Karlsruher Institut für Technologie

Algorithmen II
Vorlesung am 14.01.2014

Approximierende Algorithmen: Multiprozessor-Scheduling, Bin Packing

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

Grundlagen

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Approximationsalgorithmen

Ziel: Entwerfe polynomielle Algorithmen für $\mathcal{NP}$-schwere Probleme, die nicht die optimale, aber immer eine beweisbar gute Lösung finden.

Definition: Approximationsalgorithmus
Sei Π ein Minimierungsproblem, sowie $\mathcal{A}$ ein polynomieller Algorithmus, der für jede Instanz I von Π eine Lösung mit Wert $\mathcal{A}(I)$ liefert, sodass $\frac{\mathcal{A}(I)}{\text{OPT}(I)} \leq K$ gilt, wobei K eine Konstante und $\text{OPT}(I)$ der Wert einer optimalen Lösung ist.
Dann heißt $\mathcal{A}$ *Approximationsalgorithmus mit relativer Gütegarantie*.

Definition: Gütegarantie
Sei $\mathcal{R}_{\mathcal{A}}(I) = \frac{\mathcal{A}(I)}{\text{OPT}(I)}$. Dann ist die *Approximationsgüte* $\mathcal{R}_{\mathcal{A}}$ definiert als

$$\mathcal{R}_{\mathcal{A}} = \inf\{r \geq 1 \mid \mathcal{R}_{\mathcal{A}}(I) \leq r \text{ für alle Instanzen } I \text{ von } \Pi\}.$$

$\mathcal{A}$ heißt ε-*approximierend*, falls $\mathcal{R}_{\mathcal{A}} \leq 1 + \varepsilon$.

Maximierungsprobleme:

- Für ein Maximierungsproblem betrachte $\frac{\text{OPT}(I)}{\mathcal{A}(I)}$ statt $\frac{\mathcal{A}(I)}{\text{OPT}(I)}$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Approximationsschemata

Definition: PAS **(Definition 7.10)**

Ein *(polynomielles) Approximationsschema (PAS)* für ein Optimierungsproblem Π ist eine Familie von Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$, sodass $\mathcal{A}_\varepsilon$ ein ε-approximierender Algorithmus ist.

(häufig auch PTAS genannt)

Beachte: Die Laufzeit jedes Algorithmus $\mathcal{A}_\varepsilon$ ist nur polynomiell in der Eingabegröße und kann stark von ε abhängen.

Definition: FPAS **(Definition 7.10)**

Ein Approximationsschema $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ heißt *vollpolynomiell (FPAS)*, falls die Laufzeit jedes $\mathcal{A}_\varepsilon$ zusätzlich polynomiell in $\frac{1}{\varepsilon}$ ist.

(häufig auch FPTAS genannt)

Beispiel:

- Laufzeit $O\left(n^2 + n^{\frac{1}{\varepsilon}}\right)$ für $A_\varepsilon : \rightarrow$ PAS aber kein FPAS
- Laufzeit $O\left(\sqrt{n} \cdot 3^{\frac{1}{\varepsilon}}\right)$ für $A_\varepsilon : \rightarrow$ PAS aber kein FPAS
- Laufzeit $O\left(n^4 \cdot \left(\frac{1}{\varepsilon}\right)^2\right)$ für $A_\varepsilon : \rightarrow$ FPAS

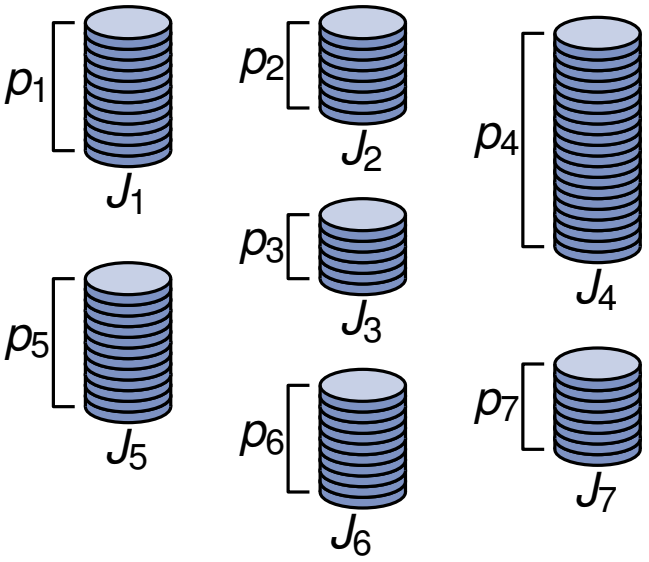
Karlsruher Institut für Technologie

Multiprozessor-Scheduling

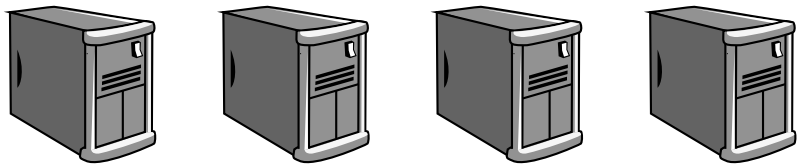
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Multiprozessor-Scheduling – Definition

n Jobs $J_1, \ldots, J_n$ (in bel. Reihenfolge)
mit Bearbeitungsdauer $p_1, \ldots, p_n$.



$m < n$



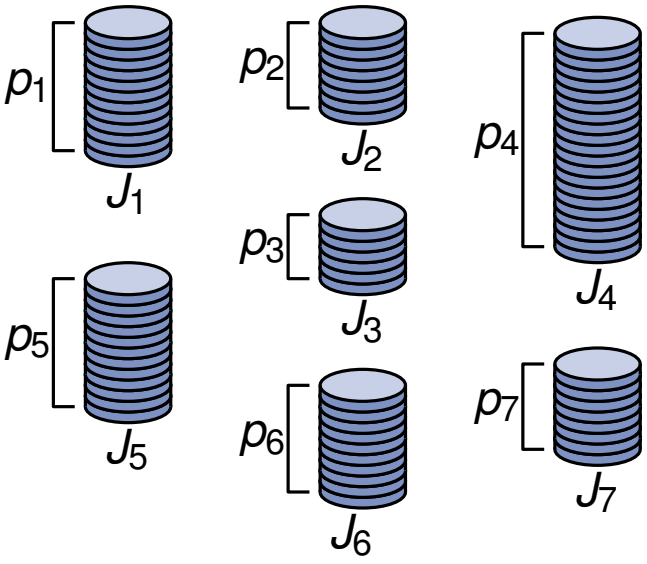
m identische Maschinen

Problem: MULTIPROZESSOR SCHEDULING
Finde eine Zuweisung der Jobs auf die Maschinen, sodass der Zeitpunkt, zu dem alle Jobs abgearbeitet sind, minimal ist.
Dieser Zeitpunkt heißt auch MAKESPAN einer Zuweisung.

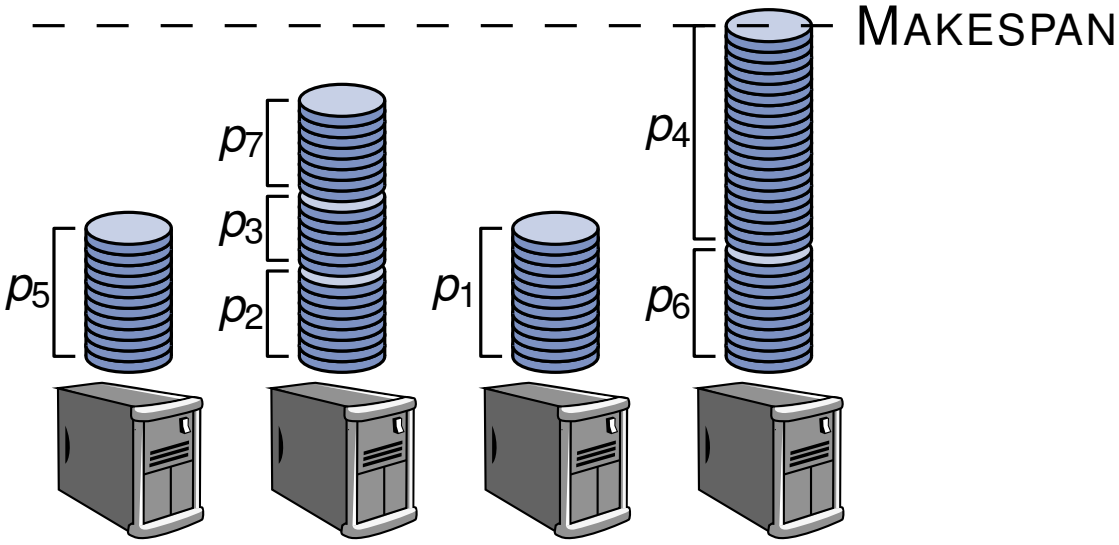
MULTIPROZESSOR SCHEDULING ist $\mathcal{NP}$-schwer.

Multiprozessor-Scheduling – Definition

n Jobs $J_1, \ldots, J_n$ (in bel. Reihenfolge)
mit Bearbeitungsdauer $p_1, \ldots, p_n$.



$m < n$



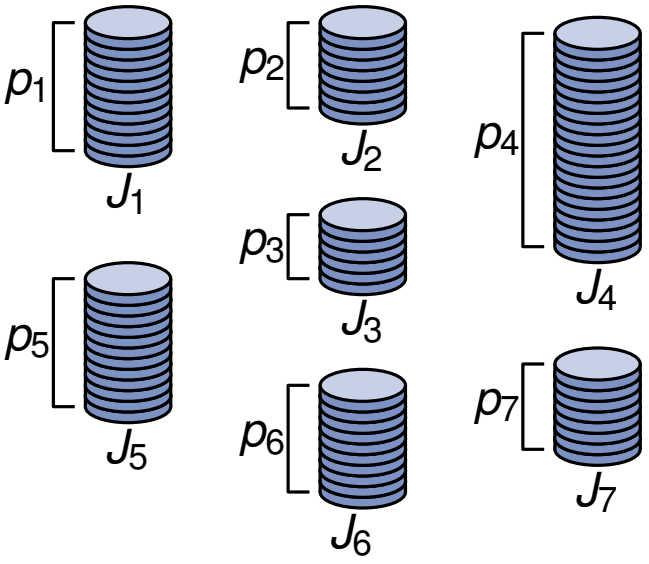
m identische Maschinen

Problem: MULTIPROZESSOR SCHEDULING
Finde eine Zuweisung der Jobs auf die Maschinen, sodass der Zeitpunkt, zu dem alle Jobs abgearbeitet sind, minimal ist.
Dieser Zeitpunkt heißt auch MAKESPAN einer Zuweisung.

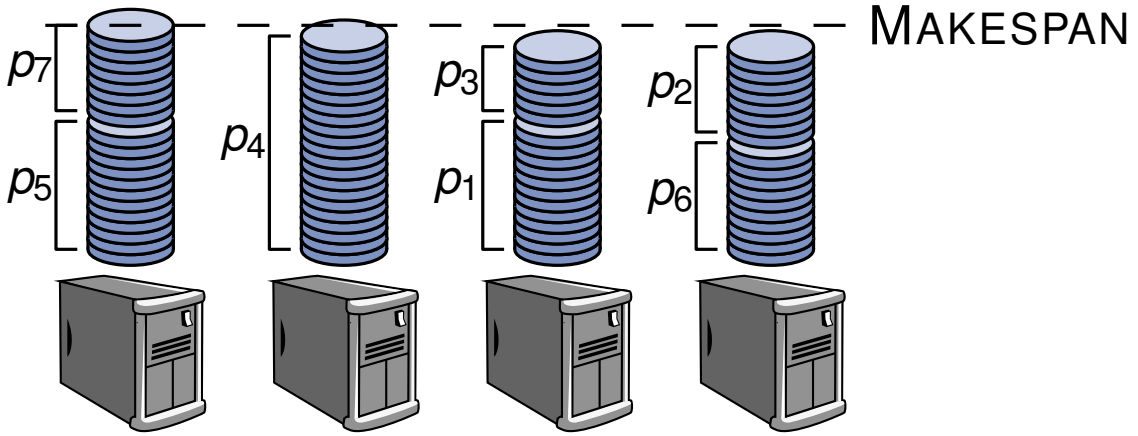
MULTIPROZESSOR SCHEDULING ist $\mathcal{NP}$-schwer.

Multiprozessor-Scheduling – Definition

n Jobs $J_1, \ldots, J_n$ (in bel. Reihenfolge)
mit Bearbeitungsdauer $p_1, \ldots, p_n$.



$m < n$



m identische Maschinen

Problem: MULTIPROZESSOR SCHEDULING
Finde eine Zuweisung der Jobs auf die Maschinen, sodass der Zeitpunkt, zu dem alle Jobs abgearbeitet sind, minimal ist.
Dieser Zeitpunkt heißt auch MAKESPAN einer Zuweisung.

MULTIPROZESSOR SCHEDULING ist $\mathcal{NP}$-schwer.

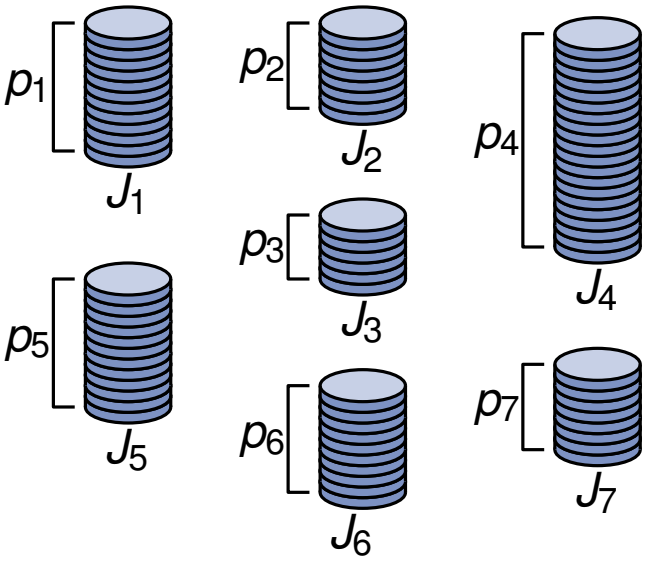
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

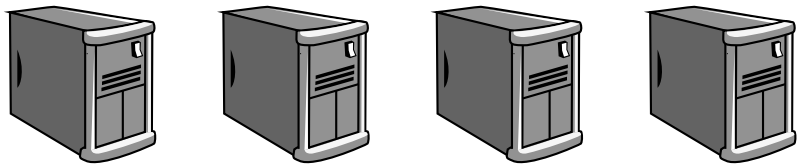
Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

Lege die ersten m Jobs auf die m Maschinen.

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen Jobs zu.





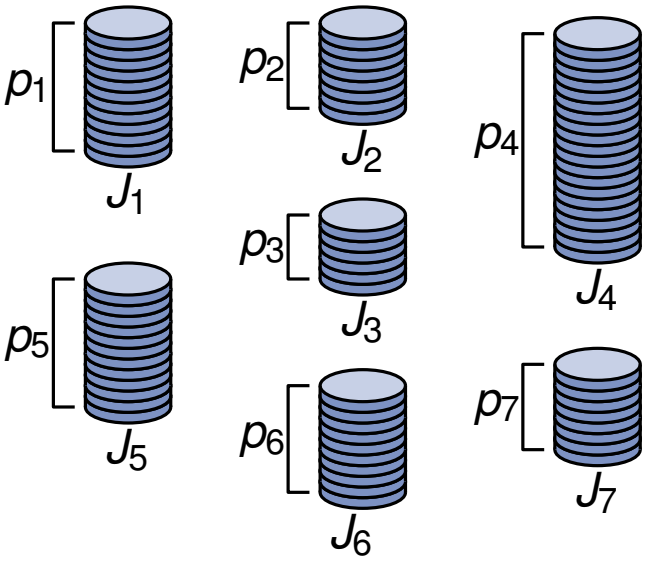
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

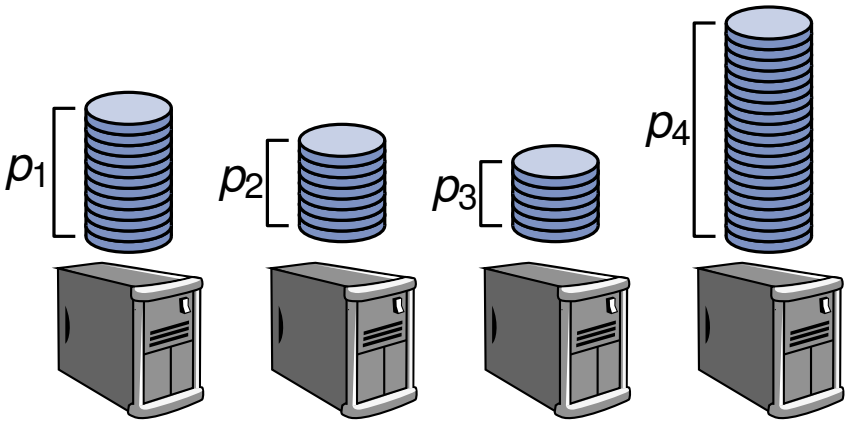
Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING$(J_1, \ldots, J_n, m)$

Lege die ersten m Jobs auf die m Maschinen.

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen Jobs zu.





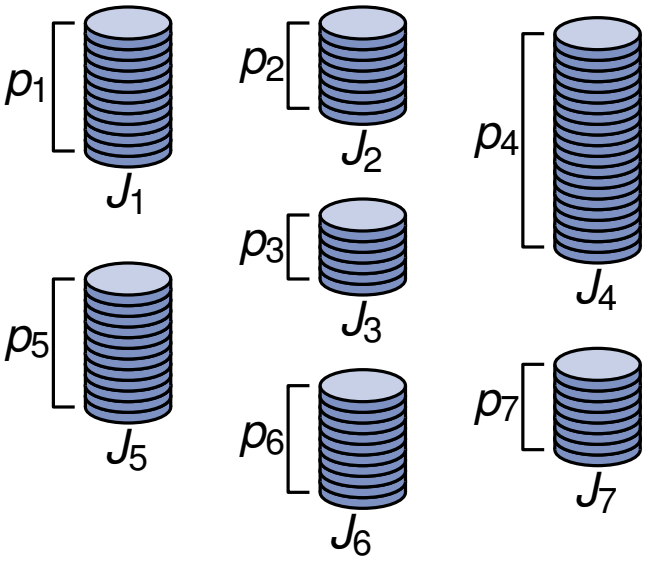
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

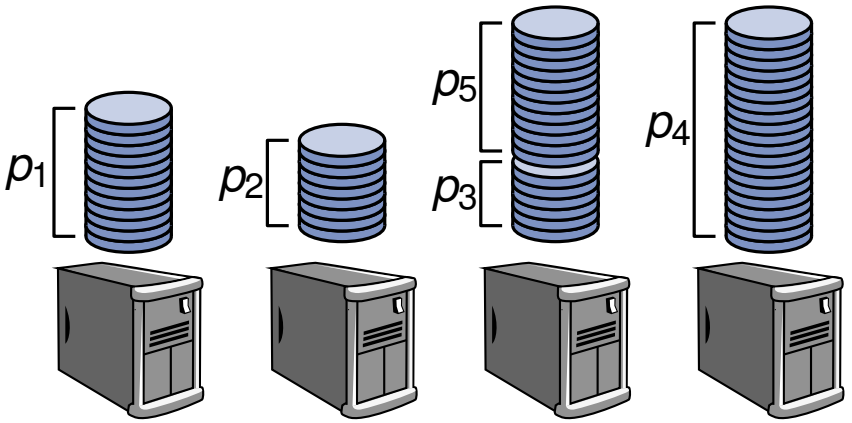
Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING$(J_1, \ldots, J_n, m)$

Lege die ersten m Jobs auf die m Maschinen.

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen Jobs zu.





