Karlsruher Institut für Technologie

Algorithmen II
Übung am 05.02.2013

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

Organisatorisches

Fragestunde und Wiederholung am 07.02.13:

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

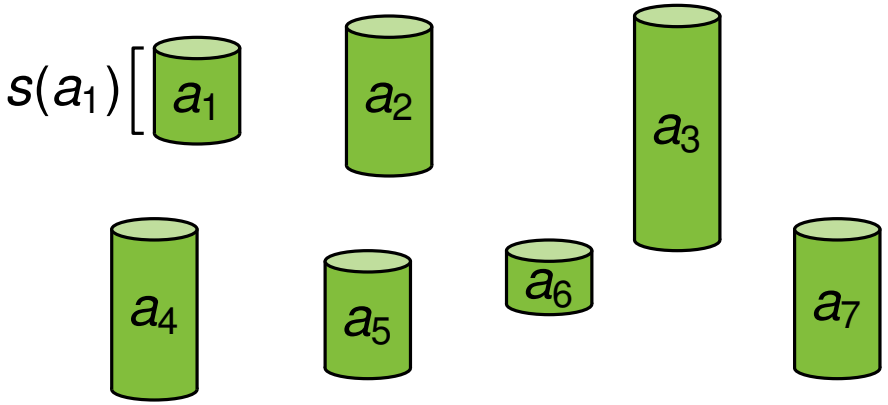
Karlsruher Institut für Technologie

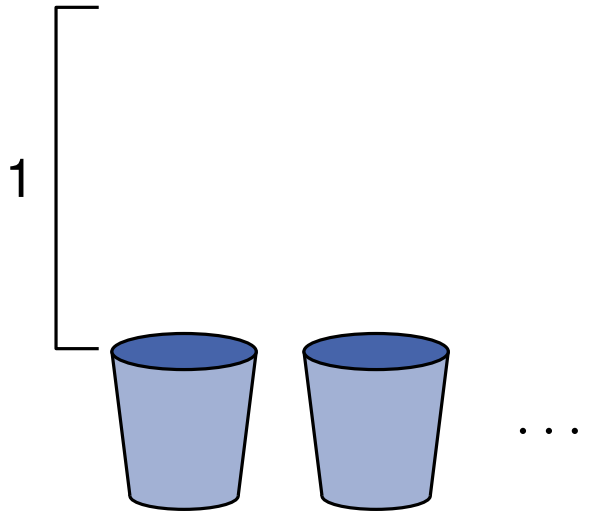
Online-Algorithmen

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:

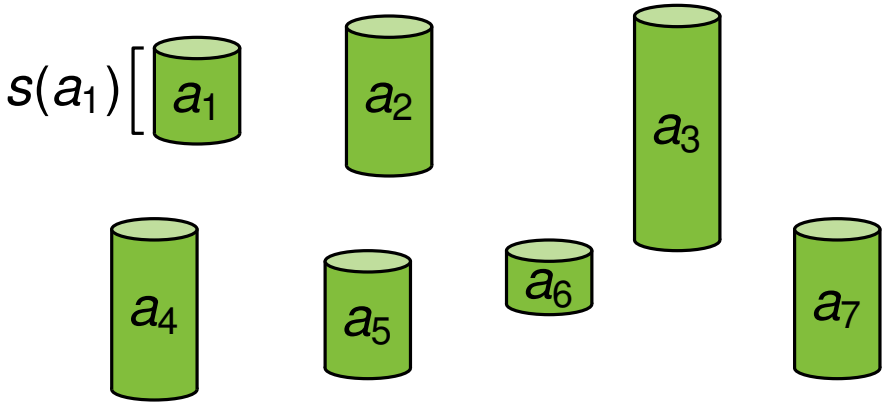
$$\sum_{a_i \in B} s(a_i) \leq 1$$

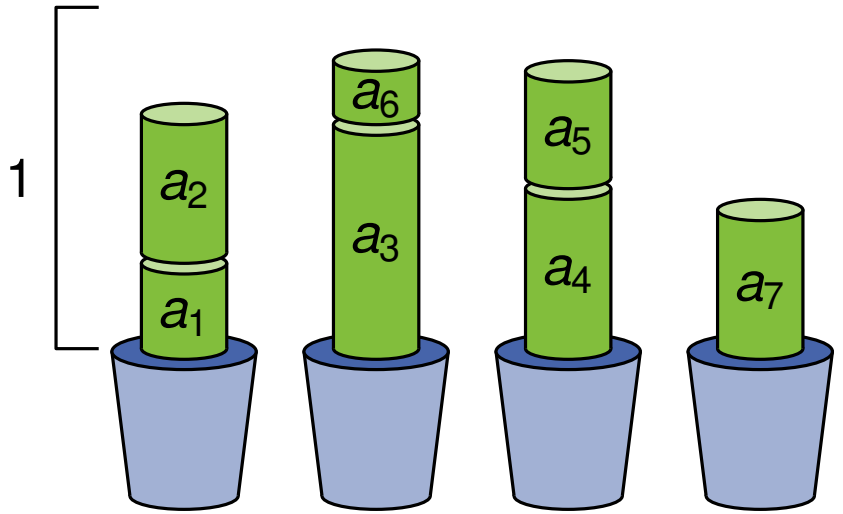
BIN PACKING ist $\mathcal{NP}$-schwer.

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:

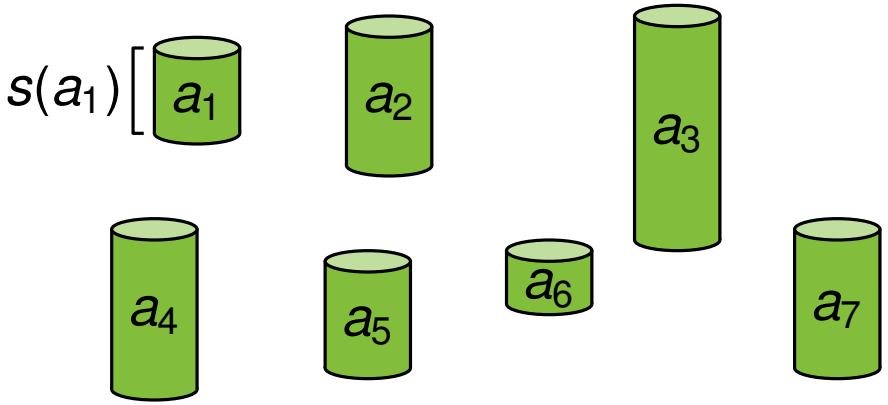
$$\sum_{a_i \in B} s(a_i) \leq 1$$

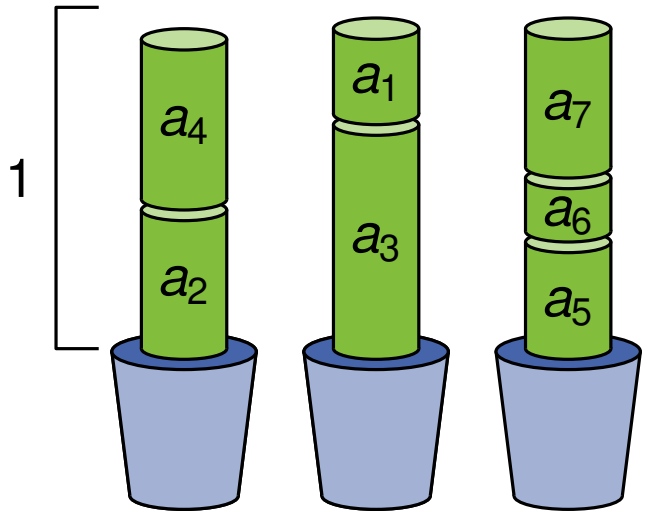
BIN PACKING ist $\mathcal{NP}$-schwer.

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

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 $a_\ell \in \{a_2, \ldots, a_n\}$ **do**

  $B_j \leftarrow$ letzter nicht-leerer Bin

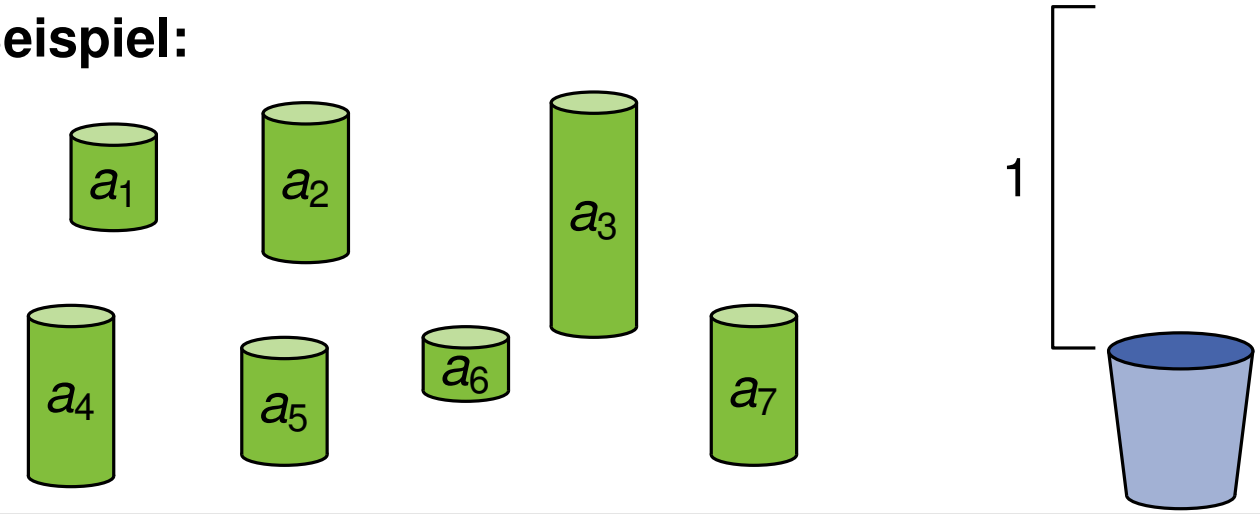
  if $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$ **then**

    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 $a_\ell \in \{a_2, \ldots, a_n\}$ **do**

  $B_j \leftarrow$ letzter nicht-leerer Bin

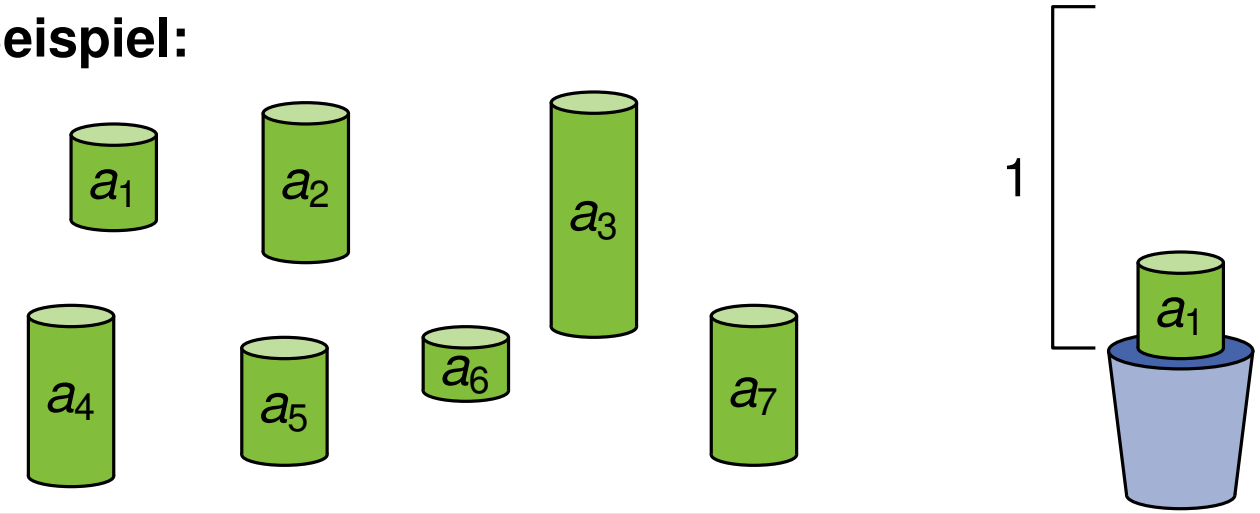
  if $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$ **then**

    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 $a_\ell \in \{a_2, \ldots, a_n\}$ **do**

  $B_j \leftarrow$ letzter nicht-leerer Bin

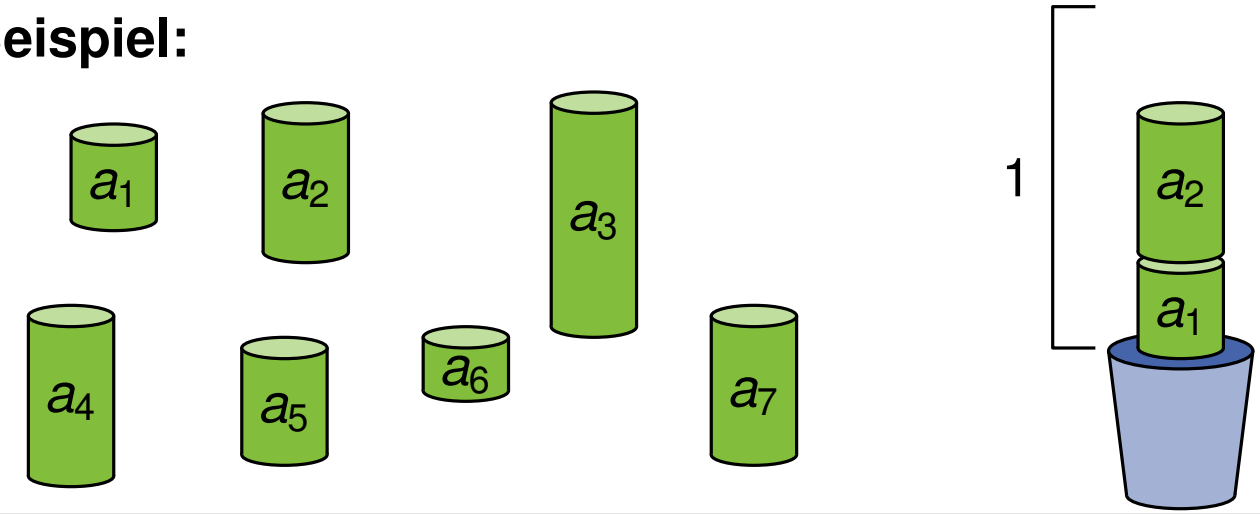
  if $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$ **then**

    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

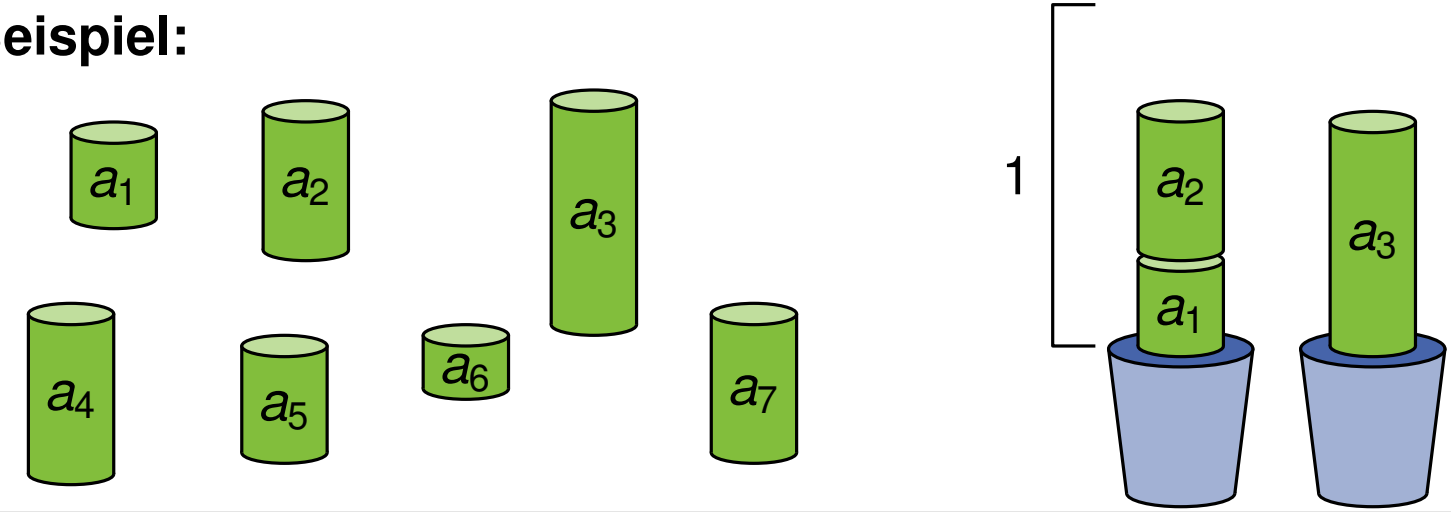
Strategie:

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

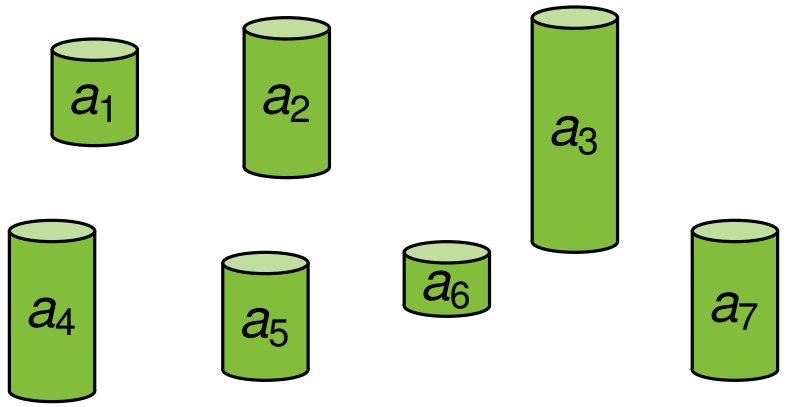
Strategie:

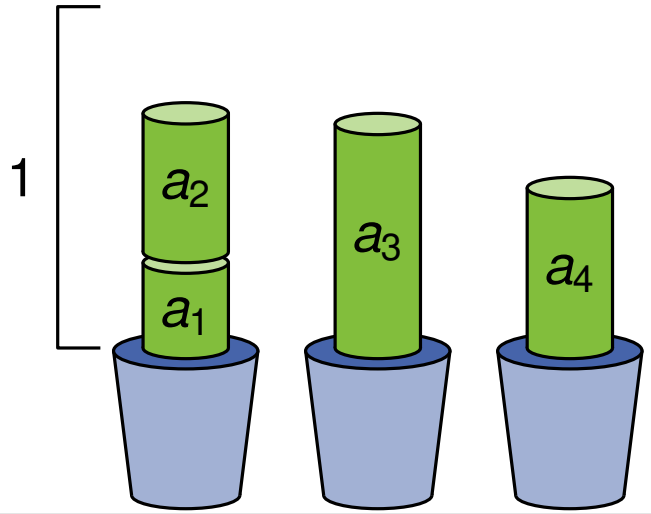
Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

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

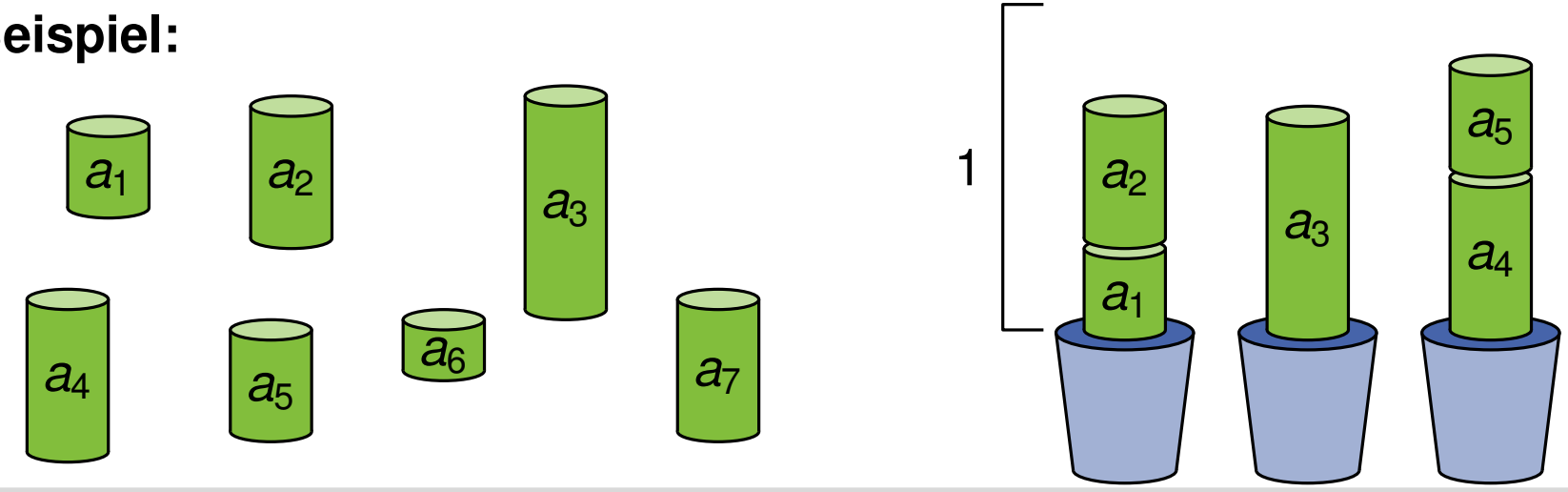
Strategie:

