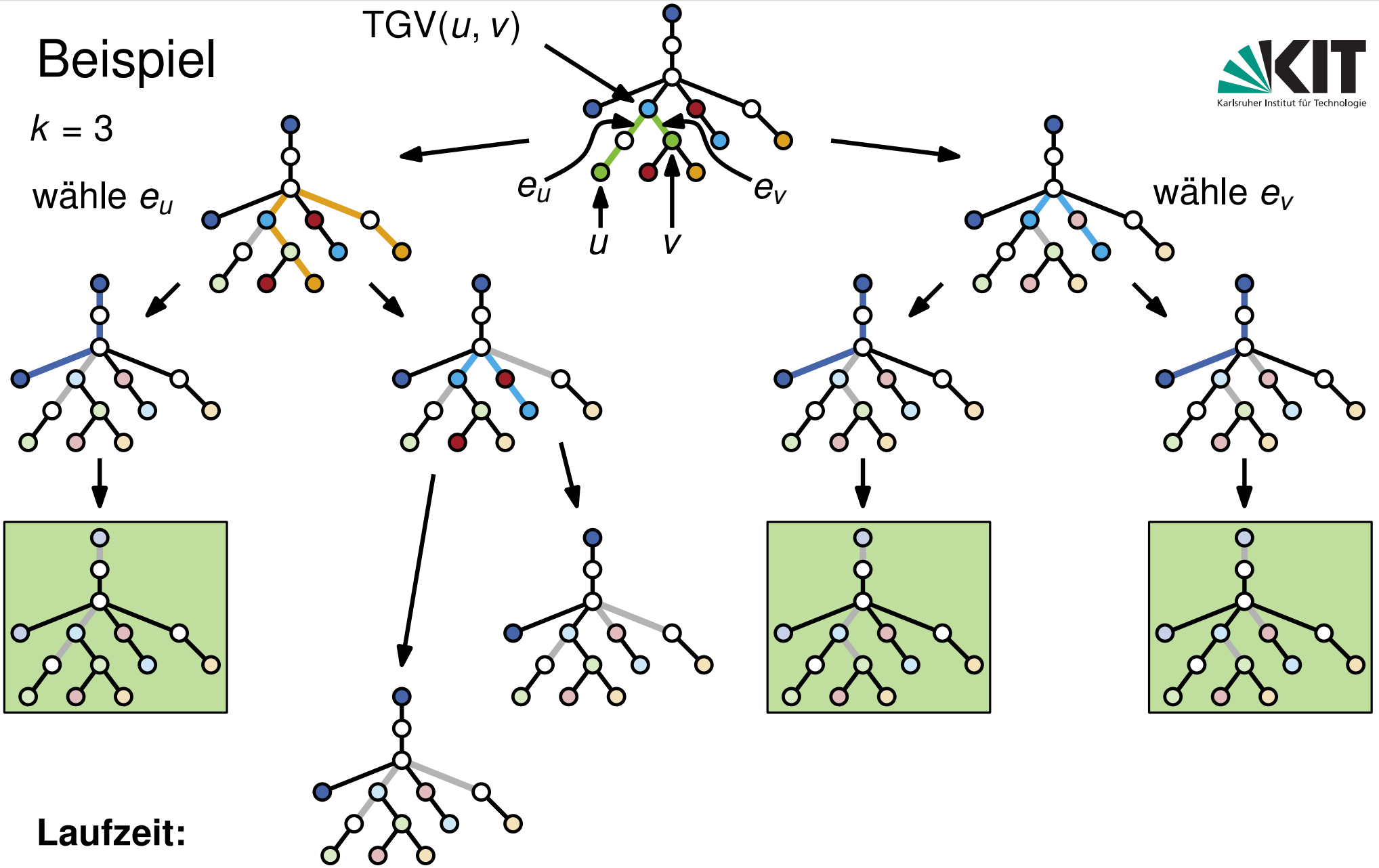
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel



Laufzeit:

- Binärbaum der Tiefe $k \Rightarrow O(2^k)$ viele Schritte.
- Laufzeit $O(n^2)$ pro Schritt. $\Rightarrow$ Gesamtlaufzeit $O(2^k \cdot n^2)$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

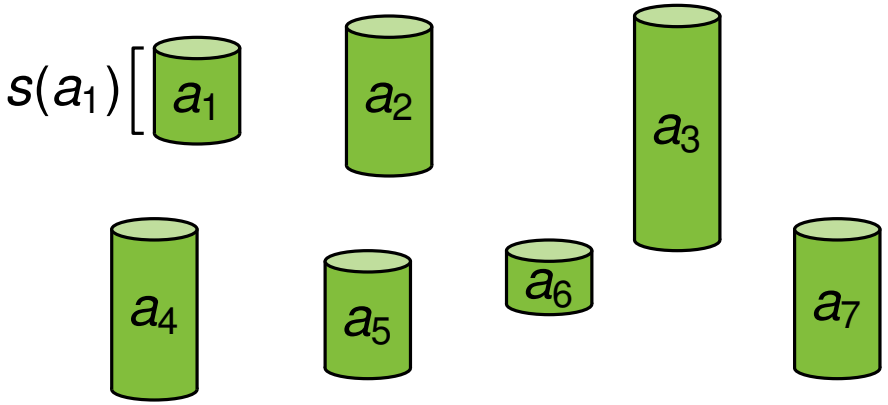
Karlsruher Institut für Technologie

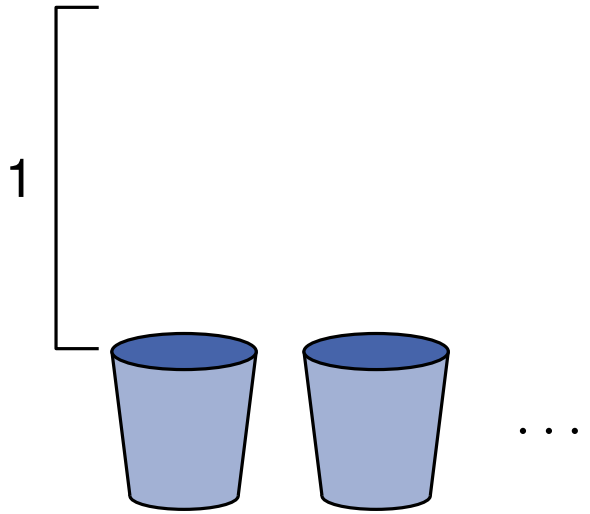
Online-Algorithmen

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Bin Packing – Definition

endliche Menge $M = \{a_1, \ldots, a_n\}$
mit Gewichtsfunktion $s\colon M \longrightarrow (0, 1]$





Eimer (Bins) mit Fassungsvermögen 1

Problem: BIN PACKING
Weise die Elemente in M einer minimalen Anzahl an Bins $B_1, \ldots, B_m$ zu, sodass für jeden Bin B gilt:

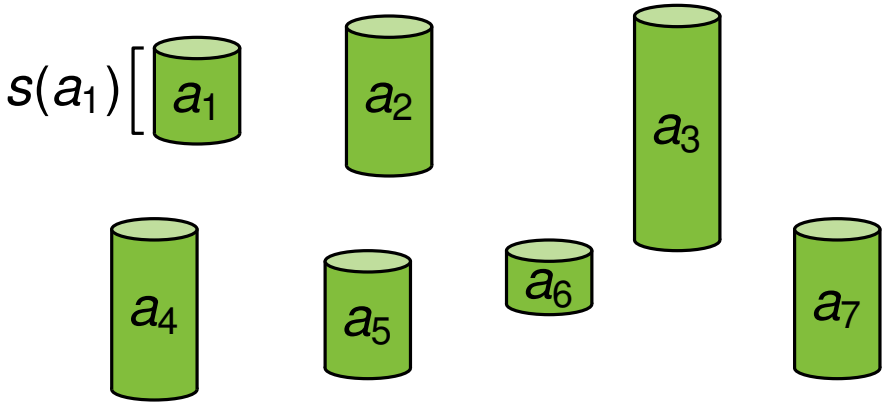
$$\sum_{a_i \in B} s(a_i) \leq 1$$

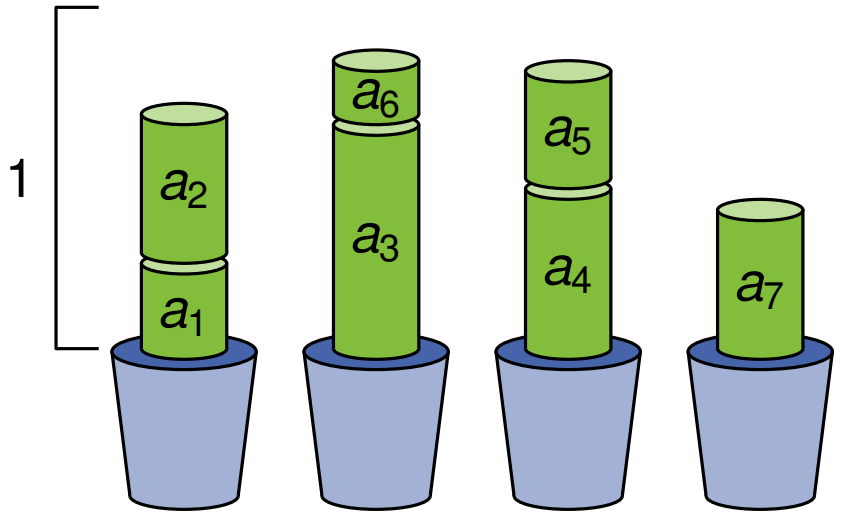
BIN PACKING ist $\mathcal{NP}$-schwer.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Bin Packing – Definition

endliche Menge $M = \{a_1, \ldots, a_n\}$
mit Gewichtsfunktion $s\colon M \longrightarrow (0, 1]$





Eimer (Bins) mit Fassungsvermögen 1

4 Bins

Problem: BIN PACKING
Weise die Elemente in M einer minimalen Anzahl an Bins $B_1, \ldots, B_m$ zu, sodass für jeden Bin B gilt:

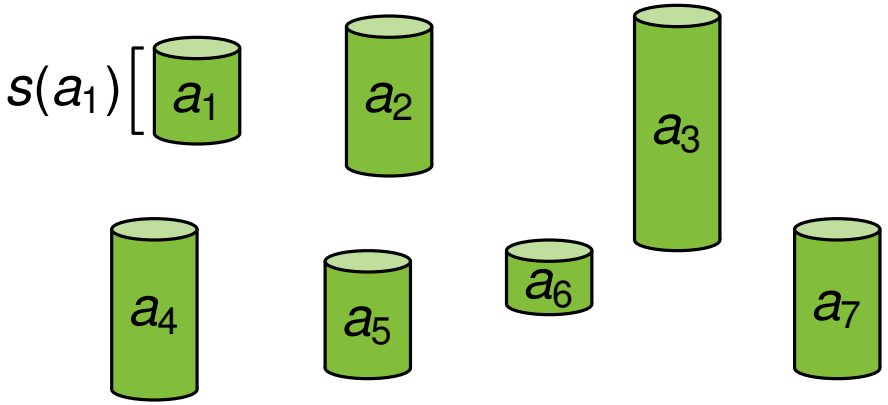
$$\sum_{a_i \in B} s(a_i) \leq 1$$

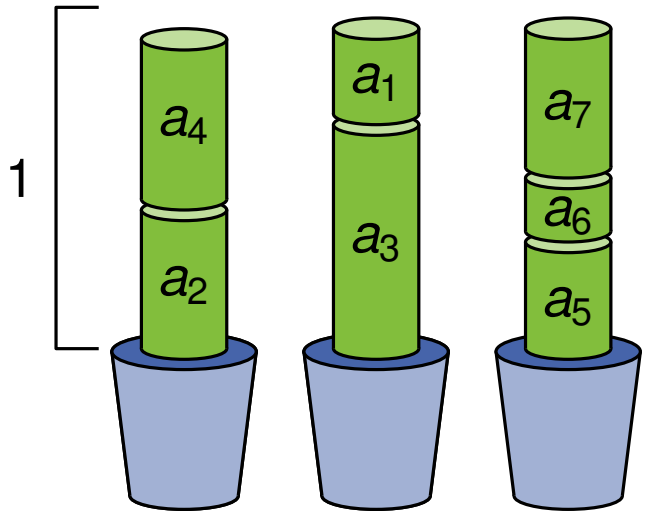
BIN PACKING ist $\mathcal{NP}$-schwer.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Bin Packing – Definition

endliche Menge $M = \{a_1, \ldots, a_n\}$
mit Gewichtsfunktion $s\colon M \longrightarrow (0, 1]$





Eimer (Bins) mit Fassungsvermögen 1

3 Bins

Problem: BIN PACKING
Weise die Elemente in M einer minimalen Anzahl an Bins $B_1, \ldots, B_m$ zu, sodass für jeden Bin B gilt:

$$\sum_{a_i \in B} s(a_i) \leq 1$$

BIN PACKING ist $\mathcal{NP}$-schwer.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Um BIN PACKING als Online-Problem aufzufassen, nehmen Sie nun an, dass M als Sequenz $(a_1, \ldots, a_n)$ gegeben ist. Betrachten Sie nun die Klasse $\mathcal{C}$ der Online-Algorithmen, die M in solcher Weise sequentiell abarbeiten, dass sie immer zuerst das aktuelle Element einem Bin zuweisen, bevor sie das nächste Element betrachten.

(b) Geben Sie einen 2-kompetitiven Algorithmus $\mathcal{A} \in C$ an.

Problem 2

Um BIN PACKING als Online-Problem aufzufassen, nehmen Sie nun an, dass M als Sequenz $(a_1, \ldots, a_n)$ gegeben ist. Betrachten Sie nun die Klasse $\mathcal{C}$ der Online-Algorithmen, die M in solcher Weise sequentiell abarbeiten, dass sie immer zuerst das aktuelle Element einem Bin zuweisen, bevor sie das nächste Element betrachten.

(b) Geben Sie einen 2-kompetitiven Algorithmus $\mathcal{A} \in C$ an.

Lösung: `NextFit`

Lösungsstrategie 1 – Next Fit

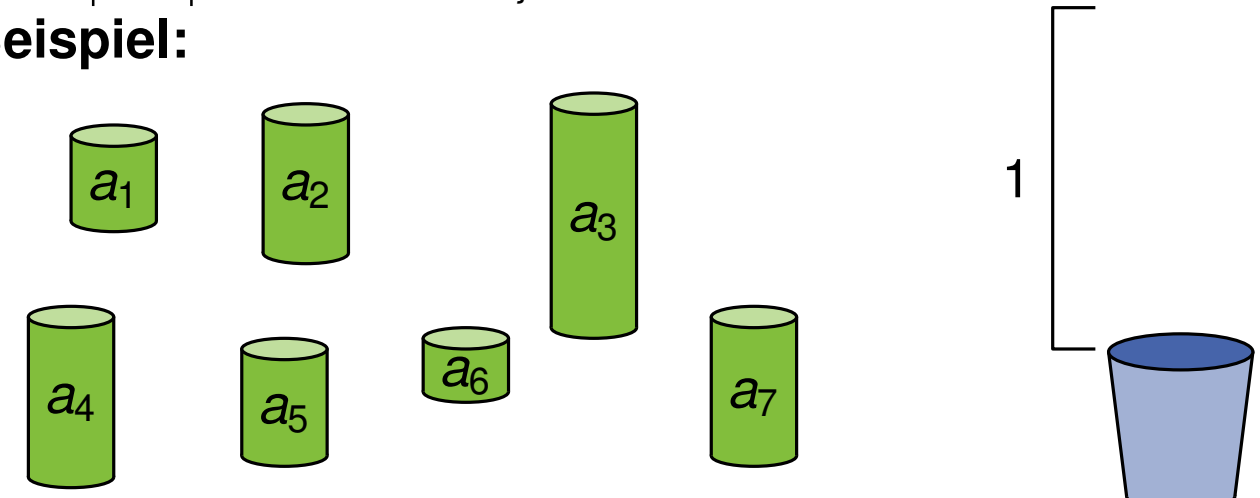
Strategie:

Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

NEXT FIT (NF)(M, s) — Laufzeit: $O(n)$

```
Füge a_1 in B_1 ein
for a_ℓ ∈ {a_2, ..., a_n} do
    B_j ← letzter nicht-leerer Bin
    if s(a_ℓ) ≤ 1 − Σ_{a_i ∈ B_j} s(a_i) then
        Füge a_ℓ in B_j ein
    else
        Füge a_ℓ in B_{j+1} ein
```

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 1 – Next Fit

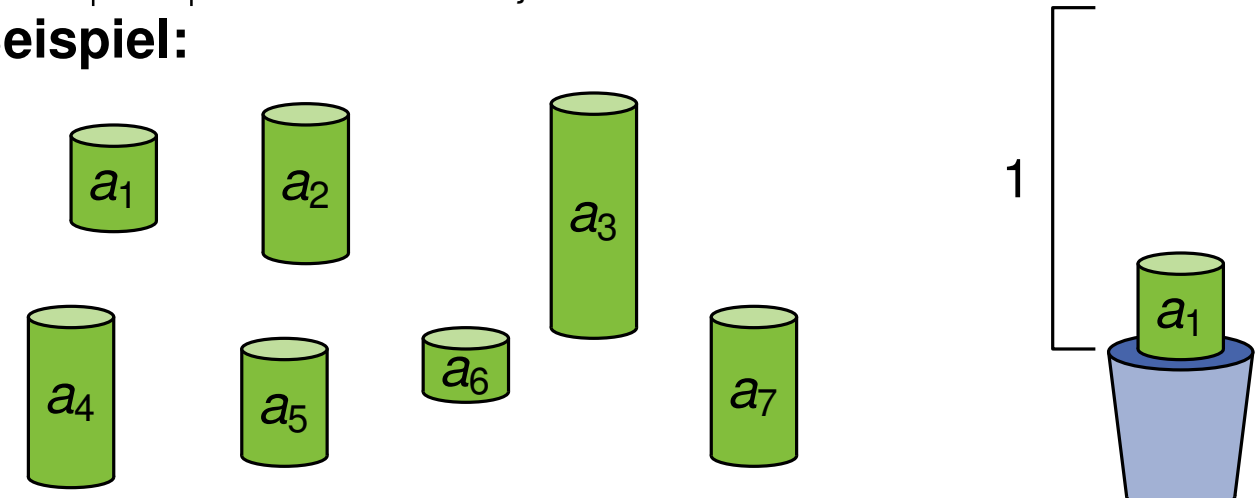
Strategie:

Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

NEXT FIT (NF)(M, s) — Laufzeit: $O(n)$

```
Füge a_1 in B_1 ein
for a_ℓ ∈ {a_2, ..., a_n} do
    B_j ← letzter nicht-leerer Bin
    if s(a_ℓ) ≤ 1 − Σ_{a_i ∈ B_j} s(a_i) then
        Füge a_ℓ in B_j ein
    else
        Füge a_ℓ in B_{j+1} ein
```

Beispiel:



Lösungsstrategie 1 – Next Fit

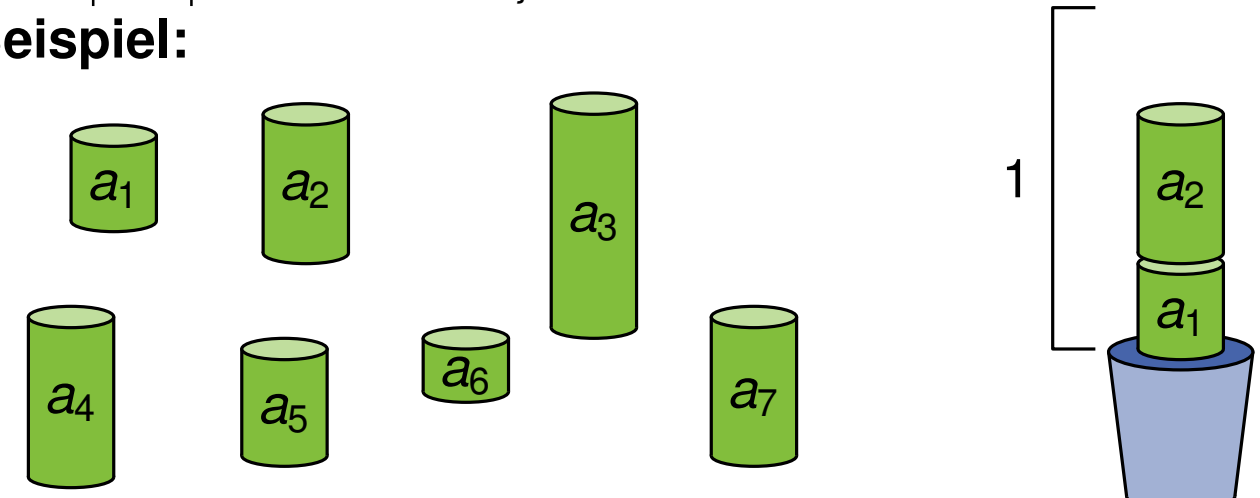
Strategie:

Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

NEXT FIT (NF)(M, s) — Laufzeit: $O(n)$

```
Füge a_1 in B_1 ein
for a_ℓ ∈ {a_2, ..., a_n} do
    B_j ← letzter nicht-leerer Bin
    if s(a_ℓ) ≤ 1 − Σ_{a_i ∈ B_j} s(a_i) then
        Füge a_ℓ in B_j ein
    else
        Füge a_ℓ in B_{j+1} ein
```

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 1 – Next Fit

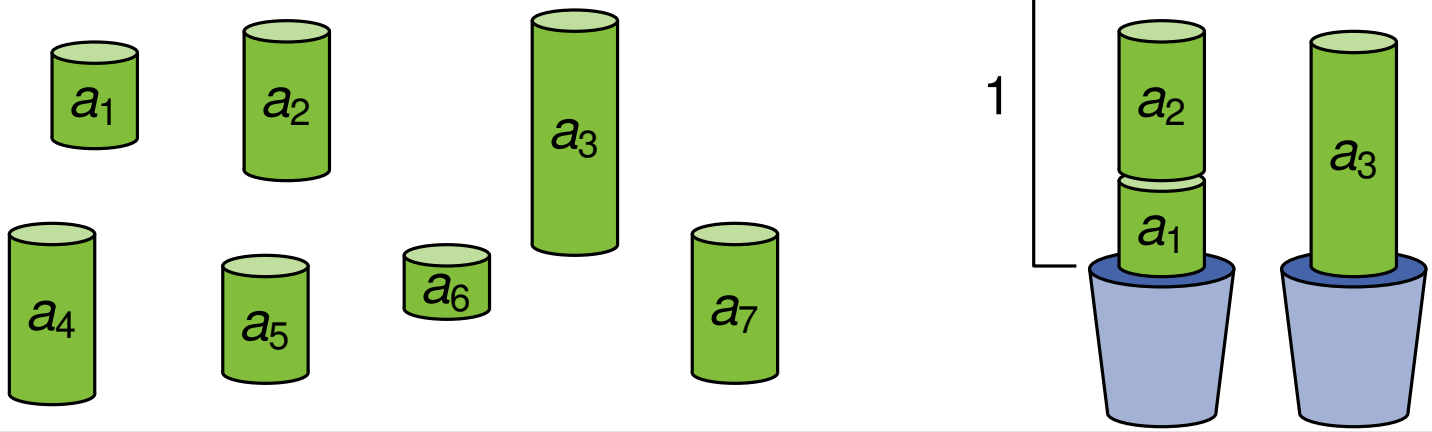
Strategie:

Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

NEXT FIT (NF)(M, s) — Laufzeit: $O(n)$

```
Füge a_1 in B_1 ein
for a_ℓ ∈ {a_2, ..., a_n} do
    B_j ← letzter nicht-leerer Bin
    if s(a_ℓ) ≤ 1 − Σ_{a_i ∈ B_j} s(a_i) then
        Füge a_ℓ in B_j ein
    else
        Füge a_ℓ in B_{j+1} ein
```

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 1 – Next Fit

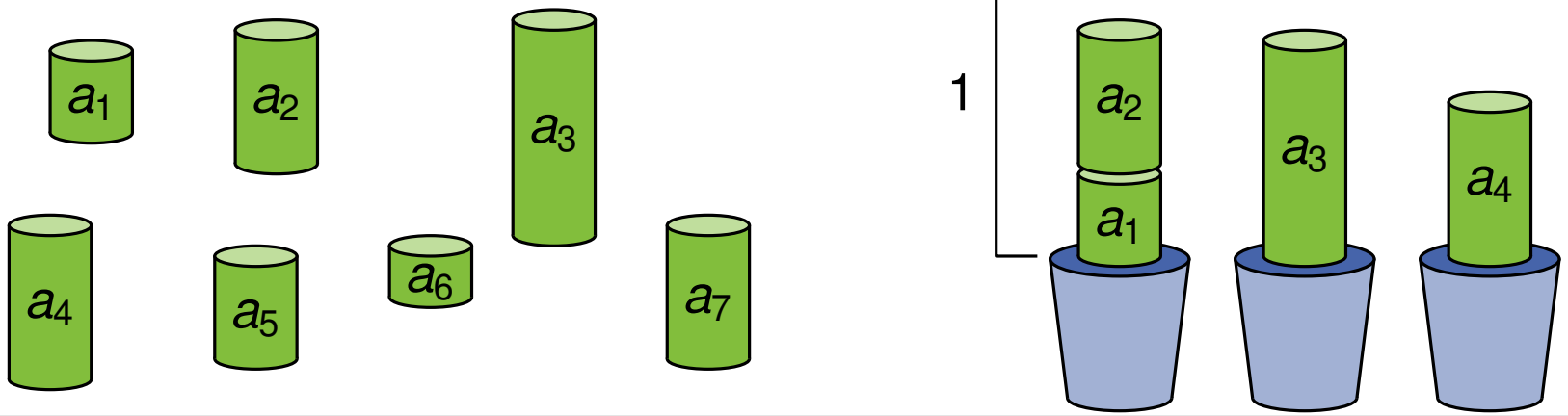
Strategie:

Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

NEXT FIT (NF)(M, s) — Laufzeit: $O(n)$

```
Füge a_1 in B_1 ein
for a_ℓ ∈ {a_2, ..., a_n} do
    B_j ← letzter nicht-leerer Bin
    if s(a_ℓ) ≤ 1 − Σ_{a_i ∈ B_j} s(a_i) then
        Füge a_ℓ in B_j ein
    else
        Füge a_ℓ in B_{j+1} ein
```

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 1 – Next Fit

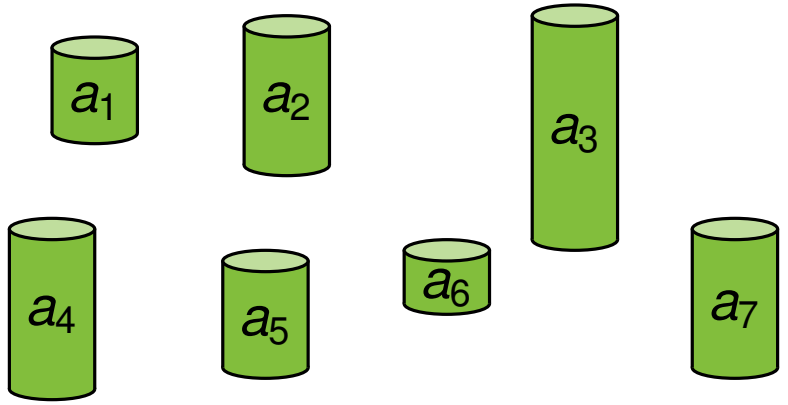
Strategie:

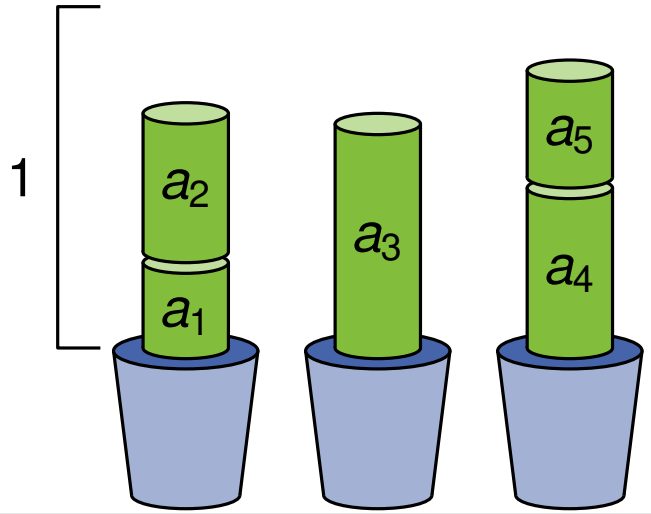
Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

NEXT FIT (NF)(M, s) — Laufzeit: $O(n)$

```
Füge a_1 in B_1 ein
for a_ℓ ∈ {a_2, ..., a_n} do
    B_j ← letzter nicht-leerer Bin
    if s(a_ℓ) ≤ 1 − Σ_{a_i ∈ B_j} s(a_i) then
        Füge a_ℓ in B_j ein
    else
        Füge a_ℓ in B_{j+1} ein
```

Beispiel:





Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 1 – Next Fit

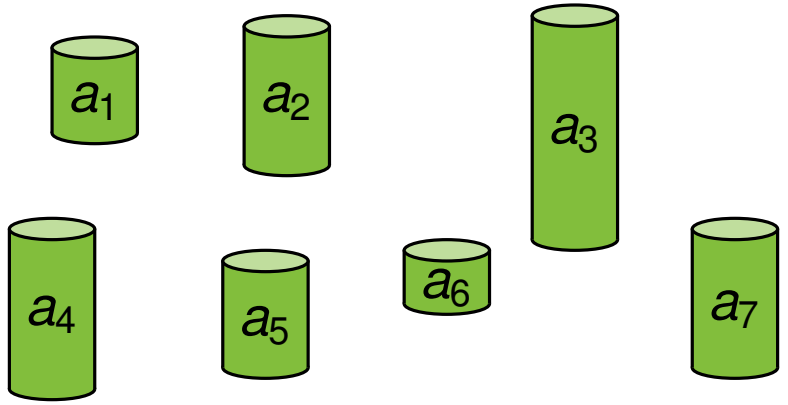
Strategie:

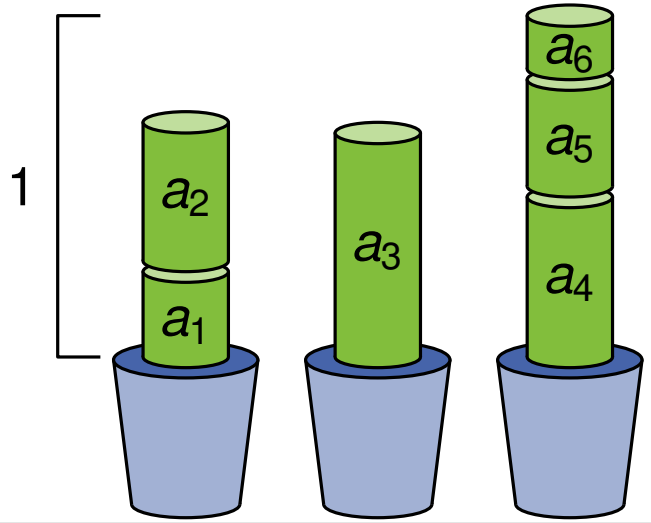
Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

NEXT FIT (NF)(M, s) — Laufzeit: $O(n)$

```
Füge a_1 in B_1 ein
for a_ℓ ∈ {a_2, ..., a_n} do
    B_j ← letzter nicht-leerer Bin
    if s(a_ℓ) ≤ 1 − Σ_{a_i ∈ B_j} s(a_i) then
        Füge a_ℓ in B_j ein
    else
        Füge a_ℓ in B_{j+1} ein
```

Beispiel:





Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 1 – Next Fit

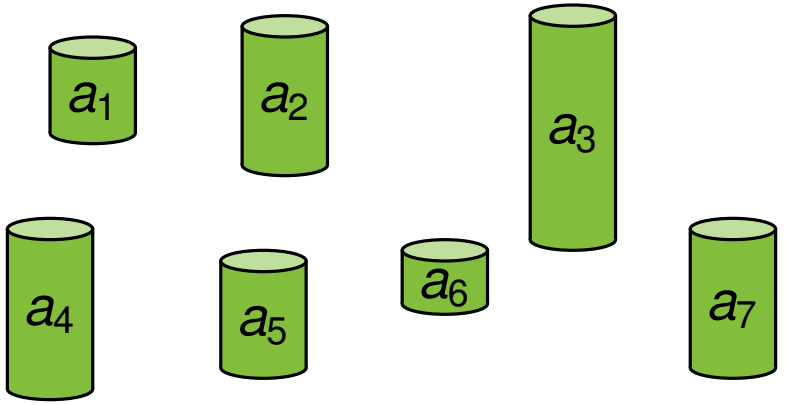
Strategie:

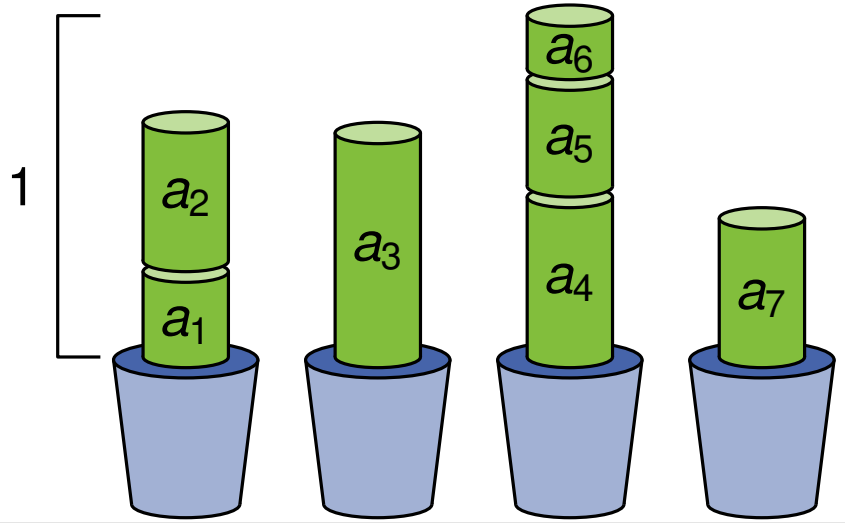
Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

NEXT FIT (NF)(M, s) — Laufzeit: $O(n)$

```
Füge a_1 in B_1 ein
for a_ℓ ∈ {a_2, ..., a_n} do
    B_j ← letzter nicht-leerer Bin
    if s(a_ℓ) ≤ 1 − Σ_{a_i ∈ B_j} s(a_i) then
        Füge a_ℓ in B_j ein
    else
        Füge a_ℓ in B_{j+1} ein
```

Beispiel:





Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Um BIN PACKING als Online-Problem aufzufassen, nehmen Sie nun an, dass M als Sequenz $(a_1, \ldots, a_n)$ gegeben ist. Betrachten Sie nun die Klasse $\mathcal{C}$ der Online-Algorithmen, die M in solcher Weise sequentiell abarbeiten, dass sie immer zuerst das aktuelle Element einem Bin zuweisen, bevor sie das nächste Element betrachten.

(a) Zeigen Sie, dass für jeden Online-Algorithmus $\mathcal{A} \in \mathcal{C}$ eine Instanz I gefunden werden kann, sodass

$$\mathcal{A}(I) \geq \frac{4}{3} \mathrm{OPT}(I),$$

wobei $\mathcal{A}(I)$ den Wert der Lösung von $\mathcal{A}$ angewendet auf I und $\mathrm{OPT}(I)$ den Wert der optimalen Lösung bezeichnet.

Problem 2

(a) Zeigen Sie, dass für jeden Online-Algorithmus $\mathcal{A} \in \mathcal{C}$ eine Instanz I gefunden werden kann, sodass

$$\mathcal{A}(I) \geq \frac{4}{3}\,\mathrm{OPT}(I),$$

wobei $\mathcal{A}(I)$ den Wert der Lösung von $\mathcal{A}$ angewendet auf I und $\mathrm{OPT}(I)$ den Wert der optimalen Lösung bezeichnet.

Annahme: Es gibt Algorithmus $\mathcal{A}$, so dass für jede mögliche Instanz gilt:

$$\mathcal{A}(I) < \frac{4}{3}\,\mathrm{OPT}(I)$$

Beobachtung: Da nicht bekannt ist, wann Sequenz aufhört (Online-Algorithmus), muss die Aussage zu jedem Zeitpunkt gelten, wenn Sequenz $(a_1, \ldots, a_n)$ von $\mathcal{A}$ abgearbeitet wird.