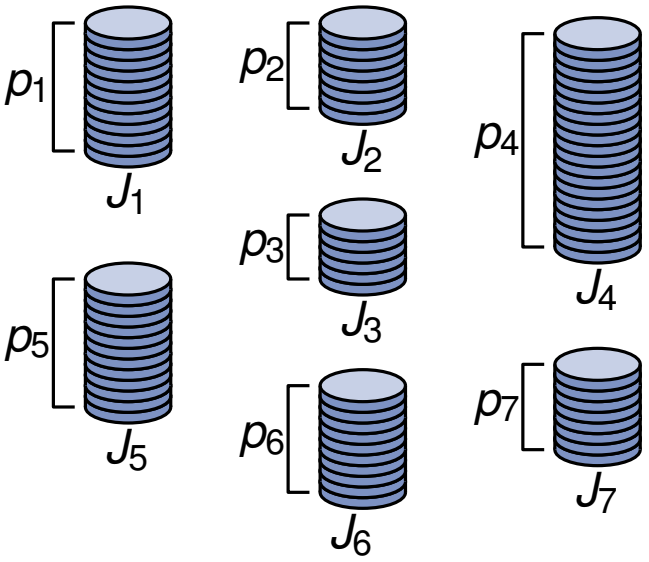
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

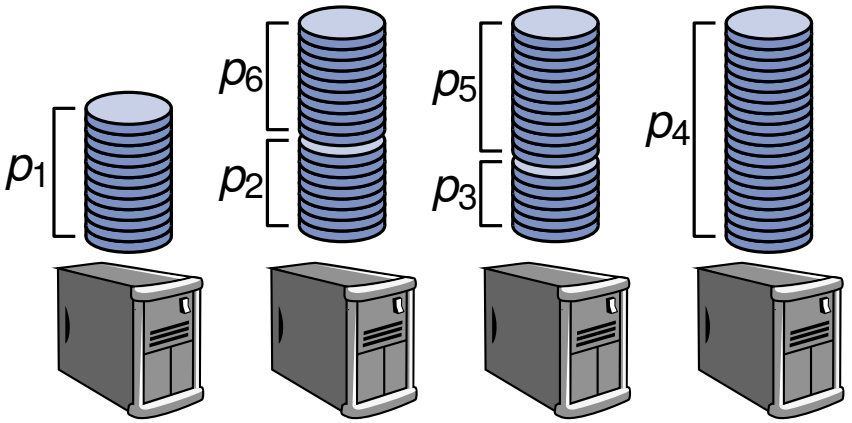
Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

Lege die ersten m Jobs auf die m Maschinen.

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen Jobs zu.



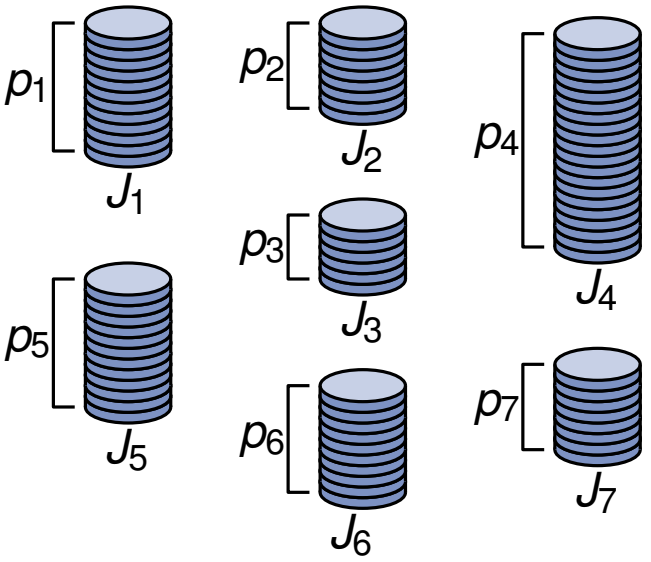


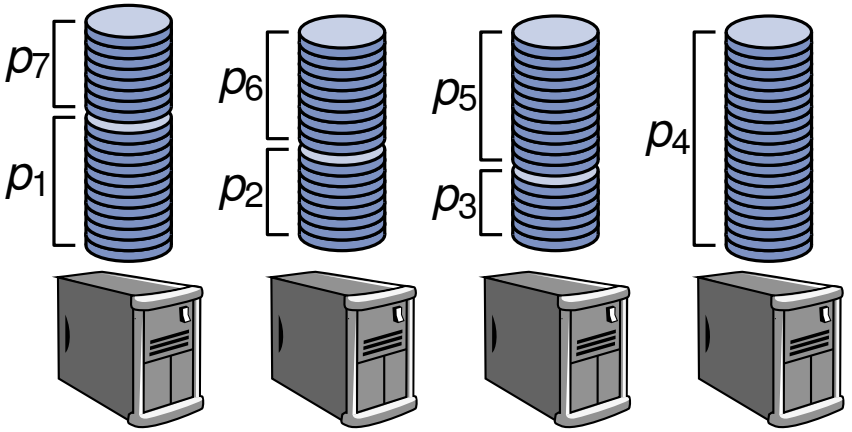
Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

Lege die ersten m Jobs auf die m Maschinen.

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen Jobs zu.





LIST SCHEDULING hat offensichtlich Laufzeit $O(n)$.

Satz: Approximation **(Satz 7.12)**

Sei $\mathcal{A}$ der Algorithmus LIST SCHEDULING. Dann gilt $\mathcal{R}_{\mathcal{A}} = 2 - \frac{1}{m}$.

Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

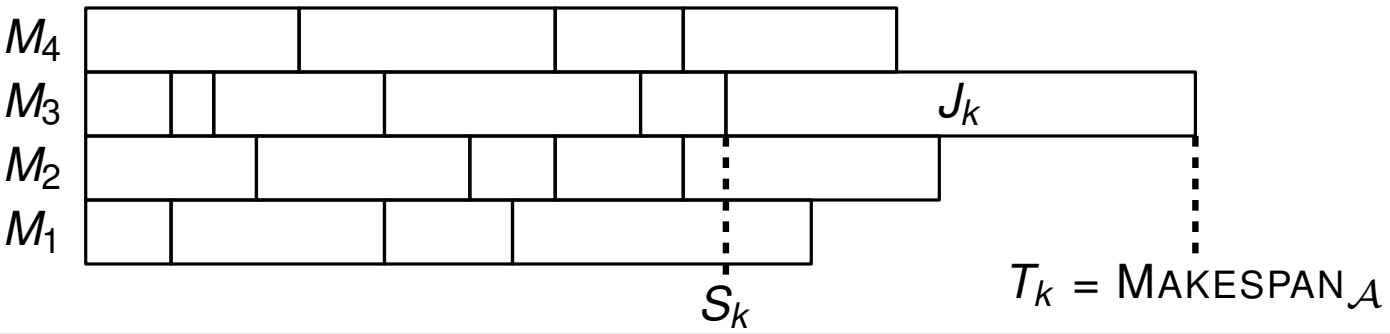
Lege die ersten m Jobs auf die m Maschinen

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen $n - m$ Jobs zu

Satz: Approximation **(Satz 7.12)**

Sei $\mathcal{A}$ der Algorithmus LIST SCHEDULING. Dann gilt $\mathcal{R}_{\mathcal{A}} = 2 - \frac{1}{m}$.

Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$



Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

Lege die ersten m Jobs auf die m Maschinen

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen $n - m$ Jobs zu

Satz: Approximation **(Satz 7.12)**

Sei $\mathcal{A}$ der Algorithmus LIST SCHEDULING. Dann gilt $\mathcal{R}_{\mathcal{A}} = 2 - \frac{1}{m}$.

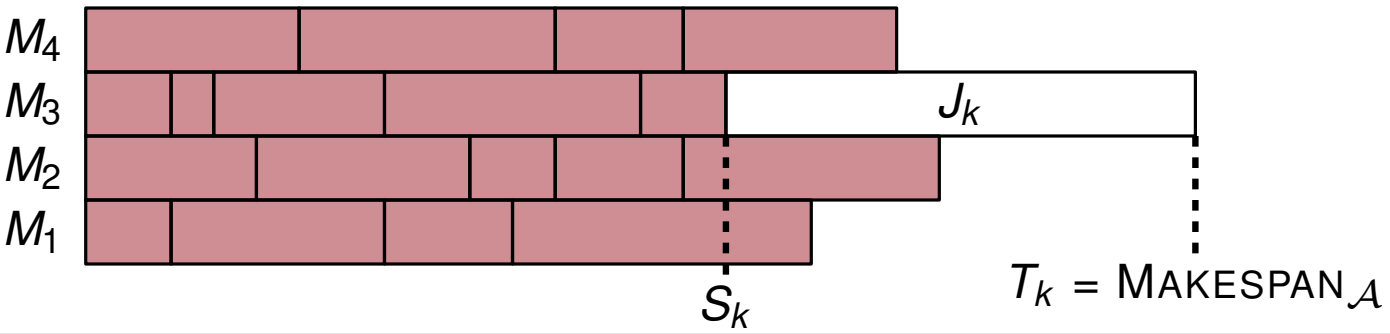
Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

- Keine Maschine ist zu einem Zeitpunkt vor S_k untätig.

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$

$\sum_{i \neq k} p_i$: Gewicht aller Jobs außer J_k

$\frac{1}{m}$: gleichmäßig auf m Maschinen verteilt



Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

Lege die ersten m Jobs auf die m Maschinen

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen $n - m$ Jobs zu

Satz: Approximation **(Satz 7.12)**

Sei $\mathcal{A}$ der Algorithmus LIST SCHEDULING. Dann gilt $\mathcal{R}_{\mathcal{A}} = 2 - \frac{1}{m}$.

Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

- Keine Maschine ist zu einem Zeitpunkt vor S_k untätig.

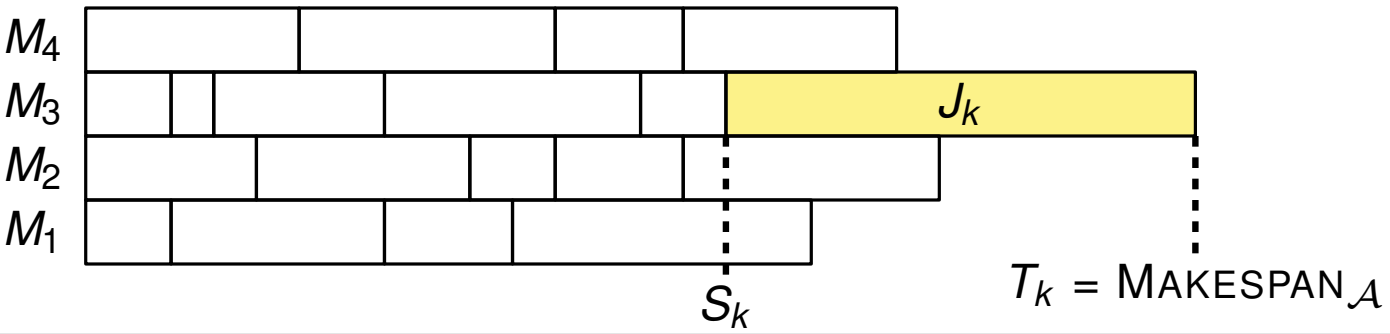
$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$

$\sum_{i \neq k} p_i$: Gewicht aller Jobs außer J_k

$\frac{1}{m}$: gleichmäßig auf m Maschinen verteilt

- Für den optimalen MAKESPAN T_{OPT} gilt:

$T_{\text{OPT}} \geq p_k$ denn J_k muss ausgeführt werden



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

Lege die ersten m Jobs auf die m Maschinen

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen $n - m$ Jobs zu

Satz: Approximation **(Satz 7.12)**

Sei $\mathcal{A}$ der Algorithmus LIST SCHEDULING. Dann gilt $\mathcal{R}_{\mathcal{A}} = 2 - \frac{1}{m}$.

Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

- Keine Maschine ist zu einem Zeitpunkt vor S_k untätig.

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$

$\sum_{i \neq k} p_i$: Gewicht aller Jobs außer J_k

$\frac{1}{m}$: gleichmäßig auf m Maschinen verteilt

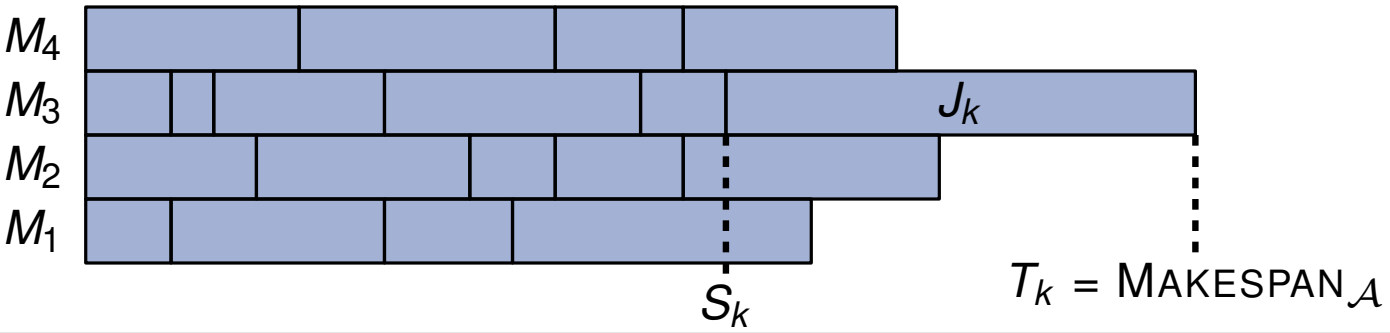
- Für den optimalen MAKESPAN T_{OPT} gilt:

$T_{\text{OPT}} \geq p_k$ denn J_k muss ausgeführt werden

$$T_{\text{OPT}} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$$

$\sum_{i=1}^{n} p_i$: Gewicht aller Jobs

$\frac{1}{m}$: gleichmäßig auf m Maschinen verteilt



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

Lege die ersten m Jobs auf die m Maschinen

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen $n - m$ Jobs zu

Satz: Approximation **(Satz 7.12)**

Sei $\mathcal{A}$ der Algorithmus LIST SCHEDULING. Dann gilt $\mathcal{R}_{\mathcal{A}} = 2 - \frac{1}{m}$.

Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

- Keine Maschine ist zu einem Zeitpunkt vor S_k untätig.

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$

$\sum_{i \neq k} p_i$: Gewicht aller Jobs außer J_k

$\frac{1}{m}$: gleichmäßig auf m Maschinen verteilt

Es folgt:

$$T_k = S_k + p_k$$

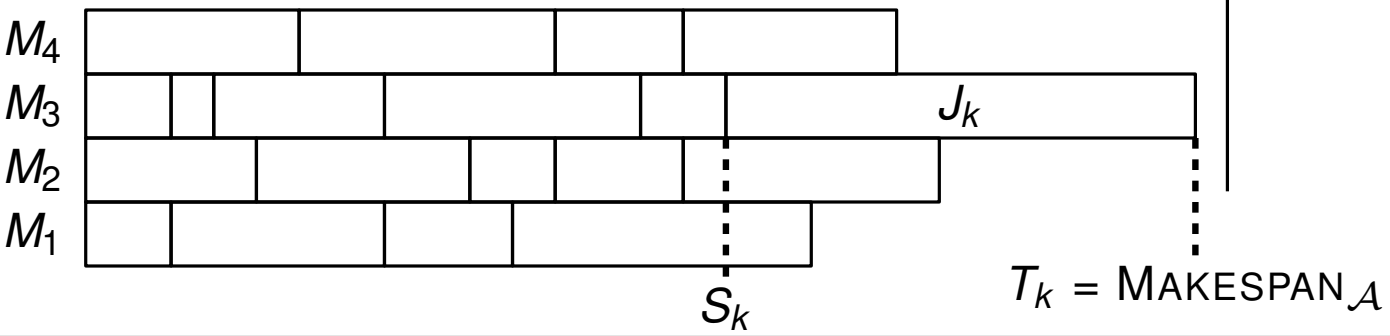
- Für den optimalen MAKESPAN T_{OPT} gilt:

$T_{\text{OPT}} \geq p_k$ denn J_k muss ausgeführt werden

$$T_{\text{OPT}} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$$

$\sum_{i=1}^{n} p_i$: Gewicht aller Jobs

$\frac{1}{m}$: gleichmäßig auf m Maschinen verteilt



Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

Lege die ersten m Jobs auf die m Maschinen

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen $n - m$ Jobs zu

Satz: Approximation **(Satz 7.12)**

Sei $\mathcal{A}$ der Algorithmus LIST SCHEDULING. Dann gilt $\mathcal{R}_{\mathcal{A}} = 2 - \frac{1}{m}$.

Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

- Keine Maschine ist zu einem Zeitpunkt vor S_k untätig.

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$

$\sum_{i \neq k} p_i$: Gewicht aller Jobs außer J_k

$\frac{1}{m}$: gleichmäßig auf m Maschinen verteilt

- Für den optimalen MAKESPAN T_{OPT} gilt:

$T_{\text{OPT}} \geq p_k$ denn J_k muss ausgeführt werden

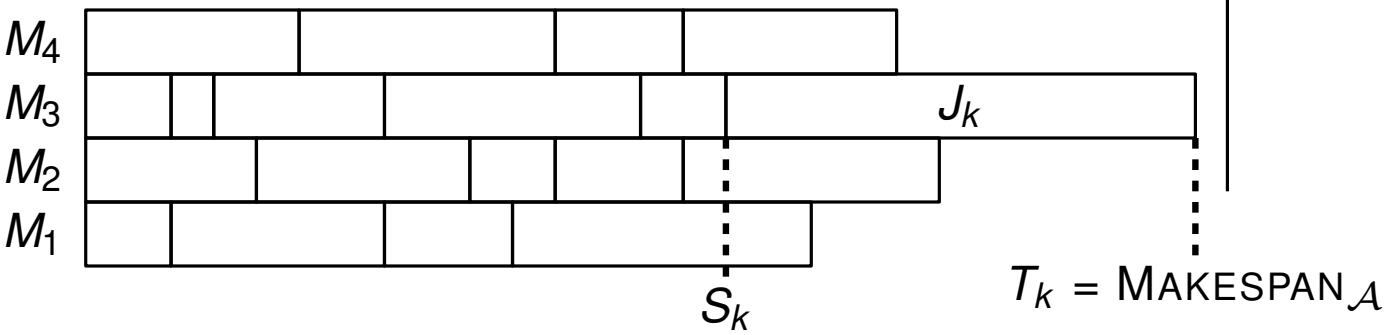
$$T_{\text{OPT}} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$$

$\sum_{i=1}^{n} p_i$: Gewicht aller Jobs

$\frac{1}{m}$: gleichmäßig auf m Maschinen verteilt

Es folgt:

$$T_k = S_k + p_k \leq \frac{1}{m} \cdot \sum_{i \neq k} p_i + p_k$$



Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)
Lege die ersten m Jobs auf die m Maschinen
Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen $n - m$ Jobs zu

Satz: Approximation **(Satz 7.12)**
Sei $\mathcal{A}$ der Algorithmus LIST SCHEDULING. Dann gilt $\mathcal{R}_{\mathcal{A}} = 2 - \frac{1}{m}$.

Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

- Keine Maschine ist zu einem Zeitpunkt vor S_k untätig.

