

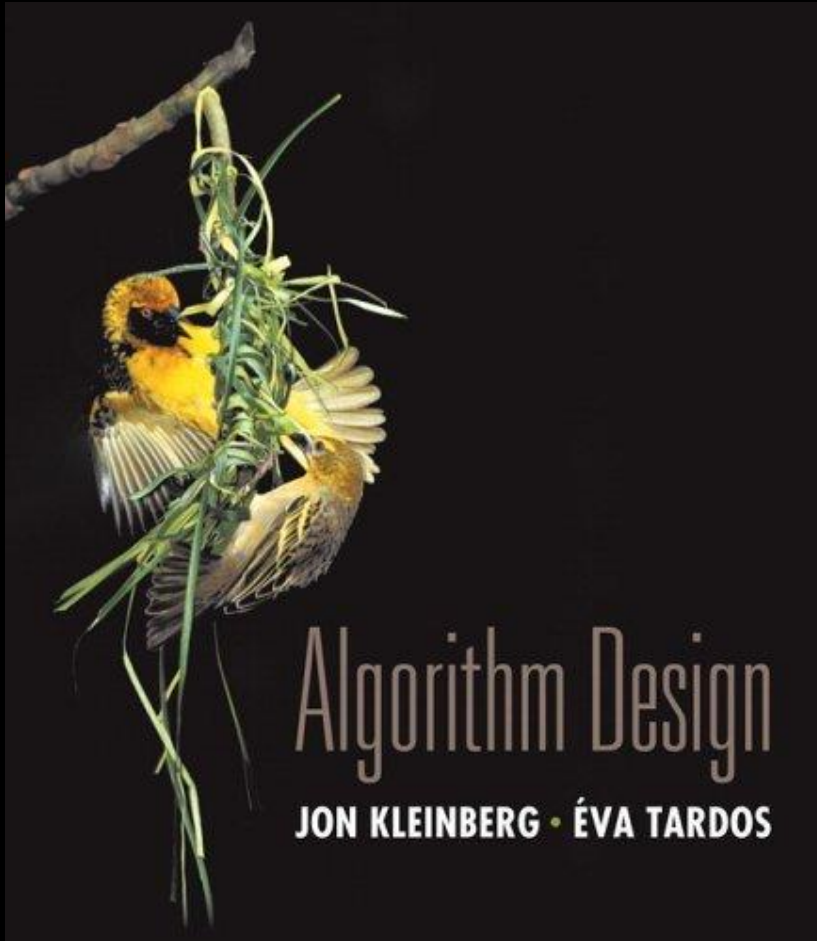
Chapter 11

Approximation Algorithms I

Die Folien basieren auf den Originalfolien von Kevin Wayne zum bekannten Lehrbuch von Jon Kleinberg und Eva Tardos.



Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.



Approximationsalgorithmen

Ein NP-schweres Problem ist zu bearbeiten. Was ist zu tun?

Man kann nicht erwarten, einen polynomiellen Algorithmus zu finden.

Eine von drei wünschenswerten Eigenschaften muss geopfert werden.

- Polynomielle Laufzeit
- Gültigkeit für sämtliche Instanzen
- **Exakte Lösung.**

Zentral ist der Begriff des ρ -Approximationsalgorithmus.

Man benutzt diesen Begriff, wenn folgende Eigenschaften vorliegen:

- Polynomielle Laufzeit
- Gültigkeit für sämtliche Instanzen
- Findet **Näherungslösung mit Gütegarantie ρ .**

Näherungslösung mit Gütegarantie

Für einen ρ -Approximationsalgorithmus soll demnach gelten:

- Polynomielle Laufzeit
- Gültigkeit für sämtliche Instanzen
- Findet **Näherungslösung mit Gütegarantie ρ** .

Eine Herausforderung, um die es dabei geht: Für eine Näherungslösung ist nachzuweisen, dass sie nahe bei der Optimallösung liegt, obwohl die Optimallösung unbekannt ist.

Im Folgenden wird erläutert, was unter einer Näherungslösung mit Gütegarantie ρ zu verstehen ist.

Näherungslösung mit Gütegarantie

Es liege ein **Minimierungsproblem** vor und $\rho \geq 1$ sei eine reelle Zahl.

Mit L^* sei der (unbekannte) optimale Zielfunktionswert bezeichnet.

A sei ein Algorithmus für das vorliegende Problem. Der Wert der von A gelieferten Lösung sei mit L_A bezeichnet.

Definition. Man sagt, dass A eine **Näherungslösung mit Gütegarantie ρ** findet, wenn für den Quotienten von L_A und L^* gilt:

$$L_A / L^* \leq \rho.$$

Beispiel: $\rho = 2$. Dann gilt $L_A / L^* \leq 2$ bzw. $L_A \leq 2 L^*$. Das bedeutet: Der Wert der gelieferten Lösung ist niemals schlechter als $2 L^*$.

Analoge Sprechweisen verwendet man für **Maximierungsprobleme**.

11.1 Das Lastverteilungsproblem

Das Lastverteilungsproblem

Eingabe. m identische Maschinen; n Jobs; Job j besitzt die Ausführungszeit t_j .

- Job j ist von einer einzigen Maschine ohne Unterbrechung in einem beliebigen Zeitintervall der Länge t_j zu bearbeiten.
- Jede Maschine kann nur einen Job zur Zeit erledigen.

Def. Mit $J(i)$ sei die Menge der Jobs bezeichnet, die Maschine i zugeordnet werden. Die **Last** der Maschine i : $L_i = \sum_{j \in J(i)} t_j$.

Def. Die **Bearbeitungsdauer** oder **Produktionsspanne** (englisch: **makespan**) ist definiert durch $L = \max_i L_i$.

Problem: **Lastverteilung (LOAD BALANCING).** Man ordne jedem Job eine Maschine zu, wobei die Produktionsspanne L zu minimieren ist.

LOAD BALANCING ist NP-schwer

LOAD BALANCING. Man ordne jedem Job eine Maschine zu, wobei die Produktionsspanne ("makespan") L zu minimieren ist.

Behauptung. LOAD BALANCING ist sogar dann NP-schwer, wenn man sich auf den Fall von zwei Maschinen beschränkt.

Beweis: wird in den Übungen behandelt.

Ein Greedy-Algorithmus für LOAD BALANCING

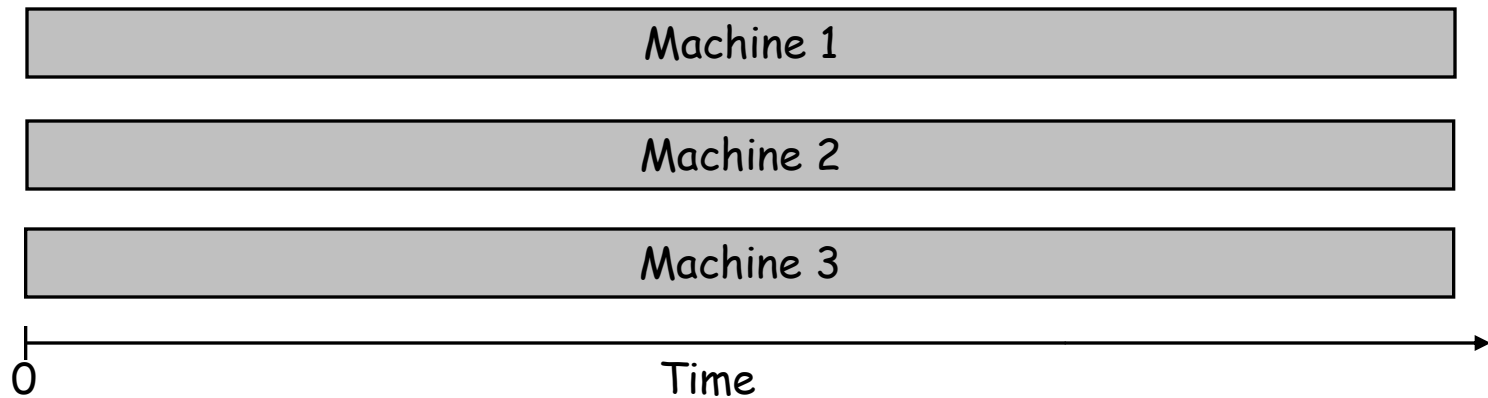
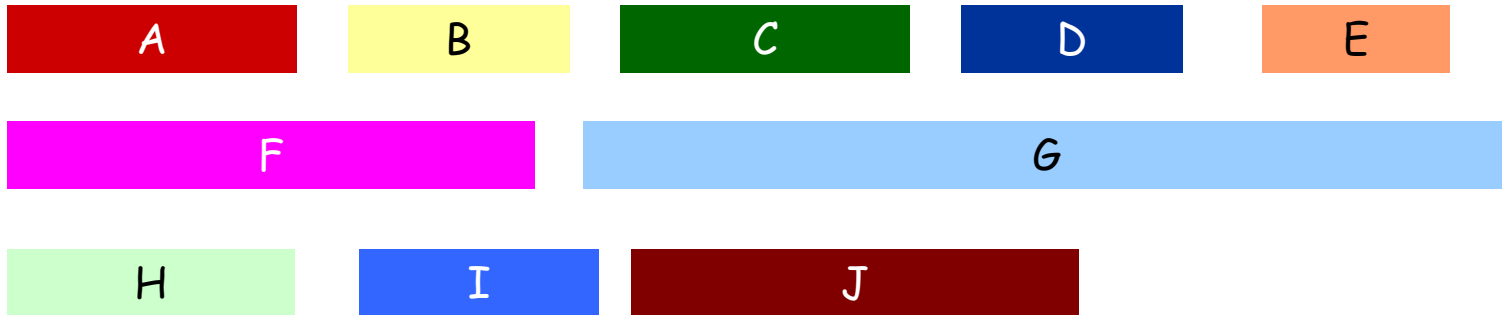
Algorithmus **Greedy-Balance** (auch **List Scheduling** genannt).

- Man betrachte die Jobs in beliebiger Reihenfolge.
- Job j wird der Maschine mit aktuell kleinster Last zugeordnet.