Problem 2

(a) Zeigen Sie, dass für jeden Online-Algorithmus $\mathcal{A} \in \mathcal{C}$ eine Instanz I gefunden werden kann, sodass

$$\mathcal{A}(I) \geq \frac{4}{3}\,\mathrm{OPT}(I),$$

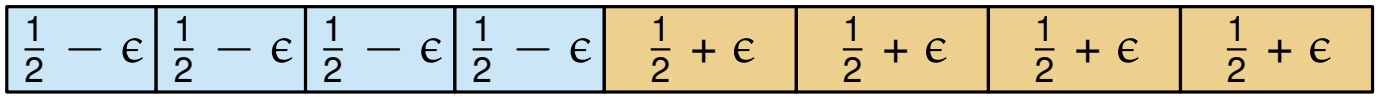
wobei $\mathcal{A}(I)$ den Wert der Lösung von $\mathcal{A}$ angewendet auf I und $\mathrm{OPT}(I)$ den Wert der optimalen Lösung bezeichnet.

Annahme: Es gibt Algorithmus $\mathcal{A}$, so dass für jede mögliche Instanz gilt:

$$\mathcal{A}(I) < \frac{4}{3}\,\mathrm{OPT}(I)$$

Beobachtung: Da nicht bekannt ist, wann Sequenz aufhört (Online-Algorithmus), muss die Aussage zu jedem Zeitpunkt gelten, wenn Sequenz $(a_1, \ldots, a_n)$ von $\mathcal{A}$ abgearbeitet wird.

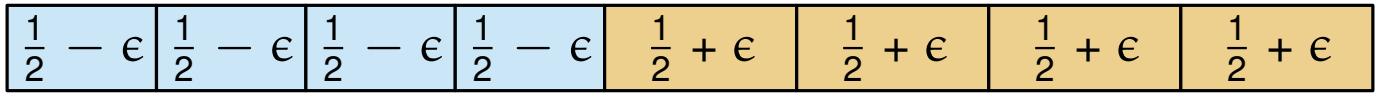
Idee: Finde Sequenz $(a_1, \ldots, a_n)$ und Zeitpunkt der Abarbeitung, sodass Annahme widersprüchlich ist.



$\frac{n}{2}$ Elemente der Größe $\frac{1}{2} - \epsilon$ $\frac{n}{2}$ Elemente der Größe $\frac{1}{2} + \epsilon$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

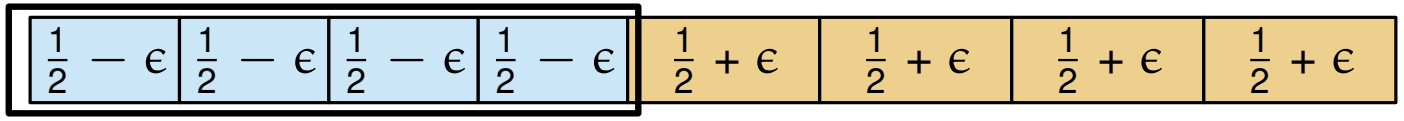
Problem 2



$\frac{n}{2}$ Elemente der Größe $\frac{1}{2} - \epsilon$ $\frac{n}{2}$ Elemente der Größe $\frac{1}{2} + \epsilon$

1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

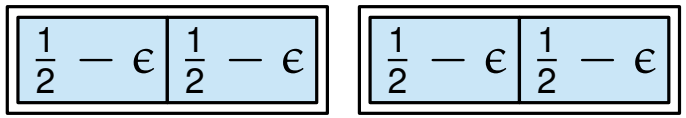
Problem 2



$\frac{n}{2}$ Elemente der Größe $\frac{1}{2} - \epsilon$ $\frac{n}{2}$ Elemente der Größe $\frac{1}{2} + \epsilon$

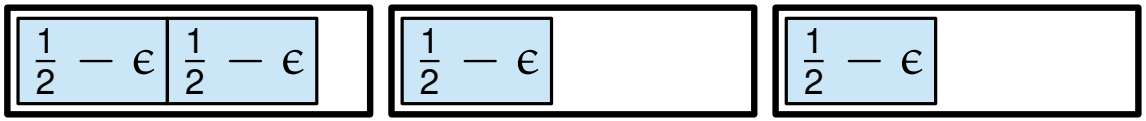
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

Optimale Lösung verwendet $\frac{n}{4}$ Bins:

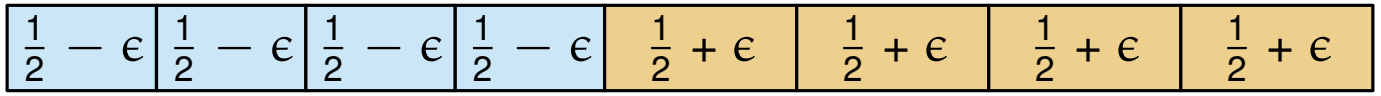


$\mathcal{A}$ verwendet b Bins und damit gilt nach Annahme:

$$\frac{b}{\frac{n}{4}} = \frac{b \cdot 4}{n} < \frac{4}{3} \Leftrightarrow \frac{b}{n} < \frac{1}{3}$$



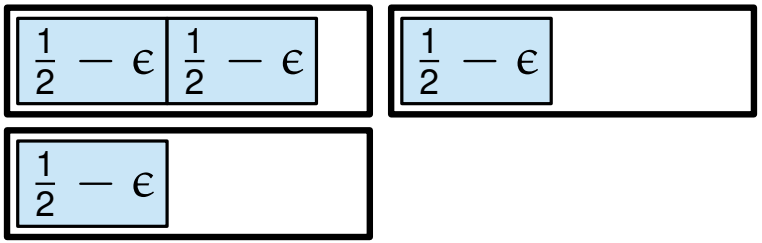
Problem 2



$\frac{n}{2}$ Elemente der Größe $\frac{1}{2} - \epsilon$ $\frac{n}{2}$ Elemente der Größe $\frac{1}{2} + \epsilon$

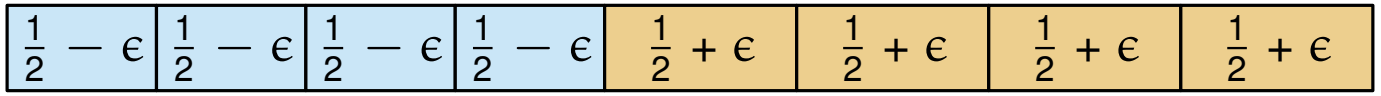
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

$OPT(I) = \frac{1}{2}$ und $\mathcal{A}(I) = b$: $\frac{b}{\frac{n}{4}} = \frac{b \cdot 4}{n} < \frac{4}{3} \Leftrightarrow \frac{b}{n} < \frac{1}{3}$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

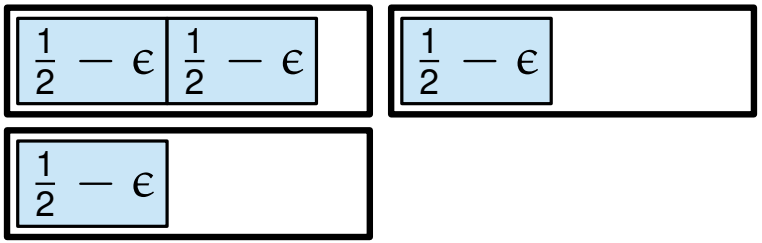
Problem 2



$\frac{n}{2}$ Elemente der Größe $\frac{1}{2} - \epsilon$ $\frac{n}{2}$ Elemente der Größe $\frac{1}{2} + \epsilon$

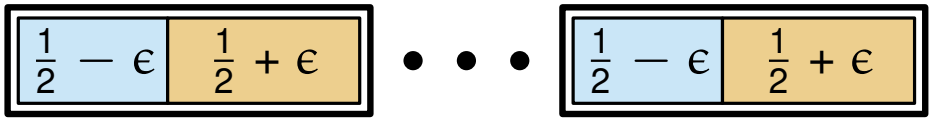
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

$OPT(I) = \frac{1}{2}$ und $\mathcal{A}(I) = b$: $\frac{b}{\frac{n}{4}} = \frac{b \cdot 4}{n} < \frac{4}{3} \Leftrightarrow \frac{b}{n} < \frac{1}{3}$



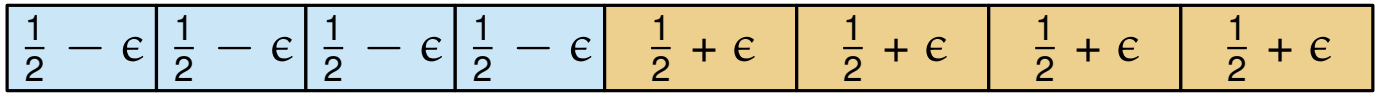
2. Fall: Betrachte Zeitpunkt, nachdem alle Elemente abgeabreitet worden sind:

Optimale Lösung verwendet $\frac{n}{2}$ Bins:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

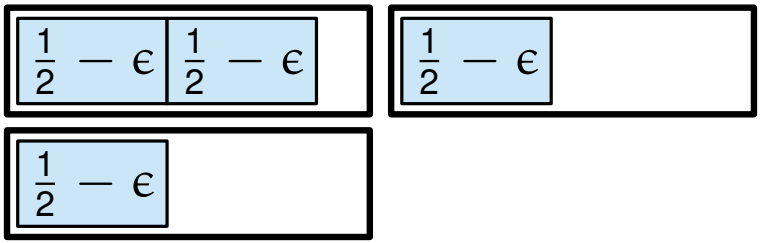
Problem 2



$\frac{n}{2}$ Elemente der Größe $\frac{1}{2} - \epsilon$ $\frac{n}{2}$ Elemente der Größe $\frac{1}{2} + \epsilon$

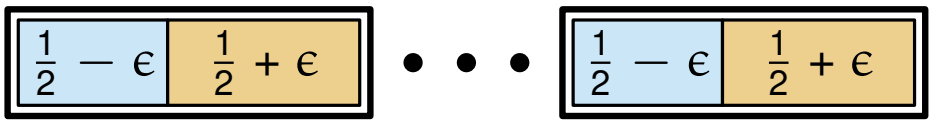
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

$OPT(I) = \frac{1}{2}$ und $\mathcal{A}(I) = b$: $\frac{b}{\frac{n}{4}} = \frac{b \cdot 4}{n} < \frac{4}{3} \Leftrightarrow \frac{b}{n} < \frac{1}{3}$

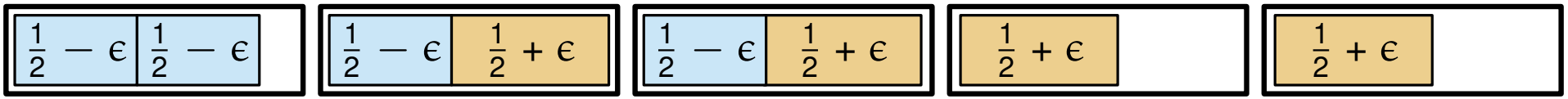


2. Fall: Betrachte Zeitpunkt, nachdem alle Elemente abgeabreitet worden sind:

Optimale Lösung verwendet $\frac{n}{2}$ Bins:

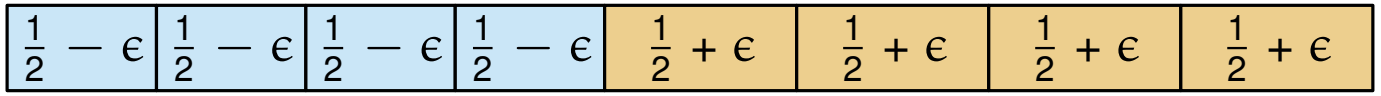


$\mathcal{A}$ kann maximal b Elemente der Größe $\frac{1}{2} + \epsilon$ in die ersten b Bins packen, da diese bereits mit $\frac{1}{2} - \epsilon$ Elementen belegt sind. Die restlichen Bins können jeweils nur ein Element enthalten.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

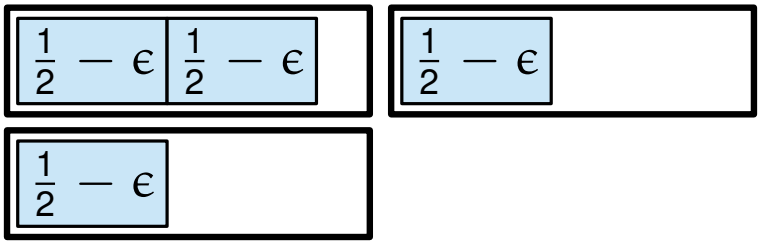
Problem 2



$\frac{n}{2}$ Elemente der Größe $\frac{1}{2} - \epsilon$ $\frac{n}{2}$ Elemente der Größe $\frac{1}{2} + \epsilon$

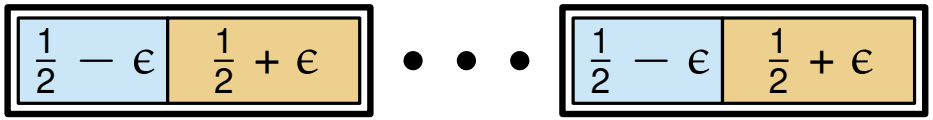
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

$OPT(I) = \frac{1}{2}$ und $\mathcal{A}(I) = b$: $\frac{b}{\frac{n}{4}} = \frac{b \cdot 4}{n} < \frac{4}{3} \Leftrightarrow \frac{b}{n} < \frac{1}{3}$

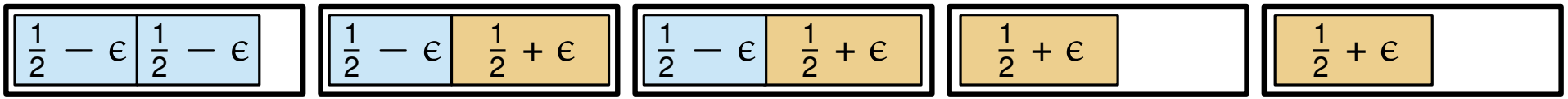


2. Fall: Betrachte Zeitpunkt, nachdem alle Elemente abgeabreitet worden sind:

Optimale Lösung verwendet $\frac{n}{2}$ Bins:



$\mathcal{A}$ kann maximal b Elemente der Größe $\frac{1}{2} + \epsilon$ in die ersten b Bins packen, da diese bereits mit $\frac{1}{2} - \epsilon$ Elementen belegt sind. Die restlichen Bins können jeweils nur ein Element enthalten.

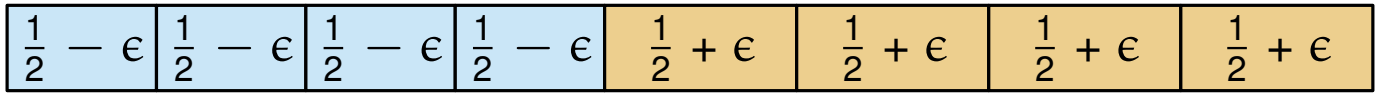


$\mathcal{A}$ benötigt mindestens $n - b$ Bins.

Nach Annahme gilt: $\frac{n-b}{\frac{n}{2}} < \frac{4}{3} \Leftrightarrow \frac{1}{3} < \frac{b}{n}$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

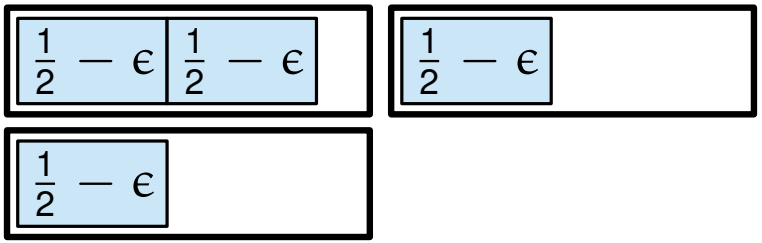
Problem 2



$\frac{n}{2}$ Elemente der Größe $\frac{1}{2} - \epsilon$ $\frac{n}{2}$ Elemente der Größe $\frac{1}{2} + \epsilon$

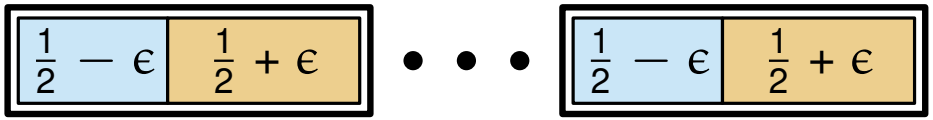
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

$OPT(I) = \frac{1}{2}$ und $\mathcal{A}(I) = b$: $\frac{b}{\frac{n}{4}} = \frac{b \cdot 4}{n} < \frac{4}{3} \Leftrightarrow \frac{b}{n} < \frac{1}{3}$

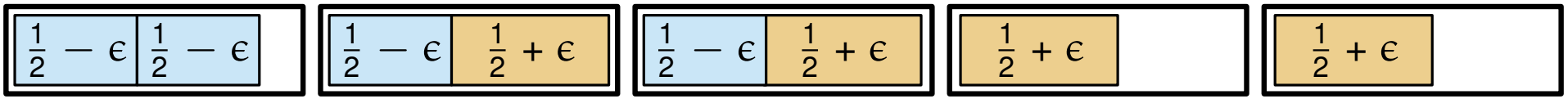


2. Fall: Betrachte Zeitpunkt, nachdem alle Elemente abgeabreitet worden sind:

Optimale Lösung verwendet $\frac{n}{2}$ Bins:



$\mathcal{A}$ kann maximal b Elemente der Größe $\frac{1}{2} + \epsilon$ in die ersten b Bins packen, da diese bereits mit $\frac{1}{2} - \epsilon$ Elementen belegt sind. Die restlichen Bins können jeweils nur ein Element enthalten.



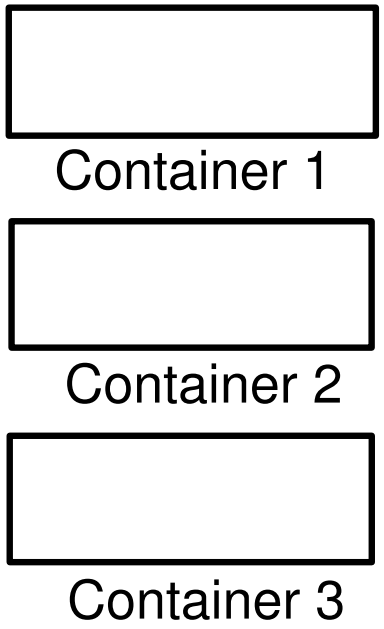
$\mathcal{A}$ benötigt mindestens $n - b$ Bins.