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$ Gewicht aller Jobs außer J_k gleichmäßig auf m Maschinen verteilt

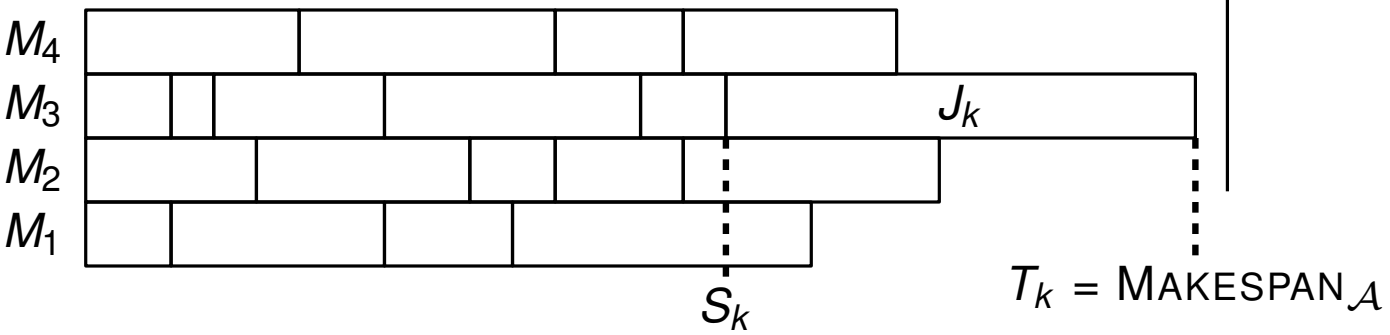
- Für den optimalen MAKESPAN T_{OPT} gilt:

$T_{\text{OPT}} \geq p_k$ denn J_k muss ausgeführt werden

$$T_{\text{OPT}} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$$ Gewicht aller Jobs gleichmäßig auf m Maschinen verteilt

Es folgt:

$$\begin{aligned} T_k &= S_k + p_k \\ &\leq \frac{1}{m} \cdot \sum_{i \neq k} p_i + p_k \\ &= \frac{1}{m} \cdot \sum_{i=1}^{m} p_i + \left(1 - \frac{1}{m}\right) \cdot p_k \end{aligned}$$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

Lege die ersten m Jobs auf die m Maschinen

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen $n - m$ Jobs zu

Satz: Approximation **(Satz 7.12)**

Sei $\mathcal{A}$ der Algorithmus LIST SCHEDULING. Dann gilt $\mathcal{R}_{\mathcal{A}} = 2 - \frac{1}{m}$.

Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

- Keine Maschine ist zu einem Zeitpunkt vor S_k untätig.

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$

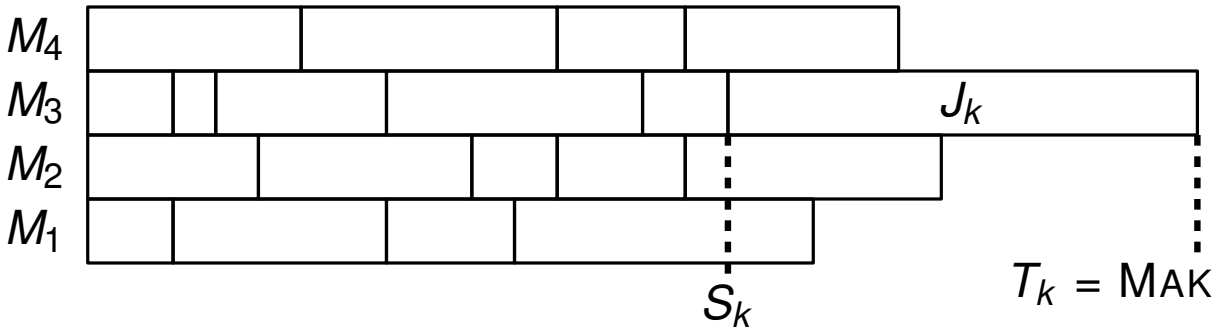
Gewicht aller Jobs außer J_k gleichmäßig auf m Maschinen verteilt

- Für den optimalen MAKESPAN T_{OPT} gilt:

$T_{\text{OPT}} \geq p_k$ denn J_k muss ausgeführt werden

$$T_{\text{OPT}} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$$

Gewicht aller Jobs gleichmäßig auf m Maschinen verteilt



Es folgt:

$$\begin{aligned} T_k &= S_k + p_k \\ &\leq \frac{1}{m} \cdot \sum_{i \neq k} p_i + p_k \\ &= \frac{1}{m} \cdot \sum_{i=1}^{m} p_i + \left(1 - \frac{1}{m}\right) \cdot p_k \\ &\leq T_{\text{OPT}} + \left(1 - \frac{1}{m}\right) \cdot T_{\text{OPT}} \end{aligned}$$

Ein einfacher Algorithmus – LIST SCHEDULING

LIST SCHEDULING($J_1, \ldots, J_n, m$)

Lege die ersten m Jobs auf die m Maschinen

Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen $n - m$ Jobs zu

Satz: Approximation **(Satz 7.12)**

Sei $\mathcal{A}$ der Algorithmus LIST SCHEDULING. Dann gilt $\mathcal{R}_{\mathcal{A}} = 2 - \frac{1}{m}$.

Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

- Keine Maschine ist zu einem Zeitpunkt vor S_k untätig.

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$

$\sum_{i \neq k} p_i$: Gewicht aller Jobs außer J_k

$\frac{1}{m}$: gleichmäßig auf m Maschinen verteilt

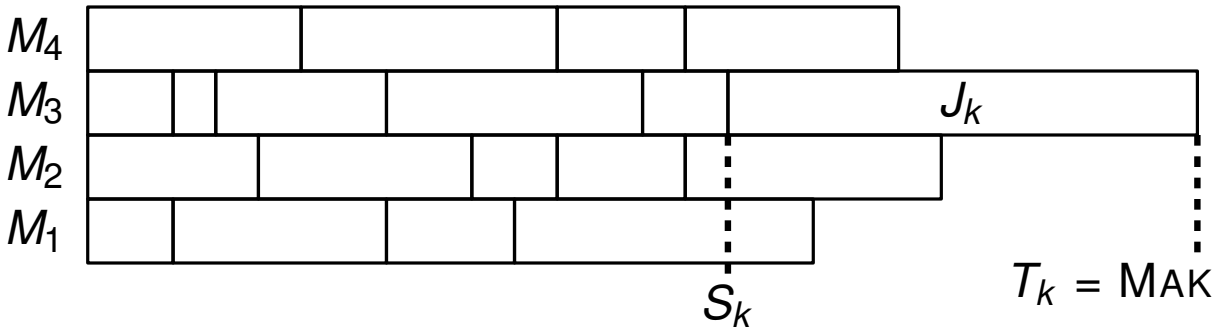
- Für den optimalen MAKESPAN T_{OPT} gilt:

$T_{\text{OPT}} \geq p_k$ denn J_k muss ausgeführt werden

$$T_{\text{OPT}} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$$

$\sum_{i=1}^{n} p_i$: Gewicht aller Jobs

$\frac{1}{m}$: gleichmäßig auf m Maschinen verteilt



Es folgt:

$$\begin{aligned} T_k &= S_k + p_k \\ &\leq \frac{1}{m} \cdot \sum_{i \neq k} p_i + p_k \\ &= \frac{1}{m} \cdot \sum_{i=1}^{m} p_i + \left(1 - \frac{1}{m}\right) \cdot p_k \\ &\leq T_{\text{OPT}} + \left(1 - \frac{1}{m}\right) \cdot T_{\text{OPT}} \\ &= \left(2 - \frac{1}{m}\right) \cdot T_{\text{OPT}} \end{aligned}$$

Ein PAS für Multiprozessor-Scheduling

Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Ein PAS für Multiprozessor-Scheduling

Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ — polynomiell für konstantes $\ell \rightarrow$ $O(m^\ell + n \log n)$

- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. — $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. — $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Ein PAS für Multiprozessor-Scheduling

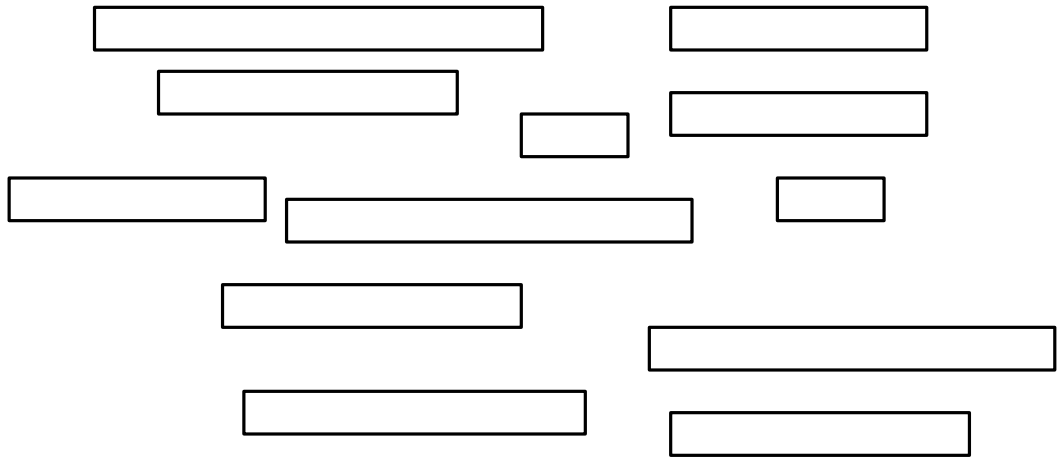
Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ — polynomiell für konstantes $\ell \rightarrow$ $O(m^\ell + n \log n)$

- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. — $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. — $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Beispiel: $\ell = 6$

Jobs:



Ein PAS für Multiprozessor-Scheduling

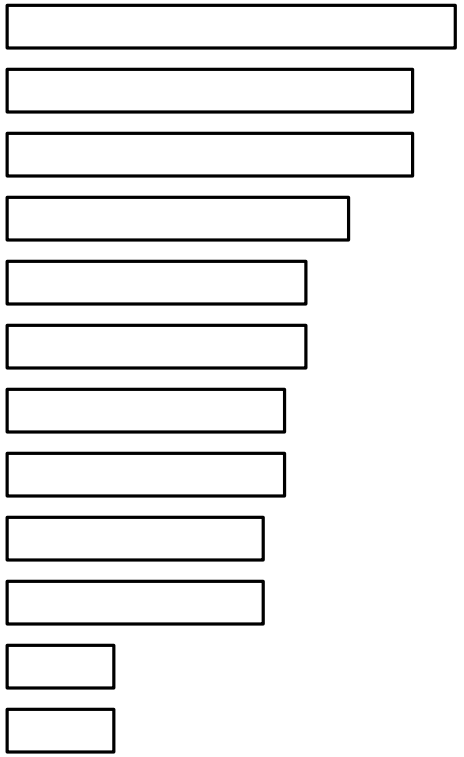
Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ — polynomiell für konstantes $\ell \rightarrow$ $O(m^\ell + n \log n)$

- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. — $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. — $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Beispiel: $\ell = 6$

sortierte Jobs:



Ein PAS für Multiprozessor-Scheduling

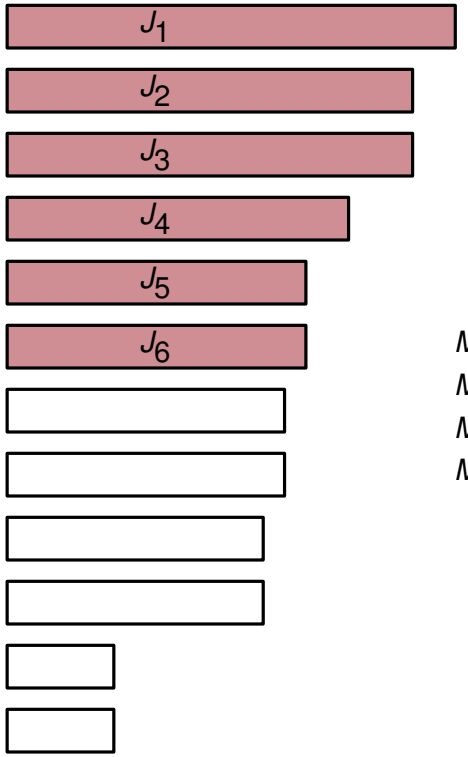
Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ — polynomiell für konstantes $\ell \rightarrow O(m^\ell + n \log n)$

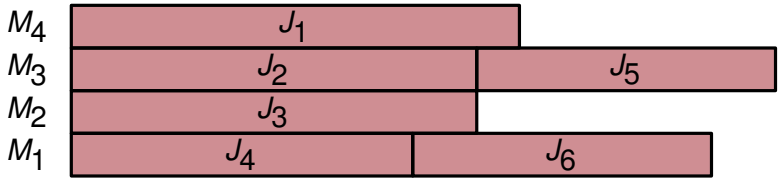
- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Beispiel: $\ell = 6$

sortierte Jobs:



- optimale Lösung für die ℓ größten Jobs



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein PAS für Multiprozessor-Scheduling

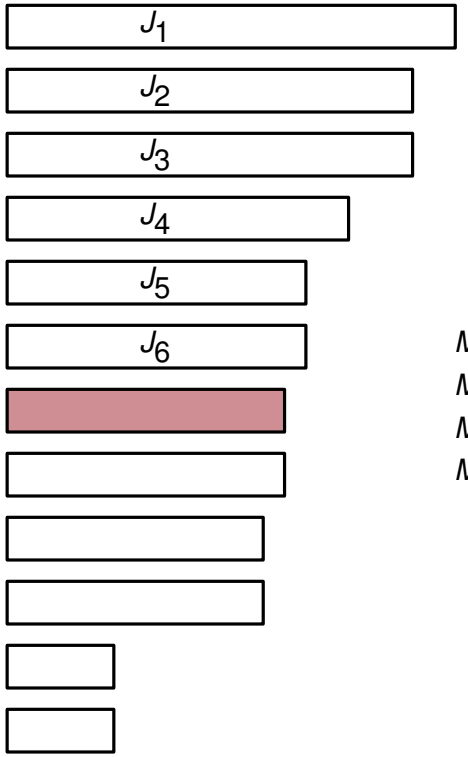
Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ — polynomiell für konstantes $\ell \rightarrow O(m^\ell + n \log n)$

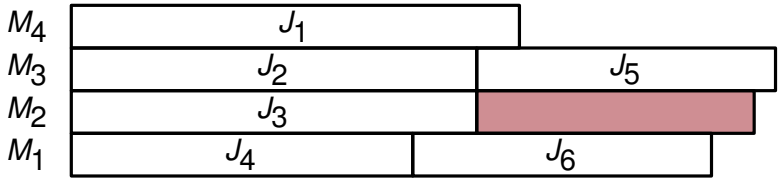
- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Beispiel: $\ell = 6$

sortierte Jobs:



- optimale Lösung für die ℓ größten Jobs
- alle Anderen mittels LIST SCHEDULING



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein PAS für Multiprozessor-Scheduling

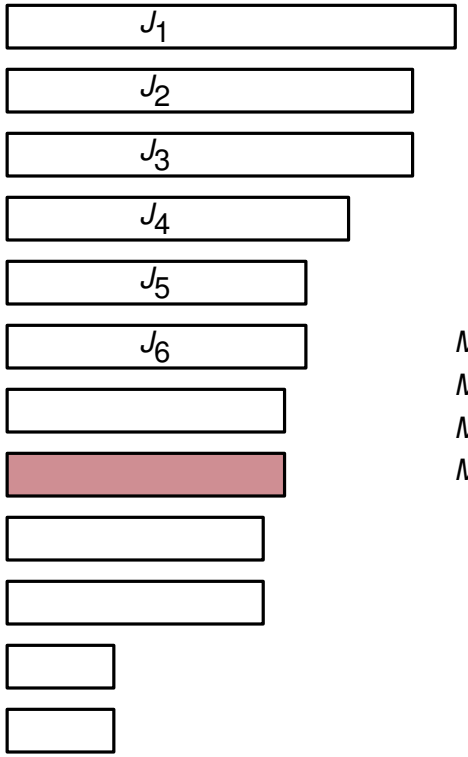
Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ — polynomiell für konstantes $\ell \rightarrow O(m^\ell + n \log n)$

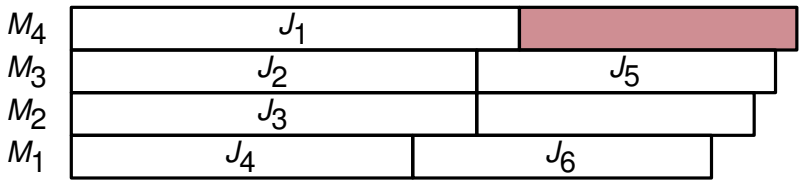
- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Beispiel: $\ell = 6$

sortierte Jobs:



- optimale Lösung für die ℓ größten Jobs
- alle Anderen mittels LIST SCHEDULING



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein PAS für Multiprozessor-Scheduling

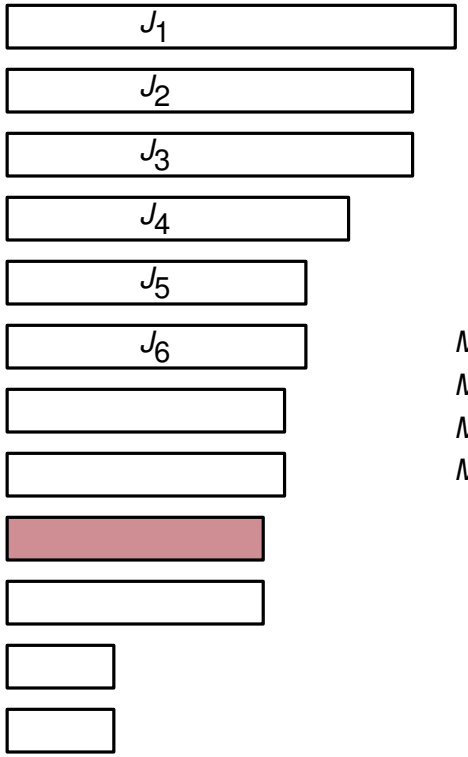
Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ — polynomiell für konstantes $\ell \rightarrow$ $O(m^\ell + n \log n)$

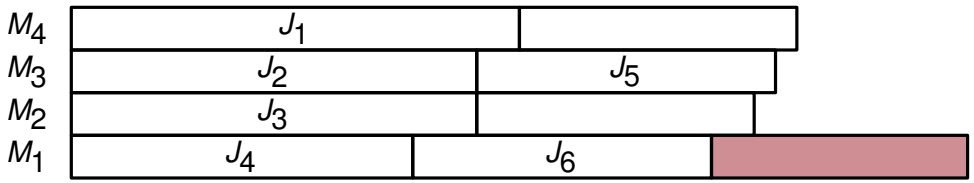
- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Beispiel: $\ell = 6$

sortierte Jobs:



- optimale Lösung für die ℓ größten Jobs
- alle Anderen mittels LIST SCHEDULING



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein PAS für Multiprozessor-Scheduling

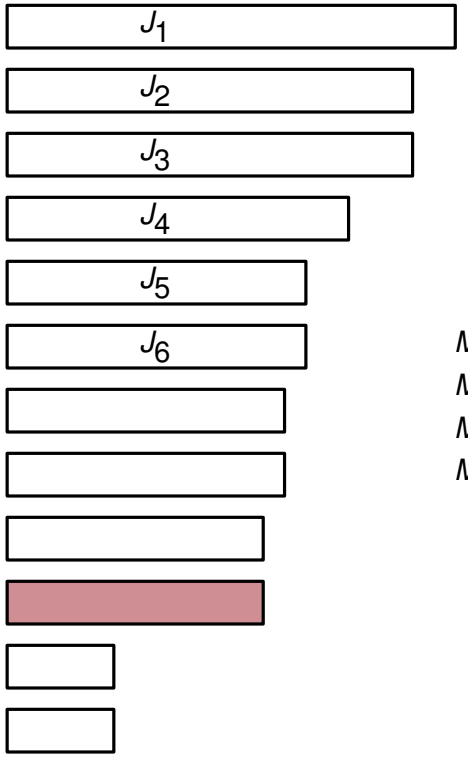
Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ — polynomiell für konstantes $\ell \rightarrow O(m^\ell + n \log n)$