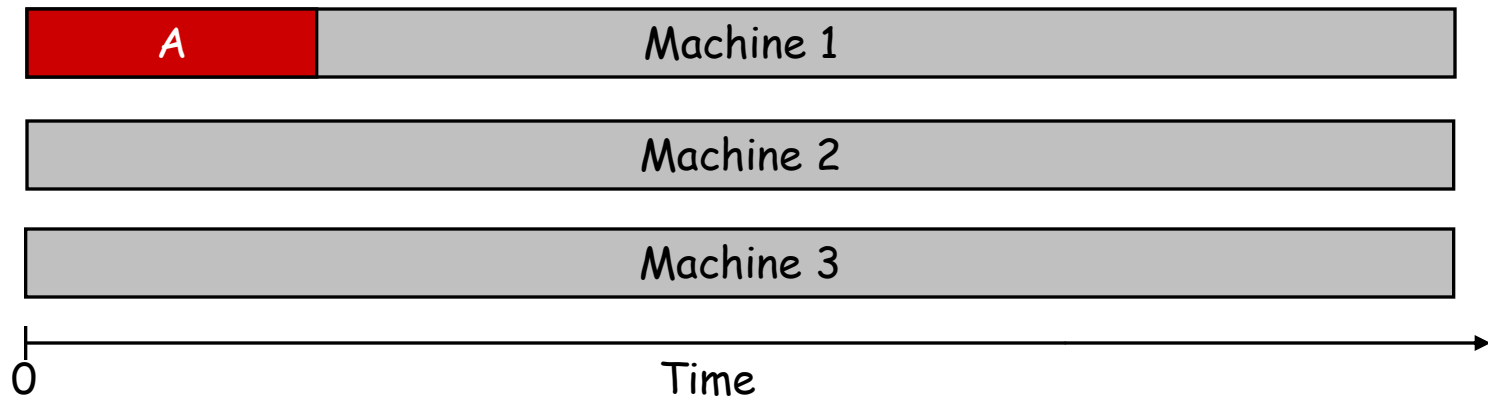
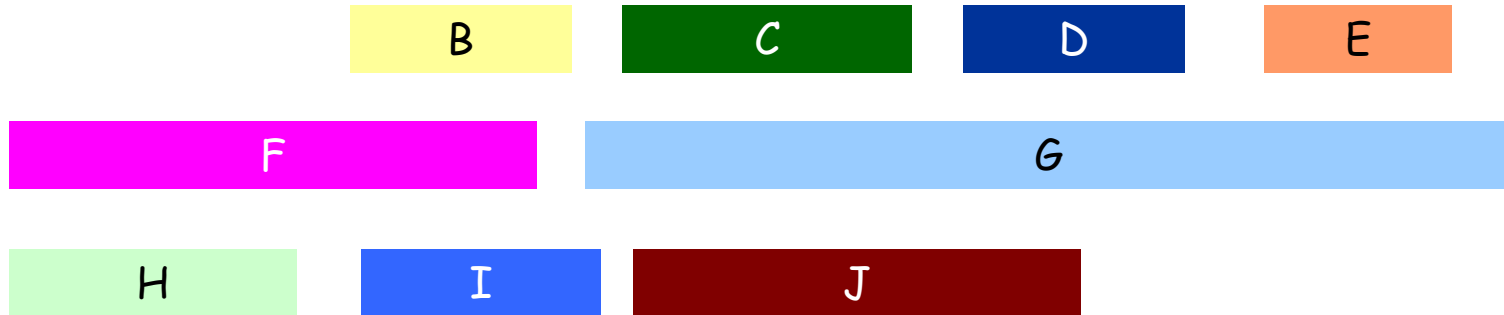
```
Greedy-Balance(m, n, t1, t2, ..., tn) {  
  for i = 1 to m {  
    Li ← 0          ← Last für Maschine i  
    J(i) ←  $\phi$        ← Jobs für Maschine i  
  }  
  
  for j = 1 to n {  
    i = argmink Lk    ← Maschine i hat kleinste Last  
    J(i) ← J(i)  $\cup$  {j} ← Job j geht an Maschine i  
    Li ← Li + tj    ← Update der Last  
  }  
  return J(1), ..., J(m)  
}
```

Implementierung. $O(n \log m)$ mittels Prioritätswarteschlange.

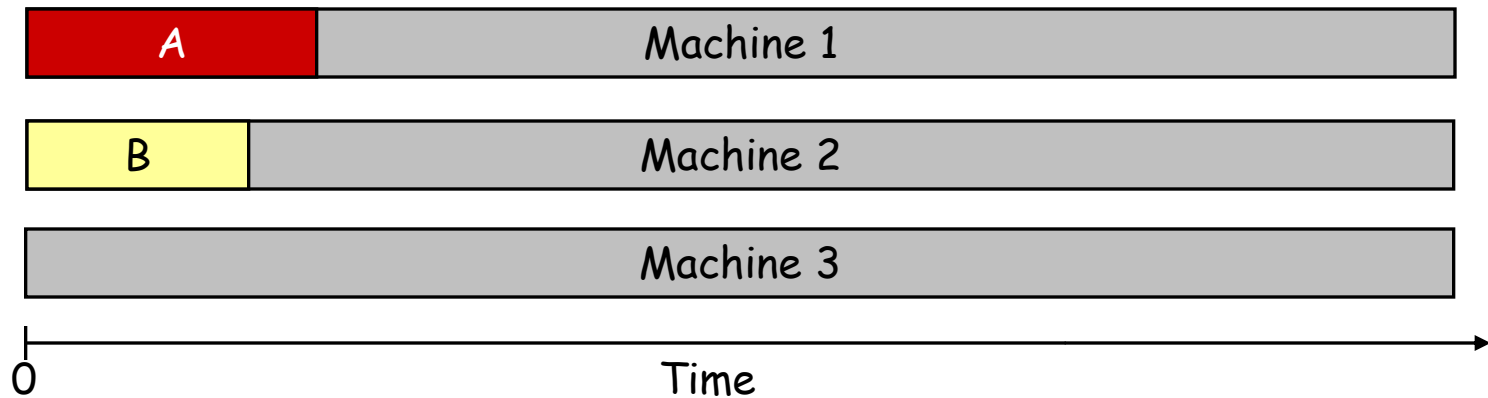
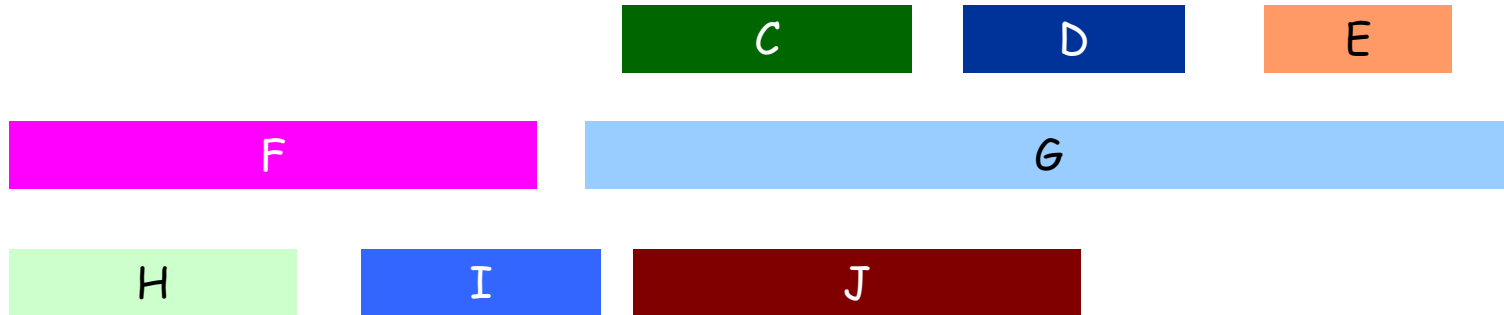
Load Balancing: List Scheduling



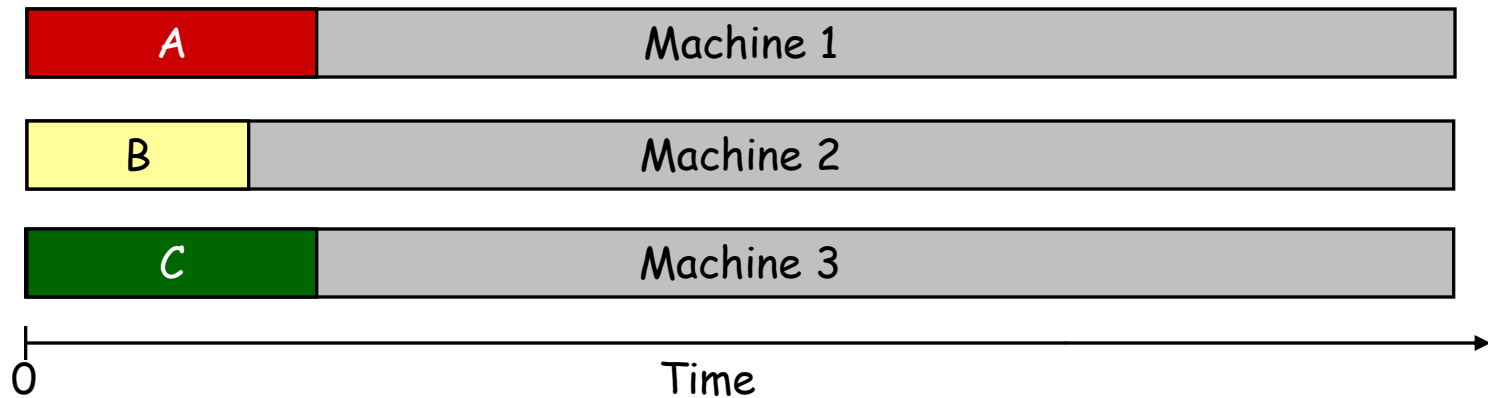
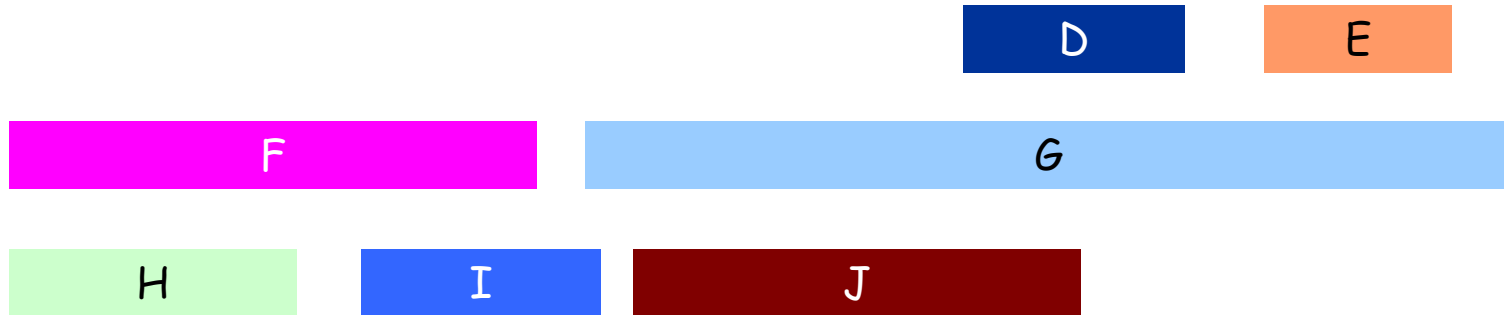
Load Balancing: List Scheduling