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 $a_\ell \in \{a_2, \ldots, a_n\}$ **do**

  $B_j \leftarrow$ letzter nicht-leerer Bin

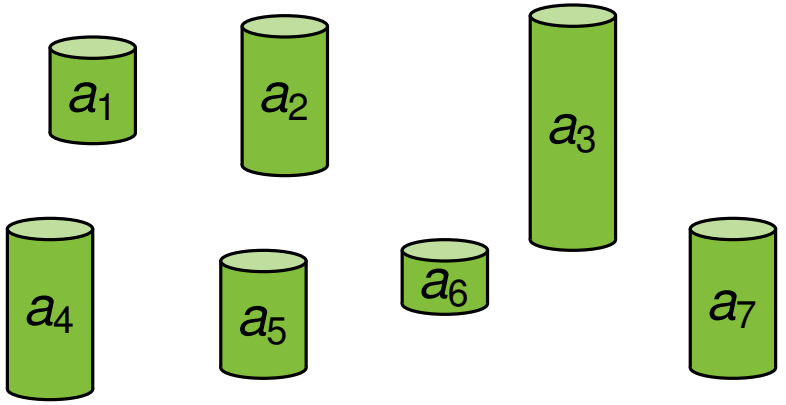
  if $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$ **then**

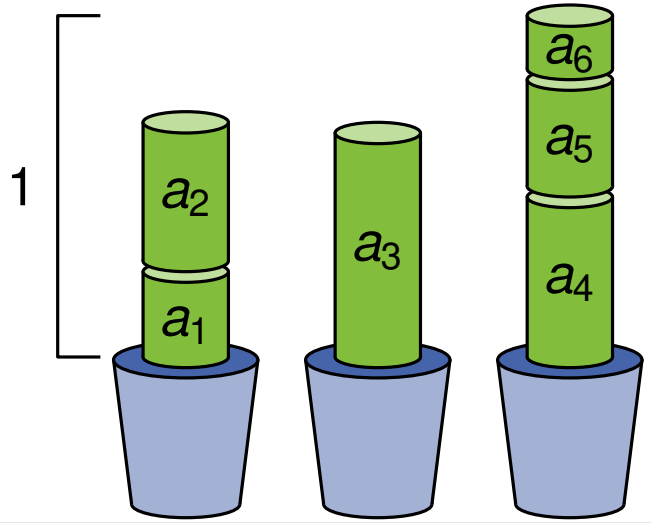
    Füge a_ℓ in B_j ein

  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 $a_\ell \in \{a_2, \ldots, a_n\}$ **do**

  $B_j \leftarrow$ letzter nicht-leerer Bin

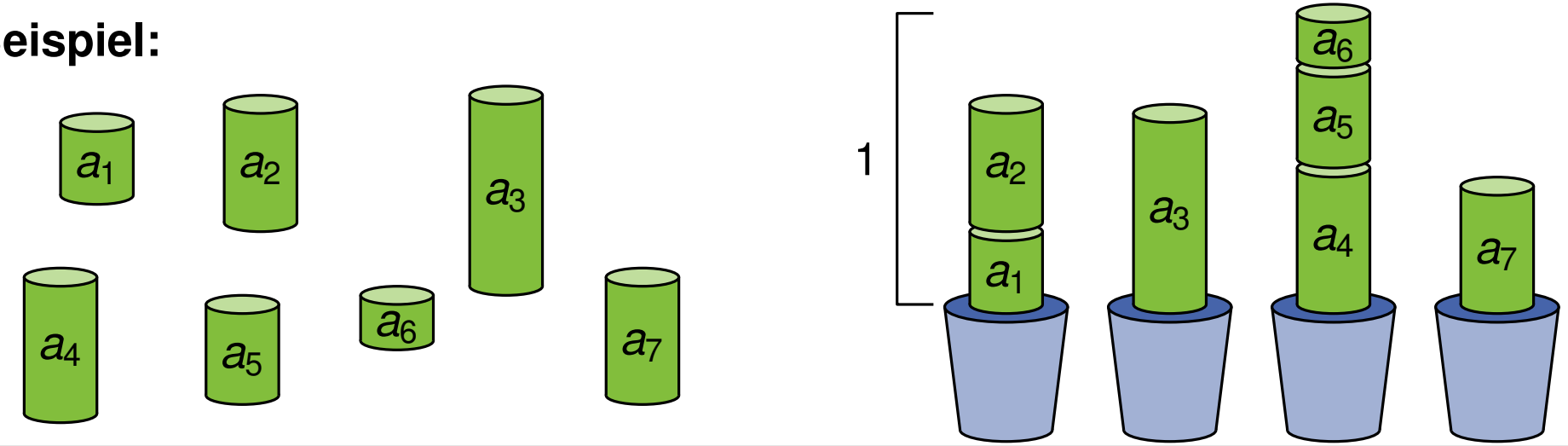
  if $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$ **then**

    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

Next Fit – Approximation

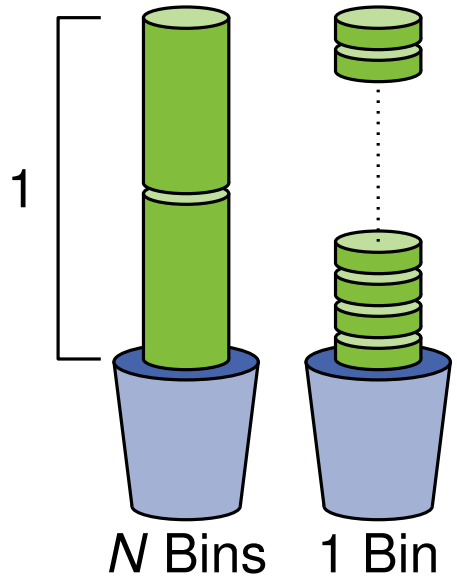
Schlechtes Beispiel

(Beispiel 7.4)

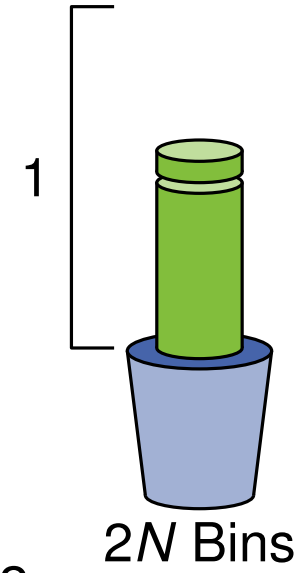
Betrachte die Menge $\{a_1, \ldots, a_n\}$ für $n = 4 \cdot N$ mit $N \in \mathbb{N}$. Sei weiterhin

$$s(a_i) = \begin{cases} \frac{1}{2} & \text{falls } i \text{ ungerade} \\ \frac{1}{2 \cdot N} & \text{sonst} \end{cases}$$

Optimale Lösung OPT(I)



Lösung von NEXT FIT NF(I)



$$\Rightarrow \text{NF}(I) = 2 \cdot \text{OPT}(I) - 2$$

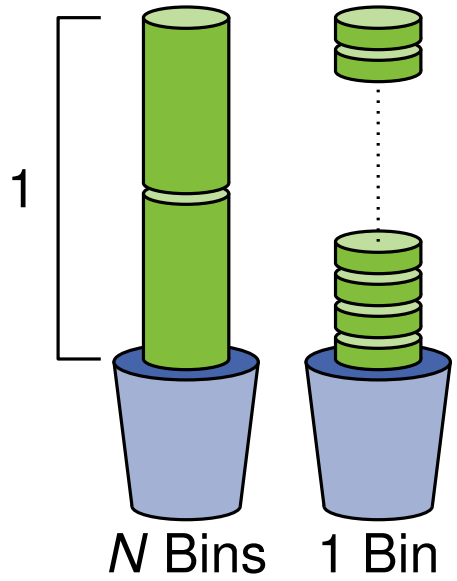
Next Fit – Approximation

Schlechtes Beispiel **(Beispiel 7.4)**

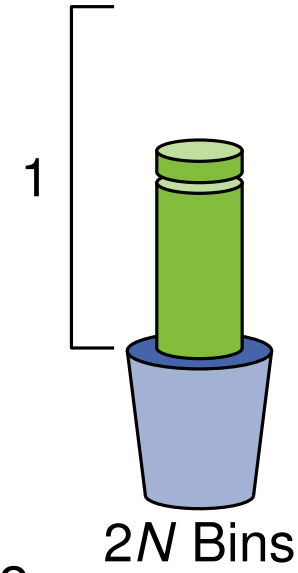
Betrachte die Menge $\{a_1, \ldots, a_n\}$ für $n = 4 \cdot N$ mit $N \in \mathbb{N}$. Sei weiterhin

$$s(a_i) = \begin{cases} \frac{1}{2} & \text{falls } i \text{ ungerade} \\ \frac{1}{2 \cdot N} & \text{sonst} \end{cases}$$

Optimale Lösung OPT(I)



Lösung von NEXT FIT NF(I)



$$\Rightarrow \text{NF}(I) = 2 \cdot \text{OPT}(I) - 2$$

Lösung Problem 1:

Satz: Approximation **(Satz 7.5)**

NEXT FIT erfüllt $\mathcal{R}_{\text{NF}} = 2$.

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: FIRST FIT, NEXT FIT

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

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

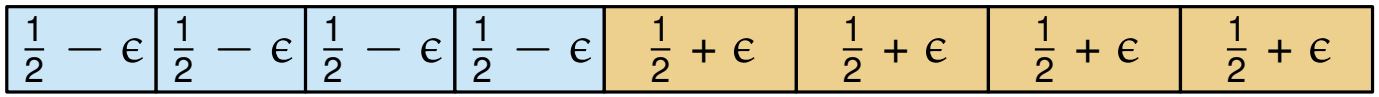
wobei $\mathcal{A}(I)$ den Wert der Lösung von $\mathcal{A}$ angewendet auf I und $\mathrm{OPT}(I)$ den Wert der optimalen Lösung bezeichnet.

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

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

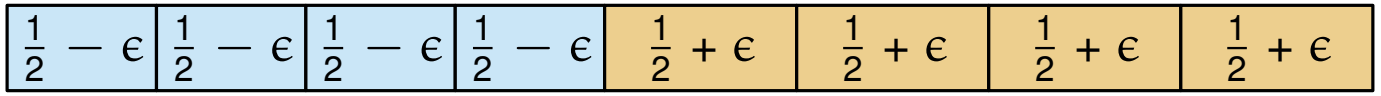
Beobachtung: Da nicht bekannt ist, wann Sequenz aufhört (Online-Algorithmus), muss die Aussage zu jedem Zeitpunkt gelten, wenn Sequenz $(a_1, \ldots, a_n)$ von $\mathcal{A}$ abgearbeitet wird.

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 $\frac{1}{2} + \epsilon$

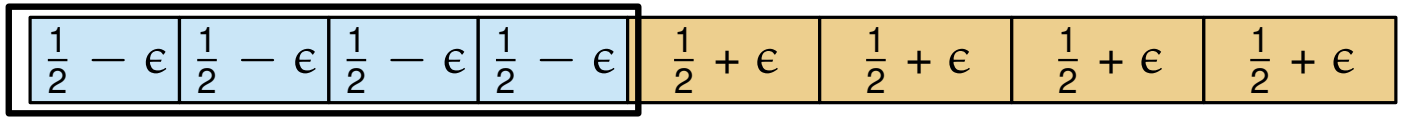
Problem 2



$\frac{n}{2}$ 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:

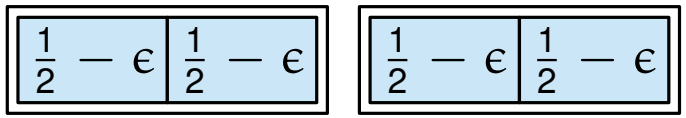
Problem 2



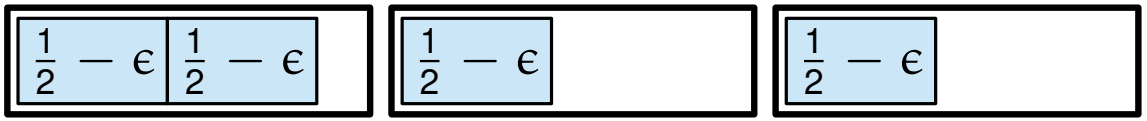
$\frac{n}{2}$ Elemente der Größe $\frac{1}{2} - \epsilon$ $\frac{n}{2}$ Elemente der Größe $\frac{1}{2} + \epsilon$

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

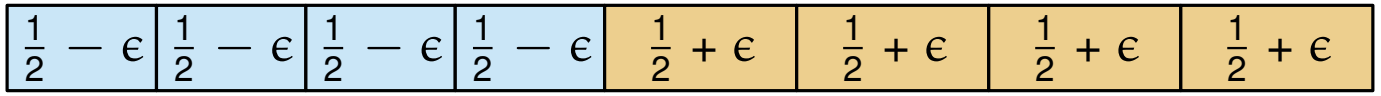
Optimale Lösung verwendet $\frac{n}{4}$ Bins:



$\mathcal{A}$ verwendet b Bins und damit gilt nach Annahme: $\frac{b}{\frac{n}{4}} = \frac{b \cdot 4}{n} < \frac{4}{3} \Leftrightarrow \frac{b}{n} < \frac{1}{3}$



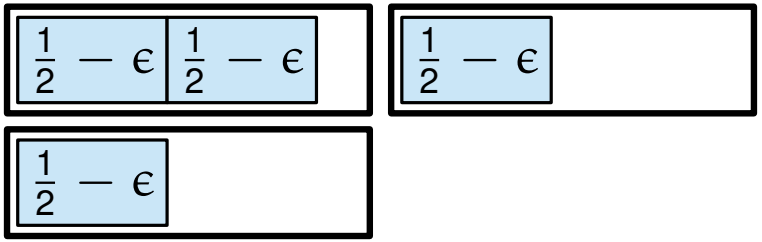
Problem 2



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

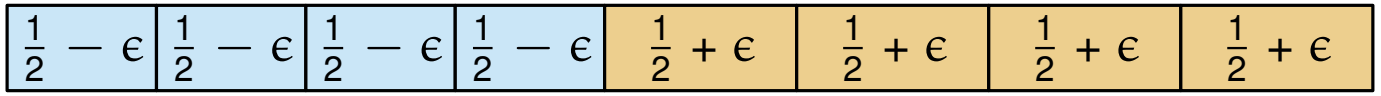
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

$OPT(I) = \frac{n}{4}$ 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

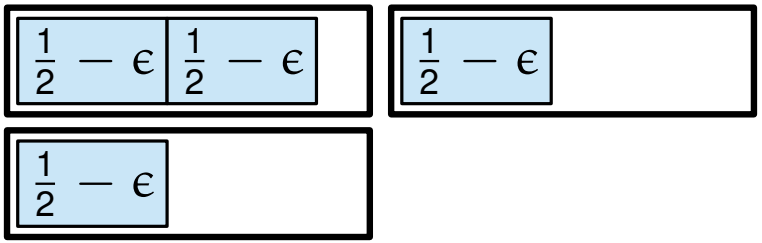
Problem 2



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

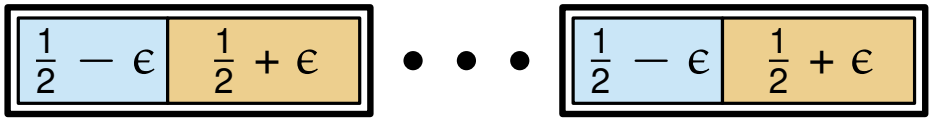
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

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

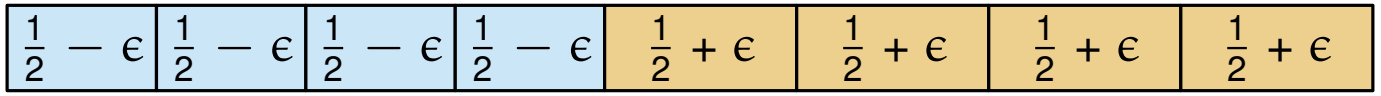


2. Fall: Betrachte Zeitpunkt, nachdem alle Elemente abgeabreitet worden sind:

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



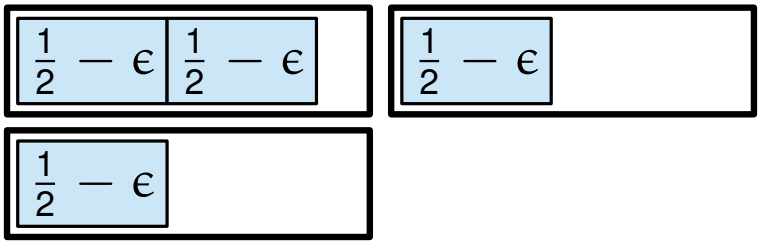
Problem 2



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

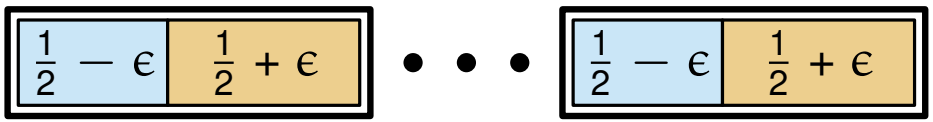
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

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

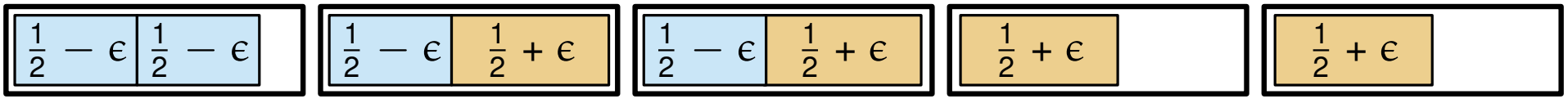


2. Fall: Betrachte Zeitpunkt, nachdem alle Elemente abgeabreitet worden sind:

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

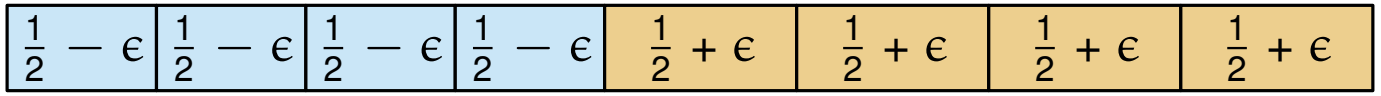


$\mathcal{A}$ kann maximal b Elemente der Größe $\frac{1}{2} + \epsilon$ in die ersten b Bins packen, da diese bereits mit $\frac{1}{2} - \epsilon$ Elementen belegt sind. Die restlichen Bins können jeweils nur ein Element enthalten.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

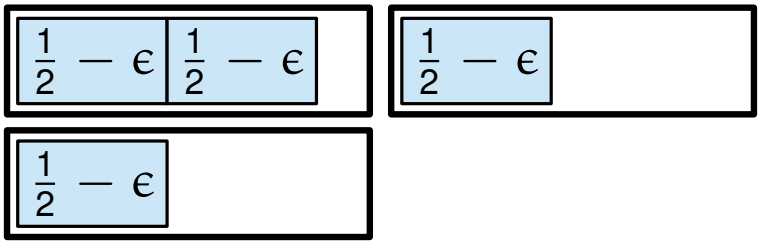
Problem 2



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

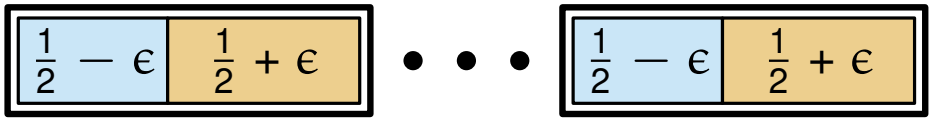
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

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

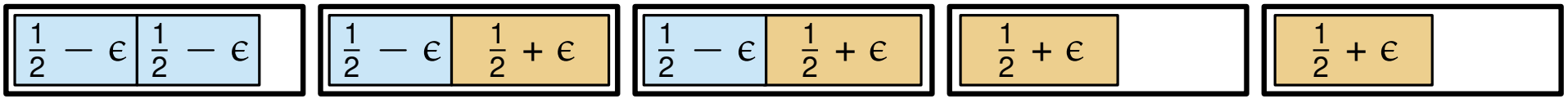


2. Fall: Betrachte Zeitpunkt, nachdem alle Elemente abgeabreitet worden sind:

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.