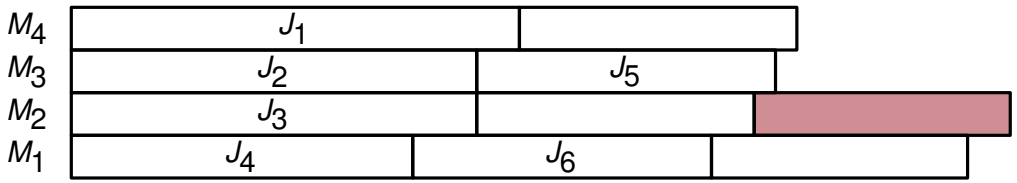
- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Beispiel: $\ell = 6$

sortierte Jobs:



- optimale Lösung für die ℓ größten Jobs
- alle Anderen mittels LIST SCHEDULING



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein PAS für Multiprozessor-Scheduling

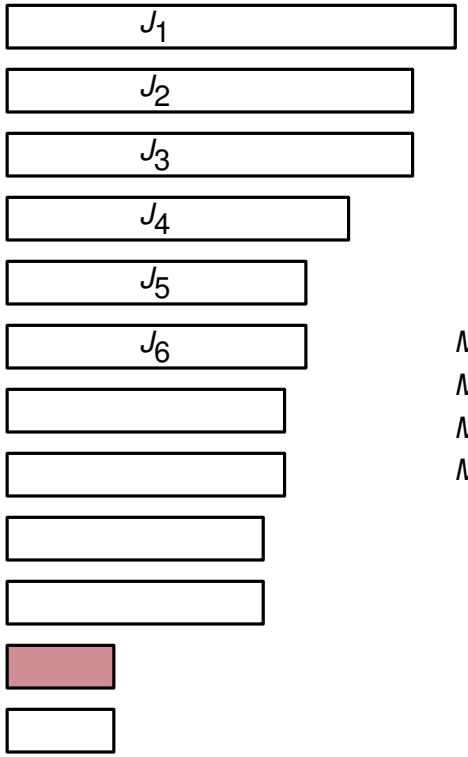
Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ — polynomiell für konstantes $\ell \rightarrow O(m^\ell + n \log n)$

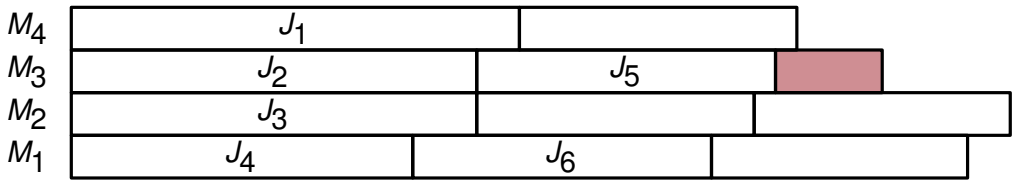
- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Beispiel: $\ell = 6$

sortierte Jobs:



- optimale Lösung für die ℓ größten Jobs
- alle Anderen mittels LIST SCHEDULING



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein PAS für Multiprozessor-Scheduling

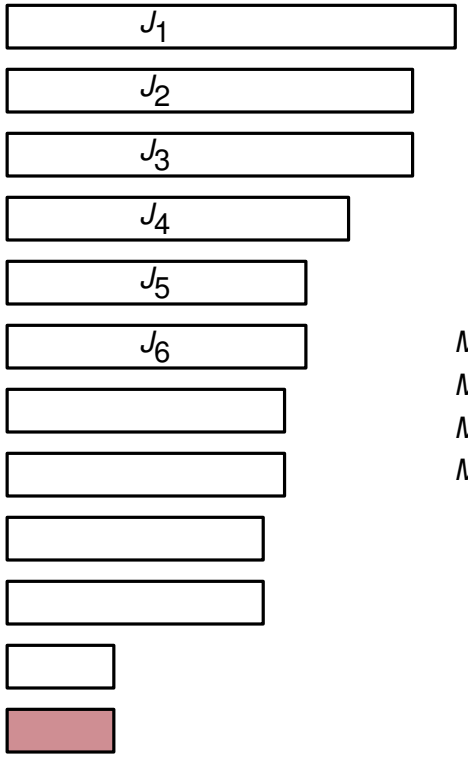
Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ — polynomiell für konstantes $\ell \rightarrow$ $O(m^\ell + n \log n)$

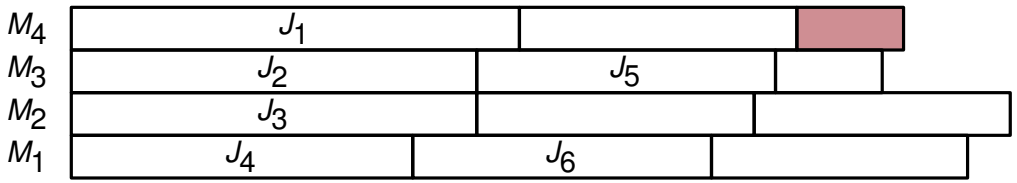
- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. — $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. — $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Beispiel: $\ell = 6$

sortierte Jobs:



- optimale Lösung für die ℓ größten Jobs
- alle Anderen mittels LIST SCHEDULING



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein PAS für Multiprozessor-Scheduling

Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ polynomiell für konstantes $\ell \rightarrow$ $O(m^\ell + n \log n)$

- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis: Folgt gleich.

Ein PAS für Multiprozessor-Scheduling

Für eine Konstante ℓ ($1 \leq \ell \leq n$) definiere den Algorithmus $\mathcal{A}_\ell$ wie folgt.

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$ polynomiell für konstantes $\ell \rightarrow O(m^\ell + n \log n)$

- Sortiere die Jobs absteigend entsprechend ihrer Laufzeit. $O(n \log n)$
- Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$. $O(m^\ell)$
- Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis: Folgt gleich.

Zu $\varepsilon > 0$ sei $\mathcal{A}_\varepsilon$ ein Algorithmus $\mathcal{A}_\ell$ mit ℓ so gewählt, dass $\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \varepsilon$.

Folgerung: PAS **(Folgerung 7.14)**

Für konstant viele Maschinen ist $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ein PAS.

$\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist kein FPAS, da die Laufzeit nicht polynomiell in $\frac{1}{\varepsilon}$ ist.

Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

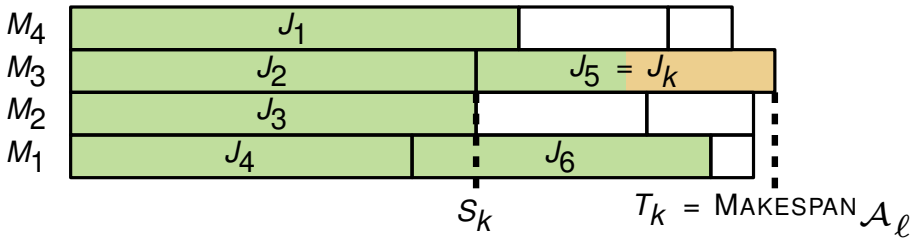
Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

Fall 1: J_k ist einer der ℓ größten Jobs $J_1, \ldots, J_\ell$.



Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

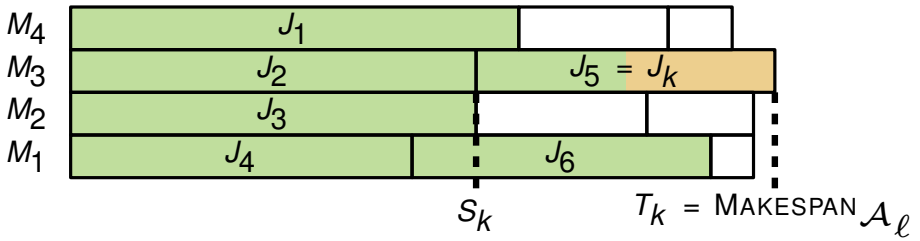
Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

Fall 1: J_k ist einer der ℓ größten Jobs $J_1, \ldots, J_\ell$.



- Für Jobs $J_1, \ldots, J_\ell$ wurde eine optimale Lösung berechnet.
- $\Rightarrow$ Die gefundene Lösung ist auch optimal für $J_1, \ldots, J_n$.

Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

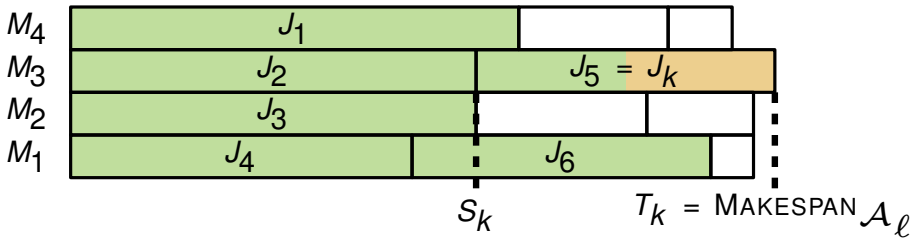
Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

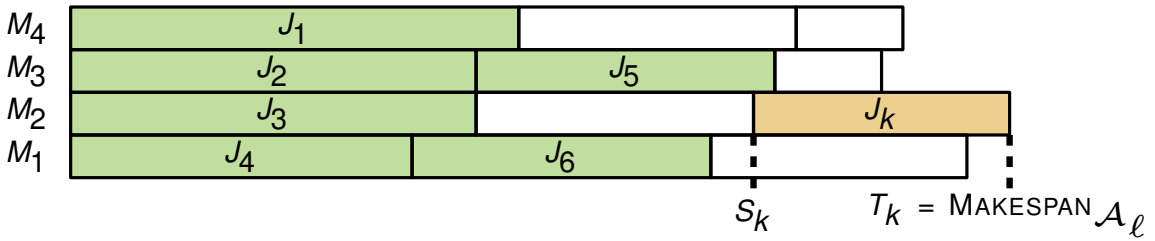
Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

Fall 1: J_k ist einer der ℓ größten Jobs $J_1, \ldots, J_\ell$.



- Für Jobs $J_1, \ldots, J_\ell$ wurde eine optimale Lösung berechnet.
- $\Rightarrow$ Die gefundene Lösung ist auch optimal für $J_1, \ldots, J_n$.

Fall 2: J_k ist nicht einer der ℓ größten Jobs $J_1, \ldots, J_\ell$.



Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

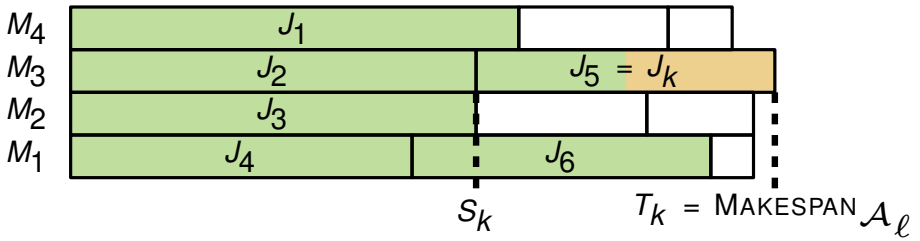
Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

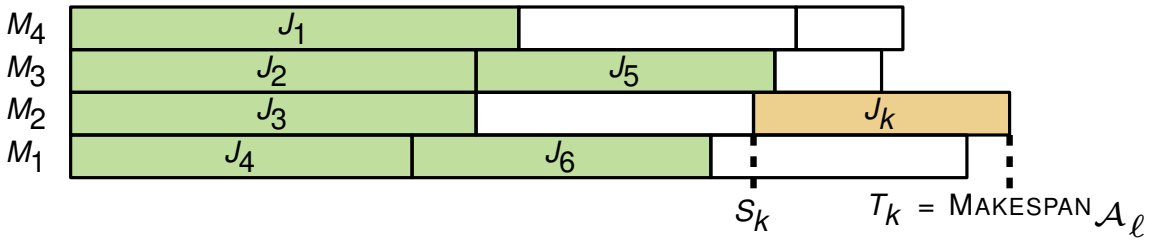
Beweis: Sei J_k letzter Job mit Startzeit S_k und Abschlusszeit $T_k = \text{MAKESPAN}_{\mathcal{A}}$

Fall 1: J_k ist einer der ℓ größten Jobs $J_1, \ldots, J_\ell$.



- Für Jobs $J_1, \ldots, J_\ell$ wurde eine optimale Lösung berechnet.
- $\Rightarrow$ Die gefundene Lösung ist auch optimal für $J_1, \ldots, J_n$.

Fall 2: J_k ist nicht einer der ℓ größten Jobs $J_1, \ldots, J_\ell$.



- Ähnliche Abschätzung wie bei LIST SCHEDULING.
- Nutze aus, dass es $\ell + 1$ Jobs gibt, die mindestens so groß sind wie J_k (einschließlich J_k).

(siehe nächste Folie)

Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

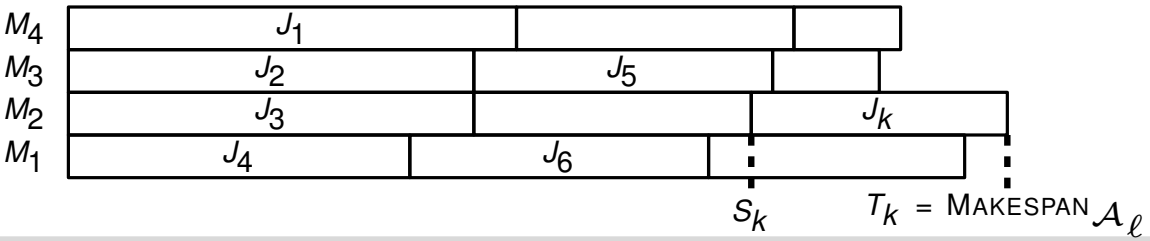
Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis Fall 2: Zunächst genauso wie bei LIST SCHEDULING



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis Fall 2: Zunächst genauso wie bei LIST SCHEDULING

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$

$\sum_{i \neq k} p_i$: Gewicht aller Jobs außer J_k

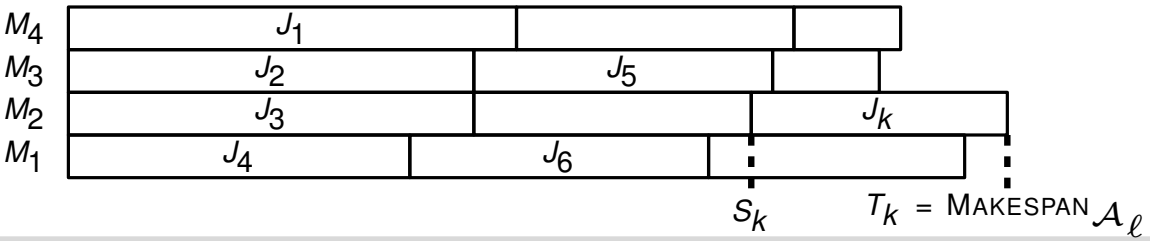
$\frac{1}{m} \sum_{i \neq k} p_i$: gleichmäßig auf m Maschinen verteilt

$T_{\text{OPT}} \geq p_k$ denn J_k muss ausgeführt werden

$$T_{\text{OPT}} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$$

$\sum_{i=1}^{n} p_i$: Gewicht aller Jobs

$\frac{1}{m} \sum_{i=1}^{n} p_i$: gleichmäßig auf m Maschinen verteilt



Es folgt:

$$T_k = S_k + p_k \leq \frac{1}{m} \cdot \sum_{i \neq k} p_i + p_k$$

Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

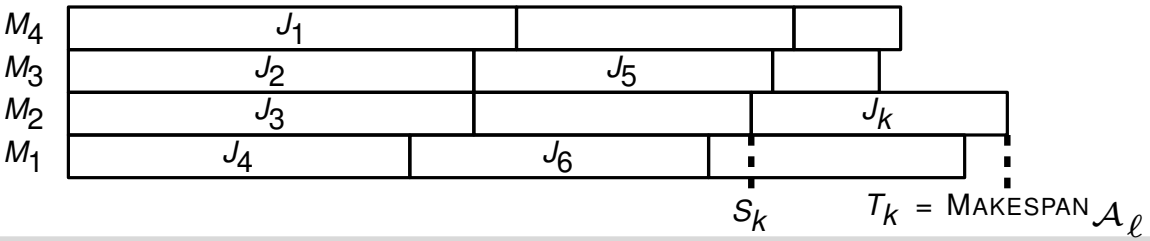
$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis Fall 2: Zunächst genauso wie bei LIST SCHEDULING

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$ Gewicht aller Jobs außer J_k gleichmäßig auf m Maschinen verteilt

$T_{\text{OPT}} \geq p_k$ denn J_k muss ausgeführt werden

$$T_{\text{OPT}} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$$ Gewicht aller Jobs gleichmäßig auf m Maschinen verteilt



Es folgt:

$$T_k = S_k + p_k$$

$$\leq \frac{1}{m} \cdot \sum_{i \neq k} p_i + p_k$$

$$= \frac{1}{m} \cdot \sum_{i=1}^{m} p_i + \left(1 - \frac{1}{m}\right) \cdot p_k$$

Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

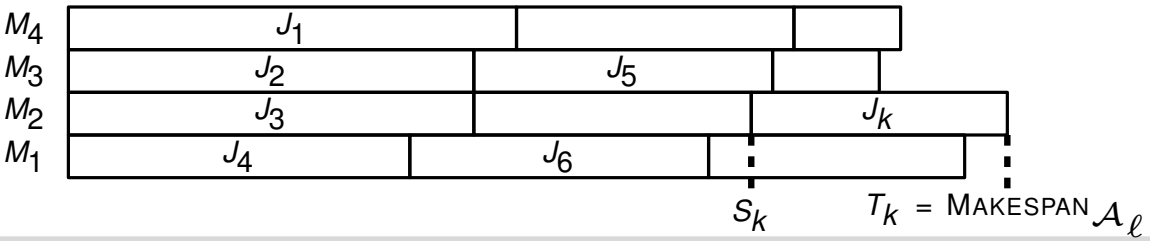
$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis Fall 2: Zunächst genauso wie bei LIST SCHEDULING

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$ Gewicht aller Jobs außer J_k gleichmäßig auf m Maschinen verteilt

$T_{\text{OPT}} \geq p_k$ denn J_k muss ausgeführt werden

$$T_{\text{OPT}} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$$ Gewicht aller Jobs gleichmäßig auf m Maschinen verteilt



Es folgt:

$$T_k = S_k + p_k$$

$$\leq \frac{1}{m} \cdot \sum_{i \neq k} p_i + p_k$$

$$= \frac{1}{m} \cdot \sum_{i=1}^{m} p_i + \left(1 - \frac{1}{m}\right) \cdot p_k$$

$$\leq T_{\text{OPT}} + \left(1 - \frac{1}{m}\right) \cdot T_{\text{OPT}}$$

geht das besser?

Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis Fall 2: Zunächst genauso wie bei LIST SCHEDULING

$$S_k \leq \frac{1}{m} \sum_{i \neq k} p_i$$

$\sum_{i \neq k} p_i$: Gewicht aller Jobs außer J_k

$\frac{1}{m} \sum_{i \neq k} p_i$: gleichmäßig auf m Maschinen verteilt

Betrachte nur die Jobs $J_1, \ldots, J_\ell, J_k$:

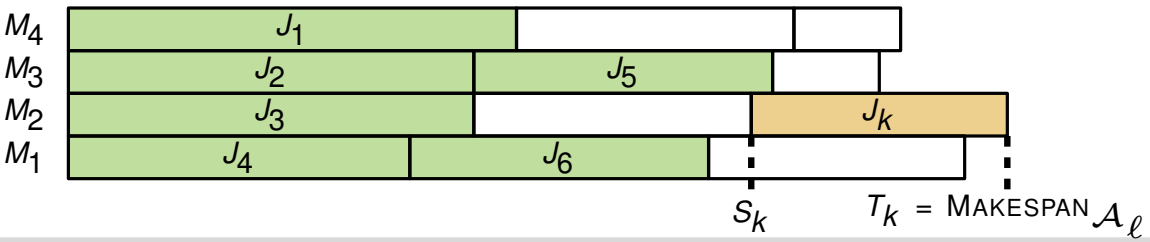
$$T_{\text{OPT}} \geq p_k \cdot \left(1 + \left\lfloor \frac{\ell}{m} \right\rfloor\right)$$

$\left(1 + \left\lfloor \frac{\ell}{m} \right\rfloor\right)$: es gibt Maschine mit so vielen Jobs $(\star)$

p_k: jeder hat Größe $\geq p_k$

Begründung für $(\star)$:

- Im Schnitt hat jede Maschine mehr als $\frac{\ell}{m}$ der $\ell + 1$ Jobs.
- Mind. eine Masch. muss den Schnitt (oder mehr) erreichen.