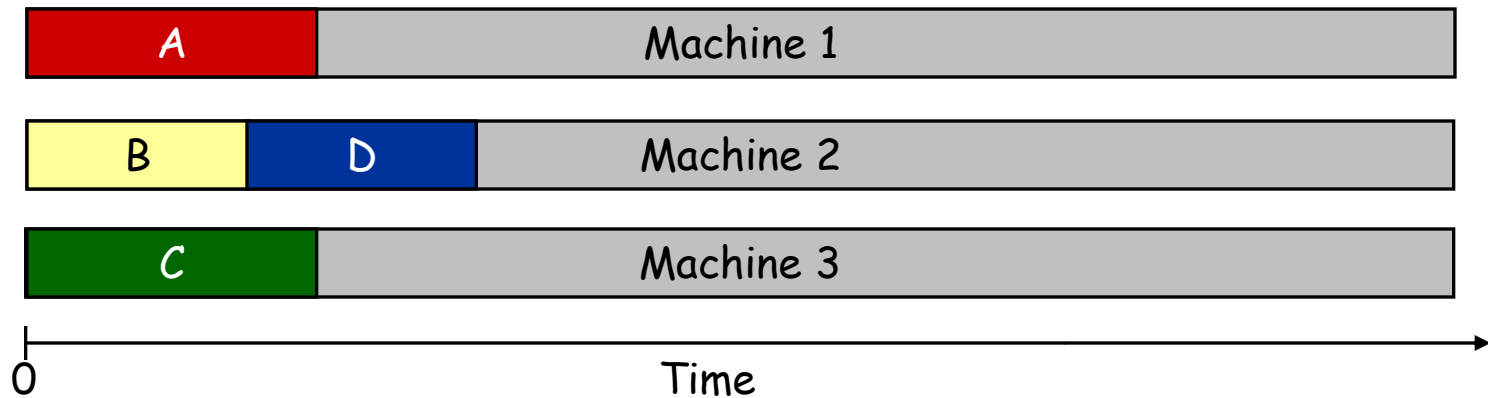
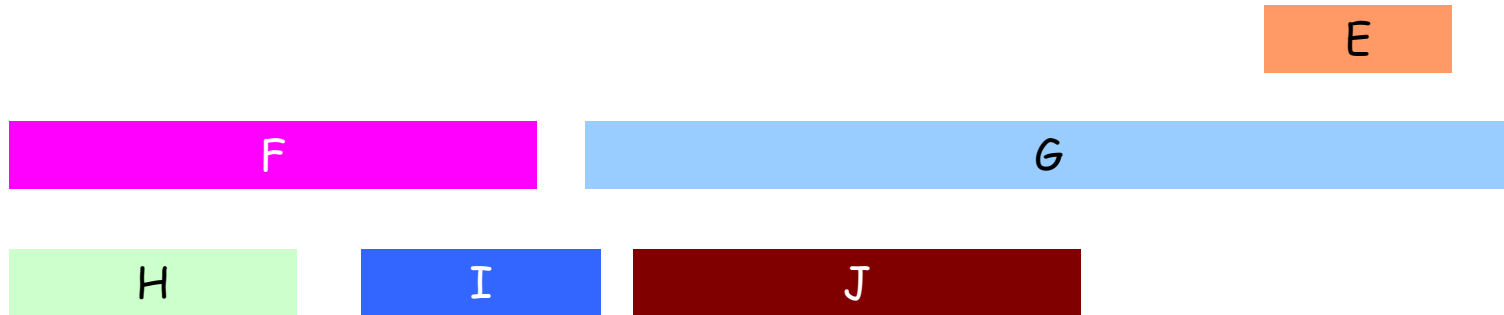
Load Balancing: List Scheduling