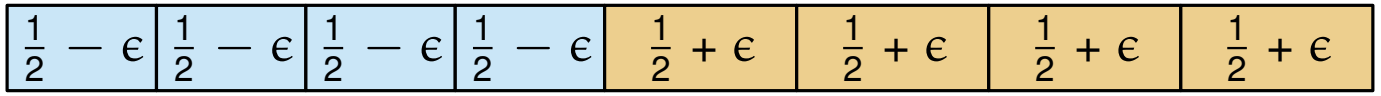


$\mathcal{A}$ benötigt mindestens $n - b$ Bins.

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

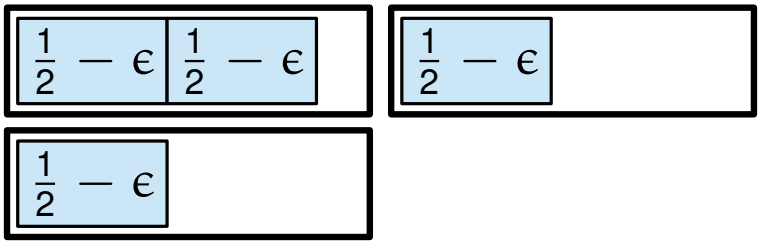
Problem 2



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

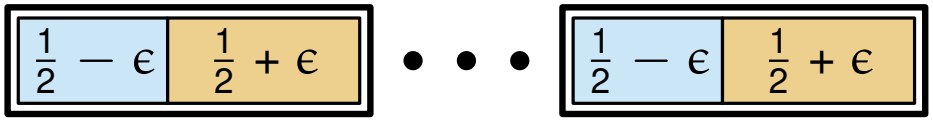
1. Fall: Betrachte Zeitpunkt, nachdem die ersten $\frac{n}{2}$ Elemente abgeabreitet worden sind:

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

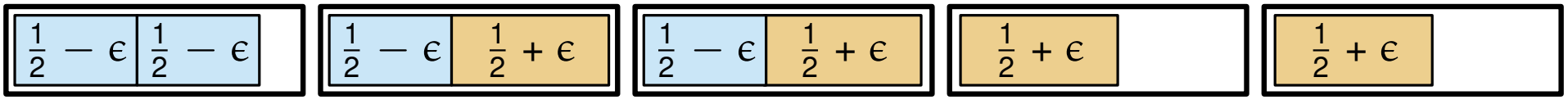


2. Fall: Betrachte Zeitpunkt, nachdem alle Elemente abgeabreitet worden sind:

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

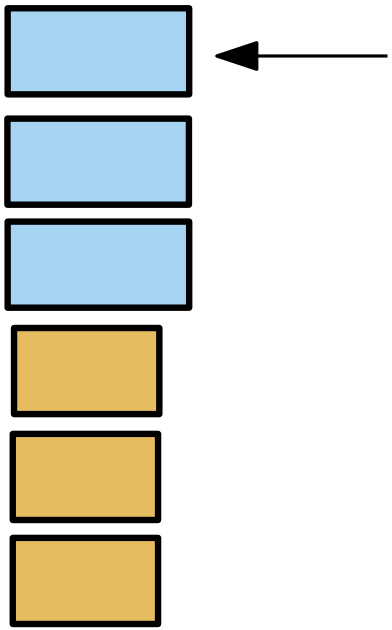
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

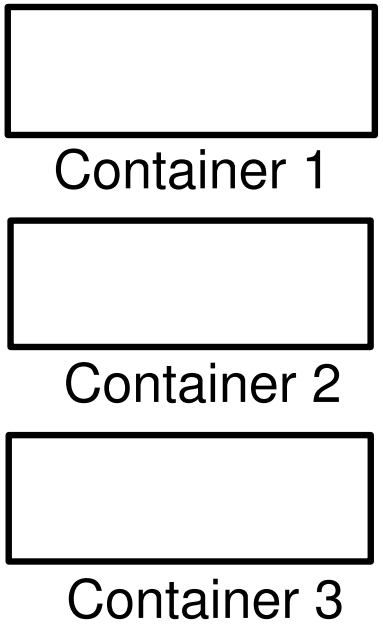
Problem 2

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?

Betrachte: FIRST FIT

Reihenfolge der Ankunft:

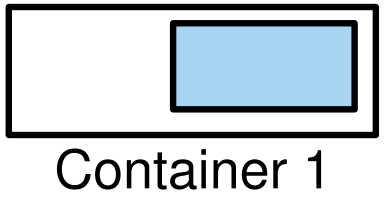




Problem 2

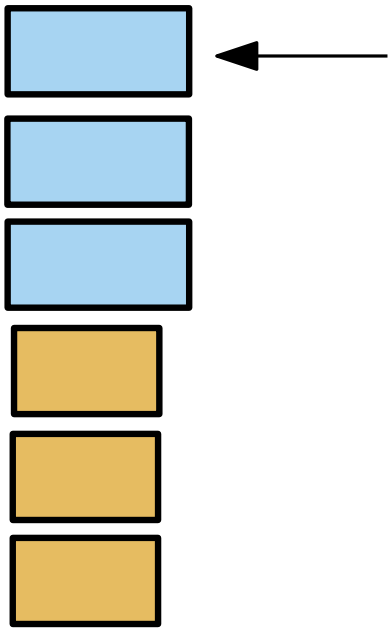
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu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

Betrachte: FIRST FIT



Container 1

Reihenfolge der Ankunft:



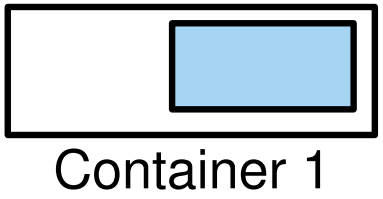
Container 2

Container 3

Problem 2

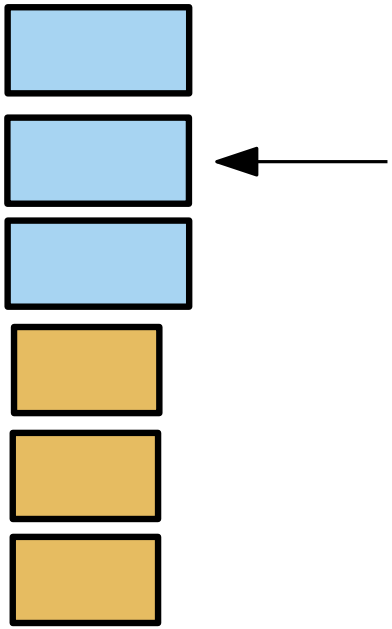
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu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

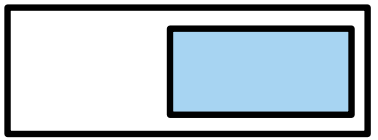
Betrachte: FIRST FIT



Container 1

Reihenfolge der Ankunft:





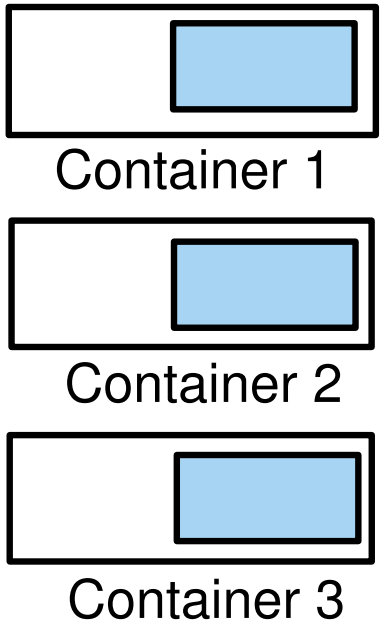
Container 2

Container 3

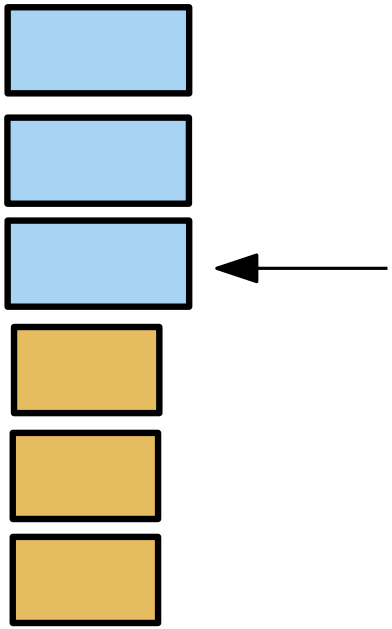
Problem 2

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?

Betrachte: FIRST FIT



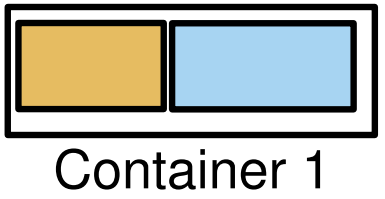
Reihenfolge der Ankunft:



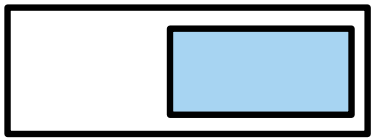
Problem 2

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?

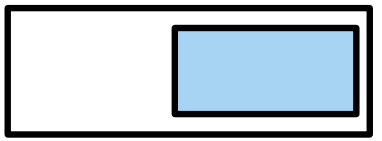
Betrachte: FIRST FIT



Container 1

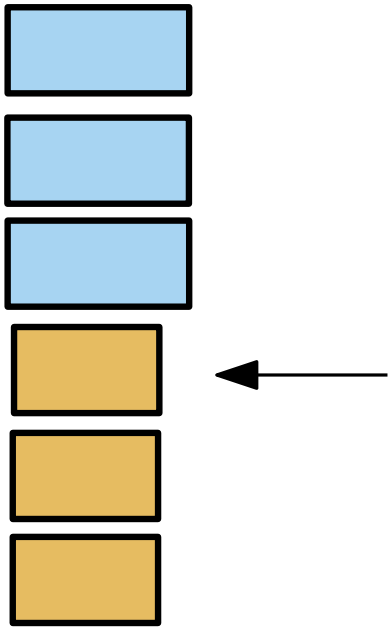


Container 2



Container 3

Reihenfolge der Ankunft:

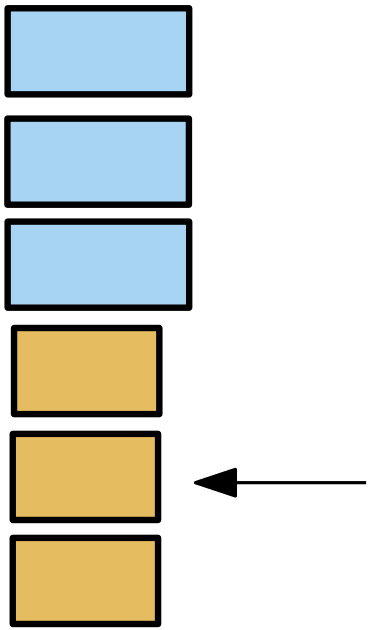


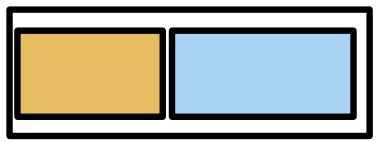
Problem 2

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?

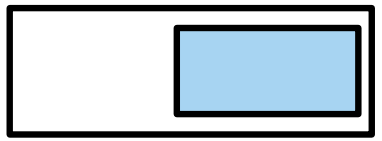
Betrachte: FIRST FIT

Reihenfolge der Ankunft:

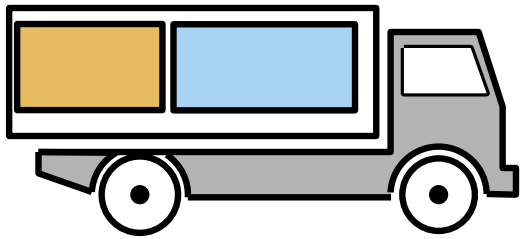




Container 2



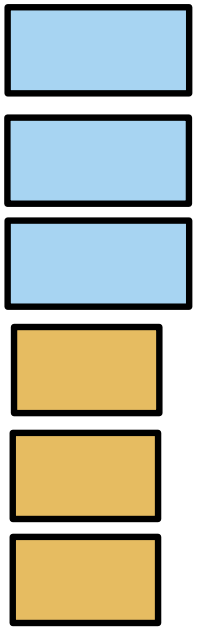
Container 3

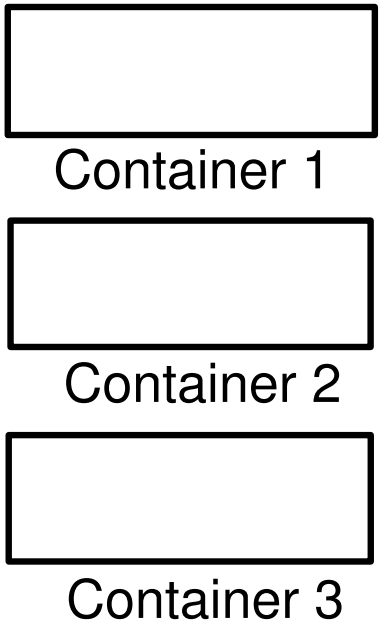


Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu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

Betrachte: NEXT FIT

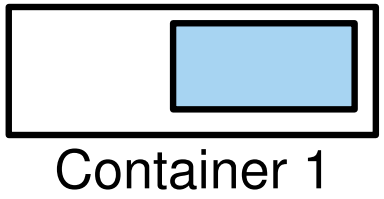
Reihenfolge der Ankunft:



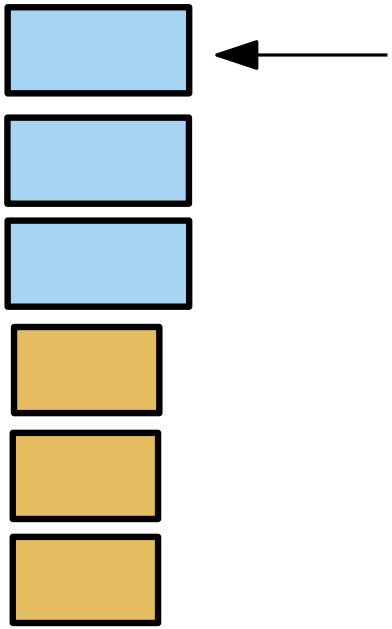


Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu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

Betrachte: NEXT FIT



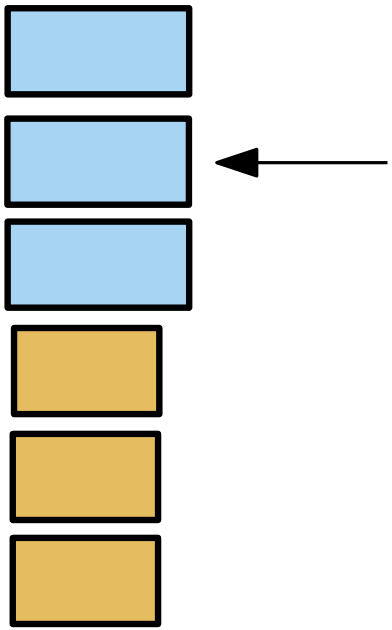
Reihenfolge der Ankunft:

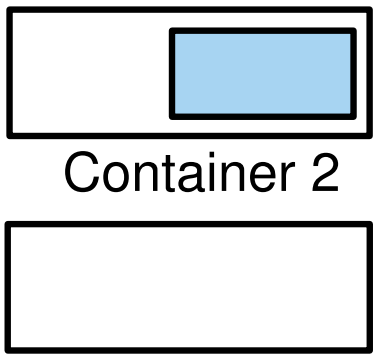


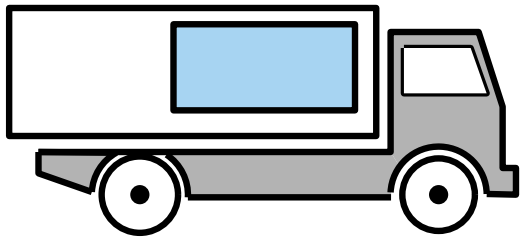
Nehmen Sie an, Sie besitzen ein Logistikunternehmen und müssen Container mit Waren beladen, die mit einzelnen Lkw-Ladungen heran transportiert werden. Da Sie keinen Platz zum Zwischenlagern der Waren haben, müssen Sie die ankommenden Waren direkt in die Container verfrachten. Sie entscheiden sich hierzu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

Betrachte: NEXT FIT

Reihenfolge der Ankunft:





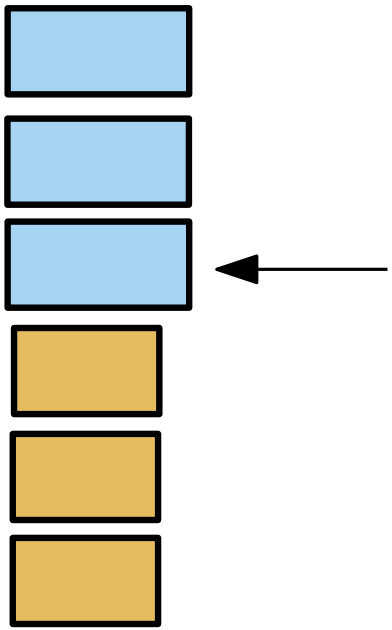


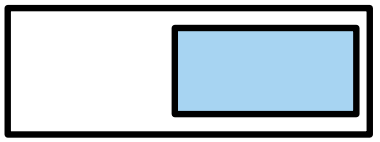
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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?

Betrachte: NEXT FIT

Reihenfolge der Ankunft:



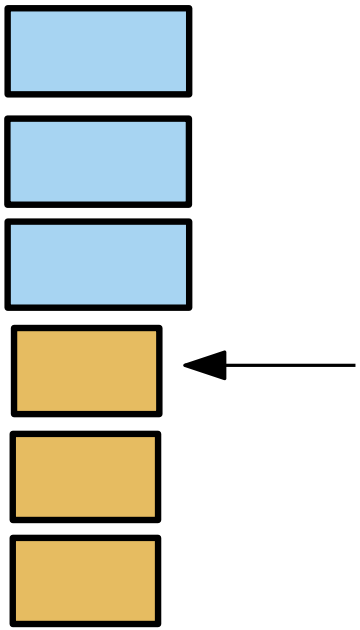


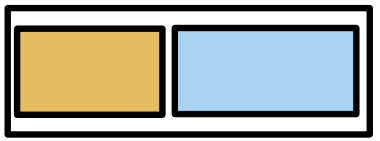
Container 3

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?

Betrachte: NEXT FIT

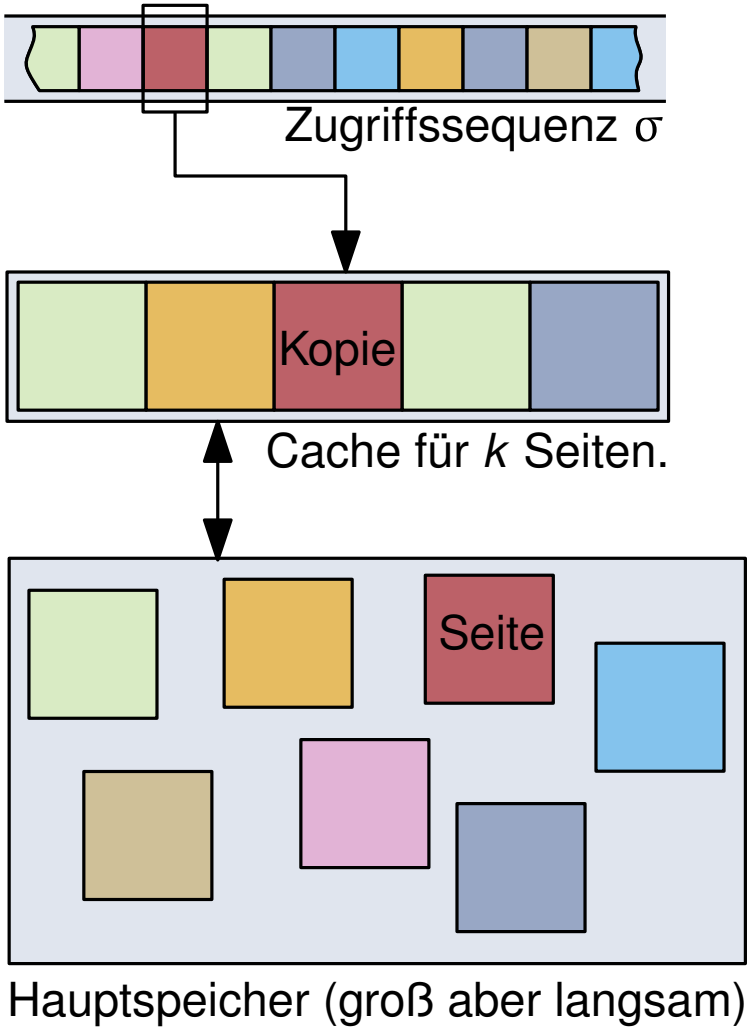
Reihenfolge der Ankunft:





Container 3

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?