Es folgt:

$$\begin{aligned} T_k &= S_k + p_k \\ &\leq \frac{1}{m} \cdot \sum_{i \neq k} p_i + p_k \\ &= \frac{1}{m} \cdot \sum_{i=1}^{m} p_i + \left(1 - \frac{1}{m}\right) \cdot p_k \\ &\leq T_{\text{OPT}} + \left(1 - \frac{1}{m}\right) \cdot T_{\text{OPT}} \end{aligned}$$

geht das besser?

Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

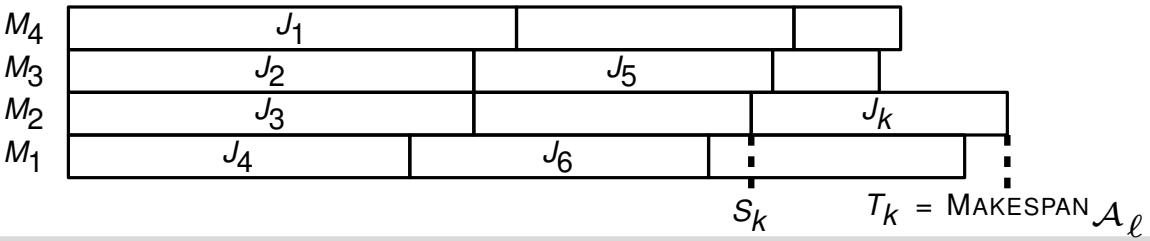
Beweis Fall 2: Zunächst genauso wie bei LIST SCHEDULING

$$S_k \leq \frac{1}{m} \underbrace{\sum_{i \neq k} p_i}_{\text{Gewicht aller Jobs außer } J_k} \quad \text{gleichmäßig auf } m \text{ Maschinen verteilt}$$

$$T_{\text{OPT}} \geq p_k \cdot \left(1 + \left\lfloor \frac{\ell}{m} \right\rfloor\right) \quad \text{es gibt Maschine mit so vielen Jobs} \quad (\star)$$

jeder hat Größe $\geq p_k$

$$T_{\text{OPT}} \geq \frac{1}{m} \underbrace{\sum_{i=1}^{n} p_i}_{\text{Gewicht aller Jobs}} \quad \text{gleichmäßig auf } m \text{ Maschinen verteilt}$$



Es folgt:

$$\begin{aligned} T_k &= S_k + p_k \\ &\leq \frac{1}{m} \cdot \sum_{i \neq k} p_i + p_k \\ &= \underbrace{\frac{1}{m} \cdot \sum_{i=1}^{m} p_i}_{\leq T_{\text{OPT}}} + \left(1 - \frac{1}{m}\right) \cdot p_k \\ &\leq T_{\text{OPT}} + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor} \cdot T_{\text{OPT}} \end{aligned}$$

Ein PAS für Multiprozessor-Scheduling

$\mathcal{A}_\ell(J_1, \ldots, J_n, m)$

Sortiere die Jobs absteigend entsprechend ihrer Laufzeit.

Bestimme optimalen Schedule für die ℓ größten Jobs $J_1, \ldots, J_\ell$.

Ordne die restlichen Jobs $J_{\ell+1}, \ldots, J_n$ den Maschinen gemäß LIST SCHEDULING zu.

Satz: Approximation **(Satz 7.13)**

Für die Approximationsrate von $\mathcal{A}_\ell$ mit $1 \leq \ell \leq n$ konstant gilt:

$$\mathcal{R}_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis Fall 2: Zunächst genauso wie bei LIST SCHEDULING

$$S_k \leq \frac{1}{m} \underbrace{\sum_{i \neq k} p_i}_{\text{Gewicht aller Jobs außer } J_k}$$

gleichmäßig auf m Maschinen verteilt

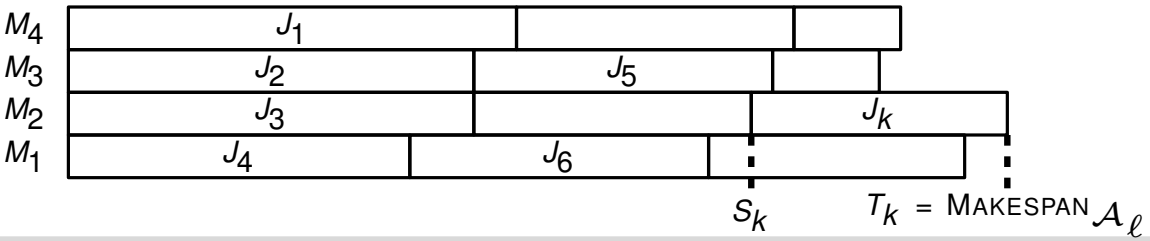
$$T_{\text{OPT}} \geq p_k \cdot \left(1 + \left\lfloor \frac{\ell}{m} \right\rfloor\right)$$

es gibt Maschine mit so vielen Jobs $(\star)$

jeder hat Größe $\geq p_k$

$$T_{\text{OPT}} \geq \frac{1}{m} \underbrace{\sum_{i=1}^{n} p_i}_{\text{Gewicht aller Jobs}}$$

gleichmäßig auf m Maschinen verteilt



Es folgt:

$$\begin{aligned} T_k &= S_k + p_k \\ &\leq \frac{1}{m} \cdot \sum_{i \neq k} p_i + p_k \\ &= \frac{1}{m} \cdot \sum_{i=1}^{m} p_i + \left(1 - \frac{1}{m}\right) \cdot p_k \\ &\leq T_{\text{OPT}} + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor} \cdot T_{\text{OPT}} \end{aligned}$$

$$\Rightarrow \frac{T_k}{T_{\text{OPT}}} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

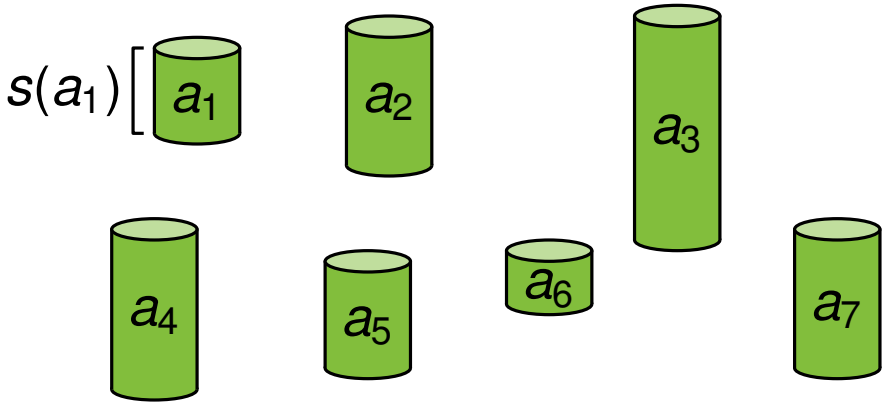
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

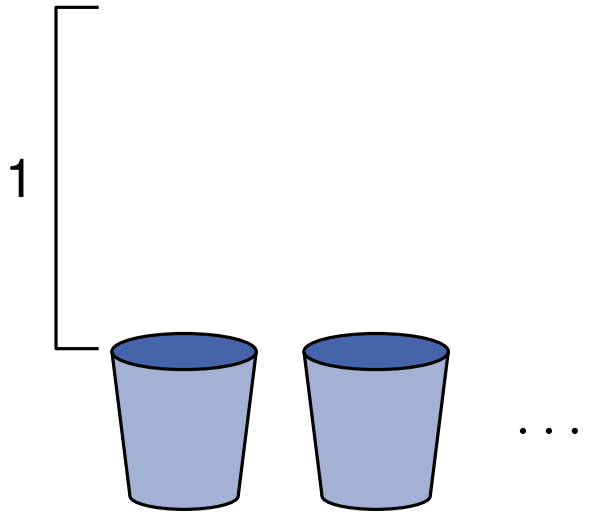
Bin Packing

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:

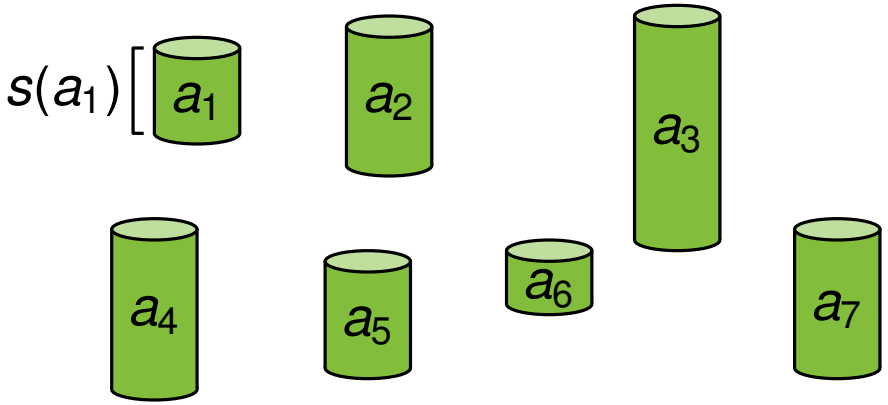
$$\sum_{a_i \in B} s(a_i) \leq 1$$

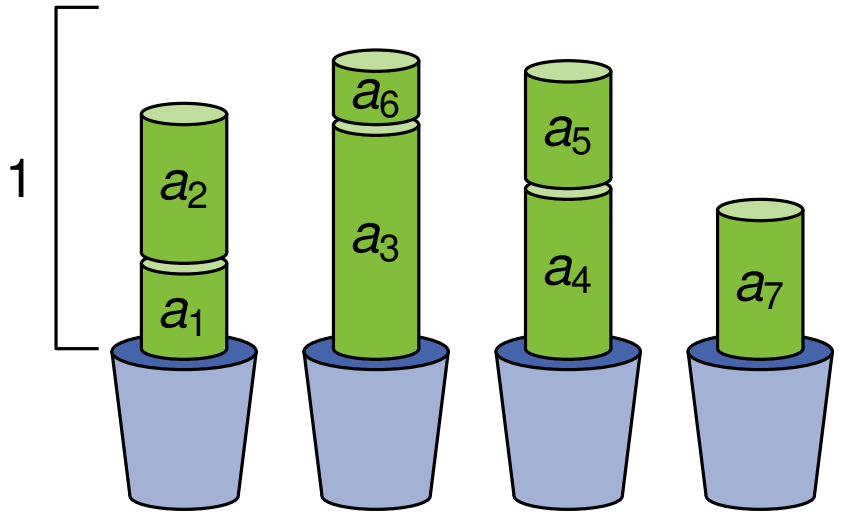
BIN PACKING ist $\mathcal{NP}$-schwer.

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:

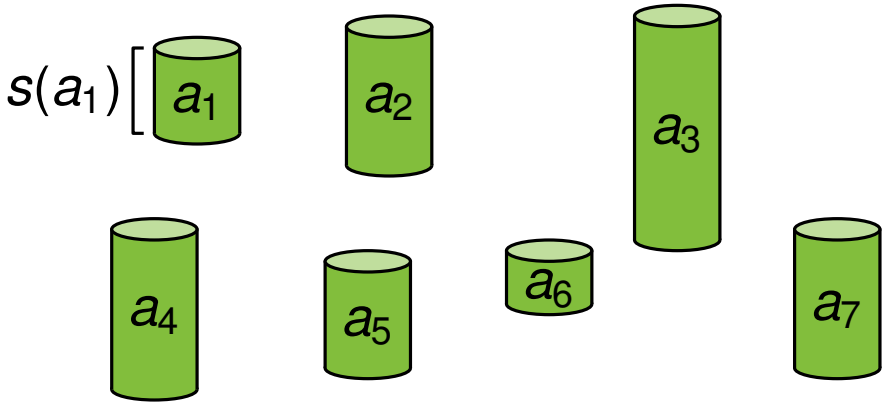
$$\sum_{a_i \in B} s(a_i) \leq 1$$

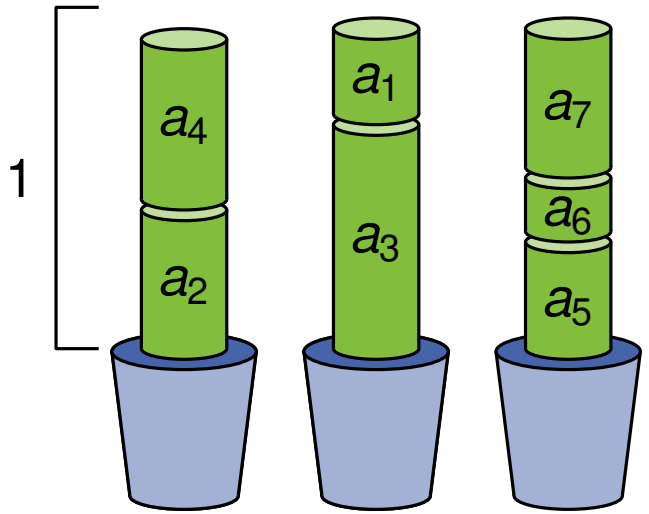
BIN PACKING ist $\mathcal{NP}$-schwer.

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

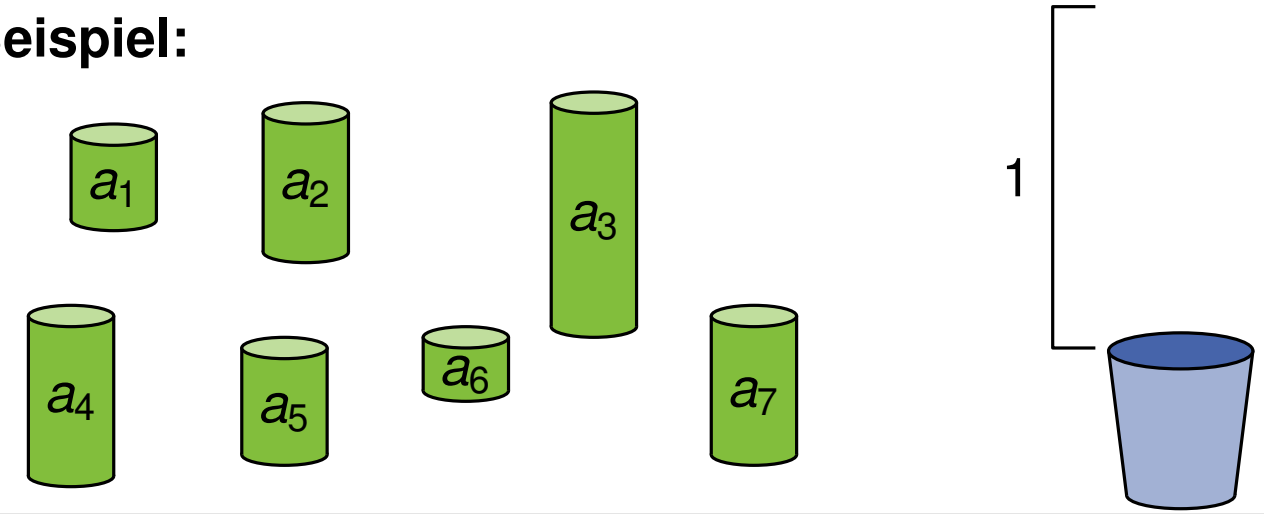
Strategie:

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:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 1 – Next Fit

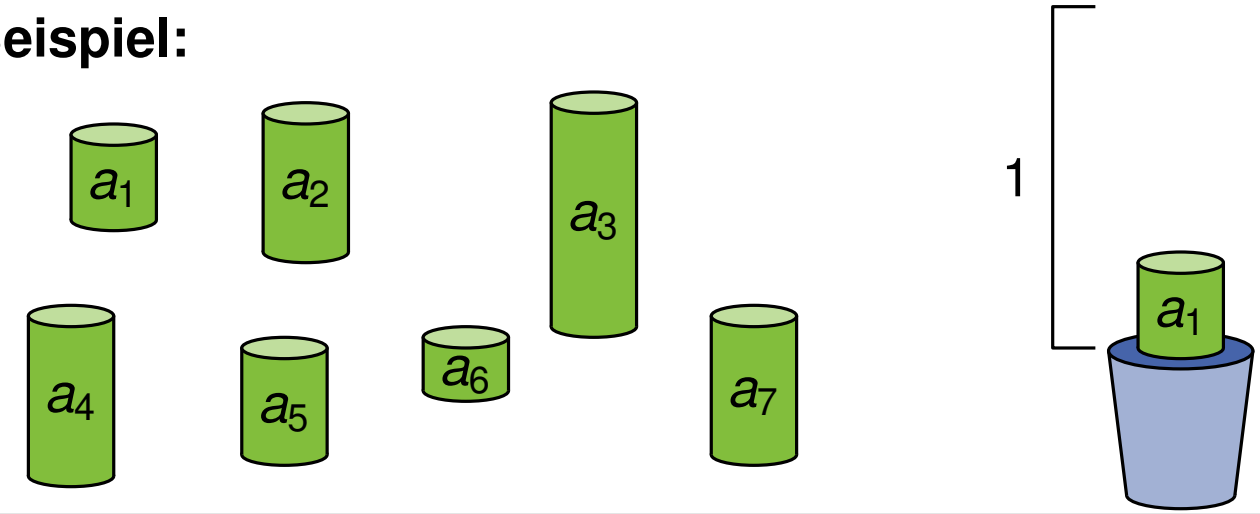
Strategie:

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

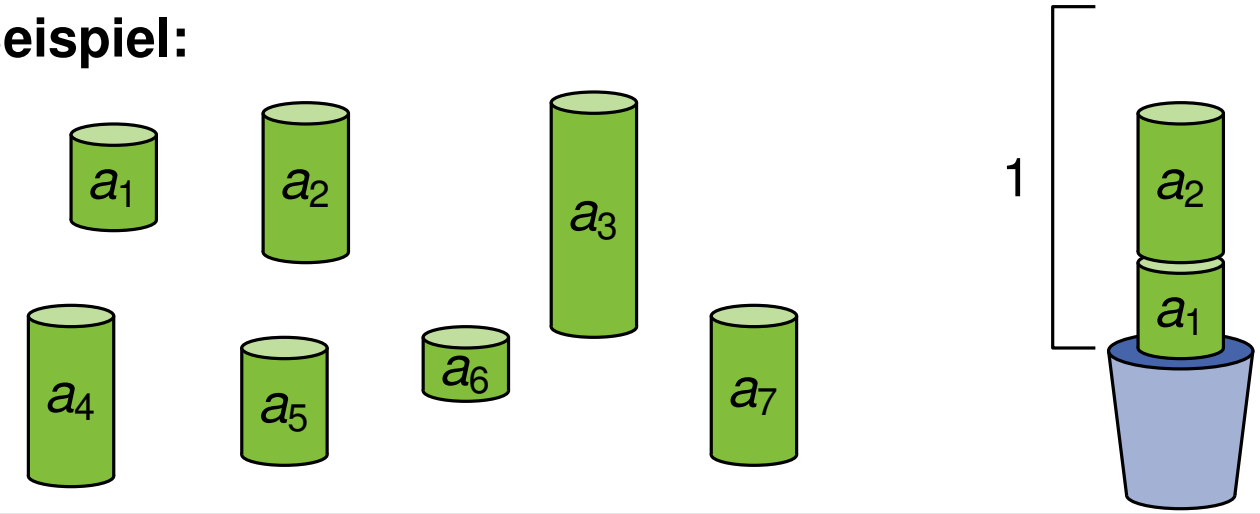
Strategie:

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

Lösungsstrategie 1 – Next Fit

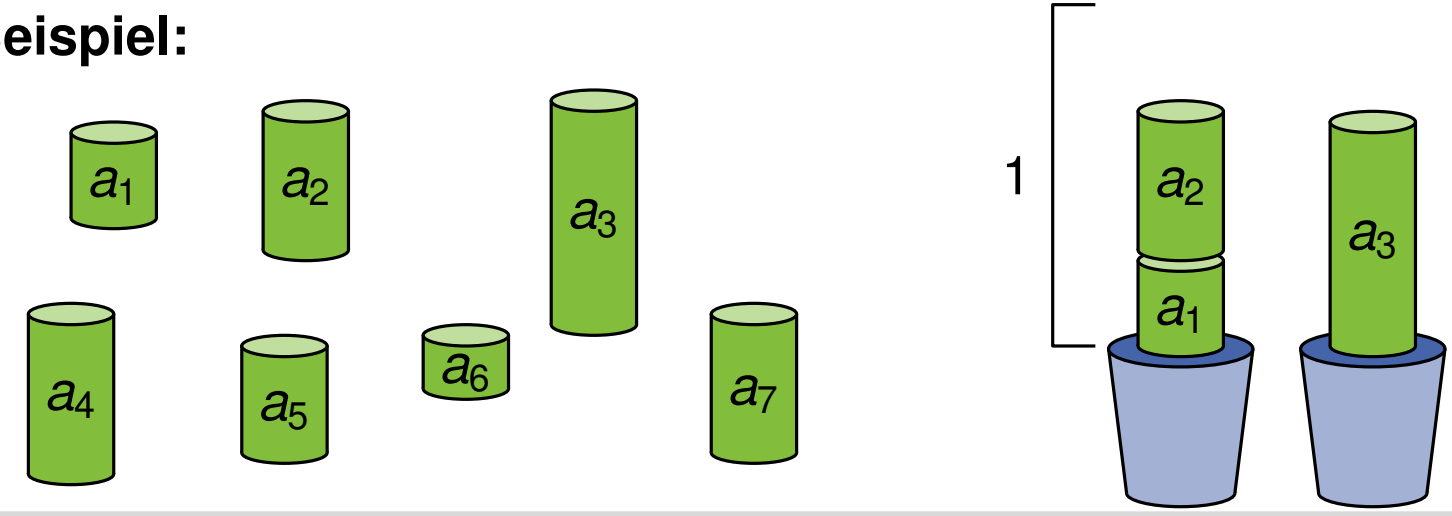
Strategie:

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

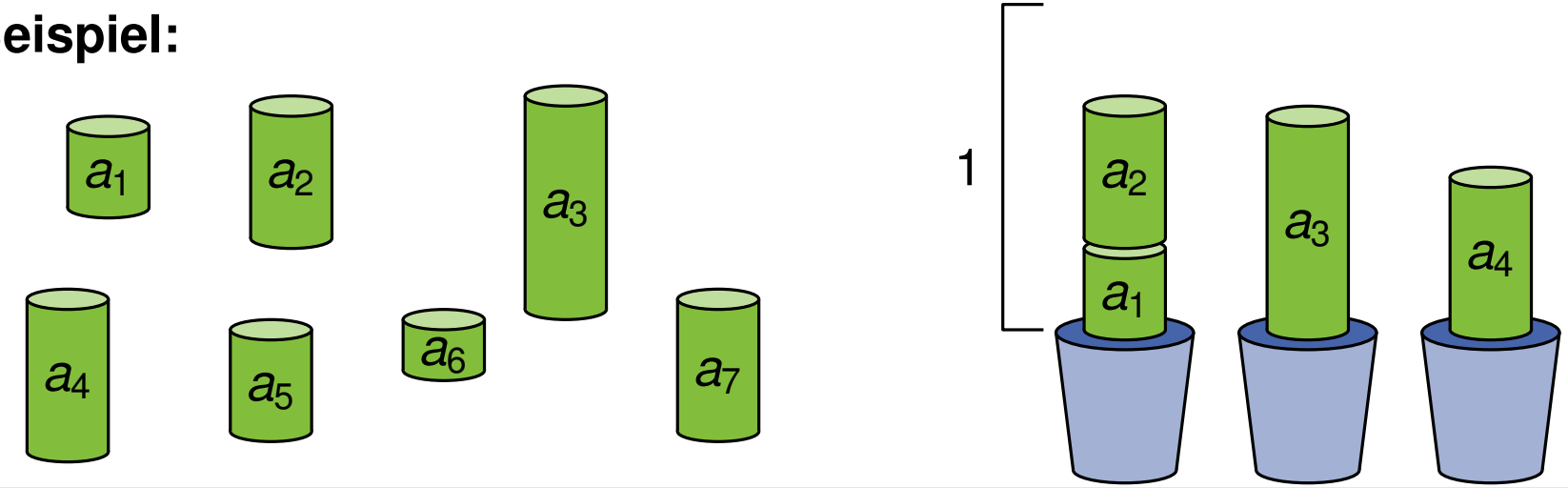
Strategie:

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

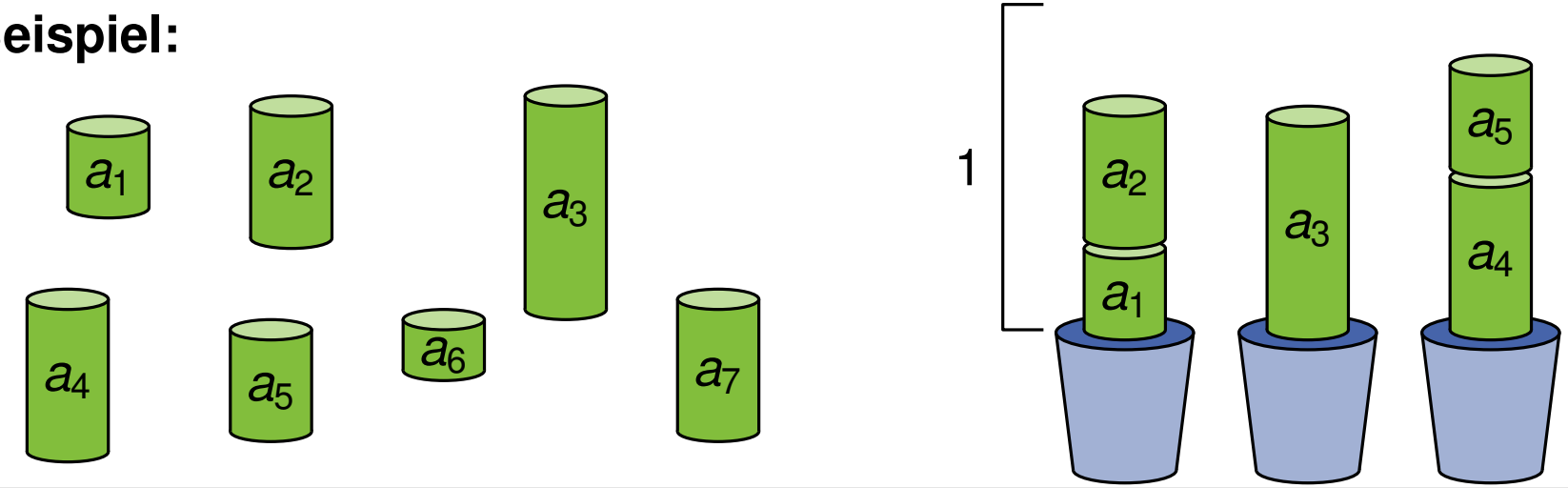
Strategie:

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_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

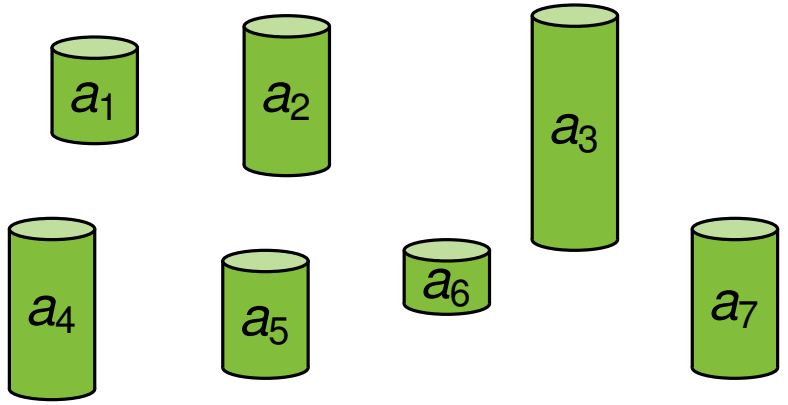
Strategie:

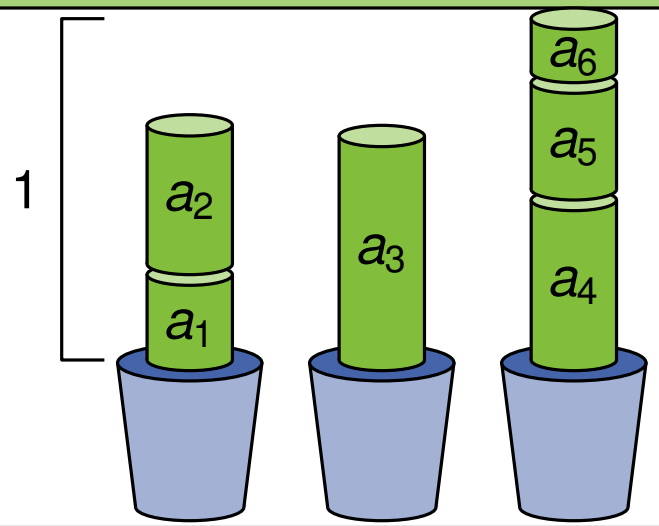
Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

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

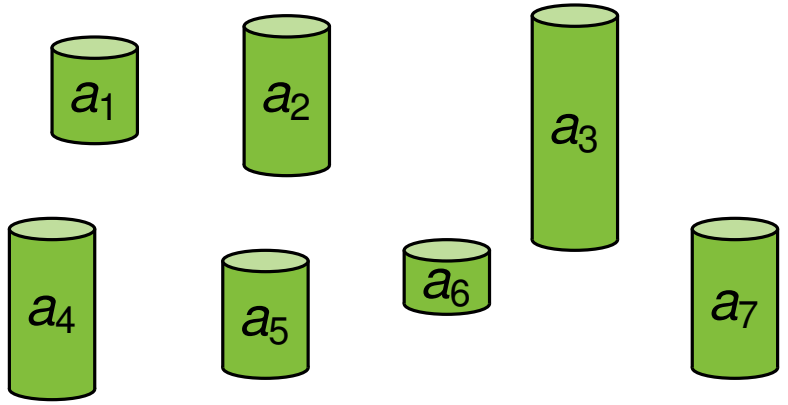
Strategie:

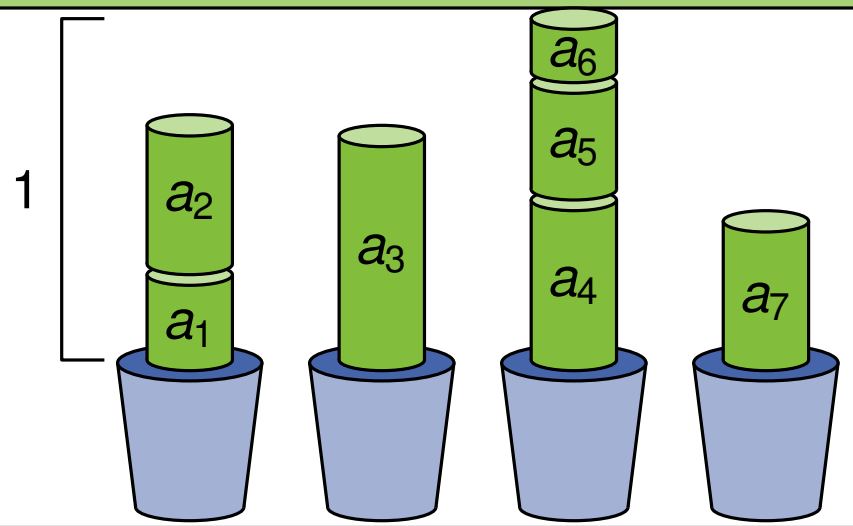
Füge Elemente nacheinander in den aktuellen Bin ein. Wenn ein Element nicht mehr passt, schließe den Bin ab und nimm einen neuen.

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

Next Fit – Approximation

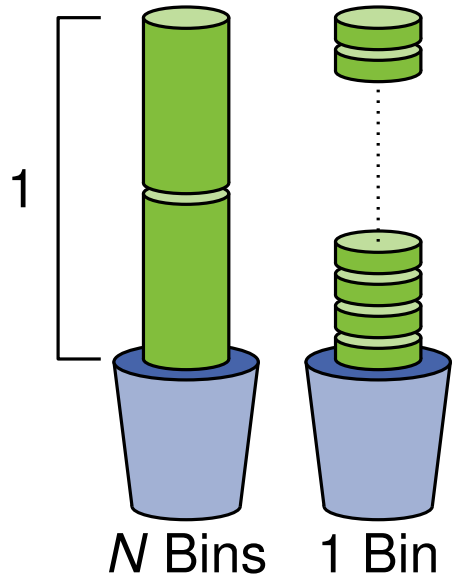
Schlechtes Beispiel

(Beispiel 7.4)

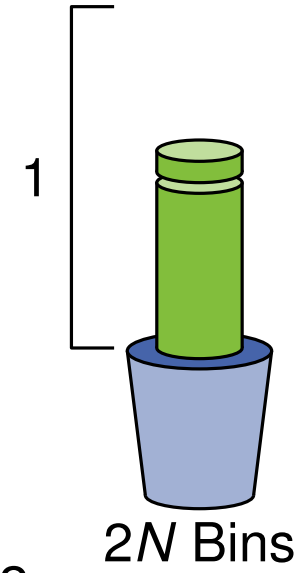
Betrachte die Menge $\{a_1, \ldots, a_n\}$ für $n = 4 \cdot N$ mit $N \in \mathbb{N}$. Sei weiterhin

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

Optimale Lösung OPT(I)



Lösung von NEXT FIT NF(I)



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

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

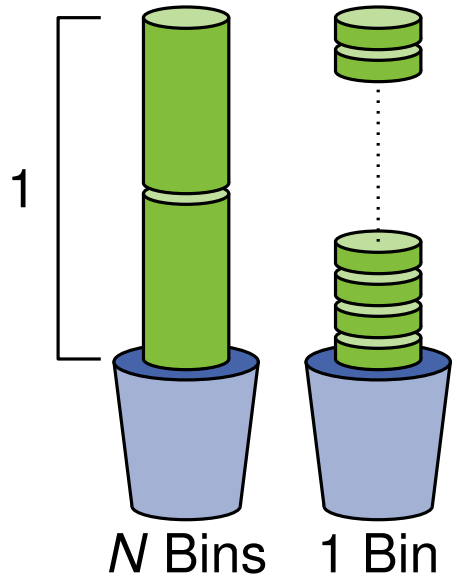
Next Fit – Approximation

Schlechtes Beispiel **(Beispiel 7.4)**

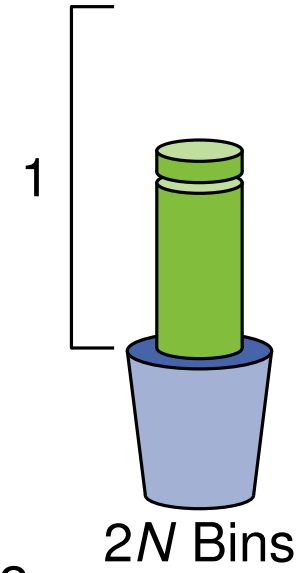
Betrachte die Menge $\{a_1, \ldots, a_n\}$ für $n = 4 \cdot N$ mit $N \in \mathbb{N}$. Sei weiterhin

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

Optimale Lösung OPT(I)



Lösung von NEXT FIT NF(I)



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

Satz: Approximation **(Satz 7.5)**

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

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

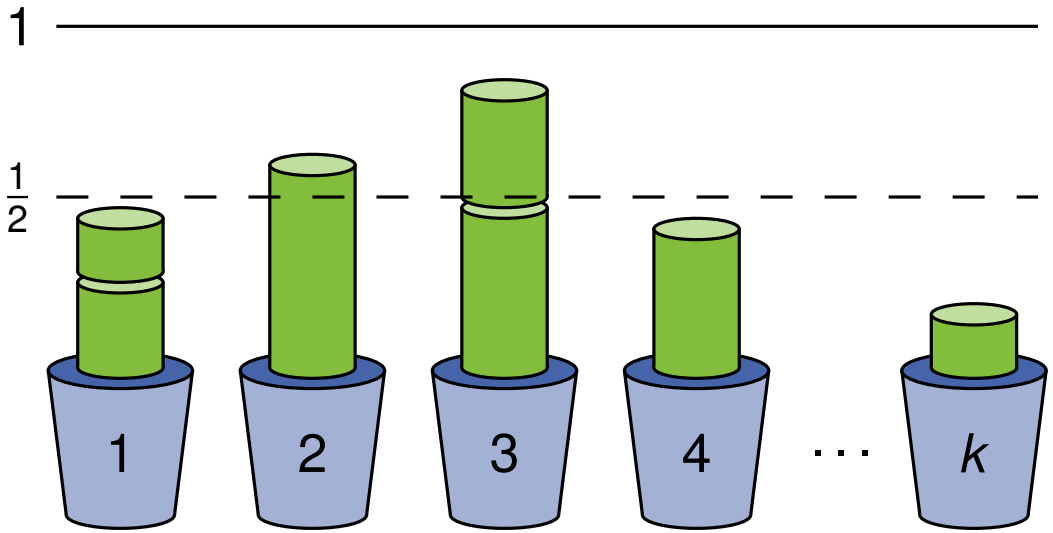
Next Fit – Approximation

Satz: Approximation **(Satz 7.5)**

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

Beweis:

- Sei $k = NF(I)$ die Anzahl an Bins, die NEXTFIT für die Instanz I benötigt.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

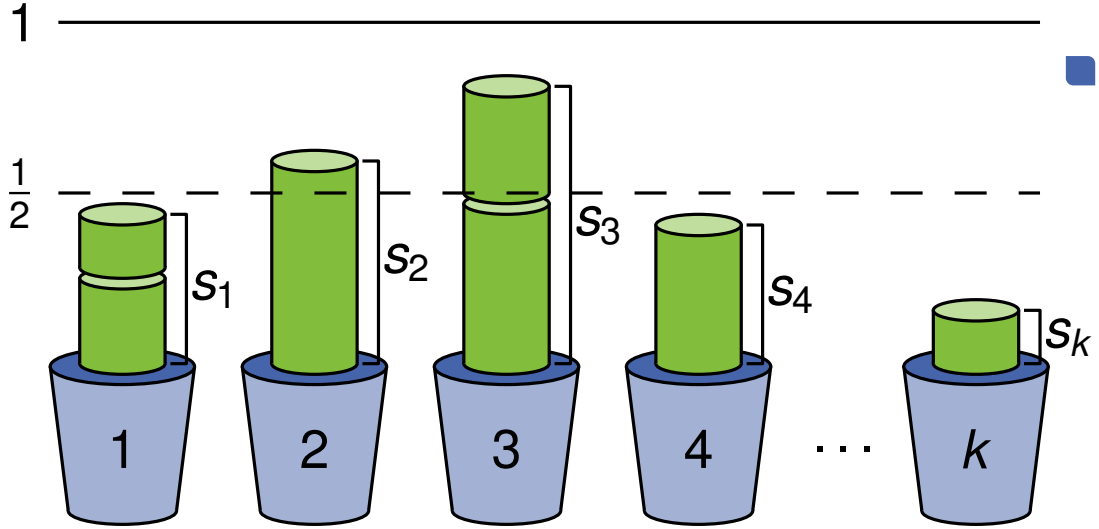
Next Fit – Approximation

Satz: Approximation **(Satz 7.5)**

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

Beweis:

- Sei $k = NF(I)$ die Anzahl an Bins, die NEXTFIT für die Instanz I benötigt.