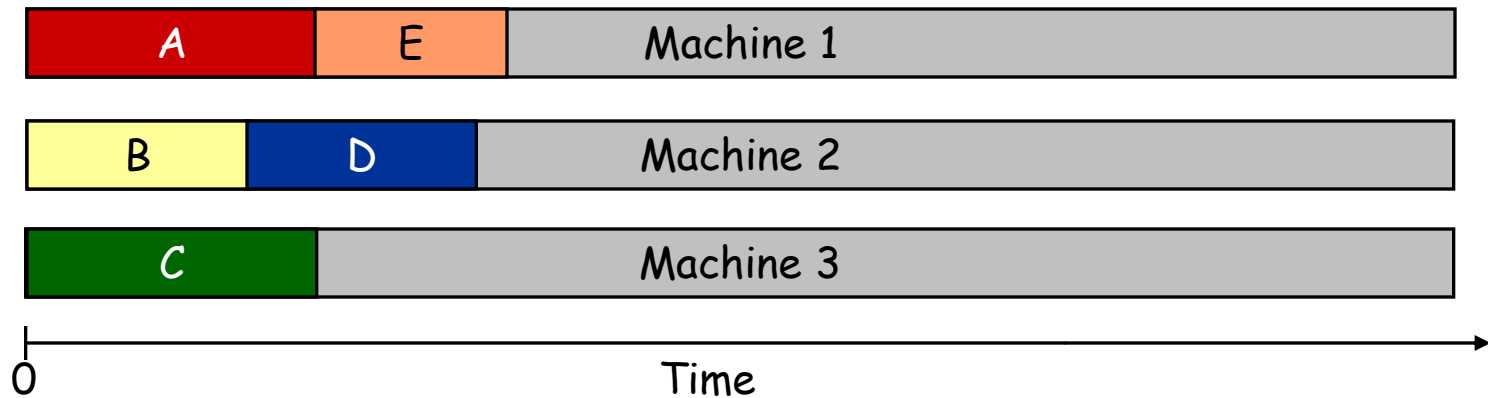
Load Balancing: List Scheduling



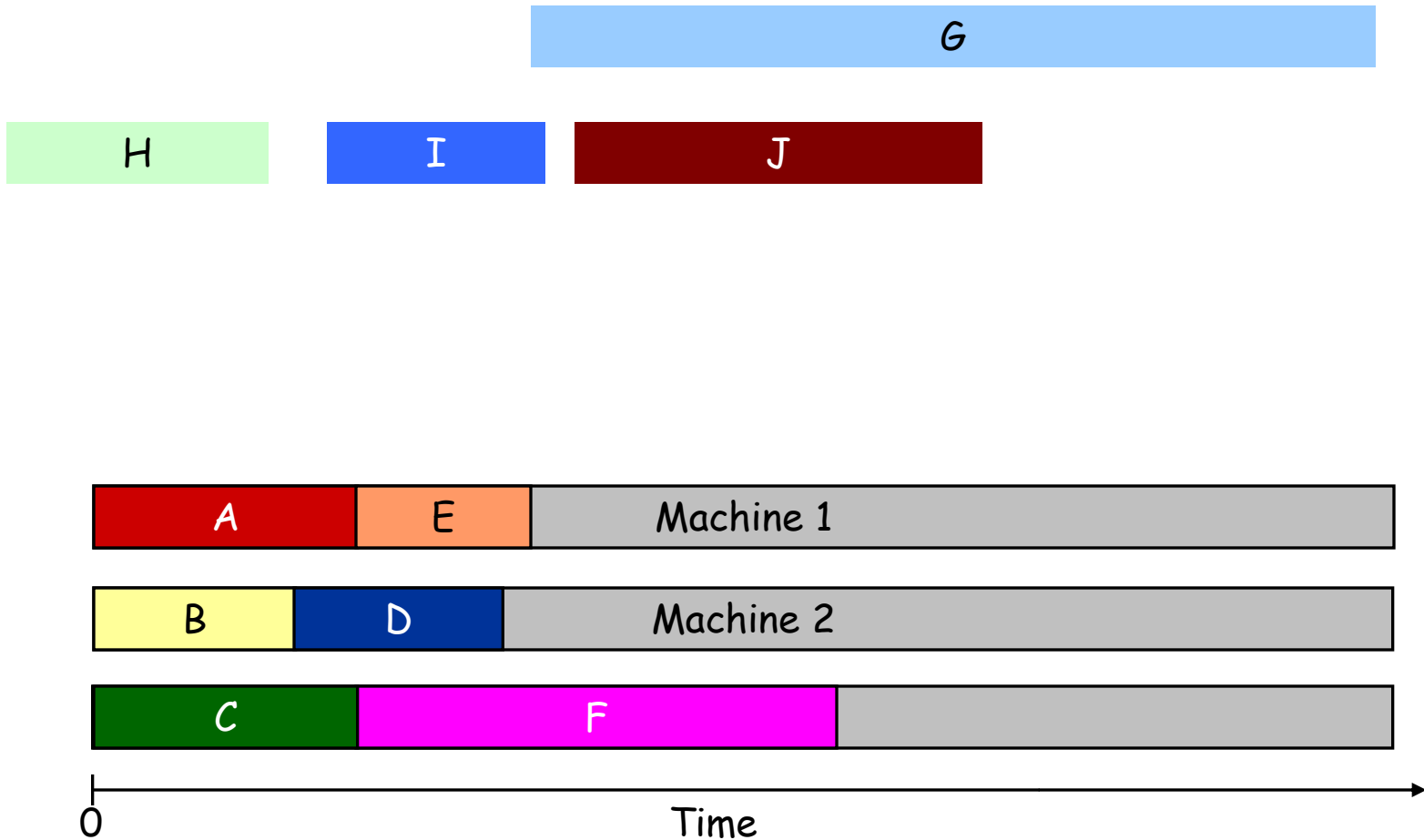
Load Balancing: List Scheduling



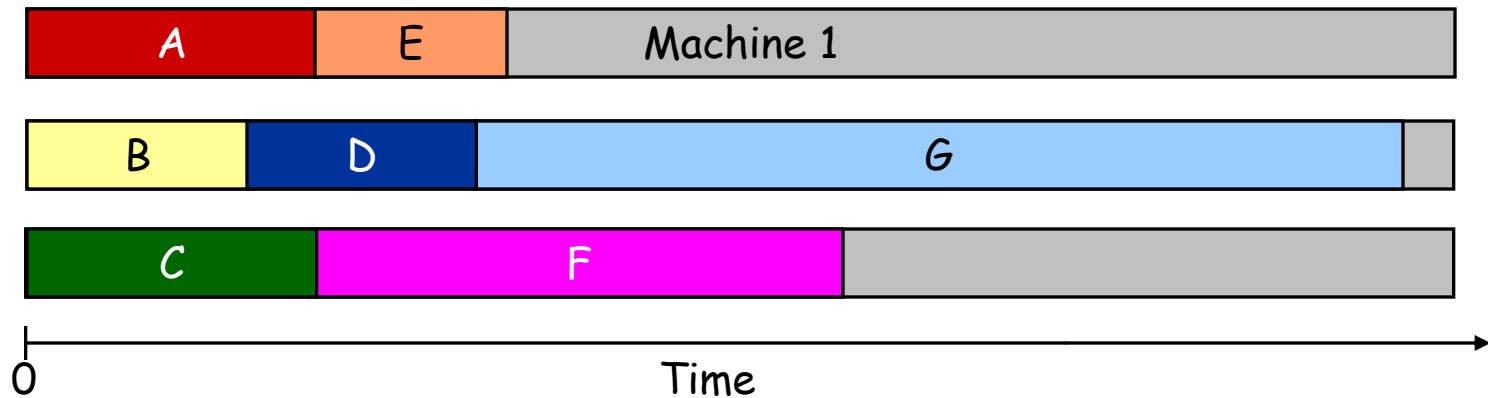
Load Balancing: List Scheduling



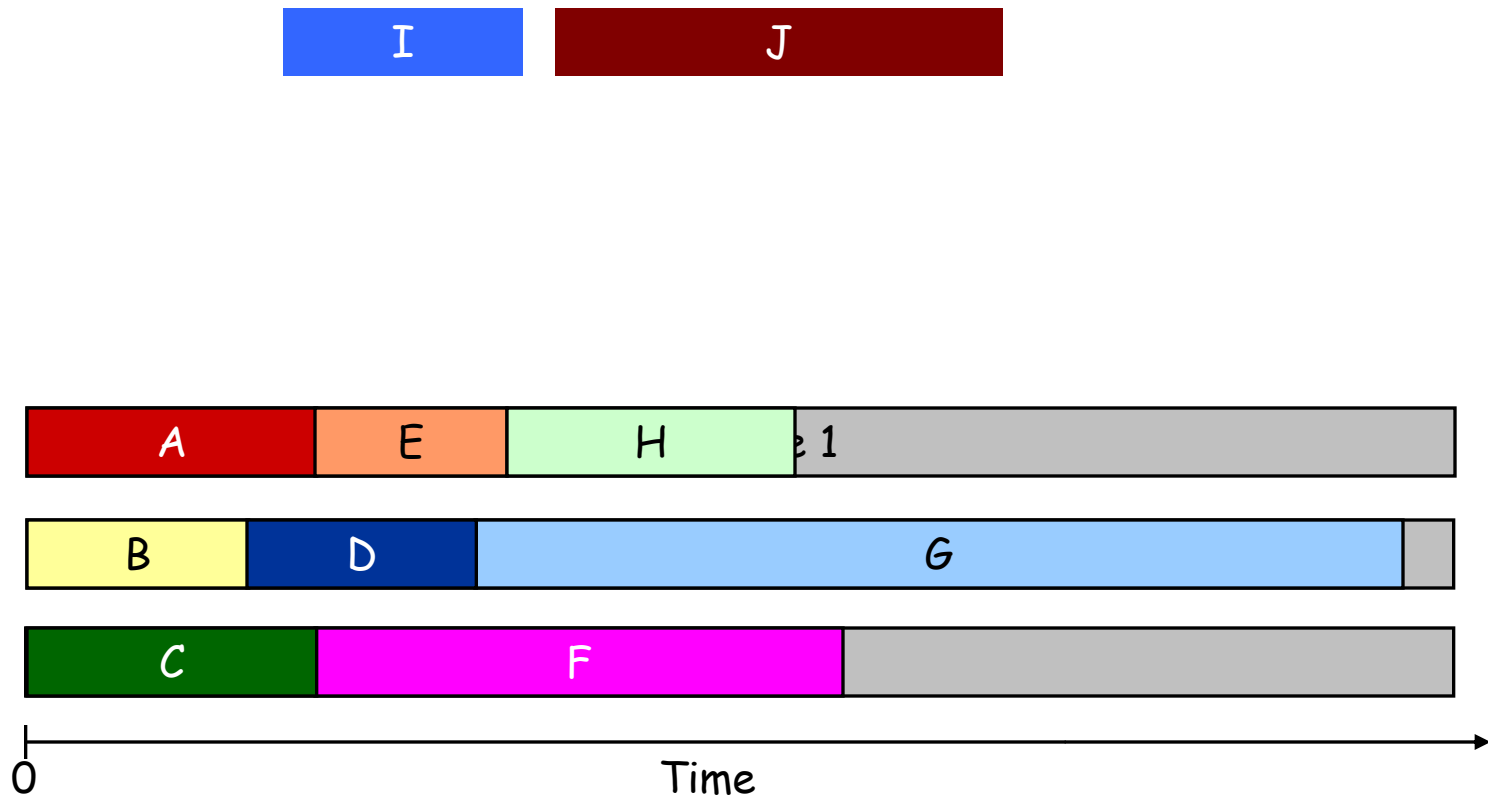
Load Balancing: List Scheduling



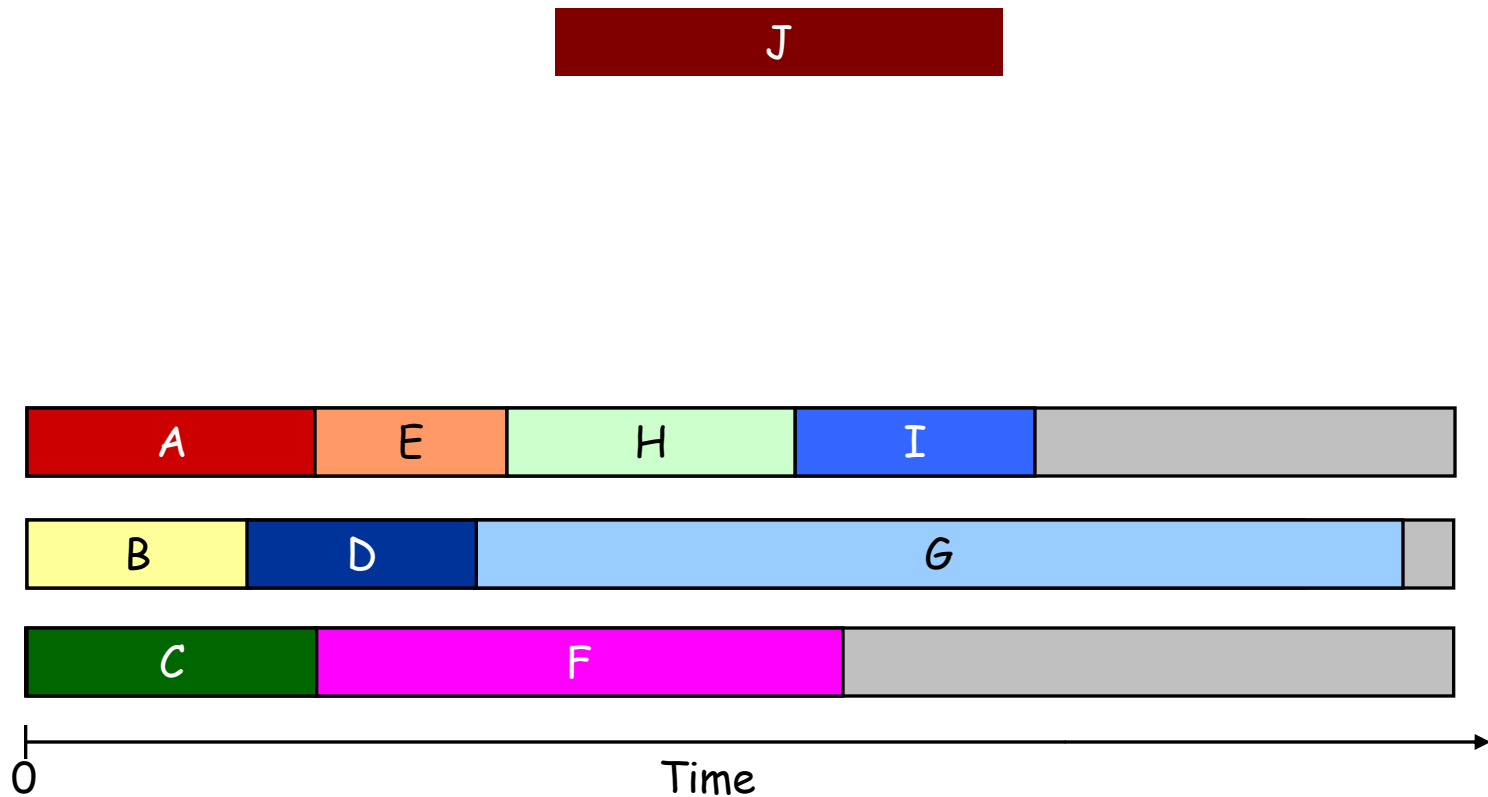
Load Balancing: List Scheduling



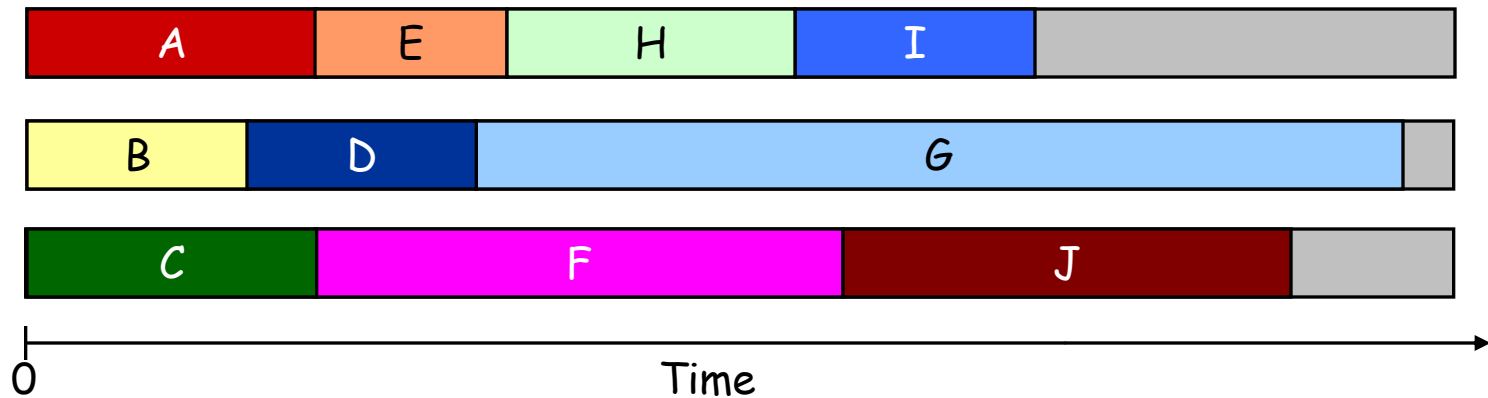
Load Balancing: List Scheduling



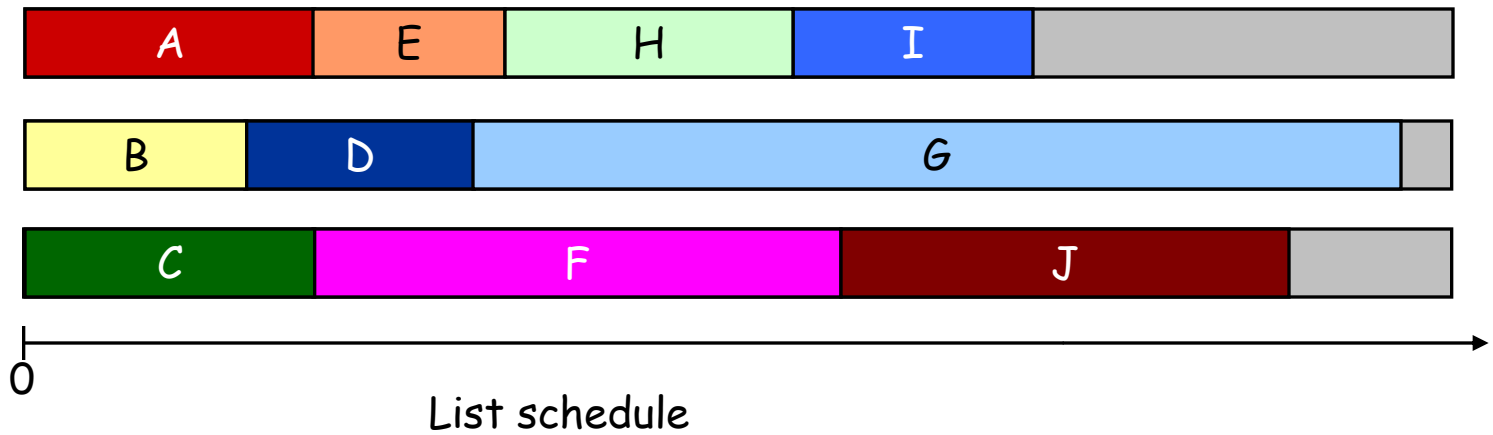
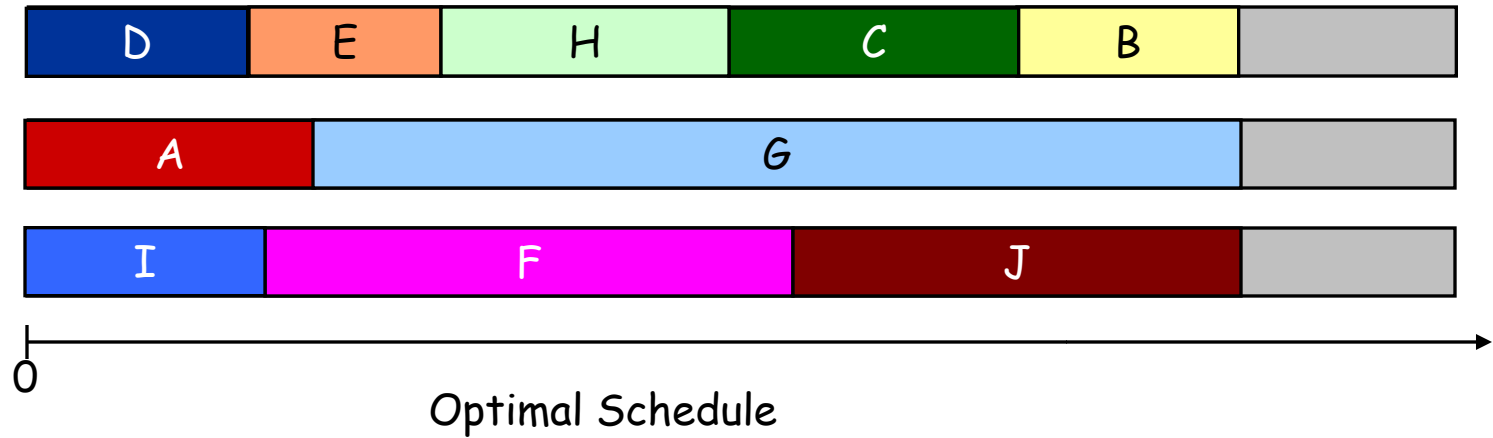
Load Balancing: List Scheduling



Load Balancing: List Scheduling



Load Balancing: List Scheduling



Analyse des Algorithmus Greedy-Balance

Theorem. [Graham, 1966] Greedy-Balance ist ein 2-Approximations - algorithmus.