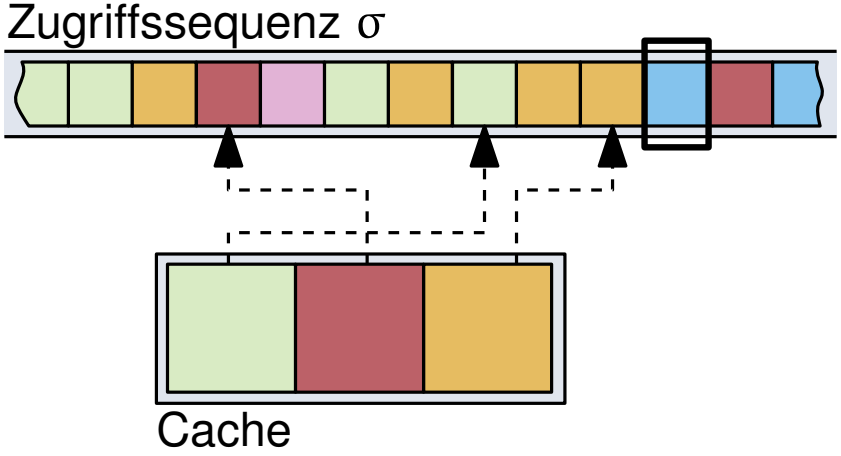
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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:



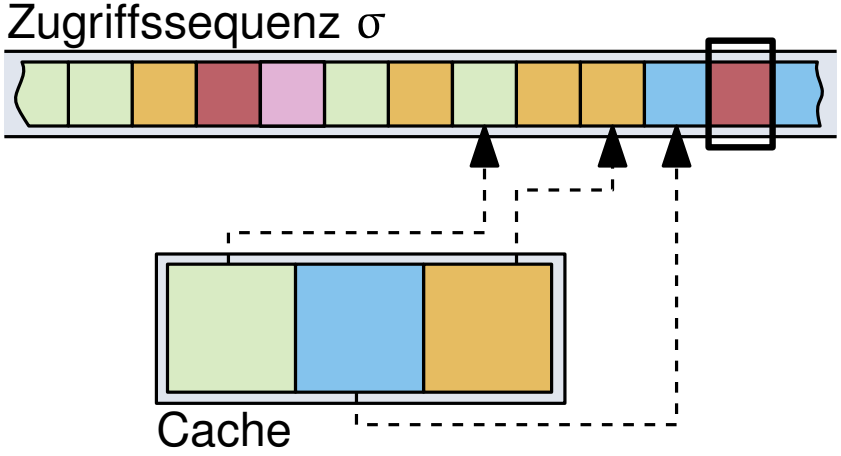
1. Speichere für jede Seite im Cache, wann sie zuletzt verwendet wurde.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Vergangenheit liegt.

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:



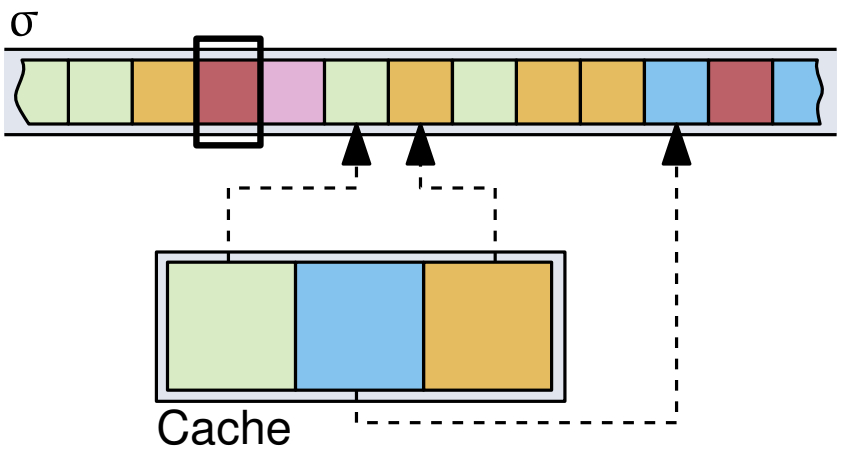
1. Speichere für jede Seite im Cache, wann sie zuletzt verwendet wurde.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Vergangenheit liegt.

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:



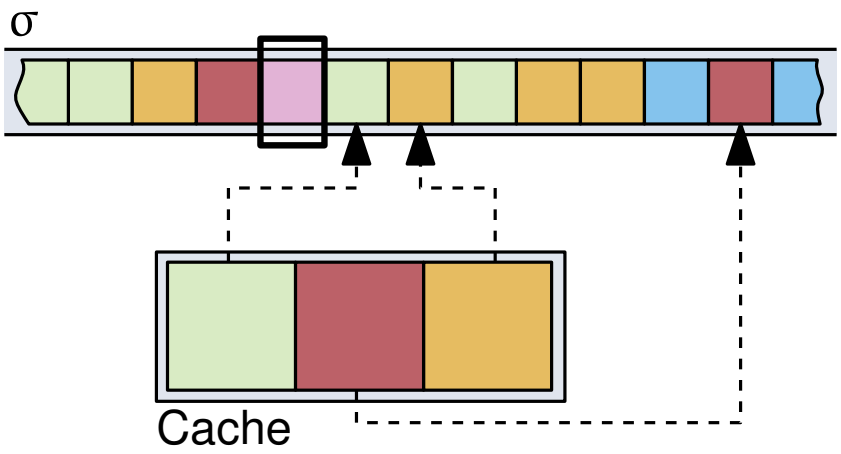
1. Speichere für jede Seite im Cache, wann sie als nächstes verwendet wird.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Zukunft liegt.

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

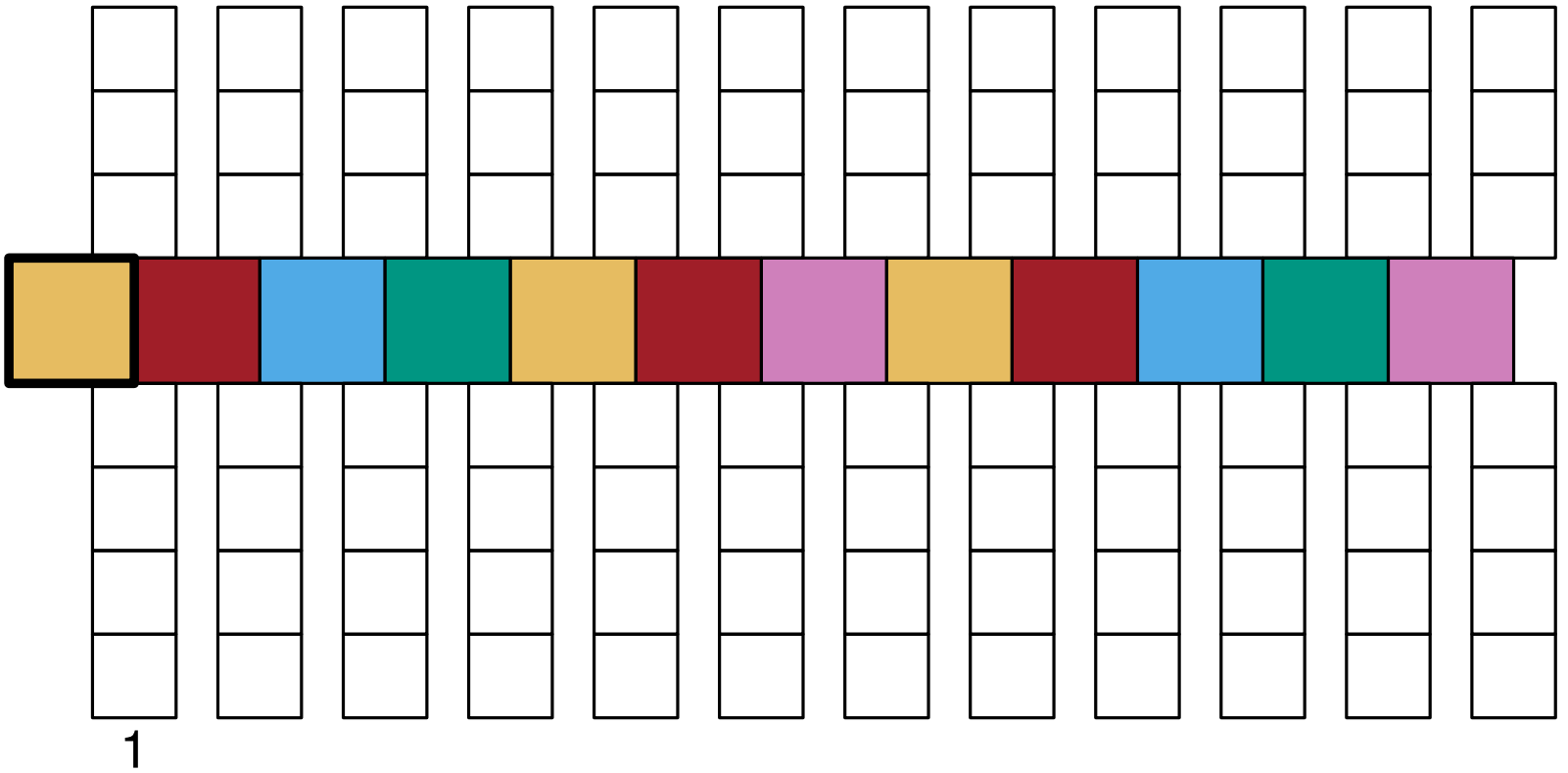
(h,k)-Paging-Problem:

- 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).

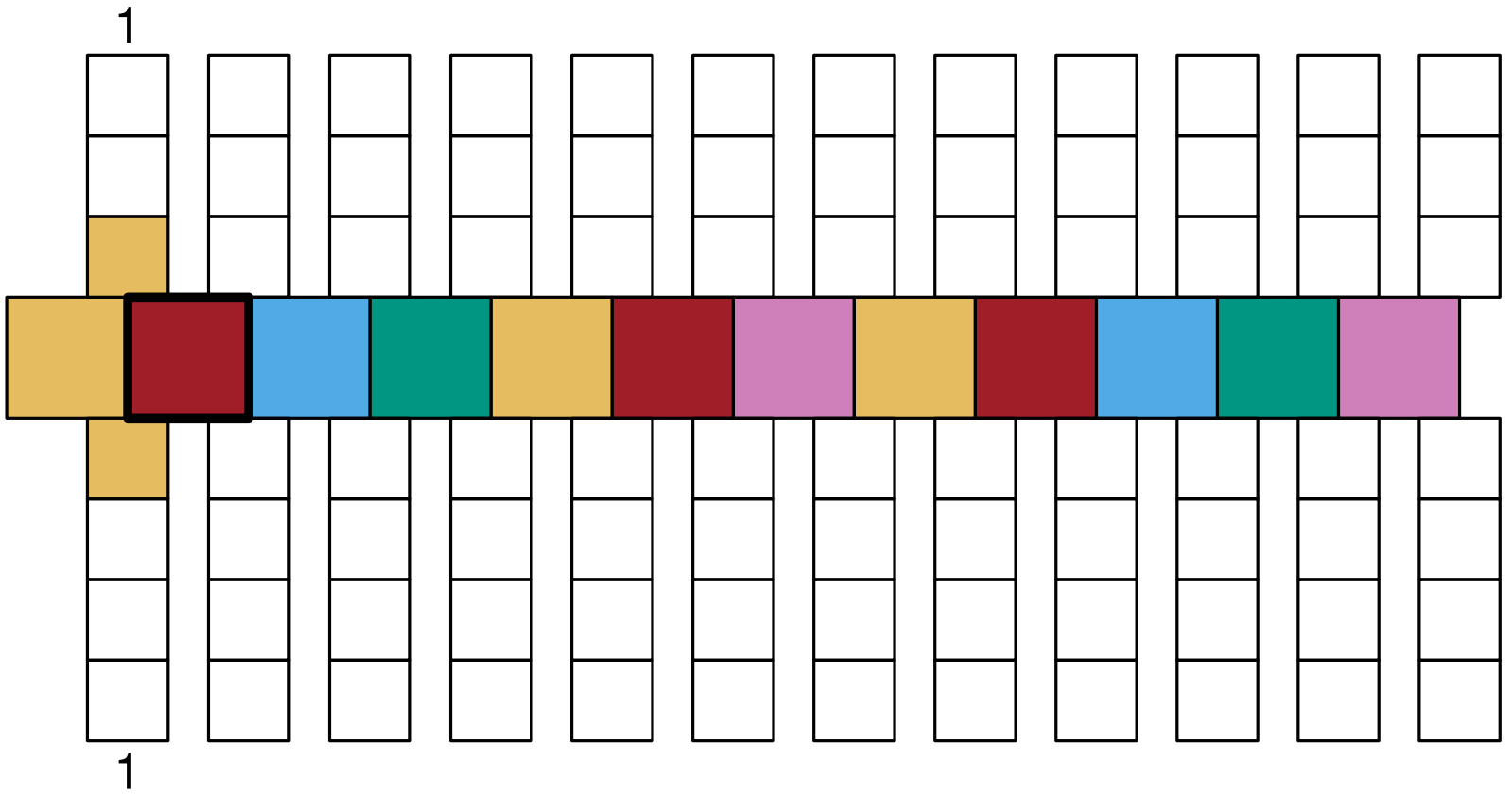
Problem 3

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.



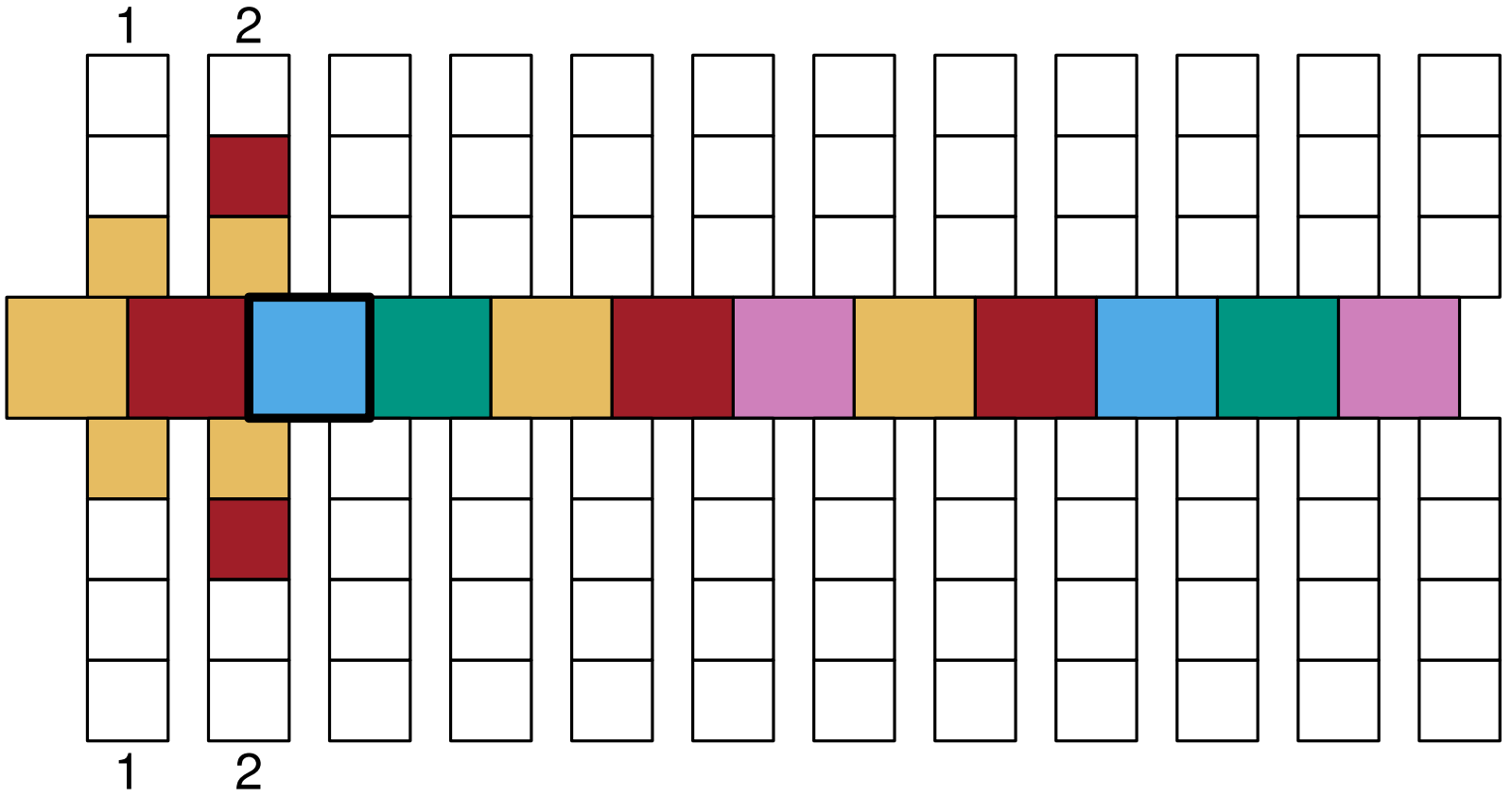
Problem 3

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

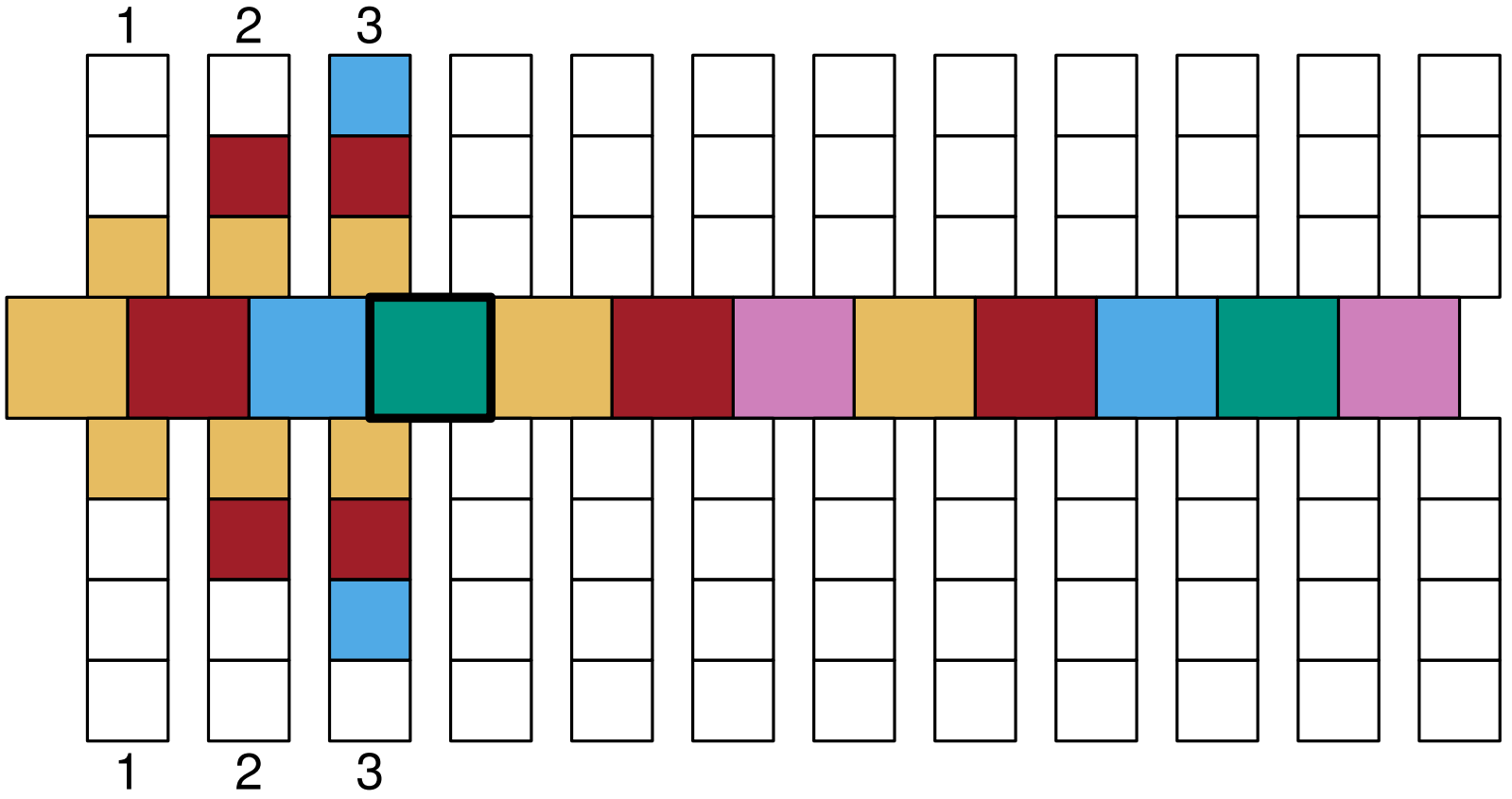
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