- Sei s_i die Größe der Elemente in Bin i.

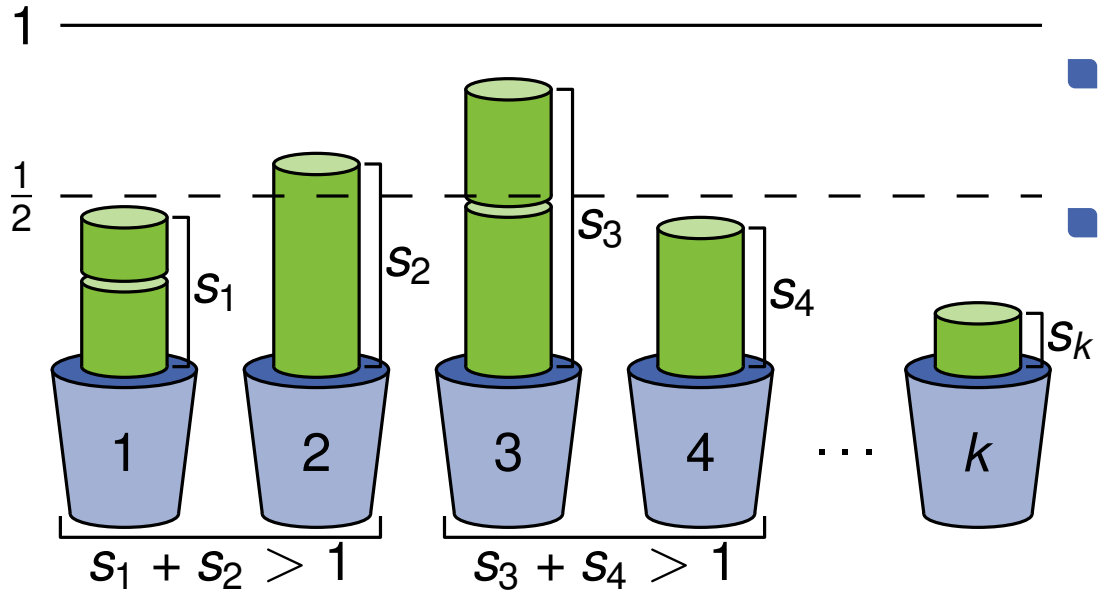
Next Fit – Approximation

Satz: Approximation **(Satz 7.5)**

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

Beweis:

- Sei $k = NF(I)$ die Anzahl an Bins, die NEXTFIT für die Instanz I benötigt.



- Sei s_i die Größe der Elemente in Bin i.
- Für zwei aufeinanderfolgende Bins gilt: $s_i + s_{i+1} > 1$
  (sonst hätten die Elemente in Bin $i + 1$ noch in Bin i gepasst)

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

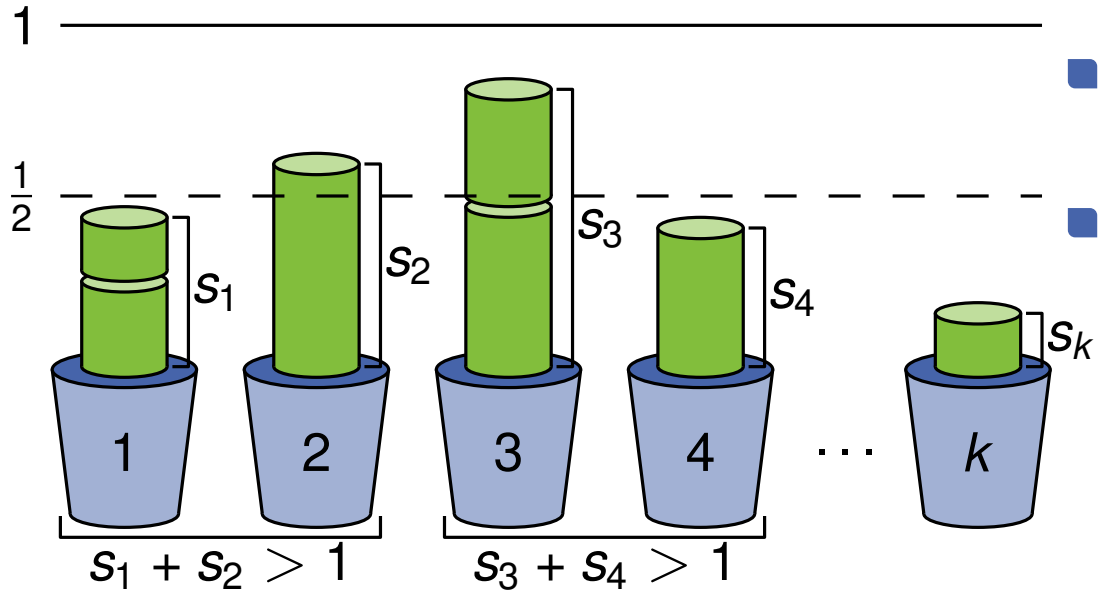
Next Fit – Approximation

Satz: Approximation **(Satz 7.5)**

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

Beweis:

- Sei $k = NF(I)$ die Anzahl an Bins, die NEXTFIT für die Instanz I benötigt.



- Sei s_i die Größe der Elemente in Bin i.
- Für zwei aufeinanderfolgende Bins gilt: $s_i + s_{i+1} > 1$
  (sonst hätten die Elemente in Bin $i + 1$ noch in Bin i gepasst)

$$\Rightarrow \sum_{i=1}^{k} s_i > \frac{k}{2} \text{ falls } k \text{ gerade bzw. } \sum_{i=1}^{k-1} s_i > \frac{k-1}{2} \text{ falls } k \text{ ungerade}$$

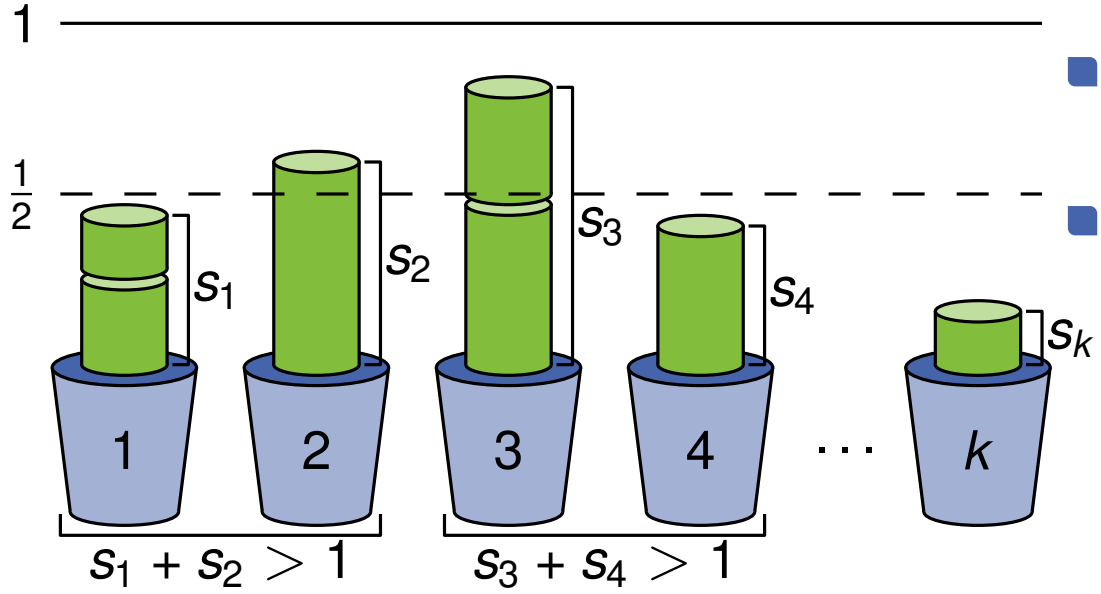
Next Fit – Approximation

Satz: Approximation **(Satz 7.5)**

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

Beweis:

- Sei $k = \mathrm{NF}(I)$ die Anzahl an Bins, die NEXTFIT für die Instanz I benötigt.



- Sei s_i die Größe der Elemente in Bin i.
- Für zwei aufeinanderfolgende Bins gilt: $s_i + s_{i+1} > 1$
  (sonst hätten die Elemente in Bin $i+1$ noch in Bin i gepasst)

$$\Rightarrow \sum_{i=1}^{k} s_i > \frac{k}{2} \text{ falls } k \text{ gerade bzw. } \sum_{i=1}^{k-1} s_i > \frac{k-1}{2} \text{ falls } k \text{ ungerade}$$

$$\Rightarrow \mathrm{OPT}(I) > \frac{k-1}{2} \Rightarrow \mathrm{NF}(I) = k < 2\mathrm{OPT}(I) + 1 \Rightarrow \mathrm{NF}(I) \leq 2\mathrm{OPT}(I)$$

Lösungsstrategie 2 – First Fit

Strategie:

Füge aktuelles Element immer in den ersten Bin ein, in den es passt.

FIRST FIT (FF)(M, s) — Laufzeit: $O(n^2)$

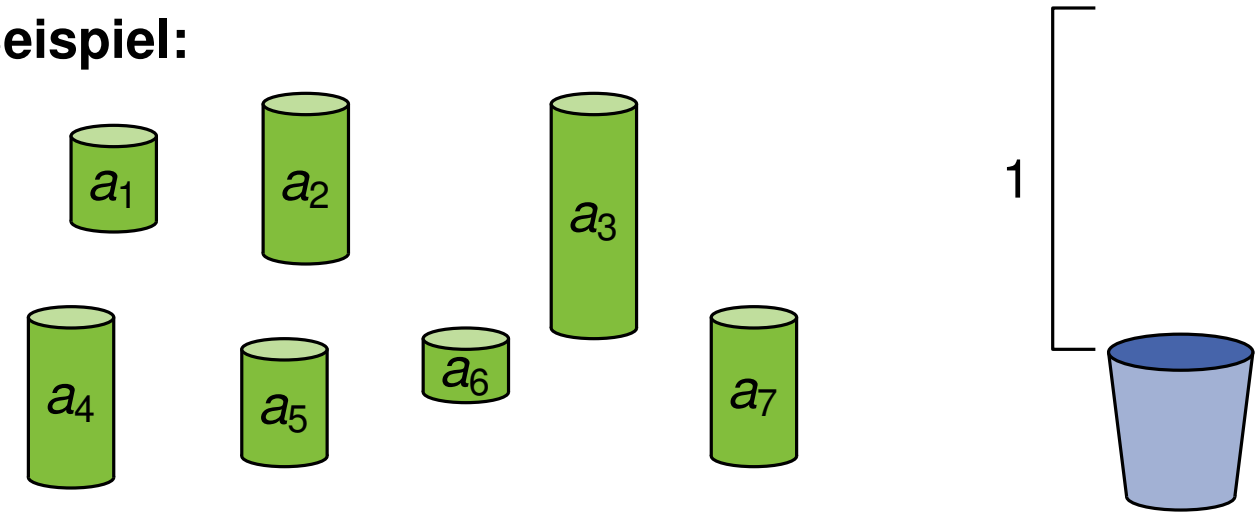
Füge a_1 in B_1 ein

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

  $B_j \leftarrow$ erster Bin mit $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$

  Füge a_ℓ in B_j ein

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 2 – First Fit

Strategie:

Füge aktuelles Element immer in den ersten Bin ein, in den es passt.

FIRST FIT (FF)(M, s) — Laufzeit: $O(n^2)$

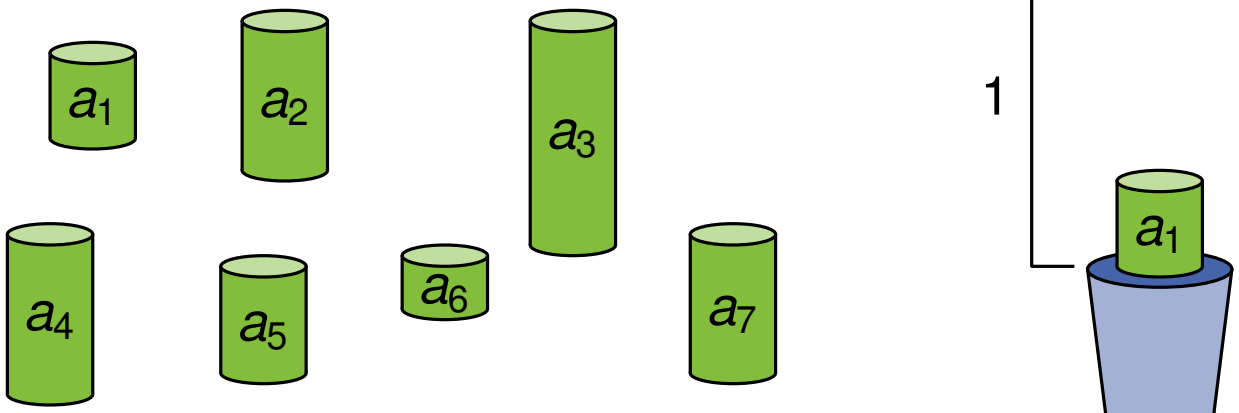
Füge a_1 in B_1 ein

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

  $B_j \leftarrow$ erster Bin mit $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$

  Füge a_ℓ in B_j ein

Beispiel:



Lösungsstrategie 2 – First Fit

Strategie:

Füge aktuelles Element immer in den ersten Bin ein, in den es passt.

FIRST FIT (FF)(M, s) — Laufzeit: $O(n^2)$

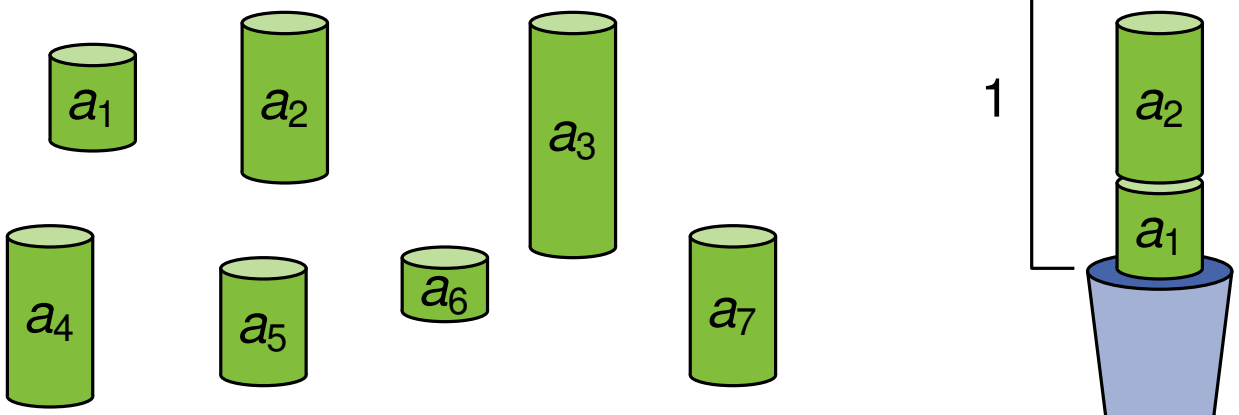
Füge a_1 in B_1 ein

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

  $B_j \leftarrow$ erster Bin mit $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$

  Füge a_ℓ in B_j ein

Beispiel:



Lösungsstrategie 2 – First Fit

Strategie:

Füge aktuelles Element immer in den ersten Bin ein, in den es passt.

FIRST FIT (FF)(M, s) — Laufzeit: $O(n^2)$

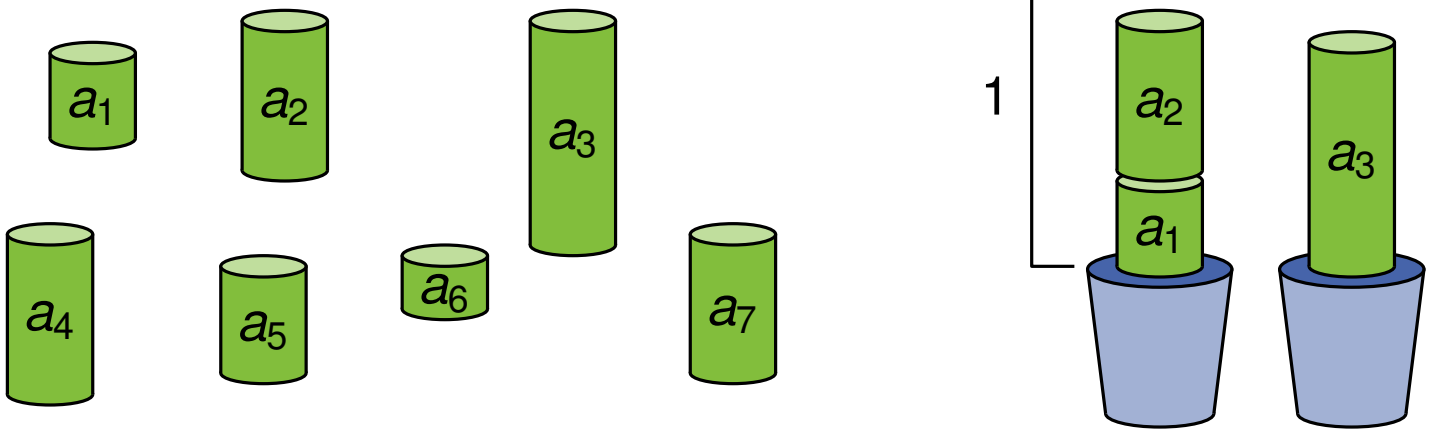
Füge a_1 in B_1 ein

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

  $B_j \leftarrow$ erster Bin mit $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$

  Füge a_ℓ in B_j ein

Beispiel:



Lösungsstrategie 2 – First Fit

Strategie:

Füge aktuelles Element immer in den ersten Bin ein, in den es passt.

FIRST FIT (FF)(M, s) — Laufzeit: $O(n^2)$

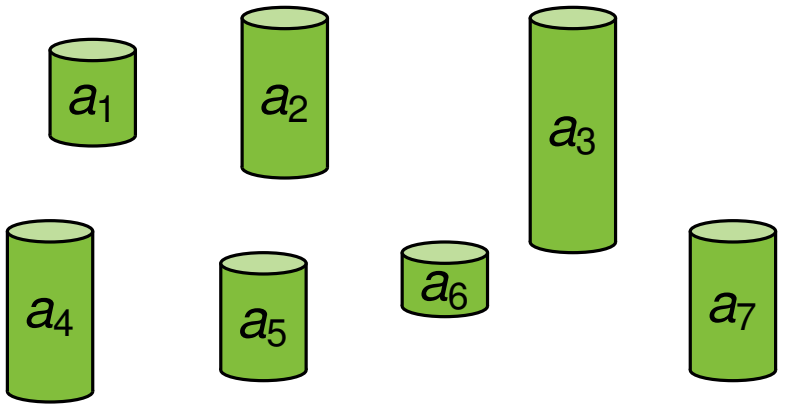
Füge a_1 in B_1 ein

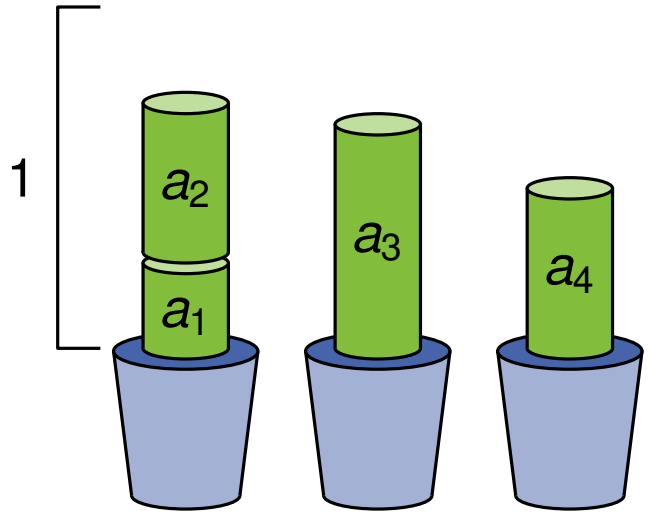
for $a_\ell \in \{a_2, \ldots, a_n\}$ **do**

  $B_j \leftarrow$ erster Bin mit $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$

  Füge a_ℓ in B_j ein

Beispiel:





Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 2 – First Fit

Strategie:

Füge aktuelles Element immer in den ersten Bin ein, in den es passt.