- Pionierarbeit von Graham.
- Die Produktionsspanne L , die der Algorithmus liefert, ist mit der (unbekannten) optimalen Produktionsspanne L^* zu vergleichen.

Lemma 1. Für L^* gilt $L^* \geq \max_j t_j$.

Beweis. Eine der Maschinen muss den längsten Job übernehmen. ▪

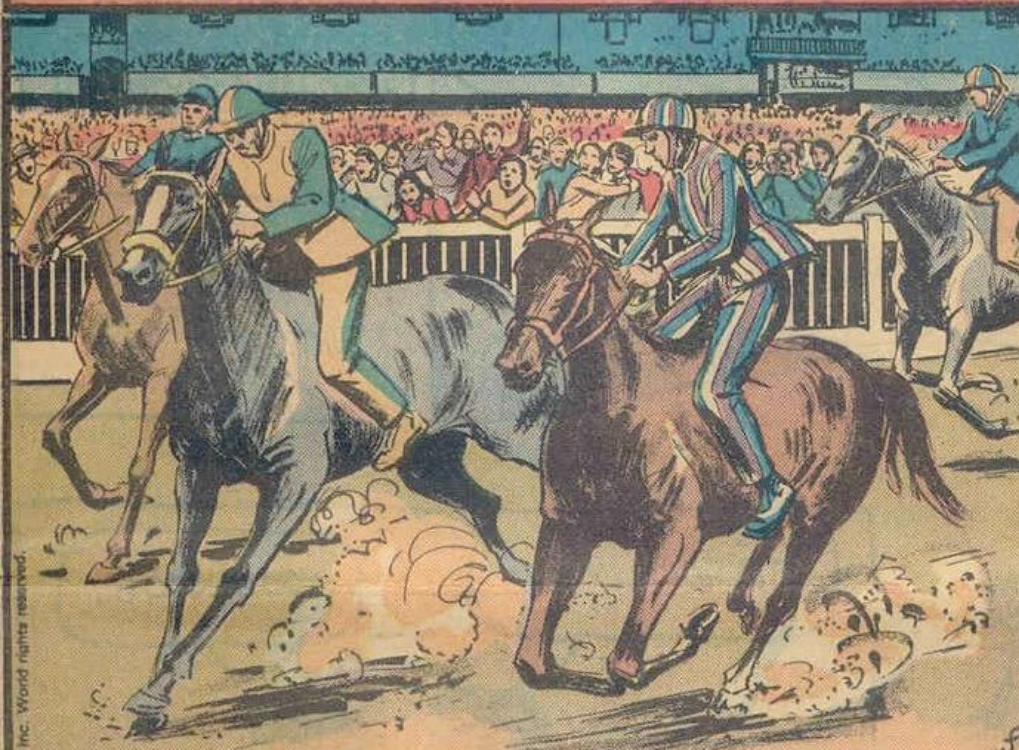
Lemma 2. Für L^* gilt $L^* \geq \frac{1}{m} \sum_j t_j$.

Beweis.

- Man summiert sämtliche Joblängen: $\sum_j t_j$.
- Wenigstens eine der m Maschinen muss mindestens $1/m$ dieser Summe (Gesamtlast) übernehmen. ▪


Believe it or not!

Ripley's **Believe It or Not!**



A RACE IN WHICH LOSING IS AKIN TO DEATH

THE PALIO, a horse race held each summer around the main square of Siena, Italy, traditionally ends with the winners holding a **MOCK FUNERAL FOR THE LOSERS**



RONALD GRAHAM
head of Bell Laboratories mathematical Studies Center in Murray Hill, N.J., is one of the world's foremost mathematicians, publishes more than 12 math papers a year and is on the editorial boards of 20 math journals — yet is a highly skilled trampolinist and juggler, and has been elected president of the International Jugglers Association

© 1983 King Features Syndicate, Inc. World rights reserved.

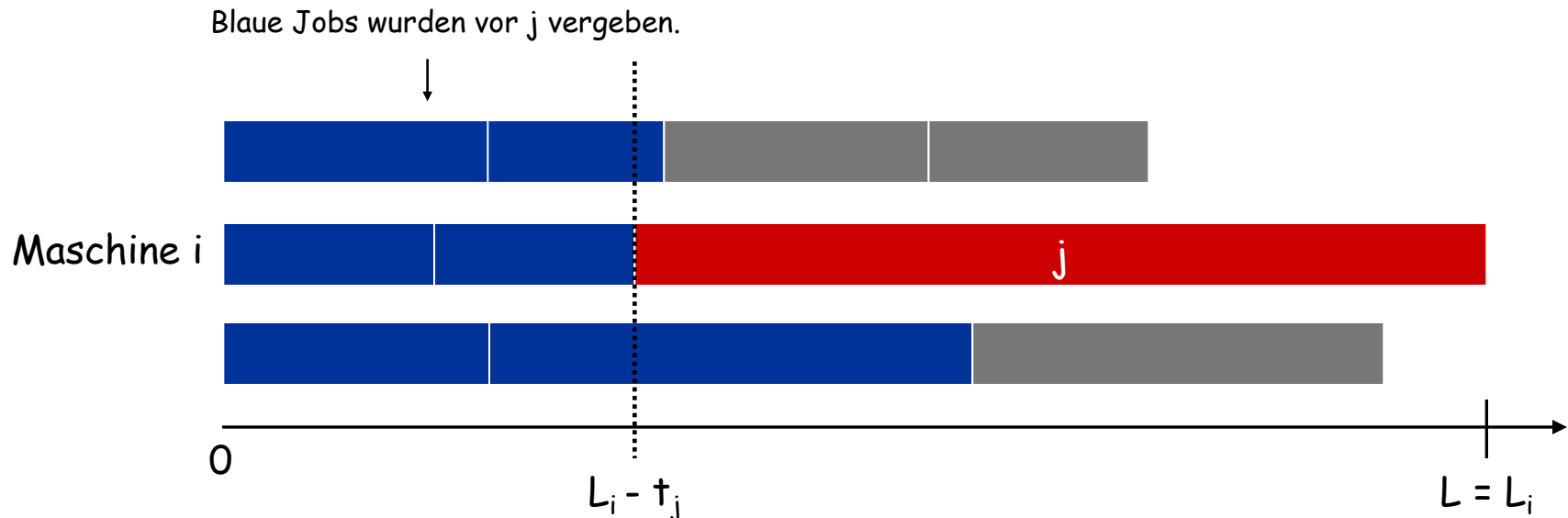
4-10

Analyse des Algorithmus Greedy-Balance

Theorem. Greedy-Balance ist ein 2-Approximationsalgorithmus.

Beweis. Es sei i die Maschine, die am Schluss die maximale Last L_i hat.

- Es sei j der Job, der i als letzter zugeteilt wurde.
- Direkt bevor Job j der Maschine i zugeteilt wurde, hatte i die geringste Last. Die Last von i zu diesem Zeitpunkt beträgt $L_i - t_j$. Es folgt $L_i - t_j \leq L_k$ für alle $1 \leq k \leq m$.



Analyse des Algorithmus Greedy-Balance

Theorem. Greedy-Balance ist ein 2-Approximationsalgorithmus.

Beweis. Es sei i die Maschine, die am Schluss die maximale Last L_i hat.

- Es sei j der Job, der i als letzter zugeteilt wurde.
- Direkt bevor Job j der Maschine i zugeteilt wurde, hatte i die geringste Last. Die Last von i zu diesem Zeitpunkt beträgt $L_i - t_j$. Es folgt $L_i - t_j \leq L_k$ für alle $1 \leq k \leq m$.
- Aufsummieren der m Ungleichungen und Division durch m ergibt:

$$L_i - t_j \leq (1/m) \sum_k L_k = (1/m) \sum_j t_j \leq L^*$$

↑
Lemma 2

▪ Es folgt

$$L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\substack{\leq L^* \\ \uparrow \\ \text{Lemma 1}}} \leq 2L^*.$$

↑
Lemma 1

Analyse des Algorithmus Greedy-Balance

Frage. Ist der Faktor 2 der bestmögliche konstante Gütegarantiefaktor für Greedy-Balance?

Oder lässt sich **durch eine bessere Analyse des Algorithmus Greedy-Balance** zeigen, dass Greedy-Balance sogar ein ρ -Approximationsalgorithmus für $\rho < 2$ ist?

Antwort. Für alle $\rho < 2$ gilt: Der Algorithmus Greedy-Balance ist **kein** ρ -Approximationsalgorithmus.