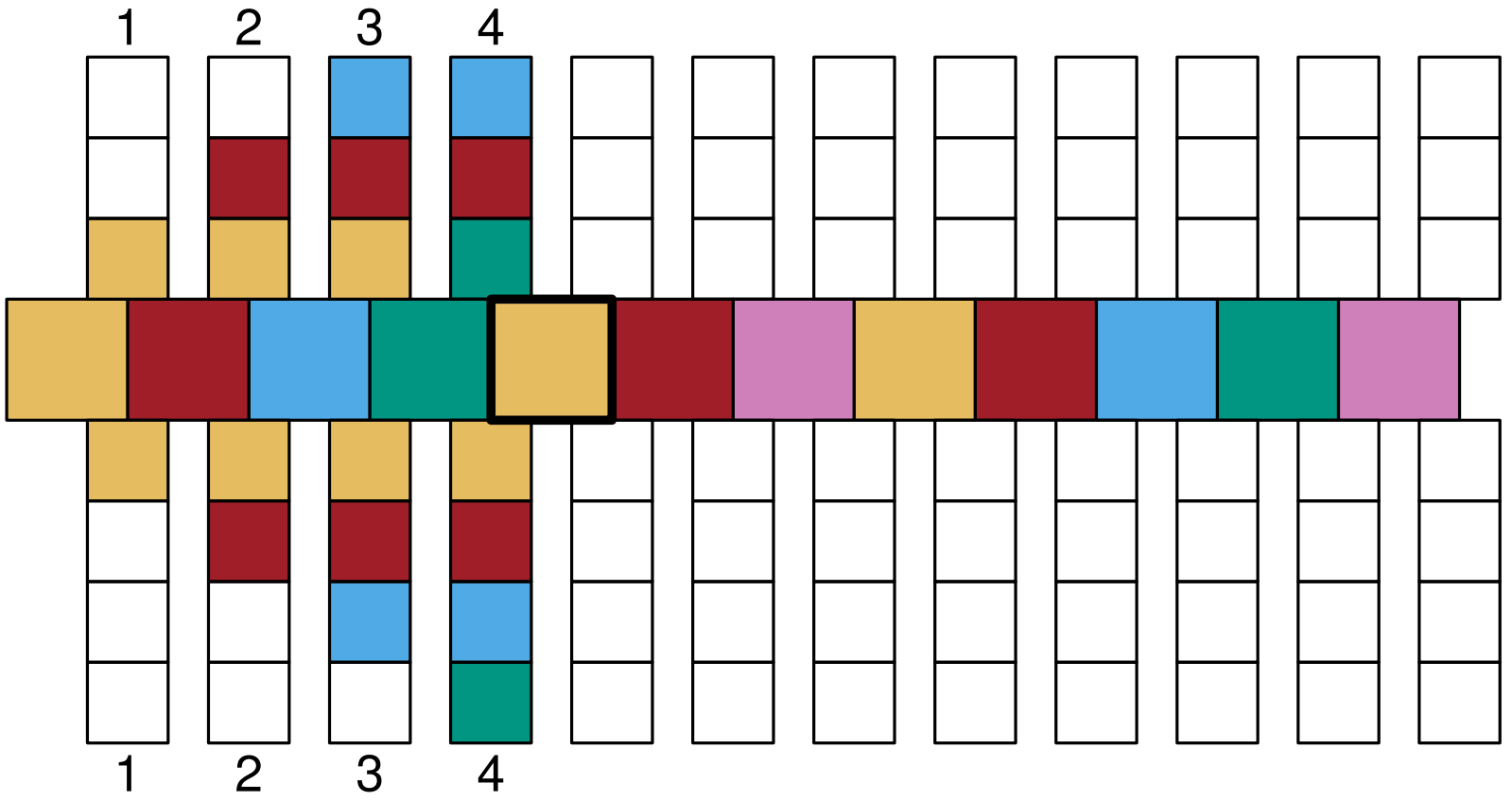
Problem 3

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

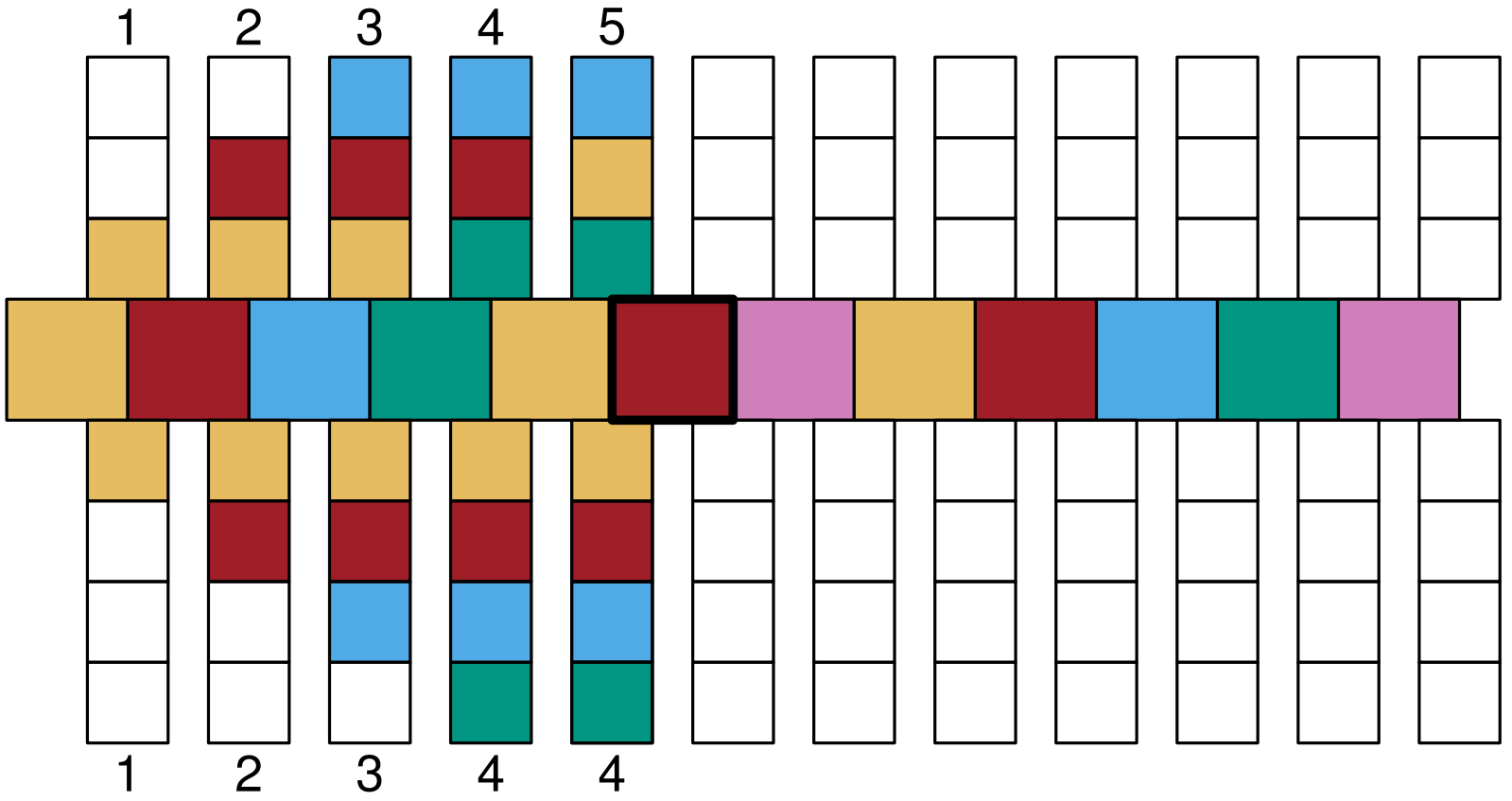
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

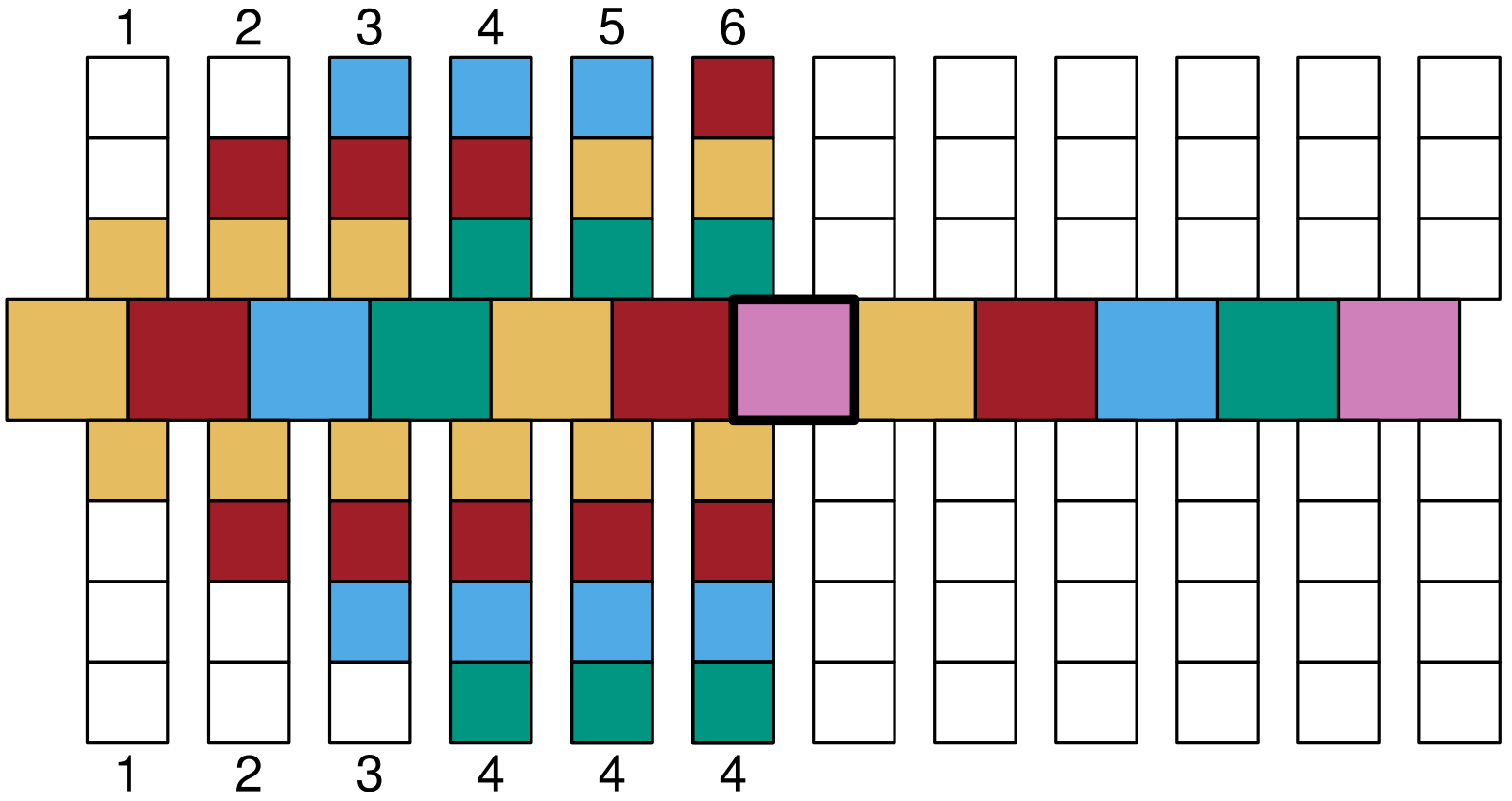
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

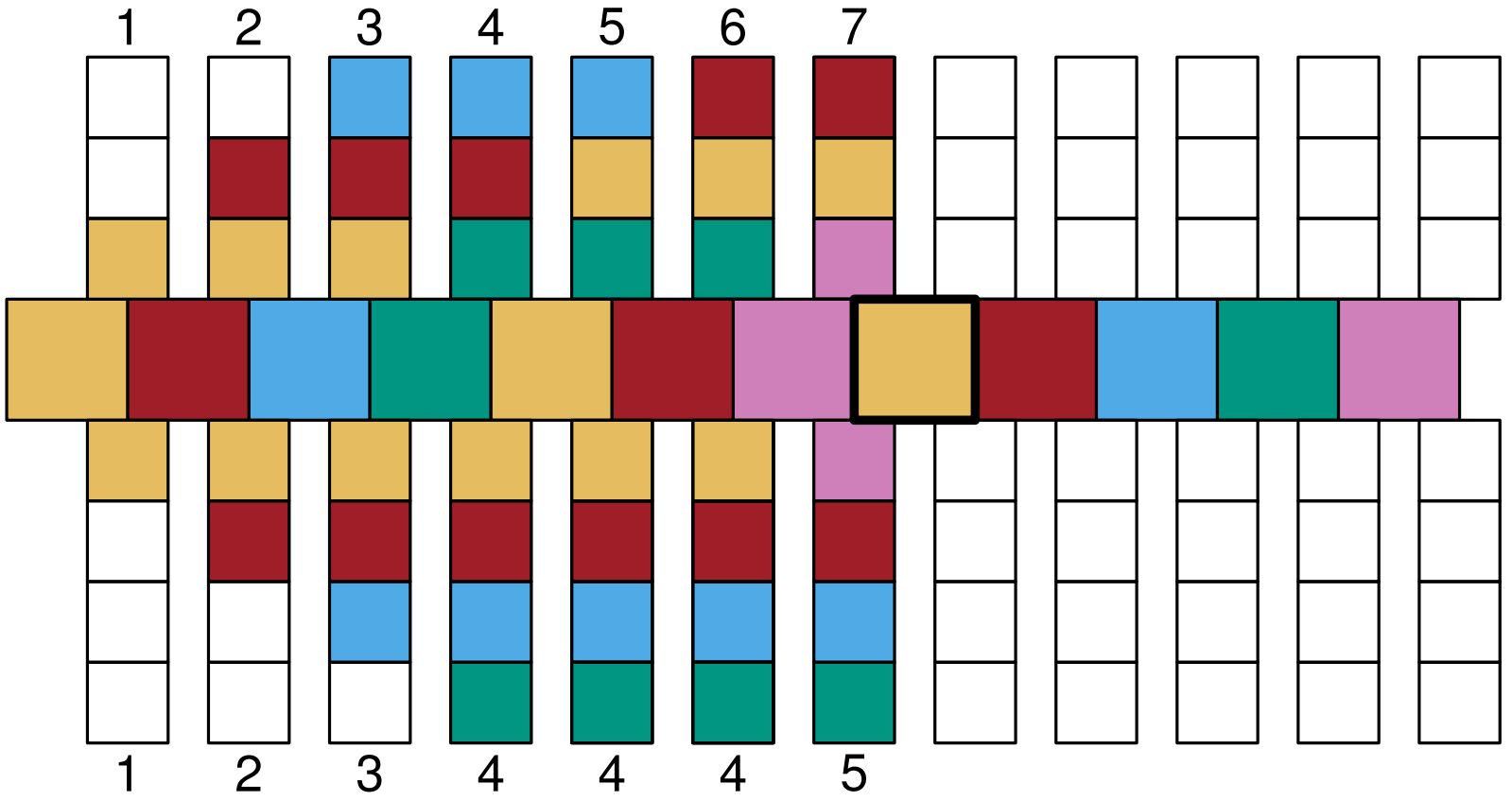
Problem 3

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.



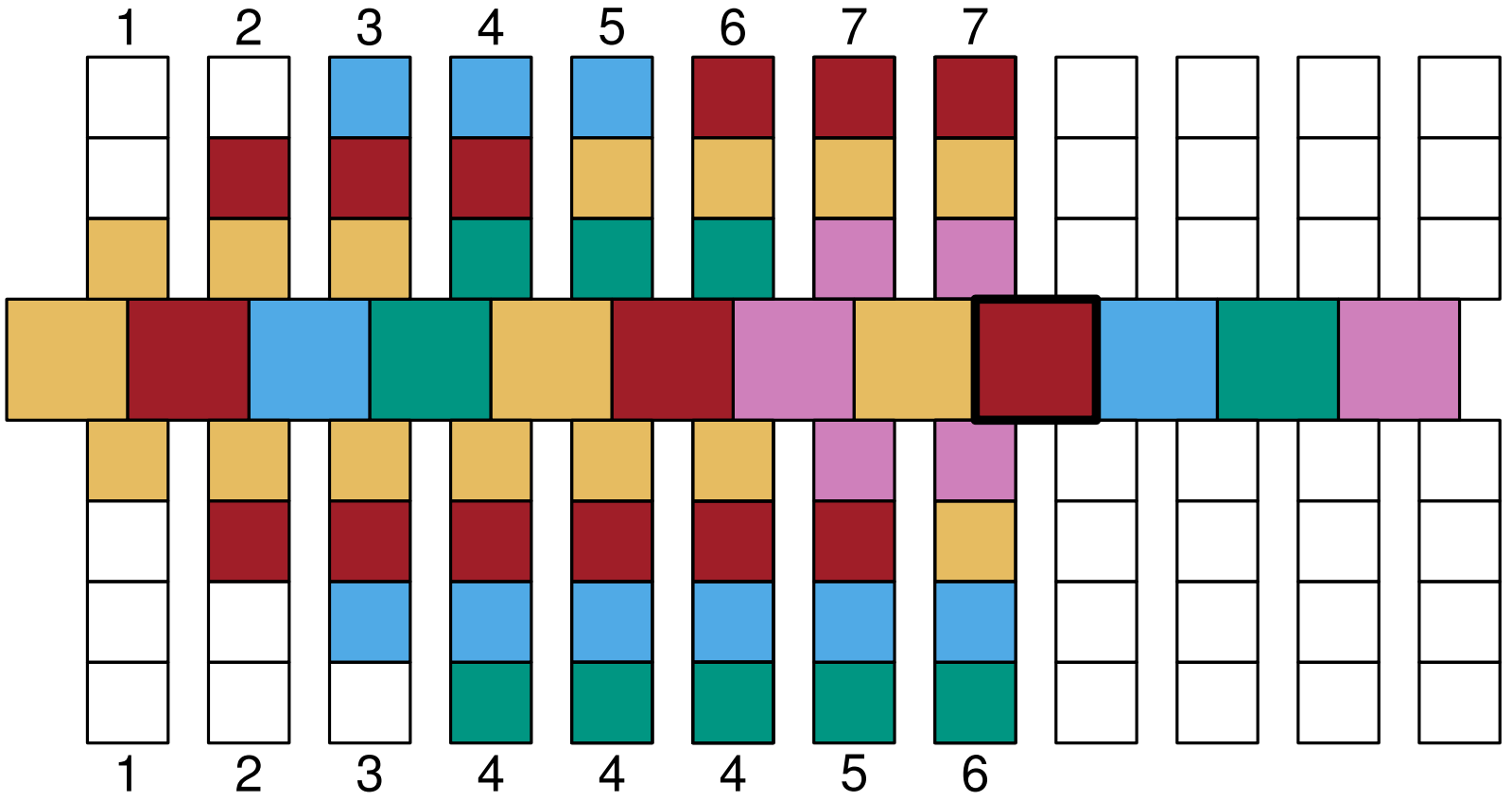
Problem 3

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.