Nach Annahme gilt: $\frac{n-b}{\frac{n}{2}} < \frac{4}{3} \Leftrightarrow \frac{1}{3} < \frac{b}{n}$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

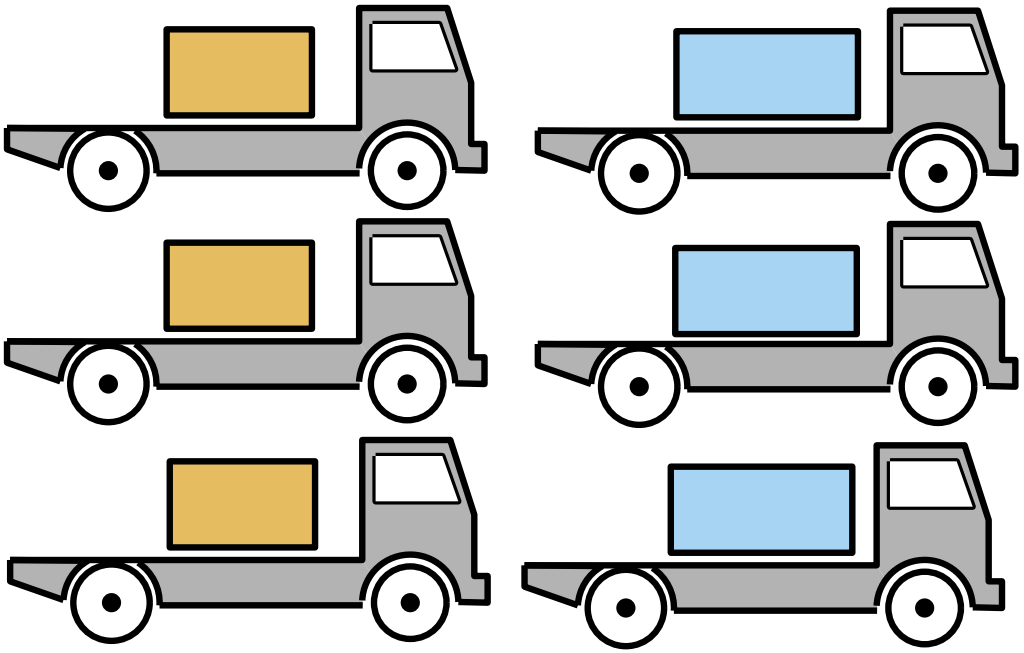
Problem 2

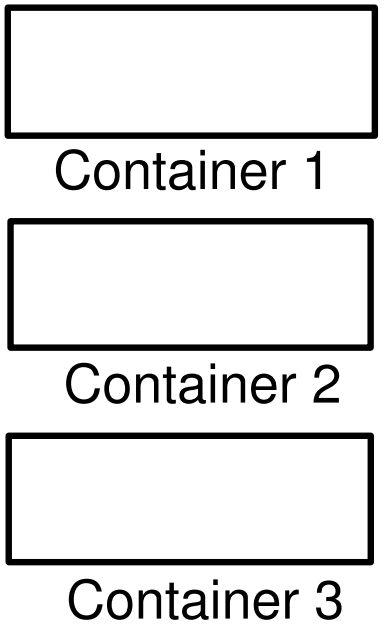
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?



Problem 2

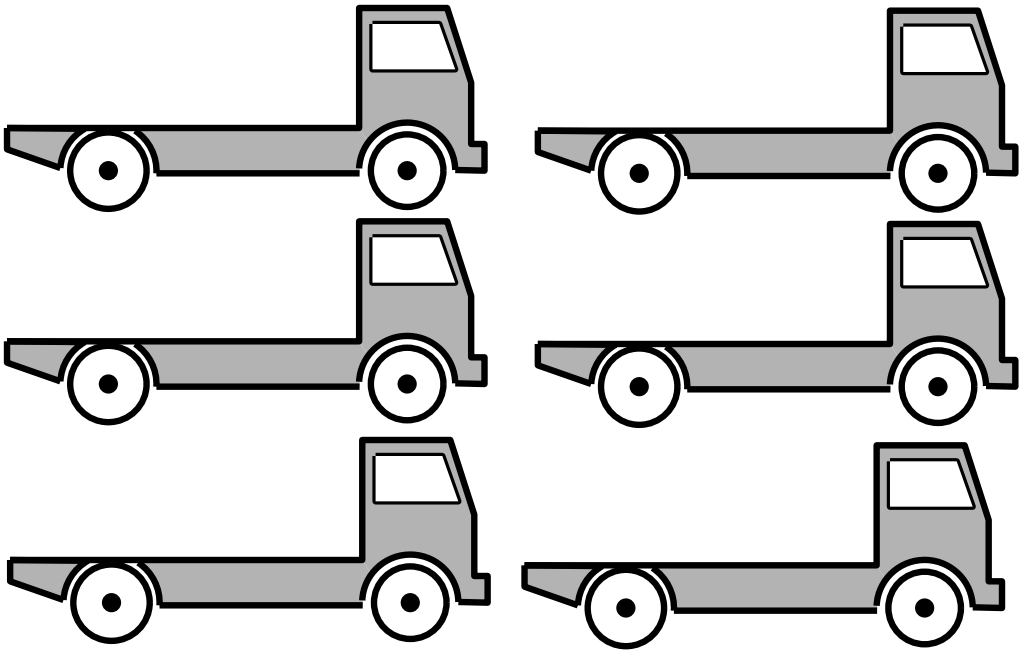
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

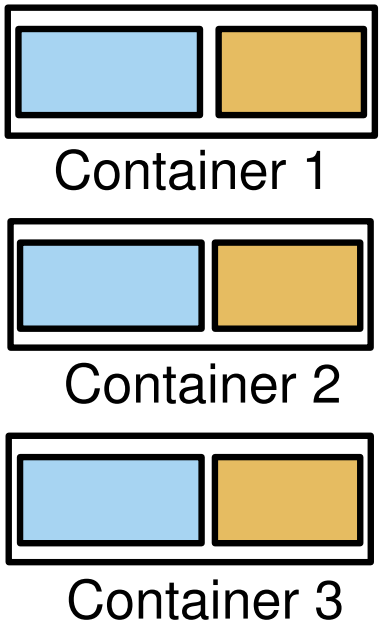




Problem 2

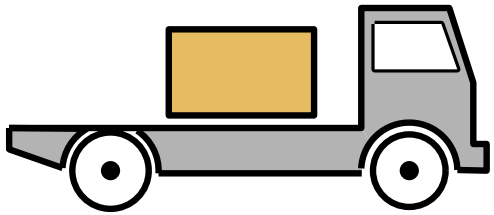
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

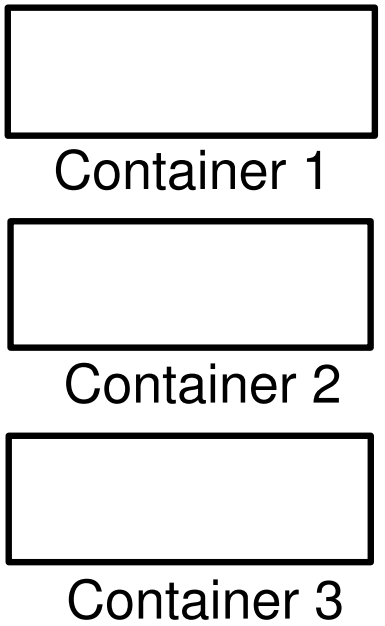




Problem 2

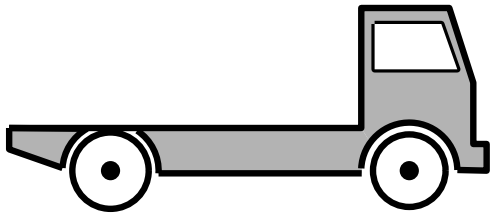
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

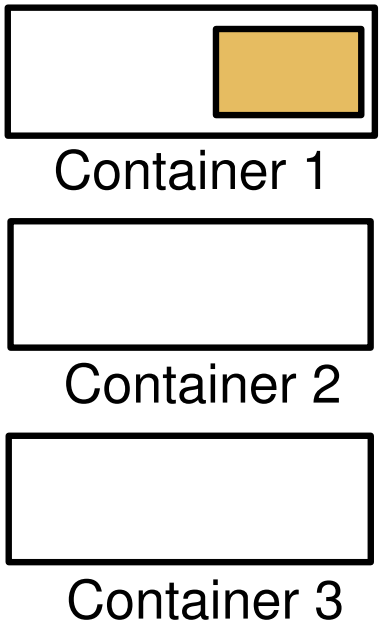




Problem 2

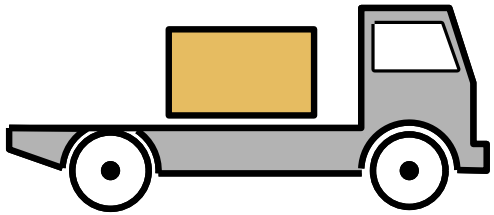
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

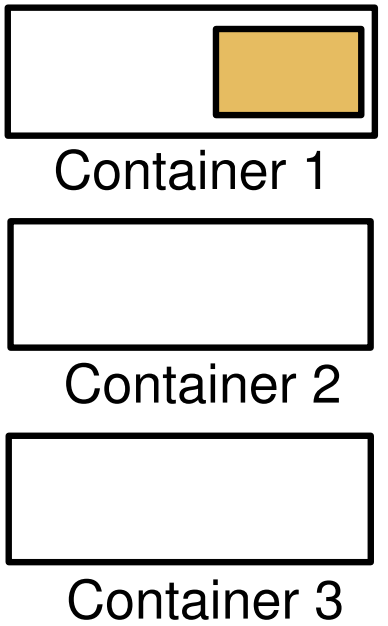




Problem 2

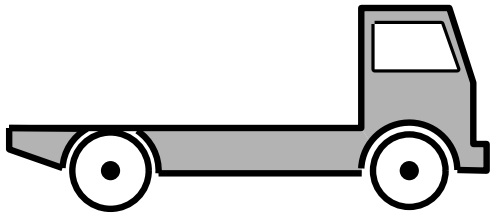
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

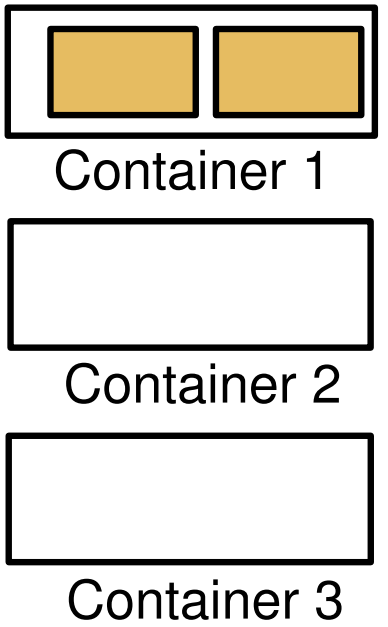




Problem 2

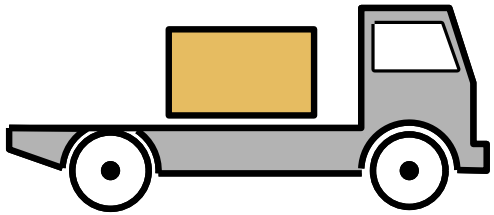
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

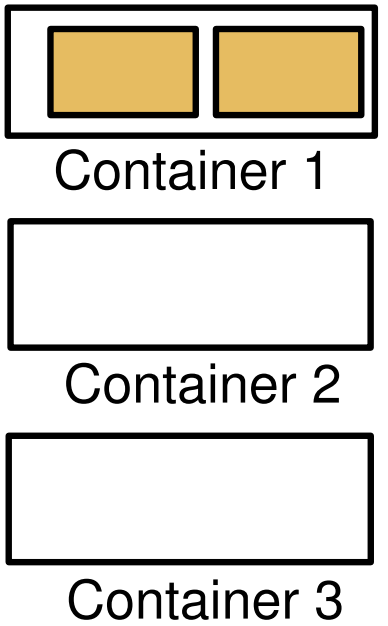




Problem 2

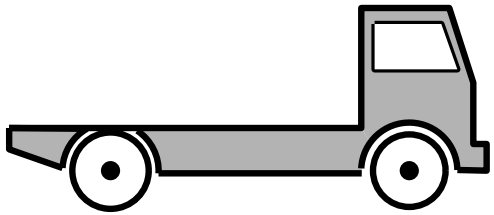
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

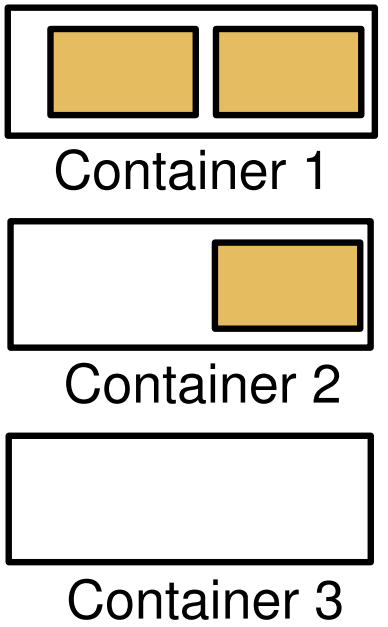




Problem 2

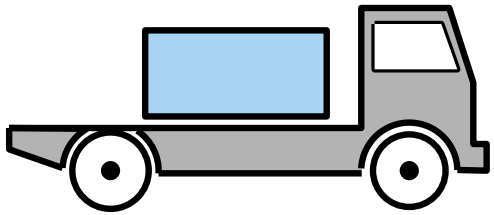
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

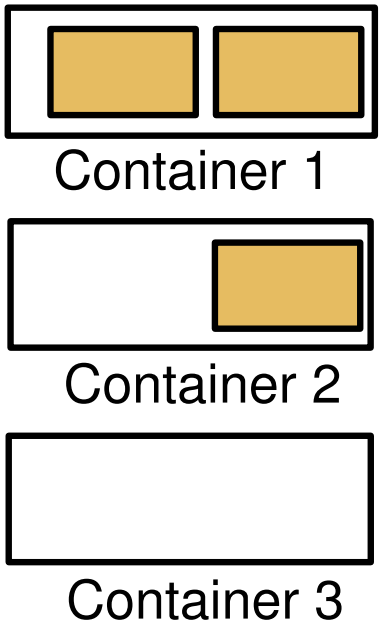




Problem 2

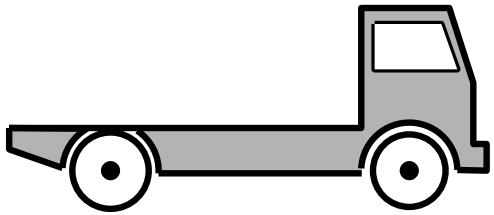
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

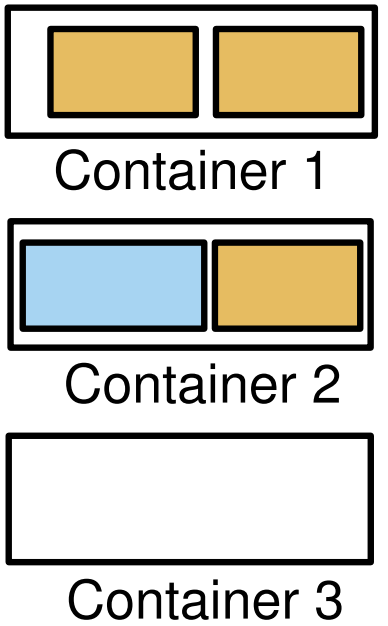




Problem 2

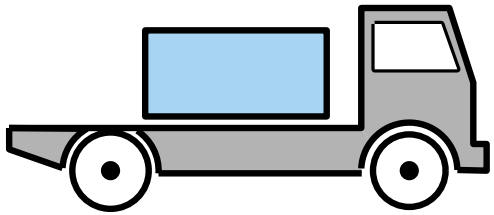
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

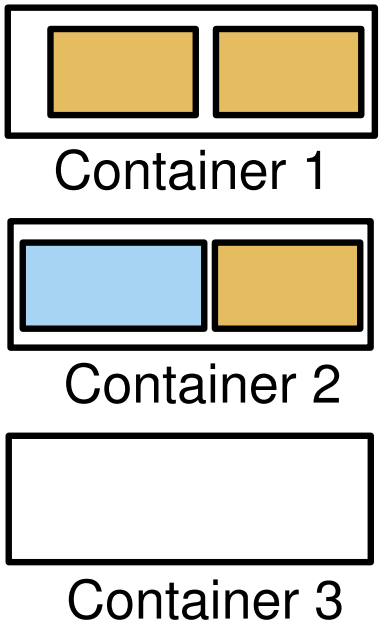




Problem 2

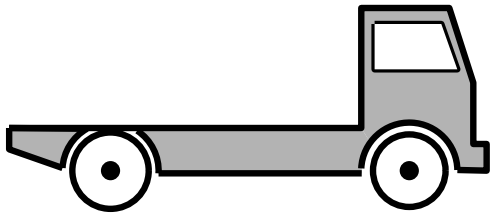
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

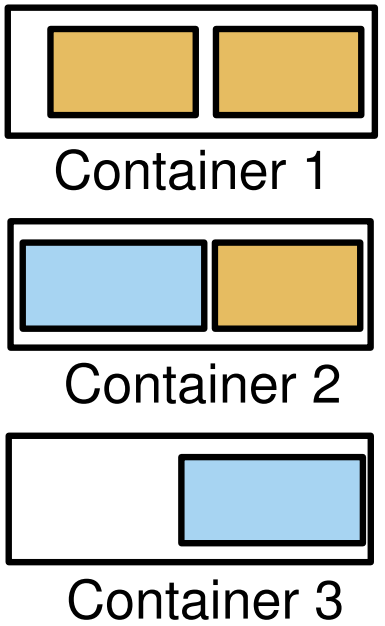




Problem 2

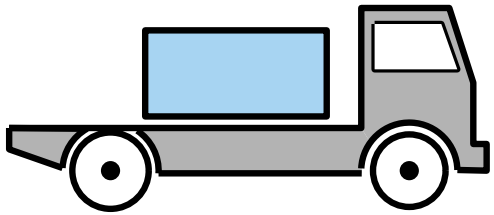
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

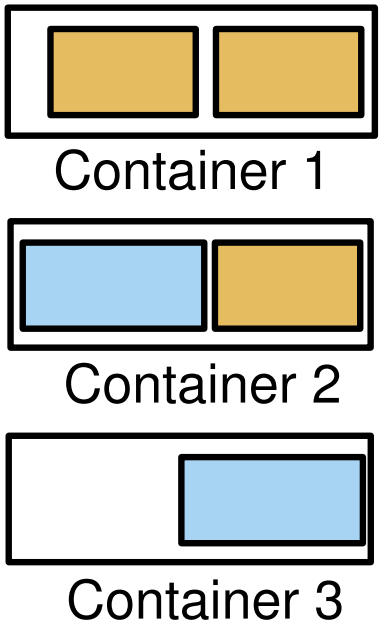




Problem 2

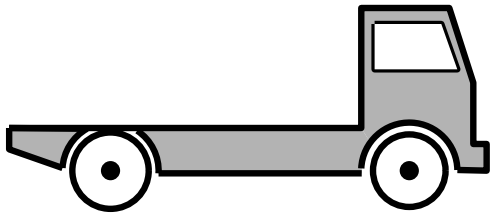
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?