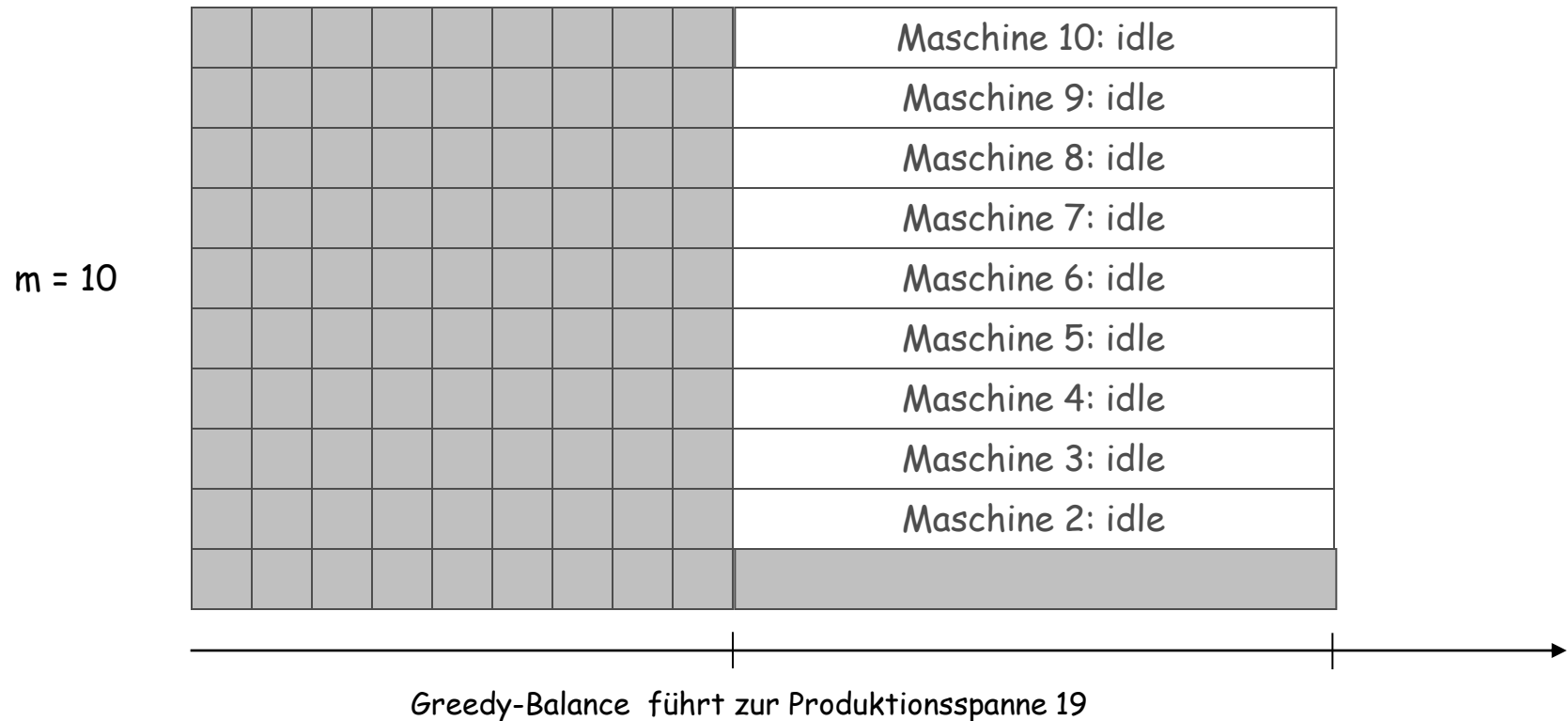
Man weist dies nach, indem man für jedes $m > 1$ ein **Beispiel** mit m Maschinen angibt, für das gilt:

- Optimale Produktionsspanne : m .
- Greedy-Balance liefert eine Lösung mit Produktionsspanne $2m - 1$.

Wir führen dies im Folgenden für **$m = 10$** vor.

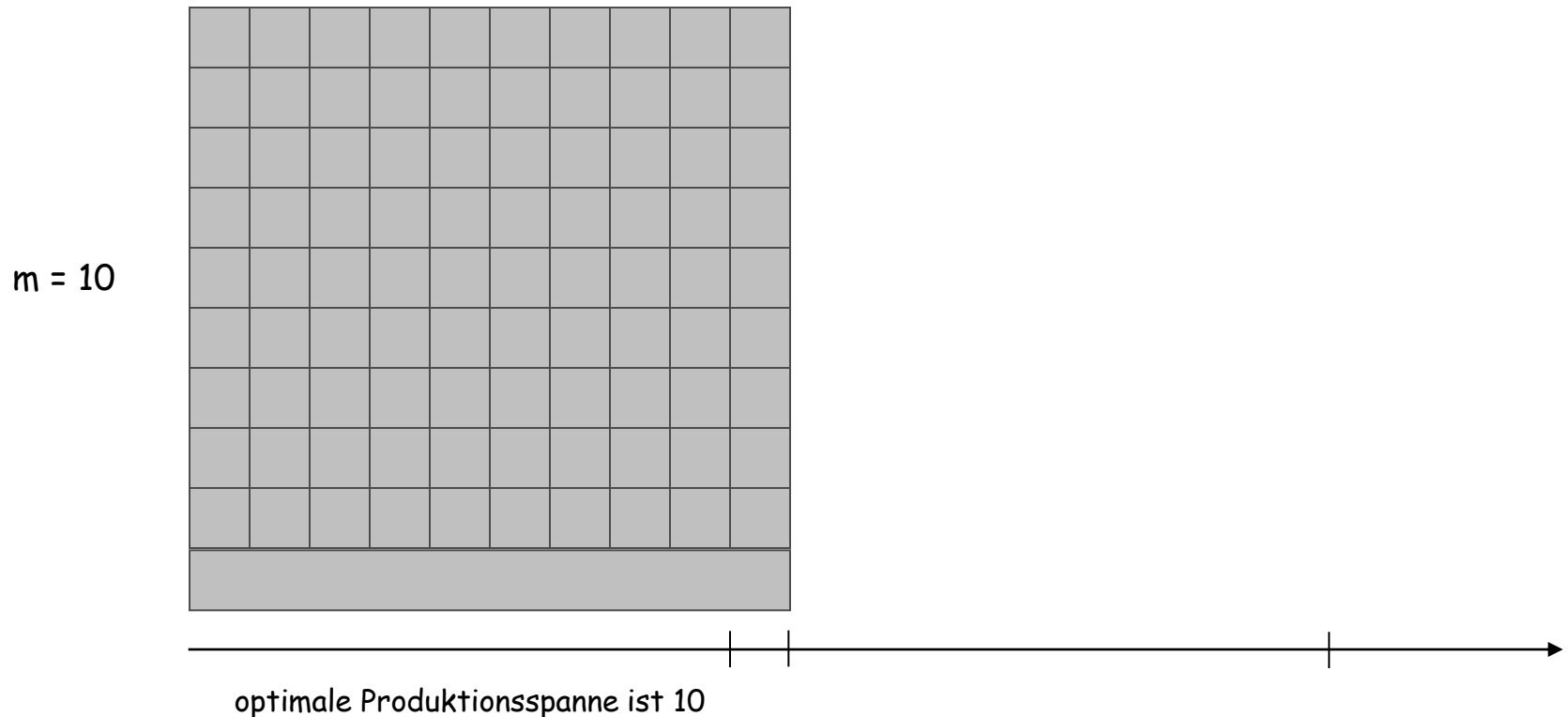
Analyse des Algorithmus Greedy-Balance

Beispiel: m Maschinen, $m(m-1)$ Jobs der Länge 1 und ein weiterer Job der Länge m , der in der betrachteten Reihenfolge der letzte ist.



Analyse des Algorithmus Greedy-Balance

Beispiel: Wie zuvor m Maschinen, $m(m-1)$ Jobs der Länge 1, ein Job der Länge m . Das "schlechte Beispiel" kam dadurch zustande, dass der Job der Länge m der letzte war. Nun sei dies der erste Job. Ergebnis:



Analyse des Algorithmus Greedy-Balance

Durch eine leichte Modifikation des obigen Beweises lässt sich zeigen:

Theorem. Für jede Eingabe von LOAD BALANCING mit m Maschinen gilt: Ist L die von Greedy-Balance gelieferte Produktionsspanne und L^* die optimale Produktionsspanne, so ist folgende Ungleichung erfüllt:

$$L \leq (2 - 1/m) L^*.$$

Als Übungsaufgabe empfohlen: Schauen Sie sich den Beweis auf den Folien 23/24 genau an und finden Sie heraus, was man verändern kann, so dass sich anstelle der Ungleichung $L \leq 2L^*$ die obige ein wenig verbesserte Ungleichung ergibt.

Fazit: Die Ungleichung $L \leq 2L^*$ lässt sich etwas verbessern, nämlich zu $L \leq (2 - 1/m) L^*$. Die Beispiele zeigen: Für Eingaben mit m Maschinen ist diese leicht verbesserte Ungleichung bestmöglich.

Greedy-Balance mit LIF-Regel

Longest Intervals First (LIF). Sortiere die gegebenen Jobs, so dass $t_1 \geq t_2 \geq \dots \geq t_n$ gilt. Danach: Führe Greedy-Balance wie zuvor aus.

```
Greedy-Balance mit LIF-Regel (m, n,  $t_1, t_2, \dots, t_n$ ) {  
    Sort jobs so that  $t_1 \geq t_2 \geq \dots \geq t_n$   
  
    for i = 1 to m {  
         $L_i \leftarrow 0$             $\leftarrow$  Last für Maschine i  
         $J(i) \leftarrow \phi$         $\leftarrow$  Jobs für Maschine i  
    }  
  
    for j = 1 to n {  
         $i = \operatorname{argmin}_k L_k$       $\leftarrow$  Maschine i hat kleinste Last  
         $J(i) \leftarrow J(i) \cup \{j\}$   $\leftarrow$  Job j geht an Maschine i  
         $L_i \leftarrow L_i + t_j$      $\leftarrow$  Update der Last  
    }  
    return  $J(1), \dots, J(m)$   
}
```

Analyse von Greedy-Balance mit LIF-Regel

Lemma 3. Es gebe mehr als m Jobs und es werde die LIF-Regel verwendet. Dann gilt $L^* \geq 2 t_{m+1}$.

Beweis. Wir betrachten die ersten $m+1$ Jobs. Da die t_i 's absteigend geordnet sind, gilt $t_i \geq t_{m+1}$ für alle $i \leq m+1$. Nach dem Schubfachprinzip gibt es eine Maschine, die zwei der ersten $m+1$ Jobs abbekommt. ▀

Theorem. Wird die LIF-Regel verwendet, so liegt ein $3/2$ -Approximationsalgorithmus vor.

Beweis. Es sei i die Maschine, die am Schluss die maximale Last L_i hat.

Falls i nur einen einzigen Job abbekommen hat, so ist die Zuordnung optimal. Wir dürfen also annehmen, dass i mehr als einen Job erhalten hat.

Analyse von Greedy-Balance mit LIF-Regel

Es sei j der Job, der i als letzter zugeteilt wurde.

Da die ersten m Jobs auf unterschiedliche Maschinen aufgeteilt werden, gilt also $j \geq m+1$. Insbesondere gibt es mehr als m Jobs, weshalb wir Lemma 3 anwenden können.

Es geht nun ebenso weiter wie zuvor im Beweis, dass Greedy-Balance ohne LIF ein 2-Approximationsalgorithmus ist. Am Schluss ergibt sich dann aufgrund von **Lemma 3** eine verbesserte Abschätzung:

$$L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\substack{\leq \frac{1}{2}L^* \\ \uparrow \\ \text{aufgrund von } j \geq m+1 \text{ und Lemma 3.}}} \leq \frac{3}{2}L^*. \quad \blacksquare$$

Analyse von Greedy-Balance mit LIF-Regel

Frage. Ist der Faktor $3/2$ für Greedy-Balance mit LIF-Regel bestmöglich?

Antwort: Nein.

Theorem. [Graham, 1969] Greedy-Balance mit LIF-Regel ist ein $4/3$ -Approximationsalgorithmus.

Beweis. Durch eine raffiniertere Analyse desselben Algorithmus.

Frage. Ist Grahams Faktor $4/3$ für Greedy-Balance mit LIF-Regel bestmöglich?

Antwort: Ja. Für alle $\rho < 4/3$ gilt: Der Algorithmus Greedy-Balance mit LIF-Regel ist **kein** ρ -Approximationsalgorithmus.