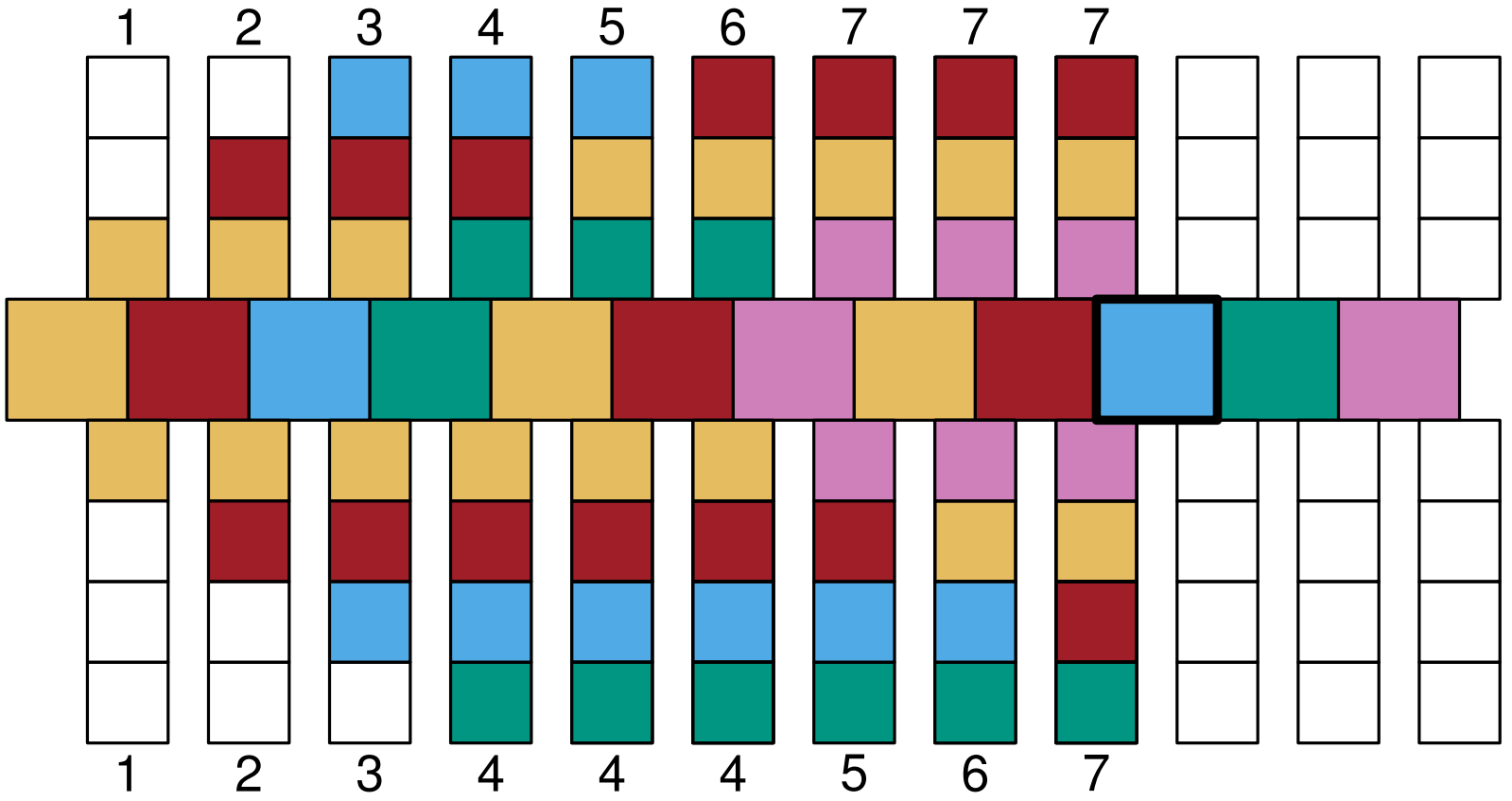
Problem 3

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

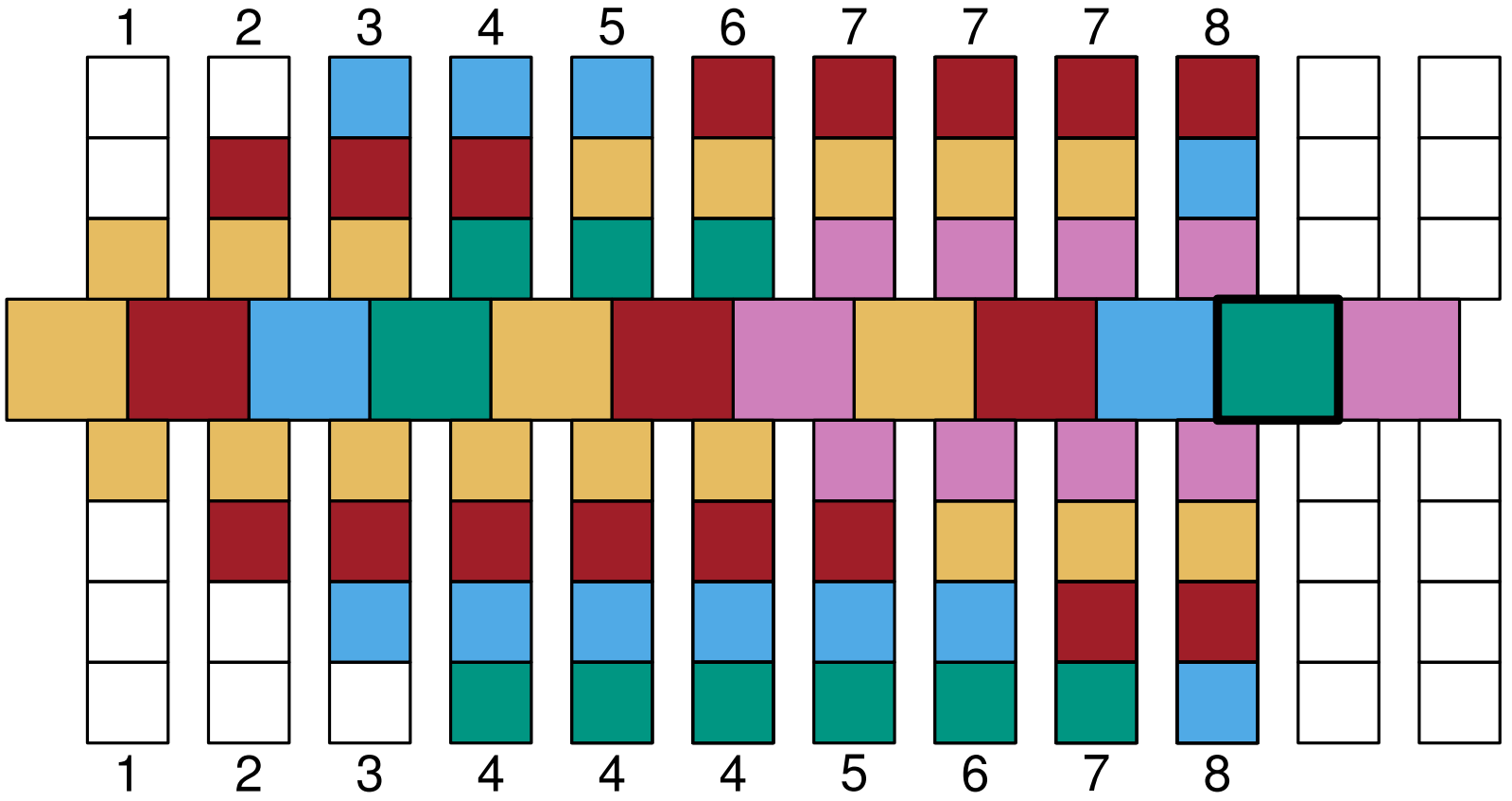
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

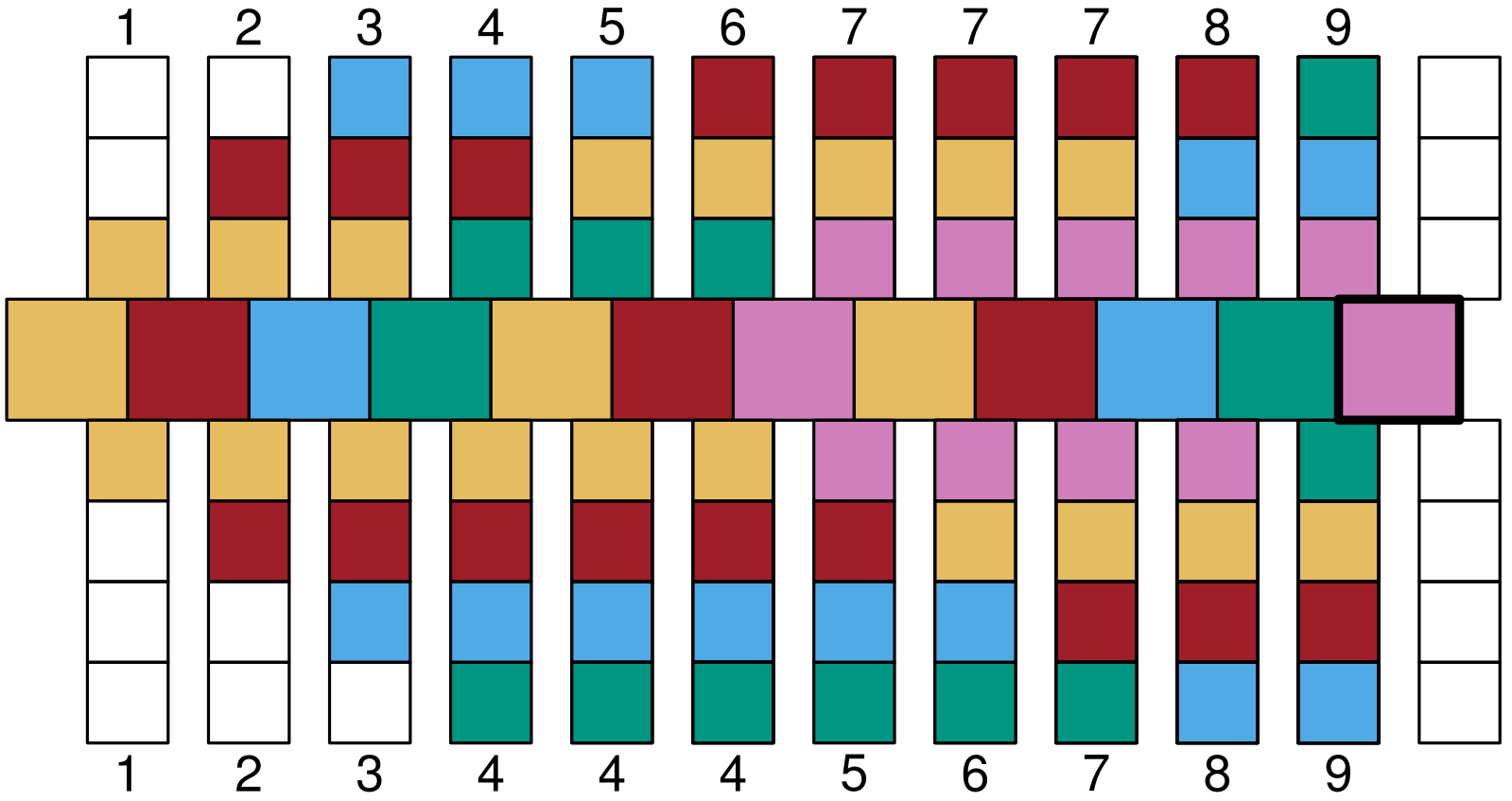
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

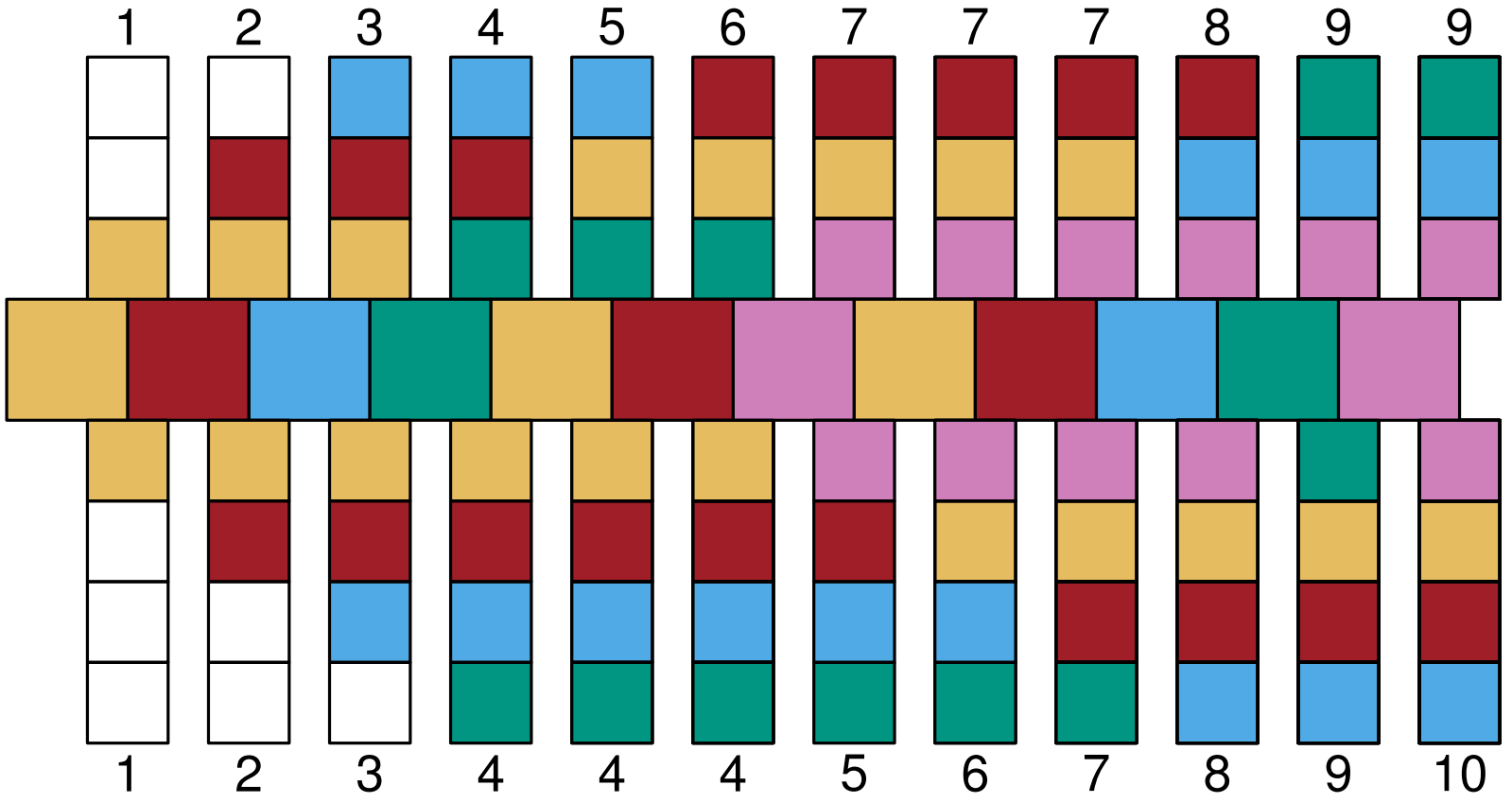
Die Béládys Anomalie besagt, dass für manche der Paging-Algorithmen man Zugriffssequenzen finden kann, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. Zeigen Sie, dass FIFO ein solcher Paging-Algorithmus ist.



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 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:

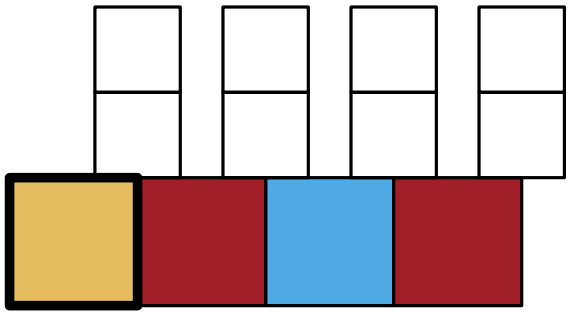
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

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:



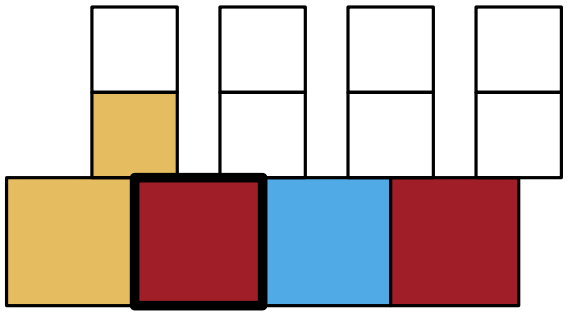
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

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:



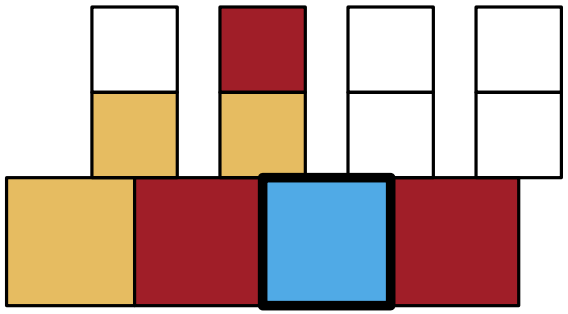
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

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:



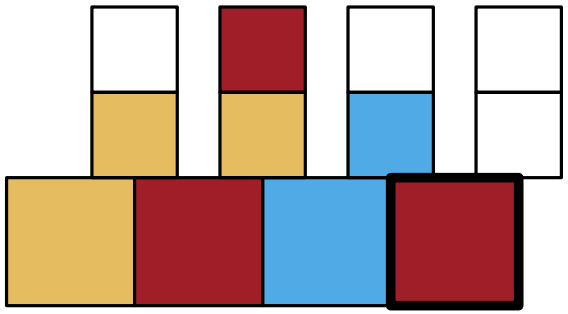
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

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:



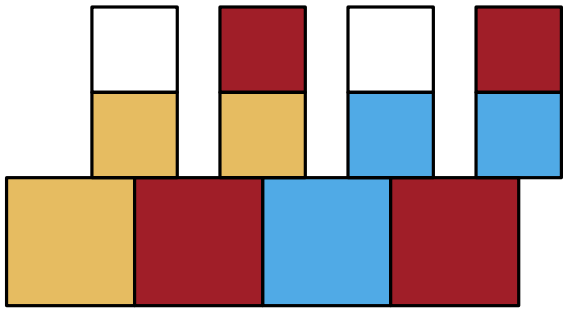
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

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:



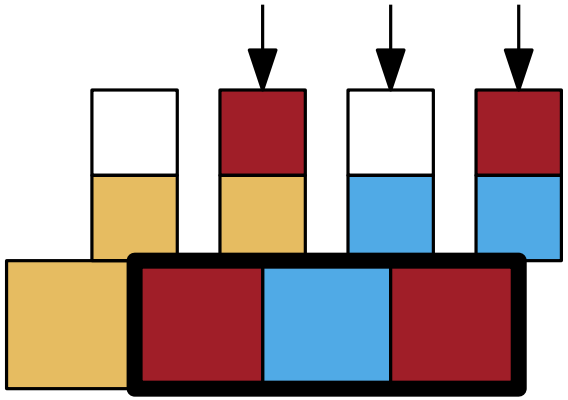
Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

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.

Karlsruher Institut für Technologie

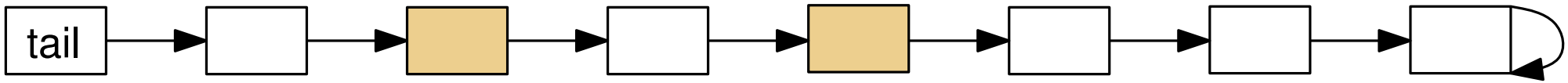
Parallele Algorithmen

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 5

Gegeben eine einfach verkettete Liste L der Länge n, sodass einige Elemnte in L markiert sind. Geben Sie einen parallelen Algorithmus an, der eine Teilliste von L ausgibt, die genau die markierten Elemente enthält. Hinweis: Nehmen Sie an, dass der Kopf von L auf sich selbst zeigt.

1. Phase
für $j = 1$ **bis** $\log n$ **tue**
  Für alle $i : 1 \leq i \leq n$ **führe parallel aus**
    wenn NEXT(i) *ist nicht markiert* **dann**
      NEXT$(i) \leftarrow$ NEXT(NEXT(i))

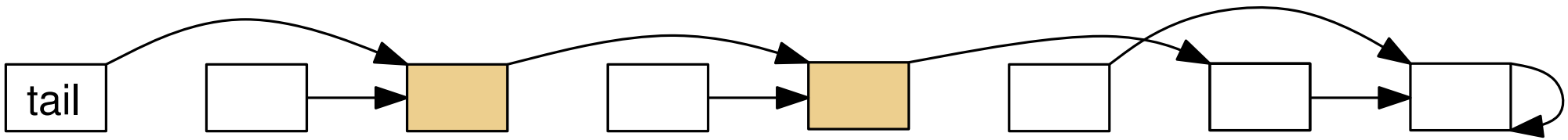


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 5

Gegeben eine einfach verkettete Liste L der Länge n, sodass einige Elemnte in L markiert sind. Geben Sie einen parallelen Algorithmus an, der eine Teilliste von L ausgibt, die genau die markierten Elemente enthält. Hinweis: Nehmen Sie an, dass der Kopf von L auf sich selbst zeigt.