FIRST FIT (FF)(M, s) — Laufzeit: $O(n^2)$

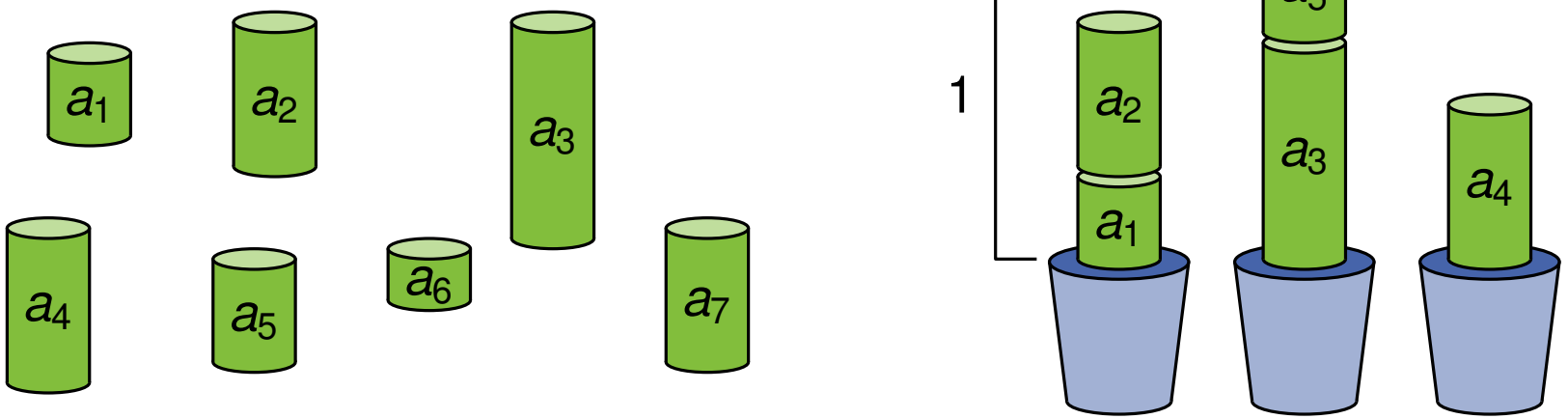
Füge a_1 in B_1 ein

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

  $B_j \leftarrow$ erster Bin mit $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$

  Füge a_ℓ in B_j ein

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 2 – First Fit

Strategie:

Füge aktuelles Element immer in den ersten Bin ein, in den es passt.

FIRST FIT (FF)(M, s) — Laufzeit: $O(n^2)$

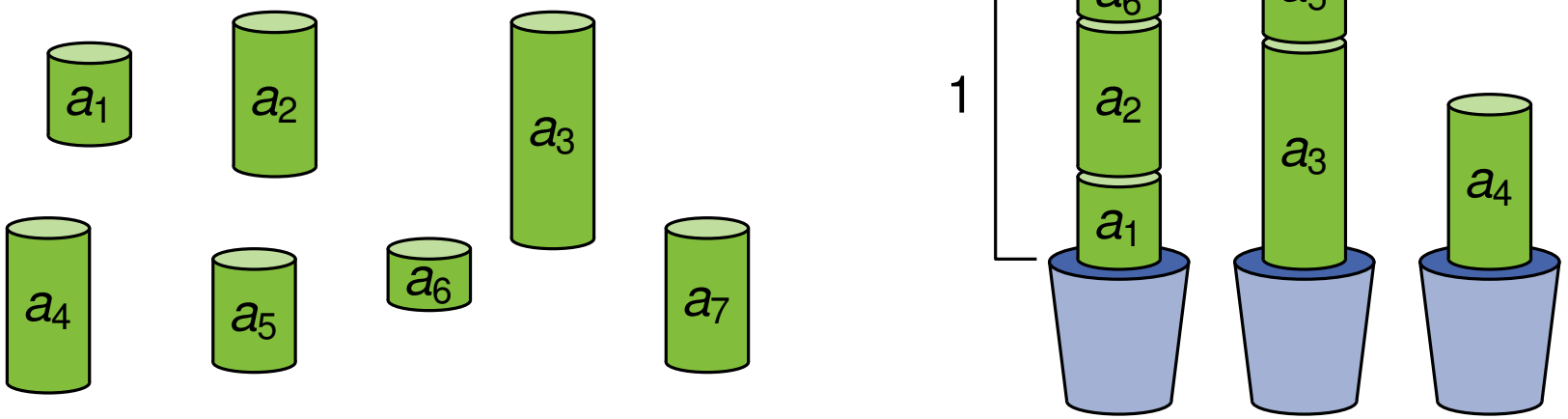
Füge a_1 in B_1 ein

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

  $B_j \leftarrow$ erster Bin mit $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$

  Füge a_ℓ in B_j ein

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Lösungsstrategie 2 – First Fit

Strategie:
Füge aktuelles Element immer in den ersten Bin ein, in den es passt.

FIRST FIT (FF)(M, s) — Laufzeit: $O(n^2)$

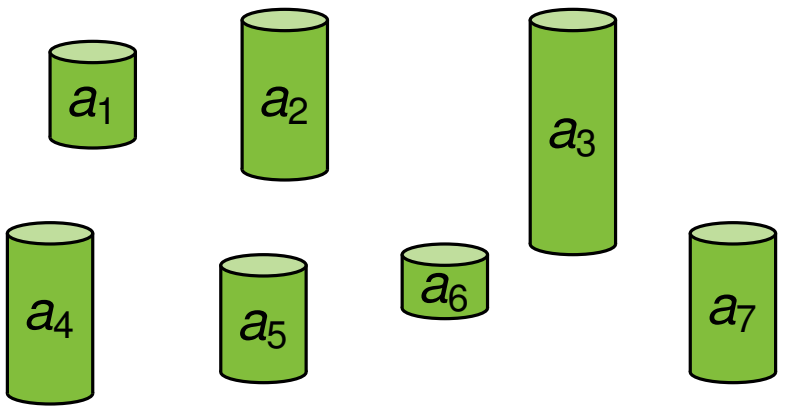
Füge a_1 in B_1 ein

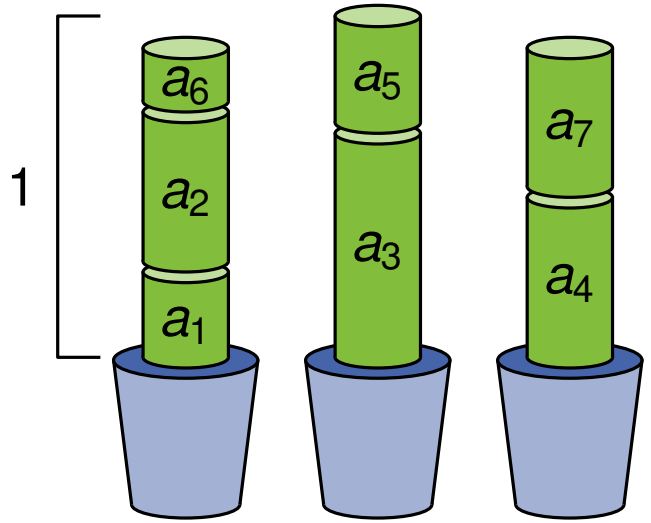
for $a_\ell \in \{a_2, \ldots, a_n\}$ **do**

  $B_j \leftarrow$ erster Bin mit $s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i)$

  Füge a_ℓ in B_j ein

Beispiel:





Für jede Instanz I gilt $FF(I) < 2 \cdot \text{OPT}(I)$

Kann man das besser abschätzen?

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

First Fit – Approximation

Schlechtes Beispiel **(Beispiel 7.7)**

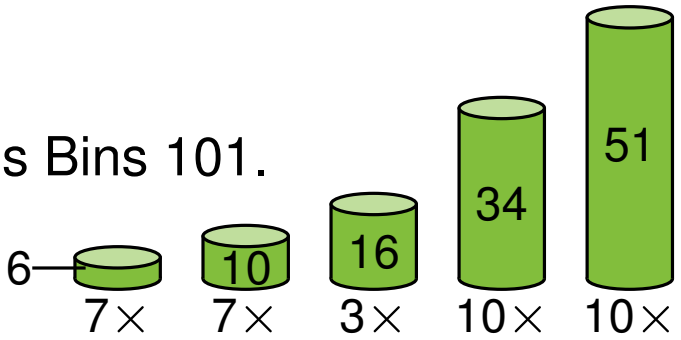
Sei das Fassungsvermögen eines Bins 101 (statt 1) und sei $n = 37$. Sei weiterhin

$$s(a_i) = \begin{cases} 6 & \text{für } 1 \leq i \leq 7 \\ 10 & \text{für } 8 \leq i \leq 14 \\ 16 & \text{für } 15 \leq i \leq 17 \\ 34 & \text{für } 18 \leq i \leq 27 \\ 51 & \text{für } 28 \leq i \leq 37 \end{cases}$$

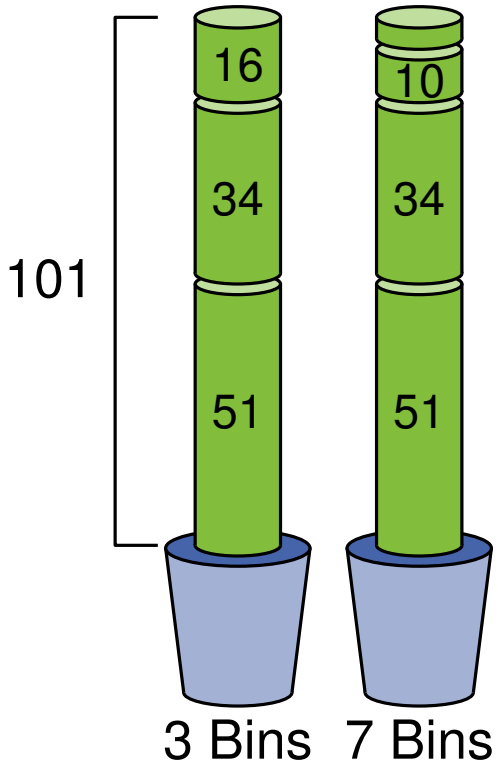
First Fit – Approximation

Schlechtes Beispiel **(Beispiel 7.7)**

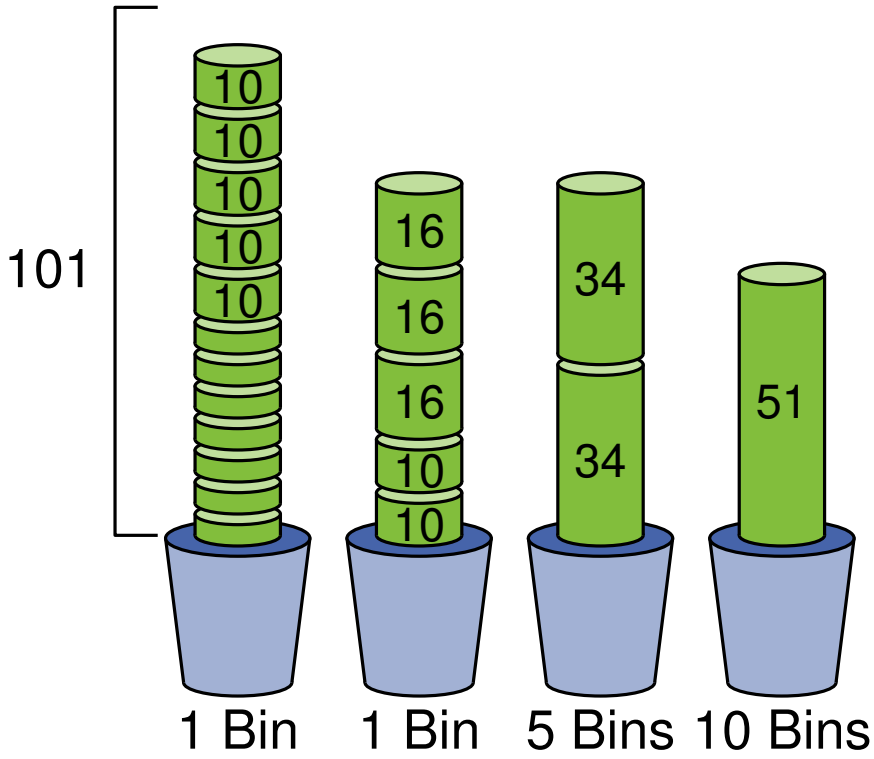
Sei das Fassungsvermögen eines Bins 101.
Betrachte folgende Elemente:



Optimale Lösung OPT(*I*)



Lösung von FIRST FIT FF(*I*)



$$\Rightarrow \mathrm{FF}(I) = 17 \text{ und } \mathrm{OPT}(I) = 10 \Rightarrow \mathcal{R}_{\mathrm{FF}}(I) = \frac{17}{10}$$

First Fit – Approximation

Gerade gesehen: Es gibt eine Instanz I, sodass $\mathcal{R}_{\mathrm{FF}}(I) = \frac{17}{10}$.

Umgekehrt kann es kein viel schlechteres Beispiel geben:

Satz: Approximation **(Satz 7.8)**

Für jede Instanz I von BIN PACKING gilt: $\mathrm{FF}(I) < \frac{17}{10} \cdot \mathrm{OPT}(I) + 1$

First Fit – Approximation

Gerade gesehen: Es gibt eine Instanz I, sodass $\mathcal{R}_{\mathrm{FF}}(I) = \frac{17}{10}$.
Umgekehrt kann es kein viel schlechteres Beispiel geben:

Satz: Approximation **(Satz 7.8)**
Für jede Instanz I von BIN PACKING gilt: $\mathrm{FF}(I) < \frac{17}{10} \cdot \mathrm{OPT}(I) + 1$

Beweisidee:

Definiere Funktion $\omega : [0,1] \to [0,1]$ die hilft, das Verhältnis $\frac{\mathrm{FF}(I)}{\mathrm{OPT}(I)}$ abzuschätzen.

Zeige:

$$\sum_{i=1}^{n} \omega(s(a_i)) \leq \frac{17}{10} \cdot \mathrm{OPT}(I)$$

$$\sum_{i=1}^{n} \omega(s(a_i)) > \mathrm{FF}(I) - 1$$

Für:

$$\omega(a) = \begin{cases} \frac{6}{5} \cdot a & \text{für } 0 \leq a < \frac{1}{6} \\ \frac{9}{5} \cdot a - \frac{1}{10} & \text{für } \frac{1}{6} \leq a < \frac{1}{3} \\ \frac{6}{5} \cdot a + \frac{1}{10} & \text{für } \frac{1}{3} \leq a \leq \frac{1}{2} \\ 1 & \text{für } \frac{1}{2} < a \leq 1 \end{cases}$$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

First Fit – Approximation

Gerade gesehen: Es gibt eine Instanz I, sodass $\mathcal{R}_{\mathrm{FF}}(I) = \frac{17}{10}$.
Umgekehrt kann es kein viel schlechteres Beispiel geben:

Satz: Approximation **(Satz 7.8)**
Für jede Instanz I von BIN PACKING gilt: $\mathrm{FF}(I) < \frac{17}{10} \cdot \mathrm{OPT}(I) + 1$

Beweisidee:

Definiere Funktion $\omega : [0,1] \to [0,1]$ die hilft, das Verhältnis $\frac{\mathrm{FF}(I)}{\mathrm{OPT}(I)}$ abzuschätzen.

Zeige:

$$\sum_{i=1}^{n} \omega(s(a_i)) \leq \frac{17}{10} \cdot \mathrm{OPT}(I)$$

$$\sum_{i=1}^{n} \omega(s(a_i)) > \mathrm{FF}(I) - 1$$

Für:

$$\omega(a) = \begin{cases} \frac{6}{5} \cdot a & \text{für } 0 \leq a < \frac{1}{6} \\ \frac{9}{5} \cdot a - \frac{1}{10} & \text{für } \frac{1}{6} \leq a < \frac{1}{3} \\ \frac{6}{5} \cdot a + \frac{1}{10} & \text{für } \frac{1}{3} \leq a \leq \frac{1}{2} \\ 1 & \text{für } \frac{1}{2} < a \leq 1 \end{cases}$$

Hinweis:
Neues Ergebnis: es gilt sogar $\mathrm{FF}(I) \leq \frac{17}{10} \cdot \mathrm{OPT}(I)$

György Dósa und Jiri Sgall, *First Fit bin packing: A tight analysis*, STACS 2013

First Fit – Approximation

Gerade gesehen: Es gibt eine Instanz I, sodass $\mathcal{R}_{\text{FF}}(I) = \frac{17}{10}$.
Umgekehrt kann es kein viel schlechteres Beispiel geben:

Satz: Approximation **(Satz 7.8)**
Für jede Instanz I von BIN PACKING gilt: $\text{FF}(I) < \frac{17}{10} \cdot \text{OPT}(I) + 1$

Bemerkung: Asymptotische Approximation **(Bemerkung 7.9)**
Definiere die *asymptotische Gütegarantie* $\mathcal{R}_{\mathcal{A}}^{\infty}$ wie folgt:

$$\mathcal{R}_{\mathcal{A}}^{\infty} := \inf \{ r \geq 1 \mid \text{Es gibt } N > 0, \text{sodass } \mathcal{R}_{\mathcal{A}}(I) \leq r \text{ für alle } I \text{ mit } \text{OPT}(I) \geq N \}$$

Dann ist $\mathcal{R}_{\text{FF}}^{\infty} = \frac{17}{10}$ (der Summand 1 ist vernachlässigbar).

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

First Fit – Approximation

Gerade gesehen: Es gibt eine Instanz I, sodass $\mathcal{R}_{\text{FF}}(I) = \frac{17}{10}$.
Umgekehrt kann es kein viel schlechteres Beispiel geben:

Satz: Approximation **(Satz 7.8)**
Für jede Instanz I von BIN PACKING gilt: $\text{FF}(I) < \frac{17}{10} \cdot \text{OPT}(I) + 1$

Bemerkung: Asymptotische Approximation **(Bemerkung 7.9)**
Definiere die *asymptotische Gütegarantie* $\mathcal{R}_{\mathcal{A}}^{\infty}$ wie folgt:

$$\mathcal{R}_{\mathcal{A}}^{\infty} := \inf \{ r \geq 1 \mid \text{Es gibt } N > 0, \text{sodass } \mathcal{R}_{\mathcal{A}}(I) \leq r \text{ für alle } I \text{ mit } \text{OPT}(I) \geq N \}$$

Dann ist $\mathcal{R}_{\text{FF}}^{\infty} = \frac{17}{10}$ (der Summand 1 ist vernachlässigbar).

Es gibt weitere Strategien mit teilweise besseren Approximationsgüten.

Beispiel: FIRST FIT DECREASING (FFD) sortiert die Elemente zunächst absteigend und fügt sie dann mittels FF ein. $\rightarrow$ asymptotische Gütegarantie: $\frac{11}{9}$