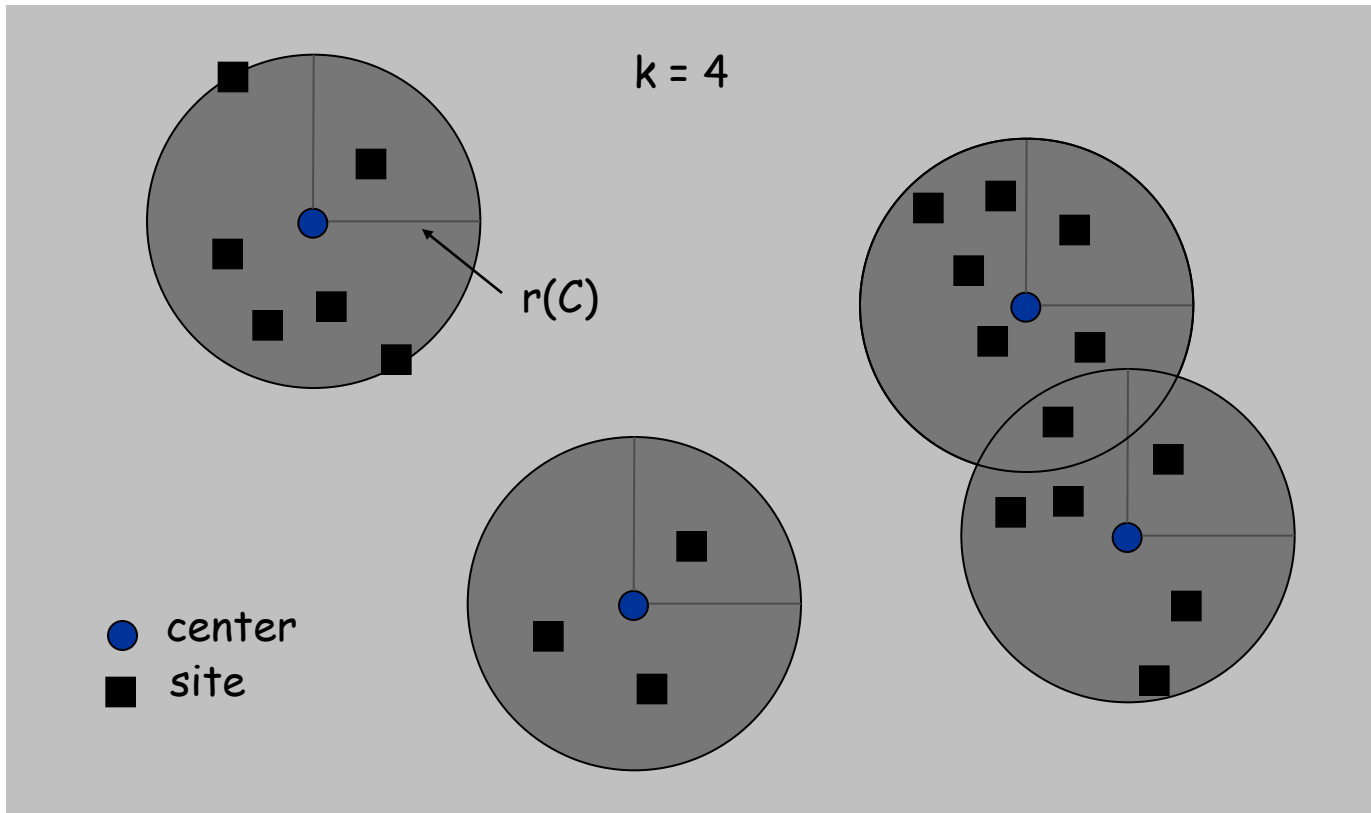
Beispiel: m Maschinen, $n = 2m+1$ Jobs. Je zwei Jobs der Längen $m, m+1, m+2, \dots, 2m-1$ und zusätzlich ein weiterer Job der Länge m .

11.2 Das Zentrenauswahlproblem

Standorte und Zentren

Gegeben: Punkte s_1, \dots, s_n in der Ebene (Standorte) sowie eine ganze Zahl k mit $n > k > 0$.

Finde eine Menge C von k Punkten in der Ebene (Zentren), so dass die maximale Distanz der Standorte zum jeweils nächsten Zentrum so klein wie möglich ist.



Ein allgemeinerer Standpunkt

Im Beispiel auf der vorangegangenen Folie waren die Standorte und Zentren Punkte in der Ebene.

Mit "Distanz" war der Euklidische Abstand gemeint.

Im Folgenden nehmen wir einen allgemeineren Standpunkt ein:

Wir betrachten eine beliebige Menge U , für die eine **Abstandsfunktion** gegeben ist.

Der Begriff der Abstandsfunktion

Gegeben sei eine beliebige Menge U zusammen mit einer **Abstandsfunktion** auf U , d.h. einer Funktion, die jedem Paar (x, y) mit $x, y \in U$ eine reelle Zahl $\text{dist}(x, y) \geq 0$ zuordnet, wobei die folgenden drei Eigenschaften erfüllt sein sollen.

Eigenschaften einer Abstandsfunktion.

- $\text{dist}(x, x) = 0$ (Identität)
- $\text{dist}(x, y) = \text{dist}(y, x)$ (Symmetrie)
- $\text{dist}(x, y) \leq \text{dist}(x, z) + \text{dist}(z, y)$ (Dreiecksungleichung)

Hinweis. Es könnte in anderen Zusammenhängen auch sinnvoll sein, auf die zweite Forderung (Symmetrie) zu verzichten und somit auch asymmetrische Abstandsfunktionen zuzulassen. Ebenfalls könnte man auf die Dreiecksungleichung verzichten. **Hier soll dergleichen nicht geschehen:** Es werden alle drei Forderungen an eine Abstandsfunktion vorausgesetzt (und auch benötigt).

Das Zentrenauswahlproblem (Center selection problem)

Das folgende Optimierungsproblem bezeichnen wir als **Center selection problem**: Gegeben sei eine Menge U zusammen mit einer Abstandsfunktion $\text{dist}(x, y)$. Außerdem seien Standorte $s_1, \dots, s_n \in U$ und eine ganze Zahl k mit $n > k > 0$ gegeben. Zu finden ist eine Menge $C \subseteq U$ von k Zentren, so dass die maximale Distanz der Standorte zum jeweils nächsten Zentrum minimal ist.

Bezeichnungen:

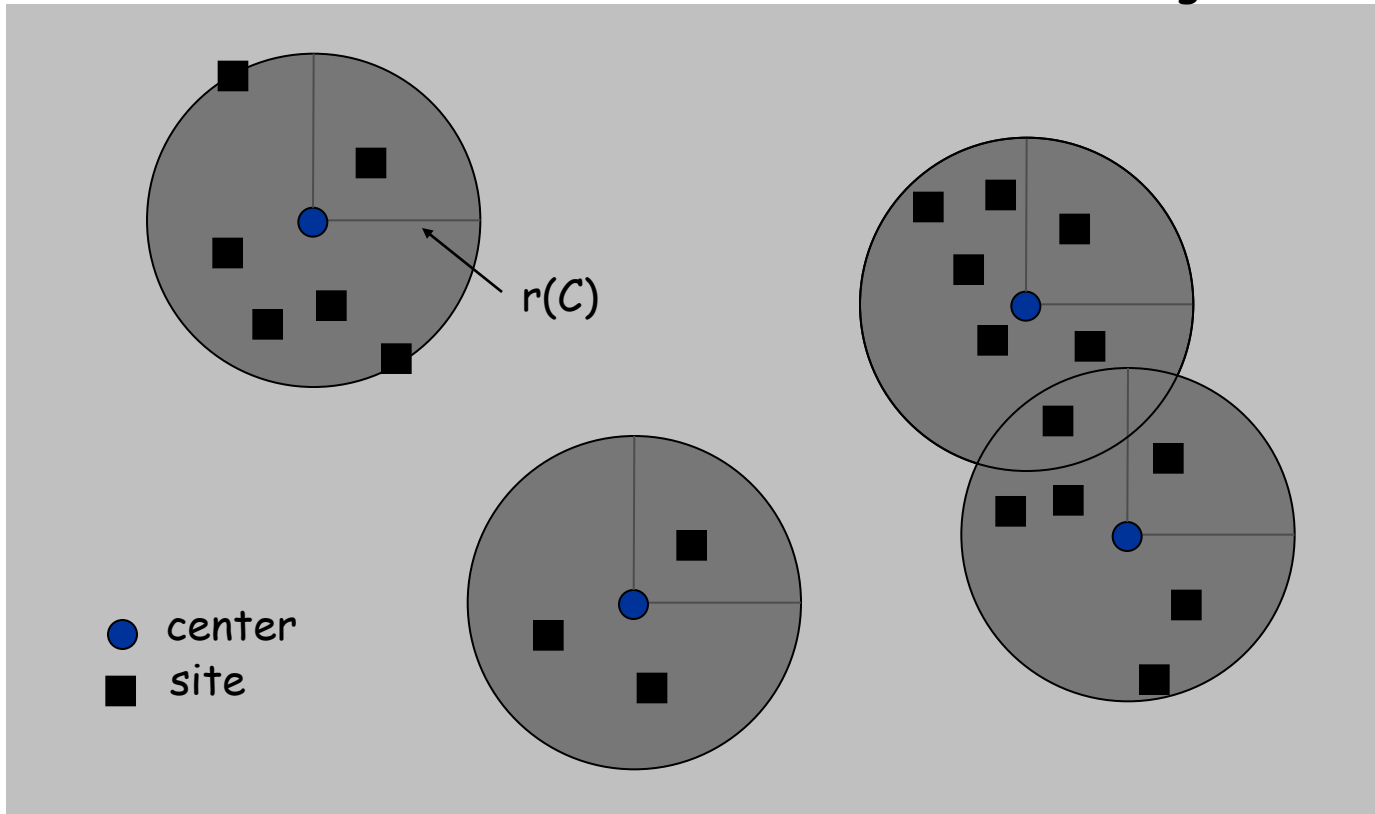
- $\text{dist}(x, y)$ = Abstand von x und y .
- $\text{dist}(s_i, C) = \min_{c \in C} \text{dist}(s_i, c)$ ("Distanz des Standorts s_i zum nächsten Zentrum").
- $r(C) = \max_{i \in \{1, \dots, n\}} \text{dist}(s_i, C)$ ("maximale Distanz der Standorte zum jeweils nächsten Zentrum").
- $r(C)$ wird **Radius** von C genannt.

Ziel. Bestimme Menge C von k Zentren, so dass $r(C)$ minimal ist.

Center Selection: Ein Beispiel

Beispiel: U sei die Euklidischen Ebene. Jeder Standort s_i ist ein beliebiger Punkt in der Euklidischen Ebene U , $\text{dist}(x, y)$ = Euklidischer Abstand.

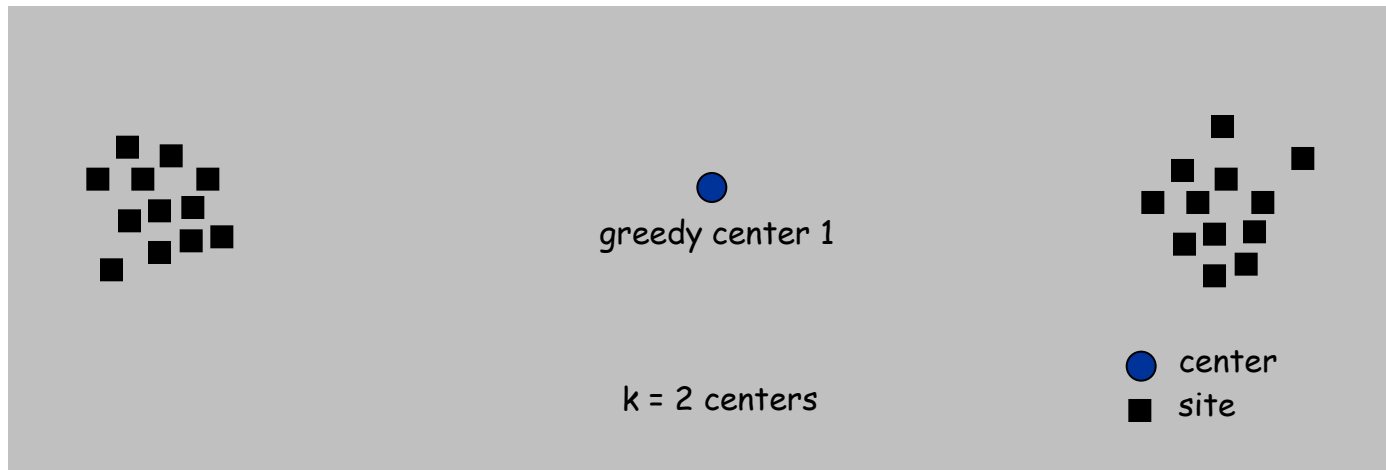
Die Suche könnte möglicherweise unendlich sein, da es in der Euklidischen Ebene unendlich viele Plätze für die Zentren gibt.