1. Phase
für $j = 1$ **bis** $\log n$ **tue**
  Für alle $i : 1 \leq i \leq n$ **führe parallel aus**
    wenn NEXT(i) *ist nicht markiert* **dann**
      NEXT$(i) \leftarrow$ NEXT(NEXT(i))

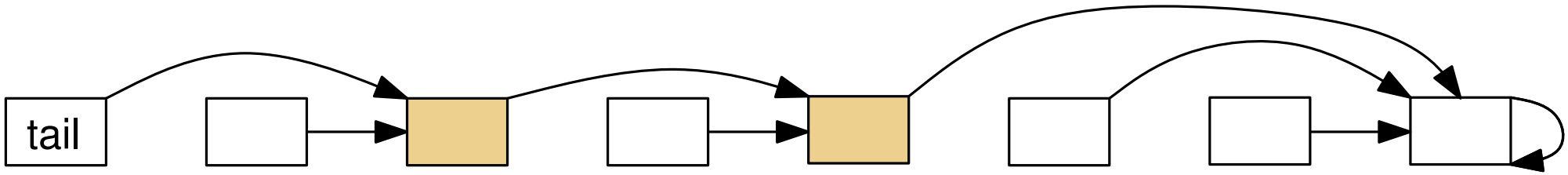


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 5

Gegeben eine einfach verkettete Liste L der Länge n, sodass einige Elemnte in L markiert sind. Geben Sie einen parallelen Algorithmus an, der eine Teilliste von L ausgibt, die genau die markierten Elemente enthält. Hinweis: Nehmen Sie an, dass der Kopf von L auf sich selbst zeigt.

1. Phase
für $j = 1$ **bis** $\log n$ **tue**
  Für alle $i : 1 \leq i \leq n$ **führe parallel aus**
    wenn NEXT(i) *ist nicht markiert* **dann**
      NEXT$(i) \leftarrow$ NEXT(NEXT(i))



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 5

Gegeben eine einfach verkettete Liste L der Länge n, sodass einige Elemnte in L markiert sind. Geben Sie einen parallelen Algorithmus an, der eine Teilliste von L ausgibt, die genau die markierten Elemente enthält. Hinweis: Nehmen Sie an, dass der Kopf von L auf sich selbst zeigt.

2. Phase

Für alle $i : 1 \leq i \leq n$ **führe parallel aus**

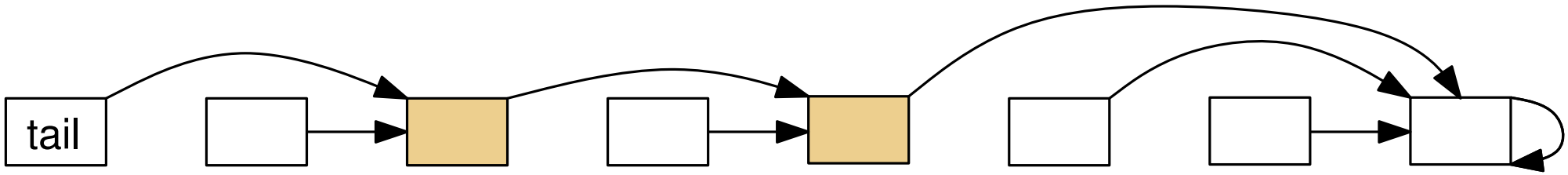
  wenn TAIL(L) = i **dann**

    wenn i *ist nicht markiert* **dann**

      TAIL(L) $\leftarrow$ NEXT(TAIL(L))

  wenn NEXT(i) *ist nicht markiert* **dann**

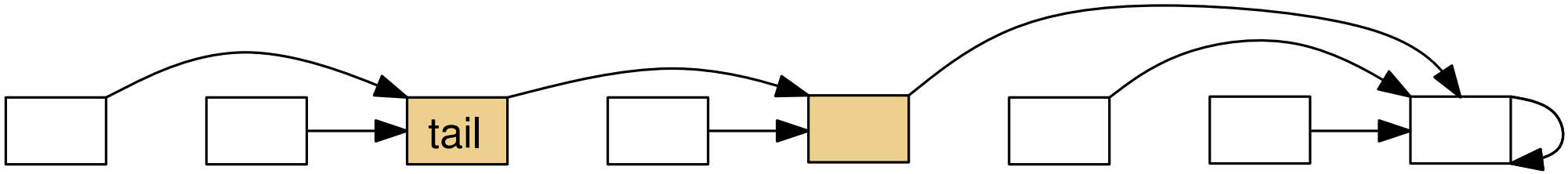
    NEXT(i) $\leftarrow$ i



Problem 5

Gegeben eine einfach verkettete Liste L der Länge n, sodass einige Elemnte in L markiert sind. Geben Sie einen parallelen Algorithmus an, der eine Teilliste von L ausgibt, die genau die markierten Elemente enthält. Hinweis: Nehmen Sie an, dass der Kopf von L auf sich selbst zeigt.

2. Phase
Für alle $i : 1 \leq i \leq n$ **führe parallel aus**
  wenn TAIL(L) = i **dann**
    wenn i *ist nicht markiert* **dann**
      TAIL(L) $\leftarrow$ NEXT(TAIL(L))
  wenn NEXT(i) *ist nicht markiert* **dann**
    NEXT(i) $\leftarrow$ i

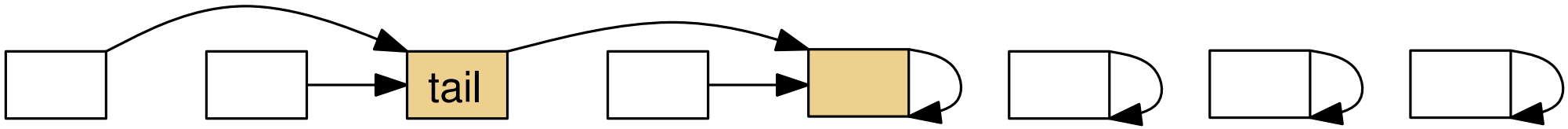


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 5

Gegeben eine einfach verkettete Liste L der Länge n, sodass einige Elemnte in L markiert sind. Geben Sie einen parallelen Algorithmus an, der eine Teilliste von L ausgibt, die genau die markierten Elemente enthält. Hinweis: Nehmen Sie an, dass der Kopf von L auf sich selbst zeigt.

2. Phase
Für alle $i : 1 \leq i \leq n$ **führe parallel aus**
  wenn TAIL(L) = i **dann**
    wenn i *ist nicht markiert* **dann**
      TAIL(L) $\leftarrow$ NEXT(TAIL(L))
  wenn NEXT(i) *ist nicht markiert* **dann**
    NEXT(i) $\leftarrow$ i

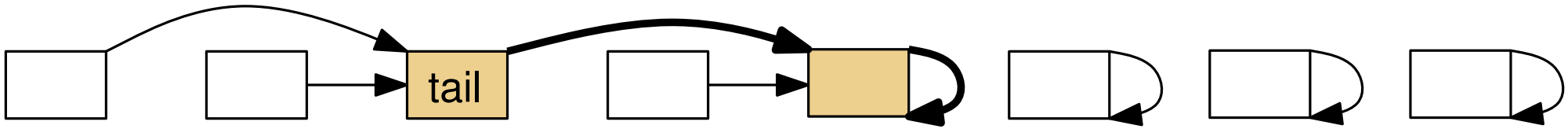


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 5

Gegeben eine einfach verkettete Liste L der Länge n, sodass einige Elemnte in L markiert sind. Geben Sie einen parallelen Algorithmus an, der eine Teilliste von L ausgibt, die genau die markierten Elemente enthält. Hinweis: Nehmen Sie an, dass der Kopf von L auf sich selbst zeigt.

2. Phase
Für alle $i : 1 \leq i \leq n$ **führe parallel aus**
  wenn TAIL(L) = i **dann**
    wenn i *ist nicht markiert* **dann**
      TAIL(L) $\leftarrow$ NEXT(TAIL(L))
  wenn NEXT(i) *ist nicht markiert* **dann**
    NEXT(i) $\leftarrow$ i



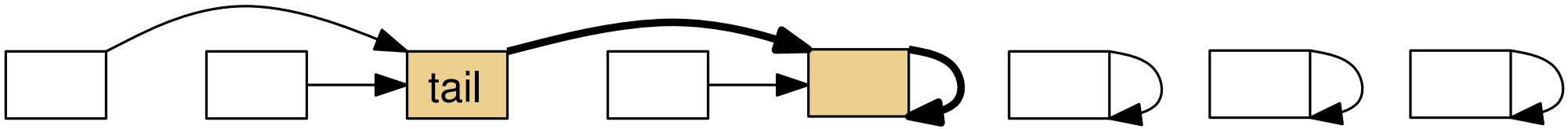
Kosten: $O(n \log n)$ Geht es besser?

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 5

Gegeben eine einfach verkettete Liste L der Länge n, sodass einige Elemnte in L markiert sind. Geben Sie einen parallelen Algorithmus an, der eine Teilliste von L ausgibt, die genau die markierten Elemente enthält. Hinweis: Nehmen Sie an, dass der Kopf von L auf sich selbst zeigt.

2. Phase
Für alle $i : 1 \leq i \leq n$ **führe parallel aus**
  wenn TAIL(L) = i **dann**
    wenn i *ist nicht markiert* **dann**
      TAIL(L) $\leftarrow$ NEXT(TAIL(L))
  wenn NEXT(i) *ist nicht markiert* **dann**
    NEXT(i) $\leftarrow$ i



Kosten: $O(n \log n)$ Geht es besser? Verwende sequentiellen Algorithmus mit Laufzeit $O(n)$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

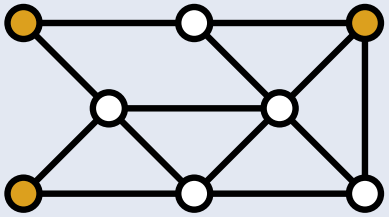
Parametrisierte Algorithmen – Nachtrag

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

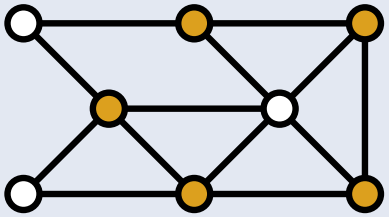
Drei Probleme

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

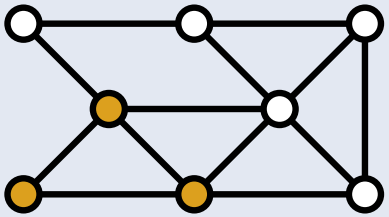
Problem: INDEPENDENT SET
Finde *unabhängige Menge* $V' \subseteq V$ mit $|V'| \geq k$. V' heißt unabhängig genau dann, wenn für alle $v, w \in V'$ gilt $\{v, w\} \notin E$.



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 $w \in V'$.



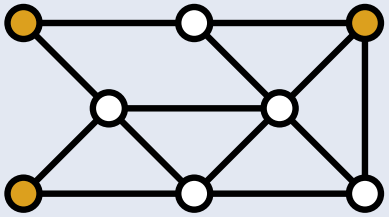
Problem: CLIQUE
Finde *Clique* $V' \subseteq V$ mit $|V'| \geq k$. V' heißt *Clique* genau dann, wenn für jedes Paar $u, v \in V'$ gilt, dass $\{u, v\} \in E$.



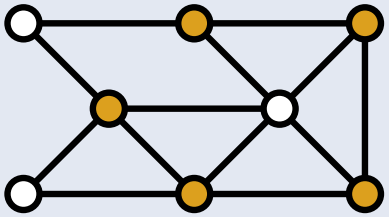
Drei Probleme

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

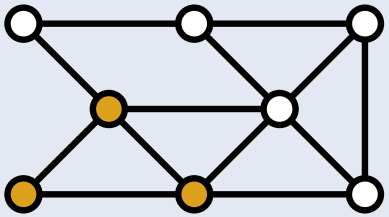
Problem: INDEPENDENT SET
Finde *unabhängige Menge* $V' \subseteq V$ mit $|V'| \geq k$. V' heißt unabhängig genau dann, wenn für alle $v, w \in V'$ gilt $\{v, w\} \notin E$.



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 $\{u, v\} \in E$.



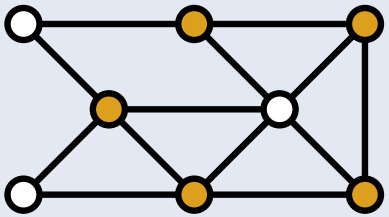
INDEPENDENT SET, VERTEX COVER und CLIQUE sind $\mathcal{NP}$-schwer.

$\rightarrow$ 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?

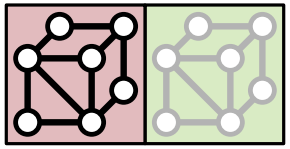
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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 $w \in V'$.



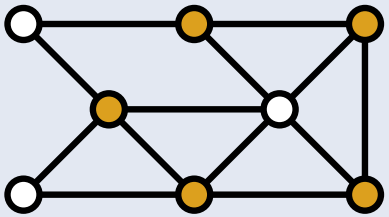
noch zu überdeckender Teilgraph ausgewählte Knoten & überdeckte Kanten



Gibt es ein VERTEX COVER mit maximal 3 Knoten?

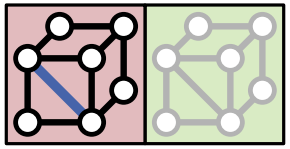
Algorithmus für VERTEX COVER

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.

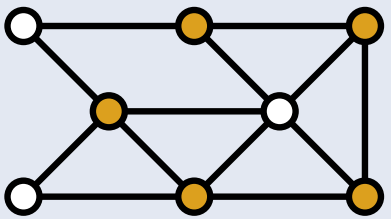


Gibt es ein VERTEX COVER mit maximal 3 Knoten?

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

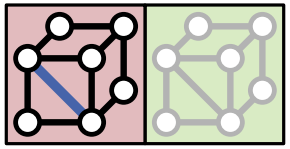
Algorithmus für VERTEX COVER

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 $w \in V'$.

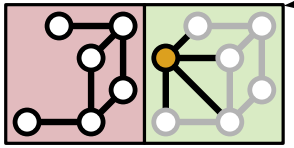


noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

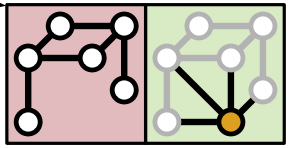
Jede Kante muss noch überdeckt werden. $\rightarrow$ wähle eine beliebige.



Gibt es ein VERTEX COVER mit maximal 3 Knoten?



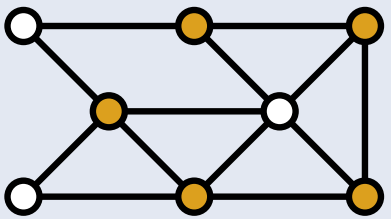
Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
$\rightarrow$ binärer Entscheidungsbaum



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

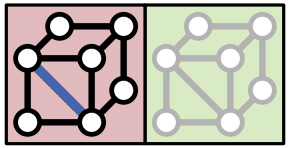
Algorithmus für VERTEX COVER

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 $w \in V'$.

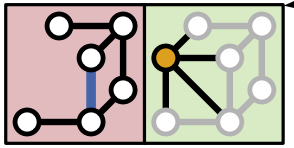


noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

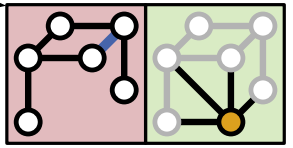
Jede Kante muss noch überdeckt werden. $\rightarrow$ wähle eine beliebige.



Gibt es ein VERTEX COVER mit maximal 3 Knoten?



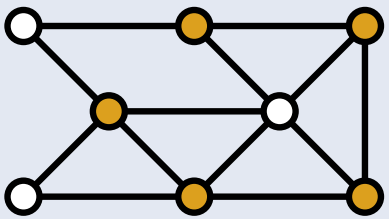
Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
$\rightarrow$ binärer Entscheidungsbaum



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

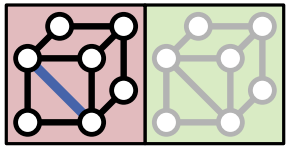
Algorithmus für VERTEX COVER

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 $w \in V'$.



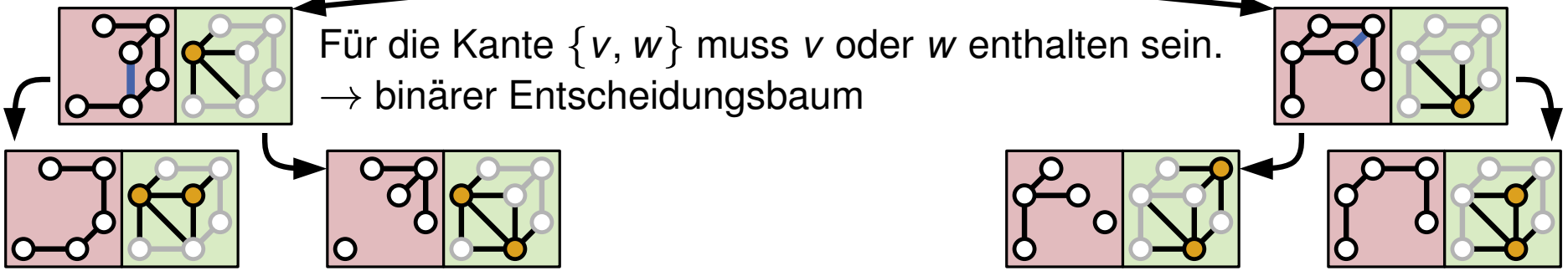
noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

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



Gibt es ein VERTEX COVER mit maximal 3 Knoten?

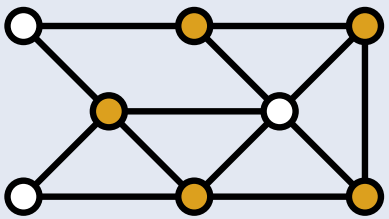
Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
→ binärer Entscheidungsbaum



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

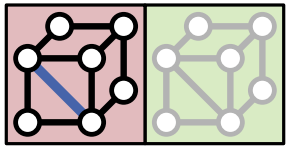
Algorithmus für VERTEX COVER

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 $w \in V'$.



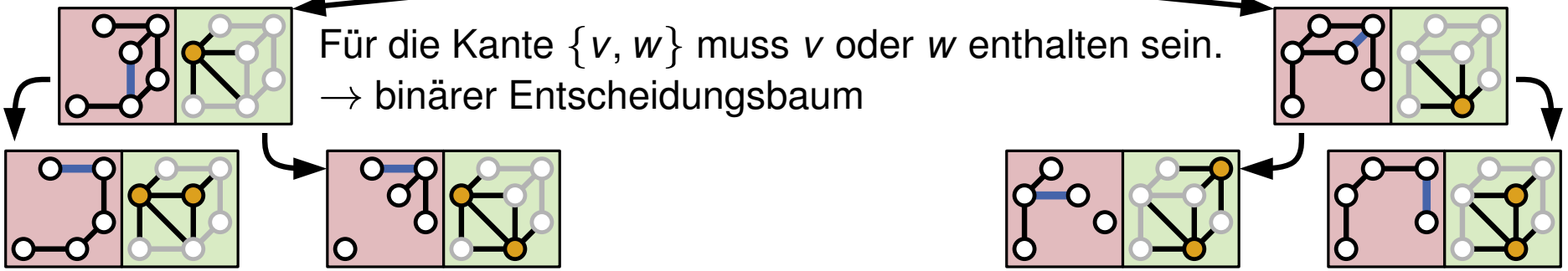
noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

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



Gibt es ein VERTEX COVER mit maximal 3 Knoten?

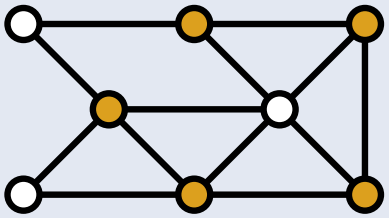
Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
→ binärer Entscheidungsbaum



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

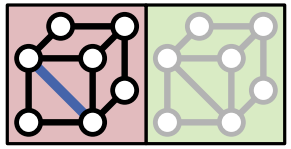
Algorithmus für VERTEX COVER

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 $w \in V'$.



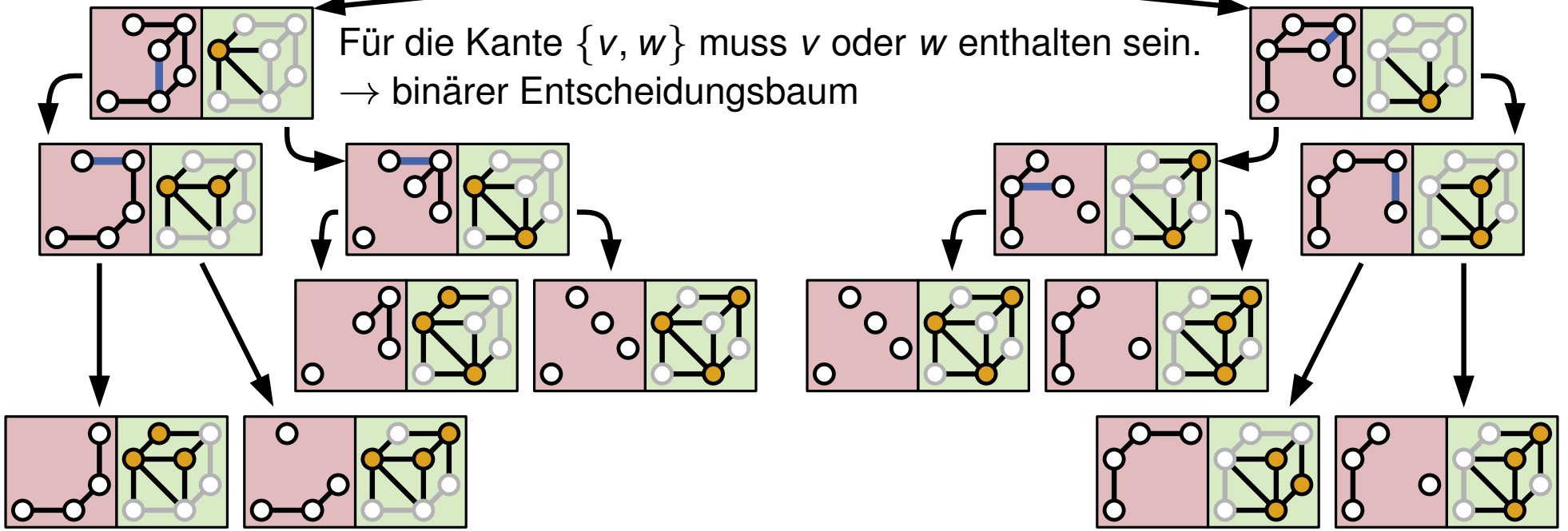
noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

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



Gibt es ein VERTEX COVER mit maximal 3 Knoten?