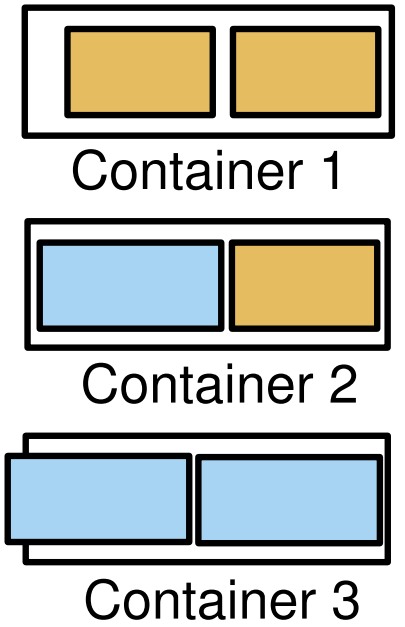




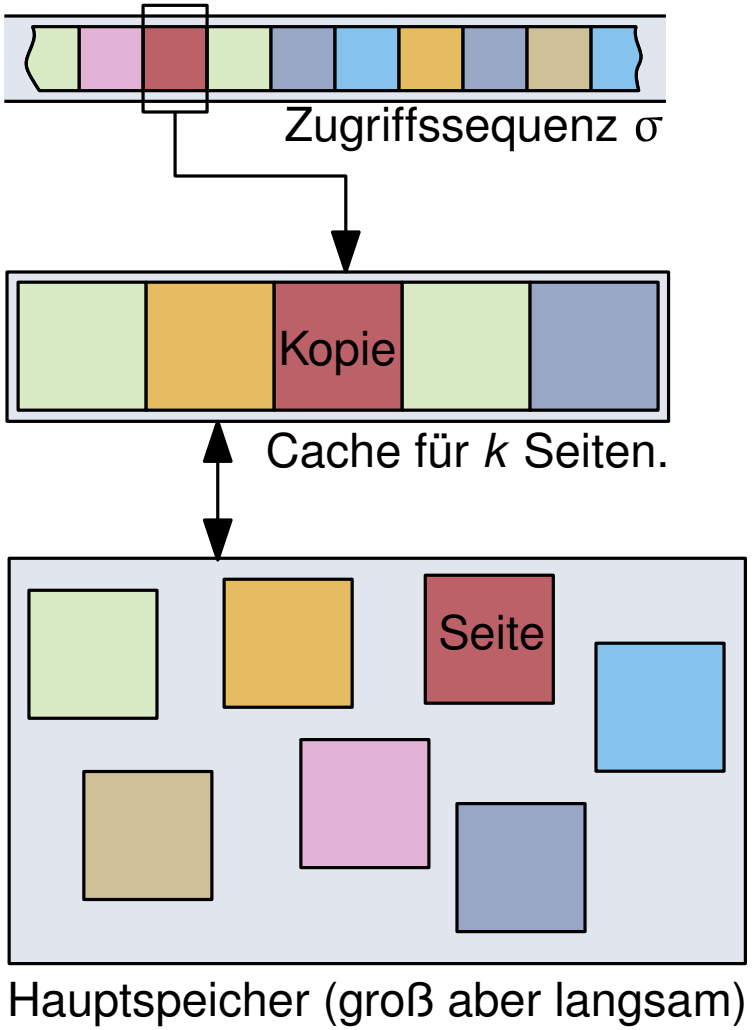
Problem 2

Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu den Algorithmus aus Teilaufgabe b) zu verwenden. Was für Probleme können auftreten?





Paging



- Hauptspeicher enthält N Seiten: $P = \{p_1, \ldots, p_N\}$
- Cache kann k Kopien von Seiten aus dem Hauptspeicher enthalten ($k < N$).
- n sequentielle Seitenzugriffe eines Programms werden beschrieben durch die Sequenz

$$\sigma\colon \{1, \ldots, n\} \to \{1, \ldots, N\}$$

Ablauf:

1. Programm fragt die i-te Seite $p_{\sigma(i)}$ an.
2. Falls $p_{\sigma(i)}$ noch nicht im Cache enthalten ist (**Fehlzugriff**), dann wird $p_{\sigma(i)}$ in den Cache geladen.
3. Programm greift auf $p_{\sigma(i)}$ im Cache zu.

Wie Seiten im Cache ersetzen, damit möglichst wenige Fehlzugriffe auftreten?

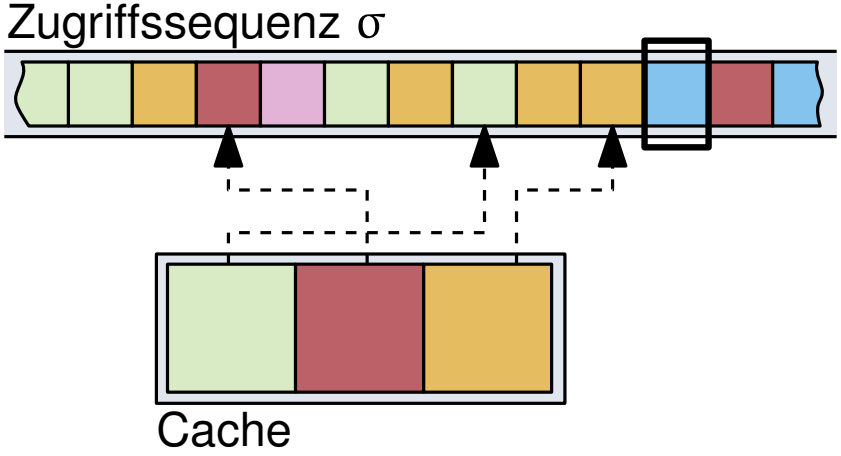
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Strategien

Name	Ersetze Seite im Cache,...
FIFO: First In/First Out	die bereits am längsten im Cache ist.
LIFO: Last In/First Out	die am neusten im Cache ist.
LFU: Least Frequently Used	die bisher am wenigsten häufig angefordert wurde.
LRU: Least Recently Used	deren Anfrage am weitesten in der Vergangenheit liegt.
FWF: Flush When Full	(Gebe alle Seiten frei, wenn Cache voll ist.)
LFD: Longest Forward Distance	deren Anfrage am weitesten in der Zukunft liegt.

Alle bis auf LFD sind Online-Algorithmen. LFD muss die komplette Anfragesequenz kennen und ist somit ein Offline-Algorithmus.

Beispiel LRU:



1. Speichere für jede Seite im Cache, wann sie zuletzt verwendet wurde.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Vergangenheit liegt.

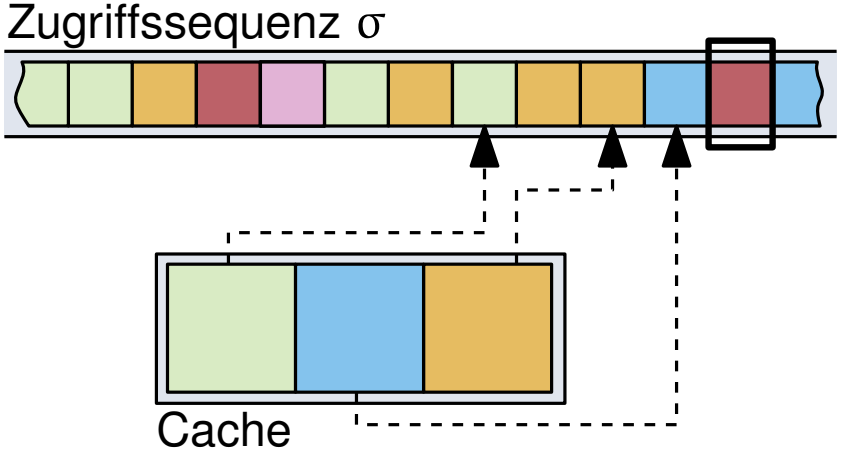
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Strategien

Name	Ersetze Seite im Cache,...
FIFO: First In/First Out	die bereits am längsten im Cache ist.
LIFO: Last In/First Out	die am neusten im Cache ist.
LFU: Least Frequently Used	die bisher am wenigsten häufig angefordert wurde.
LRU: Least Recently Used	deren Anfrage am weitesten in der Vergangenheit liegt.
FWF: Flush When Full	(Gebe alle Seiten frei, wenn Cache voll ist.)
LFD: Longest Forward Distance	deren Anfrage am weitesten in der Zukunft liegt.

Alle bis auf LFD sind Online-Algorithmen. LFD muss die komplette Anfragesequenz kennen und ist somit ein Offline-Algorithmus.

Beispiel LRU:



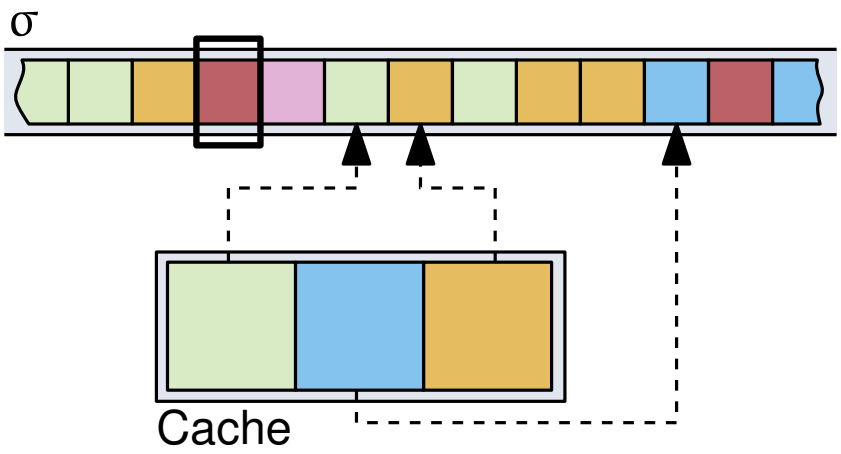
1. Speichere für jede Seite im Cache, wann sie zuletzt verwendet wurde.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Vergangenheit liegt.

Strategien

Name	Ersetze Seite im Cache,...
FIFO: First In/First Out	die bereits am längsten im Cache ist.
LIFO: Last In/First Out	die am neusten im Cache ist.
LFU: Least Frequently Used	die bisher am wenigsten häufig angefordert wurde.
LRU: Least Recently Used	deren Anfrage am weitesten in der Vergangenheit liegt.
FWF: Flush When Full	(Gebe alle Seiten frei, wenn Cache voll ist.)
LFD: Longest Forward Distance	deren Anfrage am weitesten in der Zukunft liegt.

Alle bis auf LFD sind Online-Algorithmen. LFD muss die komplette Anfragesequenz kennen und ist somit ein Offline-Algorithmus.

Beispiel LFD:



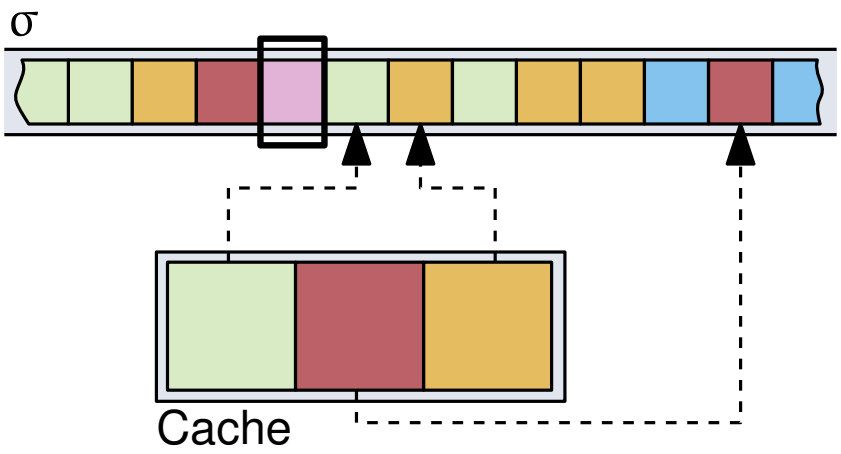
1. Speichere für jede Seite im Cache, wann sie als nächstes verwendet wird.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Zukunft liegt.

Strategien

Name	Ersetze Seite im Cache,...
FIFO: First In/First Out	die bereits am längsten im Cache ist.
LIFO: Last In/First Out	die am neusten im Cache ist.
LFU: Least Frequently Used	die bisher am wenigsten häufig angefordert wurde.
LRU: Least Recently Used	deren Anfrage am weitesten in der Vergangenheit liegt.
FWF: Flush When Full	(Gebe alle Seiten frei, wenn Cache voll ist.)
LFD: Longest Forward Distance	deren Anfrage am weitesten in der Zukunft liegt.

Alle bis auf LFD sind Online-Algorithmen. LFD muss die komplette Anfragesequenz kennen und ist somit ein Offline-Algorithmus.

Beispiel LFD:



1. Speichere für jede Seite im Cache, wann sie als nächstes verwendet wird.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Zukunft liegt.

Konservative Paging-Algorithmen

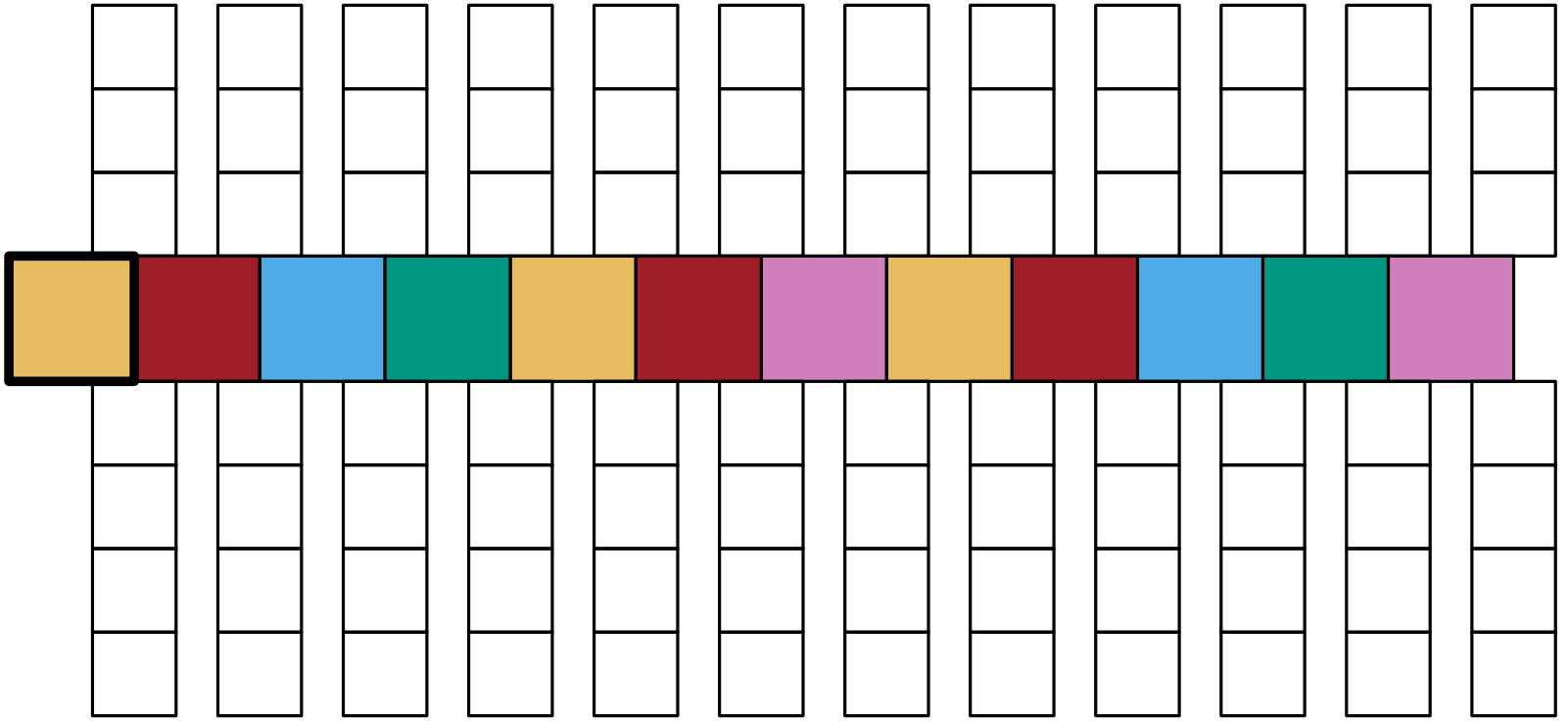
(h,k)-Paging-Problem:

- Optimaler Offline-Algorithmus für Paging arbeitet auf Cache der Größe h.
- Online-Algorithmus für Paging arbeitet auf Cache der Größe k mit $k \geq h$.
- Online-Algorithmus bekommt größeren Cache um dessen Unwissenheit auszugleichen.

Béládys Anomalie: Für manche der Paging-Algorithmen kann man Zugriffssequenzen finden, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. FIFO ist ein solcher Algorithmus (siehe Übung).