Ein falscher Start

Ein Greedy-Algorithmus (erster Versuch). Lege das erste Zentrum auf einen optimalen Platz für ein einziges Zentrum. Füge dann fortlaufend Zentren so hinzu, dass in jedem Schritt der Radius $r(C)$ möglichst stark abnimmt. (Zur Erinnerung: $r(C)$ soll minimiert werden.)

Bemerkung: Gelieferte Lösung kann beliebig schlecht werden!



Center Selection: Ein besserer Greedy-Algorithmus

Greedy Algorithmus: Man wählt **nur Standorte s_i als Zentren**. Erstes Zentrum: beliebig. Dann:

- Man wählt in jedem Schritt ein Zentrum s_i , das **möglichst weit** von den bereits gewählten Zentren entfernt ist.

```
Greedy-Center-Selection(k, n,  $s_1, s_2, \dots, s_n$ ) {  
     $C = \{s_j\}$  for some  $j \in \{1, \dots, n\}$   
    repeat k-1 times {  
        Select a site  $s_i$  with maximum  $\text{dist}(s_i, C)$   
        Add  $s_i$  to C  
    }  
    return C  
}
```

↑
am weitesten von jedem bisherigen
Zentrum entfernter Standort

Center Selection: Analyse des Greedy-Algorithmus

Es gilt: (*) Bei Verwendung dieses Greedy-Algorithmus sind am Ende alle Zentren in C paarweise mindestens $r(C)$ voneinander entfernt.

Beweis von (*) (durch Widerspruch). Die Zentren seien in der Reihenfolge c_1, \dots, c_k in C aufgenommen worden. Angenommen, es gibt $c_i, c_j \in C$ mit $i < j$, so dass $\text{dist}(c_i, c_j) < r(C)$.

Nach Definition von $r(C)$ existiert ein s_+ mit $\text{dist}(s_+, C) = r(C)$. Es folgt $0 \leq \text{dist}(c_i, c_j) < r(C) = \text{dist}(s_+, C)$. Somit gilt $s_+ \notin C$. Dies widerspricht aber der Wahl von c_j durch den Algorithmus:

Man betrachte den Schritt, in dem c_j in die Menge C aufgenommen wurde; C' sei die Menge der Zentren direkt vor Aufnahme von c_j . Es gilt $\text{dist}(c_i, c_j) < \text{dist}(s_+, C) \leq \text{dist}(s_+, C')$. Also: Im betrachteten Schritt hätte s_+ anstelle von c_j in C aufgenommen werden müssen - Widerspruch! ▀

Center Selection: Analyse des Greedy-Algorithmus

Theorem. C sei die Menge, die der Greedy-Algorithmus liefert; C^* sei eine optimale Menge von Zentren. Dann gilt $r(C) \leq 2r(C^*)$.

Beweis (durch Widerspruch). Angenommen, es gelte $r(C) > 2r(C^*)$. Es folgt $r(C^*) < \frac{1}{2} r(C)$.

Für jedes c_i aus C betrachten wir die Menge $K_i = \{u \in U: \text{dist}(u, c_i) \leq r(C^*)\}$. Wir nennen diese Menge **Kugel vom Radius $r(C^*)$ um c_i** .

Wegen Beobachtung (*) und wegen $r(C^*) < \frac{1}{2} r(C)$ sind die Kugeln K_i paarweise disjunkt.

Nach Definition von $r(C^*)$ gibt es in jeder dieser Kugeln ein Element aus C^* , das wir c_i^* nennen wollen.

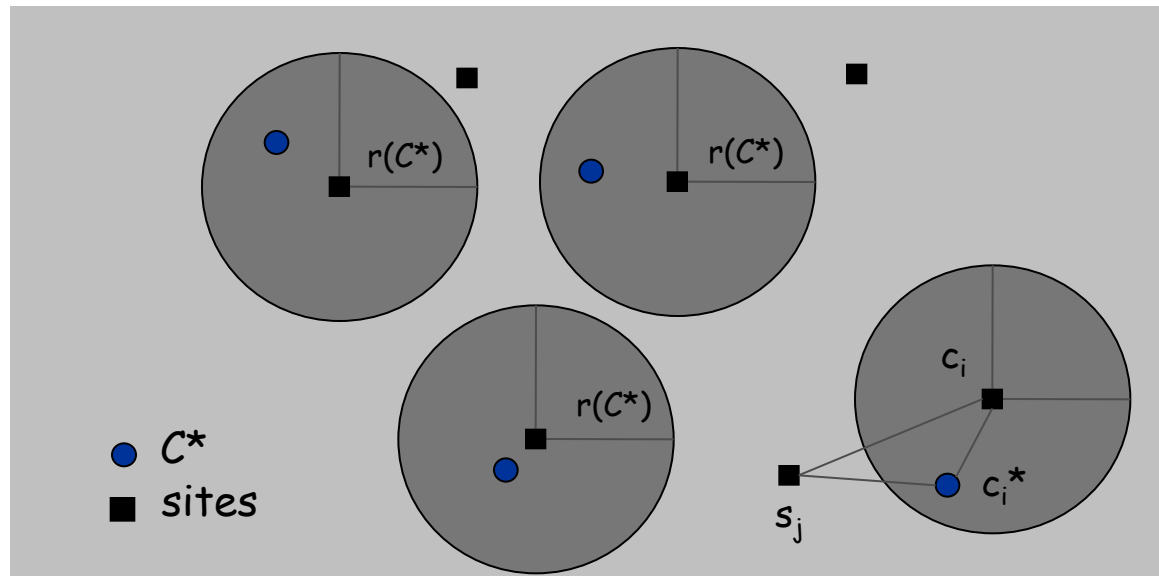
Center Selection: Analyse des Greedy-Algorithmus

Wir betrachten nun einen beliebigen Standort s_j und ein dazugehöriges möglichst nahes c_i^* aus C^* . Es folgt

$$\text{dist}(s_j, C) \leq \text{dist}(s_j, c_i) \leq \text{dist}(s_j, c_i^*) + \text{dist}(c_i^*, c_i) \leq 2r(C^*)$$

Δ -Ungleichung beide Summanden $\leq r(C^*)$

Also gilt $r(C) \leq 2r(C^*)$, im Widerspruch zur obigen Annahme. ▀



Center Selection: Analyse des Greedy-Algorithmus

Man beachte: Der Greedy-Algorithmus platziert sämtliche Zentren auf Standorten. In der optimalen Menge C^* des Beweises, können dagegen auch Zentren vorkommen, die aus der Grundmenge U stammen aber keine Standorte sind. Der Gütegarantiefaktor 2 gilt also auch für das allgemeine Problem, bei dem Zentren nicht unbedingt Standorte sind.

Unser Greedy-Algorithmus liefert also eine Näherungslösung für das Center Selection Problem mit Gütegarantiefaktor 2.

Im Folgenden setzen wir voraus, dass U endlich ist und dass es sich bei den Abständen $\text{dist}(x, y)$ um ganze Zahlen handelt. Das dazugehörige Optimierungsproblem nennen wir CENTER SELECTION.

Verbesserung der Gütegarantie: möglich oder nicht?

Man stellt fest: Unser Greedy-Algorithmus besitzt polynomielle Laufzeit und ist somit ein **2-Approximationsalgorithmus** für CENTER SELECTION.

Frage. Ist ein $3/2$ -Approximationsalgorithmus für CENTER SELECTION möglich? Oder sogar ein $4/3$ -Approximationsalgorithmus?

Antwort: Vermutlich nicht.

Es gilt nämlich das folgende Ergebnis:

DOMINATING SET kommt ins Spiel

Theorem. Falls $P \neq NP$, so gilt für alle p mit $1 \leq p < 2$: Es gibt keinen p -Approximationsalgorithmus für CENTER SELECTION.

Beweis. Wir zeigen, wie man einen p -Approximationsalgorithmus für $1 \leq p < 2$ benutzen könnte, um das NP-vollständige Problem DOMINATING SET in polynomieller Zeit zu lösen.

Zur Erinnerung: Unter einer **dominierenden Menge** von $G = (V, E)$ versteht man eine Menge $D \subseteq V$, für die gilt: Jeder Knoten aus $V \setminus D$ ist zu mindestens einem Knoten aus D benachbart ("in einem Schritt von D aus erreichbar").

DOMINATING SET

Eingabe: Ein Graph $G = (V, E)$ und eine ganze Zahl k mit $0 < k < n = |V|$.

Frage: Enthält G eine dominierende Menge $D \subseteq V$ mit $|D| = k$?

Eine Reduktion

Wir nehmen an, dass für $1 \leq \rho < 2$ ein ρ -Approximationsalgorithmus für das CENTER SELECTION existiert.

- Es sei $G = (V, E)$ zusammen mit k eine Eingabe für DOMINATING SET.
- Wir definieren eine dazugehörige Eingabe für das Center Selection Problem mit $V =$ Menge aller Standorte = Grundmenge, d.h. Menge der möglichen Zentren. Für $u \neq v$ gelte:
 - $\text{dist}(u, v) = 1$ für $(u, v) \in E$
 - $\text{dist}(u, v) = 2$ für $(u, v) \notin E$

Außerdem gelte natürlich $\text{dist}(u, u) = 0$.

Man beachte, dass die Dreiecksungleichung erfüllt ist.

Eine Reduktion

Bei dieser Konstruktion handelt es sich im Wesentlichen um eine Umformulierung des Problems DOMINATING SET in die Sprache von CENTER SELECTION. Man erkennt, dass die folgenden beiden Aussagen äquivalent sind:

- D ist eine dominierende Menge von G mit $|D| = k$.
- D ist eine Menge von k Zentren, für die $r(D) = 1$ gilt.

Man beachte: Aus der Feststellung $r(D) = \rho$ für $1 \leq \rho < 2$ folgt $\rho = 1$, da es für $1 < \rho < 2$ keine Elemente von V mit Abstand ρ gibt.

Äquivalent zu den beiden vorangegangenen Aussagen ist also auch die folgende Aussage:

- D ist eine Menge von k Zentren, für die $r(D) = \rho$ für $1 \leq \rho < 2$ gilt.

Eine Reduktion

Gegeben sei nun für $1 \leq p < 2$ ein p -Approximationsalgorithmus für CENTER SELECTION.

Nach dem zuvor Gesagten gilt:

Genau dann, wenn der Algorithmus eine Menge D von k Zentren liefert, für die $r(D) = \rho$ für $1 \leq \rho < 2$ gilt, existiert eine dominierende Menge von G mit $|D| = k$.

Es gilt also wie behauptet:

Gibt es für $1 \leq p < 2$ einen p -Approximationsalgorithmus für CENTER SELECTION, so kann man diesen Algorithmus benutzen, um das Problem DOMINATING SET in polynomieller Zeit zu lösen. ▀