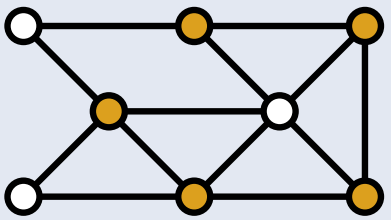
Für die Kante $\{v, w\}$ muss v oder w enthalten sein.
→ binärer Entscheidungsbaum



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

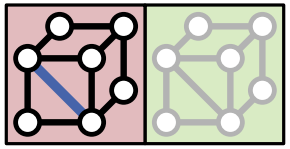
Algorithmus für VERTEX COVER

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 $w \in V'$.



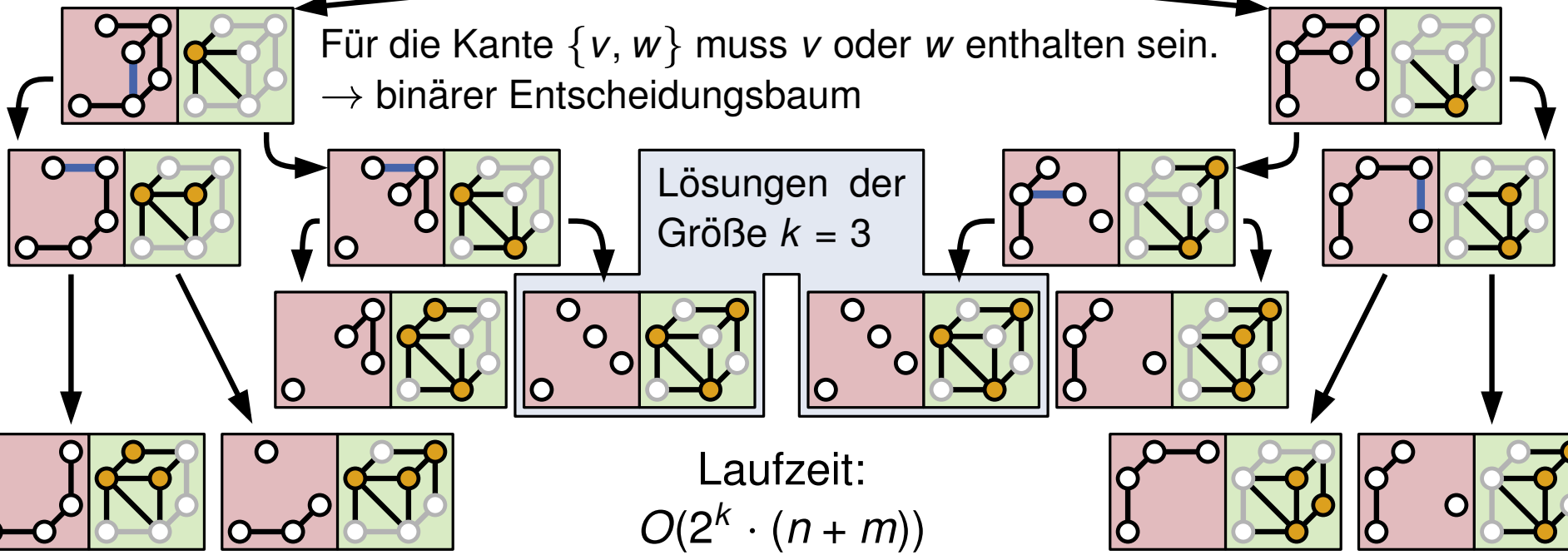
noch zu überdeckender Teilgraph — ausgewählte Knoten & überdeckte Kanten

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 Probleme, die fixed parameter tractable sind.

Bemerkung: **(Bemerkung 10.3)**

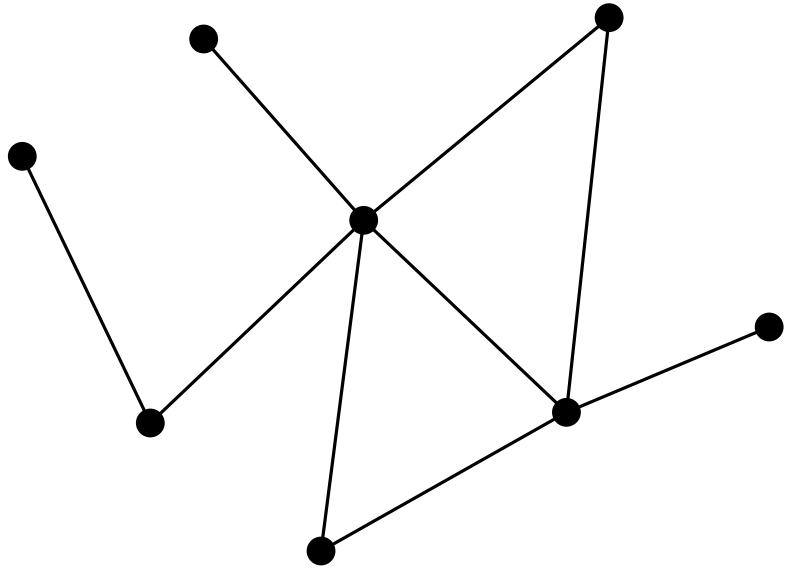
- VERTEX COVER ist in FPT.
- Es ist unbekannt ob INDEPENDENT SET oder DOMINATING SET in FPT sind. Man vermutet, dass sie nicht in FPT sind.

Independent Set

Lemma: Graph $G = (V, E)$ besitzt eine unabhängige Menge der Größe k genau dann wenn G ein Vertex Cover der Größe $n - k$ besitzt.

Versuch: Reduziere INDEPENDENT SET auf VERTEX COVER und löse damit INDEPENDENT SET:

n=8



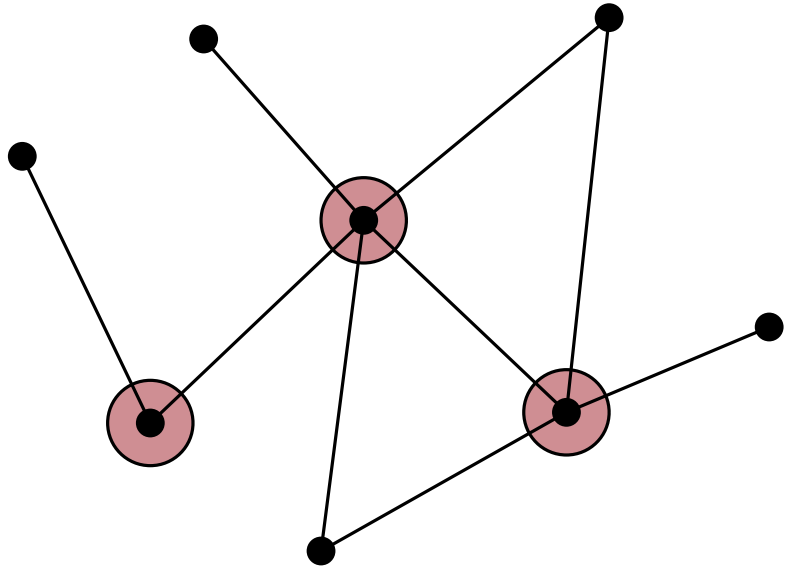
Independent Set

Lemma: Graph $G = (V, E)$ besitzt eine unabhängige Menge der Größe k genau dann wenn G ein Vertex Cover der Größe $n - k$ besitzt.

Versuch: Reduziere INDEPENDENT SET auf VERTEX COVER und löse damit INDEPENDENT SET:

n=8

Vertex Cover: 3

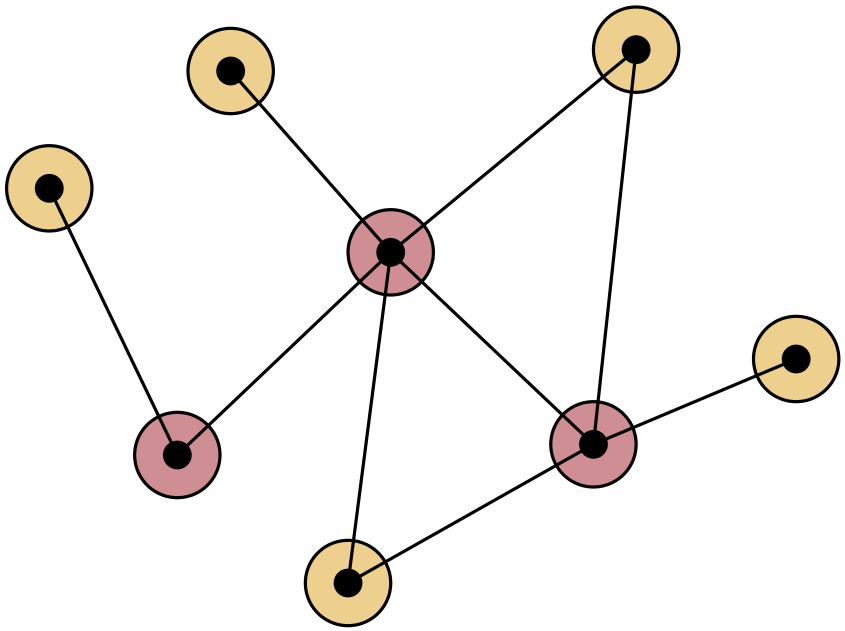


Independent Set

Lemma: Graph $G = (V, E)$ besitzt eine unabhängige Menge der Größe k genau dann wenn G ein Vertex Cover der Größe $n - k$ besitzt.

Versuch: Reduziere INDEPENDENT SET auf VERTEX COVER und löse damit INDEPENDENT SET:

n=8
Vertex Cover: 3
Independent Set: 5

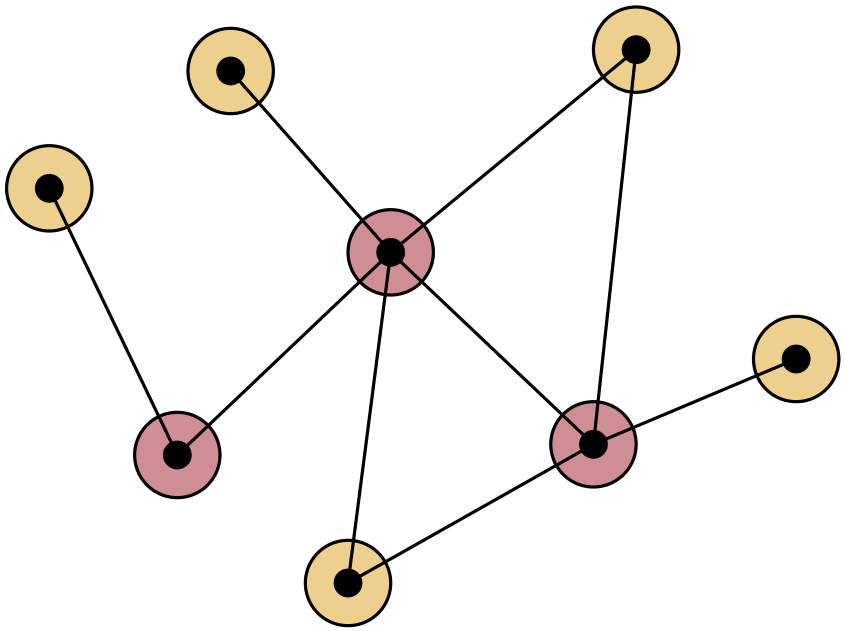


Independent Set

Lemma: Graph $G = (V, E)$ besitzt eine unabhängige Menge der Größe k genau dann wenn G ein Vertex Cover der Größe $n - k$ besitzt.

Versuch: Reduziere INDEPENDENT SET auf VERTEX COVER und löse damit INDEPENDENT SET:

n=8
Vertex Cover: 3
Independent Set: 5



Frage: Gibt es unabhängige Menge der Größe k in einem Graph G?
Gleich bedeutend mit: Gibt es Vertex Cover der Größe $n - k$ in G?

Vorgestellter Algorithmus bestimmt Vertex Cover der Größe $n - k$ in Laufzeit: $O(2^{n-k}(n + m))$, was nicht FPT ist.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Etwas formaler...

Definition: Parametrisierte Reduktion **(Definition 10.4)**

Seien Π und Π' parametrisierte Probleme. Eine *parametrisierte Reduktion* von Π auf Π' besteht aus einer Funktion f, die einer Instanz $\mathcal{I}$ mit Parameter k von Π eine Instanz $\mathcal{I}'$ von Π' zuordnet, sowie Funktionen $\mathcal{C}', \mathcal{C}'' : \mathbb{N} \to \mathbb{N}$, sodass gilt:

1. $\mathcal{I}' = f(\mathcal{I}, k)$ kann in $O(\mathcal{C}''(k) \cdot p(n))$ berechnet werden (n ist Eingabegröße von $\mathcal{I}$ und p ein Polynom).
2. $(\mathcal{I}, k)$ ist Ja-Instanz von Π genau dann wenn $(\mathcal{I}', \mathcal{C}'(k))$ ist Ja-Instanz von Π'.

Etwas formaler...

Definition: Parametrisierte Reduktion **(Definition 10.4)**

Seien Π und Π' parametrisierte Probleme. Eine *parametrisierte Reduktion* von Π auf Π' besteht aus einer Funktion f, die einer Instanz $\mathcal{I}$ mit Parameter k von Π eine Instanz $\mathcal{I}'$ von Π' zuordnet, sowie Funktionen $\mathcal{C}', \mathcal{C}'' : \mathbb{N} \to \mathbb{N}$, sodass gilt:

1. $\mathcal{I}' = f(\mathcal{I}, k)$ kann in $O(\mathcal{C}''(k) \cdot p(n))$ berechnet werden (n ist Eingabegröße von $\mathcal{I}$ und p ein Polynom).
2. $(\mathcal{I}, k)$ ist Ja-Instanz von Π genau dann wenn $(\mathcal{I}', \mathcal{C}'(k))$ ist Ja-Instanz von Π'.

Bemerkung: Vergleich zur polynomiellen Reduktion

- Ist Π polynomiell auf Π' reduzierbar, so folgt aus $\Pi' \in \mathcal{P}$ auch $\Pi \in \mathcal{P}$.
  $\to$ Π ist unter Vernachlässigung polynomieller Laufzeit nicht schwerer als Π'.
- Ist Π parametrisiert auf Π' reduzierbar, so folgt aus $\Pi' \in$ FPT auch $\Pi \in$ FPT.
  $\to$ Π ist unter Vernachläsigung von $O(\mathcal{C}(k) \cdot p(n))$ Laufzeit nicht schwerer als Π'.

Etwas formaler...

Definition: Parametrisierte Reduktion **(Definition 10.4)**

Seien Π und Π' parametrisierte Probleme. Eine *parametrisierte Reduktion* von Π auf Π' besteht aus einer Funktion f, die einer Instanz $\mathcal{I}$ mit Parameter k von Π eine Instanz $\mathcal{I}'$ von Π' zuordnet, sowie Funktionen $\mathcal{C}', \mathcal{C}'' : \mathbb{N} \to \mathbb{N}$, sodass gilt:

1. $\mathcal{I}' = f(\mathcal{I}, k)$ kann in $O(\mathcal{C}''(k) \cdot p(n))$ berechnet werden (n ist Eingabegröße von $\mathcal{I}$ und p ein Polynom).
2. $(\mathcal{I}, k)$ ist Ja-Instanz von Π genau dann wenn $(\mathcal{I}', \mathcal{C}'(k))$ ist Ja-Instanz von Π'.

Reduktion von INDEPENDENT SET auf VERTEX COVER ist keine parametrisierte Reduktion:

1. ist erfüllt:

$f(G = (V, E), k) = (G = (V, E), n - k)$ Kann in $O(1)$ berechnet werden.

2. ist nicht erfüllt:

$(G = (V, E), k)$ ist Ja-Instanz für INDEPENDENT SET genau dann, wenn $(G = (V, E), n - k)$ ist Ja-Instanz für VERTEX COVER:

Term $n - k$ ist nicht nur von k abhängig.

Etwas formaler...

Definition: Parametrisierte Reduktion **(Definition 10.4)**

Seien Π und Π' parametrisierte Probleme. Eine *parametrisierte Reduktion* von Π auf Π' besteht aus einer Funktion f, die einer Instanz $\mathcal{I}$ mit Parameter k von Π eine Instanz $\mathcal{I}'$ von Π' zuordnet, sowie Funktionen $\mathcal{C}', \mathcal{C}'' : \mathbb{N} \to \mathbb{N}$, sodass gilt:

1. $\mathcal{I}' = f(\mathcal{I}, k)$ kann in $O(\mathcal{C}''(k) \cdot p(n))$ berechnet werden (n ist Eingabegröße von $\mathcal{I}$ und p ein Polynom).
2. $(\mathcal{I}, k)$ ist Ja-Instanz von Π genau dann wenn $(\mathcal{I}', \mathcal{C}'(k))$ ist Ja-Instanz von Π'.

Bemerkung: Hierarchie von Komplexitätsklassen **(Bemerkung 10.5)**

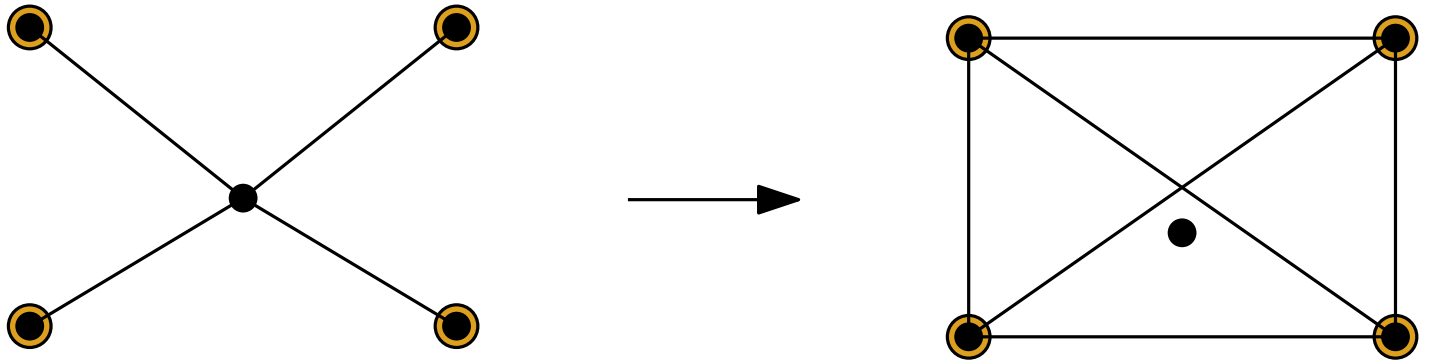
- Es gibt eine Hierarchie von Komplexitätsklassen $W[t]$, die vermutlich echt ist und FPT enthält: $\text{FPT} \subseteq W[1] \subseteq W[2] \ldots$.
- Mithilfe der parametrisierten Reduktion lässt sich ein Vollständigkeitsbegriff definieren, um die schweren Probleme in $W[t]$ zu identifizieren.
- INDEPENDENT SET ist $W[1]$-vollständig.

Clique

Betrachte Reduktion von CLIQUE auf INDEPENDENT SET:

Lemma: Graph $G = (V, E)$ besitzt eine unabhängige Menge der Größe k genau dann wenn $\overline{G}$ eine Clique der Größe k besitzt.
Dabei ist $\overline{G}$ der Komplementärgraph von G: $\overline{G} = (V, \overline{E})$ mit $\overline{E} = \{\{u, v\} \in 2^V \mid \{u, v\} \notin E\}$

Beispiel:



Independent Set der Größe 4

Clique der Größe 4

Offensichtlich: Parametrisierte Reduktion

Organisatorisches

Fragestunde und Wiederholung am 07.02.13:

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