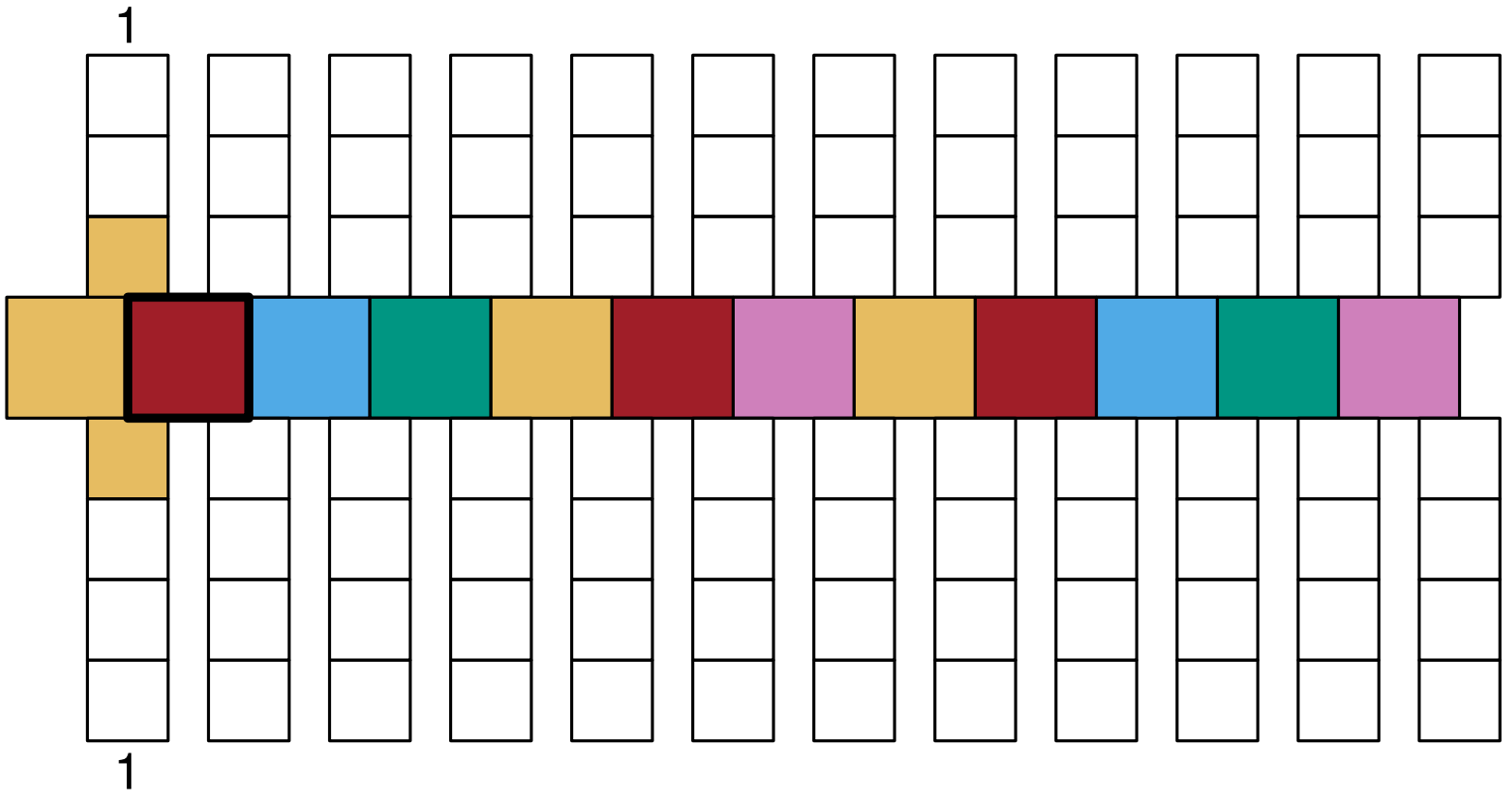
Problem 3

Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



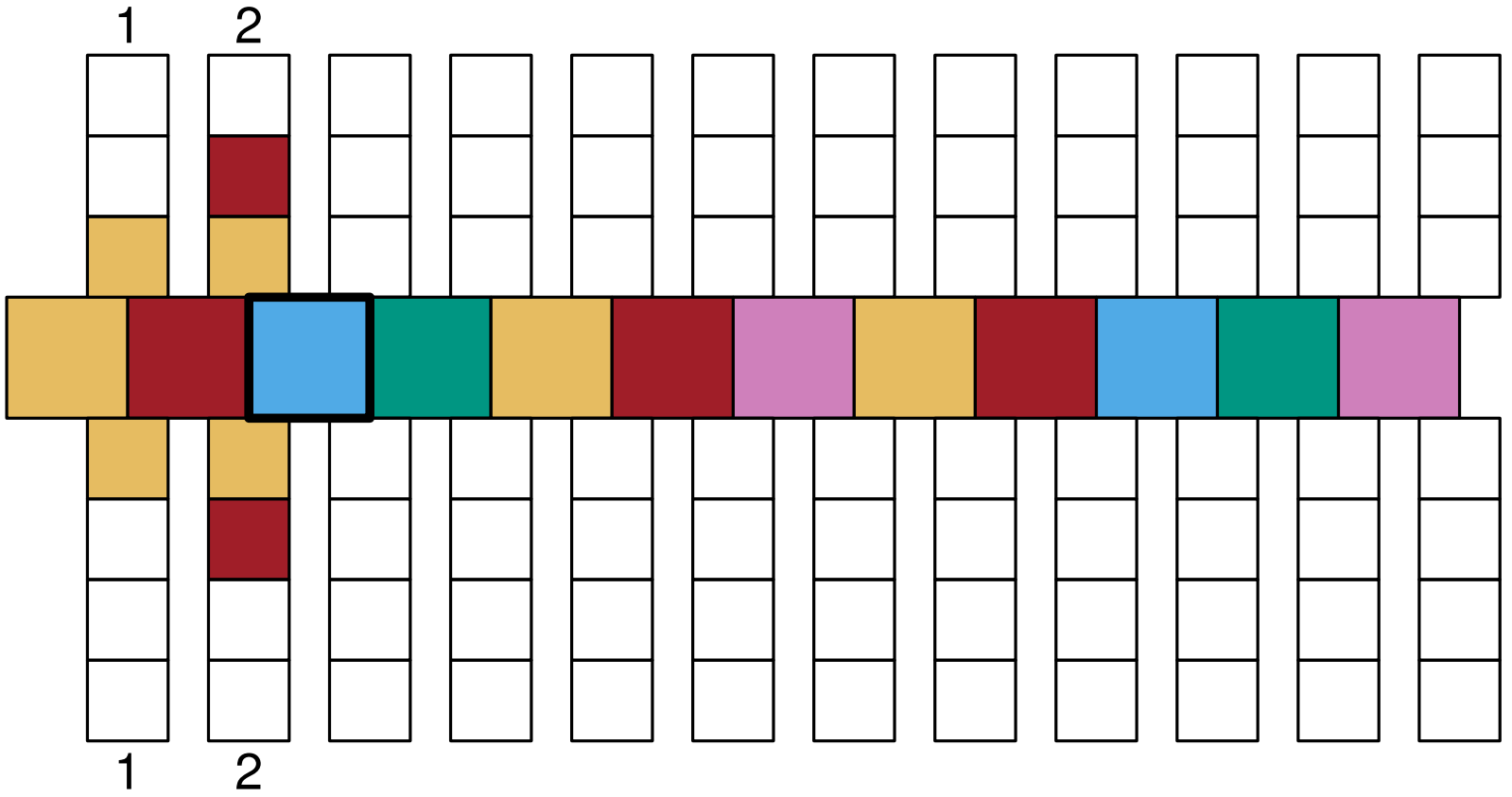
Problem 3

Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



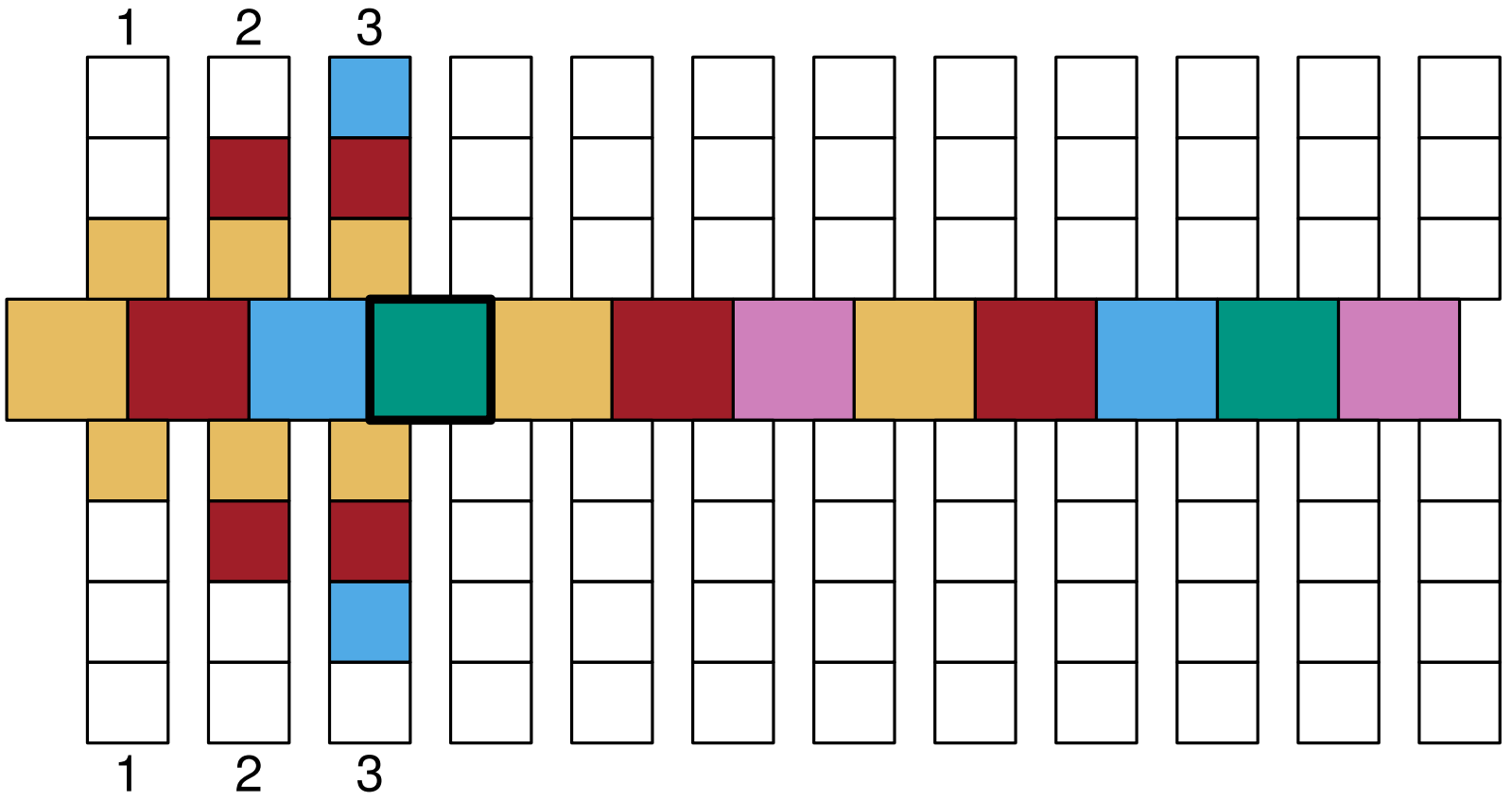
Problem 3

Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



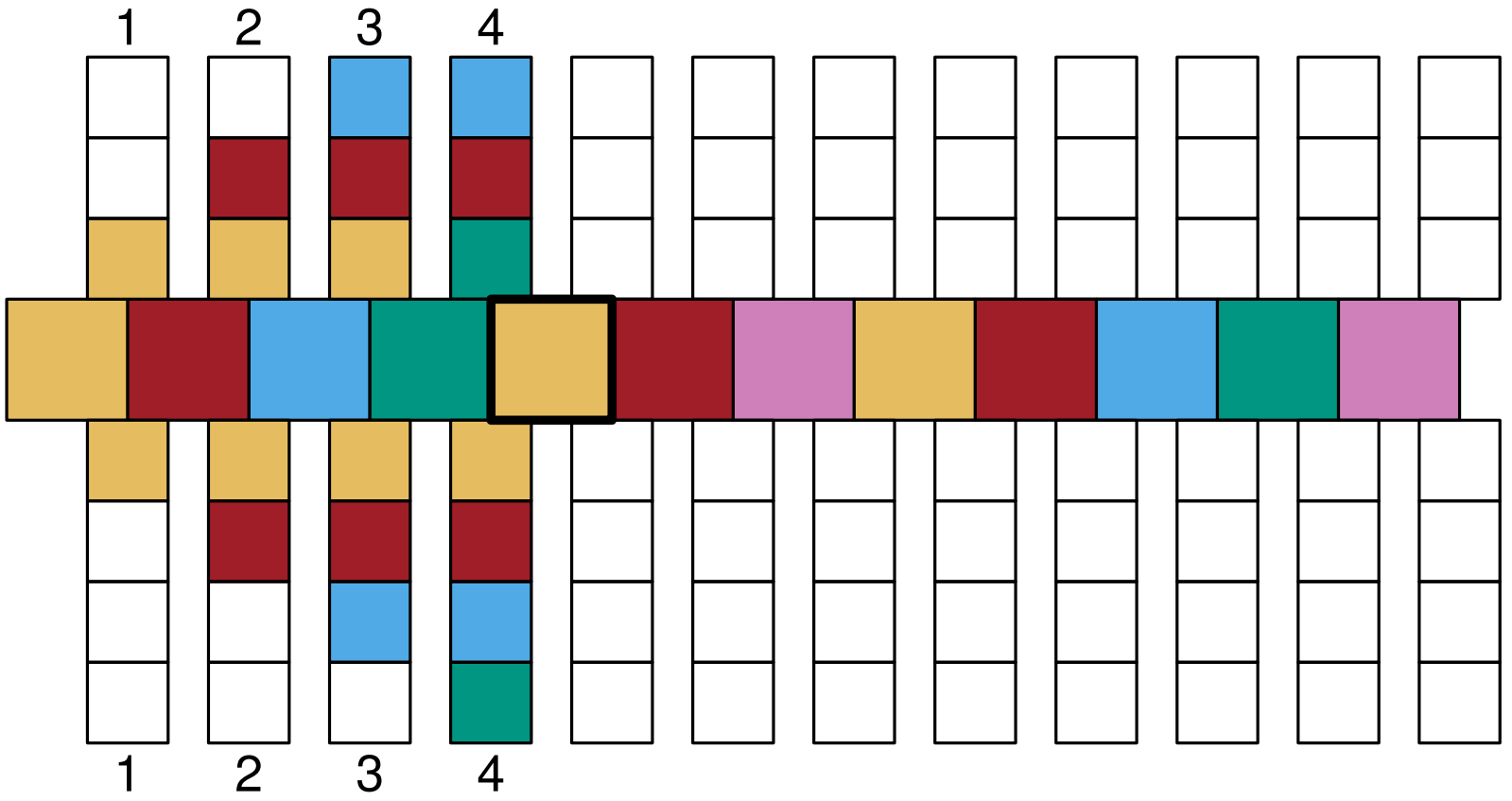
Problem 3

Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Problem 3

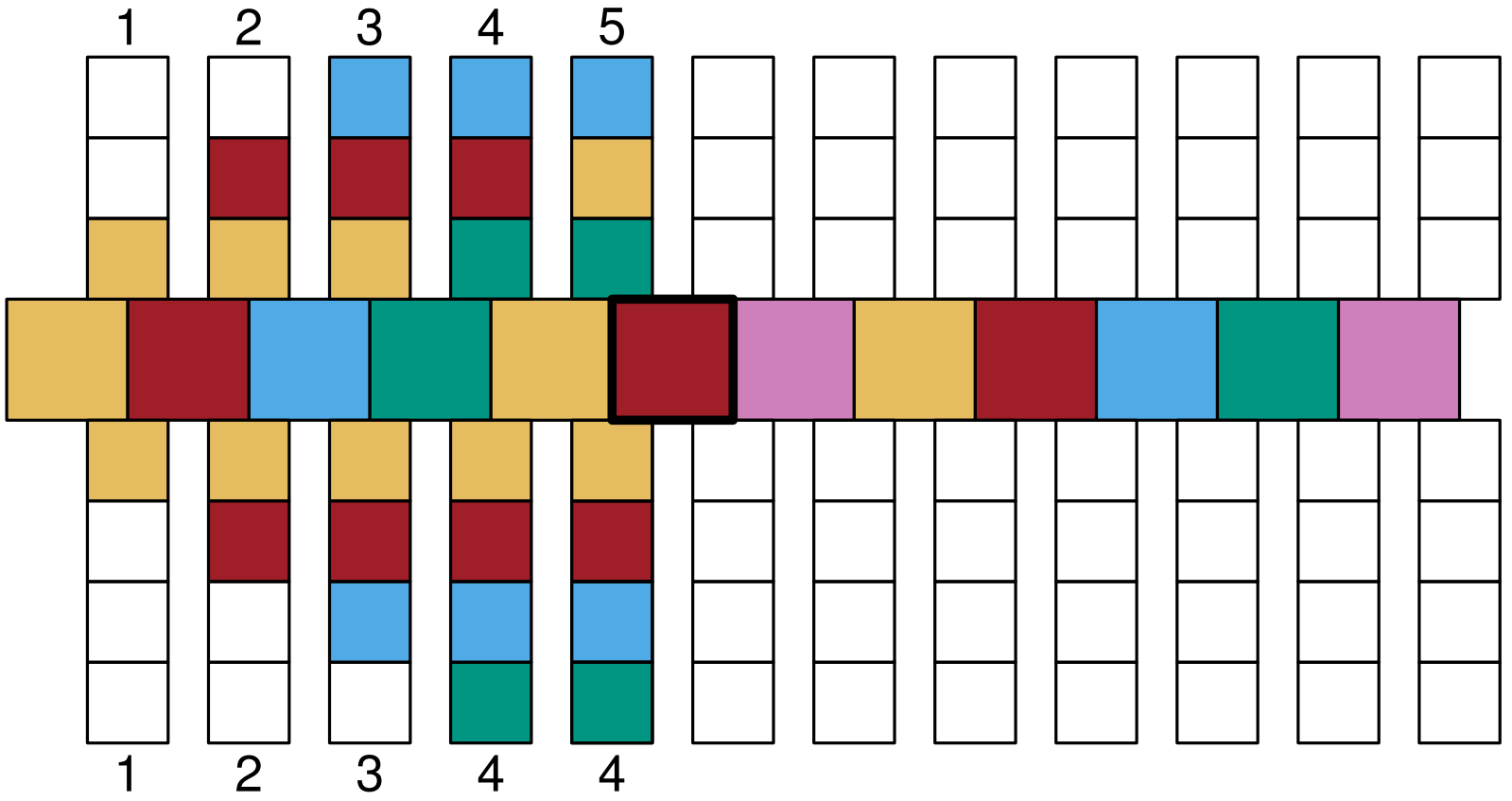
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

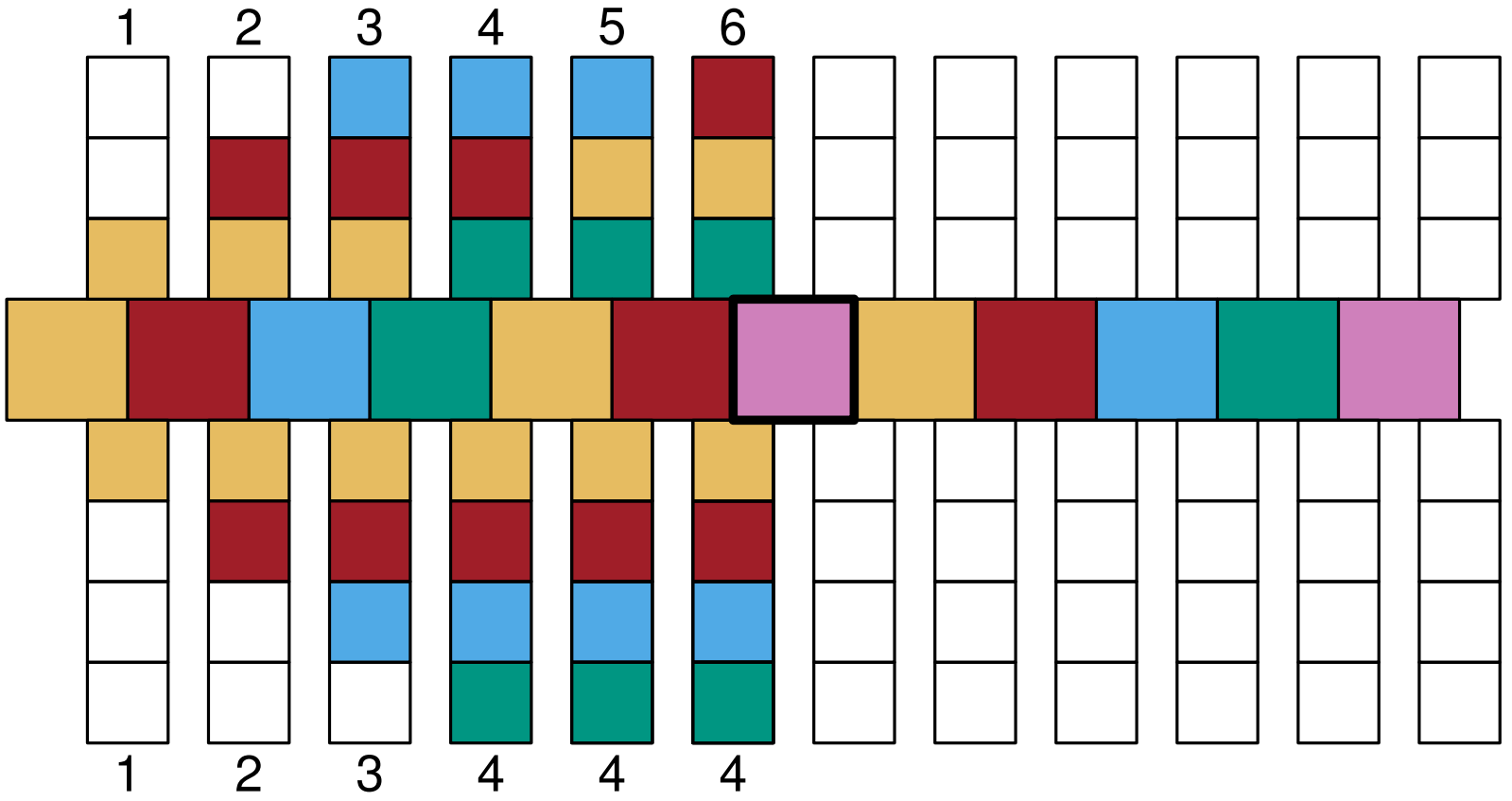
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

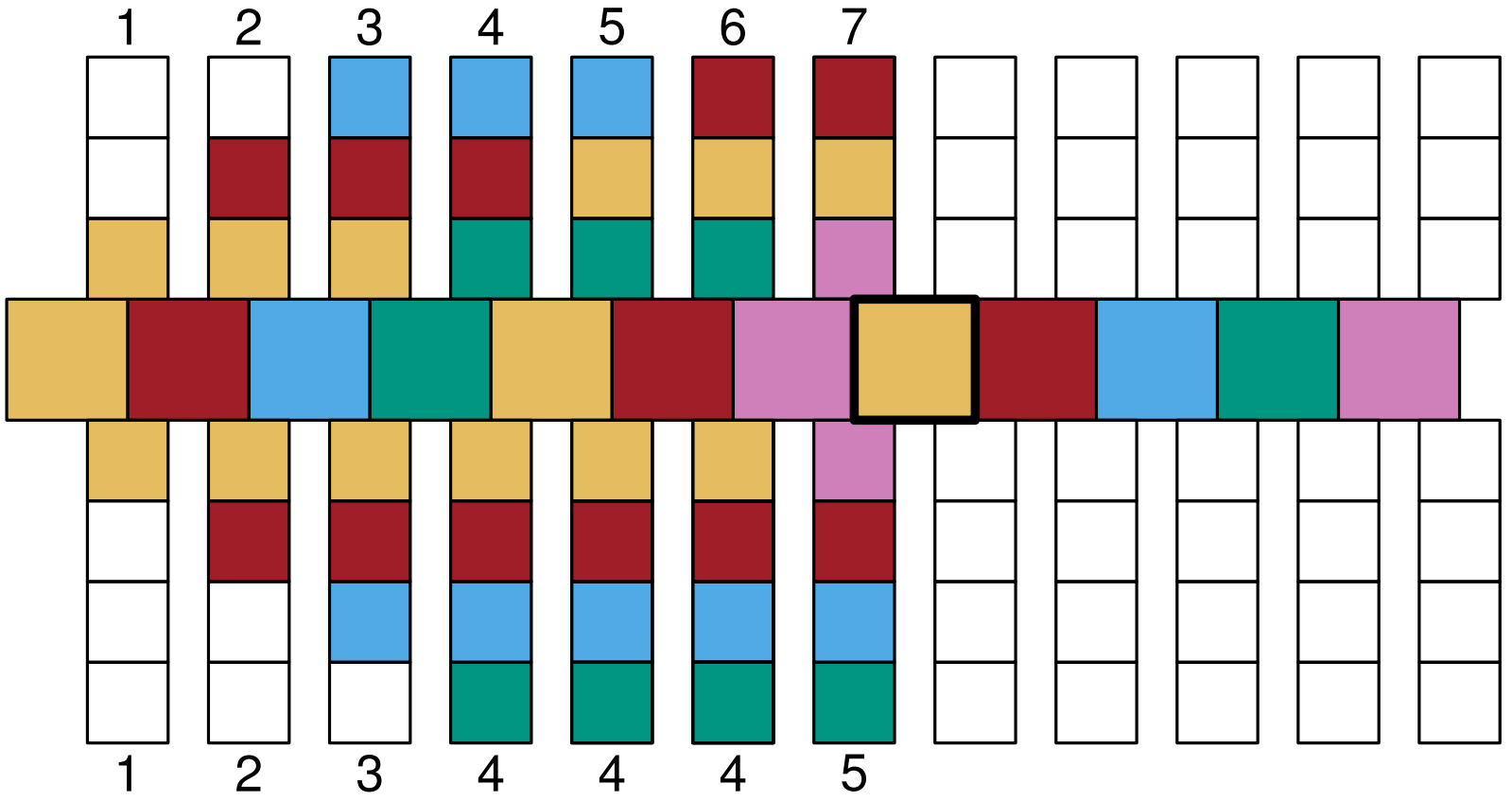
Problem 3

Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Problem 3

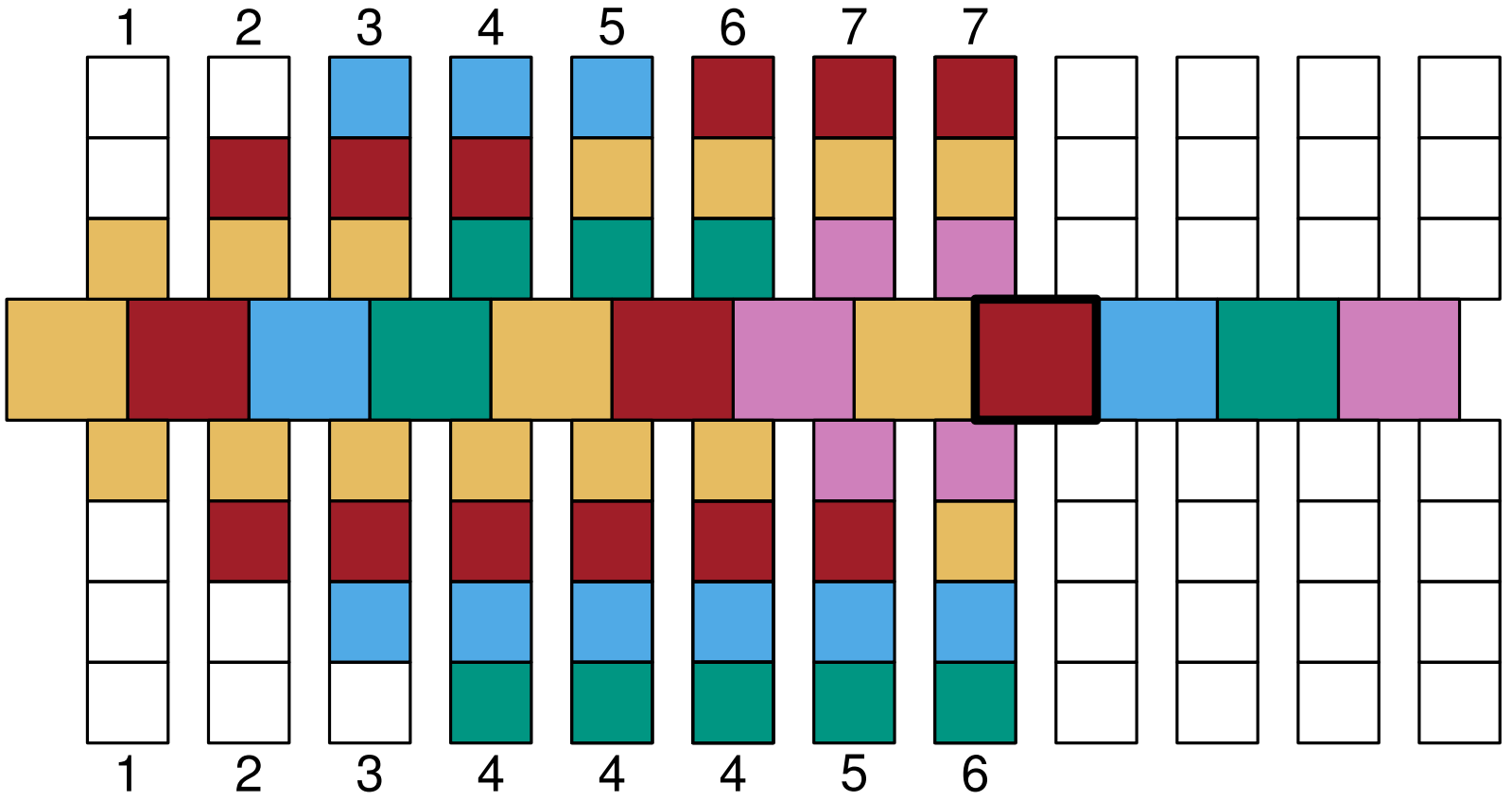
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

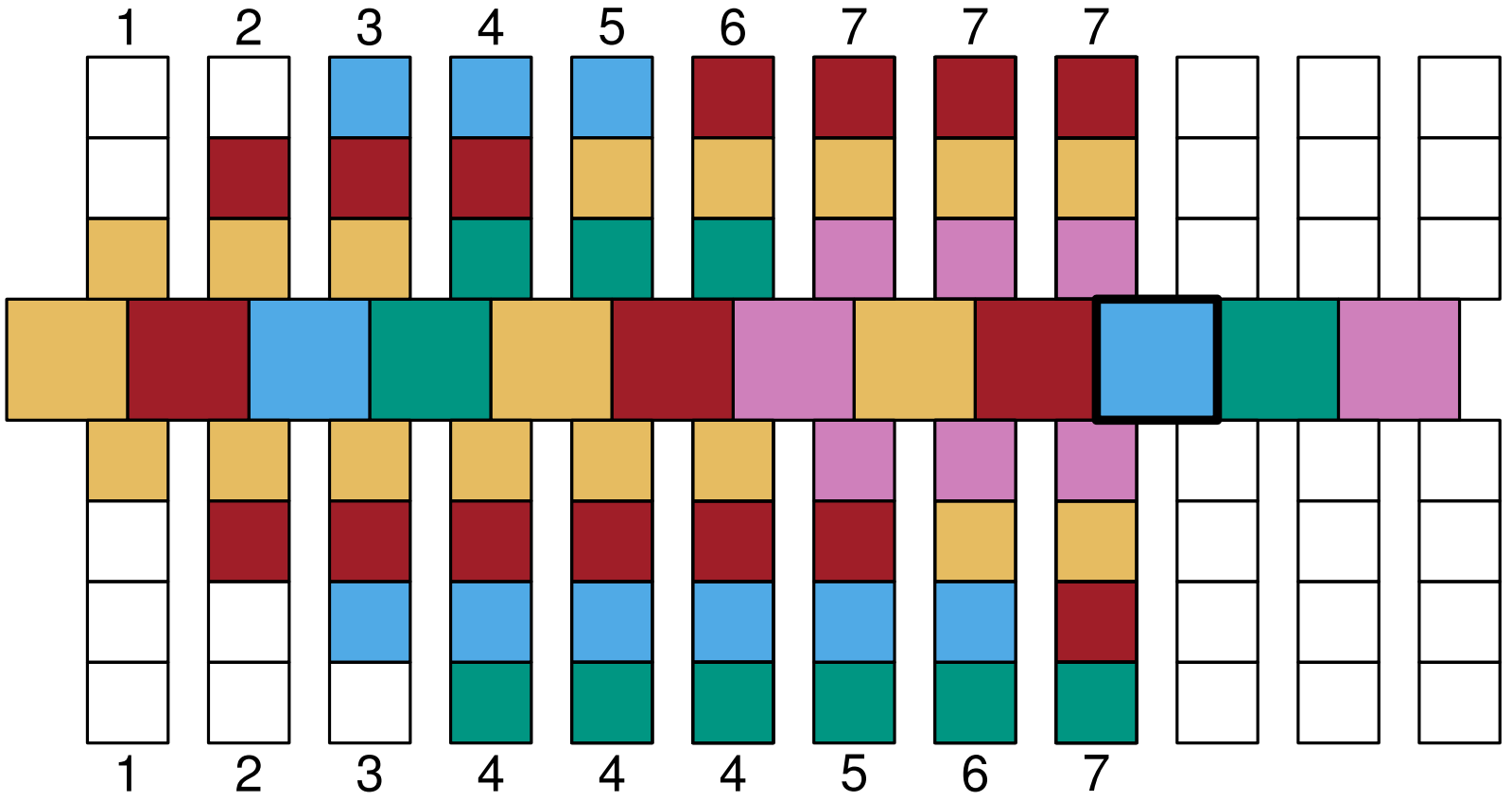
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

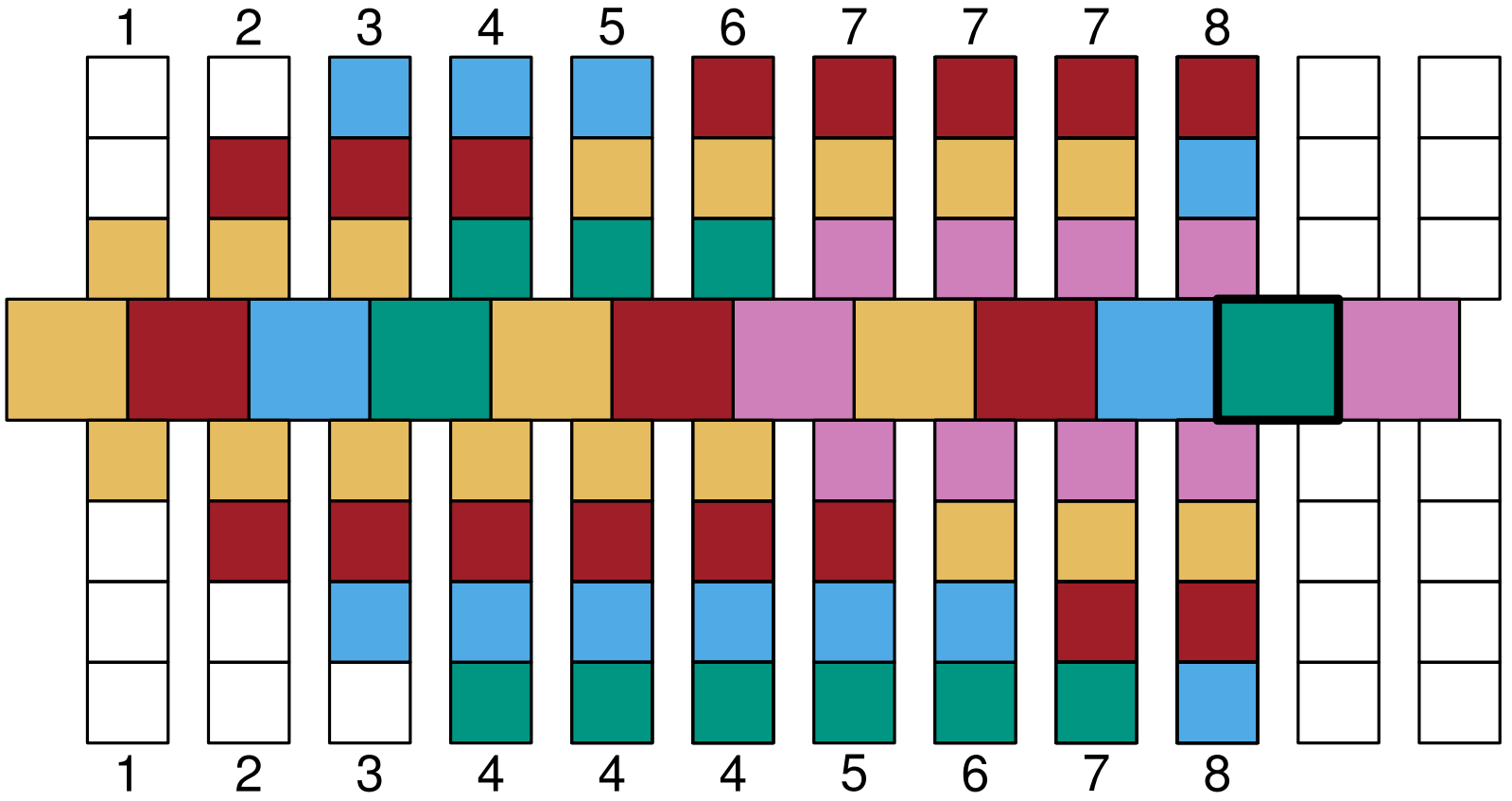
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

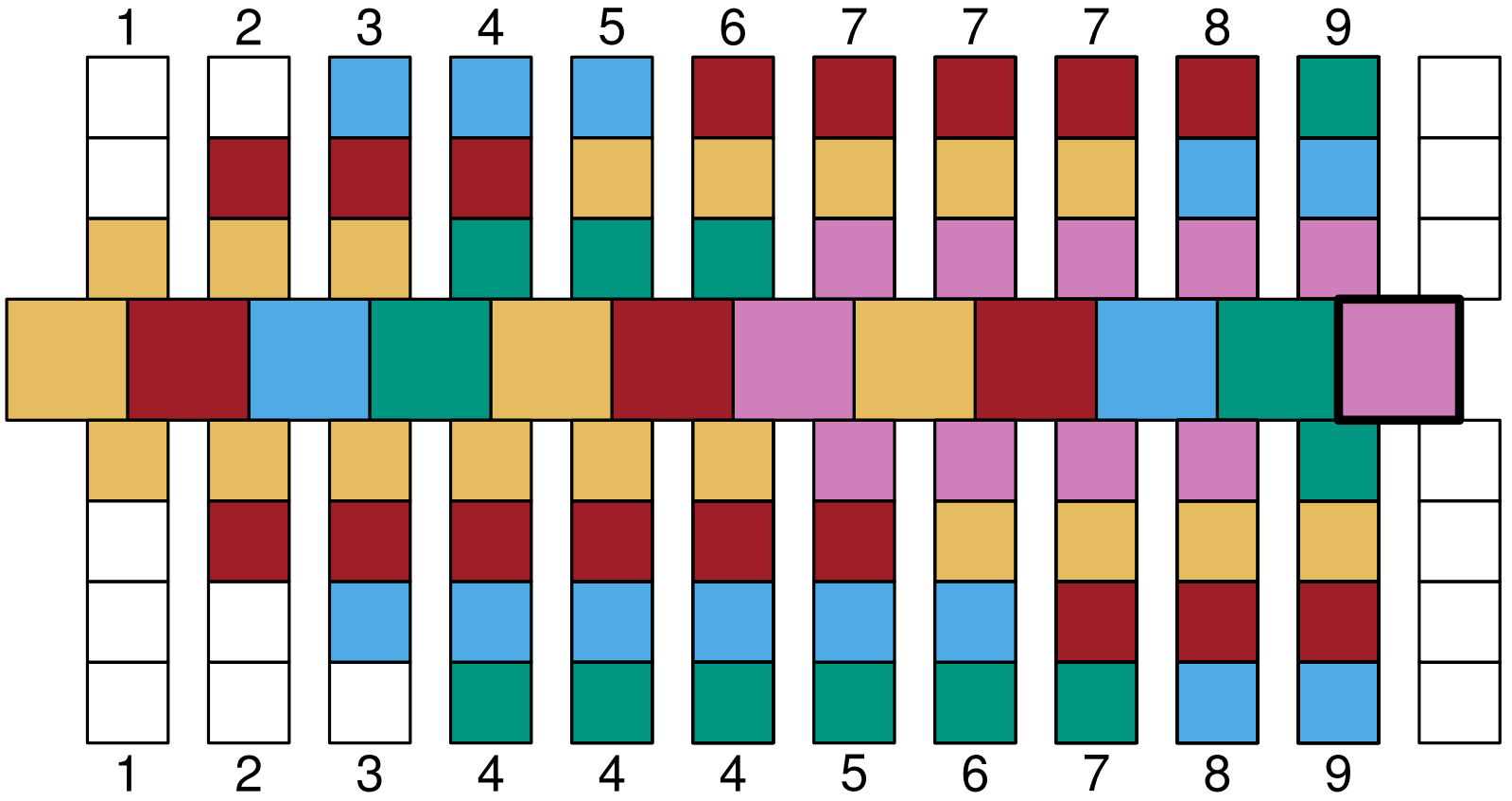
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

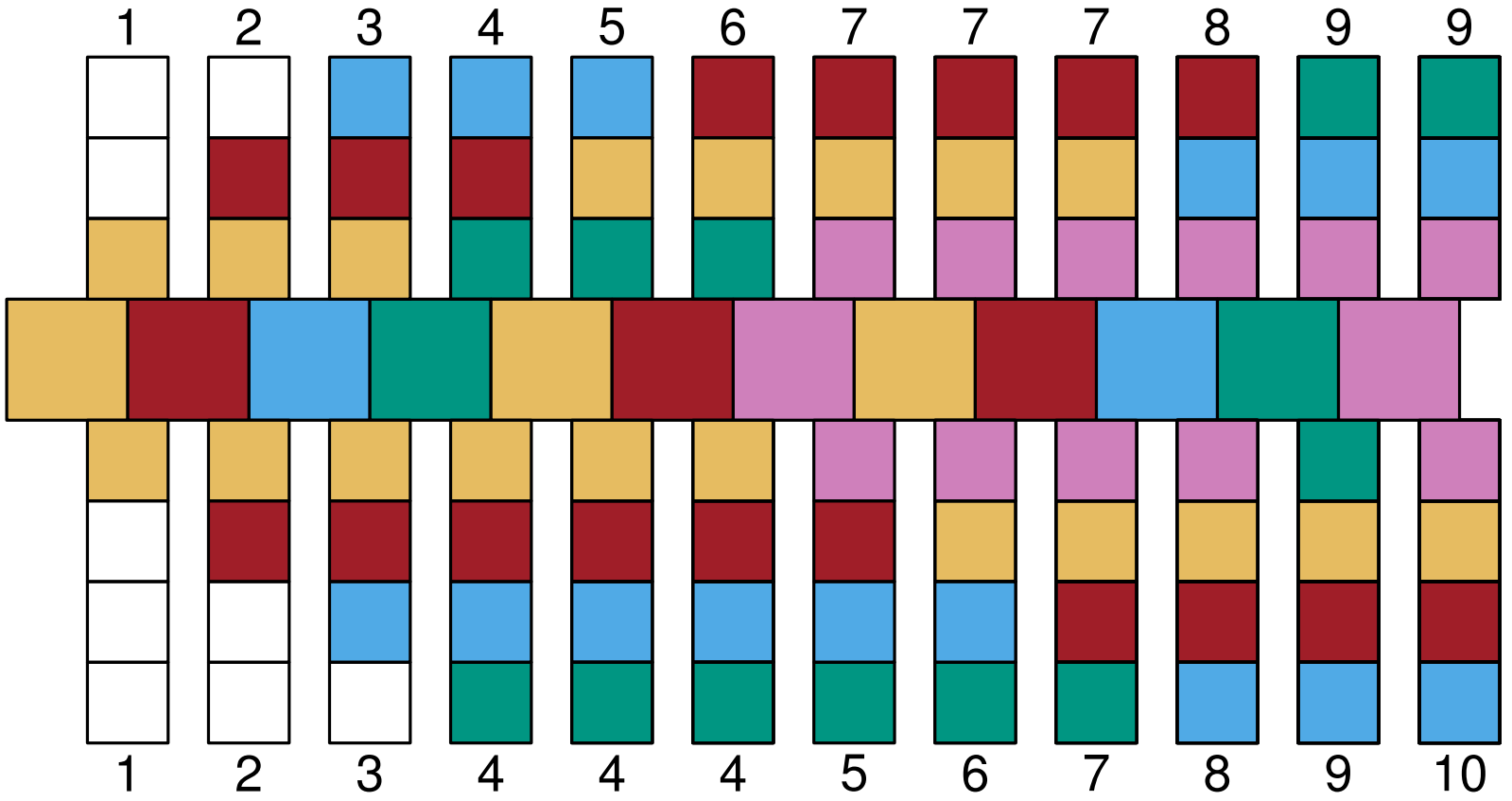
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Konservative Paging-Algorithmen

(h,k)-Paging-Problem:

- Optimaler Offline-Algorithmus für Paging arbeitet auf Cache der Größe h.
- Online-Algorithmus für Paging arbeitet auf Cache der Größe k mit $k \geq h$.
- Online-Algorithmus bekommt größeren Cache um dessen Unwissenheit auszugleichen.

Béládys Anomalie: Für manche der Paging-Algorithmen kann man Zugriffssequenzen finden, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. FIFO ist ein solcher Algorithmus (siehe Übung).

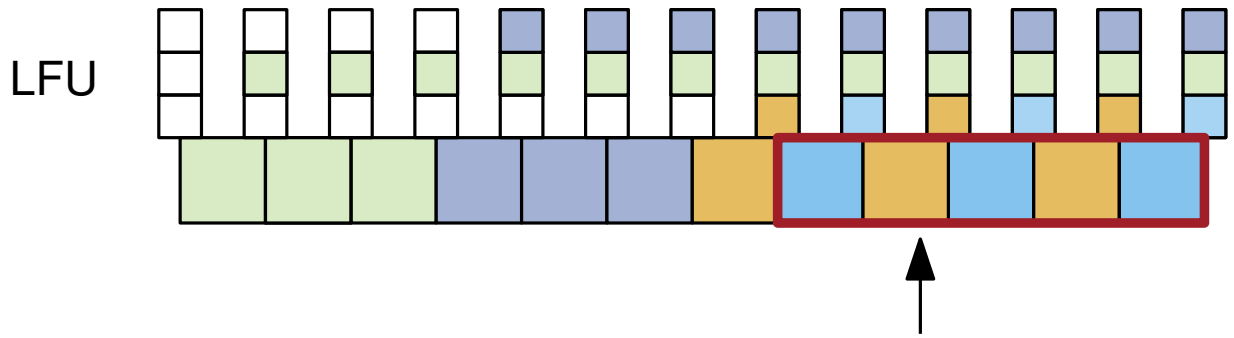
Konservative Paging-Algorithmen

(h,k)-Paging-Problem:

- Optimaler Offline-Algorithmus für Paging arbeitet auf Cache der Größe h.
- Online-Algorithmus für Paging arbeitet auf Cache der Größe k mit $k \geq h$.
- Online-Algorithmus bekommt größeren Cache um dessen Unwissenheit auszugleichen.

Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

Beispiel: LFU ist nicht konservativ.



Sequenz enthält 2 verschiedene Seiten, aber erzeugt 5 Fehlzugriffe.

Konservative Paging-Algorithmen

(h,k)-Paging-Problem:

- Optimaler Offline-Algorithmus für Paging arbeitet auf Cache der Größe h.
- Online-Algorithmus für Paging arbeitet auf Cache der Größe k mit $k \geq h$.
- Online-Algorithmus bekommt größeren Cache um dessen Unwissenheit auszugleichen.

Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

Name	Ersetze Seite im Cache,...
FIFO: First In/First Out	die bereits am längsten im Cache ist.
LIFO: Last In/First Out	die am neusten im Cache ist.
LFU: Least Frequently Used	die bisher am wenigsten häufig angefordert wurde.
LRU: Least Recently Used	deren Anfrage am weitesten in der Vergangenheit liegt.
FWF: Flush When Full	(Gebe alle Seiten frei, wenn Cache voll ist.)

konservative Paging-Algorithmen

Problem 4

Zeigen Sie, dass FWF, LIFO und LFU keine konservativen Paging-Algorithmen sind.

Lösung:

- LIFO und LFU sind nicht konservativ, weil sie noch nicht einmal kompetitiv sind.
- Betrachte für FWF folgendes Beispiel:

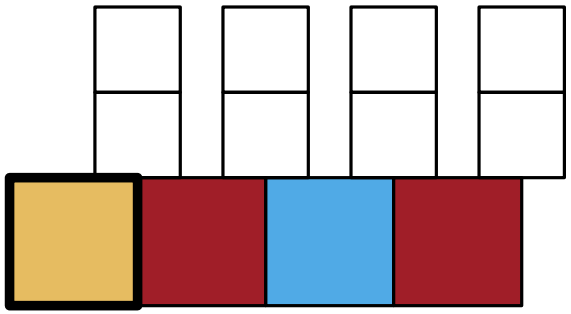
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

Problem 4

Zeigen Sie, dass FWF, LIFO und LFU keine konservativen Paging-Algorithmen sind.

Lösung:

- LIFO und LFU sind nicht konservativ, weil sie noch nicht einmal kompetitiv sind.
- Betrachte für FWF folgendes Beispiel:



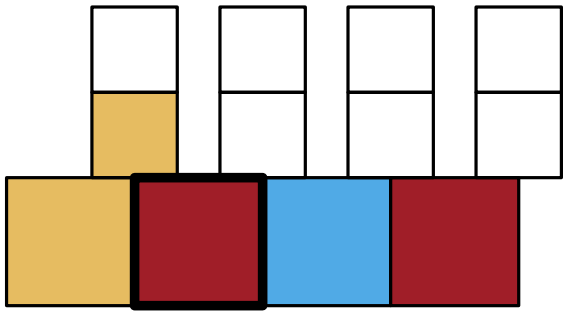
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

Problem 4

Zeigen Sie, dass FWF, LIFO und LFU keine konservativen Paging-Algorithmen sind.

Lösung:

- LIFO und LFU sind nicht konservativ, weil sie noch nicht einmal kompetitiv sind.
- Betrachte für FWF folgendes Beispiel:



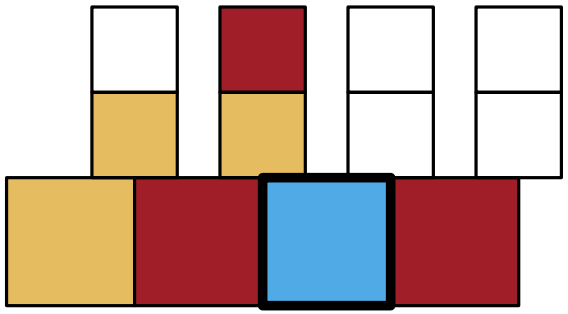
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

Problem 4

Zeigen Sie, dass FWF, LIFO und LFU keine konservativen Paging-Algorithmen sind.

Lösung:

- LIFO und LFU sind nicht konservativ, weil sie noch nicht einmal kompetitiv sind.
- Betrachte für FWF folgendes Beispiel:



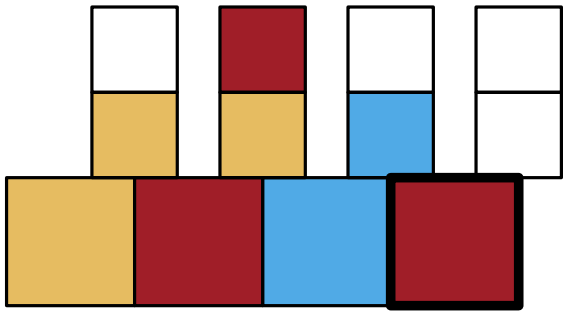
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

Problem 4

Zeigen Sie, dass FWF, LIFO und LFU keine konservativen Paging-Algorithmen sind.

Lösung:

- LIFO und LFU sind nicht konservativ, weil sie noch nicht einmal kompetitiv sind.
- Betrachte für FWF folgendes Beispiel:



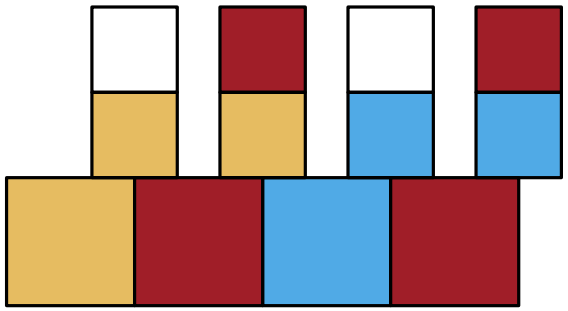
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

Problem 4

Zeigen Sie, dass FWF, LIFO und LFU keine konservativen Paging-Algorithmen sind.

Lösung:

- LIFO und LFU sind nicht konservativ, weil sie noch nicht einmal kompetitiv sind.
- Betrachte für FWF folgendes Beispiel:



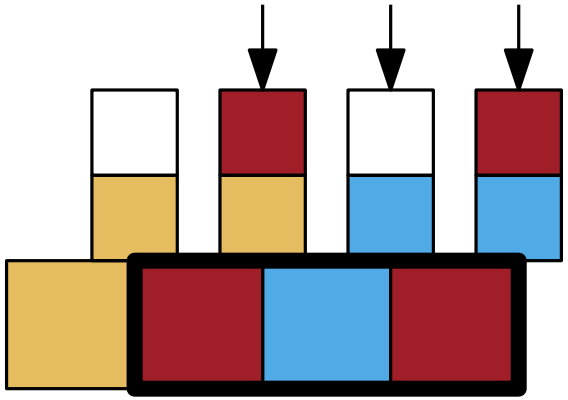
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

Problem 4

Zeigen Sie, dass FWF, LIFO und LFU keine konservativen Paging-Algorithmen sind.

Lösung:

- LIFO und LFU sind nicht konservativ, weil sie noch nicht einmal kompetitiv sind.
- Betrachte für FWF folgendes Beispiel:



Teilsequenz enthält zwei verschiedene Seiten, aber es gibt drei Fehlzugriffe.

Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

Organisatorisches

Fragestunde und Wiederholung am 13.02.14:

Fragen können auch bereits im Voraus an uns geschickt werden:
thomas.blaesius@kit.edu, benjamin.niedermann@kit.edu

Für die Klausur können Wörterbücher (Deutsch $\leftrightarrow$ Englisch) ausgeliehen werden.

↳ Bei Interesse bis zum 13.02.14 eine E-Mail schreiben.